Exploration of Deep Neural Networks

Submitted by :

Krishna Wadhwani

Roll No. 160010031

Supervisor :

Prof. Manoj Gopalkrishnan

Department of Electrical Engineering

Indian Institute of Technology, Bombay

November 2018

CERTIFICATE

This project report entitled **Exploration of Deep Neural Networks** by **Krishna Wadhwani** is approved by me for submission for fulfilment of requirements of Supervised Learning Programme. The report was further certified that to the best of my knowledge, represents the work carried out by the student.

Date :

Signature and Name of Guide

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Name of Student)

ACKNOWLEDGEMENT

I would like to express my deep sense of gratitude to **Prof. Manoj Gopalkrishnan**, for his invaluable help and guidance during the course of project. I am highly indebted to him for constantly encouraging me by giving his feedback on my work.

Krishna Wadhwani

November 2018

Indian Institute of Technology, Bombay

Contents

Chapter 1

Introduction

Neural Networks have taken the world by storm by producing groundbreaking results in the fields on Computer vision areas such as image classification, image generation, image translation, etc. and also in the areas of reinforcement learning, with the recent successes of DeepMind's AlphaGoZero and Tesla's self-driving car coming to mind.

This supervised learning project is aimed at exploring some of the latest architectures and algorithms in the fields of Deep learning. Most of the architectures explored are different from traditional neural networks applied in computer vision and reinforcement learning.

The driving motivation behind this project was to get familiar with the current state of the art architectures and their domain of applications. With this familiarity, I aim to start working towards a research project along these lines from the next semester.

1.1 Architectures covered

I started the project with Capsule networks and dynamic routing. After which, I studied some of the applications of General Adversarial Networks. After covering these algorithms that are primarily applied to vision tasks, I moved on to the most in demand area of machine learning- Deep Reinforcement Learning. I covered policy gradients, Q-learning, improvements to deep Q-Learning, actor-critic methods and the applications of these algorithms to solve various gym environments and Atari games.

All the codes and results can be seen in the gitlab repository- https://gitlab.com/krishnaw14/SLP. The codes have been written in python using TensorFlow [1] or PyTorch [2].

1.2 Organization Of the Report

Chapter 2 of this report contains the description, implementation details and results of capsule networks. Chapter 3 contains the details about the implementations of various papers on GANs . Chapter 4 focuses on Reinforcement Learning whereas, Chapter 5 discusses about the future aspects and the directions in which this project can be further taken

Chapter 2

Capsule Networks

2.1 Introduction

Capsule Network [3] is a novel architecture based on inverse computer graphics where the information is presented in hierarchical form. The architecture is formed of Capsules which is a group of neurons whose activity vector represent the instantiation parameters of a specific form of entity. Instantiation parameters of an object are basically its properties such as pose, etc. The length of the vector represents the probability that the entity is present in the image and the orientation of the vector represents the instantiation parameters.

The architecture of this network is relatively shallow and follows an encoder decoder architecture with end to end training incorporating the reconstruction loss for the decoder and classification loss for the encoder.

The encoder consists of 3 layers. The first layer is a simple convolution layer. The second layer is the Primary Capsule Layer consisting of 32 capsules. while the third layer is the digit capsule layer consisting of 10 capsules for each of the ten digits. The decoder is also a 3 layer network consisting of 3 fully connected layers.

2.1.1 Dynamic Routing

The authors propose a new algorithm for training, called Dynamic Routing which works in a hierarchical manner. It is analogous parallel attention mechanism that allows each capsule at one level to attend to some active capsules at the level below and to ignore other capsules.

How this works is that initially the output of the primary capsule is routed to all possible capsules in the DigitCaps layer. The coupling coefficients, that is, the contribution of a capsule in PrimaryCaps layer to a capsule to the digitCaps layer is scaled down to one. Each Capsule in the digitCaps layer computes a prediction vector through a weight matrix. If the prediction vector has a large scalar product with the output of the primary capsule that was initially routed to all the capsules in digitCaps layer, then we increase the coupling coefficient for that digitCaps capsule and reduce it for other capsule.

This procedure is repeated for 2-3 times in each epoch of training.

