Assignment 2 for CMPUT 653 Deep Policy Gradient Methods

Spyros Orfanos
October 18, 2020

Q1.

Table 1: Summary of Neural Network Architecture & Hyperparameters

| **Architecture/HP** | **Incremental** | **Batch** | **My** |
|---|---|---|---|
| Neurons – L1 | 128 | 128 | 150 |
| Neurons – L2 | 128 | 128 | 50 |
| Activation | ReLU | ReLU | ReLu |
| Output Layer | LogSoftMax | LogSoftMax | LogSoftMax |
| Learning Rate (Adam) | 0.003 | 0.003 | 0.003 |

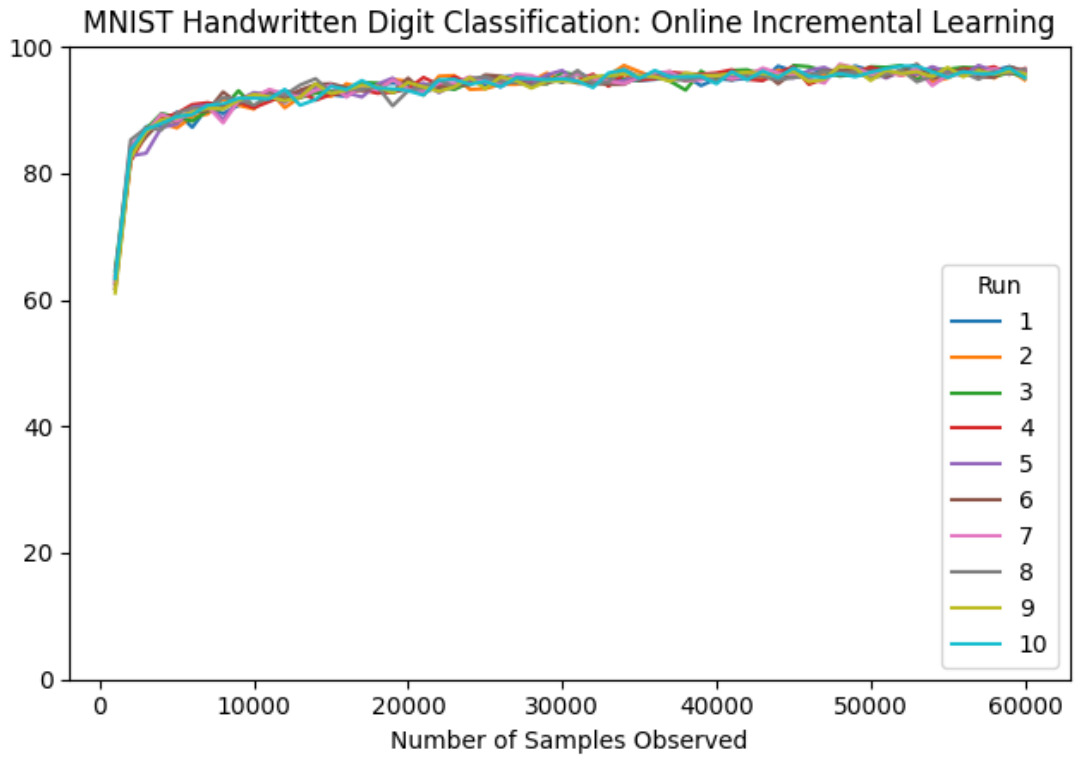
Table 2: Batch Parameters

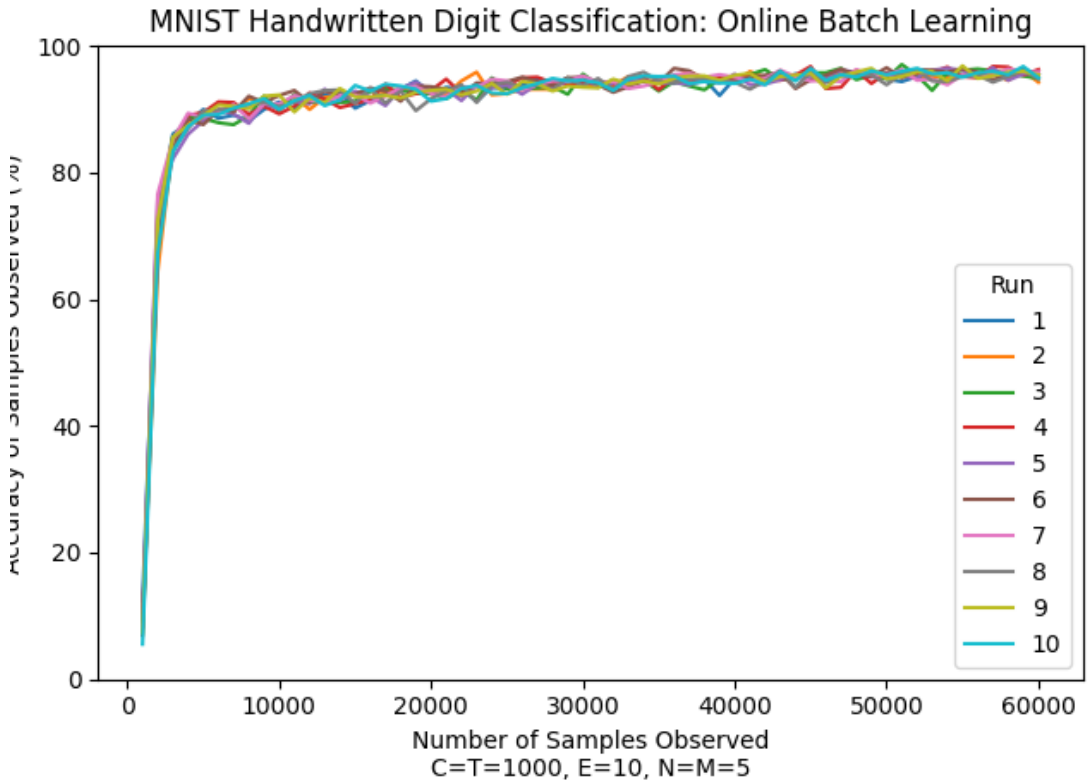
| Parameter | Batch | My |
|---|---|---|
| T | 1000 | 2000 |
| C | 1000 | 2000 |
| E | 10 | 10 or 20 |
| N | 5 | 5 |
| M | 5 | 5 |

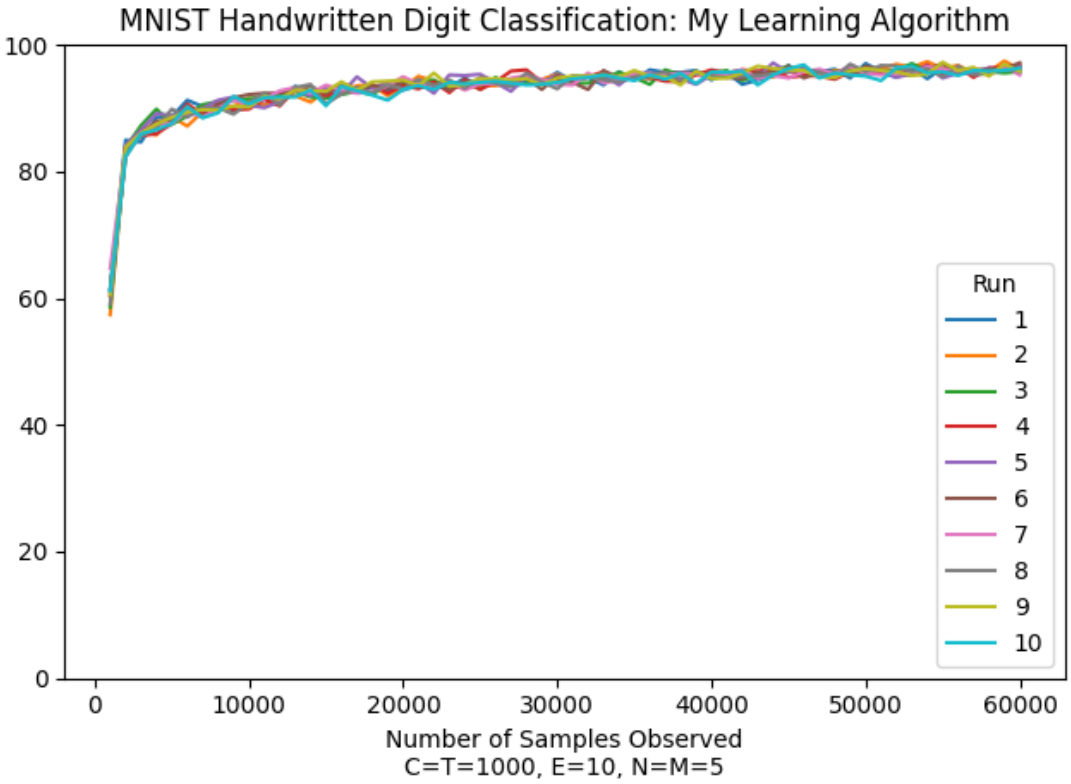
For my algorithm, I do an immediate update of the current buffer once I have a prediction with confidence below $p_t^*$ or when the buffer fills up. In the former case, I use twice the number of epochs. At beginning of training, this will act like incremental learning, but as the model improves there will be more full-batch updates. The idea is that it updates on low confidence predictions as soon as possible, and more often. This algorithm has the following pseudocode:

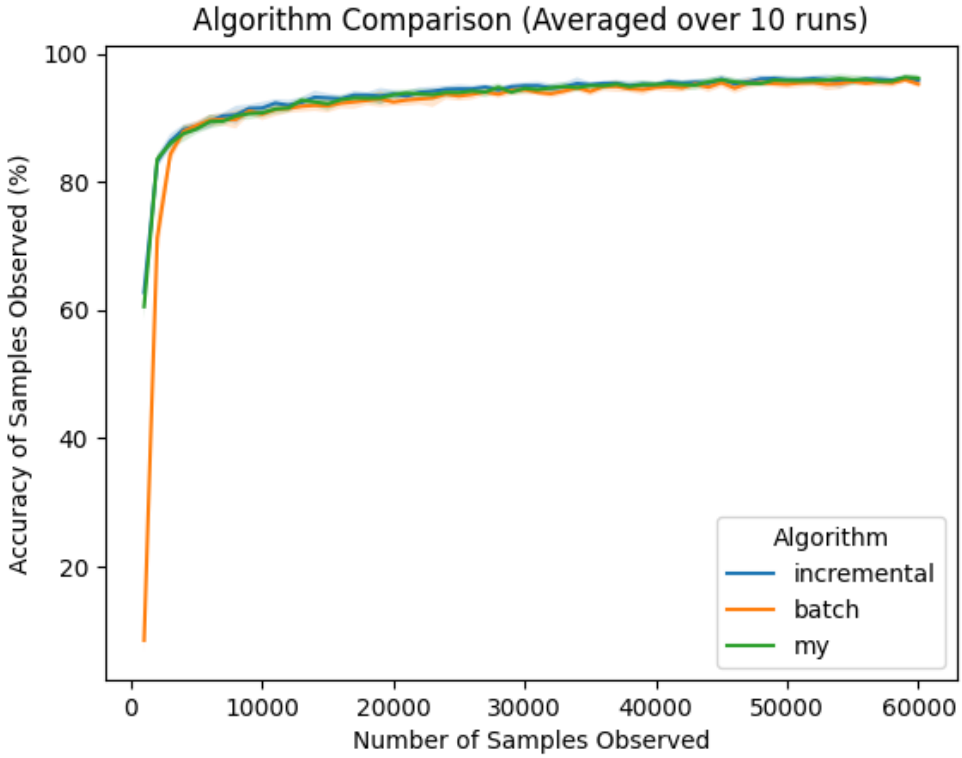
```
Initialize network with parameter θ and size TH
Initialize Buffer b with Capacity C

For t from 1 to Z:  # Of cycle time tau

    Observe input

    Predict based on the newest available θ

    Incur error/reward

    If prediction_confidence < p_t*

        If buffer.isBigEnough() = True

            θ ← Learn2(b, θ, E, 2*N, M)

        Else

            θ ← Learn2(b, θ, 1, 1, 1)

    Else if t mod T = 0:

        θ ← Learn2(b, θ, E, 2*N, M)
```

Figures 1-4: Algorithm Performance on MNIST Dataset




MNIST Handwritten Digit Classification: My Learning Algorithm
100
80
60
40
20
0
0
10000
20000
30000
40000
50000
60000
Number of Samples Observed
C=T=1000, E=10, N=M=5
Run
1
2
3
4
5
6
7
8
9
10


Algorithm Comparison (Averaged over 10 runs)
Accuracy of Samples Observed (%)
100
80
60
40
20
0
10000
20000
30000
40000
50000
60000
Number of Samples Observed
Algorithm
incremental
batch
my

Q2.

In _my.py, I tested PPO without the special clipping objective.

Table 3: Neural Network Architecture & Hyperparameters

| **Architecture/HP** | **Actor (Reinforce, Batch AC, PPO, My)** | **Critic (Batch AC)** | **Critic (PPO, My)** |
|---|---|---|---|
| Neurons – L1 | 32 | 32 | 32 |
| Neurons – L2 | 16 | 32 | 32 |
| Activation | ReLU | ReLU | ReLu |
| Output Layer | SoftMax | Linear | Linear |
| Learning Rate (Adam) | 0.001 | 0.001 | 0.005 |

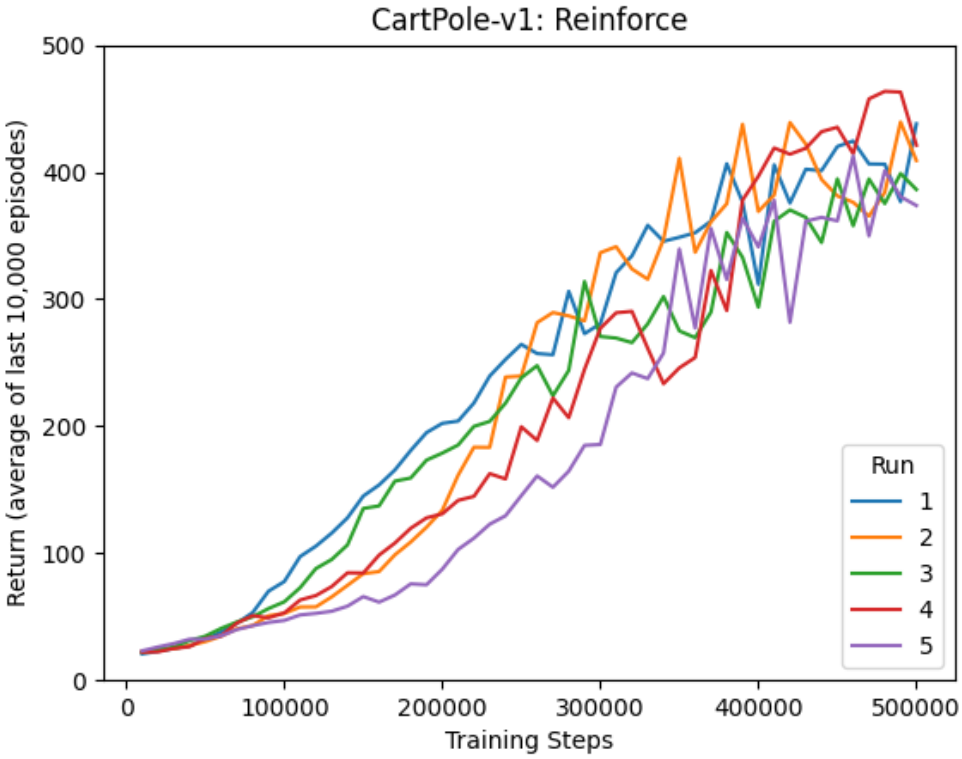
Table 4: Batch Parameters

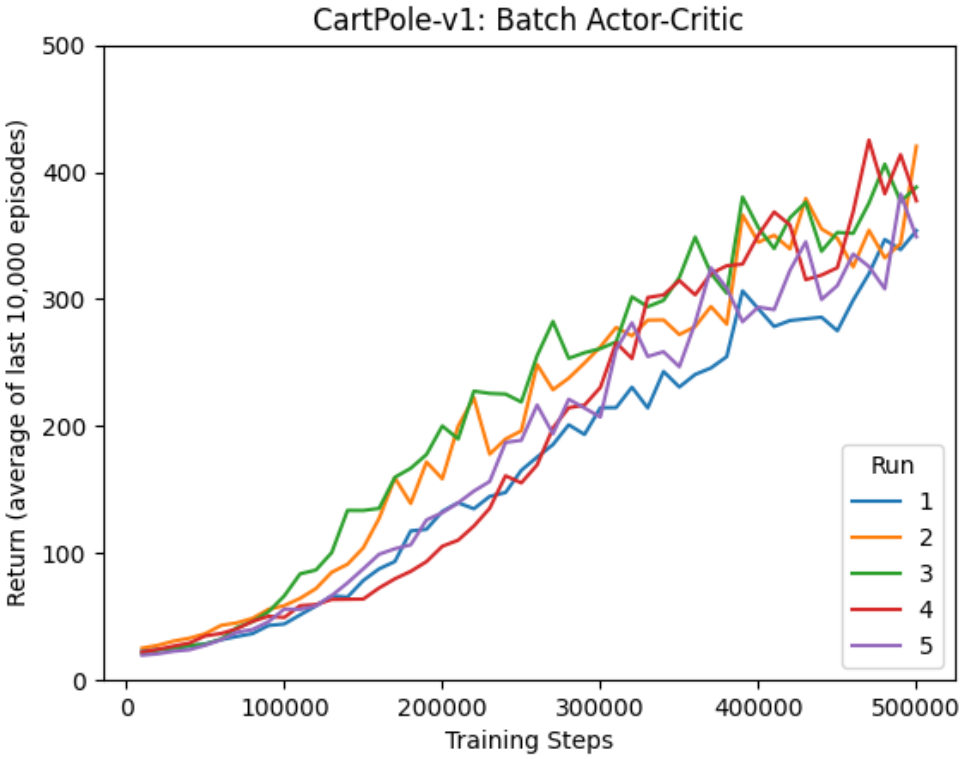
| **Hyper-parameter** | **Reinforce** | **Batch AC** | **PPO** | **My** |
|---|---|---|---|---|
| K (episodes/batch) | 20 | 30 | 20 | 200 |
| E | N/A | N/A | 10 | 5 |
| N | N/A | N/A | 20 | 6 |
| M | N/A | N/A | 20 | 6 |

