Assignment No.4

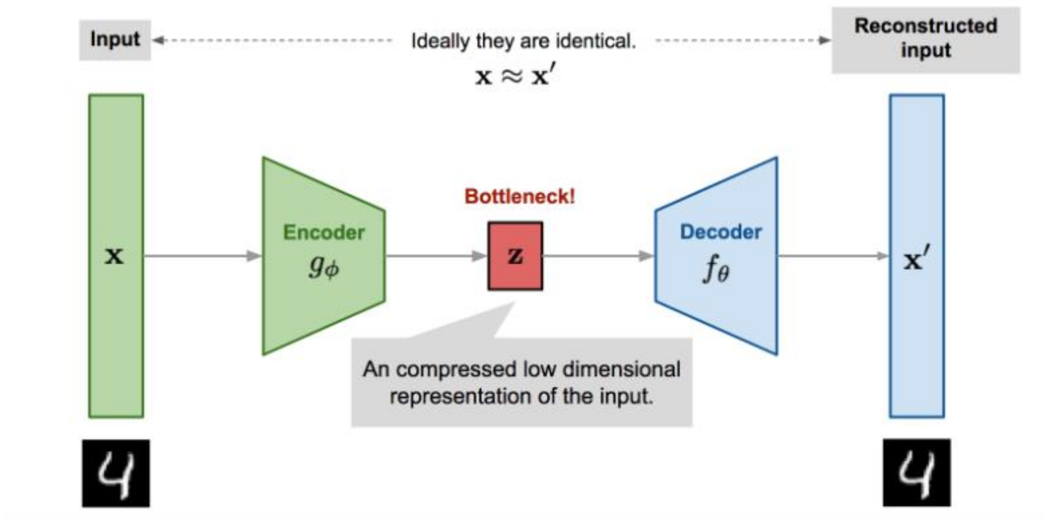
Aim: Use Auto encoder to implement anomaly detection

Objectives:

1. To study how auto encoders works
2. To perform following task as per sequence
a. Import required libraries
b. Upload / access the dataset
c. Encoder converts it into latent representation
d. Decoder networks convert it back to the original input
e. Compile the models with Optimizer, Loss, and Evaluation Metrics

Theory:

AutoEncoder is a generative unsupervised deep learning algorithm used for reconstructing high-dimensional input data using a neural network with a narrow bottleneck layer in the middle which contains the latent representation of the input data.



Autoencoder consists of an Encoder and a Decoder

- **Encoder network**: Accepts high-dimensional input data and translates it to latent low-dimensional data. The input size to an Encoder network is larger than its output size.
- **Decoder network**: The Decoder network receives the input from the Encoder coder's output. Decoder's objective is to reconstruct the input data. The output size of a Decoder network is larger than its input size.
- The Autoencoder accepts high-dimensional input data, compress it down to the latent-space representation in the bottleneck hidden layer; the Decoder takes the latent representation of the data as an input to reconstruct the original input data.

Autoencoders Usage

- **Dimensionality Reduction.** The Encoder encodes the input into the hidden layer to reduce the dimensionality of linear and nonlinear data; hence it is more powerful than PCA.
- **Recommendation Engines**
- **Anomaly Detection**: Autoencoders tries to minimize the reconstruction error as part of its training. Anomalies are detected by checking the magnitude of the reconstruction loss.

- **Denoising Images**: An image that is corrupted can be restored to its original version.
- **Image recognition**: Stacked autoencoder are used for image recognition by learning the different features of an image.
- **Image generation**: Variational Autoencoder(VAE), a type of autoencoders, is used to generate images.

Algorithm/Pseudo codes

Anomaly detection using Auto encoders

We will train an Autoencoder Neural Network (implemented in Keras) in unsupervised (or semi-supervised) fashion for Anomaly Detection in credit card transaction data. The trained model will be evaluated on pre-labeled and anonymized dataset.

Follow the following steps to detect anomalies in a high dimension dataset. You can apply this to unbalanced datasets too.

