University of Peradeniya
Department of Computer Engineering

CO542 : Neural Networks and fuzzy systems

Neural Network based inverted pendulum control system

Group 07

E/13/110 : Kesara Gamlath
E/13/277 : Hasindu Ramanayake
E/14/158 : Gihan Jayatilaka
E/14/339 : Suren Sritharan
E/14/379 : Harshana Weligampola

Contents

1 Abstract

Inverted pendulum is an unstable system that collapses if left alone. Movable inverted pendulum is a classic example of dynamic stability. The problem is analyzed with a neural network based control system in this project. Several neural network architectures along with their hyper parameter tuning is considered here. Their performance is analyzed experimentally and justified theoretically.

2 Introduction

The inverted pendulum problem is a classic problem in dynamics and control theory. It is widely used as a benchmark for testing control algorithms. An inverted pendulum is an upside down pendulum. Hence the center of mass is above its pivot point. Due to this reason it is highly unstable. At directly uptight position it is balanced. But a slight displacement from upright position will result in falling down the pendulum. To balance the pendulum in its inverted position it requires to actively apply force to the pivot point. Also needs to consistently monitor the pendulum motion and change the applied force direction and the amount.

2.1 Motivation and Justification

The mobile inverted pendulum problem has a simple yet complete solution in continuous time case (assuming ideal conditions). But the physical implementation of this differential equation based solution requires an ideal engine.
Transferring the ideal case (frictionless, light) solution to real world requires a large mathematical analysis which is virtually impossible. In contrast, a neural network solution that works for the ideal case can easily be transferred to the real world application with some more training.
The continuous time solution is impossible in the real world given the non zero computation times, non zero actuation times etc: And the engine (or other mechanism) that moves the cart is not ideal (it cannot give every force instantaneously). But an impulse based discrete time system is easily implementable in real world.

2.2 Objectives

- Build a standard neural network model.

- Modify it to suit the problem definition.
- Evaluate the model.

2.3 Problem

2.3.1 General problem

Balancing a pole on a cart is a mechanical engineering problem. The challenge is to move the cart in a way that the pole will not fall on to the ground.

2.3.2 Simplified problem

The general problem has to deal with various complexities such as the friction between the cart and the ground, the friction at the joint of the cart and the pole etc: This project abstracts out the problem to:
A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every time-step that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

The problem being solved in this project is **given input $\underline{\mathbf{X}}$ find the suitable output $\underline{\mathbf{Y}}$** where
$\underline{\mathbf{X}}$ is a vector containing information about the current state of the cartpole and $\underline{\mathbf{Y}}$ is a scalar with values +1 or -1 to mean which side the cart should be pushed in the particular instance.

$$\underline{\mathbf{X}} = \begin{pmatrix} \mathrm{x}_0 \\ \mathrm{u} \\ \theta \\ \mathrm{v} \end{pmatrix}$$

Symbol	Meaning	Minimum	Maximum
x_0	Displacement of cart length units	-4.8	4.8
u	Velocity of the cart in length units per second	-inf	inf
θ	Angle of the pole in degrees	-24	24
v	The velocity of the tip of the pole in length units per second	-inf	inf

$$\underline{\mathbf{Y}} = \begin{cases} +1, & MovetoRight \\ -1, & MovetoLeft \end{cases}$$

3 Proposed Methodology

A neural network based control system is proposed to this problem. The best neural network architecture will be picked by trying different architectures. Why one architecture performs better and why one doesn't will be justified with the knowledge on neural networks. The architectures will be further improved by fine tuning the hyper parameters.

3.1 Proposed technology stack

- Python for scripting
- Numpy and TF for heavy numerical computatiom
- TF learn and TF for implementing neural networks
- OpenAI Gym for physics simulation
- Tensorboard for model and model performance visualization

