HW2 - Report
NLP 2019/2020

Donato Crisostomi, 1754001

1 Introduction

In the last years, we have observed in NLP a progressive shift from strictly syntactic tasks to ones that also involve semantics. Among these, a notable task is Semantic Role Labeling (SRL), which essentialy aims to answer the question "**who** did **what** to **whom**", "**when**", "**where**" and "**how**". This involves the prediction of predicate argument structure, i.e., both identification of arguments as well as their assignment to an underlying semantic role. SRL is believed to be a crucial step towards natural language understanding, and for this reason the obtained representations are beneficial in many NLP applications (Zhang et al., 2018; Wang et al., 2016).

There are two representations for argument annotation, namely *span-based* and *dependency-based*. In the former arguments are syntactic constituents (spans), whereas in the latter the goal is to identify the syntactic heads of arguments rather than the entire span. This is the representation used in this work.

2 Model

The standard formulation of semantic role labeling decomposes into four subtasks: (1) predicate detection, (2) predicate sense disambiguation, (3) argument identification, and (4) argument classification. My model addresses the last three subtasks. In particular, I implemented two separate models, one tackling only (3) and (4) and one tackling (2), (3) and (4), which I will refer to as M_{34} and M_{234}. The architectures are shown in fig. 6 and in fig. 7 respectively.

2.1 Argument identification and classification

M_{34} is composed of four main components:

1. a word representation component that given a word w_i in a sentence $\mathbf{w}$ builds a word representation x_i;
2. a *BiLSTM* encoder which takes as input the word representation x_i and returns a dynamic representation h_i of the word and its context;
3. a concatenation operator that attaches the predicate hidden representation h_p to every other word's hidden representation h_i;
4. a *Multi Layer Perceptron* that takes as input the representation $h_i \oplus h_p$ of the word and maps it to the roles space.

2.1.1 Word representation

The word representation is obtained as follows

$$x_i = x_i^{pos} \oplus x_i^{fram} \oplus x_i^{lem} \oplus x_i^{bert} \oplus pred\_ind_i$$

where $\oplus$ represents the tensor concatenation. The individual components of the representation are described hereinafter.

POS tags Part-of-speech tags are fed to a trainable 64-dimensional embedding, obtaining the POS embedding $\mathbf{x}^{pos}$. Incorporating information about POS tags is helpful in many NLP applications and SRL makes no exception: for example the fact that a word has POS tag '*proper noun*' makes it a good candidate for role '*Agent*' or '*Beneficiary*', while making it a poor candidate for, e.g., '*Asset*'.

Frames Frames are obtained from *VerbAtlas* (Di Fabio et al., 2019); In contrary to existing SRL resources that have predicate-specific role semantics, each frame in VerbAtlas is a cluster of WordNet synsets, thus allowing to obtain information about the 'verb class' semantics. For example, if we know that word 'chef' usually plays an 'Agent' role with respect to the predicate 'cook', thanks to VerbAtlas frames we also know that 'Agent' is a good candidate role for ’chef’ with respect to predicates ‘bake’ and ‘prepare’, which are in the same frame of ‘cook’. VerbAtlas frames are embedded via an embedding layer which is trained along with the whole model. The resulting embeddings have size 128.

Lemmas Dataset also comes with information regarding word lemmas, with the goal of reducing inflectional forms and sometimes derivationally related forms of a word to a common base form. By leveraging this information, the model is more robust to slight variations of the same concept that often arise for syntactic reasons. In our case, ‘chef’ is still the ‘agent’ if he is ‘cooking’ or if he ‘cooks’. Pre-trained 100-dimensional *FastText* word embeddings are used for lemmas, the pretrained embeddings provide a strong benefit as they are obtained from large datasets, augmenting the generalization capability of the model. These are fine-tuned during model training, in particular because they don’t cover the whole lemma vocabulary and thus some are randomly initialized.

Contextualized embeddings Each word in a sentence is embedded taking into account the context; these embeddings are obtained by using BERT (Devlin et al., 2018) which is a pretrained neural architecture that applies Transformers (Vaswani et al., 2017) to Language Modeling producing rich representations which have been proven beneficial for almost any NLP task; moreover, BERT embeddings account for subwords, being beneficial for words never encountered during training. The chosen size of the architecture is 768, as it is a good compromise between the size of the model and performance. The BERT embedding of the sentences is a preprocessing, as the inference model is fed already embedded sentences.

Predicate indicator Many sentences have more than one predicate, while some don’t have any. In order to let the model know who the candidate roles should refer to, a bit can be concatenated to each word, being 1 for predicates and 0 for normal words, as done in (Marcheggiani et al., 2017). This leads nevertheless to the necessity of duplicating sentences containing more than one predicate, in order to let the model unambigously decide with respect to the only predicate (if any).

Duplication Each sentence must be duplicated p times, where p is the number of predicates it contains; The resulting tensor has dimension $(B + D) \times S \times E$, where B is the batch size, D is the number of duplicates, S is the sequence length and E is the embedding size; This is done for every feature considered.

2.1.2 BiLSTM

Recurrent Neural Networks are the most used architectures for sequences; these summarize the context of a certain token m with a recurrently updated vector

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}), \quad m = 1, 2, \ldots, m$$

where $\mathbf{x}_m$ is the vector embedding of the token w_m and g defines the recurrence. Nevertheless, RNNs often fail to capture long-time dependencies; for this reason, LSTMs are often used. These employ a more complex recurrence, in which a memory cell goes through a series of gates, in fact avoiding repeated applications of non-linearity (Eisenstein, 2019). The hidden state $\mathbf{h}_m$ accounts for information in the input leading up to position m, but it ignores the subsequent tokens, which may also be relevant to the tag y_m; this can be addressed by adding a second LSTM, in which the input is reversed. This architecture is called *Bidirectional LSTM* and is the model I use in this work. See Implementation for the implementation details.

Since the LSTM expects a batch of sequences of equal length, *padding* is added. Moreover, usually a batch contains sentences of different lengths that when padded cause a lot of computation to be wasted; to address this issue *packing* is used, allowing Pytorch to internally optimize the computation.

2.1.3 Predicate concatenation

The dynamic representation obtained from the BiLSTM goes through one more processing step prior to be fed to the *Multi Layer Perceptron* for classification: the hidden representation h_p of the central predicate (i.e. the one the roles refer to) is concatenated to every other word representation h_i. This allows the classifier to reason about the candidate role and predicate jointly.

2.1.4 Classifier

The obtained tensor is eventually fed to a dense layer that maps each word to the roles space, resulting in each word having a score for each possible role. Softmax is then applied to the logits to obtain a prediction for each word.

2.2 Predicate sense disambiguation

Subtask (2) is a classification problem, this part of the model has three components, namely

1. word representation component, same as section 2.1 but without the frames embeddings;
2. a *BiLSTM* encoder;
3. a *Multi Layer Perceptron* to map the LSTM representation to the frames space.

Softmax is finally applied to the logits obtained from the dense layer, resulting in frames predictions. These are then passed to the subsequent layers of the model to proceed on subtask (3) and (4) as seen in section 2.1. The complessive model is thus multi-headed, with all the trainable embeddings having shared parameters optimized jointly. The *Categorical Cross Entropy* loss is used both for the argument identification and classification model and for the predicate disambiguation one, as it is the most suited loss function for multi-label classification. The complessive loss is a linear combination of the losses of the two models,

$$Loss_{tot} = \alpha Loss_{roles} + \beta Loss_{frames}$$

where α and β are two real numbers. Various values have been tried for α and β, the ones that performed best were $\alpha = 0.7$ and $\beta = 0.3$.

3 Experiments

The input dataset was split in the usual train, dev, test sets as described in table 1. The training dataset distribution is shown in fig. 5, we can see that the dataset suffers from a severe unbalance, with few classes covering most of the instances while showcasing only a few examples from the majority of the classes.

3.1 Parameterization

A grid search has been employed to obtain the best performing parameters, resulting in the parameters listed in table 2.

3.2 Training

M_{34} has been trained with early stopping over F_1 score, which caused the training to stop after 62 epochs; if the model was trained with early stopping over loss, it would have stopped after 34 instead, with 0.01 F_1 less. The training and dev loss curves along with the F_1 curves are shown in fig. 1 and fig. 2 respectively. The training dataset was shuffled to add variance in the data ordering, making the model more generalizable.

3.3 Regularization

In order to prevent the model from overfitting, two kinds of regularizers have been employed, namely dropout and weight decay. The former consists of randomly setting some computation nodes to zero during training to prevent feature co-adaptation, while the latter prevents the size of the learned weights from growing too large.

While the former resulted in an overall increase of the performance (as seen in table 4), the latter only slowed down convergence time, failing to increase the generalization capability of the model.

4 Results

M_{34} obtained an F_1 score of 0.93 and 0.89 for argument identification and classification respectively over the test dataset, while M_{234} obtained a score of 0.93, 0.92 and 0.86 in subtasks (2), (3) and (4) respectively. How the scores are related to the added components is reported in table 4.

Unsurprisingly, the model fails to classify rare classes like ‘Recursive’ which makes up for less than 1% of all the roles in the training set while obtains excellent performance over most frequent roles like ‘Agent’ and ‘Theme’. Two confusion matrices for M_{34} are shown: one normalized by the predicted label (axis 0) and one by the true label (axis 1), shown in fig. 8 and fig. 9 respectively. It appears that a significant part of the misclassifications come from missed roles, i.e. roles that the model mistakenly classifies as non roles, rather than ambiguity among roles. This is intuitive as the ’null’ role is by far the majority class, with 91% of roles having this tag. There are however few exceptions, like ‘Predicative’ which is classified as ‘Attribute’ more than 20% of times, but this may be due to the low frequency of the role itself, as can be seen in fig. 5.

5 Conclusions

The model reaches quasi state-of-the-art results in the task, but, as can be seen in table 4, this is made easier by the richness of the dataset in terms of features; an interesting experiment would be to design a full end-to-end model in which the features are infered as well and see how it performs.

	Size
Training	39286
Test	2000
Development	1335

Table 1: Dataset split.

Hyperparameters	
Embeddings dimension	
BERT	768
POS	64
Lemmas	100
Frames	128
Roles BiLSTM	
# Layers	2
Input dropout	0.2
Output dropout	0.5
Hidden dim	100
Training	
Optim	Adam
Learning rate	$1e^{-3}$
Patience	10

Table 2: M_{34} hyperparameters. Input dropout refers to the dropout applied to the embedded sentences before being fed to the LSTM. Output dropout instead is applied to the LSTM output before being fed to the dense layer.

Resources

Implementation

The employed implementation is the one provided by Pytorch, in which for each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

Hyperparameters	
Embeddings dimension	
BERT	768
POS	64
Lemmas	100
Predicates BiLSTM	
# Layers	2
Input dropout	0.2
Output dropout	0.5
Hidden dim	100
Roles BiLSTM	
# Layers	2
Input dropout	0.2
Output dropout	0.5
Hidden dim	100
Training	
Optim	Adam
Learning rate	$1e^{-3}$
Patience	10

Table 3: M_{234} hyperparameters. Input dropout refers to the dropout applied to the embedded sentences before being fed to the LSTM. Output dropout instead is applied to the LSTM output before being fed to the dense layer.

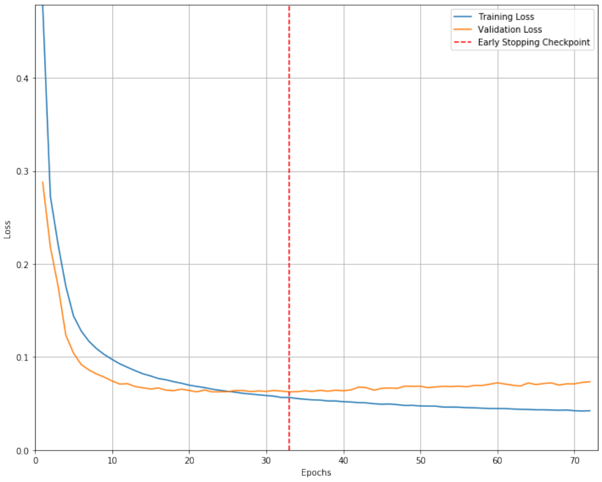


Figure 1: Loss over the training set vs loss over the validation set for M_{34}. The red vertical line identifies the epoch in which validation loss is minimum, which is the early stopping checkpoint if set to monitor validation loss.

F_1 scores	
	F1
baseline	0.728
+BERT	0.832 (+0.104)
+Dropout	0.861 (+0.029)
+POS	0.864 (+0.003)
+Lemmas	0.870 (+0.006)
+Frames	0.878 (+0.008)
$+h_p$ concatenation	0.889 (+0.011)

Table 4: F_1 scores for each model version. The plus sign in front of a feature means that it is added on top of the previous model architecture. For example POS is added to the baseline model having $BERT$ embeddings and dropout. h_p concatenation is the concatenation of the predicate hidden representation to every other word, described in section 2.1.3.

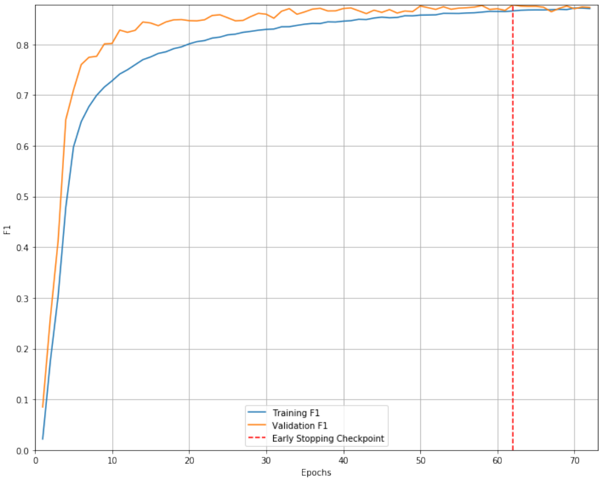


Figure 2: F_1 score over the training set vs F_1 score over the validation set for M_{34}. The red vertical line identifies the epoch in which validation F_1 score is maximum, which is the early stopping checkpoint.

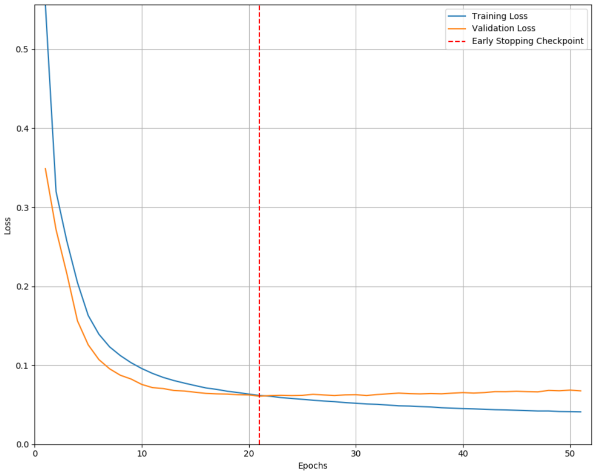


Figure 3: Loss over the training set vs loss over the validation set for M_{234}. The red vertical line identifies the epoch in which validation loss is minimum, which is the early stopping checkpoint if set to monitor validation loss.

