softmax

February 10, 2020

1 Softmax Classifier

This exercise guides you through the process of classifying images using a Softmax classifier. As part of this you will:

- Implement a fully vectorized loss function for the Softmax classifier
- Calculate the analytical gradient using vectorized code
- Tune hyperparameters on a validation set
- Optimize the loss function with Stochastic Gradient Descent (SGD)
- Visualize the learned weights

[127]:
```
# start-up code!
import random

import matplotlib.pyplot as plt
import numpy as np

%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
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[128]:
```
from load_cifar10_tvt import load_cifar10_train_val

X_train, y_train, X_val, y_val, X_test, y_test = load_cifar10_train_val()
print("Train data shape: ", X_train.shape)
print("Train labels shape: ", y_train.shape)
print("Val data shape: ", X_val.shape)
print("Val labels shape: ", y_val.shape)
```

```python
print("Test data shape: ", X_test.shape)
print("Test labels shape: ", y_test.shape)
```

```
Train, validation and testing sets have been created as
 X_i and y_i where i=train,val,test
Train data shape:  (3073, 49000)
Train labels shape:  (49000,)
Val data shape:  (3073, 1000)
Val labels shape:  (1000,)
Test data shape:  (3073, 1000)
Test labels shape:  (1000,)
```

Code for this section is to be written in `cs231n/classifiers/softmax.py`

[144]:
```python
# Now, implement the vectorized version in softmax_loss_vectorized.

import time

from cs231n.classifiers.softmax import softmax_loss_vectorized

# gradient check.
from cs231n.gradient_check import grad_check_sparse

W = np.random.randn(10, 3073) * 0.0001

tic = time.time()
loss, grad = softmax_loss_vectorized(W, X_train, y_train, 0.00001)
toc = time.time()
print("vectorized loss: %e computed in %fs" % (loss, toc - tic))

# 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)))

f = lambda w: softmax_loss_vectorized(w, X_train, y_train, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

```
vectorized loss: 2.383807e+00 computed in 0.147605s
loss: 2.383807
sanity check: 2.302585
numerical: -2.929964 analytic: -2.929964, relative error: 4.616225e-09
numerical: -3.135085 analytic: -3.135085, relative error: 2.552106e-08
numerical: -0.570058 analytic: -0.570058, relative error: 1.404715e-07
numerical: 0.337006 analytic: 0.337006, relative error: 4.889899e-08
numerical: 1.682049 analytic: 1.682049, relative error: 2.496364e-08
numerical: -0.297519 analytic: -0.297519, relative error: 8.559922e-08
numerical: 0.558544 analytic: 0.558544, relative error: 2.837774e-08
numerical: 0.629610 analytic: 0.629610, relative error: 2.700313e-08
```

```
numerical: -2.363835 analytic: -2.363835, relative error: 1.094664e-08
numerical: -1.217689 analytic: -1.217689, relative error: 2.245683e-09
```

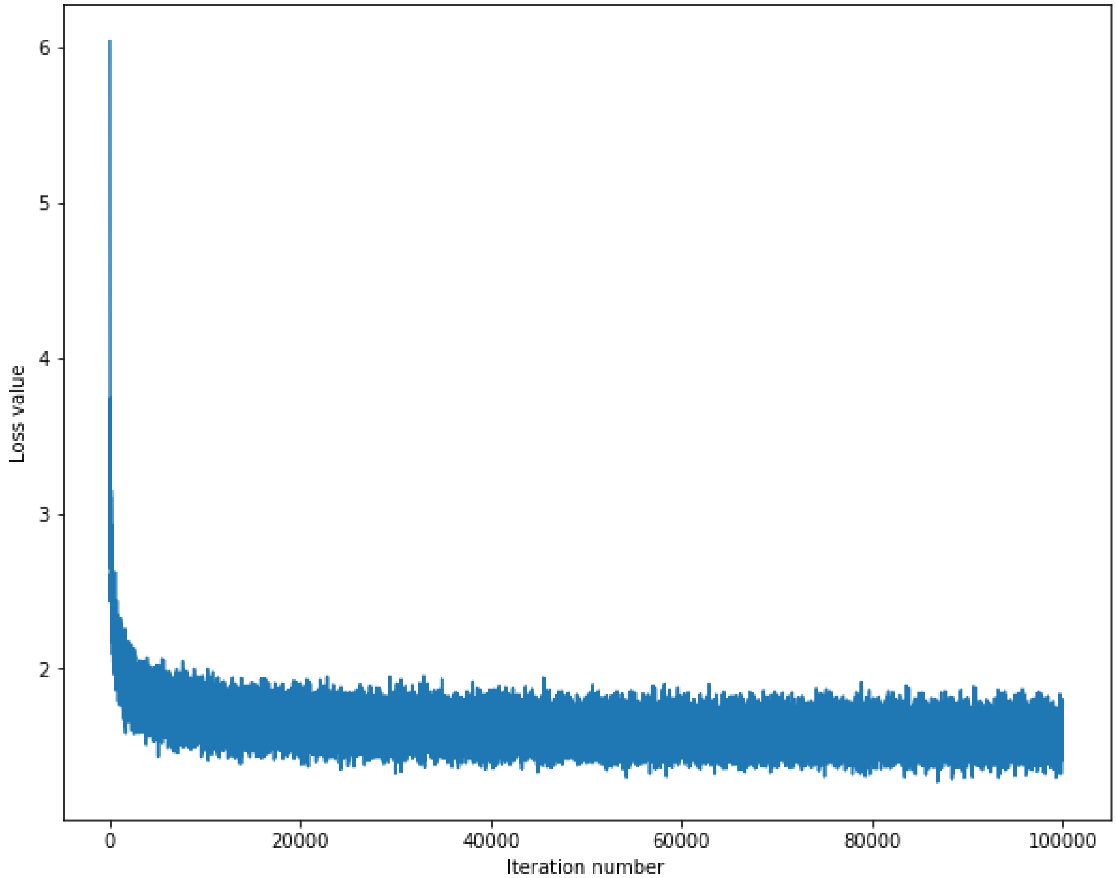
Code for this section is to be written incs231n/classifiers/linear_classifier.py

[219]:
```
# Now that efficient implementations to calculate loss function and gradient of␣
↪the softmax are ready,
# use it to train the classifier on the cifar-10 data
# Complete the `train` function in cs231n/classifiers/linear_classifier.py

from cs231n.classifiers.linear_classifier import Softmax

classifier = Softmax()
loss_hist = classifier.train(
    X_train,
    y_train,
    learning_rate=1e-6,
    reg=1e-5,
    num_iters=100,
    batch_size=200,
    verbose=False,
)
# Plot loss vs. iterations
plt.plot(loss_hist)
plt.xlabel("Iteration number")
plt.ylabel("Loss value")
```

[219]:
```
Text(0, 0.5, 'Loss value')
```



[220]:
```
# Complete the `predict` function in cs231n/classifiers/linear_classifier.py
# Evaluate on test set
y_test_pred = classifier.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: 0.379000
```

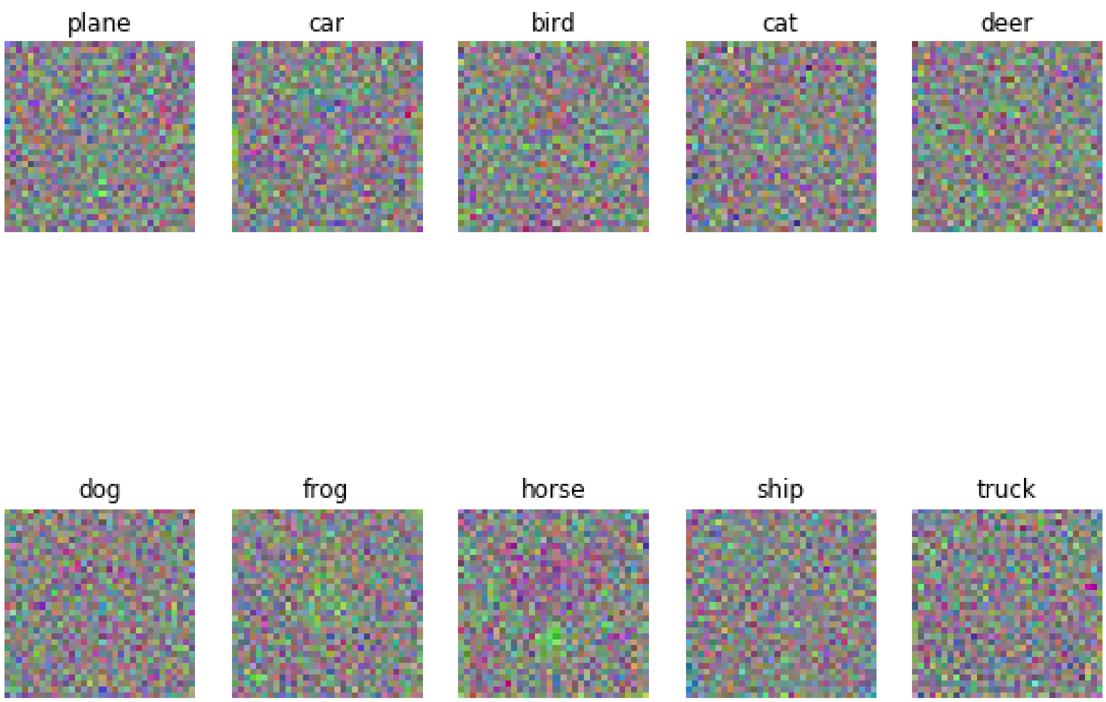
[221]:
```
# Visualize the learned weights for each class
w = classifier.W[:, :-1]  # strip out the bias
w = w.reshape(10, 32, 32, 3)

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])
```



[ ]:

```
```