Sentimental Smacking: A Comparison of ML Visualization Tools

Jarred Bettencourt, Jeffrey Tran

May 10, 2022

1 Introduction

1.1 Motivation & Problem

With technology continuously advancing and this advancement having no end in sight, the topic of computing has gained more relevance in modern society. The amount of aspiring software engineers inspired to learn about each subtopic of computing has dramatically increased as a result. Additionally, the demand for skilled and knowledgeable software engineers has led to an increase in beginner-friendly options to learn said skills. However, due to the competitive nature of the computer science field, some experience difficulty in choosing what position to apply for based on their experience with using technical tools. Simply put, the varying experiences of individuals causes a large discrepancy in computing knowledge, and this dissuades individuals from not pursuing things they enjoy due to steep learning curves. This gate keeping of individuals due to a lack of unintuitive beginner tools hurts everyone, as diversity of backgrounds constitutes a healthy environment for the advancement of computing as a discipline. Due to unintuitive beginner tools existing, we aim to observe the various aspects (or lack thereof) that could potentially steer individuals away from using a tool, like its usability, ease-of-use, and extensibility.

Figure 1: TensorBoard Toolkit

With that said, one of the more recent computer science areas to rise to prominence is machine learning. This technical topic is built on the foundation of developing models that perform tasks without specific instructions. This practice is achieved by teaching a computer model to input data, which then learns relations between data to hopefully output results that would be akin to a human outputting results in the same situation. As a result of its extraordinary usefulness, aspiring software engineers have taken an interest in machine learning and its potential uses. The flexibility present in machine learning is just simply too overarching to not utilize it for various purposes, whether it is data determination or analyzing statistics. Despite the usefulness, it would be a disservice to the discipline to understate the difficulty of machine learning. While ML models very typically achieve results even better than humans, the underlying principles are based on advanced mathematics which can steer individuals away if they lack the mathematical background needed to analyze what is happening in programs utilizing machine learning. Due to the varied backgrounds of people in the computing industry, it is the case that many individuals lack knowledge related to machine learning but have a desire to learn about the discipline. This causes difficulty in choosing which tools to utilize when approaching more specific areas of the topic.

The penultimate problem our project aims to address is whether non-programmers can use a machine learning visualization kit to understand, complete, and continue work on an advanced task in ML. From a software engineering perspective, the objective of solving this problem is observing how the usability, ease-of-use, and extensibility of a technical tool factors into the development process for any individual. By observing the results of the solution, we can gain a sense of how much general knowledge a person needs to get started with working in not only machine learning, but computing as a whole. Our hope is that the results of this paper is not only to give exposure to beginner-friendly visualization toolkits for ML, but also to encourage developers to seriously consider adopting a more beginner-friendly paradigm when designing ML tools that millions around the world could potentially end up using.

Figure 2: Weights & Biases Toolkit

For our deep dive into comparing visualization toolkits, we chose very popular visualization tools, which are Weights & Biases, and TensorBoard. Being very popular tools, we believe analyzing them from a non-programmer standpoint has wide implications for ensuring that learning and understanding neural networks is bottlenecked by understanding the concept of neural networks rather than understanding the (potentially complex) software which is used to visualize the results. We additionally chose to analyze both toolkits in the setting of sentiment analysis, which as we will see is a hot topic in the field of machine learning as a whole.

1.2 Sentiment Analysis

Sentiment analysis, broadly speaking, is the machine learning task associated with classifications of bodies of text. Whereas most machine learning tasks concern themselves with classifying numerical quantities (regression), the problem of text classification adds another layer of complexity to the task as a whole. The idea of sentiment analysis at a high level is to input a set of bodies of text along with labels, which correspond to labels A or B. Then, with a machine learning method, a model is trained which is able to take in a body of text and (hopefully) output the label that a human would in the same scenario. The most prominent case of this is having labels that correspond to positive or negative, and then creating a model which outputs whether a a body of text has a positive or negative sentiment, owing to the namesake of "sentiment" analysis.

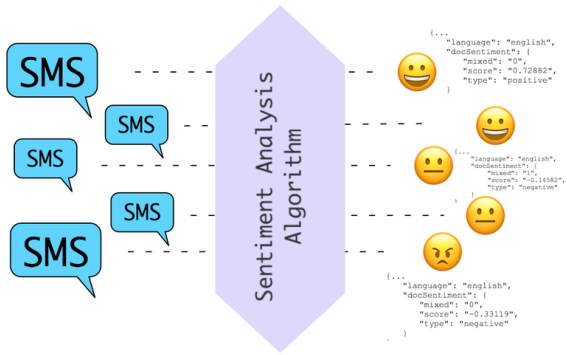

Figure 3: Graphical representaion of Sentiment Analysis

Despite its complexity, sentiment analysis is used very frequently (both successfully and unsuccessfully) in most computing environments in one form or another. The task of classifying a particular email as either spam or not spam relies heavily on the use of assigning a numerical "weight" to each word in an email, and if the sum total of all of the weights for the words in a particular email exceeds a threshold, the email is classified as spam. The hard part in this scenario is finding the optimal weights while lead to the maximum amount of true positives and true negatives (although as our inboxes show, false positives happen quite frequently!) To successfully solve the problem, data scientists have realized that it is a problem aptly suited for machine learning in one form or another, as we are essentially hoping a machine can categorize an email the same way that a human would. Sentiment analysis can solved using either a traditional machine learning or utilizing some form of deep learning technique.

In the context of the previous paragraph, traditional machine learning methods simply refer to machine learning algorithms that do not include a deep learning component in their model initialization. When referring to sentiment analysis, there are not a large amount of traditional machine learning models adapted specifically for the task, but the most popular is the so-called "bag of words" model. The Bag of Words model looks at probabilities of words and assigns a score to the words based on a combination of the likelihood of the word appearing and its likelihood of being assigned to a particular class. Then, as mentioned above, the sum total of word scores for all words in a body of text are summed, and assigned to a particular class This is of course a simplification of the method, but for our purposes it suits as background knowledge for the problem we are trying to solve.

Deep learning works slightly different for sentiment analysis. Deep learning techniques refer to machine learning models which include layers that continuously transform data as it passes through the network, and uses the transformed data to learn patterns from to ultimately make classifications. We use the term "neural nets" to refer to deep learning models which include layers as they try to mimic the human brain and this paper will continue to use the term frequently. Neural nets are frequently used to solve the problem of sentiment analysis, as they typically achieve higher accuracy when classifying any type of text compared to the Bag of Words model. Additionally, they allow a

higher degree of customization of certain layers depending on the model creator's specifications.

2 Approach

In order to evaluate the differences and ultimately compare the performance/features of visualization toolkits for deep learning, we of course have to create a deep learning model and test on it. Then, in addition to noting the performance of the neural network in classification (metrics like accuracy), we also want to note the features of the visualization toolkits which a non-programmer will experience if they choose to interface with the particular toolkit. Our pipeline, in specifics, are the following steps:

2.1 High Level Overview

1. Gather relevant dataset to fit model on (text and labels)
2. Create neural network for sentiment analysis
3. Gauge the performance of the neural network in terms of accuracy
4. Contrast the following software metrics between Weights & Biases and TensorBoard
   (a) Ease-of-Use
   (b) Extensibility
   (c) Usability

