Natural Language Understanding 2021 Project — Concept Tagging 1

Federico Pedeni
University of Trento
federico.pedeni@studenti.unitn.it

Abstract

This is a report for the project CT1 of the Natural Language Understanding course for the year 2021. The project consists in the implementation of a sequence-to-sequence model performing a concept tagging task: given a sentence, the model must learn to distinguish words that are relevant in a semantic scheme defined by the corpus used. We will test several deep-learning-based configurations starting from the "seq2seq" architecture; then we will show some alternatives for the decoder input modality, following the ideas in [10]. Additionally, we have implemented three attention mechanisms and a beam search strategy for improving performances. The code for this project is publicly available on GitHub[1].

1. Introduction

Natural language understanding is a fairly complex field: up to now, there is no unique model that can perform tasks at all the different levels of abstraction required by human communication. Thus, language understanding systems require the implementation of several components that attend tasks of different complexities, and between these concept tagging is placed at an intermediate level both of complexity and abstraction.

Concept tagging, known also as *slot filling*, is the process of retrieving, inside a sentence, those tokens that represent relevant entities or objects in a specific semantic frame, which is defined by the data source at disposal. It can be involved in the creation of knowledge bases, since it allows to recognize concepts that play a significant role in an utterance, i.e. the ones that are involved in actions or functions performed in the defined semantics. Concepts should not be conceived only as relevant entities, but more as candidate components of the logical rules in an hypothetical knowledge base built from the data. We can find those concepts in sentences as objects or complements, since they act as specifiers of actions happening in a sentence. This nature of *specification* distinguishes concept tagging from the more common task of Named-Entity Recognition (NER), which can be defined as the retrieval of some entities that have been assigned a name; in this case, named entities are more probably subjects or objects.

Token	Tag
flights	O
from	O
memphis	B-fromloc.city_name
to	O
las	B-toloc.city_name
vegas	I-toloc.city_name
on	O
sunday	B-depart_date.day_name

Table 1. A sentence with associated concept tags in IOB format.

The state of the art in this field is represented by recurrent neural networks (RNNs [3], LSTMs [7] or GRUs [2]), which in recent years have grown bigger and more complex in structure, from both perspectives of submodules' variety and internal structure. The most successful configuration is the encoder-decoder (or *seq2seq*) architecture [16], which we will describe in Section 3, alongside with additional decoding strategies and attention mechanisms which are implemented as additional modules for the standard architecture. In Section 4 will be presented results of the experiments that have been carried on, followed by some concluding remarks in Section 5.

2. Corpus

The main data source for this project was the Air Travel Information System (ATIS) corpus [6]: it is composed of a training set of 4978 utterances and a test set of 893 utterances, containing labels for both intent detection and slot filling; for our purposes we will only use the latter ones. This dataset categorizes words exploiting the IOB tagging scheme, and the relevant words (i.e. the non-'O' tagged ones) are labeled with one in 83 unique tags (127 if counting different IOB occurrences) related to the air transportation field. The utterances have been collected as dialogues of a travel-planning simulation, and they represent queries about possible flights, locations, arrival/departures and timings: more than actual questions, they seem unstructured requests of information, with a prevalence of noun phrases and imperative constructions.

It is worth noting that the usage of this dataset is restricted to its semantic field: it does not allow to general-

[1] https://github.com/adefgreen98/NLU2021-Project

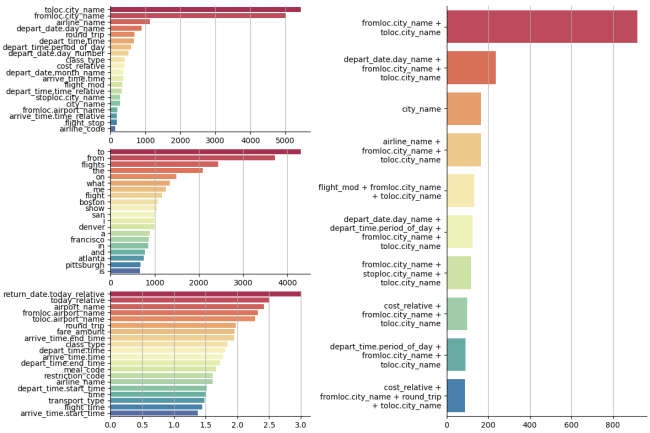

Figure 1. Training set statistics. Left (top-bottom) 20 most common tags, 20 most common words, 20 tags with highest average IOB length. Right: 10 most frequent co-occurrencies of tags.

ize to semantic areas that may be more common in daily language, its limited size can be easily overfit with deep models, and its vocabulary is also constrained. However, its small size becomes an advantage when designing experiments restricted conditions; moreover, it could be tested in experiments exploiting transfer learning to other datasets with similarly structured queries.

3. Methods

As it is usually done for sequence translation tasks, we can formalize slot filling as the problem of retrieving the best sequence of tags $y_1, ..., y_n$ of the same length of the input utterance $x_1, ..., , x_n$, that maximizes the conditional probability of the sequence given the sentence $p(\mathbf{y} \mid \mathbf{x})$. Therefore, different methods can be distinguished by how they design or parametrize this probability function. More traditional methods try to model output probability over the whole sequence, as in conditional random fields (CRF) [11]: they use graph-based methods to compute matches of nearby words' features in a context window of fixed size. We will use a simple implementation and the results in [14] as a baseline of comparison with sequence-to-sequence models.

Conversely, deep learning-based approaches shape sequence likelihood in a token-by-token manner, where the model output for each token is the conditional probability of the current tag given the previous ones and the input sequence:

$$p(\mathbf{y} \mid \mathbf{x}; \theta) = \prod_{i=1}^{n} p(y_i \mid y_{i-1}, ..., y_1, \mathbf{x}; \theta)$$

where θ is the set of values that parametrize our model.

3.1. Recurrent Neural Networks

The actual improvement of recurrent neural networks is that they directly model the conditional distribution at each timestep i: thanks to a softmax normalization, we can convert each hidden state of the network into a probability distribution over the set of tags, allowing to handle sequences of different lengths with the same architecture. In this way, we can design a loss function that we can use for finding the optimal set of parameters θ^* that minimizes the summation of each token-wise error, which we can define with the *cross entropy* loss function:

$$l(x_i, \bar{y}_i; \theta) = \sum_{j=1}^{d} -\bar{y}_{ij} p(y_{ij} \mid y_{i-1}, ..., y_1, \mathbf{x}; \theta)$$

where d is the number of categories and $\bar{y}_i$ is the ground truth for the token x_i as one-hot vector. It can be noted how summing this up across the sentence results in a negative-log-likelihood approach to optimize the conditional probability $p(\mathbf{y} \mid \mathbf{x}; \theta)$:

$$\theta^* = \arg \min_{\theta} \sum_{i=1}^{n} -\log p(\bar{y}_i \mid y_{i-1}, ..., y_1, \mathbf{x}; \theta)$$

We can then optimize this function by stochastic gradient descent until convergence, iterating over the samples in the dataset for many epochs; in particular, we will use the Adam [9] and AdamW [12] optimization methods for better gradient computation.

As for the different cell types, we have selected 3 architectures for our comparison. The first is the **Vanilla RNN** [3] cell of the Elman type: it is the most simple one, and it is composed of one hidden layer that combines the previous hidden state and the current input, to produce the next hidden state, and a softmax classifier that produces the target distribution for the current label from the current hidden state; both these layers include a $\tanh$ activation function.

A second choice is the **LSTM** [7] cell, which is more complex: it uses a *gated* architecture, where specialized hidden layers produce vector masks to modulate inputs and outputs. There are three gates: an *input gate* and an *output gate*, which are used for masking the homonym vectors; and a *forget gate*, which is used to update an intermediate vector called cell-state, which is passed to the activation function for computing the next hidden state, but is also carried on during the sequence. Keeping a cell state is a method to avoid the disruption of relevant information due to the squashing non-linearity of the activation, and it also helps with the vanishing gradient issue.

