# Assignment - 6

Gayathri Jagiri
700745729

Video link : https://drive.google.com/drive/my-drive

GitHub link : https://github.com/JGayathri3/NN6.git

Use the use case in the class:

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

```
#read the data
data = pd.read_csv('sample_data/diabetes.csv')
```

```
[ ] path_to_csv = 'sample_data/diabetes.csv'
```

```
[ ] import keras
    import pandas
    from keras.models import Sequential
    from keras.layers.core 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: 18.2141 - acc: 0.3385
```

Output:

```
18/18 [==============================] - 0s 3ms/step - loss: 0.6425 - acc: 0.6771
Epoch 97/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6370 - acc: 0.6823
Epoch 98/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6220 - acc: 0.6806
Epoch 99/100
18/18 [==============================] - 0s 4ms/step - loss: 0.6210 - acc: 0.6858
Epoch 100/100
18/18 [==============================] - 0s 3ms/step - loss: 0.6212 - acc: 0.6823
Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_89 (Dense)            (None, 20)                180

 dense_90 (Dense)            (None, 4)                 84

 dense_91 (Dense)            (None, 1)                 5

=================================================================
Total params: 269
Trainable params: 269
Non-trainable params: 0
_________________________________________________________________
None
6/6 [==============================] - 0s 4ms/step - loss: 0.7119 - acc: 0.5833
[0.7118666172027588, 0.5833333134651184]
```

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

```
#read the data
data = pd.read_csv('sample_data/breastcancer.csv')
```

```
path_to_csv = 'sample_data/breastcancer.csv'
```

```
import keras
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers.core 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))
```

```
Epoch 1/100
14/14 [==============================] - 1s 5ms/step - loss: 67.9584 - acc: 0.3803
Epoch 2/100
14/14 [==============================] - 0s 3ms/step - loss: 20.8848 - acc: 0.3897
Epoch 3/100
14/14 [==============================] - 0s 3ms/step - loss: 5.6956 - acc: 0.6901
```

# Output:

```
Epoch 90/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1810 - acc: 0.9319
Epoch 91/100
14/14 [==============================] - 0s 3ms/step - loss: 0.1508 - acc: 0.9319
Epoch 92/100
14/14 [==============================] - 0s 2ms/step - loss: 0.2480 - acc: 0.9225
Epoch 93/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2020 - acc: 0.9343
Epoch 94/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1698 - acc: 0.9343
Epoch 95/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1509 - acc: 0.9390
Epoch 96/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1522 - acc: 0.9390
Epoch 97/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1466 - acc: 0.9343
Epoch 98/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1683 - acc: 0.9319
Epoch 99/100
14/14 [==============================] - 0s 3ms/step - loss: 0.2092 - acc: 0.9178
Epoch 100/100
14/14 [==============================] - 0s 2ms/step - loss: 0.1453 - acc: 0.9484
Model: "sequential_42"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 20)                620

 dense_99 (Dense)            (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 4ms/step - loss: 0.3893 - acc: 0.8881
[0.3892970681190491, 0.8881118893623352]
```

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

```
#read the data
data = pd.read_csv('sample_data/breastcancer.csv')
```

```
path_to_csv = 'sample_data/breastcancer.csv'
```

```
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.core 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))
```

# Output:

```
14/14 [==============================] - 0s 2ms/step - loss: 0.6074 - acc: 0.9178
Epoch 92/100
14/14 [==============================] - 0s 2ms/step - loss: 0.6664 - acc: 0.9061
Epoch 93/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4741 - acc: 0.9296
Epoch 94/100
14/14 [==============================] - 0s 3ms/step - loss: 0.4954 - acc: 0.9155
Epoch 95/100
14/14 [==============================] - 0s 3ms/step - loss: 0.4736 - acc: 0.9225
Epoch 96/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4443 - acc: 0.9343
Epoch 97/100
14/14 [==============================] - 0s 3ms/step - loss: 0.4802 - acc: 0.9202
Epoch 98/100
14/14 [==============================] - 0s 2ms/step - loss: 0.4229 - acc: 0.9225
Epoch 99/100
14/14 [==============================] - 0s 3ms/step - loss: 0.5408 - acc: 0.9131
Epoch 100/100
14/14 [==============================] - 0s 3ms/step - loss: 0.3975 - acc: 0.9272
Model: "sequential_45"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_104 (Dense)           (None, 20)                620

 dense_105 (Dense)           (None, 1)                 21

=================================================================
Total params: 641
Trainable params: 641
Non-trainable params: 0
_________________________________________________________________
None
5/5 [==============================] - 0s 3ms/step - loss: 1.3143 - acc: 0.7902
[1.314283013343811, 0.7902097702026367]
```

