M. Sc (Information Technology)			Semester – IV	
Course Name: Deep Learning Practical			**Course Code: PSIT4P3a**	
Credits			2	
Hours			**Marks**	
Evaluation System	**Practical Examination**		2	50
Internal	--		--	

List of Practical:	
1.	Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow
2.	Solving XOR problem using deep feed forward network.
3.	Implementing deep neural network for performing binary classification task.
4.	a) Using deep feed forward network with two hidden layers for performing multiclass classification and predicting the class. b) Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class. c) Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.
5.	a) Evaluating feed forward deep network for regression using KFold cross validation. b) Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.
6.	Implementing regularization to avoid overfitting in binary classification.
7.	Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.
8.	Performing encoding and decoding of images using deep autoencoder.
9.	Implementation of convolutional neural network to predict numbers from number images
10.	Denoising of images using autoencoder.

Install python packages :
open new cmd and execute pip install <package name> command.

```
C:\Users\pradnya>pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.24.0-cp39-cp39-win_amd64.whl (6.9 MB)
     |████████████████████████████████| 6.9 MB 1.3 MB/s
Collecting scipy>=0.19.1
  Downloading scipy-1.6.0-cp39-cp39-win_amd64.whl (32.7 MB)
     |████████████████████████████████| 32.7 MB 1.6 MB/s
Collecting joblib>=0.11
  Downloading joblib-1.0.0-py3-none-any.whl (302 kB)
     |████████████████████████████████| 302 kB 2.2 MB/s
Collecting numpy>=1.13.3
  Downloading numpy-1.19.5-cp39-cp39-win_amd64.whl (13.3 MB)
     |████████████████████████████████| 13.3 MB 134 kB/s
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Installing collected packages: numpy, scipy, joblib, threadpoolctl, scikit-learn
Successfully installed joblib-1.0.0 numpy-1.19.5 scikit-learn-0.24.0 scipy-1.6.0 threadpoolctl-2.1.0
```

```
C:\Users\pradnya>pip install matplotlib
Collecting matplotlib
  Downloading matplotlib-3.3.3-cp39-cp39-win_amd64.whl (8.5 MB)
     |████████████████████████████████| 8.5 MB 24 kB/s
Collecting python-dateutil>=2.1
  Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
     |████████████████████████████████| 227 kB 656 kB/s
Collecting kiwisolver>=1.0.1
  Downloading kiwisolver-1.3.1-cp39-cp39-win_amd64.whl (51 kB)
     |████████████████████████████████| 51 kB 181 kB/s
Requirement already satisfied: numpy>=1.15 in c:\users\pradnya\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.19.5)
Collecting pillow>=6.2.0
  Downloading Pillow-8.1.0-cp39-cp39-win_amd64.whl (2.2 MB)
     |████████████████████████████████| 2.2 MB 1.1 MB/s
Collecting cycler>=0.10
  Downloading cycler-0.10.0-py2.py3-none-any.whl (6.5 kB)
Collecting pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
     |████████████████████████████████| 67 kB 675 kB/s
Collecting six>=1.5
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: six, python-dateutil, kiwisolver, pillow, cycler, pyparsing, matplotlib
Successfully installed cycler-0.10.0 kiwisolver-1.3.1 matplotlib-3.3.3 pillow-8.1.0 pyparsing-2.4.7 python-dateutil-2.8.1 six-1.15.0
```

Note:
If you got an error: pip is not recognised as internal or external command
Solution:
Paste path of scripts directory from python39 directory like this:
C:\Users\pradnya\AppData\Local\Programs\Python\Python39\Scripts
In environmental variables→user variables→ path→ edit→ new→paste →ok→ok
Then open new cmd and execute pip install <package name> command.

OR

You can use google colab or jupyter notebook

Why use TensorFlow?

TensorFlow is an open-source machine learning library designed by Google to meet its need for systems capable of building and training neural networks and has an Apache 2.0 license.
Created by the Google Brain team, TensorFlow presents calculations in the form of stateful dataflow graphs. The library allows you to implement calculations on a wide range of hardware, from consumer devices running Android to large heterogeneous systems with multiple GPUs. TensorFlow allows you to transfer the performance of computationally intensive tasks from a single CPU environment to a heterogeneous fast environment with multiple GPUs without significant code changes TensorFlow is designed to provide massive concurrency and highly scalable machine learning for a wide range of users.
The central object of TensorFlow is a dataflow graph representing calculations. The vertices of the graph represent operations, and the edges represent tensors (multidimensional arrays that are the basis of TensorFlow). The data flow graph is a complete description of the calculations that are implemented within the session and performed on CPU or GPU devices.
Assuming you have python3 and pip3 on your computer, to install Tensorflow you need to type in the command line:

```
pip3 install --user --upgrade tensorflow
```

Yes, it works. Now let's install Jupyter notebook. Jupyer notebook will help to enter code and run it in a comfortable environment.

```
pip3 install jupyter
```

And run Jupyter:

```
jupyter notebook
```

You will have a new window in your browser and will be ready to write code in Python with TensorFlow.

Practical no. 1

Performing matrix multiplication and finding eigen vectors and eigen values using TensorFlow

A] Matrix multiplication using TensorFlow:

```
import tensorflow as tf
print("Matrix Multiplication Demo")
x=tf.constant([1,2,3,4,5,6],shape=[2,3])
print(x)
y=tf.constant([7,8,9,10,11,12],shape=[3,2])
print(y)
z=tf.matmul(x,y)
print("Product:",z)
```

o/p:paste o/p here

B] Finding eigen vectors and eigen values usingTensorFlow:

What are *eigenvectors* and *eigenvalues*?

Suppose that we have a matrix A with the following entries:

$$A=[2 \quad 0 \quad 0 \quad -1].$$

If we apply A to any vector $v=[x,y]^T$, we obtain a vector $Av=[2x, \; -y]^T$.
This has an intuitive interpretation: stretch the vector to be twice as wide in the x-direction, and then flip it in the y-direction.
However, there are *some* vectors for which something remains unchanged. Namely $[1,0]^T$ gets sent to $[2,0]^T$ and $[0,1]^T$ gets sent to $[0,-1]^T$. These vectors are still in the same line, and the only modification is that the matrixstretches them by a factor of 2 and −1 respectively. We call such vectors *eigenvectors* and the factor they are stretched by *eigenvalues*.
In general if we can find a number λ and a vector v such that

$$Av=\lambda v.$$

We say that v is an eigenvector for A and λ is an eigenvalue.

Code:

```
import tensorflow as tf
e_matrix_A = tf.random.uniform([2, 2], minval=3, maxval=10,
dtype=tf.float32, name="matrixA")
print("Matrix A: \n{}\n\n".format(e_matrix_A))
eigen_values_A, eigen_vectors_A = tf.linalg.eigh(e_matrix_A)
print("Eigen Vectors: \n{} \n\nEigen Values: \n{}\n".format(eigen_vectors_A,
eigen_values_A))
```

o/p:

```
Matrix A:
[[5.450138 9.455662]
 [9.980919 9.223391]]
```

```
Eigen Vectors:
[[-0.76997876 -0.6380696 ]
 [ 0.6380696  -0.76997876]]

Eigen Values:
[-2.8208985 17.494429 ]
```

Reference links:

- https://www.codementor.io/@alexander-k/tensorflow-basics-matrix-operations-13kz9riblj
- https://biswajitsahoo1111.github.io/post/doing-linear-algebra-using-tensorflow-2/
- https://www.dummies.com/web-design-development/other-web-software/create-vector-matrix-operations-tensorflow/
- https://morioh.com/p/d669c3deea75
- https://github.com/akhilvasvani/Linear-Algebra-Basics/tree/master/Chapters
- https://medium.com/@mukesh.mithrakumar/https-medium-com-mukesh-mithrakumar-eigendecomposition-with-tensorflow-d80b94171106

Practical No:2

Aim: Solving XOR problem using deep feed forward network.

Theory: reference book (Deep learning_ adaptive computation and machine learning) page no 167

Code:

```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential
model=Sequential()
model.add(Dense(units=2,activation='relu',input_dim=2))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
print(model.summary())
print(model.get_weights())
X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
Y=np.array([0.,1.,1.,0.])
model.fit(X,Y,epochs=1000,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
```

o/p: paste here

Practical No:3

Aim: Implementing deep neural network for performing classification task.

Problem statement: the given dataset comprises of health information about diabetic women patient. we need to create deep feed forward network that will classify women suffering from diabetes mellitus as 1.

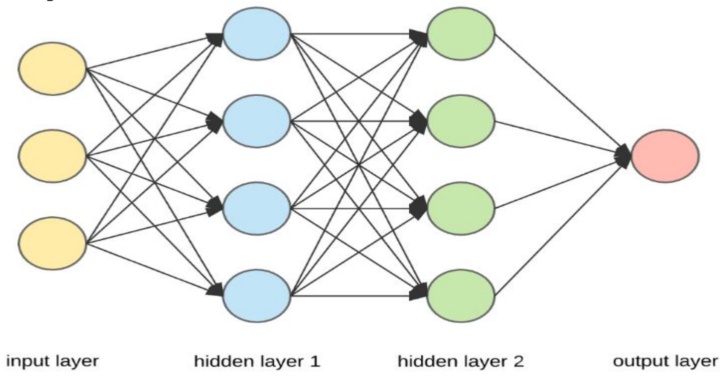
Code:

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
dataset=loadtxt(‘pima-indians-diabetes.csv’,delimiter=’,’)
dataset
X=dataset[:,0:8]
Y=dataset[:,8]
model=sequential()
model.add(Dense(12,input_dim=8,activation=’relu’))
model.add(Dense(8,activation=’relu’))
model.add(Dense(1,activation=’sigmoid’))
model.compile(loss=’binary_crossentropy’,optimizer=’adam’,metrics=[‘accuracy’])
model.fit(X,Y,epochs=150,batch_size=10)
_,accuracy=model.evaluate(X,Y)
print(‘accuracy of model is ’,(accuracy*100))
prediction=model.predict(X)
for I in range(5):
print(X[i].tolist(),prediction[i],Y[i])”)
```

o/p: paste here

Practical No:4

Theory:

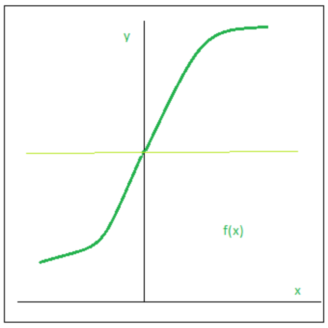


Components:

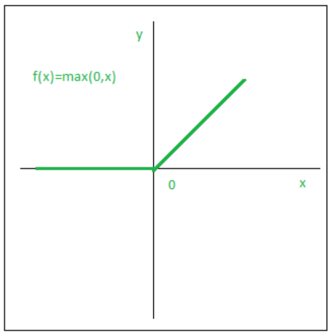
1. **Input Layers,**
2. **Neurons,**
3. **Weights,**
4. **Hidden Layers and**
5. **Output Layer**

The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output.
Several different types of activation functions are used in Deep Learning.
Some of them are explained below:

1. Sigmoid $\frac{1}{1+e^{x}}$



2. Rectified Linear Unit
   f(x) = max {0, x}



a) **Aim: Using deep feed forward network with two hidden layers for performing classification and predicting the class.**

Code:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=500)
Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
Xnew=scalar.transform(Xnew)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))
```

o/p: paste here

b) **Aim: Using a deep field forward network with two hidden layers for performing classification and predicting the probability of class.**

Code:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=500)
Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
```

```
Xnew=scalar.transform(Xnew)
Yclass=model.predict_classes(Xnew)
Ynew=model.predict_proba(Xnew)
for i in range(len(Xnew)):
print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Ycla
ss[i]))
```

o/p: paste here

c) **Aim: Using a deep field forward network with two hidden layers for performing linear regression and predicting values.**

Code:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
X,Y=make_regression(n_samples=100,n_features=2,noise=0.1,random_state=1)
scalarX,scalarY=MinMaxScaler(),MinMaxScaler()
scalarX.fit(X)
scalarY.fit(Y.reshape(100,1))
X=scalarX.transform(X)
Y=scalarY.transform(Y.reshape(100,1))
model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='mse',optimizer='adam')
model.fit(X,Y,epochs=1000,verbose=0)
Xnew,a=make_regression(n_samples=3,n_features=2,noise=0.1,random_state=1)
Xnew=scalarX.transform(Xnew)
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
print("X=%s,Predicted=%s"%(Xnew[i],Ynew[i]))
```

o/p: paste here

reference links:

https://youtu.be/qJhH2cnjD34
https://github.com/navkar/TensorFlow/blob/master/linear_regression.md

Practical No:5

a) Aim: Evaluating feed forward deep network for regression using KFold cross validation.

Code:

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
dataframe=pd.read_csv("housing.csv",delim_whitespace=True,header=None)
dataset=dataframe.values
X=dataset[:,0:13]
Y=dataset[:,13]
def wider_model():
model=Sequential()
model.add(Dense(15,input_dim=13,kernel_initializer='normal',activation='relu'))
model.add(Dense(13,kernel_initializer='normal',activation='relu'))
model.add(Dense(1,kernel_initializer='normal'))
model.compile(loss='mean_squared_error',optimizer='adam')
return model
estimators=[]
estimators.append(('standardize',StandardScaler()))
estimators.append(('mlp',KerasRegressor(build_fn=wider_model,epochs=100,batch_si
ze=5)))
pipeline=Pipeline(estimators)
kfold=KFold(n_splits=10)
results=cross_val_score(pipeline,X,Y,cv=kfold)
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

o/p: paste here

(After changing neuron)

```
model.add(Dense(20, input_dim=13,kernel_initializer='normal',activation='relu'))
```

o/p: paste here

compare both o/ps

b) **Aim: Evaluating feed forward deep network for multiclass Classification using KFold cross-validation.**

Code:

```
#loading libraries
import pandas
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
#loading dataset
df=pandas.read_csv('Flower.csv',header=None)
print(df)
#splitting dataset into input and output variables
X = df.iloc[:,0:4].astype(float)
y=df.iloc[:,4]
#print(X)
#print(y)
#encoding string output into numeric output
encoder=LabelEncoder()
encoder.fit(y)
encoded_y=encoder.transform(y)
print(encoded_y)
dummy_Y=np_utils.to_categorical(encoded_y)
print(dummy_Y)
def baseline_model():
# create model
model = Sequential()
model.add(Dense(8, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
return model
estimator=baseline_model()
estimator.fit(X,dummy_Y,epochs=100,shuffle=True)
action=estimator.predict(X)
for i in range(25):
print(dummy_Y[i])
print('^^^^^^^^^^^^^^^^^^^^^^^^')
for i in range(25):
print(action[i])
```

o/p: paste here

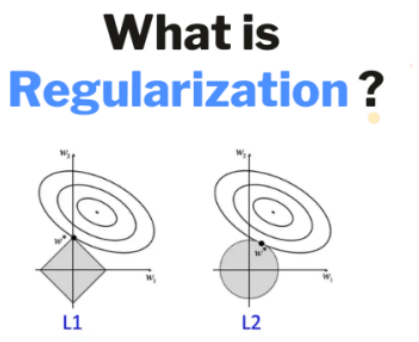
Practical No :6

Aim: implementing regularization to avoid overfitting in binary classification.

Theory:

Overfitting and regularization are the most common terms which are heard in **Machine learning** and Statistics. Your model is said to be overfitting if it performs very well on the training data but fails to perform well on unseen data.

This is one of the most common and dangerous phenomena that occurs when **training your machine learning models**. There are many techniques that you can use to fix this problem. **Regularization** is one among them.



Regularization, as the name suggests, is the process of regularizing something. Regularization shrinks the parameters of the model to zero, which **reduces** its freedom.

Hence, the model will be less likely to fit the noise of training data and will improve the generalization ability of the model.

We penalize the **cost function** by adding a penalty that regularizes or **shrinks** the coefficient estimates to zero.

Let’s look at the cost function

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (\hat{y}_i - y_i)^2$$

where

- **hθ(x(i))** is the predicted value of some datapoint x(i)
- **y(i)** is original

The penalized cost function looks like this

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \overset{\text{Regularization Term}}{\boxed{\underset{\text{Regularization Parameter}}{\boxed{\lambda}}\sum_{j=1 \,\leftarrow\, \text{start at } \theta_1}^{n}\theta_j{}^2}}$$

where-

- λ is the tuning parameter that decides how much we want to penalize the flexibility of our model. It can be tuned using cross-validation.

So each time some parameter tries to become large, it will be penalized to a small value.

There are two kinds of regularization:

- L1 Regularization
- L2 Regularization

L1 Regularization

This adds a penalty equal to the **L1 norm** of the weights vector(sum of the absolute value of the coefficients). It will shrink some parameters to **zero**.

Hence some variables will not play any role in the model. L1 regression can be seen as a way to select features in a model.

L1 = L(X,y) + λ|θ|

L2 Regularization

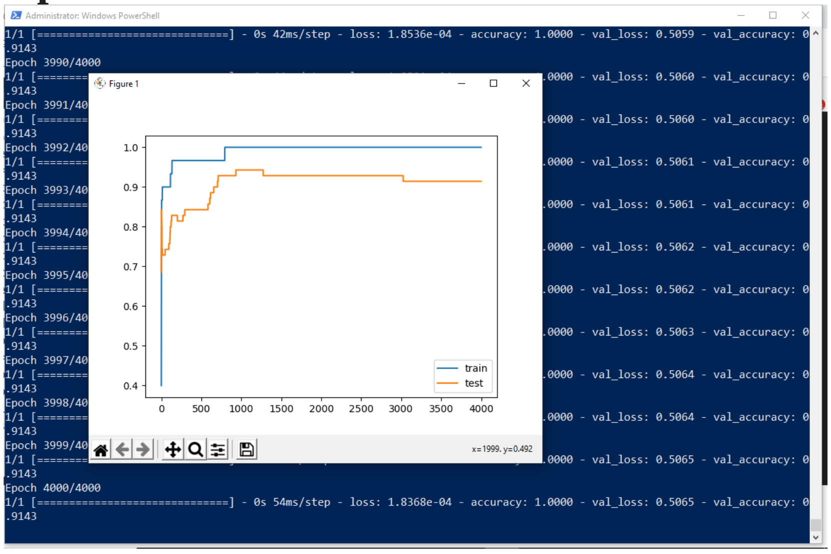
This adds a penalty equal to the L2 norm of the weights vector(sum of the squared values of the coefficients). It will force the parameters to be relatively small.

L2 = L(X,y) + λθ2

Code:
```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
#print(testY)
model=Sequential()
```

```
model.add(Dense(500,input_dim=2,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=4000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

o/p:



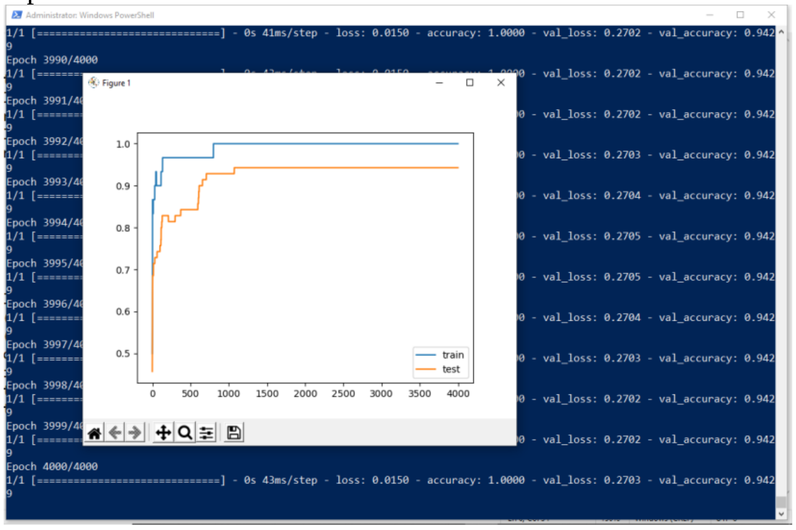
```
1/1 [==============================] - 0s 42ms/step - loss: 1.8536e-04 - accuracy: 1.0000 - val_loss: 0.5059 - val_accuracy: 0
.9143
Epoch 3990/4000
1/1 [========                                              .0000 - val_loss: 0.5060 - val_accuracy: 0
.9143
Epoch 3991/40                                              
1/1 [========                                              .0000 - val_loss: 0.5060 - val_accuracy: 0
.9143
Epoch 3992/40
1/1 [========                                              .0000 - val_loss: 0.5061 - val_accuracy: 0
.9143
Epoch 3993/40
1/1 [========                                              .0000 - val_loss: 0.5061 - val_accuracy: 0
.9143
Epoch 3994/40
1/1 [========                                              .0000 - val_loss: 0.5062 - val_accuracy: 0
.9143
Epoch 3995/40
1/1 [========                                              .0000 - val_loss: 0.5062 - val_accuracy: 0
.9143
Epoch 3996/40
1/1 [========                                              .0000 - val_loss: 0.5063 - val_accuracy: 0
.9143
Epoch 3997/40
1/1 [========                                              .0000 - val_loss: 0.5064 - val_accuracy: 0
.9143
Epoch 3998/40
1/1 [========                                              .0000 - val_loss: 0.5064 - val_accuracy: 0
.9143
Epoch 3999/40
1/1 [========                                              .0000 - val_loss: 0.5065 - val_accuracy: 0
.9143
Epoch 4000/4000
1/1 [==============================] - 0s 54ms/step - loss: 1.8368e-04 - accuracy: 1.0000 - val_loss: 0.5065 - val_accuracy: 0
.9143
```

The above code and resultant graph demonstrate overfitting with accuracy of testing data less than accuracy of training data also the accuracy of testing data increases once and then start decreases gradually.to solve this problem we can use regularization
Hence, we will add two lines in the above code as highlighted below to implement l2 regularization with alpha=0.001

Code:

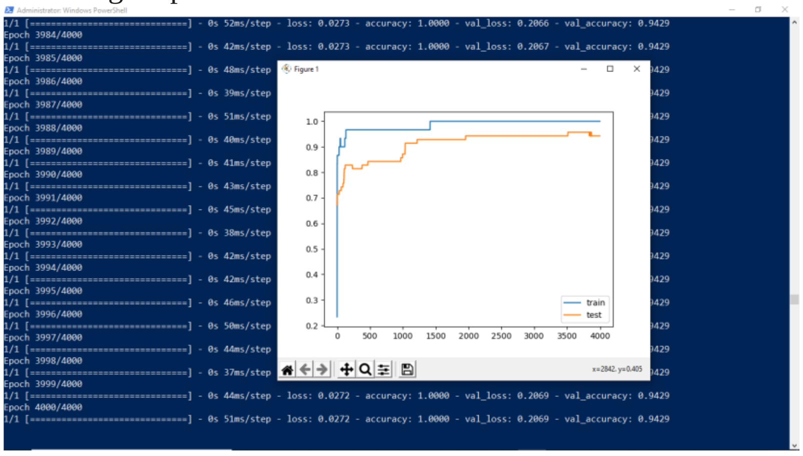
```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l2    #its L2 not twelve
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
#print(testY)
model=Sequential()
model.add(Dense(500,input_dim=2,activation='relu',kernel_regularizer=l2(0.001)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=4000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```

o/p:



```
1/1 [==============================] - 0s 41ms/step - loss: 0.0150 - accuracy: 1.0000 - val_loss: 0.2702 - val_accuracy: 0.942
9
Epoch 3990/4000
1/1 [=======                                                 0 - val_loss: 0.2702 - val_accuracy: 0.942
9
Epoch 3991/4                                                 0 - val_loss: 0.2702 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3992/4                                                 0 - val_loss: 0.2703 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3993/4                                                 0 - val_loss: 0.2704 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3994/4                                                 0 - val_loss: 0.2705 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3995/4                                                 0 - val_loss: 0.2705 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3996/4                                                 0 - val_loss: 0.2704 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3997/4                                                 0 - val_loss: 0.2703 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3998/4                                                 0 - val_loss: 0.2702 - val_accuracy: 0.942
1/1 [=======
9
Epoch 3999/4                                                 0 - val_loss: 0.2702 - val_accuracy: 0.942
1/1 [=======
9
Epoch 4000/4000
1/1 [==============================] - 0s 43ms/step - loss: 0.0150 - accuracy: 1.0000 - val_loss: 0.2703 - val_accuracy: 0.942
9
```

By replacing l2 (L2) regularizer with l1(L1) regularizer at the same learning rate 0.001 we get the following output.



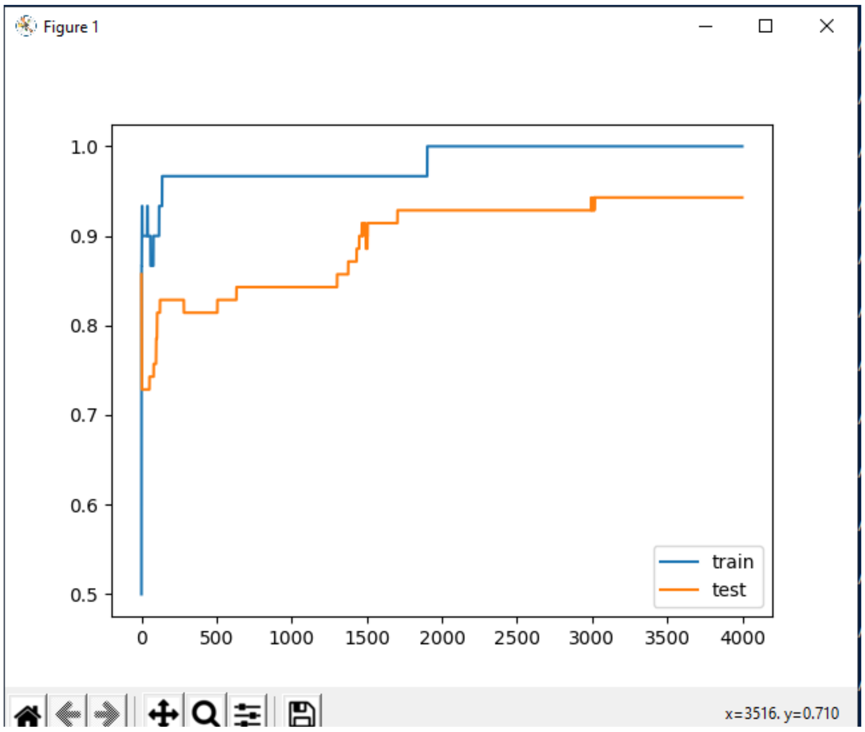
```
1/1 [==============================] - 0s 52ms/step - loss: 0.0273 - accuracy: 1.0000 - val_loss: 0.2066 - val_accuracy: 0.9429
Epoch 3984/4000
1/1 [==============================] - 0s 42ms/step - loss: 0.0273 - accuracy: 1.0000 - val_loss: 0.2067 - val_accuracy: 0.9429
Epoch 3985/4000
1/1 [==============================] - 0s 48ms/step                                                              9429
Epoch 3986/4000
1/1 [==============================] - 0s 39ms/step                                                              9429
Epoch 3987/4000
1/1 [==============================] - 0s 51ms/step                                                              9429
Epoch 3988/4000
1/1 [==============================] - 0s 40ms/step                                                              9429
Epoch 3989/4000
1/1 [==============================] - 0s 41ms/step                                                              9429
Epoch 3990/4000
1/1 [==============================] - 0s 43ms/step                                                              9429
Epoch 3991/4000
1/1 [==============================] - 0s 45ms/step                                                              9429
Epoch 3992/4000
1/1 [==============================] - 0s 38ms/step                                                              9429
Epoch 3993/4000
1/1 [==============================] - 0s 42ms/step                                                              9429
Epoch 3994/4000
1/1 [==============================] - 0s 42ms/step                                                              9429
Epoch 3995/4000
1/1 [==============================] - 0s 46ms/step                                                              9429
Epoch 3996/4000
1/1 [==============================] - 0s 50ms/step                                                              9429
Epoch 3997/4000
1/1 [==============================] - 0s 44ms/step                                                              9429
Epoch 3998/4000
1/1 [==============================] - 0s 37ms/step                                                              9429
Epoch 3999/4000
1/1 [==============================] - 0s 44ms/step - loss: 0.0272 - accuracy: 1.0000 - val_loss: 0.2069 - val_accuracy: 0.9429
Epoch 4000/4000
1/1 [==============================] - 0s 51ms/step - loss: 0.0272 - accuracy: 1.0000 - val_loss: 0.2069 - val_accuracy: 0.9429
```

By applying l1 and l2 regularizer we can observe the following changes in accuracy of both trainig and testing data. The changes in code are also highlighted.

Code:

```
from matplotlib import pyplot
from sklearn.datasets import make_moons
from keras.models import Sequential
from keras.layers import Dense
from keras.regularizers import l1_l2
X,Y=make_moons(n_samples=100,noise=0.2,random_state=1)
n_train=30
trainX,testX=X[:n_train,:],X[n_train:]
```

```
trainY,testY=Y[:n_train],Y[n_train:]
#print(trainX)
#print(trainY)
#print(testX)
#print(testY)
model=Sequential()
model.add(Dense(500,input_dim=2,activation='relu',kernel_regularizer=l1_l2(l1=0.001,l2=0.001)))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
history=model.fit(trainX,trainY,validation_data=(testX,testY),epochs=4000)
pyplot.plot(history.history['accuracy'],label='train')
pyplot.plot(history.history['val_accuracy'],label='test')
pyplot.legend()
pyplot.show()
```



Practical No:7

Aim: Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.

Theory: reference book (Deep learning_ adaptive computation and machine learning) ch 10 page no. 373

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.preprocessing import MinMaxScaler
dataset_train=pd.read_csv('Google_Stock_price_train.csv')
#print(dataset_train)
training_set=dataset_train.iloc[:,1:2].values
#print(training_set)
sc=MinMaxScaler(feature_range=(0,1))
training_set_scaled=sc.fit_transform(training_set)
#print(training_set_scaled)
X_train=[]
Y_train=[]
for i in range(60,1258):
X_train.append(training_set_scaled[i-60:i,0])
Y_train.append(training_set_scaled[i,0])
X_train,Y_train=np.array(X_train),np.array(Y_train)
print(X_train)
print('**********************************************')
print(Y_train)
X_train=np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
print('***********************************************')
print(X_train)
regressor=Sequential()
regressor.add(LSTM(units=50,return_sequences=True,input_shape=(X_train.shape[1],
1)))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50,return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50,return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(LSTM(units=50))
regressor.add(Dropout(0.2))
regressor.add(Dense(units=1))
regressor.compile(optimizer='adam',loss='mean_squared_error')
regressor.fit(X_train,Y_train,epochs=100,batch_size=32)
dataset_test=pd.read_csv('Google_Stock_price_Test.csv')
```

```
real_stock_price=dataset_test.iloc[:,1:2].values
dataset_total=pd.concat((dataset_train['Open'],dataset_test['Open']),axis=0)
inputs=dataset_total[len(dataset_total)-len(dataset_test)-60:].values
inputs=inputs.reshape(-1,1)
inputs=sc.transform(inputs)
X_test=[]
for i in range(60,80):
X_test.append(inputs[i-60:i,0])
X_test=np.array(X_test)
X_test=np.reshape(X_test,(X_test.shape[0],X_test.shape[1],1))
predicted_stock_price=regressor.predict(X_test)
predicted_stock_price=sc.inverse_transform(predicted_stock_price)
plt.plot(real_stock_price,color='red',label='real google stock price')
plt.plot(predicted_stock_price,color='blue',label='predicted stock price')
plt.xlabel('time')
plt.ylabel('google stock price')
plt.legend()
plt.show()
```

o/p: paste here

Practical No:8

Aim: Performing encoding and decoding of images using deep autoencoder.

Theory: refer reference book

Code:

```
import keras
from keras import layers
from keras.datasets import mnist
import numpy as np
encoding_dim=32
#this is our input image
input_img=keras.Input(shape=(784,))
#"encoded" is the encoded representation of the input
encoded=layers.Dense(encoding_dim, activation='relu')(input_img)
#"decoded" is the lossy reconstruction of the input
decoded=layers.Dense(784, activation='sigmoid')(encoded)
#creating autoencoder model
autoencoder=keras.Model(input_img,decoded)
#create the encoder model
encoder=keras.Model(input_img,encoded)
encoded_input=keras.Input(shape=(encoding_dim,))
#Retrive the last layer of the autoencoder model
decoder_layer=autoencoder.layers[-1]
#create the decoder model
decoder=keras.Model(encoded_input,decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
#scale and make train and test dataset
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
X_test=X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
print(X_train.shape)
print(X_test.shape)
#train autoencoder with training dataset
autoencoder.fit(X_train,X_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(X_test,X_test))
encoded_imgs=encoder.predict(X_test)
decoded_imgs=decoder.predict(encoded_imgs)
import matplotlib.pyplot as plt
n = 10 # How many digits we will display
plt.figure(figsize=(40, 4))
for i in range(10):
# display original
ax = plt.subplot(3, 20, i + 1)
```

```
plt.imshow(X_test[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display encoded image
ax = plt.subplot(3, 20, i + 1 + 20)
plt.imshow(encoded_imgs[i].reshape(8,4))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display reconstruction
ax = plt.subplot(3, 20, 2*20 +i+ 1)
plt.imshow(decoded_imgs[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```

o/p:paste here

Practical no. 9

Implementation of convolutional neural network topredict numbers from number images:

Theory: Refer uploaded file ConvolutionalNeuralNetworks.pdf for theory.

Code:

```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense,Conv2D,Flatten
import matplotlib.pyplot as plt
#download mnist data and split into train and test sets
(X_train,Y_train),(X_test,Y_test)=mnist.load_data()
#plot the first image in the dataset
plt.imshow(X_train[0])
plt.show()
print(X_train[0].shape)
X_train=X_train.reshape(60000,28,28,1)
X_test=X_test.reshape(10000,28,28,1)
Y_train=to_categorical(Y_train)
Y_test=to_categorical(Y_test)
Y_train[0]
print(Y_train[0])
model=Sequential()
# add model layers
#learn image features
model.add(Conv2D(64,kernel_size=3,activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(32,kernel_size=3,activation='relu'))
model.add(Flatten())
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
#train
model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=3)
print(model.predict(X_test[:4]))
#actual results for 1st 4 images in the test set
  print(Y_test[:4])
```

o/p: paste here

REFERENCE LINKS:

https://youtu.be/vWo7sf_afEs

Practical no. 10:
Denoising of images using autoencoder

Building and training an image denoising autoencoder using Keras with Tensorflow 2.0 as a backend.

