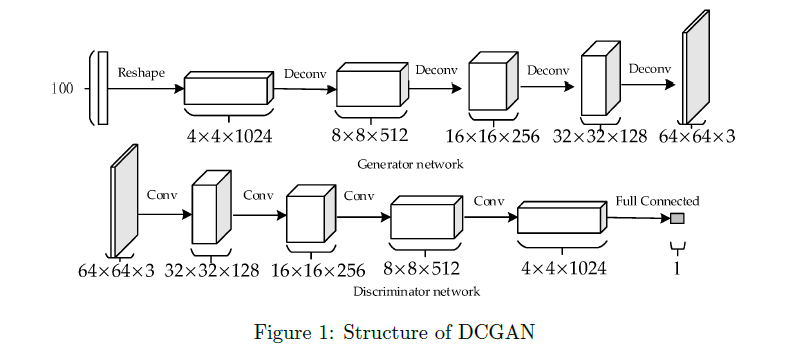
**Deep Learning Homework 3**

**0853412 吳宛儒**

**1 Generative adversarial network (GAN)**



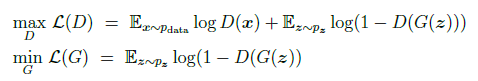
**1. Data augmentation can be used to enhance GAN training. Describe how you preprocess the dataset (such as resize, crop, rotate and flip) and explain why.**

在圖片前處理的部分，主要參考網路上的文章：[PyTorch 學習筆記（三）：transforms的二十二個方法](https://zhuanlan.zhihu.com/p/53367135)。圖片的前處理大致上可以分成裁剪、翻轉、旋轉和圖像變化等等。針對模型任務，前處理著重在裁剪上，因為發現資料集中的圖片雖然多樣，但其人臉幾乎集中在畫面中間，因此在裁剪上選擇使用CenterCrop的方式。然而直接套用CenterCrop會發現人臉被切半，多集中在鼻子以上的部分，因此前面需要先對圖片做Resize。Resize的部分則嘗試了img\_size(設定img\_size=64)、img\_size+15以及img\_size+45三種作法。當針對img\_size去作CenterCrop時，會發現其包含在畫面內的背景非常多樣，然而背景並不是我們的模型需要費心學習的部分，因此需進一步裁切。

|  |  |
| --- | --- |
| **不做Resize直接CenterCrop** | **使用img\_size做Resize** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\272B9122.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\579A168A.tmp |
| 人臉被切到只剩下上半部 | 背景多樣 |
| **使用img\_size + 15做Resize** | **使用img\_size + 45做Resize** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\891BDF28.tmp | C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\77A94AF4.tmp |
| 比較專注於人臉，但仍有部分背景成分 | 讓整個畫面幾乎只剩人臉 |

若先做resize到79\*79(64+15=79)可以發現CenterCrop後背景占比的部分減少許多，比較專注於人臉。而若是先做resize到109\*109(64+45=109)再CenterCrop後幾乎只剩下人臉，也是模型最主要學習目標，最後決定resize成img\_size+45再進行CenterCrop。除了對圖片的裁剪以外，也在transform的時候對圖片做normalize處理，讓其值在三個維度上皆是mean為0.5，std為0.5。

**2. Construct a DCGAN with vanilla GAN objective, plot the learning curves for both generator and discriminator, and draw some samples generated from your model.**



|  |  |
| --- | --- |
| **[實驗一]：**當generator和discriminator之初始weight皆設為0(最初始的code) | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\114FAF8E.tmp | 一直到最後的iteration仍非常震盪，generator和discriminator有generator稍微較弱(loss較大)的感覺。兩者相互拮抗並未達到平衡。圖片生成上，有些人臉輪廓跑掉或是沒有輪廓。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\39DB52CC.tmp | |

|  |  |
| --- | --- |
| **[實驗二]：**當generator和discriminator之初始weight為隨機數字 | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\91281D19.tmp | 收斂速度較實驗一來得快，收斂效果也比較好。不會像實驗一如此震盪。圖片生成上，相比於實驗一，在輪廓上生成大部分有改善，然而仍有色塊不平均、五官些微扭曲的情況。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\51746D80.tmp | |

|  |  |
| --- | --- |
| **[實驗三]：**real和fake的label改為soft label version 1 | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\74445EBA.tmp | 把原本real=1, fake=0的label轉換為soft label，這邊使用real label = 0.9~1.5之間，fake label=0.0~0.5之間。發現雖然生成圖片效果普通，但相比實驗二，其loss到最後有逐漸增大的趨勢，G和D相互拮抗到最後D比較強。想嘗試把soft label的範圍調小一些。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\C1F6C6D8.tmp | |

|  |  |
| --- | --- |
| **[實驗四]：**real和fake的label改為soft label version 2 | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A2B84AA6.tmp | 震盪的情形看起來比實驗三還要小一些，但是後期也有稍微增強的趨勢。在生成圖片的部分，則出現了比較多偏白的且輪廓不清楚的人臉，估計也和label設定的值域相關。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E4E655A4.tmp | |

|  |  |
| --- | --- |
| **[實驗五]：**更改transform讓圖片在-60~60度間隨機旋轉 | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\82DC4B30.tmp | 這部分本來預期在不複雜化模型的前提下，應該是比較難train到比較好的結果。上圖為training data的模樣。可以看到一開始loss就能比較低，但到後期G的loss的震盪仍偏大，推測應該就是和先前的transform有關，G比較難產圖，下面的圖也相應證：效果並沒有前面幾次來得好。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\510C48BE.tmp | |

|  |  |
| --- | --- |
| **[實驗六]：**更改模型—更改generator的activation function | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4560C8E9.tmp | 這部分將G的activation function由ReLU改成同D的LeakyReLU，然而其震盪仍大，生成的圖片雖然輪廓鰻清晰，但也有色塊不均勻的情況，相比於原本未更動前的版本，這個版本的收斂程度比較差。因此推測還是保持原本的activation function才會有比較好的模型效果。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7729C3DF.tmp | |

|  |  |
| --- | --- |
| **[實驗七]：**更改模型learning rate 讓D的lr比較大 | |
| **Loss record of Generator and Discriminator** | **觀察** |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\189F8F61.tmp | 有參考網路上提供的建議，雖然G&D兩者應該相互平衡，但一般會希望D略強一些，因此調動D的lr到0.0004。相比於實驗二的結果，雖然模型看起來有在逐漸收斂，但是效果不如實驗二，且生成的圖片有霧感，輪廓也不如實驗二清晰。 |
| **Real images and fake images** | |
| C:\Users\wwj\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\AD2DE917.tmp | |

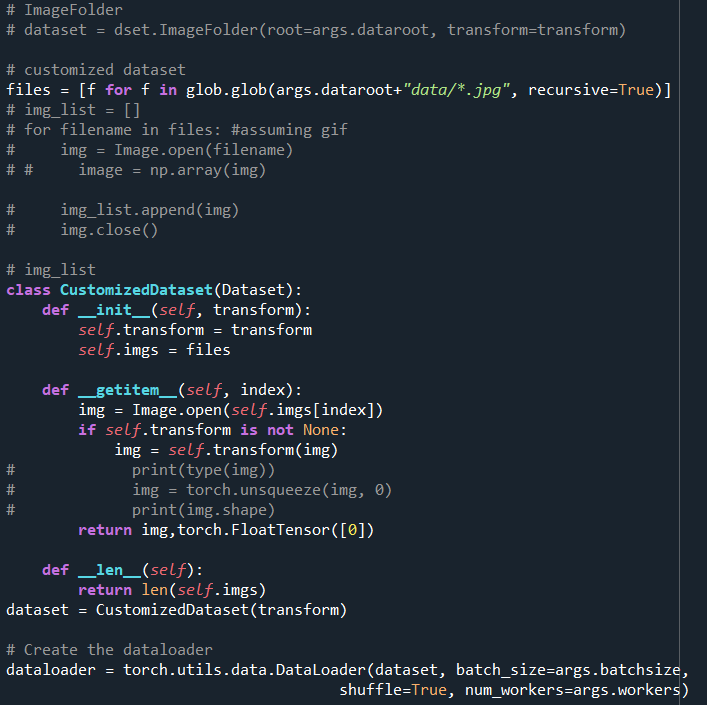
**3. Implementation details are addressed as follows**

**(a) Models has already been designed in Model.py. Feel free to modify the generator and discriminator, and you can write down how you design your model and why. (bonus 5 points)**

**更改模型中的activation function**如上述的實驗六。主要是因為之前經驗是LeakyReLU效果普遍比ReLU還要好，但是助教提供原始的model裡面generator是使用ReLU作為activation function，因此想改成和discriminator一樣是LeakyReLU並比較效果。原本也有想要更改CNN的層數、filter數或是stride大小等等，但查了許多網路上的資料，DCGAN最好效果幾乎都是用這一組參數與架構，因此這部分沒有多做更動。實驗七則是**更改learning rate**觀察變化。

**(b) In data preprocessing, the ImageFolder and Dataloader provided by pytorch are recommended. The customized dataset (without using ImageFolder) can be implemented for extra points. (bonus 5 points)**

自己寫dataset：原本是使用ImageFolder來直接作為dataset，後來使用自定義的dataset來讀取圖片資料集。首先先將資料夾底下所有圖片的路徑皆存在files這個list底下，接著使用getitem的方式來取得圖片。由於是第一次寫還不太熟悉，中間過程遇到蠻多困難，例如究竟要在init就開好圖片還是要在getitem才開好圖片？如果一開始在init就開好(而非只是給路徑)，就會遇到無法對closed image操作的問題；另外也有遇到像是getitem要return的type以及資料要是什麼，type的部分在Image、np array以及tensor上兜轉了許久才確定是回傳tensor，而回傳資料的部分則是一開始只有寫return img的時候出來的shape會錯誤，後來才發現是需要return兩個資料，一個是data一個是label，納因為我們模型不需要label，雖然不會用到但還是需要有個東西去接它，才不會使得dataloader出來的東西出錯，如果只有return img的話，dataloader出來只會有一張圖片，而不是batchsize的數量的圖片。

****

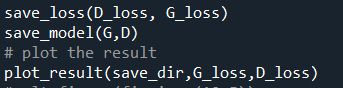
**(c) In main.py, you have to complete three functions. main(), train(), and visualizaion.**

* **main() : you have to set up dataset (dataloader), models, optimizers, and the criterion. After preparation, the train function is called to start the training procedure.**
* **train() : In every iteration, you have to perform the following jobs**
  + **send true data into the discriminator, and update the discriminator.**
  + **use generator to create fake data, and send them to the discriminator. Calculate loss for both models and update them.**
  + **record the loss in every iteration and draw some samples by the current generator after the fixed number of iterations.**
  + **after finishing all epochs, you have to save the files of your models, losses, and samples. (sampling should not be more than 20 times)**

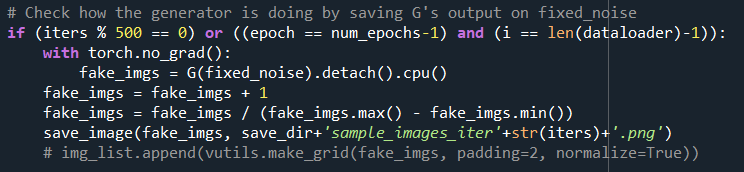
**(d) Visualization(If you don't use jupyter notebook, attach your visualization code in the folder and chart in report.)**

* **write down the visualization code in Visualization.ipynb, you can open it by using jupyter notebook.**

這部分我在main和其引入的Visualization中的程式碼包含了所有visualization的code，並沒有使用到jupyter notebook。



這是在main當中，跑完所有的iterations以後將losses, models和loss的圖都存下來。



而上面這部份則是每500個iterations會把生成的假的圖片存起來。

* **plot the following two charts and show them in the report**
  + **original samples and generated samples**
  + **learning curves for generator and discriminator**
  + **training procedure of GANs is unstable, when visualizing the loss curve you can do moving average every N steps (smooth the curve) to observe the trend easily.**
  + **we provided an extra visualizaion code in Visualization.ipynb, you can use it to see what does the generator generates through the training procedure.**

