Exploring additive and multiplicative similarity measures in multilayer perceptrons.

Santiago Aranguri aranguri@stanford.edu

March 19, 2019

1 Introduction

In this report, we are going to analyze the advantages and disadvantages of different ways of measuring the similarity between vectors in an embedding space that represents real information. Computing a similarity function is crucial for classifying objects. One can think of a multilayer perceptron (MLP) that classifies if an input image is a cat or dog as having stored in the weights of the last hidden layer an ideal cat an ideal dog. Then, the MLP can compute the similarity between a transformation of the input and the ideal cat and the ideal dog. The category with the highest similarity will be the category of the input. More generally, having a good similarity measure enables us to solve most of the machine learning problems, just by transforming the representation of the input to a convenient space and then measuring the similarity between the input and previously defined categories. All the code for this project is available at https://github.com/Aranguri/cs230

1.1 Related work

Attention mechanisms make use of similarity measures to retrieve information from a memory matrix. There is a difference in the type of attention mechanism in the literature that is the same difference we are going to study in this report. That difference is whether we have additive [1] or multiplicative [2] interactions.

This report also uses the Dynamic Memory Network [4] for one of the experiments. In the paper for the Dynamic Memory Network, the authors define a similarity function but they don't go into further details about it and they don't offer a justification for that election. However, in this work, we are going to analyze that election for the similarity function.

2 Datasets

2.1 Clustered Vectors Dataset

For the first experiment, the dataset is synthetic. We generated $m = 10000$ vectors drawn uniformly where $v \in [-1, 1]^n$ with $n = 100$. Then, for each vector, we created 2 vectors by adding to it random Gaussian noise with mean 0 and variance 1. We thus have m clusters where each cluster has 2 vectors. Given two different random vectors, the correct output is 1 if they come from the same cluster and 0 if they don't. The metric used for this dataset is the accuracy in correctly classifying whether a set of vector pairs were from the same cluster or not. The accuracy is the fraction of correctly classified vector pairs over all classified vector pairs.

2.2 bAbi dataset

The bAbI dataset[3] is a set of tasks for natural language processing. Although it includes different types of tasks, in this report we are going to focus only on the two supporting fact task. That task consists of

receiving a set of facts and answering a question that required two facts from the list to be answered. The following is a simple example from the two supporting fact task.

```
   1 Mary moved to the bathroom.
2 Sandra journeyed to the bedroom.
3 Mary got the football there.
4 John went to the kitchen.
5 Mary went back to the kitchen.
6 Mary went back to the garden.
7 Where is the football?
Expected output: garden
```

3 Methods

3.1 Similarity measure

In this section we will explain the ways of measuring similarity used in the experiments below. A similarity measure is a function $sim(q, M)$ that receives a query vector q and memory vectors M and outputs a probability distribution P over M where each $p \in P$ represents the similarity between the q and $m \in M$. In general, $sim(q, M)$ has two steps. First, we compute the similarity $f(q, m)$ for all $m \in M$. Secondly, we normalize the probabilities and softly select the maximum value by applying a softmax, that is,

$$sim(q, M)_i = \frac{exp\{f(q, M_i)\}}{\sum_j exp\{f(q, M_j)\}}$$

The intriguing part lies on $f(q, m)$. Let's consider different ways we can implement it.

3.1.1 Additive interactions

We define an additive similarity function as $f(q, m) = ||q - m||_1$.

3.1.2 Multiplicative interactions

We are going to use the element-wise product to measure multiplicative interactions. The element-wise product is defined as $f(q, m) = q \odot m$.

3.1.3 Parametric interactions

The models we used so far to measure similarity don't have trainable parameters. However, trained parameters enable the model to capture specific ways of measuring similarity that could be dependent on the task.

An example of using trained parameters is $f(q, m) = MLP(q, m; \theta)$where MLP is a multilayer perceptron with only one unit for the output and θ is a set of matrices defining the weights of the MLP. Something interesting about this way of measuring similarity is that the interactions inside the MLP are additive.

3.2 Dynamic Memory Network

For the second experiment, we used a particular memory network called the Dynamic Memory Network (DMN). The details of the model are in [4], but the main idea is that we perform multiple passes through a transformation of the input (which in the case of bAbi is a set of facts; see description of the bAbi dataset above), and after each pass we compute a memory vector that is also taken as input for the next pass. After finishing the passes, we compute the answer based on the last memory state. For each step that we compute the memory state, we need to know what facts to consider, so we have a gate for each fact. The value of that gate is given by a similarity function between the fact, the question, and the previous memory state. That similarity function is what we are going to study in the experiments section.

