Adaptive curvature exploration in Non Euclidean Graph Neural Networks

An adaptation on Point-GNN for Object Detection in Point Clouds

Parfait R. Fejou

Abstract—We propose Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt

urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

2 GRAPH THEORY

With graphs playing a crucial role in the topic of discussion, we can't ignore taking a closer look at the history and theory behind these objects. In attempting to do so, we shall not dive into, but brush over the field of Graph Theory. What are these objects, where do they come from and why should we care about them? Graph theory is a branch of mathematics that studies the relationships between objects. It represents these relationships in terms of vertices and edges that connect pairs of vertices. A graph is essentially a collection of vertices and edges that together describe a network of connections. It originated in the 18th century with the work of swiss mathematician Leonhard Euler, who solved the problem of the Seven Bridges of Königsberg. This problem - in particular Euler's approach to solving it - laid the foundation for modern Graph Theory. Since then the field has expanded significantly and is now a vital part of discrete mathematics, playing a fundamental role in - for example - Wolfram Physics Project. Anyone who has listened to Jonathan Gorard elaborate *Multiway Causal Graphs* knows how quickly talking about graphs can get dense and theoretical. The reader shall be spared this experience, as only the most fundamental aspects of graph theory as presented in [Die17; Chu09; Cai19] shall be elaborated on.

2.1 The Basics

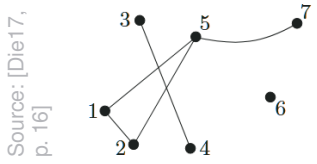


Figure 1: A graph given by $V = \{1,\dots,7\}$, $E = \{\{1,2\},\{1,5\},\{2,5\},\{3,4\},\{5,7\}\}$

Definition 2.1 (Graph)**.** *A* graph *is a pair* $G = (V, E)$ *of sets such that* $E \subseteq [V]^2$

Two vertices $x, y \in V(G)$ are *adjacent*, or neighbours, if $\{x, y\}$ is an edge of G. Two edges $e \neq f$ are *adjacent* if they have an edge in common. If all vertices of G are pairwise adjacent, G is said to be *complete*. Pairwise non-adjacent vertices or edges are called *independent*. A complete graph

with n vertices is denoted K^n. It can easily be seen that $K^3 \Leftrightarrow$ *triangle*.

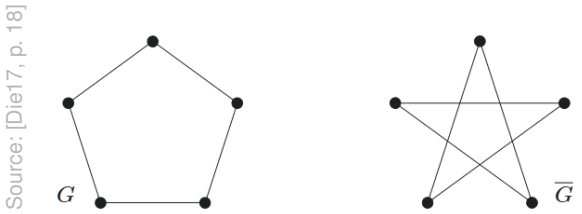


Figure 2: A path P^6 in G

Definition 2.2 (Degree of a vertex)**.** *Let* $G = (V, E)$ *be a (non-empty) graph. The set of neighbours of a vertex* v *in* G *is denoted* $N_G(v)$*. More generally for* $U \subseteq V$*, the neighbours in* $V \setminus U$ *are called* neighbours *of* U*.*

The degree $d_G(v) = d(v)$ *of vertex* v *is the number* $|E(v)|$ *of edges coinciding with* v*. This means that a a vertex of degree* 0 *is* isolated*. The number* $\delta(G) := \min\{d(v) | v \in V\}$ *is the* minimum degree *of* G*, whereas* $\Delta(G) := \max\{d(v) | v \in V\}$ *is the* maximum degree*. If* $d(v) \equiv k \in \mathbb{R}\ \forall v \in V$*, then* G *is* k-regular

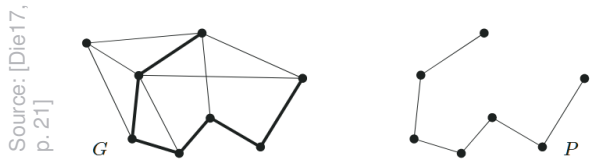


Figure 4: A path P^6 in G

Definition 2.3 (Paths & cycles)**.** *A* path *is a non-empty graph* $P = (V, E)$ *of the form*

$$V = \{x_0, x_1, \ldots, x_k\} \qquad E = \{x_0x_1, x_1, x_2, \ldots, x_{k-1}x_k\}, \quad (1)$$

where $(x_i)_i$ *are all distinct. The vertices* x_0 *and* x_k *are* linked *by* P *and referred to as the* endvertices *and all others are the* inner *vertices of* P*. The number of vertices in* P *is its length and a path of length* k *is denoted* P^k*. A path is a* cycle *if* $x_0 = x_k$*.*

If a graph has a high minimum degree, it contains long paths and cycles.

Definition 2.4 (Connectivity)**.** *A graph* G *is called* connected *if it is non-empty and any two of its vertices are linked by a path in* G*.*

Proposition 2.1. *The vertices of a connected graph* G *can always be enumerated, say* $v_1 \ldots, v_n$*, so that* $G_i := G[v_1, \ldots, v_i]$ *is connected* $\forall\ i$

Proof. Pick any vertex as v_1, and assume iductively that $v_1, \ldots, v_i$ have been chosen for some $i < |G|$. Now pick a vertex $v \in G - G_i$. As G is connected, it contains a $v - v_i$ path P. Choose as v_{i+1} the last vertex of P in $G - G_i$. Then v_{i+1} has a neighbour in G_i. The connectedness of every G_i follows by induction on i. □

Definition 2.5 (Trees & forests)**.** *An* acyclic *graph is called a* forest*. A connected forest is called a* tree*.*

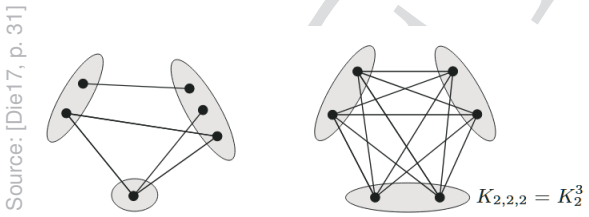


Figure 5: Two 3-partite graphs

Definition 2.6 (Bipartite graph)**.** *Let* $\mathbb{N} \ni r \geqslant 2$*. A graph* $G = (V, E)$ *is called* r-partite *if* V *admits a partition into* r *classes, such that no two edges belonging to the same class are adjacent. An* r*-partite graph in which every two vertices from different partitions are adjacent is called* complete *as shown in the right graph in fig. 5.*

2.2 Euler Tours

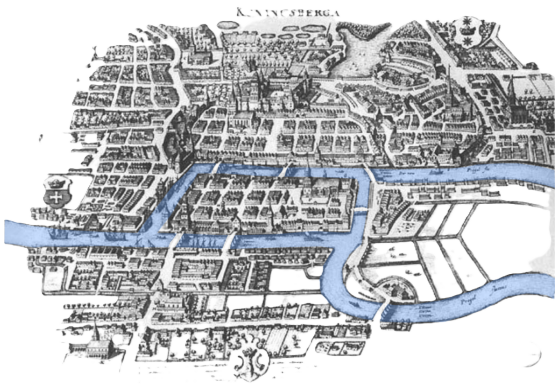
Euler tours arose from the famous Königsberg bridge problem, a recreational mathematical puzzle set in the old Prussian city of Königsberg (now Kaliningrad), which led to the development of topology and graph theory. In the early 18th century, the citizens of Königsberg spent their days walking on the intricate arrangement of bridges - 7 in total - across the waters of the Pregolya River, which surrounded two central landmasses connected by a bridge, as illustrated in fig. 3. The question arose whether it was possible to take a walk through the town in such a way that each bridge would be crossed exactly once. In 1735 Leonhard Euler proved that there was indeed no solution to this problem and in doing so gave birth to prior mentioned fields - a beautiful illustration of the German phrase *"Der Weg ist das Ziel"*.

Inspired by this problem, a closed walk in a graph a called an *Euler tour* iff it traverses every edge of the graph exactly once. A graph is *Eulerian* if it admits an Euler tour.

Theorem 2.1 (Euler 1736)**.** *A* connected *graph is* Eulerian *iff every vertex has even degree.*

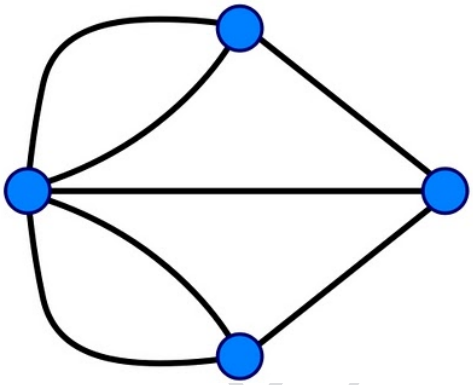
This seems too beautiful a story not to have its proof delivered, so deliver we will. In the following the number of edges shall be denoted by $\|G\|$ and the number of vertices by $|G|$.

Proof. A vertex appearing k times in an Euler tour must have degree $2k$. Conversely, we show by induction on $\|G\|$ that every connected graph G with all degrees even has an Euler tour. The induction starts with $\|G\| = 0$. Now let $\|G\| > 1$. Since all degrees are even, we find in G a non-trivial closed walk that contains no edge more than once. How? Let W be such a walk of maximal length, and write F



Sources: aprendiendomatematicas, Spektum

(a) Königsberg 18th century



(b) Graph representation

Figure 3: (a) shows the city at the time. (b) is the corresponding graph representation with bridges represented by edges

for the set of its edges. If $F = E(G)$, then W is an Euler tour. Suppose, therefore that $G' := G - F$ has an edge.

For every vertex $v \in G$, an even number of the edges of G at v lies in F, so the degrees of G' are again all even. Since G is connected, G' has an edge e incident with a vertex on W. By the induction hypothesis, the component C of G' contaianing e has an Euler tour. Concatenating this with W, we obtain a closed walk in G that contradicts the maximal length of W. □

2.3 Spectral Graph Theory

In §10.2, we shall make use of some of the properties of the adjacency matrix of the graph in order to mitigate undesired behaviour, such as putting too much weight on vertices that have a lot of neighbours. in this section, we shall scratch over the surface of spectral theory in the context of graphs. This section shall be mostly based on [Chu97; Cai19].

Spectral graph theory has a long history, being used in the early days of linear algebra and matrix theory to analyze adjacency matrices of graphs. Especially in the field of chemistry. eigenvalues were associated with the stability of molecules. Also grpah spectra arise naturally in various problems of theoretical physics and quantum mechanics, for example in minimizing energies of Hamiltonian systems.

In a graph G, let d_v denote the degree of the vertex v. We first define the *Laplacian* for graphs, by firstly defining the following matrix L:

$$L(u,v) = \begin{cases} d_v & \text{if } u = v \\ -1 & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This is equivalent to $L = D - A$, where A denotes the adjacency matrix. Let T denote the diagonal matrix with the (v,v)-th entry having value d_v. The *notmalized Laplacian* of G is defined to be the matrix

$$\mathcal{L}(u,v) = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We can write

$$\mathcal{L} = T^{-\frac{1}{2}} L T^{-\frac{1}{2}} \tag{4}$$

with he convention $T^{-\frac{1}{2}}(u,v) = 0$ for $d_v = 0$. in such a case, v is an isolated vertex.

Since $\mathcal{L}$ is symmetric, its eigenvalues are all real and non-negative. We can use the *Rayleigh quotient* of $\mathcal{L}$ to charactarize the eigenvalues. Let g denote an arbritrary function $g : V(G) \to \mathbb{R}, u \mapsto g(u)$, which acts on the vertices of the graph G.

$$\begin{aligned} \underline{\langle g, \mathcal{L} g \rangle} &= \frac{\langle g, T^{-\frac{1}{2}} L T^{-\frac{1}{2}} g \rangle}{g^T g} \\ &= \frac{\sum_{u \sim v} (f(u) - f(v))^2}{\sum_v f(v)^2 d_v} \end{aligned} \tag{5}$$

where $g = T^{-\frac{1}{2}} f$ and $\sum_{u \sim v}$ denotes the sum over all unordered pairs $\{u,v\}$ for which u and v are adjacent. $\langle f, g \rangle = \int_x f(x) g(x) dx$ denotes the standard inner product in $\mathbb{R}^n$. The largest eigenvalue of $\mathcal{L}$, also referred to as *spectral radius* is the supremum of the Rayleigh quotient. As we can see in the quotient, the degree of the vertex normalizes its importance in the Laplacian. This is important, because in order to capture the topology of a graph we need not necessarily assign vertices with more neighbours more significance. As can be seen from §10.2, we define T to be the degree matrix.x

3 Graphs in Machine Learning

The development of representation learning can broadly be summarized into four key areas

- Computer vision
- Speech recognition
- Natural language processing
- Network analysis

Assigning this paper to any of those seems rather unsatisfactory, but as a friend once said *"Syntax is without meaning"*, so we shall ponder over the problem at hand in terms of computer vision, but please do not nail me to this. Using this analogy, our image is the entirety of points in a point cloud, with the pixels being the points. Here our pixels position lies in $\mathbb{R}^3$ instead of $\mathbb{N}$. The feature is a real value representing the reflectivity at the point, so $\mathbb{R}$ instead of the rgb(a) representation for a colored image which lies in $\mathbb{R}^3$ or $\mathbb{R}^4$ if you take iinto acount opacity. Keep in mind that the term *feature* does not have the same meaning as in the usual machine learning context.

If creativity were quantifiable, it would be fair to say there is not and has never been any human innovation as the result of 100% creativity. All we do is take a close enough look at nature and let ourselves be inspired by it. Maybe this is a direct consequence of Plato's Allegory of the Cave. Our efforts in machine learning can be seen as the attempt to emulate the most effective and efficient computing machine known to man - the human brain. Advances are achieved by improving the emulators design (model architecture), as well as the sensors (data) we feed it. In our case we attempt to improve the *brains* capability to make sense of visual information by improving the architecture, and feeding it presumably richer data. In what sense richer? Well we need only think of the optical Illusions such as the one illustated in fig. 6, which only occurs due to the lack of information on depth.

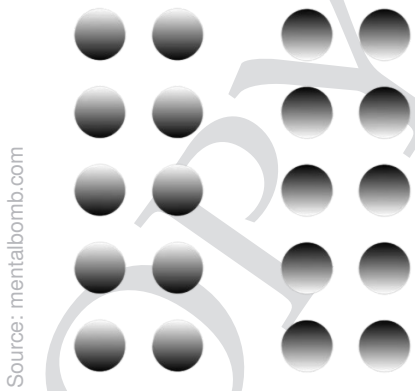


Figure 6: An optical illusion

3.1 Graph Embedding

Many complex systems take the form of graphs, such as social networks, biological systems and information networks. With graph data being often sophisticated and challenging to deal with, the first problem that arises is that of finding a suitable graph data representation, that is, how to represent the graphs consisely so that advanced analytic tasks can be conducted effectively in both space and time. Some of the problems we face when dealing with graphs in the context of machine learning are:

1) **High computational complexity:**
   The relationships encoded by the edges take most grpah processing or analysis algorithms either iterative or combinatorial computation steps. An example would be the the computation of the shortest path between two edges.

2) **Low parallelizability:**
   The problem here arises due to the fact that the connectivity between nodes are captured by the vertices, so distributing nodes in different shards or servers often causes demandingly high communication cost among servers, and posses a bottleneck to speed-up ratio.

3) **Inapplicability of machine learning methods:**
   Most machine learning methods assume independent data in some vector space, whereas graphs are explicitly desined to encouporate dependence

Traditional graph embedding methods were originally studied as dimension reduction techniques with a graph usually being constructed from a feature represented dataset, like image dataset. There are usually two goals to graph embedding, graph reconstruction and inference. The objective function of traditional graph embedding methods mainly target the goal of graph reconstruction.

Specifically [TSL00a] first constructs a neighbourhood graph G using algorithms such as k-nearest neighbours (KNN) - a topic that resembles §10.4. Then based on G, the shortest path between different data points can be computed. Finally, the classical multidimensional scalling (MDS) method is applied to the matrix to obtain the coordinate vectors. The representation learned by ISOMAP [TSL00b] approximately preserve the geodesic distances of the entry pairs in the low-dimensional space with the main problem being the computational cost §3.1 due to the need of pairwise computations. This was seemingly addressed by [RS00] which made the assumption that each entry and its neighbours lie on a or close to locally linear patch of a manifold. As mentioned in [Wu+22, p. 19], graph embeddings mostly works on graphs constructed from feature represented datasets, where the proximity among nodes encoded by the edge weights is well defined in the original feature space and while modern graph embedding mostly works on naturally arising networks, such as those mentioned in §3 or maybe even point cloud data. In those networks the proximities among nodes are not explicitly or directly defined as we shall see in the problem formulation **ADD REFERENCE TO FORMULATION**. For example, an edge between two nodes usually just implies there is a relationship between them, but cannot indicate the specific proximity and the lack of an edge does not necessarily imply the lack thereof. Therefore, moden graph embedding usually incorporates rich information, such as network (graph) structures, properties, side information, to facilitate different problems and applications.

3.2 Graph Neural Networks

At some point in the distant past, a young boy and his class were invited to Audi's headquarters in Ingolstadt to present a physics project. A talk on computer vision, marketed under the premise *"We are teaching cars to see"*, was held. Funny enough the young boy could not have been any less interested in the topic. Instead his full amazement was dedicated to the new headlights that would adapt to turns taken. That was real sci-fi to him. Somewhat analogous to Neil deGrasse Tyson saying, the most sci-fi thing in Star Trek to him where the self-opening doors. Well a decade later, the field of automotive automation is maybe not the most powerful, but fair to say amongst the most influential forces pushing research in graph neural networks. Although at the time (conventional) neural network architectures were being used for such tasks, the field has now exploded into a standalone research area. Although the selected field might not rival successes such as the unlocking of protein folding or other developments from bioinformatics, a field from which one can expect great achievements in the near future, it remains the most relatable, as almost a decade later that very same young boy - now a man - would be writing his thesis on the advances in said field. The release of the PyTorch-Geometric package or the fact that there exists a graph counterpart for almost every neural network architecture you can think of supports the claim of these architectures significance.

Despite the successes of current research, GNNs still face many challenges when used to model highly-structured data that is time evolving, multi-relational, and multi-modal. It is also very difficult to model mapping between graphs and other highly structured data, such as sequences, trees, and even other graphs. One challenge with graph-structured data lies in them not displaying as much spacial locality and structure as image or text data does. Thus, graph-like data is not naturally suitable for highly regularized neural structures such as convolutional and recurrent neural networks.

The main tasks GNNs tackle can be broadly grouped into the following:

- Node classification
- Link prediction
- Graph classification

The task at hand can be assigned to the former, as we shall see in §9.

3.2.1 General Framework of Graph Neural Networks

In this section we introduce a general framework of graph neural networks as presented in [Xu+19]. The general idea of GNNs is to iteratively update the node representations by aggregating its neighbours representations. In *node classification*, each node $v \in V$ has an associated and learnable label $y_v = f(\mathbf{h}_v)$. In graph classification we are given a set of graphs $\{G_1, \ldots, G_N\} \subseteq G$ an their label $\{y_1, \ldots, y_N\} \subseteq \mathcal{Y}$ an we aim to learn a representation vector $\mathbf{h}_G$ that helps predict the label of an entire graph, $y_G = f(\mathbf{h}_G)$.

Definition 3.1 (Graph Neural network)**.** *GNNs use a graph structure and node features X_v to learn a representation vector $\mathbf{h}_v$ of a node, or an entire graph $\mathbf{h}_G$. Modern GNNs follow a neighbourhood aggregation strategy, where we iteratively update the representation of a node by aggregating representations of its neighbours. After k iterations of aggregation, a node's representation captures the structural information within its k-hop neighbourhood. Formally, the k-th layers output is given by:*

$$\begin{aligned} a_v^{(k)} &= \text{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_u^{k-1} : u \in \mathcal{N}(v)\right\}\right), \\ \mathbf{h}_v^{(k)} &= \text{COMBINE}^{(k)}\left(\mathbf{h}_v^{k-1}, a_v^{(k)}\right), \end{aligned} \tag{6}$$

where $\mathbf{h}_v^{(k)}$ is the feature vector of node v at the k-th iteration/layer. We initialize $\mathbf{h}_v^{(0)} = X_v$, and $\mathcal{N}(v)$ is a set of nodes adjacent to v. The choice of $\text{AGGREGATE}^{(k)}$ *and* $\text{COMBINE}^{(k)}$ *in GNNs is crucial.*

A number of architectures for AGGREGATE have been proposed. In the pooling variant of GraphSAGE [HYL18], AGGREGATE is formulated as

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}(\boldsymbol{W} \cdot \mathbf{h}_u^{(k-1)}), \forall u \in \mathcal{N}(v)\right\}\right), \tag{7}$$

where $\boldsymbol{W}$ is a learnable matrix, and MAX represents an element-wise max-pooling. The COMBINE step could be a concatenation followed by a linear mapping $\boldsymbol{W} \cdot \left[\mathbf{h}_v^{(k-1)}, a_v^{(k)}\right]$ as in GraphSAGE. In Graph Convolution Networks (GCN) [KW17], the element-wise *mean* pooling is used instead, and the AGGREGATE and COMBINE steps are integrated as follows:

$$\mathbf{h}_v^{(k)} = \text{ReLU}\left(\boldsymbol{W} \cdot \text{MEAN}\left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right). \tag{8}$$

As mentioned, there exist a multitude of GNN architectures which are all based on eq. (6).

For node classification, the node representation $\mathbf{h}_v^{(K)}$ of the final layers output is used for prediction. For graph classification, the READOUT function aggregates node features from the final iteration to obtain the entire graph's representation $\mathbf{h}_G$:

$$\mathbf{h}_G = \text{READOUT}\left(\left\{\mathbf{h}_v^{(K)} \mid v \in G\right\}\right) \tag{9}$$

3.3 Performance capabilities of a GNN

Ideally, a maximally powerful GNN could distinguish different graph structures by mapping them to different representations in the embedding space. This ability to map any two different graphs to different embeddings, hovever, implies solving the challenging graph isomorphism problem described in [Xu+19, p. 4]. Thus this criterion is replaced by the slightly weaker *Weisfeiler-Lehman graph isomorphism test*,[Dou11] which is known to work well in general with a few exceptions.

Lemma 3.1. *Let G_1 and G_2 be any two non-isomorphic graphs. If a graph neural network $\mathcal{A} : G \to \mathbb{R}^d$ maps G_1 and G_2 to different*

embeddings, the Weisfeiler-Lehman graph isomorphism test also decides G_1 and G_2 are not isomorphic.

Theorem 3.1 (Weisfeiler-Lehman Graph Isomorphism Test)**.** *Let $\mathcal{A} : G \to \mathbb{R}^d$ be a GNN. With a sufficient number of GNN layers, $\mathcal{A}$ maps any graphs G_1 and G_2 that the Weisfeiler-Lehman test of isomorphism decides are non-isomorphic, to different embeddings if the following conditions hold:*

(a) $\mathcal{A}$ aggregates and updates node features iteratively with

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right),$$

where the function f, which operates on multisets and ϕ are injective.

(b) $\mathcal{A}$'s graph-level readout, which operates on the multiset of node features $\left\{h_v^{(k)}\right\}$ is injective

lemma 3.1 and theorem 3.1, for which the proofs can be found in the section A, are just a snippet of what can be said about the capabilities of GNNs. For those, who's needs were not met with the very brief coverage in this section, we refer to [Xu+19, p. 5], where *Graph isomorphic network* are also introduced a maximally expressive GNN framework.

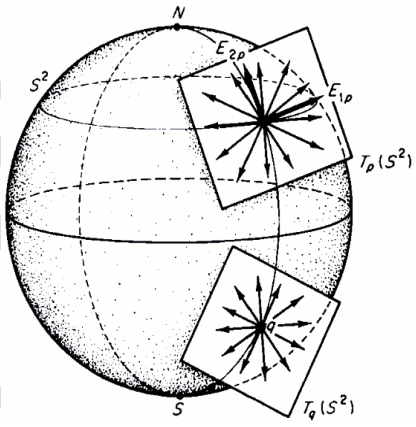
4 Riemannian Manifold

While SVMs use a kernel to do computation in low dimensional space, that corresponds to a higher dimensional embedding, the idea here is not really similar, but somewhat relateable. We assume that the data lives on some manifold and train different kinds of models whose job is to find a desciption of said manifold. However this assumes that all representive manifolds that usefully approximate the sample set live in Euclidean space. However the data could as well be embedded on some manifold in some space very different from the known Euclidean. Also Euclidean space makes it very difficult to capture incorporate concepts such as hierarchical dependencies, which come naturally with others. In order to break this down, lets consider a simple case. Lets disregard the reflectivity of a point for a second and just consider its three dimensional coordinates as input. If there is a learnable function $f : \mathbb{R}^3 \to \mathbb{N}$ that maps the coordinates of a point to its integer valued class, we can think of it as a heatmap in 3D space, which corresponds to a Surface in 4D space. Now this is all fine and good, but what if some labeled structure in the graph where made up of other structures which we were interested in capturing as well. Euclidean space makes capturing such dependencies rather contraintuitive, whereas mapping the surface (*manifold*) onto - for example - a hyperbolic space in a meaningful way would allow for the retrieval of such information. Before understanding the space we will be working with we shall first introduce the objects our data is assumed to be living on, manifolds. Some referrences to topology are made in the attempt to clearly define what a manifold, and specifically a Riemannian manifold, is. In case of any unclear notions, we recommend [Mun00] where the subject is eased into in a comprehensive and yet comprehensible manner.

Definition 4.1 (Manifold)**.** *A manifold $\mathcal{M}$ is a metric space (X, g), that satisfies the following properties [Boo75, p. 6]:*

- $x \in X \Rightarrow \exists\, B_\epsilon(x) =: U, n \geqslant 0 : U \cong \mathbb{R}^n$
- X *is a* Hausdorff space
- X *has a finite Basis*

So a manifold is the union of open sets. We shall refer to the dimension of $\mathcal{M}$ as $\dim_{\mathcal{M}}$. When $\dim_{\mathcal{M}} = 0$, then $\mathcal{M}$ is a countable space with the discrete topology. $\mathcal{M}$ being locally Euclidean is equivalent to it locally resembling the open sphere of dimension $\dim_{\mathcal{M}}$, which in $\mathbb{R}$ is an open interval, in $\mathbb{R}^2$ an open disk, in $\mathbb{R}^3$ an open sphere, in $\mathbb{R}^4$ an open hypersphere of dimension 4 and so on and so forth.



Source: [Boo75, p. 16]

Figure 7: S^2 and some elements of $T(S^2)$

Let us consider for a moment the unit sphere S^2 in $\mathbb{R}^3$, as illustarated in fig. 7. The collection of all tangent vectors to points of S^2 are denoted $T(S^2)$. The union $\cup_{p\in S^2} T(S^2)$ is a manifold called *tangent bundle* of S^2. $T(S^2)$ is a four dimensional object as 4 different coordiantes are needed to uniquely identify a point on it - that is , if we are using polar coordinates. Two being the coordinates of p and two being the coordinates on the thereby fixed tangent plane T_p. For the sake of generalization we shall from now on consider an abritarary manifold $\mathcal{M}$. Note that every point on $\mathcal{M}$ is an element of the tangent bundle. The question arises how to introduce a metric on $\mathcal{M}$. As the inner product induces a metric, we shall atempt to define an inner product on $\mathcal{M}$ by defining it on its tangent bundle $T(\mathcal{M})$.

Definition 4.2 (Inner product)**.** *The inner product over a vector field V is a map $V \times V \to F$, denoted by $(v, w) \mapsto \langle v, w\rangle$, that satisfies the following properties.*

- Symmetry*:* $\langle v, w\rangle = \langle w, v\rangle$
- Linearity*:* $\langle au + bv, w\rangle = a\langle u, w\rangle + b\langle v, w\rangle$
- Positive definiteness*:* $\langle v, v\rangle \geqslant 0\ \forall x \neq 0$

With the smooth n-dimensional manifold $\mathcal{M}$ given, a *Riemannian metric* $g = (g_p)_{p\in\mathcal{M}}$ on the tangent space of $\mathcal{M}$, is a bilinear functional $(\langle .,.\rangle_p)_{p\in M}$ that depends smoothly on p [Pet16]. Such a manifold shall be referred to as a *Riemannian manifold*. So on a Riemannian manifold, the inner product is locally defined and the *Riemannian metric* is given by:

$$g_x = \sum_{ij} g_{ij}\, dx_i \otimes dx_j,$$
$$g_{ij}(p) := \left\langle \left(\frac{\partial}{\partial x_i}\right)_p, \left(\frac{\partial}{\partial x_j}\right)_p \right\rangle_p \tag{10}$$

We can immediately see, given the definition of the inner product, that the metric tensor g_{ij} must be symmetric. With this, we have a tool at hand to define distances on $\mathcal{M}$. For the standard *Euclidean metric* on $\mathbb{R}^n$, g_x is given by $\sum_i dx_i^2$, meaning that $g_{ij} = I_n$. An example for a different kind of manifold would be the one we live in. In *Minkovski space*, which plays a fundamental role in Einsteins *G*- and *SR*, the metric tensor is given by $\pm diag(-1,1,1,1) \cdot dx^2$. A direct consequence of the definitions given is that $\mathbb{R}^n$ is a *Riemannian manifold*. The distance between any $p_1, p_2 \in \mathcal{M}$ is given by

$$d(p_1,p_2) = \inf_x \int_0^1 \sqrt{g_{x(t)}}\, dt \tag{11}$$

with $x(0) = p_1$ and $x(1) = p_2$.

Another perhaps more accessible use of the metric tensor would be the measure of distances on the surface of the earth. Lets assume for a moment the earth was perfectly round with an everything ar sea level. Let us further more use spherical coordinates (θ, ϕ) to identify points on the its surface. We can easily see that the distance between two meridians varies across parallels. If we we had in front of us a globe, that we were to cut into infinitely small pieces and place those pieces on a rectangular piece of paper in a stretched form, so that the entire glob would fill out the sheet of paper, we would see more directly the distortion of distances. So in order to measure distances correctly, we need to introduce the metric tensor

$$g_{ij} = R^2 \begin{bmatrix} 1 & 0 \\ 0 & \sin^2(\theta) \end{bmatrix} = \begin{bmatrix} g_{\theta\theta} & g_{\theta\phi} \\ g_{\phi\theta} & g_{\phi\phi} \end{bmatrix}$$

whereas is we considered the reality of the earth not being a perfect sphere but an oblate spheroid, we obtain

$$g_{\theta\theta} = \frac{(a^2\cos\theta)^2 + (ab\sin\theta)^2}{a^2}$$
$$g_{\phi\phi} = (a\cos)^2\theta$$
$$g_{\theta\phi} = 0,$$

where a, b are the equatorial and polar radii respectively. This gives us the infinitisimal distance between two "neighbouring" points in terms of the change in θ, ϕ as $ds^2 = \left(\frac{(a^2\cos\theta)^2 + (ab\sin\theta)^2}{a^2}\right) a^2 d\theta^2 + a^2\cos^2\theta d\phi^2$.

5 Geometry

' Call me Euclid of Alexandria. Some milleania ago - nevermind how long precisely - I laid the foundation for that without which equations such as $\delta S = 0$ or $G_{\mu\nu} + \Lambda g_{\mu\nu} = \kappa T_{\mu\nu}$ would mean as much to man as Wolframs rule 30 *and yet, I remain but a mistery to you. '*

It would be remiss to talk about Non Euclidean Geometry without a mention of Euclid's *Elements*. Luckily it proves itself crucial as humans natural understanding of geometry is very much Euclidean. So we shall build our understanding of Non Euclidean geometry by exploring what it is not and studying the consequenses thereof. In prior mentioned treatise, Euclid attempts a systematic development of geometry, starting with the definition of two object classes

- *A point is that which has no part*
- *A line is breadthless length*

as well as the following 5 postulates

Definition 5.1 (Euclids Postulates)**.** *1) To draw a straight line from any point to any other.*

2) To produce a finite straight line continuously in a straight line.

3) To describe a circle with any center and distance.

4) That all right angles are equal to one another.

5) That, if a straight line falling on two straight lines makes the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on the side on which the angles are less than two right angles.

The fifth *postulate which is nowadays often substituted by* Playfairs Axiom*[Har15, p. 10] which addresses the issue of parallel lines more directly*

In comparison *Fano's geometry*[Har15, p. 2] looks very different. In his, there are three object classes consisting of *points*, *lines* and *ons* with the axioms:

Definition 5.2 (Fano's postulates)**.** *1) There exists at least one line.*

2) There are exactly three points on each line.

3) Not all points are on the same line.

4) There is exactly one line on any two distinct points.

5) There is at least one point on any two distinct lines.

Both represent a geometry, which is clearly defined in [Pri19, p. 4] as a pair $G = (\Omega, I)$, where Ω represents a set with $|\Omega|$ being the number of object classes and I a symmetric and reflexive relation between said object classes.

5.1 Neutral Geometry

In this chapter, which is mostly based on [Har15] Part I & II, we shall induce the properties of *Lobachevskij geometry*. First we shall define a basic geometries which will then be split into Euclidean and non Euclidean. The *neutral geometry* which doesn't necessarily satisfy the fifth postulate is based on Hilberts work in the early 20th century. In [Hil22] he defines 5 axiomatic groups, which where adapted as follows:

- *Axioms of incidence*:

In1: There is a unique line on any two distinct points

In2: There are at least two points on any line

In3: There are at least three points that do not all lie on the same line

- *Axioms of order*: The notation $A * B * C$ indicated B lies between A and C.

Or1: If $A * B * C$,then A,B,and C are distinct collinear points,and $C * B * A$.

Or2: For any two distinct points B and D, there are points A,Cand E,such that $A * B * D$, $B * C * D$,and $B * D * E$

Or3: Of any three distinct points on a line,exactly one lies between the other two.

Or4: Given a line l and points A,B,and C that are not on l. If A and B are on the same side of l and A and C are on the same side of l, then B and C are on the same side of l. If A and B are not on the same side of l and A and C are not on the same side of l,then B and C are on the same side of l.

- *Axioms of congruence*:

Cg1: If A and B are distinct points and if A' is any point, then for each ray r with endpoint A', there is a unique point B' on r such that $AB' \simeq A'B'$.

Cg2: Segment congruence is reflexive, symmetric and transitive.

Cg3: If $A * B * C$ and $A' * B' * C'$, and if $AB' \simeq A'B'$ and $BC' \simeq B'C'$,then $AC' \simeq A'C'$

Cg4: Given $\angle BAC$ and $A'\vec{B}'$, there is a unique $A'\vec{C}'$ on a given side of $A'\vec{B}'$ such that $\angle BAC \simeq \angle B'A'C'$.

Cg5: Angle congruence is reflexive, symmetric and transitive

Cg6: Given $\triangle ABC$ and $\triangle A'B'C'$, if $AB \simeq A'B'$, $\angle B \simeq LB'$, and $BC \simeq B'C'$, then $\angle A \simeq LA'$.

- *Axioms of continuity*:

Ct1: If AB and CD are two segments, there is some positive integer n such that n congruent copies of CD constructed end-to-end from A along $\vec{AB}$ will pass beyond B.

Ct2: Let $S_<$ and $S_\geqslant$ be two nonempty subsets of a line l satisfying:

- $S_< \cup S_\geqslant = l$
- no point of $S_<$ is between two points of $S_\geqslant$
- no point of $S_<$ is between two points of $S_<$

Then there is a unique point O on l such that for any two other points P_1 and P_2 with $P_1 \in S_<$ and $P_2 \in S_\geqslant$ then $P_1 * O * P_2$.

5.2 Lobachevskij geometry

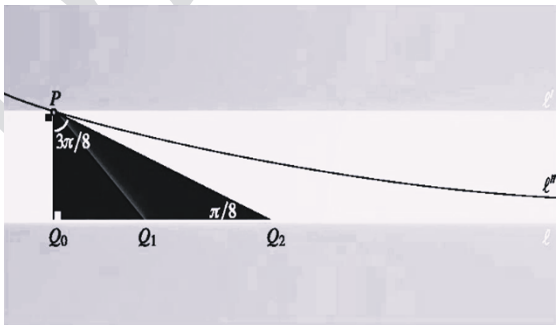
In order to explore non Euclidean geometry we will try to get a sense of the consequenses of *Playfair's axiom* being violated. The attempt to prove Euclid's parallel axiom shifted the focus on rectangles which are ubiquitous in Euclidean geometry. By definition a rectangle is a quadrilateral with four right angles and congruent opposite sides. In the following only the most essential theorems will be presented and for those interested in proofs, or more detailed derivations, the reader is referred to [Har15].

Theorem 5.1. Suppose that a rectangle R exists, with base b and height h. Then for positive real numbers x, y, there is a rectangle with base x and height y *[Har15, p. 400]*

Theorem 5.2. If rectangles of all sizes exist, then the angle sum of a triangle is π

Theorem 5.3. If the angle sum of a triangle is π, then for a line l and a point P not on l, there is a unique line through P parallel to l. That is, Playfair's axiom is true.

Proof: Assume P and l are given. We label the point defined by the intersection of the line perpendicular to l though P and l itself Q_0. Now we construct a line perpendicular to the line defined by P and Q_0. As fig. 8 illustrates, the perpendicular line is unique.



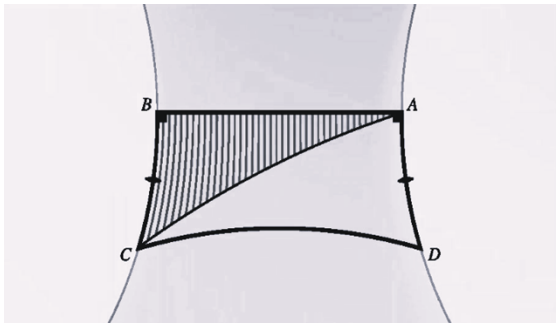
Source: [Har15, p. 402]

Figure 8: Series of lines through P

Definition 5.3 (Quadrilateral)**.** *A convex quadrilateral* $\square ABCD$ *is called a* Saccheri quadrilateral *with base* AB*, summit* CD*, and legs* BC *and* AD *if:*

- $\angle A$ *and* $\angle B$ *are right angles, and*
- *segments* BC *and* AD *are congruent.*

$\angle C$ *and* $\angle D$ *are referred to as the* summit angles *of the* Saccheri quadrilateral*, which can be constructed in a* neutral geometry*. [Har15, p. 403]*



Source: [Har15, p. 404]

Figure 9: The parts of a Saccheri quadrilateral

Theorem 5.4. Let $\square ABCD$ be a Saccheri quadrilateral with base AB. Then the summit angles $\angle C$ and $\angle D$ cannot be obtuse. *[Har15, p. 405]*

Proof The diagonal AC divides the Saccheri quadrilateral into two two triangles, and by the Saccheri-Legendre theorem the angle sum of each is at most π. Therefore the angle sum of $\square ABCD$ is at most 2π. So

$$\begin{aligned}(\angle A)+(\angle B)+(\angle C)+(\angle D) &\leqslant 2\pi\\ \pi+(\angle C)+(\angle D) &\leqslant 2\pi\\ (\angle C)+(\angle D) &\leqslant \pi\end{aligned}$$

As $\angle C$ and $\angle D$ are congruent, they must measure at most $\frac{\pi}{2}$

From here on we shall assume violation of Playfair´s axiom, resulting in the sum of inner angles of triangles being strictly less than $\frac{\pi}{2}$ and Saccheri quadrilaterals not being rectangles, as those do not exist.

Theorem 5.5. Let l be a line and P a point not on l. Then there are infinitely many lines through P that are parallel to l *[Har15, p. 412].*

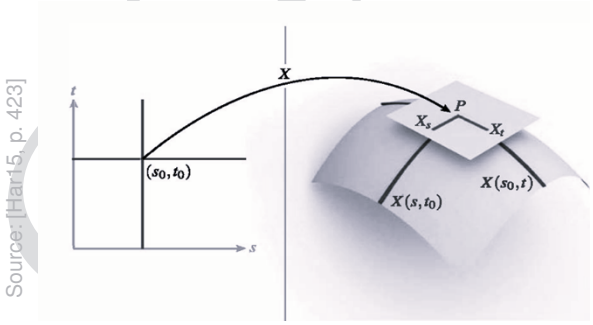
Another consequence is that for any two parallel lines l_1 and l_2, there are at most 2 points on l_1 that are any given distance away from l_2. Also a function f that measures the local distance between l_1 and l_2 is continous, has no local maxima but a unique local minimum

5.2.1 The Pseudosphere

Now lets define an object - more specificly a surface - that exhibits the characteristics of non Euclidean geometry. A sphere while curved, is curved in the wrong way as lines move towards each other rather that away from one another. There are however surfaces such as the *pseudosphere* that exhibit the properties of Non Euclidean geometry. We begin by describing a surface S in $\mathbb{R}^3$ with the following parametrized form

$$X(s,t) = [f(s,t), g(s,t), h(s,t)]^T$$

the partial derivatives of which span the tangent plane T_p, the best linear approximation of the surface S around p.



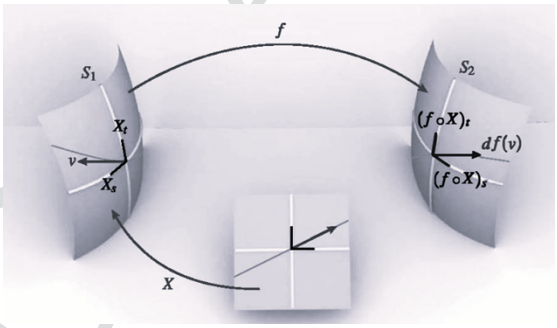
Source: [Har15, p. 423]

Figure 10: The tangent space at P

Now that we have an idea of surfaces and their tangent spaces, we can go back to defining the *pseudosphere* wich is simple the revolution of a smooth curve $\mathcal{C}$. Given $r(t) = [0, y(t), z(t)] \in \mathcal{C}^\infty$ with $\|\nabla r(t)\|_2 = 1$, the surface of revolution around the y-axis can then be parametrized resulting in

$$\begin{aligned} & X(s,t) = [z(t)\sin s, y(t), z(t)\cos s]^T \qquad 0 \leqslant s \leqslant 2\pi\\ \Rightarrow\quad & X_s = [z'(t)\cos s, 0, -z'(t)\sin s]^T\\ \Rightarrow\quad & X_t = [z'(t)\sin s, y'(t), z'(t)\cos s]^T\end{aligned}$$

5.2.2 The Gaussian Map



Source: [Har15, p. 425]

Figure 11: A map $f : S_1 \to S_2$ with $df : T_p \to T_{f(p)}$ mapping the tangent planes

Let $f : S_1 \mapsto S_2$ be a map from a surface S_1 to another S_2. With the tangent plane T_p, $p \in S_1$ being the best linear approxiamtion to S_1 around p, it can be concluded that $T_{f(p)}$ is the best linear approximation of S_2 around $f(p)$. The differential $df : T_p \to T_{f(p)}$ of f at p, illustrated in fig. 11, is given by

$$df(aX_s + bX_t) = a(f\circ X)_s + b(f\circ X)_t \tag{12}$$

Definition 5.4 (Differential map)**.** *let $\mathcal{S}$ denote the unit sphere. For a point p on a $S \in C^\infty$, let $n(p)$ be the unit normal vector to S at p. If the trail of $n(p)$ is translated to the origin, then its head will be at a point $\nu(p)$ on $\mathcal{S}$. The* Gauss map *$\nu : S \to \mathcal{S}$ is defined by $p \mapsto \nu(p)$.*

$$\begin{aligned} & \nu(P) = [y'(t)\sin s, -z'(t), y'(t)\cos s]^T\\ \Rightarrow\quad & (\nu\circ X)_s = \frac{y'(t)}{z(t)}X_s\\ \Rightarrow\quad & (\nu\circ X)_t = \frac{z''(t)}{y'(t)}X_t\\ \Rightarrow\quad & d\nu\begin{pmatrix}a\\ b\end{pmatrix} = \begin{bmatrix}\frac{y'}{z} & 0\\ 0 & -\frac{z''}{y'}\end{bmatrix}\begin{pmatrix}a\\ b\end{pmatrix}\end{aligned} \tag{13}$$

The *Gaussian curvature* $\kappa(p)$ of S at the point p is defined to be the determinant of $d\nu$ at p, which in our case evaluates to

$$\kappa(p) = -\frac{z''(t)}{z(t)} \tag{14}$$

For the case of constant negative curvature, -1 for example, the following differential equation

$$-\frac{z''(t)}{z(t)} \stackrel{!}{=} -1 \Rightarrow z''(t) = z(t), \tag{15}$$

of which e^t is a solution, arises. As the tangent vector is normalised, $y(t)$ can be obtained as follow

$$\begin{aligned} & (y'(t))^2 + (z'(t))^2 \stackrel{!}{=} 1 \\ \Rightarrow \quad & (y'(t))^2 + e^{2t} = 1 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & y'(t) = \sqrt{1-e^{2t}} \\ \Rightarrow \quad & y(t) = \int -\sqrt{1-e^{2t}}\,dt \\ & = \ln\left(\frac{1+\sqrt{1-z^2}}{z}\right) - \sqrt{1-z^2}, \qquad 0 \leqslant z \leqslant 1 \end{aligned}$$

Please refer to [Har15, pp. 425–428] for more details on the derivation of eqs. (13) and (14)

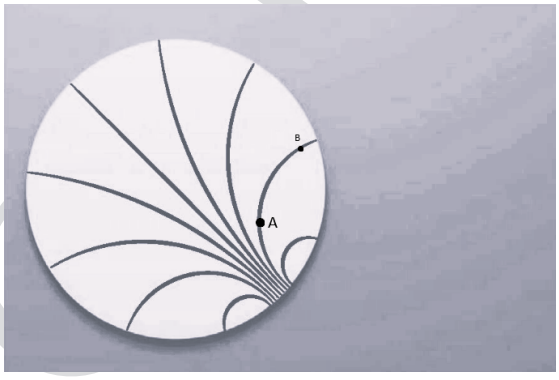
5.2.3 The Poincaré Model

The Poincaré model is one of 5 isometric models one can work with. It is defined by the manifold $\mathcal{M}_\kappa^n = \{x \in \mathbb{R}^n : \kappa\|x\| < 1\}, \quad \kappa \geqslant 0$ equiped with the following *Riemannian metric* and induced *distance metric*

$$g_x^{\mathcal{M}} = \lambda_x^2 g^{\mathbb{E}}, \qquad \lambda_x = \frac{2}{1-\kappa\|x\|^2} \tag{16}$$

$$d_{\mathcal{M}}(x,y) = \cosh^{-1}\left(1 + 2\frac{\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right) \tag{17}$$

Note that for $\kappa > 0$ the radius of the Poincaré ball is $\frac{1}{\sqrt{\kappa}}$. Also with the Poincaré ball being conformal to Euclidean space, angles are preserved with the definition being analogous to the Euclidean case as described in [GBH18, p. 3].



Source: [Har15, p. 458]

Figure 13: Some gyrolines in the Poincaré disk model. The line through A and B is a geodesic.

5.3 The κ-stereographic model

We use the gyrospace formalism to generalize vector spaces to the Poincaré model with all essential properties in Riemannian geometry representable by *Möbius vector addition* and *scalar-vector multiplication*. With this tools being limited to spaces of constant negative curvature, we extend them to all curvatures by expanding gyrovector spaces to the κ-stereographic model [BBG20]. But first a formal definition of a gyrovector space is needed.

5.3.1 The Gyrovector Space

The gyrovector space shall be established by defining the *Möbius addition*, *Möbius scalar multiplication*, *Distance metric* and *Geodecics* as follows

Definition 5.5 (Möbius Addidtion,-scalar multiplication, distance matric, geodesics)**.**

$$x \oplus_\kappa y := \frac{(1+2\kappa\langle x,y\rangle + \kappa\|y\|^2)x + (1+\kappa\|x\|^2)y}{1+2\kappa\langle x,y\rangle + (\kappa\|x\|\cdot\|y\|)^2} \tag{18}$$

$$r \otimes_\kappa x := \frac{1}{\sqrt{\kappa}}\tanh\left(r\cdot\tanh^{-1}(\sqrt{\kappa}\|x\|)\right)\frac{x}{\|x\|} \tag{19}$$

$$d_\kappa(x,y) := \frac{2}{\sqrt{\kappa}}\tanh^{-1}(\sqrt{\kappa}\|-x\oplus_\kappa y\|) \tag{20}$$

$$\gamma_{x\to y}(t) := x \oplus_\kappa (-x \oplus_\kappa y) \otimes_\kappa t, \tag{21}$$

$$\gamma_{x\to y} : \mathbb{R} \to \mathcal{M}_\kappa^n s.t., \tag{22}$$
$$\gamma_{x\to y}(0) = x$$
$$\gamma_{x\to y}(1) = y$$

We further more define the *gyrogroup coaddition* that shall play a role in **WHERE AND WHY DOES THIS PLAY A ROLE?**

Definition 5.6 (Coaddition)**.** *Let $(\mathcal{M},\oplus)$ e a gyrogroup. The gyrogroup coaddition $\boxplus$ is a binary operation in $\mathcal{M}$ related to the Möbius addition by the equation*

$$a \boxplus b = a \oplus \mathrm{gyr}[a,\ominus b]b$$

with $\mathrm{gyr}[a,\ominus b]$ given by the definition in [Ung07, p. 5].

A probably more digestable definition is given in [Ung09, pp. 48–50]. For all these operation we observe that $\lim_{\kappa\to 0}$ - so the curvature converging towards none at all - yields the corresponding representations in Euclidean space.

5.3.2 Exponential and logarithmic maps

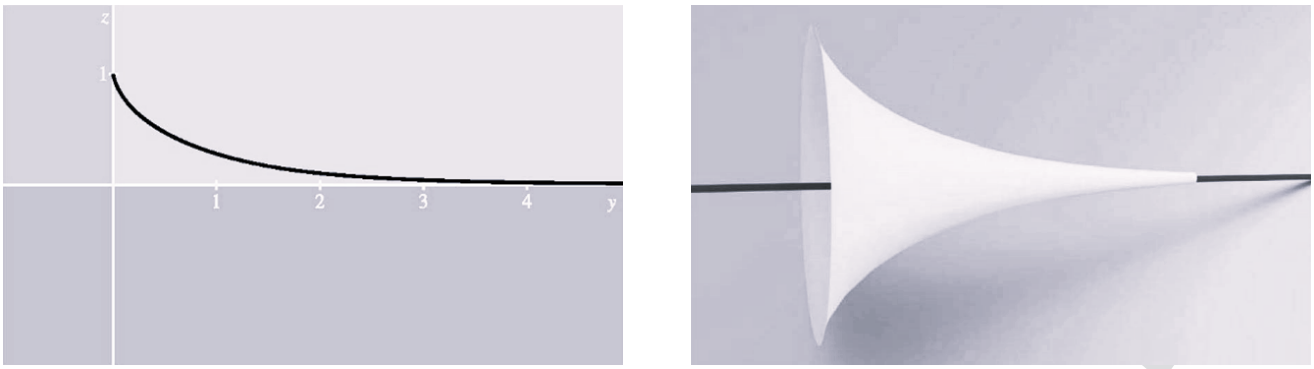
The exponential and logarithmic maps are used to map points from the tangent space $T_x\mathcal{M}$ onto the manifold and vice-versa.

Lemma 5.1. For any point $x \in \mathcal{M}_\kappa^n$, the exponential map $\exp_x^\kappa : T_x\mathcal{M}_\kappa^n \mapsto \mathcal{M}_\kappa^n$ and the logarithmic map $\log_x^\kappa : \mathcal{M}_\kappa^n \mapsto T_x\mathcal{M}_\kappa^n$ are given for $v \neq 0 \wedge y \neq x$ by:

$$\exp_x^\kappa(v) := x \oplus_\kappa \left(\tanh\left(\sqrt{\kappa}\frac{\lambda_x^\kappa\|v\|}{2}\right)\frac{v}{\sqrt{\kappa}\|v\|}\right) \tag{23}$$

$$\log_x^\kappa(y) := \frac{2}{\sqrt{\kappa}\lambda_x^\kappa}\tanh^{-1}(\sqrt{\kappa}\|-x\oplus_\kappa y\|)\frac{-x\oplus_\kappa y}{\|-x\oplus_\kappa y\|} \tag{24}$$

Here as well $\lim_{\kappa\to 0}\exp_x^\kappa(v) = x+v$ and $\lim_{\kappa\to 0}\log_x \kappa(y) = y - x$ correspond to the Euclidean case. With this tools -



(a) tractrix (b) pseudosphere

Figure 12: (a) The graph of the *tractrix* (for $y > 0$). (b) Revolution of the *tractrix* around the y-axis gives us the *pseudosphere*

of which the proves can be found in [GBH18, p. 5] - at hand, we can obtain some understanding of the *Möbius scalar multiplication* [GBH18, p. 5].

Lemma 5.2. $r \otimes_\kappa x$ can be obtained by projecting x onto the tangent space at origin $\mathbf{o}$ using the logarithmic map, multiplying this projection by the scalar r in $T_{\mathbf{o}}\mathcal{M}_\kappa^n$ and then projecting it back onto the manifold using the exponential map

$$r \otimes_\kappa x = \exp_{\mathbf{o}}^\kappa(r \log_{\mathbf{o}}^\kappa(x)), \qquad \forall r \in \mathbb{R}, x \in \mathcal{M}_\kappa^n \tag{25}$$

The parallel transport to gyrovector spaces is connected by the following theorem, which plays a crucial role in defining maps between layers in the Neural network [GBH18, p. 5]

Theorem 5.6 (Parallel transport)**.** *The parallel transport w.r.t. the Levi-Civita connection on the the manifold $(\mathcal{M}_\kappa^n, g^\kappa)$ of a vector $v \in T_{\mathbf{o}}\mathcal{M}_\kappa^n$ to another tangent space $T_x\mathcal{M}_\kappa^n$ is given by the following isometry:*

$$P_{\mathbf{o}\to x}^\kappa(v) = \log_x^\kappa(x \oplus_\kappa \exp_{\mathbf{o}}^\kappa(v)) = \frac{\lambda_{\mathbf{o}}^\kappa}{\lambda_x^\kappa} v \tag{26}$$

Given a curvature $\kappa \in \mathbb{R}$, a n-dimensional κ-stereographic model $(\mathfrak{st}_\kappa^d, g^\kappa)$ can be characterized by the manifold $\mathfrak{st}_\kappa^d$ and a Riemannian metric g^κ:

$$\begin{aligned} \mathfrak{st}_\kappa^d &= \{x \in \mathbb{R}^d \mid -\kappa\|x\|_2^2 < 1\} \\ g_x^k, \lambda_x^\kappa &:= \text{eq. (16)} \end{aligned} \tag{27}$$

with $g^{\mathbb{E}}$ being the Euclidean metric tensor. It can be seen that for $\kappa \geqslant 0$ the manifold is $\mathbb{R}^d$ whereas in the opposite case it represents the *Poincaré disk*.

Definition 5.7 (Distance)**.** *For any two distinct points $x, y \in \mathfrak{st}_\kappa^d$, the distance metric on the κ-stereographic model is defined as*

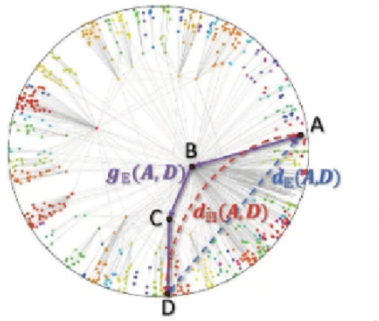
$$\begin{aligned} d_{\mathfrak{st}}^\kappa(x,y) &= \frac{2}{\sqrt{|\kappa|}} \tan_\kappa^{-1} \| -x \oplus_\kappa y\| \\ x \oplus_\kappa y &:= \frac{(1 - 2\kappa x^T y - \kappa\|y\|^2)x + (1 + \kappa\|x\|^2)y}{1 - 2\kappa x^T y + \kappa^2\|x\|^2\|y\|^2} \in \mathfrak{st}_\kappa^d \\ s \otimes_\kappa x &:= \tan_\kappa(s \cdot \tan_\kappa^{-1}\|x\|)\frac{x}{\|x\|} \in \mathfrak{st}_\kappa^d, \quad s \in \mathbb{R} \end{aligned} \tag{28}$$

The manifold $\mathfrak{st}_\kappa^d$ and the tangent space $T_x\mathfrak{st}$ can be mapped to each other via *exponential-* and *logarithmic map* which are given by

$$\begin{aligned} \exp_x^\kappa(v) &:= x \oplus_\kappa \left[\tan_\kappa\left(\sqrt{|\kappa|}\frac{\lambda_x^\kappa\|v\|}{2}\right)\frac{v}{\|v\|}\right] \\ \log_x^\kappa(y) &:= \frac{2|\kappa|^{-\frac{1}{2}}}{\lambda_x^\kappa} \tan_\kappa^{-1}\| -x \oplus_\kappa y\| \frac{-x \oplus_\kappa y}{\| -x \oplus_\kappa y\|}, \\ \tan_\kappa &:= \frac{1}{\sqrt{\kappa}}\tan(\cdot) \end{aligned} \tag{29}$$

as laid out in [Fu+23, p. 5].

5.3.3 Embedding distortion



Source: [Fu+23, p. 5]

Figure 14: Different metrics in the Poincaré model

The performance of a model is very much influenced by the feature embedding [Fu+23, p. 5], thus is can be said that the chosen embedding induces an upper limit to the model performance. This *loss* is quantified using the *embedding distortion* based on the two distance metrics induced by the Euclidean- and Minkovski inner products. §5.2.2 touched on how lines are mapped to curves. fig. 14 illustrates this behaviour where the blue line represents $d_{\mathbb{E}}(A, D)$ whereas the red line represents $d_{\mathbb{H}}(A, D)$ the hyperbolic embeding distance. The deviation of the embeding distance

$$g_{\mathfrak{st}}(A, D) = d_{\mathfrak{st}}(A, B) + d_{\mathfrak{st}}(B, C) + d_{\mathfrak{st}}(C, D) \tag{30}$$

from the graph distance defines the embedding distortion.

The shortest path between node pairs decides the minimum number of layers of GCNs needed to capture the relational information.

Definition 5.8 (Embedding distortion)**.** *Given a graph* G *with node set* V*, the embedding distortion* $\mathcal{D}$ *in the hyperbolic space* $\mathbb{H}$ *and spherical space* $\mathcal{S}$ *is defined as:*

$$\mathcal{D} = \frac{1}{|V|^2} \sum_{i,j \in V} \left| \left(\frac{d^2_{st}(i,j)}{g^2_{st}(i,j)} - 1 \right) \right| \tag{31}$$

5.4 Projective Geometry

We have talked about *Euclidean geometries*, *Neural geometries*, and *Non Euclidean Geometries*, but we could not possibly cover all Non Euclidean Geometries as that not would not only be a lifetimes work, but also of no great use to this paper. There is however one more Non Euclidean geometry we need to talk about. Although It is not necessarily in the modelling part of this paper, but for visualisation. it is a geometry that we are have all become very much accustomed to in the age of screens surrounding us in everyday life. It is ofcourse *Projective Geometry* we are talking about. Projective geometry is concerned with properties of incidence. Properties which are invariant under stretching, translation, or rotation of the plane. Thus in the axiomatic development of the theory, the notion of distance and angle will play no role. The following is mainly based on [Pri19] and [Har09].

5.4.1 Affine Geometry

We shall start with the following axioms for the synthetic development.

Definition 5.9 (Affine geometry)**.** *An* affive plane *is a set, whose elements are claeed points, and a set of subsets, called lines, satisfying the following three axioms. We will use denote Points by* P*, lines by* l*.*

A1. Euclids first postulate*. We say two lines are parallel if they are equal or have no points in common.*

A2. Playfair's *axiom*

A3. There exist three non-collinear points.

The ordinary Euclidean plane satisfies all axioms and is therefore an *affine geometry*, with a convenient way of representing the plane being the introduction of cartesian coordinates. Given four points P, Q, R, S, consider the lines PR and QS. It may happen that they meet. on the other hand, it is consistent wiht the axioms to assume they do not. in that case we have an affine plane consisting of four points and 6 lines $PQ, PR, PS, QR, QR, QS, RS$, whihc is by the way the smallest possible affine plane.

Definition 5.10 (pencil)**.** *A* pencil *of lines is either*

i the set of all lines passing through some point P*, or*

ii the set of all lines parallel to some line l*.*

In the later case, we speak of a pencil of parallel lines.

We can now complete the affine plane by addding points in infinity and thus arrive at the notion of the projective plane. Let A be an affine plane . For each line $l \in A$, we will denote by $[l]$ the pencil of parallel lines parallel to l, and we will call $[l]$ an *ideal point*, or *point at infinity*, in direction of l and we write $P^* = [l]$. We define the completion S of A as follows. The points of S are the points of A, plus all the ideal points of A. A line in S is either

i An ordinary line l of A, plus the ideal point P^* of l, or

ii the "line at infinity", consisting of all the ideal points of A.

The projetive plane is then defined as follows

Definition 5.11 (projective plane)**.** *A* projective plane S *is a set, whose elemets are called points, and a set of subsets, called lines, satisfying the following axioms as laid out in :*

P1. Two distinct points P, Q *of* S *lie on one and only one line.*

P2. VEBLEN-YOUNG-Axiom*: Any two lines have at least one point of incidence.*

P3. There exists three non-collinear points.

P4. Every line contains at least three points.

The proofs of these axioms are given in the section A.

5.4.2 Homogeneous Coordinates and Transformations

An analytic definition of the real projective plane can be given as follows. We consider lines in $\mathbb{R}^3$. A point of S is a line through the origin $\mathbf{O}$. We will represent the point P of S corresponding to l by choosing any point (x_1, x_2, x_3) on l different from $\mathbf{O}$. The numbers x_1, x_2, x_3 are homogeneous coordinates of P, with l given by $\lambda(x_1, x_2, x_3), \lambda \in \mathbb{R} \setminus \{0\}$.

Proposition 5.1. *The projective plane* S *defined by homogeneous coordinates which are real numbers, as above, is isomorphic to the projective plane obtained by completing the ordinary affine plane of Euclidean geometry. The proof [Har09] is left for the curious reader to explore.*

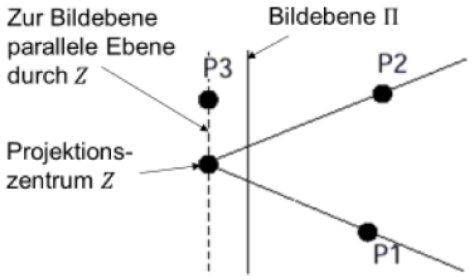
We now present some definitions of objects in the projective space, as well as transformations, in which we will cut ourselves loose from $\mathbb{R}$ and instead an abitrary field E as basis of the vector field. There will be no further elaboration on them, we however recommend [Pri19] for further reading.

Definition 5.12 (Homogeneous coordinates)**.** *Given a point* $x \in \mathbb{R}^n$ *with*

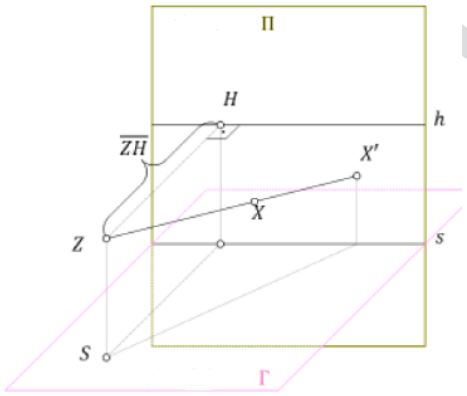
$$\tilde{x} = \begin{pmatrix} d \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \end{pmatrix} = \begin{pmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \end{pmatrix} \leftrightarrow \begin{pmatrix} \frac{\tilde{x}_1}{\tilde{x}_0} \\ \vdots \\ \frac{\tilde{x}_n}{\tilde{x}_0} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad p_0 \neq 0 \tag{32}$$

Definition 5.13 (Homogeneous hyperplane)**.** *A* homogeneous hyperplane $H \subset E^n$ *is defined by the equation*

$$\begin{aligned} H : \tilde{h}_0 \cdot 1 + \tilde{h}_1 \cdot x_1 + \cdots + \tilde{h}_n \cdot x_n = 0 \\ \rightarrow \tilde{h} = (\tilde{h}_0 \tilde{h}_1, \ldots, \tilde{h}_n)^T \in \mathbb{R}^{n+1} \end{aligned} \tag{33}$$



(a) birds-eye view



(b) 3D view

Source: [Pri19, p. 12]

Figure 15: 2 differnt illustrations of the central projections

It is homogeneous, as the equation can be scaled by any $d \in \mathbb{E}$ *without changing it solution set.*

Definition 5.14 (Affine transformation)**.** *For any transformation given by* $\alpha : \mathbb{E}^n \to \mathbb{E}^n, x \mapsto \alpha(x) := A \cdot x + a$*, the following hold* $A^T A = \mathbf{I}$ *and* $\det A = 1$*. Every tansformation with the given determinant is one of the following:*

- *rotation around* x_F
- *translation*
- *identity*

Translations: $x' = \alpha(x) = x + a$ *or* $\tilde{x}' = \alpha(x) = T(a)\tilde{x}$*, with*

$$T(a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_1 & 1 & 0 & 0 \\ a_2 & 0 & 1 & 0 \\ a_3 & 0 & 0 & 1 \end{bmatrix} \tag{34}$$

Rotation around arbitrary axis r*:*

$$R_{(3,3)}(\phi, r) = r \cdot r^T + \cos\phi \cdot (\mathbf{I} - r \cdot r^T) + \sin\phi \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{35}$$

Scaling by factor $\mathbf{s}$*:*

$$\alpha : \mathbb{E}^3 \to \mathbb{E}^3 : x \mapsto \alpha(x) := \mathbf{s} \cdot x \Rightarrow \begin{bmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{36}$$

One of, if not the greatest beneficiary of affine geometry is the video game sector who rely on it for computations in the virtual 3-dimensional world as well as its projection onto the main human interface - the screen. We shall now look at the so called *central projection*.

We can think of the objects to be projected as lying on a horizontal image plane Γ(Schnittebene) as shown in fig. 15. The intersection between image plane Π and Γ defines the *base line*, denoted s. The normal vector of Π through Z we call *main axis*. The main axis intersects Π in the *main point*. The distance $\bar{ZH}$ is called *eye distance*. Lines perpendicular to Π are called *depthlines*. Lines parallel to Π we call *contour lines*. The line h we call *horizon*.

The projection line through X and Z in homogeneous parametric form is given by

$$\tilde{y} = t_0\tilde{z} + t_1\tilde{x}, \qquad (t_0, t_1) \in \mathbb{R}^2 \setminus (0,0), \tag{37}$$

with the intersection given by $\tilde{u} = \begin{pmatrix} -d \\ n_1 \\ n_2 \\ n_3 \end{pmatrix} : \tilde{u}^T\tilde{y} = 0$

6 Game Theory

As we shall see in §10, the framework introduced, although being a supervised learning setup, utilizes ideas from unsupervised learning to determine the optimal curvature for the embedding space. This is done by adapting a game theoretic framework with two competing/cooperating models, which we shall refer to as *agents*. In order to understand this scheme we shall introduce the theoretical game theoretic learning framework and present the theoretic ideas behind game theory and deliver rigor definition of the optimal solution as well as conditions and proves of the existence thereof. We shall introduce the *Nash Q-Learning*[HW03] approach as an extention of the *Q-Learning* [WD92] algorithm to a non-cooperative multi-agent context.

Game Theory is a field of mathematics that looks at so called games that appear in various field of research, playing a role in politics, economy, gaming and machine learning among others. Researchers investigating learning in the context of multi-agent systems have been particularly attracted to

reinforcement learning techniques, especially because the agents require no prior knowledge of the environment. Q-learning has been especially well-studied, and poses a firm foundation in the theory of Markov decision processes. It is also somewhat easy to implement and has been applied to areas such as cellular telephone channel allocation [RP08], robotics [Wal00], as well as computer vision [Ban+99].

Although implementing the original Q-learning to each agent in a multi-agent domain is possible, this approach does not take into consideration two key properties of the environment. First, the environment consists of other agents adapting in the same manner, thus the environment is non stationary and theoretical guarantees, for example that of convergence, are no more given. Second, the non-stationarity of the environment is not generated by an abitrary stochastic process, but rather by the rational of other agents. Indeed it has been shown - for example in the *iterated Prisoner's Dilemma tournament*[NV19], [GKH24] - that agents who take into consideration the actions of their counterparts outperform those that do not.

In extending Q-learning to multi-agent environments, we adopt the framework of *general-sum stochastic games*. In a *stochastic game*, each agent's reward depends on the joint action of all agents and the current sate, and state transitions obey the Markov property, meaning the future state depends solely on the current state as well as the combined actions taken. It is advantageous to us, that the general-sum stochastic game allows for abitrary relation between agent's rewards. As the name suggests, They represent a generalisation of *zero-sum games*, where the rewards are always negatively correlated, and *coordination games*, where they are always positively correlated.

Let us take a step back an add bone to flesh. We shall take a closer look at the concept of *games*, *Nash equilibria* as well as Q-learning. After which we shall present the finalized *Nash* Q-*Learning* algorithm.

6.1 Games

With the term game, we associate fun activities, such as computer games, moonopoly or even sports activities where participants compete in different manners against each other, but the idea of a game can be applied to more serious topics such as geopolitical decision making or business strategies that often have far reaching consequences, influencing the lifes of millions over decades, thus approaching decision making in such scenarios with scientific rigor seems like the right way to go. However the application of game theory in this paper shall not be of such seriousness, although probabibly similarly influencial. In this section we shall go into the concept of a game and for the sake of continuity we shall use the depiction delivered in the movie dedicated to the very man whom we owe our gratitude for laying the foundation on which modern game theory is built on. His name is John Forbes Nash Jr.

Imagine a group of mathematicians at the prime of their youth, sitting in a bar, having a drink and discussing the beauty of mathematics, as another kind of beauty catches their attention. The very kind of beauty that brought Athena's wrath upon Medusa, the very kind that led the great Gatsby to his legacy as well as his demise, and the very kind that guards Victoria's secret. The young men pondered on how to approach her, as one of them quickly notices the best strategy for each of them would be not to approach her at all, but rather go after her friends, the reason being, if they all went after her, she was most likely to have an aversive reaction towards them. Desperation is after all not known to be an attribute of attraction. If they then went after her friends, those would most likely reject them, as no one likes to be a second choice. So it seems the best strategy would be for each of them to try their chances with her friends. This way the the groups average benefit is maximized. In the movie, the character *John Nash* puts it this way - *"The best thing for the group is if everyone in the group does what is best for the themselves as well as the group"*.

This is a rather simple example of a game. The mathematicians are the players. Each player has a choice between two strategies. For the sake of simplicity, we shall look at aproaching any of her friends as just one possible choice.

Definition 6.1 (Game in normal form)**.** *A game in normal form is a set consisting of* players $i \in \mathcal{I}$*, which we take to be finite, the* pure strategy space S_i *for each player* i *and the* payoff function u_i *that gives player i's von Neumann-Morgenstern utility as a function of the joint action* $s = (s_1, \dots s_I)$ *of all players.s*

	L	M	R
U	4,3	5,1	6,2
M	2,1	8,4	3,6
D	3,0	9,6	2,8

Source: [FT91, p. 17]

Figure 16: Simple two player game

In the following we shall use the game illustrated in fig. 16, where we have two players and each of them has three actions to choose from. The first entry in every box denotes player 1's reward, whereas the second denotes player 2's reward. We can think of the players as being in seperate rooms and havin before them a keyboard with three buttons each. Player 1 has the options *up,middle,down*, whereas player 2's options are *left,middle,right*. We also assume that each player is aware of the reward matrix and is aware that all other players are aware of it as well - basically the structure of the game is common knowledge and all players are alike in the sense that they are all rational and aware of this fact.

We now analyse the best strategy/strategies for both players. First we note that no matter player 1's choice, R gives player 2 strictly higher payoff than M. We say strategy M is *strictly dominated*. It would therefore be unwise for player player 2 to choose strategy M. As player 1 assumes a rational player 2, she will assume her not choosing M and thus U would be her choice, as this in average maximizes her reward. With player 2 simultaneously going through

player 1's thought process, he will would choose L, as this maximizes his reward. This process is called *iterated (strict) dominance*.

6.1.1 Stochastic Games

We shall now look at a different type of game, namely *stochastic games* [HW03]. In a stochastic game, agents choose actions simultaneously. The state space and action space are assumed to be *discrete*.

Definition 6.2 (Stochastic Games)**.** *An n-player stochastic game Γ is a tuple $\langle S, A^1, \dots, A^n, u^1, \dots, u^n, p\rangle$, where S is the state space, u^i is the action space of player $i \in \{1, \dots, n\}$, $u^i : S \times u^i \times \cdots \times u^n \to \mathbb{R}$ is the payoff function for player i, $p : S \times u^i \times \cdots u^n \to \Delta(S)$ is the transition probability map, where $\Delta(S)$ is the set of probability distribution over state space S.*

Given state s, agents independently choose actions $a^i, \dots, a^n$ and receive rewards $u^i(s, a^1, \dots a^n), i \in \{1, \dots, n\}$. The state then transits to the next state s' based on fixed transition probabilities, satisfying the constraint

$$\sum_{s' \in S} p(s'|s, a^1, \dots, a^n) = 1 \tag{38}$$

In a *discounted stochastic game*, the objective of each player is to maximize the discounted sum of rewards, with discount factor $\beta \in [0, 1)$. Let π^i be the strategy of player i. For a given initial state, player i tries to maximize

$$v^i(s, \pi^i, \dots, \pi^n) = \sum_{t=0}^{\infty} \beta^t \mathbb{E}[u_t^i | \pi^i, \dots, \pi^n, s_t = s]$$

The point at which that gives us the best balance between competition and cooperation is known as the *Nash Equilibrium* and shall be discussed in the following.

6.1.2 Nash Q-Values

With an understanding of what a game in the game theoretical sense is, we shall now define the optimal solution(s), also refered to as *Nash Equilibrium*. A Nash equilibrium[FT91, p. 35] is a profile of strategies such that each player's strategy is an optimal response to the other players' strategies.

Definition 6.3 (Nash Equilibrium)**.** *In a stochastic game Γ, a Nash equilibrium is a tuple of strategies $(\pi_*^1, \dots, \pi_*^n)$, such that*

$$v^i(s, \pi_*^1, \dots, \pi_*^n) \geqslant v^i(s, \pi_*^1, \dots, \pi_*^{i-1}, \pi^i, \pi_*^{i+1}, \dots) \quad \forall \pi^i \in \Pi^i, \tag{39}$$

where Π^i is the set of strategies available to player i. Such an equilibrium is said to be strict *if each player has a unique best response to his rivals' strategies. That is, s^* is a strict equilibrium iff it is a Nash equilibrium and the inequality is strict.*

The strategies can be behavioral or stationary. [Fin64] shows that an equilibrium always consists of stationary strategies.

Theorem 6.1. *Every n-player discounted stochastic game has at least one Nash equilibrium point in stationary strategies.*

The proof is rather dense and long, but for those interested, we refer to [Fin64]

6.2 Q-Learning

Q-learning is a form of model-free reinforcement learning that provides agents with the capability of learning to act optimally in Markovian domains by iteratively being exposed to the consequences of actions without the need of any prior knowledge of the game structure. It learns the best strategy by trying all action/state combinations repeatedly. In the following we shall formally define the algorithm and deliver some proofs on its capabilities.

Lets imagine an agent navigating some discrete and finite world, choosing one from a finite collection of actions at every time step. Such a world constitues a controlled Markov process with the agent being the controller. The agent is equiped with the capability to both register the state s_i of the world, as well as choose its action $a_i \in \mathcal{A}^i$. Based on the state-action combination, the agent receives a reward u_i, whose mean value $\bar{u}_{s_i}(a_i)$ depends only on the state and action. The state of the world changes probabilistically from s_i to s_{i+1} according to

$$\mathrm{Prob}[s_{i+1}|s_i, a_i] = P_{s_i \to s_{i+1}}[a_i] \tag{40}$$

The agent is faced with the task of having to learn the optimal strategy as one that maximizes the total discounted expected reward. By discounted reward, we mean that rewards received after i steps are worth less than immediate rewards. The discount factor $0 < \beta^i < 1$ determines how much following rewards are discounted. We can think of a strategy as a function that takes as input a state and outputs an action. This is in a way the object to be optimized. We want to learn the optimal strategy π^*. The state value s is given by

$$V^\pi \equiv \bar{u}_s(\pi(s)) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[\pi(s)] V^\pi(s_{i+1}) \tag{41}$$

As laid out in [WD92, p. 2], there is at least one optimal stationary strategy π^*, such that

$$V^*(s_i) \equiv V^{\pi^*}(s_i) = \max_a \left\{ \bar{u}_{s_i}(a) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[a] V^{\pi^*}(s_{i+1}) \right\} \tag{42}$$

is as well as an agent can do from state s_i. The Q-values (or action-values) of a strategy π, which is the discounted reward for executing action a at state s_i and following strategy π thereafter is given by

$$Q^\pi(s_i, a) = \bar{u}_{s_i}(a) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[\pi(s_i)] V^\pi(s_{i+1}) \tag{43}$$

The objective in Q-learning is to estimate the Q-values $Q^*(s_i, a)$ for an optimal strategy. It can be shown that $V^*(s_i) \equiv \max_a Q^*(s_i, a)$ and that if a^* is an action at which

the maximum is attained, then an optimal strategy can be formed as $\pi^*(s_i) \equiv a^*$. If an agent can learn this Q-values, it can easily determine the optimal strategy.

The agents expereince consists of a sequence of distinct stages (also called *episodes*). In the i^{th} episode, the agent

- observes its current state s_i
- selects and performs an action a_i
- observes the subsequent state s_{i+1}
- receives an immediate payoff u_i and
- adjusts its Q_{i-1} values using a learning factor β_i, according to:

$$Q_i(s,a) = \begin{cases} (1-\beta_i)Q_{i-1}(s,a) + \beta_i[u_i + \alpha V_{i-1}(s_{i-1})] \\ Q_{i-1}(s,a), \quad \text{if } s \neq s_i \vee \ a \neq a_i \end{cases}$$

where

$$V_{i-1} \equiv \max_b \{Q_{i-1}(s_{i+1}, b)\}$$

is the best the agent thinks it can do from state s_{i+1}. With sufficient samples, the *central limit theorem* is guaranteed to make up for any biases at the begining.

Theorem 6.2 (Convergence)**.** *Given bounded rewards* $u_i \leqslant U$, *learning rates* $0 \leqslant \beta_i < 1$ *and*

$$\sum_i^\infty \beta_{i(s_i,a)} = \infty, \sum_i^\infty [\beta_{i(s_i,a)}]^2 < \infty, \forall s, a,$$

then $Q_i(s_i,a) \to Q^*(s_i,a)$ *as* $n \to \infty, \forall s_i, a$. *in a sense, we can say that the series* β_i *lies between the harmonic and geometric series.*

The proof of the convergence is given in the section B.

6.3 Multiagent Q-learning

We shall now extend Q-learning to the multi-agent case by redefining the Q-values. Finally we shall present an algorithm capable of learning such an equilibrium and analyse its computational complexity. The following is mostly based on the work of [HW03].

To adapt Q−learning to the multi-agent context, we redefine the Q-function as $Q(s, a^1, \dots, a^n)$, rather than $Q(s,a)$ in the single-agent case.

Definition 6.4 (Multi-agent Q-function)**.**

$$Q^i_*(s, a^1, \dots, a^n) = u^i(s, a^1, \dots, a^n) + \beta \sum_{s' \in S} p(s'|s, a^1, \dots, a^n) v^i(s', \pi^1_*, \dots, \pi^n_*), \tag{44}$$

,

where $v^i(s', \pi^1_*, \dots, \pi^n_*)$ *is the agents total discounted reward over infinte periods starting from state* s'. *This can be interpreted as the sum of agent* i*'s current reward plus its future rewards when all agents follow a joint Nash equilibrium strategy.*

6.4 Nash Q-learning Algorithm

The algortihm presented in [HW03] differs from standard single-agent Q-learning not only, but especially one matter, namely the fact that the this algortihm updates with future payoffs, whereas the single-agent Q-learning updates based on agents's own maximum payoff. In order to learn equilibria, the agent needs not only observe its own payoffs, but that of the others as well. If direct measurement is not possible, a proxy that approximately represents the reward must be given.

Our learning agent i, learns about its Q-values by forming an arbitrary guess at time 0. one simple guess would be letting $Q^i_0(s, a^1, \dots, a^n) = 0 \ \forall s \in S, a^1 \in A^1, \dots, a^n \in A^n$. At each time t, agent i observes the current state, and takes an action. After that, it observes its own reward, actions taken by all other agents, others' rewards, and the new state s'. It then calculates a Nash equilibrium $\pi^1(s') \cdots \pi^n(s')$, and updates its Q-values arcording to

$$\begin{aligned} Q^i_{t+1}(s, a^1, \dots, a^n) = (1-\beta)Q^i_t(s, a^1, \dots, a^n) \\ + \beta_t \left[u^i_t + \beta \mathrm{Nash}Q^i_t(s')\right], \end{aligned} \tag{45}$$

where

$$\mathrm{Nash}Q^i_t(s') = \pi^1(s') \cdots \pi^n(s') \cdot Q^i_t(s') \tag{46}$$

Algorithm 1 The Nash Q-learning algorithm

Input: $t \leftarrow 0$, initial state s_0
Input: Learning agent indexed by i
Input: For all $s \in S$ and $a^j \in A^j$, $j = 1, \dots, n$, set $Q^j_t(s, a^1, \dots, a^n) = 0$
1: **while** learning **do**
2: Choose action a^i_t
3: Observe $r^1_t, \dots, r^n_t$, $a^1_t, \dots, a^n_t$, and $s_{t+1} = s'$
4: **for** $j = 1, \dots, n$ **do**
5: $Q^j_{t+1}(s, a^1, \dots, a^n) \leftarrow (1-\beta_t)Q^j_t(s, a^1, \dots, a^n) + \beta_t[r^j + \beta \mathrm{Nash}Q^i_t(s')]$
6: **end for**
7: $t \leftarrow t+1$
8: **end while**

We now analyse the complexity of algorithm 1. The learning agent needs to maintain n Q-functions, one for each agent in the system. These Q-functions are maintained internally by the learning agent, assuming that it can observe other agents' actions and rewards.

The learning agent updates $(Q^1, \dots Q^n)$, where each $Q^j, j = 1, \dots, n$, is made of $Q^j(s, a^1, \dots, a^n) \ \forall s, a^1, \dots, a^n$. Let $|S|$ be the number of states, and let $|A^i|$ be the size of agent i's action space A^i. Asuuming $|A^i| = |A| \ \forall i = 1, \dots, n$, the total number of entries in Q^k is $|S| \cdot |A|^n$. Since the agent has to maintain n Q-tables, the total space required is $n|S| \cdot |A|^n$. Therefore there is A linear dependency between space complexity and number of state, polynomial on number of actions and exponential in number of agents. The time complexity is dominated by the calculation of Nash equilibria used in the Q-fucntion update. The computational complexity of finding an equilibrium in matrix games is unknown, but commonly used algorithms for 2-player games

have exponential worst-case behavior. Proof of convergence is given in [HW03, pp. 9–14]

7 Neural Networks

Machine learninig is one of the fastest growing areas of computer science, with far-reaching applications. From support vector machines, wich handle infinitely dimensional embeddings with ease, to *decision trees* which are extremely simple for their expressive power all the way to *transformers* of which surely every student is certainly most grateful for - the amount of innnovation we have seen in this field might even make up for the lack of significant progress in physics for the last few decades. In this section we take a look at the baseline for GNNs, namely *neural networks* themselves. It is assumed that the reader is somewhat familiar with the concept, such as stochastic gradient decent, focus lay our focus on the expressive capabilities of neural networks.

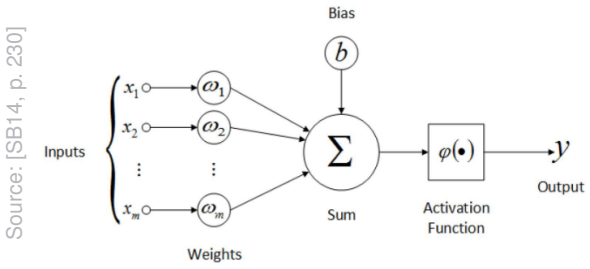


Figure 17: Artifical Neuron

An *artificial neural network* is a model of computation inspired by the structure of neural networks in the brain. It consists of a multitude of basic computing devices carrying out simple operations, that all together add up to something very powerful. The idea of the *artificial neuron* arose in the mid-20th century in [MP43] and has been shown to yield cutting-edge performance on several tasks. A neural network can be described in terms of a directed graph whose nodes correspond to neurons and edges to the link between them. Each neuron, such as the one illustrated in fig. 17, receives as input a weighted sum of the outputs of the neurons connected to its input channels. For the sake of comprehensibility we shall focus on *feedforward* networks that do not contain any cycles.

As expressed in [SB14], every predictor over n variables that can be implemented in time $T(n)$ can also be expressed as a neural network predictor of size $\mathcal{O}(T(n)^2)$, where the size of the network corresponds to the number of neurons in it. We will show that it both has polynomial sample complexity and has the minimal approaximation error among all hypothesis classes consisting of efficiently implementable predictors.

A Neural Network $\mathcal{H}_{V,E,\sigma,w}$, where V are the vertices, E the edges, σ the activation functions and w the weights defines a set of functions $\mathcal{H}_{V,E,\sigma,w}$:
$\mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$, which itself defines a hypothesis class for learning. We denote the hypothesis class by

$$\mathcal{H}_{V,E,\sigma} = \{\mathcal{H}_{V,E,\sigma,w} : w \text{ is a mapping from } E \text{ to } \mathbb{R}\} \tag{47}$$

We now look at what type of functions can be implemented using a neural network. We start by studying which type of Boolean functions $f_b : \{\pm1\}^n \to \{\pm\}$ can be implemented in $\mathcal{H}_{V,E,\sigma}$. Note that for every computer, storing real values using b bits, whenever we calculate a function $f : \mathbb{R}^n \to \mathbb{R}$ on such a computer we in fact calculate a function $g : \{\pm1\}^{nb} \to \{\pm1\}^b$. Therefore studying the sapace of possible boolean functions implicitly gives us the space of possible real functions. We begin with the simple claim that without restricing the size of the network, every Boolean function can be implemented using a neural network of depth 2.

Proposition 7.1. *For every n, there exists a graph (V, E) of depth 2, such that $\mathcal{H}_{V,E,\mathtt{sign}}$ contains all functions from $\{\pm1\}^n \to \{\pm1\}$ [SB14, p. 231].*

Proof. We construct a graph with $|V_0| = n+1$, $|V_1| = 2^n+1$, and $|V_2| = 1$. Let E be all possible edges between adjacent layers. Now, let $f : \{\pm1\}^n \to \{\pm1\}$ be some Boolean function. We need to show that we can adjust the weights so that the network implements f. Let $\mathbf{u}_1, \dots, \mathbf{u}_k$ be all vectors in $\{\pm1\}^n$ on which f evaluates to 1. Observe that for every i and every $\mathbf{x} \in \{\pm1\}^n$, if $\mathbf{x} \neq \mathbf{u}_i$ then $\langle \mathbf{x}, \mathbf{u}_i \rangle \leqslant n-2$ and if $\mathbf{x} = \mathbf{u}_i$ then $\langle \mathbf{x}, \mathbf{u}_i \rangle = n$. it follows that the function $g_i(\mathbf{x}) = \mathtt{sign}(\langle \mathbf{x}, \mathbf{u}_i \rangle - n + 1)$ equals 1 iff $\mathbf{x} = \mathbf{u}_i$. It follows that we can adapt the weights between V_0 and V_1, so that for every $i \in [k]$, the neuron $v_{1,i}$ implements the function $g_i(\mathbf{x})$. Next, we observe that $f(\mathbf{x})$ is the disjunction of the functions $g_i(\mathbf{x})$, and therefore can be written as

$$f(\mathbf{x}) = \mathtt{sign}\left(\sum_{i=1}^{k} g_i(\mathbf{x}) + k - 1\right), \tag{48}$$

which concludes the proof. □

Although this shows that neural network can implement any Boolean function, it however makes no claim on any bound of the necessary size. This means that as for now it could be that the size grows exponentially with the problem. in the construction given by the proof section 7, the number of nodes in the hidden layer is exponentially large.

Theorem 7.1. *For every n, let $s(n)$ be the minimal integer such that there exists a graph (V, E) with $|V| = s(n)$ such that the hypothesis class $\mathcal{H}_{V,E,\mathtt{sign}}$ contains all the functions from $\{0,1\}^n$to$\{0,1\}$. Then, $s(n)$ is exponential in n. Similar results hold for $\mathcal{H}_{V,E,\sigma}$ where σ is the sigmoid function.*

Proof. Suppose that for some (V, E) we have that $\mathcal{H}_{V,E,\mathtt{sign}}$ contains all functions from $\{0,1\}^n$ to $\{0,1\}$. It follows that it can shatter the set of $m = 2^n$ vectors in $\{0,1\}^n$ and hence the VC dimension [Duc17] of $\mathcal{H}_{V,E,\mathtt{sign}}$ is 2^n is 2^n. On the other hand, the VC dimension of $\mathcal{H}_{V,E,\mathtt{sign}}$ is bounded by $\mathcal{O}(|E|\log(|E|)) \leqslant \mathcal{O}(|V|^3)$. As shown in [SB14], this implies that $|V| \geqslant \Omega(w^{\frac{n}{3}})$, which concludes the proof. □

We have shown that, in theory, it is possible to express all Boolean functions using a network of polynommial size. In the following, we show that all Boolean functions that can be calculated in time $\mathcal{O}(T(n))$ can also be expressed by a network of size $\mathcal{O}(T(n)^2)$.

Theorem 7.2. *Let $T : \mathbb{N} \to \mathbb{N}$ and for every n, let $\mathcal{F}_n$ be a set of functions that can be implemented using a Turing machine using runtime of at most $T(n)$. Then, there exist constants $b, c \in \mathbb{R}_+$ such that for every n, there is a graph (V_n, E_n) of size at most $cT(n)^2 + b$ such that $\mathcal{H}_{V-n,E_n,\mathtt{sign}}$ contains $\mathcal{F}_n$*

Proof. The proof relies on the relation between the time complexity of programs and their circuit complexity [Sip10]. To summarize the idea, a Boolean circuit is a type of network in which the individual neurons implement conjunctions, disjunctions and negation of their inputs. Circuit complexity measures the size of Boolean circuits required to calculate functions. The relation between time complexity and circuit complexity can be seen intuitively as follows.

We can model each step of the execution of a computer program as a simple operation on its memory state. Therefore, the neurons at each layer of the network will reflect the memory state of the computerat the corresponding time, and the translation of the next layer of the network involves a simple calculation that can be carried out by the network. To relate Boolean circuits to networks with a sign activation function, we need to show that we can implement the operations of conjunction, disjunction, and negation, using the sign activation function. Clearly, we can implement the negation operator using the sign activation function. It is shown in [SB14] that the sign activation function can also implement conjunctions and disjunctions of its inputs. □

An upper bound of the model complexity is given by the following theorem, the proof of which is laid out in [SB14, pp. 234–235], utilizing the VC dimension.

Theorem 7.3 (VC dimension)**.** *The VC dimension of $\mathcal{H}_{V,E,\mathtt{sign}}$ is $\mathcal{O}(|E|\log(|E|))$.*

We have shown that neural networks can in theory approximate any real valued function with polynomial complexity. However the above mentioned don't say anything of the design that would emit the desired behaviour. And although there are smart choices to be made when designing an architecture a lot still remains try and error.

8 Hyperbolic Multiclass Logistic Regression

We shall generalize *multinominal regression* [HLS13] to the Pointcaré ball. First lets formulate Euclidean MLR in terms of distances to margin hyperplanes as described in [GBH18, chapter 3.1], which will pave the way for the generalisation.

Given K classes, one learns a margin hyperplane for each class using softmax probabilities:

$$\begin{aligned} &\forall\, k \in \{1,\dots,K\}, \quad p(k|x) \propto \exp(\langle a_k, x\rangle - b_k) \\ &b_k \in \mathbb{R}, x, a_k \in \mathbb{R}^n \end{aligned} \tag{49}$$

This can be interpreted as learning a hyperplane $H_k = \{x \in \mathbb{R}^n : \langle a, x\rangle - b = 0\}$ that best discribes class k. Another - maybe geometrically more accessible definition is given by $\tilde{H}_{a,p} = p + \{a\}^\perp$. The proximity to such a plane then represents the probability of a datapoint belonging to said class. Using the identity $\langle a, x\rangle - b = \mathrm{sign}(\langle a, x\rangle - b)\|a\|d(x, H_{a,b})$, eq. (49) can be reformulated as

$$\begin{aligned} p(k|x) &\propto \exp(\mathrm{sign}(\langle a_k, x\rangle - b)\|a_k\|d(x, H_{a_k,b_k})) \\ &= \exp(\mathrm{sign}(\langle -p_k + x, a_k\rangle)\|a_k\|d(x, \tilde{H}_{a_k,p_k})) \end{aligned} \tag{50}$$

This can now be adapted to the hyperbolic setting by replacing $+$ by $\oplus_\kappa$ and maybe by now an image of how this is a usefull approach for capturing hierarchical structures beging to clarify - in particular if we combine our understanding of eq. (49) with **??**

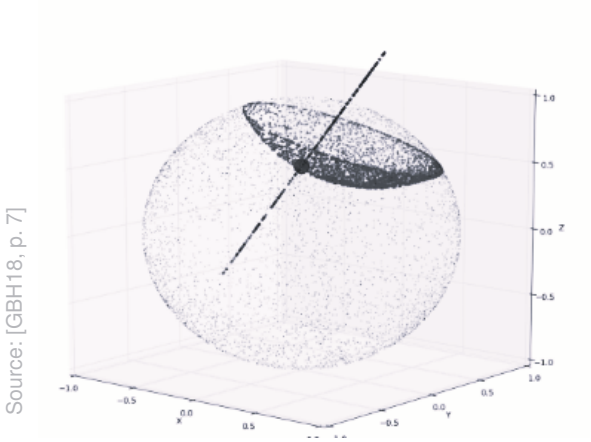
Definition 8.1 (Poincaré hyperplanes)**.** *For $p \in \mathcal{M}^n_\kappa, a \in T_p\mathcal{M}^n_\kappa \setminus \{0\}$, let $\{a\}^\perp := \{z \in T_p\mathcal{M}^n_\kappa : g^\kappa_p(z, a)\} = 0 = \{z \in T_p\mathcal{M}^n_\kappa : \langle z, a\rangle = 0\}$. Then, we define* Poincaré hyperplanes*[GBH18, p. 6] as*

$$\begin{aligned} \tilde{H}^\kappa_{a,p} &:= \{x \in \mathcal{M}^n_\kappa : \langle \log^\kappa_p(x), a\rangle_p = 0\} \\ &= \exp^\kappa_p(\{a\}^\perp) \\ &= \{x \in \mathcal{M}^n_\kappa : \langle -p \oplus_\kappa x, a\rangle = 0\} \end{aligned} \tag{51}$$

fig. 18 shows a perpendicular hyperplane and its normal vector at some point in the ball. The distaance from any point x to the plane is given by by the time

Theorem 8.1 (Hyperbolic plane distance)**.**

$$\begin{aligned} d_\kappa(x, \tilde{H}^\kappa_{a,p}) &:= \inf_{\omega \in \tilde{H}^\kappa_{a,p}} d_\kappa(x, \omega) \\ &= \frac{1}{\sqrt{\kappa}} \sinh^{-1}\left(\frac{2\sqrt{\kappa}|\langle -p \oplus_\kappa x, a\rangle|}{(1 - \kappa\| - p \oplus_\kappa x\|)\|a\|}\right) \end{aligned}$$



Source: [GBH18, p. 7]

Figure 18: Illustration of a hyperplane in the *Poincaré model*

Putting together eq. (50) and theorem 8.1 we get the hyperbolic MLR formulation

$$p(k|x) \propto \exp\left(\frac{\lambda^{\kappa}_{p_k}\|a_k\|}{\sqrt{\kappa}}\sinh^{-1}\left(\frac{2\sqrt{\kappa}\langle -p_k \oplus_\kappa x, a_k\rangle}{(1-\kappa\|-p_k\oplus_\kappa x\|^2)\|a_k\|}\right)\right), \quad \forall x \in \mathcal{M}^n_\kappa \tag{52}$$

which for $\lim_{\kappa\to 0}$ yields $\exp(\langle -p_k + x, a_k\rangle_o)$ the Euclidean softmax. The bais b is introduced by applying theorem 5.6 on $v = \log^\kappa_o(b)$. Similar to the exponential kernel for support vector machines, and as laid out in [GBH18, p. 7], the computation of a multi layer function $f_k(x) = \varphi_k(M_k x)$ with corresponding Möbius representation $f^{\otimes_\kappa}_k(x) = \varphi^{\otimes_\kappa}_k(M_k \otimes_\kappa x)$ can be computed in Euclidean space by re-writting their composition

$$f^{\otimes_\kappa}_k \circ \cdots \circ f^{\otimes_\kappa}_1 = \exp^\kappa_o \circ f_k \circ \cdots \circ f_1 \circ \log^\kappa_o \tag{53}$$

9 POINT-GNN

It is almost impossible to fail in imagining use cases in which an understanding of the 3 dimensional surrounding is vital. From autonomous driving to object detection in industrial automation to more foreign field that inherently deal with graph-like structures, such as molecular biology. The data we will be working with [GLU12], which we shall go into in more detail in **SECTION EXPLAINING DATASET** is uses in the former context. The architecture we present aims to create a graph representation for the point cloud data that combines the work of [Fu+23; SRR20]. These representations differ from common ones in the sence that the data is embedded in hyperbolic space, as elaborated in §5 and §8. We expect such an architecture to yield at least equal performance to that presented in [SRR20], as the optimal curvature is a learned feature of the predictor. The dataset however might not be ideal to demonstrate the benefits of the proposed architecture, as there are no strong dependencies (*tree-like structures*) in the embedding space to be expected, however a better class seperation and thus improved F1 score can be anticipated.

Existing architectures for object detection in images, such as convolutional neural networks often rely on operations that make constraints on the input data, such as the necessity for a regular grid of datapoints. Adapting voxels to PC data, which are to be seen as analogous to grids on image data would consequently lead to an uneven number of data points in each grid, resulting in information loss in populated voxels and unnecessary computation on empty ones. Such a model might have been more useful in the early stages of the universe where entropy was relatively low but not so much in the days of man. Other approaches [Cha+17], although allowing for heterogenous point distributions by not adapting the grid approach in the first place, do suffer from the computational cost of repeated sampling.

[SRR20] proposes a new architecture termed *Point-Graph Neural Networks* that resolves some of the prior mentioned problems by adapting the structure of the PC directly without regularizing. This approach was also motivated by the fact that although breakthroughs in classification and segmentation problems had been made, object detection still remained relatively unstudied prior to said publication.

9.1 Graph construction

The Point-GNN takes the point cloud as its input and outputs the category and bounding boxes of the objects to which each vertex belongs. Point-GNN is a one stage detection method that detects multiple objects in a single shot. As creating a graph on the original cloud P would be computationaly infeasible, we instead use a voxel downsampled cloud $\hat{P}$ for the graph construction - a step laid out in more detail in §10.4. It might seem like this process leads to some information loss, but the information of all datapoints within a voxel are used for the definition of the corresponding node. Similar to the `FastFourierTranform`, this step might loose significance due to Moore's law or the evaluation of larger point clouds would be within reach.

The PC of N points is defined as a set $P = \{p_1, \ldots, p_N\}$, where $p_i = (x_i, s_i)$ is a point with both 3D coordinates $x_i \in \mathbb{R}^3$ and the state value $s_i \in \mathbb{R}^k$ representing the point property. We construct a graph (P, E) by using P as the vertices and connecting a point to its neighbours within a fixed radius r as defined in eq. (54), i.e. Information on the original dense PC points within the corresponding voxel are incorporated within the state value s_i, which shall also embed the lidar reflection intensity as well as the relative coordinates. All this values are aggregated using MAX pooling.

$$E = \{(p_i, p_j) \mid \|x_i - x_j\|_2 < r\} \tag{54}$$

The construction of such a graph is the well known fixed radius near-neighbours search problem, with a time complexity of $\mathcal{O}(cN)$ where c is the max number of neighbours within the radius.

9.2 GNN with Auto-Registration

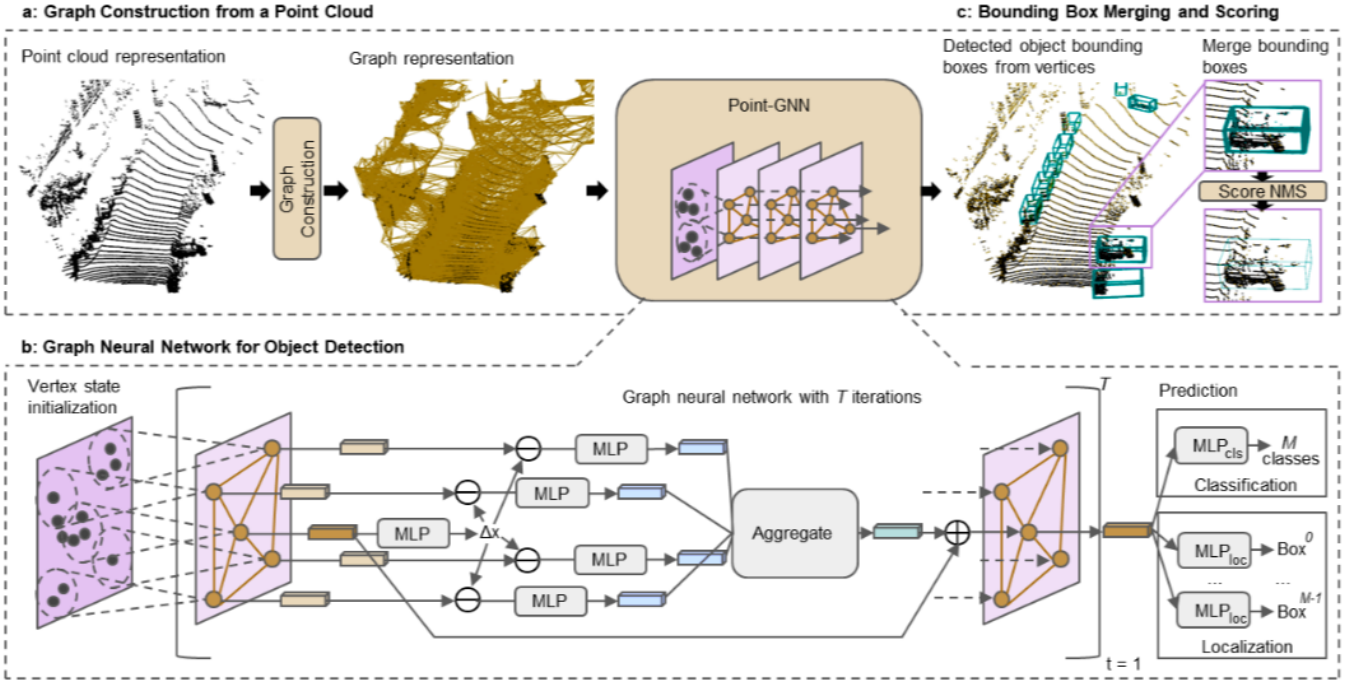
As laid out in §3.2.1, typical graph neural networks refine the vertex features by aggregating features along the edges. In the $(t+1)$th iteration, it updates each vertex feature in the form:

$$\begin{aligned} v^{t+1}_i &= g^t(\rho(e^t_{ij}|(i,j) \in E), v^t_i) \\ e^t_{ij} &= f^t(v^t_i, v^t_j) \end{aligned} \tag{55}$$

where e^t and v_t are the edge and vertex features from the t^{th} iteration. A function $f^t(.)$ is a set function which aggregates the edge features for each vertex. $g^t(.)$ takes the aggregated edge feature to update the vertex features. The vertex features are returned after the max number of iterations. A need to encorporate information about the object where the vertex belongs naturally arises, the state update rule takes a similar form as eq. (55)

$$s^{t+1}_i = g^t(\rho(\{f^t(x_j - x_i, s^t_j)|(i,j) \in E\}), s^t_i), \tag{56}$$

where the use of relative coordinates induces a translational invariance on a global scale, however it is still sensitive to translation within the neighborhood area thus increasing variance to $f^t(.)$. To reduce this behavior, we split neighbour's coordinates by their structural features instead of the center vertex coordinates. Because the center vertex contains some structural infromation from the previous iteration, it



Source: [SRR20, p. 3]

Figure 19: Point-GNN Architecture. (a) graph construction, (b) gnn for object detection, (c) bounding box merging and scoring

can be used to predict the offset by adapting the following *auto-registration* mechanism:

$$\begin{aligned} \delta x_i^t &= \mathbf{h}^t(s_i^t) \\ s_i^{t+1} &= g^t(\rho(\{f^t(x_j - x_i - \Delta x_i^t, s_j^t)\}), s_i^t) \end{aligned} \tag{57}$$

were Δx_i^t is the coordination offset for the vertices which is computed using the center vertexs' previous state value. We model $f^t(.), g^t(.), \mathbf{h}^t(.)$ using multi-layer perceptrons(MLP) and add a residual connection in $g^t(.)$ as illustrated in fig. 19. $\rho(.)$ is chosen to be MAX for its robustness as outlined in [Cha+17, p. 4]. A single iteration in the proposed graph network is given by:

$$\begin{aligned} \Delta x_i^t &= \text{MLP}_h^t(s_i^t) \\ e_{ij} &= \text{MLP}_f^t([x_j - x_i + \Delta x_i^t, s_i^t]) \\ s_i^{t+1} &= \text{MLP}_g^t(MAX(\{e_{ij} | (i,j) \in E\})) + s_i^t \end{aligned} \tag{58}$$

where $[,]$ represents the concatenation operation. The MLP^t are not shared across iterations t. The output state and vertice representations are archieved after T iterations and are then used to predict both the category and the bounding box of the object where the vertex belongs. A classification branch MLP_{cls} computes the multi-class probability. Finally a localization branch MLP_{loc} computes the bounding box for each class.

9.2.1 Loss

For the object category, the classification branch computes the multi-class probability distribution $\{p_{c_1}, \ldots, p_{c_M}\}$ for each vertex with M being the number of object classes including the *Background class*. If a vertex is not within any class bounding box, it is assigned the background class. We use the average cross-entropy loss as the classification loss.

$$l_{cls} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{c_j}^i \log(p_{c_j}^i) \tag{59}$$

where p_c^i and y_c^i are the predicted probability and the one-hot encoded class label for the i-th vertex respectively. The object bounding box b is predicted in a 7 degrees of freedom format $b = (x, y, z, l, h, w, \theta)$, where (x, y, z) represent the center of the bounding box,(l, h, w) the dimensions and θ the yaw angle. The bounding box is encoded with the vertex coordinates as follows:

$$\begin{aligned} &\delta_x = \frac{x - x_v}{l_m}, \quad \delta_y = \frac{y - y_v}{h_m}, \quad \delta_z = \frac{z - z_v}{w_m} \\ &\delta_l = \log(\frac{l}{l_m}), \quad \delta_h = \log(\frac{h}{h_m}), \quad \delta_w = \log(\frac{w}{w_m}) \\ &\delta_\theta = \frac{\theta - \theta_0}{\theta_m} \end{aligned} \tag{60}$$

where $l_m, h_m, w_m, \theta_0, \theta_m$ are constant scale factors. The localisation branch predicts the encoded bounding box $b_\delta = (\delta_x, \delta_y, \delta_z, \delta_l, \delta_h, \delta_w, \delta_\theta)$ for each class. If a vertex is within a bounding box, we compute the *Huber Loss* [Hub92] between the ground truth and the prediction. If a vertex is outside any bounding boxes or it belongs to a class that needs not be localized, we set its localization loss to zero. We then average the localization loss of all the vertices:

$$l_{loc} = \frac{1}{N}\sum_{i=1}^{N} 1(v_i \in b_{interest}) \sum_{\delta \in \delta_{b_i}} l_{Huber}(\delta - \delta^{gt}) \quad (61)$$

In order to mitigate overfitting, we add L1 regularization to each MLP. The total loss is then:

$$l_{total} = \alpha l_{cls} + \beta l_{loc} + \gamma l_{reg} \quad (62)$$

where α, β and γ are constant weights to balance each loss.

9.2.2 Box Merging and Scoring

As multiple vertices can be on the same object, the neural network can output multiple bounding boxes of the same object. It is necessary to merge these into one and also assign a confidence score. *Non-Maximum Suppression*[Pra21] has been widely used for such purposes. However the standard NMS implementation does not always choose the best bounding box, as classification confidence does not necessarily reflect localisatin accuracy. In [SRR20] this is dealt with by introducing an adaptation of the NMS algorithm. For the sake of simplicity and due to time limitations that is the implementation used in this paper, however it is worth mentioning that other generalisation approaches such as [HBS17] are to be expected to yield better results as one can expect a dependency between the classification problem and the dynamic of the resulting bounding boxes.

Algorithm 2 NMS with Box Merging and Scoring

Input: $\mathcal{B} = \{b_1, \dots, b_n\}$, $\mathcal{D} = \{d_1, \dots, d_n\}$, T_h
$\mathcal{M} \leftarrow \{\}$, $\mathcal{Z} \leftarrow \{\}$
while $\mathcal{B} \neq \emptyset$ **do**
  $i \leftarrow \arg\max \mathcal{D}$
  $\mathcal{L} \leftarrow \{\}$
  for b_j in $\mathcal{B}$ **do**
    if $iou(b_i, b_j) > T_h$ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup b_j$
      $\mathcal{B} \leftarrow \mathcal{B} - b_j$, $\mathcal{D} \leftarrow \mathcal{D} - d_j$
    end if
  end for
  $m \leftarrow median(\mathcal{L})$
  $o \leftarrow occlusion(m)$
  $z \leftarrow (o+1)\sum_{b_k \in \mathcal{L}} IoU(m, b_k) d_k$
  $\mathcal{M} \leftarrow \mathcal{M} \cup m$, $\mathcal{Z} \leftarrow \mathcal{Z} \cup z$
end while
return $\mathcal{M}, \mathcal{Z}$

To improve the localization accuracy, we propose to calculate the merged box by considering the entire overlapped box cluster. More specifically, we consider the median position and size of the overlapped bounding boxes. We also compute the confidence score as the sum of the classification scores weighted by the intersection of the Union (IoU) factor and occlusion factor. The occlusion factor represent the occupied volume ratio. Given a box b_i, let l_i, w_i, h_i be its dimensions and let v_i^l, v_i^w, v_i^h be the unit vectors that indicate their directions. x_j are the coordiantes of point p_j. The occulusion factor o_i is given by:

$$o_i = \frac{1}{l_i w_i h_i} \prod_{v \in \{v_i^l, v_i^w, v_i^h\}} \max_{p_j \in b_i}(v^T x_j) - \min_{p_j \in b_i}(v^T x_j) \quad (63)$$

We modify standard NMS as shown in algorithm 2. it returns the merged bounding boxes $\mathcal{M}$ and their confidence score $\mathcal{Z}$.

10 Adaptive Curvature Exploration GNN

In [Fu+23, pp. 8–11], an ACE-GEO framework is proposed in order to tackle the need limitations of Euclidean embeddings. We attempt not only to learn the low dimensional representation of the data living on some high dimensional manifold, but also consider the space the manifold lives in to be a learnable feature. In this section we shall digest the key ideas behind the theoretical framework, its implementation and how it can be adapted to §9. The framework consists of and *Adaptive Curvature Exploration Agent* (ACE-Agent) and a *Geometric Graph Neural Network Agent* (GEO-Agent). The former explores curvature, capturing topological heterogeneity while the later learns the node representations by the κ-stereographic model. The two models will be introduced as agents in a 2 player game as described in §6, as we shall see later

10.1 ACE-Agent

The need to model curvature as a learnable parameter in hidden layers arises due to the variation in local structures (curvatures). This is adressed by the *ACE-Agent* by disregarding curvature as a hyperparameter and instead detrmining the optimal curvature using the deviation from the Parallelogram Law [Wei].

The ideas presented in [Fu+23] related to the subject, on which most of the following will be based upon, are mostly based on the work laid out in [HW03], which is highly recommended, not only for complete appreciation of the topic but also to make sense of undefined terminology.

Definition 10.1 (ACE-Agent State S^t_{ACE})**.** *The state of* ACE-Agent *of epoch* t *is given by*

$$S^t_{ACE} = (\kappa_1^{t-1}, \dots, \kappa_L^{t-1}) \quad (64)$$

where $(\kappa_1^{t-1}, \dots, \kappa_L^{t-1})$ *are the explored curvatures of the last epoch, and* L *is the number of layers in the* Geo-Agent*.*

Definition 10.2 (ACE-Agent Action A^t_{ACE})**.** *The diviation from the Parallelogram Law is leveraged in order to explore the graph curvature under the condition of minimizing the embedding distortion definition 5.8. Let* (a, b, c) *be a Geodesic triangle in geometric space* $\mathfrak{st}^d_\kappa$*, and* m *be the (Geodesic) midpoint of* (b, c)*. The curvature is quantified by:*

$$\begin{aligned} \xi_{\mathfrak{st}}(a, b; c) &= g_{\mathfrak{st}}(a, m)^2 + \frac{g_{\mathfrak{st}}(b, c)^2}{4} \\ &\quad + \frac{g_{\mathfrak{st}}(a, b)^2 + g_{\mathfrak{st}}(a, c)^2}{2} \\ \xi_{\mathfrak{st}}(m; a, b; c) &= \frac{1}{2 g_{\mathfrak{st}}(a, m)} \xi_{\mathfrak{st}}(a, b; c) \end{aligned} \quad (65)$$

The new average curvature estimation κ^t in the t-th epoch is then defined as:

$$\kappa^t = \frac{1}{|V|}\sum_{m\in V}\left(\frac{1}{n_s}\sum_{j=0}^{n_s}\xi_{\mathfrak{st}}(\mathbf{h}_m^{t-1};\mathbf{h}_{a_j}^{t-1},\mathbf{h}_{b_j}^{t-1};\mathbf{h}_{c_j}^{t-1})\right) \tag{66}$$

where b and c are the neighbours of m, and a is any node except for $\{m, b, c\}$ in the graph G. The above sampling is done n_s times, after which the average is chosen to be the estimated curvature. Then the embeddings $\mathbf{h}^{\kappa^{t-1},t-1}$ of epoch $t-1$ from Geo-Agent as the input into the Riemmannian manifolds with the new curvature κ^t:

$$\begin{aligned}\kappa^t &= (1-\gamma)\kappa^{t-1} + \frac{\gamma}{\sqrt{-\kappa^t}}\\ \mathbf{h}^{\kappa^t,t} &\leftarrow \exp_{\mathbf{o}}^{\kappa^t}\left(\log_{\mathbf{o}}^{\kappa^{t-1}}(\mathbf{h}^{\kappa^{t-1},t-1})\right)\end{aligned} \tag{67}$$

where γ is a weight parameter of estimated curvature and new curvature and $\mathbf{o}$ the origin of the tanagent space $T_{\mathbf{o}}\mathfrak{st}$.

Definition 10.3 (ACE-Agent Reward Function R^t_{ACE})**.** *The reward is conditioned on the performance of* $ACE-Agent$ *on a given downstream task compared to the previous epoch.*

$$R^t_{ACE} = \mathcal{C}(\mathbf{h}^{\kappa^t,t}) - \mathcal{C}(\mathbf{h}^{\kappa^t,t-1}) \tag{68}$$

where $(\kappa_1^{t-1},\dots,\kappa_L^{t-1})$ *are the explored curvatures of the last epoch, and* L *is the number of layers in the* Geo-Agent.

where $\mathbf{h}^{\kappa^t,t}$ is the κ-stereographic embedding vector and $\mathcal{C}(\cdot)$ is the evaluation metric of the downstream task.

10.2 GEO-Agent

The Geo-Agent aims to learn node representations on Riemannian manifolds with particular curvatures by taking into account heterogeneous local structures and feature information. Similar to GNNs in Euclidean space, κGCN [BBG20] also follows a message passing scheme **WHICH SHALL BE ADAPTED HERE!**. Given a graph with node features $G = (V, A, \mathbf{h})$, where $\mathbf{h} \in \mathbb{R}^{n\times d}$ and adjacency matrix $A \in \mathbb{R}^{n\times n}$, the Euclidean features are first preprocessed by mapping them onto $\mathfrak{st}_\kappa^d$ via $\mathbf{h} \mapsto \frac{\mathbf{h}}{2\sqrt{|\kappa|}\cdot\|\mathbf{h}\|_{max}}$, where $\|\mathbf{h}\|_{max}$ represents the maximum Euclidean norm among all stereographic embeddings in $\mathbf{h}$. Each layer $\ell+1$ of κGCN take its input from the previous layer ℓ and transforms and aggregates them with different curvatures.

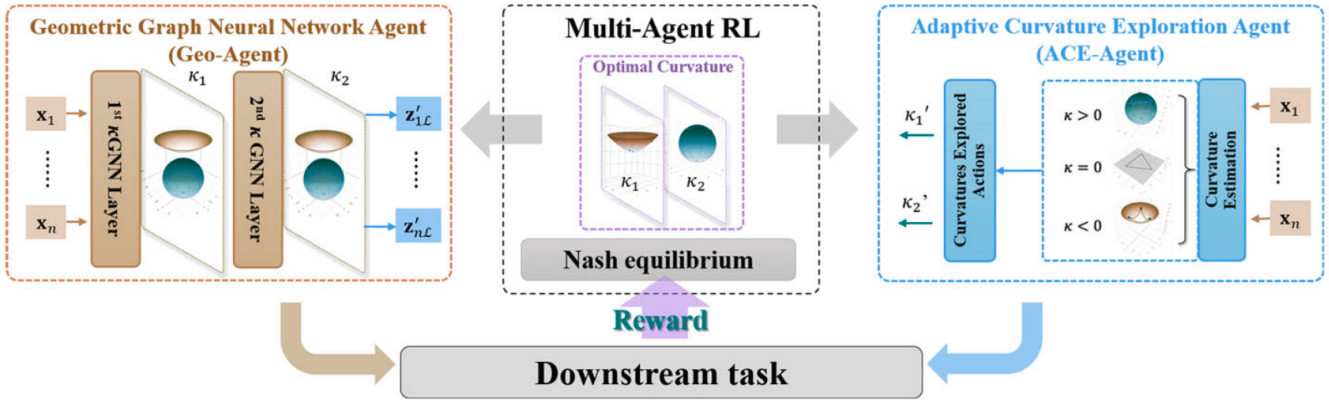
$$\mathbf{h}^{\ell+1} = \sigma^{\oplus_{\kappa',\kappa}}(\tilde{A}\boxplus^{\kappa}(\mathbf{h}^{\ell}\otimes^{\kappa}W^{\ell})) \tag{69}$$

Algorithm 3 ACE-GEO

```
Input: Graph G; Number of training epochs E; Initial curvature κ_0 for each Geo-layer; Exploration probability ϵ.
Output: Predictions of the downstream task.
Initialize model parameters.
for t = 1,2,...,E do
    Get state S_t = (k^{t-1}_GEO, k^{t-1}_ACE).
    Take an action A_t by the ϵ-greedy policy;
    // Geo-Agent
    Calculate node embeddings h^{GEO}_t by eq. (69);
    Get reward R^t_GEO for Geo-Agent by Eq. (28);
    // ACE-Agent
    Calculate node embeddings h^t_ACE by eq. (67);
    Get reward R^t_ACE for ACE-Agent by definition 10.3;
    // Update both agents
    Calculate best policy for each agent π*_GEO(S') and π*_ACE(S') by Eq. (30);
    Update Geo-Agent and ACE-Agent via Nash Q-learning by Eq. (29);
end for
```

where $\sigma^{\otimes_{\kappa',\kappa}}$ is a pointwise non-linear activation function with different curvatures, which allows us to alter the curvature at each level gradually (Gradient decline). As laid out in [Fu+23, p. 10], $\hat{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{D}^{-\frac{1}{2}}$ captures the graph conectivity, where $\tilde{\mathbf{A}} = D^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})D^{-\frac{1}{2}}$. The identity matrix is added, as the fact that a vertex is aware of its own state which also plays a role in the message passing scheme needs to be acounted for. We shall now define the following three operations

Definition 10.4 (κ-Right-Matrix-Multiplication)**.** *Given a matrix* $\mathbf{h} \in \mathbb{R}^{n\times d}$ *holding* d*-dimensional embeddings in* $\mathfrak{st}_\kappa^d$ *and a*



Source: [Fu+23, p. 8]

Figure 20: ACE-GEO framework

weight matrix $W \in \mathbb{R}^{d \times e}$, the κ-right-matrix-multiplication is defined row-wise as:

$$\begin{aligned}(h \otimes^{\kappa} W)_{i\bullet} &= \exp_o^{\kappa}(\log_o^{\kappa}(h)W)_{i\bullet} \\ &= \tan_{\kappa}(\alpha_i \tan_{\kappa}^{-1}(\|h\|_{i\bullet}))\frac{(hW)_{i\bullet}}{\|(hW)_{i\bullet}\|}\end{aligned} \tag{70}$$

where $\alpha_i = \frac{\|(hW)_{i\bullet}\|}{\|h_{i\bullet}\|}$

Definition 10.5 (κ-Left-Matrix-Multiplication)**.** *In Euclidena space, the neighborhood aggregation operation is provided by the left multiplication of the embedding metric with the adjacency matrix, which means the new representation of the node is derived by computing the weighted linear combination of all other nodes in its neighborhood, which is equivalent to applying a* κ*-scaling of a* gyromidpoint*. The* κ*-left-matrix-multiplication is defined row-wise as [Fu+23, p. 10]:*

$$\begin{aligned}(A \boxplus^{\kappa} h)_{i\bullet} &:= (\sum_j A_{ij}) \otimes^{\kappa} m_{\kappa}(h_{1\bullet}, \cdots, h_{n\bullet}; A_{i\bullet}), \\ m_{\kappa}(x_1, \cdots, x_n;) &:= \frac{1}{2} \otimes^{\kappa} \left(\sum_{i=1}^{n} \frac{\alpha_i \lambda_{x_i}^{\kappa}}{\sum_{j=1}^{n} \alpha_j (\lambda_{x_i}^{\kappa} - 1)} x_i \right)\end{aligned} \tag{71}$$

Definition 10.6 (Activation with Variable Curvature)**.** *This activaation function extends - similar to prior definitions - the Euclidean activation function to the Riemannian manifolds with of different curvatures. The tangent space $T_o \mathfrak{st}_{\kappa}^d$ is utilized for this purpose. The Geometric activation function $\sigma^{\otimes_{\kappa',\kappa}}$ with variable curvature is given by:*

$$\sigma^{\otimes_{\kappa',\kappa}}\left(h_{ist}^{d,\kappa}\right) := \exp_o^{\kappa'}\left(\sigma \log_o^{\kappa}\left(h_{ist}^{d,\kappa}\right)\right) \tag{72}$$

Analogous to the ACE-Agent, we define *state*, *action* and *reward*.

Definition 10.7 (Geo-Agent State S_{GEO}^t)**.** *Geo-Agent aims to learn node representations on Riemannian manifolds with* given *curvatures. The Geo-Agent state is defined as follows:*

$$S_{GEO}^t = \left(\kappa_1^{t-1}, \kappa_2^{t-1}, \ldots, \kappa_L^{t-1}\right), \tag{73}$$

where $\left(\kappa_1^{t-1}, \kappa_2^{t-1}, \ldots, \kappa_L^{t-1}\right)$ *are the curvatures at epoch* $t-1$ *and* L *is the number of layers in the model.*

Definition 10.8 (Geo-Agent Action A_{ACE}^t)**.** *The action of Geo-Agent is specified as to whether to update the learned embeddings with the new curvatures. The action space of Geo-Agent is given by the time:*

$$A_{GEO}^t = \kappa_{GEO}^t \leftarrow \kappa_{ACE}^t, \kappa_{GEO}^t \leftarrow \kappa_{GEO}^t \tag{74}$$

Definition 10.9 (Geo-Agent Reward Function R_{ACE}^t)**.** *Similar to definition 10.3, the reward is defined based on the downstream task compared to the last state:*

$$R_{GEO}^t = \mathcal{C}(h^{\kappa^t, t}) - \mathcal{C}(h^{\kappa^t, t-1}) \tag{75}$$

10.3 Collaborative learning

In this section we describe how the two agents are collaboratively trained. The training objective is to converge and reach Nash equillilibrium. Note that in this setup, the downstream tasks performance can only be only affected by collaboration of both agents. The two agents are updated using an ϵ-greedy policy with an exploration probability ϵ when taking actions by Nash Q-learning [HW03]. The Nash Q-lerning optimization fits the Bellman optimality equation as follows:

$$\begin{aligned}\mathrm{Nash}Q_i(S') &= \pi_{GEO}^*(S')\pi_{GEO}^*(S')\pi_{ACE}^*(S')Q_i(S'), \\ Q_i(S, A_{GEO}^t, A_{ACE}^t) &\leftarrow Q_i(S, A_{GEO}^t, A_{ACE}^t) \\ &\quad + \alpha(R_i^t + \beta \mathrm{Nash}Q_i(S') \\ &\quad - Q_i(S, A_{GEO}^t, A_{ACE}^t))\end{aligned} \tag{76}$$

where $Q(\cdot)$ is the Q-function, α is the learning rate and β the discount factor. The RL Algorithm terminates when the GEO-Agent and the ACE-Agent reach Nash equilibrium, and transfers the learned curvatrue κ to the next training process.

$$\pi_{GEO}^*(S'), \pi_{ACE}^*(S') \leftarrow \mathrm{Nash}(Q_{GEO}(S'), Q_{ACE}(S')), \tag{77}$$

where $\pi_{GEO}^*(S'), \pi_{ACE}^*(S')$ are the best policies of the two agents respectively.

10.4 Sorting-Based NN Search

Fixed-radius near neighbour search as described in [CG24] is a fundamental data operation that retrieves all data points within a user-specified distance to a query point. There are efficient algorithms that can provide fast approximate query responses, but they often have a very compute-intensive indexing phase and require careful parameter tuning. Therefore, exact brute force and tree-based search methods are still widely used. [SRR20] proposes a new fixed-radius near neighbour search method, called termed *Nearest neighbour search*, that significantly improves over brute force and tree-based methods in terms of index and query time, provably returns exact results, and requires no parameter tuning. SNN exploits a sorting of the data points by their first principal component to prune the query search space. Further speedup is gained from an efficient implementation using high-level Basic Linear Algebra Subprograms (BLAS). We shall provide theoretical analysis of said method.

Given a data point, this operation aims at finding the most similar data points using a predefined distance function. SNN has many applications in computer science and machine learning, including object recognition [Philbin et al., 2007]. There are two main types of nearest neighbour search: k-*nearest neighbour* and *fixed-radius near neighbour* search. The later aims at finding all neighbouring datapoints within a given distance from the query point. Existing solutions can be divided into two categories: exact solvers and approximate solvers. All approximate NN methods for which open-source implementations are available only address the k-nearest neighbour problem, not the fixed-radius problem

discussed here. Some of the appealing properties of SNN are

1) *simplicity*: SNN has no hyperparameters except for the necessary search radius
2) *exactness*: SNN is guaranteed to return all data points within the search radius
3) *speed*: SNN outperforms other exact NN search algorithms.
4) *flexibility*: the low indexing time of SNN makes it applicable in an online streaming setting

Suppose we have n data points $p_1, \dots, p_n \in \mathbb{R}^d$ with $d \ll n$. The fixed radius problem consists of finding the subset of data points that is closest to a given query point $x_q \in \mathbb{R}^d$ with respect to some distance metric. Although there exist a multitude of possible distance metrics [CG24], this solution implements the Euclidean norm.

10.4.1 Indexing

As we intend to work with the first principal component, we first center the data in $\mathbb{R}^d$

$$x_i := p_i - \bar{p} \tag{78}$$

This operation will not affect the pariwise euclidean distances among datapoints and has $\mathcal{O}(dn)$ time complexity. We then compute the first principal component $v_1 \in \mathbb{R}^d$. The vector can be computed using the singular value decomposition

$$\begin{aligned} X &= U\Sigma V^T \\ X &:= [x_1, \dots, x_n]^T \in \mathbb{R}^{n \times d} \end{aligned} \tag{79}$$

where $U \in \mathbb{R}^{n \times d}$ and $V \in \mathbb{R}^{d \times d}$ are orthononal columns and matrices and $\Sigma = diag(\sigma_1, \dots, \sigma_d) \in \mathbb{R}^{d \times d}$ is a diagonal matrix such that $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_d \geqslant 0$. The principal components are given as the columns of $V = [v_1, \dots, v_d]$ but we require only the first. The score of a point x_i aloong v_1 is defined as

$$\alpha_i := x_i^T v_1 (e_i^T X) v_1 = (e_i^T U \Sigma V^T) v_1 = e_i^T u_1 \sigma_1, \tag{80}$$

where e_i denotes the $i-$ th canonical unit vector in $\mathbb{R}^n$. In otehr words the scores of all points can be read off from the first column of U. The compüutational time complexity using a thin SVD requires $\mathcal{O}(nd^2)$ and is therefore linear in n. The next (and most important) step is to order the datapoints arcording to their scores in ascending order. This is a simople sorting problem with a time complexity og $\mathcal{O}(n \log n)$. We also precomppute $\bar{x_i} = \frac{(x_i^T x_i)}{2}, \quad i = 1, \dots, n$. All this computations are done exactly once, namely in the indexing phase after which only $[\alpha_i], [\bar{x_i}]$ and v_1 need to be stored. See algorithm 4 for a summary.

10.4.2 Query

Given a query point x_q and user-specific search radius R, we want to retrieve all data points p_i satisfying $\|p_i - q\| \leqslant R$ as illustrated in in fig. 21.

Algorithm 4 SNN Index

Input: Data matrix $P = [p_1, p_2, \dots, p_n]^T \in \mathbb{R}^{n \times d}$
1: Compute $\mu := \text{mean}(\{p_j\})$
2: Compute the mean-centered matrix X with rows $x_i := p_i - \mu$
3: Compute the singular value decomposition of $X = U\Sigma V^T$
4: Compute the sorting keys $\alpha_i = x_i^T v_1$ for $i = 1, 2, \dots, n$
5: Sort data points X such that $\alpha_1 \leqslant \alpha_2 \leqslant \dots \leqslant \alpha_n$
6: Compute $\bar{x}_i = \frac{x_i^T x_i}{2}$ for $i = 1, 2, \dots, n$
7: **Return:** $\mu, X, v_1, [\alpha_i], [\bar{x}_i]$

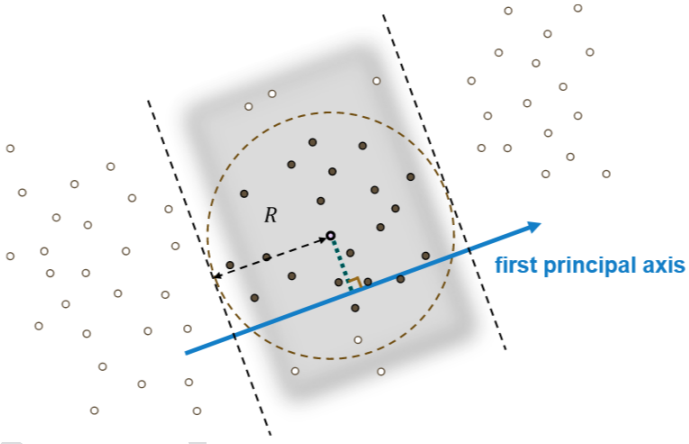


Figure 21: Query with radius R. The data points in the shaded area have their first pc within the R bound around the query point and hence are NN candidates. All data points are sorted so that all candidates have continous indices.

As the query data point comes in it original reference form, we first compute it smean-centered representation by deductin the mean. We also compute its score $\alpha_q := x_q^T v_1$. The Cauchy-Schwarz eniquality yields

$$|\alpha_i - \alpha_q| = |v_1^T x_i - v_1^T x_q| \leqslant \|x_i - x_q\|. \tag{81}$$

Since we have sorted the x_i such that $\alpha_1 \leqslant \cdots \leqslant \alpha_n$, the folllowing statements are true:

$$\begin{aligned} \alpha_q - \alpha_{j_1} > R &\Rightarrow \|x_i - x_q\| > R \forall i \leqslant j_1 \\ \alpha_{j_2} - \alpha_q > R &\Rightarrow \|x_i - x_q\| > R \forall i \geqslant j_2 \end{aligned} \tag{82}$$

This men we need only consider $candidates$ x_i whose indices are in $J = \{j_1 + 1, j_1 + 2, \dots, j_2 - 1\}$ and we can determine the smallest subset by finding the largest j_1 and the smallest j_2 satisfying eq. (82), which can be achieved via binary search in $\mathcal{O}(\log n)$ operations. Finally we filter all datapoints in the subset $X[J, :]$, retaining only those in the shaded area of fig. 21 that satify $\|x_j - x_q\|^2 \leqslant R^2$.

Retrieving $\|x_j - x_q\|^2 = (x_j - x_q)^T (x_j - x_q)$ is computationally costly, requiring $|J|(3d - 1)$ flops to compute all $|J|$ sqaured distances. There are approaches which can be used to reduce the time complexitx of this step, whcih will not be further discussed here, but can be looked into in [CG24]. The computationally more efficient appraoch is given in algorithm 5.

Algorithm 5 SNN Query

Input: Query vector q, user-specified radius R; output from Algorithm 1
Compute $x_q := q - \mu$
Compute the sorting score of x_q, i.e., $\alpha_q := x_q^T v_1$
Select candidate index range J so that $|\alpha_j - \alpha_q| \leqslant R$ for all $j \in J$
Compute $d := \bar{x}(J) - X(J,:)^T x_q$ using the precomputed $\bar{x} = [\bar{x}_i]$
Return: Points x_j with $d_j \leqslant \frac{R^2 - x_q^T x_q}{2}$ according to (4)

11 KITTI Vision Dataset

The framework is tested and benchmarked on the KITTI Vision Dataset [**KITTIVisionBenchmark**]. This choice allows for straightforward benchmarking, as it is the same used in [SRR20]. The project, which was funded by the Karlsruhe Institute of Technology and the Toyota Technological Institute at Chicago provided not only datasets for various applications, such as *multi-object tracking*, *optical flow*, *visual odometry* as well as *3D object detection*, but also offers a benchmarking framework. In the following, we shall explain what the data comprises of, how it was obtained and some of the calculations needed for working with it and by it we mean the 3D object detection dataset.

Figure 22 illustrates the setup of a data collecting vehicle. It is equiped with the following sensors:

- inertial Navigation System: OXTS RT 3003
- Laserscanner: Velodyne HDL-54E
- 2 1.4MP Greyscale cameras: Point Grey Flea 2 (FL2-14S3M-C)
- 2 1.4MP Color cameras: Point Grey Flea 2 (FL2-14S3C-C)
- 4 Varifocal lenses 4-8mm: Edmund Optics NT59-917

In our case we shall be working with the data delivered by the laserscanner, as well as the image data from camera 2. The scanner spins at 10 frames per second, capturing approximately 100k points per cycle. It has a vertical resolution of 64 and triggers the cameras at its rotation frequency with an image being captured whenever it is facing forward. The data was collected in the city of Karlsruhe, Germany in both rural areas as well as highways wiht up to 15 cars and 30 pedestrians per image. The 3D object detection dataset consists of 7481 large training set and 7518 large test set. Each of the sets consists of image data, point cloud data and a calibration file. The training set additionaly has label files, which give us information about the objects in the pointcloud.

11.1 Data Handling

In this section we shall explain how the data is embedded within the project, covering *data loading*, *-transformation* and *-visualization*. Most to all functionalities needed for interacting with the dataset can be found in the *data/kitti/* folder within the project. It is split into a training folder and a testing folder, which hold all the data needed. For interacting with the data, we created the class KittiDataset for which the structure is layed out in Figure 11.1. We shall not go to detailed into the code, as a more detailed documentation is given in the project repository.

The different sensors work with different coordinate systems. To handle this, calibration files are provided, which contain different matrices for coordiante transformations. The get_calib() function is responsible for retrieving this matrices for the corresponding datapoint (point cloud).

12 Implementation

This paper is mainly based on [SRR20; Fu+23]. The laters codebase left little to to nothing to complain about, containing only the required functionalities with sufficient documentation whereas the formers was full of redundencies, missing or incomplete documentation and a rather atypical project structure. The need to quickly get an understanding of both codebases arose. A top-down approach was chosen, first geting an understanding of the general codebasse structure by analysing the interaction between different modules. To do so, we first trying manually mapping out the different modules and their interconnections, which ended up being not at all feasable. In the end, the snakefood3 package was used to automatically map the code base and create a visual representation thereof Figure 23, which made the process of digesting the different implementation and their functionalities as part of the bigger picture more feasable.

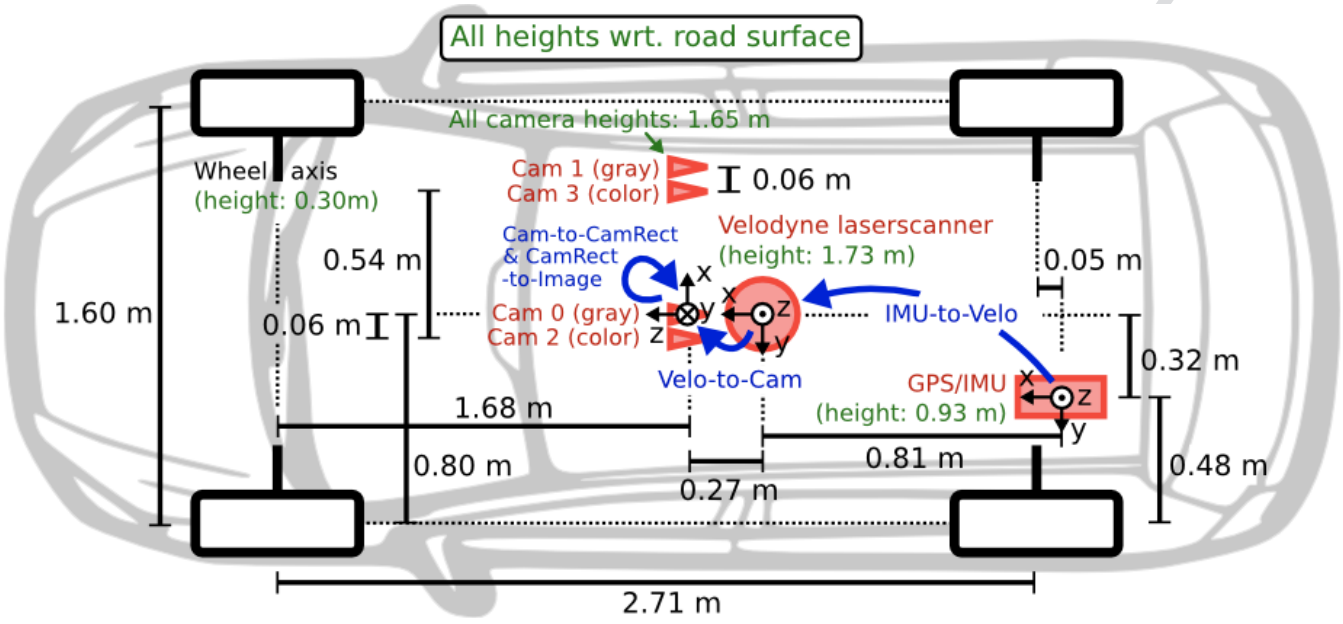
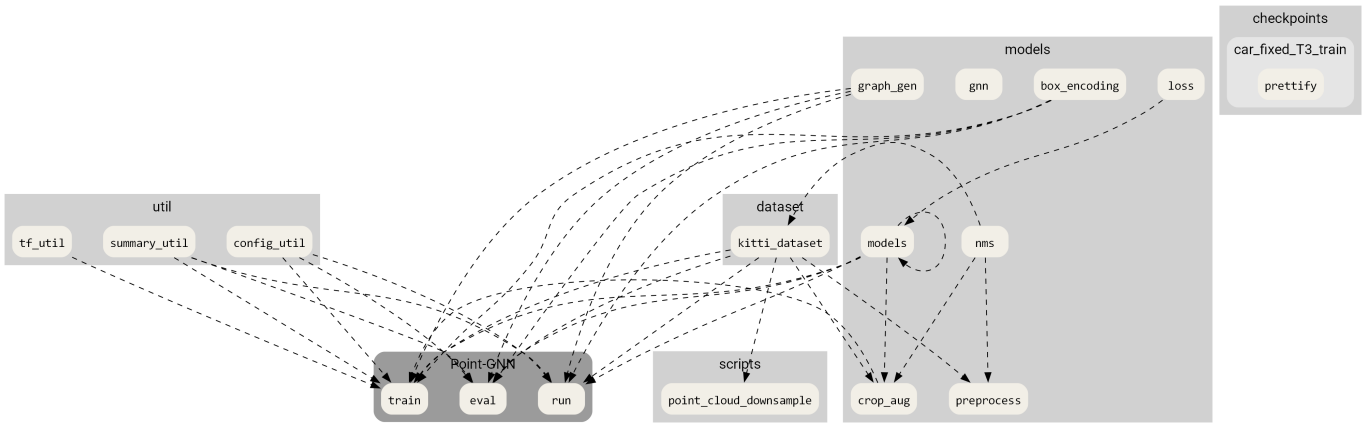


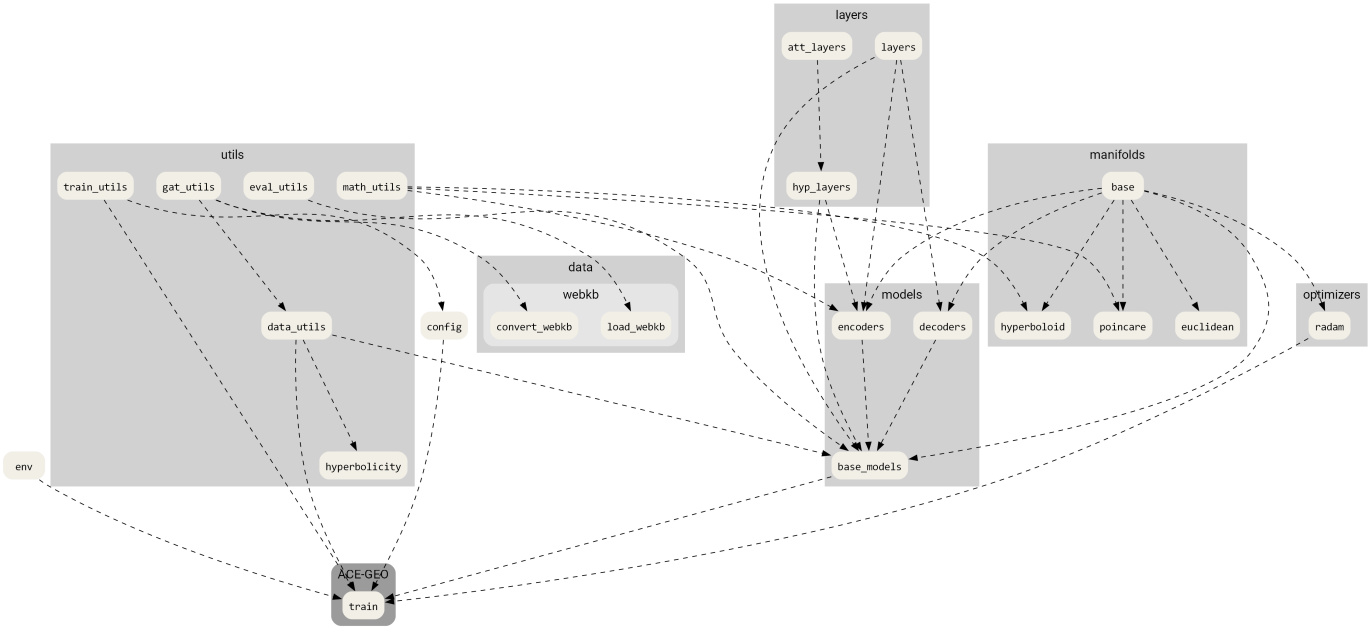
Figure 22: A fully equiped vehicle for data collection

Listing 11.1: Class for interacting wiht kitti 3D object detection dataset

```
class KittiDataset:
  def __init__():                 # Constructor
  def __str__():                  # String with information on dataset

  def get_filename()              # get filename for datapoint index
  def get_image()                 # get image for datapoint index
  def get_label()                 # get label for datapoint index
  def get_calib()                 # get calibration file
  def get_velo_points()           # get velodyne points for datapoint index
  def _get_file_list():           # Returns list of files in dataset
  def _verify_file_list()         # performs a sanity check
  def get_statics()               # returns statics of dataset
  def sqdistance()                # squared distance between points

  def boxes_3d_to_line_set()      # creates edges to the vertices of bbox
  def downsample_by_voxel()       # downsamples pointcloud
  def velo_points_to_image()      # projects velodyne points onto image

  def get_cam_points()            # converts velodyne to camera coordinates
  def get_cam_points_in_image()   # projects cam view points onto image plane
  def get_cam_points_in_image_with_rgb()  # assigns points the colors given by image

  def get_open3D_box()            # creates open3D box from label
  def inspect_points()            # visualizes pointcloud
  def rgb_to_cam_points()         # applies colors to points
  def sel_points_in_box3d()       # filters point within bbox
  def vis_draw_2d_box()           # draws 2D box on image
  def vis_draw_3d_box()           # draws 3D box on image
  def vis_points()                # visualizes points

```



(a) Point-GNN Codebase



(b) ACE-GEO Codebase

Figure 23: (a) Codebases of [HW03; Fu+23]

APPENDIX A

PROOF OF lemma 3.1

Proof. Suppose after k iterations, a graph neural network $\mathcal{A}$ has $\mathcal{A}(G_1) \neq \mathcal{A}(G_2)$ but the WL test cannot decide if G_1 and G_2 are non-isomorphic. t follows that from iteration 0 to k in the WL test, G_1 and G_2 always have the same collection of node labels. In particular, because G_1 and G_2 have the same WL node labels for iteration i and $i+1$ for any $i = 0, \ldots, k-1$, G_1 and G_2 have the same collection, i.e. multiset, of WL node labels $\{l_v^{(i)} \texttt{bigr}\}$ as well as the same collection of node neighbourhoods $\{\left(l_v^{(i)}, \{l_u^{(i)} : u \in \mathcal{N}(v)\}\right)\}$. Otherwise, the WL test would have obtained differnt collections of node labels at iteration $i+1$ for G_1 and G_2 as differnt multisets get unique new labels.

The WL test always relabels different multisets of neigbouring nodes into differnt new labels. We show that on the same graph $G = G_1$ or G_2, if WL node labels $l_v^{(i)} = l_u^{(i)}$, we always have GNN node features $\mathbf{h}_v^{(i)} = \mathbf{h}_u^{(i)}$ for any iteration i. This apperently holds for $i = 0$ beacuse WL and GNN starts with the same node features. Suppose this holds for iteration j, if for any $u, v, l_v^{(j+1)} = l_u^{(j+1)}$, then it must be the case that

$$\left(l_v^{(j)}, \{l_w^{(j)} : w \in \mathcal{N}(v)\}\right) = \left(l_u^{(j)}, \{l_w^{(j)} : w \in \mathcal{N}(u)\}\right)$$

by assumption on iteration j, we must have

$$\left(\mathbf{h}_v^{(j)}, \{\mathbf{h}_w^{(j)} : w \in \mathcal{N}(v)\}\right) = \left(\mathbf{h}_u^{(j)}, \{\mathbf{h}_w^{(j)} : w \in \mathcal{N}(u)\}\right)$$

In the aggregation process of the GNN, the same AGGREGATE and COMBINE are applied. The same input, i.e. neighbourhood features, generates the same output. Thus, $\mathbf{h}_v^{(j+1)} = \mathbf{h}_u^{(j+1)}$. By induction, if WL node labels $l_v^{(i)} = l_u^{(i)}$, we always have GNN node features $\mathbf{h}_v^{(i)} = \mathbf{h}_u^{(i)}$ for any iteration i. This creates a valid mapping ϕ such that $\mathbf{h}_v^{(i)} = \phi(l_v^{(i)}) \forall v \in G$. It also follows from collection of GNN neighbourhood features

$$\begin{aligned} &\{\left(\mathbf{h}_v^{(i)}, \{\mathbf{h}_u^{(i)} : u \in \mathcal{N}(v)\}\right)\} = \\ &\{\left(\phi(l_v^{(i)}), \{\phi(l_u^{(i)}) : u \in \mathcal{N}(v)\}\right)\} \end{aligned} \tag{83}$$

Thus, $\{\mathbf{h}_v^{(i+1)}\}$ are the same. in particular, we have the same collection of GNN node features $\{\mathbf{h}_v^{(k)}\}$ for G_1 and G_2. Because the graph level readout function is permutation invariant with respect to the collection of node features, $\mathcal{A}(G_1) = \mathcal{A}(G_2)$. Hence we have reached a contradiction. □

PROOF OF theorem 3.1

Proof. Let $\mathcal{A}$ be a graph neural network where the condition holds. Let G_1, G_2 be any graphs which the WL test decides are non isomorphic at iteration K. Because the graph-level readout function is injective, i.e., it maps distinct multiset of node fefatures into unique embeddings, it suffices to show that $\mathcal{A}$'s neighbourhood aggregation process, with sufficient iterations, embeds G_1 and G_2 into different multisets of node features. let us assume $\mathcal{A}$ updates node representations as

$$h_v^{(k)} = \phi\left(h_v^{(k-1)}, f\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right)\right) \tag{84}$$

with injective functions f and ϕ. The WL test applies a predetermined injective hash function g to update the WL node labels $l_v^{(k)}$:

$$l_v^{(k)} = g\left(l_v^{(k-1)}, \left\{l_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right) \tag{85}$$

We will show, by induction, that for any iteration k, there always exists an injective function φ such that $h_v^{(k)} = \varphi\left(l_v^{(k)}\right)$. This holds for $k = 0$ because the initial node features are the same for WL and GNN $l_v^{(0)} = h_v^{(0)} \forall v \in G_1, G_2$. So φ could be the identity function for $k = 0$. Suppose this holds for iteration $k-1$, we show that it also holds for k. Substituting $h_v^{(k-1)}$ with $\varphi\left(l_v^{(k-1)}\right)$ gives use

$$h_v^{(k)} = \phi\left(\varphi\left(l_v^{(k-1)}\right), f\left(\left\{\varphi\left(l_u^{(k-1)}\right) : u \in \mathcal{N}(v)\right\}\right)\right) \tag{86}$$

Since the composition of injective functions is injective, there exists some injective function ψ so threaten

$$h_v^{(k)} = \psi\left(l_v^{k-1}, \left\{l_u^{k-1} : u \in \mathcal{N}(v)\right\}\right) \tag{87}$$

Then we have

$$h_v^{(k)} = \psi \circ g^{-1}\left(l_v^{k-1}, \left\{l_u^{k-1} : u \in \mathcal{N}(n)\right\}\right) = \psi \circ g^{-1}\left(l_v^{(k)}\right) \tag{88}$$

$\varphi = \psi \circ g^{-1}$ is injective because of above mentioned argument. Hence for any iteration k, there always exists an injective function φ such that $h_v^{(k)} = \varphi\left(l_v^{(k)}\right)$. At the K-th iteration, the WL test decides that G_1 and G_2 are non-isomorphic, that is the multisets $\left\{l_v^{(K)}\right\}$ are different for G_1 adn G_2. The graph neural network $\mathcal{A}$'s node embeddings $\left\{h_v^{(K)}\right\} = \left\{\varphi\left(l_v^{(K)}\right)\right\}$ must also be differnt for G_1 and G_2 because of the injectivity of φ. □

PROOF OF AXIOMS IN definition 5.11

Proof.

P1: Let $P, Q \in S$

- If P, Q are ordinary points of A, then P and Q lie on only one line of A. The do not lie on the line at infinity of S, hence they lie on only one line of S.
- If P is an ordinaty point, and $Q = [l]$ is an ideal point, we can find by A2 a line m such that $P \in m$, and $m \| l$, i.e. $m \in [l]$, so that Q lies on the extension of m to S. This is clearly the only line of S containing P and Q.
- If P, Q are both ideal points, then they both lie on the line of S containing them.

P2: Let l, m be lines

- if they are both ordinary lines, and $l \| m$, then they meet in a point of A. If $l \| m$, then the ideal point $P^* = [l] = [m]$ lies on both l and m in S.

P3: Follows immediately from A3. One must check only that if P, Q, R are non-collinear in S. Indeed, the only new line is the line at infinity, which contains none of them.

P4: Each line contains at least two points. Hence in S it has also its point at infinity, so has at least three points.

□

APPENDIX B

PROOF OF theorem 6.2

A strong condition implicit in the theorem is an infinte sequence of episodes that forms the basis of learning. The key to this proof is an artificial controlled Markov process called the *action-replay process* ARP, which is constructed from the episode sequence and the learning rate sequence β_i, which shall be decribed in the following.

Definition B.1 (Action-Replay Process)**.** *The definition of the ARP is contingent on a particular sequence of episodes observed in the real process. The state space of the ARP is* $\{\langle s, i\rangle\}$*, for* s *a state of the real process and* $i \geqslant 1$ *together with one, special, absorbing state, and the action space is* $\{a\}$ *for* a *an action from the real process.*

The stochastic reward and state transistion consequent on performing action a *at state* $\langle s, i\rangle$ *is given as follows. For convenience, we define* $\tilde{i} \equiv \tilde{i}(s, a)$*, as the index of the* i^{th} *time action* a *was tried at state* s*. Define*

$$i_* = \begin{cases} \text{argmax}_i\{\tilde{i} < i\} & \text{if } s, a \text{ has been executed before episode } i \\ 0 & \text{otherwise} \end{cases}$$

such that $\tilde{i}$ *is the last time before episeode* i *that* s, a *was executed in the real process. If* $i_* = 0$*, the reward is set as* $Q_o(s, a)$*, and the ARP absobs. Otherwise, Let*

$$i_e = \begin{cases} i_* & \text{with probability } \beta_{\tilde{i}} \\ i_* - 1 & \text{with probability } (1 - \beta_{\tilde{i_*}})\beta_{\tilde{i_*-1}} \\ i_* - 2 & \text{with probability } (1 - \beta_{\tilde{i_*}})(1 - \beta_{\tilde{i_*-1}})\beta_{\tilde{i_*-2}} \\ & \vdots \\ 0 & \text{with probability } \prod_{i=1}^{i_*}(1 - \beta_{\tilde{i}}) \end{cases}$$

be the index of the episode that is replaid *or taken, chosen probabilistically from the collection of existing samples from the real process. If* $i_e = 0$*, then the reward is set at* $Q_0(s, a)$ *and the ARP absorbs, otherwise,* i_e *provides reward* $r_{\pi^{i_e}}$*, and causes a state transition to* $\langle s_{i+1(\tilde{i_e})}, \tilde{i_e} - 1\rangle$ *which is at level* $\tilde{i_e} - 1$*. This last point is crucial, taking an action in the ARP always causes a state transition to a lower level - so it ultimately terminates. The discount factor in the ARP is* γ*, the same as in the real process.*

With the definition given we can now move on to the proof

The convergence proof. Define $P^{ARP}_{\langle s_i, i\rangle, \langle s_{i+1}, k\rangle}[a]$ and $\mathcal{U}^{(i)}_{s_i}(a)$ as the transition-probaility matrices and expected rewards of the ARP. Also define:

$$P^{(i)}_{s_i s_{i+1}}[a] = \sum_{k=1}^{i-1} P^{ARP}_{\langle s_i, i\rangle, \langle s_{i+1}, k\rangle}[a] \tag{89}$$

as the probabilities that, for each s_i, i and a, executing action a at atate $\langle s_i, i\rangle$ in the ARP leads to state s_{i+1} of the real process at some lower level ín the deck.

by definition of the ARP, it is as much a controlled Markov process as is the real process. One can therefore consider sequences of states and controls, and also optimal discounted Q^* values for the ARP. Note that during such a sequence, episode cards are only removed from the deck, adn are never replaces. Therefore, after a fintie number of actions, the bottom card will always be reached. □

References

[Ban+99] Cesar Bandera et al. "Residual Q-Learning Applied to Visual Attention". In: *ICML-96* (Feb. 15, 1999).

[BBG20] Gregor Bachmann, Gary Bécigneul, and Octavian-Eugen Ganea. *Constant Curvature Graph Convolutional Networks*. May 19, 2020. DOI: 10.48550/arXiv.1911.05076. arXiv: 1911.05076 [cs]. URL: http://arxiv.org/abs/1911.05076 (visited on 01/26/2025). Pre-published.

[Boo75] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Pure and Applied Mathematics. New York: Academic Press, 1975. 424 pp. ISBN: 978-0-08-087379-4.

[Cai19] Leran Cai. "Everything about Spectral Graph Theory". In: *Cambridge University* (Dec. 4, 2019), p. 27. URL: https://www.cl.cam.ac.uk/~lc647/drafts/.

[CG24] Xinye Chen and Stefan Güttel. *Fast and Exact Fixed-Radius Neighbor Search Based on Sorting*. Jan. 29, 2024. DOI: 10.48550/arXiv.2212.07679. arXiv: 2212.07679 [cs]. URL: http://arxiv.org/abs/2212.07679 (visited on 01/11/2025). Pre-published.

[Cha+17] R. Qi Charles et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI: IEEE, July 2017, pp. 77–85. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.16. URL: http://ieeexplore.ieee.org/document/8099499/ (visited on 01/13/2025).

[Chu09] Fan R. Chung. *Spectral Graph Theory: CBMS Conference on Recent Advances in Spectral Graph Theory, Held at ... Fresno, June 6 - 10, 1994*. reprint. Regional Conference Series in Mathematics 92. Providence, RI: American Mathematical Society, 2009. 212 pp. ISBN: 978-0-8218-0315-8.

[Chu97] Fan Rong K. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics 92. Providence (R. I.): American mathematical society, 1997. ISBN: 978-0-8218-0315-8.

[Die17] Reinhard Diestel. *Graph Theory*. Vol. 173. Graduate Texts in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017. ISBN: 978-3-662-53621-6 978-3-662-53622-3. DOI: 10.1007/978-3-662-53622-3. URL: https://link.springer.com/10.1007/978-3-662-53622-3 (visited on 02/09/2025).

[Dou11] B. L. Douglas. *The Weisfeiler-Lehman Method and Graph Isomorphism Testing*. Jan. 27, 2011. DOI: 10.48550/arXiv.1101.5211. arXiv: 1101.5211 [math]. URL: http://arxiv.org/abs/1101.5211 (visited on 02/10/2025). Pre-published.

[Duc17] John Duchi. "VC-Dimension, Covering, and Packing". In: *Stanford University* (Mar. 2, 2017), p. 4. URL: https://web.stanford.edu/class/stats300b/ (visited on 02/15/2024).

[Fin64] A. M. Fink. "Equilibrium in a Stochastic n-Person Game". In: *Hiroshima Mathematical Journal* 28.1 (Jan. 1, 1964). ISSN: 0018-2079. DOI: 10.32917/hmj/1206139508. URL: https://projecteuclid.org/journals/hiroshima-mathematical-journal/volume-28/issue-1/Equilibrium-in-a-stochastic-n-person-game/10.32917/hmj/1206139508.full (visited on 02/18/2025).

[FT91] Drew Fudenberg and Jean Tirole. *Game Theory*. Cambridge (Mass.): the MIT press, 1991. ISBN: 978-0-262-06141-4.

[Fu+23] Xingcheng Fu et al. "Adaptive Curvature Exploration Geometric Graph Neural Network". In: *Knowledge and Information Systems* 65.5 (May 2023), pp. 2281–2304. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-022-01811-4. URL: https://link.springer.com/10.1007/s10115-022-01811-4 (visited on 01/24/2025).

[GBH18] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. *Hyperbolic Neural Networks*. June 28, 2018. DOI: 10.48550/arXiv.1805.09112. arXiv: 1805.09112 [cs]. URL: http://arxiv.org/abs/1805.09112 (visited on 02/04/2025). Pre-published.

[GKH24] Nikoleta E. Glynatsi, Vincent Knight, and Marc Harper. "Properties of Winning Iterated Prisoner's Dilemma Strategies". In: *PLOS Computational Biology* 20.12 (Dec. 26, 2024). Ed. by Alexandre V. Morozov, e1012644. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1012644. URL: https://dx.plos.org/10.1371/journal.pcbi.1012644 (visited on 02/17/2025).

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR. 2012. URL: https://www.cvlibs.net/datasets/kitti/.

[Har09] Robin Hartshorne. *Foundations of Projective Geometry: By Robin Hartshorne*. New edition with an added appendix. Bronx, NY: Ishi Press International, 2009. 177 pp. ISBN: 978-4-87187-837-1.

[Har15] Matthew Harvey. *Geometry Illuminated: An Illustrated Introduction to Euclidean and Hyperbolic Plane Geometry*. AMS/MAA Textbooks v. 30. Washington, D.C: Mathematical Association of America, 2015. 1 p. ISBN: 978-1-61444-618-7.

[HBS17] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. *Learning Non-Maximum Suppression*. May 9, 2017. DOI: 10.48550/arXiv.1705.02950. arXiv: 1705.02950 [cs]. URL: http://arxiv.org/abs/1705.02950 (visited on 01/13/2025). Pre-published.

[Hil22] David Hilbert. *Grundlagen der Geometrie*. 5th ed. Wissenschaft und Hypothese Ser. Wiesbaden: Springer Vieweg. in Springer Fachmedien Wiesbaden GmbH, 1922. 1 p. ISBN: 978-3-663-15383-2 978-3-663-15954-4.

[HLS13] David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. 1st ed. Wiley Series in Probability and Statistics. Wiley, Mar. 22, 2013. ISBN: 978-0-470-58247-3 978-1-118-54838-7. DOI: 10.

1002/9781118548387. URL: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118548387 (visited on 02/19/2025).

[Hub92] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *Breakthroughs in Statistics*. Ed. by Samuel Kotz and Norman L. Johnson. New York, NY: Springer New York, 1992, pp. 492–518. ISBN: 978-0-387-94039-7 978-1-4612-4380-9. DOI: 10.1007/978-1-4612-4380-9_35. URL: http://link.springer.com/10.1007/978-1-4612-4380-9_35 (visited on 01/13/2025).

[HW03] Junling Hu and Michael P. Wellman. "Nash Q-Learning for General-Sum Stochastic Games". In: *Journal of Machine Learning Research* 4 (Nov 2003), pp. 1039–1069. ISSN: ISSN 1533-7928. URL: https://jmlr.csail.mit.edu/papers/v4/hu03a (visited on 02/08/2025).

[HYL18] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. Sept. 10, 2018. DOI: 10.48550/arXiv.1706.02216. arXiv: 1706.02216 [cs]. URL: http://arxiv.org/abs/1706.02216 (visited on 02/10/2025). Pre-published.

[KW17] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. Feb. 22, 2017. DOI: 10.48550/arXiv.1609.02907. arXiv: 1609.02907 [cs]. URL: http://arxiv.org/abs/1609.02907 (visited on 02/10/2025). Pre-published.

[MP43] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259 (visited on 02/14/2025).

[Mun00] James Raymond Munkres. *Topology*. 2nd ed. Upper Saddle River (N. J.): Prentice Hall, 2000. ISBN: 978-0-13-181629-9.

[NV19] C.R. Noordman and G.A.W. Vreeswijk. "Evolving Novelty Strategies for the Iterated Prisoner's Dilemma in Deceptive Tournaments". In: *Theoretical Computer Science* 785 (Sept. 2019), pp. 1–16. ISSN: 03043975. DOI: 10.1016/j.tcs.2018.10.026. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304397518306479 (visited on 02/17/2025).

[Pet16] Peter Petersen. *Riemannian Geometry*. 3rd ed. 2016. Graduate Texts in Mathematics 171. Cham: Springer International Publishing : Imprint: Springer, 2016. 1 p. ISBN: 978-3-319-26654-1. DOI: 10.1007/978-3-319-26654-1.

[Pra21] Jatin Prakash. *Non Maximum Suppression: Theory and Implementation in PyTorch*. June 2, 2021. URL: https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/ (visited on 02/14/2025).

[Pri19] Margitta Pries. "Geometry". manuscript. Berlin University of Applied Sciences, Oct. 1, 2019. URL: https://www.bht-berlin.de/people/detail/617 (visited on 02/01/2025).

[RP08] Rajeev Ranjan and Anukriti Phophalia. "Reinforcement Learning for Dynamic Channel Allocation in Mobile Cellular Systems". In: *2008 International Conference on Recent Advances in Microwave Theory and Applications*. 2008 International Conference on Recent Advances in Microwave Theory and Applications (MICROWAVE). Jaipur, Rajasthan, India: IEEE, Nov. 2008, pp. 924–927. ISBN: 978-1-4244-2690-4. DOI: 10.1109/AMTA.2008.4763228. URL: http://ieeexplore.ieee.org/document/4763228/ (visited on 02/17/2025).

[RS00] Sam T. Roweis and Lawrence K. Saul. "Nonlinear Dimensionality Reduction by Locally Linear Embedding". In: *Science* 290.5500 (Dec. 22, 2000), pp. 2323–2326. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.290.5500.2323. URL: https://www.science.org/doi/10.1126/science.290.5500.2323 (visited on 02/10/2025).

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. New York: Cambridge university press, 2014. ISBN: 978-1-107-05713-5.

[Sip10] Michael Sipser. *Introduction to the Theory of Computation*. 2. ed., internat. ed., [Nachdr.] Boston: Course Technology, 2010. 437 pp. ISBN: 978-0-619-21764-8.

[SRR20] Weijing Shi, Ragunathan, and Rajkumar. *Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud*. Mar. 2, 2020. DOI: 10.48550/arXiv.2003.01251. arXiv: 2003.01251 [cs]. URL: http://arxiv.org/abs/2003.01251 (visited on 01/10/2025). Pre-published.

[TSL00a] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 22, 2000), pp. 2319–2323. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.290.5500.2319. URL: https://www.science.org/doi/10.1126/science.290.5500.2319 (visited on 02/10/2025).

[TSL00b] Joshua B. Tenenbaum, Vin De Silva, and John C. Langford. "A Global Geometric Framework for Nonlinear Dimensionality Reduction". In: *Science* 290.5500 (Dec. 22, 2000), pp. 2319–2323. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.290.5500.2319. URL: https://www.science.org/doi/10.1126/science.290.5500.2319 (visited on 02/11/2025).

[Ung07] Abraham A. Ungar. "Einstein's Velocity Addition Law and Its Hyperbolic Geometry". In: *Computers & Mathematics with Applications* 53.8 (Apr. 2007), pp. 1228–1250. ISSN: 08981221. DOI: 10.1016/j.camwa.2006.05.028. URL: https://linkinghub.elsevier.com/retrieve/pii/S089812210700082X (visited on 02/08/2025).

[Ung09] Abraham A. Ungar. *A Gyrovector Space Approach to Hyperbolic Geometry*. Synthesis Lectures on Mathematics and Statistics 4. San Rafael: Morgan & Claypool, 2009. 182 pp. ISBN: 978-1-59829-822-2 978-1-59829-823-9.

[Wal00] M. A. Walker. "An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email". In: *Journal of Artificial Intelligence Research* 12 (June 1, 2000), pp. 387–416. ISSN: 1076-9757. DOI: 10.1613/jair.713. arXiv: 1106.0241 [cs]. URL: http://arxiv.org/abs/1106.0241 (visited on 02/17/2025).

[WD92] Christopher J. C. H. Watkins and Peter Dayan. "Q-Learning". In: *Machine Learning* 8.3–4 (May 1992), pp. 279–292. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00992698. URL: http://link.springer.com/10.1007/BF00992698 (visited on 02/09/2025).

[Wei] Eric W. Weisstein. *Parallelogram Law*. URL: https://mathworld.wolfram.com/ParallelogramLaw.html (visited on 02/08/2025).

[Wu+22] Lingfei Wu et al., eds. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Nature Singapore, 2022. 701 pp. ISBN: 978-981-16-6053-5 978-981-16-6054-2. DOI: 10.1007/978-981-16-6054-2. URL: https://link.springer.com/10.1007/978-981-16-6054-2 (visited on 01/11/2025).

[Xu+19] Keyulu Xu et al. *How Powerful Are Graph Neural Networks?* Feb. 22, 2019. DOI: 10.48550/arXiv.1810.00826. arXiv: 1810.00826 [cs]. URL: http://arxiv.org/abs/1810.00826 (visited on 02/10/2025). Pre-published.