Assignment 2

Visualizing a CNN with CIFAR10

a) CIFAR10 Datasetntrain = 500 # per class

I choose 500 training examples and 88 test examples for each class. There are 10 classes total, each with image size 28*28. The batch size is set to be 50.

```
ntrain =  500 # per class
ntest =  88 # per class
nclass =  10 # number of classes
imsize = 28
nchannels = 3
batchsize = 50
```

b) Train LeNet5 on CIFAR10

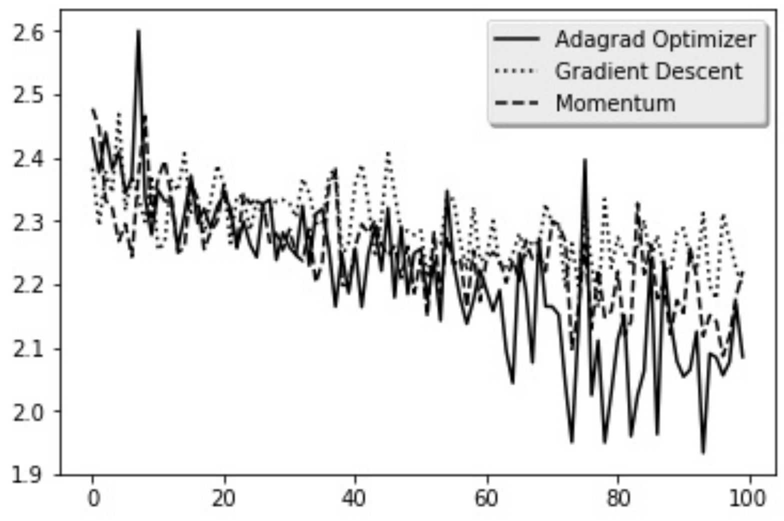
Since we have defined the functions conv2d and max*pool*2x2, we can just stacked the layers one by one. For optimization method, I checked GradientDescent, Adagrad, and Momentum method with 0.08 momentum. For learning rate, I tested 1e-2, 1e-3, and 1e-4. Those comparisions are visualized later.

```
# model
#create your model

# first convolutional layer and max pooling layer
W_conv1 = weight_variable([5, 5, 3, 32], 'W_conv1')
b_conv1 = bias_variable([32])
h_conv1 = tf.nn.tanh(conv2d(tf_data, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# second convolutional layer and max pooling layer
W_conv2 = weight_variable([5, 5, 32, 64], 'W_conv2')
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.tanh(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

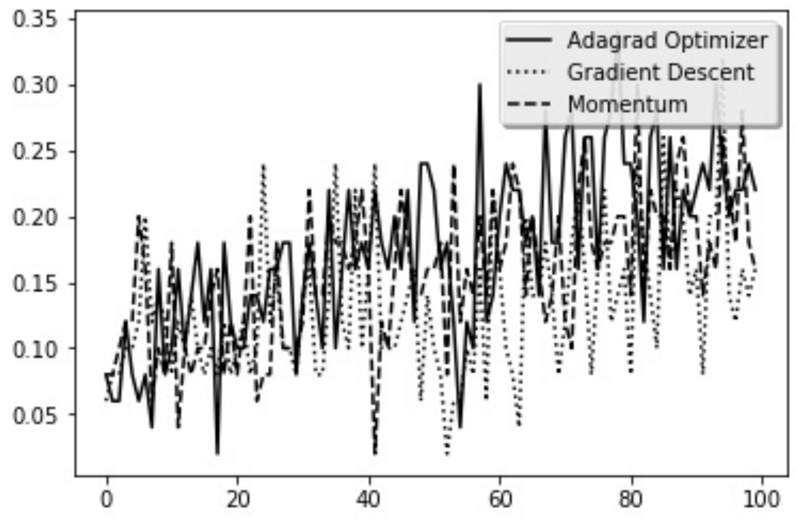
```
# first fully connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024], 'W_fc1')
b_fc1 = bias_variable([1024])
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.tanh(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# second fully connected layer
W_fc2 = weight_variable([1024, 10], 'W_fc2')
b_fc2 = bias_variable([10])
h_fc2 = tf.matmul(h_fc1_drop, W_fc2) + b_fc2

# ----------------------------------------------------
# loss
#set up the loss, optimization, evaluation, and accuracy
