softmax

October 9, 2023

1 Softmax

SVM :

- Softmax
- **analytic gradient**
-
-
- **SGD**
-

[1]:
```
import random
import numpy as np
from daseCV.data_utils import load_CIFAR10
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 extenrnal modules
# see http://stackoverflow.com/questions/1907993/
↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

[2]:
```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000,␣
↪num_dev=500):
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the linear 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'
```

```python
    # Cleaning up variables to prevent loading data multiple times (which may␣
↪cause memory issue)
    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]
    mask = np.random.choice(num_training, num_dev, replace=False)
    X_dev = X_train[mask]
    y_dev = y_train[mask]

    # Preprocessing: reshape the image data into rows
    X_train = np.reshape(X_train, (X_train.shape[0], -1))
    X_val = np.reshape(X_val, (X_val.shape[0], -1))
    X_test = np.reshape(X_test, (X_test.shape[0], -1))
    X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

    # 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
    X_dev -= mean_image

    # add bias dimension and transform into columns
    X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
    X_val = np.hstack([X_val, np.ones((X_val.shape[0], 1))])
    X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
    X_dev = np.hstack([X_dev, np.ones((X_dev.shape[0], 1))])

    return X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_dev
```

```
# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_dev =␣
 ↪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)
print('dev data shape: ', X_dev.shape)
print('dev labels shape: ', y_dev.shape)
```

```
Train data shape:  (49000, 3073)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3073)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3073)
Test labels shape:  (1000,)
dev data shape:  (500, 3073)
dev labels shape:  (500,)
```

1.1 Softmax

daseCV/classifiers/softmax.py

[3]:
```
#        softmax
#    daseCV/classifiers/softmax.py
# softmax_loss_naive  .

from daseCV.classifiers.softmax import softmax_loss_naive
import time

#     softmax
W = np.random.randn(3073, 10) * 0.0001
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# As a rough sanity check, our loss should be something close to -log(0.1).
print('loss: %f' % loss)
print('sanity check: %f' % (-np.log(0.1)))
```

```
loss: 2.333154
sanity check: 2.302585
```

1

-log 0.1

: w X 0 softmax label 10 label 1/10 0.1, loss -log 0.1

[4]:
```python
#  softmax_loss_naive        (naive)
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

#  SVM
#
from daseCV.gradient_check import grad_check_sparse
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)

#  SVM
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 5e1)
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 5e1)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

```
numerical: -1.214096 analytic: -1.214096, relative error: 1.824890e-08
numerical: -2.353034 analytic: -2.353034, relative error: 2.191966e-08
numerical: 1.838423 analytic: 1.838423, relative error: 2.319794e-09
numerical: 1.112955 analytic: 1.112955, relative error: 2.249519e-08
numerical: 1.669272 analytic: 1.669272, relative error: 3.631004e-08
numerical: 2.391128 analytic: 2.391127, relative error: 3.391090e-08
numerical: -0.368852 analytic: -0.368852, relative error: 4.638295e-08
numerical: 0.891265 analytic: 0.891265, relative error: 1.223761e-08
numerical: -0.486324 analytic: -0.486324, relative error: 1.260306e-07
numerical: 2.819382 analytic: 2.819382, relative error: 1.526460e-08
numerical: 2.535714 analytic: 2.535714, relative error: 1.212420e-08
numerical: -3.010735 analytic: -3.010735, relative error: 2.047830e-08
numerical: -2.085661 analytic: -2.085661, relative error: 8.462578e-09
numerical: -1.087635 analytic: -1.087635, relative error: 5.129784e-08
numerical: -1.482118 analytic: -1.482118, relative error: 4.980695e-08
numerical: -0.939550 analytic: -0.939550, relative error: 4.644179e-08
numerical: 2.408302 analytic: 2.408301, relative error: 1.988946e-08
numerical: -1.645823 analytic: -1.645823, relative error: 9.342905e-09
numerical: 2.148862 analytic: 2.148862, relative error: 1.639053e-08
numerical: -0.592090 analytic: -0.592090, relative error: 4.427353e-08
```

[5]:
```python
#     softmax
#     softmax_loss_vectorized      .
#
tic = time.time()
loss_naive, grad_naive = softmax_loss_naive(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('naive loss: %e computed in %fs' % (loss_naive, toc - tic))

from daseCV.classifiers.softmax import softmax_loss_vectorized
tic = time.time()
loss_vectorized, grad_vectorized = softmax_loss_vectorized(W, X_dev, y_dev, 0.
 ↪000005)
```

```python
toc = time.time()
print('vectorized loss: %e computed in %fs' % (loss_vectorized, toc - tic))

#    SVM       Frobenius
grad_difference = np.linalg.norm(grad_naive - grad_vectorized, ord='fro')
print('Loss difference: %f' % np.abs(loss_naive - loss_vectorized))
print('Gradient difference: %f' % grad_difference)
```

```
naive loss: 2.333154e+00 computed in 1.295214s
vectorized loss: 2.333154e+00 computed in 0.092856s
Loss difference: 0.000000
Gradient difference: 0.000000
```

[ ]:
```python
#                      ;
#            0.35
from daseCV.classifiers import Softmax
results = {}
best_val = -1
best_softmax = None
learning_rates = [1e-7, 5e-7]
regularization_strengths = [2.5e4, 5e4]

################################################################################
#    :
#
#   SVM
#      softmax  best_softmax
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for lr in learning_rates:
    for reg in regularization_strengths:
        #  Softmax
        softmax = Softmax()
        #
        softmax.train(X_train, y_train, learning_rate=lr, reg=reg,␣
 ↪num_iters=1500)
        #
        y_train_pred = softmax.predict(X_train)
        train_accuracy = np.mean(y_train == y_train_pred)
        y_val_pred = softmax.predict(X_val)
        val_accuracy = np.mean(y_val == y_val_pred)
        #
        results[(lr, reg)] = (train_accuracy, val_accuracy)
        #
        if val_accuracy > best_val:
            best_val = val_accuracy
```

```
            best_softmax = softmax

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

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f' %␣
 ↪best_val)
```

[ ]:
```
#
#       softmax
y_test_pred = best_softmax.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('softmax on raw pixels final test set accuracy: %f' % (test_accuracy, ))
```

2 -

SVM Softmax

:

: SVM SVM

Softmax Softmax

[ ]:
```
#
w = best_softmax.W[:-1,:] # strip out the bias
w = w.reshape(32, 32, 3, 10)

w_min, w_max = np.min(w), np.max(w)

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',␣
 ↪'ship', 'truck']
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[:, :, :, i].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype('uint8'))
    plt.axis('off')
    plt.title(classes[i])
```

1.1.1 Data for leaderboard

X leaderborad

[ ]:
```python
# leaderboard
X = np.load("./input/X_3073.npy")
################################################################################
#     :
#     softmax
#                  best_softmax
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
softmax_leaderboard = best_softmax
preds = softmax_leaderboard.predict(X)
```

submit leaderboard phase3 leaderboard

[ ]:
```python
import os
#
def output_file(preds, phase_id=3):
    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=3):
    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()
```

[ ]:
```python

```