Practical work 12 – Long Short-Term Memory Networks

Objectives

The objective of this PW is to practice applications of more complex RNN cells including LSTMs and GRU, also in combination with word embeddings.

Submission

— **Deadline** : W13 (in 1 week), before the start of the lecture.

— **Format** : Zip with the jupyter notebooks.

Exercise 1 Word polarity detection with word embedding

The objective is to build a system that takes as input a word and outputs a probability that the word has a positive or negative connotation (*polarity*). As illustrated in Figure 1, the system is composed of two parts in which the word embedding part will leverage on pre-trained vectors such as the one obtained with word2vec or GloVe.

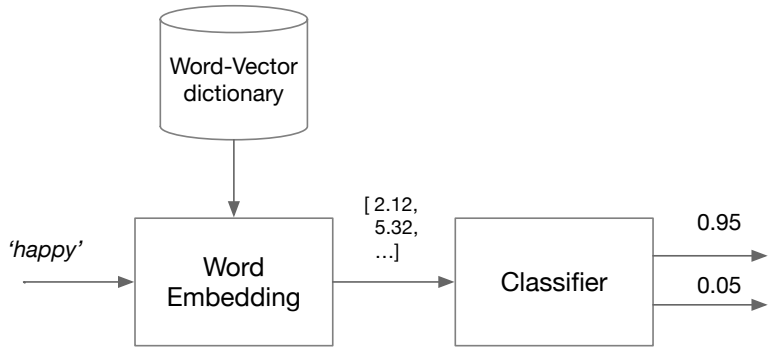

FIGURE 1 – Word polarity detection system composed of two parts : (1) the word embedding part using pre-trained dictionary mapping word to vector and (2) the classifier part.

a) **Read lexicons**. Download lists of positive and negative words from the *Opinion Lexicon* available from https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html. A zip is also provided on Moodle. Listing 1 provides an example of function to read the lexicons. You may want to complete the code to remove lines that start with ;, that end with + and to remove empty lines. You should get 2005 positive words and 4783 negative words.

```
def read_vocabulary_from_file(filename):
    with open(filename, 'r', encoding="ISO-8859-1") as f:
        content = f.readlines() # content is a list of lines
        content = [x.strip() for x in content]  # removing newline chars
        ...
        return content
```

Listing 1 – Reading lexicon

b) **Convert words into vectors**. You can here go for two options : either by querying an online API that returns you the vectors for a given word, or download a pre-trained word-vector dictionary (word2vec, GloVe, etc.). The code provided in Listing 2 shows you how to realise this using a GloVe embedding – zip also provided on Moodle.

```
# to get GloVe vectors: wget http://nlp.stanford.edu/data/glove.6B.zip
def load_glove_embeddings(path):
    embeddings = {}
    with open(path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.strip().split()
            w = values[0]
            vectors = np.asarray(values[1:], dtype='float32')
            embeddings[w] = vectors
    return embeddings

# online query
import requests
import json
word = 'happy'
response = requests.get('https://icoservices.k8s.tic.heia-fr.ch/word-
    embedding/wordvector/word2vec/en/' + word)
vector = response.json()

# off-line dictionary
word_dict = load_glove_embeddings('glove.6B/glove.6B.50d.txt')
word = 'happy'
vector = word_dict[word]   # if word is in word_dict
```

Listing 2 – Converting word to vectors

c) **Prepare the training and testing sets**. Prepare the tensors `X_train` for training by taking the corresponding vectors of 1500 positive and 1500 negative words from the lexicon. Prepare the `Y_train` target output tensor corresponding to the training set. You can, for example use the target [1.0, 0.0] for a positive word and [0.0, 1.0] for a negative word. For an embedding dimension of 50, the shapes of your `X_train` and `Y_train` tensors should be (3000,50) and (3000,2). In a similar way, prepare `X_test` and `Y_test` tensors.

d) **Train and evaluate a classifier**. Build a model, e.g. a double Dense layers in Keras (MLP) and train it. Report on the evolution of the loss and accuracy along the epochs. You should reach about 90% accuracy on the training set and 85% accuracy on the test set. Report on your model structure and fitting strategy.

e) **Analysis of results and discussions**. Report on your experiments and comment your classification performances. Find 5 to 10 words which are not in your training set and on which the system thinks it is clearly positive or negative. Find some words on which the system hesitates, i.e. with output probabilities in range [0.4-0.6]). Are these words and outputs making sense to you ?

f) **Optional – Extension to sentences**. Could you use the above system to make a polarity detection for the comments given by clients on products, e.g. on www.digitech.ch ? What would be the limitations of such system and how could we improve it ? What is happening when a word is not in the dictionary of the embedding ?

Exercise 2 IMDb Sentiment Classification

In this exercise, you explore the hyperparameter/model space - here to identify the best model for classifying the sentiment in reviews from the IMDb database.

a) Download the zip-file `imdb_sentiment.zip` with the jupyter notebook and the data folder `imdb_data`, unzip the file, open the jupyter notebook.

b) The first cells contain some helper functions to load the training and test data. In this example, the preprocessing pipeline is very important and you will need to explore different settings in combination with the classification models. Study the different steps in detail and identify and describe the most important parameters that you can tune afterwards.

c) Now implement various different models (all are many-to-one) with different hyperparameter settings :

   — Model types : $SimpleRNN$, $LSTM$, GRU, $Conv1D$ + $Dense$ (with softmax) + $Embedding$ (with/without finetuning)

   — Number of layers : 1+

   — Different hyper parameters : **Preprocessing parameters**.

   — With/without regularization (e.g. dropout, recurrent dropout)

   Analyse at least 10 different settings (combinations of model/hyperparameters) - each of the RNN model types should occur at least once. Describe quantitatively and qualitatively

the resulting performances, the shape of the learning curves and the confusion statistics. Spot potential issues for specific settings / models. Report about your findings in the notebook.

d) Identify the best model suited for the given task. Report the results (test accuracy) for three independent runs.

Can you beat the accuracy reported in class (shown on the slides) ?

Exercise 3 Optional : Review Questions

a) Give two desired properties of word embeddings when used in a classification task (such as sentiment classification).

b) What are the three strategies to use an embedding layer in a deep system ?

c) Why is the system of exercise 1 working in the end ?

d) What are the different forms of sequence *mapping* allowed by recurrent neural networks ? Give for each form an example of application.

e) Compute the number of parameters to be trained for a two-layer $SimpleRNN$ and $softmax$ with hidden state dimensions 32 and 64, respectively, 10 classes to classify in the softmax and inputs given by sequences of length 100 and each element a vector of dimension 30.

f) Compute the number of parameters to be trained for a two-layer $LSTM$ and $softmax$ with hidden state dimensions 32 and 64, respectively, 10 classes to classify in the softmax and inputs given by sequences of length 100 and each element a vector of dimension 30.

g) Why is gradient clipping rather needed in long than in short sentences ?

h) In what situations would you expect LSTMs (by its design) to perform better than SimpleRNNs ?

i) Describe why SimpleRNNs have problems in learning long-term dependencies.