Improving Ms. Pacman Deep Reinforcement Learning

Luke Griswold
lukedgriswold@knights.ucf.edu
University of Central Florida
Orlando, FL, USA

ABSTRACT

Despite advances in the last decade in Deep Reinforcement Learning, Pacman and Ms. Pacman remain a difficult to master environment. This paper attempts to modify the baseline gym environment's rewards to induce agent behavior in standard Deep Q-learning and Double Deep Q-learning networks. A novel Deep Split Q-learning architecture is also implemented in an attempt to influence agent learning during the training process by changing reward and punishment processing. Code is publicly available at https://github.com/Kickflip89/DQN_project/

CCS CONCEPTS

• **Computing methodologies** → **Intelligent agents**; **Reinforcement learning**; *Neural networks.*

KEYWORDS

reinforcement learning, Q-learning, Agents, Neural Networks

1 INTRODUCTION

Pacman has long since been a target of Artifical Intelligence (AI) research from pathfinding through A* to traditional Q-learning methods [4]. Despite advances in the last decade in deep reinforcement learning and convolutional architectures [9] capable of playing Atari games from screen images, Pacman and Ms. Pacman remain a curiously difficult game for agent design. This paper uses OpenAI's gym environment to attempt to improve on a baseline 2015 paper in deep reinforcement learning [11] for Ms. Pacman.

Ms. Pacman was released in 1982 as a follow-up to the wildly successful Pacman game and follows the same rules [1]. Pacman and Ms. Pacman are wonderful targets for reinforcement learning agents because although the rules are quit simple, the strategies for maintaining a high score can be quite complex [3]. The ghost behavior is emergent as well because their behavior is somewhat stochastic [1]. In this way, an agent must recognize not only the state of the map, but the current behavior of the ghosts and proximity to power pellets in order to achieve a successful strategy.

This paper aims to use some tuning and reward modifications from the baseline gym environment to attempt to raise performance for both Deep Q-learning [9] and Double Q-learning networks [11]. Finally, through splitting rewards into different streams via a relatively new Split Q-learning approach [8], the author hopes to tune the agent's attention to more relevant features for reward seeking or death avoidance.

The Double Deep Q-learning Network with a modified reward system had the best results over a long term experiment, although the adaptation of deep Split Q-learning showed promise with more hyper-parameter tuning.

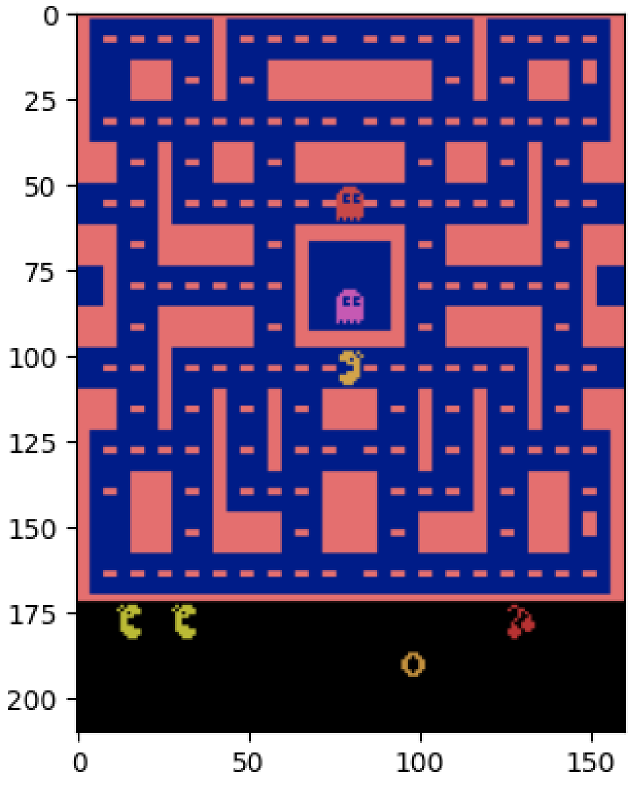


Figure 1: Starting state for the Atari version of Ms. Pacman (img size scale in pixels depicted)

2 RELATED WORK

2.1 Q-learning evolution

Q-learning is a Reinforcement Learning (RL) approach to solving Markov Decision Processes (MDPs). An MDP is defined by an environment that has a state and where taking an action in a state results in a new state and a reward, defined by a transition function [5]. There is also a discount factor for future rewards, typically denoted γ. Thus, the MDP is defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ [8]. $\mathcal{T}$ works by returning a state and reward based on a state and an action: $\mathcal{T}(s \in \mathcal{S}, a \in \mathcal{A}) = s', r \in \mathcal{R}$ is a deterministic transition function where a state and action always results in a new state, which is the main focus of this paper. There also exist non-deterministic transition functions where several states and associated rewards may transition with different probabilities. The goal of solving an MDP is to find a policy for every state in order to maximize the discounted reward:

$$\pi_{\text{opt}} : \mathcal{S} \to \mathcal{A} = \max_{\pi} \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1})$$