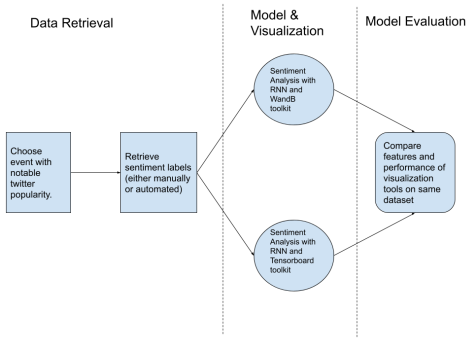

Figure 4: Pipeline steps for Sentiment Analysis

2.2 The Will Smith and Chris Rock Dataset

In order for sentiment analysis to produce accurate classifications, the dataset must include a comparable amount of examples that belong to class A and class B. That is, the dataset must have adequate examples in both sides to be able to learn effectively. As an example, moral dilemmas that an individual can agree or disagree with work well due to the large diversity of opinions and hence large diversity of training examples. By labelling each point of data based on the different feelings each person has, the accuracy of the model will improve for classifying a certain related sentiment. With that said, we chose to use the moral dilemma of Will Smith and Chris Rock at the 2022 Oscars Ceremony as our focus for the dataset. Since both are respected figures in the filming industry, many people had different sentiments when the incident initially occurred. As such, it was an excellent choice to gather data for the dataset we used.

As a brief aside for context, the event involved famous actor Chris Rock making an insensitive joke about Will Smith's wife, which prompted Will Smith to strike Chris Rock on national television with hundreds of thousands of viewers.

The incident was heavily reported on regardless of media outlet. The event was broadcasted on live television during a ceremony and many people were shocked that two celebrities were involved in a petty situation over a small comment. Despite this, according to Andrew R. Chow of TIME, the incident would cause "...a social media storm" on platforms like Twitter. [3] As a

Figure 5: Will Smith and Chris Rock altercation

result, heated arguments began to appear on social media, with different opinions on who was justified in the situation. With the two sides split between Will Smith and Chris Rock, there was enough data to compose a dataset of Twitter posts from people declaring whose side they were on. Through manually labeling each tweet using the labels "WillSmith" or "ChrisRock", the dataset had a diverse set of vocabulary for the machine learning models to train with as to develop an accurate score when performing classification later on. Additionally, each post offered more insight on the types of sentiment held for the central problem as a whole, regardless of it was in favor of Will Smith or Chris Rock. This breadth of insight only served to increase the accuracy of our model.

The dataset in question is available on our Github repository[1] listed in the references section at the end of the paper. We implore readers to run their own models on the admittedly small dataset to perhaps improve on the accuracy and efficiency.

2.3 Recurrent Neural Networks

For the creation of our model, we chose to use a Recurrent Neural Network, or RNN for short. Recurrent Neural Networks are a specific class of neural network that are specifically tailored for any kind of data that has the quality of being temporal. By being temporal, we mean that the order in which the data is positioned is important to classification in some way. A popular example of temporal data is of course any data set which includes a time component, but a perhaps not so obvious example of temporal data are English sentences. When English sentences are semantically broken down, it is clear that the positioning of words can define new meanings in two different sentences even if they contain the same words. So, a neural network suited for temporal data is quite applicable to sentiment analysis, which caused us to choose an RNN as our way to process and classify on data.

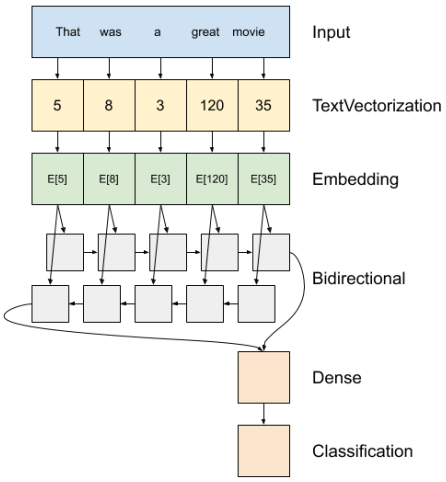

Figure 6: Model of RNN from TensorFlow example

For accurate classification, it is important that the tweets passed in to the RNN are adequately sanitized. By sanitized, we mean that the tweets are formatted in a specific way that makes processing them in the model possible. Semantic analysis simply begins to lose effectiveness when we try to factor in things like punctuation, or extremely common words like "the". We can think of it intuitively like this; does knowing that a tweet contains "the" give us more information about whether it belongs to class A or class B? Additionally, does the presence of a comma give us information in the same way? To put it simply, the answer to both of these questions is no, it does not. Some common words are simply so common in languages that their presence does not help in meaningfully distinguishing a body of text as one class or another. If we try to factor in punctuation, our model is learning too much of the semantics behind a language, which will inevitably lead to overfitting to a dataset and not generalizing well. So, for our purposes, our preprocessing step for tweets involved removing punctutation, extremely common words (stop words), and assembling a tweet as a simple string of lowercase words. This helped our model to generalize and is common practice for any task in Natural Language Processing.

After preprocessing, we created a vocabulary from the most common words in all 100 of the tweets. This step is essentially choosing how many words out of the entire dataset will contribute to classifying a tweet as being in one class or another. If a word is not included in the vocabulary, then no weight is optimized for it and hence it has no impact on the classification in general. The size of the vocabulary is a hyperparameter that is defined by the creator of a model, and finding the vocabulary size that optimizes the accuracy is a balancing act. With too high of a vocabulary size, the model will overfit to the words specific to the training dataset, and thus the model will not generalize well to data outside of the training set. This is opposed to too low of a vocabulary size, which will cause the model to not be able to classify accurately because the model will be considering too little information. For our

purposes, we chose a vocabulary size of 50, because for our small example with 100 training examples, it garnered accuracy up to our standards.

Finally, for our choice of layers, our design is largely based on an RNN example for sentiment analysis from the TensorFlow website[5]. Still, our layers were adapted to specifically work with twitter and the dataset we uniquely obtained. For our layers, we include the following

1. Embedding Layer
2. Bidirectional Layer
3. ReLU Activation Layer
4. Classification Layer

In a way so not to as stress the specifics of each layer, I will explain the job of each layer at a high level. The embedding layer transforms the vocabulary into a format that can be processed by the network, which is followed by the bidirectional layer that actually learns and applies the weight of each word for classification. Finally, the ReLU activation layer forces a classification as either being in class A or class B, and the classification layer outputs this result to the user. The bidirectional layer is the portion of the network that contains most of the "learning" in the model, with the other layers simply helping to transform the data in a way that works with the bidirectional layer. Now, with the neural network defined, we will talk about the visualization toolkits which the bulk of our project concerned itself with.

2.4 Weights & Biases Visualization Tool

Weights & Biases is a free and open source deep learning visualization tool suited for not only deep learning tasks, but all machine learning methods. It was created in 2018 to address the lack of visualization tools that are available to neural net researchers, while maintaining the belief that integration of WandB should be quick and integration should be seamless. If integration is not seamless and affects the creation of a model, then it would be more likely to passed for another tool which is not obstructive to the already complex process of defining a neural network.

Figure 7: System Analytics from WandB

Weights & Biases is nearly plug and play, with only around 1-5 lines of code needing to be added to a python file to allow visualizations. After adding these lines, a dashboard in the browser opens up with several predefined ways to visualize the training metrics (accuracy, error, etc.) or visualize system analytics inherent to the system (GPU usage, temperature, etc). This all happens with no configuration from the user. Additionally, it allows the addition of new user-defined plots to tailor WandB for any use case.

A feature touted by Weights & Biases is its ability to help hyperparameter optimization. Essentially, WandB can help ML model maintainers to find optimal hyperparameters by comparing similar iterations of a model and extrapolating on what values hyperparameters should be set to. Anything that can help maintainers with hyperparameter selection is extremely useful, as they are the core components that make a model accurate or inaccurate.

The final thing to note about Weights & Biases before truly evaluating it from a software engineering perspective is the level of interactivity that exists between it and the user. From the dashboard that WandB provides, it is possible for a user to halt the model training process even if not at the local machine where training is occurring. While the uses of this are not useful in small scale cases, it is still a convenient feature. This interactivity also lends well to the fact that WandB can be hosted locally, or cloud hosted on the WandB website. For individuals, cloud hosting is very convenient, while large organizations may prefer local hosting for data obfuscation and privacy.

2.5 TensorBoard Visualization Tool

TensorBoard is an open deep learning visualization tool which is similar to Weights & Biases in that they both help creators of deep learning models to visualize results of their created models. It is included in the overwhelmingly popular deep learning library TensorFlow, which reduces the amount of installations a user needs to make. Before explicit evaluation from a software engineering perspective, there are differences to note between the two that contribute to our overall ratings in our evaluation step.

