DNN MODEL: Classifying capital letter from 16 numerical data

Chengwei Zhu

Research School of Computer Science, The Australian National University, Canberra

{Chengwei Zhu }@u6342721@anu.edu.au

Abstract. This data set includes 20000 data and the feature of these data were summarized 16 different numerical attributes, and different combination of these 16 attributes can represent different English capital letters, and my training model can predicate the letters by the 16 different numerical attributes. This time, I used the asme data set and tried using a deep learning approach, constructing a DNN model to predict the target attribute. Also, I used some methods, including data normalization, random sampling or some other data preprocessing approaches, setting different parameters of DNN model like the number of hidden layer neuron, to make my model as much accuracy as possible. And this time the accuracy of my model arrived at 84.5%, better than the results got by Pavlov, Popescul, Pennock and Ungar (2003).

Keywords: randomly sampling, data normalization, neural net, DNN model

1 Introduction

This time, I chose the same data set as last time, from the UCI Machine Learning Repository located at http://archive.ics.uci.edu/ml/datasets.html, and this data set is called Letter recognition. As we all known, with the development of modern science and technology, data statistics and analysis techniques have been widely applied to various fields. According to Paliouras and S.Bree(2005), quantities of empircM concept learning algo-rithms have been improved since two decades ago. Also, when human experts faced with difficult situation, they always can treat these problems as special cases of familiar examples by classifying and analyzing them and then apply known solutions to work it out (de Groot, Chase & Simon, cited in W.Fery, J.Slate, 1991). As a result, I planned to find a suitable data set to check the effect of using digital features in the training set, which could help me realize the power of analytics in some degree. Finally, the data set I chose is called 'Letter Recognition Data Set', has 20000 characters which were produced by 20 randomly distorted and different fonts. And these 20 fonts could make up different character images and each of them would be identified as one of 26 English capital letters. During the whole process, I constructed a DNN model to predict the target value through other 16 attributes. And I used randomly sampling and some data preprocessing methods to get a better and reliable model.

2 Method

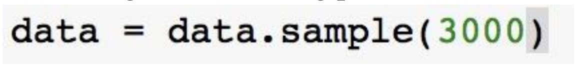
1) Change the column name and convert string target values to numeric values firstly. I will show the attribute information of this datset.

Letter	capital letter	(string)
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)

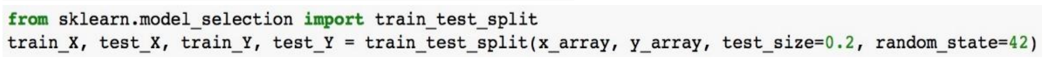
According this form, the type of target values is string, I should convert them to numeric values at first.

2) Randomly extract data and divide it into training and test sets

Firstly, I did not use this method, and I just selected the first thousand rows of data as training set and the data from 1001 row to 1400 row as testing set. However I realized that this operation had some disadvantages because the training set and testing set were stable, as a result, the accuracy of the model I trained was not representative. Also, in that case, it would be complicated if I want to use different training and testing set. Therefore I chose to use random sampling method. And this time I applied random sampling method two times. First time, I applied it to randomly extracting a certain percentage of data from original data set. Second time I used it to help me separate data into training and testing set randomly to prevent overfitting. As following pictures shown.



```
data = data.sample(3000)
```

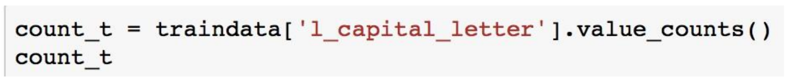


```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split(x_array, y_array, test_size=0.2, random_state=42)
```

3) data preprocessing

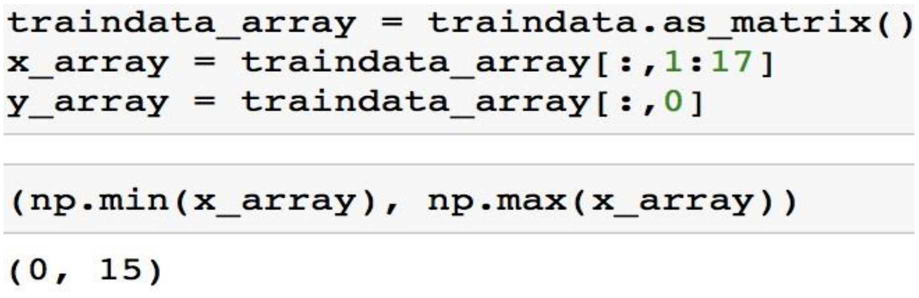
Here I will show my trial process.

i Firstly I want to apply stratified sampling method to my first step, extracting data, to train a more accurate model. However, after I counted the number of each values of target columns, I found the number of different class almost the same both in training and testing sets, hence I realized that stratified sampling methods could make little attributes to improving testing accuracy.



```
count_t = traindata['l_capital_letter'].value_counts()
count_t
```

ii Then I read one pdf file called DecryptGISData camera form Papers for NN4, it told us that normalizing data over the range 0 - 1 for the network from logistic aspect can help to deal with the unreliable data and get more accurate prediction. Also, this pdf file also told me to remove bias of lowest and highest values by using statistical Z function to reduce noise. I thought this method is feasible and then I checked the minimum value and the maximum value of the values of input data in training set and testing set as following picture shown.



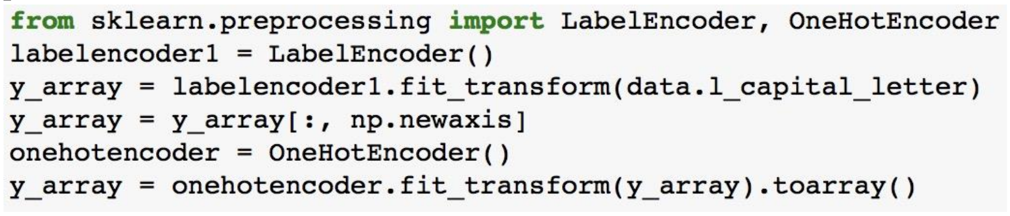
```
traindata_array = traindata.as_matrix()
x_array = traindata_array[:,1:17]
y_array = traindata_array[:,0]
```

```
(np.min(x_array), np.max(x_array))
```

```
(0, 15)
```

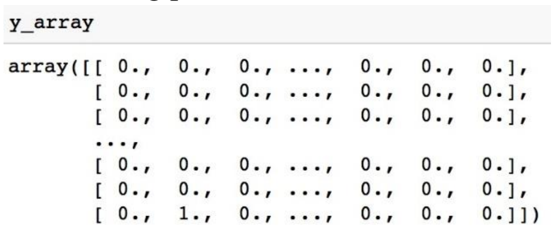
I found that the difference between the maximum and minimum values arrived at 15, therefore I tried to normalize training input data by columns. And it did work, I will show the result in the result and discussion section.

iii Last time, we cannot normalize the target data because if we normalize the target data, we will get a Tensor hold outputs whose values are changing to 0 or 1 as a result of .long() function, and in that case our model will be become extremely inaccurate. As a result, last time I only changed the target to number, from 0 to 25. However, this time I can use some other method to optimize the target data. As we all known, you can only provide values to the machine, not strings when doing machine learning. And the Scikit-learn toolkit package can provide a convenient tool, LabelEncoder, which could help us easily change category information to numerical values. And after that operation, the values of the target column were becoming numerical. And I found the value of different output were different in the size, but the predicting target letters have no size relationship with each other. So we should transfer the value of target column from one number to an array in the help of OneHotEncoder, a useful tool provided by the Scikit-learn toolkit package. Code will be shown in following pictures.



```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder1 = LabelEncoder()
y_array = labelencoder1.fit_transform(data.l_capital_letter)
y_array = y_array[:, np.newaxis]
onehotencoder = OneHotEncoder()
y_array = onehotencoder.fit_transform(y_array).toarray()
```

And after running these code shown above, the target values will become arrays as following picture shown.



```
y_array
```

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

3 Result and Discussion

1. Test with a few different simple parameters

■ Results

Test number	Data preprocessing	Number of neurons for hidden layer	Learning rate	Number of epoch on training	Testing accuracy
1	No	10	0.01	500	5.95%
2	No	10	0.01	2000	19.23%
3	No	14	0.01	2000	25.3%2
4	No	14	0.01	5000	42.83%
5	No	100	0.01	5000	63.52%
6`	Yes(normalize data)	100	0.01	5000	16.98%

■ Discussion

From the results of above six tests, we can conclude that for my training model, with the increasing of the number of neurons for hidden layer, the testing accuracy of my model show an increasing trend as well. Also, the number of epoch on training is bigger,

the testing accuracy will be higher. As a result, the number of neurons for hidden layer and epoch on training can affect the model accuracy. However, we can find an interesting fact that after I normalized data before training them, my model accuracy showed a decreasing trend, which made me confused. While the other day I read some information related to this data set, I found that every stimulus was converted into 16 primitive numerical attributes and then scaled to fit into a range of integer values from 0 through 15, so I thought that maybe lead to the situation in test 6.
When comparing to the results from paper called Mixtures of Conditional Maximum Entropy Models, we can find with the rising number of attributes, the testing accuracy would represent a increasing trend, and I have to admit their model is better than me as most of the testing accuracy of their model are higher than mine, which could reach 82.2% (Palov, popescul, Pennock, Ungar 2003). To improve my model, I still have lots of work to finish.