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

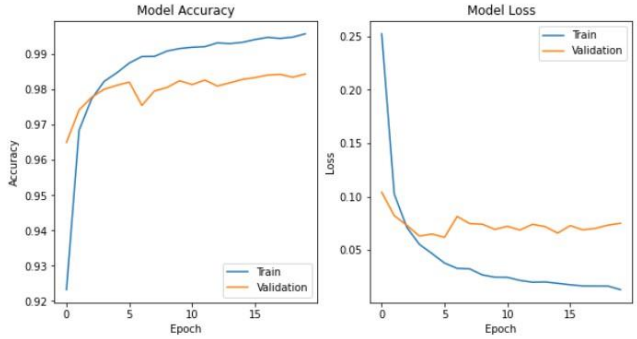
1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

```
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)

# plot the training and validation accuracy and loss curves
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
```

```
# 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, 78
                    epochs=20, batch_size=128)

# 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()
```

Output:

```
469/469 [==============================] - 16s 35ms/step - loss: 0.0201 - accuracy: 0.9932 - val_loss: 0.0741 - val_accuracy: 0.9810
Epoch 14/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0203 - accuracy: 0.9930 - val_loss: 0.0720 - val_accuracy: 0.9819
Epoch 15/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0190 - accuracy: 0.9934 - val_loss: 0.0660 - val_accuracy: 0.9829
Epoch 16/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0176 - accuracy: 0.9942 - val_loss: 0.0729 - val_accuracy: 0.9834
Epoch 17/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0165 - accuracy: 0.9948 - val_loss: 0.0690 - val_accuracy: 0.9841
Epoch 18/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0164 - accuracy: 0.9945 - val_loss: 0.0704 - val_accuracy: 0.9843
Epoch 19/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0164 - accuracy: 0.9949 - val_loss: 0.0734 - val_accuracy: 0.9835
Epoch 20/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0131 - accuracy: 0.9958 - val_loss: 0.0752 - val_accuracy: 0.9844
```

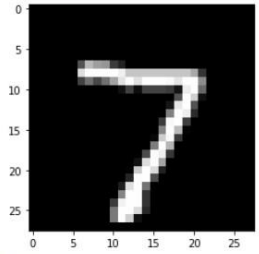



2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

```
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),
          epochs=20, batch_size=128)

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

# make a prediction on the image using the trained model
prediction = model.predict(x_test[0].reshape(1, -1))
print('Model prediction:', np.argmax(prediction))
```

Output:

```
469/469 [==============================] - 12s 26ms/step - loss: 0.0248 - accuracy: 0.9915 - val_loss: 0.0804 - val_accuracy: 0.9806
Epoch 11/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0238 - accuracy: 0.9918 - val_loss: 0.0753 - val_accuracy: 0.9807
Epoch 12/20
469/469 [==============================] - 12s 25ms/step - loss: 0.0237 - accuracy: 0.9923 - val_loss: 0.0743 - val_accuracy: 0.9814
Epoch 13/20
469/469 [==============================] - 12s 25ms/step - loss: 0.0211 - accuracy: 0.9929 - val_loss: 0.0785 - val_accuracy: 0.9803
Epoch 14/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0168 - accuracy: 0.9945 - val_loss: 0.0694 - val_accuracy: 0.9838
Epoch 15/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0192 - accuracy: 0.9934 - val_loss: 0.0786 - val_accuracy: 0.9820
Epoch 16/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0184 - accuracy: 0.9938 - val_loss: 0.0768 - val_accuracy: 0.9827
Epoch 17/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0164 - accuracy: 0.9948 - val_loss: 0.0775 - val_accuracy: 0.9823
Epoch 18/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0162 - accuracy: 0.9948 - val_loss: 0.0800 - val_accuracy: 0.9822
Epoch 19/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0145 - accuracy: 0.9951 - val_loss: 0.0873 - val_accuracy: 0.9820
Epoch 20/20
469/469 [==============================] - 11s 24ms/step - loss: 0.0140 - accuracy: 0.9957 - val_loss: 0.0807 - val_accuracy: 0.9841
```



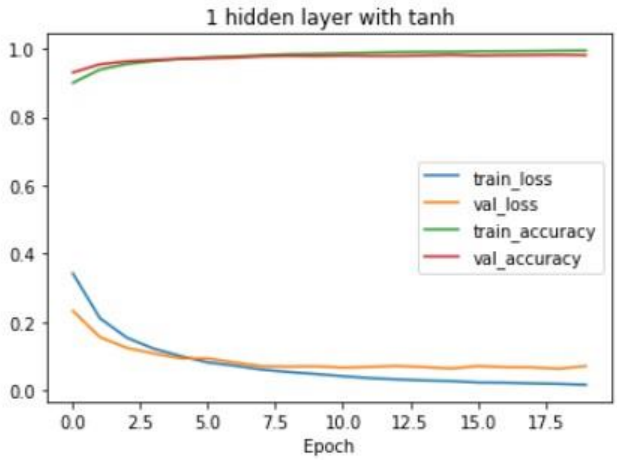

```
1/1 [==============================] - 0s 120ms/step
Model prediction: 7
```

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.

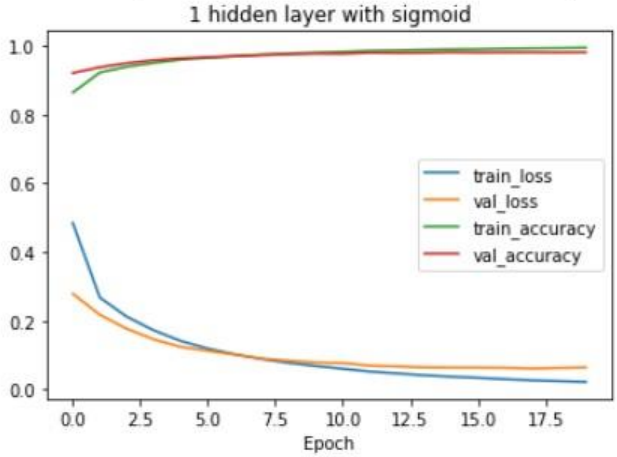
```python
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
import numpy as np

# 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 with 2 hidden layers and tanh activation
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))

# 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))
```

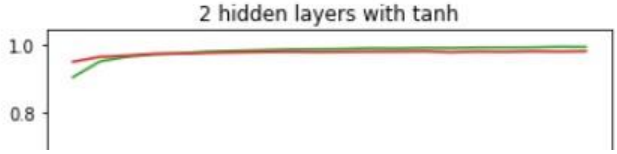
# Output:



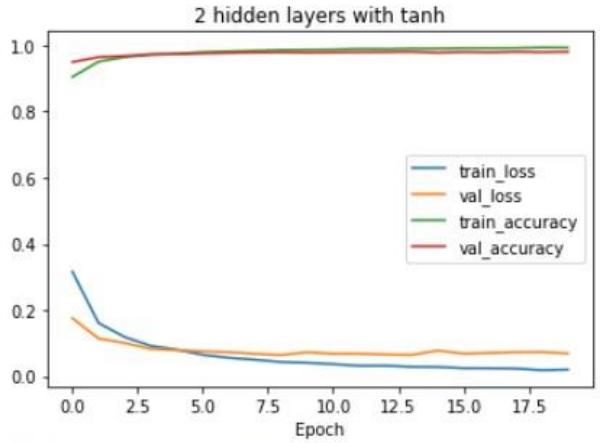

```
1 hidden layer with tanh - Test loss: 0.0716, Test accuracy: 0.9809
```



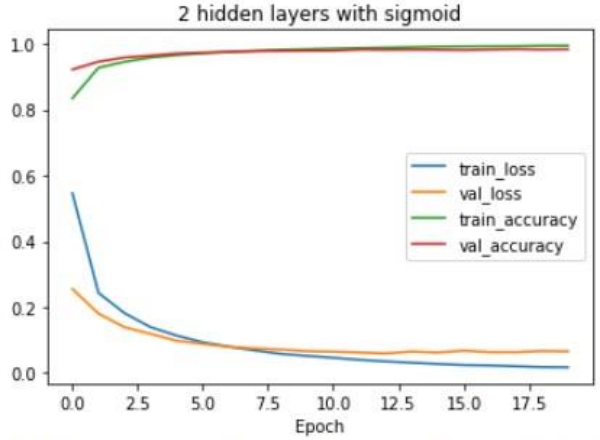

```
1 hidden layer with sigmoid - Test loss: 0.0642, Test accuracy: 0.9809
```



```
1 hidden layer with sigmoid - Test loss: 0.0642, Test accuracy: 0.9809
```




```
2 hidden layers with tanh - Test loss: 0.0686, Test accuracy: 0.9808
```