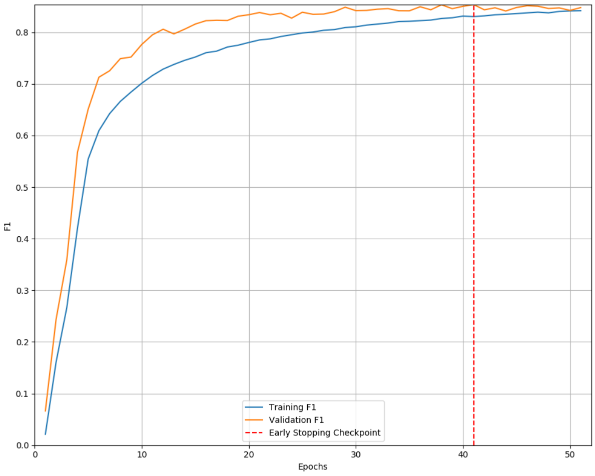


Figure 4: F_1 score over the training set vs F_1 score over the validation set for M_{234}. The red vertical line identifies the epoch in which validation F_1 score is maximum, which is the early stopping checkpoint.

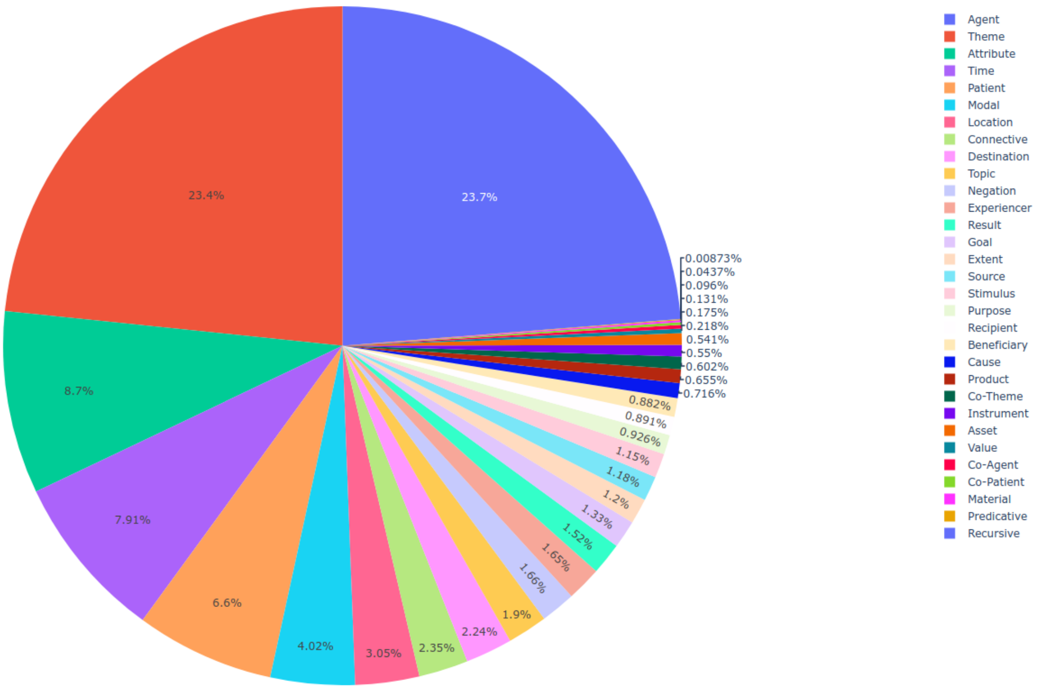
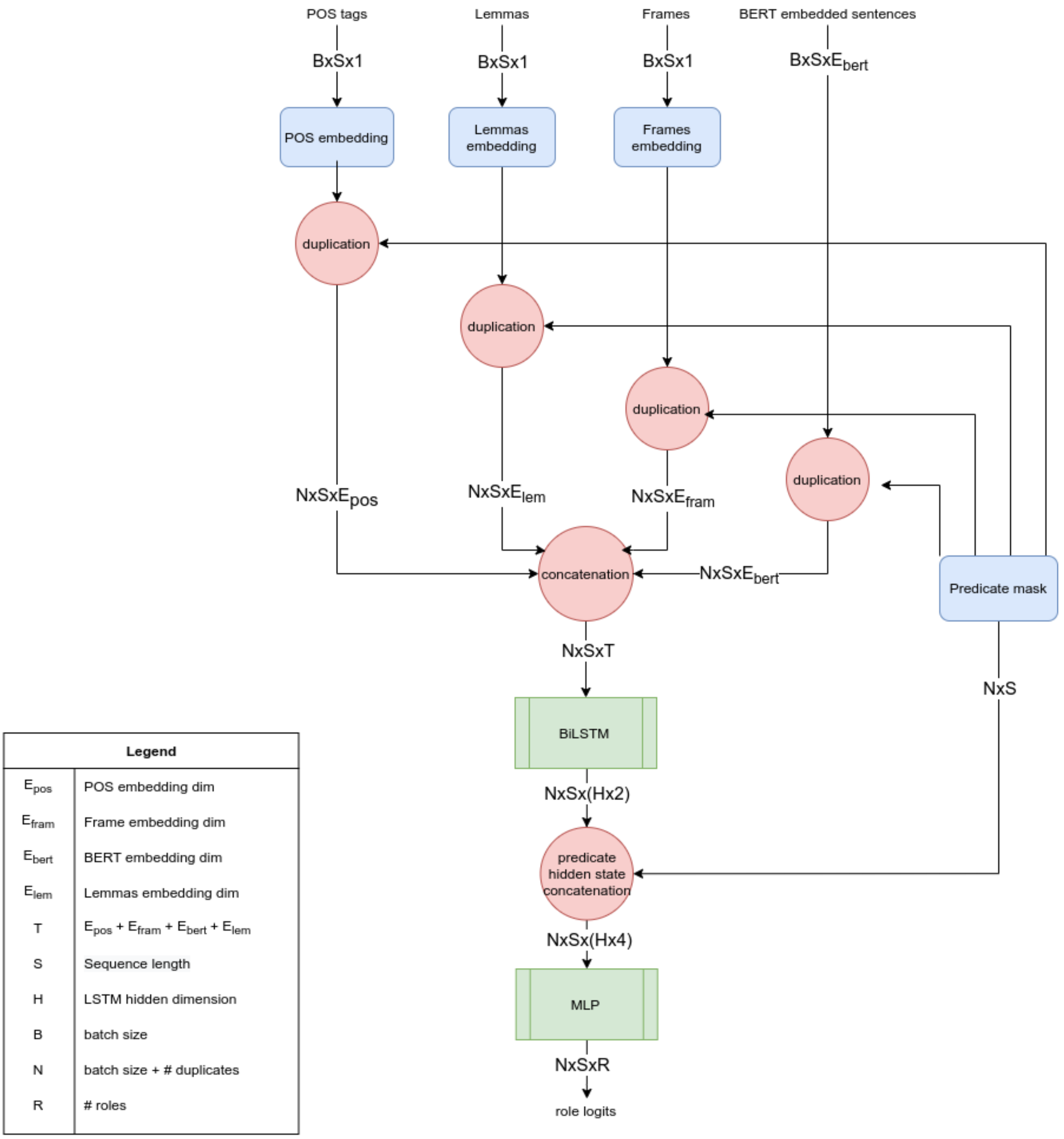
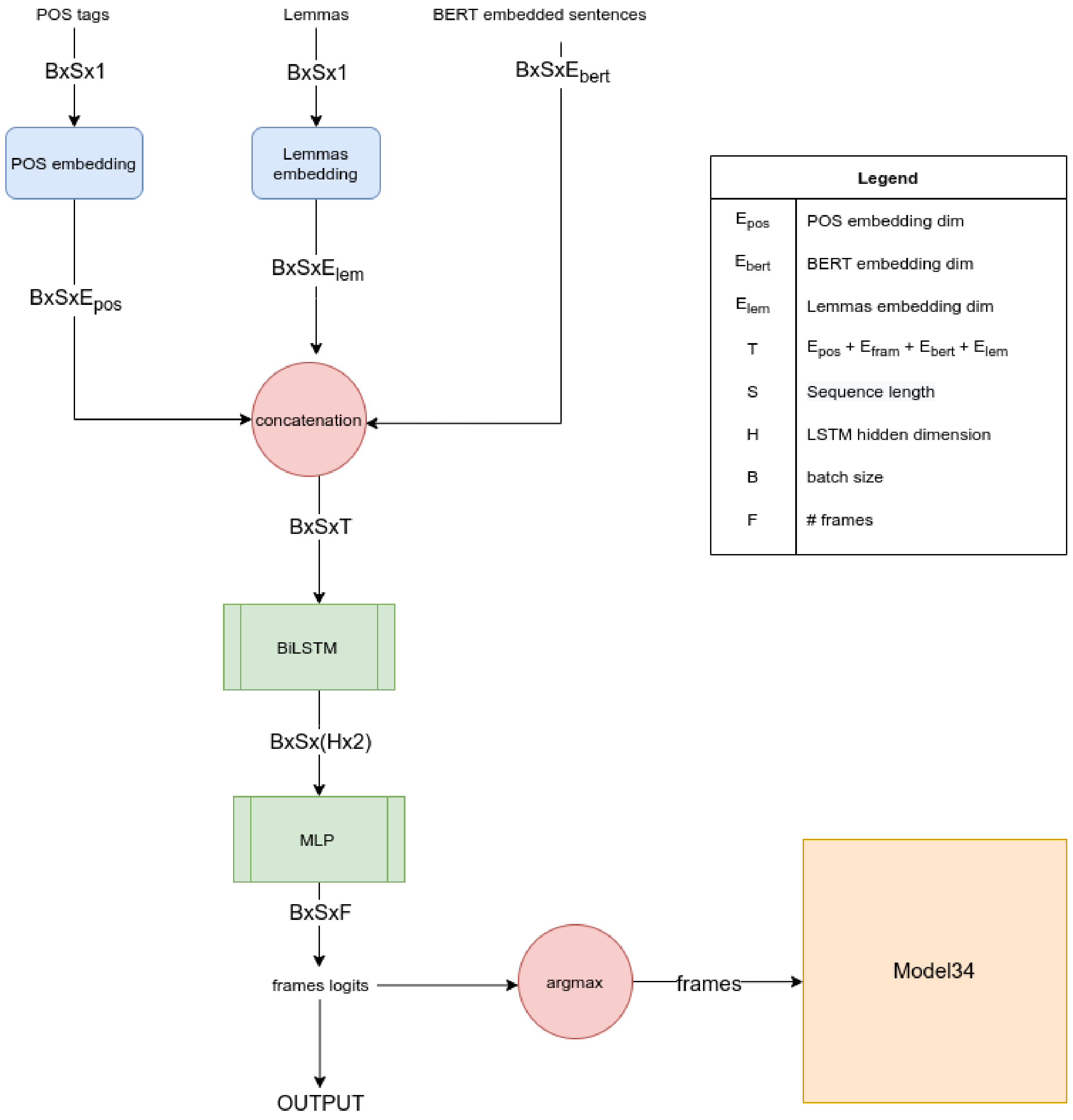


Figure 5: Pie chart of labels excluding the null role. Even without the null role which is by far the majority class, covering 91% of the tokens alone, the unbalance is evident; in fact, only 6 labels cover 75% of the roles. The choice of F_1 as metric is thus obligated, since a model predicting null role for every role would already get 91% accuracy, and could increase it much further just by accounting for the other few majority roles.



Legend	
E_{pos}	POS embedding dim
E_{fram}	Frame embedding dim
E_{bert}	BERT embedding dim
E_{lem}	Lemmas embedding dim
T	$E_{pos} + E_{fram} + E_{bert} + E_{lem}$
S	Sequence length
H	LSTM hidden dimension
B	batch size
N	batch size + # duplicates
R	# roles

Figure 6: Model architecture for M_{34}, as described in section 2.1.



Legend	
E_{pos}	POS embedding dim
E_{bert}	BERT embedding dim
E_{lem}	Lemmas embedding dim
T	E_{pos} + E_{fram} + E_{bert} + E_{lem}
S	Sequence length
H	LSTM hidden dimension
B	batch size
F	# frames

Figure 7: Model architecture for M_{234}, as described in section 2.2.

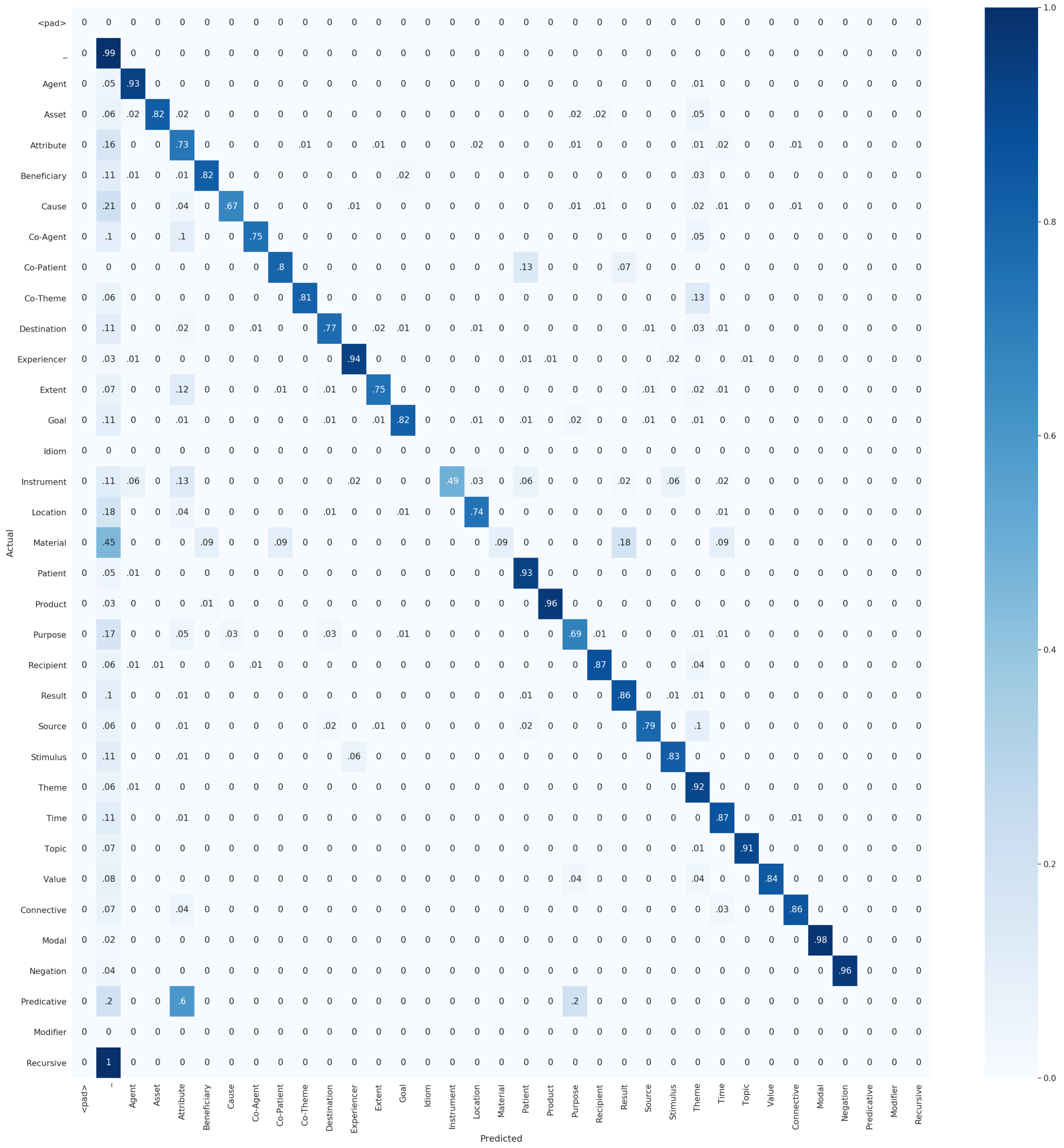
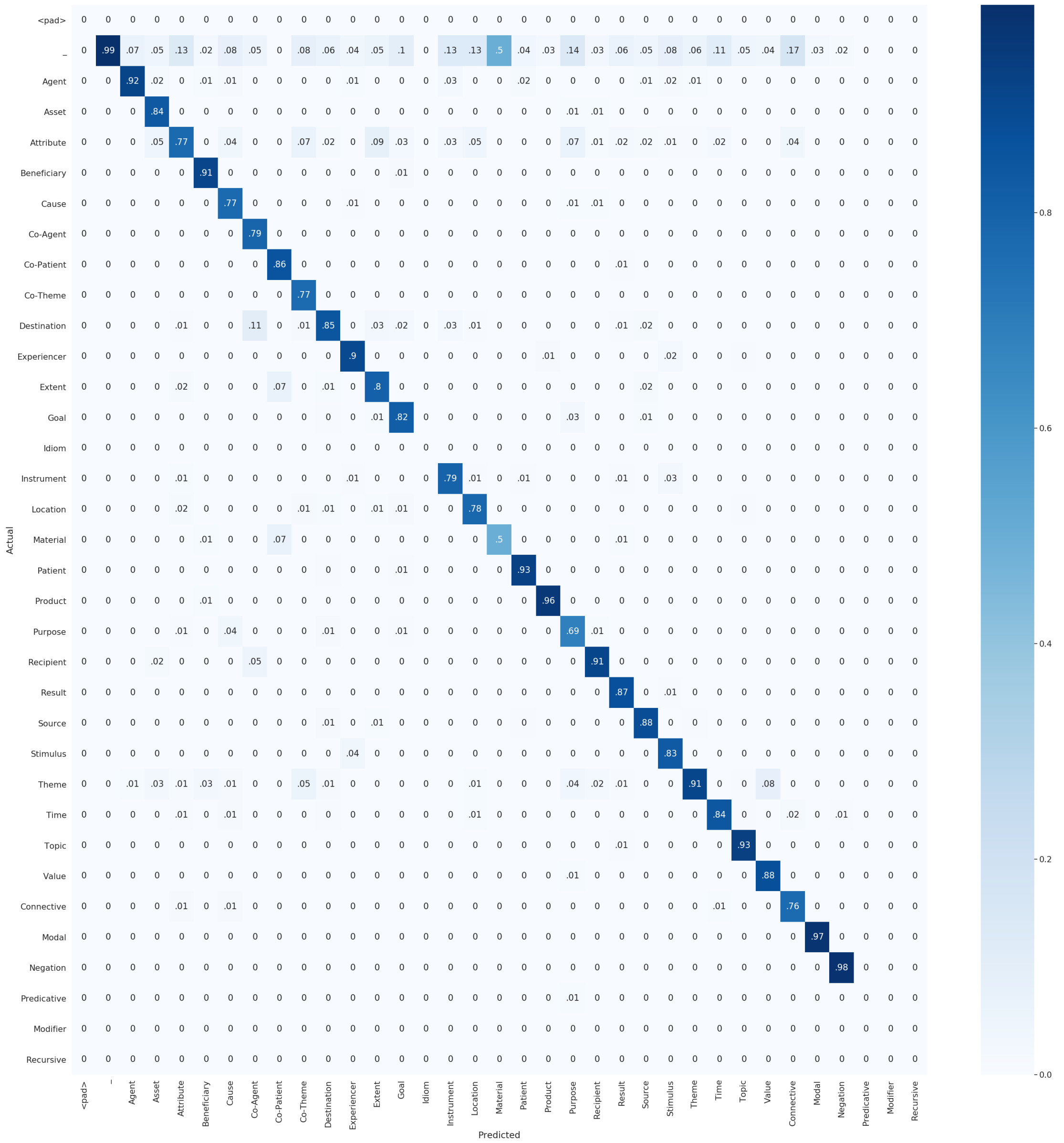


Figure 8: Confusion matrix normalized over the ground truth labels. It is not surprising that the predictions are biased towards the null role, as it is by far the majority label. There are few exceptions in which a significant part of the error comes from confusing two roles, but this happens with roles that appear in too few instances to be able to infer a cause.



	<pad>	_	Agent	Asset	Attribute	Beneficiary	Cause	Co-Agent	Co-Patient	Co-Theme	Destination	Experiencer	Extent	Goal	Idiom	Instrument	Location	Material	Patient	Product	Purpose	Recipient	Result	Source	Stimulus	Theme	Time	Topic	Value	Connective	Modal	Negation	Predicative	Modifier	Recursive
<pad>	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
_	0	.99	.07	.05	.13	.02	.08	.05	0	.08	.06	.04	.05	.1	0	.13	.13	.5	.04	.03	.14	.03	.06	.05	.08	.06	.11	.05	.04	.17	.03	.02	0	0	0
Agent	0	0	.92	.02	0	.01	.01	0	0	0	0	.01	0	0	0	.03	0	0	.02	0	0	0	0	.01	.02	.01	0	0	0	0	0	0	0	0	0
Asset	0	0	0	.84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	.01	0	0	0	0	0	0	0	0	0	0	0	0	0
Attribute	0	0	0	.05	.77	0	.04	0	0	.07	.02	0	.09	.03	0	.03	.05	0	0	0	.07	.01	.02	.02	.01	0	.02	0	0	.04	0	0	0	0	0
Beneficiary	0	0	0	0	0	.91	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cause	0	0	0	0	0	0	.77	0	0	0	0	.01	0	0	0	0	0	0	0	0	.01	.01	0	0	0	0	0	0	0	0	0	0	0	0	0
Co-Agent	0	0	0	0	0	0	0	.79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Co-Patient	0	0	0	0	0	0	0	0	.86	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0
Co-Theme	0	0	0	0	0	0	0	0	0	.77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Destination	0	0	0	0	.01	0	0	.11	0	.01	.85	0	.03	.02	0	.03	.01	0	0	0	0	0	.01	.02	0	0	0	0	0	0	0	0	0	0	0
Experiencer	0	0	0	0	0	0	0	0	0	0	0	.9	0	0	0	0	0	0	0	.01	0	0	0	0	.02	0	0	0	0	0	0	0	0	0	0
Extent	0	0	0	0	.02	0	0	0	.07	0	.01	0	.8	0	0	0	0	0	0	0	0	0	0	.02	0	0	0	0	0	0	0	0	0	0	0
Goal	0	0	0	0	0	0	0	0	0	0	0	0	.01	.82	0	0	0	0	0	0	.03	0	0	.01	0	0	0	0	0	0	0	0	0	0	0
Idiom	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Instrument	0	0	0	0	.01	0	0	0	0	0	0	.01	0	0	0	.79	.01	0	.01	0	0	0	.01	0	.03	0	0	0	0	0	0	0	0	0	0
Location	0	0	0	0	.02	0	0	0	0	.01	.01	0	.01	.01	0	0	.78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Material	0	0	0	0	0	.01	0	0	.07	0	0	0	0	0	0	0	0	.5	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0
Patient	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	.93	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Product	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	.96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Purpose	0	0	0	0	.01	0	.04	0	0	0	.01	0	0	.01	0	0	0	0	0	0	.69	.01	0	0	0	0	0	0	0	0	0	0	0	0	0
Recipient	0	0	0	.02	0	0	0	.05	0	0	0	0	0	0	0	0	0	0	0	0	0	.91	0	0	0	0	0	0	0	0	0	0	0	0	0
Result	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.87	0	.01	0	0	0	0	0	0	0	0	0	0
Source	0	0	0	0	0	0	0	0	0	0	.01	0	.01	0	0	0	0	0	0	0	0	0	0	.88	0	0	0	0	0	0	0	0	0	0	0
Stimulus	0	0	0	0	0	0	0	0	0	0	0	.04	0	0	0	0	0	0	0	0	0	0	0	0	.83	0	0	0	0	0	0	0	0	0	0
Theme	0	0	.01	.03	.01	.03	.01	0	0	.05	.01	0	0	0	0	0	.01	0	0	0	.04	.02	.01	0	0	.91	0	0	.08	0	0	0	0	0	0
Time	0	0	0	0	.01	0	.01	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	.84	0	0	.02	0	.01	0	0	0
Topic	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	.93	0	0	0	0	0	0	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	.88	0	0	0	0	0	0
Connective	0	0	0	0	.01	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	.76	0	0	0	0	0
Modal	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.97	0	0	0	0
Negation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.98	0	0	0
Predicative	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Modifier	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Recursive	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9: Confusion matrix normalized over the predicted labels. It is interesting that, not only the predictions are biased towards the null role as we have seen in fig. 8, but it also quite often happens that tokens which are not roles are classified as roles, suggesting that the model is not too much biased towards the majority class.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Andrea Di Fabio, Simone Conia, and Roberto Navigli. 2019. VerbAtlas: a novel large-scale verbal semantic resource and its application to semantic role labeling. pages 627–637.

Jacob Eisenstein. 2019. *Introduction to natural language processing*. The MIT Press.

Diego Marcheggiani, Anton Frolov, and Ivan Titov. 2017. A simple and accurate syntax-agnostic neural model for dependency-based semantic role labeling. *CoRR*, abs/1701.02593.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Rui Wang, Hai Zhao, Sabine Ploux, Bao-Liang Lu, and Masao Utiyama. 2016. A bilingual graph-based semantic model for statistical machine translation.

Zhuosheng Zhang, Yuwei Wu, Zuchao Li, Shexia He, Hai Zhao, Xi Zhou, and Xiang Zhou. 2018. I know what you want: Semantic learning for text comprehension.