HW9 - Neural networks

April 25, 2019

1 Data-X Spring 2019: Homework 9

1.1 Student name: Tirth Patel

1.2 Student id: 3034227694

Q1. You have now seen how Neural networks work. You have also seen how to create and visualize neural networks using Tensorflow and Tensorboard. In this Question, you will be working on Neural networks. You will be using MNIST data (labelled images of digits) that we discussed in the class to create vanilla dense Neural network model using **tensorflow** (version 2.x is preferred, you can use 1.x as well, **Limit the use of Keras** for solving this question) with the following characteristics: - Input layer size of 784 (Since each image is 28 * 28) - Three hidden layers of 300, 200 , 100 - Output layer of 10 (Since 0 - 9 digits) - Use stochastic gradient descent - Any other requirements can be your choice

Note that you have to define own functions for calculating loss function, optimizer to feed into the neural network. **Plot your neural network graph (using tensorboard) and the plot of performance results (Training and Validation accuracies and loss) for every epoch**

Note: You can access MNIST data from **keras.datasets** Link or any standard available MNIST datasource (http://yann.lecun.com/exdb/mnist/)

In [1]:
```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /srv/app/venv/lib/python3.6/site-packages
Requirement already satisfied: protobuf>=3.3.0 in /srv/app/venv/lib/python3.6/site-packages (f
Requirement already satisfied: wheel>=0.26 in /srv/app/venv/lib/python3.6/site-packages (from
Requirement already satisfied: tensorflow-tensorboard<0.2.0,>=0.1.0 in /srv/app/venv/lib/pytho
Requirement already satisfied: six>=1.10.0 in /srv/app/venv/lib/python3.6/site-packages (from
Requirement already satisfied: numpy>=1.11.0 in /srv/app/venv/lib/python3.6/site-packages (fro
Requirement already satisfied: setuptools in /srv/app/venv/lib/python3.6/site-packages (from p
Requirement already satisfied: html5lib==0.9999999 in /srv/app/venv/lib/python3.6/site-package
Requirement already satisfied: werkzeug>=0.11.10 in /srv/app/venv/lib/python3.6/site-packages
Requirement already satisfied: markdown>=2.6.8 in /srv/app/venv/lib/python3.6/site-packages (f
Requirement already satisfied: bleach==1.5.0 in /srv/app/venv/lib/python3.6/site-packages (fro
```

In [3]:
```
import tensorflow as tf
import keras
```

In [1]:
```
from keras.datasets import mnist
```

```
/srv/app/venv/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning: Conversion of the
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
In [2]: (X_train, y_train), (X_test, y_test) = mnist.load_data()
        #(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

```
In [4]: 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
        print("Training matrix shape", X_train.shape)
        print("Testing matrix shape", X_test.shape)
```

```
Training matrix shape (60000, 784)
Testing matrix shape (10000, 784)
```

```
In [ ]:
```

```
In [5]: from keras.utils import np_utils
        Y_train = np_utils.to_categorical(y_train,10)
        Y_test = np_utils.to_categorical(y_test, 10)
```

```
In [6]: import numpy as np
```

```
In [7]: import tensorflow as tf
        tf.reset_default_graph()
```

```
In [8]: y_test.shape
```

```
Out[8]: (10000,)
```

```
In [9]: n_inputs = 28*28  # MNIST
        n_hidden1 = 300
        n_hidden2 = 200
        n_hidden3 = 100
        n_outputs = 10
```

```
In [10]: tf.reset_default_graph()
```

```
In [11]: # Placeholders for data (inputs and targets)
         X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
         y = tf.placeholder(tf.int32, shape=None, name="y")
```

In [12]:
```
def neuron_layer(X, n_neurons, name, activation=None):
    # X input to neuron
    # number of neurons for the layer
    # name of layer
    # pass in eventual activation function

    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])

        # initialize weights to prevent vanishing / exploding gradients
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)

        # Initialize weights for the layer
        W = tf.Variable(init, name="weights")
        # biases
        b = tf.Variable(tf.zeros([n_neurons]), name="bias")

        # Output from every neuron
        Z = tf.matmul(X, W) + b
        if activation is not None:
            return activation(Z)
        else:
            return Z
```

In [33]:
```
# Define the hidden layers
with tf.name_scope("dnn"):
    hidden1 = neuron_layer(X, n_hidden1, name="hidden1",
                           activation=tf.nn.relu)
    hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2",
                           activation=tf.nn.relu)
    hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3",
                           activation=tf.nn.relu)
    logits = neuron_layer(hidden3, n_outputs, name="outputs")
```

In [45]:
```
with tf.name_scope("loss"):
    # logits are from the last output of the dnn
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")
```

In [35]:
```
learning_rate = 0.01

with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

In [36]:
```
# Evaluation to see accuracy

with tf.name_scope("eval"):
```

```
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

1.2.1 Tensorboard

```
In [17]: from datetime import datetime
         import os
         import pathlib

         t = datetime.utcnow().strftime("%Y%m%d%H%M%S")
         log_dir = "tf_logs"
         logd = "/tmp/{}/r{}/".format(log_dir, t)

         # Then every time you have specified a graph run:
         # file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

         # Make directory if it doesn't exist

         from pathlib import Path
         home = str(Path.home())

         logdir = os.path.join(os.sep,home,logd)

         if not os.path.exists(logdir):
             os.makedirs(logdir)
```

```
In [18]: # TensorBoard Graph visualizer in notebook
         import numpy as np
         from IPython.display import clear_output, Image, display, HTML

         def strip_consts(graph_def, max_const_size=32):
             """Strip large constant values from graph_def."""
             strip_def = tf.GraphDef()
             for n0 in graph_def.node:
                 n = strip_def.node.add()
                 n.MergeFrom(n0)
                 if n.op == 'Const':
                     tensor = n.attr['value'].tensor
                     size = len(tensor.tensor_content)
                     if size > max_const_size:
                         tensor.tensor_content = "<stripped %d bytes>"%size
             return strip_def

         def show_graph(graph_def, max_const_size=32):
             """Visualize TensorFlow graph."""
             if hasattr(graph_def, 'as_graph_def'):
                 graph_def = graph_def.as_graph_def()
             strip_def = strip_consts(graph_def, max_const_size=max_const_size)
```

```
        code = """
            <script src="//cdnjs.cloudflare.com/ajax/libs/polymer/0.3.3/platform.js"></sc
            <script>
              function load() {{
                document.getElementById("{id}").pbtxt = {data};
              }}
            </script>
            <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build
            <div style="height:600px">
              <tf-graph-basic id="{id}"></tf-graph-basic>
            </div>
        """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

        iframe = """
            <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></ifra
        """.format(code.replace('"', '&quot;'))
        display(HTML(iframe))
```

```
In [42]: show_graph(tf.get_default_graph())
```

```
<IPython.core.display.HTML object>
```

```
In [38]: init = tf.global_variables_initializer()
         saver = tf.train.Saver()
         train_acc = []
         val_acc = []
         n_epochs = 10
         batch_size = 50

         with tf.Session() as sess:
             init.run()
             for epoch in range(n_epochs):
                 batches = x_train.shape[0] // batch_size
                 for i in range(batches-1):
                     j = i*batch_size
                     X_batch, y_batch = X_train[j:j+batch_size], y_train[j:j+batch_size]
                     #X_val, y_val = X_test[j:j+batch_size,:], y_test[j:j+batch_size,:]
                     sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
                 acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
                 acc_val = accuracy.eval(feed_dict={X: X_test,
                                                     y: y_test})
                 print(epoch, "Train accuracy:", acc_train, "Val accuracy:", acc_val)

             save_path = saver.save(sess, "./model.ckpt") # save model
```

```
0 Train accuracy: 0.96 Val accuracy: 0.9204
1 Train accuracy: 0.98 Val accuracy: 0.9407
2 Train accuracy: 0.98 Val accuracy: 0.9483
```

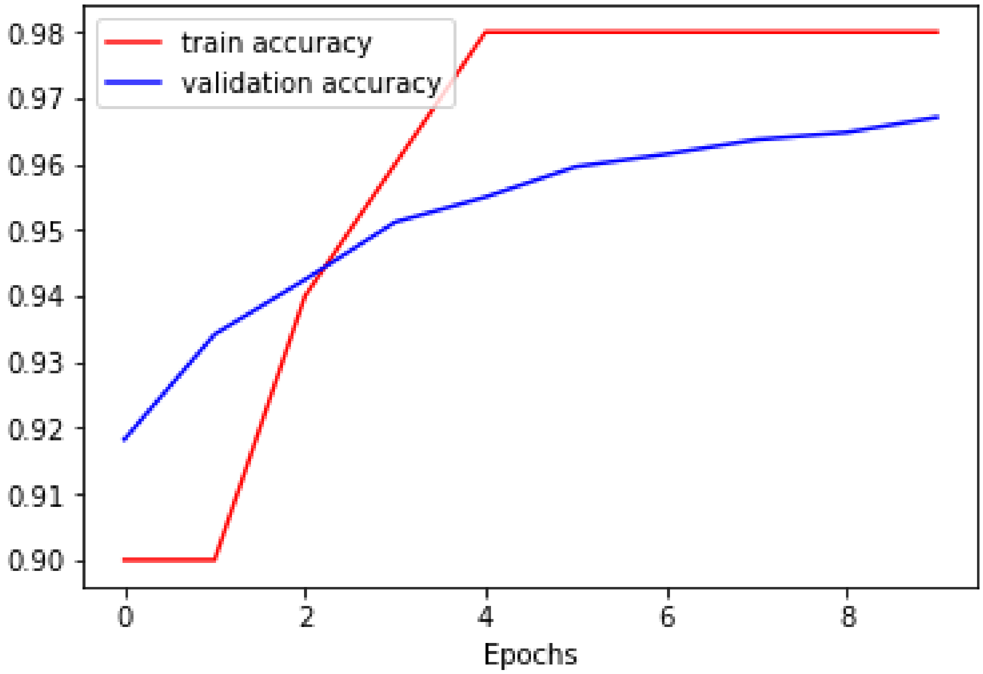
```
3 Train accuracy: 0.98 Val accuracy: 0.9543
4 Train accuracy: 0.96 Val accuracy: 0.9581
5 Train accuracy: 0.96 Val accuracy: 0.9601
6 Train accuracy: 0.96 Val accuracy: 0.9612
7 Train accuracy: 0.98 Val accuracy: 0.9643
8 Train accuracy: 0.98 Val accuracy: 0.9661
9 Train accuracy: 0.98 Val accuracy: 0.9677
```

In [47]:
```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
train_acc = []
val_acc = []
n_epochs = 10
batch_size = 50

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        batches = x_train.shape[0] // batch_size
        for i in range(batches-1):
            j = i*batch_size
            X_batch, y_batch = X_train[j:j+batch_size], y_train[j:j+batch_size]
            #X_val, y_val = X_test[j:j+batch_size,:], y_test[j:j+batch_size,:]
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        train_acc.append(acc_train)
        acc_val = accuracy.eval(feed_dict={X: X_test, y: y_test})
        val_acc.append(acc_val)

        print("Epoch:", epoch, "Train accuracy:", acc_train, "Val accuracy:", acc_val

    save_path = saver.save(sess, "./model.ckpt") # save model
```

```
Epoch: 0 Train accuracy: 0.9 Val accuracy: 0.9183
Epoch: 1 Train accuracy: 0.9 Val accuracy: 0.9342
Epoch: 2 Train accuracy: 0.94 Val accuracy: 0.9425
Epoch: 3 Train accuracy: 0.96 Val accuracy: 0.9512
Epoch: 4 Train accuracy: 0.98 Val accuracy: 0.955
Epoch: 5 Train accuracy: 0.98 Val accuracy: 0.9596
Epoch: 6 Train accuracy: 0.98 Val accuracy: 0.9615
Epoch: 7 Train accuracy: 0.98 Val accuracy: 0.9637
Epoch: 8 Train accuracy: 0.98 Val accuracy: 0.9648
Epoch: 9 Train accuracy: 0.98 Val accuracy: 0.9671
```

In [54]:
```
import matplotlib.pyplot as plt
plt.figure()
```

```
h = [i for i in range (10)]
plt.plot(h, train_acc, c = 'r')
plt.plot(h, val_acc, c = 'b')
plt.legend(['train accuracy', 'validation accuracy'], loc = 'upper left')
plt.xlabel('Epochs')
plt.show()
```



Q2. Use transfer learning and use the Imagenet VGG16 model to train on MNIST data. You can use **Keras** for solving this question. You can choose any requirements on loss function, optimizer etc. **Plot the performance results (Training and Validation accuracies & loss) for every epoch**

```
In [ ]: # Your code here
        from keras.applications.vgg16 import VGG16
        model = VGG16()
```

```
In [4]: from keras.applications import VGG16
        #Load the VGG model
        vgg_conv = VGG16(weights='imagenet', include_top=False, input_shape=(48,48,3))
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/v
54534144/58889256 [===========================>...] - ETA: 0s
```

```
In [5]: # Freeze the layers except the last 2 layers
        for layer in vgg_conv.layers[:-2]:
            layer.trainable = False
```

```
In [7]: from keras import models
        from keras import layers
        from keras import optimizers

        model = models.Sequential()

        model.add(vgg_conv)

        model.add(layers.Flatten())
        model.add(layers.Dense(256, activation='relu'))
        model.add(layers.Dense(10, activation='softmax'))
```

```
In [8]: model.compile(loss='categorical_crossentropy',
                      optimizer=optimizers.Adam(lr=0.01),
                      metrics=['acc'])
```

```
In [13]: # doing on first 5000 images due to memory constraint
         X_train2 = X_train[:5000]
         y_train2 = y_train[:5000]
```

```
In [16]: import cv2
         import numpy as np
         dim = (48, 48)
         #convert 28x28 grayscale to 48x48 rgb channels
         def to_rgb(img):
             img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
             img_rgb = np.asarray(np.dstack((img, img, img)), dtype=np.uint8)
             return img_rgb

         rgb_list = []
         #convert X_train data to 48x48 rgb values
         for i in range(len(X_train2)):
             rgb = to_rgb(X_train2[i])
             rgb_list.append(rgb)
             #print(rgb.shape)

         rgb_arr = np.stack([rgb_list],axis=4)
         rgb_arr_to_3d = np.squeeze(rgb_arr, axis=4)
```

```
In [17]: X_test2 = X_test[:2000]
         y_test2 = y_test[:2000]
         rgb_list2 = []
         #convert X_train data to 48x48 rgb values
         for i in range(len(X_test2)):
             rgb = to_rgb(X_test2[i])
             rgb_list2.append(rgb)
             #print(rgb.shape)
```

```
        rgb_arr2 = np.stack([rgb_list2],axis=4)
        rgb_arr_to_3d2 = np.squeeze(rgb_arr2, axis=4)
```

```
In [23]: Y_train2 = Y_train[:5000]
         Y_test2 = Y_test[:2000]
```

```
In [ ]: model.fit(rgb_arr_to_3d, Y_train2,
                  batch_size=128, nb_epoch=4,
                   verbose=1,
                  validation_data=(rgb_arr_to_3d2, Y_test2))

        # Due to lack of memory in datahub (1 Gb space, kernel keeps dying at this point)
```

```
Train on 5000 samples, validate on 2000 samples
Epoch 1/4
```

EXTRA CREDIT Q. (MANDATORY for students taking IND ENG 290) Customize your neural networks in **Q1** to how many ever layers you want, use batch normalization and Adam Optimizer and try different regularization techniques to combat overfitting. Also use as many iterations you want and plot every 10th iteration on the tensorboard. We will give extra credit if you achieve more than **98.5%** on the MNIST data. **Plot the neural network graph (using tensorboard) and describe the settings that you used and the performance results. Also plot performance results (Training and Validation accuracies & loss) for every epoch**

Note: You can use Keras if necessary for solving this question

If you cannot run your tensorflow notebooks locally, you can use. https://datahub.berkeley.edu/hub/home

```
In [13]: # Your code here
         # Using dropout on all layers and using different optimizers like Adam optimizer
         with tf.name_scope("dnn"):
             hidden1 = neuron_layer(X, n_hidden1, name="hidden1",
                                    activation=tf.nn.relu)
             hidden1 = tf.nn.dropout(hidden1, 0.9)
             hidden2 = neuron_layer(hidden1, n_hidden2, name="hidden2",
                                    activation=tf.nn.relu)
             hidden2 = tf.nn.dropout(hidden2, 0.9)
             hidden3 = neuron_layer(hidden2, n_hidden3, name="hidden3",
                                    activation=tf.nn.relu)
             hidden3 = tf.nn.dropout(hidden3, 0.9)

             logits = neuron_layer(hidden3, n_outputs, name="outputs")
```

```
In [26]: with tf.name_scope("loss"):
             # logits are from the last output of the dnn
             xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,logits=logits)
             loss = tf.reduce_mean(xentropy, name="loss")

         learning_rate = 0.01
```

```
with tf.name_scope("train"):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)

