Statistical Learning Models for Text and Graph Data
Neural Language Models

Yangqiu Song

Hong Kong University of Science and Technology

yqsong@cse.ust.hk

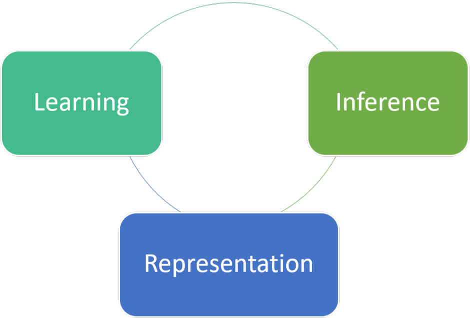
September 18, 2019

∗Contents are based on materials created by Noah Smith

Reference Content

- Noah Smith. CSE 517: Natural Language Processing
  https://courses.cs.washington.edu/courses/cse517/16wi/

Course Organization



- Representation: language models, word embeddings, topic models, knowledge graphs
- Learning: supervised learning, semi-supervised learning, distant supervision, indirect supervision, sequence models, deep learning, optimization techniques
- Inference: constraint modeling, joint inference, search algorithms

Overview

1. Neural Language Models
2. Extensions
3. Evaluation

Quick Review

- A language model is a probability distribution over $\mathcal{V}^{\dagger} = \mathcal{V} \cap \emptyset$
- Typically P decomposes into probabilities $P(x_i|\mathbf{h}_i)$. For n-gram language models, to reduce notation confusion, we set:
  - n-gram: $\mathbf{h}_i$ are $(n-1)$ previous symbols $\langle w_{i-1}, \ldots, w_{i-n+1} \rangle$; estimate by counting and normalizing (with smoothing)
  - log-linear: featurized representation of $\langle \mathbf{h}_i, x_i \rangle$; estimate iteratively by gradient descent
- Today: neural language models

Neural Network: Definitions

Warning: there is no widely accepted standard notation!

A feedforward neural network n_ν is defined by:

- function family that maps parameter values to functions of the form $\mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$; typically:
  - Non-linear
  - Differentiable with respect to its inputs
  - "Assembled" through a series of affine transformations and non-linearities, composed together
  - Symbolic/discrete inputs handled through lookups
- Parameter values ν
  - Typically a collection of scalars, vectors, and matrices
  - We often assume they are linearized into $\mathbb{R}^D$

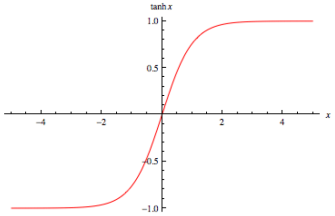
A Couple of Useful Functions

- Softmax: $\mathbb{R}^k \to \mathbb{R}^k$

$$\langle x_1, x_2, \ldots, x_k \rangle \mapsto \left\langle \frac{e^{x_1}}{\sum_{j=1}^k e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^k e^{x_j}}, \ldots, \frac{e^{x_k}}{\sum_{j=1}^k e^{x_j}} \right\rangle$$

- tanh: $\mathbb{R} \to [-1, 1]$

$$x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Generalized to be elementwise, so that it maps $\mathbb{R}^k \to [-1, 1]^k$

- Others include: ReLUs, logistic sigmoids, PReLUs, ...

"One Hot" Vectors

- Arbitrarily order the words in $\mathcal{V}$, giving each an index in $\{1, \ldots, V\}$
- Let $e_i \in \mathbb{R}^V$ contain all zeros, with the exception of a 1 in position i
- This is the "one hot" vector for the ith word in $\mathcal{V}$

$$\mathbf{e}_i = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Feedforward Neural Network Language Model

Bengio et al. (2003)

- Define the n-gram probability as follows:

$$P(\langle v_1, \ldots, v_V \rangle | \langle h_1, \ldots, h_{n-1} \rangle) = n_\nu(\langle v_1, \ldots, v_V \rangle | \langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{n-1}} \rangle) =$$

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

  where $\langle h_1, \ldots, h_{n-1} \rangle$ are $(n-1)$ previous symbols $\langle w_{i-1}, \ldots, w_{i-n+1} \rangle$, $\mathbf{e}_{h_i} \in \mathbb{R}^V$ is a one hot vector and H is the number of "hidden units" in the neural network (a "hyperparameter")
- Parameters in neural network n_ν
  - $\mathbf{M} \in \mathbb{R}^{V \times d}$: "embeddings" (row vectors), one for every word in $\mathcal{V}$
  - Forward NN parameters: $\mathbf{b} \in \mathbb{R}^V$, $\mathbf{A} \in \mathbb{R}^{(n-1) \times d \times V}$ ($\mathbf{A}_i \in \mathbb{R}^{d \times V}$), $\mathbf{W} \in \mathbb{R}^{V \times H}$, $\mathbf{u} \in \mathbb{R}^H$, $\mathbf{T} \in \mathbb{R}^{(n-1) \times d \times H}$ ($\mathbf{T}_i \in \mathbb{R}^{d \times H}$)

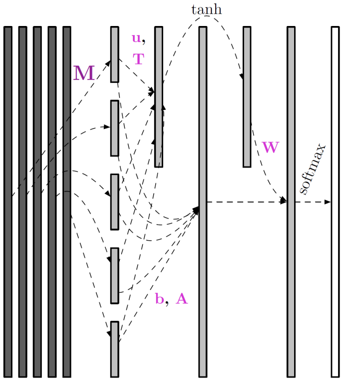
Breaking It Down

$$P(\langle v_1, \ldots, v_V \rangle | \langle h_1, \ldots, h_{n-1} \rangle) = n_\nu(\langle v_1, \ldots, v_V \rangle | \langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{n-1}} \rangle) =$$

$$\mathit{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}} \underset{d \times V}{\mathbf{A}_i} + \underset{V \times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}} \underset{d \times H}{\mathbf{T}_i} \right) \right)$$

- Look up each of the history words $h_j, \forall j \in \{1, \ldots, n-1\}$ in $\mathbf{M}$; keep two copies.

$$\underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}}$$



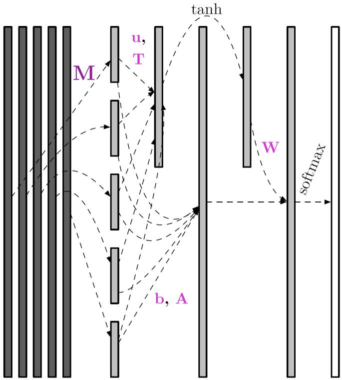
Breaking It Down

- Look up each of the history words $h_j, \forall j \in \{1, \ldots, n-1\}$ in $\mathbf{M}$; keep two copies.
- Rename them as m_{h_i}

$$\underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}} = \underset{d}{m_{h_i}}$$

$$P(\langle v_1, \ldots, v_V \rangle | \langle h_1, \ldots, h_{n-1} \rangle) = n_\nu(\langle v_1, \ldots, v_V \rangle | \langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{n-1}} \rangle) =$$

$$softmax\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V \times d}{\mathbf{M}} \underset{d \times V}{\mathbf{A}_i} + \underset{V \times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V \times d}{\mathbf{M}} \underset{d \times H}{\mathbf{T}_i} \right) \right)$$



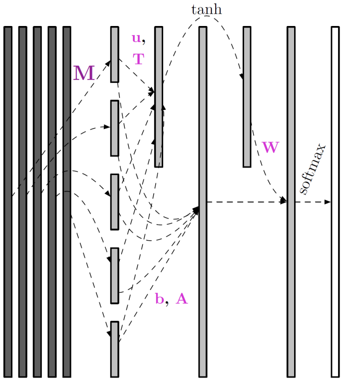
Breaking It Down

- Apply an affine transformation to the history-word embeddings ($\mathbf{u}$, $\mathbf{T}$)

$$\underset{V}{\mathbf{e}_{h_i}^\top} \underset{V\times d}{\mathbf{M}} = \underset{d}{m_{h_i}}$$

$$\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d\times H}{\mathbf{T}_i}$$

$$P(\langle v_1,\dots,v_V\rangle | \langle h_1,\dots,h_{n-1}\rangle) = n_\nu(\langle v_1,\dots,v_V\rangle | \langle \mathbf{e}_{h_1},\dots,\mathbf{e}_{h_{n-1}}\rangle) =$$
$$softmax\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V\times d}{\mathbf{M}} \underset{d\times V}{\mathbf{A}_i} + \underset{V\times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V\times d}{\mathbf{M}} \underset{d\times H}{\mathbf{T}_i}\right)\right)$$

