Programming Assignment #2 of WI 21 CSE 251B

Fangzhou Ai
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
faai@eng.ucsd.edu

Yue Qiao
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
yuq021@eng.ucsd.edu

Abstract

This paper is a report for the programming assignment 1 for Lecture CSE 251B.

1 Load and preprocess the data

Load data Data loading cold be done by easily calling the given function load_data();

Preprocess data After loading the data, we get a dictionary contains 4 different kinds of cars, the keys are the cars' names and values are a list contains a series of images, we also get a dictionary cnt which include how many images we have within each car's category. To turn the data into a more convenient form for training, we did the following things: 1. turn the list into a Numpy array for more fancy matrix operations and 2. we write our own label encoding program to turn each category into a vector (eg. 'Minivan' would be [0, 1, 0, 0]).

2 Cross validation procedure

Split the data Notice that ideally we want to split the data in a way that each type of car would evenly distributed within training set, validations set and test set, by looking at our database, we found that each category has almost the same number of pictures(Convertible: 149 # of images, Minivan: 148 # of images, Pickup: 150 # of images, Sedan: 150 # of images), therefore we choose to split each set into k parts then combine them together to form the new data set, in which case we can guarantee that each part would evenly contains all types of cars.

Cross validation To make sure each part would be test set once and validation set once, we maintained an index array [0, 1,...., k - 1], the first element indicate the index of test set, the second element indicate the index of validations set, the rest elements indicate the training set, after each training procedure, we let each element pump 1 then mod k, so the first time the test set would be the part 0, validation set is part 1, training set is 2 to k -1, the second time the test set would be the part 1, validation set is part 2, training set is [3, 4,..., k - 1, 0], at the last training procedure the test set would be the part k - 1, validation set is part 0, training set is 1 to k - 2. Hence each set has been set to test set and validations set once.

Preprint. Under review.

Table 1: Mean and std of different set.

Set	Mean	STD
Training	4.65502e-20	0.999999
Validation	-0.00031	0.860112
test	-0.00054	0.833020

Figure 1: Loss and accuracy on training set and holdout set

3 Principal components analysis

Since the original data have $200 \times 300 = 60000$ features, we use principal components analysis (PCA) to reduce the dimensions and remain the most useful features. In most of our experiments, we find that around 100 features will be a sweet point for training.

Implement PCA The verified mean and std on each set is shown in Table 1.

4 Logistic regression

Logistic model is a very simple model which has only one output and use sigmoid as the activation function. It can only identify 2 different objects while it's simplicity make it a very popular toy model to understand the principle behind machine learning. Here we use python with only Numpy library to build a trivial program to implement this model's key idea.

4.1 (a)

The logistic regression code is shown in Listing 1. We only need one output because we only have 2 categories, if we assume the output is the probability of the first category, then the probability for the second one is naturally 1 - p.

Listing 1: Logistic regression implementation

```
import numpy as np
def simple_logistic_model(w, input):
''' logistic model withou hidden layer
Args:
        input, which dimention is M * (1 + d), means M pics
        each pixel number is d, appnded by 1
        w, parameters, dimention of d + 1 * 1
Returns:
        x, dimention of M * 1
'''
        x = np.dot(input, w)
        x = 1/(1 + np.exp(-x))
        return x
```

4.2 (b)

The loss of the validation set is shown in Figure 1. And the accuracy on test set is around 63.33%, the validation loss clearly shows the overfiting problem here that loss goes down quickly at first then goes up. Here we have 29 PCA components. Looking at the PCA components(the first 4 are shown in Figure 2), we found that since the objects are not aligned, it's very hard for PCA to get accurate car's shape, thus not able to achieve high accuracy.

Figure 2: First 4 PCA components

Figure 3: Mean and std of accuracy and loss, classification between Convertible and Minivan

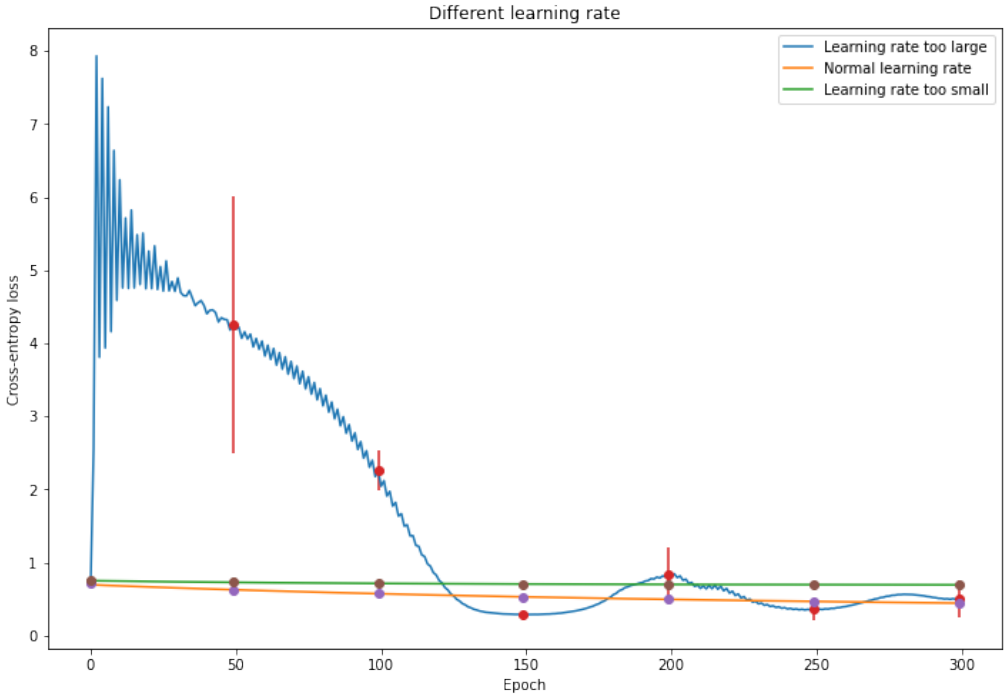

Figure 4: Different learning rate effects on training loss

4.3 (c)

By alternating the data source from resized to aligned, we see a a huge improvement on accuracy, the accuracy we achieved here is 83.5% with 300 epochs. The mean and std of accuracy and loss is shown in Figure 3. We also try different laearning rate here, one is too large, one is too small and a normal one, the difference is shown in Figure 4. As we can see that a large learning rate may lead to strong oscillation while a small one would lead to a slow converging speed.

4.4 (d)

The classification accuracy between category Pickup and Sedan is 82.67%, namely, we didn't notice any significant differences between different category classification tasks, Which is reasonable since this logistic model is simple yet powerful enough to generalized to similar objects. The mean and std of training loss is shown in Figure 5.

5 Softmax regression

Softmax regression is an improved version of logistic regression which supports multi-class classification. To label multiple classes, we use one hot encoding to encode the labels. The forward part code of softmax regression is shown in Listing 2

Listing 2: Softmax regression implementation

Figure 5: Mean and std of accuracy and loss, classification between Pickup and Sedan

Figure 6: First 4 PCA components

Figure 7: Confusion matrix for test set

```
    import numpy as np
def softmax(w, x):
    '''
    softmax model forward

    Args

            x: input data, which dimention is (M, d), means M pics,
            each pixel number is d
            w: parameters, dimention of d + 1 * number_of_classes

    Returns

    y: dimention of (M, number_of_classes)
    '''
    x = w @ x.T
    return (np.exp(x) / np.sum(np.exp(x), axis=0)).T
```

5.1 (a)

The top four principal components is shown in Figure 6. From the principal components we can see that they are different kinds of cars. The confusion matrix of the test set is shown in Figure 7. From the confusion matrix we can see that our classfier is not very accurate. We archived 58% accuracy on the test set. Loss and accuracy for training and holdout data is shown Figure 8.

5.2 (b)

Stochastic gradient descent is a common technique to accelerate the converge of the gradient decent. The comparsion of the loss curve of batch gradient decent and stochastic gradient descent is shown in Figure 9. From our result there is no much difference between batch gradient decent and stochastic gradient decent. One of the reasons may be that the problem is too easy to show the difference between batch gradient decent and stochastic gradient decent.

5.3 (c)

The weights for four car types are shown in Figure 10. The images of four weights roughly shows the outline of each car type. This is reasonable since that is how the softmax regression works.

6 Individual contributions to the project

Fangzhou Ai Cross Validation, Principal Components Analysis, Logistic Regression

Yue Qiao Cross Validation, Logistic Regression, Softmax Regression

Figure 8: Mean and std of accuracy and loss, classification between all four classes.

Figure 9: Mean and std of loss, batch gradient decent and stochastic gradient descent

Figure 10: Weights for four car types