Decrypting data and implement genetic algorithm in a neural network

Xiao TIAN
Xiao.Tian@anu.edu.au

Research School of Computer Science, Australian National University
108 North Road Acton ACT 2601 Canberra, Australia

Abstract. Some data may be not valid for training back-propagation neural networks, hence affect the implementation of neural networks. In this report, I use PyTorch to build a neural network and use data normalisation and pattern reduction method to maximise the performance of the network. The task of the network is to predict whether the product will be subscribed based on the clients' personal and financial information. In the process of improvement, I use PyTorch to process the very large number of erratically reliable data and to produce an acceptable level of accuracy and confusion matrix. The dataset was also used in the article *A data-driven approach to predict the success of bank telemarketing.* The result classification correct rates in this report are higher than the results produced in the article. However, the methods used in the published article are more thoughtful.

Keywords: back-propagation, genetic algorithm, mutation, unbalanced data set

1 Introduction

Traditional marketing selling strategy focuses on sell products to target clients. Marketing selling campaign is one of a business strategy of companies. Before publishing a new product, companies should know who are target customers of this product hence build an effective selling strategy. Directly contact customers by phone is one marketing approach helping companies to focus on possible buyers. However, by use technology, i.e. Neural network, companies access target clients without actual products. We predict whether the client would buy the product by analysing detail information of clients. Rethinking marketing strategy focus on creating the product that client want (Rust, Moorman, & Bhalla, 2010).

In this paper, I built a neural network to solve a classification problem, that to access whether a client will subscribe a term deposit by using back-propagation of error measure and then use normalise data and reduce the pattern to improve the neural network.

1.1 Dataset

There are four reasons for choosing this dataset. Firstly, it has 21 attributes and 41188 instances, which is sufficient enough for pattern pruning. Secondly, the dataset is suitable for training a classification problem-solving neural network. Moreover, a big dataset is good for parents choosing of generation in a neural network. Fourthly, this dataset is unbalanced and contains erratically reliable data and noise data. I have promoted the neural network in my last paper by balancing the dataset. Nevertheless, I focus on continuing to promote the neural network, to see how the genetic algorithm performances in pattern pruning.

The raw dataset for this report is from the article A data-driven approach to predict the success of bank telemarketing published by Moro, Cortez and Rita in 2014. It is related to marketing campaigns which were proceeded by phone inquiring. The marketing campaigns accessed whether the client would subscribe the product or not.

INPUT VARIABLES	
# bank client data	
1 - age (numeric)	
2 - job	type of job
3 - marital	marital status
4 - education	
5 - default	has credit in default?
6 - housing	has housing loan?
7 - loan	has personal loan?
# related with the last contact of the current campaign	
8 - contact	contact communication type
9 - month	last contact month of year
10 - day_of_week	last contact day of the week
11 - duration (numeric)	last contact duration, in seconds
# other attributes	
12 - campaign (numeric)	number of contacts performed during this campaign and for this client
13 - pdays (numeric)	number of days that passed by after the client was last contacted from a previous campaign
14 - previous (numeric)	number of contacts performed before this campaign and for this client
15 - poutcome	outcome of the previous marketing campaign
# social and economic context attributes	
16 - emp.var.rate (numeric)	employment variation rate - quarterly indicator
17 - cons.price.idx (numeric)	consumer price index - monthly indicator
18 - cons.conf.idx (numeric)	consumer confidence index - monthly indicator
19 - euribor3m (numeric)	euribor 3-month rate - daily indicator
20 - nr.employed (numeric)	number of employees - quarterly indicator
Output variable (desired target):	
21 - y	has the client subscribed a term deposit? (binary: 'yes','no')

Table 1 Attributes

1.2 Problem and modelling

The main driven of this report is to determine the performance of the neural network for applying genetic algorithm for pattern pruning. Generally, the application of neural network for the genetic algorithm emphasises on improve the architecture of the neural network (Uber Engineering, 2017). In this report, the genetic algorithm is applied to solve dataset pruning problem and generate an optimal dataset from original dataset.

