Analysis of The Performance Deep Learning Algorithms In Video Games

Department of Computing

Letterkenny Institute of Technology

Artificial Intelligence 2

Lecturer: Shagufta Henna

By Ultan Kearns

***Abstract*—This paper aims to analyze deep learning algorithms and methodologies by training an AI(Artificial Intelligence) Agent to perform specified tasks in simulated environments. For the purpose of this paper I will be analyzing the Agent's performance in video games and will be weighing the pros and cons of each algorithm in respect to the Agent's performance. Each game has specific goals for the Agent to complete and the Agent's performance will be evaluated based on how well the Agent has performed in achieving these goals. I will then compare and contrast different learning methods and algorithms using the Agent's performance and training time as metrics.**

Key Words: AI, Deep Learning, Games

I. Introduction

A. Purpose of Paper

The purpose of this paper is to compare and contrast different deep learning methods and algorithms with respect to an AI Agent. In the upcoming sections I will be documenting the goals and objectives of each environment, the training of the agent, and comparing and contrasting different training methods. I will evaluate the Agent's performance by reviewing their actions during the game and judging if these actions were desirable or undesirable for the given environment. What I hope to achieve in the latter sections of this paper is to determine how the Agent can maximize it's performance in the given environment and to discuss the pros and cons of each algorithm used to train the agent as well as discuss how to improve an Agent's performance by tuning hyper-parameters.

B. Deep Learning

Deep learning is a subset of AI which aims to train an Agent to perform certain tasks such as image classification, speech-recognition, and natural language processing. Deep-learning differs from Machine Learning in that it uses deep layer structures to create an Artificial Neural Network[2] which can make intelligent choices based on a given state in a simulated environment. The layers of a Deep Learning model are made up of neurons / units which will have certain activation functions such as rectified linear, sigmoid, tanh & softmax. There are many methodologies within the field of deep learning which are used to train the agent and in this paper we will analyze the following: Deep Q Learning, Reinforcement Learning & Advantage Actor Critic. We will see that each one of these methodologies will have pros & cons when we evaluate the Agent's performance in each game.

II. Games

A. Cart Pole

1) About Game: In this game the Agent will control a cart which can either move left or right

within the screen's boundaries. The cart has a pole affixed to it which the Agent will try and keep balanced so that it does not pass a certain angle threshold.[6]

2) Goals & Objectives of Agent: The goal of this game is to move the cart while keeping the pole from falling, the Agent can move either left or right to sustain the pole. When the pole passes an angle of 15 degrees the game ends. The Agent's performance is evaluated by determining how long it can keep the pole from passing the 15 degree limit and ending the game. The maximum score the agent can achieve in this game is 200.

B. Kung Fu Master

1) About Game: This game is a "beat em' up" game in which the Agent can move either left or right with the options of crouching, jumping,punching and kicking, kicking and punching can also be done when the agent is jumping or crouching. The Agent will also have a health bar which will decrease with each hit they take, when the health bar reaches 0 the Agent will lose a life.[1]

2) Goals & Objectives of Agent: The goal of the agent is to defeat all the enemies on-screen and reach the end of the game. The agent will start off on floor 1 and will work their way up to floor 5, the agent must complete this goal within a time-limit and must aim to maximize it's score by defeating enemies and making it to the end of the level with as much time left as possible. The Agent will lose the game if their health bar reaches 0 and they are out of lives. The final score is determined by how many enemies our Agent defeats and how much time it has left at the end of the level, the maximum score is 1 million.

C. Breakout

1) About Game: In this game the Agent will control a paddle which can move either left or right within a confined space.[5]

2) Goals & Objectives of Agent: The goal of this game is to hit a ball using the paddle which is controlled by the Agent to break a series of blocks without letting the ball fall beyond the paddle. The Agent also has a score which it will try to maximize by breaking as many blocks as possible, as the game progresses the Agent's paddle will get smaller and the balls velocity will increase making the game harder for the Agent. The Agent will have three lives to break all the blocks in the environment, the max possible score in Breakout is 896 which we will use as a metric when judging the Agent's performance.

III. Algorithms Used for Deep Learning

A. Deep Q Learning(DQN)

Traditional Q Learning works by maintaining a table with each state, this method works fine for environments with a small number of states however when working with a higher number of states we opt to use Deep Q Learning.
DQN differs from traditional Q Learning by using an Artificial Neural Network(ANN) which has the advantage of reducing the amount of memory utilized as DQN learns over time(iterations) and discovers commonalities between states.
This means that DQN doesn't require having to store every state the agent could be in memory, however the model does take time to train and increasing the number of iterations will increase the time it will take to train.
DQN utilizes an optimal action-value function defined by the following equation:

$$Q*(s,a) = max_{\pi}\epsilon[r_t + \gamma r_t + 1 + r_t + 2.....]$$

[8]

- Where Q is our agent
- S is our state
- A is the action
- r is the reward
- t is the time step
- π is our policy which the agent will use in making decisions
- and γ is our discount value

essentially we are approximating the best course of action for our Agent's given state, in using this function we can approximate the best course of action for our Agent to take given it's current state.

We also use a loss function defined by the following equation for each iteration:

$L_i(\theta_i) = \epsilon_(s, a, r, s')$
(r+$\gamma * max'_a Q(s', a'; \theta_i) - Q(s, a; \theta_i))^2$

- Where L is our loss
- i is given iteration
- s is our state
- a is the action we are taking
- r is our reward
- γ is our discount factor
- Q is our agent
- θ_i is our given iteration
- s' is the next state we wish to get to
- and a' is the next action we wish to take

[8] Using this loss function we calculate the output of our Agent compared to the actual target value so we can determine how well our agent is performing at a given iteration.

B. Asynchronous Advantage Actor-Critic A3C

The Asynchronous Advantage Actor Critic algorithm utilizes a Deep Neural Network(DNN) to train the Agent to perform certain tasks using Reinforcement Learning[4]. Essentially this algorithm works by implementing an Actor(our Agent) and a Critic which will be used to evaluate the Agent's performance in the given environment. It is in this way that the Agent can learn through Reinforcement Learning as the Actor will try to maximize it's score and the critic will use value based functions to determine if the Actor could be performing better. To calculate the evaluation function we use the following equation:

$$J(\theta) = E_\pi[\sum_{t=0}^{\infty} A_\theta \theta_u(s_t, a_t)$$
$$\log_{\pi_\theta}(a_t|s_t) + \beta H_\theta(\pi(s_t))$$

[4] The section of the equation '$H_\theta(\pi(s_t)$' is utilized to encourage agent exploration and is an entropy term.

This algorithm trains multiple Agents in parallel on multiple environments each with their own weights, it is in this way that the Agent can experience more states and actions in less time. The results of these Agents are then combined and used as inputs to train a single Agent or what is termed "Global Network"[3]. Using a neural network we can train the Agents to approximate the best state & the best actions when playing the game.

Our Agent will be trained using a Deep Neural Network[7] so that it can approximate the best action to take for a given state.

C. Reinforcement Learning

The premise of value-based reinforcement learning is as follows:

- The Agent takes an action(A) for a given State(S)
- The performance of the agent is then evaluated at which point we have two options:
  1) "Reward" the Agent if the action taken is beneficial to them
  2) "Punish" the Agent if the action taken is undesirable

The way we "reward" or "punish" an agent can be done in a variety of ways usually by utilizing a cost function which will either increase an Agent's score for a desirable action or punish them for an undesirable one. For our purposes we will be using the following equation for our Agent to determine the reward

$$Q(s_t, a_t) \leftarrow r_t + \gamma \cdot \max_a Q(s_{t+1}, a)$$

- Where Q is our agent
- S_t is our given state
- a is the action we wish to take
- r is our reward
- γ is our discount function
- and $\max_a Q(s_{t+1}, a)$ is the action which will give us the maximum return for our given state

There is also another variant of reinforcement learning which utilizes policies to determine which action to take next. The basic premise of policy based reinforcement learning is as follows:

- The Agent Performs an action(A) for a given State(S)

- The Agent compares the rewards for certain moves as compared to others during training
- The Agent then uses induction to determine which actions lead to the greatest rewards
- A policy is established for given scenarios when the Agent is in certain states

In this scenario an agent is trained on a number of iterations and adopts certain policies which the Agent has found yield the optimal reward, for example in a game the agent may adopt a policy to always go left at intersections or the Agent may jump when the enemy tries to kick. The Agent's choice of action for a given state may appear nonsensical and counter-intuitive to humans but the Agent has found those actions yield the optimal results during training, this is a drawback to policy based reinforcement learning, the best action is not always taken by the Agent for a given state.
For the purposes of this paper I will be using value based Reinforcement Learning when training the Agent.

IV. Results of Algorithms

A. DQN For Cartpole

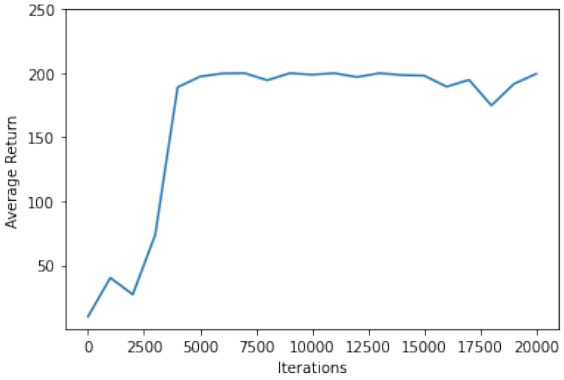
When using DQN for the CartPole game I found that the Agent's Average Return was yielding the maximum for our given hyper-parameters as the max result you can get is 200 per episode. Knowing this I decided to try to reduce the amount of time to train the model by adjusting the hyper parameters.

1) Adjusting Hyper-Parameters: The original model utilized the following hyper-parameters:

- num iterations = 20000
- initial collect steps = 1000
- collect steps per iteration = 1
- replay buffer max length = 100000
- batch size = 64
- learning rate = 1e-3
- log interval = 200
- num eval episodes = 10
- eval interval = 1000

Which took approximately five minutes to train, I have included an imagine of the average return below:

Fig. 1. Original Model

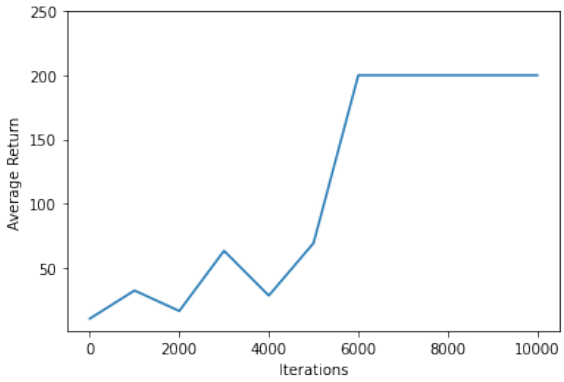


Adjusting the hyper-parameters I was able to get training down to approximately 2 minutes, I will list the hyper-parameters used for training this model below

- num iterations = 10000
- initial collect steps = 10000
- collect steps per iteration = 1
- replay buffer max length = 1000000
- batch size = 64
- learning rate = 2
- log interval = 200
- num eval episodes = 1
- eval interval = 1000

By adjusting the number of iterations, the initial collect steps, the replay buffer max length, the evaluation episodes and the learning rate parameters I was able to get the model to execute in about 2 minutes or to be exact 146 seconds. This is a big improvement from the 5 minutes it took previously to train. I will include a screen shot of the average reward graph below:

Fig. 2. Second Model



2) Final Results: I succeeded in reducing the time needed to train this model as the original model was achieving the highest score possible for the average reward. The new model implementation used a higher learning rate and a reduced number of evaluation episodes which I believe were the most significant variables to be adjusted to decrease the model's training time.

B. A3C Kung Fu Master

Kung Fu Master is in many ways a more advanced game than the previously discussed Cart-Pole game in that it has much more moves for the agent to take, and is more challenging as it has multiple enemies which appear on screen trying to defeat our agent.
Initially when starting the notebook I found the rewards were hovering around the 1.5 mark with the first model's implementation. I set about adding and removing layers to determine which configuration will yield the best results I will include the findings in the adjusting hyper-parameters setting.

1) Adjusting Hyper-Parameters: When adjusting the hyper parameters I found it necessary to increase the amount of neurons in each of the layers. I initially decided to use 3 layers each with 256 neurons and used a Sigmoidal activation function for the final layer. When running the agent I was able to achieve a score of 900 which was almost double the score of the original model but on further evaluation of the session rewards I found the model did indeed perform worse.

First model parameters

- 3 layers of 256 neurons using Relu functions the first having a stride of 3 the final 2 layers had a stride of 2
- The final layer was 256 neurons dense and used a Sigmoidal activation function
- Number of games: 3
- the strides of all layers are equal to 1, trying to increase the number of strides led to a decrease in performance for training.
- Number of Iterations: 100000

We can see below that the first model isn't performing as well as the original at this point I decided to try to switch the Sigmoidal function used for the output layer to Relu, we can see from the results of the second model this had a much better effect on the overall performance of the Agent.

First model results

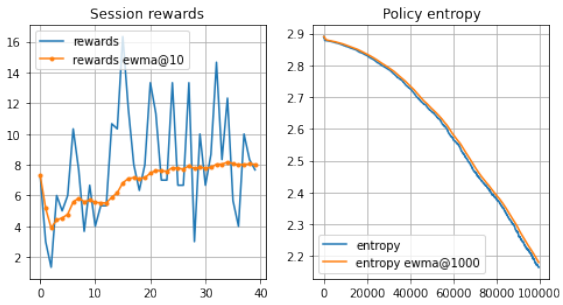
Fig. 3. First Model A3C

Second model parameters

- For this model I had used 3 layers of 64 neurons with Relu activation with the output layer having the same number of units and a Softmax activation function.
- Number of games: 3
- the strides of all layers are equal to 1
- Number of Iterations: 100000
- Gamma 0.99

Second model results

Fig. 4. Second Model A3C



The trained model seems to have a high level of entropy but a very high level of reward, I believe this trade-off is worth it as the agent seems to be performing a lot better than in the other two models, earning up to 1000 points after 3 games.

2) Final Results: The final implementation had a far higher average reward compared to the first even though it had twice the entropy. The agent also performed much better using the second model's hyper-parameters and was able to achieve a score up to 1000 points.

C. Reinforcement-Learning Breakout

Breakout is a fairly simplistic game compared to Kung-Fu Master but is far more sophisticated than Cart-Pole in that there are more considerations for an agent to make during the game. For example in Cart-Pole the Agent just had to keep track of the angle of the pole and prevent it from exceeding a certain value. In Breakout however we find that the agent must keep track of the ball which adds new variables in such as the trajectory of the ball, the velocity of the ball, the position of the paddle and the angle to hit the ball so that it will yield the optimal results.

1) Adjusting Hyper-Parameters: The original model had the following hyper-parameters for the agent:

- Four dense layers of 1024 units for the agent using a relu activation function
- replay memory size of 200000 bytes
- Learning rate of $1e$-3 and a batch size of 128 for the optimization function
- discount factor of 0.97

The first model produced a mean reward per episode of 2.4 when running 30 episodes.

For the second model I decided to increase the replay memory size and the number of units per layer below are the following hyper parameters I used:

- Increased the replay memory size to 500,000 bytes
- Halved the amount of units per layer to 512 units

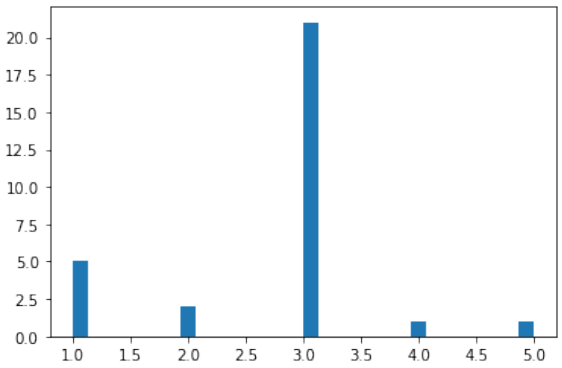
The above model performed poorly with a mean reward per episode of 1.7

The final model utilized the following parameters and achieved a mean average per episode of 2.7, the max score achieved by the model was 5.0.

- A replay memory of 350000 bytes
- The first two layer had 512 units with RELU activation and the remaining two layers utilize 1024 neurons with Relu all strides set to 1 in each layer
- and a discount factor of 0.14

2) Final Results: Achieved a final mean reward of 2.7 which is a small improvement from the original model.

Fig. 5. Histogram of Reinforcement Learning results



D. Pros & Cons

1) DQN: Pros of this algorithm:

- The algorithm doesn't require every state to be mapped before hand freeing up memory and time.
- Utilizing a DQN we can make generalizations of which actions will yield the highest reward.

- Comparatively fast compared to other algorithms

Cons of this algorithm:

- This algorithm is not strongly suited to games which require extensive planning[8]
- Depending on the performance you are trying to achieve for the Agent it can take long time to train, for cart-pole this wasn't the case but for other tasks DQN may not be suitable.

2) A3C: Pros of this algorithm:

- Utilizing a Deep Neural Network we can choose multiple activation functions such as RELU, TANH, Softmax, Sigmoidal etc.
- DNNs are scalable, we can increase the number of layers, amount of neurons used for each layer and can optimize the DNN by tweaking these.
- The Agent will be run in multiple environments, the outputs of the Agent's performance for each environment will then be used to train the final model. It is in this way we optimize the Agent's performance.

Cons of this algorithm:

- Can be slow to train models depending on the number of layers and the size.
- Adjusting hyper-parameters can be difficult as there is no "right" answer, it's a trial and error process

3) Reinforcment-Learning: Pros of this algorithm:

- Can solve some problems which are not currently solvable using other techniques
- It is very close to the way a human learns eg: by doing, making mistakes, and learning from those mistakes - this ties in very closely with the field of explainable AI
- Learns by performing tasks over and over again to establish which actions will yield the best rewards

Cons of this algorithm:

- Can take a long-time to train due to the Agent having to run through many iterations of the game to learn which actions will yield rewards and which actions are detrimental.
- Policy based reinforcement learning can lead to poor decisions in some cases
- Policy based reinforcement learning may not be suitable in some use cases - say a game where the Agent faces off against a human, the human could figure out which policy the Agent uses for determining the best move and use it to their advantage.
- Agent can wind up doing the same moves over and over again unless action is taken to prevent this

V. Conclusion

In this section I will show my findings and offer recommendations on how to improve each model.

A. Improvements in Models

I was able to achieve an increase in performance for all models I will summarize how I improved them below:

- For CartPole I was able to reduce the amount of time it took to train the model by tweaking the hyper-parameters, the original model took approximately 5 minutes to train I was able to get the training time down to 146 seconds which is a little over 2 minutes.
- For Kung-Fu-Master I was able to increase the average reward from 2 to 8 which shows the agent is performing much better than before with a slightly higher level of entropy. I achieved this by increasing the units per layer and mixing three relu layers with a softmax output layer.
- For Breakout I was able to achieve a slight improvement of 0.3 by adding more units per layer and increasing the replay memory and discount factor.

B. Future Recommendations

- For Cartpole the model is performing optimally, as the max score for this game is 200 per episode and my model achieved a final score of 200. By tweaking the hyper-parameters a bit more and adjusting them the model could perhaps be trained in less time. I didn't personally think this was worth-while as 146 seconds is not a long-time to wait for optimal results.

- For A3C adding more layers and more units per layer would increase performance, I didn't implement this as with Google Colab's limitations the Agent was taking a long time to train. However if Google Colab offered more resources I would have added more layers and units. Also adjusting the gamma value may improve the model a bit
- For Deep-reinforcement learning with breakout I ran into some limitations with memory. If there were more resources available I would've increased the number of units per layer and added a few more layers and increased the replay-memory value and adjusted the discount value to determine which configuration would be optimal.

C. In Closing

In closing from my research I have found that each of these algorithms had performed well in their given task. Although some of these models could be improved although there is a trade-off, models can take a long time to train so if a model is performing reasonably well then sometimes it's better to leave it rather than to spend a lot of time tweaking hyper-parameters for a slightly better reward.
I have learned from this research project that even the 'best trained' models can be improved, and given the amount of research and innovation in the field of Artificial Intelligence there is always a better way to train an agent to adapt to an environment.

REFERENCES

[1] Wagner Cardoso. *Kung Fu Master - Atari 2600 (761.520 pontos)*. URL: https://www.youtube.com/watch?v=uBJNpvOyo8U.

[2] IBM Cloud Education. *What are Neural Networks?* URL: https://www.ibm.com/cloud/learn/neural-networks.

[3] Sergios Karagiannakos. *The idea behind Actor-Critics and how A2C and A3C improve them*. URL: https://theaisummer.com/Actor_critics/.

[4] Multiple. *Asynchronous Methods for Deep Reinforcement Learning*. URL: https://arxiv.org/pdf/1602.01783.pdf.

[5] Multiple. *ATARI BREAKOUT: THE BEST VIDEO GAME OF ALL TIME?* URL: https://spectrum.ieee.org/atari-breakout.

[6] Multiple. *CartPole-v1*. URL: https://gym.openai.com/envs/CartPole-v1/.

[7] Multiple. *Data Mining (Fourth Edition)*. URL: https://www.sciencedirect.com/topics/computer-science/deep-neural-network.

[8] Multiple. *Human-Level Control Through Deep Reinforcement Learning*. URL: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf.

LIST OF FIGURES

1 Original Model 4
2 Second Model 5
3 First Model A3C 5
4 Second Model A3C 6
5 Histogram of Reinforcement Learning results 6