This is the 2-layer neural network workbook for ECE 247 Assignment #3

Please follow the notebook linearly to implement a two layer neural network.

Please print out the workbook entirely when completed.

We thank Serena Yeung & Justin Johnson for permission to use code written for the CS 231n class (cs231n.stanford.edu). These are the functions in the cs231n folders and code in the jupyer notebook to preprocess and show the images. The classifiers used are based off of code prepared for CS 231n as well.

The goal of this workbook is to give you experience with training a two layer neural network.

In [1]:
```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

%matplotlib inline
%load_ext autoreload
%autoreload 2

def rel_error(x, y):
    """ returns relative error """
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.ab
```

Toy example

Before loading CIFAR-10, there will be a toy example to test your implementation of the forward and backward pass

In [2]:
```
from nndl.neural_net import TwoLayerNet
```

In [3]:
```
# Create a small net and some toy data to check your implementations.
# Note that we set the random seed for repeatable experiments.

input_size = 4
hidden_size = 10
num_classes = 3
num_inputs = 5

def init_toy_model():
    np.random.seed(0)
    return TwoLayerNet(input_size, hidden_size, num_classes, std=1e-1

def init_toy_data():
    np.random.seed(1)
    X = 10 * np.random.randn(num_inputs, input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

net = init_toy_model()
X, y = init_toy_data()
```

Compute forward pass scores

In [4]:
```
## Implement the forward pass of the neural network.

# Note, there is a statement if y is None: return scores, which is wh
# the following call will calculate the scores.
scores = net.loss(X)
print('Your scores:')
print(scores)
print()
print('correct scores:')
correct_scores = np.asarray([
    [-1.07260209,  0.05083871, -0.87253915],
    [-2.02778743, -0.10832494, -1.52641362],
    [-0.74225908,  0.15259725, -0.39578548],
    [-0.38172726,  0.10835902, -0.17328274],
    [-0.64417314, -0.18886813, -0.41106892]])
print(correct_scores)
print()

# The difference should be very small. We get < 1e-7
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_scores)))
```

```
Your scores:
[[-1.07260209  0.05083871 -0.87253915]
 [-2.02778743 -0.10832494 -1.52641362]
 [-0.74225908  0.15259725 -0.39578548]
 [-0.38172726  0.10835902 -0.17328274]
 [-0.64417314 -0.18886813 -0.41106892]]

correct scores:
[[-1.07260209  0.05083871 -0.87253915]
 [-2.02778743 -0.10832494 -1.52641362]
 [-0.74225908  0.15259725 -0.39578548]
 [-0.38172726  0.10835902 -0.17328274]
 [-0.64417314 -0.18886813 -0.41106892]]

Difference between your scores and correct scores:
3.381231222787662e-08
```

Forward pass loss

In [5]:

```
loss, _ = net.loss(X, y, reg=0.05)
correct_loss = 1.071696123862817

# should be very small, we get < 1e-12
print('Difference between your loss and correct loss:')
print(np.sum(np.abs(loss - correct_loss)))
```

```
Difference between your loss and correct loss:
0.0
```

In [6]:

```
print(loss)
```

```
1.071696123862817
```

Backward pass

Implements the backwards pass of the neural network. Check your gradients with the gradient check utilities provided.

In [7]:

```
from cs231n.gradient_check import eval_numerical_gradient

# Use numeric gradient checking to check your implementation of the b
# If your implementation is correct, the difference between the numer
# analytic gradients should be less than 1e-8 for each of W1, W2, b1,

loss, grads = net.loss(X, y, reg=0.05)

# these should all be less than 1e-8 or so
for param_name in grads:
    f = lambda W: net.loss(X, y, reg=0.05)[0]
    param_grad_num = eval_numerical_gradient(f, net.params[param_name
    print('{} max relative error: {}'.format(param_name, rel_error(pa
W2 max relative error: 2.963225079453O335e-10
b2 max relative error: 1.2482660547101085e-09
W1 max relative error: 1.2832896562471202e-09
b1 max relative error: 3.1726809611748053e-09
```

Training the network

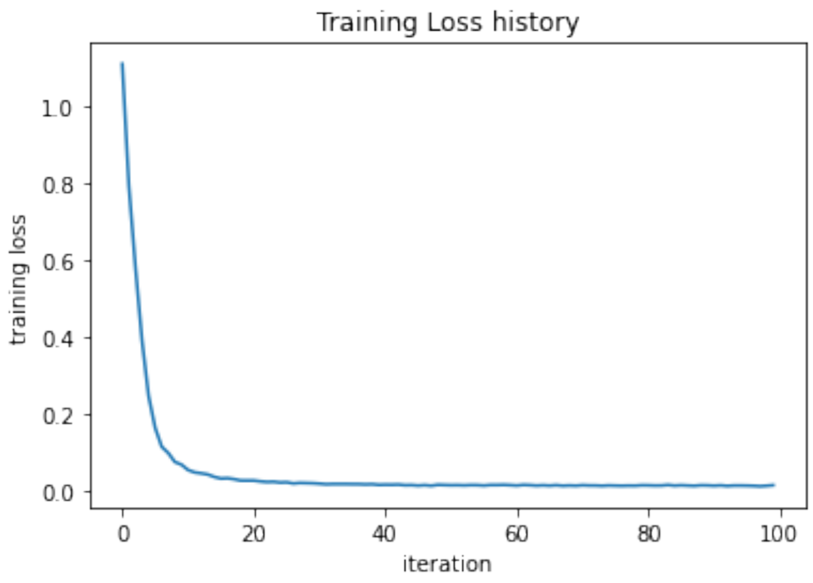
Implement neural_net.train() to train the network via stochastic gradient descent, much like the softmax and SVM.

In [8]:
```
net = init_toy_model()
stats = net.train(X, y, X, y,
            learning_rate=1e-1, reg=5e-6,
            num_iters=100, verbose=False)

print('Final training loss: ', stats['loss_history'][-1])

# plot the loss history
plt.plot(stats['loss_history'])
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')
plt.show()
```

```
Final training loss:  0.014498902952971715
```



Classify CIFAR-10

Do classification on the CIFAR-10 dataset.

In [9]:

```
from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_tes
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to
    it for the two-layer neural net classifier. These are the same st
    we used for the SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = '/Users/Jonathanchang/Downloads/HW3-code/cifar-10-b
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis=0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image

    # Reshape data to rows
    X_train = X_train.reshape(num_training, -1)
    X_val = X_val.reshape(num_validation, -1)
    X_test = X_test.reshape(num_test, -1)

    return X_train, y_train, X_val, y_val, X_test, y_test


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Train data shape:  (49000, 3072)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3072)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3072)
Test labels shape:  (1000,)
```

Running SGD

If your implementation is correct, you should see a validation accuracy of around 28-29%.

In [10…

```
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10
net = TwoLayerNet(input_size, hidden_size, num_classes)

# Train the network
stats = net.train(X_train, y_train, X_val, y_val,
            num_iters=1000, batch_size=200,
            learning_rate=1e-4, learning_rate_decay=0.95,
            reg=0.25, verbose=True)

# Predict on the validation set
val_acc = (net.predict(X_val) == y_val).mean()
print('Validation accuracy: ', val_acc)

# Save this net as the variable subopt_net for later comparison.
subopt_net = net
```

```
iteration 0 / 1000: loss 2.302757518613176
iteration 100 / 1000: loss 2.302120159207236
iteration 200 / 1000: loss 2.2956136007408703
iteration 300 / 1000: loss 2.251825904316413
iteration 400 / 1000: loss 2.188995235046776
iteration 500 / 1000: loss 2.1162527791897747
iteration 600 / 1000: loss 2.064670827698217
iteration 700 / 1000: loss 1.9901688623083942
iteration 800 / 1000: loss 2.002827640124685
iteration 900 / 1000: loss 1.9465176817856495
Validation accuracy:  0.283
```

Questions:

The training accuracy isn't great.

(1) What are some of the reasons why this is the case? Take the following cell to do some analyses and then report your answers in the cell following the one below.

(2) How should you fix the problems you identified in (1)?

In [11…

```
stats['train_acc_history']
```

Out[11…

```
[0.095, 0.15, 0.25, 0.25, 0.315]
```

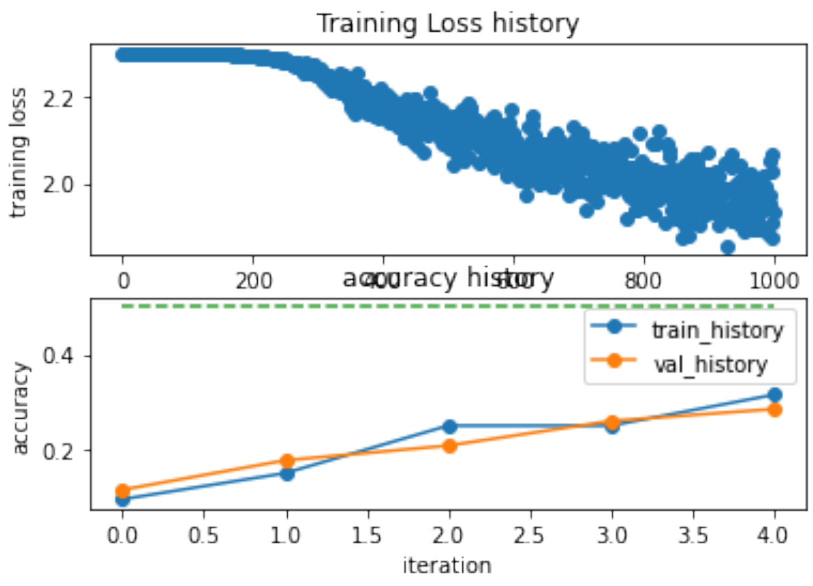
In [12…

```
# ================================================================ #
# YOUR CODE HERE:
#   Do some debugging to gain some insight into why the optimization
#   isn't great.
# ================================================================ #

# Plot the loss function and train / validation accuracies

plt.subplot(2,1,1)
plt.plot(stats['loss_history'], 'o')
plt.xlabel('iteration')
plt.ylabel('training loss')
plt.title('Training Loss history')


plt.subplot(2,1,2)
plt.plot(stats['train_acc_history'],'-o', label='train_history')
plt.plot(stats['val_acc_history'],'-o', label='val_history')
plt.plot([0.5]* len(stats['val_acc_history']),'--')
plt.xlabel('iteration')
plt.ylabel('accuracy')
plt.title('accuracy history')
plt.legend(loc = 'upper right')
plt.show()
# ================================================================ #
# END YOUR CODE HERE
# ================================================================ #
```



Answers:

(1) We can see the training isn't finished. The training accuracy wasn't saturate, so there is still room to improve.

(2) We can increase the iteration number until we see saturation in training loss and validation accuracy. Learning rate changes may can improve such training process.

Optimize the neural network

Use the following part of the Jupyter notebook to optimize your hyperparameters on the validation set. Store your nets as best_net.

In [15…

```
best_net = None # store the best model into this

# ================================================================= #
# YOUR CODE HERE:
#   Optimize over your hyperparameters to arrive at the best neural
#   network.  You should be able to get over 50% validation accuracy.
#   For this part of the notebook, we will give credit based on the
#   accuracy you get.  Your score on this question will be multiplied
#      min(floor((X - 28%)) / %22, 1)
#   where if you get 50% or higher validation accuracy, you get full
#   points.
#
#   Note, you need to use the same network structure (keep hidden_siz
# ================================================================= #
input_size = 32 * 32 * 3
hidden_size = 50
num_classes = 10

# Train the network
best_acc = 0
for batch_size in [200,400,600,800,1000]:
    for num_iters in [1000,3000,5000,7000,9000]:
        for learning_rate in [1e-3, 1e-4, 5e-4, 1e-5]:
            net_temp = TwoLayerNet(input_size, hidden_size, num_class
            stats = net_temp.train(X_train, y_train, X_val, y_val,
                num_iters=num_iters, batch_size=batch_size,
                learning_rate=learning_rate, learning_rate_decay= 0.9
                reg=0.0015, verbose=False)

            # Predict on the validation set
            val_acc = (net_temp.predict(X_val) == y_val).mean()
            print('Validation accuracy (lr=', learning_rate, 'iter=',
            if val_acc > best_acc:
                best_acc = val_acc
                best_lr = learning_rate
                best_iter = num_iters
                best_size = batch_size


net  = TwoLayerNet(input_size, hidden_size, num_classes)
stats = net.train(X_train, y_train, X_val, y_val,
                num_iters=best_iter, batch_size=best_size,
                learning_rate=best_lr, learning_rate_decay= 0.95,
                reg=0.0015, verbose=False)


print('Best accuracy (lr=', best_lr, 'iter=', best_iter, 'batch_size=
# ================================================================= #
# END YOUR CODE HERE
# ================================================================= #
best_net = net
```

```
Validation accuracy (lr= 0.001 iter= 1000 batch_size= 200 ) 0.472
Validation accuracy (lr= 0.0001 iter= 1000 batch_size= 200 ) 0.287
Validation accuracy (lr= 0.0005 iter= 1000 batch_size= 200 ) 0.452
Validation accuracy (lr= 1e-05 iter= 1000 batch_size= 200 ) 0.228
Validation accuracy (lr= 0.001 iter= 3000 batch_size= 200 ) 0.491
Validation accuracy (lr= 0.0001 iter= 3000 batch_size= 200 ) 0.379
Validation accuracy (lr= 0.0005 iter= 3000 batch_size= 200 ) 0.506
Validation accuracy (lr= 1e-05 iter= 3000 batch_size= 200 ) 0.181
Validation accuracy (lr= 0.001 iter= 5000 batch_size= 200 ) 0.514
Validation accuracy (lr= 0.0001 iter= 5000 batch_size= 200 ) 0.425
Validation accuracy (lr= 0.0005 iter= 5000 batch_size= 200 ) 0.508
Validation accuracy (lr= 1e-05 iter= 5000 batch_size= 200 ) 0.185
Validation accuracy (lr= 0.001 iter= 7000 batch_size= 200 ) 0.512
Validation accuracy (lr= 0.0001 iter= 7000 batch_size= 200 ) 0.44
Validation accuracy (lr= 0.0005 iter= 7000 batch_size= 200 ) 0.505
Validation accuracy (lr= 1e-05 iter= 7000 batch_size= 200 ) 0.2
Validation accuracy (lr= 0.001 iter= 9000 batch_size= 200 ) 0.499
Validation accuracy (lr= 0.0001 iter= 9000 batch_size= 200 ) 0.448
Validation accuracy (lr= 0.0005 iter= 9000 batch_size= 200 ) 0.518
Validation accuracy (lr= 1e-05 iter= 9000 batch_size= 200 ) 0.207
Validation accuracy (lr= 0.001 iter= 1000 batch_size= 400 ) 0.481
Validation accuracy (lr= 0.0001 iter= 1000 batch_size= 400 ) 0.281
Validation accuracy (lr= 0.0005 iter= 1000 batch_size= 400 ) 0.46
Validation accuracy (lr= 1e-05 iter= 1000 batch_size= 400 ) 0.216
Validation accuracy (lr= 0.001 iter= 3000 batch_size= 400 ) 0.526
Validation accuracy (lr= 0.0001 iter= 3000 batch_size= 400 ) 0.393
Validation accuracy (lr= 0.0005 iter= 3000 batch_size= 400 ) 0.503
Validation accuracy (lr= 1e-05 iter= 3000 batch_size= 400 ) 0.163
Validation accuracy (lr= 0.001 iter= 5000 batch_size= 400 ) 0.513
Validation accuracy (lr= 0.0001 iter= 5000 batch_size= 400 ) 0.427
Validation accuracy (lr= 0.0005 iter= 5000 batch_size= 400 ) 0.511
Validation accuracy (lr= 1e-05 iter= 5000 batch_size= 400 ) 0.196
Validation accuracy (lr= 0.001 iter= 7000 batch_size= 400 ) 0.51
Validation accuracy (lr= 0.0001 iter= 7000 batch_size= 400 ) 0.453