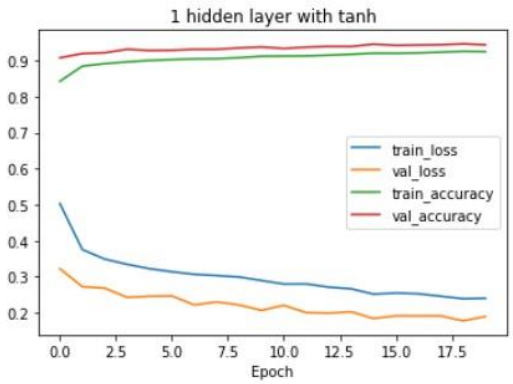

```
2 hidden layers with sigmoid - Test loss: 0.0663, Test accuracy: 0.9830
```

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

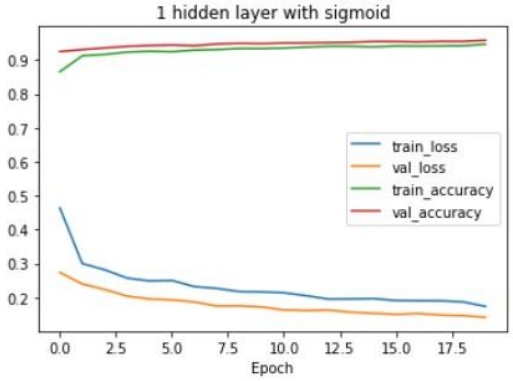
```
[ ]  import keras
     from keras.datasets import mnist
     from keras.models import Sequential
     from keras.layers import Dense, Dropout
     import matplotlib.pyplot as plt
     import numpy as np

     # 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 with 2 hidden layers and tanh activation
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))

# 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))
```

Output:



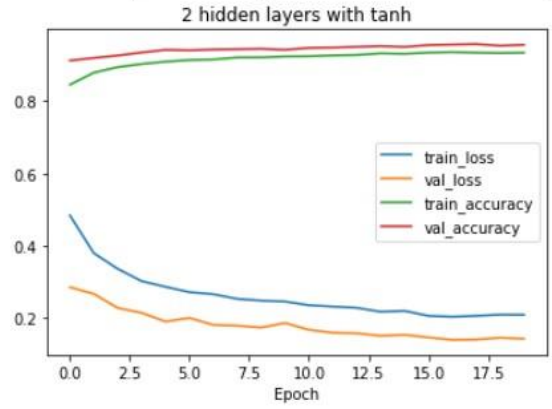

```
1 hidden layer with tanh - Test loss: 0.1895, Test accuracy: 0.9439
```



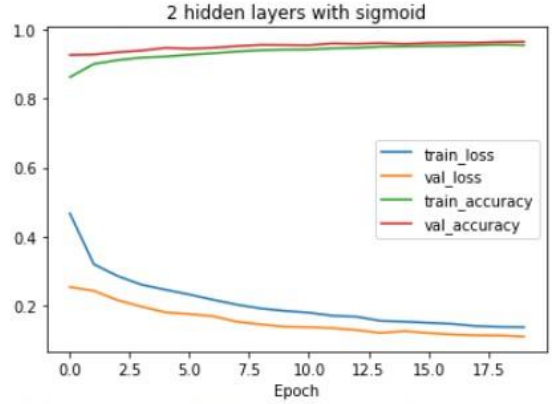

```
1 hidden layer with sigmoid - Test loss: 0.1420, Test accuracy: 0.9582
```

Epoch

1 hidden layer with sigmoid - Test loss: 0.1420, Test accuracy: 0.9582




2 hidden layers with tanh - Test loss: 0.1422, Test accuracy: 0.9563




2 hidden layers with sigmoid - Test loss: 0.1095, Test accuracy: 0.9652