CS224d
Deep NLP

Lecture 7:
Recurrent Neural Networks

Richard Socher
richard@metamind.io
Overview

- Traditional language models
- RNNs
- RNN language models
- Important training problems and tricks
  - Vanishing and exploding gradient problems
- Bidirectional RNNs
- RNNs for other sequence tasks
A language model computes a probability for a sequence of words: \( P(w_1, \ldots, w_T) \)

- Useful for machine translation
  - Word ordering:
    \[ p(\text{the cat is small}) > p(\text{small the is cat}) \]
  - Word choice:
    \[ p(\text{walking home after school}) > p(\text{walking house after school}) \]
Traditional Language Models

- Probability is usually conditioned on window of $n$ previous words

- An incorrect but necessary Markov assumption!

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

- To estimate probabilities, compute for unigrams and bigrams (conditioning on one/two previous word(s):

\[
p(w_2 \mid w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \quad \quad p(w_3 \mid w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}
\]
Traditional Language Models

• Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)

• There are A LOT of n-grams!  
  ➔ Gigantic RAM requirements!

• Recent state of the art: *Scalable Modified Kneser-Ney Language Model Estimation* by Heafield et al.: “Using one machine with 140 GB RAM for 2.8 days, we built an unpruned model on 126 billion tokens”
Recurrent Neural Networks!

- RNNs tie the weights at each time step
- Condition the neural network on all previous words
- RAM requirement only scales with number of words
Recurrent Neural Network language model

Given list of word **vectors**: \( x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T \)

At a single time step:

\[
\begin{align*}
    h_t &= \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right) \\
    \hat{y}_t &= \text{softmax} \left( W^{(S)} h_t \right) \\
    \hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) &= \hat{y}_{t,j}
\end{align*}
\]
Recurrent Neural Network language model

Main idea: we use the same set of $W$ weights at all time steps!

Everything else is the same:

\[ h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]} \right) \]

\[ \hat{y}_t = \text{softmax} \left( W^{(S)} h_t \right) \]

\[ \hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j} \]

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of $L$ at index $[t]$ at time step $t$

$W^{(hh)} \in \mathbb{R}^{D_h \times D_h} \quad W^{(hx)} \in \mathbb{R}^{D_h \times d} \quad W^{(S)} \in \mathbb{R}^{|V| \times D_h}$
Recurrent Neural Network language model

\[ \hat{y} \in \mathbb{R}^{|V|} \] is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

\[
J^{(t)} (\theta) = - \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}
\]
Recurrent Neural Network language model

Evaluation could just be negative of average log probability over dataset of size (number of words) T:

\[ J = - \frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j} \]

But more common: Perplexity: \[ 2^J \]

Lower is better!
Training RNNs is hard

- Multiply the same matrix at each time step during forward prop

- Ideally inputs from many time steps ago can modify output $y$

- Take $\frac{\partial E_2}{\partial W}$ for an example RNN with 2 time steps! Insightful!
The vanishing/exploding gradient problem

- Multiply the same matrix at each time step during backprop
The vanishing gradient problem - Details

- Similar but simpler RNN formulation:

\[ h_t = Wf(h_{t-1}) + W^{(h, x)}x_t \]
\[ \hat{y}_t = W^{(S)}f(h_t) \]

- Total error is the sum of each error at time steps t

\[ \frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \]

- Hardcore chain rule application:

\[ \frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W} \]
The vanishing gradient problem - Details

• Similar to backprop but less efficient formulation

\[
\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}
\]

• Useful for analysis we’ll look at:

\[
h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}
\]

• Remember:

\[
\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}
\]

• More chain rule, remember:

Each partial is a Jacobian:

\[
\frac{df}{dx} = \left[ \frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right] = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]
The vanishing gradient problem - Details

- From previous slide: \( \frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \)

- Remember: \( h_t = Wf(h_{t-1}) + W^{(hx)}x^t \)

- To compute Jacobian, derive each element of matrix:

\[
\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^T \text{diag}[f'(h_{j-1})]
\]

- Where: \( \text{diag}(z) = \begin{pmatrix}
z_1 & & \\
& z_2 & & 0 \\
& & \ddots & \\
0 & & & z_{n-1} \\
& & & & z_n
\end{pmatrix} \)

Check at home that you understand the diag matrix formulation.
The vanishing gradient problem - Details

• Analyzing the norms of the Jacobians, yields:

\[
\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\text{diag}[f'(h_{j-1})]\| \leq \beta W \beta_h
\]

• Where we defined \( \beta' \)'s as upper bounds of the norms

• The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

\[
\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta W \beta_h)^{t-k}
\]

• This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. \( \rightarrow \) Vanishing or exploding gradient
Why is the vanishing gradient a problem?