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=tf_labe
ls, logits=h_fc2))
optimizer = tf.train.GradientDescentOptimizer (1e-2).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(h_fc2, 1), tf.argmax(tf_labels, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# ----------------------------------------------------
# optimization and test
sess.run(tf.global_variables_initializer())
batch_xs = np.zeros((batchsize, imsize, imsize, nchannels))
batch_ys = np.zeros((batchsize, nclass))
gd_losses = list()
gd_accs = list()
def train():
    for i in range(1000):
        perm = np.arange(ntrain*nclass)
        np.random.shuffle(perm)
        for j in range(batchsize):
            batch_xs[j,:,:,:] = Train[perm[j],:,:,:]
            batch_ys[j,:] = LTrain[perm[j],:]
        loss = cross_entropy.eval(feed_dict = {tf_data:batch_xs, tf_labels:batch_ys,
keep_prob: 0.5})
        gd_losses.append(loss)
        acc = accuracy.eval(feed_dict = {tf_data:batch_xs, tf_labels:batch_ys, keep_p
rob: 0.5})
        gd_accs.append(acc)
        if i%100 == 0:
            print('{}th step, loss is {}, train accuracy is {}'.format(i,loss,acc))
        optimizer.run(feed_dict={tf_data:batch_xs, tf_labels:batch_ys, keep_prob: 0.5
})

    # test
    print("test accuracy %g"%accuracy.eval(feed_dict={tf_data: Test, tf_labels: LTest
, keep_prob: 1.0}))
```

Plot loss and accuracy changes among different optimization methods

1. Loss change with different optimization methods

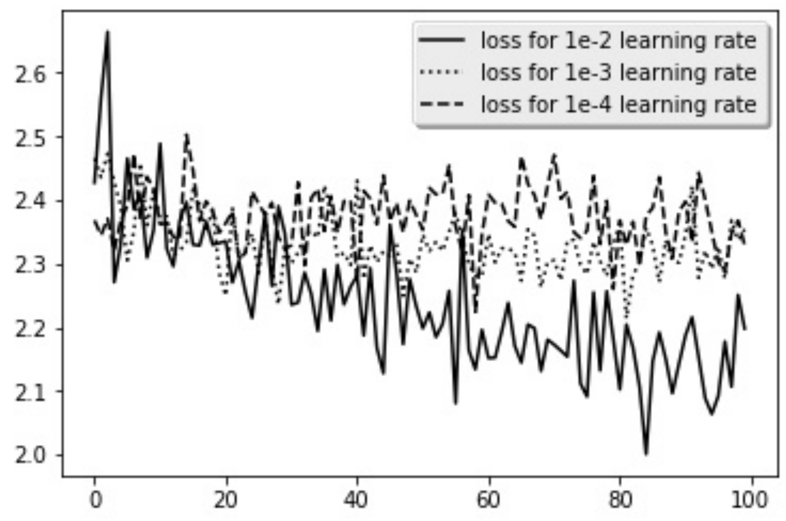


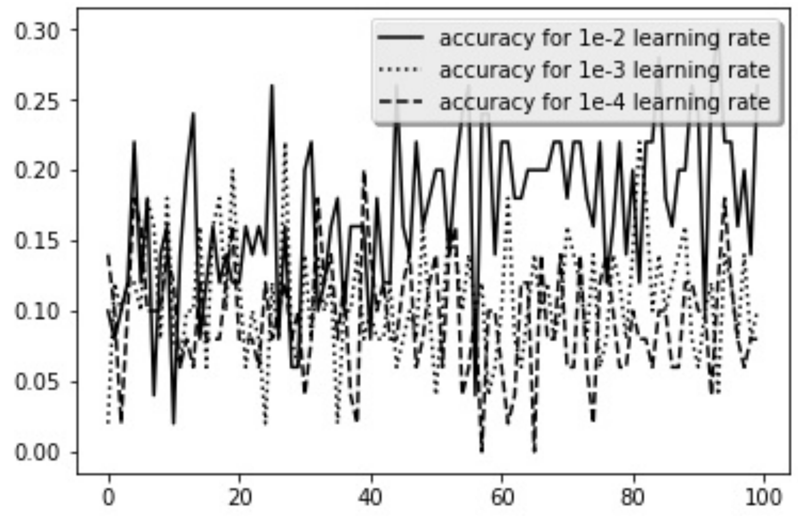
1. Accuracy change with different optimization methods



Obviously, Adagrad method performs the best than the other two with steps growing, faster in decreasing loss. Thus, in the following, I chosen the Adagrad method to train the model.

