Assignment 1 Bonus Deep Learning

Jan Alexandersson

March 27, 2020

Possible improvements

Using the settings **lambda = 1, n_epochs = 40, n_batch = 100, eta = 0.001** we got test accuracy of 0.3778 without any of the suggested improvements. We then implemented the following possible improvements:

- Shuffle the order of your training examples at the beginning of every epoch.
- Decaying the learning rate with a facor of 0.9 after each epoch.
- Train for a longer time and use your validation set to make sure you don't overfit or to keep a record of the best model before you begin to overfit.

	Test accuracy
Without any improvements	0.3778
Shuffled order	0.3602
Decaying Eta	0.3733
Longer training	0.3779

We see in the table above that we only got some minimal improvement on our test accuracy when we trained for a longer time and we actually performed slightly worse with the other methods. However, this may just be random differences since we initialize the W matirx and b with random numbers. I also tried this with some other values of the parameters but got similar results. However, surprisingly, the test accuracy seemed to be decrase when shuffeling the order of my training examples before each epoch, which might indicate that the implementation of the shuffeling is incorrect.

SVM Loss

We used the same parameters as in the non-bonus assignment and used for all runs **n_batches = 100** and **n_epochs = 40** and varied the values of **eta** and **lambda**. The results are presented in the table of the test accuracys below.

Parameters	Cross-entropy loss	SVM multi-class loss
Eta = 0.1, lambda = 0	0.2751	0.2896
Eta = 0.001, lambda = 0	0.3838	0.3879
Eta = 0.001, lambda = 0.1	0.3898	0.3918
Eta = 0.001, lambda = 1	0.3771	0.3767

We can see that there are no large differences between using cross-entropy loss compared to SVM multi-class loss with regards to test accuracy. For some parameters SVM multi-class loss outperformed cross-entropy loss, but in the last row of the table above we can see that the cross-entropy loss slightly outperformed SVM multi-class loss. However, the differences are so small that they can be regarded as random.

Here are some code used in this exercise:

```
def ComputeCostSVM(X, Y, W, b, lamb):
   N = X.shape[1]

   s = EvaluateClassifier(X, W, b)
   sc = s.T[np.arange(s.shape[1]), np.argmax(Y, axis=0)].T

   marg = np.maximum(0, s - np.asarray(sc) + 1)
   marg.T[np.arange(N), np.argmax(Y, axis=0)] = 0

   mcsvm_loss = Y.shape[0] * np.mean(np.sum(marg, axis=1))

   cost = 1/N * mcsvm_loss + 0.5 * lamb * np.sum(W**2)

   return cost, marg


def ComputeGradientsSVM(X, Y, W, b, lamb):

   N = X.shape[1]

   _, marg = ComputeCostSVM(X, Y, W, b , lamb)

   bi = marg
   bi[marg > 0] = 1
   bi_sum_rows = np.sum(bi, axis=0)

   bi.T[np.arange(N), np.argmax(Y, axis=0)] = -bi_sum_rows.T

   grad_W = np.dot(bi, X.T) / N + lamb * W

   grad_b = np.reshape(np.sum(bi, axis=1) / bi.shape[1], b.shape)
   return grad_W, grad_b
```