Machine Learning - Final Project Report

Comparing Reinforcement Learning Techniques (ACKTR & A2C) on Atari Game

Vasu Verma
2014115
BTech CSE
vasu14115@iiitd.ac.in

Vishisht Khilariwal
2014120
BTech CSE
vishisht14120@iiitd.ac.in

INTRODUCTION

Motivation

Recently OpenAI released a bot for Dota2 which defeated the world's best 1v1 mid players. The bot works on ACKTR (Actor-critic methods using Kronecker-Trust Region). Along with this new baseline, OpenAI also released A2C - modification of their state-of-art method-A3C. These research papers were released in August, 2017. Since it is a very recent feat, we decided to implement their technique and compare it with other techniques that they used earlier.

Problem Statement

Maximizing your score in an Arcade Game, and comparing the performance of ACKTR and A2C(Reinforcement learning techniques). We will be using the game StarGunner for our project. The scores of 3 algorithms - ACKTR, A2C and DQN will be compared and the results justified. We also analyse the working of ACKTR and A2C to learn the working of these algorithms and why they perform better than previous algorithms such as DQN.

Deliverables

The deliverables will be as follows:-

- Codes to train a model for both ACKTR and A2C using OpenAI baselines[5]. (run_acktr.py and run_a2c.py)
- Trained models for the both algorithms and a python code to run them (analysis.py)
- Analysis and comparison of ACKTR and A2C performance as well as comparing the accuracies with standard DQN algorithm.
- Graphs showing rewards vs timeframes during training the models and rewards obtained after successfully running the saved models.

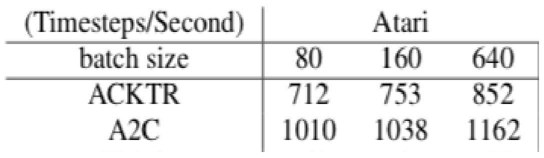
RELATED WORK

State of the art:-

The current state of the art algorithm is A3C. OpenAI's gym leaderboard for the game StarGunner[4] shows the highest score ever obtained for the game is 93480.00 with an offset of 2169.12. Modified A3C algorithm was used to obtain the score. ACKTR was supposed to be an upgrade over standard A3C, and hence we will keep that score as a benchmark. However, we do realise that the modified A3C has various non standard optimisations that our standard ACKTR might not be able to overcome. Our objective is to not only show how A2C and ACKTR perform against each other, but also to compare how well our implementations (using OpenAI baselines) perform against standard DQN and this score as well.

Research Papers

Reinforcement Learning using Kronecker-factored approximation[1]. The paper explains the working of ACKTR algorithm and how it makes it's optimisations which have been discussed later in this paper.
The following results are average timesteps per second over six Atari games. This table gives a comparison between A2C and ACKTR algorithms.



(Timesteps/Second)	Atari		
batch size	80	160	640
ACKTR	712	753	852
A2C	1010	1038	1162

Human-level control through deep reinforcement learning(DQN)[2] discusses the algorithm used in DQN and evaluates it on Atari games 30 times for 5 mins each.

Asynchronous Methods for Deep Reinforcement Learning[3] discusses Asynchronous Advantage Actor-Critic method(A3C). A3C runs simultaneously many actors on many environments. After one update in policy occurs, it changes the policies of all actors in all environments. A2C is a synchronous version of A3C where only one actor plays in one environment at one time and the weights are updated before running the next actor. $Advantage = R - V(s)$, where R is the discounted reward and V(s) is the value reward to stay in a state.

DATASET AND EVALUATION

Dataset

Dataset are RGB frames of the game with 210X160 pixels with a 128 color palette. The frames are then resized to 84X84 after extracting luminance.Number frames used lie in the range of 1x10^6 to 1x10^7.[2]

Evaluation

Performance will be evaluated for every learning technique using parameters like rewards during training, average score achieved by the trained models and time taken to train the model for a given algorithm which will give us an overall view of how these algorithms fare against each other and what are the drawbacks of using one over another.

METHODOLOGY

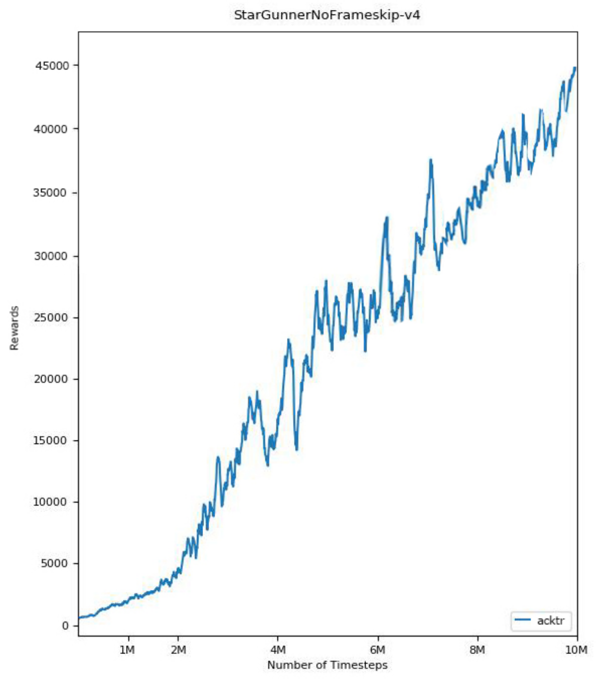
We studied DQN, ACKTR and A2C algorithms and their OpenAI codes to implement training and running of these models using baselines and ikostrikov/pytorch. We use pytorch because of it is more convenient to use while dealing with OpenAI models as compared to numpy or pandas.

Dqn

We ran the custom DQN from OpenAI baselines to train a model for 10 million timesteps (same as we will do for ACKTR and A2C). However it converged after a few timesteps. The time taken for DQN model to train was much lower as compared to ACKTR and A2C. Even though the DQN model trained much faster than the other two, it's rewards were nowhere close to what we obtained in the cases of ACKTR and A2C.

Acktr

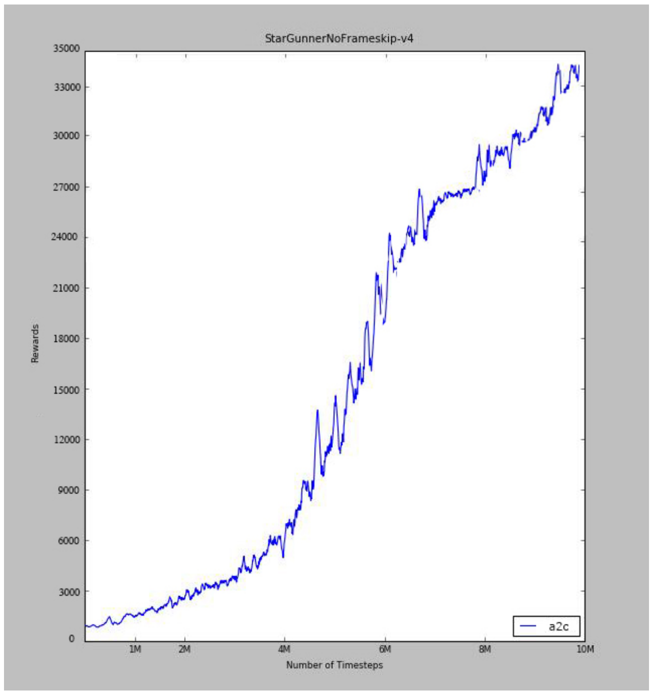
ACKTR uses Kronecker-factored approximation to modify the gradient descent and thus changes the Fisher matrix produced. The model was trained on 1M timesteps and the graph was served on a localhost using a Visdom server. The graph obtained from training the ACKTR model is shown below:



The training data graph for ACKTR shows that it increases the rewards steeply as the number of timesteps increase i.e. consecutive episodes are evaluated. This shows how using natural gradient along with Kronecker optimisations leads to a better increase in rewards as compared to Stochastic gradient descent. Further explanation to Kronecker optimisations will be discussed in the next section. The training was done on a normal CPU. The time taken to complete the trainings is an important metric as well and hence it was recorded as well. It was expected of ACKTR to take longer time as compared to DQN due to higher complexities of the training model and because DQN started to converge after a few timesteps. Also ACKTR ran faster than A2C because of the synchronous nature of training agents involved in A2C.

A2c

A2C is a modified/synchronous version of A3C. A2C waits for each actor to finish its segment of a fixed number of timesteps (example 20) and then performs an update over all the actors. This improves the performance of A2C over A3C as it can make a better use of the GPU.
As we can see from the below graph that the rewards of StarGunner increase at an almost constant rate, but the rate of increase in reward is relatively slower than ACKTR.



RESULT AND ANALYSIS

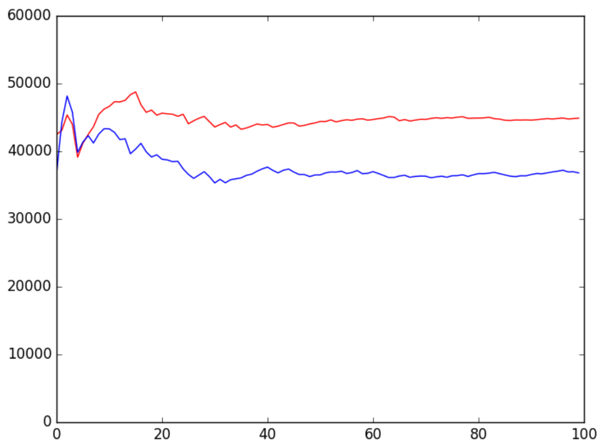
Score Obtained

Score Obtained

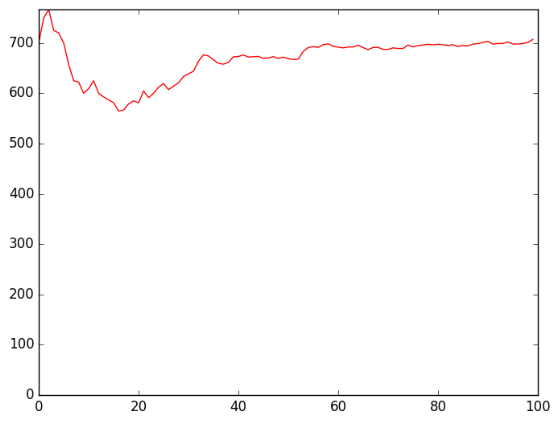
The returned models were run with analyis.py to test out their average score and compare them with each other. Below are the evaluation metrics for the two models in terms of average 100 episode rewards and max reward obtained during training are given below:-

- ACKTR
    - Max Reward :- 60,000
    - Average 100 episode reward:- 44894
- A2C
    - Max Reward :- 55,920
    - Average 100 episode reward:- 36804
- DQN
    - Max Reward :- 1100
    - Average Reward :- 698.51

The graph for the same is shown below:



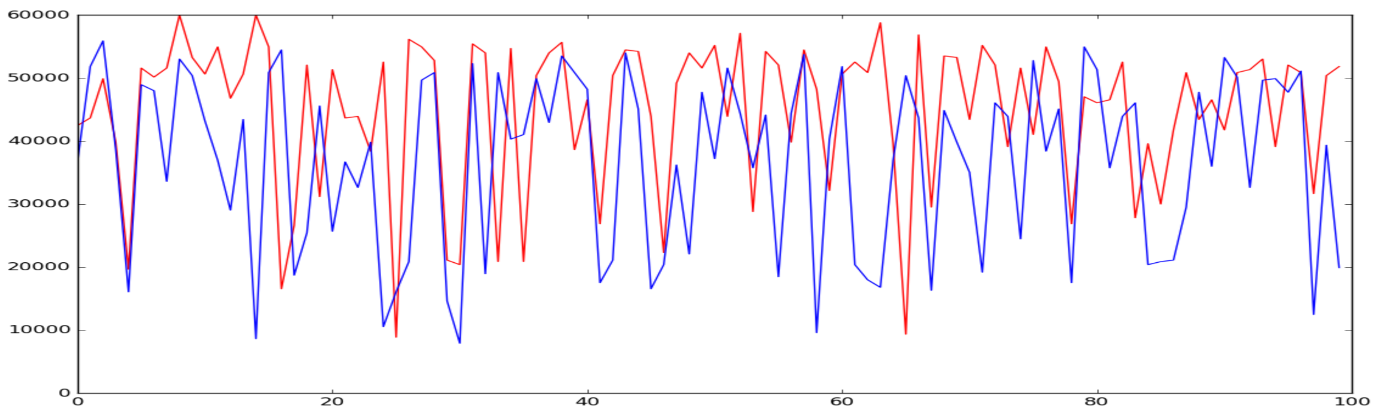
The red and blue line graphs represent ACKTR and A2C average episode score. We can clearly observe that ACKTR is superior to A2C. However, we also analysed the individual score received in each episode for both models to check if ACKTR performs better than A2C in all of them. The graph on the next page shows that A2C might perform better in some episodes as compared to ACKTR, however ACKTR performs much better in performance consistently.



The above graph represents the mean rewards for 100 episodes. Our model converged at around 680 score due to some limitations of using SGD with StarGunner game.

Time To Train

Time taken to train DQN was only a few hours as it converged before 10 million timesteps. ACKTR model took 28 hours 15 minutes to train for complete 10 million timesteps where as A2C took 20 hours 38 minutes to complete. This verifies from the paper[1] that ACKTR has



a lower timesteps/second because of the additional changes to the gradient descent which optimises the Fisher information matrix (KFAC).

Analysis

The performance of these models can be explained by the working of these algorithms

DQN

The research on ACKTR[1] also states that DQN and other algorithms that use SGD for policy descent/updation sometimes converge to a near deterministic policy and return it in their baseline as well.

ACKTR

ACKTR uses Kronecker-factored optimisations to the stochastic gradient descent used for policy updation. $W \leftarrow W - \alpha G$. It uses a natural gradient descent by whitening the received gradient. $W \leftarrow W - (BB^*)^{-1} G (AA^*)^{-1}$. $(AA^*)^{-1}$ is the activation whitening matrix and $(BB^*)^{-1}$ is the backdrop whitening matrix. The values of A & B are such to reduce the covariance of the Fisher Information matrix to 1. This led to a better policy updation algorithm as it increased the reward increase per episode.

A2C

A2C has a global network and many different agents each with an individual environment. Each of the agent and environment is independent of each other, providing a more diverse training experience. Each A2C agent estimates the value function as well as the policy loss and then gets the gradient from the losses. But unlike A3C, A2C updates the global network with the gradients synchronously.

CONTRIBUTION

Following were the deliverables promised in the project proposal:-

- Files to train acktr and a2c models - **Delivered**
- Python code for analysis.py - **Delivered**
- Comparison of performance (score and time) of ACKTR, A2C and DQN - **Delivered**
- Appropriate graphs for all testing/training - **Delivered**

Both members equally contributed in research, coding and analysis part of this project. In context of deliverables, the individual contributions are given below:-

Vasu Verma:

- Researched on ACKTR and DQN algorithms
- Trained the ACKTR model
- Analysed results and compared ACKTR and DQN.
- Graphs after running analysis.py which represent rewards of trained model.
- Files
  - Run_acktr.py file
  - Analysis.py file
  - Trained models/acktr

Vishisht Khilariwal

- Researched on ACKTR and DQN algorithms
- Trained the ACKTR model
- Analysed results and compared A2C and DQN.
- Researched on A3C model for explanation of shortcomings
- Files
  - Run_a2c.py
  - Modified enjoy.py and train.py for DQN.
  - Trained models/a2c

References

1. https://arxiv.org/pdf/1708.05144.pdf
2. https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
3. HTTPS://ARXIV.ORG/PDF/1602.01783.PDF
4. https://gym.openai.com/envs/StarGunner-v0/
5. https://github.com/openai/baselines OpenAI baselines
6. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr/blob/master/enjoy.py