Latent Models:
Sequence Models Beyond HMMs and Machine Translation Alignment

CMSC 473/673
UMBC
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
Why Do We Need Both the Forward and Backward Algorithms?

Compute posteriors

\[ \alpha(i, s) \times \beta(i, s) = \text{total probability of paths through state } s \text{ at step } i \]

\[
p(z_i = s \mid w_1, \ldots, w_N) = \frac{\alpha(i, s) \times \beta(i, s)}{\alpha(N + 1, \text{END})}
\]

\[ \alpha(i, s) \times p(s' \mid s) \times p(\text{obs at } i+1 \mid s') \times \beta(i+1, s') = \text{total probability of paths through the } s \rightarrow s' \text{ arc (at time } i) \]

\[
p(z_i = s, z_{i+1} = s' \mid w_1, \ldots, w_N) = \frac{\alpha(i, s) \times p(s'\mid s) \times p(\text{obs}_{i+1} \mid s') \times \beta(i + 1, s')}{\alpha(N + 1, \text{END})}
\]
EM for HMMs

1. Assume *some* value for your parameters

Two step, iterative algorithm

1. E-step: count under uncertainty, assuming these parameters

\[
p^*(z_i = s | w_1, \ldots, w_N) = \frac{\alpha(i, s) \cdot \beta(i, s)}{\alpha(N + 1, \text{END})}
\]

2. M-step: maximize log-likelihood, assuming these uncertain counts

\[
p^*(z_i = s, z_{i+1} = s' | w_1, \ldots, w_N) = \frac{\alpha(i, s) \cdot p(s'|s) \cdot p(\text{obs}_{i+1} | s') \cdot \beta(i + 1, s')}{\alpha(N + 1, \text{END})}
\]
\( \alpha = \text{computeForwards()} \)

\( \beta = \text{computeBackwards()} \)

\( L = \alpha[N+1][\text{END}] \)

\[
\text{for}(i = N; \ i \geq 0; \ --i) \ { }
\]

\[
\text{for}(\text{next} = 0; \ \text{next} < K*; \ ++\text{next}) \ { }
\]

\[
c_{\text{obs}}(\text{obs}_{i+1} \mid \text{next}) += \alpha[i+1][\text{next}] * \beta[i+1][\text{next}]/L
\]

\[
\text{for}(\text{state} = 0; \ \text{state} < K*; \ ++\text{state}) \ { }
\]

\[
u = p_{\text{obs}}(\text{obs}_{i+1} \mid \text{next}) * p_{\text{trans}}(\text{next} \mid \text{state})
\]

\[
c_{\text{trans}}(\text{next} \mid \text{state}) += \alpha[i][\text{state}] * u * \beta[i+1][\text{next}]/L
\]

\[
\}
\]

\[
\}
\]

update \( p_{\text{obs}}, p_{\text{trans}} \) using \( c_{\text{obs}}, c_{\text{trans}} \)
Semi-Supervised Learning

labeled data:
- human annotated
- relatively small/few examples

unlabeled data:
- raw; not annotated
- plentiful
Semi-Supervised Parameter Estimation for HMMs

Transition Counts

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>V</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Emission Counts

<table>
<thead>
<tr>
<th></th>
<th>w₁</th>
<th>w₂</th>
<th>W₃</th>
<th>w₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Mixed Transition Counts

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>3.8</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>N</td>
<td>2.5</td>
<td>2.8</td>
<td>2.1</td>
</tr>
<tr>
<td>V</td>
<td>3.4</td>
<td>2.1</td>
<td>.4</td>
</tr>
</tbody>
</table>

Mixed Emission Counts

<table>
<thead>
<tr>
<th></th>
<th>w₁</th>
<th>w₂</th>
<th>W₃</th>
<th>w₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>2.4</td>
<td>.3</td>
<td>1.2</td>
<td>2.2</td>
</tr>
<tr>
<td>V</td>
<td>.1</td>
<td>2.6</td>
<td>1.3</td>
<td>.3</td>
</tr>
</tbody>
</table>

Expected Transition Counts

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>V</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>1.8</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>N</td>
<td>1.5</td>
<td>.8</td>
<td>.1</td>
</tr>
<tr>
<td>V</td>
<td>1.4</td>
<td>1.1</td>
<td>.4</td>
</tr>
</tbody>
</table>

Expected Emission Counts

<table>
<thead>
<tr>
<th></th>
<th>w₁</th>
<th>w₂</th>
<th>W₃</th>
<th>w₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>.4</td>
<td>.3</td>
<td>.2</td>
<td>.2</td>
</tr>
<tr>
<td>V</td>
<td>.1</td>
<td>.6</td>
<td>.3</td>
<td>.3</td>
</tr>
</tbody>
</table>
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
Warren Weaver’s Note

When I look at an article in Russian, I say “This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.”

(Warren Weaver, 1947)

Noisy Channel Model

written in 
(clean) English

observed Russian (noisy) text

Decode

translation/ decode model

Rerank

(clean) language model

English

Slides courtesy Rebecca Knowles
Noisy Channel Model

\[ p(X | Y) \propto p(Y | X) \times p(X) \]

written in (clean) English

observed Russian (noisy) text

Decode

translation/decode model

Rerank

(clean) language model

English

Slides courtesy Rebecca Knowles
Noisy Channel Model

\[ p(X \mid Y) \propto p(Y \mid X) \times p(X) \]

written in (clean) English

observed Russian (noisy) text

language

Decide

translation/decode model

Rerank

(clean) language model

English

Slides courtesy Rebecca Knowles
Translation

Translate French (observed) into English:

Le chat est sur la chaise.

The cat is on the chair.

\[ p(English|French) \]
Translation

Translate French (observed) into English:

Le chat est sur la chaise.

The cat is on the chair.

\[ p(\text{English}|\text{French}) \propto p(\text{French}|\text{English}) \times p(\text{English}) \]
Translation

Translate French (observed) into English:

Le chat est sur la chaise.

The cat is on the chair.

$p(\text{English}|\text{French}) \propto p(\text{French}|\text{English}) \ast p(\text{English})$
Alignment

Le chat est sur la chaise.
The cat is on the chair.

$p(\text{English} | \text{French}) \propto p(\text{French} | \text{English}) \ast p(\text{English})$
Whereas recognition of the inherent dignity and of the equal and inalienable rights of all members of the human family is the foundation of freedom, justice and peace in the world,

Whereas disregard and contempt for human rights have resulted in barbarous acts which have outraged the conscience of mankind, and the advent of a world in which human beings shall enjoy freedom of speech and belief and freedom from fear and want has been proclaimed as the highest aspiration of the common people,

Whereas it is essential, if man is not to be compelled to have recourse, as a last resort, to rebellion against tyranny and oppression, that human rights should be protected by the rule of law,

Whereas it is essential to promote the development of friendly relations between nations,

Yolki, pampa ni tlatepanitalotl, ni tlasenkauajkayotl iuan ni kuali nemilistli ipan ni tlapalpan, yaya ni moneki moixmatis uan monemilis, ijkinoj nochi kuali tiitstosej ika tlaampoyouaj.

Pampa tlaj amo tikixmatij tlatepanitalistli uan tlen kuali nemilistli ipan ni tlapalpan, yeka onkatok kualantli, onkatok tlanteuilistli, onkatok majmajtli uan sekinok tlamantli teixpanolistli; yeka moneki ma kuali timouikakaj ika nochi touampoyouaj, ma amo onkaj majmajyotl uan teixpanolistli; moneki ma onkaj yejyektalalistli, ma titlajtlajtokaj uan ma tij detrimental tlen tijniountij tijnekij tijnextokasej uan amo tlen ma topanti, kenke, pampa tijnekij ma onkaj tlatepanitalistli.

Pampa ni tlatepanitalotl moneki ma tiyejekokaj, ma tijchiuakaj uan ma tijmanauikaj; ma nojkia kiixmatikaj tkeuakaj, ujeuyij tekiuakaj, ijkinoj amo onkas nopeka se akajya touampoj san tlen ueli kinekis techchiuilis, technauatis, kinekis technauatis ma tijchiuakaj se tlamantli tlen amo kuali; yeka ni tlatepanitalotl tlauel moneki ipan tonemilis ni tlapalpan.

Pampa nojkia tlauel moneki ma kuali timouikakaj, ma tielikaj keuak tijniountej, nochi tlen tlakamej uan siuamej tlen tijnextokej ni tlapalpan. ...


http://www.ohchr.org/EN/UDHR/Pages/Language.aspx?LangID=nhn

Slides courtesy Rebecca Knowles
Preprocessing

- Sentence align
- Clean corpus
- Tokenize
- Handle case
- Word segmentation (morphological, BPE, etc.)
- Language-specific preprocessing (example: pre-reordering)
- ...
Alignments

If we had word-aligned text, we could easily estimate $P(f|e)$. But we don’t usually have word alignments, and they are expensive to produce by hand...

If we had $P(f|e)$ we could produce alignments automatically.
IBM Model 1 (1993)

• Lexical Translation Model
• Word Alignment Model
• The simplest of the original IBM models
• For all IBM models, see the original paper (Brown et al, 1993):
Simplified IBM 1

• We’ll work through an example with a simplified version of IBM Model 1
• Figures and examples are drawn from A Statistical MT Tutorial Workbook, Section 27, (Knight, 1999)
• Simplifying assumption: each source word must translate to exactly one target word and vice versa
IBM Model 1 (1993)

- $f$: vector of French words
- $e$: vector of English words
- $a$: vector of alignment indices

Le chat est sur la chaise verte
The cat is on the green chair

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
</table>
IBM Model 1 (1993)

\( f \): vector of French words

*visualizaton of alignment*

\( e \): vector of English words

\( a \): vector of alignment indices

\[
P(a, f | e) = \prod_{j=1}^{m} t(f_j | e_{a_j}) = t(f_1 | e_{a_1}) \cdots t(f_m | e_{a_m})
\]

\( t(f_j | e_i) \): translation probability of the word \( f_j \) given the word \( e_i \)

\[
P(f | e) = \sum_{a} P(a, f | e)
\]

Le chat est sur la chaise verte

The cat is on the green chair

0 1 2 3 4 6 5

Slides courtesy Rebecca Knowles
Want: \( P(f|e) \)

But don’t know how to train this directly...

Solution: Use \( P(a, f|e) \), where \( a \) is an alignment

Remember:

\[
P(f|e) = \sum_a P(a, f|e)
\]
Model and Parameters: Intuition

Translation prob.: \[ t(f_j | e_i) \]

Example:
\[ t(\text{chaise} | \text{chair}) > t(\text{chaise} | \text{the}) \]

Interpretation:
How probable is it that we see \( f_j \) given \( e_i \)

Slides courtesy Rebecca Knowles
Model and Parameters: Intuition

Alignment/translation prob.: \( P(a, f | e) \)

**Example** (visual representation of \( a \)):

\[
P( \text{le chat} \mid \text{“the cat”}) < P( \text{le chat} \mid \text{“the cat”})
\]

**Interpretation:**
How probable are the alignment \( a \) and the translation \( f \) (given \( e \))
Model and Parameters: Intuition

Alignment prob.:

Example:

\[ P(a|e, f) \]

\[ P( \times | "le chat", "the cat") < P( | | | "le chat", "the cat") \]

Interpretation:

How probable is alignment \( a \) (given \( e \) and \( f \))

Slides courtesy Rebecca Knowles
Model and Parameters

How to compute:

\[ P(a, f \mid e) = \prod_{j=1}^{m} t(f_j \mid e_{a_j}) = t(f_1 \mid e_{a_1}) \cdots t(f_m \mid e_{a_m}) \]

\[ P(f \mid e) = \sum_{a} P(a, f \mid e) \]

\[ P(a \mid e, f) = \frac{P(a, f \mid e)}{\sum_{a'} P(a', f \mid e)} \]
Parameters

For IBM model 1, we can compute all parameters given translation parameters:

$$t(f_j | e_i)$$

How many of these are there?
Parameters

For IBM model 1, we can compute all parameters given translation parameters:

$$t(f_j | e_i)$$

How many of these are there?

$$|French\ vocabulary| \times |English\ vocabulary|$$
Data

Two sentence pairs:

<table>
<thead>
<tr>
<th>English</th>
<th>French</th>
</tr>
</thead>
<tbody>
<tr>
<td>b c</td>
<td>x y</td>
</tr>
<tr>
<td>b</td>
<td>y</td>
</tr>
</tbody>
</table>
All Possible Alignments

(French: x, y)

(English: b, c)

Remember:

simplifying assumption that each word must be aligned exactly once
Expectation Maximization (EM)

0. Assume some value for $t(f_j | e_i)$ and compute other parameter values

Two step, iterative algorithm

1. E-step: count alignments and translations under uncertainty, assuming these parameters

   $t(f_j | e_i)P(a | e, f) P(a, f | e) 

2. M-step: maximize log-likelihood (update parameters), using uncertain counts

Slides courtesy Rebecca Knowles
Review of IBM Model 1 & EM

Iteratively learned an alignment/translation model from sentence-aligned text (without “gold standard” alignments)

Model can now be used for alignment and/or word-level translation

We explored a simplified version of this; IBM Model 1 allows more types of alignments
Why is Model 1 insufficient?

Why won’t this produce great translations?
   Indifferent to order (language model may help?)
   Translates one word at a time
   Translates each word in isolation
   ...

Slides courtesy Rebecca Knowles
Uses for Alignments

Component of machine translation systems

Produce a translation lexicon automatically

Cross-lingual projection/extraction of information

Supervision for training other models (for example, neural MT systems)
Evaluating Machine Translation

Human evaluations:
Test set (source, human reference translations, MT output)

Humans judge the quality of MT output (in one of several possible ways)

Evaluating Machine Translation

Automatic evaluations:
Test set (source, human reference translations, MT output)

Aim to mimic (correlate with) human evaluations

Many metrics:
TER (Translation Error/Edit Rate)
HTER (Human-Targeted Translation Edit Rate)
BLEU (Bilingual Evaluation Understudy)
METEOR (Metric for Evaluation of Translation with Explicit Ordering)
Machine Translation Alignment Now

Explicitly with fancier IBM models

Implicitly/learned jointly with *attention* in recurrent neural networks (RNNs)
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
Recall: **N-gram** to Maxent to Neural Language Models

given some context...

compute beliefs about what is likely...

\[ p(w_i | w_{i-3}, w_{i-2}, w_{i-1}) \propto \text{count}(w_{i-3}, w_{i-2}, w_{i-1}, w_i) \]

predict the next word
Recall: N-gram to Maxent to Neural Language Models

given some context...

compute beliefs about what is likely...

\[ p(w_i | w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta \cdot f(w_{i-3}, w_{i-2}, w_{i-1}, w_i)) \]

predict the next word
Hidden Markov Model Representation

$$p(z_1, w_1, z_2, w_2, ..., z_N, w_N) = p(z_1 | z_0)p(w_1 | z_1) \cdots p(z_N | z_{N-1})p(w_N | z_N)$$

represent the probabilities and independence assumptions in a graph
A Different Model’s Representation

$z_1 \rightarrow z_2 \rightarrow z_3 \rightarrow z_4 \rightarrow \ldots$ represent the probabilities and independence assumptions in a graph.
A Different Model’s Representation

\[ p(z_1, z_2, ..., z_N|w_1, w_2, ..., w_N) = p(z_1|z_0, w_1) \ldots p(z_N|z_{N-1}, w_N) \]

\[ = \prod_i p(z_i|z_{i-1}, w_i) \]

represent the probabilities and independence assumptions in a graph
A Different Model’s Representation

\[ p(z_1, z_2, \ldots, z_N | w_1, w_2, \ldots, w_N) = p(z_1 | z_0, w_1) \cdots p(z_N | z_{N-1}, w_N) = \prod_i p(z_i | z_{i-1}, w_i) \]

\[ p(z_i | z_{i-1}, w_i) \propto \exp(\theta^T f(w_i, z_{i-1}, z_i)) \]

represent the probabilities and independence assumptions in a graph
Maximum Entropy Markov Model (MEMM)

A Different Model’s Representation

\[ p(z_1, z_2, ..., z_N | w_1, w_2, ..., w_N) = p(z_1 | z_0, w_1) \cdots p(z_N | z_{N-1}, w_N) \]

\[ = \prod_i p(z_i | z_{i-1}, w_i) \]

\[ p(z_i | z_{i-1}, w_i) \propto \exp(\theta^T f(w_i, z_{i-1}, z_i)) \]

represent the probabilities and independence assumptions in a graph
MEMMs

Discriminative: don’t care about generating observed sequence at all

Maxent: use features

Problem: Label-Bias problem
Label-Bias Problem

\[ z_i \]

\[ w_i \]
Label-Bias Problem

incoming mass must sum to 1
Label-Bias Problem

Incoming mass must sum to 1

Outgoing mass must sum to 1
Label-Bias Problem

Incoming mass must sum to 1

Outgoing mass must sum to 1

Observe, but do not generate (explain) the observation

Take-aways:
• the model can learn to ignore observations
• the model can get itself stuck on “bad” paths
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
(Linear Chain) Conditional Random Fields

Discriminative: don’t care about generating observed sequence at all

Condition on the *entire* observed word sequence $w_1 \ldots w_N$

Maxent: use features

Solves the label-bias problem
(Linear Chain) Conditional Random Fields

\[ p(z_1, \ldots, z_N \mid w_1, \ldots, w_N) \]
\[ \propto \prod_i \exp(\theta^T f(z_{i-1}, z_i, w_1, \ldots, w_N)) \]
(Linear Chain) Conditional Random Fields

\[ p(z_1, \ldots, z_N \mid w_1, \ldots, w_N) \]

\[ \propto \prod_i \exp(\theta^T f(z_{i-1}, z_i, w_1, \ldots, w_N)) \]

condition on entire sequence
Conditional vs. Sequence

Naive Bayes
Conditional vs. Sequence
Conditional vs. Sequence

- Naive Bayes
- Logistic Regression
- HMMs
- Linear-chain CRFs
Approaches to Modeling Sequences

**Conditional Models**
- Naive Bayes
- Logistic Regression

**Sequence Models**
- HMMs
- Linear-chain CRFs

**General Graph Models**
- Generative directed models
- General CRFs

---

CRF Tutorial, Fig 1.2, Sutton & McCallum (2012)
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
Recall: N-gram to Maxent to Neural Language Models

given some context...
create/use “distributed representations”...
combine these representations...
compute beliefs about what is likely...
predict the next word

\[ p(w_i| w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot f(w_{i-3}, w_{i-2}, w_{i-1})) \]
A More Typical View of Recurrent Neural Language Modeling
A More Typical View of Recurrent Neural Language Modeling

observe these words one at a time
A More Typical View of Recurrent Neural Language Modeling

predict the next word

observe these words one at a time

\[
\begin{align*}
w_{i-2} & \\
& \quad \rightarrow \\
h_{i-3} & \quad \rightarrow \\
& \quad \rightarrow \\
w_{i-3} & \\

w_{i-1} & \\
& \quad \rightarrow \\
h_{i-2} & \quad \rightarrow \\
& \quad \rightarrow \\
w_{i-2} & \\

w_i & \\
& \quad \rightarrow \\
h_{i-1} & \quad \rightarrow \\
& \quad \rightarrow \\
w_{i-1} & \\

w_{i+1} & \\
& \quad \rightarrow \\
h_i & \quad \rightarrow \\
& \quad \rightarrow \\
w_i & \\
\end{align*}
\]
A More Typical View of Recurrent Neural Language Modeling

predict the next word

from these hidden states

observe these words one at a time
A More Typical View of Recurrent Neural Language Modeling

predict the next word

from these hidden states

observe these words one at a time

"cell"
A Recurrent Neural Network Cell
A Recurrent Neural Network Cell
A Recurrent Neural Network Cell
A Recurrent Neural Network Cell

\[ W_i \]

\[ S \]

\[ h_{i-1} \]

\[ W \]

\[ h_i \]

\[ S \]

\[ U \]

\[ w_{i-1} \]

\[ w_i \]
A Simple Recurrent Neural Network Cell

\[ h_i = \sigma(W h_{i-1} + U w_i) \]
A Simple Recurrent Neural Network Cell

\[ h_i = \sigma(W h_{i-1} + U w_i) \]

\[ \sigma(x) = \frac{1}{1 + \exp(-x)} \]
A *Simple* Recurrent Neural Network Cell

\[
\begin{align*}
  h_i &= \sigma(W h_{i-1} + U w_i) \\
  \sigma(x) &= \frac{1}{1 + \exp(-x)}
\end{align*}
\]
A Simple Recurrent Neural Network Cell

\[
h_i = \sigma(W h_{i-1} + U w_i)
\]

\[
\sigma(x) = \frac{1}{1 + \exp(-x)}
\]
A Simple Recurrent Neural Network Cell

\[ h_i = \sigma(W h_{i-1} + U w_i) \]

\[ \hat{w}_{i+1} = \text{softmax}(Sh_i) \]

\[ \sigma(x) = \frac{1}{1 + \exp(-x)} \]
A Simple Recurrent Neural Network Cell

\[
h_i = \sigma(Wh_{i-1} + Uw_i)
\]

\[
\hat{w}_{i+1} = \text{softmax}(Sh_i)
\]

must learn matrices U, S, W
A Simple Recurrent Neural Network Cell

\[
\begin{align*}
h_i &= \sigma(W h_{i-1} + U w_i) \\
\hat{w}_{i+1} &= \text{softmax}(S h_i)
\end{align*}
\]

must learn matrices U, S, W

suggested solution: gradient descent on prediction ability
A Simple Recurrent Neural Network Cell

\[ h_i = \sigma(W h_{i-1} + U w_i) \]

\[ \hat{w}_{i+1} = \text{softmax}(S h_i) \]

must learn matrices U, S, W

suggested solution: gradient descent on prediction ability

problem: they’re tied across inputs/timesteps
A Simple Recurrent Neural Network Cell

\[ h_i = \sigma(W h_{i-1} + U w_i) \]

\[ \hat{w}_{i+1} = \text{softmax}(S h_i) \]

must learn matrices \( U, S, W \)

suggested solution: gradient descent on prediction ability

problem: they’re tied across inputs/timesteps

good news for you: many toolkits do this automatically
Why Is Training RNNs Hard?

Conceptually, it can get strange

But really getting the gradient just requires many applications of the chain rule for derivatives
Why Is Training RNNs Hard?

Conceptually, it can get strange

But really getting the gradient just requires many applications of the chain rule for derivatives

Vanishing gradients

Multiply the *same* matrices at *each* timestep ➔ multiply *many* matrices in the gradients
Why Is Training RNNs Hard?

Conceptually, it can get strange

But really getting the gradient just requires many applications of the chain rule for derivatives

Vanishing gradients

Multiply the \textit{same} matrices at \textit{each} timestep \Rightarrow multiply \textit{many} matrices in the gradients

One solution: clip the gradients to a max value
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch
Deep Learning
Natural Language Processing
Pick Your Toolkit

<table>
<thead>
<tr>
<th>PyTorch</th>
<th>Keras</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deeplearning4j</td>
<td>MxNet</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Gluon</td>
</tr>
<tr>
<td>DyNet</td>
<td>CNTK</td>
</tr>
<tr>
<td>Caffe</td>
<td>...</td>
</tr>
</tbody>
</table>

Comparisons:
- [https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch](https://deeplearning4j.org/compare-dl4j-tensorflow-pytorch)
- [https://github.com/zer0n/deepframeworks](https://github.com/zer0n/deepframeworks) (older---2015)
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden = self.i2h(combined)), output = self.i2o(combined), output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, n_letters))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden = self.i2h(combined))
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden = self.i2h(combined))
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, combined):
        hidden = self.hidden_size
        output = self.softmax(output)
        return output

    def initHidden(self):
        return Variable(torch.zeros(1, self.n_hidden))

n_hidden = 128
rnn = RNN(n_letters)
```
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(RNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, self.hidden_size))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```
Defining A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```python
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        combined = torch.cat(((input, hidden), 1))
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, self.hidden_size))

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

decode
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

criterion = nn.NLLLoss()

learning_rate = 0.005 # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

Negative log-likelihood

criterion = nn.NLLLoss()

learning_rate = 0.005  # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```
criterion = nn.NLLLoss()

learning_rate = 0.005  # If you set this too high, it might explode

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
```
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

Negative log-likelihood

criterion = nn.NLLLoss()

learning_rate = 0.005  # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

```
criterion = nn.NLLLoss()

learning_rate = 0.005  # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
```
Training A Simple RNN in Python

(Modified Very Slightly)

http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

Negative log-likelihood

criterion = nn.NLLLoss()

learning_rate = 0.005  # If you set this too high, it might explode. If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning_rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.data[0]
Another Solution: LSTMs/GRUs

LSTM: Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)

GRU: Gated Recurrent Unit (Cho et al., 2014)

Basic Ideas: *learn to forget*

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Outline

Review: EM for HMMs

Machine Translation Alignment

Limited Sequence Models
  Maximum Entropy Markov Models
  Conditional Random Fields

Recurrent Neural Networks
  Basic Definitions
  Example in PyTorch