Long-Distance Dependencies don't have to be Long: Simplifying through Provably (Approximately) Optimal Permutations

Rishi Bommasani
Department of Computer Science
Cornell University
rb724@cornell.edu

A Implementation Details

We implement our models in PyTorch (Paszke et al., 2017) using the Adam optimizer (Kingma and Ba, 2014) with its default parameters in PyTorch. We split the dataset using a 80/10/10 split and the results in Table 2 (in the paper) are on the test set whereas those in Figure 1 are on the development set. We use ELMo embeddings (Peters et al., 2018)[1], for the initial pretrained word representations by concatenating the two 1024 dimensional pretrained vectors, yielding a 2048 dimensional initial pretrained representation for each token. These representations are frozen based on the results of Peters et al. (2019) and passed through a single-layer bidirectional LSTM with output dimensionality 256. The outputs of the forward and backward LSTMs at position i are concatenated and a sentence representation is produced by max-pooling as was found to be effective in Howard and Ruder (2018) and Peters et al. (2019). The sentence representation is passed through a linear classifier $M \in \mathbb{R}^{512\times 2}$ and the entire model is trained to minimize cross entropy loss. All models are trained for 13 epochs with a batch size of 16 with the test set results reported being from the model checkpoint after epoch 13. We also experimented with changing the LSTM task-specific encoder to be unidirectional but found the results were strictly worse.

B Efficiency Analysis

Model Size The changes we introduce only impact the initial preprocessing and ordering of the pretrained representations for the model. As a result, we make no changes to the number of model parameters and the only contribution to the model footprint is we need to store the permutation on a per example basis. This can actually be avoided in the case where we have frozen pretrained embeddings as the permutation can be computed in advance. Therefore, for the results in this paper, the model size is entirely unchanged.

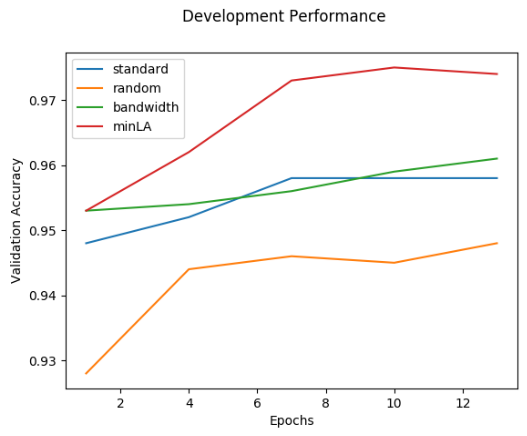


Figure 1: Development set performance for each ordering. Values are reported beginning at epoch 1 in intervals of 3 epochs.

Runtime The wall-clock training time, i.e. the wall-clock time for a fixed number of epochs, and inference time are unchanged as we do not change the underlying model in any way and the permutations can be precomputed. As noted in the paper, on a single CPU it takes **21** minutes to complete the entire preprocessing process and 25% of this time is a result of computing bandwidth optimal permutations and 70% of this time is a result of computing minLA optimal permutations. The preprocessing time scales linearly in the number of examples and we verify this as it takes **10** minutes to process only the subjective examples (and the dataset is balanced). Figure 1 shows the development set performance for each of the permutation

[1] Specifically, we use embeddings available at: https://s3-us-west-2.amazonaws.com/allennlp/models/elmo/2x4096_512_2048cnn_2xhighway/elmo_2x4096_512_2048cnn_2xhighway_options.json

types over the course of the fine-tuning process.

C End-to-End Permutations

In order to approach differentiable optimization for permutations, we must specify a representation. A standard choice that is well-suited for linear algebraic manipulation is a permutation matrix, i.e $P_\pi \in \mathbb{R}^{n \times n}$, where $P_\pi[i, j] = 1$ if $\pi(i) = j$ and 0 otherwise. As a result, permutation matrices are discrete, and therefore sparse, in the space of real matrices. As such they are poorly suited for the gradient-based optimization that supports most neural models. A recent approach from vision has considered a generalization of permutation matrices to the associated class of *doubly stochastic matrices* and then considered optimization with respect to the manifold they define (the *Sinkhorn Manifold*) to find a discrete permutation (Santa Cruz et al., 2017). This approach cannot be immediately applied for neural models for sentences since the algorithms exploits that images, and therefore permutations of the pixels in an image, are of fixed size between examples. That being said we ultimately see this as being an important direction of study given the shift from discrete optimization to soft/differentiable alternatives for similar problems in areas such as structured prediction.

References

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339.

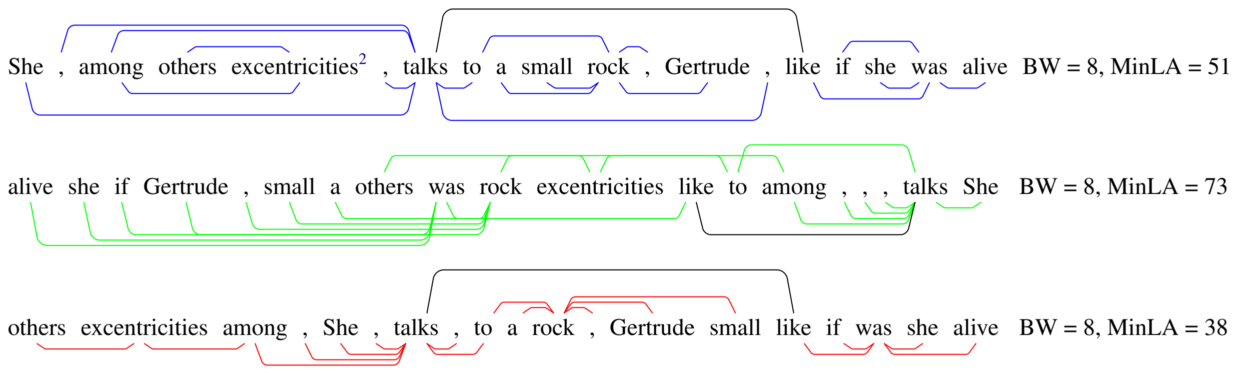
Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Matthew Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks.

Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. 2017. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3949–3957.



She , among others excentricities[2] , talks to a small rock , Gertrude , like if she was alive BW = 8, MinLA = 51
alive she if Gertrude , small a others was rock excentricities like to among , , , talks She BW = 8, MinLA = 73
others excentricities among , She , talks , to a rock , Gertrude small like if was she alive BW = 8, MinLA = 38

Figure 2: Addition example sentence with sentence permutations and overlayed dependency parses. Blue indicates the standard ordering, green indicates the bandwidth optimal ordering, and red indicates the minLA optimal ordering. Black indicates the longest dependency arc in the original ordering.