Homework 2

Tommaso Pasini & Valentina Pyatkin

(pasini | pyatkin)@di.uniroma1.it

Supervised POS Tagger with LSTM

The Tasks

1. **Implement a sequence POS tagger with a LSTM.**

2. **Extends your model to cope with other languages. (Extra Points)**

What we provide:

- We will provide you a folder that has the following structure:
  - homework2/
    - slides.pdf
    - src/
      - homework2.py
    - data/
      - en-ud-train.conllu
      - en-ud-test.conllu
      - en-ud-dev.conllu

data/ - Universal Dependencies TreeBank

- Universal Dependencies (http://universaldependencies.org/) is a project that is developing cross-linguistically consistent treebank annotation for many languages.
- The tag set contains 17 POS.
- All the languages have the same tas.
- This enable multi lingual POS-tagging.
- CoNLL format is used. (http://universaldependencies.org/format.html)

Input Data - The CoNLL Format

- CoNLL format has one token per line.

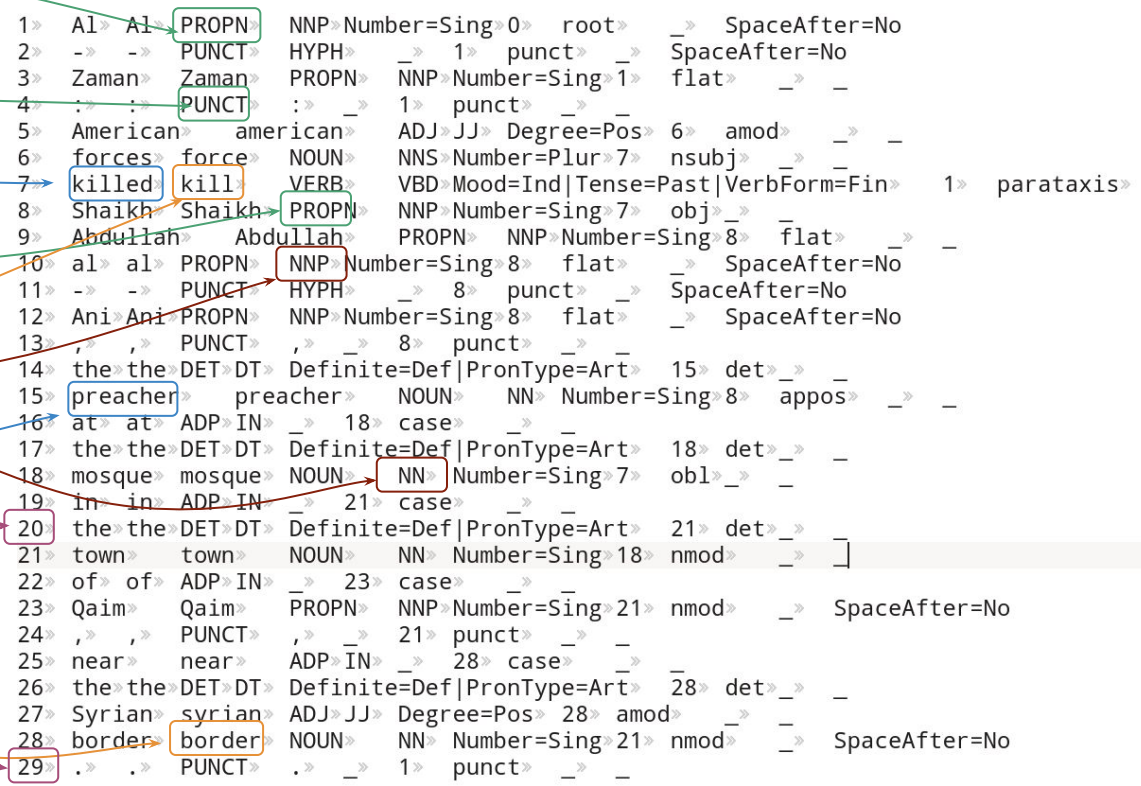
- Each line is divided in 10 column tab separated:
  - 1 Word ID: starts from 1 for each sentence and may be a range for multiwords
  - 2 FORM: the word form or the punctuation symbol.
  - 3 LEMMA: the lemma of the word form.
  - 4 U-POSTAG: universal part of speech.
  - 5 X-POSTAG: language specific pos tag (_ if not defined).
  - 6 FEATS: list of morphological features from universal feature inventory.
  - 7 HEAD: Word ID of the head of the word.
  - 8 DEPREL: universal dependency relation.
  - 9 DEPS: enhanced dependency graph.
  - 10 MISC: any other annotation.

Input Data - The CoNLL Format

- CoNLL format has one token per line.
- Each line is divided in 10 column tab separated:
  - 1 Word ID: starts from 1 for each sentence and may be a range for multiwords
  - 2 FORM: the word form or the punctuation symbol.
  - 3 LEMMA: the lemma of the word form.
  - 4 U-POSTAG: universal part of speech.
  - 5 X-POSTAG: language specific pos tag (_ if not defined).
  - 6 FEATS: list of morphological features from universal feature inventory.
  - 7 HEAD: Word ID of the head of the word.
  - 8 DEPREL: universal dependency relation.
  - 9 DEPS: enhanced dependency graph.
  - 10 MISC: any other annotation.

Input Data - The CoNLL Format

- UPOS
- XPOS
- FORM
- LEMMA
- WORD ID



```
1   Al      Al      PROPN   NNP     Number=Sing     0       root    _       SpaceAfter=No
2   -       -       PUNCT   HYPH    _       1       punct   _       SpaceAfter=No
3   Zaman   Zaman   PROPN   NNP     Number=Sing     1       flat    _       _
4   :       :       PUNCT   :       _       1       punct   _       _
5   American        american        ADJ     JJ      Degree=Pos      6       amod    _       _
6   forces  force   NOUN    NNS     Number=Plur     7       nsubj   _       _
7   killed  kill    VERB    VBD     Mood=Ind|Tense=Past|VerbForm=Fin        1       parataxis
8   Shaikh  Shaikh  PROPN   NNP     Number=Sing     7       obj     _       _
9   Abdullah        Abdullah        PROPN   NNP     Number=Sing     8       flat    _       _
10  al      al      PROPN   NNP     Number=Sing     8       flat    _       SpaceAfter=No
11  -       -       PUNCT   HYPH    _       8       punct   _       SpaceAfter=No
12  Ani     Ani     PROPN   NNP     Number=Sing     8       flat    _       SpaceAfter=No
13  ,       ,       PUNCT   ,       _       8       punct   _       _
14  the     the     DET     DT      Definite=Def|PronType=Art       15      det     _       _
15  preacher        preacher        NOUN    NN      Number=Sing     8       appos   _       _
16  at      at      ADP     IN      _       18      case    _       _
17  the     the     DET     DT      Definite=Def|PronType=Art       18      det     _       _
18  mosque  mosque  NOUN    NN      Number=Sing     7       obl     _       _
19  in      in      ADP     IN      _       21      case    _       _
20  the     the     DET     DT      Definite=Def|PronType=Art       21      det     _       _
21  town    town    NOUN    NN      Number=Sing     18      nmod    _       _
22  of      of      ADP     IN      _       23      case    _       _
23  Qaim    Qaim    PROPN   NNP     Number=Sing     21      nmod    _       SpaceAfter=No
24  ,       ,       PUNCT   ,       _       21      punct   _       _
25  near    near    ADP     IN      _       28      case    _       _
26  the     the     DET     DT      Definite=Def|PronType=Art       28      det     _       _
27  Syrian  syrian  ADJ     JJ      Degree=Pos      28      amod    _       _
28  border  border  NOUN    NN      Number=Sing     21      nmod    _       SpaceAfter=No
29  .       .       PUNCT   .       _       1       punct   _       _
```

Input Data - Tag Set

- Universal dependencies define 17 different POS tags: (http://universaldependencies.org/u/pos/index.html)
  - ADJ: adjective
  - ADP: adposition
  - ADV: adverb
  - AUX: auxiliary
  - CONJ: coordinating conjunction
  - DET: determiner
  - INTJ: interjection
  - NOUN: noun
  - NUM: numeral
  - PART: particle
  - PRON: pronoun
  - PROPN: proper noun
  - PUNCT: punctuation
  - SCONJ: subordinating conjunction
  - SYM: symbol
  - VERB: verb
  - X: other

Input Data - Split

- We will provide you the data.

- Data will be splitted in:
  - Training data: 229.672 tokens.
  - Development data: 29.152 tokens.
  - Test data: 29.250 tokens.

- Do not merge them!

homework2.py

- homework2.py define 3 abstract classes:
  - AbstractPOSTaggerTrainer
  - AbstractPOSTaggerTester
  - AbstractLSTMPOSTagger

- Homework2.py also define a Test class that implements a test method.
  - Run the homework2.py in order to test your implementation:
    python homework2.py [--no-train] <homework_dir> <model_path>

- **You need to pass the test in order to be evaluated on this homework**.

- homework2.py also contains an utility class (ModelIO) which implements methods to save and load a model.

What and how to code:

- You have to **implement** the 3 python **abstract classes** provided in homework2.py:
  - POSTaggerTrainer
  - POSTaggerTester
  - LSTMPOSTagger

- Each method in each class must be implemented.

- The implementation must be tested with the test() method in the test class Test

AbstractPOSTaggerTrainer:

- AbstractPOSTaggerTrainer represents a class to train a model.
- Has 1 abstract method train(trainnig_path) which takes as input the path to the training data and outputs a Sequential model.

```
class AbstractPOSTaggerTrainer:
    __metaclass__ = ABCMeta

    @abstractmethod
    def train(self, training_path):
        """
        Train the keras model from the training data.

        :param training_path: the path to training file
        :return: the keras model of type Sequential
        """
        pass
```

AbstractPOSTaggerTester:

- AbstractPOSTaggerTester represents a class to **test a trained model**.

- Has 1 abstract method test(model, test_file_path) which takes as input a trained sequential model and the path to the gold standard data. It **outputs a dictionary** containing **precision**, **recall**, **coverage** and **f1**.

```
class AbstractPOSTaggerTester:
    __metaclass__ = ABCMeta

    @abstractmethod
    def test(self, model, test_file_path):
        """
        Test the input model against the gold standard.

        :param model: a Sequential model that has to be tested.
        :param test_file_path: a path to the gold standard file.

        :return: a dictionary that has as keys 'precision', 'recall',
        'coverage' and 'f1' and as associated value their respective values.

        Additional info:
        - Precision has to be computed as the number of correctly predicted
          pos tag over the number of predicted pos tags.
        - Recall has to be computed as the number of correctly predicted
          pos tag over the number of items in the gold standard
        - Coverage has to be computed as the number of predicted pos tag over
          the number of items in the gold standard
        - F1 has to be computed as the armonic mean between precision
          and recall (2* P * R / (P + R))
        """
        pass
```

AbstractLSTMPOSTagger:

- AbstractLSTMPOSTagger is an abstract pos tagger that can predict the pos tags given a tokenized sentence.
- Has 1 abstract method predict(sentence) which given a list of tokens outputs a list of same length with the pos tags for each word.

```
class AbstractLSTMPOSTagger:
    __metaclass__ = ABCMeta

    def __init__(self, model):
        self._model = model

    def get_model(self):
        return self._model

    @abstractmethod
    def predict(self, sentence):
        """
        predict the pos tags for each token in the sentence.
        :param sentence: a list of tokens.
        :return: a list of pos tags (one for each input token).
        """
        pass
```

ModelIO:

- ModelIO it’s an utility class and implements two methods:
  - save(model, path) that save the model on the specified path.
  - load(path) that load a keras model from the path

- You should use this class in order to ensure that you save and load your model correctly.

- This class will be also used to test that your implementations return the correct objects.

```
class ModelIO:
    @staticmethod
    def save(model, output_path):
        '''
        Save the model to in the file pointed by the output_path variable

        :param model: the trained Sequential model
        :param output_path: the path to the file on which the model have to
                            be saved
        :return: no return value is required
        '''
        model.save(output_path)

    @staticmethod
    def load(model_file_path):
        '''
        Load a sequential model saved in the file pointed by model_file_path

        :parah model_file_path: the path to the file that has to be loaded
        :return: a sequential model loaded from the file
        '''
        import keras
        return keras.models.load_model(model_file_path)
```

The Test class:

- Test class implements a simple type checking test method to ensure that the classes you have implemented return the correct objects.
- You have to run successfully this test in order to ensure that you correctly implemented the homework.
- If the test fail, your homework will not be evaluated.

```
class Test:
    def __init__(self, training_path, model_path, gold_stanrdar_path):
        self._training_path = training_path
        self._model_path = model_path
        self._gold_standard_path = gold_stanrdar_path

    def test(self, lstm_trainer_implementation, lstm_tester_implementation, no_trai

        if no_train:
            model = ModelIO.load(self._model_path)
            print 'TEST 0\t\tNO-TRAIN'
        else:
            model = lstm_trainer_implementation.train(self._training_path)
            assert type(model) == Sequential
            print 'TEST 0\t\tPASSED'
            ModelIO.save(model, self._model_path)
            model = ModelIO.load(self._model_path)

        assert type(model) == Sequential

        print 'TEST 1\t\tPASSED'

        results = lstm_tester_implementation.test(model, self._gold_standard_path)
        assert type(results) == dict
        assert 'precision' in results.keys()
        assert 'recall' in results.keys()
        assert 'coverage' in results.keys()
        assert 'f1' in results.keys()

        print 'TEST 2\t\tPASSED'

        postagger = LSTMPOSTagger(model)
        test_sentence = ['this', 'is', 'an', 'easy', 'test']
        prediction = postagger.predict(test_sentence)

        assert len(test_sentence) == len(prediction)

        print 'TEST 3\t\tPASSED'

        return results
```

How to run the test

- python homework2.py /path/to/model.keras /path/to/your/submission_folder/

- **Be careful!!!** If /path/to/model.keras exist and the *--no-train* option is not specified then it will be overridden.

- The /path/to/your/submission_folder should be structured as specified in the previous slide, especially, it should contains the src/ folder with your implementation and the data/ folder

```
if __name__ == '__main__':
    """
    Main to run the test of the homework 2.
    Use the parameter --no-train in order to skip the training of the model
    @@@@@@@@@@@@@@@@@@@@ WARNING!!! @@@@@@@@@@@@@@@@@@@@
    the program will not check if model_path already exist and will overwrite it if it
    is the case
    """
    if len(sys.argv) < 3:
        print 'usage: python', sys.argv[0], '[--no-train] model_path, homework_dir'
        sys.exit(-1)
    model_index = 1
    homework_dir_index = 2
    no_train = False
    if '--no-train' in sys.argv:
        model_index = 2
        homework_dir_index = 3
        no_train = True

    model_output_path = sys.argv[model_index]
    homework_dir = sys.argv[homework_dir_index]

    src_dir = homework_dir + 'src/'

    print ''
    print 'model output:', model_output_path
    print 'homeword dir:', homework_dir
    print 'src dir:', src_dir
    print ''

    # dynamic import of modules
    sys.path.append(src_dir)
    from POSTaggerTester import POSTaggerTester
    from LSTMPOSTagger import LSTMPOSTagger
    from POSTaggerTrainer import POSTaggerTrainer

    ## get files
    training_data = homework_dir + '/data/en-ud-train.conllu'
    test_data = homework_dir + '/data/en-ud-test.conllu'

    test = Test(training_data, model_output_path, test_data)
    trainer = POSTaggerTrainer()
    tester = POSTaggerTester()

    name = ''
    if homework_dir.endswith('/'):
        name = homework_dir[:-1]
    else:
        name = homework_dir

    name = name[name.rfind("/"):]

    try:
        results = test.test(trainer, tester, no_train=no_train)
        print results
        print name, "PASSED"
    except Exception as e:
        print name, "FAILED"
        raise traceback.print_exc(e)
```

What to submit

- You have to submit a folder structured as follow:
  - cognome_matricola_homework_2/
    - homework2.py
    - report.pdf
    - src/
      - POSTaggerTrainer.py
      - POSTaggerTester.py
      - LSTMPOSTagger.py
    - data/
      - en-ud-train.conllu
      - en-ud-test.conllu
      - en-ud-dev.conllu
    - output/
      - pos_tagged_sentences.txt
      - results.txt
      - model.keras

What to submit - src

- src/ has to contain your source code. Specifically it has to contain **one file for each implemented class** and the contained **class has to be named with the file name.**

- **POSTaggerTrainer** and **POSTaggerTester** must have a **default empty constructor.**

- **LSTMPostTagger** has to implement a constructor that takes as input a Sequential model.

- You can include additional files auxiliaries file.

What to submit - output

- **pos_tagged_sentences.txt:**
  Must contain the output of LSTMPOSTagger#predict called on all test instances (From the test set). Has to be formatted as follow:
  sentence_1
  pos-tagged sentence_1
  ...

- **results.txt**:
  Must contain a line for each measure (precision, recall, f1, coverage) with its score.

- **model.keras**: your trained model

The report

- In the root folder you have to include:
  - the **homework2.py** library that we provide.
  - the **report.pdf** in pdf format.
- The report must be of maximum 2 pages and has to follow the following format:

  Abstract: a brief task description.

  Model description: description of your NN model, what layers you used, how many layers, activation function ecc.

  Optimization: description of how and which parameters you tuned; how and which word-vector representations you used.

  Experiments: description of your test and discussion of the results. Create a confusion matrix (12 x 12) of the POStags in which each cell (i,j) contains the number of times the model predicted the i-th POStag but should had predicted the j-th POStag. (A heat map of the confusion matrix would be very appreciated). Discuss interesting findings from the confusion matrix, using examples.

Extra points

- Universal dependencies enables multilingual POS tagging.
- Trainig sets are available for other languages at https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1983
- We will provide word-embeddings for other languages.

Extra points - The Task

- Merge English and Italian universal training set.
- Download Italian pre-trained word embeddings: http://hlt.isti.cnr.it/wordembeddings/skipgram_wiki_window10_size300_neg-samples10.tar.gz
- Use the same code you wrote to train your model with the new training set in Italian and English.
- Merge dev and test set of the two languages.
- Tune and Test the multilingual model on the new datasets.

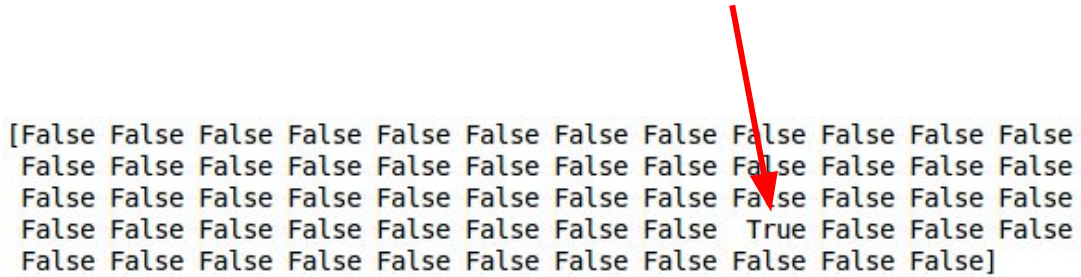
Extra points - What to submit

- Add multilingual/ folder to the root of your submission.
- Multilingual has to be structured as follow:
  - multilingual/
    - data/
      - ud-merged-train.conll
      - ud-merged-test.conll
      - ud-merged-dev.conll
    - output/
      - pos_tagged_sentences.txt
      - results.txt
      - multilingual_model.keras

How-To

How-To: transforming your labels into y vectors

- Reminder from practical session 4 and 5:
  - Do a label-to-index mapping
  - Do a index-to-label mapping
  - Represent your label as a one-hot-encoding using the label-to-index mapping
  - Retrieve the label from the y-vector given the index-to-label mapping
  - You can use zeros and ones, or booleans (like in the example)



How-To: Getting your input matrix

- Convert your words into appropriate vector representations that can be used as input to your NN (look at slides from practical sessions 4+5)
- Reminder from the last two practical sessions:
  - represent each word as a vector
  - add the word-vectors into another vector to represent the context of the temporal sequence you are looking at
  - add the temporal sequences to the final matrix that you will use as input for the network
  - ! your input is three-dimensional !

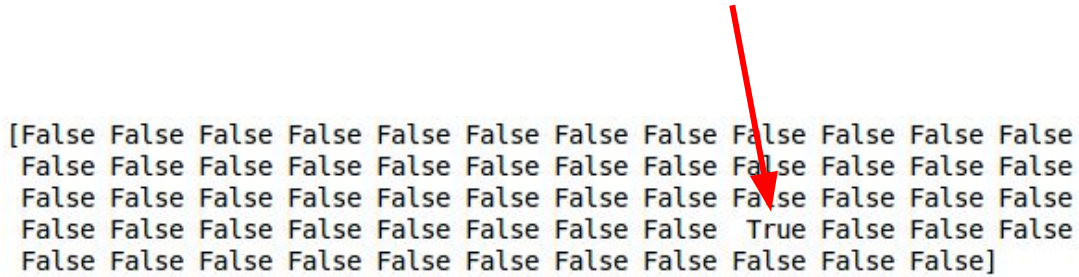
```
[[[False False False ..., False False False]
  [False False False ..., False False False]
  [False False False ..., False False False]
  ...,
  [False False False ..., False False False]
  [False False False ..., False False False]
  [False False False ..., False False False]]

 [[False False False ..., False False False]
  [False False False ..., False False False]
  [False False False ..., False False False]
  ...,
  [False False False ..., False False False]
  [False False False ..., False False False]
  [False False False ..., False False False]]

 [[False False False ..., False False False]
  [ True False False ..., False False False]
  [ True False False ..., False False False]
  ...,
  [False False False ..., False False False]
  [False False False ..., False False False]
  [False False False ..., False False False]]

 ...,
```

How-To: Word-Vector Representations

- There are many ways to represent your input words as vectors.
- It is up to you to choose an appropriate representation.
- The following slides will give you some ideas on what kind of vectors you can work with.

Word-Vector Representations: One-hot encodings

- Reminder: dimensionality of the length of the set of your predefined vocabulary
- Binary or boolean
- Cons: high dimensionality, sparse



Word-Vector Representations: GloVe

- Use pretrained GloVe vectors: https://nlp.stanford.edu/projects/glove/
- Distributional representation
- Word to word co-occurrence statistics in a corpus
- Cons: stopwords are not included... -> you have to solve this problem yourself:
  - a solution:
    - have a special "unknown-words"/UNK vector
- Pros: captures semantics of words
  - related words are closer in the (distributional) semantic space

Word-Vector Representations: Word2Vec

- Use pretrained Word2Vec vectors: https://github.com/mmihaltz/word2vec-GoogleNews-vectors
- Neural model: Skip-Gram or CBOW with either hierarchical softmax or negative sampling
- Cons: stopwords are not included... -> you have to solve this problem yourself:
  - a solution:
    - have a special "unknown-words"/UNK vector
- Pros: captures semantics of words
  - embeddings
  - reduced dimensionality

Word-Vector Representations: Build-your-own

- Ideas:
  - Build your own vectors by taking distributional counts, over a fixed number of content words and context, and weighting them with tfldf (term frequency - inverse document frequency)
  - You could also work with other dimensionality reduction methods like PCA and LSA

The Neural Network Layers

- **Input layer**
  - takes your word vector representations
    - Input dimension: your vector dimensions (depend on the representation you chose)
    - Output dimension: same (it's a linear layer)
- **Embedding layer**:
  - Input: one hot representation of the word.
  - Output: latent (embedded) vector with the chosen size.
- **LSTM layer**
  - Long-Short-Term-Memory Network
    - you can define your own hidden layer size
- **Output layer**
  - does a softmax over the hidden layer from the LSTM, in order to predict the most probable PoS
    - you get a matrix with *batchsize x y-vector-size* dimensions, e.g. every vector in your matrix is a probability distribution for the PoS tags for a single word.

Our Tips

- Start by transforming your input into vector representation(s)
- Start with a simple network and see how it performs. Then make it more complex.
- Keep in mind that training NNs might take some time (start soon!)

The Deadline

- Sunday, 7th of May
- http://robertonavigli.com/nlp2017/