Optimizing Blackjack Strategies

Clarissa Nunez, Jacob Fischbach, Jacob Klucher, James Elander & Luke Milton

Project Overview

Goal

Develop and analyze optimized strategies for playing Blackjack using machine learning algorithms.

Specifics

Use various techniques to build a model that predicts the best possible move based on the player’s hand, the dealer’s upcard, and the game state.

Limitations

Due to potential issues from complexity, the model will not split, double, or use card counting.

Technologies Used

Machine Learning

Python

Matplotlib

Keras & TensorFlow

Used to create the Actor Critic Default Program

Python

Used to create the Blackjack game

Matplotlib

Used to create plots for model predictions

The Process

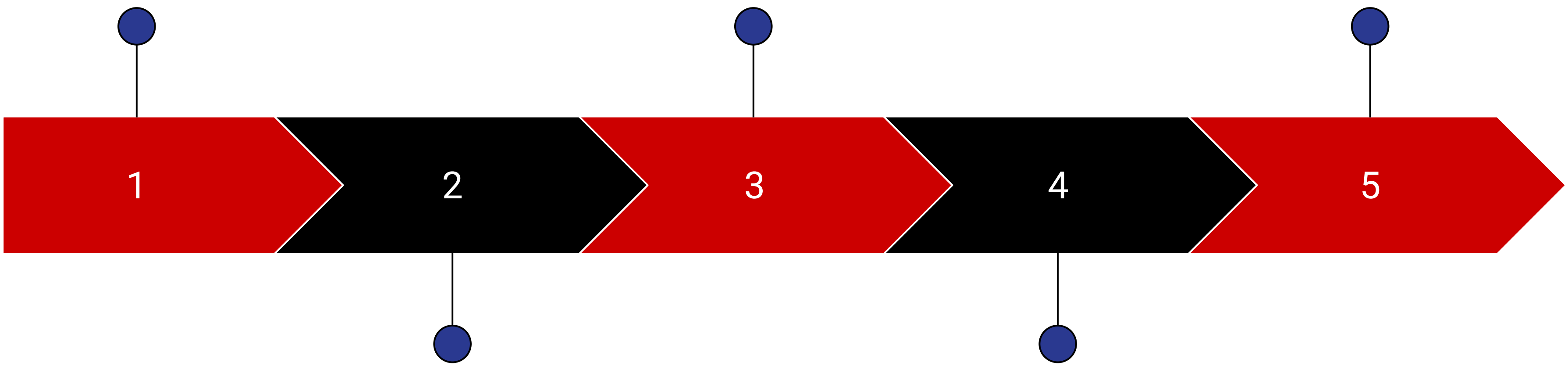
Create a Blackjack environment

Construct an Actor-Critic Reinforcement Learning Network

Use the Blackjack environment to train the Actor-Critic algorithm

Optimize the decision-making process for Blackjack players

The Actor-Critic algorithm should be able to predict the best action for the player: hit or stay



Blackjack Environment

```python
import random

# Card values (Ace can be 1 or 11)
CARD_VALUES = {
    "2": 2, "3": 3, "4": 4, "5": 5, "6": 6, "7": 7, "8": 8, "9": 9, "10": 10,
    "J": 10, "Q": 10, "K": 10, "A": 11
}

# Function to calculate hand value
def calculate_hand_value(hand):
    value = sum(CARD_VALUES[card] for card in hand)
    aces = (variable) value: int
    while value > 21 and aces:
        value -= 10  # Convert Ace from 11 to 1
        aces -= 1
    return value

# Function to deal a card
def deal_card():
    return random.choice(list(CARD_VALUES.keys()))

# Function to play a round of blackjack
def play_blackjack():
    player_hand = [deal_card(), deal_card()]
    dealer_hand = [deal_card(), deal_card()]

    print(f"Your hand: {player_hand}, Value: {calculate_hand_value(player_hand)}")
    print(f"Dealer's first card: {dealer_hand[0]}")

    # Player's turn
    while calculate_hand_value(player_hand) < 21:
        action = input("Do you want to hit or stay? (h/s): ").lower()
        if action == 'h':
            player_hand.append(deal_card())
            print(f"Your hand: {player_hand}, Value: {calculate_hand_value(player_hand)}")
        else:
            break
    player_value = calculate_hand_value(player_hand)
    if player_value > 21:
        print("You busted! Dealer wins.")
        return

    # Dealer's turn
    print(f"Dealer's hand: {dealer_hand}, Value: {calculate_hand_value(dealer_hand)}")
    while calculate_hand_value(dealer_hand) < 17:
        dealer_hand.append(deal_card())
        print(f"Dealer hits: {dealer_hand}, Value: {calculate_hand_value(dealer_hand)}")

    dealer_value = calculate_hand_value(dealer_hand)
    if dealer_value > 21:
        print("Dealer busted! You win!")
    elif dealer_value > player_value:
        print("Dealer wins.")
    elif dealer_value < player_value:
        print("You win!")
    else:
        print("It's a tie!")

# Run the game
if __name__ == "__main__":
    while True:
        play_blackjack()
        again = input("Play again? (y/n): ").lower()
        if again != 'y':
            break
```

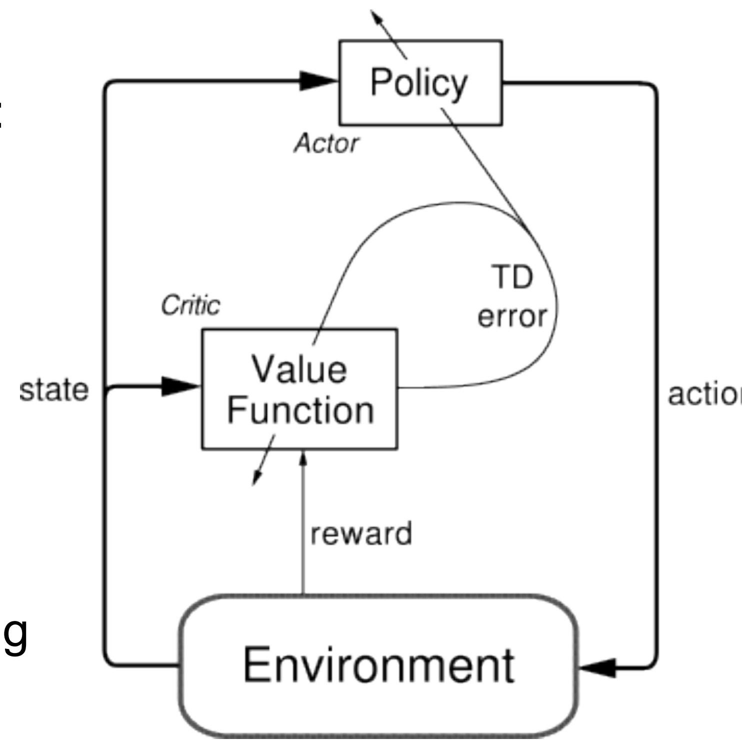
Actor-Critic Explained

- Actor-Critic reinforcement learning combines policy-based and value-based methods

- ACs use two neural networks: an "actor" that selects actions and a "critic" that evaluates those actions, by estimating their value or quality

- Actor: makes decisions by selecting actions based on current policy, it's goal is to maximize rewards by exploring the action space

- Critic: evaluates the actions taken by the actor, it estimates the value/quality and provides feedback on the actor's performance



Actor-Critic Algorithm

Two Inputs

- Player’s hand value
- Dealer’s visible card

Two Hidden Layers

- Two dense layers with ReLU activation for processing input data

Action Prediction

- Actor - The model outputs probabilities of the actions (Hit or Stay) and chooses the action based on either the probabilities or by exploring

Value Estimation

- Critic - The model predicts the expected future reward for the given state

Actor-Critic Algorithm

At each step, the model observes the current state of the game and selects an action (Hit or Stay) based on the actor's output.

The model receives a reward after each action

- Win: +1
- Loss: -1
- Tie: 0

Then it uses these rewards to update its parameters

- Actor Loss - How well the actor chose the action
- Critic Loss - How accurate the critic's value estimation was

Actor-Critic Algorithm

```
# Setup
import keras
import tensorflow as tf
from keras import layers
import numpy as np
from blackjack_env import BlackjackEnv
import matplotlib.pyplot as plt
import pandas as pd
import random

# Custom blackjack environment
env = BlackjackEnv()

# Configuration parameters for the whole setup
gamma = 0.99  # Discount factor for past rewards
num_inputs = 2 # Player hand value, Dealer's visible card
num_actions = 2 # Hit or stay
num_hidden1 = 64 # 8, 16, 32, 64, 128, 256
num_hidden2 = 64

# Set random seed
seed_value = 42
np.random.seed(seed_value)
random.seed(seed_value)
tf.random.set_seed(seed_value)
```

Actor-Critic Algorithm

```
# Define the Actor-Critic Model
# Input
inputs = layers.Input(shape=(num_inputs,))

# Hidden
hidden1 = layers.Dense(num_hidden1, activation="relu")(inputs)
hidden2 = layers.Dense(num_hidden2, activation="relu")(hidden1)

# Actor
action = layers.Dense(num_actions, activation="softmax")(hidden2)

# Critic
critic = layers.Dense(1)(hidden2)

# Create model
model = keras.Model(inputs=inputs, outputs=[action, critic])
####################################################################################
initial_learning_rate = 0.005
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate, decay_steps=100, decay_rate=0.96, staircase=True
)
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
####################################################################################
#optimizer = keras.optimizers.Adam(learning_rate=0.005) # Different rates: 0.005, 0.001 , 0.0005, 0.0001
huber_loss = keras.losses.Huber()
```

Actor-Critic Algorithm

```python
epsilon = 0.1 # Exploration factor
epsilon_min = 0.01
epsilon_decay = 0.995 # 0.999, 0.995, 0.99, 0.95
```

```python
for hand_count in range(MAX_HANDS):
    state = env.reset()
    episode_reward = 0
    action_probs_history, critic_value_history, rewards_history = [], [], []

    with tf.GradientTape() as tape:
        for _ in range(100):  # Play rounds
            state_tensor = tf.convert_to_tensor(state)
            state_tensor = tf.expand_dims(state_tensor, 0)

            # Predict action probabilities and estimated future rewards
            action_probs, critic_value = model(state_tensor)
            critic_value_history.append(critic_value[0, 0])

            # Epsilon decay
            if epsilon > epsilon_min:
                epsilon *= epsilon_decay

            # Epsilon-greedy action selection
            if np.random.rand() < epsilon:
                action = np.random.choice(num_actions) # Explore
            else:
                action = np.argmax(action_probs) # Exploit

            # Choose an action based on probabilities
            # action = np.random.choice(num_actions, p=np.squeeze(action_probs))
            action_probs_history.append(tf.math.log(action_probs[0, action]))
```

Actor-Critic Algorithm

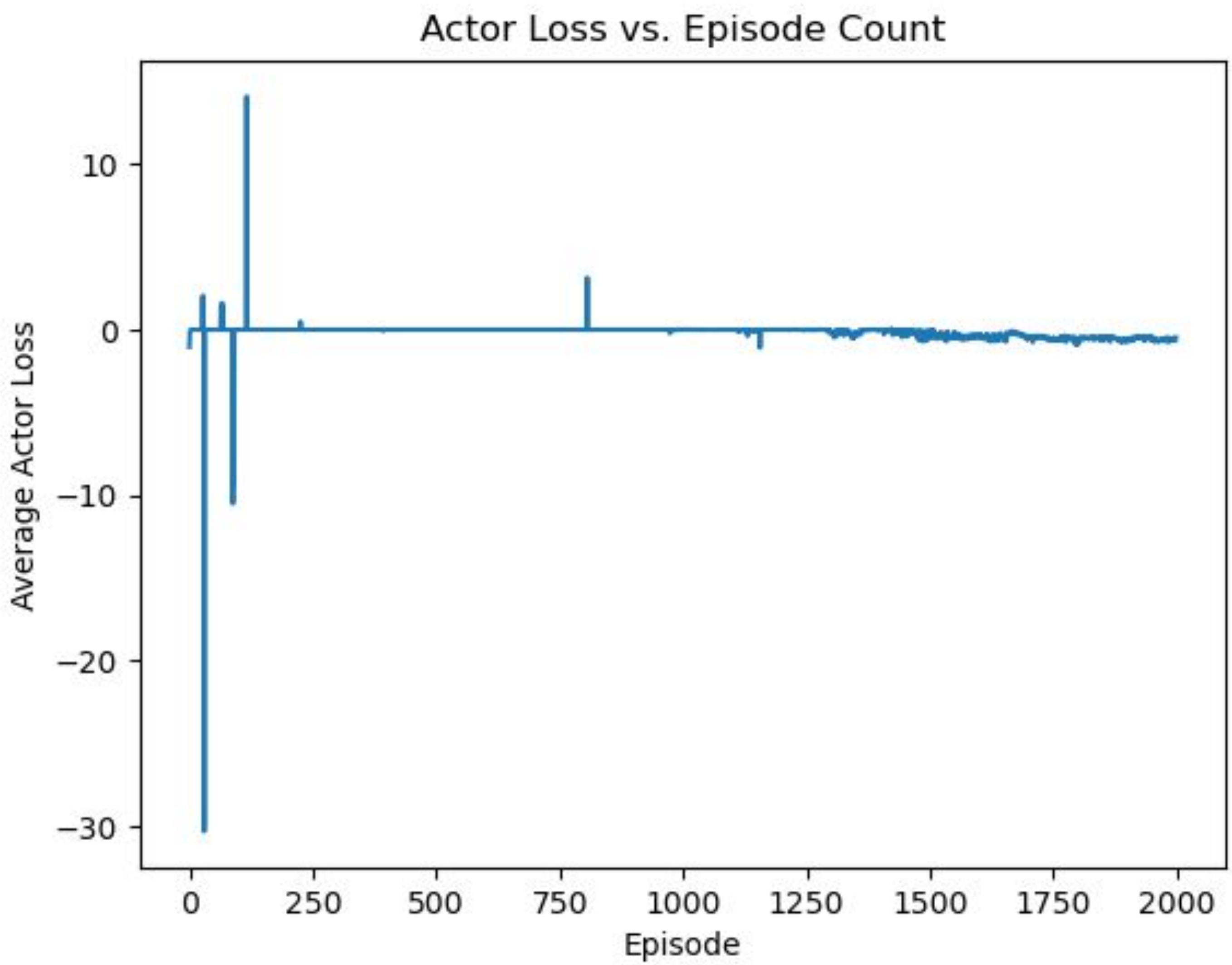
```python
# Apply the action in Blackjack
state, reward, done, player_value, dealer_value = env.step(action)
#####################################################################################
if reward == 1:
    won = 1
elif reward == -1:
    won = -1
else:
    won = 0

if player_value >= 17 and reward == 1:
    reward *= 1.1
if player_value >= 22 and reward == -1:
    reward *= 1.1
#####################################################################################
rewards_history.append(reward)
episode_reward += reward
```

```
Initial Actor Loss: -1.0488
Final Actor Loss: -0.4770
Initial Critic Loss: 1.4647
Final Critic Loss: 0.5615
Average Actor Loss: -0.1528
Average Critic Loss: 0.2477
Average Reward after 2000 episodes: -0.20
Total Wins: 738
Total Losses: 1162
Total Ties: 100
```

Optimizing the Model:

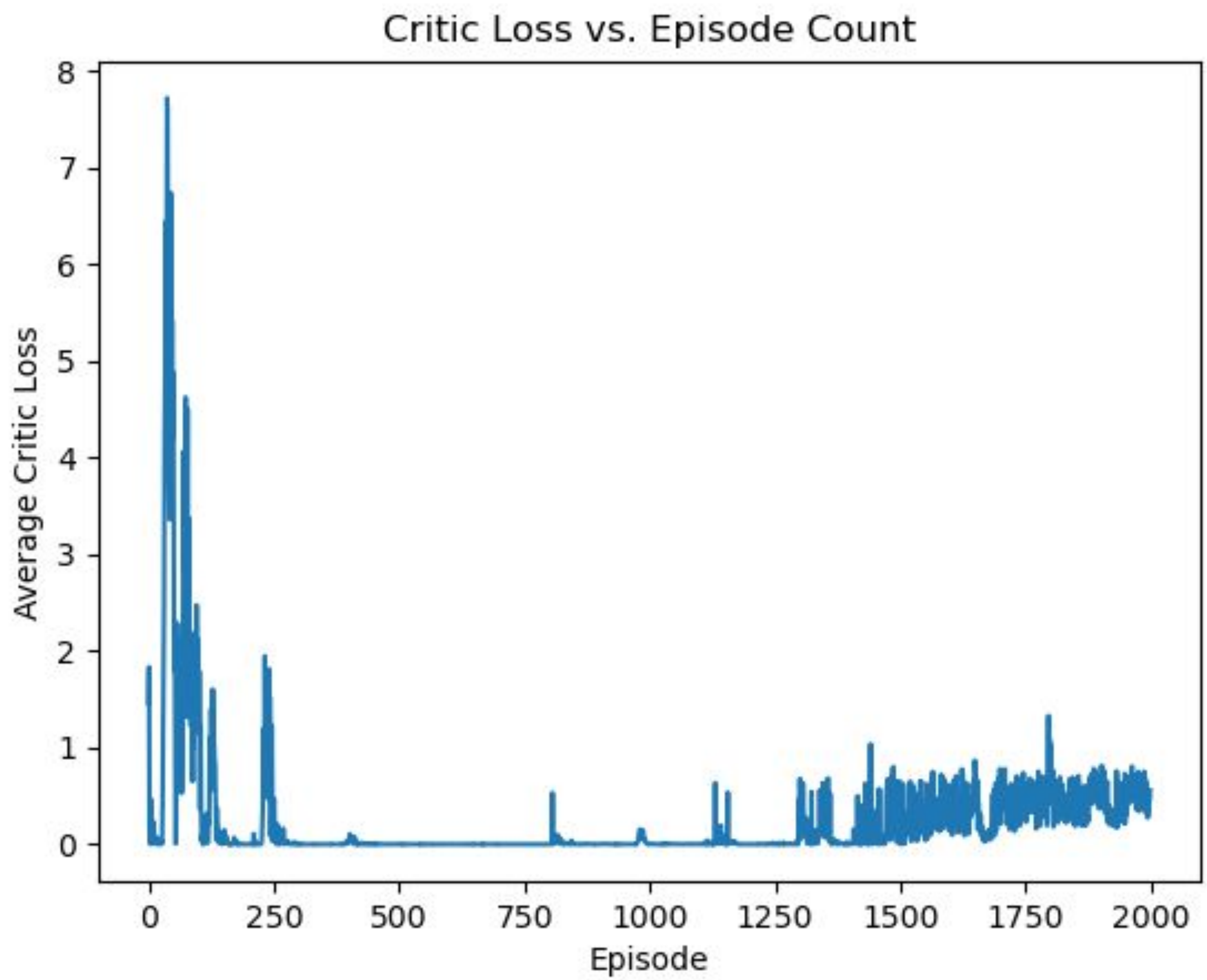
- Two hidden layers
  - **64 neurons**
- learning_rate = 0.005
- Epsilon-Greedy:
  - **epsilon = 0.1**
  - **epsilon_min = 0.01**
  - **epsilon_decay = 0.995**

Actor Loss

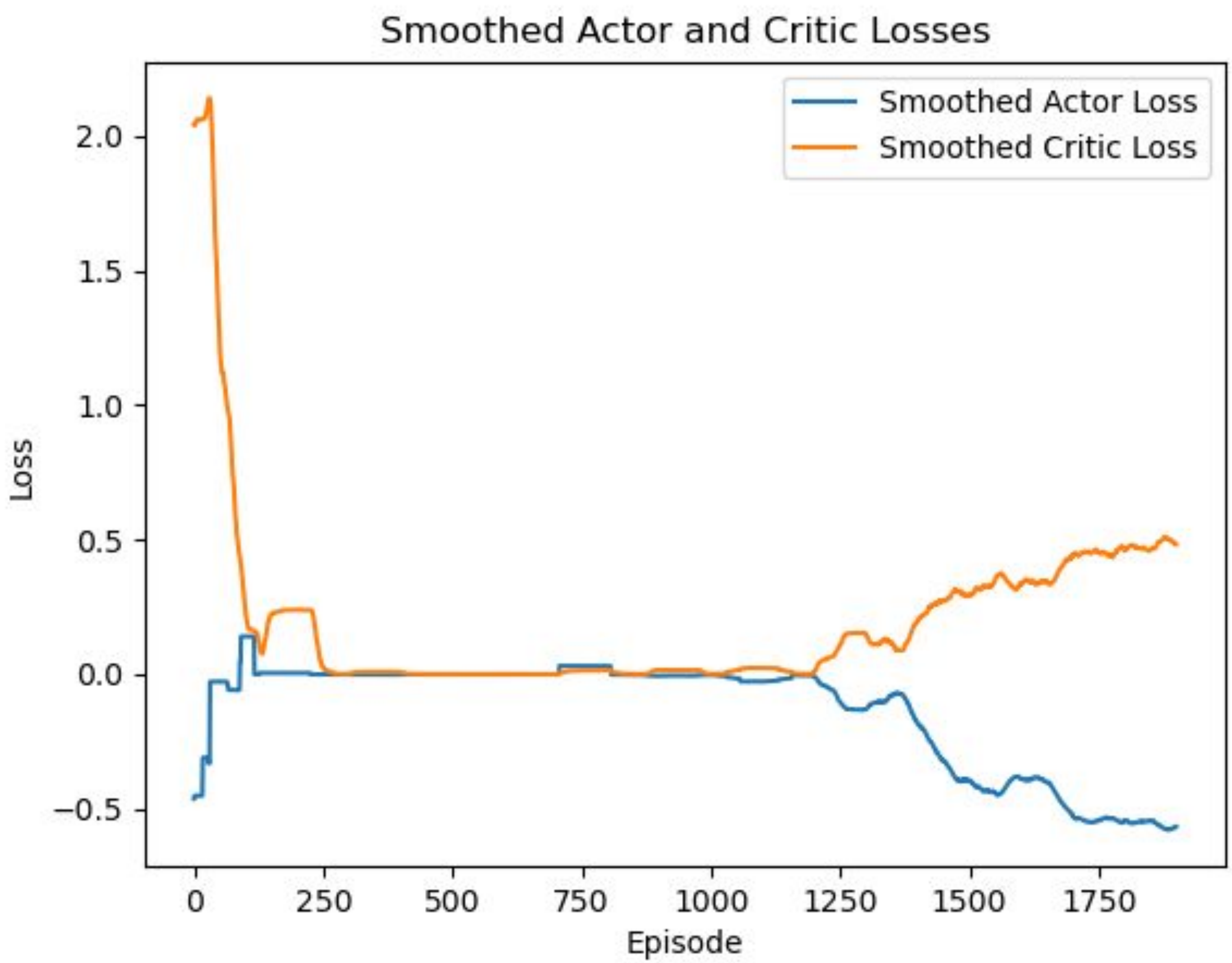


Actor Loss vs. Episode Count
Average Actor Loss
10
0
−10
−20
−30
0
250
500
750
1000
1250
1500
1750
2000
Episode

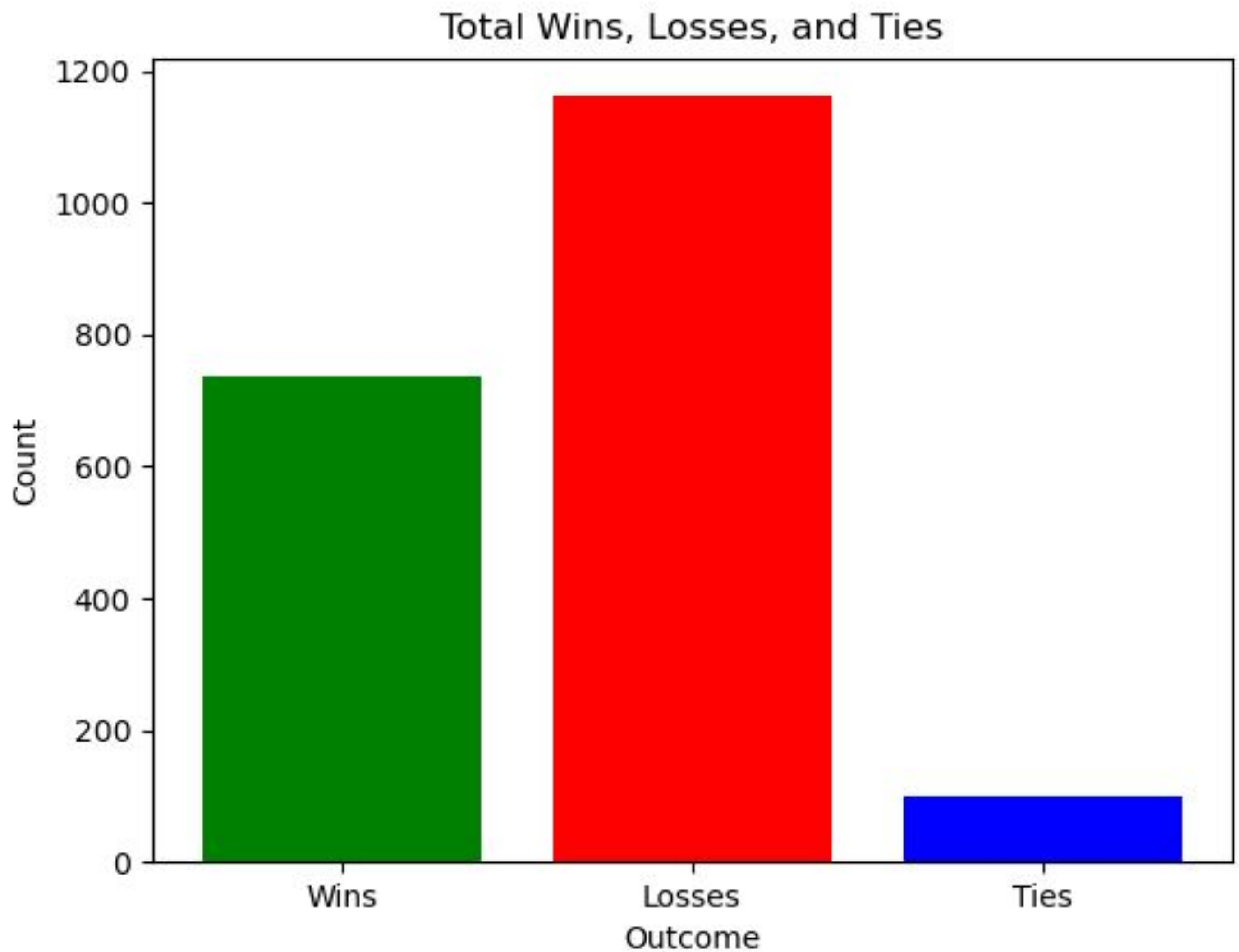
Critic Loss


Critic Loss vs. Episode Count
Average Critic Loss
Episode
0
1
2
3
4
5
6
7
8
0
250
500
750
1000
1250
1500
1750
2000

Actor and Critic Loss



Win Rate


Total Wins, Losses, and Ties
Count
0
200
400
600
800
1000
1200
Wins
Losses
Ties
Outcome

Conclusion

- Created a Blackjack environment in Python
- Created an Actor-Critic Algorithm
- The Actor-Critic Algorithm informs the player whether to hit or stay based on the information available
- Our group was able to optimize the Actor-Critic as best we could within the time constraint
- Next Steps:
  - More actions and more inputs, which likely would increase accuracy
  - Ex: having an Ace in your hand, counting cards, or further hyper tuning the model

Questions?