ECE471, Selected Topics in Machine Learning – Assignment 4
cifar10 and cifar100 Classifiers
Nikola Janjušević

October 3, 2018

Remarks

I've put a lot of time into this project, sadly, to no avail. My best results are from one of my early adaptations of the MNIST assignment where my code is incredibly ugly and I'm fairly sure batch normalization is implemented incorrectly.

One of the limiting factors in my explorations this week has honestly been my computing power. I just don't have time to run training with an insane number of parameters on my laptop. Starting next week I'll (somehow) be running my code on something more capable.

Results

Cifar10:

- Top-1 Accuracy: 70%

Cifar100:

- Top-1 Accuracy: 18%
- Top-5 Accuracy: 40%

Program: cifar10v2.py

```
from fcns import *
import numpy as np
from tqdm import tqdm
import tensorflow as tf
import matplotlib.pyplot as plt

logs_path = "./tf_logs/"

# hyper-parameters
input_size = 32 # 32x32 imgs
num_channels = 3 # RGB
num_classes = 10
dim_layer = [num_channels, 32, 32, 32, 32, 32, 32, 64, 64, num_classes]
cp = { # conv parameters
    1:{
        'k':5, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':2, # conv stride
        'ps':1, # pool stride,
        'kpad':"SAME",
        'ppad':"SAME"
    },
    2:{
        'k':3, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':1, # conv stride
        'ps':2, # pool stride,
        'kpad':"SAME",
        'ppad':"SAME"
    },
    3:{
        'k':3, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':1, # conv stride
        'ps':1, # pool stride,
        'kpad':"VALID",
        'ppad':"VALID"
    },
    4:{
        'k':3, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':1, # conv stride
        'ps':1, # pool stride,
        'kpad':"VALID",
        'ppad':"VALID"
    },
    5:{
        'k':3, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':1, # conv stride
        'ps':1, # pool stride,
        'kpad':"SAME",
        'ppad':"SAME"
    },
    6:{
        'k':3, # conv window size (kxk)
        'p':2, # pool window size (pxp)
        'ks':1, # conv stride
        'ps':1, # pool stride,
        'kpad':"SAME",
        'ppad':"SAME"
    }
}

d1 = dim(input_size,cp[1])
d2 = dim(d1,cp[2])
d3 = dim(d2,cp[3])
d4 = dim(d3,cp[4])
d5 = dim(d4,cp[5])
d6 = dim(d4,cp[6])
print(d1,d2,d3,d4,d5,d6)
```

```
72
73  BATCH_SIZE = 300
74  NUM_EPOCHS = 30
75  learning_rate = .1
76  display_epoch = 1
77
78  # regularization parameters
79  l2_lambda = .01/(sum([dim**2 for dim in dim_layer]))
80  print(l2_lambda)
81  drop_out = .9
82
83  def main():
84      class Data(object):
85          def __init__(self):
86              np.random.seed(31415)
87              ([self.x_train, self.x_val, self.x_test],
88                  [self.y_train, self.y_val, self.y_test])  = \
89                  loadData("./cifar10_data")
90
91              self.index = np.arange(self.x_train.shape[0])
92
93          def get_batch(self):
94              choices = np.random.choice(self.index, size=BATCH_SIZE)
95              return self.x_train[choices,:,:,:], self.y_train[choices,:]
96
97      x = tf.placeholder(tf.float32, [None,input_size,input_size,num_channels])
98      y = tf.placeholder(tf.float32, [None,num_classes])
99      keep_prob = tf.placeholder(tf.float32) # for dropout
100
101     # f:R28x28 -> R10
102     def f(x):
103         layer_1 = tf.nn.dropout( conv_layer(x, W[1], b[1], cp[1]), keep_prob )
104         layer_2 = tf.nn.dropout( conv_layer(layer_1, W[2], b[2], cp[2]), keep_prob )
105         layer_3 = tf.nn.dropout( conv_layer(layer_2, W[3], b[3], cp[3]), keep_prob )
106         layer_4 = tf.nn.dropout( conv_layer(layer_3, W[4], b[4], cp[4]), keep_prob )
107         layer_5 = tf.nn.dropout( conv_layer(layer_4, W[5], b[5], cp[5]), keep_prob )
108         layer_6 = tf.nn.dropout( conv_layer(layer_5, W[6], b[6], cp[6]), keep_prob )
109         layer_7 = tf.nn.dropout( fc_layer(layer_6, W[7], b[7]), keep_prob )
110         layer_8 = tf.nn.dropout( fc_layer(layer_7, W[8], b[8]), keep_prob )
111         return tf.squeeze(
112             tf.add( tf.matmul(layer_8, W['out']), b['out'] )
113         )
114
115     # Store layers weight & bias
116     # default dtype=float32
117     # WEIGHTS
118     W = {
119         1: tf.Variable(tf.random_normal( [cp[1]['k'],cp[1]['k'], num_channels, dim_layer
                [1]] )),
120         2: tf.Variable(tf.random_normal( [cp[2]['k'],cp[2]['k'], dim_layer[1], dim_layer
                [2]] )),
121         3: tf.Variable(tf.random_normal( [cp[3]['k'],cp[3]['k'], dim_layer[2], dim_layer
                [3]] )),
122         4: tf.Variable(tf.random_normal( [cp[4]['k'],cp[4]['k'], dim_layer[3], dim_layer
                [4]] )),
123         5: tf.Variable(tf.random_normal( [cp[5]['k'],cp[5]['k'], dim_layer[4], dim_layer
                [5]] )),
124         6: tf.Variable(tf.random_normal( [cp[6]['k'],cp[6]['k'], dim_layer[5], dim_layer
                [6]] )),
125         7: tf.Variable(tf.random_normal( [d6*d6*dim_layer[6], dim_layer[7]] )),
126         8: tf.Variable(tf.random_normal( [dim_layer[7], dim_layer[8]] )),
127         'out': tf.Variable(tf.random_normal( [dim_layer[8], num_classes] ))
128     }
129     # BIASES
130     b = {
131         1: tf.Variable(tf.random_normal([dim_layer[1]])),
132         2: tf.Variable(tf.random_normal([dim_layer[2]])),
133         3: tf.Variable(tf.random_normal([dim_layer[3]])),
134         4: tf.Variable(tf.random_normal([dim_layer[4]])),
135         5: tf.Variable(tf.random_normal([dim_layer[5]])),
136         6: tf.Variable(tf.random_normal([dim_layer[6]])),
137         7: tf.Variable(tf.random_normal([dim_layer[7]])),
138         8: tf.Variable(tf.random_normal([dim_layer[8]])),
```

```
        'out': tf.Variable(tf.random_normal([num_classes]))
    }

    # models
    logits = f(x)
    prediction = tf.nn.softmax(logits)
    correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

    # binar cross entropy loss with L2 penalty on weights
    loss = tf.reduce_mean( tf.losses.softmax_cross_entropy(y, logits) ) + \
        l2_lambda*tf.reduce_sum(
            [tf.nn.l2_loss(var) for var in
            tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES)]
        )
    optim = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)
    init = tf.global_variables_initializer()

    # Create a summary to monitor cost tensor
    tf.summary.scalar("loss", loss)
    # Create a summary to monitor accuracy tensor
    tf.summary.scalar("accuracy", accuracy)
    # Merge all summaries into a single op
    merged_summary_op = tf.summary.merge_all()

    with tf.Session() as sess:
        data = Data()
        sess.run(init)
        # op to write logs to Tensorboard
        summary_writer = tf.summary.FileWriter(logs_path,
            graph=tf.get_default_graph())
        # training
        for epoch in range(NUM_EPOCHS):
            avg_cost = 0.
            num_batches = int( np.ceil( data.index[-1] / BATCH_SIZE ) )

            for i in tqdm(range(num_batches)):
                xb, yb = data.get_batch()
                loss_np, _, summary = sess.run([loss, optim, merged_summary_op],
                    feed_dict={x: xb, y: yb, keep_prob: drop_out})
                # logs every batch
                summary_writer.add_summary(summary, epoch * num_batches + i)
                avg_cost  += loss_np/num_batches

            # Display logs per epoch step
            if (epoch+1) % display_epoch == 0:
                print("Epoch:", '%02d' % (epoch+1),
                    "cost=", "{:.6f}".format(avg_cost))
            print('Validation Set Accuracy:',
                accuracy.eval({x: data.x_val, y: data.y_val, keep_prob: 1.0}))

        # Test the model on separate data
        print('Test Set Accuracy:',
            accuracy.eval({x: data.x_test, y: data.y_test, keep_prob: 1.0}))

        print("Run the command line:\n--> tensorboard --logdir=./tf_logs ")

# -------------- MODEL FUNCTIONS -----------------

# convolution layer: conv-> +bias -> activation -> pool
def conv_layer(x, W, b, cp):
    x = tf.nn.conv2d(x, W, strides=[1, cp['ks'], cp['ks'], 1],
        padding=cp['kpad'])
    x = tf.add(x,b)
    mean, var = tf.nn.moments(x, axes=[0,1,2])
    x = tf.nn.batch_normalization(x, mean, var, 0, 1, .001)
    x = tf.nn.relu6(x)
    return tf.nn.avg_pool(x, ksize = [1, cp['p'], cp['p'], 1],
        strides = [1, cp['ps'], cp['ps'], 1], padding=cp['ppad'])

# fully connected layer
def fc_layer(x, W, b):
    x = tf.add( tf.matmul( tf.reshape(x, [-1, tf.shape(W)[0]]), W ), b)
```

```
212     mean, var = tf.nn.moments(x, axes=[0])
213     x = tf.nn.batch_normalization(x, mean, var, 0, 1, .001)
214     return tf.nn.relu6(x)
215
216 if __name__ == '__main__':
217 main()
```

Program: cifar100.py

```
1  import os
2  import pickle
3  import numpy as np
4  from tqdm import tqdm
5  import tensorflow as tf
6  import matplotlib.pyplot as plt
7  from tensorflow.keras.datasets.cifar100 import load_data
8
9  logs_path = "./tf_logs/"
10
11 input_size = 32 # 32x32 imgs
12 num_channels = 3 # RGB
13 num_classes = 100
14
15 # hyper-parameters
16 BATCH_SIZE = 200
17 NUM_EPOCHS = 15
18 display_epoch = 1
19 LEARNING_RATE = 5
20 # regularization parameters
21 drop_prob = 0.1
22 reg_scale = 1e-6
23
24 # https://stackoverflow.com/questions/38592324/one-hot-encoding-using-numpy/38592416
25 def get_one_hot(targets, nb_classes):
26     res = np.eye(nb_classes)[np.array(targets).reshape(-1)]
27     return res.reshape(list(targets.shape)+[nb_classes])
28
29 class Data(object):
30     def __init__(self):
31         np.random.seed(31415)
32         (self.x_train, y_train), (x_test, y_test) = load_data()
33
34         (self.x_train, x_test) = (self.x_train / 255.0, x_test / 255.0)
35
36         [(self.x_val, self.x_test), (y_val, y_test)] = \
37             [np.split(var,2) for var in [x_test, y_test]]
38
39         [self.y_train, self.y_val, self.y_test] = \
40             [np.squeeze(get_one_hot(y, num_classes)) for y in [y_train,y_val,y_test]]
41
42         self.index = np.arange(self.x_train.shape[0])
43
44     def get_batch(self):
45         choices = np.random.choice(self.index, size=BATCH_SIZE)
46         return self.x_train[choices,:,:,:], self.y_train[choices,:]
47
48 # -------------- MODEL FUNCTIONS -----------------
49
50 # --- CONV LAYER WRAPPER --- w/ L2 regularization
51 # conv -> dropout -> BN -> relu -> max_pool
52 def conv_layer(input, filters, kernel_size, strides=2, is_training=True):
53     x = tf.layers.conv2d(
54         input, filters, kernel_size, strides=strides, padding='same',
55         kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale)
56     )
57     x = tf.layers.batch_normalization(x, training=phase, renorm=False)
58     x = tf.nn.relu6(x)
59     return x
60
61 # --- FULLY CONNECTED LAYER WRAPPER ---
62 # matmul -> dropout -> BN -> relu
63 def fc_layer(input, units, is_training=True):
64     x = tf.layers.dense(
65         input, units,
66         kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale)
67     )
68     x = tf.layers.batch_normalization(x, training=is_training, renorm=False)
69     x = tf.nn.relu6(x)
70     x = tf.layers.dropout(x, rate=drop_prob, training=is_training)
71     return x
```

```
72
73  x = tf.placeholder(tf.float32, [None,input_size,input_size,num_channels])
74  y = tf.placeholder(tf.float32, [None,num_classes])
75  phase = tf.placeholder(tf.bool) # is_training
76  lr = tf.placeholder(tf.float32) # is_training
77
78  # ---- NN -----
79
80  def f(x):
81      x = conv_layer(x, 32, 3, strides=2, is_training=phase)
82      print(x.get_shape())
83      x = conv_layer(x, 32, 3, strides=2, is_training=phase)
84      print(x.get_shape())
85
86      x = tf.layers.max_pooling2d(x, 3, 2, padding='same')
87      print(x.get_shape())
88
89      x = conv_layer(x, 32, 3, is_training=phase)
90      print(x.get_shape())
91      x = conv_layer(x, 32, 3, is_training=phase)
92      print(x.get_shape())
93
94      x = tf.layers.max_pooling2d(x, 3, 2, padding='same')
95      print(x.get_shape())
96
97      x = tf.layers.flatten(x)
98      x = fc_layer(x, 64, is_training=phase)
99      x = tf.layers.dense(x, num_classes)
100     return x
101
102 # models
103 logits = f(x)
104 prediction = tf.nn.softmax(logits)
105
106 correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))
107 accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
108
109 # FROM KAROL
110 trash, guess_5 = tf.nn.top_k(prediction, k=5)
111 actual = tf.argmax(y, axis=1)
112 actual = tf.transpose(tf.stack([actual,actual,actual,actual,actual]))
113 correct_pred_5 = tf.equal(guess_5, tf.cast(actual, tf.int32))
114 accuracy_5 = 5.0*tf.reduce_mean(tf.cast(correct_pred_5, tf.float32))
115
116 # LOSS
117 update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
118 with tf.control_dependencies(update_ops):
119     loss = tf.reduce_mean( tf.losses.softmax_cross_entropy(y, logits) ) \
120         + tf.reduce_mean( tf.losses.get_regularization_loss() )
121     optim = tf.train.GradientDescentOptimizer(learning_rate=lr).minimize(loss)
122
123 init = tf.global_variables_initializer()
124
125 # Create a summary to monitor cost tensor
126 tf.summary.scalar("loss", loss)
127 # Create a summary to monitor accuracy tensor
128 tf.summary.scalar("accuracy", accuracy)
129 # Merge all summaries into a single op
130 merged_summary_op = tf.summary.merge_all()
131
132 with tf.Session() as sess:
133     data = Data()
134     sess.run(init)
135     # op to write logs to Tensorboard
136     summary_writer = tf.summary.FileWriter(logs_path,
137         graph=tf.get_default_graph())
138     learning_rate = LEARNING_RATE
139     # training
140     for epoch in range(NUM_EPOCHS):
141         avg_cost = 0.
142         num_batches = int( np.ceil( data.index[-1] / BATCH_SIZE ) )
143
144         if epoch%3 == 0:
```

```
145             learning_rate = learning_rate/100
146
147         for i in tqdm(range(num_batches)):
148             xb, yb = data.get_batch()
149             loss_np, _, summary = sess.run([loss, optim, merged_summary_op],
150                 feed_dict={x: xb, y: yb, phase: True, lr: learning_rate})
151             # logs every batch
152             summary_writer.add_summary(summary, epoch * num_batches + i)
153             avg_cost  += loss_np/num_batches
154
155         # Display logs per epoch step
156         if (epoch+1) % display_epoch == 0:
157             print("Epoch:", '%02d' % (epoch+1),
158                 "cost=", "{:.6f}".format(avg_cost))
159         print('TOP-1 Validation Set Accuracy:',
160             accuracy.eval({x: data.x_val, y: data.y_val, phase: False}))
161
162         print('TOP-5 Validation Set Accuracy:',
163             accuracy_5.eval({x: data.x_val, y: data.y_val, phase: False}))
164
165     # Test the model on separate data
166     print('TOP-1 Test Set Accuracy:',
167         accuracy.eval({x: data.x_test, y: data.y_test, phase: False}))
168
169     print('TOP-5 Test Set Accuracy:',
170         accuracy_5.eval({x: data.x_test, y: data.y_test, phase: False}))
171
172     print("Run the command line:\n--> tensorboard --logdir=./tf_logs ")
```