Deep Learning and Practice HW1

0751901 YengHung Ou

2019 年 3 月 22 日

1 Introduction

The most common and basic learning category in the field of machine learning is classifier. The goal of a classifier is: *with a pre-defined learning network, given the inputs and the corresponding output labels, using forward propagation to get a prediction, then calculate the loss between the prediction and the label for backward propagation to calculate the partial derivatives of loss and each weight, which represents the direction and strength to adjust the weight, then multiply the learning rate and the derivative and update the weight with the original weight minus the multiplication output. Repeat the process until the loss is low enough.* The purpose of this experiment is: *Under the constraint that only Python build-in package and Numpy package available, build a simple classification network and use it to train the classification of two kinds of data.*

2 Experiment setups

2.1 Sigmoid functions

Since the relationship between the input data and its label is not linear sometimes, in order to let the network learn enough property of the data, adding an activation function after the output of every layer can increase the

non-linearity of the model.
The activation function used in this homework is Sigmoid function, the math equation is

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Its property is that the output is between 0 and 1, and it's strictly increasing. Generally we use it in some simpler classification neural network, because there's more detailed features need to be found in a more complicated network, output between 0 and 1 makes it not obvious enough. The derivative of Sigmoid is used when doing backward propagation, its math equation is also simple, the equation is

$$f'(x) = f(x) * (1 - f(x) \tag{2}$$

The f(x) here is the output of 1

2.2 Neural Network

The neural network used in this experiment has an input layer which contains N 2x1 data, where N means the number of data and each data has 2 values representing its location in a 2-dimension plane, two fully connected layers with four neurons in each and activation function after each layer's output, and an output layer with one output representing the prediction.
The hyper-parameter I chose to train this network is: learning rate=0.5, epoch = 1000, and the loss function is

$$L(y) = \frac{1}{2} * (label - y)^2 \tag{3}$$

The reason for choosing this is because the 0.5 eliminate the square after calculating derivatives.。

2.3 Backward propagation

The output of every neuron when doing forward propagation can be calculated by multiplying the output of each neuron in last layer with its

corresponding weight and pass the result through the activation function. A similar way can be used in backward propagation. The i-th Neuron's output in the n-th layer can be defined as

$$output_{n,i} = \sigma(\sum_{j=1}^{m} x_{n-1,j} * w_{n,j} + b_{n,i}) \tag{4}$$

where x in the right means the j-th neuron's output over m neurons in last layer, and w and b means its corresponding weight and bias. On the other hand, the equation when doing backward propagation becomes
Output Layer:

$$delta_{outputlayer} = (label - prediction) * \sigma^{-1}(output_{outputlayer}) \tag{5}$$

Hidden Layer:

$$delta_{n,i} = \sum_{j=1}^{m} delta_{n+1,j} * w_{n+1,i} * \sigma^{-1}(output_{n+1,j}) \tag{6}$$

3 Result

3.1 Screenshot and comparison figure

```
epoch 0 loss -0.48965028676487815
epoch 100 loss -0.004091614839203584
epoch 200 loss -0.002143377732487599
epoch 300 loss -0.0009798270595109879
epoch 400 loss -0.0005782938966039173
epoch 500 loss -0.0003948441047807545
epoch 600 loss -0.00030331516860305287
epoch 700 loss -0.00024826482832983885
```

图 1: Training loss for linear data

```
2.41660185638e-05
0.999986641242
0.988389870954
3.02956053332e-05
2.6305104631e-05
0.999984066484
0.999973011533
0.999985926058
0.999985339423
0.999977586571
2.92860227539e-05
0.999987680601
0.999984041792
2.37657965402e-05
0.999984375011
2.37934713305e-05
0.9999838403
0.999984319701
```

图 2: Prediction of linear data

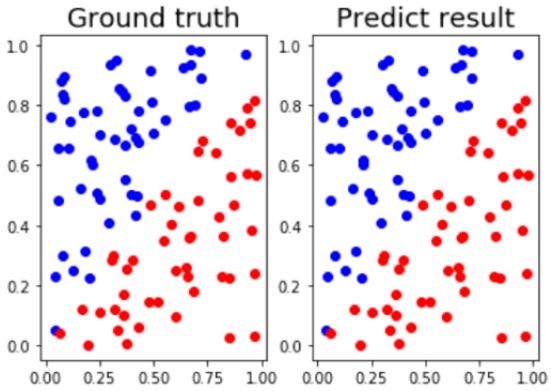

图 3: Comparison of ground truth and prediction

```
epoch 0 loss 0.5397057071484019
epoch 100 loss 0.20532434766812657
epoch 200 loss 0.06655074086127855
epoch 300 loss 0.015671382446190596
epoch 400 loss 0.002741418063618606
epoch 500 loss 0.0015801302634509407
epoch 600 loss 0.0011028775207317931
epoch 700 loss 0.0008481097310985009
```

图 4: Training loss for xor data

```
0.00243870642974
0.999786172059
0.00301789052604
0.999762215175
0.00688796749268
0.999678973365
0.0214202388452
0.998986620057
0.0485817481887
0.922456599734
0.0592635153408
0.044677121211
0.943357908252
0.0264501763073
0.999512904735
0.0147185965107
0.999677285976
0.00847944518594
0.999568970585
0.00524785461771
0.999305641664
```

图 5: Prediction of xor data

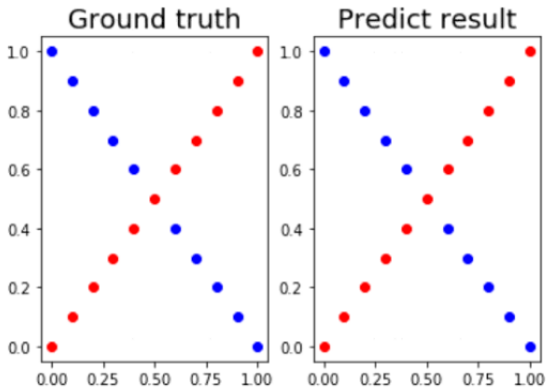

图 6: Comparison of ground truth and prediction

3.2 Extra thing to present

In order to test whether the network could still distinguish the dataset under current setting of learning rate and number of hidden layer and the neurons in each layer, The test method is: If current epoch makes zero prediction error, increase 100 data to the data-set, otherwise add 200 epoch until we reach 10000 in epoch or 1000 in number of data-set.
The definition of prediction error is:

$$abs(prediction - label) > 0.1 \tag{7}$$

The test result as following

epoch	number of data
1000	100
1200	200
1600	300
1600	400
2000	500
2600	600
3600	700
3600	800
4000	900
4000	1000

From the result we know that the network could still find the pattern of the data, and more epoch is needed when training with larger data-set, which is logically.

4 Discussion

I think this exercise is very educational since it not only let us understand more about the math behind machine learning and appreciate those who

develop those deep learning framework such as caffe, tensorflow, pytorch...etc, since it really took us lots of time to build just a simple neural network structure. I'm also curious about is there any relationship between learning rate and the choice of loss function, since it seems that different loss function needs a different learning rate with other parameters unchanged.