HW2: Nonlinear Classifiers

Yanze Wang
Department of Computer Science
University of Southern California
Los Angeles, CA 90089 USA
yanzewan@usc.edu

Abstract

This document is about classifying four different kinds of classification tasks using Nonlinear Classifiers (specifically, a one-layer feed-forward network). The performance of a from-scratch implementation and a PyTorch implementation is compared in this document. Also a discussion about different hidden layer sizes, learning rates, epochs, number of words chosen per item is included. Different batch sizes are measured to speed up the training. We also tried some different weight initialization strategies to see if it makes any difference in the early stage of model training and improves the accuracy.

1 Introduction

Nowadays, we are usually troubled by the very large amount of online documents. It becomes a big problem to classify these documents into correct categories. Many algorithms are developed by researchers in this area to deal with the problem and great results are achieved.

I use the same data sets as in HW1 Linear Classifiers paper (Yanze, 2022) while this time I also use some word embedding files as the featurization of the input document which are feed into the network.

In this paper, I examine the performance of Nonlinear Classifiers (one-layer feed-forward network) which have two different implementations towards the four different data sets. Overall, there's not much difference between from-scratch implementation and PyTorch implementation towards different data sets. Compared with the performance of the classifiers from the last homework on the same data sets, the performance of Nonlinear Classifiers is quite poor. But it still improves the label classification a lot compared with the baseline (random classification). Unlike Assignment 1, this time I do most of my experiments on local because the model file is quite big and it takes long to upload it on Vocareum.

Different values of hidden layer size, number of words allowed per item, learning rate, number of epochs, and batch sizes are examined. To save time and get a better reasonable representation of result, I don't examine them on all data sets. I use questions data set to examine these parameters, except for the number of words allowed per item I used products data set to examine because questions data set is a short sentence data set. More detailed analysis is included in Experiment part.

2 Data sets

As mentioned above, I use the same data sets as in Assignment 1. The size of 4dim data set is 1560 sentences and it has four labels. The size of odiya data set is 15200 sentences and it has three labels. The size of products data set is 32592 sentences and it has two labels. The size of questions data set is 4089 sentences and it has six labels. The baseline of these classification tasks is 0.25,0.333,0.5,0.167. In addition, These data sets are split into 90% as training data and 10% as testing data in local. Some word embedding files are used as word features. As for the short sentence input file (i.e., odiya data set), a word padding of a vector of zeroes are added. I also use a mean of all the vectors in the embedding file (labeled as 'UNK' in embedding file) to deal with unseen and unknown words.

3 Nonlinear Classifiers

In this task, we used Nonlinear classifiers to do classification. Usually, the linear model is quite flexible but most data are not linearly separable and thus, the performance of linear model on these data is not so good. We can use Neural Network here to deal with these data. Neural Network is a kind of classifier with a non-linear decision boundary. The basic computational unit is a neuron which has a weight vector w and a bias scalar b. It takes in a vector of input x and gives the output based on the weight, bias and the **activation** non-linear functions f. So a single layer neuron can be represented as:

$$y = f(z) = f(w \cdot x + b) \tag{3.1}$$

With these single units, we can build layers and thus build the neural network to solve a classification task. The whole process are mainly divided into two parts: Feedforward computation and Backpropagation which will be introduced below. In addition, we use **cross-entropy loss** as the loss function that models the distance between the classifier output and the correct output:

$$L_{CE}(\hat{y}, y) = -\log \hat{y_c} \tag{3.2}$$

where $\hat{y}$ is the estimate output, y is the true label and c represents the correct class.

3.1 Feedforward networks

The core part of the feedforward neural network is the hidden layer h which is formed of neural unit as defined in Equation 3.1. We only use single hidden layer here so the whole feedforward process in our implementation contains an input layer, a hidden layer and an output layer which can be represented as:

$$k = H \cdot x + B_1 \tag{3.3}$$

$$h = RELU(k) \tag{3.4}$$

$$z = U \cdot h + B_2 \tag{3.5}$$

$$o = Softmax(z) \tag{3.6}$$

where H and U are the weight vector, B_1 and B_2 are the bias scalar. There are also two functions $RELU$ and $Softmax$ which is defined as :

$$RELU(k) = max(k, 0) \tag{3.7}$$

and

$$Softmax(z_i) = \frac{\exp z_i}{\sum_{j=1}^{n} \exp z_j} 1 \leq i \leq n \tag{3.8}$$

for any vector z of dimensionality n.

As shown above, the feedforward neural network actually contains two linear function and one non-linear activation function. At the beginning, the weight vector and bias scalar are randomly initialized which will be updated in the backpropagation stage. To calculate cross-entropy loss, $Softmax$ function is applied to the output of the output layer in Equation 3.6 which converts a vector of numbers into a vector of probabilities.

3.2 Backpropagation

Since the weight w and bias b are initialized randomly, the output at the beginning is meaningless. So we need to update w and b through training to create more accurate output values. The neural network compares the outputs of its nodes with the desired label values. We use gradient descent to update w and b as:

$$\theta^{t+1} = \theta^t - \lambda \frac{\partial l}{\partial \theta}$$

where λ is the learning rate, l is the cross-entropy loss function in Equation 3.2 and θ represents the four weights and bias respectively. Since the learning rate is a given number and l is calculated above, we now need to calculate the derivative of weights and bias using the chain rule to make parameter updates sequentially. The process of derivative is complicated, so we just put the final equation here:

$$\frac{\partial l}{\partial U} = \frac{\partial l}{\partial z} \times \frac{\partial z}{\partial U}$$

$$\frac{\partial l}{\partial B_2} = \frac{\partial l}{\partial z} \times \frac{\partial z}{\partial B_2}$$

$$\frac{\partial l}{\partial H} = \frac{\partial l}{\partial k} \times \frac{\partial k}{\partial H}$$

$$\frac{\partial l}{\partial B_1} = \frac{\partial l}{\partial k} \times \frac{\partial k}{\partial B_1}$$

It's obvious for calculating $\frac{\partial z}{\partial U}$, $\frac{\partial z}{\partial B_2}$, $\frac{\partial k}{\partial H}$ and $\frac{\partial k}{\partial B_1}$ based on Equation 3.3 and 3.5. We now calculate $\frac{\partial l}{\partial z}$ and $\frac{\partial l}{\partial k}$ as:

$$\frac{\partial l}{\partial z} = \begin{cases} o_y - 1, i = y \\ o_i, otherwise \end{cases}$$

where o represent Equation 3.6. And:

$$\frac{\partial l}{\partial k} = \frac{\partial l}{\partial h} \times \frac{\partial h}{\partial k}$$

$$\frac{\partial l}{\partial h} = \frac{\partial l}{\partial z} \times U^T$$

$$\frac{\partial h}{\partial k} = \begin{cases} 1, k \geq 0 \\ 0, otherwise \end{cases}$$

More details are covered in book (Speech and Language Processing 3rd edition - Jurafsky, Martin) and notes(Non-Linear Models - Jonathan May).

Model	4dim	odiya	products	questions
By-hand local	0.43	0.734	0.618	0.764
PyTorch local	**0.53**	**0.822**	**0.69**	**0.658**
By-hand Vocareum	0.4	**0.838**	0.605	0.733
PyTorch Vocareum	**0.5**	0.774	**0.647**	**0.805**
HW1 best	0.95	0.935	0.835	0.75

Table 1: Classification accuracies resulting from from-scratch Implementation and PyTorch. (both on local test and hidden test)

Model	4dim	odiya	products	questions
Random Normal	0.36	0.755	0.611	0.634
Constant Initialization	0.28	0.56	0.54	0.212
Xavier Initialization	**0.5**	**0.815**	**0.613**	**0.75**

Table 2: Performance comparation between different ways of initialization of weight and bias

4 Evaluation

4.1 Experimental Set-up

To test accuracy in local environment, I split the four data sets into 90 percent training data and 10 percent testing data in local. The lines are split into two parts which are the document and the labels. I then use the document and word embedding to create input matrix which have a (number of sentence, word embedding size $*$ number of words per item) dimension to divide into small batches. I add vector of zeros to ensure robustness and support short input size to make Neural Network stable.

To test performance of the initialization of weight and bias, I used Constant Initialization, Random Normal and Xavier Initialization (Neural Network: Breaking The Symmetry - Luthfi Ramadhan) to make a comparation for from-scratch implementation and Linear layer Initialization for PyTorch. Batch-size input data then pass through the forward pass and I use the output of forward pass to calculate the derivative of weight and bias to update the weight and bias in backward function. Different values of hidden layer size (u), number of words allowed per item (f), learning rate (l), number of epochs (e), and batch sizes (b) are examined based on questions data set for both implementations. The detailed comparation is covered in Result part.

The parameters used frequently in the Experiments is $u = 30$, $l = 0.01$, $f = 10$, $e = 200$ and $b = 10$. For specific parameter comparation, only the parameter examined changes and the other parameters still keep the value given above.

4.2 Results

Performance comparation between from-scratch Implementation and PyTorch Implementation This part is related to the accuracy performance of from-scratch Model and PyTorch Model. As shown in Table 1, PyTorch model is much better than from-scratch Model. There may exist several reasons. Firstly, I used Random Normal Initialization in this comparation which is not so good for weight and bias Initialization. Secondly, the way of updating weight and bias may be different between two implementations. The overall performance of nonlinear classifier is much worse than linear classifier in HW1. For different data sets, nonlinear classifier performs well in odiya and questions data sets and performs bad in 4dim and products data sets (not a very high improvement compared with baselines). I think the main reason is that the odiya and questions data sets are short sentence data sets and number of words allowed per item is suitable for these two data sets while 4dim and products data sets may need another ways to use the feed forward model or a different neural architecture. I will leave this for future work.

Performance comparation between different values of parameters Let's consider to change the values of different parameters to see whether we can get better accuracies. The classification accuracy of different values of different parameters are shown in Table 3. For hidden layer size (u), accuracy is positively correlated with the size of hidden layer both for from-scratch Model and PyTorch Model. For learning rate (l), both models

Datasets	By-hand Accuracy	PyTorch Accuracy	By-hand Loss	PyTorch Loss
$u = 5$	0.406	0.608	1.48	1.43
$u = 25$	0.522	0.604	1.675	1.44
$u = 50$	0.548	0.746	2.21	1.33
$u = 100$	0.538	0.744	4.27	1.31
$u = 200$	**0.586**	**0.752**	14.8	1.31
$l = 0.001$	0.422	0.424	1.59	1.65
$l = 0.01$	**0.542**	**0.764**	2.11	1.32
$l = 0.1$	0.478	0.762	5.1	1.27
$f = 5$	**0.652**	0.698	1.18	1.40
$f = 10$	0.562	**0.754**	1.42	1.31
$f = 20$	0.544	0.646	1.744	1.40
$e = 20$	0.492	0.380	1.45	1.70
$e = 50$	0.494	0.552	1.46	1.53
$e = 200$	0.664	0.742	1.22	1.33
$e = 1000$	**0.692**	**0.774**	3.12	1.28

Table 3: Classification accuracies and loss for different u, l, f, e.(on local test)

performs better when $l = 0.01$ which shows that we need to find a suitable learning rate for better accuracy. For number of words allowed per item (f), from-scratch model performs better when $f = 5$ and PyTorch model performs better when $f = 10$. Their accuracy goes down as the parameters get bigger because questions data set is a short sentence dataset and more zero vectors are added when f increase. For number of epochs (e), It's obvious that accuracy is positively correlated with the number of epochs, but their time consumption also goes up. We need to find a trade-off between the accuracy and time consumption.

Performance comparation between different ways of initialization of weight and bias I tested the performance when the weight and bias are initialized differently. Using $u = 30$, $l = 0.01$, $f = 5$, $e = 200$ and $b = 10$ as parameters, I tested three different ways. The performance of them are shown in Table 2. We can see that the performance of Xavier Initialization is much better than the other two Initialization methods. According to my training accuracy on local, the training accuracy improves much quickly during the early phase of training with Xavier Initialization.

Training time consumption with different batch sizes For the relationship between training time consumption and batch sizes, I use $u = 20$, $l = 0.01$, $f = 10$, $e = 100$ to test odiya data set and the result time consumption for $b = 5$,$b = 20$,$b = 50$,$b = 100$ is 288 seconds,262 seconds,260seconds, 254 seconds. There is no obvious improvement here.

5 Discussion

Some of the discussion and analysis is covered in the previous section. I will discuss some more here.

For learning rates, we need to find a suitable rate which is Neither too high nor too low. A too high learning rate may make the learning process jump over the minimal value of the loss function and a too low learning rate may take too long to converge or ends into a undesirable local minimal value.

For overfitting, it seems to exist during my training data for model. I recorded the training accuracy of 4 data sets. For 4dim and products data sets, the training accuracy approaches one after several epochs, but the testing accuracy is not so good. For odiya and questions, the training accuracy seems to be close to their testing accuracy. This may be due to the sentence length of the data set which can be tested more in future work.

For loss convergence, from-scratch model performs not so well as compared with PyTorch model but it is still acceptable when using suitable parameters.

6 Conclusion

We usually need to find a trade-off between the time consumption and the accuracy when choosing parameters. Both from-scratch model and PyTorch model do not perform as well as the linear model. Future work will focus on more hidden layers and different models.

References

Amani Peddada Richard Socher Qiaojing Yan Christopher Manning, Rohit Mundra. 2019. Natural language processing with deep learning.

James H. Martin Daniel Jurafsky. Speech and language processing.

Jacob Eisenstein. 2018. Natural language processing. *Jacob Eisenstein*.

Jonathan May. 2022. Non-linear models.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86. Association for Computational Linguistics.

Luthfi Ramadhan. 2020. Neural network: Breaking the symmetry.