Overview of work done in BTP-1

Reinforcement Learning for Playing Games

Project Mentor - Prof. Pawan Kumar

Geethika Dudyala
Thapaswini Chowdary
Lakshmee Sravya

May 11, 2019

Objectives

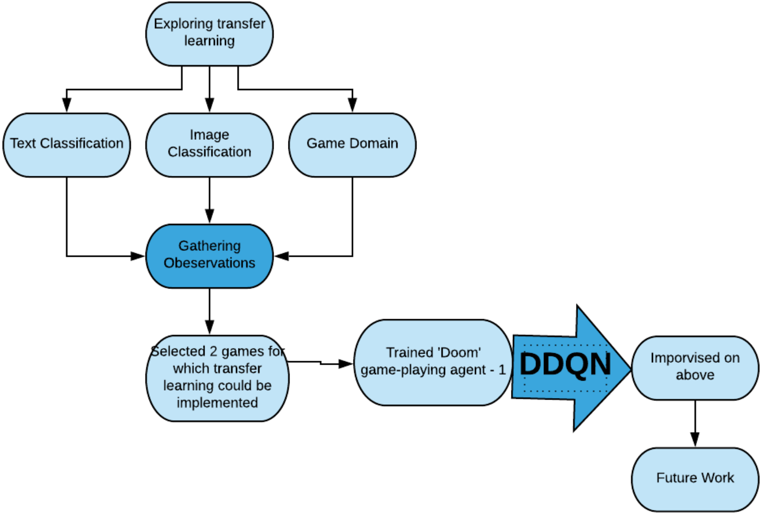
Objective-1

Explore the use of transfer learning in general and more specifically in the game domain.

Objective-2

Analyse whether making use of transfer learning would further boost the performance and any other potential benefits of transfer learning.

Methodology - Project Pipeline



Transfer Learning Case Studies

Case - Study 1

Transfer learning from pre-trained models to solve any image classification problem.

Case - Study 2

Objective here is to fine-tune a pre-trained model and use it for text classification on a new dataset.

Image Classification Problem using Transfer Learning - 1

1. How is Transfer Learning used in computer vision?

CNN

1. Convolutional base
2. Classifier

The main goal of the convolutional base is to generate features from the image.
Classifier is usually composed by fully connected layers. The main goal of the classifier is to classify the image based on the detected features.

Architecture of CNN

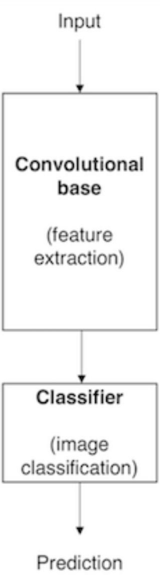


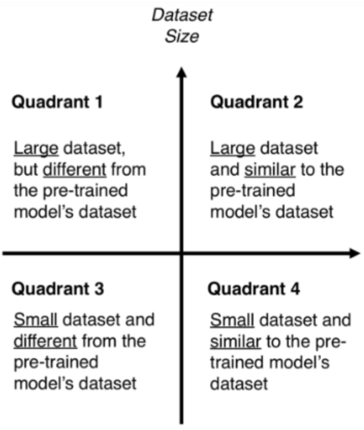
Image Classification

Implementation

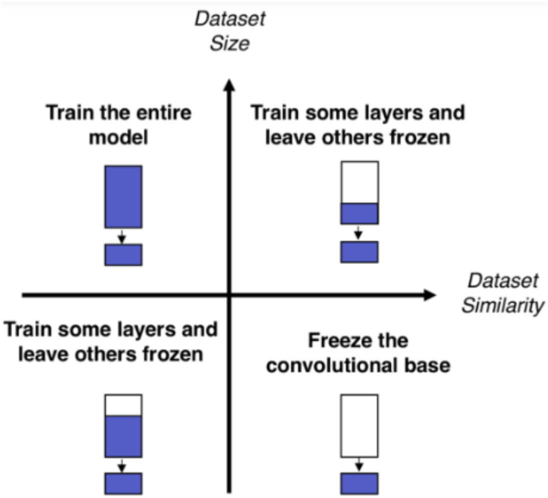
A transfer learning solution to solve image classification problem. We used a dataset consisting of around 2000 images each of cats and dogs. Goal is to build a model that classifies into dog or cat (binary classification problem - 1=dog, 0=cat). Used Keras framework.

