Learning to play Atari games with deep Expected Sarsa

Kirsten H. L. Timm and Sofie Amalia Boye
IMADA, SDU
kitim19@student.sdu.dk, soboy16@student.sdu.dk

May 31, 2024

1 Introduction

The project aims to implement deep Expected Sarsa with a Bayesian neural network to train an agent to play Atari games. In this report, we will present our results on three different Atari games after an introduction to the theory behind and our design choices towards an Expected Sarsa algorithm. We will end with a discussion of the results, which limitations there are and a conclusion on the project.

1.1 Atari games

The project will focus on a small selection of Atari 2600 games implemented in The Arcade Learning Environment (ALE). As these are designed to be challenging for humans, they also present a challenge for our agent. Atari games have a continuous state space and a discrete action space.
The agent must learn from high dimensional visual input (210×160 RGB video at 60Hz), here we employ convolutional layers and common Atari-based pre-processing techniques, such as down-scaling and gray-scaling[Sei16]. Prepossessing of frames considerably reduces runtime.
Further, a common procedure for Atari games is frame-skipping. Here, if frame-skip = 4, only every fourth frame is considered. This technique has proven to greatly improve performance and it results in faster training.[Kal+21] It is very important to choose an appropriate frame skip parameter, for example 'Breakout' and 'Pong' achieve best results with a low frame skip, as these need fine actions to reach the specific states that will block the ball. [Bra+15]

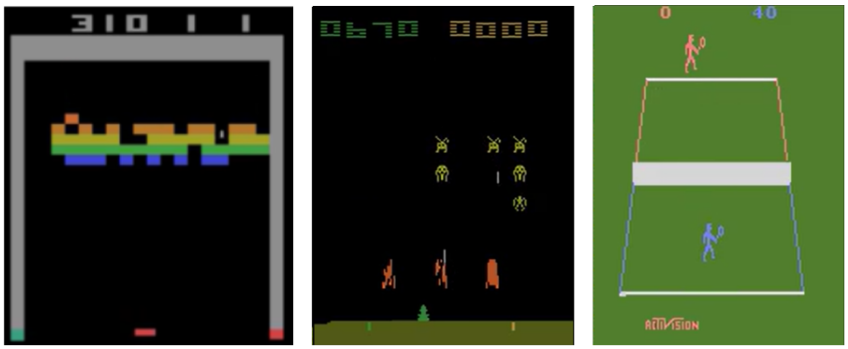

Figure 1: Agent playing Atari Breakout, Space Invaders and Tennis

2 DQN on Atari games

As a starting point for our project, we have searched for a GitHub repository where Deep Q-Learning (DQN) was implemented for Atari games. We chose to use Adhi Setiawan's work[Set23] as a baseline and have modified this to reach our own Bayesian Expected Sarsa. His repository is inspired by CleanRL [Hua+22], which in turn is based on the work of Mnih and colleagues, published in the paper "Human-level control through deep reinforcement learning"[Mni+15].

The main file dqn_atari.py sets up an Atari game environment and prepares the input for the neural network. A Q-network is defined to use for updating the Q-values of the states during the training. The main training loop is run for a defined number of time steps. It uses an epsilon-greedy policy to choose actions, stores the relevant data, and ends with updating the Q-network using Q-learning. The training process is saved using TensorBoard which makes it possible to plot the process from a saved model.

As an intermediate result we saw that the DQN model performed quite well on the 'Breakout' environment, keeping up with performance of Clean RL's DQN. From here the next step is to modify the code to Expected Sarsa with a Bayesian neural network.

3 Bayesian Deep Expected Sarsa

We take inspiration in the pseudo-code for "DQN with Experience Replay" from the paper Playing Atari with Deep Reinforcement Learning [Mni+13] as a baseline for our final algorithm. Since this algorithm uses Q-learning, we change the pseudo-code from Q-learning to Expected Sarsa and integrate a Bayesian neural network instead of the standard neural network. Below, we will briefly explore the differences in the methods.

