Report

Submitted in partial fulfilment of

CS F441 Selected Topics from Computer Science

JOINT VIDEO FRAME AND REWARD PREDICTION IN PONG

Inspired by

A Deep Learning Approach for Joint Video Frame and Reward Prediction in Atari Games

Authors	*Student IDs*
Danish Mohammad	2018A7PS0103H
Ashna Swaika	2018A7PS0027H
Rickston Pinto	2018A8PS0986H
Aayush Jain	2018A8PS0320H

Under guidance of

Dr. Paresh Saxena

Birla Institute of Technology and Science, Pilani

Hyderabad Campus

April 21, 2021

Contents

List of Figures

Chapter 1

Paper Implementation and Results

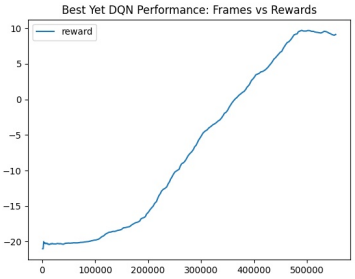
The following steps were involved in paper implementation:

1.1 DQN Model

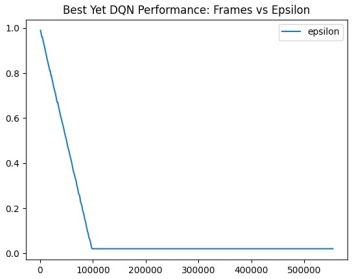
Create Deep-Q Network following Minh et al. A DQN network predicts the action value function for each state action pair. There are two networks with identical structures involved: a policy network and a target network. The Gradient Descent algorithm is implemented only on the policy network while the weights of the target network are updated periodically. The target network is used to determine the error in the prediction of the policy network. The inputs to these networks are the states (in the form of the images) while the outputs are the Q-values for all actions for the current state

1.2 Data Generation

500,000 frames of data are used in training. The frames are stacked one on top of the other, 4 at a time, thus 2 million frames must really be generated. In addition to the frames themselves, the action and reward must also be



(a) A plot of the number of frames vs the average cumulative reward per game



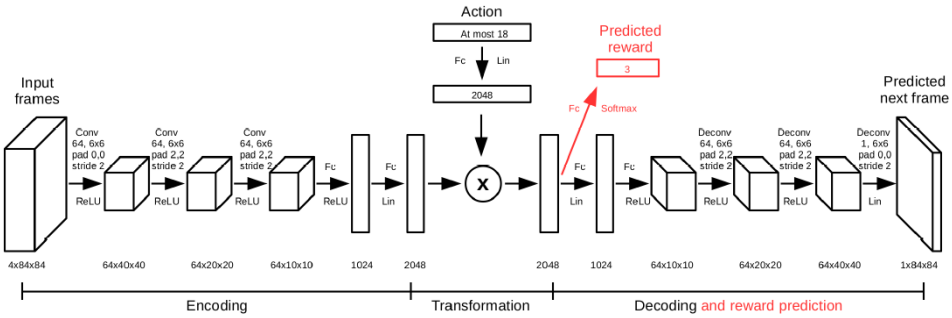
(b) A plot of the number of frames vs epsilon (from ϵ-greedy approach)

Figure 1.1: Some metrics for the DQN

saved. We have simple extended the Replay Buffer from the DQN to get the frames. 50,000 frames (counting a stack of 4 as 1) were also generated for training.

1.3 Model Implementation

Create Autoencoder with action conditional representation added to the latent encoding. . We made a static auto encoder with three convolutional layers for encoding and decoding respectively and two linear layers before and after the transformation. To implement the model we used 2d convolution and linear layer implementation by Pytorch library. The input to the model were 1* 84 *84 grey scale images which were produced by the DQN model in previous step. As suggested in the paper all weights in the network were initialized according to (Glorot and Bengio, 2010). The images were pre-processed so that each pixel has range of [-1,1]. The layers which were a part of element wise multiplication that is the action processing layer, their weights were initialized in the range of [-1,1]. The action was fed in through the DQN model in the previous step and then passed through a fully connected layer for the transformation phase where it was integrated with the compressed latent representation of input frames.



(a) The architecture of the model.

```python
class Encoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 64, 6, stride= 2 ,padding=(0,0))
        self.conv2 = nn.Conv2d(64, 64, 6, stride= 2 ,padding=(2,2))
        self.conv3 = nn.Conv2d(64, 64, 6, stride= 2 ,padding=(2,2))
        self.fc1 = nn.Linear(64*10*10, 1024)
        self.fc2 = nn.Linear(1024, 2048)


    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.view(-1, 64*10*10)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

class Decoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(2048,1024)
        self.fc2 = nn.Linear(1024,64*10*10)
        self.deconv1 = nn.ConvTranspose2d(64, 64, 6, stride = 2, padding=(2,2))
        self.deconv2 = nn.ConvTranspose2d(64, 64, 6, stride = 2, padding=(2,2))
        self.deconv3 = nn.ConvTranspose2d(64, 1, 6, stride = 2, padding=(0,0))

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(self.fc2(x))
        x = x.view(-1,64,10,10)
        x = F.relu(self.deconv1(x))
        x = F.relu(self.deconv2(x))
        x = self.deconv3(x)
        print(x.shape)
        return x
```

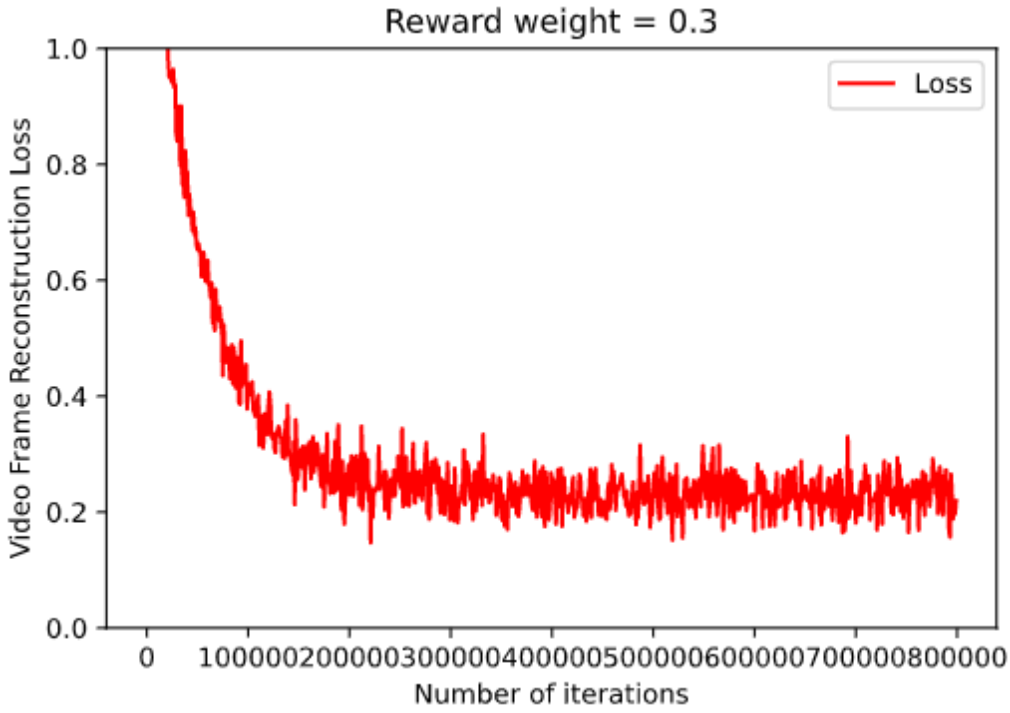
(b) The code for the encoder and decoder.

1.4 Training

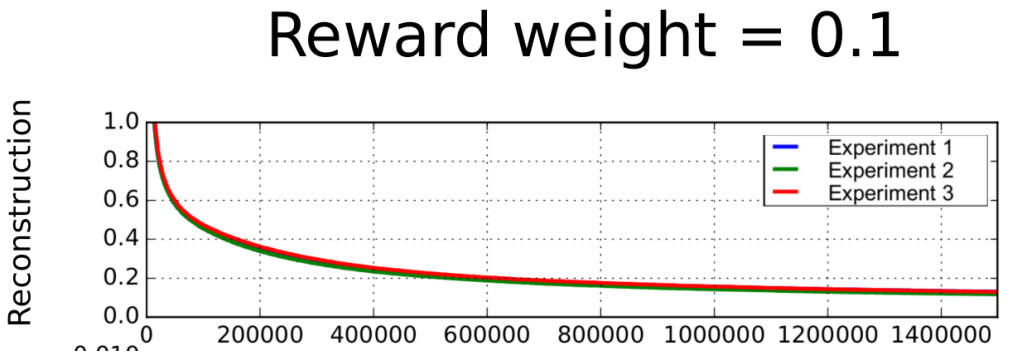
Train the model by using Curriculum Learning through the new combined video frame prediction and cross-entropy reward prediction loss given in the paper. [Add citation] In curriculum learning the look ahead parameter is increased every 500,000 steps. This enables the model to slowly learn to predict farther and farther into the future. The Cost Function combines reward estimation and video frame prediction.

1.5 Testing

Testing involved seeing how far (in terms of look ahead steps) the model could accurately predict.

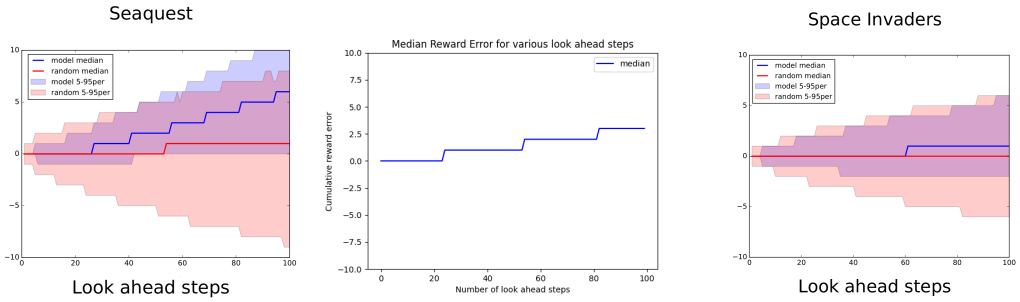


(a) The Video Frame Reconstruction Loss that we obtained. The graph is noisier, both since the reward weight is slightly higher and no smoothing has been done. Additionally, due to lack of access to powerful graphic cards, we were only able to plot the graph till 800k frames.



