Importing Libraries

```
import torch
import torch.nn as NeuralNet
import torch.nn.functional as funct
from torch.autograd import Variable
import torch.utils.data as utils
import torch.utils.data as td
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import numpy as np
from numpy import dot
from numpy.linalg import norm
from queue import PriorityQueue
```

Defining neural network and its components

```
class neural_network(NeuralNet.Module):

    #h1 and h2 represent the two hidden layers of our network
    #Tanh is the activation function used
    def __init__(self, numoffeatures, h1, h2,output):
        super(neural_network, self).__init__()
        self.first   = NeuralNet.Linear(numoffeatures, h1)
        self.tanh1 = NeuralNet.Tanh()
        self.second   = NeuralNet.Linear(h1, h2)
        self.tanh2 = NeuralNet.Tanh()
        self.third   = NeuralNet.Linear(h2, output)

    def forward(self, x):
        out = self.first(x)
        out = self.tanh1(out)
        out = self.second(out)
        out = self.tanh2(out)
        out = self.third(out)

        return out
```

Defining important parameters

```
lr = 0.01
#this will make the first layer of our network
numoffeatures = 4
#number of nodes in 1st and 2nd hidden layers
h1 = 10
h2 = 5
#this will be last layer of our network
numofclasses = 3
#num of iterations for which we want to backpropagate
epochs = 25
```

Implementing the Model

```
m = neural_network(numoffeatures,h1,h2,numofclasses)

#choosing optimizer and desired loss function
criterion = NeuralNet.CrossEntropyLoss()
#Adam Algorithm is an algorithm for stochastic optimization
optimizer = torch.optim.Adam(m.parameters(), lr=lr)
```

Getting the dataset and splitting it

```
#load dataset
ds = load_iris()
x , y = ds.data, ds.target
#splitting into train and test as 0.8 and 0.2
x_train , x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)
#splitting test into test and validation as 0.1 and 0.1
x_test , x_val , y_test , y_val = train_test_split(x_test, y_test, test_size=0.5, random_stat
```

Creating datasets and loaders for train, validation and test data

```
#getting tensors

#Train
x_train_tnsr = torch.Tensor(x_train).float()
y_train_tnsr = torch.Tensor(y_train).long()

ds_train_tnsr = utils.TensorDataset(x_train_tnsr,y_train_tnsr)

train_data_loader = td.DataLoader(ds_train_tnsr, batch_size=10, shuffle=True, num_workers=1)
```

```
#Validation
x_val_tnsr = torch.Tensor(x_val).float()
y_val_tnsr = torch.Tensor(y_val).long()
ds_val_tnsr = utils.TensorDataset(x_val_tnsr,y_val_tnsr)
val_data_loader = td.DataLoader(ds_val_tnsr, batch_size=10, shuffle=True, num_workers=1)

#Test
x_test_tnsr = torch.Tensor(x_test).float()
y_test_tnsr = torch.Tensor(y_test).long()

ds_test_tnsr = utils.TensorDataset(x_test_tnsr,y_test_tnsr)

test_data_loader = td.DataLoader(ds_test_tnsr, batch_size=10, shuffle=True, num_workers=1)
```

Function to get train loss and validation loss

```
def train(model, loader, loader1, criterion, optimizer):

    model.train()

    t_loss = 0
    v_loss = 0

    for b, tnsr in enumerate(loader):
        curr_data, curr_target = tnsr
        #setting gradients to zero
        optimizer.zero_grad()
        next = model(curr_data)
        loss = criterion(next, curr_target)
        #calculating gradients
        loss.backward()
        #Updating weights
        optimizer.step()
        t_loss += loss.item()

    #evaluating our model
    model.eval()
    for b, tnsr in enumerate(loader1):
        curr_data, curr_target = tnsr
        next = model(curr_data)
        loss = criterion(next, curr_target)
        v_loss += loss.item()


    #Return loss
    avg_loss_t = t_loss / len(loader.dataset)
    avg_loss_v = v_loss / len(loader1.dataset)
    return avg_loss_t, avg_loss_v
```

Function to get test loss

```
def test(model, loader, criterion):

    #evaluate model
    model.eval()

    t_loss = 0
    count = 0

    with torch.no_grad():
        for b, tnsr in enumerate(loader):
            curr_data, curr_target = tnsr

            #find answer by passing data through model
            output = model(curr_data)

            #find loss
            t_loss += criterion(output, curr_target).item()

            #find accuracy
            _, prediction = torch.max(output, 1)
            count += torch.sum(curr_target == prediction).item()


    avg_acc = count / len(loader.dataset)
    avg_loss = t_loss / len(loader.dataset)

    return avg_acc, avg_loss
```

Using our model

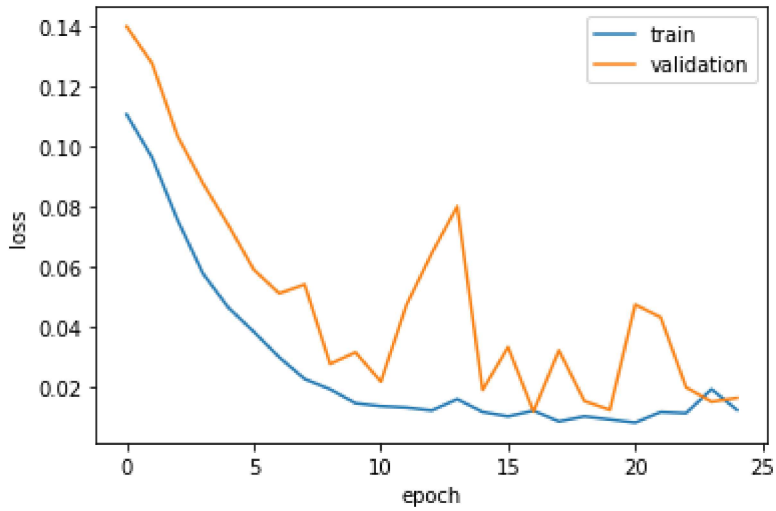
```
enums = []
tr_losses = []
val_losses = []


for epoch in range(epochs):

    #train our model
    tr_loss, val_loss = train(m, train_data_loader, val_data_loader, criterion, optimizer)

    enums.append(epoch)
    tr_losses.append(tr_loss)
    val_losses.append(val_loss)
```

```
print("Epoch ",epoch+1,":")
print("Training Loss :",tr_loss," Validation Loss :",val_loss)
```

```
Epoch  1 :
Training Loss : 0.11043494294087092  Validation Loss : 0.13962940375010172
Epoch  2 :
Training Loss : 0.09614286720752716  Validation Loss : 0.12741331656773885
Epoch  3 :
Training Loss : 0.07543891419967015  Validation Loss : 0.10331085522969564
Epoch  4 :
Training Loss : 0.05762854690353076  Validation Loss : 0.08755773703257243
Epoch  5 :
Training Loss : 0.04631331140796344  Validation Loss : 0.07364566326141357
Epoch  6 :
Training Loss : 0.03830399985114733  Validation Loss : 0.05893575350443522
Epoch  7 :
Training Loss : 0.029833713298042616  Validation Loss : 0.051031259695688884
Epoch  8 :
Training Loss : 0.022628767291704814  Validation Loss : 0.05400635202725728
Epoch  9 :
Training Loss : 0.019209833443164827  Validation Loss : 0.027625330289204917
Epoch  10 :
Training Loss : 0.014578550308942794  Validation Loss : 0.03145942290623983
Epoch  11 :
Training Loss : 0.013592283179362615  Validation Loss : 0.02173994282881419
Epoch  12 :
Training Loss : 0.013127941607187192  Validation Loss : 0.047159045934677124
Epoch  13 :
Training Loss : 0.012159376529355843  Validation Loss : 0.06436891158421834
Epoch  14 :
Training Loss : 0.015920428931713103  Validation Loss : 0.0799131174882253
Epoch  15 :
Training Loss : 0.011663611140102148  Validation Loss : 0.018979272246360777
Epoch  16 :
Training Loss : 0.010188808074841897  Validation Loss : 0.03317921062310537
Epoch  17 :
Training Loss : 0.012090176142131289  Validation Loss : 0.01178117295106252
Epoch  18 :
Training Loss : 0.008515627030283213  Validation Loss : 0.032126254340012865
Epoch  19 :
Training Loss : 0.010198805139710505  Validation Loss : 0.015311317642529805
Epoch  20 :
Training Loss : 0.009180616665010652  Validation Loss : 0.01235609215994676
Epoch  21 :
Training Loss : 0.008114123592774074  Validation Loss : 0.047323753436406456
Epoch  22 :
Training Loss : 0.01164391920901835  Validation Loss : 0.04311015009880066
Epoch  23 :
Training Loss : 0.011270690135036905  Validation Loss : 0.019843202829360963
Epoch  24 :
Training Loss : 0.019215710011000434  Validation Loss : 0.015068810433149338
Epoch  25 :
Training Loss : 0.012357099168002605  Validation Loss : 0.016311604777971903
```

Plotting and analysis

```
plt.plot(enums, tr_losses)
plt.plot(enums, val_losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'validation'])
plt.show()

accuracy, tst_loss = test(m, test_data_loader, criterion)

print("Accuracy :",accuracy * 100,"% Test Loss: ",tst_loss)

#set model to evaluate mode
m.eval()

#get predictions
x = torch.Tensor(x_test).float()
_, predicted = torch.max(m(x).data, 1)

#Getting confusion matrix
k = accuracy_score(y_test, predicted.numpy())

cm = confusion_matrix(y_test, predicted.numpy())
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Greens)
plt.colorbar()
tick_marks = np.arange(len(ds.target_names))
plt.xticks(tick_marks, ds.target_names)
plt.yticks(tick_marks, ds.target_names)
plt.xlabel("Predicted Species")
plt.ylabel("True Species")
plt.show()
```




```
Accuracy : 100.0 % Test Loss:  0.0025202353795369466
```

Reading data from vectors.txt

```
with open('vectors.txt') as f:
    lines = f.readlines()

all_words = {}

for line in lines:
  #deleting last char '\n'
  line = line[:-1]
  #splitting at spaces
  line = line.split(" ")
  #taking word and vectors
  word = line[0]
  vector = line[1:]
  #converting vector coordinates to floats
  for i in range(len(vector)):
    vector[i] = float(vector[i])

  all_words[word] = vector

print("Done!")
```

```
Done!
```

Function to get closest words

```
def get_20_closest(word):

    #converting word to lower case
    word = word.lower()
```

```
    #priority queue so we can pop the element with
    #smallest cosine when size exceeds 21
    q = PriorityQueue()
    for x in all_words:
        #calculate cosine
        cosine_sim = dot(all_words[x],all_words[word])/(norm(all_words[x])*norm(all_words[wor
        q.put((cosine_sim,x))
        if(q.qsize() > 21):
          q.get()

    closest = []
    while(q.empty() is False):
      #converting it to list
      closest.append(np.asarray(q.get())[::-1])

    #converting it to dataframe
    closest = pd.DataFrame(closest[::-1],columns = ['Words','Cosine Similarity Index'])
    return closest
```

Closest to 'life'

```
closest_to_life = get_20_closest('life')
closest_to_life
```

	Words	Cosine Similarity Index
0	life	1.0
1	mind	0.8514841863412058
2	love	0.8403438472391275
3	lives	0.8392689150907717
4	own	0.8369905081691961
5	kind	0.833887260989752

Closest to 'market'

```
closest_to_market = get_20_closest('market')
closest_to_market
```

	Words	Cosine Similarity Index
0	market	1.0
1	markets	0.9401178795959132

Closest to 'Stanford'

3	stock	0.8650148480690682

```
closest_to_stanford = get_20_closest('Stanford')
closest_to_stanford
```

	Words	Cosine Similarity Index
0	stanford	1.0
1	ucla	0.8524495684262241
2	harvard	0.8466463005377866
3	yale	0.8393530271238332
4	princeton	0.8349352254305833
5	rutgers	0.8128204094909456
6	university	0.7906179668757196
7	baylor	0.7722881793675541
8	graduate	0.7668696881925167
9	georgetown	0.763642714665397
10	cornell	0.7606510734805616
11	fordham	0.7571069148302746
12	asu	0.754074594688501
13	usc	0.7407924400170456
14	uc	0.7352783869511291
15	hopkins	0.7346757278421222
16	usf	0.7338851710107053
17	professor	0.7232831200910937
18	berkeley	0.722014733764268
19	college	0.7217255393021689
20	villanova	0.7199799887991107

Uncomment this and run with a valid input in order to find 20 closest words of that input

```
# closest_words_to_this = get_20_closest('*enter_word_present_in_vectors.txt_here*')
# closest_words_to_this
```

Select GPU as runtime type in colab before running this

TSNE visualization for all points

```
import pandas as pd

df = pd.DataFrame.from_dict(all_words, orient='index')
df = df[:-1]
```

```
df
```

	0	1	2	3	4	5	6	
the	0.418000	0.249680	-0.41242	0.121700	0.34527	-0.044457	-0.496880	-0.1786
,	0.013441	0.236820	-0.16899	0.409510	0.63812	0.477090	-0.428520	-0.5564
.	0.151640	0.301770	-0.16763	0.176840	0.31719	0.339730	-0.434780	-0.3108
of	0.708530	0.570880	-0.47160	0.180480	0.54449	0.726030	0.181570	-0.5239
to	0.680470	-0.039263	0.30186	-0.177920	0.42962	0.032246	-0.413760	0.1322
...	...	...	...	...	...	...	...	
chanty	0.232040	0.025672	-0.70699	-0.045465	0.13989	-0.628070	0.726250	0.3410
kronik	-0.609210	-0.672180	0.23521	-0.111950	-0.46094	-0.007462	0.255780	0.8563
rolonda	-0.511810	0.058706	1.09130	-0.551630	-0.10249	-0.126500	0.995030	0.0797
zsombor	-0.758980	-0.474260	0.47370	0.772500	-0.78064	0.232330	0.046114	0.8401
sandberger	0.072617	-0.513930	0.47280	-0.522020	-0.35534	0.346290	0.232110	0.2309

400000 rows × 50 columns

```
!pip install -q condacolab
import condacolab
condacolab.install()
```

```
✨🏰✨ Everything looks OK!
```

```
!wget https://anaconda.org/CannyLab/tsnecuda/2.1.0/download/linux-64/tsnecuda-2.1.0-cuda100.t
!tar xvjf tsnecuda-2.1.0-cuda100.tar.bz2
!cp -r site-packages/* /usr/local/lib/python3.6/dist-packages/
```

```
lib/libgmock_main.a
lib/libgmock.a
lib/libgtest.a
info/recipe/README.md
info/recipe/meta.yaml.template
info/recipe/cmake/write_python_version_string.cmake
info/recipe/cross-linux.cmake
info/recipe/cmake/Modules/FindOpenBLAS.cmake
info/recipe/cmake/Modules/FindFFTW.cmake
info/recipe/cmake/Modules/FindMKL.cmake
info/recipe/src/python/tsnecuda/__init__.py
site-packages/tsnecuda/__init__.py
info/recipe/src/python/setup.py
info/recipe/visualization/vis_rt.py
info/recipe/visualization/visualize_mnist.py
info/recipe/visualization/visualize.py
info/recipe/docs/test_dist_cont.py
link.py
info/recipe/.ycm_extra_conf.py
info/recipe/src/python/tsnecuda/TSNE.py
site-packages/tsnecuda/TSNE.py
site-packages/tsnecuda/test/__init__.py
info/recipe/src/python/tsnecuda/test/__init__.py
info/recipe/src/python/MANIFEST.in
info/recipe/docs/Doxyfile.in
info/recipe/src/include/kernels/rep_forces.h
info/recipe/src/include/util/debug_utils.h
info/recipe/src/include/fit_tsne.h
info/recipe/src/include/kernels/apply_forces.h
info/recipe/src/include/kernels/attr_forces.h
info/recipe/src/include/util/random_utils.h
info/recipe/src/include/common.h
info/recipe/src/include/util/cuda_utils.h
info/recipe/src/include/util/thrust_transform_functions.h
info/recipe/src/include/kernels/perplexity_search.h
info/recipe/src/include/util/thrust_utils.h
info/recipe/src/include/ext/pymodule_ext.h
info/recipe/src/include/kernels/nbodyfft.h
info/recipe/src/include/util/matrix_broadcast_utils.h
info/recipe/src/include/util/data_utils.h
info/recipe/src/include/test/test_distance.h
info/recipe/src/include/util/reduce_utils.h
info/recipe/src/include/test/test_math.h
info/recipe/src/include/util/math_utils.h
info/recipe/src/include/util/distance_utils.h
info/recipe/src/include/test/test_reduce.h
info/recipe/src/include/options.h
info/recipe/src/include/test/test_tsne.h
info/recipe/docs/results/benchmark_results.pkl
lib/libfaiss.so
site-packages/tsnecuda/libtsnecuda.so
info/recipe/.gitignore
```

```
info/recipe/.gitmodules
info/recipe/LICENSE
info/recipe/src/python/MANIFEST
Scripts/.tsnecuda-pre-link.bat
```

```
!conda install --offline /content/tsnecuda-2.1.0-cuda100.tar.bz2
```

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

```
import tsnecuda
tsnecuda.test()
```

Making an array of colors

```
import numpy as np
colors = []
for i in range(256):
  for j in range(256):
    for k in range(256):
      colors.append(np.asarray((i,j,k)))

# import random
# random.shuffle(colors)

colors = np.asarray(colors)
```

