Semantic Role Labeling
NLP 2020 - Prof. Navigli

Shayan Alipour
Sapienza Università di Roma
alipour.1889595@studenti.uniroma1.it

Abstract

This project addresses the issue of Semantic Role Labeling (SRL) using Attention mechanism and Transformer architecture. Our specific goal was to identify the arguments of the predicates in different sentences and associate each argument with its class. We used GloVe for word embedding and created multiple embedding layers in order to feed data with more features and improve the results. The best F1-scores we garnered for the tasks of argument identification and classification were **90.62**% and **85.74**%, respectively.

1 Introduction

In natural language processing, semantic role labeling is the process that assigns labels to words or phrases in a sentence that indicate their semantic role in the sentence, such as that of an agent, goal, or result(Jurafsky and Martin, 2009). Simply put, it is the task of addressing "Who did what to whom, how, where and when?"

2 Data Preprocessing

The dataset has been divided into three different subsets: Train, Dev, and Test with 39279, 1334, and 2000 distinct sentences respectively. The structure of the dataset used in this project is shown in figure 1. We benefited both from features available directly in the dataset and those that we created based on the given data to boost the results.

Before diving into the details of the embedding layers, we first had to deal with multi-predicate sentences. To that aim, we duplicated each sentence with respect to the number of existing predicates in that sentence, each time considering a single predicate of that sentence and its role. In doing so, we masked all other predicates with a Null token ('_') and omitted their roles.

The circumstances led to the approach we employed for data preprocessing in test set sentences differing slightly from that applied to the Train and Dev datasets because during training phase we had access to the roles. We created batches in order to handle training more smoothly, which required us to consider and add a padding token which was unnecessary in the test dataset because according to the project fixed code, we received the test dataset sentence by sentence. To sum it all up, we didn't utilize the Torch DataLoader library and implemented the whole process from scratch for the Train and Dev datasets. In each batch, the longest sentence counted for the baseline and PAD token added to the other sentences.

2.1 Features

We selected 'lemmas' and part-of-speech tags ('pos_tags') for each sentence and created two other features named predicate indicator and lemmas indicator that work as follows for the given example sentence with the EAT/BITE predicate shown in Table 1. The predicate indicator works like a flag to determine the predicate's position in the sentence, and the lemmas indicator shows the positional distance between each word and the predicate. We believe it can help the model in cases where specific roles only occur before or after the predicate. In addition to the original roles for each predicate, we created a feature named 'bi_roles' which is basically a non-zero flag for arguments and a zero flag for Null tokens.

We added a predicate position and sentence ID to the above-mentioned features (plus replace 'words' with 'lemmas') because we have to apply merger and decoder at the end of the task. Ergo, we needed grounds to merge sentences based upon that. We also used Positional Encoding before feeding the data to our Transformer model, which

Words	The cat ate the fish.
Predicate indicator	0, 0, 1, 0, 0, 0
Lemmas indicator	-2, -1, 0, 1, 2, 3
Roles	_, Agent, _, _, Patient, _
Bi_Roles	0, 1, 0, 0, 1, 0

Table 1: Dataset Features

we will discuss in the Model section.

We used the lemmas feature instead of words as previously mentioned because it was more convenient to build the word vectors based on GloVe's vocabulary. GloVe (Pennington et al., 2014) is a word representation technique that helps us assign vectors to each word in which the distance between vectors is related to the words' semantic similarity.

3 Transformer Model

In 2017 the Transformer neural network architecture was introduced in the paper titled "Attention is All You Need" (Vaswani et al., 2017). The network employs encoder-decoder architecture which is well-suited for sequence-to-sequence learning (Seq2Seq). The main difference between Transformer and Long Short-term Memory (LSTM) is that in LSTM, or generally in Recurrent Neural Networks (RNNs), the input data needs to pass sequentially or serially one after the other. However, in the Transformer model, the input sequence can be passed in parallel. The General Transformer model architecture can be seen in figure 2 and we present a step-by-step description of it in the next sections, though we've only used the encoder unit of this architecture for the SRL task.

3.1 Positional Embedding

Due to the fact that the Transformer processes the sequential data in parallel, we have to apply a positional encoder to keep track of the positions of words in the sentence. The implementation is based on the PyTorch documentation (PyTorch), which was based on the original paper (Vaswani et al., 2017). We used the same approach proposed in the paper, utilizing sine and cosine functions to make use of the order of the words in a sentence.

3.2 Multi-Head Attention

For each word we have Q, K, and V vectors, we used them to compute the attention vector using the following formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Q is the vector representation of one word in the sequence, K vector is the representation of all the words in the sequence, and V is also another vector representation of all the words in the sequence. The values in V are multiplied and summed with some weights (attentions). The weights are defined by how each word (Q) has an effect on other words in the sequence (K).

To obtain a model learned from the different representations of those three vectors and improve the results, instead of generating one attention for each word, we employ, say, eight parallel attention layers or heads. The attention mechanism is repeated multiple times with linear projections of Q, K and V. These linear representations are done by multiplying Q, K and V by weight matrices that are learned during training. The structure of an operator inside the attention layer and multi-head attention is shown in figure 3 and 4 respectively.

3.3 Feed Forward

A simple neural network that is applied to every one of the attention vectors. These feed forward nets are used in practice to transform the attention vectors into a form that is digestible by the next block.

3.4 Base Model

As mentioned earlier, we only implemented the encoder unit of this architecture for the SRL task. The reason behind that is given the whole sentence, we considered that labels(arguments) are independent from each other.

$$x = (x_1, x_2, .., x_n)$$

$$y = (y_1, y_2, .., y_n)$$

$$P(y|x) = P(y_1|x)P(y_2|x)..P(y_n|x)$$

In the above formulae, x is the input sentence, and y is the output labels in which we assumed is an independent random variable. Therefore, the occurrence of each label given the whole sentence is independent of the rest of the labels in that sentence.

The model that gave us the best results has six identical encoder layers. Each layer has two sub-layers, the first one was a 10-head self-attention mechanism and the second was a simple fully-connected feed-forward network. Based on the PyTorch documents for the Transformer model, the source and target shapes are (*S, N, E*) and (*T, N, E*) respectively, in which S is the source sequence length, T is the target sequence length, N is the batch size, and E is the feature number.

At the end, we defined a linear layer which gets the Transformer encoder's output as an input and gives out the labels as an output. We refer to this model as a base model in this report.

3.5 Improved Model

After observing the base model results, we found out that the model struggles to find the position of the arguments, so we decided to add the feature 'bi_roles' which we defined earlier in the data preprocessing section, and instead of one linear layer at the end of the base model, we added two linear layers, both with the same input features, but one with only two classes as an output and the other with output classes that are equal to the number of total labels seen in the Train set, which was 36. The model's forward function returns both linear outputs.

Until now, there has not been much difference among the two, and the main dissimilarity stems from the trainer class. We get both results in the Trainer and calculate the loss for each one. We refer to the loss value for the multi-label output as multi_loss and bi_loss for the binary label output. We then multiplied the bi_role feature by the multi_loss which makes the loss of the null tokens (non-arguments) zero, so during the backward phase, the model mainly focuses on the positions that arguments exist.

4 Experiments/Results/Discussion

For hyperparameters, we conducted several random fine-tuning procedures such as dropout, feed-forward dimension, encoder layers, by hand. We utilized Cross-entropy loss, which is commonly applied as a loss function for classification problems.

We observed that after about 10-12 epochs models started to overfit, however training them for move epochs made the results better. So we trained models for at least 10 epochs, and because the training was time-consuming and Google Colab offers limited GPU usage, we trained models that showed better results for 20, 30, and 40 epochs.

We used ReLu as an activation function for our Transformer encoder layer, however, we tried GeLu (Hendrycks and Gimpel, 2016) too because based on its paper, it outperformed networks using ReLu as an activation function. That being said, it showed slightly weaker results on our task in comparison to ReLu, as can be seen in table 2. Last but not least, Adam optimizer is used to minimize the loss function during back-propagation (Kingma and Adam, 2014). Adam has proven to be one of the best choices for deep learning and has benefited from previous optimizers.

The method we described in section 3.5 helped our model predict the position of the arguments better. We trained the same model for an equal number of epochs multiple times regarding these two approaches and the second model almost always surpassed the former basic approach and improved the F1 scores for both argument identification and classification tasks by $\sim 1 - 2\%$.

4.1 Models Result

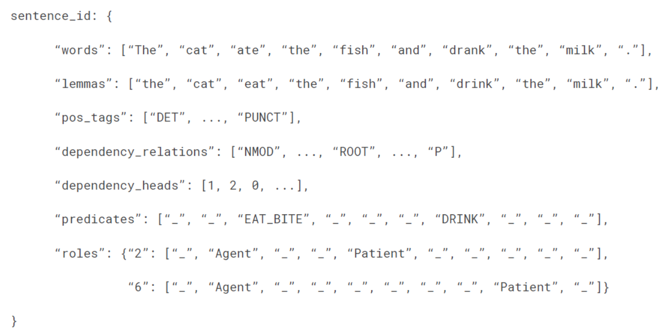
In table 2, "Adv." refers to models which are based on our second approach with two loss functions, "Dr" refers to the model's dropout rate, "FF-D" refers to models feed-forward dimension, "Atten. Heads" refers to the number of heads in the multi-head attention models.

5 Conclusion

In this project, we investigate the performance of Transformer models when it comes to SRL tasks by constructing Transformer encoders with different variants. We concluded that training more complicated models with dropout for at least 30 epochs shows better results. Implementing our novel approach in determining the arguments' positions by multiplying the "bi_roles" feature with calculated loss function, improved the F1 score. Nonetheless, there is always room for improvement. In future efforts, implementing the decoder section and taking into account the influence of roles on each other might resulted in higher F1 scores.

Adv.	Dr	Activ.	FF-D	Atten. Heads	Encoder Layers	Epochs	Arg Ident-F1	Arg Class-F1
N	0	ReLu	512	5	3	10	88.73	83.83
N	0	GeLu	512	5	3	10	88.21	83.09
N	0.2	ReLu	512	5	3	10	88.92	84.04
N	0.2	GeLu	512	5	3	10	89.01	83.75
Y	0.2	ReLu	512	5	3	10	89.37	83.87
Y	0.1	ReLu	1024	5	2	10	89.17	84.1
Y	0.1	GeLu	1024	5	2	10	88.93	83.94
Y	0.2	ReLu	1024	10	6	30	**90.62**	**85.74**
N	0.2	ReLu	1024	10	6	30	89.40	84.83
Y	0.2	ReLu	512	5	3	30	89.81	85.11
N	0	ReLu	512	5	3	40	89.36	83.91

Table 2: Hyperparameter Tuning Results



```
sentence_id: {
      "words": ["The", "cat", "ate", "the", "fish", "and", "drank", "the", "milk", "."],
      "lemmas": ["the", "cat", "eat", "the", "fish", "and", "drink", "the", "milk", "."],
      "pos_tags": ["DET", ..., "PUNCT"],
      "dependency_relations": ["NMOD", ..., "ROOT", ..., "P"],
      "dependency_heads": [1, 2, 0, ...],
      "predicates": ["_", "_", "EAT_BITE", "_", "_", "_", "DRINK", "_", "_", "_"],
      "roles": {"2": ["_", "Agent", "_", "_", "Patient", "_", "_", "_", "_", "_"],
                "6": ["_", "Agent", "_", "_", "_", "_", "_", "_", "Patient", "_"]}
}
```

Figure 1: Dataset Structure

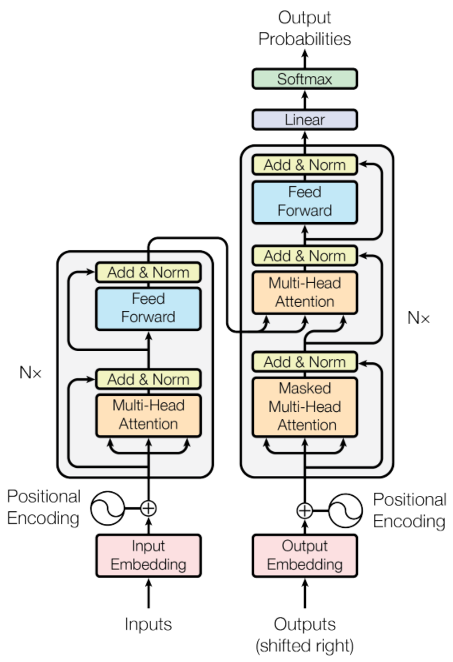


Figure 2: Transformer General Architecture

References

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv: Learning*.

Dan Jurafsky and James H. Martin. 2009. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, Upper Saddle River, N.J.

Diederik P Kingma and Jimmy Ba. Adam. 2014. Adam: A method for stochastic optimization. *arXiv preprint*, arXiv:1412.6980.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

PyTorch. Sequence-to-Sequence Modeling with nn.Transformer and Torchtext.

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.

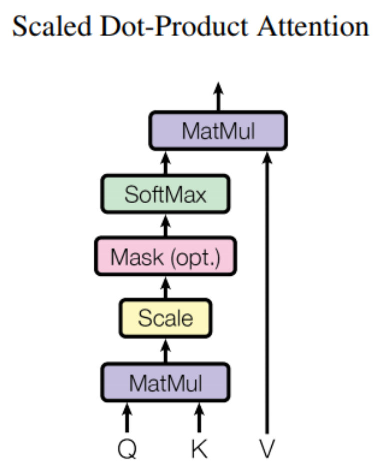


Figure 3: Scaled Dot-Product Attention

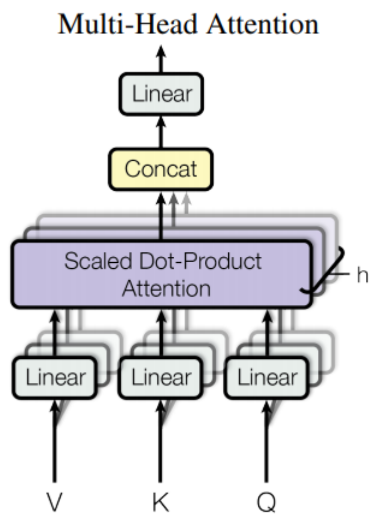


Figure 4: Multi-Head Attention