**(e) Please do some discussion about your implementation. You can write down the difficulties you face in this homework, e.g. hyperparameter settings, analysis of the**

**generated images, or anything you want to address.**

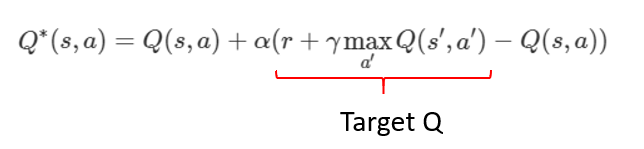
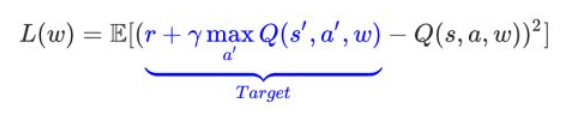
**DCGAN總結：**

在這次GAN的實作中，發現GAN的training蠻困難的，先前僅有使用keras實作過簡單的GAN，而這次第一次使用Pytorch，也第一次訓練DCGAN。有可能因為原本的Model架構與參數都已經是多人經驗中最好的tuning，因此在前面七次實驗中，使用同樣的model架構下，似乎都沒辦法有效地提升模型訓練結果，往往都只會讓效果更差。也發現在改善圖片生成上，無法確定究竟更改哪個部分會對應到圖片生成的哪個部分？例如色塊不均、輪廓不明顯或是有霧感，都沒辦法確切找出究竟是模型中哪項因子影響到，或是說其實這些都是整體綜合下的結果，因此在訓練這種generative model的時候，評估生成圖片的好壞其實是相對困難的，之前也有看過有人試著將圖片生成之好壞量化，但詳細作法還要再去瞭解。而模型效能提升的部分應該還有待更進一步去survey近年有針對GAN上作improvement的paper，方能找到更有效的方法去改善實驗結果。