Implementation

1. We selected a pre-trained model - used VGG (available from Keras). We initialised the model's weights as same as that of ImageNet



Transfer Learning Process



Observations

1. When we used a large data set that is similar to the pretrained model's dataset, there was not much difference in the accuracy obtained whether we train the entire model or we train some layers and leave the others frozen.
2. When we reduced the size of the dataset, we found that we get a higher accuracy when we freeze the convolutional base layers and retrain the classifier part. (the transfer learnt model performs better)

Results obtained

1. FOR SMALL DATASET -
   1. With transfer learning - 85.57% accuracy
   2. Without transfer learning - 79.32% accuracy
2. FOR LARGE DATASET -
   1. With transfer learning - 89.12% accuracy
   2. Without transfer learning - 90.47% accuracy

Transfer Learning for Text Classification

1. Pre-trained word embeddings like word2vec, Glove, fastText can be used to initialise the first layer of a neural network but the rest of the model has to be learnt from scratch which makes it inefficient.
2. To avoid this problem we required pretrained models which can be fine tuned and used on different data sets.
   E.g. ULMFIT
3. ULMFIT
   1. Take a pretrained Language model
   2. Fine tune the selected language model
   3. Fine tune the classifier
4. Used a python library called FastAI

Results obtained

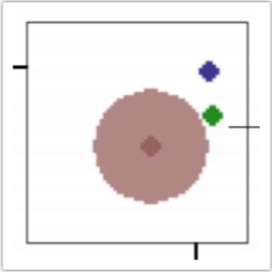
1. Without transfer learning - 92.25% accuracy

1. With transfer learning
    1. retraining the last layers - 90.14% accuracy
    2. reinitialising and retraining the last layers - 95.36% accuracy

Game Domain

To review if we can use a pre-trained Deep Q-network that has been proven to play a particular game well, to play another similar such game, with a small amount of additional training or layers, we have read a Stanford paper "Using Transfer Learning Between Games" which implemented transfer learning for 2 games - 'Snake' and 'PuckWorld'.

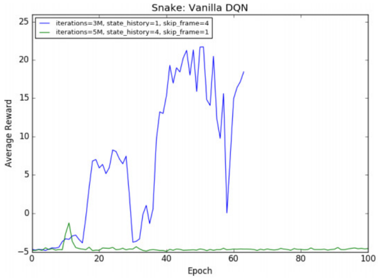
A little about the paper

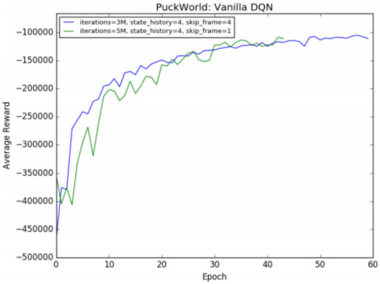


1. About the games used - 'Snake' and 'PuckWorld'
2. Both games have a sense of similarity - the agent has to move towards an object while at the same time avoiding some other object.
3. They have used Deep Q-Learning algorithm (DQN architecture) to implement their agents.

Observations - (1)

1. They have considered various combinations of retraining layers and reinitialising layers in the neural network.
2. Results for snake and PuckWorld which they have obtained





Observations - (2)

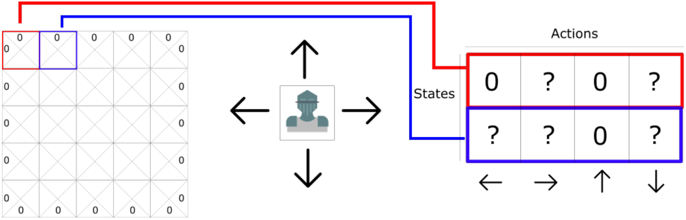
1. They have experimented with DDQN architechture, and have found that the performance of the snake has increased and also the model is faster converging.
2. Transfer learning from puckworld to snake has decreased oscillations in the average reward obtained for the latter.
3. The performance of puckworld declined when transfer learning was performed from snake to puckworld.

Reinforcement Learning

1. Exploration/Exploitation trade off
2. Idea of Deep Reinforcement Learning

Q-Learning

1. Idea of Q-Learning
2. Implementation of Q-table

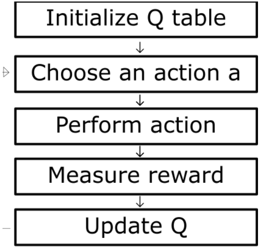


3. Learning the action value function

$$\underbrace{Q^{\pi}(s_t, a_t)}_{\text{Q value for that state given that action}} = \underline{E}[\underbrace{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots}_{\text{Expected discounted cumulative reward ...}} | \underbrace{s_t, a_t}_{\text{given that state and that action}}]$$

Q-Learning

1. Q-learning algorithm Process



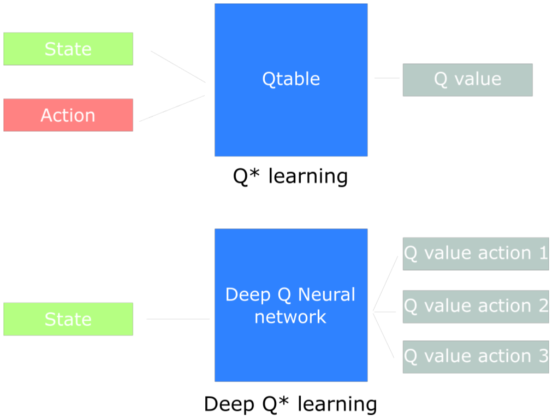
1. Initialize Q-values ($Q(s,a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s,\cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s,a) := Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

'Doom' Game

1. Reward for shooting monster 100, -1 for each second monster is alive, -15 for missing target

Deep Q-Learning - Implemented 'Doom' playing agent

1. Idea of Deep Q-learning - Why not Q-table?



'Doom' playing agent - Technical Details (1)

1. Preprocessing
   1. Converting to grayscale
   2. Cropping image

1. Stacked Frames (to handle the problem of temporal limitation)

'Doom' playing agent - Technical Details (2)

1. We implemented 'Experience Replay'
   1. Our agent should not forget its previous experiences

   2. This reduces correlations between experiences.
      1. We broke this correlation by sampling the replay buffer randomly.

'Doom' playing agent - Technical Details (3)

1. Deep Q-Learning algorithm that we used

$$\underbrace{\Delta w}_{\text{Change in weights}} = \underset{\text{learning rate}}{\alpha} \underbrace{[\underbrace{(R + \gamma \; max_a \, \hat{Q}(s', a, w))}_{\text{Maximum possible Qvalue for the next\_state (= Q\_target)}} - \underbrace{\hat{Q}(s, a, w)}_{\text{Current predicted Q-val}}]}_{\text{TD Error}} \; \underbrace{\nabla_w \hat{Q}(s, a, w)}_{\text{Gradient of our current predicted Q-value}}$$

Fixed Q-targets

(First introduced by DeepMind - Google)

$$\Delta w = \alpha[(R + \gamma\, max_a\, \hat{Q}(s', a, w)) - \hat{Q}(s, a, w)]\, \nabla_w \hat{Q}(s, a, w)$$

Change in weights

learning rate

Maximum possible Qvalue for the next_state (= Q_target)

Current predicted Q-val

TD Error

Gradient of our current predicted Q-value

$$\Delta w = \alpha[(R + \gamma\, max_a\, \hat{Q}(s', a, \boxed{w^-})) - \hat{Q}(s, a, \boxed{w})]\, \nabla_w \hat{Q}(s, a, w)$$

Change in weights

learning rate

Maximum possible Qvalue for the next_state (= Q_target)

Current predicted Q-val

TD Error

Gradient of our current predicted Q-value

At every T steps:

$$w^- \leftarrow w$$

Update fixed parameters

Double Deep Q Network Architecture (Double DQN)

$$\underbrace{Q(s,a)}_{\text{Q target}} = \underbrace{r(s,a)}_{\text{Reward of taking that action at that state}} + \underbrace{\gamma max_a Q(s',a)}_{\text{Discounted max q value among all possibles actions from next state.}}$$

Q target

Reward of taking that action at that state

Discounted max q value among all. possibles actions from next state.

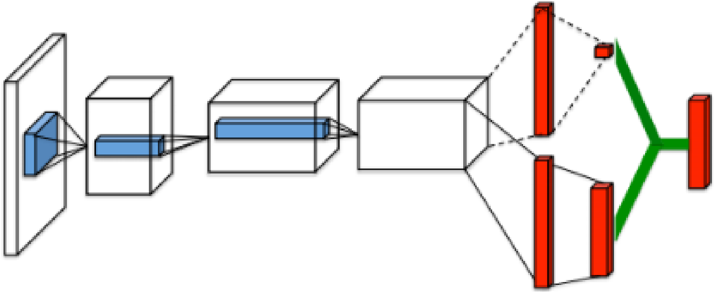
$$Q(s,a) = r(s,a) + \gamma Q(s', argmax_a Q(s',a))$$

TD target

DQN Network choose action for next state

Target network calculates the Q value of taking that action at that state

Dueling Deep Q Network Architecture (DDQN) - (1)



It separates how valuable a state is with what the advantage of different actions are from a state by using different streams.

$$Q(s, a) = A(s, a) + V(s)$$

Dueling Deep Q Network Architecture (DDQN) - (2)

$$\cancel{Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)}$$

In the above, there is an issue of identifiability. (problem for back propagation!) To avoid this, we have implemented for our agent :

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |\mathcal{A}|} A(s, a'; \theta, \alpha) \right)$$

This was proposed in the paper "Dueling Network Architectures for Deep Reinforcement Learning".

'Doom' playing agent - Improvised (Summary)

1. We improvised our agent in the following ways :
   1. Fixed Q-targets
   2. Double DQNs
   3. Dueling DQN (aka DDQN)

In total we have implemented a Dueling Double Deep Q Learning agent with fixed q-targets.

'Doom' playing agent - Results obtained

1. Average reward for DQN agent over 10 episodes - 62.91
2. Average reward for DDDQN agent over 10 episodes - 70.37

Future Work

1. We have selected a similar game to 'Doom' named 'Space Invaders' for conducting the transfer learning.
2. Space invaders is a game in which the agent moves horizontally and shoots at a number of aliens that are continuously moving.
3. As part of BTP-2 we would like to implement transfer learning experiments (both ways).