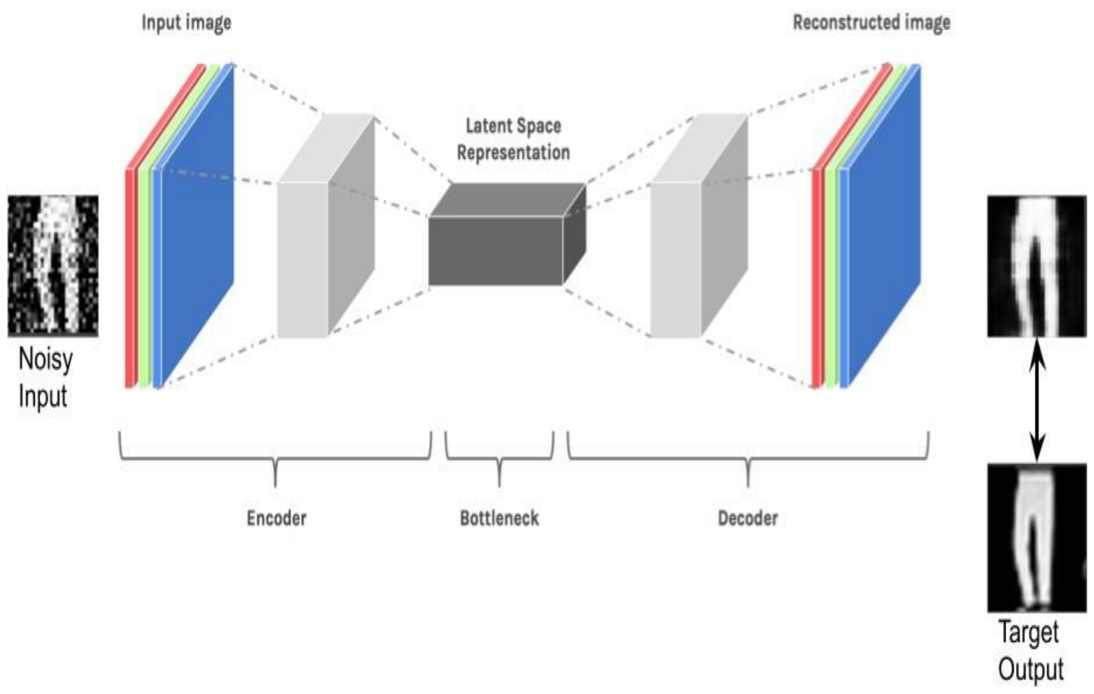
Overview

- Import Key libraries, dataset and visualize images
- Perform image normalization, pre-processing, and add random noise to images
- Build an Autoencoder using Keras with Tensorflow 2.0 as a backend
- Compile and fit Autoencoder model to training data
- Assess the performance of trained Autoencoder using various KPIs

Understanding the theory

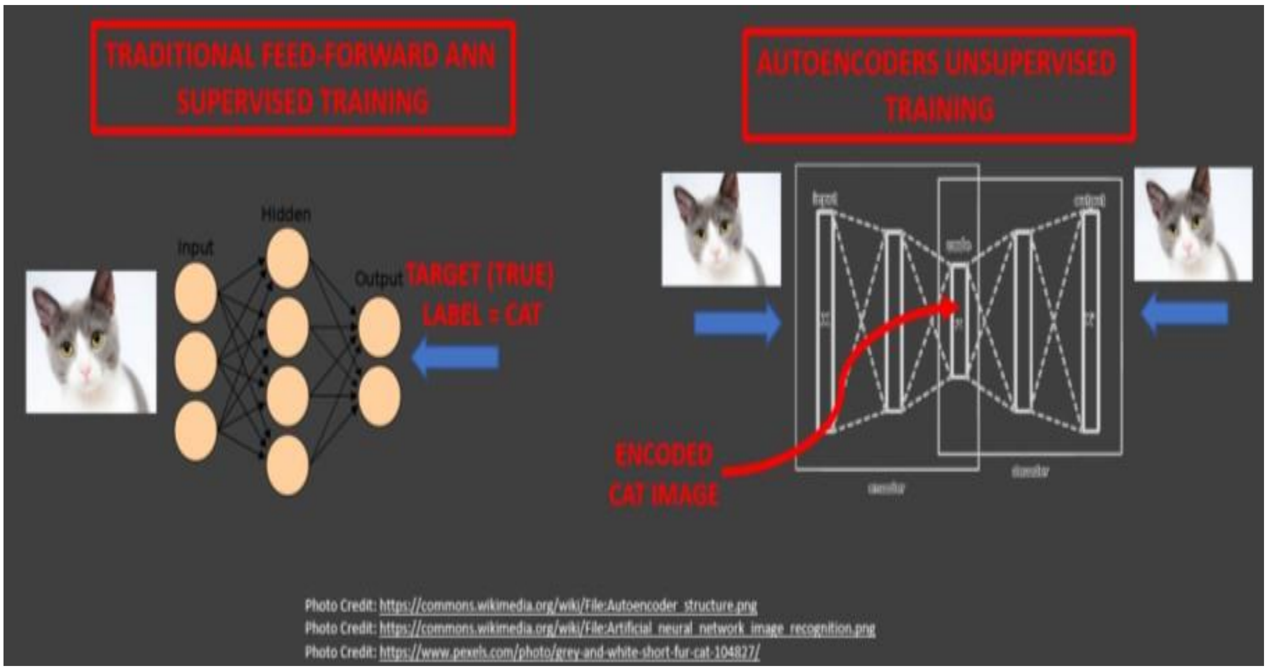
Autoencoder Intuition

Autoencoders are a type of Artificial Neural Networks that are used to perform a task of data encoding (representation learning). Autoencoders use same input data for input as well as output, crazy right?



Code Layer

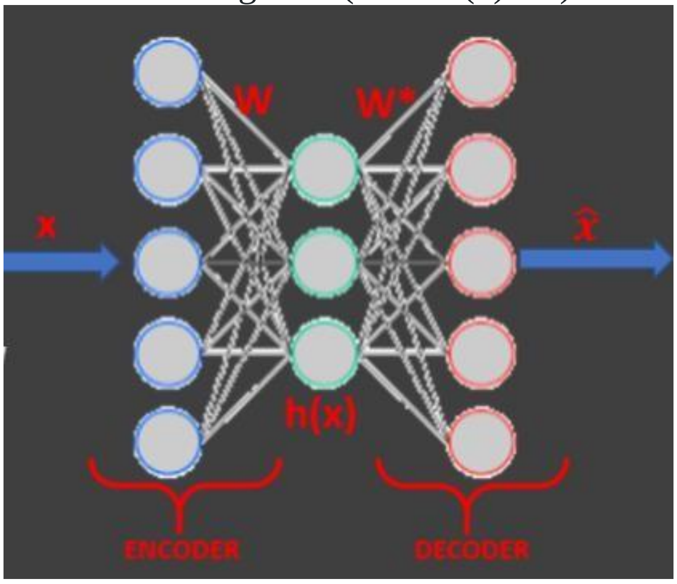
Autoencoders work by adding a bottleneck in the network. This bottleneck forces the network to create a compressed (encoded) version of the original input. Autoencoders work well if correlations exist between input data and (performs poorly if all the input data is independent).