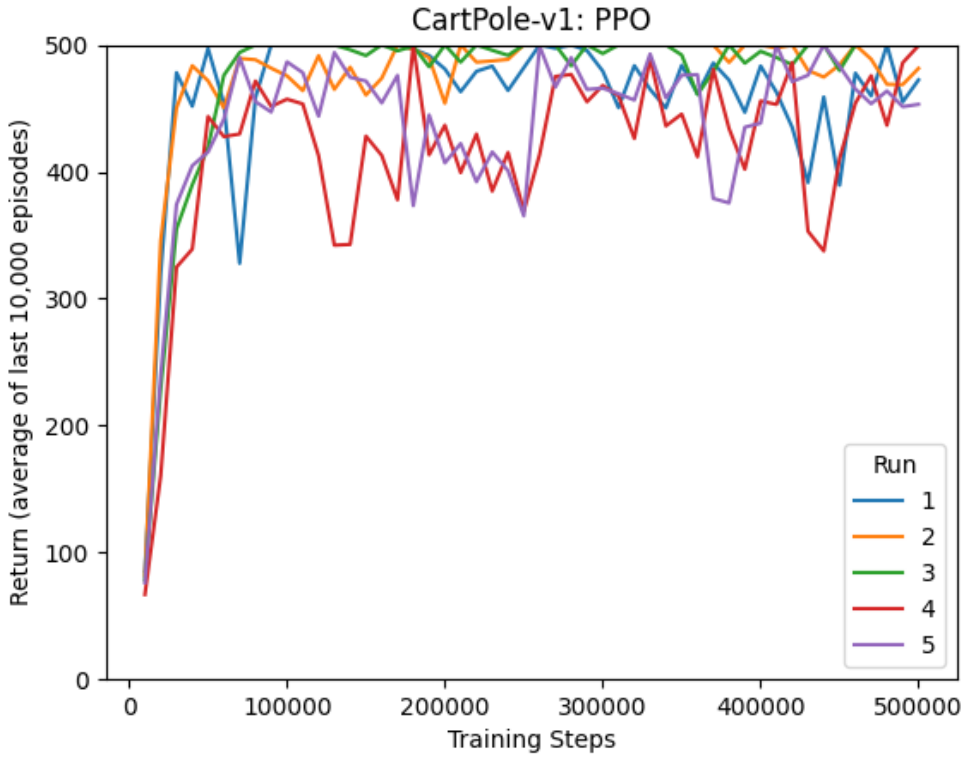
Table 5: Critic Parameters

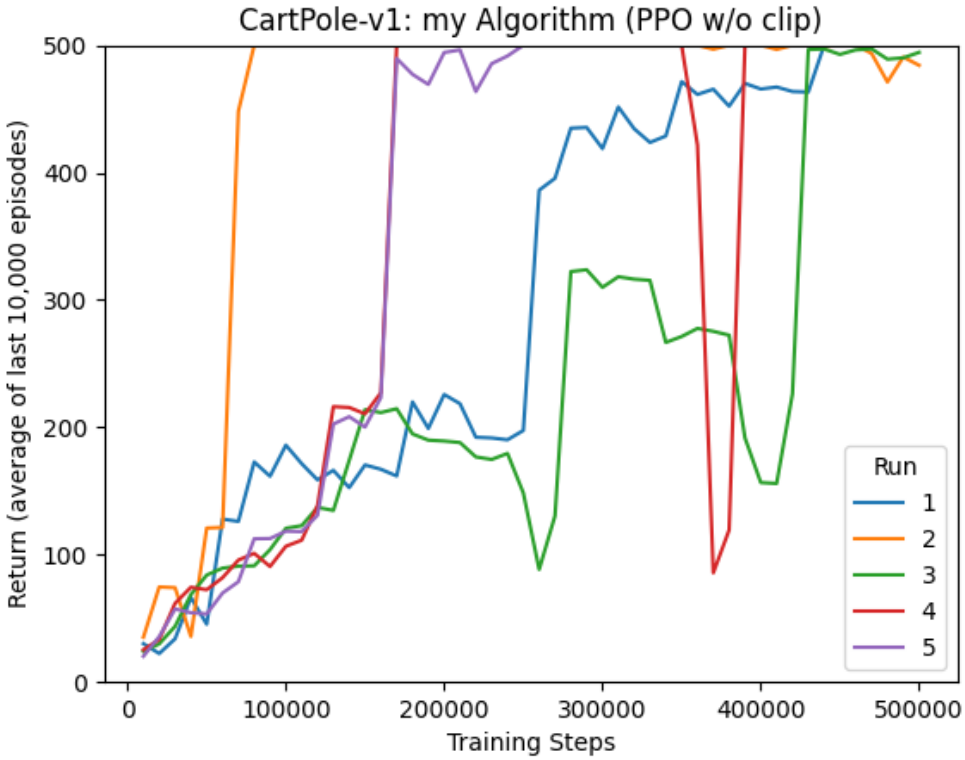
| **Hyper-parameter** | **Batch AC** | **PPO** | **My** |
|---|---|---|---|
| Gamma | 1 | 1 | 1 |
| Lambda | 1 | 1 | 1 |
| Epsilon | N/A | 0.2 | N/A |

