Rethinking Self-Attention: An Interpretable Self-Attentive Encoder-Decoder Parser

Khalil Mrini[1], **Franck Dernoncourt**[2], **Trung Bui**[2], **Walter Chang**[2], **and Ndapa Nakashole**[1]

[1] University of California, San Diego, La Jolla, CA 92093

khalil@ucsd.edu, nnakashole@eng.ucsd.edu

[2]Adobe Research, San Jose, CA 95110

{dernonco, bui, wachang}@adobe.com

Abstract

Attention mechanisms have improved the performance of NLP tasks while providing for appearance of model interpretability. Self-attention is currently widely used in NLP models, however it is difficult to interpret due to the numerous attention distributions. We hypothesize that model representations can benefit from label-specific information, while facilitating interpretation of predictions. We introduce the Label Attention Layer: a new form of self-attention where attention heads represent labels. We validate our hypothesis by running experiments in constituency and dependency parsing and show our new model obtains new state-of-the-art results for both tasks on the English Penn Treebank. Our neural parser obtains 96.34 F1 score for constituency parsing, and 97.33 UAS and 96.29 LAS for dependency parsing. Additionally, our model requires fewer layers, therefore, fewer parameters compared to existing work.

1 Introduction

Since their introduction in Machine Translation, attention mechanisms (Bahdanau et al., 2014; Luong et al., 2015) have been extended to other tasks such as text classification (Yang et al., 2016), natural language inference (Chen et al., 2016) and language modeling (Salton et al., 2017).

Self-attention and transformer architectures (Vaswani et al., 2017) are now the state of the art in language understanding (Devlin et al., 2018; Yang et al., 2019), extractive summarization (Liu, 2019), semantic role labeling (Strubell et al., 2018) and machine translation for low-resource languages (Rikters, 2018; Rikters et al., 2018).

Attention mechanisms provide explainable attention distributions that can help to interpret predictions. For example, for their machine translation predictions, Bahdanau et al. (2014) show a heat map of attention weights from source language words to target language words. Similarly, a self-attention head produces attention distributions from the input words to the same input words, as shown in the second row of the right side of Figure 1. However, self-attention mechanisms have multiple heads, making the combined outputs difficult to interpret.

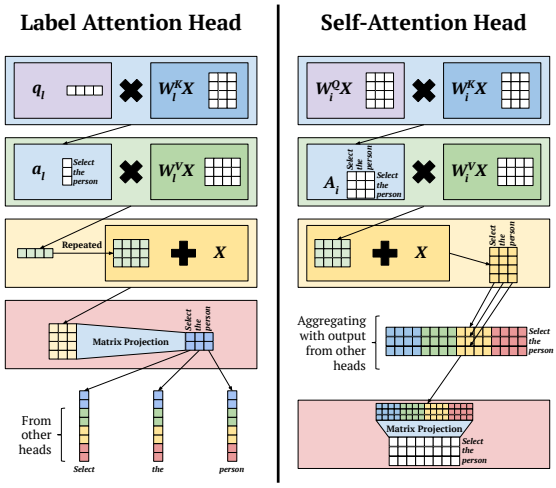

Figure 1: Comparison of the attention head architectures of our proposed Label Attention Layer and a Self-Attention Layer (Vaswani et al., 2017). The input matrix $\mathbf{X}$ contains the word vectors for the example input sentence "*Select the person*".

We hypothesize that label-specific representations can increase performance and provide interpretable predictions. We introduce the Label Attention Layer: a modified version of self-attention, where each attention head represents a label. We project the output at the attention head level, rather than after aggregating all outputs, to preserve the source of label-specific information.

To test our proposed Label Attention Layer, we build upon the parser of Zhou and Zhao (2019) and establish a new state of the art for both con-

stituency and dependency parsing. We also release our trained parser, as well as our code to encourage experiments with models that include the Label Attention Layer[1].

The rest of this paper is organized as follows: we explain the architecture and intuition behind our proposed Label Attention Layer in Section 2. In Section 3 we describe our syntactic parsing model, and Section 4 presents our experiments and results. Finally, we survey related work in Section 5 and lay out conclusions and suggest future work in Section 6.

2 Label Attention Layer

The self-attention mechanism of Vaswani et al. (2017) propagates information between the words of a sentence. Each resulting word representation contains its own attention-weighted view of the sentence. We hypothesize that a word representation can be enhanced by including each label's attention-weighted view of the sentence, on top of the information obtained from self-attention.

