How we Achieved a Production Ready Slot Filling Deep Neural Network without Initial Natural Language Data

François Torregrossa*†, Nihel Kooli*, Robin Allesiardo*, and Erwan Pigneul

Solocal/Pages Jaunes Digital/Search Tribe, Rennes, France
{ftorregrossa}|{nkooli}|{rallesiardo}@solocal.com

Abstract. Training deep networks requires large volumes of data. However, for many companies developing new products, those data may not be available and public data-sets may not be adapted to their particular use-case. In this paper, we explain how we achieved a production ready slot filling deep neural network for our new single-field search engine without initial natural language data. First, we implemented a baseline by using recurrent neural networks trained on expert defined templates with parameters extracted from our knowledge databases. Then, we collected actual natural language data by deploying this baseline in production on a small part of our traffic. Finally, we improved our algorithm by adding a knowledge vector as input of the deep learning model and training it on pseudo-labeled production data. We provide detailed experimental reports and show the impact of hyper-parameters and algorithm modifications in our use-case.

Keywords: Deep Learning · Slot Filling · Data Generation · Pseudo-Labeling · Knowledge Database

1 Introduction: a journey to a single-field search engine

PagesJaunes is a french search engine specialized in the search of local businesses. Historically, the search engine inputs are separated in two fields, the *WhoWhat* and the *Where*. To be able to understand queries in natural language, we replaced the two fields with a single-field (see Figure 1). This change is motivated by the democratization of dialogue systems available in smart-phones (Google Assistant, Siri), home-devices (Google Home, Amazon Echo) and the development of our own chat-bot. Being able to proceed queries in natural language is an important challenge to tackle to achieve a presence on those supports. However, the single-field is only a proof of concept and our back-end still requires

*the authors made equal contributions and are sorted at random

†François Torregrossa is also a member of The Research Institute on Informatics and Random Systems (IRISA) and of the Inria Rennes - Bretagne Atlantique research center

two separated fields. To be able to use the existing back-end, we needed to map the component of single-field queries to the *WhoWhat* and *Where* fields. The task of extracting sub-concepts from a sentence is commonly known as slot filling.

Fig. 1: The PagesJaunes single-field search engine

In the following sections, we relate the challenges we faced and how we overcame them. In section 2, we detail why we chose to implement our own slot filling solution instead of using a commercial one. In section 3 we show how we used generated synthetic natural language data to train a recurrent neural network (RNN) and the results we obtained by deploying it on production on a small part of our traffic. Then, in section 4, we present the improvements we made to our algorithm, by including features extracted from our knowledge data-bases and by auto-labeling the queries collected in production. Through the paper, we show how each modification impacted the performances of our solution. The complete results of our experiments are detailed in section 5.

2 Slot Filling

2.1 Problem Setting

The Slot Filling problem consists of extracting sub-concepts, defined as entities, from a semantic frame, namely a user query. This is commonly performed as a supervised sequence classification problem where tokens are associated with corresponding slots. Usually, the slot filling task is proceeded after an intent detection task, whose purpose is to understand the general meaning of the user query. However, in our proof of concept, only one intention is supported by default: the search of local businesses. Table 1 shows an example of query where the slots are labeled following the In/Out/Begin (IOB) representation [13].

Query	I	am	looking	for	a	vegan	restaurant	in	Paris
Slot	O	O	O	O	O	B-WhoWhat	I-WhoWhat	O	B-Where
Intent	FIND_BUSINESS								

Table 1: Example of slot filling task with IOB representations of the slots.

2.2 Commercial Solutions and State of the Art

Several commercial solutions can be used on slot filling tasks, such as Google DialogFlow* or Microsoft Luis†. Those softwares are based on the definition of intents and entities. Each intent is associated with several sentences, that provide examples of what a user could ask the system. Each of them can contains several slots that are filled with entities. For example, the sentence "I am looking for a

*Google DialogFlow, https://dialogflow.com/

†Microsoft Luis, https://www.luis.ai/

restaurant in Paris" can be associated with the intent FIND_BUSINESS where the slots *WhoWhat* and *Where* are respectively filled with the entities *restaurant* and *Paris*.

DialogFlow and Luis can accept a limited number of entities by slot, in the order of ten thousands. Our solution must be able to manage several million entities by slot. As seen in Table 2, those commercial solutions failed to provide acceptable performances when challenged on our use-case[‡]. The goal was to use the final solution inside our production environment, thus it had to achieve better performances. We chose to implement a customized solution based on state of the art research.

Algorithm	**Accuracy**
Google DialogFlow	0.74
Microsoft Luis	0.64

Table 2: Performances of commercial solutions on a slot filling task performed on hand labeled production queries. The accuracy is the proportion of queries where the slots have been successfully labeled.

The slot filling task is a challenging problem widely studied in the natural language understanding literature. One common method to address the slot filling task is to use expert knowledges, templates and dictionaries [14]. This approach has the advantage to be straightforward to implement but is domain dependent and can struggle to generalize on new domains. An alternative to those methods was to use conditional modeling algorithms, such as conditional random fields (CRF)[7, 15]. Several works [9, 8] challenged various network architectures against networks using recurrent layers. Models built with recurrent layers showed state of the art performances on the slot filling task and outperformed existing algorithms.

3 Deploying the First Model in Production

During this project, we implemented different algorithms from the literature: an algorithm based on our knowledge database; a CRF; several RNN based neural networks. Our experimentations confirmed the insight provided by the literature as the RNN based models outperformed the knowledge based algorithm and the CRF. In the following, we present the methodology used to obtain our first model without initial natural language data and the results obtained after deploying the RNN on a small traffic in our production environment.

3.1 Generation of synthetic training data from templates

The main challenge we faced when beginning this project was the absence of natural language data related to the search of local businesses, even unlabeled ones. Nearly exhaustive databases of businesses, keywords and localities were

[‡]Accuracies reported in Table 2 are obtained on our most recent testing set. At the beginning of the project, we challenged both solutions on a smaller set containing queries made by experts.

available but were only used in the two fields search. Taking example on DialogFlow, we implemented a text generator based on sentence templates where the slots would be filled by entities from our databases (see Fig 2).

(Give me|Tell me|What is) the (phone|number|num) of (BUSINESS_NAME).

Fig. 2: An example of template

Around fifty templates were defined by experts and added to the text generator. The main advantage of this approach is to know the label of each word by construction. Indeed, when generating a query, the indexes of each slot is known, as well as their type (*Who* *What* or *Where*). By using those templates we generated a labeled dataset of several million queries.

3.2 The Initial Slot Filling Deep Neural Network

In this subsection we detail the steps used to process the queries before feeding it to the network.

Word Embedding is a technique used to map textual data to dense real valued vectors. When learned with appropriate algorithms they produce vector space where distance encode semantic similarity. Algorithms using word embeddings as inputs show good generalization properties on natural language processing tasks [12]. FastText library [2] is used here for its capacity to create vector for unseen word during training. This is particularly crucial in our use-case as users can make mistakes when writing queries.

Query Tokenization consits of, in our case, extracting words in the query and normalizing them by removing punctuation, accents and uppercase. In some cases, those special characters contain an important part of the information by highlighting the *Who* *What* and the *Where* from the rest of the sentence. However, most of the real-world queries do not contain this sort of characters, so we did not consider them. Moreover, if the RNN achieves a high accuracy with those generic conditions, it would be able to cope with every way to write words.

Recurrent Neural Networks [9] are used for their good performances on the slot filling task. Actually, their ability to capture individual words and their contextual information is beneficial for the slot filling task. More specifically, bi-GRUs [3, 5] shown to be particularly promising among other tested RNN architectures on our use-case. The implementation details are listed in section 5.

3.3 Labeling Production Data to build a Validation/Test Dataset

We constituted a dataset of 6000 labeled single-field queries from production. Table 3 shows the accuracy of different baselines and of the retained RNN on this dataset. The RNN clearly outperformed the knowledge based algorithm and the CRF. The implementation of our knowledge based algorithm was quite naive and only looked for the largest matching expressions in our database. However, it showed unexpected performances and we decided to take advantage of those

knowledge by combining them with the RNN. This improvement is detailed in the next section.

Algorithm	Accuracy
Knowledge Based Algorithm	0.764
Conditional Random Fields	0.778
RNN	**0.840**

Table 3: Performances of several algorithms on a slot filling task performed on our use-case (details are provided in section 5). The accuracy is the proportion of queries where the slots have been successfully labeled.

