University of Trento - Language Understanding Systems Course

Final Project – Discriminative approach for sequence labelling through the use of CRFs and RNNs

Federico Marinelli, 187419, federico.marinelli@studenti.unitn.it

Abstract

The extraction of flat concepts out of a given word sequence is usually one of the first steps in building a spoken language understanding (SLU) or dialogue system. This project aims to compare the performance of labelling a word sequence using initially Conditional Random Fields (CRF) and then Recurrent Neural Networks (RNN). Will be used different features and parameters in order to improve the baseline performances, outlined on the spring- project of this course.

1 Introduction

The objective of this project is to develop a Spoken Language Understanding (SLU) for the movie domain. It can be seen as a sequential labelling problem that assign a domain-related tag to each word in an utterance. During the spring-project of this course I've modelled the the extraction of flat concepts out of a sentence using Stochastic Final State Transducers (SFSTs). In this final project I use discriminative approaches to model the concept labelling. In particular, I evaluate algorithms based Conditional Random Fields (CRF) and Recurrent Neural Networks (RNN). I compare them in terms of concept accuracy (and F1 measure), generalization and robustness to annotation ambiguities. I also show how non-local features (e.g. a-priori knowledge) can be modelled with CRF which is the best performing algorithm across tasks.

2 Dataset

The given dataset is named NL-SPARQL and it concerns the movie domain. It is already split in train and test files. In particular, there are two different instances of it: the first one is composed by a train-set and a test-set in token-per-line format, with tokens and concept tags for sequence labelling; the second one is composed by a train-set and a test-set with POS-tag and Lemmas in token-per-line format. This second data-set will be used to embed additional features for sequence labelling. Both of them have concepts that are represented through "BIO" encoding. The dataset used is the same as that I used in the spring-project and the full data analysis is available in the previous report.

3 Conditional Random Fields (CRFs)

3.1 General idea

In what follows, X is a random variable over data sequences to be labelled, and Y is a random variable over corresponding label sequences. A conditional random field [1] is defined by a dependency graph G and a set of features f to which are associated weights λ. The conditional probability of an annotation given an observation is given by:

$$p(y|x) = \frac{1}{Z(x)} exp(\sum_{c \in C} \sum_{k} \lambda_k f_k(y_c, x, c))$$

$$with$$

$$Z(x) = \sum_{y} exp(\sum_{c \in C} \sum_{k} \lambda_k f_k(y_c, x, c))$$

With the using of CRF it is possible to encode, into the model, a-priori concept relations and semantic features using the previous defined functions. Usually these features are represented by binary function (they return 1 is there is a match, 0 otherwise). The weights λ, associated to each features, are the parameters of the model. Learning a CRF is to compute the weights λ. Using CRFs it is possible to incorporate together the best of generative and classification models. In fact, they are able to work on many statistically correlated features from the input space and discriminate them during the training. Moreover, like generative models, they can trade off decisions at different sequence positions. With the using of conditional random fields, we can easy integrate domain knowledge as features, we can also model long dependences on the observation x and take global decision on a sequence. However, even if they represent a cutting-edge methodology for sequence labelling, they are slow to train and do not scale very well with large amounts of training data.

3.2 Implementation

The tool used in order to implement CRFs is the following:

- *CRF++:* it is an open source and customizable implementation of Conditional Random Fields segmenting/labelling sequential data [2].

3.3 Methodology and results

Within the CRF++ tool it is possible to define a template file in order to add and manage additional features for the model. The template file has one feature template per line with the following syntax *"%x [row, column]"*. Both row and column are integer numbers, the first one is the row relative positions with regard to current token, the second corresponds to the absolute position of the column. In particular, there are two kinds of templates: unigram and bigram; it is possible to define them using the letters U and B respectively in the template. CRF++ automatically generates feature function using such macros. In this section I report the most significant results that I obtained using different features set for training the conditional random fields. First of all, I started to search the best window size. For each scanned word w, the best classification influence was associated to w, the two words preceding w and the three words following w. At this point, before trying to add new features, I decided to find the best window size for output bigrams as well. After a few tests, I identified a window going from - 2 (B00:%x[-2,0]) to 2 (B04:%x[2,0]) as the best possible option.

Using POS as additional feature

After that I found the best windows size (both for unigram and bigram), I decided to add part-of-speech tag relative to each word as additional feature. In the following table (Table 3.1) it is possible to find the template that I used for training the conditional random fields and in the Table 3.2 I report the results obtained.

# Unigram	*#Bigram*
U00:%x[-2,0]	B00:%x[-2,0]
U01:%x[-1,0]	B01:%x[-1,0]
U02:%x[0,0]	B02:%x[0,0]
U03:%x[1,0]	B03:%x[1,0]
U04:%x[2,0]	B04:%x[2,0]
U05:%x[3,0]	
U06:%x[0,0]/%x[1,0]	
U07:%x[0,1]	

Table 3.1: Template used to add POS and Lemmas (separately) and additional features

Using POS as feature			
Accuracy	Precision	Recall	FB1
94.38%	86.44%	79.47%	82.81%

Table 3.2: Results obtained using POS as feature

Using Lemmas as additional feature

I used the lemma relative to each word as additional feature during the training. The template is the same as the previous one. The results are reported in the table 3.3.

Using Lemmas as feature			
Accuracy	Precision	Recall	FB1
94.35%	82.68%	79.56%	82.78%

Table 3.3: Results obtained using Lemmas as feature

Using Lemma and POS jointly as feature

In this subsection I report the results that I obtained using Lemma and POS jointly as additional features. In the following table (Table 3.4) it is possible to find the template that I used to train the conditional random fields. The table 3.5 shows the results obtained.

# Unigram	*#Bigram*
U00:%x[-2,0]	B00:%x[0,0]
U01:%x[-1,0]	B01:%x[-1,0]
U02:%x[0,0]	B02:%x[-2,0]
U03:%x[1,0]	B03:%x[1,0]
U04:%x[2,0]	
U05:%x[3,0]	
U06:%x[0,2]	
U07:%x[0,1]	
U08:%x[-1,2]	
U09:%x[0,0]/%x[0,2]	
U10:%x[0,0]/%x[1,0]	
U11:%x[-1,0]/%x[0,0]	
U12:%x[-1,2]/%x[0,2]	
U13:%x[-1,0]/%x[0,0]/%x[1,0]	

Table 3.4: Template used to add POS and Lemmas jointly as additional features

Accuracy	Precision	Recall	FB1
94.74%	86.97%	80.11%	83.40%

Table 3.5: Results obtained using POS and lemmas jointly

Suffixes and prefixes as additional feature

I added suffixes and prefixes as additional features. The template used is the same as the previous subsection (I've just specified in the template that with these additional features there are more columns to take in consideration). I tried different combination and the best performances were obtained using a suffix and a prefix of length 3 for each word. The results are reported in the Table 3.6.

Using Lemmas and POS jointly, suffixes and prefixes as feature			
Accuracy	Precision	Recall	FB1
94.72%	86.84%	80.48%	83.54%

Table 3.6: Results obtained using POS, lemmas, suffixes and prefixes (both of length 3)

4 Recurrent Neural Networks (RNNs)

Recurrent neural networks (RNNs) are a class of artificial neural network architecture that uses iterative function loops to store information. They can

be used to boil a sequence down into a high-level understanding, to annotate sequences, and even to generate new sequences from scratch. RNNs have several properties that make them an interesting choice also for sequence labelling: they are flexible in their use of context information (because they can learn what to store and what to ignore); they accept many different types and representations of data; and they can recognise sequential patterns in the presence of sequential distortions [4]. However, they also have several drawbacks, one of them is that it is very difficult to get standard RNNs to store information for long periods of time, which is of critical importance to sequence labelling. Another issue with the standard RNN architecture is that it can only access contextual information in one direction (typically the past).

4.1 General idea

A recurrent neural network could be seen as a Multi layer perceptron (MLP) where we relax the condition whose connections do not form cycles, and allow cyclical connection as well. An MLP can only map from input to output vectors, whereas an RNN can map from the entire history of previous inputs to each output. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. They are able to memorize previous input in the network's internal state, and thereby influence the network output. Here is what a typical RNN looks like:

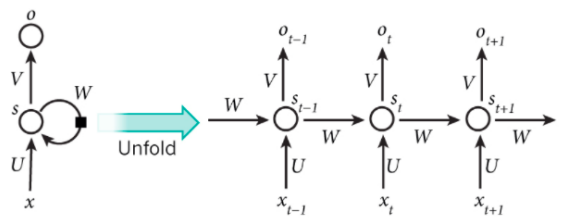

Figure 4.1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation.

The above diagram shows a RNN being unrolled into a full network. By unrolling we simply mean that we write out the network for the complete sequence.

- X_t is the input at time step t.
- S_t is the hidden state at time step t. It's the "memory" of the network.
- O_t is the output at step t.

During the course we have seen two types of RNNS that use recurrent connection: the Elman type, that feeds the activation of the hidden layer at previous step with the input; and the Jordan type, that feeds the activation of the output layer at previous time step with the input. For both of them, during the training, the network is unrolled in time backwards and backpropagation is applied, it's usually called Backpropagation Through Time (BPTT). Words are fed into the neural networks by using 1-of-n encoding. It's possible, in order to make model more accurate, to implement the word embedding, where the neural networks map the words (given in input) onto a d-dimensional continuous space and it's able to learn distributed representation for these words (similar words are mapped to close position at the space). In our task the output of the network is the probability for each class (concept-tag) for the current word.

4.2 Implementation

The code for the implementation of RNNs was given by the course teaching assistant and it's based on two paper of Grégoire Mesnil, Xiaodong He [5][6].
Using this implementation, it is possible to train and test RNNs of Jordan and Elman types. It implements also word-embedding. There is a configuration file that permits us to play with the hyper-parameters, I report all of them in the following table. (Table 4.1).

Parameter	Definition
lr	Learning rate value
win	Number of words in the context window
bs	Number of BPTT steps
nhidden	Number hidden units
seed	randomness
emb_dimension	Dimension of word embedding
nepochs	Maximum number of back-propagation steps

Table 4.1: Hyper-parameters of the RNNs

4.2 Methodology and results

Also for the RNNs I've used NL-SPARQL dataset. I've took the 10% of the training-set shuffled (300 sentences) in order to create the validation test, that is necessary during the training for the stopping-criterion.

Standard Recurrent Neural Networks
In this section I report the results that I've obtained using the basic implementation of the RNNs given by the course. I've tried different parameters settings and the best performance is reported in the following table.

RNNs Elman results			
Accuracy	Precision	Recall	FB1
95.39%	79.80%	78.70%	79.34%

Table 4.2: Results obtained by using RNNs Elman type

They are achieved using RNNs Elman type. The hyper-parameters used are reported in the following table.

Parameter	Value
lr	0.1
win	9
bs	5
nhidden	100
seed	38428
emb_dimension	100
nepochs	25

Figure 4.2: Configuration that gave me the best performances with the Elman type RNNs

LSTM Recurrent Neural Networks

Take for instance that we are trying to predict the last word in sentence "the clouds are in the sky"; we don't need any further context, it is pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information. But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in Itay…I speak fluent Italy." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of Italy. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. Long Short Term Memory networks are a special kind of RNN, capable of learning these long-term dependencies [7]. They were introduced by Hochreiter & Schmidhuber in 1997. LSTMs are explicitly designed to combat vanishing gradients through a gating mechanism.

I found an implementation of them(https://github.com/nifenggege/lstm- SequenceLabeling) and I've integrated the LSTM model into the code of RNNs given by the course. (I've edited the Elman model provided). In the following table it is possible to see the results that I've obtained.

Accuracy	Precision	Recall	FB1
95,47%	82,39%	77.78%	80.02%

Table 4.4: Results obtained by using LSTMs

The hyper-parameters used are reported in the following table.

Parameter	Value
lr	0.62714
win	9
bs	10
nhidden	100
seed	38428
emb_dimension	100
nepochs	25

Table 4.5: Configuration that gave me the best performances with the Elman type RNNs

The code of the LSTM model is available here: https://github.com/feedmari/Language-Understanding-System- Pro ject/blob/ master/final_project/rnn/rnn_slu/rnn/lstm.py

5 Evaluation and Conclusion

During the spring-project and the final-project I've adopted discriminative models (CRFs and RNNs) and generative models (SFSTs). While generative approaches models the joint distribution p(x, y), discriminative approaches focus solely on the posterior distribution p(y | x). The main difference is that discriminative models have only to model the conditional distribution and they completely disregard the prior distribution of the training samples p(x), this gives to discriminative models more freedom to fit the training data because they have to tune only the parameters that maximize p(y|x). Generally, in a classification task, discriminative models work better than generative ones, but many factor can influence them, like the size of the training data or the error present in the training. In such cases generative models have the advantage to use the prior probability in order to identify outlier sample and assign them lower probability.

The next table reports all the best results obtained during these 2 projects (Table 5.1).

Model	Accuracy	Precision	Recall	FB1	Training time
SFSTs	94.94%	82.35%	82.95%	82,65%	3 min
CRFs	94.72%	86.84%	80.48%	83.54%	10 min
RNNs	95.39%%	79.80%	78.70%	79.34%	30 min
LSTMs	95,47%	82,39%	77.78%	80.02%	40min

Figure 5.1: Final comparison between all models.

As it is possible to see from the image above, CRFs provided the best classification performances in term of F1 measure (**83,54**%) with the usage of different features (POS, lemma, prefixes, suffixes). The model with the best accuracy is the LSTMs with **95.47**%.

However, the distance between generative and discriminative models is not so big.
I've found that CRFs are really good for the task of this project, in fact we are able to incorporate a lot of additional features in order to improve the final performances. The RNNs and LSTMs is the worst model in term of F1 measure, by the way I think that with a larger dataset (more than 3338 sentences) they can perform better, potentially surpassing CRFs as shown in Bayer 2017 [8].
In term of training time I can confirm what expected, in fact generative models are very fast to train (2-3 minutes), instead discriminative models are slower, this is due to the fact that they have much more parameters and computation to do (e.g. backpropagation for the RNNs). To conclude, I think that with a larger corpus discriminative models can further outperform the generative ones in term of performances.

Appendices

The project can be found on GitHub at the following link: https://github.com/feedmari/Language-Understanding-System-Project

References

[1] Sutton C., An Introduction to Conditional Random Fields*, arxiv.org/pdf/1011.408.*
[2] CRF++, Taku Kudo, dof: *taku910.github.io/crfpp*
[3]Raymond C. and Riccardi G., Discriminative and Generative Algorithms for Spoken Language Understanding*, Proc. Interspeech, Antwerp, 2007*
[4] Alex Graves, Supervised Sequence Labelling with Recurrent Neural Networks.
[5]Grégoire Mesnil, Xiaodong He, Li Deng and Yoshua Bengio - *Investigation of Recurrent Neural Network Architectures and Learning Methods for Spoken Language Understanding*
[6]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*
[7]Christopher Olah, Undertanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[8] Ali Orkan Bayer, 2017. Recurrent neural network for language model and SLU.