4 Procedure and Methods

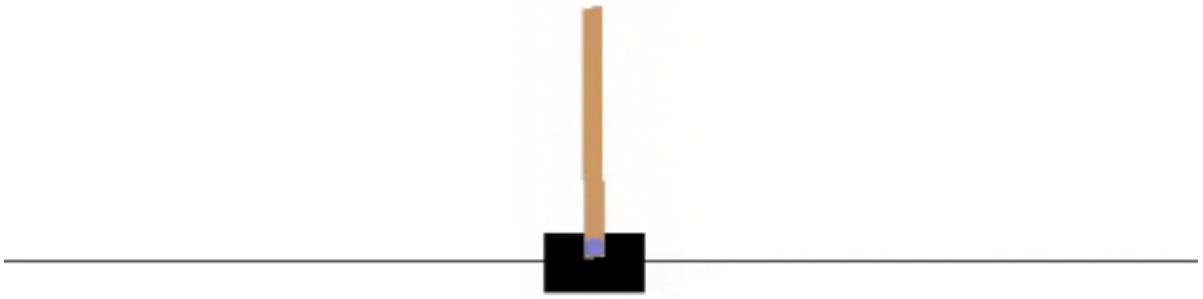

Figure 1: OpenAI Gym Cart Pole Simulator

For the inverted pendulum we selected the OpenAi Gym Cartpole as our simulation tool. It is a graphical simulation tool. Hence we could apply inputs and graphically see the resulting movements of the inverted pendulum. This Cart-pole required 1 input which is the direction of the force. 0 for left and 1 for right. After applying the input it outputs 4 parameters that is the result of the input. They are the displacement change from the x, velocity of the cart, angle of the pole and pole velocity at the tip. Also it checks whether the pole is straight. It is considered straight if the pole has an angle less than 12 degrees from the upright position. If the pole is straight after application of the input, it will output a reward and if not, the reward will be 0 and pole is considered fell down. The maximum number of score obtained by the cartpole is 200 and it will be terminated of the cart moves more than 2.4 from the center.

4.1 Generating the dataset

This problem is handled as a supervised learning problem in this project. This requires a **labelled dataset**. We needed to create a labelled dataset from the simulation environment.
The simulation environment works in discrete time steps. The input for the simulation environment could be given manually and the output (the state of the cart

pole system after a time step) is returned.
The simulation environment itself is not a dataset. The dataset should contain a set of inputs and a set of target outputs. A single input vector and the desired output scalar is generated this way,
A random sequence of 200 numbers (+1,-1) are generated. Then the simulation is run using these as the push given to the cart. If the pole does not fall down for more than 50 time steps, the sub sequence of the random sequence up to the point where the pole falls down (if the pole does not fall down this sub sequence is equal to the random sequence) is isolated.
The sub sequence of push directions is considered as the target output given the state of the cart pole just before the particular push. This a set of $\underline{\mathbf{X}}$ s and corresponding $\underline{\mathbf{Y}}$s.

4.2 Neural Network

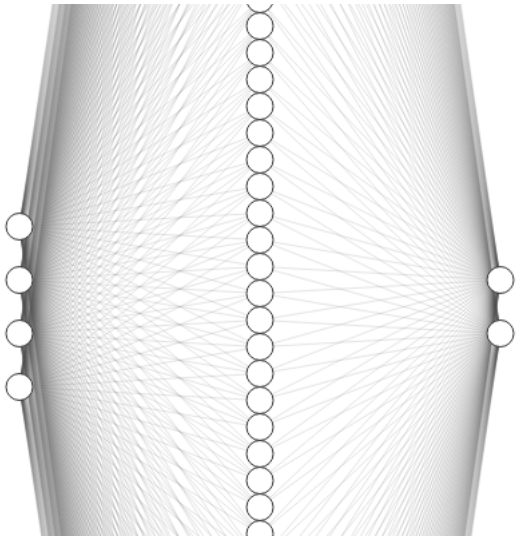

Figure 2: Neural network architecture with 4 node input layer, 2 node output layer and the hidden layer

Then the neural network was created. The neural network had 3 layers which are input layer, hidden layer and the output layer. Hidden layer had nodes to the size of input size. The hidden layer was a fully connected layer with 128 nodes and ReLu as its activation function. Then the output layer had 2 nodes to select left or right with

softmax as its activation function. Adam (adaptive moment estimator) optimizer is used as the optimizer and the loss function is categorical cross-entropy.

4.3 Evaluating the model

The model is evaluated by using the OpenAI Gym environment itself. The ability of the neural network to keep the pole balanced over a longer time period is considered as a positive result.

5 Results Discussion

The experimentation gave the following results.

5.1 Hyper parameter tuning

5.1.1 Different epochs

Here a neural network with 1 hidden layer of 128 nodes was used and it was trained for different numbers of epochs. The activation layer for the hidden layer was ReLu and the activation function for the output layer was softmax.

No of epochs	Average score
1	194.94
2	200.00
3	187.10
4	181.94
5	180.36

Table 1: Performance of the NN against the epoch number

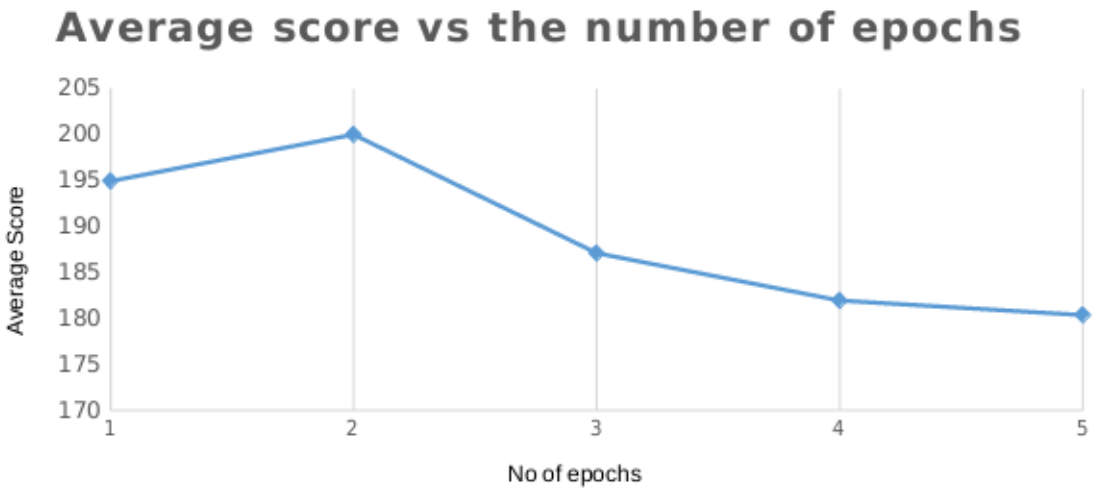

Figure 3: Caption

The perfect accuracy (please note: this is not the perfect cart pole balancing accuracy. This is the perfect ability to keep the cart pole balanced at least up to 200 units of time) comes at 2 epochs. The average score goes down after that. This is evidence for an over-fitting.

5.1.2 Number of nodes in the hidden layer

The nerual network was trained and tested with different numbers of nodes in the hidden layer. The activation function was ReLu (for hidden layer) and softmax (for output layer). 2 epochs were allowed for training.

Nodes in hidden layer	Average score
32	198.88
64	199.87
128	200
256	192.9
512	146.1

Table 2: The variation of accuracy with the number of nodes in hidden layer

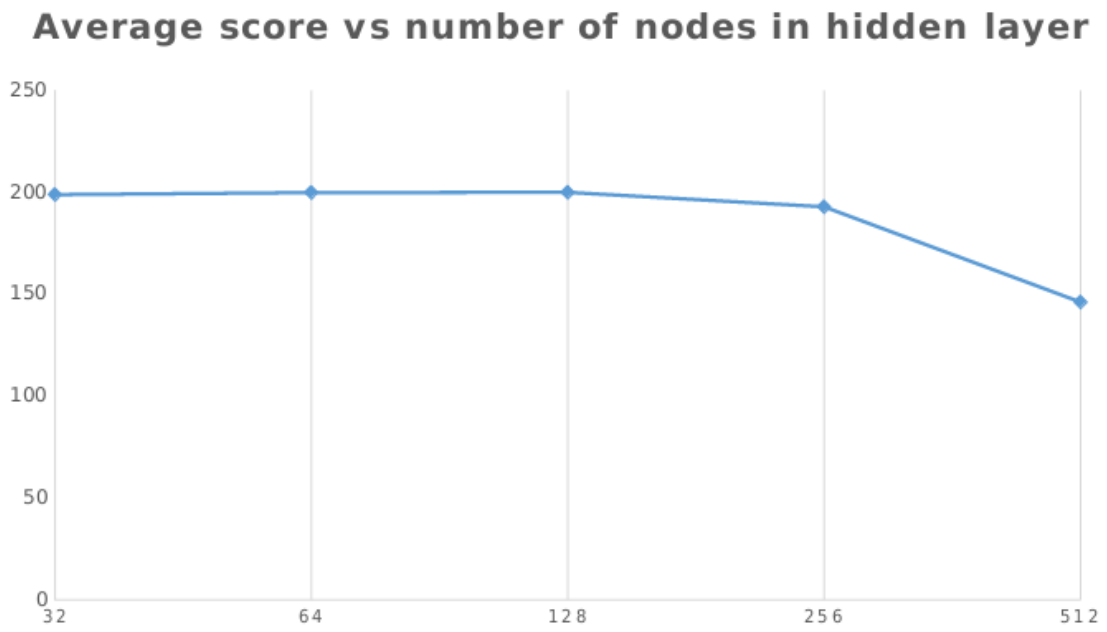

Figure 4: Caption

It is evident that higher number of nodes in the hidden layer (complicated models) causes overfitting. This is understandable since the problem at hand is a classical mechanics problem of very low complexity.

5.1.3 Number of hidden layers

The neural network was trained and tested with different numbers of hidden layers. The activation function used for all hidden layers was ReLu and the output layer used the softmax activation function. The number of nodes in hidden layers were fixed at 128. Dropout layers were not used in between the hidden layers.

No of hidden layers	Average score
0	161.88
1	200.00
2	197.22
3	183.05

Table 3: Caption

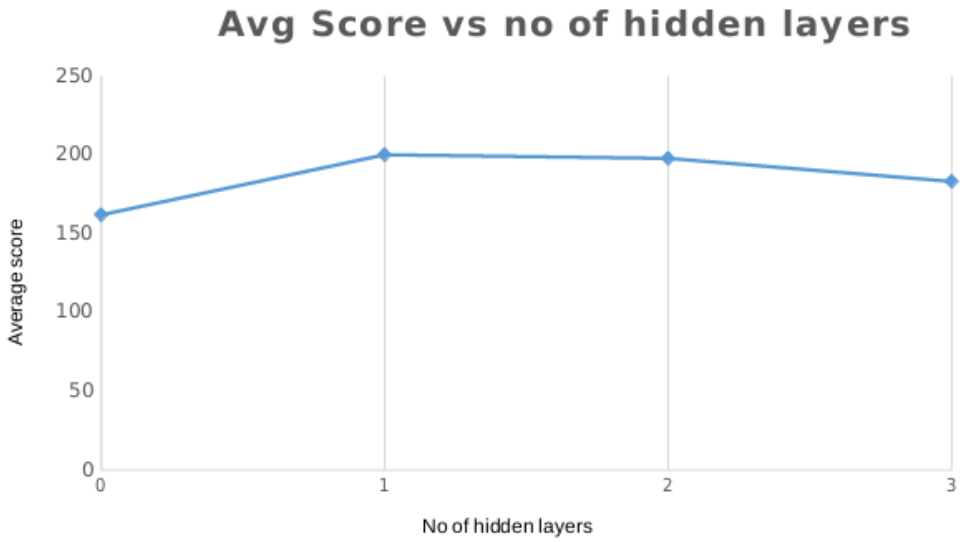

Figure 5: Caption

The following conclusions can be derived out of the graph.

- The neural network gives a good accuracy even without any hidden layers. This is because the mechanics problem can be approximated easily with a linear model. Even if we think with intuition – "push the cart to the direction on which the pole is falling down" we can keep the pole steady for a long time. This is why zero hidden layers works.
- The accuracy peaks at one hidden layer. This is because of the multiple layers causing overfitting. Multiple layers are not needed for a a simple mechanics problem.
- Absense of a overfit avoidance mechanism is a problem

5.1.4 Activation function in hidden layer

The neural network was trained and tested with different activation functions for the hidden layer. But output layer used the softmax activation function for all cases. The number of nodes in hidden layers were fixed at 128.

Activation function	Average score
Sigmoid	88.60
Relu	200.00
Tanh	200.00
Linear	200.00

