

# IMPLEMENTATION OF REINFORCEMENT LEARNING METHODOLOGY ON OPENAI-GYM CARTPOLE ENVIRONMENT

---

by

**Name** : Kompalli V M Jwala Seethal Chandra

**UWM Campus ID**: 991367119 50

**Email** : seethal2@uwm.edu

# Overview

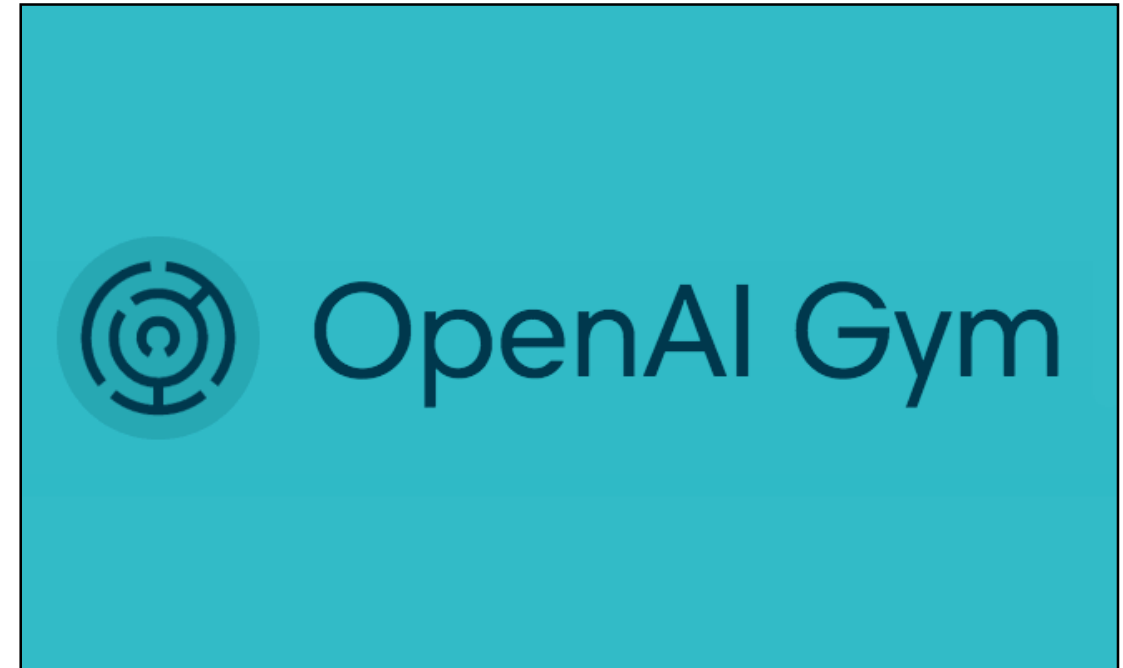
- Due to Deep-Learning's desire for large datasets, anything that can be modeled or simulated, can be easily learned by AI.
- For that, we need to create an environment, which can be done with Python.
- But fortunately, there are many open source and free libraries out there, for us to use, to create an environment.
- I used one of the well known libraries, for gaming, called OpenAI Gym.
- This is a project report on how I used one of the environments provided by Gym and applied Reinforcement Learning, an A.I Technique on it.

# Dependencies used

- Just like any other software, We need to install multiple software dependencies and libraries for us to run this project.
- These are the following that should be installed for this project to work on PC:
  1. **Open A.I Gym**
  2. **Tensor Flow ( GPU version )**
  3. **TFLearn ( Version 1.4 )**
  4. **NumPy**
  5. **CUDA.** I recommend to install it, as we are using the GPU Version of TensorFlow
- In the following slides, I'll go through each one of the above listed dependencies.

# 1. Open AI Gym

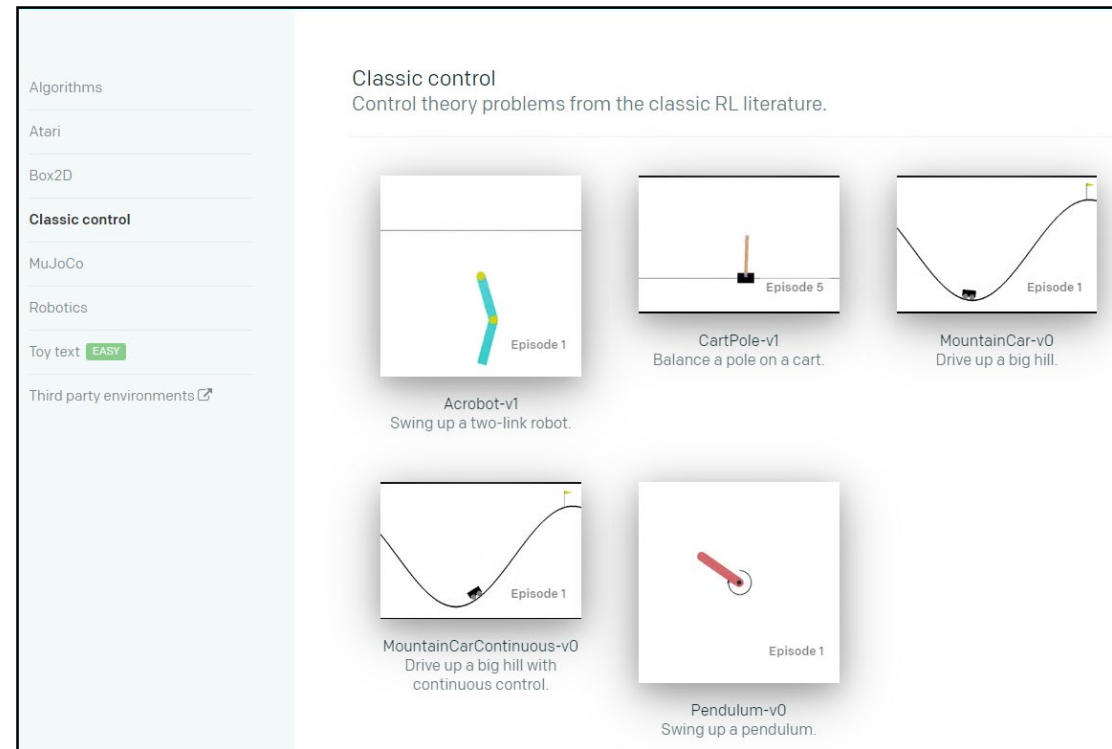
- Gym is a toolkit for developing and comparing reinforcement learning algorithms.
- The [gym](#) library provides an easy-to-use suite of reinforcement learning tasks
- There are many environments that are provided by Gym. I used “**CartPole**” environment for this project.



# Why Gym?

- Reinforcement learning (RL) is the subfield of machine learning concerned with decision making and motor control. It studies how an agent can learn how to achieve goals in a complex, uncertain environment.
- It's exciting for two reasons:
  1. RL is very general, encompassing all problems that involve making a sequence of decisions: for example, controlling a robot's motors so that it's able to run and jump, or playing video games and board games.
  2. RL algorithms have started to achieve good results in many difficult environments. RL has a long history, but until recent advances in deep learning, it required lots of problem-specific engineering. **DeepMind's Atari results** and **AlphaGo** all used deep RL algorithms which did not make too many assumptions about their environment, and thus can be applied in other settings.

- These are some of the environments provided by Gym
- Please feel free to look at the list of environments, present and provided by the website, at the following link
- [https://gym.openai.com/envs/#classic\\_control](https://gym.openai.com/envs/#classic_control)



## 2. Tensor Flow

- **TensorFlow** is a free and open-source software library for dataflow and differentiable programming across a range of tasks.
- It is a symbolic math library, and is also used for machine learning applications such as neural networks.
- I used TensorFlow ( GPU Version ) for this project. There is also a CPU Version, but for this project GPU version works the best.



**TensorFlow**

### 3. TFLearn

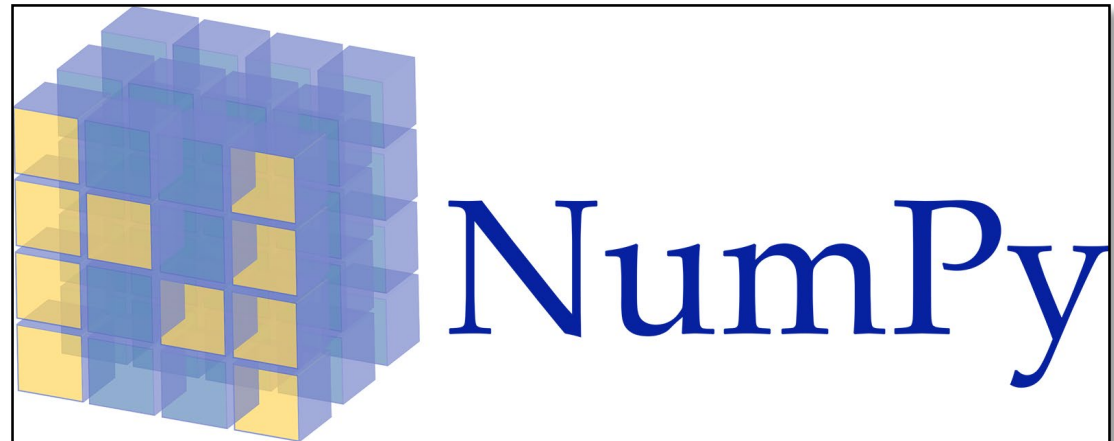
- **TFLearn** is a modular and transparent deep learning library built on top of Tensorflow
- It was designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentations, while remaining fully transparent and compatible with it.





