An Attention-based Sequence-to-Sequence Model for Concept Tagging

Matteo Destro (221222)
University of Trento
Via Sommarive, 9, 38123 Povo, Trento TN
matteo.destro@studenti.unitn.it

Abstract

This project focuses on the task of concept tagging, usually one of the main components of more complex Natural Language Processing applications. The aim of this work is to develop a model able to extract concepts from sentences of a specific domain, in particular from queries of users asking for flights information. The proposed model is based on a sequence-to-sequence architecture, for which different techniques and hyperparameters combinations are explored to improve the performance. The final model features an attention mechanism and a beam search strategy at inference time. The results shows that this architecture can achieve a good accuracy in this specific domain.

1 Introduction

A *concept tagging* model is a key component in many NLU systems, since a lot of applications needs a mechanisms to understand the meaning of each word in a sentence in order to solve more complex tasks. The aim of concept tagging is to create a mapping between words and concepts, where each concept represents the underlying domain-specific meaning of a word. For instance, given a sentence "*I need a flight going from Phoenix to Orlando*", the objective is to obtain a sequence "`O O O O O O fromloc.city_name O toloc.city_name`" that can be used to extract the relevant information of the sentence.

Many different approaches to this problem can be found in the literature. This work focuses on the application of *sequence-to-sequence* (also known as *seq2seq*) models to solve a concept tagging task over the ATIS dataset. Further improvements are explored by implementing different *attention mechanisms* and using a *beam search* algorithm at inference time.

In Section 2 an overview of the dataset used for training and evaluating the model is given. Then, Section 3 describes the developed model while Section 4 presents how the experiments were organized and the obtained results. Finally, Section 5 briefly discusses some of the limitations and possible further improvements of this work. The code and dataset used in this project can be found in the GitHub repository[1].

2 Dataset

Training and evaluation of the models were performed on the *ATIS* dataset. It consists of transcriptions of sentences from humans asking for flight information on automated airline travel inquiry systems. The dataset was given already split into a train, validation and test set. The training set consists of 3,983 sentences, while the validation and test sets contain respectively 995 and 893 sentences. Each sentence contains on average 11 words, with a minimum of 1 word and a maximum of 46 words.

For each sentence, a sequence of tags (concepts) is given as target for the the concept tagging task. There are a total of 127 unique tags. Each tag is composed of an IOB prefix and a type suffix. Considering all the splits, there are 83 unique tag types, representing the possible classes of relevant information inside the sentences, e.g. date or city of departure. The training set has a total of 8 missing tag types, that are instead present in the other splits, while the validation set has 9 and the test set 14. Figure 1 shows the distribution of the concepts types in the dataset.

3 Model

This section describes the model developed in this project and the additional mechanisms implemented to improve the results.

3.1 Sequence-to-Sequence model

A *sequence-to-sequence* (Sutskever et al., 2014) model is a popular architecture for many NLU tasks. It consists of an encoder and a decoder. Both are usually

[1] https://github.com/materight/seq2seq-atis

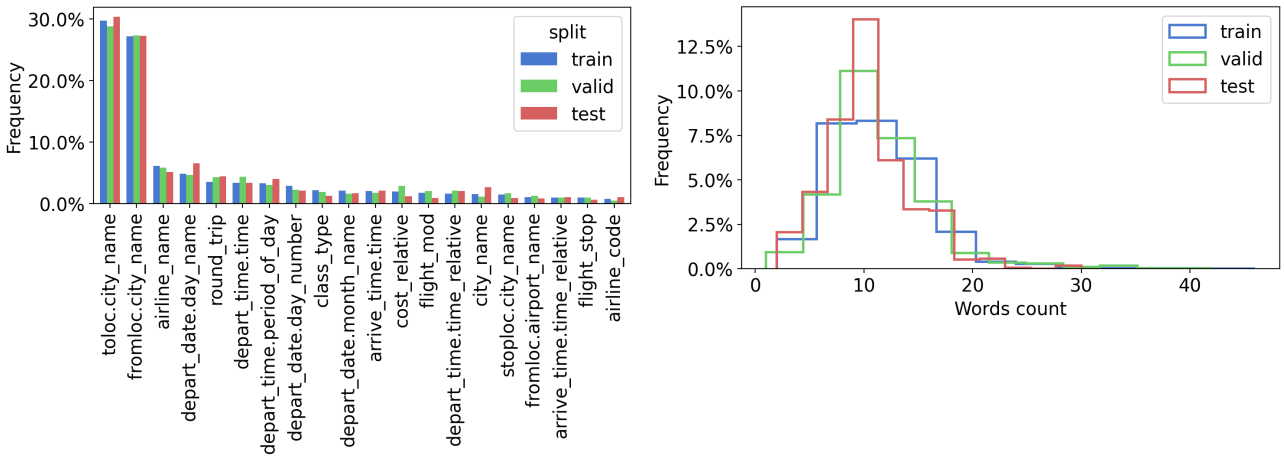

Figure 1: Distributions of the concept types and sentences lengths in the dataset. The three splits (train, validation and test) have similar distributions.

implemented with a *recurrent neural network* (RNN), a type of neural network that maintains an internal state based on the previous inputs, and therefore is particularly suited for variable-length sequence data. The encoder takes a sentence in input and creates a single fixed-length vector, called *context*, that summarizes the whole input sentence. This context is then passed to the decoder as initial hidden state and used to predict the target tokens. At each step, the decoder takes in input the previous predicted label and outputs the next one, applying a fully connected classifier to the RNN output to obtain a probability score for each token in the vocabulary, until a special `<end>` token is produced. The first seq2seq model was based on LSTM cells, an alternative to vanilla RNN cells, but various alternatives have been proposed, for instance the *encoder-decoder* architecture based on GRU cells (Cho et al., 2014). In this project, different encoder and decoder configurations were tested, with different recurrent cell types (Vanilla RNN, LSTM, GRU), uni/bidirectional encoder, different number of layers and hidden units. A comparison of the results for the different combinations can be found in Table 1.

3.2 Word Embeddings

Before giving a sentence in input to a sequence-to-sequence model, each word needs to be converted to a vector representation. To do so, an *embedding layer* can be defined, that would be responsible of converting an index representing a word in the domain vocabulary to a vector. The embedding layer is trained in the same way as the other layers and it learns a set of vector representations that most suits the kind of task we are solving.

To speed up the training process and improve the accuracy, the embedding layer can be initialized with weights from a pre-trained model, that has learned to encode the semantic similarity between words. Besides training an embedding layer from scratch, two pre-trained models were tested: the first one based on a Word2Vec model (Mikolov et al., 2013) trained over Google News data, the second one on a GloVe embedding (Pennington et al., 2014) trained on articles from Wikipedia and Gigaword. Both return embeddings of 300 values and were kept ”unfrozen”, i.e. they were fine-tuned during the training process.

3.3 Attention

One of the main issue with seq2seq models is that they try to describe an entire input sentence with a single fixed-length vector. If the sentence is long, the context vector may not have enough expressive power to encode it while maintaining all the relevant information. To solve this issue, the model can be extended with an *attention* mechanism to automatically detect the words in the input sentence that are relevant in predicting the next target label. Two kind of attention were tested in this work: *global* and *local*. The global attention (Bahdanau et al., 2016) computes the context vector using a combination of *all* the hidden states produced by the encoder. The final context vector is recomputed at each decoder step t as:

$$c_t = \sum_{j=1}^{T} \alpha_{tj} \overline{h}_j \tag{1}$$

where:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})} \tag{2}$$

is the *alignment score* at step t for the encoder hidden state $\overline{h}_j$, computed using:

$$e_{tj} = score(s_{t-1}, \overline{h}_j) \quad (3)$$

with:

$$score(s_{t-1}, \overline{h}_j) = v^\top \tanh(W s_{t-1} + U \overline{h}_j) \quad (4)$$

where v, W and U are weight matrices that must be learned (implemented as three linear layers without bias for simplicity), and s_{t-1} is an embedding for the decoder output of the previous step (obtained from an embedding layer). The input for the decoder layer can be then computed as a concatenation of the embedding of the previous decoder output s_{t-1} and the context vector c_t.

Luong et al. (Luong et al., 2015) proposed an improvement of this mechanism, where instead of the concatenation, a dot product is used to combine the previous output and the context vector, as:

$$score(s_t, \overline{h}_j) = s_t^\top \cdot \overline{h}_j \quad (5)$$

It must be noted that in this case instead of using the previous decoder output s_{t-1}, they first apply the decoder layer to compute s_t and only then the alignment scores and context vector are computed. After that, the input to the final classifier layer is obtained as:

$$\widetilde{h}_t = \tanh(W_c \cdot [c_t; h_t]) \quad (6)$$

where h_t is the current hidden state of the decoder.

They also described another type of attention: the *local attention*. In this case, instead of using all the words from the encoder to compute the context vector, the local attention is able to focus only on a small relevant subset of words around a certain *aligned position* p_t, computed as:

$$p_t = S \cdot \text{sigmoid}(v^\top \tanh(W h_t)) \quad (7)$$

where S is the length of the input sentence. To favor the alignment scores near p_t, a Gaussian function is used, centered on p_t and with standard deviation set to $D/2$, where D is the window size (in the experiments, the value of D was empirically determined and set to 3). The alignment weights therefore becomes:

$$\alpha'_{tj} = \alpha_{tj} \cdot \exp\left(-\frac{(j - p_t)^2}{2\sigma^2}\right) \quad (8)$$

where α_{tj} is computed as in Eq. 2 using as score function Eq. 5.

Figure 2 shows an example of the alignment scores learned by the model.

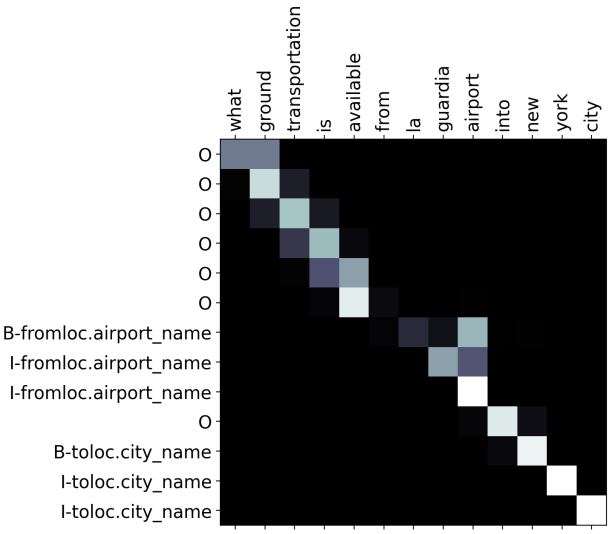

Figure 2: Example of the alignment scores learned by the model. Each row contains the alignment scores for each input word obtained when predicting the corresponding tag in the vertical axis. It is clear how the model is focusing on the word "*airport*" when predicting that "*la guardia*" refers to an airport name.

3.4 Beam Search

In the standard seq2seq model, at each step the tag with the highest output probability is selected. This is a simple greedy approach, but it might not yield the best results. Given a set of input words $x_1, ..., x_n$ (a sentence) and a set of corresponding tags (concepts) $y_1, ..., y_n$ that we want to predict, by using the greedy approach at each step t we are simply maximizing $p(y_t \mid x_n, ...x_1)$, while the actual task we are trying to solve would require to maximize $p(y_n, ..., y_1 \mid x_n, ..., x_1)$. Finding the optimal solution is infeasible, since it would require to test all the possible combinations of tags outputted by the model. However, a good approximation can be obtained using *beam search* (Freitag and Al-Onaizan, 2017). We define a *beam width* K and for each step of the decoder we maintain the K top-scoring candidate sentences according to their log-likelihood, computed as:

$$\log P(y_t, .., y_1 \mid x) = \sum_{i=1}^{t} \log P(y_i \mid y_{i-1}, ..., y_1, x)$$

Using the log-likelihood instead of simply the likelihood allows to avoid problems of numerical instability with long sentences. Once we reach the final tag, i.e. when $t = n$, the sequence with the highest log-likelihood is selected and given in output. This strategy can be used only at inference time, since the cross-entropy loss used to train the model assumes a

cell type	embedding	hidden	layers	bidir.	attention	drop.	lr	min F_1	avg (σ) F_1	max F_1
LSTM	Word2Vec	256	1	True	Local	0.3	0.001	**91.91**	**93.39 (1.58)**	**94.24**
GRU	Word2Vec	256	1	True	Dot	0.1	0.005	91.60	92.33 (1.31)	93.05
GRU	GloVe	128	2	True	Local	0.3	0.001	91.83	92.29 (2.94)	92.76
GRU	Word2Vec	256	1	True	Concat	0.5	0.002	87.04	89.94 (3.63)	91.92
LSTM	None	128	1	True	Dot	0.1	0.001	87.96	89.31 (3.97)	91.56
GRU	None	128	2	True	Concat	0.5	0.001	84.43	87.23 (5.01)	89.86
GRU	Word2Vec	256	1	True	Local	0.3	0.001	88.50	88.97 (1.78)	89.32
LSTM	None	256	1	True	Concat	0.3	0.005	84.79	85.90 (2.95)	87.05
LSTM	None	256	2	True	Concat	0.1	0.001	80.41	82.90 (4.39)	86.47
LSTM	None	256	1	True	None	0.3	0.002	80.96	82.24 (3.08)	83.51
Vanilla	GloVe	256	1	True	Local	0.1	0.001	80.63	82.43 (2.50)	83.30
GRU	None	256	2	True	Concat	0.1	0.002	73.39	77.04 (3.48)	78.84

Table 1: Minimum, average (with standard deviation) and maximum F_1 score for the 12 hyper-parameters configurations that obtained the best performance, evaluated on the test set.

greedy candidate selection. Table 2 shows a comparison of the performances obtained with different beam width values.

4 Experiments and Results

Different hyper-parameters combinations were evaluated, to determine the best configuration. Each combination was trained and tested 15 separate times, each time for 30 epochs and with a batch size of 8 sentences (value determined empirically). The sentences in each batch were padded with a special padding index, in order to have a consistent sentence length. 25 different configurations were tested, for a total of 375 runs. The training was executed using a cross-entropy loss and Adam optimizer. To improve the convergence speed, a technique called *teacher enforcing* was applied during training: at each step of the decoder, the input to the recurrent layer is selected randomly between the previous *predicted* and *expected* output. Further implementation details can be found in the code repository.

The minimum, maximum and average F_1 score obtained by the 12 best models are reported in Table 1. The results show that the largest improvement is given by the attention mechanism, since without any attention, the best model obtained a F_1 score of only 83.51. The local attention seems to give the best results, even if the other mechanisms are still relevant. Another, smaller, improvement is given by the pre-trained embedding, where both Word2Vec and GloVe seems to give good results. Regarding the beam search strategy, a beam width of 3 gives the largest improvement, even if not very significant, as can be seen in Table 2.

It must be noted that the dataset contains some incorrect or ambiguous labelling, that affect the models final scores. For instance, in the test set, in the sentence "*what ground transportation is available from la guardia airport into new york city*", "*la guardia*" and "*new york city*" are labeled respectively as "`airport_name`" and "`city_name`", while the final model outputs "`fromloc.airport_name`" and "`toloc.airport_name`" that given the context, should be considered the correct labels.

K	1	2	3	4	8	16	32
$\mathbf{F_1}$	92.23	92.37	92.44	92.35	92.35	92.35	92.25

Table 2: A comparison of the results obtained at inference time by different beam width (K) values, tested using the best hyper-parameter configuration (see Table 1).

5 Conclusions

The proposed model is able to approach the performances of other state-of-the-art techniques (Louvan and Magnini, 2020) and could be easily adapted to solve similar tasks in other domains. A major limitation of this work is given by the available computational power, since the pool of hyper-parameters configurations explored was relatively small. A more exhaustive search could discover better configurations and improve the performance.

Another limitation is given by the implementation of the beam search, that as already explained in Section 3.4 can only be used at inference time. This introduces a discrepancy between training and testing, that might lead to poor performances and instability. A possible improvement would be to include the beam search strategy in the training process, using for instance a *beam search optimization* (Wiseman and Rush, 2016).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk,

and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*.

Samuel Louvan and Bernardo Magnini. 2020. Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality.

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.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization.