NEURAL NETWORKS AND DEEP LEARNING – ICP3

N. Manesh
700756918

GitHub link: https://github.com/Manesh1712/ICP3_neural

Video Link:
https://drive.google.com/file/d/1OKKOgulthO4GcVyuXy8vR71jCUe-3gLG/view?usp=drive_link

Use Case Description:

Predicting the diabetes disease

Programming elements: Keras Basics

In class programming:

1. Use the use case in the class: a. Add more dense layers to the existing code and check how the accuracy changes.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense

# Load dataset
dataset = pd.read_csv('diabetes.csv')
# Split dataset into features (X) and target variable (Y)
X = dataset.iloc[:, :-1]  # Features are all columns except the last one
Y = dataset.iloc[:, -1]   # Target variable is the last column

# Split dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=87)

# Define the model
np.random.seed(155)
my_first_nn = Sequential()
# Add dense layers
my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
# Additional dense layer
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3

# Output layer
my_first_nn.add(Dense(1, activation='sigmoid'))
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
print(my_first_nn.summary())
evaluation_result = my_first_nn.evaluate(X_test, Y_test)
print("Accuracy : ", evaluation_result[1]*100)
```

```
18/18 [==============================] - 0s 4ms/step - loss: 0.4891 - accuracy: 0.7722
Epoch 91/100
18/18 [==============================] - 0s 5ms/step - loss: 0.5025 - accuracy: 0.7513
Epoch 92/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5110 - accuracy: 0.7461
Epoch 93/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4990 - accuracy: 0.7583
Epoch 94/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4890 - accuracy: 0.7617
Epoch 95/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4828 - accuracy: 0.7670
Epoch 96/100
18/18 [==============================] - 0s 5ms/step - loss: 0.5209 - accuracy: 0.7391
Epoch 97/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4877 - accuracy: 0.7722
Epoch 98/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4881 - accuracy: 0.7687
Epoch 99/100
18/18 [==============================] - 0s 5ms/step - loss: 0.4862 - accuracy: 0.7635
Epoch 100/100
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                180

 dense_1 (Dense)             (None, 15)                315

 dense_2 (Dense)             (None, 10)                160

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 666 (2.60 KB)
Trainable params: 666 (2.60 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
6/6 [==============================] - 1s 4ms/step - loss: 0.5622 - accuracy: 0.7292
Accuracy :  72.91666865348816
```

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.

```python
# Importing the libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Loading the breast cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Splitting the dataset into the training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Normalizing the data using StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Building the model
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=6, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Training the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2)

# Evaluating the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 32/100
```

```
Epoch 84/100
37/37 [==============================] - 0s 9ms/step - loss: 0.0189 - accuracy: 0.9918 - val_loss: 0.0392 - val_accuracy:
0.9780
Epoch 85/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0175 - accuracy: 0.9945 - val_loss: 0.0423 - val_accuracy:
0.9780
Epoch 86/100
37/37 [==============================] - 0s 7ms/step - loss: 0.0173 - accuracy: 0.9973 - val_loss: 0.0435 - val_accuracy:
0.9780
Epoch 87/100
37/37 [==============================] - 0s 7ms/step - loss: 0.0156 - accuracy: 0.9973 - val_loss: 0.0427 - val_accuracy:
0.9780
Epoch 88/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0152 - accuracy: 0.9973 - val_loss: 0.0418 - val_accuracy:
0.9780
Epoch 89/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0145 - accuracy: 0.9973 - val_loss: 0.0420 - val_accuracy:
0.9780
Epoch 90/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0136 - accuracy: 0.9973 - val_loss: 0.0421 - val_accuracy:
```

```
37/37 [==============================] - 0s 6ms/step - loss: 0.0114 - accuracy: 1.0000 - val_loss: 0.0422 - val_accuracy: 0.9780
Epoch 96/100
37/37 [==============================] - 0s 7ms/step - loss: 0.0107 - accuracy: 1.0000 - val_loss: 0.0417 - val_accuracy: 0.9780
Epoch 97/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0103 - accuracy: 1.0000 - val_loss: 0.0417 - val_accuracy: 0.9780
Epoch 98/100
37/37 [==============================] - 0s 7ms/step - loss: 0.0104 - accuracy: 0.9973 - val_loss: 0.0407 - val_accuracy: 0.9780
Epoch 99/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0101 - accuracy: 1.0000 - val_loss: 0.0433 - val_accuracy: 0.9780
Epoch 100/100
37/37 [==============================] - 0s 6ms/step - loss: 0.0093 - accuracy: 1.0000 - val_loss: 0.0416 - val_accuracy: 0.9780
4/4 [==============================] - 0s 5ms/step - loss: 0.2150 - accuracy: 0.9474
Accuracy: 94.74
```

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import load_breast_cancer

# Load Breast Cancer dataset
data = load_breast_cancer()
X, Y = data.data, data.target

# Normalize the data
sc = StandardScaler()
X_normalized = sc.fit_transform(X)

# Split dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_normalized, Y, test_size=0.25, random_state=87)

# Define the model
np.random.seed(155)
my_first_nn = Sequential()

# Add dense layers
my_first_nn.add(Dense(20, input_dim=X_train.shape[1], activation='relu')) # hidden layer 1
my_first_nn.add(Dense(15, activation='relu')) # hidden layer 2
# Additional dense layer
my_first_nn.add(Dense(10, activation='relu')) # hidden layer 3

# Output layer
my_first_nn.add(Dense(1, activation='sigmoid'))
# Compile the model
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100, initial_epoch=0)
print(my_first_nn.summary())
# Evaluate the model on test data
evaluation_result = my_first_nn.evaluate(X_test, Y_test)
print("Accuracy : ", evaluation_result[1]*100)  # Print accuracy value
```

```
14/14 [==============================] - 0s 6ms/step - loss: 0.0020 - accuracy: 1.0000
Epoch 93/100
14/14 [==============================] - 0s 5ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 94/100
14/14 [==============================] - 0s 5ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 95/100
14/14 [==============================] - 0s 6ms/step - loss: 0.0019 - accuracy: 1.0000
Epoch 96/100
14/14 [==============================] - 0s 6ms/step - loss: 0.0018 - accuracy: 1.0000
Epoch 97/100
14/14 [==============================] - 0s 5ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 98/100
14/14 [==============================] - 0s 5ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 99/100
14/14 [==============================] - 0s 5ms/step - loss: 0.0017 - accuracy: 1.0000
Epoch 100/100
14/14 [==============================] - 0s 6ms/step - loss: 0.0016 - accuracy: 1.0000
Model: "sequential_2"
_________________________________________________________________
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 20)                620

 dense_8 (Dense)             (None, 15)                315

 dense_9 (Dense)             (None, 10)                160

 dense_10 (Dense)            (None, 1)                 11

=================================================================
Total params: 1106 (4.32 KB)
Trainable params: 1106 (4.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
5/5 [==============================] - 0s 7ms/step - loss: 0.3594 - accuracy: 0.9650
Accuracy :  96.50349617004395
```

In class programming:

Use Image Classification on the hand written digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

```python
#Given image classification source code

from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                   validation_data=(test_data, test_labels_one_hot))
```

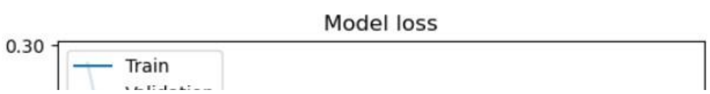
```
                   validation_data=(test_data, test_labels_one_hot))
```

```
(28, 28)
784
Epoch 1/10
235/235 [==============================] - 8s 26ms/step - loss: 0.2844 - accuracy: 0.9133 - val_loss: 0.1291 - val_accuracy: 0.9576
Epoch 2/10
235/235 [==============================] - 6s 26ms/step - loss: 0.0996 - accuracy: 0.9692 - val_loss: 0.1385 - val_accuracy: 0.9550
Epoch 3/10
235/235 [==============================] - 4s 15ms/step - loss: 0.0619 - accuracy: 0.9807 - val_loss: 0.1220 - val_accuracy: 0.9610
Epoch 4/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0439 - accuracy: 0.9858 - val_loss: 0.0878 - val_accuracy: 0.9751
Epoch 5/10
235/235 [==============================] - 5s 22ms/step - loss: 0.0309 - accuracy: 0.9900 - val_loss: 0.0668 - val_accuracy: 0.9801
Epoch 6/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0227 - accuracy: 0.9926 - val_loss: 0.0670 - val_accuracy: 0.9797
Epoch 7/10
235/235 [==============================] - 6s 27ms/step - loss: 0.0167 - accuracy: 0.9947 - val_loss: 0.0930 - val_accuracy: 0.9748
Epoch 8/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0122 - accuracy: 0.9962 - val_loss: 0.0820 - val_accuracy: 0.9799
Epoch 9/10
235/235 [==============================] - 5s 23ms/step - loss: 0.0096 - accuracy: 0.9971 - val_loss: 0.0723 - val_accuracy: 0.9825
Epoch 10/10
235/235 [==============================] - 6s 25ms/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0901 - val_accuracy: 0.9783
```