The Label Attention Layer is a novel, modified form of self-attention, where only one query vector is needed per attention head. Each attention head represents a label, and this allows the model to learn label-specific views of the input sentence.

We explain the architecture and intuition behind our proposed *Interpretable Label Attention Layer* through the example application of constituency parsing.

Figure 2 shows one of the main differences between our Label Attention mechanism and self-attention: the absence of the Query matrix $\mathbf{W^Q}$. Instead, we have a learned matrix $\mathbf{Q}$ of query vectors representing the labels. More formally, for the attention head of label l and an input matrix $\mathbf{X}$ of word vectors, we compute the corresponding attention weights vector $\mathbf{a}_l$ as follows:

$$\mathbf{a}_l = \text{softmax}\left(\frac{\mathbf{q}_l * \mathbf{K}_l}{\sqrt{d}}\right) \tag{1}$$

where d is the dimension of query and key vectors, $\mathbf{K}_l$ is the matrix of key vectors. Given a learned label-specific key matrix $\mathbf{W}_l^K$, we compute $\mathbf{K}_l$ as:

$$\mathbf{K}_l = \mathbf{W}_l^K \mathbf{X} \tag{2}$$

[1]Code and Model to be released soon at https://github.com/KhalilMrini/LAL-Parser.

Each attention head in our Label Attention layer has an attention *vector*, instead of an attention *matrix* as in self-attention. Consequently, we do not obtain a *matrix* of vectors, but a *single* vector that contains label-specific context information. This *context* vector corresponds to the green vector in Figure 3. We compute the context vector $\mathbf{c}_l$ of the label l as follows:

$$\mathbf{c}_l = \mathbf{a}_l * \mathbf{V}_l \tag{3}$$

where $\mathbf{a}_l$ is the vector of attention weights in Equation 1, and $\mathbf{V}_l$ is the matrix of value vectors. Given a learned label-specific value matrix $\mathbf{W}_l^V$, we compute $\mathbf{V}_l$ as:

$$\mathbf{V}_l = \mathbf{W}_l^V \mathbf{X} \tag{4}$$

The context vector gets added to each individual input vector, as shown in the red box in Figure 3. We project the resulting matrix of word vectors to a lower dimension before normalizing. We then distribute the word vectors computed by each label attention head, as shown in Figure 5.

Our Label Attention Layer contains one attention head per label. The values coming from each label are identifiable within the final word representation, as shown in the color-coded vectors in the middle of Figure 5.

The activation functions of the position-wise feed-forward layer make it difficult to follow the path of the contributions. Therefore we can remove the position-wise feed-forward layer, and compute the contributions from each label. We provide an example in Figure 6, where the contributions are computed using normalization and averaging. In this case, we are computing the contributions of each label to the span vector. The span representation for "*the person*" is computed following the method of Gaddy et al. (2018) and Kitaev and Klein (2018). However, forward and backward representations are not formed by splitting the entire word vector at the middle, but rather by splitting each label-specific word vector at the middle.

In the example in Figure 6, we show averaging one way of computing contributions, but other functions, such as softmax, can be used. Another way of interpreting predictions would be to look at the label-to-word attention distributions, which are the output vectors in the computation in Figure 2.

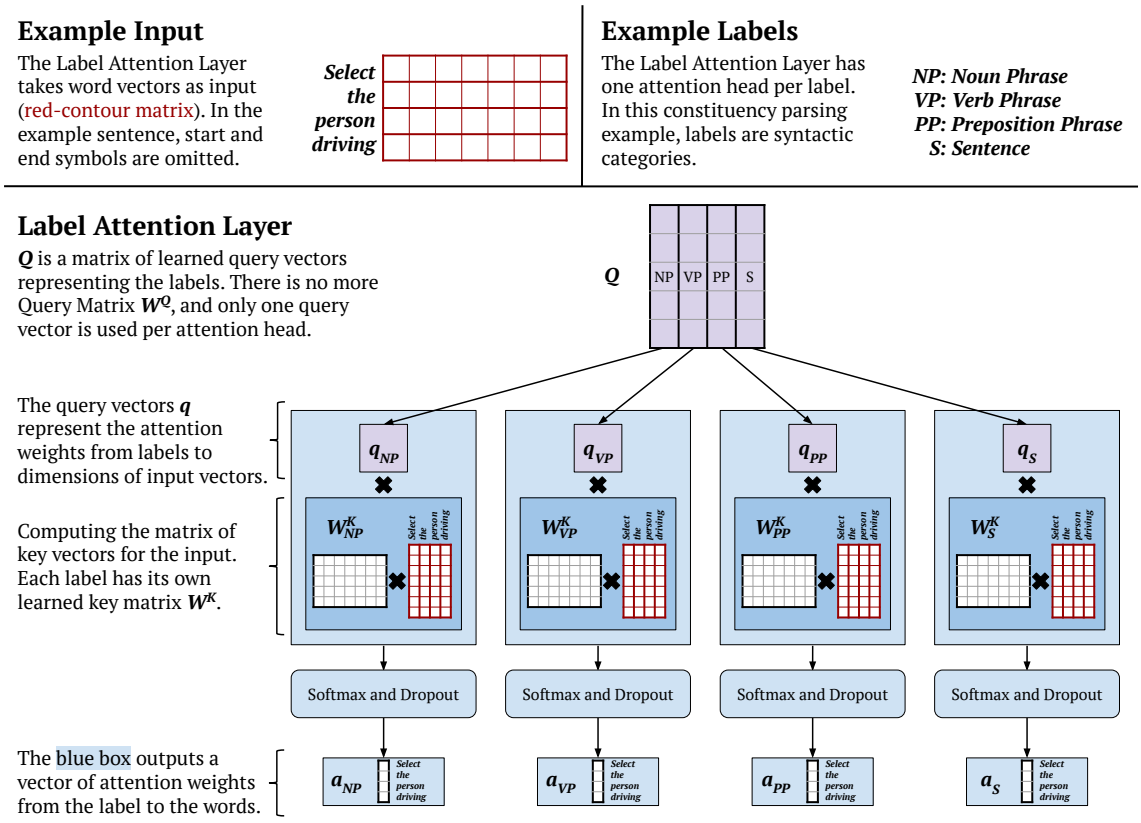

Figure 2: The architecture of our proposed Label Attention Layer. In this figure, the example application is constituency parsing, and the example input sentence is "*Select the person driving*".

3 Syntactic Parsing Model

3.1 Encoder

Our parser has an encoder-decoder architecture. The encoder has self-attention layers (Vaswani et al., 2017), preceding the Label Attention Layer. We follow the attention partition of Kitaev and Klein (2018), who show that separating content embeddings from position embeddings increases performance.

Sentences are pre-processed following Zhou and Zhao (2019). They represent trees using a simplified Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994). They propose two kinds of span representations: the division span and the joint span. We choose the joint span representation after determining that it was the best performing one. We show in Figure 4 how the example sentence in Figure 2 is represented.

The token representations for our model are a concatenation of content and position embeddings. The content embeddings are a sum of word and part-of-speech embeddings.

3.2 Constituency Parsing

For constituency parsing, span representations follow the definition of Gaddy et al. (2018) and Kitaev and Klein (2018). For a span starting at the i-th word and ending at the j-th word, the corresponding span vector s_{ij} is computed as:

$$\mathbf{s_{ij}} = \left[\overrightarrow{\mathbf{h_j}} - \overrightarrow{\mathbf{h_{i-1}}}; \overleftarrow{\mathbf{h_{j+1}}} - \overleftarrow{\mathbf{h_i}}\right] \quad (5)$$

where $\overleftarrow{\mathbf{h_i}}$ and $\overrightarrow{\mathbf{h_i}}$ are respectively the backward and forward representation of the i-th word obtained by splitting its representation in half. An example of a span representation is shown in the middle of Figure 6.

The score vector for the span is obtained by applying a one-layer feed-forward layer:

$$\mathbf{S}(i,j) = \mathbf{W_2}\text{ReLU}(\text{LN}(\mathbf{W_1}\mathbf{s_{ij}} + \mathbf{b_1})) + \mathbf{b_2} \quad (6)$$

where LN is Layer Normalization, and $\mathbf{W_1}$, $\mathbf{W_2}$, $\mathbf{b_1}$ and $\mathbf{b_2}$ are learned parameters. For the l-th syntactic category, the corresponding score $s(i,j,l)$ is then the l-th value in the $\mathbf{S}(i,j)$ vector.

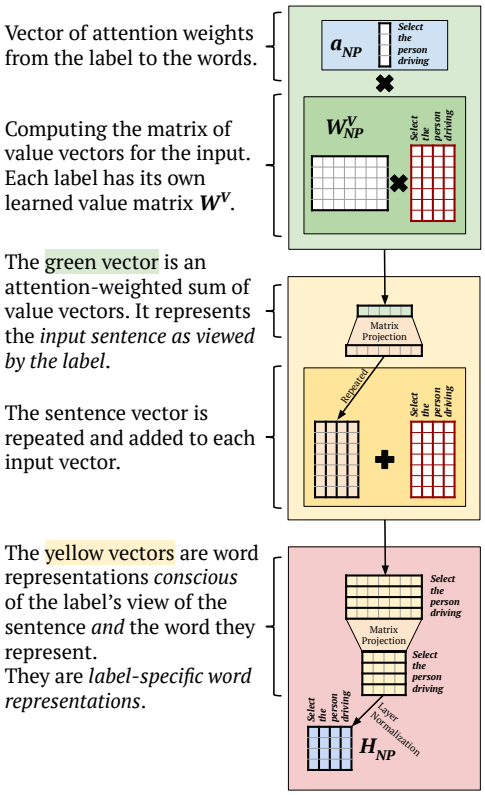

Figure 3: The Value vector computations in our proposed Label Attention Layer.

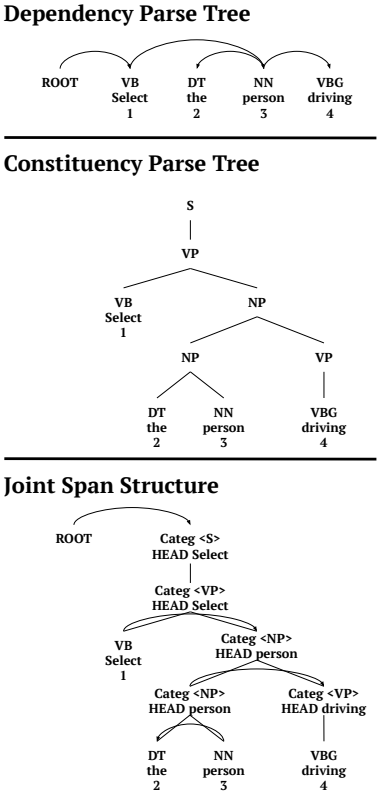

Figure 4: Parsing representations of the example sentence in Figure 2.

Consequently, the score of a constituency parse tree T is the sum of all of the scores of its spans and their syntactic categories:

$$s(T) = \sum_{(i,j,l) \in T} s(i, j, l) \tag{7}$$

We then use a CKY-style agorithm (Stern et al., 2017; Gaddy et al., 2018) to find the optimal tree $\hat{T}$ with the highest score. The model is trained to find the correct parse tree T^*, such that for all trees T, the following margin constraint is satisfied:

$$s(T^*) \geq s(T) + \Delta(T, T^*) \tag{8}$$

where Δ is the Hamming loss on labeled spans. The corresponding loss function is the hinge loss:

$$L_c = \max\left(0, \max_T[s(T) + \Delta(T, T^*)] - s(T^*)\right) \tag{9}$$

3.3 Dependency Parsing

We use the biaffine attention mechanism (Dozat and Manning, 2016) to compute a probability distribution for the dependency head of each word. The child-parent score α_{ij} for the j-th word to be the head of the i-th word is:

$$\alpha_{ij} = \mathbf{h_i^{(d)}}^T \mathbf{W} \mathbf{h_j^{(h)}} + \mathbf{U}^T \mathbf{h_i^{(d)}} + \mathbf{V}^T \mathbf{h_j^{(h)}} + b \tag{10}$$

where $\mathbf{h_i^{(d)}}$ is the dependent representation of the i-th word obtained by putting its representation $\mathbf{h_i}$ through a one-layer perceptron. Likewise, $\mathbf{h_j^{(h)}}$ is the head representation of the j-th word obtained by putting its representation $\mathbf{h_j}$ through a separate one-layer perceptron. The matrices $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are learned parameters.

The model trains on dependency parsing by minimizing the negative likelihood of the correct dependency tree. The loss function is cross-entropy:

$$L_d = -\log\left(P\left(h_i|d_i\right) P\left(l_i|d_i, h_i\right)\right) \tag{11}$$

where h_i is the correct head for dependent d_i, $P\left(h_i|d_i\right)$ is the probability that h_i is the head of d_i, and $P\left(l_i|d_i, h_i\right)$ is the probability of the correct dependency label l_i for the child-parent pair (d_i, h_i).

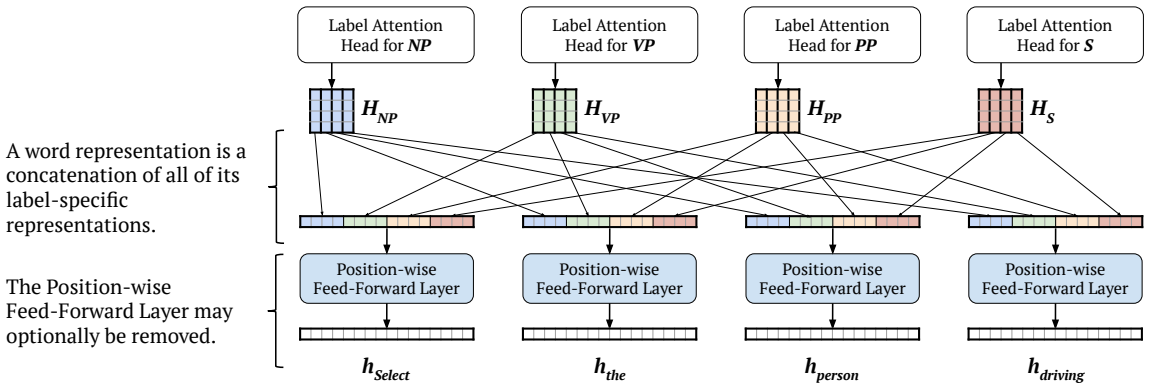

Figure 5: Redistribution of the label-specific word representations to form word vectors by concatenation. The label color scheme follows Figure 2. We do not use colors for the vectors resulting from the position-wise feed-forward layer, as the label-specific information moved.

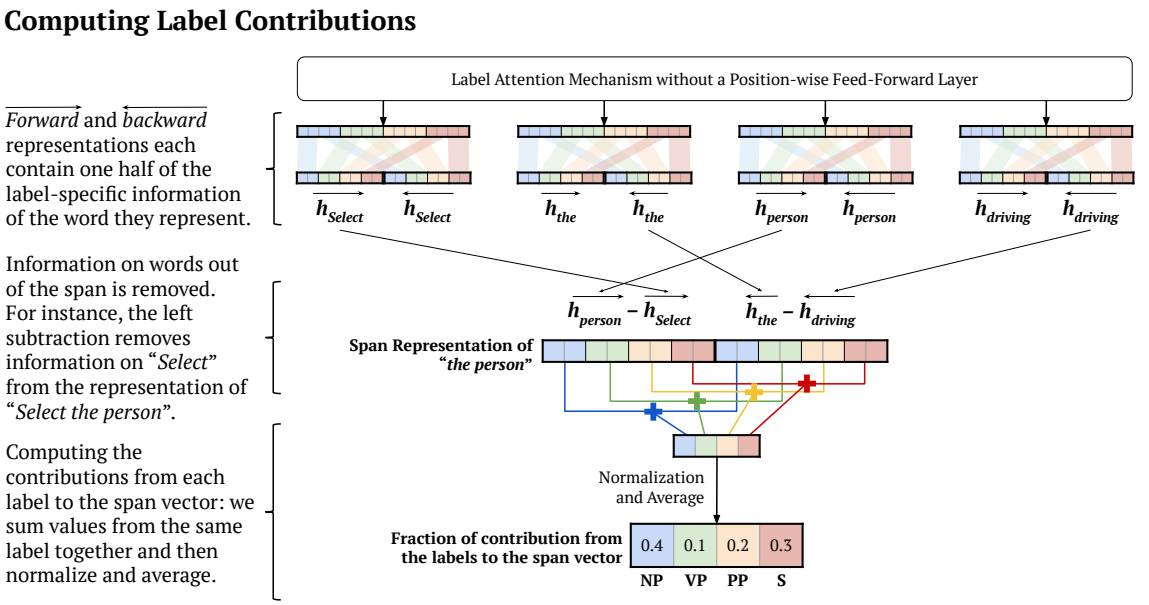

Figure 6: If we remove the position-wise feed-forward layer, we can compute the contributions from each label attention head to the span representation, and thus offer interpretability. This illustrative example follows the label color scheme in Figure 2.

3.4 Decoder

The model jointly trains on constituency and dependency parsing by minimizing the sum of the constituency and dependency losses:

$$L = L_c + L_d \tag{12}$$

The decoder is a CKY-style (Kasami, 1966; Younger, 1967; Cocke, 1969; Stern et al., 2017) algorithm, modified by Zhou and Zhao (2019) to include dependency scores.

4 Experiments

We evaluate our model on the English Penn Treebank (Marcus et al., 1993) benchmark dataset, and use the Stanford tagger (Toutanova et al., 2003) to predict part-of-speech tags.

For the evaluation, we follow standard practice: using the `EVALB` algorithm (Sekine and Collins, 1997) for constituency parsing, and reporting results without punctuation for dependency parsing.

4.1 Setup

In our experiments, the Label Attention Layer has 128 dimensions for the query, key and value vectors, as well as for the output vector of each label attention head. For the dependency and span scores, we use the same hyperparameters as Zhou and Zhao (2019). We use the large cased pre-trained XLNet (Yang et al., 2019) as our embedding model. We use a batch size of 100 and each model is trained on a single 32GB GPU.

4.2 Ablation Study

As shown in Figure 6, our Label Attention Layer is interpretable only if there is no position-wise feed-forward layer. We investigate the impact of removing this component from the LAL.

We show the results of our ablation study on the Residual Dropout and Position-wise Feed-forward Layer in Table 1. The second row shows that the performance of our parser decreases significantly when removing the Position-wise Feed-forward Layer and keeping the Residual Dropout. However, that performance is recovered when removing the Residual Dropout as well, as shown in the last row. In fact, removing both the Residual Dropout and Position-wise Feed-forward layer is the best-performing option for a LAL parser with 2 self-attention layers.

4.3 Self-Attention Layers

Table 2 shows the results of our experiments varying the number of self-attention layers in our parser's encoder. The best-performing option is 3 layers. However, removing the position-wise feed-forward layer and residual dropout actually decreases performance.

4.4 State of the Art

Finally, we compare our results with the state of the art in constituency and dependency parsing. Table 3 compares our results with the parsers of Zhou and Zhao (2019). Our LAL parser establishes new state-of-the-art results, improving significantly in dependency parsing.

5 Related Work

Before the HPSG-based model of Zhou and Zhao (2019), the previous state-of-the-art model architecture in constituency parsing was held by Kitaev and Klein (2018). The latter use an encoder-decoder parser. The novelty of Kitaev and Klein (2018) is in their self-attentive encoder, where they stack multiple levels of self-attention to embed words. The resulting word embeddings are fed onto a decoder, which they borrow from Stern et al. (2017).

Gaddy et al. (2018) use a bidirectional LSTM to compute forward and backward representations of each word. A contiguous span of words from position i to position j of the form $\mathbf{w}_i, \mathbf{w}_{i+1}, ..., \mathbf{w}_j$ therefore has forward representations $\overrightarrow{\mathbf{h}_i}, \overrightarrow{\mathbf{h}_{i+1}}, ..., \overrightarrow{\mathbf{h}_j}$ and backward representations $\overleftarrow{\mathbf{h}_i}, \overleftarrow{\mathbf{h}_{i+1}}, ..., \overleftarrow{\mathbf{h}_j}$. The scores for this span to be attributed a non-terminal are computed as the following output vector, as per Stern et al. (2017):

$$\mathbf{s}(i,j) = \mathbf{W_2}\text{ReLU}(\mathbf{W_1}\mathbf{r}_{ij} + \mathbf{b_1}) + \mathbf{b_2} \tag{13}$$

where $\mathbf{b_1}$ and $\mathbf{b_2}$ are biases, $\mathbf{s}(i,j)$ is a vector of dimensionality L (the number of possible nonterminal labels), and $\mathbf{r}_{ij}$ is computed as:

$$\mathbf{r}_{ij} = [\overrightarrow{\mathbf{h}_j} - \overrightarrow{\mathbf{h}_i}; \overleftarrow{\mathbf{h}_i} - \overleftarrow{\mathbf{h}_j}] \tag{14}$$

Therefore, $s(i,j,l)$ indicates the score for the span of words $\mathbf{w}_i, ..., \mathbf{w}_j$ to be labelled with nonterminal l. These scores are then used in a CKY-style algorithm that produces the final parse tree.

Kitaev and Klein (2018) redefine $\mathbf{r}_{ij}$ as the following:

Self-Attention Layers	Position-wise Feed-forward Layer	Residual Dropout	Precision	Recall	F1	UAS	LAS
2	Yes	Yes	96.23	**96.03**	96.13	97.16	96.09
2	No	Yes	84.85	84.88	84.86	87.14	82.98
2	No	No	**96.33**	95.98	**96.16**	**97.30**	**96.17**

Table 1: Ablation study on our parser with a Label Attention Layer and 2 self-attention layers. The parameters are the Position-wise Feed-forward Layer and Residual Dropout of the Label Attention Layer. The evaluation is done on the Penn Treebank test set.

Self-Attention Layers	Position-wise Feed-forward Layer	Residual Dropout	Precision	Recall	F1	UAS	LAS
2	Yes	Yes	96.23	96.03	96.13	97.16	96.09
3	Yes	Yes	**96.47**	**96.20**	**96.34**	**97.33**	**96.29**
3	No	No	96.30	95.92	96.11	97.12	95.94
12	Yes	Yes	96.27	96.06	96.16	97.24	96.14

Table 2: Performance on the Penn Treebank test set of our LAL parser according to the number of self-attention layers.

Model	F1	UAS	LAS
HPSG + 12 self-attention + BERT	95.84	97.00	95.43
HPSG + 12 self-attention + XLNet	96.33	97.20	95.72
Our LAL + HSPG + 3 self-attention + XLNet	**96.34**	**97.33**	**96.29**

Table 3: Comparison of the performance for Constituency and Dependency Parsing of our Label Attention Layer (LAL) parser and the HPSG parser of Zhou and Zhao (2019) on the Penn Treebank test set.

$$\mathbf{r}_{ij} = [\mathbf{y}_{2j} - \mathbf{y}_{2i}; \mathbf{y}_{2j+1} - \mathbf{y}_{2i+1}] \quad (15)$$

where $\mathbf{y}$ is the output of the encoder layer, i.e. the sum of all self-attention layers of the encoder.

None of these papers provided visualisations of their learned attention distributions, but Kitaev and Klein (2018) and Gaddy et al. (2018) do extensive interpretation studies, using ablation and probing of linguistic theories.

6 Conclusions and Future Work

In this paper, we introduce a revised form of self-attention: the Label Attention Layer. In our proposed architecture, attention heads represent labels. We have only one learned vector as query, rather than a matrix, thereby diminishing the number of parameters per attention head. We incorporate our Label Attention Layer into the HPSG parser (Zhou and Zhao, 2019) and obtain state-of-the-art results on the English Penn Treebank benchmark dataset. Our results show 96.34 F1 score for constituency Parsing, and 97.33 UAS and 96.29 LAS for dependency parsing.

In future work, we want to investigate the interpretability of the Label Attention Layer, notably through the label-to-word attention distributions and the contributions of each label attention head. We also want to incorporate it in more self-attentive NLP models for other tasks.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, and Hui Jiang. 2016. Enhancing and combining sequential and tree lstm for natural language inference. *arXiv preprint arXiv:1609.06038*.

John Cocke. 1969. Programming languages and their compilers: Preliminary notes.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. Whats going on in neural constituency parsers? an analysis. 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 999–1010.

Tadao Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686.

Yang Liu. 2019. Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Carl Pollard and Ivan A Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press.

Matīss Rikters. 2018. Impact of corpora quality on neural machine translation. *arXiv preprint arXiv:1810.08392*.

Matīss Rikters, Mārcis Pinnis, and Rihards Krišlauks. 2018. Training and adapting multilingual nmt for less-resourced and morphologically rich languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*.

Giancarlo Salton, Robert Ross, and John Kelleher. 2017. Attentive language models. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 441–450.

Satoshi Sekine and Michael Collins. 1997. Evalb bracket scoring program. *URL: http://www. cs. nyu. edu/cs/projects/proteus/evalb*.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 818–827.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. *arXiv preprint arXiv:1804.08199*.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on penn treebank. *arXiv preprint arXiv:1907.02684*.