The initial design principle is that, in the beginning, the genetic algorithm selects 3 datasets at random from the current dataset to be parents and uses them as pairwise coupling to produce 3 children datasets for the next generation. Therefore, there are 9 children dataset in total at each generation. Then the algorithm keeps the top 3 well-performance children datasets to produce next generation and discards the remaining datasets in this generation. The measurement of performance will be discussed in section 1.3. Over successive generations, the population "evolves" toward an optimal dataset.

This report implements an evolutionary neural network, at each generation, the network is trained using a training set of input patterns with desired outputs, using the back-propagation of error measures. The network is tested using a validation set of patterns which are never seen by the network during training and thus can provide a good measure of the generalisation capabilities of the network. The separation of the total set of patterns into training and test sets is generally at random to avoid introducing experimenter bias.

1.3 Performance measurement

There are 6 factors are considered to reflect the performance of the evolutionary neural network. Loss of training dataset, the accuracy of training and testing dataset, confusion matrix of training and testing dataset and the training time. As in an optimization problem, the goal is to minimize a loss. Therefore, after every training epoch, the network calculates and prints out the loss by using loss function (discussed in section 2.2) and calculates and prints out accuracy as well.

The reason why the accuracy of the testing dataset is needed is to avoid overfitting. The neural network may be too familiar with the training dataset (as discussed in section 1.2) to effectively classify other data. In other words, even the training accuracy is perfect, it doesn't mean the network performances well, it may overfit the training dataset. In this case, I calculate the accuracy of the testing dataset and compare it with the accuracy of training dataset to see whether the network performs well or it just overfit the training dataset.

Nevertheless, accuracies are not the whole story. For example, if the dataset is unbalanced, 90% of the target is class 1, then the network performances with 90% of accuracy even if it classifies all input as class 1. Therefore, I use confusion matrix to see the number of actual class and predicated class. The last measurement is the training time, or training speed, to evaluate if the network is effective. Moreover, another problem with a long training time is that I need to put effect to prevent it to be interrupted.

2 Method

2.1 Dataset pre-processing

As can be seen in section 1.1, 11 attributes, including output, in the dataset are non-numeric. Therefore, the first step I do to process the data is changing them to number. In this report, I use Python to implement a neural network. Firstly, import all required libraries, here including torch, numpy, pandas, Variables form torch.autograd and activation function. Secondly, read and process data by using PyTorch. Before processing the dataset, I delimit it by attributes, i.e. divide the dataset into 21 columns. Then, replace all non-numeric attributes to sequential numbers.

According to Bustos and Gede, in some case, the raw data contains a significant number of erratically reliable data may affect the means of prediction. After pre-processed as above, the data set is shown in Table 1. In this case, I divide every number by the maximum absolute value of the attributes it belongs to. Hence, every attribute value should be normalised over the range -1 to 1 for the logistic function. This is because the values of attributes, for example, "duration" which represents last contact duration, in seconds, are erratically and contain considerable gaps, which may relatively affect the performance of the network. Then, I split attributes into two parts, as the first 20 columns are clients' information while the last one is the target and convert pandas dataframe to array and create x_array and y_array to store inputs and outputs.

NO.	ATTRIBUTES	VARIABLES
1	age	17, 18, …, 92, 94, 95, 98
2	job	0, 1, …, 11
3	marital	0, 1, 2, 3
4	education	0, 1, …, 7
5	default	0, 1, 2
6	housing	0, 1, 2
7	loan	0, 1, 2

8	contact	0 1
9	month	0, 1, …, 9
10	day_of_week	0, 1, 2, 3, 4
11	duration	0, 1, 2, ..., 3785, 4199, 4918
12	campaign	1, 2, …, 42, 43, 56
13	pdays	0, 1, 2, …,25, 26
14	previous	0, 1, …, 7
15	poutcome	0, 1, 2
16	emp.var.rate	-3.4, -3.0, -2.9, -1.8, -1.7, -1.1, -0.2, -0.1, 1.1, 1.4
17	cons.price.idx	92.201, 92.379, ..., 94.601, 94.767
18	cons.conf.idx	-50.8, -50.0, -49.5, …, -26.9
19	euribor3m	0.634, 0.635, …, 5.045
20	nr.employed	4963.6, 4991.6, …, 5195.8, 5228.1
21	y	0, 1

