TRAINING DEEP AUTOENCODERS WITH GENETIC ALGORITHMS

Norbert Váradi

DTU 2021

ABSTRACT

Backpropagation is a well-known technique for adjusting the model parameters of a neural network. It involves calculations based on differential calculus with values depending on the gradients of the activation functions, the output of the artificial neurons, and a learning rate. One less known but more intuitive approach to the model parameter adjustment is genetic algorithms. I have done an investigation of this approach to discover that genetic algorithms in their core essence are not flexible enough to optimize a deep autoencoder network to big data sets such as MNIST, but might be able to reconstruct a small number of samples from 2D latent representations in reasonable time given that the training hyperparameters are set to optimum values.

1. INTRODUCTION

This paper demonstrates an evolutionary approach to the optimization of a deep autoencoder network, presenting a wide variety of scenarios involving different training hyperparameter settings, and discussing their limitations in terms of performance and accuracy when training the system for a small number of image samples.

2. THEORY

2.1. Autoencoders and VAEs

The general idea of autoencoders consists of setting an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimizational process. As data is fed to the autoencoder architecture, the output is compared with the initial data and the model parameters are updated given the calculated error (most commonly using backpropagation.) Training the autoencoder generates latent space representations of the input data through dimensionality reduction. [1]

When building an autoencoder, several basis can be chosen to describe the same optimal subspace and several different encoder/decoder pairs can give the optimal reconstruction error. There are no orthogonality constraints in the architecture of such a neural network. [1]

Even in case of no information loss, the autoencoder might show severe overfitting, meaning that some points of the latent space might give meaningless content once decoded. This is because the autoencoder is solely trained to encode and decode with as few loss as possible, no matter how the latent space is organized. According to Joseph Rocca, a Mathematics instructor at UTC, a variational autoencoder (VAE) is defined as an autoencoder whose training is regularised to avoid overfitting and ensure good properties of the latent space enabling the generative process. Training of a VAE is performed in a way that the input is encoded as a distribution in the latent space, from which a single point is selected to be fed forward, while in standard autoencoder there is no variance applied to the calculated data point. [1]

2.2. Convolution and transposed convolution

Convolutional neural networks are frequently used in computer vision problems, where features and patterns are extracted from an image. The goal of a CNN is to transform the input image into concise abstract representations of the original input. Each convolutional layer tries to find more complex patterns from the information in the previous layer. [2]

This suits well with the idea of the encoder part of a VAE or any autoencoder and intuitively, a CNN will provide a representation of the samples based on whether a pattern is present in the sample image rather than the overall pixel data that might be very different depending on the sample.

Performing the convolution involves a kernel that is a matrix of arbitrary size, that iterates through the previous layer's output to generate a new output. [2] In such a way, not every output pixel is connected to that of the previous layer and hence it is assumed that the training process can be improved both in terms of efficiency and effectiveness.

The rectified linear activation function (ReLU) can be applied to all pixels of the resulting feature maps to introduce a simple but effective nonlinearity to the system. [3]

One problem with the output feature maps created by convolution is that they are sensitive to the location of the features in the input. One approach to address this sensitivity is to downsample the feature maps. This is performed by a pooling layer, where most commonly a maximum value is chosen from patches of 2x2 pixels. This is called maximum pooling

or max pooling. [3]

For the decoding process of the autoencoder, a transposed convolutional layer performs upsampling of the image data by reversing the process of convolution. Transposed convolutional layers can be used in autoencoders and GANs, or in any network for the purpose of image reconstruction. As a result of such operation, it can be achieved that the output dimensions are higher than that of the input. [2] Furthermore, the opposite of the pooling process is called unpooling, where the compressed image is upsampled to higher resolution. [4]

2.3. Genetic algorithms

Genetic algorithms is an unsupervised approach to optimize agents in certain environments, and they can be used to adjust the model parameters of neural networks as well. They are based on the assumption that by applying crossover and mutation, better neural networks can be created. [5]

As the first step, random values are assigned to the model parameters of the neural networks that are the agents of the initial population. At every new generation, a series of tests are carried out to determine the fitness of each agent, following which an arbitrary number of the fittest individuals are selected to be available to reproduce, from which two agents are chosen. Their model parameters are crossed over and undergo mutation to produce new data. [5]

This process slowly optimizes the accuracy of an agent, as the agents slowly adapt to the environment. Though the process is not computationally heavy, while still adaptable and understandable, it reportedly takes a long period of time to work. [5]

