A. Project Overview

- **Goals: 1)** Find out how good of a predictor the names of the titles of two articles by use of embeddings for whether they are “friends” or not. **2)** Find out the article with the most links pointing to it, the article with the most outgoing links, and the average distance by graph traversal between any two given articles

- **Dataset**: https://snap.stanford.edu/data/wikispeedia.html

B. Data Processing

The main alteration I did to the data before loading it into rust was by using Python’s Gensim library to create embeddings for each of the titles which would be used as predictors in my neural network as well as splitting up the titles from ids columns so as not to have to deal with typing complications in Rust.

In addition to this, in Rust, I created a function which will be described later to create “fake” data, in other words, this was a graph which was intentionally wrong so as to train the neural net to distinguish right from wrong connections. And finally, I took the difference between the source and target of each of these connections (for both the true and false data) to create the data that was actually passed into the neural network.

My theory behind this was that the difference between embeddings that were closer in meaning would result in there being dimensions of the output vector which were much closer to zero producing sort of a “negative” of the shared meaning of the two words where as the difference between two unrelated words would be more or less evenly spread out on average. However, it turns out that this is not a good predictor at all for what I was trying to do.

I would also like to note that for the sake of this project in the graphing functions, I made another file with Python which the code actually runs on called “tiny graph”. This is same as the cleaned_graph file except it has every 50th line from it. The reasoning for this was that it took my computer *so long* to run the average distance algorithm on the original so I made this smaller one just to prove that my code runs. However, if you want to try it out yourself i left the original in the folder as well.

C. Code Structure

1. **Modules**
   - Graph Operations - This included anything that had to do with the actual information from the connections in the graph, so the operations on the “most popular”, “most extroverted” nodes as well as the “average” distance between nodes

- Nerual Network - This is exactly what is sounds like: it provided the infrastructure for the neural network that I built.

-File In and Out - This module contained anything that had to do with reading from the files and turning it into data types that were actually useful in my operations later on.

2. **Key Functions & Types (Structs, Enums, Traits, etc)**

File in and Out

- **Read-ids -** This function reads the ids from the csv file and returns a hashmap of the ids and the values. The first column is the id and the second column is the value. It takes the path to the file as input and returns a hashmap of the ids and the values
- **Read names-** This function reads the names from the csv file and returns a hashmap of the ids and the names. The first column is the id and the second column is the name
- **Read embeddings -** This function reads the embeddings from the csv file and returns a hashmap of the ids and the embeddings. The first column is the id and the second column is the embedding. It takes the path to the file as input and returns a hashmap of the ids and the embeddings.

Graph Operations

- **Graph Struct -** This struct represents the graph. It contains a hashmap of the keys and the values of the graph and a hashmap of the ids and the names of the nodes. The keys are the ids of the nodes and the values are the ids of the nodes they point to
- **New Graph -** This function creates a new graph. It initializes the keys_vertex_graph and id_names to empty hashmaps and returns the graph
- **Read to Graph -** This function reads the graph from a csv file. The first column is the id of the node and the second column is the id of the node it points to
- **Most Extroverted -** This function returns the node with the most outgoing edges. It iterates through the keys of the keys_vertex_graph and finds the one with the most outgoing edges. It returns the name of the node and the number of outgoing edges and takes self as input
- **Most Popular -** This function returns the node with the most incoming edges by iterating through the keys of the keys_vertex_graph and counting the number of times each key appears in the values of the keys_vertex_graph and keeping track of the one with the mode It takes self as input and returns the name of the node and the number of incoming edges
- **Distance Recursive -** This function returns the distance between two nodes with breadth first search algorithm but should be called from distance() function which

initializes the visited vector and the distance It takes the two nodes, the initial distance, the current layer of nodes, and the already visited nodes as input
- **Distance -** This function returns the distance between two nodes. Helpful because it initializes the visited vector and the distance. It takes the two nodes as input and returns the distance between them. It uses the distance_recursive function to calculate the distance
- **Average Distance -** This function returns the average distance between any two nodes in the graph. It takes the accuracy as input and returns the average distance between any two nodes in the graph. It uses the distance function to calculate the distance between two random nodes in the graph // via a random walk algorithm
- **Try Graph - T**his function is used to test the graph operations and takes the paths to the files you want it to work on as input

Neural Network

- **False Data-** Creates a new hashmap full of wrong data for training the neural net. You should use the path to wikilink_graph_cleaned for this. The keys are the ids of the nodes and the values are the ids of the nodes they point to. The function takes the path to the file as input and returns a hashmap of the ids and the values. It works by reading the file line by line and splitting the lines into ids and values. It then checks if the key already exists in the hashmap and if it doesn't it pushes the value to the vector of values
- **Create Training Data -** for the embeddings paramter use the "read_embeddings" function from the file_in_and_out.rs file for the pairs paramter its the read_ids for true data and false_data for the false data. The function takes the embeddings and pairs as input and returns a vector of arrays. The function works by iterating through the keys of the pairs and and taking the difference of the embeddings
- **Neural Network Struct -** This struct represents the neural network. It contains the weights and biases of the network and the activations of the layers. It also contains the learning rate and the output size of the network
- **Xavier Initialization -** This function initializes the weights and biases of the network. It takes the input size, layer sizes and output size as input. It uses the Xavier uniform initialization method for the weights and initializes the biases to 0. Used for the last layer in conjunction with the sigmoid activation function
- **Kaiming Initialization -** This function initializes the weights and biases of the network. It takes the input size, layer sizes and output size as input. It uses the Kaiming uniform initialization method for the weights and initializes the biases to 0. Used for the hidden layers in conjunction with the ReLU activation function
- **New -** This function creates a new neural network. It takes the input size, layer sizes and output size as input It initializes the weights and biases of the network using the Xavier initialization method and returns the new neural net
- **Relu -** ReLU activation function and takes as input a matrix and returns a matrix. The ReLU function is defined as $f(x) = max(0, x)$. This is used for the inner layers

