Neural Network

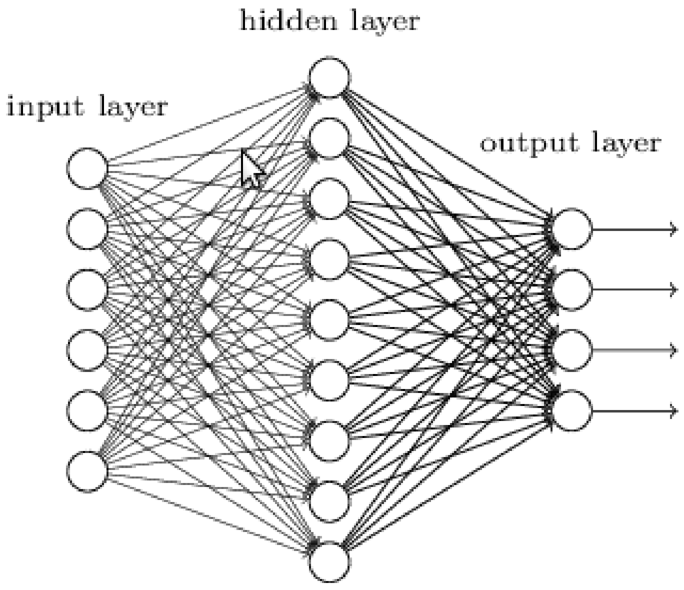
March 29, 2018

** Xiangyi Cheng (xxc273)**

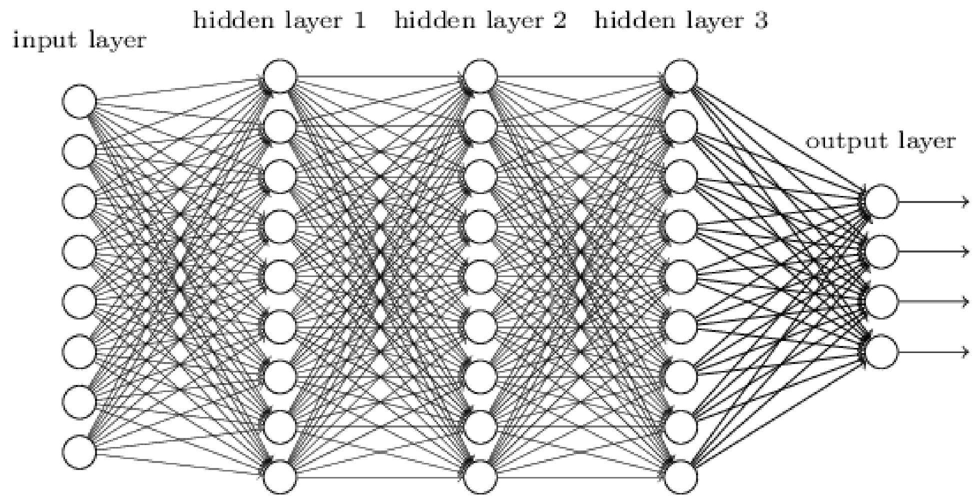
1 Concept and Background

In this assignment, a baseline and a deep neural network are required to be constructed. A dataset needs to be trained based on this neural network. The performance trained from the baseline and the deep neural network should be compared and discussed.

A neural network is an artificial neural network with one or more hidden layers between the input and output layers which is inspired by biological neural networks that constitude animal brains. For the simplest case that only one layer exists (the diagram is shown below). Input is a set of observations, each an N-dimension vector. Weights are assigned to generate the hidden layer from the input layer. After obtaining the hidden layer, new weights are calculated to get the output layer. Specifically saying, the hidden layer is made of nodes. A node combines input from the data with a set of weights which called net input function. Then the result is passed through a node called activation function to determine if and how the signal progresses affect the output. In this case, weights are fully connected which means each node owns its weights from an input.



Extend this basic idea to a deep neural network which includes multiple layers. Once the first layer is obtained, the weights are assigned again to generate a new layer. After several times, the network consists of multiple layers which causes more accurate ultimate output. The diagram below illuminates a three-hidden-layer neural network.



The weights mentioned above are the key to get a good model with accurate predictions. Proper weights are obtained by "back progagation algorithm" which was imbedded into many packages. And the common method of optimization algorithm is "gradient descent". Applying this algorithm, optimal weights are chosen to minimize the errors.

The applications of deep neural network are broad and play an important role in several fields such as object detection and recognition, automatic speech recognition, visual art processing, etc.. The packages developed based on neural network are mature and various. In this assignment, TensorFlow is used to construct a deep neural network and train it on mnist dataset which is a large database of handwritten digits included 60000 training images and 10000 test images. The accuracy will be obtained after applying the training.

2 Implement

TensorFlow is an open-source software library used for machine learning applications such as neural networks. Basically, we will use TensorFlow to train the mnist database based on the models we build. The accuracy will be computated and printed out. To achieve this goal, TensorFlow and mnist database should be imported first.

```
In [2]: import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data
```

Load the mnist and extract the data and label seperately. Class number is 10 due to 10 digit numbers.

```
In [3]: mnist=input_data.read_data_sets('/tmp/data/',one_hot=True)


        mnist_data=tf.placeholder('float',[None,784])
        label=tf.placeholder('float')

        # 10 different digit numbers: 0-9
        n_classes=10
        n_each_group=100
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

2.1 Baseline Construction

A model include only one hidden layer is constructed. The compulational idea is:

$$HiddenLayer = InputData * weight + bias$$

$$OutputLayer = HiddenLayer * weight + bias$$

The nodes in hidden layer is set to 500 which could offer us a good result without consuming heavy computations. The input value is 784 since each image in mnist is in 28*28 dimenions. An activation function is called after obtaining the hidden layer.

```
In [4]: def baseline_neural_network(data):
                # input_data * weight + bias
                # the
                n_hidden1=500

                # each image has 784 pixels which is calculated by 28*28.
                hidden_layer1={'weights':tf.Variable(tf.random_normal([784,n_hidden1])),
                           'biases':tf.Variable(tf.random_normal([n_hidden1]))}

                output_layer={'weights':tf.Variable(tf.random_normal([n_hidden1,n_classes])),
                          'biases':tf.Variable(tf.random_normal([n_classes]))}

                layer1 = tf.add(tf.matmul(data,hidden_layer1['weights']),
                            hidden_layer1['biases'])
                act_layer1 = tf.nn.relu(layer1)

                model = tf.matmul(act_layer1,output_layer['weights'])+
            output_layer['biases']

                return model
```

The training function is defined as below. The baseline neural network model we built above is passed into the training function. In the code, tf.train.AdamOptimizer().minimize() is applied to optimize the weights by reducing the errors. To view the training process, 10 epoches should be printed out to show the loss.

```
In [5]: def train_baseline(models):

                prep_model=baseline_neural_network(models)
```

```
        cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                (logits=prep_model,labels=label))
        opitimizer=tf.train.AdamOptimizer().minimize(cost)

        step_epochs=10

        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())

            for epoch in range(step_epochs):
                Loss=0
                for _ in range(int(mnist.train.num_examples/n_each_group)):
                    epoch_x,epoch_y = mnist.train.next_batch(n_each_group)
                    _,loss = sess.run([opitimizer,cost],feed_dict=
                      {models:epoch_x,label:epoch_y})
                    Loss+=loss
                print 'Step',epoch,'is completed out of',step_epochs,
          '. Loss is ',Loss

            correct=tf.equal(tf.argmax(prep_model,1),tf.argmax(label,1))
            accuracy=tf.reduce_mean(tf.cast(correct,'float'))
            print 'baseline accuracy:',
        accuracy.eval({models:mnist.test.images,label:mnist.test.labels})
```

It is time to pass our mnist data into the defined trainning function by simply calling the function.

```
In [6]: train_baseline(mnist_data)

Step 0 is completed out of 10 . Loss is  13619.7621614
Step 1 is completed out of 10 . Loss is  3424.94012737
Step 2 is completed out of 10 . Loss is  2286.27826652
Step 3 is completed out of 10 . Loss is  1654.57457437
Step 4 is completed out of 10 . Loss is  1238.81159588
Step 5 is completed out of 10 . Loss is  946.172238963
Step 6 is completed out of 10 . Loss is  735.818641031
Step 7 is completed out of 10 . Loss is  568.784084868
Step 8 is completed out of 10 . Loss is  452.778618537
Step 9 is completed out of 10 . Loss is  354.861006843
baseline accuracy: 0.948
```

The final result is shown above that the accuracy is 0.948 based on one-hidden-layer-model.

2.2 Deep Neural Network Construction

To improve the training result, more hidden layers should be added to construct a more complex and deeper neural network. After trained by multiple layers, the accuracy should increase somehow. To test this concept, a eight-layer-neural-network is built.

The nodes of each hidden layer are initialized to 500, 1000, 1500, 2000, 2500, 2500, 2500, 2500, respectively. The whole building is similar to baseline construction but adding more hidden layers.

```
In [7]: def deep_neural_network(data):
            # input_data * weight + bias
            n_hidden1=500
            n_hidden2=1000
            n_hidden3=1500
            n_hidden4=2000
            n_hidden5=2500
            n_hidden6=2500
            n_hidden7=2500
            n_hidden8=2500

            hidden_layer1={'weights':tf.Variable(tf.random_normal([784,n_hidden1])),
                       'biases':tf.Variable(tf.random_normal([n_hidden1]))}
            hidden_layer2={'weights':tf.Variable(tf.random_normal([n_hidden1,n_hidden2])),
                       'biases':tf.Variable(tf.random_normal([n_hidden2]))}
            hidden_layer3={'weights':tf.Variable(tf.random_normal([n_hidden2,n_hidden3])),
                       'biases':tf.Variable(tf.random_normal([n_hidden3]))}
            hidden_layer4={'weights':tf.Variable(tf.random_normal([n_hidden3,n_hidden4])),
                       'biases':tf.Variable(tf.random_normal([n_hidden4]))}
            hidden_layer5={'weights':tf.Variable(tf.random_normal([n_hidden4,n_hidden5])),
                       'biases':tf.Variable(tf.random_normal([n_hidden5]))}
            hidden_layer6={'weights':tf.Variable(tf.random_normal([n_hidden5,n_hidden6])),
                       'biases':tf.Variable(tf.random_normal([n_hidden6]))}
            hidden_layer7={'weights':tf.Variable(tf.random_normal([n_hidden6,n_hidden7])),
                       'biases':tf.Variable(tf.random_normal([n_hidden7]))}
            hidden_layer8={'weights':tf.Variable(tf.random_normal([n_hidden7,n_hidden8])),
                       'biases':tf.Variable(tf.random_normal([n_hidden8]))}


            output_layer1={'weights':tf.Variable(tf.random_normal([n_hidden1,n_classes])),
                       'biases':tf.Variable(tf.random_normal([n_classes]))}

            output_layer2={'weights':tf.Variable(tf.random_normal([n_hidden2,n_classes])),
                       'biases':tf.Variable(tf.random_normal([n_classes]))}

            output_layer8={'weights':tf.Variable(tf.random_normal([n_hidden8,n_classes])),
                       'biases':tf.Variable(tf.random_normal([n_classes]))}


            layer1 = tf.add(tf.matmul(data,hidden_layer1['weights']),
                        hidden_layer1['biases'])
            act_layer1 = tf.nn.relu(layer1)

            layer2 = tf.add(tf.matmul(act_layer1,hidden_layer2['weights']),
                hidden_layer2['biases'])
        act_layer2 = tf.nn.relu(layer2)

