Applied Machine Learning - EDAN95
Assignment 4

Recognizing Named Entities in Text
Managing a Text Dataset
Reccurent Neural Networks
Embeddings

Hicham Mohamad
hsmo@kth.se

November 30, 2018

1 Introduction

The objectives of this assignment are to:

- Write a program to recognize named entities in text
- Learn how to manage an text data set
- Apply **Recurrent Neural Networks** to text
- Know what embeddings are

Programming

- Each group will have to write Python programs to recognize named entities in text.
- You will have to experiment different architectures, namely RNN and LSTM, and compare the results you obtained

2 Preliminaries: Recurrent Neural Networks (RNN)

As mentioned in chapter 10 in [2], **Recurrent neural networks**, or RNNs (Rumelhart et al., 1986a), are a family of neural networks for processing *sequential data*. Much as a **convolutional** network is a neural network that is specialized for processing a grid of values X such as an **image**, a **recurrent** neural network is a neural network that is specialized for processing a **sequence of values** $x^{(1)}, \cdots, x^{(\tau)}$.

The factors determining how well a machine learning algorithm will perform are its ability to

1. Make the training error small.
2. Make the gap between training and test error small.

These two factors correspond to the two central challenges in machine learning: underfitting and overfitting. *Underfitting* occurs when the model is not able to obtain a sufficiently low error value on the training set. *Overfitting* occurs when the gap between the training error and test error is too large.

We can control whether a model is more likely to overfit or underfit by altering its **capacity**. Informally, a model's capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set. Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

One way to control the capacity of a learning algorithm is by choosing its **hypothesis space**, the set of functions that the learning algorithm is allowed to select as being the solution.

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. Regularization is one of the central concerns of the field of machine learning, rivaled in its importance only by optimization.

The **no free lunch theorem** has made it clear that there is no best machine learning algorithm, and, in particular, no best form of regularization. Instead we must choose a form of regularization that is well suited to the particular task we want to solve. The philosophy of deep learning in general is that a wide range of tasks (such as all the intellectual tasks that people can do) may all be solved effectively using very general-purpose forms of regularization.

3 Collecting a Dataset

1. You will use a dataset from the **CoNLL conferences** that benchmark natural language processing systems and tasks. There were two conferences on named entity recognition: CoNLL 2002 (Spanish and Dutch)

   `https://www.clips.uantwerpen.be/conll2002/ner/`

   and CoNLL 2003 (English and German)

   `https://www.clips.uantwerpen.be/conll2003/ner/`.

   In this assignment, you will work on the English dataset. Read the description of the task.

2. The datasets are protected by a license and you need to obtain it to reconstruct the data. Alternatively, you can use a local copy or try to find one on github (type conll2003 in the search box) or use the Google dataset search: `https://toolbox.google.com/datasetsearch`. You can find a local copy in the `/usr/local/cs/EDAN95/datasets/NER-data` folder.

3. The dataset comes in the form of three files: a training set, a development set, and a test set. You will use the test set to evaluate your models. For this, you will apply the conlleval script that will compute the **harmonic mean of the precision and recall**: F1. You have a local copy of this script in `/usr/local/cs/EDAN95/datasets/ner/bin`. `conlleval` is written in Perl. Be sure to have it on your machine to run it.

4 Collecting the Embeddings

1. Download the **GloVe embeddings 6B** from `https://nlp.stanford.edu/projects/glove/` and keep the 100d vectors.

2. You have a local copy of this script in `/usr/local/cs/EDAN95/datasets/`;

3. Write a function that reads GloVe embeddings and store them in a dictionary, where the keys will be the words and the values, the embeddings.

4. Using a **cosine similarity**, compute the 5 closest words to the words table, france, and sweden.

5 Reading the Corpus and Building Indices

You will read the corpus with programs available from `https://github.com/pnugues/edan95`. These programs will enable you to load the files in the form of a **list of dictionaries**.

1. Write a function that for each sentence returns the **x** and **y** lists of symbols consisting of words and **NER tags**. For the second sentence of the training set, you should have:

x = ['eu', 'rejects', 'german', 'call', 'to', 'boycott', 'british', 'lamb', '.']

y = ['I-ORG', 'O', 'I-MISC', 'O', 'O', 'O', 'I-MISC', 'O', 'O']

Some datasets you may find on the web use a different NER tagset, where I- is replaced with B-, like B-ORG instead of I-ORG. This will not change the results.

2. Apply this function to your datasets so that you create **X** and **Y** lists of lists consisting of words and NER tags.

3. Create a **vocabulary** of all the words observed in the training set and the words in GloVe. You should find 402,595 different words.

4. Create **indices** and inverted indices for the words and the NER: i.e. you will associate each word with a number. You will use index 0 for the padding symbol and 1 for unknown words.

6 Building the Embedding Matrix

1. Create a matrix of dimensions (M, N), where M, will the size of the vocabulary: The unique words in the training set and the words in **GloVe**, and N, the dimension of the embeddings.
The padding symbol and the unknown word symbol will be part of the vocabulary.
The shape of your matrix should be: (402597, 100). Initialize it with random values.

2. Fill the matrix with the GloVe embeddings when available. You will use the indices from the previous section.

7 Creating the X and Y Sequences

You will now create the input and output sequences with numerical indices

1. Convert the **X** and **Y** lists of symbols in lists of numbers using the indices you created.

2. Pad the sentences using the `pad_sequences` function. As maximum length and `maxlen` argument, you will use 150 or greater. What matters is that you have a length that is larger than the maximum length observed in your training and development corpora. After padding, the second sentence you look like (the indices are not necessarily the same).

```
x = [     0      0      0      0      0      0      0      0      0
          0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0      0      0      0      0      0      0      0      0      0
         0 142143 307143 161836  91321 363368  83766  85852 218260    936]
y = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
     0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
       0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 9 6 9 9
       9 6
 9 9]
```

3. Do the same for the development set.

8 Building a Simple Recurrent Neural Network

1. Create a simple recurrent network and train a model with the training set. As layers, you will use `Embedding`, `SimpleRNN`, and `Dense`.
2. Compile and fit your network. You will report the training and validation losses and accuracies and comment on the possible **overfit**.
3. Apply your network to the test set and report the accuracy you obtained. You will use the `evaluate` method.

9 Evaluating your System

You will use the official script to evaluate the performance of your system

1. Use the `predict` method to predict the tags of the whole test set
2. Write your results in a file, where the two last columns will be the hand-annotated tag and the predicted tag. The fields must be separated by a space and each line must end with a new line: `\n`.
3. If you save your results on a Windows machine, Python will use the default end-of-line sequence: `\r\n`. You will then need either to convert your file or to modify the way you save your file.
4. Apply `conlleval` to your output. Report the F1 result. Be aware that `conlleval` was designed for Unix and will break with Windows end-of-line conventions.
5. Try to improve your model by modifying some parameters, adding layers, adding `Bidirectional` and `Dropout`.
6. Evaluate your network again

10 Building a LSTM Network

1. Create a simple LSTM network and train a model with the train set. As layers, you will use `Embedding`, `LSTM`, and `Dense`.
2. Apply conlleval to your output. Report the F1 result.
3. Try to improve your model by modifying some parameters, adding layers, adding `Bidirectional`, `Dropout`, possibly mixing `SimpleRNN`.
4. Apply your network to the test set and report the accuracy you obtained. you need to reach a F1 of 82 to pass.

11 Report (Not Applicable in 2018)

You will write a short report of about two pages on your experiments:

1. You will describe the architectures you designed and the results you obtained;
2. You will read the article, *Contextual String Embeddings for Sequence Labeling* by Akbik et al. (2018)
   https://www.aclweb.org/anthology/C18-1139/
   and you will outline the main differences between their system and yours. You will tell the performance they reached on the corpus you used in this laboratory. This part should be of about one page.

To write the report, you will use Overleaf.com. The submission procedure is described here: `http://cs.lth.se/edan95/lab-programming-assignments/`, but send me the Overleaf link only.

You must submit this report no later than one week after you have complete the lab.

12 Appendix

References

[1] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*, The MIT Press, Cambridge, Massachusetts.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning.* MIT Press, 2016, ISBN: 9780262035613.

[3] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[4] Nugues, EDAN95 github: `https://github.com/pnugues/edan95`.

[5] Akbik et al., Contextual String Embeddings for Sequence Labeling, 2018.

[6] Toft, Python ID3 Trees, github: `https://github.com/tofti/python-id3-trees`.

[7] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`

[8] Aurélien Géron: "Hands-On Machine Learning with Scikit-Learn and TensorFlow, Concepts, Tools, and Techniques to Build Intelligent Systems". O'Reilly Media, 2017, ISBN: 9781491962299.

[9] François Chollet: "Deep Learning with Python". Manning, 2018, ISBN: 9781617294433.

[10] Tom Mitchell: "Machine Learning". McGraw Hill, 1997, ISBN: 0070428077.

[11] David L. Poole, Alan K. Mackworth: "Artificial Intelligence - Foundations of Computational Agents (2e)". Cambridge University Press, 2017, ISBN:9781107195394.

[12] Stuart Russel, Peter Norvig: Artificial Intelligence - A Modern Approach (3e). Pearson, 2010, ISBN:10: 0132071487

[13] Richard Johansson, "Scientific Computing with Python". `https://github.com/jrjohansson/scientific-python-lectures`

[14] Yann LeCun, Marc'Aurelio Ranzato. Deep Learning Tutorial, ICML, Atlanta, 2013-06-16 [Online]. `http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf`