CS5720 Neural Networks & Deep Learning ASSIGNMENT –6

NAME : MANDHA SAIKUMAR REDDY

STUDENT ID : 700765227

GITHUB LINK:

https://github.com/saikumarreddyMandha/Assignment-4/blob/main/ICP4.ipynb

VIDEO LINK :

https://drive.google.com/file/d/1f4__mIKfWs_5bXu-8AN9sALISxRS6Xbl/view?usp=drive_link

1. Use the use case in the class:

a. Add more Dense layers to the existing code and check how the accuracy changes.

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

c. 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()

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

```
[2] #read the data
    import pandas as pd
    data = pd.read_csv('diabetes.csv')
```

```
[3] path_to_csv = 'diabetes.csv'
```

```
[4] import keras
    import pandas
    from keras.models import Sequential
    from keras.layers import Dense, Activation

    # load dataset
    from sklearn.model_selection import train_test_split
    import pandas as pd
    import numpy as np

    dataset = pd.read_csv(path_to_csv, header=None).values

    X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
                                                        test_size=0.25, random_state=87)
    np.random.seed(155)
    my_first_nn = Sequential() # create model
    my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
    my_first_nn.add(Dense(4, activation='relu')) # hidden layer
    my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
    my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                         initial_epoch=0)
    print(my_first_nn.summary())
    print(my_first_nn.evaluate(X_test, Y_test))
```

```
Epoch 1/100
18/18 [==============================] - 1s 2ms/step - loss: 5.1130 - acc: 0.5972
Epoch 2/100
18/18 [==============================] - 0s 2ms/step - loss: 2.1959 - acc: 0.6024
Epoch 3/100
18/18 [==============================] - 0s 2ms/step - loss: 1.0648 - acc: 0.6059
Epoch 4/100
18/18 [==============================] - 0s 3ms/step - loss: 0.8923 - acc: 0.5938
Epoch 5/100
18/18 [==============================] - 0s 2ms/step - loss: 0.8121 - acc: 0.5955
Epoch 6/100
18/18 [==============================] - 0s 2ms/step - loss: 0.7410 - acc: 0.6146
Epoch 7/100
18/18 [==============================] - 0s 3ms/step - loss: 0.7091 - acc: 0.6198
Epoch 8/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6817 - acc: 0.6285
Epoch 9/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6785 - acc: 0.6128
Epoch 10/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6582 - acc: 0.6441
Epoch 11/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6489 - acc: 0.6424
Epoch 12/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6647 - acc: 0.6424
Epoch 13/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6523 - acc: 0.6632
Epoch 14/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6244 - acc: 0.6736
Epoch 15/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6247 - acc: 0.6771
Epoch 16/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6252 - acc: 0.6736
Epoch 17/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6112 - acc: 0.6771
Epoch 18/100
```

```
Epoch 19/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6103 - acc: 0.6840
Epoch 20/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5969 - acc: 0.6892
Epoch 21/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5948 - acc: 0.6858
Epoch 22/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6157 - acc: 0.6806
Epoch 23/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6200 - acc: 0.6684
Epoch 24/100
18/18 [==============================] - 0s 2ms/step - loss: 0.6101 - acc: 0.6840
Epoch 25/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5914 - acc: 0.6892
Epoch 26/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5902 - acc: 0.6840
Epoch 27/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5819 - acc: 0.6962
Epoch 28/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5884 - acc: 0.6910
Epoch 29/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5832 - acc: 0.6892
Epoch 30/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5827 - acc: 0.6962
Epoch 31/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5787 - acc: 0.6944
Epoch 32/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5750 - acc: 0.6997
Epoch 33/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5860 - acc: 0.6771
Epoch 34/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5857 - acc: 0.6823
Epoch 35/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5734 - acc: 0.6910
Epoch 36/100
```

```
Epoch 93/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5417 - acc: 0.7083
Epoch 94/100
18/18 [==============================] - 0s 4ms/step - loss: 0.5462 - acc: 0.7101
Epoch 95/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5420 - acc: 0.7153
Epoch 96/100
18/18 [==============================] - 0s 2ms/step - loss: 0.5391 - acc: 0.7188
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5370 - acc: 0.7135
Epoch 98/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5363 - acc: 0.7101
Epoch 99/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5425 - acc: 0.7153
Epoch 100/100
18/18 [==============================] - 0s 3ms/step - loss: 0.5425 - acc: 0.7188
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 20)                180

 dense_1 (Dense)             (None, 4)                 84

 dense_2 (Dense)             (None, 1)                 5

=================================================================
Total params: 269 (1.05 KB)
Trainable params: 269 (1.05 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
6/6 [==============================] - 0s 3ms/step - loss: 0.5981 - acc: 0.6771
[0.598054826259613, 0.6770833134651184]
```

```
[5] #read the data
    data = pd.read_csv('breastcancer.csv')
```

```
[6] path_to_csv = 'sample_data/breastcancer.csv'
```

```
import keras            (module) pd
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

```
14/14 [==============================] - 0s 2ms/step - loss: 0.1872 - acc: 0.9202
Epoch 96/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2868 - acc: 0.9178
Epoch 97/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1735 - acc: 0.9390
Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1446 - acc: 0.9249
Epoch 99/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1459 - acc: 0.9437
Epoch 100/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1356 - acc: 0.9460
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 20)                620

 dense_4 (Dense)             (None, 1)                 21

=================================================================
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
5/5 [==============================] - 0s 5ms/step - loss: 0.2368 - acc: 0.9091
[0.2367641180753708, 0.9090909361839294]
```

```
[8] #read the data
    data = pd.read_csv('breastcancer.csv')
```

```
[9] path_to_csv = 'breastcancer.csv'
```

```
[10] from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
```

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# load dataset
cancer_data = load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(cancer_data.data, cancer_data.target,
                                                    test_size=0.25, random_state=87)
np.random.seed(155)
my_nn = Sequential() # create model
my_nn.add(Dense(20, input_dim=30, activation='relu')) # hidden layer 1
my_nn.add(Dense(1, activation='sigmoid')) # output layer
my_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_nn_fitted = my_nn.fit(X_train, Y_train, epochs=100,
                         initial_epoch=0)
print(my_nn.summary())
print(my_nn.evaluate(X_test, Y_test))
```

Loading...

```
Epoch 1/100
14/14 [==============================] - 1s 2ms/step - loss: 81.4649 - acc: 0.3803
Epoch 2/100
14/14 [==============================] - 0s 2ms/step - loss: 48.0038 - acc: 0.3803
Epoch 3/100
14/14 [==============================] - 0s 2ms/step - loss: 15.7359 - acc: 0.4155
Epoch 4/100
14/14 [==============================] - 0s 3ms/step - loss: 1.8353 - acc: 0.8474
Epoch 5/100
14/14 [==============================] - 0s 2ms/step - loss: 1.3822 - acc: 0.8991
Epoch 6/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4995 - acc: 0.9131
Epoch 7/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4021 - acc: 0.9272
Epoch 8/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3421 - acc: 0.9249
Epoch 9/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3255 - acc: 0.9178
Epoch 10/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3129 - acc: 0.9178
Epoch 11/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3079 - acc: 0.9225
Epoch 12/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3394 - acc: 0.9249
Epoch 13/100
14/14 [==============================] - 0s 2ms/step - loss: 0.3118 - acc: 0.9178
Epoch 14/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2987 - acc: 0.9178
Epoch 15/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2943 - acc: 0.9155
Epoch 16/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2984 - acc: 0.9202
Epoch 17/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2873 - acc: 0.9225
Epoch 18/100
```

```
[11] Epoch 98/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1983 - acc: 0.9249
Epoch 99/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2011 - acc: 0.9366
Epoch 100/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1773 - acc: 0.9249
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 20)                620

 dense_6 (Dense)             (None, 1)                 21

=================================================================
Total params: 641 (2.50 KB)
Trainable params: 641 (2.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 0.3693 - acc: 0.8671
[0.36931079626083374, 0.867132842540741]
```

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.

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.

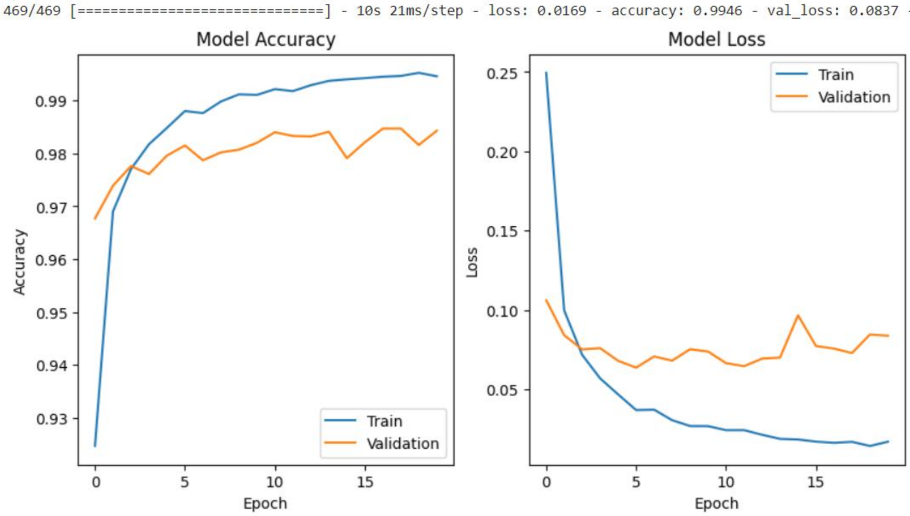
3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

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

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

# 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 simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the model and record the training history
history = model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
                    epochs=20, batch_size=128)
```

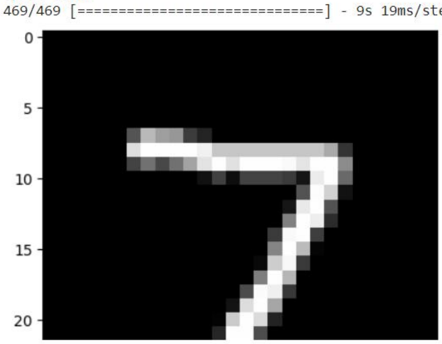
```python
# plot the training and validation accuracy and loss curves
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

plt.subplot(1, 2, 2)
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 right')

plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/20
469/469 [==============================] - 12s 24ms/step - loss: 0.2495 - accuracy: 0.9247 - val_loss: 0.1061 - val_accuracy: 0.9677
Epoch 2/20
469/469 [==============================] - 11s 22ms/step - loss: 0.0997 - accuracy: 0.9690 - val_loss: 0.0840 - val_accuracy: 0.9739
Epoch 3/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0717 - accuracy: 0.9771 - val_loss: 0.0751 - val_accuracy: 0.9776
Epoch 4/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0568 - accuracy: 0.9817 - val_loss: 0.0759 - val_accuracy: 0.9761
Epoch 5/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0467 - accuracy: 0.9848 - val_loss: 0.0679 - val_accuracy: 0.9796
Epoch 6/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0369 - accuracy: 0.9880 - val_loss: 0.0636 - val_accuracy: 0.9815
Epoch 7/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0371 - accuracy: 0.9876 - val_loss: 0.0707 - val_accuracy: 0.9787
Epoch 8/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0305 - accuracy: 0.9898 - val_loss: 0.0680 - val_accuracy: 0.9802
Epoch 9/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0268 - accuracy: 0.9912 - val_loss: 0.0752 - val_accuracy: 0.9807
Epoch 10/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0267 - accuracy: 0.9911 - val_loss: 0.0737 - val_accuracy: 0.9820
Epoch 11/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0242 - accuracy: 0.9922 - val_loss: 0.0664 - val_accuracy: 0.9840
Epoch 12/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0242 - accuracy: 0.9918 - val_loss: 0.0645 - val_accuracy: 0.9833
Epoch 13/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0213 - accuracy: 0.9929 - val_loss: 0.0693 - val_accuracy: 0.9832
Epoch 14/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0187 - accuracy: 0.9937 - val_loss: 0.0700 - val_accuracy: 0.9841
Epoch 15/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0183 - accuracy: 0.9940 - val_loss: 0.0966 - val_accuracy: 0.9791
Epoch 16/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0169 - accuracy: 0.9942 - val_loss: 0.0772 - val_accuracy: 0.9821
Epoch 17/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0162 - accuracy: 0.9945 - val_loss: 0.0756 - val_accuracy: 0.9847
Epoch 18/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0168 - accuracy: 0.9947 - val_loss: 0.0728 - val_accuracy: 0.9847
Epoch 19/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0142 - accuracy: 0.9952 - val_loss: 0.0845 - val_accuracy: 0.9816
Epoch 20/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0169 - accuracy: 0.9946 - val_loss: 0.0837 - val_accuracy: 0.9843
```

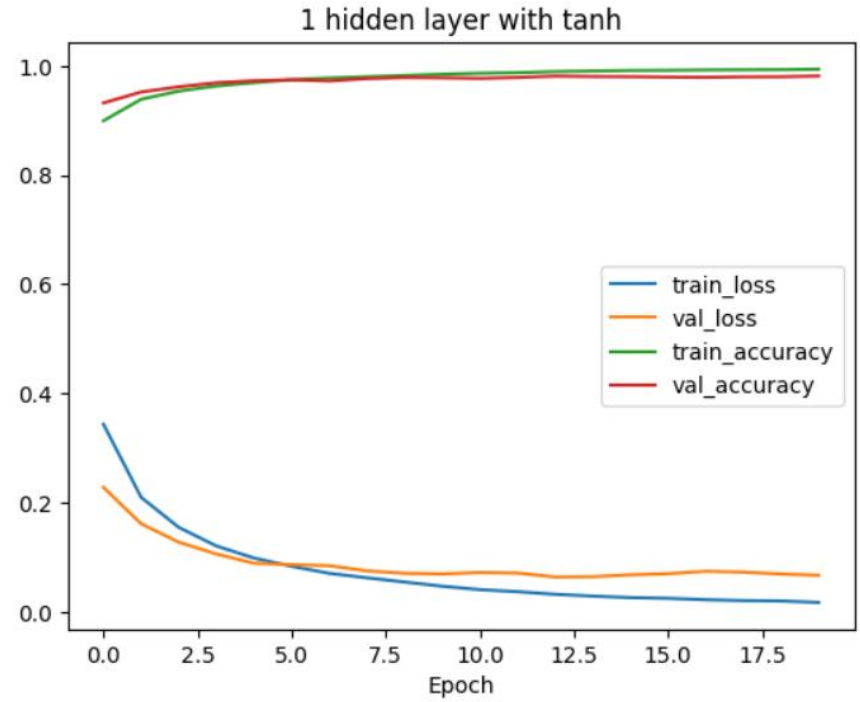


```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# 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 simple neural network model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# train the model
model.fit(x_train.reshape(-1, 784), y_train, validation_data=(x_test.reshape(-1, 784), y_test),
```

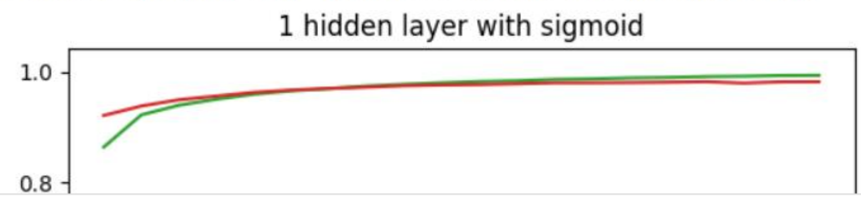
```
469/469 [==============================] - 9s 20ms/step - loss: 0.0205 - accuracy: 0.9933 - val_loss: 0.0763 - val_accuracy: 0.9803
Epoch 13/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0217 - accuracy: 0.9926 - val_loss: 0.0759 - val_accuracy: 0.9826
Epoch 14/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0177 - accuracy: 0.9942 - val_loss: 0.0756 - val_accuracy: 0.9814
Epoch 15/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0200 - accuracy: 0.9936 - val_loss: 0.0739 - val_accuracy: 0.9822
Epoch 16/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0193 - accuracy: 0.9937 - val_loss: 0.0647 - val_accuracy: 0.9845
Epoch 17/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0162 - accuracy: 0.9947 - val_loss: 0.0811 - val_accuracy: 0.9807
Epoch 18/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0170 - accuracy: 0.9946 - val_loss: 0.0790 - val_accuracy: 0.9820
Epoch 19/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0791 - val_accuracy: 0.9808
Epoch 20/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0148 - accuracy: 0.9949 - val_loss: 0.0826 - val_accuracy: 0.9829
```

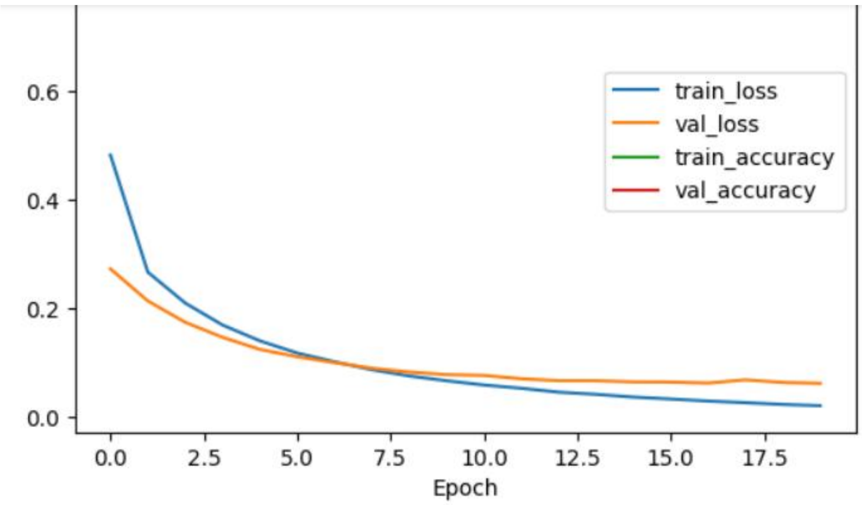


```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# 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,)))
```

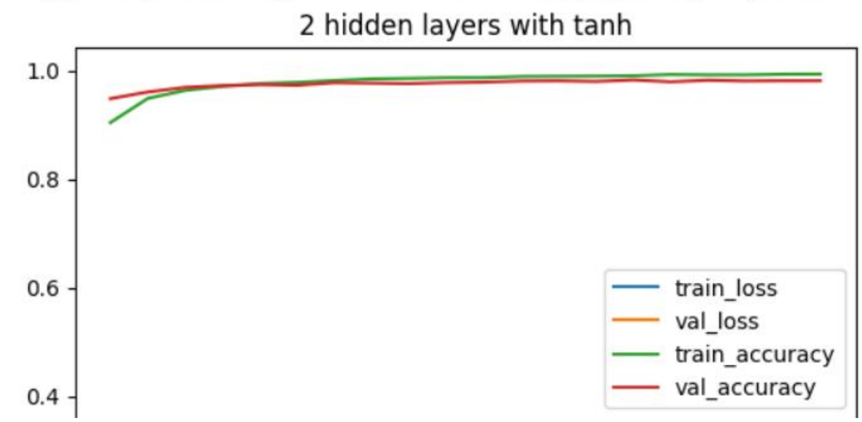


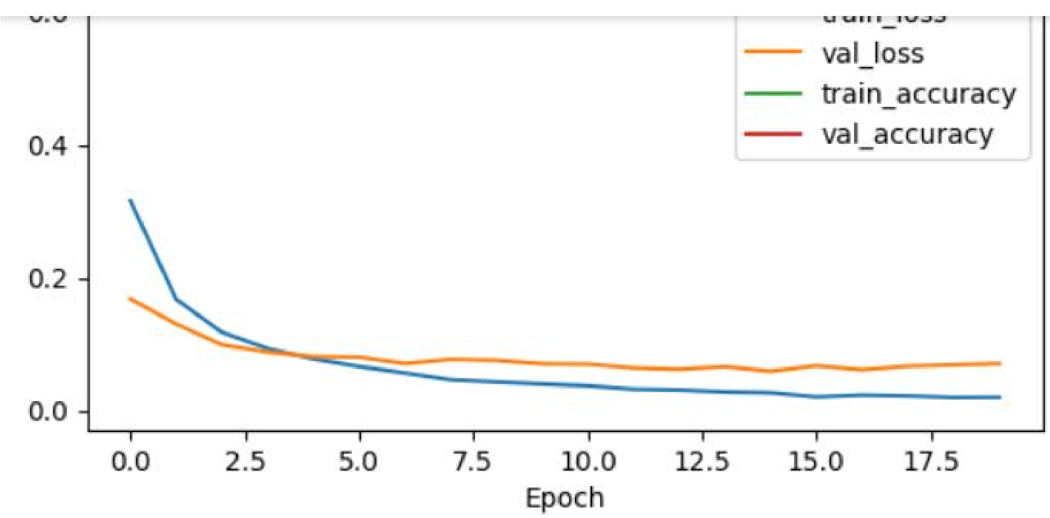

