Policy Network on Atari Game

Y Yinghuan Wang

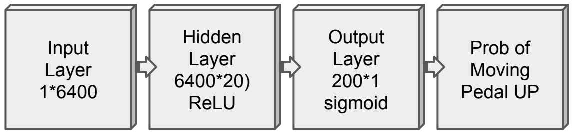
P Peter Yoon(advisor)

Introduction

Policy Gradient (PG) and Reinforcement Learning (RL) are the frontier of neural network. Neural network requires continuous loss to backprop and to learn. However, some problems like Go have no immediate loss because we do not know if a move is good or not until many moves later. PG can help RL solve problems with no immediate loss. PG + RL can be applied to Atari games, Go(AlphaGo), even helicopter control. I focused the Pong game and I created the pipeline to preprocess the observation, forwardprop and backprop. I also developed neural networks using NumPy then I refactored the neural network using PyTorch which greatly improved learning speed.

Methods

The pipeline first cuts edges and subsamples the input then removes the color and flatten the input into a vector of size 6400. Then the pipeline feeds the vector into the neural network. It goes through the hidden layer of size 6400*200 with ReLU activation then outputs from the output layer of size 200*1 with sigmoid activation. Then the neural network computes the loss and gradient and backprop the gradient to update parameters..



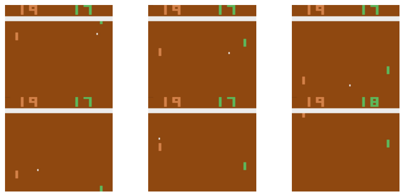
Implementation

The core of the pipeline and the neural network is the Policy Gradient(PG). We only know if a move is good or bad when we win or lose a round after hundreds of moves. Thus PG assumes all moves are good right after a move is done then adjust based on the actual rewards(+1 if win, -1 if lose) after a round is finished. The rewards are also discounted because actions toward the end of the game matter more.

Before Discount	0	0	0	0	1
After Discount	0.92	0.94	0.96	0.98	1

Result

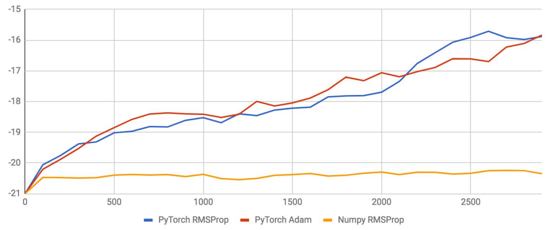
In a 21 rounds game, a reward of -21 means the neural network player lost all 21 rounds. With 3000 games training using PyTorch and Adam, the neural network players running reward increased from -21 to -16 which is significant. The neural network also seem to developed a strategy to allure the opponent to go to the bottom then bounce the ball to the top.



Conclusion

In conclusion, Policy Gradient and Reinforcement Learning really solved the traditional difficult no immediate reward problem. Using PyTorch to build neural network also shows significantly increase in learning speed comparing to using Numpy.

Comparison Between PyTorch RMSProp, PyTorch Adam and Numpy RMSProp



Acknowledgement

I would like to thank Professor Peter Yoon, Professor Takunari Miyazaki and Professor Ewa Syta from the Trinity College Department of Computer Science for their guidance. I would also like to thank Meiling Zhang' 19, Shufan Wang' 18, Samchoukbs777, Zimo1995, GAIGO and luzhaorui for sharing their pearls of wisdom. My project is also powered by the following technologies:

PYTORCH python NumPy OpenAI facebook