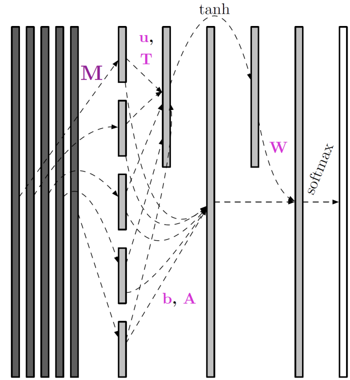


Breaking It Down

- Apply an affine transformation to the history-word embeddings ($\mathbf{u}$, $\mathbf{T}$)
- and a tanh nonlinearity

$$\underset{V}{\mathbf{e}_{h_i}^\top}\underset{V\times d}{\mathbf{M}} = \underset{d}{m_{h_i}}$$

$$\tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d\times H}{\mathbf{T}_i}\right)$$



Breaking It Down

- Apply an affine transformation to everything ($\mathbf{b}$, $\mathbf{A}$, $\mathbf{W}$)

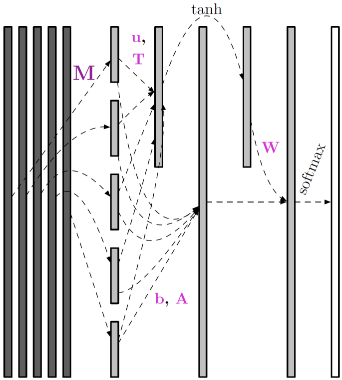
$$\underset{V}{\mathbf{e}_{h_i}^{\top}} \underset{V \times d}{\mathbf{M}} = \underset{d}{m_{h_i}}$$

$$\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d \times V}{\mathbf{A}_i} +$$

$$\underset{V \times H}{\mathbf{W}} \tanh \left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d \times H}{\mathbf{T}_i} \right)$$

$$P(\langle v_1, \ldots, v_V \rangle | \langle h_1, \ldots, h_{n-1} \rangle) = n_\nu(\langle v_1, \ldots, v_V \rangle | \langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{n-1}} \rangle) =$$

$$softmax\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V \times d}{\mathbf{M}} \underset{d \times V}{\mathbf{A}_i} + \underset{V \times H}{\mathbf{W}} \tanh \left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V \times d}{\mathbf{M}} \underset{d \times H}{\mathbf{T}_i} \right) \right)$$

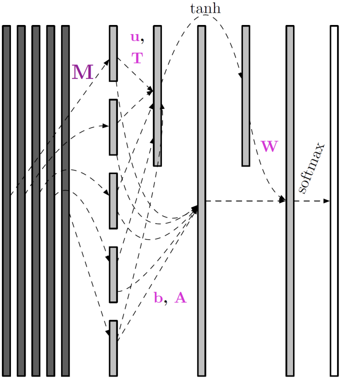


Breaking It Down

- Apply a softmax transformation to make the vector sum to one.

$$\text{softmax}\left(\underset{V}{\mathbf{b}} + \textstyle\sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d\times V}{\mathbf{A}_i} \right.$$
$$\left. + \underset{V\times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \textstyle\sum_{i=1}^{n-1} \underset{d}{m_{h_i}} \underset{d\times H}{\mathbf{T}_i}\right)\right)$$

$$P(\langle v_1,\ldots,v_V\rangle|\langle h_1,\ldots,h_{n-1}\rangle) = n_\nu(\langle v_1,\ldots,v_V\rangle|\langle \mathbf{e}_{h_1},\ldots,\mathbf{e}_{h_{n-1}}\rangle) =$$
$$\mathit{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V\times d}{\mathbf{M}} \underset{d\times V}{\mathbf{A}_i} + \underset{V\times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^T} \underset{V\times d}{\mathbf{M}} \underset{d\times H}{\mathbf{T}_i}\right)\right)$$



Breaking It Down

$$\text{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}} \underset{d \times V}{\mathbf{A}_i} + \underset{V \times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V \times d}{\mathbf{M}} \underset{d \times H}{\mathbf{T}_i}\right)\right)$$

- Like a log-linear language model with two kinds of features:
  - Concatenation of context-word embeddings vectors $(\mathbf{m}_{h_1}, \ldots, \mathbf{m}_{h_{n-1}})$, mapped by $\mathbf{A} = (\mathbf{A}_1^\top, \ldots, \mathbf{A}_{n-1}^\top)^\top)$
  - tanh-affine transformation of the above
- New parameters arise from (i) embeddings and (ii) affine transformation "inside" the nonlinearity.

Number of Parameters

$$\text{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V\times d}{\mathbf{M}} \underset{d\times V}{\mathbf{A}_i} + \underset{V\times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V\times d}{\mathbf{M}} \underset{d\times H}{\mathbf{T}_i}\right)\right)$$

$$D = \underbrace{Vd}_{\mathbf{M}} + \underbrace{V}_{\mathbf{b}} + \underbrace{(n-1)dV}_{\mathbf{A}} + \underbrace{VH}_{\mathbf{W}} + \underbrace{H}_{\mathbf{u}} + \underbrace{(n-1)dH}_{\mathbf{T}}$$

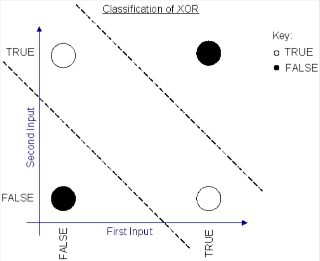
- For Bengio et al. (2003):
  - $V \approx 18,000$ (after OOV processing)
  - $d \in \{30, 60\}$
  - $H \in \{50, 100\}$
  - $n - 1 = 5$
- So $D = 461V + 30,100 = 8.3M$ parameters, compared to $O(V^n)$ for classical n-gram models
  - Forcing $\mathbf{A} = 0$ eliminated $300V$ parameters and performed a bit better, but was slower to converge
  - If we averaged m_{h_i} instead of concatenating, we'd get to $221V + 6,100$ (this is a variant of "continuous bag of words," Mikolov et al. (2013))

Why Does It Work?

- Historical answer: multiple layers and nonlinearities allow feature combinations a linear model can't get.
  - Suppose we want
    $y = xor(x_1, x_2) = x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2 = (x_1 + x_2) \cdot (\bar{x_1} + \bar{x_2})$

Example (xor)

x_1	x_2	$y = xor(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0



https://accu.org/index.php/journals/1915

  - this can't be expressed as a linear function of x_1 and x_2.

Why Does It Work?

- Historical answer: multiple layers and nonlinearities allow feature combinations a linear model can't get.
  - Suppose we want $y = xor(x_1, x_2) = x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2 = (x_1 + x_2) \cdot (\bar{x_1} + \bar{x_2})$
  - this can't be expressed as a linear function of x_1 and x_2, but
    - $z = x_1 \cdot x_2$
    - $y = x_1 + x_2 - 2z$

    (a non-linear function $z = x_1 \cdot x_2$ can help resolve it)

xor Example (D = 13)

https://github.com/clab/dynet/tree/master/examples/xor

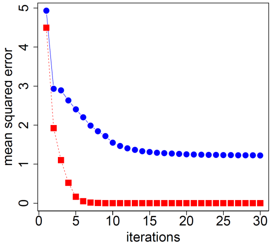
Regression of $y = xor(x_1, x_2)$: $\arg\min_{\boldsymbol{\theta}} (y - f(\mathbf{x}, \boldsymbol{\theta}))^2$

- Linear:

$$f(\mathbf{x}) = \underset{3}{\mathbf{v}}^\top (\underset{3\times 2}{\mathbf{W}} \underset{2}{\mathbf{x}} + \underset{3}{\mathbf{b}}) + \underset{1}{a}$$

- Non-linear:

$$f(\mathbf{x}) = \underset{3}{\mathbf{v}}^\top \tanh(\underset{3\times 2}{\mathbf{W}} \underset{2}{\mathbf{x}} + \underset{3}{\mathbf{b}}) + \underset{1}{a}$$



Why Does It Work?

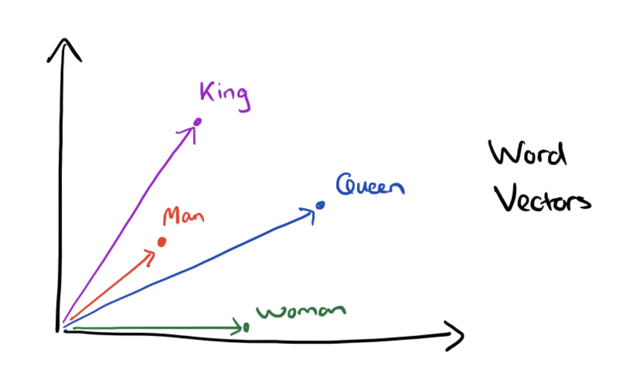
- Historical answer: multiple layers and nonlinearities allow feature combinations a linear model can't get.
  - Suppose we want $y = xor(x_1, x_2) = x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2 = (x_1 + x_2) \cdot (\bar{x_1} + \bar{x_2})$
  - This can't be expressed as a linear function of x_1 and x_2, 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, 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. You should think of this as a generalization of the discrete view of $\mathcal{V}$
- Deerwester et al. (1990) explored dimensionality reduction techniques for information retrieval-style querying of text collections
  - We will come back to this later
- Considerable ongoing research on learning word representations to capture linguistic similarity (Turney and Pantel (2010)); this is known as **vector space semantics**

Words as Vectors: Example

Example



https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Parameter Estimation

- Bad news for neural language models:
  - Log-likelihood function is not concave
  - Calculating log-likelihood and its gradient is very expensive (5 epochs took 3 weeks on 40 CPUs, in Bengio et al. (2003))
- Good news:
  - n_ν is differentiable with respect to $\mathbf{M}$ (from which its inputs come) and ν (its parameters), so gradient-based methods are available
  - Essential: the chain rule from calculus (sometimes called "backpropagation")
  - Lots more details in Bengio et al. (2003) and (for NNs more generally) in Goldberg (2016)

Overview

Next Up

- More examples of neural language models (in brief):
  - The log-bilinear language model
  - Recurrent neural network language models

Log-Bilinear Language Model

Mnih and Hinton (2007)

- In neural language model developed by Bengio et al. (2003):

$$P(\langle v_1, \ldots, v_V\rangle | \langle h_1, \ldots, h_{n-1}\rangle) = n_\nu(\langle v_1, \ldots, v_V\rangle | \langle \mathbf{e}_{h_1}, \ldots, \mathbf{e}_{h_{n-1}}\rangle) =$$

$$\text{softmax}\left(\underset{V}{\mathbf{b}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V\times d}{\mathbf{M}} \underset{d\times V}{\mathbf{A}_i} + \underset{V\times H}{\mathbf{W}} \tanh\left(\underset{H}{\mathbf{u}} + \sum_{i=1}^{n-1} \underset{V}{\mathbf{e}_{h_i}^\top} \underset{V\times d}{\mathbf{M}} \underset{d\times H}{\mathbf{T}_i}\right)\right)$$

- we haven't considered the current word

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

$$P(v|\langle h_1, \ldots, h_{n-1}\rangle) = \frac{\exp\left(\sum_{i=1}^{n-1}\left(\underset{d}{\mathbf{m}_{h_i}^\top}\underset{d\times d}{\mathbf{A}} + \underset{d}{\mathbf{b}}\right)^\top \underset{d}{\mathbf{m}_v} + c_v\right)}{\sum_{v'\in\mathcal{V}}\exp\left(\sum_{i=1}^{n-1}\left(\underset{d}{\mathbf{m}_{h_i}^\top}\underset{d\times d}{\mathbf{A}} + \underset{d}{\mathbf{b}}\right)^\top \underset{d}{\mathbf{m}_{v'}} + c_{v'}\right)}$$

- Number of parameters: $\underbrace{Vd}_{\mathbf{M}} + \underbrace{(n-1)d^2}_{\mathbf{A}} + \underbrace{d}_{\mathbf{b}} + \underbrace{V}_{\mathbf{c}}$
- 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 earlier)
- Later work explored variations to make learning faster

Observations about Neural Language Models (So Far)

- There's no knowledge built in that the most recent word h_{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 $\mathbf{A}_i$ and $\mathbf{T}_i$ in the feedforward model correspond to the importance of history position i
    - 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: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l\}$
- 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, $\mathbf{x}_t = \mathbf{e}_{h_t}$ and $\mathbf{s}_t$).

RNN Language Model

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

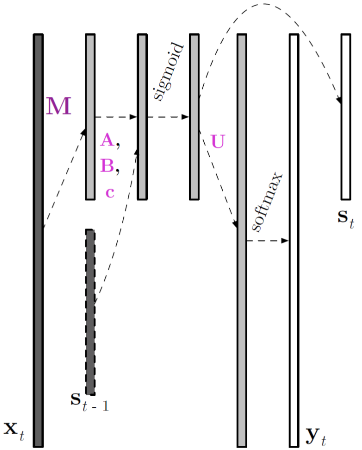
$$\mathbf{s}_t = f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{s}_{t-1}) =$$

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

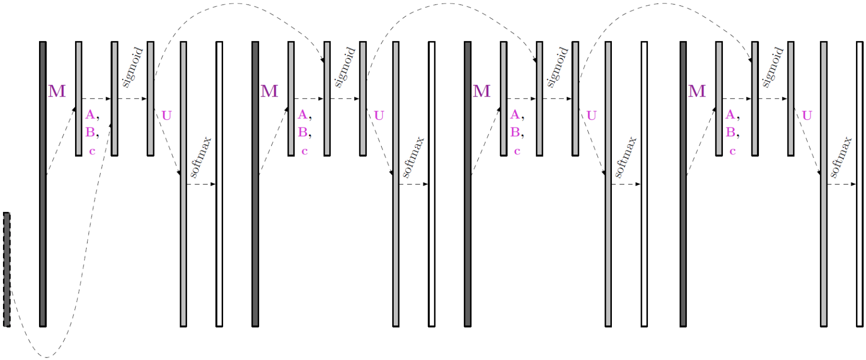
$$\mathbf{y}_t = f_{\text{output}}(\mathbf{s}_t) = \text{softmax}(\mathbf{s}_t^\top \mathbf{U})$$

$$P(v|h_1, \ldots, h_{n-1}) = [\mathbf{y}_t]_v$$

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



RNN Model 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) 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
  - Józefowicz 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

- I said very little about estimating the parameters
  - At present, this requires a lot of engineering
  - 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	(n-1)-gram	feature vector
2003–2010	(n-1)-gram	embedded vector
since 2010	unrestricted	embedded vector

Overview

Neural Language Model Results (Bengio et al. (2003))

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319
class-based back-off	3	2000						343	326
class-based back-off	4	500						327	312
class-based back-off	5	500						327	312

Table 1: Comparative results on the Brown corpus. The deleted interpolation trigram has a test perplexity that is 33% above that of the neural network with the lowest validation perplexity. The difference is 24% in the case of the best n-gram (a class-based model with 500 word classes). n : order of the model. c : number of word classes in class-based n-grams. h : number of hidden units. m : number of word features for MLPs, number of classes for class-based n-grams. *direct*: whether there are direct connections from word features to outputs. *mix*: whether the output probabilities of the neural network are mixed with the

Recent Results (Merity et al. (2018))

Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M‡	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	–	125.7
Mikolov & Zweig (2012) - RNN	6M‡	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	–	92.0
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal & Ghahramani (2016) - Variational LSTM	20M	–	78.6
Gal & Ghahramani (2016) - Variational LSTM	66M	–	73.4
Kim et al. (2016) - CharCNN	19M	–	78.9
Merity et al. (2016) - Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) - LSTM	–	–	82.3
Grave et al. (2016) - LSTM + continuous cache pointer	–	–	72.1
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	24M	75.7	73.2
Inan et al. (2016) - Variational LSTM (tied) + augmented loss	51M	71.1	68.5
Zilly et al. (2016) - Variational RHN (tied)	23M	67.9	65.4
Zoph & Le (2016) - NAS Cell (tied)	25M	–	64.0
Zoph & Le (2016) - NAS Cell (tied)	54M	–	62.4
Melis et al. (2017) - 4-layer skip connection LSTM (tied)	24M	60.9	58.3
AWD-LSTM - 3-layer LSTM (tied)	24M	60.0	57.3
AWD-LSTM - 3-layer LSTM (tied) + continuous cache pointer	24M	53.9	52.8

Table 1: Single model perplexity on validation and test sets for the Penn Treebank language modeling task. Parameter numbers with ‡ are estimates based upon our understanding of the model and with reference to (Merity et al., 2016). Models noting *tied* use weight tying on the embedding and softmax weights. Our model, AWD-LSTM, stands for AvSGD Weight-Dropped LSTM.

Further Reading

- Goldberg (2016). A primer on neural network models for natural language processing.

References I

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Cho, K., van Merrienboer, B., Gülccehre, **c**C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734.

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

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.*, 57:345–420.

Józefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *ICML*, pages 2342–2350.

Merity, S., Keskar, N. S., and Socher, R. (2018). Regularizing and optimizing LSTM language models. In *ICLR*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *ICLR*.

References II

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., and Ranzato, M. (2014). Learning longer memory in recurrent neural networks. *CoRR*, abs/1412.7753.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Mnih, A. and Hinton, G. E. (2007). Three new graphical models for statistical language modelling. In *ICML*, pages 641–648.

Pietra, S. D., Pietra, V. J. D., and Lafferty, J. D. (1997). Inducing features of random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(4):380–393.

Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *INTERSPEECH*, pages 194–197.

Tsvetkov, Y., Faruqui, M., Ling, W., Lample, G., and Dyer, C. (2015). Evaluation of word vector representations by subspace alignment. In *EMNLP*, pages 2049–2054.

Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *J. Artif. Intell. Res.*, 37:141–188.