(b) The reconstruction loss from the paper. Exponential smoothing with a window size of 1000 has been done.

Figure 1.3: Reconstruction Loss comparison, between implementation and paper.



(a) Seaquest (b) Pong (c) Space Invaders

Figure 1.4: Comparison of the median cumulative reward error for Seaquest, Space Invaders (from the paper) and Pong. As we can see, Space Invaders is much easier for the model than Seaquest. This is possibly because of the higher frequency of unpredictable, random events. Since even the opponent (controlled by the computer) in Pong is quite predictable, its mean reward error is more similar to Seaquest.

Chapter 2

Improvement Implementation and Results

The improvement that we were considering was to incorporate a Variational Autoencoder in-place of the regular Autoencoder. In a Variational Autoencoder instead of mapping an input to a fixed vector we map it onto a distribution. This introduces 2 new vectors: one for the mean and one for the standard deviation. And the problem of doing backpropagation through this is solved by re-parameterization, where we represent the sampled latent vector in terms of the mean and standard deviation.

During implementation stage we struggled a lot with re-parameterization and were ultimately unsuccessful in creating an error free implementation. The structure of the reward loss was such that it first decreased and then increased.

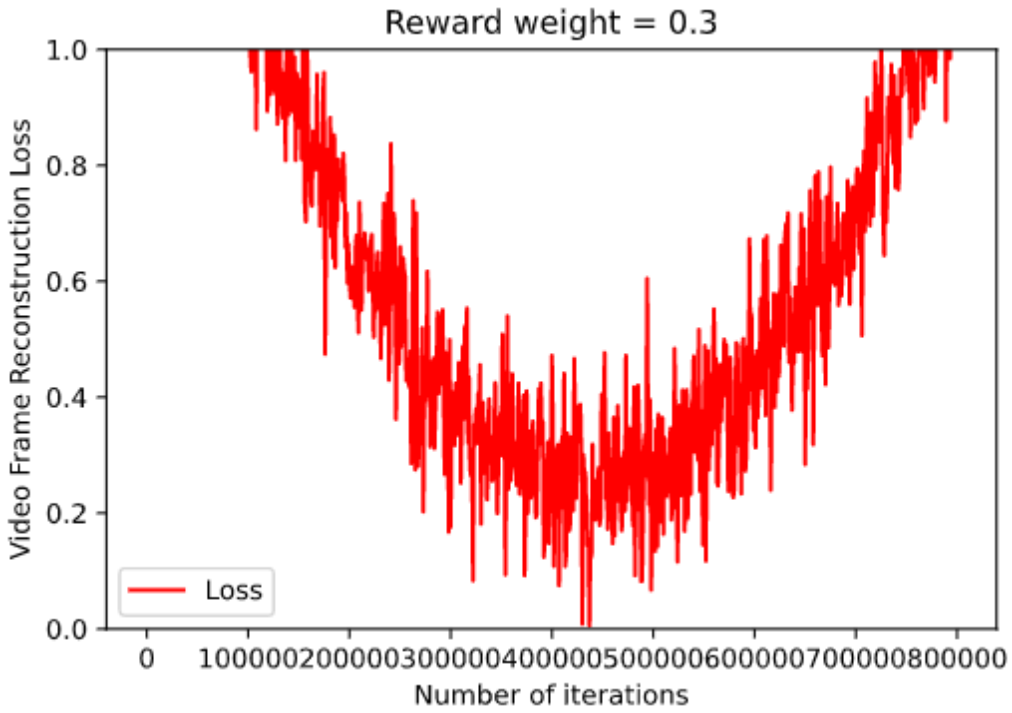


Figure 2.1: Variational Autoencoder Video frame Reconstruction Loss

Chapter 3

Main Findings

We were successful in recreating the basic results of the paper. The look ahead values and the reward loss was comparable.

After training for over 7 hours and over 5,00,000 iterations the reward of the initial DQN model peaked at 10.23. We tried several implementations of it and changing the decay rate and update frequency we found the best decay rate and update frequency 100000 and 10k respectively. The other hyperparameters we are using are 0.00025 as learning rate, 0.99 discount factor and 32 as mini batch size.

Chapter 4

Members and Their Contributions

1. **Danish Mohammad** extracted the training images using the DQN model and made the reward structure for the baseline model, trained the main model on google colab.

2. **Ashna Swaika** made the simple autoencoder model and worked with the variational auto encoder model

3. **Aayush Jain** made the simple autoencoder and worked with the variational autoencoder implementation and tried out several modifications

4. **Rickston Pinto** Worked with the transformation phase and integrating the action vector with autoencoder phase. Also helped with variational autoencoder