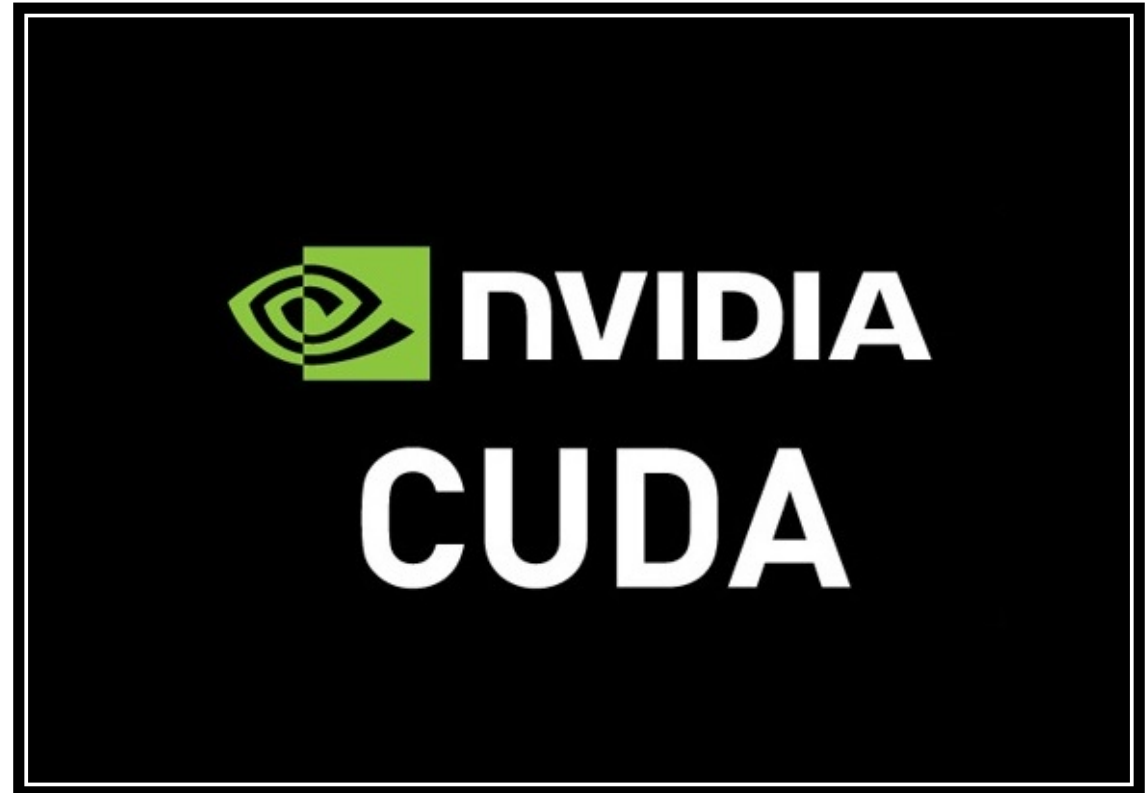
## 4. NumPy

- NumPy is the fundamental package for scientific computing with Python
- NumPy can also be used as an efficient multi-dimensional container of generic data.
- This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.
- As we are going to see, we will generate large training datasets for Neural Network to use.
- We use NumPy, to store that data



## 5.CUDA

- CUDA is a parallel computing platform and programming model for general computing on graphical processing units (GPUs). With CUDA, we are able to dramatically speed up computing applications by harnessing the power of GPUs.
- You can accelerate deep learning and other compute-intensive apps by taking advantage of CUDA and the parallel processing power of GPUs.



Please Install, only if you have a Powerful GPU

# Installing Dependencies

- You need to Install Python's latest version, before installing these dependencies
- After Installing, Python, Run the following listed commands in Python Shell

Dependency	Command to Install
1. Open AI Gym	<code>pip install gym</code>
2. TensorFlow	<code>pip install tensorflow-gpu</code>
3. TFLearn	<code>pip install tflearn- version 1.4</code>
4. NumPy	<code>pip install numpy</code>
5. CUDA	Download it from the NVIDIA Website

# About CartPole

- Now since we are done with setting up the system to facilitate our project, I will discuss about the Crux of the project from this page.
- “**CartPole**” is one of the environments that is provided by Open AI Gym.
  1. We are going to generate some random data first, by letting the default library to play itself.
  2. Then we will generate some training data for Neural Network to use
  3. Then using NumPy, we will store that data, and use it for testing.

- Idea

- A pole is attached by a 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.

- Goal

- The pendulum starts upright, and the goal is to prevent it from falling over
- The Goal is to get a target of 200 for any episode.
- Or an average of 185, for all games

