# Part 1

### To run the code:

#CREATE VIRTUAL ENVIRONMENT (do once)

**python3 -m venv env**

#ACTIVATE YOUR VIRTUAL ENVIRONMENT

**source env/bin/activate** OR source env/bin/activate.csh (this second likely your default shell system)

#INSTALL PACKAGES (do once)

**pip install numpy**

**pip install matplotlib**

**pip install Pillow**

**pip install pyscreenshot**

**pip install tensorflow**

#CHECK IF YOU ARE IN YOUR VIRTUAL ENV (should also see [env] in shell)

which python

#RUN CODE

**python3 MNIST.py**

### Explanation/Parameters:

I first started by changing the activation function in the last layer from sigmoid to softmax, so that the output could be better interpreted as an array of 10 prediction values between 0 and 1.

I also changed the optimizer function so that it had a momentum of 0.9. The default is 0. Having more momentum makes the model arrive at a correct prediction sooner.

The accuracy was still unacceptable, so I increased the number of layers, adding in 2 hidden dense layers; One layer with an output size of 500, and the other 300. Both have activation function Rectified Linear Unit. I read on a forum somewhere a good starting place for output sizes of dense layers was 2/3 of the input size (28\*28 \* 2/3 ~= 500, 500\*2/3 ~= 300). I also increased the epochs from 1 to 15.

This ended up working to arrive at a high accuracy, with 100% on the training data and 98.4% accuracy on the test data.

# Part 2:

### To run code:

Same as part 1, but replace **MNIST.py** with **notMNISTModel.py** (see above)

### Explanation/Parameters

My original model consists of 5 layers; 1 flatten layer (to reshape data from 28x28 to 1x784), and 4 Dense layers. The dense layers are of decreasing size outputs (600, 400, 200, 10), and the first 3 use the Rectified Linear Unit activation function.

The fourth dense layer uses softmax activation function, as this is best for outputting a prediction between 0 and 1 for each possible result. Since the output size of this layer is 10, it is producing a prediction for the input (image) to be classified into each possible category (A-J, 10 categories).

I used 30 epochs as this got to an acceptable level of accuracy (98% train, 94% test)

After drawing the letter H (in imageH.png), I tested the model on this image. It incorrectly labelled it as D.

#Changes:

Added 2 Convolution layers, and 2 max pooling layers, to train on smaller image patterns.

First convolution layer has output size of 16, second has output of 32.

## 

## Changed lines in predict:

Added

```

model = tf.keras.models.load\_model(sys.argv[2])

```

To get model loading to work.

\

Added

```

prediction = model.predict(img)

```

To make a prediction on the **\*\*img\*\***

\

Added

```

predicted\_label = np.argmax(prediction)

```

To get the prediction index