- During the training, input only normal transactions to the Encoder. The bottleneck layer will learn the latent representation of the normal input data.
- The Decoder will use the bottleneck layers output to reconstruct the normal transactions of the original input data.
- A fraudulent transaction will be different from a normal transaction. The Autoencoder will have trouble reconstructing the fraudulent transaction, and hence the reconstruction error will be high.
- You can flag a new transaction is fraudulent based on a specified threshold value for the reconstruction error.

Dataset used here is Credit Card Fraud Detection

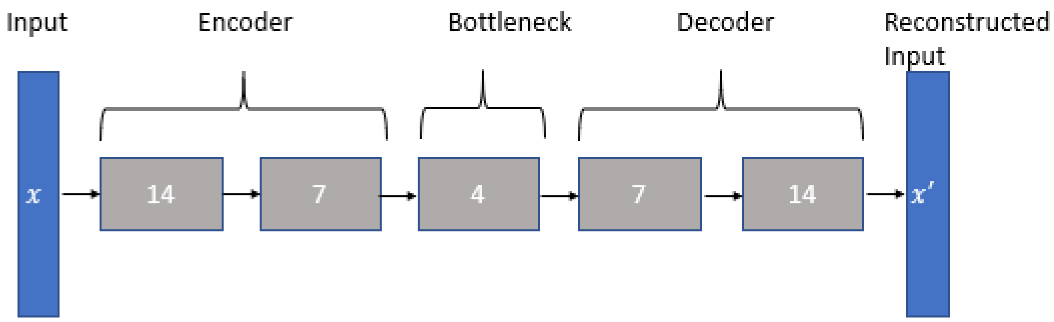
- https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download

About dataset:

It contains data about credit card transactions that occurred during a period of two days, with 492 frauds out of 284,807 transactions. All variables in the dataset are numerical. The data has been transformed using PCA transformation(s) due to privacy reasons. The two features that haven't been changed are Time and Amount. Time contains the seconds elapsed between each transaction and the first transaction in the dataset.

1. Create the Autoencoder

- The architecture of the autoencoder is shown below.



/*Sample code****/**

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split


pip install tensorflow --user
!pip install keras
!pip install daytime
!pip install torch


from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score,
precision_score
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]


dataset = pd.read_csv("E:\Teachning material\Deep learning BE IT 2019 course
\creditcard.csv")
#dataset.head
print(list(dataset.columns))
dataset.describe()


#check for any  nullvalues
print("Any nulls in the dataset ",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ",dataset.Class.unique())
#0 is for normal credit card transaction
#1 is for fraudulent credit card transaction
print('-------')
print("Break down of the Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'], sort = True) )


#Visualizing the imbalanced dataset
count_classes = pd.value_counts(dataset['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");


# Save the normal and fradulent transactions in separate dataframe
```

```
normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]
#Visualize transactionamounts for normal and fraudulent transactions
bins = np.linspace(200, 2500, 100)
plt.hist(normal_dataset.Amount, bins=bins, alpha=1, density=True, label='Nor
mal')
plt.hist(fraud_dataset.Amount, bins=bins, alpha=0.5, density=True, label='Fr
aud')
plt.legend(loc='upper right')
plt.title("Transaction amount vs Percentage of transactions")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions");
plt.show()


#'''Time and Amount are the columns that are not scaled, so applying Standar
dScaler to only Amount and Time columns.
Normalizing the values between 0 and 1 did not work great for the dataset.''
'
sc=StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1, 1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape(-
1, 1))


#'''The last column in the dataset is our target variable.'''

raw_data = dataset.values
# The last element contains if the transaction is normal which is represente
d by a 0 and if fraud then 1
labels = raw_data[:, -1]
# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=2021


#'''Normalize the data to have a value between 0 and 1'''
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)



#Use only normal transactions to train the Autoencoder.

Normal data has a value of 0 in the target variable. Using the target variab
le to create a normal and fraud dataset.'''
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
```

```
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train Data=",len(fraud_train_data))
print(" No. of records in Normal Train data=",len(normal_train_data))
print(" No. of records in Fraud Test Data=",len(fraud_test_data))
print(" No. of records in Normal Test data=",len(normal_test_data))


nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1] #num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7


#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))

#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",

                        activity_regularizer=tf.keras.regularizers.l2(learni
ng_rate))(input_layer)
encoder=tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_2, activation=tf.nn.leaky_relu)(e
ncoder)

# Decoder
decoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
decoder=tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder)

#Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()


#""Define the callbacks for checkpoints and early stopping"""
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               mode='min', monitor='val_loss', verbose=2, sa
ve_best_only=True)
# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
```

```
    mode='min',
    restore_best_weights=True)

#Compile the Autoencoder


autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')
#Train the Autoencoder
history = autoencoder.fit(normal_train_data, normal_train_data,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_data, test_data),
                    verbose=1,
                    callbacks=[cp, early_stop]
                    ).history

#Plot training and test loss

plt.plot(history['loss'], linewidth=2, label='Train')
plt.plot(history['val_loss'], linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()


#"""Detect Anomalies on test data

Anomalies are data points where the reconstruction loss is higher

To calculate the reconstruction loss on test data,
predict the test data and calculate the mean square error between the test d
ata and the reconstructed test data."""

test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,
                        'True_class': test_labels})


#Plotting the test data points and their respective reconstruction error set
s a threshold value to visualize
#if the threshold value needs to be adjusted.

threshold_fixed = 50
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()
for name, group in groups:
```

```
    ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, lin
estyle='',
            label= "Fraud" if name == 1 else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", z
order=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();


'''Detect anomalies as points where the reconstruction loss is greater than
a fixed threshold.
Here we see that a value of 52 for the threshold will be good.

Evaluating the performance of the anomaly detection'''

threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_e
rror.values]
error_df['pred'] =pred_y
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True,
 fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
# print Accuracy, precision and recall
print(" Accuracy: ",accuracy_score(error_df['True_class'], error_df['pred'])
)
print(" Recall: ",recall_score(error_df['True_class'], error_df['pred']))
print(" Precision: ",precision_score(error_df['True_class'], error_df['pred'
]))
#'''As our dataset is highly imbalanced, we see a high accuracy but a low re
call and precision.

Things to further improve precision and recall would add more relevant featu
res,
different architecture for autoencoder, different hyperparameters, or a diff
erent algorithm.'''
```

Conclusion:

- Autoencoder can be used as an anomaly detection algorithm when we have an unbalanced dataset where we have a lot of good examples and only a few anomalies.
- Autoencoders are trained to minimize reconstruction error. When we train the autoencoders on normal data or good data, we can hypothesize that the anomalies will have higher reconstruction errors than the good or normal data.

Output:

Assignment No.5

Aim: Implement the Continuous Bag of Words (CBOW) Model

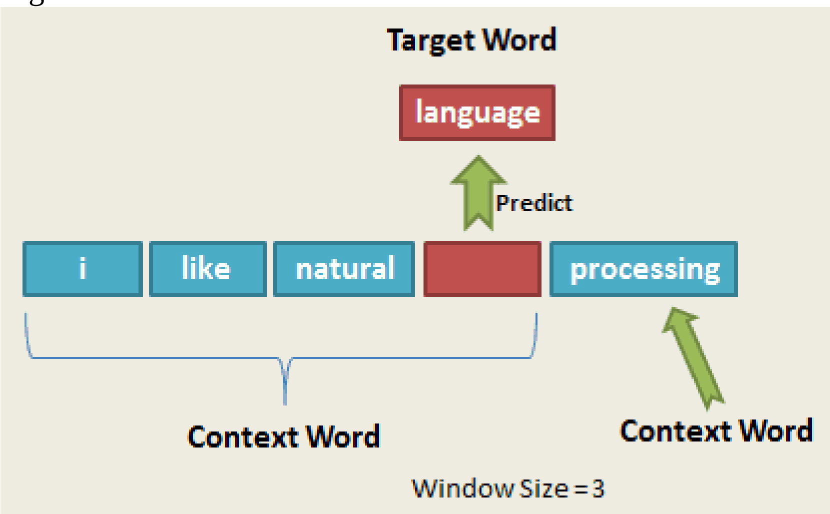
Objectives:

1. Study of Continuous Bag of Words (CBOW) Model
2. Stages can be:
   - **a.** Data preparation
   - **b.** Generate training data
   - **c.** Train model
   - **d.** Output

Theory:

What is the CBOW Model?

- The CBOW model tries to understand the context of the words and takes this as input. It then tries to predict words that are contextually accurate. Let us consider an example for understanding this.



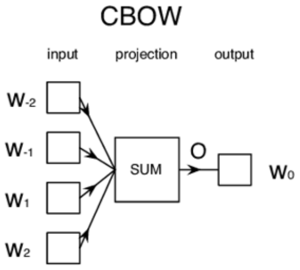
Consider the sentence:

'It is a pleasant day'

the word 'pleasant' goes as input to the neural network. We are trying to predict the word 'day' here.

We will use the one-hot encoding for the input words and measure the error rates with the one-hot encoded target word. Doing this will help us predict the output based on the word with least error.

The Model Architecture



Continuous Bag of Words (CBOW) single-word model:

- **Data Preparation:** Defining corpus by tokenizing text.
- **Generate Training Data:** Build vocabulary of words, one-hot encoding for words, word index.
- **Train Model:** Pass one hot encoded words through **forward pass**, calculate error rate by computing loss, and adjust weights using **back propagation**.
- **Output:** By using trained model calculate word vector and find similar words.

/*Sample code***/**

```
# Import the libraries
from numpy import array
from string import punctuation
from os import listdir
from collections import Counter
from nltk.corpus import stopwords
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from pandas import DataFrame
from matplotlib import pyplot
import nltk
nltk.download('stopwords')


# load doc into memory
def load_doc(filename):
 # open the file as read only
 file = open(filename, 'r')
 # read all text
 text = file.read()
 # close the file
 file.close()
 return text
```

```python
# turn a doc into clean tokens
def clean_doc(doc):
 # split into tokens by white space
 tokens = doc.split()
 # remove punctuation from each token
 table = str.maketrans('', '', punctuation)
 tokens = [w.translate(table) for w in tokens]
 # remove remaining tokens that are not alphabetic
 tokens = [word for word in tokens if word.isalpha()]
 # filter out stop words
 stop_words = set(stopwords.words('english'))
 tokens = [w for w in tokens if not w in stop_words]
 # filter out short tokens
 tokens = [word for word in tokens if len(word) > 1]
 return tokens


# load doc and add to vocab
def add_doc_to_vocab(filename, vocab):
  # load doc
  doc = load_doc(filename)
  # clean doc
  tokens = clean_doc(doc)
  # update counts
  vocab.update(tokens)


# load doc, clean and return line of tokens
def doc_to_line(filename, vocab):
 # load the doc
 doc = load_doc(filename)
 # clean doc
 tokens = clean_doc(doc)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 return ' '.join(tokens)

# load all docs in a directory
def process_docs(directory, vocab, is_trian):
 lines = list()
 # walk through all files in the folder
 for filename in listdir(directory):
  # skip any reviews in the test set
  if is_trian and filename.startswith('cv9'):
   continue
  if not is_trian and not filename.startswith('cv9'):
   continue
  # create the full path of the file to open
  path = directory + '/' + filename
  # load and clean the doc
```

```
    line = doc_to_line(path, vocab)
    # add to list
    lines.append(line)
  return lines

# load all docs in a directory
def process_docs1(directory, vocab):
    # walk through all files in the folder
    for filename in listdir(directory):
      # skip any reviews in the test set
      if filename.startswith('cv9'):
        continue
      # create the full path of the file to open
      path = directory + '/' + filename
      # add doc to vocab
      add_doc_to_vocab(path, vocab)

 # define vocab
vocab = Counter()
# add all docs to vocab
process_docs1('txt_sentoken/pos', vocab)
process_docs1('txt_sentoken/neg', vocab)
# print the size of the vocab
print(len(vocab))
# print the top words in the vocab
print(vocab.most_common(50))

# evaluate a neural network model
def evaluate_mode(Xtrain, ytrain, Xtest, ytest):
 scores = list()
 n_repeats = 30
 n_words = Xtest.shape[1]
 for i in range(n_repeats):
  # define network
  model = Sequential()
  model.add(Dense(50, input_shape=(n_words,), activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
  # compile network
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accu
racy'])
  # fit network
  model.fit(Xtrain, ytrain, epochs=50, verbose=2)
  # evaluate
  loss, acc = model.evaluate(Xtest, ytest, verbose=0)
  scores.append(acc)
  print('%d accuracy: %s' % ((i+1), acc))
 return scores

# prepare bag of words encoding of docs
def prepare_data(train_docs, test_docs, mode):
 # create the tokenizer
 tokenizer = Tokenizer()
```

```
 # fit the tokenizer on the documents
 tokenizer.fit_on_texts(train_docs)
 # encode training data set
 Xtrain = tokenizer.texts_to_matrix(train_docs, mode=mode)
 # encode training data set
 Xtest = tokenizer.texts_to_matrix(test_docs, mode=mode)
 return Xtrain, Xtest


# keep tokens with a min occurrence

min_occurane = 2
tokens = [k for k,c in vocab.items() if c >= min_occurane]
print(len(tokens))

# save list to file
def save_list(lines, filename):
  # convert lines to a single blob of text
  data = '\n'.join(lines)
  # open file
  file = open(filename, 'w')
  # write text
  file.write(data)
  # close file
  file.close()

# save tokens to a vocabulary file
save_list(tokens, 'vocab.txt')

# load the vocabulary
vocab_filename = 'vocab.txt'
vocab = load_doc(vocab_filename)
vocab = vocab.split()
vocab = set(vocab)
# load all training reviews
positive_lines = process_docs('txt_sentoken/pos', vocab, True)
negative_lines = process_docs('txt_sentoken/neg', vocab, True)
train_docs = negative_lines + positive_lines
# load all test reviews
positive_lines = process_docs('txt_sentoken/pos', vocab, False)
negative_lines = process_docs('txt_sentoken/neg', vocab, False)
test_docs = negative_lines + positive_lines
# prepare labels
ytrain = array([0 for _ in range(900)] + [1 for _ in range(900)])
ytest = array([0 for _ in range(100)] + [1 for _ in range(100)])

modes = ['binary', 'count', 'tfidf', 'freq']
results = DataFrame()
for mode in modes:
 # prepare data for mode
 Xtrain, Xtest = prepare_data(train_docs, test_docs, mode)
 # evaluate model on data for mode
```

```
    results[mode] = evaluate_mode(Xtrain, ytrain, Xtest, ytest)
# summarize results
print(results.describe())
# plot results
results.boxplot()
pyplot.show()

# classify a review as negative (0) or positive (1)
def predict_sentiment(review, vocab, tokenizer, model):
 # clean
 tokens = clean_doc(review)
 # filter by vocab
 tokens = [w for w in tokens if w in vocab]
 # convert to line
 line = ' '.join(tokens)
 # encode
 encoded = tokenizer.texts_to_matrix([line], mode='freq')
 # prediction
 yhat = model.predict(encoded, verbose=0)
 return round(yhat[0,0])

# test positive text
text = 'Best movie ever!'
print(predict_sentiment(text, vocab, tokenizer, model))
# test negative text
text = 'This is a bad movie.'
print(predict_sentiment(text, vocab, tokenizer, model))
```

Conclusion:

Thus we have studied working of CBOW model.

Ouput: