CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 28: Neural approaches to NLP

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

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)

Review:
Back to the Basics…

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)
- Other machine 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.

Motivation for neural approaches to NLP: Features can be brittle

Word-based features:

- How do we handle unseen/rare words?

Many features are produced by other NLP systems (POS tags, dependencies, NER output, etc.)
These systems are often trained on labeled data.

- Producing labeled data can be very expensive.
- We typically don't have enough labeled data from the domain of interest.
- We might not get accurate features for our domain of interest.

Features in neural approaches

Many of the current successful neural approaches to NLP do not use traditional discrete features.

Words in the input are often represented as dense vectors (aka. word embeddings, e.g. word2vec)

- Traditional approaches: each word in the vocabulary is a separate feature. No generalization across words that have similar meanings.
- Neural approaches: Words with similar meanings have similar vectors. Models generalize across words with similar meanings

Other kinds of features (POS tags, dependencies, etc.) are often ignored.

Motivation for neural approaches to NLP: Markov assumptions

Traditional sequence models (n-gram language models, HMMs, MEMMs, CRFs) make rigid Markov assumptions (bigram/trigram/n-gram).

Recurrent neural nets (RNNs, LSTMs) can capture arbitrary-length histories without requiring more parameters.

Neural approaches to NLP

What is “deep learning”?

Neural networks, typically with several hidden layers

- (depth = # of hidden layers)
- Single-layer neural nets are 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.

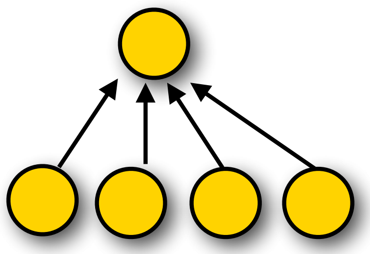
Neural nets have been around for decades.
Why have they suddenly made a comeback?

- 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

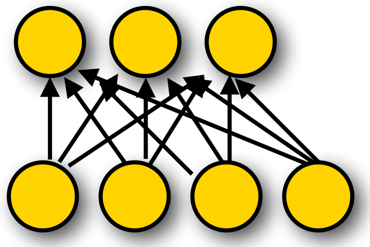
For binary classification tasks:
Single output unit
Return 1 if $y > 0.5$
Return 0 otherwise



Output unit: scalar *y*

Input layer: vector **x**

For multiclass classification tasks:
K output units (a vector)
Each output unit
y_i = class i
Return $\text{argmax}_i(y_i)$



Output layer: vector **y**

Input layer: vector **x**

Multiclass models: softmax(y_i)

Multiclass classification = predict one of K classes.

Return the class i with the highest score: $\mathrm{argmax}_i(y_i)$

In neural networks, this is typically done by using the **softmax** function, which maps real-valued vectors in R^N into a distribution over the N outputs

For a vector $\mathbf{z} = (z_0 \ldots z_K)$: $P(i) = \mathrm{softmax}(z_i) = \exp(z_i) \,/\, \sum_{k=0..K} \exp(z_k)$

(NB: This is just logistic regression)

Single-layer feedforward networks

Single-layer (linear) feedforward network

$y = \mathbf{wx} + b$ (binary classification)

$\mathbf{w}$ is a weight vector, b is a bias term (a scalar)

This is just a linear classifier (aka Perceptron)

(the output y is a linear function of the input $\mathbf{x}$)

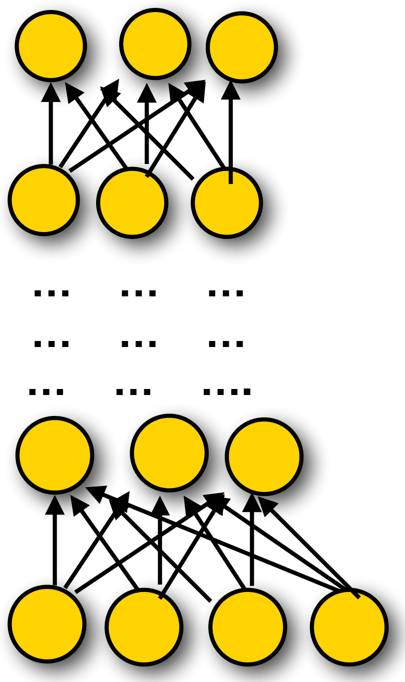
Single-layer non-linear feedforward networks:

Pass $\mathbf{wx} + b$ through a non-linear activation function,

e.g. $y = \tanh(\mathbf{wx} + b)$

Multi-layer feedforward networks

We can generalize this to multi-layer feedforward nets



Output layer: vector **y**

Hidden layer: vector $\mathbf{h}_n$

Hidden layer: vector $\mathbf{h}_1$

Input layer: vector **x**

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?

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)

Sequence-to-sequence (seq2seq) models:

- From 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, etc.

Recursive neural networks:

- Used for parsing

Neural Language Models

LMs define a distribution over strings: $P(w_1 \ldots. w_k)$
LMs factor $P(w_1 \ldots. w_k)$ into the probability of each word:
$P(w_1 \ldots. w_k) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1 w_2) \cdot \ldots \cdot P(w_k \mid w_1 \ldots. w_{k-1})$

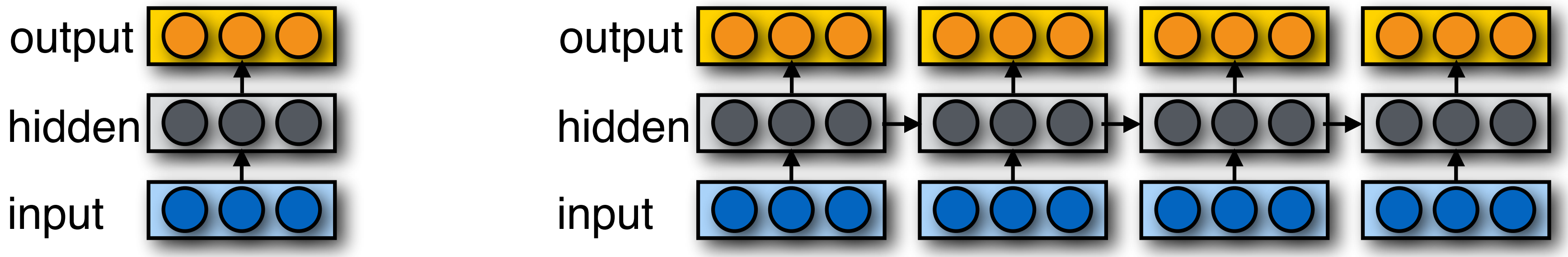
A neural LM needs to define a distribution over the V words in the vocabulary, conditioned on the preceding words.

Output layer: V units (one per word in the vocabulary) with softmax to get a distribution
Input: Represent each preceding word by its d-dimensional embedding.
- Fixed-length history (n-gram): use preceding n–1 words
- Variable-length history: use a recurrent **neural net**

Recurrent neural networks (RNNs)

Basic RNN: Modify the standard feedforward architecture (which predicts a string $w_0 \ldots w_n$ one word at a time) such that the output of the current step (w_i) is given as additional input to the next time step (when predicting the output for w_{i+1}).

“Output” — typically (the last) hidden layer.



Feedforward Net **Recurrent Net**

Word Embeddings (e.g. word2vec)

Main idea:
If you use a feedforward network to predict the probability of words that appear in the context of (near) an input word, the hidden layer of that network provides a dense vector representation of the input word.

Words that appear in similar contexts (that have high distributional similarity) wils have very similar vector representations.

These models can be trained on large amounts of raw text (and pretrained embeddings can be downloaded)

Seq2seq models

Task (e.g. machine translation):

Given one variable length sequence as input,
return another variable length sequence as output

Main idea:

Use one RNN to encode the input sequence (“encoder”)
Feed the last hidden state as input to a second RNN (“decoder”) that then generates the output sequence.

Further reading

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

Yoav Goldberg's Primer on Neural Nets for NLP
http://arxiv.org/pdf/1510.00726.pdf

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