Welcoming the Era of Deep Neuroevolution

Handout for the SZ1 class, Matyáš Skalický, 14.10.2018

Original article: https://eng.uber.com/deep-neuroevolution/

Introduction

On the upcoming seminar, I will talk about the idea of so called neuroevolution. The technique that **uses evolution to train and optimize neural networks**. This newly rediscovered approach has shown its capability to be used in the field of deep learning, where neural networks of many hidden layers and millions of connections need to be trained and optimized efficiently. This technique was found to be efficient for reinforcement learning (RL) problems. The original Uber AI labs article sums up this technique based on 5 papers describing the topic of neuroevolution.

Common techniques to train deep neural networks

gradient *– a vector (direction to move) to minimize a given loss function*

Gradient Descend (GD)

GD is an iterative **algorithm for finding a minimum of a function**. It works by tweaking function’s parameters (weights of a NN) iteratively to find a local minimum. You can imagine a blind climber going up a mountain that always chooses the next step around (vector - gradient) that rewards him with the most elevation gain.

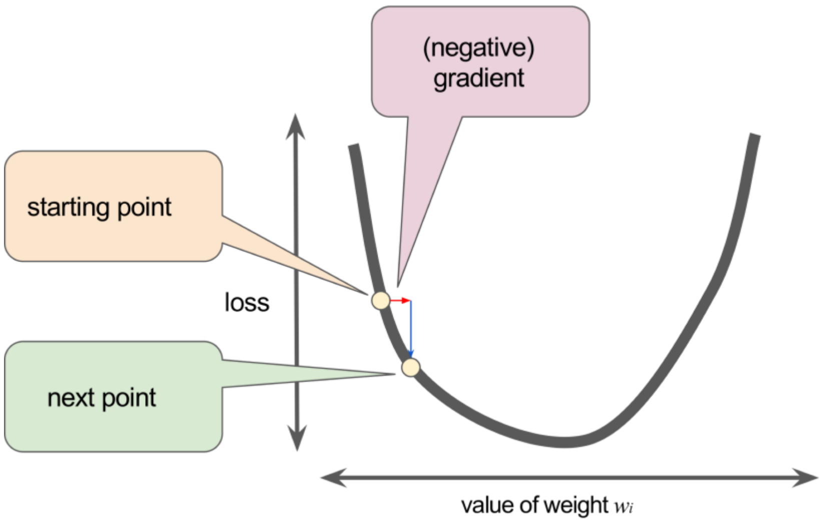


Figure 1 - the weight is adjusted towards the negative gradient

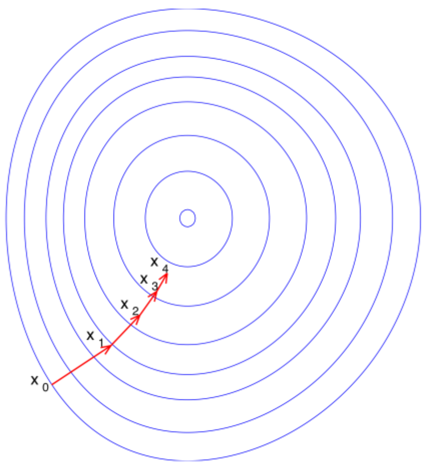


Figure 2 – gradient visualization

Stochastic Gradient Descend (SGD)

GD can be slow to train deep NNs on huge datasets. The dataset might not even fit into the memory! SGD in contrast, uses only a small **randomly selected data sample to obtain the gradient**. This gradient is only an approximation, but it is much **faster to compute** so it can be repeated many more times. This method scales well and is used widely in the field of deep learning.

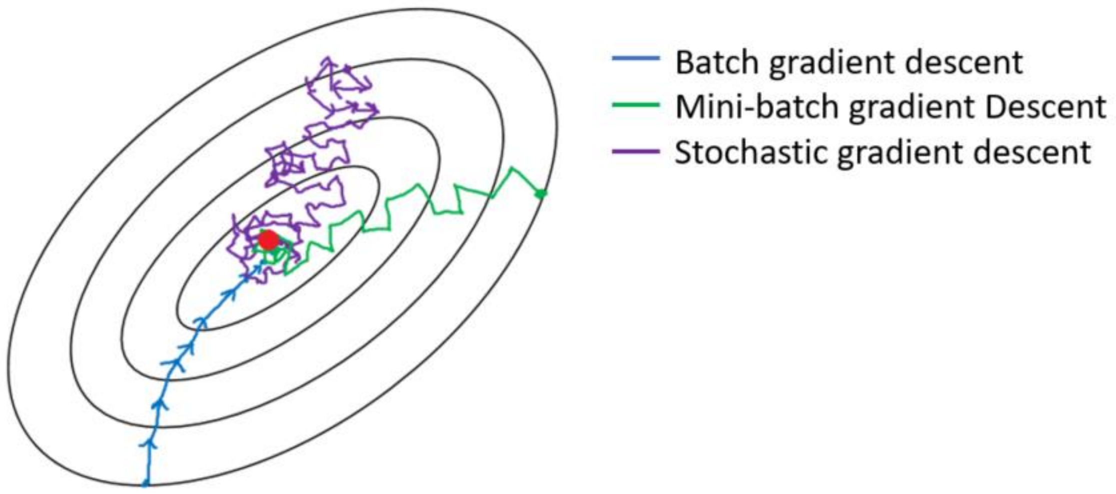


Figure 3 – batch (traditional) GD always gets the most precise gradient, however it is computationally hard to calculate.

Neuroevolution

Neuroevolution is an example of **a gradient-less training method** that can be used to train the neural networks especially for the reinforcement learning (RL) problems. As the article explains, the researchers were surprised to discover, that an **extremely simple genetic algorithm** (GA) can outperform the modern RL algorithms by playing Atari games from pixels while also being **faster due to better parallelization**. The neuroevolution is **less likely to get stuck in local minima** too.

The Uber AI Labs article sums up 5 papers describing the use of neuroevolution – the technique to train NNs through evolution.

I have chosen to dig deeper into the "[Genetic algorithms as a competitive alternative for training deep neural networks]" paper. The following chapter describes the main ideas from the paper:

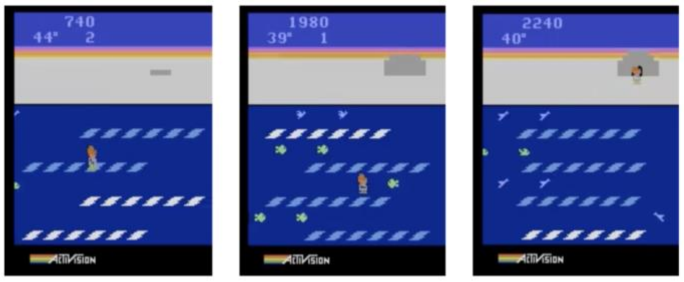


Figure 4 – Frostbite was used as one of the benchmark games

Genetic algorithms as a competitive alternative for training deep neural networks

Paper by Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman Kenneth O. Stanley and Jeff Clune from Uber AI labs

https://arxiv.org/abs/1712.06567

Deep genetic algorithm (GA) described in the paper successfully evolves networks with over four million parameters, the largest neural network ever evolved with a traditional evolutionary algorithm. Atari games played from the pixel inputs are used as a benchmark.

Genetic algorithm (GA)

Genetic algorithm evolves a population P of N individuals (neural network parameter vectors θ, often called genotypes):

- At every generation, each θ_i is evaluated, producing a ***fitness* score $\boldsymbol{F(\theta_i)}$.**

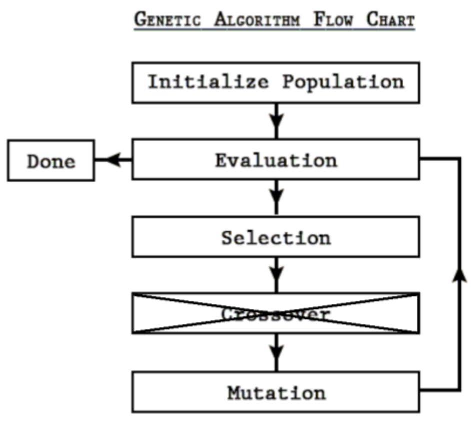
- GA performs ***truncation selection***, where the top T individuals become the parents of the next generation.

To produce the next generation, the following process is repeated $N - 1$ times:

- A **parent is selected uniformly at random with replacement** and is **mutated by applying additive Gaussian noise** to the parameter vector: $\theta' = \theta + \sigma\varepsilon$ where $\varepsilon \sim N(0, 1)$.

- The Nth individual is an unmodified copy of the best individual from the previous generation, a technique called *elitism*.

The process repeats for G generations or until some other stopping criterion is met.



Genetic algorithms often involve *crossover* (combining parameters from multiple parents to produce an offspring), but for simplicity it is not used.

Figure 5 – typical genetic algorithm flowchart. The GA proposed in the paper does not use crossover for simplicity reasons.

A new approach to store the parameters (genomes)

The paper proposes an **innovative new approach to store the parameters (genomes)**.

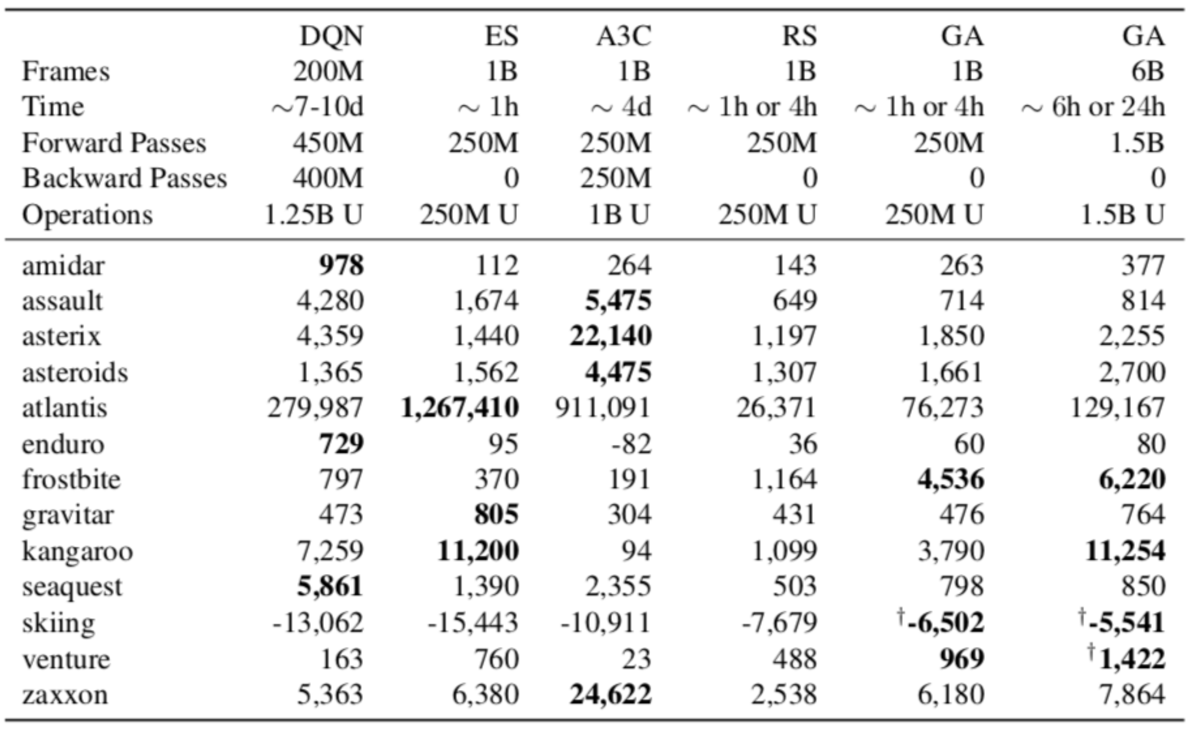
The traditional approach is to store each individual θ as a parameter vector. However, this approach scales poorly. Especially in large neural networks with large populations.

Instead, the paper proposes to store the parameters as an **initialization seed** plus the **list of random seeds of mutations that produced each θ**. This way, θ can be easily reconstructed while being stored in a very memory-efficient way.

This innovation allowed the researchers to create an **effective distributed GA**.

Benchmarking the algorithm – Atari games

GA was shown to perform better than ES, A3C, and DQN on 6 games each out of 13.



	DQN	ES	A3C	RS	GA	GA
Frames	200M	1B	1B	1B	1B	6B
Time	∼7-10d	∼ 1h	∼ 4d	∼ 1h or 4h	∼ 1h or 4h	∼ 6h or 24h
Forward Passes	450M	250M	250M	250M	250M	1.5B
Backward Passes	400M	0	250M	0	0	0
Operations	1.25B U	250M U	1B U	250M U	250M U	1.5B U
amidar	**978**	112	264	143	263	377
assault	4,280	1,674	**5,475**	649	714	814
asterix	4,359	1,440	**22,140**	1,197	1,850	2,255
asteroids	1,365	1,562	**4,475**	1,307	1,661	2,700
atlantis	279,987	**1,267,410**	911,091	26,371	76,273	129,167
enduro	**729**	95	-82	36	60	80
frostbite	797	370	191	1,164	**4,536**	**6,220**
gravitar	473	**805**	304	431	476	764
kangaroo	7,259	**11,200**	94	1,099	3,790	**11,254**
seaquest	**5,861**	1,390	2,355	503	798	850
skiing	-13,062	-15,443	-10,911	-7,679	†**-6,502**	†**-5,541**
venture	163	760	23	488	**969**	†**1,422**
zaxxon	5,363	6,380	**24,622**	2,538	6,180	7,864

Figure 6 – game scores

Conclusion

The paper shows, that even a simple, traditional GA performs well on hard RL problems. It comes up with a new innovative way to store the parameters (genomes) of the each individual to allow for effective parallelization. It is suggested that in some cases, the traditional gradient is not the best way to optimize performance.

Interestingly it is also shown, that on some games the random search substantially outperforms the traditional RL algorithms although it never outperforms the GA.