Practical 2: Text Classification

[Chris Dyer, Yannis Assael]

TED stands for "Technology, Entertainment, and Design". Each talk in the corpus is labeled with a series of open labels by annotators, including the labels "technology", "entertainment", and "design". Although some talks are about more than one of these, and about half aren't labeled as being about any of them! In this assignment, you will build a text classification model that predicts whether a talk is about technology, entertainment, or design–or none of these.

This is an instance of what is called "multi-label classification" (MLC), where each instance may have many different labels. However, we will start off by converting it into an instance of multi-class classification, where each document receives a single label from a finite discrete set of possible labels.

Setup and installation

To answer the following questions you are allowed to use any machine learning framework of your taste. The practical demonstrators can provide help for:

- PyTorch (example for regression)
- TensorFlow (example for logistic regression)

Other suggested frameworks: CNTK, Torch, Caffe.

Multi-class classification

Data preparation

You should reserve the first 1585 documents for training, the subsequent 250 for validation, and the final 250 for testing. Each document will be represented as a pairs of (text, label).

Text of talks Using the training data, you should determine what vocabulary you want for your model. A good rule of thumb is to tokenise and lowercase the text (you did this in the intro practical).

At test time, you will encounter words that were not present in the training set (and they will therefore not have an embedding). To deal with this, map these words to a special token. You will also want to ensure that your training set contains tokens.

Labels Each document should be labeled with a label from the set: {Too, oEo, ooD, TEo, ToD, oED, TED, ooo}.

- "Technology" $\mapsto$ Too
- "Entertainment" $\mapsto$ oEo
- "Design" $\mapsto$ ooD
- "Technology" and "Entertainment" $\mapsto$ TEo
- "Technology" and "Design" $\mapsto$ ToD

Model

A simple multilayer perceptron classifier operates as follows:

```
$\mathbf{x} = \text{embedding}(\textit{text})$
$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$
$\mathbf{u} = \mathbf{V}\mathbf{h} + \mathbf{c}$
$\mathbf{p} = \text{softmax}(\mathbf{u})$
if testing:
    prediction $= \arg\max_{y'} \mathbf{p}_{y'}$
else:     # training, with y as the given gold label
    loss $= -\log(\mathbf{p}_y)$     # cross entropy criterion
```

We will discuss the embedding function that represents the text as a vector ($\mathbf{x}$) is discussed below. $\mathbf{W}$ and $\mathbf{V}$ are appropriately sized matrices of learned parameters, $\mathbf{b}$ and $\mathbf{c}$ are learned bias vectors. The other vectors are intermediate values.

Text embedding function The text embedding function converts a sequence of words into a fixed sized vector representation. Effective models for representing documents as vectors is an open area of research, but in general, trying a few different architectures is important since the optimal architecture depends both on the availability of data and the nature of the problem being solved.

- An astoundingly simple but effective embedding model is the "bag-of-means" representation. Let each word wi in the document (where i ranges over the tokens) be represented by an embedding vector $\mathbf{x}_i$. The bag of means representation is

$$\mathbf{x} = \frac{1}{N}\sum_i \mathbf{x}_i.$$

  Word embeddings can be learned as parameters in the model (either starting from random values or starting from a word embedding model, such as word2vec or GloVe), or they you can use fixed values (again, word2vec or Glove).

- A more sophisticated model uses an bidirectional RNN (/LSTM/GRU) to "read" the document (from left to right and from right to left), and then represents the document by pooling the hidden states across time (e.g., by simply taking their arithmetic average or componentwise maximum) and using that as the document vector. You can explore this next week for your practical assignment on RNNs.

Questions

1. Compare the learning curves of the model starting from random embeddings, starting from GloVe embeddings (http://nlp.stanford.edu/data/glove.6B.zip; 50 dimensions) or fixed to be the GloVe values. Training in batches is more stable (e.g. 50), which model works best on training vs. test? Which model works best on held-out accuracy?
2. What happens if you try alternative non-linearities (logistic sigmoid or ReLU instead of tanh)?
3. What happens if you add dropout to the network?
4. What happens if you vary the size of the hidden layer?
5. How would the code change if you wanted to add a second hidden layer?
6. How does the training algorithm affect the quality of the model?
7. Project the embeddings of the labels onto 2 dimensions and visualise (each row of the projection matrix V corresponds a label embedding). Do you see anything interesting?

(Optional, for enthusiastic students) Try the same prediction task using a true multi-label classification (MLC) set up. Although a very simple approach is to make a bunch of binary classification decisions (one for each label), MLC is an interesting problem in its own right (the name of the game is modeling label decisions jointly, exploiting correlation structure between them), and neural networks offer some really interesting possibilities for modeling MLC problems that have yet to be adequately explored in the literature.

Handin On paper, show a practical demonstrator your response to these to get signed off.