two_layer_net

June 7, 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.

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

        import numpy as np
        import matplotlib.pyplot as plt

        from cs231n.classifiers.neural_net import TwoLayerNet

        %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))))
```

We will use the class `TwoLayerNet` in the file `cs231n/classifiers/neural_net.py` to represent instances of our network. The network parameters are stored in the instance variable `self.params` where keys are string parameter names and values are numpy arrays. Below, we initialize toy data and a toy model that we will use to develop your implementation.

```
In [2]: # 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()
```

2 Forward pass: compute scores

Open the file `cs231n/classifiers/neural_net.py` and look at the method `TwoLayerNet.loss`. This function is very similar to the loss functions you have written for the SVM and Softmax exercises: 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.

```
In [3]: scores = net.loss(X)
        print('Your scores:')
        print(scores)
        print()
        print('correct scores:')
        correct_scores = np.asarray([
          [-0.81233741, -1.27654624, -0.70335995],
          [-0.17129677, -1.18803311, -0.47310444],
          [-0.51590475, -1.01354314, -0.8504215 ],
          [-0.15419291, -0.48629638, -0.52901952],
          [-0.00618733, -0.12435261, -0.15226949]])
        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:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]
```

```
correct scores:
[[-0.81233741 -1.27654624 -0.70335995]
 [-0.17129677 -1.18803311 -0.47310444]
 [-0.51590475 -1.01354314 -0.8504215 ]
 [-0.15419291 -0.48629638 -0.52901952]
 [-0.00618733 -0.12435261 -0.15226949]]

Difference between your scores and correct scores:
3.6802720745909845e-08
```

3 Forward pass: compute loss

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

```
In [4]: loss, _ = net.loss(X, y, reg=0.05)
        correct_loss = 1.30378789133

        # 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:
1.7985612998927536e-13
```

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:

```
In [5]: 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 = 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], verbose=False)
            print('%s max relative error: %e' % (param_name, rel_error(param_grad_num, grads[pa
```

```
W1 max relative error: 3.561318e-09
b1 max relative error: 2.738421e-09
W2 max relative error: 3.440708e-09
b2 max relative error: 4.447625e-11
```

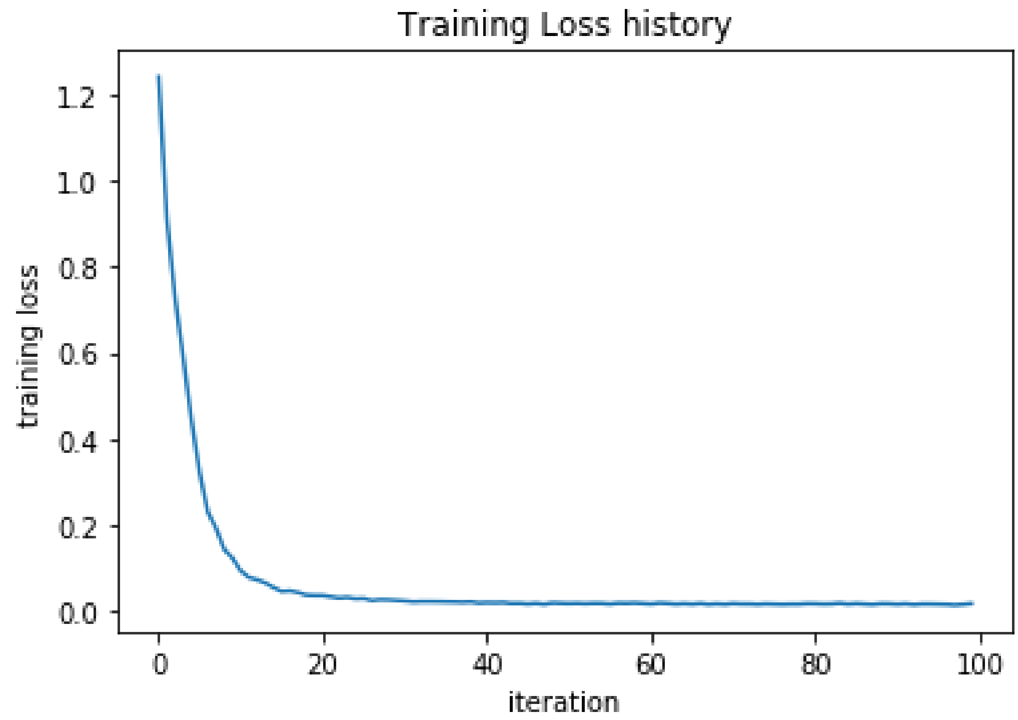
5 Train the network

To train the network we will use stochastic gradient descent (SGD), similar to the SVM and Softmax classifiers. Look at the function `TwoLayerNet.train` and fill in the missing sections to implement the training procedure. This should be very similar to the training procedure you used for the SVM and Softmax classifiers. You will also have to implement `TwoLayerNet.predict`, as the training process periodically performs prediction to keep track of accuracy over time while the network trains.

Once you have implemented the method, run the code below to train a two-layer network on toy data. You should achieve a training loss less than 0.02.

```
In [6]: 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:  0.017149607938732093
```



6 Load the data

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

```
In [7]: 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. These are the same steps as
            we used for the SVM, but condensed to a single function.
            """
            # Load the raw CIFAR-10 data
            cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'

            # Cleaning up variables to prevent loading data multiple times (which may cause me
            try:
               del X_train, y_train
               del X_test, y_test
               print('Clear previously loaded data.')
            except:
```

```
            pass

        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,)
```

7 Train a network

To train our network we will use SGD. 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.

```
In [8]: 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=1500, batch_size=200,
                    learning_rate=1e-3, learning_rate_decay=0.9,
                    reg=0.5, verbose=True)

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

        # got 46.2 ! sweet...
```

```
iteration 0 / 1500: loss 2.303339
iteration 100 / 1500: loss 1.983483
iteration 200 / 1500: loss 1.779908
iteration 300 / 1500: loss 1.648644
iteration 400 / 1500: loss 1.750870
iteration 500 / 1500: loss 1.589080
iteration 600 / 1500: loss 1.693090
iteration 700 / 1500: loss 1.485547
iteration 800 / 1500: loss 1.527232
iteration 900 / 1500: loss 1.554494
iteration 1000 / 1500: loss 1.587535
iteration 1100 / 1500: loss 1.529995
iteration 1200 / 1500: loss 1.626180
iteration 1300 / 1500: loss 1.596511
iteration 1400 / 1500: loss 1.618159
Validation accuracy:  0.462
```

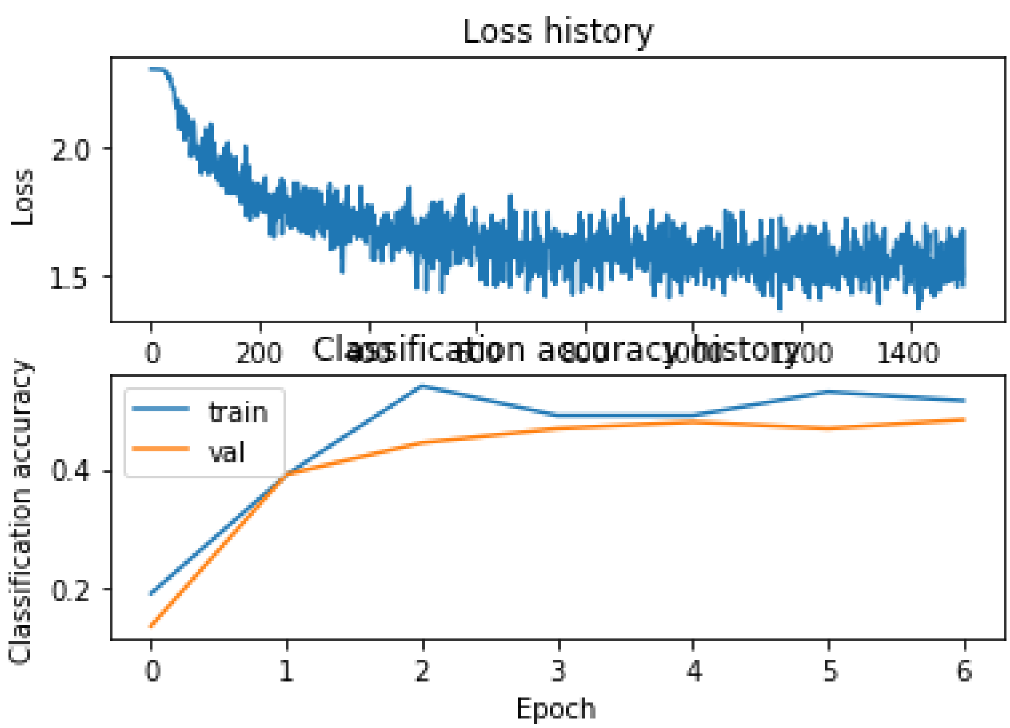
8 Debug the training

With the default parameters we provided above, you should get a validation accuracy of about 0.29 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.

```
In [9]: # Plot the loss function and train / validation accuracies
        plt.subplot(2, 1, 1)
        plt.plot(stats['loss_history'])
        plt.title('Loss history')
        plt.xlabel('Iteration')
        plt.ylabel('Loss')

        plt.subplot(2, 1, 2)
        plt.plot(stats['train_acc_history'], label='train')
        plt.plot(stats['val_acc_history'], label='val')
        plt.title('Classification accuracy history')
        plt.xlabel('Epoch')
        plt.ylabel('Classification accuracy')
        plt.legend()
        plt.show()
```




```
In [10]: from cs231n.vis_utils import visualize_grid

          # Visualize the weights of the network
```

```python
def show_net_weights(net):
    W1 = net.params['W1']
    W1 = W1.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(net)
```

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

Approximate results. You should be aim to achieve a classification accuracy of greater than 48% on the validation set. Our best network gets over 52% on the validation set.

Experiment: You goal in this exercise is to get as good of a result on CIFAR-10 as you can (52% could serve as a reference), with a fully-connected Neural Network. Feel free implement your own techniques (e.g. PCA to reduce dimensionality, or adding dropout, or adding features to the solver, etc.).

Explain your hyperparameter tuning process below.

Your Answer : I was able to get 47.3 percent val accuracy just by increasing the learning rate, number of iterations and the regularisation.

In [12]:
```
best_net = None # store the best model into this
best_val = -1
best_res = []
#################################################################################
# TODO: Tune hyperparameters using the validation set. Store your best trained  #
# model in best_net.                                                            #
#                                                                               #
# 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 exercises.                          #
#################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

lrs = [1e-3, 5e-3, 1e-4]
hidden_layer_s = [40, 80, 160, 200]
regs = [0.5, 0.75, 1.25, 1.5]
iters = 1500
for lr in lrs:
    for hls in hidden_layer_s:
        for r in regs:
            model = TwoLayerNet(32*32*3, hls, 10)
            res = model.train(X_train, y_train, X_val, y_val,
                        num_iters=iters, batch_size=200,
                        learning_rate=lr, learning_rate_decay=0.9,
                        reg=r, verbose=False)
            # training set
            train_acc = (model.predict(X_train) == y_train).mean()
            # val set
            val_acc = (model.predict(X_val) == y_val).mean()
            # save model
            if val_acc > best_val:
                best_val = val_acc
                best_net = model
                best_res = res

            # Print results
```

```python
            print('learning_rate %e reg_strength %e hid %d  train_acc: %f val_accaccu
    print('Best validation accuracy achieved: %f' % best_val)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

```
learning_rate 1.000000e-03 reg_strength 5.000000e-01 hid 40  train_acc: 0.491980 val_accaccura
learning_rate 1.000000e-03 reg_strength 7.500000e-01 hid 40  train_acc: 0.483939 val_accaccura
learning_rate 1.000000e-03 reg_strength 1.250000e+00 hid 40  train_acc: 0.469816 val_accaccura
learning_rate 1.000000e-03 reg_strength 1.500000e+00 hid 40  train_acc: 0.458102 val_accaccura
learning_rate 1.000000e-03 reg_strength 5.000000e-01 hid 80  train_acc: 0.509898 val_accaccura
learning_rate 1.000000e-03 reg_strength 7.500000e-01 hid 80  train_acc: 0.496816 val_accaccura
learning_rate 1.000000e-03 reg_strength 1.250000e+00 hid 80  train_acc: 0.480184 val_accaccura
learning_rate 1.000000e-03 reg_strength 1.500000e+00 hid 80  train_acc: 0.467082 val_accaccura
learning_rate 1.000000e-03 reg_strength 5.000000e-01 hid 160  train_acc: 0.514571 val_accaccur
learning_rate 1.000000e-03 reg_strength 7.500000e-01 hid 160  train_acc: 0.502408 val_accaccur
learning_rate 1.000000e-03 reg_strength 1.250000e+00 hid 160  train_acc: 0.486163 val_accaccur
learning_rate 1.000000e-03 reg_strength 1.500000e+00 hid 160  train_acc: 0.477653 val_accaccur
learning_rate 1.000000e-03 reg_strength 5.000000e-01 hid 200  train_acc: 0.521857 val_accaccur
learning_rate 1.000000e-03 reg_strength 7.500000e-01 hid 200  train_acc: 0.506347 val_accaccur
learning_rate 1.000000e-03 reg_strength 1.250000e+00 hid 200  train_acc: 0.485000 val_accaccur
learning_rate 1.000000e-03 reg_strength 1.500000e+00 hid 200  train_acc: 0.476408 val_accaccur
learning_rate 5.000000e-03 reg_strength 5.000000e-01 hid 40  train_acc: 0.201469 val_accaccura
learning_rate 5.000000e-03 reg_strength 7.500000e-01 hid 40  train_acc: 0.207408 val_accaccura
learning_rate 5.000000e-03 reg_strength 1.250000e+00 hid 40  train_acc: 0.171939 val_accaccura
learning_rate 5.000000e-03 reg_strength 1.500000e+00 hid 40  train_acc: 0.138531 val_accaccura
learning_rate 5.000000e-03 reg_strength 5.000000e-01 hid 80  train_acc: 0.160102 val_accaccura
learning_rate 5.000000e-03 reg_strength 7.500000e-01 hid 80  train_acc: 0.153429 val_accaccura
learning_rate 5.000000e-03 reg_strength 1.250000e+00 hid 80  train_acc: 0.163490 val_accaccura
learning_rate 5.000000e-03 reg_strength 1.500000e+00 hid 80  train_acc: 0.168143 val_accaccura
learning_rate 5.000000e-03 reg_strength 5.000000e-01 hid 160  train_acc: 0.174082 val_accaccur
learning_rate 5.000000e-03 reg_strength 7.500000e-01 hid 160  train_acc: 0.181000 val_accaccur
learning_rate 5.000000e-03 reg_strength 1.250000e+00 hid 160  train_acc: 0.171061 val_accaccur
learning_rate 5.000000e-03 reg_strength 1.500000e+00 hid 160  train_acc: 0.176388 val_accaccur
learning_rate 5.000000e-03 reg_strength 5.000000e-01 hid 200  train_acc: 0.171959 val_accaccur
learning_rate 5.000000e-03 reg_strength 7.500000e-01 hid 200  train_acc: 0.163245 val_accaccur
learning_rate 5.000000e-03 reg_strength 1.250000e+00 hid 200  train_acc: 0.171633 val_accaccur
learning_rate 5.000000e-03 reg_strength 1.500000e+00 hid 200  train_acc: 0.161796 val_accaccur
learning_rate 1.000000e-04 reg_strength 5.000000e-01 hid 40  train_acc: 0.299143 val_accaccura
learning_rate 1.000000e-04 reg_strength 7.500000e-01 hid 40  train_acc: 0.301143 val_accaccura
learning_rate 1.000000e-04 reg_strength 1.250000e+00 hid 40  train_acc: 0.303694 val_accaccura
learning_rate 1.000000e-04 reg_strength 1.500000e+00 hid 40  train_acc: 0.296020 val_accaccura
```

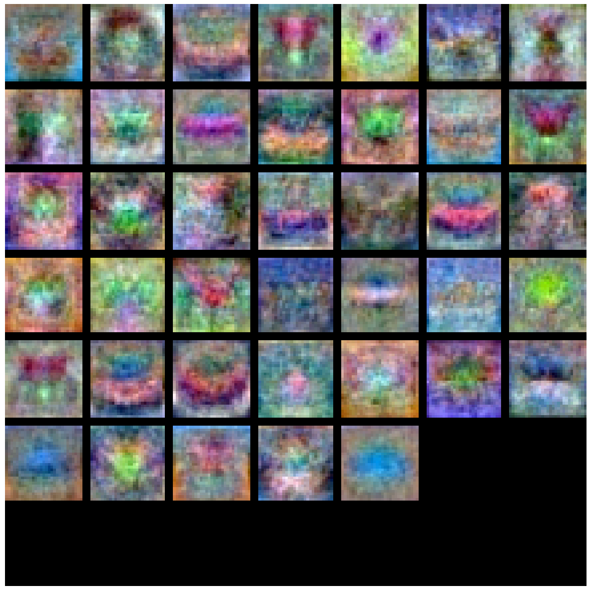
```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)
```

```
<ipython-input-12-a25e34d967c9> in <module>
 27                              num_iters=iters, batch_size=200,
 28                              learning_rate=lr, learning_rate_decay=0.9,
---> 29                              reg=r, verbose=False)
 30                 # training set
 31                 train_acc = (model.predict(X_train) == y_train).mean()


~/Desktop/cs231n/cs231n/assignment1/cs231n/classifiers/neural_net.py in train(self, X,
177
178                 # Compute loss and gradients using the current minibatch
--> 179             loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
180                 loss_history.append(loss)
181


~/Desktop/cs231n/cs231n/assignment1/cs231n/classifiers/neural_net.py in loss(self, X,
 79             # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
 80
---> 81             layer_1 = np.dot(X, W1) + b1  # FC1
 82             a_1 = np.maximum(0, layer_1)  # relu
 83             scores = np.dot(a_1, W2) + b2  # FC2


KeyboardInterrupt:
```

In [13]:
```
# Print your validation accuracy: this should be above 48%
val_acc = (best_net.predict(X_val) == y_val).mean()
print('Validation accuracy: ', val_acc)
```

```
Validation accuracy:  0.492
```

In [14]:
```
# Visualize the weights of the best network
show_net_weights(best_net)
```



10 Run on the test set

When you are done experimenting, you should evaluate your final trained network on the test set; you should get above 48%.

```
In [15]: # Print your test accuracy: this should be above 48%
         test_acc = (best_net.predict(X_test) == y_test).mean()
         print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.48
```

Inline Question

Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

$YourAnswer:$ 1, 3

$YourExplanation:$ More data of each class prevents overfitting and increases model's tendency to generalize better on the test set.

regularisation also makes the model better by punishing it for large weights, and enables training of complex models without overfitting.

---

11 IMPORTANT

This is the end of this question. Please do the following:

1. Click `File -> Save` to make sure the latest checkpoint of this notebook is saved to your Drive.
2. Execute the cell below to download the modified `.py` files back to your drive.

```
In [ ]: import os

        FOLDER_TO_SAVE = os.path.join('drive/My Drive/', FOLDERNAME)
        FILES_TO_SAVE = ['cs231n/classifiers/neural_net.py']

        for files in FILES_TO_SAVE:
          with open(os.path.join(FOLDER_TO_SAVE, '/'.join(files.split('/')[1:])), 'w') as f:
            f.write(''.join(open(files).readlines()))
```

```
In [ ]:
```

```
In [ ]:
```