

CS - 520

Introduction to Artificial Intelligence

Project 3

By:

Shaurya Jain (sj1029)

Pavan Madamsetty (pm950)

Sai Srichandra Ravuri (sr1972)

Problem Statement

Bot navigates a 50x50 grid (representing a ship) to rescue crew members while avoiding aliens. The bot has sensors that can detect aliens and crew members within a certain range. The assumptions are as follows,

1. The BOT doesn't know the cell location of the crew. It gets a beep if the CREW is d hidden steps away from the BOT with a probability of $e^{-\alpha(d-1)}$
2. The BOT gets a beep **always** if the ALIEN is within $(2k+1) \times (2k+1)$ square centered at the location. The movement of ALIEN remains the same.
3. Since the BOT doesn't know the exact location of the CREW and ALIEN, a **Machine Learning Model** on the probabilities will be used by the BOT to navigate on the ship, detect aliens, and find crew members.
4. The Ship consists of only One Alien and One Crew.

The probability of the Alien and Crew being in a cell keeps changing and this will be used to move the BOT. The key point in solving this project is how we use the data from the simulations in our model to help decide the BOT and the next steps it should take to save the CREW.

1. Setup

- a. **The Ship:** 50x50 grid is generated. The bot, aliens, and crew members are placed on this grid.
- b. **The Bot, the Sensors, the Aliens, the Crew:** The bot can move in four directions or stay in place. It has two types of sensors: one for detecting aliens within a $(2k+1) \times (2k+1)$ square centered on the bot, and another for detecting crew members based on a probability formula. Aliens move randomly to an adjacent empty square every timestep. If the bot and an alien occupy the same cell, the execution is over.
- c. **One Alien, One Crew Member:** In this scenario, the bot starts with the equal belief that the alien and crew member can be in any open cell (with some restrictions). The bot updates its knowledge based on the sensor data and moves towards the cell most likely to contain the crew member while avoiding the alien.
- d. Alien detection square: $k = 16$
- e. Crew detection sensor: α (alpha) = 0.06

2. Bot Challenges

- a. **Grid-Based Environment:** The ship's layout is represented as a grid, where each cell can be either open or closed. The bot's movements are constrained to up, down, left, and right within the grid.

- b. **Crew Member Rescue:** The primary objective is to rescue crew members who are randomly placed in open cells. The bot must find the shortest path to reach the crew members while adapting to dynamic alien movements.
- c. **Hostile Aliens:** The bot encounters a significant challenge posed by Aliens moving across the grid. Its main goal is to move across the grid without collisions with these aliens, prioritizing the safety of the crew members.
- d. **Bot Knowledge:** The bot's knowledge of the alien and crew member locations should be updated based on the sensor data.
- e. **Alien Location:** The bot knows that the alien is equally likely to be in any open cell outside the bot's detection square. When the alien moves, the bot should update its belief about the alien's location. This can be done by considering all possible moves the alien could have made from its possible locations and updating the probabilities accordingly. If the bot's sensor detects the alien, the bot can update its belief to reflect that the alien is within the detection square.
- f. **Crew Member Location:** The bot knows that the crew member is equally likely to be in any cell other than the bot's initial cell. When the bot receives a beep from a crew member, it should update its belief about the crew member's location based on the given crew detection probability formula. If the bot does not receive a beep, it can update its belief to reflect that the crew member is less likely to be in cells close to it.

MODEL 1:

Problem Statement:

This project aims to develop a predictive model that anticipates the next move taken by the Bot within the current state of the simulation. This model will be trained using data generated through repeated iterations of Bot.

Input Space:

The information available for the predictive model includes essential elements crucial for decision-making:

- a) Ship Layout: Represents the ship's structure.
- b) Probability Maps: Two 50x50 maps are flattened to 2500-length arrays, capturing alien and crew probabilities in the ship.
- c) Bot Position: Indicates the bot's current location within the ship.
- d) All possible BOT moves: Defines allowable bot movements in the current state.

Data Preprocessing:

Before training the model, the flattened probability maps and ship undergo preprocessing steps to transform the data into a dataset with 49 columns and around 30,000 rows. These steps likely involve normalization, feature extraction, or dimensionality reduction techniques.

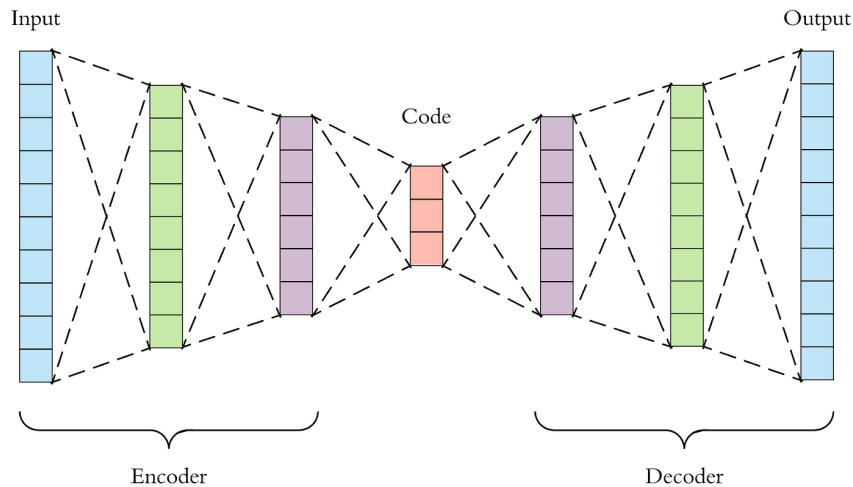
Subsequently, a normalization process is applied to the data using NumPy's PowerTransformer. This method normalizes each feature separately, aiming to approximate a Gaussian-like distribution. The choice of the Yeo-Johnson transformation method is deliberate, as it accommodates input values that may include zeros, ensuring robust handling of such cases during the normalization process.

$$\psi(\lambda, y) = \begin{cases} ((y + 1)^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y + 1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y + 1)^{2-\lambda} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y + 1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Next, we use a MinMax scaler to scale the data into an appropriate range of 0 to 1.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

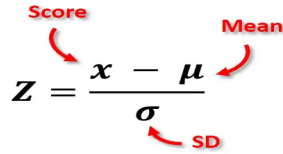
Subsequently, an auto-encoder, a specialized neural network architecture, is implemented for dimensionality reduction purposes. We have two autoencoders, one for reducing the alien probability map dimensions and another for crew probability map dimensions. For the autoencoder, we are using the neural network architecture implemented for bot 1, but here the change is we have 5 hidden layers with 256,64,25,64,256 neurons. Here for the autoencoder, the input and output are the same so that the network learns to reduce the dimensions of data as well as replicate the actual data from the reduced encoding. Thus after training it for 25 epochs with a learning rate of 0.1 using the Gradient Descent Algorithm, we take the data as input for prediction but get the 3rd hidden layer (i.e. the layer with 25 neurons) representation as our reduced data



This method compresses the complex information within the crew and alien probability maps into a more concise latent representation, effectively reducing dimensions to 25. This dimensionality reduction is pivotal for streamlining the dataset while retaining essential information, facilitating more efficient processing and in-depth analysis.

Following this transformation, the NaN values within the dataset are removed. These are done to ensure the integrity and reliability of the dataset for subsequent analytical procedures. This comprehensive NaN value elimination process ensures a consistent and accurate dataset representation, ensuring robustness in subsequent analyses.

Subsequently, a z-score transformation is systematically applied to standardize the dataset. This statistical normalization technique involves centering the mean of the dataset at 0 and scaling its standard deviation to 1. By normalizing the data in this manner, uniformity in scale is achieved across all variables, enabling equitable comparisons between diverse features. This crucial step prepares the dataset for rigorous analyses and modeling techniques, ensuring unbiased contributions from each feature irrespective of their original scales. Ultimately, this meticulous process enhances the dataset's suitability for subsequent comprehensive analyses and model training.

$$Z = \frac{x - \mu}{\sigma}$$


Output space:

The model generates an output in the form of a one-hot encoded vector, delineating four potential classes that correspond to distinct directional movements within the given context. These classes represent the available actions: left, right, up, and down, designated numerically as (0, 1, 2, 3) respectively. This encoding scheme allows for categorical representation, enabling the model to indicate the most probable directional movement for the bot based on training.

Model Space:

The predictive model utilizes a standard neural network architecture to predict the next move of the bot.

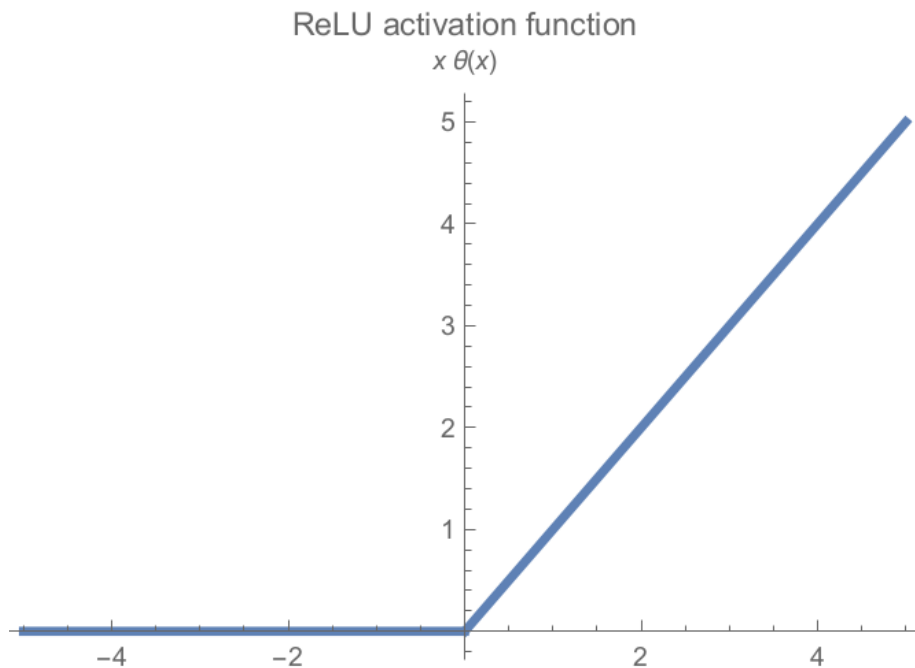
Neural Network:

The neural network comprises four hidden layers with neurons: 32, 16, 8, and 4 respectively. The input layer aligns with the size of the data input, while the output layer size is set to 4, accommodating the directional movement classes.

ReLU(Rectified Linear Activation Function):

Activation functions used in the hidden layers are ReLU (Rectified Linear Activation Function). ReLU functions as a linear activation method that maintains the input as the output if it is positive; otherwise, it outputs zero. This activation scheme is applied across all hidden layers, facilitating the extraction of nonlinear relationships and patterns within the data.

$$f(x) = \max(0, x)$$

**Output Layer: Masked Softmax**

Masked Softmax operates uniquely compared to traditional Softmax by exclusively considering valid predictions. Within our context, the model's accuracy suffers from predictions of moves obstructed by blocked cells. To mitigate this issue, we maintain a log of permissible moves corresponding to each dataset entry. This log guides the scaling of predicted values: valid moves retain their original values (multiplied by 1), while invalid moves are discounted (multiplied by 0), effectively nullifying their impact. This strategic adjustment enables the model to prioritize valid moves, leading to more precise decision-making and enhanced predictive capabilities overall.

