Module 4: Critical Thinking

Toxicology Testing

Nolan Byrnes

CSC580 – Applying Machine Learning and Neural Networks - Capstone

Colorado State University – Global Campus

Dr. Isaac Gang

October 8th, 2023

Module 4: Critical Thinking

With a dataset containing chemical compounds and whether the compound is toxic, we can develop a machine learning model that is able to predict whether a compound will be toxic or not. With the Tox21 dataset there are a few challenges that occur, one being that the dataset is imbalanced. As with any other machine learning model, we also want to avoid overfitting on the training dataset. This paper goes over the architecture of the model developed, an overview of the TensorBoard graph and loss curve, a review of the model’s accuracy, as well as the steps that can be taken to avoid overfitting.

The architecture of the neural network included an input layer, two hidden layers, an output, loss, and optimizer. The input layer takes in the x values, y values, as well as the dropout rate. I had two hidden layers where both contained a dropout. Ramsundar et. al. defines a dropout as “a form of regularization that randomly drops some proportion of the nodes that feed into a fully connected layer” (Ramsundar et. al., 2018, pg. 91). By dropping random nodes through the hidden layers, it prevents the model from having to rely on the strongest nodes to make its predictions, which in turn helps the model generalize better. For both of my hidden layers, the keep\_probability set to 0.9 when the model was going through training, and 1.0 when the model was predicting. Because of the keep\_probability being set to .9, each of the two hidden layers would drop 10 percent of the nodes. When using the model for predictions, we do not want any of the nodes to be dropped, so we have the keep\_probability set to 1.0 when the model is making predictions.

After the two hidden layers, the data goes to the Output layer, where the data provided from the second hidden layer is multiplied by the weight, and then the bias is added. That value then gets passed through the sigmoid function to give a probability of the compound being toxic. That probability then gets rounded, to give us the final prediction on whether it is toxic. You can observe the TensorBoard Graph of the model in figure 1 below.

Figure 1.

Model TensorBoard Graph

A diagram of a loss

Description automatically generated

Note. This figure displays the architecture of the machine learning model that was trained to detect toxic compounds.

The two classes within the dataset were 0 and 1, 0 representing a non-toxic compound, and 1 representing a toxic compound. Within the training data set, the two classes were not represented evenly. In fact, 95.99% of the training data set was labeled as non-toxic, as displayed in figure 2 below.

Figure 2.

Percentages of Toxic and Nontoxic compounds in training data set

A black background with white text

Description automatically generated

Note. This figure displays the percentages of toxic and non-toxic compounds in the training data set.

To balance out the disproportionate dataset, Ramsundar et. al. recommended we use MoleculeNet’s per-example weights, which weighted the toxic samples 19 times more than the non-toxic samples (Ramsundar et. al., 2018, pg. 99). If all classes were weighted the same, then the model could predict everything to be non-toxic, and the accuracy score would come back as 95.99%, which would not be useful. By giving toxic samples a weight that is 19 times more than the non-toxic samples, when we calculate the accuracy rate of the model’s predictions, we get a better representation on how accurate the model is when predicting if a compound is toxic or not. In figure 3 below, you can observe the weighted accuracy scores of the training, evaluation, and test data sets.

Figure 3.

Weighted Classification Accuracy

A computer screen with white text

Description automatically generated

Note. This figure displays the weighted accuracies of the trained model on the training, validation, and test datasets.

As observed in figure 3 above, the training data set had a better accuracy than the evaluation and test data sets. This could be a sign that the model is becoming overfit, which is where the model is memorizing the training data set. When I set the hyperparameter keep\_probability, I observed that the training accuracy was much higher compared to the validation and test accuracies.

The model could potentially have had better results if it was trained with less epochs. To keep the model on its toes during training, I could also have had the training data randomized between each epoch, to prevent the model from memorizing the dataset. To improve the model, hyperparameter optimization, and perhaps shuffling the training dataset between epochs could be performed.

In figure 4 below, it is noticed that there are segments that are noticeably higher than the following segments, and over the time series, the change in the average y-axis of these segments decrease over time. Instead of updating the weights after each step of training within an epoch, the model used mini-batches, which is where we update the weights and biases after a defined set of steps within the epoch.

After each mini batch, the optimization function is run, which is an algorithm “used to decrease loss (an error) by tuning various parameters and weights, hence minimizing the loss function” (GeeksForGeeks, 2022, para. 1). One of the parameters that gets updated by the optimization function is the weight, and the amount the weight gets changed is the step size. Each time the optimization function is run, the weight gets changed by the configured step size in the direction that would minimize the loss function. After each weight change, it is overshooting the optimal values for the loss function, and throughout training, after each weight change, it is getting closer to the optimal values. You can observe this behavior in the time-series TensorBoard Graph, which represents the loss over the training period in figure 3 below.

Figure 4.

Trained Model Loss Curve

A graph with red lines

Description automatically generated

Note. This figure displays the time-series loss curve of the training model.

Conclusion

There are a few considerations that must be considered when training a dataset to classify a dataset, especially when you have an imbalanced dataset to train with. Weights must be included when calculating the accuracy of the model, to help more accurately give an accuracy rate of the model’s predictions. In the tox21 dataset, 95% of the chemical compounds were non-toxic, so when the model makes predictions, if there is not a weight attributed to the chemical compounds that were toxic, then we would not get a fair representation on how well the model is able to predict a toxic compound. Implementing a dropout in the hidden layers, it helps the model with regularization, preventing it from relying too much on its strongest nodes to make a prediction, forcing it to adapt and find new patterns. There was a peculiar pattern with the resulting time-series loss curve, with segments appearing high, then suddenly dropping, with an alternating pattern. This sudden shift could represent the shift in the weights due to the optimization function, and since they were getting closer together throughout its training, indicates that the model was getting closer to the local minima. Shuffling the data sets between each epoch could have also kept the training model on its toes, preventing it from memorizing the data set, and perhaps reducing the discrepancy between the training and test data set prediction accuracies. Hyperparameter optimization could also be performed to increase the performance of this model, perhaps by adjusting the number of epochs, or the training rate.

**REFERENCES**

GeeksforGeeks. (2022, December 21). Optimizers in TensorFlow. [https://www.geeksforgeeks.org/optimizers-in-tensorflow/#](https://www.geeksforgeeks.org/optimizers-in-tensorflow/)

Ramsundar, B., & Zadeh, R. B. (2018). TensorFlow for deep learning: From linear regression to reinforcement learning (1st ed.). O'REILLY.