2.2 Comparison with CNNs

The whole motivation behind this novel architecture is Hinton's dislike for the way in which Convolutional Neural Networks are implemented with the max-pool layer. The max-pooling layer throws away information about the precises position of the entity within an image, something which is preserved by max-pooling layer.

Moreover the output of the Capsule Network is a vector instead of a scalar in case of CNN. So we also retain information about the direction of the object.

Capsule Networks match the state of the art accuracy of the classification of MNIST dataset while outperforming the state of the art architectures in case of classification of multi-digit MNIST dataset. But despite their success with MNIST dataset, Capsule networks have failed to come close to match the accuracy of other datasets set by CNN based networks. [4]

2.3 Implementation and Results

Capsule Network was implemented in TensorFlow and PyTorch and the code roughly takes 15 minutes to run on a GPU. I was able to achieve **99.53 percent accuracy** on the MNIST dataset. The squashing function is stabilized by adding a small number ($\sim 10^{-7}$). The classification loss was computed in a similar manner to SVM using a square hinge loss. For each digit, the loss is computed as:

$$L_k = T_k \max(0, m^+ - ||v_k||)^2 + \lambda(1 - T_k) \max(0, ||v_k|| - m^-)^2$$

$$L_{classification} = \sum_k L_k$$

where $T_k = 1$ when the kth digit is classified and 0 otherwise. m^+ and m^- are taken as 0.9 and 0.1 and denote the desired confidence about the correct prediction and incorrect prediction.

The reconstruction loss is simply the squared difference the squared difference between the reconstructed image and input image to the network.

$$L_{final} = L_{reconstructed} + L_{classification}$$

Chapter 3

General Adversarial Networks

3.1 Introduction

“ Generative Adversarial Networks is the most interesting idea in the last ten years in machine learning.” — Yann Lecun, Director, Facebook

Generative Adversarial Networks or GANs were introduced by Goodfellow, et al[5] in 2014. GANs can learn about data and learn to generate never before seen data to augment a dataset. GANs are a kind of neural networks that is composed of 2 separate deep neural networks competing each other. The two networks are called Generator and Discriminator. Essentially, Generator generates new instances by modelling the probability distribution of the data using a defined number of parameters whereas Discriminator evaluates them to determines whether it belongs to the original training set or not. [7] So, the Generator and Discriminator try to outdo one another in this game during the training process. From the original paper:

“ The Generator model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the Discriminator model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles. ”

3.1.1 Working of GANs

The Generator takes a random input z and tries to generate a sample of data - G(z), where z is a assumed to be a sample from probability p(z). The generated image is then fed into Discriminator who predicts whether the input image to the Discriminator is real or generated.

The game between the Generator and Discriminator is mathematically represented in the form of the following optimization function:

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim P_{data}(x)}[log(D(x)] + \mathbb{E}_{z\sim P_z(z)}[log(1-D(G(x))]$$

D(x) represents the probability that x came from the data rather than Generator.

Discriminator is trained to maximise probability of assigning correct label to both training examples (1 for real data and 0 for generated data). Simultaneously, Generator is trained to minimise the second term in the above expression, i.e., it is trained such that discriminator outputs 1 for the generated image.

3.2 Application, Implementations and Results

GANs are being extensively used in a lot of areas [16], such as image super resolution, text to image synthesis, video predictions, image impainting, deocclusion, image segmentation, reinforcement learning etc. In this project, I implemented three research papers on GANs.

3.2.1 Vanilla GANs

The original GAN [5] paper used fully connected layers for the Generator and Discriminator network. It works well to generate decent images but is rather difficult to train and the convergence rate is very slow. The hyperparameters used for training are generally different from what is traditionally used in similar architectures and instead of using a min-max loss function, we minimize the loss of classification of discriminator on real and generated images [6]. So Discriminator is trained on a cross-entropy loss to output 1 on real images and 0 on generated images whereas Generator is also trained using cross-entropy loss so that the Discriminator outputs 1 on the generated images. At convergence, The Discriminator should output 1 and 0 on the generated images with equal probability or the mean cross-entropy loss converges to 0.5.

The code was written in keras and can be run on CPU alone.

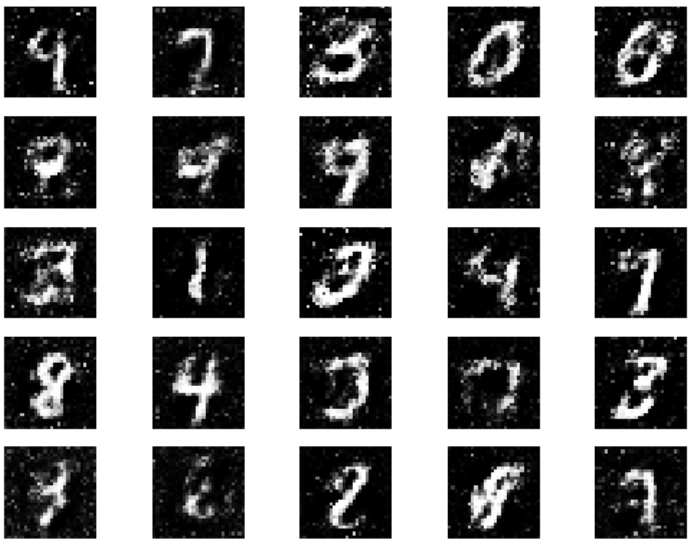


Figure 3.1: Vanilla GANs - Result after 3200 epochs

3.2.2 Deep Convolutional GANs

This model [8] used a convolutional neural network model for the Discriminator and Deconvolution for Generator. The results achieved were much better as convolution is able to capture the details of the images in hierarchical manner. Moreover the convergence rate of the network is also way faster as we are able to see good images even after 5 epochs.

The code was written in PyTorch and 200 epochs of training roughly take an hour to run on GPU.

3.2.3 InfoGANs

The paper [9] introduces a variant of GAN called InfoGAN, which uses the concept of mutual information from Information Theory to learn a disentangled representation of the dataset in a completely unsupervised manner. In the DC-GAN paper [8], it was shown that a lot of interesting things can be done on GANS when doing arithmetic on the noise vector that is fed as input to the generator. In this paper, the input noise vectors of men with glasses are manipulated to give vectors that result in women with sunglasses once fed into the generator. This shows that there are structures in the noise vectors that have meaningful and consistent effects on the generator output.

However, there was no systematic way to learn these structures. These noise vectors are the only parameter that can be modified to have a desired effect in the images generated and since

Figure 3.2: DCGAN - Results after 200 epoch

we basically feed noise into the images, there is no intuition on how to modify it to get a desired effect. The reason for this problem is that the representation of the underlying structures in the images in 'entangled'. That is where InfoGANs come into the picture to provide a disentangled representation that allows us to manipulate the structures in a generated images.

The way InfoGAN works works is by splitting the Generator input into two parts: the traditional noise vector and a new "latent code" vector. These latent codes are then made meaningful by maximizing the Mutual Information between the code and the generator output. It is expected that these latent codes shall capture the significant features like shape or color or thickness of objects, while the traditional noise vector shall represent some irrelevant details of the image. In order to strengthen the relationship between latent code c and the image, the authors proposed that the Mutual Information between them should be as large as possible. This was achieved by modifying the loss function to incorporate this mutual information term. As it is diffcult to mutual information explicitly, the authors derived a variational lower bound on the mutual information.

3.2.3.1 Implementation and results

The code for InfoGAN is written in PyTorch and the network takes around an hour to run on a GPU. The results on MNIST dataset matched the ones reported in the paper. A remarkable observation is that the images of variation in the digits is not achieved via stretching and hence, the resultant images are very natural looking.

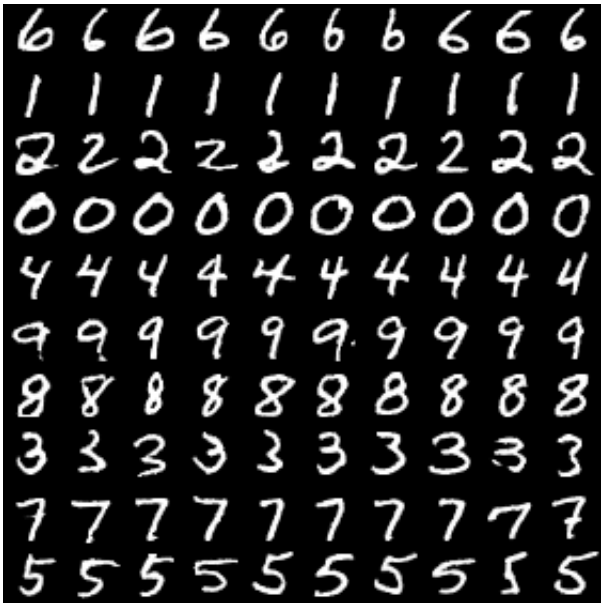


Figure 3.3: Variation of discrete code on InfoGAN - We are able to control the type of digit

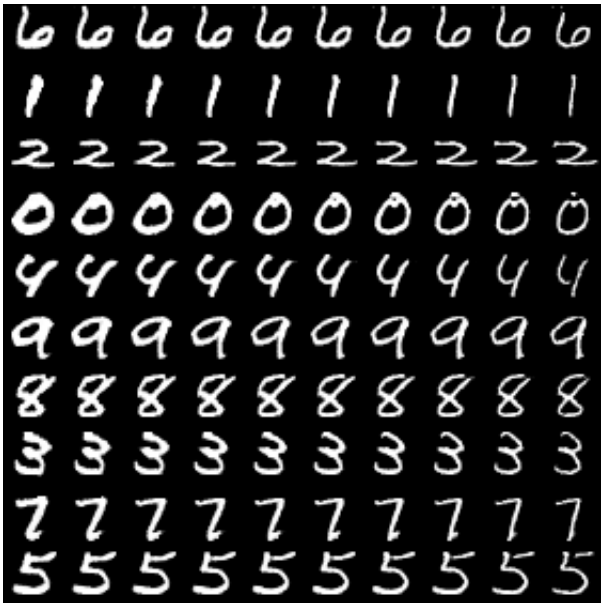


Figure 3.4: Variation of 1st continuous dimension on InfoGAN - Thickness of digits

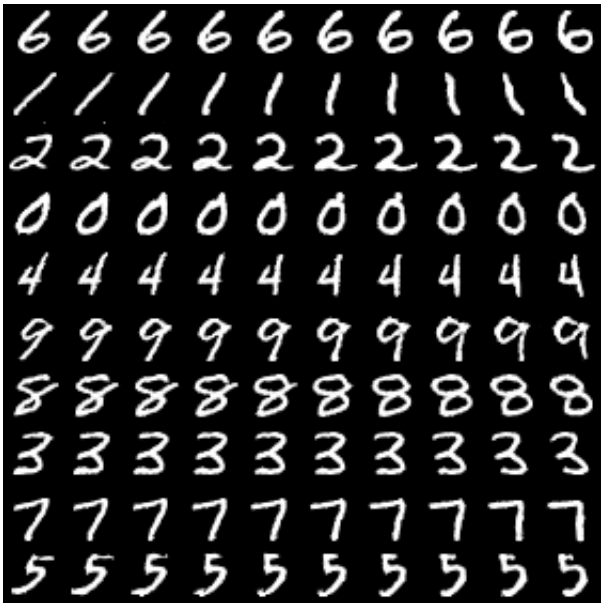


Figure 3.5: Variation of 2nd continuous dimension on InfoGAN - Rotation of digits

Chapter 4

Deep Reinforcement Learning

Reinforcement Learning is the most hyped and in-demand Machine Learning field at present. With the large computation power, this field has produced a number of success stories such as Deepmind's Alpha Go zero and Tesla's self driving cars. The field has enormous potential in all Machine Learning domains.

The aim is to explore the performance of multi agent reinforcement learning in modelling of financial ecosystem and stock prediction. As I had no idea about this field, in the remainder of the project, I studied about Reinforcement learning and various reinforcement learning algorithms.

Reinforcement Learning is the machine learning field that is based on agent-environment interaction. The algorithm takes certain action and based on that action, it gets some reward from the world which acts as a feedback for the agent to improve its policy.

4.1 Deep Q-Learning

Q-Learning is a value-based reinforcement learning algorithm that produces a Q-table that is used by agent to find the best possible action in each state using a greedy approach. For every action of the agent, we get a reward and new state of the environment. With this information, we update the value of the action for that state using Bellman equation.

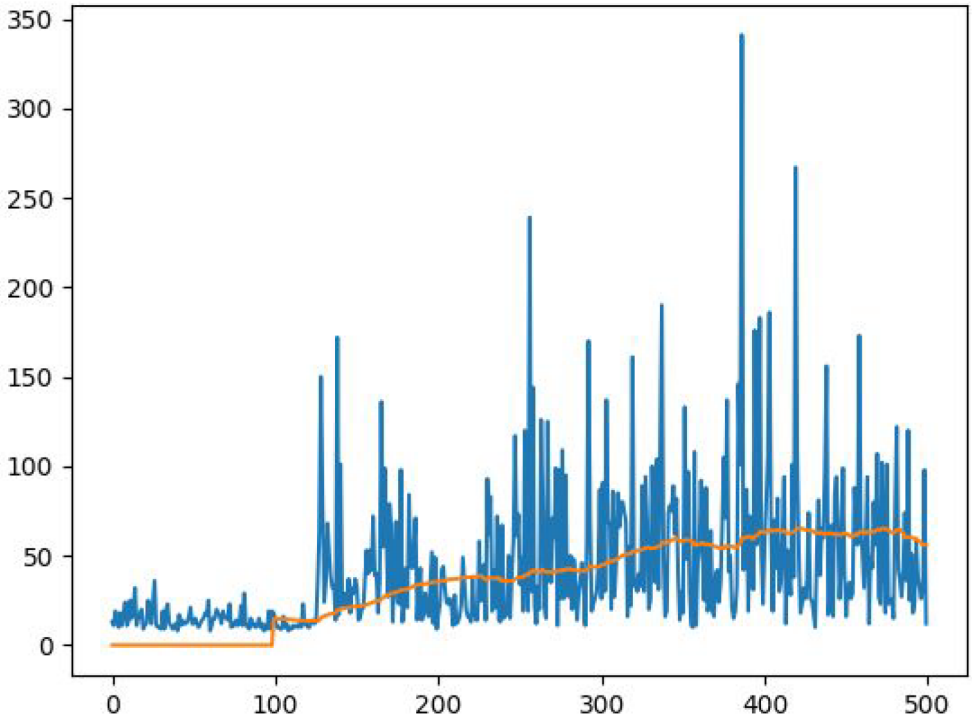
4.1.1 Implementation and Results

How deep Q learning works is that instead of creating a Q table, the neural network takes a state and approximates Q-values for each action based on that state. The parameters of the network are updated using TD error.

$$NewQ(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma(\max Q'(s,a) - Q(s,a)))$$

One of the critical things in the settings of Deep Q-Learning is the exploitation-exploration trade off. We define a exploration threshold that is steadily reduced as the training episode increases.

Deep Q-Learning is further improved via experience replay that stores all the episodes alsong with the actions, states and rewards at each timestep. We stack k number of frames together to capture temporal and motion details and pass it onto our network for the approximation of Q-Values.



4.2 Policy Gradients

In policy gradient algorithm, we directly learn a policy function that maps state to the corresponding action. Our neural network outputs a probability distribution over the action space from which we can chose the action with maximum probability for a deterministic case. During training, we treat the environment as a stochastic process and accordingly let the agent explore different policies. We used Monte Carlo learning, that is, we used end of the episode reward in the loss function to update our network's parameters.

Policy gradients have a certain advantages over Q-Learning in regards that the policy function have been found to have a low variability. Moreover, Policy gradients can be used in cases where the action space is large or the policy to be learned is stochastic. However, it suffers from the problem of credit assignment.

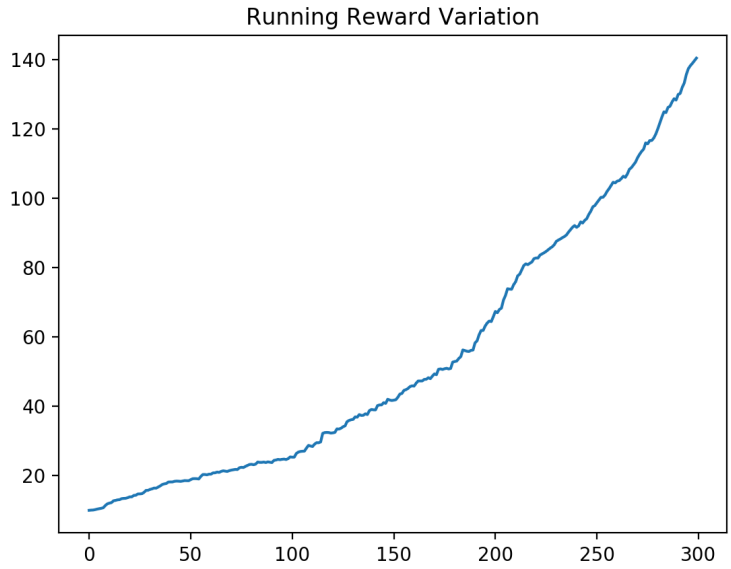
4.2.1 Implementation and Results

I implemented the Policy Gradients on the CartPole environment of gym environment. The code is written in TensorFlow and can be run on CPU. The agent consists of a 4 fully connected layers and trained via the following update rule[10][14]:

$$\Delta\theta = \alpha \times \nabla_\theta\big(log(\Pi(s, a; \theta))\big)R(t)$$

The logarithmic term in the loss function is estimated by taking a cross-entropy loss between the probability distribution over the action space that is the output of the neural network and the actual actions taken by the agent during training.

The following plot shows the variation of running reward plotted against the number of training episode. The agent outperforms Q-learning agent.



4.3 Actor-Critic Algorithm

While both Q-learning and policy gradients algorithms work well in many cases, they both have their sets of limitations. Q-learning cannot be used for stochastic processes or processes with continuous action space while policy gradients can be used in these cases but they suffer from the problem of credit assignment, that is, the algorithm does not know which of its action in the episode contributed to its reward as in Monte Carlo method, we are using end of episode rewards as a feedback to the agent.

So most of the state of the art architectures use a hybrid network architecture combining Policy gradient and Q-Learning methods. We explored one such hybrid architecture that is

the A2C or the Advantage Actor-Critic Algorithm[?]. In this algorithm, we have 2 Neural Networks - Actor and Critic that are simultaneously trained. Actor learns a state to action mapping or the policy function whereas Critic measures how good is the action taken or the Q function[12].

The way this works is that instead of waiting until the end of the episode ;like we do in Monte Carlo learning, we make an update at each step using TD error. At each timestep, both the networks are fed the state of the environment at that time. The Actor maps that state to the corresponding action and receives a new state and reward. With the new state and reward, the critic computes the value of taking that action at that stage. The Actor updates its weight using this q-value. With this updated weight, the Actor produces the next action which is used by the critic to update its own weight.

As value based functions have high variability, A2C algorithm defines an advantage function[13] that is as the difference of q-value for an action in a given state and average value of that state.

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

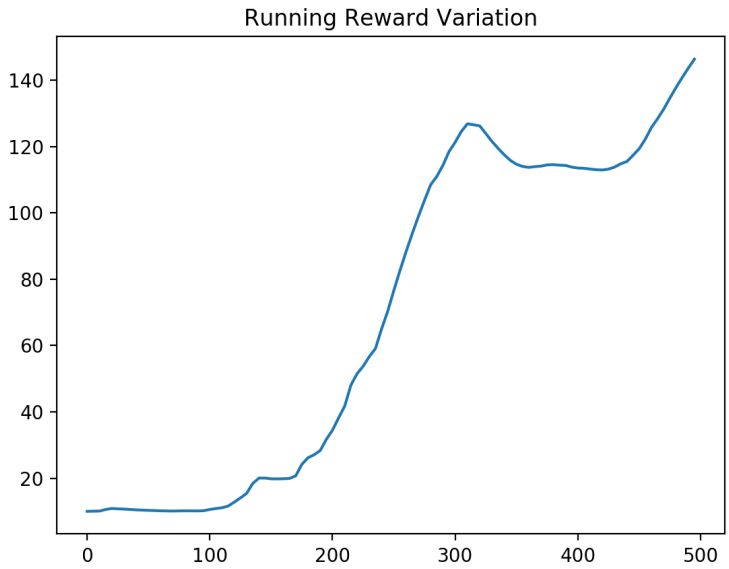
This function tells us the improvement compared to the average, the action taken at that state is. We can use TD error as an estimator of the advantage function. So,

$$A(s,a) = r + \gamma V'(s) - V(s)$$

4.3.1 Implementation and Results

I implemented the advantage actor-critic algorithm to train the agent on the CartPole environment of gym. The code is written in pytorch and can be run on CPU alone. Since the environment is relatively simple, both actor and critic comprise just 2 fully connected layers and have the same learning rate. The critic is trained using the usual Q-learning loss whereas the actor is trained using by using the modified policy gradient loss function (replacing end of the episode reward by advantage function).

In the following plot, we can see how the agent steadily improves over the training episodes.



In just over 500 episodes, the agent is trained and surpasses the results of policy gradient and Q-learning algorithms.

Chapter 5

Learning and Future Extensions

The Supervised Learning Project was a truly enriching experience that allowed me to explore some of the latest and state of the art deep learning architectures. Before this project, I had done a few online Deep Learning courses and worked on one deep learning project based on Neural Machine Translations which I didn't start from scratch. This project allowed me to strengthen my grasp on Deep Learning concepts and applications. It made me aware of the State of the Art algorithms in various fields.

I learned about the Deep Learning framework that is being most actively developed in PyTorch and implemented 4 research papers in Capsule Networks and GANs. After this, I explored the domain of Reinforcement learning and covered the most fundamental algorithms and implemented them on OpenAI gym's environment.

The next step for the project seems like working on a research problem with real world datasets. I have been fascinated by the field of Reinforcement Learning and would like to continue working in this field and as this field has not really been implemented in real world technologies and commercial businesses, it will be really exciting to work on a problem statement on application of deep reinforcement learning on some real world application.

Meanwhile in the winter vacations, I will continue reading and implementing some of the latest Deep Reinforcement Learning algorithms including Multi-Agent Reinforcement Learning.

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.* arXiv:1603.04467 [cs.DC]

[2] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. *Automatic differentiation in PyTorch.* NIPS-W 2017.

[3] Sara Sabour, Nicholas Frosst, Geoffrey E. Hinton. *Dynamic Routing Between Capsules.* arXiv:1710.09829 [cs.CV]

[4] Edgar Xi, Selina Bing, Yang Jin *Capsule Network Performance on Complex Data* arXiv:1712.03480 [stat.ML]

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. *Generative Adversarial Networks.* arXiv:1406.2661 [stat.ML].

[6] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen. *Improved Techniques for Training GANs.* arXiv:1606.03498 [cs.LG]

[7] Shravan Murali *GANs, a modern perspective.* Link

[8] Alec Radford, Luke Metz, Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* arXiv:1511.06434 [cs.LG].

[9] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, Pieter Abbeel. *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.* arXiv:1606.03657 [cs.LG].

[10] Richard Sutton, David McAllester, Satinder Singh, and Yishay Mansour. *Policy gradient methods for reinforcement learning with function approximation.* In Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference, pages 1057 − 1063, Denver, Colorado, 2000.

[11] Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou, et al. *Playing Atari with Deep Reinforcement Learning.* arXiv:1312.5602 [cs.LG].

[12] Vijay R. Konda, John N. Tsitsiklis *Actor-Critic Algorithms*. Advances in Neural Information Processing Systems (2000), p. 12

[13] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu *Asynchronous Methods for Deep Reinforcement Learning*. arXiv:1602.01783 [cs.LG]

[14] Thomas Simonini *Deep Reinforcement Learning Course*. Link

[15] Lectures by David Silver *UCL Course on RL*. Link

[16] JiChao Zhang Github Repository of Research Papers