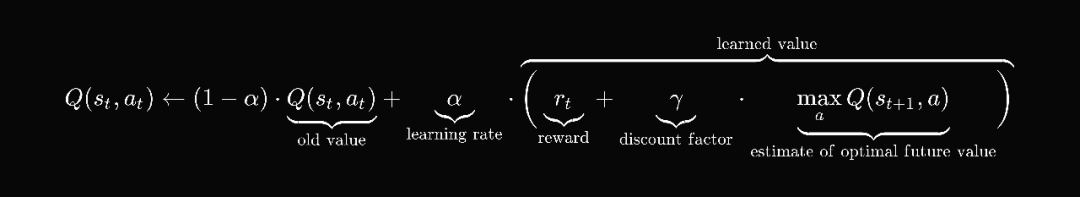
**2 Deep Q Network (DQN)**

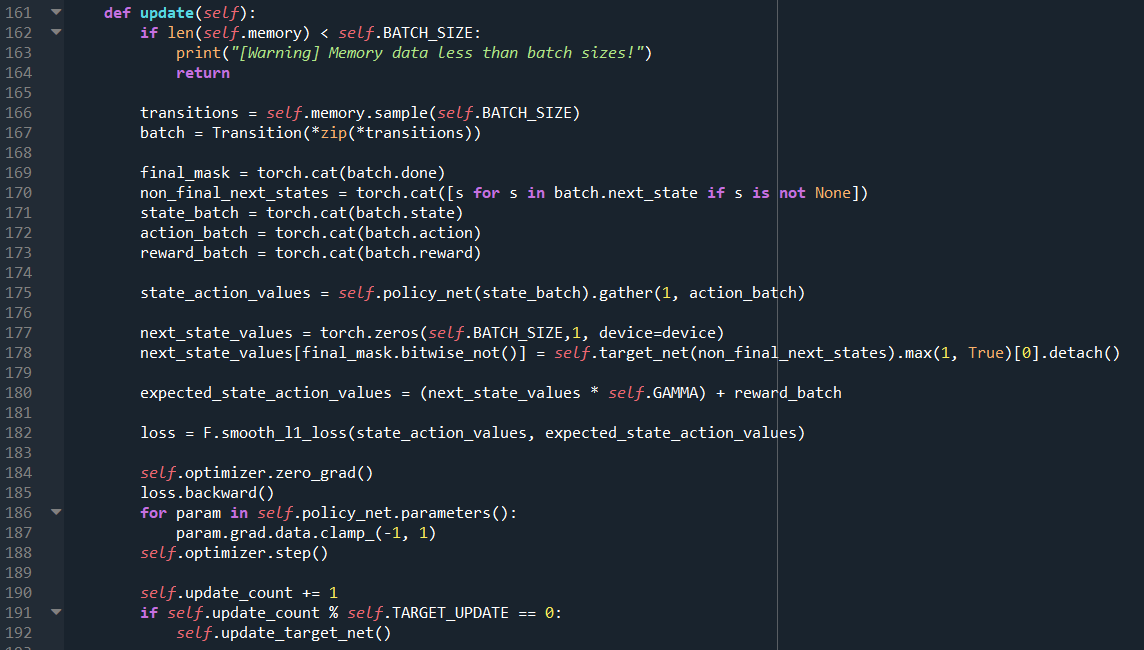
**1. Please indicate the code paragraph about the updating based on the temporal difference learning in your implementation or from the given source code**



目標是尋找最佳的Q，也就是Q\*。可以把這樣的target Q視為一個label，在不斷更新的過程中，讓Q function可以逼近target Q function。在定義好這樣的loss以後就能進行NN訓練。





**Explain the purpose of the following hyperparameters: updating step α, discount factor γ, target network update period τ , and ϵ for ϵ -greedy policy.**

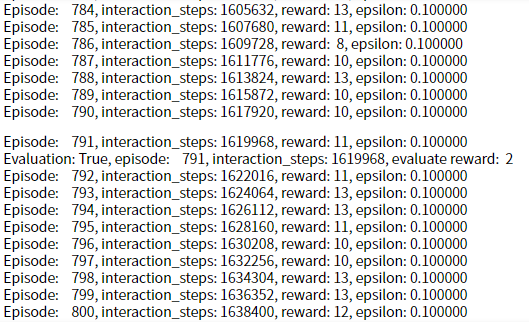
**Updating step(α):** 為0~1之間的值。α=0代表新的Q(s,a)會直接使用舊的Q(s,a)，表示沒有學習能力、Q完全不改變，而若α=1則是代表新的Q(s,a)會採取「當下獎勵值R(s,a)和後續新狀態之最大可能獎勵總和maxQ」來共同決定，而不考慮舊的Q(s,a)。

**Discount factor(γ):** 為0~1之間的值。通常是小於1的值。當下的反饋是相對重要的，當時間間隔越久，影響就該越小。

**Target network update period(τ):** DQN算法中的target跟policy network是分開的，policy network做一段時間以後複製到target network，Target network update period即是決定多久更新一次target network。

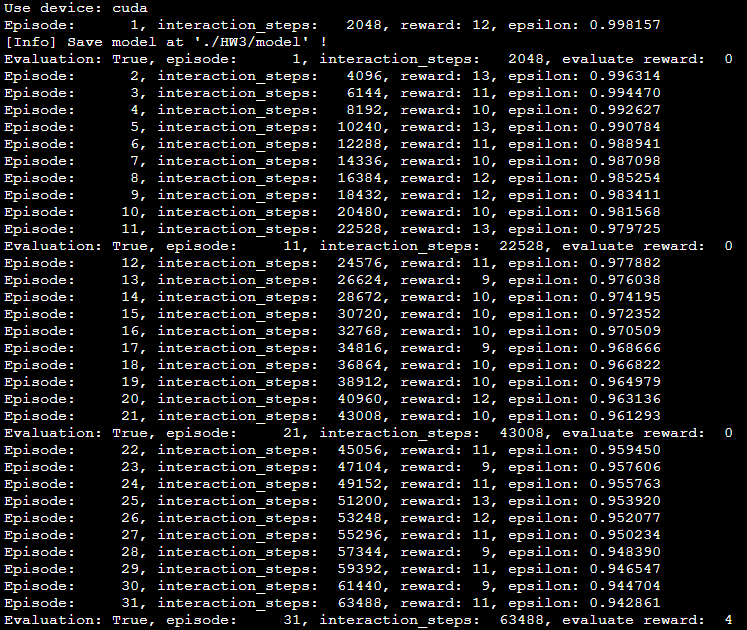
**ϵ-greedy policy(ϵ):** ϵ介於0~1之間，代表在決定下一個action的時候，有ϵ的機會會採用隨機的方式挑選動作，有(1- ϵ)的機率會挑選具有最大reward的動作。所以ϵ代表了隨機性的策略機率。

**2. For deep Q-Learning, exploring the environment is an important procedure. Because the reward signals in the given environment is very sparse, it takes a lot of computation time to explore the state space and perform the updating. To speed up the training process, you can simply change the probability of random agent: [ NOOP (0.3), UP (0.6), DOWN(0.1) ]. Please show the total reward of sample episodes for this configuration.**



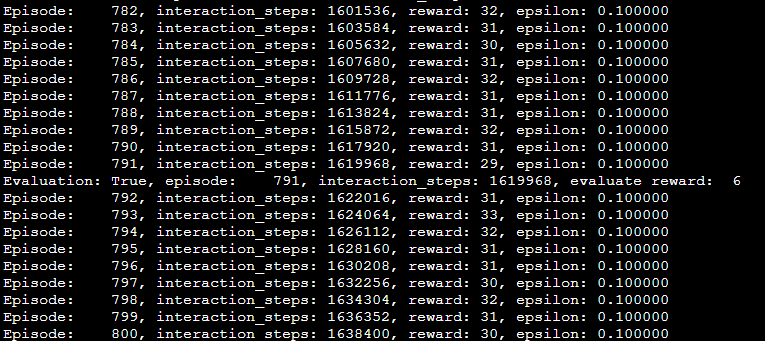
**原始的code：**三種action操作的機率均等。訓練到最後，episode結果如下：

在這樣的設定狀況下，前面有很長一段時間reward皆是0。一直到800 episode附近的時候才有約莫10的reward。效果不甚好。



**更改過後：**將三種action操作的機率改為**[ NOOP (0.3), UP (0.6), DOWN(0.1) ]**。結果如下：

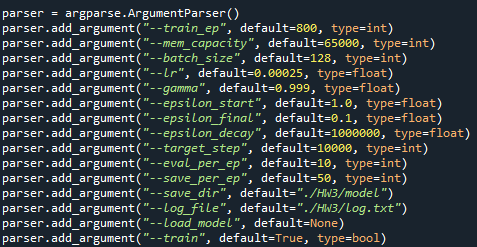
可以發現在最一開始就可以將reward來到10上下，而到最後快要結束的時候，則可以來到平均31左右的reward。明顯效果有提升，agent玩遊戲玩得比較好。



**3. Use the modified random agent in the ϵ -greedy and keep training. Here, you should tune hyperparameters to let model converge. Plot the episode reward in learning time and evaluation time (ϵ = final epsilon) (2 charts). Show your configuration and discuss what you find in training phase.**