Validation accuracy (lr= 0.0005 iter= 7000 batch_size= 400 ) 0.509
Validation accuracy (lr= 1e-05 iter= 7000 batch_size= 400 ) 0.201
Validation accuracy (lr= 0.001 iter= 9000 batch_size= 400 ) 0.52
Validation accuracy (lr= 0.0001 iter= 9000 batch_size= 400 ) 0.449
Validation accuracy (lr= 0.0005 iter= 9000 batch_size= 400 ) 0.51
Validation accuracy (lr= 1e-05 iter= 9000 batch_size= 400 ) 0.199
Validation accuracy (lr= 0.001 iter= 1000 batch_size= 600 ) 0.462
Validation accuracy (lr= 0.0001 iter= 1000 batch_size= 600 ) 0.294
Validation accuracy (lr= 0.0005 iter= 1000 batch_size= 600 ) 0.46
Validation accuracy (lr= 1e-05 iter= 1000 batch_size= 600 ) 0.198
Validation accuracy (lr= 0.001 iter= 3000 batch_size= 600 ) 0.495
Validation accuracy (lr= 0.0001 iter= 3000 batch_size= 600 ) 0.421
Validation accuracy (lr= 0.0005 iter= 3000 batch_size= 600 ) 0.497
Validation accuracy (lr= 1e-05 iter= 3000 batch_size= 600 ) 0.162
Validation accuracy (lr= 0.001 iter= 5000 batch_size= 600 ) 0.493
Validation accuracy (lr= 0.0001 iter= 5000 batch_size= 600 ) 0.466
Validation accuracy (lr= 0.0005 iter= 5000 batch_size= 600 ) 0.51
Validation accuracy (lr= 1e-05 iter= 5000 batch_size= 600 ) 0.232
Validation accuracy (lr= 0.001 iter= 7000 batch_size= 600 ) 0.48
Validation accuracy (lr= 0.0001 iter= 7000 batch_size= 600 ) 0.469
Validation accuracy (lr= 0.0005 iter= 7000 batch_size= 600 ) 0.519
Validation accuracy (lr= 1e-05 iter= 7000 batch_size= 600 ) 0.25
Validation accuracy (lr= 0.001 iter= 9000 batch_size= 600 ) 0.49
Validation accuracy (lr= 0.0001 iter= 9000 batch_size= 600 ) 0.481
Validation accuracy (lr= 0.0005 iter= 9000 batch_size= 600 ) 0.507
Validation accuracy (lr= 1e-05 iter= 9000 batch_size= 600 ) 0.287
Validation accuracy (lr= 0.001 iter= 1000 batch_size= 800 ) 0.476
Validation accuracy (lr= 0.0001 iter= 1000 batch_size= 800 ) 0.285
Validation accuracy (lr= 0.0005 iter= 1000 batch_size= 800 ) 0.459
Validation accuracy (lr= 1e-05 iter= 1000 batch_size= 800 ) 0.25
Validation accuracy (lr= 0.001 iter= 3000 batch_size= 800 ) 0.53
Validation accuracy (lr= 0.0001 iter= 3000 batch_size= 800 ) 0.393
Validation accuracy (lr= 0.0005 iter= 3000 batch_size= 800 ) 0.499
Validation accuracy (lr= 1e-05 iter= 3000 batch_size= 800 ) 0.205
Validation accuracy (lr= 0.001 iter= 5000 batch_size= 800 ) 0.489
Validation accuracy (lr= 0.0001 iter= 5000 batch_size= 800 ) 0.42
Validation accuracy (lr= 0.0005 iter= 5000 batch_size= 800 ) 0.511
Validation accuracy (lr= 1e-05 iter= 5000 batch_size= 800 ) 0.18
Validation accuracy (lr= 0.001 iter= 7000 batch_size= 800 ) 0.504
Validation accuracy (lr= 0.0001 iter= 7000 batch_size= 800 ) 0.438
Validation accuracy (lr= 0.0005 iter= 7000 batch_size= 800 ) 0.53
Validation accuracy (lr= 1e-05 iter= 7000 batch_size= 800 ) 0.201
Validation accuracy (lr= 0.001 iter= 9000 batch_size= 800 ) 0.507
Validation accuracy (lr= 0.0001 iter= 9000 batch_size= 800 ) 0.45
Validation accuracy (lr= 0.0005 iter= 9000 batch_size= 800 ) 0.527
Validation accuracy (lr= 1e-05 iter= 9000 batch_size= 800 ) 0.206
Validation accuracy (lr= 0.001 iter= 1000 batch_size= 1000 ) 0.47
Validation accuracy (lr= 0.0001 iter= 1000 batch_size= 1000 ) 0.251
Validation accuracy (lr= 0.0005 iter= 1000 batch_size= 1000 ) 0.431
Validation accuracy (lr= 1e-05 iter= 1000 batch_size= 1000 ) 0.177
Validation accuracy (lr= 0.001 iter= 3000 batch_size= 1000 ) 0.5
Validation accuracy (lr= 0.0001 iter= 3000 batch_size= 1000 ) 0.289
Validation accuracy (lr= 0.0005 iter= 3000 batch_size= 1000 ) 0.452
Validation accuracy (lr= 1e-05 iter= 3000 batch_size= 1000 ) 0.177
Validation accuracy (lr= 0.001 iter= 5000 batch_size= 1000 ) 0.499
Validation accuracy (lr= 0.0001 iter= 5000 batch_size= 1000 ) 0.284
Validation accuracy (lr= 0.0005 iter= 5000 batch_size= 1000 ) 0.451
Validation accuracy (lr= 1e-05 iter= 5000 batch_size= 1000 ) 0.174
Validation accuracy (lr= 0.001 iter= 7000 batch_size= 1000 ) 0.483
Validation accuracy (lr= 0.0001 iter= 7000 batch_size= 1000 ) 0.288
Validation accuracy (lr= 0.0005 iter= 7000 batch_size= 1000 ) 0.456
Validation accuracy (lr= 1e-05 iter= 7000 batch_size= 1000 ) 0.207
Validation accuracy (lr= 0.001 iter= 9000 batch_size= 1000 ) 0.485
Validation accuracy (lr= 0.0001 iter= 9000 batch_size= 1000 ) 0.287
Validation accuracy (lr= 0.0005 iter= 9000 batch_size= 1000 ) 0.464
Validation accuracy (lr= 1e-05 iter= 9000 batch_size= 1000 ) 0.216
Best accuracy (lr= 0.001 iter= 3000 batch_size= 800 ) 0.53
```

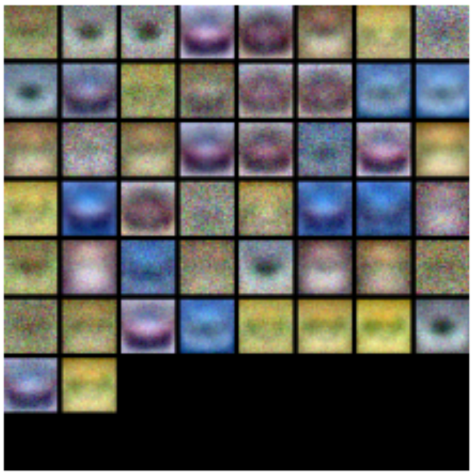
In [16…

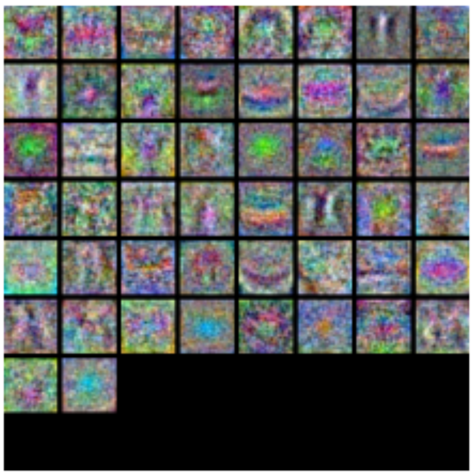
```python
from cs231n.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(net):
    W1 = np.array(net.params['W1'])
    W1 = W1.T.reshape(32, 32, 3, -1).transpose(3, 0, 1, 2)
    plt.imshow(visualize_grid(W1, padding=3).astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(subopt_net)
show_net_weights(best_net)
```





Question:

(1) What differences do you see in the weights between the suboptimal net and the best net you arrived at?

Answer:

(1) The best situation have a sharper image and more colorful compare to unoptimize conditions. Because best net have the better validation accuracy, which means fiture are identified better.

Evaluate on test set

In [17…

```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.527
```

In [ ]: