Neural Network Project Demo

February 2019

1 Topic

In this project I'll build a neural network and train it on a GPU-enabled server to recognize handwritten digits (from 0 to 9) using the MNIST dataset. At the end, I made a GUI using Tkinter to show my results.

2 Dataset

MNIST contains 70,000 images of handwritten digits: 60,000 for training and 10,000 for testing. The images are grayscale, 28x28 pixels, and centered to reduce preprocessing and get started quicker.

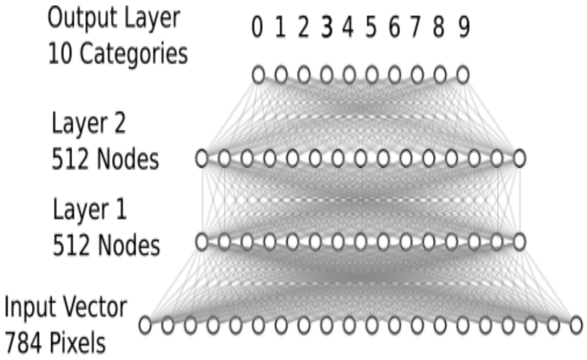
3 DNN Model

3.1 Architecture

This model is built on a linear stack of layers with the sequential model. There are two hidden layers in total. All of them are dense layers. A snippet of code of constructing the model/network is shown as follow:

```
        model = Sequential()
        model.add(Dense(512, input_shape=(784,)))
        model.add(Activation('relu'))
        model.add(Dropout(0.2))

        model.add(Dense(512))
        model.add(Activation('relu'))
        model.add(Dropout(0.2))

        model.add(Dense(10))
        model.add(Activation('softmax'))

```

Listing 1: Architecture



The above shows a graph of the architecture used in this project. Notice in the code section I added dropout as a way to prevent overfitting.

3.2 Input: Shape of Tensor

X_train shape is (60000, 28, 28)
X_test shape is (10000, 28, 28)
We transform it into a float32 array of shape (60000, 28 * 28).

3.3 Output: Shape of Tensor

y_train shape is (60000,)
y_test shape is (10000,)

3.4 Shape of Output Tensor for Each Layer

Output of first hidden layer: (60000, 512)
Output of second hidden layer: (60000, 512)
Output of the output layer: (60000, 10)

4 Hyperparameters

4.1 List of Hyperparameters

In this project, there are three hyperparameters I choose to tune: batch size; epochs and dropout.

4.2 Range of Value of Hyperparameters Tried

Batch Size	32, 64 and 128
Epochs	10 – 40
Dropout	0.1 – 0.5

4.3 Optimal Hyperparameters Found

Batch Size	128
Epochs	20
Dropout	0.2

5 Annotated Code

```
        import numpy as np
        import os
        from keras.datasets import mnist
        from keras.models import Sequential, load_model
        from keras.layers.core import Dense, Dropout, Activation
        from keras.utils import np_utils

        (X_train, y_train), (X_test, y_test) = mnist.load_data()

        X_train = X_train.reshape(60000, 784)
        X_test = X_test.reshape(10000, 784)
        X_train = X_train.astype('float32')
        X_test = X_test.astype('float32')
        X_train /= 255
        X_test /= 255

        n_classes = 10
        Y_train = np_utils.to_categorical(y_train, n_classes)
        Y_test = np_utils.to_categorical(y_test, n_classes)

        model = Sequential()
        model.add(Dense(512, input_shape=(784,)))
        model.add(Activation('relu'))
```

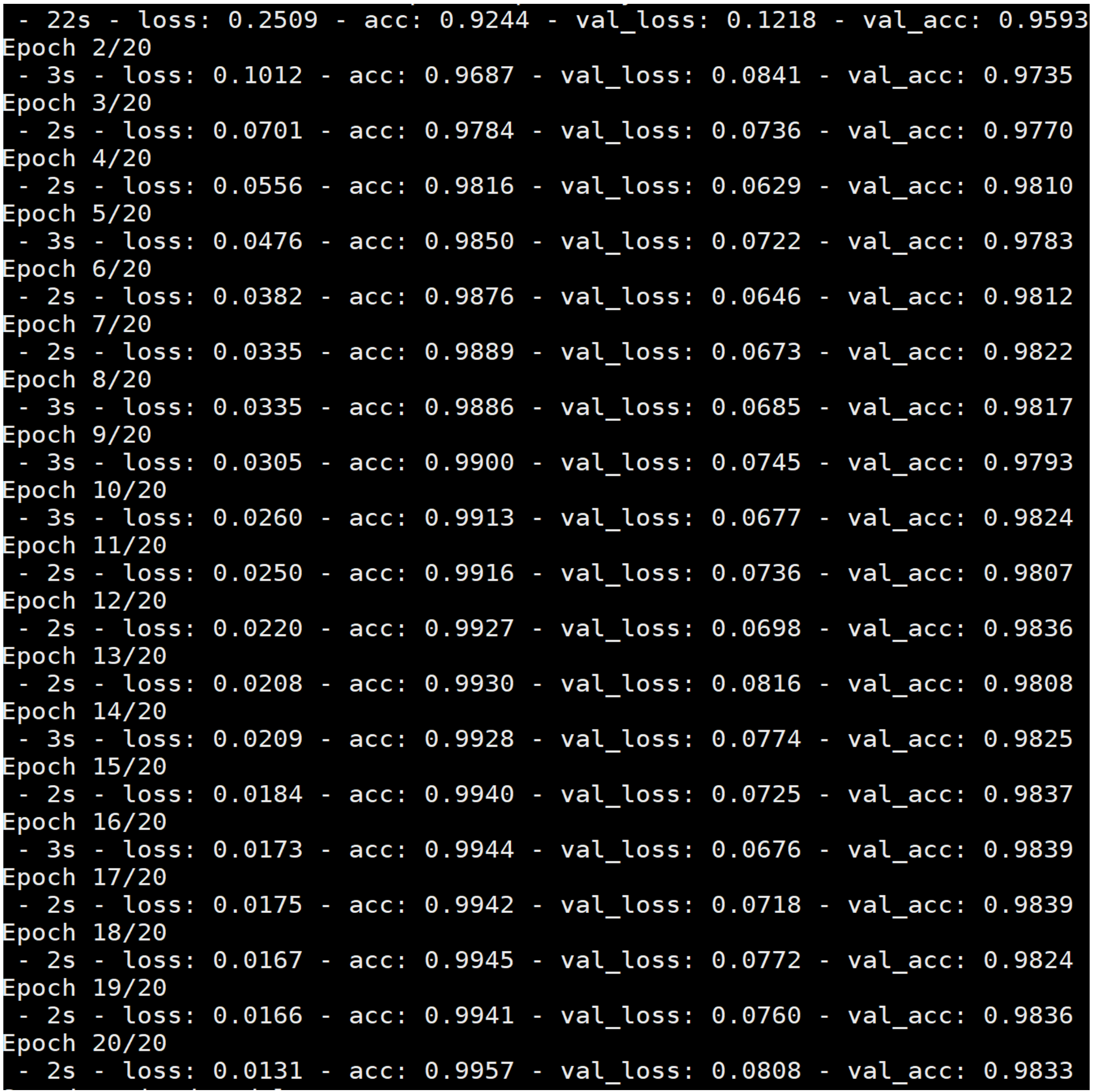
```
    model.add(Dropout(0.2))

    model.add(Dense(512))
    model.add(Activation('relu'))
    model.add(Dropout(0.2))

    model.add(Dense(10))
    model.add(Activation('softmax'))

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

    # training
    history = model.fit(X_train, Y_train,
              batch_size=128, epochs=20,
              verbose=2,
              validation_data=(X_test, Y_test))

    # saving the model
    save_dir = "./results"
    model.save(save_dir)
    print('Saved trained model')

```

Listing 2: Training

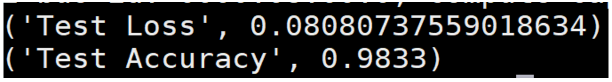
```
    mnist_model = load_model("results")
    loss_and_metrics = mnist_model.evaluate(X_test, Y_test, verbose=2)

    print("Test Loss", loss_and_metrics[0])
    print("Test Accuracy", loss_and_metrics[1])

```

Listing 3: Testing

6 Training and Testing Performance



```
 - 22s - loss: 0.2509 - acc: 0.9244 - val_loss: 0.1218 - val_acc: 0.9593
Epoch 2/20
 - 3s - loss: 0.1012 - acc: 0.9687 - val_loss: 0.0841 - val_acc: 0.9735
Epoch 3/20
 - 2s - loss: 0.0701 - acc: 0.9784 - val_loss: 0.0736 - val_acc: 0.9770
Epoch 4/20
 - 2s - loss: 0.0556 - acc: 0.9816 - val_loss: 0.0629 - val_acc: 0.9810
Epoch 5/20
 - 3s - loss: 0.0476 - acc: 0.9850 - val_loss: 0.0722 - val_acc: 0.9783
Epoch 6/20
 - 2s - loss: 0.0382 - acc: 0.9876 - val_loss: 0.0646 - val_acc: 0.9812
Epoch 7/20
 - 2s - loss: 0.0335 - acc: 0.9889 - val_loss: 0.0673 - val_acc: 0.9822
Epoch 8/20
 - 3s - loss: 0.0335 - acc: 0.9886 - val_loss: 0.0685 - val_acc: 0.9817
Epoch 9/20
 - 3s - loss: 0.0305 - acc: 0.9900 - val_loss: 0.0745 - val_acc: 0.9793
Epoch 10/20
 - 3s - loss: 0.0260 - acc: 0.9913 - val_loss: 0.0677 - val_acc: 0.9824
Epoch 11/20
 - 2s - loss: 0.0250 - acc: 0.9916 - val_loss: 0.0736 - val_acc: 0.9807
Epoch 12/20
 - 2s - loss: 0.0220 - acc: 0.9927 - val_loss: 0.0698 - val_acc: 0.9836
Epoch 13/20
 - 2s - loss: 0.0208 - acc: 0.9930 - val_loss: 0.0816 - val_acc: 0.9808
Epoch 14/20
 - 3s - loss: 0.0209 - acc: 0.9928 - val_loss: 0.0774 - val_acc: 0.9825
Epoch 15/20
 - 2s - loss: 0.0184 - acc: 0.9940 - val_loss: 0.0725 - val_acc: 0.9837
Epoch 16/20
 - 3s - loss: 0.0173 - acc: 0.9944 - val_loss: 0.0676 - val_acc: 0.9839
Epoch 17/20
 - 2s - loss: 0.0175 - acc: 0.9942 - val_loss: 0.0718 - val_acc: 0.9839
Epoch 18/20
 - 2s - loss: 0.0167 - acc: 0.9945 - val_loss: 0.0772 - val_acc: 0.9824
Epoch 19/20
 - 2s - loss: 0.0166 - acc: 0.9941 - val_loss: 0.0760 - val_acc: 0.9836
Epoch 20/20
 - 2s - loss: 0.0131 - acc: 0.9957 - val_loss: 0.0808 - val_acc: 0.9833
```



```
('Test Loss', 0.08080737559018634)
('Test Accuracy', 0.9833)
```

7 Instruction on how to test the trained DNN and how to use the GUI

7.1 Install Dependencies

- Python 3
- Tkinter
- Keras
- Tensorflow
- Numpy,Sicpy

7.2 Execution

Run classfier.py file and you can play around with the demo.

7.3 Code

Attached.

7.4 Video Link

Video for the GUI Demo is here.
(https://www.youtube.com/watch?v=0kXoEgENPTAfeature=youtu.be)