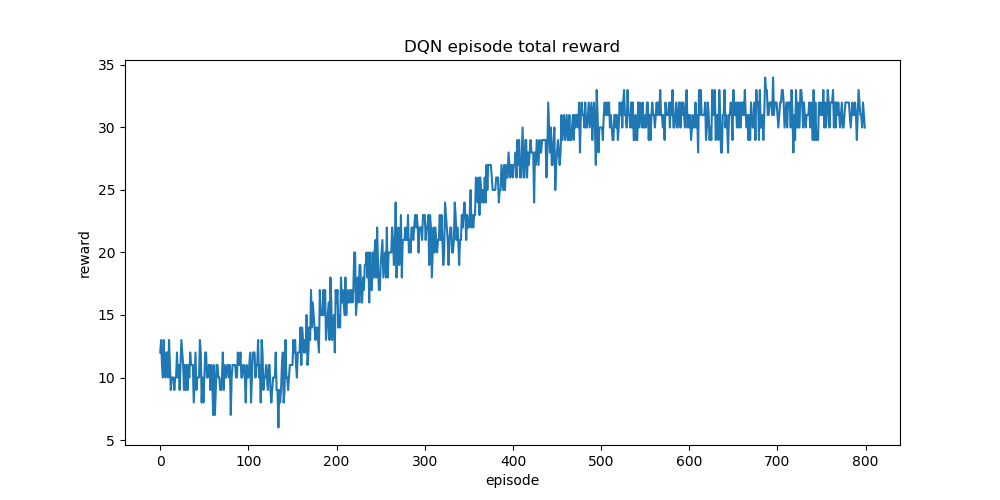
使模型收斂的方法：參考論文使用的超參數設定，並適度更改mem\_capacity。

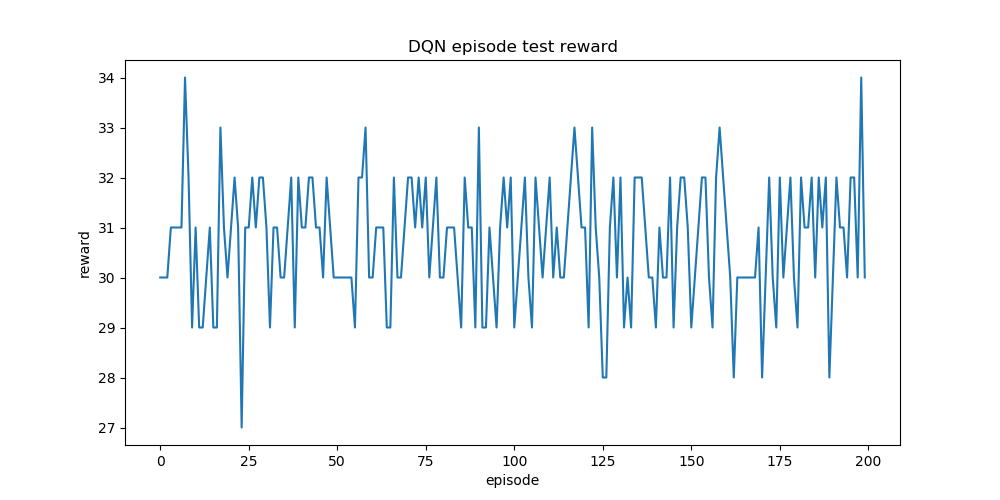
1. **參考論文中的超參數設定：**



1. **調整mem\_capacity：**

因為使用實驗室的server上的GPU，發現原本default設定的mem\_capacity=90000容易在幾次episode之後產生Runtime error，會發生不足使用的問題，因此最後調整為mem\_capacity=65000讓實驗能夠順利跑完，但也要花上4~5小時之久。





4. After training, you will obtain the model parameters for the agent. Show total reward in some episodes for deep Q-network agent.

在testing phase中，將原本訓練好的agent之model parameters load進來，呈現episode結果，並和paper中的效果作對照：

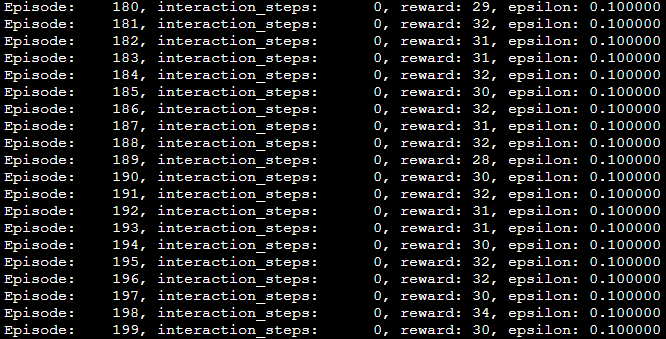
Paper中的結果：



模型結果：

Mean: 30.705

Std: 1.2074663556389469



在testing的時候模型平均獲得的reward為30.705左右，略好於paper上的紀錄。在後面遊戲畫面的截圖上也可以看到agent有許多策略是偏向高難度、高風險，是人在遊玩的時候比較不會做出的、卻可以大幅提升reward的操作與決定。

**5. Sample some states, show the Q values for each action, analyze the results, and answer**

* **Is DQN decision in the game the same as yours? Any good or bad move?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **遊戲畫面** | **Q value** | **choice** | **avg** | **我的 不同於我的決定** |
|  | [1.4895419 1.5018126 1.3285295] | UP | 1.4399613 | 車子距離我的前方還有些微距離，選擇前進。 |
|  | [0.7968872 0.86689264 0.746572] | UP | 0.8034506 | 前面有車子靠近(褐色)但agent決定要往前。 |
|  | [0.86951584 0.847171 0.83254826] | NOOP | 0.84974504 | 前方車子剛離去但agent沒打算前進反而是經歷了兩次的NOOP決定。 |
|  | [1.5487285 1.477078 1.5487279] | NOOP | 1.5248448 | 當逼近車子的時候，選擇NOOP。在此前一個動作為DOWN來跟車子保持一定的距離。 |
|  | [1.6127053 1.6073669 1.6194779] | DOWN | 1.6131834 | 最後一段路程，agent上上下下了很多次，但是這時車道其實是相對空曠安全的，卻花費了很多時間與步伐在最後兩道上。 |
|  | [0.6470836 0.98710275 0.65066534] | UP | 0.7616172 | 在連續三道都即將有車子行進過來的狀態下，agent直接火力全開向前行，毫不猶豫。 |
|  | [1.4956856 1.5925587 1.4161783] | UP | 1.5014743 | 類似上一個，這次的agent幾乎一路衝到最後，即將驚險閃過亮綠色的車子。 |
|  | [1.206667 1.4406475 0.86545074] | UP | 1.1709218 | 發現agent可以許多次都能夠驚險地度過難關，例如這張截圖，通常人在玩的時候可能沒辦法很精準知道怎樣是「允許的邊界」，因此不會做這麼極端的決定。 |
|  | [1.1888151 1.3931128 0.9849134] | UP | 1.1889471 | 發現agent在兩道都有車子剛好經過的時候，很喜歡從中間剛好穿過去。我想這是人比較不會做的決定，也會是相對較困難、具有風險的決定。 |
|  | [0.20109983 0.2006028 0.2055704] | DOWN | 0.20242435 | 前方兩道車子都剛好像又離去，但是agent卻連做了四次DOWN的決定。 |

* **Why the averaged Q-value of three actions in some state is larger or less than those of the other states?**

從上面的遊戲截圖可以看出來，三種action的Q value平均較高者(avg>1)，agent在畫面上都已經走完了一半，也就是超過中間的雙黃線，而平均較低者(avg<1)，則是大多還沒走到一半(如上面表格中最後一個遊戲截圖)，也就是距離成功獲得reward還有一大段距離，因此由此推測，當越接近獲得reward的時候，其平均的Q value也會越高，當遊戲在離獲得reward還有段距離時，平均Q value會較低。

**DQN總結：**

在這次DQN的實作當中，透過助教給予的輔助程式碼，能讓我更快理解實作細節。原本在理論上比較不清楚的部分也可以透過code來去驗證。之前總是耳聞RL很難訓練，這次可能是因為有了現有的固定程式以及架構，才會讓reward在改變機率以後就有很好的提升。日後希望有機會能再survey有關DQN的實作細節以及近年來更多的改善方法，並試著用在更多的應用層面上。

**GAN以及DQN model參數連結：**

<https://drive.google.com/drive/folders/1WuuSSY5m-BpI2b41BTaWpIzouYNT1rmB?usp=sharing>