

Quora

Intro to common NLP techniques

Ryan Cheu - ML Engineer @ Quora, MIT '15

Setup

- `git clone https://github.com/ryancheu/NLPWorkshop`
- Download <http://nlp.stanford.edu/data/glove.6B.zip>
 - Unzip to `NLPWorkshop/data/glove.6B`
- `jupyter-notebook NLPWorkshop`

Common Natural Language Processing Use-cases

- Text classification
 - Is this a positive or negative review?
 - Is this comment spam?
 - What topic is this news article on?
- Text similarity
 - Given a news article, find similar articles on the same subject
 - Given some text, find the ad that's most relevant
- Text Search
- Machine Translation
- Text Summarization
- Information Retrieval, Question Answering

Today

- Basics and building blocks of NLP commonly used in industry
- Traditional text operations
- Text embeddings, machine learning

Text Processing (part 1)

Question Duplicates

Text1	Text2	Duplicate
How can one increase concentration?	How can I improve my concentration?	1
What year did the letter J come?	When was the Letter J created?	1
Is on-line gambling legal in Norway?	Is on-line gambling legal in Sweden?	0
"Tennis: What is it called when you win a game from 40-0?"	"In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"	0

Raw word comparison

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"

def preprocess(question):
    question = question.split()
    return question

set(preprocess(q1)).intersection(preprocess(q2))
```

Out:

```
{'called'}
```


Lower case

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"

def preprocess(question):
    question = question.lower()
    question = question.split()
    return question

set(preprocess(q1)).intersection(preprocess(q2))
```

Out:

```
{'called'}
```

Lower case

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"

def preprocess(question):
    question = question.lower()
    question = question.split()
    return question
print(preprocess(q1))
print(preprocess(q2))
```

Out:

```
['tennis:', 'what', 'is', 'it', 'called', 'when', 'you', 'win', 'a', 'game', 'from', '40-0?']
['in', 'tennis,', 'why', "isn't", '30-30', 'called', 'deuce?', 'and', '40-30', '(or', '30-40)',
'called', 'advantage?']
```

Tokenization

- Break text up into small “tokens”
- Choices about punctuation
 - “Isn’t”
 - Isn ’t
 - Is n’t
 - Isn ’ t
 - “U.S.A.”
 - U . S . A .
 - U.S.A.
 - U. S. A.

Tokenization

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"
def preprocess(question):
    question = question.lower()
    question = nltk.word_tokenize(question)
    return question

print(set(preprocess(q1)).intersection(preprocess(q2)))
print(preprocess(q2))
```

Out:

```
{'called', 'is', 'tennis', '?'}
```

```
['in', 'tennis', ',', 'why', 'is', "n't", '30-30', 'called', 'deuce', '?', 'and', '40-30', '(', 'or', '30-40', ')', 'called', 'advantage', '?']
```

Stopwords

- Not all words equally important
- Remove most common words (“stopwords”)
 - Short function words: “the”, “he”, “a” ...
 - “below”, “above”, “between”
- Frequently used in search
- Can help save memory

What is the difference between an interpreter and a compiler?

Stopwords

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"
stopwords = set(nltk.corpus.stopwords.words("english"))
def preprocess(question):
    question = question.lower()
    question = nltk.word_tokenize(question)
    question = [t for t in question if t not in stopwords]
    return question
print(set(preprocess(q1)).intersection(preprocess(q2)))
```

Out:

```
{'called', 'tennis', '?'}
```

In []:

Small note

- Previous slide has a bug
- NLTK tokenizer does not use same format as stopwords
- `preprocess(q2)`
 - `['tennis', ',', 'n't', '30-30', 'called', 'deuce', '?', '40-30', '(', '30-40', ')', 'called', 'advantage', '?']`
- `[s for s in stopwords if s.startswith('is')]`
 - `['isn', 'is', 'isn't']`

Stemming

- Several words with the same base:
 - “calculate”, “calculating”, “calculated”
- Stemmers reduce words to a base word
 - Do not take context into account
- More useful in languages with more forms
 - “faire”, “fait”, “faisons”, “faite”, “fais”

```
["calculate", "calculating", "calculated"] -> <stemmer> ->  
["calcul", "calcul", "calcul"]
```


Stemming

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"
stemmer = nltk.stem.PorterStemmer()
def preprocess(question):
    question = question.lower()
    question = nltk.word_tokenize(question)
    question = [t for t in question if t not in stopwords]
    question = [stemmer.stem(t) for t in question]
    return question
print(set(preprocess(q1)).intersection(preprocess(q2)))
```

Out:

```
{'tenni', '?', 'call'}
```

Lemmatization

- Similar to stemming, but also considers part of speech / morphology

```
[("calculate", verb), ("calculating", adjective),  
("calculated", verb), ("calculated", noun)] ->
```

```
<lemmatization> ->
```

```
["calculate", "calculating", "calculate", "calculated"]
```

Lemmatization

In:

```
q1 = "Tennis: What is it called when you win a game from 40-0?"
q2 = "In tennis, why isn't 30-30 called Deuce? And 40-30 (or 30-40) called Advantage?"
lemmatizer = nltk.stem.WordNetLemmatizer()
def preprocess(question):
    question = question.lower()
    question = nltk.word_tokenize(question)
    token_pos_pairs = nltk.pos_tag(question) # returns pairs like ("tennis", "NN")
    question = [lemmatizer.lemmatize(p[0], get_wordnet_pos(p[1])) for p in token_pos_pairs]
    return question

print(set(preprocess(q1)).intersection(preprocess(q2)))
```

Out:

```
{'call', '?', 'be', 'tennis'}
```

Classification

- Goal: Given two questions, return a probability that they're duplicates
- Steps:
 - Compute similarity for question pair
 - Return True if similarity > threshold
 - Compare with real labels

Classification

In:

```
def basic_predictor(q1, q2, threshold=3):  
    """Takes two questions, returns True if the intersection of their tokens is larger than threshold."""  
    q1_tokens = set(preprocess(q1))  
    q2_tokens = set(preprocess(q2))  
    return len(q1_tokens.intersection(q2_tokens)) >= threshold  
print(basic_predictor(q1, q2))
```

Out:

```
True
```

Measurement

- Measure:
 - Accuracy
 - what percent did we classify correctly?
 - Precision
 - Of pairs marked as duplicates, what percent were true duplicates?
 $|True\ Positive| / (|True\ Positive| + |False\ Positive|)$
 - Recall
 - Of all duplicate pairs, what percent did we detect as duplicates? $|True\ Positive| / (|True\ Positive| + |False\ Negative|)$

Measurement

In:

```
def eval_predictor_binary(predictor, question_pairs, labels, **kwargs):
    """Given a predictor function and data to evaluate on, calculate recall, precision and accuracy."""
    y_pred = []
    for pair in question_pairs:
        y_pred.append(int(predictor(*pair, **kwargs)))

    # Of all labels that should have been predicted positive, how many were?
    recall = sklearn.metrics.recall_score(y_true=labels,
                                           y_pred=y_pred)

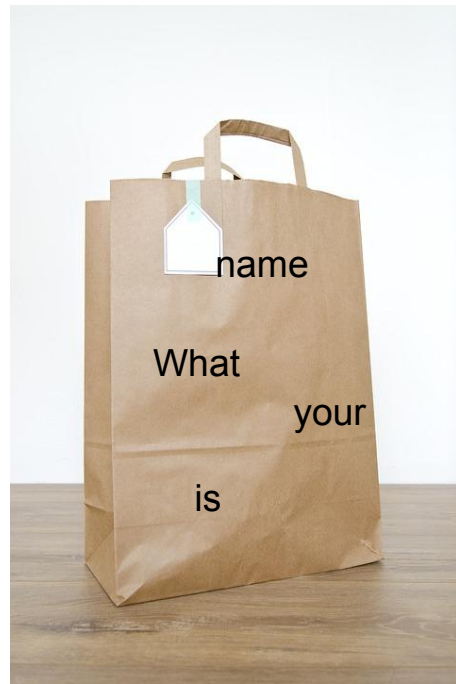
    print('recall:', recall)
```

Out:

(demo)

Similarity

- Looking at text as a “Bag of Words”
 - Ignore ordering for now
- Scale numbers to $[0,1]$ range for ease of use
- Some options:
 - Number of tokens in common / 10.0
 - $\text{Number of tokens in common} / \max(\text{Number of tokens})$
 - Jaccard Similarity
 - $\text{Number of tokens in common} / \text{union}(\text{tokens of } q1, \text{tokens of } q2)$



Similarity

In:

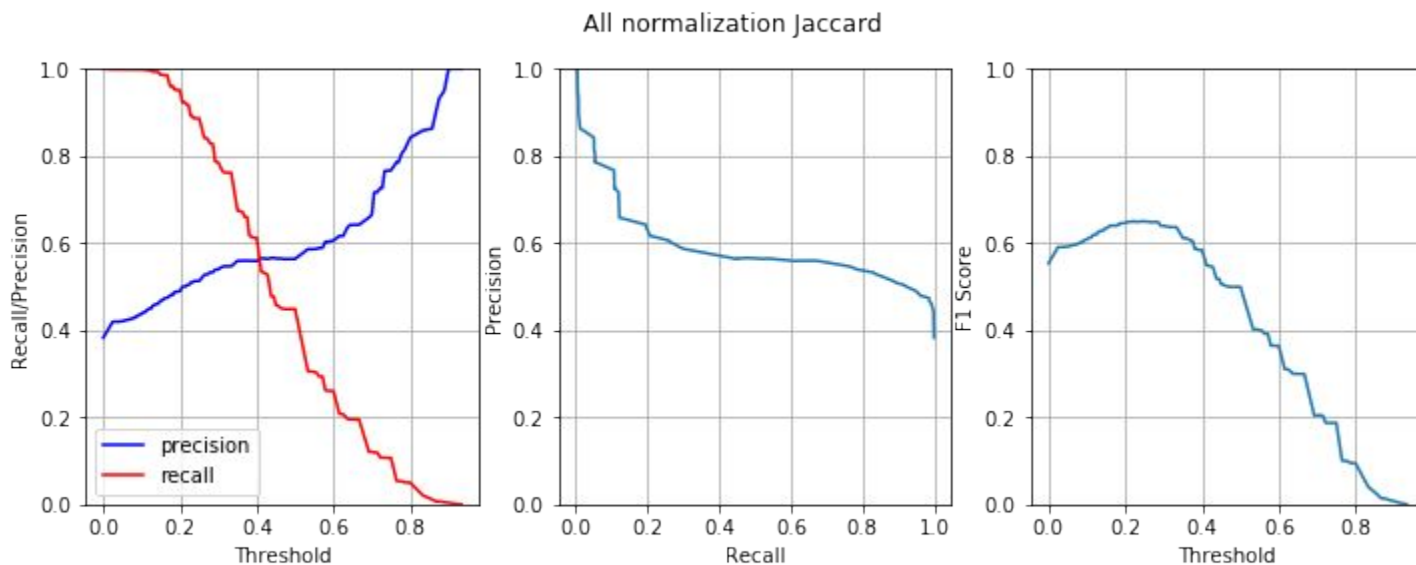
```
def basic_prob_predictor(q1, q2, use_jaccard=False, **kwargs):
    """Basic real valued prediction using jaccard similarity, or a simple measure of intersection/10."""
    q1_tokens = set(preprocess_question(q1, **kwargs))
    q2_tokens = set(preprocess_question(q2, **kwargs))
    q_intersect = len(q1_tokens.intersection(q2_tokens))
    if use_jaccard:
        q_union = 1 + len(q1_tokens.union(q2_tokens))
        return q_intersect/q_union
    else:
        return q_intersect/10
```

Out:

(demo)

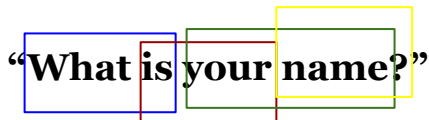
Measurement (pt 2)

- Precision Recall Curves
- F1 Score



N-grams

- Sometimes, the order of words in a sentence matter
- Word Bi-grams (2-gram) of: “What is your name?”
 - “What is”
 - “is your”
 - “your name”
 - “name ?”
- May also see character n-grams when using machine learning



“What is your name?”

- See: `def produce_word_n_grams(tokens, n_values=[1,2]):`

TF-IDF

- Not all words have equal information
- Rarer words often more important than common words
- Term Frequency:
 - How often a term appears in a given document.
- Document Frequency:
 - Percent of documents a term occurs in
- Inverse Document Frequency
 - Inverse of Document Frequency, usually log scaled.
 - **$\text{idf} = \log(1/\text{Document Frequency})$**
- **tf-idf**
 - **$\text{tf} * \text{idf}$**

TF-IDF

- “What is the difference between an interpreter and a compiler?”
- In this case, term frequency is 1 for all terms.

IDF	Term
1.94	What
2.14	is
1.97	the
5.1	difference
4.69	between
4.14	an
11.94	interpreter
2.89	and
<removed>	a
9.67	compiler

Didn't have time for:

- Dependency parsing/trees
 - What is the structure of a sentence?
 - Surprisingly rarely used in modern models
- Named Entity Recognition
 - Determine what people, places, companies, etc. are present

SpaCy

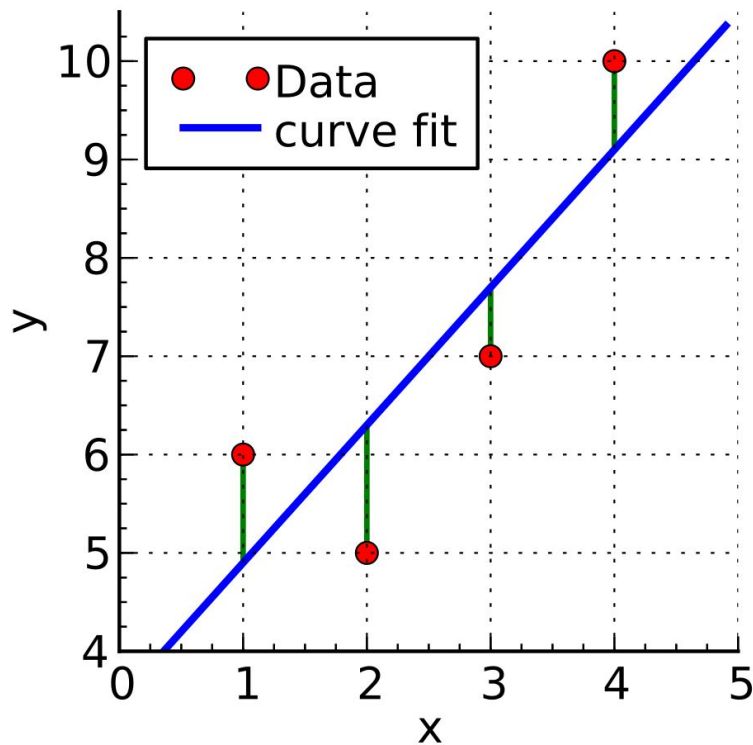
- Replacement for many NLTK operations
- Generally faster, more accurate
- A little more complicated initially, so wasn't used in this presentation

	SPACY	SYNTAXNET	NLTK	CORENLP
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

Machine Learning (part 2)

What is machine learning?

- Algorithms that learn parameters
- “Glorified Curve Fitting”
 - (kind of)
- Wide field, too much to teach today

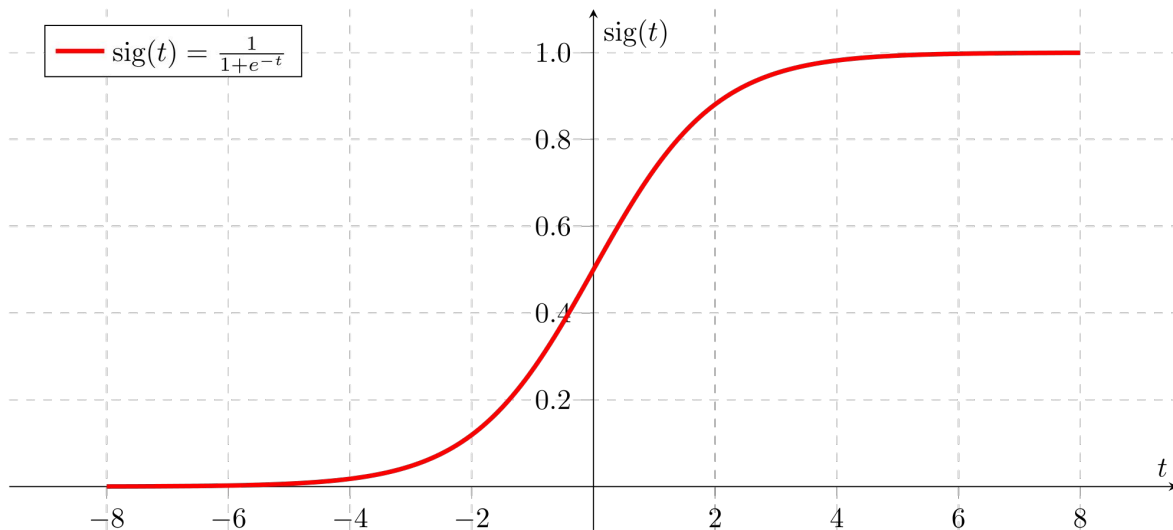


Steps

- Acquire training data
- Think of numbers that could describe your data in useful ways
 - How many tokens in common?
 - How many tokens in each question?
 - Does the sentence start with “Why” (1 or 0)
 - How many nouns are in each question?
 - How many n-grams are in common?

Logistic Regression

- Simplest model for predicting classification labels
- If you have real valued data instead of labels, use linear regression instead
- Very fast to compute, but does not handle complex relationships between features



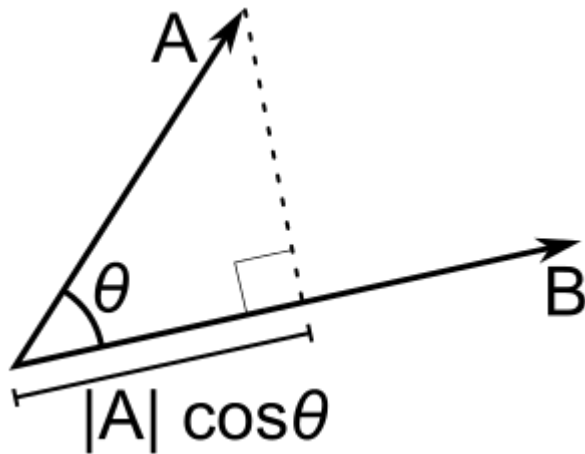
Vectors

- Essentially, lists of numbers
- **One hot** encoding:
 - Represent words by marking 1 if word is present, 0 if not present
 - Vector is the length of the vocabulary (every word that could be present)
 - Example of “What is the difference between an interpreter and a compiler?”

an	1
and	1
ant	0
are	0
...	...
between	1
but	0
...	...
compiler	1
commuter	0
....	0

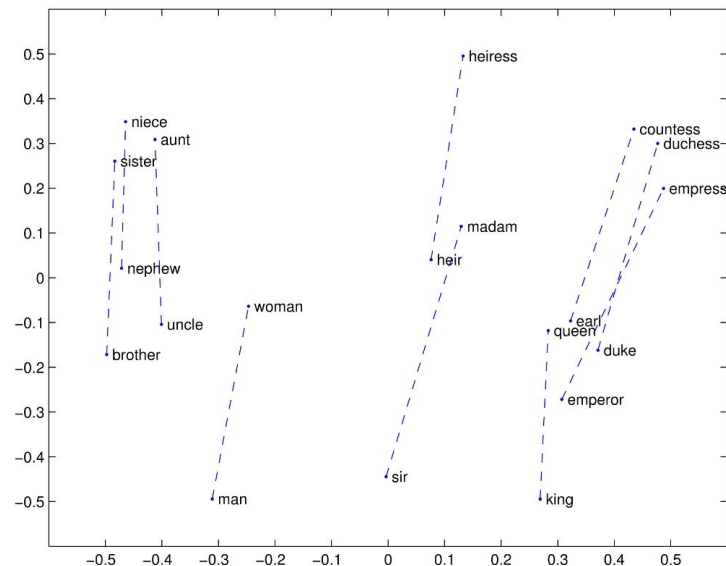
Cosine Similarity and Dot Product

- Given a two vectors, several ways to compute the similarity
- Element wise subtraction
- Dot Product
 - Sum of element-wise products
 - $|A| |B| \cos(\theta)$
 - $\text{dot}([1,2,3],[4,5,6]) = \text{sum}(1*4, 2*4, 3*6) = 30$
- Cosine Similarity
 - Cosine of angle between vectors
 - $\cos(\theta)$
 - Equivalent to the dot product of l2 normalized vectors



Word Embeddings

- Represent a word as vector such that similar words have similar vectors
- Word2Vec
 - Embedding based on other words that occur nearby
 - king - man + woman = queen
- Other word embeddings:
 - GloVe
 - fastText



Combining Word Embeddings

- Element-wise average of word embeddings
- Sentence, document embeddings
 - Universal Sentence Encoder
 - Doc2Vec
 - Active area of research
- Various Neural Network architectures

Word Embeddings for Features

In:

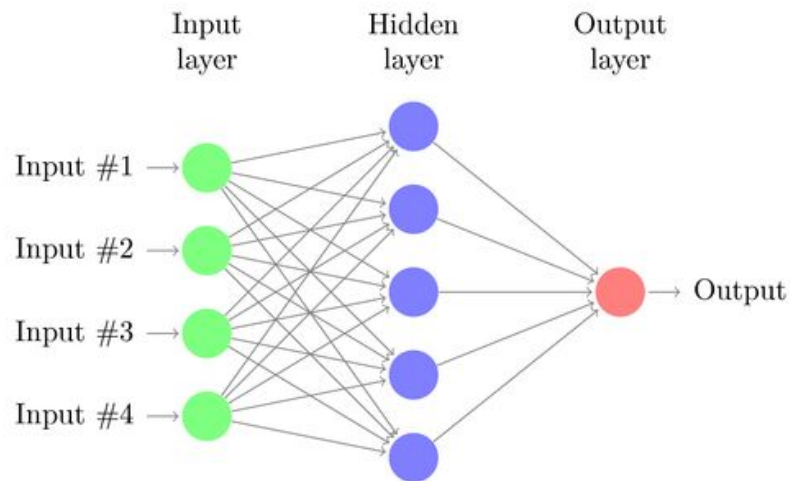
```
def add_embedding_features(X, *q_datas):  
    # will discard tokens not found in GloVe  
    word_embeddings = [[glove_dict[t] for t in q_data.tokenized  
                        if t in glove_dict] for q_data in q_datas]  
  
    # nv is a function that normalizes the vector  
    question_embeddings = [nv(np.average(word_embeds, axis=0)) for word_embeds in word_embeddings]  
    cos_sim = np.dot(*question_embeddings)
```

Out:

(demo)

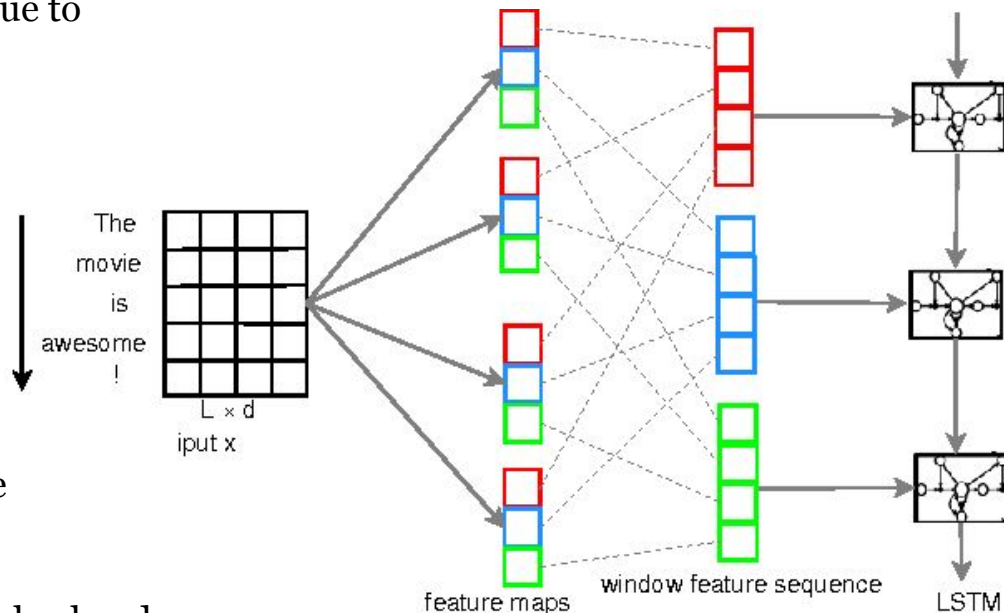
Neural Networks

- Model more relationships between variables
- Equivalent to linear/logistic regression without hidden layer
- More layers -> more complex functions can be modeled



Neural Networks for NLP

- Especially popular lately due to deep models
 - LSTM
 - CNN
 - Attention
- But!
 - Simple models still surprisingly effective
- Two popular libraries:
 - Keras w/ Tensorflow backend
 - PyTorch



Q&A