Table 4: Caption

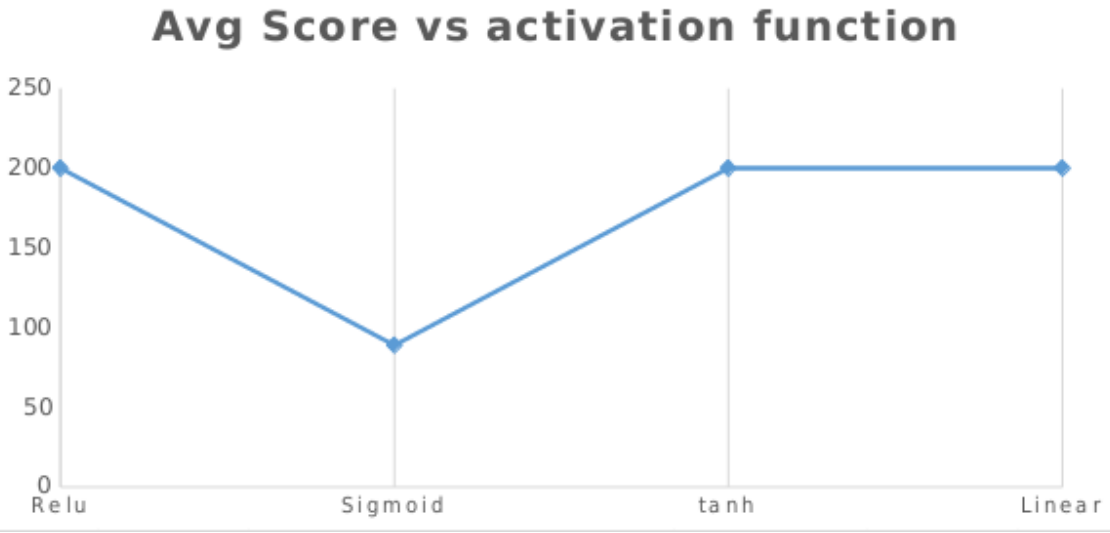

Figure 6: Caption

- Sigmoid functions are not suitable for this problem considering the gradient loss in back propagation.
- The experimental result of linear activation function giving very good performance cannot be explained in our scope.

5.2 Deep neural network architectures

The hyper parameter tuning experiments showed that the neural network tends to overfit. As a solution to this, a multi-layer deep neural network with dropout layers to avoid overfitting was proposed. The neural network architecture is as follows.

Layer no.	Layer type	No of nodes	Activation
1	Input	4	None
2	Fully connected	128	Relu
3	Dropout (80%)		
4	Fully connected	64	Relu
5	Dropout (60%)		
6	Output	2	Soft max

Table 5: Deep neural network

This neural network showed the following performance for training,

No of epochs	Average score
1	190.27
2	198.53
3	200.00
4	200.00
5	200.00

Table 6: Performance of the DNN against the epoch number

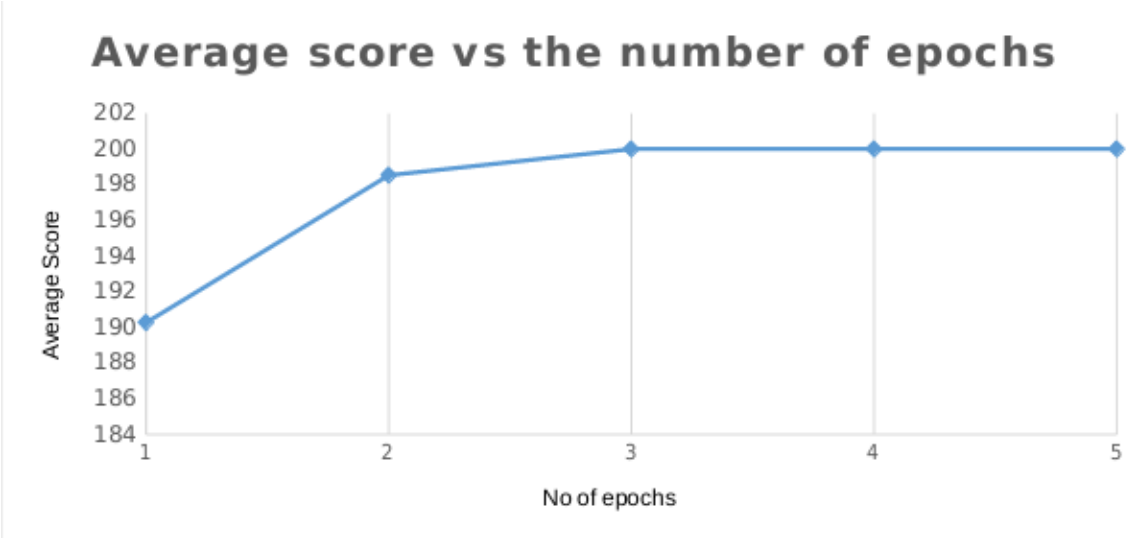

Figure 7: Performance of the DNN against different epoch numbers

It is evident that the previous overfitting problem does not occur when dropout layers are used. But training the deep neural network takes more epochs than the shallow neural network.

6 Conclusion

An inverted pendulum can be balanced using a neural network. But there are certain constraints which limit the performance of this Neural network.

- The Neural network is trained to keep the inverted pendulum balanced for 200 epochs. Thus the stability of the system after these fixed number of cycles is not guaranteed.
- Since the number of epochs are fixed, the decision taken by the model is based on a greedy and short term oriented.Thus, the decision may not be the optimal solution in the long term.

Therefore, even though the neural network model emulates the PID up to a certain extent, this may not be able to perfectly replace a PID: the main reason being that the neural network is made to be simple. Since the expectation of the neural network is optimized to get the greedy solution, the neural network is more similar to a P (proportional) control system. Because in a P model it predicts the output of the next instance by analyzing variables of the current instance only. This neural network can be further improved by including the information about history using a feedback, thus incorporating the functionality of a PID controller to this neural network.

7 Insights

This section discusses the insights gained through the project and a set of open questions we tried to answer.

7.1 The correct and efficient way of generating the dataset

The dataset was generated as per the reported methodology. But this methodology raised a few questions.

1. Can we judge a push decision to be correct based on a random sequence of it as a member giving 50+ time units of stability?
2. Simulating random sequences on the OpenAI Gym is very lightweight. But what if the sampling was expensive? Expensive sampling means a computationally heavy simulation environment (one with complicated models for friction etc:) or a physical system.

It appears as if "a better way of judging which move is right" could improve the neural network training process. For example we can pick moves which tries to make the pole horizontal. But such a step would undermine the actual value of using a neural network. If we try to put our understanding on how we perceive a move to be good into the training process it will reduce the ability of the neural network to learn from the simulation data "with an open mind".

But we can improve the bar for moves to be "correct moves" by reducing the tolerance which disqualifies a sequence or increasing the number of moves we need for a sequence to qualify ($50 \rightarrow 120$)

In case of expensive sampling, we can use reinforcement learning techniques. This was not attempted in this course project

7.2 Optimizer

The optimizer used in this project was **adam (Adaptive Momentum Estimation)**. Experimentally, this optimizer gives very fast convergence for this problem. Theoretically naive gradient descend itself should give the correct convergence to this problem with ideal condition assumptions (directionless system). But a huge dataset against limited RAM prohibits the use of naive gradient descend algorithm. Batch wise gradient descend (stochastic or not) can solve this problem.

7.3 The correct way of modeling the neural network

The main drawback of the the current neural network is that it uses the information of the current instance to predict the motion of the cart-pole. As a result, this neural network inherit the disadvantages of a usual P control system even-though it uses a data set generated by a PDE. Increasing the complexity of the neural network (by adding new hidden layers, adding new nodes to hidden layers) does not solve these issues because increasing these parameters means creating a more complex relationship between the variables of current instance . If we want the neural network to predict the output based on historical data, we need to add a feedback to the neural network. This feed back is more generally implemented using Recurrent Neural Networks (RNN). In RNNs the input layer contains information about the previous instances (as a vector or as raw data). Then, in the hidden layers can use these information to mimic the mathematical functions like integration, derivation

which results in a model which is much similar to a PID control system.

8 Other Learning Exercises

This section lists the tasks done in the project time period as learning exercises (not directly relevant to the topic).

- Implementing neural networks manually by numpy
  - AND gate
  - XOR gate
- Implementing a neural network library

- Trying different NN related problems
  - MINST OCR task
  - Wine dataset classification

9 References

1. Aaron C. Courville, Ian Goodfellow, and Yoshua Bengio. Deep Learning 2015. ISBN: 9780262035613
2. Eric Jones, Travis Oliphant, Pearu Peterson and others. *SciPy*: Open source scientific tools for *Python*
3. Martín Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
4. Abien Fred M. Agarap. Deep Learning using Rectified Linear Units (ReLU)
5. Adam Coates and Andrew Ng and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. Proceedings of Machine Learning Research 2011
6. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting 2014
7. Vijayanand Kurdekar1, Samarth Borkar. Inverted Pendulum Control: A Brief Overview. International Journal of Modern Engineering Research (IJMER). Vol. 3, Issue. 5, Sep - Oct. 2013 pp-2924-2927 ISSN: 2249-6645