Natural Language Understanding

Lecture 3: Language modeling with neural networks

Adam Lopez

23 January 2018

School of Informatics
University of Edinburgh
alopez@inf.ed.ac.uk

Language Models

Feedforward language models

Reading: Bengio et al. 2003

Background: Jurafsky and Martin (ed. 3) 4.0-4.3

Language Models

Predict the next word!

Summer is hot winter is _____

Predict the next word!

She is drinking a hot cup of _____

Predict the next word!

In the park I saw a _____

Image captioning

A language model is a probabilistic generative model of strings

A language model assigns probabilities to sequences

- Often a simple n-gram model. Trigrams models often work well.
- applications:
  - speech recognition
  - machine translation
  - text completion
  - optical character recognition
  - image captioning
  - grammar checking

Applications of Language Modeling

Machine translation:

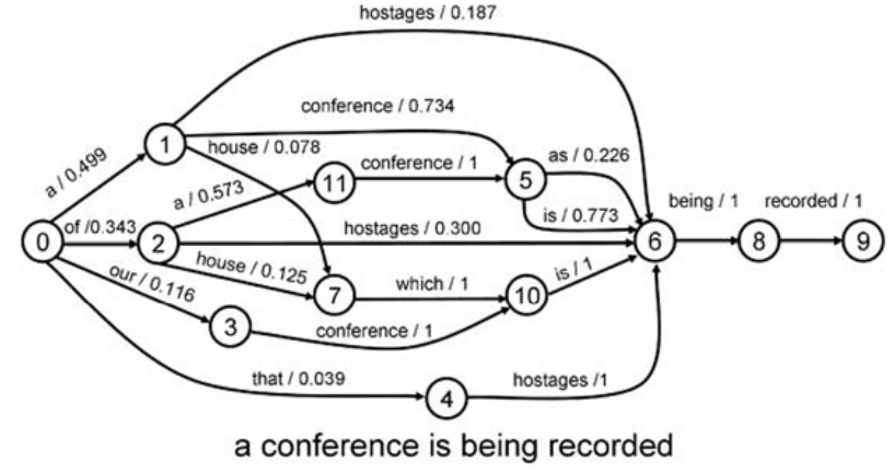
- word ordering: P(the cat is small) $>$ P(small the is cat);
- word choice: P(walking home after school) $>$ P(walking house after school).

Grammar checking:

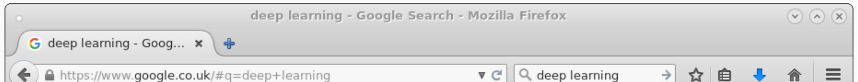
- word substitutions: P(the principal resigned) $>$ P(the principle resigned);
- agreement errors: P(the cats sleep in the basket) $>$ P(the cats sleeps in the basket).

Applications of Language Modeling

Speech recognition:



Text completion:



Language modeling as probabilistic prediction

Given a finite vocabulary V, we want to define a probability distribution $P : V^* \to \mathbb{R}_+$.

The *finite vocabulary* bit should worry you. We'll come back to this, but not today!

Revision questions:

- What is the sample space?
- What might be some useful random variables?
- What constraints do we need to satisfy?

How to derive an n-gram language model

Given a sequence of words $w_1 \dots w_k$, how do we define $P(w_1 ... w_k)$?

Let W_i be a r.v. taking value of word at position i.

Use the chain rule:

$$
\begin{aligned}
P(w_1 ... w_k) = & P(W_1 = w_1) \times \\
& P(W_2 = w_2 \mid W_1 = w_1) \times \\
& ... \\
& P(W_k = W_k \mid W_1 = w_1, ..., W_{k-1} = w_{k-1}) \\
& P(W_{k+1} = \langle \textsc{Stop} \rangle \mid W_1 = w_1, ..., W_k = w_k)
\end{aligned}
$$

Written more concisely

Use the chain rule:

$$
\begin{aligned}
P(w_1 ... w_k) = & P(w_1) \times \\
& P(w_2 \mid w_1) \times \\
& ... \\
& P(w_k \mid w_1, ..., w_{k-1}) \\
& P(\langle \textsc{Stop} \rangle \mid w_1, ..., w_k) \\
= & \prod_{i=1}^{k+1} P(w_i | w_1, ..., w_k)
\end{aligned}
$$

Defines *joint distribution* over infinite sample space in terms of *conditional distributions*, each over finite sample spaces (but with potentially infinite history!)

Make it work by making a Markov assumption: n-gram models

$$P(w_i \mid w_1, ..., w_{i-1}) \sim P(w_i \mid w_{i-n+1}, ..., w_{i-1})$$

What is $P(w_i \mid w_{i-n+1}, ..., w_{i-1})$?

Given $w_{i-n+1}, ..., w_{i-1}$, P is a probability distribution, hence:

$$P : V \rightarrow \mathcal{R}_+$$

$$\sum_{w \in V} P(w \mid w_{i-n+1}, ..., w_{i-1}) = 1$$

How can we define such a function?

Simplest idea: let $P(w_i \mid w_{i-n+1}, ..., w_{i-1})$ be a parameter (i.e. a real number) in a table indexed by $w_{i-n+1}, ..., w_i$. What are some problems with this?

Estimating n-gram Probabilities

We can get maximum likelihood estimates for the conditional probabilities from n-gram counts in a corpus:

$$P(w_2|w_1) = \frac{n_{(w_1,w_2)}}{n_{(w_1)}} \qquad P(w_3|w_1, w_2) = \frac{n_{(w_1,w_2,w_3)}}{n_{(w_1,w_2)}}$$

But building good n-gram language models can be difficult:

- the higher the n, the better the performance
- but most higher-order n-grams will never be observed—are these *sampling zeros* or *structural zeros*?
- good models need to be trained on billions of words
- this entails large memory requirements
- smoothing and backoff techniques are required.

Using n-gram Language Models

If we have a sequence of words $w_1 \dots w_k$ then we can use the language model to predict the next word w_{k+1}:

$$\hat{w}_{k+1} = \underset{w_{k+1}}{\operatorname{argmax}} P(w_{k+1}|w_1 \dots w_k)$$

Being able to predict the next word is useful for applications that process input in real time (word-by-word).

Feedforward language models

What can we estimate with a universal function approximator?

Probability simply requires us to obey the following rules (remember: V is finite):

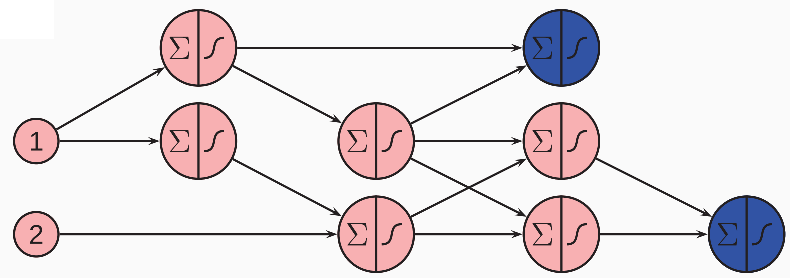
$$P : V \to \mathcal{R}_+$$

$$\sum_{w \in V} P(w \mid w_{i-n+1}, \ldots, w_{i-1}) = 1$$

In the last lecture we learned that multi-layer perceptrons were universal function approximators. And, we have a learning algorithm for them.

Can we use them to learn P?

Multilayer perceptrons have only one data type



Input: a vector of real numbers.

Output: a vector of real numbers. (Vectors of size 1 are still vectors!)

Probability distributions are vectors!

Summer is hot winter is ____

Turn any vector into a probability with the softmax function!

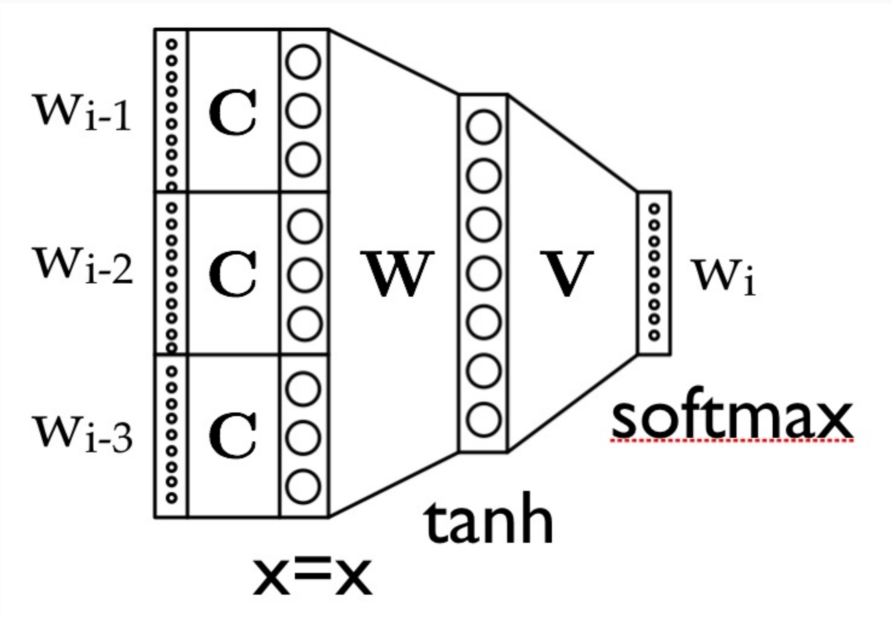
$$P(Y = y \mid X) = \frac{\exp(y \cdot w)}{\sum_{y' \in Y} \exp(y' \cdot w)}$$

- Softmax is a generalization of the logistic function
- Takes the inner product of the *representation* of every possible outcome y (a vector) and the weights w to produce a real value for the outcome.
- Exponentiation makes every value positive.
- Normalization makes everything sum to one.

Elements of discrete vocabularies are vectors!

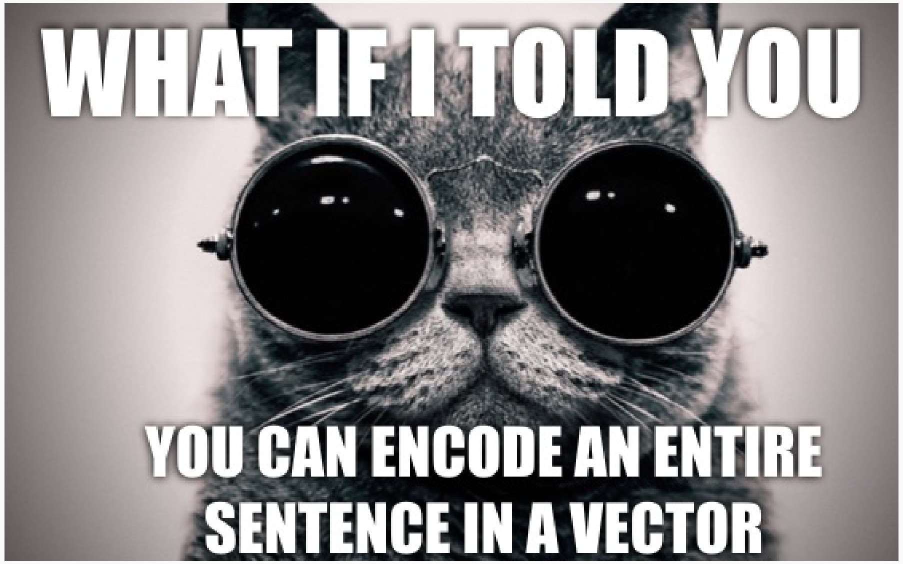
	Summer	is	hot	winter	is
is	0	1	0	0	1
cold	0	0	0	0	0
grey	0	0	0	0	0
hot	0	0	1	0	0
summer	1	0	0	0	0
winter	0	0	0	1	0

Feedforward LM: function from a vectors to a vector



Summary

- Language models assign string probabilities
- Useful for word prediction in many NLP applications
- *n*-gram models simplify language modeling via a Markov assumption
- *n*-gram models can be parameterized with simple multilayer neural network
- Many conditional probability distribution can be parameterized with neural networks using a similar strategy


WHAT IF I TOLD YOU
YOU CAN ENCODE AN ENTIRE SENTENCE IN A VECTOR