- **Relu Derivative -** Derivative of the ReLU function takes as input a matrix and returns a matrix. The derivative of the ReLU function is defined as f'(x) = 1 if x > 0 else 0
- **Sigmoid -** Sigmoid activation function takes as input a matrix and returns a matrix. The sigmoid function is defined as f(x) = 1 / (1 + e^(-x)). This is used for the final layer
- **Sigmoid Derivative -** Derivative of the sigmoid function takes as input a matrix and returns a matrix. The derivative of the sigmoid function is defined as f'(x) = f(x) * (1 - f(x))
- **Forward -** Forward propagation. Returns all of the intermediate and final outputs. Implements both weights and biases.
- **Backward -** Backpropagation pass through the network. No return values but it is supposed to update all weights and biases. It accepts the input, intermediate outputs and final outputs as parameters as well as the target values.
- **Try Neural Net -** Gives use accuracy of the neural network on the test data

3. **Main Workflow**

    ○ The graph operations and neural network modules both just rely on the file in and out module to read and convert the files into useable datatypes

D. Tests

- `cargo test` **output** (paste logs or provide screenshots).

- **Test Most Extroverted -** this function is used to test the function "most extroverted" in the graph_operations module. it takes the paths to the files you want it to work on as input. This was important because it alson inherently tests the file reading functions which are used in the Neural Network
- **Test Most Popular -** this function is used to test the function "most popular" in the graph_operations module it takes the paths to the files you want it to work on as input
- There was a lot of randomization in my code which makes it function more accurately but doesnt lend well to testing in the traditional sense. Because of this, although I only have two tests, the functions that are implied inside of these tested functions are also tested.

```
     Running unittests src\main.rs (target\debug\deps\DS210finalproject-68faeab66cec1a38.exe)

running 2 tests
test test_most_popular ... ok
test test_most_extroverted ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

E. Results

```
    Finished `release` profile [optimized] target(s) in 1.00s
     Running `target\release\DS210finalproject.exe`
Graph Facts:
Most extroverted node: 'List of mathematics articles' with 23 outgoing edges
Most popular node: 'Asia' with 766 incoming edges
Average distance between any two nodes: 1.2
```

```
warning: `DS210finalproject` (bin "DS210finalproject") generated
    Finished `release` profile [optimized] target(s) in 0.08s
     Running `target\release\DS210finalproject.exe`
Neural Net Accuracy: 53.4445%
```

**As I was testing these, I ran them each separately but you can run them together as they are in the main function

Neural Network
As far as the neural network accuracy goes, I think that this just means that the word embeddings, at least those produced by the Gensim library are poor predictors for whether to words are "friends". One explanation for this is that even if two words share a similar meaning, they can only have so many connections practically speaking. So there may be many words in the dataset that are semantically closely related but don't have links to each other causing he neural network to not train in such a way that would make that connection. Perhaps a good addition would be to include the number of outgoing connections a given node has as part of its training data to show how relevant it is.

Graph Operations
For the graph operations, I think that the results are pretty self explanatory as far as the given data. That is Asia being so popular and the mathematics article being quite relevant as well. As or the average distance between nodes, I was surprised at how low this was. It would be interesting for a future project to find the most popular connector between any two given nodes.

F. Usage Instructions

- All you need to do is provide the command “cargo run –release” in the terminal. Everything is setup already in the main() file. You should expect a run of time of ***several minutes.***

G. AI-Assistance Disclosure and Other Citations

Citations for improving Neural Network (weight initialization and bias)

https://www.geeksforgeeks.org/the-role-of-weights-and-bias-in-neural-networks/

https://www.geeksforgeeks.org/kaiming-initialization-in-deep-learning/

https://www.geeksforgeeks.org/xavier-initialization/

https://www.youtube.com/watch?v=tYFO434Lpm0

- **AI** was most notably used for learning how to make randmoized values that are spread over a gaussian distribution for the weight initialization of my neural network. The response is pasted below (I did not just copy and paste all of the code) but like I said this help was rather minimal).

To get a random number with a **normal distribution** in Rust, you can use the `rand_distr` crate, which extends the functionality of the popular `rand` crate by providing support for various probability distributions—including the **Normal** (Gaussian) distribution.

Here's how you can do it:

1. Add dependencies to `Cargo.toml`:

- [dependencies]
- rand = "0.8"
- rand_distr = "0.4"

2. Generate a normally distributed random number:

- use rand::thread_rng;
- use rand_distr::{Normal, Distribution};
-
- fn main() {
-     // Create a normal distribution with mean = 0.0 and standard deviation = 1.0
-     let normal = Normal::new(0.0, 1.0).unwrap();
-     let mut rng = thread_rng();
-
-     // Sample a random number from the distribution
-     let sample = normal.sample(&mut rng);
-     println!("Random sample from N(0, 1): {}", sample);
- }

Explanation:

- `Normal::new(mean, std_dev)` creates the normal distribution.
- `sample(&mut rng)` gives you one value from it.
- You can generate many samples if you need a sequence.

Would you like an example of generating and plotting a histogram of samples in Rust?