Deep Q-Networks and Transfer Learning

Abstract

This is an exploratory project to investigate the usefulness of transfer learning in the context of Deep Reinforcement Learning. We are using OpenAI Gym's classic control environments as a tool. Intuition is that knowledge acquired in one environment should aid Agents learning in different environment with similar objectives. We are using two related environments requiring agent to acquire similar knowledge to maximize rewards. Specifically, we are using OpenAI Gym's Acrobot-v1 and MountainCar-v0 environments, both of which need agent to gain knowledge of momentum. We are using DQN, a simple 3-layer fully connected network (for estimating Q-value) and transferring pre-trained weights from MountainCar-v0 to Acrobot-v1. We explore two techniques, information extraction from weights and changes in architecture. We observed that although there are only minute gains in average reward, the training becomes more stable. Training time stability may be crucial for agents acting in real-world and could decrease side-effects of exploration.

1. Introduction

Reinforcement learning is about training agents to take a sequence of decisions in an environments where the goal is to maximize the reward determined by a utility function. Deep Reinforcement Learning elegantly combines the best of Deep Learning and Reinforcement Learning, where learnable parameters in the RL framework use deep neural networks. Although it is very effective, it can be brittle and requires large amount of data to work efficiently. Humans, on the other hand, have an innate ability to use the knowledge acquired in one domain and apply it to other domains. We don't learn everything from scratch and, rather, apply knowledge gained in past to learn novel tasks. An elaboration of human priors in solving a simple game is investigated in the paper [1]. A simple example is learning car driving is easier for humans because we have some knowledge about object shapes and have the idea to follow the road. Transfer learning is significant in Deep Learning as it works on the same principle. A pre-trained model is used as a starting point (or sometimes frozen) for another task. It allows us to overcome the issue of collecting large data sets especially for niche problems where data set is scarce. It also reduces computational power, time required to train a model and stabilises trainning.

In this project, we are training agents for CartPole-v1[2], Acrobot-v1[3] and MountainCar-v0[4]. All the environments are part of OpenAI gym. Idea is to transfer the pre-trained weights for Q-value approximation from MountainCar-v0 to Acrobat-v1 to deduce if the model can adapt to a different game. Intuitively, knowing to work with one problem should help ease the training for other problem. We are also exploring the changes required in architecture to improve computation efficiency or reduce the number of epochs.

2. Background/Related Work

In the recent years, Transfer Learning has played key role in advancements in CV and NLP. We explore the same idea in context of Deep RL. In "Playing Atari with Deep Reinforcement learning" paper by DeepMind [5], they developed a single DQN which learned to play different Atari games. The model has both convolutional and fully-connected layers. A variant of Q-learning with a experience replay and target network is used for the training of model. The experience replay removes correlation in the observations and prevent the network from only learning about what it is immediately doing in the environment which in turn allows it to learn from knowledge gained in past. Target network helps reduce oscillations which in turn stabilizes the training process. Input of model is raw image pixels of a game and output is Q value. An algorithm to achieve human-level control with the use of deep reinforcement learning is explored in the paper [6]. We are using a tailored implementation of Deep Q-learning as explained in aforementioned papers which is suitable for our environments.

A method named as "Actor-Mimic" is described in paper[7] uses deep RL to train a single policy network that learns how to behave in a set of different tasks. There are various publications which explored the idea of transfer learning in context of deep RL such as [8], [9], [10] and [11] which we are using as a reference to implement our project. However, environments such as Open AI classic control are

less explored in context of transfer learning using deep RL. We are trying different methods of transferring weights with DQN and measuring the results on two such environments.

3. Environment Specifications

We are using three Classic Control environments from OpenAI Gym library .

1. **CartPole-v1** is a 4 state environment with Discrete Action Space of two - 0 (Push cart to left) and 1 (Push Cart to right). In this game, a pole is attached to a cart, which moves along a track. Application of force with value +1 or -1 controls the system. The goal is to prevent pole from falling over. Pole should stay upright. A reward of +1 is provided for every time step when the pole stays upright. Initially, all observations are assigned a uniform random value in -0.05 to 0.05. Environment terminates, if Pole Angle is more than 12 degrees / center of the cart reaches the edge of the display / Episode length is greater than 200. Environment is considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive episodes.

2. **Acrobot-v1** is a 6 state environment with Discrete action space of three - +1, 0 or -1 torque. Acrobot environment has two joints and two links. Only the second joint (between two links) is actuated. Links are hanging downwards in the beginning. Swinging the end of lower link to a given height is the goal. Reward is -1 for each time step until termination. Environment terminates when agent reaches goal / episode length is greater than 500.

3. **MountainCart-v0** is continuous state environment with discrete action space of three - +1, 0, -1 torque. A car is positioned in a valley between two "mountains". The goal is to drive up the mountain on the right. The only way to achieve this goal is to drive back and forth to build up sufficient momentum. Reward is -1 for each time step, until termination. No penalty for climbing left hill. Initially all observations are assigned a uniform random value in -0.6 to -0.4 with no velocity. Environment terminates when position 0.5 (top of hill) is achieved / Episode length is greater than 200. Environment is considered solved when average reward has a value of -110.

4. **Comparison of environments** CartPole-v1 is inherently a balancing problem. On the other hand, Acrobot-v1 and MountainCar-v0 are momentum gain problem. Thus transfer learning between CartPole-v1 and the other two environments may or may not work. Although, CartPole-v1 can be used to verify the intuition, as of now we will be using MountainCar-v0 and Acrobot-v1 for transfer learning experimentation. Objective of both the games is similar that is to gain enough momentum to reach a given position.

4. Approach

In our project, we are using Deep Q-Learning, a variant of Q-learning involving neural networks to predict Q-value function for future discounted rewards of two environments. The agents are trained independently for baseline comparison. We then use the learned weights of MountainCar-v0 to train the Acrobat-v1 and evaluate if the model approaches convergence faster than baseline implementation. The project can be divided into 3 problems - DQN Implementation, Neural Network Implementation and Transfer Learning experimentation. The approaches are explained in detail in the following subsections:

4.1. DQN Architecture

Q(s, a) is the action-value function which describes how good it is to take an action a from any particular state s. These values are stored in the table. Once we have these Q(s, a) values, the best action to take from a given state will be the one that has maximum value of Q(s, a). If the state-action space is too large then computation and memory requirement for Q-learning will be very high. To address the issue, neural networks are being used to approximate Q(s, a). The learning algorithm is known as Deep Q-learning. Traditional Q-learning approach was non-parametric that is to remember the Q(s, a) values. DQN generalizes the approximation of Q-value function. Given a state, the neural network is trained to predict a set of continuous values, each representing scores for every state action pair. Based on the prediction, we choose the optimal action to maximize the total rewards. The agent takes the final action depending on the value of a hyper-parameter epsilon which promotes exploration in early stages and decays slowly to allow network to choose optimal solutions. This approach for choosing optimal solution is called Epsilon-Greedy. We generate a random number and based on the number we make an exploration-exploitation trade-off.

Below equation is used to update the Q-values [12]

$$q(s,a) = q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q(s',a') - q(s,a)) \quad (1)$$

4.2. Transfer Learning

Transfer learning exploits the fact that during training the information learned by agent has two components. It learns the features specific to environment and it also learns general features which can be used as starting point for the training of another environment with similar objectives. To evaluate feasibility of transfer learning, we are broadly using two approaches - information extraction from weights and changes in architecture. This will include:

1. Fine tune the pre-trained model.
2. Retaining pre-trained layers that contribute to convergence.
3. Randomly reinitialize other layers for environment specific learning. Overriding layers that do not contribute to training time improvement.
4. Add extra layers on top of current ones to aid environment specific learning.
5. Extract information from weight matrix and try to inject the same in the network.

5. Baseline Implementation

We use a basic architecture to quantify effects of transfer learning. Below are the details of results of baseline implementation:

5.1. Acrobot-v1

Model is trained for 5000 episodes and average reward is -82.40.

6. Transfer Learning Experiments

We implemented baseline for two games Acrobot-v1 and MountainCar-v0, and tried transfer learning between Acrobot-v1 and MountainCar-v0. As the two environments (MountainCar-v0 and Acrobot-v1) need similar learning to solve the environment. If the agent has learned to solve one environment, it has fundamentally learned some idea closely related to momentum as there is nothing else that would help maximize the reward. The intuition is regardless of the other environment specific dynamics (like gravity, velocity, force) the agent should capture the idea of momentum if it has to solve the environment. The only difference in learning between the two environments is that agent needs to learn a sense of direction for MountainCar-v0 in addition to momentum as the environment can only be solved moving forward. Also, both the environments have similar action space which makes transferring knowledge more comparable. Most of the experiments we performed revolved around transferring the weights from MountainCar-v0 to Acrobot-v1. As there would have been uncertainty around direction in case of Acrobot-v1 to MountainCar-v0.
The problem in transferring knowledge across these two environments boils down to differences in state space and environment specifics. To overcome this, and test our hypothesis, we did the several experiments as explained in next section.

