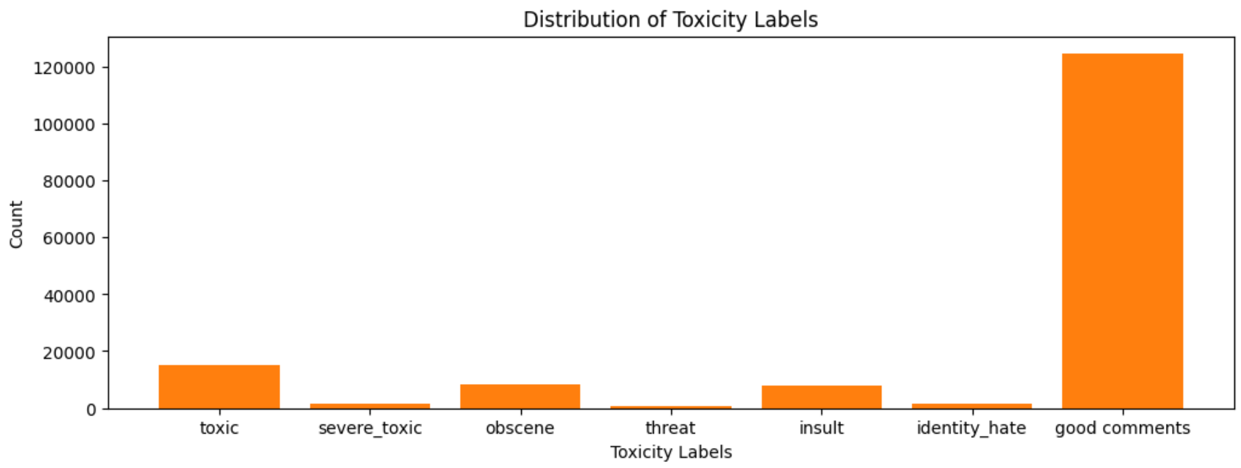
Project Group 64
Minh Nguyen - 38731850
Raymond Chou - 20123271
Venkata Neti - 72723519

Overview

For our project, we chose to analyze comments from Wikipedia's talk page edits and label them according to different levels of toxicity using a model. Types of toxicity labels within this dataset include *toxic*, *severe_toxic*, *obscene*, *threat*, *insult*, and *identity_hate*. Each comment is uniquely identified by an *id* and contains unfiltered text with all forms of characters.

To better visualize the training dataset, Raymond and Minh created a bar plot to visualize the distribution of the different toxic labels in the dataset. During this process, we created a new label *good comments* for comments with no toxicity label, or where all their target labels were 0, as shown below:



In total, there were 159,571 comments for the training dataset above that will be eventually compared to the 153,164 comments of the testing dataset.

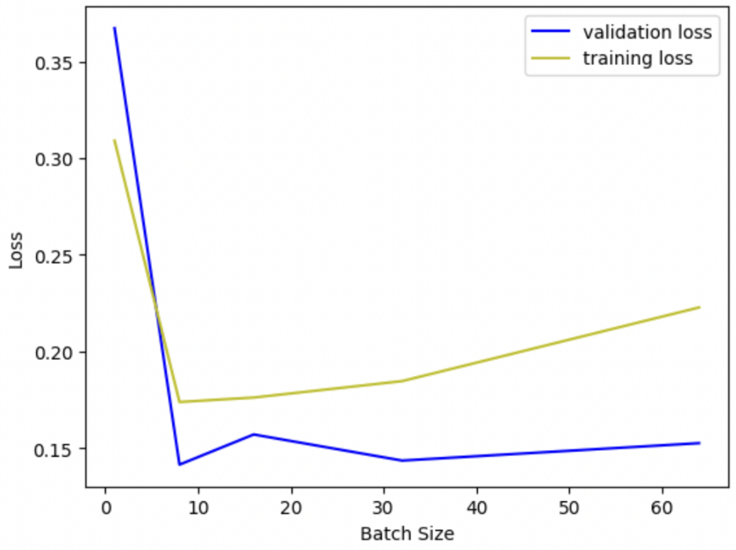
Preparation

Minh preprocessed the data by cleaning the text to remove any unnecessary elements from the text such as HTML tags, punctuation, or links. We then stored the cleaned text and tokenized it to allow for more accurate data analysis and improve the performance of our model. Stop words and capitalization of letters were kept to retain the context of the actual comments, which is especially crucial for contextual models like BERT that we were going to use. While 80% of the training data was kept as is, we split the other 20% into our validation set for less overfitting and better model training/testing.

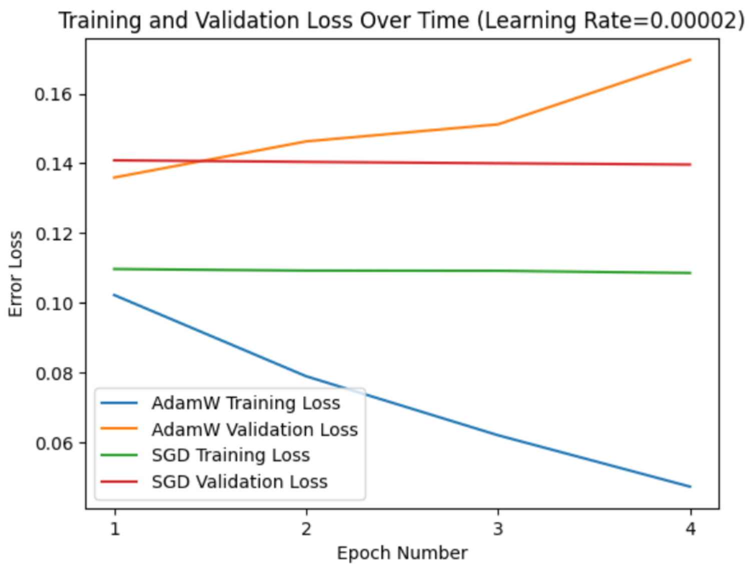
Analysis

For our data analysis and model training, we all used BERT, a natural language processing model that can understand the context and relationships between words in a sentence. This made it easier to decipher meaning behind different comments. We used the pre-existing PyTorch libraries, to get the BERT Encoder and BERT Classifier that we later fine tuned for our specific task. We encoded the training comments into vectors and trained the BERT classifier on these encodings using AdamW and Stochastic Gradient Descent.

For the model, Venkata experimented with different batch sizes but settled on a batch size of 32, as that landed a balance between efficiency and accuracy when running our model. For the activation function, Raymond experimented between the sigmoid function and the softmax function but ultimately landed on the sigmoid function because we wanted to implement binary classification and distinguish between toxic/non-toxic comments. Furthermore, it yielded a higher accuracy compared to softmax when evaluating our model later on using our predicted labels with the true labels within the dataset.



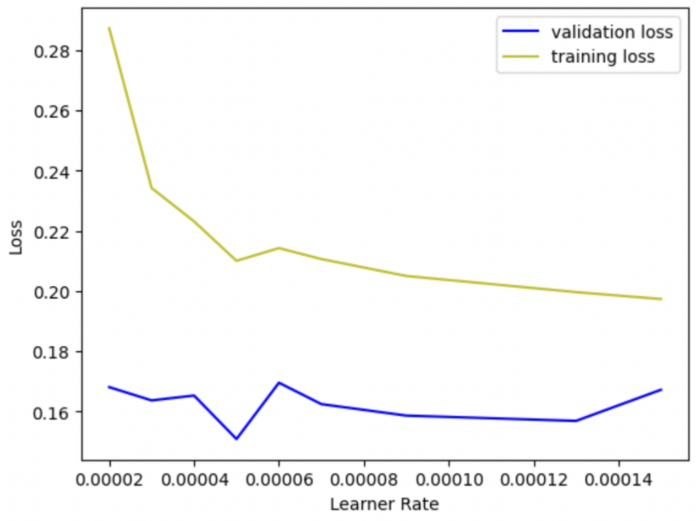
Speaking of our model, Raymond and Minh first optimized the model using AdamW, which minimizes the prediction loss and regularizes the weights of the model with weight decay. Weight decay helps prevent overfitting by penalizing large weights in the model, which allows for a more generalized learning process. We then optimized our model using Stochastic Gradient Descent (SGD), as it fits well with large datasets since it only chooses a small, random subset of the training data to tune our model. The results between these two models are shown below:



Project Group 64
Minh Nguyen - 38731850
Raymond Chou - 20123271
Venkata Neti - 72723519

Raymond and Minh experimented with a different number of Epoch iterations, and found that the training loss for AdamW at epoch = 1 did the best. Therefore, we went with the AdamW optimizer and set the number of epochs to 1 for our model. Regardless, the training loss is much lower than the validation loss in general. This indicates that the model might have trouble generalizing to new types of data.

Meanwhile, Venkata then tried to figure out a good learner rate (step size) for our optimization function that's about to do the gradient descent. We let BERT do mini-batch gradient descent multiple times with different learner rates and compared the results, as shown below:



The graph showed that 0.00005 would be the optimal learner rate as both the training and validation losses are the lowest at that rate.

Finally after tuning our parameters to the optimal (perfect fit) value, we got 73% accuracy and 86% precision!