The first thing to note is that TensorBoard is extremely customizable. As a tool, it really assumes that a user will spend the time learning how to customize it to their needs and consequently does not provide many of the starting plots that WandB includes, like system analytics. If one wishes to visualize system analytics like in WandB, the user needs to explicitly keep track of the analytics, and incorporate it into the TensorBoard instance manually. This extreme customization is not inherently a bad thing, but it assumes a large level of effort and knowledge from the developer.

In contrast to WandB, the TensorBoard instance must be locally hosted. Again, this isn't always a bad thing, especially when the majority of deep learning instances in the world today are done locally on a single computer. Still, for large organizations that use data centers around the world, the lack of cloud hosting could dissuade them from considering TensorBoard as an option.

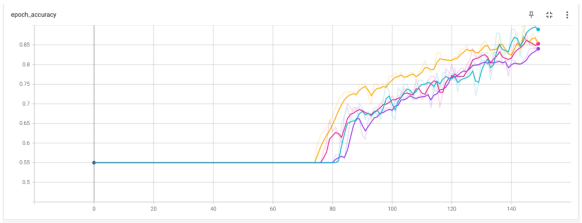

Figure 8: TensorBoard Accuracy vs. Epoch plot

Finally, in our experience, it was necessary to downgrade the google authentication package which TensorBoard relies on from 2.6.4 to 1.3.0 before TensorBoard was able to display visualizations correctly. We weren't entirely sure the reason for this - perhaps TensorBoard had to revert a dependency because of a security flaw? Nonetheless, this required downgrading a package in the python environment, which we both agree is not a trivial task especially for a new developer who simply wished to visualize a basic deep learning task. This is opposed to the extremely intuitive "plug and play" nature of Weights & Biases.

3 Evaluation

3.1 Evaluating the Neural Network

There are several metrics by which we can evaluate a machine learning model of any form. Perhaps the most intuitive metric is accuracy on a dataset, and this is exactly the metric we aimed to optimize our neural network on. To do this, we first trained our model with our training examples, and then classified the exact examples that we trained with. This classification returned predicted labels which we compared with our true labels to obtain an accuracy score. Our accuracy at the end of 150 epochs averaged around 87% on the training set, which we were extremely satisfied with, given that our vocabulary size (50) was relatively low to avoid overfitting. Additionally, 150 epochs was chosen because similar to vocabulary size, high values tend to overfit, so 150 seemed like an adequate sweet spot for accuracy optimization.

There is a caveat to this 87% accuracy. It is unfortunately the case that we both fit our model on the training set and trained on the training set. This was done for feasibility; as only a two person team, it was not feasible to obtain thousands of hand labeled training examples and testing examples. This of course means that our accuracy is somewhat inflated because intuitively our model should perform well on the dataset used to train it. Still, the practive of training and testing on the dataset is used commonly in industry, so we reasoned that for our dummy example to demonstrate the toolkits, it would suffice to reuse the training set. Despite this, we are still extremely satisfied with an 87% training accuracy.

3.2 Evaluating Weights & Biases

After performing sentiment analysis with the Weights & Biases Visualization tool, the model was able to output an efficient accuracy score with a set of statistics to accompany it. This result was due to how user-friendly the tool was in terms of utilizing it to perform the natural language process task. More specifically, the ease of use the tool presents allowed our team to complete multiple tasks with rarely any difficulty.

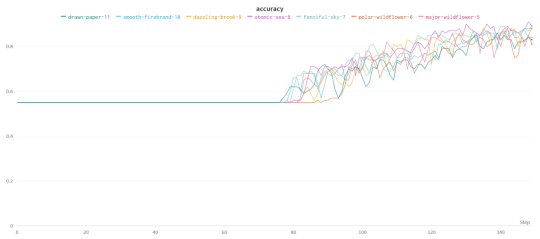

Figure 9: WandB Accuracy vs. Epoch Plot

Additionally, other aspects of the process were displayed using other features of the visualization tool, such as the memory access and GPU usage. The additional data statistics were able to be produced since the extensibility of the Weights & Biases tool allows users to generate graphs for different purposes. Since the tool works with any deep learning library, we were able to input various data points for the model and tool to work with. As such, the usability of the tool made it easy to perform sentiment analysis on the dataset and generate a visualized graph of the accuracy score.

The usability of the Weights & Biases visualization tool enabled our team to complete multiple tasks for the purpose of sentiment analysis. To be more specific, the tool allowed us to interact with the model while it was performing the task. This meant that we could interrupt the dataset evaluation to insert another statistic for more accurate data if we desired. As such, we were able to generate statistics that presented the system and model process during the sentiment analysis. This allowed us to make observations that stood out to us based on our initial assumptions before using the tool. In addition, the tool's interactivity allowed us to observe any small errors made in developing the models. From this, we smoothly completed the sentiment analysis to compose a result for our initial problem.

From a software engineering perspective, the Weights & Biases visualization tool was an excellent tool for completing sentiment analysis. The level of ease that comes with its usability allows the tool to be used for various purposes. As a result, non-programmers will easily understand how to use the tool from the user-friendly GUI of the tool. For people experienced in the field, the tool has a wide variety of visualization templates for more types of data evaluation. They will be able to get a sense of the level of accuracy their models will produce based on interaction made with the model. Additionally, they can monitor their system to make note of any odd occurrences during the evaluation process. With that said, the Weights & Biases visualization tool was efficient in completing our sentiment analysis due to its functionality for both non-programmers and programmers alike.

3.3 Evaluating TensorBoard

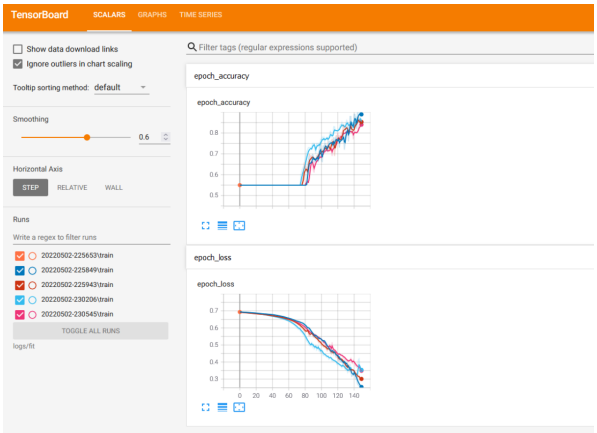

Figure 10: TensorBoard dashboard displaying plots

Although the TensorBoard machine learning tool produced an accurate result, the usability of it was not as efficient. To start, the package required a downgrade for the built-in Google authentication to operate on our local machines. Since the tool uses this particular verification field, it was strenuous for our team during the initial setup. As a result, there was a need for a large amount of customization to get the tool functioning properly in an optimal fashion. Additionally, the level of usability was low due to the lack of interaction with the model that integrated the TensorBoard tool. More specifically, the tool only analyzed the dataset, meaning there was no need for user interaction with the model. However, there was more extensibility present within the tool that allowed us to properly utilize its

functions.

Despite the limitations set by the usability of the TensorBoard tool, there was some extensibility present. To be more specific, there were a large amount of data plot templates available for generating data like the accuracy score from sentiment analysis on our dataset. As a result, we were able to record the development process of the model during its training with our dataset.

Though it only works with the TensorFlow library, it was able to train and test accurately for our data representation. As such, multiple plot graphs were recorded to observe the accuracy rate of the model each time a new point of data was added to the dataset. With the integrated library and vocabulary size, the extensibility made it possible to accomplish this. But, as a standalone visualization tool, we did not find TensorBoard to be extremely user friendly.

As a visualization tool for tasks like sentiment analysis, TensorBoard is efficient in completing it. However, for non-programmers and programmers unfamiliar with machine learning, this tool is not as proficient as other tools. Due to the extensive optimization needed to get the tool working, users at a beginner level will have a hard time understanding how to initially set it up. Additionally, downgrading the package on some machines might mess with their internal systems, making its usability and functionality worse. As a result, the ease of use factor when considering which tool to use for multiple purposes is lower in comparison to tools like Weights & Biases. The extensibility of the tool is excellent for users to generate data plot graphs, but it is restricted to using one library. As such, the TensorBoard visualization tool is not a user-friendly tool for aspiring software engineers to use for machine learning.

