Advanced Learning for Text and Graph Data
Lab 4: Deep Learning for NLP (1/2)

Lecture: Prof. Michalis Vazirgiannis
Lab: Antoine Tixier and Kostas Skianis

December 18, 2018

1 Introduction

In this lab, we will get familiar with a basic Deep Learning architecture for NLP: the 1D CNN. We will use Python 3.6 and `Keras`[1] (version 2.2.0), a very popular Python DL library. Code can be found in the `main.py` script. As usual, fill the gaps wherever needed, and submit your script here: https://goo.gl/forms/V7pCOxfkELWixzNx1. **Note**: you must use Keras with the `TensorFlow` backend!

2 IMDB movie review dataset

We will be performing binary classification (positive/negative) on reviews from the Internet Movie Database (IMDB) dataset[2]. This task is known as *sentiment analysis* or *opinion mining*. The IMDB dataset contains 50K movie reviews, labeled by polarity (pos/neg). The data are partitioned equally into training and testing, and into positive and negative. In the entire collection, no more than 30 reviews are allowed for any given movie, and the train and test sets contain a disjoint set of movies.

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

2.1 Binary classification objective function

The objective function that our model will learn to *minimize* is the *log loss*, also known as the *cross entropy*[3]. More precisely, in a binary classification setting with 2 classes (0 and 1) the log loss is defined as:

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

Where N is the number of training examples, p_i is the probability assigned to class 1, $\big(1-p_i\big)$ is the probability assigned to class 0, and y_i is the true label of the i^{th} observation (0 or 1). You can see that only the term associated with the true label of each observation contributes to the overall score. For a given observation, assuming that the true label is 1, and the probability assigned by the model to that class is 0.8 (quite good prediction), the log loss will be equal to $-\log(0.8) = 0.22$. If the prediction is slightly worse, but not completely off, say with $p_i = 0.6$,

[1] https://keras.io/getting-started/faq/
[2] http://ai.stanford.edu/~amaas/data/sentiment/
[3] https://keras.io/objectives/

the log loss will be equal to 0.51, and for 0.1, the log loss will reach 2.3. Thus, the further away the model gets from the truth, the greater it gets penalized. Obviously, a perfect prediction (probability of 1 for the right class) gets a null score.

3 Convolutional Neural Networks (CNNs)

3.1 Convolution and pooling

Input. We can represent a document as a real matrix $A \in \mathbb{R}^{s \times d}$, where s is the document length, and d is the dimension of the word embedding space. Since s is fixed at the collection level but the documents are of different sizes, we truncate the longer documents to their first s words, and pad the shorter documents with a special zero vector as many times as necessary. The word vectors may either be initialized randomly or be pre-trained, and can be further tuned during training or not ("non-static" vs. "static" approach [4]).

Convolution layer. The convolution layer implements a linear operation followed by a nonlinear transformation. The linear operation consists in multiplying (elementwise) each instantiation of a 1D window applied over the input document by a *filter*, represented as a matrix of parameters $W \in \mathbb{R}^{h \times d}$. Note that the filter, just like the window, has only one spatial dimension, but it extends fully through the input depth (the d dimensions of the word embedding space). W is initialized randomly and learned during training.

The instantiations of the window over the input are called *regions* or *receptive fields*. There are $(s-h)/\text{stride} + 1$ of them, where stride corresponds to the number of words by which we slide the window at each step. With a stride of 1, there are therefore $s - h + 1$ receptive fields. The output of the convolution layer for a given filter is thus a vector $o \in \mathbb{R}^{s-h+1}$ whose elements are computed as:

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

Where $A[i : i + h - 1, :] \in \mathbb{R}^{h \times d}$ is the i^{th} region matrix, , and $\cdot$ is an operator returning the sum of the row-wise dot product of two matrices[4]. Note that for a given filter, the same W is applied to all instantiations of the window regardless of their positions in the document. In other words, the parameters of the filter are shared across receptive fields. This is what gives the spatial invariance property to the model, because the filter is trained to recognize a pattern wherever it is located. It also greatly reduces the total number of parameters.

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

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

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

For short sentence classification, best region sizes are generally found between 1 and 10, and in practice, n_f filters (with $n_f \in [100, 600]$) are applied to each region to give the model the ability to learn different, complementary features for each region [9]. 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 [9]. In that case, different parallel branches are created (one for each region size), and the outputs are concatenated after pooling, as shown in Figure 1.

[4] see 'Convolution demo' here: http://cs231n.github.io/convolutional-networks/#conv

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 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're interested in here is classification. The softmax outputs a *probability distribution* over the categories:

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

In the case of binary classification, the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ can be used instead of the softmax.

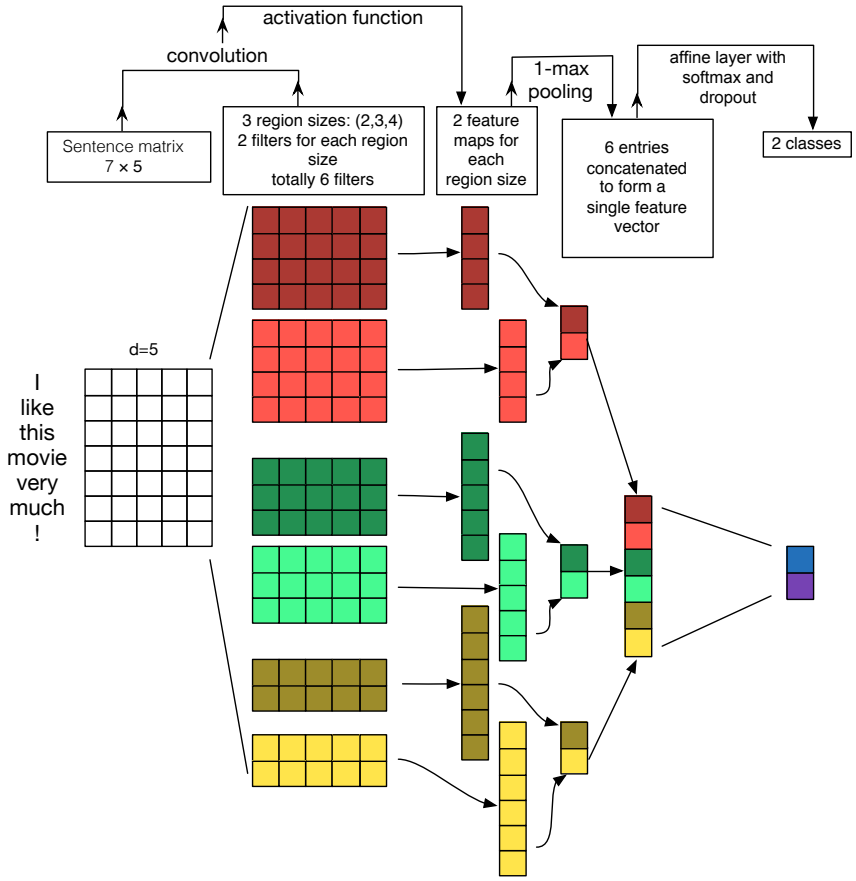

Figure 1: CNN architecture for document classification, taken from Zhang and Wallace (2015) [9]. $s = 7$, $d = 5$. 3 regions of respective sizes $h = \{2, 3, 4\}$ are considered, with associated output vectors of resp. lengths $s-h+1 = \{6, 5, 4\}$ for each filter (produced after convolution, not shown). There are 2 filters per region size. For the three region sizes, the filters are resp. associated with feature maps of lengths $\{6, 5, 4\}$ (the output vectors after elementwise application of f and addition of bias).

3.2 Visualizing and understanding inner representations and predictions

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. We can thus visualize whether there is correlation between document embeddings and labels. Figures 2 and 3 prove that indeed, our model is learning meaningful representations.

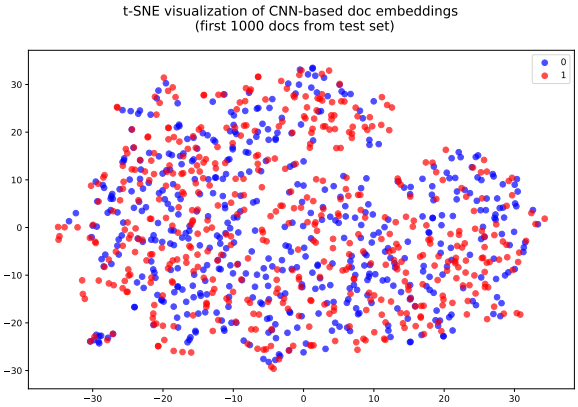

Figure 2: Doc embeddings before training.

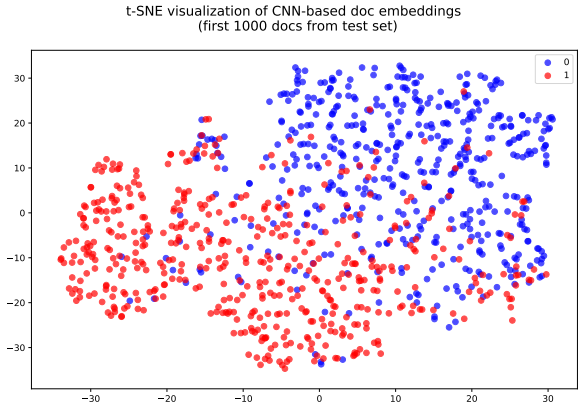

Figure 3: Doc embeddings after 2 epochs.

Predictive regions identification. This approach is presented in section 3.6 (Tables 5 & 6) of [3]. 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 ${}^{(s-h)}/_{\text{stride}} + 1$ (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"...

Saliency maps. Another way to understand how the model is issuing its predictions was described by [8] and applied to NLP by [7]. 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). Figures 5 and 4 show saliency map examples for negative and positive reviews, respectively.

Questions. Note: just as an in-class activity. You don't have to submit your answers.
- what is the formula for the total number of parameters of the model?
- what are some limitations of the CNN architecture?

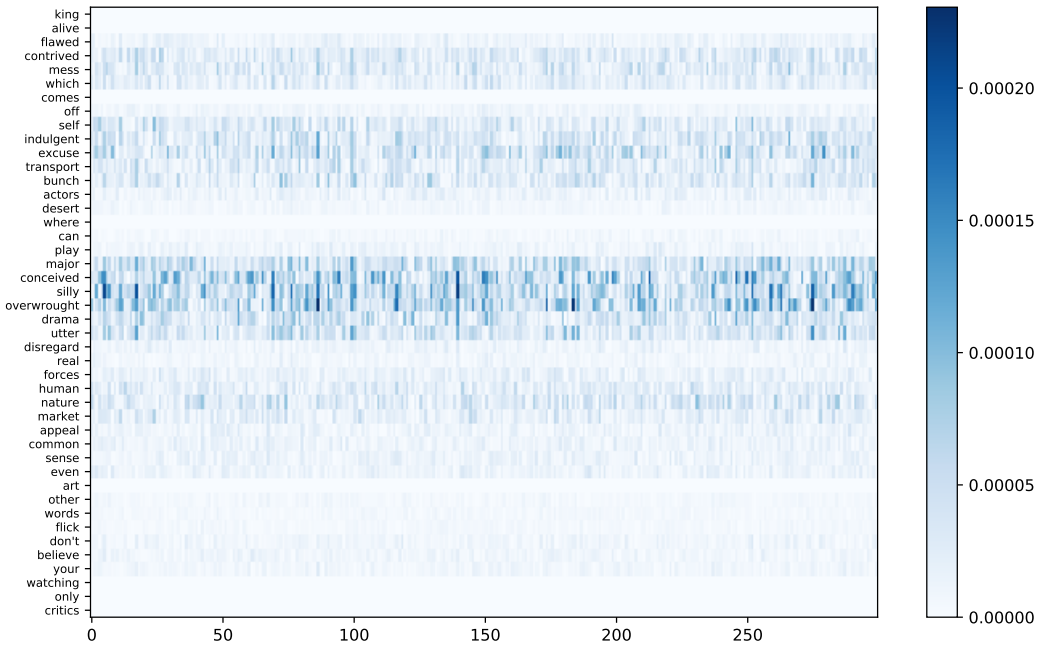

Figure 4: Saliency map example (true label: negative)

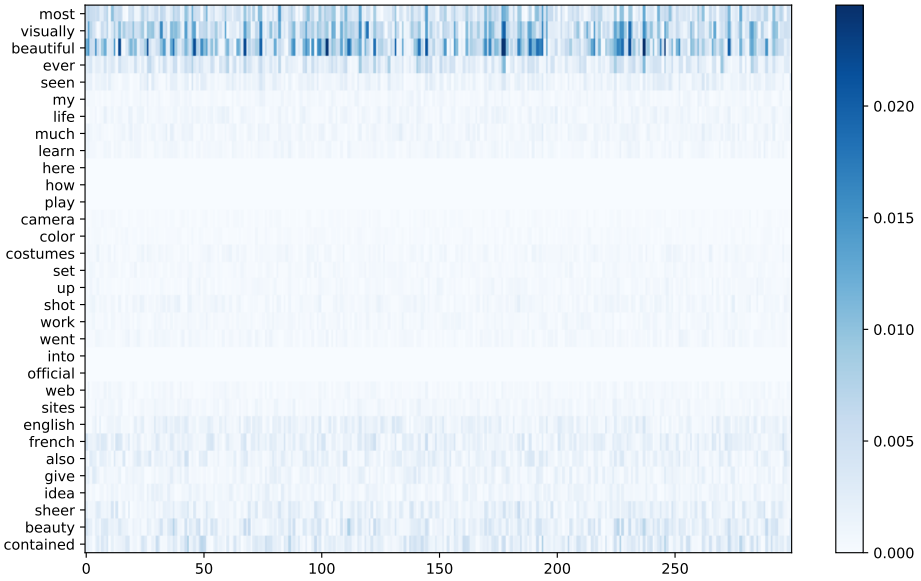

Figure 5: Saliency map example (true label: positive)

References

[1] Goldberg, Y. (2015). A primer on neural network models for natural language processing. Journal of Artificial Intelligence Research, 57, 345-420.

[2] Hubel, David H., and Torsten N. Wiesel (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 160.1:106-154.

[3] Johnson, R., Zhang, T. (2015). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. To Appear: NAACL-2015, (2011).

[4] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[6] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[7] Li, J., Chen, X., Hovy, E., and Jurafsky, D. (2015). Visualizing and understanding neural models in nlp. arXiv preprint arXiv:1506.01066.

[8] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034.

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