CS388: Natural Language Processing

Lecture 8: RNNs

Greg Durrett

Credit: Chelsea Voss csvoss.com
Mini 1 back today

Project 1 due tonight

Mini 2 out tonight
Recall: Word Vectors

[Finch and Chater 92, Shuetze 93, many others]
Recall: Skip-Gram

- Predict one word of context from word
  - \textit{the dog bit the man}

\[
P(w' | w) = \text{softmax}(W e(w))
\]

- Another training example: \textit{bit} -> \textit{the}

- Parameters: \(d \times |V|\) vectors, \(|V| \times d\) output parameters (W) (also usable as vectors!)

Mikolov et al. (2013)
This Lecture

- Evaluating word embeddings
- Recurrent neural networks: basics, issues
- LSTMs / GRUs
- Applications / visualizations
Evaluating Word Embeddings
Evaluating Word Embeddings

- What properties of language should word embeddings capture?
  - Similarity: similar words are close to each other
  - Analogy: good is to best as smart is to ???
    Paris is to France as Tokyo is to ???

Diagram:
- Nodes: cat, dog, tiger, wolf, good, enjoyable, great, bad, was, is
- Connections indicating similarity and analogy

Visual representation of word embeddings showing how similar words are placed close to each other in a high-dimensional space.
### Similarity

<table>
<thead>
<tr>
<th>Method</th>
<th>WordSim Similarity</th>
<th>WordSim Relatedness</th>
<th>Bruni et al. MEN</th>
<th>Radinsky et al. M. Turk</th>
<th>Luong et al. Rare Words</th>
<th>Hill et al. SimLex</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPMI</td>
<td>.755</td>
<td>.697</td>
<td>.745</td>
<td>.686</td>
<td>.462</td>
<td>.393</td>
</tr>
<tr>
<td>SVD</td>
<td>.793</td>
<td>.691</td>
<td>.778</td>
<td>.666</td>
<td>.514</td>
<td>.432</td>
</tr>
<tr>
<td>SGNS</td>
<td>.793</td>
<td>.685</td>
<td>.774</td>
<td>.693</td>
<td>.470</td>
<td>.438</td>
</tr>
<tr>
<td>GloVe</td>
<td>.725</td>
<td>.604</td>
<td>.729</td>
<td>.632</td>
<td>.403</td>
<td>.398</td>
</tr>
</tbody>
</table>

- SVD = singular value decomposition on PMI matrix
- GloVe does not appear to be the best when experiments are carefully controlled, but it depends on hyperparameters + these distinctions don’t matter in practice

Levy et al. (2015)
Analogies

\((\text{king} - \text{man}) + \text{woman} = \text{queen}\)

\(\text{king} + (\text{woman} - \text{man}) = \text{queen}\)

- Why would this be?
- woman - man captures the difference in the contexts that these occur in
- Dominant change: more “he” with man and “she” with woman — similar to difference between king and queen
- Can evaluate on this as well
What can go wrong with word embeddings?

- What’s wrong with learning a word’s “meaning” from its usage?
- What data are we learning from?
- What are we going to learn from this data?
What do we mean by bias?

- Identify *she* - *he* axis in word vector space, project words onto this axis.

- Nearest neighbor of \((b - a + c)\)

<table>
<thead>
<tr>
<th>Extreme <em>she</em> occupations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. homemaker</td>
</tr>
<tr>
<td>2. nurse</td>
</tr>
<tr>
<td>3. receptionist</td>
</tr>
<tr>
<td>4. librarian</td>
</tr>
<tr>
<td>5. socialite</td>
</tr>
<tr>
<td>6. hairdresser</td>
</tr>
<tr>
<td>7. nanny</td>
</tr>
<tr>
<td>8. bookkeeper</td>
</tr>
<tr>
<td>9. stylist</td>
</tr>
<tr>
<td>10. housekeeper</td>
</tr>
<tr>
<td>11. interior designer</td>
</tr>
<tr>
<td>12. guidance counselor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extreme <em>he</em> occupations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. maestro</td>
</tr>
<tr>
<td>2. skipper</td>
</tr>
<tr>
<td>3. protege</td>
</tr>
<tr>
<td>4. philosopher</td>
</tr>
<tr>
<td>5. captain</td>
</tr>
<tr>
<td>6. architect</td>
</tr>
<tr>
<td>7. financier</td>
</tr>
<tr>
<td>8. warrior</td>
</tr>
<tr>
<td>9. broadcaster</td>
</tr>
<tr>
<td>10. magician</td>
</tr>
<tr>
<td>11. fighter pilot</td>
</tr>
<tr>
<td>12. boss</td>
</tr>
</tbody>
</table>

Bolukbasi et al. (2016)

<table>
<thead>
<tr>
<th>Racial Analogies</th>
</tr>
</thead>
<tbody>
<tr>
<td>black → homeless</td>
</tr>
<tr>
<td>caucasian → hillbilly</td>
</tr>
<tr>
<td>asian → laborer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Religious Analogies</th>
</tr>
</thead>
<tbody>
<tr>
<td>jew → greedy</td>
</tr>
<tr>
<td>christian → familial</td>
</tr>
<tr>
<td>muslim → uneducated</td>
</tr>
</tbody>
</table>

Manzini et al. (2019)
Debiasing

- Identify gender subspace with gendered words
- Project words onto this subspace
- Subtract those projections from the original word

Bolukbasi et al. (2016)
Hardness of Debiasing

- Not that effective...and the male and female words are still clustered together

- Bias pervades the word embedding space and isn’t just a local property of a few words

(a) The plots for HARD-DEBIASED embedding, before (top) and after (bottom) debiasing.

