ADVANCED LEARNING FOR TEXT AND GRAPH DATA

Lab session 3: 1D Convolutional Neural Network

Lecture: Prof. Michalis Vazirgiannis
Lab: Antoine Tixier and Jean-Baptiste Remy

Monday, October 28, 2019

This handout includes theoretical introductions, coding tasks and questions. Before the deadline, you should submit here a **.zip** file (max 10MB in size) containing a `/code/` folder (itself containing your scripts with the gaps filled) and an answer sheet named `firstname_lastname.pdf`, following the template available here, and containing your answers to the questions. Your answers should be well constructed and well justified. They should not repeat the question or generalities in the handout. When relevant, you are welcome to include figures, equations and tables derived from your own computations, theoretical proofs or qualitative explanations. **One submission is required for each student. The deadline for this lab is November 4, 2019 11:59 PM**. No extension will be granted. Late policy is as follows: $]0, 24]$ hours late $\rightarrow$ -5 pts; $]24, 48]$ hours late $\rightarrow$ -10 pts; > 48 hours late $\rightarrow$ not graded (zero).

---

1 Learning objective

In this lab, you will learn about a basic Deep Learning architecture for NLP: the 1D CNN. The original papers for this model are [2, 1]. We will use Python 3.6, **Keras 2.2.5** with **TensorFlow version <2** (e.g., 1.9.0) as backend.

We will be performing binary classification (positive/negative) on reviews from the Internet Movie Database (IMDB) dataset[1]. This task is known as *sentiment analysis* or *opinion mining*.

Note: the documents have already been preprocessed and turned into a list of integers starting from 2. The integers correspond to indexes in the vocabulary that has been constructed from the training set, in which the most frequent word has index 2 and 0 and 1 are reserved for the padding and out-of-vocabulary (OOV) tokens.

2 Loss

The loss that our model will learn to *minimize* is the *log loss*, also known as the *cross entropy*. In a binary classification setting with 0/1 classes, the log loss is defined as:

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N}\big(y_i \log p_i + (1-y_i)\log(1-p_i)\big) \tag{1}$$

[1] `http://ai.stanford.edu/~amaas/data/sentiment/`

Where N is the number of training examples, p_i is the probability assigned to class 1, $(1 - p_i)$ is the probability assigned to class 0, and y_i is the true label of the i^{th} observation (0 or 1). Only the term associated with the true label of each observation contributes to the overall score.

Question 1 (2 points)
For a given observation with true label 1, compute the log loss for a confident correct prediction, an unsure correct prediction, and a strongly incorrect prediction. What do you observe?

3 Model description

3.1 Input

A given document is represented as an ordered (by row) matrix $\mathbf{A} \in \mathbb{R}^{s \times d}$ of its word vectors drawn from an embedding matrix in $\mathbb{R}^{V+2 \times d}$, where s is the document length[2], V is the size of the vocabulary, and d is the dimensionality of the word embedding space.

Note: $\mathbf{A}$ is a 1D input in the spatial sense: it has only one spatial dimension (s), unlike an image which has two (height and width). The depth dimension, d, is called *channels* in computer vision.

3.2 Convolutional layer

Each instantiation of a 1D window slided over the input document is convolved with n_f filters of length h. Each filter is parameterized by a matrix $\mathbf{W} \in \mathbb{R}^{h \times d}$, initialized randomly and learned during training. Filters have one spatial dimension and extend fully through the depth dimension. The instantiations of the window over the input are called *regions* or *receptive fields*. If we slide the window by one word at a time (stride of 1), there are $s - h + 1$ receptive fields. The output of the convolution layer for a given filter is a vector $\mathbf{o} \in \mathbb{R}^{s-h+1}$, whose entries correspond to the output of the convolution of the filter with the corresponding region:

$$o_i = \mathbf{W} \cdot \mathbf{A}[i : i + h - 1, :] + b_W \tag{2}$$

Where $\mathbf{A}[i : i + h - 1, :] \in \mathbb{R}^{h \times d}$ is the i^{th} region matrix, and $\cdot$ is an operator returning the sum of the row-wise dot product of two matrices. An illustration is given in Fig. 1.

Question 2 (3 points)
What is the missing value in Fig 1? Detail how you get your result.

Then, a nonlinear activation function f, such as `ReLU` ($\max(0, x)$) or `tanh` ($\frac{e^{2x}-1}{e^{2x}+1}$), is applied element-wise to $\mathbf{o}$, returning what is known as the *feature map* $\mathbf{c} \in \mathbb{R}^{s-h+1}$ associated with the filter:

$$c_i = f(o_i) + b \tag{3}$$

Where $b \in \mathbb{R}$ is a trainable bias.

For text, generally, $h \in [1, 10]$, and $n_f \in [100, 600]$ [5]. Using multiple filters allows the model to learn different, complementary features for each region.

Since each filter generates a feature map, each region is thus embedded into an n_f-dimensional space. Moreover, using regions of varying size around the optimal one improves performance [5]. In

[2] s is fixed at the collection level, but since the documents are of different sizes, we need to use truncation and padding.

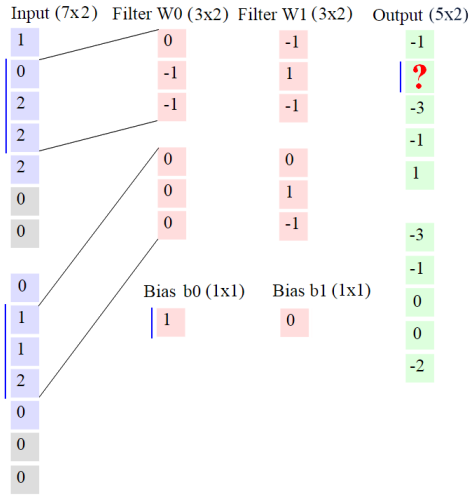

Figure 1: Illustration of unidimensional convolutions. Adapted from the 'Convolution Demo' section of Stanford's `CS231n` CNN course notes: `http://cs231n.github.io/convolutional-networks/#conv`

that case, different parallel branches are created (one for each region size), and the outputs are concatenated after pooling, as shown in Fig. 2.

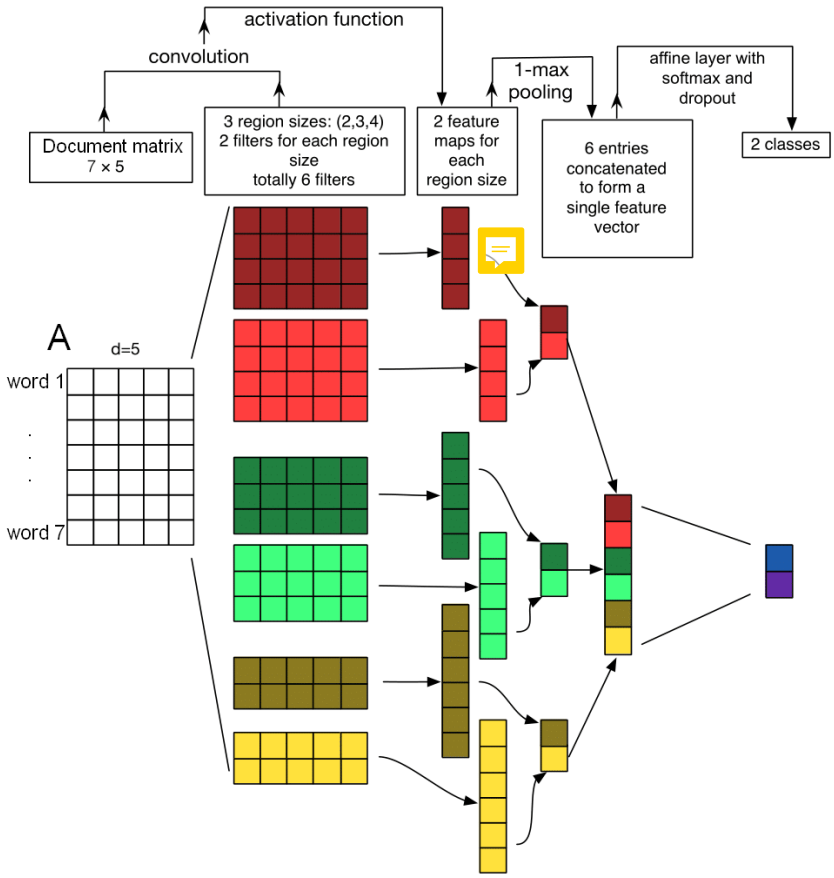

Figure 2: Toy example of 1D CNN architecture for document classification, with an input document of size $s = 7$, word vectors of dimensionality $d = 5$, 3 branches corresponding to filters of sizes $h = \{2, 3, 4\}$, $n_f = 2$ filters per branch, 1-max global pooling, and 2 categories. Adapted from [5].

Pooling layer. The assumption is that the exact positions of the features in the input document do not

matter. What matters is only whether certain features are present or absent. For instance, to classify a review as positive, whether "best movie ever" appears at the beginning or at the end of the document is not important. To inject such robustness into the model, *global max pooling* is employed. This approach extracts the greatest value from each feature map and concatenates them.

Softmax layer. The final embedding of the document is passed to a softmax layer, as the task we are interested in here is classification. The softmax outputs a *probability distribution* over the categories:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{4}$$

Question 3 (3 points)
What other activation function could we use at the output layer? Remember that we are performing binary classification. What are the consequences in terms of number of units in the final layer?

Task 1
Fill the gaps in the `cnn_branch()` function and 'defining architecture' section of the `main.py` script to implement the model.

Question 4 (4 points)
What is the formula for the total number of trainable parameters of the model, assuming that we use one branch with filters of size h, the word embeddings are updated during training, and we perform binary classification? Do not give a number, but a formula using the variables that have been defined in the current section.

4 Visualizing and understanding inner representations and predictions

4.1 Document embeddings

An easy way to verify that our model is effectively learning is to check whether its internal document representations make sense. Recall that the feature vector which is fed to the softmax layer can be seen as an n_f-dimensional embedding of the input document.

Task 2
Fill the gaps in the 'visualizing doc embeddings' section to extract the internal document representations of the model.

Question 5 (3 points)
Show and interpret your figures of the document embeddings before and after training.

4.2 Predictive regions identification

This approach is presented in section 3.6 (Tables 5 & 6) of [1]. Recall that before we lose positional information by applying pooling, each of the n_f filters of size h is associated with a vector of size $s - h + 1$ (a feature map) whose entries represent the output of the convolution of the filter with the corresponding receptive field in the input, after application of the nonlinearity and addition of the bias.

Therefore, each receptive field is embedded into an n_f-dimensional space. Thus, after training, we can identify the regions of a given document that are the most predictive of its category by finding the fields that have the highest norms. For instance, some of the most predictive regions for negative IMDB reviews are: "worst movie ever", "don't waste your money", "poorly written and acted", "awful picture quality". Conversely, some regions very indicative of positivity are: "amazing soundtrack", "visually beautiful", "cool journey", "ending quite satisfying"...

Task 3
Fill the gaps in the 'predictive text regions' section to identify predictive regions.

4.3 Saliency maps

Another way to understand how the model is issuing its predictions was described by [4] and applied to NLP by [3]. The idea is to rank the elements of the input document $A \in \mathbb{R}^{s \times d}$ based on their influence on the prediction. An approximation can be given by the magnitudes of the first-order partial derivatives of the output of the model $\text{CNN} : A \mapsto \text{CNN}(A)$ with respect to each row a of A:

$$\text{saliency}(a) = \left| \frac{\partial(\text{CNN})}{\partial a} \, |_a \right| \tag{5}$$

The interpretation is that we identify which words in A *need to be changed the least to change the class score the most*. The derivatives can be obtained by performing a single back-propagation pass (based on the prediction, not the loss).

Task 4
Fill the gaps in the 'saliency map' section to implement the method described above.

Question 6 (2 points)
Show and briefly interpret your saliency map.

Question 7 (3 points)
What are some limitations of the CNN model?

References

[1] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

[2] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[3] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[4] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[5] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.