Norbert Váradi (s202778)

Adjusting deep autoencoders using genetic algorithm

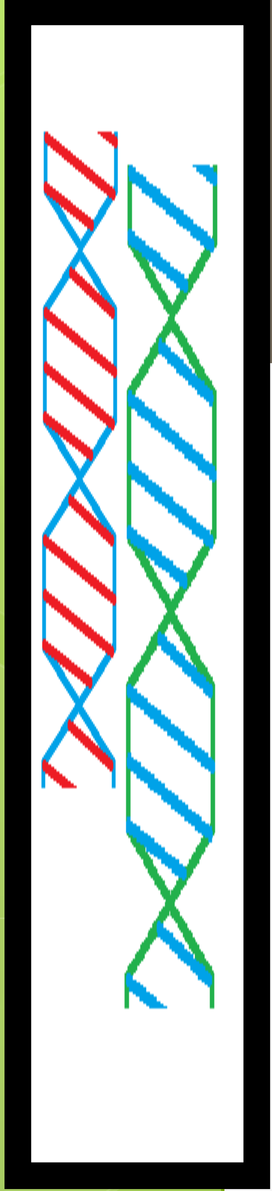
Can the model parameters of a convolutional autoencoder be optimized without backpropagation, purely using an evolutionary approach? The short answer is yes. But things are not so simple.

THE FIRST STEPS...

A simple fully connected NN with one hidden layer was created to prepare the ground for the research question. The system was assigned random model parameters which were adjusted to map pairs of arbitrary 2D vectors to each other. So there were three input-output pairs of two-dimensional vectors. The system successfully learned to map them to each other using techniques combining mutation and crossover.

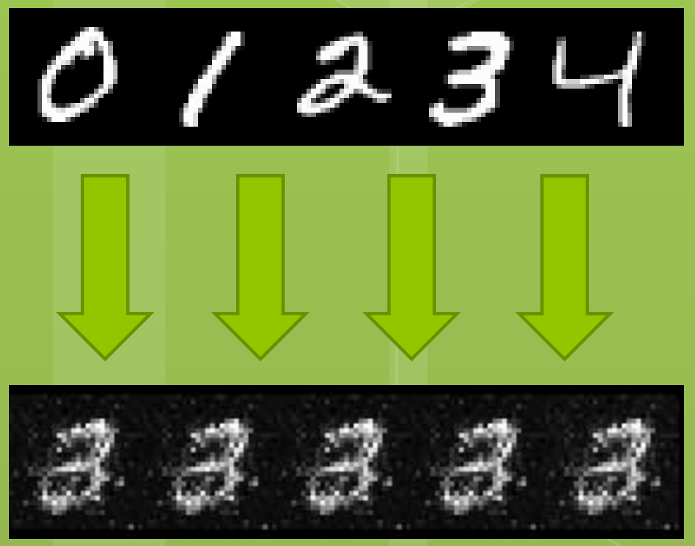
But how does it work?

According to ScienceDirect [1], there are 3 main processes involved in evolutionary algorithms:
Crossover: Swaping parts of the solution with another in chromosomes or solution representations. The main role is to provide mixing of the solutions and convergence in a subspace.
Mutation: The change of parts of one solution randomly, which increases the diversity of the population and provides a mechanism for escaping from a local optimum.
***Selection** of the fittest, or elitism*. The use of the solutions with high fitness to pass on to next generations, which is often carried out in terms of some form of selection of the best solutions.
Source [1]: https://www.sciencedirect.com/topics/computer-science/cross-over-mutation)



Bigger data: time vs. accuracy

Targeting the model parameters to be adjusted with mutation (with or without crossover) becomes a complex task for bigger data. The MNIST dataset was used [2], which contains samples of 784 bytes each. At every new generation of layout for the model parameters, when the best layout was chosen, different mutation ratios gave different results.
Source [2]: http://yann.lecun.com/exdb/mnist/
For every offspring, if only a single gene (weight) was chosen to be mutated, for the learning process it took minutes to distinguish between two shapes which were also kept constant.A high number of weights to be adjusted made the signal converge towards ether the average of the training set data or one of the samples fed into the neural network giving low accuracy. The most optimal number of mutations is one challenge to make the system work efficiently and effectively.



Convolutional neural networks

It was assumed that parameter sharing will speed up the learning process and through higher flexibility, more accurate results can be obtained. However, the system was still very likely to converge towards a learned average. For static samples, far better results were obtained with a simple fully connected neural network with a hidden layer having 2 neurons and faster learning was achieved. Due to the speed of learning with a CNN, the program was not tried for big data sets and therefore the robustness of the approach is not yet investigated for variational data and data generation.

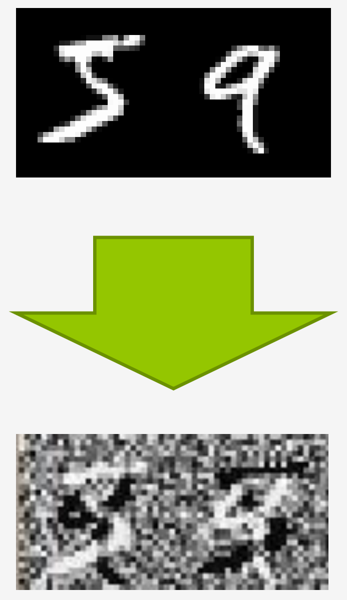
With high mutation ratio the system was shown to learn quickly, though the result was not satisfying.

Which samples to optimize the system to?
Since the samples were kept constant at each iteration, the system is optimized for a small subset of the entire dataset. Learning with new data at every generation was too slow to observe meaningful changes in the output.

Why was C++ used for the implementation?
Memory management was an important aspect of the implementation for which pointers and other low-level data types give a very clear understanding to write a NN library and an evolutionary algorithm from scratch. For example, mutation involves randomly selecting a number of weights from multiple tensors to which a list of pointers were necessary.

Other ways to speed up learning

In order to tackle the convergence problem, it was assumed that for each output sample in an iteration, the higher the error between the output and the average of the input for the other samples is, the further can the error be reduced to a minimum and hence the fitness function can be increased for the given offspring. Though the learning was very quick in such a way, and for two samples the approach works very well, the results are still highly objectionable in the sense of similarity to the original samples. So far, where the result was satisfying enough, the fitness function was kept to measure the absolute difference between only the input sample and their corresponding output pixels.



CONCLUSION

It can be stated that it is possible to train an autoencoder for image recognition for a small subset of samples under a relatively short timespan and the signal convergence can be avoided by decreasing the mutaiton ratio, which, however, might slow down the learning. The right balance has to be found for such a task. In such a way, it was observed that the number of model parameters selected for mutation behave similarly to learning rate in classic neural networks. Due to performance issues, robustness was not investigated on the bigger MNIST data set.