Neural Network with MNIST

October 25, 2017

1 Neural Classifier

This notebook shows an expample of a neural network classifier applied to MNIST data. This notebook was written after taking the online course cs231 offered at Stanford University and its objective is test ones familiarity with both **Tensor FLow** and the **MNIST** database.

1.1 MNIST

The following section shows how to import, manipulate and show elements of the MNIST data base. All the imports are done using **python-mnist 0.3**. For more details on this library please check: *https://pypi.python.org/pypi/python-mnist/*

```
In [1]: #Imports and a little bit of setup
        from __future__ import print_function
        import numpy as np
        import matplotlib.pyplot as plt
        from mnist import MNIST
        import tensorflow as tf
        import math

        #A little bit of matplotlib magic so that the images can be showed on the IPython note
        %matplotlib inline
        plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
        plt.rcParams['image.interpolation'] = 'nearest'
        plt.rcParams['image.cmap'] = 'gray'

        #Small function that helps displays images
        def imshow_noax(img, normalize=True):
            """ Tiny helper to show images as uint8 and remove axis labels """
            if normalize:
                img_max, img_min = np.max(img), np.min(img)
                img = 255.0 * (img - img_min) / (img_max - img_min)
            plt.imshow(img.astype('uint8'))
            plt.gca().axis('off')
```

1.1.1 MNIST data

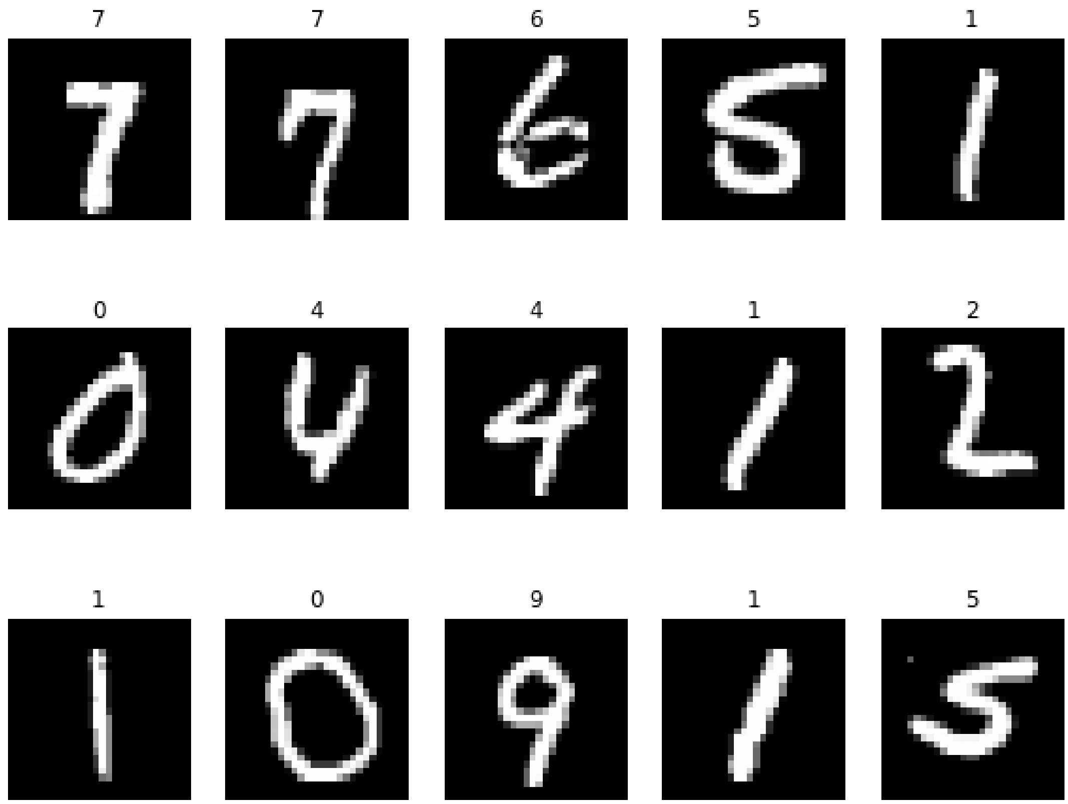
The following cell imports the data. The data base consists of 60,000 images for training and 10,000 for testing. Each image corresponds to a list of 784 of values between 0 and 255, representing the gray value of the pixel (where 0 equals black and 255 equals white). In turn, each image is of size 28 by 28 pixels in black and white.

```
In [3]: #Imports the data and converts it to numpy arrays
        mndata = MNIST('../python-mnist/data')
        X_train, y_train = mndata.load_training()
        X_test, y_test = mndata.load_testing()

        #converts to numpy array
        X_train = np.array(X_train)
        X_test = np.array(X_test)
        y_train = np.array(y_train)
        y_test = np.array(y_test)

        #Train size and dimension size
        N, D = X_train.shape
```

1.1.2 Some Examples

The next cell shows some examples from the data base of digits

```
In [135]: n_row = 3
          n_col = 5
          for i in range(n_row*n_col):
              ran = np.random.randint(N)
              plt.subplot(n_row, n_col, i+1)
              imshow_noax(X_train[ran].reshape(28,28), normalize=False)
              plt.title(str(y_train[ran]))
          plt.show()
```



1.2 Neural Netwok

We will now proceed with the creation, trianing and testing of a neural network calssifier. To do this we will use exclusively the **Tensor Flow** library. For more details on this library please go to: *https://www.tensorflow.org/*.

1.2.1 Architecture

The architecture of this neural netwok will be as follows:

- 4 Hidden layers of a 100 neurons each, with Bathnorm, Relu activation and dropout. Specifically, each of this four layers will have:
  - Affine layer: Linear classifier of dimension D x 100, with biased.
  - BatchNorm layer: Layer that normalizaes the batch input
  - Dropout layer: dropout layer that drops 50% of its values
  - Activation layer: Activation layer using the Relu function.
- Classification layer that outputs the 10 categories of the MNIST data
- Loss function will be SoftMax

We will optimize using RMSprop gradient descent (proposed by Geoff Hinton in Lecture 6e of his Coursera Class: *http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf*)

The following cell delcares the architecture:

In [136]:
```
# Clears old variables (just in case)
tf.reset_default_graph()

# Define our inputs
#The dimension None lets us train by batches of data
X = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.int64, [None])
is_training = tf.placeholder(tf.bool)

# define model
def mnist_model(X, y, is_training,hidden_layers = 4, num_neurons = 100):
    """
    A function that returns the inital node of the neural network graph.

    Input:
    - x: Input data of shape (N, D)
    - y: Input data labels (N,)
    - is_training: a boolean indicating if we are testing or trianing the graph
    - hidden_layers: The number of hidden layers the net will have (must be > 0)
    - num_neurons: the number of neurons the layers will have

    Returns:
    The leading tensor flow node
    """
    prob = 0.5
    keep_prob = tf.constant(prob)
    N, D = X.shape
    C = 10
    #------------------------------
    #---------First Layer----------
    #------------------------------
    #Affine layer
    with tf.name_scope("HiddenLayer0"):
        W = tf.get_variable("W0", shape=[D, num_neurons])
        b = tf.get_variable("b0", shape=[num_neurons])
        current_layer = tf.matmul(X, W) + b

    #batchNorm
    with tf.name_scope("BatchNorm0"):
        current_layer = tf.layers.batch_normalization(current_layer,
                                                      axis=1,
                                                      training=is_training)
```

```
    #Dropout layer
    with tf.name_scope("Dropout0"):
        current_layer = tf.nn.dropout(current_layer, keep_prob = keep_prob )

    #Relu activation layer
    with tf.name_scope("ReluLayer0"):
        current_layer = tf.nn.relu(current_layer)



    #------------------------------
    #--------Hidden Layer----------
    #------------------------------
    for i in range(hidden_layers):
        l = i+1
        #Affine layer
        with tf.name_scope("HiddenLayer" + str(l)):
            W = tf.get_variable("W" + str(l), shape=[num_neurons, num_neurons])
            b = tf.get_variable("b" + str(l), shape=[num_neurons])
            current_layer = tf.matmul(current_layer, W) + b

        #batchNorm
        with tf.name_scope("BatchNorm" + str(l)):
            current_layer = tf.layers.batch_normalization(current_layer,
                                                          axis=1,
                                                          training=is_training)

        #Dropout layer
        with tf.name_scope("Dropout" + str(l)):
            current_layer = tf.nn.dropout(current_layer, keep_prob = keep_prob )

        #Relu activation layer
        with tf.name_scope("ReluLayer" + str(l)):
            current_layer = tf.nn.relu(current_layer)


    #------------------------------
    #--------Last Layer----------
    #------------------------------
    with tf.name_scope("LastLayer"):
        W = tf.get_variable("W" + str(hidden_layers + 1), shape=[num_neurons, C])
        b = tf.get_variable("b" + str(hidden_layers + 1), shape=[C])
        current_layer = tf.matmul(current_layer, W) + b

    return current_layer


#Our prediction labels
```

```
y_out = mnist_model(X,y,is_training)

#Loss function
mean_loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=tf.one_hot(y, 10),
                                                                logits=y_out))

#Declares the optimizaer
optimizer = tf.train.RMSPropOptimizer(1e-3)

# batch normalization in tensorflow requires this extra dependency
extra_update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(extra_update_ops):
    train_step = optimizer.minimize(mean_loss)
```

1.2.2 Training the Model

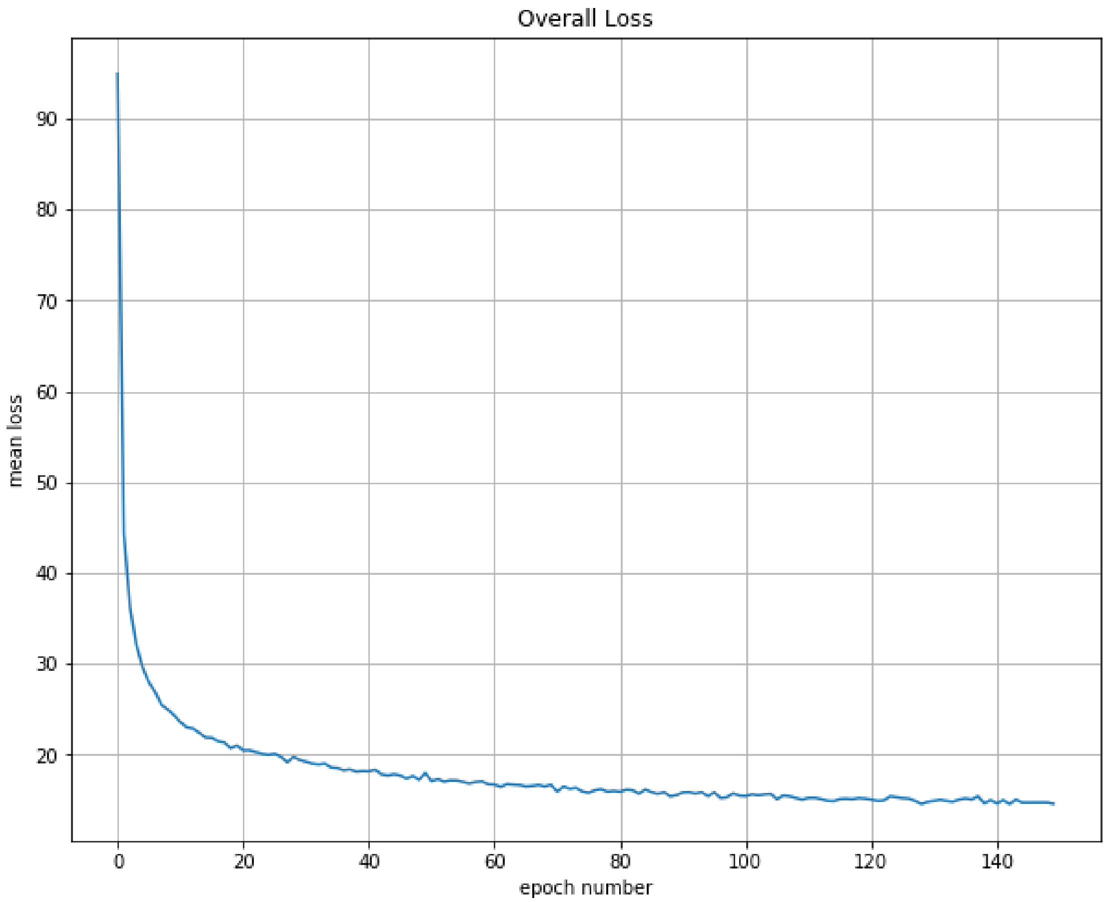
The following cells provides a method that trains the model and outputs its progress, and trains the previous model with 150 batch iterations (Epochs)

In [139]:
```
#Function that runs the model and outputs the loss function plot

def run_model(session, predict, loss_val, Xd, yd,
              epochs=1, batch_size=64, print_every=100,
              training=None, plot_losses=False, print_every_epoch = 1):

    # have tensorflow compute accuracy
    correct_prediction = tf.equal(tf.argmax(predict,1), y)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    # shuffle indicies
    train_indicies = np.arange(Xd.shape[0])
    np.random.shuffle(train_indicies)

    training_now = training is not None

    # setting up variables we want to compute (and optimizing)
    # if we have a training function, add that to things we compute
    variables = [mean_loss,correct_prediction,accuracy]
    if training_now:
        variables[-1] = training

    global_losses = []
    # counter
    iter_cnt = 0
    for e in range(epochs):
        # keep track of losses and accuracy
        losses = []
```

```
        correct = 0
        # make sure we iterate over the dataset once
        for i in range(int(math.ceil(Xd.shape[0]/batch_size))):
            # generate indicies for the batch
            start_idx = (i*batch_size)%Xd.shape[0]
            idx = train_indicies[start_idx:start_idx+batch_size]

            # create a feed dictionary for this batch
            feed_dict = {X: Xd[idx,:],
                         y: yd[idx],
                         is_training: training_now }
            # get batch size
            actual_batch_size = yd[idx].shape[0]

            # have tensorflow compute loss and correct predictions
            # and (if given) perform a training step
            loss, corr, _ = session.run(variables,feed_dict=feed_dict)

            # aggregate performance stats
            losses.append(loss*actual_batch_size)
            correct += np.sum(corr)

            # print every now and then
            if training_now and ((iter_cnt +1) % print_every) == 0:
                print("""Iteration {0}: with minibatch training loss = {1:.3g}
                      and accuracy of {2:.2g}"""\
                      .format(iter_cnt,loss,np.sum(corr)/actual_batch_size))
            iter_cnt += 1

        total_correct = correct/Xd.shape[0]
        total_loss = np.sum(losses)/Xd.shape[0]

        if (e  % print_every_epoch) == 0:
            print("Epoch {2}, Overall loss = {0:.3g} and accuracy of {1:.3g}"\
                  .format(total_loss,total_correct,e))

        #adds to global losses for plotting
        global_losses = global_losses + [np.mean(losses)]

    if plot_losses:
        plt.plot(global_losses)
        plt.grid(True)
        plt.title('Overall Loss')
        plt.xlabel('epoch number')
        plt.ylabel('mean loss')
        plt.show()

    return total_loss,total_correct
```

```
In [140]: #Runs the model and outputs its progress

          #Starts the session
          sess = tf.Session()
          sess.run(tf.global_variables_initializer())

          #Start Run
          result_val = run_model(session = sess,
                                 predict = y_out,
                                 loss_val = mean_loss,
                                 Xd = X_train,
                                 yd = y_train,
                                 epochs = 150,
                                 batch_size = 60,
                                 print_every = math.inf,
                                 training = train_step,
                                 plot_losses = True,
                                 print_every_epoch = 10)
```

```
Iteration 0: with minibatch training loss = 3.12 and accuracy of 0.12
Epoch 9, Overall loss = 0.406 and accuracy of 0.9
Epoch 19, Overall loss = 0.35 and accuracy of 0.915
Epoch 29, Overall loss = 0.323 and accuracy of 0.921
Epoch 39, Overall loss = 0.303 and accuracy of 0.927
Epoch 49, Overall loss = 0.3 and accuracy of 0.928
Epoch 59, Overall loss = 0.279 and accuracy of 0.933
Epoch 69, Overall loss = 0.278 and accuracy of 0.932
Epoch 79, Overall loss = 0.266 and accuracy of 0.935
Epoch 89, Overall loss = 0.259 and accuracy of 0.937
Epoch 99, Overall loss = 0.258 and accuracy of 0.937
Epoch 109, Overall loss = 0.251 and accuracy of 0.939
Epoch 119, Overall loss = 0.253 and accuracy of 0.939
Epoch 129, Overall loss = 0.247 and accuracy of 0.94
Epoch 139, Overall loss = 0.25 and accuracy of 0.939
Epoch 149, Overall loss = 0.243 and accuracy of 0.94
```



1.2.3 Testing

Now that we have trained the model, we test it to see what we get.

```
In [141]: result_test = run_model(sess,y_out,mean_loss,X_test,y_test,1,64, print_every_epoch =

          print("Model acuracy over the test set: {0:.2f}%".format(result_test[1]*100))
```

```
Model acuracy over the test set: 93.87%
```

The model was able to correcly predict the label of the given image with an accuracy of **93.87%**