Q-learning is an approach to iteratively learn the best actions to take in a given state by estimating the value of the next state [5]. The Q-value of a state-action pair $Q(s,a)$ is the expected discounted reward of the next state. In a traditional Q-value approach these Q-values are stored in lookup tables and are updated via the difference between the expected and experienced reward based on Bellman's equation for the definition of the Q-value:

$$Q(s,a) := r + \gamma \max_{a'}(Q(s', a')) \tag{1}$$

$$\text{diff} = r + \gamma \max_{a'}(Q(s', a')) - Q(s,a) \tag{2}$$

Where $Q(s', a')$ is the expected discounted reward of the next state. How this is applied to update the Q values is a matter of choice, although a common update is to update the Q-values by $|\text{diff}|^2$, similar to an MSE loss. Equation 2 for an L1 type loss leads to the normal Q-table update rule with learning rate α:

$$Q(s,a) = Q(s,a) + \alpha\,(\text{diff}) \tag{3}$$

In [7, 8], Lin et. al propose splitting the rewards into positive and negative streams and changing both the reward and update rules based off of two hyper-parameters for each stream (total of four): W_+, W_-, λ_+, and λ_-:

$$\text{diff}^{+/-} = W_{+/-}r + \gamma \max_{a'}(Q(s', a')) - Q(s,a) \tag{4}$$

$$Q^{+/-}(s,a) = \lambda_{+/-}Q^{+/-}(s,a) + \alpha(\text{diff}^{+/-}) \tag{5}$$

action taken is based off combining the two streams and maximizing the Q-value through both streams

$$a_t = \max_{a}(Q_t^+(s,a) + Q_t^-(s,a)) \tag{6}$$

This paper had interesting results in modeling human reward processing disorders and showed some emergent behavior in agents that matches clinical descriptions and is the basis for the Deep Split Q-learning approach. The λ hyper parameter is chosen between [.1,1] and represents the ability to remember long-term information about rewards or punishments. The W hyperparameter represents the sensitivity of the agent to instantaneous reward or instantaneous punishment. This paper adopts smaller modifications than in [8], where the authors were looking at reward processing disorders. The main issue with adopting this to a deep Q-learning approach is that equations 4, 5 do not follow from equations 1, 2 which are the normal baseline for the deep Q-learning update rules.

2.2 Convolutional DQNs and Ms. Pacman

Gallagher wrote two early papers about Q-learning in Pacman [3, 4]. In [4], Gallagher wrote about tree-based and script-based approaches to controlling agent behavior. Gallagher later wrote [3] about the ghosts logic and its emergent quasi-stochastic nature due to variety in agent play and differences in implementation.

It appears the best policy for Ms. Pacman is a balance between avoiding the ghosts, drawing the ghosts to power pellets, and devouring pellets at a constant rate [1]. Ms. Pacman is a difficult game for a convolutional architecture because most Q-learning approaches to Pacman type games involve representing the state space with features such as “distance to nearst ghost”, “number of power pellets” etc. One of the more difficult things for an image-based model to learn will likely be a representation of when the ghosts are devourable. After all, there is nothing in the state that is passed that would indicate a power pellet has just been eaten. For this reason, it is advisable to pass several frames, since the only indication of a devourable ghost is a color change and blinking status indicates they are about to change back.

In recent years, the posted Berkeley AI course has released a stochastic Pacman MDP for students and aficionados to implement basic Q-learning approaches[1], which is still adopted for research today [8]. The issue with the full Ms. Pacman game is that the state space is incredibly large when one considers all the possible arrangements of dots, ghosts, and protagonist. As such, feature representations of the state space or simplified versions of the game are common [1]. Indeed, the modern approach is to use a deep neural network to learn the features and state representations from images [9]. Mnih et al's work [9] and later Van Hasselt et. al [11] laid down a framework for deep convolutional networks and reinforcement learning in Atari games. The state representation is usually some number of past frames (usually around 3-4) that are downsampled and grayscaled and passed through convolutional layers and output as Q-values for the various actions in the space. With this approach, the Q-value is a function of the model as well: $Q(s,a;\boldsymbol{\theta}_t)$ with model $\boldsymbol{\theta}$. Equation 3 becomes an update rule for the parameters:

$$Y_t^Q = R_{t+1} + \gamma \max_{a}(Q(S_{t+1}, a; \boldsymbol{\theta}_t) \tag{7}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q(S_t, a; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, a_t; \boldsymbol{\theta}_t) \tag{8}$$

Equation 8 resembles an L1 loss function where α is the learning rate and in practice, this is exactly how the model is updated and is called a Deep Q-learning Network (DQN). One issue with this approach in a deep neural network is that the “true” Q-value depends on the model's output. Two ways to deal with this are to use a policy network that is updated every iteration [9], or to use Double-Q learning [11]. The first approach uses a second network for calculating the Q-values in equation 7 that is held stable for τ iterations and then copied over from the policy network.

In double-Q learning, there are also two networks, $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ where one is used to select the best action in equation 7 while the other is used to calculate the Q-value. In this paradigm, equation 7 becomes:

$$Y_t^Q = R_{t+1} + \gamma(Q(S_{t+1}, \max_{a}(Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}') \tag{9}$$

$\boldsymbol{\theta}'$ is updated by copying $\boldsymbol{\theta}_t$ ever τ episodes. This architecture will be referred to as a Double Deep Q-learning Network (DDQN). Despite these approaches, Ms. Pacman remains one of the worst performing games in the architecture described in [11], performing at less than 10% of a human at random starts, although most Atari games were able to achieve scores more comparable with a human competitor. In fact, Ms. Pacman is in the worst 10 performing games tested in [11] out of 56 total games.

[1]http://ai.berkeley.edu/reinforcement.html

3 METHODS

3.1 Baseline Architecture

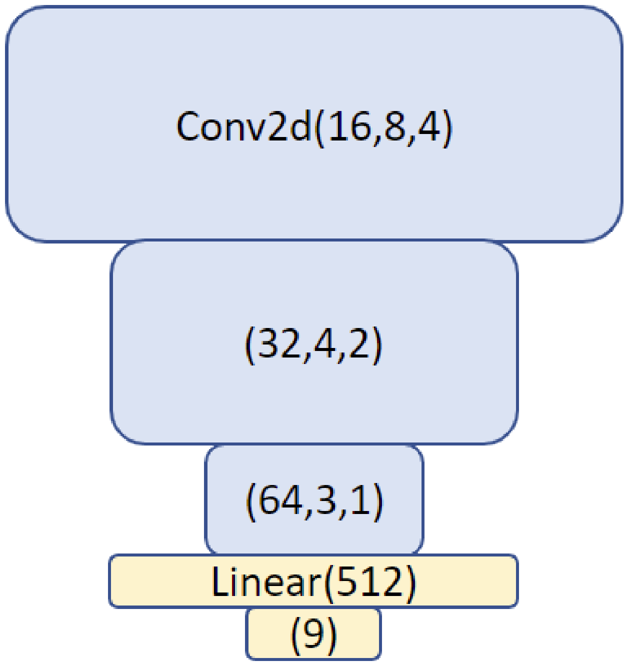


Figure 2: Architecture for the CNN used in all models. Convolutions are in (channels, kernel size, stride) and relu activations follow all layers except the final layer

The baseline architecture for the CNN in this paper are based on [11] since they achieved good results in other Atari games and can be assumed to deliver a good feature representation of the state space. It is possible that Ms. Pacman did not perform well because the feature representation is inadequate but this paper does not explore that possibility. The baseline MDP is based on the gym environment[2] although a deterministic version is used instead of one that randomly assigns a number of frames per action. A state is represented by four successive Atari screen images of original size 3x210x160 which is grayscaled and downsampled to 1x105x80 and concatenated for an input of size 4x105x80. Three convolutional layers then process the images with relu activations. These are then flattened and fed through two linear layers to output into the action space of nine possible actions. As in [11] and [9] an experience replay buffer is maintained with a memory size of 50000 states and sampled in batches of 64.

The idea of using three convolution layers instead of two is targeted toward the split Q-learning approach. The idea is that higher fidelity on a number of high-level features will allow the split streams of reward and punishment to focus on what is important for their specific streams, while the W and λ parameters tune how much the agent focuses on those feature maps. It would be possible to build a ResNet type architecture [6], but as this is an Atari, that is probably too complicated for the types of images the agent has to interpret.

The loss function chosen in [11] is not explicitly stated although it seems to be some form of L1 loss. The authors do mention that they clip the rewards to be in the range (-1, 1) which helps the DQN converge on an optimal policy faster. This poses a problem for one of the proposed changes to the environment (reward modification), so instead OpenAI's[3] interpretation of this clipping is used and Huber Loss is chosen:

$$\mathcal{L}(x,y) = \begin{cases} \frac{1}{2}(x-y)^2, & |x-y| < 1 \\ |x-y| - \frac{1}{2} & \text{otherwise} \end{cases}$$

Gradient clamping is also used so the gradient elements are in the range (-1,1), but this allows the rewards themselves to be scaled at will. The gradient steps are optimized using the RMSprop optimizer with momentum of .95.

Another modification to the baseline architectures is that with one computer, it is difficult to perform multiple experiments running for upwards of 50M iterations in a timely fashion. Instead, 6000 epochs are chosen, where an epoch is defined as one run of reaching a terminal state. The percentage of time a random action vice the policy optimal action is chosen, ϵ, is scaled from 1 to .01 for between 100K and 500K iterations (about 35% of the training time). The updating between $\boldsymbol{\theta}$ and $\boldsymbol{\theta'}$ in DQN and DDQN are done at intervals between 100 and 500 epochs. The point of these parameters is to ensure the space is adequately explored, and the predicted Q-values are stable enough to encourage learning, without being stagnant enough to develop bad habits (see figure 3).

3.2 Reward Modifications and Split Q-learning

Without access to the game's RAM, it is not possible to develop robust incentives for the agent. However, one strategy for Ms. Pacman is to simply stay alive to fight another day rather than chase after dots. This may be accomplished by changing the reward for losing a life from 0 to -10. In order to make a fair comparison, however, the original reward is used to report the overall score for the agent. It is likely that this would result in a policy that is not optimal for maximizing the score, but this could be a useful step in a type of transfer learning experiment where we are teaching the agent to survive first. A successful evaluation of this technique would yield an agent that learns to achieve higher scores more quickly or learns to stay alive for more frames. A type of reward modification not possible while using images as the input is to reward eating power pellets or eating ghosts (beyond the increase in score) as a type of transfer learning to train the agent to seek this behavior. This is problematic because standard Q-learning may not realize that the only way to increase the score is to eat all of the pellets in a level (thus reloading the pellets for the next level).

The split Q-learning approach was adapted by Lin et. al in order to more fully model some of the issues in human cognition that aren't completely addressed in Reinforcement Learning paradigms such as adaptive-aversive interactions [2]. The separate streams are actually modelled off an emerging idea in neuro biology that biological agents can process rewards and punishments through

[2]https://gym.openai.com/envs/MsPacman-v0/

[3]https://openai.com/blog/openai-baselines-dqn/

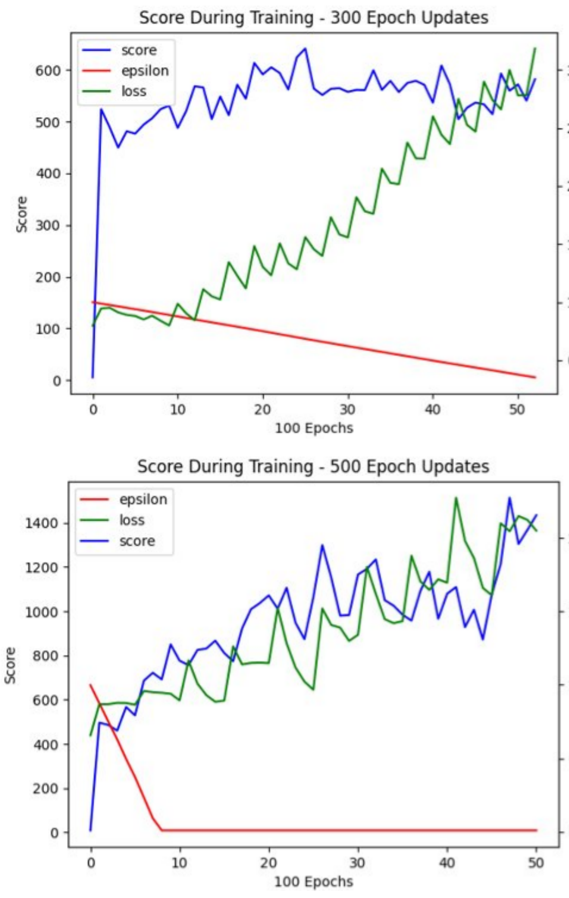


Figure 3: Top: Short target updates with slow exploration ramping leading to increasing loss but stagnating score. Bot: Longer target updates with faster exploration ramping leading to increasing loss and score. Both models are baseline DQN architectures.

independent channels, leading to different behaviors than a single signal [8].

The other form of reward modification is not to modify what rewards are reported to the agent, but how the agent responds to positive and negative rewards. This Split Q-learning Network (SQN) is adopted from [8]. The W parameters for the positive and negative networks are easy to implement by scaling the reward returned by the environment, but in order to implement the λ parameter from equation 5 into equation 8, the effects of equation 5 on the normal Q-learning formulation in equation 3 have to be assessed to develop an equivalent Y_t^Q in equation 7. The issue is there are essentially two updates in equation 5. The first is updating the Q-value toward the diff value, and the second is updating the Q-value toward $\lambda Q(s,a)$. If we modify equation 5 to look like Bellman's equation and calculate the update, we'll have something that is

Algorithm for Deep Split-Q Learning

```
$\theta_0^+, \theta^{+'}, \theta_0^-, \theta^{-'}$ ←init_model()
for each epoch e, do:
  Initialize state $s_0$ = initial_state
  Repeat for each step t in e (until terminal)
    Initialize $\epsilon \leftarrow$ get_epsilon()
    if random.rand() < $\epsilon$
      $a_t \leftarrow$ action_space.sample()
    end if
    else
      $a_t \leftarrow \max_a(Q^+(s_t, a; \theta_t^+) + Q^-(s_t, a; \theta_t^-))$
    end else
    Observe $s_{t+1}, r_{t+1}^+, r_{t+1}^-$ from action $a_t$
    Push $s_t, a_t, r_{t+1}^+, r_{t+1}^-, s_{t+1}$ to replay buffer
    $S$ ←Sample batch from replay buffer
    for $s_t, a_t, r_{t+1}^+, r_{t+1}^-, s_{t+1}$ in $S$
      $Y_t^+ \leftarrow W_+ r_{t+1}^+ + \gamma \max_a(Q^+(s_{t+1}, a; \theta^{+'}))$
          $-(1-\lambda_+)Q^+(s, a; \theta^{+'})$
      $Y_t^- \leftarrow W_- r_{t+1}^- + \gamma \max_a(Q^-(s_{t+1}, a; \theta^{-}{}'))$
          $-(1-\lambda_-)Q^-(s_t, a; \theta^{-'})$
      $\theta_{t+1}^+ \leftarrow \theta_t^+ + \alpha \nabla_\theta^+ \mathcal{L}(Y_t^+, Q^+(s_t, a_t; \theta^+))$
      $\theta_{t+1}^- \leftarrow \theta_t^- + \alpha \nabla_\theta^- \mathcal{L}(Y_t^-, Q^-(s_t, a_t; \theta^-))$
    end for
  end Repeat
  if e mod $\tau$ == 0
    $\theta^{+}{}', \theta^{-}{}' \leftarrow \theta_t^+, \theta_t^-$
  end if
end for
```

transferable to the Deep Q-learning paradigm:

$$Q^+(s,a) := \lambda_+ Q^+(s,a) + W_+ r + \gamma \max_{a'}(Q^+(s',a')) - Q^+(s,a) \tag{10}$$

$$\text{diff}^+ = (\lambda_+ - 1)Q^+(s,a) + W_+ r + \gamma \max_{a'}(Q^+(s',a')) - Q^+(s,a) \tag{11}$$

If we set $\lambda = 1$, equation 12 simplifies to Bellman's equation. This interpretation of the update is a modified definition of the Q-value but allows us to generalize to DQN. If we calculate the $(\lambda - 1)Q^+(s,a)$ term with the target network and leave the $Q^+(s,a)$ as the policy network, then the update rules for a Deep Q-learning version of Split Q-learning become:

$$Y_t^+ = W_+ R_{t+1} + \gamma \max_a(Q(S_{t+1}, a; \theta_t^{+'})) - (1-\lambda_+)Q^+(s,a;\theta_t^{+'}) \tag{12}$$

$$\theta_{t+1}^+ = \theta_t^+ + \alpha(Y_t^+ - Q(S_t, a; \theta_t^+))\nabla_{\theta_t^+} Q(S_t, a_t; \theta_t^+) \tag{13}$$

These equations are used for the training loop in the full split Q-learning algorithm.

Because of the DQN nature of this implementation, four total networks must be used, a target and policy network for both the reward stream and punishment stream. This decreases the available memory on the GPU. It is possible to run the policy models on the GPU and the target models on the CPU but in practice this slowed down the training too much. To help with the slower training time, the models were trained for 5000 epochs and the replay buffer memory size was reduced. Although there have been attempts at a split Deep Q-learning architecture, as in [10], scaling the rewards and gradients as an adaptation of [8] appears to be a novel approach. In the algorithm presented for Deep Split Q-learning, the practical update to $\boldsymbol{\theta}$ is done via the Huber loss and gradient descent using RMSprop as in the deep learning architectures.

The two experiments run in split-Q learning are to decrease the gradient for positive rewards α_+ to .7 and to increase the immediate response to negative rewards λ_- to a factor of 10. In this way, the agent may learn to pay more attention to the behavior of the ghosts (remembering negative future rewards more than positive and more attention to immediate punishments than positive rewards). This is compared to the opposite modifications in case seeking rewards may induce the agent to clear the board more often and reload the pellets. The modified penalty for losing a life is also used in both experiments to simulate a feedback stream for being closer to the end of the game. Additionally, a penalty of -1 is applied to each frame in order to keep the negative stream receiving non-zero input and to impart a sense of urgency to the agent as in [8]. A novel approach is to *also* impart a +1 reward for staying alive another set of frames. In this manner, the *net* effect is zero, but it allows the different reward processing from the λ and α parameters to choose which is more important.

In order to see if the act of splitting the streams alone has an effect, a split-Q learning experiment was also performed with 1s for all of the hyper-parameters. Although this should converge to the basic Q-learning implementation, it is possible to improve because there are separate convolutional models to predict and process rewards and punishments. This may lead to finer attention to relevant features in the separate models' parameters.

Agent	α_+	α_-	λ_+	λ_-
Baseline	1	1	1	1
Prolong life	.7	1	1	10
Seek reward	1	.7	10	1

Table 1: Hyperparameters for SQN experiments. The Prolong life agent is sensitive to negative rewards (losing a life), while the Seek reward agent is more sensitive to rewards and less sensitive to future loss

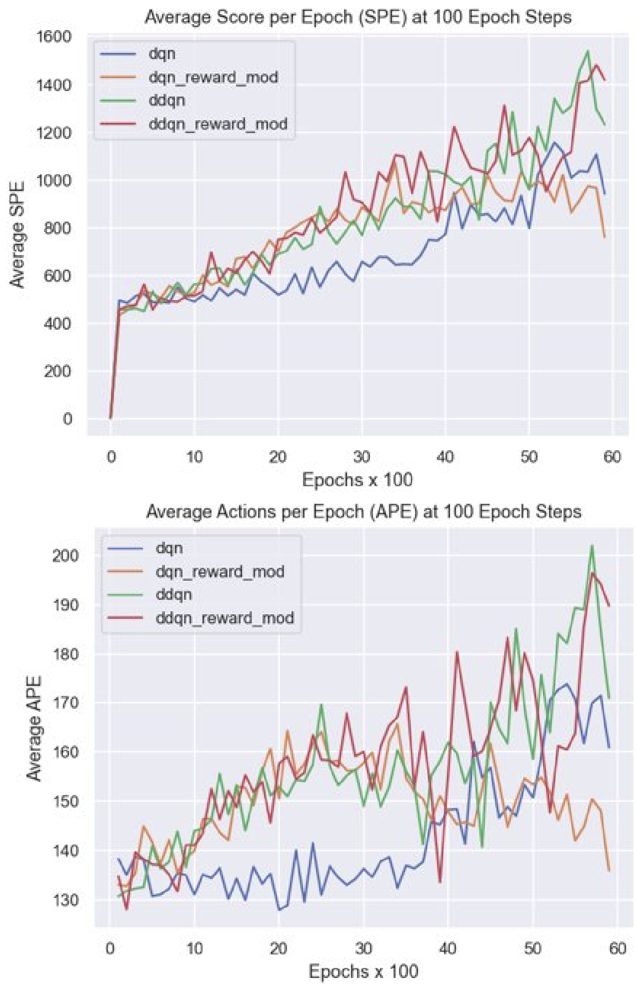


Figure 4: Score per Epoch (SPE) and Actions per Epoch (APE) averaged over 100 epoch periods for the baseline and reward modified versions of DQN and DDQN.

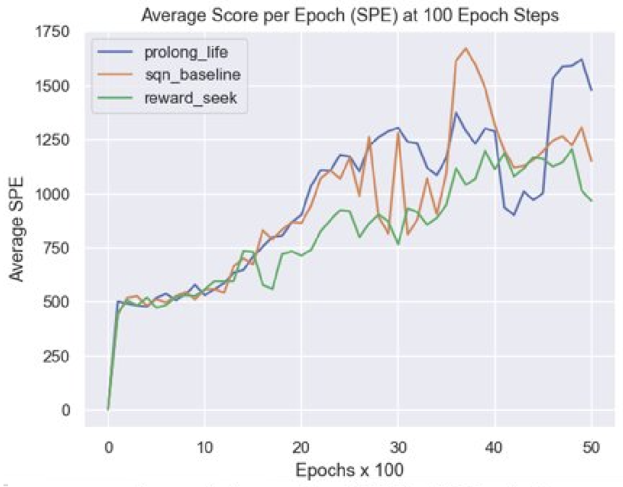
4 EVALUATION

4.1 Deep Q-Learning and Double Deep Q-Learning

As mentioned in section 3.1, these training sessions were done over 6000 epochs while scaling epsilon from 1 to .01 over 300,000 iterations with a replay buffer of 50K states. The baseline results for Ms. Pacman are found in [11], although they trained their models for 50M iterations (6,000 epochs is around 1M iterations) with a replay buffer of 1M states.