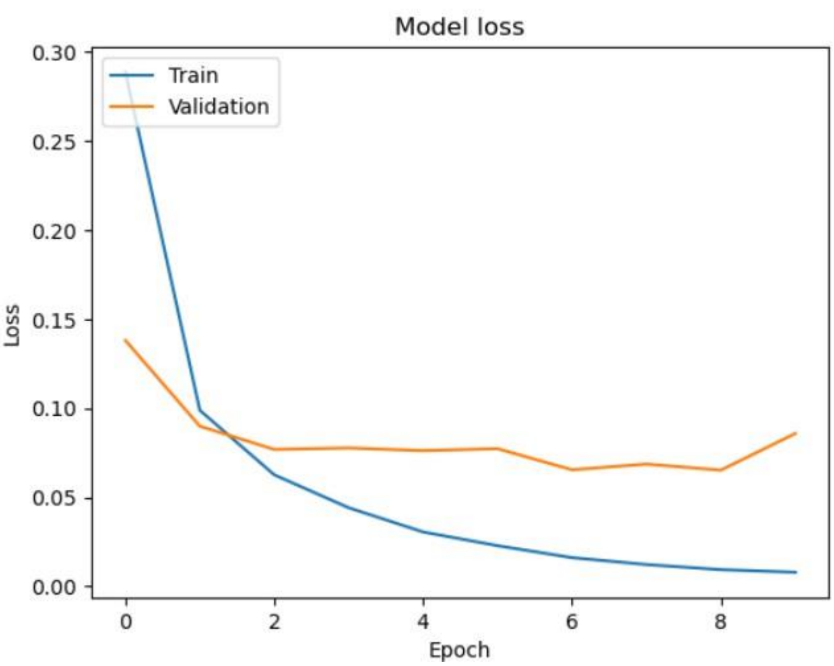
]:

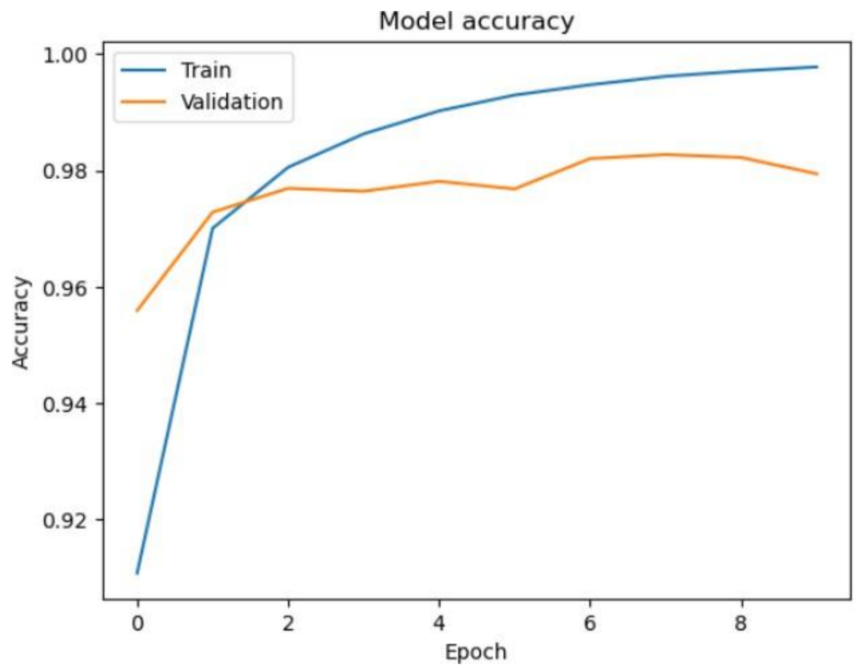
```python
#Task 1
import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils import to_categorical
# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float') / 255.0
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float') / 255.0
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
# Define the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
# Fit the model
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10,
                    verbose=1,validation_data=(test_data, test_labels_one_hot))

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 6s 20ms/step - loss: 0.2886 - accuracy: 0.9108 - val_loss: 0.1380 - val_accuracy:
0.9559
Epoch 2/10
235/235 [==============================] - 5s 21ms/step - loss: 0.0988 - accuracy: 0.9700 - val_loss: 0.0899 - val_accuracy:
0.9728
Epoch 3/10
235/235 [==============================] - 5s 19ms/step - loss: 0.0626 - accuracy: 0.9805 - val_loss: 0.0769 - val_accuracy:
0.9769
Epoch 4/10
235/235 [==============================] - 5s 19ms/step - loss: 0.0442 - accuracy: 0.9862 - val_loss: 0.0777 - val_accuracy:
0.9764
Epoch 5/10
235/235 [==============================] - 4s 18ms/step - loss: 0.0306 - accuracy: 0.9902 - val_loss: 0.0762 - val_accuracy:
0.9781
Epoch 6/10
235/235 [==============================] - 5s 19ms/step - loss: 0.0229 - accuracy: 0.9929 - val_loss: 0.0773 - val_accuracy:
0.9768
Epoch 7/10
235/235 [==============================] - 4s 19ms/step - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.0654 - val_accuracy:
0.9820
Epoch 8/10
235/235 [==============================] - 5s 20ms/step - loss: 0.0123 - accuracy: 0.9961 - val_loss: 0.0686 - val_accuracy:
0.9827
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0094 - accuracy: 0.9970 - val_loss: 0.0653 - val_accuracy:
0.9822
Epoch 10/10
235/235 [==============================] - 5s 20ms/step - loss: 0.0080 - accuracy: 0.9977 - val_loss: 0.0858 - val_accuracy:
0.9794
```




```
235/235 [==============================] - 5s 20ms/step - loss: 0.0080 - accuracy: 0.9
0.9794
```

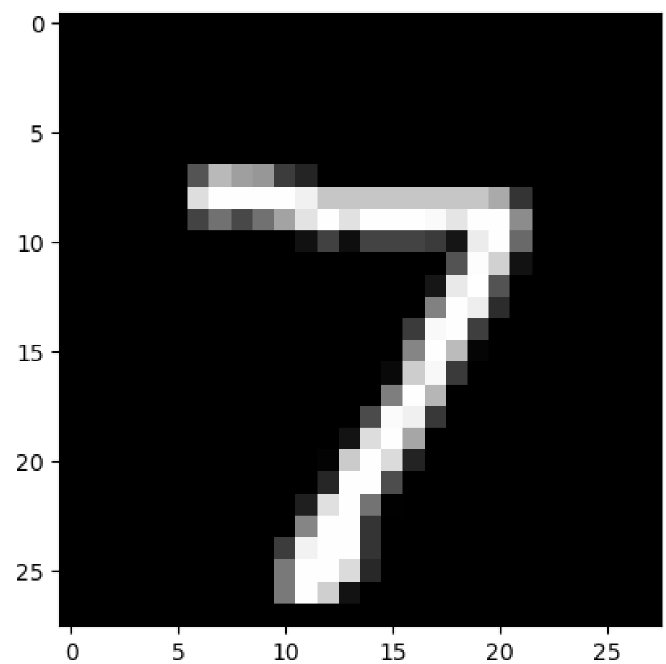





2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```python
import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils import to_categorical

# Load dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Process the data
dimData = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimData).astype('float') / 255.0
test_data = test_images.reshape(test_images.shape[0], dimData).astype('float') / 255.0
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Define the model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

