CS230

The Unordered Transformer For Web-page Classification

Ramy El Garawany
garawany@stanford.edu

Abstract

Word embeddings are useful tools for working with textual data. Most state-of-the-art[1] deep NLP networks work on ordered sequences of embedding vectors. However, it is unclear how to train models with *unordered* sets of embedding vectors. This paper investigates the usage of Transformers[2] without positional-encoding as a way of handling such sets.

1 Introduction

This project attempts to improve an existing classifier that classifies web-pages into one or more of ~700 product categories. The product categories form a predefined taxonomy. Furthermore, the web-page must be of a commercial nature in order to be classified in a product category. This provides an extra challenge to the classifier, compared to simply sorting web-pages into categories.

The classifier receives as input unordered sets of embedding vectors. The embedding vectors represent values in an embedding of approximately 13 million words. Since the order of the vectors is mostly random, running a regular recurrent network is not ideal, since there are no temporal dependencies between words. The model might simply over-fit on the order that happens to exist in the training data, and fail to generalize on the validation set.

We use a slight modification of the Transformer network to solve this problem. The Transformer model is a state-of-the-art network that outperforms recurrent neural networks in many sequence modeling problems.

2 Related work

The simplest solution when dealing with an unbounded collection of embedding vectors is to use a bag-of-words approach[3]. This is what the existing classifier currently uses. In this case, the embedding vectors are simply averaged to produce a single value. Various methods, like Paragraph Vectors[4][5], attempt to ameliorate this problem by instead learning fixed-length representations of pieces of consecutive words in an unsupervised fashion. While this method does significantly improve the accuracy over bag-of-words, it is not applicable to our model, since we do not have control on the embedding space used for the input features. Those embedding vectors are derived from an incrementally-evolving unsupervised machine learning algorithm. Furthermore, our data-set is simply to small to train a separate embedding that covers the entire input vocabulary space.

One promising solution is to use a modified attention-based approach in order to learn an input order-independent hidden state using an LSTM[6]. However, the model sometimes receives up to 100 embedding vectors, and a recurrent model would be too slow during prediction in our latency-constrained environment. In recurrent networks, the model cannot distribute and parallelize the computation across the various words.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Yet another potential solution is to use convolutional neural networks[7]. However, an inherent bias towards the training set's ordering will still be present. Perhaps convolutional neural networks can be used with data augmentation (e.g., randomly permuting the embedding vectors) in order to aid generalization.

3 Data-set and Features

All input features are derived from the same embedding, which is trained by separate external models. There are two different yet related categories of features: "words", and "clusters". "Words" are simply words extracted from the web-page, after filtering unimportant page elements and cleaning up the HTML source. "Clusters" are features learned by an unsupervised model[1]. The cluster model groups together words that are commonly found together into pre-defined "clusters". However, the number of clusters is dwarfed by the number of words in the embedding.

The embedding vector is 100-dimensional. The number of entries in the embedding is approximately 13 million. The embedding covers many different languages. However, the model is trained on only 9 major languages.

Furthermore, the input data is split into page-level words and clusters, site-level words and clusters, and url-level words.

The labels are in the form of a sparse list of category ids. If the web-page is not commercial in nature, the label is a special "none" value. The labels are then transformed into a 773-dimensional one-hot vector to be used with a sigmoid cross-entropy loss.

There are approximately 2.5 million training examples, and 220 thousand validation examples.

4 Methods

The Transformer network is composed of a stack of "encoders", followed by a stack of "decoders". Each encoder and decoder contain sub-layers that perform what is called "self-attention". Self-attention is simply attention performed on each layer's own inputs (rather than the traditional attention models[14], where only the decoder gets to attend, and only on the encoder's outputs).

Encoders are composed of a self-attention sub-layer, followed by a simple shallow feed-forward network. The encoders operate process each input independently, and simply feed it to the next layer in the stack. Decoders are almost identical to encoders, except that they contain an extra "encoder-decoder" attention layer after the self-attention step. The encoder-decoder attention allows it to attend on all the values of the final encoder's outputs.

Attention is computed by linearly projecting the inputs into "keys", "queries", and "values". A score is then computed by taking the dot product between each key and query, and normalizing it with the inverse square root of the model dimension, d_k. Finally, the score is used to linearly combine the values of all the inputs. This essentially allows the model to learn what to focus on given each input word.

$$\text{Attention}(Queries, Keys, Values) = \text{Softmax}(\frac{Queries \cdot Keys^T}{\sqrt{d_k}}) \cdot Values$$

In the case of self-attention, the sub-layer's input is used to compute the keys, the queries and the values. In the decoder-encoder attention, however, the previous decoder's output is used to compute the queries, while the keys and values are computed from the final encoder's outputs.

Finally, one crucial component of the Transformer model when working with ordered sequences is the addition of positional encoding. Positional encodings allow the encoders and decoders to reason about the relative ordering of the words, without which the model would not be able to learn any temporal dependencies.

The small (yet important) change that we do, is to simply drop the positional encoding! This allows us to retain the benefits of the Transformer network, while removing any temporal dependencies

[1] `http://bayesian-networks.blogspot.com/2014/11/case-study-googles-rephil.html`

Table 1: Transformer and Baseline Metrics Comparison

	Macro F-Score	Macro Precision	Macro Recall	P@5	R@5	Micro F-Score	Micro Precision	Micro Recall
Transformer Model	78.5	84.1	73.6	58.8	89.8	88.1	90.6	85.8
Baseline Model	77.5	81.4	74.1	57.3	87.2	85.81	87.6	84.1

between words. One other change is that since the output of this model is not a sequence, the decoder is significantly simplified by only having it output one value. Since there are no decoder outputs to attend to, the decoder self-attention sub-layer is also removed.

One notable side-effect of having the Transformer operate on each word independently, is that the computation can be performed in parallel for each word. This allows the model to be trained in a fraction of the cost of recurrent networks.

For a more in-depth explanation of the Transformer network, Harvard's NLP group provides an excellent guide, along with a sample implementation.[2]

5 Experiments/Results/Discussion

The model is trained and evaluated using Tensorflow[8]. It is trained on Google's Cloud TPU[9] in order to speed up learning.

The hyper-parameters of the Transformer model are mostly identical to the hyper-parameters given in the original paper. However, only 3 encoders and decoders are used, as opposed to the paper's original 6. More encoders/decoders did not seem to help, and only served to over-fit. Between each sub-layer is a dropout layer, followed by a residual connection[10] and layer normalization[11].

The model is composed of 8 attention heads, of size 64 each, for a total of 512 dimensions for each value. The same learning rate schedule is used as in the original paper: A linear warm-up of 4000 steps, followed by a decrease inversely proportional to the square root of the step number. The batch size used was 2048.

The existing model is used as a baseline. The existing model is a simple 3-layer feed-forward network, with hidden layer sizes of 1024 and 512. Between each hidden layer is a dropout layer with a dropout rate of 0.1. The model is trained with Adam[12], and a learning rate of 0.0002.

The primary metric used to evaluate the model is the macro F-score[13]. The macro F-score is simply the average of all 773 class F-scores. The focus on macro- vs. micro- metrics is justified by the fact that the training dataset is relatively imbalanced, with a larger number of training examples available for the more common classes. Therefore, the micro- statistics are potentially misleading if the model learns how to classify the common classes really well.

Precision@k and recall@k for $k = 5$ are also provided. This metric most closely matches the qualitative goal of the model: for a specific web-page, does the classifier accurately classify the 5 most important categories? After the 5 or so most important classifications, the rest of the classifications tend not to matter as much for the consumers of this model's classifications.

As seen in Table 1, the Transformer model outperforms the simple bag-of-words approach. Precision is especially higher compared to the baseline model.

6 Conclusion/Future Work

This project attempted to modify the Transformer model in order to use it with unordered sets of data. While it was shown to outperform the baseline model, it is unfortunately not a dramatic improvement as expected. The cost to evaluate the model is significantly higher than the simple baseline model. The fact that it was not able to take advantage of the entire input dataset, rather than an averaged bag-of-words approach, hints that perhaps the classification difficulty lies elsewhere. Perhaps the existing input features are simply not enough to accurately discern the various subtle differences between categories.

[2] `http://nlp.seas.harvard.edu/2018/04/03/attention.html`

One potential avenue is to rewrite the model pipelines to pass through the entire ordered set of webpage content. While it would significantly complicate the pipeline, it might perhaps offer benefits. Order might, after all, matter!

Furthermore, the baseline model hyper-parameters were carefully chosen after a fairly long parameter-space exploration. I did not have enough time to spend searching for better hyper-parameters. Better hyper-parameters might theoretically improve performance by a substantial amount.

Finally, due to the complicated architecture of the model, most of the time was spent trying to implement and debug the model. In hindsight, had the model been implemented earlier, more time could have been spent figuring out where the Transformer model was finding difficulty in classification.

References

[1] Devlin, J., Chang, M. W., Lee, K.,& Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (pp. 5998-6008)*.

[3] Harris, Z. S. (1954). Distributional structure. *Word, 10(2-3)*, 146-162.

[4] Le, Q., & Mikolov, T. (2014, January). *Distributed representations of sentences and documents. In International Conference on Machine Learning* (pp. 1188-1196).

[5] Dai, A. M., Olah, C., & Le, Q. V. (2015). Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998.*

[6] Vinyals, O., Bengio, S., & Kudlur, M. (2015). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391.*

[7] Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882.*

[8] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Kudlur, M. (2016, November). Tensorflow: a system for large-scale machine learning. In *OSDI* (Vol. 16, pp. 265-283).

[9] Cloud TPUs - ML Accelerators for TensorFlow. *Google*, cloud.google.com/tpu/

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).*

[11] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450.*

[12] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[13] Wu, X. Z., & Zhou, Z. H. (2016). A unified view of multi-label performance measures. arXiv preprint arXiv:1609.00288.

[14] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473.*