Effect of Data Pre-processing on the Accuracy of Classifying Mobile Device Interaction Tasks Based on Eye-gaze Pattern

Zheng, Huang[0000−0001−6212−3119]

Australian National University, Canberra ACT 2600, Australia
amoyhuangzheng@yahoo.com

Abstract. A huge amount of data is known to be the basis for making good predictions in a machine learning problem. However, in reality we do not always have the data in our desired form and quantity, such as the one we will be using here. Changing the encodings of data before they are plugged into the neural network could be the key to yield good prediction accuracy in this case. This study uses some eye-tracking data coupled with a few user feedbacks. The goal of the network is to predict the task a user is performing based on the corresponding recorded data. The choice of goal is trivial here as the target of the study is to investigate the effect of altering input encoding on a neural network's ability to classify and generalise with respect to number of epochs. A huge difference was observed with source identified in the end.

Keywords: Neural Networks · Data Pre-processing · Encoding · Auto-encoder.

1 Introduction

Machine learning in general, relies heavily on data, whether it is generated or given. When the data is given, it often has some of the following problems [1]:

1. The data is encoded in some ways, either with some domain knowledge or for the purpose of the study that prompt them to obtain the raw data.
2. The data has an uneven distribution of instances belonging to each category, during training, the category with higher occurrences may be favoured, leading to overfitting and bad generalisation.
3. Within the data, the scale and units may be inconsistent, which could have a huge impact on training efficiency.

While the latter 2 can be solved almost automatically [2], the first one always requires some manual decoding or transformation for the data to be suitable for learning. This involves primarily the conversion of categorical data to numberics and the filtering or re-encoding of the inputs to the neural network.

In this study, I will compare a custom encoding with the straightforward one-hot encoding for categorical data, both before and after the application of an re-encoding by auto-encoder. My hypothesis is that we can conclude some heuristics

for this process that will always improve the neural network's ability to learn and generalise, as examples given by Bustos R.A. and Gedeon T.D. [1]. More formally, to learn well means the prediction accuracy of the network is positively related to the number of iterations, to generalise well means the testing accuracy grows alongside the training network. If these two traits were non-existent when one encoding is used, and changing the encoding allow us to observe them, encoding can be seen as played an essential role. If both encoding shows these traits, the extent of improvement would be compared. If neither encoding shows these traits, I could exclude two "bad" encodings.

Take the data set we have and the goal we decided as example. To be able to improve the accuracy towards 100%, we first need to ensure that there exist a set of parameters that replicates the entire data set. Therefore, a neural network with 2 hidden layers is used due to it being totipotent and not deep (deep networks require more data to train). However, this is insufficient if we can not approach the set of parameters via learning.

While huge amount of data is clearly a key to better learning, the underlying requirement is actually the information contained in the data. In the case of this data set, the size is limited to only 162 entries, but it has a variety of inputs, each associates with a rich semantic. If we can preserve most of the information contained within the original data set when we do the conversion, we might still be able to get a pretty good prediction accuracy.

Encoding itself can imply many things, such as the distance between two objects in a class, or the relationship between different classes. In general, we consider encoding in machine learning as a way to store the desired information in a binary form so it may be processed by a computer. There's another level of abstraction above this, that is the partition of binary code base and their assigned function (things they represent). Number is an example of such abstraction. What we are facing in the given data, however, is at a even higher level, and the specific meaning of the inidividual columns are unlikely to be preserved without a table or description external to the model. Therefore, one of the targets of a custom encoding is to pass the implications down the abstraction level, so that it can be utilised by a feed-forward neural network that treats all normalised inputs equally.

Besides manual interpretation of the data, an auto-encoder is also a tool we may integrate into the procedures of our prediction. It was claimed that this unsupervised pre-training would improve the prediction accuracy in deep learning settings, and the reason could be its forced extraction of those features that takes the most responsibility in reconstructing the original data, (much like Principle Component Analysis (PCA)[4] and other dimesionality reduction or feature extraction techniques, but in a non-linear way,) and using the encoder's output as the input would allow us to start training at a better position (better in the sense being closer to the static point representing the optimum) in the target space (initialisation_as_regularization [3]), hence avoid being caught in a not so good local minimum and possibly yield a better result in the end.

The idea of adopting this seems appealing, especially when we are dealing with a data with small number of entries but a large number of dimensions. When we are using neural net for prediction, which is a form of local search, the chanllenge comes with this characteristics is exactly to escape from the local minimum. This is hard as stochastic gradient descent (SGD) is dependent on example ordering [3], and the effect of initial entries may not be reversed by the confilicting entries if the size of training set is small.

With all this considered, and the prevalence of data with the above characteristics in the real world (especially for the problems where the cause is unknown and noise were included as the input), it's worth investigating how would a change in data encoding affects learning with our data set. The result would show whether incorporating more information into the encoding via human intervention improves learning as our intuition leads, and the absolute accuracy we can achieve by adopting these methods.

The data I used in this study comes from an experiment that aims at improving search result presentation on smartphones by analysing people's search behavior on mobile devices of different sizes, as captured by an eye-tracker and a JavaScript program. [5] The outputs of the eye-tracker were processed to give the statistics as described in the appendix, checking them is strongly recommended as it would help in understanding the choice of encodings in the method section.

2 Method

When I was processing the data, I followed the following ideas:

1. avoid using certain encodings that prevent the magnitude of each input from being meaningful
2. depict known properties of the data to make the model closer to the target network
3. normalise the value so they will have similar effects (are of similar importance) on the network

2.1 Numerical Data

The data was first analysed by visually inspecting the histograms. Specifically, I plotted the histograms for all the columns that contain numerical values in the initial data. While some exhibited a clear long tail distribution, some of the histograms were very messy, Mean_fixation_duration is one of those.

I then tried to draw the histograms of this column for each test subject, although the number of instances is small, the pattern that's shown is already sufficiently close to something sampled from normal distributions of different variance and mean. This is consistent with our common sense about human-reaction-time-related measurements. Based on this finding, all the time-related columns were normalised by person, rather than normalised all together. Since the information about the Subject is already used here, and it's not useful in generalisation, the column of "Subject" is discarded from the input.

For the remaining columns, no clear pattern was seen on the histograms regardless of the grouping, hence they are kept as is. Since they are all positive numbers, only scaling was performed in normalisation, the position of 0 is unchanged, hence the numbers' relative ratio is preserved.

2.2 Categorical Data

For the remaining two categorical data that is going to be encoded differently, I use the following two approaches:

1. Use a one-hot encoding to encode all the information directly.
2. Use a custom encoding that minimizes the number of inputs, also encode any additional information from the paper with the values representing each class. This way, the number of free parameters will be reduced, the network will also be able to learn from the relationship between categories.

While reversing the process of categorising seems to be a valid method that's generally applicable, it's not feasible here as the raw data was not retained and the description is not sufficiently comprehensive. Investigation was then taken to understand the characteristic of the categorical data.

In the original paper, they were using devices with the same pixel per inch (ppi) for emulating devices of different screen size. Therefore, the resolution is directly proportional to the area of the screen. Since the search results were presented on a 2D space, the area is the value of interest to be considered. Therefore, in case 2, the Screen_size column was converted then to the ratio between its screen area and the screen area of the largest device, with the largest device "L" converted to 1.

To compensate for the low occurrence of numbers greater or equal to 3 in the Clicked_link column, I group them together and make them one big category in both cases. Encoding 2 was adjusted accordingly. Since the value is only weakly positively related to the depth of search, the proportion of values is not important, and for categorical data, increasing the distance between classes would be beneficial for making distinctions, the 3 categories were converted to 0, 0.5 and 1 in the same column. While we could create noisy duplicate of the inputs with lower occurrence, that would cause the other inputs to be uneven, hence it's not implemented.

The eventual encodings are presented as follows: (Table 1 on next page)

2.3 Auto-encoder

Another decision to make is the size of hidden layers in the auto-encoder.

What we are plugging into the neural networks are numbers, and we know that the underlying representation is 32 or 64 bits of 0s or 1s. Take the idea of Minimum Description Length [7] and apply it at the abstraction level of floating numbers, we can determine a minimal number of neurons required in the output of the encoder in the auto-encoder. The number would also be the optimal

Table 1. The encoding I used for each column in the original data for both networks.

