Design, Evaluation, Explanation & Math Mapping

The model utilizes historical game data stored in a file named model evaluation.txt. Each row in this file represents a single game and contains three integers:

Human Move (human_move): Encoded as 0 (Rock), 1 (Paper), or 2 (Scissors).

Computer Move (computer_move): Encoded similarly as 0, 1, or 2.

Outcome (outcome): Encoded as 1 (Human Win), 0 (Tie), or -1 (Human Loss).

To effectively train the RNN, the categorical data (moves and outcomes) are transformed into numerical representations using one-hot encoding. This technique converts categorical variables into a binary vector format, allowing the model to process them mathematically.

Human and Computer Moves: Each move is represented as a 3-dimensional one-hot vector.

Rock: [1, 0, 0]

Paper: [0, 1, 0]

Scissors: [0, 0, 1]

Combined Input Vector (inp_vec): For each game, the human and computer moves are put into a single 6-dimensional input vector:

Target Vector (comp_vec): The target for the RNN is the move that would beat the computer's current move, effectively encouraging the model to learn strategies that maximize the human player's chances of winning.

The model employs a Recurrent Neural Network (RNN), which is particularly suited for sequential data analysis. RNNs are capable of maintaining a hidden state that captures information from previous inputs, making them ideal for tasks where context and history influence predictions. We decided to take this on as it was different from what we knew and would require creativity and research

Input Layer:

Size: 6 neurons (3 for human moves and 3 for computer moves).

Encoding: One-hot encoded vectors representing the current human and computer moves.

Hidden Layer:

Size: 50 neurons.

Activation Function: Hyperbolic tangent (tanh), which introduces non-linearity and helps in capturing complex patterns.

Output Layer:

Size: 3 neurons (representing Rock, Paper, Scissors).

Activation Function: Softmax, which converts the output logits into probability distributions, facilitating classification.

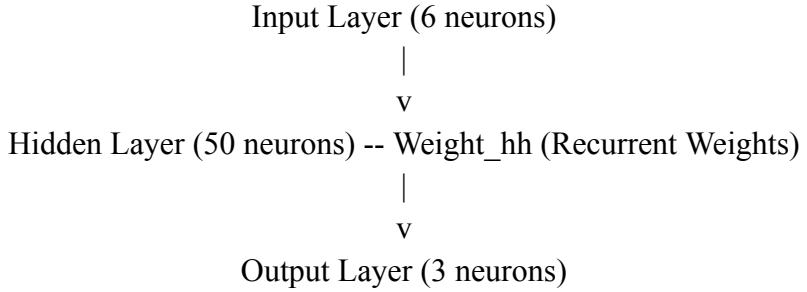
Weights and Biases:

Input-to-Hidden Weights (weight_xh): Initialized randomly with small values to break symmetry.

Hidden-to-Hidden Weights (weight_hh): Handle the recurrence in the network.

Hidden-to-Output Weights (weight_hy): Connect the hidden layer to the output layer.

Biases (bias_h, bias_y): Initialized to zeros.



The model employs a Cross-Entropy Loss function (we found too much overfitting when use MSE, so we had to revise this, which lead to the realization that we also need to use a softmax) combined with L2 Regularization to measure the discrepancy between the predicted probabilities and the actual targets while preventing overfitting.

Next, we designed the Training Loop; initially, I was spending too much time thinking about forward and back propagation for the entire network, but did not focus on the training loop itself. It all goes to waste without this as shown in the first couple attempts of playing this network

against the computer. In other words, my initial version’s training loop didn’;t fully implement the network and I was seeing barely any improvement!

Forward Propagation:

> For each epoch, the model processes the entire training dataset in sequence; it computes the hidden states and output probabilities for each input vector.

Backpropagation Through Time (BPTT):

> The model computes gradients of the loss with respect to all weights by iterating backward through the sequence. Then, Gradients are accumulated over time steps to update the weights.

Gradient Descent Updates:

> The model updates the weights and biases using the computed gradients scaled by the learning rate. It then incorporates L2 regularization into the gradients.

Validation and Early Stopping:

> After each epoch, the model evaluates its performance on a validation set. If the validation loss does not improve for a specified number of consecutive epochs (patience), training is halted to prevent overfitting.

Learning Rate Decay:

> The learning rate is reduced periodically to fine-tune the model's convergence. This number needed to be tweaked quite a bit as we implemented regularization and early stopping.

Hyperparameters (changed periodically throughout work testing, except for dimensions)

Input Dimension (input_dim): 6

Hidden Dimension (hidden_dim): 50

Output Dimension (output_dim): 3

Learning Rate (lr): 0.001

Number of Epochs (num_epochs): 500

Regularization Strength (lambda_reg): 1e-5

Patience for Early Stopping: 10 epochs

To evaluate the model's effectiveness, the following metrics are computed from the training data:

Number of Wins (wins): Instances where the human player's move beats the computer's move.

Number of Ties (ties): Instances where both players choose the same move.

Number of Losses (losses): Instances where the computer's move beats the human player's move.

Calculation Logic:

Tie: human_move == computer_move

Win: (human_move - computer_move) % 3 == 1

Loss: Otherwise

Ultimately, after tweaking parameters and implementing many things shown above to prevent overfitting, I saw an improvement to over 80% of a win rate (5:1!) There is more improvement to be done of course, and the number of epochs can become a problem with larger data sets.