- Reward

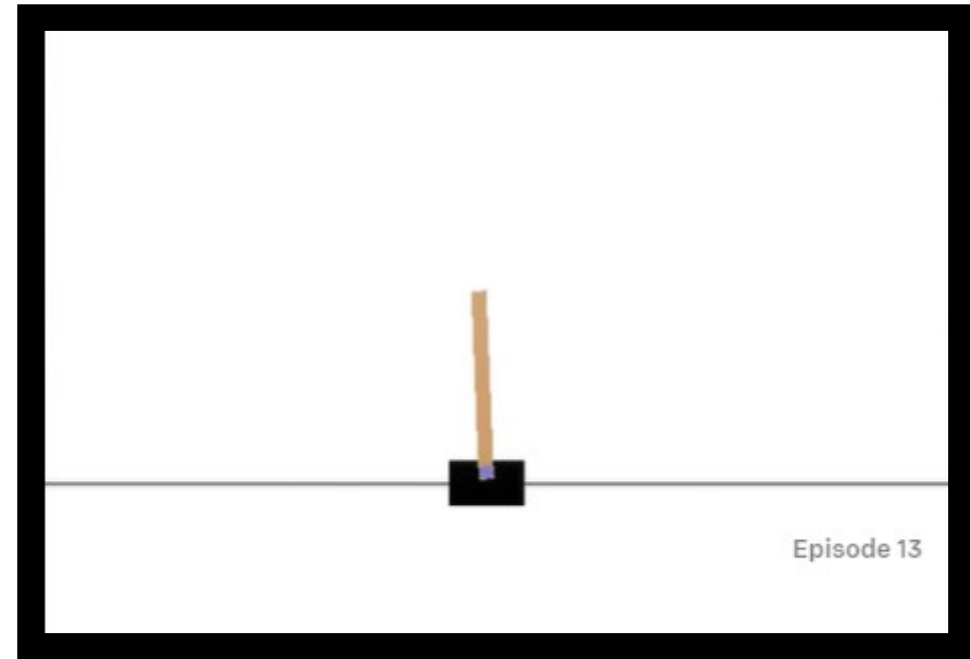
- A reward of +1 is provided for every timestep(frame) that the pole remains upright.

- Reset Condition:

- The episode( one game) ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

For full detailed documentation please visit:

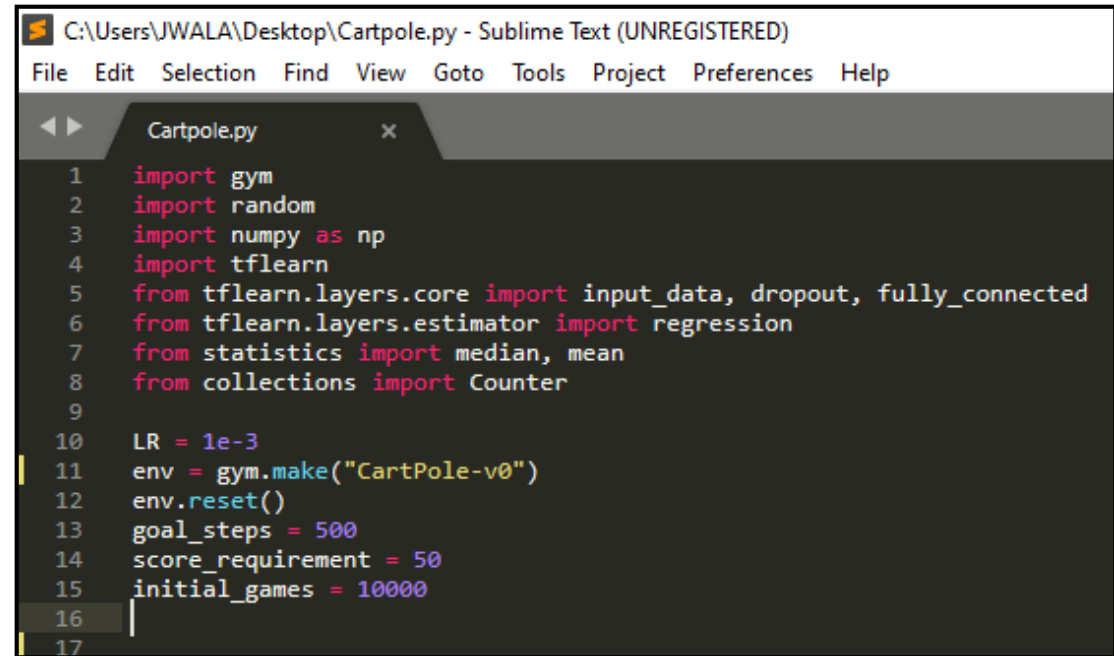
<https://gym.openai.com/docs/>



AG Barto, RS Sutton and CW Anderson, "**Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem**", *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

# Making the Environment

- This code snippet here makes the proper environment, where we train the neural network to solve the given problem
- As you can see, we have imported various installed libraries, defined, No.of Games and winning condition for each game.
- Learning Rate:
  - For this algorithm learning rate is  $1e-3$  that is 0.001



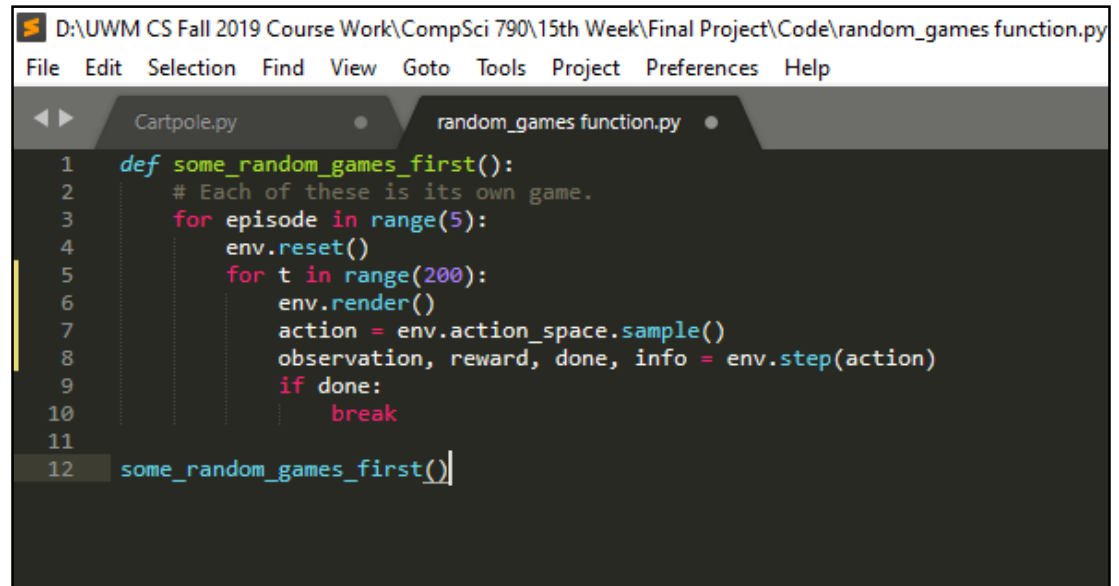
```
C:\Users\JWALA\Desktop\Cartpole.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Cartpole.py
1 import gym
2 import random
3 import numpy as np
4 import tflearn
5 from tflearn.layers.core import input_data, dropout, fully_connected
6 from tflearn.layers.estimator import regression
7 from statistics import median, mean
8 from collections import Counter
9
10 LR = 1e-3
11 env = gym.make("CartPole-v0")
12 env.reset()
13 goal_steps = 500
14 score_requirement = 50
15 initial_games = 10000
16
17
```

Mean, Median and Counter are used to print out the aggregate results of all games in the end.

# Making some random games

- Here we make some random games, about 5
- Each game has 200 frames defined for them but, they won't make it that far
- This is because, the random input values are being generated, which are not consistent and might cause the game to reset, as it is losing.
- I created a sample action that can be used in any other Gym environment.
- In this CartPole environment, the action can be **0** or **1**, which is left or right



```
D:\UWM CS Fall 2019 Course Work\CompSci 790\15th Week\Final Project\Code\random_games function.py
File Edit Selection Find View Goto Tools Project Preferences Help

Cartpole.py random_games function.py

1 def some_random_games_first():
2     # Each of these is its own game.
3     for episode in range(5):
4         env.reset()
5         for t in range(200):
6             env.render()
7             action = env.action_space.sample()
8             observation, reward, done, info = env.step(action)
9             if done:
10                 break
11
12 some_random_games_first()
```

We are executing the environment with an action and in return we will get, the observation of the environment, the reward, if the env is over, and other info.

# Generating the Training Data

- We are generating the Training Data for about 10000 games.
- This is only possible only with the power of GPU. Hence recommended to install GPU version of TensorFlow and CUDA
- This stores the data regarding the games that have higher values than the defined Threshold
- **Note:**
  - If our score is higher than our threshold, we will save every move we made
  - This is the **reinforcement methodology**.
  - Here, we're reinforcing the score, and not trying to influence the machine in any way as to how that score is reached.

```
D:\UWM CS Fall 2019 Course Work\CompSci 790\15th Week\Final Project\Code\Initial_Population.py -
File Edit Selection Find View Goto Tools Project Preferences Help

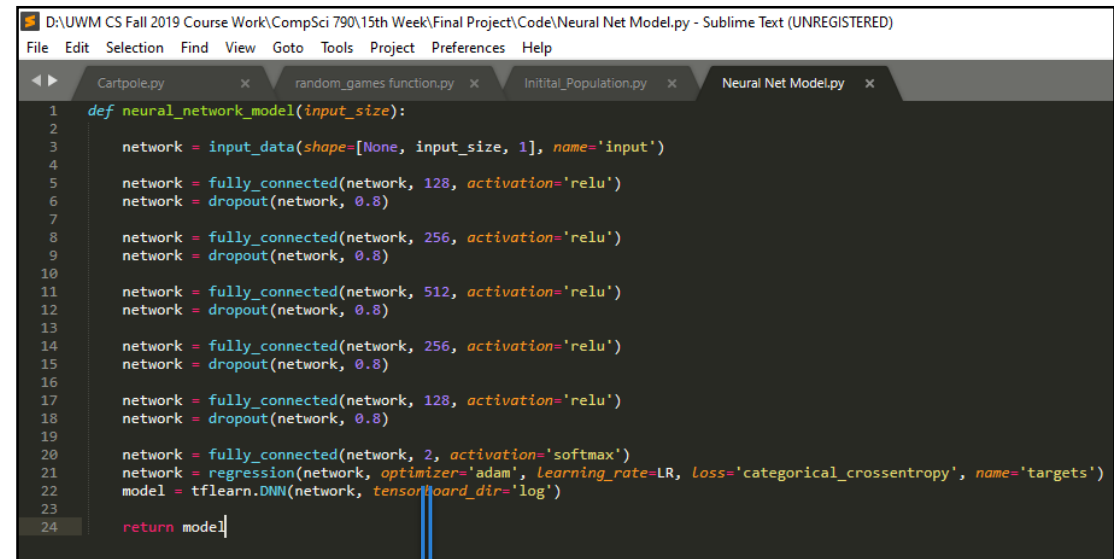
Cartpole.py x random_games function.py x Initial_Population.py x

1 def initial_population():
2     # [OBS, MOVES]
3     training_data = []
4     scores = []
5     accepted_scores = []
6     # iterate through however many games we want:
7     for _ in range(initial_games):
8         score = 0
9         game_memory = []
10        prev_observation = []
11        for _ in range(goal_steps):
12            action = random.randrange(0,2)
13            observation, reward, done, info = env.step(action)
14            if len(prev_observation) > 0 :
15                game_memory.append([prev_observation, action])
16                prev_observation = observation
17                score+=reward
18                if done: break
19            if score >= score_requirement:
20                accepted_scores.append(score)
21                for data in game_memory:
22                    if data[1] == 1:
23                        output = [0,1]
24                    elif data[1] == 0:
25                        output = [1,0]
26                    training_data.append([data[0], output])
27            env.reset()
28            scores.append(score)
29        training_data_save = np.array(training_data)
30        np.save('saved.npy',training_data_save)
31        print('Average accepted score:',mean(accepted_scores))
32        print('Median score for accepted scores:',median(accepted_scores))
33        print(Counter(accepted_scores))
34
35    return training_data
36
```



# Creating the Neural Net

- Here this method takes the “Raw Input Data” generated from `initial_population()` method
- This raw data is not reshaped, for now.
- **Reshape:**
  - During the design of our ML solution, we are going to make decisions regarding speed of processing, accuracy, scalability and other system dependent factors. All of these decisions will drive what transformations are needed for your data.
  - Hence the data is reshaped to facilitate above requirements

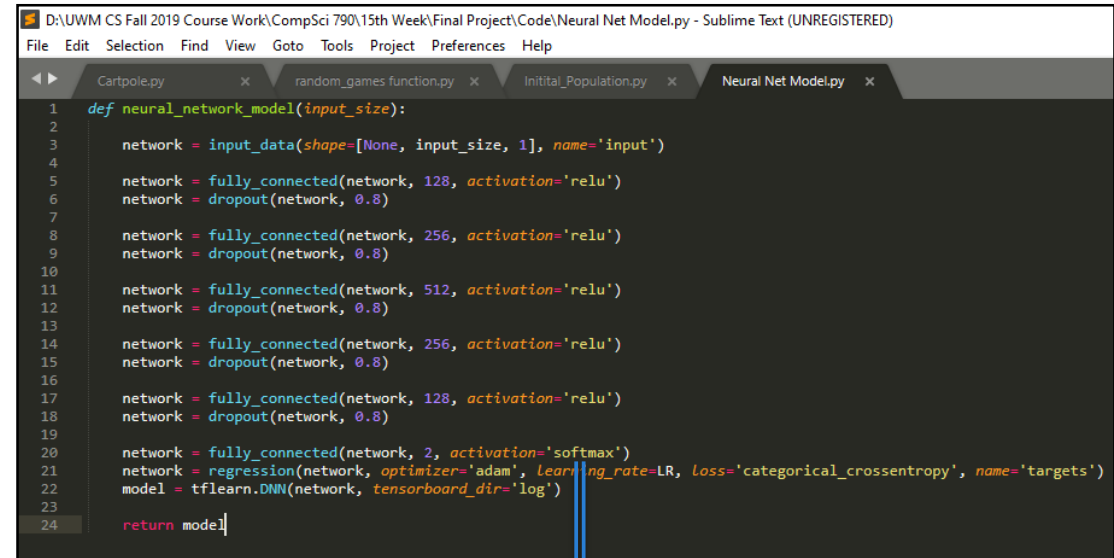


```
def neural_network_model(input_size):  
    network = input_data(shape=[None, input_size, 1], name='input')  
    network = fully_connected(network, 128, activation='relu')  
    network = dropout(network, 0.8)  
    network = fully_connected(network, 256, activation='relu')  
    network = dropout(network, 0.8)  
    network = fully_connected(network, 512, activation='relu')  
    network = dropout(network, 0.8)  
    network = fully_connected(network, 256, activation='relu')  
    network = dropout(network, 0.8)  
    network = fully_connected(network, 128, activation='relu')  
    network = dropout(network, 0.8)  
    network = fully_connected(network, 2, activation='softmax')  
    network = regression(network, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')  
    model = tflearn.DNN(network, tensorboard_dir='log')  
    return model
```

- **Adam** is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks.
- The algorithm leverages the power of adaptive learning rates methods to find individual learning rates for each parameter.