Column	**Network 1**	**Network 2**
Subject	used in normalisation, then removed	
Screen Size	one-hot categorical, 3 inputs	1 continuous input, maxed at 1
Time to first click	1 continuous input, normalised by person	
Fixation count	1 continuous input, normalised by person	
Fixation loss	1 continuous input, normalised by person	
Mean fixation duration	1 continuous input, normalised by person	
Task completion duration	1 continuous input, normalised by person	
Page Visit	1 continuous input, normalised	
Wrong answer	Unchanged, 1 input of 0 or 1	
Clicked Link	one-hot categorical, links greater than or equal to 3 were combined	1 continuous input to preserve the depth information of the link
Scrolled	1 continuous input, normalised	
Compressed scanpath value	1 continuous input, normalised	
Minimal scanpath value	1 continuous input, normalised	
Complete	Unchanged, 1 input of 0 or 1	
Linear	Unchanged, 1 input of 0 or 1	
ID (immediate decision)	Unchanged, 1 input of 0 or 1	
Strictly linear	Unchanged, 1 input of 0 or 1	
Mean fixation duration for one link	1 continuous input, normalised by person	
Skip count	1 continuous input, normalised	
Total skip distance	1 continuous input, normalised	
Regression count	1 continuous input, normalised	
Total regression distance	1 continuous input, normalised	
Trackback	1 continuous input, normalised	
Task satisfaction	1 continuous input, normalised	

number for an auto-encoder as it would give the highest level of compression without losing information.

This is however hard to compute in a real world problems like this, but luckily the number of features is not extremely large, hence I tried to test all the possible output sizes for the encoder in the auto-encoder, and select the size with the highest occurance in sizes that gives the minimum loss.

2.4 Output Encoding

For both cases, the output was encoded with equilateral encoding [8][9]. This is chosen as it maximises the loss between each class when squared loss function is used. It also allows each target to be equidistance under this loss function, which is especially suitable for this study as there were no particular relations between the tasks.

2.5 Network

The network I constructed is a fully connected network with two hidden layers, the connections were linear and the activation function is sigmoid. The loss function was squared loss function. Since not close data were found to be working on a similar task, I decide on the size of hidden layers by inspection of input categories and make the convergence gradual. The output base on this network structure seems reasonable.

3 Results

At the end of 20000 epochs, the result is as follows:

Table 2. The result generated by both networks at 10000 epochs

Metric	**Network 1**	**Network 2**
Training Accuracy	98.41 %	96.83 %
Testing Accuracy	69.44 %	83.33 %
Loss	2.3142	3.8604

I had also plotted each metric with respect to number of epochs to reveal the trend.

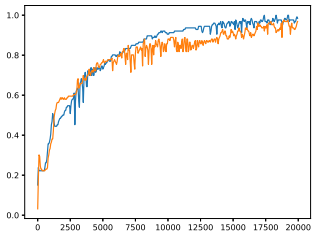

Fig. 1. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

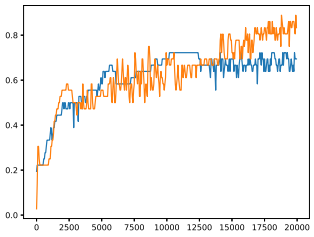

Fig. 2. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

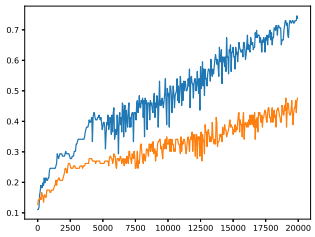

Fig. 3. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

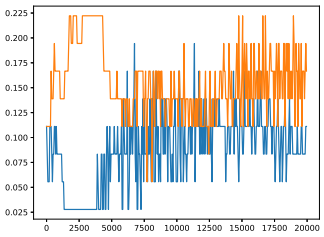

Fig. 4. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

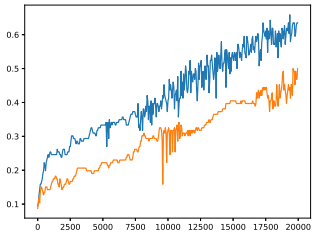

Fig. 5. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

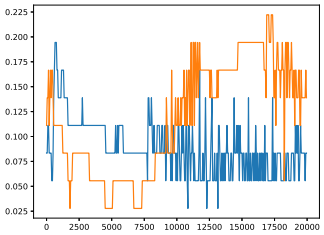

Fig. 6. The training accuracy of both implementations. The blue line is the case with one-hot encoding. The orange line is the case with custom encoding. The horizontal axis is the number of epochs. The vertical axis is the accuracy in decimals.

4 Conclusion

The initial result as submitted in the previous version was aligned with my hypothesis, but the magnitude of improvement (the accuracy has increased from 25% to 75% on testing, a 50% improvement) was out of my expectation. Upon checking the training set I used, I discovered that I had wrongly added the initial target column (before it was converted to equilateral encoding) into the inputs for network 2 (the one with custom encoding), and the Screen_Size input was omitted. Therefore, what network 2 has done could simply be replicating the equilateral encoding based on this single column.

After fixing this error in the setup, I did some further comparison and found that network 2 still outperformed network 1 (the one with straightforward one-hot encoding) most of the time. This suggests that the difference does not come from random initialisation of parameters and differences in data sampling solely, and must have a systematic source, which is the controlled variable: difference in encoding. A typical trend shown in the figures (see Fig.1 and Fig.2) is that for network 2, both the training and testing accuracy would improve very quickly in the first few iterations, and overfitting is prevented asymptotically.

There were some subsequent change to the model due to the change in encoding, which could possibly explain the difference. Firstly, the size of input vector is smaller in case 2, meaning that there are slightly fewer free parameters between the input layer and the first hidden layer, which would allow faster training and prevent overfitting. Secondly, the custom encoding contains slightly more information about the relationship between the categories in the two categorical data. The relative magnitude could be connected to the task in some way, leading to network 2 having a better performance.

To further control the number of free parameters and look at the exact benefit coming from the change in encoding, I changed the input of the prediction network to be the output of the encoder in auto-encoder, and forced their size to be the same smaller number.

The first thing I noticed is the clear drop in both training and testing accuracy. This suggests that the extraction of features, which was desirable in deep learning settings, is actually unfavoured when working on shallower networks and a rather simple task. Having the features combined so that the input size is small, actually restricted the fitting of the network, reducing it's prediction accuracy. The second thing I noticed is the trend shown by both networks are clearly much far apart now, proving my above idea of difference in number of free parameters being a source of difference.

With this idea, I did a further testing, this time instead of compressing the network, I use the auto-encoder to exapand the network. Network still performs better than the network 1 in terms of testing accuracy. Comparing the trend with what we have in the previous experiment, the addition of redundant hidden neurones have not changed the feature that was extracted and repeated testing shows a similar trend each time, suggesting that using an auto-encoder do act as a powerful initialiser.

The takeaway from this is that re-encoding the data based on our knowledge about data generation and the field of study is important in neural network construction, an appropriate encoding will speed up learning and prevent overfitting. Additionally, adding an auto-encoder to the data pre-processing procedure will allow us to reveal a consistent underlying trend even when only a small amount of data is used, regardless of the number of hidden neurones chosen. This would be helpful when designing network structure for large data sets as we can experiment with this and come up with sturcutures closer to the target. Finally, if we are involved in the data collection process and the aim is to use the data on a neural network, associating meaningful numerical values with categorical data would be very helpful.

References

1. Bustos R.A., Gedeon T.D. (1995) Decrypting Neural Network Data: A Gis Case Study. In: Artificial Neural Nets and Genetic Algorithms. Springer, Vienna
2. Alberto Fernández, Salvador García, Francisco Herrera, and Nitesh V. Chawla. 2018. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. J. Artif. Int. Res. 61, 1 (January 2018), 863–905.
3. Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why Does Unsupervised Pre-training Help Deep Learning? J. Mach. Learn. Res. 11 (3/1/2010), 625–660.
4. Plaut, E. (2018). From principal subspaces to principal components with linear autoencoders. arXiv preprint arXiv:1804.10253.
5. Kim, J., Thomas, P., Sankaranarayana, R., Gedeon, T. and Yoon, H.-J. (2016), Understanding eye movements on mobile devices for better presentation of search results. J Assn Inf Sci Tec, 67: 2607-2619. https://doi.org/10.1002/asi.23628
6. Salvucci, Dario & Goldberg, Joseph. (2000). Identifying fixations and saccades in eye-tracking protocols. Proceedings of the Eye Tracking Research and Applications Symposium. 71-78. https://doi.org/10.1145/355017.355028
7. A. Barron, J. Rissanen and Bin Yu, T̈he minimum description length principle in coding and modeling,ïn IEEE Transactions on Information Theory, vol. 44, no. 6, pp. 2743-2760, Oct. 1998, doi: 10.1109/18.720554.
8. Masters, T, Practical Neural Network Recipes in C, Academic Press, Boston, 1993.
9. Equilateral Encoding, https://www.heatonresearch.com/2017/02/10/equilateral.html. Last accessed 31 May 2020