        layer3 = tf.add(tf.matmul(act_layer2,hidden_layer3['weights']),
                hidden_layer3['biases'])
        act_layer3 = tf.nn.relu(layer3)

        layer4 = tf.add(tf.matmul(act_layer3,hidden_layer4['weights']),
                hidden_layer4['biases'])
        act_layer4 = tf.nn.relu(layer4)

        layer5 = tf.add(tf.matmul(act_layer4,hidden_layer5['weights']),
                hidden_layer5['biases'])
        act_layer5 = tf.nn.relu(layer5)

        layer6 = tf.add(tf.matmul(act_layer5,hidden_layer6['weights']),
                hidden_layer6['biases'])
        act_layer6 = tf.nn.relu(layer6)

        layer7 = tf.add(tf.matmul(act_layer6,hidden_layer7['weights']),
                hidden_layer7['biases'])
        act_layer7 = tf.nn.relu(layer7)

        layer8 = tf.add(tf.matmul(act_layer7,hidden_layer8['weights']),
                hidden_layer8['biases'])
        act_layer8 = tf.nn.relu(layer8)


        model_1 = tf.matmul(act_layer1,output_layer1['weights'])+
    output_layer1['biases']

        model_2 = tf.matmul(act_layer2,output_layer2['weights'])+
    output_layer2['biases']

        model_8 = tf.matmul(act_layer8,output_layer8['weights'])+
    output_layer8['biases']


        return model_1,model_2,model_8
```

To have a better comparison, three models with 1 hidden layer, 2 hidden layers and 8 hidden layers are extracted. Next step is to train the deep neural network. Although three models are trained seperately, they share the same weights and bias assigned in the deep_neural_network(data) function. So the results are comparable and meaningful.

```
In [8]: def train_neural_network(models):

            prep_model1,prep_model2,prep_model8=deep_neural_network(models)
```

```
cost1=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                 (logits=prep_model1, labels=label))
opitimizer1=tf.train.AdamOptimizer().minimize(cost1)

cost2=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                 (logits=prep_model2, labels=label))
opitimizer2=tf.train.AdamOptimizer().minimize(cost2)

cost8=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                 (logits=prep_model8, labels=label))
opitimizer8=tf.train.AdamOptimizer().minimize(cost8)

step_epochs=10

with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        # baseline
        for epoch in range(step_epochs):
                Loss=0
                for _ in range(int(mnist.train.num_examples/n_each_group)):
                        epoch_x,epoch_y = mnist.train.next_batch(n_each_group)
                        _,loss = sess.run([opitimizer1,cost1],
                          feed_dict={models:epoch_x,label:epoch_y})
                        Loss+=loss

        correct1=tf.equal(tf.argmax(prep_model1,1),tf.argmax(label,1))
        accuracy1=tf.reduce_mean(tf.cast(correct1,'float'))
        print 'baseline accuracy:',
accuracy1.eval({models:mnist.test.images,label:mnist.test.labels})


        # 2 layes
        for epoch in range(step_epochs):
                Loss=0
                for _ in range(int(mnist.train.num_examples/n_each_group)):
                        epoch_x,epoch_y = mnist.train.next_batch(n_each_group)
                        _,loss = sess.run([opitimizer2,cost2],
                          feed_dict={models:epoch_x,label:epoch_y})
                        Loss+=loss

        correct2=tf.equal(tf.argmax(prep_model2,1),tf.argmax(label,1))
        accuracy2=tf.reduce_mean(tf.cast(correct2,'float'))
        print '2 layers accuracy:',
accuracy2.eval({models:mnist.test.images,label:mnist.test.labels})
```

```
            # 8 layers
            for epoch in range(step_epochs):
                    Loss=0
                    for _ in range(int(mnist.train.num_examples/n_each_group)):
                            epoch_x,epoch_y = mnist.train.next_batch(n_each_group)
                            _,loss = sess.run([opitimizer8,cost8],
                              feed_dict={models:epoch_x,label:epoch_y})
                            Loss+=loss
                    print 'Step',epoch,'is completed out of',step_epochs,
        '. Loss is ',Loss

            correct8=tf.equal(tf.argmax(prep_model8,1),tf.argmax(label,1))
            accuracy8=tf.reduce_mean(tf.cast(correct8,'float'))
            print '8 layers accuracy:',
    accuracy8.eval({models:mnist.test.images,label:mnist.test.labels})
```

As the same as training the baseline, we pass our mnist data into the defined trainning function by simply calling the function.