According to Marco Castellani, a researcher at the University of Birmingham, many practitioners do not use crossover to evolve artificial neural networks, and rely instead only on mutation. Castellani assumes that the purpose of crossover is swapping meaningful functional elements and assembling good solutions of them. For instance, if a genetic algorithm is used to optimize a car prototype, genes coding for the width of the tyres or the shape of the body can be crossed over but the distributed nature of knowledge base in artificial neural networks makes it difficult to pinpoint these units. Castellani himself declared that he never used crossover when optimizing neural networks and based his strategy on a research by Thierens et al. [6]

2.4. Fitness calculation

According to Qureshi et al., there are multiple ways to calculate a fitness score for an agent for image reconstruction algorithms. They have performed a study involving binary genetic algorithm and compared different error calculations for the fitness function. They have found that root mean square error and mean absolute error have been found superior compared with the remaining fitness functions as far as better convergence and higher sensitivity are concerned. [7]

2.5. Encoding

There are various ways to encode the model parameters of an agent as a genotype. In direct encoding methods, a direct genotype-phenotype mapping is used to describe a neural network. It means that all parameters of the network are clearly understandable from the genes without any repeated process of transcription or growing. When a network designer does not design the network directly and he decides to use evolutionary heuristics, he is facing the problem of selecting the right method of encoding. Other categories of encodings are parametric and indirect encoding. Accoring to Fekiač et al., Cellular encoding and cellular graph grammar-based encoding are proven to be the most flexible, hence their implementation is considered. [8]

3. DESIGN AND IMPLEMENTATION

The program was implemented in C++ using a framework specifically designed for this project. The choice of the programming language is due to the fact that originally it was assumed that there will be a significant emphasis on memory addressing when it comes to encoding the model parameters, as some quite advanced encoding methods were described to be the most flexible by Fekiač et al. [8]). It only became clear later that the scope of this research and time pressure do not allow such complexity, therefore direct encoding was implemented, but the program was still written in C++.

The other reason for the choice of language is that personal observations suggest that other tools such as Python is much slower than C++ when it comes to tasks involving neural networks so that it is assumed that the C++ program will learn at a higher pace.

The MNIST dataset was used ([9]) to feed samples through a deep autoencoder. In order to efficiently classify and reconstruct images, the autoencoder incorporates convolution and transpose convolution layers as well.

The SDL graphics library was used to display the images reconstructed by the neural networks both to provide visual feedback at different stages of the development and to provide illustrations for this paper. SDL is a cross-platform multimedia tool. [10]

3.1. Building the neural networks

A simple framework was implemented that can be used to create neural network structures of any size, with randomly initialized weights, where the type of each layer can be specified without manually adjusting their parameters such as the total number of weights or explicitly addressing the location of the input it takes. This kind of flexibility was proven very important for testing different neural network architectures so that large pieces of code do not have to be rewritten every time when a new layout is being tested. The reason for the custom

framework is the apparent lack of popular C++ libraries available that support features to create autoencoders with both convolution operations and genetic optimization.

Using a few lines of code, any layout of neural network can be created with all the vital features mentioned in the Theory section. The following code creates an autoencoder with randomly initialized model parameters, that is able to generate 2D latent space representations and complete reconstructions of different MNIST data samples. The number of layers does not have to be defined in advance as the layers are part of a linked list (for the complete implementation see "neural.h" in the Git repository.) When calling the *list_params* function, an array containing the direct encoding of all the model parameters is created (of all weights and biases respectively), from which later the system randomly picks at least one parameter that will undergo mutation for every new offspring generated.

```
1  nn::NN nn;
2  nn.setInputLayer(28, 28);
3  nn.addConvLayer(5, 5, 1);
4  nn.addMaxPoolLayer();
5  nn.addConvLayer(5, 5, 1);
6  nn.addMaxPoolLayer();
7  nn.addSigmLayer(5);
8  nn.addSigmLayer(2);
9  nn.addSigmLayer(5);
10 nn.addSigmLayer(160);
11 nn.addReshapeLayer(4, 4, 10);
12 nn.addUnpoolLayer();
13 nn.addTrConvLayer(5, 5, 1);
14 nn.addUnpoolLayer();
15 nn.addTrConvLayer(5, 5, 10);
16 nn.list_params();
```

When adding a convolution layer (*NN::addConvLayer()*), given the kernel size and the number of channels the network diverges to, the system automatically calculates the output dimensions and the number of the resulting feature maps.

