

Master in Artificial Intelligence

Advanced Human Language Technologies

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner
- Classifier
- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Session 5 - NERC using neural networks

Assignment

Write a python program that parses the given XML and recognizes and classifies drug names. The program must use a neural network approach.

```
$ python3 ./nn-NER.py devel.xml result.out
DDI-DrugBank.d278.s0|0-9|Enoxaparin|drug
DDI-DrugBank.d278.s0|93-108|pharmacokinetics|group
DDI-DrugBank.d278.s0|113-124|eptifibatide|drug
DDI-MedLine.d88.s0|15-30|chlordiazepoxide|drug
DDI-MedLine.d88.s0|33-43|amphetamine|drug
DDI-MedLine.d88.s0|49-55|cocaine|drug
DDI-MedLine.d88.s1|82-95|benzodiazepine|drug
...
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner
- Classifier
- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

General Structure

The general structure is basically the same than for the traditional ML approach:

- B-I-O schema
- Two programs: one learner and one classifier.
- The learner loads the training (Train) and validation (Devel) data, formats/encodes it appropriately, and feeds it to the model, together with the ground truth.
- The classifier loads the test data, formats/encodes it in the same way that was used in training, and feeds it to the model to get a prediction.

In the case of NN, we don't need to extract features (though we **do need** proper input encoding)

Input Encoding

- The input/output layers of a NN are vectors of neurons, each set to 0/1.
- Modern deep learning libraries handle this in the form of *indexes* (i.e. just provided the *position* of active neurons, omitting zeros).
- For instance, in a LSTM, each input word in the sequence may be encoded as the concatenation of different vectors each containing information about some aspect of the word (form, lemma, PoS, suffix...)
- Each vector will have only one active neuron (*one-hot-encoding*), indicated by its *index*. This input is usually fed to an embedding layer.
- Our learner will need to create and store *index* dictionaries to be able to map the index number assigned to each word, label, or any other used piece of information. See class *Codemaps* below.

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner
- Classifier
- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner

- Classifier

- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure
Learner

Core task

Goals &
Deliverables

Learner - Main program

```
1 def train(trainfile, validationfile, params, modelname) :
2     '''
3     learns a NN model using trainfile as training data, and validationfile
4     as validation data. Saves learnt model in a file named modelname
5     '''
6     # load train and validation data in a suitable form
7     traindata = Dataset(trainindir)
8     valdata = Dataset(validationindir)
9
10    # create indexes from training data
11    codes = Codemaps(traindata, params)
12    # encode datasets
13    train_loader = encode_dataset(traindata, codes, params)
14    val_loader = encode_dataset(valdata, codes, params)
15
16    # build network
17    network = nercLSTM(codes)
18
19    # save indexes
20    os.makedirs(modelname, exist_ok=True)
21    codes.save(modelname+"/codemaps")
22
23    # train network, keep the best performing model
24    best = 0
25    for epoch in range(params['epochs']):
26        acc = train(network, epoch, train_loader)
27        if acc>best :
28            best = acc
29        torch.save(network, os.path.join(modelname, f"network.nn"))
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Learner

Core task

Goals &
Deliverables

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner

- **Classifier**

- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure
Classifier

Core task

Goals &
Deliverables

Classifier - Main program

```
1 def predict(modelname, datafile, params, outfile) :
2     '''
3     Loads a NN model from 'modelname' and uses it to extract drugs
4     in datafile. Saves results to 'outfile' in the appropriate format.
5     '''
6     # Load model
7     model = torch.load(os.path.join(modelname, "network.nn"),
8                        map_location=torch.device(used_device))
9
10    model.eval()
11    # load indexes
12    codes = Codemaps(os.path.join(modelname, "codemaps"), params)
13    # load data to annotate
14    testdata = Dataset(datafile)
15    test_loader = encode_dataset(testdata, codes, params)
16    # run each sentence through the NN, get results
17    Y = []
18    for X in test_loader:
19        y = model.forward(*X)
20        Y.extend([[codes.idx2label(torch.argmax(w)) for w in s] for s in y] )
21
22    # print results
23    output_entities(testdata, Y, codes, outfile)
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Classifier

Core task

Goals &
Deliverables

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner

- Classifier

- **Auxiliary classes**

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables

Auxiliary classes - Dataset

```
1 class Dataset:
2     ## constructor: parses datafile XML file, tokenizes each sentence, and
3     ## stores a list of sentences, as well as ground truth for each token
4     def __init__(self, datafile):
5
6     ## iterator to get all sentences in the data set.
7     ## for each sentence returns a triplet (text, tokens, labels)
8     def sentences(self):
9
10    ## iterator to get ids for sentence in the data set
11    def sentence_ids(self):
12
13    ## get tokens for one given its id
14    def get_sentence_tokens(self, sid) :
15    ## get text for one sentence given its id
16    def get_sentence_text(self, sid) :
17    ## get labels for one sentence given its id
18    def get_sentence_labels(self, sid) :
19    , , ,
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables

Auxiliary classes - Codemaps

```
1 class Codemaps :
2     # Constructor: create code mapper either from training data, or
3     #               loading codemaps from given file.
4     #               If 'data' is a Dataset, and lengths are not None,
5     #               create maps from given data.
6     #               If data is a string (file name), load maps from file.
7     def __init__(self, data, params)
8     # Save created codemaps in file named 'name'
9     def save(self, name)
10    # Convert a Dataset into lists of word codes and suffix codes
11    # Adds padding and unknown word codes.
12    def encode_words(self, data)
13    # Convert the gold labels in given Dataset into a list of label codes.
14    # Adds padding
15    def encode_labels(self, data)
16    # get word index size
17    def get_n_words(self)
18    # get suf index size
19    def get_n_sufs(self)
20    # get label index size
21    def get_n_labels(self)
22    # get index for given word
23    def word2idx(self, w)
24    # get index for given suffix
25    def suff2idx(self, s)
26    # get index for given label
27    def label2idx(self, l)
28    # get label name for given index
29    def idx2label(self, i)
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables

Required functions - network.py

```
1 class nercLSTM(nn.Module):
2     def __init__(self, codes):
3         super(nercLSTM, self).__init__()
4         # get sizes from index
5         n_words = codes.get_n_words()
6         n_sufs = codes.get_n_sufs()
7         n_labels = codes.get_n_labels()
8         # create embedding layers
9         embW_sz, embS_sz = 100, 50
10        self.embW = nn.Embedding(n_words, embW_sz)
11        self.embS = nn.Embedding(n_sufs, embS_sz)
12        self.dropW = nn.Dropout(0.1)
13        self.dropS = nn.Dropout(0.1)
14        # create LSTM layer + final linear classification layer
15        lstm_in_sz, lstm_out_sz = embW_sz+embS_sz, 200
16        self.lstm = nn.LSTM(lstm_in_sz, lstm_out_sz,
17                            bidirectional=True, batch_first=True)
18        self.out = nn.Linear(2*lstm_out_sz, n_labels)
19
20    def forward(self, w, s):
21        x = self.embW(w) # apply embedding layers to input
22        y = self.embS(s)
23        x = self.dropW(x) # apply dropout to embeddings output
24        y = self.dropS(y)
25        # concatenate embeddigns for word + suffix
26        x = torch.cat((x, y), dim=2)
27        x = self.lstm(x)[0] # feed concatenated vector to LSTM
28        x = self.out(x) # feed LSTM output to classification layer
29        return x
30
```

Neural
Networks
NERC

General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables

Network architecture

Neural
Networks
NERC

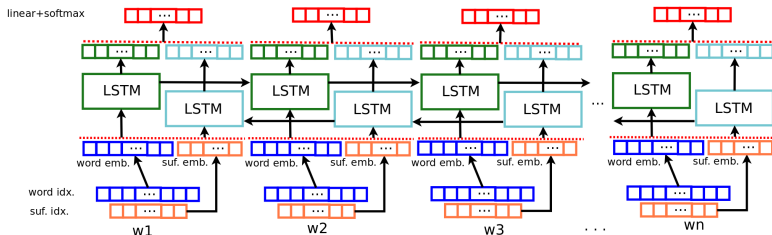
General
Structure

Detailed
Structure

Auxiliary classes

Core task

Goals &
Deliverables



Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner
- Classifier
- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Build a good NN-based drug NERC

Strategy: Experiment with different architectures and possibilities.
Some elements you can play with:

- Embedding dimension
- Initializing word embeddings with available pretrained models
- Max length and suffix length values
- Number of LSTM units
- Used optimizer, learning rate, batch size, ...
- Number and kind of layers or activation functions
- Additional input layers (maybe with embeddings). **Attention:**
This will require extending class `Codemaps` to handle the codes of added input layers.
 - lowercased words
 - different length suffixes and/or prefixes
 - PoS tags
 - feature layer (with information about capitalization, dashes, presence in external resources, etc)

Build a good NN-based drug NERC

Warnings:

- Neural Network training uses randomization, so different runs of the same program will produce different results. For repeatable results, use a random seed (and/or run the training several times).
- During training, *accuracy* on training and validation sets is reported. Those values are usually over 98%. However, this is due to the fact that most of the words have label “0” (non-drug). Accuracy values around 98% roughly correspond to F_1 values under 25%. To get a reasonable F_1 , validation set accuracy should reach about 99.5%.

To precisely evaluate how your model is doing, **do not rely** on reported accuracy: run the classifier on the development set and use the evaluator.

Outline

1 Neural Networks NERC

2 General Structure

3 Detailed Structure

- Learner
- Classifier
- Auxiliary classes

4 Core task

5 Goals & Deliverables

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables

Exercise Goals

What you should do:

- Work on your architecture and input vectors. It is the component of the process where you have most control.
- Experiment with different architectures and hyperparameters.
- Experiment with different input information
- Keep track of tried variants and parameter combinations.

What you should **NOT** do:

- Alter the suggested code structure (i.e. change only `network.py` and `Codemaps`).
- Produce an overfitted model: If performance on the test dataset is much lower than on devel dataset, you probably are overfitting your model.

Exercise Goals

Orientative results:

- A biLSTM with 2 input layers (word and suffix embeddings) is enough to get a macroaverage F1 about 50% on devel.
- Adding input layers with lowercased words and additional features (capitalization, dashes, numbers, presence in external files, ...), and some additional fully-connected layer at the end, raises the score over 65% on devel.

Results much lower than these orientative scores is an indication that you are doing something wrong or not elaborated enough.

Deliverables

- You'll be expected to produce a report on neural approaches to NER and DDI.
- By now, just keep track of the information you'll need later:
 - Experimented architectures/hyperparameters
 - Experimented input information
 - Performance results on devel corpus using different configurations
 - Performance results on test corpus using different configurations

Neural
Networks
NERC

General
Structure

Detailed
Structure

Core task

Goals &
Deliverables