Natural Language Processing (CSEP 517): Language Models, Continued

Noah Smith

University of Washington
nasmith@cs.washington.edu

April 3, 2017

To-Do List

- Online quiz: due Sunday
- Print, sign, and return the academic integrity statement (if you haven't already)
- Read: Smith (2017);
  optionally, Jurafsky and Martin (2016), Collins (2011) §2, and Goldberg (2015) §0–4, 10–13 if you want to know more about neural networks
- A1 now due April 9 (Sunday)
- Late policy: four late days

Language Models: Definitions

- $\mathcal{V}$ is a finite set of (discrete) symbols (☺ "words" or possibly characters); $V = |\mathcal{V}|$
- $\mathcal{V}^\dagger$ is the (infinite) set of sequences of symbols from $\mathcal{V}$ whose final symbol is $\bigcirc$
- $p : \mathcal{V}^\dagger \rightarrow \mathbb{R}$, such that:
  - For any $\boldsymbol{x} \in \mathcal{V}^\dagger$, $p(\boldsymbol{x}) \geq 0$
  - $\displaystyle\sum_{\boldsymbol{x} \in \mathcal{V}^\dagger} p(\boldsymbol{X} = \boldsymbol{x}) = 1$

  (I.e., p is a proper probability distribution.)

Language modeling: estimate p from examples, $\boldsymbol{x}_{1:n} = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \rangle$.
Evaluation on test data $\bar{\boldsymbol{x}}_{1:m}$: perplexity, $2^{-\frac{1}{M}\sum_{i=1}^{m} \log_2 p(\bar{\boldsymbol{x}}_i)}$

Log-Linear Models: Definitions

We define a conditional log-linear model $p(Y \mid X)$ as:

- $\mathcal{Y}$ is the set of events/outputs (☺ for language modeling, $\mathcal{V}$)
- $\mathcal{X}$ is the set of contexts/inputs (☺ for n-gram language modeling, $\mathcal{V}^{\mathsf{n}-1}$)
- $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a feature vector function
- $\mathbf{w} \in \mathbb{R}^d$ are the model parameters

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\displaystyle\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y')}$$

Breaking It Down

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}$$

Breaking It Down

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}$$

linear score $\quad \mathbf{w} \cdot \boldsymbol{\phi}(x, y)$

Breaking It Down

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\displaystyle\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}$$

linear score	$\mathbf{w} \cdot \boldsymbol{\phi}(x, y)$
nonnegative	$\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)$

Breaking It Down

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}$$

linear score	$\mathbf{w} \cdot \boldsymbol{\phi}(x, y)$
nonnegative	$\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)$
normalizer	$\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y') = Z_{\mathbf{w}}(x)$

Breaking It Down

$$p_{\mathbf{w}}(Y = y \mid X = x) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)}$$

linear score	$\mathbf{w} \cdot \boldsymbol{\phi}(x, y)$
nonnegative	$\exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y)$
normalizer	$\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(x, y') = Z_{\mathbf{w}}(x)$

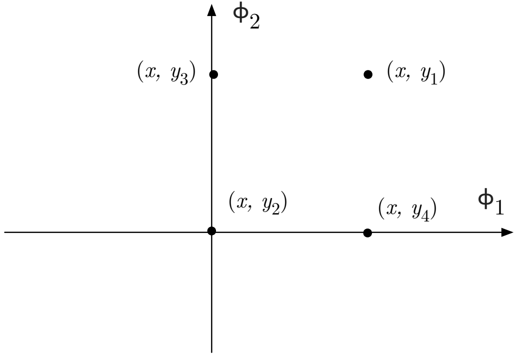
"Log-linear" comes from the fact that:

$$\log p_{\mathbf{w}}(Y = y \mid X = x) = \mathbf{w} \cdot \boldsymbol{\phi}(x, y) - \underbrace{\log Z_{\mathbf{w}}(x)}_{\text{constant in } y}$$

This is an instance of the family of **generalized linear models**.

The Geometric View

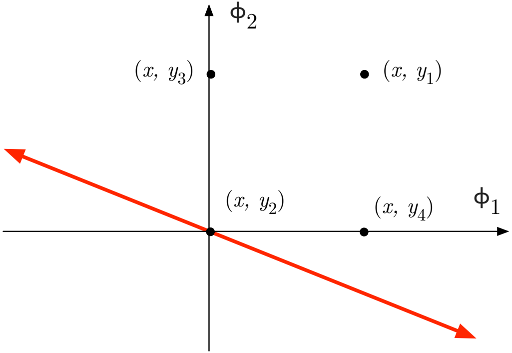
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



As a simple example, let the two features be binary functions.

The Geometric View

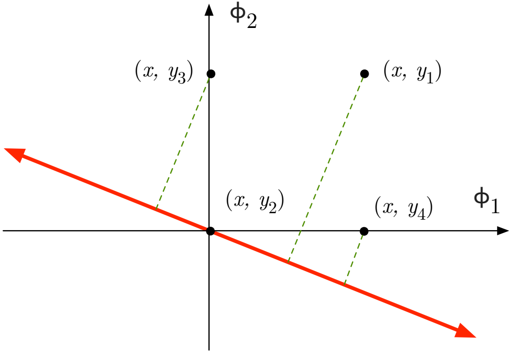
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



$$\mathbf{w} \cdot \boldsymbol{\phi} = w_1\phi_1 + w_2\phi_2 = 0$$

The Geometric View

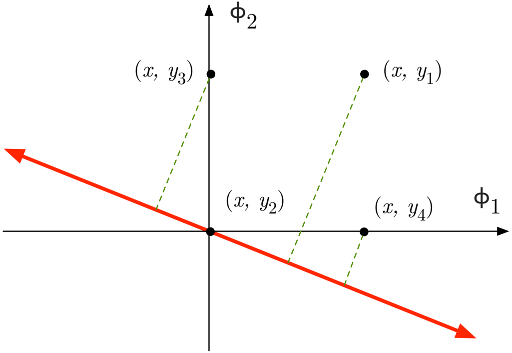
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



$$\text{distance}(\mathbf{w} \cdot \boldsymbol{\phi} = 0, \boldsymbol{\phi}_0) = \frac{|\mathbf{w} \cdot \boldsymbol{\phi}_0|}{\|\mathbf{w}\|_2} \propto |\mathbf{w} \cdot \boldsymbol{\phi}_0|$$

The Geometric View

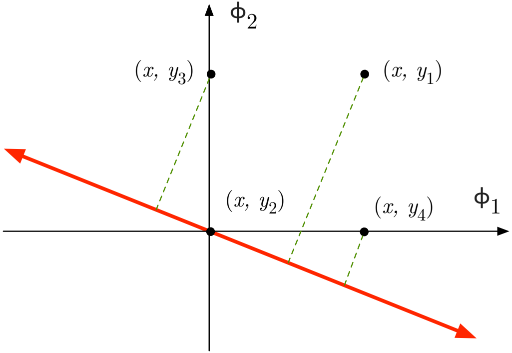
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



$$\mathbf{w} \cdot \boldsymbol{\phi}(x, y_1) > \mathbf{w} \cdot \boldsymbol{\phi}(x, y_3) > \mathbf{w} \cdot \boldsymbol{\phi}(x, y_4) > 0 \geq \mathbf{w} \cdot \boldsymbol{\phi}(x, y_2)$$

The Geometric View

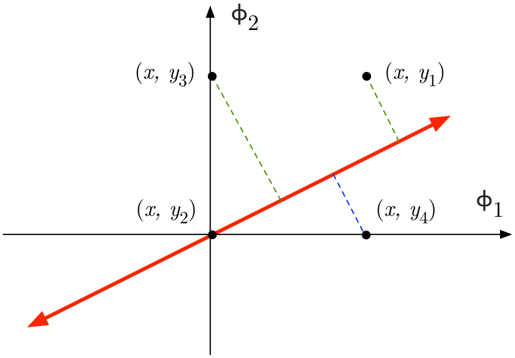
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



$$p_{\mathbf{w}}(y_1 \mid x) > p_{\mathbf{w}}(y_3 \mid x) > p_{\mathbf{w}}(y_4 \mid x) > p_{\mathbf{w}}(y_2 \mid x)$$

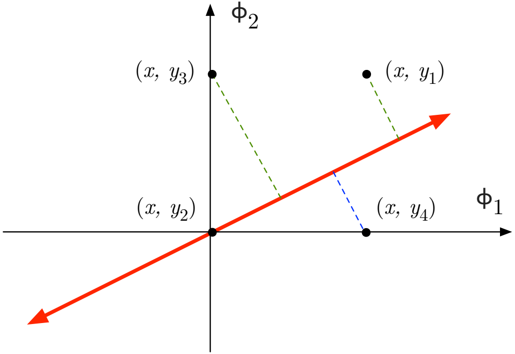
The Geometric View

Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



The Geometric View

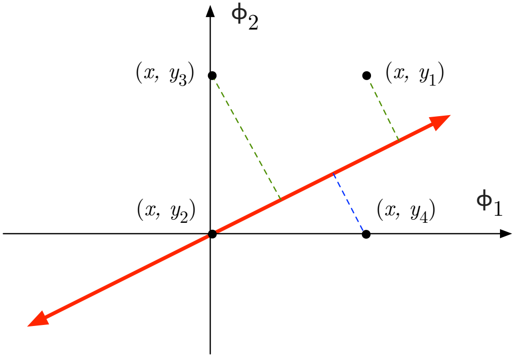
Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



$$p_{\mathbf{w}}(y_3 \mid x) > p_{\mathbf{w}}(y_1 \mid x) > p_{\mathbf{w}}(y_2 \mid x) > p_{\mathbf{w}}(y_4 \mid x)$$

The Geometric View

Suppose we have instance x, $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$, and there are only two features, ϕ_1 and ϕ_2.



Log-linear parameter estimation tries to choose $\mathbf{w}$ so that $p_{\mathbf{w}}(Y \mid x)$ matches the empirical distribution, $\frac{c(x,Y)}{c(x)}$.

Why Build Language Models This Way?

- Exploit **features** of histories for sharing of statistical strength and better smoothing (Lau et al., 1993)
- Condition the whole text on more interesting variables like the gender, age, or political affiliation of the author (Eisenstein et al., 2011)
- Interpretability!
  - Each feature ϕ_k controls a factor to the probability (e^{w_k}).
  - If $w_k < 0$ then ϕ_k makes the event less likely by a factor of $\frac{1}{e^{w_k}}$.
  - If $w_k > 0$ then ϕ_k makes the event more likely by a factor of e^{w_k}.
  - If $w_k = 0$ then ϕ_k has no effect.

Log-Linear n-Gram Models

$$
\begin{aligned}
p_{\mathbf{w}}(\boldsymbol{X} = \boldsymbol{x}) &= \prod_{j=1}^{\ell} p_{\mathbf{w}}(X_j = x_j \mid X_{0:j-1} = x_{0:j-1}) \\
&= \prod_{j=1}^{\ell} \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x_{0:j-1}, x_j)}{Z_{\mathbf{w}}(x_{0:j-1})} \\
&\overset{\text{assumption}}{=} \prod_{j-1}^{\ell} \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(x_{j-\mathsf{n}+1:j-1}, x_j)}{Z_{\mathbf{w}}(x_{j-\mathsf{n}+1:j-1})} \\
&= \prod_{j=1}^{\ell} \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_j, x_j)}{Z_{\mathbf{w}}(\boldsymbol{h}_j)}
\end{aligned}
$$

Example

The man who knew too

much
many
little
few
$\vdots$
hippopotamus

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"
- Spelling features: "X_j's first character is capitalized"

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"
- Spelling features: "X_j's first character is capitalized"
- Class features: "X_j is a member of class 132"

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"
- Spelling features: "X_j's first character is capitalized"
- Class features: "X_j is a member of class 132"
- Gazetteer features: "X_j is listed as a geographic place name"

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"
- Spelling features: "X_j's first character is capitalized"
- Class features: "X_j is a member of class 132"
- Gazetteer features: "X_j is listed as a geographic place name"

You can define any features you want!

- Too many features, and your model will overfit ☹
- Too few (good) features, and your model will not learn ☹

What Features in $\phi(X_{j-\mathsf{n}+1:j-1}, X_j)$?

- Traditional n-gram features: "$X_{j-1} = \text{the} \wedge X_j = \text{man}$"
- "Gappy" n-grams: "$X_{j-2} = \text{the} \wedge X_j = \text{man}$"
- Spelling features: "X_j's first character is capitalized"
- Class features: "X_j is a member of class 132"
- Gazetteer features: "X_j is listed as a geographic place name"

You can define any features you want!

- Too many features, and your model will overfit ☹
  - "Feature selection" methods, e.g., ignoring features with very low counts, can help.
- Too few (good) features, and your model will not learn ☹

“Feature Engineering”

- Many advances in NLP (not just language modeling) have come from careful design of features.

“Feature Engineering”

- Many advances in NLP (not just language modeling) have come from careful design of features.
- Sometimes “feature engineering” is used pejoratively.

“Feature Engineering”

- Many advances in NLP (not just language modeling) have come from careful design of features.
- Sometimes “feature engineering” is used pejoratively.
  - Some people would rather not spend their time on it!

"Feature Engineering"

- Many advances in NLP (not just language modeling) have come from careful design of features.
- Sometimes "feature engineering" is used pejoratively.
  - Some people would rather not spend their time on it!
- There is some work on automatically inducing features (Della Pietra et al., 1997).

“Feature Engineering”

- Many advances in NLP (not just language modeling) have come from careful design of features.
- Sometimes “feature engineering” is used pejoratively.
  - Some people would rather not spend their time on it!
- There is some work on automatically inducing features (Della Pietra et al., 1997).
- More recent work in neural networks can be seen as *discovering* features (instead of engineering them).

“Feature Engineering”

- Many advances in NLP (not just language modeling) have come from careful design of features.
- Sometimes “feature engineering” is used pejoratively.
  - Some people would rather not spend their time on it!
- There is some work on automatically inducing features (Della Pietra et al., 1997).
- More recent work in neural networks can be seen as *discovering* features (instead of engineering them).
- But in much of NLP, there’s a strong preference for *interpretable* features.

How to Estimate $\mathbf{w}$?

	n-gram	log-linear n-gram
$p_{\boldsymbol{\theta}}(\boldsymbol{x}) =$	$\displaystyle\prod_{j=1}^{\ell} \theta_{x_j \mid \boldsymbol{h}_j}$	$\displaystyle\prod_{j-1}^{\ell} \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_j, x_j)}{Z_{\mathbf{w}}(\boldsymbol{h}_j)}$
Parameters:	$\theta_{v \mid \boldsymbol{h}}$ $\forall v \in \mathcal{V}, \boldsymbol{h} \in (\mathcal{V} \cup \{\bigcirc\})^{\mathsf{n}-1}$	w_k $\forall k \in \{1, \ldots, d\}$
MLE:	$\hat{\theta}_{v \mid \boldsymbol{h}} = \dfrac{c(\boldsymbol{h}v)}{c(\boldsymbol{h})}$	no closed form

MLE for $\mathbf{w}$

- Let training data consist of $\{(\boldsymbol{h}_i, x_i)\}_{i=1}^{N}$.

MLE for $\mathbf{w}$

- Let training data consist of $\{(\boldsymbol{h}_i, x_i)\}_{i=1}^{N}$.
- Maximum likelihood estimation is:

$$
\begin{aligned}
& \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log p_{\mathbf{w}}(x_i \mid \boldsymbol{h}_i) \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i)}{Z_{\mathbf{w}}(\boldsymbol{h}_i)} \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \underbrace{\sum_{v\in\mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v)}_{Z_{\mathbf{w}}(\boldsymbol{h}_i)}
\end{aligned}
$$

MLE for $\mathbf{w}$

- Let training data consist of $\{(\boldsymbol{h}_i, x_i)\}_{i=1}^{N}$.
- Maximum likelihood estimation is:

$$
\begin{aligned}
&\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log p_{\mathbf{w}}(x_i \mid \boldsymbol{h}_i) \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log \frac{\exp \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i)}{Z_{\mathbf{w}}(\boldsymbol{h}_i)} \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \underbrace{\sum_{v\in\mathcal{V}} \exp \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, v)}_{Z_{\mathbf{w}}(\boldsymbol{h}_i)}
\end{aligned}
$$

- This is *concave* in $\mathbf{w}$.

MLE for $\mathbf{w}$

- Let training data consist of $\{(\boldsymbol{h}_i, x_i)\}_{i=1}^{N}$.
- Maximum likelihood estimation is:

$$\begin{aligned}
&\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log p_{\mathbf{w}}(x_i \mid \boldsymbol{h}_i) \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \log \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i)}{Z_{\mathbf{w}}(\boldsymbol{h}_i)} \\
&= \max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \underbrace{\sum_{v\in\mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v)}_{Z_{\mathbf{w}}(\boldsymbol{h}_i)}
\end{aligned}$$

- This is *concave* in $\mathbf{w}$.
- $Z_{\mathbf{w}}(\boldsymbol{h}_i)$ involves a sum over V terms.

MLE for $\mathbf{w}$

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \underbrace{\mathbf{w} \cdot \phi(\boldsymbol{h}_i, x_i) - \log Z_{\mathbf{w}}(\boldsymbol{h}_i)}_{f_i(\mathbf{w})}$$

MLE for $\mathbf{w}$

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \underbrace{\mathbf{w}\cdot\phi(\boldsymbol{h}_i, x_i) - \log Z_{\mathbf{w}}(\boldsymbol{h}_i)}_{f_i(\mathbf{w})}$$

Hope/fear view: for each instance i,

- increase the score of the correct output x_i, $score(x_i) = \mathbf{w}\cdot\phi(\boldsymbol{h}_i, x_i)$
- decrease the "softened max" score overall, $\log \sum_{v\in\mathcal{V}} \exp score(v)$

MLE for $\mathbf{w}$

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \underbrace{\mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log Z_{\mathbf{w}}(\boldsymbol{h}_i)}_{f_i(\mathbf{w})}$$

Gradient view:

$$\nabla_{\mathbf{w}} f_i = \underbrace{\boldsymbol{\phi}(\boldsymbol{h}_i, x_i)}_{\text{observed features}} - \underbrace{\sum_{v\in\mathcal{V}} p_{\mathbf{w}}(v \mid \boldsymbol{h}_i)\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, v)}_{\text{expected features}}$$

Setting this to zero means getting model's expectations to match empirical observations.

MLE for $\mathbf{w}$: Algorithms

- Batch methods (L-BFGS is popular)
- Stochastic gradient ascent/descent more common today, especially with special tricks for adapting the step size over time
- Many specialized methods (e.g., "iterative scaling")

Stochastic Gradient Descent

Goal: minimize $\sum_{i=1}^{N} f_i(\mathbf{w})$ with respect to $\mathbf{w}$.

Input: initial value $\mathbf{w}$, number of epochs T, learning rate α

For $t \in \{1, \ldots, T\}$:

- Choose a random permutation π of $\{1, \ldots, N\}$.
- For $i \in \{1, \ldots, N\}$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} f_{\pi(i)}$$

Output: $\mathbf{w}$

Avoiding Overfitting

Maximum likelihood estimation:

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log Z_{\mathbf{w}}(\boldsymbol{h}_i)$$

- If $\phi_j(\boldsymbol{h}, x)$ is (almost) always positive, we can always increase the objective (a little bit) by increasing w_j toward $+\infty$.

Avoiding Overfitting

Maximum likelihood estimation:

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log Z_{\mathbf{w}}(\boldsymbol{h}_i)$$

- If $\phi_j(\boldsymbol{h}, x)$ is (almost) always positive, we can always increase the objective (a little bit) by increasing w_j toward $+\infty$.

Standard solution is to add a regularization term:

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v\in\mathcal{V}} \exp \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda\|\mathbf{w}\|_p^p$$

where $\lambda > 0$ is a hyperparameter and $p = 2$ or 1.

MLE for $\mathbf{w}$

If we had more time, we'd study this problem more carefully!

Here's what you must remember:

- There is no closed form; you must use a numerical optimization algorithm like stochastic gradient descent.
- Log-linear models are powerful but expensive ($Z_{\mathbf{w}}(\boldsymbol{h}_i)$).
- Regularization is very important; we don't actually do MLE.
  - Just like for n-gram models! Only even more so, since log-linear models are even more expressive.

Quick Recap

Two kinds of language models so far:

	representation?	estimation?	think about?
n-gram	$\boldsymbol{h}_i$ is $(\text{n} - 1)$ previous symbols	count and normalize	smoothing
log-linear	featurized representation of $\langle \boldsymbol{h}_i, x_i \rangle$	iterative gradient descent	features

Neural Network: Definitions

Warning: there is no widely accepted standard notation!

A feedforward neural network $n_{\boldsymbol{\nu}}$ is defined by:

- A function family that maps parameter values to functions of the form $n : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$; 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 $\boldsymbol{\nu}$
  - Typically a collection of scalars, vectors, and matrices
  - We often assume they are linearized into $\mathbb{R}^D$

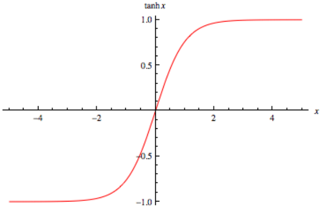
A Couple of Useful Functions

- $\text{softmax} : \mathbb{R}^k \rightarrow \mathbb{R}^k$

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

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

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



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

- 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 $\mathbf{e}_i \in \mathbb{R}^V$ contain all zeros, with the exception of a 1 in position i.

This is the "one hot" vector for the ith word in $\mathcal{V}$.

Feedforward Neural Network Language Model

(Bengio et al., 2003)

Define the n-gram probability as follows:

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

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

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 $\mathbf{b} \in \mathbb{R}^V$, $\mathbf{A} \in \mathbb{R}^{(\mathsf{n}-1) \times d \times V}$, $\mathbf{W} \in \mathbb{R}^{V \times H}$, $\mathbf{u} \in \mathbb{R}^H$, $\mathbf{T} \in \mathbb{R}^{(\mathsf{n}-1) \times d \times H}$

Breaking It Down

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

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

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

Breaking It Down

Look up each of the history words $h_j, \forall j \in \{1, \ldots, \mathsf{n} - 1\}$ in $\mathbf{M}$; keep two copies.
Rename the embedding for h_j as $\mathbf{m}_{h_j}$.

$$\mathbf{e}_{h_j}{}^\top \mathbf{M} = \mathbf{m}_{h_j}$$

$$\mathbf{e}_{h_j}{}^\top \mathbf{M} = \mathbf{m}_{h_j}$$

Breaking It Down

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

$$\mathbf{m}_{h_j}$$

$$\underset{H}{\mathbf{u}} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \underset{d\,\times\,H}{\mathbf{T}_j}$$

Breaking It Down

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

$$\mathbf{m}_{h_j}$$

$$\tanh\left(\mathbf{u} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j}\mathbf{T}_j\right)$$

Breaking It Down

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

$$\underset{V}{\mathbf{b}} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \underset{d \times V}{\mathbf{A}_j}$$

$$+ \underset{V \times H}{\mathbf{W}} \tanh\left(\mathbf{u} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \mathbf{T}_j \right)$$

Breaking It Down

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

$$\text{softmax}\left(\mathbf{b} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j}\ \mathbf{A}_j \right.$$
$$\left. + \mathbf{W}\ \tanh\left(\mathbf{u} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j}\ \mathbf{T}_j \right) \right)$$

Breaking It Down

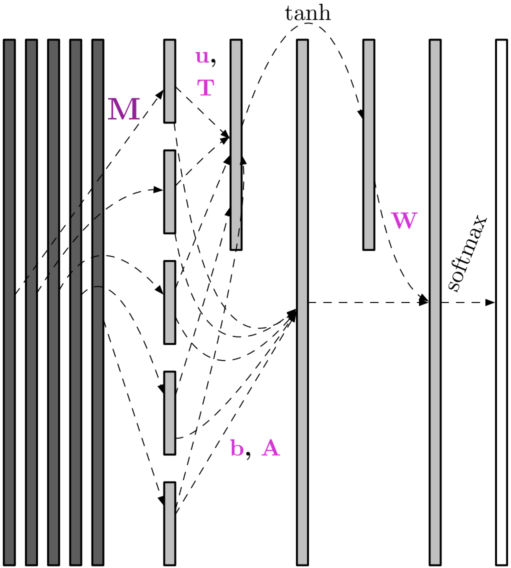
$$\text{softmax}\left(\mathbf{b} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \; \mathbf{A}_j \right.$$
$$\left. + \mathbf{W} \; \tanh\left(\mathbf{u} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \; \mathbf{T}_j \right) \right)$$

Like a log-linear language model with two kinds of features:

- Concatenation of context-word embeddings vectors $\mathbf{m}_{h_j}$
- $\tanh$-affine transformation of the above

New parameters arise from (i) embeddings and (ii) affine transformation "inside" the nonlinearity.

Visualization



Number of Parameters

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

For Bengio et al. (2003):

- $V \approx 18000$ (after OOV processing)
- $d \in \{30, 60\}$
- $H \in \{50, 100\}$
- $\mathsf{n} - 1 = 5$

So $D = 461V + 30100$ parameters, compared to $O(V^{\mathsf{n}})$ for classical n-gram models.

- Forcing $\mathbf{A} = \mathbf{0}$ eliminated $300V$ parameters and performed a bit better, but was slower to converge.
- If we averaged $\mathbf{m}_{h_j}$ instead of concatenating, we'd get to $221V + 6100$ (this is a variant of "continuous bag of words," Mikolov et al., 2013).

Why does it work?

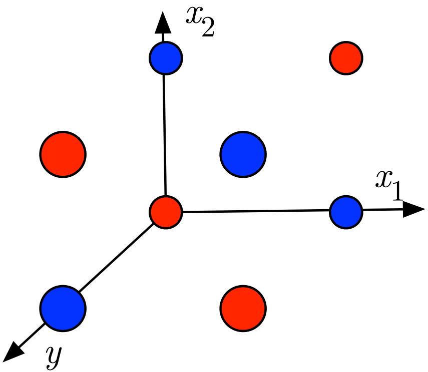
Why does it work?

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

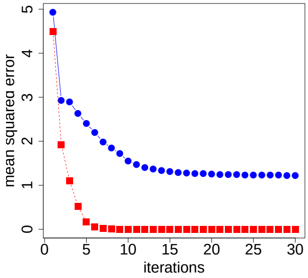
Why does it work?

- 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`)



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

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

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:

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

  - 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.

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.

Important Idea: Words as Vectors

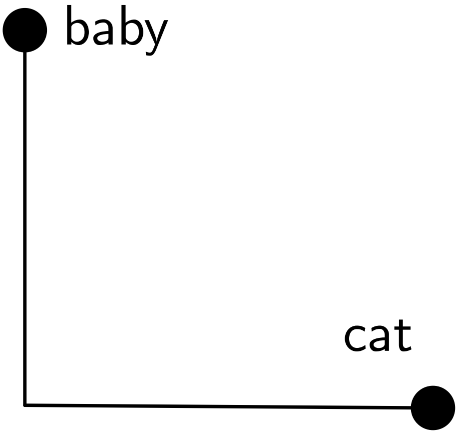
The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models.
You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

Important Idea: Words as Vectors

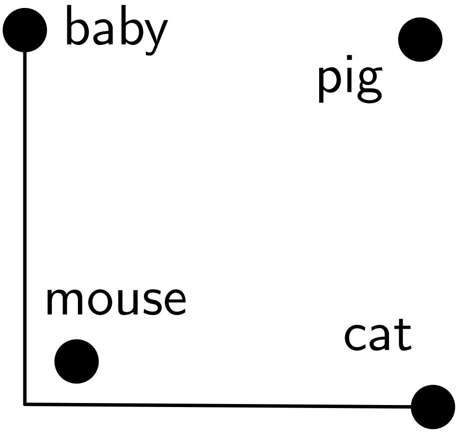
The idea of "embedding" words in $\mathbb{R}^d$ is much older than neural language models. You should think of this as a *generalization* of the discrete view of $\mathcal{V}$.

- 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



Words as Vectors: Example



Parameter Estimation

Bad news for neural language models:

- Log-likelihood function is not concave.
  - 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 concave.
  - 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:

- $n_{\boldsymbol{\nu}}$ is differentiable with respect to $\mathbf{M}$ (from which its inputs come) and $\boldsymbol{\nu}$ (its parameters), so gradient-based methods are available.
- Essential: the chain rule from calculus (sometimes called "backpropagation")

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

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)

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

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

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

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).

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.

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.

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:

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

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

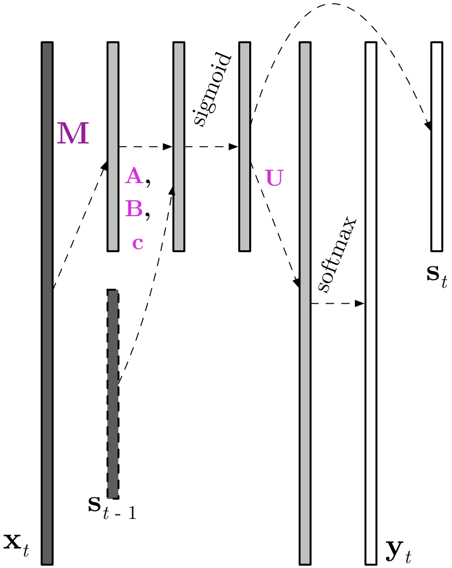
RNN Language Model

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

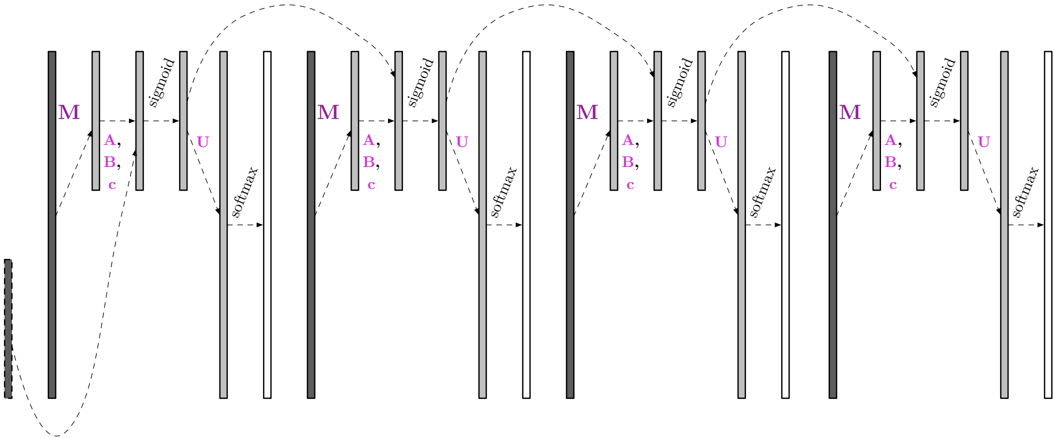
$$
\begin{aligned}
\mathbf{s}_t &= f_{\text{recurrent}}(\mathbf{e}_{x_t}, \mathbf{s}_{t-1}) = \text{sigmoid}\left(\left(\mathbf{e}_{x_t}^\top \mathbf{M}\right)^\top \mathbf{A} + \mathbf{s}_{t-1}^\top \mathbf{B} + \mathbf{c}\right) \\
\mathbf{y}_t &= f_{\text{output}}(\mathbf{s}_t) = \text{softmax}\left(\mathbf{s}_t^\top \mathbf{U}\right) \\
p(v \mid x_1, \ldots, x_{t-1}) &= [\mathbf{y}_t]_v
\end{aligned}
$$

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) 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, this requires a lot of engineering.

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.

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.

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		feature vector
2003–2010		embedded vector
since 2010	unrestricted	embedded

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.

Michael Collins. Log-linear models, MEMMs, and CRFs, 2011. URL `http://www.cs.columbia.edu/~mcollins/crf.pdf`.

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.

Jacob Eisenstein, Amr Ahmed, and Eric P Xing. Sparse additive generative models of text. In *Proc. of ICML*, 2011.

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`.

Daniel Jurafsky and James H. Martin. N-grams (draft chapter), 2016. URL `https://web.stanford.edu/~jurafsky/slp3/4.pdf`.

References II

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: A maximum entropy approach. In *Proc. of ICASSP*, 1993.

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`.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013. URL `http://arxiv.org/pdf/1301.3781.pdf`.

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.

Noah A. Smith. Probabilistic language models 1.0, 2017. URL `http://homes.cs.washington.edu/~nasmith/papers/plm.17.pdf`.

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`.