Loss Functions: Categorical cross-entropy is being used as the loss function.

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Categorical Cross-Entropy stands as a fundamental loss function widely utilized in multi-class classification scenarios, primarily when dealing with categorical output spaces. This particular function effectively quantifies the dissimilarity between the predicted probabilities and the actual categorical output. When employed alongside the masked softmax activation function, it becomes instrumental in calculating the probabilities associated with the bot's potential valid moves within the given context.

By leveraging Categorical Cross-Entropy coupled with masked softmax activation, the model is adept at discerning the likelihood of the bot's valid moves, essentially highlighting the probabilities attributed to each potential action. This mechanism serves as a critical facet in steering the model towards prioritizing valid movements, thereby refining its decision-making process.

Post this probabilistic evaluation, the prediction mechanism delves into discerning the output class exhibiting the highest probability among the potential valid moves. This approach effectively pinpoints the most probable action for the bot to undertake, maximizing the accuracy and precision of its decisions within the multi-class classification framework.

Training Algorithm:

We are using Gradient Descent for training the algorithm. The gradient descent function tries to converge to a local minima of any differential function. In any ML model, we try to minimize our cost function.

$$*W_x = W_x - a \left(\frac{\partial \text{Error}}{\partial W_x} \right)$$

Old weight Derivative of Error with respect to weight
 ↓ ↓
 New weight Learning rate

Gradient Descent Formula

In our model's architecture, the computation of gradients involves applying the derivative of the Rectified Linear Unit (ReLU) function to determine the gradient of the loss concerning the inputs of neurons activated by ReLU. This derivative calculation is pivotal during the backward pass, contributing to the model's understanding of how changes in these inputs affect the overall loss.

Following this gradient computation, the model's iterative learning process involves updating both weights and biases based on these calculated gradients. This update mechanism, stemming from the derivative of the ReLU function, influences adjustments to the model's parameters during training. The weight and bias updates are instrumental in refining the model's predictive capabilities, iteratively improving its ability to capture intricate patterns and relationships within the data.

Overall, this process of utilizing the ReLU derivative for gradient computation and subsequent parameter updates plays a crucial role in the model's learning journey. It aids in fine-tuning the model's parameters to optimize its predictive accuracy and enhance its capacity to generalize patterns from the provided data.

Learning Rate: 0.05

Batch Size: 128

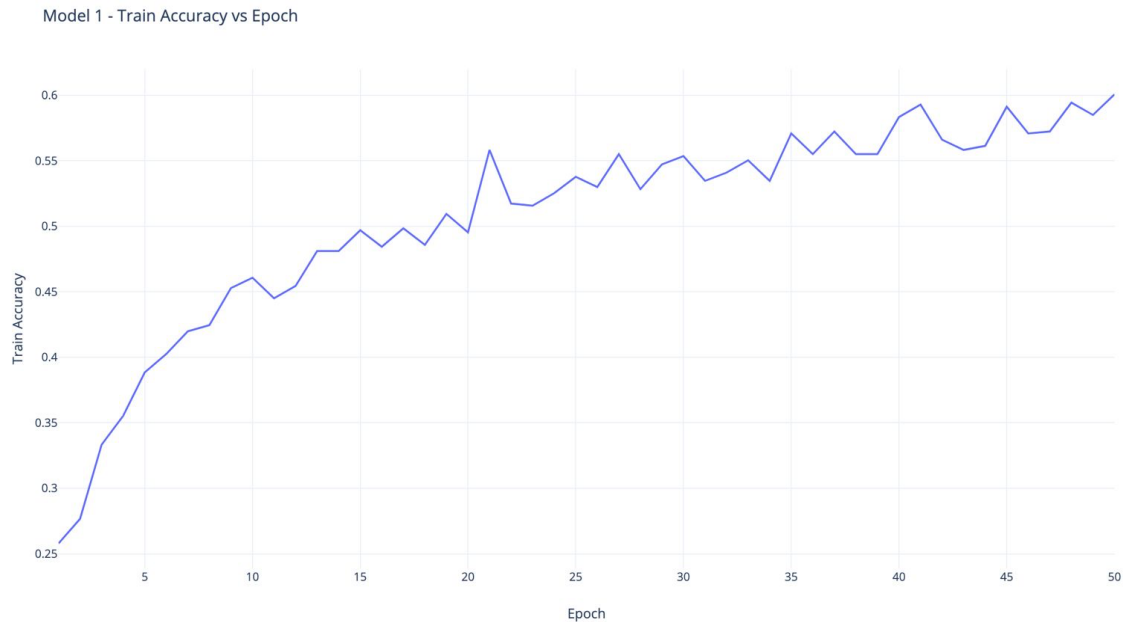
Epochs: 50

Training Results:

Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1	1.450	0.258	1.392	0.269
2	1.384	0.277	1.377	0.319
3	1.353	0.333	1.383	0.319
4	1.334	0.355	1.339	0.369
5	1.305	0.388	1.332	0.369
6	1.287	0.403	1.288	0.469
7	1.260	0.420	1.272	0.388
8	1.242	0.424	1.277	0.356
9	1.207	0.453	1.241	0.406
10	1.194	0.461	1.210	0.425
11	1.168	0.445	1.231	0.419
12	1.158	0.454	1.188	0.444
13	1.145	0.481	1.181	0.469
14	1.126	0.481	1.164	0.444
15	1.110	0.497	1.141	0.512
16	1.097	0.484	1.164	0.481
17	1.097	0.498	1.126	0.494
18	1.086	0.486	1.159	0.506
19	1.078	0.509	1.119	0.519
20	1.065	0.495	1.113	0.512
21	1.043	0.558	1.112	0.500
22	1.049	0.517	1.085	0.519
23	1.042	0.516	1.163	0.438
24	1.027	0.525	1.107	0.450
25	1.042	0.538	1.079	0.494
26	1.014	0.530	1.112	0.456
27	1.012	0.555	1.076	0.494
28	1.022	0.528	1.097	0.463
29	1.011	0.547	1.050	0.556
30	0.996	0.553	1.110	0.512
31	1.000	0.535	1.062	0.575
32	0.995	0.541	1.095	0.519
33	0.972	0.550	1.072	0.531
34	0.973	0.535	1.043	0.575
35	0.972	0.571	1.087	0.506
36	0.981	0.555	1.053	0.519
37	0.969	0.572	1.037	0.562
38	0.966	0.555	1.023	0.544
39	0.969	0.555	1.041	0.481
40	0.938	0.583	1.242	0.463
41	0.940	0.593	1.044	0.487
42	0.936	0.566	1.085	0.512
43	0.945	0.558	1.008	0.556
44	0.928	0.561	1.099	0.500
45	0.934	0.591	1.012	0.569
46	0.936	0.571	1.025	0.562
47	0.909	0.572	0.983	0.537
48	0.916	0.594	1.076	0.494
49	0.927	0.585	1.020	0.531
50	0.905	0.601	0.993	0.575

VISUALISATIONS AND INFERENCES:

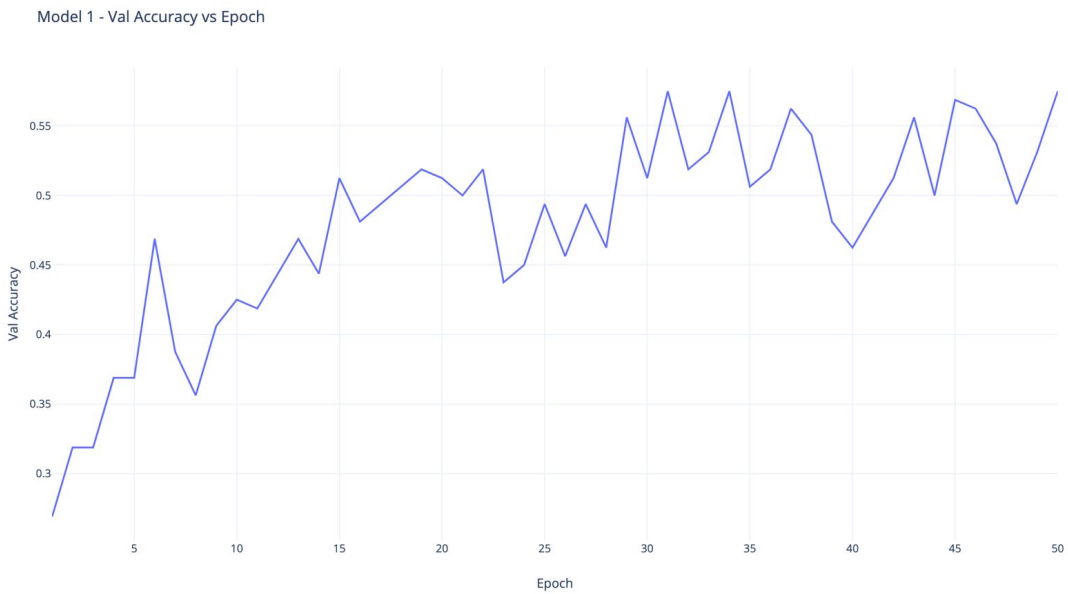
Training Accuracy vs Epoch:



Inferences from the graph:

- The model demonstrates a significant increase in training accuracy, starting at 0.25 and reaching 0.6 by epoch 50 showing that it is effectively learning from the training data.
- The model's reach toward a stable accuracy of 0.6 indicates minimal overfitting on the training data.
- The absence of sharp fluctuations in accuracy throughout the training process suggests a steady and controlled learning pace
- The relatively high final accuracy of 0.6 implies successful task acquisition by the model.

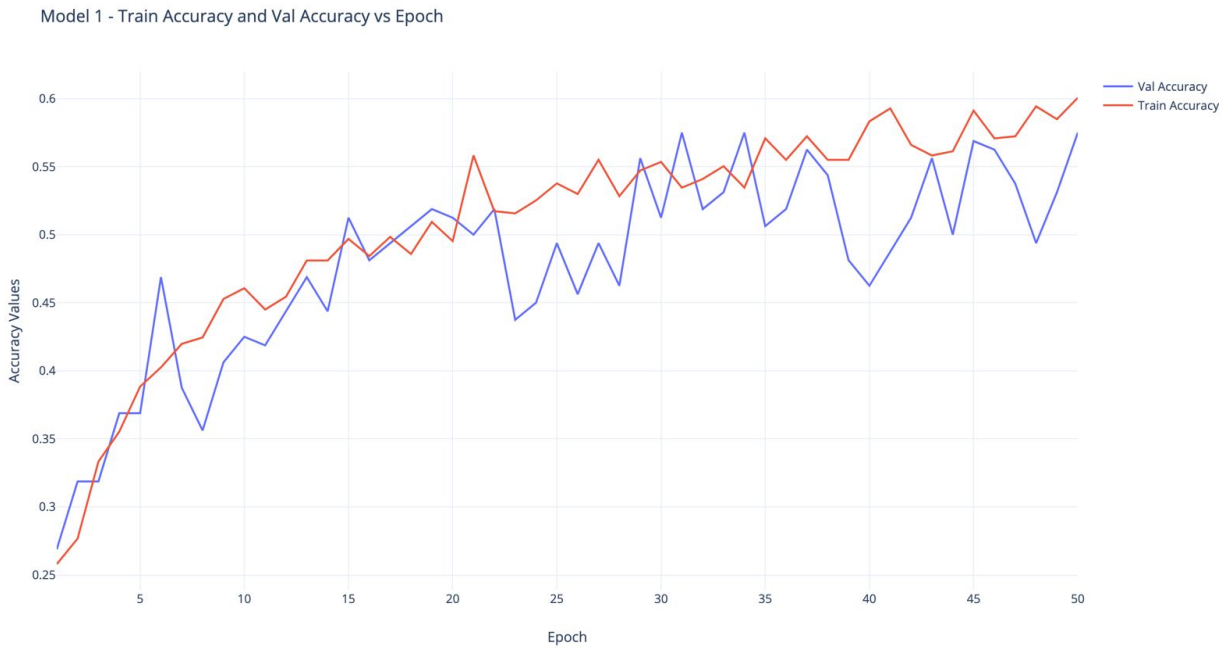
Validation Accuracy vs epoch:



Inferences from the graph:

- The initial rise in accuracy is steep, reaching 0.5 by epoch 10. This suggests rapid learning in the early stages of training.
- By achieving a final accuracy of 0.6, the model demonstrates successful prediction of the target task, suggesting strong potential for accurate predictions on unseen data

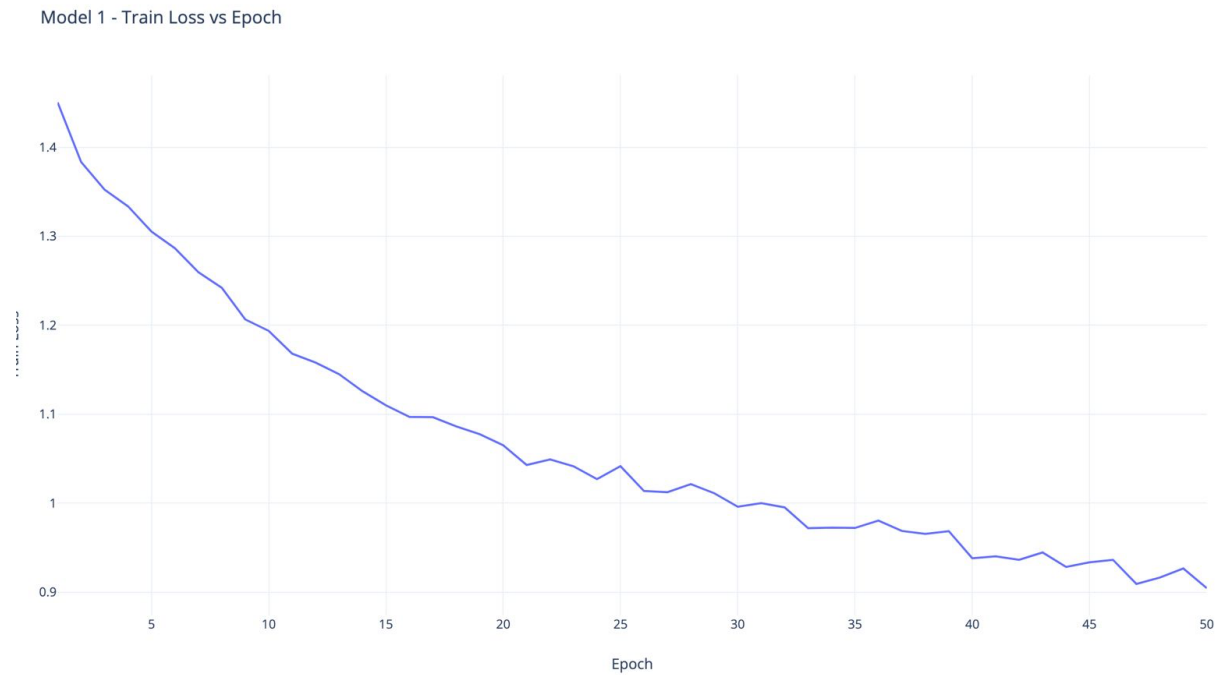
Training and Validation Accuracy VS Epoch :



Inferences from the graph:

- The model exhibits a significant increase in validation accuracy, starting from around 0.3 and reaching nearly 0.6 by epoch 50. This suggests effective learning from the training data.
- The accuracy curve converges towards a stable value around 0.6, indicating to have minimal overfitting on the training data.
- The absence of sharp fluctuations throughout the training process implies the model is steady and has controlled learning.

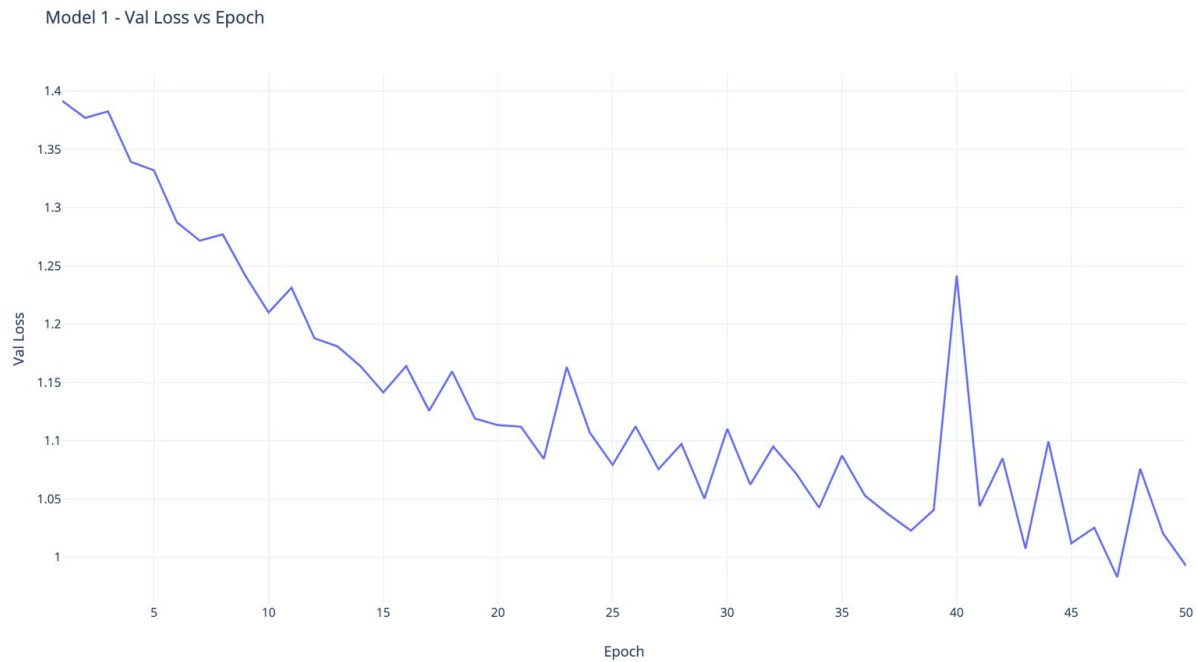
Training Loss Vs Epoch:



Inferences from the graph:

- The initial drop in loss is steep, reaching around 1.2 by epoch 5 which highlights rapid optimization in the early stages of training.
- The rate of decrease gradually slows down after epoch 5, reaching a stable value of around 0.8 by epoch 40. This could be due to diminishing returns with further training time or approaching the limits of the model's capacity.

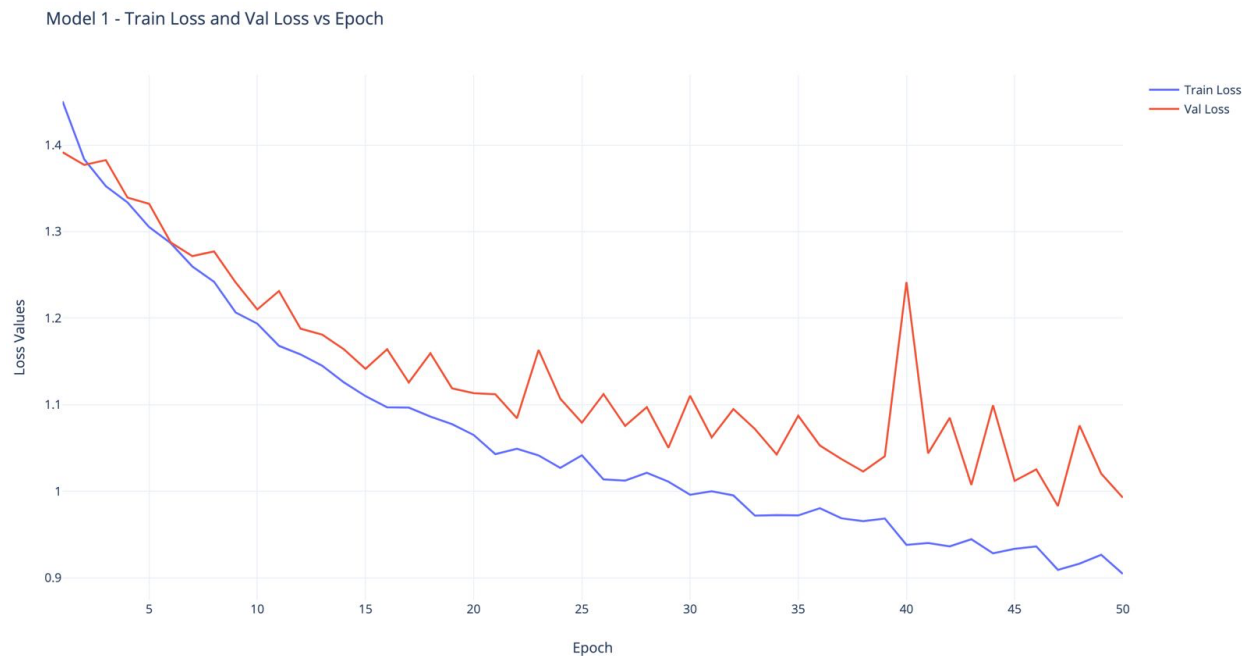
Validation Loss Vs Epoch:



Inferences from the graph:

- Since it has reached a value loss of 0.8 by epoch 45 signifies successful error minimization, potentially translating to good performance on unseen data.
- The low final value loss suggests successful minimization of the model's value metric errors and potential for good performance on unseen data regarding that metric

Training and Validation Loss VS Epoch :



Inferences from the graph:

- The initial drop in loss is steep, highlighting rapid optimization in the early stages of training.
- The rate of decrease gradually slows down later, potentially indicating diminishing returns with further training time.

Performance review:

Test loss: 1.0140

Test Accuracy: 0.5512

The model reaches an accuracy of around 56%. This is inferior as compared to Bot-1.

It will, 56% of the time, take the same steps as the Bot-1. Hence, a bot following the steps predicted by the model will make the steps as taken by the Bot-1 56% of the time.

MODEL - 2

Problem statement:

Considering the bot's current position and the board state, this model estimates the probability of the bot saving the crew.

Input Space:

The information available for the predictive model includes essential elements crucial for decision-making:

- a) *Ship Layout*: Represents the ship's structure
- b) *Probability Maps*: Two 50x50 maps are flattened to 2500-length arrays, capturing alien and crew probabilities in the ship.
- c) *Bot Position*: Indicates the bot's current location within the ship.
- d) *Specific BOT movement*: The BOT movement at each step

Data Preprocessing:

We have used the same data processing strategy as used for model 1 as follows:

Before training the model, the flattened probability maps and ship undergo preprocessing steps to transform the data into a dataset with 51 columns and around 30,000 rows. These steps likely involve normalization, feature extraction, or dimensionality reduction techniques.

