SUMTREE

Used to store and retrieve the experiences. This Class of Object is faster and more efficient than a list or dictionary

SumTree is Binary Tree where each parent node is the sum of its child (left, Right).

We will use the SumTree to store from each experience the data = [S, A, R, Next S, done] and its priority value "p"

Example:

Capacity = 2**3 = 8 (number of experiences we can store -- always power of 2)

priority values = [8. 3. 7. 4. 11. 1. 5. 9.] -- added to the leaf nodes

After we add all value above, we will have the SumTree below

[48] - calculated (Ex: 48 = 22 + 26)

[22, 26] - calculated (Ex: 26 = 12 + 14)

[11, 11, 12, 14.] - calculated (Ex: 11 = 8 + 3)

[8, 3, 7, 4, 11, 1, 5, 9] - added

tree index -- it has 15 positions from 0 to 14

[0]

[1, 2]

[3, 4, 5, 6]

[7, 8, 9, 10, 11, 12, 13, 14]

tree = np. array

[48. 22. 26. 11. 11. 12. 14. 8. 3. 7. 4. 11. 1. 5. 9.]

PRIORITIZE EXPERIENCE REPLAY (PER)

experiences that has reward equal to zero, it means that the agent didn't collect any banana.

The goal is to select the best episodes on the memory to train the DQN.

Episodes with higher Error, are more important, they can make the DQN agent learn much more. But maybe they not happen as frequent as other episodes. Said that we can't just random sample the episodes in a uniform way.

We will create an Algorithm to choose these episodes more often. But we don't want to discard the others, because in earlier stages even if we have episodes with zero error, it doesn't mean that this error is correct. The model just started to learn.

If we don't have enough experiences with rewards 1 (agent collects yellow bananas) or -1 (agent collects blue bananas), it will be very difficult for the DQN to learn.

Using PER, we will be able to select much more often experiences that has a higher error, and in the beginning of the training all experience with rewards -1 or 1, will have the higher error than experiences with zero reward.

Error = abs (Expected Q Value- Target Q value)

$$p_t = |\text{Error}| + \varepsilon$$

$$P(i) = \frac{{p_i}^a}{\sum_k {p_k}^a}$$

$$IS(i) = \left(\frac{1}{n} * \frac{1}{P(i)}\right)^b$$

$$IS_{max} = \left(\frac{1}{n} * \frac{1}{P_{min}}\right)^b$$

$$IS_{noralized}(i) = \frac{IS\ (i)}{IS_{max}}$$

IS = Importance Weights Sampling

n = $number\ of\ samples$ – mini batch size

$P(i)$ = priority of samples(i)

P_{min} = minimum priority on the memory

b = Controls how much IS will affect learning. Start with 0, and during the training move slowly to 1

Modify Loss Function - MSE

$$\text{Loss} = IS_{noralized} * (\text{Expected} - \text{Target})^2$$

DUELING DQN

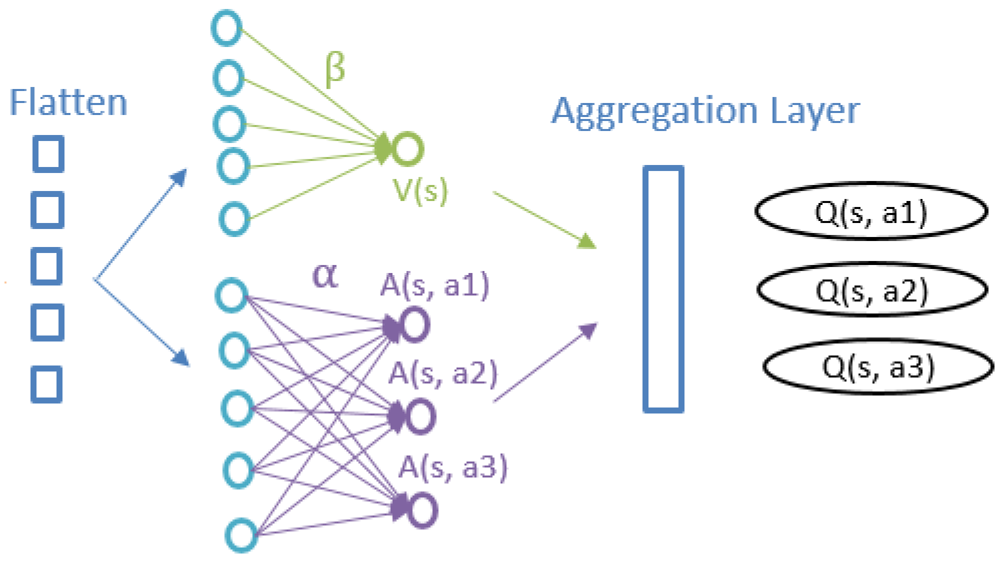
The goal is to let the DQN learn the V(s) (Value of a State) and the A (s, a) Value of Take an Action separately.

Each State has an intrinsic value, independently of the action the agent will take.

For example, if it is a State that any action will take to a terminal state, the A (s, a) won’t mater, the Q (s, a) will be the same for all actions

DDQN can learn which states are (or are not) valuable without having to learn the effect of each action at each state (since it’s also calculating V(s)).

Q = V + A - Average A



$$\underbrace{Q(s, a, \overset{\substack{\text{Common}\\ \text{parameters}}}{\theta}, \overset{\substack{\text{value}\\ \text{parameters}}}{\beta}, \overset{\substack{\text{advantage}\\ \text{parameters}}}{\alpha}\ )}_{\substack{Q(s,\ a)\\ \text{Output}}} = \underbrace{\underbrace{V(s, \theta, \beta)}_{\text{State Value}} + \overbrace{[\ \underbrace{A(s, a, \theta, \alpha)}_{\text{Action Advantage}} - \underbrace{\frac{1}{A}\sum_{a'} A(s, a', \theta, \alpha)}_{\text{Average Advantage}}]}^{\text{How much one action is better than the average}}}_{\text{Aggregation Layer}}$$

DOUBLE DQN

FIXED TARGET

We will create 2 DQN.

State_DQN -- Used to calculate the Q (state). Expected Q value for each action. This NN will be trained
Target_DQN -- Used to calculate the Q (Next State). This value will be used to train de DQN

Bellman Equation: Q(state) = reward + gamma * Q(Next_State)

The goal is to use the State_DQN to forecast the best action of the Next State and use Target_DQN to calculate the Q value of the Next State using the best action selected on the State_DQN.

It will help the algorithm not to overestimate the Max Q value of the Next State. In the beginning, all Q values are not accrue. Having 2 DQN, both should be correct in order to get the max Q value, otherwise the State DQN can choose one action, but this action has not the highest value on the Target DQN. It will create a good balance.

For example

Experience 1: State_DQN ==> highest value is action = 1 Target_ DQN ==> Highest Q value is related to action = 1. In this case both NN agree.

Experience 2: State DQN (Next State) ==> highest value is action = 0 Target_DQN (Next State) ==> Highest Q value is related to action = 3. In this case the NN disagree. So instead of choosing the Q value associated to action 3, we will use the Q value on the Target_DQN associated to action 0. It doesn't need to be the highest

This balance will help the agent to choose the most accrue Q (next State). We need to rely in both NN and not in only one.

HYPERPARAMETERS

Hyperparameters are extremely important.

You can have a perfect Model, but if you don't tune them well, your model won't learn.

Environment

env = env	instance of an environment
state_size = 37	state size input of the DQN (number of features)
action_size = 4	number of action -- output DQN

Exploration Rate

epsilon = 1	exploration rate
epsilon_decay = 0.9985	factor to decrease the exploration rate at each episode
epsilon_min = 0.01	min exploration rate

Train DQN

gamma = 0.97	discount rate of the future rewards
tau = 0.01	learning rate to update the parameters of the DQN_Target
batch_size = 64	number of examples the DQN will train at the same time
update_every = 4	number of time steps we need to take, before update DQN / DQN_Target
learning_rate = 0.01	learning rate to train the DQN
counter = 0	count number of steps in order to update the DQN and DQN_Target
terminal_state = None	Q Value of the Terminal State

Prioritize Experience Replay - PER

capacity = 2**14	maximum number of experiences (leaf nodes)
epsilon_per = 0.0001	add to the priority, avoid priority equal zero
a_per = 1	[0,1] 0 uniform distribution
b_per = 0.5	[0,1] calculate Importance Sampling Weights IS_Weights
increment_b = 0.0000025	how fast move b_per to 1.
absolute_error = 1	maximum Priority value

Dueling DQN

3 hidden layers of 64 neurons

Future Steps to Improve the Model

We can apply the Rainbow and include more 3 algorithms to our code

Combination of all algorithm below:

- Double DQN (done)
- Prioritize Experience Replay (done)
- Dueling DQN (done)

- AC3
- Distributional DQN
- Noise DQN

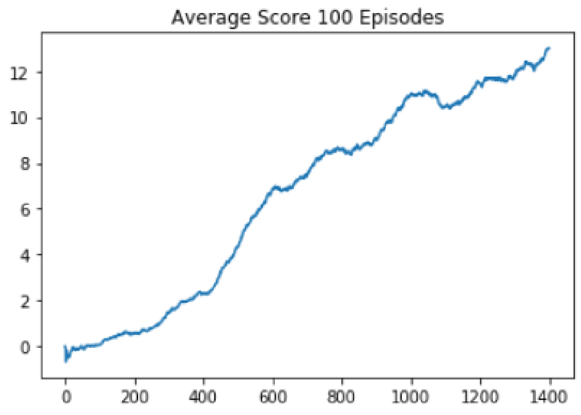
Average Score of 100 episodes

```
Using PER, DUELING DQN, DOUBLE DQN the agnet solved the problem in 1400 episodes
The agent achives a score of 13+ on average in 100 consecutives episodes
```

In [7]:
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import pickle
%matplotlib inline
```

In [8]:
```
file_in = open('all_scores3.pickle', 'rb')
a = pickle.load(file_in)
```

In [10]:
```
plt.plot(a)
plt.title('Average Score 100 Episodes')
plt.show()
```


Average Score 100 Episodes