4 Continuous Improvements

After the labelization of production data, we continued to improve our solution. We made two main improvements, adding a knowledge vector in input of the network (see subsection 4.1) and integrating pseudo-labeled production data to the training set (see subsection 4.2).

4.1 Taking advantage of our existing knowledge databases

During the review of miss-classified queries we noticed that, for some of them, human experts had to use database queries to know if some sub-concepts of the query were business names and/or localities. As the current model did not achieve human-like performances, we had the intuition that to be able to perform better, we could include information from our databases as inputs of the network. Indeed, as those concepts are quite rare in Wikipedia documents, using the only information from the embedding may not be enough to solve the task.

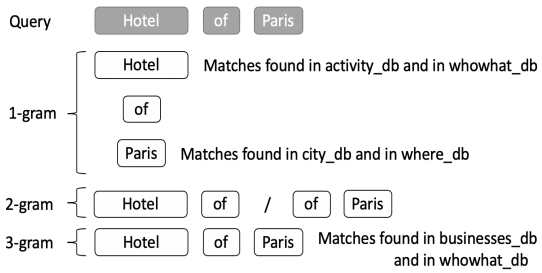

Fig. 3: Decomposition of a query in n-gram and search of matches in several database. In this example, activity_db contains the business activities, city_db the cities and whowhat/where_db the history of the inputs inside the *WhoWhat* and *Where* fields.

N-gram of words. A list of every word N-gram is generated from the user query and each database is queried to see if it contains the N-gram. Each of

them is thus associated with its frequency inside the *WhoWhat* and *Where* fields and its presence inside each database (businesses names and activities/keywords; the localities (cities/points of interest/addresses). See Figure 3 for an example of N-gram decomposition.

A Knowledge Matrix by token of the query is built by aggregating every information about it. The two dimensions of this matrix present: the size N of the N-grams ($N = 1, 2, 3, 4+$); the binning of frequencies inside the different fields (i.e. *WhoWhat* and *Where*) and the presence of the N-grams inside the different databases. Notice that features activated by an N-grams are activated on the knowledge matrix of all words compounding the N-grams. The knowledge matrix is finally flattened to form a **knowledge vector**. As shown in Figure 4, the knowledge vector is forwarded through a dense layer before being fed to the RNN to expose non-linear relationships between the raw knowledge features. We call one output of this layer a **knowledge embedding**.

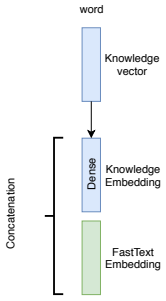

Fig. 4: The final architecture used by our RNN taking advantage of our knowledge database.

4.2 Achieving a better sentence embedding with pseudo-labeling

The main drawback of our model was the training set which is generated and does not include actual production data. Hand labeled data are not in sufficient quantities and are only used for validation and test. To include actual data, we create a pseudo-labeled data-set by labeling the production data with a RNN trained on the templates. This method is called pseudo-labeling [4] and achieves an effect similar to entropy regularization. It has the main advantage to allow the network to be trained on the actual distribution of queries. Indeed, while the sentences generated by the template are correctly labeled, the distribution of each type of queries may not reflect ground truth - actually, it does not. We train our network by incorporating pseudo-labeled data in the template dataset.

5 Experimentation

Optimizing Hyper-Parameters with Random Search. We optimize the hyper-parameters through a Random Search [1] : many networks are trained with sets of hyper-parameters sampled over distributions of Table 4. It has been shown that this approach is more efficient than grid-search or manual-search, especially when the hyper-parameter space is high dimensional.

Implementation details. The network is implemented in Pytorch [11]. Intermediate dense layers are activated through rectified linear units [10] and the prediction layer through a sigmoid function. The network is trained using the Adam optimizer [6]. The 6000 hand labeled queries are shuffled and separated into validation and test sets, each containing 3000 queries. Best models are selected using the accuracy on the validation set.

Parameter	Sampling Distribution	Description
batch_size	choice([128,256,512])	Batch size
learning_rate	log_uniform(10^{-5},10^{-1})	Learning Rate
knowledge_emb_size	int.uniform(32, 512)	Size of the knowledge embedding layer
concat_dropout	float.uniform(0.01, 0.4)	Dropout value of the concatenation layer
rnn_layers	choice([1, 2])	Number of RNN layers (bi-GRU)
rnn_size	int.uniform(32, 1024)	Size of the RNN layers (bi-GRU)
rnn_dropout	float.uniform(0.1, 0.4)	Dropout value of the RNN outputs
dense_size	int.uniform(32,1024)	Size of the dense layer after the RNN
dense_dropout	float.uniform(0.1, 0.4)	Dropout value of the dense layer after the RNN
pseudo_label_ratio	choice([0, 0.1, 0.2, 0.3, 0.4, 0.5])	Ratio of pseudo-labeled data in training
use_knowledge_vector	choice([True, False])	Availability of the Knowledge Vector

Table 4: The sampling distribution of each hyper-parameter.

Results. More than 300 networks were trained using the random search. Table 5 presents the accuracies of the best networks obtained through the random search for each combination of improvements. Without surprise, **the baseline RNN** shows a lower accuracy than the improved networks. Indeed, the models suffer from the absence of actual natural language data and from the large number of entity values (several million). On another side, networks trained with a part of **pseudo-labeled data** or with **knowledge vectors** as input data achieve an higher accuracy. Combining both shows significant improvement over other networks. More than half of the best networks trained using both **Knowledge Vectors and Pseudo-Labeling** outperform the networks obtained by training the RNN alone or by using only one of the improvements.

	Accuracy		
Algorithm	**Templates**	**Validation**	**Test**
RNN	0.936	0.824	0.840
RNN+PL	0.928	0.856	0.867
RNN+KV	**0.939**	0.857	0.866
RNN+KV+PL	0.921	**0.875**	**0.886**

Table 5: Recapitulation of the results obtained by the Recurrent Neural Network (RNN), the RNN with Knowledges Vectors (KV) as additional inputs, the RNN trained on templates and pseudo-labeled data and the combination of both.

We also observe that pseudo-labeled data decrease the accuracy of the networks on the data generated from templates. This can be explained by the difference of distributions between generated and production data. However, as seen previously, including pseudo-labeled data increases the accuracy of the models significantly on validation and test.

6 Conclusion

During this work, many obstacles were faced, including the lack of real word labeled data. We showed that on the slot filling task, a baseline good enough (when manually tested by experts) to be used on a proof of concept can be

achieved with data generated from templates. After manually labeling production data, we experimentally demonstrated that, on our use-case, combining pseudo-labeling with knowledge data allows to perform better. This work is an actual example of pseudo-labeling allowing to raise the performances of a network to a next level on a real world problem. Moreover, the use of random search was crucial in the search of a good combination of hyper-parameters. We think that the methodology implemented during this project will be useful to many practitioners trying to solve real world natural language understanding tasks.

References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. In: Journal of Machine Learning Research, vol. 13, 2012 (2012)
2. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. In: arXiv preprint arXiv:1607.04606 (2016)
3. Cho, Kyunghyun; van Merrienboer, B.G.C.B.D.B.F.S.H.B.Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing (2014)
4. Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization. In: NIPS (2004)
5. Graves, Alex; Schmidhuber, J.: Framewise phoneme classification with bidirectional lstm and other neural network architectures. In: IJCNN (2005)
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015 (2015)
7. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: ICML (2001)
8. Ma X., H.E.H.: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: End-to-end sequence labeling via bi-directional lstm-cnns-crf. In proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (2016)
9. Mesnil G., He X., D.L.B.Y.: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: INTERSPEECH (2013)
10. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: ICML 2010 (2010)
11. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS Workshop (2017)
12. R. Collobert, J. Weston, L.B.M.K.K.K., Kuksa, P.: Natural language processing (almost) from scratch. In: Journal of Machine Learning Research, vol. 12 (2011)
13. Sang, E.F.T.K., Veenstra, J.: Representing text chunks. In: Proceedings of the Ninth Conference on European Chapter of the Association for Computational Linguistics. pp. 173–179. EACL '99, Association for Computational Linguistics (1999)
14. Sun, A., Grishman, R., Xu, W., Min, B.: New york university 2011 system for kbp slot filling. In: TAC. NIST (2011)
15. Ye-Yi Wang, L.D., Acero, A.: Semantic frame based spoken language understanding. In: Chapter 3, Tur and De Mori (eds) Spoken Language Understanding: Systems for Extracting Semantic Information from Speech. pp. 35–80 (January 2011)