Plot loss and accuracy changing with respect to different learning rate

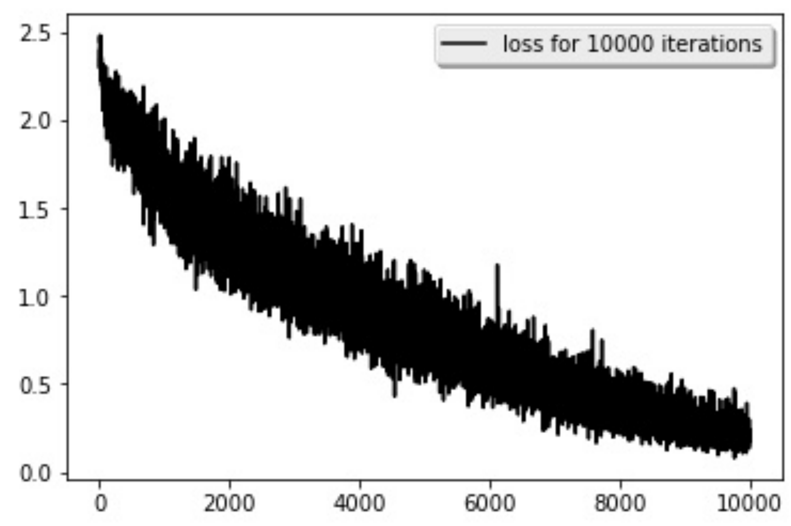




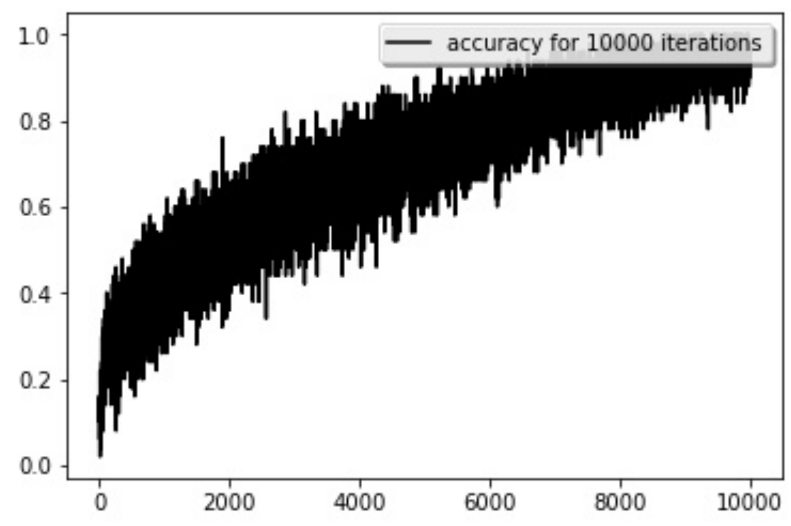
Obivously, higher learning rate could fasten the iteration, however 1e-4 will be too slow to decrease loss and converge. Also, the accuracy increases obviously faster with learnig rate 1e-2. Thus I chose 1e-2 as learning rate to final train. Notice, 1e-1 will be too large to decrease the loss, thus abadoned.

Plot train loss and accuracy changing

The loss iteration over 10000 steps is:



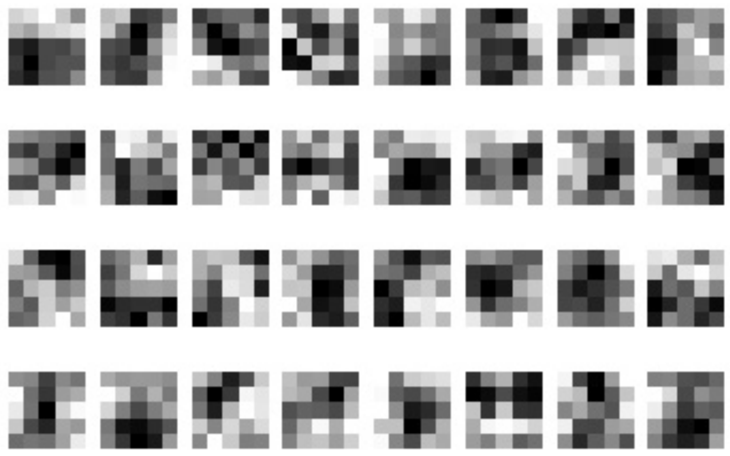
The accuracy iteration over 10000 steps is:



Obviously, the loss is decreasing and train accuracy is increasing. The test accuracy is 0.972727 after 10000 steps.

c) Visualize the Trained Network

The first layer weight is quite interpretable which is looking directly at the raw pixel data. The weight of first layer is visualized as below:



From the visualization, we could see the 32 filter maps focus on different parts of original images. That could help the CNN best capture the structures of images.

For activations on test images, I picked up the feature map with highest activation within. The mean and variance for first layer is -0.026873315 and 0.023991108, while we could see the mean and variance for second layer is -0.15013 and 0.07322. The variance is increasing with ascend layers, which suggests the second activations may have more sparser but focused mapping.

Visualizing and Understanding Convolutional Networks

The paper provides a new way to visualize the Convet, in step furthur, he also shows how to use the visualization to remedy the model.

1. Deconvnet visualization

The author presents a novel way to map these activities back to the input pixel space, showing what input pattern originally caused a given activation in the feature maps, which is called Deconvnet.

Deconvnets were proposed as a way of performing unsupervised learning, just as a probe of an already trained convnet. To examine a convnet, a deconvnet is attached to each of its layers. For each input image in convnet model, the activations except for the just computed part are set to be 0 and sent to deconvet model to recontruct the small region of raw input. The following steps are to be done:

- Unpooling
    - an approximate inverse by recording the locations of the maxima within each pooling region in a set of switch variables.
- Rectification
- Filtering

    - Use transposed versions of learnt filters, applied to rectified maps.

The reconstruction obtained from a single activation thus resembles a small piece of the original input image, with structure weighted according to their contributions to feature activation. Thus they implicitly show which parts of input image are discriminative.

2. Convnet Visualization

- Feature Visualization
    - The projections from each layer show the hierachical nature of features, with different layer features capture different aspect of images.
- Feature Evolution during Training
    - Visualize the strongest activation(across all trainig examples) within a feature map, trying to capture the source of raw image that results in the jump.
- Feature Invariance
    - Translate, rotate and scaled by varying degress to look at the changes in feature vectors at top and bottom layers, related to untransformed features. It turns out the small changes will influence the first layer largely, with little influence on top layers.

3. Architecture Selection

The visualization of trained model could also assist with selecting good architectures. For example, by visualing the first and second layers, he found the first layers filter are a mixed of extremely high and low frequency information. Thus he reduced the 1st layer from 11*11 to 7*7.

4. Occlusion Sensitivity

To examine if the model is truly identifying the location of the object in the image, or just using the surrounding context, the occlusion sensitiviey is presented. It systematically occluding different portions of input image and monitor the output of classfier. The result is the model is localizing the object within the scence and the visualization corresponding to the image structure that stimulates the fearure map. If the object is occluded from the image, the output result will perform badly.

5. Correspondence Analysis

The author suggests that deep models might be implicitly computing the relations between specific object parts in different images. To explore this, he investigate the different between feature vectors before and after occluding the smale part of face in each image. The table results show the model implicitly established some form of correspondence of parts, since the difference for eyes and nose is lower than random occluding.

7. Experiment

The author also presents a experiment for ImageNet 2012, where he explores the architecture of model, the ability of features extraction layers to generalize to other dataset, and analyze the how discriminative the features in each layer of Imagenet-pretrained model.

8. Summary

The author explored large Convent, trained for image classification, in various ways. First is to present a novel way to visualize the activity within the model, which shows many intuitively desirable properties, e.g. compositionally, increasing invariance and class discimination with ascent layers. He also showed how to remedy model by visualizations. Then he demonstrated occlusion experiment to prove the local strutures are captured rather than context scence.

Build and Train an RNN on MNIST

(a). Setup an RNN:

- Number of Node in the hidden layer

nHidden, which represents the number of node in each hidden layer is set to be 128.

- Learning rate

learningRate is set to be 1e-1 to speed up.

- Number of iterations

trainingIters is set to be 60000 steps.

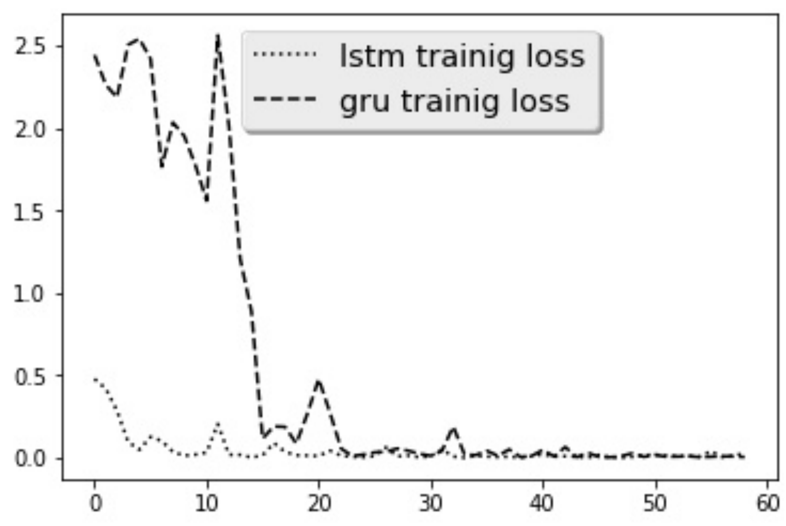
- Cost

I choose the cross-entropy as *cost*, which is the loss.

