# Appunti su *natural language processing with recursive neural network*

**A.A. 2023/2024**

Rosso Carlo

# Contents

# 1 Introduction

Since in the second week I am asked to develop the RNN model and to check its accuracy over the *Sentiment Penn TreeBank* dataset, I took the `https://icml.cc/2011/papers/125_icmlpaper.pdf` paper, where it is introduces for the first time. These notes are a summary of such paper.
It seems like the new model discovered a recursive structure that helps identifying the units that an image or sentence contains, plus it identifies their relationships to form a whole.
Super interesting: the same algorithm can be used both to provide a competitive syntactic parser for natural language sentences and to outperform alternative approaches for semantic scene segmentation, annotation and classification.
So this notes are parted in two: the first part is about the application in the language field, which is the one I am going to use for the project, and the second part is about the application in the image field, which I will summarize just for completeness' sake. Finally, note that some parts of the NLP section might be needed to understand the following sections. On the other hand, the other sections are not needed to understand the NLP section.

# 2 Natural Language Processing

Words are first mapped into a semantic space and then they are merged into phrases in a syntactically and semantically meaningful order. The RNN computes:

- a score that is higher when neighboring word should be marged into a words' sequence. Note that the order of the words is important;
- a new semantic feature representation for the words' sequence;
- the label of the words' sequence.

RNN jointly learns how to parse and how to represent phrases in a continuous vector space of features. This allows to embed both single lexical units (words) and unseen, variable-sized phrases ina syntactically coherent order.

The proposed model uses deep learning for NLP, particularly it handles variable sized sentences in a natural way and captures the recursive nature of natural language. Furthermore, it jointly learns parsing decisions, categories for each phrase and phrase feature embeddings which capture the semantics of their consituents.

## 2.1 Input representation for natural language sentences

In order to efficiently use neural networks in NLP, neural language models map words to a vector representation. These representations are stored in a word embedding matrix $L \in \mathbb{R}^{n \times |V|}$, where $|V|$ is the size of the vocabulary and $n$ is the dimensionality of the semantic space. This matrix usually captures cooccurence statistics and its values are learned. Assume we are given an ordered list of $N_{words}$. Each word $i = 1, \ldots, N_{words}$ has an associated vocabulary index $k$ into the columns of the embedding matrix. Here is how to get the word embedding for the

$i$-th word:

$$\mathbf{w}_i = L\mathbf{e}_k \in \mathbb{R}^n \tag{2.1}$$

## 2.2 RNNs for Structured Prediction

In the discriminative parsing architecture, the goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is the set of all possibl ebinary parse trees. The input $x$ consists of two parts: a set of activation vectors, which represents the words in the sentence; and a symmetric adjacency matrix $A$, where $A_{ij} = 1$ if the $i$-th and $j$-th words are adjacent in the sentence, and $A_{ij} = 0$ otherwise.

We denote the set of all possible trees that can be constructed from an input $x$ as $\mathcal{T}(x)$. During the training phase, we have labels $l$ for all the words in the sentence. Using these lables, we can define an equivalence set of correct trees $Y(x, l)$. A tree is correct if all adjacent segments that belong to the same class are merged into one super node before merges occur with super segments of different classes. For training the language parser, the set of correct trees only has one element, the annotated groud truth tree: $Y(x) = \{y\}$.

#### Max-margin Estimation

It is defined a margin loss $\Delta(x, l, \hat{y})$ for proposing a parse $\hat{y}$ for input $x$ with labels $l$. We can formulate the margin loss by checking whether the subtree $subTree(d)$ underneath a nonterminal node $d$ in $\hat{y}$ is correct:

$$\Delta(x, l, \hat{y}) = \kappa \sum_{d \in N(\hat{y})} 1\{subTree(d) \notin Y(x, l)\} \tag{2.2}$$

where $N(\hat{y})$ is the set of all nonterminal nodes in $\hat{y}$, and $\kappa$ is a hyperparameter that controls the importance of the margin loss relative to the likelihood of the parse.

Given a training set, we search for a function $f$ with small expected loss on unseen inputs. We consider the following functions:

$$f_\theta(x) = \arg\max_{\hat{y} \in \mathcal{T}(x)} s(RNN(\theta, x, \hat{y})) \tag{2.3}$$

where $\theta$ are all the parameters needed to compute a score $s$ with an RNN. The highest the score, the more confident the model is that the parse is correct. We want to ensure that the highest scoring tree is in the set of correct trees: $f_\theta(x_i) \in Y(x_i, l_i)$, for all training examples $(x_i, l_i)$. Furthermore, we want the score of the highest scroring correct tree $y_i$ to be larger up to a margin defined by the loss:

$$s(RNN(\theta, x_i, y_i)) \geq s(RNN(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y}), \quad \forall i, \hat{y} \in \mathcal{T}(x_i) \tag{2.4}$$

Which leads to the following regularized risk function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} r_i(\theta) + \frac{\lambda}{2} ||\theta||^2 \tag{2.5}$$

where

$$r_i(\theta) = \max_{\hat{y} \in \mathcal{T}(x_i)} (s(RNN(\theta, x_i, \hat{y})) + \Delta(x_i, l_i, \hat{y}) - s(RNN(\theta, x_i, y_i))) \tag{2.6}$$

Indeed minimizing $r_i$ is equivalent to minimizing the margin loss, which means that $\hat{y} \to y_i$ is the correct transition.

## 2.3 Greedy Structure Predicting RNNs

We find a greedy approximation of the next best merge in the parse tree.
Using the adjacency matrix $A$, the algorithm finds the psirs of neighboring segments and adds their activations to a set of potential child node pairs:

$$C = \{[a_i, a_j) | A_{ij} = 1\} \tag{2.7}$$

Each pair of activations is concatenated and given as input to a neural network. The network computes the potential parent representation for these possible child nodes:

$$p_{(i,j)} = f(W[c_i; c_j] + b) \tag{2.8}$$

Given the parent representation, the score is computed with a row vector $W^{score} \in \mathbb{R}^{1 \times n}$:

$$s_{(i,j)} = W^{score} p_{(i,j)} \tag{2.9}$$

Training will aim to increse scores of good segment pairs and decrease scores of pairs with different labels, unless no more good pairs are left. Then a new column is added to the adjacency matrix, the one representing the new merged node. The process repeats until all pairs are merged and only one parent activation is left in the set $C$. Hence, the same network (with parameters $W$, b, $W^{score}$) is recursively applied until all vector pairs are collapsed.

The final score that we need for structure prediction is simply the sum of all the local decisions:

$$s(RNN(\theta, x, y)) = \sum_{d \in N(y)} s_d \tag{2.10}$$

## 2.4 Category Classifiers in the Tree

One of the main advantages of this approach is that each node of the tree built by the RNN has associated with it a distributed feature representation. You can leverage this representation by adding to each RNN parent node a simple softmax layer to predict class labels, such as part-of-speech tags or named entity labels:

$$label_p = softmax(W^{label} p) \tag{2.11}$$

When minimizing the cross-entropy error, of this softmax layer, the error will backpropagate and influence both the RNN parameters and the word representations.

## 2.5 Improvements for Language Parsing

Since in a sentence each word only has 2 neighbors you can use bottom-up beam search to find the best parse tree.
Since there is only a single correct tree, the second maximization in the objective of Equation 2.6 can be dropped.

# 3 Learning

The objective $J$ is not differentiable due to the hinge loss. Therefore, gradient descent is generalized via the subgradient method.

Let $\theta = (L, W, W^{score}, W^{label})$ be the set of the model's parameters. The gradient becomes:

$$\frac{\partial J}{\partial \theta} = \frac{1}{n} \sum_i \frac{\partial s(\hat{(}y_i)}{\partial \theta} - \frac{\partial s(y_i)}{\partial \theta} + \lambda \theta \tag{3.1}$$

where $s(\hat{y}_i) = s(\text{RNN}(\theta, x_i, \hat{y}_{max(\mathcal{T}(x_i))}))$ and $s(y_i) = s(\text{RNN}(\theta, x_i, y_{\max(Y(x_i, l_i))}))$. To backpropagate the error, it is used another's paper idea in which the error is split in two parts, to the left and to the right child (Goller & Kuchler, 1996).

# 4 Experiments

The only parameters to be tuned are:

- $n$ the size of the hidden layer;
- $\kappa$ the penalization term for incorrect parsing decisions;
- $\lambda$ the regularization parameter;

Finally, the chosen values are $n = 100$, $\kappa = 0.05$ and $\lambda = 0.001$.