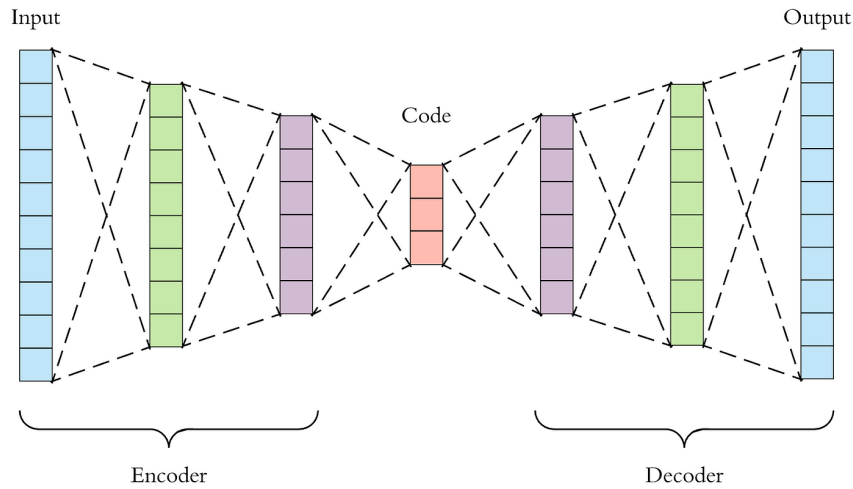
Subsequently, a normalization process is applied to the data using NumPy's PowerTransformer. This method normalizes each feature separately, aiming to approximate a Gaussian-like distribution. The choice of the Yeo-Johnson transformation method is deliberate, as it accommodates input values that may include zeros, ensuring robust handling of such cases during the normalization process.

$$\psi(\lambda, y) = \begin{cases} ((y + 1)^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y + 1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y + 1)^{2-\lambda} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y + 1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Next, we use a MinMax scaler to scale the data into an appropriate range of 0 to 1.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Here too we are using auto-encoder, a specialized neural network architecture for dimensionality reduction purposes. We have two autoencoders, one for reducing the alien probability map dimensions and another for crew probability map dimensions. For the autoencoder, we are using the neural network architecture implemented for bot 1, but here the change is we have 5 hidden layers with 256,64,25,64,256 neurons. Here for the autoencoder, the input and output are the same so that the network learns to reduce the dimensions of data as well as replicate the actual data from the reduced encoding. Thus after training it for 25 epochs with a learning rate of 0.1 using the Gradient Descent Algorithm, we take the data as input for prediction but get the 3rd hidden layer (i.e. the layer with 25 neurons) representation as our reduced data

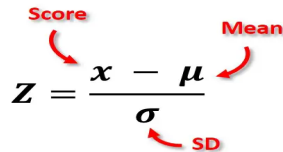


This method compresses the complex information within the crew and alien probability maps into a more concise latent representation, effectively reducing dimensions to 25. This dimensionality reduction is pivotal for streamlining the dataset while retaining essential information, facilitating more efficient processing and in-depth analysis.

Following this transformation, the NaN values within the dataset are removed. These are done to ensure the integrity and reliability of the dataset for subsequent analytical procedures. This comprehensive NaN value elimination process ensures a consistent and accurate dataset representation, ensuring robustness in subsequent analyses.

Subsequently, a z-score transformation is systematically applied to standardize the dataset. This statistical normalization technique involves centering the mean of the dataset at 0 and scaling its

standard deviation to 1. By normalizing the data in this manner, uniformity in scale is achieved across all variables, enabling equitable comparisons between diverse features. This crucial step prepares the dataset for rigorous analyses and modeling techniques, ensuring unbiased contributions from each feature irrespective of their original scales. Ultimately, this meticulous process enhances the dataset's suitability for subsequent comprehensive analyses and model training.

$$Z = \frac{x - \mu}{\sigma}$$


Output Space:

The output of the Model 2 is the probability that the move being taken is the best move from the given possible set of moves to successfully save the crew i.e. the output provides the probability of the bot rescuing the crew with the given input state.

Model Space:

To test the efficiency of the bot in rescuing crew members, we have tested it with various models. Each model brings unique strengths, but they also have their limitations that might affect their performance.

We gathered data on the Bot from various simulations, data such as positions of aliens, crew members, bot movements, and contextual knowledge. This data amalgamation is transformed into a structured format conducive to mathematical analysis, enabling the models to interpret it effectively.

Our primary objective revolves around estimating the probability of the bot saving the crew and this probability scale ranges from 0 to 1, denoting 0 for failure and 1 for success, respectively.

For the models, we have considered Logistic Regression, Decision Trees, Random Forests, Support Vector Machines, and Neural Networks. Each model exhibits distinct advantages and drawbacks; for instance, Logistic Regression might struggle with intricate relationships, Decision Trees might overfit, Random Forest might lack in capturing nuanced patterns, SVM might falter with large datasets, and **Neural Networks provide you a more reliable prediction model to capture the patterns.**

In assessing model performance, we have also used metrics like log loss or mean squared error to serve as indicators, quantifying the disparities between predicted and actual outcomes. But since

the output layer has only one neuron, we have decided to use Binary cross-entropy for the loss entropy.

Considering all the metrics and the efficiency of the model prediction, we have seen that Neural Network has performed the best. Hence, we have used a Neural network for the MODEL - 2

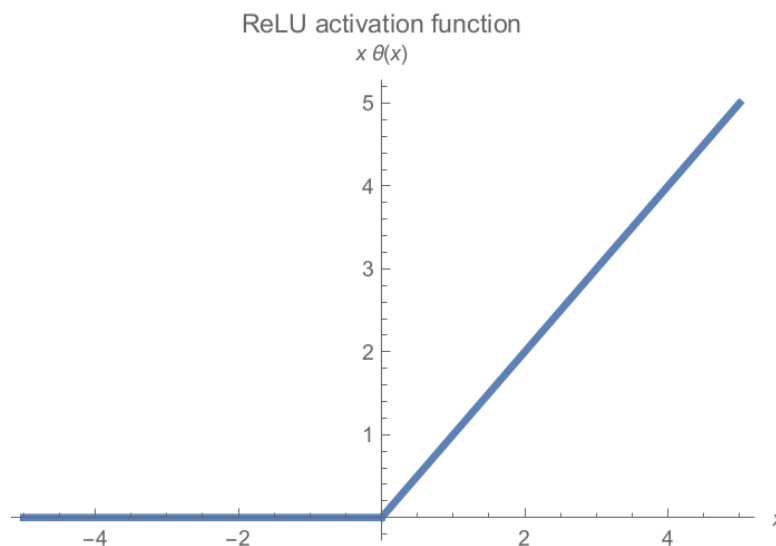
Neural Network:

The neural network we used comprises three hidden layers with dimensions: 16, 8, and 4 respectively. The input layer aligns with the size of the data input, while the output layer size is set to 1, accommodating the probability scale ranges from 0 to 1, denoting the chances of crew being rescued if we take the given step in a given state.

ReLU(Rectified Linear Activation Function):

Activation functions used in the hidden layers are ReLU (Rectified Linear Activation Function). ReLU functions as a linear activation method that maintains the input as the output if it is positive; otherwise, it outputs zero. This activation scheme is applied across all hidden layers, facilitating the extraction of nonlinear relationships and patterns within the data.

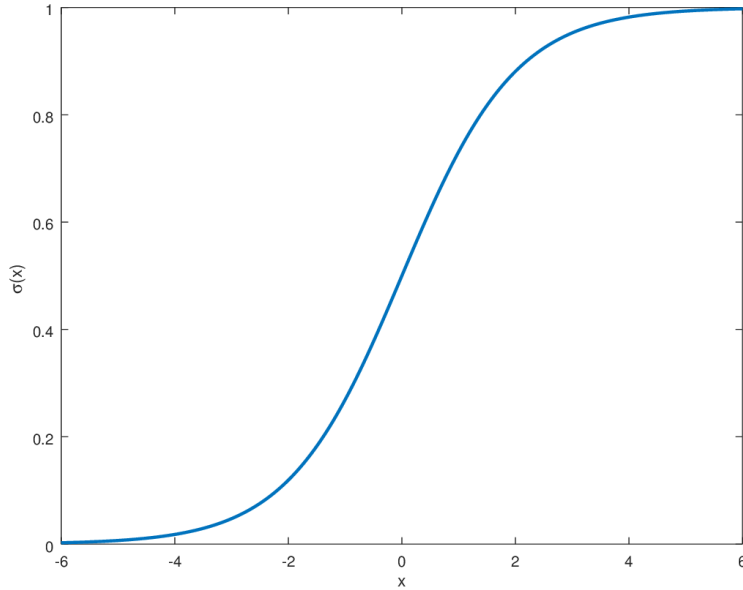
$$f(x) = \max(0, x)$$



Output Layer: Sigmoid

The Sigmoid function's characteristic S-shaped curve adds non-linearity to our model, enabling the neural network to comprehend intricate decision boundaries. Its pivotal advantage lies in its range from 0 to 1, aligning perfectly with our need to predict probabilities within that specific range. Employing a threshold of 0.5 on the output grants us both the predicted probability and a definitive output label, aiding in determining whether the outcome leans towards saved or not saved.

$$S(x) = \frac{1}{1 + e^{-x}}$$



Loss Function:

Since the output layer has only one neuron, we have decided to use Binary cross-entropy for the loss entropy.

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^N - (y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Binary Cross-Entropy is a pivotal loss function used in binary classification problems, primarily when dealing with binary output spaces. This specific function proficiently measures the divergence between the predicted probabilities and the actual binary output. When used in conjunction with the sigmoid activation function, it becomes crucial in calculating the probabilities associated with the two possible outcomes in a binary classification scenario.

By utilizing Binary Cross-Entropy along with sigmoid activation, the model becomes proficient at determining the likelihood of the two possible outcomes, effectively highlighting the probabilities attributed to each potential class. This mechanism plays a key role in guiding the model towards making accurate predictions, thereby enhancing its decision-making process.

After this probabilistic evaluation, the prediction mechanism focuses on identifying the output class with the highest probability among the two possible outcomes. This method effectively

identifies the most probable class for the model to predict, maximizing the accuracy and precision of its decisions within the binary classification framework.

Training Algorithm:

We are using Gradient Descent for training the algorithm. The gradient descent function tries to converge to a local minima of any differential function. In any ML model, we try to minimize our cost function.

$$\begin{array}{c} \text{Old weight} \quad \text{Derivative of Error} \\ \text{with respect to weight} \\ \downarrow \quad \downarrow \\ *W_x = W_x - \alpha \left(\frac{\partial \text{Error}}{\partial W_x} \right) \\ \uparrow \quad \uparrow \\ \text{New weight} \quad \text{Learning rate} \end{array}$$

Gradient Descent Formula

In our model's architecture, the computation of gradients involves applying the derivative of the Rectified Linear Unit (ReLU) function to determine the gradient of the loss concerning the inputs of neurons activated by ReLU. This derivative calculation is pivotal during the backward pass, contributing to the model's understanding of how changes in these inputs affect the overall loss.

Following this gradient computation, the model's iterative learning process involves updating both weights and biases based on these calculated gradients. This update mechanism, stemming from the derivative of the ReLU function, influences adjustments to the model's parameters during training. The weight and bias updates are instrumental in refining the model's predictive capabilities, iteratively improving its ability to capture intricate patterns and relationships within the data.

Overall, this process of utilizing the ReLU derivative for gradient computation and subsequent parameter updates plays a crucial role in the model's learning journey. It aids in fine-tuning the model's parameters to optimize its predictive accuracy and enhance its capacity to generalize patterns from the provided data.

Learning Rate: 0.01

Batch Size: 128

Epochs: 35

Avoiding success being overrepresented in the training:

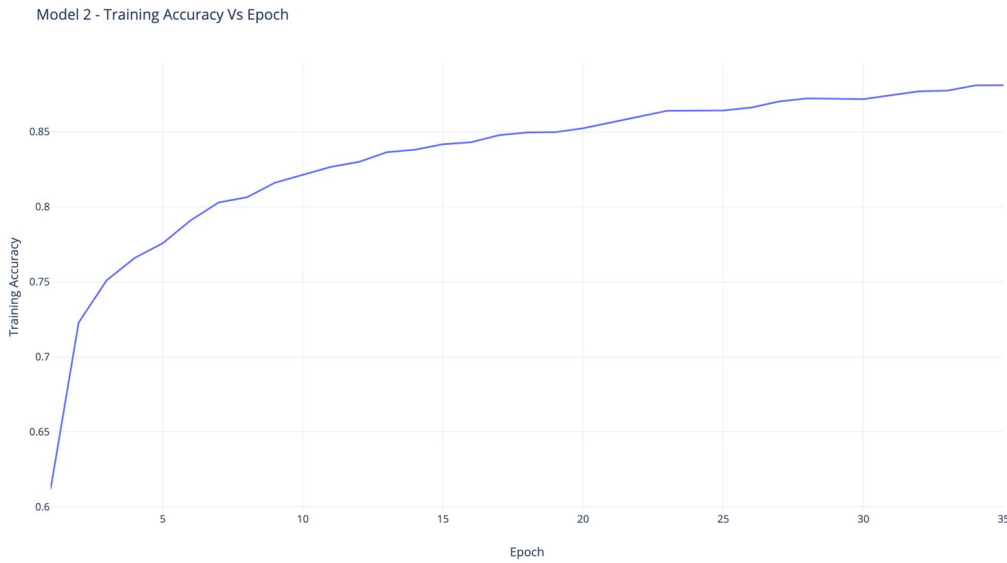
Acknowledging Bot-1's exceptional success in rescuing the crew, we take proactive steps to prevent its success from dominating the model's training. To counterbalance this potential bias, we implemented the technique of Random Oversampling within our training data methodology. This method strategically addresses the imbalance between successful and unsuccessful outcomes by artificially augmenting the representation of the minority class, which in this case corresponds to instances where the rescue mission fails.

Random Oversampling is a data augmentation technique that addresses class imbalance by randomly duplicating data points from the underrepresented class and incorporating them into the training dataset. Specifically, it selectively amplifies instances where the crew is not saved, ensuring a more equitable representation of both successful and unsuccessful outcomes in our training data. By artificially increasing the occurrences of unsuccessful outcomes, the model gains a more balanced understanding of these scenarios, thereby reducing the risk of overemphasizing Bot-1's remarkable success in the training process. This approach fosters a more unbiased and comprehensive learning experience for the model, allowing it to better navigate and predict outcomes across a spectrum of success and failure scenarios.

Training Results:

Epoch	Train Acc	Train Loss	Val Acc	Val Loss
1	0.727	0.612	0.588	0.643
2	0.540	0.723	0.501	0.748
3	0.469	0.751	0.447	0.758
4	0.429	0.766	0.409	0.759
5	0.405	0.776	0.387	0.784
6	0.384	0.791	0.381	0.791
7	0.370	0.803	0.367	0.809
8	0.358	0.806	0.343	0.807
9	0.349	0.816	0.338	0.818
10	0.340	0.821	0.324	0.824
11	0.334	0.827	0.318	0.833
12	0.327	0.830	0.333	0.808
13	0.322	0.836	0.306	0.843
14	0.319	0.838	0.303	0.845
15	0.312	0.842	0.290	0.850
16	0.308	0.843	0.286	0.854
17	0.304	0.848	0.283	0.862
18	0.300	0.850	0.280	0.856
19	0.296	0.850	0.290	0.850
20	0.293	0.852	0.299	0.846
21	0.289	0.856	0.275	0.862
22	0.283	0.860	0.270	0.864
23	0.280	0.864	0.279	0.858
24	0.275	0.864	0.279	0.856
25	0.273	0.864	0.254	0.874
26	0.272	0.866	0.263	0.865
27	0.267	0.870	0.254	0.879
28	0.263	0.872	0.247	0.887
29	0.261	0.872	0.247	0.881
30	0.259	0.872	0.245	0.883
31	0.256	0.874	0.238	0.888
32	0.254	0.877	0.238	0.887
33	0.252	0.877	0.243	0.879
34	0.249	0.881	0.244	0.879
35	0.247	0.881	0.239	0.884

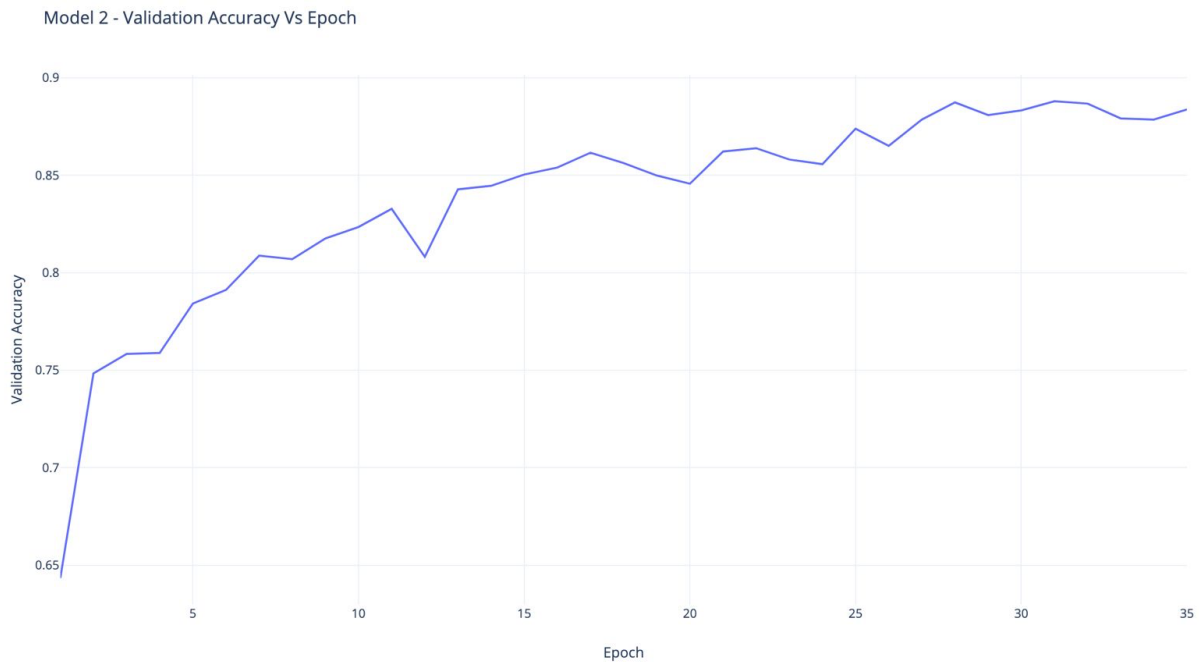
Training Accuracy vs Epoch:



Inferences from the graph:

- The model exhibits a substantial increase in training accuracy, starting from around 0.62 and reaching nearly 0.88 by epoch 25 which suggests effective learning from the training data.
- The accuracy curve steadily climbs and then plateaus around epoch 25, indicating the model approaching its optimal performance on the training data.

Validation Accuracy vs epoch:



Inferences from the graph:

- The initial rise in accuracy is gradual, reaching around 0.4 by epoch 10. This suggests a more measured learning process compared to the rapid initial progress sometimes observed in training accuracy curves.

Training and Validation Accuracy VS Epoch :

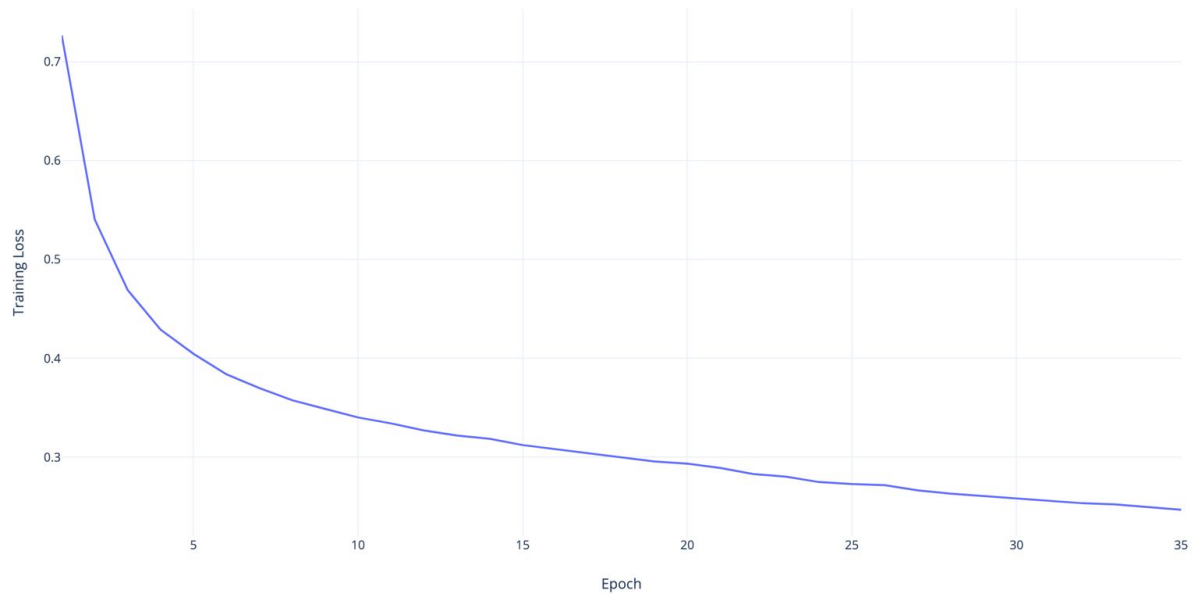


Inferences from the graph:

- Both curves exhibit a significant increase in accuracy, suggesting the model effectively learns from the training data. Training accuracy reaches nearly 0.8 by epoch 25, while validation accuracy steadily climbs to around 0.65 by epoch 50.
- The validation accuracy consistently lags behind the training accuracy, suggesting the model is learning generalizable patterns rather than simply memorizing the training data.

Training Loss Vs Epoch:

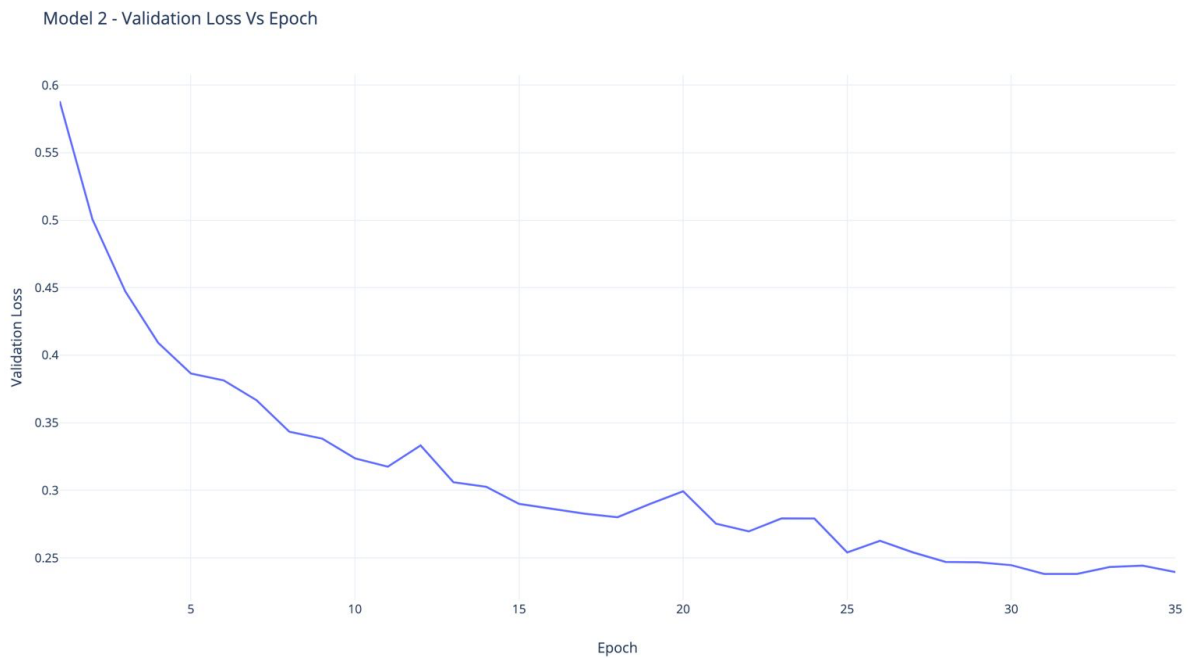
Model 2 - Training Loss Vs Epoch



Inferences from the graph:

- The training loss curve exhibits a consistent decrease throughout training, suggesting successful parameter optimization to minimize errors. Reaching a final loss of 0.09 by epoch 50 indicates a significant error reduction.
- The training loss curve shows a gradual decline until epoch 10, followed by a steeper decrease until epoch 20. This suggests a measured learning process initially, followed by more efficient optimization later on.

Validation Loss Vs Epoch:

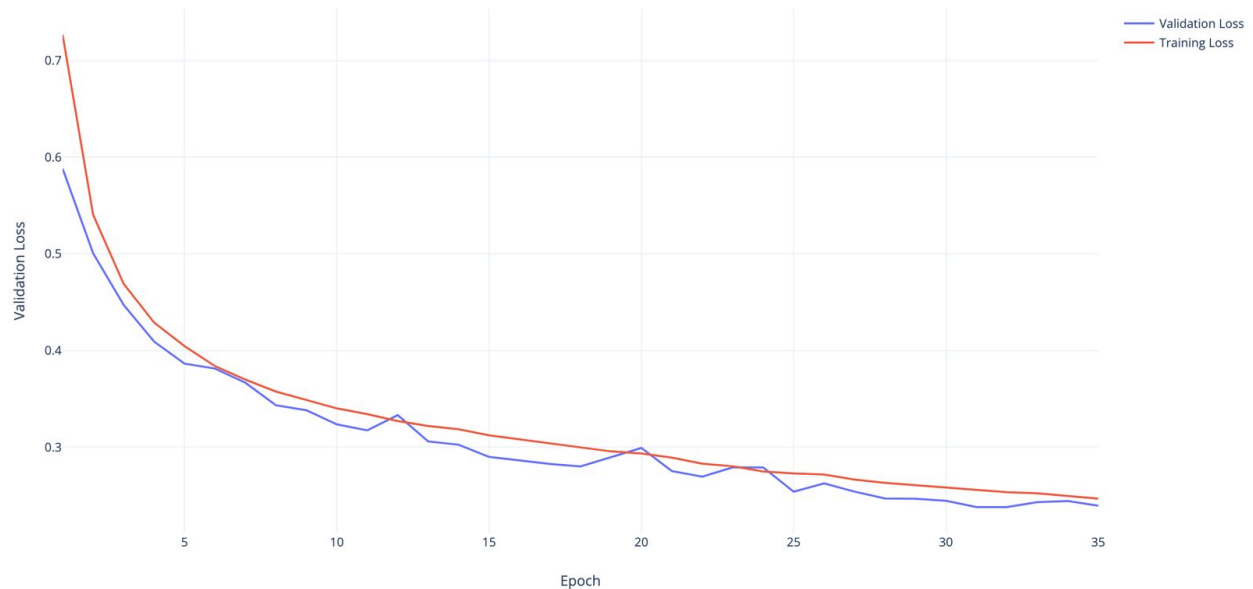


Inferences from the graph:

- The dip in validation loss around epoch 15 could be due to stochasticity in the training process or encountering more challenging examples in the data.
- The final validation loss value will indicate how well the model minimizes errors on unseen data, potentially translating to good performance in your specific research application.

Training and Validation Loss VS Epoch :

Model 2 - Validation Loss and Training Loss Vs Epoch



Inferences from the graph:

- Both curves exhibit a consistent decrease in loss, suggesting successful parameter optimization to minimize errors. Training loss reaches around 0.1 by epoch 25, while validation loss reaches around 0.2 by epoch 50.
- The absence of sharp fluctuations throughout the training process implies stable and consistent learning.

Performance review:

Test loss: 0.2406

Test Accuracy: 0.8786

While the model achieves an accuracy of approximately 88%, it falls short of Bot-1's performance. This translates to correctly predicting the outcome 88% of the time, meaning a bot following its recommendations would mimic Bot-1's actions with the same frequency.

Model 3:

Problem Statement:

The aim is to develop an Actor-Critic model for this semi-supervised learning task. The task is not explicitly defined, but it involves an agent (the "Actor") taking actions in a certain state to achieve a goal. The "Critic" evaluates the actions taken by the Actor and provides feedback, which is used to improve the Actor's performance. But this is a semi-supervised learning task because we are using the labeled data from the bot's actual performance to start the actor-critic loop by first training the critic model using that labelled data.

The specific goals are:

1. Train the Actor model to take actions based on the current state.
2. Train the Critic model to evaluate the Actor's actions and predict the expected reward.
3. Improve the Actor's performance by using the feedback from the Critic.
4. The problem involves iterating over these steps to continuously improve the Actor's actions based on the Critic's evaluations.
5. The ultimate aim is to have the Actor model perform actions that maximize the expected reward.

The model is composed of two main components: the Actor and the Critic.

- **Critic:**
 - The Critic evaluates the actions taken by the Actor by estimating the probability that the actor succeeds in finally rescuing the crew i.e. Critic evaluates the expected return of those actions.
 - The Critic guides the Actor by providing feedback on the quality of the actions chosen, helping the Actor improve its policy by learning which actions are better in different states.
 - The Critic, typically implemented as a value function approximator using methods like temporal difference learning, takes the state and action as input and predicts the expected final success probability (value) for that action in that state.
 - By comparing the predicted value with the actual observed success, the Critic learns to approximate the performance prediction function more accurately, which in turn helps the Actor improve its policy.
 - **The architecture of the Critic model is the same as that used in Model 2.**

- **Actor:**
 - It learns a mapping from observed states to actions, which is known as the policy.
 - The Actor's primary goal is to learn the optimal action for each state to maximize the critic performance probability in that state
 - It is responsible for deciding the actions to take in an environment based on the learned policy.
 - The Actor aims to improve its policy using feedback from the critics about the value of the chosen actions.
 - **The architecture of the Actor model is the same as that used in Model 1.**

Input Space:

The information available for both the models includes essential elements crucial for decision-making:

- a) *Ship Layout*: Represents the ship's structure.
- b) *Probability Maps*: Two 50x50 maps are flattened to 2500-length arrays, capturing alien and crew probabilities in the ship.
- c) *Bot Position*: Indicates the bot's current location within the ship.

We also provide additional data as follows:

For Actor Model:

All possible BOT moves: Defines allowable bot movements in the current state.

For Critic Model:

Specific BOT movement: The BOT movement at each step by the Actor model.

Data Preprocessing:

Before training the model, the flattened probability maps and ship undergo preprocessing steps to transform the data into a dataset with 49 columns and around 30,000 rows. These steps likely involve normalization, feature extraction, or dimensionality reduction techniques.

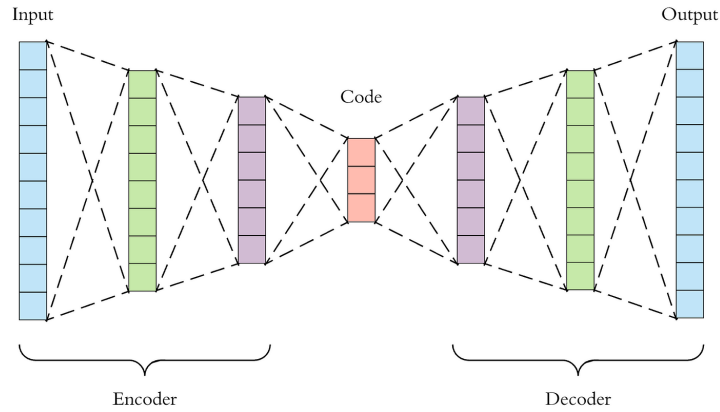
Subsequently, a normalization process is applied to the data using NumPy's PowerTransformer. This method normalizes each feature separately, aiming to approximate a Gaussian-like distribution. The choice of the Yeo-Johnson transformation method is deliberate, as it accommodates input values that may include zeros, ensuring robust handling of such cases during the normalization process.

$$\psi(\lambda, y) = \begin{cases} ((y + 1)^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y + 1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y + 1)^{2-\lambda} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y + 1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

Next, we use a MinMax scaler to scale the data into an appropriate range of 0 to 1.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

The same types of AutoEncoders as used in the previous models are used but we have implemented the AutoEncoders here using Keras Library. We have two autoencoders, one for reducing the alien probability map dimensions and another for crew probability map dimensions. The same architecture with 5 hidden layers with 256,64,25,64,256 neurons is used. For training it for 25 epochs we use a learning rate of 0.1 but here the change is we use Adam Optimiser from Keras Library,



Following this transformation, the NaN values within the dataset are removed. These are done to ensure the integrity and reliability of the dataset for subsequent analytical procedures. This comprehensive NaN value elimination process ensures a consistent and accurate dataset representation, ensuring robustness in subsequent analyses.

Subsequently, a z-score transformation is systematically applied to standardize the dataset. This statistical normalization technique involves centering the mean of the dataset at 0 and scaling its standard deviation to 1.

$$Z = \frac{x - \mu}{\sigma}$$

Score Mean
SD

Output Space:

- **Actor model:** The model produces an output represented as a one-hot encoded vector, outlining four distinct classes linked to different directional movements within the context. These classes signify specific actions: left, right, up, and down, identified numerically as (0, 1, 2, and 3) respectively. This encoding method facilitates categorical representation, empowering the model to identify the most likely directional movement for the bot based on the input state.
- **Critic model:** The output of the critic model is the probability that the move being taken is the best move from the given possible set of moves to successfully save the crew i.e. the output provides the probability of the bot rescuing the crew with the given input state.

Model Space:

We are using Keras for the Model:

What is Keras:

Keras is a high-level, open-source neural network API written in Python, designed to simplify and accelerate deep learning development. It sits on top of other low-level libraries like TensorFlow, PyTorch, and MXNet, providing a user-friendly interface for building and training deep learning models.

Why Keras:

Keras boasts a concise and intuitive API, allowing you to focus on model design and problem-solving rather than intricate code details. It also supports a wide range of neural network architectures, including convolutional neural networks (CNNs) for image recognition, recurrent neural networks (RNNs) for sequences like text and audio, and generative models for creative tasks.

Neural Network:

There are two Neural networks as explained earlier: One for the Actor and another for Critic.

For Actor:

The neural network comprises four hidden layers with neurons: 32, 16, 8, and 4 respectively. The input layer aligns with the size of the data input, while the output layer size is set to 4, accommodating the directional movement classes.

For Critic:

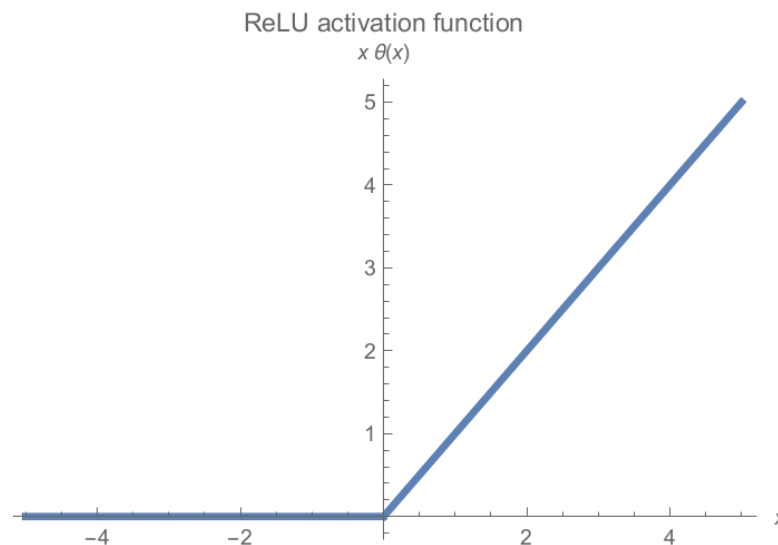
The neural network comprises three hidden layers with neurons: 16, 8, and 4 respectively. The input layer aligns with the size of the data input, while the output layer size is set to 1, accommodating the probability scale ranges from 0 to 1, denoting the chances of crew being rescued if we take the given step in a given state.

We use the ReLU activation function for both Actor and Critic.

ReLU(Rectified Linear Activation Function):

Activation functions used in the hidden layers are ReLU (Rectified Linear Activation Function). ReLU functions as a linear activation method that maintains the input as the output if it is positive; otherwise, it outputs zero. This activation scheme is applied across all hidden layers, facilitating the extraction of nonlinear relationships and patterns within the data.

$$f(x) = \max(0, x)$$



Actor Output Layer: Softmax

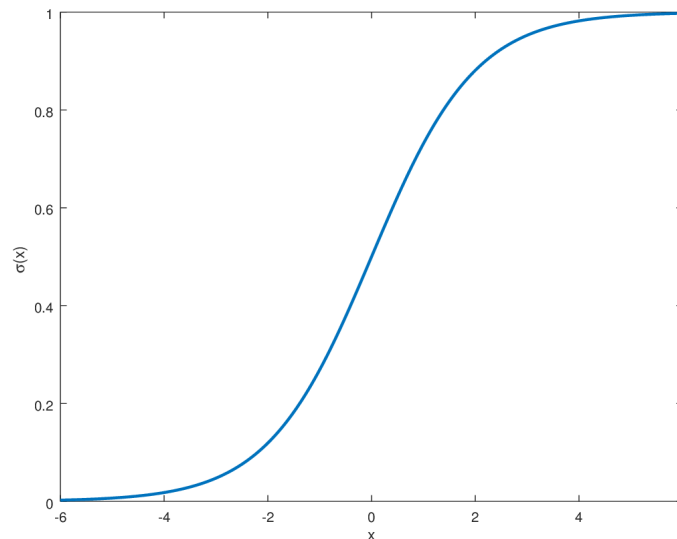
The Softmax activation function reigns supreme as a probabilistic output layer. Its significance lies in its ability to transform the network's raw, uncalibrated class scores into a set of well-defined probability distributions. This formalization allows for nuanced decision-making and facilitates the quantification of confidence in each predicted class.

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K$$

Critic Output Layer: Sigmoid

The Sigmoid function's characteristic S-shaped curve adds non-linearity to our model, enabling the neural network to comprehend intricate decision boundaries. Its pivotal advantage lies in its range from 0 to 1, aligning perfectly with our need to predict probabilities within that specific range. Employing a threshold of 0.5 on the output grants us both the predicted probability and a definitive output label, aiding in determining whether the outcome leans towards saved or not saved.

$$S(x) = \frac{1}{1 + e^{-x}}$$



Actor Loss Functions:

Categorical cross-entropy is being used as the loss function.

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Categorical Cross-Entropy stands as a fundamental loss function widely utilized in multi-class classification scenarios, primarily when dealing with categorical output spaces. This particular function effectively quantifies the dissimilarity between the predicted probabilities and the actual categorical output. When employed alongside the masked softmax activation function, it becomes instrumental in calculating the probabilities associated with the bot's potential valid moves within the given context.

By leveraging Categorical Cross-Entropy coupled with masked softmax activation, the model is adept at discerning the likelihood of the bot's valid moves, essentially highlighting the probabilities attributed to each potential action. This mechanism serves as a critical facet in steering the model towards prioritizing valid movements, thereby refining its decision-making process.

Post this probabilistic evaluation, the prediction mechanism delves into discerning the output class exhibiting the highest probability among the potential valid moves. This approach effectively pinpoints the most probable action for the bot to undertake, maximizing the accuracy and precision of its decisions within the multi-class classification framework.

Critic Loss Function:

Since the output layer has only one neuron, we have decided to use Binary cross-entropy for the loss entropy.

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^N - (y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Binary Cross-Entropy is a pivotal loss function used in binary classification problems, primarily when dealing with binary output spaces. This specific function proficiently measures the divergence between the predicted probabilities and the actual binary output. When used in conjunction with the sigmoid activation function, it becomes crucial in calculating the probabilities associated with the two possible outcomes in a binary classification scenario.

By utilizing Binary Cross-Entropy along with sigmoid activation, the model becomes proficient at determining the likelihood of the two possible outcomes, effectively highlighting the

probabilities attributed to each potential class. This mechanism plays a key role in guiding the model towards making accurate predictions, thereby enhancing its decision-making process.

After this probabilistic evaluation, the prediction mechanism focuses on identifying the output class with the highest probability among the two possible outcomes. This method effectively identifies the most probable class for the model to predict, maximizing the accuracy and precision of its decisions within the binary classification framework.

Training algorithm: Adam

Adam, short for Adaptive Moment Estimation, is a popular optimization algorithm used in deep learning models. It is known for its efficiency and low memory requirements.

Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It uses moving averages of the parameters (momentum) to navigate the parameter space effectively.

The algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters alpha and beta control the decay rates of these moving averages.

The initial learning rate is adapted based on the average of the recent step sizes, which results in an automatic scheduling of the learning rate. This makes Adam a very effective algorithm for deep and complex neural networks.

Mathematically,

Loading...

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t - \epsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t} \right]$$

Where,

$$v_t = \beta v_{t-1} + (1 - \beta) * \left[\frac{\delta L}{\delta w_t} \right]^2$$

Dropout:

Dropout, as a regularization technique, can have a significant impact on the Actor-Critic model during its learning. When Dropout is applied to the Actor-Critic model, it randomly deactivates a fraction of neurons in the network during each training update. This means that during each training step, a slightly different network is used. This helps to prevent the model from relying too heavily on any single neuron and therefore reduces overfitting.

The impact of Dropout on the Actor-Critic model can be summarized as follows:

- Improved Generalization: By preventing overfitting, Dropout helps the model to generalize better to unseen states in the environment, leading to more robust policies.
- Increased Exploration: The randomness introduced by Dropout can encourage the model to explore the environment more thoroughly, potentially leading to better long-term policies.
- Reduced Overfitting: Overfitting is a common problem in deep learning models where the model performs well on the training data but poorly on unseen data. Dropout helps to mitigate this issue in the Actor-Critic model.
- Stabilized Training: Dropout can also help to stabilize the training of the Actor-Critic model by reducing the variance of the updates.

For each loop of Actor training:

Learning rate: 0.001

Batch Size: 32

Epochs: 50

For each loop of Critic Training:

Learning rate: 0.05

Batch Size: 32

Epochs: 50

Avoiding success being overrepresented in the training:

Acknowledging Bot-1's exceptional success in rescuing the crew, we take proactive steps to prevent its success from dominating the model's training. To counterbalance this potential bias, we implemented the technique of Random Oversampling within our training data methodology. This method strategically addresses the imbalance between successful and unsuccessful outcomes by artificially augmenting the representation of the minority class, which in this case corresponds to instances where the rescue mission fails.

Random Oversampling is a data augmentation technique that addresses class imbalance by randomly duplicating data points from the underrepresented class and incorporating them into the training dataset. Specifically, it selectively amplifies instances where the crew is not saved, ensuring a more equitable representation of both successful and unsuccessful outcomes in our training data. By artificially increasing the occurrences of unsuccessful outcomes, the model gains a more balanced understanding of these scenarios, thereby reducing the risk of overemphasizing Bot-1's remarkable success in the training process. This approach fosters a more unbiased and comprehensive learning experience for the model, allowing it to better navigate and predict outcomes across a spectrum of success and failure scenarios.

Methodology:

Initially, as this is a semi-supervised learning algorithm we take the available simulation data from bot1 of project 2 and form a dataset with an alien probability map, crew probability map, bot position, ship layout, and all possible moves in that state, preprocess these as mentioned above to get the final reduced input and train the actor model based on this

To generate the train data for the Critic we take the fixed ship layout, and instantiate a state by placing the bot, alien, and crew. Now we give this as input to the Actor model to get the next move for the bot, we perform probability updates based on the move predicted by the model and generate the next state input to the Actor model and we repeat this to generate all the data for the critic. Now here while generating the critic train data we also have another column in the data that tells whether in that simulation we saved the crew within 150 steps .i.e while generating the critic train data if the crew is saved within 150 moves (threshold) the simulation is a success and put 1 in performance column else if it exceeds 150 steps we stop the simulation we put a 0 in the performance column and re-initialize the state of the ship and then do this process again until we have enough data to train the critic model.

Now we use the state and move taken as the input and the performance column as the output and train the critic model to make it learn to predict the probability of success of a given move in a given state. Now to generate the train data for the Actor model we take the states and all possible moves in that state from the Actor Model generated data in the previous step, we create a dataset by combining each state with every possible move in that state, for example, if a state has two moves we get two rows of data with state and one of the possible move. Now this data is put into the critic model to predict the probability of success in a given state for each possible move in that state. Now we select the move that yielded the most performance prediction from the critic as the best move in that state. We create the Actor train dataset by combining the state and possible moves as input and the best move predicted by the critic as the label. Here we completed one Actor-Critic loop. Now this data is used in training the actor in the next loop. Here in every train step, a small portion of data separated beforehand is used as a test set to get test accuracy.

