Learning To Play Atari Games With Bayesian Value Networks

Frederik Gram Kortegaard, Lasse Usbeck Andresen, 16/12/2022

Introduction

In recent years, games developed for the Atari 2600 video game console have become common sources of high-dimensional sensory data for testing state-of-the-art reinforcement learning algorithms. These environments present several challenges such as high-level feature extraction from raw pixel data, and reliable value-function approximation of high feature-spaces.

In this report we present our implementation of a deep reinforcement learning agent attempting to solve a subset of these environments, its design, architecture, and hyper-parameter definitions, as well as our results in section V. We further refer to sections I through IV for a mathematical description of the various methodologies used in this paper.

Design & Architecture

During the initial phase of this project, we spent a great deal of time reading current scientific literature regarding common deep learning approaches, and relevant artificial neural network architectures. What resulted from this period of research - combined with numerous tests - was a deep deterministic policy gradient (DDPG) agent, which in our implementation, utilizes the actor-critic method. An in depth explanation of the theoretical components of the DDPG algorithm can be seen in section II.

The DDPG algorithm was developed by DeepMind and is described in the paper "Continuous Control With Deep Reinforcement Learning" (Lillicrap et al, 2015), and has since its inception shown good results in the Atari environments
DDPG works by having a neural network (the actor) learn and optimize the policy, while another state-value neural network (the critic) continuously evaluates the actions of the policy network. This technique results in an increased level of stability in the network as explained in section II. Use of the DDPG algorithm on the Atari environments have proven to be relatively successful across a wide range of scientific literature, and therefore we decided on using DDPG as the foundation for our project as well. Prior to this conclusion, we also considered options such as variations of both SARSA and Expected SARSA, but as we were unable to achieve promising results, we decided to no longer pursue these approaches.

Following our implementation of the DDPG algorithm, we turned our attention to the artificial neural networks (ANNs), their architectures, and further design choices which ultimately serve as the value-function approximators inside of our DDPG agent. The architecture we settled on utilizes convolutional neural network layers as the input layers of our model, this decision was clear to us, as their primary purpose is to handle multi-dimensional feature spaces such as the images we observe from our Atari environments. The key idea behind the convolutional layers is that the use of *kernel filters* will allow the model to perform relatively complex feature extraction, finding elements such as borders, moving projectiles, and sprites, and is common practice in most studies working on the Atari 2600 environments. However, in contrast to most other studies, we wanted to experiment with using a higher quantity of layers to perform feature extractions, in the hopes of being able to further reduce the dimensionality of the data on which classification is performed. Sadly, this idea also poses specific challenges which are commonly associated with networks featuring a high amount of layers. One such obstacle is the phenomenon of *vanishing gradients*. This occurs when the amount of layers reach a point, where during back-propagation, the gradients being multiplied in the process end up approaching zero, decreasing the agents ability to learn.

In an attempt to overcome this, we decided to take inspiration from the paper concerning what is known as *RESnet*, *"Deep Residual Learning for Image Recognition"* by *K. He, X. Zhang, S. Ren and J. Sun* [1]. As we show in section I, this architecture allows back-propagation to reach earlier layers, which in turn allows larger networks to still converge efficiently while still utilizing the back-propagation algorithm, optimized using the ADAM optimizer.

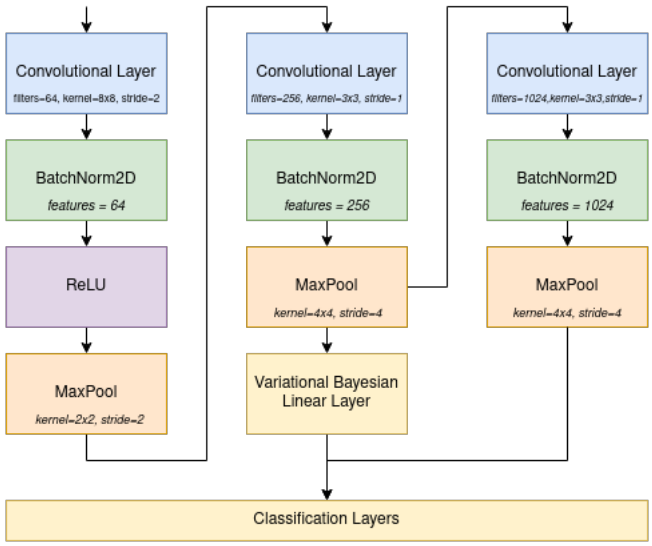

Figure 1: *Architecture of the Feature Extraction Layers in our pseudo-RESnet network.*

Another important design choice was the use of *Bayesian classification layers*. This term refers to linear neural network layers which use *Bayesian inference* to learn a probability distribution as opposed to typical neural networks which learn probabilities themselves, and not an entire distribution. In practice, this increases both stability and the networks ability to generalize, but at the cost of potentially slower convergence. Given that other agents in the current literature running on the Atari environments already converge at a relatively slow rate - typically trained for 10 to 20 million frames[2] - we can expect even slower convergence rates for our agent. This is important to consider when evaluating the performance of our agent, especially when comparing to similar agents with modelled Q-factors, such as those using more typical neural networks.

When training neural networks, a more disturbing tendency can also manifest; that of catastrophic forgetting. This problem can stem from various causes and can cause the symptom of drastic divergence in the network. Some of the most common

[1] https://ieeexplore.ieee.org/document/7780459, *"Deep Residual Learning for Image Recognition"*, *K. He, X. Zhang, S. Ren and J. Sun*, pp. 770-778
[2] https://arxiv.org/pdf/1312.5602.pdf, "Playing Atari with Deep Reinforcement Learning", Minh, Volodymyr, et al.

causes across both reinforcement learning and regular supervised learning, seems to be a gradient which is either exploding or crashing. If we consider the case of gradient exploding, we are faced with an unstable network, that will approach random behaviour. With gradient crashing however, we will experience a steady decrease in performance over the duration of our training.

This is a clear problem for models which needs to train over the same span of observations that agents working on Atari environments are typically trained for, and thus, we decided to look into potential solutions to modify either our loss function or our neural network, hoping to protect weights which are considered highly important for a given learning task. Our attempt at mitigating this problem, is by using *Elastic Weight Consolidation* (EWC) described in the paper *"Overcoming catastrophic forgetting in neural networks"* by *Kirkpatrick, et al.*[3].

By using this technique, we attempt to protect weights which are computed to be important for the current learning task, and protect them from drastic modification during the back-propagation process. The exact mechanisms with which this is done is described in section IV. Here it is important to note that this still allows the modification of such weights, but in an *"elastic"* manner, hence its name. In practice, whenever new task or sub-tasks are registered, old weights that are no longer considered important will slowly be forgotten by the network, resulting in a stabilization of the training process.

Lastly we note that by continual experimentation with different hyper-parameters, tested north of 80 configuration, we found that the best-performing values across all environments were:

Minimum Epsilon	0.1
Epsilon Decay	0.99992
Memory Size	5000
EWC Parameter Update	8
Gamma	0.99
Framestack/skip	4
Batch Size	32
Learning Rate	0.01

Preprocessing

While it is not required, further steps were taken to optimize not only the learning rate of our agent, but also the feasibility of training an agent on a plethora of hyper-parameter configurations which - if no further processing had been done - would not be possible within the time span of this project. With basis in the paper *"Playing Atari with Deep Reinforcement Learning"* by *Minh, Volodymyr, et al.*[4] we implemented a relatively basic prepossessing pipeline that converts our raw input data into grey-scale, and re-scaling it from dimensionality *(210,160,3)* to dimensionality *(84,84,1)*, ultimately resulting in an 85.7142% reduction of our input feature-space.

For the *Space Invaders* environment specifically, we further reduce the dimensionality of our input by way of cropping the observed environment and removing unimportant features. This gave us some valuable insight into how manual feature manipulation can be used in conjunction with generalized learning methods to further optimize performance.

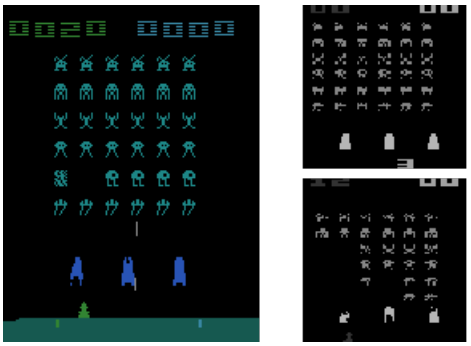

Figure 2: *Samples of observations from the Space Invaders environment before and after preprocessing*

While this preliminary preprocessing proved beneficial with regards to agent performance and required training duration, further methods were implemented targeting more specific areas of our agent. One of these techniques is called *frame-stacking*, which combines multiple observations and combines them into a higher-dimensional set of data, effectively encoding temporal data into our set of input features. We found this technique invaluable to our project, as it allows the agent to learn the velocities of features such as projectiles in *Space Invaders*, or the ball in *Tennis*. As our input features now encode a sense of motion, we are left with an agent whose action, and in turn its reward, is based on multiple frames at once, partially disjointing the input from its *true outcome*. As a way to mitigate this, we implemented *frame-skipping*, which forces an agent to repeat a given action for a set number of steps completely invariant of its input, which helps retain the correlation between input and outcome.

Theory

I. RESnet

As we briefly mentioned during the design discussion, our neural network uses an architecture loosely based around the RESnet architecture as proposed in the paper *"Deep Residual Learning for Image Recognition"*[5]

That being said, it should be noted that while our implementation is heavily inspired by RESnet, it is not a *true* RESnet architecture. This is a result of typical RESnet architectures - such as RESnet50 - being too large for the machines we have available for this project, and as such, we chose to design our own pseudo-RESnet architecture. The idea is still the same as that of a true RESnet architecture; we allow the input of a *"block"* of layers to propagate, while skipping over other blocks. This allows more direct back-propagation to gradients found in earlier layers of the network. Thus, this architecture allows us to implement more layers than usually would commonly be seen in agents playing Atari games.

II. The DPPG Method

DDPG uses four different function approximators:
the Q-value network θ^Q, a deterministic policy function θ^μ
a target Q network $\theta^{Q'}$ and finally a target policy network $\theta^{\mu'}$
Target networks are introduced to reduce inter-dependency between values calculated by the networks in the update function.

[3]https://arxiv.org/pdf/1612.00796.pdf, "Overcoming catastrophic forgetting in neural networks", Kirkpatrick, James, et al.
[4]https://arxiv.org/pdf/1312.5602.pdf, "Playing Atari with Deep Reinforcement Learning", Minh, Volodymyr, et al.
[5]https://ieeexplore.ieee.org/document/7780459, *"Deep Residual Learning for Image Recognition", K. He, X. Zhang, S. Ren and J. Sun* pp. 770-778

This results in a reduction of the risk of divergence in the network, which can manifest itself as catastrophic forgetting in the case of ANN's when used as function approximators. In our case, we experimented with on-policy methods for replacing the target networks, we did this with an approach inspired by the double SARSA-learning algorithm introduced by Sutton.

The action taken to the environment is then the average computed by both actors. For the updates, the target network is simply replaced by the second actor / critic to achieve the same effect. Instead of say, a soft replace to update targets, a parallel version of the update functions are provided for the second actor-critic. To stick with typical double learning methods, the actors update function also uses the averaged estimate of both critic networks.

The update function of DDPG is defined as first getting an updated Q-value using the bellman-equation, which we then use to minimize the squared loss between the standard Q-value and the updated Q-value:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}\|\theta^{\mu'})\|\theta^{Q'})$$
$$Loss = \frac{1}{N}\Sigma_i(y_i - Q(s_i, a_i\|\theta^Q))^2$$

Regarding the actor / policy update, we want to optimize the expected return:

$$J(0) = \mathbb{E}[Q(S, a)|s = s_t, a_t = \mu(s_t)]$$

and then calculate the derivative with respect to the policy parameter, applying the chain rule:

$$\nabla\theta^\mu J(0) \approx \nabla_a Q(s, a)\nabla_{\theta^\mu}\mu(s\|\theta^\mu)$$

Using experience replay, we can take the mean of the sum of the different gradients calculated from the batches:

$$\nabla\theta^\mu J(0) \approx$$
$$\frac{1}{N}\Sigma_i[\nabla_a Q(s, a)\nabla_{\theta^\mu}\|s = s_i, a_i = \mu(s_i)\nabla_{\theta^\mu}\mu(s\|\theta^\mu)\|s = s_i]$$

which gives the update function for the actor. To try to improve the rate of learning in the agent to ease experimentation, we also implemented N-step bootstrapping for the agent. The N-step method modifies the rewards such that they take into account how many steps away rewards are observed, applying a factor of the n-step and the gamma hyper parameters to them, and is explained in great detail in Sutton.

III. Variational Bayesian Inference Networks

Bayesian Neural network build on Bayes rule $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$, where $p(x|\theta)$ is the likelihood given the parameters on the data, $p(\theta)$ is the prior, which describes the distribution before receiving data. $p(x)$ is the evidence, which corresponds simply to the observed data and $p(\theta|x)$ is the posterior, which describes the model parameters after the next data is observed.

To choose between different models or parameters in this Bayesian settings, we can treat our model parameters as Hypothesis, and can calculate the posterior as $p(H|x) = \frac{p(x|H)p(H)}{p(x)}$, we can then further treat this hypothesis as a random variable, such that different distributions can be written as $PH_1(x|\theta_1) = p(x|\theta_1, H = 1)$ and $PH_2(x|\theta_2) = p(x|\theta_2, H = 2)$

We can then use Kullback-Leibler divergence to calculate average difference between the bit encoding of two hypothesis, where one hypothesis can represent the data and the other can represent an approximation. This is written in the following terms:

$$D_{KL}(P|Q) = \Sigma_{x\in X}P(x)log\frac{P(x)}{Q(x)}$$

For the case of optimizing on parameters we have

$$D_{KL}[(P(\theta|x)\|Q(\theta, \Omega)] = \int P(\theta|x)log\frac{P(\theta|x)}{Q(\theta|\Omega)}d\theta$$

Since the loss function, depends on the unknown of $p(\theta|x)$ we must try to find the posterior in different way than direct computation. We now introduce the concept of Variational Bayes for modeling the posterior:

$$D_{KL}[(Q(\theta|\Omega)\|P(\theta, x)] = \int Q(\theta|\Omega)logQ(\theta|\Omega)d\theta +$$
$$\int Q(\theta|\Omega)logp(X)d\theta - \int Q(\theta|\Omega)logP(\theta|X)d\theta$$

we can arrange the terms to Evidence Lower Bound (ELBO):

$$(x) = E_{q(\theta,\Omega)}[logp(\theta, x)] + H_{(\theta,\Omega)}[\theta] + D_{KL}[(Q(\theta|\Omega)\|P(\theta, x)$$

and use this inference to optimize the loss by calculating the gradient of ELBO on Ω and optimizing:

$$argmax_{q(\theta,\Omega)}L(\Omega) =$$
$$argmax_\Omega\Sigma_{n=1}^N E_{q(\theta,\Omega)}[logp(x_n|\theta)] - D_{KL}[(Q(\theta|\Omega)\|P(\theta, x)]$$

we then define our Bayesian Neural Network as:

$$p(D|\theta) = \Pi_{n=1}^N\mathcal{N}(y_n|f_\theta(x_n), g_\theta(x_n)) \text{ and } p(\theta) = \mathcal{N}(\theta|0, k^{-1}\mathbf{I})$$

and assuming the mean-field, we choose the variational distribution:

$$q(\theta; \Omega) = \Pi_{j\in\theta}\mathcal{N}(\theta_j|m_j, s_j^2),$$
$$\text{with } \Omega = \{m, S\}, m = \{m_j|j \in \theta\}, S = \{s_j^2|j \in \theta\}$$

we can then compute the loss, combining all the previous formulas:

$$L(\theta) = \Sigma_{n=1}^N E_{\mathcal{N}(\theta|m,S)}[log\mathcal{N}(y_n|f_\theta(x_n, g_\theta(x_n))]$$
$$- D_{KL}(\mathcal{N}(\theta|m, S)||\mathcal{N}(\theta|0, k^{-1}\mathbf{I})$$

IV. Elastic Weight Consolidation

Elastic weight Consolidation works with many of the same concepts as Bayesian Inference. Given $p/\theta)$, which describes the prior probability of the models parameters and D then $logp(\theta|D) = logp(D|\theta) + p(\theta) - logp(D)$. If we split the data into separate "tasks" then we get $logp(\theta|D) = logp(D_B|\theta) + p(\theta|D_A) - logp(D_B)$

As such the right hand side depends only on $logp(D_B)$. The posterior of task A must contain information about which parameters were important for it. Since it is intractable, it is calculated using Gaussian distribution and Fisher importance matrix. The updated loss function is then $L(0) = L_B(\theta) + \Sigma_i\frac{\lambda}{2}F_i(\theta - \theta_{a,i*})^2$

Results & Discussion

V. Agent Performance

We now present our results from testing the Agent on the Atari environments. Due to the nature of training on these high-feature space environments, we focused on producing agents with good results in the short term that continued to converge towards a better average score. Seeing as normally agents are trained for upwards of 20 million frames in the Atari sphere, and we typically only trained on a very small fraction of this number (the most being 1 million frames), we did not expect to achieve the same level of results as those seen in the studies we have cited throughout; this owes to the fact that even with our relatively powerful home-pc setups, training our particular agent took very long, given the fact that we use a relatively deep layered model compared to other studies.

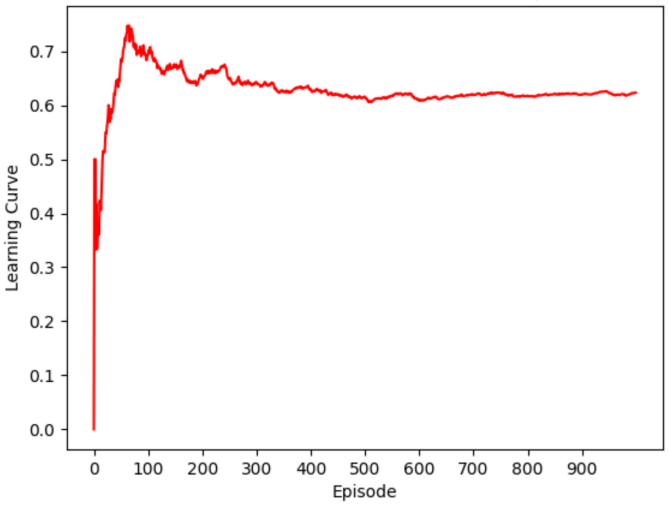

Figure 3: *Learning Curve from Atari 2600 Breakout*

As can be seen on Figure 3. showing the learning curve of the agent performing on the Breakout environment, the agent continues to improve over the initial randomness, then very slowly converges upward after an initial drop-off. Given the precautions we have taken to ensure we do not experience signs of catastrophic forgetfulness, we expect that this convergence would be continued given a longer exploration period and longer training overall.

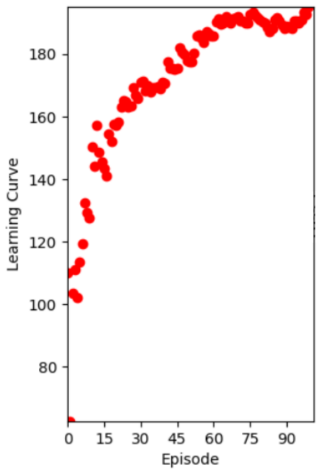

Figure 4: *Learning Curve from Atari 2600 Space Invaders*

This next result is from the agent training with hyper-parameters more optimized for the short term, with a much faster decay, and a minimum epsilon value corresponding to 10% randomness of action. This agent steadily continues to do better, and at the point it reached 100 episodes of training or about 100,000 frames, the convergence was still going in strong upwards trajectory. These results leads us to believe, that given a project timeline, where we have the knowledge that we posses now, we could perform long term training that would result in very respectable performance given our agents stability and upwards tendency.

We also tried to gain similar promising results from the Tennis environment, as we were to find out however, this one was quite a bit more challenging than the other two. There is many more actions available at each frame of the game, naturally increasing the feature-space complexity. Looking at other attempts online to, we fund similar problems would be encountered by other people of similar levels of experience to ourselves. Most of the time, when the environment was solved, it was due to even longer training periods than those seen for Breakout and Space Invaders.

On Figure 5. A learning curve from an agent which did not utilize the double setup of actor-critics, nor the n-step, pseudo-RESnet, n-step or EWC loss, but otherwise trained on the same hyper-parameters as the agent in Figure 4.

This agent quickly experienced quite alarming levels of divergence, to the point that we decided to stop the experiment a bit early. From this, we gain more confidence that the methods we employed helped stabilize the learning of the agent.

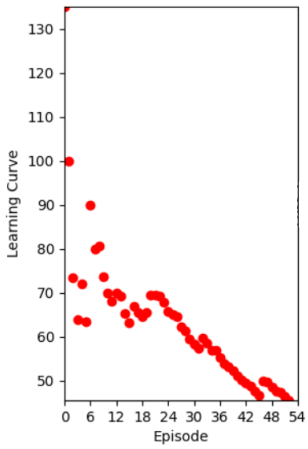

Figure 5: *Learning Curve from Atari 2600 Space Invaders, no EWC or pseudo-RESnet*

VI. Final Remarks

Through this project, we have refined our understanding of deep reinforcement learning, and experienced challenges that comes with these field and large feature-spaces. We also extended our methods with techniques from computer vision to be able to operate upon the pixel input information.

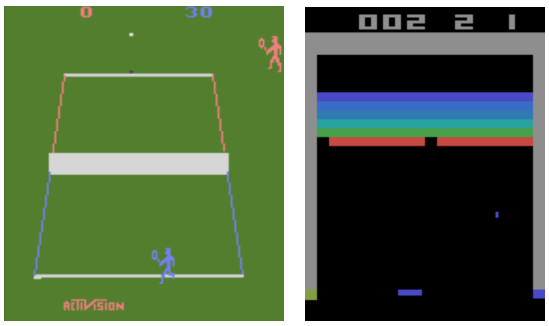

Figure 6: *Snippets of the Atari 2600 Tennis (Left) and Breakout (Right) Environments*

The agent we present is able to achieve *non-trivial* levels of learning on the tested environments despite their complexity. One of the lessons we learnt throughout this project, was that creating *generalizeable* agents, inferring a very limited amount of methods such as reward manipulation, greatly increases the difficulty of the task at hand.

While none of our results can compare with the DeepMind and EWC paper, we feel that the methods we employed have proven to have some validity with regards to long-term improvements.