Finally, we examine the **GRU** [2] architecture: this is a simplified version of the LSTM, since it allows to safeguard the hidden state from non-linear disruptions but it only implements two gates (an *update gate* and a *reset gate*), thus has a lower number of parameters. It has shown to be equally good in terms of performances and it can be found in many state-of-the-art works. In order to better en-

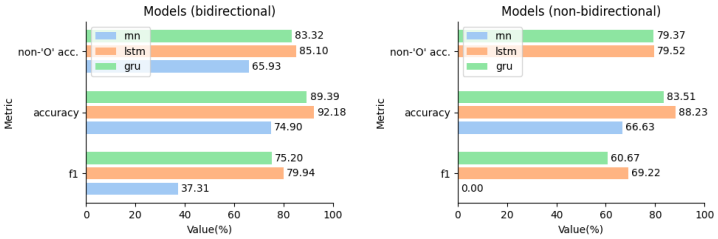

Figure 2. Cell types performances, scores averaged across 10 iterations per each configuration.

code overall sentence meaning, we also test **bidirectionality**, which is a mechanism that allows a recurrent model to have both a forward and a backward representation of the sentence at each timestep.

3.2. Seq2Seq Model

Sequence to sequence models [16] are composed by two recurrent networks, one that maps the input sequence to a fixed length vector and the other that has to extract relevant information for each label in the sequence from that vector, using it as its initial hidden state. This structure decouples the processes of classification and compression, allowing for a smarter usage of the hidden state.

While for the encoder the input sequence is some kind of word embedding (either one-hot, learnable or pre-trained), the decoder is usually fed at each timestep with the label produced in the previous one; an additional embedding layer may be used also here, as it is common in machine translation, for increasing model capacity. Based on some previous work regarding varying the **decoder input** modality [10], we have tested some methods for designing decoder inputs.

A. 1-hot encoding.. The labels do not undergo any modification and are fed in the decoder 1-hot encodings of the predicted tag.

B. Label embedding. Labels are transformed in indices that are used to train an embedding layer sized $\frac{3}{4}$ of the number of classes, as additional layer to increase model capacity.

C. Previous output. The decoder is fed back directly with its previous output, rescaled to the range $[0, 1]$ with a softmax function; this could allow to propagate the gradient back similarly to what happens for residual connections in ResNets [5].

D. Sentence tokens. The input is the same of the encoder: embedded tokens for each timestep.

E. Word + Label. Following the work by Kurata *et al*. [10], we concatenate token embeddings and label embeddings and feed them through a fully connected layer, to preserve sentence-level information.

In addition, for cases *A*-*C* we are also enabling *teacher forcing*: with a random probability (fixed at 0.5) we feed the correct label in the decoder, instead of the predicted one, according to each different modality. This is needed to stabilize the training and to avoid overfitting. This feature is deactivated during evaluation mode.

Another mechanism we introduce for the decoder is **beam search**. The process of choosing the most probable tag for each token could be seen as a *greedy* approach to decoding; but because next predictions are results of a highly nonlinear function, we have no guarantee that this will produce the optimal sequence overall. Therefore, since exploring the whole space of possible sequences would have an exponential cost, we introduce a method to explore only some possibilities for each step: we keep track of the top *beam-size* scoring sequences and at each timestep we only generate *beam-size* candidates for every one of them. Concretely, this means extracting more than one predicted tag from the predicted distribution of each element in the beam, in order to choose from all the generated sequences the top *beam-size* scoring ones.

Notwithstanding its efficiency in other natural language tasks [4], this technique has been shown to be defective in output originality, compared to other methods based on *sampling* from the target distribution [8]: during the beam-update process, the best scoring items are usually generated from the same sequence, yielding results that only differ for one final part. This may not be the case for slot filling, since each token usually is not associated to many different targets and therefore does not change much its role depending on context; however, this could be an interesting research topic for future studies.

3.3. Attention mechanisms

Attention is a technique that allows to directly model the network's internal weighting process. It consists in an additional module that maps a decoder hidden state h_t to a set of weights $\alpha_{t1}, ..., \alpha_{tn}$ spanning the input sequence; these values are used to compute a context vector c_t as a weighted sum of the encoder hidden vectors h'_s:

$$c_t = \sum_{s=1}^{n} h'_s \alpha_{ts}$$

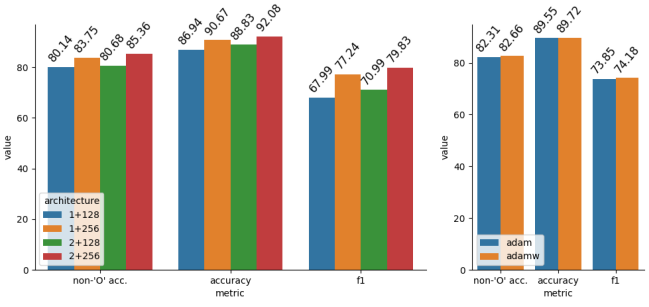

Figure 3. Choosing best configuration for bidirectional LSTM; 8 configurations, averaged for architecture (left) and optimizer (right), mean results over 10 iterations.

The context vector is a summary of the information from the input sequence, but differently from the final encoder hidden state it depends on the current decoder hidden state: by learning weights over the sentence, the decoder learns to discriminate which parts of it are most relevant at each timestep, both making it more efficient in extracting information and better in terms of model explainability. Attention weights are computed through a *scoring function* that takes as input the decoder hidden state h_t and one encoder hidden state h'_s (respectively at timesteps t and s); as suggested in the literature [1, 13], we have tested three different types of scores.

The first is the *concatenation* type: the two vectors h'_s and h_{t-1} are concatenated and fed into a fully connected network with one hidden layer and a $\tanh$ activation:

$$score\,(s,t) = \mathbf{v}_a^\top \tanh\left(\mathbf{W}_a h'_s + \mathbf{U}_a h_{t-1}\right)$$

It must be underlined that this type takes as input the previous decoder hidden state h_{t-1}, because the current one will be computed from the input and the context vector: the computation path is therefore $h_{t-1} \longrightarrow c_t \longrightarrow h_t$. Secondly, we test a *dot product* type, where the score is the dot product between the two vectors:

$$score\,(s,t) = h_t^\top \cdot h'_s$$

This type requires fewer parameters, therefore is helps in both keeping the model lightweight and reducing overfitting. In this case the computation is $h_t \longrightarrow c_t \longrightarrow \tilde{h}_t$, where the final vector is the one actually used for classification, but it is not fed into the recurrent cell. Finally, we implement a *local* alignment, where we exploit also the current token's index for an additional Gaussian re-weighting of the scores, according to a predicted focus position across sentence length. In fact, this latter method, since it involves only a re-weighting operation, can be combined with both the former score functions; we have tested it only for the *dot* type, since it is the most promising method.

Local attention requires one more step for computing the predicted focus position in the sentence ($p_t \in [0, S-1]$):

$$p_t = S \cdot \text{sigmoid}\left(\mathbf{v}_a^\top \tanh\left(\mathbf{W}_p h_t\right)\right)$$

$$local\,(s,t) = score\,(s,t) \exp\left(-\frac{(s-p_t)^2}{2\sigma^2}\right)$$

where σ is a parameter chosen empirically.

After computing those scores, proper alignment factors α_{ts} are obtained by rescaling the scores into a probability distribution with a softmax function. The newly-obtained context vector c_t is then concatenated with the decoder input and is fed either to the decoder (for *concatenation*) or to the classification layer (for the others).

4. Results

We will evaluate our models using the metrics of F_1 and mean accuracy across target classes both including and excluding the O tag. All models discussed here have been trained for 30 epochs with a batch size of 64 and a learning rate of 0.001, using early stopping for choosing the best model across each run; test results have been averaged across 10 iterations.

Together with the architecture, we tested three different sets of word embeddings: the one provided by SpaCy which is kept fixed during training; a GloVe [15] implementation in Pytorch, that is instead updated with model parameters; and a custom non-pretrained embedding over ATIS' vocabulary. No significant differences have been found, except for a slight advantage of GloVe; thus we have excluded the other possibilities for pruning hyperparameters.

In Figure 2 are shown results for the three cell types (all with 2 layers and hidden size of 256): we can see an evident effect of bidirecionality, that enhances every metric for every model. Interestingly, the RNN cell overfits in predicting the O tag without a bidirectional encoder, probably because without other architectural tricks it is not efficient in learning good hidden states. Due to its superior performances, we have chosen the LSTM cell for the next experiments. In a second experiment, we search the best LSTM configuration for number of layers (1, 2) and size of hidden states (128, 256): indeed the largest configuration is also the best scoring one, even though hidden-size seems to be more effective than multiple layers. In Figure 3 we also average across those configuration to find the best optimizer, but we do not find relevant differences by adding the weight decay constraint.

As shown in Figure 4, leveraging sentence-level information results in much higher scores, especially for the F_1; this is probably due to the fact that we constrain much more the dependency on input, improving hidden states with more information that both helps the classifier and reduces the problem of error propagation in predicted labels.

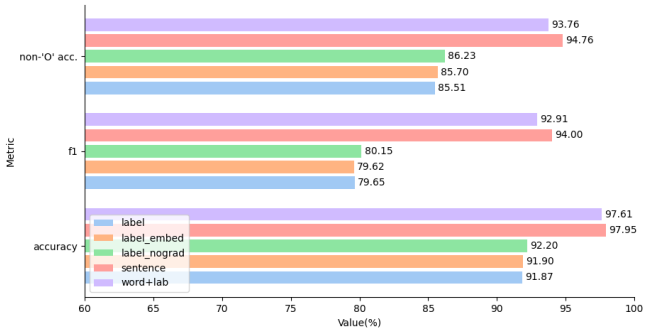

Figure 4. Different input modalities for decoder; average across 10 runs per each mode.

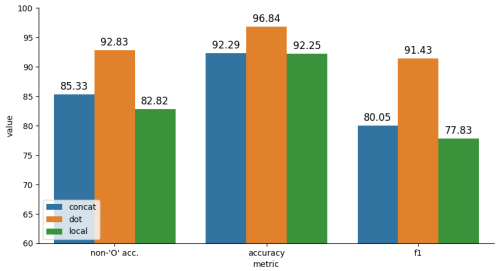

Figure 5. Performance per attention type on bidirectional LSTM (2+256); average across 10 runs per each mode.

This kind of architecture can be seen as an alternative to attention, where we force the decoder to focus on one specific part of the input; thus we have not included it in the analysis of attention.

Finally, we discuss results of different attention types. Global attention yields the best model, with an F_1 score of 91.43 over the test set. Lower scores for local attention could be due to a simple fact: this mechanism was designed for managing long sequences where the length of decoder output can be very different from the encoder's one, while in slot filling they have the same length and alignment is less complex than in other tasks; thus, local attention introduces complexity without improving results.

As for beam search, it yields only slight improvements on smaller beam sizes, as shown in Table 2: probably, in case of higher values more priority is given to the very common classes, which results in overfit sequences that do not generalize well.

Model	F_1
Simple CRF	89.72
CRF (Mesnil *et al.*, 2015) [14]	92.94
Bidir. LSTM	79.94
Sentence LSTM	94.00
W+L LSTM	92.91
DotLSTM	91.43
DotLSTM (beam = 2)	91.48
DotLSTM (beam = 4)	91.48
Dot-LSTM (beam = 6)	91.33
Dot-LSTM (beam = 8)	89.34

Table 2. Comparison with baselines. LSTMs are of the type 2+256. "DotLSTM" implements dot product attention.

5. Conclusions

In this project, we have evaluated several configurations for sequence-to-sequence models applied to slot filling: the effect of bidirectionality is evident both for simple and more complex models, as it is the inclusion of sentence-level information in the decoding phase; instead, there have not been found differences in the label input modality. Attention has a noticeable effect and it seems that the best choice is global attention; however, a wider research in the hyper-parameter space could yield better results also for local attention. Finally, beam search is able to slightly improve performances but only on limited beam sizes. Future directions of research could explore better sentence-level information, and eventually include Transformer-based architectures both in training or as frozen embeddings to use as backbone for other modules; furthermore, sampling methods could be implemented as decoding strategies to enhance effectiveness beam-based decoding.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015. 4

[2] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. 1, 2

[3] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 1, 2

[4] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *CoRR*, abs/1702.01806, 2017. 3

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 3

[6] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, 1990. 1

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. 1, 2

[8] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. 3

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014. 2

[10] Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. Leveraging sentence-level information with encoder LSTM for semantic slot filling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2077–2083, Austin, Texas, Nov. 2016. Association for Computational Linguistics. 1, 3

[11] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 2

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 2

[13] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. 4

[14] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2015. 2, 5

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. 4

[16] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014. 1, 3