Advanced Human Language Technologies

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Outline

Assignment

Write a python program that parses all XML files in the folder given as argument and recognizes and classifies drug names. The program must use a neural network approach.

```
$ python3 ./nn-NER.py data/Devel/
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
...
```

Outline

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 appropiately, and feeds it to the model, toghether 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, ommitting 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, 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 code assigned to each word, label, or any other used piece of information. See class *Codemaps* below.

Outline

Outline

Learner - Main program


```
def learn(traindir, validationdir, n_epochs, modelname) :
   '''
   learns a NN model using traindir as training data, and validationdir
   as validation data. Saves learnt model in a file named modelname
   '''
   # load train and validation data in a suitable form
   traindata = Dataset(traindir)
   valdata = Dataset(validationdir)
   # create indexes from training data
   max_len = 150
   suf_len = 5
   codes = Codemaps(traindata, max_len, suf_len)
   # encode datasets
   train_loader = encode_dataset(traindata, codes)
   val_loader = encode_dataset(valdata, codes)
   # build network
   network = nercLSTM(codes)
   optimizer = optim.Adam(network.parameters())
   # train network
   for epoch in range(n_epochs):
      train(epoch)
      test()
   # save model and indexs
   os.makedirs(modelname,exist_ok=True)
   torch.save(network, modelname+"/network.nn")
   codes.save(modelname+"/codemaps")
```

Outline

Classifier - Main program


```
def predict(modelname, datadir) :
   '''
   Loads a NN model from file 'modelname' and uses it to extract drugs
   in datadir. Saves results to 'outfile' in the appropriate format.
   '''

   # load model and associated encoding data
   model = torch.load(modelname+"/network.nn")
   model.eval()
   codes = Codemaps(modelname+'/codemaps.txt')

   # load and encode data to annotate
   testdata = Dataset(datadir)
   test_loader = encode_dataset(testdata, codes)

   Y = []
   for X in test_loader:
      y = model.forward(*X)
      Y.extend([[codes.idx2label(torch.argmax(w)) for w in s] for s in y] )

   # extract entities and dump them to output file
   output_entities(testdata, Y)
```

Outline

Auxiliary classes - Dataset


```
class Dataset:
   ## constructor: parses all XML files in datadir, tokenizes
   ## each sentence, and
   ## stores a list of sentences, each of them as a sequence of
   ## tokens (word, start, end, gold_label)
   def __init__(self, datadir)

   ## iterator to get all sentences in the data set
   def sentences(self)

   ## iterator to get ids for sentence in the data set
   def sentence_ids(self)

   ##  get one sentence (list of tokens) given its id
   def get_sentence(self, sid) :
  '''
```

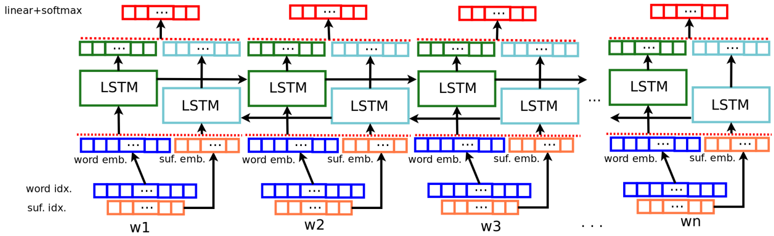
Auxiliary classes - Codemaps


```
class Codemaps :
    # Constructor: create code mapper either from training data, or
    #              loading codemaps from given file.
    #              If 'data' is a Dataset, and lengths are not None,
    #              create maps from given data.
    #              If data is a string (file name), load maps from file.
    def __init__(self, data, maxlen=None, suflen=None)
    # Save created codemaps in file named 'name'
    def save(self, name)
    # Convert a Dataset into lists of word codes and sufix codes
    # Adds padding and unknown word codes.
    def encode_words(self, data)
    # Convert the gold labels in given Dataset into a list of label codes.
    # Adds padding
    def encode_labels(self, data)
    # get word index size
    def get_n_words(self)
    # get suf index size
    def get_n_sufs(self)
    # get label index size
    def get_n_labels(self)
    # get index for given word
    def word2idx(self, w)
    # get index for given suffix
    def suff2idx(self, s)
    # get index for given label
    def label2idx(self, l)
    # get label name for given index
    def idx2label(self, i)
```

Required functions - `network.py`

```python
class nercLSTM(nn.Module):
   def __init__(self, codes):
      super(nercLSTM, self).__init__()

      n_words = codes.get_n_words()
      n_sufs = codes.get_n_sufs()
      n_labels = codes.get_n_labels()

      self.embW = nn.Embedding(n_words, 100)
      self.embS = nn.Embedding(n_sufs, 50)

      self.dropW = nn.Dropout(0.1)
      self.dropS = nn.Dropout(0.1)

      self.lstm = nn.LSTM(150, 200, bidirectional=True, batch_first=True)
      self.out = nn.Linear(400, n_labels)


   def forward(self, w, s):
      x = self.embW(w)
      y = self.embS(s);
      x = self.dropW(x)
      y = self.dropS(y)

      x = torch.cat((x, y), dim=2)
      x = self.lstm(x)[0]
      x = self.out(x)
      return x
```

Network architecture



Outline

Build a good NN-based drug NERC

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

- Embedding dimension
- Initialitzing word embeddings with available pretrained models
- Max length and suffix length values
- Number of LSTM units
- Used optimizer
- 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 "O" (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

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.

Neural Networks NERC

General Structure

Detailed Structure

Core task

Goals & Deliverables

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 70% 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