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

Noah Smith
© 2018

University of Washington
nasmith@cs.washington.edu

April 6, 2018
Quick Review

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

Typically $p$ decomposes into probabilities $p(x_i \mid h_i)$.

▶ n-gram: $h_i$ is $(n-1)$ previous symbols
▶ Probabilities are estimated from data.

Today: more details on log-linear language models, then neural language models
Log-Linear n-Gram Models

\[ p_w(X = x) = \prod_{j=1}^{\ell} p_w(X_j = x_j \mid X_{1:j-1} = x_{1:j-1}) \]

\[ = \prod_{j=1}^{\ell} \frac{\exp w \cdot \phi(x_{1:j-1}, x_j)}{Z_w(x_{1:j-1})} \]

assumption \[ = \prod_{j=1}^{\ell} \frac{\exp w \cdot \phi(x_{j-n+1:j-1}, x_j)}{Z_w(x_{j-n+1:j-1})} \]

\[ = \prod_{j=1}^{\ell} \frac{\exp w \cdot \phi(h_j, x_j)}{Z_w(h_j)} \]
How to Estimate \( \mathbf{w} \)?

**n-gram**

\[
p_{\theta}(x) = \prod_{j=1}^{\ell} \theta_{x_j|h_j}
\]

**Parameters:**

\[
\theta_{v|h}, \quad \forall v \in \mathcal{V}, h \in (\mathcal{V} \cup \{\circ\})^{n-1}
\]

**MLE:**

\[
\frac{c(hv)}{c(h)}
\]

**log-linear n-gram**

\[
\prod_{j=1}^{\ell} \exp \mathbf{w} \cdot \phi(h_j, x_j) / Z_w(h_j)
\]

**Parameters:**

\[
w_k, \quad \forall k \in \{1, \ldots, d\}
\]

**MLE:**

no closed form
MLE for $w$

$$\max_{w \in \mathbb{R}^d} \sum_{i=1}^{N} w \cdot \phi(h_i, x_i) - \log Z_w(h_i)$$
MLE for $w$

\[
\max_{w \in \mathbb{R}^d} \sum_{i=1}^{N} w \cdot \phi(h_i, x_i) - \log Z_w(h_i)
\]

Hope/fear view: for each instance $i$,

- increase the score of the correct output $x_i$, $score(x_i) = w \cdot \phi(h_i, x_i)$
- decrease the “softened max” score overall, $\log \sum_{v \in V} \exp score(v)$
MLE for $w$

$$
\max_{w \in \mathbb{R}^d} \sum_{i=1}^{N} w \cdot \phi(h_i, x_i) - \log Z_w(h_i)
$$

Gradient view:

$$
\nabla_w f_i = \phi(h_i, x_i) - \sum_{v \in V} p_w(v \mid h_i) \cdot \phi(h_i, v)
$$

observed features

$$
\nabla_w F = \sum_{i=1}^{N} \left( \phi(h_i, x_i) - \sum_{v \in V} p_w(v \mid h_i) \cdot \phi(h_i, v) \right)
$$

expected features

Setting this to zero means getting model’s expectations to match empirical expectations.
MLE for \( w \): 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 $\sum_{i=1}^{N} f_i(w)$ with respect to $w$.

Input: initial value $w$, number of epochs $T$, learning rate $\alpha$

For $t \in \{1, \ldots, T\}$:
  ▶ Choose a random permutation $\pi$ of $\{1, \ldots, N\}$.
  ▶ For $i \in \{1, \ldots, N\}$:

\[
    w \leftarrow w - \alpha \cdot \nabla_w f_{\pi(i)}
\]

Output: $w$
Avoiding Overfitting

Maximum likelihood estimation:

$$\max_{w \in \mathbb{R}^d} \sum_{i=1}^{N} w \cdot \phi(h_i, x_i) - \log Z_w(h_i)$$

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

Maximum likelihood estimation:

\[
\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \phi(h_i, x_i) - \log Z_{\mathbf{w}}(h_i)
\]

- If \( \phi_j(h, x) \) 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:

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

where \( \lambda > 0 \) is a hyperparameter and \( p = 2 \) or 1.
This case warrants a little more discussion:

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

Note that:

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

- This results in sparsity (i.e., many \(w_j = 0\)).
\textbf{\(\ell_1\) Regularization}

This case warrants a little more discussion:

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

Note that:

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

- This results in \textbf{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.
This case warrants a little more discussion:

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

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)!
\[ \ell_1 \text{ Regularization} \]

This case warrants a little more discussion:

\[
\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{N} \mathbf{w} \cdot \phi(h_i, x_i) - \log \sum_{v \in V} \exp \mathbf{w} \cdot \phi(h_i, v) - \lambda \| \mathbf{w} \|_1
\]

