Natural Language Processing

Info 159/259
Lecture 12: Neural sequence labeling (Feb 25, 2021)

David Bamman, UC Berkeley
MEMM Training

\[ \prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta) \]

For all training data, we want probability of the true label \( y_i \) conditioned on the previous true label \( y_{i-1} \) to be high.

This is simply multiclass logistic regression.
Label bias

\[
\prod_{i=1}^{n} P(y_i \mid y_{i-1}, x, \beta)
\]

- For a given conditioning context, the probability of a tag (e.g., VBZ) only competes against other tags with that same context (e.g., NN)

Bottou 2001; Lafferty et al. 2001
These probabilities must sum to 1 because the conditioning context is the same (local normalization).

Different conditioning contexts; \( P(y_2 = \text{TO}) \) will be 1 no matter the context.

\[
\begin{array}{|c|c|}
\hline
P(y_1 = \text{MD} \mid y_0 = \text{START}, x, \beta) & 0.9 \\
P(y_1 = \text{NN} \mid y_0 = \text{START}, x, \beta) & 0.1 \\
\hline
P(y_2 = \text{TO} \mid y_1 = \text{MD}, x, \beta) & 1.0 \\
P(y_2 = \text{TO} \mid y_1 = \text{NN}, x, \beta) & 1.0 \\
\hline
P(y_3 = \text{VB} \mid y_1 = \text{TO}, x, \beta) & 1.0 \\
\end{array}
\]
Here, the information that TO \textit{almost never} follows MD is lost and can't influence the tagging decision. We end up with an incorrect tagging.

\[
\begin{array}{|c|c|}
\hline
P(y_1 = \text{MD} \mid y_0 = \text{START}, x, \beta) & 0.9 \\
\hline
P(y_1 = \text{NN} \mid y_0 = \text{START}, x, \beta) & 0.1 \\
\hline
P(y_2 = \text{TO} \mid y_1 = \text{MD}, x, \beta) & 1.0 \\
\hline
P(y_2 = \text{TO} \mid y_1 = \text{NN}, x, \beta) & 1.0 \\
\hline
P(y_3 = \text{VB} \mid y_1 = \text{TO}, x, \beta) & 1.0 \\
\hline
\end{array}
\]

P(\text{NN TO VB} \mid x, \beta) = 0.1

P(\text{MD TO VB} \mid x, \beta) = 0.9
Conditional random fields

\[ P(y \mid x, \beta) = \frac{\exp(\Phi(x, y) \top \beta)}{\sum_{y' \in Y} \exp(\Phi(x, y') \top \beta)} \]

\[ \Phi(x, y) = \sum_{i=1}^{n} \phi(x, i, y_i, y_{i-1}) \]

Feature vector scoped over the entire input and label sequence

\( \phi \) is the same feature vector we used for local predictions using MEMMs
In an MEMM, we estimate $P(y_t \mid y_{t-1}, x, \beta)$ from each $\phi(x, t, y_t, y_{t-1})$ independently.

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$\phi(x, 1, y_1, y_0)$</th>
<th>$\phi(x, 2, y_2, y_1)$</th>
<th>$\phi(x, 3, y_3, y_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>will</td>
<td>NN</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>to</td>
<td>TO</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>fight</td>
<td>VB</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
In a CRF, we use features from the entire sequence (by summing the individual features at each time step)

<table>
<thead>
<tr>
<th>$x_i=$will $\land y_i = \text{NN}$</th>
<th>$x_i=$will $\land y_i = \text{MD}$</th>
<th>$x_i=$to $\land y_i = \text{TO}$</th>
<th>$x_i=$fight $\land y_i = \text{VB}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_{i-1}=$START $\land y_i = \text{NN}$</td>
<td>$y_{i-1}=$START $\land y_i = \text{MD}$</td>
<td>$y_{i-1}=$NN $\land y_i = \text{TO}$</td>
<td>$y_{i-1}=$MD $\land y_i = \text{TO}$</td>
</tr>
<tr>
<td>$\phi(x, 1, y_0)$</td>
<td>$\phi(x, 2, y_1)$</td>
<td>$\phi(x, 3, y_3)$</td>
<td>$\phi(x, \text{NN TO VB})$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
This lets us isolate the global sequence features that separate good sequences (in our training data) from bad sequences (not in our training data).

<table>
<thead>
<tr>
<th>Feature</th>
<th>(\phi(x, \text{NN TO VB})) GOOD</th>
<th>(\phi(x, \text{MD TO VB})) BAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_i = \text{will} \land y_i = \text{NN})</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(y_{i-1} = \text{START} \land y_i = \text{NN})</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(x_i = \text{will} \land y_i = \text{MD})</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(y_{i-1} = \text{START} \land y_i = \text{MD})</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(y_{i-1} = \text{NN} \land y_i = \text{TO})</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(y_{i-1} = \text{MD} \land y_i = \text{TO})</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(x_i = \text{to} \land y_i = \text{TO})</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(x_i = \text{fight} \land y_i = \text{VB})</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(y_{i-1} = \text{TO} \land y_i = \text{VB})</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

these are the different (and so are potentially predictive of a good label sequence)

these are the same (and so are not)
Conditional random fields

With a CRF, we have exactly the same parameters as we do with an equivalent MEMM; but we learn the best values of those parameters that leads to the best probability of the sequence overall (in our training data)
Recurrent neural network

- RNNs allow arbitrarily-sized conditioning contexts and condition on the entire sequence history.
RNNs for language modeling are already performing a kind of sequence labeling: at each time step, predict the word from \( \mathcal{Y} \) conditioned on the context.
For POS tagging, predict the tag from $y$ conditioned on the context.
RNNs for POS

• To make a prediction for \(y_t\), RNNs condition on all input seen through time \(t\) \((x_1, \ldots, x_t)\)

• But knowing something about the future can help \((x_{t+1}, \ldots, x_n)\)
Bidirectional RNN

• A powerful alternative is make predictions conditioning both on the past and the future.

• Two RNNs
  • One running left-to-right
  • One right-to-left

• Each produces an output vector at each time step, which we concatenate
I loved the movie!
Bidirectional RNN

I loved the movie!
I loved the movie!
Training BiRNNs

• Given this definition of an BiRNN:

\[
s_b^i = R_b(x^i, s_b^{i+1}) = g(s_b^{i+1}W_s^b + x^iW_x^b + b_b)
\]

\[
s_f^i = R_f(x^i, s_f^{i-1}) = g(s_f^{i-1}W_s^f + x^iW_x^f + b_f)
\]

\[
y_i = \text{softmax} \left( [s_f^i; s_b^i]W^o + b^o \right)
\]

• We have 8 sets of parameters to learn (3 for each RNN + 2 for the final layer)
Goldberg 2017
I loved the movie bigly

How do we fix this?
Subword information

• We saw subword information used for creating embeddings (FastText)

• Another alternative is to use standard word embeddings and reason about subword information within a model.
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

Lample et al. (2016), “Neural Architectures for Named Entity Recognition”
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

Lample et al. (2016), “Neural Architectures for Named Entity Recognition”
Character CNN for each word; concatenate character CNN output and word embedding as representation for a word.

Chu et al. (2016), “Named Entity Recognition with Bidirectional LSTM-CNNs”
RNNs for POS

Amazon and Spotify’s streaming services are going to devour **apple** and its music purchasing model.
RNNs for POS

Amazon and Spotify's streaming services are going to devour Apple and its music purchasing model.

Prediction: Can the information from far away get to the time step that needs it?

Training: Can error reach that far back during backpropagation?
RNNs

• Recurrent networks are deep in that they involve one “layer” for each time step (e.g., words in a sentence)

• Vanishing gradient problem: as error is back propagated through the layers of a deep network, they tend toward 0.
Long short-term memory network (LSTM)

- Designed to account for the vanishing gradient problem

- Basic idea: split the $s$ vector propagated between time steps into a memory component and a hidden state component
LSTMs

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

• LSTMs gates control the flow of information

• A sigmoid squashes its input to between 0 and 1
• By multiplying the output of a sigmoid elementwise with another vector, we forget information in the vector (if multiplied by 0) or allow it to pass (if multiplied by 1)
<table>
<thead>
<tr>
<th>input</th>
<th>3.7</th>
<th>1.4</th>
<th>-0.7</th>
<th>-1.4</th>
<th>7.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>gate</td>
<td>0.01</td>
<td>0.99</td>
<td>0.5</td>
<td>0.98</td>
<td>0.01</td>
</tr>
<tr>
<td>output</td>
<td>0.03</td>
<td>1.4</td>
<td>-0.35</td>
<td>-1.38</td>
<td>0.08</td>
</tr>
</tbody>
</table>
Forget gate: as a function of the previous hidden state and current input, forget information in the memory.
Input gate (but forget some information about the current observation)
Update the memory (but forget some information about the current observation)
The memory passes directly to the next state
Output gate: forget some information to send to the hidden state
The hidden state is updated with the current observation and new context.
Khandelwal et al. (2018), "Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context" (ACL)

How much context?

• For language modeling, LSTMs are aware of about 200 words of context

• Ignores word order beyond 50 words
GRU

• A gated recurrent unit adopts the same gating mechanism as an LSTM, but reduces the number of parameters to learn.

• Only one context vector (not a separate memory and hidden state vector) gets passed between timesteps.

• 2 gates (reset and update) instead of 3.
Neural sequence labeling

- Large design space for exploration in these models:
  - RNN/LSTM/GRU
  - Stacking
  - Hidden dimension size
  - Training with dropout and other forms of regularization.
LSTM/RNN

• Is an RNN the same kind of sequence labeling model as an MEMM or CRF?

