Caleb Zulawski

Homework 3 - MNIST

My convolutional network network obtained slightly better than 98.0% accuracy.

Homework 4 - CIFAR-10/100

State of the art performance for the CIFAR-10 dataset seems to be 95+%, and about 75% for CIFAR-100. I implemented a residual network which scored about 74.4% on CIFAR-10 and 33.9% on CIFAR-100. This is not very close to current performance, but was not unexpected due to the network construction. To reduce training time on available hardware I used only 4 residual blocks, as opposed to the 16 in ResNet-50, and even more in larger networks. I also had to reduce the number of features, and train on the original 32x32 images rather than upsampled 224x224 that many networks seem to use. It's also possible that the values used were not optimal for a 4 residual block network.

Beyond increasing the size of the network, I would have liked to try more optimizers. I used the momentum optimizer with a momentum of 0.9, based on some successes I saw in other papers. I would have also liked to use a learning rate schedule to speed up early training while training better in later epochs, but I compromised and used a constant rate of 0.01. Finally, I would have liked to use augmented data.

```
#!/usr/bin/env python3

import tensorflow as tf
import tensorflow.contrib.slim as slim
from tensorflow.contrib.learn.python.learn.datasets import mnist
import numpy as np

class Model():
    def __init__(self, sess, pixels, n_batch, n_classes, learning_rate):
        self.sess = sess
        self.learning_rate = learning_rate
        self.pixels = pixels
        self.n_batch = n_batch
        self.n_classes = n_classes
        self.build_model()

    def build_model(self):
        self.inputs = tf.placeholder(tf.float32, [None, self.pixels, self.pixels, 1])
        self.labels = tf.placeholder(tf.float32, [None, self.n_classes])
        with slim.arg_scope([slim.conv2d, slim.fully_connected],
                            activation_fn=tf.nn.relu,
                            weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                            weights_regularizer=slim.l2_regularizer(0.0005)):
            net = self.inputs
            net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], scope=’conv1’)
            net = slim.max_pool2d(net, [2, 2], scope=’pool1’)
            net = slim.repeat(net, 1, slim.conv2d, 64, [5, 5], scope=’conv2’)
            net = slim.max_pool2d(net, [2, 2], scope=’pool2’)
            net = slim.flatten(net, scope=’flatten2’)
            net = slim.fully_connected(net, 1024, scope=’fc3’)
            net = slim.dropout(net, 0.8, scope=’dropout3’)
            net = slim.fully_connected(net, self.n_classes, activation_fn=None, normalizer_fn=None, scope=’fc4’)
        self.predictions = net
        slim.losses.softmax_cross_entropy(self.predictions, self.labels)
        self.loss = slim.losses.get_total_loss(add_regularization_losses=True)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate).minimize(self.loss)

        correct = tf.equal(tf.argmax(self.predictions,1), tf.argmax(self.labels,1))
        self.accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    def train_minibatch(self, xs, ys):
        self.sess.run(self.optimizer, feed_dict={self.inputs: xs, self.labels: ys})

    def train(self, xs, ys, xvs, yvs, epochs):
        num_minibatches = xs.shape[0] // self.n_batch
        for epoch in range(epochs):
            p = np.random.permutation(xs.shape[0])
            for i in range(num_minibatches):
                # print("Minibatch " + str(i) + ’/’ + str(num_minibatches))
                start = i * self.n_batch
                end = (i + 1) * self.n_batch
                self.train_minibatch(xs[p][start:end], ys[p][start:end])
            accuracy = self.validate(xvs, yvs)
            print("Epoch {} validation accuracy: {}".format(epoch, accuracy))

    def validate(self, xs, ys):
        return self.sess.run(self.accuracy, feed_dict={self.inputs: xs, self.labels: ys})

    def test(self, xs):
        return self.sess.run(self.predictions, feed_dict={self.inputs: xs})

with tf.Session() as sess:
    data = mnist.read_data_sets("MNIST_data/", dtype=tf.uint8, reshape=False, one_hot=True)
    m = Model(sess, 28, 50, 10, 0.001)
    sess.run(tf.global_variables_initializer())
    m.train(data.train.images, data.train.labels, data.validation.images, data.validation.labels, 2)
    test_accuracy = m.validate(data.test.images, data.test.labels)
    print("Test set accuracy: {}".format(test_accuracy))
```

```
#!/usr/bin/env python3

import tensorflow as tf
import tensorflow.contrib.slim as slim
import numpy as np

# from tflearn.datasets import cifar10
from tflearn.datasets import cifar100

def residual(inputs, depth, kernel, scope=’residual’):
    with tf.variable_scope(scope, ’residual’, [inputs]) as sc:
        residual = inputs
        residual = slim.batch_norm(residual, activation_fn=tf.nn.relu, scope=’bn1’)
        residual = slim.conv2d(residual, depth, kernel, scope=’conv1’)
        residual = slim.batch_norm(residual, activation_fn=tf.nn.relu, scope=’bn2’)
        residual = slim.conv2d(residual, inputs.get_shape()[3], kernel, scope=’conv2’)
        residual = residual + inputs
    return residual

def resnet_small(inputs,
                 num_classes=None,
                 global_pool=True,
                 output_stride=None,
                 reuse=None,
                 scope=’resnet_small’):
    with tf.variable_scope(scope, ’resnet_small’, [inputs]) as sc:
        net = inputs
        net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], stride=1, scope=’conv1’)
        net = slim.max_pool2d(net, [2, 2], scope=’pool1’)
        net = slim.repeat(net, 1, slim.conv2d, 32, [5, 5], stride=1, scope=’conv2’)
        net = slim.max_pool2d(net, [2, 2], scope=’pool2’)
        net = residual(net, 32, [3, 3], scope=’block3’)
        net = residual(net, 32, [3, 3], scope=’block4’)
        net = residual(net, 64, [3, 3], scope=’block5’)
        net = residual(net, 64, [3, 3], scope=’block6’)
        net = slim.flatten(net, scope=’flatten6’)
        net = slim.fully_connected(net, 1024, scope=’fc7’)
        net = slim.dropout(net, 0.5, scope=’dropout8’)
        net = slim.fully_connected(net, num_classes, activation_fn=None, normalizer_fn=None, scope=’logits’)
    return net

class Model():
    def __init__(self, sess, n_batch, n_classes, learning_rate):
        self.sess = sess
        self.learning_rate = learning_rate
        self.n_batch = n_batch
        self.n_classes = n_classes
        self.build_model()

    def build_model(self):
        self.inputs = tf.placeholder(tf.float32, [None, 32, 32, 3])
        self.labels = tf.placeholder(tf.float32, [None, self.n_classes])
        with slim.arg_scope([slim.conv2d, slim.fully_connected],
                            activation_fn=tf.nn.relu,
                            weights_initializer=tf.truncated_normal_initializer(0.0, 0.01),
                            weights_regularizer=slim.l2_regularizer(0.002)):
            net = self.inputs
            net = resnet_small(net, num_classes=self.n_classes)
        self.predictions = net
        slim.losses.softmax_cross_entropy(self.predictions, self.labels)
        self.loss = slim.losses.get_total_loss(add_regularization_losses=True)
        self.optimizer = tf.train.MomentumOptimizer(learning_rate=self.learning_rate, momentum=0.9).minimize(self.loss)

        correct = tf.equal(tf.argmax(self.predictions,1), tf.argmax(self.labels,1))
        self.accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

    def train_minibatch(self, xs, ys):
        self.sess.run(self.optimizer, feed_dict={self.inputs: xs, self.labels: ys})

    def train(self, xs, ys, xvs, yvs, epochs):
        num_minibatches = xs.shape[0] // self.n_batch
        for epoch in range(epochs):
            p = np.random.permutation(xs.shape[0])
            for i in range(num_minibatches):
                # print("Minibatch " + str(i) + '/' + str(num_minibatches))
                start = i * self.n_batch
                end = (i + 1) * self.n_batch
                self.train_minibatch(xs[p][start:end], ys[p][start:end])
            accuracy = self.validate(xvs, yvs)
            print("Epoch {} validation accuracy: {}".format(epoch, accuracy))

    def validate(self, xs, ys):
        return self.sess.run(self.accuracy, feed_dict={self.inputs: xs, self.labels: ys})

    def test(self, xs):
        return self.sess.run(self.predictions, feed_dict={self.inputs: xs})

with tf.Session() as sess:
    # (train_images, train_labels), (test_images, test_labels) = cifar10.load_data(one_hot=True)
    # m = Model(sess, 128, 10, 0.01)
    (train_images, train_labels), (test_images, test_labels) = cifar100.load_data(one_hot=True)
    m = Model(sess, 128, 100, 0.01)
    sess.run(tf.global_variables_initializer())
    validation_images = train_images[0:5000]
    validation_labels = train_labels[0:5000]
    train_images = train_images[5000:]
    train_labels = train_labels[5000:]
    m.train(train_images, train_labels, validation_images, validation_labels, 100)
    test_accuracy = m.validate(test_images, test_labels)
    print("Test set accuracy: {}".format(test_accuracy))
```