# Creating the Neural Net

- Here we are creating 5 Layers with 128,256,512,256 and 128 nodes respectively.
- All are fully connected.
- **Activation Functions:**
  - **Softmax:** for output layer
  - **ReLU ( Linear Rectified Units ):** for all layers, except for output layer.



```
1 def neural_network_model(input_size):
2
3     network = input_data(shape=[None, input_size, 1], name='input')
4
5     network = fully_connected(network, 128, activation='relu')
6     network = dropout(network, 0.8)
7
8     network = fully_connected(network, 256, activation='relu')
9     network = dropout(network, 0.8)
10
11    network = fully_connected(network, 512, activation='relu')
12    network = dropout(network, 0.8)
13
14    network = fully_connected(network, 256, activation='relu')
15    network = dropout(network, 0.8)
16
17    network = fully_connected(network, 128, activation='relu')
18    network = dropout(network, 0.8)
19
20    network = fully_connected(network, 2, activation='softmax')
21    network = regression(network, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')
22    model = tflearn.DNN(network, tensorboard_dir='log')
23
24    return model]
```

- The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1.
- The output values are between the range [0,1] which is nice because we are able to avoid binary classification and accommodate as many classes or dimensions in our neural network model

# Creating the Neural Net

This is an implementation of LSTM Cells

- **Loss Function:**

- Machines learn by means of a loss function.
- It's a method of evaluating how well specific algorithm models the given data.
- If predictions deviates too much from actual results, loss function would output a very large number.
- Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.

## Cross Entropy Loss/Negative Log Likelihood:

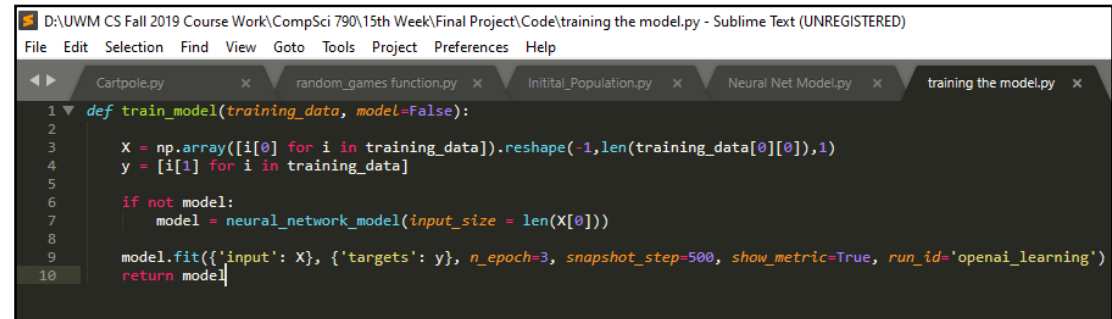
Here I used, this, which is the most common setting for classification problems. Cross-entropy loss increases as the predicted probability diverges from the actual label.

A screenshot of a Sublime Text editor window titled "D:\UWM CS Fall 2019 Course Work\CompSci 790\15th Week\Final Project\Code\Neural Net Model.py - Sublime Text (UNREGISTERED)". The editor shows a Python script for a neural network model. The script defines a function `neural_network_model(input_size)` which builds a neural network with four hidden layers and one output layer. The layers are fully connected with ReLU activation and 0.8 dropout. The output layer is fully connected with a softmax activation. The model is trained using the Adam optimizer and categorical crossentropy loss. The script is as follows:

```
1 def neural_network_model(input_size):
2
3     network = input_data(shape=[None, input_size, 1], name='input')
4
5     network = fully_connected(network, 128, activation='relu')
6     network = dropout(network, 0.8)
7
8     network = fully_connected(network, 256, activation='relu')
9     network = dropout(network, 0.8)
10
11    network = fully_connected(network, 512, activation='relu')
12    network = dropout(network, 0.8)
13
14    network = fully_connected(network, 256, activation='relu')
15    network = dropout(network, 0.8)
16
17    network = fully_connected(network, 128, activation='relu')
18    network = dropout(network, 0.8)
19
20    network = fully_connected(network, 2, activation='softmax')
21    network = regression(network, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')
22    model = tflearn.DNN(network, tensorboard_dir='log')
23
24    return model
```

# Training the Model

- Here we are reshaping the data for output layer first, to get the best possible results
  - **Epochs:**
    - No. of Epochs : 3
    - Too many epochs will result in overfitting
  - **Snapshot Steps: 500**
- 
- Model ensembles can achieve lower generalization error than single models, but are expensive in Computational cost
  - An alternative is to train multiple model snapshots during a single training run and combine their predictions to make an ensemble prediction.