- Optimizer

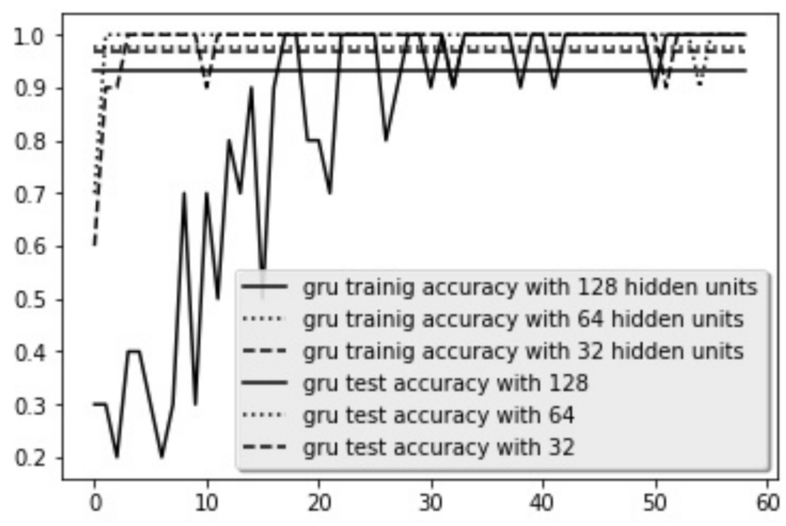
GradientDescentOptimzer, AdagradOptimizer, MomentumOptimzer are selected to compare.

(b). How about using an LSTM or GRU?



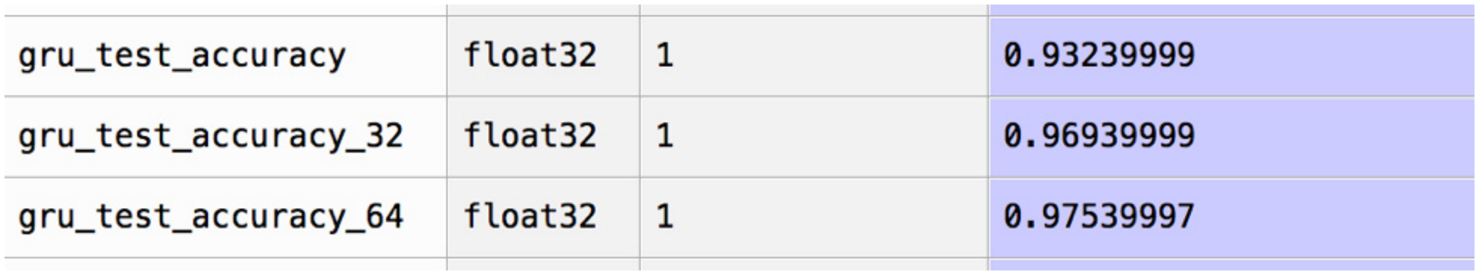
Compare the loss with LSTM and GRU cell as above. The GRU doesn't perform well as LSTM at the begining, but the same well as LSTM. As we know, GRU is a simplied version of LSTM.

Next, let's investigate how the number of hidden units influence the performance.



The more units we used to train the model, the slower accuracy increases. The 128 hidden units may be too large for the model. For this model, 64 units performs quite well, similar speed with 32 units. I suppose 32 is an appropriate number of unit for this model, that is model with unit number less than 32 doesn't perform so well as 32.

The specific value of test accuracy is also attached:



gru_test_accuracy	float32	1	0.93239999
gru_test_accuracy_32	float32	1	0.96939999
gru_test_accuracy_64	float32	1	0.97539997

The test accuracy also proves our analysis. The test accuracy of 128 units, which is 0.9324, much lower than the other two. The test accuracy of 64 units is slightly better than 32 units, similar convergence speed.

(c). Compare against the CNN:

RNN focuses on the dependency between lines of images, while CNN focuses on capturing the same patterns on all the different subfields of the image. CNN allows to process and classify data in a more hierarchical way, like first step, eys, second step, ears. This somewhat mimics the human visual system. Since RNN could handle the dependency, it can be used for sequence of image, video.