Multi-layer Perceptron

计 83　肖易佳

October 2019

1 Preparation

1.1 Principals

Basic knowledge We can learn from previous programming exercises that sometimes simply using perceptron is not enough, so we consider adding more layers(also known as hidden layers) to perceptron model, to make it more capable of dealing with complex data.

Parameters and Notations For each layer(except for the input and output layer), there are two parameters - a matrix M_{i*j} and a vector b_{i*1}, and two temporary vectors - r_{i*1} and h_{i*1}; besides, we mark the input vector of hidden-layer i as H_i(a vector). In every layer:

$$r_{i*1} = M_{i*j}H_i + b_{i*1}$$

And in this experiment, we use **tanh** as the activation function.

$$h_{i*1} = tanh(r_{i*1})$$

Here, the activation function working on a vector means $h_i = tanh(r_i)$

Derivative In my experiment, I employed a 3-layer perceptron; and after several trials, I find 2 ways are feasible, one is the traditional backward propagation, and the other is deriving the derivative by hand(only one hidden layer makes the work relatively easy).

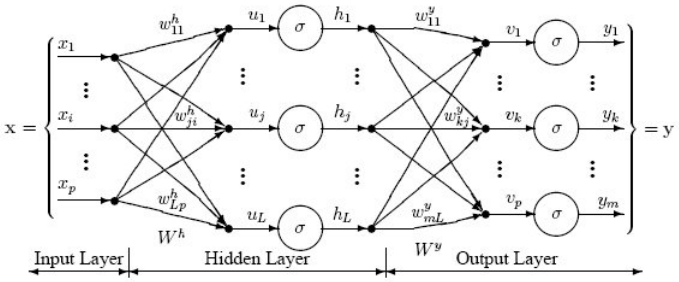

图 1: Multi-layer Perceptron

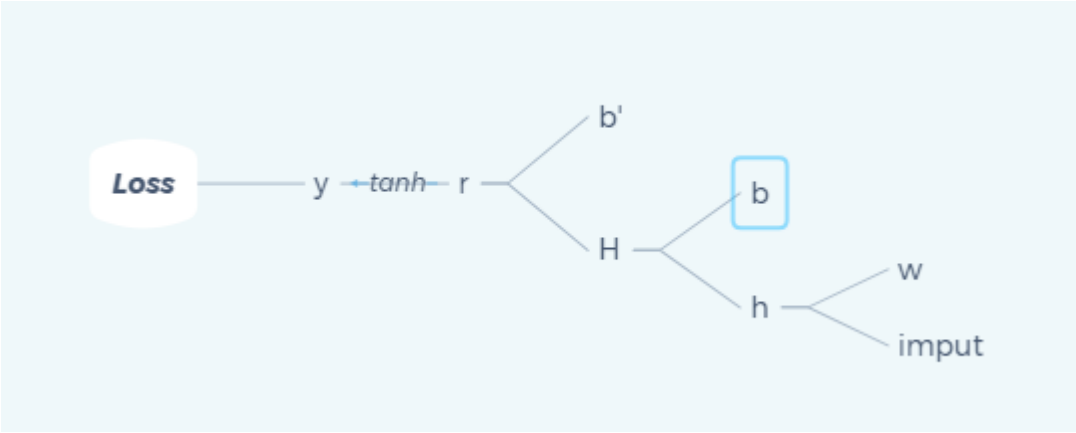

图 2: My method of derivation

Learning Just like the basic perceptron, MLP also utilizes the gradient descend method. The difference lies in the way of derivation: since multi layer is more complex to find derivative, we need to adopt another method. MLP uses backward propagation to calculate the derivative. In this way, it can save many unnecessary calculation.
As for the updating formula, it stays the same.

P.S. This tree is optional in training, we can still find the derivative - $\frac{\partial LOSS}{\partial w_{ij}}$ and $\frac{\partial LOSS}{\partial b_{ij}}$ by using the chain rule. And as for MLPs with more than one hidden layers, manual derivation is no longer feasible, we need to

use **Backward Propagation** to lower down the complexity.(Explained in detail in the previous homework)

1.2 Procedure

Derivative Using a tree structure, it will be easier to find the target derivative. And as for those MLPs with only one or two hidden layers, I think deriving the derivative by hand is possible.

$$y(x) = tanh(w' * tan(w * x + b) + b')$$

Initialization Having established the math foundation(derivative part), what we need to do is initialize all the matrix and vectors. To make sure gradient descend works, I initialize these parameters with random values.

Updating After constructing and initializing the models, we can feed data to it and get the output $\hat{y}$. Next we need to find $\Delta y = y - \hat{y}$, here Δy is the **loss function**, and our task is minimizing the loss function.

2 Programming Analysis

Packages numpy, tensorflow, pandas, random and etc.

Training Steps

1. initialize matrix and vector
2. load and feed data into the model
3. find ΔJ and use **back propagation** to update the parameters
4. loop until $\frac{\partial \Delta J}{\partial params} <$ threshold
5. validate the model on test set

Tensorflow We can also use tensorflow to construct an MLP, the principal is similar - adding hidden layers to the model, in this way, we have more hidden nodes in it(e.g. the input data is a 10-dim vector, then by using a 32 * 10 scale matrix M and a 32 * 1 scale bias vector b, there will be 32 nodes in the hidden layer).

3 Simple Analysis

Overview Intuitively, Multi-layer Perceptron are derived from perceptron, so the relationship among learning rate η, accuracy, loss function should be similar, so we can draw an analogy.

Analysis

1. Training period: When we change the learning rate (in my test, from 1e-5 1e-3), the error rate of MLPs with more layers is more stable than that of those with less.
   I think the reason is that more hidden layers can better extract the features of training data, and thus the model's performance improves with the increase of layers in a certain range(if there are too many layers, the result might be worse).

Core Codes for Visualization def visualization():
fig, ax = plt.subplots()
plt.scatter(range(len(Accu)), Accu, c= r, marker=dot)
plt.show()
fig.savefig(C:path + svmFig.eps, dpi=600, format= eps)

call the function:
visualization()

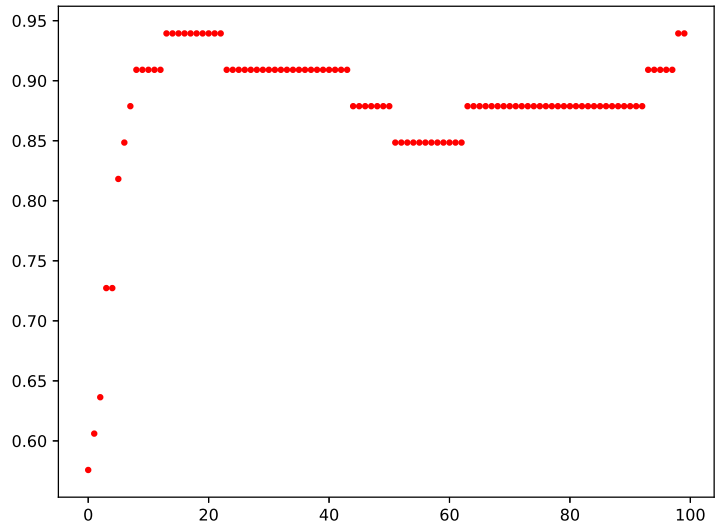

图 3: My accuracy - training epoch figure

4 Reference

1. Multi-layer Perceptron - Wikipedia
2. Machine Learning - Zhihua ZHou
3. Deep Learning - Ian Goodfellow, Yoshua Bengio and Aaron Courville
4. Introduction to AI - Andrew Ng - deeplearning.ai
5. Introduction to MLP - CSDN