Appunti del paper della seconda settimana del piano di lavoro

A.A. 2023/2024

Rosso Carlo

Contents

1 Introduction

In this paper they are discussed the differences between three models:

- Recursive Neural Networks (RNN);

- Matrix-Vector RNN (MV-RNN);

- Recursive Neural Tensor Network (RNTN).

Finally, it is introduced the Standford Sentiment Treebank (SST) dataset, which is used to evaluate the models. In particular, the paper focuses on the improvement of the RNTN model, which seems to be the best one among the three models:

- it is able to capture the sentiment of single phrases;

- it is the only model able to capture the effect of negation.

1.1 Semantic Vector Space

Semantic Vector Space for single words have been widely used as features. Particularly, it is interesting how to combine these features to obtain a representation of longer phrases properly.
Semantic vector spaces uses distributional similarities of single words. Often, co-occurrence statistics of a word and its context are used to describe each word. Though, such approach doesn't appropriately capture the differences in antonyms because they have similar co-occurrence statistics.
The problem can be solved using neural word vectors. Either way the models here describe are able to capture the sentiment of single phrases.

1.2 Compositionality in Vector Spaces

Most of the compositionality algorithms capture two word compositions. Somebody compute matrix representations for longer phrases and define composition as matrix multiplication. Somebody else analyze subject-verb-object triplets and find a matrix-based categorical model to correlate well with human judgments.

1.3 Deep Learning

The idea to relate inputs through three way interactions, parameterized by a tensor have been proposed for relation classification extending Restricted Boltzmann machines and as a special layer for speech recognition.

2 Recursive Neural Models

When an n-gram is given to the compositional models, it is parsed into a binary tree and each leaf node corresponds to a word (my professor doens't use just binary trees). Recursive neural models will then cmpute parent vectors in a bottom up fashion using different thpes of compositionality functions g.

2.1 Word Vector Representations and Classification

Each word is represented as a d-dimensional vector. We initialize all word vectors by randomly sampling each value from a uniform distribution: $\mathcal{U}(-r, r)$, where $r = 0.0001$. All the word vectors are stacked in teh word embedding matrix $L \in \mathbb{R}^{d \times |V|}$, where $|V|$ is the vocabulary size. Note that the L matrix is seen as a parameter that is learned jointly with the compositionality models.

For classification into five classes, we compute the posterior probability over labels given the word vector via:

$$y^a = \text{softmax}(W_s a) \tag{2.1}$$

where $W_s \in \mathbb{R}^{5 \times d}$ is the sentiment classification matrix, given a word vector representation a. The main difference between the models is the compositionality function g used to compute the parent vector from the child vectors l and r.

3 Recursive Neural Network

The simplest model of the three is the standard recursive neural network. First, it is determined which parent already has all its children computed (since it is a binary tree, it is only one). RNNs use the following equations to compute the parent vectors:

$$p = f\left(W \begin{bmatrix} l \\ r \end{bmatrix}\right) \tag{3.1}$$

where l and r are the left and right children of the parent, $f = \tanh$ is a standard element-wise nonlinearity, $W \in \mathbb{R}^{d \times (2d+1)}$ is the main parameter, where the $+1$ is for the bias term, note that it is needed to add a bias term to the input vectors. Each parent vector p_i is given to the same softmax classifier to compute its label probabilities.

4 Matrix-Vector RNN

The main idea of the MV-RNN is to represent every word and longer phrase in a parse tree as both a vector, $\vec{v}$, and a matrix, m. When two consituents are combined the matrix of one is multiplied by the vector of the other and vice versa.

Each word's matrix is initialized as a $d \times d$ identity matrix, plus a small amout of noise. The MV-RNN computes the first parent vector and its matrix via two equations:

$$p = f\left(W \begin{bmatrix} Rl \\ Lr \end{bmatrix}\right), \quad P = f\left(W \begin{bmatrix} L \\ R \end{bmatrix}\right) \tag{4.1}$$

where $W_M \in \mathbb{R}^{d \times 2d}$ and the result is again a $d \times d$ matrix. The vectors are used for classyfing each phrase using the same softmax classifier.

5 Recursive Neural Tensor Network

The main idea is to use the same tensor-based composition function for all nodes. The composition function is defined as follows:

$$p = f\left(\begin{bmatrix} l \\ r \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} l \\ r \end{bmatrix} + W \begin{bmatrix} l \\ r \end{bmatrix}\right) \tag{5.1}$$

So it is introduced a new tensor V that is used to compute the tensor product of the left and right children. The tensor W is defined as for the RNN. Here follows the multiplication with the dimensions:

$$d = (1 \times 2d) \times (2d \times 2d \times d) \times (2d \times 1) + (d \times 2d) \times (2d \times 1) \tag{5.2}$$

5.1 Tensor Backpropagation through structure

We assume the target distribution vector at each node has a 0-1 distribution econding. If there are C classes, then it has length C and a 1 at the correct label. All other entries are 0.
We want to minimize the cross-entropy error between the predicted distribution $y^i \in [0,1]^{C\times 1}$ at node i and the target distribution $t^i \in \{0,1\}^{C\times 1}$ at that node. This is equivalent to minimizing the KL-divergence between the two distributions.

Error computation

The error is calculated as follows:

$$E(\Theta) = \sum_i \sum_j t_j^i \log y_j^i + \lambda ||\Theta||^2 \tag{5.3}$$

where $\Theta = (V, W, W_s, L)$ is the set of all parameters in the model. Let's repeat, V is the nonlinear transformation of the children, W is is the linear transformation of the children, W_s is used to convert a word vector representation to a class in the form of y, and L is the dictionary, which makes the word vector representation given the word.
Apparently, i stand for the node index and j for the class index.
Note that $t_j^i \log y_j^i$ has a non-zero value only when $t_j^i = 1$. This is meaningful because we are only interested in the correct class. On the other hand $\sum_j y_j = 1$ because y is a probability distribution. Which means that increasing some j will decrease the others.

Softmax Gradient

Let $x^i \in \mathbb{R}^{d\times 1}$ be the word vector representation at node i. Each node backpropagates its error through to the recursively used weights V, W. Let $\delta^{i,s} \in \mathbb{R}^{d\times 1}$ be the softmax error vector at node i:

$$\delta^{i,s} = \left(W_s^T (y^i - t^i)\right) \otimes f'(x^i) \tag{5.4}$$

where $\otimes$ is the element-wise product and f' is the elemnt-wise derivative of f.

Tensor Gradient

The remaining derivatives can only be computed in a top-down fashion from the top node through the tree and into the leaf nodes. We define the complete incoming error for node i as $\delta^{i,com}$. The top node, only receives errors from the top node's softmax. Hence, $\delta^{root,com} = \delta^{root,s}$. For the derivative of each slide $k = 1, \ldots, d$ we get:

$$\frac{\partial E^{root}}{\partial V^{[k]}} = \delta_k^{root,com} \begin{bmatrix} l \\ r \end{bmatrix} \begin{bmatrix} l \\ r \end{bmatrix}^T \tag{5.5}$$

where $\cdot^{[k]}$ is the k-th element of the vector.

So we can compute the error of the two children of the root node as follows:

$$\delta^{root,down} = \left(W^T \delta^{root,com} + S\right) \otimes f'\left(\begin{bmatrix} l \\ r \end{bmatrix}\right) \tag{5.6}$$

where we define

$$S = \sum_{k=1}^{d} \delta_k^{root,com} \left(V^{[k]} + \left(V^{[k]}\right)^T\right) \begin{bmatrix} l \\ r \end{bmatrix} \tag{5.7}$$

Note the similarity between S and $W^T \delta^{root,com}$. S can be seen as the error that is propagated through the tensor V, which is 0 in the RNNs. Note that both W and V are used to combine the children, here they are used to propagate the error back, indeed $\delta^{root,down}, S \in \mathbb{R}^{2d \times 1}$.

Finally

$$\delta^{left,com} = \delta^{left,s} + \delta^{root,down}[1:d] \tag{5.8}$$

$$\delta^{right,com} = \delta^{right,s} + \delta^{root,down}[d+1:2d]. \tag{5.9}$$