When adding a max pooling layer (*NN::addMaxPoolLayer()*), the resolution is reduced to half the input dimensions ([3]) and evidently, the number of channels is kept constant.

By introducing an additional nonlinearity by the insertion of a sigmoid layer (with an arbitrary number of neurons, line 7) at the end of the convolutions before the bottleneck layer, a flexible placement of the data points within the latent subspace can be possible.

By performing a series of tests and visualizing the results using SDL, it was found out that a single channel of convolution is enough to generate unique representations of images showing one of the 10 different digits from the MNIST library. Due to the expected performance of the system, the tests will only attempt to compress and reconstruct one sample of each of the ten digits to optimize the system.[9]

As the article by Joseph Rocca ([1]) explicitly stated that

Fig. 1. Ten distinct feature maps of 4x4 pixels produced by a series of convolution and maximum pooling.

there are no constraints on the latent subspace representation and the orthogonality of the network, the system will work with a simple two-dimensional latent space and will incorporate multiple channels on the decoder side, while the rest of the model is symmetric. Many different channels on the decoder side is important so that the model can be optimized to reconstruct each of the digits from a wide variety of "brushes", aka. transposed convolution kernels. To maintain a balanced architecture, this number was defined to be the same as the number of samples to be used.

It is further important to note that as the sigmoid layer merges all the incoming input into a single array of neurons, in order for the decoder side to restore the original 4x4 input format, but this time with 10 channels, a "reshape layer" has to be incorporated into the network before performing the upsampling process. This simply packs the data of the most recent output to a new format with a specific width, height and number of channels.

3.2. Training the network

Based on the article by Victor Sim ([5]), pieces of information to be stored by an agent are the unique model parameters and a variable for the score to be calculated for the agent. Based on the research by Qureshi et al. ([7]) mean absolute error was one of the two fitness function that outperformed all other ones, and since its calculation is the simplest, it will be used to calculate the score for each of the agents when training for a given number of samples. Evidently, this is performed by taking the absolute difference between the output pixel values and their corresponding input pixel values and adding them all together. The lower is the error, the higher the accuracy.

As it was stated by Marco Castellani ([6]), there is no need for crossover when performing the optimization whatsoever. This leads to a single parent agent to be selected as the best candidate for reproduction, whose data will be copied to all the other agents before performing the mutations on them.

The mutation is performed as a random adjustment to a number of randomly selected weights. In cases where only a slight fine-tuning is necessary on the model to achieve the most satisfying result, the adjustment takes place relative to the current values of the parameters, which means adding randomly generated numbers between -1.0 and 1.0 to the model parameters instead of applying the new values on them directly.

4. RESULTS

During the first tests of the network it was observed that the signal bursts out and explodes when the target is normalized to values between 0.0 and 1.0 (following the range of the Sigmoid activation function). Therefore, a range whose corresponding x-values can more easily be acheived by summing up the net input was chosen, namely a range with a lower bound of 0.1 and an upper bound of 0.9.

When training the model for one sample, the system converges very quickly in a few seconds while achieving a fairly accurate result, showing especially high performance under conditions where applying mutation to about 10 parameters for a population of about 10 agents. For higher population size, the training becomes slower but does not show apparent increase in the quality of the output. The same goes for selecting a small number of weights to undergo mutation. When the population is small (2-5 agents), the difference between the error on the best and the worst solution is evidently smaller, hence the model can easily either reach an evolutionary dead end from where it is unlikely to come up with a satisfying output or again takes a long time to evolve despite many generations of offsprings, and the same goes for the situation of having a large number of model parameters to be selected for mutation, where the likeliness of achieving mutations of high advantage is reduced.

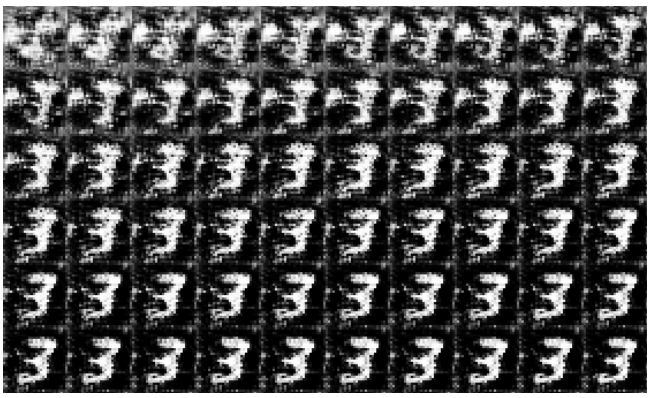
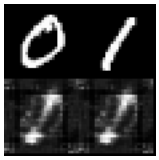


Fig. 2. Stages of evolution. Every new frame in the image above is a result of 5 generations of mutations since the previous output generated by the autoencoder. Note the growth/shrinking of certain areas indicating changes in the generated feature maps.

When training for more than one sample, it was observed that a very small mutation rate has to be set in order to achieve a satisfying result, otherwise the system converges in a way that the output will be the same for all inputs, looking like as if the average of all the input images was taken. A satisfying result can be achieved by setting the mutation rate to 1 and increasing the size of the population. In such a way, the training becomes slow, but there will be a higher likeliness for a mutation to occur that makes the network generate differentiated outputs that slightly correspond with the input data. The following images show the difference between training rate with a mutation rate of 1 and a mutation rate of 10, respectively. In the following images, the first line of digits show the target output and the second line shows the actual output.

Mutation of 1 weight



Mutation of 10 weights

Fig. 3. Different results after training for 1000 generations

According to Paulo Gaspar, a researcher at the University of Aveiro, too small mutation rate does not lead to good results either. [11] Hence adjusting the mutation ratio heavily depends on the number of samples to train the model for. The following image shows the result of a 12-hour training session with 10 different digits where the mutation rate was set to 1.

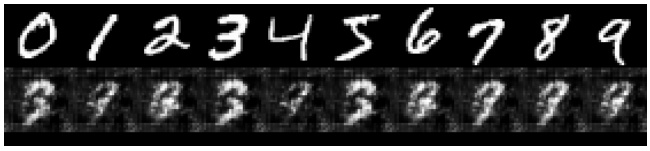


Fig. 4. Even though the system was trained for about 12 hours with a population size of 20, the end results are only slightly differentiated and hardly show any resemblance to the input data.

5. EVALUATION AND CONCLUSION

Due to the high reconstruction loss and time pressure, the presence of interpretable and exploitable structures in the latent space (regularity) ([1]) was not investigated. Hence it is not known whether the training method has implicitly regularized the latent space or not.

Investigating this feature does not even make too much sense given that the system is very sensitive to the mutation rate and that a large population size is needed to achieve satisfying results for a very small amount of data compared with the size of the actual MNIST data set.

The results not being satisfying enough can be due to a technique of applying mutations that do not include heuristics that promote genetic diversity.

As for future prospects, using advanced encoding techniques and training methods is proposed as a potential opportunity to increase the robustness of the system to be capable for faster and flexible training and at the same time to desensitize the neural network to the mutation rate and other training parameters.

6. REFERENCES

[1] Joseph Rocca, "Understanding variational autoencoders (vaes)," *Towards Data Science*, July 2020.

[2] Mars Xiang, "Convolutions: Transposed and deconvolution," https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6, July 2020.

[3] Jason Brownlee, "A gentle introduction to pooling layers for convolutional neural networks," https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/, April 2019.

[4] Harshit Kumar, "Autoencoder: Downsampling and upsampling," https://kharshit.github.io/blog/2019/02/15/autoencoder-downsampling-and-upsampling, February 2019.

[5] Victor Sim, "Using genetic algorithms to train neural networks," https://towardsdatascience.com/using-genetic-algorithms-to-train-neural-networks-b5ffe0d51321, September 2020.

[6] Marco Castellani, "What is the most suitable crossover method to train ann using ga?," October 2012.

[7] Shahzad Ahmad Qureshi and Sikander M. Mirza, "Fitness function evaluation for image reconstruction using binary genetic algorithm for parallel ray transmission," January 2007.

[8] Jozef Fekiač, Ivan Zelinka, and Juan C. Burguillo, "A review of methods for encoding neural network topologies in evolutionary computation," *25th EUROPEAN Conference on Modelling and Simulation*, June 2011.

[9] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges, "The mnist database," http://yann.lecun.com/exdb/mnist/, May 2013.

[10] Anonymous, "About sdl," https://www.libsdl.org/.

[11] Paulo Gaspar, "Why is the mutation rate in genetic algorithms very small?," January 2013.