Lecture 28: Review and Outlook

Today’s class

A very cursory overview of neural nets for NLP. (without assuming any machine learning background or going into much technical depth).

Why neural nets/deep learning?
Very active area of current research.
A lot of hype in the media (brains! AI! etc.)
Challenges many of the assumptions we’ve been making in this class.
Likely to become a part of the standard NLP toolbox in the future (although it may not make traditional NLP obsolete either)

The topics of this class

We want to identify the structure and meaning of words, sentences, texts and conversations
N.B.: we do not deal with speech (no signal processing)

We mainly deal with language analysis/understanding, not language generation/production

We focus on fundamental concepts, methods, models, and algorithms, not so much on current research:
- Data (natural language): linguistic concepts and phenomena
- Representations: grammars, automata, etc.
- Statistical models over these representations
- Learning & inference algorithms for these models
The NLP Pipeline

An NLP system may use some or all of the following steps:

**Tokenizer/Segmenter**
- to identify words and sentences

**Morphological analyzer/POS-tagger**
- to identify the part of speech and structure of words

**Word sense disambiguation**
- to identify the meaning of words

**Syntactic/semantic Parser**
- to obtain the structure and meaning of sentences

**Coreference resolution/discourse model**
- to keep track of the various entities and events mentioned

Two core problems for NLP

**Ambiguity:** Natural language is highly ambiguous
- Words have multiple senses and different POS
- Sentences have a myriad of possible parses
- etc.

**Coverage** (compounded by Zipf’s Law)
- Any (wide-coverage) NLP system will come across words or constructions that did not occur during training.
- We need to be able to generalize from the seen events during training to unseen events that occur during testing (i.e. when we actually use the system).

Statistical models for NLP

We’ve talked a lot about various statistical models as a way to handle both the ambiguity and the coverage issues.
- Probabilistic models (e.g. HMMs, MEMMs, CRFs, PCFGs)
- Learning-based classifiers

Basic approach:
- Decide what kind of model to use
- Define features that could be useful (especially for classifiers), or decide on the tag set/grammar to be used
- Train and evaluate the model.

Features for NLP

Many systems use explicit features:
- Words (does the word “river” occur in this sentence?)
- POS tags
- Chunk information, NER labels
- Parse trees or syntactic dependencies (e.g. for semantic role labeling, etc.)

Feature design is usually a big component of building any particular NLP system.

Which features are useful for a particular task and model typically requires experimentation, but I hope this class has exposed you to many of the commonly used ones.
“Deep learning” for NLP

What is “deep learning”?

Neural networks, typically with several hidden layers (depth = # of hidden layers)
Single-layer neural nets can be equivalent to linear classifiers
Multi-layer neural nets are more expressive

Very impressive performance gains in computer vision (ImageNet) and speech recognition over the last several years.

Neural nets have been around for decades.
Why have they suddenly made a comeback?
Fast computers (GPUs!) and (very) large datasets have made it possible to train these very complex models.

What are neural nets?

Simplest variant: single-layer feedforward net

For binary classification tasks:
Return 1 if y > 0.5
Return 0 otherwise

For multiclass classification tasks:
Each element
yi = class i
Return max(yi)

Feedforward networks

Single-layer (linear) feedforward network
\[ y = wx + b \] (binary classification)
\( w \) is a weight vector, \( b \) is a bias term (also a scalar)

This is just a linear classifier (aka Perceptron)
(the output \( y \) is a linear function of the input \( x \))

Single-layer non-linear feedforward networks:
Pass \( wx + b \) through a non-linear activation function, e.g. \( y = \tanh(wx + b) \)
What are neural nets?
We can generalize this to multi-layer feedforward nets:

- **Input layer:** vector \( x \)
- **Hidden layer:** vector \( h_1 \)
- **Hidden layer:** vector \( h_n \)
- **Output layer:** vector \( y \)

Motivation for deep learning in NLP: Features can be brittle

Word-based features:
- How do we handle unseen/rare words?

Any supervised NLP model has to be trained on labeled data.
- Producing labeled data can be very expensive.
- We typically don’t have enough labeled data from the domain of interest.
- If features are produced by other NLP systems (POS tags, dependencies, etc.), these systems may not work well on our domain of interest.

Challenges in using NNs for NLP

Our input and output variables are discrete: words, labels, structures.

NNs work best with continuous vectors.
- We typically want to learn a mapping (embedding) from discrete words (input) to dense vectors.
- We can do this with (simple) neural nets and related methods.

The input to a NN is (traditionally) a fixed-length vector. How do you represent a variable-length sequence as a vector?
- We can use recurrent neural nets: read in one word at the time to predict a vector, use that vector and the next word to predict a new vector, etc.

NLP applications of NNs

Word embeddings (word2vec, Glove, etc.)
- Train a NN to predict a word from its context (or the context from a word).
- This gives a dense vector representation of each word

Neural language models:
- Use recurrent neural networks (RNNs) to predict word sequences
- More advanced: use LSTMs (special case of RNNs)
- In machine translation: use one RNN to encode source string, and another RNN to decode this into a target string.
- Also used for automatic image captioning, speech, etc.

Recursive neural networks:
- Used for parsing
Further reading

Stanford class on deep learning for NLP
(Richard Socher)
http://cs224d.stanford.edu

Yoav Goldberg’s Primer on Neural Nets for NLP

More generally on deep learning:
http://deeplearning.net