```
In [9]: train_neural_network(mnist_data)

baseline accuracy: 0.9424
2 layers accuracy: 0.9564
Step 0 is completed out of 10 . Loss is  7.08814258316e+13
Step 1 is completed out of 10 . Loss is  1.30684436299e+13
Step 2 is completed out of 10 . Loss is  6.75410713926e+12
Step 3 is completed out of 10 . Loss is  4.75541507501e+12
Step 4 is completed out of 10 . Loss is  4.30675179694e+12
Step 5 is completed out of 10 . Loss is  3.19045525925e+12
Step 6 is completed out of 10 . Loss is  2.64274842516e+12
Step 7 is completed out of 10 . Loss is  2.78216182691e+12
Step 8 is completed out of 10 . Loss is  2.39636090822e+12
Step 9 is completed out of 10 . Loss is  1.70913980897e+12
8 layers accuracy: 0.9637
```

Based on the outcome computed above, the accuracy is increasing with more hidden layers although less than 0.01 accuracy improved from single layer to 2-layer model. However, the accuracy is improve from 0.9424 to 0.9637 when 8 layers are applied.

3 Exploration

The models we built above are all based on the traditional neural network with fully connected weights. Besides the traditional approach, improvements are achieved by modifying the models in different ways. The relatively popular ones are Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). As the exploration, I used CNN to construct a network and trained it with mnist dataset.

Similar to the traditional neural network, a CNN consists of an input and an output layer, as well as multiple hidden layers. However, the hidden layers of a CNN typically includes convolutional layers, pooling layers, fully connected layers and normalization layers.

To achieve the goal to constuct a CNN model, TensorFlow is still used due to its useful imbedded functions. A function called convolutional_neural_network(data) is defined by constructing a 2-hidden-layer-CNN. Firstly, weights and bias are specified. Then convolutions are done to the data and each hidden layer. Next, pool the values by picking the maximum among them. Lastly, activate the function to get the model. The basic computational idea is still:

$$HiddenLayer = InputData * weight + bias$$

$$OutputLayer = HiddenLayer * weight + bias$$

```
In [10]: def convolutional_neural_network(data):

                 weights={'W_conv1':tf.Variable(tf.random_normal([5,5,1,32])),
                      'W_conv2':tf.Variable(tf.random_normal([5,5,32,64])),
                          'W_fc':tf.Variable(tf.random_normal([7*7*64,1024])),
                      'out':tf.Variable(tf.random_normal([1024,n_classes]))}
                 biases={'b_conv1':tf.Variable(tf.random_normal([32])),
                     'b_conv2':tf.Variable(tf.random_normal([64])),
                          'b_fc':tf.Variable(tf.random_normal([1024])),
                     'out':tf.Variable(tf.random_normal([n_classes]))}

                 data=tf.reshape(data,shape=[-1,28,28,1])

                 conv1=tf.nn.conv2d(data,weights['W_conv1'],
                                    strides=[1,1,1,1],padding='SAME')
                 conv1=tf.nn.max_pool(conv1,ksize=[1,2,2,1],
                                      strides=[1,2,2,1],padding='SAME')

                 conv2=tf.nn.conv2d(conv1,weights['W_conv2'],
                                    strides=[1,1,1,1],padding='SAME')
                 conv2=tf.nn.max_pool(conv2,ksize=[1,2,2,1],
                                      strides=[1,2,2,1],padding='SAME')

                 fc=tf.reshape(conv2,[-1,7*7*64])
                 fc=tf.nn.relu(tf.matmul(fc,weights['W_fc'])+biases['b_fc'])

                 model_CNN=tf.matmul(fc,weights['out'])+biases['out']

                 return model_CNN
```

After getting the CNN model, it is required to train the data into with CNN. The whole procedure is similar to the traditional neural network except for calling the CNN model.

```
In [11]: def train_CNN(models):

                 prep_model=convolutional_neural_network(models)
```

```
cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
                (logits=prep_model, labels=label))
opitimizer=tf.train.AdamOptimizer().minimize(cost)

step_epochs=10

with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(step_epochs):
                Loss=0
                for _ in range(int(mnist.train.num_examples/n_each_group)):
                        epoch_x,epoch_y = mnist.train.next_batch(n_each_group)
                        _,loss = sess.run([opitimizer,cost],
                          feed_dict={models:epoch_x,label:epoch_y})
                        Loss+=loss
                print 'Step',epoch,'is completed out of',step_epochs,
    '. Loss is ',Loss

        correct=tf.equal(tf.argmax(prep_model,1),tf.argmax(label,1))
        accuracy=tf.reduce_mean(tf.cast(correct,'float'))
        print 'CNN accuracy:',
accuracy.eval({models:mnist.test.images,label:mnist.test.labels})
```

Train the mnist dataset with the CNN model we built.

```
In [13]: train_CNN(mnist_data)

Step 0 is completed out of 10 . Loss is  2218924.89516
Step 1 is completed out of 10 . Loss is  487783.264467
Step 2 is completed out of 10 . Loss is  295234.748389
Step 3 is completed out of 10 . Loss is  202000.191294
Step 4 is completed out of 10 . Loss is  125114.0923
Step 5 is completed out of 10 . Loss is  99300.5750742
Step 6 is completed out of 10 . Loss is  72220.6646907
Step 7 is completed out of 10 . Loss is  54282.5147519
Step 8 is completed out of 10 . Loss is  45919.4728761
Step 9 is completed out of 10 . Loss is  46508.69596
CNN accuracy: 0.9657
```

4 Conclusion and Discussion

Based on the assignment requirements, a neural network with single hidden layer called baseline is constructed by TensorFlow. Although I made two baseline models, the discussion will be based on the second one whose accuracy is 0.9424 after passing the mnist datasat into the training function simply because it shares the same model with 2-layer and 8-layer model. The same model ensures the comparability is robust and and the comparison is meaningful. After adding one more hidden layer to the baseline, the accuracy is improved to 0.9564. The procedure was repeated until 8 hidden layers are obtained. The accuracy is 0.9637 with the 8-layer model. We could say that the accuracy is increasing with more hidden layers. However, more hidden layers also need more computations to support. It is critical for us to determine how many layers we will use by balancing the ultimate outcome and the computational costs.

As an exploration, Convolutional Neural Network was applied to mnist using TensorFlow as well. To save the computational time, a 2-layer CNN model is constructed. The accuracy is surprisingly high, 0.9657. Compared this result to the 2-layer traditional neural network model outcome whose accuracy is 0.9564, the CNN is more efficient and more accurate than the result of 8-layer one. But the computational time of CNN is much longer than the traditional one since more steps such as convolution, pooling are needed when building the model.

In general, the performances of neural network are outstanding. We could tell this based on the accuracy 0.9424 from the baseline within several seconds. And there are still spaces to improve the outcomes such as using multiple hidden layers and other neural network models, for instance, CNN and RNN.

5 Reference

https://deeplearning4j.org/neuralnet-overview

https://en.wikipedia.org/wiki/Deep_learning

https://www.deeplearningtrack.com/single-post/2017/07/09/Introduction-to-NEURAL-NETWORKS-Advantages-and-Applications

https://en.wikipedia.org/wiki/Convolutional_neural_network