Project proposal: Evolutionary ANN to play Games

** use visual studio code or any other standard IDE. Using Jupyter notebook for this project, would introduce overheads leading to failing kernels*

"Developing agents that can accomplish challenging tasks in complex, uncertain environments are a key goal of artificial intelligence. Recently, the most popular paradigm for analyzing such problems has been using a class of reinforcement learning (RL) algorithms based on the Markov Decision Process (MDP) formalism and the concept of value functions. Successes of this approach include systems that learn to play Atari from pixels [Mnih et al., 2015], perform helicopter aerobatics Ng et al. [2006], or play expert-level Go [Silver et al., 2016]. "

- Salimans, Tim, et al. "Evolution strategies as a scalable alternative to reinforcement learning." arXiv preprint arXiv:1703.03864 (2017).

An alternative, more basic approach to solve such problems could be through the use of Simple Artificial Neural Networks (ANN) with the support of Evolutionary Algorithms such as Genetic Algorithms (GA). The objective of this project is to introduce you to agents (ANNs) don't have training datasets but learn based on their interaction with its environment and evolve over a number of generations using GAs, to better interact with its environment.

In this project, you will create a population of untrained agents that is gradually able to learn from its environment to better balance a pole attached by an un-actuated joint to a cart that moves along a frictionless track. OpenAI GYM[1] provides you an environment to render/visualize this interactive environment, that allows your agents to observe the environment and perform actions that would balance the pole. 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 timestep 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. Additional details on setting up the environment can be found here: https://gym.openai.com/docs/

You are required to create a population of ANNs that over multiple generations learn from its environment to better balance the pole. You will generate a n population of ANNs with 3 layers: i-input neurons in the 1st layer, h-hidden neurons in the 2nd layer and o-output neurons in the 3rd layer. A population of ANNs means that you will generate n-numbers of ANNs with the above structure each having random weights and biases. You will also initialize/train each ANNs with only one random set of inputs resulting in a random output. Run the ANNs over multiple generations to create a population that is on average better able to balance the pole. In each generation, each of the ANN would try to balance the pole by reading the observation from the environment and predicting an action. If the prediction is right the ANN retrains itself with the new observation and the predicted action. If the prediction is wrong, it stops training in that generation. After all the ANNs have stopped training in a given generation, two ANNs are chosen from the population having the best scores to create two children. The children have part of their weights and bias from each of their parents. After, creating the same number population of children, we retire the parents. The next generation of ANNs has the children ANNs and we continue this cycle until we exhaust the number of specified generations.

[1] https://gym.openai.com/envs/CartPole-v1/

Once we have executed the final generation, choose the network with the best score and also generate another network with the average weights and bias of the population. We render these two networks/agents to balance the pole on screen.

In brief the following are the expectations from this project. You can take the following as suggestions. The goal is that you use the best practices that you learned during this course on numpy number crunching, vectorization, ML and EA to train ANNs over multiple generation using GA to solve the CartPole-v1 problem

1. Setup OpenAI GYM correctly and render the CartPole-v1 environment. You do this to verify that your environment is up and running with all necessary pre-requisites present.
2. Specify an initial population size of ANNs. Between 10 to 100, depending on your CPU capacity. Suggested: 50
3. Specify number of generations that you want to evolve the population over between 10 to 20. Suggested: 15
4. Specify a mutation rate of 0.1 to 0.001. Suggested: 0.001
5. Specify the number of iterations to run over. In case of CartPole-v1, we can specify it as np.inf. Each ANN in a generation, runs until this max iteration or fails because it could not balance the pole, whichever is first
6. Create an Initial Population of ANN Classifiers. You can use Scikit-learn MLPClassifier[2].

```
-   MLPClassifier(batch_size=1,   max_iter=1,   solver='sgd',   activation='relu',
    learning_rate='invscaling',
-       hidden_layer_sizes=hlayer_size,
-       random_state=1)
```

- You will have only one hidden layer with number of neurons as **2/3 the size of the input layer, plus the size of the output layer**.

```
-       partial_fit(np.array([env.observation_space.sample()]),
-       np.array([env.action_space.sample()] ),
-       classes=np.arange(env.action_space.n)
```

- Partially train each ANN with a random input and output
- The **env.observation_space.sample()** creates a random input tuple, were each value is an input to the input neurons
- env.action_space.sample() generates a random output
- Since we are using partial fit, meaning that we will be updating / retraining the model later, we also provide all possible list of classes/output values for the neurons using the **classes** attribute. Here **classes=np.arange(env.action_space.n),** where **env.action_space.n** defines the total number of output actions starting from 0.
- Return this initial population of ANNs

7. Iterate over the number of generations to evolve the initial population over each generation

- Calculate the score of each ANN
    i. Use each ANN to balance the pole in the CartPole-v1 environment.
    ii. Reset the environment first using **env.reset()** before running an ANN
    iii. You may use **env.render()** to display these trial runs on screen. time.sleep(0.05) can also be used to delay the rendering of each frame

[2] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

iv. For the first time your run an ANN in a generation pass a random action to generate the results/**observation** from the environment when you performed that action. It can be done by using **env.step(action)**. It will also give you a **reward** if the action is successful, a **done** value that indicates if the ANN failed to balance the pole after performing the action. An **info** value is also returned that provides diagnostic information. We can ignore this.
v. We provide the retrieved observation to the network's **predict** method to predict the next **action** based on this observation
vi. As a sanity check we also check after predicting this next action, that the next action is not repeating since the past 5 steps. We do this because, we start with an untrained ANN and there is a good chance at the beginning that we will get the same prediction for every observation. If we do see that the action is being repeated since the last 5 steps, we reassign the predicted next action for the to a random action using env.action_space.sample()
vii. If the environment hasn't failed (done=False) or we haven't exhausted the max number of iterations per network, update the network by calling **partial_fit** and now provide the recorded observation from performing the previous action as X features and the next action as y predictor
viii. We recursively run this process until the done condition is fulfilled or we exhaust max iteration.
ix. After the fulfilment of a recursive call we return the reward value and add it to the parent reward. Ultimately getting to total reward for the specific ANN

- Calculate the probabilities of each ANN by dividing the total reward of each ANN by the sum of total rewards of all the ANNs in the given generation
- Select two parent ANNs based on the probability distribution you calculated above to generate two children for the pair of parent
    i. A network's **.coefs_** attribute gives you the weights between the input to hidden neurons and hidden neurons to output neurons. Suggestion: You can assign to them as well
    ii. A network's **. intercepts_** attribute gives you the bias between the input to hidden neurons and hidden neurons to output neurons. Suggestion: You can assign to them as well
    iii. Perform crossover (on rows) to select part of the weights from one parent and another part from the second parent
    iv. Perform the same for biases
    v. Perform these steps to generate a new population of the same size as the previous
    vi. Depending on the mutation rate, also perform mutation as an when required on the weights and bias.
    vii. *Refer to the lecture on Evolutionary Algorithms GA for more information*
    viii. *Suggestion, but you should use your own strategies*
        1. *Crossover: select a start and end index randomly. Use these indices to copy/swap part of the weights and bias in each level (input to hidden or hidden to output) for the parents*
        2. Mutation: Randomly choose 0 or 1 with probability (1 – mutation rate) for 0 and (mutation rate) for 1. If mutation is 1, randomly select a level and a row index. Swap them between the parents

ix. Return the new population for the next generation

- Perform the same process on the new generation, until you exhaust the number for generation iterations.
- In each generation iteration report on the average and best score/reward of the population. Also Identify the network that performed the best score

8. After all the generations are executed. Select the best network. Also create a new network based on the average weights and bias over all levels.
9. Run these two networks separately over the CartPole-v1 environment. With env.render() and time.sleep(0.1) to visualize them