3.1 Q-Learning

Q-learning is an off-policy method that learns the optimal action-value function $Q(s, a)$ by updating Q-values based on observed steps in the environment. It selects actions greedily concerning the current estimate of the optimal Q-values. The Q-value update below reflects the temporal difference error, where the new Q-value is a combination of the immediate reward, the discounted maximum Q-value of the next state, and the learning rate α:

$$Q\left(s_t, a_t\right) = Q\left(s_t, a_t\right) + \alpha \left(r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right)$$

3.2 Sarsa

Sarsa, which stands for State Action Reward State Action, is in contrast to Q-learning an on-policy method. This means it is updating the policy while following it, that is the target policy and the behavior policy is the same. It selects actions according to the current policy and updates the Q-values based on the actions taken in the next state. The Sarsa update equation reflects this on-policy nature where the new Q-value is updated based on the immediate reward received, the discounted Q-value of the next state, and the learning rate.

$$Q\left(S_t, A_t\right) = Q\left(S_t, A_t\right) + \alpha \left(R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right)$$

3.3 Expected Sarsa

Expected Sarsa is a variant of Sarsa, that estimates the Q-value of each action in a given state and uses these estimates to choose which action to take in the next state. Expected Sarsa estimates the Q-value as the expectation of the Q-value of the next state with respect to the policy. In practice this is taking a weighted average of the Q-values of all possible actions in the next state. The Q-value update equation in Expected Sarsa shows the Q-value is updated based on the immediate reward received, the discounted Expected future reward considering all possible actions weighted by their probabilities under the current policy, and the learning rate.

$$Q\left(s_t, a_t\right) = Q\left(s_t, a_t\right) + \alpha \left(r_{t+1} + \gamma \sum_a \pi\left(a \mid s_{t+1}\right) Q\left(s_{t+1}, a\right) - Q\left(s_t, a_t\right)\right)$$

3.4 Bayesian Neural Network

A Bayesian neural network (BNN) is a stochastic neural network trained using Bayesian inference. That is, a neural network where each weight follows a probability distribution rather than being

learned deterministic values. BNNs can be considered as an ensemble method since the stochasticity of the weights simulates multiple possible models θ with their associated probability $p(\theta)$.[Jos+20] Additionally, the randomness of the weights will let us directly inherit a soft policy from the BNN.

For given data $\mathcal{D} = \{(x_n, y_n) \mid n = 1, \ldots, N\}$, a Bayesian neural net is defined as the data generating process below [Kan22]

$$p(\mathcal{D} \mid \theta) = \prod_{n=1}^{N} \mathcal{N}\left(y_n \mid f_\theta\left(x_n\right), g_\theta\left(x_n\right)\right) p(\theta) = \mathcal{N}\left(\theta \mid 0, \kappa^{-1} I\right),$$

where

$$f_\theta(x) = W_2^T \sigma\left(W_1^T x\right), \quad g_\theta(x) = \exp\left(W_3^T \sigma\left(W_1^T x\right)\right),$$

$\kappa \in \mathbb{R}_+$and $\theta = \{W_1, W_2, W_3\}$.

From this, we see that both f_θ and g_θ are neural networks. Notice that both networks share W_1, while they have different weights, W_2 and W_3, at the top layer. This design is called head-split. Although this a common practice, the network would still be a BNN if f_θ and g_θ were two separate networks.

3.4.1 Network Design

The design of our Bayesian neural network is illustrated in figure 2. Since the data from the Atari games is images the network primarily consists of convolutional layers. Convolutional layers employ kernel filters which preserves the positional information of pixels and allows for very complex feature extraction, such as identifying borders and the position of a ball. Between the convolutional layers we have a ReLu activation function. This is a very commonly used activation function for convolutional neural networks. As for all activation functions, ReLu introduces non-linearity into the model, which means that it can learn more complex patterns. Further, ReLu is a computationally efficient function as it only needs to find the max of values.

As the last layer we have a Bayesian layer. In this layer the network learns a distribution for each weight instead of learning a deterministic value as in traditional linear layers. This allows the network to model an uncertainty. Although, uncertainty estimates accommodate robust control, direct exploration and safe exploration, it is still only employed in few works. This might be because it is quite time consuming to implement and train these models.[**tran2019Bayesian**]

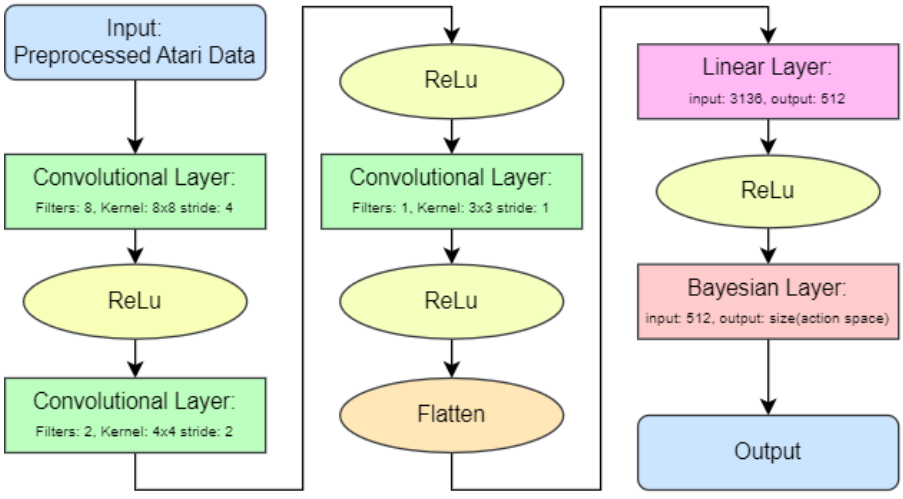

Figure 2: Bayesian Neural Network design

3.5 Final Pseudo-code

After adding the described changes we end up with the following pseudo-code for our Bayesian Expected Sarsa algorithm:

Algorithm 1 Bayesian Expected Sarsa

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode=1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for t=1, T **do**
    With probability ϵ select a random action a_t
    otherwise $\pi(\cdot|s_t) = Q_\theta(\cdot, s_t)$ ▷ Soft policy from Bayesian network
    Execute action a_t and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    if t mod training frequency == 0 **then**
      Sample random mini batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
      Set $y_j = \begin{cases} r_j & \textit{For terminal } \phi_{j+1} \\ r_j + \gamma \sum_{a_{t+1}} \pi\left(a_{t+1} \mid \phi_{t+1}\right) Q\left(\phi_{t+1}, a_{t+1}; \theta\right) & \textit{For non-terminal } \phi_{j+1} \end{cases}$
      Perform a gradient step on $(y_j - Q(\phi_j, a_j; \theta))^2$

4 Results

In this section, we will discuss the result from running DQN, Expected Sarsa and Expected Sarsa with a Bayesian neural network on Breakout, Space Invaders and Tennis. Code, plots, and a video for each game can be seen at Github `https://github.com/soboy16/RL_Project`. To preserve comparability, all runs shared the same parameters listed in the table below.

Total timesteps	5,000,000
Learning rate	0.0001
Buffer size	100,000
Discount factor	0.99
Starting epsilon	1
Epsilon decay	over the first 10% of total time steps
Minimum epsilon	0.01
Learning starts	50,000
Training frequency	4

4.1 Breakout

We have run DQN, Expected Sarsa, and Bayesian Expected Sarsa. Each of the three has been run for 5 million steps and the resulting learning curve can be seen in figure 3. The plot shows an average curve for all algorithms with the actual values as a shaded area behind. The average curve is created with the smooth function in Python which applies a moving average technique over the data points. It calculates the mean inside a window size of 500. The average curve is created to better represent the trends for each of the different algorithms.

By looking at the average curves we observe all three learning curves mostly follows the same slope before reaching 2 million episodes. From here the three models seems to have converged, where DQN performs better than both Expected Sarsa and Bayesian Expected Sarsa. Up until 4 million episodes Expected Sarsa performs better than Bayesian Expected Sarsa, but after the 4 million episodes the Bayesian Expected Sarsa overtakes the Expected Sarsa.

Figure 3: DQN (blue), Expected Sarsa (green) and Bayesian Expected Sarsa (red) learning curves for Breakout. Solid lines represent the average episodic return, while the shaded curve represent the actual episodic return

4.2 Space Invaders

Also for Space invaders, we have run the three algorithms DQN, Expected Sarsa, and Bayesian Expected Sarsa. A plot with the average curve was created in the same way as described for Breakout and can be seen in figure 4.

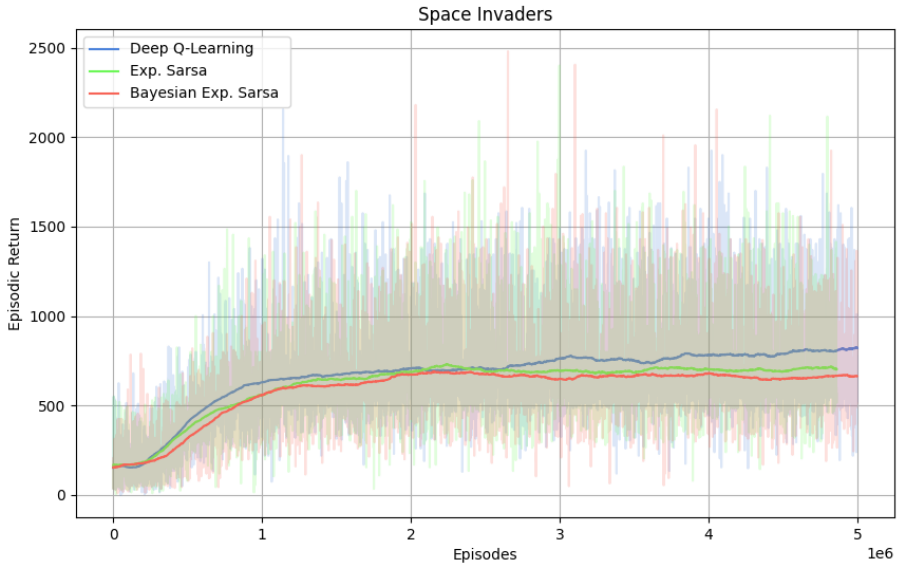

Figure 4: DQN (blue), Expected Sarsa (green) and Bayesian Expected Sarsa (red) learning curves for Space Invaders. Solid lines represent the average episodic return, while the shaded curve represent the actual episodic return

Space Invaders follow the same trend as Breakout where DQN has the best overall performance. Again all three curves seem to largely follow the same slop during the first million episodes, where they each converge to an episodic return around 750. Although, DQN has a slightly larger average episodic return, the two other models follows very close behind. Further, when looking at the actual curve for Bayesian Expected Sarsa (shaded red), we observe a couple of very high peaks reaching values significantly larger than 2000.

4.3 Tennis

Tennis has shown to be more difficult for our algorithms than Breakout and Space Invaders. Since the results for Tennis weren't as promising, and the training for a long time proved to not give any real change from the initial episodic return of -24, we have only run DQN and Bayesian Expected Sarsa for this game. We have also chosen to only run for half as many episodes; 2.5 million because Tennis was training slower than the others and we wanted to prioritize time to test other things.

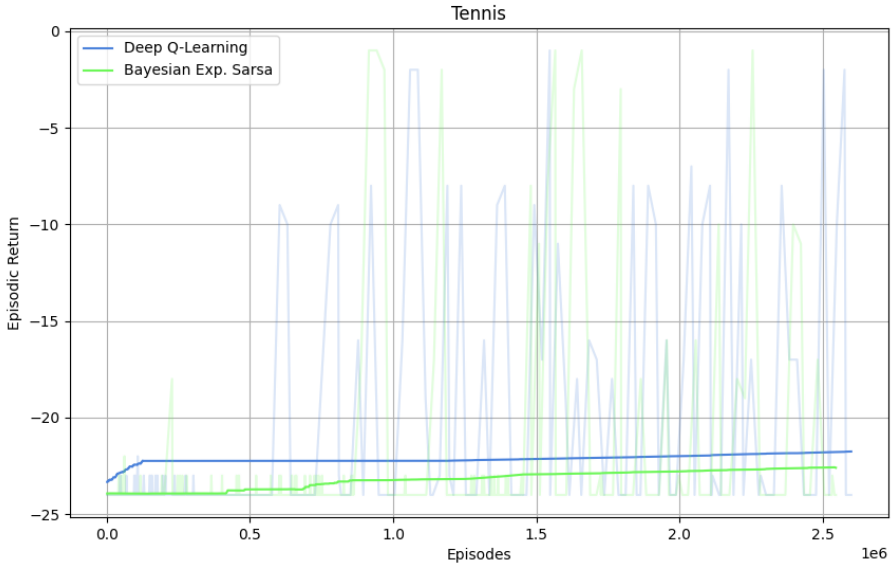

Figure 5: DQN (blue) and Bayesian Expected Sarsa (green) for Tennis. Solid lines represent the average episodic return, while the shaded curves represent the actual episodic return

While we observe that the actual episodic for both DQN and Bayesian Sarsa return reaches values of -1 several times, the average episodic return curves stays below -20. As for Breakout and space invaders, DQN has greater average episodic return.

4.4 Training frequency

To test how the performance is impacted by the model update interval, we have run Bayesian Expected Sarsa with different training frequency updates for Breakout. Our default value is the training frequency at 4, but we have furthermore tested with training frequency on 12, 25, 50 and 100. The results can be seen in figure 6.

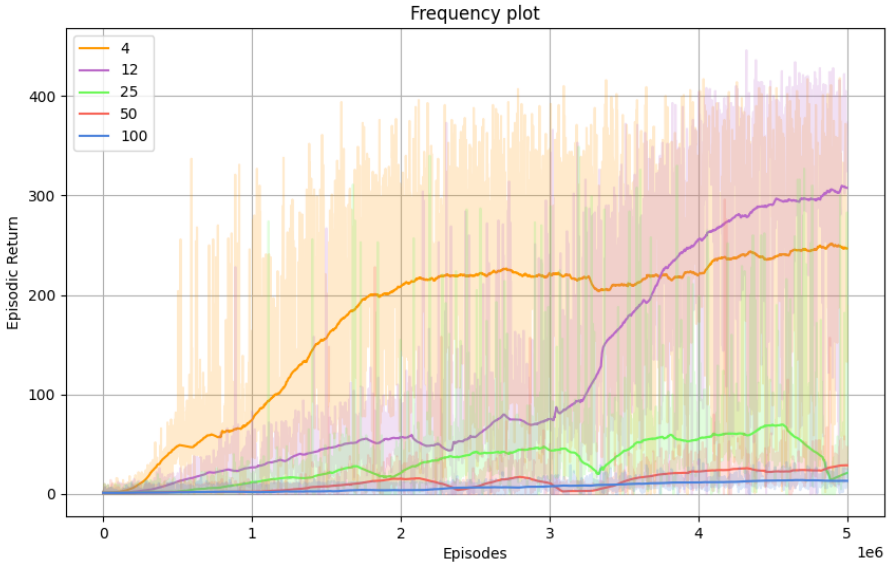

Figure 6: Results from different training frequencies on Breakout; 4(orange), 12(purple), 25(green), 50(red) and 100(blue). Solid lines represent the average episodic return, while the shaded curve represent the actual episodic return

From the frequency plot, we observe that best performance is achieved by the training frequency at 12, while the highest value for training frequency gives the worst performance.
The curve for training frequency of 100 is mostly just a flat line, indication that the model is only learning very little. For training frequency of 50, we begin to some improvement in the performance, with an average episodic return slightly larger than the average for training frequency 100. Also around 4.1 million there is a very high peak the actual episodic return for training frequency 50 reaches 296. Again for training frequency 25, we see an improvement compared to the average for training frequency 50, but it is very unstable and at close to 5 million takes a dive in the curve. For the lower frequencies on 4 and 12, we see a significant improvement in the training. The model with training frequency of 12 reaches the highest performance with the average episodic return climbing to values larger than 300 at 5 million episodes. We can also see that even though the training frequency on 4 has the steepest curve at the beginning it is overtaken at 3.8 million steps by the one with frequency 12 and for the last million steps the one with frequency 12 has a higher average than 4.

4.5 Maximal episodic return

Another measure of performance is to look at the maximal episodic return for each of the models. State of the art and other algorithms are evaluated and their performance compared by the maximal episodic return on Papers with Code, so by looking at this measure for our models, we will be able to compare with the state of the art algorithms.[AI]

	Training frequency	Breakout	Space Invaders	Tennis
DQN	4	484	2155	-1
Expected Sarsa	4	427	2400	-
Bayesian Expected Sarsa	4	418	2480	-1
	12	446	-	-
	25	349	-	-
	50	296	-	-
	100	35	-	-

5 Discussion

5.1 Performance on Atari games

From the results we saw that looking at the average episodic return, DQN had the best performances across all three games, although the margin between the models performances were small for Space Invaders an tennis. This result might be the effect of DQN usage of a target network that provides stable target values for updating the Q-values. Another advantage of DQN compared with Expected Sarsa is the use of experience replay. Further Sarsa tends to be more conservative, as it takes into account the possible penalties from exploratory moves, while DQN can ignore these. This means that Sarsa might not choose optimal paths close to large negative rewards, even if these path might give the best performance in the end.

Although DQN had the overall best performance, we did observe that the Bayesian Expected Sarsa showed to be promising. For Breakout, the average episodic return for the Bayesian Expected Sarsa started as the lowest of the three algorithms, but after 4 million it overtook Expected Sarsa and moved closer to the average episodic return for DQN up until the and of the plot at 5 million. This might be the effect that as the Bayesian layers are stochastic it might be more time consuming to train the network. Further, for Space Invaders we saw that Bayesian Expected Sarsa had some very tall peaks, and looking at the maximal episodic return it is confirmed that Bayesian Expected Sarsa has achieved the best result if we look at a single episode. This could be explained by the stochasticity in the layers which open up for exploring considerably new paths even after convergence.

For Tennis, we have not achieved the same learning curves as for Breakout and Space Invaders. This might be because Tennis is a more complex game. First of all, in Tennis, the player can move both horizontally and vertically while in the other games the only moves are horizontal. Tennis is also a two-player game that adds the challenge of predicting the other players' moves.

The result for testing different training frequencies showed that the high values of 25, 50, and 100 tend to have a slow learning curve and not achieve good results. Even though the training frequency at 25 and 50 reaches a max episodic return of 296 and 348, it is only a few peaks that get over 100 and the average is still low. For the lower values, at the model update intervals at 4 and 12, we get a better learning curve. The one with 4 has the steepest learning curve in the beginning but is overtaken by 12 which reaches the best performance. The training frequency at 12 also reaches the highest episodic return for our Bayesian Expected Sarsa at 446. As it doesn't look like the learning curve for training frequency of 12 has converged within our plotting window, it is possible that it continue to improve after the 5 million steps.

Comparing to the state of the art, we see that some variations of DQN have shown better results for the different games. For Breakout, Bootstrapped DQN has a higher performance on average and reaches a maximum evaluation score of 855. [Osb+16]. This is a significantly larger episodic return than our best model for Breakout, which was DQN at 484. For Space Invaders and Tennis, Double Deep Q-Learning has shown better results [HGS15]. Here the maximum return for tennis reached 11.8, compared to our models with only -1 as the maximum episodic return. For Space Invaders we achieved a result of 2480 for Bayesian Expected Sarsa, while the maximal return is at 4715.8 for Double Deep Q-Learning.

In the table presented in Deep Exploration via Bootstrapped DQN [Osb+16] they also compare to human performance which are 30.5 for Breakout, 1668.7 for Space Invaders, and -23.8 for Tennis. So our agents have scored better than human performance in all three games.

5.2 Limitations

We have limited computer resources to train the different algorithms on the three Atari games. A run with 5 million steps takes around 26 hours to complete and together with a time limit for this project, we have limited possibility to test out different settings and hyperparameters. It is also normal for Atari games to be trained on for example 10 million steps as the CleanRL does on DQN. However, due to limited resources and the time it takes, we have chosen to train our algorithms on 5 million steps. A better setup, for example with a GPU, could result in better convergence for the algorithms when trained on more steps.

We are only using single environment training, which in some cases can be a limitation. When having multiple parallel environments the agent can collect more experiences simultaneously which can lead to faster learning and better exploration.

5.3 Future improvements

There are different opportunities to improve the neural networks used in the algorithms. Both in terms of trying out different layers and positions of layers. Some possibilities include implementing techniques such as:

- Batch normalization to stabilize the training process by normalizing the input for the layers.
- Max pooling to reduce the dimensions and computational load by downsampling the feature maps.
- Dropout to make the models more robust by dropping out random units during training.

Another improvement is to test out different training frequencies below 20, because the difference we see between the learning curves of 4 and 12 is interesting to dive more into. When finding the optimal training frequency it could be insightful to test the different algorithms again. Or just to test with the training frequency of 12, since the results were very promising in our Bayesian Expected Sarsa compared to 4, which we ran our experiments on. However, we discovered this too late to make new experiments and therefore leave it to future work.

6 Conclusion

In this project, we have worked on implementing Bayesian Deep Expected Sarsa to play three Atari games: Breakout, Space Invaders, and Tennis. Atari games pose a significant challenge for the model, where it must learn from high-dimensional visual input. To tackle this we have done some preprocessing of the frames from the games and modeled the Q-network using a convolutional neural network.

We base our code on the GitHub repository 'atari-dqn' by Adhi Setiawan. This code implements DQN for Atari games. We have then modified the code to achieve an implementation of Bayesian Expected Sarsa.

We reached a good performance and learning curve for Breakout and Space Invaders even though it can't compete with the best state of the art algorithms such as Double Deep Q-Learning. However, the results lay very close to DQN and also beats human performance in all three games.

From testing different training frequencies, we saw performance was very sensitive to change in this hyperparameter. In general, choosing a lower value for training frequency, i.e. training the network more often, gives the best results. Although this is the overall picture, we saw that the performance when choosing a training frequency of 12 exceeded the one for a training frequency of 4.

To achieve a better performance in future work, it would be relevant to look at testing different neural network settings and either running the algorithms with a training frequency of 12 or testing for an optimal training frequency.

Bibliography

[Mni+13] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[Bra+15] Alex Braylan et al. "Frame skip is a powerful parameter for learning to play atari". In: *Workshops at the twenty-ninth AAAI conference on artificial intelligence*. 2015.

[HGS15] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533. URL: https://api.semanticscholar.org/CorpusID:205242740.

[Osb+16] Ian Osband et al. "Deep Exploration via Bootstrapped DQN". In: *CoRR* abs/1602.04621 (2016). arXiv: 1602.04621. URL: http://arxiv.org/abs/1602.04621.

[Sei16] Daniel Seita. "Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games". In: (2016). URL: https://danieltakeshi.github.io/2016/11/25/frame-skipping-and-preprocessing-for-deep-q-networks-on-atari-2600-games/.

[Jos+20] Laurent Valentin Jospin et al. "Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users". In: *CoRR* abs/2007.06823 (2020). arXiv: 2007.06823. URL: https://arxiv.org/abs/2007.06823.

[Kal+21] Shivaram Kalyanakrishnan et al. "An analysis of frame-skipping in reinforcement learning". In: *arXiv preprint arXiv:2102.03718* (2021).

[Hua+22] Shengyi Huang et al. "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms". In: *Journal of Machine Learning Research* 23.274 (2022), pp. 1–18. URL: http://jmlr.org/papers/v23/21-1342.html.

[Kan22] Melih Kandemir. *8 - Modern Bayesian Inference*. 2022.

[Set23] Adhi Setiawan. *atari-dqn*. https://github.com/adhiiisetiawan/atari-dqn. 2023.

[AI] Meta AI. *Papers with code - Atari Games Benchmark*. URL: https://paperswithcode.com/task/atari-games.