Zaid Abdulrehman

Professor Jesse Thomason

Csci 499 Natural Language Processing

13 October 2022

Hw #2: Skip-gram Model

How To Run

https://github.com/zabdulre/CSCI499_NaturalLanguageforInteractiveAI/tree/main/hw2

Please be sure to add the data directory before running. Please note, the context length refers to the total of left and right context combined. datadirectory2 stores the implemented model. datadirectory stores the short context, extra credit, model

python train.py --downstream_eval --analogies_fn analogies_v3000_1309.json --data_dir books/ --len_context=10 --learning_rate=0.01 --embedding_dim=50 --outputs_dir=datadirectory2

Report

I have implemented a skip gram model which takes left and right context in order to predict the context of a given word. I chose to implement a skip-gram model over a continuous bag of words model because my understanding is that skip gram models are more common in practice. Another reason why I chose this model is because I wanted to better understand the details of producing a context from a word. When learning about language models, I found it much more intuitive to calculate a word given the context. So, my motivation has been to consider the components which make reliably predicting a context given a word possible.

Implementation Decisions

Implementing the skip gram model has required me to make implementation decisions regarding extracting word and context pairs, memory optimizations, and custom metrics. In extracting word and context pairs, I made the decision to limit which words are part of the final dataset. During my initial implementation, I added all words, including the start, end, and padding tokens to the dataset. This introduced two problems. Firstly, when training on all 30 books, there was a total of 300 million word and context pairs generated using this method. This led to a major slow-down in training time and a huge increase in memory consumption. With this method, I was unable to complete an epoch on the full dataset. Furthermore, the program required 50 GB of memory to load the dataset, hindering performance further. Secondly, a majority of the pairs added to the dataset using this method were either, start or end tokens with only padding as left or right context respectively, or padding tokens with only padding tokens as bidirectional context. I hypothesized that by training on such examples, I would hinder the model's ability to learn from the words with adequate context. So, to avoid the 2 issues, I disabled the automatic padding feature and only added a word context pair to the dataset if no extra padding was required.

Memory optimization was required in order to train on all 30 books and webpages in the data set. While changing which words were added to the dataset reduced the size of the dataset from 300 million to only 3 million, further memory optimization was required to be able to more extensively tune hyperparameters and calculate accuracy. The first implementation decision which allowed me to optimize memory was storing vocabulary indices only during the encoding step. Instead of using a multi-hot vector of the same length as the vocabulary to store context, it

was advantageous to store a vector of the same length as the context with vocabulary indices only. During training, the vocabulary indices would then be converted to a multi-hot vector so that they may be used to calculate loss. In this way, only tensors of size batch * vocabulary_size would be stored in memory, as compared to tensors of size words * vocabulary_size. This optimization allowed memory requirements to drop from nearly 10 GB to below 1 GB. The second implementation decision which I made to improve memory was to introduce a rolling-count of accuracy. By calculating the accuracy at each iteration, it became unnecessary to store the large prediction and label vector at each time step. With this optimization, memory usage remained constant.

I implemented my own accuracy metric for this task in order to account for the multi-label, multi-classification output of the skip-gram model. The initial method of measuring accuracy did not account for prediction errors, such as predicting too many words. Furthermore, it allowed low confidence predictions to affect performance metrics to a great extent. The new accuracy metrics prevents low-confidence predictions from being considered by only selecting the top k logits as 1, where k is equal to the context length. Only selecting the top k logits also prevents placing high confidence on a larger number of words from being advantageous. At each iteration, the accuracy is calculated by dividing the number of words which the model predicted correctly by the context length. This total is then averaged at the end of each epoch.

Model Configuration

Finding the best model configuration and hyperparameters involved testing mostly on the in vitro task. I considered configurations with an Adam optimizer and a Stochastic Gradient

Descent optimizer and tested large and small embedding dimensions, vocabularies, context lengths, and batch sizes.

I chose a base configuration which could be run quickly and began my search. First, I tried various vocabulary sizes. Using a vocabulary size of 1500 and 2000 improved in vitro performance, but caused missing key errors in the in vivo task. The default vocabulary size of 3000 maintained a comparable in vitro accuracy and did not cause any missing key errors during the in vivo task. So, I decided to use a vocabulary size of 3000.

The batch size was based on my goal to be able to run the model many times during testing. Accordingly, I increased batch size from 32 to 128. This lead to a minor decrease in training time without a change in performance. Since the decrease in training time was under 1 minute, I did not experiment with larger batch sizes and chose the batch size to be 128.

Next, I chose an appropriate context length. I first tried a context length of only 5 (3 context words to the left of the target, and 2 to the right). While this allowed me to preprocess the training data quickly, it led the model to have an accuracy of less than 10%. I experimented with a context size of 8 and noticed a significant increase in processing time for the training data. I also noticed an improvement in accuracy. I decided on a context size of 10 (5 left of the target and 5 to the right). A context size of 10 allowed for high quality training examples from the dataset, while avoiding too many examples from being cut from the dataset due to a lack of context. 2.7 million examples with 10 context words are present in the dataset. With access to a large amount of data, it might be beneficial to consider an even larger context size.

Finally, to decide between Adam and SGD, I conducted multiple trial runs with both configurations. Adam had higher in vitro and in vivo performance. SGD showed a consistent

increase in accuracy per epoch, but ultimately did not surpass the Adam configuration, likely due to compute-related time constraints. SGD was trained for the longest time period, at 8 minutes per epoch for 10 epochs. During training, SGD showed a consistent improvement in validation accuracy for the in vitro task. Accuracy improved at an average of 1.8% per epoch, consistently. It is most notable that the SGD configuration showed consistent improvement even from the 9th epoch to the 10th epoch. As a result, I believe that an SGD model, if trained for a longer period of time, has the potential to achieve an accuracy above the Adam configuration's best of 29%.

Results

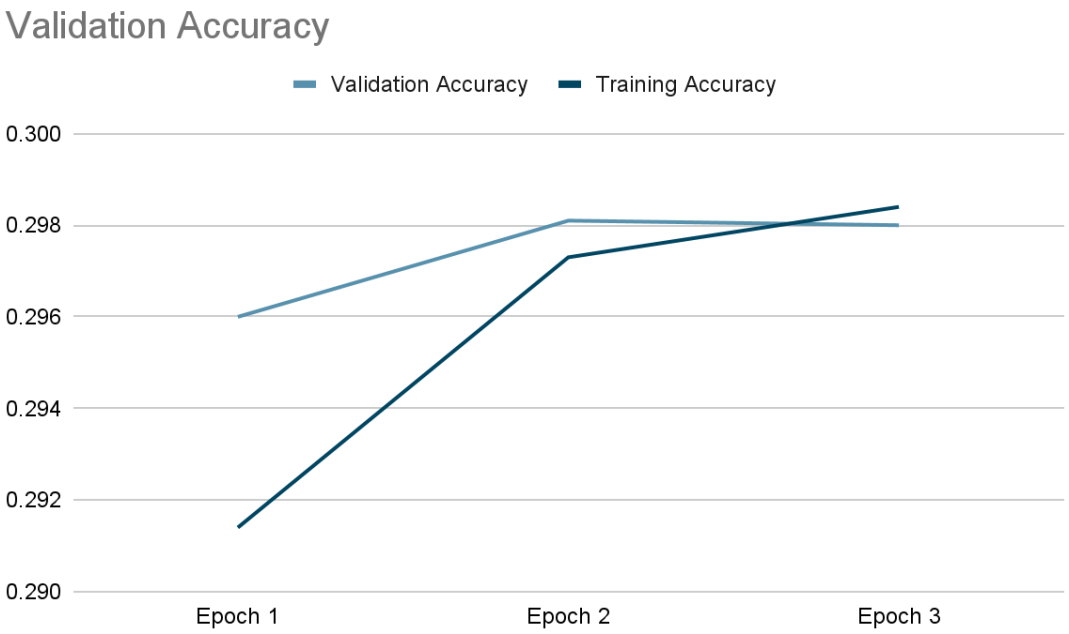
The best in vitro performance and in vivo performance was achieved by the Adam optimizer model with an embedding dimension of 50.

Validation Accuracy	Train Accuracy	Downstream Exact	Downstream MRR	Downstream MR
0.2980	0.2984	0.0130	0.0288	35

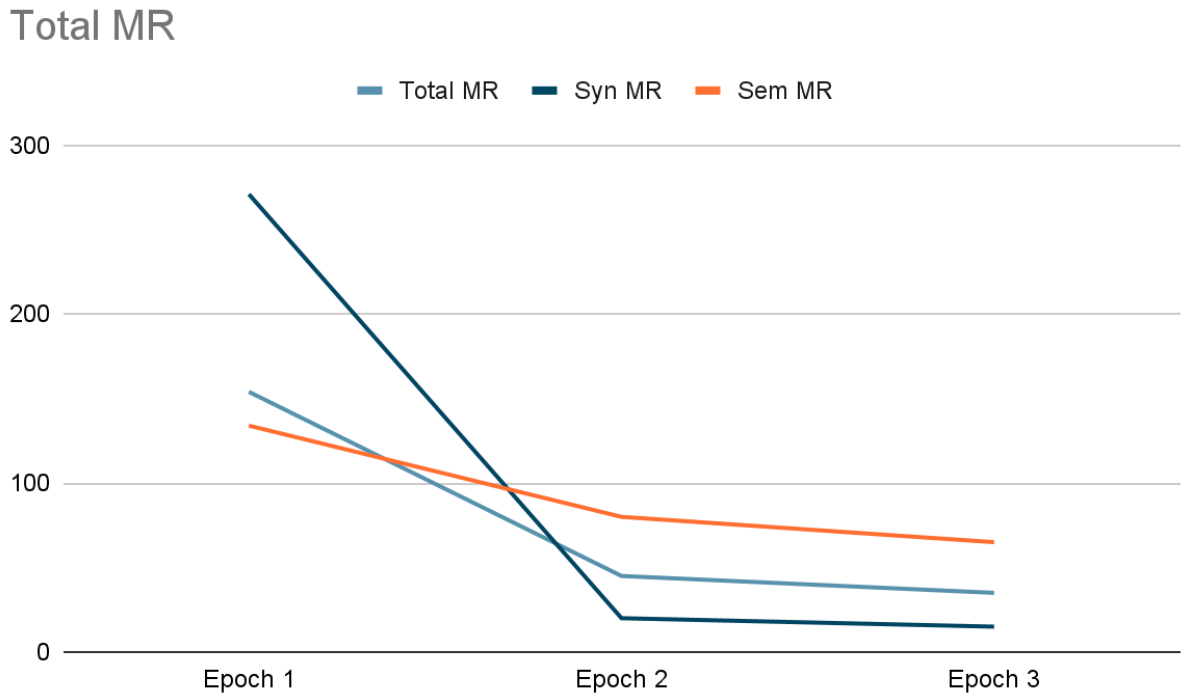
Syn Exact	Syn MRR	Syn MR
0.0324	0.0669	15

Sem Exact	Sem MRR	Sem MR
0.0062	0.0155	65

In vitro performance between the same model with an embedding dimension of 100, was nearly identical. However, an embedding dimension of 50 has superior in vivo performance.



Training for more than 3 epochs resulted in a slight dip in validation accuracy. Validation accuracy peaked after the 2nd epoch. Downstream performance peaks after the 3rd epoch.



Analysis

For the skip gram model, the in vitro task, or the task which the model is trained on, is predicting a context given a word. The model is given a word index and outputs a set of logits which correspond to words in a vocabulary. The implementation achieves a validation accuracy of 29.8%. While this metric is under 50%, it is able to demonstrate the skip-gram model's ability to learn from the dataset. In predicting the context, the chance of a false positive is extremely high at 1/(vocab_size - context_length). So, an accuracy of 30% represents the ability of the model to very clearly discern from context words which do not correspond to the given word. Furthermore, words, even though they may be associated with a particular context, are not always used in the same context. This means that the chance a word appears in exactly the same context twice, especially if the total context length is over 3, is very low. In order for a model to achieve a high score with a context length of over 10, it must overfit to the data.

The in vivo performance of the model clearly demonstrates the model's ability to learn correct word embeddings. The in vivo task measures the model's ability to cluster words based on semantic and syntactic relations. For example, if two words have a causal relationship, then the corresponding vector arithmetic should result in a prediction of one of the words from the other. There are 3 metrics by which both are measured, exact matches, mean reciprocal rank, and mean rank.

The exact matches metric is used to convey the ratio of words which are perfectly grouped by the model's embeddings. It is calculated by keeping track of the number of words which have the lowest cosine similarity and are also the top syntactic or semantic choice, which is then divided by the total number of words considered. The model achieves a total exact

matches score of 0.013. This score clearly demonstrates the skip-gram model's ability to learn because it is considerably higher than the expected value of random selection, which is 1/3000 = 0.0003. It is worth mentioning that the exact matches metric can underestimate model performance. It is assumed that each match is the only exact match. As such, if the model correctly identifies a different word vector which is a match of equivalent quality, the model may be penalized.

The Mean Reciprocal Rank and the mean rank measure the correctness of the ranking which the model produces. It rewards the model based on the ranking of the word which the model identifies correctly. If a word is of higher semantic similarity, for example, then the model is assigned a higher score based on the rank of the word which was correctly identified. Using this method, the model is able to gain credit for grouping words which are not necessarily the top match. Our implementation scores 0.0288, which is considerably higher than the expected value of a random model. An assumption made in using MRR is that there are always k correct words. This can lead to an overestimate of performance because the model can be rewarded for clustering words which do not have any meaningful semantic or syntactic significance.

Extra Credit

In my implementation of the skip-gram model, I have added support for changing context length. The user is able to input the context length as an argument.

In class, we hypothesized that a language model with a shorter context will be able to learn syntactic relations better than semantic relations. I have tested the implemented model with a shorter context length of 4 to test this hypothesis (the total of left plus right context is 4).

Syn Exact	Syn MRR	Syn MR
0.0324	0.0905	11

Sem Exact	Sem MRR	Sem MR
0.0083	0.0195	51

It is clear that the shorter context model performs better on syntactic tasks. Both the exact performance and the mean rank performance is more than triple the semantic performance. Furthermore, the shorter context model has a significantly better syntactic exact metric and mean rank in comparison to the longer context model. The semantic performance of both models is surprisingly comparable, with both models performing similarly. However, it is important to note that the shorter context model has a lower validation accuracy of 26%. On syntactic relations such as embedding superlative information, the short context model is able to perform very well with an exact metric of 0.2857 and an MR of 3 for a test set of 3 words. The longer context model has an exact metric of 0 and and MR of 44. The best syntactic example from the longer context model is plural nouns with an exact score of 0.0748 and an MR of 9 tested on 107 words. The shorter context model has an exact score of 0.0187 and an MR of 12 tested on 107 words, meaning that the longer context model is better in this category. However, as a whole, our hypothesis that shorter context models perform better on syntactic tasks stands.