• It doesn’t use nearby labels in making predictions! (More like logistic regression in this respect)
# Sequence labeling models

<table>
<thead>
<tr>
<th>model</th>
<th>form</th>
<th>label dependency</th>
<th>rich features?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden Markov Models</td>
<td>$\prod_{i=1}^{N} P(x_i \mid y_i) \cdot P(y_i \mid y_{i-1})$</td>
<td>Markov assumption</td>
<td>no</td>
</tr>
<tr>
<td>MEMM</td>
<td>$\prod_{i=1}^{N} P(y_i \mid y_{i-1}, x, \beta)$</td>
<td>Markov assumption</td>
<td>yes</td>
</tr>
<tr>
<td>CRF</td>
<td>$P(y \mid x, \beta)$</td>
<td>pairwise through entire sequence</td>
<td>yes</td>
</tr>
<tr>
<td>RNN</td>
<td>$\prod_{i=1}^{N} P(y_i \mid x_{1:i}, \beta)$</td>
<td>none</td>
<td>distributed</td>
</tr>
</tbody>
</table>
The information that’s passed between states is not the categorical choice (VBZ) but a hidden state that generated the distribution.
If we knew the categorical choice of VBZ at $t_2$, $P(VB)$ at $t_3$ would be much lower.
Recurrent neural network

• How could we incorporate nearby label information into a single-direction RNN?
Huang et al. 2015, "Bidirectional LSTM-CRF Models for Sequence Tagging"
Ma and Hovy (2016), "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF"
<table>
<thead>
<tr>
<th>Model</th>
<th>Dev Acc.</th>
<th>Test Acc.</th>
<th>Dev Prec.</th>
<th>Dev Recall</th>
<th>Dev F1</th>
<th>Test Prec.</th>
<th>Test Recall</th>
<th>Test F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRNN</td>
<td>96.56</td>
<td>96.76</td>
<td>92.04</td>
<td>89.13</td>
<td>90.56</td>
<td>87.05</td>
<td>83.88</td>
<td>85.44</td>
</tr>
<tr>
<td>BLSTM</td>
<td>96.88</td>
<td>96.93</td>
<td>92.31</td>
<td>90.85</td>
<td>91.57</td>
<td>87.77</td>
<td>86.23</td>
<td>87.00</td>
</tr>
<tr>
<td>BLSTM-CNN</td>
<td>97.34</td>
<td>97.33</td>
<td>92.52</td>
<td>93.64</td>
<td>93.07</td>
<td>88.53</td>
<td>90.21</td>
<td>89.36</td>
</tr>
<tr>
<td>BRNN-CNN-CRF</td>
<td>97.46</td>
<td>97.55</td>
<td>94.85</td>
<td>94.63</td>
<td>94.74</td>
<td>91.35</td>
<td>91.06</td>
<td>91.21</td>
</tr>
</tbody>
</table>

• Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.

• Generates multiple layers of representations for each token sensitive to its context of use.
Each token in the input starts out represented by token and position embeddings

-0.2  1  0.1 -0.8 -1.1
  e_{1,1}

0.3  0.3 -1.7  0.7 -1.1
  e_{1,2}

1.6 -0.3 -0.9 -0.7  0.2
  e_{1,3}

The  dog  barked
The value for time step \( j \) at layer \( i \) is the result of attention over all time steps in the previous layer \( i-1 \)

\[
egin{array}{cccccc}
-0.7 & -1.3 & 0.4 & -0.4 & -0.7 \\
\hline
e_{2,1} \\
\end{array}
\]

\[
egin{array}{cccccc}
-0.2 & 1 & 0.1 & -0.8 & -1.1 \\
\hline
e_{1,1} \\
\end{array}
\]

\[
egin{array}{cccccc}
0.3 & 0.3 & -1.7 & 0.7 & -1.1 \\
\hline
e_{1,2} \\
\end{array}
\]

\[
egin{array}{cccccc}
1.6 & -0.3 & -0.9 & -0.7 & 0.2 \\
\hline
e_{1,3} \\
\end{array}
\]

The, dog, barked
The dog barked.
The dog barked.
The dog barked
The dog barked
The dog barked.
The dog barked.
At the end of this process, we have one representation for each layer for each token.
• BERT can be used not only as a language model to generate contextualized word representations, but also as a predictive model whose parameters are fine-tuned to a task.
neutral

softmax

$b_{12,0} W^o$

```
[CLS] The dog bark #ed [SEP]
```
The dog bark

[CLS] The dog bark #ed [SEP]
BERT

• Pre-training: train BERT through masked language modeling and next-sentence prediction to learn the parameters of BERT layers. Trained on Wikipedia + BookCorpus.

• Task fine-tuning: add additional linear transformation + softmax to get distribution over output space. Trained on annotated data.