Poincare algorithm:

In [22]:

```
import nltk
from nltk.corpus import wordnet as wn
from math import *
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
```

In [23]:

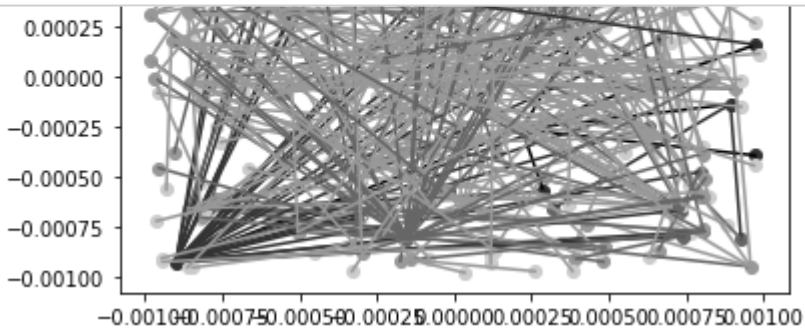
```
STABILITY = 0.00001 # to avoid overflow while dividing
network = {} # representation of network (here it is hierarchical)

last_level = 4

# plots the embedding of all the nodes of network
def plotall(ii):
    fig = plt.figure()
    # plot all the nodes
    for a in emb:
        plt.plot(emb[a][0], emb[a][1], marker = 'o', color = [levelOfNode[a]/(last_level+1)
    # plot the relationship, black line means root level relationship
    # consecutive relationship lines fade out in color
    for a in network:
        for b in network[a]:
            plt.plot([emb[a][0], emb[b][0]], [emb[a][1], emb[b][1]], color = [levelOfNode[a
    # plt.show()
    fig.savefig("adam" + str(last_level) + '_' + str(ii) + '.png', dpi=fig.dpi)

# network: the actual network of which node is connected to whom
# levelOfNode: level of the node in hierarchical data
levelOfNode = {}

# recursive function to popoulate the hyponyms of a root node in `network`
# synset: the root node
# last_level: the level till which we consider the hyponyms
def get_hyponyms(synset, level):
    if (level == last_level):
        levelOfNode[str(synset)] = level
        return
    # BFS
    if not str(synset) in network:
        network[str(synset)] = [str(s) for s in synset.hyponyms()]
        levelOfNode[str(synset)] = level
    for hyponym in synset.hyponyms():
        get_hyponyms(hyponym, level + 1)

mammal = wn.synset('mammal.n.01')
get_hyponyms(mammal, 0)
levelOfNode[str(mammal)] = 0

# embedding of nodes of network
emb = {}

# Randomly uniform distribution
for a in network:
    for b in network[a]:
        emb[b] = np.random.uniform(low=-0.001, high=0.001, size=(2,))
    emb[a] = np.random.uniform(low=-0.001, high=0.001, size=(2,))

vocab = list(emb.keys())
random.shuffle(vocab)

# the leave nodes are not connected to anything
for a in emb:
    if not a in network:
        network[a] = []
```

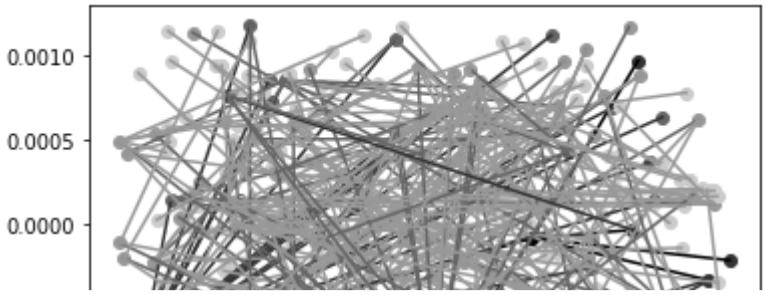
```
# Partial derivative as given in the paper wrt theta
def partial_der(theta, x, gamma): #eqn4
    alpha = (1.0-np.dot(theta, theta))
    norm_x = np.dot(x, x)
    beta = (1-norm_x)
    gamma = gamma
    return 4.0/(beta * sqrt(gamma*gamma - 1) + STABILITY)*((norm_x- 2*np.dot(theta, x)+1)/(

# a little modified update equation for adam
def update(emb, error_): #eqn5
    try:
        update =  pow((1 - np.dot(emb,emb)), 2)*error_/4
        # print (update)
        emb = emb - update
        if (np.dot(emb, emb) >= 1):
            emb = emb/sqrt(np.dot(emb, emb)) - 0.00001
        return emb
    except Exception as e:
        print (e)
        temp = input()

# Distance in poincare disk model
def dist(vec1, vec2): # eqn1
    return 1 + 2*np.dot(vec1 - vec2, vec1 - vec2)/ \
             ((1-np.dot(vec1, vec1))*(1-np.dot(vec2, vec2)) + STABILITY)
```

In [24]:

```
num_negs = 5

plotall("init")

# adam variables as per adam paper
update_emb = emb.copy()

m = {}
v = {}
t = {}
for a in emb:
    m[a] = np.asarray([0.0, 0.0])
    v[a] = np.asarray([0.0, 0.0])

eps = 0.00000001
b1 = 0.9
b2 = 0.999
b1_t = 0.9
b2_t = 0.999
alpha = 0.001


for epoch in range(200):
    print (epoch)
    # update_emb is the update we would like to make to an embedding
    for i in update_emb:
        update_emb[i] = np.zeros(2)
    # pos2 is related to pos1
    # negs are not related to pos1
    for pos1 in vocab:
        if not network[pos1]: # a leaf node
            continue
        pos2 = random.choice(network[pos1]) # pos2 and pos1 are related
        dist_p_init = dist(emb[pos1], emb[pos2]) # distance between the related nodes
        if (dist_p_init > 700): # this causes overflow, so I clipped it here
            print ("got one very high") # if you have reached this zone, the training is un
            dist_p_init = 700
        elif (dist_p_init < -700):
            print ("got one very high")
            dist_p_init = -700
        dist_p = cosh(dist_p_init) # this is the actual distance, it is always positive
        # print ("distance between related nodes", dist_p)
        negs = [] # pairs of not related nodes, the first node in the pair is `pos1`
        dist_negs_init = [] # distances without taking cosh on it (for not related nodes)
        dist_negs = [] # distances with taking cosh on it (for not related nodes)
        while (len(negs) < num_negs):
            neg1 = pos1
            neg2 = random.choice(vocab)
            if not (neg2 in network[neg1] or neg1 in network[neg2] or neg2 == neg1): # neg2
                dist_neg_init = dist(emb[neg1], emb[neg2])
                if (dist_neg_init > 700 or dist_neg_init < -700): # already dist is good, l
                    continue
                negs.append([neg1, neg2])
                dist_neg = cosh(dist_neg_init)
                dist_negs_init.append(dist_neg_init) # saving it for faster computation
                dist_negs.append(dist_neg)
                # print ("distance between non related nodes", dist_neg)
        loss_den = 0.0
```

```
        # eqn6
        for dist_neg in dist_negs:
            loss_den += exp(-1*dist_neg)
        loss = -1*dist_p - log(loss_den + STABILITY)
        # derivative of loss wrt positive relation [d(u, v)]
        der_p = -1
        der_negs = []
        # derivative of loss wrt negative relation [d(u, v')]
        for dist_neg in dist_negs:
            der_negs.append(exp(-1*dist_neg)/(loss_den + STABILITY))
        # derivative of loss wrt pos1
        der_p_pos1 = der_p * partial_der(emb[pos1], emb[pos2], dist_p_init)
        # derivative of loss wrt pos2
        der_p_pos2 = der_p * partial_der(emb[pos2], emb[pos1], dist_p_init)
        der_negs_final = []
        for (der_neg, neg, dist_neg_init) in zip(der_negs, negs, dist_negs_init):
            # derivative of loss wrt second element of the pair in neg
            der_neg1 = der_neg * partial_der(emb[neg[1]], emb[neg[0]], dist_neg_init)
            # derivative of loss wrt first element of the pair in neg
            der_neg0 = der_neg * partial_der(emb[neg[0]], emb[neg[1]], dist_neg_init)
            der_negs_final.append([der_neg0, der_neg1])
        # update embeddings
        update_emb[pos1] -= der_p_pos1
        update_emb[pos2] -= der_p_pos2
        for (neg, der_neg) in zip(negs, der_negs_final):
            update_emb[neg[0]] -= der_neg[0]
            update_emb[neg[1]] -= der_neg[1]
    # adam update now
    b1_t *= b1
    b2_t *= b2
    for i in update_emb:
        m[i] = b1*m[i] + (1.0 - b1) * update_emb[i]
        v[i] = b2*v[i] + (1.0 - b2) * update_emb[i] * update_emb[i]
        update_emb[i] =  alpha * (m[i] / (1.0 - b1_t)) / np.sqrt((v[i] / (1.0 - b2_t)) + ep
        emb[i] = update(emb[i], update_emb[i])
    # plot the embeddings
    if ((epoch)%20 == 0):
        plotall(epoch+1)
```





Poincare using functions

In [25]:

```
#%load_ext autoreload
import autoreload
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
#%reload_ext autoreload %autoreload 2
import os
import logging
import numpy as np

from gensim.models.poincare import PoincareModel, PoincareKeyedVectors, PoincareRelations

logging.basicConfig(level=logging.INFO)
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [26]:

```
model = PoincareModel(train_data=[('node.1', 'node.2'), ('node.2', 'node.3')])
```

```
INFO:gensim.models.poincare:loading relations from train data..
INFO:gensim.models.poincare:loaded 2 relations from train data, 3 nodes
```

In [27]:

```
wordnet_mammal_file = os.path.join(os.getcwd(), 'wordnet_mammal_hypernyms.tsv')
```

In [28]:

```
model = PoincareModel(train_data=[('node.1', 'node.2'), ('node.2', 'node.3')])
```

```
INFO:gensim.models.poincare:loading relations from train data..
INFO:gensim.models.poincare:loaded 2 relations from train data, 3 nodes
```

In [29]:

```
relations = PoincareRelations(file_path=wordnet_mammal_file, delimiter='\t')
model = PoincareModel(train_data=relations)
```

```
INFO:gensim.models.poincare:loading relations from train data..
INFO:gensim.models.poincare:loaded 7724 relations from train data, 1182 node
s
```

In [30]:

```
model = PoincareModel(train_data=relations, size=2, burn_in=0)
model.train(epochs=1, print_every=500)
```

```
INFO:gensim.models.poincare:loading relations from train data..
INFO:gensim.models.poincare:loaded 7724 relations from train data, 1182 node
s
INFO:gensim.models.poincare:training model of size 2 with 1 workers on 7724
relations for 1 epochs and 0 burn-in epochs, using lr=0.10000 burn-in lr=0.0
1000 negative=10
INFO:gensim.models.poincare:starting training (1 epochs)----------------------
----------------------
INFO:gensim.models.poincare:training on epoch 1, examples #4990-#5000, loss:
23.57
INFO:gensim.models.poincare:time taken for 5000 examples: 0.97 s, 5131.37 ex
amples / s
INFO:gensim.models.poincare:training finished
```

In [31]:

```
model.train(epochs=1, print_every=500)
```

```
INFO:gensim.models.poincare:training model of size 2 with 1 workers on 7724
relations for 1 epochs and 0 burn-in epochs, using lr=0.10000 burn-in lr=0.0
1000 negative=10
INFO:gensim.models.poincare:starting training (1 epochs)----------------------
----------------------
INFO:gensim.models.poincare:training on epoch 1, examples #4990-#5000, loss:
22.37
INFO:gensim.models.poincare:time taken for 5000 examples: 0.96 s, 5195.32 ex
amples / s
INFO:gensim.models.poincare:training finished
```

In [32]:

```
# Saves the entire PoincareModel instance, the loaded model can be trained further
model.save(os.path.join(os.getcwd(),'test_model'))
# or use PoincareModel.load(os.path.join(os.getcwd(),'test_model'))
```

```
INFO:gensim.utils:saving PoincareModel object under C:\Users\SMH_2\Documents
\Anagha mam project\test_model, separately None
INFO:gensim.utils:not storing attribute _node_counts_cumsum
INFO:gensim.utils:not storing attribute _node_probabilities
INFO:gensim.utils:saved C:\Users\SMH_2\Documents\Anagha mam project\test_mod
el
```

In [10]:

```
# Saves only the vectors from the PoincareModel instance, in the commonly used word2vec for
model.kv.save_word2vec_format(os.path.join(os.getcwd(),'test_vectors'))
PoincareKeyedVectors.load_word2vec_format(os.path.join(os.getcwd(),'test_vectors'))
```

```
INFO:gensim.models.utils_any2vec:storing 1182x2 projection weights into C:\U
sers\SMH_2\Documents\Anagha mam project\test_vectors
INFO:gensim.models.utils_any2vec:loading projection weights from C:\Users\SM
H_2\Documents\Anagha mam project\test_vectors
INFO:gensim.models.utils_any2vec:loaded (1182, 2) matrix from C:\Users\SMH_2
\Documents\Anagha mam project\test_vectors
```

Out[10]:

```
<gensim.models.poincare.PoincareKeyedVectors at 0x23d8e9d7c48>
```

Train Poincare Model on WordNet data

In [11]:

```
relations = PoincareRelations(file_path=wordnet_mammal_file, delimiter='\t')
size=50
burn_in=0
workers=1 # multi-threaded version wasn't implemented yet
negative=15
epochs=100
print_every=500
batch_size=10
model = PoincareModel(train_data=relations, size=size, burn_in=burn_in, workers=workers, ne
model.train(epochs=epochs, print_every=print_every,batch_size=batch_size)
```

```
INFO:gensim.models.poincare:time taken for 5000 examples: 2.78 s, 1800.97
examples / s
INFO:gensim.models.poincare:training on epoch 89, examples #4990-#5000, lo
ss: 4.95
INFO:gensim.models.poincare:time taken for 5000 examples: 3.17 s, 1578.76
examples / s
INFO:gensim.models.poincare:training on epoch 90, examples #4990-#5000, lo
ss: 4.94
INFO:gensim.models.poincare:time taken for 5000 examples: 2.41 s, 2078.56
examples / s
INFO:gensim.models.poincare:training on epoch 91, examples #4990-#5000, lo
ss: 4.94
INFO:gensim.models.poincare:time taken for 5000 examples: 2.53 s, 1975.94
examples / s
INFO:gensim.models.poincare:training on epoch 92, examples #4990-#5000, lo
ss: 4.98
INFO:gensim.models.poincare:time taken for 5000 examples: 2.58 s, 1935.43
examples / s
INFO:gensim.models.poincare:training on epoch 93, examples #4990-#5000, lo
ss: 4.86
```

In [33]:

```
# Saves the entire PoincareModel instance, the loaded model can be trained further
model.save(os.path.join(os.getcwd(),'gensim_model_batch_size_10_burn_in_0_epochs_100_neg_15

# Saves only the vectors from the PoincareModel instance, in the commonly used word2vec for
model.kv.save_word2vec_format(os.path.join(os.getcwd(),'gensim_model_batch_size_10_burn_in_
PoincareKeyedVectors.load_word2vec_format(os.path.join(os.getcwd(),'gensim_model_batch_size
```