6.1. Weight Extraction

We extracted the parameters for each environment using a different "over-trained" network to capture detailed information from each environment. This networks is not used in the experiment apart from initial weight extraction. We trained the agent for 50,000 epochs with 4 layered fully connected network to distribute learning across various layers with the idea to use these different layers for transfer learning. This included subtle changes in the environment - MountainCar-v0 (no cap on rewards and making the environment unsolvable). No changes to default Acrobot-v1 configuration is done. The network is same for both environments apart from initial layer where 2×72 is used for MountainCar-v0 and 6×72 for Acrobot-v1

Layer	Activation	Size
fc1	tanh	$2 \times 72, 6 \times 72$
fc2	tanh	72 x 144
fc3	tanh	144 x 192
fc4	Softmax	192 x 3

Table 1. Network Architecture for weights extraction

6.2. Experiments from MountainCar-v0 to Acrobot-v1

To quantify the results, we plotted following graphs. First, weight transfer over consistent architecture to see if rewards increase and/or epoch count decrease for a certain reward value. Second, changing the architecture to see if lesser number of parameters can perform comparable to baseline.

6.2.1 Information Extraction from Weights

We started by extracting weights of last two layers from network trained on MountainCar environment and directly used them on Acrobot. Agent did not solve the environment in this case because it had no knowledge of local fixed environment specifications like friction, force, torque. We then picked the last layer of the trained network of MountainCar-v0, applied some basic operations (flattening, reshaping) to match the state space of Acrobot (6,). We used these weights to initialize the first layer of the Acrobot's network. The experiment, however, did not show any significant improvement over the baseline as correlation between state and action might have been lost due to reshaping. Further, we tried a set of experiments aimed at extracting general features (non-environment-specific information) and covering differences in state from the last layer of the learned weights of MountainCar-v0.

The experiments performed show a significant reduction in noise and oscillation over the training period, although the average reward and convergence time has insignificant differences.

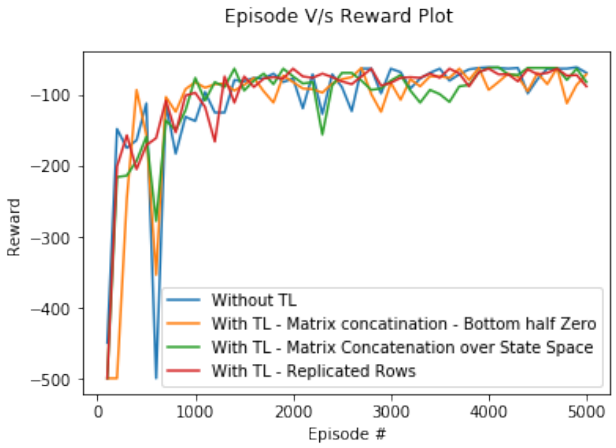

Figure 1. Transfer learning weights

The above graphs shows the same network trained with different initial weights. The blue line represents random initialization of a three layered fully connected network. We further tried following operations on the final layer weights to extract the aforementioned information:

1. Matching the final layer of learned weights to the input of Acrobot's Q-value network by introducing zeros in the second half of the matrix. The intuition behind performing this experiment is to provide learnable parameters for the environment specific (local) features. The outcome of this experiment is represented by orange line in the above graph. The average reward over 5000 epochs is -75.52.

2. As the state space for both environment differ, we stacked the weight matrix multiple times to reduce any stochasticity introduced by this difference. For MountainCar-v0 state (2,) is sampled from continuous space while for Acrobot-v1 state is represented by discrete (6,). The outcome of this experiment is represented by green line in the above graph. The average reward over 5000 epochs is -73.02.

3. We further modified the weight matrix by equally distributing the rows (supposedly representing actions - +1, 0, -1) over the discrete space. The assumption here is that the neighbouring states generally require same action. This helped in retaining a strict correlation between the state and action space better by keeping intact the row information. Hence, it performed the best when compared to other methods. The outcome of this experiment is represented by red line in the above graph. The average reward over 5000 epochs is -69.64.

We repeated this experiment with and without softmax on the last layer of the learned weights. For plotting the above graphs we used the weights without softmax, as it performed better than the one with softmax.

6.2.2 Transfer Learning with changes in Architecture

We experimented with a few combinations of transferring the learned weights from different layers and feeding into a smaller network. Most of the experiments in the section revolve around transferring weights from hidden layers and assigning to middle layers of the second environment's network directly, after reducing the initial layers.

These set of experiments performed as good as baseline but nothing significantly shines. Also in a few cases, the noise increased. The important observation is the detail represented by second line which performed equally well considering it had 1 layer lesser than the others.

Figure 2. Performance with changes in architecture

The above graphs shows the same network trained with different initial weights. The blue line represents random initialization of a three layered fully connected network with $\tanh$ non-linearity. We further tried following operations on the final layer weights to extract the aforementioned information:

1. Transferring the learned weights from last but one layer to the initial layer of the new network. The intuition behind the experiment revolved around the idea that general concepts are learned by the initial layers of the network while detailed features are learned by subsequent layers. As we are seeking to transfer knowledge of general idea of momentum and not environment specific learning (like value of torque), we tried transferring earlier layers to see if we can further improve the learning of the agent. This is represented by yellow line in the graph.