Table 2 Variables

A feature of the banking dataset is that it is unbalanced, 88.7% clients choose not to subscribe the product, i.e. "no" in column y. This may be resulting in a significant bias in the network. there still is an 88.7% of accuracy even if the network classifies all inputs as "no". For later genetic algorithm implementation, I divide the dataset into two set, one contains all the instances that target 0, which presents 88.7% of the original dataset, named data_train_0; another contains all the instances that target 1, which presents 11.3% of the original dataset, data_train_1.

When training a neural network, a huge set of patterns not always be effectiveness (Chauvin, 1990). The risk of overfitting rises as the size of dataset increase. In this case, I divided each of data_train_0 and data_train_1 into two parts, which are 80% and 20% of them, for training and testing respectively. Then, concatenate them to be training dataset (named data_train_array) and testing dataset (named data_test_array). The last step is to create Tensors to hold inputs and outputs and wrap them in Variables since Torch only trains neural network on Variables.

2.2 Implement a neural network using PyTorch

After pre-processing dataset, I implement a simple three layers neural network with 20 input neurons, 30 hidden neurons and 2 output neurons. Cross Entropy Loss function is used to compute the amount by which the prediction deviates from the actual values, which will be stored for visualisation in this network. Adam is used as the optimiser to train the network, which will hold the current state and will update the parameters based on the computed gradients. The reason why I choose Adam rather than SGD or Rprop will be discussed later. The learning rate is set as 0.01. Before performing backward pass, the gradients need to be cleared. And the final step is to implement a neural network is call the step function on an optimiser to update its parameters.

The network was initially trained for 1,000 epochs and print out the accuracy every 50 epochs. After implementing the GNN network, I encapsulate it to be as a function package and will call it in genetic algorithm later (will be discussed in section 2.3).

2.2.1 Optimiser

There are three optimisers that I think may be helpful to this network. I experiment with them separately to find the most suitable one in the GNN network. The training accuracies and testing accuracies are shown in figure 1 and figure 2. The SGD contributes to a higher testing accuracy while it has the lowest training accuracies among them. Using SGD may lead to the overfitting problem. On the other hand, Adam and Rprop performance quite similar whether on the

training speed or training accuracies, but Adam contributes to higher training accuracies. Each of time costs about the same time to train. Therefore, I choose Adam to be the optimiser.

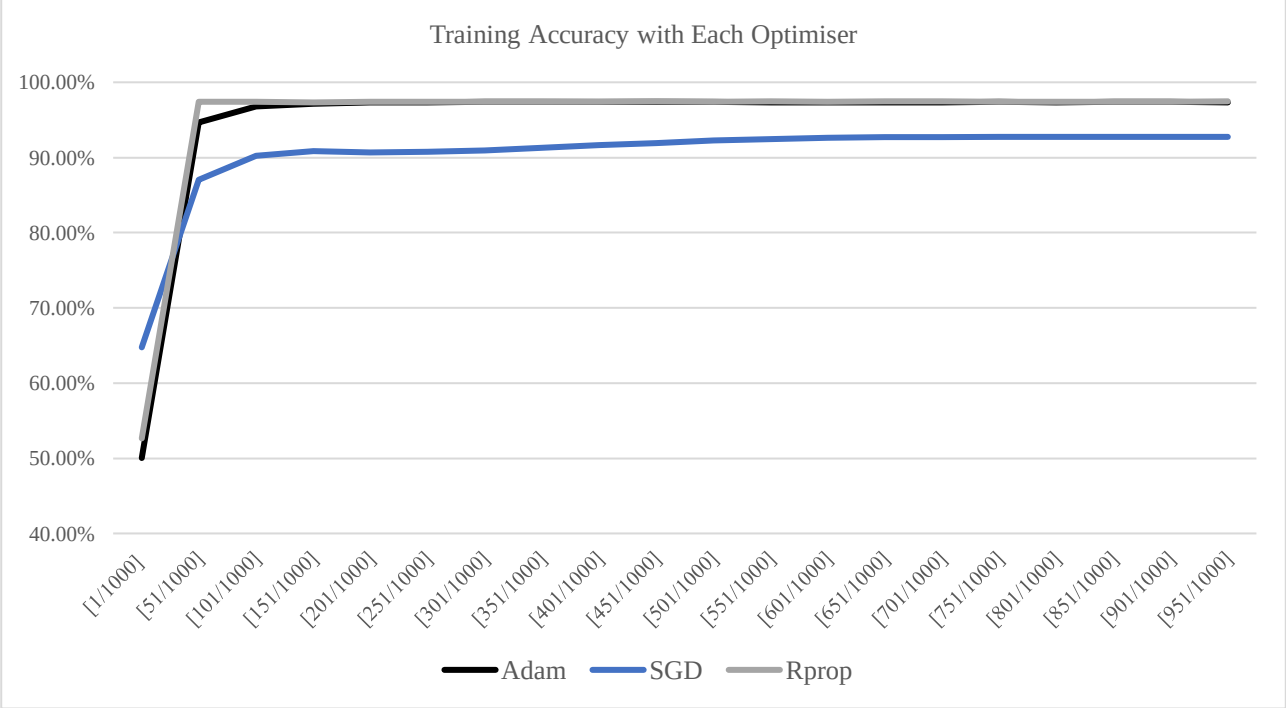

Figure 2 Training Accuracy with each Optimiser

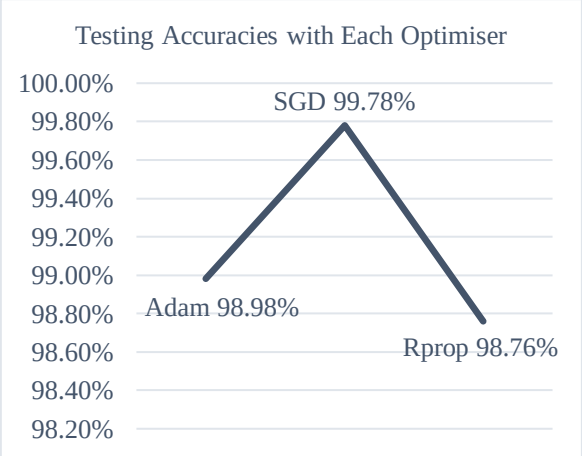

Figure 1 Testing Accuracy with Each Optimiser

Epoch [951/1000] Loss: 0.0581Accuracy: 97.52 %
confusion matrix for training:

3645	67
125	3587

Testing Accuracy: 98.87 %
confusion matrix for testing:

907	21
0	928

Figure 3 Results

2.3 Genetic algorithm

In another python file, I implement the genetic algorithm and import GNN, apply the algorithm by calling GNN. Firstly, I generate three datasets as parent datasets by randomly deleting 30% of the total training dataset. Then every two parent datasets are in a pair, i.e. A and B, B and C, C and A, and produce three children datasets. At every generation, 9 datasets are produced. While the best performed three children datasets are kept and will produce next generation, the other six datasets will be discarded. The performance here mainly depends on testing accuracy. This is because genetic neural network costs time to train (100 generations production spent 3 hours), the training time is not considered in the genetic algorithm.

2.3.1 Further improvement

The significance of genetic algorithm is that it evolves bio-inspired operators such as mutation, crossover and selection to generate an optimal solution. In this report, I adopt mutation to complete genetic algorithm to maintain diversity. The testing accuracies in first 100 generations float between 57% to 62%. Which is not as good as I expected. This may be resulting from the lack of genetic mutation. In this case, children datasets always have same features as that of parents. Therefore, I further improve the genetic algorithm by mutating every generation. 10% data of every child dataset are replaced to equivalent randomly selected data from the original dataset.

The algorithm produced 10000000 generations, the testing accuracies of 3 best-performance children datasets in each generation float between 52% to 64%.

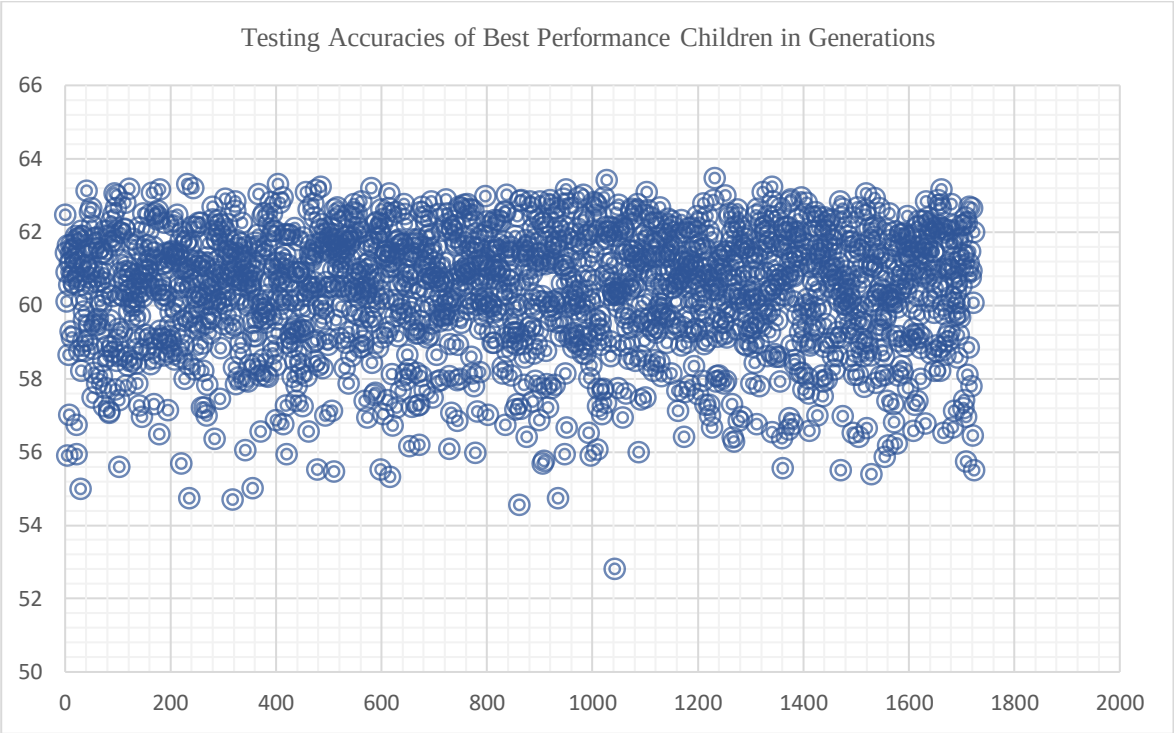

Figure 4 Best accuracies in each generation

2.3.2 Problems

The genetic algorithm is time-consuming, it takes hours, even days to train. In the process of training, the data may be lost, for example, the training is interrupted. Therefore, I create a file to store the top 3 well-performance children datasets and update them after every generation. Once the training process is interrupted, the newest 3 children datasets and be used to produce next generation. Moreover, the PyTorch pushes automatically after every generation.

3 Results and Discussion

As discussed in section 2, I implement a back-propagation neural network to solve classification problem and a genetic algorithm. Then genetic algorithm selects three parents (A, B, C), each of them is randomly 70% of the original training dataset. every parent and the other two are made in pairs respectively, hence there are 3 pairs in total. Every pair produces 3 children datasets, hence there are 9 children datasets in total at every generation. Mutation is implemented here to maintain diversity.

The results of the initial network are shown as in Figure 3. Although the training speed is fast, the loss is acceptable, and the accuracies are high, this network does not perform well. The reason is as discussed above that the dataset is unbalanced. The results of the genetic neural network are shown in Figure 4. The network is as effective as I expected.

The banking dataset was previously used by Moro, Cortez and Rita in 2014. They used "using the rminer package and R tool and conducted on a Linux server, with an Intel Xeon 5500 2.27 GHz processor" (Moro, Cortez, & Rita, 2014). The results are shown below:

Final set of selected attributes.

Factor	Attributes	Description	AUC
1: Interest rate	nat.avg.rate	National monthly average of deposit interest rate	0.781
	suited.rate	Most suited rate to the client according to bank criteria	
	dif.best.rate.avg	Difference between best rate offered and the national average	
2: Gender	ag.sex	Sex of the agent (male/female) that made (outbound) or answered (inbound) the call	0.793
3: agent experience	ag.generic	If generic agent, i.e. temporary hired, with less experience (yes/no)	0.799
	ag.created	Number of days since the agent was created	
5: Client–bank relationship	cli.house.loan	If the client has a house loan contract (yes/no)	0.805
	cli.affluent	If the client is an affluent client (yes/no)	
	cli.indiv.credit	If the client has an individual credit contract (yes/no)	
	cli.salary.account	If the client has a salary account (yes/no)	
7: Phone call context	call.dir	Call direction (inbound/outbound)	0.809
	call.nr.schedules	Number of previously scheduled calls during the same campaign	
	call.prev.durations	Duration of previously scheduled calls (in *s*)	
8: Date and time	call.month	Month in which the call is made	0.810
9: Bank profiling indicators	cli.sec.group	Security group bank classification	0.927
	cli.agreggate	If the client has aggregated products and services	
	cli.profile	Generic client profile, considering assets and risk	
10: Social and economic indicators	emp.var.rate	Employment variation rate, with a quarterly frequency	0.929
	cons.price.idx	Monthly average consumer price index	
	cons.conf.idx	Monthly average consumer confidence index	
	euribor3m	Daily three month Euribor rate	
	nr.employed	Quarterly average of the total number of employed citizens	

(Moro, Cortez, & Rita, 2014)

In Moro, Cortez and Rita's experiment, they use AUC to evaluate the network. The attributes are classified as 8 different factors. Each type of factor is used individually to predict the results. In other words, the possibility of product subscribed by a client is affected by different factors, including interest rate, gender, agent experience, client–bank relationship, phone call context, date and time, bank profiling indicators, social and economic indicator. In Moro, Cortez and Rita's article, they considered the actual factor in the real world. Although those result figures are all lower than mine, those results are more truthfulness. In this case, my results are more general when compare to results in Moro, Cortez and Rita's article.

4 Conclusion and Future Work

The initial experiment shows that a simple three-layer neuron network cannot do a good job in marketing classification of an unbalanced dataset. While a complex network may take a lot of time to train, the genetic network still cannot perform well in dataset optimisation problem.

One guess why the genetic neural network cannot achieve a high-quality performance is that the dataset is unbalanced, as the number of class 0 and class 1 are 32950: 8598, hence the network is not familiar with class 1. Therefore, no matter how good the dataset is after pruned by genetic algorithm, it still is not balanced.

For future work, I would like to, as Moro, Cortez and Rita did in their experiment, detail analyse and categories data according to the actual world for network improvement and may use AUC as the measurement to evaluate the network.

5 References

Bustos, R. A., & Gedeon, T. D. (1995). Decrypting Neural Network Data: A Gis Case Study. *Artificial Neural Nets and Genetic Algorithms*, 231-234.

Chauvin, H. (1990). Dynamic Behaviour of Constrained Back-Propagation Networks. *Proc. NIPS-2*, 642-649.

Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems, 62*, 22-31.

Rust, R. T., Moorman, C., & Bhalla, G. (2010). Rethinking Marketing. *Harvard Business Review, 88*(1,2), 94-101.

Uber Engineering. (2017, 12). *Welcoming the Era of Deep Neuroevolution*. Retrieved 5 2018, from UBER Engineering: https://eng.uber.com/deep-neuroevolution/