ENS CACHAN, MASTER MVA

Deep Learning for NLP

Word and sentence embeddings
Sentence Classification with LSTMs/ConvNets

Alexis Conneau

2016/2017

1 INTRODUCTION

The first part of the assignment will guide you through the use of **word2vec** for word embeddings. We will explore the word embedding space, and we will see how we can generate sentence embeddings from word embeddings. The 2nd and 3rd part of the assignment will introduce two types of Deep Learning models to tackle the problem of **sentence classification**:

- **Recurrent Neural Networks (LSTMs)**
- **Temporal Convolutional Neural Networks (1D ConvNets)**

We will use Python and two of its packages **Gensim** (for word2vec) and **Keras** (LSTMs/ConvNets). For all the questions in the assignmnent, **short answers are preferred**. We expect a **a zip/tar.gz/.tgz file** named "surname_firstname_deliverable_NLP", containing a **pdf**, and **python scripts (*only your code*)**.
The first part will be on the installation of Keras and gensim. The installation of Keras (and the setting of the theano backend) can be tricky, especially for windows users. Make sure you are able to install the python packages before going to part 3.

2 GETTING STARTED WITH PYTHON AND KERAS

Python is among the most important languages for data science because of its simplicity, its large community and especially for its useful packages (**scikit-learn** for Machine Learning, **nltk** for NLP, **Pytorch/Theano/TensorFlow/Keras** for Deep Learning etc). In this assignment, we will guide you through the installation of python and its packages : gensim (word2vec part) and Keras (sentence classification part).
Among the well-known frameworks for Deep Learning (Pytorch, Torch, Caffe, TensorFlow, Theano), Keras is the most famous in Kaggle competitions because it enables easy and fast prototyping. In the following, we guide you through the installation of python and these packages.

2.1 INSTALL PYTHON

1. Download the latest[1] **Anaconda distribution for python 2.7** : from the link HERE.
2. Choose **python 2.7** and the graphical installation.
3. Download the Pycharm IDE from the link HERE (for windows users : "create associations .py" when installing).
4. Open Pycharm, and follow these instructions:
   a) "Create a new project".
   b) Choose the Anaconda interpreter (**IMPORTANT!**).
   c) Create a new python script ("File", "New...", "Python file", and type the name of your file).
   d) Write in this script : print 'Hello World'
   e) Run the script ("Run" option in the menu bar, and then "Run").

   Please first ask a colleague for help if you have troubles.

2.2 PYTHON

If you are new to python, don't worry, with the right IDE (PyCharm for instance) the environment is very similar to the one of Matlab. Take some time to read this short tutorial on Python : Python in 10 minutes.

2.3 INSTALL THE PYTHON PACKAGES GENSIM AND KERAS

Gensim and Keras are "packages" of python. It means that these are libraries written in python and available to anyone. In a python script, you can "import" those packages to use them. For instance "import gensim" will import the gensim package. Installing them is rather easy with the "pip" command line. In the following, we guide you through the installation:

[1] The latest python version includes the easy package installation tool : "pip"

1. Open a terminal
   - **Windows users** : how to open the command prompt in Windows
   - **Linux/MAC users** if you probably are on MAC/Linux you already know how to open the terminal.

2.3.1 INSTALL GENSIM (WORD2VEC)

2. Type in the terminal : **pip install gensim** (or try **sudo pip install gensim**).
3. In the python file you created, type **import gensim**. Run the file. If there is no error, gensim has been installed properly.

2.3.2 INSTALL KERAS (DEEP LEARNING FRAMEWORK)

4. Type in the terminal : **pip install keras==1.2.0** (or try **sudo pip install keras==1.2.0**).
5. In the python file you created, type **import keras**. Run the file. If there is no error, keras has been installed properly.
6. You now need to change the backend from tensorflow (default) to theano to avoid code errors:
   - **Windows users** : Type "**notepad %USERPROFILE%/.keras/keras.json**" in the command prompt, it will open a file. In this file, change the line **"backend": "tensorflow"** to **"backend": "theano"**.
   - **Linux/MAC users** : in the script "~/.keras/keras.json", change **"backend": "tensorflow"** to **"backend": "theano"**.
7. Once you've done it : "import keras" should output the following line "Using Theano backend", and run without problem.[2]

   https://github.com/fchollet/keras/blob/master/docs/templates/backend.md

That's it : you have python, gensim and keras installed[3]. **Congrats!**. Now that this is done, follow the instructions of the next sections.

For the assignment, you will need to report your results in a pdf file, and include the python scripts (we won't try to debug your scripts so please double-check before submitting it).

[2] Note that the windows installation of python packages can be tricky. Make sure this works before going further in the assignment.

[3] You can even "pip install sklearn" and take a look at the Machine Learning package that is scikit-learn.

3 WORD AND SENTENCES EMBEDDINGS WITH WORD2VEC

You are provided with the following script : **embedding_word2vec_students.py**. In this script, we import gensim, and from gensim we import the word2vec functionality. The 'first part' trains the model, but you don't need to train it until the end. You are provided with "text8.model" and "text8.phrase.model" in the "models/" directory, which are pretrained word2vec models that you will load in 'second part'.

QUESTION 1 Run the **"first part"** (you don't need to finish the training of the model). From the INFO that is shown while training :

1. What is the total number of raw words found in the corpus?
2. What is the number of words retained in the word2vec vocabulary (with default min_count = 5)?

QUESTION 2 The **"second part"** loads two models, model and model_phrase.

1. What is the similarity between ('apple' and 'mac'), between ('apple' and 'peach'), between ('banana' and 'peach')? In your opinion, why are you asked about the three previous examples?
2. What is the closest word to the word 'difficult', for 'model' and 'model_phrase'. Comment about the difference between model and model_phrase. Find the three phrases that are closest to the word 'clinton'.
3. Find the closest word to the vector "vect(france) - vect(germany) + vect(berlin)" and report it similarity measure.
4. Explore the embedding space using these functions and report some interesting behaviour (of your choice).

QUESTION 3 The **"third part"** proposes a way to construct sentence embeddings from word embeddings, by simply taking the mean of the word embeddings contained in the sentence. The "avgword2vec" vector of a sentence is computed :

$$\boxed{\text{avgword2vec(s)} = \frac{\sum_{i=1}^{S} \alpha_i \text{word2vec}(w_i)}{\sum_{i=1}^{S} \alpha_i}} \tag{3.1}$$

with $\alpha_i = 1, \forall i = 1, \ldots, S$, the weights of the word vectors.

1. Fill the blanks of the "cosine_similarity" function. Report the closest sentence to the sentence with idx "777", and their similarity score.
2. Fill the blanks of the "most_5_similar" function. Report the 5 closest sentences to the sentence with idx "777", and the associated similarity scores.

QUESTION 4 (OPTIONAL) In the **"fourth part"**, we are going to define the weights α_i as the IDF scores of each word. The TF-IDF is a classical method to construct a representation of a document. TF-IDF is composed of TF (Term Frequency) and IDF (Inverse Document Frequency). In our case, documents are just sentences, and the term frequency of each word inside a sentence is not very informative (they usually all appear once or twice max). But the IDF score is informative, and we are going to use them to embed a sentence using a weighted average of word2vec embeddings.

$$\boxed{\text{IDF\_avgword2vec(s)} = \frac{\sum_{i=1}^{S} \text{idf}(w_i).\text{word2vec}(w_i)}{\sum_{i=1}^{S} \text{idf}(w_i)}} \tag{3.2}$$

1. Implement the "IDF" function. Report the IDF score of the word "the", the word "a", and the word "clinton".
2. Implement the "avg_word2vec_idf" (very small modification from "avg_word2vec" function) using the "word2idf" dictionary. Report the closest sentence to sentence with idx 777.

4 SIMPLE LSTM FOR SEQUENCE CLASSIFICATION

You are provided with the following script : **lstm_imdb.py**. The script contains the following classical steps for training a simple LSTM for sequence classification :

- **Load the data** (IMDB).
- **Build the model** (assemble the modules - embedding, lstm, classifier - in a container).
- Define the **loss function (cross entropy), the metrics (accuracy), and the optimizer** (sgd or adam).
- **Train the model** on train, evaluate generalization error on a held-out valid set.
- **Evaluate** your model **on test set**.

The dataset we use is the **IMDB dataset for sequence classification**. It's a binary classification (good/bad) from movie reviews written on the imdb website. Take a look at the doc here : imdb dataset.
Keras has a very simple function for training a model : the "fit" function. that will train the model on a certain number of epoches, using a certain optimizer (sgd, adam..).

QUESTION 1 Read the **lstm_imdb.py** code and the associated comments. What is the (minibatch-)shape of :

1. the input of the embedding layer.
2. the input of the LSTM layer.
3. the output of the LSTM layer.

QUESTION 2

1. From the output of the **lstm_imdb.py** script, report the number of parameters of the model with the standard set of hyper-parameters. Report also the number of sentences in the training set. In standard statistics, a rule of thumb is to have less parameters than samples in your dataset. How do you think it's possible to train a model that has so many parameters compared to this number of samples? (we expect one key word)

QUESTION 3 The word embeddings are fed to the LSTM, which outputs a sentence embedding. This sentence embedding is then fed to a classifier, that outputs the prediction of the label.

1. For a single sentence, the LSTM has states $h_1, \ldots, h_T$ where T is the number of words in the sentence. The sentence embeddings that is fed to the classifier is thus computed as $f(h_1, \ldots, h_T)$. What is the exact form of $f(h_1, \ldots, h_T)$ used in the python script (look at the keras doc of the lstm function)?

QUESTION 4 Run the model with and without dropout (training should take less than 10 minutes).

1. For each, plot the evolution of the train and valid accuracy per epoch, and write the test errors that you obtain. You can use the python package "matplotlib" that's already installed (import matplotlib). Matplotlib tutorial.

QUESTION 5

1. Explain what is the difference between SGD and Adam. (no more than 3-4 lines) (more info here : Optimization methods).

5 SIMPLE CONVNET FOR SEQUENCE CLASSIFICATION

You are provided with the script **cnn_imdb.py** that is very similar to **lstm_imdb.py** except the model used is a 1D Convolution (or Temporal Convolution).
For Computer Vision, 2D Convolution is used directly on top of the image (3D : (RGB, width, height). For text processing, Convolution1D is done over the matrix of the embeddings of the words contained in the sentence (2D : (number of words, size of embedding)). To understand the difference, take a look at the the documentation : here
The model implemented in the script is composed of several modules :

1. It takes a sequence of words as input and transforms it into a sequence of word embeddings of dimension E using a lookup table (the "embedding" module).

2. The lookup table provides a matrix of word embeddings of size $S \times E$ which is the input to the 1D Convolution. The 1D Convolution applies N kernels to it (here N=256, with kernel size K=3). This outputs several "feature maps" ($\text{FM}_t \in \mathbb{R}^N$, $t = 1, \ldots, S$).

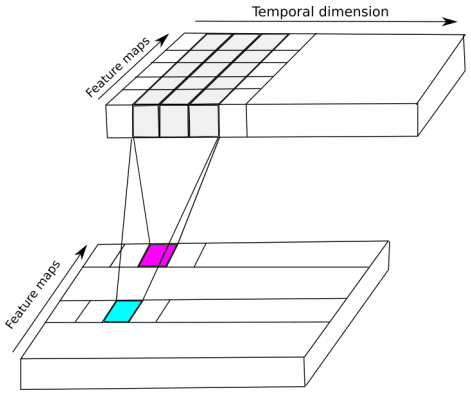

Figure 5.1: Temporal convolution : input is on top, output is on the bottom. Each filter is of the size of the grey part (in terms of number of parameters). One filter will output a (1, height, width) output : for instance, a filter will be applied to the grey zone and output the cyan output (a scalar).

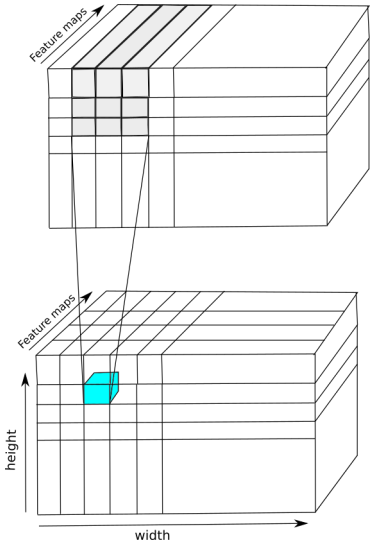

Figure 5.2: Spatial Convolution : input is on top, output is on the bottom. Each filter is of the size of the grey part (in terms of number of parameters). One filter will output a (1, height, width) output : for instance, a filter will be applied to the grey zone and output the cyan output (a scalar).

QUESTION 1 Fill the gaps of **cnn_imdb.py**. Run the script, and report the results (test loss and test error) that you obtain.

QUESTION 2

1. In **cnn_imdb.py**: what is the input and output shape of Convolution1D?

QUESTION 3 (+1.5 points)

1. In a new **cnn_lstm_imdb.py** script that you create (and include in your assignment), build a model where on top of the convolution, you have an LSTM. It means that the input of the LSTM will be the output of your ConvNet (yes, it's possible, and usually done in the litterature). Run the model with the best parameters you find (sufficiently small so that you have time to run the model, limiting over/under fitting). Report your best results.

6 POINTS TO FURTHER STUDY

Using Deep Learning models on rather small datasets is often not the best practice. In general, if you have a small dataset, take a look at simpler methods that use unigrams or bigrams, such as the "TF-IDF" method, with PCA ("Latent Semantic Analysis"), or more recently "FastText" from Facebook. This provides very fast and strong baselines that are usually hard to beat. Deep Learning methods such as LSTMs can be applied for a various number of applications, not only text classification and are thus efficient tool for Natural Language Processing. They are state-of-the art for many different text processing tasks as we have seen in the Deep Learning for NLP course.