# Fit the model
model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1)

# Plot one of the images in the test data
plt.imshow(test_images[0], cmap='gray')
plt.show()

# Make prediction on the single image
prediction = model.predict(test_data[0].reshape(1, 784))
print("Prediction:", np.argmax(prediction))
```

```
Epoch 1/10
235/235 [==============================] - 4s 15ms/step - loss: 0.2848 - accuracy: 0.9128
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0979 - accuracy: 0.9696
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0628 - accuracy: 0.9804
Epoch 4/10
235/235 [==============================] - 3s 14ms/step - loss: 0.0424 - accuracy: 0.9868
Epoch 5/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0305 - accuracy: 0.9902
Epoch 6/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0217 - accuracy: 0.9932
Epoch 7/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0157 - accuracy: 0.9952
Epoch 8/10
235/235 [==============================] - 4s 15ms/step - loss: 0.0129 - accuracy: 0.9958
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.0093 - accuracy: 0.9972
Epoch 10/10
235/235 [==============================] - 4s 16ms/step - loss: 0.0066 - accuracy: 0.9978
```




```
1/1 [==============================] - 0s 234ms/step
Prediction: 7
```

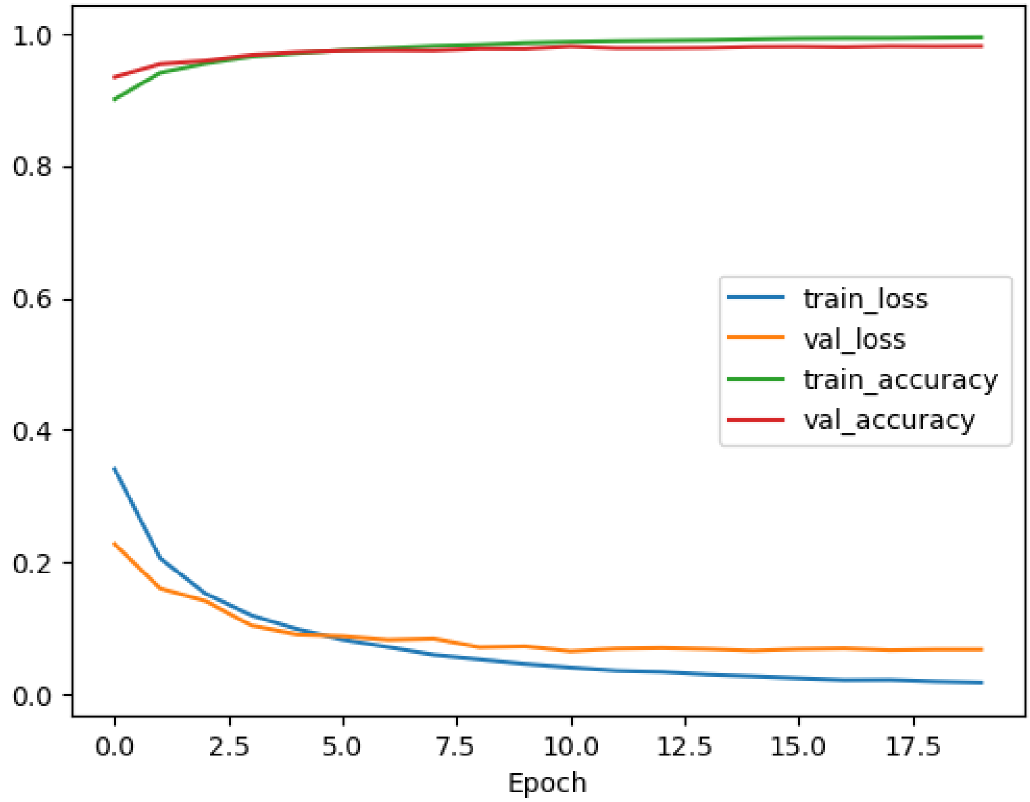
3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize pixel values to range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# convert class labels to binary class matrices
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# create a list of models to train
models = []

# model with 1 hidden layer and tanh activation
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with tanh', model))

# model with 1 hidden layer and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('1 hidden layer with sigmoid', model))
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))
```