4 Discussion

We now begin a light discussion on observations related to the process of obtaining, labeling, and ultimately using data. While compiling the central dataset, there were issues with gathering posts related to the incident to use as data for it. For example, the initial shock value of the event did not last long, meaning that its relevance on social media waned very quickly. Additionally, a majority of the related posts had a neutral sentiment towards the problem, which would make it difficult for the model to accurately classify when performing sentiment analysis. Despite these obstacles, there were interesting observations made about valid posts for the data. More specifically, Twitter posts that had high amounts of engagement were in favor of Chris Rock being justified whereas ones with lower amounts of engagement favored Will Smith as the justified party. Despite minor obstacles in creating the dataset, the process of doing so was completed for the functionality and readability of the data to be adequate for the sentiment analysis process.

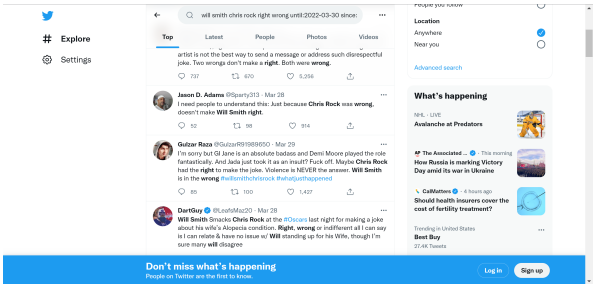

Figure 11: Tweet Relevance waning quickly

In terms of observations related to the neural net, we noticed that while the model was not particularly sensitive to large vocabulary sizes, large epochs caused a high degree of overfitting. It is hard to tell whether the model was overfitting, or just performing exceptionally well on classifying the training set, but nonetheless a moderate amount of epochs garnered a high enough level of accuracy for our purposes. Additionally, it is worth noting that our neural net only included 4 layers, making it a relatively simple model in the grand scheme of how complex neural nets can become. Even still, the fact that with 4 layers we were able to achieve such a high accuracy is a testament to how efficient correctly tuned neural nets can be. It is o belief that our adapted neural net included in our Github repository[1] could be used in a learning setting as a "Hello, world" example for neural net development.

Finally, we remark on observations we made while comparing the visualization toolkits. The biggest observation is that we had an overwhelming preference for Weights & Biases in total. As relative beginners to visualizing neural net results (despite experience in building neural nets), we really liked the fact that we could halt the model while not being local to the machine that is currently training

a model. Additionally, cloud hosting is too convenient of a factor to consider continuously local host results especially if we are working in a group which lends itself well to cloud hosting.

We overall present Weights & Biases as a suitable deep learning visualization toolkit for experienced and non-experienced programmers alike. Its ability to host results in the cloud, and provide analytics for the system in which training is running on is simply too convenient to pass for any other tool which requires a large amount of customization before reaching usability similar to Weights & Biases.

References

[1] Jarred Bettencourt and Jeffrey Tran. *Sentimental Smacking: An ML Toolkit Exploration*. Mar. 2022. URL: https://github.com/jbettencourt10/Sentimenal-Smacking.

[2] Lukas Biewald. *Weights amp; Biases – developer tools for ML*. 2018. URL: https://wandb.ai/site.

[3] Andrew R Chow. *The history that led to Will Smith slapping Chris Rock at the 2022 oscars*. Mar. 2022. URL: https://time.com/6161372/will-smith-chris-rock-oscars-2022/.

[4] Jack Dorsey. Mar. 2006. URL: https://twitter.com/.

[5] Google. *Text classification with an RNN and tensorflow*. Apr. 2022. URL: https://www.tensorflow.org/text/tutorials/text_classification_rnn.

[6] *What is sentiment analysis?* URL: https://www.twilio.com/docs/glossary/what-is-sentiment-analysis.