Playing Atari with Six Neurons

Abi Raveenthiran s4010132
Kyriakos Antoniou s5715881

March 2024

1 Introduction

In this project we aim to reproduce the results of the paper "Playing Atari with six Neurons" by Cuccu, Togelius, and Cudré-Mauroux (2018). In this paper, the authors test a model defined by a single layer fully connected Recurrent Neural Network (RNN), with 6-18 neurons, on several Atari games, and compare them to other state-of-the-art works.
The authors reason that neural networks unnecessarily have the responsibility of learning the representation as well as the policy. Instead the authors suggest to separate these components by having a smaller network that is dedicated to policy learning. The authors introduce two novel algorithms to achieve this, namely, Increasing Dictionary Vector Quantization (IDVQ) and Direct Residual Sparse Coding (DRSC). In this project we will reimplement these algorithms as well as the evolutionary algorithm named Exponential Natural Evolution Strategies (XNES) by Glasmachers, Schaul, Yi, Wierstra, and Schmidhuber (2010), which is used by Cuccu et al. to optimise the weights of the network.

2 Methods

In this section we will describe the algorithms that were used in the model of the paper by Cuccu et al. (2018).

2.1 Increasing Dictionary Vector Quantization

Increasing Dictionary Vector Quantization is a data compression method introduced by Cuccu et al. (2018) specifically designed for online learning. In contrast to standard Vector Quantization, IDVQ starts with an empty dictionary that does not have a fixed size. This dictionary does not need to be initialised and increases in size over training iterations.
The algorithm starts as follows, it takes an observation and encodes it with the DRSC algorithm. This returns a binary encoding of the centroids are

used/unused. Taking the sum of the used centroids (the centroids that are encoded with a 1) contains all the relevant information that is already contained within the current dictionary. By subtracting this information from the original image, a matrix is obtained where the positive numbers are information that is not yet covered by the centroids. All negative elements (artifacts) are already covered in the centroids, these are ignored by setting them all to 0. Lastly, check whether the sum of the remaining information is above the threshold δ. If it is above the threshold, append it to the dictionary.

Procedure 1 Increasing Dictionary Vector Quantization

```
Inputs:
  $\mathscr{X}$: training set, $X \in \mathscr{X}$
  $D$: current dictionary
  $\delta$: minimal aggregated residual for inclusion
Initialize:
  $D \leftarrow \varnothing$                                  ▷ dictionary initialized empty
  for $X$ in $\mathscr{X}$ do
      $\mathcal{P} \leftarrow X$                              ▷ residual information to encode
      $\vec{c} \leftarrow DRSC(X, \mathcal{D}, \epsilon, \Omega)$
      $\hat{\mathcal{P}} \leftarrow \vec{c}\mathcal{D}$
      $\mathscr{R} \leftarrow \mathcal{P} - \hat{\mathcal{P}}$
      $r_i \leftarrow \max(0, r_i), \forall_i \in \mathscr{R}$            ▷ remove artifacts
      if $\sum|\mathscr{R}| > \delta$ then
          $\mathcal{D} \ll \mathscr{R}$                            ▷ append $\mathscr{R}$ to $\mathcal{D}$
  return $\mathcal{D}$
```

2.2 Direct Residual Sparse Coding

Direct Residual Sparse Coding is an encoding method introduced by Cuccu et al. (2018); the author has used this algorithm as it has been shown by (Mairal, Bach, Ponce, et al., 2014) (Zhang, Xu, Yang, Li, & Zhang, 2015) that, in combination with IDVQ, it allows the algorithm to better focus on the important characteristics. It takes an observation and dictionary and returns a binary encoding of the centroids that are used/unused in the dictionary. The algorithm works as follows: firstly, the algorithm begins by learning a dictionary of the input data, representing the sample by a linear combination of independent properties (vectors). After the representation is constructed, a residual or reconstruction error is calculated. This error is minimized to improve the dictionary, allowing for a better representation and reconstruction of the initial data. The algorithm aims to reconstruct the data as well as possible while aiming to have the least number of variables representing a sample. The algorithm is used for both data compression and in our case, feature extraction.

Procedure 2 Direct Residual Sparse Coding

```
Inputs:
  $X$: vector to encode (observation)
  $D$: dictionary trained with IDVQ
  $\epsilon$ minimal aggregated residual loss
  $\Omega$: maximum nonzero elements in the code
Initialize:
  $\mathcal{P} \leftarrow X$                                ▷ residual information to encode
  $\vec{c} \leftarrow \vec{0}$                                ▷ output code
  $\omega \leftarrow 0$                                ▷ non-zero elements in the code
  while $\sum|\mathcal{P}| > \epsilon$ and $\omega < \Omega$ do
      $\mathcal{S} \leftarrow \mathrm{sim}(\mathcal{P}, \dot{i}), \forall d_i \in \mathcal{D}$
      msc $\leftarrow$ index of max $(\mathcal{S})$
      $\vec{c}_{\mathrm{msc}} \leftarrow 1$                                ▷ $\vec{c} = [\vec{c}_1 ... \vec{c}_n]$
      $\omega \leftarrow \omega + 1$
      $\mathcal{P} \leftarrow \mathcal{P} - d_{\mathrm{msc}}$                                ▷ $\mathcal{D} = [d_1 ... d_n]$
      $\rho_i \leftarrow \max(0, \rho_i), \forall \rho_i \in \mathcal{P}$
  return $\mathcal{D}$
```

2.3 Exponential Natural Evolution Strategies

Exponential Natural Evolution Strategies, introduced by Glasmachers et al. (2010), is an evolutionary algorithm stemming from the family of Natural Evolution Strategies (NES). Rather than maintaining a distribution over the population NES algorithms maintain a distribution over the parameter space. It makes use of natural gradient update method, this is done by taking the standard gradient and rescaling it based on the Fischer information matrix. The Fischer information matrix $\tilde{\nabla} = \mathbf{F}^{-1}\nabla_0 J(\theta)$ is used to measure how much information is contained within a observation.
The fitness function of NES algorithms is defined in 1. Here, θ are the given parameters, $\mathbf{z}$ is the given sample and $p(\mathbf{z}|\theta)$ is a conditional probability function.

$$J(\theta) = \mathbb{E}_\theta[f(\mathbf{z}] = \int f(\mathbf{z})p(\mathbf{z}|\theta)d\mathbf{z} \tag{1}$$

In XNES the distribution over the parameter space is a multivariate Gaussian distribution. The parameters in XNES are $\theta = (\mu, \sigma)$ and are updated as follows:

$$\begin{aligned} \mu &\leftarrow \mu + \eta_\mu + \sum_{k=1}^{\lambda} u_k \mathbf{z}_k \\ A &\leftarrow A \exp\left(\frac{\eta_A}{2} \sum_{k=1}^{\lambda} u_k (\mathbf{z}_k \mathbf{z}_k^{\mathsf{T}} - I)\right) \end{aligned} \tag{2}$$

here σ is the Cholesky decomposition matrix of $A^{\mathsf{T}}A$. A is an upper triangular matrix and $A^\intercal$ is the conjugate transpose of A, η_μ and η_A are the learning rates, λ is the population size and u_k are the fitness shaping utilities.

2.4 System process

In this section we will provide brief explanation of how the system of (Cuccu et al., 2018) works.
The first part of the system is the compressor, the compressor consists of two algorithms, IDVQ (1) and DRSC (2). IDVQ trains a dictionary with the help of the encodings of observations from DRSC. The compressed observation is then sent to the controller, which is a single-layer fully connected RNN. The number of inputs that the input layer receives is equal to the size of the dictionary coming from the compressor. The received input is passed through weighted connections. With the weights for these connections being optimised by the XNES algorithm described in 2.3. The neurons of the output layer are set to be equal to the action space of the environment. For the Atari games tested in the paper this meant that it would range from 6 to 18 neurons in the output layer. The network did not make use of any hidden layers.

3 Experimental setup

In this section a detailed explanation of how the approach to reproducing the experiment was conducted.

3.1 Experimental path

The coding language of choice was selected to be Python due to the available choices of libraries that are useful for implementing a variety of machine learning tasks. Additionally, Python was chosen as the author of the original experiment used the Atari back-bone library gym to run the experiments on. Unfortunately, gym is no longer used, and the functionality of the library has been moved to a new library called a gymnasium; for replication of the experiment, the function of the libraries was nearly identical. Furthermore, Python was also the common personal language of choice.
Two major machine learning language libraries are found in Python, Keras / Tensorflow and Pytorch where either did not have the required functions in order to implement XNES, more specifically missing the XNES optimizer in the first place, not having the ability to change the input layer size as the algorithm is updated and requiring a complicated weight manipulation to implement XNES. Due to the above reasons, libraries where they had built-in ones optimizer function were found, and a single library in Python was found; this library is Pints. Although Pints had an XNES implementation, the required parameters needed to use a Pints error class to use the optimizer making us unable to use the

gymnasium rewards parameter to improve the model with out forcing us to rewrite the gymnasium games as a Pints error class. This has forced us into a corner where we had to construct XNES, the compressor methods, IDVQ and DRSC as well as at least a basic neural network creator from scratch. Referencing Github code was found helping us create the algorithms themselves but in combining the algorithm's logical errors forced us to get an even deeper understanding of the fundamentals of the algorithms and what exactly each piece of code was doing and should have been modified in order to be used with the other algorithms, due to time limitations and the amount of material we need to look for we took a look at the ruby code, but as we were both unfamiliar with the language itself and its underlying functional programming concepts we were unable to understand the code without a good understanding of the basics of functional programming. At the end we managed to combine the IDVQ and DRSC functions to create a working compressor method but were unable to adapt it to work with gym, XNES and the basic neural network constructor. We have also managed to combine gym, XNES, and a constant input size Tensorflow neural network in order to show the working and learning capabilities of the method, which, even in the most basic of tasks, did not result in noticeable learning, this could also be because of performance issues that forced us to run the tests only couple of times on small parameter space of gymnasium input shape with six hidden layer neurons and gym output shape for the specific gymnasium environment (tested with lunar lander v2 and cart pole v1 with input and output shapes of 8,4 and 4,2 respectively while run until termination of up to 200 turns).

4 Conclusion

Through trying to replicate the experiment, we have managed to create working functions for IDVQ, DRSC, and XNES, as well as a basic bones neural network. All three functions of IDVQ, DRSC, and XNES seem to be working correctly for specific shapes of input data; combinations of methods such as the IDVQ and DRSC combined and gym with XNES and a constant TensorFlow neural network have also shown working results.
Although we have failed to combine all methods together with the environment to achieve replication results of the experiment.
There is a clear path to how more work into understanding and adapting the functions to combine them could lead to achieving results that could confirm or contradict the results of (Cuccu et al., 2018)

References

Cuccu, G., Togelius, J., & Cudré-Mauroux, P. (2018). Playing atari with six neurons. *arXiv preprint arXiv:1806.01363*.

Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., & Schmidhuber, J. (2010). Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on genetic and evolutionary computation* (pp. 393–400).

Mairal, J., Bach, F., Ponce, J., et al. (2014). Sparse modeling for image and vision processing. *Foundations and Trends® in Computer Graphics and Vision*, *8*(2-3), 85–283.

Zhang, Z., Xu, Y., Yang, J., Li, X., & Zhang, D. (2015). A survey of sparse representation: algorithms and applications. *IEEE access*, *3*, 490–530.