1 Packages Used

- OpenAI Gym: The gym package is used to initialize and create a game environment for the ping pong game
- Numpy package:Used for mathematical operations
- Pickle: Used for saving and loading model weight parameters.

2 How the model works

The model is a simple two layer linear regression neural network model constructed as follows:

$$\begin{aligned} z_1 &= w_1 x_1 + b_1 \qquad \text{First layer} \qquad (1) \\ z_2 &= w_2 x_2 + b_2 \qquad \text{Second layer} \qquad (2) \end{aligned}$$

where w_1, w_2 are the weight parameters and b_1, b_2 are the bias parameters. Each layer is activated by the sigmoid activation function, this activation function introduces non-linearities in the neural network. The input x_1 in our case will be the image-array (matrix) that is flattened into a 6400×1 vector from an 80×80 image array during the pre-processing phase. The weights w_1 in the first layer of the neural network will have dimensions of 200×6400 matrix representing the the weights of the hidden layer, from the dimensions of the weight w_1 we can see that the hidden layer contains 200 neurons. The second output layer has dimensions of 1×200 matrix representing the weights of the output, the input x_2 will the output vector coming from the hidden layer activation function, $\sigma(z_1) = \sigma(w_1 x_1 + b_1)$, this vector will have dimensions of 200×1. The matrix multiplication between the w_2 and x_2 will produce a dimension 1 value. It follows that the dimensions of the bias parameters b_1 and b_2 should be 1×200 and 1 respectively. The sum of the output layer $w_2 x_2 + b_2$ will activated by a sigmoid function which will yield a value between 0 and 1, which will be interpreted as a probability.

The overall function of the model will be to take in a sequence of images representing the frame state of the Pong game. The image will be feed forwarded into the neural network and the model will output a probability (between 0 and 1), this value will used to decide whether the AI pong agent should move-up or down based on some policy rule. The model will be rewarded +1 if it makes a move that helps it win the game, it will get -1 reward for moves that makes it lose the game, and the 0 reward will be when the move doesn't help to win or lose the game. An episode count will be when the AI agent wins or loses the game, this will be after 21 rounds and the overall score will be calculated. If the score is -21:0, this means the AI agent didn't win any game, if the score is -16:5, at least the AI agent has won 5 games. The main objective of the model is to train the AI agent such that it learns to play the pong game so well to an extent that it can beat the opponent. The learning of the AI agent will be achieved by applying back propagation gradients to the weights and updating the weights model after the end of each episode. The goal of the policy gradient (back propagation) is to optimize the weights towards the direction that led to moves which resulted in the AI agent winning a around during an episode, so that the network can generate more these actions. Furthermore, moves that took place before an episode will be discounted (weighted less) as compared to moves that happend toward the end of the episode, since the moves that happened towards the end of an episode would have largely contributed to the AI agent winning the game.

Since we are using back propagation to train an unsupervised problem, we will need to create artificial or fake labels in order to turn the problem into a classification problem, so that the gradients of the weights can be properly directed in an optimum direction using the labels. We will create an artificial label 1 when the movement of the AI is up and artificial label 0 when the movement is down.

2.1 Feed forward Neural Network

Equations for the first layer (hidden layer) of neural networks,

$$\begin{aligned} z_1 =& w_1 \cdot x + b_1 && (3)\\ \sigma_1(z_1) =& \sigma(w_1 \cdot x + b_1) && (4)\\ =& \frac{1}{1+e^{-z_1}} && (5) \end{aligned}$$

Equations for the second layer (output layer) of neural networks,

$$\begin{aligned} z_2 =& w_2 \cdot \sigma_1(z_1) + b_2 && (6)\\ \sigma_2(z_2) =& \sigma(w_2 \cdot \sigma_1(z_1) + b_2) && (7)\\ =& \frac{1}{1+e^{-z_2}} && (8) \end{aligned}$$

2.2 Gradient Descent (Policy Gradient)

For computation of back propagation we will use the loss function (cost function)

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} y_n \log \hat{y} \tag{9}$$

where $y = \sigma(z_2)$ is the ground truth and $\hat{y}$ is the estimated outcome. The deriavative of the loss function with respect the second layer (output layer) weight w_2 yields:

$$\begin{aligned} \frac{\partial L(y,\hat{y})}{\partial w_2} =& \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2} \frac{\partial z_2}{w_2} && (10)\\ =& (y - \hat{y}) * \frac{\partial \hat{y}}{\partial w_2} * \sigma(z_1) && (11)\\ =& (y - \sigma(z_2)) * \sigma'(z_2) * \sigma(z_1). && (12) \end{aligned}$$

The derivative of the loss function with respect the first layer w_1 gives,

$$\begin{aligned}
\frac{\partial L(y,\hat{y})}{\partial w_1} =& \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_1}\frac{\partial z_1}{w_1} && (13)\\
=&(y-\hat{y})\frac{\partial \hat{y}}{\partial z_1} * x && (14)\\
=&(y-\hat{y}) * \frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial \sigma(z_1)}\frac{\partial \sigma(z_1)}{\partial z_1} * x && (15)\\
=&(y-\hat{y}) * \sigma'(z_2) * w_2 * \sigma'(z_1) * x && (16)
\end{aligned}$$

where x will be the image matrix (array).
The derivatives for the cost function with respect the bias terms from the first and second layers will be respectively:

$$\begin{aligned}
\frac{\partial L(y,\hat{y})}{\partial b_2} =& \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{b_2} && (17)\\
=&(y-\hat{y}) * \frac{\partial \hat{y}}{\partial b_2} && (18)\\
=&(y-\sigma(z_2)) * \sigma'(z_2). && (19)\\
\frac{\partial L(y,\hat{y})}{\partial b_1} =& \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_1}\frac{\partial z_1}{w_1} && (20)\\
=&(y-\hat{y}) * \frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial \sigma(z_1)}\frac{\partial \sigma(z_1)}{\partial z_1} * x && (21)\\
=&(y-\hat{y}) * \sigma'(z_2) * w_2 * \sigma'(z_1) && (22)
\end{aligned}$$

The weights and bias parameters will be updated using RMSProp. The update using RMSProp for the weights parameters will be,

$$\begin{aligned}
E^w[g^2]_t =& \beta * E^w[g^2]_{t-1} + (1-\beta)(g_t)^2, \qquad \beta = 0.9 && (23)\\
=& 0.9 * E^w[g^2]_{t-1} + 0.1 g_t^2 && (24)\\
w_{t+1} =& w_t - \frac{\eta}{E^w[g^2]_t + \epsilon} g_t && (25)
\end{aligned}$$

And the update RMSProp for the bias parameters will be,

$$\begin{aligned}
E^b[g^2]_t =& \beta * E^b[g^2]_{t-1} + (1-\beta) g_t^2, \qquad \beta = 0.9 && (26)\\
=& 0.9 * E^b[g^2]_{t-1} + 0.1 * g_t^2 && (27)\\
b_{t+1} =& b_t - \frac{\eta}{E^b[g^2]_t + \epsilon} g_t && (28)
\end{aligned}$$

3 Bonus Questions

1. Instead of using a linear regression Artificial neural network we can use a Convolutional Neural Network. When we are using the CNN's, it will be unnecessary to first pre-process the data, by first removing the background details, removing the color, reducing and flattening the image dimensions. This process can be taken care in the CNN's layers, using valid padding and pooling operations in the padding

and pooling layers. Both the pooling and valid layers can reduce the dimension of the input image through their matrix multiplication operations while at the same time learning and retaining high level features from the image. The CNN's can also reduce significantly the number of neurons needed in each layer, since in CNN's each neuron is reponsible for a certain area of receptive field and the neurons don't need to be fully connected with all the neurons in the next level layer of neurons. A neuron could only connect to certain neurons in the receptive field. This greatly reduces reduntant information that the neurons can learn and also speeds the process of learning important features from an image.

We can also adopt a q-learning approach for the model instead of using a policy reward approach. Q-learning is an off policy reinforcement learning method, the AI agent will have to find the best actions based on its current state without using any policies, this can be achieved by performing random actions. A q-table and actions table will have to be created and the information about the q-values and their corresponding actions will be stored on those tables, and this information will be used as a reference table for the AI agent to select the best possible moves.

2. Supervised learning involves training a model with data that has labels or classes, the labels helps to direct the gradients of the weights towards an optimum.

In unsupervised learning the model is trained without any labeled data. The model has to discover patterns and hidden structures from the data without any assistance.

3. Classification has to do with labels that are discrete, it is about training a model with labeled data in order for the model to predict labels. Regression has to do with continous values or quantity, the model is trained with an output data that is continous, the model is trained to predict continous values or quantities.

4. Data with class imbalance will lead into the model being biased towards the prevalent class, the accuracy of the model can also be misleadingly high, since the model might only predict the prevalent class and obtain a high accuracy.

5. To mitigate class imbalance we can use a random over sampling technique where the number of the under represented (minority) class is replicated several times so that they are fairly represented. We can also do the opposite and use random under sampling technique where the number of majority class is randomly reduced or removed from the the data.

6. Deep learning has to do with artificial neural networks (ANN), deep learning models are models that are built using ANN structures. Non-deep learning methods could be machine learning methods that are built without ANN structures. These methods might have to do with models that learn from the data, then they apply what they have learned from the data. This may include both classical supervised and unsupervised models such as SVM, KNN and Kmeans clustering models.