softmax

January 21, 2023

[1]:
```
# This mounts your Google Drive to the Colab VM.
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# assignment folder, e.g. 'cs231n/assignments/assignment1/'
FOLDERNAME = 'cs231n/assignments/assignment1/'
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))

# This downloads the CIFAR-10 dataset to your Drive
# if it doesn't already exist.
%cd /content/drive/My\ Drive/$FOLDERNAME/cs231n/datasets/
!bash get_datasets.sh
%cd /content/drive/My\ Drive/$FOLDERNAME
```

```
Mounted at /content/drive
/content/drive/My Drive/cs231n/assignments/assignment1/cs231n/datasets
/content/drive/My Drive/cs231n/assignments/assignment1
```

1 Softmax exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the assignments page on the course website.

This exercise is analogous to the SVM exercise. You will:

- implement a fully-vectorized **loss function** for the Softmax classifier
- implement the fully-vectorized expression for its **analytic gradient**
- **check your implementation** with numerical gradient
- use a validation set to **tune the learning rate and regularization** strength
- **optimize** the loss function with **SGD**

- **visualize** the final learned weights

[2]:
```
import random
import numpy as np
from cs231n.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
```

[3]:
```
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 = 'cs231n/datasets/cifar-10-batches-py'

    # 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))
```

```python
    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 Classifier

Your code for this section will all be written inside `cs231n/classifiers/softmax.py`.

[13]:
```python
# First implement the naive softmax loss function with nested loops.
# Open the file cs231n/classifiers/softmax.py and implement the
# softmax_loss_naive function.

from cs231n.classifiers.softmax import softmax_loss_naive
import time

# Generate a random softmax weight matrix and use it to compute the loss.
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: 2.355952
sanity check: 2.302585
```

Inline Question 1

Why do we expect our loss to be close to -log(0.1)? Explain briefly.

$\color{blue}{\textit{Your Answer :}}$ Because if a classifier randomly predict what the label is, the distribution should be uniform distribution, then the probability of 10 classes is 0.1, then cross entropy loss is 1*-log(0.1)

[15]:
```python
# Complete the implementation of softmax_loss_naive and implement a (naive)
# version of the gradient that uses nested loops.
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# As we did for the SVM, use numeric gradient checking as a debugging tool.
# The numeric gradient should be close to the analytic gradient.
from cs231n.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)

# similar to SVM case, do another gradient check with regularization
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: -0.122120 analytic: -0.122120, relative error: 2.009192e-07
numerical: 0.162887 analytic: 0.162887, relative error: 3.573343e-07
numerical: -0.622939 analytic: -0.622939, relative error: 1.047175e-08
numerical: 1.692220 analytic: 1.692220, relative error: 3.152539e-08
```

```
numerical: 0.312521 analytic: 0.312521, relative error: 9.953121e-09
numerical: 1.588910 analytic: 1.588910, relative error: 1.201498e-08
numerical: 1.968978 analytic: 1.968978, relative error: 3.096733e-08
numerical: -2.333424 analytic: -2.333425, relative error: 4.945110e-09
numerical: 0.175838 analytic: 0.175838, relative error: 3.612075e-07
numerical: 2.708760 analytic: 2.708760, relative error: 1.016469e-08
numerical: -1.330195 analytic: -1.330195, relative error: 4.143655e-09
numerical: 1.351909 analytic: 1.351909, relative error: 3.176034e-08
numerical: 1.382078 analytic: 1.382078, relative error: 7.886105e-08
numerical: -1.409880 analytic: -1.409880, relative error: 1.147507e-08
numerical: -0.446422 analytic: -0.446422, relative error: 2.719596e-08
numerical: 1.766858 analytic: 1.766858, relative error: 1.441881e-08
numerical: 0.292911 analytic: 0.292911, relative error: 1.276280e-07
numerical: 0.541392 analytic: 0.541392, relative error: 1.585247e-07
numerical: -1.256063 analytic: -1.256063, relative error: 1.622886e-08
numerical: -0.731748 analytic: -0.731748, relative error: 1.069327e-08
```

[16]:

```python
# Now that we have a naive implementation of the softmax loss function and its␣
↪gradient,
# implement a vectorized version in softmax_loss_vectorized.
# The two versions should compute the same results, but the vectorized version␣
↪should be
# much faster.
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 cs231n.classifiers.softmax import softmax_loss_vectorized
tic = time.time()
loss_vectorized, grad_vectorized = softmax_loss_vectorized(W, X_dev, y_dev, 0.
↪000005)
toc = time.time()
print('vectorized loss: %e computed in %fs' % (loss_vectorized, toc - tic))

# As we did for the SVM, we use the Frobenius norm to compare the two versions
# of the gradient.
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: 2.355952e+00 computed in 0.069077s
vectorized loss: 2.355952e+00 computed in 0.010538s
Loss difference: 0.000000
Gradient difference: 0.000000
```

[23]:

```python
# Use the validation set to tune hyperparameters (regularization strength and
# learning rate). You should experiment with different ranges for the learning
# rates and regularization strengths; if you are careful you should be able to
# get a classification accuracy of over 0.35 on the validation set.

from cs231n.classifiers import Softmax
results = {}
best_val = -1
best_softmax = None

################################################################################
# TODO:                                                                        #
# Use the validation set to set the learning rate and regularization strength. #
# This should be identical to the validation that you did for the SVM; save    #
# the best trained softmax classifer in best_softmax.                          #
################################################################################

# Provided as a reference. You may or may not want to change these␣
↪hyperparameters
learning_rates = np.linspace(1e-7, 1e-6, 5)
regularization_strengths = np.linspace(2.5e3, 5e3, 5)

# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
for lr in learning_rates:
    for reg in regularization_strengths:
        model = Softmax()
        model.train(X_train, y_train, lr, reg, num_iters=500)
        y_pre_train, y_pre_val = np.mean(model.predict(X_train) == y_train), np.
↪mean(model.predict(X_val) == y_val)
        results[(lr, reg)] = (y_pre_train, y_pre_val)
        if y_pre_val > best_val:
            best_val = y_pre_val
            best_softmax = model
# *****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)
```

```
lr 1.000000e-07 reg 2.500000e+03 train accuracy: 0.211000 val accuracy: 0.202000
lr 1.000000e-07 reg 3.125000e+03 train accuracy: 0.209551 val accuracy: 0.226000
lr 1.000000e-07 reg 3.750000e+03 train accuracy: 0.204755 val accuracy: 0.224000
```

```
lr 1.000000e-07 reg 4.375000e+03 train accuracy: 0.209408 val accuracy: 0.205000
lr 1.000000e-07 reg 5.000000e+03 train accuracy: 0.221571 val accuracy: 0.234000
lr 3.250000e-07 reg 2.500000e+03 train accuracy: 0.298327 val accuracy: 0.284000
lr 3.250000e-07 reg 3.125000e+03 train accuracy: 0.305551 val accuracy: 0.300000
lr 3.250000e-07 reg 3.750000e+03 train accuracy: 0.319980 val accuracy: 0.327000
lr 3.250000e-07 reg 4.375000e+03 train accuracy: 0.332857 val accuracy: 0.338000
lr 3.250000e-07 reg 5.000000e+03 train accuracy: 0.341490 val accuracy: 0.387000
lr 5.500000e-07 reg 2.500000e+03 train accuracy: 0.349714 val accuracy: 0.347000
lr 5.500000e-07 reg 3.125000e+03 train accuracy: 0.361327 val accuracy: 0.372000
lr 5.500000e-07 reg 3.750000e+03 train accuracy: 0.364633 val accuracy: 0.371000
lr 5.500000e-07 reg 4.375000e+03 train accuracy: 0.364531 val accuracy: 0.372000
lr 5.500000e-07 reg 5.000000e+03 train accuracy: 0.366469 val accuracy: 0.384000
lr 7.750000e-07 reg 2.500000e+03 train accuracy: 0.374694 val accuracy: 0.374000
lr 7.750000e-07 reg 3.125000e+03 train accuracy: 0.370551 val accuracy: 0.392000
lr 7.750000e-07 reg 3.750000e+03 train accuracy: 0.380531 val accuracy: 0.382000
lr 7.750000e-07 reg 4.375000e+03 train accuracy: 0.363490 val accuracy: 0.384000
lr 7.750000e-07 reg 5.000000e+03 train accuracy: 0.374163 val accuracy: 0.385000
lr 1.000000e-06 reg 2.500000e+03 train accuracy: 0.383918 val accuracy: 0.392000
lr 1.000000e-06 reg 3.125000e+03 train accuracy: 0.373224 val accuracy: 0.378000
lr 1.000000e-06 reg 3.750000e+03 train accuracy: 0.373388 val accuracy: 0.377000
lr 1.000000e-06 reg 4.375000e+03 train accuracy: 0.374143 val accuracy: 0.363000
lr 1.000000e-06 reg 5.000000e+03 train accuracy: 0.361531 val accuracy: 0.377000
best validation accuracy achieved during cross-validation: 0.392000
```

[24]:

```
# evaluate on test set
# Evaluate the best softmax on test set
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, ))
```

```
softmax on raw pixels final test set accuracy: 0.384000
```

Inline Question 2 - *True or False*

Suppose the overall training loss is defined as the sum of the per-datapoint loss over all training examples. It is possible to add a new datapoint to a training set that would leave the SVM loss unchanged, but this is not the case with the Softmax classifier loss.

$Your Answer:$True

$Your Explanation:$ Because SVM classifier is about its margin and the softmax classifier is about probability. If we add a new datapoint, maybe the loss of this point would be zero when the right label's score of this point is margin Δ larger than than others.
But for softmax, add a new data point will cause a new loss $-y_i log(y_i)$

[25]:

```
# Visualize the learned weights for each class
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)
```

```python
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])
```