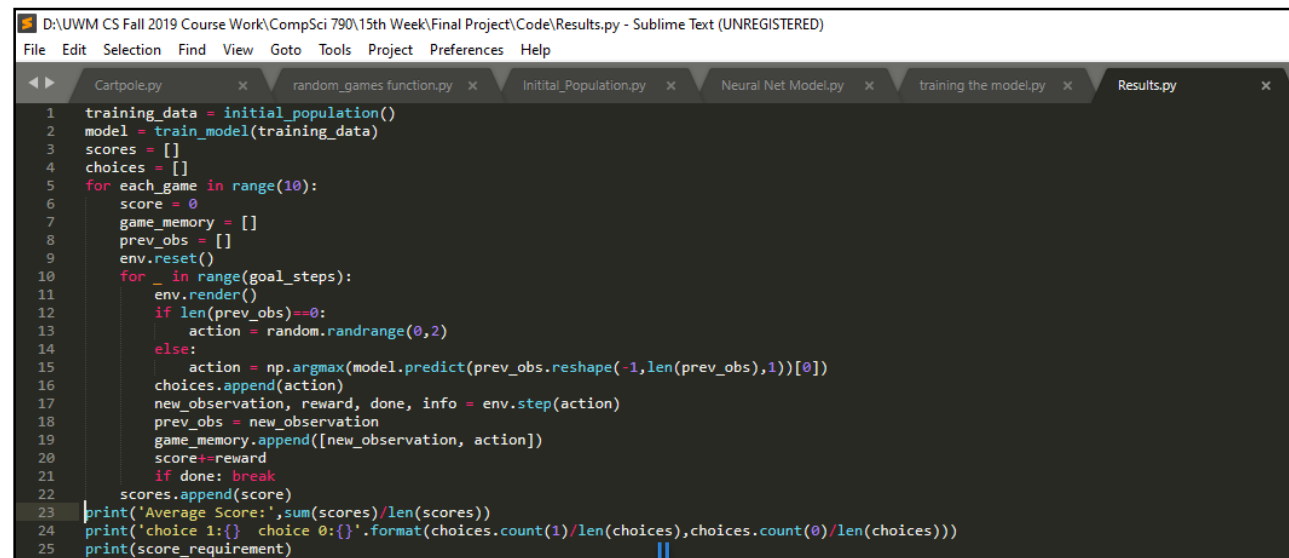


```
D:\UWM CS Fall 2019 Course Work\CompSci 790\15th Week\Final Project\Code\training the model.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

1 def train_model(training_data, model=False):
2
3     X = np.array([i[0] for i in training_data]).reshape(-1,len(training_data[0][0]),1)
4     y = [i[1] for i in training_data]
5
6     if not model:
7         model = neural_network_model(input_size = len(X[0]))
8
9     model.fit({'input': X}, {'targets': y}, n_epoch=3, snapshot_step=500, show_metric=True, run_id='openai_learning')
10    return model
```

# Results for each game

- This code snippet, prints out the accuracy and loss values for each game, after testing on 10 games, with the trained model and also their average.



```
D:\UWM CS Fall 2019 Course Work\CompSci 790\15th Week\Final Project\Code\Results.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Cartpole.py x random_games function.py x Initial_Population.py x Neural Net Model.py x training the model.py x Results.py x

1 training_data = initial_population()
2 model = train_model(training_data)
3 scores = []
4 choices = []
5 for each_game in range(10):
6     score = 0
7     game_memory = []
8     prev_obs = []
9     env.reset()
10    for _ in range(goal_steps):
11        env.render()
12        if len(prev_obs)==0:
13            action = random.randrange(0,2)
14        else:
15            action = np.argmax(model.predict(prev_obs.reshape(-1,len(prev_obs),1))[0])
16        choices.append(action)
17        new_observation, reward, done, info = env.step(action)
18        prev_obs = new_observation
19        game_memory.append([new_observation, action])
20        score+=reward
21        if done: break
22    scores.append(score)
23    print('Average Score:',sum(scores)/len(scores))
24    print('choice 1:{ } choice 0:{ }'.format(choices.count(1)/len(choices),choices.count(0)/len(choices)))
25    print(score_requirement)
```

- Here we are finding the ratio among the things ( which are choices here ) that are being predicted.
  - These choices are: **Either moving left ( Choice 0 ) or moving right ( choice 1 ).**

# Please watch the video attached

- Before you proceed to see the end result in the next slide, please watch the video which will show you the running prototype of this project
- Please turn on the volume
- Please skip to 1:15 in the video, if you would want to see the actual demonstration.

# Result

- As you can see, The average score for all 10 testing games is 200, which is greater than 185.
- Hence this problem is considered solved.
- If you observe, we got a ratio of Choice 1 and Choice 2 too.
- Those are in (0,1) range, justifying the use of Softmax function for Output layer.

```
<0x1b>[2K
Training Step: 1052 | total loss: <0x1b>[1m<0x1b>[32m0.65775<0x1b>[0m<0x1b>[0m | time: 1.476s
<0x1b>[2K
| Adam | epoch: 003 | loss: 0.65775 - acc: 0.6170 -- iter: 22400/22447
Training Step: 1053 | total loss: <0x1b>[1m<0x1b>[32m0.66691<0x1b>[0m<0x1b>[0m | time: 1.481s
<0x1b>[2K
| Adam | epoch: 003 | loss: 0.66691 - acc: 0.5975 -- iter: 22447/22447
--
Average Score: 200.0
choice 1:0.4975 choice 0:0.5025
50
[Finished in 51.1s]
```

188, 179 and 197 were some other results that I got when I ran it before this go. The average is >185

# Conclusion

- It would be highly unfair of me to say that, I had included everything regarding the capabilities of TensorFlow and TfLearn and their potential, to solve problems like Open AI Gym
- **Implementation of RNN's in the game development has already started to take place**, in form of Non-Playable Characters (NPC's) in games like Dota 2 and The Legend of Zelda.



# Conclusion

- Well known, Google's Alpha Go and Deep Blue actually demonstrate the RNN's power in playing games!
- I believe this project of mine is nothing but a glimpse of what future holds for A.I in Video Games!

