UNIVERSIDAD
DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Redes neuronales para grafos en variedades pseudo-riemannianas

Presentado por:
Germán José Padua Pleguezuelo

Tutores:
Juan Gómez Romero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Francisco Torralbo Torralbo
Departamento de Geometría y Topología

Curso académico 2023-2024

Redes neuronales para grafos en variedades pseudo-riemannianas

Germán José Padua Pleguezuelo

Germán José Padua Pleguezuelo *Redes neuronales para grafos en variedades pseudo-riemannianas*. Trabajo de fin de Grado. Curso académico 2023-2024.

Responsable de tutorización

Juan Gómez Romero
Departamento de Ciencias de la Computación e Inteligencia Artificial

Francisco Torralbo Torralbo
Departamento de Geometría y Topología

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Universidad de Granada

Declaración de originalidad

D. Germán José Padua Pleguezuelo

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2023-2024, es original, entendido esto en el sentido de que no he utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 15 de julio de 2024

Fdo: Germán José Padua Pleguezuelo

Redes neuronales para grafos en variedades pseudo-riemannianas

Germán José Padua Pleguezuelo

Palabras clave:
redes pseudo-hiperbólicas neuronales convolucionales para grafos ($\mathcal{Q}$GCN), variedades pseudo-riemannianas, aplicación exponencial, geodésicas, aprendizaje automático, grafos, métodos de optimización.

Resumen

Este trabajo se centra en la familiarización con las redes neuronales para grafos, prestando especial atención a las redes neuronales de convolución para grafos en variedades pseudo-riemannianas.

El aprendizaje de datos estructurados en forma de grafos es una de las áreas más importantes del aprendizaje automático. Las redes neuronales convolucionales para grafos se han consolidado como las técnicas estándar en este campo. Sin embargo, tienen una limitación: el espacio de características considerado es normalmente el espacio euclídeo. Este espacio no es el más adecuado para representar estructuras más complejas, como árboles o ciclos en los grafos, que podrían representarse de manera más apropiada en variedades hiperbólicas o esféricas.

En este contexto, surgen como solución las redes pseudo-hiperbólicas neuronales convolucionales ($\mathcal{Q}$GCN). Estas redes trabajan con los pseudo-hiperboloides, una familia de subvariedades pseudo-riemannianas. Las variedades pseudo-riemannianas son variedades diferenciables equipadas con una métrica pseudo-riemanniana. No obstante, estas redes se enfrentan a un nuevo desafío: los pseudo-hiperboloides no son geodésicamente conexos, lo que significa que existen pares de puntos que no pueden ser conectados por una geodésica. Esto se solucionará mediante el uso de la aplicación exponencial, que permite proyectar vectores del espacio tangente de un punto de la variedad a un punto de la variedad.

El objetivo principal de este trabajo es comprender los conceptos tras las redes pseudo-hiperbólicas neuronales convolucionales para grafos. Para conseguirlo, estudiaremos las redes neuronales, las redes neuronales convolucionales y sus extensiones para grafos. Además, veremos los fundamentos matemáticos de las variedades pseudo-riemannianas, en particular de los pseudo-hiperboloides. De esta familia estudiaremos sus propiedades, calcularemos sus geodésicas y aplicación exponencial. Adicionalmente, investigaremos los conceptos de completitud geodésica y conectividad geodésica, destacando su equivalencia únicamente en variedades riemannianas. También, se explorarán dos métodos de optimización útiles para la geometría pseudo-riemanniana.

Finalmente, realizaremos experimentos para probar las distintas aplicaciones de estos modelos y el potencial de $\mathcal{Q}$GCN para representar datos del mundo real. Los resultados son esperanzadores, en la mayoría de los datos y tareas, se obtiene el mejor resultado con el modelo $\mathcal{Q}$GCN. Sin embargo, todavía existen diversas líneas de investigación posibles, como la aplicación de estos modelos a otras tareas computacionales o la mejora de los métodos de optimización.

En resumen, este trabajo muestra la relación entre la Ingeniería Informática y las Matemáticas y destaca cómo ambos campos pueden beneficiarse mutuamente.

Pseudo-riemannian graph neural networks

Germán José Padua Pleguezuelo

Keywords:
pseudo-hyperbolic graph convolutional networks, pseudo-riemannian manifolds, exponential map, geodesics, deep learning, graph-structured data, optimization methods.

Abstract

This undergraduate thesis focuses on the study and understanding of graph neural networks, with a special emphasis on pseudo-riemannian graph convolutional networks.

Learning from graph-structured data is one of the main areas of investigation in deep learning. Graph convolutional neural networks (GCNs) have established themselves as the main framework in this area. However, they have a significant limitation: the feature space is usually the Euclidean space. This space fails to represent more complex structures such as tree-like graphs or graphs with cycles, which are more suitably represented by hyperbolic and spherical manifolds, respectively.

In this context, pseudo-hyperbolic graph convolutional networks ($\mathcal{Q}$GCN) emerge as a solution. These networks work with pseudo-hyperboloids, a family of pseudo-riemannian submanifolds. Pseudo-riemannian manifolds are differentiable manifolds equipped with a pseudo-riemannian metric. Nevertheless, this innovative network rises a new challenge: pseudo-hyperboloids are geodesically disconnected, meaning there exist pairs of points in the manifold which cannot be connected by a geodesic. The solution to this is the use of the expontial map, which projects a vector in the tangent space of a point on the manifold to a point on the manifold.

The main objective of this work is to understand the concepts behind pseudo-hyperbolic graph convolutional networks. To achieve this, we will embark on a comprehensive study of neural networks, including both the foundational principles and advanced methodologies. This will include an in-depth exploration of convolutional neural networks (CNNs) and graph neural networks (GNNs). We will explain their operations, detail the learning process they employ and discuss the most common arquitectures used in practice.

Next, we will present the mathematical fundamentals of pseudo-riemannian manifolds, specifically pseudo-hyperboloids. We will look into their properties, their geodesics and exponential maps. In addition, we will investigate the concepts of geodesic completeness and geodesic connectedness, noting that they are equivalent only in riemannian manifolds. This distinction is crucial for understanding the unique challenges posed by pseudo-riemannian manifolds. Moreover, we will examine a couple of optimization methods in pseudo-riemannian geometry, discussing their theory as well as their implementations.

Finally, we will conduct experiments to demonstrate the different applications of these models and the potential of $\mathcal{Q}$GCN to represent real-world data. The results of the experiments are encouraging, in most datasets and tasks, the best result is achieved with the $\mathcal{Q}$GCN model. However, there is still substantial future work, such as applying these models to other

computational tasks and improving optimization methods, among others.

This project illustrates the close relationship between Computer Science and Mathematics and highlights how these fields can mutually benefit from each other.

Índice general

Introducción

Este capítulo tiene como objetivo contextualizar, describir y estructurar el trabajo realizado, además de indicar los objetivos perseguidos y las principales fuentes consultadas.

En primer lugar, se presentará la contextualización, donde se explicarán los antecedentes importantes para el desarrollo de este trabajo. Seguidamente, se describirá el trabajo realizado y se detallará la estructura del documento. También se citarán las principales fuentes consultadas y se enumerarán los objetivos establecidos. Para finalizar, se realizará una planificación y una estimación de costes del proyecto.

1. Contextualización

El aprendizaje de datos estructurados en forma de grafos es una de las áreas más importantes en el ámbito del aprendizaje automático. Las redes neuronales convolucionales para grafos (GCNs) se han establecido como técnicas potentes en este campo, ya que aprovechan tanto la estructura del grafo como las características de sus nodos. Sin embargo, generalmente los grafos se estudian en el espacio euclídeo, lo que limita la capacidad de representación de estas redes, especialmente en grafos con estructuras más complejas, como árboles o ciclos, que podrían representarse de manera más adecuada en variedades hiperbólicas o esféricas, respectivamente.

Las topologías de los grafos de la vida real normalmente exhiben una estructura topológica muy heterogénea, la cual está mejor representada con otras estructuras geométricas distintas al espacio euclídeo. En este contexto, surgen las redes neuronales convolucionales para grafos en variedades pseudo-riemannianas. Estas variedades, equipadas con una métrica indefinida, generalizan otras variedades como la hiperbólica y la esférica.

No obstante, trabajar con este tipo de variedades presenta un desafío significativo: la ausencia de herramientas geodésicas adecuadas para extender las operaciones de las redes neuronales a la geometría pseudo-riemanniana. El problema radica principalmente en la no conectividad geodésica, es decir, la existencia de puntos que no pueden ser conectados por una geodésica.

La base de la solución que aparece, es la aplicación exponencial, que permite proyectar un vector del espacio tangente en un punto a un punto de la variedad. A partir de esta aplicación y otras herramientas, se pueden definir métodos de optimización para variedades pseudo-riemannianas, permitiendo así explotar las propiedades intrínsecas de los grafos de la vida real.

2. Descripción del trabajo

Nuestro trabajo se focalizará en la familiarización con las redes neuronales para grafos, con especial atención a las redes de convolución para grafos en variedades pseudo-riemannianas.

En la primera parte del presente trabajo se estudiarán los fundamentos informáticos de las redes neuronales para grafos. Se recordarán en primer lugar conceptos básicos sobre redes neuronales y redes neuronales convolucionales. Más tarde, se introducirán las redes neuronales para grafos, explicando sus operaciones específicas, el proceso de aprendizaje y las arquitecturas más extendidas. Para terminar este bloque, se explicará la arquitectura de la red pseudo-hiperbólica convolucional para grafos ($\mathcal{Q}$GCN), una red neuronal para grafos en variedades pseudo-riemannianas.

En la siguiente parte, se abordarán los fundamentos matemáticos de las variedades pseudo-riemannianas. Este estudio incluirá las geodésicas, la aplicación exponencial y la diferencia entre completitud y conectividad geodésica. Se aplicarán estos conceptos al estudio de los pseudo-hiperboloides, una familia de subvariedades pseudo-riemannianas que juega un papel fundamental en la técnica de aprendizaje de las redes neuronales convoluciones para variedades pseudo-riemannianas.

En la tercera parte, se realizarán experimentos con el código de diferentes redes neuronales para grafos, incluyendo $\mathcal{Q}$GCN, proporcionando ejemplos que ilustren la capacidad de estos modelos.

3. Estructura del trabajo y bibliografía fundamental

En esta sección comentaremos la estructura que sigue este trabajo. Disponemos de tres partes diferenciadas, cada una de ellas estructurada en capítulos.

1. El primer bloque se estructura en 3 capítulos. El capítulo 1 nos recordará los fundamentos de redes neuronales y redes neuronales convolucionales. En el capítulo 2 se expondrán las definiciones y propiedades de los grafos. Esta parte finalizará con el capítulo 3, que se centra en introducir las redes neuronales para grafos. Se comentará brevemente su origen y las distintas operaciones específicas para este tipo de red. Además, se estudiarán las arquitecturas más comunes, junto con las redes pseudo-hiperbólicas convolucionales para grafos. Las principales referencias consultadas para esta parte han sido [MT21, SAV20], para el capítulo 3 también se han revisado [LS21, XZP+22a].
2. El segundo bloque consta de 6 capítulos. El capítulo 4 recordará los conceptos necesarios para comprender las variedades pseudo-riemannianas, estudiadas en el capítulo siguiente. En el capítulo 5 se estudiarán sus propiedades, curvatura y geodésicas. En el capítulo 6 se realizará un estudio completo de los pseudo-hiperboloides. Los capítulos 7 y 8 se centrarán en discutir los conceptos de distancia en variedades, y de completitud y conectividad geodésica, respectivamente. En el último capítulo de esta parte discutiremos dos métodos de optimización para los pseudo-hiperboloides. En el desarrollo de esta parte se ha recurrido a las siguientes referencias clave: [O'N83, KS24a, VC10].
3. El tercer bloque se divide en dos capítulos. En el primero de ellos se muestran todos los detalles de los experimentos realizados, así como las herramientas utilizadas y

el código implementado. En el capítulo 11 se verán los resultados obtenidos en la experimentación. Finalmente, se expondrán las conclusiones obtenidas de este trabajo y se destacarán algunas líneas de trabajo futuro.

4. Objetivos del trabajo

En este capítulo se enumerarán y describirán los objetivos principales de este trabajo, especificando en qué parte del documento se encuentran detallados.

1. Informática: El objetivo principal es describir los fundamentos de las redes neuronales para grafos, con el particular estudio de la red pseudo-hiperbólica convolucional para grafos, y desarrollar experimentos que ilustren la capacidad de estas técnicas para resolver problemas de aprendizaje con grafos.
   a) Recordar todos los conceptos fundamentales sobre las redes neuronales y las CNN. Capítulo 1.
   b) Estudiar los fundamentos informáticos de las redes neuronales para grafos, donde se incluye las operaciones de agregación de información, un estudio del aprendizaje de estos modelos para distintas tareas y las arquitecturas más comunes. Capítulo 3.
   c) Estudiar la red pseudo-hiperbólica neuronal convolucional para grafos, las capas que la componen, su proceso completo y el optimizador RiemannianAdam. Sección 3.6.
   d) Llevar a cabo experimentos ilustrativos que muestren la aplicación de estas redes neuronales en problemas de clasificación de nodos y predicción de enlaces. Parte III.
2. Matemáticas: El objetivo principal es estudiar los conceptos esenciales sobre variedades pseudo-riemannianas, para describir de forma rigurosa los fundamentos matemáticos de las redes neuronales que operan sobre grafos en este tipo de variedades.
   a) Recordar los conceptos necesarios para poder definir las variedades pseudo-riemannianas. Capítulo 4.
   b) Estudiar en profundidad las variedades pseudo-riemannianas, dando las definiciones de tensor de curvatura, geodésicas y aplicación exponencial, entre otros. Capítulo 5.
   c) Aplicar lo anterior al estudio detallado de los pseudo-hiperboloides, familia de subvariedades del espacio $\mathbb{R}^d_t$ con una métrica pseudo-riemanniana. Capítulo 6.
   d) Estudiar los métodos de optimización en geometría pseudo-riemanniana. Capítulo 9.

5. Planificación y estimación de costes

5.1. Planificación

Se enumeran las cuatro etapas que conforman la planificación:

1. **Estudio inicial del problema:** Esta tarea consiste en revisar toda la documentación y bibliografía necesaria, con el objetivo de alcanzar una primera toma de contacto sobre las dos ramas principales de este proyecto: por un lado, las redes neuronales para grafos, leyendo referencias clave como [GBC16] y [MT21]; y por otro lado las variedades pseudo-riemannianas, revisando los apuntes [KS24a].

2. **Investigación:** Tras una toma de contacto de los temas, en esta etapa nos encargamos de investigar a fondo la bibliografía. Primero, se desarrollarán los preliminares matemáticos para entender la teoría de las variedades, en particular, las variedades pseudo-riemannianas. Más tarde, se procederá con la parte informática, introduciendo los fundamentos de aprendizaje profundo y de grafos. Posteriormente, se estudiarán las particularidades de los pseudo-hiperboloides y las propiedades de las geodésicas y aplicación exponencial. Esta etapa finalizará con una investigación de las redes neuronales para grafos y de la arquitectura $\mathcal{Q}$GCN.

3. **Experimentación:** Se pondrán en práctica los modelos estudiados y se compararán los resultados, prestando especial detalle al modelo $\mathcal{Q}$GCN.

4. **Revisiones finales:** En esta última etapa se realizarán los últimos cambios y retoques sobre todo el trabajo.

Debido al alto volumen de trabajo durante el curso académico y a que las bases del trabajo eran nuevas para mí, el desarrollo de este comenzó en el verano de 2023. Mostramos en la siguiente imagen la planificación inicial de las distintas etapas del proyecto:

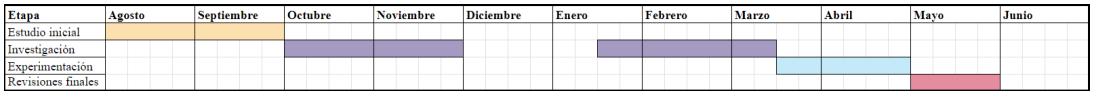


Figura 1.: Planificación inicial de las etapas del trabajo.

No obstante, se tuvo que invertir un mayor tiempo en la fase de investigación, como consecuencia, la planificación inicial sufrió unos leves cambios. Además, se comenzó la etapa de experimentación a finales de enero, con motivo de obtener retroalimentación de los tutores y revisar la etapa de investigación:

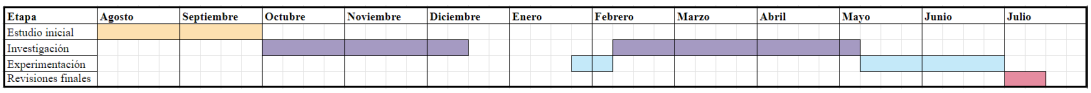


Figura 2.: Planificación final de las etapas del trabajo.

5.2. Estimación de costes

Se estiman que se han invertido aproximadamente 10 horas semanales desde el comienzo del desarrollo de este trabajo, siendo un total de 400 horas. Realizamos la estimación de coste teniendo en cuenta el sueldo promedio para un científico de datos sin experiencia en España, que es 27.250€ al año, es decir, 13,10 €/hora. Por lo que teniendo en cuenta las horas invertidas, el coste sería de 5.240€. Además, debemos tener en cuenta los costes de los recursos y herramientas utilizadas, como consumo eléctrico, conexión a internet y

la suscripción a Google Colab Pro. El desglose total del presupuesto de este proyecto se encuentra en la Tabla 1.

Recurso	**Coste (€)**
Personal	5.240
Google Colab Pro	25
Consumo eléctrico	100
Conexión a internet	200
Total	5.565

Tabla 1.: Desglose del presupuesto del trabajo.

Parte I.

Fundamentos informáticos

1. Fundamentos de aprendizaje profundo

Este capítulo describirá los conceptos fundamentales del aprendizaje profundo (*deep learning*) que nos serán de gran utilidad para la comprensión de conceptos más avanzados como las redes neuronales para grafos. Las definiciones y resultados expuestos en este capítulo se han visto en las asignaturas cursadas a lo largo de mi formación y además se basan en referencias clave como [MT21, GBC16, SAV20].

1.1. Redes neuronales

En esta sección, comenzaremos con una breve introducción a las redes neuronales, explorando su arquitectura básica, algunas de las funciones de activación más conocidas que permiten introducir relaciones no lineales en la red, y comentaremos los diferentes tipos de salida y funciones de pérdida para la evaluación y ajuste de los modelos.

Las redes neuronales de propagación hacia delante, también llamadas redes neuronales *feedforward*, o perceptrones multicapa (MLPs), son los modelos de aprendizaje profundo por excelencia. Su objetivo es aproximar una función f^* definiendo una función $f(\mathbf{x};\theta)$ a la que se aplica un proceso de optimización. Este proceso le permite aprender el valor de los parámetros θ que consiguen la mejor aproximación de la función f^*.

En las redes feedforward, la información $\mathbf{x}$ fluye desde la entrada, pasando por cálculos intermedios, hasta llegar finalmente a la salida $\mathbf{y}$. Se consideran redes porque f se obtiene como la composición de varias funciones. Por ejemplo, la red de la Figura 1.1 tiene cuatro funciones $f^{(1)}, f^{(2)}, f^{(3)}, f^{(4)}$ conectadas en cadena y $f(x) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))))$. En este caso, $f^{(1)}$ es la primera capa de la red, $f^{(2)}$ la segunda capa, $f^{(3)}$ es la tercera y la capa final $f^{(4)}$ es la capa de salida. La longitud total de la cadena determina la profundidad del modelo.

Se llaman neuronales porque están inspiradas en la neurociencia. La operación en un nodo se asemeja a lo que ocurre en una neurona en el cerebro, la cual recibe y transforma información proveniente de muchos otros nodos y luego la pasa a través de una función de activación, que determina hasta qué punto la información puede pasar al siguiente paso.

1.1.1. Arquitectura

En una red neuronal feedforward totalmente conectada, un nodo en una capa está conectado a todos los nodos de las capas anterior y siguiente. Ahora, introduciremos los detalles de la computación necesaria en una red. La neurona, la unidad básica de procesamiento, se encargará de realizar una combinación lineal de las entradas con los pesos sumándole un valor de sesgo y, a continuación, aplicará una transformación no lineal en este resultado.

En adelante, utilizaremos la siguiente notación para una neurona: $\mathbf{x}$ es el vector con los valores de entrada, $\mathbf{w}$ es el vector de pesos, $\mathbf{b}$ el valor del sesgo y $\alpha()$ es una función de

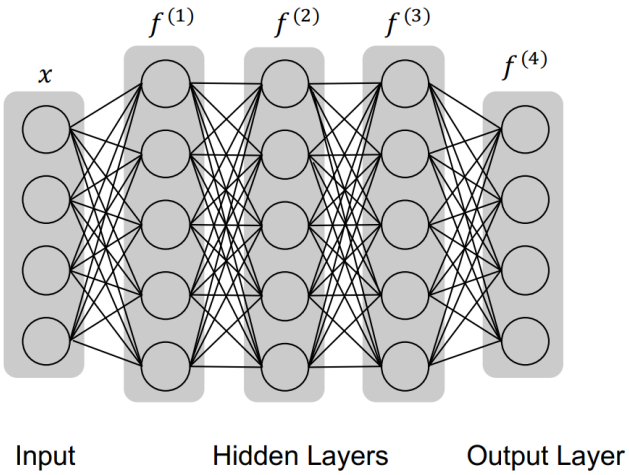


Figura 1.1.: Un ejemplo de redes de propagación hacia delant. Imagen obtenida de [MT21].

activación. Ilustramos estas conexiones entre los nodos de una capa y un nodo arbitrario de la capa siguiente en la Figura 1.2.

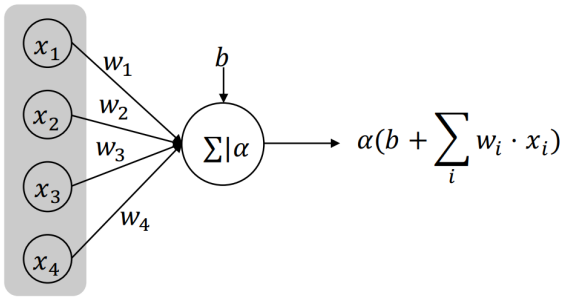


Figura 1.2.: Operaciones en un nodo. Imagen obtenida de [MT21].

Vamos a generalizar la operación a una capa oculta arbitraria. Asumiendo que en la capa k-ésima tenemos $N^{(k)}$ nodos y la salida la representamos con $\mathbf{h}^{(k)}$. Entonces, para calcular $\mathbf{h}^{(k+1)}$ en la capa $(k+1)$-ésima:

$$\mathbf{h}^{(k+1)} = \alpha(\mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k)}),$$

donde $\mathbf{W}^{(k)} \in \mathbb{R}^{N^{(k+1)} \times N^{(k)}}$ es una matriz conteniendo todos los pesos, $\mathbf{W}_{ji}^{(k)}$ denota el peso correspondendiente a la conexión entre $\mathbf{h}_i^{(k)}$ y $\mathbf{h}_j^{(k+1)}$, $\mathbf{b}^{(k)}$ consiste en todos los términos de sesgo, siendo $\mathbf{b}_j^{(k)}$ el sesgo para calcular $\mathbf{h}_j^{(k+1)}$. Usando que $f^{(k+1)}$ representa la operación de la capa $(k+1)$ en la red, tendremos:

$$f^{(k+1)}(\mathbf{h}^{(k)}) = \mathbf{h}^{(k+1)}.$$

1.1.2. Funciones de activación

Las funciones de activación introducen la no-linearidad en la red neuronal, lo que mejorará su capacidad de aproximación. Veamos las funciones más comunes.

1.1.2.1. Rectificador

Los rectificadores son las funciones de activación más utilizadas. Su funcionamiento consiste en no realizar ninguna operación para las entradas positivas o nulas y devolver cero cuando la entrada es negativa:

$$\mathrm{ReLU}(z) = \max\{0, z\}.$$

En cada capa, solo unas pocas unidades son activadas, lo que asegura eficiencia computacional. Por otro lado, una desventaja es que su gradiente es 0 en la mitad negativa del dominio. Existen algunas modificaciones de ReLU para superar este inconveniente.

LeakyReLU introduce una pendiente pequeña en los valores negativos, normalmente se asigna un valor entre 0.01 y 0.2. Matemáticamente se expresa:

$$\mathrm{LeakyReLU}_\gamma(z) = \begin{cases} \gamma \cdot z & \text{si } z < 0, \\ z & \text{si } z \geq 0, \end{cases}$$

ELU utiliza una transformación exponencial para los valores negativos:

$$\mathrm{ELU}(z) = \begin{cases} c \cdot \exp(z) - 1 & \text{si } z < 0 \\ z & \text{si } z \geq 0, \end{cases}$$

donde c es una constante positiva que controla la pendiente de la función exponencial.

1.1.2.2. Sigmoide logístico y tangente hiperbólico

La función de activación sigmoide se puede representar matemáticamente como sigue:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

La imagen de esta función está acotada entre 0 y 1. Cuanto más negativa es la entrada, más cerca de 0 se encuentra la salida y cuanto más positiva es, más cerca de 1.

Por otro lado, la función de activación de tangente hiperbólico está relacionada con la sigmoide:

$$\tanh(z) = \frac{2}{1 + \exp(-2z)} - 1 = 2 \cdot \sigma(2z) - 1.$$

Las salidas de esta función están entre -1 y 1.

1.1.2.3. Softmax

La función softmax es una función de activación que generaliza la sigmoide a múltiples clases. Normaliza la salida de la red tal que cada predicción corresponde a la probabilidad de que la entrada corresponda a esa clase.

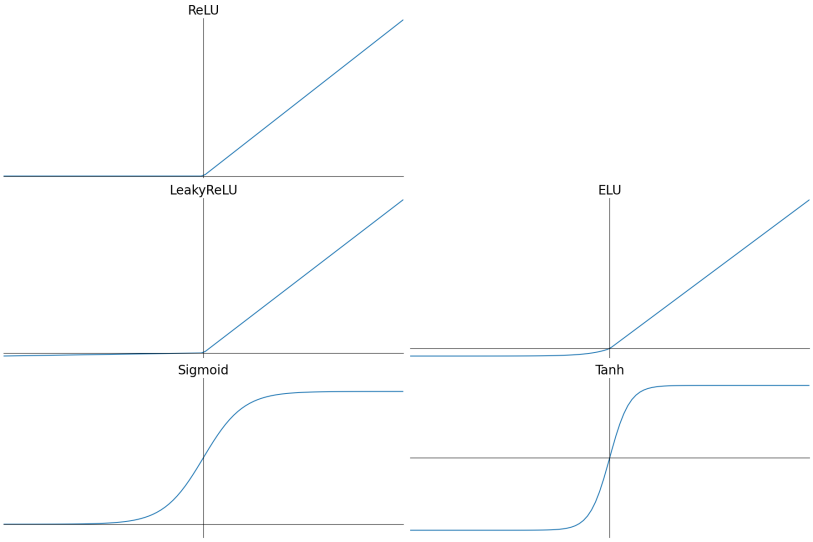


Figura 1.3.: Distintas funciones de activación

$$\text{softmax}(\mathbf{x}) = \frac{\exp(x_i)}{\sum_{i=1}^{n} \exp(x_i)} \quad \forall i \in \{1, \ldots, n\}$$

1.1.3. Salida y función de pérdida

La elección de la capa de salida y de la función de pérdida depende específicamente del tipo de problema a abordar. A continuación, describimos las configuraciones más comunes para diferentes tipos de problemas.

En problemas de regresión, se requieren salidas de valores continuos. Esto se logra comúnmente mediante una capa lineal sin activación no lineal. Dadas las características de la capa previa $\mathbf{h} \in \mathbb{R}^{d_{in}}$, una capa lineal sería:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h} + \mathbf{b}, \quad \hat{\mathbf{y}} \in \mathbb{R}^{d_{ou}},$$

donde $\mathbf{W} \in \mathbb{R}^{d_{ou} \times d_{in}}$ y $\mathbf{b} \in \mathbb{R}^{d_{ou}}$ son los parámetros a aprender.

Existen numerosas funciones de pérdida, la más extendida es el error cuadrático medio, que mide la diferencia entre el valor predicho $\hat{\mathbf{y}}$ y el actual $\mathbf{y}$ de la siguiente manera:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = (\mathbf{y} - \hat{\mathbf{y}})^2.$$

Para problemas de clasificación con n clases, consideramos que cada clase se corresponde con un número entero de 0 a $n-1$. Representamos la clase objetivo mediante un vector $\mathbf{y} \in \{0,1\}^n$, donde el valor 1 en la posición i indica la pertenencia de la muestra a la clase i. La transformación de las características de entrada $\mathbf{h}$ en un vector n-dimensional se realiza a través de una capa lineal:

$$\mathbf{z} = \mathbf{W}\mathbf{h} + \mathbf{b},$$

con $\mathbf{W} \in \mathbb{R}^{n \times d_{in}}$ y $\mathbf{b} \in \mathbb{R}^n$. Seguidamente, podemos aplicar la función softmax para normalizar $\mathbf{z}$ en una distribución de probabilidad discreta sobre las clases:

$$\hat{\mathbf{y}}_i = \text{softmax}(\mathbf{z})_i \ ,$$

donde $\hat{\mathbf{y}}_i$ indica la probabilidad de que la entrada sea predicha con la clase $i-1$. Una entrada se predecirá que pertenece a la clase $i-1$ si $\hat{\mathbf{y}}_i$ es el valor más grande de todas las salidas. Con la predicción $\hat{\mathbf{y}}$ podemos emplear la pérdida de entropía cruzada (*cross-entropy loss*) para medir la diferencia entre el valor real y la predicción:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=0}^{n-1} \mathbf{y}_i \log(\hat{\mathbf{y}}_i).$$

1.2. Redes neuronales convolucionales

Introducidas por Kunihiko Fukushima en 1980 [Fuk80] y aplicadas al reconocimiento de imágenes por primera vez por LeCun en 1989 [LBD+89], las redes neuronales convolucionales (CNNs) constituyen una categoría especializada de redes neuronales ideadas para el procesamiento de datos organizados en forma de cuadrícula, típicamente imágenes. La característica distintiva de las CNNs es la utilización de una operación matemática llamada convolución, en lugar de la simple multiplicación de matrices presente en las redes previamente presentadas. Esta operación se realiza en las llamadas capas convolucionales. Adicionalmente, las CNNs incorporan capas de pooling, las cuales tienen el propósito de reducir la dimensionalidad de los datos procesados, condensando la información de neuronas cercanas. En esta sección, introduciremos tanto la operación de convolución como el concepto y función de las capas de pooling.

1.2.1. Convolución

La convolución, representada por un asterisco, es una operación matemática que se define de la siguiente manera:

$$s(t) = (f * w)(t) = \int_{-\infty}^{\infty} f(\tau) w(t-\tau)\, d\tau.$$

El primer argumento f hace referencia a la entrada de la red, el segundo argumento w es el *kernel* o el filtro, y s es el resultado denominado mapa de características. Para secuencias discretas, donde f y w son funciones de valores enteros, la convolución se expresa como:

$$s(t) = (f * w)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau) w(t-\tau).$$

En el caso de redes neuronales, t se puede considerar como los índices de las unidades en la capa de entrada. Las capas convolucionales aplican el kernel w sobre la entrada f, desplazándolo a lo largo de ella para generar el mapa de características.

Veamos un ejemplo de como funcionaría una capa convolucional.

Ejemplo 1.1. La Figura 1.4 muestra un ejemplo de una capa convolucional, donde la entrada y la salida tienen igual tamaño. Para conseguir mantenerlo, esto se logra aplicando un relleno

(*padding*) a la entrada con dos unidades adicionales con valor 0, conocido como *zero-padding*, para compensar el desplazamiento del kernel.

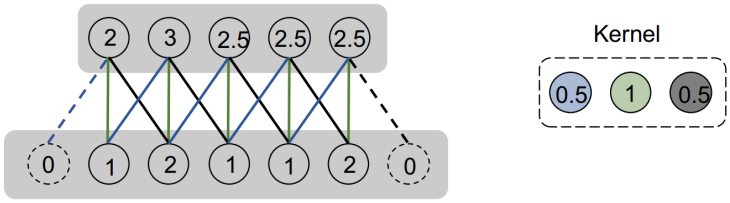


Figura 1.4.: Ejemplo de una capa convolucional. Imagen obtenida de [MT21].

Cuando tratamos con datos con más de una dimensión, como por ejemplo imágenes, la convolución se puede generalizar. Por ejemplo, para una imagen 2-dimensional I, la operación se puede llevar a cabo con un kernel K 2-dimensional:

$$S(i,j) = (I * K)(i,j) = \sum_{\tau=i-n}^{i+n} \sum_{j=\gamma-n}^{\gamma+n} I(\tau,\gamma)K(i-\tau, j-\gamma).$$

Las capas convolucionales exhiben tres propiedades clave:

- **Conexiones dispersas (*sparse connections*)**: Cada unidad de salida está conectada solo a una región pequeña de la entrada.
- **Compartición de parámetros (*parameter sharing*)**: El mismo kernel se aplica a toda la entrada, reduciendo la cantidad de parámetros.
- **Representación equivariable (*equivariant representation*)**: La convolución garantiza que si la entrada se desplaza, la salida también se desplazará de manera correspondiente.

1.2.2. Pooling

Tras una capa convolucional y la subsiguiente activación no lineal, a menudo se incorpora una capa de agregación, o como se denomina habitualmente, *pooling*. Su función principal es reducir la dimensionalidad de los mapas de características, lo cual disminuye la cantidad de parámetros y cálculos necesarios en la red. La operación de pooling resume la información mediante una función de agregación específica.

Existen varios tipos de pooling, pero los más comunes son el pooling máximo (*max pooling*) y el pooling promedio (*average pooling*):

- **Max Pooling**: Selecciona el valor máximo de los elementos dentro del área definida por el tamaño de la ventana de pooling. Esta técnica es efectiva para preservar las características más destacadas dentro de la ventana de pooling.
- **Average Pooling**: Calcula el promedio de los elementos dentro del área de la ventana de pooling. Esta forma de pooling tiende a suavizar las características, siendo útil en ciertos contextos donde se desea evitar la dominancia de valores extremos.

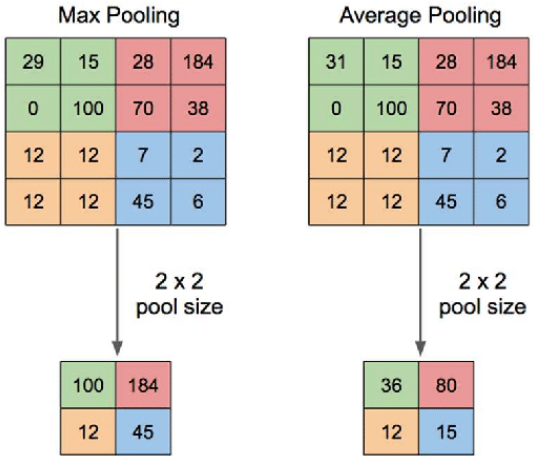


Figura 1.5.: Ejemplo de Max Pooling y Average Pooling. Imagen obtenida de [YIS19].

Además de max y average pooling, existen variantes como el pooling global, donde se realiza el pooling sobre toda la extensión de cada mapa de características, produciendo un único valor por mapa. Esto es particularmente útil para reducir drásticamente las dimensiones antes de una capa completamente conectada, permitiendo que la red se ajuste a tamaños de entrada variables.

1.3. Entrenamiento de redes neuronales profundas

En esta sección, describiremos el proceso de entrenamiento de las redes neuronales profundas, enfocándonos en el descenso del gradiente y en el algoritmo de retropropagación (*backpropagation*), el cual es fundamental para calcular los gradientes de los parámetros de las redes.

1.3.1. Descenso del gradiente

El entrenamiento de modelos de aprendizaje profundo implica minimizar una función de pérdida $\mathcal{L}$ con respecto a los parámetros del modelo. La función de pérdida, denotada como $\mathcal{L}(\mathbf{W})$, donde $\mathbf{W}$ representa el conjunto de todos los parámetros a optimizar, evalúa qué tan bien el modelo está realizando la tarea deseada.

El descenso del gradiente es un algoritmo de optimización iterativo de primer orden que busca minimizar $\mathcal{L}$. En cada iteración, actualiza los parámetros $\mathbf{W}$ moviéndolos en la dirección opuesta al gradiente de la función de pérdida con respecto a $\mathbf{W}$:

$$\mathbf{W}' = \mathbf{W} - \eta \cdot \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}),$$

donde $\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$ es el gradiente de la función de pérdida con respecto a los parámetros, y η es la tasa de aprendizaje, un escalar positivo que determina el tamaño del paso hacia la dirección de descenso.

La función de pérdida generalmente es una suma de penalizaciones sobre un conjunto de muestras de entrenamiento:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^{N_s} \mathcal{L}_i(\mathbf{W}),$$

con $\mathcal{L}_i(\mathbf{W})$ la pérdida de la muestra i-ésima y N_s el número de muestras.

Una variante popular para el cálculo del gradiente es el descenso del gradiente por mini lotes, que utiliza un subconjunto aleatorio de muestras para estimar el gradiente.

1.3.2. Backpropagation

El algoritmo de retropropagación es esencial para el entrenamiento de redes neuronales, permitiendo calcular eficientemente los gradientes de la función de pérdida con respecto a los parámetros de la red mediante el uso de programación dinámica. Este proceso se divide en dos fases principales:

1. **Fase de propagación hacia adelante (*Forward Phase*)**: Durante esta etapa, los datos de entrada se procesan secuencialmente a través de las capas de la red. Esta fase termina con la producción de la salida final de la red, que se utiliza para calcular la función de pérdida.

2. **Fase de retropropagación (*Backward Phase*)**: El objetivo de esta fase es calcular los gradientes de la función de pérdida con respecto a cada uno de los parámetros, lo cual se logra aplicando la regla de la cadena. Este proceso se inicia en la capa de salida y procede en dirección inversa a través de las capas, calculando el gradiente de la función de pérdida con respecto a los parámetros de cada capa.

Vamos a detallar el paso hacia detrás. Durante la fase de retropropagación, para cada capa l desde la salida hasta la entrada, calculamos el gradiente de la función de pérdida $\mathcal{L}$ con respecto a los pesos $\mathbf{W}^{(l)}$ y los sesgos $\mathbf{b}^{(l)}$ de dicha capa. Utilizando la regla de la cadena, determinamos cómo afectan pequeños cambios en estos parámetros al error total, permitiendo así su ajuste óptimo en pasos subsecuentes de entrenamiento mediante algoritmos de optimización, como el descenso del gradiente.

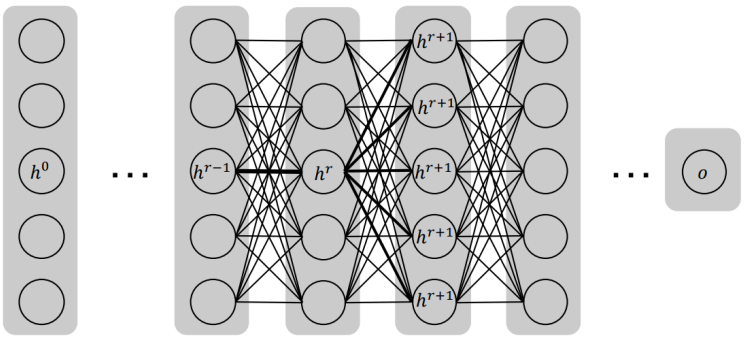


Figura 1.6.: Descomposición de caminos. Imagen obtenida de [MT21].

En la Figura 1.6, se ilustra una secuencia de neuronas conectadas $h^0, h^1, \ldots, o$ de distintas capas donde h^i denota una neurona de la capa i-ésima, con h^0 la capa de entrada y o la salida. Calculamos la derivada de la función de pérdida con respecto a un peso específico $w_{(h^{r-1},h^r)}$ utilizando la regla de la cadena como sigue:

$$\frac{\partial \mathcal{L}}{\partial w_{(h^{r-1},h^r)}} = \frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{\left[h^r, h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r}^{k-1} \frac{\partial h^{i+1}}{\partial h^i} \right] \frac{\partial h^r}{\partial w_{(h^{r-1},h^r)}},$$

donde $\mathcal{P}$ es el conjunto de caminos desde h^r a o, y $w_{(h^{r-1},h^r)}$ denota el parámetro entre h^{r-1} y h^r. La segunda parte de la ecuación, $\frac{\partial h^r}{\partial w_{(h^{r-1},h^r)}}$, se puede calcular directamente. Mientras que la primera parte, $\frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{\left[h^r, h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r}^{k-1} \frac{\partial h^{i+1}}{\partial h^i}\right]$, puede ser calculada recursivamente, veamos cómo:

$$\Delta(h^r, o) = \frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{\left[h^r, h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r}^{k-1} \frac{\partial h^{i+1}}{\partial h^i} \right] = \frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{\left[h^r, h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r}^{k-1} \frac{\partial h^{i+1}}{\partial h^i} \cdot \frac{\partial h^{r+1}}{\partial h^r} \right]$$

Podemos dividir cualquier camino $P \in \mathcal{P}$ en dos partes, desde h^r hasta h^{r+1} y la parte restante desde h^{r+1} hasta o. Denotando el conjunto de caminos que comparten la primera arista (h^r, h^{r+1}) como $\mathcal{P}_{r+1}$, cualquier camino de este conjunto se caracteriza por el camino restante salvo la primera arista. Al conjunto de caminos restantes lo denotamos como $\mathcal{P}'_{r+1}$. Podemos seguir simplificando la ecuación:

$$\begin{aligned}
\Delta(h^r, o) &= \frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{(h^r, h^{r+1}) \in \mathcal{E}} \frac{\partial h^{r+1}}{\partial h^r} \cdot \left[\sum_{\left[h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r+1}^{k-1} \frac{\partial h^{i+1}}{\partial h^i} \right] \right] \\
&= \sum_{(h^r, h^{r+1}) \in \mathcal{E}} \frac{\partial h^{r+1}}{\partial h^r} \cdot \frac{\partial \mathcal{L}}{\partial o} \cdot \left[\sum_{\left[h^{r+1}, \ldots, h^k, o\right] \in \mathcal{P}} \frac{\partial o}{\partial h^k} \prod_{i=r+1}^{k-1} \frac{\partial h^{i+1}}{\partial h^i} \right] \\
&= \sum_{(h^r, h^{r+1}) \in \mathcal{E}} \frac{\partial h^{r+1}}{\partial h^r} \cdot \Delta(h^{r+1}, o),
\end{aligned}$$

donde $\mathcal{E}$ es el conjunto de todas las conexiones desde h^r hasta h^{r+1} en la capa $(r+1)$-ésima. Para evaluar $\frac{\partial h^{r+1}}{h^r}$, sean a^{r+1} los valores de h^{r+1} justo antes de la función de activación $\alpha()$, es decir, $h^{r+1} = \alpha(a^{r+1})$. Entonces, usando la regla de la cadena obtenemos:

$$\frac{\partial h^{r+1}}{\partial h^r} = \frac{\partial \alpha(a^{r+1})}{\partial h^r} = \frac{\partial \alpha(a^{r+1})}{\partial a^{r+1}} \cdot \frac{\partial a^{r+1}}{\partial h^r} = \alpha'(a^{r+1}) \cdot w_{(h^r, h^{r+1})},$$

con $w_{(h^r, h^{r+1})}$ el parámetro entre h^r y h^{r+1}. Entonces podemos reescribir $\Delta(h^r, o)$ como

sigue:

$$\Delta(h^r, o) = \sum_{(h^r, h^{r+1}) \in \mathcal{E}} \alpha'(a^{r+1}) \cdot w_{(h^r, h^{r+1})} \cdot \Delta(h^{r+1}, o).$$

Y finalmente podemos podemos evaluar $\frac{\partial h^r}{\partial w_{(h^r, h^{r+1})}}$:

$$\frac{\partial h^r}{\partial w_{(h^r, h^{r+1})}} = \alpha'(a^r) \cdot h^{r-1}.$$

Estas dos últimas ecuaciones nos permiten evaluar recursivamente $\frac{\partial \mathcal{L}}{\partial w_{(h^{r-1}, h^r)}}$.

1.3.3. Sobreentrenamiento

Las redes neuronales profundas tienen una gran capacidad de aprendizaje, lo que las hace excepcionalmente buenas en la captura de patrones complejos en los datos. Sin embargo, esta capacidad también puede llevar a que el modelo aprenda demasiado bien los detalles específicos del conjunto de datos de entrenamiento. Como resultado, el modelo puede fallar al generalizar su aprendizaje a nuevos datos, un fenómeno conocido como sobreentrenamiento (*overfitting*). Para mitigar este problema, se han desarrollado varias técnicas.

La regularización de pesos es una técnica que consiste en incluir un término de regularización de los parámetros en la función de pérdida. Este término penaliza los valores de los pesos de la red para mantenerlos pequeños, lo que normalmente implica que el modelo generalice mejor. Dos de los regularizadores más comunes son las normas L_1 y L_2.

Dropout es una técnica de regularización efectiva y simple que, durante el entrenamiento, aleatoriamente pone a cero la salida de algunas neuronas con una probabilidad predefinida p. En cada iteración se determinan aleatoriamente qué neuronas se desactivan, lo que evita que el modelo dependa demasiado de cualquier conjunto de neuronas. En la práctica, dropout se aplica generalmente después de las capas densamente conectadas dentro de la red. Además, esta técnica se utiliza solamente durante el entrenamiento, a la hora de realizar las predicciones se utiliza la red completa.

La normalización por lotes (*Batch Normalization*) es una técnica que estabiliza y acelera el entrenamiento de las redes neuronales al normalizar la entrada de cada capa para que tenga una media cercana a 0 y una desviación estándar cercana a 1. Esto se logra calculando la media y la varianza de los datos en cada lote y utilizando estos valores para normalizar los datos antes de pasar a la siguiente capa.

2. Fundamentos de grafos

En este capítulo se introducirán los conceptos esenciales de la teoría de grafos.

Comenzamos dando la definición de grafo.

Definición 2.1. Un **grafo** se denota por $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, donde:

- $\mathcal{V} = \{v_1, \ldots, v_N\}$ representa un conjunto de N nodos.
- $\mathcal{E} = \{e_1, \ldots, e_M\}$ denota un conjunto de M aristas.

El conjunto $\mathcal{E}$ describe las conexiones entre los nodos en $\mathcal{V}$. Una arista $e \in \mathcal{E}$ conecta dos nodos distintos v_i y v_j. Se dice que un nodo v_i es adyacente a v_j si, y solo si, existe una arista e entre ellos.

Definición 2.2. Para un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, su **matriz de adyacencia** se denota por $\mathbf{A}$, donde $\mathbf{A} \in \{0,1\}^{N \times N}$. La entrada $\mathbf{A}_{i,j}$ indica la conectividad entre los nodos v_i y v_j, con $\mathbf{A}_{i,j} = 1$ si v_i es adyacente a v_j, y $\mathbf{A}_{i,j} = 0$ en caso contrario.

Definición 2.3. En un **grafo no dirigido**, la relación de adyacencia es simétrica; es decir, para cualquier par de nodos v_i y v_j, si v_i es adyacente a v_j, entonces v_j es adyacente a v_i. Esto implica que la matriz de adyacencia $\mathbf{A}$ de un grafo no dirigido es simétrica, cumpliendo que $\mathbf{A}_{i,j} = \mathbf{A}_{j,i}$ para todo i, j.

Ejemplo 2.1. Sea $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ y la siguiente matriz de adyacencia:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Entonces podemos visualizar el grafo como sigue:

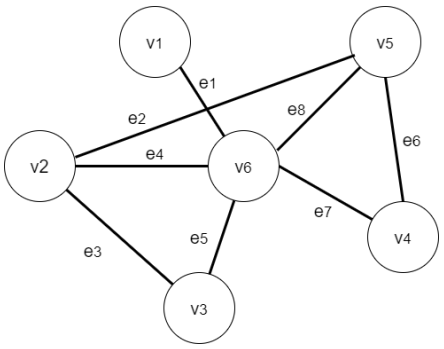


Figura 2.1.: Ejemplo de un grafo con 6 nodos y 8 aristas

2.1. Medidas

En esta sección presentaremos las diferentes medidas que caracterizan a los grafos. Estas medidas nos ayudarán a comprender la estructura de los grafos.

2.1.1. Grado

Definición 2.4. En un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, el **grado** de un nodo $v_i \in \mathcal{V}$, denotado como $d(v_i)$, es el número total de aristas que inciden en v_i, es decir, el número de nodos adyacentes a v_i. El grado puede ser calculado a partir de la matriz de adyacencia $\mathbf{A}$:

$$d(v_i) = \sum_{j=1}^{N} \mathbf{A}_{i,j},$$

donde N es el número total de nodos en el grafo.

Definición 2.5. El **conjunto de vecinos** de un nodo v_i, representado como $\mathcal{N}(v_i)$, incluye todos los nodos que están directamente conectados a v_i por una arista. Por tanto, $d(v_i) = |\mathcal{N}(v_i)|$.

2.1.2. Conexión

Definición 2.6. Un **camino** en un grafo es una sucesión de nodos $v_1, v_2, \ldots, v_{n+1}$ y aristas $e_1, e_2, \ldots, e_n$, donde cada arista e_i conecta el nodo que le precede v_i con el nodo siguiente v_{i+1}. La longitud de un camino es el número de aristas que contiene.

Definición 2.7. Un **recorrido** es un camino sin aristas repetidas.

Definición 2.8. Un **camino simple** es un camino en el que todos los nodos, excepto posiblemente el inicial y el final, son distintos entre sí.

Definición 2.9. Un **subgrafo** $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ de un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ está formado por un subconjunto de nodos $\mathcal{V}' \subset \mathcal{V}$ y un subconjunto de aristas $\mathcal{E}' \subset \mathcal{E}$, donde $\mathcal{V}'$ incluye todos los nodos incidentes en las aristas de $\mathcal{E}'$.

Definición 2.10. Un subgrafo $\mathcal{G}'$ es una **componente conexa** de $\mathcal{G}$ si para cualquier par de nodos en $\mathcal{V}'$, existe un camino entre ellos en $\mathcal{G}'$, y ningún nodo en $\mathcal{V}'$ es adyacente a algún nodo en $\mathcal{V} \setminus \mathcal{V}'$.

Definición 2.11. Un grafo $\mathcal{G}$ es **conexo** si tiene exactamente una componente conexa, lo que significa que existe un camino entre cualquier par de nodos.

Dado un par de nodos $v_i, v_s \in V$ de un grafo $\mathcal{G}$, denotamos al conjunto de caminos simples desde el nodo v_i al nodo v_s como $\mathcal{P}_{i,j}$.

Definición 2.12. El **camino más corto** entre el nodo v_i y el nodo v_j se define como sigue:

$$p_{i,j} = \arg \min_{p \in \mathcal{P}_{i,j}} |p|,$$

donde $|p|$ denota la longitud del camino p.

Definición 2.13. El **diámetro** de un grafo $\mathcal{G}$ es la mayor longitud entre todos los caminos más cortos del grafo,

$$\text{diámetro}(\mathcal{G}) = \max_{v_i, v_j \in \mathcal{V}} \min_{p \in \mathcal{P}_{i,j}} |p|.$$

2.1.3. Centralidad

La centralidad de un nodo es una medida de su importancia dentro de la estructura del grafo. En esta sección, presentamos diversas métricas de centralidad utilizadas para evaluar la relevancia de los nodos.

Definición 2.14. La **centralidad de grado** de un nodo v_i, $c_d(v_i)$, es su grado,

$$c_d(v_i) = d(v_i) = \sum_{j=1}^{N} \mathbf{A}_{i,j}.$$

Esta métrica trata a todos los vecinos de un nodo por igual, sin tener en cuenta la importancia de cada uno.

Definición 2.15. La **centralidad de vector propio** de un nodo v_i se define como:

$$c_e(v_i) = \frac{1}{\lambda} \sum_{j=1}^{N} \mathbf{A}_{i,j} \cdot c_e(v_j),$$

que se puede reescribir de forma matricial:

$$\mathbf{c}_e = \frac{1}{\lambda} \mathbf{A} \cdot \mathbf{c}_e,$$

donde $\mathbf{c}_e \in \mathbb{R}^N$ es el vector de centralidades y λ es el mayor valor propio de $\mathbf{A}$, de acuerdo al teorema de Perron-Frobenius [Per07, Fro12]. La elección de este λ implica que su correspondiente vector propio, $\mathbf{c}_e$, tenga todos sus elementos positivos.

Definición 2.16. La **centralidad de intermediación** cuantifica la frecuencia que un nodo aparece en los caminos más cortos de otros nodos. Formalmente, se define:

$$c_b(v_i) = \sum_{v_s \neq v_i \neq v_t} \frac{\sigma_{s,t}(v_i)}{\sigma_{s,t}},$$

donde $\sigma_{s,t}$ el número total de caminos más cortos de v_s a v_t y $\sigma_{s,t}(v_i)$ es el número de esos caminos que pasan por v_i.

La centralidad de intermediación se normaliza para permitir comparaciones entre grafos de diferentes tamaños, resultando en:

$$c_{nb}(v_i) = \frac{2 \sum_{v_s \neq v_i \neq v_t} \frac{\sigma_{s,t}(v_i)}{\sigma_{s,t}}}{(N-1)(N-2)},$$

considerando $\frac{(N-1)(N-2)}{2}$ como el número total de pares de nodos en un grafo no dirigido, que coincide con el máximo posible de esta medida.

2.2. Teoría espectral de grafos

La teoría espectral de grafos estudia las propiedades de un grafo en términos de los valores y vectores propios de matrices asociadas al grafo, como su matriz de adyacencia y Laplaciana. En esta sección introducimos la matriz Laplaciana y exploramos algunas de sus propiedades esenciales.

Definición 2.17. Dado un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, con $\mathbf{A}$ su matriz de adyacencia, se define su **matriz Laplaciana L** como:

$$\mathbf{L} = \mathbf{D} - \mathbf{A},$$

donde $\mathbf{D}$ es la matriz diagonal que contiene los grados de los nodos, $\mathbf{D} = \text{diag}(d(v_1), \ldots, d(v_N))$.

La matriz Laplaciana de grafos no dirigidos es simétrica debido a que tanto $\mathbf{D}$ como $\mathbf{A}$ lo son. Sea $\mathbf{f}$ un vector cuya componente i-ésima corresponde al nodo v_i, la operación $\mathbf{Lf}$ resulta en un nuevo vector $\mathbf{h}$:

$$\mathbf{h} = \mathbf{Lf} = (\mathbf{D} - \mathbf{A})\mathbf{f} = \mathbf{Df} - \mathbf{Af}.$$

Por tanto, el elemento i-ésimo de $\mathbf{h}$ es:

$$\mathbf{h}_i = d(v_i) \cdot \mathbf{f}_i - \sum_{j=1}^{N} \mathbf{A}_{i,j} \cdot \mathbf{f}_i = d(v_i) \cdot \mathbf{f}_i - \sum_{v_j \in \mathcal{N}(v_i)} \mathbf{A}_{i,j} \cdot \mathbf{f}_i = \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}_i - \mathbf{f}_j),$$

lo que significa que $\mathbf{h}_i$ es la suma de las diferencias entre v_i y sus vecinos. Calculemos ahora $\mathbf{f}^T\mathbf{Lf}$:

$$\mathbf{f}^T\mathbf{Lf} = \sum_{v_i \in \mathcal{V}} \mathbf{f}_i \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}_i - \mathbf{f}_j) = \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}_i \cdot \mathbf{f}_i - \mathbf{f}_i \cdot \mathbf{f}_j) =$$

$$= \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (-\frac{1}{2}\mathbf{f}_i \cdot \mathbf{f}_i - \mathbf{f}_i \cdot \mathbf{f}_j + \frac{1}{2}\mathbf{f}_j \cdot \mathbf{f}_j) = \frac{1}{2} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{N}(v_i)} (\mathbf{f}_i - \mathbf{f}_j)^2.$$

Entonces, $\mathbf{f}^T\mathbf{Lf}$ es la suma de las diferencias al cuadrado entre nodos adyacentes. Además, a partir de este resultado deducimos que la matriz Laplaciana es semidefinida positiva.

A continuación, vamos a ver algunas de las propiedades principales relacionados con los valores y vectores propios de la Laplaciana.

Teorema 2.1. *Los valores propios de la matriz Laplaciana $\mathbf{L}$ de un grafo $\mathcal{G}$ son no negativos.*

Esta propiedad se deriva directamente del hecho de que $\mathbf{L}$ es semidefinida positiva.

Para un grafo $\mathcal{G}$ con N nodos, hay en total N valores propios, considerando su multiplicidad. Es importante señalar que siempre existe al menos un valor propio igual a cero. Por ejemplo, el vector $\mathbf{u}_1 = \frac{1}{\sqrt{N}}(1, \ldots, 1)$ satisface $\mathbf{Lu}_1 = \mathbf{0}$, lo que indica que es un vector propio correspondiente al valor propio cero. Por conveniencia, ordenaremos los valores propios en orden no decreciente $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$, y los correspondientes vectores propios normalizados $\mathbf{u}_1, \ldots, \mathbf{u}_N$.

Teorema 2.2. *La multiplicidad del valor propio cero de la matriz Laplaciana $\mathbf{L}$ es igual al número de componentes conexas del grafo $\mathcal{G}$.*

La demostración de este teorema se puede encontrar en [MWAT19, pág. 28].

2.3. Procesamiento de señales en grafos

El procesamiento de señales en grafos extiende las nociones clásicas del procesamiento de señales a los datos estructurados en grafos. Esto permite capturar las características de los nodos y la conectividad entre ellos. Una señal sobre un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consiste en una función $f : \mathcal{V} \to \mathbb{R}^{N \times d}$, donde d es la dimensión del vector asociado a cada nodo. Sin pérdida de generalidad, en esta sección, asumiremos que $d = 1$ y denotamos $\mathbf{f} \in \mathbb{R}^N$ como la imagen de f, con $\mathbf{f}_i$ la componente correspondiente al nodo v_i.

Podemos analizar la señal de grafos tanto en el dominio espacial como en el espectral. La transformación entre estos dominios se facilita mediante la transformada de Fourier para grafos y la transformada inversa. La transformada de Fourier clásica descompone una señal temporal $f(t)$ en componentes de frecuencia mediante el uso de exponenciales complejas $\exp(-2\pi i t\xi)$, donde ξ representa la frecuencia:

$$\hat{f}(\xi) = \langle f(t), \exp(-2\pi i t\xi) \rangle = \int_{-\infty}^{\infty} f(t) \exp(-2\pi i t\xi) dt$$

Análogamente, se define la transformada de Fourier de Grafos de una señal $\mathbf{f}$ sobre un grafo $\mathcal{G}$ como:

$$\hat{\mathbf{f}}_l = \langle \mathbf{f}, \mathbf{u}_l \rangle = \sum_{i=1}^{N} \mathbf{f}_i \mathbf{u}_{l,i},$$

donde $\mathbf{u}_{l,i}$ es la i-ésima componente del l-ésimo vector propio de la matriz Laplaciana $\mathbf{L}$ del grafo. El correspondiente valor propio λ_l representa un concepto parecido a la frecuencia en el análisis espectral clásico. Los vectores propios $\{\mathbf{u}_l\}$ forman una base ortogonal que permite la descomposición y reconstrucción de señales en el dominio espectral.

La transformada de Fourier de Grafos se puede expresar en forma matricial:

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f},$$

donde $\mathbf{U}$ es la matriz que tiene como columnas a los vectores propios $\mathbf{u}_l$.

Para recuperar la señal en el dominio espacial desde su representación espectral, utilizamos la transformada inversa de Fourier de Grafos:

$$\mathbf{f} = \mathbf{U}\hat{\mathbf{f}},$$

que reconstruye la señal $\mathbf{f}$ a partir de sus coeficientes de Fourier $\hat{\mathbf{f}}$ utilizando la base de vectores propios $\mathbf{U}$.

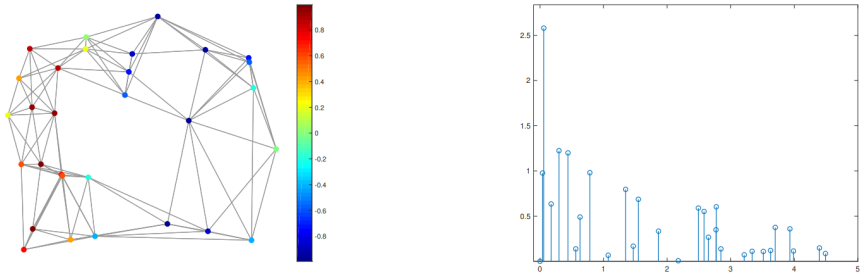


Figura 2.2.: Representaciones de la señal de un grafo en el dominio espacial (izquierda) y en el dominio espectral (derecha). Imagen obtenida de [MT21].

2.4. Tipos de grafos

En esta sección, describiremos y daremos ejemplos de los tipos de grafos más conocidos.

2.4.1. Grafo heterogéneo

Definición 2.18. Un **grafo heterogéneo** $\mathcal{G}$ consiste en un conjunto $\mathcal{V} = \{v_1, \ldots, v_N\}$ y un conjunto de aristas $\mathcal{E} = \{e_1, \ldots, e_M\}$ donde cada nodo y cada arista están asociados a un tipo específico.

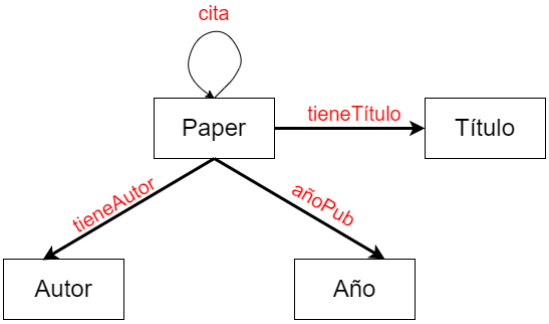


Figura 2.3.: Ejemplo de grafo heterogéneo.

2.4.2. Grafo bipartido

Definición 2.19. Un **grafo bipartido** $\mathcal{G}$ es aquel cuyo conjunto de nodos $\mathcal{V}$ se divide en dos subconjuntos disjuntos $\mathcal{V}_1$ y $\mathcal{V}_2$ donde cada lado en $\mathcal{E}$ conecta un nodo en $\mathcal{V}_1$ con un nodo en $\mathcal{V}_2$.

Formalmente, un grafo $\mathcal{G}$ es bipartido, si y solo si, $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ y $v_{e,1} \in \mathcal{V}_1$ mientras que $v_{e,2} \in \mathcal{V}_2$ para todo $e = (v_{e,1}, v_{e,2}) \in \mathcal{E}$.

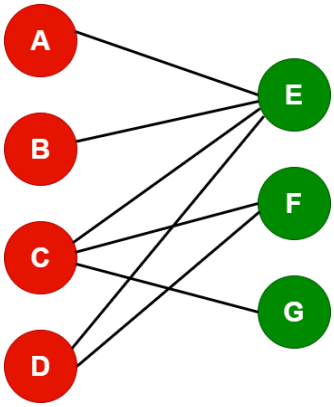


Figura 2.4.: Ejemplo de grafo bipartido.

2.4.3. Grafo multidimensional

Definición 2.20. Un **grafo multidimensional** consiste en un conjunto de N nodos $\mathcal{V}$ y D conjuntos distintos de lados. Cada conjunto de lados, $\mathcal{E}_d$, describe el tipo de relación entre los nodos en la dimensión d-ésima. Estos D tipos se pueden expresar con D matrices de adyacencia, cada una correspondiente a una dimensión.

2.4.4. Grafo signado

Definición 2.21. Los **grafos signados** incluyen lados de dos tipos: positivos y negativos. Dado $\mathcal{G} = (\mathcal{V}, \mathcal{E}^+, \mathcal{E}^-)$, donde $\mathcal{V}$ es el conjunto de N nodos mientras que $\mathcal{E}^+, \mathcal{E}^- \subset \mathcal{V} \times \mathcal{V}$ denotan los conjuntos de lados positivos y negativos, respectivamente, con $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$. la matriz de adyacencia refleja estas relaciones, donde $\mathbf{A}_{i,j} = 1$ cuando existe un lado positivo entre el nodo v_i y el nodo v_j, y $\mathbf{A}_{i,j} = -1$ si el lado es negativo.

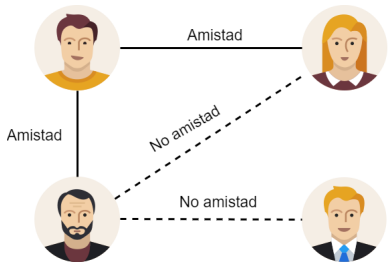


Figura 2.5.: Ejemplo de grafo signado.

2.4.5. Hipergrafo

Definición 2.22. Un **hipergrafo** $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ consta de un conjunto de nodos $\mathcal{V}$, un conjunto de hiperaristas $\mathcal{E}$, y una matriz diagonal de pesos $\mathbf{W} \in \mathbb{R}^{|\mathcal{E}| \times |\mathcal{E}|}$ donde $\mathbf{W}_{j,j}$ denota el peso de la hiperarista e_j. A diferencia de los grafos tradicionales, las hiperaristas pueden conectar más de dos nodos.

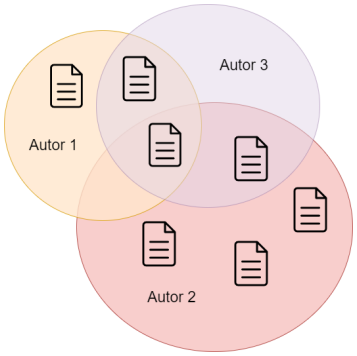


Figura 2.6.: Ejemplo de hipergrafo. Imagen modificada de [MT21].

2.4.6. Grafo dinámico

Definición 2.23. En un **grafo dinámico** $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ cada nodo y/o lado está asociado con un instante de tiempo, que representa el momento en que surgieron.

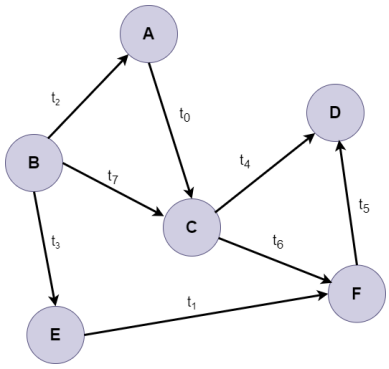


Figura 2.7.: Ejemplo de grafo dinámico.

3. Redes neuronales para grafos

Este capítulo estudiará las redes neuronales aplicadas a grafos. Estas técnicas son esenciales para el análisis de datos como redes sociales, moléculas químicas y sistemas de recomendación. Para el desarrollo de este capítulo he utilizado [MT21] como referencia principal.

Las redes redes neuronales para grafos (GNNs) constituyen un conjunto de técnicas diseñadas para aplicar redes neuronales profundas a datos estructurados en grafos. Las primeras investigaciones en este campo surgen a comienzos del siglo XXI, cuando surge el primer modelo inicial propuesto por Gori, Monfardini y Scarselli en 2005, [GMS05]. Para tareas centradas en nodos, las GNNs apuntan a aprender buenas características para cada nodo. Mientras que para tareas centradas en grafos, buscan aprender características representativas del grafo completo.

Introduciremos la notación utilizada en este trabajo. Representamos un grafo como $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ y a su matriz de adyacencia con N nodos como $\mathbf{A}$. Las características asociadas a los nodos se denotan como $\mathbf{F} \in \mathbb{R}^{N \times d}$, donde cada fila de $\mathbf{F}$ corresponde a un nodo y d es la dimensión de las características.

El proceso de aprendizaje generalmente aprovecha tanto las características de los nodos de entrada como la estructura del grafo. Este proceso se puede describir mediante la fórmula:

$$\mathbf{F}^{(\text{of})} = h(\mathbf{A}, \mathbf{F}^{(\text{if})}),$$

donde $\mathbf{F}^{(\text{of})}$ y $\mathbf{F}^{(\text{if})}$ denotan las matrices de las características de entrada y salida, respectivamente. A la operación h, que toma las características y la estructura, y tiene como salida un nuevo conjunto de características, se le conoce como filtro.

En la clasificación de nodos, múltiples filtros se aplican secuencialmente junto con capas de activación para obtener las características finales de nodos. Sin embargo, para clasificación de grafos, necesitamos otras operaciones para extraer las características a nivel del grafo. Similar al caso de de las CNNs clásicas, existe una operación de pooling, la cual devolverá una nueva matriz de adyacencia y unas características:

$$\mathbf{A}^{(\text{op})}, \mathbf{F}^{(\text{op})} = \text{pool}(\mathbf{A}^{(\text{ip})}, \mathbf{F}^{(\text{ip})}),$$

donde $\mathbf{A}^{(\text{ip})} \in \mathbb{R}^{N_{ip} \times N_{ip}}$, $\mathbf{f}^{(\text{ip})} \in \mathbb{R}^{N_{ip} \times d_{ip}}$ y $\mathbf{A}^{(\text{op})} \in \mathbb{R}^{N_{op} \times N_{op}}$, $\mathbf{f}^{(\text{op})} \in \mathbb{R}^{N_{op} \times d_{op}}$ son las matrices de adyacencia y las características antes y despúes del pooling, respectivamente.

3.1. Filtros

Hay dos categorías de filtros de grafos, espaciales y espectrales. Los primeros se centran en las conexiones entre los nodos para procesar las características, mientras que los filtros

espectrales utilizan la teoría espectral de grafos para hacer el filtro en el dominio espectral.

3.1.1. Filtros espectrales

La idea del filtrado espectral en grafos es modular las frecuencias de la señal del grafo para que algunas de sus componentes sean amplificadas o mantenidas mientras que se eliminen o reduzcan otras. Por tanto, lo primero que debemos hacer es aplicar la transformada de Fourier en grafos a la señal $\mathbf{f} \in \mathbb{R}^N$ del grafo, luego modularemos estos coeficientes y reconstruiremos la señal en el dominio espacial.

La transformada para una señal $\mathbf{f} \in \mathbb{R}^N$ definida en un grafo $\mathcal{G}$ se define como:

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f},$$

donde la matriz $\mathbf{U}$ está compuesta por los vectores propios de la matriz Laplaciana de $\mathcal{G}$. El i-ésimo elemento de $\hat{\mathbf{f}}$, $\mathbf{u}_i$, se corresponde con la i-ésima componente de Fourier del grafo con frecuencia λ_i, que es el valor propio asociado a $\mathbf{u}_i$. Filtramos los coeficientes de Fourier como sigue:

$$\hat{\mathbf{f}}'[i] = \hat{\mathbf{f}}[i] \cdot \gamma(\lambda_i), \quad \text{con } i = 1, \ldots, N,$$

donde $\gamma(\lambda_i)$ es una función que determina cómo modular las frecuencias. Este proceso de forma matricial se expresa:

$$\hat{\mathbf{f}}' = \gamma(\mathbf{\Lambda}) \cdot \hat{\mathbf{f}} = \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^\top \mathbf{f},$$

donde $\mathbf{\Lambda}$ es una matriz diagonal con los valores propios de la matriz Laplaciana.

Podemos ahora reconstruir la señal al dominio del grafo usando la transformada inversa de Fourier en grafos:

$$\mathbf{f}' = \mathbf{U}\hat{\mathbf{f}}' = \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^\top \mathbf{f},$$

donde $\mathbf{f}'$ es la señal de grafo filtrada. Normalmente nos referiremos a la función $\gamma(\mathbf{\Lambda})$ como filtro.

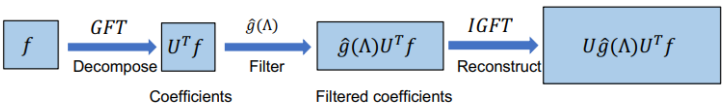


Figura 3.1.: Proceso de filtrado espectral. Imagen obtenida de [MT21].

Una vez introducida la operación de filtro en grafos, lo interesante es que la red aprenda estos filtros, ya que usualmente no se conocen qué frecuencias son más importantes. Un intento natural pero poco eficiente es definir un parámetro para cada nodo del grafo, lo que requiere mucha memoria. Por tanto, surgieron los llamados filtros polinómicos [DBV17]. La función $\gamma(\cdot)$ se puede modelar con un polinomio truncado de orden K:

$$\gamma(\lambda_l) = \sum_{k=0}^{K} \theta_k \lambda_l^k.$$

En forma matricial:

$$\gamma(\mathbf{\Lambda}) = \sum_{k=0}^{K} \theta_k \mathbf{\Lambda}^k,$$

donde el número de parámetros es $K+1$, que no depende del tamaño de $\mathcal{G}$. Además, la expresión $\mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^\top$ se puede simplificar en un polinomio de la matriz Laplaciana, lo que significa que no hará falta una descomposición en vectores propios y que el operador de filtro polinómico solamente necesita un número pequeño de nodos para calcular cada elemento de $\mathbf{f}'$.

Aplicando el operador de filtro polinómico en $\mathbf{f}$, obtenemos $\mathbf{f}'$:

$$\mathbf{f}' = \mathbf{U} \cdot \gamma(\mathbf{\Lambda}) \cdot \mathbf{U}^\top \mathbf{f} = \mathbf{U} \cdot \sum_{k=0}^{K} \theta_k \Lambda^k \cdot \mathbf{U}^\top \mathbf{f} = \sum_{k=0}^{K} \theta_k \mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^\top \mathbf{f}$$

Veamos también que $\mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^\top = \mathbf{L}^k$:

$$\mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^\top = \mathbf{U} \cdot (\mathbf{\Lambda}\mathbf{U}^\top\mathbf{U})^k \cdot \mathbf{U}^\top = (\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^\top) \cdots (\mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^\top) = \mathbf{L}^k$$

Por lo que la ecuación anterior queda simplificada:

$$\mathbf{f}' = \sum_{k=0}^{K} \theta_k \mathbf{U} \cdot \mathbf{\Lambda}^k \cdot \mathbf{U}^\top \mathbf{f} = \sum_{k=0}^{K} \theta_k \mathbf{L}^k \mathbf{f}.$$

El valor de la señal de salida en el nodo v_i es una combinación lineal de la señal original en todos los nodos ajustada al peso $\sum_{k=0}^{K} \theta_k \mathbf{L}^k_{i,j}$ y se calcula como sigue:

$$\mathbf{f}'[i] = \sum_{v_j \in \mathcal{V}} \left(\sum_{k=0}^{K} \theta_k \mathbf{L}^k_{i,j} \right) \mathbf{f}[j]. \tag{3.1}$$

Sin embargo, vamos a ver que no todos los nodos se utilizan en este cálculo, solamente están involucrados los nodos en un radio de K saltos del nodo v_i.

Lema 3.1. *Sea $\mathcal{G}$ un grafo y $\mathbf{L}$ su matriz Laplaciana. Entonces el elemento (i, j) de la potencia k-ésima de la matriz Laplaciana $\mathbf{L}^k_{i,j} = 0$ si la longitud del camino más corto desde el nodo v_i hasta el nodo v_j es mayor a k, esto es, si $dist(v_i, v_j) > k$.*

Por lo tanto, podemos reorganizar la ecuación (3.1) para que incluya solamente los nodos necesarios:

$$\mathbf{f}'[i] = b_{i,j}\mathbf{f}[i] + \sum_{v_j \in \mathcal{N}^K(v_i)} b_{i,j}\mathbf{f}[j],$$

donde $\mathcal{N}^K(v_i)$ es el conjunto de nodos que se encuentran a una distancia menor o igual a K del nodo v_i, que denominaremos vecindario de radio k de v_i, y el parámetro $b_{i,j}$ que se define:

$$b_{i,j} = \sum_{k=\text{dis}(v_i,v_j)}^{K} \theta_k \mathbf{L}^k_{i,j}.$$

La mayor limitación de estos filtros polinómicos es que la base $B = \{1, x, x^2, \ldots\}$ no es

ortogonal, por lo que los coeficientes dependen de otros, haciendo el aprendizaje inestable a perturbaciones. Como solución a este problema surge el polinomio de Chebyshev, que sí tiene una base ortogonal.

Los polinomios de Chebyshev $T_k(y)$ se pueden generar recurrentemente de la siguiente manera:

$$T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y),$$

con $T_0(y) = 1$ y $T_1(y) = y$. Para $y \in [-1, 1]$, los polinomios de Chebyshev se pueden representar con la expresión trigonométrica:

$$T_k(y) = \cos(k \arccos(y)),$$

lo que quiere decir que cada $T_k(y)$ está acotado por el intervalo $[-1, 1]$. Además, los polinomios de Chebyshev satisfacen lo siguiente:

$$\int_{-1}^{1} \frac{T_l(y)T_m(y)}{\sqrt{1-y^2}} dy = \begin{cases} \delta_{l,m}\pi/2 & \text{si } m, l > 0, \\ \pi & \text{si } m = l = 0, \end{cases}$$

donde $\delta_{l,m} = 1$ solo si $l = m$, en cualquier otro caso $\delta_{l,m} = 0$. Esta ecuación indica que los polinomios de Chebyshev son ortogonales entre sí, por lo que forman una base ortonormal.

Como el dominio de estos polinomios es $[-1, 1]$, para aproximar el filtro con los polinomios de Chebyshev, debemos reescalar y desplazar los valores propios de la matriz Laplaciana:

$$\tilde{\lambda}_l = \frac{2 \cdot \lambda_l}{\lambda_{max}} - 1,$$

donde $\lambda_{max} = \lambda_N$ es el mayor valor propio. De esta forma, los valores propios se normalizan en el intervalo $[-1, 1]$. En forma matricial, la matriz diagonal de estos valores se expresa:

$$\tilde{\mathbf{\Lambda}} = \frac{2\mathbf{\Lambda}}{\lambda_{max}} - \mathbf{I},$$

donde $\mathbf{I}$ es la matriz identidad.

El filtro Cheby (*Cheby-Filter*), el cual se parametriza con los polinomios de Chebyshev truncados se puede formular como sigue:

$$\gamma(\mathbf{\Lambda}) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Lambda}}).$$

Y el proceso de aplicar este filtro a una señal $\mathbf{f}$ se puede definir así:

$$\mathbf{f}' = \mathbf{U} \cdot \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{\Lambda}})\mathbf{U}^\top \mathbf{f} = \sum_{k=0}^{K} \theta_k \mathbf{U} T_k(\tilde{\mathbf{\Lambda}})\mathbf{U}^\top \mathbf{f}.$$

Esta ecuación se puede simplificar más gracias al siguiente teorema.

Teorema 3.1. *Sea la matriz Laplaciana* $\boldsymbol{L}$ *de un grafo* $\mathcal{G}$*, se cumple la siguiente ecuación para* $k \geq 0$

$$\boldsymbol{U}T_k(\tilde{\boldsymbol{\Lambda}})\boldsymbol{U}^\top = T_k(\tilde{\boldsymbol{L}}),$$

donde $\tilde{\boldsymbol{L}} = \frac{2\boldsymbol{L}}{\lambda_{max}} - \boldsymbol{I}$

La demostración se encuentra en [MT21, pág. 119].

Este resultado nos permite simplificar la ecuación anterior:

$$\mathbf{f}' = \sum_{k=0}^{K} \theta_k \mathbf{U} T_k(\tilde{\boldsymbol{\Lambda}}) \mathbf{U}^\top \mathbf{f} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\mathbf{L}}) \mathbf{f}.$$

Por tanto, el filtro de Chebyshev tiene las ventajas del filtro polinómico y además es más estable a perturbaciones.

Todo el proceso para señales de grafo se ha explicado con un solo canal, sin embargo, normalmente cada nodo tiene un vector de características. Para extender los filtros a señales multi-canales, utilizamos las señales de todos los canales de entrada para generar la señal de salida:

$$\mathbf{f}_{\text{out}} = \sum_{d=1}^{d_{in}} \mathbf{U} \cdot \gamma_d(\boldsymbol{\Lambda}) \cdot \mathbf{U}^\top \mathbf{F}_{:,d},$$

donde $\mathbf{f}_{\text{out}} \in \mathbb{R}^N$ es la señal de salida de un canal y $\mathbf{F}_{:,d} \in \mathbb{R}^N$ es el canal d-ésimo de la señal de entrada. Por tanto, el proceso de filtrado puede verse como aplicar el filtro en cada canal de entrada y sumar todos los resultados. Para generar d_{out} canales en la señal de salida, se utilizan d_{out} filtros:

$$\mathbf{F}'_{:,j} = \sum_{d=1}^{d_{in}} \mathbf{U} \cdot \gamma_{j,d}(\boldsymbol{\Lambda}) \cdot \mathbf{U}^\top \mathbf{F}_{:,d}, \quad \text{para } j = 1, \ldots, d_{out},$$

donde $\mathbf{F}'_{:,j}$ es el canal j-ésimo de la señal de salida.

3.1.2. Filtros espaciales

En el primer modelo de GNN, cada nodo estaba asociado con una etiqueta de entrada. Para el nodo v_i, su etiqueta correspondiente se denota l_i. Para el filtrado, las características de entrada se denotan como $\mathbf{F}$, donde la i-ésima fila son las características del nodo v_i; y los atributos de salida se representan como $\mathbf{F}'$. La operación de filtrado para el nodo v_i se define así:

$$\mathbf{F}'_i = \sum_{v_j \in \mathcal{N}(v_i)} g(l_i, \mathbf{F}_j, l_j),$$

donde $g(\cdot, \cdot, \cdot)$ es una función paramétrica llamada función de transición local, que es localizada espacialmente. El filtrado para el nodo v_i solamente involucra sus vecinos de un salto.

3.2. Pooling

De manera similar a las CNNs, las capas de pooling en grafos son un método para generar representaciones a nivel de grafo. Los primero diseños fueron planos, esto es, se generaba esta representación a partir de las representaciones de los nodos en un solo paso. Por ejemplo, las capas de average pooling y max pooling introducidas anteriormente se pueden adaptar a este tipo de redes aplicándolas a cada canal de características. Más tarde surgió el pooling jerárquico, formado por varias capas de pooling cada una con numerosos filtros.

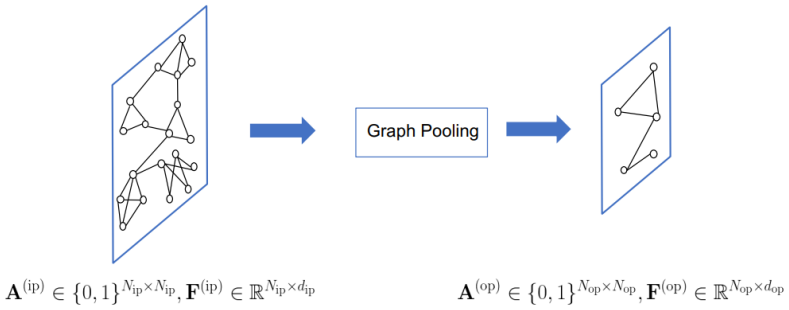


Figura 3.2.: Operación de pooling en grafos. Imagen obtenida de [MT21]

3.2.1. Pooling plano de grafos

En las capas de pooling plano solamente se genera un solo nodo que resume todo el grafo. El proceso se puede definir como:

$$\mathbf{f}_{\mathcal{G}} = \text{pool}(\mathbf{A}^{(\text{ip})}, \mathbf{F}^{(\text{op})}),$$

donde $\mathbf{f}_{\mathcal{G}} \in \mathbb{R}^{1 \times d_{\text{op}}}$ es la representación del grafo. La operación de max pooling se puede expresar así:

$$\mathbf{f}_{\mathcal{G}} = \max(\mathbf{F}^{(\text{ip})}),$$

donde la operación de máximo se aplica a cada canal:

$$\mathbf{f}_{\mathcal{G}}[i] = \max(\mathbf{F}^{(\text{ip})}_{:,i}).$$

De forma análoga el pooling promedio se aplica también a cada canal:

$$\mathbf{f}_{\mathcal{G}} = \text{ave}(\mathbf{F}^{(\text{ip})}).$$

3.2.2. Pooling jerárquico de grafos

El objetivo de las capas de pooling jerárquico es mantener la información jerárquica de la estructura del grafo. Esto se consigue reduciendo el tamaño del grafo poco a poco.

3.2.2.1. Pooling basado en reducción

Esta técnica se basa en la selección de un conjunto de nodos importantes y resumir la estructura y las características de acuerdo a esos nodos. Se puede dividir en tres etapas diferenciadas: 1) elección de la medida para la reducción; 2) generación de la estructura para el grafo reducido; 3) generación de las características de los nuevos nodos. Ejemplos de capas de este tipo son: gPool [GJ19], y SAGPool [LLK19].

3.2.2.2. Pooling basado en supernodos

A diferencia del pooling anterior, donde la información de los nodos descartados era perdida, esta técnica intenta reducir el grafo generando supernodos. Se puede dividir en tres partes: 1) generación de los supernodos; 2) generación de la estructura del grafo reducido; 3) generación de las características para los nuevos nodos. Los supernodos se definen a partir de los nodos de entrada tras aplicar un algoritmo de clustering, donde estos clústers serán los supernodos. Ejemplos de capas de este tipo: diffpool [YYM+19] y EigenPooling [MWAT19].

3.3. Tareas de aprendizaje automático con grafos

En esta sección se explicarán brevemente las 6 tareas más importantes de aprendizaje que se pueden realizar con redes neuronales para grafos y se darán ejemplos sencillos de cada una.

3.3.1. Clasificación de grafos

Sea un conjunto de grafos etiquetados $\mathcal{D} = \{(\mathcal{G}_i, y_i)\}$ con y_i la etiqueta del grafo $\mathcal{G}_i$. El objetivo de la clasificación de grafos es aprender una función $\phi : \mathcal{D} \to \mathcal{C}$, tal que ϕ prediga la clase de los grafos no etiquetados.

Por ejemplo, en la predicción de tráfico, cada grafo podría representar una red de carreteras, donde las etiquetas indican el nivel de congestión. La función ϕ podría predecir la congestión en redes de carreteras no observadas previamente.

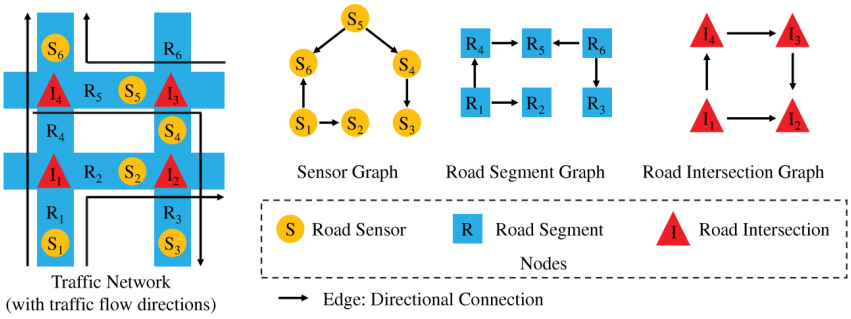


Figura 3.3.: Ejemplos de grafos a nivel de carreteras. Imagen obtenida de [JL22].

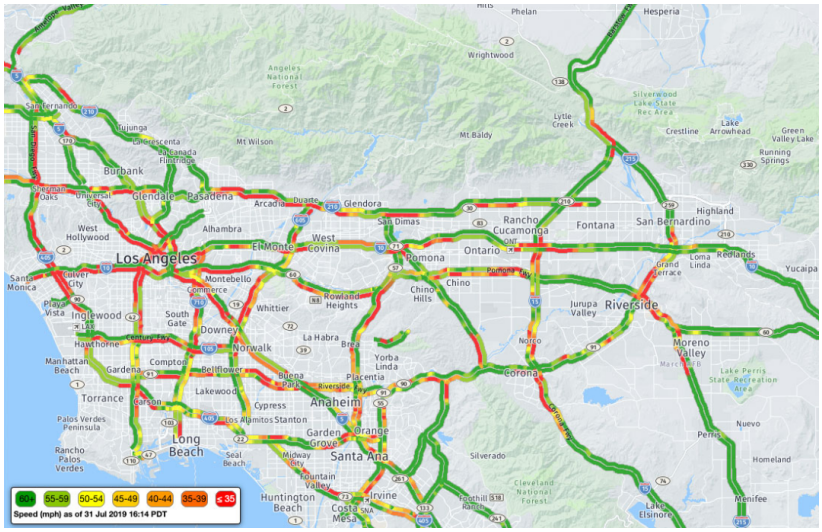


Figura 3.4.: Red de carreteras de Estados Unidos donde cada sensor es un nodo. Imagen obtenida de [JL22].

3.3.2. Clasificación de nodos

Sea $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ un grafo, cuyo conjunto de nodos se divide en dos subconjuntos disjuntos: $\mathcal{V}_l$, los nodos etiquetados, de los que conocemos la clase a la que pertenecen; y $\mathcal{V}_u$, los nodos no etiquetados. El objetivo de la clasificación de nodos es aprender una aplicación $\phi : \mathcal{V} \to \mathcal{C}$, donde $\mathcal{C}$ es el conjunto de clases posibles, de tal manera que ϕ pueda predecir con precisión la clase de los nodos no etiquetados.

Un ejemplo con el que realizaremos los experimentos es el dataset Cora. Este conjunto de datos consiste en citas de documentos científicos, donde los nodos son los documentos y las aristas las citaciones entre ellos. Cada documento pertenece a una de entre siete clases y el objetivo será predecir la clase de los documentos no etiquetados.

3.3.3. Predicción de enlaces

Sea $\mathcal{G}$ un grafo y sea $\mathcal{M}$ todas las posibles aristas entre los nodos de $\mathcal{V}$. Entonces, el complementario de $\mathcal{E}$ con respecto a $\mathcal{M}$ contiene los lados no observados entre los nodos. El objetivo de la predicción de aristas es predecir cuáles de estos enlaces no observados existen realmente.

Un ejemplo de predicción de enlaces es un sistema recomendador, como PinSage, [YHC+18]. La predicción de enlaces ayuda a recomendar productos no observados que un usuario podría estar interesado en comprar. En la Figura 3.5 se muestra un ejemplo de una predicción en PinSage.

Figura 3.5.: Ejemplo de una predicción en PinSage. Hay ejemplos negativos que son más difíciles de distinguir que otros. Imagen obtenida de [YHC+18].

3.3.4. Detección de comunidades

Sea un grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, la detección de comunidades en ese grafo consiste en identificar subconjuntos de nodos que estén más densamente conectados entre sí que con el resto. La tarea es encontrar particiones de $\mathcal{V}$ en comunidades $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, donde los nodos dentro de cada comunidad C_i están altamente interconectados.

Un ejemplo de este tipo de tarea es la identificación de grupos de amigos en redes sociales, véase la Figura 3.6.

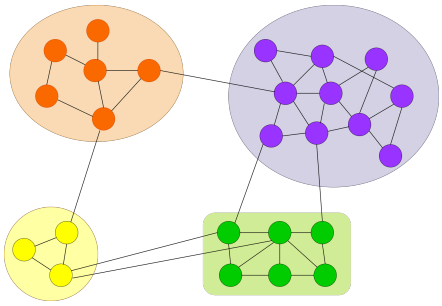


Figura 3.6.: Ejemplo de detección de comunidades en un grafo.

3.3.5. Embedding de grafos

El *embedding* de grafos busca mapear los nodos de un grafo a un espacio vectorial continuo de baja dimensión, manteniendo la estructura y propiedades del grafo. Formalmente, se trata de aprender una función $\phi : \mathcal{V} \to \mathbb{R}^d$, donde d es la dimensión del espacio de embedding.

3.3.6. Generación de grafos

Sea un conjunto de grafos $\mathcal{D} = \{\mathcal{G}_i\}$, la generación de grafos tiene como objetivo aprender una función $P(\mathcal{G})$ que sea capaz de crear nuevos grafos similares a los del conjunto de entrenamiento.

Un ejemplo de generación de grafos es la síntesis de nuevas moléculas en química computacional, donde cada grafo representa una estructura molecular. La generación de grafos

puede ayudar a descubrir nuevas moléculas con propiedades deseadas, como fármacos potenciales.

3.4. Formulación de problemas de clasificación y predicción

En esta sección se verá el proceso de aprendizaje y cómo sería un modelo de red neuronal para grafos para las tareas de clasificación de nodos, clasificación de grafos y predicción de enlaces, que son las de mayor interés para este trabajo.

3.4.1. Aprendizaje para clasificación de nodos

El objetivo de la clasificación de nodos es entrenar un modelo con los nodos etiquetados $\mathcal{V}_l$ para predecir las clases de los nodos sin etiquetar $\mathcal{V}_u$. Sea un modelo GNN con capas de filtrado, denotado por GNN_{node}, que tiene como entrada la estructura y las características de los nodos del grafo y como salida las características transformadas de los nodos:

$$\mathbf{F}^{(\text{out})} = GNN_{\text{node}}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta}_1),$$

donde $\mathbf{\Theta}_1$ representa los parámetros del modelo, $\mathbf{A} \in \mathbb{R}^{N \times N}$ es la matriz de adyacencia, $\mathbf{F} \in \mathbb{R}^{N \times d_i n}$ son las características de los nodos del grafo original y $\mathbf{F}^{(\text{out})} \in \mathbb{R}^{N \times d_{out}}$ son las características de salida. Estas características se utilizan para clasificar los nodos como sigue:

$$\mathbf{Z} = \text{softmax}(\mathbf{F}^{(\text{out})}\mathbf{\Theta}_2),$$

con $\mathbf{Z} \in \mathbb{R}^{N \times C}$, la fila i-ésima de $\mathbf{Z}$ indica la distribución de propabilidad de las clases predichas para el nodo v_i, seleccionándose normalmente la clase con mayor probabilidad; $\mathbf{\Theta}_2 \in \mathbb{R}^{d_{out} \times C}$ es la matriz de parámetros que transforma las características $\mathbf{F}^{(\text{out})}$ a la dimensión del número de clases C. El proceso completo se puede expresar como:

$$\mathbf{Z} = f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta}),$$

donde $\mathbf{\Theta}$ son todos los parámetros del modelo, que se pueden aprender minimizando la siguiente función durante el entrenamiento:

$$\mathcal{L}_{train} = \sum_{v_i \in \mathcal{V}_l} l(f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_i, y_i),$$

$f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta})_i$ denota la i-ésima fila de la salida, y_i es la clase asociada y $l(\cdot, \cdot)$ es la función de pérdida escogida.

3.4.2. Aprendizaje para clasificación de grafos

La clasificación de grafos busca aprender un modelo a partir del conjunto de entrenamiento $\mathcal{D} = \{\mathcal{G}_i, y_i\}$, donde y_i es la clase del grafo $\mathcal{G}_i$, que prediga la clase de grafos de los que se desconoce la clase. La red neuronal para grafos suele utilizarse como un codificador de características, que lleva un grafo de entrada a una representación de sus atributos:

$$\mathbf{f}_{\mathcal{G}} = GNN_{\text{graph}}(\mathcal{G}; \mathbf{\Theta}_1),$$

donde GNN_{graph} es el modelo que aprende representaciones a nivel de grafo, que consiste en capas de filtrado y de pooling; $\mathbf{f}_{\mathcal{G}} \in \mathbb{R}^{1\times d_{out}}$ es la representación resultado. Este resultado se utiliza a continuación para la clasificación del grafo:

$$\mathbf{z}_{\mathcal{G}} = \text{softmax}(\mathbf{f}_{\mathcal{G}}\mathbf{\Theta}_2),$$

donde $\mathbf{\Theta}_2 \in \mathbb{R}^{d_{out}\times C}$ es la matriz que transforma la representación a la dimensión del número de clases C y $\mathbf{z}_{\mathcal{G}} \in \mathbb{R}^{1\times C}$ denota las probabilidades predichas para cada clase del grafo. El proceso completo se puede expresar como:

$$\mathbf{z}_{\mathcal{G}} = f_{GNN}(\mathcal{G}; \mathbf{\Theta}),$$

con $\mathbf{\Theta}$ incluyendo los parámetros $\mathbf{\Theta}_1$ y $\mathbf{\Theta}_2$, que se aprenden minimizando la siguiente función objetivo:

$$\mathcal{L}_{train} = \sum_{\mathcal{G}_i\in\mathcal{D}} l(f_{GNN}(\mathcal{G}_i; \mathbf{\Theta}), y_i),$$

donde y_i es la clase asociada al grafo $\mathcal{G}_i$ y $l(\cdot,\cdot)$ es la función de pérdida.

3.4.3. Aprendizaje para predicción de enlaces

La predicción de enlaces tiene como objetivo entrenar un modelo para predecir la existencia de aristas entre pares de nodos en un grafo. Sea un modelo GNN con capas de filtrado, denotado por GNN_{link}, que toma como entrada la estructura del grafo y las características de los nodos y produce representaciones transformadas de los nodos:

$$\mathbf{H} = GNN_{\text{link}}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta}_1),$$

donde $\mathbf{\Theta}_1$ representa los parámetros del modelo, $\mathbf{A} \in \mathbb{R}^{N\times N}$ es la matriz de adyacencia, $\mathbf{F} \in \mathbb{R}^{N\times d_{in}}$ son las características de los nodos del grafo original y $\mathbf{H} \in \mathbb{R}^{N\times d_{out}}$ son las características de salida. Para predecir la existencia de un enlace entre dos nodos v_i y v_j, se combinan las características transformadas $\mathbf{h}_i$ y $\mathbf{h}_j$ de los nodos:

$$s_{ij} = \sigma(\mathbf{h}_i^\top \mathbf{W}\mathbf{h}_j),$$

donde $\sigma(\cdot)$ es la función sigmoide y $\mathbf{W} \in \mathbb{R}^{d_{out}\times d_{out}}$ es una matriz de parámetros aprendibles. El valor s_{ij} representa la probabilidad predicha de existencia de un enlace entre v_i y v_j. El proceso completo se puede expresar como:

$$\mathbf{S} = f_{GNN}(\mathbf{A}, \mathbf{F}; \mathbf{\Theta}),$$

donde $\mathbf{S} \in \mathbb{R}^{N\times N}$ es la matriz de probabilidades de enlaces y $\mathbf{\Theta}$ son todos los parámetros del modelo.

Para entrenar el modelo, se minimiza la siguiente función de pérdida basada en un conjunto de enlaces observados $\mathcal{E}_l$ y no observados $\mathcal{E}_u$:

$$\mathcal{L}_{train} = \sum_{(v_i,v_j)\in\mathcal{E}_l} l(s_{ij}, 1) + \sum_{(v_i,v_j)\in\mathcal{E}_u} l(s_{ij}, 0),$$

donde $l(\cdot,\cdot)$ es la función de pérdida, y 1 y 0 indican la existencia o ausencia de un enlace, respectivamente.

3.5. Arquitecturas de redes neuronales para grafos

En esta sección presentaremos las capas más relevantes de las redes neuronales para grafos que pueden considerarse estándar en la literatura, antes de pasar a la arquitectura principal del trabajo, la novedosa *QGCN*. El diseño y el estudio de las capas de GNN es una de las áreas más activas en el ámbito del aprendizaje profundo. Sin embargo, la gran mayoría de los trabajos se derivan de una capa conocida como paso de mensajes. Para el desarrollo de esta sección se ha seguido el libro [BBCV21] y los apuntes de la asignatura *Machine Learning with Graphs* de la universidad de Stanford [Sta24].

3.5.1. Paso de mensajes

Las redes neuronales de paso de mensajes (MPNNs) son un marco generalizado para el aprendizaje en grafos, presentado por Gilmer et al. en 2017 [GSR+17]. Las arquitecturas construidas con estas capas son equivariantes a permutaciones, esto es, que dos grafos isomorfos obtienen los mismos resultados. Estas estructuras tienen el siguiente esquema.

En primer lugar, las características del vecindario del nodo v son transformadas con una función $MSG(\cdot)$, conocida como mensaje. Normalmente, es una transformación afín con una función de activación:

$$\mathbf{m}_u^{(l)} = MSG^{(l)}(f_u^{(l-1)}) = \mathbf{W}^{(l)} f_u^{(l-1)} + \mathbf{b}^{(l)}, u \in \{\mathcal{N}(v) \cup v\}$$

donde $\mathbf{W}^{(l)}$ y $\mathbf{b}^{(l)}$ son parámetros que se pueden aprender, (l) indica el número de capa de la red, y f_u son las características del nodo u. Se añade el cálculo del mensaje del nodo v para que su información no se pierda. El siguiente paso es agregar estas características transformadas por medio de una operación como la suma, la media, el máximo o incluso una red neuronal:

$$AGG^{(l)}(\{\mathbf{m}_u^{(l)}, u \in \mathcal{N}(v)\}),$$

para no perder la información del nodo v, se puede concatenar o sumar ese mensaje:

$$CONCAT(AGG^{(l)}(\{\mathbf{m}_u^{(l)}, u \in \mathcal{N}(v)\}), \mathbf{m}_v^{(l)}).$$

Finalmente, se añade la no linealidad mediante una función de activación $\sigma(\dot{})$ como ReLU o sigmoide:

$$f_v^{(l)} = \sigma(CONCAT(AGG^{(l)}(\{\mathbf{m}_u^{(l)}, u \in \mathcal{N}(v)\}), \mathbf{m}_v^{(l)}).$$

Véase la Figura 3.7, que ilustra un ejemplo de grafo junto con dos capas de paso de mensajes tomando como nodo objetivo el nodo A.

3.5.2. Redes neuronales convolucionales para grafos

Las redes neuronales convolucionales para grafos (GCNs) surgieron como una extensión de las convoluciones tradicionales aplicadas a datos estructurados en grafos. Fueron introduci-

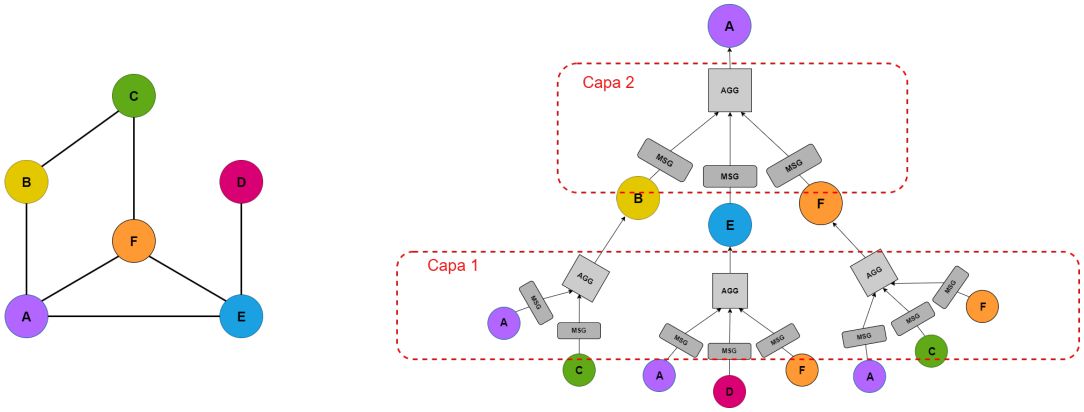


Figura 3.7.: Ejemplo de un grafo junto con dos capas de paso de mensajes tomando como nodo objetivo el nodo A.

das por primera vez por Kipf y Welling en 2016 [KW16].

La operación de convolución en los grafos se define como:

$$f_v^{(l)} = \sigma\Big(\sum_{u \in \mathcal{N}(v)} \mathbf{W}^{(l)} \frac{f_u^{(l-1)}}{\|\mathcal{N}(v)\|}\Big).$$

Por tanto, el mensaje sería:

$$\mathbf{m}_u^{(l)} = MSG^{(l)}(f_u^{(l-1)}) = \frac{1}{|\mathcal{N}(v)|}\mathbf{W}^{(l)} f_u^{(l-1)},$$

aunque en el paper original se utiliza una normalización un poco diferente. En este caso, la función de agregación usada es la suma de todos los mensajes de los vecinos. Los filtros espectrales vistos anteriormente en la Subsección 3.1.1 entran en esta categoría, ya que aplican operadores locales fijos (la Laplaciana) a las señales de los nodos.

3.5.3. Redes neuronales de atención para grafos

Las redes neuronales de atención se introdujeron en el año 2018 por Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò y Yoshua Bengio [VCC+18]. Entre los beneficios de esta arquitectura destacamos su eficiencia en memoria y computacional, esto último debido a que la operación es paralelizable a lo largo de las aristas y a que la agregación también se puede paralelizar sobre todos los nodos.

Vamos a describir la capa de atención en grafos. Sean las características de los nodos de entrada $\mathbf{f} = \{f_1, f_2, \ldots, f_{N_{in}}\}$, $\vec{f}_i \in \mathbb{R}^d_{in}$, donde N_{in} es el número de nodos de entrada, y d_{in} es el número de características de cada nodo. La capa tiene como salida un nuevo conjunto de características $\mathbf{f}' = \left\{f'_1, f'_2, \ldots, f'_{N_{op}}\right\}$, con $\vec{f}'_i \in \mathbb{R}^{d_{ou}}$.

Como paso inicial, se aplica a cada nodo una transformación lineal, parametrizada por

una matriz de pesos $\mathbf{W} \in \mathbb{R}^{d_{ou} \times d_{in}}$. Lo siguiente es calcular los coeficientes de atención $e_{ij} = a(\mathbf{W}\vec{h_i}, \mathbf{W}\vec{h_j})$, Figura 3.8, aplicando una función $a : \mathbb{R}^{d_{ou}} \times \mathbb{R}^{d_{ou}} \to \mathbb{R}$. Estos coeficientes indican la importancia de las características del nodo v_j en el nodo v_i, y la función a normalmente es una red neuronal. En la práctica, solamente se calculan los coeficientes de los nodos en un vecindario de radio k del nodo v_i, e_{ij} con $j \in \mathcal{N}^k(v_i)$. Consideramos a partir de ahora que $k = 1$. Para comparar los coeficientes entre distintos nodos, se normalizan haciendo uso de la función softmax:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{v_k \in \mathcal{N}(v_i)} \exp(e_{ik})}.$$

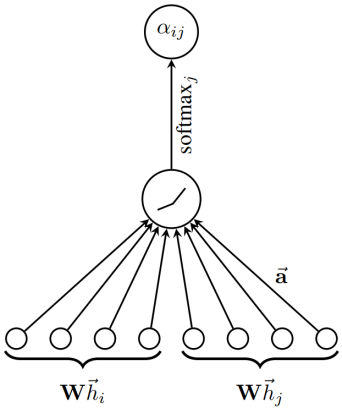


Figura 3.8.: Mecanismo de atención en grafos. Imagen obtenida de [VCC+18].

A continuación, los coeficientes de atención normalizados se utilizan para calcular una combinación lineal de las características junto la aplicación de no linealidad, obteniendo las características de cada nodo de salida.

Finalmente, para estabilizar el proceso de aprendizaje, se puede aplicar un mecanismo conocido como atención múltiple. Esto consiste en ejecutar R mecanismos de atención independientes y concatenar las características de salida. Aunque si se aplica este proceso en la última capa de la red, correspondiente a la predicción, se emplearía una media en lugar de la concatenación:

$$f_i' = \Big\|_{r=1}^{R} \sigma\left(\sum_{v_j \in \mathcal{N}(v_i)} \alpha_{ij}^r \mathbf{W}^r f_j\right),$$

donde $\|$ denota la concatenación, α_{ij}^r son los coeficientes de atención normalizados calculados por el mecanismo de atención r-ésimo (a^r), y $\mathbf{W}^r$ es la correspondiente matriz de pesos de la transformación lineal de entrada, véase la Figura 3.9.

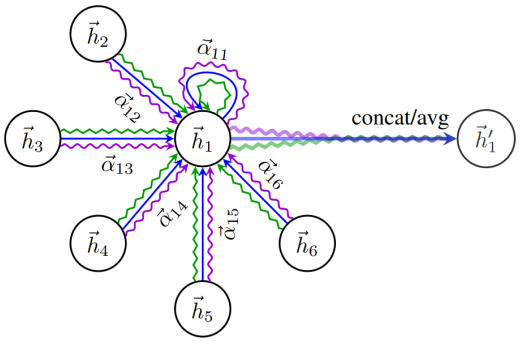


Figura 3.9.: Mecanismo de atención múltiple con $R = 3$. Imagen obtenida de [VCC+18].

3.6. Arquitectura $\mathcal{Q}$GCN

En esta sección se introduce la arquitectura principal de estudio de este trabajo: una red neuronal para grafos en variedades pseudo-riemannianas. Comenzaremos definiendo las operaciones básicas necesarias para el correcto funcionamiento del modelo. Seguidamente, se explicará esta arquitectura, y finalmente el optimizador implementado. El código de este modelo es del artículo [XZP+22a] y el del optimizador pseudo-riemmaniano de [LS21].

3.6.1. Operaciones básicas

Antes de explicar esta arquitectura, debemos introducir tres operaciones fundamentales que se encontrarán en la mayoría de capas de la red:

- *proj_tan*: Proyección de un vector z sobre el espacio tangente en un punto de la variedad. Este punto normalmente será el origen.

```python
def proj_tan(self, z, x, beta, time_dim=None):
    # Establece el valor de time_dim si no se proporciona
    time_dim = self.time_dim if time_dim is None else time_dim

    # Calcula el producto interno entre z y x en la metrica de la
        variedad
    inner_zx = self.inner(z, x, time_dim=time_dim)

    # Calcula el producto interno de x consigo mismo
    inner_xx = self.inner(x, x, time_dim=time_dim)

    # Proyecta z sobre el espacio tangente en x
    res = z - (inner_zx / inner_xx).unsqueeze(1) * x
    return res
```

- *expmap*: Aplicación exponencial, que permite proyectar vectores del espacio tangente en un punto (que en general será el origen) sobre la variedad pseudo-riemanniana. La definición de esta función para el caso de los pseudo-hiperboloides se ha realizado en la Sección 6.3, y dependerá del tipo de vector (espacial, temporal o luminoso).

```python
def expmap(self, x, v, beta, time_dim=None):
    time_dim = self.time_dim if time_dim is None else time_dim
```

```
    epsilon = 0.000001
    n = v.shape[0]
    d = v.shape[1]
    inner = self.inner(v, v, time_dim=time_dim)
    norm_product = torch.clamp(inner.abs(), min=self.min_norm).sqrt()
    norm_product = torch.clamp(norm_product, max=50).view(norm_product.
        size(0), -1)

    space_like = inner < -epsilon
    time_like = inner > epsilon
    null_geodesic = (~space_like) & (~time_like)
    other = (~time_like) & (~space_like) & (~null_geodesic)
    U = v.clone()
    abs_beta = 1 / (abs(beta) ** 0.5)

    if True in time_like:
        beta_product = torch.clamp(abs_beta * norm_product[time_like],
            max=self.max_norm)
        U[time_like, :] = x[time_like, :] * torch.clamp(torch.cosh(
            beta_product), max=self.max_norm) + torch.clamp(v[time_like,
            :] * torch.sinh(beta_product) / beta_product, max=self.
            max_norm)
        assert not torch.isnan(U[time_like, :]).any()

    if True in space_like:
        beta_product = torch.clamp(abs_beta * norm_product[space_like],
            max=self.max_norm)
        U[space_like, :] = x[space_like, :] * torch.clamp(torch.cos(
            beta_product), max=self.max_norm) + torch.clamp(v[space_like,
             :] * torch.sin(beta_product) / beta_product, max=self.
            max_norm)
        assert not torch.isnan(U[space_like, :]).any()

    if True in null_geodesic:
        U[null_geodesic, :] = torch.clamp(x[null_geodesic, :] + v[
            null_geodesic, :], max=self.max_norm)
        assert not torch.isnan(v[null_geodesic, :]).any()

    assert not torch.isnan(U).any()
    return self.proj(U, beta)
```

- *logmap*: Aplicación inversa de la exponencial, que permite proyectar los vectores de la variedad sobre el espacio tangente en un punto. La definición matemática de esta función en el caso de los pseudo-hiperboloides se encuentra en la Sección 6.3.

3.6.2. Capas

Una vez definidas estas operaciones, veamos cómo es la arquitectura. La arquitectura principal estudiada en este trabajo es una **red pseudo-hiperbólica neuronal convolucional para grafos** o *Pseudo-hyperbolic Graph Convolutional Network* ($\mathcal{Q}$GCN). Esta red está diseñada para operar en una variedad pseudo-riemanniana (la definición se encuentra en el Capítulo 5), aprovechando las ventajas que ofrecen estos espacios mientras soluciona los problemas que surgen. A continuación, describimos los componentes principales de la arquitectura.

La capa base que compone este modelo es *HyperbolicGraphConvolution*, que implementa la convolución de grafos en un espacio hiperbólico, combinando las siguientes capas:

- *HypLinear*: Esta capa realiza la multiplicación matricial en el espacio hiperbólico haciendo uso de las aplicaciones exponencial y logarítmica, y proyecta los resultados de vuelta a la variedad. Además, se puede añadir un valor de sesgo al final.
- *HypAgg*: Esta capa se encarga de la agregación de las características de los nodos vecinos en el espacio tangente mediante la suma ponderada. Esta capa finaliza proyectando de nuevo los datos a la variedad haciendo uso de la función exponencial.
- *HypAct*: Aplica una función de activación RELU en el espacio tangente y proyecta los resultados de vuelta a la variedad hiperbólica.

Las capas lineal y de agregación utilizan Dropout para reducir el sobreajuste. La estructura seguida por este modelo es una estructura codificador-decodificador, la Figura 3.10 muestra un diagrama de la arquitectura:

- **Codificación:** Las características de entrada se proyectan al espacio tangente, para luego proyectar sobre la variedad de estudio y pasar estos datos a través de las capas convolucionales.
- **Decodificación:** En el caso de clasificación de nodos, se utiliza un decodificador lineal, que consiste en una red neuronal feedforward que proyecta las representaciones de los nodos al espacio tangente, aplica una transformación lineal y genera las probabilidades para la clasificación. En cambio, para la predicción de enlaces, se utiliza un decodificador Fermi-Dirac, que toma las distancias cuadradas entre las representaciones embebidas de los nodos y calcula la probabilidad de existencia de un enlace utilizando la función de distribución Fermi-Dirac [KPK+10].

3.6.3. Procesamiento

En esta sección vamos a entrar más en detalle en el proceso completo de la arquitectura:

- **Inicialización de características:** En primer lugar, las características de entrada del espacio euclídeo se llevan al pseudo-hiperboloide. Esto se hará llevando las características a las variedades difeomorfas $\mathcal{S}^{q-1} \times \mathbb{R}^{d-q}$ a través de la función $\psi(\cdot)$, y luego llevarlas al pseudo-hiperboloide $\mathcal{Q}^q_\beta$ con $\psi^{-1}(\cdot)$. Estas funciones están definidas en la Sección 9.1.
- **Agregación tangencial:** La combinación lineal de las características de los vecinos se lleva al espacio tangente. En este caso, el modelo agrega los embeddings de los vecinos en el espacio tangente del punto de referencia $\mathbf{o}$, después se aplica una función de activación y se proyecta la representación actualizada a la variedad. Formalmente, en cada capa l, se actualizan las características como sigue:

$$\mathbf{h}_i^{l+1} = \widehat{\exp}_{\mathbf{o}}^{\beta_{l+1}}(\sigma(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \widehat{\log}_{\mathbf{o}}^{\beta_l}(\mathbf{W}^l \otimes^{\beta_l} \mathbf{h}_j^l \oplus^{\beta_l} \mathbf{b}^l))),$$

donde $\sigma(\cdot)$ es la función de activación, β_l y β_{l+1} son dos curvaturas, $\mathcal{N}(i)$ son los vecinos de un salto del nodo i, y $\otimes$, $\oplus$ denotan la transformación tangencial y la traslación del sesgo, respectivamente. La función $\widehat{\exp}_{\mathbf{o}}^{\beta_{l+1}}$ denota la función *expmap* y $\widehat{\log}_{\mathbf{o}}^{\beta_l}$ es *logmap*.

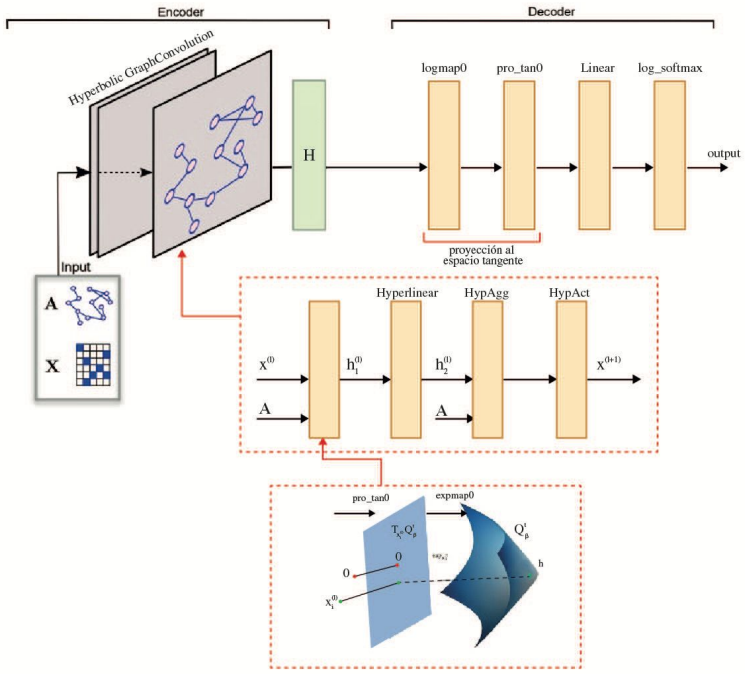


Figura 3.10.: Diagrama de la arquitectura principal para clasificación de nodos. Imagen modificada de [CYRL19].

- **Transformación tangencial:** Primero se proyectan las características al espacio tangente del polo sur $\mathbf{o} = (|\beta|, 0, \ldots, 0)$ mediante la función *logmap* y se lleva a cabo la multiplicación de matrices. Por último, las características transformadas se llevan a la variedad mediante la función *expmap*. Formalmente, en cada capa l, el proceso se describe:

$$\mathbf{W}^l \otimes^\beta \mathbf{h}^l = \widehat{\exp}^\beta_{\mathbf{o}}(\mathbf{W}^l \widehat{\log}_{\mathbf{o}^\beta}(\mathbf{h}^l)),$$

  donde $\mathbf{W}^l$ es la matriz de parámetros en el espacio euclídeo.

- **Traslación del sesgo:** Para evitar que el modelo colapse, se introduce la traslación del sesgo tras la transformación tangencial. Esto consiste en transportar paralelamente un vector tangente $\mathbf{b}^l \in T_{\mathbf{o}}\mathcal{Q}^q_\beta$ al espacio tangente del punto de interés. Finalmente, el vector tangente transportado se lleva de vuelta a la variedad con la función *expmap*. Formalmente, la traslación de sesgo viene dada por:

$$\tilde{\mathbf{h}}^l \oplus^\beta \mathbf{b}^l = \begin{cases} \exp^\beta_{\tilde{\mathbf{h}}^l}(P^\beta_{\mathbf{o}\to\tilde{\mathbf{h}}^l}(\mathbf{b}^l)), & \text{si } \langle \mathbf{o}, \tilde{\mathbf{h}}^l \rangle_q < |\beta| \\ -\exp^\beta_{-\tilde{\mathbf{h}}^l}(P^\beta_{\mathbf{o}\to-\tilde{\mathbf{h}}^l}(\mathbf{b}^l)), & \text{si } \langle \mathbf{o}, \tilde{\mathbf{h}}^l \rangle_q \geq |\beta| \end{cases}$$

donde $P^{\beta}_{\mathbf{o}\to\tilde{\mathbf{h}}^l}$ denota el transporte paralelo. El transporte paralelo es un método que permite mover vectores tangentes a lo largo de una curva en la variedad, manteniendo constante el ángulo entre los vectores y la curva en cada punto. Esto asegura que las características del modelo se trasladan coherentemente. Para los casos que se cumple $\langle \mathbf{o}, \tilde{\mathbf{h}}^l \rangle_q \geq |\beta|$, $P^{\beta}_{\mathbf{o}\to\tilde{\mathbf{h}}^l}$ no está definido, por lo que se hace es transportar paralelamente $\mathbf{b}^l$ al espacio tangente del punto antípoda $-\tilde{\mathbf{h}}^l$, y luego aplicar $\exp^{\beta}_{-\tilde{\mathbf{h}}^l}$ para llevarlo de vuelta a la variedad.

3.6.4. Optimizador

El optimizador utilizado en el entrenamiento de esta red, *Riemannian Adam*, es una extensión del optimizador Adam adaptado para trabajar en el contexto de geometrías riemannianas y pseudo-riemannianas. Este optimizador no solo aprovecha las ventajas de Adam, como el uso de medias móviles exponenciales de los gradientes y sus cuadrados para mejorar la eficiencia y estabilidad, sino que también incorpora técnicas específicas para mantener los parámetros dentro de la variedad durante el proceso de optimización.

La definición matemática de una simplificación de este optimizador se encuentra en la Subsección 9.2.4, donde se detalla el método de optimización pseudo-riemanniana. El código fuente del optimizador se ha sacado del repositorio proporcionado de [CYRL19].

El funcionamiento principal de Riemannian Adam es como sigue:

- **Inicialización del estado**: Se inicializan el promedio exponencial de los gradientes (*exp_avg*) y el promedio exponencial de los cuadrados de los gradientes (*exp_avg_sq*) a vectores nulos.
- **Paso de Optimización**: En cada paso, se realiza lo siguiente:
  - Se calcula el gradiente en la variedad utilizando la función *egrad2rgrad*.
  - Se actualizan los promedios exponenciales de los gradientes y sus cuadrados.
  - Se calcula el tamaño del paso (*step_size*) ajustado por las correcciones de sesgo.
  - Se actualiza el punto en la variedad utilizando la función *expmap* y se proyecta de vuelta a la variedad con *proj*.
  - Se transporta el promedio exponencial de los gradientes al nuevo punto utilizando *ptransp*, que realiza el transporte paralelo del vector de gradiente desde el espacio tangente en el punto actual al nuevo punto en la variedad.
- **Estabilización**: Cada ciertos pasos, se ejecuta la función *stabilize*, que asegura que los parámetros se mantengan en la variedad correcta.

A continuación, detallamos el código del optimizador:

```
class RiemannianAdam(OptimMixin, torch.optim.Adam):
    def step(self, closure=None):
        loss = None
        if closure is not None:
            loss = closure()
        with torch.no_grad():
```

```
            for group in self.param_groups:
                if "step" not in group:
                    group["step"] = 0
                betas = group["betas"]
                weight_decay = group["weight_decay"]
                eps = group["eps"]
                learning_rate = group["lr"]
                for point in group["params"]:
                    grad = point.grad
                    if grad is None:
                        continue
                    if isinstance(point, (ManifoldParameter)):
                        manifold = point.manifold
                        c = point.c
                    else:
                        manifold = _default_manifold
                        c = None

                    state = self.state[point]

                    # State initialization
                    if len(state) == 0:
                        state["step"] = 0
                        # Exponential moving average of gradient values
                        state["exp_avg"] = torch.zeros_like(point)
                        # Exponential moving average of squared gradient
                            values
                        state["exp_avg_sq"] = torch.zeros_like(point)

                    # make local variables for easy access
                    exp_avg = state["exp_avg"]
                    exp_avg_sq = state["exp_avg_sq"]
                    # actual step
                    grad.add_(weight_decay, point)
                    grad = manifold.egrad2rgrad(point, grad, c)
                    exp_avg.mul_(betas[0]).add_(1 - betas[0], grad)
                    exp_avg_sq.mul_(betas[1]).add_(
                            1 - betas[1], manifold.inner(point, point, c, grad
                                , keepdim=True)
                    )

                    denom = exp_avg_sq.sqrt().add_(eps)

                    group["step"] += 1
                    bias_correction1 = 1 - betas[0] ** group["step"]
                    bias_correction2 = 1 - betas[1] ** group["step"]
                    step_size = (
                        learning_rate * bias_correction2 ** 0.5 /
                            bias_correction1
                    )

                    # copy the state, we need it for retraction
                    # get the direction for ascend
                    direction = exp_avg / denom
                    # transport the exponential averaging to the new point
                    new_point = manifold.proj(manifold.expmap(-step_size *
                        direction, point, c), c)
                    exp_avg_new = manifold.ptransp(point, new_point, exp_avg,
                        c)
                    # use copy only for user facing point
```

```
                    copy_or_set_(point, new_point)
                    exp_avg.set_(exp_avg_new)

                    group["step"] += 1
                if self._stabilize is not None and group["step"] % self.
                    _stabilize == 0:
                    self.stabilize_group(group)
        return loss

    @torch.no_grad()
    def stabilize_group(self, group):
        for p in group["params"]:
            if not isinstance(p, ManifoldParameter):
                continue
            state = self.state[p]
            if not state:
                continue
            manifold = p.manifold
            c = p.c
            exp_avg = state["exp_avg"]
            p.copy_(manifold.proj(p, c))
            exp_avg.set_(manifold.proj_tan(exp_avg, u, c))
```

Parte II.

Fundamentos matemáticos

4. Conceptos previos

En este capítulo nos enfocaremos en establecer una base sólida de conceptos preliminares para una comprensión profunda del marco matemático que sostiene este trabajo. Primero, vamos a recordar los conceptos de formas bilineales, introduciremos los distintos tipos de vectores, así como la nomenclatura utilizada. Además, definiremos las variedades regulares y diferenciables, dando ejemplos en cada caso. Los resultados expuestos en esta sección se han estudiado de las asignaturas cursadas a lo largo de mi formación en el grado y también se ha usado el libro [VC10].

4.1. Formas bilineales

Definición 4.1. Una **forma bilineal** en un espacio vectorial V es una aplicación $g : V \times V \to \mathbb{R}$, con las siguientes propiedades: para todo $u, v, w \in V$, y $\lambda \in \mathbb{R}$, tenemos,

- $g(u + v, w) = g(u, w) + g(v, w)$
- $g(u, v + w) = g(u, v) + g(u, w)$
- $g(\lambda u, v) = \lambda g(u, v)$
- $g(u, \lambda v) = \lambda g(u, v)$

Si además se satisface la propiedad $g(u, v) = g(v, u)$ para todo $u, v \in V$, esto es, que sea una forma bilineal simétrica, se dice que g es una **métrica**.

Pongamos algunos ejemplos de formas bilineales:

1. La aplicación bilineal cero, $g = 0$, es decir, $g(u, v) = 0$ para todo $u, v \in V$.
2. La métrica euclídea en $\mathbb{R}^n$:
$$g_E(x, y) = \sum_{i=1}^{n} x_i y_i$$
3. La métrica de Lorentz-Minkowski en $\mathbb{R}^n$:
$$g_L(x, y) = -x_1 y_1 + \sum_{i=2}^{n} x_i y_i$$

Definición 4.2. Sea $g : V \times V \to \mathbb{R}$ una métrica. Diremos que g es:

- **definida positiva** (resp. **definida negativa**) si $g(v, v) > 0$ (resp. $g(v, v) < 0$) para todo $v \in V \setminus \{0\}$,
- **semidefinida positiva** (resp. **semidefinida negativa**) si $g(v, v) \geq 0$ (resp. $g(v, v) \leq 0$) para todo $v \in V \setminus \{0\}$,

- **indefinida** si no es ni semidefinida positiva ni semidefinida negativa,
- **no degenerada** si la condición $g(v,w) = 0$ para todo $w \in V$ implica que $v = 0$. En otro caso, diremos que g es **degenerada**.

Definición 4.3. Un **producto escalar** g en V es una métrica no degenerada.

Definición 4.4. Un **espacio métrico** es un par (V, g) donde V es un espacio vectorial y g es una métrica definida en V.

Definición 4.5. Sea (V, g) con V espacio vectorial y g una métrica, diremos que $v \in V$ es:

- **temporal** si $g(v,v) < 0$,
- **espacial** si $g(v,v) > 0$,
- **luminoso** si $g(v,v) = 0$ y $v \neq 0$,
- **causal** si v es temporal o luminoso.

Definición 4.6. Si (V, g) es un espacio métrico, se llama **cono de luz** al conjunto

$$C(g) = \{v \in V : g(v,v) = 0\}.$$

Por tanto, el cono de luz es el conjunto formado por los vectores autoconjugados.

Ejemplo 4.1. En $(\mathbb{R}^2, g_L)$ con g_L la métrica de Minkowski,

$$C(g) = \{(x,x) : x \in \mathbb{R}\} \cup \{(x,-x) : x \in \mathbb{R}\}.$$

El cono de luz no es necesariamente un subespacio vectorial.

A continuación, presentamos dos teoremas sobre bases de espacios métricos que nos serán de utilidad para demostrar que el número de $1, -1$ y 0 en la diagonal principal de cualquier expresión matricial de g respecto a una base ortonormal no depende de la base escogida.

Teorema 4.1. *Sea (V, g) un espacio métrico. Entonces existe una base donde la métrica diagonaliza.*

Teorema 4.2. *Sea un espacio métrico (V, g). Si B es una base tal que la métrica diagonaliza con B, entonces el número de 0 en la diagonal principal coincide con $dim(V) - rango(\mathcal{M}_B(g))$, siendo B una base de V*

Definición 4.7. Sea (V, g) un espacio vectorial con g un producto escalar. Si $v, w \in V$, diremos que v es **ortogonal** a w, $v \perp w$, si $g(v,w) = 0$. Consecuentemente, para $A, B \subseteq V$, diremos que A es **ortogonal** a B, $A \perp B$, si $v \perp w$ para cada $v \in A$ y cada $w \in B$. Denotaremos

$$A^\perp = \{w \in V : g(v,w) = 0, \forall v \in A\}, \text{ al } \textbf{espacio ortogonal} \text{ a A}$$

Lema 4.1. *Un subespacio W de un espacio vectorial V con producto escalar es no degenerado si y solo si*

$$V = W \oplus W^\perp$$

Demostración. Sabemos que $\dim(W + W^\perp) + \dim(W \cap W^\perp) = \dim(W) + \dim(W^\perp) = \dim(V)$. Por lo que el lema se cumplirá si y solo si $\dim(W \cap W^\perp) = 0 \iff \{0\} = W \cap W^\perp = \{w \in W : w \perp W\}$, que equivale a que W sea no degenerado. □

Definición 4.8. Una **base ortonormal** B en un espacio métrico es una base donde $\mathcal{M}_B(g)$ es una matriz diagonal y los elementos de la diagonal principal son 1, -1 o 0.

O dicho de otra manera, una base tal que $g(e_i, e_i)$ es 1, -1 o 0, y $g(e_i, e_j) = 0$ si $i \neq j$.

Teorema 4.3. ***(Sylvester)*** *En todo espacio vectorial métrico existen bases ortonormales. Además, el número de 1, -1 y 0 de la diagonal principal de cualquier expresión matricial de g respecto de una base ortonormal, es independiente de la base escogida en (V, g).*

Corolario 4.1. *Toda base ortonormal de un subespacio no degenerado U de V se puede extender a una base ortonormal de (V, g).*

Definición 4.9. Sea (V, g) un espacio métrico. Se llama:

1. **signatura** de g a $\sigma(g) = (k, m)$, donde k y m es el número de 1 y -1, respectivamente, obtenidos de una base ortonormal.
2. **índice** de g al número de -1.
3. **nulidad** de g al número de 0.

Por tanto:

1. $\text{rango}(g) = k + m$.
2. $\text{nulidad}(g) + \text{rango}(g) = \text{nulidad}(g) + k + m = \dim(V)$.

Corolario 4.2. *Sea (V, g) un espacio métrico de dimensión n y sea $\sigma(g) = (k, m)$. Tenemos las siguientes equivalencias:*

- *La métrica es definida positiva (resp. def. negativa) si, y sólo si, $\sigma(g) = (n, 0)$ (resp. $\sigma(g) = (0, n)$).*
- *La métrica es semidefinida positiva (resp. semidefinida negativa) si, y sólo si, $\sigma(g) = (k, 0)$ con $0 < k < n$ (resp. $\sigma(g) = (0, m)$ con $0 < m < n$).*
- *La métrica es no degenerada si, y sólo si, $k + m = n$.*
- *La métrica es degenerada si, y sólo si, $k + m < n$.*

Observación 4.1. Una métrica g en V es no degenerada, si y solo si, $V^\perp = \{0\}$. Un subespacio $W \subset V$ es no degenerado si $g|_W$ es no degenerada.

Si g es definida positiva no degenerada, entonces cualquier subespacio W será no degenerado. Sin embargo, si g es indefinida, existen subespacios degenerados, por ejemplo $W = L\{w\}$ para cualquier vector w luminoso, donde L indica espacio generado. Para una métrica indefinida, siempre existen vectores luminosos.

Sea V un espacio vectorial de dimensión d y g una métrica con signatura $\sigma(g) = (k, m)$, con $k + m = d$ y $k > 0,\ m > 0$. Entonces podemos expresar la métrica respecto a una base ortonormal tal que $g(x, y) = -\sum_{i=1}^{m} x_i y_i + \sum_{i=m+1}^{k+m} x_i y_i$. El vector $v = (v_1, v_2, \ldots, v_d)$, con $v_1 = v_2 = \ldots = v_m = a$, $v_{q+1} = \ldots = v_d = \sqrt{\frac{m}{k}}a$, y a una constante distante de cero, v es luminoso ya que $g(v, v) = -\sum_{i=1}^{m} a^2 + \sum_{i=m+1}^{k+m} \frac{m}{k}a^2 = -ma^2 + k\frac{m}{k}a^2 = 0$.

Por tanto, un subespacio W de un espacio vectorial con producto escalar no es en general un espacio vectorial con producto escalar.

Corolario 4.3. *W es no degenerado si y solo si $W^\perp$ es no degenerado.*

4.2. Variedades regulares

Definición 4.10. Una **variedad regular** de dimensión n en $\mathbb{R}^k, k \geq n$, es un subespacio topológico $M \subseteq \mathbb{R}^k$ tal que para todo $p \in M$ existen abiertos $U \subseteq \mathbb{R}^n$ y $p \in V \subseteq \mathbb{R}^k$ y una aplicación $X : U \to \mathbb{R}^k$ diferenciable tal que:

- $X : U \to V \cap M$ es un homeomorfismo.
- La diferencial $dX_q : U \to \mathbb{R}^k$ es inyectiva para todo $q \in U$. A la aplicación $X : U \to V \cap M \subset \mathbb{R}^k$ se le llama carta o parametrización en M alrededor del punto $p \in M$.

Proposición 4.1. *Sean $W \subseteq \mathbb{R}^k$ un abierto, $F : W \to \mathbb{R}^n$ una aplicación diferenciable y $p \in W$ un punto. Si $dF_p : \mathbb{R}^k \to \mathbb{R}^n$ es sobreyectiva $(k \geq n)$ entonces existen abiertos $p \in O \subset W, V \subseteq \mathbb{R}^k$ y un difeomorfismo $\phi : V \to O$ tales que $F \circ \phi = \pi|_V$, donde π es la proyección*

$$\pi : \mathbb{R}^{k-n} \times \mathbb{R}^n \equiv \mathbb{R}^k \to \mathbb{R}^n, \quad \pi(x, y) = y.$$

Como consecuencia, $F^{-1}(a) \subset W$ es una variedad regular de $\mathbb{R}^k$ con dimensión $k - n$ para todo valor regular $a \in F(W)$ de F.

A continuación, presentamos algunos ejemplos de variedades regulares:

Ejemplo 4.2. La esfera en $\mathbb{R}^{n+1}$ de centro q y radio r, con $q = (a_1, \ldots, a_{n+1}) \in \mathbb{R}^{n+1}$ y $r > 0$

$$\mathbb{S}^n(q, r) := \{p \in \mathbb{R}^{n+1} : \|x - q\| = r\}, \text{donde } \|\cdot\| \text{es la norma euclídea,}$$

es una variedad regular de dimensión n de $\mathbb{R}^{n+1}$, ya que $0 \in \mathbb{R}$ es un valor regular para la aplicación diferenciable

$$F : \mathbb{R}^{n+1} \to \mathbb{R}, \quad F(x_1, \ldots, x_n) = \sum_{j=1}^{n+1} (x_j - a_j)^2 - r^2,$$

y aplicamos la proposición Proposición 4.1.

Ejemplo 4.3. Toros n-dimensionales. Sean $\xi_1, \ldots, \xi_n \in \mathbb{C}$ y $r_1, \ldots, r_n > 0$. El toro n-dimensional

$$\mathbb{T}^n = \{(z_1, \ldots, z_n) \in \mathbb{C}^n = \mathbb{R}^{2n} : |z_j - \xi_j|^2 = r_j^2,\ j = 1, \ldots, n\}$$

es una variedad regular de dimensión n de $\mathbb{R}^{2n}$. En este caso $(0, \ldots, 0) \in \mathbb{R}^n$ es un valor regular para la aplicación diferenciable

$$F : \mathbb{C}^n = \mathbb{R}^{2n} \to \mathbb{R}^n, \quad F(z_1, \ldots, z_n) = (|z_1 - \xi_1|^2 - r_1^2, \ldots, |z_n - \xi_n|^2 - r_n^2).$$

Y aplicamos la proposición Proposición 4.1 como antes.

Pasamos ahora a definir el espacio tangente de una variedad regular.

Definición 4.11. El **espacio tangente** a una variedad regular M^n de $\mathbb{R}^k$ en un punto $p \in M,\ n \leq k$, denotado por T_pM, es el subespacio vectorial n-dimensional de $\mathbb{R}^k$ dado por

$$T_pM = \left\{ \left.\frac{d}{dt}\right|_{t_0} \alpha(t) :\ \alpha \colon I \subseteq \mathbb{R} \to M, \text{ una curva diferenciable con } t_0 \in I \text{ y } \alpha(t_0) = p \right\},$$

donde $I \subseteq \mathbb{R}$ indica un intervalo abierto. Si $X : U \to \mathbb{R}^k$ es una parametrización de M con $p = X(q) \in X(U)$ para algún $q \in U$, entonces el espacio tangente de M en p, se define como

$$T_pM = L\left\{ \left.\frac{\partial X}{\partial u_j}\right|_q, j = 1, \ldots, n \right\} = dX_q(\mathbb{R}^n) \subset \mathbb{R}^k,$$

donde L indica el espacio generado por los vectores $\left.\frac{\partial X}{\partial u_j}\right|_q$.

4.3. Variedades diferenciables

En esta sección, daremos el concepto abstracto de variedad diferenciable y comentaremos cómo generalizar algunas de las definiciones dadas para variedades regulares.

La siguiente proposición nos permite formular el concepto de diferenciabilidad de funciones sobre variedades regulares.

Proposición 4.2. *Sea $M^n \subseteq \mathbb{R}^k, n \leq k$, una variedad regular y $X_j : U_j \subset \mathbb{R}^n \to \mathbb{R}^k$ parametrización de M, $j = 1, 2$. Si $O = X_1(U_1) \cap X_2(U_2) \neq$ entonces*

$$X_2^{-1} \circ X_1 : X_1^{-1}(O) \to X_2^{-1}(O)$$

es un difeomorfismo (que llamaremos cambio de parámetros).

Definición 4.12. Dada $M^n \subseteq \mathbb{R}^n, n \leq k$, variedad regular, una función $f : M \to \mathbb{R}$ se dice **diferenciable** en $p \in M$ si existe $X : U \subset \mathbb{R}^n \to \mathbb{R}^k$ parametrización de M con $p \in X(U)$ tal que $f \circ X : U \to \mathbb{R}$ es diferenciable.

A partir de esta definición, podemos definir objetos más abstractos que las variedades regulares sobre los que dar sentido al concepto de diferenciabilidad.

Definición 4.13. Se llama **variedad diferenciable** de dimensión n a un conjunto M^n junto con una familia $\mathcal{D} = \{(U_i, \phi_i), i \in I\}$ verificando:

1. $\{U_i\}_{i \in I}$ es un recubrimiento abierto de M.
2. Existe $n \in \mathbb{N}$ tal que para todo $i \in I$ la aplicación ϕ_i es un homeomorfismo $U_i \to O_i$, donde $O_i \subseteq \mathbb{R}^n$ es un abierto.
3. Si $U_{ij} := U_i \cap U_j \neq \varnothing$, la aplicación $\phi_j \circ \phi_i^{-1} : \phi_i(U_{ij}) \to \phi_j(U_{ij})$ es un difeomorfismo.
4. La familia $\mathcal{D}$ es maximal con respecto a las condiciones anteriores.

A un par (U, ϕ) se le llama **entorno coordenado** o **carta** de la variedad y a un recubrimiento (no necesariamente maximal) $\{(U_i, \phi_i), i \in I\}$ de M por cartas se le llama **atlas** de M.

Ejemplo 4.4. La esfera $\mathbb{S}^n \equiv \mathbb{S}^n(0, 1) \subset \mathbb{R}^{n+1}$. Un atlas sobre $\mathbb{S}^n$ compatible con la estructura diferenciable canónica viene dado por $\mathcal{A} = \{(U_i^{\pm}, \phi_i^{\pm}) : i = 1, \ldots, n+1\}$, donde $U_i^{\pm} := \{(x_1, \ldots, x_{n+1}) \in \mathbb{S}^n : \pm x_i > 0\}$ y

$$\phi_i^{\pm} : U_i^{\pm} \to \mathbb{B}^n = \{x \in \mathbb{R}^n : \|x\| < 1\}, \;\; \phi_i^{\pm}(x_1, \ldots, x_{n+1}) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n+1}).$$

$U_i^{\pm} \subset \mathbb{S}^2$ es abierto en $\mathbb{S}^n$ para todo $i = 1, \ldots, n+1$,

$$\mathbb{S}^n = (\cup_{i=1}^{n+1} U_i^+) \cup (\cup_{i=1}^{n+1} U_i^-)$$

y $\phi_i^{\pm} : U_i^{\pm} \to \mathbb{B}^n$ es un homeomorfismo con inversa $(\phi_i^{\pm})^{-1} : \mathbb{B}^n \to U_i^{\pm}$,

$$(\phi_i^{\pm})^{-1}(y_1, \ldots, y_n) = (y_1, \ldots, y_{i-1}, \pm\sqrt{1 - \sum_{j=1}^{n} y_j^2},\ y_{i+1}, \ldots, y_n) \text{ para todo } i = 1, \ldots, n+1.$$

Los cambios de carta son diferenciables (funciones polinómicas y radicales).

Ejemplo 4.5. El espacio proyectivo real n-dimensional $\mathbb{RP}^n$. La topología de $\mathbb{RP}^n$ es la cociente para la proyección $\Pi : \mathbb{R}^{n+1} \setminus \{0\} \to \mathbb{RP}^n$. Para cada $i \in \{1, \ldots, n+1\}$ el conjunto

$$O_i = \{(x_1, \ldots, x_{n+1} \in \mathbb{R}^{n+1} : x_i \neq 0\} \text{ es } \Pi\text{-saturado.}$$

Por tanto, $U_i = \Pi(O_i)$ es abierto en $\mathbb{RP}^n$, con $\cup_{i=1}^{n} U_i = \mathbb{RP}^n$. La aplicación

$$\phi : U_i \to \mathbb{R}^n, \ \ \phi(\Pi(x_1, \ldots, x_{n+1})) = (\frac{x_1}{x_i}, \ldots, \frac{x_{i-1}}{x_i}, \frac{x_{i+1}}{x_i}, \ldots, \frac{x_{n+1}}{x_i}),$$

está bien definida, es continua, y su inversa

$$\phi_i^{-1} : \mathbb{R}^n \to U_i, \ \ \phi_i^{-1}(x_1, \ldots, x_n) = \Pi(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

es también continua, por lo que ϕ_i es un homeomorfismo para todo $i = 1, \ldots, n+1$. Los cambios de cartas son claramente diferenciables. La familia de homeomorfismos $\mathcal{A} = \{(U_i, \phi_i) : i = 1, \ldots, n+1\}$ es un atlas en $\mathbb{RP}^n$.

La definición que dimos de espacio tangente a una variedad regular depende del ambiente euclídeo $\mathbb{R}^k$ que contiene a la variedad M, por lo que no puede extenderse a variedades diferenciables abstractas. Necesitaremos conocer las curvas diferenciables y el álgebra de las funciones diferenciables sobre M, que sólo dependen de la estructura diferenciable, para darle sentido a un concepto de vector tangente como derivación sobre $\mathcal{C}^{\infty}(M)$.

Definición 4.14. Sea M una variedad diferenciable y $p \in M$ un punto. Un **vector tangente** v en p a M es una derivación en el álgebra $\mathcal{C}_p^{\infty}(M)$, esto es, es una aplicación $v : \mathcal{C}_p^{\infty}(M) \to \mathbb{R}$ satisfaciendo:

1. v es lineal, esto es, $v(\lambda f + \mu g) = \lambda v(f) + \mu v(g) \ \ \forall f, g \in \mathcal{C}_p^{\infty}(M), \ \lambda, \mu \in \mathbb{R}$.
2. v cumple la regla del producto: $v(f \cdot g) = f(p)v(g) + g(p)v(f) \ \ \forall f, g \in \mathcal{C}_p^{\infty}(M)$.

El conjunto de los vectores tangentes en p a M es un espacio vectorial con la suma y multiplicación por escalares de derivaciones, y será denotado por T_pM.

Proposición 4.3. *Sean M^n una variedad diferenciable. $(U, \phi = (x_1, \ldots, x_n))$ una carta en M y $p \in U$ un punto. Entonces para todo $v \in T_pM$ se tiene que*

$$v = \sum_{j=1}^{n} v(x_j) \left.\frac{\partial}{\partial x_j}\right|_p .$$

En consecuencia el conjunto de derivaciones

$$B_\phi(p) := \left\{ \left.\frac{\partial}{\partial x_1}\right|_p, \ldots, \left.\frac{\partial}{\partial x_n}\right|_p \right\}$$

es una base de T_pM, que llamaremos inducida por la carta $(U, \phi = (x_1, \ldots, x_n)$ en el tangente en $p \in U$; en particular $dim(T_pM) = dim(M)$.

Definición 4.15. Una **curva diferenciable** en una variedad M es una aplicación diferenciable $\alpha : I \to M$, donde $I \subseteq \mathbb{R}$ es un intervalo abierto. Si $\alpha(t_0) = p, t_0 \in I$, la aplicación

$$\alpha'(t_0) : \mathcal{C}^\infty(M) \to \mathbb{R}, \;\; \alpha'(t_0)(f) := \left.\frac{d}{dt}\right|_{t_0} (f \circ \alpha)(t) \equiv (f \circ \alpha)'(t_0)$$

es una derivación en T_pM llamada **vector tangente** de α en t_0.

Proposición 4.4. *T_pM es el conjunto de los vectores tangentes a curvas diferenciables en M pasando por p, esto es,*

$$T_pM = \{\alpha'(t_0) : \alpha : I \to M \text{ diferenciable}, t_0 \in I \text{ y } \alpha(t_0) = p\}.$$

Sea M una variedad diferenciable n-dimensional.

Definición 4.16. El **fibrado tangente** a M es el conjunto

$$TM = \bigcup_{p \in M} T_pM,$$

donde los conjuntos $\{T_pM : p \in M\}$ son disjuntos dos a dos.

Definición 4.17. Un **campo de vectores** en M es una aplicación $X : M \to TM$ tal que $\pi \circ X = Id_M$. Denotaremos por $X_p = X(p)$ al representante del campo X en el punto $p \in M$.

Un campo X sobre M es diferenciable en un punto $p \in M$ si la aplicación es diferenciable en p. Denotaremos por $\mathfrak{X}(M)$ al conjunto de los campos diferenciables en todo punto de M.

5. Variedades pseudo-riemannianas

En este capítulo, introduciremos las variedades pseudo-riemannianas, definiremos el tensor de curvatura y veremos los conceptos de geodésicas y aplicación exponencial, comentando algunas de sus propiedades y ejemplos.

En primer lugar, definiremos las variedades pseudo-riemannianas. No obstante, necesitamos previamente la definición de métrica pseudo-riemanniana, que será un campo de tensores de tipo $(2,0)$ simétricos.

Definición 5.1. El fibrado de los tensores de tipo $(2,0)$, $0 \leq r, s$, se define como

$$\mathcal{T}_2^0(M) = \bigcup_{p \in M} \mathcal{T}_2^0(T_pM).$$

Los elementos de $\mathcal{T}_2^0(M)$ se denotan por $\mathrm{Y}_p \in \mathcal{T}_2^0(T_pM)$. Representaremos por $\pi : \mathcal{T}_2^0(M) \to M$, $\pi(\mathrm{Y}_p) = p$, a la proyección natural.

Definición 5.2. Un **campo de tensores** Y de tipo $(2,0)$ sobre una variedad diferenciable M, es una correspondencia que asocia a cada punto $p \in M$ un tensor $\mathrm{Y} \in \mathcal{T}_2^0(M)$. En otras palabras, $\pi \circ \mathrm{Y} = Id_M$. En términos de aplicaciones, Y es una aplicación de M a $\mathcal{T}_2^0(M)$. Denotaremos por $\mathrm{Y}_p = \mathrm{Y}(p)$ al representante del tensor Y en el punto $p \in M$.

Un campo de tensores Y de tipo $(2,0)$ sobre M se dirá diferenciable en un punto $p \in M$ si la aplicación $\mathrm{Y} : M \to \mathcal{T}_2^0(M)$ es diferenciable en p. Denotaremos por

$$\mathfrak{T}_2^0(M) = \{\mathrm{Y} \in \mathcal{C}^\infty(M, \mathcal{T}_2^0(M)) : \pi \circ \mathrm{Y} = Id_M\}$$

al conjunto de los campos tensoriales de tipo $(2,0)$ sobre M diferenciables en todo punto de M.

Definición 5.3. Un campo de tensores $\mathrm{Y} : M \to \mathcal{T}_2^0(M)$, se dice **simétrico** si $\mathrm{Y}_p \in \mathcal{S}_2(T_pM)$ para todo $p \in M$, con $\mathcal{S}_2(T_pM)$ denotando el espacio vectorial de tensores simétricos de orden 2 en el espacio tangente T_pM.

Los campos de tensores simétricos sobre M se pueden presentar como secciones de su fibrado natural, a saber, el de los tensores simétricos de orden 2:

$$\mathcal{S}_2(M) = \bigcup_{p \in M} \mathcal{S}_2(T_pM).$$

Denotaremos por

$$\mathfrak{S}_2(M) = \{\mathrm{Y} \in \mathcal{C}^\infty(M, \mathcal{S}_2(M)) : \pi \circ \mathrm{Y} = Id_M\} = \{\mathrm{Y} \in \mathfrak{T}_2^0(M) : \mathrm{Y} \text{ simétrico}\}.$$

El conjunto $\mathfrak{S}_2(M)$ es un espacio vectorial y un módulo sobre el anillo $\mathcal{C}^\infty(M)$.

Definición 5.4. Un campo de tensores $g \in \mathfrak{S}_2(M)$ se dirá una **métrica** sobre la variedad M. Una métrica g sobre M se dirá:

- **Riemanniana** si $g_p \in \mathcal{S}_2(T_pM)$ es euclídea para todo $p \in M$.
- **Lorentziana** si $g_p \in \mathcal{S}_2(T_pM)$ tiene signatura $(n-1,1)$ para todo $p \in M$.
- **Pseudo-riemanniana** si $g_p \in \mathcal{S}_2(T_pM)$ tiene signatura (r,s), con $0 < r < r+s = n$ para todo $p \in M$.

Definición 5.5. Una **variedad pseudo-riemanniana** es un par (M,g) formado por una variedad diferenciable M y una métrica pseudo-riemanniana.

Definición 5.6. La **norma** de un vector $v \in T_pM$ en una métrica pseudo-riemanniana g se define como

$$\|v\| = \sqrt{|g(v,v)|}.$$

Definición 5.7. Clasificamos las curvas $\gamma : I \to M$, siendo I un intervalo real, según el carácter de sus vectores tangentes en cada punto:

- **Espacial**: Una curva γ se dice espacial si, para todo vector tangente $\gamma'(t)$ de γ, $g(\gamma'(t),\gamma'(t)) > 0$, es decir, si todos sus vectores tangentes son espaciales.
- **Temporal**: Una curva γ se dice temporal si, para todo vector tangente $\gamma'(t)$ de γ, $g(\gamma'(t),\gamma'(t)) < 0$, significando que todos sus vectores tangentes son temporales.
- **Luminosa**: Una curva γ se dice luminosa si, para todo vector tangente $\gamma'(t)$ de γ y $\gamma'(t) \neq 0$, $g(\gamma'(t),\gamma'(t)) = 0$, indicando que todos sus vectores tangentes no nulos son luminosos.
- **Causal**: Una curva γ se dice causal si es temporal o luminosa.

Esta clasificación depende estrictamente del comportamiento de la métrica g con respecto a los vectores tangentes a la curva en la variedad.

Ejemplo 5.1. El espacio de Lorentz-Minkowski, denotado como $\mathbb{M}^n$, es el escenario fundamental de la teoría de la relatividad especial. Se puede definir como el conjunto $\mathbb{R}^n$ equipado con la métrica de Lorentz g_L, donde

$$g_L(\mathbf{x},\mathbf{y}) = -x_1y_1 + \sum_{i=2}^{n} x_iy_i,$$

para vectores $\mathbf{x},\mathbf{y} \in \mathbb{R}^n$. Esta métrica define una estructura lorentziana en $\mathbb{R}^n$, donde el tiempo y el espacio se tratan de manera asimétrica. El espacio de Minkowski es fundamental para la formulación matemática de la física en cuatro dimensiones, donde una dimensión corresponde al tiempo y las tres restantes al espacio [O'N83].

Ejemplo 5.2. El espacio de De Sitter se define como:

$$\mathcal{Q}^1_{\alpha^2} = \left\{ \mathbf{x} \in \mathbb{R}^{d+1} : g(\mathbf{x},\mathbf{x})_L = -x_1^2 + \sum_{i=2}^{d+1} x_i^2 = \alpha^2 \right\} \text{ con } \alpha \in \mathbb{R} \setminus \{0\}$$

donde g_L es la métrica de Lorentz en $\mathbb{R}^{d+1}$. Este conjunto $\mathcal{Q}^1_{\alpha^2}$ es una subvariedad de $\mathbb{R}^{d+1}$ equipada con la métrica inducida por g_L, que hereda una estructura lorentziana.

5.1. Tensor de curvatura

Toda métrica, ya sea riemanniana o pseudo-riemanniana, tiene una conexión afín canónicamente asociada que permite derivar campos vectoriales y, a partir de ellos, cualquier tipo de campos tensoriales sobre la variedad.

Teorema 5.1. *Sea (M, g) una variedad pseudo-riemanniana. Entonces existe un única conexión afín $\nabla(\equiv \nabla_g)$, es decir, una aplicación $\nabla : \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}(M)$, $(X, Y) \mapsto \nabla_X Y$, $\mathcal{C}^\infty(M)$-lineal en el primer argumento y $\mathbb{R}$ lineal en el segundo que además satisface que $\nabla_X(fY) = X(f)Y + f\nabla_X Y \ \ \forall f \in \mathcal{C}^\infty(M)$, que cumple:*

1. *Es libre de torsión, lo que significa que*

$$\nabla_X Y - \nabla_Y X = [X, Y], \ \ \forall X, Y \in \mathfrak{X}(M),$$

   donde $[X, Y]$ es el corchete de Lie de los campos vectoriales X e Y.

   El corchete de Lie de dos campos de vectores diferenciables X e Y sobre una variedad M se define como el único campo de vectores que satisface $[X, Y](f) = X(Y(f)) - Y(X(f))$.

2. *Paraleliza a la métrica,*

$$X(g(Y, Z)) = g(\nabla_X Y, Z) + g(Y, \nabla_X Z) \ \ \forall X, Y, Z \in \mathfrak{X}(M),$$

   donde $X(g(Y, Z))$ denota la derivada de $g(Y, Z)$ a lo largo del campo vectorial X.

Esta conexión afín, ∇, se le conoce como la conexión de Levi-Civita asociada a la métrica g.

En particular, si $(\mathcal{U}, \psi = (x^1, \ldots, x^n))$ es un entorno coordenado y $\partial_i (\equiv \partial/\partial x^i)$ denota el i-ésimo campo vectorial coordenado, es posible obtener los símbolos de Christoffel $\nabla_{\partial_i}\partial_j$ de ∇, esto es, las componentes de $\nabla_{\partial_i}\partial_j$ en la base $\{\partial_i\}$:

$$\nabla_{\partial_i}\partial_j = \sum_{k=1}^{n} \Gamma_{ij}^k \partial_k,$$

entonces

$$\Gamma_{ij}^k = \frac{1}{2}\sum_{l=1}^{n} g^{kl}(\partial_i g_{jl} + \partial_j g_{li} - \partial_l g_{ij}), \tag{5.1}$$

donde g^{kl} es el componente inverso de la métrica g, y Γ_{ij}^k son los símbolos de Christoffel.

Definición 5.8. Definimos el **tensor de curvatura** R, un tensor de tipo $(3, 1)$, en los campos vectoriales por

$$R(X, Y)Z = \nabla_X \nabla_Y Z - \nabla_Y \nabla_X Z - \nabla_{[X,Y]} Z, \ \ \forall X, Y, Z \in \mathfrak{X}(M).$$

Podemos expresar las componentes del tensor de curvatura en términos de los símbolos de Christoffel:

$$R_{ijk}^l = \partial_k \Gamma_{ji}^l - \partial_j \Gamma_{ki}^l + \Gamma_{ji}^m \Gamma_{km}^l - \Gamma_{ki}^m \Gamma_{jm}^l. \ \text{[KS24b]} \tag{5.2}$$

Sea ∇ la conexión de Levi-Civita de la métrica pseudo-riemanniana g, podemos construir el tensor de curvatura 4-covariante:

$$R(X,Y,Z,W) := g(R(X,Y)Z,W).$$

Estos tensores presentan las siguientes simetrías:

1. $R(X,Y,Z,W) = -R(Y,X,Z,W)$
2. $R(X,Y,Z,W) = -R(X,Y,W,Z)$
3. $R(X,Y,Z,W) = -R(Z,W,X,Y)$
4. $R(X,Y)Z + R(Z,X)Y + R(Y,Z)X = 0$
5. $(\nabla_Z R)(X,Y) + (\nabla_Y R)(Z,X) + (\nabla_X R)(Y,Z) = 0$

Las únicas contracciones, no necesariamente nulas, del tensor de curvatura se conocen como el tensor de Ricci, *Ric*, y la curvatura escalar, S. El tensor de Ricci es un tensor simétrico 2-covariante que se obtiene como la contracción con respecto a los últimos índices del tensor de curvatura:

$$Ric(Y,Z)(p) := \text{traza}R(\cdot,Y_p)Z_p = \sum_{i,j=1}^{n} g^{ij} g(R(\partial_i|_p, Y_p)Z_p, \partial_j|_p) = \sum_{i,j=1}^{n} g^{ij} R(\partial_i|_p, Y_p, Z_p, \partial_j|_p).$$

La curvatura escalar S es la contracción métrica del tensor de Ricci:

$$S(p) = \sum_{i,j=1}^{n} g^{ij}(p) Ric(\partial_i|_p, \partial_j|_p).$$

Definición 5.9. Si g es una métrica riemanniana y $\Pi = \langle u, v\rangle_{\mathbb{R}} \subseteq T_pM$ es un plano del espacio tangente de M en p, se define la **curvatura seccional** de Π como:

$$K_S(\Pi) = \frac{R(u,v,v,u)}{g(u,u)g(v,v) - g(u,v)^2},$$

que es independiente de la base (u,v) de Π escogida, ya que sean otros dos vectores $u' = au + bv$, $v' = cu + dv$, entonces por las simetrías del tensor de curvatura se tiene que:

$$R(u',v',v',u') = (ad-bc)^2 R(u,v,v,u)\ \forall u,v,u',v' \in T_pM, p \in M.$$

Por tanto, podemos escoger una base ortonormal de T_pM y entonces

$$g(u,u)g(v,v) - g(u,v)^2 = \pm 1,$$

simplificando así la definición de curvatura seccional:

$$K_S(\Pi) = \pm R(u,v,v,u).$$

En el caso de que g sea pseudo-riemanniana, la definición es igual salvo que solo está definida si el espacio tangente Π es no degenerado. Esto es debido a que si $g|_\Pi$ es degenerado, entonces $Q(u,v) = g(u,u)g(v,v) - g(u,v)^2 = 0$. De hecho, se tienen las siguientes

equivalencias:

$$\begin{cases} Q(u,v) > 0 \iff g|_\Pi \text{ es definida positiva o definida negativa} \\ Q(u,v) = 0 \iff g|_\Pi \text{ es degenerada} \\ Q(u,v) < 0 \iff g_\Pi \text{ es indefinida} \end{cases}$$

En el caso de que Π sea un plano degenerado, podemos definir la signatura de la curvatura:

$$\mathcal{N}(\Pi) = \begin{cases} +1 \text{ si } R(u,v,v,u) > 0 \\ 0 \text{ si } R(u,v,v,u) = 0 \\ -1 \text{ si } R(u,v,v,u) < 0 \end{cases}$$

Sea una variedad pseudo-riemanniana $(\overline{M}, \overline{g})$. Vamos a estudiar qué ocurre para una subvariedad diferenciable (M, g), $M \subseteq \overline{M}$, con $g = j^*\overline{g}$, siendo $j : M \hookrightarrow \overline{M}$ la inclusión y $j^*\overline{g}$ el pullback de la métrica $\overline{g}$ a través de la inclusión j. Esto significa que la métrica g en la subvariedad M es la métrica inducida de la métrica $\overline{g}$ en la variedad ambiente $\overline{M}$, es decir, si $p \in M$ y $v, w \in T_pM$ son vectores tangentes en M en el punto p, la métrica inducida g en M se define por:

$$g(v,w) = \overline{g}(dj(v), dj(w)),$$

donde dj es el diferencial de la inclusión j, que lleva vectores tangentes en M a vectores tangentes en $\overline{M}$.

Definición 5.10. Definimos un $\overline{M}$-campo vectorial en M como X un campo vectorial en $j : M \hookrightarrow \overline{M}$, esto es, $X : M \to T\overline{M}$ tal que $\pi_{\overline{M}} \circ X = j$.

De igual forma a como lo hacíamos en Def. 4.17,

$$\overline{\mathfrak{X}}(M) = \{X \in \mathcal{C}^\infty(M, T\overline{M}) : X \text{ es un campo vectorial sobre } j\}.$$

Destacamos las siguientes observaciones.

Observación 5.1.

1. $X \in \overline{\mathfrak{X}}(M)$ asigna a cada punto $p \in M$ algún $X_p \in T_p\overline{M}$ de una forma diferenciable. Para todo $f \in \mathcal{C}^\infty(M)$, $X(f) \in \mathcal{C}^\infty$, por tanto, $\overline{\mathfrak{X}}(M)$ es un $\mathcal{C}^\infty(M)$-módulo y $\mathfrak{X}(M)$ es un submódulo de $\overline{\mathfrak{X}}(M)$. Además, si $X \in \mathfrak{X}(\overline{M})$, entonces $X|_M \in \overline{\mathfrak{X}}(M)$.

2. Sea M una subvariedad pseudo-riemanniana, por definición $\overline{g}|_{T_pM} = j^*\overline{g}(p)$ es no degenerada y por tanto, $T_pM \subseteq T_p\overline{M}$ y $T_pM^\perp$ son no degenerados. Esto implica que

$$T_p\overline{M} = T_pM \oplus T_pM^\perp.$$

Además, la dimensión de $T_pM^\perp$ es la codimensión de M en $\overline{M}$. Al índice de $\overline{g}|_{T_pM^\perp}$ se le conoce como coíndice de M en $\overline{M}$:

$$\text{ind}(\overline{M}) = \text{ind}(M) + \text{coind}(M).$$

Definición 5.11. A un campo vectorial Z se le dice **normal a** M si $Z_p \in T_pM^\perp \ \ \forall p \in M$.

El conjunto de todos estos campos vectoriales, $\mathfrak{X}(M)^\perp$ es un $C^\infty(M)$-submódulo de $\overline{\mathfrak{X}}(M)$.

Proposición 5.1. *La conexión inducida* $\overline{\nabla} : \mathfrak{X}(M) \times \overline{\mathfrak{X}}(M) \to \overline{\mathfrak{X}}(M)$ *de* $M \subseteq \overline{M}$ *cumple las siguientes propiedades para* $V, W \in \mathfrak{X}(M)$, $X, Y \in \overline{\mathfrak{X}}(M)$ *y* $f \in \mathcal{C}^\infty$:

1. $\overline{\nabla}_V X$ *es* $\mathcal{C}^\infty(M)$*-lineal en* V.
2. $\overline{\nabla}_V X$ *es* $\mathbb{R}$*-lineal en* X.
3. $\overline{\nabla}_V(fX) = V(f)X + f\overline{\nabla}_V X$.
4. *Es libre de torsión, es decir:* $\overline{\nabla}_V W - \overline{\nabla}_W V = [V, W]$, *donde* $[V, W]$ *es el corchete de Lie.*
5. *Paraleliza la métrica,* $V(g(X, Y)) = g(\overline{\nabla}_V X, Y) + g(X, \overline{\nabla}_V Y)$.

Definición 5.12. La función $II : \mathfrak{X}(M) \times \mathfrak{X}(M) \to \mathfrak{X}^\perp$ definida como

$$II(V, W) = (\overline{\nabla}_V W)^\perp,$$

es $\mathcal{C}^\infty(M)$-bilineal y simétrica, se le conoce como segunda forma fundamental de M en $\overline{M}$

Teorema 5.2. ***(Ecuación de Gauss).*** *Sea* M *una subvariedad pseudo-riemanniana de* $\overline{M}$ *con tensores de curvatura* R *y* $\overline{R}$*, respectivamente. Entonces* $\forall V, W, X, Y \in \mathfrak{X}(M)$

$$\langle R_{VW}X, Y\rangle = \langle \overline{R}_{VW}X, Y\rangle + \langle II(V, X), II(W, Y)\rangle - \langle II(V, Y), II(W, X)\rangle,$$

donde II *es la segunda forma fundamental de* M *en* $\overline{M}$.

Veamos algunos ejemplos de tensores de curvatura para distintas variedades.

Ejemplo 5.3. Consideremos $\mathbb{R}^n$ con la métrica euclídea $g_E(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n x_i y_i$. Expresada con deltas de Kronecker:

$$g_{E_{ij}} = \delta_{ij},$$

con $\delta_{ij} = 0$ si $i \neq j$, y $\delta_{ij} = 1$ si $i = j$. Como todas sus componentes son constantes, todas sus derivadas parciales son cero:

$$\partial_i g_{E_{jl}} = \partial_j g_{Eil} = \partial_l g_{E_{ij}} = 0.$$

Por la ecuación (5.1), se tiene que los símbolos de Christoffel también son nulos

$$\Gamma^k_{ij} = \frac{1}{2} g^{kl}(\partial_i g_{jl} + \partial_j g_{il} - \partial_l g_{ij}) = 0.$$

Finalmente, sustituyendo en la expresión (5.2), obtenemos que:

$$R^l_{ijk} = 0.$$

Por tanto, como era de esperar, el tensor de curvatura R para cualquier plano en $\mathbb{R}^n$ con la métrica euclídea es cero.

Otra forma de verlo es la siguiente. La conexión de Levi-Civita en este caso es $\nabla_X Y = dY(X)$. De lo que obtenemos que si $X, Y, Z \in \mathfrak{X}(\mathbb{R}^n)$, entonces

$$\nabla_Y Z = \sum_j Y_j dZ(e_j), \quad \nabla_X \nabla_Y Z = \sum_{i,j} X_i dY_j(e_i) dZ(e_j) + \sum_{i,j} X_i Y_j (d^2 Z)(e_i, e_j),$$

donde $\{e_1, \ldots, e_n\}$ es la base canónica de $\mathbb{R}^n$ y $X = \sum_i X_i e_i$, $Y = \sum_i Y_i e_i$. Por tanto,

$$\nabla_X \nabla_Y Z - \nabla_Y \nabla_X Z = \sum_{i,j} [X_i dY_j(e_i) - Y_i dX_j(e_i)] dZ(e_j).$$

Además,

$$\nabla_{[X,Y]} Z = \sum_{i,j} \nabla_{[X_i e_i, Y_j e_j]} Z$$

$$\sum_{i,j} \nabla_{X_i e_i(Y_j) e_j - Y_j e_j(X_i) e_i} Z = \sum_{i,j} [X_i dY_j(e_i) - Y_i dX_j(e_i)] dZ(e_j).$$

Por lo que finalmente tenemos que $R(X,Y)Z = 0 \ \ \forall X, Y, Z \in \mathfrak{X}(\mathbb{R}^n)$.

Ejemplo 5.4. Sea la esfera $\mathbb{S}^2 = \{\mathbf{s} = (x,y,z) \in \mathbb{R}^3 : g(\mathbf{s},\mathbf{s}) = x^2 + y^2 + z^2 = r\}$ de radio r, con la métrica euclídea inducida de $\mathbb{R}^3$.

La conexión de Levi-Civita es

$$(\nabla_X Y)_p = dY_p(X_p) + \langle X_p, Y_p \rangle p$$

Deducimos que

$$\nabla_X \nabla_Y Z = \overline{\nabla}_X \overline{\nabla}_Y Z + X(\langle Y, Z \rangle) p + \langle Y, Z \rangle X + \langle X, dZ(Y) \rangle p,$$

donde $\overline{\nabla}$ es la conexión de Levi-Civita de $(\mathbb{R}^3, g_E = \langle \cdot, \cdot \rangle)$. Si cambiamos X por Y y usando que $\nabla_{[X,Y]} Z = \overline{\nabla}_{[X,Y]} Z + \langle [X,Y], Z \rangle p$, obtenemos que

$$R(X,Y)Z = \langle Y, Z \rangle X - \langle X, Z \rangle Y, \ \ \forall X, Y, Z \in \mathfrak{X}(\mathbb{S}^2).$$

Ejemplo 5.5. Sea el espacio de Sitter definido como:

$$\mathcal{Q}^1_{\alpha^2} = \left\{ \mathbf{x} \in \mathbb{R}^{d+1} : g(\mathbf{x},\mathbf{x})_L = -x_1^2 + \sum_{i=2}^{d+1} x_i^2 = \alpha^2 \right\},$$

donde $g(\mathbf{x},\mathbf{x})_L$ es la métrica de Lorentz. Para el espacio de Sitter, utilizamos la conexión inducida desde $\mathbb{R}^{d+1}$ con la métrica de Lorentz:

$$(\nabla_X Y)_p = dY_p(X_p) + \langle X_p, Y_p \rangle_L p.$$

En $\mathbb{R}^{d+1}$ con la métrica de Lorentz, el tensor de curvatura es cero, es decir, $\overline{R}(X,Y)Z = 0$.

El gradiente de la función $f(\mathbf{x}) = -x_1^2 + \sum_{i=2}^{d+1} x_i^2$ es:

$$\text{grad}(f) = 2 \sum_{i=1}^{d+1} x_i \frac{\partial}{\partial x_i}.$$

El vector normal unitario al espacio de Sitter es:

$$\nu(\mathbf{x}) = \frac{1}{\alpha} \sum_{i=1}^{d+1} x_i \frac{\partial}{\partial x_i}.$$

La segunda forma fundamental $II(V,W)$ se expresa como:

$$II(V,W) = -\frac{1}{\alpha}\langle V,W\rangle \nu,$$

donde $\langle\cdot,\cdot\rangle$ es la métrica de Lorentz.

Utilizamos la ecuación de Gauss para relacionar la curvatura del espacio de Sitter con la curvatura del espacio ambiente $\mathbb{R}^{d+1}$:

$$\langle R_{\mathcal{Q}^1_{\alpha^2}}(V,W)X,Y\rangle = \langle II(V,X),II(W,Y)\rangle - \langle II(V,Y),II(W,X)\rangle.$$

Sustituyendo $II(V,W)$ en la ecuación anterior, obtenemos:

$$\langle R_{\mathcal{Q}^1_{\alpha^2}}(V,W)X,Y\rangle = \frac{1}{\alpha^2}\left(\langle V,X\rangle\langle W,Y\rangle - \langle W,X\rangle\langle V,Y\rangle\right).$$

Finalmente, el tensor de curvatura del espacio de Sitter es:

$$R_{\mathcal{Q}^1_{\alpha^2}}(V,W)X = \frac{1}{\alpha^2}\left(\langle V,X\rangle W - \langle W,X\rangle V\right),$$

donde $\langle\cdot,\cdot\rangle$ es la métrica de Lorentz.

Observación 5.2. Algunos de los teoremas que no se cumplen para el caso pseudo-riemanniano son los siguientes:

1. No es verdadero que toda variedad diferenciable admite una métrica pseudo-riemanniana de una signatura dada (véase [O'N83, Cap. 5, Prop. 37]). Por ejemplo, la 2-esfera no admite ninguna métrica pseudo-riemanniana de signatura indefinida.

2. Una subvariedad no siempre hereda la estructura de una variedad pseudo-riemanniana. Esto se debe a que en una variedad pseudo-riemanniana, el vector tangente a una curva luminosa tiene una longitud cero según la métrica pseudo-riemanniana. Esto implica que la métrica restringida a la curva no puede ser no degenerada. Sin embargo, si la curva es espacial o temporal, la métrica induce un tensor métrico no degenerado. Esta característica resalta una diferencia con el caso riemanniano, donde la métrica inducida en cualquier subvariedad siempre es no degenerada.

5.2. Geodésicas y aplicación exponencial

En esta sección, presentaremos el concepto de geodésica e introduciremos la aplicación exponencial, una herramienta que nos permite establecer una relación local entre una variedad diferenciable M y su espacio tangente en un punto dado T_pM. Empezaremos recordando la definición de longitud de una curva, para así poder introducir las geodésicas en variedades riemannianas y extender esta definición para el caso pseudo-riemanniano. Continuaremos describiendo algunas propiedades interesentas de las geodésicas y mostrando ejemplos concretos. Finalmente, daremos la definición de aplicación exponencial, veremos sus propiedades más importantes y exploraremos el caso de la familia de estudio de este trabajo. Se han consultado principalmente los siguientes libros [O'N83, VC10]

5.2.1. Geodésicas

Antes de introducir el concepto de geodésicas, es indispensable definir la longitud de una curva en una variedad M.

Definición 5.13. Sea $\alpha : [a, b] \to M$ una curva diferenciable a trozos, su longitud desde a hasta b ($a, b \in \mathbb{R}, a < b$) se define como:

$$L_a^b(\alpha) := \int_a^b \|\alpha'(t)\| dt,$$

donde $\|\cdot\|$ denota la norma respecto a la métrica g de la variedad M.

Lema 5.1. *La longitud de una curva diferenciable a trozos no cambia bajo reparametrizaciones monótonas. Si α es una curva regular ($\|\alpha'\| > 0$), entonces existe una reparametrización h estrictamente creciente tal que $\beta = \alpha \circ h$ y $\|\beta'\| = 1$. Se dice que β está parametrizada por el arco.*

Definimos primero las geodésicas en el contexto de la geometría riemanniana.

Definición 5.14. En una variedad riemanniana M, una **geodésica** es una curva $\gamma : [a, b] \to M$ que es un punto crítico del funcional de longitud, es decir, minimiza localmente la distancia entre puntos cercanos. Esto significa que el vector tangente γ' es paralelo a lo largo de γ:

$$\nabla_{\gamma'} \gamma' = 0,$$

donde ∇ es la conexión afín asociada con la métrica de M y γ' es el vector tangente a γ.

Para extender esta definición a variedades pseudo-riemannianas, se necesita de una motivación adicional debido a que el concepto de longitud puede no aplicarse de forma directa (en el caso de curvas luminosas tienen longitud cero). En este contexto más general, una geodésica seguirá siendo una curva γ cuyo vector tangente γ' es paralelo a lo largo de γ:

$$\nabla_{\gamma'} \gamma' = 0.$$

Definición 5.15. En variedades pseudo-riemannianas regulares $M \subseteq \mathbb{R}^n$, una geodésica también puede ser descrita como una curva cuyo vector de aceleración $\gamma''(t)$ es perpendicular al espacio tangente $T_{\gamma(t)}M$ en cada punto de la curva:

$$\gamma''(t) \perp T_{\gamma(t)}M, \quad \forall t \in [a, b].$$

Esto es posible porque aquí $\nabla_{\gamma'} \gamma' = (\gamma'')^\top$, donde $(\gamma'')^\top$ denota la proyección del vector aceleración sobre el espacio tangente.

Corolario 5.1. *Sea $x^1, \ldots, x^n$ un sistema de coordenadas en $\mathcal{V} \subset M \subseteq \mathbb{R}^n$. Una curva γ en $\mathcal{V}$ es una geodésica de M si, y solo si, sus funciones coordenadas $x^k \circ \gamma$ cumplen que*

$$\frac{d^2(x^k \circ \gamma)}{dt^2} + \sum_{i,j=1}^{n} \Gamma_{ij}^k(\gamma) \frac{d(x^i \circ \gamma)}{dt} \frac{d(x^j \circ \gamma)}{dt} = 0, \quad \forall 1 \leq k \leq n.$$

Estas expresiones son las componentes de γ'' respecto a los campos de vectores coordenados $\partial_1, \ldots, \partial_n$.

Gracias al Teorema de existencia y unicidad para ecuaciones diferenciales ordinarias tenemos el siguiente resultado local.

Lema 5.2. *Si $v \in T_pM$, entonces existe un intervalo I que contiene al 0 y una única geodésica $\gamma_{p,v} : I \to M$ tal que $\gamma'_{p,v}(0) = v$ y $\gamma_{p,v}(0) = p$, por lo que denotaremos $\gamma_{p,v}$ a la geodésica que parte del punto p con velocidad inicial v.*

Lema 5.3. *Sean $\alpha, \beta : I \to M$ geodésicas. Si existe $a \in I$ tal que $\alpha'(a) = \beta'(a)$, entonces $\alpha(t) = \beta(t) \ \ \forall t \in I$.*

Demostración. Supongamos que $\alpha \neq \beta$, entonces debe existir un $t_0 \in I$ tal que $\alpha(t_0) \neq \beta(t_0)$. Si $t_0 > a$, entonces el conjunto $\{t \in I : t > a \text{ y } \alpha(t) \neq \beta(t)\}$ tiene ínfimo igual a b, con $b \geq a$. Si $b = a$, entonces $\alpha'(b) = \beta'(b)$. Si $b > a$, entonces α y β coinciden en el intervalo (a, b), y debido a la continuidad y diferenciabilidad de $\alpha'(t)$ y $\beta'(t)$, se tiene que:

$$\alpha'(b) = \lim_{t \to b^-} \alpha'(t) = \lim_{t \to b^-} \beta'(t) = \beta'(b),$$

Y como $\alpha(t + b)$ y $\beta(t + b)$ también son geodésicas, el lema anterior afirma que $\alpha = \beta$ en algún intervalo alrededor de b. Pero esto contradice la definición de b como ínfimo.

□

Proposición 5.2. *Para cualquier vector tangente $v \in T_pM$ existe una única geodésica $\gamma_{p,v}$ en M tal que:*

1. *La velocidad inicial de $\gamma_{p,v}$ es v, es decir, $\gamma'_{p,v}(0) = v$.*
2. *El dominio I de $\gamma_{p,v}$ es el más grande posible. Si $\alpha_{p,v} : J \to M$ es otra geodésica con velocidad inicial v, entonces $J \subset I$ y $\alpha_{p,v} = \gamma_{p,v}|_J$*

Demostración. Sea $\mathcal{G}$ el conjunto de todas las geodésicas $\gamma_{p,v} : I_\gamma \to M$ con velocidad inicial v, por el Lema 5.2 $\mathcal{G}$ no es vacío. Aplicando ahora el Lema 5.3, obtenemos que todo par de geodésicas α y β en $\mathcal{G}$ coinciden en $I_\alpha \cap I_\beta$. Por tanto, $\mathcal{G}$ define una sola curva $\gamma_{p,v}$ en el intervalo $I = \cup I_\gamma$ y $\gamma_{p,v}$ cumple las condiciones de geodésica. □

Diremos que una geodésica $\gamma : [a, b) \to M$ es extendible de forma continua si tiene una extensión continua a una curva definida en $[a, b]$ y diremos que es **geodésicamente extendible** si tiene una extensión a una geodésica definida en $[a, c)$, con $c > b$.

Definición 5.16. Una geodésica $\gamma_{p,v}$ es **maximal** si no existe geodésica $\tilde{\gamma} : \tilde{I} \to M$ que extienda a $\gamma_{p,v}$, es decir, tal que $\tilde{I}$ contenga estrictamente a I y $\tilde{\gamma}(t) = \gamma_{p,v}(t), \ \forall t \in I$.

Corolario 5.2. *Una geodésica $\gamma : [0, b) \to M$ es geodésicamente extendible si, y solo si, es extendible de forma continua.*

Definición 5.17. Una curva diferenciable a trozos es una **geodésica rota** cuando su restricción a cada uno de los intervalos donde es diferenciable sea geodésica.

Lema 5.4. *Una variedad diferenciable M es conexa si, y solo si, dos puntos cualesquiera de M se pueden unir por una geodésica rota.*

Demostración. La implicación hacia la izquierda es directa. Ahora, supongamos que M es conexa y sea un punto $p \in M$. Sea $\mathcal{C}$ el conjunto de puntos que pueden ser conectados a p por una geodésica rota. Sea $\mathcal{U}_q$ un entorno normal de $q \in M$ (la definición de entorno normal se encuentra en Subsección 5.2.3). Si $q \in \mathcal{C}$, entonces $\mathcal{U}_q \subset \mathcal{C}$. Y si $q \in M \setminus \mathcal{C}$, entonces $\mathcal{U} \subset M \setminus \mathcal{C}$. Entonces por conectividad, $M = \mathcal{C}$. □

Veamos algunos ejemplos de geodésicas:

Ejemplo 5.6. En $\mathbb{R}^n$, las geodésicas son segmentos de recta. En efecto, consideremos la curva $\gamma_{p,v}(t) = p + tv$, donde p es un punto en $\mathbb{R}^n$ y v es un vector en $\mathbb{R}^n$. La derivada de $\gamma_{p,v}$ es constante, $\gamma'_{p,v}(t) = v \ \forall t \in \mathbb{R}$.

Demostremos que esa curva es una geodésica. Calculamos su primera derivada:

$$\gamma'_{p,v}(t) = \frac{d}{dt}(p + tv) = v, \text{ que es constante,}$$

por lo que la segunda derivada, $\gamma''_{p,v}(t) = \frac{d}{dt}(v) = 0$. Por lo que se tiene que $\nabla_{\gamma'}\gamma'_{p,v} = \gamma''_{p,v}(t) = 0$ cumpliendo así la condición de que $\gamma_{p,v}$ es geodésica.

Ejemplo 5.7. En la esfera $\mathbb{S}^2$, las geodésicas son los círculos máximos. Sea la curva γ en $\mathbb{S}^2$ definida por:

$$\gamma_{p,v}(t) = \cos(\|v\|t)p + \sin(\|v\|t)\frac{v}{\|v\|},$$

con p es un punto de $\mathbb{S}^2$, $v \in T_p\mathbb{S}^2, v \neq 0$ y $\|v\|$ es su norma.

Su derivada es:

$$\gamma'_{p,v}(t) = -\|v\|\sin(\|v\|t)p + \|v\|\cos(\|v\|t)\frac{v}{\|v\|},$$

y su segunda derivada:

$$\gamma''_{p,v}(t) = -\|v\|^2\cos(\|v\|t)p - \|v\|^2\sin(\|v\|t)\frac{v}{\|v\|} = -\|v\|^2\gamma(t)$$

Como $\gamma''_{p,v}(t)$ es proporcional a $\gamma_{p,v}$, la curva no tiene aceleración tangencial, lo que significa que $\nabla_{\gamma'_{p,v}}\gamma'_{p,v} = 0$, y por tanto es una geodésica que pasa por p en $t = 0$ con velocidad v.

Ejemplo 5.8. Por último, veamos una geodésica en una variedad lorentziana, el espacio de De Sitter:

$$\mathcal{Q}^1_{\alpha^2} = \left\{ \mathbf{x} \in \mathbb{R}^{d+1} : g(\mathbf{x},\mathbf{x})_L = -x_1^2 + \sum_{i=2}^{d+1} x_i^2 = \alpha^2 \right\},$$

donde g_L es la métrica de Lorentz en $\mathbb{R}^{d+1}$.

Sea la curva $\gamma_{p,v}(t)$ en $\mathcal{Q}^1_{\alpha^2}$ definida por:

$$\gamma_{p,v}(t) = \cosh\left(\frac{\|v\|t}{\alpha}\right)p + \alpha\sinh\left(\frac{\|v\|t}{\alpha}\right)\frac{v}{\|v\|},$$

donde p es un punto en $\mathcal{Q}^1_{\alpha^2}$, $v \in T_p\mathcal{Q}^1_{\alpha^2}$, $v \neq 0$, y $\|v\|$ es su norma con respecto a la métrica de Lorentz.

Su derivada es:

$$\gamma'_{p,v}(t) = \frac{\|v\|}{\alpha}\sinh\left(\frac{\|v\|t}{\alpha}\right)p + \|v\|\cosh\left(\frac{\|v\|t}{\alpha}\right)\frac{v}{\|v\|},$$

y su segunda derivada:

$$\gamma''_{p,v}(t) = \left(\frac{\|v\|}{\alpha}\right)^2\cosh\left(\frac{\|v\|t}{\alpha}\right)p + \frac{\|v\|^2}{\alpha}\sinh\left(\frac{\|v\|t}{\alpha}\right)\frac{v}{\|v\|}.$$

Usando la identidad $\cosh^2(x) - \sinh^2(x) = 1$, podemos escribir:

$$\gamma''_{p,v}(t) = \frac{\|v\|^2}{\alpha^2}\gamma_{p,v}(t).$$

Por tanto, al igual que pasaba en el ejemplo anterior, como $\gamma''_{p,v}(t)$ es proporcional a $\gamma_{p,v}(t)$, la curva no tiene aceleración tangencial con respecto a la conexión de Levi-Civita de la métrica, lo que significa que $\nabla_{\gamma'_{p,v}}\gamma'_{p,v} = 0$, y por tanto es una geodésica en $\mathcal{Q}^1_{\alpha^2}$.

5.2.2. Aplicación exponencial

En este apartado definiremos y examinaremos la aplicación exponencial. Como veremos, provee con medios para relacionar localmente M y T_pM.

Definición 5.18. Definimos la **aplicación exponencial** en $p \in M$ como:

$$\exp_p : W_p \subseteq T_pM \to M, \quad \exp(v) = \gamma_v(1)$$

donde γ_v es la única geodésica no extendible con velocidad en 0 igual a v ($\gamma'_v(0) = v$), y W_p es un entorno estrellado de $0 \in T_pM$ tal que γ_v está definida en 1 para todo $v \in W_p$; asumiremos que W_p es maximal y, por tanto, único. Entonces, podemos tener la siguiente definición de aplicación exponencial en M:

$$\exp : W \subseteq TM \to M, \quad \exp(v) = \exp_p(v), \ \ \forall v \in T_pM \cap W, \forall p \in M,$$

donde $W = \bigcup_{p\in M} W_p$.

La aplicación exponencial nos permite proyectar un vector $v \in T_pM$ a un punto $\exp_p(v) \in M$ de la variedad.

Definición 5.19. Sea $p \in M, W_p$ un entorno de p donde $\exp_p$ es inyectiva y $U_p = \exp_p(W_p)$. Se define la **aplicación logarítmica** $\log_p : U_p \to W_p$ como la inversa de la exponencial.

5.2.3. Propiedades

1. Para cada conexión afín, la aplicación diferenciable

   $$(d\exp_p)_0 : T_0(T_pM) \to T_pM$$

   es igual, con las identificaciones usuales, a la identidad en T_pM. Por tanto, podemos encontrar un entorno estrellado del origen $\tilde{\mathcal{U}} \subset T_pM$ tal que la restricción de $\exp_p$ a $\tilde{\mathcal{U}}$ es un difeomorfismo en su imagen $\mathcal{U} := \exp_p(\tilde{\mathcal{U}})$. Si elegimos una base B_p de T_pM, obtenemos una carta $(\mathcal{U}, \psi)$ que lleva cualquier $q \in \mathcal{U}$ en las coordenadas de $\exp_p^{-1}(q)$ en la base B_p. A $(\mathcal{U}, \psi)$ se le llama **entorno normal** de p. Entonces, en coordenadas normales en p,

   $$\Gamma^k_{ij}(p) = 0, \quad \forall i, j, k \in 1, \ldots, n.$$

   Para una métrica pseudo-riemanniana, la base B_p se asume siempre ortonormal para g_p y, entonces,

   $$g_{ij}(p) = \epsilon_i\delta_{ij}, \text{ con } \epsilon_i = \pm 1,$$

donde δ_{ij} es el delta de Kronecker, que es igual a 1 si $i = j$ y 0 si $i \neq j$. Estas identidades en un principio se mantienen solamente en p. [O'N83]

2. Para todo $p \in M$ es posible encontrar un entorno convexo $\mathcal{W}$, esto es, un entorno normal de cada uno de sus puntos. Para todo par de puntos $p, q \in \mathcal{W}$ existe una única geodésica $\gamma_{pq} : [0,1] \to \mathcal{W}$ que los une y está contenida en $\mathcal{W}$. Además, la aplicación

$$\mathcal{W} \times \mathcal{W} \to TM, \quad (p,q) \to \gamma_{pq}(0),$$

es diferenciable.

3. Para toda conexión afín, podemos hablar de puntos críticos de la aplicación $\exp_p$ y por tanto, podemos definir puntos conjugados a lo largo de una geodésica. Sin embargo, las propiedades de los puntos conjugados en el caso pseudo-riemanniano difieren del riemanniano. Por ejemplo, los puntos conjugados a lo largo de una geodésica no tienen que ser aislados. [PTooa, MSo8, PToob]

4. En general, para cualquier variedad diferenciable no es posible referirse a un punto de corte, esto es, aquel punto a partir del cual la geodésica deja de ser la ruta más corta entre su punto de origen y cualquier otro punto a lo largo de la geodésica. Para una geodésica $\gamma_{p,v} : [0,b] \to M$ en una variedad riemanniana M, el punto $\gamma_{p,v}(t_1)$ será un punto de corte si existe algún $t_2 > t_1$ tal que cualquier otra curva entre p y $\gamma_{p,v}(t_2)$ tiene una longitud menor que la longitud de $\gamma_{p,v}$ en $[0,t_2]$.

   En el caso de variedades lorentzianas y para geodésicas temporales [BEE96] o luminosas [MSo8], se conservan propiedades análogas a las encontradas en las geodésicas riemannianas.

5. Se cumple el Lema de Gauss: si $p \in M$, $0 \neq x \in T_pM$ y $v_x, w_x \in T_x(T_pM)$ con v_x paralelo a x, entonces

$$g_p((d\exp_p)_x(v_x), (d\exp_p)_x(w_x)) = \langle v_x, w_x \rangle,$$

   donde $\langle \cdot, \cdot \rangle$ es el producto escalar en $T_x(T_pM)$ determinado por g_p.

5.2.4. Ejemplos

A continuación, mostremos algunos ejemplos de aplicación exponencial en distintas variedades.

Ejemplo 5.9. Veamos la aplicación exponencial en $\mathbb{R}^n$ con la métrica euclídea. Recordemos que las geodésicas en esta variedad son de la forma $\gamma_{p,v}(t) = p + tv$, donde p es un punto en $\mathbb{R}^n$ y v es un vector en $\mathbb{R}^n$.

La aplicación exponencial $\exp_p$ en un punto $p \in \mathbb{R}^n$ se define como:

$$\exp_p(v) = \gamma_v(1) = p + v,$$

donde $\gamma_v(t)$ es la geodésica con $\gamma_v(0) = p$ y $\gamma_v'(0) = v$.

En cambio, la aplicación logarítmica es la inversa de $\exp_p$:

$$\log_p(q) = q - p,$$

para cualquier punto $q \in \mathbb{R}^n$.

Ejemplo 5.10. En la esfera $\mathbb{S}^2$, vimos que las geodésicas se definen así:

$$\gamma_{p,v}(t) = \cos(\|v\|t)p + \sin(\|v\|t)\frac{v}{\|v\|},$$

con p es un punto de $\mathbb{S}^2$, $v \in T_p\mathbb{S}^2, v \neq 0$ y $\|v\|$ es su norma.

Por tanto, la aplicación exponencial en $p \in \mathbb{S}^2$ se define como sigue:

$$\exp_p(v) = \gamma_v(1) = \cos(\|v\|)p + \sin(\|v\|)\frac{v}{\|v\|},$$

donde $\gamma_v(t)$ es la geodésica con $\gamma_v(0) = p$ y $\gamma_v'(0) = v$.

Ejemplo 5.11. Sea el espacio de De Sitter:

$$\mathcal{Q}^1_{\alpha^2} = \left\{ \mathbf{x} \in \mathbb{R}^{d+1} : g(\mathbf{x},\mathbf{x})_L = -x_1^2 + \sum_{i=2}^{d+1} x_i^2 = \alpha^2 \right\},$$

vimos que un ejemplo de geodésica era:

$$\gamma_{p,v}(t) = \cosh\left(\frac{\|v\|t}{\alpha}\right) p + \alpha \sinh\left(\frac{\|v\|t}{\alpha}\right)\frac{v}{\|v\|},$$

donde p es un punto en $\mathcal{Q}^1_{\alpha^2}$, $v \in T_p\mathcal{Q}^1_{\alpha^2}$, $v \neq 0$, y $\|v\|$ es su norma con respecto a la métrica. Además, se verifica que $\gamma_{p,v}(0) = p$ y $\gamma'_{p,v}(0) = v$

Por tanto, se define la aplicación exponencial en un punto $p \in \mathcal{Q}^1_{\alpha^2}$ como:

$$\exp_p(v) = \gamma_{p,v}(1) = \cosh\left(\frac{\|v\|}{\alpha}\right) p + \alpha \sinh\left(\frac{\|v\|}{\alpha}\right)\frac{v}{\|v\|}.$$

6. Pseudo-hiperboloides

En este capítulo, exploramos los pseudo-hiperboloides, que son subconjuntos interesantes de $\mathbb{R}^d$ dotados de una métrica particular. Presentamos estas variedades regulares y diferenciables, calculamos las geodésicas y las aplicaciones exponencial y logarítmica asociadas.

6.1. Variedad regular y diferenciable

Comenzamos introduciendo la métrica de $\mathbb{R}^d_q$ con índice q, dada por:

$$g_q(\mathbf{x},\mathbf{y}) = \langle \mathbf{x},\mathbf{y}\rangle_q = -\sum_{i=1}^{q} x_i y_i + \sum_{i=q+1}^{d} x_i y_i,$$

que define una estructura pseudo-riemanniana en $\mathbb{R}^d$.

Definición 6.1. Los pseudo-hiperboloides son una familia de subvariedades en $\mathbb{R}^d$ definida como sigue:

$$\mathcal{Q}^q_\beta = \left\{ \mathbf{x} \in \mathbb{R}^d : g_q(\mathbf{x},\mathbf{x}) = \beta \neq 0 \right\},$$

donde $\beta \in \mathbb{R}$, si $\beta > 0$ se le conoce como pseudo-esfera y si $\beta < 0$ como pseudo-hiperboloide.

Veamos que los pseudo-hiperboloides son variedades regulares, para ello, hacemos uso de la Proposición 4.1:

Demostración. Podemos considerar $\mathcal{Q}^q_\beta$ como la imagen inversa en 0 de la función

$$F : \mathbb{R}^d \to \mathbb{R}, \quad F(p) = \|p\|^2_q - \beta.$$

La diferencial de F en un punto p, denotada como dF_p, está dada por

$$(dF)_p = (-2x_1, -2x_2, \ldots, -2x_q, 2x_{q+1}, \ldots, 2x_d).$$

Es claro que $dF_p = (0,\ldots,0)$ si, y sólo si, $p = (0,\ldots,0)$, indicando que el único punto crítico de F es el origen, y el único valor que no es regular es $F(0,\ldots,0) = -\beta$.
Por tanto, 0 es un valor regular de F, demostrando así que la familia está compuesta por variedades regulares. □

A continuación, como consecuencia de la Proposición 4.2 los pseudo-hiperboloides son variedades diferenciables. De hecho, toda variedad regular M^n en $\mathbb{R}^k, n \leq k$ es variedad diferenciable con atlas canónico

$$\mathcal{A}_u = \left\{ (X(U), X^{-1} : X(U) \to U) : (U, X) \text{ parametrización de } \mathcal{Q}^t_\beta \right\}.$$

Demostración. Construyamos una parametrización para los pseudo-hiperboloides dependiendo del signo de β:

- Caso $\beta > 0$. En este caso, el conjunto forma un hiperboloide de dos hojas. Podemos definir una parametrización usando coordenadas hiperbólicas para las primeras q coordenadas y esféricas para las restantes $d - q$ coordenadas:

$$X(u_1, \dots, u_{d-1}) = \Bigg(\sqrt{\beta + r^2} \sinh u_1, \dots, \sqrt{\beta + r^2} \sinh u_t, \\ \sqrt{r^2} \cos u_{q+1}, \dots, \sqrt{r^2} \prod_{i=q+2}^{d-1} \sin u_i \cos u_{q+i} \Bigg)$$

  donde $r^2 = \sum_{i=q+1}^{d} x_i^2 - \sum_{i=1}^{q} x_i^2$.

- Caso $\beta < 0$. Este caso corresponde a un hiperboloide de una hoja y la parametrización se puede dar de forma similar:

$$X(u_1, \dots, u_{d-1}) = \Bigg(\sqrt{-\beta + r^2} \cosh u_1, \dots, \sqrt{-\beta + r^2} \cosh u_q, \\ \sqrt{r^2} \cos u_{q+1}, \dots, \sqrt{r^2} \prod_{i=q+2}^{d-1} \sin u_i \cos u_{q+i} \Bigg)$$

□

6.1.1. Métrica inducida y plano tangente

La métrica en los pseudo-hiperboloides $\mathcal{Q}_\beta^q$ es la inducida por la métrica pseudo-riemanniana g_q de $\mathbb{R}^d$. La métrica inducida en cada punto $p \in \mathcal{Q}_\beta^q$ es la restricción de g_q al plano tangente en p.

El plano tangente $T_p\mathcal{Q}_\beta^q$ en un punto $p \in \mathcal{Q}_\beta^q$ se describe como:

$$T_p\mathcal{Q}_\beta^q = \{v \in \mathbb{R}^d : g_q(p, v) = 0\}.$$

Probamos que este plano tangente se identifica con un plano afín de $\mathbb{R}_q^d$ con normal el vector de posición. El plano afín de $\mathbb{R}_q^d$ que es tangente a $\mathcal{Q}_\beta^q$ se describe como $p + T_p\mathcal{Q}_\beta^q$. El vector p actúa como vector normal debido a que por definición, cualquier $v \in T_p\mathcal{Q}_\beta^q$ verifica que $g_q(p, v) = 0$. Sabemos que $g_q(p, p) = \beta$, por lo que podemos discutir la causalidad de este vector normal según el signo de β:

- Si $\beta > 0$ (pseudo-esfera), p es un vector espacial porque $g_q(p, p) > 0$.

- Si $\beta < 0$ (pseudo-hiperboloide), p es un vector temporal, ya que $g_q(p, p) < 0$.

6.2. Tensor de curvatura

6.2.1. Segunda forma fundamental y conexión de Levi-Civita

Para calcular la segunda forma fundamental y derivar la conexión de Levi-Civita en $\mathcal{Q}^q_\beta$, observamos que el vector normal unitario en p viene dado por:

$$\nu(p) = \frac{1}{\sqrt{|\beta|}} p.$$

La segunda forma fundamental II en un punto p para dos campos vectoriales tangentes X e Y se calcula como la proyección de la derivada covariante de Y en la dirección de X sobre el vector normal $\nu(p)$:

$$II(X, Y) = \langle \nabla_X Y, \nu(p) \rangle \nu(p).$$

Dado que en $\mathbb{R}^d_q$ la conexión Levi-Civita es la derivada usual porque la métrica g_q no depende de q (de forma análoga al E.j. 5.3), $\nabla_X Y$ simplifica a la derivada parcial de Y en la dirección de X. Por lo tanto, $\nabla_X Y$ será la proyección de $\frac{\partial Y}{\partial X}$ sobre el plano tangente a $\mathcal{Q}^q_\beta$. La componente de $\nabla_X Y$ perpendicular al plano tangente es precisamente $II(X, Y)$. Esta componente es la aceleración normal de Y cuando se mueve en la dirección de X, y se obtiene de la siguiente forma:

$$II(X, Y) = \frac{g_q(X, Y)}{\sqrt{|\beta|}} \nu(p).$$

La conexión de Levi-Civita en $\mathcal{Q}^q_\beta$ se obtiene restando la componente normal a la derivada usual:

$$(\nabla_X Y)_p = dY_p(X_p) - \frac{g_q(X, Y)}{|\beta|} p.$$

6.2.2. Tensor de curvatura

Aplicamos la ecuación de Gauss para obtener el tensor de curvatura:

$$g_q(R_{\mathcal{Q}^q_\beta}(V, W)X, Y) = g_q(II(V, X), II(W, Y)) - g_q(II(V, Y), II(W, X)).$$

Sustituyendo $II(V, W)$ en la ecuación anterior, obtenemos:

$$g_q(R_{\mathcal{Q}^q_\beta}(V, W)X, Y) = \frac{1}{\beta} \left(g_q(V, X) g_q(W, Y) - g_q(W, X) g_q(V, Y) \right).$$

Por tanto, el tensor de curvatura es:

$$R_{\mathcal{Q}^q_\beta}(V, W)X = \frac{1}{\beta} \left(g_q(V, X)W - g_q(W, X)V \right).$$

6.2.3. Curvatura seccional

Para finalizar esta sección, veamos que la curvatura seccional de cada plano es constante. Dado un plano generado por los vectores V y W en $T_p\mathcal{Q}^q_\beta$, la curvatura seccional $K(V, W)$ se

define como:

$$K(V,W) = \frac{g_q(R_{\mathcal{Q}^q_\beta}(V,W)V,W)}{g_q(V,V)g_q(W,W) - g_q(V,W)^2}.$$

Sustituyendo el tensor de curvatura calculado, obtenemos:

$$K(V,W) = \frac{\frac{1}{\beta}\, g_q(g_q(V,V)W - g_q(W,V)V,W)}{g_q(V,V)g_q(W,W) - g_q(V,W)^2} =$$

$$= \frac{\frac{1}{\beta}\left(g_q(V,V)g_q(W,W) - g_q(V,W)^2\right)}{g_q(V,V)g_q(W,W) - g_q(V,W)^2} = \frac{1}{\beta}.$$

Por lo tanto, la curvatura seccional de cada plano en $\mathcal{Q}^q_\beta$ es constante e igual a $\frac{1}{\beta}$.

6.3. Geodésicas y aplicación exponencial

En esta sección presentaremos las geodésicas que existen en los pseudo-hiperboloides y daremos la definición de las aplicaciones exponencial y logarítmica en este caso.

Comencemos por las geodésicas. Las geodésicas de $\mathcal{Q}_\beta$ son una combinación de los casos hiperbólicos, planos y esféricos. La naturaleza de la geodésica $\gamma_{x,v}$ depende del signo de $\langle v,v\rangle_q$. Para todo $t \in \mathbb{R}$, la geodésica $\gamma_{x,v}$ de $\mathcal{Q}^q_\beta$ con $\beta < 0$:

$$\gamma_{x,v}(t) = \begin{cases} \cosh\left(\frac{t\sqrt{\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) x + \frac{\sqrt{|\beta|}}{\sqrt{\langle v,v\rangle_q}} \sinh\left(\frac{t\sqrt{\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) v & \text{si } \langle v,v\rangle_q > 0 \\ x + tv & \text{si } \langle v,v\rangle_q = 0 \\ \cos\left(\frac{t\sqrt{-\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) x + \frac{\sqrt{|\beta|}}{\sqrt{-\langle v,v\rangle_q}} \sin\left(\frac{t\sqrt{-\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) v & \text{si } \langle v,v\rangle_q < 0 \end{cases}$$

Demostremos que es una geodésica en cada caso:

Caso 1: $\langle v,v\rangle_q > 0$.

$$\text{Primera derivada:} \quad \gamma'_{x,v}(t) = \sqrt{|\beta|} \sinh\left(\frac{t\sqrt{\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) x + \sqrt{\langle v,v\rangle_q} \cosh\left(\frac{t\sqrt{\langle v,v\rangle_q}}{\sqrt{|\beta|}}\right) v,$$

$$\text{Segunda derivada:} \quad \gamma''_{x,v}(t) = \langle v,v\rangle_q \gamma_{x,v}(t).$$

Caso 2: $\langle v,v\rangle_q = 0$.

$$\text{Primera derivada:} \quad \gamma'_{x,v}(t) = v,$$
$$\text{Segunda derivada:} \quad \gamma''_{x,v}(t) = 0.$$

Caso 3: $\langle v, v\rangle_q < 0$.

$$\text{Primera derivada:} \quad \gamma'_{x,v}(t) = -\sqrt{-\langle v, v\rangle_q}\,\sin\left(\frac{t\sqrt{-\langle v, v\rangle_q}}{\sqrt{|\beta|}}\right) x + \sqrt{|\beta|}\cos\left(\frac{t\sqrt{-\langle v, v\rangle_q}}{\sqrt{|\beta|}}\right)\frac{v}{\sqrt{-\langle v, v\rangle_q}},$$

$$\text{Segunda derivada:} \quad \gamma''_{x,v}(t) = -\langle v, v\rangle_q \gamma_{x,v}(t).$$

Como ya vimos anteriormente, x es ortogonal a cualquier vector en $T_x\mathcal{Q}^q_\beta$ bajo la métrica g_q, ya que $\langle x, y\rangle_q = 0$ para todo $y \in T_x\mathcal{Q}^q_\beta$.

Para los casos 1 y 3, $\gamma''_{x,v}(t)$ es proporcional a $\gamma_{x,v}$, esto es, la curva no tiene aceleración tangencial, lo que significa que $\nabla_{\gamma'_{x,v}}\gamma'_{p,v} = 0$. Por tanto, es una geodésica que pasa por x en $t = 0$ con velocidad v.

Para el caso 2, es una línea recta en la dirección del vector v. La aceleración es cero, lo que confirma que $\gamma_{x,v}(t) = x + tv$ es una geodésica sin aceleración tangencial, y que $\nabla_{\gamma'_{x,v}}\gamma'_{p,v} = 0$.

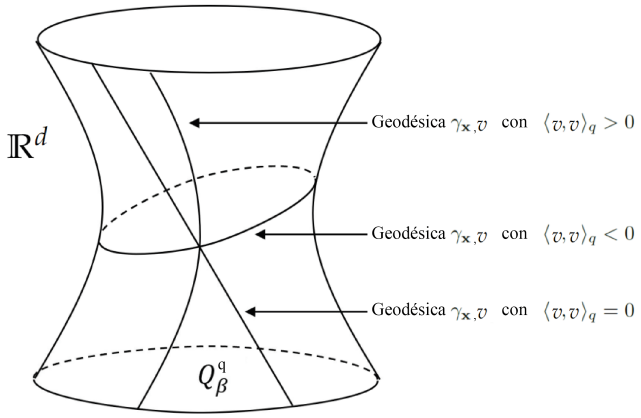


Figura 6.1.: Ilustración de los tres tipos de geodésicas de un pseudo-hiperboloide. Imagen modificada de [LS21].

En el pseudo-hiperboloide $\mathcal{Q}^q_\beta$ definimos la aplicación exponencial en un punto $x \in \mathcal{Q}^q_\beta$ como $\exp_x : T_x\mathcal{Q}^q_\beta \to M$, $\exp_x(v) = \gamma_{x,v}(1)$.

$$\exp_x(v) = \begin{cases} \cosh\left(\frac{\sqrt{\langle v,v\rangle}}{\sqrt{|\beta|}}\right) x + \frac{\sqrt{|\beta|}}{\sqrt{\langle v,v\rangle}}\sinh\left(\frac{\sqrt{\langle v,v\rangle}}{\sqrt{|\beta|}}\right) v & \text{si } \langle v, v\rangle > 0 \\ x + v & \text{si } \langle v, v\rangle = 0 \\ \cos\left(\frac{\sqrt{-\langle v,v\rangle}}{\sqrt{|\beta|}}\right) x + \frac{\sqrt{|\beta|}}{\sqrt{-\langle v,v\rangle}}\sin\left(\frac{\sqrt{-\langle v,v\rangle}}{\sqrt{|\beta|}}\right) v & \text{si } \langle v, v\rangle < 0 \end{cases}$$

Definimos ahora la aplicación logarítmica $\log_x : \mathcal{U}_x \to T_x\mathcal{Q}^q_\beta$ ($\log_x = \exp_x^{-1}$):

$$\log_x(y) = \begin{cases} \frac{\cosh^{-1}(\frac{\langle x,y\rangle_q}{\beta})}{\sqrt{(\frac{\langle x,y\rangle_q}{\beta})^2-1}}(y - \frac{\langle x,y\rangle_q}{\beta}x) & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} < -1 \\ y - x & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} = -1 \\ \frac{\cos^{-1}(\frac{\langle x,y\rangle_q}{\beta})}{\sqrt{1-(\frac{\langle x,y\rangle_q}{\beta})^2}}(y - \frac{\langle x,y\rangle_q}{\beta}x) & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} \in (-1,1) \end{cases}$$

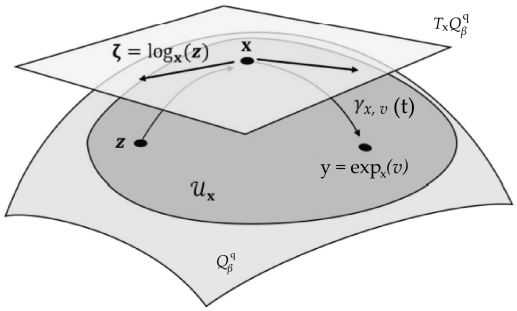


Figura 6.2.: Aplicación exponencial y logarítmica en el pseudo-hiperboloide. Imagen modificada de [LS21].

Para terminar este capítulo, veamos cuándo dos puntos $x, y \in \mathcal{Q}^q_\beta$ pueden ser conectados mediante una geodésica. La aplicación exponencial en el punto x se define como $\exp_x(v) = \gamma_{x,v}(1)$. Para que $y = \exp_x(v)$, debe ser $\log_x(y) = v$. Pero $\log_x(y)$ solamente está definido si $\frac{\langle x,y\rangle_q}{|\beta|} < 1$, es decir, si $\langle x, y\rangle_q < |\beta|$. Si $\langle x, y\rangle_q \geq |\beta|$, no existe un vector tangente v tal que $y = \exp_x(v)$, lo que implica que no hay una geodésica que conecta x e y.

El conjunto de todos los puntos y que pueden ser conectados a x mediante una geodésica se denomina el entorno normal de x, y se denota $\mathcal{U}_x$:

$$\mathcal{U}_x = \{y \in \mathcal{Q}^q_\beta : \langle x, y\rangle_t < |\beta|\}.$$

7. Distancia intrínseca en variedades pseudo-riemannianas

Este capítulo comenzará con la definición de la distancia geodésica en variedades riemannianas y se probará que cumple las propiedades de distancia para todo par de puntos de la variedad. En cambio, no será posible definir una distancia similar en las variedades pseudo-riemannianas, debido a la existencia de curvas luminosas. Por tanto, definiremos dos alternativas: la distancia lorentziana y la pseudo-distancia, a pesar de que ninguna de ellas satisface todas las propiedades para ser distancia.

La primera, como su nombre indica, se introduce para variedades lorentzianas. Por otro lado, daremos dos soluciones distintas para definir una pseudo-distancia para los pseudo-hiperboloides. Las referencias consultadas para este capítulo han sido [O'N83] y [KS24a]

7.1. Distancia geodésica en variedades riemannianas

Definición 7.1. Sea $\mathcal{U}$ un entorno normal de $x \in M$ con M una variedad diferenciable con métrica g. La función **radio** $r_x : \mathcal{U}_x \to \mathbb{R}$ se define como:

$$r_x(y) = \| \exp_x^{-1}(y) \|_g = \| \log_x(y) \|_g,$$

donde $\| \cdot \|_g$ es la norma asociada a la métríca g en x.

Lema 7.1. *Si $\sigma_{x,v}$ es la geodésica desde x a $y \in \mathcal{U}_x$ con $v = \log_x(y)$, entonces la longitud de arco de $\sigma_{x,v}$ es igual a $r_x(y)$.*

Demostración. Si $\sigma'_{x,v}(0) = v$, entonces sabemos que $v = \exp_p^{-1}(p)$. Y como $|\sigma'|$ es constante,

$$L(\sigma_{x,v}) = \int_0^1 \|\sigma'(s)\|_g ds = \int_0^1 \|v\|_g ds = \|v\|_g = r_x(p)$$

□

En esta sección, consideramos M una variedad riemanniana, donde su métrica g dota a cada espacio tangente con un producto escalar asociado. Definimos la norma en T_pM como $\tilde{r}(v) = \|v\|_g = \langle v, v \rangle_g^{1/2}$, entonces en cualquier entorno normal $\mathcal{U}_p$ de p la función radio $r = \tilde{r} \circ \exp_p^{-1}$ es diferenciable en todo punto excepto en p. Sea $\tilde{q}$ la función que asigna a cada vector en T_pM su norma al cuadrado, es decir, $\tilde{q}(v) = \|v\|_g^2$. Si $q = \tilde{q} \circ \exp_p^{-1}$ en $\mathcal{U}_p$, entonces $q(y) = \| \exp_p^{-1}(y) \|_g^2$. Sea P el campo de posición vectorial local cerca de p, esto es, $P(y) = \exp_p^{-1}(y)$ para $y \in \mathcal{U}_p$, entonces

$$\|P\|_g = \langle P, P \rangle_g^{1/2} = \sqrt{q} = r.$$

Por tanto, definimos $U = P/r$ como el campo vectorial unitario en $\mathcal{U}_p \setminus \{p\}$. Además, el gradiente de r es U en $\mathcal{U}_p \setminus \{p\}$. [O'N83, Cap. 5]

Lema 7.2. *Sea $\mathcal{U}_p$ un entorno normal de un punto $p \in M$. Si $q \in \mathcal{U}_p$, entonces la geodésica $\sigma : [0,1] \to \mathcal{U}_p$ desde p a q es la única curva más corta en desde p hasta q en $\mathcal{U}_p$.*

Demostración. Por el Lema 5.1 la unicidad de σ será salvo reparametrizaciones monótonas. Sea $\alpha : [0,b] \to \mathcal{U}_p$ una curva desde p hasta q. Primero, probaremos que:

$$L(\alpha) \geq L(\sigma)$$

y después veremos que si se da la igualdad, entonces α es una reparametrización monótona de σ.

Descomponemos α' en componentes paralelas y perpendiculares a U, el campo vectorial unitario:

$$\alpha' = \langle \alpha', U\rangle_g U + N,$$

donde N es el campo vectorial en α ortogonal a U. Para $t = 0$, tomamos $U = \alpha'(0)$ y $N = 0$. Entonces

$$\|\alpha'\|_g = [\langle \alpha', U\rangle_g^2 + \langle N, N\rangle_g]^{1/2} \geq |\langle \alpha', U\rangle_g|.$$

Para demostrar que U es el gradiente de r, consideramos que r está definida como $r = \tilde{r} \circ \exp_p^{-1}$. El gradiente de r, ∇r, en cualquier punto $x \neq p$ dentro de $\mathcal{U}_p$, apunta en la dirección del incremento máximo de r. Como $\exp_p$ lleva las curvas de T_pM a geodésicas de M, y ∇r en M es tangente a estas geodésicas. Por definición, $U = \frac{P}{r}$, donde P es la posición vectorial desde p a un punto en $\mathcal{U}_p$, es precisamente este vector unitario tangente a la geodésica, apuntando en la dirección de aumento de r.

Por ser U el gradiente de r, tenemos que $\langle \alpha', U\rangle_g = \frac{d(r\circ\alpha)}{dt}$, por el Teorema Fundamental del Cálculo:

$$L(\alpha) = \int_0^b \|\alpha'(t)\|_g dt \geq r(\alpha(b)) - r(\alpha(0)) = r(q),$$

y por el Lema 7.1, $r(q) = L(\sigma)$.

Supongamos ahora que $L(\alpha) = L(\sigma)$, entonces las desigualdades anteriores se vuelven igualdades, implicando:

$$N = 0, \quad \text{y} \quad \frac{dr}{dt} = \langle \alpha', U\rangle_g = |\langle \alpha', U\rangle_g| \geq 0.$$

Por lo que $\alpha' = \frac{dr}{dt} U$, probando que α es monótona a lo largo de la geodésica σ desde p a q. De hecho, $\alpha(t) = \sigma(\frac{r(t)}{r(b)})$

□

Definición 7.2. Para todo par de puntos p y q de una variedad riemanniana conexa M, definimos la **distancia riemanniana** desde p hasta q como

$$\mathrm{d}(p,q) = \text{ínfimo}\{L(\alpha) : \alpha \in \Omega(p,q)\},$$

donde $\Omega(p,q)$ es el conjunto de todas las curvas diferenciables por partes desde p hasta q en M.

Sea un punto $q \in M$ y $\epsilon > 0$, al conjunto $\mathcal{N}_\epsilon(q)$ de puntos $p \in M$ tales que $\mathrm{d}(q,p) < \epsilon$, se le conoce como **entorno de radio** ϵ de q en M. Con estas nuevas nociones, podemos fortalecer el lema anterior.

Proposición 7.1. *Dado un punto $p \in M$.*

1. *Para $\epsilon > 0$ suficientemente pequeño, $\mathcal{N}_\epsilon(q)$ es normal, esto es, cada punto p en $\mathcal{N}_\epsilon(q)$ puede ser alcanzado desde q mediante una única geodésica que es mínima dentro de este entorno.*

2. *Si es $\mathcal{N}_\epsilon(q)$ normal, la geodésica σ de q a $p \in \mathcal{N}_\epsilon(q)$ es la única curva más corta en M de q a p. En particular,*
$$L(\sigma) = r(p) = d(q, p).$$

La distancia introducida, d, cumple las propiedades de distancia:

Proposición 7.2. *Sea d la distancia riemanniana de una variedad riemanniana conexa M, $d : M \times M \to \mathbb{R}$, para todo $p, q, r \in M$ cumple:*

1. $d(p, q) \geq 0$, *y* $d(p, q) = 0$ *si, y solo si,* $p = q$.

2. $d(p, q) = d(q, p)$

3. $d(p, q) + d(p, r) \geq d(p, r)$

Además, d es compatible con la topología de M.

Demostración. Las primeras dos propiedades son triviales, excepto que $\mathrm{d}(p, q) = 0 \implies p = q$. Si $p \neq q$, entonces ya que M es Hausdorff, existe un entorno normal U_p de p que no contiene a q. La prueba de la anterior proposición muestra que U_p contiene un entorno de p de radio ϵ, por lo que $\mathrm{d}(p, q) \geq \epsilon > 0$.

Para probar la desigualdad triangular, sea $\epsilon > 0$, elegimos $\alpha \in \Omega(p, q)$ y $\beta \in \Omega(q, r)$ tal que
$$L(\alpha) < \mathrm{d}(p, q) + \epsilon \quad \text{y} \quad L(\beta) < \mathrm{d}(q, r) + \epsilon.$$
Concatenando α y β obtenemos una curva $\gamma \in \Omega(p, r)$ que cumple
$$\mathrm{d}(p, r) \leq L(\gamma) = L(\alpha) + L(\beta) < \mathrm{d}(p, q) + \mathrm{d}(q, r) + 2\epsilon,$$
y como ϵ es arbitrario el resultado se cumple.

Como todo entorno de un punto de M contiene un entorno de radio ϵ, y al ser estos últimos abiertos de M, la métrica M es compatible con la topología de M, esto es, un subconjunto $\mathcal{V}$ de M es abierto si, y solo si, para cada $p \in \mathcal{V}$ existe un $\epsilon > 0$ tal que $\mathcal{N}_\epsilon(p) \subset \mathcal{V}$. □

La compatibilidad de d significa que los entornos de radio ϵ pueden verse de forma similar que en un espacio euclídeo. Por ejemplo, una sucesión de puntos $\{p_i\}$ en M converge a $p \in M$ si, y solo si la sucesión $\{\mathrm{d}(p, p_i)\}$ converge a $0 \in \mathbb{R}$.

Por definición de la distancia riemanniana, una curva σ de p a q en M es la curva más corta que une p y q si, y solo si, $L(\sigma) = \mathrm{d}(p, q)$. En caso de que exista dicha curva, diremos que σ minimiza la longitud de arco desde p hasta q, o simplemente que σ es **minimizante**.

Corolario 7.1. *En una variedad riemanniana, una curva minimizante α de p a q es una reparametrización monótona de una geodésica (no rota) desde p hasta q.*

Demostración. Podemos descomponer el dominio I de α en subintervalos I_i tal que $\alpha_i = \alpha|_{I_i}$ cae en un conjunto anierto convexo, además podemos asumir que α_i no es constante, pues si lo fuera podríamos unir I_i con un subintervalo adyacente. Al ser α_i minimizante y por la

unicidad del *Lema* 5.3, es una reparametrización monótona de una geodésica σ_i con velocidad unitaria. La unión de las distintas σ_i es una geodésica rota σ desde p a q. De forma análoga, podemos juntar las reparametrizaciones para que α sea una reparametrización monótona de σ.

Ya que $L(\sigma) = L(\alpha) = \mathrm{d}(p,q)$, el siguiente hecho implicará que σ no es rota: Si una geodésica γ_1 que termina en p y γ_2 una geodésica que parte de p se concatenan formando una curva minimizante γ, entonces γ es una geodésica (no rota). Si asumimos que γ tiene velocidad constante y sea $\mathcal{C}$ un entorno convexo de p. Entonces una curva inicial γ_2 y una final γ_1 se combinan dando una curva minimizante $\tilde{\gamma}$ en $\mathcal{C}$. Al ser $\mathcal{C}$ un entorno normal del punto inicial de $\tilde{\gamma}$, aplicando el *Lema* 5.3 obtenemos que $\tilde{\gamma}$ es una reparametrización con velocidad constante de una geodésica. Consecuentemente, $\tilde{\gamma}$ y γ son geodésicas no rotas. □

7.2. Distancia en variedades pseudo-riemannianas

7.2.1. Distancia lorentziana

En variedades pseudo-riemannianas, especialmente aquellas con una métrica de Lorentz, la noción de distancia no es directa debido a la presencia de curvas luminosas. En este contexto, se define una distancia lorentziana.

Definición 7.3. En una variedad lorentziana, definimos el **intervalo causal** $J^+(x)$ de un punto x como el conjunto de todos los puntos y tales que existe una curva causal desde x a y. De forma similar, el **intervalo cronológico** $I^+(x)$ es el conjunto de todos los puntos y tales que existe una curva temporal desde x a y.

Definición 7.4. Sea $\gamma : [0,1] \to M$ una curva causal en una variedad lorentziana con métrica g. El **tiempo propio** $\tau(\gamma)$ de la curva γ se define como:

$$\tau(\gamma) = \int_0^1 \sqrt{-g(\gamma'(t), \gamma'(t))}\, dt,$$

donde $\gamma'(t)$ es el vector tangente a la curva en el punto $\gamma(t)$.

La **distancia lorentziana** $d_L(x,y)$ se define como:

$$\mathrm{d}_L(x,y) = \begin{cases} \sup\{\tau(\gamma) : \gamma \text{ es una curva causal de } x \text{ a } y\} & \text{si } y \in J^+(x) \\ 0 & \text{si } y \notin J^+(x) \end{cases}$$

donde $\tau(\gamma)$ es el tiempo propio de la curva causal γ.

Observación 7.1. La distancia lorentziana $\mathrm{d}_L(x,y)$ no cumple las mismas propiedades que la distancia riemanniana. En particular, no siempre satisface la desigualdad triangular y puede ser no simétrica.

7.2.2. Pseudo-distancia en los pseudo-hiperboloides

Para la familia de estudio de este trabajo, definimos la pseudo-distancia geodésica entre $x \in \mathcal{Q}_\beta^q$ e $y \in \mathcal{U}_x$ como la longitud de arco de $\sigma_{x,\log_x(y)}$:

$$\mathrm{d}_\gamma(x,y) = \sqrt{|\log_x(y)|_q^2} = \begin{cases} \sqrt{|\beta|}\cosh^{-1}\left(\frac{\langle x,y\rangle_q}{\beta}\right) & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} < -1 \\ 0 & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} = -1 \\ \sqrt{|\beta|}\cos^{-1}\left(\frac{\langle x,y\rangle_q}{\beta}\right) & \text{si } \frac{\langle x,y\rangle_q}{|\beta|} \in (-1,1) \end{cases}$$

A pesar de cumplir que $\mathrm{d}_\gamma(x,y) = \mathrm{d}_\gamma(y,x) \geq 0$ y $\mathrm{d}_\gamma(x,x) = 0$, no es una distancia métrica. Por ejemplo, pueden existir puntos $x,y,z \in M$ tales que $\mathrm{d}_\gamma(x,y) = \mathrm{d}_\gamma(x,z) = 0$ pero $\mathrm{d}_\gamma(y,z) > 0$, esto es el caso para $x = (1,0,0,0)^\intercal, y = (1,1,1,0)^\intercal, z = (1,1,0,1)^\intercal \in \mathcal{Q}_{-1}^1$. A pesar de esto, estas dos propiedades son suficientes para cuantificar la noción de cercanía en una bola de radio ϵ centrada en x: $B_q^\epsilon = \{y : \mathrm{d}_\gamma(x,y) < \epsilon\}$.

Debido a que la aplicación logarítmica no está definida si $\langle x,y\rangle_q \geq |\beta|$, se proponen dos soluciones distintas.

Definición 7.5. En primer lugar, en [LS21], se define la **pseudo-distancia geodésica** entre $x \in \mathcal{Q}_\beta^q$ e $y \in \mathcal{Q}_\beta^q$ como:

$$\mathrm{D}_\gamma(x,y) = \begin{cases} \mathrm{d}_\gamma(x,y) & \text{si } \langle x,y\rangle_q \leq 0, \\ \sqrt{|\beta|}\left(\frac{\pi}{2} + \frac{\langle x,y\rangle_q}{|\beta|}\right) & \text{si } \langle x,y\rangle_q > 0. \end{cases}$$

Haciendo uso del siguiente teorema, en [XZP+22a] se define otra pseudo-distancia diferente.

Teorema 7.1. *Para todo punto $x \in \mathcal{Q}_\beta^q$, la unión del entorno normal de x y del entorno normal de su punto antípoda $-x$ es toda la variedad.*

$$\mathcal{U}_x \cup \mathcal{U}_{-x} = \mathcal{Q}_\beta^q$$

Demostración. Sean dos puntos cualquiera x, y tales que $x \in \mathcal{Q}_\beta^q$, $y \notin \mathcal{U}_x$. Por la definición de entorno normal, $\langle x,y\rangle_q \geq |\beta|$ lo que implica que $\langle -x,y\rangle_q \leq |\beta|$. Por tanto, $y \in \mathcal{U}_{-x}$, y se tiene que la unión de estos entornos es toda la variedad. □

Definición 7.6. En [XZP+22a], se define la pseudo-distancia geodésica entre $x \in \mathcal{Q}_\beta^q$ e $y \in \mathcal{Q}_\beta^q$ como:

$$\mathrm{D}_\gamma(x,y) = \begin{cases} \mathrm{d}_\gamma(x,y) = \sqrt{\|\log_x(y)\|_q^2} & \text{si } \langle x,y\rangle_q < |\beta|, \\ \pi\sqrt{|\beta|} + \mathrm{d}_\gamma(x,-y) = \pi\sqrt{|\beta|} + \sqrt{\|\log_x(-y)\|_q^2} & \text{si } \langle x,y\rangle_q \geq |\beta|. \end{cases}$$

La intuición tras esta definición es que cuando $\log_x(y)$ no está definida, es decir, cuando no existe ninguna geodésica que une x e y, entonces definimos la distancia como $\mathrm{d}_\gamma(x,y) = \mathrm{d}_\gamma(x,-x) + \mathrm{d}_\gamma(-x,y)$ o $\mathrm{d}_\gamma(x,y) = \mathrm{d}_\gamma(x,-y) + \mathrm{d}_\gamma(-y,y)$. Dado que $\mathrm{d}_\gamma(x,-x) = \mathrm{d}_\gamma(-y,y) = \pi\sqrt{|\beta|}$ es constante y usando que $\mathrm{d}_\gamma(-x,y) = \mathrm{d}_\gamma(x,-y)$, entonces tenemos que:

$$\mathrm{D}_\gamma(x,y) = \pi\sqrt{|\beta|} + \mathrm{d}_\gamma(x,-y) = \pi\sqrt{|\beta|} + \sqrt{\|\log_x(-y)\|_q^2}.$$

8. Completitud y conectividad geodésica

En este capítulo vamos a introducir y explorar los conceptos de completitud y conectividad geodésica en el contexto de las variedades. Estos conceptos son fundamentales en la geometría diferencial y la teoría de las variedades, ya que describen propiedades sobre cómo se pueden extender las geodésicas y cómo se pueden conectar puntos dentro de una variedad mediante estas curvas.

En las siguientes secciones, formalizaremos estas definiciones y examinaremos sus implicaciones a través del Teorema de Hopf-Rinow, que establece una equivalencia entre la completitud y la conectividad geodésica en el contexto riemanniano. Además, veremos que esta equivalencia no se cumple en el caso pseudo-riemanniano.

Definición 8.1. Una variedad es **geodésicamente completa** si toda geodésica maximal está definida en toda la recta real, $\mathbb{R}$.

Definición 8.2. Diremos que una variedad es **geodésicamente conexa** si, y sólo sí, todo par de puntos pueden ser conectados por una geodésica.

8.1. Teorema de Hopf-Rinow

En el caso riemanniano, los conceptos de completitud y conexión geodésica son equivalentes. Sin embargo, como veremos más adelante, el pseudo-hiperboloide es geodésicamente completo pero no es conexo.

Teorema 8.1. *(Hopf-Rinow). Sea M una variedad riemanniana conexa. Los siguientes enunciados son equivalentes:*

1. *M es completo como espacio métrico con la distancia riemanniana.*
2. *M es geodésicamente completo.*
3. *Existe un punto $p \in M$ tal que el dominio de la aplicación exponencial es T_pM. Lo que implica que existe una geodésica que parte de p para cada vector de dirección v, indicando que M es geodésicamente conexa.*
4. *Un subconjunto de M es compacto si, y sólo si, es cerrado y acotado.*

Para el desarrollo de la demostración del teorema se ha seguido [dS16], necesitamos previamente dos lemas.

Lema 8.1. *Sean $\gamma_1 : [a, b] \to M$ una geodésica de p a q, $\gamma_2 : [b, c] \to M$ una geodésica de q a r, y supongamos que γ_1 y γ_2 tienen igual velocidad. Si la curva $\gamma : [a, c] \to M$ obtenida concatenando γ_1 y γ_2 tiene longitud $L(\gamma) = d(p, r)$, entonces γ es una geodésica.*

Lema 8.2. *Si existe $p \in M$ tal que $W_p = T_pM$, es decir, el dominio de la exponencial en p es T_pM; entonces para cualquier $q \in M$ existe un segmento de geodésica minimizante de p a q.*

Procedemos ahora a demostrar el Teorema.

Demostración. I $\implies$ II. Es suficiente con mostrar que una geodésica con velocidad unidad $\gamma : [0, b) \to M$ es geodésicamente extendible. Sea (t_n) una sucesión en $[0, b)$ tal que $t_n \to b$. Entonces $\gamma(t_n)$ es una sucesión de Cauchy ya que $d(\gamma(t_n), \gamma(t_m)) \leq |t_n - t_m|$, luego converge a un punto p. Si (s_n) es otra sucesión cualquiera en $[0, b)$ tal que $s_n \to b$ entonces $\gamma(s_n)$ converge a p porque $d(\gamma(t_n), \gamma(s_n)) \leq |t_n - s_n|$. Por tanto, definiendo $\gamma(b) = p$, es una extensión continua de γ, luego γ es geodésicamente extendible por el Corolario 5.2.

II $\implies$ III. Para todo $v \in T_pM$, la geodésica maximal γ_v está definida en $\mathbb{R}$. En particular, está definida en 1, luego $v \in W_p$.

III $\implies$ IV. Ya que M es un espacio métrico, cualquier compacto es cerrado y acotado. Por contra, sea $A \subset M$ cerrado y acotado. Por el lema previo, para cada $q \in A$ existe un segmento de geodésica minimizante $\sigma_q : [0, 1] \to M$ de p a q. Por ser A acotado, los valores $|\sigma_q'(0)| = L(\sigma_q) = d(p, q)$ están acotados por la desigualdad triangular, sea $R \geq d(p, q)$ para cada $q \in A$. Entonces cada $\sigma_q'(0)$ está contenido en una bola compacta $B_R = \{v \in T_pM : |v| \leq R\}$. Si $q \in A$, entonces $\exp_p(\sigma_q'(0)) = q$, luego $A \subset \exp_p(B_R)$. Ya que $\exp_p(B_R)$ es compacto y A es cerrado, esto implica que A es compacto.

IV $\implies$ I. Sea (x_n) una sucesión de Cauchy en M. El conjunto $\{x_n\}$ está acotado, por lo que su clausura es compacta. Por tanto, (x_n) tiene una subsucesión convergente y al ser (x_n) de Cauchy, debe converger al límite de la subsucesión. □

Corolario 8.1. *Si M es una variedad riemanniana conexa y completa, entonces existe una geodésica minimizante entre dos puntos cualesquiera de M.*

Este corolario se demuestra a partir de la completitud geodésica y del Lema 8.2.

Demostración. Dado que M es una variedad riemanniana conexa y completa, el Teorema de Hopf-Rinow garantiza que M es geodésicamente completa, lo que significa que toda geodésica maximal en M está definida en $\mathbb{R}$, esto es, toda geodésica puede extenderse indefinidamente. Además, el Teorema también garantiza que M es geodésicamente conexa, lo que implica que para cualquier par de puntos en M, existe al menos una geodésica que los conecta. Por el Lema 8.2, desde cualquier punto $p \in M$ hacia cualquier otro punto $q \in M$, la geodésica que los une es minimizante. □

8.2. Caso pseudo-riemanniano

En los pseudo-hiperboloides, como ya vimos en la Sección 6.3, existe una geodésica que une dos puntos $x, y \in \mathcal{Q}^q_\beta$ si y solo si, $\langle x, y\rangle_q < |\beta|$. Esto implica que pueden existir parejas de puntos que no pueden ser conectadas por una geodésica, lo que conllevará a que los pseudo-hiperboloides no sean geodésicamente completos. Veámoslo con un ejemplo que se ha seguido de [KS24a].

Ejemplo 8.1. El espacio De Sitter $(d-1)$-dimensional, $\mathcal{Q}^1_{\alpha^2} = \left\{x \in \mathbb{R}^d : -x_1^2 + \sum_{i=2}^d x_i^2 = \alpha^2\right\}$, también contradice el Teorema de Hopf-Rinow. En particular, este espacio es geodésicamente completo pero no es geodésicamente conexo.

Vamos a ver en primer lugar sus geodésicas. Consideremos el punto $p \in \mathcal{Q}^1_{\alpha^2}$ y $v \in T_p\mathcal{Q}^1_{\alpha^2} \setminus \{0\}$. Y sea el plano $E \in \mathbb{R}^d$, $E := L\{p, v\}$.

Si v es espacial o temporal, entonces E es no degenerado (al ser p espacial) y por tanto $\mathbb{R}^d = E \oplus E^\perp$.

Si v es espacial, entonces $\langle \cdot, \cdot \rangle_1|_E$ es definido positivo. Por tanto,

$$E \cap \mathcal{Q}^1_{\alpha^2} = \{y \in E : \langle y, y \rangle_1 = \alpha^2\},$$

es una elipse, ver Figura 8.1.

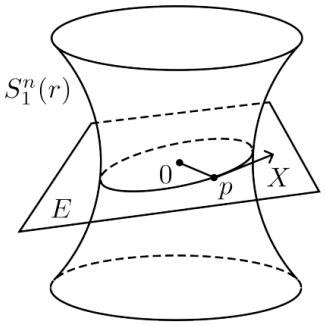


Figura 8.1.: Visualización de $E \cap \mathcal{Q}^1_{\alpha^2}$ cuando v es espacial. Imagen obtenida de [KS24a].

Si v es luminoso, entonces $\langle \cdot, \cdot \rangle_1|_E$ es semidefinido positivo y degenerado. En este caso, $E \cap \mathcal{Q}^1_{\alpha^2}$ es un par de rectas paralelas:

$$E \cap \mathcal{Q}^1_{\alpha^2} = \left\{\lambda p + \beta v : \langle \lambda p + \beta v, \lambda p + \beta v \rangle_1 = \alpha^2\right\}$$

$$= \left\{\lambda p + \beta v : \lambda^2 \langle p, p \rangle_1 + 2\lambda\beta \langle p, v \rangle_1 + \beta^2 \langle v, v \rangle_1 = \alpha^2\right\}$$

Dado que $\langle p, p \rangle_1 = \alpha^2$, $\langle p, v \rangle_1 = 0$ (con $p \perp v$) y $\langle v, v \rangle_1 = 0$ (con v luminoso), obtenemos:

$$E \cap \mathcal{Q}^1_{\alpha^2} = \left\{\lambda p + \beta v : \lambda^2 = 1, \beta \in \mathbb{R}\right\} = \{\pm p + \beta v : \beta \in \mathbb{R}\}$$

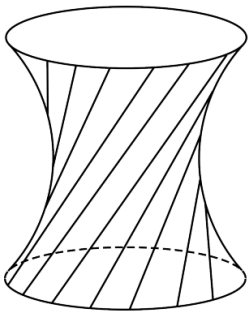


Figura 8.2.: Visualización de las geodésicas cuando v es luminoso. Imagen obtenida de [KS24a].

Si v es temporal, entonces $\langle \cdot, \cdot \rangle_1|_E$ es indefinida y no degenerada. En este caso, $E \cap \mathcal{Q}^1_{\alpha^2}$ es

una hipérbola con dos componentes conexas:

$$E \cap \mathcal{Q}^1_{\alpha^2} = \left\{ \lambda p + \beta v : \lambda^2 \alpha^2 + \beta^2 \langle v, v \rangle_1 = \alpha^2 \right\} = \left\{ \lambda p + \beta v : \lambda^2 + \frac{\langle v, v \rangle_1}{\alpha^2} \beta^2 \right\},$$

que es la ecuación de una hipérbola ya que $\langle v, v \rangle_1 < 0$. Ver Figura 8.3.

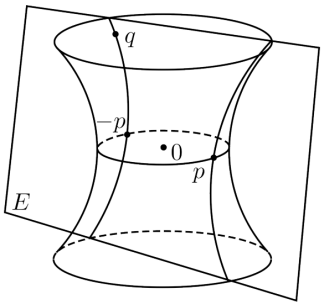


Figura 8.3.: Visualización de $E \cap \mathcal{Q}^1_{\alpha^2}$ cuando v es temporal. Imagen obtenida de [KS24a].

Concluimos que $\mathcal{Q}^1_{\alpha^2}$ es geodésicamente completo porque $\exp_p$ está definida en todo T_pM para todo $p \in M$.

Fijamos un punto $p \in \mathcal{Q}^1_{\alpha^2}$ y examinamos qué puntos $q \in \mathcal{Q}^1_{\alpha^2}$ pueden ser alcanzados por una geodésica que parte de p:

- Si $q = p$ o $q = -p$ entonces la conexión es trivial pues existen infinitos planos que contienen a p y a $\pm p$.
- Si $q \neq \pm p$ entonces p y q son linealmente independientes y existe un único plano E que los contiene. Si este plano no es espacial y p y q están en componentes conexas distintas de $E \cap \mathcal{Q}^1_{\alpha^2}$, entonces no existe una geodésica que los conecte.

Además, todos los puntos q que pueden ser alcanzados desde $-p$ por una geodésica causal, no pueden ser alcanzados desde p. El conjunto de estos q se define como:

$$\{q \in \mathcal{Q}^1_{\alpha^2} : \langle q + p, p \rangle \leq 0 \text{ y } q \neq p\}.$$

Tomando una base ortonormal $\{e_0, e_1\}$ en $E = L\{p, q\}$, entonces $E \cap \mathcal{Q}^1_{\alpha^2}$ consiste en dos ramas de la hipérbole $-(x_0)^2 + (x_1)^2 = \alpha^2$. Si $p = p_0 e_0 + p_1 e_1$, entonces $-p_0^2 + p_1^2 = \alpha^2$ y $-q_0^2 + q_1^2 = \alpha^2$ y por tanto podemos reparametrizar p y q como

$$p = \alpha(\sinh(t), \cosh(t))$$

$$q = \alpha(\sinh(s), \pm\cosh(s)).$$

Y estarán en diferentes ramas si, y solo si, $q = \alpha(\sinh(s), -\cosh(s))$. Entonces

$$\langle p, q \rangle = -\alpha^2 \sin(t)\sin(s) - \alpha^2 \cosh(t)\cosh(s) = -\alpha^2 \cosh(t+s) \leq -\alpha^2.$$

En cambio, estarán en la misma rama si, y solo si, $q = \alpha(\sinh(s), \cosh(s))$, por lo que

$$\langle p, q \rangle = \alpha^2 \cosh(t - s) \geq \alpha^2.$$

Por tanto, p y q están en diferentes ramas si, y solo si, $\langle p, q\rangle \leq -\alpha^2$, que ocurre si, y solo si, $\langle p+q, p\rangle \leq 0$.

9. Métodos de optimización en geometría pseudo-riemanniana

En este capítulo introducimos los dos métodos presentados en [LS21], utilizados para la optimización en geometría pseudo-riemanniana. El primero de ellos, es una extensión de la optimización euclídea, en el que se utiliza el descenso del gradiente de forma usual y proyecta los resultados sobre el pseudo-hiperboloide. El descenso del gradiente es el método más extendido en la optimización de modelos de aprendizaje profundo (véase la Subsección 1.3.1 para más detalles). Este método es un algoritmo iterativo de primer orden que busca minimizar una función de pérdida $\mathcal{L}$. En cada iteración, se actualizan los parámetros del modelo moviéndolos en la dirección opuesta al gradiente de la función de pérdida con respecto a los parámetros anteriores. El segundo método presentado en este capítulo tiene especial interés, ya que introduce una nueva dirección de descenso que no se sale del pseudo-hiperboloide, esto lo consigue por medio de la aplicación exponencial.

9.1. Optimización euclídea

El primer método presentado lleva representaciones euclídeas en $\mathbb{R}^d$ al pseudo-hiperboloide $\mathcal{Q}^q_\beta$, y se usa la regla de la cadena para llevar a cabo el descenso del gradiente estándar.

Sea $\mathcal{S}^{q-1} = \{x \in \mathbb{R}^q : \|x\| = 1\}$ la $(q-1)$-esfera unidad. Introducimos los siguientes difeomorfismos:

Proposición 9.1. *Para cualquier $\beta < 0$, existe un difeomorfismo $\psi : \mathcal{Q}^q_\beta \to \mathcal{S}^{q-1} \times \mathbb{R}^{d-q}$. Sea $x = \begin{pmatrix} t \\ s \end{pmatrix} \in \mathcal{Q}^q_\beta$ con $t \in \mathbb{R}^q \setminus \{0\}$ y $s \in \mathbb{R}^{d-q}$, y sea $z = \begin{pmatrix} u \\ v \end{pmatrix} \in \mathcal{S}^{q-1} \times \mathbb{R}^{d-q}$, donde $u \in \mathcal{S}^{q-1}$ y $v \in \mathbb{R}^{d-q}$. La aplicación ψ se define:*

$$\psi(x) = \begin{pmatrix} \frac{1}{\|t\|} t \\ \frac{1}{\sqrt{|\beta|}} s \end{pmatrix}$$

y su inversa ψ^{-1}:

$$\psi^{-1}(z) = \sqrt{|\beta|} \begin{pmatrix} \sqrt{1 + \|v\|^2} u \\ v \end{pmatrix}$$

Con estas aplicaciones, cualquier vector $x \in \mathbb{R}^q \setminus \{0\} \times \mathbb{R}^{d-q}$ puede llevarse a $\mathcal{Q}^q_\beta$ vía $\phi = \psi^{-1} \circ \psi$. ϕ es diferenciable en todo punto salvo cuando $x_1 = \ldots = x_q = 0$, que nunca debería ocurrir en la práctica.

9.2. Optimización pseudo-riemanniana

Introducimos ahora un método para optimizar cualquier función diferenciable $f : \mathcal{Q}_\beta^q \to \mathbb{R}$.

9.2.1. Gradiente pseudo-riemanniano

Como $x \in \mathcal{Q}_\beta^q$ también se encuentra en el espacio ambiente euclídeo $\mathbb{R}^d$, la función f tiene un gradiente euclídeo $\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_d}\right) \in \mathbb{R}^d$. Sea $\mathbf{G} = \mathbf{G}^{-1} = I_{q,d-q}$, la matriz diagonal con los primeros q elementos igual a -1 y los $d-q$ restantes igual a 1. Entonces, el gradiente de f en el espacio ambiente pseudo-euclídeo $\mathbb{R}_q^d$ es $(\mathbf{G}^{-1}\nabla f(x)) = (\mathbf{G}\nabla f(x)) \in \mathbb{R}^d$. Por ser $\mathcal{Q}_\beta^q$ una subvariedad de $\mathbb{R}_q^d$, el gradiente pseudo-rimenniano $Df(x) \in T_x\mathcal{Q}_\beta^q$ de f en $\mathcal{Q}_\beta^q$ es la proyección ortogonal Π_x de $\mathbf{G}\nabla f(x)$ en $T_x\mathcal{Q}_\beta^q$:

$$Df(x) = \Pi_x(\mathbf{G}\nabla f(x)) = \mathbf{G}\nabla f(x) - \frac{\langle \mathbf{G}\nabla f(x), x\rangle_q}{\langle x, x\rangle_q} x = \mathbf{G}\nabla f(x) - \frac{\langle \nabla f(x), x\rangle}{\langle x, x\rangle_q} x. \quad (9.1)$$

9.2.2. Optimización iterativa

El objetivo es decrementar el valor de f siguiendo alguna dirección de descenso. Ya que $\mathcal{Q}_\beta^q$ no es un espacio vectorial, no se puede seguir la dirección del descenso de la forma usual, añadiendo la dirección multiplicada por un tamaño de paso, porque esto resultaría en un punto que no necesariamente se encuentra en $\mathcal{Q}_\beta^q$. Por tanto, para mantener el nuevo punto en la variedad, se utiliza la aplicación exponencial. Dado un tamaño de paso $t > 0$, un paso de descenso a lo largo del vector tangente $v \in T_x\mathcal{Q}_\beta^q$ viene dado por:

$$y = \exp_x(tv) \in \mathcal{Q}_\beta^q.$$

9.2.3. Dirección del descenso

Cuando el dominio de la función a optimizar f es un espacio euclídeo con la métrica usual, el negativo del gradiente es una dirección de descenso. Sin embargo, esto no ocurre para las métricas pseudo-riemannianas.

Para todo $t \in \mathbb{R}$ y todo $v \in \mathcal{Q}_\beta^q$, se tiene que

$$\exp_x(tv) = \gamma_{x,tv}(1) = \gamma_{x,v}(t),$$

por lo que se puede fijar $t = 1$ y escalar v apropiadamente. Aplicando la aproximación de primer orden de Taylor, existe un vector tangente suficientemente pequeño $w \neq 0$, esto es, tal que $\exp_x(w)$ pertenece a un entorno convexo de x [GLY18], que satisface lo siguiente:

$$\gamma_{x,w}(0) = x \in \mathcal{Q}_\beta^q,$$

$$\gamma'_{x,w}(0) = w \in T_x\mathcal{Q}_\beta^q,$$

$$\gamma_{x,w}(1) = y \in \mathcal{Q}_\beta^q,$$

Además, usando que $\forall t, (f \circ \gamma)'(t) = df(\gamma'(t)) = g_{\gamma(t)}(Df(\gamma(t)), \gamma'(t))$ [O'N83], observamos que una aproximación en $t = 1$ de la función $f \circ \gamma_{x,w} : \mathbb{R} \to \mathbb{R}$ viene dada por:

$$f(y) = f \circ \gamma_{x,w}(1) \approx f \circ \gamma_{x,w}(0) + (f \circ \gamma_{x,w})'(0) = f(x) + \langle Df(x), w \rangle_q,$$

donde Df es la gradiente de f.

Para que la dirección de búsqueda w sea una dirección de descenso en x, tal que $f(y) < f(x)$, w debe satisfacer que $\langle Df(x), w \rangle_q < 0$. No obstante, elegir $w = -\eta Df(x)$, con η un tamaño de paso (positivo), puede aumentar el valor de la función si el producto escalar no es definido positivo. Si $d \geq 1$, entonces $\langle \cdot, \cdot \rangle_q$ es definido positivo si, y solo si, $q = 0$ y es definido negativo si, y solo si, $q = d$.

Una solución sencilla sería elegir el signo de $w = \pm \eta Df(x)$ dependiendo del signo de $\langle Df(x), w \rangle_q$, pero este producto puede ser 0 incluso si $Df(x) \neq 0$ (el producto escalar puede ser indefinido), por lo que esta solución no nos vale.

9.2.4. Solución propuesta

La solución que se estudia en [LS21] es la siguiente. Para asegurar que $w \in T_x \mathcal{Q}_\beta^q$ es una dirección de descenso, se propone una expresión que satisfaga $\langle Df(x), w \rangle_q < 0$ si $Df(x) \neq 0$ y que $\langle Df(x), w \rangle_q = 0$ si $Df(x) = 0$. Se propone tomar el vector tangente w como $w = -\eta \Pi_x(\mathbf{G} Df(x)) \in T_x \mathcal{Q}_\beta^q$ y definir el siguiente vector tangente $v = -\frac{1}{\eta} w = \Pi_x(\mathbf{G} Df(x))$, y utilizando (9.1):

$$v = \nabla f(x) - \frac{\langle \nabla f(x), x \rangle}{\langle x, x \rangle_q} \mathbf{G} x - \frac{\langle \nabla f(x), x \rangle_q}{\langle x, x \rangle_q} x + \frac{\|x\|^2 \langle \nabla f(x), x \rangle}{\langle x, x \rangle_q^2} x.$$

La fórmula para v ajusta la dirección del gradiente, $\nabla f(x)$, teniendo en cuenta tanto la métrica pseudo-riemanniana como la euclídea, asegurando que v sea una dirección de descenso válida en el contexto de los pseudo-hiperboloides.

El vector tangente w es una dirección de descenso porque $\langle Df(x), w \rangle_q = -\eta \langle Df(x), v \rangle_q \leq 0$:

$$\langle Df(x), v \rangle_q = \|\nabla f(x)\|^2 - 2 \frac{\langle \nabla f(x), x \rangle \langle \nabla f(x), x \rangle_q}{\langle x, x \rangle_q} + \frac{\langle \nabla f(x), x \rangle^2 \|x\|^2}{\langle x, x \rangle_q^2} =$$

$$= \|\mathbf{G} \nabla f(x) - \frac{\langle \nabla f(x), x \rangle}{\langle x, x \rangle_q} x\|^2 = \|Df(x)\|^2 \geq 0.$$

También tenemos las siguientes implicaciones:

- $Df(x) = 0$ (x es un punto estacionario) si, y solo si, $\langle Df(x), v \rangle_q = \|Df(x)\|^2 = 0$.
- $Df(x) = 0$ implica que $v = \Pi_x(\mathbf{G} 0) = 0$
- $v = 0$ implica que $\|Df(x)\|^2 = \langle Df(x), 0 \rangle_q = 0$. Por tanto, $v = 0$ si, y solo si $Df(x) = 0$.

El algoritmo para el problema de minimizar $f(x)$ para $x \in \mathcal{Q}_\beta^q$ puede ser aplicado a cualquier función $f : \mathcal{Q}_\beta^q \to \mathbb{R}$ y es el siguiente:

Algorithm 1 Optimización pseudo-riemanniana en $\mathcal{Q}_\beta^q$

Input: función diferenciable $f : \mathcal{Q}_\beta^q \to \mathbb{R}$, valor inicial de $x \in \mathcal{Q}_\beta^q$

1: **while** no converge **do**
2: Calcular $\nabla f(x)$
3: $v \leftarrow \Pi_x(\mathbf{G}\Pi_x(\mathbf{G}\nabla f(x)))$
4: $x \leftarrow \exp_x(-\eta v)$ ▷ con $\eta > 0$ un tamaño de paso
5: **end while**

La implementación del código de una adaptación de este algoritmo se encuentra en la Subsección 3.6.4. En el Capítulo 11 se estudiará el comportamiento de este método en la práctica y lo compararemos con la optimización euclídea definida en la Sección 9.1, con la que se obtendrán unos resultados ligeramente inferiores.

Parte III.

Experimentación

10. Descripción y configuración de los experimentos

En este capítulo se detalla la configuración de los experimentos realizados para evaluar los modelos de redes neuronales para grafos. Se explican los datasets utilizados, los modelos implementados, y los parámetros de configuración empleados durante el entrenamiento y la evaluación.

10.1. Metodología

Se ha decidido experimentar con dos de los modelos fundamentales de redes neuronales para grafos: GCN y GAT; junto con la arquitectura principal de este trabajo, una red neuronal para grafos en variedades pseudo-riemannianas: $\mathcal{Q}$GCN. Para este modelo experimentaremos con dos métodos de optimización vistos en el Capítulo 9: la optimación euclídea, con el optimizador Adam, y la optimización pseudo-riemanniana, con el optimizador Riemannian Adam, cuyo código se explicó en la Subsección 3.6.4. Las tres arquitecturas se introdujeron teorícamente en la Sección 3.5 y Sección 3.6, y en este capítulo resumiremos cómo han sido implementadas. Estudiaremos estos modelos para las tareas de clasificación de nodos y predicción de enlaces, con datasets de diversos tipos y tamaños para observar los distintos comportamientos de las arquitecturas.

Dataset
Cora
Citeseer
Disease (NC)
Disease (LP)
Photo
Airport
PPI

Tabla 10.1.: Datasets utilizados para la experimentación.

Modelo
$\mathcal{Q}$GCN (optimizador Adam)
$\mathcal{Q}$GCN (optimizador RiemannianAdam)
GCN
GAT

Tabla 10.2.: Modelos con los que se realizarán el estudio.

El proceso de experimentación se divide en las siguientes etapas:

1. Para cada modelo, cada dataset y cada tarea, se realiza una sencilla búsqueda de hiperparámetros. Esta búsqueda no se ha realizado para el modelo $\mathcal{Q}$GCN con el

optimizador RiemannianAdam por el alto coste de tiempo que conlleva entrenar este modelo. Por tanto, en este caso se seleccionarán los mismos hiperparámetros que el modelo $\mathcal{Q}$GCN con el optimizador Adam.

2. Cada caso se entrenará y probará con los mejores parámetros.
3. Se compararán las representaciones del espacio latente aprendidas por GCN y $\mathcal{Q}$GCN (radam) mediante visualizaciones en 3 y 2 dimensiones.

10.2. Datos

En esta sección describimos los datos utilizados para los experimentos.

10.2.1. Cora

El dataset Cora es un estándar en el campo de las redes neuronales para grafos. Consiste en una red de citación formada por 2708 publicaciones científicas clasificadas en una de entre 7 clases distintas, con los enlaces representando las citas entre ellos. Cada publicación tiene 1433 características binarias que indican la presencia (1) o la ausencia (0) de una palabra en la publicación.

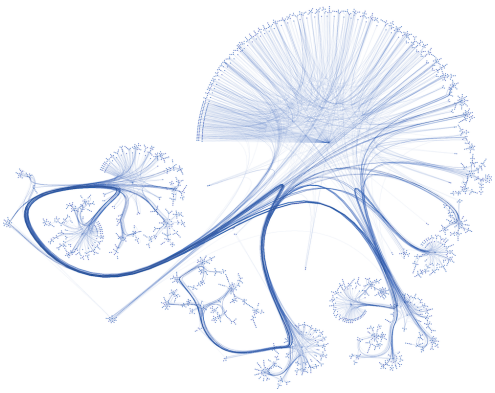


Figura 10.1.: Visualización del dataset Cora. Imagen obtenida de [Con24].

10.2.2. Citeseer

El dataset Citeseer, otra red de citación, consiste en 3327 publicaciones científicas clasificadas en seis categorías: Agents, AI, DB, IR, ML y HCI. El grafo tiene 4676 enlaces y cada publicación está asociada con un vector de 3703 características que indican la ausencia o presencia de palabras.

10.2.3. Disease

Este dataset consiste en la simulación del modelo SIR de propagación de enfermedades, donde la etiqueta de un nodo indica si está infectado o no. Basado en este modelo, se construyen redes de árboles, donde las características de los nodos indican la susceptibilidad

a la enfermedad. Disponemos de dos conjuntos de diferente tamaño y diferente número de atributos, uno para cada tarea a realizar.

10.2.4. Photo

El conjunto de datos Photo proviene del grafo de compras de Amazon [MTSvdH15], donde los nodos son los productos, las características son elementos de una representación de una *bag-of-words* de opiniones y las etiquetas representan la categoría de cada producto. Si dos productos son comprados juntos frecuentemente, existe un enlace entre ellos.

Figura 10.2.: Ejemplo de producto (izquierda) y productos que se suelen comprar junto a él (derecha). Imagen obtenida de [MTSvdH15].

10.2.5. PPI

Este conjunto de datos es una colección de interacciones físicas proteína-proteína de un gran número de tejidos humanos. Los nodos representan proteínas y los enlaces son interacciones específicas de los tejidos. El dataset consta de 20 grafos de entrenamiento, 2 de validación y 2 de test, cada uno representando un tejido diferente. Debido a su gran tamaño, para predicción de enlaces entrenaremos con un quinto de los datos y reduciremos a la mitad los conjuntos de validación y test. Las características incluyen aquellas compartidas por al menos un 10 % de las proteínas que aparecen en cualquier grafo PPI, por lo que el espacio de características es muy disperso (el 42 % de los nodos tienen todos los atributos nulos).

10.2.6. Airport

El dataset Airport representa una red de vuelos en la que los nodos son aeropuertos, su etiqueta es el país en el que se encuentra y los enlaces son las rutas aéreas. El dataset original se encuentra en [ZC18], para este problema se han aumentado el número de nodos y las características, añadiendo información geográfica (longitud, latitud y altitud), y el PIB del país correspondiente.

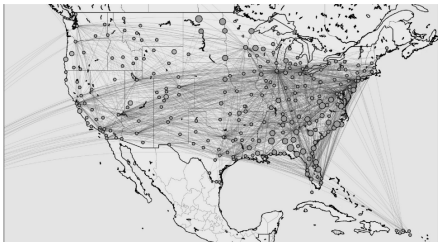


Figura 10.3.: Ejemplo de una red de aeropuertos. Imagen obtenida de [FCER14].

Dataset	Nº de nodos	Nº de enlaces	Nº de características	Nº de clases
Cora	2708	5278	1433	7
Citeseer	3327	4676	3703	6
Disease (NC)	1044	1043	1000	2
Disease (LP)	2665	2664	11	2
Photo	7650	71505	745	8
Airport	3188	18631	11	4
PPI	56944	212860	50	26

Tabla 10.3.: Resumen de los conjuntos de datos utilizados.

10.3. Modelos

En esta sección, describimos los modelos que se han utilizado. Todos ellos se explicaron en la Sección 3.5 y Sección 3.6, y aquí veremos un breve esquema de su código. Los modelos comparten la estructura codificador-decodificador, solamente diferenciándose en las capas del codificador. El decodificador dependerá de la tarea a realizar. Para clasificación de nodos se utiliza un decodificador lineal, una red neuronal feedforward que genera las probabilidades para la clasificación. Por otro lado, para predicción de enlaces se utiliza un decodificador Fermi-Dirac, que toma las distancias cuadradas entre las representaciones embebudas de los nodos y calcula la probabilidad de existencia de un enlace utilizando la función de distribución de Fermi-Dirac [KPK+10].

10.3.1. Red convolucional para grafos (GCN)

El modelo GCN está formado por capas convolucionales para grafos en el codificador. La clase GraphConvolution define una capa de convolución para grafos, aplicando una transformación lineal seguida de dropout y una multiplicación con la matriz de adyacencia.

Mostramos un ejemplo de un modelo de este tipo con 2 capas utilizado para predicción de enlaces:

```
LPModel(
  (encoder): GCN(
    (layers): Sequential(
      (0): GraphConvolution(input_dim=1433, output_dim=128
        (linear): Linear(in_features=1433, out_features=128, bias=True)
      )
      (1): GraphConvolution(input_dim=128, output_dim=128
        (linear): Linear(in_features=128, out_features=128, bias=True)
      )
    )
  )
  (decoder): FermiDiracDecoder()
)
```

En este ejemplo, la dimensión del espacio latente por defecto es 128, por lo que en la primera capa convolucional, la dimensión de entrada es el número de características (1433 en el dataset Cora) y la de salida es igual a la dimensión del embedding (128). Las siguientes capas

en el codificador tendrán como dimensión de entrada y salida la dimensión del embedding escogida.

10.3.2. Red de atención para grafos (GAT)

El modelo GAT introduce el mecanismo de atención en los grafos. Presentamos las clases principales que componen su implementación:

- La clase SpGraphAttentionLayer implementa la capa de atención para grafos, donde se calcula la atención entre nodos vecinos y se ajustan los valores utilizando una función de activación con LeakyReLU y dropout.
- La clase GraphAttentionLayer permite múltiples cabezas de atención, lo que mejora la capacidad del modelo para enfocarse en diferentes partes del vecindario de un nodo simultáneamente. Por defecto, el número de cabezas es 4.

El siguiente ejemplo muestra un modelo GAT con dos capas para la tarea de predicción de enlaces:

```
LPModel(
  (encoder): GAT(
    (layers): Sequential(
      (0): GraphAttentionLayer(
        (attention_0): SpGraphAttentionLayer (1433 -> 32)
        (attention_1): SpGraphAttentionLayer (1433 -> 32)
        (attention_2): SpGraphAttentionLayer (1433 -> 32)
        (attention_3): SpGraphAttentionLayer (1433 -> 32)
      )
      (1): GraphAttentionLayer(
        (attention_0): SpGraphAttentionLayer (128 -> 32)
        (attention_1): SpGraphAttentionLayer (128 -> 32)
        (attention_2): SpGraphAttentionLayer (128 -> 32)
        (attention_3): SpGraphAttentionLayer (128 -> 32)
      )
    )
  )
  (decoder): FermiDiracDecoder()
)
```

En este ejemplo, se ha elegido una dimensión del embedding de 128 y 4 cabezas de atención. Por tanto, la primera capa de atención está formada por 4 capas de atención especial, cada una con una dimensión de entrada igual al número de características (1433 en el dataset Cora) y como salida la fracción $\frac{\text{dim. embedding}}{\text{núm. cabezas}}$. Al igual que pasaba con el modelo GCN, las siguientes capas tienen como dimensión de entrada la del embedding.

10.3.3. $\mathcal{Q}$GCN

El modelo $\mathcal{Q}$GCN extiende las redes neuronales convolucionales para grafos a un espacio pseudo-hiperbólico. Recordemos brevemente las clases que lo forman.

La clase HyperbolicGraphConvolution define una capa de convolución para grafos en este espacio, utilizando operaciones específicas para manejar las propiedades y problemas del espacio. Esta clase, como ya se vió en la Subsección 3.6.2, está compuesta por tres componentes

principales: HypLinear, HypAgg y HypAct.

La siguiente imagen es un ejemplo de un modelo de este tipo con 2 capas utilizado para predicción de enlaces.

```
LPModel(
  (encoder): HGCN(
    (layers): Sequential(
      (0): HyperbolicGraphConvolution(
        (linear): HypLinear(in_features=1434, out_features=128, c=tensor
            ([-1.], device='cuda:0', grad_fn=<ToCopyBackward0>))

        (agg): HypAgg(c=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>))

        (hyp_act): HypAct(c_in=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>), c_out=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>))
      )
      (1): HyperbolicGraphConvolution(
        (linear): HypLinear(in_features=128, out_features=128, c=tensor([-1.],
             device='cuda:0', grad_fn=<ToCopyBackward0>))

        (agg): HypAgg(c=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>))

        (hyp_act): HypAct(c_in=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>), c_out=tensor([-1.], device='cuda:0', grad_fn=<
            ToCopyBackward0>))
      )
    )
  (decoder): FermiDiracDecoder()
)
```

En el ejemplo, la primera capa de convolución pseudo-hiperbólica tiene como entrada el número de características (1433 en el dataset Cora) y una característica adicional que corresponde a la curvatura del espacio. La salida de esta capa y la entrada de la siguiente tienen una dimensión igual a la del embedding (128).

10.4. Configuración

10.4.1. Parámetros de ejecución

Antes de explicar la configuración de los distintos modelos, listamos todos los parámetros con los que se puede ejecutar el código:

- `h`: Muestra todos los parámetros para el modelo.
- `lr`: Tasa de aprendizaje. Por defecto: 0.01.
- `dropout`: Probabilidad de dropout. Por defecto: 0.0.
- `cuda`: Qué dispositivo CUDA utilizar (-1 para CPU). Por defecto: 0.
- `epochs`: Máximo número de épocas durante el entrenamiento. Por defecto: 2000.

- `weight-decay`: Fuerza de la regularización L2. Por defecto: 0.
- `optimizer`: Qué optimizador usar: Adam o RiemannianAdam. Por defecto: Adam.
- `momentum`: Momentum en el optimizador. Por defecto: 0.999.
- `patience`: Paciencia para early stopping. Por defecto: 100.
- `seed`: Semilla para entrenamiento. Por defecto: 1234.
- `log-freq`: Frecuencia para imprimir las métricas de entrenamiento y validación. Por defecto: 1.
- `eval-freq`: Frecuencia para calcular las métricas de validación. Por defecto: 1.
- `save`: Guardar o no el modelo y los logs (1 para guardar, 0 para no). Por defecto: 1.
- `save-dir`: Directorio para guardar los logs y los pesos del modelo. Por defecto: None.
- `lr-reduce-freq`: Frecuencia con la que se reduce el lr o None para mantenerla constante. Por defecto: None.
- `gamma`: Parámetro gamma para lr scheduler. Por defecto: 0.5.
- `print-epoch`: Imprimir o no las épocas. Por defecto: True.
- `grad-clip`: Máxima norma para gradient clipping, o None para no hacer gradient clipping. Por defecto: None.
- `min-epochs`: Mínimo número de épocas que completar durante el entrenamiento. Por defecto: 100.
- `task`: Tarea a realizar, nc para clasificación de nodos o lp para predicción de enlaces. Por defecto: nc.
- `model`: Encoder a utilizar de entre Shallow, MLP, HNN, GCN, GAT, HGCN. Por defecto: GCN.
- `dim`: Dimensión del embedding. Por defecto: 128.
- `manifold`: Variedad a usar de entre Euclidean, Hyperboloid, PoincareBall y PseudoHyperboloid. Por defecto: Euclidean.
- `c`: Radio hiperbólico, None para curvatura entrenable. Por defecto: -1.
- `r`: Parámetro para el decoder fermi-dirac para predicción de enlaces. Por defecto: 2.0.
- `t`: Parámetro para el decoder fermi-dirac para predicción de enlaces. Por defecto: 1.0.
- `pretrained-embeddings`: Directorio a los embeddings preentrenados (fichero .npy) para la clasificación de nodos en el modelo Shallow. Por defecto: None.
- `pos-weight`: Aumentar o no la ponderación de la clase positiva en clasificación de nodos. Por defecto: 0.
- `num-layers`: Número de capas ocultas en el encoder. Por defecto: 2.

- `bias`: Usar (1) o no (0) sesgo. Por defecto: 1.
- `act`: Función de activación, None para ninguna. Por defecto: relu.
- `n-heads`: Número de cabezas de atención para redes de atención para grafos, debe ser un divisor de dim. Por defecto: 4.
- `alpha`: alpha para leakyrelu en las redes de atención para grafos. Por defecto: 0.02.
- `double-precision`: Usar o no precisión doble. Por defecto: 0.
- `use-att`: Usar o no atención hiperbólica. Por defecto: 0.
- `local-agg`: Usar o no agregación local del espacio tangente. Por defecto: 0.
- `space-dim`: Dimensión espacial para el modelo HGCN. Por defecto: 9.
- `time-dim`: Dimensión temporal para el modelo HGCN. Por defecto: 1.
- `dataset`: Conjunto de datos a usar. Por defecto: cora.
- `val-prop`: Proporción de enlaces para validación en predicción de enlaces. Por defecto: 0.05.
- `test-prop`: Proporción de enlaces para test en predicción de enlaces. Por defecto: 0.1.
- `use-feats`: Usar o no las características de los nodos. Por defecto: 1.
- `normalize-feats`: Normalizar o no las características de entrada de los nodos. Por defecto: 0.
- `normalize-adj`: Normalizar o no por filas la matriz de adyacencia. Por defecto: 1.
- `split-seed`: Semilla para la división de los datos para train, validación y test. Por defecto: 1000.

Una vez especificados los parámetros posibles, debido a la gran cantidad de parámetros, se ha decidido hacer los experimentos con los parámetros por defecto, con algunas excepciones que veremos más adelante. Por lo tanto, los modelos se han entrenado y ejecutado de la siguiente forma:

- **GCN**:

```
!python train.py --model GCN --manifold Euclidean --task {task} --dataset
    {dataset} --cuda {cuda}
```

  donde `{task}` será la tarea a realizar, nc o lp; `{dataset}` los datos del problema y `{cuda}` será el dispositivo GPU a utilizar, que será 0, salvo para el dataset PPI, que por su tamaño debemos entrenar el modelo con la CPU (-1).

- **GAT**:

```
!python train.py --model GAT --manifold Euclidean --task {task} --dataset
    {dataset} --cuda {cuda}
```

El modelo GAT estudiado tendrá 4 cabezas de atención y la función LeakyReLu de la clase SpGraphAttentionLayer tendrá un alpha de 0.02; la descripción de los demás parámetros es igual que para GCN.

- **$\mathcal{Q}$GCN adam**:

```
!python train.py --model HGCN --manifold PseudoHyperboloid --task {task}
    --dataset {dataset} --cuda {cuda}
```

- **$\mathcal{Q}$GCN radam**:

```
!python train.py --model HGCN --manifold PseudoHyperboloid --task {task}
    --optimizer RiemannianAdam --dataset {dataset} --cuda {cuda}
```

Los modelos $\mathcal{Q}$GCN tendrán los siguientes parámetros por defecto específicos:

- La curvatura se aprenderá durante el entrenamiento.
- La dimensión espacial del pseudo-hiperboloide será 9.
- La dimensión temporal será 2.

Por tanto, el pseudo-hiperboloide utilizado para la experimentación ha sido $\mathcal{Q}^2$ de $\mathbb{R}^{11}$.

10.4.2. Búsqueda de hiperparámetros

Para cada modelo, cada dataset y cada tarea, se ha llevado a cabo una búsqueda de hiperparámetros no muy compleja, donde el entrenamiento ha sido limitado a 50 épocas en todos los datasets, a excepción de PPI, donde el límite ha sido 10 épocas. Esta búsqueda no se ha llevado a cabo para el modelo $\mathcal{Q}$GCN radam por el alto coste que conlleva entrenar con el optimizador RiemannianAdam. Por tanto, en este caso se seleccionarán los mismos parámetros que el modelo $\mathcal{Q}$GCN adam. La elección de búsqueda sobre estos hiperparámetros ha sido de forma que estos sean comunes a las tres arquitecturas, evitando así ajustar mucho un modelo sobre los demás.

Hiperparámetro	Espacio de Búsqueda
Número de capas	$2, 3$
Weight decay	$0, 0.001$
Dropout	$0, 0.2, 0.5$
Activación	ReLU, Tanh, ELU

Tabla 10.4.: Espacio de búsqueda para los hiperparámetros del modelo.

10.5. Protocolo de validación

10.5.1. Clasificación de nodos

Para la tarea de clasificación de nodos, la validación de estos modelos se realiza mediante una división de los datos en conjuntos de entrenamiento, validación y prueba. En la siguiente tabla se describe el proceso de validación para cada dataset:

Dataset	Total de nodos	Entrenamiento	Validación	Test
Cora	2708	140 (5 %)	500 (18 %)	1000 (37 %)
Citeseer	3327	120 (4 %)	500 (15 %)	1000 (30 %)
Disease_nc	1044	626 (60 %)	314 (30 %)	104 (10 %)
Airport	3188	478 (15 %)	2232 (70 %)	478 (15 %)
Photo	7650	1148 (15 %)	5354 (70 %)	1148 (15 %)
PPI	56944	44906 (79 %)	6514 (11 %)	5524 (10 %)

Tabla 10.5.: Distribución de nodos en conjuntos de entrenamiento, validación y prueba para los diferentes datasets.

10.5.2. Predicción de enlaces

Para la predicción de enlaces, utilizamos diferentes tamaños de conjuntos de entrenamiento, validación y prueba según el dataset:

- **PPI**: Reducimos su tamaño a 4 redes para entrenamiento, 1 red para validación y 1 red para prueba.
- **Los demás conjuntos de datos**: Se utilizan los valores por defecto, esto es, 5 % para validación, 10 % para prueba y el 85 % restante para entrenamiento.

10.6. Función de pérdida

La función de pérdida utilizada es la usual para problemas de clasificación, la entropía cruzada, cuya fórmula es la siguiente:

$$-\sum_{i=1}^{D}\sum_{n=1}^{N}\mathbf{y}_{i,n}\log(\hat{\mathbf{y}}_{i,n}),$$

donde D es el número de muestras, N el número de clases, $\mathbf{y}_{i,n}$ es 1 si $\mathbf{y}_i$ pertenece a la clase n y es 0 en caso contrario; $\hat{\mathbf{y}}_{i,n}$ es la probabilidad predicha del ejemplo i para la clase n.

10.7. Métricas de rendimiento

Para medir el rendimiento de los distintos modelos implementados, se han utilizado funciones del paquete `sklearn.metrics`. Sin embargo, antes de introducir las métricas usadas, definimos algunos conceptos básicos. Sea un problema de clasificación binario, esto es, que tiene dos clases, positiva y negativa, se definen:

- **Verdaderos Positivos (TP)**: los ejemplos positivos que han sido clasificados correctamente.
- **Verdaderos Negativos (TN)**: los ejemplos negativos que han sido clasificados correctamente.
- **Falsos Positivos (FP)**: los ejemplos negativos que han sido clasificados como positivos.
- **Falsos Negativos (FN)**: los ejemplos positivos que han sido clasificados como negativos.

A partir de estos conceptos, se definen las métricas utilizadas.

10.7.1. Clasificación de nodos

Para la clasificación de nodos, se han utilizado las siguientes métricas:

- **Accuracy**: Esta métrica mide la tasa de acierto de predicciones correctas entre el número total de predicciones. Matemáticamente, se formula así:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **F1-Score**: Esta métrica es la media armónica de la precisión (predicciones positivas correctas entre el total de predicciones positivas) y la exhaustividad (predicciones positivas correctas entre el total de positivos), proporcionando un equilibrio entre ambas. Se define como:

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

  En `sklearn`, se puede calcular utilizando:

```
f1 = f1_score(preds, labels, average=average)
```

  Donde `average` es igual a `'binary'` para problemas binarios y `'micro'` para problemas multiclase. El promedio micro calcula las métricas globalmente contando todos los verdaderos positivos, falsos negativos y falsos positivos.

10.7.2. Predicción de enlaces

Para la predicción de enlaces, se han utilizado las siguientes métricas:

- **Área bajo la curva ROC (AUC)**: La curva ROC es una representación en 2 dimensiones del rendimiento del modelo, que muestra el compromiso entre los verdaderos positivos y los falsos positivos. El área bajo esta curva (AUC) es una métrica utilizada para medir el rendimiento de clasificadores. En `sklearn`, se puede calcular utilizando:

```
auc = roc_auc_score(labels, preds)
```

- **Average Precision (AP)**: Esta métrica resume la relación entre precision (predicciones positivas correctas entre el total de predicciones positivas) y recall (predicciones positivas correctas entre el total de positivos) en una sola cifra. Esto lo consigue calculando el promedio ponderado de las precisiones alcanzadas en cada umbral, con el incremento en recall como el peso. Es una medida que es especialmente útil en problemas con un desbalance alto entre clases, como ocurre en nuestro caso. En `sklearn`, se puede calcular utilizando:

```
ap = average_precision_score(labels, preds)
```

10.8. Herramientas

Se ha utilizado la plataforma Google Colab y el lenguaje Python para para toda la experimentación. Google Colab es una herramienta gratuita ofrecida por la empresa Google que permite escribir y ejecutar código de Python desde el navegador de manera sencilla. La decisión de usar Google Colab radica en que proporciona una GPU a un coste bajo, una gran memoria RAM, permite trabajar con cuadernos de Python y almacenarlos en Google Drive. A continuación, mostramos el hardware usado en el análisis experimental:

- GPU Nvidia Tesla T4.
- 15 GB de RAM de la GPU.
- 51 GB de RAM del sistema.
- 202.2 GB de memoria de disco.
- 6× CPU Intel(R) Xeon(R) CPU @ 2.00 GHz.

También se han utilizado librerías de Python básicas para el desarrollo de redes neuronales para grafos, análisis y procesamiento de datos. Describimos a continuación estas librerías, así como las versiones utilizadas:

- **NumPy** (versión 1.25.2): es una biblioteca utilizada para operaciones con arrays y matrices de manera eficiente.
- **Torch** (versión 2.3.0) y **Torchvision** (versión 0.18.0): son librerías principales para el desarrollo de modelos de aprendizaje profundo, que proporcionan estructuras de datos como tensores, básicos para el desarrollo de redes neuronales.
- **Scikit-Learn** (versión 1.2.2): Utilizada para tareas de preprocesamiento de datos, evaluación de modelos y selección de hiperparámetros.
- **SciPy** (versión 1.11.4): librería empleada para cálculos científicos y técnicos avanzados, como algoritmos de optimización.
- **networkx** (versión 2.5): paquete de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas [Net24].
- **Matplotlib** (versión 3.7.1): biblioteca que permite crear visualizaciones en Python.
- **Seaborn** (versión 0.13.1): biblioteca de visualización de datos más avanzada basada en Matplotlib.
- **UMAP** (versión 0.5.6): técnica de reducción de dimensionalidad para la visualización de datos de alta dimensión, lo usaremos para la visualización de los embeddings.

El código desarrollado en este proyecto se encuentra en el siguiente enlace de Github[1].

[1] https://github.com/germanpadua/GCN-Pseudo-Riemannian-Manifold

10.9. Implementación

En esta sección se detallará de dónde se ha obtenido el código relevante, la fuente de los datasets, y las diversas modificaciones y adaptaciones que se han desarrollado al código para que este funcione correctamente.

El código de la arquitectura principal, $\mathcal{Q}$GCN, proviene del artículo [XZP+22b], el cual incluye la adaptación para trabajar con pseudo-hiperboloides de las redes implementadas en [CYRL19]. En [CYRL19] también se incluye el código de los modelos GCN y GAT.

De los conjuntos de datos utilizados en este proyecto, solamente Cora se encontraba en el repositorio de [XZP+22b]. Los datos de Citeseer se obtuvieron del repositorio de [KW17], los datasets de Disease y Airport se descargaron de [CYRL19], Photo se obtuvo del repositorio de [SMBG18], y el dataset PPI se descargó de la página [WLH24].

A continuación, se detallan las modificaciones introducidas en el código para asegurar su correcto funcionamiento:

- Se cambió `np.int` por `int`, ya que `np.int` no se utiliza en la versión actual de NumPy.
- Se implementó la lectura de los datasets Photo y PPI. A continuación, se muestra el código para cargar el dataset PPI:

```python
def load_data_ppi(data_path, return_label=True):
    # Cargar los datos del grafo
    with open(os.path.join(data_path, "ppi-G.json")) as f:
        G_data = json.load(f)
    G = json_graph.node_link_graph(G_data)

    # Conversion de nodos a enteros si es necesario
    if isinstance(list(G.nodes())[0], int):
        conversion = lambda n: int(n)
    else:
        conversion = lambda n: n

    # Cargar caracteristicas de los nodos si existen
    feats_path = os.path.join(data_path, "ppi-feats.npy")
    if os.path.exists(feats_path):
        feats = np.load(feats_path)
    else:
        print("No features present.. Only identity features will be used.
            ")
        feats = None

    # Cargar mapa de IDs y etiquetas de clases
    with open(os.path.join(data_path, "ppi-id_map.json")) as f:
        id_map = json.load(f)
    id_map = {conversion(k): int(v) for k, v in id_map.items()}

    with open(os.path.join(data_path, "ppi-class_map.json")) as f:
        class_map = json.load(f)
    if isinstance(list(class_map.values())[0], list):
        lab_conversion = lambda n: n
    else:
        lab_conversion = lambda n: int(n)
```

```
    class_map = {conversion(k): lab_conversion(v) for k, v in class_map.
        items()}

    # Borrar nodos que no tienen anotaciones de validacion/prueba
    broken_count = 0
    for node in list(G.nodes()):
        if not 'val' in G.nodes[node] or not 'test' in G.nodes[node]:
            G.remove_node(node)
            broken_count += 1

    # Asegurarse de que el grafo tiene anotaciones de enlaces eliminados
        para entrenamiento
    for edge in G.edges():
        if (G.nodes[edge[0]]['val'] or G.nodes[edge[1]]['val'] or
                G.nodes[edge[0]]['test'] or G.nodes[edge[1]]['test']):
            G[edge[0]][edge[1]]['train_removed'] = True
        else:
            G[edge[0]][edge[1]]['train_removed'] = False

    # Normalizar caracteristicas si existen
    if feats is not None:
        train_ids = np.array([id_map[n] for n in G.nodes() if not G.nodes
            [n]['val'] and not G.nodes[n]['test']])
        train_feats = feats[train_ids]
        scaler = StandardScaler()
        scaler.fit(train_feats)
        feats = scaler.transform(feats)

    # Convertir la matriz de adyacencia
    adj = nx.adjacency_matrix(G)

    # Crear la matriz de etiquetas
    labels = np.zeros((len(G.nodes()), len(class_map[list(class_map.keys
        ())[0]])))
    for node, label in class_map.items():
        labels[id_map[node]] = label

    if return_label:
        idx_train = [id_map[node] for node in G.nodes() if not G.nodes[
            node]['val'] and not G.nodes[node]['test']]
        idx_val = [id_map[node] for node in G.nodes() if G.nodes[node]['
            val']]
        idx_test = [id_map[node] for node in G.nodes() if G.nodes[node]['
            test']]

        return sp.csr_matrix(adj), torch.Tensor(feats), torch.LongTensor(
            labels), idx_train, idx_val, idx_test
    else:
        return sp.csr_matrix(adj), torch.Tensor(feats), G
```

Además, se creó una función `mask_edges_ppi` para reducir el tamaño del conjunto de entrenamiento en el dataset PPI mientras evita incluir grafos incompletos.

- Se implementó la lectura del dataset Amazon Photo:

```
def load_data_amazon_photo(data_path, return_label=False):
    # Cargar datos desde el archivo .npz
    data = np.load(os.path.join(data_path, 'amazon_electronics_photo.npz'
        ))

```

```
    # Extraer la matriz de adyacencia
    adj_matrix = sp.csr_matrix((data['adj_data'], data['adj_indices'],
        data['adj_indptr']), shape=data['adj_shape'])

    # Extraer las caracteristicas de los nodos
    features = sp.csr_matrix((data['attr_data'], data['attr_indices'],
        data['attr_indptr']), shape=data['attr_shape'])

    # Extraer las etiquetas
    labels = data['labels']

    # Convertir las caracteristicas a formato denso
    features = features.toarray()

    # Normalizar las caracteristicas
    scaler = StandardScaler()
    features = scaler.fit_transform(features)

    # Convertir las caracteristicas a tensor
    features = torch.tensor(features, dtype=torch.float)

    # Convertir las etiquetas a tensor
    labels = torch.tensor(labels, dtype=torch.long)

    return sp.csr_matrix(adj_matrix), features, labels
```

- Se mostró información sobre los datos del problema y del dataset antes de comenzar el entrenamiento.
- Se evitó la división por cero durante el entrenamiento del modelo GAT.

```
# Sumamos un valor cercano a cero para evitar error en la division
h_prime = h_prime.div(e_rowsum + 1e-10)
```

Listing 10.1: Modificación de la clase SpGraphAttentionLayer.

- Se modificó la clase `Euclidean`, que representa un espacio euclídeo, para que sea consistente con el código de ambos optimizadores. Esta modificación permite que la clase acepte parámetros adicionales `space_dim` y `time_dim` en su constructor, asegurando la compatibilidad con diferentes configuraciones de modelo. El código modificado se muestra a continuación:

```
class Euclidean(Manifold):
    """
    Euclidean Manifold class.
    """

    def __init__(self, space_dim=None, time_dim=None):
        super(Euclidean, self).__init__()
        self.name = 'Euclidean'
        self.space_dim = space_dim
        self.time_dim = time_dim
```

Listing 10.2: Modificación de la clase Euclidean.

11. Resultados y discusión

En este capítulo se presentarán todos los resultados obtenidos en la experimentación, entrando en detalle en la evolución durante el entrenamiento de algunos datasets. Además, compararemos los embeddings aprendidos en GCN y en $\mathcal{Q}$GCN radam para las dos tareas estudiadas. Finalizaremos el capítulo con las conclusiones finales de este proyecto y las posibles líneas de investigación para trabajos futuros.

11.1. Resultados

11.1.1. Clasificación de nodos

En primer lugar, presentamos en la siguiente tabla los resultados de la búsqueda de hiperparámetros de los modelos para todos los datasets.

Tabla 11.1.: Resultados de grid search para los datasets.

Dataset	Modelo	Núm. Capas	Weight Decay	Dropout	Activación
Cora	*QGCN*	2	0.001	0.5	ReLU
	GCN	2	0.001	0.2	Tanh
	GAT	2	0.001	0.5	ELU
Citeseer	*QGCN*	2	0.001	0	ReLU
	GCN	2	0.001	0.2	Tanh
	GAT	2	0.001	0	ELU
Disease NC	*QGCN*	2	0.001	0.5	Tanh
	GCN	2	0.001	0.2	ELU
	GAT	2	0	0	Tanh
Photo	*QGCN*	2	0	0	Tanh
	GCN	2	0	0	Tanh
	GAT	3	0.001	0	Tanh
Airport	*QGCN*	3	0	0	ELU
	GCN	3	0	0	ReLU
	GAT	3	0	0	ReLU
PPI	*QGCN*	2	0	0	ReLU
	GCN	3	0	0.5	ELU
	GAT	3	0	0	ReLU

Una vez vistos los hiperparámetros con los que hemos entrenado los modelos, destacamos y comentamos algunas de las curvas de aprendizaje que nos han parecido interesantes.

11.1.1.1. Entrenamiento y validación para Cora

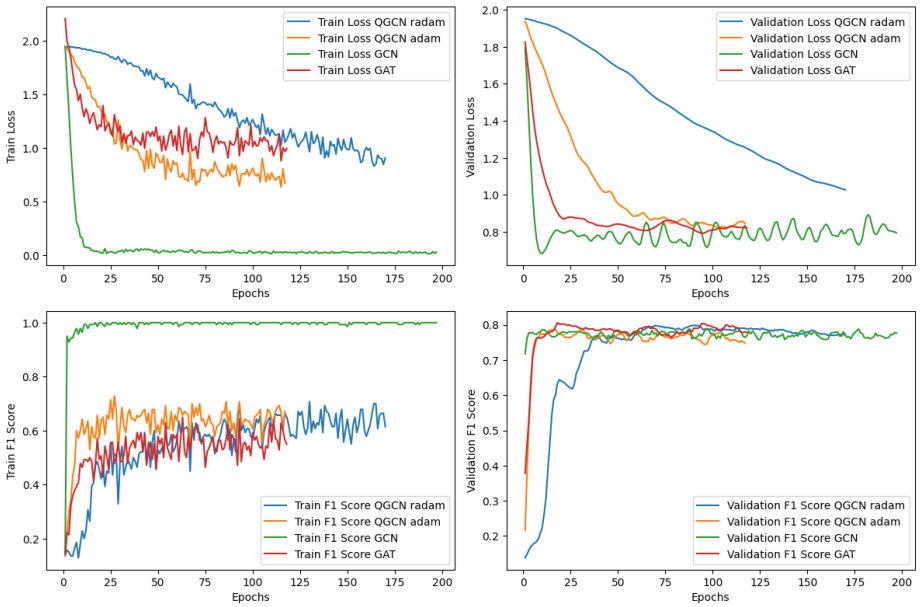


Figura 11.1.: Curvas de pérdida y $F1$ durante entrenamiento y validación para Cora.

En los gráficos se observa un rendimiento muy similar sobre el conjunto de validación con el optimizador Adam. Aunque durante el entrenamiento, la pérdida y $F1$ son mejores en el modelo GCN, lo que puede indicar que este modelo sufre de sobreaprendizaje. También destacamos que la pérdida, tanto en entrenamiento como en validación, para el modelo QGCN radam es la que disminuye más lentamente, lo que puede significar que la tasa de aprendizaje seleccionada no es la adecuada.

11.1.1.2. Entrenamiento y validación para Photo

Para el dataset Photo, también se observan unos resultados parecidos entre los modelos, véase la Figura 11.2. De nuevo, QGCN radam disminuye su pérdida más lento que los demás, mientras que el modelo GCN es el que consigue reducir la pérdida más rápidamente. Sin embargo, en ambos modelos las pérdidas y $F1$ se igualan en las últimas épocas.

11.1.1.3. Entrenamiento y validación para Airport

En este caso, véase la Figura 11.3, el rendimiento de los modelos es muy similar durante el entrenamiento. Pero sí que se observan diferencias en el conjunto de validación, donde la pérdida del modelo GAT empeora a partir de la época 75, lo que puede indicar sobreentrenamiento. Los otros dos modelos con optimizador Adam tienen curvas similares, aunque sobre la época 225 el modelo QGCN adam empeora considerablemente pero consigue volver en unas pocas épocas a las métricas anteriores.

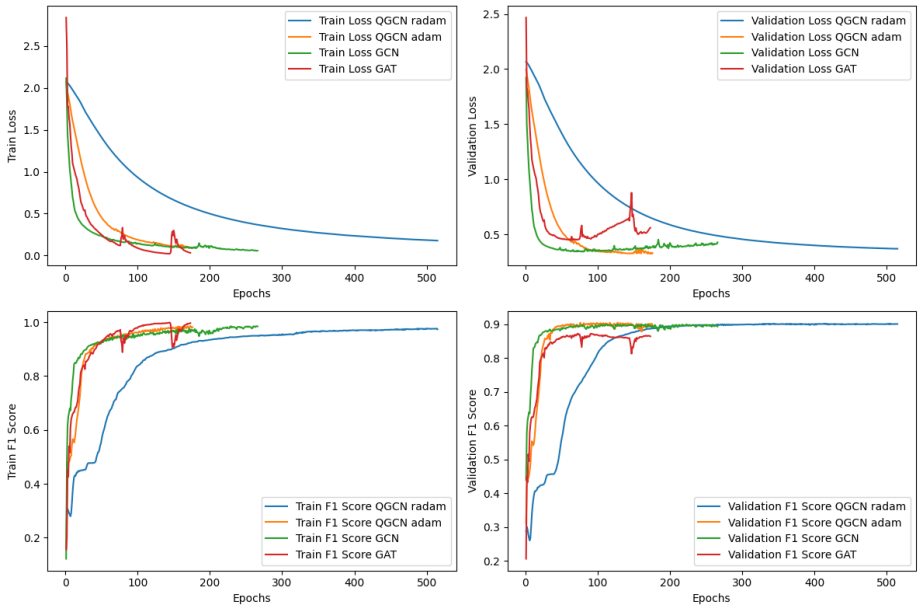


Figura 11.2.: Curvas de pérdida y $F1$ durante entrenamiento y validación para Photo.

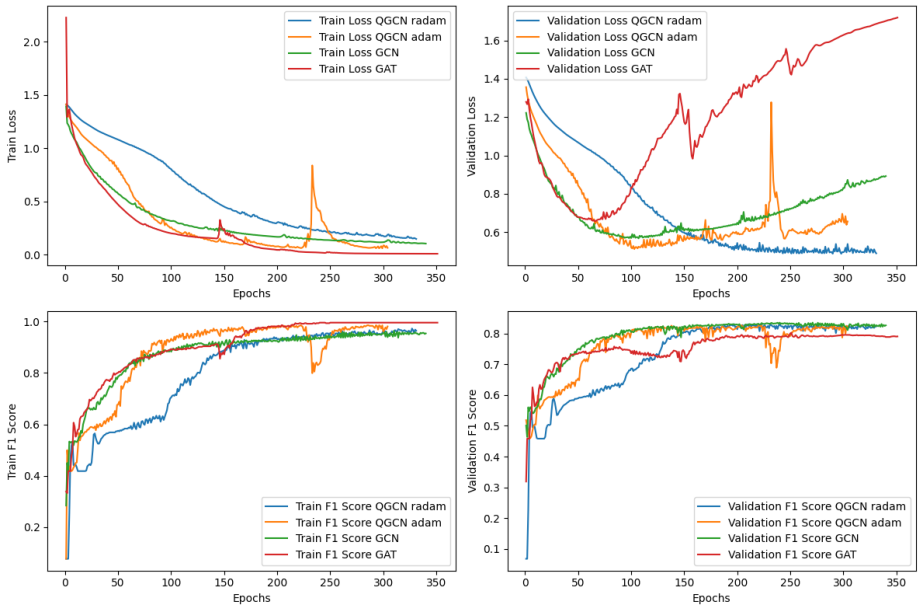


Figura 11.3.: Curvas de pérdida y $F1$ durante entrenamiento y validación para Airport.

11.1.1.4. Entrenamiento y validación para PPI

Por último, en PPI los modelos mejoran muy rápidamente pero se estancan y no consiguen aumentar su rendimiento en las siguientes épocas. Debido al alto tiempo de entrenamiento para el modelo GAT con estos datos, se limitó el entrenamiento a 20 épocas. El modelo GCN

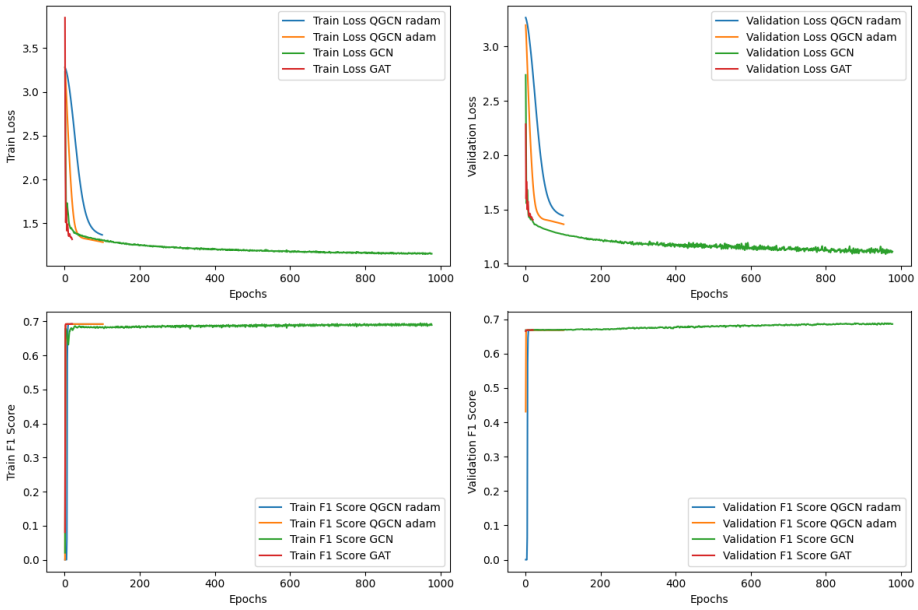


Figura 11.4.: Curvas de pérdida y $F1$ durante entrenamiento y validación para PPI.

ha sido entrenado durante 1000 épocas, debido a que mejoraba poco a poco, el early stopping no ha ocurrido hasta ese número de épocas. Además, este modelo ha sido el que ha obtenido los mejores resultados de pérdida y de $F1$.

11.1.1.5. Resultados sobre Test

Tabla 11.2.: Resultados de test para clasificación de nodos.

Dataset	Cora		Citeseer		Disease NC		Photo		Airport		PPI	
Modelo	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1
$\mathcal{Q}$GCN radam	0.81	0.81	**0.706**	**0.706**	0.9231	0.8	**0.9207**	**0.9207**	0.8138	0.8138	0.6736	0.6736
$\mathcal{Q}$GCN adam	0.8060	0.8060	0.6780	0.6780	0.9038	0.7500	0.9199	0.9199	0.7782	0.7782	0.6736	0.6736
GCN	**0.8210**	**0.8210**	0.6820	0.6820	**0.9327**	**0.8205**	0.9146	0.9146	**0.8180**	**0.8180**	**0.6986**	**0.6986**
GAT	0.8050	0.8050	0.6950	0.6950	0.8077	0.0000	0.8850	0.8850	0.7929	0.7929	0.6743	0.6743

Sobre los resultados en el conjunto de prueba de cada dataset, vemos que en los conjuntos más pequeños como Cora, Citeseer y Disease NC, las métricas son muy similares, destacando que el modelo GCN es el que mejor resultado obtiene en general. Por otro lado, para el dataset Photo, el mejor resultado se tiene con $\mathcal{Q}$GCN radam, lo que indica que este modelo es el que mejor capta las relaciones entre los distintos productos que se compran conjuntamente y sus categorías. Las peores métricas se obtienen con el dataset PPI, de gran tamaño, con un rendimiento más alto de GCN sobre los demás. Cabe resaltar que el modelo $\mathcal{Q}$GCN con el optimizador RiemannianAdam ha superado en todos los resultados al optimizador Adam. Por lo que concluimos que el método de optimización pseudo-riemanniano explicado es más adecuado que la optimización euclídea para pseudo-hiperboloides.

11.1.2. Predicción de enlaces

Seguimos el mismo procedimiento que para clasificación de nodos. En la Tabla 11.3 presentamos los resultados de la búsqueda de hiperparámetros para todos los datasets.

Tabla 11.3.: Resultados de grid search para los datasets.

Dataset	Modelo	Núm. Capas	Weight Decay	Dropout	Activación
Cora	*QGCN*	2	0	0.5	ELU
	GCN	2	0	0.2	Ninguna
	GAT	2	0.001	0	ELU
Citeseer	*QGCN*	2	0	0	Tanh
	GCN	2	0	0.2	Ninguna
	GAT	2	0.001	0.2	ReLU
Disease LP	*QGCN*	3	0.001	0	ReLU
	GCN	2	0	0.5	Ninguna
	GAT	2	0.001	0.5	ELU
Photo	*QGCN*	3	0.001	0	ReLU
	GCN	2	0	0	Ninguna
	GAT	2	0	0.5	Tanh
Airport	*QGCN*	2	0.001	0	ReLU
	GCN	2	0	0	Ninguna
	GAT	2	0	0	ReLU
PPI	*QGCN*	3	0.001	0	ELU
	GCN	3	0.001	0	Ninguna
	GAT	2	0.001	0.2	Tanh

Una vez vistos los hiperparámetros con los que hemos entrenado los modelos, destacamos y comentamos algunas de las curvas de aprendizaje que nos han parecido interesantes.

11.1.2.1. Entrenamiento y validación para Cora

Las curvas de entrenamiento y validación para el dataset Cora se encuentran en la Figura 11.5. En el conjunto de entrenamiento $\mathcal{Q}$GCN adam tiene un mayor rendimiento en comparación con los otros modelos. Sin embargo, durante la validación, aunque las métricas son similares, la curva de la pérdida para el modelo $\mathcal{Q}$GCN adam oscila significativamente. Esto puede indicar una falta de estabilidad en el proceso de entrenamiento, lo que podría deberse a una variedad de factores, tales como sobreajuste, regularización insuficiente o inestabilidad del optimizador. En cambio, en el modelo $\mathcal{Q}$GCN radam las oscilaciones son menores, por lo que en este caso el optimizador RiemannianAdam es más estable que el Adam.

11.1.2.2. Entrenamiento y validación para Photo

Las curvas se encuentran en la Figura 11.6. A pesar de haber entrenado durante más épocas para GCN, las diferencias entre los modelos son claras, siendo ambos $\mathcal{Q}$GCN los mejores, seguidos por GCN y por GAT. En $\mathcal{Q}$GCN adam se observan unas oscilaciones mayores en la función de pérdida de validación, en comparación con $\mathcal{Q}$GCN radam.

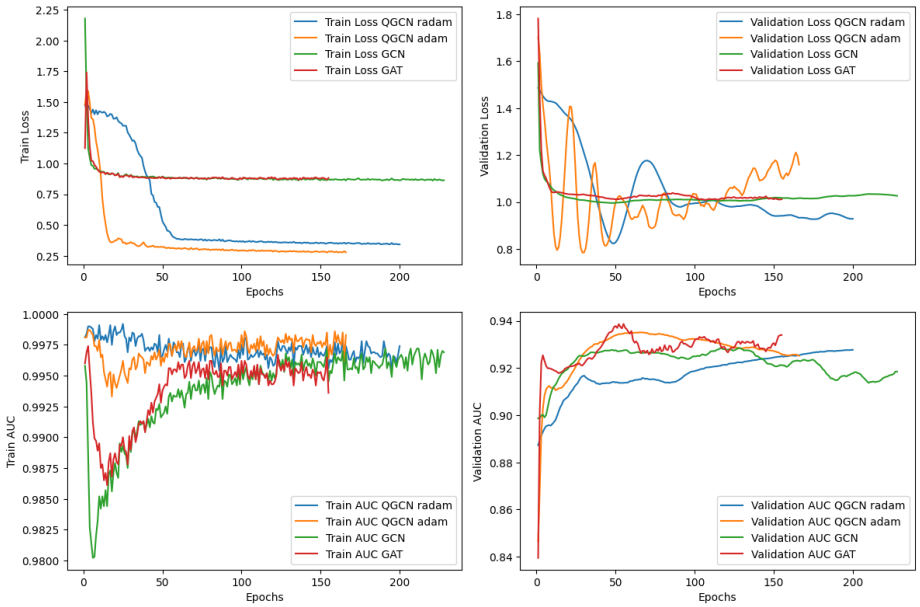


Figura 11.5.: Curvas de pérdida y AUC durante entrenamiento y validación para Cora.

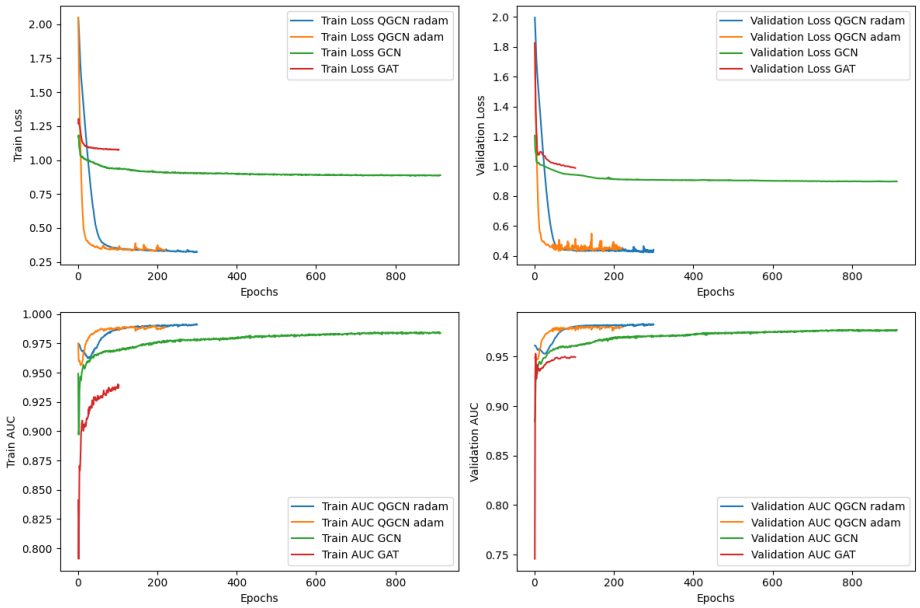


Figura 11.6.: Curvas de pérdida y AUC durante entrenamiento y validación para Photo.

11.1.2.3. Entrenamiento y validación para Airport

Para el dataset Airport, los cuatro modelos obtienen muy buenos resultados, por encima de 0.9 de AUC. Los modelos $\mathcal{Q}$GCN destacan por su alto rendimiento tanto en pérdida como en AUC, aunque se observa una ligera inestabilidad en la pérdida de validación.

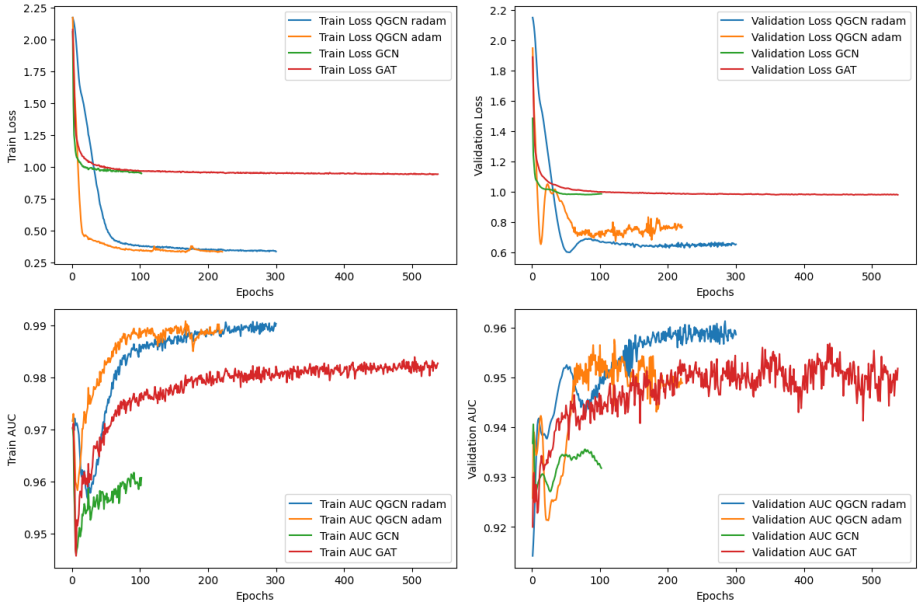


Figura 11.7.: Curvas de pérdida y AUC durante entrenamiento y validación para Airport.

11.1.2.4. Entrenamiento y validación para PPI

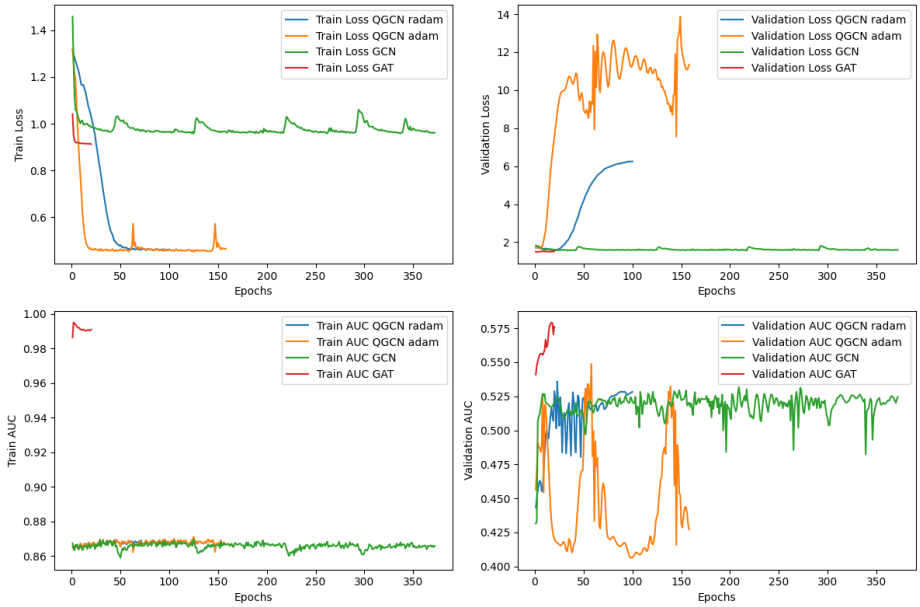


Figura 11.8.: Curvas de pérdida y AUC durante entrenamiento y validación para PPI.

Finalizamos comentando los resultados del dataset PPI durante el entrenamiento y validación reflejados en la Figura 11.8. Los modelos QGCN muestran una pérdida de entrenamiento inicial muy baja, pero en validación se muestra un grave empeoramiento tanto con Adam

como con RiemannianAdam. El modelo GCN mantiene una pérdida constante, aunque alta. A pesar de entrenar un número de épocas mucho más bajo, el modelo GAT es el que obtiene mejores resultados en todas las métricas exceptuando la pérdida en entrenamiento. En resumen, para este dataset, los modelos GCN y GAT podrían ser los más adecuados debido a su mayor estabilidad y consistencia en el rendimiento.

11.1.2.5. Resultados sobre Test

Tabla 11.4.: Resultados de test para predicción de enlaces.

Dataset	Cora		Citeseer		Disease LP		Photo		Airport		PPI	
Modelo	ROC	AP	ROC	AP	ROC	AP	ROC	AP	ROC	AP	ROC	AP
$\mathcal{Q}$GCN radam	0.9331	0.9224	0.944	0.9388	**0.944**	**0.9388**	**0.9852**	**0.9805**	**0.9608**	**0.9541**	0.5393	0.5256
$\mathcal{Q}$GCN adam	0.9333	0.9279	**0.9561**	**0.9474**	0.8101	0.7426	0.9822	0.9776	0.9551	0.9480	0.5480	0.5333
GCN	0.9224	0.9242	0.9405	0.9409	0.6832	0.6212	0.9786	0.9728	0.9323	0.9227	0.5293	0.5211
GAT	**0.9338**	**0.9280**	0.9480	0.9376	0.8006	0.7249	0.9551	0.9458	0.9465	0.9383	**0.5793**	**0.5413**

Sobre los resultados en el conjunto de prueba de cada dataset (Tabla 11.4), vemos que para esta tarea los modelos $\mathcal{Q}$GCN sobresalen sobre los otros, obteniendo las mejores medidas en todos los datasets, exceptuando en Cora, donde la diferencia es mínima con GAT; y en PPI, donde GAT es claramente el mejor modelo. Estos resultados indican que $\mathcal{Q}$GCN con ambos optimizadores tiene un rendimiento excepcional en muchos casos. Pero para el dataset PPI, que está formado por múltiples redes, GAT es el modelo más adecuado.

11.1.3. Tiempos de entrenamiento

La siguiente tabla incluye los tiempos que han tardado los modelos en entrenar para los distintos datasets en las dos tareas estudiadas, predicción de enlaces (LP) y clasificación de nodos (NC):

Tabla 11.5.: Tiempos de entrenamiento.

Dataset	Cora		Citeseer		Disease		Photo		Airport		PPI	
Tarea	LP	NC	LP	NC	LP	NC	LP	NC	LP	NC	LP	NC
$\mathcal{Q}$GCN radam	253.09	110.08	187.5	87.92	187.58	176.06	610.9	347.52	387.74	387.07	1651.13	531.53
$\mathcal{Q}$GCN adam	210.85	78.71	155.59	74.40	585.44	91.77	456.17	122.56	287.70	363.75	2666.28	562.68
GCN	8.51	3.69	7.50	2.49	4.04	3.52	166.88	6.19	6.89	6.92	497.37	869.79
GAT	19.00	8.78	18.89	13.35	30.54	115.71	30.85	29.63	78.67	40.64	1049.68	1339.43

Al analizar los tiempos de entrenamiento, destacamos los siguientes puntos clave:

- **$\mathcal{Q}$GCN**: En prácticamente todos los casos, esta arquitectura tiene los tiempos de entrenamiento más largos. Esto se debe a la complejidad adicional de estos modelos, que incluyen cálculos y funciones adicionales.
- **GCN**: El modelo GCN es consistentemente el más rápido de entrenar en la mayoría de los datasets. Sus tiempos de entrenamiento son significativamente menores en comparación con $\mathcal{Q}$GCN y GAT, lo que se debe a su arquitectura más simple y menos exigente en términos computacionales.

- **GAT**: El modelo GAT tiene tiempos de entrenamiento intermedios. Aunque es más lento que GCN debido a la atención adicional que debe calcular, sigue siendo más rápido que $\mathcal{Q}$GCN en la mayoría de los casos.
- **Comparación entre optimizadores (Adam vs RiemannianAdam)**: Salvo en 3 casos, los tiempos de entrenamiento con el optimizador RiemannianAdam han sido mayores que con Adam. Esto sugiere que este método introduce una sobrecarga computacional significativa en comparación con Adam.
- **Comparación entre tareas (LP vs NC)**: En general, como era predecible, los tiempos de entrenamiento para la tarea de predicción de enlaces son mayores que para la tarea de clasificación de nodos. Esto es por la diferencia en los tamaños de los datos para las tareas, ya que para predicción de enlaces hay que tener en cuenta los enlaces existentes, así como los inexistentes.

En conclusión, aunque el modelo $\mathcal{Q}$GCN radam suele ofrecer un rendimiento superior, estos beneficios vienen a costa de tiempos de entrenamiento significativamente más altos. Por otro lado, GCN proporciona una opción eficiente en términos de tiempo, aunque a veces a expensas de una menor precisión. GAT se encuentra en un punto intermedio, ofreciendo un balance entre precisión y tiempo de entrenamiento.

11.1.4. Visualizaciones del espacio latente

En este apartado presentamos varias visualizaciones del espacio latente aprendido para los modelos GCN y $\mathcal{Q}$GCN radam en 2 casos: para el dataset Citeseer en predicción de enlaces y para el dataset Disease en clasificación de nodos. Para cada caso hacemos dos gráficos, en el primero utilizamos la técnica de reducción de dimensionalidad PCA para reducir la dimensión del embedding a 3 y luego proyectar esos datos sobre la esfera, visualizando la esfera desde dos puntos opuestos para conseguir observar todos sus puntos. El segundo gráfico consiste en reducir el embedding a 2 dimensiones con los métodos PCA o UMAP.

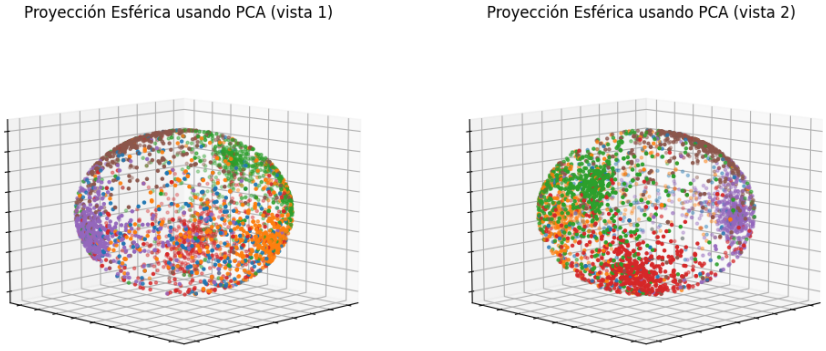


Figura 11.9.: Visualización esférica del espacio latente de $\mathcal{Q}$GCN radam en Citeseer. Cada color indica una clase.

En las imágenes observamos que, aunque la tarea no era clasificación de nodos, el espacio latente, especialmente en el modelo $\mathcal{Q}$GCN radam, muestra una notable separabilidad entre

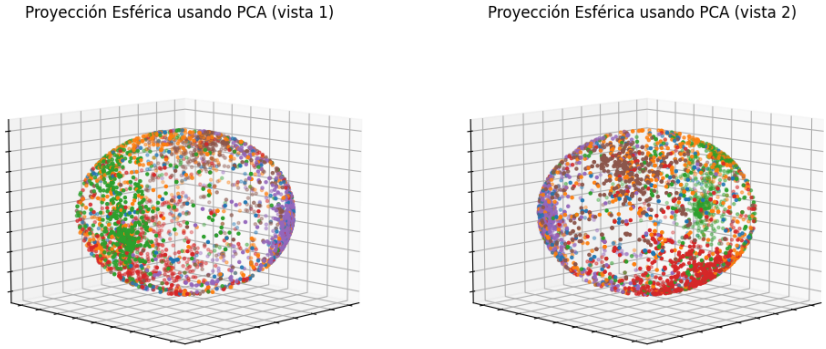
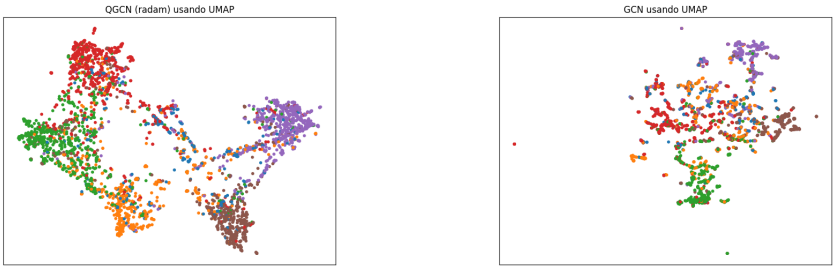


Figura 11.10.: Visualización esférica del espacio latente de GCN en Citeseer. Cada color indica una clase.



(a) $\mathcal{Q}$GCN radam Citeseer 2D (b) GCN Citeseer 2D

Figura 11.11.: Visualización 2D con UMAP del espacio latente en Citeseer. Cada color indica una clase.

las distintas clases. Esto sugiere que $\mathcal{Q}$GCN radam es capaz de capturar características estructurales que facilitan la diferenciación de las clases de nodos, incluso sin estar explícitamente entrenado para ello. En comparación, el modelo GCN también muestra cierta separabilidad, pero de manera menos pronunciada.

Por otro lado, en el dataset Disease para clasificación de nodos, destacamos que el espacio latente aprendido por ambos modelos (figuras 11.12, 11.13 y 11.14) consigue separar ambas clases de una manera casi perfecta, siendo la separación observada en $\mathcal{Q}$GCN radam levemente superior.

En resumen, las visualizaciones del espacio latente nos ofrecen otra comparativa entre los modelos, la cual proporciona una evidencia visual de la eficacia de $\mathcal{Q}$GCN radam para capturar estructuras complejas en los datos de redes.

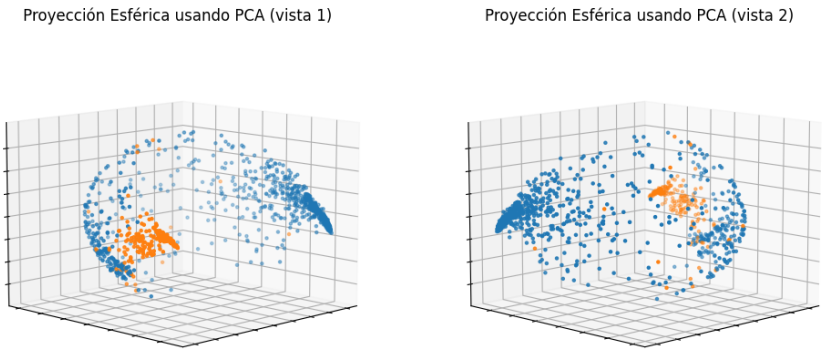


Figura 11.12.: Visualización esférica del espacio latente de $\mathcal{Q}$GCN radam en Disease. Cada color indica una clase.

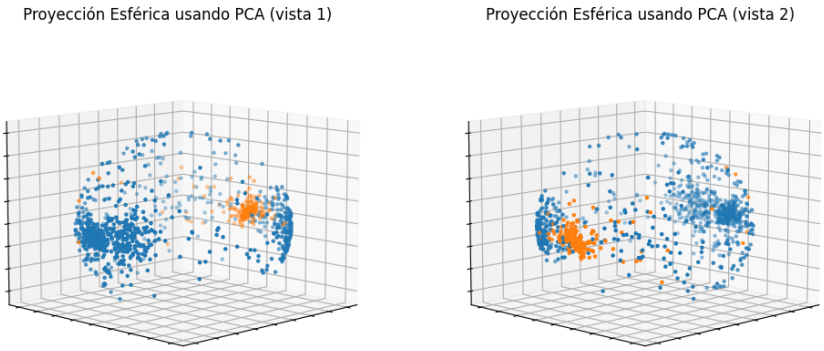
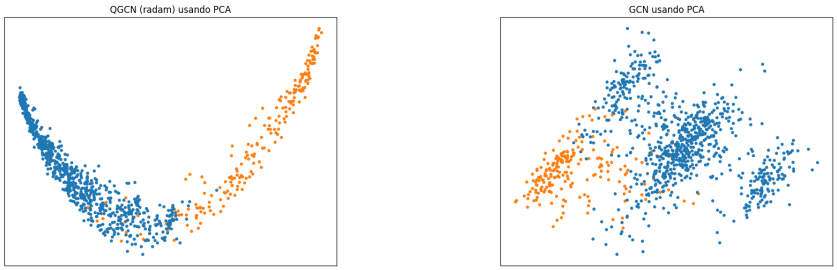


Figura 11.13.: Visualización esférica del espacio latente de GCN en Disease. Cada color indica una clase.



(a) $\mathcal{Q}$GCN radam Disease 2D (b) GCN Disease 2D

Figura 11.14.: Visualización 2D con PCA del espacio latente en Disease. Cada color indica una clase.

11.2. Conclusiones y trabajo futuro

Finalmente, se discuten las conclusiones obtenidas al terminar este proyecto junto con una serie de propuestas de líneas de trabajo futuros que se podrían realizar más adelante para expandir esta investigación.

Este trabajo nos ha proporcionado todos los conceptos necesarios para comprender las redes neuronales para grafos. Se han estudiado sus fundamentos teóricos y sus distintas aplicaciones prácticas. También se han introducido las arquitecturas más usuales de este tipo de red y se ha analizado en detalle un innovador modelo de red: la red pseudo-hiperbólica neuronal convolucional para grafos ($\mathcal{Q}$GCN). Este modelo es una extensión de las redes neuronales convolucionales para grafos diseñado específicamente para trabajar sobre pseudo-hiperboloides, una familia de subvariedades en $\mathbb{R}^d$. Se han realizado experimentos para comparar $\mathcal{Q}$GCN, GCN y GAT. Asimismo, se han estudiado dos métodos distintos de optimización para $\mathcal{Q}$GCN.

La parte matemática nos ha ofrecido todos los fundamentos para entender la teoría de las variedades pseudo-riemannianas, con un enfoque particular en los pseudo-hiperboloides. Se han investigado sus propiedades, curvatura y geodésicas, entre otros aspectos. Además, en esta parte se han abordado las dificultades de introducir una distancia en este tipo de variedades y se han explorado algunas soluciones posibles. Esta teoría nos ha proporcionado la base para comprender los dos métodos de optimización discutidos en el Capítulo 9.

Concluimos que $\mathcal{Q}$GCN es un modelo de red neuronal para grafos con un gran potencial para trabajar con datos estructurados en grafos, capaz de capturar relaciones y jerarquías que otros modelos estudiados no consiguen. Adicionalmente, se ha visualizado cómo estos modelos capturan las relaciones entre los nodos de los grafos y los agrupan, incluso cuando la tarea de entrenamiento no era específicamente la clasificación de nodos.

Teniendo en cuenta los resultados que se han obtenido, generalmente positivos, se proponen las siguientes líneas de trabajo futuro:

- Explorar otras tareas computacionales aplicables a las redes neuronales para grafos, como la clasificación de grafos, detección de comunidades y generación de grafos. Esto permitirá evaluar la efectividad de $\mathcal{Q}$GCN en diversos problemas.
- Realizar experimentos variando otros parámetros disponibles en la arquitectura, tales como la dimensión del embedding y las dimensiones espacial y temporal del pseudo-hiperboloide, para conseguir maximizar el rendimiento del modelo.
- Adaptar el método de optimización pseudo-riemanniana a otros optimizadores como Adadelta y Adamax, con el objetivo de investigar si estos optimizadores pueden ofrecer mejoras adicionales en el rendimiento de $\mathcal{Q}$GCN.
- Definición de pseudo-distancias alternativas y nuevos métodos de optimización para los pseudo-hiperboloides

Bibliografía

[BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, y Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, 2021.

[BEE96] J.K. Beem, P. Ehrlich, y K. Easley. *Global Lorentzian Geometry, Second Edition*. Chapman & Hall/CRC Pure and Applied Mathematics. Taylor & Francis, 1996.

[Con24] Orbifold Consulting. The Cora Dataset. https://graphsandnetworks.com/the-cora-dataset, 2024. [Recurso online, accedido el 20 de junio de 2024].

[CYRL19] Ines Chami, Zhitao Ying, Christopher Ré, y Jure Leskovec. Hyperbolic graph convolutional neural networks. En *Advances in Neural Information Processing Systems*, páginas 4869–4880, 2019.

[DBV17] Michaël Defferrard, Xavier Bresson, y Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, 2017.

[dS16] Daniella Van der Spiegel. The Hopf-Rinow Theorem. 2016.

[FCER14] Pablo Fleurquin, B. Campanelli, Víctor Eguíluz, y Jose Javier Ramasco. Trees of reactionary delay: Addressing the dynamical robustness of the US air transportation network, 11 2014.

[Fro12] G. Frobenius. *Über Matrizen aus nicht negativen Elementen*. Preussische Akademie der Wissenschaften Berlin: Sitzungsberichte der Preußischen Akademie der Wissenschaften zu Berlin. Reichsdr., 1912.

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193 – 202, 1980. Cited by: 3480.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, y Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[GJ19] Hongyang Gao y Shuiwang Ji. Graph U-Nets, 2019.

[GLY18] Tingran Gao, Lek-Heng Lim, y Ke Ye. Semi-Riemannian Manifold Optimization, 2018.

[GMS05] M. Gori, G. Monfardini, y F. Scarselli. A new model for learning in graph domains. En *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volumen 2, páginas 729–734 vol. 2, 2005.

[GSR+17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, y George E. Dahl. Neural Message Passing for Quantum Chemistry, 2017.

[JL22] Weiwei Jiang y Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 207:117921, 2022.

[KPK+10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, y Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3), September 2010.

[KS24a] M. Kunzinger y R. Steinbauer. Lorentzian Geometry. https://diana.mat.univie.ac.at/teaching/skripten/lorentzian.pdf, 2024. [Recurso online, accedido el 13 de mayo de 2024].

[KS24b] M. Kunzinger y R. Steinbauer. Riemannian Geometry. https://www.mat.univie.ac.at/~stein/teaching/skripten/rg.pdf, 2024. [Recurso online, accedido el 13 de mayo de 2024].

[KW16] Thomas N. Kipf y Max Welling. Variational Graph Auto-Encoders, 2016.

[KW17] Thomas N. Kipf y Max Welling. Semi-Supervised Classification with Graph Convolutional Networks, 2017.

[LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, y L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[LLK19] Junhyun Lee, Inyeop Lee, y Jaewoo Kang. Self-Attention Graph Pooling, 2019.

[LS21] Marc T. Law y Jos Stam. Ultrahyperbolic Representation Learning, 2021.

[MS08] E. Minguzzi y M. Sanchez. The causal hierarchy of spacetimes, 2008.

[MT21] Yao Ma y Jiliang Tang. *Deep Learning on Graphs*. Cambridge University Press, 2021.

[MTSvdH15] Julian McAuley, Christopher Targett, Qinfeng Shi, y Anton van den Hengel. Image-based Recommendations on Styles and Substitutes, 2015.

[MWAT19] Yao Ma, Suhang Wang, Charu C. Aggarwal, y Jiliang Tang. Graph Convolutional Networks with EigenPooling, 2019.

[Net24] NetworkX. NetworkX — Network Analysis in Python. https://networkx.org, 2024. [Recurso online, accedido el 1 de julio de 2024].

[O'N83] Barrett O'Neill. *Semi-Riemannian Geometry With Applications to Relativity, 103, Volume 103 (Pure and Applied Mathematics)*. Academic Press, 1983.

[Per07] Oskar Perron. Zur Theorie der Matrices. *Mathematische Annalen*, 64:248–263, 1907.

[PT00a] Paolo Piccione y Daniel V. Tausk. On the Distribution of Conjugate Points along semi-Riemannian Geodesics, 2000.

[PT00b] Paolo Piccione y Daniel V. Tausk. The Morse Index Theorem in semi-Riemannian Geometry, 2000.

[SAV20] Eli Stevens, Luca Antiga, y Thomas Viehmann. *Deep Learning with PyTorch*. 2020.

[SMBG18] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, y Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[Sta24] Stanford. Machine Learning with Graphs. https://web.stanford.edu/class/cs224w/, 2024. [Recurso online, accedido el 23 de abril de 2024].

[VC10] M.A.J. Victoria y M.S. Caja. *An Introduction to Lorentzian Geometry and Its Applications: XVI Escola de Geometria Diferencial [de 12 a 16 de Julho de 2010, Universidade de São Paulo]*. RiMa, 2010.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, y Yoshua Bengio. Graph Attention Networks, 2018.

[WLH24] Jure Leskovec William L. Hamilton, Rex Ying. Graph SAGE: Inductive Representation Learning on Large Graphs. https://snap.stanford.edu/graphsage/, 2024. [Recurso online, accedido el 20 de junio de 2024].

[XZP+22a] Bo Xiong, Shichao Zhu, Nico Potyka, Shirui Pan, Chuan Zhou, y Steffen Staab. Pseudo-Riemannian Graph Convolutional Networks, 2022.

[XZP+22b] Bo Xiong, Shichao Zhu, Nico Potyka, Shirui Pan, Chuan Zhou, y Steffen Staab. Pseudo-Riemannian Graph Convolutional Networks. En *Advances in Neural Information Processing Systems*, 2022.

[YHC+18] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, y Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. En *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18. ACM, July 2018.

[YIS19] Muhamad Yani, S Irawan, y Casi Setianingsih. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019.

[YYM+19] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, y Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling, 2019.

[ZC18] Muhan Zhang y Yixin Chen. Link Prediction Based on Graph Neural Networks, 2018.