TRADITIONAL FEED-FORWARD ANN SUPERVISED TRAINING
Input
Hidden
Output
TARGET (TRUE) LABEL = CAT
AUTOENCODERS UNSUPERVISED TRAINING
ENCODED CAT IMAGE
Photo Credit: https://commons.wikimedia.org/wiki/File:Autoencoder_structure.png
Photo Credit: https://commons.wikimedia.org/wiki/File:Artificial_neural_network_image_recognition.png
Photo Credit: https://www.pexels.com/photo/grey-and-white-short-fur-cat-104827/

Math behind Autoencoder

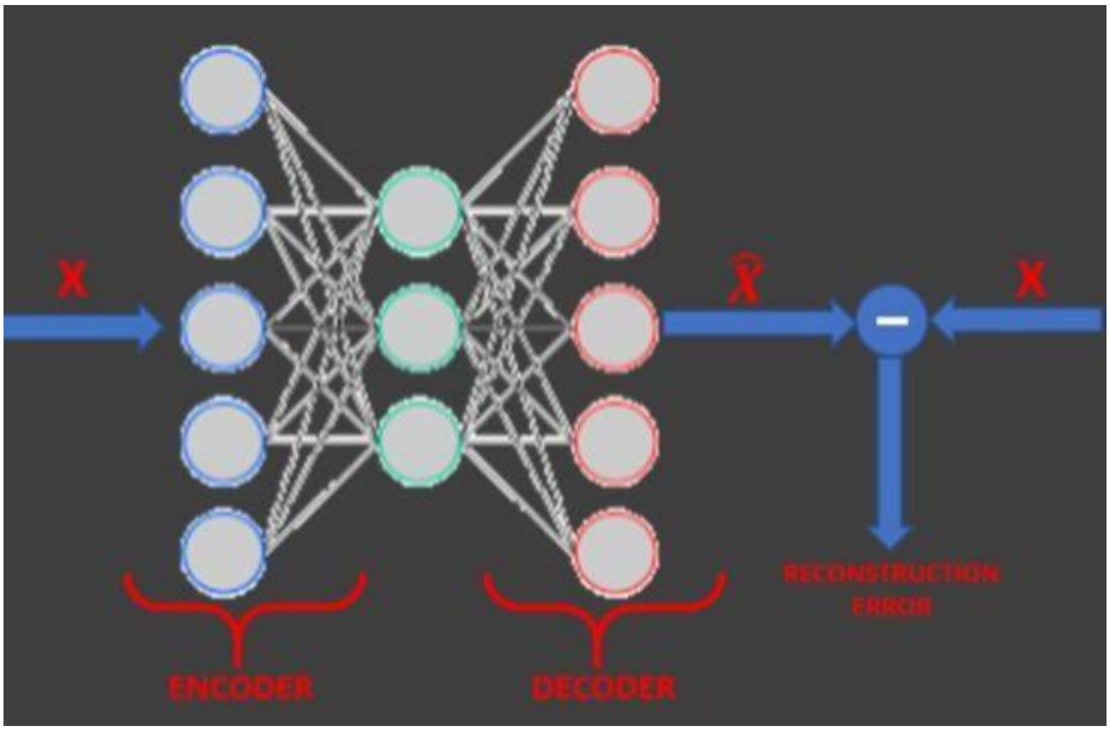
Encoder: h(x) = sigmoid (W * x + b)

Decoder: x̂ = sigmoid (W* * h(x) + c)



Reconstruction Error

Autoencoders objective is to minimize the reconstruction error which is the difference between the input X and the network output X̂ .



Autoencoders dimensionality reduction (latent space) is quite similar to PCA (Principal Component Analysis) if linear activation functions are used.

Step 1

Install following modules using pip:

notebook
tensorflow==2.0
pandas
numpy
matplotlib
seaborn
random

for example :

```
pip install pandas
pip install seaborn
```

Step 2

Open jupyter notebook on your local host or you can use Google Colab too.

Step 3

Code:

```
import keras
from keras.datasets import mnist
from keras import layers
import numpy as np
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt
(X_train,_),(X_test,_)=mnist.load_data()
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
```

```
X_test=np.reshape(X_test,(len(X_test),28,28,1))
noise_factor=0.5
X_train_noisy=X_train+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_tra
in.shape)
X_test_noisy=X_test+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_test.s
hape)
X_train_noisy=np.clip(X_train_noisy,0.,1.)
X_test_noisy=np.clip(X_test_noisy,0.,1.)
n=10
plt.figure(figsize=(20,2))
for i in range(1,n+1):
ax=plt.subplot(1,n,i)
plt.imshow(X_test_noisy[i].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
input_img=keras.Input(shape=(28,28,1))
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input_img)
x=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
encoded=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
x=layers.UpSampling2D((2,2))(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
x=layers.UpSampling2D((2,2))(x)
decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)
autoencoder=keras.Model(input_img,decoded)
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.fit(X_train_noisy,X_train,
epochs=3,
batch_size=128,
shuffle=True,
validation_data=(X_test_noisy,X_test),
callbacks=[TensorBoard(log_dir='/tmo/tb',histogram_freq=0,write_graph=False)])
predictions=autoencoder.predict(X_test_noisy)
m=10
plt.figure(figsize=(20,2))
for i in range(1,m+1):
ax=plt.subplot(1,m,i)
plt.imshow(predictions[i].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```

o/p: paste here