2. Test with different activation function and different optimizer (all without data preprocessing)

■ Results

Test number	Number of neurons for hidden layer	Learning rate	Number of epoch on training	Type of activation function	Type of optimizer	Testing accuracy
7	100	0.01	2000	sigmoid	SGD	53.83%
8	100	0.01	2000	tanh	SGD	70.67%
9	100	0.01	2000	relu	SGD	76.9%
10	100	0.01	2000	sigmoid	ASGD	51.52%
11	100	0.01	2000	sigmoid	Adam	95.2%
12	100	0.01	2000	sigmoid	Adamax	94.88%

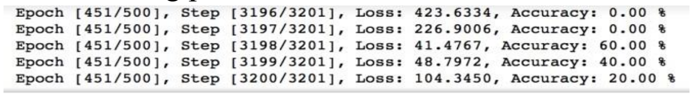
■ Discussion

From above 6 tests we can find that the activation function for hidden layer which called tanh or relu can make more attributes to training a more accurate model. Also, the optimizer called Adam or Adamax can make our model becaome extremely accurate, however, I thought these situation happened because these two optimizer could make my training data become overfitting. Different optimizer has different function, and the data set I chosen maybe appropriate for SGD optimizer while Adam and Adamax optimizer may make may model become overfitting.

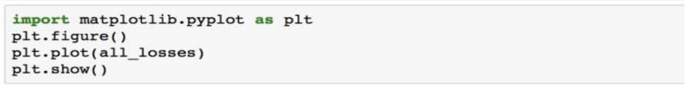
3. Using mini-batch gradient descent

1) Define the size of input as 16, the size of hidden layer neuro as 100, the number of output classes as 26, the number of epochs as 500, batch size as 5, learning rate as 0.01

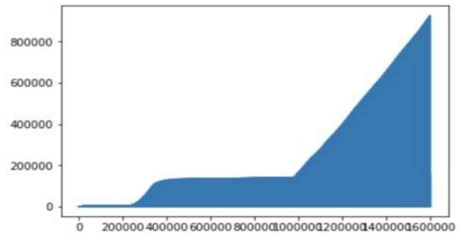
   Following pictures showed the results

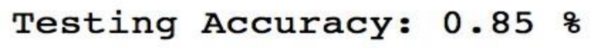


```
Epoch [451/500], Step [3196/3201], Loss: 423.6334, Accuracy: 0.00 %
Epoch [451/500], Step [3197/3201], Loss: 226.9006, Accuracy: 0.00 %
Epoch [451/500], Step [3198/3201], Loss: 41.4767, Accuracy: 60.00 %
Epoch [451/500], Step [3199/3201], Loss: 48.7972, Accuracy: 40.00 %
Epoch [451/500], Step [3200/3201], Loss: 104.3450, Accuracy: 20.00 %
```



```python
import matplotlib.pyplot as plt
plt.figure()
plt.plot(all_losses)
plt.show()
```

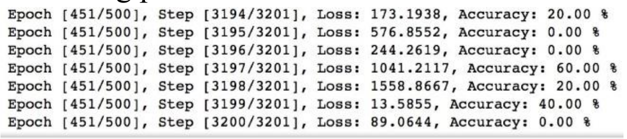




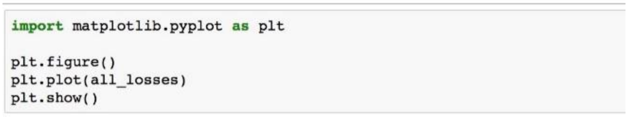
```
Testing Accuracy: 0.85 %
```

2) Define the size of input as 16, the size of hidden layer neuro as 50, the number of output classes as 26, the number of epochs as 500, batch size as 5, learning rate as 0.01

   Following pictures show the results

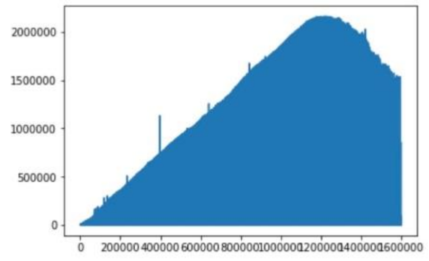


```
Epoch [451/500], Step [3194/3201], Loss: 173.1938, Accuracy: 20.00 %
Epoch [451/500], Step [3195/3201], Loss: 576.8552, Accuracy: 0.00 %
Epoch [451/500], Step [3196/3201], Loss: 244.2619, Accuracy: 0.00 %
Epoch [451/500], Step [3197/3201], Loss: 1041.2117, Accuracy: 60.00 %
Epoch [451/500], Step [3198/3201], Loss: 1558.8667, Accuracy: 20.00 %
Epoch [451/500], Step [3199/3201], Loss: 13.5855, Accuracy: 40.00 %
Epoch [451/500], Step [3200/3201], Loss: 89.0644, Accuracy: 0.00 %
```



```python
import matplotlib.pyplot as plt

plt.figure()
plt.plot(all_losses)
plt.show()
```



■ Discussion

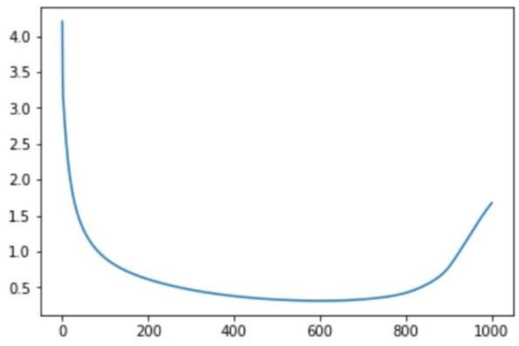
According this two result of using mini-batch method, we can find it didn't work well and even it might make the model become less accurate. In my opinion, maybe this data set is not appropriate for this method as the data set has fewer outliers or some other reasons.

4. Constructing a DNN model (The ordinate in below pictures is the loss rate, and the abscissa in below pictures is the number of iteration)

○ 1. No randomly sampling, no data preprocessing, separating data into training and testing set through manual operation. (selected the first thousand rows of data as training set and the data from 1001 row to 1400 row as testing set)
Setting 4-layer neuron, the number of 1st hidden layer neuron as 128, the number of 2nd hidden layer neuron as 64, the number of maximum iteration as 1000.

Results:

```
Accuracy: 0.35
```

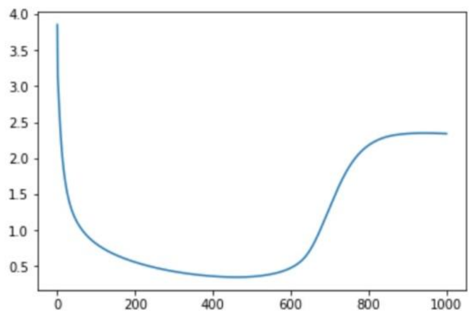


Discussion:
We found the accuracy of the model was very low, only arriving at about 30%, and we can interestingly find that after the number of iteration exceeded 500, the loss rate of my model represented an increasing trend. And this result was worse than that of the model built by Palov, popescul, Pennock, Ungar (2003), and the highest accuracy of their model arrived at 82.2%.

○ 2. Randomly sampling, transferring the target values to arrays. (randomly extracted just 1000 row data to train this model)
Setting 4-layer neuron, the number of 1st hidden layer neuron as 128, the number of 2nd hidden layer neuron as 64, the number of maximum iteration as 1000.

Results:

```
Accuracy: 0.21
```

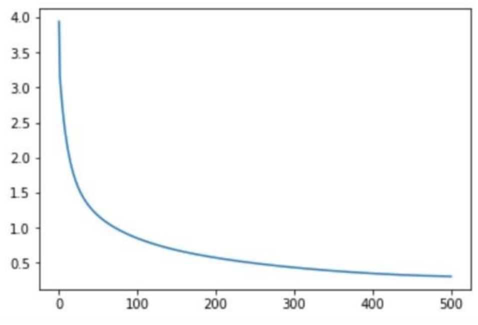


Discussion:
This time, the accuracy of my model was only 21%, even worse than that of my model in last trial. And I found that the loss rate was extremely low when the number of maximum iteration arrived at about 500, but later the loss rate began to increase with the increasing number of maximum iteration. Therefore, I set the number of maximum iteration as 500 to train a new model.

- 3. Randomly sampling, transferring the target values to arrays. (randomly extracted just 1000 row data to train this model)
  Setting 4-layer neuron, the number of 1st hidden layer neuron as 128, the number of 2nd hidden layer neuron as 64, the number of maximum iteration as 500.

  Results:

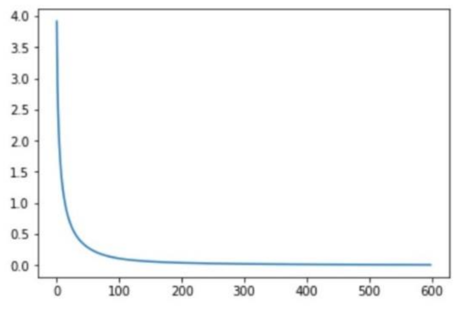
```
Accuracy: 0.725
```



  Discussion:
  We found the accuracy of the model had a dramatic increase, achieving 72.5%, which was very closely to the result of the model built by Palov, popescul, Pennock, Ungar (2003). And this time, the loss rate always decreased as the number of iteration grew, which might be a good phenomenon.
- 4. Randomly sampling, transferring the target values to arrays and normalizing the input data. (randomly extracted just 1000 row data to train this model)
  Setting 4-layer neuron, the number of 1st hidden layer neuron as 128, the number of 2nd hidden layer neuron as 64, the number of maximum iteration as 1000

  Results:

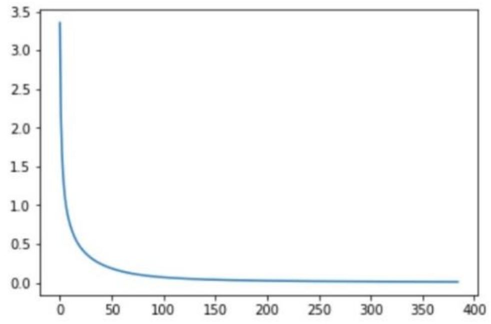
```
Accuracy: 0.72
```



  Discussion:
  After I normalized the values of input atributes, the accuracy was still around 72%, therefore I tried to use a bigger data set to train the model to check whether this method is helpful. In that case, I did the next trial.
- 5. Randomly sampling, transferring the target values to arrays and normalizing the input data. (randomly extracted just 3000 row data to train this model)
  Setting 4-layer neuron, the number of 1st hidden layer neuron as 128, the number of 2nd hidden layer neuron as 64, the number of maximum iteration as 1000

Results:

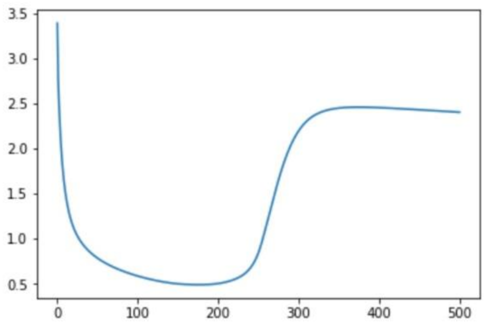
```
Accuracy: 0.845
```



Discussion:

After I extracted 3000 row data to train the model, the accuracy of the model arrived at 84.5%, which was even higher than the best accuracy of the model built by Palov, popescul, Pennock, Ungar (2003). Therefore, I removed this method, normalizing the input data, and then repeat the 3rd trial above, however, this time I extracted 3000 row data as well. Finally, the result come out, and the accuracy was only 23.1% as following picture shown. As a result, it told me that the method of normalizing the input data was useful.

```
Accuracy: 0.23166666666666666
```



4 Conclusion and Future Work

I have constructed a DNN model to achieve the goal that predicting the letter from other 16 attributes. And I finally successfully used these useful information to predict the target letter and got a reliable model after I applied some helpful methods, like randomly sampling and some data preprocessing approaches.

And during my trial this time, I found some methods which may be not useful to get a high accuracy when I was constructing neural network last time played a great role in improving the accuracy of my model this time. And this time, I applied some new methods like transferring the target values to arrays and normalizing the input data, and the results come out showed us that these methods worked well and were very helpful to reduce machine learning errors. Also, according the results of my test, I found that the number of maximum iteration and the size of training data can affect the accuracy of my DNN model. And I guessed that there exists a critical point and when we set the critical value of the number of maximum iteration or a critical value of the size of training data, the accuracy of model will arrive at the highest value and the model will perform pretty good. However, I haven't figured out the reason, which I will focus on during my future work.

As for my future work, I still have a lot work to do, I should and I will keep on working how to construct a CNN or a RNN model, and I should have a better and deeper understanding of deep machine learning. Moreover, I should read more paper about how to reduce data set outliers to get a more accurate model.

References list.

1. Gedeon, T. D. (1995, November). Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour. In *Artificial Neural Networks and Expert Systems, 1995. Proceedings, Second New Zealand International Two-Stream Conference on* (pp. 26-29). IEEE.
2. Gedeon, T. D., & Bowden, T. G. (1992). Heuristic pattern reduction. In *International Joint Conference on Neural Networks* (Vol. 2, pp. 449-453).
3. Frey, P. W., & Slate, D. J. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine learning*, *6*(2), 161-182.
4. 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).