- The error at a time step ideally can tell a previous time step from many steps away to change during backprop.
The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____
IPython Notebook with vanishing gradient example

- Example of simple and clean NNet implementation
- Comparison of sigmoid and ReLu units
- A little bit of vanishing gradient
In [21]: plt.plot(np.array(relu_array[:6000]), color='blue')
plt.plot(np.array(sigm_array[:6000]), color='green')
plt.title('Sum of magnitudes of gradients -- hidden layer neurons')

Out[21]: <matplotlib.text.Text at 0x10a331310>
Trick for exploding gradient: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a maximum value.

\[
\hat{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \theta} \\
\text{if } \|\hat{g}\| \geq \text{threshold} \text{ then} \\
\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g} \\
\text{end if}
\]

**Algorithm 1** Pseudo-code for norm clipping the gradients whenever they explode

- Makes a big difference in RNNs.
Gradient clipping intuition

- Error surface of a single hidden unit RNN,
- High curvature walls
- Solid lines: standard gradient descent trajectories
- Dashed lines gradients rescaled to fixed size

Figure from paper: On the difficulty of training Recurrent Neural Networks, Pascanu et al. 2013
For vanishing gradients: Initialization + ReLus!

- Initialize $W^{(*)}$‘s to identity matrix $I$ and $f(z) = \text{rect}(z) = \max(z, 0)$
- $\rightarrow$ Huge difference!

- Initialization idea first introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

- New experiments with recurrent neural nets 2 weeks ago (!) in *A Simple Way to Initialize Recurrent Networks of Rectified Linear Units*, Le et al. 2015
Perplexity Results

KN5 = Count-based language model with Kneser-Ney smoothing & 5-grams

Table 2. Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).

<table>
<thead>
<tr>
<th>Model</th>
<th>Penn Corpus</th>
<th></th>
<th>Switchboard</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NN</td>
<td>NN+KN</td>
<td>NN</td>
<td>NN+KN</td>
</tr>
<tr>
<td>KN5 (baseline)</td>
<td>-</td>
<td>141</td>
<td>-</td>
<td>92.9</td>
</tr>
<tr>
<td>feedforward NN</td>
<td>141</td>
<td>118</td>
<td>85.1</td>
<td>77.5</td>
</tr>
<tr>
<td>RNN trained by BP</td>
<td>137</td>
<td>113</td>
<td>81.3</td>
<td>75.4</td>
</tr>
<tr>
<td>RNN trained by BPTT</td>
<td>123</td>
<td>106</td>
<td>77.5</td>
<td>72.5</td>
</tr>
</tbody>
</table>

Table from paper *Extensions of recurrent neural network language model* by Mikolov et al 2011
Problem: Softmax is huge and slow

Trick: Class-based word prediction

\[ p(w_t|\text{history}) = p(c_t|\text{history})p(w_t|c_t) \]
\[ = p(c_t|h_t)p(w_t|c_t) \]

The more classes, the better perplexity but also worse speed:

<table>
<thead>
<tr>
<th>Classes</th>
<th>RNN</th>
<th>RNN+KN5</th>
<th>Min/epoch</th>
<th>Sec/test</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>134</td>
<td>112</td>
<td>12.8</td>
<td>8.8</td>
</tr>
<tr>
<td>50</td>
<td>136</td>
<td>114</td>
<td>9.8</td>
<td>6.7</td>
</tr>
<tr>
<td>100</td>
<td>136</td>
<td>114</td>
<td>9.1</td>
<td>5.6</td>
</tr>
<tr>
<td>200</td>
<td>136</td>
<td>113</td>
<td>9.5</td>
<td>6.0</td>
</tr>
<tr>
<td>400</td>
<td>134</td>
<td>112</td>
<td>10.9</td>
<td>8.1</td>
</tr>
<tr>
<td>1000</td>
<td>131</td>
<td>111</td>
<td>16.1</td>
<td>15.7</td>
</tr>
<tr>
<td>2000</td>
<td>128</td>
<td>109</td>
<td>25.3</td>
<td>28.7</td>
</tr>
<tr>
<td>4000</td>
<td>127</td>
<td>108</td>
<td>44.4</td>
<td>57.8</td>
</tr>
<tr>
<td>6000</td>
<td>127</td>
<td>109</td>
<td>70</td>
<td>96.5</td>
</tr>
<tr>
<td>8000</td>
<td>124</td>
<td>107</td>
<td>107</td>
<td>148</td>
</tr>
<tr>
<td>Full</td>
<td>123</td>
<td>106</td>
<td>154</td>
<td>212</td>
</tr>
</tbody>
</table>
Sequence modeling for other tasks

- Classify
  - NER
  - the sentiment of each word in its context
  - opinionated expressions

- Example application and slides from paper *Opinion Mining with Deep Recurrent Nets* by Irsoy and Cardie 2014
Opinion Mining with Deep Recurrent Nets

Goal: Classify each word as

direct subjective expressions (DSEs) and expressive subjective expressions (ESEs).

DSE: Explicit mentions of private states or speech events expressing private states

ESE: Expressions that indicate sentiment, emotion, etc. without explicitly conveying them.
Example Annotation

In BIO notation (tags either begin-of-entity (B_X) or continuation-of-entity (I_X)):
The committee, [as usual]_{ESE}, [has refused to make any statements]_{DSE}.

The committee, as usual, has refused to make any statements.

O   O   O   B_ESE  I_ESE  O   B_DSE

I_DSE  I_DSE  I_DSE  I_DSE  I_DSE  O
Approach: Recurrent Neural Network

- Notation from paper (so you get used to different ones)

\[
h_t = f(Wx_t + Vh_{t-1} + b) \\
y_t = g(Uh_t + c)
\]

- \(x\) represents a token (word) as a vector.
- \(y\) represents the output label (B, I or O) – \(g = \text{softmax}\)!
- \(h\) is the memory, computed from the past memory and current word. It summarizes the sentence up to that time.
Bidirectional RNNs

Problem: For classification you want to incorporate information from words both preceding and following

\[
\begin{align*}
\tilde{h}_t &= f(\overrightarrow{W}x_t + \overrightarrow{V}h_{t-1} + \overrightarrow{b}) \\
\tilde{h}_t &= f(\overleftarrow{W}x_t + \overleftarrow{V}h_{t+1} + \overleftarrow{b}) \\
y_t &= g(U[\tilde{h}_t; \tilde{h}_t] + c)
\end{align*}
\]

\[h = [\overrightarrow{h}; \overleftarrow{h}]\] now represents (summarizes) the past and future around a single token.
Deep Bidirectional RNNs

Each memory layer passes an intermediate sequential representation to the next.

\[
\begin{align*}
\vec{h}_t &= f(\vec{W} \, \vec{h}_{t}^{(i-1)} + \vec{V} \, \vec{h}_{t-1}^{(i)} + \vec{b}^{(i)}) \\
\overrightarrow{h}_t &= f(\overrightarrow{W} \, \overrightarrow{h}_t^{(i-1)} + \overrightarrow{V} \, \overrightarrow{h}_{t+1}^{(i)} + \overrightarrow{b}^{(i)}) \\
y_t &= g(U[\vec{h}_t^{(L)} ; \overrightarrow{h}_t^{(L)} ] + c)
\end{align*}
\]
Data

- MPQA 1.2 corpus (Wiebe et al., 2005)
- consists of 535 news articles (11,111 sentences)
- manually labeled with DSE and ESEs at the phrase level
- Evaluation: \( F_1 \)

\[
\begin{align*}
\text{precision} &= \frac{tp}{tp + fp} \\
\text{recall} &= \frac{tp}{tp + fn} \\
F_1 &= 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\end{align*}
\]
Results: Deep vs Shallow RNNs

Prop F1: DSE
Bin F1: # Layers

24k
200k