4 Experiments

4.1 Multiplicative interactions experiment

The task for this experiment was to compute whether two random vectors drawn from the "clustered vectors dataset" described above are from the same cluster or not.

We are going to test whether multiplicative interactions are useful for this task by using two MLPs. MLP_1 receives the two random vectors as input and has one output unit representing the similarity between the two random vectors. MLP_2 receives as input the two random vectors and the element-wise product of the two vectors, and it has one output unit.

The hypothesis was that MLP_2 would reach a greater accuracy than MLP_1, given that the two MLP have only two layers with 300 units each layer because having two layers would imply that the $MLPs$ aren't complex enough to model multiplicative interactions. The activation function between the hidden layers is ReLU. And the last activation function is the sigmoid function. The learning rate used for these experiments was 1e-3 using the Adam optimizer, and we used a batch size of 200.

The hypothesis resulted to be true. Using a variance of 1 for the random Gaussian noise, MLP_2 reached an accuracy of 95% while MLP_1 couldn't capture underlying information from the input, getting an accuracy of around 50%.

We also wanted to look at the performance of the two MLPs with different variances. The higher the variance, the more difference between the vectors in the same cluster. Thus, the harder it is to distinguish whether two vectors are from the same cluster or not.

For this experiment, we used a variance of .5 for the random Gaussian noise. MLP_2 reached an accuracy of around 90% after 200 batches, where the batch size is 100 (that's a total of 20000 examples.) On the other hand, MLP_1 is strictly inferior, the accuracy was always less than 90%, and the accuracy was generally around 60%. See figure 1 for more detailed information.

We also tried using a variance of 2 for the noise. Although neither of the two $MLPs$ reached a high accuracy, MLP_1 barely distinguished between the random vectors while MLP_2 achieved an accuracy of around 65%. See figure 2.

4.2 Dynamic Memory Network

For this experiment, we used the bAbi dataset and the DMN as described above. For the first similarity measure, we computed the values for the gates as follows.

$$z(c, m, q) = [c, m, q, |c - q|, |c - m|, c \odot q, c \odot m, c^T W^{(a)} q, c^T W^{(b)} m]$$
$$G(c, m, q) = (W^{(2)} tanh(W^{(1)} z(c, m, q) + b^{(1)}) + b^{(2)})$$

where c, m, and q are representations of the fact, memory state, and question respectively. All W's and B's are trainable parameters. G is the result of the similarity function. This structure is taken from the original paper for the Dynamic Memory Network.

The hyperparameters used were: learning rate 1e-4, batch size 64, embeddings size 256, all hidden layers had 512 units, and the hidden layers in G also had 512 units.

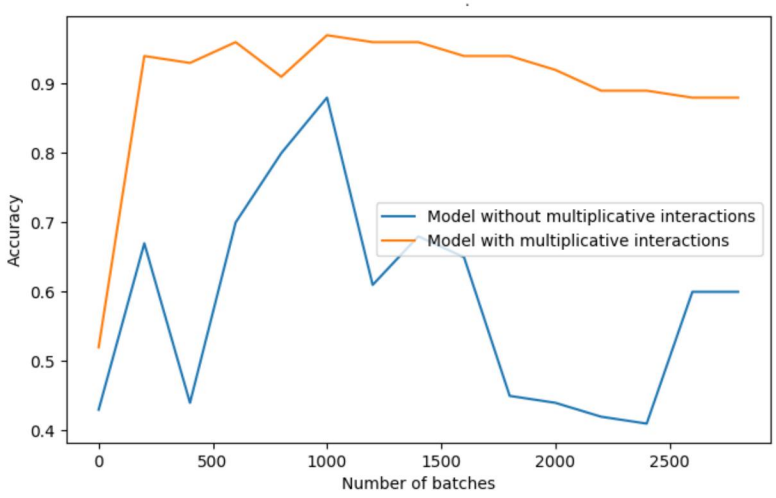

Figure 1: Computing similarity with and without multiplicative interactions. The variance in the random gaussian noise added was .5

After 3300 batches (ie around 200k examples), the DMN reached an accuracy of 99% in the validation set. However, when we removed the multiplicative interactions from z, it took 4900 batches to reach an accuracy of 99%. In around 8000 batches, the accuracy in the validation set for this experiment reached 100%. Removing the multiplicative interactions from z means that z ended up as follows

$$z(c, m, q) = [c, m, q, |c - q|, |c - m|]$$

We also tried using z as follows

$$z(c, m, q) = [c, m, q, c \odot q, c \odot m]$$

This values for z gave us a comparable experiment to the one we did with additive interactions, but in this case, we had multiplicative interactions. We say they are comparable because we don't add parameters as we would add with $c^T W^{(a)} q$. Surprisingly, this experiment converged to an accuracy of 13% in the validation set. This suggests that the model that only has multiplicative interactions couldn't capture important underlying information about the training cases.

5 Conclusion

The conclusion from the multiplicative interactions experiment is that the model that receives the multiplicative interactions as input always outperforms the model that doesn't receive the multiplicative interactions. Also, for some values of the variances (eg, variance equals 2), the model that doesn't receive the multiplicative interactions can't get better than doing random guesses. In the language from the methods section, we could say that MLP_1 only models additive interactions of the input. However, MLP_2 is able to use both additive and multiplicative interactions. We can conclude that to compute the similarity between noisy vectors, being able to model multiplicative interactions offers an advantage.

With respect to the dynamic memory network, the efficacy of the multiplicative vs additive interactions was the reverse. In particular, the DMN still converged to a perfect solution even when we removed multiplicative interactions. However, when we removed the additive interactions, the model couldn't capture most of the important information, resulting in a very low 13% accuracy on the validation set.

Both experiments suggest that depending on the problem one is trying to solve, using different interactions between the input data can prove useful. However, as we saw for the first experiment for the

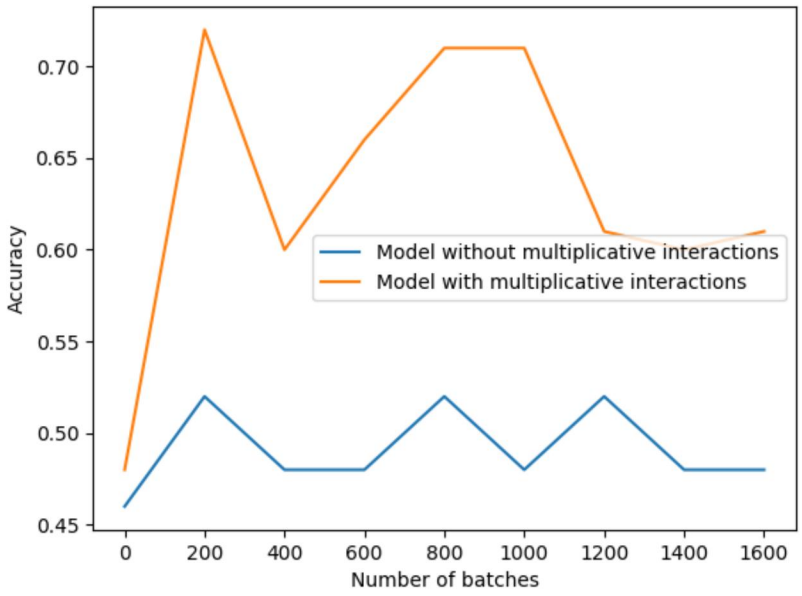

Figure 2: Computing similarity with and without multiplicative interactions. The variance in the random gaussian noise added was 2

DMN when we removed the multiplicative interactions, the change could be very small (just delaying the arrival of a perfect solution by some batches,) or the change could be huge (going from 13% to 100% accuracy) when we changed from multiplicative to additive interactions for the DMN.

References

[1] Sutskever et al. Neural Machine Translation by Jointly Learning to Align and Translate

[2] Luong et al. Effective Approaches to Attention-based Neural Machine Translation

[3] Weston et al. Towards AI Complete Question Answering: A Set of Prerequisite Toy Tasks.

[4] Kumar et al. Ask Me Anything: Dynamic Memory Networks for Natural Language Processing.

[5] Hochreiter and Schmidhuber LONG SHORT-TERM MEMORY.

[6] Taeuk Kim https://github.com/galsang/BiDAF-pytorch.