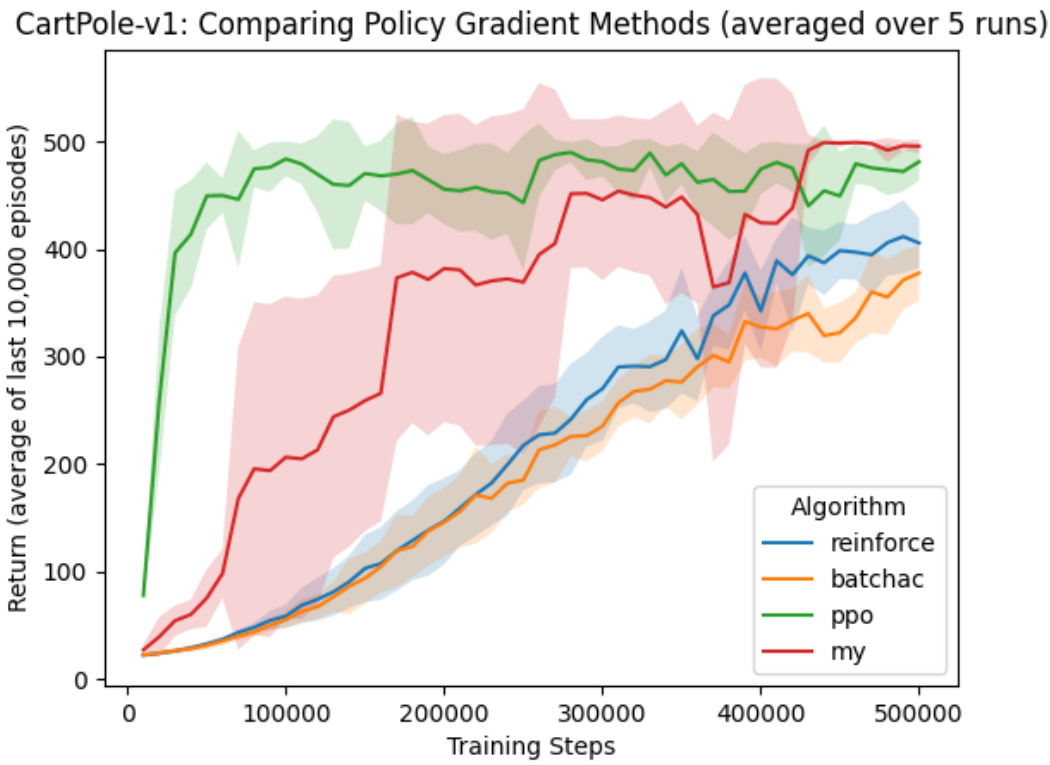
Figure 5-10: Algorithm Performance for CartPole-v1


CartPole-v1: Reinforce
Return (average of last 10,000 episodes)
Training Steps
0
100
200
300
400
500
0
100000
200000
300000
400000
500000
Run
1
2
3
4
5


CartPole-v1: Batch Actor-Critic
Return (average of last 10,000 episodes)
Training Steps
0
100
200
300
400
500
0
100000
200000
300000
400000
500000
Run
1
2
3
4
5


CartPole-v1: PPO
Return (average of last 10,000 episodes)
Training Steps
0
100
200
300
400
500
0
100000
200000
300000
400000
500000
Run
1
2
3
4
5


CartPole-v1: my Algorithm (PPO w/o clip)
Return (average of last 10,000 episodes)
Training Steps
0
100
200
300
400
500
0
100000
200000
300000
400000
500000
Run
1
2
3
4
5


CartPole-v1: Comparing Policy Gradient Methods (averaged over 5 runs)
Return (average of last 10,000 episodes)
Training Steps
0
100
200
300
400
500
0
100000
200000
300000
400000
500000
Algorithm
reinforce
batchac
ppo
my