Natural Language Processing (CSE 517): Neural Language Models

Noah Smith

University of Washington
nasmith@cs.washington.edu

April 11, 2018

Quick Review

A language model is a probability distribution over $\mathcal{V}^\dagger$.

Typically p decomposes into probabilities $p(x_i \mid \boldsymbol{h}_i)$.

- n-gram: $\boldsymbol{h}_i$ is $(\mathsf{n} - 1)$ previous symbols
- Probabilities are estimated from data.

Today: neural language models

Feedforward Neural Network Language Model

(Bengio et al., 2003)

Define the n-gram probability as follows:

$$p(\cdot \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = n_{\boldsymbol{\nu}}\left(\langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{\mathsf{n}-1}} \rangle\right) =$$

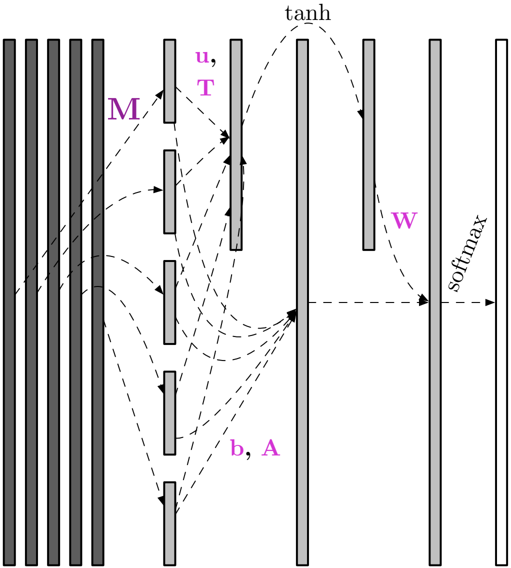
$$\mathrm{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{j=1}^{\mathsf{n}-1} \underset{V}{\mathbf{e}_{h_j}}^\top \underset{V \times d}{\mathbf{M}} \underset{d \times V}{\mathbf{A}_j} + \underset{V \times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{e}_{h_j}^\top \mathbf{M} \underset{d \times H}{\mathbf{T}_j} \right) \right)$$

where each $\mathbf{e}_{h_j} \in \mathbb{R}^V$ is a one-hot vector and H is the number of "hidden units" in the neural network (a "hyperparameter").

Parameters $\boldsymbol{\nu}$ include:

- $\mathbf{M} \in \mathbb{R}^{V \times d}$, which are called "embeddings" (row vectors), one for every word in $\mathcal{V}$
- Feedforward NN parameters $\mathbf{b} \in \mathbb{R}^V$, $\mathbf{A} \in \mathbb{R}^{(\mathsf{n}-1) \times d \times V}$, $\mathbf{W} \in \mathbb{R}^{V \times H}$, $\mathbf{u} \in \mathbb{R}^H$, $\mathbf{T} \in \mathbb{R}^{(\mathsf{n}-1) \times d \times H}$

Visualization



Why does it work?

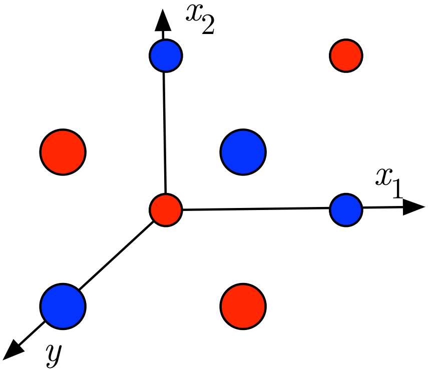
Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.

Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \mathrm{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2.

xor Example



Tuples where $y = \text{xor}(x_1, x_2)$ are red; tuples where $y \neq \text{xor}(x_1, x_2)$ are blue.

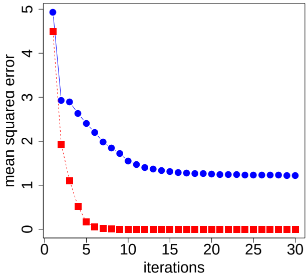
Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \mathrm{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2. But:

$$z = x_1 \cdot x_2$$
$$y = x_1 + x_2 - 2z$$

xor Example ($D = 13$)

Credit: Chris Dyer (`https://github.com/clab/cnn/blob/master/examples/xor.cc`)



$$\min_{\mathbf{v},a,\mathbf{W},\mathbf{b}} \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left(\text{xor}(x_1, x_2) - \underset{3}{\mathbf{v}}^\top \left(\underset{3\times 2}{\mathbf{W}} \underset{2}{\mathbf{x}} + \underset{3}{\mathbf{b}} \right) + a \right)^2$$

$$\min_{\mathbf{v},a,\mathbf{W},\mathbf{b}} \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \left(\text{xor}(x_1, x_2) - \underset{3}{\mathbf{v}}^\top \tanh \left(\underset{3\times 2}{\mathbf{W}} \underset{2}{\mathbf{x}} + \underset{3}{\mathbf{b}} \right) + a \right)^2$$

Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \mathrm{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2. But:

$$z = x_1 \cdot x_2$$
$$y = x_1 + x_2 - 2z$$

  - With high-dimensional inputs, there are a lot of conjunctive features to search through. For log-linear models, Della Pietra et al. (1997) did this, greedily.

Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \mathrm{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2. But:

$$z = x_1 \cdot x_2$$
$$y = x_1 + x_2 - 2z$$

  - With high-dimensional inputs, there are a lot of conjunctive features to search through. For log-linear models, Della Pietra et al. (1997) did this, greedily.
  - Neural models seem to smoothly explore lots of approximately-conjunctive features.

Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \text{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2. But:

$$z = x_1 \cdot x_2$$
$$y = x_1 + x_2 - 2z$$

  - With high-dimensional inputs, there are a lot of conjunctive features to search through. For log-linear models, Della Pietra et al. (1997) did this, greedily.
  - Neural models seem to smoothly explore lots of approximately-conjunctive features.
- Modern answer: representations of words and histories are tuned to the prediction problem.

Why does it work?

- Historical answer: multiple layers and nonlinearities allow feature *combinations* a linear model can't get.
  - Suppose we want $y = \mathrm{xor}(x_1, x_2)$; this can't be expressed as a linear function of x_1 and x_2. But:

$$
\begin{aligned}
z &= x_1 \cdot x_2 \\
y &= x_1 + x_2 - 2z
\end{aligned}
$$

  - With high-dimensional inputs, there are a lot of conjunctive features to search through. For log-linear models, Della Pietra et al. (1997) did this, greedily.
  - Neural models seem to smoothly explore lots of approximately-conjunctive features.
- Modern answer: representations of words and histories are tuned to the prediction problem.
- Word embeddings: a powerful idea . . .

Important Idea: Words as Vectors

The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models.
ou should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

Important Idea: Words as Vectors

The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models. You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

- Why?

Important Idea: Words as Vectors

The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models. You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

- Why?
- Deerwester et al. (1990) explored dimensionality reduction techniques for information retrieval-style querying of text collections.

Important Idea: Words as Vectors

The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models. You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

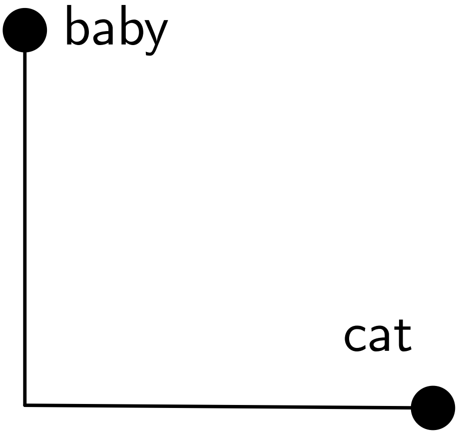
- Why?
- Deerwester et al. (1990) explored dimensionality reduction techniques for information retrieval-style querying of text collections.
- Considerable ongoing research on learning word representations to capture linguistic *similarity* (Turney and Pantel, 2010); this is known as **vector space semantics**.
  - Why "semantics"?

Important Idea: Words as Vectors

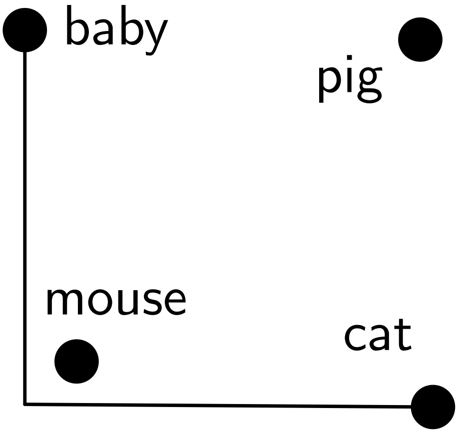
The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models. You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

- Why?
- Deerwester et al. (1990) explored dimensionality reduction techniques for information retrieval-style querying of text collections.
- Considerable ongoing research on learning word representations to capture linguistic *similarity* (Turney and Pantel, 2010); this is known as **vector space semantics**.
  - Why "semantics"?
- Something like this also turns up in traditional linguistic theories, e.g., marking nouns as "animate" or not.