Using TSNE

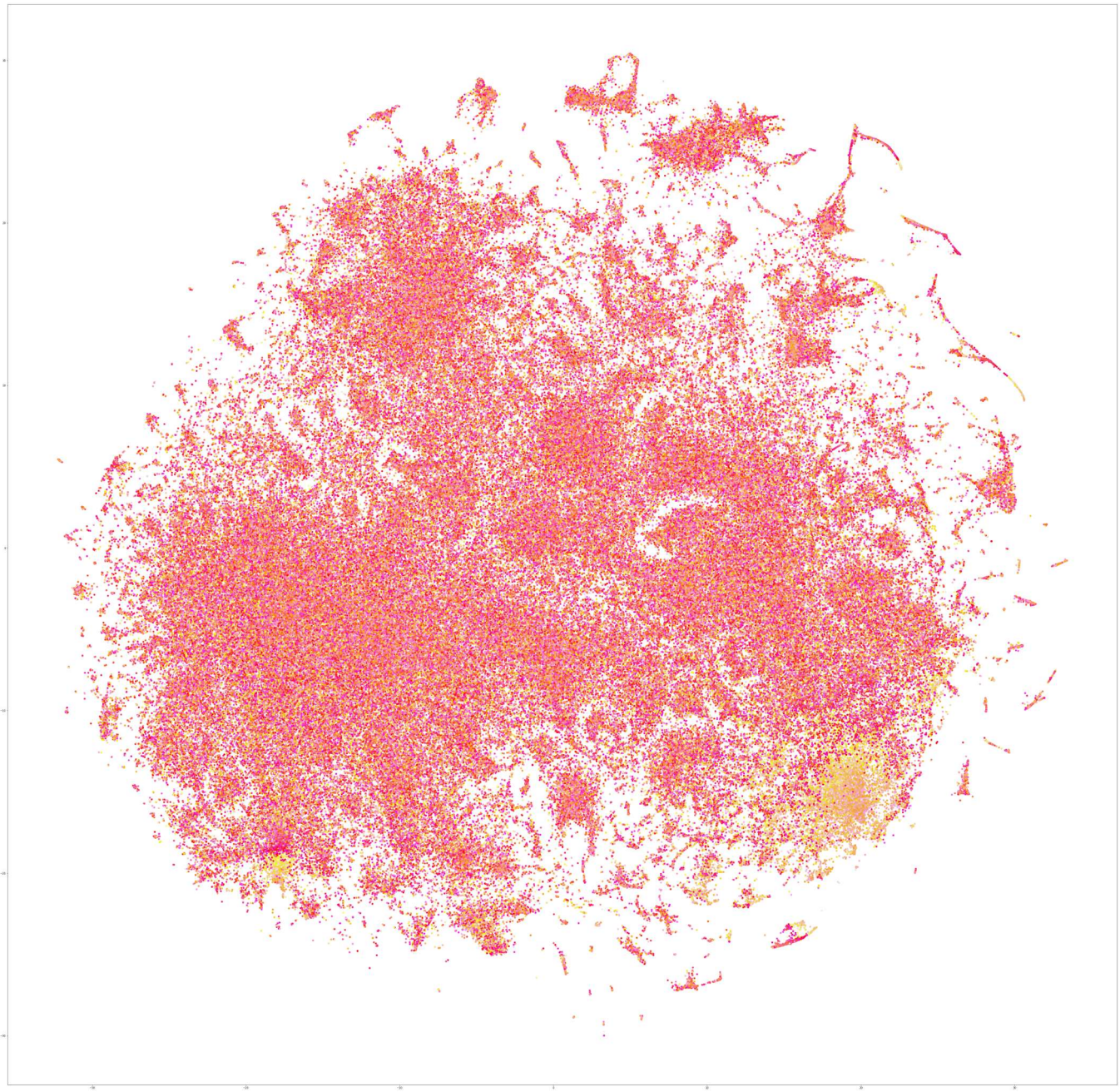
```
from tsnecuda import TSNE
new_values = TSNE(perplexity = 20, learning_rate = 1000).fit_transform(df)
```

Assigning values to respective words

```
import numpy as np
import matplotlib.pyplot as plt


df['x'] = new_values[:,0]
df['y'] = new_values[:,1]
arrx = np.asarray(df['x'])
arry = np.asarray(df['y'])
```

Plotting the words on a graph (may cause google to disconnect from runtime but be patient output will be generated)

```
plt.figure(figsize=(75, 75))
plt.scatter(arrx,arry,c=colors[15900000:16300000]/255.0)
plt.show()
```



Plotting the given words and their closest 20 points

```
#creating a dataframe containing only vectors of 63 points
frames = [closest_to_life,closest_to_market,closest_to_stanford]
temp = pd.concat(frames)
temp = np.asarray(temp)
all_closest = []
for x in temp:
  all_closest.append(np.asarray(all_words[x[0]]))
```

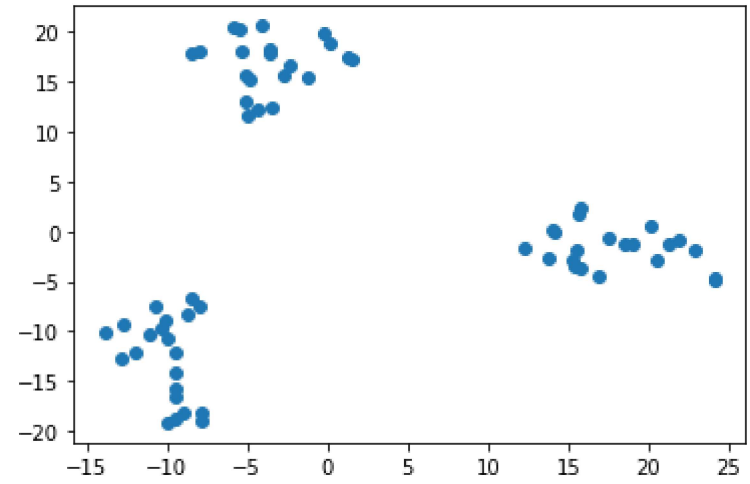
```
all_closest = pd.DataFrame(all_closest)
```