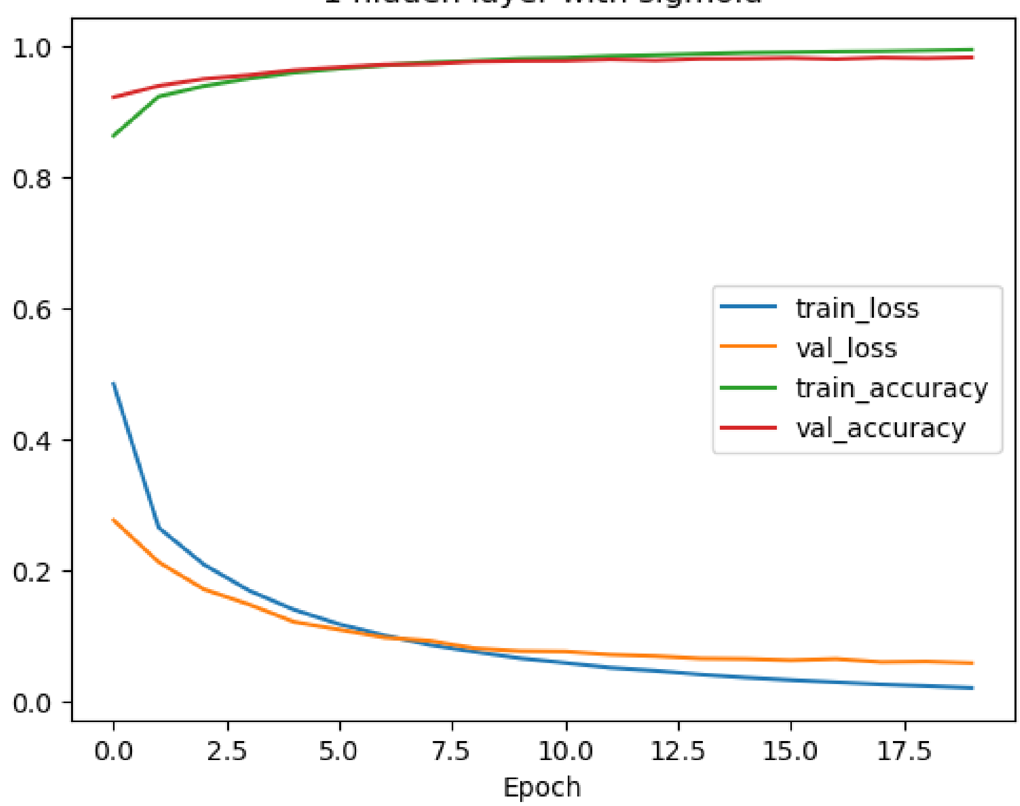
```python
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))

# model with 2 hidden layers and sigmoid activation
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='sigmoid'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with sigmoid', model))

# train each model and plot loss and accuracy curves
for name, model in models:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                        epochs=20, batch_size=128, verbose=0)
    # plot loss and accuracy curves
    plt.plot(history.history['loss'], label='train_loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.plot(history.history['accuracy'], label='train_accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.title(name)
    plt.xlabel('Epoch')
    plt.legend()
    plt.show()

    # evaluate the model on test data
    loss, accuracy = model.evaluate(x_test.reshape(-1, 784), y_test, verbose=0)
    print('{} - Test loss: {:.4f}, Test accuracy: {:.4f}'.format(name, loss, accuracy))
```

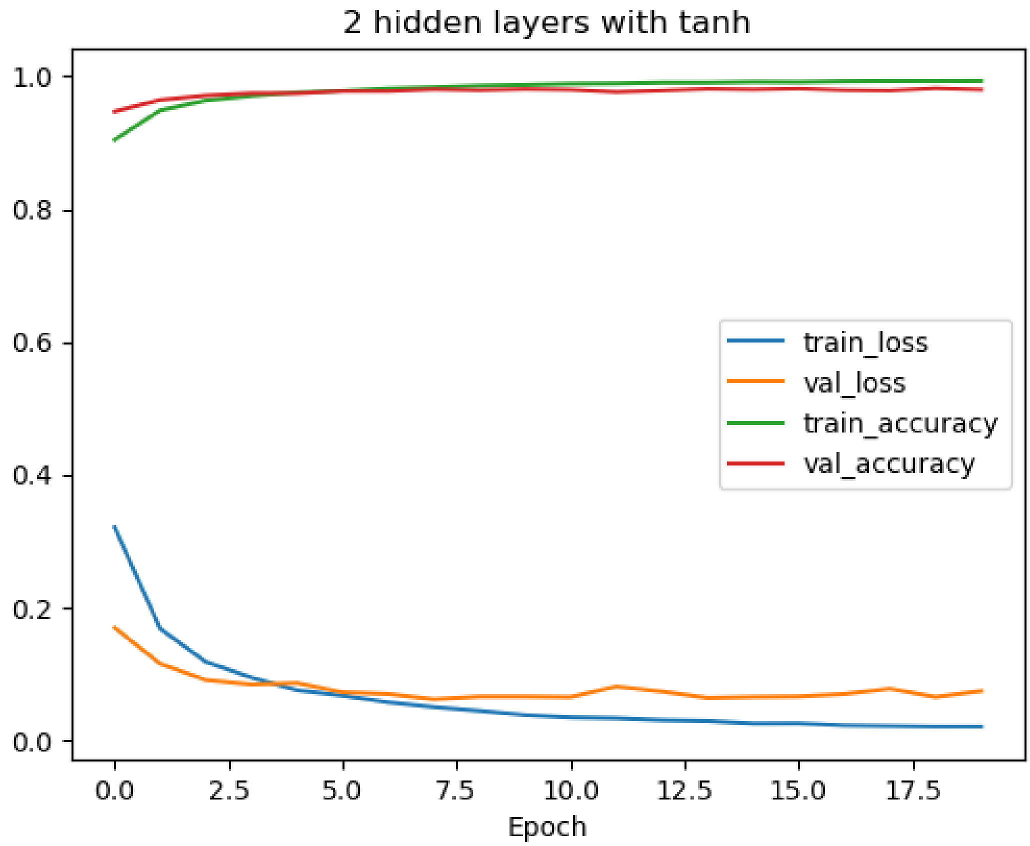
0.8

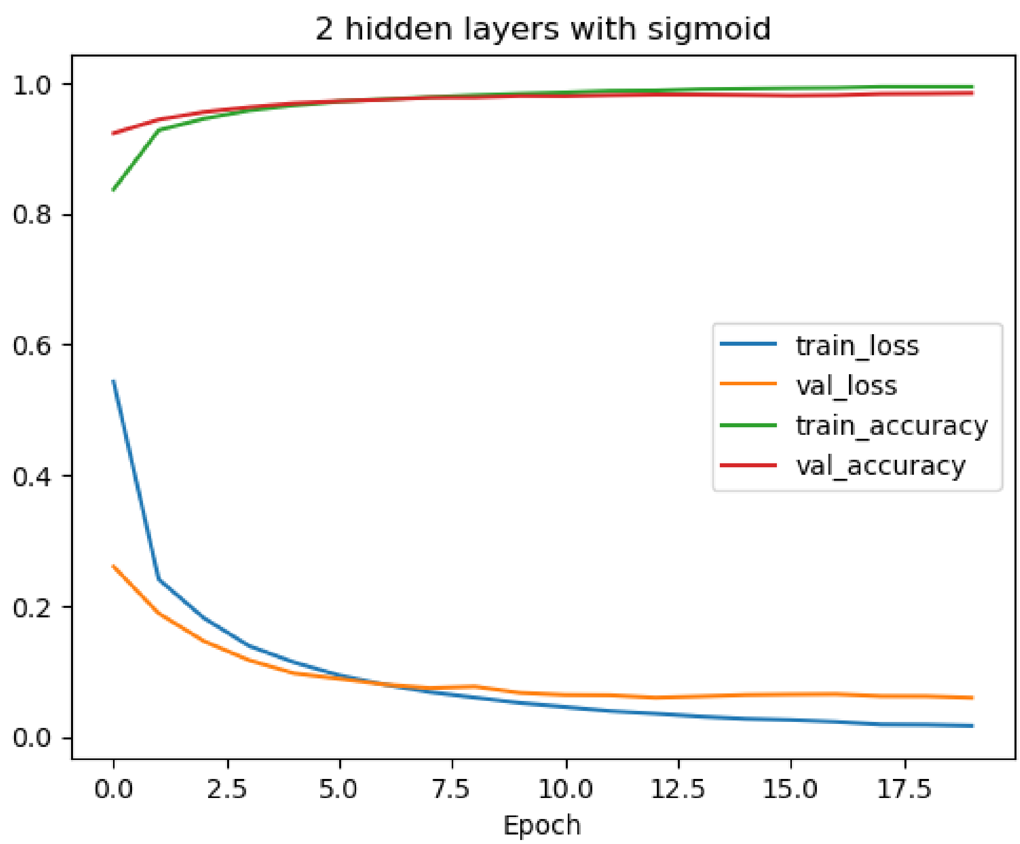
1 hidden layer with tanh

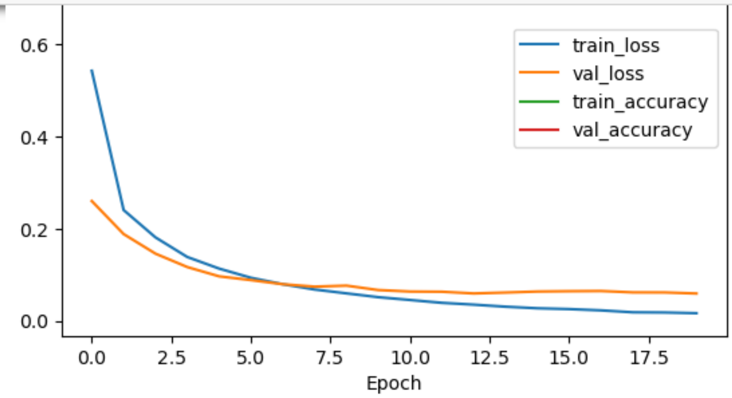


1 hidden layer with sigmoid




2 hidden layers with tanh
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy


2 hidden layers with sigmoid
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch
train_loss
val_loss
train_accuracy
val_accuracy




```
2 hidden layers with sigmoid - Test loss: 0.0594, Test accuracy: 0.9843
```

4. Run the same code without scaling the images and check the performance?

```python
# Load the MNIST dataset
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam  # Import Adam optimizer separately

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Convert the pixel values to floats and normalize them to the range 0-1
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# Convert the target variable to a one-hot encoding using to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Create a neural network model with 3 hidden layers and tanh activation
model = Sequential()
model.add(Dense(256, input_dim=784, activation='tanh'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(10, activation='softmax'))

# Compile the model using the Adam optimizer
model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

# Train the model
history = model.fit(x_train.reshape(-1, 784), y_train, epochs=10, validation_data=(x_test.reshape(-1, 784), y_test))

