03-softmax

June 14, 2020

1 Atividade 03: Classificador Softmax

Complete e entregue toda essa atividade (incluindo suas saídas e qualquer código adicional que você desenvolva) juntamente com a submissão de seu trabalho prático. Maiores detalhes podem ser vistos na página da disciplina.

Esta atividade é análoga a feita anteriormente sobre SVM. Você irá:

- implementar uma **função de perda** (**loss function**) de forma completamente paralela para o classificador Softmax
- implementar a expressão para seu **gradiente analítico** de forma completamente paralela
- **verificar sua implementação** utilizando gradiente numérico
- usar um conjunto de validação para **ajustar a taxa de aprendizado e regularização**
- **otimizar** a função de perda com **SGD**
- **visualizar** os pesos finais que foram obtidos

```
In [6]: import random
        import numpy as np
        from dl.data_utils import load_CIFAR10
        import matplotlib.pyplot as plt

        from __future__ import print_function

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

        %load_ext autoreload
        %autoreload 2

        print('Okay!')
```

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

In [7]:
```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000, num_dev=500
    """
    Carrega CIFAR-10 dataset a partit do disco e realiza preprocessamento para preparar
    os dados para o classificador linear. Estes são os mesmos passos usados para o model
    SVM, porém condensado em uma única função.
    """
    # Carregga os dados CIFAR-10 brutos
    cifar10_dir = 'dl/datasets/cifar-10-batches-py'
    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # Subdivide os dados em conjuntos
    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]

    # Preprocessamento: redimensiona as imagens de matrizes para vetores
    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))

    # Normaliza os dados: subtrai a imagem média
    mean_image = np.mean(X_train, axis = 0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image
    X_dev -= mean_image

    # Acrescenta uma dimensão (viés/bias) e transforma em colunas
    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


# Usar a função definida acmina para obter os dados.
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)

        print('Okay!')
```

```
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,)
Okay!
```

1.1 Classificador Softmax

Seu código para está seção será todo escrito dentro do arquivo `dl/classifiers/softmax.py`.

```
In [8]: # Primeiramente, implemente a versão ingênua da função de perda softmax usando laços ani
        # Abra o arquivo dl/classifiers/softmax.py e implemente a função softmax_loss_naive.

        from dl.classifiers.softmax import softmax_loss_naive
        import time

        # Produz uma matrix pesos aleatórios e a usa para calcular a perda.
        W = np.random.randn(3073, 10) * 0.0001
        loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

        # Apenas como uma verificação, o valor de perda deve ser algo próximo a -log(0.1).
        print('loss: %f' % loss)
        print('sanity check: %f' % (-np.log(0.1)))

        print('Okay!')
```

```
loss: 2.319804
sanity check: 2.302585
Okay!
```

1.2 Pergunta 01:

Por que se deve esperar que a perda calculada seja próxima de -log(0.1)? Explique brevemente.

Sua Resposta: *Pois como o W é aleatório, e sua probabilidade é de 1/10, o valor é 10% (0.1)*

In [9]:
```
# Complete a implementação da função softmax_loss_naive, acrescentando uma
# versão (ingênua) de cálculo do gradiente que utilize laços aninhados.
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# Como foi feito para o SVM, use a estimativa numérica do gradiente como
# ferramenta de verificação e depuração.
# O valor do gradiente numérico deve ser próximo do obtido analiticamente.
from dl.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 ao SVM, faça outra verificação utilização um valor de regularização não nulo
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)

print('Okay!')
```

```
numerical: 0.506040 analytic: 0.506040, relative error: 1.872356e-09
numerical: -4.855755 analytic: -4.855756, relative error: 1.473418e-08
numerical: -0.600113 analytic: -0.600113, relative error: 5.381051e-09
numerical: -0.871733 analytic: -0.871733, relative error: 1.842650e-08
numerical: 1.909631 analytic: 1.909631, relative error: 2.660218e-08
numerical: -2.038483 analytic: -2.038483, relative error: 4.373347e-08
numerical: -1.360065 analytic: -1.360065, relative error: 6.311857e-08
numerical: 3.023074 analytic: 3.023074, relative error: 1.310259e-08
numerical: -1.094234 analytic: -1.094234, relative error: 3.624755e-08
numerical: 1.155066 analytic: 1.155066, relative error: 3.672406e-08
numerical: -2.525703 analytic: -2.525703, relative error: 3.909059e-08
numerical: -1.235736 analytic: -1.235736, relative error: 2.113284e-08
numerical: 1.798328 analytic: 1.798327, relative error: 3.627317e-08
numerical: -1.620961 analytic: -1.620961, relative error: 9.297531e-10
numerical: 2.709871 analytic: 2.709871, relative error: 2.286845e-08
numerical: -0.078640 analytic: -0.078640, relative error: 1.417349e-07
numerical: 1.582100 analytic: 1.582100, relative error: 1.823098e-08
numerical: -2.555849 analytic: -2.555850, relative error: 6.215271e-09
numerical: 1.040267 analytic: 1.040267, relative error: 1.021206e-08
numerical: 1.449088 analytic: 1.449088, relative error: 5.103152e-08
Okay!
```

In [10]:
```
# Agora que se tem uma versão básica de para cálculo da função de perda e de seu gradie
# você deve implementar uma versão vetorial/paralela na função softmax_loss_vectorized.

# Novamente, os valores devem coincidir porém a versão vetorial deve ser mais rápida.
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 dl.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))

        # Como para o SVM, utiliza-se a norma de Frobenius para comparar as duas versões
        # de gradiente.
        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)

        print('Okay!')
```

```
naive loss: 2.319804e+00 computed in 0.158620s
vectorized loss: 2.319804e+00 computed in 0.003905s
Loss difference: 0.000000
Gradient difference: 0.000000
Okay!
```

In [13]:
```
        # Utilize o conjunto de validação para ajustar os hiperparâmetros (taxa de
        # aprendizado e regularização). Você deve experimentar diferentes intervalos
        # para ambos. Se você proceder corretamente deve ser capaz de obter uma acurácia
        # acima de 0.35 no conjunto de validação.

        from dl.classifiers import Softmax
        results = {}
        best_val = -1
        best_softmax = None
        learning_rates = [1e-8, 1e-7, 2e-7]
        regularization_strengths = [1e4, 2e4, 3e4, 4e4, 5e4, 6e4, 7e4, 8e4, 1e5]

        ################################################################################
        # TODO:                                                                        #
        # Escrever o código que escolhe os melhores hiperparâmetros usando o conjunto  #
        # de validação. Deve ser muito similar ao código feito para o SVM.             #
        # Além disso, deve-se armazenar a melhor acurácia de validação em best_val     #
        # e o objeto Softmax que obteve esse resultado em best_softmax.                #
        ################################################################################

        iters = 2000
        for lr in learning_rates:
            for rs in regularization_strengths:
```

```
                softmax = Softmax()
                softmax.train(X_train, y_train, learning_rate=lr, reg=rs, num_iters=iters)

                y_train_pred = softmax.predict(X_train)
                acc_train = np.mean(y_train == y_train_pred)
                y_val_pred = softmax.predict(X_val)
                acc_val = np.mean(y_val == y_val_pred)

                results[(lr, rs)] = (acc_train, acc_val)

                if best_val < acc_val:
                    best_val = acc_val
                    best_softmax = softmax

        ################################################################################
        #                                FIM DE SEU CÓDIGO                             #
        ################################################################################

        # Exibe os resultados.
        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)

        print('Okay!')
```

```
lr 1.000000e-08 reg 1.000000e+04 train accuracy: 0.187612 val accuracy: 0.178000
lr 1.000000e-08 reg 2.000000e+04 train accuracy: 0.180020 val accuracy: 0.185000
lr 1.000000e-08 reg 3.000000e+04 train accuracy: 0.187551 val accuracy: 0.210000
lr 1.000000e-08 reg 4.000000e+04 train accuracy: 0.190714 val accuracy: 0.170000
lr 1.000000e-08 reg 5.000000e+04 train accuracy: 0.192857 val accuracy: 0.187000
lr 1.000000e-08 reg 6.000000e+04 train accuracy: 0.210102 val accuracy: 0.185000
lr 1.000000e-08 reg 7.000000e+04 train accuracy: 0.217755 val accuracy: 0.228000
lr 1.000000e-08 reg 8.000000e+04 train accuracy: 0.213122 val accuracy: 0.220000
lr 1.000000e-08 reg 1.000000e+05 train accuracy: 0.255449 val accuracy: 0.274000
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.355694 val accuracy: 0.392000
lr 1.000000e-07 reg 2.000000e+04 train accuracy: 0.359020 val accuracy: 0.373000
lr 1.000000e-07 reg 3.000000e+04 train accuracy: 0.347408 val accuracy: 0.358000
lr 1.000000e-07 reg 4.000000e+04 train accuracy: 0.336408 val accuracy: 0.342000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.332184 val accuracy: 0.347000
lr 1.000000e-07 reg 6.000000e+04 train accuracy: 0.319306 val accuracy: 0.332000
lr 1.000000e-07 reg 7.000000e+04 train accuracy: 0.322673 val accuracy: 0.331000
lr 1.000000e-07 reg 8.000000e+04 train accuracy: 0.311673 val accuracy: 0.331000
lr 1.000000e-07 reg 1.000000e+05 train accuracy: 0.300714 val accuracy: 0.318000
lr 2.000000e-07 reg 1.000000e+04 train accuracy: 0.374571 val accuracy: 0.392000
lr 2.000000e-07 reg 2.000000e+04 train accuracy: 0.355939 val accuracy: 0.376000
```

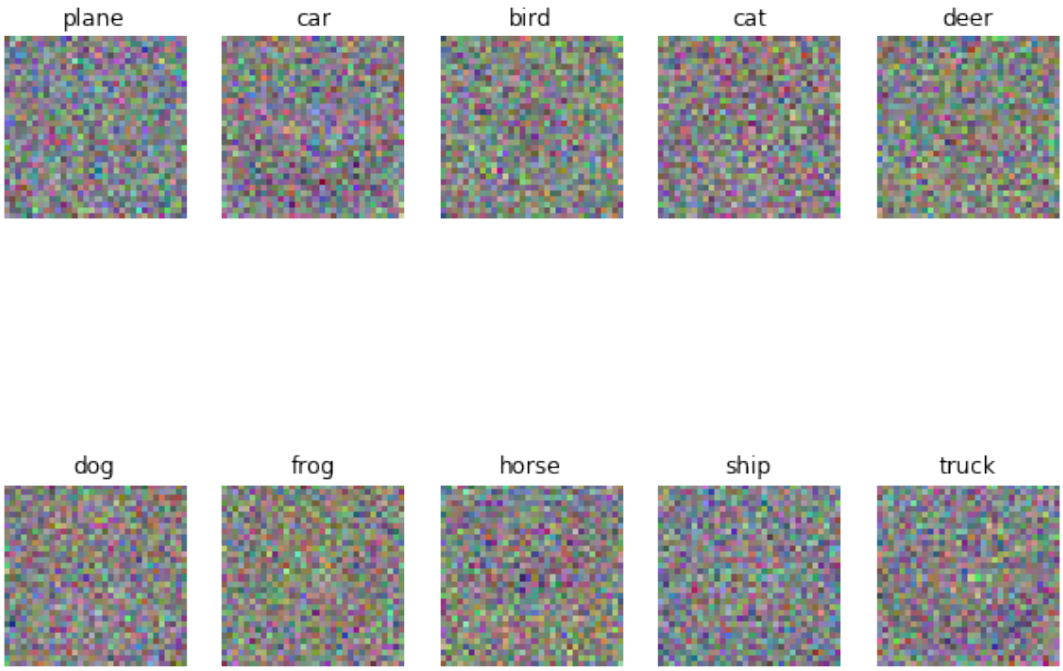
```
lr 2.000000e-07 reg 3.000000e+04 train accuracy: 0.348204 val accuracy: 0.362000
lr 2.000000e-07 reg 4.000000e+04 train accuracy: 0.327082 val accuracy: 0.345000
lr 2.000000e-07 reg 5.000000e+04 train accuracy: 0.332694 val accuracy: 0.350000
lr 2.000000e-07 reg 6.000000e+04 train accuracy: 0.320265 val accuracy: 0.322000
lr 2.000000e-07 reg 7.000000e+04 train accuracy: 0.313592 val accuracy: 0.328000
lr 2.000000e-07 reg 8.000000e+04 train accuracy: 0.310490 val accuracy: 0.335000
lr 2.000000e-07 reg 1.000000e+05 train accuracy: 0.317429 val accuracy: 0.329000
best validation accuracy achieved during cross-validation: 0.392000
Okay!
```

In [14]:
```
# Avalia o melhor modelo Softmax no conjunto de teste
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, ))

print('Okay!')
```

```
softmax on raw pixels final test set accuracy: 0.354000
Okay!
```

In [16]:
```
# Visualiza os pesos obtidos para cada classe.
w = best_softmax.W[:-1,:] # remove o viés/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', 'truc
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Redimensiona os pesos para o intervalo entre 0 e 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])

print('Okay!')
```

```
Okay!
```


plane
car
bird
cat
deer
dog
frog
horse
ship
truck