Training TSE model and plotting it on a graph

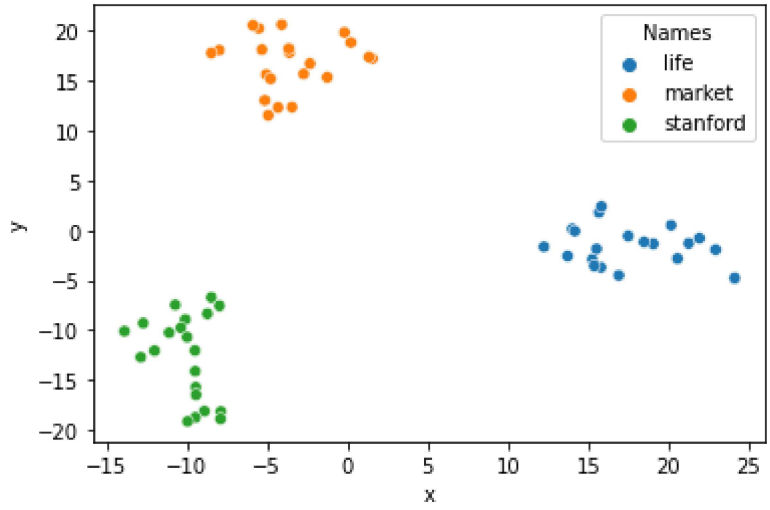
```
closest_new_values = TSNE(perplexity = 5, learning_rate=10).fit_transform(all_closest)
all_closest['x'] = closest_new_values[:,0]
all_closest['y'] = closest_new_values[:,1]
tempax = np.asarray(all_closest['x'])
tempay = np.asarray(all_closest['y'])


#creating temp dataframe to plot different colored clusters
all_temp_closest = all_closest[['x', 'y']]

vals = []
for i in range(63):
  if(i < 21):
    vals.append('life')
  if(i >= 21 and i < 42):
    vals.append('market')
  if(i >= 42 and i < 63):
    vals.append('stanford')

vals = np.asarray(vals)

all_temp_closest.insert(2,"Names",vals,True)

all_temp_closest
```

	x	y	Names

```
plt.scatter(tempax,tempay)
plt.show()

import seaborn as sns
sns.scatterplot(x = "x", y = "y", hue = "Names", data = all_temp_closest)
```



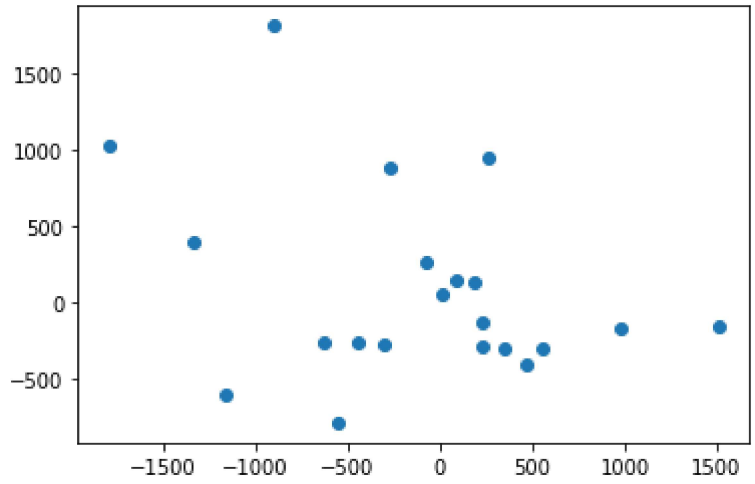
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5976a78790>
```



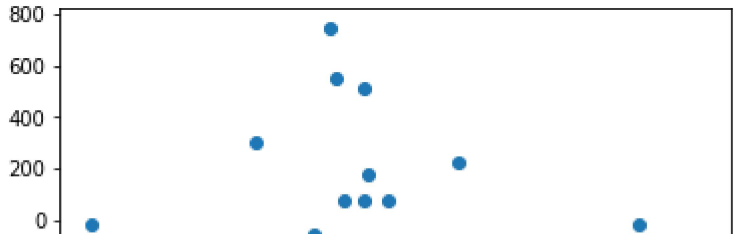
Plotting words and their 20 neighbours seperately

```
#LIFE
forlife = np.asarray(closest_to_life)
vec_life = []
for x in forlife:
  vec_life.append(np.asarray(all_words[x[0]]))

vec_life = pd.DataFrame(vec_life)
```

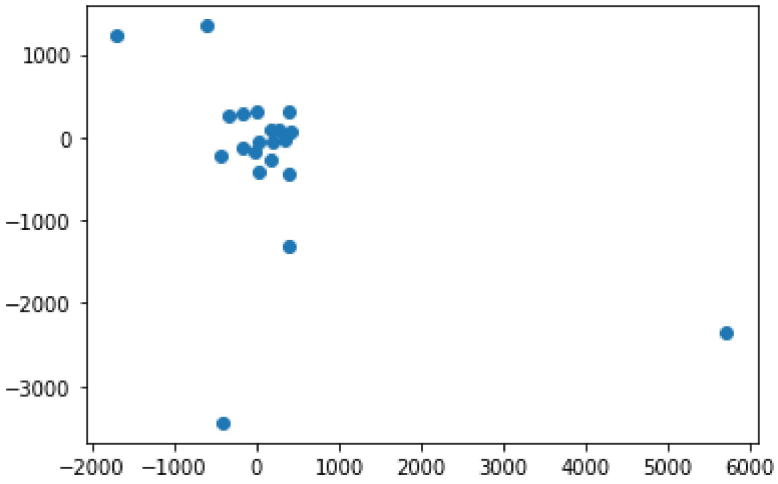
```
closest_new_values_life = TSNE(perplexity = 5).fit_transform(vec_life)

vec_life['x'] = closest_new_values_life[:,0]
vec_life['y'] = closest_new_values_life[:,1]

temp_sx = np.asarray(vec_life['x'])
temp_sy = np.asarray(vec_life['y'])

plt.scatter(temp_sx,temp_sy)
plt.show()
```




```
#MARKET
formarket = np.asarray(closest_to_market)
vec_market = []
for x in formarket:
  vec_market.append(np.asarray(all_words[x[0]]))

vec_market = pd.DataFrame(vec_market)

closest_new_values_market = TSNE(perplexity = 5).fit_transform(vec_market)

vec_market['x'] = closest_new_values_market[:,0]
vec_market['y'] = closest_new_values_market[:,1]

temp_sx = np.asarray(vec_market['x'])
temp_sy = np.asarray(vec_market['y'])

plt.scatter(temp_sx,temp_sy)
plt.show()
```




```
#STANFORD
forstanford = np.asarray(closest_to_stanford)
vec_stanford = []
for x in forstanford:
  vec_stanford.append(np.asarray(all_words[x[0]]))

vec_stanford = pd.DataFrame(vec_stanford)

closest_new_values_stanford = TSNE(perplexity = 5).fit_transform(vec_stanford)

vec_stanford['x'] = closest_new_values_stanford[:,0]
vec_stanford['y'] = closest_new_values_stanford[:,1]

temp_sx = np.asarray(vec_stanford['x'])
temp_sy = np.asarray(vec_stanford['y'])

plt.scatter(temp_sx,temp_sy)
plt.show()
```



✓ 0s completed at 00:01