Classifying capital letter from letter-recognition data in deep nerual netwroks based on Back Propagation

Yinheng Chen

Reserarch School of Computer Science, The Austrilian National University, Canberra
{Yinheng Chen}@u6341895@anu.edu.au

Abstract. The object of my dataset is to identify 26 capital letters which is displayed by a large number of black-and-white rectangular pixel in English alphabet. These capitals are also from 20 different fonts and each letter is randomly distorted to produce a file of 20000 unique stimuli. Each stimuli contains 16 numerical attributes whose value ranges from 0 to 15 and one attributes of capital letter. I finally used the 4-layers neural network to get the predication of selected dataset. After trying different size of dataset, changing the parameter of deep neural network, pre-processing method and implementing a function to detect whether there are some biases in dataset and remove the biases, the result was significantly improved. Dmitry, Alexandrin, David and Lyle (2003) got the highest result of accuracy of 76.62%, in my last experiment, my result was worse than theirs. However, after using a deep neural network, this time I found my result was better than theirs.

Keywords. Dnn, capital letter, target as matrix, normalization

1 Introduction

This dataset was used as a classification task, and this dataset contains enough instances (around 20000) and effective attributes. In addition, this dataset can be used to detect the letter categories linked with 16 vectors which is processed from pixel (Peter and David, 1991), it is helpful for us to guess a value which is determined by a series of value in others dataset. During my experiment, I previously used a neural network with batch size. However, there are a number of problems of this neural network. In my previous assignment, I used a neural network to predict the results: however, I found the accuracy was not ideal enough, and the loss was high. So, in this experiment, I used a deep neural network to train and test, and I used more pre-processed methods on dataset, and the result was greatly improved compared to the result in last experiment.

2 Method.

2.1 Simple dataset analysis.

In this experiment, I used the dataset which I had used in assignment 1.

Firstly, I should find whether there are some missing values in this dataset or not. The dataset description showed there are no missing value and I check the dataset in DataFrame and confirmed it.

Now, in the next step, I intended to sample the dataset to reduce the unbalance and bias. However, I found the following information from Data Description

Class Distribution:

789 A	766 B	736 C	805 D	768 E	775 F	773 G
734 H	755 I	747 J	739 K	761 L	792 M	783 N
753 O	803 P	783 Q	758 R	748 S	796 T	813 U
764 V	752 W	787 X	786 Y	734 Z		

I found the class distribution is pretty balanced in this dataset, as we can see the max number of one class is 813 of U, and the min number of one class is 734 of H. These two number are closed. So, any method of sampling would not help to increase the accuracy of result.

In addition, I found each attribute cannot be dropped for the target value is determined by them, and all attribute is necessary so I keep all attribute in my dataset.

2.3 Method to process dataset

I found the information of dataset with its attributes and target value.

Here below is the information of dataset.

Attribute Information:

Attribute	Explanation	Type
lettr	capital letter	object
x-box	horizontal position of box	(integer)
y-box	vertical position of box	(integer)
width	width of box	(integer)
high	height of box	(integer)
onpix	total # on pixels	(integer)
x-bar	mean x of on pixels in box	(integer)
y-bar	mean y of on pixels in box	(integer)
x2bar	mean x variance	(integer)
y2bar	mean y variance	(integer)
xybar	mean x y correlation	(integer)
x2ybr	mean of x * x * y	(integer)
xy2br	mean of x * y * y	(integer)
x-ege	mean edge count left to right	(integer)
xegvy	correlation of x-ege with y	(integer)
y-ege	mean edge count bottom to top	(integer)
yegvx	correlation of y-ege with x	(integer)

According the attribute information, we found the first feature should be our target value, however, the type of this attribute is object, so the first step is to numeric the features, and provide a suitable distribution of the 26 classes (Peter and David, 1991).

In my last experiment, I set A as 1, B as 2 ……. Z as 26. However, I found this method had some disadvantages. Manual setting the numeric values for 1 to 26 in order is not objective. Besides, are there really some size and sequence relationship among these 26 letters? Does Z (26) is greater than A (1)? Does A is front of Z?

The answers are no. So, one solution is to transform target values to matrix.

In that case, I firstly used a fit_transform function in LabelEncoder to set 26 target values to the numeric values from 0 to 25 randomly. Then, I used to OneHotEncoder to transform 26 numeric values to 26 matrixes as below pictures shows.

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

This means, every target value was represented by 26 columns.

2.4 Methods from selected paper

I used two methods from a paper in NN4.

There are following two methods:

1. Need to normalize data.

   Because all feature values are integers. I normalized all 16 feature values in this experiment.

2. Consider using statistical Z function to remove bias of lowest and highest values.

As we can see, the value of input attributes ranges from 0 to 15, so we should check whether values are correct in this dataset, I used max () and min () functions for array in python, I found all inputs are correct and the max value is 15 and the min value is 0.

2.5 Setting training dataset and testing dataset

Because this dataset contains 20000 instances, so I divided the original dataset randomly in python by using train_test_split function to avoid there were overlapped data in training dataset and testing dataset. The training dataset contains 16000 instances while testing dataset contains 4000 instances.

2.6 Setting Deep neural network

In this experiment, I used a deep neural network with 1 inputs layer, 2 hidden layers and 1 outputs layer. The activation function in hidden layers was sigmoid, the activation function in outputs layer was softmax, and the loss function was cross-entropy.

3 Result and Discussion

In my result, I will show the results of different tests with different neural network parameters, activation functions and whether I normalise the dataset.

Parameters:

Size: size of dataset

Normalization: whether inputs were normalized or not.

Learning rate

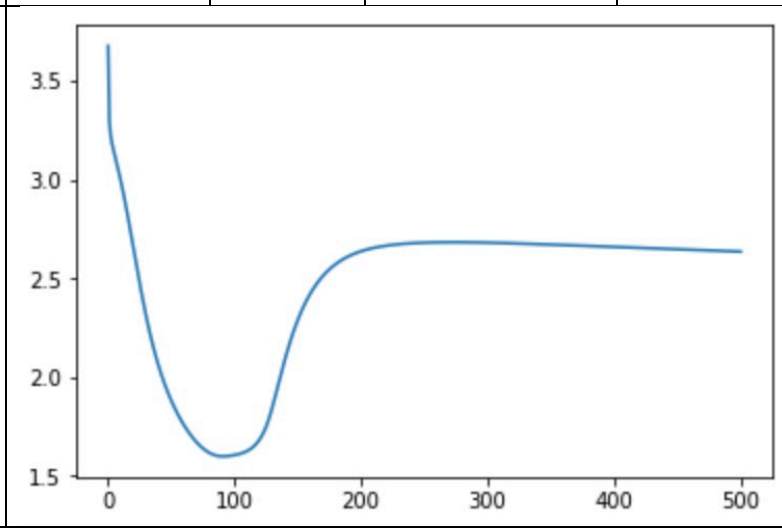
Num_iter: number of iteration

Hidden_neurons: number of hidden_neurons in two hidden layers.

Plot:

The ordinate represents the loss, and the abscissa represents the number of iterations.

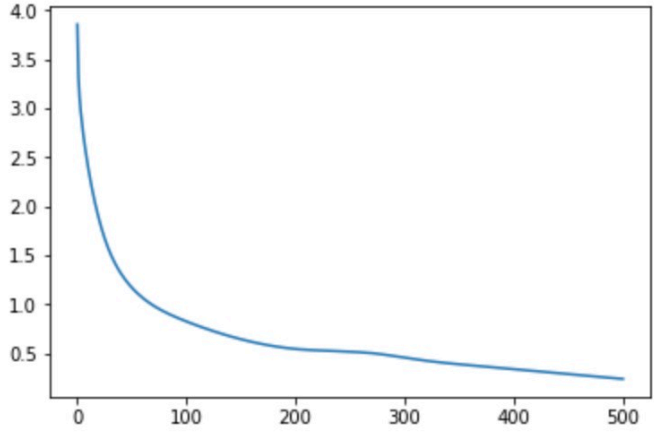
Test 1:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
1000	**0.005**	**500**	**30,30**	**no**	**19%**
Plot					

.

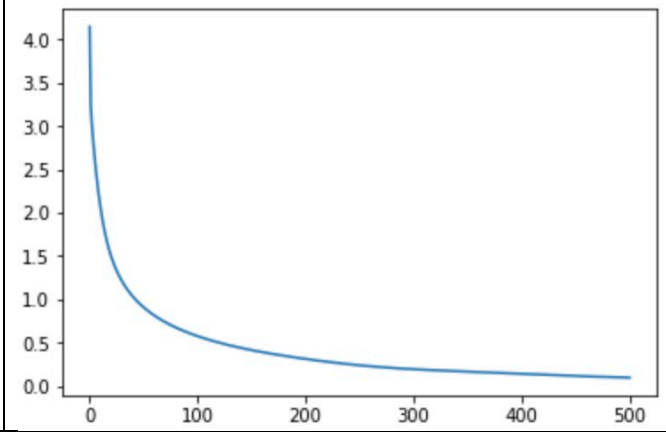
The plot showed that with the number of iterations increased, the loss firstly decreased, and then increased, and then become stable. The accuracy was very low as well. I guess the main problem was that the inputs data wasn't normalized. So, in test 2, I normalized inputs data.

Test 2:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
1000	**0.005**	**500**	**30,30**	**yes**	**67%**
Plot					

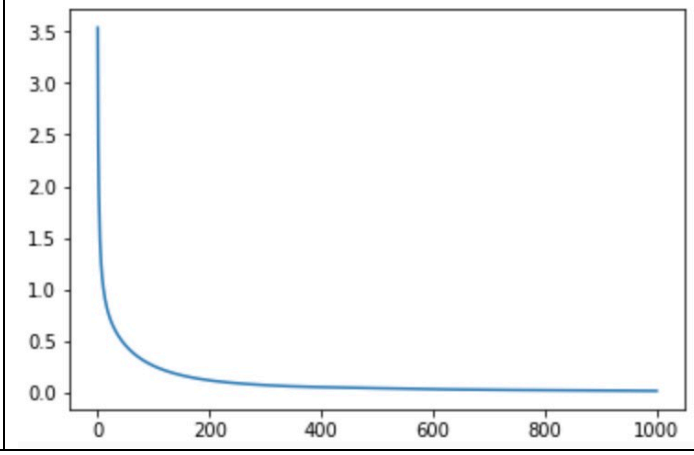
After the inputs data was normalized, the performance was significantly improved. The accuracy also reached 67%. The line in plot had a tend to increase and become stable in low loss. Because the target value had 26 classes, so I guess the performance would be better if I increased the number of hidden neurons.

Test 3:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
1000	**0.005**	**500**	**50,50**	**yes**	**76%**
Plot					

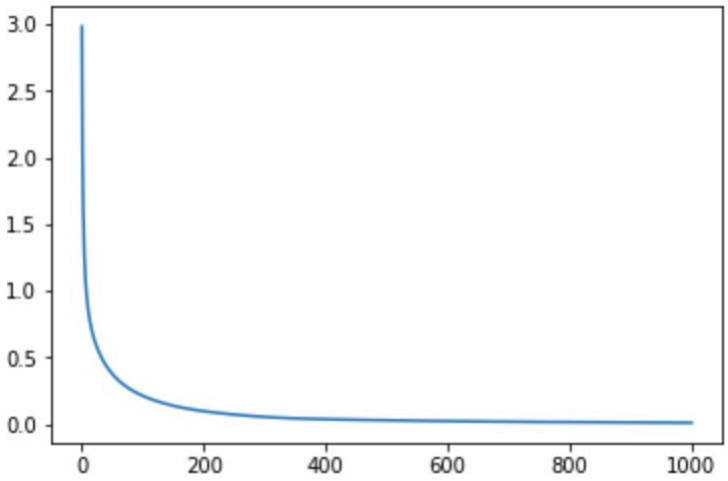
The accuracy reached 76% after increased the number of hidden neurons. Because the performance was improved significantly in this test. I thought the performance still could be improved more if I changed some parameters.

Test 4:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
2000	**0.005**	**1000**	**75,100**	**yes**	**82%**
Plot					

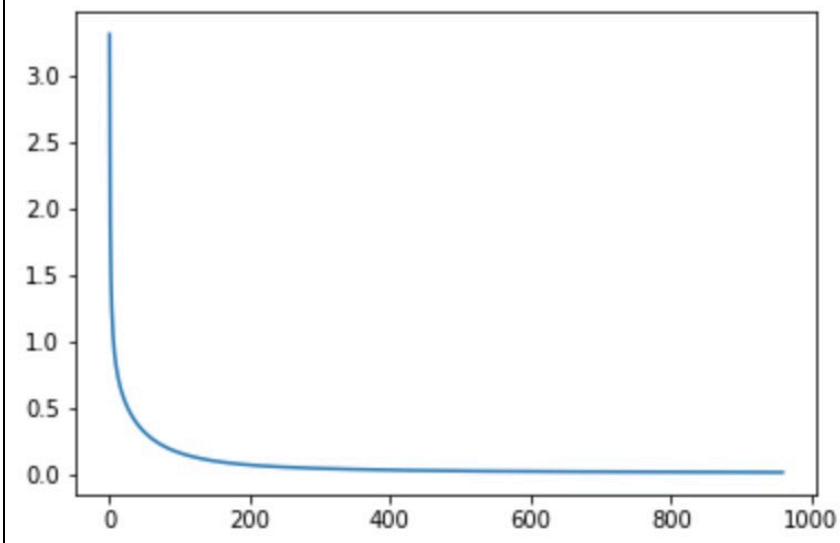
We still can see a clear increase in accuracy and the accuracy was good, but I still want to try increased the performance by changing the parameters.

Test5:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
3000	**0.005**	**1000**	**75,100**	**yes**	**85.2%**
Plot					

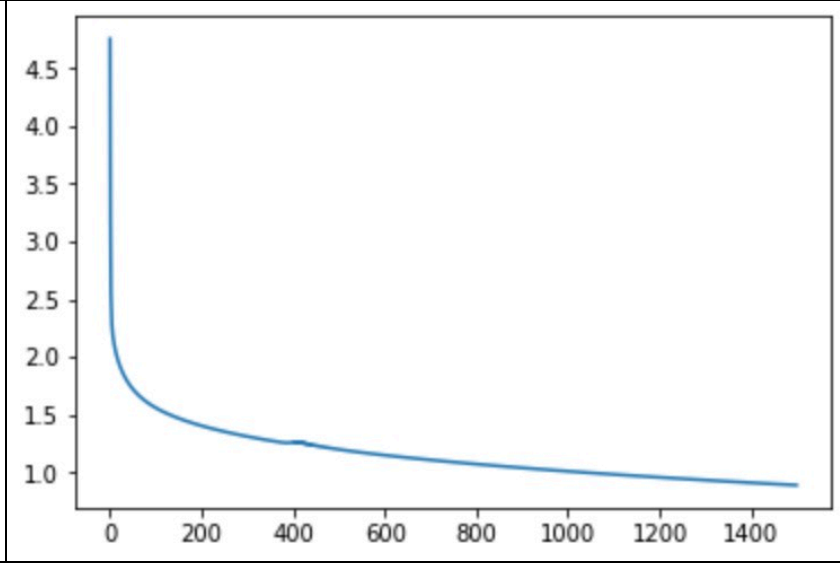
I found the accuracy still increase but not too much. So, I considered to improve hidden neurons in next test.

Test 6:

Size	learning rate	num_iter	Hidden_neurons	Normalization	Accuracy
3000	**0.005**	**1000**	**100,125**	**yes**	**87.7%**
Plot					

After increasing the number of hidden neurons, the performance improved again, and the accuracy reached 86.7%, which was ideal.

Previous test:

hidden_neurons	learning rate	num_epoch	function	Normalization	Result
100	0.01	1500	relu	no	68.47%
Plot					

The best result of my previous experiment in simple nn is showed above. As we can see, the best accuracy was around 68.47%. However, the best accuracy in this experiment in dnn reached 87.7%. So, there was a clear improvement in my experiment after changing simple neural network to deep neural work.

Compare results with selected paper.

The results below showed were did by Dmitry, Alexandrin, David and Lyle (2003) in the paper of Mixtures of Conditional Maximum Entropy Models.

Name		1	3	5	7	9	11	13	15
Letter recognition	A	72.42	74.65	76.62	76.47	76.45	76.35	76.07	—
	L	-1.069	-0.974	-0.862	-0.868	-0.840	-0.870	-0.914	—
	T	3004	11992	20800	29540	40801	45913	—	—

The highest accuracy they did in this paper was 76.62%. However, in my later 3 tests, all results of accuracy were over 80% and the highest was 86.7%. After using a deep neural network, the performance in my experiment was better than that in paper.

4 Conclusion and Future Work

I have used a pixel data of letter recognition and other useful information, and I have shown the different results with different situations. These data have been used to derived classification for letter predication by using different methods of processing data and different neural networks.

In my last experiment in simple neural network, some basic methods to process data do not be used for the original data is quite good and there are no missing values and bias in this dataset. I found sometimes changing an activation function is much useful than trying other methods to process data. I did a lot of analysis task in data but a little process in data. It is true that analysis can be very helpful for it can help us to reduce some unnecessary methods and steps to process dataset.

After using a deep neural network in this experiment, I found more pre-process methods are necessary before I did some tests, so the performance was significantly improved in this experiment. This time I didn't change activation function. Besides, I also spend many times on analysing dataset for it is so important, and I found change a target value to a matrix is effectively to increase the performance. Also, normalization in one neural network has less impact do not means that it does not work on other neural network. The most noticeable improvement in this experiment can attribute to normalization.

In deep neural network, changing the number of hidden neurons can be very useful.

As for future work. I think I can try more pre-process methods on my dataset. I think there are still some methods can improve my dataset. Besides, I also can try to increase the number hidden layers in deep learning network. For example, I can use 5, 6 or more layers neural network to do some prediction.

In my two assignments, I used a csv dataset to do classification and prediction. In the future, I think I can use a images document by using other deep learning methods, such as cnn, rnn ,lstm and more to do some classification and prediction works on dataset and to get good performance in my future experiment.

References

Bustos, R. A., & Gedeon, T. D. (1995). Decrypting Neural Network Data: A GIS Case Study. In Artificial Neural Nets and Genetic Algorithms (pp. 231-234). Springer, Vienna.

Frey, P. W., & Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. Machine Learning, 6(2), 161-182. doi:10.1007/BF00114162

Pavlov, D., Popescul, A., Pennock, D. M., & Ungar, L. H. (2003). Mixtures of conditional maximum entropy models. In Proceedings of the 20th International Conference on Machine Learning (ICML-03) (pp. 584-591).