Words as Vectors: Example



Words as Vectors: Example



Parameter Estimation

Bad news for neural language models:

- Log-likelihood function is not convex.
  - So any perplexity experiment is evaluating the model *and* an algorithm for estimating it.
- Calculating log-likelihood and its gradient is very expensive (5 epochs took 3 weeks on 40 CPUs).

Parameter Estimation

Bad news for neural language models:

- Log-likelihood function is not convex.
  - So any perplexity experiment is evaluating the model *and* an algorithm for estimating it.
- Calculating log-likelihood and its gradient is very expensive (5 epochs took 3 weeks on 40 CPUs).

Good news:

- $\nu_{\boldsymbol{\nu}}$ is differentiable with respect to $\mathbf{M}$ (from which its inputs come) and $\boldsymbol{\nu}$ (its parameters), so gradient-based methods are available.

Lots more details in Bengio et al. (2003) and (for NNs more generally) in Goldberg (2015).

What's Coming Up

- The log-bilinear language model
- Recurrent neural network language models

Log-Bilinear Language Model

(Mnih and Hinton, 2007)

Define the n-gram probability as follows, for each $v \in \mathcal{V}$:

$$p(v \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = \frac{\exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top}\right) \underbrace{\mathbf{m}_v}_{d} + c_v\right)}{\sum_{v' \in \mathcal{V}} \exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top}\right) \underbrace{\mathbf{m}_{v'}}_{d} + c_v\right)}$$

Log-Bilinear Language Model

(Mnih and Hinton, 2007)

Define the n-gram probability as follows, for each $v \in \mathcal{V}$:

$$p(v \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = \frac{\exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^\top \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^\top \right) \underbrace{\mathbf{m}_v}_{d} + c_v \right)}{\sum_{v' \in \mathcal{V}} \exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^\top \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^\top \right) \underbrace{\mathbf{m}_{v'}}_{d} + c_v \right)}$$

- Number of parameters: $D = \underbrace{Vd}_{\mathbf{M}} + \underbrace{(\mathsf{n}-1)d^2}_{\mathbf{A}} + \underbrace{d}_{\mathbf{b}} + \underbrace{V}_{\mathbf{c}}$

Log-Bilinear Language Model

(Mnih and Hinton, 2007)

Define the n-gram probability as follows, for each $v \in \mathcal{V}$:

$$p(v \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = \frac{\exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^\top \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^\top\right) \underbrace{\mathbf{m}_v}_{d} + c_v\right)}{\sum_{v' \in \mathcal{V}} \exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^\top \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^\top\right) \underbrace{\mathbf{m}_{v'}}_{d} + c_v\right)}$$

- Number of parameters: $D = \underbrace{Vd}_{\mathbf{M}} + \underbrace{(\mathsf{n}-1)d^2}_{\mathbf{A}} + \underbrace{d}_{\mathbf{b}} + \underbrace{V}_{\mathbf{c}}$
- The predicted word's probability depends on its vector $\mathbf{m}_v$, not just on the vectors of the history words.

Log-Bilinear Language Model

(Mnih and Hinton, 2007)

Define the n-gram probability as follows, for each $v \in \mathcal{V}$:

$$p(v \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = \frac{\exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top} \right) \underbrace{\mathbf{m}_v}_{d} + c_v \right)}{\sum_{v' \in \mathcal{V}} \exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top} \right) \underbrace{\mathbf{m}_{v'}}_{d} + c_v \right)}$$

- Number of parameters: $D = \underbrace{Vd}_{\mathbf{M}} + \underbrace{(\mathsf{n}-1)d^2}_{\mathbf{A}} + \underbrace{d}_{\mathbf{b}} + \underbrace{V}_{\mathbf{c}}$
- The predicted word's probability depends on its vector $\mathbf{m}_v$, not just on the vectors of the history words.
- Training this model involves a sum over the vocabulary (like log-linear models we saw last time).

Log-Bilinear Language Model

(Mnih and Hinton, 2007)

Define the n-gram probability as follows, for each $v \in \mathcal{V}$:

$$p(v \mid \langle h_1, \ldots, h_{\mathsf{n}-1} \rangle) = \frac{\exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top} \right) \underbrace{\mathbf{m}_v}_{d} + c_v \right)}{\sum_{v' \in \mathcal{V}} \exp\left(\sum_{j=1}^{\mathsf{n}-1} \left(\underbrace{\mathbf{m}_{h_j}}_{d}{}^{\top} \underbrace{\mathbf{A}_{j,*,*}}_{d \times d} + \underbrace{\mathbf{b}}_{d}{}^{\top} \right) \underbrace{\mathbf{m}_{v'}}_{d} + c_v \right)}$$

- Number of parameters: $D = \underbrace{Vd}_{\mathbf{M}} + \underbrace{(\mathsf{n}-1)d^2}_{\mathbf{A}} + \underbrace{d}_{\mathbf{b}} + \underbrace{V}_{\mathbf{c}}$
- The predicted word's probability depends on its vector $\mathbf{m}_v$, not just on the vectors of the history words.
- Training this model involves a sum over the vocabulary (like log-linear models we saw last time).
- Later work explored variations to make learning faster (related to class-based models we saw earlier).

Observations about Neural Language Models (So Far)

- There's no knowledge built in that the most recent word $h_{\mathsf{n}-1}$ should generally be more informative than earlier ones.
  - This has to be learned.
- In addition to choosing n, also have to choose dimensionalities like d and H.
- Parameters of these models are hard to interpret.
- Architectures are not intuitive.
- Still, impressive perplexity gains got people's interest.

Observations about Neural Language Models (So Far)

- There's no knowledge built in that the most recent word $h_{\mathsf{n}-1}$ should generally be more informative than earlier ones.
  - This has to be learned.
- In addition to choosing n, also have to choose dimensionalities like d and H.
- Parameters of these models are hard to interpret.
  - Example: ℓ_2-norm of $\mathbf{A}_{j,*,*}$ and $\mathbf{T}_{j,*,*}$ in the feedforward model correspond to the importance of history position j.
  - Individual word embeddings can be clustered and dimensions can be analyzed (e.g., Tsvetkov et al., 2015).
- Architectures are not intuitive.
- Still, impressive perplexity gains got people's interest.

Recurrent Neural Network

- Each input element is understood to be an element of a sequence: $\langle \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell \rangle$
- At each timestep t:
  - The tth input element $\mathbf{x}_t$ is processed alongside the previous state $\mathbf{s}_{t-1}$ to calculate the new **state** $(\mathbf{s}_t)$.
  - The tth output is a function of the state $\mathbf{s}_t$.
  - The *same functions* are applied at each iteration:

$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{x}_t, \mathbf{s}_{t-1})$$
$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t)$$

In RNN language models, words *and* histories are represented as vectors (respectively, $\mathbf{x}_t = \mathbf{e}_{x_t}$ and $\mathbf{s}_t$).

RNN Language Model

The original version, by Mikolov et al. (2010) used a "simple" RNN architecture along these lines:

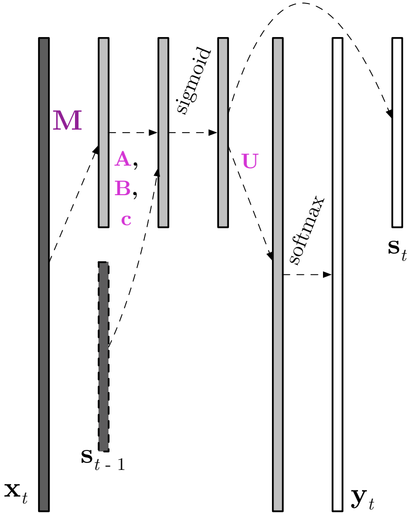
$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{s}_{t-1}) = \text{sigmoid}\left(\left(\mathbf{e}_{x_t}^\top \mathbf{M}\right)^\top \mathbf{A} + \mathbf{s}_{t-1}^\top \mathbf{B} + \mathbf{c}\right)$$

$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t) = \text{softmax}\left(\mathbf{s}_t^\top \mathbf{U}\right)$$

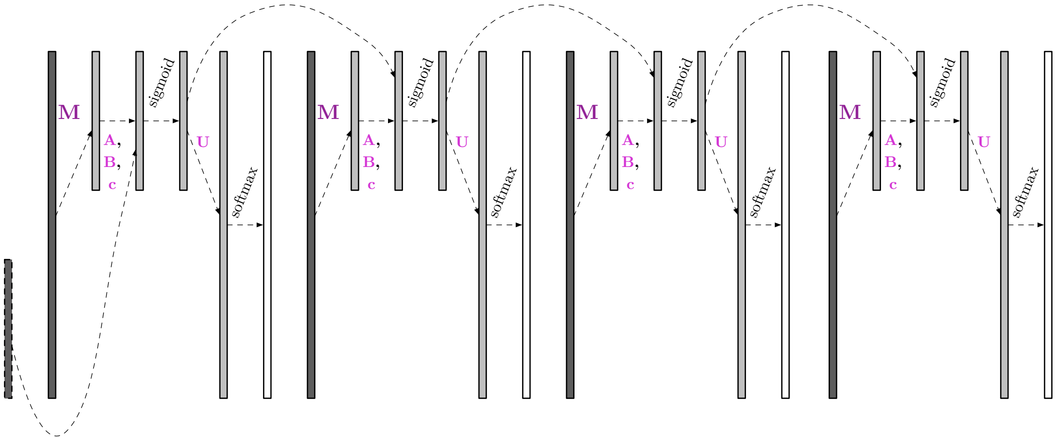
$$p(v \mid x_1, \ldots, x_{t-1}) = [\mathbf{y}_t]_v$$

Note: this is *not* an n-gram (Markov) model!

Visualization



Visualization



Improvements to RNN Language Models

The simple RNN is known to suffer from two related problems:

- "Vanishing gradients" during learning make it hard to propagate error into the distant past.
- State tends to change a lot on each iteration; the model "forgets" too much.

Some variants:

- "Stacking" these functions to make deeper networks.
- Sundermeyer et al. (2012) use "long short-term memories" (LSTMs; see Olah, 2015) and Cho et al. (2014) use "gated recurrent units" (GRUs) to define $f_{\text{recurrent}}$.
- Mikolov et al. (2014) engineer the linear transformation in the simple RNN for better preservation.
- Jozefowicz et al. (2015) used randomized search to find even better architectures.

Comparison: Probabilistic vs. Connectionist Modeling

	Probabilistic	**Connectionist**
What do we engineer?	features, assumptions	architectures
Theory?	as N gets large	not really
Interpretation of parameters?	often easy	usually hard

Parting Shots

Parting Shots

- I said very little about *estimating* the parameters.

Parting Shots

- I said very little about *estimating* the parameters.
  - At present, it's almost always stochastic gradient descent with heavy use of the chain rule from calculus ("backpropagation").

Parting Shots

- I said very little about *estimating* the parameters.
  - At present, it's almost always stochastic gradient descent with heavy use of the chain rule from calculus ("backpropagation").
  - New libraries to help you are coming out all the time.

Parting Shots

- I said very little about *estimating* the parameters.
  - At present, it's almost always stochastic gradient descent with heavy use of the chain rule from calculus ("backpropagation").
  - New libraries to help you are coming out all the time.
  - Many of them use GPUs to speed things up.

Parting Shots

- I said very little about *estimating* the parameters.
  - At present, it's almost always stochastic gradient descent with heavy use of the chain rule from calculus ("backpropagation").
  - New libraries to help you are coming out all the time.
  - Many of them use GPUs to speed things up.
- This progression is worth reflecting on:

	history:	represented as:
before 1996	$(n-1)$-gram	discrete
1996–2003		feature vector
2003–2010		embedded vector
since 2010	unrestricted	embedded

Parting Shots

- I said very little about *estimating* the parameters.
  - At present, it's almost always stochastic gradient descent with heavy use of the chain rule from calculus ("backpropagation").
  - New libraries to help you are coming out all the time.
  - Many of them use GPUs to speed things up.
- This progression is worth reflecting on:

	history:	represented as:
before 1996	$(n-1)$-gram	discrete
1996–2003		feature vector
2003–2010		embedded vector
since 2010	unrestricted	embedded

- Next, we'll let go of the text-as-sequence idea and think about probabilistic models relating a word and its **cotext** (textual context).

References I

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003. URL http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proc. of EMNLP*, 2014.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6): 391–407, 1990.

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

Yoav Goldberg. A primer on neural network models for natural language processing, 2015. URL http://u.cs.biu.ac.il/~yogo/nnlp.pdf.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proc. of ICML*, 2015. URL http://www.jmlr.org/proceedings/papers/v37/jozefowicz15.pdf.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. of Interspeech*, 2010. URL http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf.

References II

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks, 2014. arXiv:1412.7753.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proc. of ICML*, 2007.

Christopher Olah. Understanding LSTM networks, 2015. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Proc. of Interspeech*, 2012.

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. Evaluation of word vector representations by subspace alignment. In *Proc. of EMNLP*, 2015.

Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37(1):141–188, 2010. URL `https://www.jair.org/media/2934/live-2934-4846-jair.pdf`.