INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 5, Issue 1)
Available online at: www.ijariit.com

Improving generalization in reinforcement learning on Atari 2600 games

Abhi Savaliya
abhisavaliya01@gmail.com
Simon Fraser University, Burnaby, Canada

Chirag Ahuja
cahuja19@gmail.com
Simon Fraser University, Burnaby, Canada

Chirayu Shah
cshah952@gmail.com
Simon Fraser University, Burnaby, Canada

Sagar Parikh
sagarparikh2010@gmail.com
Simon Fraser University, Burnaby, Canada

ABSTRACT

Deep Reinforcement Learning (DRL) is poised to revolutionize the field of artificial intelligence (AI) and represents a crucial step towards building autonomous systems with a higher-level understanding of the world around them. In particular, deep reinforcement learning has changed the landscape of autonomous agents by achieving superhuman performance on board game Go, a significant milestone in AI research. In this project, we attempt to train a Deep RL network on Demon Attack - an Atari 2600 game and test the model on different game environments to investigate the feasibility of applying Transfer Learning on environments with same action space but slightly different state space. We further extend the project to use established Reinforcement Learning techniques such as DQN, Dueling DQN and SARSA to examine whether RL agents can be generalized on unfamiliar environments by fine-tuning the hyperparameters. Finally, we borrow classic regularization techniques like l2 regularization and dropout from the world of supervised learning and probe whether these techniques which have received very limited attention in the domain of reinforcement learning are effective in reducing overfitting of Deep RL networks. Deep Networks are expensive to train and complex models take weeks to train using expensive GPUs. We find that the use of above techniques prevents the network from overfitting on current environment and gives satisfactory results when tested on slightly different environments thus enabling substantial savings in training time and resources.

Keywords— *Machine learning, Artificial Intelligence, Deep learning, Keras RL, Reinforcement learning*

1. INTRODUCTION

Reinforcement Learning is an area of Machine Learning where an agent learns to behave in an environment by performing stochastic actions, observing the rewards/results it gets from its actions and then tries to maximize the cumulative reward. Google's DeepMind [1] has demonstrated how an AI agent can learn to play games without prior information of what it's supposed to do from a given action space resulting in a superhuman performance on the board game Go. This kind of AI learns by trial and error where after each action the agent receives some feedback in the form of a reward. This reward is generated by the reward function and the new state that is determined based on the interaction between the RL agent with its environment. The holy grail of reinforcement learning is to produce fully autonomous agents that interact with hitherto unseen environments and learn optimal behaviours.

Training a reinforcement model on a single environment takes an abundance of compute time and very often the trained model fails spectacularly when tested in unfamiliar yet similar environments. In particular, our aim is to build and train an RL agent that performs reasonably well in different environments having the same action space. To tackle this problem, we borrow some well-known ideas to avoid the overfitting problem from the field of supervised learning.

2. METHODOLOGY

We attempt to improve the generalization problem in Deep RL networks using well known supervised learning techniques like L2 regularization and dropout to improve performance in unfamiliar environments. Crafting an AI model requires a lot of trial and error and usage of different approaches and techniques. We found that Deep RL agents almost every time over fit on the training environment which we plan to reduce. We have used three established Deep RL networks as published by Google's DeepMind namely DQN (Deep Q-Networks), Dueling- DQN and SARSA (State-Action-Reward-State-Action).

2.1 Overview of the methodology

The below figure depicts our approach to training various Deep RL models using different techniques for hyper parameter optimization and get the best possible training result.

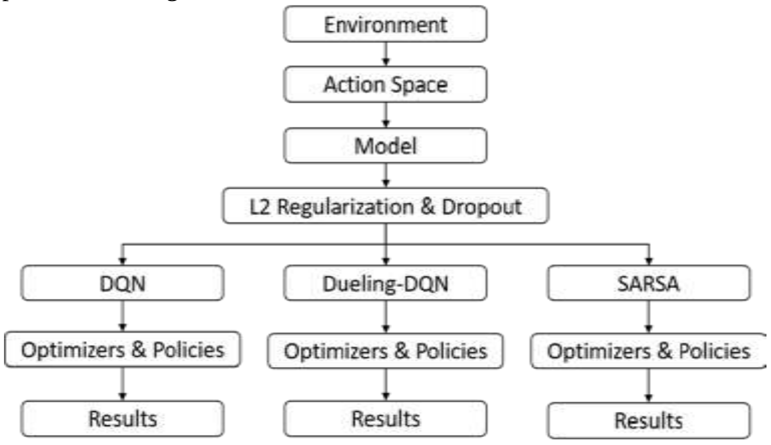


Fig. 1: Overview of the methodology for implementation (top-down)

We have chosen the Atari – 2600 game emulator as our environment for training the model. The model is trained on Demon Attack which belongs to the Atari 2600 collection. It gives us the optimum amount of state and action space which can help us to generalize the model better for performance in different environments.

2.2 Methods to control overfitting: L2 regularization and Dropout

In reinforcement learning, we train the model using various agents and save those weights. Again, we use the same saved weights to test on the same environment. However, when we use these same set of weights in different environments, the model doesn't perform well even though it has same action space. This basically means that the model overfits on the environment.

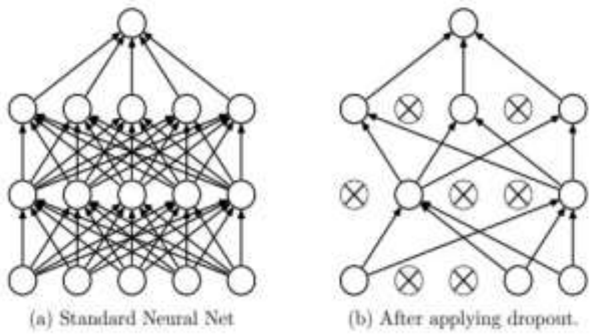
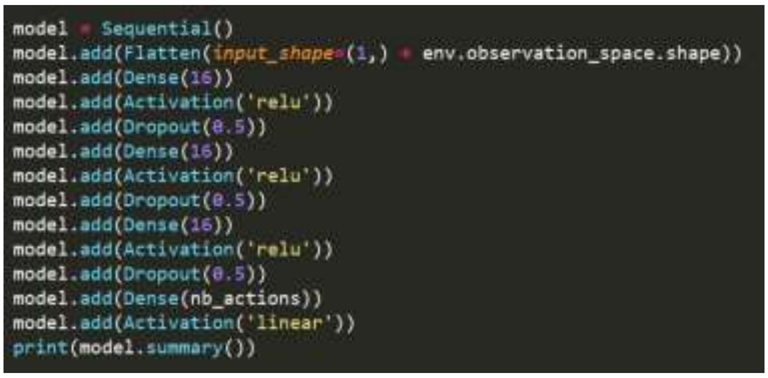


Fig. 2: Thinned network after applying Dropout (*Left-to-right*) [6]



```
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_actions))
model.add(Activation('linear'))
print(model.summary())
```

Fig. 3: Neural Network Architecture with Dropout (rate - 0.5)

Our hypothesis is that introducing regularization will allow the Deep RL network to learn more general features in the environment which can be then reutilized in other similar environments. The above code snippet in Figure 3 shows the architecture adopted by us to overcome overfitting of the Deep RL networks. The neural net consists of an input layer, 3 hidden layers and an output layer which outputs the probabilities of the action to be taken by the agent. After experimenting with both Dropout and L2 regularization, we found that introducing dropout outperformed L2 regularizer. Also, it was found that individually L2 and dropout gave good performances, but they didn't work so well together. Consequently, we decided to move forward with the only Dropout.

2.3 Agents

2.3.1 DQN - Deep Q Learning: DQN uses one particularly successful architecture, the deep convolutional network, which uses hierarchical layers of tiled convolutional filters to mimic the effects of receptive fields. It considers tasks in which the agent interacts with an environment through a sequence of observations, actions and rewards. The goal of the agent is to select actions in a fashion that maximizes cumulative future reward. More formally, we use a deep convolutional neural network to approximate the optimal action-value function which is the maximum sum of rewards r_t discounted by γ at each time step t, achievable by a behavior policy, after making an observation (s) and taking an action (a). [2]

$$Q^*(s,a) = \max_{\pi} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a, \pi]$$

2.3.2 DDQN - Dueling Deep Q Learning: The Dueling DQN neural network architecture is a combination of 2 streams- one stream for approximating state value function *V(s)* and another for approximating the corresponding advantage function $A(s, a)$ given the state [8]. The output of these two streams is aggregated to estimate an optimal action-value function Q. One advantage of having two components to the action-value function Q is that since the estimation of state representation and its corresponding advantage are separate entities, it leads to a more simplified model where the network need not be changed for future Deep RL algorithms. Yet another convenience we derive from using the Dueling architecture is that it learns to evaluate which states are to be ignored and which states play a crucial role in completing the task thus making the choice of action more relevant to the importance of current the state. [8] The relationship between the state value function *V(s)*, advantage function *A(s,a)* and action - value function *Q* can be written as:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha) \right)$$

Where, $\boldsymbol{\theta}$ denotes the parameters of convolutional layers and α,β represent the parameters of the two streams of fully connected layers. A depiction of Deep DQN network can be seen in Figure 4.

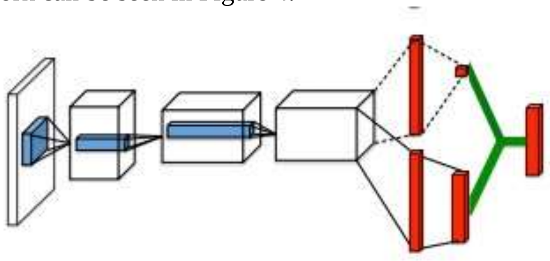


Fig. 4: The two streams of Dueling architecture approximate the state-value and advantage for each action - the green output module combines both these values to generate a Q-value

2.3.3 SARSA - State Action Reward State Action: SARSA, an on-policy algorithm is incorporated with deep convolutional networks to train the model on a specific video game. SARSA represents the current state of the agent "*S1*", the action the agent chooses "*A1*", the reward "*R*" the agent gets for choosing this action, the state "*S2*" that the agent enters after taking that action, and finally the next action "*A2*" the agent choose in its new state, which collectively updates the Q-value. The acronym for (s_t, a_t, r_t, s_{t+1}, a_{t+1}) is SARSA. [7]

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1})Q - (s_t,a_t)]$$

The Q value is updated as shown above. SARSA interacts with the environment and updates the policy depending on the selected actions, and hence it is an on-policy algorithm. The Q value for a state-action is updated by an error, adjusted by the learning rate alpha. Q values represent the possible reward received in the next time step for taking action A in state S, plus the discounted future rewards received from the next state-action observation.

2.4 Optimizers and policies

In policy-based reinforcement learning, we directly learn a policy function which maps state space SS to action space A without worrying about value functions. A policy can be thought of as brain of the RL agent. In simpler terms, a policy π is a function which takes as input a state S and outputs an action A i.e. $\pi(\text{S}) \rightarrow a$ [5]. To find an optimal policy for the RL agent, fine tune the hyper-parameters and improve training accuracy, we tried training the model using ADAM optimizer, but the results left a lot to be desired. Hence, we experimented with different optimizers like ADAM, ADAMDELTA, SGD, RMSprop, Adagrad, Adadelta etc. in search of the most efficient optimizer for our use case.

Adaptive Moment Estimation (Adam) [4] is an optimizer that tailors or adapts learning parameter for each of the input features. Adam computes the decaying averages of past and past squared gradients which are estimates of the first moment and second moment of the gradients respectively. Hence this optimizer prefers flat minima in the error surface which can be a limitation in some cases.

The Adagrad optimizer also tailors the learning parameters to features, preferring low learning rates for parameters associated with frequently occurring features and higher learning rates for parameters associated infrequent features [9]. One of the problems with Adagrad optimizer is that as the training progresses, the learning rate tends to be zero at which point the learning stops.

Adadelta is an extension of Adagrad optimizer which aims to remove the deficiencies of the Adagrad by restricting the number of accumulated past gradients to some fixed window size w. [10] Incorporating Nesterov momentum into ADAM gives rise to NADAM optimizer which allows us to take a more accurate step in the gradient direction with the momentum step before computing the gradient [11]. We train on our models using the above-mentioned optimizers and find optimal value of hyperparameters.

2.5 Training the Model

One of the most challenging aspects of Reinforcement Learning is formulating a model which performs well over many environments [3]. Our model had 3 hidden layers in addition to one input layer and an output layer. In order to check the performance, we tried to hyper-tune the model and compare various results. We trained the model with 0-4 hidden layers and found that the performance of the model with 3 and 4 hidden layers was nearly the same. As a result, 3 hidden layers were chosen over 4 hidden layers to minimize usage of computation resources. With this, in order to avoid overfitting of the model on other games, we introduced Dropout Regularization. The new regularized model had 3 hidden layers in addition to one input and one output layer.

The model was trained with 3 Agents, namely DQN, Dueling-DQN and SARSA. Initially, Adam optimizer was used with Boltzmann policy, but the results were underwhelming. We trained each RL agent for 100000 epochs due to limitations of time and computing power although letting the model train for 10 Million+ epochs would have generated even better results in terms of rewards. With this in mind, we compiled the model with different optimizers and policies, the results of which are mentioned in 2.6. Use of different optimizers and policies resulted in a much better performance in terms of rewards. Later, weights of individual agents were saved to test on unfamiliar environments.

2.6 Results

The saved weights were loaded back to test on the same game as well as other similar games. A number of episodes for testing were chosen to be 100. The model was trained on Demon Attack game and saved weights were loaded back again to test on the same game. The results in terms of rewards for a testing phase on the Demon Attack Game using various policies and optimizers can be seen in figure 6.

Fig. 5: Screenshots of testing the model on Demon Attack and Space Invaders (*Left-to-Right*)

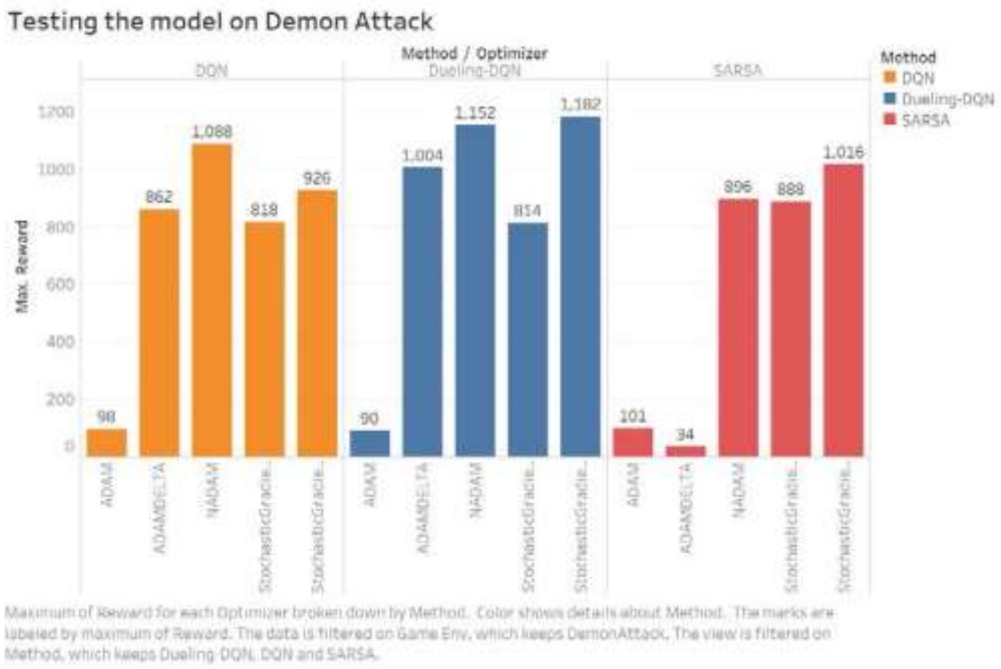


Fig. 6: Maximum rewards obtained after testing the model trained with different methods on Demon Attack

The results when tested on the same environment as it was trained on are satisfactory and it can be noted that Dueling DQN with Stochastic Gradient optimizer performs the best here. Next, we examine how this model generalizes on other games in the Experiments section below.

3. EXPERIMENTS

3.1 Transfer Learning

In this section, we focus on whether the knowledge gained by the trained model is generalizable by transferring it to different environments. For this, we chose to test the model on similar environments which have the same action space. Figure 7 below depicts our approach for testing the saved model on a new environment.

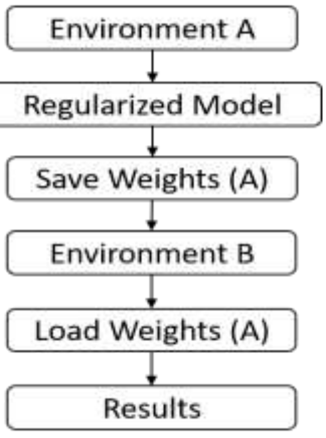


Fig. 7: Overview of the methodology for Transfer Learning

We test the model on two new environments, namely Pong and Space Invaders which possess the same action space. The results of these experiments in terms of total rewards accumulated has been illustrated in Figure 8.

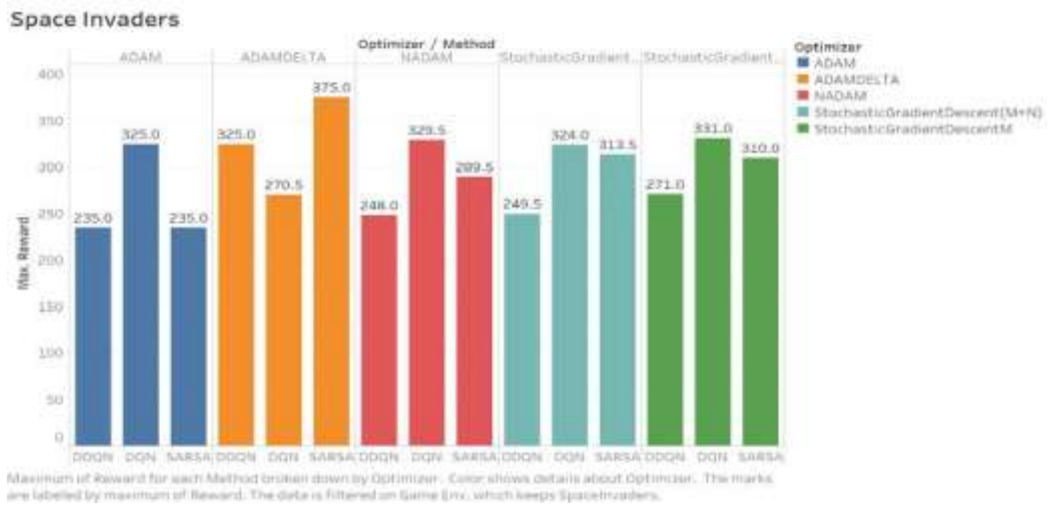


Fig. 8: Maximum rewards obtained after testing the model trained with different methods on another environment (Space Invaders)

We achieved a reasonably good score with almost all the models out of which SARSA gave the best result on the Space Invaders game. This is in contrast with testing on the Demon Attack game where the best performing agent was Dueling DQN. We further tested the findings further on Pong.

As can be seen from Figure 9, the results on Pong were not substantial. Out of all the models, Dueling DQN gave the maximum score on Pong. Though the results were average, it can be evidently seen by these results that it is possible to select a model for Transfer Learning and save computing power to get better performance through further research.

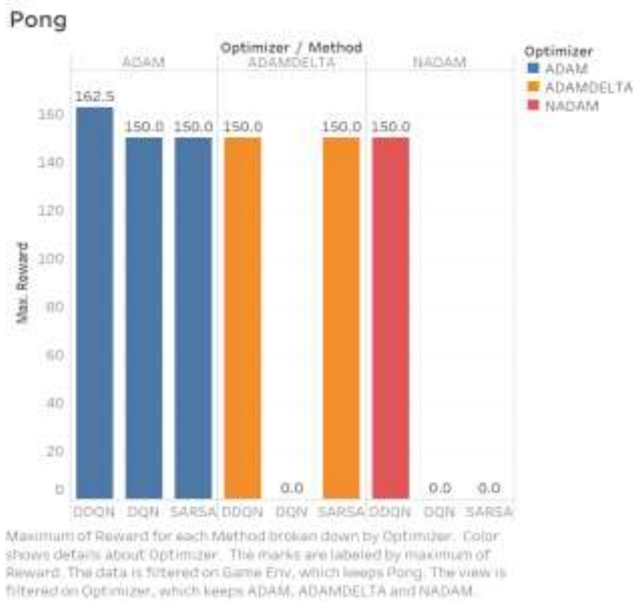


Fig. 9: Maximum rewards obtained after testing the model trained with different methods on another environment (Pong)

4. CONCLUSION

In this project, we investigated the generalization aspect of RL models in various environments with similar action spaces. The focus here was to study the effects of change in policies and optimizers as implemented by the agents in the RL model with respect to the performance of that model in same as well as different environments. We achieved optimal performance using various optimizers and policies on one game and successfully transferred the same model to other games with the same action space and achieved a reasonable reward score. One of the major reasons attributing to the mediocre score is the training time constraints. The results can be improved further by training the model for 10 million+ epochs.

To devise a generalized model which gives optimal performance on unseen environments is a challenging aspect. Introducing Dropout on the model decreased overfitting on the environment. We found that specific environments favored a specific set of parameters such that even if one set of parameters performed weakly on the same environment it was trained on, it gave good performance on other environments. In this project, we also investigated the convergence of relevant optimization algorithms and policies.

5. CONTRIBUTIONS

All the group members contributed equal time and research for this project. We divided the task of research, exploration and implementation amongst ourselves. The choice of agents, policies and optimizers specific to environments and the required hyperparameters were formulated by Abhi and Chirag. Sagar and Chirayu integrated numerous modules of keras-rl library, improving and analyzing the model outputs and visualizing the results. Together, all the group members researched, met advisors, designed experiments and talked through various implementations to create the poster as well as this report.

6. REFERENCES

[1] Volodymyr Mni, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602v1 [cs.LG] 19 Dec 2013.
[2] Mnih et al Human-level control through deep reinforcement learning Nature volume 518, pages 529–533 (26 February 2015)
[3] John Levine et al General Video Game Playing 1998 ACM Subject Classification I.2.1.
[4] Matthias Plappert keras-rl (2016) https://github.com/keras-rl/keras-rl
[5] John Schulman et al Proxy Policy Optimization Algorithms (2017) arXiv:1707.06347.
[6] Nitish Srivastava et al Dropout: A Simple Way to Prevent Neural Networks from Overfitting Journal of Machine Learning Research 15 (2014) 1929-1958.
[7] Dongbin Zhao et al Deep reinforcement learning with experience replay based on SARSA December 2016.
[8] Ziyu Wang et al Dueling Network Architectures for Deep Reinforcement Learning arXiv:1511.06581v3 [cs.LG] 5 Apr 2016.
[9] John Duchi et al Adaptive Subgradient Methods for Online Learning and Stochastic Optimization Journal of Machine Learning Research 12 (2011) 2121-2159.
[10] Matthew D. Zeiler ADADELTA: An Adaptive Learning Rate Method arXiv:1212.5701v1 [cs.LG] 22 Dec 2012.
[11] Timothy Dozat Incorporating Nesterov Momentum into Adam Workshop track - ICLR 2016.