In order to find the driver of agent behavior, experiments were conducted with baseline DQN and DDQN networks and evaluated on average Actions per Epoch (APE) over 100 epochs and average Score per Epoch (SPE) over 100 epochs. These results are shown in figure 4. It appears that the reward modification helped the vanilla DQN the most at the start of training, although was less helpful toward the end of the 600 epochs. Although the average of the last 400 epochs was highest for the DDQN with reward modification, it appears from figure 4 that the reward modification did not significantly boost the performance of the DDQN.

An interesting result from figure 4 is the relationship between APE and SPE. For both the vanilla and reward modified versions, it appears that there is an overall march toward maximizing efficiency in SPE while the overall shape of the APE is more highly varied and its effects can be seen in the SPE. It is as if the agent still doesn't really care about remaining alive (even with a -10 penalty for losing a life) but is still able to maximize its rewards over a slowly growing lifespan in each game. This suggests that longer training time may be needed for the game state to progress deep enough for higher level strategies to become apparent in the Q-values.



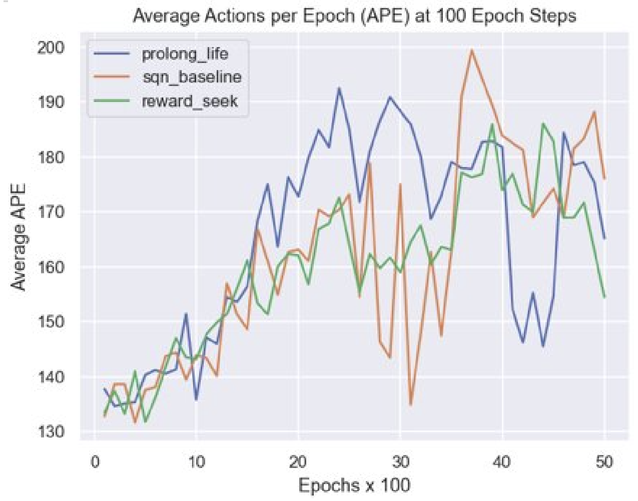


Figure 5: Score per Epoch (SPE) and Actions per Epoch (APE) averaged over 100 epoch periods for the various Split Q-learning models.

4.2 Split Q-Learning

This was the most interesting idea to evaluate as Split Q-learning allows models to focus on what is important for their stream. In this way, it may be possible for the negative stream model to pay much more attention to the ghosts than the positive stream model for example. Because of the need to use four models (a policy and target model for both the reward and punishment streams), the training time and memory requirements are more robust than for the DQN and DDQN implementations. To combat this, the training is done over 5000 epochs and the replay buffer size is reduced from 50K to 30K states. In these experiments all four models were run on the GPU to improve epoch time to process.

As initially predicted, the Prolong life agent performed the best out of not only the Split Q-learning agents, but all of the author's agents. Interestingly, the Prolong life agent did not actually have the longest APE indicating a longer survival time but did have the highest score. In general, the SQN agents also exhibited more stability in the training process as evidenced by figure 5. It could be that training a "Prolong life" type agent for several million episodes and then switching to a more traditional approach could encourage the kind of *careful* but guided exploration needed for a game like Ms. Pacman. In all, it is interesting that the DDQN with reward modification had the best APE but the "Prolong life" agent managed the best score even with a smaller APE.

Agent	Training iterations	SPE	APE
DQN	860k	1014.6	166.8
DQN + reward mod	898k	883.3	412.7
DDQN	944k	1437.4	191.4
DDQN + reward mod	952k	1491.5	**196.8**
SQN Baseline	807k	1237.5	182.3
Prolong life	828k	**1528.3**	178.5
Seek reward	800k	1083.7	164.6
Random	-	307.3	-
DQN baseline	50M	2311	-
DDQN baseline	50M	3210	-
Human baseline	-	15693	-

Table 2: APE and SPE comparisons between models and baseline. APE and SPE are the mean of the last 400 epochs of training. Best of author's results are in bold, and the baselines are from [11]

.

4.3 DSQN and Longer Training Time Experiments

In order to compare some of the hyper-parameters and nuances between models, a Double Split Q-learning Network (DSQN) implementation was tested against the baseline and reward modified DDQN. The Double Split Q-learning network is implemented by calculating the Y_t^+, Y_t^- via the Double Q-learning method in equation 9. These experiments were trained over 10,000 epochs with an ϵ ramp from 1 to .1 over 500k iterations. The reward modifications, DSQN hyperparameters, and average from the last 500 epochs of training in APE and SPE is shown in table 3

Agent	Iterations	Death penalty	SPE	APE
DDQN	2.01M	0	2154 ± 78	223 ± 4
DDQN	2.02M	10	2172 ± 83	224 ± 4
DDQN	2.01M	100	**2317** ± 132	**227** ± 4
PL	2.04M	100	2162 ± 73	226 ± 4
SR	1.49M	100	729 ± 51	154 ± 6
CP	-	100	2216 ± 87	225 ± 5

Table 3: APE and SPE comparisons between DDQN and DSQN models. DSQN models are Prolong life (PL), Seek reward (SR), and Chronic pain (CP). APE and SPE are the mean $\pm\sigma$ of the last 5000 epochs of training.

Additionally, the Chronic Pain model in [8] showed some clinical signs of chronic pain patients in the Q-table implementation of Pacman so an attempt was made to replicate those results in a deep Q-learning framework. Lin et. al's definition of a model representing

"Chronic Pain" is a setting of 1 for λ_-, α_- and a setting of .5 for λ_+, α_+. This was trained in the same manner as the longer training experiments. What is interesting about this environment is the chronic pain performed fairly well both in terms of overall score and living time. For comparison, video of this agent is provided [4]. This could have been due to the muting of the reward system, which can sometimes be in the 100s of points in one time step. The best performing model was the DDQN model that punished deaths with a -100 reward, both in average APE and SPE over the last 5000 epochs (well after ϵ had reached .1).

5 CONCLUSION AND FUTURE WORK

This work was a look at modifications to the gym Ms. Pacman default environment in order to improve performance. Given more time, more training epochs and more hyperparameters would have been tested in order to find the best improvements to the reward function. As far as the author is aware, this paper also represents a novel implementation of Lin et. al's Split Q-learning approach with a deep neural network architecture. The deep split Q-learning approach does appear to have an interesting effect on the model's prioritization of staying alive vice seeking pellets.

Aside from training the models for a longer time period to ensure the score becomes asymptotic, another good modification would be to use the game's RAM state in order to further modify the rewards and punishments, especially for the split-Q learning. In this way, the agent could receive incentives for eating power pellets or ghosts, something that it seems the agent does not learn to prioritize in the proposed architectures. The author believes Deep Split Q-learning is a viable approach to improving the scores for reinforcement learning in Pacman / Ms. Pacman and further work on tweaking the hyper parameters and reward streams may yield fruit. It is unfortunate this attempt ran into computation bottlenecks and a more robust evaluation couldn't be conducted. I do intend to at least train the Prolong life split Q-learning model to asymptotic behavior to see if it can beat the baseline in [11].

REFERENCES

[1] Luuk Bom, Ruud Henken, and Marco Wiering. 2013. Reinforcement learning to train Ms. Pac-Man using higher-order action-relative inputs. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. 156–163. https://doi.org/10.1109/ADPRL.2013.6615002

[2] Peter Dayan and Yael Niv. 2008. Reinforcement learning: The Good, The Bad and The Ugly. *Current Opinion in Neurobiology* 18, 2 (2008), 185–196. https://doi.org/10.1016/j.conb.2008.08.003 Cognitive neuroscience.

[3] Marcus Gallagher and Mark Ledwich. 2007. Evolving Pac-Man Players: Can We Learn from Raw Input?. In *2007 IEEE Symposium on Computational Intelligence and Games*. 282–287. https://doi.org/10.1109/CIG.2007.368110

[4] M. Gallagher and A. Ryan. 2003. Learning to play Pac-Man: an evolutionary, rule-based approach. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, Vol. 4. 2462–2469 Vol.4. https://doi.org/10.1109/CEC.2003.1299397

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[6] H. Kaiming, Z. Xiangyu, and R. Shaoqing. 2015. *Deep residual learning for image recognition.* arXiv preprint arXiv:1512.03385.

[7] Baihan Lin, Djallel Bouneffouf, and Guillermo Cecchi. 2019. Split Q Learning: Reinforcement Learning with Two-Stream Rewards. arXiv:1906.12350 [cs.LG]

[8] Baihan Lin, Guillermo Cecchi, Djallel Bouneffouf, Jenna Reinen, and Irina Rish. 2020. A Story of Two Streams: Reinforcement Learning Models from Human Behavior and Neuropsychiatry. arXiv:1906.11286 [cs.LG]

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[10] Iason Sarantopoulos, Marios Kiatos, Zoe Doulgeri, and Sotiris Malassiotis. 2020. Split Deep Q-Learning for Robust Object Singulation. arXiv:1909.08105 [cs.RO]

[11] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs.LG]

[4]https://github.com/Kickflip89/DQN_project/videos/