Gonen and Goldberg (2019)
RNN Basics
RNN Motivation

- Feedforward NNs can’t handle variable length input: each position in the feature vector has fixed semantics
  
  ![Diagram showing different positions in feature vectors for words]

- Instead, we need to:
  1) Process each word in a uniform way
  2) …while still exploiting the context that that token occurs in

- These don’t look related (*great* is in two different orthogonal subspaces)
RNN Abstraction

- Cell that takes some input $x$, has some hidden state $h$, and updates that hidden state and produces output $y$ (all vector-valued)
RNN Uses

- Transducer: make some prediction for each element in a sequence.
  
  ```
  DT  | NN  | VBD | JJ
  the | movie | was | great
  ```

  Output $y = \text{score for each tag, then softmax}$

- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose.

  ```
  the | movie | was | great
  ```

  Predict sentiment (matmul + softmax), translate, paraphrase/compress.
Elman Networks

\[ h_t = \tanh(W x_t + V h_{t-1} + b_h) \]

- Updates hidden state based on input and current hidden state

\[ y_t = \tanh(U h_t + b_y) \]

- Computes output from hidden state

Elman (1990)
Training Elman Networks

“Backpropagation through time”: build the network as one big computation graph, some parameters are shared.

RNN potentially needs to learn how to “remember” information for a long time!

It was my favorite movie of 2016, though it wasn’t without problems. “Correct” parameter update is to do a better job of remembering the sentiment of favorite.
Vanishing Gradient

- Gradient diminishes going through tanh; if not in [-2, 2], gradient is almost 0
- Repeated multiplication by $V$ causes problems $h_t = \tanh(Wx_t + Vh_{t-1} + b_h)$
LSTMs/GRUs
Gated Connections

- Designed to fix “vanishing gradient” problem using gates

\[
\begin{align*}
    h_t &= h_{t-1} \odot f + \text{func}(x_t) \\
    h_t &= \tanh(Wx_t + Vh_{t-1} + b_h)
\end{align*}
\]

- Vector-valued “forget gate” \( f \) computed based on input and previous hidden state

\[
f = \sigma(W^{xf}x_t + W^{hf}h_{t-1})
\]

- Sigmoid: elements of \( f \) are in \((0, 1)\)

- If \( f \approx 1 \), we simply sum up a function of all inputs — gradient doesn’t vanish! More stable without matrix multiply \((V)\) as well
LSTMs

- “Long short-term memory” network: hidden state is a “short-term” memory

- “Cell” $c$ in addition to hidden state $h$
  
  \[ c_t = c_{t-1} \odot f + \text{func}(x_t, h_{t-1}) \]

- Vector-valued forget gate $f$ depends on the $h$ hidden state
  \[ f = \sigma(W^{xf}x_t + W^{hf}h_{t-1}) \]

- Basic communication flow: $x$ -> $c$ -> $h$ -> output, each step of this process is gated in addition to gates from previous timesteps
LSTMs

- $f, i, o$ are gates that control information flow
- $g$ reflects the main computation of the cell

\[ c_j = c_{j-1} \odot f + g \odot i \]
\[ f = \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \]
\[ g = \tanh(x_j W^{xg} + h_{j-1} W^{hg}) \]
\[ i = \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \]
\[ h_j = \tanh(c_j) \odot o \]
\[ o = \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \]

Goldberg lecture notes
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
LSTMs

- Can we ignore the old value of $c$ for this timestep?
- Can an LSTM sum up its inputs $x$?
- Can we ignore a particular input $x$?
- Can we output something without changing $c$?

\[
\begin{align*}
\mathbf{c}_j &= \mathbf{c}_{j-1} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{f} &= \sigma(\mathbf{x}_j \mathbf{W}^{xf} + \mathbf{h}_{j-1} \mathbf{W}^{hf}) \\
\mathbf{g} &= \tanh(\mathbf{x}_j \mathbf{W}^{xg} + \mathbf{h}_{j-1} \mathbf{W}^{hg}) \\
\mathbf{i} &= \sigma(\mathbf{x}_j \mathbf{W}^{xi} + \mathbf{h}_{j-1} \mathbf{W}^{hi}) \\
\mathbf{h}_j &= \tanh(\mathbf{c}_j) \odot \mathbf{o} \\
\mathbf{o} &= \sigma(\mathbf{x}_j \mathbf{W}^{xo} + \mathbf{h}_{j-1} \mathbf{W}^{ho})
\end{align*}
\]
LSTMs

- Ignoring recurrent state entirely:
  - Lets us get feedforward layer over token
- Ignoring input:
  - Lets us discard stopwords
- Summing inputs:
  - Lets us compute a bag-of-words representation

Goldberg lecture notes
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Gradient still diminishes, but in a controlled way and generally by less — sometimes initialize forget gate $= 1$ to remember everything to start

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
What do RNNs produce?

- **Encoding of the sentence** — can pass this a decoder or make a classification decision about the sentence

- **Encoding of each word** — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)

- **RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors**
Multilayer Bidirectional RNN

- Sentence classification based on concatenation of both final outputs
- Token classification based on concatenation of both directions’ token representations
Training RNNs

- Loss = negative log likelihood of probability of gold label (or use SVM or other loss)
- Backpropagate through entire network
- Example: sentiment analysis
Training RNNs

\[ P(t_i | x) \]

- Loss = negative log likelihood of probability of gold predictions, summed over the tags
- Loss terms filter back through network
- Example: language modeling (predict next word given context) or POS tagging
Applications
What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)
Visualizing LSTMs

- Train character LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code.
- Visualize activations of specific cells (components of c) to understand them.
- Counter: know when to generate \n
---

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action—the one Kutuzov and the general mass of the army demanded—namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all—carried on by vis inertiae—pressed forward into boats and into the ice-covered water and did not, surrender.

Karpathy et al. (2015)
Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Binary switch: tells us if we’re in a quote or not

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire. Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Karpathy et al. (2015)
Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code
- Visualize activations of specific cells to see what they track
- Stack: activation based on indentation

```
#define CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

Karpathy et al. (2015)
Visualizing LSTMs

- Train *character* LSTM language model (predict next character based on history) over two datasets: War and Peace and Linux kernel source code

- Visualize activations of specific cells to see what they track

- Uninterpretable: probably doing double-duty, or only makes sense in the context of another activation

```c
/* Unpack a filter field's string representation from user-space */
char *audit_unpack_string(void **bup, size_t *remain, size_t len)
{
    char *str;
    if (!*bupp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX defines the longest valid length. */
}
```

Karpathy et al. (2015)
What can LSTMs model?

- Sentiment
  - Encode one sentence, predict
- Language models
  - Move left-to-right, per-token prediction
- Translation
  - Encode sentence + then decode, use token predictions for attention weights (later in the course)
- Textual entailment
  - Encode two sentences, predict
Semi-supervised method: initialize the language model by training to reproduce the document in a seq2seq fashion (a type of pre-training called a sequential autoencoder)

<table>
<thead>
<tr>
<th>Model</th>
<th>Test error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM with tuning and dropout</td>
<td>13.50%</td>
</tr>
<tr>
<td>LSTM initialized with word2vec embeddings</td>
<td>10.00%</td>
</tr>
<tr>
<td>LM-LSTM (see Section 2)</td>
<td>7.64%</td>
</tr>
<tr>
<td>SA-LSTM (see Figure 1)</td>
<td>7.24%</td>
</tr>
<tr>
<td>Full+Unlabeled+BoW [21]</td>
<td>11.11%</td>
</tr>
<tr>
<td>WRRBM + BoW (bnc) [21]</td>
<td>10.77%</td>
</tr>
<tr>
<td>NBSVM-bi (Naïve Bayes SVM with bigrams) [35]</td>
<td>8.78%</td>
</tr>
<tr>
<td>seq2-bown-CNN (ConvNet with dynamic pooling) [11]</td>
<td>7.67%</td>
</tr>
<tr>
<td>Paragraph Vectors [18]</td>
<td>7.42%</td>
</tr>
</tbody>
</table>

Dai and Le (2015)
# Natural Language Inference

<table>
<thead>
<tr>
<th>Premise</th>
<th>Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>A boy plays in the snow</td>
<td>entails</td>
</tr>
<tr>
<td>The man is sleeping</td>
<td>A boy is outside</td>
</tr>
<tr>
<td>A man inspects the uniform of a figure</td>
<td>contradicts</td>
</tr>
<tr>
<td>Two men are smiling and laughing at cats playing</td>
<td>The man is sleeping</td>
</tr>
<tr>
<td>An older and younger man smiling</td>
<td>neutral</td>
</tr>
<tr>
<td>Two men are smiling and laughing at cats playing</td>
<td>Two men are smiling and laughing at cats playing</td>
</tr>
</tbody>
</table>

- Long history of this task: “Recognizing Textual Entailment” challenge in 2006 (Dagan, Glickman, Magnini)
- Early datasets: small (hundreds of pairs), very ambitious (lots of world knowledge, temporal reasoning, etc.)
SNLI Dataset

- Show people captions for (unseen) images and solicit entailed / neural / contradictory statements
- >500,000 sentence pairs
- Encode each sentence and process
  - 100D LSTM: 78% accuracy
  - 300D LSTM: 80% accuracy (Bowman et al., 2016)
  - 300D BiLSTM: 83% accuracy (Liu et al., 2016)
- Later: better models for this
Takeaways

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector.

- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation.