University of BRISTOL

Neural networks: Chapter 7, Speech and Language Processing, 3rd edition draft, Jurafsky & Martin (2021).

7.1 Deep Learning for Text

Edwin Simpson

Department of Computer Science,

University of Bristol, UK.

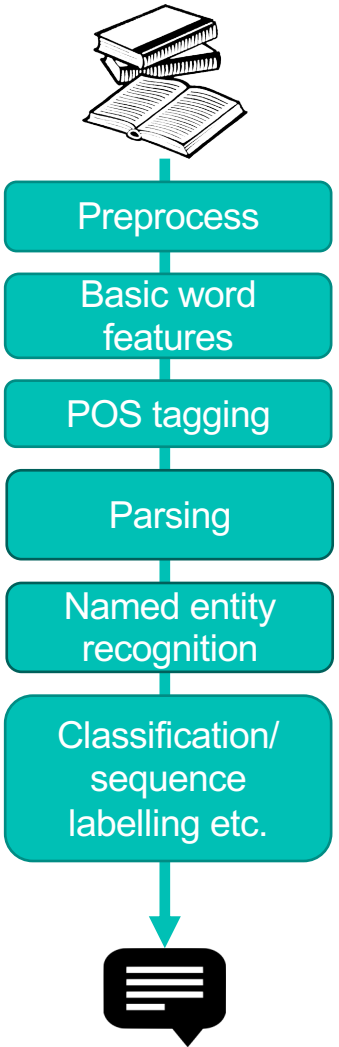
bristol.ac.uk

Advanced Text Analytics

- The remaining lectures on this unit introduce you to neural network-based methods for text analytics
- Prerequisite: Introduction to Data Analytics
- Coursework:
  - 30% of the unit
  - Two Jupyter lab notebooks
  - Complete the tasks, write your answers in the notebooks, submit the notebooks on Blackboard
  - See Blackboard for further information.

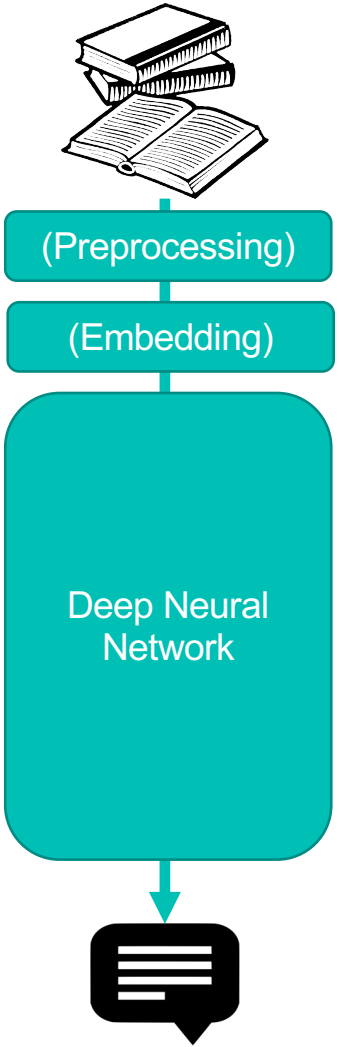
NLP Pipelines

- The approaches we have seen before depend on **feature engineering**
- We first extract features that encode words, then syntax, then semantics
- Alternatively, use end-to-end deep learning with neural networks.



End-to-end Deep Learning

- DNNs use multiple hidden layers to encode **complex nonlinear functions**.
- DNNs **learn features** at various levels of abstraction.
- Analogous to our NLP pipeline, which extracts word-level, syntactic and then semantic features.
- Use deep neural networks (DNNs) to map almost raw text to the required outputs.
- Why do we need more complex, nonlinear functions?



Interacting Features

- Sentiment analysis example: “The movie was not exciting.”
  - Naïve Bayes and logistic regression treat ‘exciting’ as indicating positive sentiment.
  - But ‘not’ and ‘exciting’ features interact to mean negative sentiment.
  - These classifiers cannot model this, as they are linear.
- We use regular expressions to replace ‘exciting’ with ‘NOT_exciting’ in the bag of words representation.
- Interactions between words can be much more complex: wouldn’t it be better if the model learned to handle them, rather hand-crafting solutions like ‘NOT_exciting’?

A Two-Step Text Classifier

We could build a small text classification pipeline as follows:

1. Train a series of logistic regression classifiers to predict different features, ***z,*** from the text, ***x***.
2. Train a logistic regression classifier to predict the class label, *y,* from the probabilities of features ***z*** produced in step 1.

The two steps allow the classifier to model more complex functions of the raw text.

Section 3.1, Introduction to Natural Language Processing, Eisenstein, J. (2019).

A Two-Step Text Classifier

Both steps have the same functional form:

1. $h_i = p(z_i|\boldsymbol{x}) = \sigma(\boldsymbol{w}_i^{(1)}\boldsymbol{x})$
   - $\boldsymbol{w}_i^{(1)}$ are weight parameters of the classifier for feature i as a row vector.
   - $\boldsymbol{x}$ is a column vector representing a sequence of words.
2. $p(y = j|\boldsymbol{h}) = \sigma(\boldsymbol{w}_j^{(2)}\boldsymbol{h})$
   - $\boldsymbol{w}_j^{(2)}$ are weight parameters of the classifier for class label j as a row vector.
   - $\boldsymbol{h}$ is a column vector containing feature probabilities from the first step.

Each step computes a weighted sum of its inputs, then applies a sigmoid function.

Section 3.1, Introduction to Natural Language Processing, Eisenstein, J. (2019).

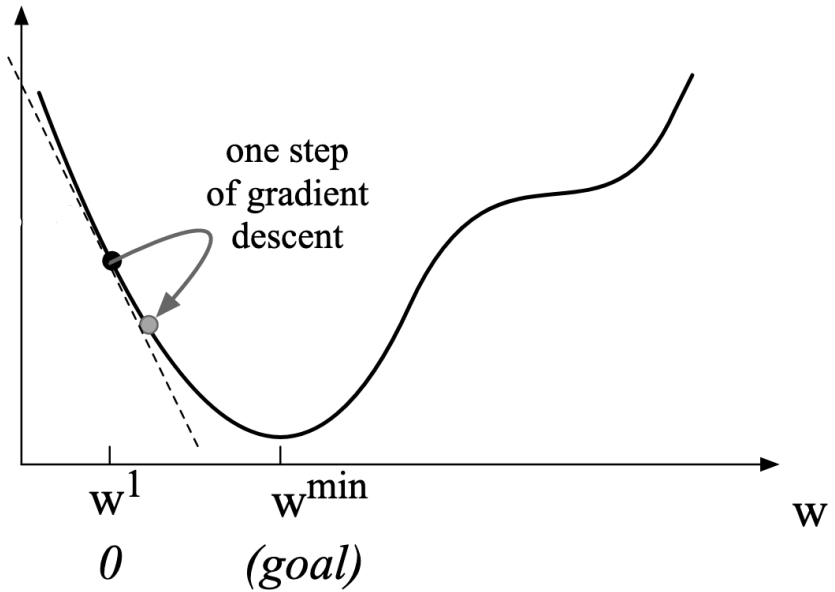
A Two-Step Text Classifier

- Each logistic regression classifier is identical to a node in a neural network!
- Our two-step classifier is a **feedforward neural network**, sometimes called a **multilayer perceptron**.
  - Here, we use the logistic sigmoid as the **activation function.**
  - The probabilities $\boldsymbol{h}$ produced by the first step are **hidden layer activations.**
  - The weights $\boldsymbol{w}_i^{(1)}$ for all i can be arranged into a matrix $\boldsymbol{W}^{(1)}$.

Section 3.1, Introduction to Natural Language Processing, Eisenstein, J. (2019).

Training: Gradient Descent

- As in skip-gram, we iteratively adjust each weight to minimise loss function L.
- To compute the gradient for the hidden layer (step 1), we pass partial derivatives back from step 2.
- This **backpropagation** allows us to train the hidden layer without any examples of ***z***.
- Hence, we can learn feature representations with a NN!



Limitations of NNs

- So we can model nonlinear functions and extract features automatically, where's the catch?
- Integrating knowledge of how language works – hard to learn all that from a typical training set.
- Interpretability: hard to understand what the features mean and how the network assigns outputs, so it's hard to trust it on unseen data.
- Complex model requires big training datasets and can be prone to overfitting.
- Computational costs of both training and prediction.