Natural Language Processing (CSE 517): Featurized and Neural Language Models

Noah Smith

University of Washington
nasmith@cs.washington.edu

April 6, 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: more details on log-linear language models, then neural language models

Log-Linear n-Gram Models

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

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:	$\dfrac{c(\boldsymbol{h}v)}{c(\boldsymbol{h})}$	no closed form

MLE for $\mathbf{w}$

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

MLE for $\mathbf{w}$

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

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 $\mathbf{w}$

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

Gradient view:

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

$$\nabla_{\mathbf{w}} F = \sum_{i=1}^{N}\left(\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \sum_{v\in\mathcal{V}} p_{\mathbf{w}}(v \mid \boldsymbol{h}_i)\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, v)\right)$$

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

MLE for $\mathbf{w}$: Algorithms

- Batch methods (L-BFGS is popular)
- Stochastic gradient 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.

ℓ_1 Regularization

This case warrants a little more discussion:

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v \in \mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda \|\mathbf{w}\|_1$$

Note that:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

- This results in **sparsity** (i.e., many $w_j = 0$).

ℓ_1 Regularization

This case warrants a little more discussion:

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v \in \mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda \|\mathbf{w}\|_1$$

Note that:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

- This results in **sparsity** (i.e., many $w_j = 0$).
  - Many have argued that this is a good thing (Tibshirani, 1996); it's a kind of feature selection.

ℓ_1 Regularization

This case warrants a little more discussion:

$$\max_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v\in\mathcal{V}} \exp \mathbf{w}\cdot\boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda\|\mathbf{w}\|_1$$

Note that:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

- This results in **sparsity** (i.e., many $w_j = 0$).
  - Many have argued that this is a good thing (Tibshirani, 1996); it's a kind of feature selection.
  - Do not confuse it with data *sparseness* (a problem to be overcome)!

ℓ_1 Regularization

This case warrants a little more discussion:

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v \in \mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda \|\mathbf{w}\|_1$$

Note that:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

- This results in **sparsity** (i.e., many $w_j = 0$).
  - Many have argued that this is a good thing (Tibshirani, 1996); it's a kind of feature selection.
  - Do not confuse it with data *sparseness* (a problem to be overcome)!
- This is not differentiable at $w_j = 0$.

ℓ_1 Regularization

This case warrants a little more discussion:

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, x_i) - \log \sum_{v \in \mathcal{V}} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{h}_i, v) - \lambda \|\mathbf{w}\|_1$$

Note that:

$$\|\mathbf{w}\|_1 = \sum_{j=1}^{d} |w_j|$$

- This results in **sparsity** (i.e., many $w_j = 0$).
  - Many have argued that this is a good thing (Tibshirani, 1996); it's a kind of feature selection.
  - Do not confuse it with data *sparseness* (a problem to be overcome)!
- This is not differentiable at $w_j = 0$.
- Optimization: special solutions for batch (e.g., Andrew and Gao, 2007) and stochastic (e.g., Langford et al., 2009) settings.

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.

Maximum Entropy

Consider a distribution p over events in $\mathcal{X}$. The Shannon entropy (in bits) of p is defined as:

$$H(p) = -\sum_{x \in \mathcal{X}} p(X = x) \begin{cases} 0 & \text{if } p(X = x) = 0 \\ \log_2 p(X = x) & \text{otherwise} \end{cases}$$

This is a measure of "randomness"; entropy is zero when p is deterministic and $\log |\mathcal{X}|$ when p is uniform.

Maximum entropy principle: among distributions that fit the data, pick the one with the greatest entropy.

Maximum Entropy

If "fit the data" is taken to mean

$$\forall k \in \{1, \ldots, d\}, \mathbb{E}_p[\phi_k] = \tilde{\mathbb{E}}[\phi_k]$$

then the MLE of the log-linear family with features ϕ is the maximum entropy solution.

This is why log-linear models are sometimes called "maxent" models (e.g., Berger et al., 1996)

"Whole Sentence" Log-Linear Models

(Rosenfeld, 1994)

Instead of a log-linear model for each word-given-history, define a single log-linear model over event space $\mathcal{V}^\dagger$:

$$p_{\mathbf{w}}(\boldsymbol{x}) = \frac{\exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})}{Z_{\mathbf{w}}}$$

- Any feature of the sentence could be included in this model!
- $Z_{\mathbf{w}}$ is deceptively simple-looking!

$$Z_{\mathbf{w}} = \sum_{\boldsymbol{x} \in \mathcal{V}^\dagger} \exp \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{x})$$

Quick Recap

Two kinds of language models so far:

	representation?	estimation?	think about?
n-gram	$\boldsymbol{h}_i$ is $(\mathrm{n}-1)$ 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 $\mathbb{R}^D$

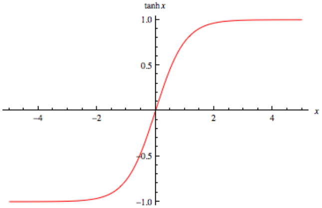
A Couple of Useful Functions

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

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

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

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



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 $\mathbf{e}_i \in \mathbb{R}^V$ 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.

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

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

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

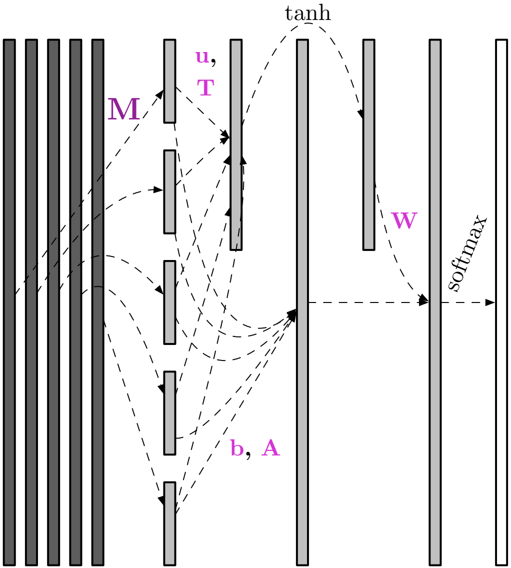
$$\text{softmax}\left(\mathbf{b} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \; \mathbf{A}_j \right.$$
$$\left. + \; \mathbf{W} \; \tanh\left(\mathbf{u} + \sum_{j=1}^{\mathsf{n}-1} \mathbf{m}_{h_j} \; \mathbf{T}_j \right) \right)$$

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).

References I

Galen Andrew and Jianfeng Gao. Scalable training of ℓ_1-regularized log-linear models. In *Proc. of ICML*, 2007.

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`.

Adam Berger, Stephen Della Pietra, and Vincent Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. In *NIPS*, 2009.

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`.

Roni Rosenfeld. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. PhD thesis, Carnegie Mellon University, 1994.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.