Generating fake data for use in testing machine learning methods using evolutionary data generators

Edan Landow

Research School of Computer Science, Australian National University
u6378020@anu.edu.au

Abstract. Most experimental machine learning algorithms rely on data to test their performance and practicality when evaluating them in experiments. However real-world data is not always available for this, and testers often must create their own dataset for use in testing. Sometimes these datasets can be of very low quality and impede the validity of the results. Instances of good fake data generation is typically very specific to the problem, and it is hard to find generalised fake data for use in a larger range of algorithms. This paper attempted to make an evolutionary algorithm that could produce better quality fake data for these experiments that also generalised well to a larger range of problems. It was not successful, and the data produced was of poor quality, however this may have been due to the flaws in the experiment which are expected to have reduced the quality of the results.

Keywords: Data Generation, Evolutionary Algorithm, Functional Transformation

1 Introduction

There is a limited amount of freely available data for testing machine learning algorithms, making it difficult in many cases to find relevant data to test new techniques in machine learning, where the testers of the new techniques can fail to find relevant, real world data for their tests. As a result, testers often must rely on fake data constructed by the testers, which may have large flaws and reduce the validity of experiments done on the constructed dataset. In the paper analysing fuzzy signatures to improve SARS detection, by Kok Wai W, Tamas G and Laszlo K [1], there was no real-world data to use in the experiments. Instead, the authors constructed their own data to use in the experiment. This data was not good for testing, and a paper by Edan Landow [2] which used this dataset found it to be classifiable with 100% accuracy using a linear model with 0 hidden layers. Due to this, improvements caused by the method in the paper could not be identified in testing, as the accuracy could not be improved by positive changes.
Evolutionary algorithms are commonly used to refine algorithms for specific tasks, in cases where humans are unable to produce suitable parameters directly. In this paper, the use of evolutionary algorithms to produce good quality fake data is tested, for use in experiments where real world data cannot be feasibly obtained.

Many other papers have also explored ways to generate data for machine learning, in the case of real data being infeasible to obtain. The paper by Mohammad F, Ananda G and Katarina G [3] explored the use of a generative adversarial network to generate realistic fake data using a smaller amount of real-world data. This was very effective and produced models with equivalent performance to those trained on the real data. However, it is dependent on a small amount of real data and is unsuitable for instances where no real-world data is available. In these cases, another method is needed.

In a paper by 6 authors [4], fake data was generated for testing related to Cyber-physical systems which did not rely on any real data. However, it was very specific to the problem and the data produced was of a form that is not relevant to most machine learning techniques: four state matrices were generated by their technique. This was successful in creating high quality data for the purpose of testing Cyber-physical systems but does not generalise well.

So there has already been success in developing many techniques to produce high quality fake data, however these techniques tend to address specific problems and do not generalise as well as the model of this paper attempts to.

Evolutionary algorithms have also been used to successfully generalise things that are typically produced for specific cases. In the paper by Peter A, Gregory S and Jordan P [5], evolutionary algorithms were used to produce recurrent neural networks which could automatically develop suitable features and hyperparameters for the chosen training data, instead of manually testing a set of cases.

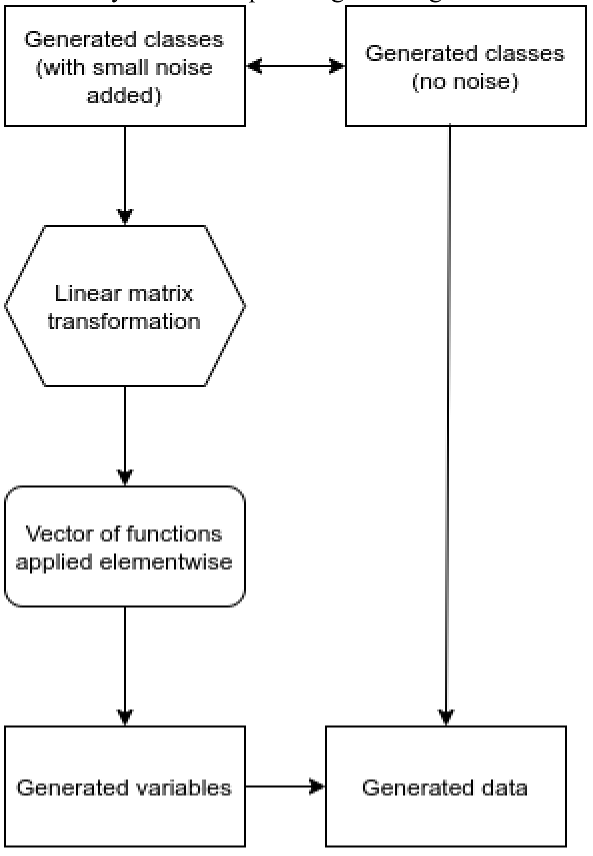
2 Method

A population of data generating organisms was first initialised. For an organism to produce data, the classes for the data are first generated. These are distributed evenly between classes, and there is no over-representation of a certain class.

Classes were coded using one-hot representation. The data was then duplicated, and small Gaussian noise was added to the ones in the duplicate, so there was some variation that could propagate through the transformations. The data then had a series of linear and functional transformations applied to it. In each linear transformation, a stored matrix was multiplied by the input vectors, producing new data vectors and allowing the length of the vectors to change. In each functional transformation, a vector of the same length as the data was stored where each element was either a function or 'None'. An additional vector was also stored of the same length containing weights, stored as floats. For each element in the function vector that was not 'None', the function was applied to the corresponding element in the input vector. The element times the corresponding weight element was then subtracted, and the weight element times the result of the applied function was added. Since the weight could fluctuate, this allowed organisms to evolve helpful functions to be more impactful and harmful functions to affect the output less.

Each organism had a constant and predefined number of linear transformations. These converged on the desired number of variables at a linear rate, however, were rounded down to the nearest integer when this would have caused it to attempt to map to a vector of non-integer length. Each transformation was applied sequentially until the outputted variables were reached. This was returned along with the original, untransformed classes as the generated data. During initialisation, a single functional layer was added as the first transformation. It was initialised with every functional vector element as 'None' and every weight as 0.1. The linear transformations were initialised as a matrix with elements sampled from a Gaussian probability function of mean 0 and variance 1, then normalised by dividing each element by the length of the input. Each type of transformation has been used in machine learning previously with good results. The linear transformations are a staple of many machine learning techniques including the most basic, fully connected neural networks. Functional transformations have also been used, for example in the paper by Zhang H, Lilong C and Zeungnam B [6], a smooth function vector was used to develop a fuzzy basis for nonlinear systems, where conventional bases are unsuitable. It also adds the function result to a linear result, as is achieved by this technique using the weight vector.

After initialising organisms, their data was tested in 'duels'. There were 2 types, asexual and sexual, that were alternately repeated. During asexual duels, 2 organisms were randomly selected. In addition, a fully connected classifier was created with a predetermined number of layers. Using the same approach as the linear transformations in the data creators, the neurons in each layer of the classifier decreased linearly and was rounded down to an integer if this would cause a decimal count. The activation function for each layer of neurons was chosen randomly from a set of activations with equal probability for each function. The activation functions that could be used were from the set containing the linear activation, sigmoid activation and hyperbolic tangent activation. The inconsistent use of activation functions was to improve generality and to avoid the generators learning to cater to a specific classifier model. The data produced by the creator was split into 80% training data and 20% test data. Keeping data for validation was not needed, as the parameters were being learned genetically and the main goal is to produce good data rather than a good model, which is achieved with a validation classifier later. The training data was then fed into the classifier, cycling through the datapoints for a predetermined number of iterations. Learning in the classifier was done through conventional back propagation. The same classifier was used between each organism in a duel, but the weights were re-initialised after each organism was tested so the ordering didn't bias the result.


Generated classes (with small noise added)
Generated classes (no noise)
Linear matrix transformation
Vector of functions applied elementwise
Generated variables
Generated data

Simplified procedure of data generation

After training, the trained classifier was used to predict the classes of the testing data and the accuracy was recorded. In each case, the class predicted by the classifier was the class corresponding to the output vector element with the highest value. The performance error of the organism was then given by the difference between this accuracy and the ideal accuracy, which is the goal of the data creators to converge on. The data creator with the highest error was killed and replaced with a child of the remaining data creator(s). After asexual duels, the child of the creator is created by copying the creator, and then applying the mutation function to its layers 3 times which gives them a chance to mutate based on the mutation chance of the layers. After sexual duels, the child of the remaining creators is formed by aligning the linear transforms of each creator, since they are guaranteed to have the same count and dimensions. Between linear transforms, each functional transform from both parents is transferred to the child with a 44% probability. This means on average the child will have 88% of the functional transforms of its parents, which puts an upper bound on the expected complexity of organisms after many time steps. The matrix of the linear transformations of the child is the elementwise average of the matrix of the linear transform for those dimensions from each parent. Each child also has a 20% chance of having a fresh functional transform layer is inserted, initialised with all functions as 'None' and weights as 0.1. This gives the system the chance to develop more complexity if it becomes useful to the goal. This sequence of an asexual duel, sexual duel and functional insertion was repeated a predetermined number of times.

Whenever a dataset was evaluated, the same approach was used. First, the data was split into 80% training data and 20% testing data. No hyperparameters were tuned by testing data, so no validation data was needed. A simple fully connected neural network was trained on the training set, using one hidden layer with 14 neurons and a sigmoid activation function. The output layer was linear. It was then used to classify the testing data, using the output with the highest value as the prediction. The accuracy of this network on the testing data was recorded. All datasets produced and tested had 23 inputs, 4 outputs and 3996 datapoints.

Each organism in the system was tested this way before and after evolution. This shows if the improvement in the data generation caused by the evolutionary algorithm or the initialised method of generating the data. 2 control datasets were also tested in this way. A dataset with all entries randomly generated was tested, to ensure the test was correctly implemented. Additionally, the SARS identification dataset was used as a comparison of data created by humans without using any special techniques to avoid bias or ensure good complexity.

These accuracies were recorded and are the results for this experiment.

The values of parameters used in testing are listed:

Number of data creating organisms	25
Variables	23
Classes	4
Layers in classifiers	4
Linear transformations in data creators	3
Datapoints generated by each data creator	3996
Epochs for each classifier training	6394
Optimal classification accuracy for data creators	0.85
Chance of element in function vector being replaced by a random, new one	0.05
Chance of element in weight vector mutating	0.3
Chance of linear transformation mutating	0.2
Normalised standard deviation of linear transform mutation	0.1
Classifier learning rate	0.01
Number of pairs of duels done in evolution	1874

3 Results and Discussion

The accuracies of each dataset tested in the experiment are listed:

Untrained organism 1	0.255
Untrained organism 2	0.24
Untrained organism 3	0.50375
Untrained organism 4	0.25625
Untrained organism 5	0.2625
Untrained organism 6	0.2575
Untrained organism 7	0.24
Untrained organism 8	0.24
Untrained organism 9	0.23375
Untrained organism 10	0.26125
Untrained organism 11	0.26625
Untrained organism 12	0.2375
Untrained organism 13	0.24625
Untrained organism 14	0.27125
Untrained organism 15	0.25375
Untrained organism 16	0.23875

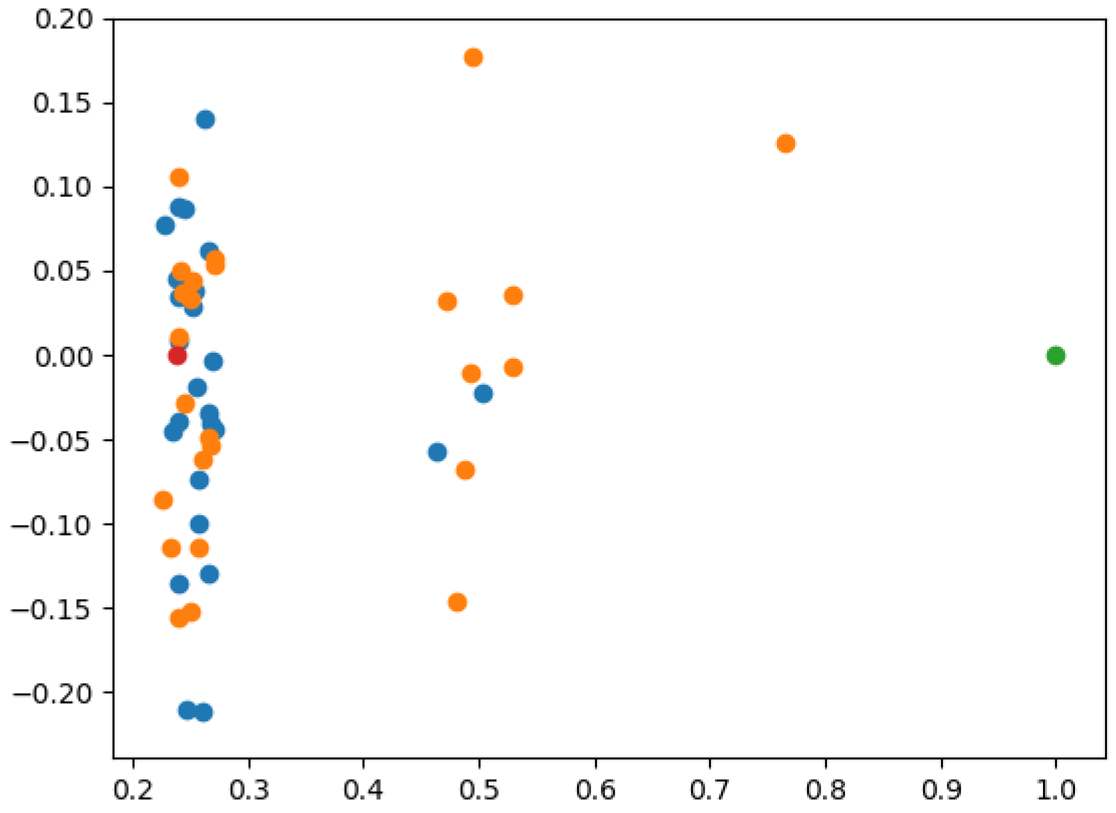
Untrained organism 17	0.2675
Untrained organism 18	0.25125
Untrained organism 19	0.2275
Untrained organism 20	0.46375
Untrained organism 21	0.245
Untrained organism 22	0.26625
Untrained organism 23	0.265
Untrained organism 24	0.26875
Untrained organism 25	0.24
SARS identification reference data	1.0
Pseudo randomly generated noise	0.2375
Trained organism 1	0.225
Trained organism 2	0.49375
Trained organism 3	0.27125
Trained organism 4	0.24
Trained organism 5	0.26
Trained organism 6	0.765
Trained organism 7	0.2425
Trained organism 8	0.52875
Trained organism 9	0.4725
Trained organism 10	0.2325
Trained organism 11	0.48125
Trained organism 12	0.52875
Trained organism 13	0.2675
Trained organism 14	0.25
Trained organism 15	0.27
Trained organism 16	0.2525
Trained organism 17	0.4875
Trained organism 18	0.23875
Trained organism 19	0.4925
Trained organism 20	0.25625
Trained organism 21	0.26625
Trained organism 22	0.25
Trained organism 23	0.24125
Trained organism 24	0.24
Trained organism 25	0.245

As the SARS data was too simple and linearly separable, the simple neural network achieved perfect accuracy on it as expected from the other paper that used it [2]. The random data had an accuracy very close to 0.25, which is expected given that there are 4 output classes.

In most cases, the data generated by the data creators before evolution had similar accuracies, with 2 of the 25 organisms as exceptions. These started with higher accuracy which shows that the initialisation can sometimes produce data with a solvable pattern by chance, but it is not consistent and likely to still be very flawed. This is expected as the initialisation method was coded by a human and would have intrinsic biases as a result that can reduce its quality.

The evolutionary algorithm was not effective in teaching the data creators to make useful or interesting data. Out of the 25 data creators after evolution was completed, only 8 of them showed results with enough deviation from the expected random distribution results to be considered to contain a measurable pattern when tested with the basic 3-layer neural net. Additionally, in these cases the neural net turned out to have solved for one or two classes very easily, while guessing the remaining classes with accuracy expected from random guessing. While the results were not good and the data produced by this algorithm should not be used in its goal of testing other machine learning techniques, the improvement in the number of classes that a pattern was found in was increased by the evolutionary learning.

This is easily visualised in plotting the resulting accuracies.



Accuracies recorded. The x-axis shows the accuracy, and the y-axis is random noise used to separate datapoints for visibility. Blue points are data creators before evolving, orange points are data creators after evolution, the green dot is the SARS data and the red dot is the random noise.

Training was also very slow and resulted in many of the parameters being much smaller than optimal. The results shown required 6 hours to compute, and the program scaled linearly with all the datapoints generated, evolution pairs and classifier epochs causing full scalings of the technique to increase with O(x^3) complexity.

Interestingly, the variables produced were often also of a very small size, and variables from the input with a value larger than 0.1 were very rare. This may have heavily impeded the learning and results by making the learning of the classifiers very slow. As the weights of the parameters were initialised to small values and 2 of the 3 activation functions used (tanh, simgoid) have a very small gradient for edge cases, the algorithm may have been very slow in converging on the solution. This also explains the property of the results in that classes were either solved for well or not at all, as the data may have been more easily classifiable with a large amount of time, but the classifier failed to converge before running out of epochs.

4 Conclusion

The experiment attempted to train a population of data creating algorithms evolutionarily with the goal of producing useful, highly generalised fake data for testing machine learning algorithms that cannot find suitable real data for their tests. Both generating fake data for testing models and using evolutionary algorithms to generalise conventionally case-specific procedures have been done before and showed great results. The experiments in this paper showed the technique was not effective in this given the hyperparameters used in testing, but there were some small improvements to the baseline which shows there might be potential for this approach to produce better results with enough modifications.

5 Future Work

There were many ways the experiment could be improved to have a better chance of producing real data. Firstly, the list of functions randomly available to the classifiers and the data creators was very short and limited the number of ways it can develop. The technique could be extended with many more functions to learn to improve the ways the data creating algorithms could develop. Many of the parameters were smaller than ideal, and the experiment could be re-run with more time and/or computational power to use better parameters, such as increased epochs for classifiers to have a better chance of converging. The number of organisms in the system could also have been increased, as it was easy for the best solution to be lost due to losing by chance to worse solutions a countable number of times. The number of epochs could also have been increased to allow the classifiers time to classify the data, allowing it to filter the better-quality data creators more reliably. Evolutionary algorithms are also not the only approach to this problem, and other techniques could be tested for their performance in generating generalised data that can be useful in machine learning experiments. Different underlying structures for the unlearnable elements of the data creators could be tested and may have better performance. Finally, parts of the algorithm could be optimised to improve their speed and solve the problem with larger hyperparameters in the same time.

References

1. Construction of Fuzzy Signature from Data: An Example of SARS Pre-clinical Diagnosis System http://users.cecs.anu.edu.au/~Tom.Gedeon/pdfs/Construction%20of%20Fuzzy%20Signature%20from%20Data%20An%20Example%20of%20SARS%20Pre-clinical%20Diagnosis%20System.pdf
2. Using functional analysis of input and hidden nodes of a neural network to adjust learning rate of neurons during training Support/ABCs2021_paper_142.pdf
3. Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks https://www.mdpi.com/1996-1073/13/1/130/htm
4. Generating Artificial Sensor Data for the Comparison of Unsupervised Machine Learning Methods https://www.mdpi.com/1424-8220/21/7/2397/htm
5. An Evolutionary Algorithm that Constructs Recurrent Neural Networks https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=265960
6. A Fuzzy Basis Function Vector-Based Multivariable Adaptive Controller for Nonlinear Systhttps://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=826963