Language Understanding Systems
Final project

Kateryna Konotopska
198234
kateryna.konotopska@studenti.unitn.it

Abstract

The final project regards concept tagging of speech text making use of recurrent neural networks and their comparison with conditional random fields and weighted final state transducers.
This document describes different architectures implemented, their results and main issues.

1 Introduction

The task consists in assigning concept tags from the movie domain to the speech text: given a set of sentences, assign for each word in a sentence a concept tag.
Recurrent neural networks are known to be suitable for sequential data. Thus in this project we want to investigate how some simple models (Elman, Jordan RNNs) and their improvements (GRU/LSTM) performs on our task.
Conditional Random Fields (CRF) is another state-of-art tool for named entity recognition. It can be useful to see the results of trying a probabilistic model for the given problem.
In the first project a simpler model with the weighted final state transducers was described and implemented, therefore it serves as additional baseline for the comparison.
The next sections are organized as follows: we analyze again the dataset keeping attention this time also on the dev and test sets, since the dev set is important for parameters tuning of our networks. The following sections describe each model architecture, followed by results and error analysis.

2 Methods

2.1 Data analysis

The description of the dataset, test and train sets have already been discussed in the first report, thus this time we will not report all the details.
Lets recall that the dataset is from the movie domain and contains sentences with concept tags in IOB format for each word of each sentence. Test set is taken from the same file as in the first project, while train set this time was split in train and dev sets, since it is useful to have a dev set for neural networks hyperparameter tuning. Since dataset is not big, train-dev split was done with 80-20% proportion.
It is important to have dev and test sets with the same distribution in order to archive similar performances on dev and test sets. The following table reports some statistics about the train, dev and test sets:

	train	**dev**	**test**
tokens	1559	805	1039
tags	38	35	39
unknown words with respect to the train set	0	169	272

Dev and test sets has also unknown to the test set tags which will never be learned in the training phase: B-movie.description, I-movie.release_region, I-actor.nationality, I-award.category, B-movie.type.

2.2 Neural network architectures

All neural networks used in this project are of recurrent type. In a recurrent neural network inputs and outputs are sequences which can be denoted as $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_n)$. Their main difference with the feed forward neural networks is the presence of edges connecting nodes of the network at time t with some nodes of the network at time $t-1$. This permits them to introduce the concept of time dependency.

2.2.1 RNN Elman type

Elman type RNN is the simplest neural network architecture implemented in this project: its hidden state represented by a single layer is a linear combination of current input and hidden state from the previous time step. This hidden state is then passed through the output nonlinearity.

$$\mathbf{h}_t = \tanh(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \quad (1)$$

2.2.2 RNN Jordan type

This type can be seen as a variation of Elman type network. In this case, at each time step, the output of the network on the previous step, additionally to its current input is used to feed the hidden state of the network. As in Elman type, the nonlinearity is applied on top of the linear combination:

$$\mathbf{h}_t = \tanh(W_{hx}\mathbf{x}_t + W_{ho}\mathbf{o}_{t-1} + \mathbf{b}_h) \quad (2)$$

One limitation of this network can be the dimension of the hidden layer: it cannot be larger than the output size.

2.2.3 GRU

Both Jordan and Elman type architectures have the problem of vanishing gradient on long sequences, as it frequently happens for the deep neural networks with large number of layers.
Gated Recurrent Unit (GRU) is the modification of the RNN hidden layer which adaptively forgets and remembers the flow of information inside the unit. The actual hidden state $\mathbf{h}_t$ is the interpolation between the previous hidden state and the candidate hidden state $\tilde{\mathbf{h}}_t$:

$$\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{(t-1)} + \mathbf{b}_{hr}) \\
\mathbf{z}_t &= \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{(t-1)} + \mathbf{b}_{hz}) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_{in} + \mathbf{r}_t(\mathbf{W}_{hn}\mathbf{h}_{(t-1)} + \mathbf{b}_{hn})) \\
\mathbf{h}_t &= (1 - \mathbf{z}_t)\tilde{\mathbf{h}}_t + \mathbf{z}_t\mathbf{h}_{(t-1)}
\end{aligned} \quad (3)$$

where $\mathbf{z}_t$ is the *update gate* which decides how much the unit updates its hidden layer and $\mathbf{r}_t$ is the *reset gate*, computed in similar way.

2.2.4 LSTM

Long Short Term Memory unit further improves long-term dependencies by introducing the *cell state* different from the hidden state, which permits to keep even longer time dependencies:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ii}\mathbf{x}_t + \mathbf{b}_{ii} + \mathbf{W}_{hi}\mathbf{h}_{(t-1)} + \mathbf{b}_{hi}) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{if}\mathbf{x}_t + \mathbf{b}_{if} + \mathbf{W}_{hf}\mathbf{h}_{(t-1)} + \mathbf{b}_{hf}) \\
\mathbf{g}_t &= \tanh(\mathbf{W}_{ig}\mathbf{x}_t + \mathbf{b}_{ig} + \mathbf{W}_{hg}\mathbf{h}_{(t-1)} + \mathbf{b}_{hg}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{io}\mathbf{x}_t + \mathbf{b}_{io} + \mathbf{W}_{ho}\mathbf{h}_{(t-1)} + \mathbf{b}_{ho}) \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{(t-1)} + \mathbf{i}_t\mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t)
\end{aligned} \quad (4)$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the previous layer at time $t-1$, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively, and σ is the sigmoid function.

2.3 Probabilistic models

2.3.1 CRF

Conditional Random Field model is a discriminative model that encodes a conditional distribution $P(\mathbf{Y}|\mathbf{X})$. A conditional random field is represented by a dependency graph and a set of features defined by the user to which are associated weights which are going to be learned from data. The posterior probability of each example is given by:

$$\begin{aligned}
P(y|x) &= \frac{1}{Z(x)} exp(\sum_{c \in C} \sum_{k} (\lambda_k f_k(y_c, x, c))) \\
Z(x) &= \sum_{y} exp(\sum_{c \in C} \sum_{k} (\lambda_k f_k(y_c, x, c)))
\end{aligned} \quad (5)$$

where f_k are semantic features which usually have value 1 if there is a match or 0 otherwise. Every feature function has as parameters random variables y_c of the clique c to which they apply, and the observation x. The prediction is done by maximizing this probability.

3 Results

3.1 Baseline

Results of all models are compared to the best result of the WFST model implemented in the first project, which gave an F1-score equal to 82.74%.

3.2 CRF

After investigation of parameters such as window size, unigram\bigram model for the output label, addition of prefixes and suffixes, frequency cut-off

and c hyperparameter (representing level of overfitting of the model to the train set) the best configuration gave F1-score of 83.8%.
For CRF evaluation CRF++ tool was used.

3.3 Neural network architectures

For training of the recurrent neural networks there were quite a lot of parameters to be tuned:

- **hidden layer size**: in case of bidirectional architecture this number was divided by two for each direction
- **learning rate**: learning rate of the optimizer, parameter of the gradient descent algorithm, the Adam optimizer was used in this project (with other parameters set to defaults)
- **word embeddings**: choice between word2vec embeddings and embeddings trained from random values (embeddings size tuning in this case)
- **batch size**: dimension of batch used to update gradient on each step
- **drop rate**: drop out rate of embeddings and network parameters applied on each step during training
- **token/tag drop**: in order to deal with unknown words some of the tokens and concept tags in training set were tagged as unknown words, drop rate indicates the probability of a token/tag to be tagged as unknown
- **epochs**: number of times a network "passes" through the training set; different epoch numbers were tried, with early stopping regularization to avoid overfitting on train set

To all parameters were assigned different ranges of values and, after computing all possible combinations of these parameters, a fixed number of them were extracted randomly for each architecture to be tried.
In all models the "unfrozen" word2vec embeddings lead to the best results, freeze options decide the possibility of the algorithm to learn during training, using word2vec as initialization values.
In the following table the best results found by each architecture are reported:

	hidden layer size	**learning rate**	**drop rate**	**token drop**	**tag drop**	**batch size**	**epochs**	**F1-score**
RNN Elman	300	0.001	0.7	0.0	0.0	100	30	75.2
Bi-RNN Elman	400	0.001	0.6	0.001	0.01	25	16	82.4
RNN Jordan	300	0.001	0.6	0.0	0.01	8	19	71.4
Bi-RNN Jordan	300	0.001	0.6	0.0	0.0	40	70	78.7
GRU	200	0.001	0.8	0.0	0.0	25	44	76.2
Bi-GRU	300	0.001	0.7	0.0	0.001	10	19	**84.6**
LSTM	300	0.005	0.7	0.0	0.0	100	30	76.6
Bi-LSTM	300	0.001	0.7	0.001	0.0	10	30	**84.9**

We can observe that the Elman architecture without update and forget gates of models such as GRU and LSTM obtained quite good results. It can be explained by the fact that all phrases of the dataset are short and we don't have the vanishing gradient problem. The Jordan architecture obtained the worst results as expected because of the hidden layer size limitation.
Both bidirectional GRU and bidirectional LSTM models outperformed baseline and CRF results.

3.4 Conclusions

We have seen how models with different approaches perform on the given task. WFST model is the one with the worst performance, however it has the smallest number of hyperparameters to be tuned.
CRF performed quite good with additional knowledge of the token prefixes and suffixes.
The best results were archived by the neural network models, but in this case we have a big number of hyperparameters: related to the architecture, training parameters, word embeddings features.

The following table summarizes the best results of the models:

model	configuration	#parameters	F1-score
WFST	improved model, smoothing = 'kneser_ney', ngram_order = 4	arcs = 27.834 final states = 2.348	82.7
CRF	token window = [-4,2] POS tag window = [-4,2] prefix window = [0,0], suffix window = [0,0] bigram option = True, default unigram conjunctions = previous/current token, current/next token, c = 25, a = CRF-L2, f = 1, m = 100	966.370	83.8
NN	Bi-LSTM, hidden layer size = 300 learning rate = 0.001, drop rate = 0.7, token drop = 0.001 tag drop = 0.0, batch size = 10 epochs = 30	1.761.042	84.9

4 Error analysis

In this section we discuss error analysis of the neural network models. As expected, categories such as **movie.release_region**, **award.category** and **movie.type** are not labeled correctly since they are not present in the train set.
Even when some tags were present in the train set, but in small number of occurrences, there would be a difficulty to use them for labeling words in the test set, as we have already seen in the WFST project. For example the **director.nationality** tag, which appeared in the train set only once, is not tagged correctly by none of the architectures, even in predicting the dev set tags with the highest F1-scores; usually it is confused with the **director.name** tag.
One of the tags with the lowest F1-score is **movie.gross_revenue**. It may be due to the fact that neural network treats numbers as separate words and cannot label this tag to numbers not which have not been seen in the train set, for instance only sequences containing word *million* have a chance to be recognized as **movie.gross_revenue**. There is a similar problem with **movie.release_date** tag. One possible solution to this problem can be the initial elaboration of all numbers by substituting them with one special token for numbers.
Another group of tags on which all models perform badly includes the names of **movie.language**, **country.name** and **movie.location** since they are often tagged to the same words.

5 GitHub project

Each neural network architecture was implemented with python and the PyTorch library.
The project can be found on GitHub: https://github.com/katerynak/lus_project2

References

Raymond, Christian / Riccardi, Giuseppe (2007): "Generative and discriminative algorithms for spoken language understanding", In INTERSPEECH-2007, 1605-1608.

Zachary Chase Lipton (2015): A Critical Review of Recurrent Neural Networks for Sequence Learning

Junyoung Chung and Çaglar Gülçehre and KyungHyun Cho and Yoshua Bengio (2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

PyTorch documentation, https://pytorch.org/docs/stable/nn.html#torch.nn