# Evaluation to see accuracy

with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

In [19]:
```
show_graph(tf.get_default_graph())
```

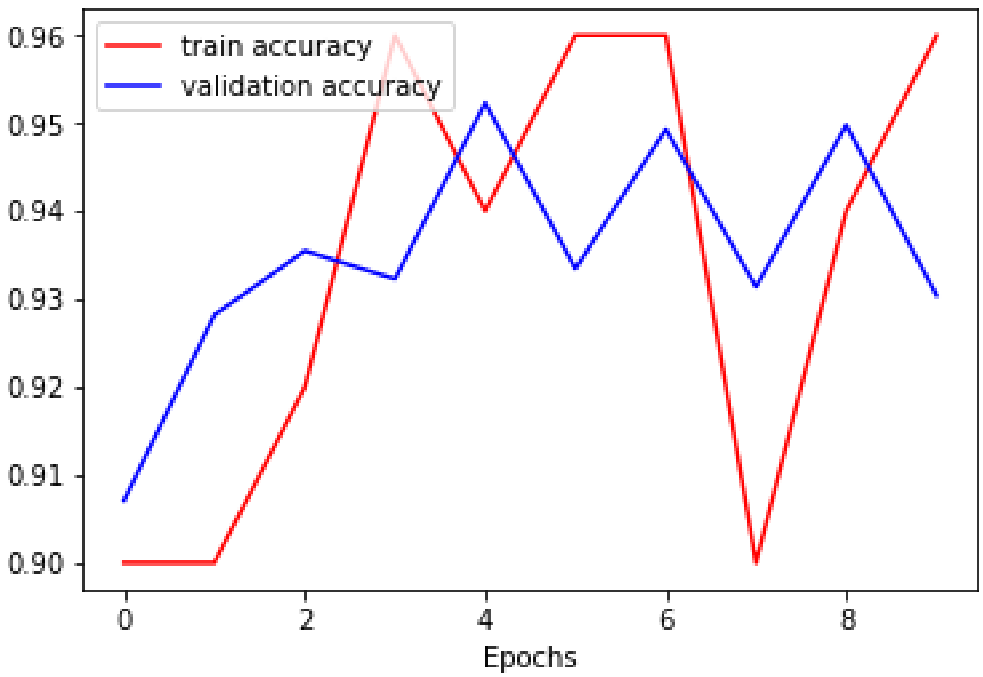
```
<IPython.core.display.HTML object>
```

In [27]:
```
init = tf.global_variables_initializer()
saver = tf.train.Saver()
train_acc = []
val_acc = []
n_epochs = 10
batch_size = 50

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        batches = X_train.shape[0] // batch_size
        for i in range(batches-1):
            j = i*batch_size
            X_batch, y_batch = X_train[j:j+batch_size], y_train[j:j+batch_size]
            #X_val, y_val = X_test[j:j+batch_size,:], y_test[j:j+batch_size,:]
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        train_acc.append(acc_train)
        acc_val = accuracy.eval(feed_dict={X: X_test, y: y_test})
        val_acc.append(acc_val)

        print("Epoch:", epoch, "Train accuracy:", acc_train, "Val accuracy:", acc_val)

    save_path = saver.save(sess, "./model.ckpt") # save model
```

```
Epoch: 0 Train accuracy: 0.9 Val accuracy: 0.9071
Epoch: 1 Train accuracy: 0.9 Val accuracy: 0.9282
Epoch: 2 Train accuracy: 0.92 Val accuracy: 0.9355
Epoch: 3 Train accuracy: 0.96 Val accuracy: 0.9323
Epoch: 4 Train accuracy: 0.94 Val accuracy: 0.9523
Epoch: 5 Train accuracy: 0.96 Val accuracy: 0.9335
Epoch: 6 Train accuracy: 0.96 Val accuracy: 0.9493
Epoch: 7 Train accuracy: 0.9 Val accuracy: 0.9314
```

```
Epoch: 8 Train accuracy: 0.94 Val accuracy: 0.9498
Epoch: 9 Train accuracy: 0.96 Val accuracy: 0.9304
```

In [28]:
```python
import matplotlib.pyplot as plt
plt.figure()
h = [i for i in range (10)]
plt.plot(h, train_acc, c = 'r')
plt.plot(h, val_acc, c = 'b')
plt.legend(['train accuracy', 'validation accuracy'], loc = 'upper left')
plt.xlabel('Epochs')
plt.show()
```



In [ ]: