CSC4444 Final Project Report

Problem Domain:

The problem domain of this project was to create a basic GAN, or Generative Adversarial Network, for a simple game. The game would be based off of the Stroop Task, a psychological experiment that helps measure executive function. The theory behind the task is relatively easy to understand, that humans have difficulty reading a color word whenever the word itself is written in a different color. The game portion of this project consists of 4 square tiles that appear on the screen, each one containing a random background color, color word, color font, and font. In only one of these squares, the color word will be congruent; the word and the font it is written in will match. The rest of the squares will be incongruent. The game takes input from the numpad on the keyboard, 4 to select the top right square, 5 for top left, 1 for bottom right, 2 for bottom left. Each keypress will result in a correct or incorrect answer, ending the turn and refreshing the tiles. These turns are also ended if the user does not select an answer for four seconds, and a timer to indicate that turn time appears on screen. The game ends after 100 turns.

The AI used in this project is a GAN used to generate new data based on what background color, word color, font color, and font resulted in the most incorrect answers. The AI model takes in the testing data gathered through repeated plays of the game, in order to train a generator, which then produces enough instances of data to feed back into a version of the game that will draw use these instances to create a more “difficult” version of the game, where these specific combinations are more likely to result in the wrong answer based off of previously chosen wrong solutions. I believe that GAN would be the most helpful for this particular task because of how it generates new data, so that I could not only take a look at what kind of color combinations resulted in incorrect answers, but also find a way to incorporate that knowledge back into the game to give it a new level of difficulty.

Methodology:

The first step for this project was creating the game itself so that I could begin to gather test data to feed into an AI model whose architecture I had not decided at this point. I decided to create the game in p5, a Javascript library that allows for quick online dev with its user friendly library and documentation. Once the game was developed, I set it up so that every 100 turns, the game would end and the user data for that game would be saved to a text file. This user data consisted of each turn’s selected answer, whether or that answer was correct, the data associated with the selected answer, or, if no answer was selected, the data of the unselected correct answer. Due to time constraints, I was only able to gather the game data for 36 games, so 3600 data points, with which to train with.

After gathering the training data, I set out to determine what kind of AI would be best for implementing a way to produce new instances of data.

```
azure,brown,brown,Courier New,0
aliceblue,red,red,Verdana,1
moccasin,black,black,Courier,1
beige,gray,gray,Times New Roman,1
moccasin,pink,pink,Sans-Serif,1
```

This is the format of the training data. At first, I looked into a logistic regression model, as every piece of data had a binary result: 0 for a wrong answer and 1 for a correct answer. While this was simple enough to set up, this kind of model was more helpful for deciding whether new combinations of colors and fonts would be more likely to provide a right or wrong answer based on the combination in the training data. Though this model could be used to generate adversarial data theoretically, maybe evaluate sets of random combinations and only allow through combinations that the model predicts will result in an incorrect answer, that would require an entire separate generator and evaluator. Instead, I decided to go with a GAN model.

A GAN consists of a generator and a discriminator. The generator creates instances similar to the training data, while the discriminator attempts to determine the difference between fake examples provided from the generator and real examples from the training data. The generator takes in a random noise vector, a linear layer that maps that noise into a hidden layer of 128 neurons, a reLU activation, a second linear layer that maps the hidden representation to the output dimension, and a sigmoid activation to make sure that the output is between [0,1]. The discriminator similarly has two linear layers with 128 neurons each, a reLU, and a sigmoid activation. The loss function utilized is a Binary Cross-Entropy Loss: for the discriminator it measures how it did in determining whether the data was fake or real, for the generator it measures how well it was able to fool the discriminator. The models here both us and Adam optimizer with a learning rate of 0.001, and were trained over 1000 epochs. For this project, I utilized the p5 library in Javascript, and in Python, pandas, sklearn, and torch neural networks and dataloaders. The game portion of the project was developed in VSCode, while the GAN was developed in Google Collab, as the notebook format made development straightforward.

After finishing the GAN, I had to go back into the p5 code and create a new game mode that incorporated a text file which would feed it specific color and font combinations. This also meant allowing the user to select which game mode they would like to play, either the true randomized version of the game that was used to gather training data or the version of the game with the ai trained prompts.

Outcome:

```
Background_Color    Foreground_Color    Font_Color  Font    Target
aliceblue   brown   pink    Verdana 0
aliceblue   blue    blue    Verdana 0
azure   blue    blue    Courier New 0
moccasin    blue    blue    Georgia 0
lavender    brown   black   Arial   0
moccasin    blue    brown   Georgia 0
floralwhite blue    blue    Arial   0
aliceblue   blue    blue    Verdana 0
ivory   brown   gray    Verdana 0
floralwhite black   blue    Arial   0
moccasin    brown   brown   Arial   0
azure   brown   pink    Helvetica   0
aliceblue   brown   gray    Verdana 0
aliceblue   brown   pink    Verdana 0
lavender    brown   gray    Verdana 0
ivory   brown   gray    Verdana 0
ivory   brown   gray    Verdana 0
aliceblue   brown   pink    Verdana 0
lavender    blue    blue    Verdana 0
```

After training the generator, I used it to generate a text file that would have enough instances to play a full, 100 turn game. This required a decoder, to translate the generated data into usable text, before saving to a text file named 'generated_answers.txt'. The data provided by this generator was far from perfect, as you can see above, multiple instances where the color and font color match for incorrect answers. This could be due to a variety of reasons, but I think that the lack of a huge amount of training data and the relatively simple neural network architecture are the main culprits here. Both of these would be easy to improve if I wanted to continue working on this project. I also did not notice any particular increase in difficulty with the ai trained data. On average, from the training data, there was an average score of 49.34286, and most of my attempts to play the trained version ended around the same (I was not able to get a similar number of test trials for this version).

A Link to the game:
https://khoi-d-truong.github.io/ai_final_stroop_game/

A Link to the GitHub Repository:
https://github.com/khoi-d-truong/ai_final_stroop_game