two_layer_net

October 9, 2023

1

CIFAR-10

[1]:
```
#
import numpy as np
import matplotlib.pyplot as plt
from daseCV.classifiers.neural_net import TwoLayerNet

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

#
# 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))))
```

daseCV/classifiers/neural_net TwoLayerNet self.params , numpy

[2]:
```
#
#

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 scores

daseCV/classifiers/neural_net TwoLayerNet.loss SVM Softmax scores loss

scores

[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 :

[4]:
```
loss, _ = net.loss(X, y, reg=0.05) #reg 0.1
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:
0.01896541960606335
```

4

W1, b1, W2, b2 (hopefully!) debug :

[5]:
```
from daseCV.gradient_check import eval_numerical_gradient

#
#         W1 W2 b1 b2
#           1e-8

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[param_name])))
```

```
W2 max relative error: 3.440708e-09
b2 max relative error: 4.447646e-11
W1 max relative error: 4.090896e-09
b1 max relative error: 2.738421e-09
```

5

(SGD) SVM Softmax TwoLayerNet.train SVM Softmax TwoLayerNet.predic

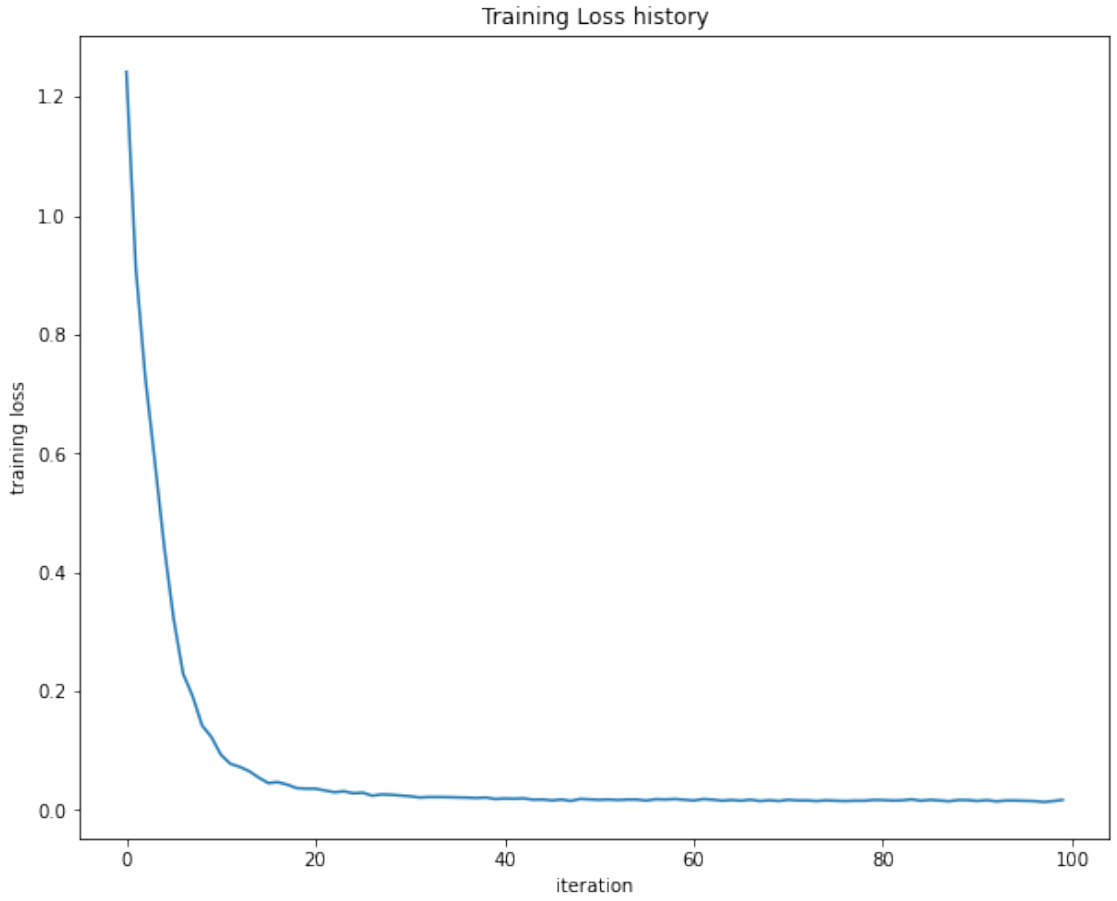
0.02

[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:  0.017143643532923757
```



6

, CIFAR-10 ( )

[7]:
```
from daseCV.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 = 'daseCV/datasets/cifar-10-batches-py'

    #   ,     (     )
    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

SGD , ,

[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=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)
```

```
iteration 0 / 1000: loss 2.302762
iteration 100 / 1000: loss 2.302358
iteration 200 / 1000: loss 2.297404
iteration 300 / 1000: loss 2.258897
iteration 400 / 1000: loss 2.202975
iteration 500 / 1000: loss 2.116816
iteration 600 / 1000: loss 2.049789
iteration 700 / 1000: loss 1.985711
iteration 800 / 1000: loss 2.003726
iteration 900 / 1000: loss 1.948076
```

```
Validation accuracy:  0.287
```

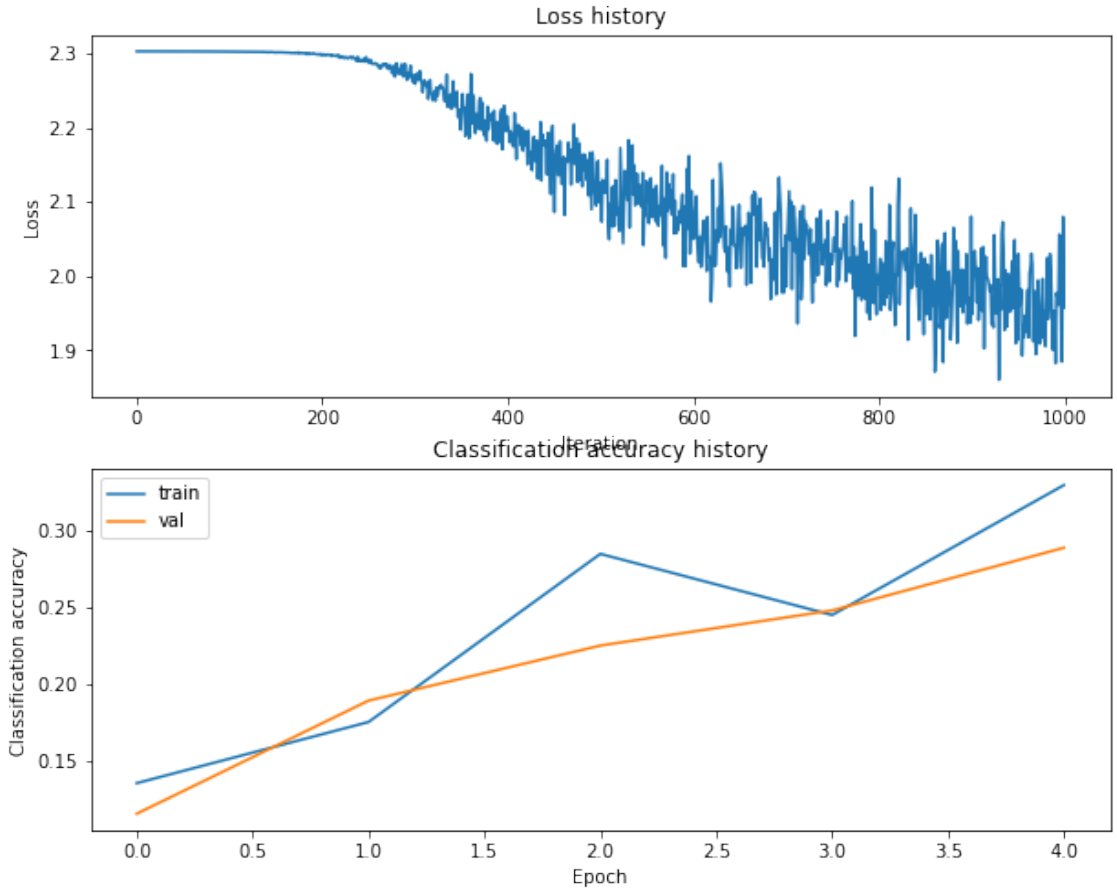
8 Debug

0.29

,

[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()
```

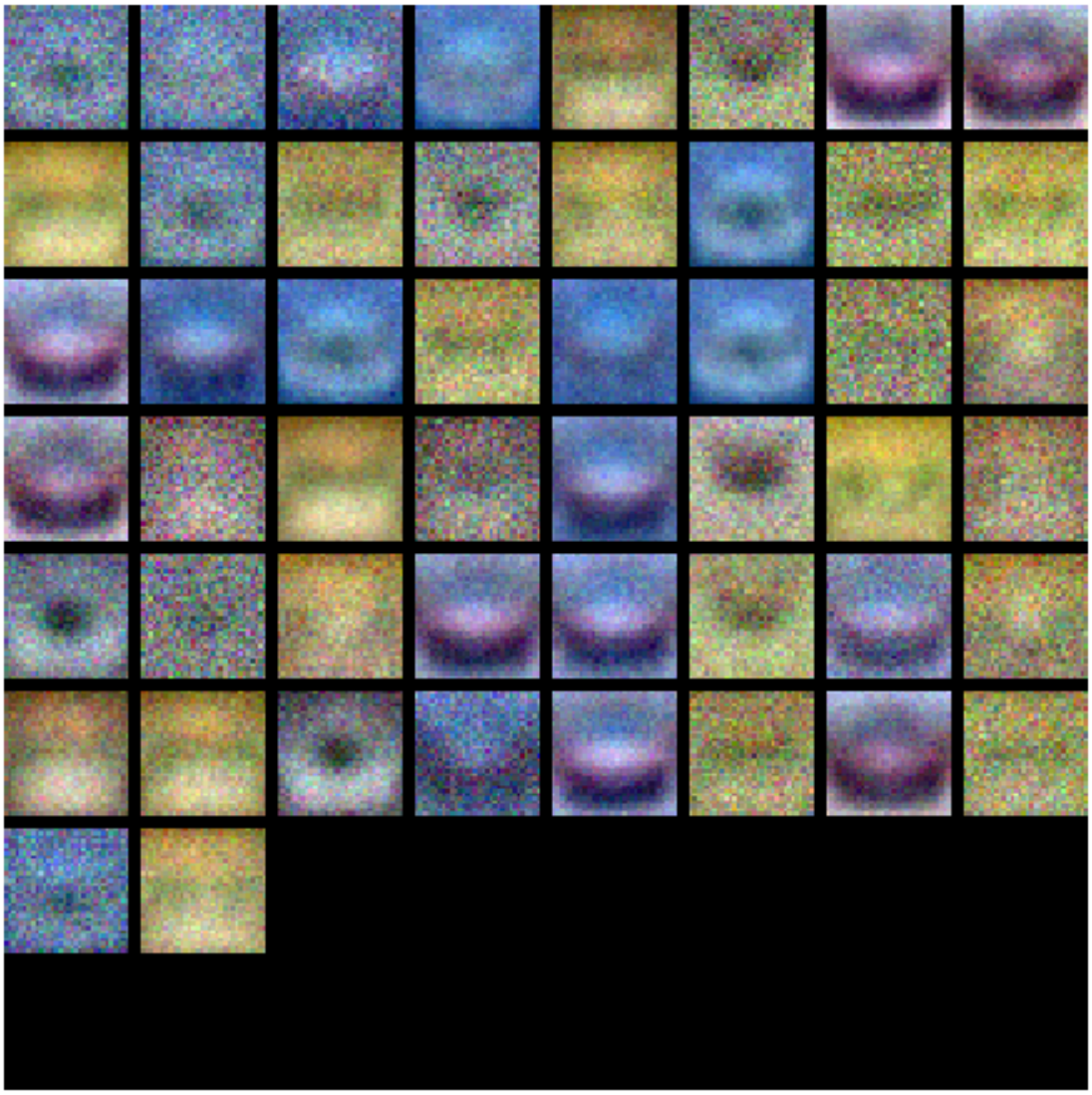


[10]:
```
from daseCV.vis_utils import visualize_grid

# Visualize the weights of the network

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

What's wrong?.

Tuning.

Approximate results. 48% 52%

Experiment: CIFAR-10 (52%) (PCA dropout)

:

[11]:
```
best_net = None # store the best model into this

#################################################################################
# TODO             best_net
#
#
#
#################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

iters_num = 1000
best_val = 0
learning_rates = [1e-4,3e-4,5e-4,7e-4]
regularization_strengths = [0.2, 0.25, 0.3, 0.35, 0.4]
net = TwoLayerNet(input_size, hidden_size,num_classes)
for lr in learning_rates:
    for reg in regularization_strengths:
        stats = net.train(X_train, y_train, X_val, y_val,
                          num_iters=iters_num,batch_size=200,
                          learning_rate=lr,learning_rate_decay=0.95,
                          reg=reg, verbose=False)
        val_acc = (net.predict(X_val) == y_val).mean()
        if val_acc > best_val:
            best_val = val_acc
            best_net = net
        print ("lr ",lr, "reg ", reg, "val accuracy:", val_acc)
print ("best validation accuracyachieved during cross-validation: ", best_val)

pass

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

```
lr  0.0001 reg  0.2 val accuracy: 0.285
lr  0.0001 reg  0.25 val accuracy: 0.363
lr  0.0001 reg  0.3 val accuracy: 0.406
lr  0.0001 reg  0.35 val accuracy: 0.441
lr  0.0001 reg  0.4 val accuracy: 0.452
lr  0.0003 reg  0.2 val accuracy: 0.471
lr  0.0003 reg  0.25 val accuracy: 0.485
lr  0.0003 reg  0.3 val accuracy: 0.492
lr  0.0003 reg  0.35 val accuracy: 0.508
lr  0.0003 reg  0.4 val accuracy: 0.508
lr  0.0005 reg  0.2 val accuracy: 0.505
lr  0.0005 reg  0.25 val accuracy: 0.511
lr  0.0005 reg  0.3 val accuracy: 0.517
lr  0.0005 reg  0.35 val accuracy: 0.507
lr  0.0005 reg  0.4 val accuracy: 0.51
```

```
lr  0.0007 reg  0.2 val accuracy: 0.502
lr  0.0007 reg  0.25 val accuracy: 0.509
lr  0.0007 reg  0.3 val accuracy: 0.503
lr  0.0007 reg  0.35 val accuracy: 0.492
lr  0.0007 reg  0.4 val accuracy: 0.482
best validation accuracyachieved during cross-validation:  0.517
```

[ ]:
```
# visualize the weights of the best network
show_net_weights(best_net)
```

10

48%

[13]:
```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

```
Test accuracy:  0.509
```

2

?

1.
2.
3.
4.

YourAnswer : 1,3

YourExplanation :

10.0.1 Data for leaderboard

X leaderborad

---

[15]:
```
# leaderboard
X = np.load("./input/X_3072+.npy")
################################################################################
#     :
#    net
#                best_net
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
net_leaderboard = best_net
preds = net_leaderboard.predict(X)
```

submit leaderboard phase4 leaderboard

[16]:
```
import os
#
def output_file(preds, phase_id=4):
    path=os.getcwd()
    if not os.path.exists(path + '/output/phase_{}'.format(phase_id)):
        os.mkdir(path + '/output/phase_{}'.format(phase_id))
    path=path + '/output/phase_{}/prediction.npy'.format(phase_id)
    np.save(path,preds)
def zip_fun(phase_id=4):
    path=os.getcwd()
    output_path = path + '/output'
    files = os.listdir(output_path)
    for _file in files:
        if _file.find('zip') != -1:
            os.remove(output_path + '/' + _file)
    newpath=path+'/output/phase_{}'.format(phase_id)
    os.chdir(newpath)
    cmd = 'zip ../prediction_phase_{}.zip prediction.npy'.format(phase_id)
    os.system(cmd)
    os.chdir(path)
output_file(preds)
zip_fun()
```

[ ]:
```

```