```
INFO:gensim.utils:saving PoincareModel object under C:\Users\SMH_2\Documents
\Anagha mam project\gensim_model_batch_size_10_burn_in_0_epochs_100_neg_15_d
im_50, separately None
INFO:gensim.utils:not storing attribute _node_counts_cumsum
INFO:gensim.utils:not storing attribute _node_probabilities
INFO:gensim.utils:saved C:\Users\SMH_2\Documents\Anagha mam project\gensim_m
odel_batch_size_10_burn_in_0_epochs_100_neg_15_dim_50
INFO:gensim.models.utils_any2vec:storing 1182x2 projection weights into C:\U
sers\SMH_2\Documents\Anagha mam project\gensim_model_batch_size_10_burn_in_0
_epochs_100_neg_15_dim_50_vectors
INFO:gensim.models.utils_any2vec:loading projection weights from C:\Users\SM
H_2\Documents\Anagha mam project\gensim_model_batch_size_10_burn_in_0_epochs
_100_neg_15_dim_50_vectors
INFO:gensim.models.utils_any2vec:loaded (1182, 2) matrix from C:\Users\SMH_2
\Documents\Anagha mam project\gensim_model_batch_size_10_burn_in_0_epochs_10
0_neg_15_dim_50_vectors
```

Out[33]:

```
<gensim.models.poincare.PoincareKeyedVectors at 0x23d986602c8>
```

What the embedding can be used for

In [13]:

```
# Load an example model
test_model_path = os.path.join(os.getcwd(), 'gensim_model_batch_size_10_burn_in_0_epochs_10
model = PoincareModel.load(test_model_path)
```

```
INFO:gensim.utils:loading PoincareModel object from C:\Users\SMH_2\Documents
\Anagha mam project\gensim_model_batch_size_10_burn_in_0_epochs_100_neg_15_d
im_50
INFO:gensim.utils:loading kv recursively from C:\Users\SMH_2\Documents\Anagh
a mam project\gensim_model_batch_size_10_burn_in_0_epochs_100_neg_15_dim_50.
kv.* with mmap=None
INFO:gensim.utils:setting ignored attribute _node_counts_cumsum to None
INFO:gensim.utils:setting ignored attribute _node_probabilities to None
INFO:gensim.utils:loaded C:\Users\SMH_2\Documents\Anagha mam project\gensim_
model_batch_size_10_burn_in_0_epochs_100_neg_15_dim_50
```

Simple operations

In [34]:

```
# Distance between any two nodes
#cosh distance
model.kv.distance('leopard.n.02', 'mammal.n.01')
```

Out[34]:

```
0.16969888876370057
```

In [35]:

```
# Nodes most similar to a given input node - distance
model.kv.most_similar('carnivore.n.01')
```

Out[35]:

```
[('liver-spotted_dalmatian.n.01', 0.019930942259252987),
 ('large_poodle.n.01', 0.031878422913733495),
 ('bedlington_terrier.n.01', 0.037929437112001356),
 ('procyonid.n.01', 0.04125047907155917),
 ('paranthropus.n.01', 0.04199631498656061),
 ('australian_terrier.n.01', 0.043201999039537406),
 ('boarhound.n.01', 0.048827917859845374),
 ('european_hare.n.01', 0.0554946031893309),
 ('ice_bear.n.01', 0.05608605077263669),
 ('canine.n.02', 0.057727525123027826)]
```

In [16]:

```
# Rank of distance of node 2 from node 1 in relation to distances of all nodes from node 1
model.kv.rank('dog.n.01', 'carnivore.n.01')
```

Out[16]:

```
2
```

In [17]:

```
# Finding Poincare distance between input vectors
vector_1 = np.random.uniform(size=(100,)) # vector with 100 dim
vector_2 = np.random.uniform(size=(100,))
vectors_multiple = np.random.uniform(size=(5, 100)) # 5 vectors of 100 dim each

# Distance between vector_1 and vector_2
print(PoincareKeyedVectors.vector_distance(vector_1, vector_2))
# Distance between vector_1 and each vector in vectors_multiple
print(PoincareKeyedVectors.vector_distance_batch(vector_1, vectors_multiple))
```

```
0.24251971319121177
[0.26017805 0.25871261 0.26684892 0.27271526 0.27392107]
```

VISUALISATION