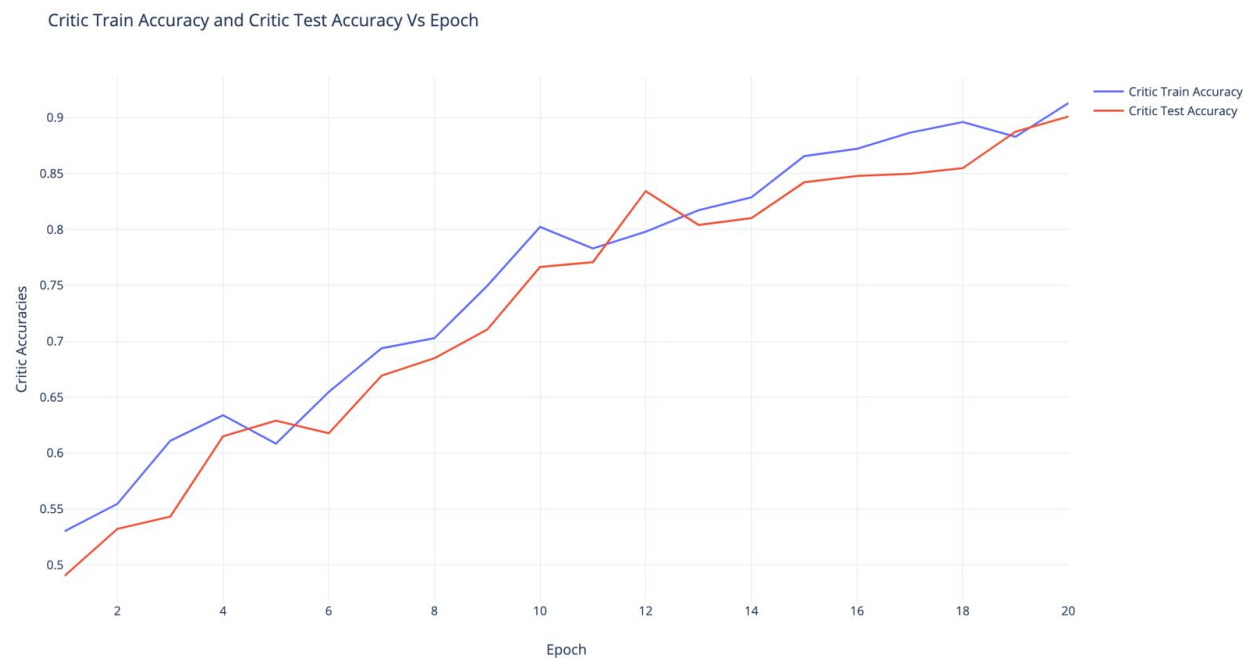
This Actor-Critic loop has been run 20 times and the results are below

Training Results:

Epoch	Critic Train Acc	Critic Train Loss	Critic Test Acc	Actor Train Acc	Actor Train Loss	Actor Test Acc
1	0.530	0.482	0.490	0.286	0.539	0.259
2	0.555	0.462	0.532	0.324	0.483	0.253
3	0.611	0.431	0.543	0.303	0.492	0.328
4	0.634	0.432	0.615	0.369	0.455	0.354
5	0.608	0.380	0.629	0.395	0.399	0.359
6	0.655	0.386	0.618	0.418	0.435	0.343
7	0.694	0.335	0.669	0.398	0.429	0.454
8	0.703	0.370	0.685	0.435	0.396	0.435
9	0.750	0.306	0.711	0.491	0.353	0.452
10	0.802	0.339	0.766	0.479	0.326	0.459
11	0.783	0.281	0.771	0.487	0.314	0.527
12	0.798	0.325	0.834	0.542	0.294	0.552
13	0.817	0.245	0.804	0.540	0.310	0.544
14	0.829	0.256	0.810	0.593	0.281	0.562
15	0.866	0.254	0.842	0.565	0.261	0.560
16	0.872	0.244	0.848	0.571	0.251	0.567
17	0.886	0.231	0.850	0.581	0.245	0.571
18	0.896	0.222	0.855	0.596	0.233	0.552
19	0.883	0.226	0.887	0.633	0.259	0.602
20	0.913	0.206	0.901	0.627	0.231	0.612

Training and Test Accuracy VS Epoch :

For Critic:

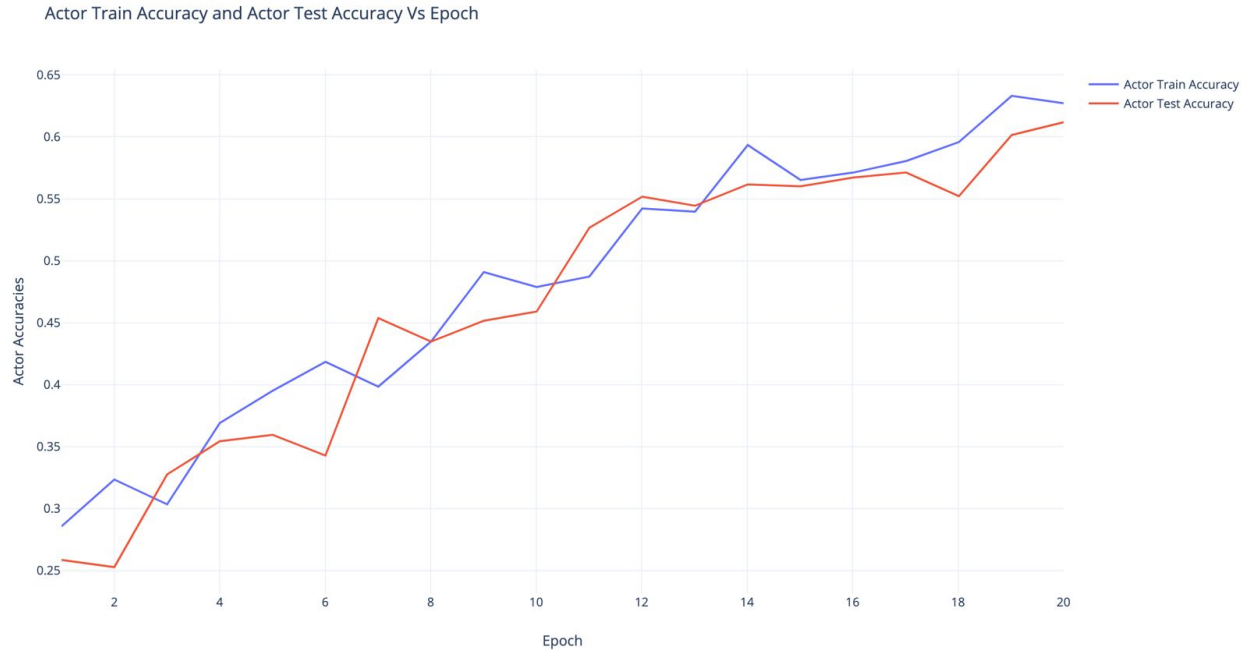


Inferences from the graph:

- Both the Critic Train Accuracy curve and the Critic Test Accuracy curve exhibit a significant increase throughout training, reaching around 0.83 for train accuracy and 0.68 for test accuracy by epoch 50. This says the critic model effectively learns from the training data and generalizes well to unseen data.

- Both curves plateau towards the end of the training, indicating the model approaches its optimal performance without substantial overfitting. This is further shown in the graph by the relatively small gap between train and test accuracy throughout training.

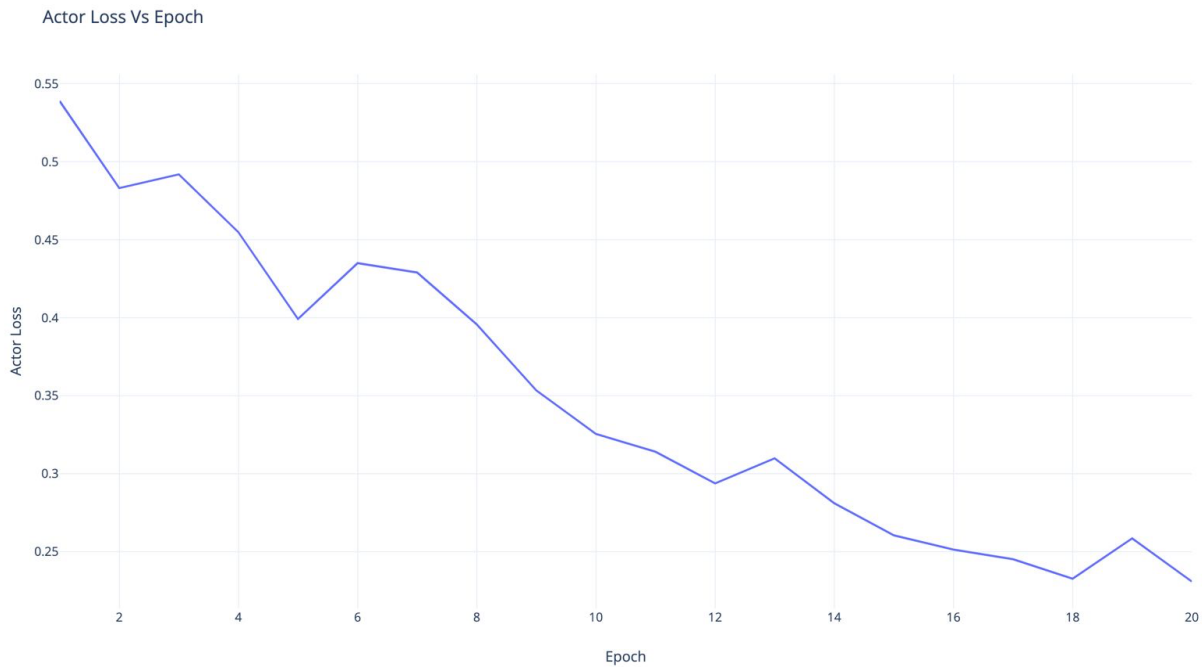
For Actor:



Inferences from the graph:

- Both Actor Train Accuracy and Actor Test Accuracy exhibit an initial steep increase in the first few epochs, suggesting rapid learning in the beginning.
- The curves continue to steadily increase throughout training, demonstrating continued learning and improvement.
- Both curves plateau around epoch 20-25, indicating convergence towards their optimal performance.
- The gap between train and test accuracy remains small, suggesting good control over overfitting while maintaining learning progress.

Actor Training Overtime VS Epoch :



Inferences from the graph:

- The blue line representing the actor loss clearly and consistently decreases throughout the training epochs. This suggests the actor-network is effectively minimizing the loss function, indicating better performance and alignment with the desired outcome.



Inferences from the graph:

- The Actor Train Accuracy curve steadily increases throughout training, reaching around 0.85 by epoch 30. This confirms that the actor is improving its ability to take effective actions.

Actor and Critic Loss VS Epoch:



Inferences from the graph:

- The decreasing loss curves for both the critic and actor indicate they are effectively minimizing their respective loss functions. This suggests they are improving their performance and aligning with the desired outcome.

Performance review:

After running the Actor-Critic loop for 20 epochs, we have observed that the Actor Model has a Test-set accuracy of 61.2% and the critic model has a Test-set accuracy of 91.3%. Here we can see from the above graphs that the actor is learning to mimic the bot's performance based on inputs from the critic and the critic is learning to provide the bot with the correct measure of performance in taking a particular move.

But we can see that the actor model has a test-set accuracy of 61.2% which means it is better than the model 1 test accuracy (55.1%) but still it only predicts the bot 1 movement with 61.2% accuracy. This means that the model takes the best move 61% of the time, so it is slightly worse performing than bot 1. Thus the actor model cannot beat the performance of bot 1.