```
1 hidden layer with tanh - Test loss: 0.0668, Test accuracy: 0.9815
```





1 hidden layer with sigmoid - Test loss: 0.0622, Test accuracy: 0.9821

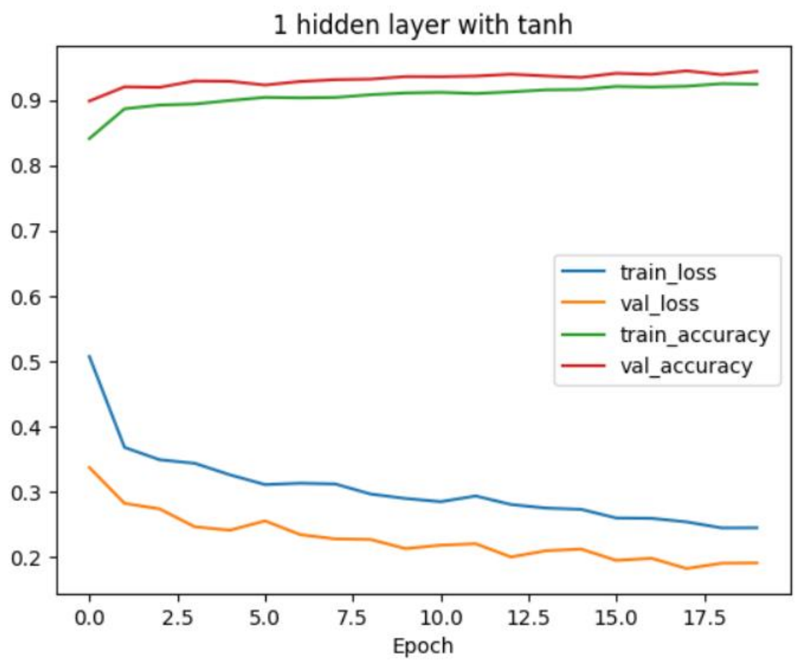




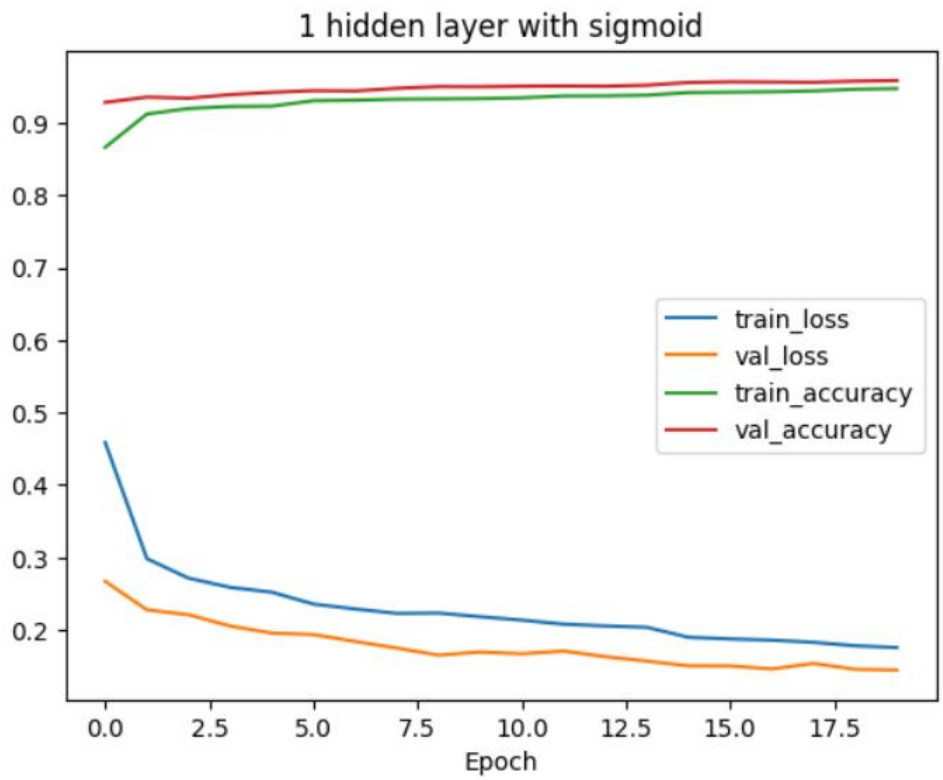
2 hidden layers with tanh - Test loss: 0.0704, Test accuracy: 0.9813

```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()

# 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))
```

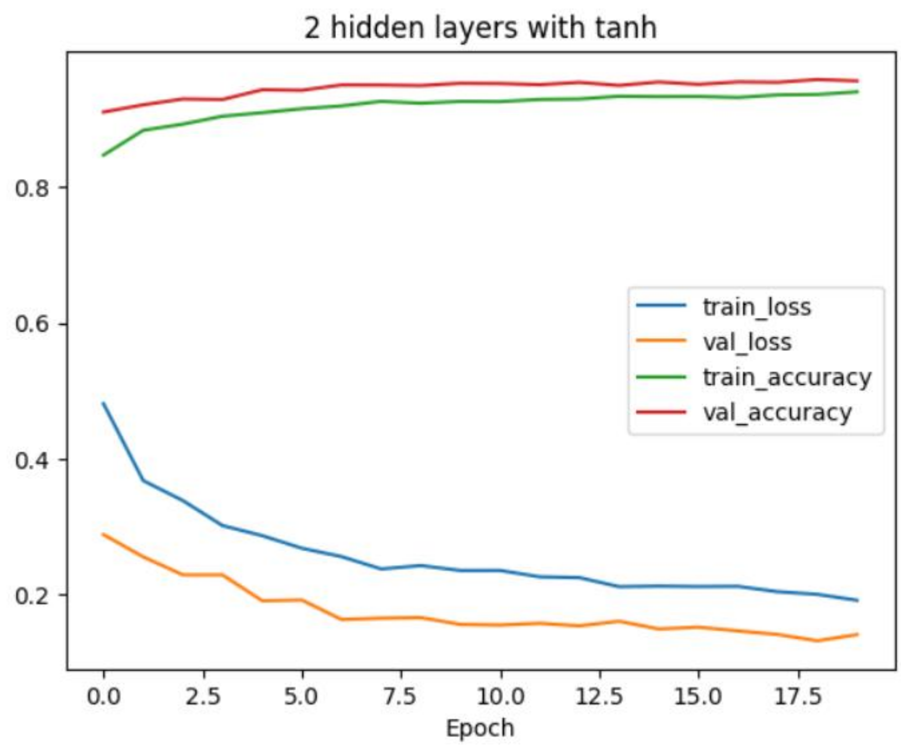
```
model.add(Dense(num_classes, activation='softmax'))
models.append(('2 hidden layers with tanh', model))

# 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))
```

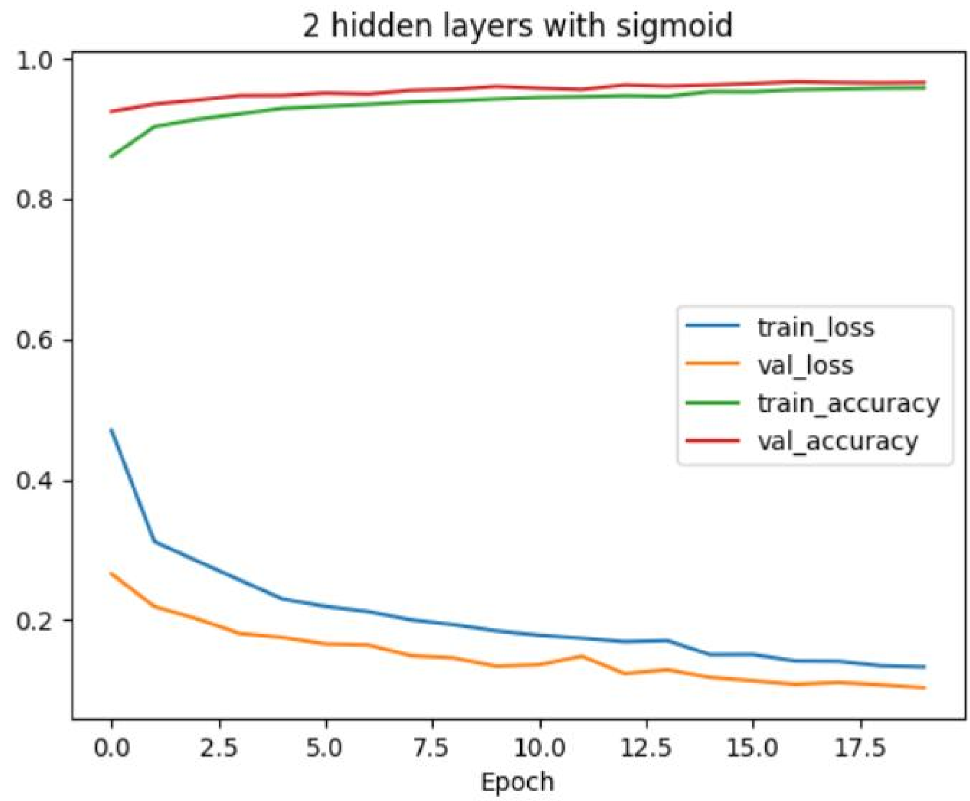



```
1 hidden layer with tanh - Test loss: 0.1909, Test accuracy: 0.9436
```




```
1 hidden layer with sigmoid - Test loss: 0.1449, Test accuracy: 0.9580
```






```
2 hidden layers with sigmoid - Test loss: 0.1039, Test accuracy: 0.9652
```