Learning Word Representations with Projective Geometry

Patrick Baker

Supervisors: Dr. D. Inkpen & Dr. Y. Mao

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for

Master of Computer Science
Concentration: Applied Artificial Intelligence

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those concerning consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Abstract

Recent work has demonstrated the impressive efficacy of computing representations in hyperbolic space rather than in Euclidean space. This is especially true for multi-relational data and for data containing latent hierarchical structures. In this work, we seek to understand why this is the case.

We reflect on the intrinsic properties of hyperbolic geometry and then zero in on one of these as a possible explanation for the performance improvements — projection. To validate this hypothesis, we propose our projected cone model, $\mathcal{PC}$. This model is designed to capture the effects of projection while not exhibiting other distinguishing properties of hyperbolic geometry.

We define the $\mathcal{PC}$ model and determine all of the properties we need in order to conduct machine learning experiments with it. The model is defined as the stereographic projection of a cone into a unit disk. This is analogous to the construction of the Beltrami-Poincaré model of hyperbolic geometry by stereographic projection of one sheet of a two-sheet hyperboloid into the unit disk. We determine the mapping formulae between the cone and the unit disk, its Riemannian metric, and the distance formula between two points in the $\mathcal{PC}$ model. We investigate the learning capacity of our model. Finally, we generalize our model to higher dimensions so that we can perform representation learning in higher dimensions with our $\mathcal{PC}$ model. Because generalizing models into higher dimensions can be difficult, we also introduce a baseline model for comparison. This is a product space model, $\mathcal{PCP}$. It is built up from our rigourously developed, two-dimensional version of the $\mathcal{PC}$ model.

We run experiments and compare our results with those obtained by others using the Beltrami-Poincaré model. We find that our model performs almost as well as their Beltrami-Poincaré model, far outperforming representation learning in Euclidean space. We thus conclude that projection indeed is key in explaining the success which hyperbolic geometry brings to representation learning.

Acknowledgements

I would like to thank my parents, Bill and Ghyslaine Baker, for their support and encouragement at all moments of my life. I owe them a great debt of gratitude. Professor Tim Merrett, of McGill University, supervisor of my first master's degree in computer science, helped me with my application to the University of Ottawa by providing a letter of reference. I thank him for this and for his friendship and support. I have learned much from him. I thank my supervisors, Dr. Diana Inkpen and Dr. Yongyi Mao, for guiding me through this thesis. From our early discussions on the cone model, they encouraged me to continue my research, guided my explorations and readings, and steered me in the right direction throughout the journey. They provided prompt and helpful feedback on my writings. Thank you very much. Finally, I would like to thank my wife Mora, and children Amélie, Chloë, and Benjamin. Balancing work, graduate studies, and family life is no easy task. I am afraid that I was not alone in carrying the burden of completing this degree. I am especially grateful to Mora for encouraging me to continue when I felt it was all too much. I love you very much.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Representation Learning

Representation learning is a machine learning discipline concerned with determining favourable representations of entities. Well-known algorithms for learning representations include Word2Vec [55, 56], GloVe [67], and BERT [21]. One thing that most of these algorithms have in common is that they compute representations as vectors in a multi-dimensional vector space. These approaches have been highly successful in improving performance on downstream machine learning tasks. This is achieved by creating representations that position similar entities close to one another other, while separating unrelated entities. Additionally, interesting relationships such as analogies between entities are often captured. The classic example provided in connection to the Word2Vec algorithm is that the vector from *Montreal* to *Toronto* is much like that from *Montreal Canadiens* to *Toronto Maple Leafs* [56].

More recent works investigate learning representations in non-Euclidean spaces, with hyperbolic geometry receiving the most attention, [28, 33, 62]. Significant improvements over Euclidean embeddings have been found when using embeddings in hyperbolic space. This is especially the case when there are multiple latent hierarchical relationships between the entities. This is often the case in practice. Natural languages are laden with such

hierarchies. For example, between words there are syntactic relationships, dependency relationships, and various semantic relationships.

1.2 Hyperbolic Geometry

Hyperbolic geometry has a rich history. It follows from Euclid's five postulates for geometry if we make a small, but significant, change to just one of them. Euclid's fifth postulate guarantees the existence of a unique parallel line to a given line going through a given point not on this line. If we alter this to guarantee the existence of more than one such parallel line, we are led to the fascinating world of hyperbolic geometry. It was later discovered that this hyperbolic geometry is in fact the geometry of surfaces of constant negative curvature. Several models of this geometry have been discovered, allowing it to be visualized at long last. These models can be obtained by projection. For example, the Beltrami-Klein and Beltrami-Poincaré models may both be obtained by projecting one sheet of a two-sheet hyperboloid into a unit disk. Norman Wildberger's model of hyperbolic geometry is essentially entirely based on circular inversion and projective geometry [89–91]. It is clear that projective geometry underpins hyperbolic geometry.

There is in some sense a hierarchy of geometries in which projective geometry lies at the very foundation. Euclidean geometry allows for the construction of parallel and perpendicular lines. It can be constructed by straight edge and (floppy) compass. This leads to the notion of a metric, and thus the notions of distance and angle belong to the realm of Euclidean geometry. Affine geometry is more general and does not admit a notion of perpendicularity (which can be constructed with the help of a compass). It only allows for the creation of parallel lines. One can create a ruler in affine geometry, and thereby, a regular grid. But the axes of the grid won't be perpendicular, and as a consequence the usual notions of distance and angle do not apply. Projective geometry is even more general. It is the geometry that can be constructed using a straight edge only. It studies geometric properties that are invariant under projection such as the cross-ratio of four points.

Hyperbolic geometry also admits a very rich set of isometries. In Euclidean geometry, isometries are limited to translations, rotations, and reflections. In two dimensions, and in

terms of complex numbers, these correspond to affine transformations of the form $az + b$. With hyperbolic geometry, the isometries are much richer. They are captured by Möbius transformations. Möbius transformations have the form of a ratio of complex linear functions[1]. In addition to translations, rotations, and reflections, the isometries of hyperbolic geometry incorporate spherical inversion[2].

1.3 Hypothesis and Research Questions

Clearly hyperbolic geometry is very special and has many desirable properties. Do all of these remarkable properties deserve equal credit for its perceived benefits to representation learning?

> **Hypothesis**: We hypothesize that projection plays a key role in enabling the benefits of hyperbolic geometry for representation learning.

We seek to demonstrate this by proposing an alternative projective construction. The Beltrami-Poincaré model of the hyperbolic plane may be obtained by projecting a hyperboloid (itself a surface of positive curvature), into the unit disk, thus yielding an infinite "plane" of constant negative curvature. We replace the hyperboloid in this projective model with a cone, itself a surface of zero curvature. The result is our projected cone model. This model will not inherit all of the properties of hyperbolic geometry. Thus, it provides some insight into assessing the credit that projection alone deserves for the improvements to representation learning attained by using hyperbolic geometry. As well, the cone is a simpler structure than the hyperboloid, leading us to ask:

> **Research Question 1**: Can a non-Euclidean model that is *simpler* than hyperbolic geometry in terms of computational complexity be used effectively

[1] The isometries of the Beltrami-Poincaré model of hyperbolic geometry correspond to Möbius transformations of the form $\frac{az+b}{\overline{b}z+\overline{a}}$, where $|a| > |b|$. The isometries for the half-plane model are Möbius transformations of the form $\frac{az+b}{cz+d}$, where a, b, c, d are real, and $(ad - bc) > 0$.

[2] A Möbius transformations can be decomposed into a translation ($z \to z + \frac{d}{c}$), followed by spherical inversion and complex conjugation ($z \to \frac{1}{z}$), followed by an expansion and a rotation $z \to \frac{-(ad-bc)}{c^2}z$), followed by another translation ($z \to z + \frac{a}{c}$).

for representation learning?

The study of the projected cone model leads us to a few more questions.

Research Question 2: What is the learning capacity of the projected cone model, and how does it compare to the Beltrami-Poincaré model of hyperbolic geometry?

Research Question 3: What are the mathematical properties of the projected cone model that need to be understood in order to implement representation learning with it?

Research Question 4: How can the projected cone model be extended to higher dimensions?

Research Question 5: Is the projected cone model effective for representation learning?

1.4 Projected Cone Model

We will define the projected cone model via *lift* and *project* operations from the unit disk to the cone, and vice versa. We will study its learning capacity. We will discuss the geodesics on the cone and determine the distance between pairs of points in the projected cone model. We determine the Riemannian metric of the mapping from the cone into the unit disk.

Extending the model to higher dimensions is a challenge. We provide intuition for doing so, and generalize our two-dimensional formulas to n-dimensions. As a baseline, we introduce an alternative method of generalizing the projected cone model to higher dimensions using a product space. This allows us to assess the legitimacy of the n-dimensional extension of our model.

We conduct experiments with our two realizations of the projected cone model in higher dimensions, and compare our results to those obtained by others using hyperbolic geometry.

1.5 Contributions

The main contributions of this thesis are the introduction of the projected cone model, and through it, an explanation as to why hyperbolic geometry is effective for representation learning. We introduce the model and analyze its mathematical properties. We design objective functions, train the resulting models, and report on their performance. We analyze the results in order to validate our hypothesis and answer the reseach questions which have been posed.

1.6 Outline

The remainder of this thesis is structured as follows:

Chapter 2, *Background*, reviews related works dealing with representation learning and the use of hyperbolic geometry in machine learning.

Chapter 3, *Rudiments of Hyperbolic Geometry*, provides a general introduction to the mathematical foundations of hyperbolic geometry.

Chapter 4, *The Projected Cone Model*, introduces our projected cone model, and investigates its mathematical properties.

Chapter 5, *Methodology*, describes the methodology used for our experiments. This includes the data set, the objective function, an adaptation of stochastic gradient descent, use and role of negative sampling, evaluation methodology, and other training details.

Chapter 6, *Results*, reports and discusses the results of the experiments that were conducted.

Chapter 7, *Conclusions and Future Work*, summarizes our work and discusses ideas for future improvement.

Chapter 2

Background

2.1 Representation Learning

Representation learning is a key component of machine learning. The goal of machine learning is to leverage input data in order to train a model to perform a pre-defined task, or set of tasks, sufficiently well. Raw input data is transformed into a representation that can be fed into the model so that it may be optimized to meet its objective. The transformed raw data consists of features. The input data, now in the form of feature vectors, are fed into a model (manifesting suitable inductive biases, i.e., learning preferences). The model's parameters are then optimized based on a carefully selected learning objective.

The representations selected for raw data can signficantly impact performance on downstream tasks. As a consequence, the techniques of machine learning have been applied to the problem of learning high-quality representations of entities comprising raw data, such as the vocabulary of languages. Representation learning applies machine learning to analyze patterns and discover latent features in, and across, entities.

While it may often seem reasonable to handcraft a set of features using common sense and best intentions, the discipline of *deep learning* [31] has shown that computers are much more proficient at learning representations that lead to quality models that learn well and are

better able to generalize.

2.1.1 Representation learning for NLP

In the context of natural language processing (NLP), we need a representation for the words of the language(s) under study. It may seem reasonable to employ feature engineering to manually identify a feature set based on grammatical (e.g., part of speech), statistical (e.g., co-location information), or other properties (domain of discourse). We can invent many such properties to define a feature vector for each word.

In contrast, we can use machine learning to compute feature vectors that we may subsequently employ for other machine learning tasks. A typical approach is to pick a Euclidean space of some suitable, but fixed dimensionality in which each word will be embedded as a vector. Then, employing machine learning with a suitable model and learning objective, vector representations of the words may be learned. In this case, the meaning of the vector representing a word is opaque. It is just some position in a high-dimensional Euclidean space. Nevertheless, due to the learning objective, the relative positioning of these vectors may be highly advantageous to subsequent machine learning initiatives that choose to leverage such learned vector representations for words. No dimension has any specific meaning, and vectors may be placed anywhere in the Euclidean space. Yet, their placement may capture several interesting relationships between words. We may think of the high-dimensional Euclidean space as mostly empty, but containing multiple, interesting manifolds on which word vectors are placed and which carry some significance. This is analogous to the hidden state of variational auto-encoders [42]. For this reason, these are called *continuous* representations. In a recent work, Cartuyvels et al. discuss the advantages and disadvantages of discrete vs continuous representations [13]. They postulate that a hybrid approach combining discrete and continuous representations may be best. But the trend is clear, continuous representations are now dominant.

Let us briefly review some of the more influential approaches to learning continuous representations for natural language processing [7, 49, 59].

Earlier works gathered statistical information for representation learning (*latent semantic*

analysis [20], *the HAL model* [51], the COALS model [73] by Rohde et al. A key statistical measure is word co-occurence. This is captured by a very large matrix that is quadratic in vocabulary size and quite sparse. Singular value decomposition (SVD) was often used to reduce its dimensionality. Other tricks were introduced to deal with the co-occurency matrix, leading to the just cited models. Nevertheless, SVD itself is prohibitively expensive, makes it difficult to add new words, and isn't a good fit for modern deep learning methods that are used downstream as it uses a different learning regime. Many of these statistical techniques incorporate little knowledge regarding the sequential order in which words tend to appear in corpora. They are, or approximate, *bag-of-words* models [36].

Word2Vec, introduced by Mikolov et al. in 2013 [55, 56], is a seminal work on continuous vector representations for words. It does not use pre-computed corpora statistics. Two algorithms were introduced, Skip-gram and Continuous Bag of Words (CBOW). Both leverage a shallow neural network to train word embeddings. The difference is in the objective function used when training. As our approach to respresentation learning will also be based on a shallow network, let us go into a bit more detail on Word2Vec. The algorithm uses self-supervised training and can be applied to any text corpus or collection of corpora. The input is processed using a sliding window of some size, a hyperparameter. Each word in turn is treated as the *focus* word, and other words within the sliding window are called *context* words.

With CBOW, the objective is to omit the focus word and try to guess it based on the context words. The current representations of the context word vectors are summed, and the result is passed through a linear layer followed by a softmax classifier. The target label is that of the omitted focus word. For Skip-gram, the objective is the inverse—the goal is to guess a context word from the focus word. One context word is selected from the sliding window. The focus word is fed to a linear layer and a softmax classifier with the target label being that of the chosen context word. Word2Vec has proven to be very effective. It is based on the assumption that words with similar meaning are found in similar contexts. As well, words that are in some sense analogous will be found in similar contexts, hence the capture by Word2Vec of the "man is to woman is as king is to queen" analogy.

The GloVe [67] model combines earlier statistical methods with the sliding window-based,

SGD-optimized, shallow neural learning approach used by Word2Vec. A co-occurence matrix is used, and tricks are applied to it. For example, co-occurrences are weighted by the distance between words in the sliding window.

FastText [8] uses character n-grams to work at a finer subword granularity. This allows it to deal with out-of-vocabulary words, something Word2Vec and GloVe cannot do.

Despite distinguishing betweeen focus and context words, the Word2Vec model is not considered a contextual word representation because the sliding window is very narrow. Context2Vec [53] and CoVe [52] (Contextualized word representation Vectors) are two models which address this. CoVe, for instance, uses an LSTM [38] with an attention mechanism in a machine translation task to provide deeper context for the focus word. Embeddings from Language Models (ELMo) [68] is another model that provides deep contextual word representations.

After the introduction of the Transformer model by Vaswani et al. [84], word representations based on it began to appear. These leverage the encoder half of the encoder-decoder Transformer model. The most popular of these is BERT by Devlin et al. [21]. There are many variations of the BERT model such as RoBERTa [47], ALBERT [44], XLNet [93], and DistilBERT [74].

2.2 Representations with Hyperbolic Geometry

In recent years, efforts have been made to apply *non-Euclidean* geometry to machine learning, and to deep learning in particular. In 2022, W. Peng et al. published a survey of hyperbolic geometry applied to deep neural networks [66]. M. Bronstein et al. are promoting an *Erlangen Programme of Machine Learning* [12]. We will return to these more general topics, but for the moment we focus on the application of hyperbolic geometry to representation learning for words in NLP.

2.2.1 Poincaré embeddings

An influential paper on this subject is *Poincaré Embeddings for Learning Hierarchical Representations* by M. Nickel et al. [62]. We reproduce some results from this article in Figure 2.1. The improved performance of hyperbolic over Euclidean embeddings is dramatic for the use case they studied. We will review their work in some detail. But before we do, let us make a few observations regarding their results. Three types of embeddings were evaluated: Euclidean, Translational, and Poincaré. The last is an embedding in hyperbolic space. The experiment was run on the hypernymy relationship on the nouns of WordNet. The transitive closure of the hypernymy relationship was fed into the model, obfuscating whether a relation between nouns is direct or indirect. The reconstruction evaluation then consists of trying to reconstruct the hierarchy from the learned embeddings. The reported figures are the mean rank and mean average precision (MAP) of the ranking following a reconstruction effort. We will describe these evaluation measures fully in section 6.1. A low mean rank indicates successful reconstruction of the hierarchy, so lower is better. For MAP, a higher value is better, with 1 being a perfect score. We see that with only 5 dimensions, the hyperbolic embedding achieves a mean rank of 4.9, which is much better than the Euclidean result of 3542.3. In fact, even with 200 dimensions, the Euclidean mean rank was still much higher at 1157.3.

			Dimensionality					
			5	10	20	50	100	200
WordNet Reconstruction	**Euclidean**	Rank	3542.3	2286.9	1685.9	1281.7	1187.3	1157.3
		MAP	0.024	0.059	0.087	0.140	0.162	0.168
	Translational	Rank	205.9	179.4	95.3	92.8	92.7	91.0
		MAP	0.517	0.503	0.563	0.566	0.562	0.565
	Poincaré	Rank	4.9	4.02	3.84	3.98	3.9	**3.83**
		MAP	0.823	0.851	0.855	0.86	0.857	**0.87**

Figure 2.1: Highlighted cells indicate the best Euclidean and Translational embeddings as well as the Poincaré embeddings which acheive equal or better results. Bold numbers indicate absolute best results.
Figure reproduced from *Poincaré Embeddings for Learning Hierarchical Representations* by M. Nickel et al. [62].

Nickel et al. use an energy-based model [24, 46], and a shallow neural network to train all of their models. In an energy-based model, an energy function $\mathcal{E} : \mathcal{X} \times \mathcal{Y} \to \mathcal{R}$ provides a measure of an association between $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. The more compatible the pair (x, y) is, the lower the energy between them should be (think of it as a repulsive force). Thus the objective is to compute representations of the elements of $\mathcal{X}$ and $\mathcal{Y}$ that minimize the pairwise energy: $argmin_\theta \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \mathcal{E}(x, y)$, where θ represents the set of parameters to be optimized. For word embedding, Nickel et al. use distance as the energy function. The formula for distance depends on the space the embedding is performed in. For Euclidean space, it is the usual L^2 norm; in the translation model, it is $||\mathbf{x}-\mathbf{y}+\mathbf{r}||^2$; and for Poincaré, it is the Poincaré distance, see equation 3.17. Bordes et al. introduced TransE [10], a translational embedding model. Nickel et al. use it as an additional baseline, along with the Euclidean embedding. TransE was introducted for embedding multi-relational data. This is when the set of entities to be embedded display multiple relationships. For example, words may be hypernyms, synonyms, etc. Bordes et al. argue that for hierarchical relationships, translations are the natural transformations between them. That is, for a pair (x, y), their embeddings are to be optimized such that the embedding of x should be close to the embedding of y, plus some vector that depends on the relationship between them. The translational vector $\mathbf{r}$ is a learned parameter.

As mentioned, the training set used is the noun subset of WordNet [57] limited to the hypernymy relationship. WordNet is a lexical database of English words. It includes several relations between the words. The most prominent is synonymy. WordNet groups synonyms into what it calls *synsets*. These synsets are then the subject of other relations. Hypernymy is the *ISA* relation. A hyponym is more specific in meaning than its hyperyms. For example, *elephant* is a hyponym of *mammal*, while *mammal* is a hypernym of *elephant*. Training is done on the transitive closure of the hypernymy relationship. So, during training, it is not known if an input pair is directly or indirectly related.

The model is shallow. When training, Nickel et al. compute the energy function on training samples augmented with negative samples, and then compute a cross-entropy loss.

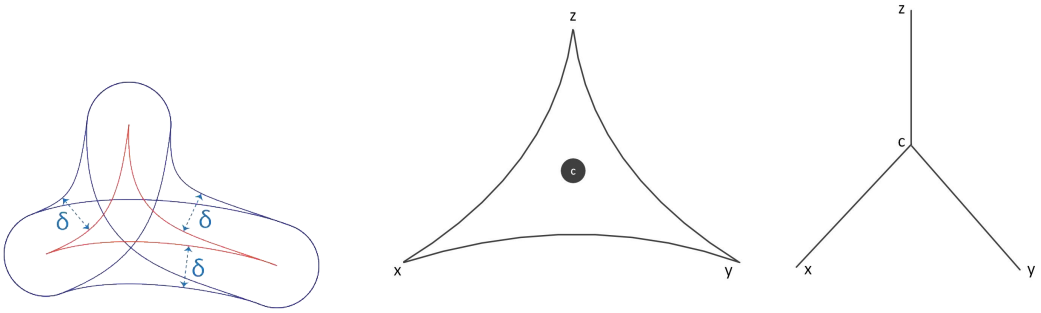
The reconstruction evaluation is not part of the training. It is meant to provide an indication of the capacity of the model. The *reconstruction* evaluation strategy considers

each hypernymy relation $(n_s, n_g) = (hyponym, hypernym)$ in the data set, $\mathcal{D}$. It checks to see if these are well positioned relative to one another. To achieve this, it ranks the distance between n_s and n_g, $d(n_s, n_g)$, among the distances between n_s and nouns to which it is not related, i.e., the set of all negative gound-truth samples involving n_s as a hyponym: $\{(n_s, n'_g)|(n_s, n'_g) \notin \mathcal{D}\}$. A ranking of 1 is ideal. A good result in the reconstruction ranking indicates that the knowledge of the hypernymy relation has been well captured by the learned embedding. We can perfectly recreate the hierarchy of the relation if the average reconstruction ranking is 1.

2.2.2 Motivation

Motivation for leveraging the setting of hyperbolic geometry focuses on higher model *capacity*. Consider vectorial embeddings in n-dimensional Euclidean space. The capacity of this model is governed by the amount of space enclosed by an origin-centered sphere. In two dimensions, with a radius of r, this increases as r^2, because the area of a disk is given by $A = 2\pi r^2$. In general, the volume of a Euclidean n-ball is proportional to r^n. In contrast, in 2-dimensional hyperbolic geometry models such as the Beltrami-Klein model or the Beltrami-Poincaré model, the volume of a disk is exponential in r. These two models are confined to the unit ball, centered at the origin. As vectors approach the boundary of the unit ball, their magnitudes increase quickly. Two vectors that are near to each other from a Euclidean perspective may actually be very far apart if near the ball's boundary. Due to this much higher capacity, far fewer dimensions are required to effectively embed vectors in hyperbolic space.

A second advantage of hyperbolic embeddings is their ability to capture *latent hierarchies* across a collection of entities. This is due to the fact that hyperbolic space has negative curvature, and thus hyperbolic triangles have a negative angular excess. That is, the sum of the internal angles of a hyperbolic triangle is less than π. (We will discuss this in more detail in chapter 3.) The extent of this *defect* is related to the intensity of the curvature of the hyperbolic space. The more curved it is, the thinner triangles will appear. In the limit, these triangles approximate connected sticks, i.e. a section of a discrete tree.



(a) A δ-slim triangle.

(b) A δ-slim triangle shrunk to a tripod, or tree. The bottom edge of the triangle on the left corresponds to the combination of the segments [xc] and [cy] on the right.

Figure 2.2

(a) *Delta thin triangle condition* (https://commons.wikimedia.org/wiki/File:Delta_thin_triangle_condition.svg), licensed under CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0/deed.en).

The mathematician M. Gromov introduced the notion of a hyperbolic metric space [32], and of a δ-slim triangles. A triangle whose sides are geodesics is δ-slim if each of its sides is contained in the δ-neighborhood of the union of the other two sides, as in the left image of figure 2.2. The fully thin triangle on the right of the figure is 0-slim. In effect, hyperbolic space is akin to a continuous version of a discrete tree. In this way, hyperbolic space can capture hierachical relationships. Extending the hyperbolic plane to hyperbolic space allows for more hierarchical relationships to be captured. Another way to view the situation is through tilings. In the Euclidean plane, there are only a few possibilities for creating regular tilings with polygons. In contrast, the hyperbolic plane is incredibly rich in the tilings it admits. One can readily see the infinite trees embedded in such tilings as in Figure 2.3. While the sizes of the triangles appear to diminish towards the edge of the disk, they are in fact all of the same hyperbolic size.

2.2.3 Riemannian SGD

When learning representations in hyperbolic space, we are confining the learned vectors to a Riemannian manifold. In order to train a model in hyperbolic space, the stochastic gradient

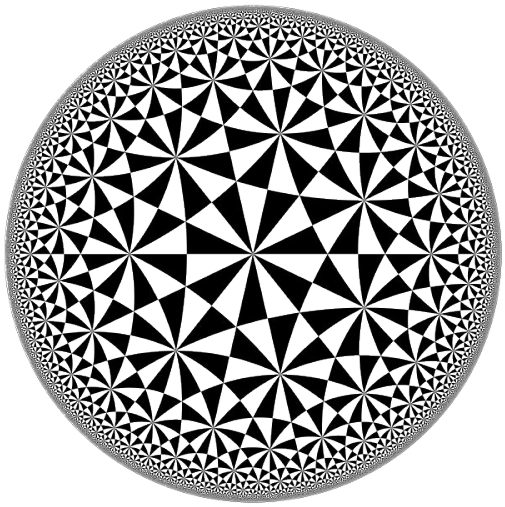


Figure 2.3: Tiling of Beltrami-Poincaré disk
732 tiling on the Poincaré Disk (https://commons.wikimedia.org/wiki/File:*732_tiling_on_the_Poincaré_Disk.svg), licensed under CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/).

descent (SGD) algorithm needs to be adapted to respect this. This is to ensure that we stay on the manifold when performing parameter updates based on the computed gradients of the loss function on a batch of training samples. Bonnabel [9] introduced Riemannian stochastic gradient descent (RSGD) for this purpose. We will discuss it more fully in chapter 5.4. Adaptive versions of RSGD optimization have been introduced [4, 40].

2.2.4 Related works

There have been several follow-up works on representation learning with hyperbolic geometry. Ganea et al. refine the approach of Nickel et al. by introducing a regularizer that encourages the placement of entailment vectors within a cone of some angular opening emanating from the head vector [28]. Balažević et al. focus on better capturing multiple hierachical relationships between entities with their approach to Poincaré embedding [3]. Takeuchi et al. investigate another approach to creating embeddings of knowledge graphs containing multiple relationships [81]. They use products of hyperbolic spaces.

2.3 Contrastive Learning

2.3.1 Overview

An important approach to representation learning is *contrastive learning*. Other approaches include adversarial training [23, 54, 71], few-shot learning, meta-learning, reinforcement learning [45], etc. [49]. This work is strongly aligned with the contrastive learning approach, and so we now provide some background on its evolution.

Contrastive learning became popular in the context of vision AI tasks such as classification, object detection, and segmentation. The SimCLR model [18] by Google Brain is a popular and successful model. Despite this, we find early roots of contrastive learning in NLP.

In 2005, Smith and Eisner introduced contrastive estimation (CE) to address the task of sequence labeling (e.g., part of speech labeling for a run of text) [79]. Earlier approaches, such as those based on conditional random fields, required labeled data for training. Smith's and Eisner's approach is *self-supervised*. They attribute the success of the method to its exploitation of *implicit negative evidence*. They showed that CE significantly outperforms Expectation-Maximization (EM), another class of algorithms that do not require labeled training data.

Whereas with Expectation-Maximization, the maximum likelihood estimate of the unknown parameters $\boldsymbol{\theta}$ is determined from

$$L(\boldsymbol{\theta}; X_1, ..., X_n) = \prod_i^n p(X_i|\boldsymbol{\theta}) = \prod_i^n \sum_y p(X_i, Y = y|\boldsymbol{\theta}).$$

(note: the y are the marginalized, latent (unobserved) variables), contrastive estimation maximizes the following to estimate $\boldsymbol{\theta}$:

$$L(\boldsymbol{\theta}; X_1, ..., X_n) = \prod_i^n p(X = x_i|X_i \in \mathcal{N}(x_i); \boldsymbol{\theta}).$$

The "neighbourhood", $\mathcal{N}(x_i)$, contains negative examples in addition to the positive example

x_i itself.

Another way to understand contrastive learning is to consider positive and a negative examples of an *anchor* sample. For an anchor sample, $\mathbf{x}$, let $\mathbf{x}^+$ be a positive sample (should be similar), and $\mathbf{x}^-$ be a negative sample (should be distinct). Contrastive learning seeks to ensure that

$$similarity(\mathcal{E}(\mathbf{x}), \mathcal{E}(\mathbf{x}^+)) >> similarity(\mathcal{E}(\mathbf{x}), \mathcal{E}(\mathbf{x}^-)).$$

where $\mathcal{E}(\mathbf{x})$ is the learned representation, or embedding, of $\mathbf{x}$. To summarize, contrastive estimation not only considers *to* where the probability mass, p, should be moved during optimization, but also *from* where the mass is taken [79].

2.3.2 Neighbourhood selection

That is the *science* of contrastive estimation. The *art* comes (1) in choosing loss functions that respect the contrastive estimation principle, and (2) in identifying appropriate samples for the neighbourhood, $\mathcal{N}(x)$, of an anchor point, x.

Regarding the latter point, data augmentation techniques are often used. These approaches can depend on the loss function and on the task. Techniques may be applied directly on input data, or between network layers in intermediate embedding spaces. Data augmentation techniques for text classification tasks are discussed in [27, 77, 85, 92]. Gao introduces SimCSE, for contrastive learning of sentence embeddings which uses only dropout as noise. Shen et al. introduce the *cutoff* technique whereby some rows and columns of the input embedding matrix of a text sequence are zeroed out [78]. A refinement is proposed in [17].

Also regarding point (2) on neighbourhood selection, if the embedding of the positive example is already close to that of the anchor sample, relative to the negative samples, then there is not much for the model to learn from this training sample. In effect, the sample is too easy to learn from. It is therefore beneficial to select good positive and negative samples for the neighbourhood of the anchor. Unfortunately, doing so can be computationally expensive.

Hard positive mining is discussed in [41], and hard negative mining is discussed in [72].

2.3.3 Loss functions

We will now review some of the more influential loss functions recently proposed for contrastive learning.

Schroff introduce the triplet loss for Facenet [75]. The goal is to perform well on face recognition and clustering tasks. A triplet consists of an anchor, $\mathbf{x}$, a positive sample for the anchor, $\mathbf{x}^+$, and a negative sample, $\mathbf{x}^-$. The embedding is performed in Euclidean space on the boundary of the unit hypersphere, $||\mathcal{E}(\mathbf{x})||_2 = 1$. To be clear, what is being embedded is an image of a face. A margin loss is used. That is, we want the positive pair to be more similar than the negative pair by at least some margin, m:

$$||f(\mathbf{x}) - f(\mathbf{x}^-)||_2^2 - ||f(\mathbf{x}) - f(\mathbf{x}^+)||_2^2 > m > 0.$$

The loss function used is thus

$$\mathcal{L}(\mathbf{x}, \mathbf{x}^+, \mathbf{x}^-) = \max(0, m + ||f(\mathbf{x}) - f(\mathbf{x}^+)||_2^2 - ||f(\mathbf{x}) - f(\mathbf{x}^-)||_2^2).$$

We see that the loss is smaller when $\mathbf{x}$ is closer to $\mathbf{x}^+$, and when it is farther from $\mathbf{x}^-$. The loss will be zero when the distance from the anchor to the negative sample is at least m more than the distance from the anchor to the positive sample.

Sohn introduces the N-pair loss which is very much like the triplet loss, but with multiple negative samples for each anchor [80]. The neighbourhood consists of 1 positive pair and N-1 negative pairs. The loss becomes

$$\mathcal{L}(\mathbf{x}, \mathbf{x}^+, \{\mathbf{x}^-\}_{i=1}^{N-1}) = \log\left(1 + \sum_{i=1}^{N-1} \exp(f(\mathbf{x})^T f(\mathbf{x}^-) - f(\mathbf{x})^T f(\mathbf{x}^+))\right).$$

We see that this loss is non-negative. It is based on cosine similarity. Note that, unlike with Schroff, the embeddings are not restricted to the unit hypersphere, but instead are

regularized to be "small". Hence the terms aren't actually cosine similarities.

2.3.4 Noise-contrastive estimation

Gutmann and Hyvärinen introduced noise-contrastive estimation (NCE) in 2010 [34], [35]. This influential work embodies a new principle in parameter estimation for *unnormalized* statistical models. Parameter estimation for probabilistic models requires that any solution be a proper probability density function. That is, it must intergrate (or sum) to 1. This usually involves computing a normalization constant (also called a partition function), which is computationally expensive. As a special case, neural approaches to word representation are classifiers which are trained by guessing a target word, w_t, from their context, c. Thus, there is a softmax layer at the end of the network which yields, for each word of the vocabulary, V, the probability of it being the target word. As the vocabulary can be quite large, this is an expensive operation. This is the softmax computation required for each target word:

$$p_\theta(w_t|c) = \frac{e^{s_\theta(w_t,c)}}{\sum_{w \in V} e^{s_\theta(w,c)}},$$

where $s_\theta()$ is the computational result of the neural network just before the softmax layer, i.e., the logits layer (we use s for "score").

More efficient ways of approximating the normalization constant have been used such as importance sampling [64] and contrastive divergence [37]. The idea of learning this normalization constant as yet another learnable parameter of the model had been considered. Unfortunately, this is not compatible with methods based on maximum likelihood estimation because the MLE estimate can be made arbitrarily large by making the constant go to zero. With NCE, a new objective function is introduced to learn this normalization constant at the same time as the other parameters, θ.

The idea is related to the contrastive estimation principle. A data set, Y, of noise is artificially generated of the same size, T, as the sample data set, X. This then can be seen as a binary classification task, so logistic regression can be applied to the task of discriminating between the sample set X, and the noise, Y.

Let us briefly review the objective function for logistic regression. Given inputs $X = (x_1, ..., x_T)$, and class labels for each $C = (c_1, ..., c_T)$, where $c_t \in \{0, 1\}$ for all t, the objective function is

$$Obj(x_t;\ \theta) = c_t \ln\left(\sigma(s_\theta(x_t))\right) + (1 - c_t) \ln\left(1 - \sigma(s_\theta(x_t))\right). \tag{2.1}$$

The first term only applies for positive samples, having class 1, while the second term applies to negative samples which have a class of 0. σ is the logistic function. s_θ is a score function, with high scores being associated with class 1. The goal is to find the θ that will maximize the objective function, given the sample X.

With NCE we have no labels, and so we generate a noise sample, y_t, for each input sample, x_t, and we associates x_t with the positive class 1, and y_t with the negative class 0. We use log odds for the score function. Odds are a ratio of the probability of a success to that of a failure:

$$Odds(u) = \frac{p(success)}{p(failure)} = \frac{p_m(u;\ \theta)}{p_n(u)}.$$

$p_m(.;\ \theta)$ is the probability model generating X for which we are estimating θ. $p_n(.)$ is some probability density function that governs the generation of the noise. It is not parameterized by θ. Assuming neither probability is exactly 0, the log odds are

$$s_\theta(u) = \ln\ p_m(u;\ \theta) - \ln\ p_n(u).$$

If we apply the logistic function σ to this, we find

$$\begin{aligned}
\sigma(s_\theta(u)) &= \frac{1}{1 + e^{-(\ln\ p_m(u;\,\theta) - \ln\ p_n(u))}} \\
&= \frac{1}{1 + \frac{e^{\ln\ p_n(u)}}{e^{\ln\ p_m(u;\,\theta)}}} \\
&= \frac{p_m(u;\ \theta)}{p_m(u;\ \theta) + p_n(u)}.
\end{aligned}$$

Under our assumption that neither probability is ever exactly 0, we see that $0 < \sigma(s_\theta(u)) < 1$, with higher values being associated with the positive class (i.e., $u \in X$). Thus for $u \in X$, our objective is to maximize $\sigma(s_\theta(u))$, and for $u \in Y$, it is to minimize this quantity, which

can be achieved by maximizing $1 - \sigma(s_\theta(u))$. Comparing all of this to the logistic objective function of equation 2.1, our final NCE objective function, to be maximized, is

$$\begin{aligned} Obj(X, Y, \theta) &= \frac{1}{2T} \sum_{t \in T} [\ln \sigma(s_\theta(x_t)) + \ln(1 - \sigma(s_\theta(y_t)))] \\ &= \frac{1}{2T} \sum_{t \in T} [\ln \sigma(\ln p_m(x_t; \theta) - \ln p_n(x_t)) + \ln(1 - \sigma(\ln p_m(y_t; \theta) - \ln p_n(y_t)))] \,. \end{aligned}$$

Each invocation of the summation takes into consideration both a positive example, x_t, and a negative one, y_t. The authors have also included a factor of $\frac{1}{2T}$ for taking an average.

To summarize, NCE estimates the parameters of a model by constrasting observed data with artificially generated noise. It seeks to learn the probability normalization constant as yet another parameter of the model. To achieve this, it uses contrastive estimation rather than maximum likelihood estimation. It uses logistic regression to perform binary classification, discriminating between observed data and the generated noise based on their log odds.

NCE in practice

That is the *science* of noise-contrastive estimation. The *art* comes in choosing a noise distribution that will be effective on the target task. A key principle is that the generated noise samples should be similar to the original samples. Otherwise, the binary classification challenge will be too easy to contribute a meaningful update to the model's parameters. As we are free to select the noise distribution, $p_n(.)$, it is good to choose one that is analytically tractable. Further recommendations include choosing noise that can be sampled easily (typically from the observed data), and using as much of it as is computationally feasable [35].

InfoNCE uses multiple noise samples per positive sample [83]. A softmax loss is used as this now becomes a multi-class classification problem.

Negative sampling, whose use is popular in word representation algorithms, can be seen as an approximation to NCE. It was used used in Word2Vec in 2013 [56]. Rather than comparing the probability of the target word to the probabilities of all other words in the vocabulary,

negative sampling constrasts the probability of the target word to just a sampling of words of the vocabulary. Mikolov et al. used sample sizes between 5 and 20. Word2Vec uses a simplified form of NCE that avoided logistic regression. The objective is

$$\log(\sigma(s_\theta(w'_f, w_c))) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim p_n(w)}[\log(1 - \sigma(s_\theta(w'_i, w_c)))].$$

Here we use w' for the output-vector representation of a word, and w for its input-vector representation (recall that Word2Vec uses both input and output vector representations for each word). The subscript f is for the *focus* word, and c is for the *context* word. For the noise distribution, $p_n(.)$, a unigram distribution, raised to the 3/4 power (and normalized), was used. This exponent was used to encourage the selection of less frequent words.

2.4 Hyperbolic Geometry in Deep Learning

There is a vast literature on hyperbolic geometry in deep learning. We will provide an introduction to hyperbolic geometry in chapter 3. We have already mentioned the survey by Peng et al. [66].

Hyperbolic geometry has been applied to several machine learning tasks, not just representation learning. In *Hyperbolic Neural Networks* [29] Ganeo et al. propose hyperbolic adaptations of neural network layers that were designed for Euclidean space. These include feed-forward and recurrent neural networks, such as gated recurrent units. Applications include classification and multinomial logistic regression.

There is work on adapting transformer-based architectures to hyperbolic space. Gulcehre et al. introduce *Hyperbolic Attention Networks* [33]. Liu et al. introduce *THG, a Transformer with Hyperbolic Geometry* [48].

Hyperbolic geometry has been applied to the NLP tasks of text classification [95] and sentence entailment classification [28]. It has been applied to recommender systems [14] and [82]. Applications in the field of biology include single-cell RNA sequence analysis [22], cell developmental processes [65], and gene expression analysis [94]. Applications to

hierachical clustering are discussed in [58] and [15]. Chami et al. introduce a hyperbolic version of graph convolution networks [16].

In *Fully Hyperbolic Neural Networks* [19], Chen et al. eschew the use of computation in tangent spaces of the hyperbolic manifold, instead proposing to work in the Lorentz model with Lorentz transformations. They show that linear transformations in tangent spaces are a relaxation of Lorentzian operations.

2.5 Geometric Deep Learning

Geometric deep learning (GDL) is an umbrella term introduced by Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković [11, 12] for approaching deep learning from a geometrical perspective. They have referred to their initiative as the *Erlangen Programme of Machine Learning*. This refers to the Erlangen programme for geometry ushered in by Felix Klein in 1872 [43]. Klein addressed the question of *what is geometry*? If it is to be the study of geometrical properties of geometric objects, then how are these properties to be identified? His answer was that the geometric properties of an object are those that are left unchanged by a certain set of transformations. For example, these may be rotations or translations. Different sets of transformations would preserve different geometric properties, and so would lead to different geometries. Additionally, not all spaces admit the same set of meaningful transformations, and hence they cannot all offer the same geometry. For instance, the distance and orientation preserving transformations, i.e., isometries, of Euclidean space are limited to rotations and translations. In contrast, a very rich set of Möbius transformations form the group of orientation-preserving isometries of the hyperbolic half-plane [60]. In essence, the transformations we are referring to are symmetries of the underlying space. They are formally studied as the actions of a group on a space. Thus, with the Erlangen programme, group theory was put at the heart of geometry.

The PointNet [69] model exemplifies several principles of geometric deep learning. The authors consider classification and segmentation of 3D objects represented as point clouds. They do not preprocess the data by meshing or creating voxels (a 3D pixel). They seek a model that will not be sensitive to the order in which the model is fed the many points of

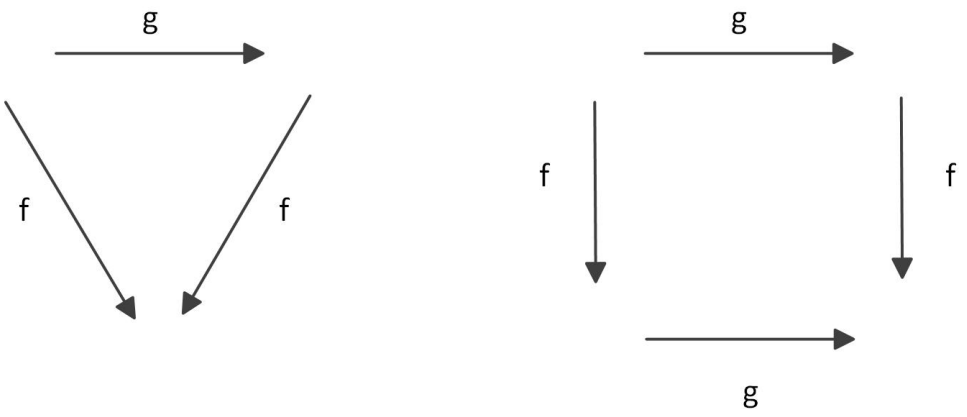


Figure 2.4: On the left, f is an invariant map under the group operation g. On the right, f is equivariant under g.

the 3D point cloud. Thus, they want to learn a symmetric function of the inputs: $f(x_1, x_2, ..., x_n) = f(x_{\pi_1}, x_{\pi_2}, ..., x_{\pi_n})$, $x_i \in \mathbb{R}^D$. This can be achieved by inserting a symmetric layer in the neural network. For example, the *maximum* and *summation* functions do the trick. The function $f = f_2 \circ \sigma \circ f_1$, is guaranteed to be symmetric if σ is symmetric. PointNet also attempts to make 3D object classification and segmentation robust to rotation and mild scaling. They achieve this by introducing a simple linear transformation, T, that is 3x3 matrix multiplication. This matrix consists of learnable parameters. The function $f = f_2 \circ T \circ f_1$ will be robust to rotations if T is a matrix that is regularized to be close to orthogonal.

Thus PointNet trains a model that is independent of the permutation order of the points, and that is robust to rotations. This corresponds to two key principles of GDL: invariance and equivariance. A function, $f(x)$, is invariant under a group operation, g, if it makes no difference whether we compute f on x or on $g(x)$. A function is equivariant if computing f on x and then applying g is the same as doing things in the reverse order. In this case, the group operation should be defined on the image of f. As examples, the area of a triangle is invariant under rotation, while the center of a triangle is equivariant under rotation. See Figure 2.4.

GDL holds that inherent symmetries in real world data constitute a geometric prior. When doing machine learning, we should restrict our search space to functions that respect the geometric priors of the data. Geometric priors often have to do with spatial symmetries such as rotation, permutation, scaling, etc. Geometric deep learning is a principled approach to doing this. Respecting the geometric prior greatly reduces the function search space and helps to address the curse of dimensionality. GDL categorizes the types of symmetries found in input data and categorizes techniques for dealing with these different symmetries. According to M. Bronstein, the purpose of GDL is "to provide a common mathematical framework to derive the most successful neural network architectures, and to give a constructive procedure to build future architectures in a principled way".

Philip Anderson, Nobel prize winner for physics (1977), has said: "It is only slightly overstating the case to say that physics is the study of symmetry." Geometric deep learning is testament to the applicability of this sentiment to the field of machine learning.

We will see that when contrasting representation learning in Euclidean and various non-Euclidean spaces, the principles of geometric deep learning, namely those of invariance and equivariance, can be an important consideration.

Chapter 3

Rudiments of Hyperbolic Geometry

3.1 Euclid's Parallel Postulate

For thousands of years, geometry was studied in the context of the flat plane. This geometry was elegantly synthesized by Euclid of Alexandria, Egypt, ca. 300 BCE in his *Elements*. Receiving immediate respect when it was published, the *Elements* continued to dominate geometrical discourse for over two millenia. With over one thousand editions since its first appearance in print in 1482, it is among the most influential works on scientific thinking [26].

Based on five *common notions*, reflecting obvious rules of reasoning such as *things which are equal to the same thing are also equal to one another*, and five *axioms*, or *postulates*, corresponding to permissible geometric constructions (essentially those governed by the use of a straight edge and (floppy) compass), Euclid derives a rich geometry in the form of proven propositions based on *definitions* (e.g., *a point is that which has no part*, and *a line is a breadthless length*) [25].

While the first axioms are quite trivial in nature (e.g., *any line can be extended indefinitely*), the fifth axiom is more subtle[1]:

> Given a line, L, and a point p not on L, there is precisely one line through p that is parallel to L. (3.1)

The term *parallel* means that the two lines never meet.

For centuries, mathematicians tried to prove that this fifth axiom was actually a consequence of the previous four, without success. Changing tack, others tried to show that negating the fifth axiom would lead to a contradiction, thereby establishing Euclidean geometry as the one true geometry. Once again, this effort failed.

An immediate consequence of the parallel postulate is that the interior angles of a triangle sum to π, as can be seen immediately from Figure 3.1.

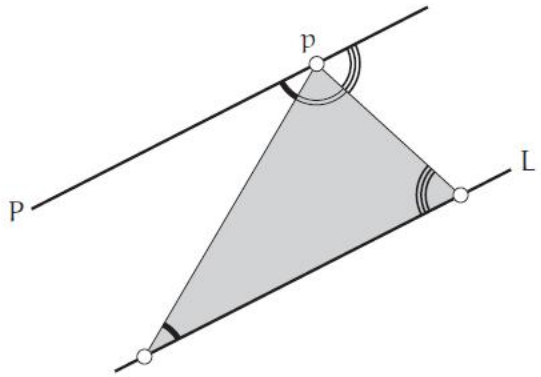


Figure 3.1: The parallel postulate implies the interior angles of a triangle sum to π.

Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

[1]Euclid's formulation was: *If a line segment intersects two straight lines forming two interior angles on the same side that are less than two right angles, then the two lines, if extended indefinitely, meet on that side on which the angles sum to less than two right angles.* The wording we offer, based on Playfair's axiom, is equivalent.

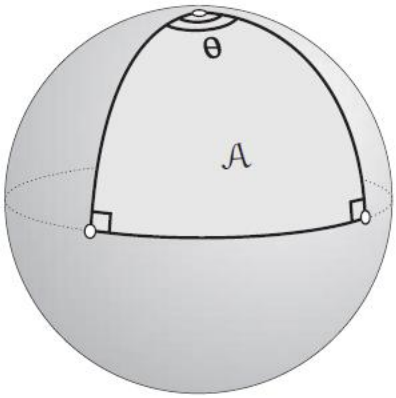


Figure 3.2: A triangle on a sphere with angles adding up to more than π.

Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

3.2 Triangles in non-Euclidean Geometry

One way to deny the fifth postulate is to say that

> Given a line, L, and a point p not on L, there is no line through p that does not meet L. (3.2)

This formulation leads to *spherical* non-Euclidean geometry. Geodesic lines on a sphere are great circles. Any pair of great circles will meet in two anti-podal points. A triangle on a sphere is determined by three such great circles. Thomas Harriot proved in 1603 that the internal angles of a triangle on a sphere exceed π. We define the *angular excess* of a triangle, $\mathcal{E}(\Delta)$, to be its (signed) deviation from π, equation 3.3. More precisely, Harriot showed that the angular excess is proportional to the area, $\mathcal{A}$, of the spherical triangle, with the proportion depending on the radius, R, of the sphere, equation 3.4.

$$\mathcal{E}(\Delta) = (\text{angle sum of triangle}) - \pi \tag{3.3}$$

$$\mathcal{E}(\Delta_{spherical}) = \frac{1}{R^2}\,\mathcal{A} \tag{3.4}$$

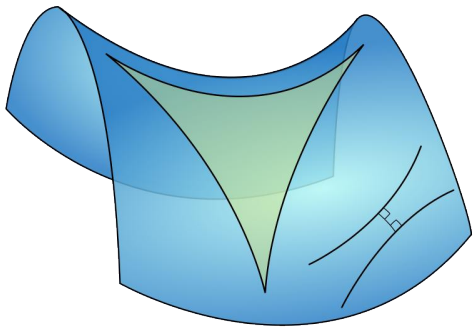


Figure 3.3: A hyperbolic triangle with angles that sum to less than π.

Attribution: Image in public domain.

An alternative way to deny the parallel postulate is as follows:

$$\text{Given a line, } L, \text{ and a point } p \text{ not on } L, \text{ there exists more than one line through } p \text{ that does not meet } L. \tag{3.5}$$

This leads to hyperbolic geometry. Before such a notion existed, Johann Heinrich Lambert, in 1766, attempted to use this formulation of the denial of the parallel postulate to find a contradiction. He did not succeed. He did, however, uncover some of the most significant results of hyperbolic geometry. In particular, he found that in this setting the angular excess of a triangle is negative. That is, the angles of a triangle sum to less that π. He also established that the angular excess is proportional to the triangle area, just as Harriot had shown for spherical triangles, equation 3.4. However, in this case the constant of proportionality is negative.

Thus, the following relation holds in the presence of Euclid's fifth postulate, with $\mathcal{K} = 0$, as well as in the presence of either denial of it, with $\mathcal{K} > 0$ in one case, and $\mathcal{K} < 0$ in the other:

$$\mathcal{E}(\Delta) = \mathcal{K}\,\mathcal{A}. \tag{3.6}$$

3.3 Curvature

In 1827, Carl Friedrich Gauss published his monumental work on differential geometry *Disquisitiones generales circa superficies curvas* [30], in which he studied the geometry of curved surfaces. He introduced the notion of *intrinsic curvature*. Curvature is a measure of how much a surface bends at a given point. By intrinsic, it is meant that the curvature may be determined without reference to the space in which the surface is itself embedded. Rather, it can be found by examining the surface itself. Another way of putting this is that the intrinsic curvature of a surface does not depend on the shape the surface takes in three-dimensional space. For example, a rectangle can be rolled into a cylinder (without stretching or bending), and this does not affect its intrinsic curvature (which remains 0 at all points). Curvature is a local property of a surface. It may differ from point to point.

For curvature, three posibilites exist — it may be postive, zero, or negative. The zero case corresponds to the flat plane of Euclidean geometry. Positive curvature is like the area at the top of a hill. The surface bends in a similar way (e.g., down for a hill) in all directions. Curvature is negative at saddle points where the surface bends in two different ways (up and down from the tangent plane at that point) depending on the direction. The absolute value of the curvature indicates the intensity of the bending.

Looking back at equation 3.6 and rearranging it, we see that we may think of the constant $\mathcal{K}$ as the angular defect per unit area of a triangle. As we saw, this is positive at the top of a hill, and negative at a saddle point. It also captures the magnitude of the effect. The magnitude of a triangle's angular defect will be larger when the surface is more severely bent. This allows us to define curvature at a point by taking the following limit for smaller and smaller triangles surrounding a point, p:

$$\mathcal{K}(p) = \lim_{\Delta_p \to p} \frac{\mathcal{E}(\Delta_p)}{\mathcal{A}(\Delta_p)}. \tag{3.7}$$

One key result of differential geometry is the Local Gauss-Bonnet theorem:

$$\mathcal{E}(\Delta) = \alpha + \beta + \gamma - \pi = \iint_{\Delta} \mathcal{K}\, \mathcal{A}. \tag{3.8}$$

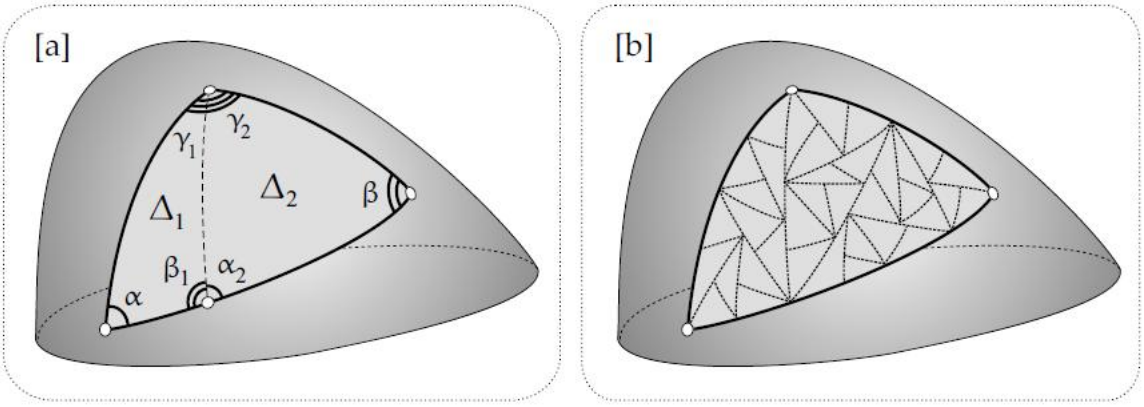


Figure 3.4: The local Gauss-Bonnet theorem follows from the *additivity* of angular excess.
Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

We note the evident fact that when curvature is constant, this simplifies to equation 3.6.

What this theorem says is that the angular excess of a triangle is equal to the total curvature it contains. We see from [3.4a] that angular exess is *additive*:

$$\mathcal{E}(\Delta_1) + \mathcal{E}(\Delta_2) = (\alpha + \beta_1 + \gamma_1 - \pi) + (\alpha_2 + \beta + \gamma_2 - \pi) = \alpha + \beta + \gamma - \pi = \mathcal{E}(\Delta). \quad (3.9)$$

We may therefore repeat the process *ad infinitum* to obtain the Local Gauss Bonnet theorem, equation 3.8.

3.4 The Riemannian Metric

We are now almost ready to get to the main point of this chapter — tying all of the above together by constructing models for hyperbolic geometry. But, before we do, it is worth taking a moment to discuss the *metric* of a surface.

A metric is a rule for the infinitesimal distance between neighbouring points. It is actually a property of a *map* of the surface. When measuring distance, a unit of measure is required. Changing the unit changes the distance measurement. The map provides this

unit of measure. Typically, though not always, we map a curved surface to a flat one, e.g., the sphere to the plane. We then seek to find the correspondance between a displacement on the curved surface and the corresponding displacement on the mapped, flat surface. We can then report distances on the curved surface in terms of the units of the flat surface. We can also the think of the map as providing a *model* of the surface. It is a different *extrinsic* shape that nevertheless locally preserves distance. If we change the map, we change the metric. However, an intrinsic property such as curvature at a point will be constant regardless of the metric chosen. Once we have a metric giving us infinitesimal distance measurements, we are then in a position to compute distances between any pair of points, and angles between lines, etc.

A mapping is constructed for a patch of a crookneck squash in Figure 3.5. First u-lines are chosen, somewhat at random, on a patch of the surface. These should cover the patch, and none of these lines should intersect. In the image of the map, these u-lines become the horizontal lines of a Cartesian grid. We likewise select v-lines. These should cross the u-lines, but they themselves do not cross each other, and they cover the patch. We can thus associate each point in the patch on the squash with a (u, v) coordinate in the flat, image grid. The metric is the relation between an infinitesimal displacement on the squash, with that in the $u\,v$-grid. The general metric formula is equation 3.10. In this formula, A is the stretch factor along the u direction (i.e., $d\hat{s}_1/du$), and B is the strecth factor along the v direction. If the u and v lines are chosen to be orthogonal, as of course they may be, then this simplifies to equation 3.11. Finally, if the amount of stretching is the same in all directions, then the map is said to be conformal, and the metric formula simplifies further to equation 3.12.

$$d\hat{s}^2 = A^2 du^2 + B^2 dv^2 + 2AB\cos\omega\, du\, dv \tag{3.10}$$

$$d\hat{s}^2 = A^2 du^2 + B^2 dv^2 \tag{3.11}$$

$$d\hat{s} = \Lambda(z) ds \tag{3.12}$$

Finally, a key result ties the curvature to the metric:

$$\mathcal{K} = -\frac{1}{AB}\left(\partial_v[\frac{\partial_v A}{B}] + \partial_u[\frac{\partial_u B}{A}]\right). \tag{3.13}$$

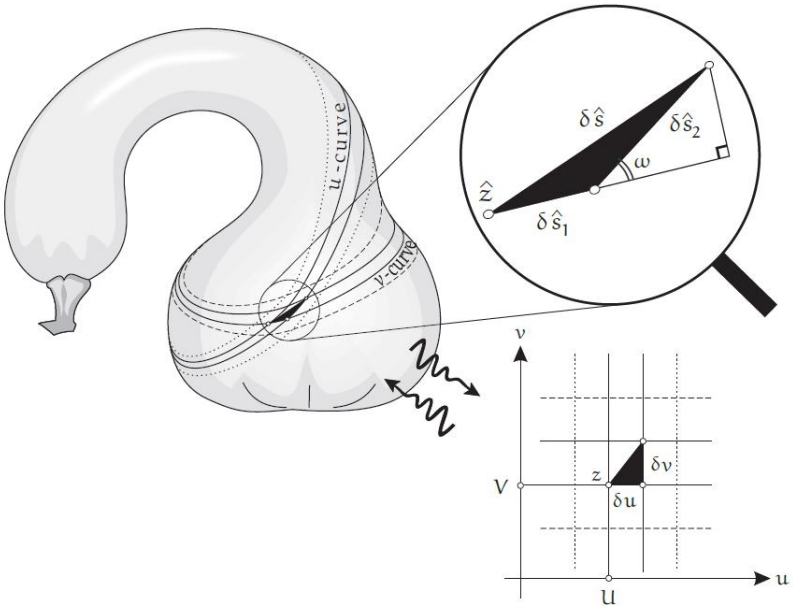


Figure 3.5: A mapping of a patch of a crook-neck squash.

Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

3.5 Models of Hyperbolic Geometry

Eugenio Beltrami was aware of the consequences we have mentioned of denying the parallel postulate in the hyperbolic manner of equation 3.5, as well as of the continued work in this direction by János Bolyai and Nikolai Ivanovich Lobachevsky. He was also aware of Gauss's 1827 work on differential geometry and the notion of curvature. In two articles published in 1868, [5] and [6], he combined these ideas to yield the insight that, as a consequence of the Local Gauss-Bonnet theorem, if he could find a surface of constant negative curvature, then triangles on this surface would necessarily obey the key result of hyperbolic geometry, equation 3.6, with $\mathcal{K}$ being negative, thus opening the way to *interpretations* of hyperbolic geometry. As well, Beltrami proved the equiconsistency of hyperbolic and Euclidean geometry, that is the consistency of one of these theories implies that of the other. Note that some of this work was presaged by Minding in 1840.

For a surface of constant (intrinsic) curvature, three posibilites exist — it is postive, zero,

Figure 3.6: A pseudosphere is a surface of constant negative curvature.

Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

or negative. The zero case corresponds to the flat plane of Euclidean geometry. A surface of constant positive curvature is the sphere, though other extrinsic shapes are possible. However, the sphere is the only closed surface having this property.

Curvature is negative at saddle points of a saddle surface where the surface bends in two different ways (up and down from the tangent plane at that point) depending on the direction. It is harder to imagine a surface with *constant* negative curvature.

The pseudosphere

The pseudosphere shown in Figure 3.6 is a surface of revolution of the *tractrix*. The tractrix is a curve of constant negative curvature. It was studied by Claude Perrault and Isaac Newton in the 1670s. The pseudosphere was studied by Christiaan Huygens in 1693. He found that, despite its infinite extent, a pseudoshere with an edge radius of R at its base has has area $4\pi R^2$, just like a sphere. Though it is not closed, it is a surface of constant negative curvature, $\mathcal{K} = -1/R^2$, where R is the radius of the sphere at the edge of the surface, as was known by Gauss's student Minding by 1839 [61].

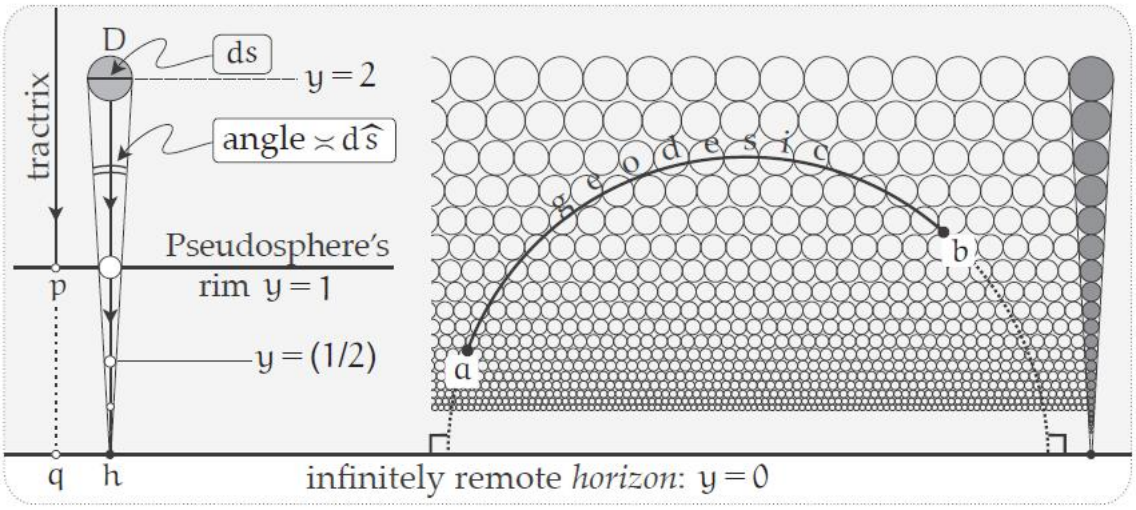


Figure 3.7: The hyperbolic diameter of the disk D is the (Euclidean) angle it subtends on the horizon. Thus, as it moves downward, its image in the map shrinks. The disks on the right are all the same size, and a geodesic line segment [a b] therefore passes through the smallest number of them.

Attribution: Image hand-drawn by Tristan Needham [61]. Used with permision of the author.

The upper half-plane model

Beltrami was interested in finding a model of a *plane* having constant negative curvature that permitted lines to be extended indefinitely, as required by Euclid's second postulate. This is not possible on the pseudosphere because it is a surface of revolution, and it has a rim at its base. He came up with the half-plane model. The idea for the half-plane model is to use the pseudosphere as you would use a cylindrical paint roller. In doing so, you first stretch out its surface according to its metric. Once you've rolled out a full revolution of 2π, you just keep on rolling to fill a half plane. See Figure 3.7. For the full explanation, see [61]. While it is not obvious, it is not difficult to show that the geodesics in the model are circular arcs centered on the x-axis as well as vertical straight lines. The latter correspond to the generators of the pseudosphere corresponding to a tractrix.

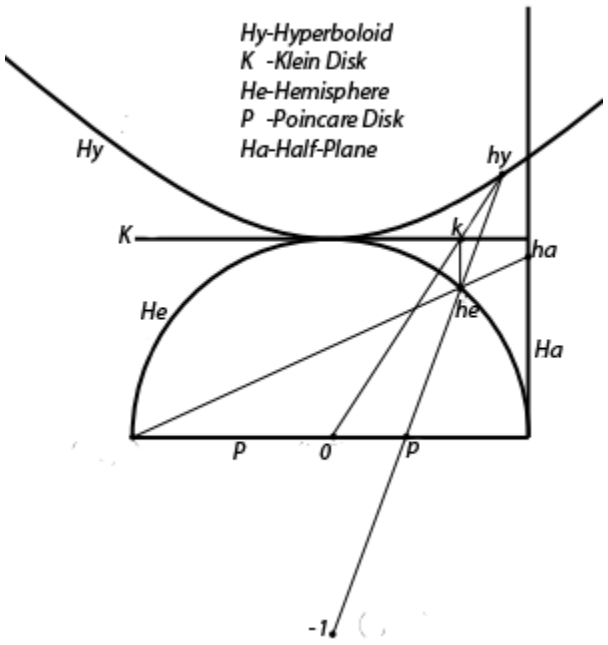


Figure 3.8: Relationship between hyperbolic models.

Attribution: Image in public domain.

The Beltrami-Klein and Beltrami-Poincaré models

In fact, Beltrami proposed several models of the hyperbolic plane in his 1868 papers. These were based on his hemisphere model. By projecting it in various ways, he obtained the upper half-plane model, as well as two models that are known as the Klein and Poincaré models. But it is Beltrami that first introduced these, and we shall we refer to them as the Beltrami-Klein and Beltrami-Poincaré models. The connections are summarized in Figure 3.8. The Beltrami-Klein and Beltrami-Poincaré models are both based in the unit disk. That is, the points of the model lie entirely within the unit disk, centerered at the origin. They do not include the unit circle at its boundary. As shown in Figure 3.9, these may be obtained by projecting the upper sheet of a two-sheet hyperboloid onto the plane. (Note, these are *not* the projections used by Beltrami.) The red model in Figure 3.9 was studied by Klein, and the green model in the same figure was studied by Poincaré in 1882. In terms of construction, the difference between these two models is that in the Beltrami-Klein model

the projection is made from the origin onto a unit disk centered on the z-axis at $z = 1$. In constrast, the projection for the Beltrami-Poincaré model is made from the point $z = -1$ onto a disk that lies in the xy-plane.

The geodesics (straight lines) on the hyperboloid are the intersections of the hyperboloid with planes through the origin, just as with the geodesics of an origin-centered sphere. In the Beltrami-Klein model, it is clear that the such a plane will cut the floating disk in a straight line (the interestion of a plane and this disk). While in the Beltrami-Poincaré model, straight lines are circular arcs that meet the unit disk at right angles. This does seem plausible. The intersection of the plane and the hyperboloid will extend radially out as z increases, and will do so without bound. Thus, in the projected model the lines must reach toward the boundary of the unit disk, and by symmetry, will meet it at right angles. The fact that these are indeed *circular* arcs follows from the fact that geodesics in the upper half-plane model are circular arcs. Indeed, the upper half-plane model can be mapped to the Beltrami-Poincaré model via the Möbius transformation of equation 3.14. It is well-known that Möbius transformations map circles to circles. A Möbius transformation is a fractional linear transformation, that is, a ratio of two linear transformations, see equation 3.15.

$$f(z) = \frac{iz + 1}{z + i} \tag{3.14}$$

$$\text{Möb}(z) = \frac{az + b}{cz + d} \tag{3.15}$$

As such, it can be decomposed into a translation, followed by complex inversion, followed by scaling and a rotation, followed by another transformation. Each of these steps maps circles to circles.

As well, diameters of the unit disk are also straight lines of the Beltrami-Poincaré model, with these corresponding to planes that are not tilted with respect to the z-axis.

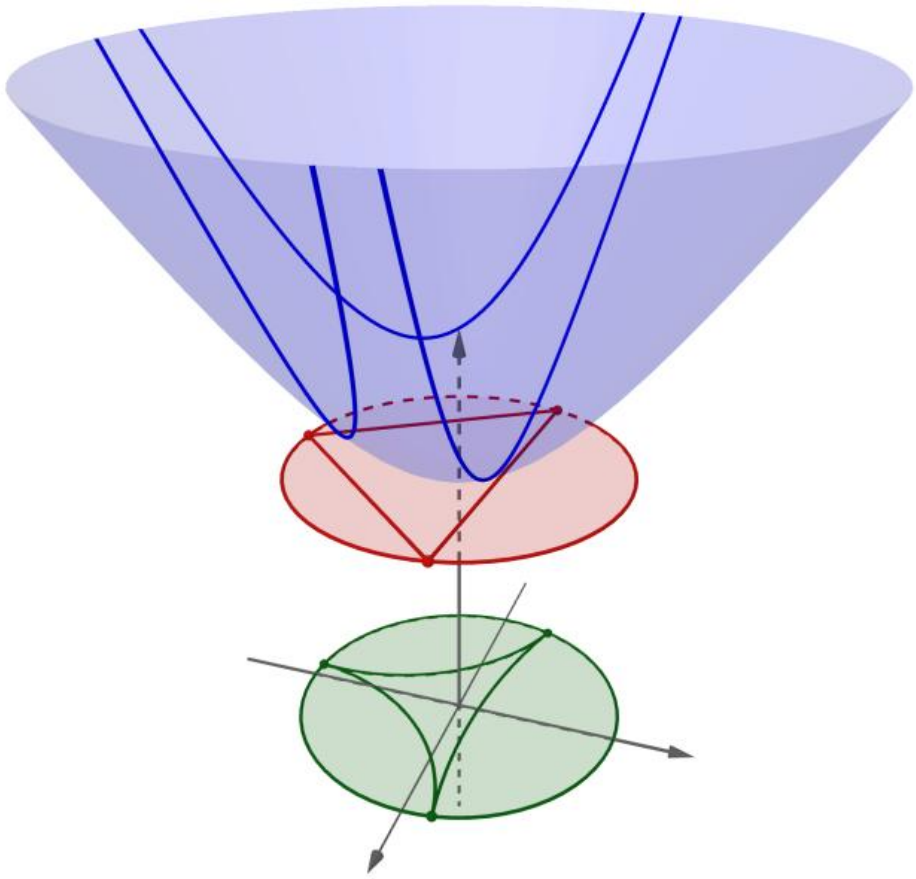


Figure 3.9: The Beltrami-Klein (red) and Beltrami-Poincaré disk (green) models as projections of the hyperboloid (blue).

Attribution: Image by Brice Loustau [50]. Used with permision of the author.

As the Beltrami-Poincaré model will be of particular interest to us, we report its metric in equation 3.16 , and its distance function in equation 3.17.

$$g_x = \left(\frac{2}{1 - ||x||^2}\right) g^E \tag{3.16}$$

$$d(u, v) = arccosh(1 + 2 \frac{||u - v||^2}{(1 - ||u||^2)(1 - ||v||^2)}) \tag{3.17}$$

Chapter 4

The Projected Cone Model

We now describe our proposed projected cone model. We begin by explaining our motiviation for introducing this model. We then define the mapping from the 2-dimensional cone to the open unit disk, and go on to deduce several properties including learning capacity, lines and geodesics, the Riemannian metric of the mapping, and distance measurement on the cone surface. Finally, we extend the projected cone model to higher dimensions in two ways.

4.1 Motivation for the Projected Cone Model

Representation Learning based in hyperbolic space has shown itself to be very effective. We saw how two of these models of hyperbolic space can be constructed as projections of one sheet of a two-sheet hyperboloid onto a unit disk. An interesting property of a hyperbola is that it has asymptotes. As the hyperbola reaches to infinity, it clings to these asymptotes. Indeed, as seen from Figure 4.1, we may say that it gets close to these asymptotes quite quickly, and then only gets closer and closer. As the hyperboloid is a surface of revolution, in three dimensions, the hyperboloid $x^2 + y^2 - z^2 = -1$ approaches the cone $x^2 + y^2 = z^2$.

If representation learning on a projected hyperboloid is so effective, and the hyperboloid lies so close to its asymptotes, it seems reasonable to enquire whether projecting the cone to the

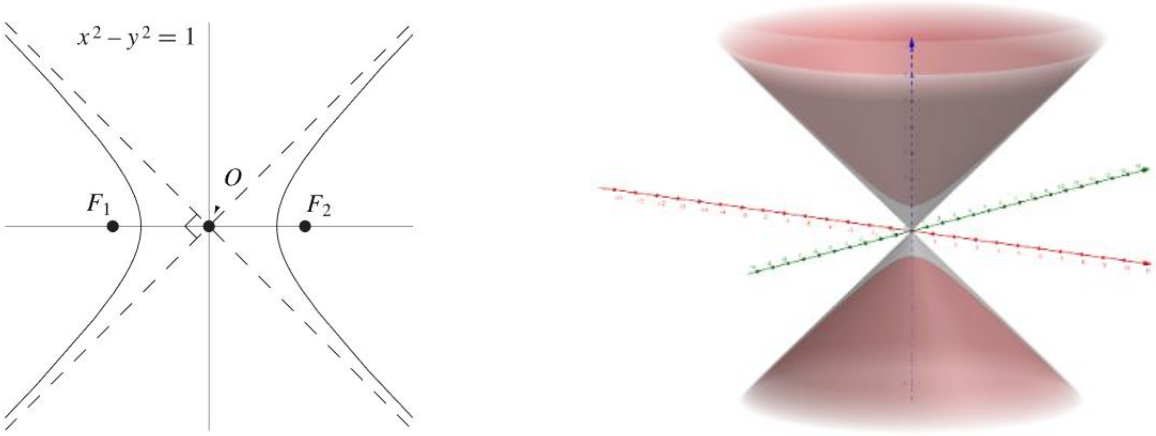


Figure 4.1: Asymptotes of a 2-D hyperbola and of a 3-D hyperboloid.

unit disk might also be an effective strategy for representation learning. Our central aim is to investigate this possibility. To be clear, just as the Beltrami-Klein and Beltrami-Poincaré are the projections of one sheet of a two-sheet hyperboloid, the cone we will be investigating will be the single-cone, with $z \geq 0$, not the double-cone.

We begin with the 2D cone surface (which we may view as embedded in 3D Euclidean space). We define the cone projection and discover some of its properties. We will extend the model to higher dimensions in section 4.9. Subsequently, we will turn our attention to using this model for machine learning.

The projected cone model, as we have just described it, has a similar construction to the Beltrami-Klein and Beltrami-Poincaré models for hyperbolic geometry. However, in a very concrete sense, our model can be said to leverage nothing but projection. A cone can be constructed by cutting a wedge in a disk and rolling it up at the seams. The cutting of the wedge can be thought of as an angular compression of the Euclidean plane. There is a bijective mapping from the entire plane to a plane with a wedge removed. Rolling this up into a cone doesn't change the nature of the surface at all. It merely embeds it in 3D space in a different way. It remains a surface of zero curvature. Thus, apart from the uninteresting angular compression to form a wedge, the projection from the cone to the unit disk is the only aspect of any importance in the construction of our projected cone model. We can therefore say that the projected cone model is the ideal model for investigating our hypothesis that

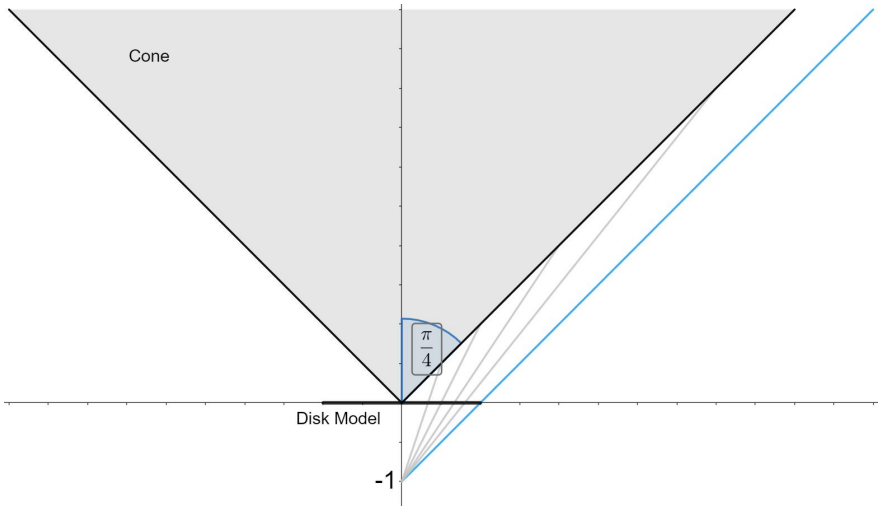


Figure 4.2: The construction of the projected cone model.

projection is key in explaining the benefits that hyperbolic geometry brings to representation learning.

4.2 Lift and Projection

Let us begin by describing our map from the cone to the unit disk. Thoughout, we will be working with the cone $x^2 + y^2 = z^2$ which makes an angle of $\pi/4$ with the z-axis as well as with the xy-plane. As with the Beltrami-Poincaré projection, the point of perspective is at $z = -1$, and we are projecting onto the xy-plane. The result is that the entire cone is projected into the unit circle. See Figure 4.2.

It is not uncommon to name the map from the curved surface (cone) to the flat surface a *projection*, regardless of the direction of the mapping, as this is usually clear from context. But we will call the mapping from the disk to the cone a *lift*, and use *projection* solely for the inverse mapping.

A horizontal circle on the cone (with constant z value) is mapped to a circle in the projected disk model lying in the xy-plane. As we can see from Figure 4.3, the radius of the former,

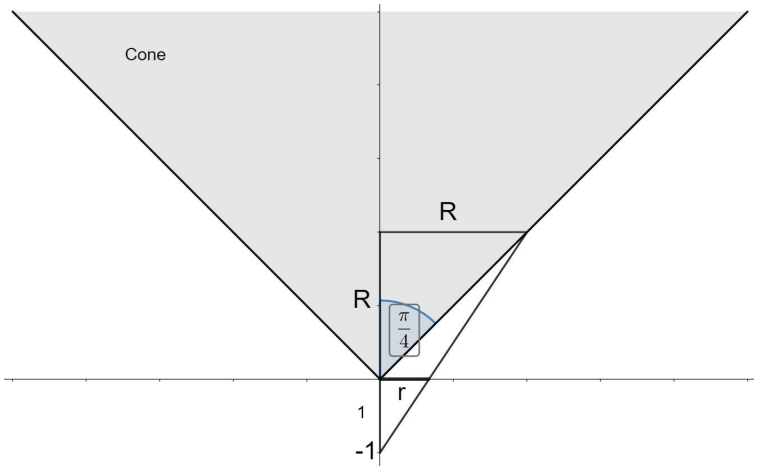


Figure 4.3: Determining the lift and projection mappings.

R, is related to that of the latter, r, in the following way:

$$\frac{R}{r} = \frac{1+R}{1},$$

$$R = \frac{r}{1-r}, \tag{4.1}$$

$$1 + R = \frac{1}{1-r}, \text{ and} \tag{4.2}$$

$$r = \frac{R}{1+R}. \tag{4.3}$$

When lifting or projecting, the radial angle about the z-axis stays the same. The distance from the z-axis is governed by the equations 4.1 and 4.3.

We use the following notational conventions: points on the cone will have a hat above them, those in the disk will be given a dot. Given a point $\hat{p} = (\hat{x}, \hat{y}, \hat{z})$ on the cone, R will always be understood to be the distance of this point from the z-axis, i.e., $R = \sqrt{\hat{x}^2 + \hat{y}^2}$. Similarly, for a point $\dot{p} = (\dot{x}, \dot{y})$ in the disk, $r = \sqrt{\dot{x}^2 + \dot{y}^2}$.

For lift, the scaling factor is R/r, yielding the lift formula

$$\hat{p} = \frac{1}{1-r}(\dot{x}, \dot{y}, r). \tag{4.4}$$

For projection, the scaling factor is r/R, yielding the projection formula

$$\dot{p} = \frac{1}{1+R}(\hat{x}, \hat{y}). \tag{4.5}$$

4.3 Some Observations

We note that the hyperboloid itself has positive curvature at all points, and that the cone has 0 curvature at all points (ignoring its apex). Indeed, the cone can be cut and unrolled onto the flat plane. Nevertheless, the projected models have negative curvature at all points. We see that projection is absolutely essential to the construction of these models.

Unlike the upper half-plane, Beltrami-Klein, and Beltrami-Poincaré models we have discussed, the projected cone model is *not* a model of hyperbolic geometry. That is, it does not have *constant* negative curvature.

4.4 Learning Capacity

In section 2.2.2, we argued that the Beltrami-Poincaré model has a much higher capacity for learning than Euclidean space. This was based on the fact that the hyperbolic radius of concentric circles in the Beltrami-Poincaré disk increases exponentially. Let us see if the same phenemenon holds for the projected cone.

A point $\dot{p}$ at a distance r from the origin in the projected cone disk corresponds to a point $\hat{p}$ at a distance of $\sqrt{2}R = \sqrt{2}r/(1-r)$ from the origin. As r goes to 1, the only interesting part of this quantity is $1/(1-r)$. This is just the sum of an infinite geometric series with $r < 1$:

$$\frac{1}{1-r} = \sum_{k=0}^{\infty} r^k. \tag{4.6}$$

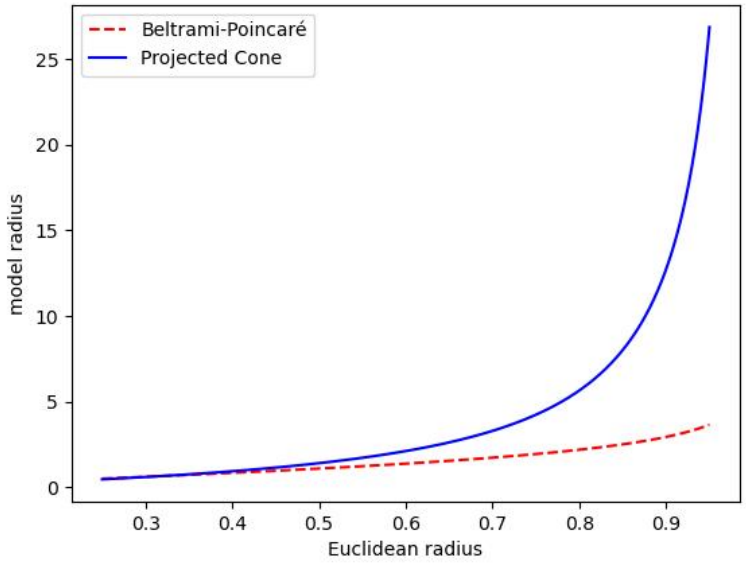


Figure 4.4: Model radii corresponding to Euclidean radii within the unit disk.

Now, recall the MacLaurin series expansion of the exponential function:

$$e^r = \sum_{k=0}^{\infty} \frac{r^k}{k!}. \tag{4.7}$$

Comparing coefficients, we conclude that the learning capacity of the projected cone model exceeds that of the Beltrami-Poincaré model for hyperbolic geometry. In figure 4.4, we plot the model radii for the Beltrami-Poincaré and projected cone models against Euclidean radii within unit disk. We see that the projected cone model radii grow much more quickly than those of the the Beltrami-Poincaré model.

4.5 Lines and Geodesics

As the cone is a surface of 0-curvature, it can be cut and unrolled so as to lie flat in the plane. Indeed, to construct a finite portion of a cone, we may begin by cutting a wedge out

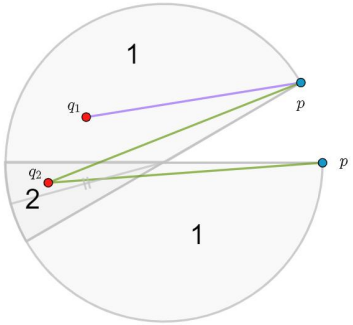
of a disk, and then glueing the wedge sides together. Doing this does not stretch the surface at all, so distances are preserved by these manipulations. Given two points in the unrolled and flattened cone, the shortest distance between them will be the length of the Euclidean line segment joining them. We can then roll up the disk into a cone to see the straight line on the cone.

A geodesic is a curve that intrinsically appears straight. That is, to an inhabitant of the surface, following a geodesic feels like walking in a straight line, without curving to the left or to the right. Geodesics are associated with shortest paths between pairs of points. However, there may be more than one geodesic path connecting a pair of points on a surface. The shortest path will be one of these geodesic paths simply because veering off the course of a geodesic lenghtens that path.

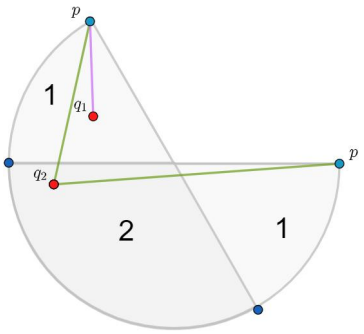
Now let us determine the geodesics on the cone surface. Begin by considering a cone with a very wide opening. It may be constructed by cutting a narrow wedge into a disk, see Figure 4.5(a). When rolled into a cone, this cone will be fairly flat. Consider the points p and q_1. We may arbitrarily place the point p on the cut line of the wedge. It is shown twice in the figure as these are actually exactly the very same point on the cone. Also note that q_1 is in a region labelled as 1. The straight line from p to q_1 gives the geodesic, and thus the shortest path from p to q_1. The situation is different for q_2, which is in region 2. In this case, we may reach q_2 from either of the illustrated points p. Thus, on the cone, there are two geodesics from p to q_2. Each passes by on a different side of the apex of the cone (the origin in the unrolled disk segment). The top path is shorter than the bottom one, as q_2 resides in the upper half of region 2. We have drawn a bisector of the region 2 wedge.

Now consider what happens as the wedge angle increases, as in Figure 4.5(b). We see that the 1 regions decrease in size, and that region 2 increases in size. When the wedge forms an angle π, we have the 30° cone (angle made with the z-axis). In this case, the 1-geodesic region has all but vanished. The exception is along the wedge cut itself, see Figure 4.5(c).

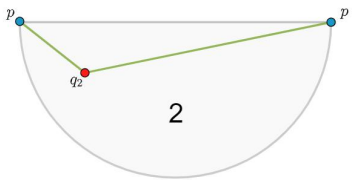
If we continue to shrink the wedge, a 3-geodesic region will appear as shown in Figure 4.5(d). However, this situation does not concern us as we are working with the 45° cone, which corresponds to the situations in Figure 4.5(b).



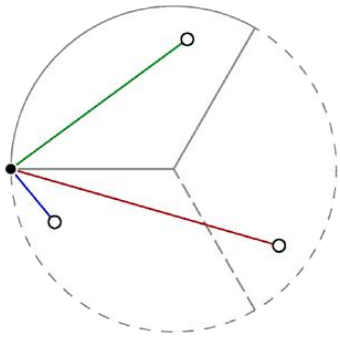
(a) With a small wedge, there are 1-geodesic and 2-geodesic regions.



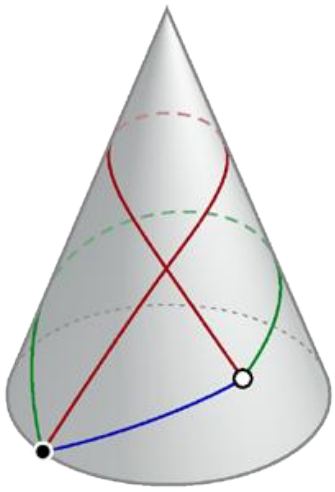
(b) With a somewhat larger wedge, the 1-geodesic regions shrinks while the 2-geodesic region grows.



(c) When the wedge is a semi-circle, the 1-geodesic region degenerates to a line.



(d) A 3-geodesic region appears.



(e) The three geodesics seen on the cone.

Figure 4.5: Cone Geodesics

4.6 Wedge Angle

Not surprisingly, there is a definite relationship between the cone angle (as measured to the z-axis), and the angle of the missing wedge opening of the unrolled cone in the 2D plane, see

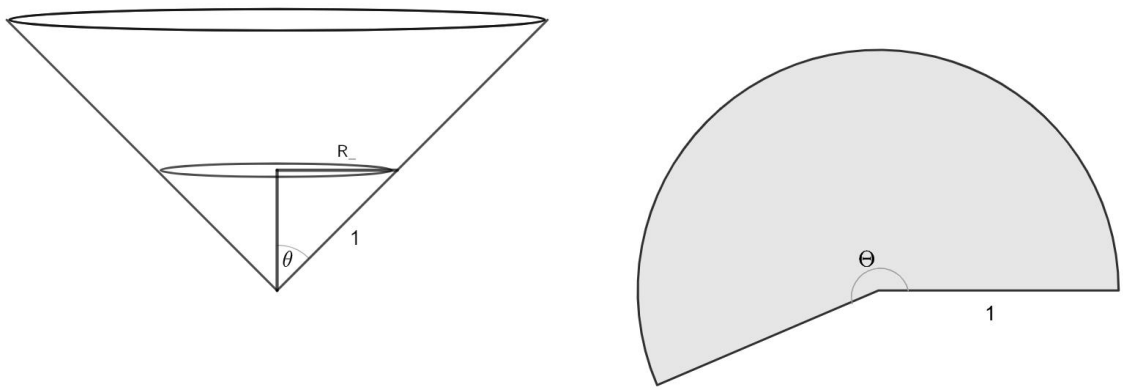


Figure 4.6: Cone angle to wedge angle.

Figure 4.6. The angle Θ is equal to the arc length of the semicircle of radius 1 on the right side of the figure. This length corresponds to the circumferernce of the circle with radius $R_{\_}$ in the left hand side of the figure. Thus,

$$\Theta = 2\pi R_{\_} = 2\pi \sin\theta. \tag{4.8}$$

For the 45° cone, Θ is $\sqrt{2}\pi \approx 1.414\pi$, and for the 30° cone, Θ is π. We therefore see that the 45° cone we will be working with indeed corresponds to Figure 4.5(b) which has both 1-geodesic and 2-geodesic regions.

4.7 Lift, Unroll, and Measure

We call our strategy for computing the distance between two points on a cone *lift, unroll, and measure*. It reflects the distance between lifted points precisely in terms of the shortest geodesic path between the points. First, we lift the two points from the disk to the cone. Then, we unroll the cone into the plane by cutting it along a radial line starting at the cone's apex. Because the cone has 0 curvature, it can be unrolled to lie flat in the plane without stretching any part of its surface, as has already been mentioned. We may now take the Euclidean distance between these points, and this will be *exactly* the geodesic distance between them on the cone. We will work this out for the 2D cone.

Consider two points $\dot{p}$ and $\dot{q}$ in the unit disk, their lifted counterparts $\hat{p}$ and $\hat{q}$ on the cone, and their unrolled counterparts $\bar{p}$ and $\bar{q}$ in the plane. Ignoring their z-coordinates, $\hat{p}$ and $\hat{q}$ point in the same radial direction as $\dot{p}$ and $\dot{q}$. When the cone is unrolled, this angle gets compressed. The amount of this compression is in proportion to the wedge angle we worked out in the previous section. Indeed, the full circular angle of 2π is compressed to Θ. And so for the 45° cone, the scaling factor is $\frac{\Theta}{2\pi} = \frac{\sqrt{2}\pi}{2\pi} = \frac{1}{\sqrt{2}}$. Thus, for our cone,

$$\angle(\bar{p}, \bar{q}) = \frac{1}{\sqrt{2}}\angle(\dot{p}, \dot{q}) = \frac{1}{\sqrt{2}} arccos(\frac{(\dot{p} \cdot \dot{q})}{||\dot{p}||\,||\dot{q}||}). \tag{4.9}$$

There is also a straightforward relation between the $||\dot{p}||$ and $||\bar{p}||$. From equation 4.1, and noting that the length of $\hat{p} = \sqrt{2}R$ for the 45° cone, we see that

$$||\bar{p}|| = ||\hat{p}|| = \sqrt{2}\left(\frac{||\dot{p}||}{1 - ||\dot{p}||}\right). \tag{4.10}$$

The distance between $\bar{p}$ and $\bar{q}$ follows from the cosine law:

$$d(\bar{p}, \bar{q})^2 = ||\bar{p}||^2 + ||\bar{q}||^2 - 2||\bar{p}||\,||\bar{q}|| \cos\angle(\bar{p}, \bar{q}) \tag{4.11}$$

$$= ||\bar{p}||^2 + ||\bar{q}||^2 - 2||\bar{p}||\,||\bar{q}|| \cos\left(\frac{1}{\sqrt{2}} arccos(\frac{(\dot{p} \cdot \dot{q})}{||\dot{p}||\,||\dot{q}||})\right). \tag{4.12}$$

Now, if we are in the 2-geodesic region, then it seems we have to calculate two distances and take the minimum. In fact, this isn't necessary. The above computation always yields the minimum value. The $\angle(\dot{p}, \dot{q})$, as computed by equation 4.9, belongs to the interval $[0, \pi)$. So, for the compressed angle, $\angle(\bar{p}, \bar{q}) \in [0, \frac{1}{\sqrt{2}}\pi)$. For a point in the 2-geodesic region, we need to compare the angle between $\bar{p}$ and $\bar{q}$ to the angle between $\bar{p'}$ and $\bar{q}$, where $\bar{p'}$ is located on the other straight edge of the wedge (both are labelled "p" in Figure 4.5(b)). We are therefore looking at the minimum of $\angle(\bar{p}, \bar{q})$ and $\sqrt{2}\pi - \angle(\bar{p}, \bar{q})$. But as $\angle(\bar{p}, \bar{q}) \in [0, \frac{1}{\sqrt{2}}\pi)$, $\sqrt{2}\pi - \angle(\bar{p}, \bar{q}) \in [\sqrt{2}\pi - \frac{1}{\sqrt{2}}\pi, \sqrt{2}\pi) = [\frac{1}{\sqrt{2}}\pi, \sqrt{2}\pi)$. Another, perhaps easier, way of seeing this is to notice that because $\angle(\dot{p}, \dot{q}) = arccos(\frac{(\dot{p} \cdot \dot{q})}{||\dot{p}||\,||\dot{q}||}) \in [0, \pi)$, we are already focused on the shorter of the two geodesic paths on the cone.

4.8 Metric

We compute the metric of the mapping from the cone to the the projected cone. We seek to determine the relationship between an infinitesimal displacement $d\hat{s}$ on the cone and the corresponding displacement in the disk, ds. We decompose these displacements into a radial component (with subscript 2), and a component perpendicular to the radial component (with subscript 1). By the Pythogrean theorem, we have

$$d\hat{s}^2 = d\hat{s}_1^2 + d\hat{s}_2^2.$$

We can deduce the relationship between the radial components $d\hat{s}_1$ and ds_1 using Figure 4.7 and equation 4.1:

$$\frac{d\hat{s}_1}{ds_1} = \frac{Rd\theta}{rd\theta} = \frac{R}{r} = \frac{1}{1-r}. \tag{4.13}$$

For the relation between $d\hat{s}_2$ and ds_2, refer to Figure 4.8. Using similar triangles and equation 4.2, we have[1]

$$\begin{aligned} \frac{x}{ds_2} = \frac{[p\ b]}{[p\ a]} = \frac{1+R}{1} = \frac{1}{1-r}, \\ x = \frac{1}{1-r} ds_2. \end{aligned} \tag{4.14}$$

We next note that the angle Ψ is *ultimately* equal to $\tilde{\Psi}$ as $d\Psi \to 0$, and we focus on the triangle with bold sides. One angle is $\pi/4$ because that is the cone's angle. The triangle's other two angles are

$$\begin{aligned} \beta &= \frac{\pi}{4} - \Psi, \text{and} \\ \alpha &= \pi - \frac{\pi}{4} - (\frac{\pi}{4} - \Psi) = \frac{\pi}{2} + \Psi. \end{aligned}$$

[1] We notate the distance between two points a and b as $[a\ b]$.

Applying the sine law and the trigonometric addition identities, we have

$$\frac{d\hat{s}_2}{\sin\alpha} = \frac{x}{\sin\beta},$$
$$d\hat{s}_2 = x \times \frac{\sin\alpha}{\sin\beta}.$$

We can simplify $\sin\alpha$ and $\sin\beta$:

$$\begin{aligned}\sin\alpha &= \sin(\frac{\pi}{2} + \Psi) = \sin\frac{\pi}{2}\cos\Psi + \cos\frac{\pi}{2}\sin\Psi \\ &= cos\Psi;\end{aligned}$$

$$\begin{aligned}\sin\beta &= \sin(\frac{\pi}{4} - \Psi) = \sin\frac{\pi}{4}\cos\Psi - \cos\frac{\pi}{4}\sin\Psi = \frac{1}{\sqrt{2}}(\cos\Psi - sin\Psi) \\ &= \frac{1}{\sqrt{2}}(\cot\Psi - 1)\sin\Psi \\ &= \frac{1}{\sqrt{2}}(\frac{1}{r} - 1)\sin\Psi \\ &= \frac{1}{\sqrt{2}}\frac{1-r}{r}\sin\Psi.\end{aligned}$$

Thus,

$$\begin{aligned}d\hat{s}_2 &= x \times \frac{\cos\Psi}{\frac{1}{\sqrt{2}}\frac{1-r}{r}\sin\Psi} \\ &= \frac{ds_2}{1-r} \times \frac{\sqrt{2}r}{1-r} \times \cot\Psi \\ &= \frac{\sqrt{2}rds_2}{(1-r)^2} \times \frac{1}{r} \\ &= \frac{\sqrt{2}ds_2}{(1-r)^2}.\end{aligned} \tag{4.15}$$

Combining equations 4.13 and 4.15, we finally obtain the full metric for the mapping:

$$d\hat{s}^2 = \frac{1}{(1-r)^2}\left(ds_1^2 + \frac{2}{(1-r)^2}ds_2^2\right). \tag{4.16}$$

We see that our metric is not conformal. It stretches more in the radial direction.

4.9 Cone Model in Higher Dimensions

A little intuition

It is very difficult to imagine any space, or object in it, that is more than 3-dimensional (3D for short). The entertaining book *Flatland* by Edwin A. Abbott is fascinating attempt at doing so [2]. Let us try to imagine what a 3D cone might look like, working by analogy with 2D[2]. Reconsider our projected model of the 2D cone into the unit disk. We can think of this disk in the xy-plane as being constituted of concentric circles of radius less than 1. Then, we smoothly elevate each disk along the z-axis, expanding each circle as we raise it. For our cone, which has an angle of $\pi/4$ with respect to the z-axis, we elevate a disk of radius r to a height of $R = r/1 - r$, and we stretch it by a factor of $R/r = 1/1 - r$. By analogy, if we begin with a 3D unit ball consisting of concentric spheres of radius r, with $r \epsilon [0, 1)$, then we may *elevate* each such sphere to a height of $r/1 - r$ in a fourth dimension.

In the 2D case, the elevated circles created a smooth 2D surface with each elevated circle *just touching the next*. Now, our elevated spheres create a smooth 3D "surface" in 4 dimensions, with the spheres, once again, just touching one another. We will continue to call the direction of elevation the z-axis, and we will refer to the other dimensions as x_1, ..., $x_{\mathcal{D}}$. Whether useful or not, with this model in mind, we will now look to generalize our earlier formulas to higher dimensions.

[2]We refer to the usual cone residing in 3D space as the 2D cone, as it is a surface having no volume. This aligns with standard practice as the unit circle is known as $\mathcal{S}^1$ and the unit sphere is known as $\mathcal{S}^2$. The circle is a 1-dimensional object that can live in a higher dimensional space (not just 2D).

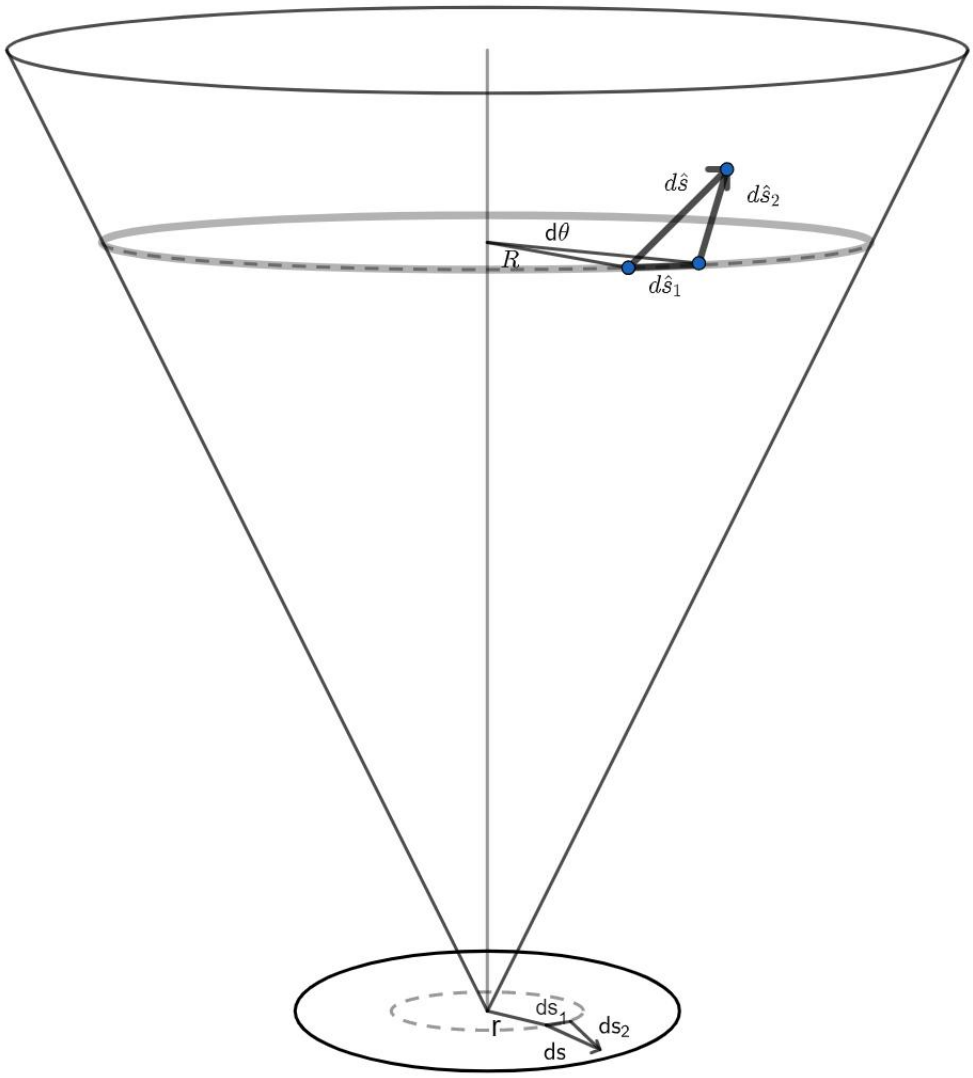


Figure 4.7: Cone metric I

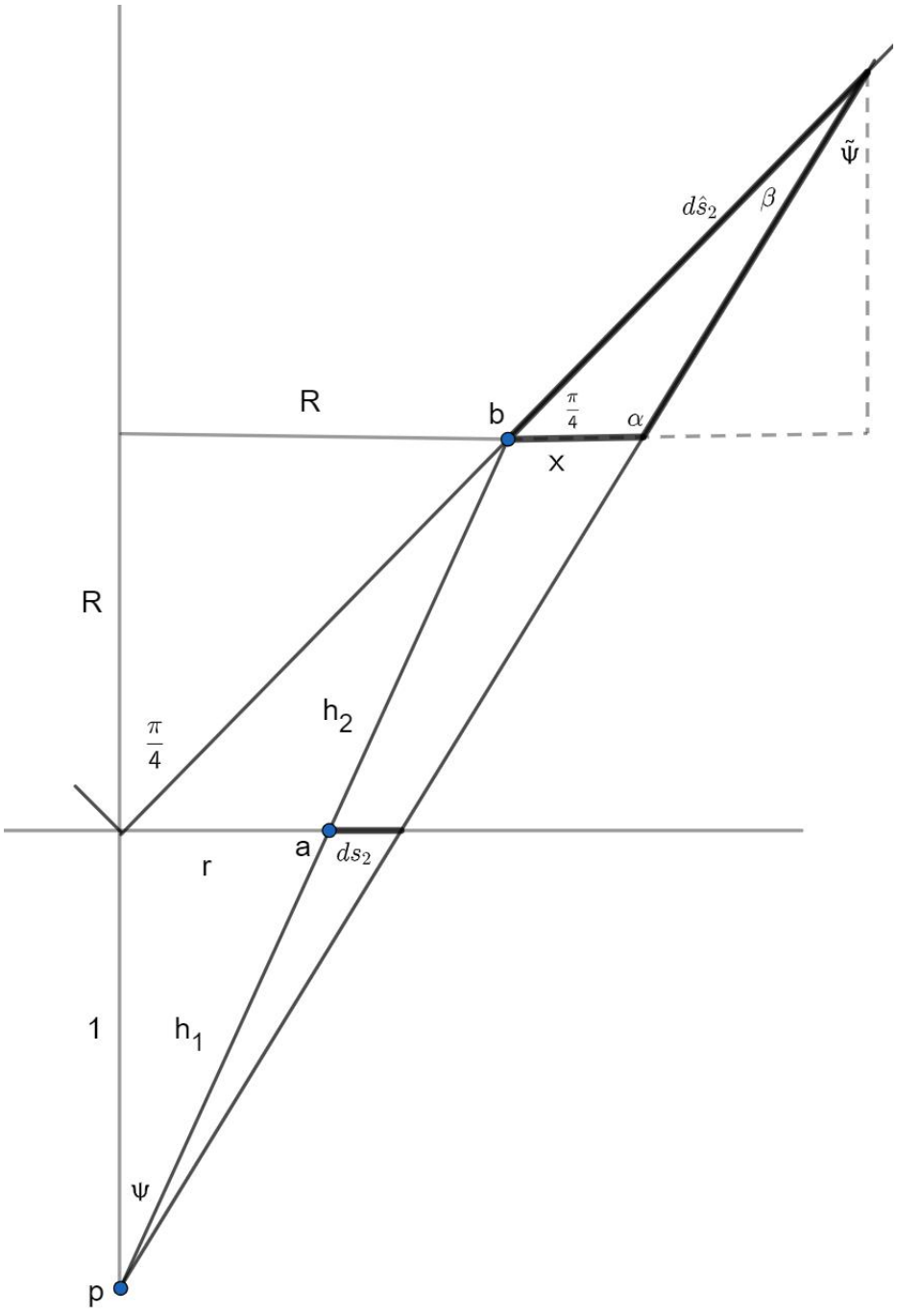


Figure 4.8: Cone metric II

Lift and project

Consider a point $\dot{p} = (\dot{x}_1, ..., \dot{x}_{\mathcal{D}})$ in the unit ball of dimension $\mathcal{D}$. It sits on a $(D-1)$—hypersphere of radius $||\dot{p}||$. This hypersphere will be lifted to height of $R = \frac{||\dot{p}||}{1-||\dot{p}||}$, and thus its z coordinate will be R. The point $\dot{p}$ will retain its orientation with respect to all other axes, x_1, $...x_{\mathcal{D}}$. The hypersphere containing $\dot{p}$ will be stretched to a radius of $\frac{1}{1-||\dot{p}||}$. Thus, the higher-dimensional lift equation corresponding to equation 4.4 is

$$\hat{p} = \frac{1}{1-r}(\dot{x}_1, ..., \dot{x}_{\mathcal{D}}, r). \tag{4.17}$$

For the projection of a point $\hat{p} = (\hat{x}_1, ..., \hat{x}_{\mathcal{D}}, R)$, we remove the z-dimension, and we have the same scaling factor of $\frac{1}{1+R}$ for our hyperspheres that we had for the horizontal circles making up the 2D cone. This yields the following higher dimensional analog of equation 4.5.

$$\dot{p} = \frac{1}{1+R}\,(\hat{x}_1, ..., \hat{x}_{\mathcal{D}}), \tag{4.18}$$

where $R = \sqrt{\hat{x_1}^2 + ... + \hat{x_{\mathcal{D}}}^2}$.

Metric

The metric formula 4.16 can be easily adapted to higher dimensions. Recall that it is a non-conformal metric. The expansion is $\frac{1}{1-r}$ in the non-radial direction and is $\frac{\sqrt{2}}{(1-r)^2}$ in the radial direction. The non-radial scaling factor of $\frac{1}{1-r}$ is an immediate consequence of the fact that, in the unit disk, the circle has radius r, but when lifted it has radius $R = \frac{r}{1-r}$. According to our previously described intuition for high dimensional cones, there is an analogy between the $\mathcal{D}-1$-dimensional hyperspheres making up the $\mathcal{D}$-dimensional cone, and the circles making up the 2D-cone. When lifted, these hyperspheres will be stretched by the same factor of $\frac{1}{1-r}$. The dimensions of these spheres are affected in the same way as the non-radial dimension in the 2D case. We may in some sense consider these dimensions as the non-radial dimensions.

The z-dimension is the dimension in which the lift occurs. Just as in the 2D case, it will be

stretched more than the non-radial dimensions. This is due to the combined affect of the spheres being stretched and the cone having an incline of $\pi/4$. The analysis is exactly the same as in the 2D case, so that we may conclude that an extra stretching factor of $\frac{\sqrt{2}}{(1-r)}$ applies.

Thus, the higher dimensional analog of the metric 4.16 is

$$d\hat{s}^2 = \frac{1}{(1-r)^2}\left(ds_1^2 + ... + ds_{\mathcal{D}-1}^2 + \frac{2}{(1-r)^2}ds_{\mathcal{D}}^2\right). \tag{4.19}$$

Note that there are $\mathcal{D}-1$ non-radial dimensions, and there is just one radial dimension, for a total of $\mathcal{D}$ dimensions, as expected.

Lift, unroll, and measure

Lift, unroll, and measure was our strategy for determining the distance between two points on a 2D-cone. Because the cone has 0 curvature, it can be unrolled into the plane without stretching. The Euclidean distance between the two points once unrolled is therefore precisely the same as the distance between them while on the cone. We now see how this applies in higher dimensions. The higher-dimensional lift has already been covered, but what does it mean to unroll a $\mathcal{D}$-dimensional cone in $\mathcal{D}+1$-dimensional space on to a flat, Euclidean space of $\mathcal{D}$ dimensions? We follow the same approach we used in working out equation 4.12. There is no issue with the computation of $||\bar{p}||$ from $||\dot{p}||$. Next, we figure that the angle between $||\bar{p}||$ and $||\bar{q}||$ is also compressed as we "unroll" $\mathcal{D}$ dimensions about the z-axis. Thus, we likewise compress their angle by $1/\sqrt{2}$.

While grounded in intuition, this extension of our 2D distance formula for the projected cone is somewhat speculative. We therefore take another approach to generalizing the cone model to higher dimensions.

4.10 Product Space Model

We may obtain a second higher-dimensional version of our 2D-cone by taking several copies of it. This is accomplished using products of the 2D-cone. Let X be a set. The product space

$$X^N = X \times ... \times X$$

consists of N copies of X. So, if $x_1 \in X$, ..., $x_N \in X$, then $(x_1, ..., x_N) \in X^N$. In the case of the 2D cone, each X is itself a two dimensional pair. So, a product of N cones will have $2N$ dimensions. We will refer to this as $\mathcal{PC}^{2N}$, or more simply as the $\mathcal{PCP}$ model when it is not important to call out its dimensionality.

For the distance between two points $p, q \in \mathcal{PC}^{2N}$, we compute the distance between them pair by pair by pair, and then combine these by applying the Pythagorean theorem. That is, $\mathcal{PC}^{2N}$ is treated as a Euclidean product of 2D cone spaces. Let us define the projection operator which extracts the i^{th} cone component from $\mathcal{PC}^{2N}$ as $\Pi_i(p) = (p_{2i-1}, p_{2i})$. We then have

$$d(p, q)^2 = \sum_{i=1}^{N} d(\Pi_i(p), \Pi_i(q))^2, \tag{4.20}$$

where $d(p, q)$ is as given by equation 4.12.

4.10.1 Learning capacity analysis

We will refer to the projected cone model as the $\mathcal{PC}$ model, and to the projected cone product model of section 4.10 as the $\mathcal{PCP}$ model. In this section we provide an estimate of the learning capacity of the $\mathcal{PCP}$ model and compare it to that of $\mathcal{PC}$. The approach is based on comparing the volumes of n-dimensional hyperspheres in each model.

In general, the volume of a hypersphere of radius r in n-dimensions is proportional to r^n, [87]. We disregard the precise constants, which depend on the dimension. Indeed, Figure 4.9 shows that at larger dimensions these constants become less relevant for values of r near 1, which is precisely our case.

In section 4.4, *Learning Capacity*, we described the learning capacity of the $\mathcal{PC}$ model. To do so, we discussed the relationship between lengths in the 2D $\mathcal{PC}$ disk model and the corresponding points lifted to the 2D cone surface. Consider a point $\dot{p}$ in the disk model that is at a Euclidean distance r from the origin. The length of its lift, $\hat{p}$, from the origin is $\sqrt{2}R = \sqrt{2}r/(1-r)$.

Thus, for the $\mathcal{PC}$ model, the volume of an n-dimension hypersphere is

$$Volume_{\{\mathcal{PC}\}} \propto \left(\sqrt{2}r/(1-r)\right)^n . \tag{4.21}$$

In the above, $r \in [0, 1)$ represents the Euclidean distance from the origin of a point in the $\mathcal{PC}$ model.

The $\mathcal{PCP}$ model is the repeated product of two-dimensional $\mathcal{PC}$ spaces. In n dimensions, there are $n/2$ such spaces in the product. Thus,

$$\begin{aligned} Volume_{\{\mathcal{PCP}\}} &\propto \left[\left(\sqrt{2}r/(1-r)\right)^2\right]^{\frac{n}{2}} \\ &= \left(\sqrt{2}r/(1-r)\right)^n . \end{aligned} \tag{4.22}$$

We conclude that the learning capacities of the $\mathcal{PC}$ and $\mathcal{PCP}$ models are similar, and of the same order.

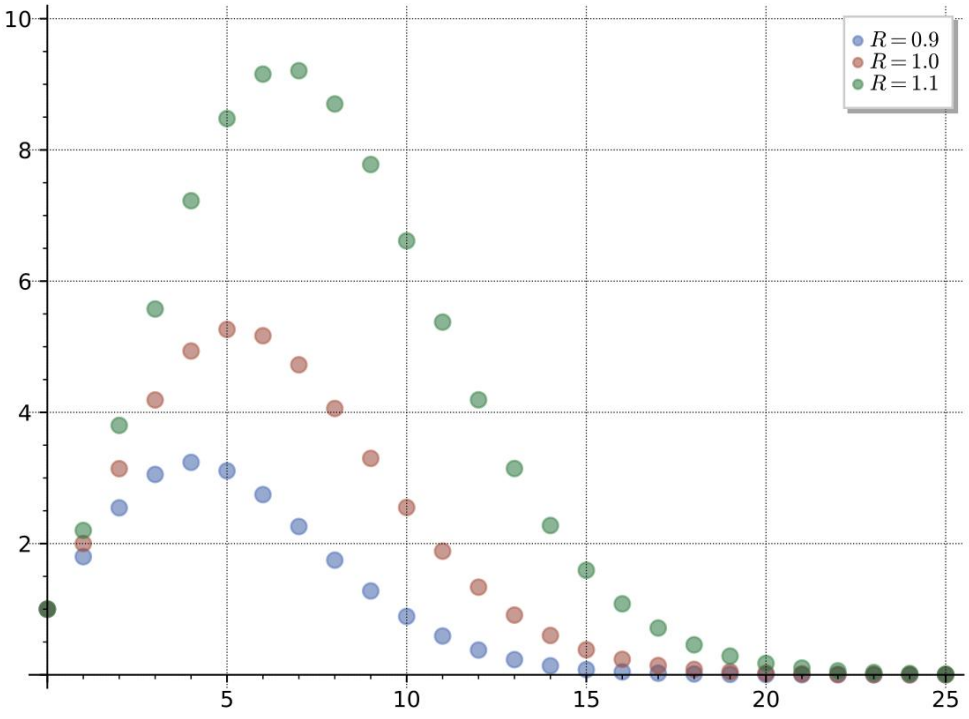


Figure 4.9: Volumes of hyperspheres in various dimensions.
Volumes of unit balls (`https://en.wikipedia.org/wiki/File:Volumes_of_unit_balls.svg`), licensed under Creative Commons CC0 1.0 Universal Public Domain Dedication (`https://creativecommons.org/publicdomain/zero/1.0/deed.en`).

Chapter 5

Methodology

Our experimental setup is based on work from Facebook (now Meta) [62], [63]. They have created a framework for learning word representations from WordNet [57] using a shallow neural network whose weights are restricted to some Riemannian manifold. They provide a training loop leveraging Bonnabel's Riemannian stochastic gradient descent [9]. They have also provided code for evaluating the generated word embeddings. This infrastructure has been used by them, and by others, to experiment learning with different manifolds, loss functions, and regularization strategies. We hook into this framework to learn respresentations on the manifolds of our projected cone and projected cone product models.

5.1 Training Data: WordNet

WordNet is a lexical database of English words. It includes several relations between the words. The most prominent is synonymy. WordNet groups synonyms into what it calls *synsets*. These synsets are then the subject of other relations. The relation with which we will be working is hypernymy. This is the *ISA* relation. A hyponym is more specific in meaning than its hyperyms. For example, *elephant* is a hyponym of *mammal*, while *mammal* is a hypernym of *elephant*.

The hypernym relationships between nouns are extracted from WordNet, and the transitive closure of this set is computed. The result is a set of pairs of words that are either directly or indirectly related by the hypernymy relationship. These are used for training our models.

5.2 Objective

Our goal is to find vector representations of the words that preserve the hypernymy relation by keeping the vectors of such related words near each other. Words that are directly related should be more closely positioned to each other than words that are indirectly related. That is, we hope to discover the latent hypernymy hierarchy (recall that we train on the transitive closure of the hypernymy relation). And, of course, words that are not related at all should be positioned further apart on average as compared to words that are either directly or indirectly related.

5.3 Model

The training model used is shallow, as was often the case for learning word representations prior to the introduction of Transformer-based [84] architectures such as GPT [70], and BERT [21] and their many variants. Examples include the Word2Vec [56] [55] and GloVe [67] models.

A training sample consists of a pair of nouns, n_{f} and n_{o}, which we will call the *focus* noun and the *other* noun, respectively. These are related by hypernymy, although not necessarily directly as we draw samples from the transitive closure of this relationship. We then create $\#_{\mathrm{neg}}$ negative samples consisting of n_{f} and other randomly selected words from the vocabulary. The quantity $\#_{\mathrm{neg}}$ is a hyperparameter of the model. Thus the model receives two vectors of size $\#_{\mathrm{neg}} + 1$:

$$\vec{n_{\mathrm{f}}} = [\ n_{\mathrm{f}},\ n_{\mathrm{f}},\ \ ...,\ \ n_{\mathrm{f}}\ ],\ \text{and}$$
$$\vec{n_{\mathrm{o}}} = \left[\ n_{\mathrm{o}},\ neg_1,\ ...,\ neg_{\#_{\mathrm{neg}}}\ \right].$$

Essentially, we have a $(\#_{\mathrm{neg}} + 1)$-way classification problem, with the correct answer always found in the first column: $\overrightarrow{target} = [\ 1,\ \ 0, ...,\ 0\ ]$, The model computes the distance column-wise between each entry of $\vec{n_{\mathrm{f}}}$ and $\vec{n_{\mathrm{o}}}$, negates these distances, computes the softmax of these results, and finally applies the cross-entropy loss function.

5.3.1 Role of negative sampling

The negative samples affect the softmax calculation prior to the cross-entropy loss being applied. If some of the negative samples are close in distance to n_{f}, then these will lower the result of the softmax for the true answer found in the first column, which will lead to a larger cross-entropy with $\overrightarrow{target}$. On the other hand, if all of the negative samples are reasonable far from n_{f}, then the result of the softmax will be almost equal to $\overrightarrow{target}$ and the cross-entropy loss will be very small indeed. We may therefore expect that the loss will decrease as more suitable locations are found for the vector representations of *all* nouns.

5.4 Riemannian SGD

For doing machine learning on a Riemannian manifold, the usual stochasting gradient descent (SGD) optimization needs to be adapted. S. Bonnabel has done this [9]. We begin by reviewing standard SGD.

5.4.1 Standard SGD

Stochastic gradient descent, SGD, is an optimization strategy that is effective at finding local minima of functions. In machine learning, these functions are called *cost* functions. A cost function is a function of (many) parameters, w. It is these parameters that we seek to adjust

so as to minimize the cost function. To do so, we leverage data. A loss function, $Q(z, w)$, is defined in terms of data, z, and the parameters, w. The cost is the average value of the loss function evaluated over all of the available data. However, not all data is assumed to be equally likely. We thus model the data as a probability distribution, P. Therefore,

$$C(w) = \mathbb{E}\, Q(z, w) = \int Q(z, w)\, dP(z). \tag{5.1}$$

In practice, the probability distribution is unknown. Thus, in machine learning, we use training data to compute an appoximate loss. With SGD, we use a single training instance, z_t, to compute an estimate of the loss. (With batch or mini-batch SGD, we would take the average loss computed over several training samples as an estimate of the loss.) SGD then updates the parameters to minimize the loss. This is accomplished by taking a step along the negative of the estimated gradient of the cost function, $H(z_t, w_t)$, where w_t are the weights at the start of step t. That is, H is the gradient of the cost function and is evaluated at (z_t, w_t). We expect this to be a good approximation of the average gradient of the cost function over all possible data:

$$H(z_t, w_t) \approx \mathbb{E}_z\, H(z, w_t) = \int H(z, w_t)\, dP(z) = \nabla C(w_t) \tag{5.2}$$

The size of our step is further scaled by a hyper-parameter, γ, called the learning rate. A parameter update is thus

$$w_{t+1} = w_t - \gamma H(z_t, w_t). \tag{5.3}$$

5.4.2 Riemannian SGD

Riemannian SGD (RSGD) introduced by Bonnabel [9], is a stochastic gradient descent algorithm for Riemannian manifolds. The parameters, w, are now constrained to belong to a Riemannian manifold. The overall approach is much like SGD. The gradient step follows a vector that is in the tangent space of the manifold at w_t. Following this step, we make an

adjustment to ensure that the final update keeps the parameters on the manifold. This is done by applying the *exponential* map:

$$w_{t+1} = \exp_{w_t}(-\gamma H(z_t, w_t)) \tag{5.4}$$

Recall that in Riemannian geometry the exponential map at a point p of a manifold is defined as a mapping from the vectors v of the tangent space at p, $T_p(\mathcal{M})$, to points that are near said vectors, but which reside on $\mathcal{M}$. This is done by following a geodesic along $\mathcal{M}$ from p in the direction of v. Let $\gamma(t)$ be a geodesic curve on $\mathcal{M}$ such that $\gamma(0) = p$, and $\gamma'(0) = v$. Then the exponential map is $exp_p(v) = \gamma(1)$. Determining the exponential map is very often a difficult problem to solve. In practice, it is sufficient to use a *retraction* in place of the exponential map. A retraction is a first-order approximation to the exponential map. Exact definitions of the notion depend on use cases. The idea is that once again there is a curve $\gamma(t)$ on the $\mathcal{M}$ such that $\gamma(0) = p$, and $\gamma'(0) = v$. But it is not required to be a geodesic curve for a retraction. The parameter update then takes the form

$$w_{t+1} = \mathrm{ret}_{w_t}(-\gamma H(z_t, w_t)). \tag{5.5}$$

Leveraging the smoothness properties of Riemannian manifolds, Bonnabel proves convergence results for RSGD similar to those for SGD, and this for both when the exponential map or a retraction is used.

5.4.3 Retraction for projected cone model

The retraction we used for our experiments with the projected cone model is the same as that used by Nickel et al. for training their Beltrami-Poincaré model. It is simply the Euclidean exponential map, i.e., vector addition: $ret_{w_t}(u) = u + w_t$.

As the unit sphere is dense, this retraction keeps us within the model, so long as the update doesn't lead outside the sphere. This is very unlikely because of the involvement of the Riemannian metric in the update. Nevertheless, we clamp the update to the unit sphere, as

do Nickel et al.

The final form of the parameter update for the projected cone model is as follows. We adjust the gradient update vector according to the Riemannian metric for the projected cone, q.v. equation 4.16, apply the learning rate, and then the chosen retraction. Denoting by $\mathcal{R}_{w_t}$ the metric formula for the projected cone, the update is

$$\begin{aligned} w_{t+1} &= ret_{w_t}(-\gamma \mathcal{R}_{w_t}(H(z_t, w_t))) \\ &= -\gamma \mathcal{R}_{w_t}(H(z_t, w_t)) + w_t. \end{aligned} \tag{5.6}$$

5.5 Training Loop

Algorithms 1 and 2 show pseudocode for standard and Riemannian stochastic gradient descent, respectively. We see that the difference is in the optimization step in which gradients are used to update the parameters (called weights in the pseudocode) of the model. With Riemannian SGD, the gradients must take into account the metric of the mapping from the model to the manifold. As well, the exponential map, or a retraction, must be used to ensure the step along the gradient is brought back onto a suitable near point of the manifold surface. The implementation supports the exploration of learning on different manifolds by allowing different modules to plug in to algorithm 2 by providing implementations for the functions *metricAdjustment* and *exponentialMap*.

Algorithm 1 Standard SGD

```
for epoch from 1 to numEpochs do
    for (inputs, targets) in batches do
        predictions ← forwardPass(inputs)
        loss ← computeLoss(predictions, targets)
        gradients ← backPropGradients(loss)
        weights ← weights − learningRate × gradients
    end for
end for
```

Algorithm 2 Riemannian SGD

1: **for** epoch from 1 to numEpochs **do**
2: **for** (inputs, targets) in batches **do**
3: $predictions \leftarrow forwardPass(inputs)$
4: $loss \leftarrow computeLoss(predictions, targets)$
5: $gradients \leftarrow backPropGradients(loss)$
6:
7: $gradients \leftarrow metricAdjustment(predictions, gradients)$ ▷ start of optimization
8: $gradients \leftarrow learningRate \times gradients$
9: $weights \leftarrow exponentialMap(predictions, gradients)$ ▷ could be a retraction
10: **end for**
11: **end for**

5.5.1 Initialization of vector embeddings

We did not adjust the implementation by Nickel et al. for the initialization of vector embeddings. Our $\mathcal{PC}$ model is based on the unit disk just like the Poincaré model. Initializing our vectors in the same manner as for the Poincaré model makes perfect sense. According to Nickel et al., all embeddings are randomly initialized from the uniform distribution $\mathcal{U}(-0.001; 0.001)$ so as to be close to the origin. A *burn-in* phase is used to help create an initial layout having good angular dispersion. During the *burn-in* phase, the learning rate is reduced so that vectors are forced to disperse in an angular fashion, rather than in radial directions.

5.5.2 Metric update for $\mathcal{PCP}$

Let us comment on how the gradient update works for the $\mathcal{PCP}$ model. In line 7 of algorithm 2, we see that the update to accommodate the metric is applied to the gradient vector.

The metric for the mapping of the 2D cone to the unit disk applies to each component of the product space model independently. Thus, for an n-dimensional model there are $n/2$ gradient updates computed. Each of these is 2-dimensional. The concatenation of these provides an n-dimensional vector update to accomodate the $\mathcal{PCP}$ metric.

5.5.3 Sparse gradient tensor

In order to speed up the training, a sparse tensor implementation of the gradient tensor was undertaken. The gradient computation and update of the weights is more complicated in Riemannian SGD as compared with standard SGD. But the number of gradients updated when processing a single batch is very small. It will depend on the size of the batch and the number of negative samples used. The number of words that will receive a parameter update is therefore much smaller than the size of the vocabulary. We can extract the impacted rows from the sparse gradient tensor and create a very small, dense tensor with which to compute the metric adjustment, weight update, and the exponential map.

Chapter 6

Results

In section 6.1 of this chapter, we present the reconstruction evaluation measure that was used to assess the quality of the learned word representations. In section 6.2, we report the performance of our projected cone and projected cone product models for a variety of embedding dimensions. In section 6.3, we review these results and discuss our opening hypothesis and research questions in light of the results obtained.

6.1 Evaluation

As previously mentioned, the framework created by Nickel et al. [63] includes code for evaluating generated word embeddings. The method establishes rankings reflecting the desirableness of the placement of a noun on the targeted manifold. The mean rank over all nouns is reported as well as the mean average precision of the ranking. We describe this evalutation metric in the next section.

6.1.1 Reconstruction evaluation

The *reconstruction* evaluation strategy considers each hypernymy relation $(n_s, n_g) = (hyponym, hypernym)$ in the data set, $\mathcal{D}$. The subscripts were chosen such that n_s is more *specific* (e.g., primate), as compared to the more *general* noun n_g (e.g., mammal). The evaluation assesses if n_s and n_g are well positioned relative to one another. To achieve this, it ranks the distance between n_s and n_g, $d(n_s, n_g)$, among the distances between n_s and nouns to which it is not related, i.e., the set of all negative gound-truth samples involving n_s as a hyponym: $\{(n_s, n'_g)|(n_s, n'_g) \notin \mathcal{D}\}$. A ranking near 1 is desired.

Recall that we trained on the transitive closure of the hypernymy relationship. So, the direct connections in this graph (with nouns as nodes and edges representing hypernymy) were not available at training time. A good result in the reconstruction ranking indicates that knowledge of the full hierarchy of the hypernymy relation has been effectively learned.

More formally, given a hypernymy relation (n_s, n_g), construct the set

$$\mathcal{O}_{n_s,n_g} = \{d(n_s, n_g)\} \cup \{d(n_s, n'_g)|(n_s, n'_g) \notin \mathcal{D}\}.$$

Treat $\mathcal{O}_{n_s,n_g}$ as an order set of distances, ordered from small to big. The *reconstruction ranking* of the hypernymy relation (n_s, n_g), is the position of $d(n_s, n_g)$ in the ordered set $\mathcal{O}_{n_s,n_g}$. Repeating this for all hypernymy relations and taking the mean gives us the mean rank.

The mean average precision of the ranking is computed using the *average_precision_score*() function from scikit-learn [76]. The average precision metric summarizes a precision-recall curve in a single number. It is a weighted average of the precisions at each threshold, with the increase in recall from the previous thresholds used as the weights [86]:

$$average\_precision = \sum_n (Recall_n - Recall_{n-1}) Precision_n$$

We obtain the mean average precision of the ranking by taking the mean of the average precisions computed for each reconstruction ranking described above.

We note that the reconstruction evaluation selected by Nickel et al. is an instance of *resubstitution error estimation* [39]. This method uses the complete data set for evaluating a model. While this may be somewhat unusual in machine learning, it is a recognized statistical technique and serves our purposes. What it cannot do is provide an accurate evaluation of the generalization ability of the model. However, this is not our goal. The reconstruction evaluation provides us with an assessment of how well the models have learned the latent hierarchies in the data set. This approach is indeed common for evaluating graph embeddings [62].

6.2 Results

We once again refer to the projected cone model as the $\mathcal{PC}$ model, and to the projected cone product model as the $\mathcal{PCP}$ model. We discuss the results of the following tables and figures in the next section. We ran our experiments for $\mathcal{PC}$ and $\mathcal{PCP}$ five times. The full details of each run are provided in appendix A. In the tables below, we show the best results obtained for our models. The results for the Euclidean and Poincaré models are from Nickel et al. [62].

Note: For the mean rank, lower is better. For mean average precision, higher is better.

Tables 6.1 and 6.2 show the mean rank and the mean average precision of embeddings for all of the models at various embedding dimensions.

Table 6.3 reports the training times for a single epoch for the $\mathcal{PC}$, $\mathcal{PCP}$, and Poincaré models. The average epoch training time over ten epochs was used. As well, we compare the training times for both $\mathcal{PCP}$ and Poincaré to that of $\mathcal{PC}$.

	5/6[1]	**10**	**20**	**50**	**100**	**200**
Euclidean[2]	3542.3	2286.9	1685.9	1281.7	1187.3	1157.3
Poincaré[2]	4.90	4.02	3.84	3.98	3.90	3.83
$\mathcal{PC}$	269.06	133.73	74.07	35.71	19.29	12.35
$\mathcal{PCP}$	246.24	109.99	54.45	32.01	25.61	26.18

Table 6.1: Mean rank by dimension for each model.

	5/6[1]	**10**	**20**	**50**	**100**	**200**
Euclidean[2]	0.024	0.059	0.087	0.140	0.162	0.168
Poincaré[2]	0.823	0.851	0.855	0.860	0.857	0.870
$\mathcal{PC}$	0.29	0.34	0.43	0.57	0.66	0.74
$\mathcal{PCP}$	0.45	0.57	0.66	0.73	0.75	0.76

Table 6.2: Mean average precision by dimension for each model.

Dimension	$\mathcal{PC}$	$\mathcal{PCP}$	**Poincaré**	$\mathcal{PC}$ **: Poincaré**	$\mathcal{PCP}$ **:** $\mathcal{PC}$
5/6	25.4	22.4	20.3	1.26	0.88
10	25.5	24.7	20.1	1.27	0.97
20	26.1	31.7	20.2	1.29	1.21
50	26.7	54.9	20.5	1.30	2.05
100	29.6	94.8	20.7	1.43	3.20
200	34.2	117.7	21.6	1.58	5.20

Table 6.3: Training time for an epoch, in seconds.

[1]The cone product model is a repeated product of two dimensional spaces and must therefore be of even dimension. We have used 6 dimensions, as 5 is not possible. All other models in this column are 5 dimensional.

[2]These results are from Nickel et al. [62].

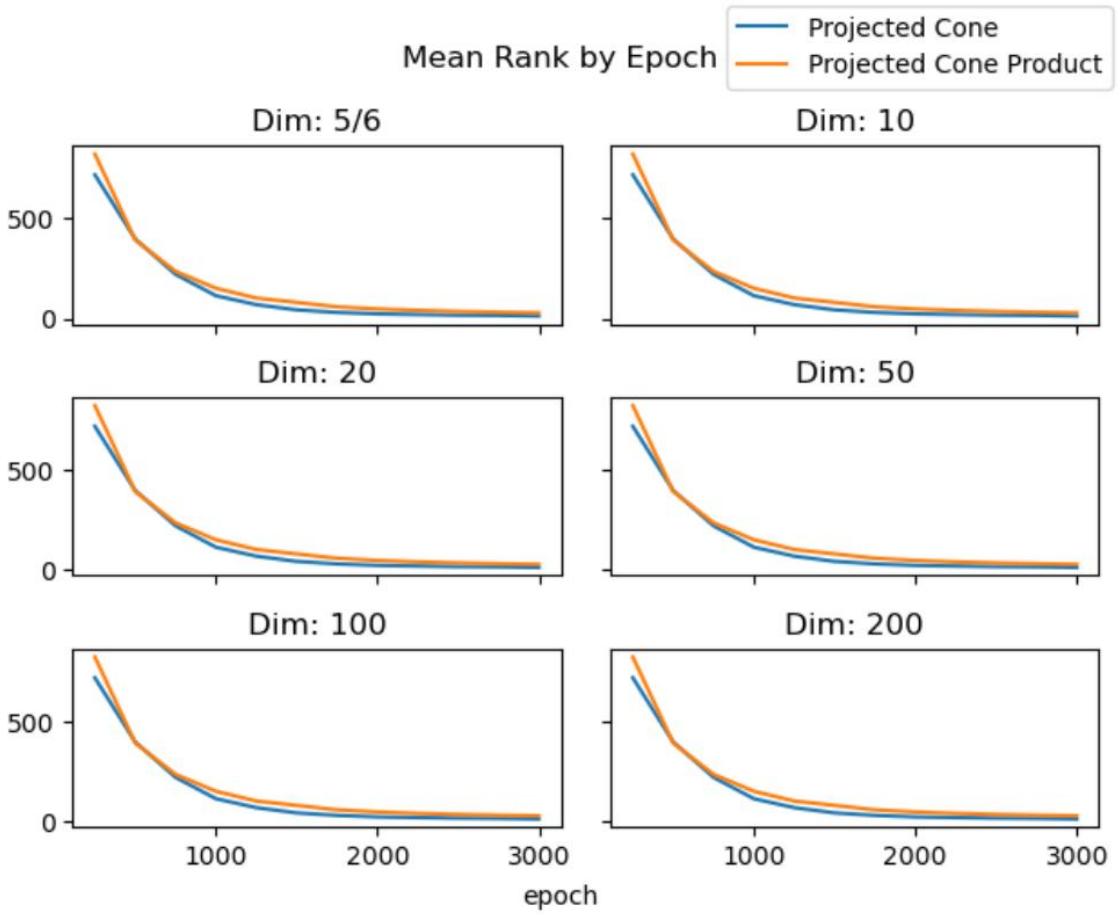


Figure 6.1: Mean ranks while training the $\mathcal{PC}$ and $\mathcal{PCP}$ models.

Graphs of the mean rank by epoch and mean average precision by epoch for our $\mathcal{PC}$ and $\mathcal{PCP}$ models are shown in Figures 6.1 and 6.2, respectively.

Table 6.4 shows the mean squared norm of embedding vectors for the $\mathcal{PC}$ and Poincaré models at 200 dimensions. Euclidean and $\mathcal{PCP}$ embeddings are not included as they are not constrained to the unit ball and so comparisons with those models are not applicable. The **Euclidean** column is the norm of the vector as measured in Euclidean space, and thus is constrained to the unit ball. The **Model** column is the measurement made using the model's own distance metric.

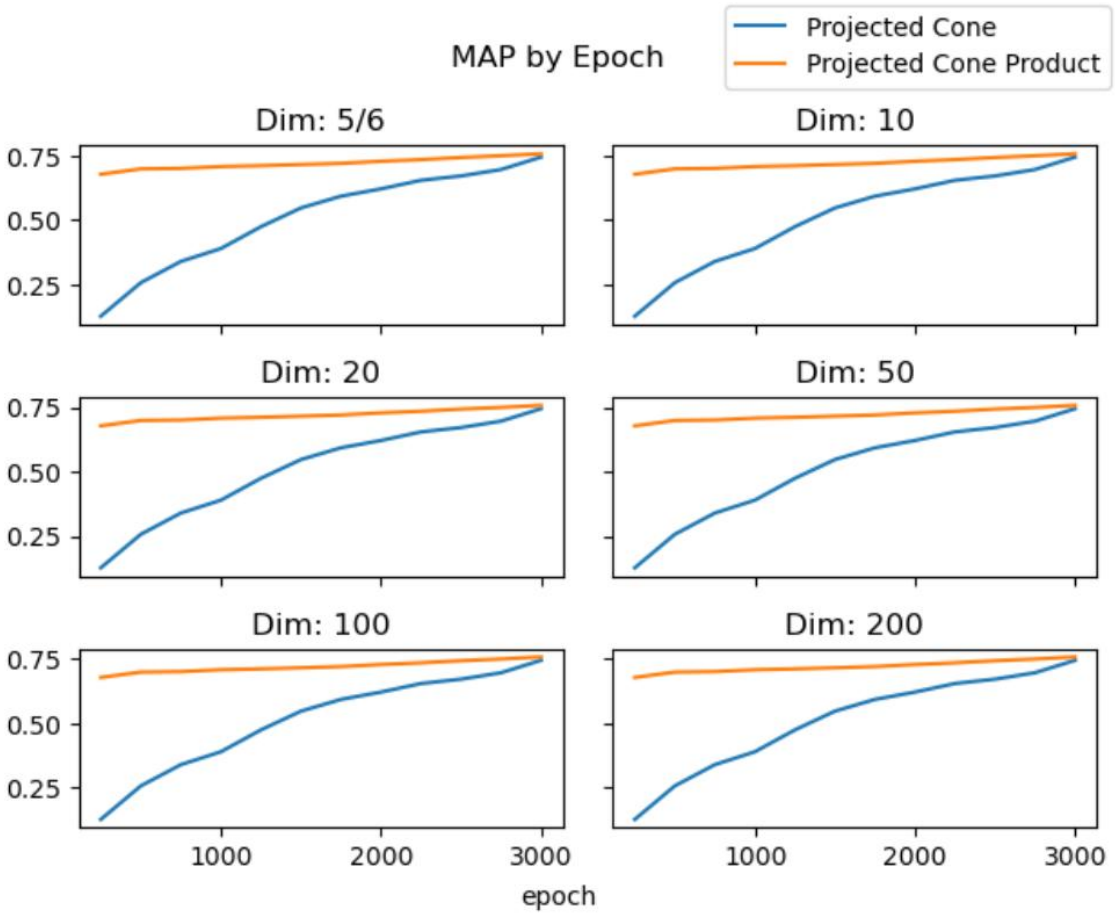


Figure 6.2: Mean average precision of ranking while training the $\mathcal{PC}$ and $\mathcal{PCP}$ models.

	Mean Squared Norm	
	Euclidean	**Model**
$\mathcal{PC}$	0.952181	28.16
Poincaré	0.999821	9.32

Table 6.4: Mean squared norm of trained models at 200 dimensions.

6.3 Discussion

We begin by reviewing the results of the experiments. In the following subsection, we discuss our opening hypothesis and research questions.

6.3.1 Review of results

From Figure 6.1, we observe that as training progressed, the mean rank dropped nearly monotonically. This indicates that training was progressing well and in a stable fashion. Performance mostly levelled out after epoch 1500, with only mild gains thereafter. Figure 6.2 tells a similar story for mean average precision, though in some cases it appears that continued training may have led to further improvements for mean average precision as a few of the curves do not appear to have levelled off. Nevertheless, given that the mean rank evaluation had stabilised, we did not continue to train beyond 3000 epochs.

Tables 6.1 and 6.2 report the mean rank and the mean average precision of all models, for each dimensionality trained. These dimensions correspond to those used by Nickel et al. to enable us to compare our results to theirs.

These two tables show that both of our cone models significantly outperform the Euclidean baseline. However, neither is quite as good as the Poincaré-based results from Nickel et al. That being said, the cone models are much closer to the Poincaré-based results than to the Euclidean ones. For example, the $\mathcal{PC}$ model is 3.35 times worse than the Poincaré model, but it is 90 times better than the Euclidean model.

An interesting pattern in the mean rank results is that the Poincaré model is extremely efficient at very low dimensions. While the cone models still greatly outperform the Euclidean model at these lower dimensions, the gap between the cone models and the Poincaré model is larger in these cases.

Continuing with mean rank results, and focusing on the two cone models, we observe that the $\mathcal{PCP}$ model outperforms the $\mathcal{PC}$ model slightly at lower dimensions. As the dimensionality increases, the $\mathcal{PC}$ model is the better of the two. At 200 dimensions, $\mathcal{PC}$ is more that twice as good as $\mathcal{PCP}$. The reason for this is unclear. Earlier, in section 4.10.1, we provided a rough argument that the model capacities of $\mathcal{PC}$ and $\mathcal{PCP}$ are of the same order. It is possible that the constants we brushed over in that argument are not entirely insignificant. This remains an open question.

We now turn our attention to the mean average precision results. Again, the two cone models greatly outperform the Euclidean baseline, while falling short of the Poincaré model. The Poincaré model demonstrates good performance even at low dimensions. Our cone models close the gap considerably as the model dimension increases.

For mean average precision, the $\mathcal{PCP}$ model outperforms the $\mathcal{PC}$ model at all dimensions. However, the gap is all but closed at 200 dimensions. This is similar to what we observed for mean ranks, where we saw $\mathcal{PCP}$ outperform $\mathcal{PC}$ at lower dimensions. But with mean rank, $\mathcal{PC}$ ended up overtaking $\mathcal{PCP}$, whereas with mean average precision, it only caught up.

All in all, it is fair to conclude the cone models did very well, but not as well as the Poincaré model. We may consider the $\mathcal{PC}$ model as superior to the $\mathcal{PCP}$ model as it did better at 200 dimensions. In practice, an embedding dimension of 200 is still considered quite small.

6.3.2 Addressing of hypothesis and research questions

At the outset of this work, we hypothesized that projection plays a key role in enabling the benefits of hyperbolic geometry for representation learning. We argued in section 4.1 that our $\mathcal{PC}$ model leverages nothing but projection in its construction. We proposed to investigate our hypothesis by investigating the performance of our $\mathcal{PC}$ model relative to the

Poincaré model and the Euclidean baseline model.

The results we have presented indicate a positive outcome to our investigation as both the $\mathcal{PC}$ and $\mathcal{PCP}$ models significantly outperform the Euclidean model. While there may be other factors at play for explaining the far superior performance of the Poincaré model over the Euclidean model for respresentation learning, we have provided good evidence to back our hypothesis.

The reason we suspected projection was a critical factor has to due with the capacity arguments we presented. It is projection that maps the infinite surface of the hyperboloid into the unit disk of the Beltrami-Poincaré model of hyperbolic geometry. Thus, there must be ever-increasing area as we move toward the edge of the disk. The same applies to the projection of the cone to the unit disk. Any ring in the disk with its outer edge at the boundary circle of radius 1 must have infinite area. Our capacity argument has been validated by experiment.

We now revisit the research questions that were raised at the beginning of this work.

> **Research Question 1**: Can a non-Euclidean model that is *simpler* than hyperbolic geometry in terms of computational complexity be used effectively for representation learning?

The $\mathcal{PC}$ model is certainly *not* Euclidean, though it does not correspond to the usual notion of a non-Euclidean space because it does not exhibit *constant* curvature. Not withstanding this terminological nitpick, it does seem that the projected cone model can be used effectively for representation learning, certainly as compared to the Euclidean regime. As well, it is quite likely that our results for the cone models could be improved upon with continued research.

Is the projected cone model *simpler* than the Beltrami-Poincaré model? It may seem so as it is constructed by the projection of a simpler surface, namely the cone versus the hyperboloid. Indeed, to a large degree this observation is what motivated our research into the projected cone model. But this isn't necessarily a good way to characterize the simplicity of the model. It is more reasonable to compare the mathematical formulas involved or implementation

complexity in order to judge the simplicity of the model. We can also compare the training time required for each model. This is a very pragmatic measure of comparison, and it captures the complexity of the formulas computed. We take this approach.

Table 6.3 shows the time required to train a single epoch for the $\mathcal{PC}$, $\mathcal{PCP}$, and Beltrami-Poincaré models. The numbers are the average training time over 10 epochs. As it turns out, the training time for the $\mathcal{PC}$ model is longer than for Beltrami-Poincaré. The second last column shows the ratio of these. The average value of this last columns is 1.36. (While we are commenting on training times, we also note that the training time for the $\mathcal{PCP}$ model is high and that it increases quickly with added dimensions.)

We conclude that our $\mathcal{PC}$ model is somewhat *less* simple than the Beltrami-Poincaré model. This is an unanticipated outcome of our research.

> **Research Question 2**: What is the learning capacity of the projected cone model, and how does it compare to the Beltrami-Poincaré model of hyperbolic geometry?

We compared the capacities of the $\mathcal{PC}$ and Beltrami-Poincaré models in section 4.4. We found that $\mathcal{PC}$ has the greater capacity in the sense that the model space grows more quickly as a function of the distance from the origin. In table 6.4, we see that the mean squared norm of embedding vectors for the converged $\mathcal{PC}$ model is 0.952181, while it is 0.999821 for the Poincaré model. In the model's own space, these correspond to mean norms of 28.16 for $\mathcal{PC}$ and 9.32 for Poincaré. Thus the embedding vectors are on average longer in the converged $\mathcal{PC}$ model than in the Poincaré model.

These arguments lend support to the conclusion that the $\mathcal{PC}$ model has higher learning capacity than the Beltrami-Poincaré model. But then why did the $\mathcal{PC}$ model not match or exceed the performance of the Beltrami-Poincaré model? Perhaps the radial distance from the origin in the $\mathcal{PC}$ model increases so rapidly that the computing resolution available is unable to cope. All experiments (including those by Nickel et al.) use double-precision IEEE floating point numbers. These offer 15 to 17 significant decimal digits of precision [1]. If the resolution of double-precision floating point numbers is insufficient, the adjustment of vector representations during backpropagation updates may be somewhat erratic in terms

of $\mathcal{PC}$ distances (i.e., small Euclidean updates correspond to large changes according to the model's metric). This is clearly speculation. On the other hand, perhaps given more time and resources a better job could have been done training the $\mathcal{PC}$ model. Determining the true reason why our results with $\mathcal{PC}$ do not match those obtained by Nickel et al. for the Beltrami-Poincaré model remains an open question.

> **Research Question 3**: What are the mathematical properties of the projected cone model that need to be understood in order to implement representation learning with it?

In chapter 4, *The Projected Cone Model*, we determined these required properties. There, we found formulas for the lift and project operators, discussed geodesics of the cone surface, derived a precise formula for the distance between points in the projected cone model, and derived the Riemannian metric of the mapping from the cone into the space of the projected cone model.

> **Research Question 4**: How can the projected cone model be extended to higher dimensions?

Once again, in chapter 4, *The Projected Cone Model*, we addressed the generalization of the projected cone model to higher dimensions. We provided intuitive arguments for our approach. However, intuition may be an unreliable guide when contemplating high dimensions. We therefore also proposed the projected cone product model as a baseline. Vectors in this model belong to a product space consisting of some number of repetitions of our well-undestood, two dimensional projected cone model.

Our results show that the approach worked reasonably well. It is true that $\mathcal{PC}$ did not do quite as well as Poincaré. Could this be because of an error in how the $\mathcal{PC}$ model was extended into higher dimensions? It may be, but given that the $\mathcal{PC}$ model's performance continued to improve as dimensionality increase (see tables 6.1 and 6.2), this doesn't appear to be the case. As well, for mean rank, we saw the $\mathcal{PC}$ model catch up and out pace the well-founded $\mathcal{PCP}$ baseline model as dimensionality was increased, thus providing more evidence the the $\mathcal{PC}$ model is on a stable footing. That being said, it may very well be the case that refinements are possible to the method of extension we developed.

Research Question 5: Is the projected cone model effective for representation learning?

Our results show that, indeed, the $\mathcal{PC}$ model has shown itself to be effective for representation learning. While we did not quite achieve the results reported for the Beltrami-Poincaré model by Nickel et al., we significantly outperformed the Euclidean model. In the next chapter, we will indicate avenues for future research that are aimed at explaining the discrepancy between the Beltrami-Poincaré and $\mathcal{PC}$ models.

6.3.3 Applying manifold learning in other contexts

In principle, manifold-based learning can be applied to any machine learning model, see Bonnabel for some excellent examples [9]. The key difference is as described in algorithm 2 of section 5.5. When using manifold-based learning, there are certain considerations to attend to. For example, many networks use residual connections and some form of normalization (e.g., batch normalization, or layer normalization). These may not work as intended with manifold-based learning. For example, a residual connection is implemented as simple vector addition in Euclidean-based models. This will likely need to be adapted with a suitable addition operator when working on a non-Euclidean manifold. Likewise, the assumptions used to justify normalization techniques will also likely need to be revisitied. In section 2.4, we mentioned approaches taken by others to leverage hyperbolic geometry in deep learning models such as the transformer model.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The main goal of this work has been to investigate the reason for the performance improvements that have been achieved by computing embeddings in hyperbolic space rather than in Euclidean space. Our hypothesis was that projection, a key component of hyperbolic geometry, is a critical factor. We introduced an alternative, non-Euclidean, projective model based on a very simple shape, the cone, to support this hypothesis.

By computing embeddings in our project cone model, we have provided evidence that other properties of hyperbolic geometry are not as critical as projection for achieving quality embeddings. We discussed differences between our model and the Beltrami-Poincaré model of Nickel et al. Our space does not possess constant negative curvature. The projective mapping for our model is not conformal, and thus the isometries for our model are not as rich. We also argued that projection was responsible for explaining the high capacity of the projected cone model and the Beltrami-Poincaré model.

We raised and addressed a number of research questions related to the investigation of our hypothesis. To begin with, we defined our proposed projected cone model, studied its properties, extended it to high dimensions, and proposed a high-dimensional baseline to

ensure our extension was sound. Next, we designed and conducted experiments to assess performance of our two projected cone models, and compare them to results obtained by others in Euclidean space and in the Beltrami-Poincaré model of hyperbolic space. We were able to conclude that a non-Euclidean model that is not based on hyperbolic space can be effective for machine learning. While our results did not match those of the Beltrami-Poincaré model, we came close, and we greatly outperformed embeddings in Euclidean space.

We contrasted the complexity of our model with that of the Beltrami-Poincaré model. Based on computation efficiency, we found our projected cone model was more complex than the Beltrami-Poincaré, with our training time per epoch being 1.36 times that of the Beltrami-Poincaré model on average.

7.2 Future Work

We have observed that the Beltrami-Poincaré model outperformed our projected cone models. Compared to our gains relative to the Euclidean baseline, the difference between the Beltrami-Poincaré and our projected cone models is rather small. It is still interesting to ask what may be the cause for the difference. One consideration is the conformality of the projective mapping. We indicated that our cone mapping is not conformal, but the Beltrami-Poincaré projection of the hyperboloid is known to be. Conformality is a desirable property. It means that at the local level, i.e., in the small, similar shapes are mapped to similar shapes. We can see this as an instance of equivariance in the spirit of Geometric Deep Learning, which we discussed in section 2.5. Without conformality, a shape is distorted by a mapping. This could be a factor in explaining the better performance of the Beltrami-Poincaré model over the projected cone models. More research would be required.

We have noted that the isometries of hyperbolic space include spherical inversion. This is a key difference between hyperbolic space and Euclidean space. We may ask ourselves what role, if any, does spherical inversion have in the performance improvement of hyperbolic geometry over Euclidean geometry in representation learning? Just as with the projective mappings of the Beltrami-Poincaré and projected cone models, an effect of spherical inversion is to pack infinite space into a bounded space. Consider circular inversion in the unit circle

centered at the origin of the Euclidean two-dimensional plane. All of the space exterior to the circle is mapped to its interior, and vice versa. Surely there is high capacity induced into the unit disk after performing inversion.

Indeed, we experimented with this possibility. Applying the same framework we used thoughout this work, we implemented manifold-learning via Riemannian stochastic gradient descent for the mapping of spherical inversion. Initialization of vectors was more challening, as vectors of small norm are no longer a good choice. We found that the learning process quickly diverged. Deeper research into this may yield more insighful results. Certainly, this would be a very *simple* model implementation-wise. The formulae involved would be simple and fast to compute.

Another model of hyperbolic geometry has recently been proposed by Norman Wildberger. He calls it *Universal Hyperbolic Geometry* [89–91]. An interesting aspect of universal hyperbolic geometry (UHG) is that the distance formula[1] does not involve transcendental functions such as inverse hyperbolic cosine. All UHG formulas are polynomials, quadratics in fact. To achieve this the UHG model of hyperbolic geometry does not confine itself to the unit disk. The model encompasses all points of the plane. Nevertheless, projective geometry and spherical inversion comprise the foundation of UHG. Other nice properties of UHG are that triangles are better behaved than in the conventional models of hyperbolic geometry that were developed in the 19^{th} century. Once again, in the spirit of Geometric Deep Learning, we can consider the equivariance of the centers of triangles, namely centroids, orthocenters, and circumcenters. In general, these are not preserved by the usual mappings of hyperbolic geometry. For example, the orthocenter of a triangle on the hyperboloid does not, in general, map to the orthocenter of the mapped triangle in the Beltrami-Poincaré model. In UHG, such triangle properties are preserved. Thus, in the same way that conformality of a mapping is an important property, preservation of this triangle geometry may also benefit representation learning. How to extend UHG to higher dimensions, and determining all that is needed to enable machine learning with it (e.g., the metric of the mapping, embedding vector initialization) are topics for further research.

[1] In fact, N. Wildberger avoids the notions of distance and angle, arguing that they are the wrong notions for doing geometry. He introduces the notions of *quadrature*, a measure of the separation of points, and *spread*, a measure of the separation of lines, in their stead [88].

Finally, the models we have been using are shallow networks. It would be interesting to extend our investigations to the deep, transformer-based models which are now dominant.

Appendix A

Detailed Results of Five Runs

A.1 Mean Rank $\mathcal{PC}$

run #	5	10	20	50	100	200
run 1	277.71	133.73	81.14	37.27	21.41	12.86
run 2	282.68	140.00	77.54	35.71	22.36	15.73
run 3	269.39	141.17	74.07	36.14	19.29	13.99
run 4	269.06	139.83	75.78	40.32	20.17	14.54
run 5	276.64	141.32	78.55	37.20	21.34	12.35
Mean	275.10	139.21	77.42	37.33	20.91	13.89
Std. Dev.	5.21	2.80	2.41	1.61	1.07	1.20

A.2 Mean Rank $\mathcal{PCP}$

run #	6	10	20	50	100	200
run 1	256.60	109.99	58.91	32.01	25.61	28.96
run 2	254.05	111.94	57.04	32.93	26.64	26.18
run 3	250.04	119.29	54.45	36.08	31.57	26.02
run 4	246.24	140.37	59.19	33.56	30.19	27.62
run 5	260.25	136.33	57.00	35.23	30.48	27.78
Mean	253.44	123.58	57.32	33.96	28.9	27.31
Std. Dev.	4.90	12.51	1.70	1.49	2.33	1.09

A.3 MAP Rank $\mathcal{PC}$

run #	5	10	20	50	100	200
run 1	0.29	0.33	0.42	0.56	0.65	0.74
run 2	0.28	0.33	0.43	0.57	0.66	0.70
run 3	0.28	0.34	0.43	0.56	0.66	0.68
run 4	0.26	0.34	0.43	0.55	0.66	0.72
run 5	0.28	0.33	0.42	0.55	0.66	0.71
Mean	0.28	0.33	0.43	0.56	0.66	0.71
Std. Dev.	0.01	0.0	0.0	0.01	0.0	0.02

A.4 MAP Rank $\mathcal{PCP}$

run #	6	10	20	50	100	200
run 1	0.44	0.55	0.65	0.72	0.75	0.76
run 2	0.44	0.57	0.66	0.73	0.74	0.76
run 3	0.44	0.55	0.65	0.72	0.74	0.74
run 4	0.45	0.55	0.65	0.71	0.75	0.75
run 5	0.45	0.56	0.66	0.71	0.74	0.76
Mean	0.44	0.56	0.65	0.72	0.74	0.75
Std. Dev.	0.00	0.01	0.00	0.01	0.00	0.01

Bibliography

[1] IEEE standard for floating-point arithmetic. Standard IEEE Std 754-2008, IEEE Computer Society, New York, NY, USA, August 2008. URL: https://web.archive.org/web/20160806053349/http://www.csee.umbc.edu/~tsimo1/CMSC455/IEEE-754-2008.pdf.

[2] Edwin A Abbott. *Flatland: A Romance of Many Dimensions*. Dover Publications Inc., 1992.

[3] Ivana Balažević, Carl Allen, and Timothy Hospedales. *Multi-Relational Poincaré Graph Embeddings*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[4] Gary Becigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=r1eiqi09K7.

[5] Eugenio Beltrami. Saggio di interpretazione della geometria non-euclidea. *Giornale di Matematiche*, VI:284–312, 1868.

[6] Eugenio Beltrami. Teoria fondamentale degli spazii di curvatura costante. *Ann. Mat. Pura App.*, II:232–255, 1868.

[7] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012. URL: http://arxiv.org/abs/1206.5538, arXiv:1206.5538.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational*

Linguistics, 5:135–146, 2017. URL: https://aclanthology.org/Q17-1010, doi:10.1162/tacl_a_00051.

[9] Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, sep 2013. URL: https://doi.org/10.1109%2Ftac.2013.2254619, doi:10.1109/tac.2013.2254619.

[10] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 2787–2795, 2013. URL: http://dblp.uni-trier.de/db/conf/nips/nips2013.html#BordesUGWY13.

[11] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. arXiv:2104.13478.

[12] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning, 2022. URL: https://geometricdeeplearning.com/.

[13] Ruben Cartuyvels, Graham Spinks, and Marie-Francine Moens. Discrete and continuous representations and processing in deep learning: Looking forward. *CoRR*, abs/2201.01233, 2022. URL: https://arxiv.org/abs/2201.01233, arXiv:2201.01233.

[14] Benjamin Paul Chamberlain, Stephen R. Hardwick, David R. Wardrope, Fabon Dzogang, Fabio Daolio, and Saúl Vargas. Scalable hyperbolic recommender systems, 2019. arXiv:1902.08648.

[15] Ines Chami, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15065–15076. Curran Associates, Inc., 2020. URL: https://proceedings.neurips.cc/paper/2020/file/ac10ec1ace51b2d973cd87973a98d3ab-Paper.pdf.

[16] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/0415740eaa4d9decbc8da001d3fd805f-Paper.pdf.

[17] Jiaao Chen, Dinghan Shen, Weizhu Chen, and Diyi Yang. HiddenCut: Simple data augmentation for natural language understanding with better generalizability. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4380–4390, Online, August 2021. Association for Computational Linguistics. URL: https://aclanthology.org/2021.acl-long.338, doi:10.18653/v1/2021.acl-long.338.

[18] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020. URL: https://proceedings.mlr.press/v119/chen20j.html.

[19] Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5672–5686, Dublin, Ireland, May 2022. Association for Computational Linguistics. URL: https://aclanthology.org/2022.acl-long.389, doi:10.18653/v1/2022.acl-long.389.

[20] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science (JASIS)*, 41(6):391–407, 1990.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019. doi:10.18653/v1/n19-1423.

[22] Jiarui Ding and Aviv Regev. Deep generative model embedding of single-cell rna-seq profiles on hyperspheres and hyperbolic spaces. *Nature Communications*, 12, 05 2021. doi:10.1038/s41467-021-22851-4.

[23] Jeff Donahue and Karen Simonyan. *Large Scale Adversarial Representation Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[24] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/378a063b8fdb1db941e34f4bde584c7d-Paper.pdf.

[25] Euclid. *The Elements of Euclid*. Dover Publications Inc., New York, 2nd edition, 1956. URL: https://archive.org/details/euclid_heath_2nd_ed.

[26] Howard W Eves. *An Introduction to the History of Mathematics*. Saunders college publishing, 6th edition, 1990.

[27] Hongchao Fang, Sicheng Wang, Meng Zhou, Jiayuan Ding, and Pengtao Xie. Cert: Contrastive self-supervised learning for language understanding. *ArXiv*, abs/2005.12766, 2020.

[28] Octavian Ganea, Gary Becigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1646–1655. PMLR, 10–15 Jul 2018. URL: https://proceedings.mlr.press/v80/ganea18a.html.

[29] Octavian Ganea, Gary Becigneul, and Thomas Hofmann. Hyperbolic neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett,

editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/dbab2adc8f9d078009ee3fa810bea142-Paper.pdf.

[30] C.F. Gauss. *General Investigations of Curved Surfaces of 1827 and 1825*. General Investigations of Curved Surfaces of 1827 and 1825. Princeton university library, 1902.

[31] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[32] M. Gromov. *Hyperbolic Groups*, pages 75–263. Springer New York, New York, NY, 1987. doi:10.1007/978-1-4613-9586-7_3.

[33] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=rJxHsjRqFQ.

[34] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y.W. Teh and M. Titterington, editors, *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 9 of *JMLR W&CP*, pages 297–304, 2010.

[35] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012. URL: http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#GutmannH12.

[36] Zellig Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954. URL: https://link.springer.com/chapter/10.1007/978-94-009-8467-7_1, doi:10.1007/978-94-009-8467-7_1.

[37] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] Alan Julian Izenman. *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[40] Hiroyuki Kasai, Pratik Jawanpuria, and Bamdev Mishra. Adaptive stochastic gradient algorithms on riemannian manifolds. *ArXiv*, abs/1902.01144, 2019.

[41] Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial self-supervised contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2983–2994. Curran Associates, Inc., 2020. URL: `https://proceedings.neurips.cc/paper_files/paper/2020/file/1f1baa5b8edac74eb4eaa329f14a0361-Paper.pdf`.

[42] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL: `http://arxiv.org/abs/1312.6114`.

[43] Felix C. Klein. A comparative review of recent researches in geometry, 2008. `arXiv:0807.3161`.

[44] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*. OpenReview.net, 2020. URL: `http://dblp.uni-trier.de/db/conf/iclr/iclr2020.html#LanCGGSS20`.

[45] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020. URL: `https://proceedings.mlr.press/v119/laskin20a.html`.

[46] Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, volume R5 of *Proceedings of Machine Learning Research*, pages 206–213. PMLR, 06–08 Jan 2005. URL: https://proceedings.mlr.press/r5/lecun05a.html.

[47] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL: http://arxiv.org/abs/1907.11692, arXiv:1907.11692.

[48] Zhe Liu and Yibin Xu. Thg: Transformer with hyperbolic geometry, 06 2021.

[49] Zhiyuan Liu, Yankai Lin, and Maosong Sun. *Representation Learning for Natural Language Processing*. Springer, 2020. doi:10.1007/978-981-15-5573-2.

[50] Brice Loustau. *Hyperbolic geometry*. 2021. arXiv:2003.11180.

[51] Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996. URL: http://link.springer.com/article/10.3758/BF03204766, doi:10.3758/BF03204766.

[52] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *CoRR*, abs/1708.00107, 2017. URL: http://arxiv.org/abs/1708.00107, arXiv:1708.00107.

[53] Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61, Berlin, Germany, August 2016. Association for Computational Linguistics. URL: https://aclanthology.org/K16-1006, doi:10.18653/v1/K16-1006.

[54] Deshui Miao, Jiaqi Zhang, Wenbo Xie, Jian Song, Xin Li, Lijuan Jia, and Ning Guo. Simple contrastive representation adversarial learning for NLP tasks. *CoRR*,

abs/2111.13301, 2021. URL: https://arxiv.org/abs/2111.13301, arXiv:2111.13301.

[55] Tomas Mikolov, Kai Chen, G S Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[56] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[57] George A. Miller. WordNet: A lexical database for English. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. URL: https://aclanthology.org/H94-1111.

[58] Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 714–722, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3292500.3330997.

[59] Usman Naseem, Imran Razzak, Shah Khalid Khan, and Mukesh Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models, 2020. arXiv:2010.15036.

[60] T. Needham. *Visual Complex Analysis*. Clarendon Press, 1997.

[61] T. Needham. *Visual Differential Geometry and Forms: A Mathematical Drama in Five Acts*. Princeton University Press, 2021.

[62] Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6341–6350. Curran Associates, Inc., 2017. URL: http://papers.n

ips.cc/paper/7213-poincare-embeddings-for-learning-hierarchical-representations.pdf.

[63] Maximilian Nickel and Douwe Kiela. facebookresearch/poincare-embeddings: PyTorch implementation of the NIPS-17 paper "Poincaré Embeddings for Learning Hierarchical Representations". https://github.com/facebookresearch/poincare-embeddings, 2017–2021 (accessed March 23, 2023).

[64] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

[65] Ernesto Paas-Oliveros, Enrique Hernández-Lemus, and Guillermo de Anda-Jáuregui. Computational single cell oncology: state of the art. *Frontiers in Genetics*, 14, 2023. URL: https://www.frontiersin.org/articles/10.3389/fgene.2023.1256991, doi:10.3389/fgene.2023.1256991.

[66] W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao. Hyperbolic deep neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(12):10023–10044, dec 2022. doi:10.1109/TPAMI.2021.3136921.

[67] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. URL: https://aclanthology.org/D14-1162, doi:10.3115/v1/D14-1162.

[68] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. URL: http://arxiv.org/abs/1802.05365, arXiv:1802.05365.

[69] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016. URL: http://arxiv.org/abs/1612.00593, arXiv:1612.00593.

[70] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[71] Daniela N. Rim, DongNyeong Heo, and Heeyoul Choi. Adversarial training with contrastive learning in NLP. *CoRR*, abs/2109.09075, 2021. URL: `https://arxiv.org/abs/2109.09075`, `arXiv:2109.09075`.

[72] Joshua David Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: `https://openreview.net/forum?id=CR1XOQ0UTh-`.

[73] Douglas L. T. Rohde, Laura M. Gonnerman, and David C. Plaut. An improved model of semantic similarity based on lexical co-occurence. 2006.

[74] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[75] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[76] scikit-learn contributors. sklearn.metrics.average_precision_score, 2007 - 2023. [Online; accessed 2023-05-10]. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html`.

[77] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany, August 2016. Association for Computational Linguistics. URL: `https://aclanthology.org/P16-1009`, `doi:10.18653/v1/P16-1009`.

[78] Dinghan Shen, Ming Zheng, Yelong Shen, Yanru Qu, and Weizhu Chen. A simple but tough-to-beat data augmentation approach for natural language understanding and generation. *ArXiv*, abs/2009.13818, 2020.

[79] Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association for*

Computational Linguistics (ACL), pages 354–362, Ann Arbor, Michigan, June 2005. URL: http://cs.jhu.edu/~jason/papers/#acl05.

[80] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf.

[81] Jun Takeuchi, Noriki Nishida, and Hideki Nakayama. Neural networks in a product of hyperbolic spaces. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*, pages 211–221, Hybrid: Seattle, Washington + Online, July 2022. Association for Computational Linguistics. URL: https://aclanthology.org/2022.naacl-srw.27, doi:10.18653/v1/2022.naacl-srw.27.

[82] Lucas Vinh Tran, Yi Tay, Shuai Zhang, Gao Cong, and Xiaoli Li. Hyperml: A boosting metric learning approach in hyperbolic space for recommender systems, 2019. arXiv:1809.01703.

[83] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL: http://arxiv.org/abs/1807.03748.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[85] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388,

Hong Kong, China, November 2019. Association for Computational Linguistics. URL: https://aclanthology.org/D19-1670, doi:10.18653/v1/D19-1670.

[86] Wikipedia contributors. Average precision, 2017. [Online; accessed 2023-05-10]. URL: https://en.wikipedia.org/w/index.php?title=Information_retrieval&oldid=793358396#Average_precision.

[87] Wikipedia contributors. Volume of an n-ball, 2022. [Online; accessed 2023-08-04]. URL: https://en.wikipedia.org/wiki/Volume_of_an_n-ball.

[88] Norman Wildberger. *Divine Proportions: Rational Trigonometry to Universal Geometry*. Wild Egg, 2005.

[89] Norman Wildberger. Universal hyperbolic geometry i: Trigonometry. *Geometriae Dedicata*, 163, 09 2009. doi:10.1007/s10711-012-9746-9.

[90] Norman Wildberger. Universal hyperbolic geometry ii: A pictorial overview. *KoG*, 14, 12 2010.

[91] Norman Wildberger. Universal hyperbolic geometry. iii: First steps in projective triangle geometry. *KoG*, 01 2011.

[92] Xing Wu, Shangwen Lv, Liangjun Zang, Jizhong Han, and Songlin Hu. Conditional bert contextual augmentation. In *International Conference on Conceptual Structures*, 2018.

[93] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

[94] Yuansheng Zhou and Tatyana Sharpee. Hyperbolic geometry of gene expression. *iScience*, 24:102225, 02 2021. doi:10.1016/j.isci.2021.102225.

[95] Yudong Zhu, Di Zhou, Jinghui Xiao, Xin Jiang, Xiao Chen, and Qun Liu. HyperText: Endowing FastText with hyperbolic geometry. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1166–1171, Online, November 2020. Association for Computational Linguistics. URL: https://aclanthology.org/2020.findings-emnlp.104, doi:10.18653/v1/2020.findings-emnlp.104.