2. Removing the first layer altogether and initialing

the weights to subsequent layer with the learned parameter. The new network resulted in two-layered fully connected network. The idea behind experiment was to test if lesser number of parameters would result in almost similar performance. This is represented by green line in the graph.

3. In this experiment we went further up the learned network and initialize with the second layer. This is represented by red line in the graph.

6.3. Experiments from Acrobot-v1 to MountainCar-v0

Out of curiosity, we also tried transferring weights from Acrobot to MountainCar-v0 to see if it works. As with MountainCar to Acrobot experimentation, we tried using the weights directly. We also tried flattening/reshaping to match the state space. However the network performed poor than baseline. Further, we tried a set of experiments aimed at extracting action information and general features (non-environment-specific information). As the states and the local variables differ, these set of experiments required different methodology than the experiments explained in previous subsection. To do so, we map weights corresponding to different actions to plot the below graph. Yellow line represents when knowledge of only action -1 and 0 was transferred. Similarly, green line represents for 0 and +1, and red for -1 and +1. The weights instead of improving performance, added further noise making the graph almost incomprehensible. This follows the intuition that agent trying to solve Acrobot, does not learn the idea of direction and tries to solve a different environment both ways. This results in adding noise to MountainCar-v0 as it is solvable only in one direction.

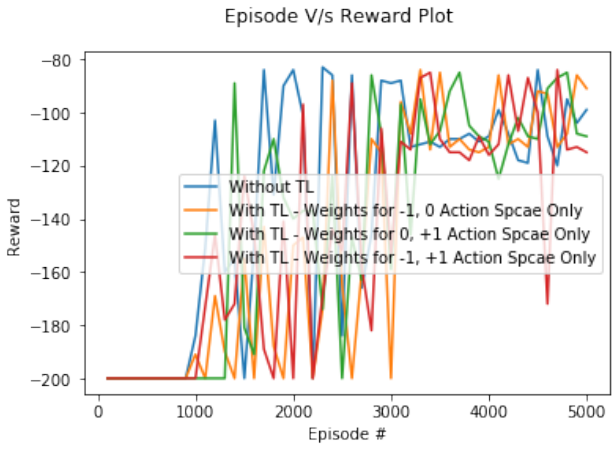

Figure 3. Transfer learning from Acrobot to MountainCar.

7. Conclusion

Transfer Learning in context of Deep Reinforcement learning seems feasible as the agent learns general features (beyond local environment) that can be used in different environments with similar objectives to improve training stability and reduce noise. Additionally, failure to reduce noise in MountainCar from Acrobot weights asserts the understanding that agent is learning key principles to solve the game like momentum and sense of direction. However there is no significant improvement in average rewards over the baseline suggesting that transferred weights carry a lot of information that is not exactly applicable in a different environment. Further work is required to understand how to differentiate environment specific knowledge from general learning.

8. Future Works

In context of Deep Reinforcement Learning, transfer learning is very specific. It works on the environments which we experimented on but may not work with different environments. We would like to further see if the similar approach works across different environment with minimal changes to the experimentation setup. Also, we would like to further work on understanding how to separate local and general information as it still stays a black box. Understanding the same would be the key on developing a generalized approach of transfer learning which will allow same model to works for distinct environments. We aim to explore following approaches:
1. Using information gates to better control information flow and distill specifics.
2. Instead of initializing, using pointer networks to provide direct path for information where scores are sampled from pre-trained weights.

References

[1] Deepak Pathak Thomas L. Griffiths Alexei A. Efros Rachit Dubey, Pulkit Agrawal. Investigating human priors for playing video games. 2015.

[2] RS Sutton AG Barto and CW Anderson. https://gym.openai.com/envs/cartpole-v1/. 1983.

[3] RH Klein W Dabney J How A Geramifard, C Dann. https://gym.openai.com/envs/acrobot-v1/. 2015.

[4] Andrew Moore. https://gym.openai.com/envs/mountaincar-v0/. 1990.

[5] Playing atari with deep reinforcement learning. 2013.

[6] Human-level control through deep reinforcement learning. 2015.

[7] Ruslan Salakhutdinov Emilio Parisotto, Jimmy Lei Ba. Actor-mimic: Deep multitask and transfer reinforcement learning. 2016.

[8] Using transfer learning between games to improve deep reinforcement learning performance and stability. 2017.

[9] Transferring deep reinforcement learning with adversarial objective and augmentation. 2018.

[10] Improving deep reinforcement learning via transfer - doctoral consortium. 2019.

[11] Keyu Duan Dongbo Xi Yongchun Zhu Hengshu Zhu Hui Xiong Qing He Fuzhen Zhuang, Zhiyuan Qi. A comprehensive survey on transfer learning. 2019.

[12] Professor Philip S. Thomas. Compsci 687: Reinforcement learning - fall 2019. 2019.