CS525, Assignment 3

Hanshen Yu

Worcester Polytechnic Institute

Student ID: 664892122

Individual Assignment

CS525, Assignment 3

I followed the instructions of in class instructions, pytorch DQN tutorial, paper offered[1] and other online resources[2][3]. I found a blog interesting[4][5], it mentioned how to tune the params to get a similar result from DeepMind breakout results of over 400.

Include a section describing the set of experiments that you performed
I trained an agent with Double DQN, with a CNN to learn how to play Atari breakthrough, with the suggested GCP settings.

At first I tried vanilla DQN with all suggested params, but it only gives me around 16 mean reward in test. I later added some improvements, and implemented double DQN to make the performance better, yet still not changing the params. I got a mean reward of 73 in 11.5k episodes and 5 hrs of training time. The training is still going on.

what structures you experimented with (i.e., number of layers, number of neurons in each layer)
I strictly followed the suggested paper to set up the structure of neural network, the calculation process is mentioned in the comments.

```
# this structure is mentioned in this vanilla paper, quote as below:
# The exact architecture, shown schematically in Fig. 1, is as follows.
# The input tothe neural network consists of an 84x84x4 image produced by the preprocess-ing mapw.
# The first hidden layer convolves 32 filters of 8x8
# with stride 4 with the input image and applies a rectifier nonlinearity[31,32].
self.conv1 = nn.Conv2d(4, 32, kernel_size=8, stride=4)
self.bn1 = nn.BatchNorm2d(32)
# The second hidden layer con-volves 64 filters of 434 with stride 2,
# again followed by a rectifier nonlinearity.
self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
self.bn2 = nn.BatchNorm2d(64)
# This is followed by a third convolutional layer that convolves 64 filters of 3x3
# with stride 1 followed by a rectifier.
self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)
self.bn3 = nn.BatchNorm2d(64)
# The final hidden layer is fully-connected and con-sists of 512 rectifier units.
## here it is (9-3+1)^2*64=7*7*64=3136
self.fc4 = nn.Linear(3136, 512)
# The output layer is a fully-connected linear layer with a single output for each valid action.
# The number of valid actions varied between 4 and 18 on the games we considered.
self.fc5 = nn.Linear(512, 4)
```

what hyperparameters you varied (e.g., number of epochs of training, batch size and any other parameter values, weight initialization schema, activation function)
For training, I made a loading mode so that the training could start from where it paused. Fortunately, it helped when I was accidentally disconnected. For this reason, I got more than 1 sessions in training, the graph is split but nothing else is ruined.

During testing, I find that the agent sometimes get stuck in a loop. It might be because the score is so high and it cannot hit a brick, or some bugs in the environment.

I was using the Double DQN algorithm while training. It was performing better than the vanilla DQN in both training time and training results.

My model is trained under these parameters, except for the algorithm. Due to this, I am actually using fairly a large learning rate, a small memory and a short exploring time. I could have a better model if I could change them, but unfortunately the time is limited so I have to accept that.

```
parser.add_argument('--batch_size', type=int, default=32, help='batch size for training')
parser.add_argument('--learning_rate', type=float, default=1.5e-4, help='learning rate for training')
parser.add_argument('--gamma', type=float, default=0.99, help='discount factor')
parser.add_argument('--memory_cap', type=int, default=10000, help='memory capacity')
parser.add_argument('--n_episode', type=int, default=60000, help='num of total training episodes')
parser.add_argument('--n_step', type=int, default=5000, help='num of training steps')
parser.add_argument('--f_update', type=int, default=5000, help='frequency of network update steps')
parser.add_argument('--explore_step', type=int, default=200000, help='steps of epsilon decay')
parser.add_argument('--load_model', type=bool, default=False, help='load model to continue training')
parser.add_argument('--action_size', type=int, default=4, help='number of valid actions')
parser.add_argument('--algorithm', type=str, default='DQN', help='type of training algorithm')
# epsilon decay
self.epsilon = 1.0
self.epsilon_min = 0.025
self.epsilon_decay = (self.epsilon - self.epsilon_min) / self.explore_step
```

The activation function is ReLU, with Kaiming He initial function[3]

what kind of loss function you used and what kind of optimizer you used.
Adam optimizer (learning rate 1.5e-4), huber loss(nn. Smooth-l1-loss)

Special skills: Include the skills which can improve the generation quality. Here are some [tips](https://arxiv.org/pdf/1710.02298.pdf) may help. (Optional)

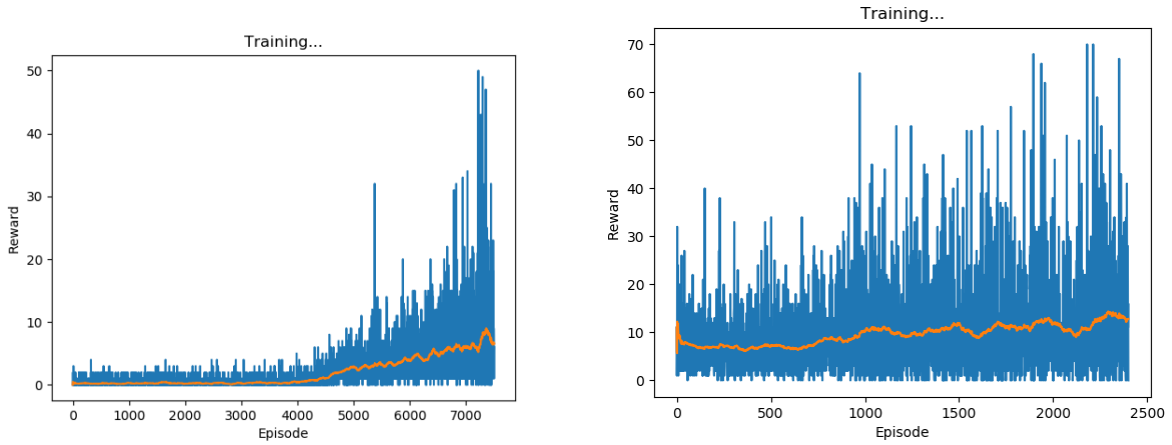
- Using double DQN
- Normalizing the input states by dividing 255
- Changed the initializer for nn to the one mentioned in paper[3].
- I saved an additional flag indicating that the agent lost a life in memory, although it won't help for this training since it has only one life, it still might help the agent to understand the importance of keeping a life.

Visualization: Learning curve of DQN.
This is the screenshot for session 2, since I have to move my computer around and pause training.

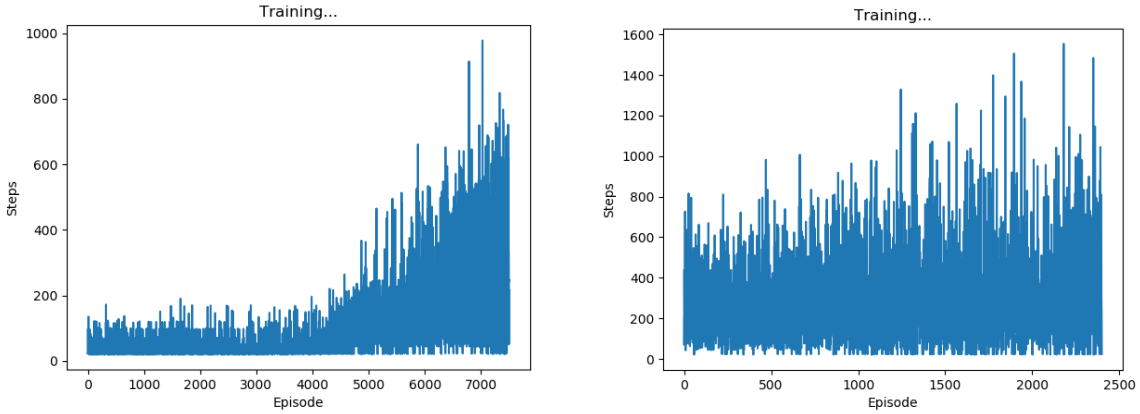
```
(base) hanson11666@instance-1:~/DQN$ python main.py --train_dqn --algorithm 'DDQN' --load_model True
/home/hanson11666/anaconda3/lib/python3.7/site-packages/gym/envs/registration.py:14: PkgResourcesDepr
  result = entry_point.load(False)
WARN: gym.spaces.Box autodetected dtype as <class 'numpy.uint8'>. Please provide explicit dtype.
WARN: gym.spaces.Box autodetected dtype as <class 'numpy.uint8'>. Please provide explicit dtype.
using algorithm  DDQN
using device  cuda
```

After 3hrs of online training (session 1 and 2 combined), with training reward 13.46,

```
================================================
Current steps =  532092
Current epsilon =  0.025
Current episode =  2000
Current mean reward =  12.37
Current memory allocated =  10000
Best mean reward =  12.88
<<<target net updated at step, 535000 >>>
<<<Model updated with best reward =  13.3 >>>
<<<Model updated with best reward =  13.44 >>>
<<<target net updated at step, 540000 >>>
<<<Model updated with best reward =  13.46 >>>
```



10k episodes with 7500 under session 1 and 2000 under session 2 (which is undergoing)



Approximately 2m total steps (sum of session 1 and 2)

And the result is Mean: 46.94, reaches the goal already.

```
WARN: gym.spaces.Box autodete
WARN: gym.spaces.Box autodete
using device  cpu
loading trained model
WARN: <class 'atari_wrapper.F
Episode  1 ,reward =  65.0
Episode  2 ,reward =  100.0
Episode  3 ,reward =  43.0
Episode  4 ,reward =  127.0
Episode  5 ,reward =  1.0
Episode  6 ,reward =  75.0
Episode  7 ,reward =  44.0
Episode  8 ,reward =  42.0
Episode  9 ,reward =  0.0
Episode  10 ,reward =  0.0
Episode  11 ,reward =  97.0
Episode  12 ,reward =  66.0
Episode  13 ,reward =  30.0
Episode  14 ,reward =  135.0
Episode  15 ,reward =  4.0
Episode  16 ,reward =  8.0
Episode  17 ,reward =  160.0
Episode  18 ,reward =  19.0
Episode  19 ,reward =  16.0
Episode  20 ,reward =  12.0
Episode  21 ,reward =  8.0
Episode  22 ,reward =  160.0
Episode  23 ,reward =  19.0
Episode  24 ,reward =  16.0
Episode  25 ,reward =  12.0
Episode  26 ,reward =  8.0
Episode  27 ,reward =  160.0
Episode  28 ,reward =  19.0
Episode  29 ,reward =  16.0
Episode  30 ,reward =  12.0
Episode  31 ,reward =  65.0
Episode  32 ,reward =  100.0
Episode  33 ,reward =  43.0
Episode  34 ,reward =  127.0
Episode  35 ,reward =  1.0
Episode  36 ,reward =  65.0
Episode  37 ,reward =  100.0
Episode  38 ,reward =  43.0
Episode  39 ,reward =  127.0
Episode  40 ,reward =  1.0
Episode  41 ,reward =  8.0
Episode  42 ,reward =  160.0
Episode  43 ,reward =  19.0
Episode  44 ,reward =  16.0
Episode  45 ,reward =  12.0
Episode  46 ,reward =  75.0
Episode  47 ,reward =  44.0
Episode  48 ,reward =  42.0
Episode  49 ,reward =  0.0
Episode  50 ,reward =  0.0
Episode  51 ,reward =  65.0
Episode  52 ,reward =  100.0
Episode  53 ,reward =  43.0
Episode  54 ,reward =  127.0
Episode  55 ,reward =  1.0
Episode  56 ,reward =  97.0
Episode  57 ,reward =  66.0
Episode  58 ,reward =  30.0
Episode  59 ,reward =  135.0
Episode  60 ,reward =  4.0
Episode  61 ,reward =  75.0
Episode  62 ,reward =  44.0
Episode  63 ,reward =  42.0
Episode  64 ,reward =  0.0
Episode  65 ,reward =  0.0
Episode  66 ,reward =  8.0
Episode  67 ,reward =  160.0
Episode  68 ,reward =  19.0
Episode  69 ,reward =  16.0
Episode  70 ,reward =  12.0
Episode  71 ,reward =  8.0
Episode  72 ,reward =  160.0
Episode  73 ,reward =  19.0
Episode  74 ,reward =  16.0
Episode  75 ,reward =  12.0
Episode  76 ,reward =  75.0
Episode  77 ,reward =  44.0
Episode  78 ,reward =  42.0
Episode  79 ,reward =  0.0
Episode  80 ,reward =  0.0
Episode  81 ,reward =  8.0
Episode  82 ,reward =  160.0
Episode  83 ,reward =  19.0
Episode  84 ,reward =  16.0
Episode  85 ,reward =  12.0
Episode  86 ,reward =  75.0
Episode  87 ,reward =  44.0
Episode  88 ,reward =  42.0
Episode  89 ,reward =  0.0
Episode  90 ,reward =  0.0
Episode  91 ,reward =  8.0
Episode  92 ,reward =  160.0
Episode  93 ,reward =  19.0
Episode  94 ,reward =  16.0
Episode  95 ,reward =  12.0
Episode  96 ,reward =  75.0
Episode  97 ,reward =  44.0
Episode  98 ,reward =  42.0
Episode  99 ,reward =  0.0
Episode  100 ,reward =  0.0
Run 100 episodes
Mean: 46.94
```

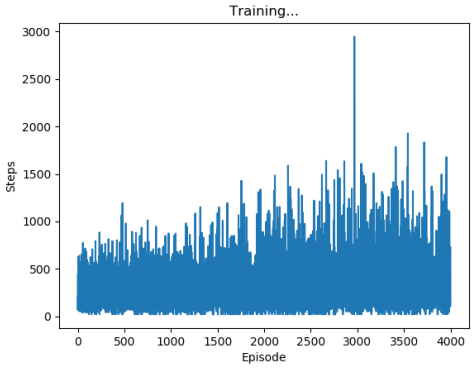
After that I am constantly testing my best model:

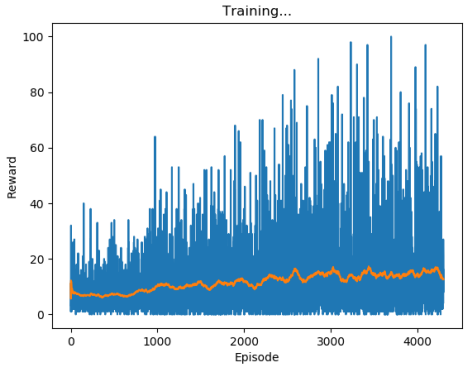
3.5hrs, 10k episodes, 15.15 max reward, 2.2m steps with mean 60.1

```
Current steps =  695729
Current epsilon =  0.025
Current episode =  2500
Current mean reward =  11.8
Current memory allocated =  10000
Best mean reward =  15.15
<<<target net updated at step, 700000 >>>
Episode  96 ,reward =  12.0
Episode  97 ,reward =  2.0
Episode  98 ,reward =  165.0
Episode  99 ,reward =  16.0
Episode  100 ,reward =  4.0
Run 100 episodes
Mean: 60.1
```

5hrs, 11.5k episodes, 17.97 max reward, 2.8m steps with mean 73.3

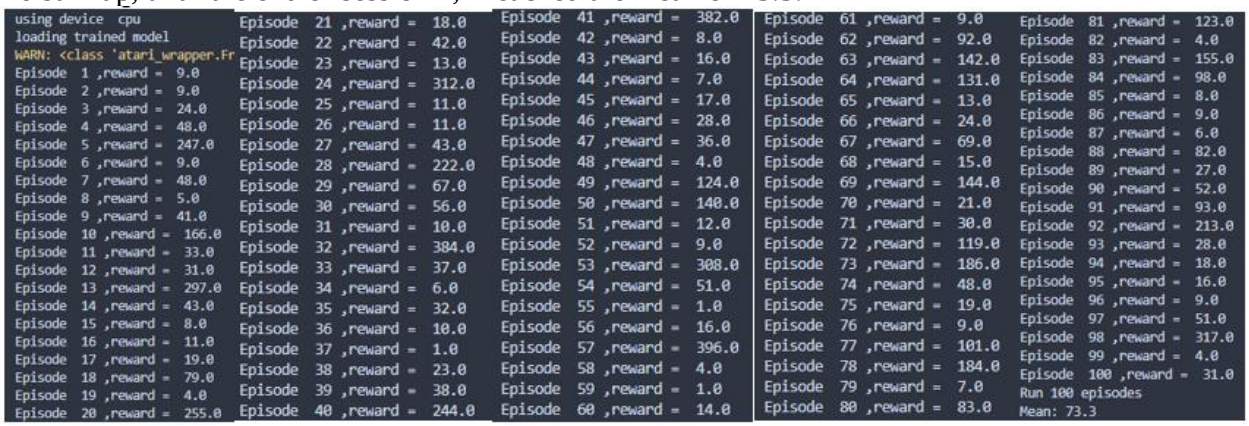
```
Current steps =  1299509
Current epsilon =  0.025
Current episode =  4100
Current mean reward =  15.59
Current memory allocated =  10000
Best mean reward =  17.38
<<<target net updated at step, 1300000 >>>
<<<target net updated at step, 1305000 >>>
<<<target net updated at step, 1310000 >>>
<<<target net updated at step, 1315000 >>>
<<<target net updated at step, 1320000 >>>
<<<target net updated at step, 1325000 >>>
<<<Model updated with best reward =  17.45 >>>
<<<target net updated at step, 1330000 >>>
<<<Model updated with best reward =  17.46 >>>
<<<Model updated with best reward =  17.51 >>>
<<<Model updated with best reward =  17.97 >>>
Episode  90 ,reward =  52.0
Episode  91 ,reward =  93.0
Episode  92 ,reward =  213.0
Episode  93 ,reward =  28.0
Episode  94 ,reward =  18.0
Episode  95 ,reward =  16.0
Episode  96 ,reward =  9.0
Episode  97 ,reward =  51.0
Episode  98 ,reward =  317.0
Episode  99 ,reward =  4.0
Episode  100 ,reward =  31.0
Run 100 episodes
Mean: 73.3
```





Steps and rewards for session 2

To sum up, until the end of session 2, I reached the mean of 73.3.



Reference:
[1] V. Mnih, Playing Atari with Deep Reinforcement Learning
[2] Z. Wang, Dueling Network Architectures for Deep Reinforcement Learning
[3] K. He, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification
[4] https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756
[5] https://github.com/dennybritz/reinforcement-learning/issues/30