Predicting crashes in climate model simulations through artificial neural networks

C Treglown
Research School of Computer Science
Australian National University

Abstract:

Climate simulation models are extremely large and complex pieces of software that are expensive to run. In this study a neural network model was developed to predict the successful completion or crashing of a climate simulation by examination of Parallel Ocean Program components of the climate simulation input. Attempts have been made to improve the network performance and speed using genetic algorithms and further improvement through network pruning was planned. A genetic algorithm was designed to identify salient features or combinations of features and eliminate irrelevant features. The dataset used was extremely unbalanced and attempts were made to improve the training of the neural network, but these attempts were unsuccessful, resulting in a model that always predicts a successful simulation.

1. Introduction:

In this paper I have attempted to investigate the problem of climate simulation crashes using artificial neural networks. Climate simulations using high-end models are incredibly complex. They are large pieces of software and typically use hundreds or thousands of files. Correspondingly, running such simulations is computationally expensive and therefore avoiding failures is desirable. This problem was investigated by Lucas in 2013 [1]. In the system they were interested in, the failure rate of simulations was 8.5 %. [1] Using Lucas' work as inspiration, I sought to investigate the same problem and dataset and to compare results to those found by Lucas. The data was taken from the ICS dataset archive and was originally donated by Lawrence Livermore National Laboratory [2] and is a classification problem.

Lucas sought to determine in his 2013 study which of 18 parameters or combinations of parameters in the Parallel Ocean Program (POP2) component of the model were predictors of failure. The support vector machine classification machine learning approach applied in the study led to a system which could predict the likely outcome of a simulation run with above 97% accuracy [1]. Lucas et al. were also able to determine the predictive features of the dataset [1]. I sought to build a simple back-propagation neural network with one hidden layer that could achieve similar results and to determine the sensitivity contributions of the input parameters for comparison with those identified by Lucas et al.

The performance of neural networks can be improved through a variety of techniques. One of these techniques is network pruning [3]. Individual neuron contributions can be determined through a variety of metrics and low performing or redundant neurons removed by setting their weights to 0, effectively removing them from the network. This can have the benefit of making the network smaller and faster while retaining similar and sometimes improved functionality compared to the originally trained network. The impact of pruning becomes more noticeable in deeper networks because deep hidden layers are more likely to be pruned [4]. While my original goal in this research was to implement pruning on a network that had been trained with the aid of genetic algorithms, I couldn't generate a working implementation of the network.

Genetic algorithms can also be used to improve the performance of genetic algorithms, for example they may be used to select salient features or to find optimal network hyper parameters. In this study, a genetic algorithm was employed in an attempt to determine which features were important in correctly predicting a simulation's outcome. Genetic algorithms work by generating a 'population' of 'chromosomes' with randomised values. Chromosomes which give the best results have a higher chance of propagating their genes through further generations, allowing the algorithm to find an optimal solution for a given problem. There are various methods through which gene propagation can be achieved including tournament selection, proportional selection, rank-based selection and others. The genetic algorithm used in this study employs a mixture of proportional selection, random selection and generation of new chromosomes.

2. Methods:

The model design in this study was a fully-connected feed-forward neural network, originally using stochastic gradient descent as a gradient computing technique. The model was implemented using the PyTorch library which is an open source piece of software for Python developed for machine learning, largely by Facebook. The purpose of this model design was to generate a neural network that could be trained on the data set and accurately predict between two classes. The model itself consists of 18 input neurons which represent the input parameters of the POP2 dataset. The one hidden layer of the network consists of 14 neurons. There are two neurons in the output layer, representing the prediction of a successful or unsuccessful simulation. Hyperparameter selection was based on earlier work with 300 epochs chosen as the standard for this study and a learning rate of 0.001.

2.2 – The Data and Preprocessing

The data when used in Lucas' study [1] was split into 3 sub-studies. In my work I combined the data into a larger set as there was no need for smaller studies with my investigation goals. The first two columns of the data were not included as they were identification numbers and therefore not relevant to the investigation. The data investigated had 540 instances and 18 attributes. Each time the program is run it randomly splits the data into two groups, a training group (80% of the total) and a testing group (20% of the total). It was found during training that the net was always predicting a successful study, resulting in false negatives where the study is predicted to succeed but actually failed when the simulation was run. Datasets that are imbalanced are known to result in class overlap [5]. The POP2 dataset is extremely imbalanced with successful simulations greatly outnumbering unsuccessful simulations with only 46 failures out of 540 simulations. This was determined to be the reason for the net always predicting a successful outcome.

2.3 – The Genetic Algorithm

The genetic algorithm was designed to determine which features in the POP2 dataset had the highest sensitivity contributions to correct predictions. To achieve this, a chromosome of random bits equal in length to the number of input features was designed. A chromosome of this type can be used to mask the input data, effectively removing the input features from the network training and testing. The fitness of a network which had been masked by a particular chromosome could be determined by measuring its testing accuracy, thus that fitness would be associated with that chromosome.

An initial population of chromosomes was to be generated and the top two fittest would produce 2-4 offspring, the number would be tweaked based on the population size and number of generations chosen. The goal was to maintain genetic diversity while applying sufficient selective pressure to ensure convergence on the ideal selection of features. Crossover of the two fittest chromosomes was achieved through an equal chance of selecting the value from either chromosome for a particular gene, the resulting children should be approximately an even mix between the two fittest individuals. The remainder of the population would be made up from a mixture of chromosomes from the previous generation's population and some new randomly generated chromosomes to introduce some new genetic diversity. Mutation of the population was to be achieved by flipping only one random bit if a chromosome was selected for mutation by the defined mutation rate.

The parameters to be used in the genetic algorithm were to be tweaked based on performance, but as the net they were being applied to was not correctly predicting outcomes, it was impossible to extract meaningful data about the genetic algorithm performance. A population of 50 chromosomes was planned as the initial value as that is a high enough value to ensure some genetic diversity while not extending the program running time excessively. The number of generations was to be 20.

The genetic algorithm was to be implemented with a copy of the same data set and neural net, so each genetic algorithm was tested with the same variable (initial network weights and data split between training and testing). The logic here is that the results would be more comparable as we are keeping as many variables controlled as possible.

2.4 – Efforts to Overcome the Imbalanced Data

In efforts to overcome the previously described data imbalance issues, the dataset was spiked with extra copies of the minority class, these copies were modified by adding random noise to effectively create more unique examples of the minority class for training. The minority class examples were padded until they reached a ratio of 2:3 with the majority class. These extra examples were added in excel to a copy of the dataset and interspersed randomly through the data set.

Another attempt made to overcome the data imbalance was to increase the learning rate to increase the impact of training examples for the minority class.

3. Results and Discussion:

3.1 – Predicting Simulation Failures:

Initial results produced by the network appeared to be in line with those found by Lucas et. al. [1], with testing discrimination accuracy ranging between 87 % and 93%. However, results produced by the genetic algorithm were identical between generations and between chromosomes within the same generation, i.e. each chromosome was producing the same testing discrimination accuracy. To investigate this, a confusion matrix was included for both training and testing, the results were very telling, the results are shown in the confusion matrices in figure 1.

		Predicted Class	
		Fail	Succeed
Actual Class	Fail	0	66
	Succeed	0	374

		Predicted Class	
		Fail	Succeed
Actual Class	Fail	0	13
	Succeed	0	106

Figure 1 - Average confusion matrix for training (left) and testing (right)

In all training examples and test examples, the neural network had an output of 1, indicating that it was always predicting a successful simulation run. As described in the methods section, this is owing to the extremely unbalanced data set. This one-way class prediction is common in unbalanced data sets [5] and so is not surprising in this case. The net's current design was not appropriate to this data set and steps were taken in an attempt to correct the behavior. The reason that the network's poor performance was not detected earlier is because the ratio of success to fail cases in the data percentage is only slightly lower than the results achieved by Lucas et.al. which is what I was expecting in terms of neural net performance. This is called the 'accuracy paradox' and occurs when the results are merely reflecting the underlying class distribution.

3.2 – Correcting imbalanced data problems:

The addition of failed simulation data examples with random noise added did not improve the network's performance in this case, the results are summarised in the confusion matrices in figure 2.

		Predicted Class	
		Fail	Succeed
Actual Class	Fail	0	238
	Succeed	0	370

		Predicted Class	
		Fail	Succeed
Actual Class	Fail	0	70
	Succeed	0	109

Figure 2 - Average confusion matrix for training with added fail examples for training (left) and testing (right)

It was expected that the net would make some predictions of a failed simulation run but none were made. However, as the dataset is still somewhat unbalanced this is not entirely surprising. Additionally, the most sensitive features may be vulnerable to small changes and the randomly added noise may have been inappropriate and diluted the effect of those examples.

The learning rate was increased incrementally starting with small increments and finally up to 0.4. There was no measurable difference in network outputs. Although this approach was unsuccessful, it was a sensible approach to solving the problem as a higher learning rate may have gleaned a greater impact from those examples of failed simulations and the salient features associated with them.

As neither of these investigations yielded improvements in the network's performance, I would have liked to try some other approaches and try combinations of approaches to yield some results but was not able to due to time constraints. Training the network using batches on mini batches may also help improve the net's performance.

3.3 – Genetic Algorithm:

Owing to the neural network not performing as expected, the genetic algorithm couldn't be used to identify the sensitivity contribution of individual parameters. Despite this, the individual functions that make up the algorithm were all independently tested and found to work correctly. The genetic algorithm correctly updates the population with each generation although the fittest members of the population are effectively being randomly chosen for cross-over and breeding because there is never an accurate measure of fitness. The updating of the population can be visualised by uncommenting the appropriate print statements in the provided code.

It's worth noting that using a genetic algorithm for feature selection with a relatively small number of features such as occurs in the POP2 data of this study may not be a useful exercise in terms of achieving performance improvement. However, it is worthwhile from the point of view of trying to confirm the findings in the aforementioned results as to the features with the highest sensitivity contribution [1].

4. Conclusion and Future Work:

In this study I was able to generate a genetic algorithm which appears to work but needs to be appropriately tested with another dataset. This algorithm could be easily adapted to fit many binary classification problems. The next step for this part of the research is to apply the genetic algorithm to a more balanced binary classification dataset, preferably with a large number of features so-as to properly leverage the value provided by the feature selection functionality.

The neural network did not produce correct predictions as hoped, the net invariably overfits to predicting a successful outcome of the simulation. The network could be used on a different, more balanced dataset. Further work to improve the performance of the net on the POP2 dataset could be very broad, approaching the problem from both the data side and the net side. The number of hidden neurons and layers of hidden neurons could be tweaked. Activation functions and optimisers could be varied to find combinations that produce better results. The net could also use a penalised model which would bias the net to paying more attention to the minority class, although this is certainly outside the scope of this study. The data could be modified as described in the methods section but with different random noises added. Batch training combined with other net strategies described above may result in a net which is better able to handle the extreme imbalance in the dataset.

The final step within the scope of this research, after a working net has been produced through use of the genetic algorithm, the resulting net could be pruned using the techniques described by Gedeon and Harris in their 1990 paper [3]. The pruning techniques would each be applied to a copy of the same already-trained network to allow comparison of their effectiveness against each other.

References:

[1] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic and Y. Zhang, "Failure analysis of parameter-induced simulation crashes in climate models," *Geoscientific Model Development,* pp. 1157-1171, 2013.

[2] "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes. [Accessed April 2018].

[3] T. Gedeon and D. Harris, "Network Reduction Techniques," in *Proc. Int. Conf. on Neural Neworks Methodologies and Applications*, San Diego, 1991.

[4] J. Gil, "Pruning deep neural networks to make them fast and small," 2017. [Online]. Available: https://jacobgil.github.io/deeplearning/pruning-deep-learning. [Accessed April 2018].

[5] R. &. B. G. &. M. M.-C. Prati, "Class Imbalances versus Class Overlapping: An Analysis of a Learning System Behavior," in *LNCS (LNAI)*, 2004.

[6] E. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks,* pp. 239-242, 1990.

[7] J. Sietsma and R. J. F. Dow, "Neural net pruning-why and how," in *International Conference on Neural Networks*, San Diego, 1988.

[8] Y. L. Cun, J. S. Denker and S. A. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2*, San Francisco, Morgan Kaufmann Publishers Inc., 1990, pp. 598-605.