

# two\_layer\_net

February 10, 2020

## 1 Implementing a Neural Network

In this exercise we will develop a neural network with fully-connected layers to perform classification, and test it out on the CIFAR-10 dataset.

```
[1]: # A bit of setup

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading external modules
# see http://stackoverflow.com/questions/1907993/
# → autoreload-of-modules-in-ipython
%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.abs(y))))
```

The neural network parameters will be stored in a dictionary (`model` below), where the keys are the parameter names and the values are numpy arrays. Below, we initialize toy data and a toy model that we will use to verify your implementations.

```
[2]: # Create some toy data to check your implementations

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

def init_toy_model():
    model = {}
```

```

    model['W1'] = np.linspace(-0.2, 0.6, num=input_size*hidden_size).
    ↪reshape(input_size, hidden_size)
    model['b1'] = np.linspace(-0.3, 0.7, num=hidden_size)
    model['W2'] = np.linspace(-0.4, 0.1, num=hidden_size*num_classes).
    ↪reshape(hidden_size, num_classes)
    model['b2'] = np.linspace(-0.5, 0.9, num=num_classes)
    return model

def init_toy_data():
    X = np.linspace(-0.2, 0.5, num=num_inputs*input_size).reshape(num_inputs,
    ↪input_size)
    y = np.array([0, 1, 2, 2, 1])
    return X, y

model = init_toy_model()
X, y = init_toy_data()

```

## 2 Forward pass: compute scores

Open the file `cs231n/classifiers/neural_net.py` and look at the function `two_layer_net`. This function is very similar to the loss functions you have written for the Softmax exercise in HW0: It takes the data and weights and computes the class scores, the loss, and the gradients on the parameters.

Implement the first part of the forward pass which uses the weights and biases to compute the scores for all inputs.

```

[12]: from cs231n.classifiers.neural_net import two_layer_net

scores = two_layer_net(X, model)
print(scores)
correct_scores = [[-0.5328368, 0.20031504, 0.93346689],
                  [-0.59412164, 0.15498488, 0.9040914 ],
                  [-0.67658362, 0.08978957, 0.85616275],
                  [-0.77092643, 0.01339997, 0.79772637],
                  [-0.89110401, -0.08754544, 0.71601312]]

# the difference should be very small. We get 3e-8
print('Difference between your scores and correct scores:')
print(np.sum(np.abs(scores - correct_scores)))

```

```

[[-0.5328368  0.20031504  0.93346689]
 [-0.59412164  0.15498488  0.9040914 ]
 [-0.67658362  0.08978957  0.85616275]
 [-0.77092643  0.01339997  0.79772637]
 [-0.89110401 -0.08754544  0.71601312]]

```

Difference between your scores and correct scores:  
3.848682278081994e-08

### 3 Forward pass: compute loss

In the same function, implement the second part that computes the data and regularization loss.

```
[13]: reg = 0.1
      loss, _ = two_layer_net(X, model, y, reg)
      correct_loss = 1.38191946092

      # should be very small, we get 5e-12
      print('Difference between your loss and correct loss:')
      print(np.sum(np.abs(loss - correct_loss)))
```

Difference between your loss and correct loss:  
4.6769255135359344e-12

### 4 Backward pass

Implement the rest of the function. This will compute the gradient of the loss with respect to the variables  $W1$ ,  $b1$ ,  $W2$ , and  $b2$ . Now that you (hopefully!) have a correctly implemented forward pass, you can debug your backward pass using a numeric gradient check:

```
[14]: from cs231n.gradient_check import eval_numerical_gradient

      # Use numeric gradient checking to check your implementation of the backward
      # pass.
      # If your implementation is correct, the difference between the numeric and
      # analytic gradients should be less than 1e-8 for each of W1, W2, b1, and b2.

      loss, grads = two_layer_net(X, model, y, reg)

      # these should all be less than 1e-8 or so
      for param_name in grads:
          param_grad_num = eval_numerical_gradient(lambda W: two_layer_net(X, model, y,
          reg)[0], model[param_name], verbose=False)
          print('%s max relative error: %e' % (param_name, rel_error(param_grad_num,
          grads[param_name])))
```

W2 max relative error: 9.913918e-10  
b2 max relative error: 8.190173e-11  
W1 max relative error: 4.426512e-09  
b1 max relative error: 5.435431e-08

## 5 Train the network

To train the network we will use SGD with Momentum. Last assignment you implemented vanilla SGD. You will now implement the momentum update and the RMSProp update. Open the file `classifier_trainer.py` and familiarize yourself with the `ClassifierTrainer` class. It performs optimization given an arbitrary cost function data, and model. By default it uses vanilla SGD, which we have already implemented for you. First, run the optimization below using Vanilla SGD:

```
[17]: from cs231n.classifier_trainer import ClassifierTrainer

model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient
↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                              model, two_layer_net,
                                              reg=0.001,
                                              learning_rate=1e-1, momentum=0.0,
↪learning_rate_decay=1,
                                              update='sgd', sample_batches=False,
                                              num_epochs=100,
                                              verbose=False)
print('Final loss with vanilla SGD: %f' % (loss_history[-1], ))

starting iteration  0
starting iteration 10
starting iteration 20
starting iteration 30
starting iteration 40
starting iteration 50
starting iteration 60
starting iteration 70
starting iteration 80
starting iteration 90
Final loss with vanilla SGD: 0.940686
```

Now fill in the **momentum update** in the first missing code block inside the `train` function, and run the same optimization as above but with the momentum update. You should see a much better result in the final obtained loss:

```
[18]: model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient
↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
                                              model, two_layer_net,
```

```

reg=0.001,
learning_rate=1e-1, momentum=0.9,
↪learning_rate_decay=1,
↪sample_batches=False,

update='momentum',

num_epochs=100,
verbose=False)

correct_loss = 0.494394
print('Final loss with momentum SGD: %f. We get: %f' % (loss_history[-1],
↪correct_loss))

```

```

starting iteration 0
starting iteration 10
starting iteration 20
starting iteration 30
starting iteration 40
starting iteration 50
starting iteration 60
starting iteration 70
starting iteration 80
starting iteration 90
Final loss with momentum SGD: 0.494394. We get: 0.494394

```

The **RMSProp** update step is given as follows:

```

cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-8)

```

Here, `decay_rate` is a hyperparameter and typical values are [0.9, 0.99, 0.999].

Implement the **RMSProp** update rule inside the `train` function and rerun the optimization:

```

[19]: model = init_toy_model()
trainer = ClassifierTrainer()
# call the trainer to optimize the loss
# Notice that we're using sample_batches=False, so we're performing Gradient
↪Descent (no sampled batches of data)
best_model, loss_history, _, _ = trainer.train(X, y, X, y,
reg=0.001,
learning_rate=1e-1, momentum=0.9,
↪learning_rate_decay=1,
↪sample_batches=False,

update='rmsprop',

num_epochs=100,
verbose=False)

correct_loss = 0.439368
print('Final loss with RMSProp: %f. We get: %f' % (loss_history[-1],
↪correct_loss))

```

```
starting iteration 0
starting iteration 10
starting iteration 20
starting iteration 30
starting iteration 40
starting iteration 50
starting iteration 60
starting iteration 70
starting iteration 80
starting iteration 90
Final loss with RMSProp: 0.439368. We get: 0.439368
```

## 6 Load the data

Now that you have implemented a two-layer network that passes gradient checks, it's time to load up our favorite CIFAR-10 data so we can use it to train a classifier.

```
[20]: from cs231n.data_utils import load_CIFAR10

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the two-layer neural net classifier.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

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

```

## 7 Train a network

To train our network we will use SGD with momentum. In addition, we will adjust the learning rate with an exponential learning rate schedule as optimization proceeds; after each epoch, we will reduce the learning rate by multiplying it by a decay rate.

```

[21]: from cs231n.classifiers.neural_net import init_two_layer_model

model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number
↳ of classes
trainer = ClassifierTrainer()
best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train,
↳ X_val, y_val,

                                model, two_layer_net,
                                num_epochs=5, reg=1.0,
                                momentum=0.9, learning_rate_decay
↳ = 0.95,

                                learning_rate=1e-5, verbose=True)

```

```

starting iteration 0
Finished epoch 0 / 5: cost 2.302593, train: 0.094000, val 0.108000, lr
1.000000e-05
starting iteration 10

```

starting iteration 20  
starting iteration 30  
starting iteration 40  
starting iteration 50  
starting iteration 60  
starting iteration 70  
starting iteration 80  
starting iteration 90  
starting iteration 100  
starting iteration 110  
starting iteration 120  
starting iteration 130  
starting iteration 140  
starting iteration 150  
starting iteration 160  
starting iteration 170  
starting iteration 180  
starting iteration 190  
starting iteration 200  
starting iteration 210  
starting iteration 220  
starting iteration 230  
starting iteration 240  
starting iteration 250  
starting iteration 260  
starting iteration 270  
starting iteration 280  
starting iteration 290  
starting iteration 300  
starting iteration 310  
starting iteration 320  
starting iteration 330  
starting iteration 340  
starting iteration 350  
starting iteration 360  
starting iteration 370  
starting iteration 380  
starting iteration 390  
starting iteration 400  
starting iteration 410  
starting iteration 420  
starting iteration 430  
starting iteration 440  
starting iteration 450  
starting iteration 460  
starting iteration 470  
starting iteration 480  
Finished epoch 1 / 5: cost 2.285003, train: 0.193000, val 0.173000, lr



9.500000e-06

starting iteration 490  
starting iteration 500  
starting iteration 510  
starting iteration 520  
starting iteration 530  
starting iteration 540  
starting iteration 550  
starting iteration 560  
starting iteration 570  
starting iteration 580  
starting iteration 590  
starting iteration 600  
starting iteration 610  
starting iteration 620  
starting iteration 630  
starting iteration 640  
starting iteration 650  
starting iteration 660  
starting iteration 670  
starting iteration 680  
starting iteration 690  
starting iteration 700  
starting iteration 710  
starting iteration 720  
starting iteration 730  
starting iteration 740  
starting iteration 750  
starting iteration 760  
starting iteration 770  
starting iteration 780  
starting iteration 790  
starting iteration 800  
starting iteration 810  
starting iteration 820  
starting iteration 830  
starting iteration 840  
starting iteration 850  
starting iteration 860  
starting iteration 870  
starting iteration 880  
starting iteration 890  
starting iteration 900  
starting iteration 910  
starting iteration 920  
starting iteration 930  
starting iteration 940  
starting iteration 950

starting iteration 960  
starting iteration 970  
Finished epoch 2 / 5: cost 2.154377, train: 0.256000, val 0.240000, lr  
9.025000e-06  
starting iteration 980  
starting iteration 990  
starting iteration 1000  
starting iteration 1010  
starting iteration 1020  
starting iteration 1030  
starting iteration 1040  
starting iteration 1050  
starting iteration 1060  
starting iteration 1070  
starting iteration 1080  
starting iteration 1090  
starting iteration 1100  
starting iteration 1110  
starting iteration 1120  
starting iteration 1130  
starting iteration 1140  
starting iteration 1150  
starting iteration 1160  
starting iteration 1170  
starting iteration 1180  
starting iteration 1190  
starting iteration 1200  
starting iteration 1210  
starting iteration 1220  
starting iteration 1230  
starting iteration 1240  
starting iteration 1250  
starting iteration 1260  
starting iteration 1270  
starting iteration 1280  
starting iteration 1290  
starting iteration 1300  
starting iteration 1310  
starting iteration 1320  
starting iteration 1330  
starting iteration 1340  
starting iteration 1350  
starting iteration 1360  
starting iteration 1370  
starting iteration 1380  
starting iteration 1390  
starting iteration 1400  
starting iteration 1410

starting iteration 1420  
starting iteration 1430  
starting iteration 1440  
starting iteration 1450  
starting iteration 1460  
Finished epoch 3 / 5: cost 1.928161, train: 0.264000, val 0.282000, lr  
8.573750e-06  
starting iteration 1470  
starting iteration 1480  
starting iteration 1490  
starting iteration 1500  
starting iteration 1510  
starting iteration 1520  
starting iteration 1530  
starting iteration 1540  
starting iteration 1550  
starting iteration 1560  
starting iteration 1570  
starting iteration 1580  
starting iteration 1590  
starting iteration 1600  
starting iteration 1610  
starting iteration 1620  
starting iteration 1630  
starting iteration 1640  
starting iteration 1650  
starting iteration 1660  
starting iteration 1670  
starting iteration 1680  
starting iteration 1690  
starting iteration 1700  
starting iteration 1710  
starting iteration 1720  
starting iteration 1730  
starting iteration 1740  
starting iteration 1750  
starting iteration 1760  
starting iteration 1770  
starting iteration 1780  
starting iteration 1790  
starting iteration 1800  
starting iteration 1810  
starting iteration 1820  
starting iteration 1830  
starting iteration 1840  
starting iteration 1850  
starting iteration 1860  
starting iteration 1870

starting iteration 1880  
starting iteration 1890  
starting iteration 1900  
starting iteration 1910  
starting iteration 1920  
starting iteration 1930  
starting iteration 1940  
starting iteration 1950  
Finished epoch 4 / 5: cost 1.799440, train: 0.329000, val 0.331000, lr  
8.145063e-06  
starting iteration 1960  
starting iteration 1970  
starting iteration 1980  
starting iteration 1990  
starting iteration 2000  
starting iteration 2010  
starting iteration 2020  
starting iteration 2030  
starting iteration 2040  
starting iteration 2050  
starting iteration 2060  
starting iteration 2070  
starting iteration 2080  
starting iteration 2090  
starting iteration 2100  
starting iteration 2110  
starting iteration 2120  
starting iteration 2130  
starting iteration 2140  
starting iteration 2150  
starting iteration 2160  
starting iteration 2170  
starting iteration 2180  
starting iteration 2190  
starting iteration 2200  
starting iteration 2210  
starting iteration 2220  
starting iteration 2230  
starting iteration 2240  
starting iteration 2250  
starting iteration 2260  
starting iteration 2270  
starting iteration 2280  
starting iteration 2290  
starting iteration 2300  
starting iteration 2310  
starting iteration 2320  
starting iteration 2330

```
starting iteration 2340
starting iteration 2350
starting iteration 2360
starting iteration 2370
starting iteration 2380
starting iteration 2390
starting iteration 2400
starting iteration 2410
starting iteration 2420
starting iteration 2430
starting iteration 2440
Finished epoch 5 / 5: cost 1.724871, train: 0.351000, val 0.356000, lr
7.737809e-06
finished optimization. best validation accuracy: 0.356000
```

## 8 Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.37 on the validation set. This isn't very good.

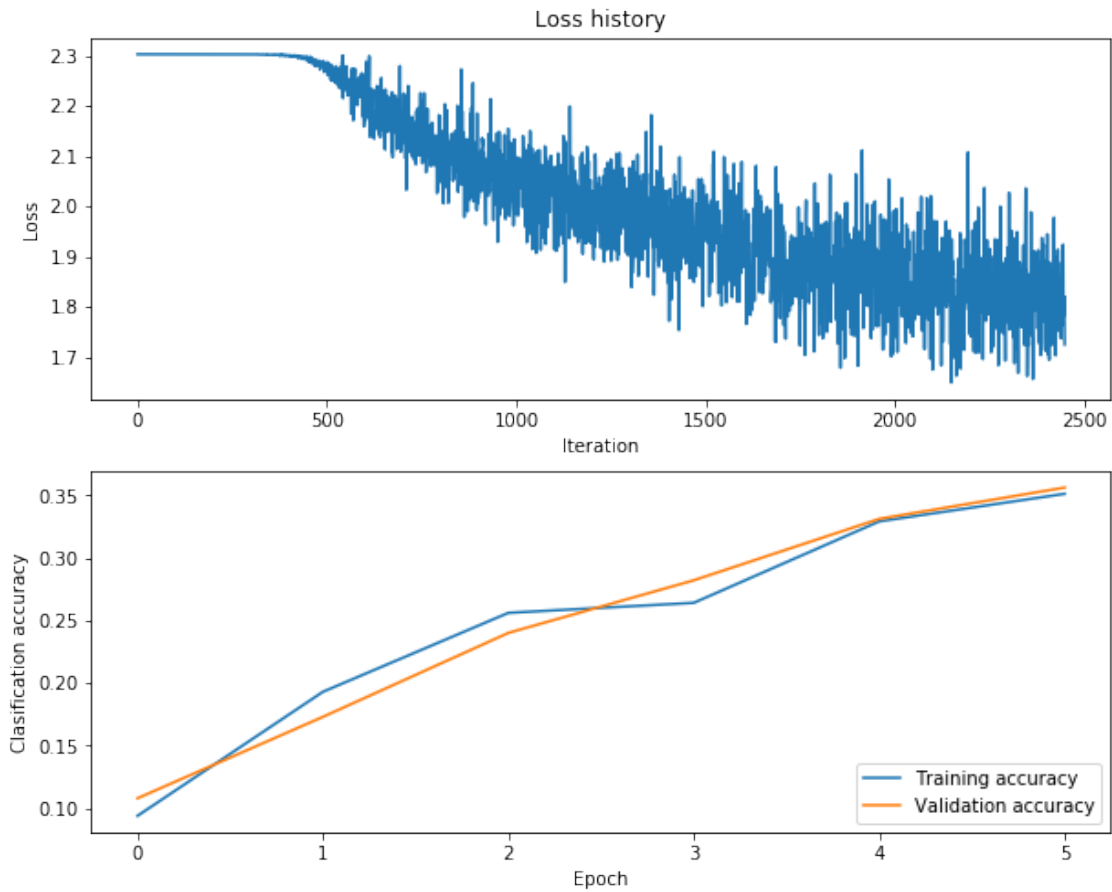
One strategy for getting insight into what's wrong is to plot the loss function and the accuracies on the training and validation sets during optimization.

Another strategy is to visualize the weights that were learned in the first layer of the network. In most neural networks trained on visual data, the first layer weights typically show some visible structure when visualized.

```
[22]: # Plot the loss function and train / validation accuracies
plt.subplot(2, 1, 1)
plt.plot(loss_history)
plt.title('Loss history')
plt.xlabel('Iteration')
plt.ylabel('Loss')

plt.subplot(2, 1, 2)
plt.plot(train_acc)
plt.plot(val_acc)
plt.legend(['Training accuracy', 'Validation accuracy'], loc='lower right')
plt.xlabel('Epoch')
plt.ylabel('Clasification accuracy')
```

```
[22]: Text(0, 0.5, 'Clasification accuracy')
```

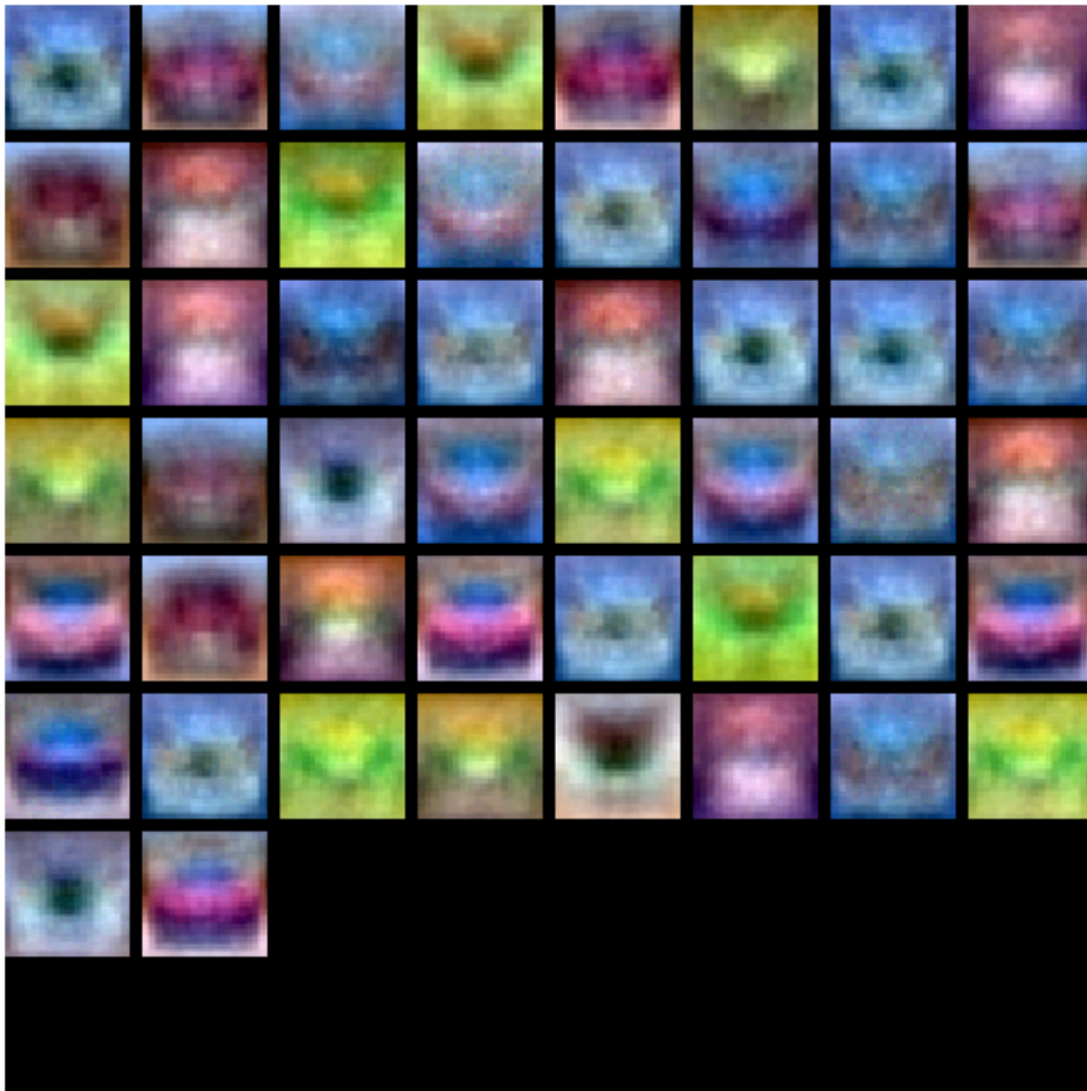


```
[23]: from cs231n.vis_utils import visualize_grid

# Visualize the weights of the network

def show_net_weights(model):
    plt.imshow(visualize_grid(model['W1'].T.reshape(-1, 32, 32, 3), padding=3).
        ↪astype('uint8'))
    plt.gca().axis('off')
    plt.show()

show_net_weights(model)
```



## 9 Tune your hyperparameters

**What's wrong?.** Looking at the visualizations above, we see that the loss is decreasing more or less linearly, which seems to suggest that the learning rate may be too low. Moreover, there is no gap between the training and validation accuracy, suggesting that the model we used has low capacity, and that we should increase its size. On the other hand, with a very large model we would expect to see more overfitting, which would manifest itself as a very large gap between the training and validation accuracy.

**Tuning.** Tuning the hyperparameters and developing intuition for how they affect the final performance is a large part of using Neural Networks, so we want you to get a lot of practice. Below, you should experiment with different values of the various hyperparameters, including hidden layer

size, learning rate, number of training epochs, and regularization strength. You might also consider tuning the momentum and learning rate decay parameters, but you should be able to get good performance using the default values.

**Approximate results.** You should be able to achieve a classification accuracy of greater than 50% on the validation set. Our best network gets over 56% on the validation set.

**Experiment:** Your goal in this exercise is to get as good of a result on CIFAR-10 as you can, with a fully-connected Neural Network. For every 1% above 56% on the Test set we will award you with one extra bonus point. Feel free to implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

```
[28]: best_model = None # store the best model into this

#####
# TODO: Tune hyperparameters using the validation set. Store your best trained
# model in best_model.
#
# To help debug your network, it may help to use visualizations similar to the
# ones we used above; these visualizations will have significant qualitative
# differences from the ones we saw above for the poorly tuned network.
#
# Tweaking hyperparameters by hand can be fun, but you might find it useful to
# write code to sweep through possible combinations of hyperparameters
# automatically like we did on the previous assignment.
#####
# input size, hidden size, number of classes
model = init_two_layer_model(32*32*3, 1000, 10)
trainer = ClassifierTrainer()
best_model, loss_history, train_acc, val_acc = trainer.train(X_train, y_train,
                                                             X_val, y_val,
                                                             model, two_layer_net,
                                                             num_epochs=10, reg=0.0001,
                                                             momentum=0.9,
                                                             learning_rate_decay=0.7,
                                                             learning_rate=1e-4, verbose=True)
#####
```



```
# END OF YOUR CODE ⏏  
↪ #  
#####
```

```
starting iteration 0  
Finished epoch 0 / 10: cost 2.302586, train: 0.137000, val 0.105000, lr  
1.000000e-04  
starting iteration 10  
starting iteration 20  
starting iteration 30  
starting iteration 40  
starting iteration 50  
starting iteration 60  
starting iteration 70  
starting iteration 80  
starting iteration 90  
starting iteration 100  
starting iteration 110  
starting iteration 120  
starting iteration 130  
starting iteration 140  
starting iteration 150  
starting iteration 160  
starting iteration 170  
starting iteration 180  
starting iteration 190  
starting iteration 200  
starting iteration 210  
starting iteration 220  
starting iteration 230  
starting iteration 240  
starting iteration 250  
starting iteration 260  
starting iteration 270  
starting iteration 280  
starting iteration 290  
starting iteration 300  
starting iteration 310  
starting iteration 320  
starting iteration 330  
starting iteration 340  
starting iteration 350  
starting iteration 360  
starting iteration 370  
starting iteration 380  
starting iteration 390  
starting iteration 400
```

starting iteration 410  
starting iteration 420  
starting iteration 430  
starting iteration 440  
starting iteration 450  
starting iteration 460  
starting iteration 470  
starting iteration 480  
Finished epoch 1 / 10: cost 1.498791, train: 0.451000, val 0.450000, lr  
7.000000e-05  
starting iteration 490  
starting iteration 500  
starting iteration 510  
starting iteration 520  
starting iteration 530  
starting iteration 540  
starting iteration 550  
starting iteration 560  
starting iteration 570  
starting iteration 580  
starting iteration 590  
starting iteration 600  
starting iteration 610  
starting iteration 620  
starting iteration 630  
starting iteration 640  
starting iteration 650  
starting iteration 660  
starting iteration 670  
starting iteration 680  
starting iteration 690  
starting iteration 700  
starting iteration 710  
starting iteration 720  
starting iteration 730  
starting iteration 740  
starting iteration 750  
starting iteration 760  
starting iteration 770  
starting iteration 780  
starting iteration 790  
starting iteration 800  
starting iteration 810  
starting iteration 820  
starting iteration 830  
starting iteration 840  
starting iteration 850  
starting iteration 860

starting iteration 870  
starting iteration 880  
starting iteration 890  
starting iteration 900  
starting iteration 910  
starting iteration 920  
starting iteration 930  
starting iteration 940  
starting iteration 950  
starting iteration 960  
starting iteration 970  
Finished epoch 2 / 10: cost 1.667420, train: 0.510000, val 0.482000, lr  
4.900000e-05  
starting iteration 980  
starting iteration 990  
starting iteration 1000  
starting iteration 1010  
starting iteration 1020  
starting iteration 1030  
starting iteration 1040  
starting iteration 1050  
starting iteration 1060  
starting iteration 1070  
starting iteration 1080  
starting iteration 1090  
starting iteration 1100  
starting iteration 1110  
starting iteration 1120  
starting iteration 1130  
starting iteration 1140  
starting iteration 1150  
starting iteration 1160  
starting iteration 1170  
starting iteration 1180  
starting iteration 1190  
starting iteration 1200  
starting iteration 1210  
starting iteration 1220  
starting iteration 1230  
starting iteration 1240  
starting iteration 1250  
starting iteration 1260  
starting iteration 1270  
starting iteration 1280  
starting iteration 1290  
starting iteration 1300  
starting iteration 1310  
starting iteration 1320

starting iteration 1330  
starting iteration 1340  
starting iteration 1350  
starting iteration 1360  
starting iteration 1370  
starting iteration 1380  
starting iteration 1390  
starting iteration 1400  
starting iteration 1410  
starting iteration 1420  
starting iteration 1430  
starting iteration 1440  
starting iteration 1450  
starting iteration 1460  
Finished epoch 3 / 10: cost 1.220875, train: 0.513000, val 0.484000, lr  
3.430000e-05  
starting iteration 1470  
starting iteration 1480  
starting iteration 1490  
starting iteration 1500  
starting iteration 1510  
starting iteration 1520  
starting iteration 1530  
starting iteration 1540  
starting iteration 1550  
starting iteration 1560  
starting iteration 1570  
starting iteration 1580  
starting iteration 1590  
starting iteration 1600  
starting iteration 1610  
starting iteration 1620  
starting iteration 1630  
starting iteration 1640  
starting iteration 1650  
starting iteration 1660  
starting iteration 1670  
starting iteration 1680  
starting iteration 1690  
starting iteration 1700  
starting iteration 1710  
starting iteration 1720  
starting iteration 1730  
starting iteration 1740  
starting iteration 1750  
starting iteration 1760  
starting iteration 1770  
starting iteration 1780

starting iteration 1790  
starting iteration 1800  
starting iteration 1810  
starting iteration 1820  
starting iteration 1830  
starting iteration 1840  
starting iteration 1850  
starting iteration 1860  
starting iteration 1870  
starting iteration 1880  
starting iteration 1890  
starting iteration 1900  
starting iteration 1910  
starting iteration 1920  
starting iteration 1930  
starting iteration 1940  
starting iteration 1950  
Finished epoch 4 / 10: cost 1.415856, train: 0.548000, val 0.508000, lr  
2.401000e-05  
starting iteration 1960  
starting iteration 1970  
starting iteration 1980  
starting iteration 1990  
starting iteration 2000  
starting iteration 2010  
starting iteration 2020  
starting iteration 2030  
starting iteration 2040  
starting iteration 2050  
starting iteration 2060  
starting iteration 2070  
starting iteration 2080  
starting iteration 2090  
starting iteration 2100  
starting iteration 2110  
starting iteration 2120  
starting iteration 2130  
starting iteration 2140  
starting iteration 2150  
starting iteration 2160  
starting iteration 2170  
starting iteration 2180  
starting iteration 2190  
starting iteration 2200  
starting iteration 2210  
starting iteration 2220  
starting iteration 2230  
starting iteration 2240

starting iteration 2250  
starting iteration 2260  
starting iteration 2270  
starting iteration 2280  
starting iteration 2290  
starting iteration 2300  
starting iteration 2310  
starting iteration 2320  
starting iteration 2330  
starting iteration 2340  
starting iteration 2350  
starting iteration 2360  
starting iteration 2370  
starting iteration 2380  
starting iteration 2390  
starting iteration 2400  
starting iteration 2410  
starting iteration 2420  
starting iteration 2430  
starting iteration 2440  
Finished epoch 5 / 10: cost 1.094622, train: 0.580000, val 0.527000, lr  
1.680700e-05  
starting iteration 2450  
starting iteration 2460  
starting iteration 2470  
starting iteration 2480  
starting iteration 2490  
starting iteration 2500  
starting iteration 2510  
starting iteration 2520  
starting iteration 2530  
starting iteration 2540  
starting iteration 2550  
starting iteration 2560  
starting iteration 2570  
starting iteration 2580  
starting iteration 2590  
starting iteration 2600  
starting iteration 2610  
starting iteration 2620  
starting iteration 2630  
starting iteration 2640  
starting iteration 2650  
starting iteration 2660  
starting iteration 2670  
starting iteration 2680  
starting iteration 2690  
starting iteration 2700

starting iteration 2710  
starting iteration 2720  
starting iteration 2730  
starting iteration 2740  
starting iteration 2750  
starting iteration 2760  
starting iteration 2770  
starting iteration 2780  
starting iteration 2790  
starting iteration 2800  
starting iteration 2810  
starting iteration 2820  
starting iteration 2830  
starting iteration 2840  
starting iteration 2850  
starting iteration 2860  
starting iteration 2870  
starting iteration 2880  
starting iteration 2890  
starting iteration 2900  
starting iteration 2910  
starting iteration 2920  
starting iteration 2930  
Finished epoch 6 / 10: cost 1.398334, train: 0.613000, val 0.532000, lr  
1.176490e-05  
starting iteration 2940  
starting iteration 2950  
starting iteration 2960  
starting iteration 2970  
starting iteration 2980  
starting iteration 2990  
starting iteration 3000  
starting iteration 3010  
starting iteration 3020  
starting iteration 3030  
starting iteration 3040  
starting iteration 3050  
starting iteration 3060  
starting iteration 3070  
starting iteration 3080  
starting iteration 3090  
starting iteration 3100  
starting iteration 3110  
starting iteration 3120  
starting iteration 3130  
starting iteration 3140  
starting iteration 3150  
starting iteration 3160

starting iteration 3170  
starting iteration 3180  
starting iteration 3190  
starting iteration 3200  
starting iteration 3210  
starting iteration 3220  
starting iteration 3230  
starting iteration 3240  
starting iteration 3250  
starting iteration 3260  
starting iteration 3270  
starting iteration 3280  
starting iteration 3290  
starting iteration 3300  
starting iteration 3310  
starting iteration 3320  
starting iteration 3330  
starting iteration 3340  
starting iteration 3350  
starting iteration 3360  
starting iteration 3370  
starting iteration 3380  
starting iteration 3390  
starting iteration 3400  
starting iteration 3410  
starting iteration 3420  
Finished epoch 7 / 10: cost 1.084483, train: 0.562000, val 0.544000, lr  
8.235430e-06  
starting iteration 3430  
starting iteration 3440  
starting iteration 3450  
starting iteration 3460  
starting iteration 3470  
starting iteration 3480  
starting iteration 3490  
starting iteration 3500  
starting iteration 3510  
starting iteration 3520  
starting iteration 3530  
starting iteration 3540  
starting iteration 3550  
starting iteration 3560  
starting iteration 3570  
starting iteration 3580  
starting iteration 3590  
starting iteration 3600  
starting iteration 3610  
starting iteration 3620

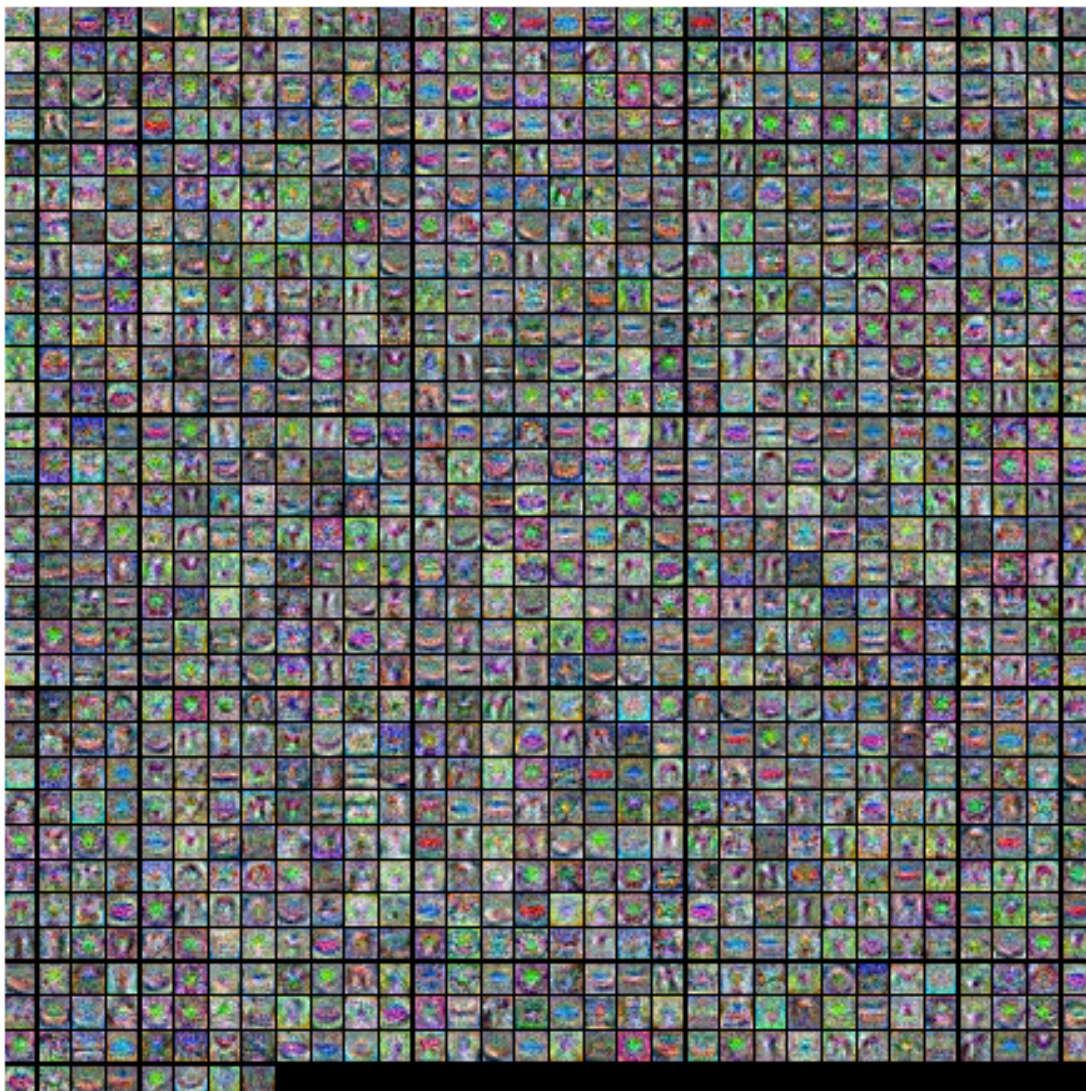


starting iteration 3630  
starting iteration 3640  
starting iteration 3650  
starting iteration 3660  
starting iteration 3670  
starting iteration 3680  
starting iteration 3690  
starting iteration 3700  
starting iteration 3710  
starting iteration 3720  
starting iteration 3730  
starting iteration 3740  
starting iteration 3750  
starting iteration 3760  
starting iteration 3770  
starting iteration 3780  
starting iteration 3790  
starting iteration 3800  
starting iteration 3810  
starting iteration 3820  
starting iteration 3830  
starting iteration 3840  
starting iteration 3850  
starting iteration 3860  
starting iteration 3870  
starting iteration 3880  
starting iteration 3890  
starting iteration 3900  
starting iteration 3910  
Finished epoch 8 / 10: cost 1.069327, train: 0.624000, val 0.534000, lr  
5.764801e-06  
starting iteration 3920  
starting iteration 3930  
starting iteration 3940  
starting iteration 3950  
starting iteration 3960  
starting iteration 3970  
starting iteration 3980  
starting iteration 3990  
starting iteration 4000  
starting iteration 4010  
starting iteration 4020  
starting iteration 4030  
starting iteration 4040  
starting iteration 4050  
starting iteration 4060  
starting iteration 4070  
starting iteration 4080

```
starting iteration 4090
starting iteration 4100
starting iteration 4110
starting iteration 4120
starting iteration 4130
starting iteration 4140
starting iteration 4150
starting iteration 4160
starting iteration 4170
starting iteration 4180
starting iteration 4190
starting iteration 4200
starting iteration 4210
starting iteration 4220
starting iteration 4230
starting iteration 4240
starting iteration 4250
starting iteration 4260
starting iteration 4270
starting iteration 4280
starting iteration 4290
starting iteration 4300
starting iteration 4310
starting iteration 4320
starting iteration 4330
starting iteration 4340
starting iteration 4350
starting iteration 4360
starting iteration 4370
starting iteration 4380
starting iteration 4390
starting iteration 4400
Finished epoch 9 / 10: cost 1.126785, train: 0.651000, val 0.557000, lr
4.035361e-06
starting iteration 4410
starting iteration 4420
starting iteration 4430
starting iteration 4440
starting iteration 4450
starting iteration 4460
starting iteration 4470
starting iteration 4480
starting iteration 4490
starting iteration 4500
starting iteration 4510
starting iteration 4520
starting iteration 4530
starting iteration 4540
```

```
starting iteration 4550
starting iteration 4560
starting iteration 4570
starting iteration 4580
starting iteration 4590
starting iteration 4600
starting iteration 4610
starting iteration 4620
starting iteration 4630
starting iteration 4640
starting iteration 4650
starting iteration 4660
starting iteration 4670
starting iteration 4680
starting iteration 4690
starting iteration 4700
starting iteration 4710
starting iteration 4720
starting iteration 4730
starting iteration 4740
starting iteration 4750
starting iteration 4760
starting iteration 4770
starting iteration 4780
starting iteration 4790
starting iteration 4800
starting iteration 4810
starting iteration 4820
starting iteration 4830
starting iteration 4840
starting iteration 4850
starting iteration 4860
starting iteration 4870
starting iteration 4880
starting iteration 4890
Finished epoch 10 / 10: cost 1.272121, train: 0.601000, val 0.543000, lr
2.824752e-06
finished optimization. best validation accuracy: 0.557000
```

```
[29]: # visualize the weights
      show_net_weights(best_model)
```



## 10 Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set.

```
[30]: scores_test = two_layer_net(X_test, best_model)
      print('Test accuracy: ', np.mean(np.argmax(scores_test, axis=1) == y_test))
```

Test accuracy: 0.538

```
[ ]:
```