# 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.

At some point in the distant past, a young boy and his class were invited to Audi's headquarters in the german city 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 automated automobiles is amongst the field who have directly benefited from the propgress made in the field. 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 recent developments in computer vision might not rival successes such as the unlocking of protein folding or other developments in 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 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 one 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 §**??**.

The basic idea is to bring together the ideas presented in [**shiPointGNNGraphNeural2020**; **fuAdaptiveCurvatureExploration2023**]. The Point-GNN architecture estimates a high dimensional manifold in Euclidean space that describes node classes. This however makes a crucial assumption, which is that the best descriptive manifold is embedded in a geometry with no curvature. It can be thought of as visiting a new culture and trying a dishj at a local restaurant and then rating said cultures quizine relative to other quizines based on that one restaurant. Even though the food might have very well been tasty, one restaurant can hardly be represent an entire quizine. What if it was possible to identify the best restaurant? Even more - what if you could find the restaurant best suited for your individual tase and preferences. Using this analogy, the Point-GNN architecture is the culture and the different restaurants in said culture are differently curved spaces and your individual taste represents the downstream task.

## 2 Graphs in Machine Learning

### 2.1 Neural Networks

A prerequisit of understanding how graphs (§**??**)s can be incorporated into machine learning frameworks is a broad repetoire of the general machine learning architectures. So in this section an introduction into the history of machine learning as well as a consise explanation of the mathematical ideas on how silicon can be taught to learn - in a very rudamentary sense that is.

Machine learninig is one of the fastest growing areas of computer science, with far reaching implications. 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 expectation, which this novel technology that has grown into a healthy forrest full of trees of all kinds and even more seeds waiting to be planted, will lead to significant advances across the entire spectrum of natural sciences that might rival those made in the early $20^{\text{th}}$ century does not seem to far fetched.

An *artificial neural network* is a model of computation inspired by the neurons found in the human brain itself. 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-$20^{\text{th}}$ century [**mccullochLogicalCalculusIdeas1943**] and has been shown to yield cutting-edge performance on several tasks. A neural network can be described in terms of a directed graph

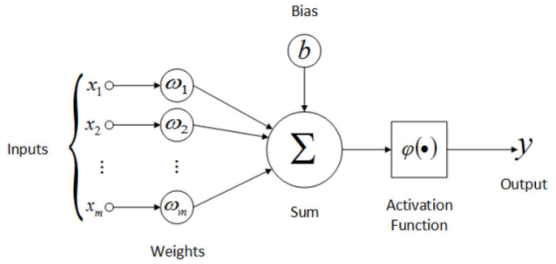



Figure 1: Artifical Neuron, source: [**shalev-shwartzUnderstandingMachineLearning2014**]

whose nodes correspond to neurons and edges to the link between them. Each neuron, such as the one illustrated in **??**, 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 *feed-forward* networks that do not contain any cycles.

As expressed in [**shalev-shwartzUnderstandingMachineLearning2014**], 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. Such a predictor has both polynomial sample complexity as well as the minimal approximation 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, $\sigma$ 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. The hypothesis class is denoted

$$\mathcal{H}_{V,E,\sigma} = \{\mathcal{H}_{V,E,\sigma,w} : w \text{ is a mapping } E \to \mathbb{R}\} \tag{1}$$

Naturally the question of what kinds of functions can be approximated to satisfaction using such a predictor arises. 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 $g : \{\pm 1\}^{nb} \to \{\pm 1\}^b$. Therefore studying the space of possible boolean functions implicitly gives us the space of possible real functions. One therefore studies the boolean functions $f_b : \{\pm 1\}^n \to \{\pm\}$, which can be implemented in $\mathcal{H}_{V,E,\sigma}$. A simple claim is made that the set of all boolean functions can be implemented using a neural network of depth 2.

**Proposition 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 $\{\pm 1\}^n \to \{\pm 1\}$* [**shalev-shwartzUnderstandingMachineLearning2014**]*.*

**Proof 1.** *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 $f : \{\pm 1\}^n \to \{\pm 1\}$ be some Boolean function. One needs to show that the weights can be adjusted, so that the network implements $f$. Let $u_1, \ldots, u_k$ be all vectors in $\{\pm 1\}^n$ on which $f$ evaluates to 1. Observe that for every $i$ and every $x \in \{\pm 1\}^n$, if $x \neq u_i$ then $\langle x, u_i \rangle \leqslant n - 2$ and if $x = u_i$ then $\langle x, u_i \rangle = n$. it follows that the function $g_i(x) = \mathrm{sign}(\langle x, u_i \rangle - n + 1)$ equals 1 iff $x = u_i$. 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 $g_i(x)$. Next, we observe that $f(x)$ is the disjunction of the functions $g_i(x)$, and therefore can be written as*

$$f(x) = \mathrm{sign}\left(\sum_{i=1}^{k} g_i(x) + k - 1\right), \tag{2}$$

*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 in **??**, the number of nodes in the hidden layer is exponentially large.

**Theorem 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 $\sigma$ is the sigmoid function.*

**Proof 2.** *Suppose that for some $(V, E)$ we have that $\mathcal{H}_{V,E,\mathtt{sign}}$ contains all functions $\{0,1\}^n$ 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* [**duchiVCDimensionCoveringPacking2017**] *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 bound by $\mathcal{O}(|E|\log(|E|)) \leqslant \mathcal{O}(|V|^3)$. As shown in* [**shalev-shwartzUnderstandingMachineLearning2014**]*, 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 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 3.** *The proof relies on the relation between the time complexity of programs and their circuit complexity* [**sipserIntroductionTheoryComputation2010**]*. 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*

*[**shalev-shwartzUnderstandingMachineLearning2014**] 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 [**shalev-shwartzUnderstandingMachineLearning2014**], utilizing the VC dimension.

**Theorem 3** (VC dimension)**.** *The VC dimension of* $\mathcal{H}_{\mathrm{V},\mathrm{E},\mathtt{sign}}$ *is* $\mathcal{O}(|\mathrm{E}|\log(|\mathrm{E}|))$.

### 2.2 Graph Neural Networks

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

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

Although in theory, the framework presented could be adapted to a multitude of fields and use cases, the idea here is to formulate a framework in the domain of *computer vision*, in which a dataset is a point cloud, with the pixels being the points. Here our pixel positions lie in $\mathbb{R}^4$ instead of $\mathbb{N}^4$. 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 into acount opacity. Keep in mind that the term *feature* does not have the same meaning as in the usual machine learning context. More details on the dataset is presented in §**??**.

If creativity were quantifiable, it would be fair to say there is not and has never been any human innovation as the result of pure creativity. All humans can do is take a close enough look at nature and let themselves be inspired by it - maybe a direct consequence of Plato's allegory of the cave. The 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) fed to it. In this case an attempt is made 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, consider optical Illusions, such as the one presented in **??**, which only occurs due to the lack of information on depth and/or the location of the luminating source. Naturally the left dots might appear concave and the right ones convex. This is only the case, because the human brain has been conditioned over millenia to expect light rays to come from high above (the sun). The truth is, given only the visual information it is simply impossible to tell which is what. This problem would immeadiately be resolved is the image was presented as a point cloud.

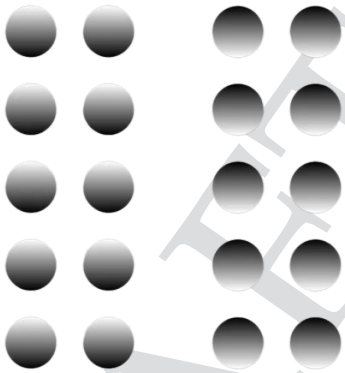


Figure 2: An optical illusion, source: mentalbomb.com

#### 2.2.1 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. Graph embedding is an effective yet efficient way to solve the graph analytics problem. It converts the graph data into a low dimensional space in which the graph structural information and graph properties are maximumly preserved [**caiComprehensiveSurveyGraph2018**]. 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 [**tenenbaumGlobalGeometricFramework2000**] first constructs a neighbourhood graph G using algorithms

such as k-nearest neighbours (KNN). 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 representations learned by ISOMAP [**tenenbaumGlobalGeometricFramework2000a**] approximately preserve the geodesic distances of the entry pairs in the low-dimensional space with the main problem being the computational cost due to the need of pairwise computations. This was seemingly addressed by [**roweisNonlinearDimensionalityReduction2000**] 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 [**wuGraphNeuralNetworks2022**], graph embedding 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 molecular structures, traffic networks, social networks or even point cloud data. In those networks the proximities among nodes are not explicitly or directly defined as hinted in §**??**. 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, modern graph embedding usually incorporates rich information, such as network (graph) structures, properties, side information, to facilitate different problems and applications.

#### 2.2.2 General Framework of Graph Neural Networks

In this section the general framework of graph neural networks is introduced as presented in [**xuHowPowerfulAre2019a**]. 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 one is given a set of graphs $\{G_1, \dots, G_N\} \subseteq G$ and their corresponding labels $\{y_1, \dots, y_N\} \subseteq \mathcal{Y}$. The goal is to learn a representation vector $\mathbf{h}_G$ that helps predict the label of an entire graph, $y_G = f(\mathbf{h}_G)$.

**Definition 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:*

$$\begin{aligned} a_v^{(k)} &= \text{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_u^{k-1} : u \in \mathcal{N}(v)\right\}\right), \\ \mathbf{h}_v^{(k)} &= \text{COMBINE}^{(k)}\left(\mathbf{h}_v^{k-1}, a_v^{(k)}\right), \end{aligned} \tag{3}$$

*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 and it can be said that this two define an architecture.*

A visual analogy would be heat diffusion process within a compound. In such a scenario the process is ofcourse non discrete. However if the compund displays a non homogenous heat distribution at an initial state and is observed over time, one will be able to directly observe the $2^{nd}$ law of thermodynamics with heat (information) flowing from hotter to colder areas. This analogy has one problem though, as the flow of information seems directed (hot to cold), however if we think of the lack of temperature as not being the negation thereof, the analogy becomes complete. We have different sections (vertices) of a medium (graph) exchanging heat or the lack thereof (information). Well it is almost complete. Another problem persist in the fact that the analogy converges, wheres that is not necessarily the case for a graph, but this minor inconsistency shall be forgiven.

A number of architectures for AGGREGATE have been proposed. In the pooling variant of GraphSAGE [**hamiltonInductiveRepresentationLearning2018**], AGGREGATE is formulated as

$$a_v^{(k)} = \text{MAX}\left(\left\{\text{ReLU}(\boldsymbol{W} \cdot \mathbf{h}_u^{(k-1)}), \forall u \in \mathcal{N}(v)\right\}\right), \tag{4}$$

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 [**kipfSemiSupervisedClassificationGraph2017**], the element-wise *mean* pooling is used instead, and the AGGREGATE and COMBINE steps are integrated as follows:

$$\mathbf{h}_v^{(k)} = \text{ReLU}\left(\boldsymbol{W} \cdot \text{MEAN}\left\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right). \tag{5}$$

As mentioned, there exist a multitude of GNN architectures which are all based on **??**.

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 $\mathbf{h}_G$:

$$\mathbf{h}_G = \text{READOUT}\left(\left\{\mathbf{h}_v^{(K)} \mid v \in G\right\}\right) \tag{6}$$

### 2.3 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 [**xuHowPowerfulAre2019a**]. Thus this criterion is replaced by the slightly weaker *Weisfeiler-Lehman graph isomorphism test* [**douglasWeisfeilerLehmanMethodGraph2011**], which is known to work well in general with a few exceptions.

**Lemma 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 4** (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 functions $f$, which operates on multisets and $\phi$ are injective.*

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

**??** and **??**, for which the proofs can be found in the **??**, are just a snippet of what can be said about the capabilities of GNNs. For those, who seek a more in depth understanding of the performance capabilities of GNNs, we refer [**xuHowPowerfulAre2019a**], which covers the topic in great detail.

## 3 Graph Theory

The preceeding sections deliver a gentle introduction into the foundation of the framework presented in this paper. However a formal definition and elaboration of graphs, as well as the mathematical ideas used remains an unsettled debt, as a full understanding demands some insight on graph theory. This section shall be dedicated to resolving said debt by giving a brief summary on the history of these objects as well as some key ideas around them.

Informed by the definition presented in [**diestelGraphTheory2017**], on which this section is based to a high degree, it can be said that graph theory is a branch of mathematics that studies the relationships between objects. It represents these relationships in terms of vertices and edges. A graph is essentially a collection of vertices and edges that together describe a network of connections where the vertices are the objects and the edges the connections amongst them. The field originated in the $18^{th}$ century as a result og the work of swiss mathematician Leonhard Euler, who in solving the famous problem of the seven bridges of Königsberg gave birth to said field. Since then, the field has expanded significantly and is now a vital part of discrete mathematics, playing a fundamental role in - for example - the *Wolfram Physics Project*. Anyone who has listened to Jonathan Gorard elaborate *Multiway Causal Graphs* knows how quickly talking about graphs can get theoretical and dense. Although some mathematical ideas are presented, the section covers only the most fundermental ideas needed for a full understanding of the framework as presented in [**diestelGraphTheory2017**; **chungSpectralGraphTheory2009**; **caiEverythingSpectralGraph2019**].

### 3.1 The Basics

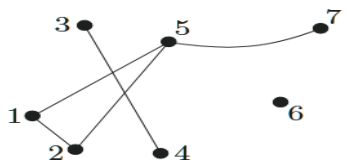



Figure 3: A graph given by $V = \{1,\dots,7\}$, $E = \{\{1,2\},\{1,5\},\{2,5\},\{3,4\},\{5,7\}\}$, source: [**diestelGraphTheory2017**]

**Definition 2** (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* (neighbours), if $\{x, y\}$ is an edge of $G$ and vice versa. Two edges $e \neq f$ are *adjacent* iff 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 $K^3 \Leftrightarrow triangle$.

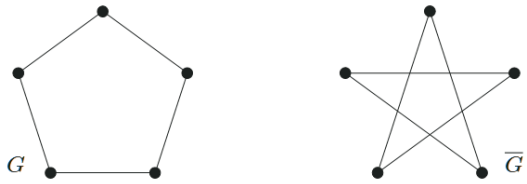



Figure 4: A path $P^6$ in $G$, source: [**diestelGraphTheory2017**]

**Definition 3** (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 a vertex $v$ is the number $|E(v)|$ of edges coinciding with $v$. This means that 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

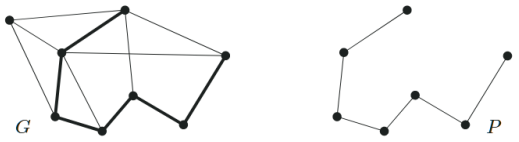
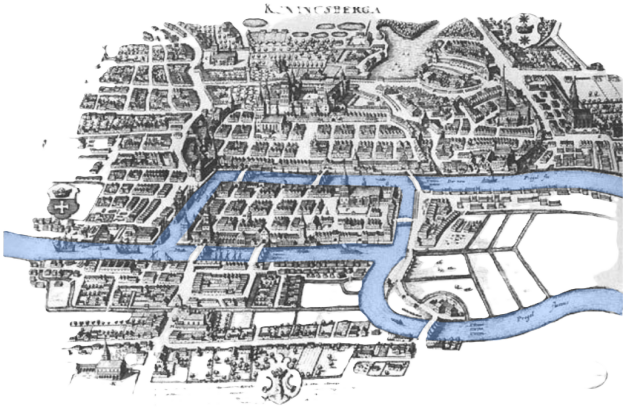



Figure 6: A path $P^6$ in $G$, source: [**diestelGraphTheory2017**]

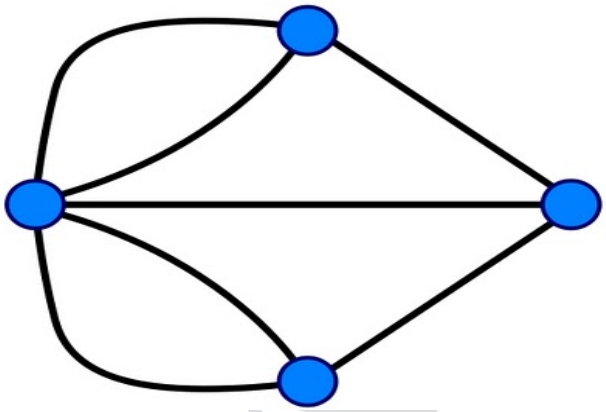
**Definition 4** (Paths & cycles)**.** *A* path *is a non-empty graph $P = (V, E)$ of the form*

$$V = \{x_0, x_1, \dots, x_k\} \qquad E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}, \tag{7}$$

*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



(a) Königsberg 18th century



(b) Graph representation

Figure 5: (a) City at the time. (b) Corresponding graph representation with bridges represented by edges, sources: aprendiendomatematicas, Spektum

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

**Proof 4.** *Pick any vertex as* $v_1$, *and assume iductively that* $v_1, \dots, v_i$ *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 6** (Trees & forests)**.** *An* acyclic *graph is called a* forest. *A connected forest is called a* tree.

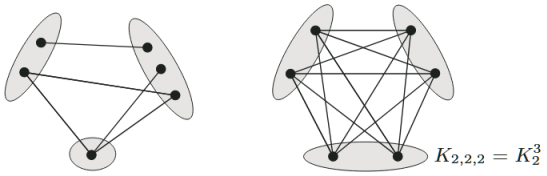



Figure 7: Two 3-partite graphs, source: [**diestelGraphTheory2017**]

**Definition 7** (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* **??**.

### 3.2 Euler Tours

As already hinted, Euler tours arose from the famous Königsberg bridge problem [**KonigsbergBridgeProblem2024**], a recreational mathematical puzzle set in the old prussian city of Königsberg (now Kaliningrad), which led to the development of the fields 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 **??**. 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 is 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 5** (Euler 1736)**.** *A* connected *graph is* Eulerian *iff every vertex has even degree.*

This just seems too beautiful a story not to have its proof layed out. In the following the number of edges shall be denoted $\|G\|$ and the number of vertices $|G|$.

**Proof 5.** *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* $F$ *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* $W$.

### 3.3 Spectral Graph Theory

In §**??**, we make use of some of the properties of the adjacency matrix of the graph in order to mitigate undesired behaviours, such as putting too much weight on vertices that have a lot of neighbours. This behaviour can be thought of as a highly connected vertexes not only knowing more about the graph due to to its connectivity, but also having its knowledge assigned more significance, which can not be assumed. In this section, we shall scratch over the surface of spectral theory in the context of graphs. This section shall be mostly based on [**chungSpectralGraphTheory1997**; **caiEverythingSpectralGraph2019**].

*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. It proofed itself particularly useful in the field of chemistry - one of the main beneficiaries of modern graph based machiine learninig architectures. Eigenvalues were associated with the stability of molecules. Also graph 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 matrix L, which acts upon the graphs vertices:

$$L(u,v) = \begin{cases} d_v & \text{if } u = v \\ -1 & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

This is equivalent to $L = D - A$, where $A$ denotes the adjacency matrix and D the diagonal degree matrix where $D_{v,v} = d_v$. The *normalized Laplacian* of G is defined as:

$$\begin{aligned} \mathcal{L}(u,v) &= \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{1}{\sqrt{d_u d_v}} & \text{if } u \text{ and } v \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \\ &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} \end{aligned} \tag{9}$$

As previously mentioned, the normalization of the laplacian regularizes the flow of information within the graph framework. This is important, because in order to capture the topology of a graph we need not necessarily assign vertices with more neighbours more significance.

Since $\mathcal{L}$ is symmetric, its eigenvalues are all real and non-negative. The *Rayleigh quotient* can be used to characterize the eigenvalues of $\mathcal{L}$. 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, then the following holds:

$$\begin{aligned} \frac{\langle g, \mathcal{L} g \rangle}{\|g\|_2^2} &= \frac{\langle g, D^{-\frac{1}{2}} L D^{-\frac{1}{2}} g \rangle}{g^T g} \\ &= \frac{\sum_{u \sim v} (f(u) - f(v))^2}{\sum_v f(v)^2 d_v} \end{aligned} \tag{10}$$

where $g = D^{-\frac{1}{2}} f$ and $\sum_{u \sim v}$ denotes the sum over all unordered pairs $\{u, v\}$ for which $u$ and $v$ are adjacent. $\langle f, g \rangle = \int_\chi f(x) g(x) dx$ denotes the standard inner product in $\mathbb{R}^n$. The largest eigenvalue of $\mathcal{L}$, also referred to as *spectral radius* of the graph is the supremum of the Rayleigh quotient.

At this point some might still be missing a visual understanding of the Eigenvalues and Eigenvectors in the context of spectral theory. For the case, we fall back to the heat diffusion analogy given in §**??**, which finds a dicretized representation in **??**. The illustration shows a graph representing a set of objects - 8 in total - with an initial temperature, such that the temperature variance in the graph is non zero. Some are hotter (red) and some colder (black). The analogy of heat shall represent a high dimensional feature representation. over time the vector representation of the state of each object is updated according to the information passing scheme. In this context, an eigenvector can be thought of as an initial state for which the temperature relation between vertices is time invariant and the eigenvalue can be thought of as the rate at which said initial state diffuses over time.

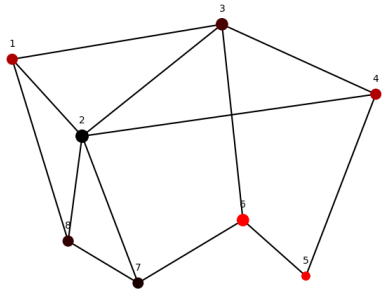



Figure 8: An example of a heat diffusion graph

## 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 dissimilar. The assumption that the data lives on some high dimensional manifold is made and the goal is then to learn a parametric representation of said manifold. However a subtle but yet very consequential assumption is often made, namely the assumption that the space in which all learnable manifolds live is euclidean. This assumption proves to be false in many cases. Also depending on the problem at hand features such at the incorporation of hierachal information might be of great importance, even so much that one would be willing to sacrifice some amount of predictive performance for the sake of interpretability. Such cases can arise when for example regulatory organs demand models high model interpretability. Hyperbolic space, for which a formal definition is given in §**??**, is an example of a space which naturally embodies such structures as **????** show. It should be said about the former illustration, that the demons in the figure are all of equal size. A practical example would be a pointcloud dataset where some portions of the dataset that require identification are subsets of others. Think for example of a pointcloud consisting of a chair and one would like to identify both the chair as well as the legs.

In such a case a low dimensional projection of the learned features can be expected to result in something not too disimilar to **??**, which has high interpretability in terms of hierachal structure. 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 the objects on which the data is assumed to live on shall be defined - 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 [**munkresTopology2000**] where the subject is eased into in a comprehensive and yet comprehensible manner.

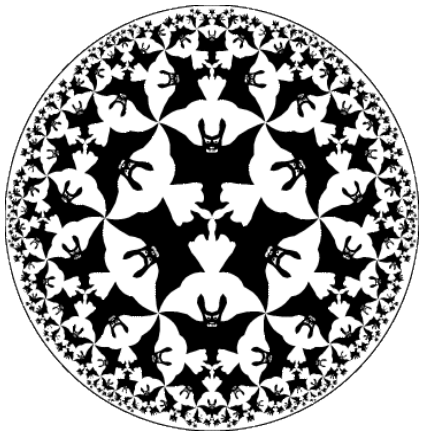


Figure 9: A rendition of Escher's Circle Limit IV pattern, source: Douglas Dunham University of California

**Definition 8** (Manifold)**.** *A manifold* $\mathcal{M}$ *is a metric space* $(X, g)$*, that satisfies the following properties [***boothbyIntroductionDifferentiableManifolds1975***]:*

- $x \in X \Rightarrow \exists\, B_\epsilon(x) =: U, n \geqslant 0 : U \cong \mathbb{R}^n$
- $X$ *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.

