Question 1: MountainCar-v0 and MountainCarContinuous-v0 (Policy Gradient Methods)

Use MountainCar-v0 and MountainCarContinuous-v0 from OpenAI gym for below questions.

Consider two environments having different action spaces - Discrete and Continuous. MountainCar-v0 has a descrete action space with three possible actions; while MountainCarContinuous-v0, which is an extended version of the MountainCar-v0, operates in a continuous action space.

Demonstrate the efficacy of the following family of algorithms in both environments and compare **(5 points)**. For this, plot the number of iterations vs reward graphs for each method. Indicate variance in the reward for each method in the same plot.

1. **Vanilla Policy Gradient [2x5 points]:** Learn the set of parameters θ which approximates the function $\pi(a|s;\theta)$ to obtain "expected" maximum return.
2. **Policy Gradients with Baselines [2x5 points]:** Ensure your baselines are not learnt in this case. Justify the baselines you have put in place. Compare the variance in rewards versus Vanilla Policy Gradient.
3. **Actor-Critic Method [2x5 points]:** In this method, you have to train an additional network to predict a state dependent baseline - $V(s;\theta_b)$.
4. **Gaussian Policy [5 points]:** It's often advantageous to predict a range of actions in the continuous space which are favourable rather than a single action. Demonstrate a functioning Gaussian Policy network in the suitable environment for the same. Compare training and test dynamics and comment on the observations with a case where the solution predicts only the mean.

Implementation Hints and Guidelines

- Use tried and tested network architectures from implementations available somewhere online and similarly pre-processing and loading codes (attribute authors correctly). **Implementing the RL side of things is important in this assignment** and not tinkering with architectural choices or pre-processing.
- Figure out how to write the forward pass defining the loss functions (eg: PyTorch). Do not write gradient code yourself. **Indicate in your reports how a differentiable path exists from the composition of modules enabling automatic-differentiation**.
- Use standard deep learning techniques for stability and testing. Ensure the networks which represent your θ and θ_b overfit to a neat function (before plugging into the actual environment) with well-defined input outputs.
- Over each iteration where you compute the loss from several Monte-Carlo roll-outs, normalize the rewards such that half of them are positive and the other half are negative (subtract mean, divide by variance). At each update step, the network should reject half actions and reinforce the other half.
- Below are some good resources to read before starting the assignment [credits]
  - A Recipe for Training Neural Networks
  - Troubleshooting Deep Neural Networks
  - Things I wish we had known before we started our first Machine Learning project
  - How to unit test machine learning code
  - Practical Advice for Building Deep Neural Networks

- Few other resources [optional read]
  - Deep Reinforcement Learning: Pong from Pixels
  - Log Derivative Trick
  - The Policy of Truth
  - Automatic Differentiation

Submission Instructions

- Submit your source code in main.ipynb, also exported into a main.html. No trained models.
- You are expected to use pyTorch for this assignment.
- Clearly indicate at the start which components you have attempted.
- Write suitable comments to describe the methods you have implemented.
- Provide proper attribution to any reference implementations you used.

Plagiarism Policy: Plagiarism detection software is guaranteed to be run before any evaluation. Trying to beat any such software will make your code significantly unreadable and easy to prove malicious intent. In case of heavy plagiarism - all parties involved (giver, taker) will get a 0.