FORNAX

Poincaré Embeddings for Learning Hierarchical Representations

Krzysztof Kolasiński, 5.09.2017

WWW.FORNAX.AI

Content

- Introduction with word embeddings
- What's wrong with w2v?
- The need of hyperbolic spaces
- Understanding Poincare embeddings

Disclaimer

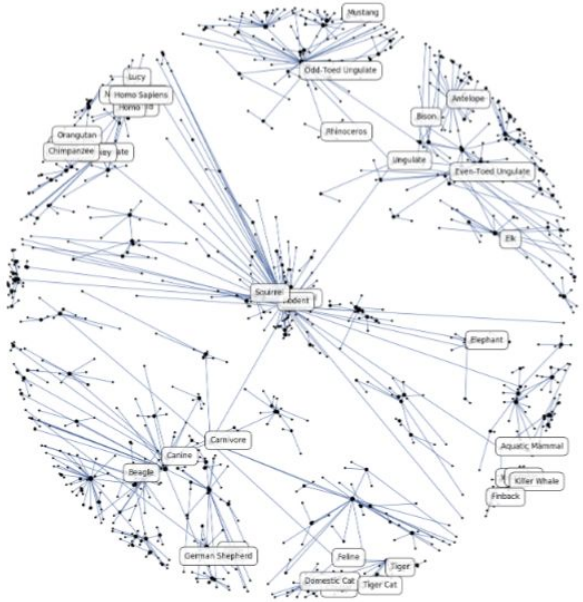
- Not expert in Riemannian optimization methods
- Quite hard mathematics behind
- My point of view on some problems, not stated in literature

Poincaré Embeddings for Learning Hierarchical Representations (22 May 2017)

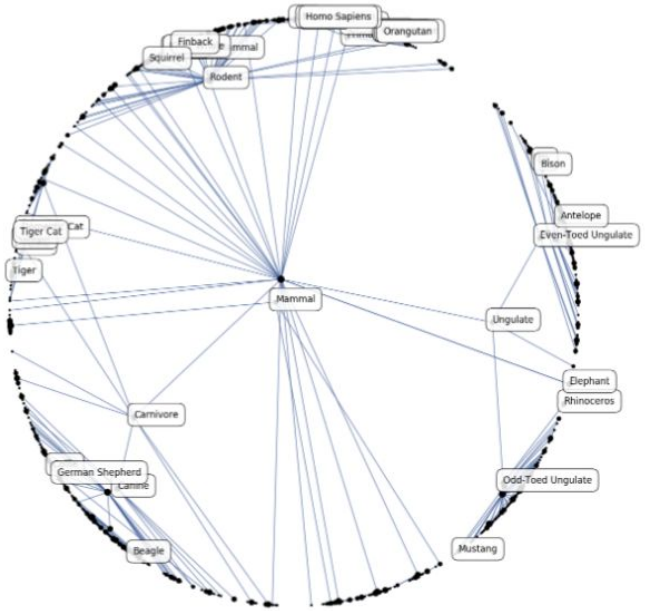
Poincaré Embeddings for Learning Hierarchical Representations

Maximilian Nickel
Facebook AI Research
maxn@fb.com

Douwe Kiela
Facebook AI Research
dkiela@fb.com



(a) Intermediate embedding after 20 epochs



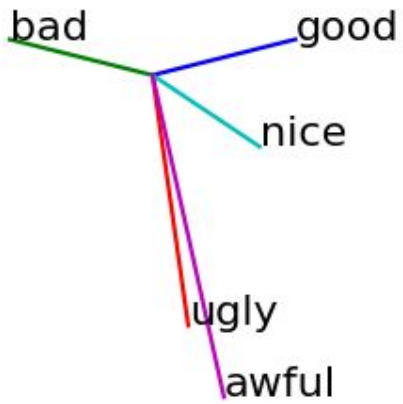
(b) Embedding after convergence

Vector representation of words - motivation

Word embeddings (toy example):

word	vector	id
good	=[1,0,0,0,0]	1
bad	=[0,1,0,0,0]	2
ugly	=[0,0,1,0,0]	3
nice	=[0,0,0,1,0]	4
awful	=[0,0,0,0,1]	5

word	VSM
good	[0.4,0.1]
bad	[-0.4,0.1]
ugly	[0.1,-0.7]
nice	[0.3,-0.2]
awful	[0.2,-0.9]



Why word embeddings:

- We want to embed high dimensional vectors in low dimensional space
- We want to have meaningful vector representations

Simple implementation of Glove algorithm

$$L_{\text{simple}} = \sum_{i,j}^{V} \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j \right)^2$$

Word: kolasinski

Row	model_als_X	model_als_X_log	model_ials	model_w2v	nmf_vec	svd_U_vec
1	"kolasinski"	"kolasinski"	"kolasinski"	"kolasinski"	"kolasinski"	"kolasinski"
2	"nowak"	"lent"	"heun"	"radial"	"brun"	"authork"
3	"mrenca"	"mrenca"	"miseikis"	"detected"	"piazza"	"lent"
4	"osika"	"piazza"	"pisa"	"beveren"	"beltram"	"roddaro"
5	"wach"	"bibitemlent"	"mrenca"	"discretized"	"strambini"	"beltram"
6	"bibitemchwiej"	"kirkner"	"zwierzycki"	"strengths"	"roddaro"	"piazza"
7	"bednarek"	"beltram"	"bagwell"	"spinconserving"	"heun"	"bibitemlent"
8	"bibitemszafran"	"authork"	"zebrowski"	"particle"	"poniedzialek"	"mrenca"
9	"poniedzia"	"roddaro"	"ferry"	"participate"	"silvestro"	"khomyakov"
10	"bibitemnowak"	"khomyakov"	"nanoscale"	"close"	"pisa"	"kirkner"

Word: energy

Row	model_als_X	model_als_X_log	model_ials	model_w2v	nmf_vec	svd_U_vec
1	"energy"	"energy"	"energy"	"energy"	"energy"	"energy"
2	"appear"	"levels"	"potential"	"vppsigma"	"levels"	"levels"
3	"corresponding"	"spectrum"	"magnetic"	"refosmy"	"spectrum"	"spectrum"
4	"lowestenergy"	"corresponding"	"field"	"isinfracphi"	"metal"	"electron"
5	"nearly"	"correspond"	"electron"	"ground"	"kinetic"	"wunsch"
6	"bright"	"corresponds"	"spin"	"forces"	"plate"	"fermi"
7	"anticrossing"	"lines"	"system"	"differences"	"transmitted"	"bibitemasm"
8	"near"	"localized"	"function"	"displayed"	"inducton"	"pereira"
9	"change"	"energies"	"electrons"	"klein"	"refu"	"calculation"
10	"appears"	"field"	"wave"	"simulations"	"partial"	"santos"

Word2vec basic ideas

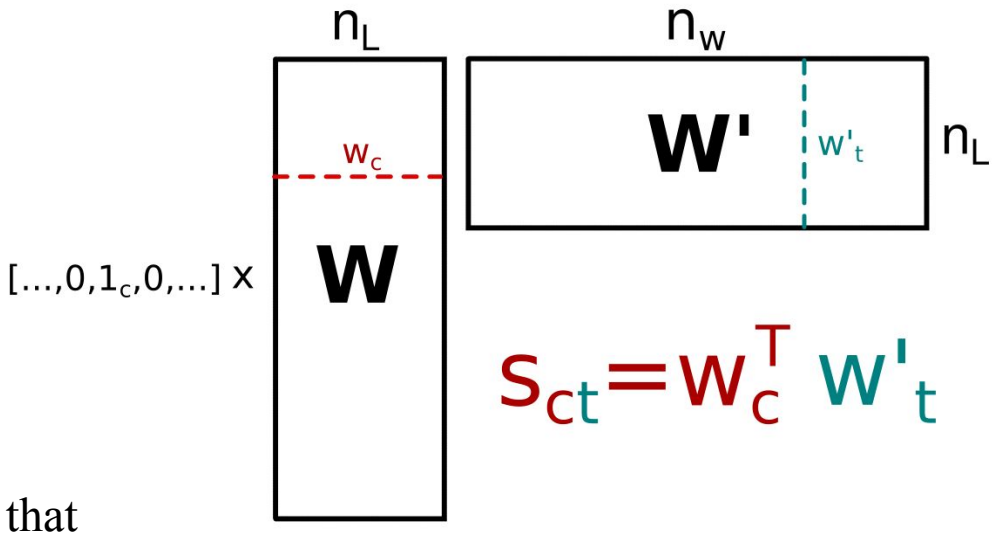
- **Distributional Hypothesis** - states that words that appear in the same contexts share semantic meaning
- It's a **predictive** algorithm - based on local co-occurrence

1. ***One word context***

"the quick brown fox jumped over the lazy dog"

- We start from creating context/target pairs:
  (**the** : **quick**, **quick** : **brown**, **brown** : **fox**, ...)
- The words are converted to sparse orthogonal representation:
  (**1** : **2**, **2** : **3**, **3** : **4**, ...)
- For each input word **k** we will maximize the probability of that the target word (**brown**) will appear after context word (**quick**) i.e. we want to maximize score $\mathbf{s_{kp}}$.

This is nothing else like shallow neural network with one hidden layer:



Word2vec basic ideas

1. *One word context*

"the math"

For a given context/target pair we compute score

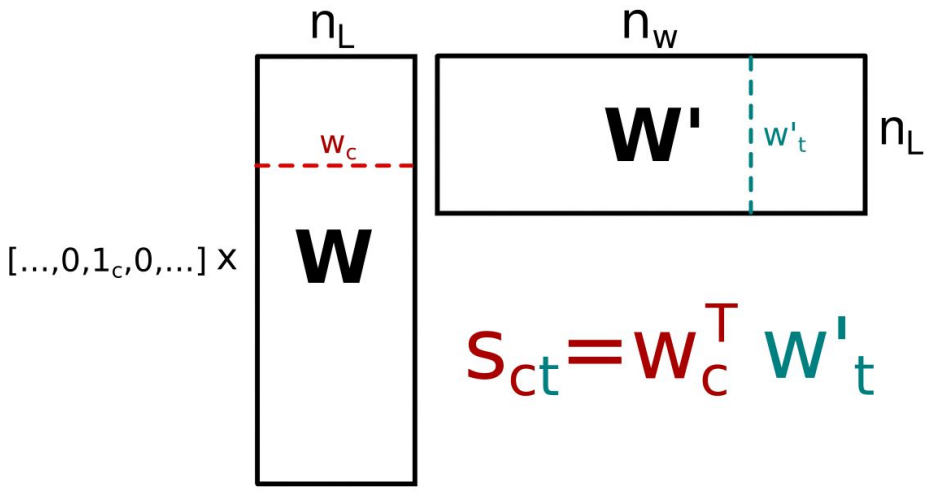
$$s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$$

The probability of observing word **p** for given input word **k** is computed with **Softmax**

$$p_{ct} = \frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}$$

The **context/target pair loss** is defined as

$$\begin{aligned} l_{ct} &= -\log(p_{ct}) = -\log\left(\frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}\right) \\ &= -s_{ct} + \log\left(\sum_i \exp(s_{ci})\right) \end{aligned}$$



The same but expressed in terms of **W** and **W'** matrices

$$l_{ct} = -\mathbf{w}_c^T \mathbf{w}'_t + \log\left(\sum_i \exp(\mathbf{w}_c^T \mathbf{w}'_i)\right)$$

Total loss is defined as

$$L = \frac{1}{N} \sum_{\{c,t\}} \left\{ -\mathbf{w}_c^T \mathbf{w}'_t + \log\left(\sum_i \exp(\mathbf{w}_c^T \mathbf{w}'_i)\right) \right\}$$

Word2vec basic ideas - in practice

1. Take your text and prepare word pairs table:

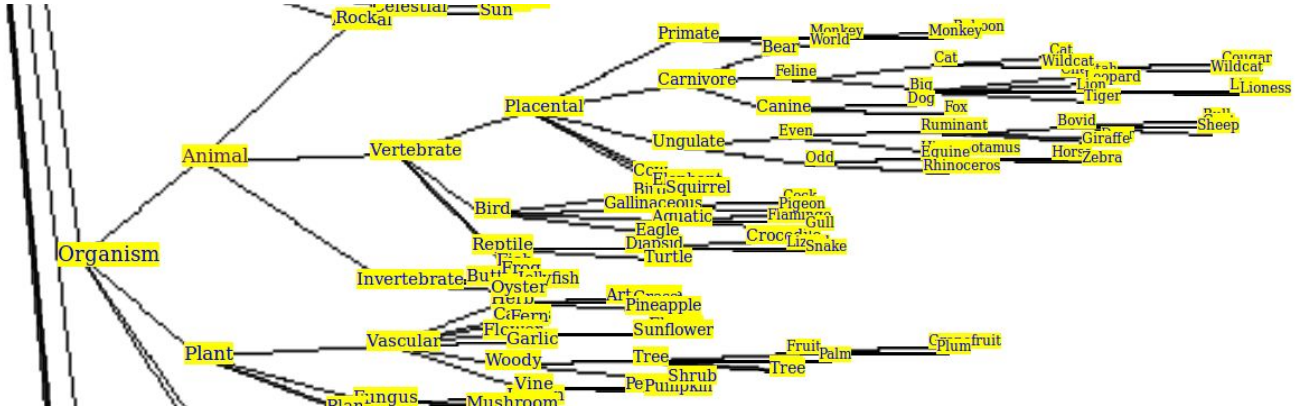
```
df = pd.read_csv("data/mammal.csv", index_col='id')
print(df.shape)
df[:3]
```

executed in 54ms, finished 14:32:30 2017-09-04

```
(6542, 2)
```

id	A	B
0	big_cat.n.01	carnivore.n.01
1	musteline_mammal.n.01	carnivore.n.01
2	yak.n.02	ox.n.02

Example fragment of wordnet tree



Example pairs from the tree are (same for every mammal noun)

```
('musk_kangaroo', 'mammal')
('musk_kangaroo', 'metatherian')
('musk_kangaroo.n.01', 'marsupial.n.01'),
('musk_kangaroo.n.01', 'kangaroo.n.01')
```

This gives: **1181** unique nouns

Word2vec basic ideas - in practice

1. Take your text and prepare word pairs table:

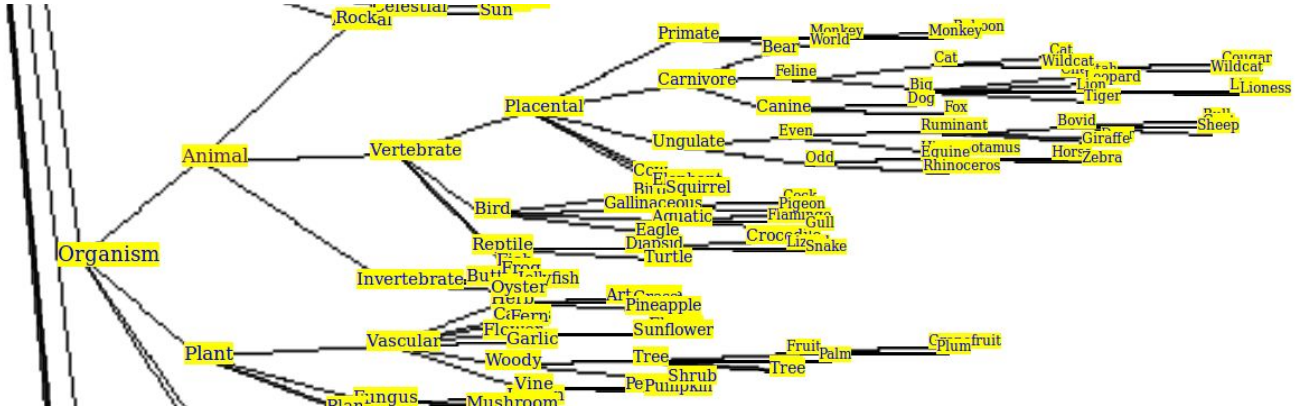
Example fragment of wordnet tree

```
df = pd.read_csv("data/mammal.csv", index_col='id')
print(df.shape)
df[:3]
```

executed in 54ms, finished 14:32:30 2017-09-04

```
(6542, 2)
```

id	A	B
0	big_cat.n.01	carnivore.n.01
1	musteline_mammal.n.01	carnivore.n.01
2	yak.n.02	ox.n.02



2. Convert to indices

```
df.A = df.A.map(word2index)
df.B = df.B.map(word2index)
df[:3]
```

executed in 36ms, finished 14:32:41 2017-09-04

id	A	B
0	180	1124
1	797	1124
2	391	714

Create dataset for keras

```
x_train, y_train = df.values[:, 0], df.values[:, 1]
x_train = np.expand_dims(x_train, -1)
y_train = np.expand_dims(y_train, -1)
x_train.shape, y_train.shape
```

executed in 20ms, finished 17:16:40 2017-09-03

```
((6542, 1), (6542, 1))
```

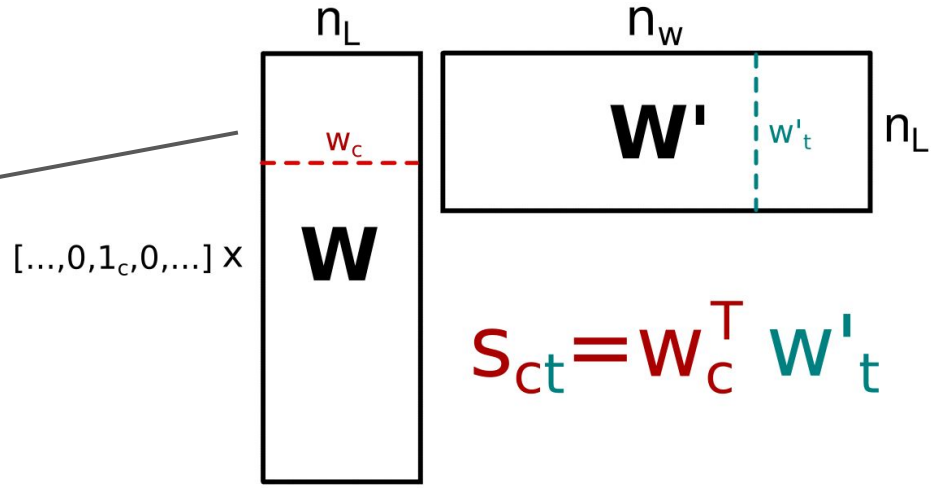
Word2vec basic ideas - in practice

3. Define ultra simple model:

```
left_input = Input(shape=(1,))
word_embd = Embedding(num_words, output_dim=latent_dim, name='embeddings')
u_word = word_embd(left_input)

def u_w_distance(u):
    w = word_embd.weights[0]
    return K.dot(u, K.transpose(w))

x = Lambda(u_w_distance)(u_word)
x = Reshape(target_shape=[-1])(x)
x = Activation('softmax')(x)

model = Model(inputs=left_input, outputs=x)
```

$$s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$$

$$p_{ct} = \frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}$$



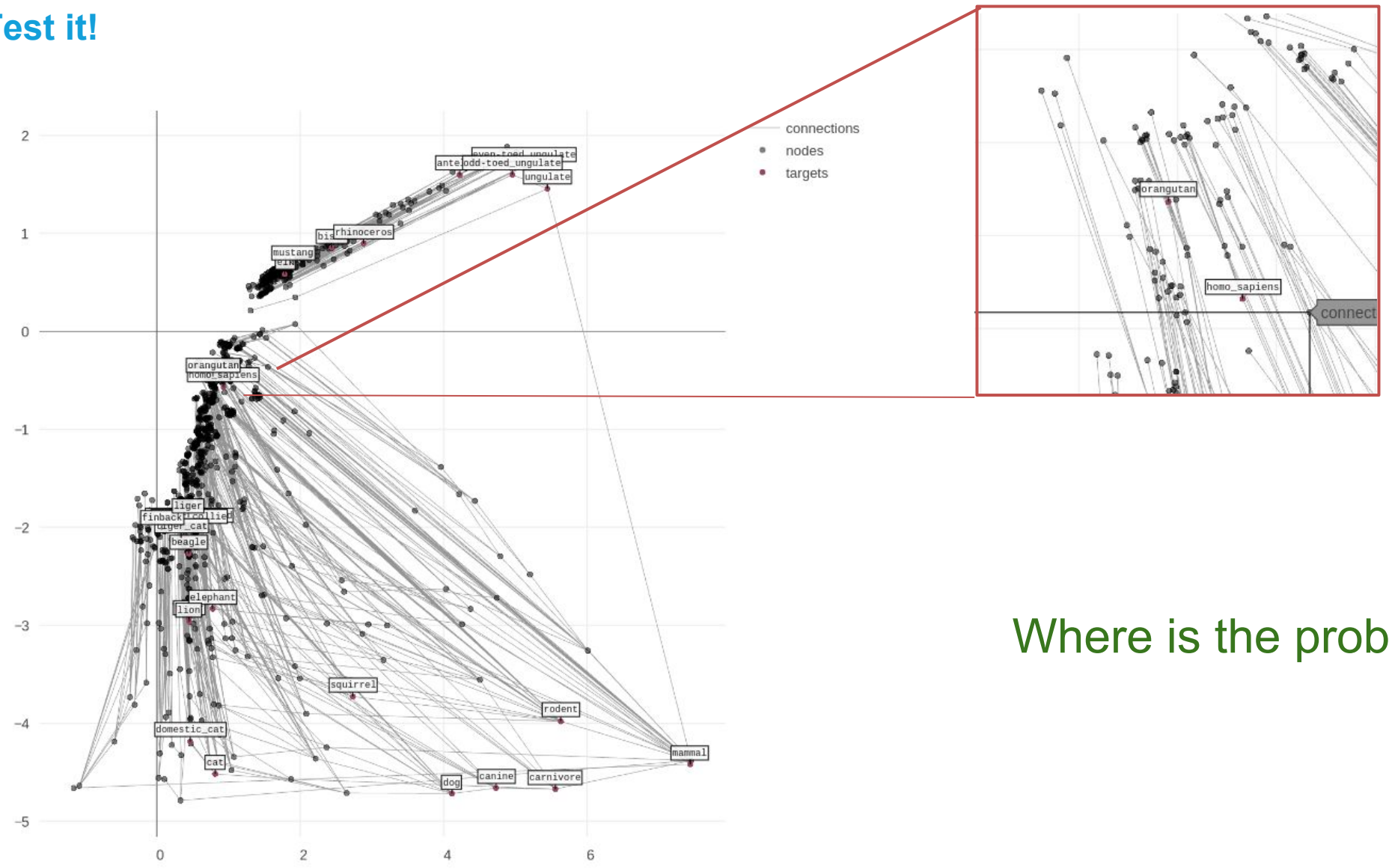
4. Train model

```
model.compile(optimizer=Adam(0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=20)
```

```
Epoch 1/20
6542/6542 [==============================] - 1s - loss: 7.0749 - acc: 0.0017
Epoch 2/20
6542/6542 [==============================] - 1s - loss: 7.0725 - acc: 0.0335
Epoch 3/20
6542/6542 [==============================] - 1s - loss: 7.0580 - acc: 0.0706
Epoch 4/20
6542/6542 [==============================] - 1s - loss: 7.0199 - acc: 0.0827
Epoch 5/20
6542/6542 [==============================] - 1s - loss: 6.9575 - acc: 0.0914
Epoch 6/20
6542/6542 [==============================] - 1s - loss: 6.8736 - acc: 0.0969
Epoch 7/20
6542/6542 [==============================] - 1s - loss: 6.7714 - acc: 0.1042
Epoch 8/20
6542/6542 [==============================] - 1s - loss: 6.6530 - acc: 0.1102
Epoch 9/20
6542/6542 [==============================] - 1s - loss: 6.5212 - acc: 0.1128
```

Word2vec basic ideas - in practice

5. Test it!



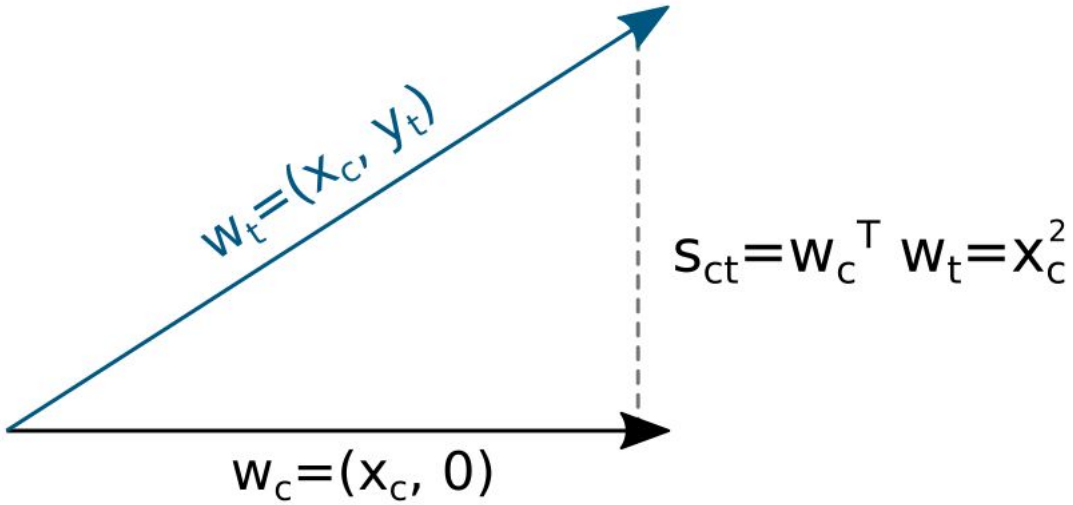
Where is the problem?

Word2vec basic ideas - limitations

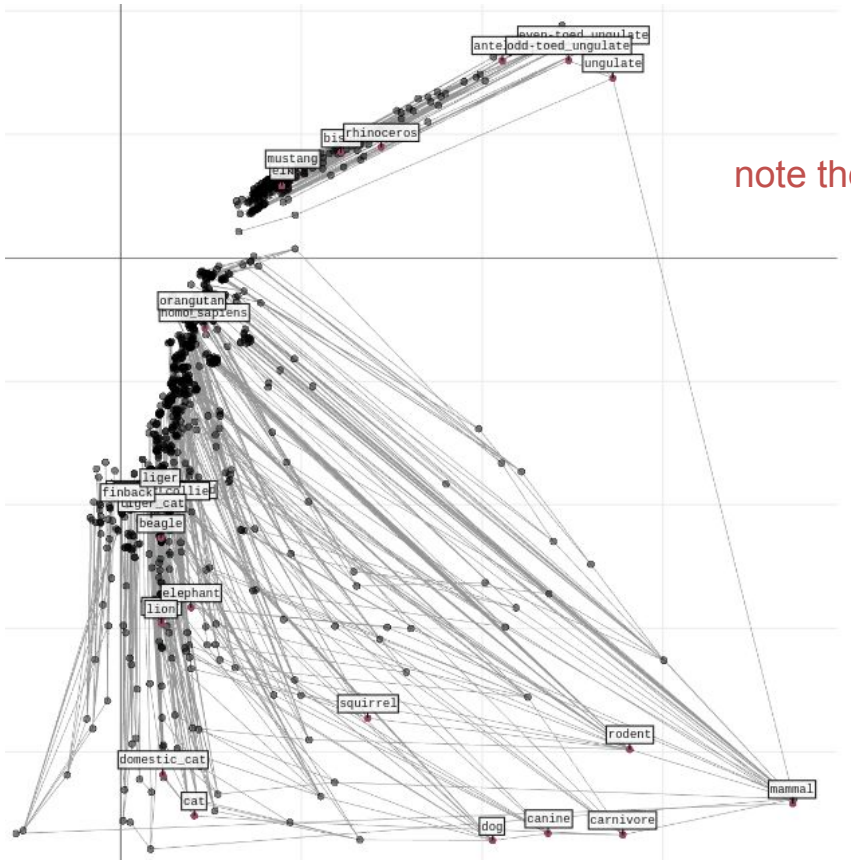
Where is the problem?

- The definition of the score is a dot product: $s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$

Any ideas for solution?



In consequence, infinitely separated points may result in same score...



note the structure

Word2vec basic ideas - limitations

Where is the problem?

Any ideas for solution?

- The definition of the score is a dot product: $s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$
- Normalization over all words - this leads to the loss from synonymes:

$$p_{ct} = \frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}$$

Consider two similar words, they will lead to the same score hence low p_{ct}

Word2vec basic ideas - limitations

Possible solutions:

- Redefine score function, to distance function: $s_{ct} = d(w_c, w_t)$
- Use sampling (clever one may reject synonyms from sampling): $p_{ct} = \frac{e^{-d(w_c,w_t)}}{\sum_{n \in N(w_c)} e^{-d(w_c,w_n)}}$
- Use different learning scheme (discussed later) e.g. matrix factorization for Poincare embeddings.

Things worth to check:

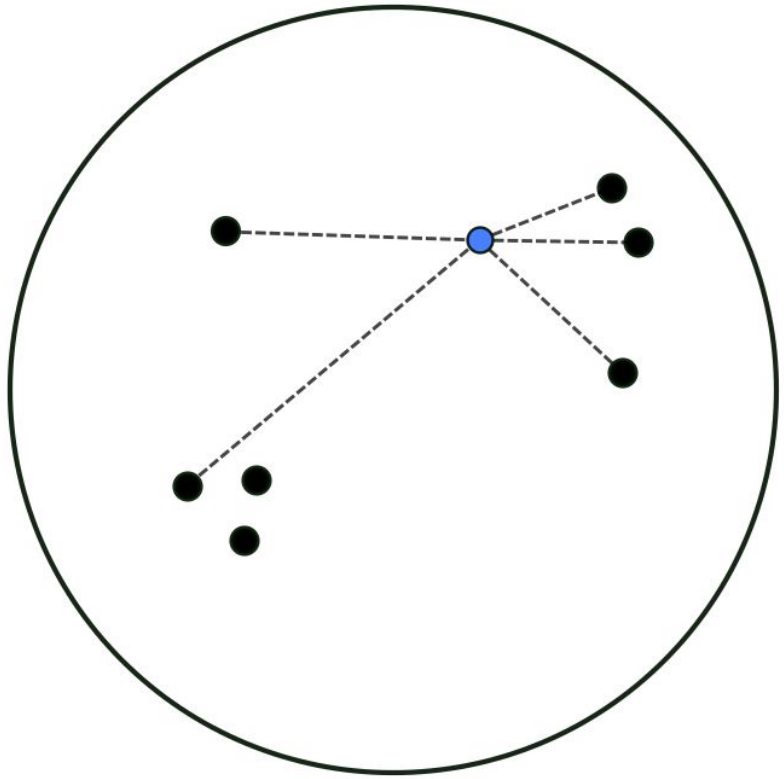
- Same things may happen in image classification, maybe a clever sampling of softmax function is not a bad idea?
- We are going to use predictive approach for computing word vectors, however maybe there is a possibility to derive count based method for general distance? Like in the paper below:

Neural Word Embedding as Implicit Matrix Factorization

Moving to hyperbolic spaces

Poincare embeddings - big picture

Use **distance** function instead of inner product



In Euclidean space those things are close

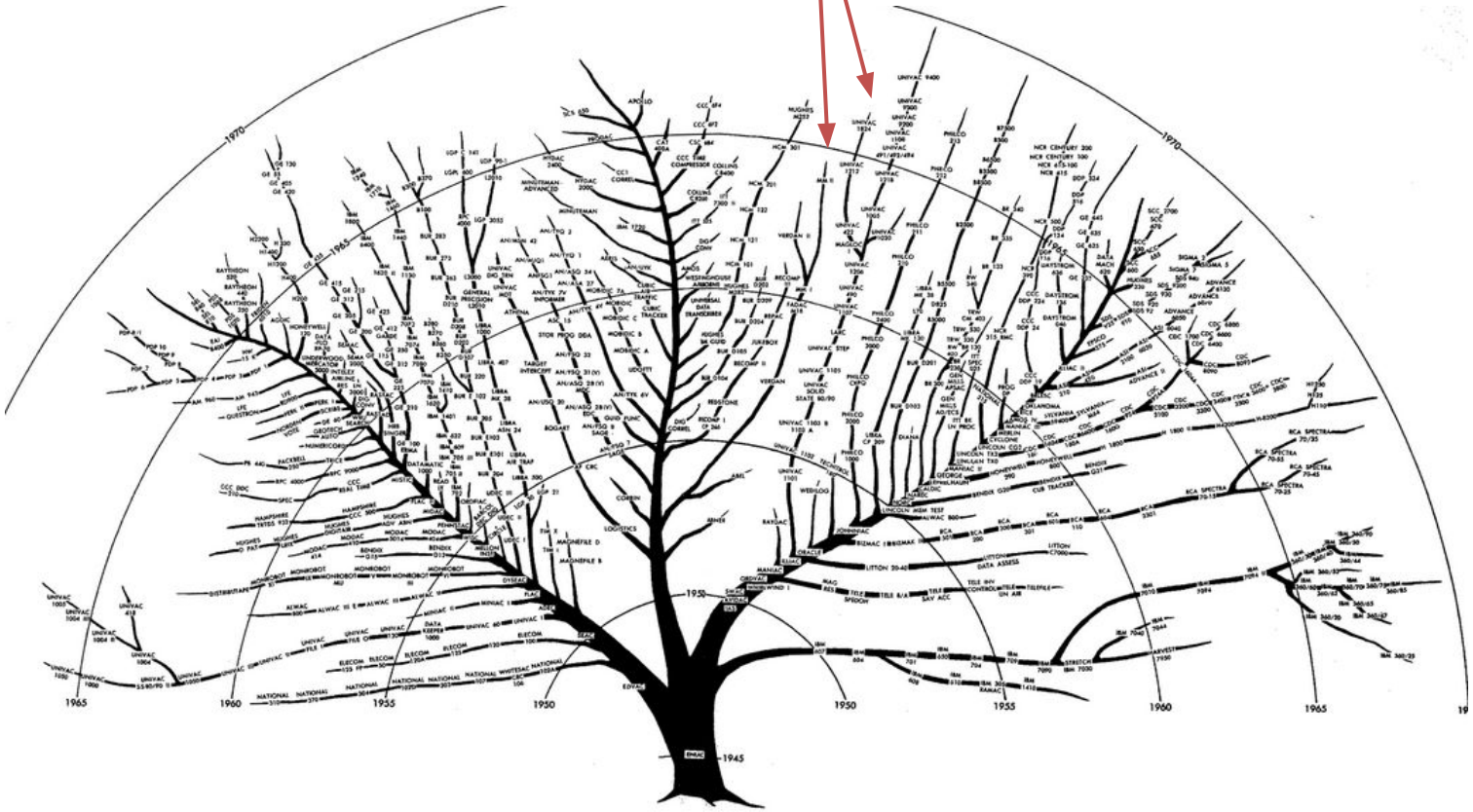
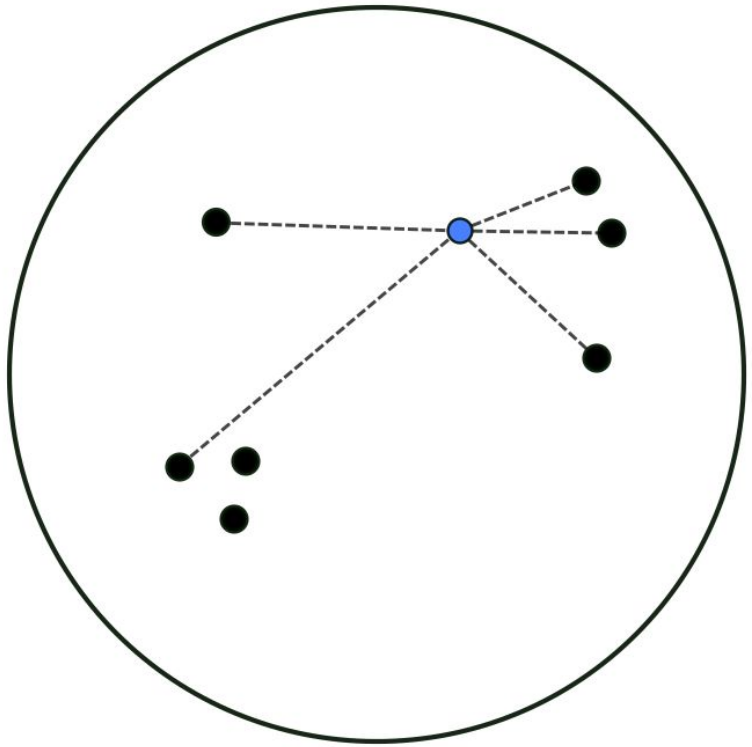


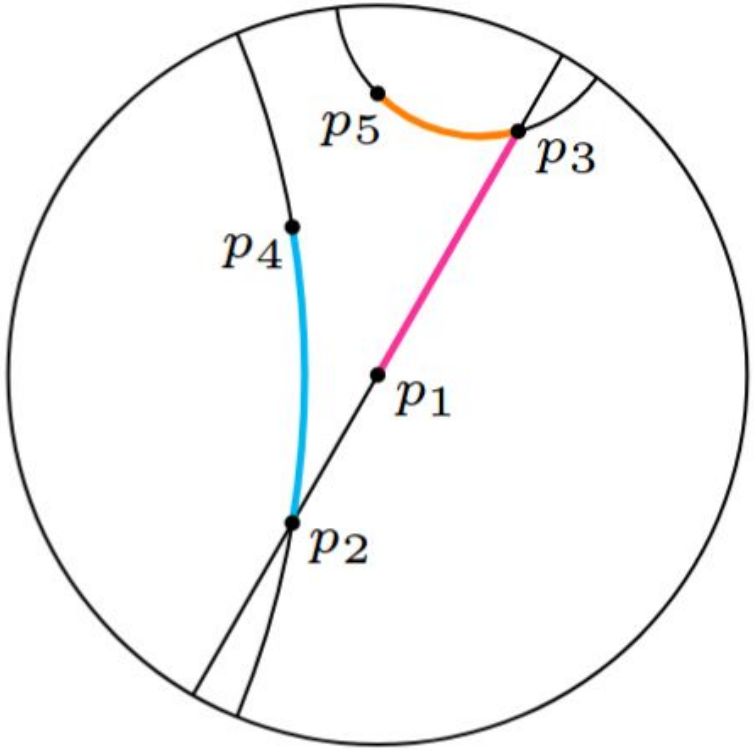
Image we want to embed complex tree structure in planar space

Poincare embeddings - big picture

The idea is to use the **hyperbolic** space instead **euclidean** one

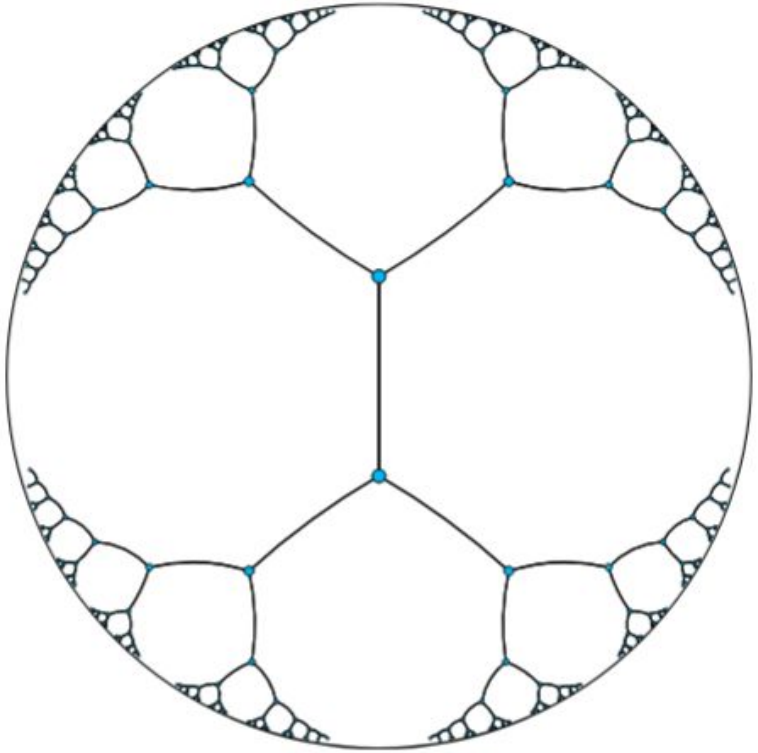


Euclidean space



(a) Geodesics of the Poincaré disk

“hypertree“



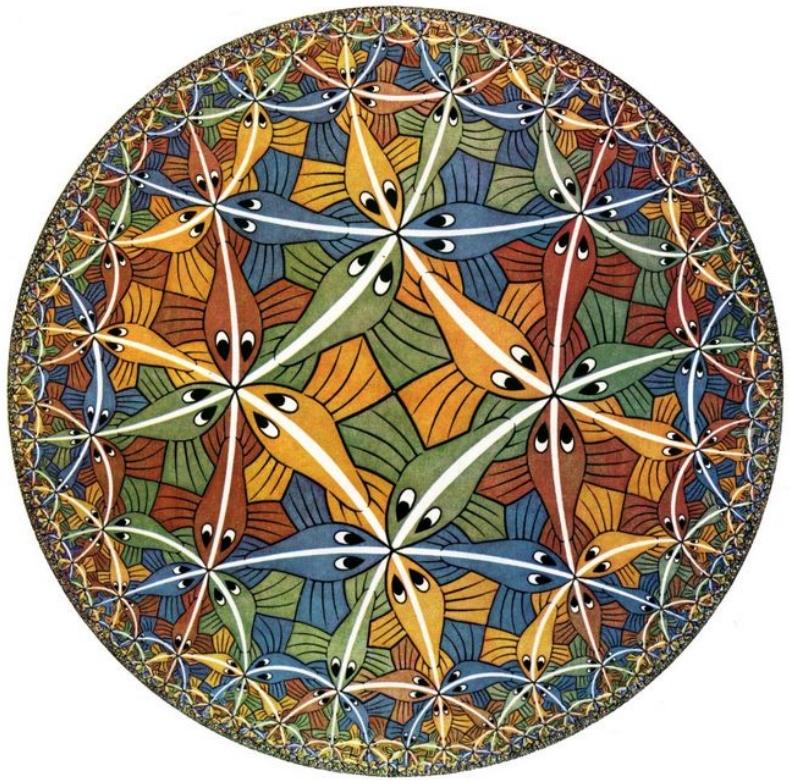
(b) Embedding of a tree in $\mathcal{B}^2$

In Poincare model of hyperbolic space:

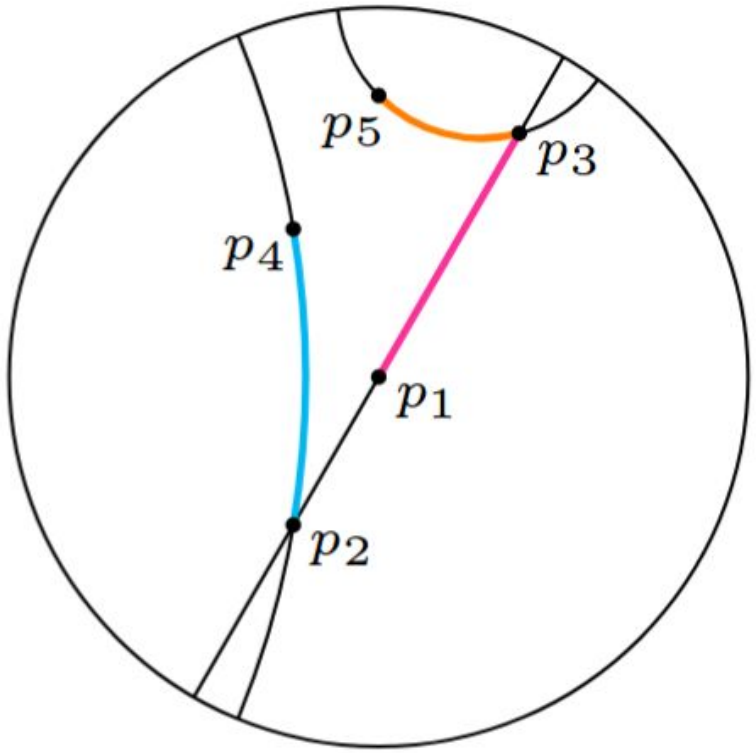
- The circle represents the surface of infinite area
- Geodesics are circles perpendicular to the boundaries
- Example of non-euclidean geometry: violation of 5th Euclid's postulate: the parallel postulate
- can be viewed as continuous version of discrete trees

Poincare embeddings - big picture

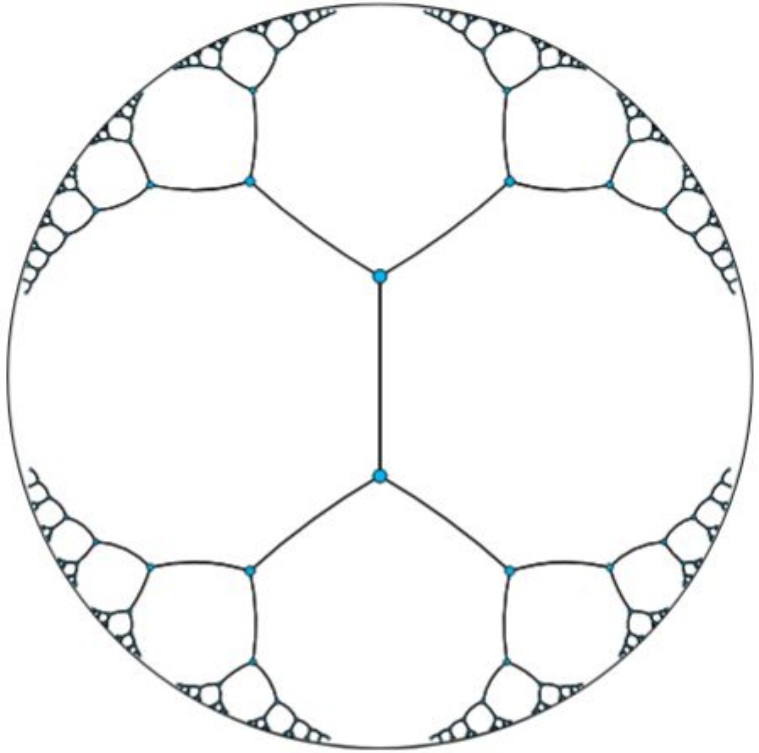
The idea is to use the **hyperbolic** space instead **euclidean** one



from Escher:
each fish has
the same area



(a) Geodesics of the Poincaré disk

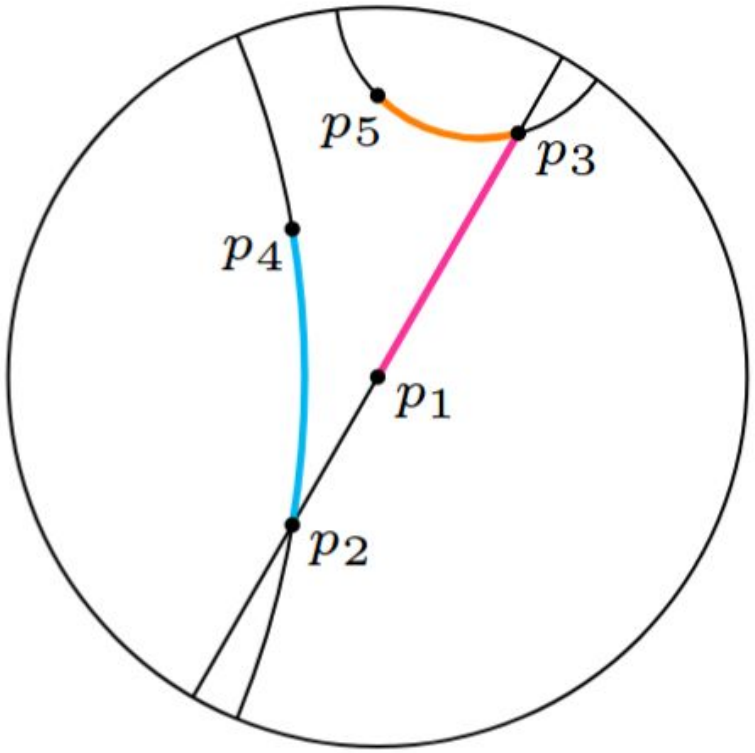


(b) Embedding of a tree in $\mathcal{B}^2$

Poincare embeddings - big picture

Distance in disc model is given by equation:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$



(a) Geodesics of the Poincaré disk

The authors provide optimization method

3.1 Optimization

Since the Poincaré Ball has a Riemannian manifold structure, we can optimize Equation (2) via stochastic Riemannian optimization methods such as RSGD [5] or RSVRG [34]. In particular, let $\mathcal{T}_\theta\mathcal{B}$ denote the tangent space of a point $\boldsymbol{\theta} \in \mathcal{B}^d$. Furthermore, let $\nabla_R \in \mathcal{T}_\theta\mathcal{B}$ denote the Riemannian gradient of $\mathcal{L}(\boldsymbol{\theta})$ and let ∇_E denote the Euclidean gradient of $\mathcal{L}(\boldsymbol{\theta})$. Using RSGD, parameter updates to minimize Equation (2) are then of the form

$$\boldsymbol{\theta}_{t+1} = \mathfrak{R}_{\theta_t}\left(-\eta_t \nabla_R \mathcal{L}(\boldsymbol{\theta}_t)\right)$$

where $\mathfrak{R}_{\theta_t}$ denotes the retraction onto $\mathcal{B}$ at $\boldsymbol{\theta}$ and η_t denotes the learning rate at time t. Hence, for the minimization of Equation (2), we require the Riemannian gradient and a suitable retraction. Since

Which simplifies to:

$$\text{proj}(\boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta}/\|\boldsymbol{\theta}\| - \varepsilon & \text{if } \|\boldsymbol{\theta}\| \geq 1 \\ \boldsymbol{\theta} & \text{otherwise}, \end{cases}$$

where ε is a small constant to ensure numerical stability. In all experiments we used $\varepsilon = 10^{-5}$. In summary, the full update for a single embedding is then of the form

$$\boldsymbol{\theta}_{t+1} \leftarrow \text{proj}\left(\boldsymbol{\theta}_t - \eta_t \frac{(1 - \|\boldsymbol{\theta}_t\|^2)^2}{4} \nabla_E\right). \quad (5)$$

Euclidean gradient $\nabla_E = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial d(\boldsymbol{\theta}, \boldsymbol{x})} \frac{\partial d(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \boldsymbol{\theta}}$

Poincare embeddings - big picture

The basic algorithm for Keras would be:

1. Take w2v simple model
2. Replace inner product with distance
3. Compute sampling
4. Compute gradients with SGD, multiply it by factor and project

```
left_input = Input(shape=(1,))
word_embd = Embedding(num_words, output_dim=latent_dim, name='embeddings')
u_word = word_embd(left_input)

def u_w_distance(u):
    w = word_embd.weights[0]
    return K.dot(u, K.transpose(w))

x = Lambda(u_w_distance)(u_word)
x = Reshape(target_shape=[-1])(x)
x = Activation('softmax')(x)

model = Model(inputs=left_input, outputs=x)
```

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$

$$p_{ct} = \frac{e^{-d(w_c, w_t)}}{\sum_{n \in N(w_c)} e^{-d(w_c, w_n)}}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \operatorname{proj}\left(\boldsymbol{\theta}_t - \eta_t \frac{(1 - \|\boldsymbol{\theta}_t\|^2)^2}{4} \nabla_E\right)$$

This step requires writing custom optimizer or extending existing one, but that is simple

Poincare embeddings - results

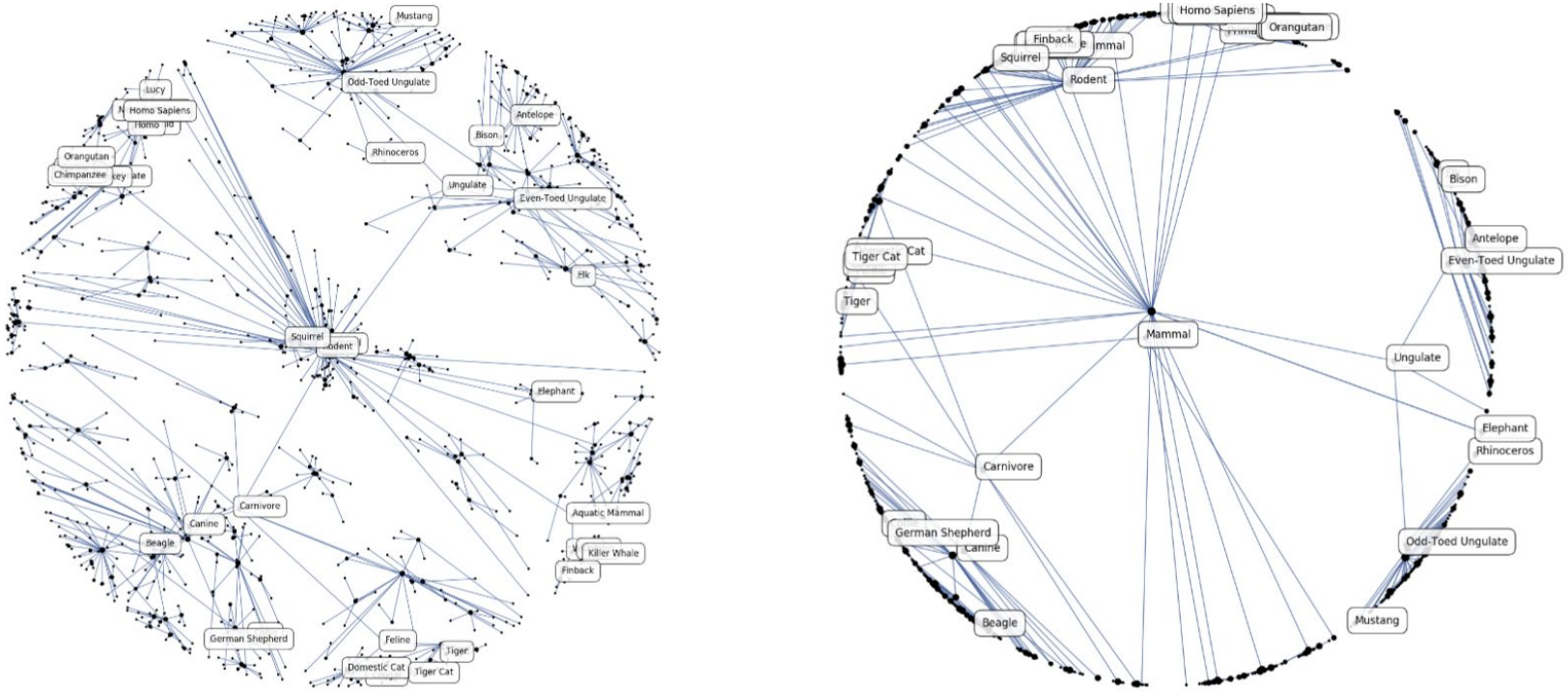
Selected results:

Table 1: Experimental results on the transitive closure of the WORDNET noun hierarchy. Highlighted cells indicate the best Euclidean embeddings as well as the Poincaré embeddings which acheive equal or better results. Bold numbers indicate absolute best results.

			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**
WORDNET **Link Pred.**	**Euclidean**	Rank	3311.1	2199.5	952.3	351.4	190.7	81.5
		MAP	0.024	0.059	0.176	0.286	0.428	0.490
	Translational	Rank	65.7	56.6	52.1	47.2	43.2	40.4
		MAP	0.545	0.554	0.554	0.56	0.562	0.559
	Poincaré	Rank	5.7	**4.3**	4.9	4.6	4.6	4.6
		MAP	0.825	0.852	0.861	**0.863**	0.856	0.855

Poincare embeddings - results

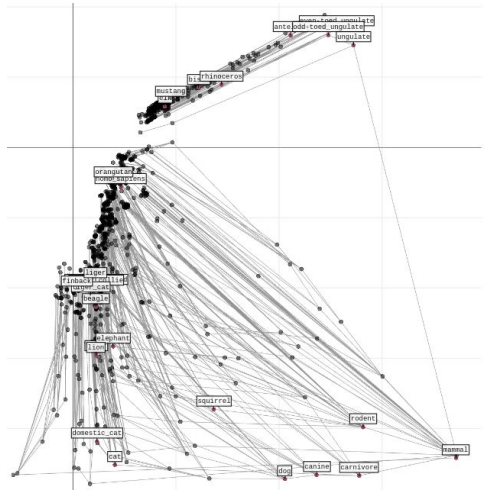
Selected results: on **mammal** sub-tree



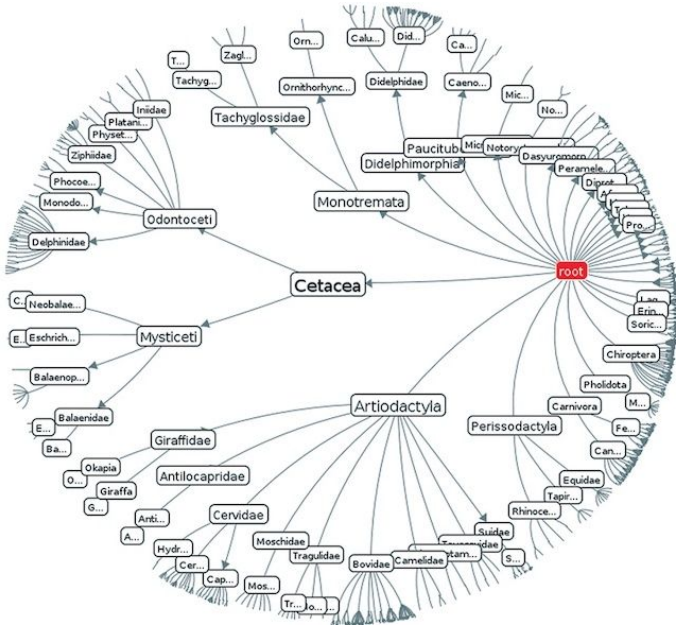
(a) Intermediate embedding after 20 epochs

(b) Embedding after convergence

Figure 2: Two-dimensional Poincaré embeddings of transitive closure of the WORDNET mammals subtree. Ground-truth `is-a` relations of the original WORDNET tree are indicated via blue edges. A Poincaré embedding with $d = 5$ achieves mean rank 1.26 and MAP 0.927 on this subtree.



compare with **that thing** obtained with inner product

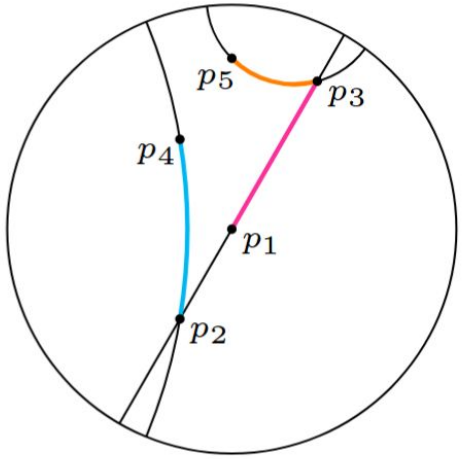


compare with example generated hypertree

Understanding Poincare embeddings

Understanding Poincare embeddings

We are going to answer the following questions (with limited number of math)



(a) Geodesics of the Poincaré disk

- Why the shortest path is not the straight one?
- What is a metric space ?
- What is natural gradient ?
- What is blablabla ...

For now let's us assume that we know what is Poincare model of hyperbolic space

Understanding Poincare embeddings: Euclidean space

Consider following problem (check out the paper in the link!!!)

$$L(x) = (x-1)^2 + y^2$$

Find minimum of that function using gradient descent

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu(k)\frac{\partial \mathcal{J}(\mathbf{w}(k))}{\partial \mathbf{w}},$$

Let's re-parameterize our problem with following mapping:

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \end{aligned} \longrightarrow L(r,\theta) = (r\cos(\theta)-1)^2 + (r\sin(\theta))^2$$

Our gradients will change with new parametrization

$$\nabla_x, \nabla_y \overset{?}{\Longrightarrow} \nabla_r, \nabla_\theta$$

~~$\mathbf{w}(k+1) = \mathbf{w}(k) - \mu(k)\frac{\partial \mathcal{J}(\mathbf{w}(k))}{\partial \mathbf{w}},$~~ This is no more valid

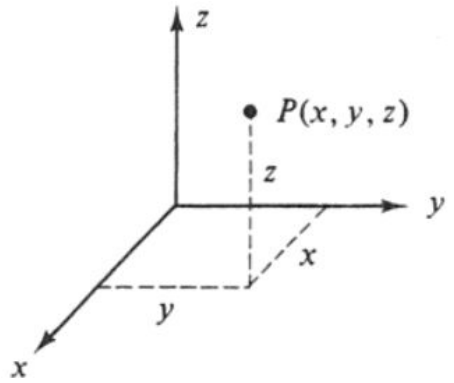
Understanding Poincare embeddings: Riemannian space

We are going to derive general stochastic gradient descent for arbitrary spaces

In Euclidean space the (x, y) coordinate system is orthonormal

$$x = r\cos(\theta)$$
$$y = r\sin(\theta)$$

polar coordinates



The infinitesimal translation is given by

$$\Delta = (x + dx, y + dy) - (x, y) = (dx, dy)$$

With squared length - “the interval”:

$$\Delta^2 = (dx, dy)\begin{pmatrix} dx \\ dy \end{pmatrix} = dx^2 + dy^2$$

The increment in x by change in transformed frame

$$\begin{aligned} x + dx &= (r + dr)\cos(\theta + d\theta) \\ &= (r + dr)\left(\cos(\theta) - \sin(\theta)d\theta\right) \\ &= \underbrace{r\cos(\theta)}_{\mathbf{x}} + dr\cos(\theta) - rd\theta\sin(\theta) \end{aligned}$$

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} dr\cos(\theta) - rd\theta\sin(\theta) \\ dr\sin(\theta) + rd\theta\cos(\theta) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix}\begin{pmatrix} dr \\ d\theta \end{pmatrix}$$

from this we can compute interval in new coordinate frames

Understanding Poincare embeddings: Riemannian space

We are going to derive general stochastic gradient descent for arbitrary spaces

With squared length - “the interval”:

$$\Delta^2 = (dx, dy) \begin{pmatrix} dx \\ dy \end{pmatrix} = dx^2 + dy^2$$

$$\begin{aligned} x &= r\cos(\theta) \\ y &= r\sin(\theta) \end{aligned}$$

polar coordinates

$$\begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} dr\cos(\theta) - rd\theta\sin(\theta) \\ dr\sin(\theta) + rd\theta\cos(\theta) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix} \begin{pmatrix} dr \\ d\theta \end{pmatrix}$$

$$\Delta^2 = (dx, dy) \begin{pmatrix} dx \\ dy \end{pmatrix} = \begin{pmatrix} dr & d\theta \end{pmatrix} \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix}^T \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix} \begin{pmatrix} dr \\ d\theta \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$

Riemannian metric tensor

In polar coordinate frames the interval depends on position!!!

$$\Delta^2 = dr^2 + r^2 d\theta^2 = dx^2 + dy^2$$

In general we write: $\Delta^2 = \sum_{ij} g_{ij} du_i du_j$ $\quad du = \{dr, d\theta\}$ for polar case

Understanding Poincare embeddings: Riemannian space

Consider following problem

$$L(x) = (x-1)^2 + y^2 \longrightarrow L(r,\theta) = (r\cos(\theta)-1)^2 + (r\sin(\theta))^2$$

We are looking for general expression for gradient update for arbitrary spaces

$$L(\mathbf{u}+\mathbf{du}) = L(\mathbf{u}) + \nabla L(\mathbf{u})^T \mathbf{du}$$

Standard question: which direction of du minimizes L at most?

We put $\mathbf{du} = \epsilon \mathbf{n}$ with constraint $|\mathbf{n}|^2 = 1 = \sum_{ij} g_{ij} n_i n_j = \mathbf{n}^T \mathbf{G} \mathbf{n}$

$$L(\mathbf{u}+\mathbf{du}) = L(\mathbf{u}) + \epsilon \nabla L(\mathbf{u})^T \mathbf{n}$$

By the Lagrangean method, we want to minimize **this** with condition on **n**

$$\frac{\partial}{\partial n_i}\left[\nabla L(\mathbf{u})^T \mathbf{n} + \lambda\left(1 - \mathbf{n}^T \mathbf{G} \mathbf{n}\right)\right] = 0 \rightarrow \nabla L(\mathbf{u}) = 2\lambda \mathbf{G} \mathbf{n}$$

$$\mathbf{n} = \frac{1}{2\lambda} \mathbf{G}^{-1} \nabla L(\mathbf{u}) \rightarrow \mathbf{du} = \frac{\epsilon}{2\lambda} \mathbf{G}^{-1} \nabla L(\mathbf{u})$$

update direction

Understanding Poincare embeddings: Riemannian space

Consider following problem

$$L(x) = (x-1)^2 + y^2 \longrightarrow L(r,\theta) = (r\cos(\theta)-1)^2 + (r\sin(\theta))^2$$

We are looking for general expression for gradient update for arbitrary spaces

$$L(\mathbf{u}+\mathbf{du}) = L(\mathbf{u}) + \nabla L(\mathbf{u})^T \mathbf{du} \longrightarrow \mathbf{du} = \frac{\epsilon}{2\lambda}\mathbf{G}^{-1}\nabla L(\mathbf{u})$$

The gradient descent update is then:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr}\mathbf{G}^{-1}\nabla L(\mathbf{u})$$

This equation is called natural gradient descent

Euclides gradient: same as computed by Keras

$$L(r,\theta) = (r\cos(\theta)-1)^2 + (r\sin(\theta))^2$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$

Metric generated by parametrization of Euclidean space in our case

Understanding Poincare embeddings: paper

Let's go back to the paper for a while and see what we know so far:

the **distance**, not discussed yet, but it is induced by metric tensor

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$

$$g_{\boldsymbol{x}} = \left(\frac{2}{1 - \|\boldsymbol{x}\|^2}\right)^2 g^E$$

Metric tensor for Poincare model - **done** - we know that it may come from some parametrization of space? what space - not euclidean...

The authors provide optimization method

3.1 Optimization

Since the Poincaré Ball has a Riemannian manifold structure, we can optimize Equation (2) via stochastic Riemannian optimization methods such as RSGD [5] or RSVRG [34]. In particular, let $\mathcal{T}_\theta\mathcal{B}$ denote the tangent space of a point $\boldsymbol{\theta} \in \mathcal{B}^d$. Furthermore, let $\nabla_R \in \mathcal{T}_\theta\mathcal{B}$ denote the Riemannian gradient of $\mathcal{L}(\boldsymbol{\theta})$ and let ∇_E denote the Euclidean gradient of $\mathcal{L}(\boldsymbol{\theta})$. Using RSGD, parameter updates to minimize Equation (2) are then of the form

$$\boldsymbol{\theta}_{t+1} = \mathfrak{R}_{\theta_t}\left(-\eta_t \nabla_R \mathcal{L}(\boldsymbol{\theta}_t)\right)$$

where $\mathfrak{R}_{\theta_t}$ denotes the retraction onto $\mathcal{B}$ at $\boldsymbol{\theta}$ and η_t denotes the learning rate at time t. Hence, for the minimization of Equation (2), we require the Riemannian gradient and a suitable retraction. Since

Retraction - projects gradient into Poincare Ball - here they used simple update (u + grad) leading to natural gradient method.

Which simplifies to:

$$\operatorname{proj}(\boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta}/\|\boldsymbol{\theta}\| - \varepsilon & \text{if } \|\boldsymbol{\theta}\| \geq 1 \\ \boldsymbol{\theta} & \text{otherwise}, \end{cases}$$

projection - keeps params in Poincare Ball

where ε is a small constant to ensure numerical stability. In all experiments we used $\varepsilon = 10^{-5}$. In summary, the full update for a single embedding is then of the form

$$\text{Euclidean gradient } \nabla_E = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial d(\boldsymbol{\theta}, \boldsymbol{x})} \frac{\partial d(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \boldsymbol{\theta}}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \operatorname{proj}\left(\boldsymbol{\theta}_t - \eta_t \frac{(1 - \|\boldsymbol{\theta}_t\|^2)^2}{4} \nabla_E\right). \quad (5)$$

natural gradient

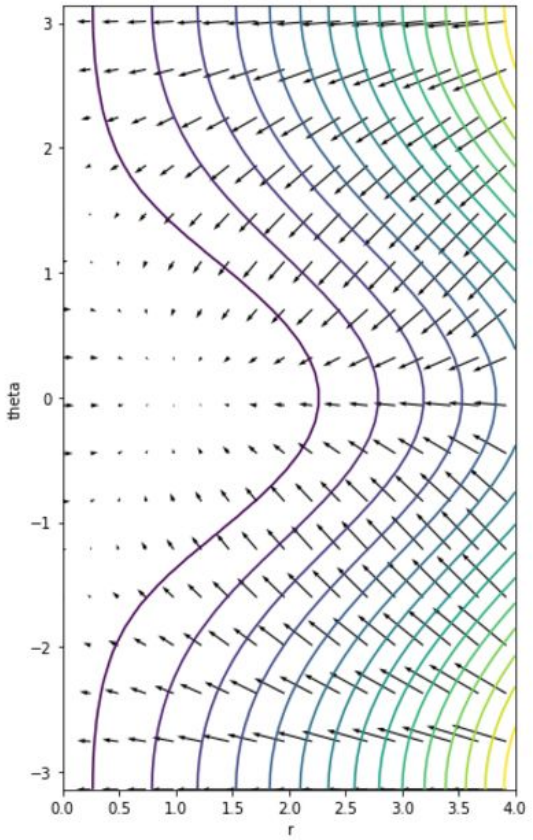
Understanding Poincare embeddings: Toy example

Consider following problem

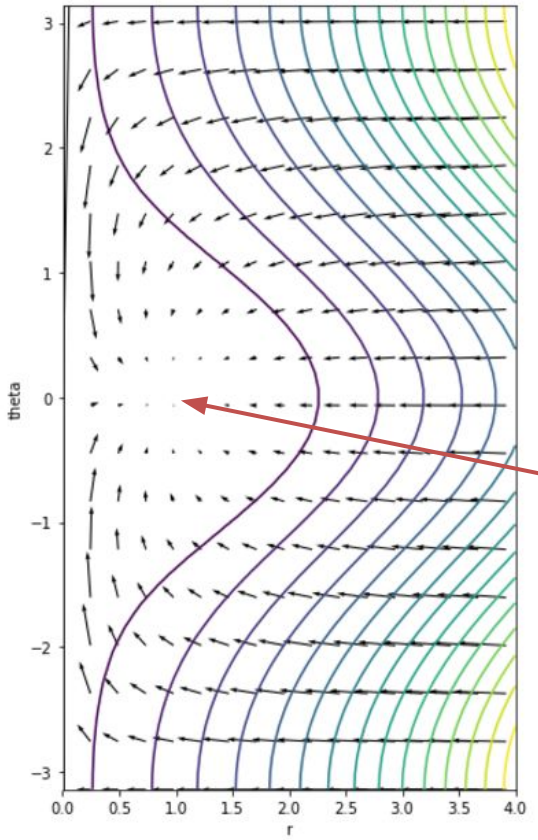
$$L(x) = (x-1)^2 + y^2 \longrightarrow L(r,\theta) = (r\cos(\theta) - 1)^2 + (r\sin(\theta))^2$$

Using natural gradient method: $\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr}\mathbf{G}^{-1}\nabla L(\mathbf{u})$ $\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$

Gradients



Natural gradient



$$du = \{dr, d\theta\}$$

here is the minimum

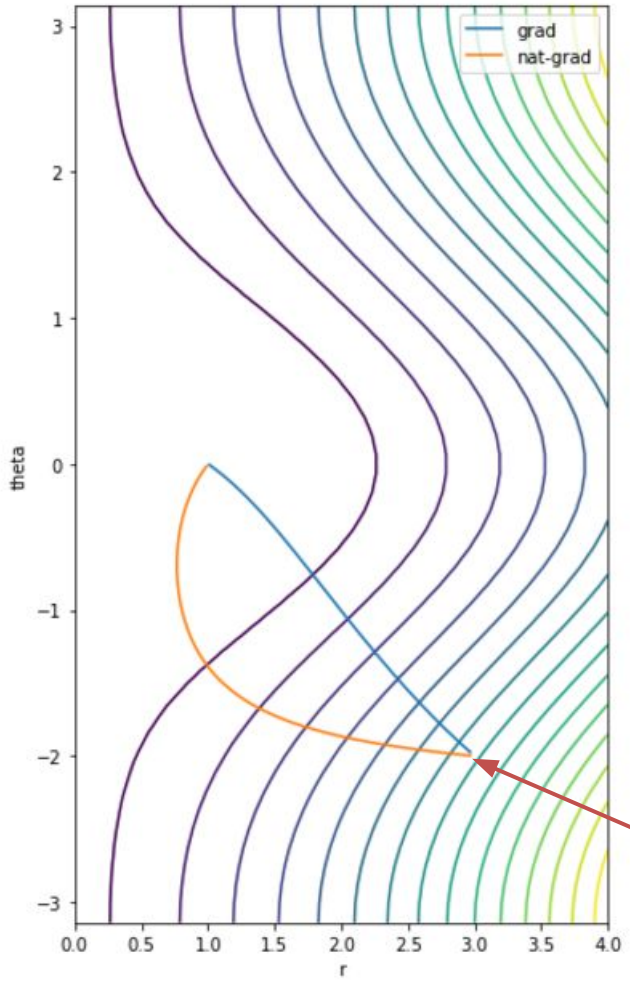
Note that this plot is in polar coordinates

Understanding Poincare embeddings: The distance

Comparison between standard gradient and natural one

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr}\mathbf{G}^{-1}\nabla L(\mathbf{u})$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$



initial point

It seems that with natural grad the trajectory goes along longer path :(what's wrong?

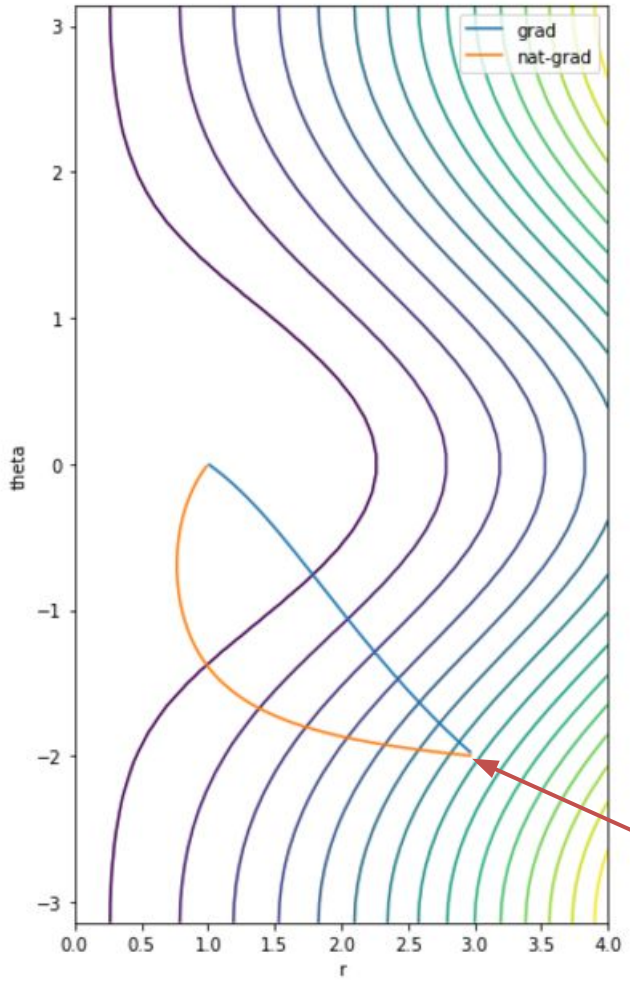
We are not in the cartesian coordinates, hence the illusion :)

Understanding Poincare embeddings: The distance

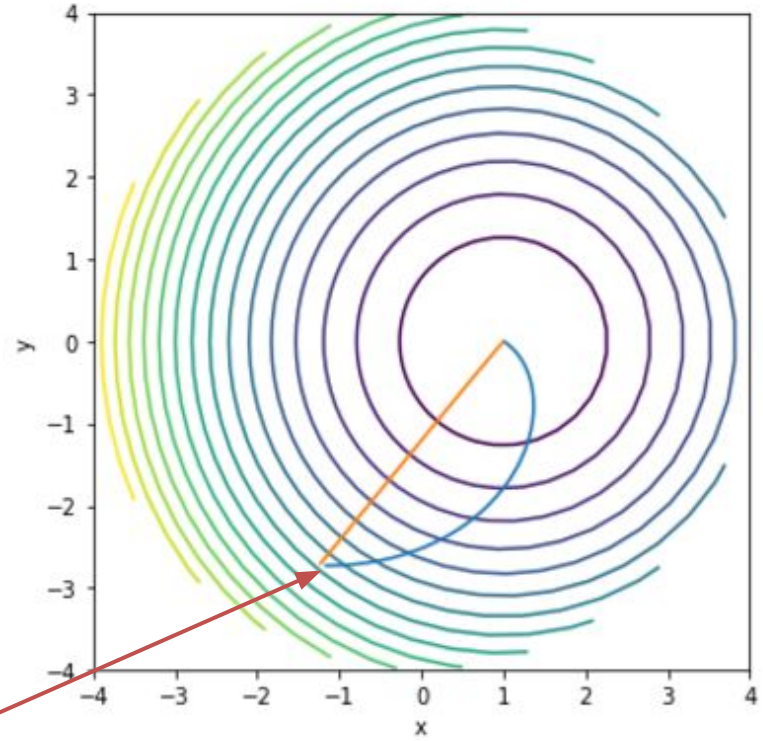
Comparison between standard gradient and natural one

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr}\mathbf{G}^{-1}\nabla L(\mathbf{u})$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$



Same picture but in original frame



initial point

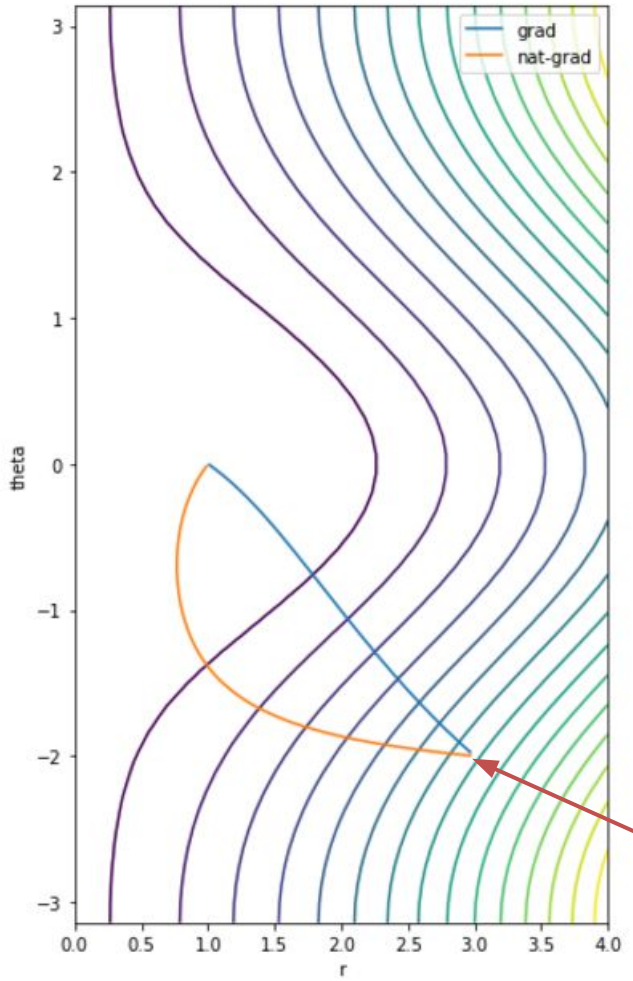
Natural gradient goes along straight line!

Understanding Poincare embeddings: The distance

Comparison between standard gradient and natural one

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr}\mathbf{G}^{-1}\nabla L(\mathbf{u})$$

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 0 & r^2 \end{pmatrix}$$



initial point

Same picture but in original frame

We can numerically compute the length of the path, by integrating over infinitesimal intervals

$$\Delta^2 = dr^2 + r^2 d\theta^2 = dx^2 + dy^2$$

$$l = \sum_i \sqrt{dr^2 + r^2 d\theta^2}$$

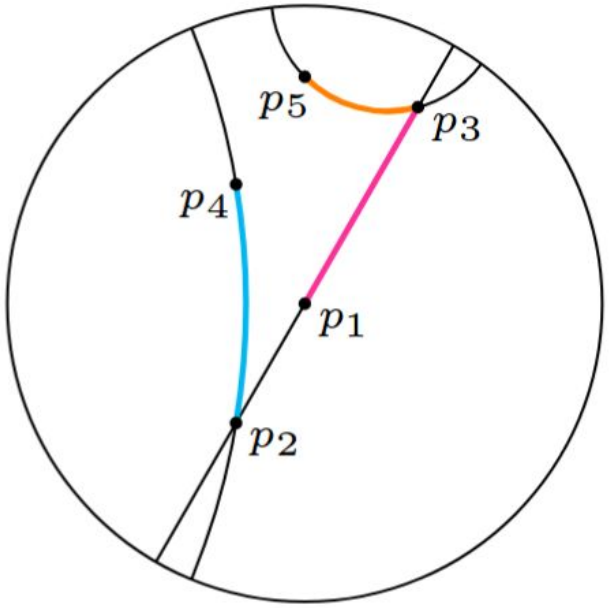
In general the geodesic is the **shortest path between two points**

$$l = \sum_i dt \sqrt{\left(\frac{dr}{dt}\right)^2 + r^2\left(\frac{d\theta}{dt}\right)^2} \rightarrow \int dt \sqrt{g_{ij}(t)v_i(t)v_j(t)}$$

In **euclidean we have straight lines**, in polar coordinates we have non trivial curves.

Understanding Poincare embeddings: The distance

Let's go back to the paper: how to find geodesics?



(a) Geodesics of the Poincaré disk

Having metric tensor we can compute the interval

$$g_{\boldsymbol{x}} = \left(\frac{2}{1-\|\boldsymbol{x}\|^2}\right)^2 g^E \qquad \Delta^2 = \sum_{ij} g_{ij} du_i du_j$$

And we get:

$$ds^2 = 4\frac{dx^2+dy^2}{(1-x^2-y^2)^2}$$

Derive the expression of distance between two points (p_i, p_j) which minimizes the integral

$$d(p_i, p_j) = \int_{p_i}^{p_j} ds$$

The solution is:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u}-\boldsymbol{v}\|^2}{(1-\|\boldsymbol{u}\|^2)(1-\|\boldsymbol{v}\|^2)}\right)$$

Understanding Poincare embeddings: Summary #1

So far we were working with trivial flat space i.e. Euclidean one

- We know that we can parametrize euclidean space and compute metric tensor from interval definition

$$\Delta^2 = \sum_{ij} g_{ij} du_i du_j$$

- This allows us to derive natural gradient update for efficient decent

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \eta_{lr} \mathbf{G}^{-1} \nabla L(\mathbf{u})$$

- There are methods which are able to deduce G from data - e.g. Fisher matrix approach, so even if we don't know what is the efficient gradient we can compute it. However this is computationally expensive…

See: Natural Gradient Works Efficiently in Learning

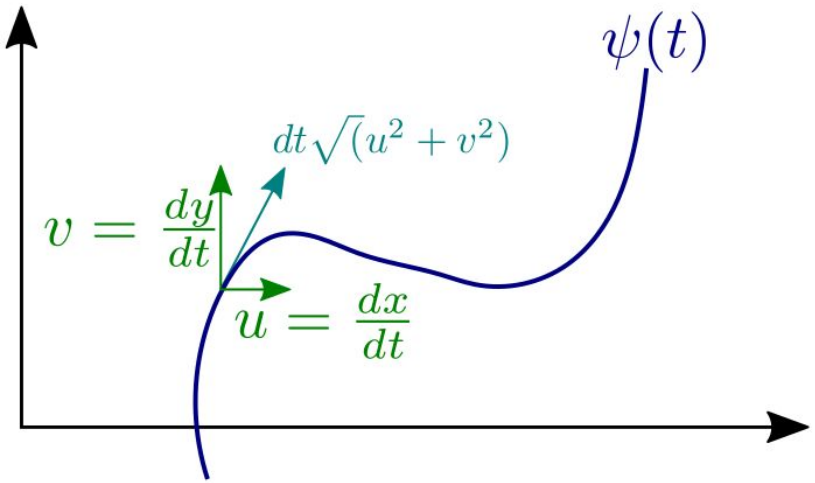
- Having the metric we can compute geodesics d(u, v) which are correct for given parametrization. Consider computing naive distance between two points in (r, theta) space.

Understanding Poincare embeddings: Curved spaces

link to document

So far we were working with trivial flat space i.e. Euclidean one

- There are other spaces which are curved - this has nothing to do with parametrization of space.
- Curvature of space is an intrinsic property of space (**demo**)
- The curvature of a space can be described by Gauss curvature **K** parameter.
- In general **K** may vary in space, we restrict ourselves to the constant curvature spaces.
- The intuition behind this parameter is following:



- The first derivative w.r.t *t* is the tangent vector

$$\psi'(t) = (x'(t), y'(t))$$

- The norm of the second derivative is the curvature e.g. straight line has zero curvature

Definition 16 *The curvature $k(s)$ of φ at $\varphi(s)$ is the length of $V'(s)$.*

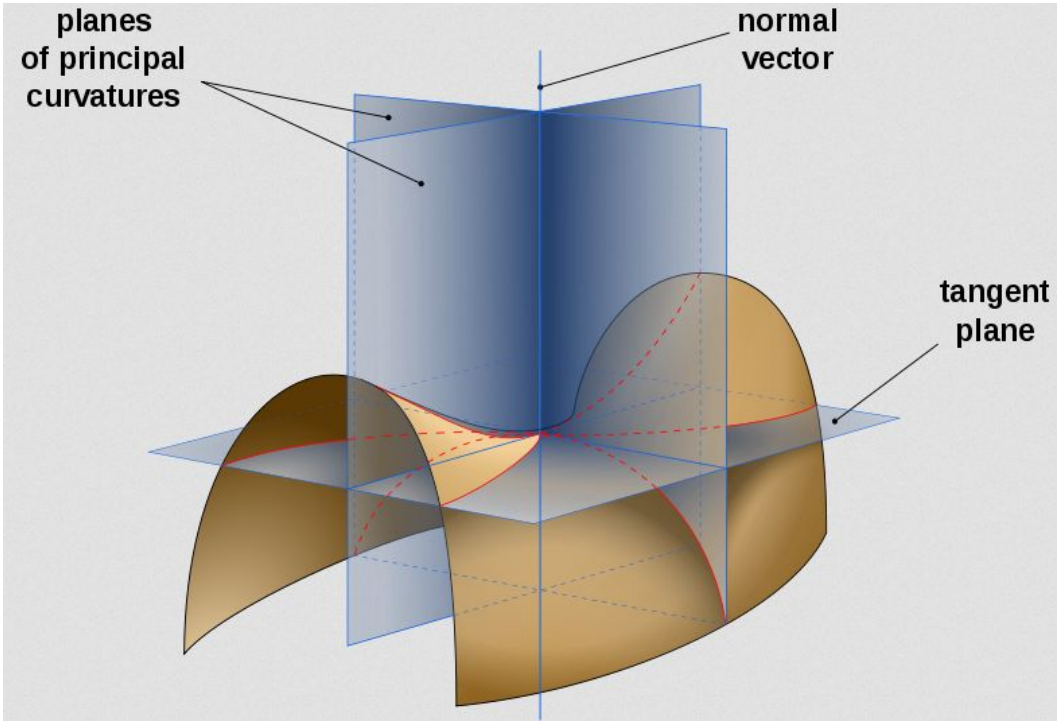
$$k(s) = \|V'(s)\| = \|\varphi''(s)\|.$$

Understanding Poincare embeddings: Curved spaces

link to document

Going from curve to surface:

- For example the curvature of the circle is: 1/r (larger circles have smaller curvature)
- For surfaces we can find cross sections which give the minimal and maximal value for k



source

The Gauss curvature

Definition 17 *The product of the two principal curvatures is called the Gauss curvature* k

$$K(P) = k_{min} k_{max}.$$

The value of K can be positive, negative or zero

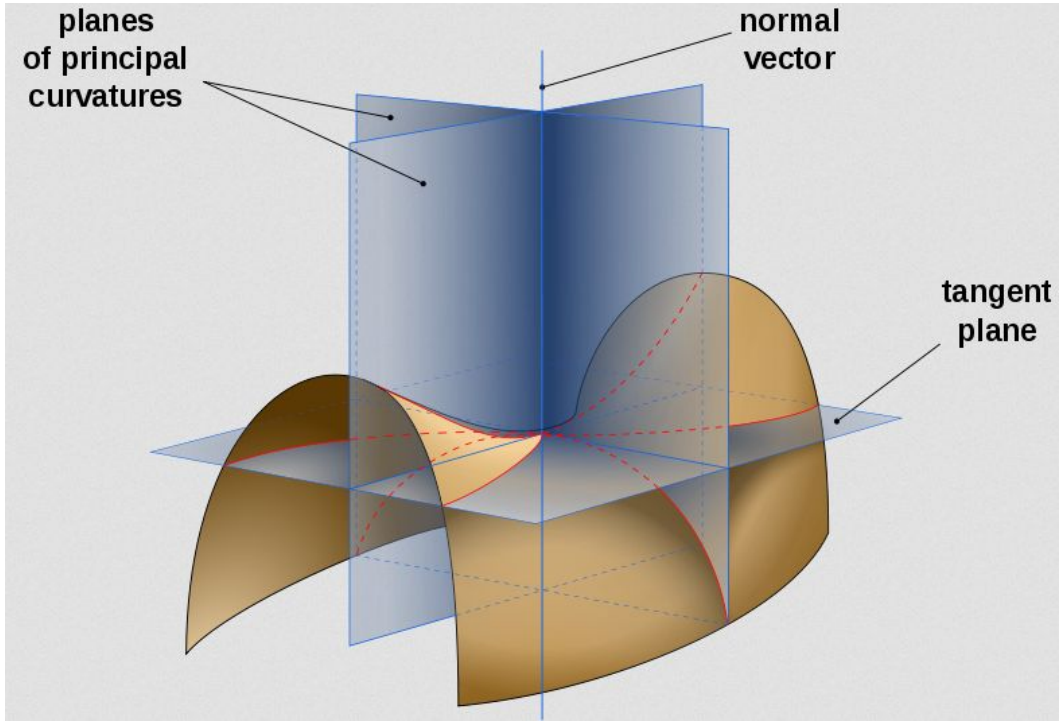
For sphere we have:

$$K = \frac{1}{r}\frac{1}{r} = \frac{1}{r^2}.$$

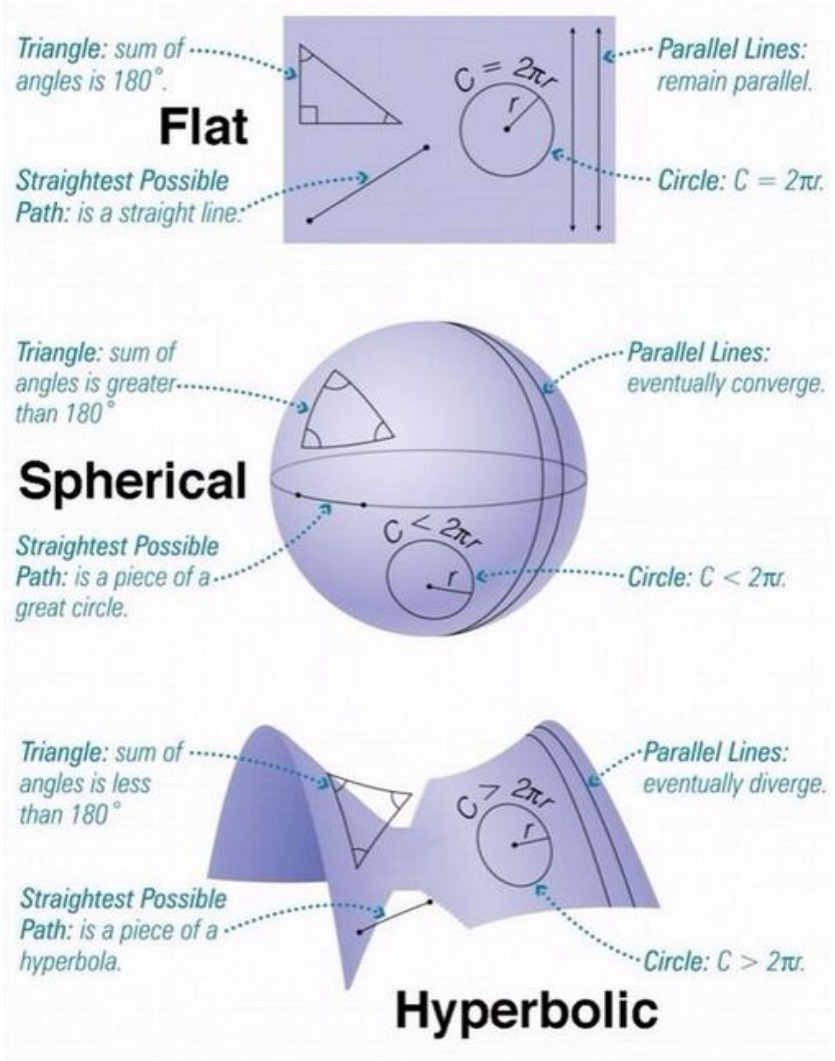
Understanding Poincare embeddings: Curved spaces

link to document

The curvature of the space we live in determines everything…..



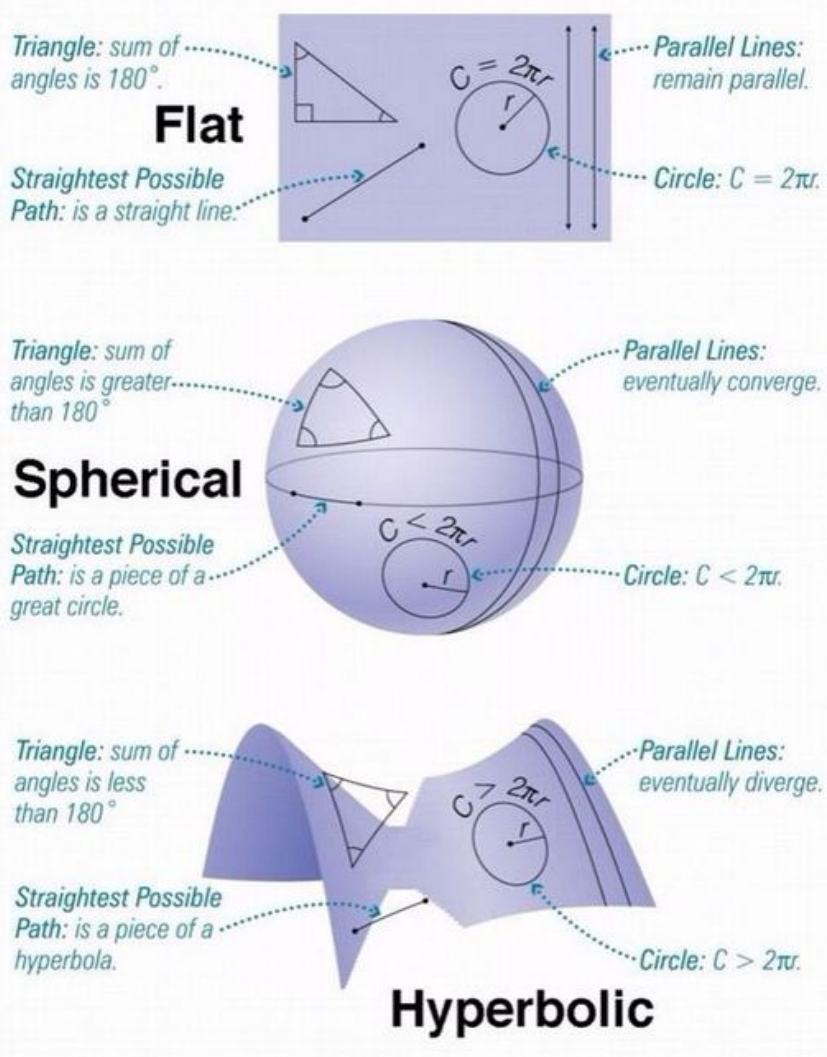
source



source

Understanding Poincare embeddings: The idea of embeddings

We want to embed curved space in our flat Euclides world



- We are going to put slowly all the pieces together
- Consider R^3 space.
- The simplest possible elliptic surface is sphere 2D with radius 1

$$x^2 + y^2 + z^2 = 1$$

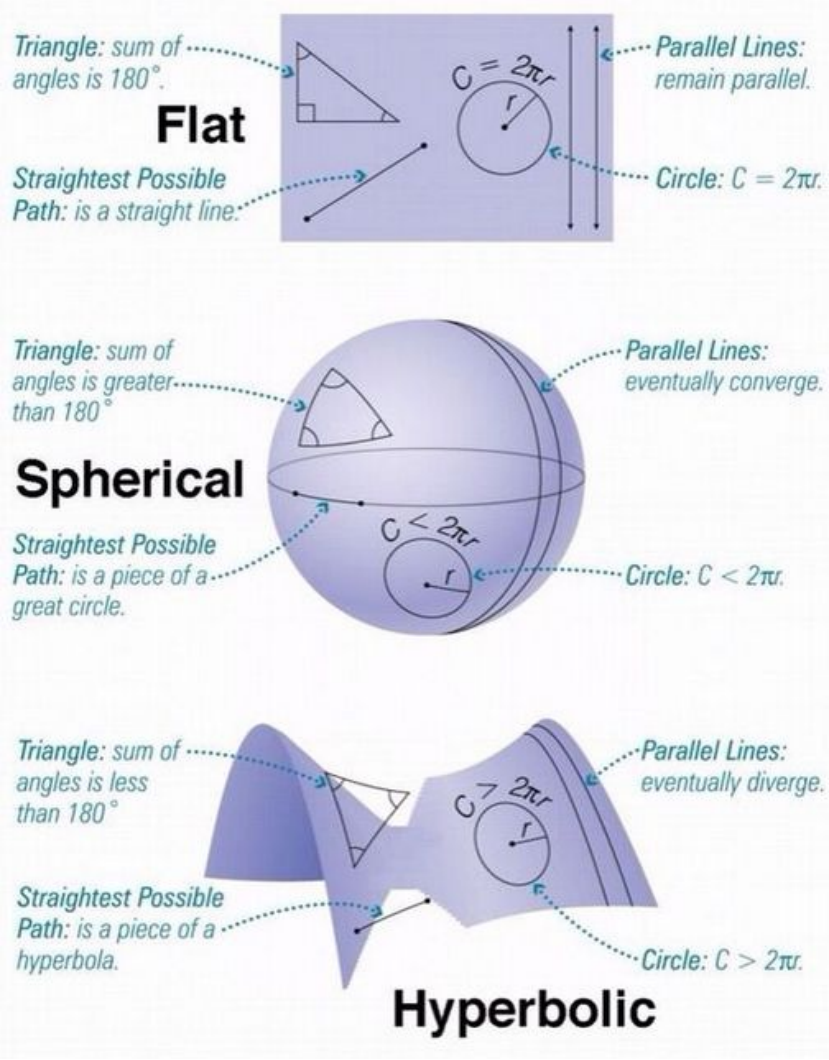
- It has K=+1
- In spherical coordinates we have (r=1)

$$x = r\ \sin\theta\ \cos\varphi$$
$$y = r\ \sin\theta\ \sin\varphi$$
$$z = r\ \cos\theta$$

- We have created a model for 2D sphere space embedded in euclidean space (note that dr=0)
- Then compute metric tensor, find geodesics (the arc of great circle), compute distances etc.

Understanding Poincare embeddings: The idea of embeddings

What about hyperbolic spaces?



- So far we have discussed flat spaces and 2D spheres.
- 2D sphere is the simplest example of elliptic space which we can visualize in our 3D world.
- With hyperbolic space **H** is much harder, we cannot visualize it easily in our world.
- Imagine that for **H** the K=-1, which means that in any position in the space it can be locally approximated by saddle point surface.
- We need models!!! Like Poincare disc model.
- Those models are not ideal: some of them don't preserve angles, some distances.
- Poincare disc model preserves angles (conformal model). Hence at least directions of gradients are correct (see discussion in paper).

Understanding Poincare embeddings: The hyperbolic models

How do we get hyperbolic models?

- Recall the Gauss curvature of sphere: $K = \frac{1}{r}\frac{1}{r} = \frac{1}{r^2}.$
- For hyperbolic space (with constant curvature) we want to have **K=-1**
- Let's put r=i (imaginary radius) then K=-1 - a pseudo sphere or hyperboloid model

$$\mathcal{H} = \{v \in \mathbb{L}^3 : \|v\| = i\}$$

a hyperboloid model

- We can embed it in cartesian coordinates (check x=y=0):

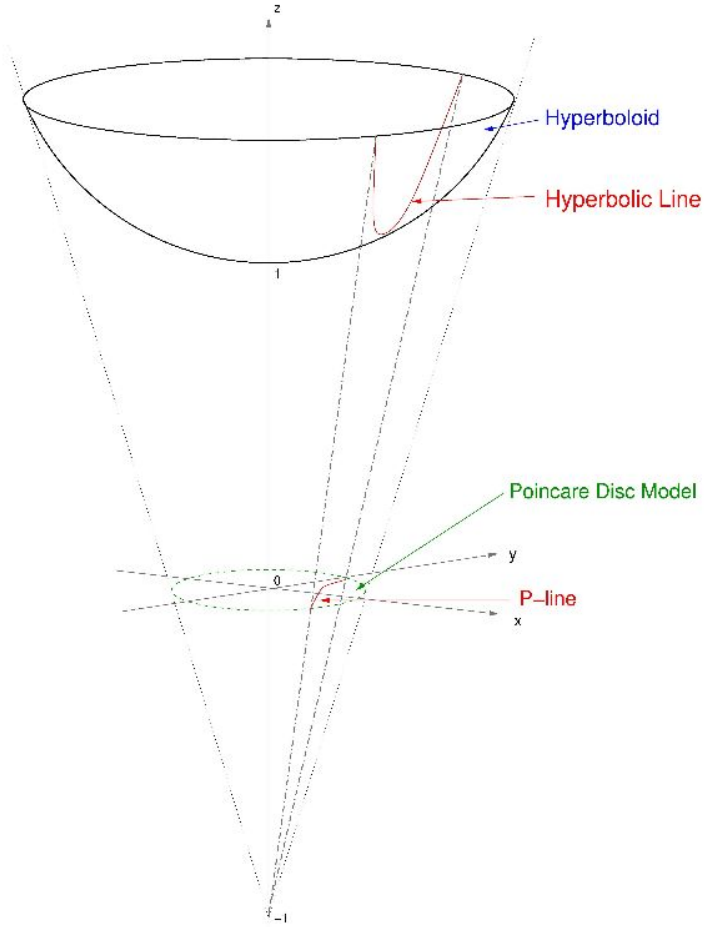
$$x^2 + y^2 - z^2 = 1$$

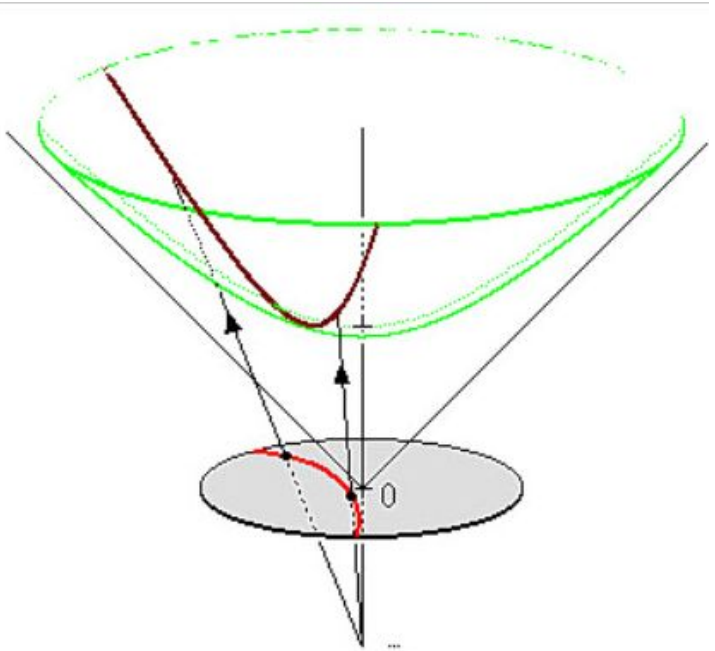
- This model is still hard to visualize :(Hard to work with 3 variables, we want less.

Understanding Poincare embeddings: The hyperbolic models

How do we get hyperbolic models?

- The Poincare disc model is obtained as stereographic projection of H model





The hyperboloid model can be represented as the equation $t^2=x_1^2+x_2^2+1$, $t>1$. It can be used to construct a Poincaré disk model as a projection viewed from $(t=-1,x_1=0,x_2=0)$, projecting the upper half hyperboloid onto the unit disk at t=0. The red geodesic in the Poincaré disk model projects to the brown geodesic on the green hyperboloid.

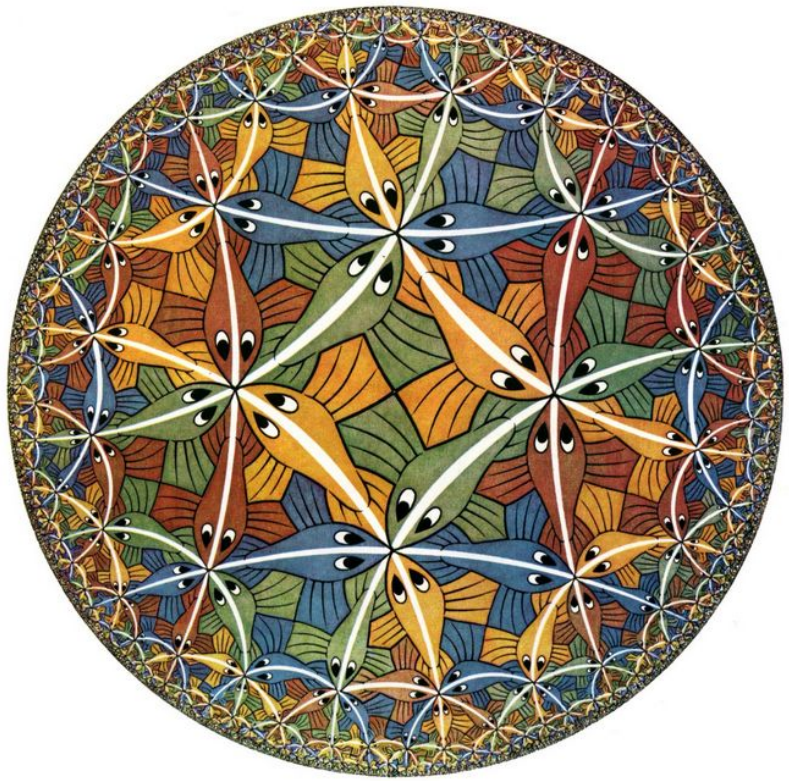
Figure 3.4: Stereographic projection of the hyperboloid to the Poincaré disc.

Understanding Poincare embeddings: The hyperbolic models

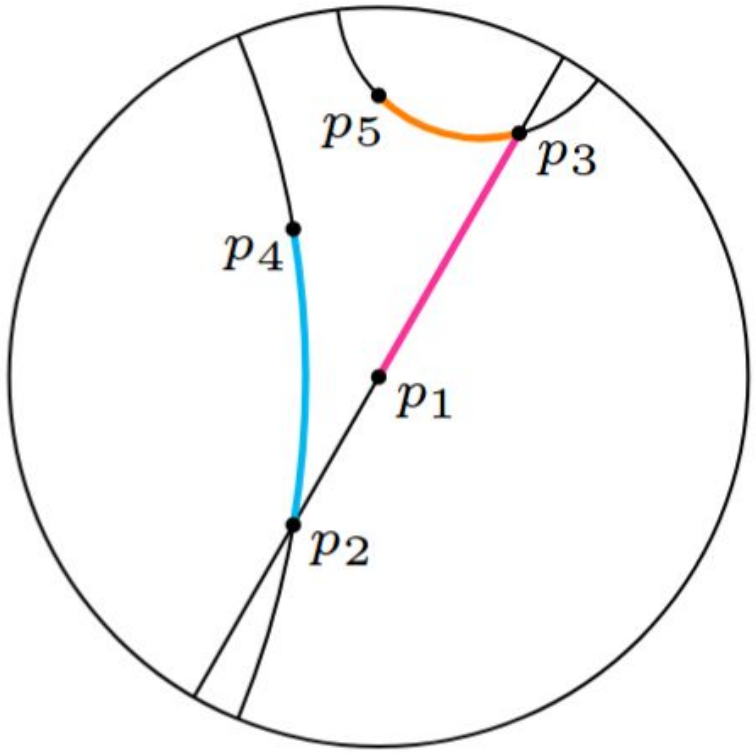
An analogy can be find in map projections: different projections lead to different surface parameterizations and apparent geodesics.

Poincare embeddings - big picture

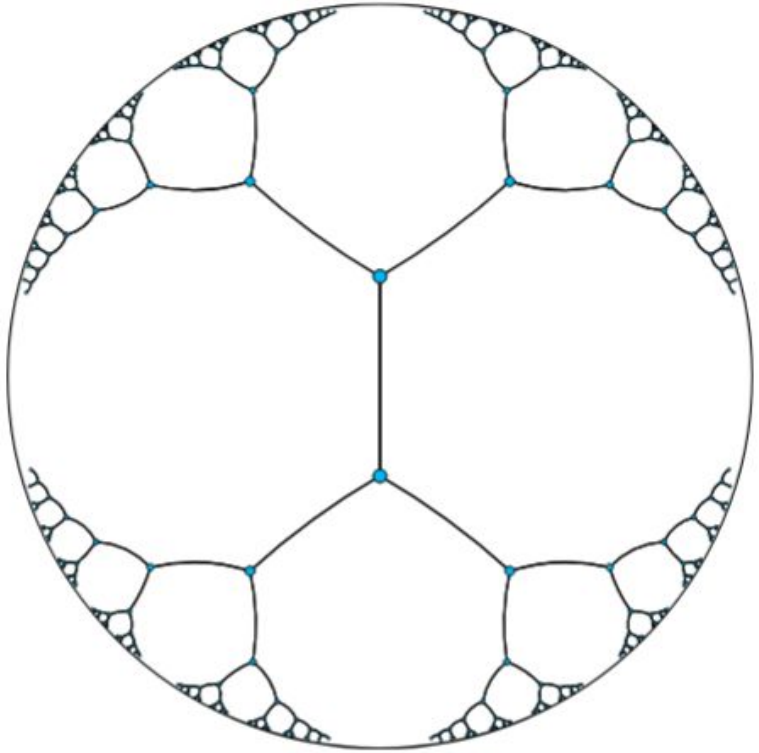
The idea is to use the **hyperbolic** space instead **euclidean** one



from Escher:
each fish has
the same area

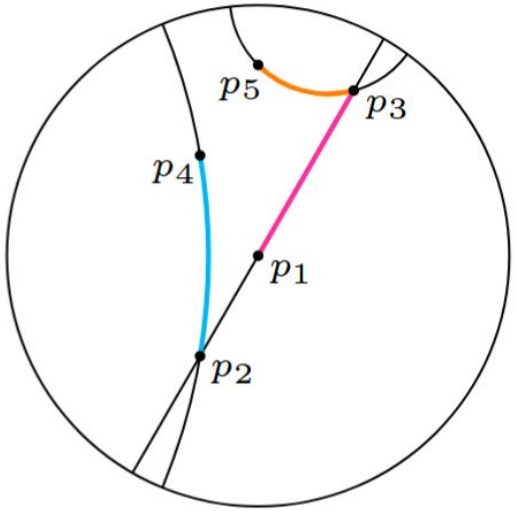


(a) Geodesics of the Poincaré disk



(b) Embedding of a tree in $\mathcal{B}^2$

Poincare embeddings - final summary



(a) Geodesics of the Poincaré disk

Distance in disc model is given by equation:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$

The authors provide optimization method

3.1 Optimization

Since the Poincaré Ball has a Riemannian manifold structure, we can optimize Equation (2) via stochastic Riemannian optimization methods such as RSGD [5] or RSVRG [34]. In particular, let $\mathcal{T}_\theta\mathcal{B}$ denote the tangent space of a point $\boldsymbol{\theta} \in \mathcal{B}^d$. Furthermore, let $\nabla_R \in \mathcal{T}_\theta\mathcal{B}$ denote the Riemannian gradient of $\mathcal{L}(\boldsymbol{\theta})$ and let ∇_E denote the Euclidean gradient of $\mathcal{L}(\boldsymbol{\theta})$. Using RSGD, parameter updates to minimize Equation (2) are then of the form

$$\boldsymbol{\theta}_{t+1} = \mathfrak{R}_{\theta_t}\left(-\eta_t \nabla_R \mathcal{L}(\boldsymbol{\theta}_t)\right)$$

where $\mathfrak{R}_{\theta_t}$ denotes the retraction onto $\mathcal{B}$ at $\boldsymbol{\theta}$ and η_t denotes the learning rate at time t. Hence, for the minimization of Equation (2), we require the Riemannian gradient and a suitable retraction. Since

Which simplifies to:

$$\operatorname{proj}(\boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta}/\|\boldsymbol{\theta}\| - \varepsilon & \text{if } \|\boldsymbol{\theta}\| \geq 1 \\ \boldsymbol{\theta} & \text{otherwise}, \end{cases}$$

where ε is a small constant to ensure numerical stability. In all experiments we used $\varepsilon = 10^{-5}$. In summary, the full update for a single embedding is then of the form

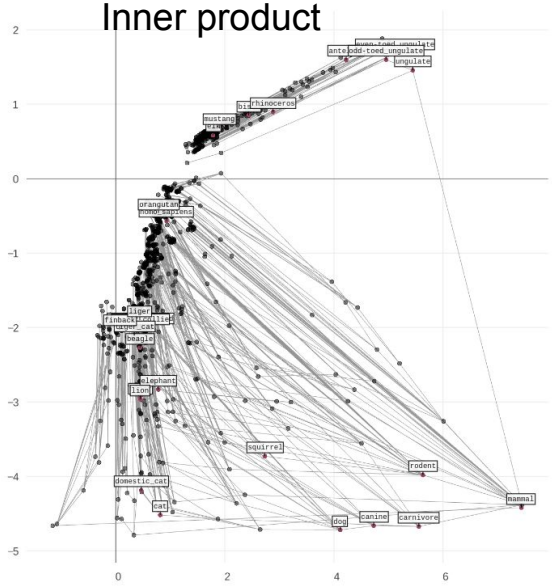
$$\boldsymbol{\theta}_{t+1} \leftarrow \operatorname{proj}\left(\boldsymbol{\theta}_t - \eta_t \frac{(1 - \|\boldsymbol{\theta}_t\|^2)^2}{4} \nabla_E\right). \tag{5}$$

Euclidean gradient $\nabla_E = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial d(\boldsymbol{\theta}, \boldsymbol{x})} \frac{\partial d(\boldsymbol{\theta}, \boldsymbol{x})}{\partial \boldsymbol{\theta}}$

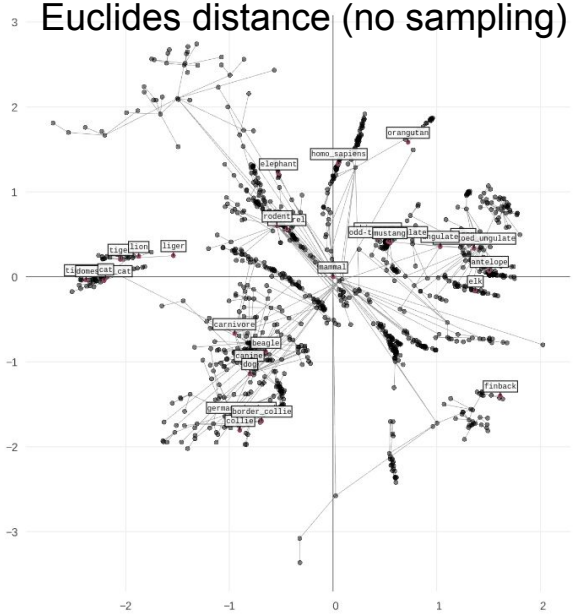
Experiments

My experiments - WDIL

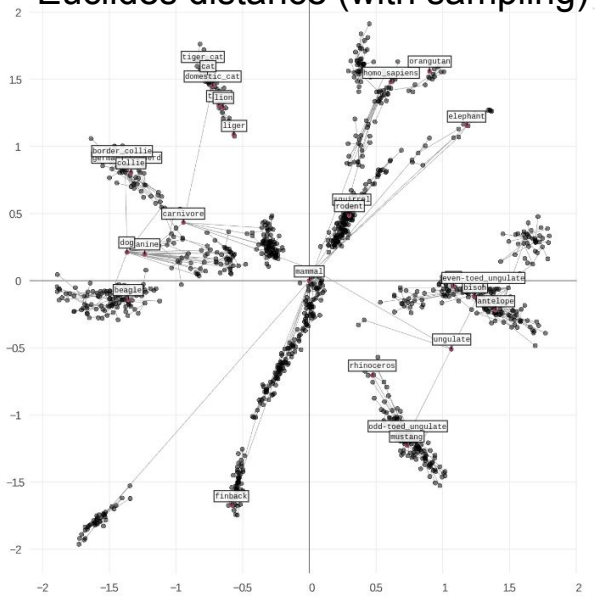
Inner product



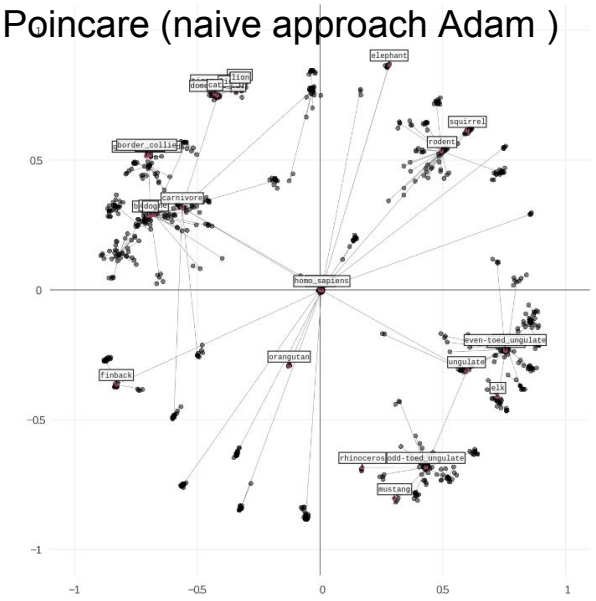
Euclides distance (no sampling)



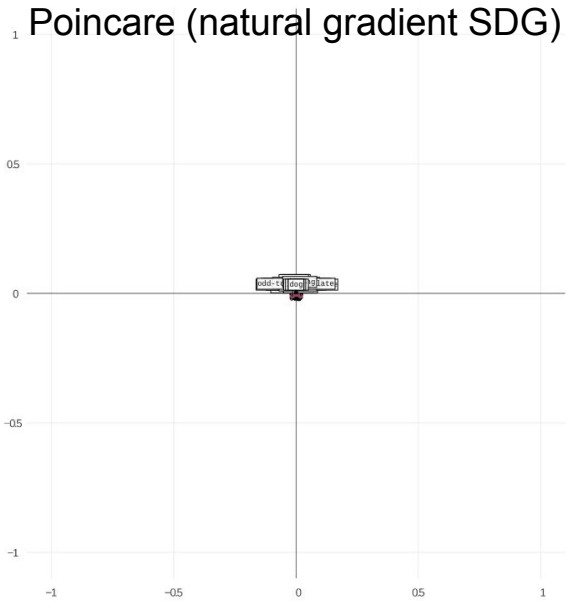
Euclides distance (with sampling)



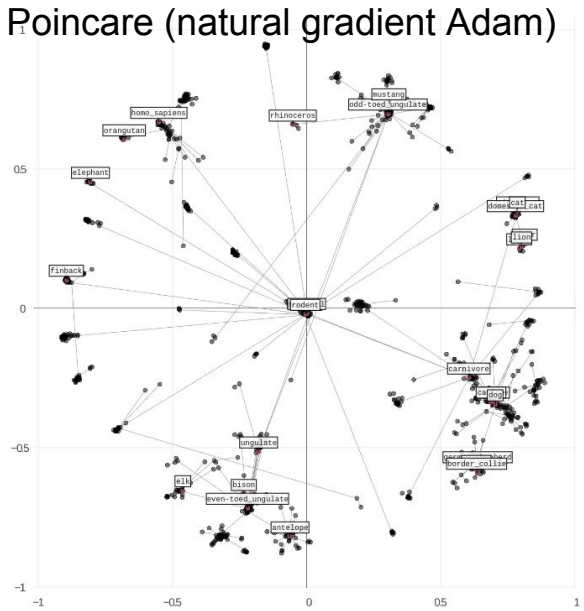
Poincare (naive approach Adam)



Poincare (natural gradient SDG)

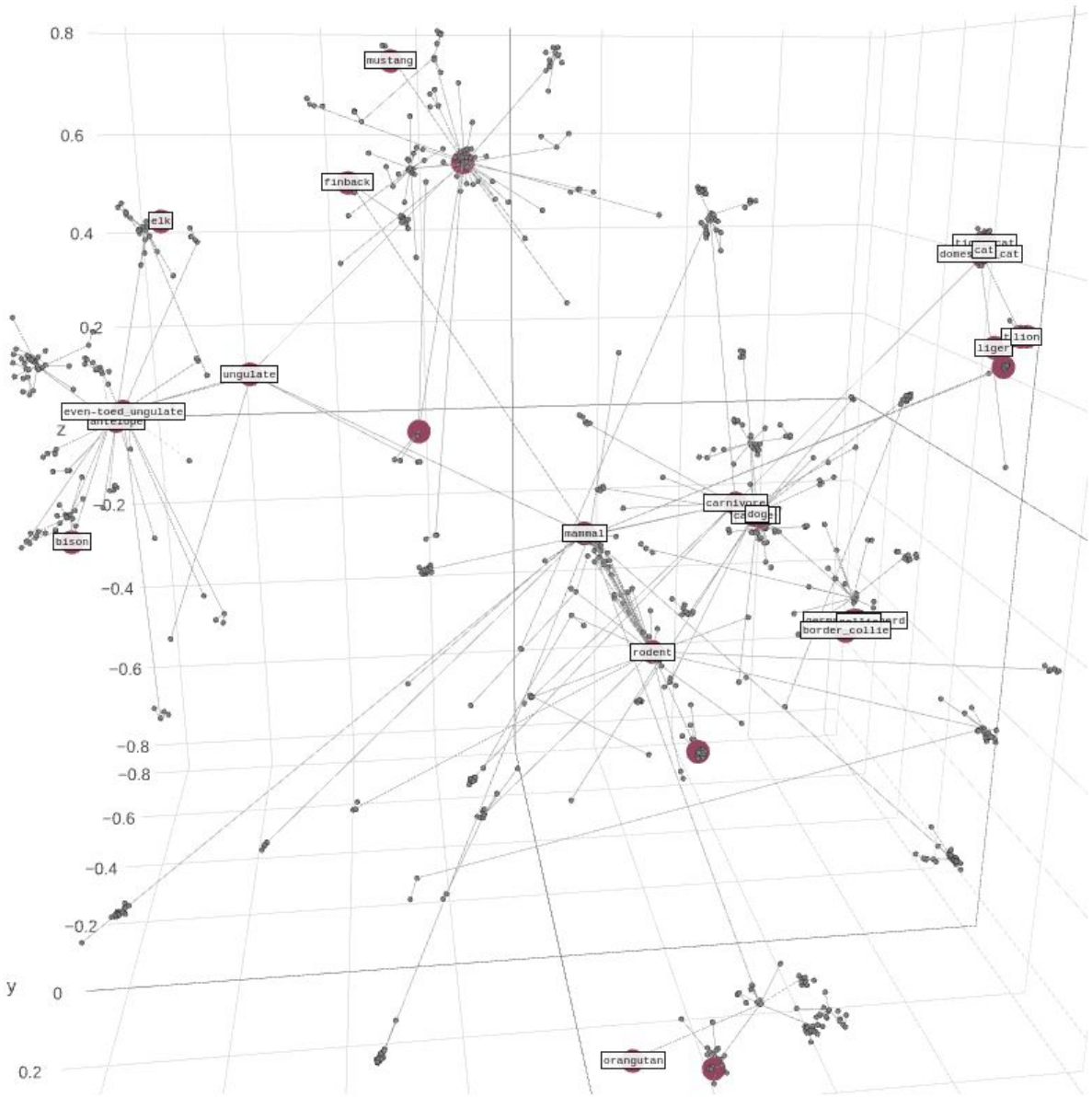


Poincare (natural gradient Adam)



My experiments - WDIL

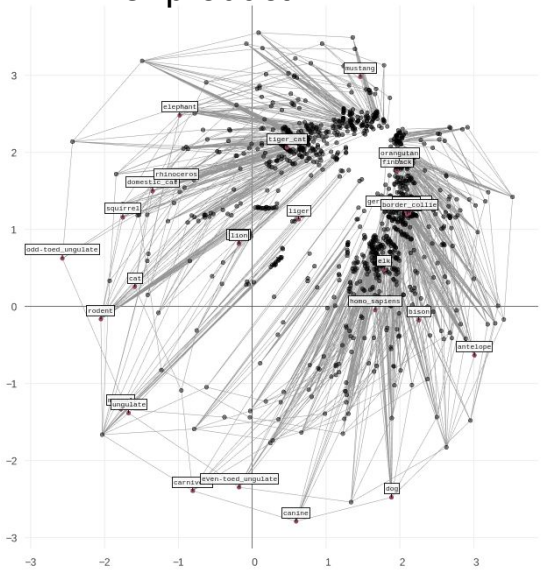
Poincare 3D - Adam



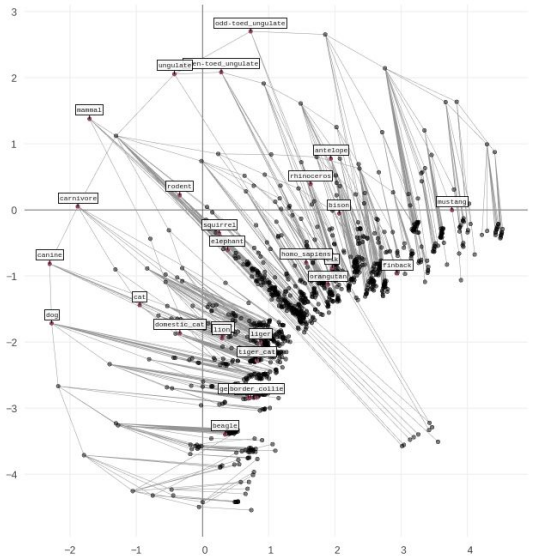
My experiments - WDIL

Another approach - trained on pairs but only nearest neighbors

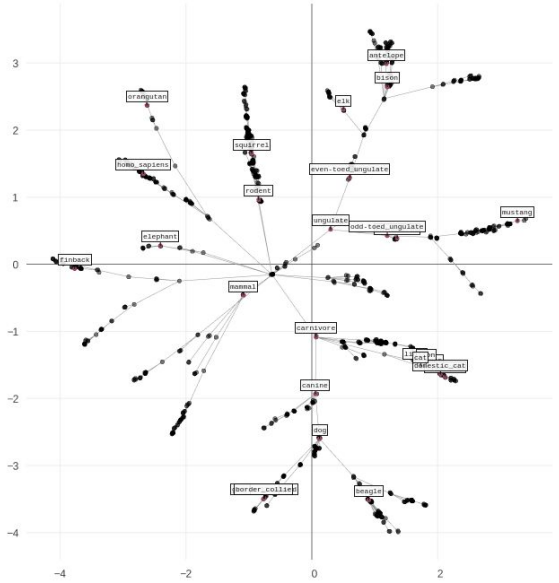
Inner product



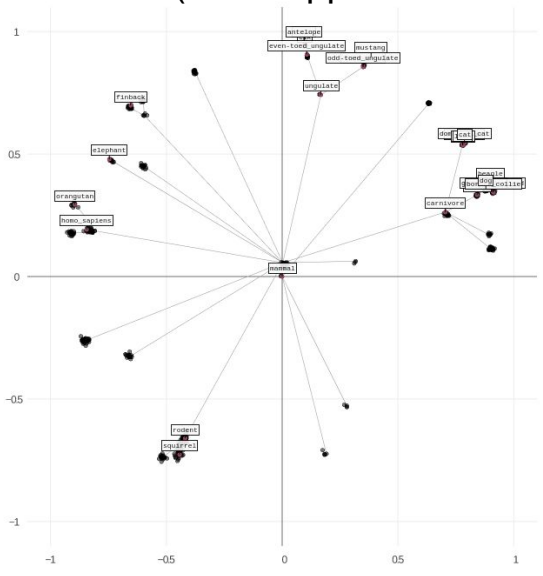
Euclides distance (no sampling)



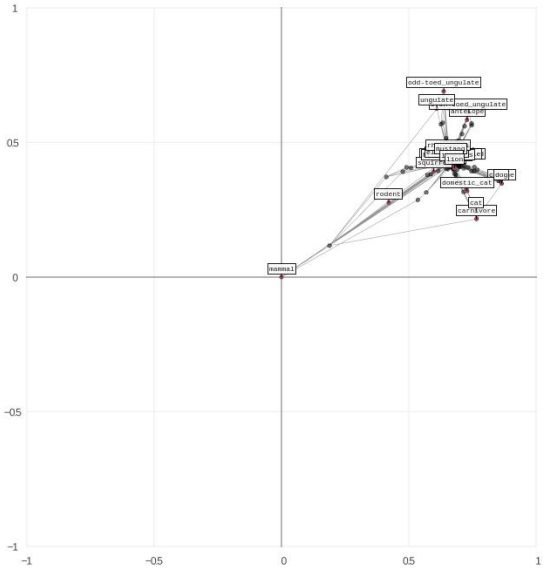
Euclides distance (with sampling)



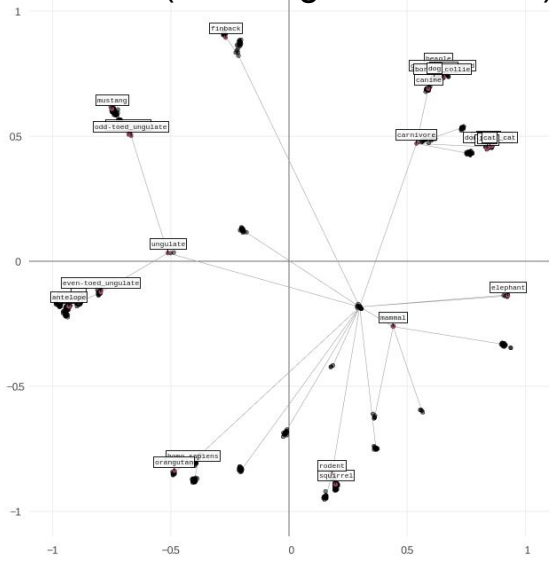
Poincare (naive approach Adam)



Poincare (natural gradient SDG)



Poincare (natural gradient Adam)



My experiments - WDIL

Keras customized SGD implementation and possible problem with word embeddings efficient update

```
class SGDPoincare(SGD):

    def get_updates(self, params, constraints, loss):

        grads = self.get_gradients(loss, params)
        self.updates = []

        lr = self.lr
        if self.initial_decay > 0:
            lr *= (1. / (1. + self.decay * self.iterations))
            self.updates .append(K.update_add(self.iterations, 1))

        # momentum
        shapes = [K.get_variable_shape(p) for p in params]
        moments = [K.zeros(shape) for shape in shapes]
        self.weights = [self.iterations] + moments

        for p, g, m in zip(params, grads, moments):

            normalization, p_norm = get_normalization(p)
            g = normalization * g

            v = self.momentum * m - lr * g  # velocity
            self.updates.append(K.update(m, v))

            if self.nesterov:
                new_p = p + self.momentum * v - lr * g
            else:
                new_p = p + v

            new_p = project(new_p, p_norm)

            # apply constraints
            if p in constraints:
                c = constraints[p]
                new_p = c(new_p)

            self.updates.append(K.update(p, new_p))
        return self.updates
```

project(...)

$$\text{proj}(\boldsymbol{\theta}) = \begin{cases} \boldsymbol{\theta}/\|\boldsymbol{\theta}\| - \varepsilon & \text{if } \|\boldsymbol{\theta}\| \geq 1 \\ \boldsymbol{\theta} & \text{otherwise}, \end{cases}$$

where ε is a small constant to ensure numerical stability. In all experiments we used $\varepsilon = 10^{-5}$. In summary, the full update for a single embedding is then of the form

$$\boldsymbol{\theta}_{t+1} \leftarrow \text{proj}\left(\boldsymbol{\theta}_t - \eta_t \frac{(1 - \|\boldsymbol{\theta}_t\|^2)^2}{4} \nabla_E\right). \quad (5)$$

normalization

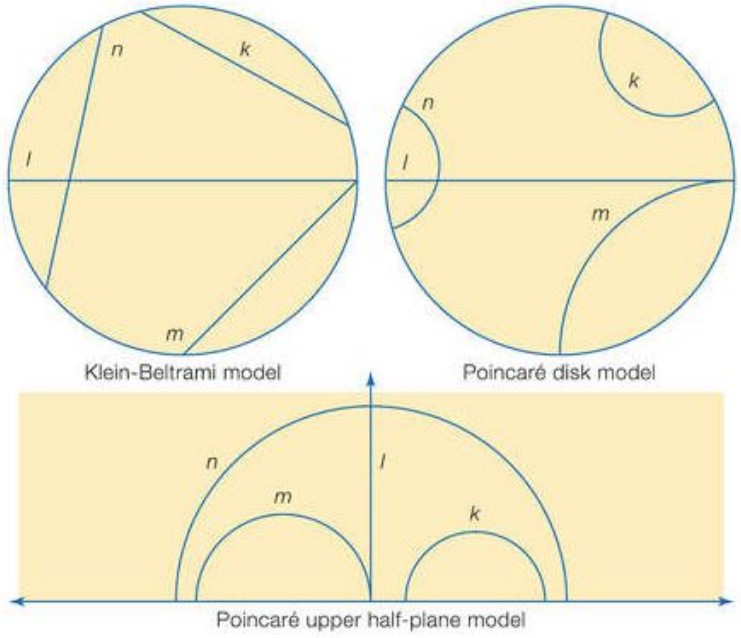
At this point g is an indexes tensor, however after assignment it becomes regular dense one. Does it mean that in Keras this kind of update is not efficient???

Epilog

The epilog - inspired by

Fisher information distance:a geometrical reading

Poincare disc model is not the only one we have: the other one is e.g. upper half plane model



A Mobius transform is used to derive Poincare disc model from Half Plane

$$\phi(z) = \frac{iz + 1}{-(z + i)}$$

Half Plane model arises also when one consider a measure of distance between two gaussians

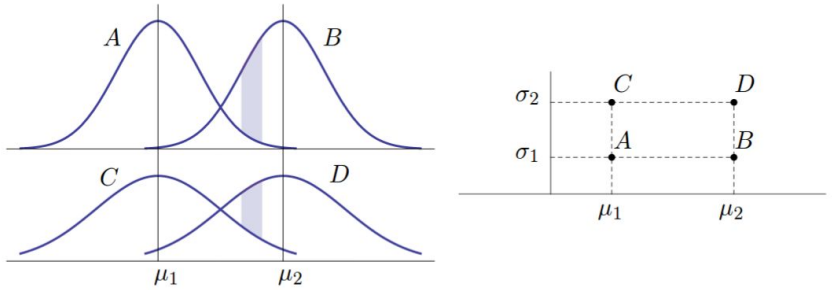
2.1 The hyperbolic model of the mean × standard deviation half-plane

The geometric model of the mean × standard deviation half-plane associates each point in the half upper plane of $\mathbb{R}^2$ with a univariate Gaussian PDF

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-|x - \mu|^2}{2\sigma^2}\right).$$

Hence, a classic parametric space for this family of PDF's is

$$H = \{(\mu, \sigma) \in \mathbb{R}^2 \mid \sigma > 0\}.$$



The epilog - inspired by

Fisher information distance:a geometrical reading

Poincare disc model is not the only one we have: the other one is e.g. upper half plane model

A proper distance arises from the Fisher information matrix, which is a measure of the amount of information of the location parameter ([11], ch. 12). For univariate distributions parametrized by an n-dimensional space, the coefficients of this matrix, which define a metric, are calculated as the expectation of a product involving partial derivatives of the logarithm of the PDF's:

$$g_{ij}(\boldsymbol{\beta}) = \int_{-\infty}^{\infty} f(x, \boldsymbol{\beta}) \frac{\partial \ln f(x, \boldsymbol{\beta})}{\partial \beta_i} \frac{\partial \ln f(x, \boldsymbol{\beta})}{\partial \beta_j} dx.$$

A metric matrix $G = (g_{ij})$ defines an inner product as follows:

$$\langle u, v \rangle_G = u^T (g_{ij}) v \quad \text{and} \quad \|u\|_G = \sqrt{\langle u, u \rangle_G}.$$

Can we use this metric for B**E**GANs instead of WGAN distance ???

By recalling that the Fisher distance d_F and the hyperbolic distance d_H are related by ([4]) we obtain the following closed expression for the Fisher information distance:

$$d_F((\mu_1, \sigma_1), (\mu_2, \sigma_2)) = \sqrt{2} \ln \frac{\left|(\frac{\mu_1}{\sqrt{2}}, \sigma_1) - (\frac{\mu_2}{\sqrt{2}}, -\sigma_2)\right| + \left|(\frac{\mu_1}{\sqrt{2}}, \sigma_1) - (\frac{\mu_2}{\sqrt{2}}, \sigma_2)\right|}{\left|(\frac{\mu_1}{\sqrt{2}}, \sigma_1) - (\frac{\mu_2}{\sqrt{2}}, -\sigma_2)\right| - \left|(\frac{\mu_1}{\sqrt{2}}, \sigma_1) - (\frac{\mu_2}{\sqrt{2}}, \sigma_2)\right|} \tag{5}$$

Questions?

References

- https://arxiv.org/pdf/1705.08039.pdf - Poincaré Embeddings for Learning Hierarchical Representations
- https://arxiv.org/pdf/1705.10359v1.pdf - Neural Embeddings of Graphs in Hyperbolic Space
- http://web.colby.edu/thegeometricviewpoint/category/hyperbolic-geometry/ - nice and easy article about hyperbolic geometry
- https://en.wikipedia.org/wiki/Poincar%C3%A9_disk_model - Poincaré disk model
- https://arxiv.org/pdf/1704.08059.pdf - Riemannian Optimization for Skip-Gram Negative Sampling

References

- http://sma.epfl.ch/~anchpcommon/publications/tensorcompletion.pdf - Riemanian optimization
- https://arxiv.org/pdf/1704.08059.pdf - Riemannian Optimization for Skip-Gram Negative Sampling
- http://press.princeton.edu/chapters/absil/Absil_Chap4.pdf - Retraction
- Natural Gradient Works Efficiently in Learning - paper, definitely nice to start with. Easy to read.
- http://www.maths.dur.ac.uk/Ug/projects/highlights/CM3/Hayter_Hyperbolic_report.pdf mobius transform, good almost step by step theory. Nice to start with.
- http://math2.uncc.edu/~frothe/3181all.pdf - book about general geometry about 800pages
- http://www.maths.manchester.ac.uk/~khudian/Teaching/Geometry/GeomRim11/riemgeom11.pdf

References

- https://sites.uclouvain.be/absil/Publi/OOM_BFG09/Absil_BFG09_08PA_UCL-INMA-2009-043-v2.pdf - slides from some workshop.
- http://www.yaroslavvb.com/papers/amari-why.pdf - probably the most easiest article I have found.
- https://perso.uclouvain.be/pa.absil/Talks/IAP_Study_Day_070416_oom_03.pdf - A little bit about retraction
- https://arxiv.org/pdf/1210.2354.pdf - Poincare Half plane analogy on difference between two gaussians
- http://image.diku.dk/MLLab/IG4.php - large amount of materials

FORNAX

WWW.FORNAX.AI