# Plot the loss and accuracy for both training and validation data
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 17s 8ms/step - loss: 0.2392 - accuracy: 0.9294 - val_loss: 0.1523 - val_accurac
y: 0.9548
Epoch 2/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.1143 - accuracy: 0.9651 - val_loss: 0.1311 - val_accurac
y: 0.9591
Epoch 3/10
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0810 - accuracy: 0.9749 - val_loss: 0.0917 - val_accurac
y: 0.9719
Epoch 4/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0608 - accuracy: 0.9810 - val_loss: 0.0909 - val_accurac
y: 0.9735
Epoch 5/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0470 - accuracy: 0.9848 - val_loss: 0.0931 - val_accurac
y: 0.9728
Epoch 6/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0398 - accuracy: 0.9871 - val_loss: 0.0914 - val_accurac
y: 0.9733
Epoch 7/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0332 - accuracy: 0.9893 - val_loss: 0.0944 - val_accurac
y: 0.9747
Epoch 8/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0292 - accuracy: 0.9905 - val_loss: 0.0904 - val_accurac
y: 0.9754
Epoch 9/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0270 - accuracy: 0.9909 - val_loss: 0.0886 - val_accurac
y: 0.9746
Epoch 10/10
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0239 - accuracy: 0.9923 - val_loss: 0.0996 - val_accurac
y: 0.9739
```