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 \( w_j = 0 \).
$\ell_1$ Regularization

This case warrants a little more discussion:

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

Note that:

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

- 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 $w$

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 ($Z_w(h_i)$).
▶ 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.
If “fit the data” is taken to mean

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

then the MLE of the log-linear family with features $\phi$ 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 $\mathcal{V}^\dagger$:

$$p_w(x) = \frac{\exp w \cdot \phi(x)}{Z_w}$$

- Any feature of the sentence could be included in this model!
- $Z_w$ is deceptively simple-looking!

$$Z_w = \sum_{x \in \mathcal{V}^\dagger} \exp w \cdot \phi(x)$$
Quick Recap

Two kinds of language models so far:

<table>
<thead>
<tr>
<th>representation?</th>
<th>estimation?</th>
<th>think about?</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-gram</td>
<td>count and normalize</td>
<td>smoothing</td>
</tr>
<tr>
<td>( h_i ) is ( (n - 1) ) previous symbols</td>
<td>iterative gradient descent</td>
<td>features</td>
</tr>
<tr>
<td>log-linear</td>
<td>featurized representation of ( \langle h_i, x_i \rangle )</td>
<td></td>
</tr>
</tbody>
</table>
Neural Network: Definitions

Warning: there is no widely accepted standard notation!

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

- A function family that maps parameter values to functions of the form $n : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$; 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 $\nu$
  - 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 $i$th word in $\mathcal{V}$.
Feedforward Neural Network Language Model

(Bengio et al., 2003)

Define the n-gram probability as follows:

\[
p(\cdot \mid \langle h_1, \ldots, h_{n-1} \rangle) = n_\nu \left( \langle e_{h_1}, \ldots, e_{h_{n-1}} \rangle \right) =
\]

\[
\text{softmax} \left( b + \sum_{j=1}^{n-1} e_{h_j}^\top M A_j + W \tanh \left( u + \sum_{j=1}^{n-1} e_{h_j}^\top M T_j \right) \right)
\]

where each \( e_{h_j} \in \mathbb{R}^V \) is a one-hot vector and \( H \) is the number of “hidden units” in the neural network (a “hyperparameter”).

Parameters \( \nu \) include:

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

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

$$\mathbf{e}_{h_j}^{\top} \mathbf{M}_{\nu \times d}$$
Breaking It Down

Look up each of the history words \( h_j, \forall j \in \{1, \ldots, n - 1\} \) in \( M \); keep two copies. Rename the embedding for \( h_j \) as \( m_{h_j} \).

\[
e_{h_j}^\top M = m_{h_j}
\]

\[
e_{h_j}^\top M = m_{h_j}
\]
Apply an affine transformation to the second copy of the history-word embeddings $(u, T)$

$$u + \sum_{j=1}^{n-1} m_{h_j} T_j$$
Apply an affine transformation to the second copy of the history-word embeddings \((\mathbf{u}, \mathbf{T})\) and a \(\tanh\) nonlinearity.

\[
\begin{align*}
\mathbf{m}_{h_j} \\
\tanh \left( \mathbf{u} + \sum_{j=1}^{n-1} \mathbf{m}_{h_j} \mathbf{T}_j \right)
\end{align*}
\]
Apply an affine transformation to everything \((b, A, W)\).
Breaking It Down

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

$$\text{softmax} \left( b + \sum_{j=1}^{n-1} m_{h_j} A_j \right)$$

$$+ W \ \text{tanh} \left( u + \sum_{j=1}^{n-1} m_{h_j} T_j \right)$$
Breaking It Down

\[
\text{softmax} \left( b + \sum_{j=1}^{n-1} m_{h_j} A_j \right) + W \text{ tanh} \left( u + \sum_{j=1}^{n-1} m_{h_j} T_j \right)
\]

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

- Concatenation of context-word embeddings vectors \( m_{h_j} \)
- \( \text{tanh} \)-affine transformation of the above

New parameters arise from (i) embeddings and (ii) affine transformation “inside” the nonlinearity.
Number of Parameters

\[ D = V_d + V_b + (n - 1)dV + V_H + H_u + (n - 1)dH \]

For Bengio et al. (2003):
- \( V \approx 18000 \) (after OOV processing)
- \( d \in \{30, 60\} \)
- \( H \in \{50, 100\} \)
- \( n - 1 = 5 \)

So \( D = 461V + 30100 \) parameters, compared to \( O(V^n) \) for classical n-gram models.
- Forcing \( A = 0 \) eliminated 300V parameters and performed a bit better, but was slower to converge.
- If we averaged \( m_{h,j} \) instead of concatenating, we’d get to \( 221V + 6100 \) (this is a variant of “continuous bag of words,” Mikolov et al., 2013).
References I