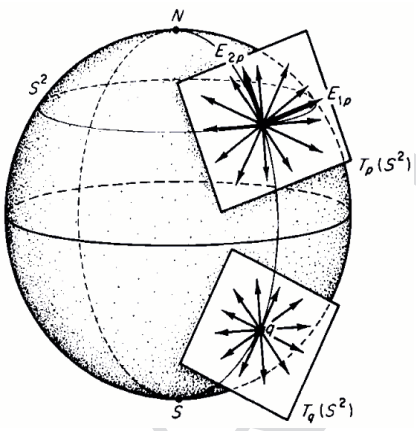



Figure 10: $S^2$ and some elements of $T(S^2)$, source: [**boothbyIntroductionDifferentiableManifolds1975**]

Let us consider for a moment the unit sphere $S^2$ in $\mathbb{R}^3$, as illustarated in **??**. The collection of all tangent vectors to points of $S^2$ are denoted $T(S^2)$. The union $\cup_{p\in S^2} T_p(S^2)$ 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 9** (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 [**petersenRiemannianGeometry2016**]. 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:

$$\begin{aligned} g_x &= \sum_{ij} g_{ij}\, dx_i \otimes dx_j, \\ g_{ij}(p) &:= \left\langle \left(\frac{\partial}{\partial x_i}\right)_p, \left(\frac{\partial}{\partial x_j}\right)_p \right\rangle_p \end{aligned} \tag{11}$$

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 general- and special relativity, the metric tensor is given by $\pm diag(-1,1,1,1) \cdot dx^2$. A direct consequence of the definitions given for example is that $\mathbb{R}^n$ is a *Riemannian manifold*. The distance between any $p_1, p_2 \in \mathcal{M}$ is given by

$$d(p_1,p_2) = \inf_x \int_0^1 \sqrt{\dot{g}_{x(t)}} dt \tag{12}$$

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 at sea level. Let us further more use spherical coordinates $(\theta, \phi)$ to identify points on the its surface. We can easily see that the distance between two meridians varies across parallels. If 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 globe 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

$$\begin{aligned} g_{\theta\theta} &= (\cos\theta)^2 + (b\sin\theta)^2 \\ g_{\phi\phi} &= (a\cos)^2\theta \\ g_{\theta\phi} &= 0, \end{aligned}$$

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 $\theta, \phi$ as $ds^2 = \left(\cos\theta^2 + (b\sin\theta)^2\right) d\theta^2 + a^2\cos^2\theta d\phi^2$.

## 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 man. '*

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 10** (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 $5^{th}$ postulate which is nowadays often substituted by* Playfairs Axiom **[harveyGeometryIlluminatedIllustrated2015]**, which states that *for any line $l$ and for any point $P$ that is not on $l$, there is exactly one line through $P$ that is parallel to $l$*. This adresses the issue of parallelity more directly

A completely different kind of geometry would be *Fano's geometry* **[harveyGeometryIlluminatedIllustrated2015]** for example, which looks very different. In his, there are three object classes consisting of *points, lines* and *ons* with the axioms:

**Definition 11** (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 defined in **[priesGeometry2019]** as a pair $G = (\Omega, I)$, where $\Omega$ represents a set of objects, with $|\Omega|$ being the number of object classes and $I$ a symmetric and reflexive relation between said object classes.

### 5.1 Neutral Geometry

By discussing Euclidean geometry in the preceeding, we skipped a step. There is a baseline geometry, on which both Euclidean-, as well as Non-Euclidean Geometry are built upon. It is the *Neutral Geometry* we are reffering to and it can be said that both Euclidean and Non-Euclidean geometries are Neutral Geometries. The *neutral geometry*, which needs not necessarily satisfy the $5^{th}$ postulate, is based on Hilberts work of the early $20^{th}$ century. In **[hilbertGrundlagenGeometrie1922]** 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 $P_1 * O * P_2$.

As different as the subsequent geometries may be, there are many comonalities shared by them and can be studied in the context of neutral geometries. We shall not go any further into this topic, but for those interested in further reading, we recommend **[harveyGeometryIlluminatedIllustrated2015]**

### 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 using the others shifted the focus to 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 **[harveyGeometryIlluminatedIllustrated2015]**.

**Theorem 6.** If the angle sum of a triangle is $\pi$, 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 6.** *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* **??** *illustrates, the perpendicular line is unique. We then construct a series of lines as the figure illustrates. The series covers ä every line through* P *except for one and it can be shown [***harveyGeometryIlluminatedIllustrated2015***],that that non of them is parallel to the base and thus the parallel line is unique.*

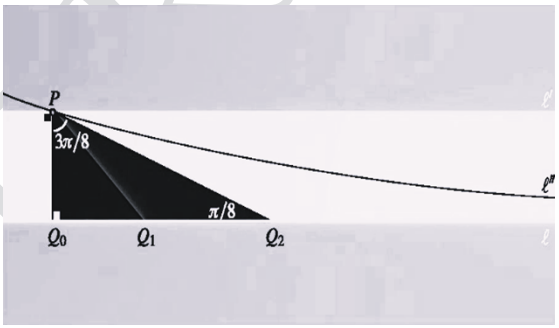



Figure 11: Series of lines through P, source: **[harveyGeometryIlluminatedIllustrated2015]**

**Definition 12** (Quadrilateral)**.** *A convex quadrilateral* □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. *[***harveyGeometryIlluminatedIllustrated2015***]*

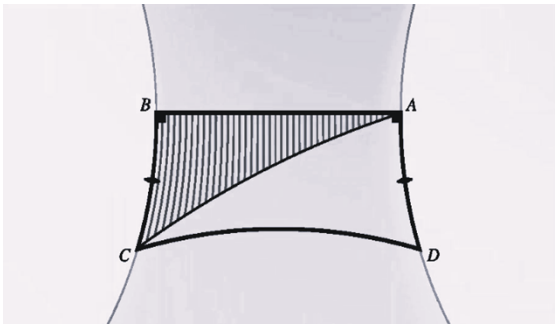



Figure 12: The parts of a Saccheri quadrilateral, source: **[harveyGeometryIlluminatedIllustrated2015]**

**Theorem 7.** Let □ABCD be a Saccheri quadrilateral with base AB. Then the summit angles $\angle C$ and $\angle D$ cannot be obtuse.*[***harveyGeometryIlluminatedIllustrated2015***]*

**Proof 7.** *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* $\pi$*. Therefore the angle sum of* $\square$ABCD *is at most* $2\pi$*. 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* $\frac{\pi}{2}$*, which concludes the proof.*

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 8.** Let $l$ be a line and P a point not on $l$. Then there are infinitely many lines through P that are parallel to $l$ *[***harveyGeometryIlluminatedIllustrated2015***].*

Delivering all the devrivations and insight, that lead to proving one of the most interesting theorems of all would span the scope of this section and is therrefore not attempted for the sake of consistency. The theorem is given in the follwoing however and we immeadeately see the ressemblance to **????**.

**Theorem 9.** *If* $l_1$ *and* $l_2$ *are parallel, then they are either asymptotic or divergent.*

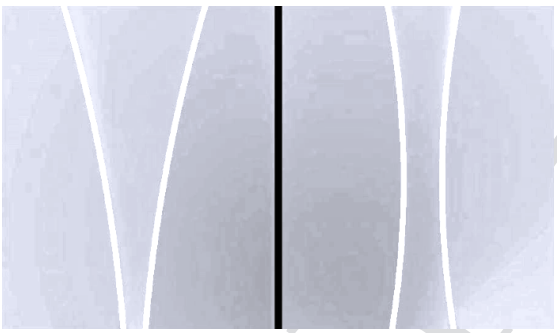


Figure 13: Series of lines through P, source: **[harveyGeometryIlluminatedIllustrated2015]**

Unfortunately for before mentioned reason the proof is not given here.

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 and has no local maxima but a unique local minimum

### 5.2.1 The Pseudosphere

Remember that the goal of this formulations is to create a way of embedding feature representations in a space of constant negative but learnable curvature. So the question arises, which kind of objects display constant negative curvature. The *pseudosphere*, ilustrated in **??** and of which the parametric form is given by **??**, is one such geometry, which is by definition Non-Euclidean. In **??**, $y(t) \sim z(t)$ is the formular for the tractrix in **??** and $X(s,t)$ the revolution thereof, as formulated in **??**

$$\begin{aligned} X(s,t) &= [z(t)\sin s, y(t), z(t)\cos s]^T \qquad 0 \leqslant s \leqslant 2\pi,\\ y(t) &= \ln\left(\frac{1+\sqrt{1-z(t)^2}}{z(t)}\right) - \sqrt{1-z(t)^2}, \qquad 0 \leqslant z \leqslant 1\\ \Rightarrow \quad X_s &= [z'(t)\cos s, 0, -z'(t)\sin s]^T\\ X_t &= [z'(t)\sin s, y'(t), z'(t)\cos s]^T \end{aligned} \tag{13}$$

### 5.2.2 The Gauss Map

Because we are now studying surfaces or manifolds §**??**, we need a way to characterize their curvature. That is where the Gauss map - a clever idea that allows us to study curvature - comes to play. In this section and inspired by the formulations found in **[harveyGeometryIlluminatedIllustrated2015]**, we derive the Gauss map.

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 first order Taylor approximation of $S_1$ around p, it can be concluded that $T_{f(p)}$ is the best linear approximation of $S_2$ around $f(p)$.

Given $X : D \to S_1$ with $(s,t) \mapsto X(s,t)$, every vector in $T_p$ can be decribed as a linear combination $aX_s + bX_t$. $f \circ X \to S_2$ is not guaranteed to be a parametrization of $S_2$, nevertheless, one can calculate the partial derivatives $(f \circ X)_s$ $(f \circ X)_t$. A parametrized line $p(\omega)$ in D is mapped to a curve $(f \circ X(p(\omega)))$ on $S_2$ with

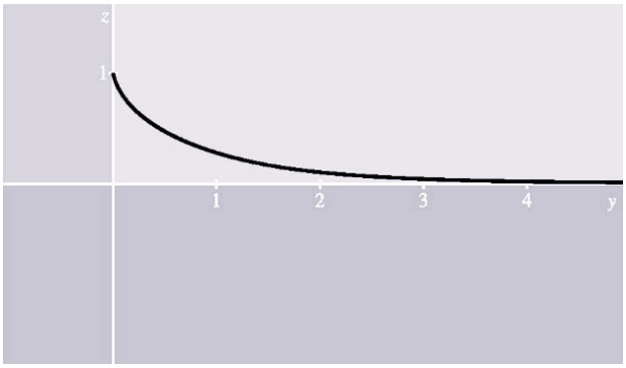
$$\begin{aligned} [f \circ X(p(\omega))]' &= (f \circ X)_s \cdot a + (f \circ X)_t \cdot b,\\ a = \frac{\partial s}{\partial \omega}, \quad b &= \frac{\partial t}{\partial \omega}\\ \Rightarrow \qquad df(aX_s + bX_t) &= a(f \circ X)_s + b(f \circ X)_t \end{aligned} \tag{14}$$

The idea behind the Gauss map is as simple as elegant. Think of the tangent plane as a surfboard balancing on a surface. Now imagine moving around the neighbourhood $\epsilon(P)$ of a pointP on the surface. The higher the magnitude of the curvature, the more the surfboard must tilt in order to keep up witht the surface and vice-versa.

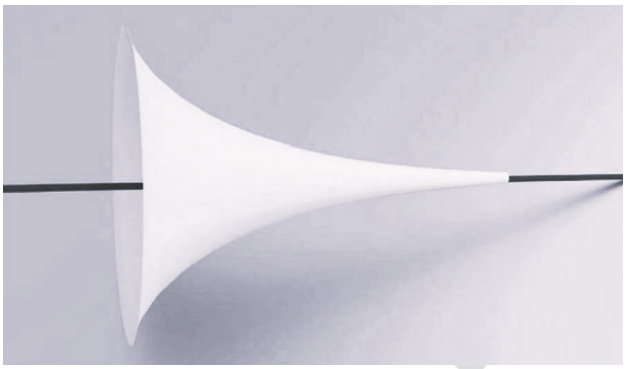
**Definition 13** (Gauss map)**.** *let* $\mathcal{S}$ *denote the unit sphere. For a point* p *on a* $S \in C^\infty$*, let* $n(P)$ *be the unit normal vector to* S *at* P*. If the trail of* $n(P)$ *is translated to the origin, then its head will be at a point* $\nu(P)$ *on* $\mathcal{S}$*. The* Gaussian map $\nu : S \to \mathcal{S}$ *is defined by* $P \mapsto \nu(P)$*.*

As $\nu$ depends only on what is hapening in $\epsilon(P)$, it can be substituded by its first order approximation $\partial\nu$. Now we compute the Gauss map and its differential for the pseudosphere **??**.

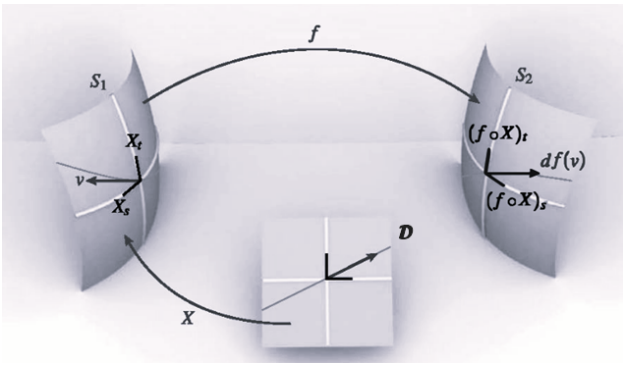
$$\begin{aligned} N &= [y'(t)z(t)\sin s, -z'(t)z(t), y'(t)\\ \Rightarrow n &= \frac{N}{\|N\|}, \qquad \|N\| = z(t)\\ \Rightarrow n &= [y'(t)\sin s, -z'(t), y'(t)\cos s]^T \end{aligned} \tag{15}$$



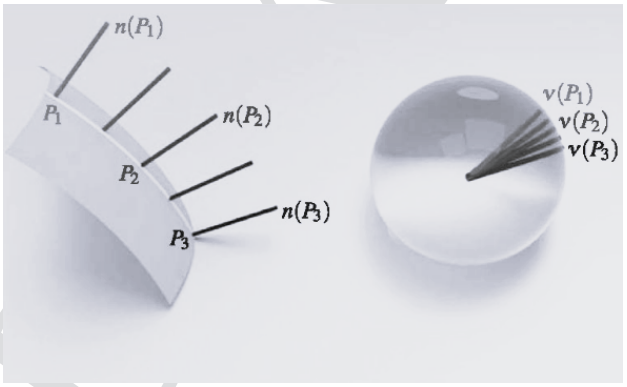

(a) tractrix



(c) pseudosphere




(b) A map between two manifolds




(d) Gauss map

Figure 14: (a) The graph of the *tractrix* (for $y > 0$). (c) Revolution of the *tractrix* around the y-axis gives results us the pseudosphere. (b) Mapping $D \to S_1 \to S_2$. (d) Gauss map

We now compute $\partial\nu$ and express it in terms of the basis elements $X_s$ and $X_t$.

$$\begin{aligned}(\nu \circ X)_s &= [y'(t)\cos s, 0, -y'(t)\sin s]^T \\ &= \frac{y'(t)}{z()}(\nu \circ X)_t \\ &= [y''(t)\sin s, -z''(t), y''(t)\cos s]^T\end{aligned}$$

$$\begin{aligned}\nu(P) &= [y'(t)\sin s, -z'(t), y'(t)\cos s]^T \\ \Rightarrow \qquad (\nu \circ X)_s &= \frac{y'(t)}{z(t)}X_s \\ \Rightarrow \qquad (\nu \circ X)_t &= \frac{z''(t)}{y'(t)}X_t \\ \Rightarrow \qquad d\nu\begin{pmatrix}a\\b\end{pmatrix} &= \begin{bmatrix}\frac{y'}{z} & 0\\ 0 & -\frac{z''}{y'}\end{bmatrix}\begin{pmatrix}a\\b\end{pmatrix}\end{aligned} \tag{16}$$

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

$$\kappa(p) = -\frac{z''(t)}{z(t)} \tag{17}$$

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

$$-\frac{z''(t)}{z(t)} \stackrel{!}{=} -1 \Rightarrow z''(t) = z(t), \tag{18}$$

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

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

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

Please refer to **[harveyGeometryIlluminatedIllustrated2015]** for more details on the derivation of **????**

### 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*

$$g_x^{\mathcal{M}} = \lambda_x^2 g^{\mathbb{E}}, \qquad \lambda_x = \frac{2}{1-\kappa\|x\|^2} \tag{19}$$

$$d_{\mathcal{M}}(x,y) = \cosh^{-1}\left(1 + 2\frac{\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right) \tag{20}$$

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 [**ganeaHyperbolicNeuralNetworks2018**].

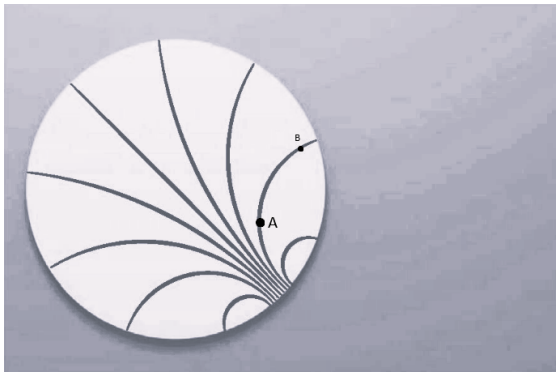



Figure 15: Some gyrolines in the Poincaré disk model. The line through A and B is a geodesic, source: [**harveyGeometryIlluminatedIllustrated2015**]

### 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 [**bachmannConstantCurvatureGraph2020**]. 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 14** (Möbius Addidtion,-scalar multiplication, distance matric, geodesics)**.**

$$x \oplus_\kappa y := \frac{(1+2\kappa\langle x,y\rangle + \kappa\|y\|^2)x + (1+\kappa\|x\|^2)y}{1+2\kappa\langle x,y\rangle + (\kappa\|x\|\cdot\|y\|)^2} \tag{21}$$

$$r \otimes_\kappa x := \frac{1}{\sqrt{\kappa}}\tanh\left(r\cdot\tanh^{-1}(\sqrt{\kappa}\|x\|)\right)\frac{x}{\|x\|} \tag{22}$$

$$d_\kappa(x,y) := \frac{2}{\sqrt{\kappa}}\tanh^{-1}(\sqrt{\kappa}\|-x\oplus_\kappa y\|) \tag{23}$$

$$\gamma_{x\to y}(t) := x \oplus_\kappa (-x\oplus_\kappa y)\otimes_\kappa t, \tag{24}$$

$$\gamma_{x\to y} : \mathbb{R}\to\mathcal{M}_\kappa^n s.t., \tag{25}$$
$$\gamma_{x\to y}(0) = x$$
$$\gamma_{x\to y}(1) = y$$

We further more define the *gyrogroup coaddition*

**Definition 15** (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* [**ungarEinsteinsVelocityAddition2007**]*.*

A probably more digestable definition is given in [**ungarGyrovectorSpaceApproach2009**]. 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 2.** 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:

$$\exp_x^\kappa(v) := x\oplus_\kappa\left(\tanh\left(\sqrt{\kappa}\frac{\lambda_x^\kappa\|v\|}{2}\right)\frac{v}{\sqrt{\kappa}\|v\|}\right) \tag{26}$$

$$\log_x^\kappa(y) := \frac{2}{\sqrt{\kappa}\lambda_x^\kappa}\tanh^{-1}(\sqrt{\kappa}\|-x\oplus_\kappa y\|)\frac{-x\oplus_\kappa y}{\|-x\oplus_\kappa y\|} \tag{27}$$

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 - of which the proves can be found in [**ganeaHyperbolicNeuralNetworks2018**] - at hand, we can obtain some understanding of the *Möbius scalar multiplication*.

**Lemma 3.** $r\otimes_\kappa x$ 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

$$r\otimes_\kappa x = \exp_{\mathbf{o}}^\kappa(r\log_{\mathbf{o}}^\kappa(x)), \qquad \forall r\in\mathbb{R}, x\in\mathcal{M}_\kappa^n \tag{28}$$

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 [**ganeaHyperbolicNeuralNetworks2018**]

**Theorem 10** (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:*

$$P_{\mathbf{o}\to x}^\kappa(v) = \log_x^\kappa(x\oplus_\kappa\exp_{\mathbf{o}}^\kappa(v)) = \frac{\lambda_{\mathbf{o}}^\kappa}{\lambda_x^\kappa}v \tag{29}$$

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 $g^\kappa$:

$$\begin{aligned}\mathfrak{st}_\kappa^d &= \{x\in\mathbb{R}^d \mid -\kappa\|x\|_2^2 < 1\}\\ g_x^k, \lambda_x^\kappa &:= \textbf{??}\end{aligned} \tag{30}$$

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

$$\begin{aligned} d_{\mathfrak{st}}^\kappa(x,y) &:= \frac{2}{\sqrt{\kappa}} \tanh^{-1}(\sqrt{\kappa}\| -x \oplus_\kappa y\|) \\ x \oplus_\kappa y &:= \frac{(1-2\kappa x^T y - \kappa\|y\|^2)x + (1+\kappa\|x\|^2)y}{1-2\kappa x^T y + \kappa^2\|x\|^2\|y\|^2} \in \mathfrak{st}_\kappa^d \\ s \otimes_\kappa x &:= \tan_\kappa(s \cdot \tan_\kappa^{-1}\|x\|)\frac{x}{\|x\|} \in \mathfrak{st}_\kappa^d, \quad s \in \mathbb{R} \end{aligned} \tag{31}$$

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

$$\begin{aligned} \exp_x^\kappa(v) &:= x \oplus_\kappa \left[\tan_\kappa\left(\sqrt{|\kappa|}\frac{\lambda_x^\kappa\|v\|}{2}\right)\frac{v}{\|v\|}\right] \\ \log_x^\kappa(y) &:= \frac{2|\kappa|^{-\frac{1}{2}}}{\lambda_x^\kappa}\tan_\kappa^{-1}\|-x\oplus_\kappa y\|\frac{-x\oplus_\kappa y}{\|-x\oplus_\kappa y\|}, \\ \tan_\kappa &:= \frac{1}{\sqrt{\kappa}}\tan(\cdot) \end{aligned} \tag{32}$$

as laid out in [**fuAdaptiveCurvatureExploration2023**].

#### 5.3.3 Embedding distortion

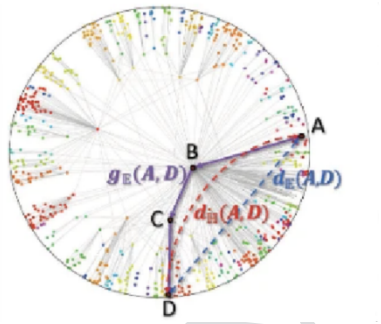



Figure 16: Different metrics in the Poincaré model, source: [**fuAdaptiveCurvatureExploration2023**]

The performance of a model is very much influenced by the feature embedding [**fuAdaptiveCurvatureExploration2023**], 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. §**??** touched on how lines are mapped to curves. **??** 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

$$g_{\mathfrak{st}}(A,D) = d_{\mathfrak{st}}(A,B) + d_{\mathfrak{st}}(B,C) + d_{\mathfrak{st}}(C,D), \tag{33}$$

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 17** (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:*

$$\mathcal{D} = \frac{1}{|V|^2}\sum_{i,j\in V}\left|\left(\frac{d_{\mathfrak{st}}^2(i,j)}{g_{\mathfrak{st}}^2(i,j)} - 1\right)\right| \tag{34}$$

### 5.4 Projective Geometry

We have talked about *Neural geometries*, *Euclidean geometries* and *Non-Euclidean Geometries*, but one could not possibly cover all Non Euclidean Geometries as that would not only be a lifetimes work, but also of no great use to this paper. There is however one more Non-Euclidean geometry that demands coverage. Although It is not of significance to the model itself, it shall play a crucial role in creating a visual interface for data visualisation. It is a geometry that we are have all become very much accustomed to, but give little appreciation. This section shall be dedicated to delivering brief insights on *projective geometry*. 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 [**priesGeometry2019**] and [**hartshorneFoundationsProjectiveGeometry2009**].

#### 5.4.1 Affine Geometry

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

**Definition 18** (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 19** (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 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 20** (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 **??**.

### 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 3.** *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 [***hartshorneFoundationsProjectiveGeometry2009***] 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 [**priesGeometry2019**] for further reading.

**Definition 21** (Homogeneous coordinates)**.** *Given a point $x \in \mathbb{R}^n$ with*

$$\tilde{x} = \begin{pmatrix} d \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \end{pmatrix} = \begin{pmatrix} \tilde{x}_0 \\ \tilde{x}_1 \\ \vdots \\ \tilde{x}_n \end{pmatrix} \leftrightarrow \begin{pmatrix} \frac{\tilde{x}_1}{\tilde{x}_0} \\ \vdots \\ \frac{\tilde{x}_n}{\tilde{x}_0} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad p_0 \neq 0 \tag{35}$$

**Definition 22** (Homogeneous hyperplane)**.** *A* homogeneous hyperplane $H \subset E^n$ *is defined by the equation*

$$\begin{aligned} H : \tilde{h}_0 \cdot 1 + \tilde{h}_1 \cdot x_1 + \cdots + \tilde{h}_n \cdot x_n = 0 \\ \rightarrow \tilde{h} = (\tilde{h}_0 \tilde{h}_1, \ldots, \tilde{h}_n)^\intercal \in \mathbb{R}^{n+1} \end{aligned} \tag{36}$$

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

**Definition 23** (Affine transformation)**.** *For any transformation given by $\alpha : E^n \to E^n, x \mapsto \alpha(x) := A \cdot x + a$, the following hold $A^\intercal A = \mathbf{I}$ 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*

$$T(a) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a_1 & 1 & 0 & 0 \\ a_2 & 0 & 1 & 0 \\ a_3 & 0 & 0 & 1 \end{bmatrix} \tag{37}$$

*Rotation around arbitrary axis $r$:*

$$R_{(3,3)}(\phi, r) = r \cdot r^\intercal + \cos\phi \cdot (\mathbf{I} - r \cdot r^\intercal) + \sin\phi \cdot \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \tag{38}$$

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

$$\begin{aligned} \alpha : E^3 \to E^3 : x \mapsto \alpha(x) := \mathbf{s} \cdot x \\ \Rightarrow \begin{bmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{39}$$

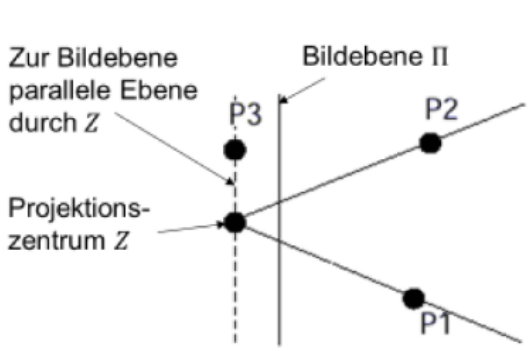
One of, if not the greatest beneficiary of affine geometry is the video game sector who rely on it for computations in the virtual 3-dimensional world as well as its projection onto the main human interface - the screen. We shall now look at the so called *central projection*.

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

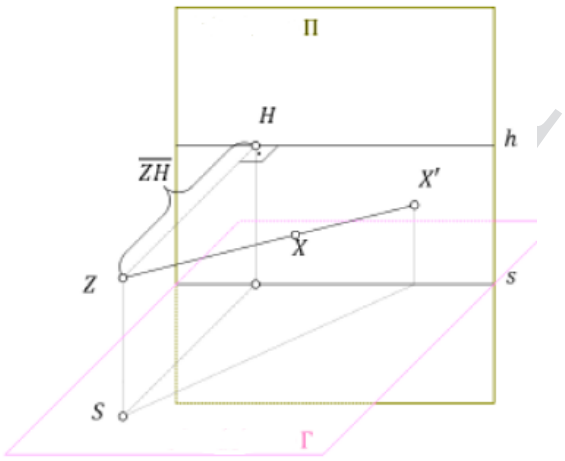
The projection line through $X$ and $Z$ in homogeneous parametric form is given by

$$\tilde{y} = t_0 \tilde{z} + t_1 \tilde{x}, \qquad (t_0, t_1) \in \mathbb{R}^2 \setminus (0, 0), \tag{40}$$

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




(a) birds-eye view




(b) 3D view

Figure 17: 2 different illustrations of the central projections, source: [**priesGeometry2019**]

## 6 Game Theory

As shown in §**??**, 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, referred to as *agents*. In this section the game theoretic concepts, as well as the learning framework adopted in the framework are detaily layed out. A rigor definition of the optimal solution as well as conditions and proves of the existence thereof are presented as well. We shall introduce the *Nash* Q-*Learning*[**huNashQLearningGeneralSum2003**] approach as an extention of the Q-*Learning* [**watkinsQlearning1992**] algorithm to a non-cooperative multi-agent context.

Game Theory is a field of mathematics arose in the attempt to study so called games that appear in various areas of every day llife, such as politics, economy, gaming as well as 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 [**ranjanReinforcementLearningDynamic2008**], robotics [**walkerApplicationReinforcementLearning2000**], as well as computer vision [**banderaResidualQLearningApplied1999**].

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*[**noordmanEvolvingNoveltyStrategies2019**; **glynatsiPropertiesWinningIterated2024**] - 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 state, 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, board games 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 and sophisticated 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.

Figure 18: Simple two player game, source: youtube.com

The idea of a Nash Equilibrium shall be explained on the example of the game *Risk - global domination*. In the game players that are distinguished by colors aim to conquer all the oponents territories. Those familiar with the game will know that often times alliances have to be made in order to win. using the situation in **??** for example, it would seem logical for blue and yellow to form an alliance, as together they have less borders to protect and without each other have very weak positions. In doing so they would not only be following their own interest, but that of them as a union. Although such cooperations might seem very obvious at times, in reality they are often difficult to achieve, as the current migration debates in the european union display. It has to be said that the algorithm for achieving such an equilibrium assumes that all participants are rational. A far fetched statement would be to say that this assumption can be used as an indirect proof of the senselessness of some of the few elected to govern the many, but that shall be left for smarter people to decide.

**Definition 24** (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 |

Figure 19: Simple two player game, source: [**fudenbergGameTheory1991**]

In the following we shall use the game illustrated in **??** with players, each of them having 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* [**huNashQLearningGeneralSum2003**]. In a stochastic game, agents choose actions simultaneously. The state space and action space are assumed to be *discrete*.

**Definition 25** (Stochastic Games)**.** *An n-player stochastic game* $\Gamma$ *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

$$\sum_{s' \in S} p(s'|s, a^1, \dots, a^n) = 1 \tag{41}$$

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 $\pi^i$ 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[**fudenbergGameTheory1991**] is a profile of

strategies such that each player's strategy is an optimal response to the other players' strategies.

**Definition 26** (Nash Equilibrium)**.** *In a stochastic game* $\Gamma$*, a Nash equilibrium is a tuple of strategies* $(\pi^1_*, \ldots, \pi^n_*)$*, such that*

$$v^i(s, \pi^1_*, \ldots, \pi^n_*) \geqslant v^i(s, \pi^1_*, \ldots, \pi^{i-1}_*, \pi^i, \pi^{i+1}_*, \ldots) \quad \forall \pi^i \in \Pi^i, \tag{42}$$

*where* $\Pi^i$ *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. **[finkEquilibriumStochastic$n$person1964]** shows that an equilibrium always consists of stationary strategies.

**Theorem 11.** *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 **[finkEquilibriumStochastic$n$person1964]**

### 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

$$\mathrm{Prob}[s_{i+1}|s_i, a_i] = P_{s_i \to s_{i+1}}[a_i] \tag{43}$$

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 $\pi^*$. The state value $s$ is given by

$$V^\pi \equiv \bar{u}_s(\pi(s)) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[\pi(s)] V^\pi(s_{i+1}) \tag{44}$$

As laid out in **[watkinsQlearning1992]**, there is at least one optimal stationary strategy $\pi^*$, such that

$$V^*(s_i) \equiv V^{\pi^*}(s_i) = \max_a \left\{ \bar{u}_{s_i}(a) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[a] V^{\pi^*}(s_{i+1}) \right\} \tag{45}$$

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

$$Q^\pi(s_i, a) = \bar{u}_{s_i}(a) + \beta \sum_{s_{i+1}} P_{s_i \to s_{i+1}}[\pi(s_i)] V^\pi(s_{i+1}) \tag{46}$$

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 $i^{\text{th}}$ 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 $\beta_i$, 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 12** (Convergence)**.** *Given bounded rewards* $u_i \leqslant U$*, learning rates* $0 \leqslant \beta_i < 1$ *and*

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

*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* $\beta_i$ *lies between the harmonic and geometric series.*

The proof of the convergence is given in the **??**.

### 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 **[huNashQLearningGeneralSum2003]**.

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 27** (Multi-agent Q-function)**.**

$$Q^i_*(s, a^1, \dots, a^n) = u^i(s, a^1, \dots, a^n) + \beta \sum_{s' \in S} p(s'|s, a^1, \dots, a^n) v^i(s', \pi^1_*, \dots, \pi^n_*), \tag{47}$$

,

*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 **[huNashQLearningGeneralSum2003]** 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 $Q^i_0(s, a^i, \dots, a^n) = 0 \; \forall s \in S, a^1 \in A^1, \dots, a^n \in A^n$. 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

$$Q^i_{t+1}(s, a^1, \dots, a^n) = (1 - \beta) Q^i_t(s, a^1, \dots, a^n) + \beta_t \left[ u^i_i + \beta NashQ^i_t(s') \right], \tag{48}$$

where

$$NashQ^i_t(s') = \pi^1(s') \cdots \pi^n(s') \cdot Q^i_t(s') \tag{49}$$

We now analyse the complexity of **??**. 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 $|A^i| = |A| \; \forall i = 1, \dots, n$,

**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 $Q^j_t(s, a^1, \dots, a^n) = 0$
**while** learning **do**
  Choose action $a^i_t$
  Observe $r^1_t, \dots, r^n_t$, $a^1_t, \dots, a^n_t$, and $s_{t+1} = s'$
  **for** $j = 1, \dots, n$ **do**
    $Q^j_{t+1}(s, a^1, \dots, a^n) \leftarrow (1 - \beta_t) Q^j_t(s, a^1, \dots, a^n) + \beta_t [r^j + \beta NashQ^i_t(s')]$
  **end for**
  $t \leftarrow t + 1$
**end while**

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 **[huNashQLearningGeneralSum2003]**

## 7 Hyperbolic Multiclass Logistic Regression

We shall generalize *multinominal regression* **[hosmerAppliedLogisticRegression2013]** to the Pointcaré ball. First lets formulate Euclidean MLR in terms of distances to margin hyperplanes as described in **[ganeaHyperbolicNeuralNetworks2018]**, which will pave the way for the generalisation.

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

$$\begin{aligned} &\forall k \in \{1, \dots, K\}, \quad p(k|x) \propto \exp(\langle a_k, x \rangle - b_k) \\ &b_k \in \mathbb{R}, x, a_k \in \mathbb{R}^n \end{aligned} \tag{50}$$

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 $\langle a, x \rangle - b = sign(\langle a, x \rangle - b) \|a\| d(x, H_{a,b})$, **??** can be reformulated as

$$\begin{aligned} p(k|x) &\propto \exp(sign(\langle a_k, x \rangle - b) \|a_k\| d(x, H_{a_k, b_k})) \\ &= \exp(sign(\langle -p_k + x, a_k \rangle) \|a_k\| d(x, \tilde{H}_{a_k, p_k})) \end{aligned} \tag{51}$$

This can now be adapted to the hyperbolic setting by substituting the standard euclidean addition with $\oplus_\kappa$ 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 **??** with **??**

**Definition 28** (Poincaré hyperplanes)**.** *For* $p \in \mathcal{M}_\kappa^n, a \in T_p\mathcal{M}_\kappa^n \setminus \{0\}$*, let* $\{a\}^\perp := \{z \in T_p\mathcal{M}_\kappa^n : g_p^\kappa(z,a)\} = 0 = \{z \in T_p\mathcal{M}_\kappa^n : \langle z, a\rangle = 0\}$*. Then, we define* Poincaré hyperplanes*[**ganeaHyperbolicNeuralNetworks2018**] as*

$$\begin{aligned}\tilde{H}_{a,p}^\kappa &:= \{x \in \mathcal{M}_\kappa^n : \langle \log_p^\kappa(x), a\rangle_p = 0\} \\ &= \exp_p^\kappa(\{a\}^\perp) \\ &= \{x \in \mathcal{M}_\kappa^n : \langle -p \oplus_\kappa x, a\rangle = 0\}\end{aligned} \tag{52}$$

**??** 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 13** (Hyperbolic plane distance)**.**

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

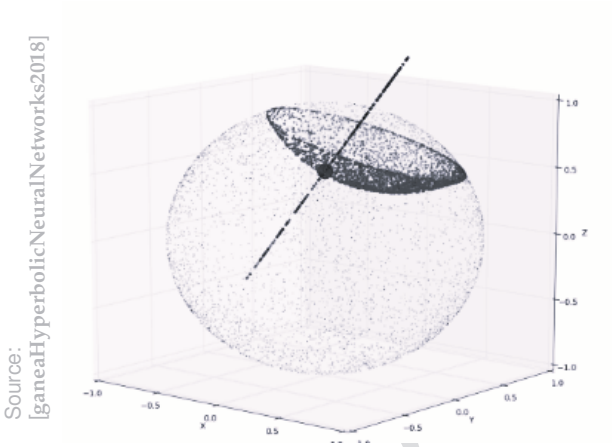



Figure 20: Illustration of a hyperplane in the *Poincaré model*

Putting together **??** and **??** we get the hyperbolic MLR formulation

$$\begin{aligned}p(k|x) \propto \exp\left(\frac{\lambda_{p_k}^\kappa \|a_k\|}{\sqrt{\kappa}} \sinh^{-1}\left(\frac{2\sqrt{\kappa}\langle -p_k \oplus_\kappa x, a_k\rangle}{(1-\kappa\|-p_k\oplus_\kappa x\|^2)\|a_k\|}\right)\right), \\ \forall x \in \mathcal{M}_\kappa^n\end{aligned} \tag{53}$$

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 **??** on $v = \log_o^\kappa(b)$. Similar to the exponential kernel for support vector machines, and as laid out in [**ganeaHyperbolicNeuralNetworks2018**], the computation of a multi layer function $f_k(x) = \varphi_k(M_k x)$ with corresponding Möbius representation $f_k^{\otimes\kappa}(x) = \varphi_k^{\otimes\kappa}(M_k \otimes_\kappa x)$ can be computed in Euclidean space by re-writting their composition

$$f_k^{\otimes\kappa} \circ \cdots \circ f_1^{\otimes\kappa} = \exp_o^\kappa \circ f_k \circ \cdots \circ f_1 \circ \log_o^\kappa \tag{54}$$

# 8 POINT-GNN

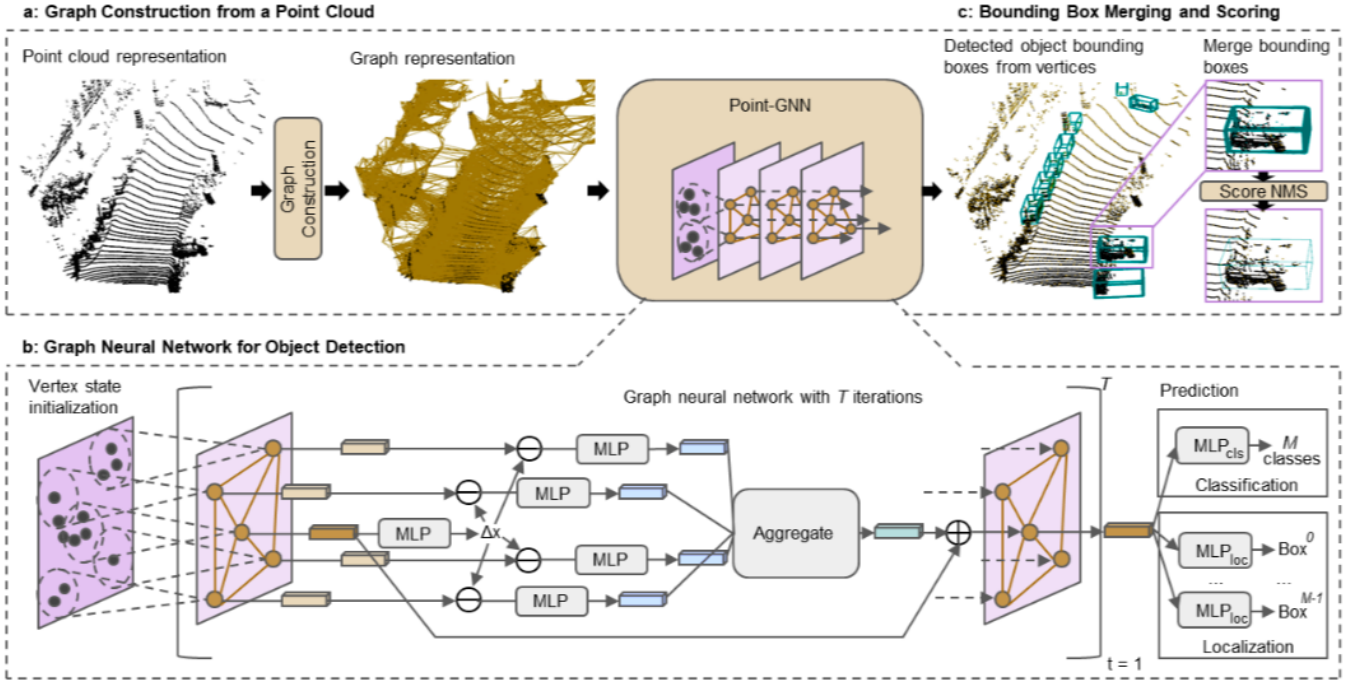
It is almost impossible to fail in imagining use cases in which an understanding of the 3 dimensional surrounding is vital. From autonomous driving to object detection in industrial automation to more foreign field that inherently deal with graph-like structures, such as molecular biology. The data we will be working with [**Geiger2012CVPR**], 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 [**fuAdaptiveCurvatureExploration2023**; **shiPointGNNGraphNeural2020**]. These representations differ from common ones in the sence that the data is embedded in hyperbolic space, as elaborated in §**??** and §**??**. We expect such an architecture to yield at least equal performance to that presented in [**shiPointGNNGraphNeural2020**], 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 [**charlesPointNetDeepLearning2017**], 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.

[**shiPointGNNGraphNeural2020**] 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.

## 8.1 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 §**??**. 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




Source: [shiPointGNNGraphNeural2020]

Figure 21: Point-GNN Architecture. (a) graph construction, (b) gnn for object detection, (c) bounding box merging and scoring

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 **??**, 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.

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

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.

### 8.2 GNN with Auto-Registration

As laid out in §**??**, 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:

$$\begin{aligned} v_i^{t+1} &= g^t(\rho(e_{ij}^t | (i,j) \in E), v_i^t) \\ e_{ij}^t &= f^t(v_i^t, v_j^t) \end{aligned} \tag{55}$$

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 **??**

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

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 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 $\Delta x_i^t$ 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 **??**. $\rho(.)$ is chosen to be MAX for its robustness as outlined in [**charlesPointNetDeepLearning2017**]. A single iteration in

the proposed graph network is given by:

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

where $[,]$ represents the concatenation operation. The $\mathrm{MLP}^t$ 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 $\mathrm{MLP}_{cls}$ computes the multi-class probability. Finally a localization branch $\mathrm{MLP}_{loc}$ computes the bounding box for each class.

#### 8.2.1 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 $\theta$ 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* **[huberRobustEstimationLocation1992]** 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:

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

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

$$l_{total} = \alpha l_{cls} + \beta l_{loc} + \gamma l_{reg} \tag{62}$$

where $\alpha, \beta$ and $\gamma$ are constant weights to balance each loss.

#### 8.2.2 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*[**prakashNonMaximumSuppression2021**] 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 **[shiPointGNNGraphNeural2020]** 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 **[hosangLearningNonmaximumSuppression2017]** 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:** $\mathcal{B} = \{b_1, \ldots, b_n\}$, $\mathcal{D} = \{d_1, \ldots, d_n\}$, $T_h$
$\mathcal{M} \leftarrow \{\}$, $\mathcal{Z} \leftarrow \{\}$
**while** $\mathcal{B} \neq \emptyset$ **do**
  $i \leftarrow \arg\max \mathcal{D}$
  $\mathcal{L} \leftarrow \{\}$
  **for** $b_j$ in $\mathcal{B}$ **do**
    **if** $\mathrm{iou}(b_i, b_j) > T_h$ **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**
  $m \leftarrow \mathrm{median}(\mathcal{L})$
  $o \leftarrow \mathrm{occlusion}(m)$
  $z \leftarrow (o+1)\sum_{b_k \in \mathcal{L}} \mathrm{IoU}(m, b_k) d_k$
  $\mathcal{M} \leftarrow \mathcal{M} \cup m$, $\mathcal{Z} \leftarrow \mathcal{Z} \cup z$
**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:

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

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

## 9 Adaptive Curvature Exploration GNN

In **[fuAdaptiveCurvatureExploration2023]**, 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 §**??**. 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 $\kappa$-stereographic model. The two models will be introduced as agents in a 2 player game as described in §**??**, as we shall see later

### 9.1 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 [**weissteinParallelogramLaw**].

The ideas presented in [**fuAdaptiveCurvatureExploration2023**] related to the subject, on which most of the following will be based upon, are mostly based on the work laid out in [**huNashQLearningGeneralSum2003**], which is highly recommended, not only for complete appreciation of the topic but also to make sense of undefined terminology.

**Definition 29** (ACE-Agent State $S^t_{ACE}$)**.** *The state of* ACE-Agent *of epoch* $t$ *is given by*

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

*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 30** (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* **??**. *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:*

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

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

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

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 $h^{\kappa^{t-1}, t-1}$ of epoch $t-1$ from Geo-Agent as the input into the Riemmannian manifolds with the new curvature $\kappa^t$:

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

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

**Definition 31** (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.*

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

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

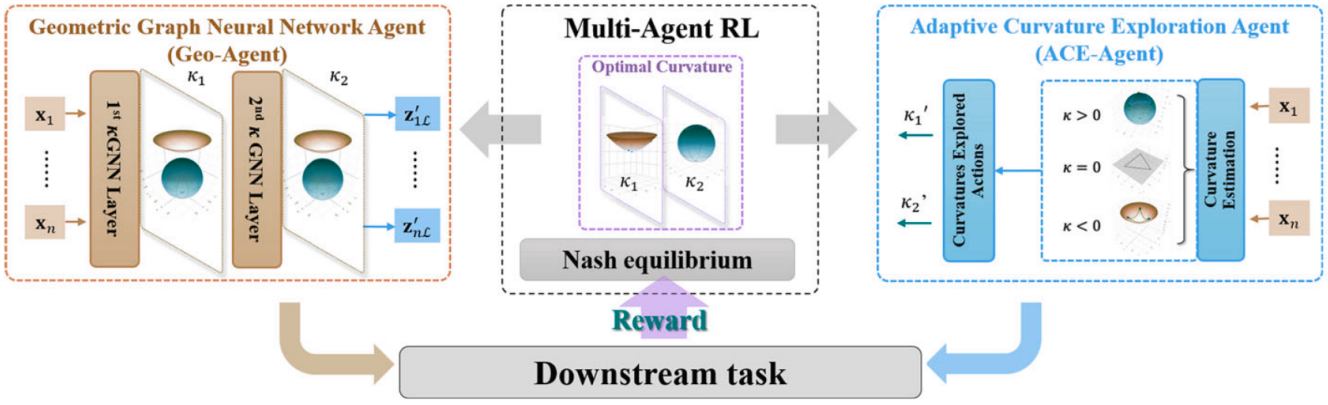



Figure 22: ACE-GEO framework

### 9.2 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 [**bachmannConstantCurvatureGraph2020**] 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 $\ell$ 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_GEO^{t-1}, k_ACE^{t-1}).
    Take an action A_t by the ϵ-greedy policy;
    // Geo-Agent
    Calculate node embeddings h_t^GEO by ??;
    Get reward R_GEO^t for Geo-Agent by Eq. (28);
    // ACE-Agent
    Calculate node embeddings h_ACE^t by ??;
    Get reward R_ACE^t for ACE-Agent by ??;
    // 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 [**fuAdaptiveCurvatureExploration2023**], $\hat{\mathbf{A}} = \tilde{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{D}^{-\frac{1}{2}}$ captures the graph conectivity, where $\tilde{\mathbf{A}} = D^{-\frac{1}{2}}(\boldsymbol{A}+\mathbf{I})D^{-\frac{1}{2}}$. 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 32** (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 weight matrix $\boldsymbol{W} \in \mathbb{R}^{d\times e}$, the κ-right-matrix-multiplication is defined row-wise as:*

$$\begin{aligned}(\mathbf{h}\otimes^{\kappa}\boldsymbol{W})_{i\bullet} &= \exp_{\mathbf{o}}^{\kappa}(\log_{\mathbf{o}}^{\kappa}(\mathbf{h})\boldsymbol{W})_{i\bullet}\\ &= \tan_{\kappa}(\alpha_i \tan_{\kappa}^{-1}(\|\mathbf{h}\|_{i\bullet}))\frac{(\mathbf{h}\boldsymbol{W})_{i\bullet}}{\|(\mathbf{h}\boldsymbol{W})_{i\bullet}\|}\end{aligned} \tag{70}$$

*where* $\alpha_i = \frac{\|(\mathbf{h}\boldsymbol{W})_{i\bullet}\|}{\|\mathbf{h}_{i\bullet}\|}$

**Definition 33** (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 [**fuAdaptiveCurvatureExploration2023**]:*

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

**Definition 34** (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 $T_{\mathbf{o}}\mathfrak{st}_\kappa^d$ is utilized for this purpose. The Geometric activation function $\sigma^{\otimes_{\kappa',\kappa}}$ with variable curvature is given by:*

$$\sigma^{\otimes_{\kappa',\kappa}}\left(\mathbf{h}_{ist}^{d,\kappa}\right) := \exp_{\mathbf{o}}^{\kappa'}\left(\sigma\log_{\mathbf{o}}^{\kappa}\left(\mathbf{h}_{ist}^{d,\kappa}\right)\right) \tag{72}$$

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

**Definition 35** (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 36** (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 37** (Geo-Agent Reward Function $R_{ACE}^t$)**.** *Similar to **??**, the reward is defined based on the downstream task compared to the last state:*

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

### 9.3 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 [**huNashQLearningGeneralSum2003**]. The Nash Q-lerning optimization fits the Bellman optimality equation as follows:

$$
\begin{aligned}
\mathrm{NashQ}_i(S') &= \pi^*_{GEO}(S')\pi^*_{GEO}(S')\pi^*_{ACE}(S')Q_i(S'),\\
Q_i(S, A^t_{GEO}, A^t_{ACE}) &\leftarrow Q_i(S, A^t_{GEO}, A^t_{ACE})\\
&\quad + \alpha(R^t_i + \beta \mathrm{NashQ}_i(S')\\
&\quad - Q_i(S, A^t_{GEO}, A^t_{ACE}))
\end{aligned} \tag{76}
$$

where $Q(\cdot)$ is the Q-function, $\alpha$ is the learning rate and $\beta$ the discount factor. The RL Algorithm terminates when the GEO-Agent and the ACE-Agent reach Nash equilibrium, and transfers the learned curvatrue $\kappa$ 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.

### 9.4 Sorting-Based NN Search

Fixed-radius near neighbour search as described in [**chenFastExactFixedradius2024**] 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. [**shiPointGNNGraphNeural2020**] 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 $p_1, \ldots, p_n \in \mathbb{R}^d$ 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 [**chenFastExactFixedradius2024**], this solution implements the Euclidean norm.

#### 9.4.1 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

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

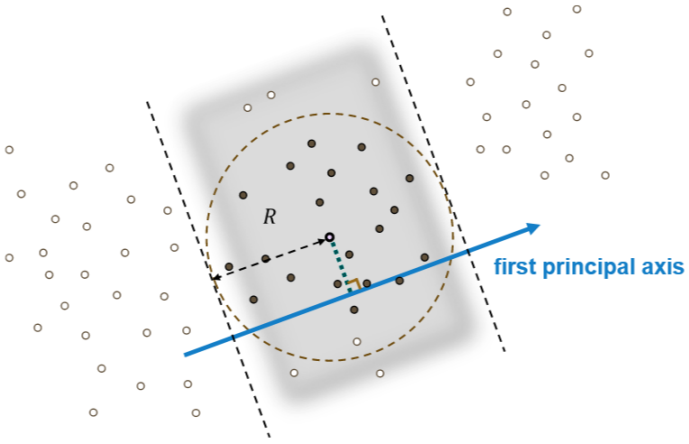
where $U \in \mathbb{R}^{n\times d}$ and $V \in \mathbb{R}^{d\times d}$ are orthononal columns and matrices and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_d) \in \mathbb{R}^{d\times d}$ is a diagonal matrix such that $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_d \geqslant 0$. The principal components are given as the columns of $V = [v_1, \ldots, v_d]$ 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 $\bar{x_i} = \frac{(x_i^T x_i)}{2}, \quad i = 1, \ldots, n$. 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 **??** for a summary.

**Algorithm 4** SNN Index

**Input:** Data matrix $P = [p_1, p_2, \ldots, p_n]^T \in \mathbb{R}^{n\times d}$
1: Compute $\mu := \mathrm{mean}(\{p_j\})$
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 $i = 1, 2, \ldots, n$
5: Sort data points $X$ such that $\alpha_1 \leqslant \alpha_2 \leqslant \ldots \leqslant \alpha_n$
6: Compute $\bar{x}_i = \frac{x_i^T x_i}{2}$ for $i = 1, 2, \ldots, n$
7: **Return:** $\mu, X, v_1, [\alpha_i], [\bar{x}_i]$




Source: [chenFastExactFixedradius2024]

Figure 23: 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.

#### 9.4.2 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 **??**.

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 $J = \{j_1 + 1, j_1 + 2, \ldots, j_2 - 1\}$ and we can determine the smallest subset by finding the largest $j_1$ and the smallest $j_2$ satisfying **??**, 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 **??** 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 **[chenFastExactFixedradius2024]**. The computationally more efficient appraoch is given in **??**.

**Algorithm 5** SNN Query

```
Input: Query vector q, user-specified radius R; output from Algorithm 1
Compute x_q := q − μ
Compute the sorting score of x_q, i.e., α_q := x_q^T v_1
Select candidate index range J so that |α_j − α_q| ⩽ R for all j ∈ J
Compute d := x̄(J) − X(J,:)^T x_q using the precomputed x̄ = [x̄_i]
Return: Points x_j with d_j ⩽ (R^2 − x_q^T x_q)/2 according to (4)
```

## 10 Data Related Implementation

This section is dedicated to presenting step by step and in a comprehensive manner, the implementation of all functionalities related to data interaction and manipulation. As already mentioned, this paper builds upon the ideas presented in **[fuAdaptiveCurvatureExploration2023; fuAdaptiveCurvatureExploration2023]**, the former of which is of more significance, as the implementations covered in this section utilizes parts of the codebase in said papers corresponding repository - mainly the data interaction and manipulation part - as a basis. §**??** deals with the general explanation of the data's origin and nature as well as presenting the implementation of an interface meant to handle data interaction as well as manipulation. §**??** covers in a brief manner other functions, which handle data preprocessing and augmentation tasks. It should be said that this section does not substitute the documentation of the codebase given in the corresponding repository Adaptive-Point-HGNN, but is rather meant to be an entry point for gaining an understanding of the codebase. Furthermore a jupyter notebook *notebook.ipynb* is provided, which gives some practical insight into the data manipulation and augmentation aspects. It is highly recommended to go through the notebook for a more hands on experience of some of the following elaborations.

### 10.1 KITTI Vision Dataset

The framework is tested and benchmarked on the KITTI Vision Dataset **[geigerKITTIVisionBenchmark2012]**. This choice allows for straightforward benchmarking, as it is the same used in **[shiPointGNNGraphNeural2020]**. 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.

**??** 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)

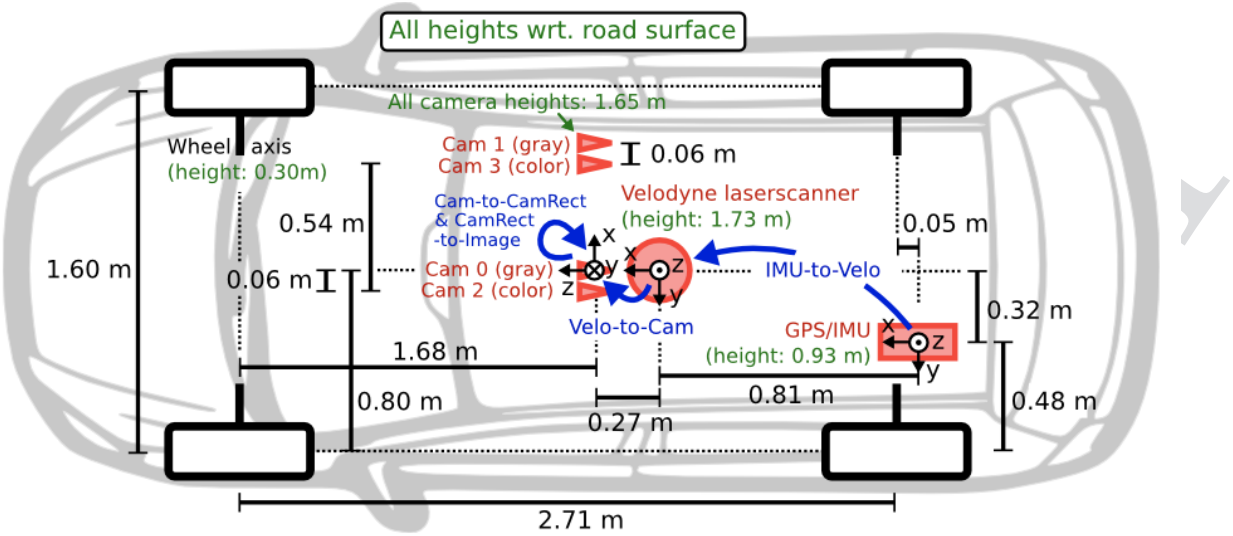



Figure 24: A fully equiped vehicle for data collection

- 2 1.4MP Color cameras: Point Grey Flea 2 (FL2-14S3C-C)
- 4 Varifocal lenses 4-8mm: Edmund Optics NT59-917

The velodyne data is used for the downstream task and the image data to enrich visual representation. 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 with 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 used for coordinate transformations as the differnt sensors have their own coordiante systems. The training set additionaly has label files, which contain information on bounding boxes that are used to identify objects in the Point cloud. The dataset is stored in the *kitti* subfolder of the *data* folder. Here you can find a folder for testing dataset and one for the training dataset, in both of them the different data types mentioned are stored in individual folders. The different data types have identical names for the same datapoint. The pointcloud is provided in binary format, the labels as text files and the images as jpg files.

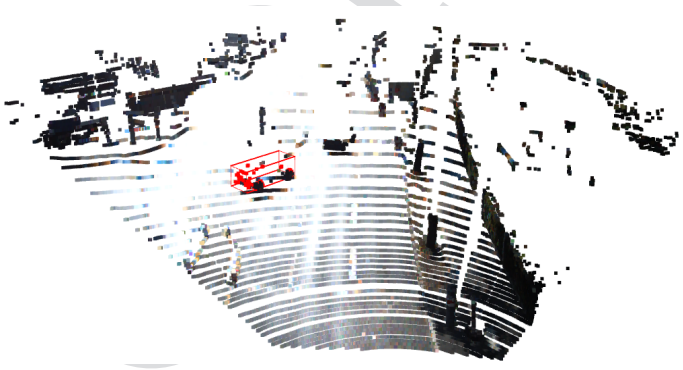


Figure 25: A scene from the KITTI Vision Dataset with a bounding box around a car

In this implementation, the dataset interaction modalities can be seen as grouped into two sets. One is meant to be used for interacting with the dataset, that is *data retrieval*, *coordinate transformations* or *merging image and point cloud data*. The other is used for more model related transformations, such as *noise introduction* and *scalling* among others. Most of the former functionalities are brought together in a class KittiDataset (**??**). Complementary to the class is the *transformations.py* file, which contains some functionalities which are used in the class, but did not necessarily make sense to define within it. Those functions are appended in **??** as well

For the sake of understanding how the transformation from velodyne to camera coordinates works, a closeup view of the implementation of **????** is provided. The former applies a transformation in 3D space, whereas the later reduces projects the datapoints onto a 2D subspace.

In **??** the function takes a label and an expend factor tuple ass input. The label is the dataset representation of bounding boxes and the expend factor is used to scale them. First, the rotation matrix is initialized using a globally defined lambda operation

$$\phi \mapsto \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{83}$$

which takes in the yaw angle and returns the corresponding rotation matrix in inhomogenous coordinates, after which the dimensions are extracted from the labels dictionary and scaled according to the expend factors. The construction of the cuboid from the dimensions is rather straightforward, but it then has to be rotated using prior mentioned matrix

Listing 10.1: Class for interacting wiht kitti 3D object detection dataset

```
class KittiDataset:
  def __init__():                # Constructor

  # ================================================================================#
  # ================================= GETTERS ======================================#
  # ================================================================================#

  def __str__():                 # String with information on dataset
  def get_filename()             # get filename for datapoint index
  def _get_file_list()           # Returns list of files in dataset
  def _verify_file_list()        # performs a sanity check
  def get_calib()                # checks that all files needed do exist
  def get_velo_points()          # get velodyne points
  def get_image()                # get image data
  def get_label()                # get label data
  def get_cam_points()           # get points in camera coordinates
  def sqdistance()               # returns the squared distance between points
  def get_cam_points_in_image()  # projects points onto image plane
  def get_cam_points_in_image_with_rgb()  # assigns points the colors given by image
  def boxes_3d_to_line_set()     # creates edges to the vertices of bbox
  def get_open3D_box()           # creates open3D bounding box
  def sel_points_in_box3d()      # identifies points within a 3D bounding box
  def sel_points_in_box2d()      # identifies points within a 2D bounding box
  def inspect_points()           # visualizes point cloud  incl. bounding boxes
  def vis_points()               # visualizes points

  # ================================================================================#
  # ================================= SETTERS ======================================#
  # ================================================================================#

  def downsample_by_voxel()      # downsamples pointcloud
  def rgb_to_cam_points()        # applies colors to points
  def vis_draw_2d_box()          # draws 2D box on image
  def velo_points_to_image()     # projects velodyne points onto image
  def vis_draw_3d_box()          # draws 3D box on image

  # ================================================================================#
  # ========================== DATA AUGMENTATION ===================================#
  # ================================================================================#

  def save_cropped_boxes()       # saves cropped boundary boxes in json format
  def load_cropped_boxes()       # loads cropped boundary boxes from json file
  def vis_cropped_boxes()        # visualizes cropped boundary boxes
  def parser_without_collision()          # NOT YET FULLY COMPREHENDED!!
  def crop_aug()                 # takes samples of bounding boxes

  # ================================================================================#
  # ========================== TRANSFORMATION.PY ===================================#
  # ================================================================================#

  def box3d_to_cam_points()      # transforms bbox from velodyne to camera
  def box3d_to_normals()         # transforms bbox into 2D camera representation
  def boxes_3d_to_corners()      # translates bbox from origin to scene position
  def cam_points_to_image()      # projects points onto image plane
  def velo_to_cam()              # transforms velodyne points to camera coordinates
                                 # using homogeneous coordinates
  def cam_to_velo()              # inverse of velo_to_cam()
```

and translated from the origin to its place in the scene, using the midpoint coordinate given by the labels. Keep in mind that the order of rotation and translation is noncommutative.

The implementation in **??** is easily understood. First the inhmogenous coordinates of the points are converted to homogenous ones, then they are transformed to camera coordinates using a calibation matrix **??**, after which the points are projected onto the plane $z = 1$ by dividing by the z-value. It might seem like this leads to a distortion, however it can be visually confirmed, that this is not the case, which can only lead to the assumption that prio distortion is introduced by the calibration matrix. This could not be confirmed, as no documentation on the dataset found touched on the topic to such detail.

Prior mentioned calibration matrix is retrieved from the calibration files, which the different matrices for transformation amongst coordinate systems. The get_calib() function is responsible for retrieving said matrices for the corresponding datapoint (point cloud). The function get_cam_points() for ex-

Listing 10.2: Transformations bounding boxes from velodyne to camera coordinates

```
def box3d_to_cam_points(label:tuple, expend_factor:list=(1.0, 1.0, 1.0)):

  yaw = label['yaw']

  R = M_ROT(yaw)                                    # get rotation matrix
  h = label['height']                               # get label dimensions
  delta_h = h*(expend_factor[0]-1)
  w = label['width']*expend_factor[1]
  l = label['length']*expend_factor[2]
  corners = np.array([
    [ l/2,  delta_h/2,  w/2],     # front up right
    [ l/2,  delta_h/2, -w/2],     # front up left
    [-l/2,  delta_h/2, -w/2],     # back up left
    [-l/2,  delta_h/2,  w/2],     # back up right
    [ l/2, -h-delta_h/2,  w/2],   # front down right
    [ l/2, -h-delta_h/2, -w/2],   # front down left
    [-l/2, -h-delta_h/2, -w/2],   # back down left
    [-l/2, -h-delta_h/2,  w/2]])  # back down right

  r_corners = corners.dot(np.transpose(R))          # rotated corners
  tx,ty,tz = label['x3d'],label['y3d'],label['z3d']  # get translation values
  cam_points_xyz = r_corners+np.array([tx, ty, tz])  # translate corner by
                                                    # center coordinates

  return Points(xyz = cam_points_xyz, attr = None)

```

ample transforms the velodyne coordinate system into camera points. Transformations are done using homogeneous coordinates, which allows for easy matrix multiplication in projective space. For example the transformation matrix $M_{v\to c}^{(1)}$ for the velodyne to camera for the first datapoint has the following homogeneous form:

$$\begin{bmatrix} -0.0\ldots & -0.9\ldots & -0.0\ldots & 0.0\ldots \\ -0.0\ldots & 0.0\ldots & -0.9\ldots & -0.0\ldots \\ 0.9\ldots & -0.0\ldots & -0.0\ldots & -0.3\ldots \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{84}$$

During the initialization process of the class, various member variables are initializes, the data paths are defined, the list of file names is internally stored for data quering, it is verified that all files indeed do exist and all Pedestrians, Cars and Cyclists are extracted from the dataset. Also statistical metrics of the dataset are computed during the this process and stored in corresponding dictionaries.

Once the velodyne data is loaded, it can be projected onto the image plane for - example by using the function get_cam_points_to_image() or get_cam_points_to_image_with rgb(), if the points are to be assigned the corresponding image colors. The retrieval of bounding boxes is rather straight forward. This is handles by boxes_3d_to_corners(), which takes a list of array representations of labels containing, 3d coordinates of bounding boxes, their sizes, as well as their yaw angles. This is a 7 degrees of freedeom representation of bounding boxes, which is also the representation we aim to learn. The function first creates a bounding box with its center at the origin, rotates it using by the yaw angle and then translates it by the center coordinate. This is repeated for all arrays in the list and the resulting bounding box list is returned. The function boxes_3d_to_line_set() is improved in comparison tho the original implementation, making use

Listing 10.3: Projects pointcloud onto image plane

```
def cam_points_to_image(points, calib):
  """Convert velodyne points to image plane points.

  @param calib: a dictionary containing calibration information.
  @return:  points on image plane: a [M, 2] float32 numpy array,
            a mask indicating points: a [N, 1] boolean numpy array.
  """

  cam_points_xyz1 = np.hstack([points.xyz, np.ones([points.xyz.shape[0],1])])    # homogeneous
coordinates
  img_points_xyz  = np.matmul(                              # transform to camera coordinates
    cam_points_xyz1, np.transpose(calib['cam_to_image']))
  img_points_xy1  = img_points_xyz/img_points_xyz[:,[2]]    # project onto camera plane
  img_points      = Points(img_points_xy1, points.attr)     # convert into Points object

  return img_points

```

prior mentioned boxes_3d_to_corners(). It takes in a list of array valued labels and optional bounding box colors and returns a list of bounding boxes, their corresponding vertices as well as the object colors. The open3d package is used for visual representation of point clouds and bounding boxes. The function get_open3D_box() for example takes a label and creates the open3d edges of the bounding box with the object class color.

### 10.2 Preprocessing and Augmentation

In the process of training a model, the problem arises, that models fail to incorporate invariances that are to be expected from a well performing model, due to the lack of variations wihtin the trraining set. For example if a model trained to identify the number 6 from a training dataset is trained only on perfectly vertical representations of the number, it is highly likely to missclassify on slightly rotated or tilted representations. This is accounted for by including a data augmentation step in the data manipulation process, where the training set is modified acording to desired variance. Using the example mentioned, the perfectly vertical images of the numbers would be randomly rotated and/or tilted, thus creating new training data. This can be seen as introducing random noise into the training set, thus making the model more stable and reducing overfitting. In the case of point clouds this is done by extracting the objects to be identified frome scenes and inserting them into the same or even other scenes with slight variations in orientation and/or size, for example by flipping them horizontally. Another method is for example shifting the bounding box, in order to imporove not only the classification precition, but the location precisiton as well.

In this section we cover all the functions **??** that are dedicated to preprocessing the data in this manner. The functions will be explained in a brief manner with reference to the documentation of the codebase, which includes more elaborate and in depth documentation of the entire code base.

Most functions are rather trivial and shall not be explained here, but some demand a brief explanation. Those shall be explained in the following. The function random_voxel_downsample() downsamples the point cloud by scaling the voxels randomly by individual factors in the range $[0, 3]$. random_rotation_all() rotates all objects in the scene by a global random angle. One could argue that random_box_rotation() and random_rotation_all() are equivalent in the context of the desired result, as can the function random_box_shift() be seen as equivalent to randomly dropping some points of the objects to be classified, thus being equivalent to random_drop(). split_background() is a helper function utilized by the succeeding functions that, as the name suggest, splits the points within bounding boxes (foreground) from the others (background) and return two points objects representing foreground and background. This is semantically rather simple, as it is a matter of creating a boolean mask that is constrained on the relationship between all points and all bounding boxes. The function dilute_background() is used to downsample the background points.

Listing 10.4: Data Augmentation Module

```
    def random_jitter()                 # adds noise to pointcloud coordinates
    def random_drop()                   # drops randomly chosen points from pointcloud
    def random_global_drop()            # container for \emph{random_drop()}
    def random_voxel_downsample()       # downsamples pointcloud by random voxel size
    def random_rotation_all()           # applies random rotation to pointcloud
    def random_flip_all()               # flips points in bounding boxes along x-axis
    def random_scale_all()              # scales bounding boxes randomly
    def random_box_rotation()           # rotates bounding boxes randomly
    def random_box_global_rotation()    # randomly rotates all bounding boxes og f object we care about
    def random_box_shift()              # shifts bounding boxes randomly
    def split_background()              # splits background points from object points
    def dilute_background()             # dilutes background points by downsampling
    def remove_background()             # removes background points
    def random_transition()             # translates bounding boxes by random vector

    AUG_METHOD_MAP                      # dictionary used to store augmentation method mappings
    def get_data_aug()                  # retrieves augmentation method based on configuration

```

## APPENDIX A
### PROOF OF ??

**Proof 8.** *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 diffrent collections of node labels at iteration $i+1$ for $G_1$ and $G_2$ as diffrent multisets get unique new labels.*

*The WL test always relabels different multisets of neigbouring nodes into diffrent 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 $\phi$ such that $\mathbf{h}_v^{(i)} = \phi(l_v^{(i)}) \forall v \in G$. It also follows from collection of GNN neighbourhood features*

$$\begin{aligned} &\{\left(\mathbf{h}_v^{(i)}, \{\mathbf{h}_u^{(i)} : u \in \mathcal{N}(v)\}\right)\} = \\ &\{\left(\phi(l_v^{(i)}), \{\phi(l_u^{(i)}) : u \in \mathcal{N}(v)\}\right)\} \end{aligned} \tag{85}$$

*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 ??

**Proof 9.** *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*

$$\mathbf{h}_v^{(k)} = \phi\left(\mathbf{h}_v^{(k-1)}, f\left(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)\right) \tag{86}$$

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

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

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

$$\mathbf{h}_v^{(k)} = \phi\left(\varphi\left(l_v^{(k-1)}\right), f\left(\{\varphi\left(l_u^{(k-1)}\right) : u \in \mathcal{N}(v)\}\right)\right) \tag{88}$$

*Since the composition of injective functions is injective, there exists some injective function $\psi$ so threaten*

$$\mathbf{h}_v^{(k)} = \psi\left(l_v^{k-1}, \{l_u^{k-1} : u \in \mathcal{N}(v)\}\right) \tag{89}$$

*Then we have*

$$\mathbf{h}_v^{(k)} = \psi \circ g^{-1}\left(l_v^{k-1}, \{l_u^{k-1} : u \in \mathcal{N}(n)\}\right) = \psi \circ g^{-1}\left(l_v^{(k)}\right) \tag{90}$$

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

## PROOF OF AXIOMS IN ??

**Proof 10.**

*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 ??

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 $\beta_i$, which shall be decribed in the following.

**Definition 38** (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* $\gamma$*, the same as in the real process.*

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

**Proof 11** (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:*

$$P^{(i)}_{s_i s_{i+1}}[a] = \sum_{k=1}^{i-1} P^{ARP}_{\langle s_i, i\rangle, \langle s_{i+1}, k\rangle}[a] \tag{91}$$

*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.*