Automatic Speech Recognition (CS753)
Lecture 14: Language Models (Part III)

Instructor: Preethi Jyothi
Sep 18, 2017
Recap of Ngram language models

- For a word sequence $W = w_1, w_2, \ldots, w_{n-1}, w_n$, an Ngram model predicts $w_i$ based on $w_{i-(N-1)}, \ldots, w_{i-1}$

- Practically impossible to see most Ngrams during training

- This is addressed using smoothing techniques involving interpolation and backoff models
Looking beyond words

• Many unseen word Ngrams during training

This guava is yellow

This *dragonfruit* is yellow  [*dragonfruit* → unseen]

• What if we move from word Ngrams to "**class**" Ngrams?

\[
\Pr(\text{Color}|\text{Fruit, Verb}) = \frac{\pi(\text{Fruit, Verb, Color})}{\pi(\text{Fruit, Verb})}
\]

• Function mapping each word w to one of C classes
Computing word probabilities from class probabilities

- \( \Pr(w_i | w_{i-1}, ..., w_{i-n+1}) \approx \Pr(w_i | c(w_i)) \times \Pr(c(w_i) | c(w_{i-1}), ..., c(w_{i-n+1})) \)

- We want \( \Pr(\text{Red} | \text{Apple}, \text{is}) \)

\[
= \Pr(\text{COLOR} | \text{FRUIT, VERB}) \times \Pr(\text{Red} | \text{COLOR})
\]

- How are words assigned to classes? Unsupervised clustering algorithm that groups “related words” into the same class [Brown92]

- Using classes, reduction in number of parameters:
  \( V^N \rightarrow VC + C^N \)

- Both class-based and word-based LMs could be interpolated
Interpolate many models vs build one model

- Instead of interpolating different language models, can we come up with a single model that combines different information sources about a word?

- Maximum-entropy language models [R94]

---

[R94] Rosenfeld, “A Maximum Entropy Approach to SLM”, CSL 96
Maximum Entropy LMs

Probability of a word $w$ given history $h$ has a log-linear form:

$$P_{\Lambda}(w|h) = \frac{1}{Z_{\Lambda}(h)} \exp \left( \sum_{i} \lambda_i \cdot f_i(w, h) \right)$$

where

$$Z_{\Lambda}(h) = \sum_{w' \in V} \exp \left( \sum_{i} \lambda_i \cdot f_i(w', h) \right)$$

Each $f_i(w, h)$ is a feature function. E.g.

$$f_i(w, h) = \begin{cases} 1 & \text{if } w = a \text{ and } h \text{ ends in } b \\ 0 & \text{otherwise} \end{cases}$$

$\lambda$’s are learned by fitting the training sentences using a maximum likelihood criterion.
Word representations in Ngram models

- In standard Ngram models, words are represented in the discrete space involving the vocabulary.

- Limits the possibility of truly interpolating probabilities of unseen Ngrams.

- Can we build a representation for words in the continuous space?
Word representations

• 1-hot representation:

  • Each word is given an index in \( \{1, \ldots, V\} \). The 1-hot vector \( f_i \in \mathbb{R}^V \) contains zeros everywhere except for the \( i^{th} \) dimension being 1

• 1-hot form, however, doesn’t encode information about word similarity

• Distributed (or continuous) representation: Each word is associated with a dense vector. E.g. 
  dog → \{-0.02, -0.37, 0.26, 0.25, -0.11, 0.34\}
Word embeddings

• These distributed representations in a continuous space are also referred to as “word embeddings”
  • Low dimensional
  • Similar words will have similar vectors

• Word embeddings capture semantic properties (such as man is to woman as boy is to girl, etc.) and morphological properties (glad is similar to gladly, etc.)
## Word embeddings

<table>
<thead>
<tr>
<th>Country</th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
<th>Word 4</th>
<th>Word 5</th>
<th>Word 6</th>
<th>Word 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Jesus</td>
<td>Xbox</td>
<td>Reddish</td>
<td>Scratched</td>
<td>Megabits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td>God</td>
<td>Amiga</td>
<td>Greenish</td>
<td>Nailed</td>
<td>Octets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belgium</td>
<td>Sati</td>
<td>PlayStation</td>
<td>Bluish</td>
<td>Smashed</td>
<td>Mb/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>Christ</td>
<td>Msx</td>
<td>Pinkish</td>
<td>Punched</td>
<td>Baud</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td>Satan</td>
<td>Ipod</td>
<td>Purplish</td>
<td>Popped</td>
<td>Carats</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greece</td>
<td>Kali</td>
<td>Sega</td>
<td>Brownish</td>
<td>Crimped</td>
<td>Kbit/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sweden</td>
<td>Indra</td>
<td>Psnumber</td>
<td>Greyish</td>
<td>Scraped</td>
<td>Megahertz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Norway</td>
<td>Vishnu</td>
<td>HD</td>
<td>Grayish</td>
<td>Screwed</td>
<td>Megapixels</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Europe</td>
<td>Ananda</td>
<td>Dreamcast</td>
<td>Whitish</td>
<td>Sectioned</td>
<td>Gbit/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hungary</td>
<td>Parvati</td>
<td>Geforce</td>
<td>Silvery</td>
<td>Slashed</td>
<td>Ampere</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>Grace</td>
<td>Capcom</td>
<td>Yellowish</td>
<td>Ripped</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Relationships learned from embeddings

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>France - Paris</td>
<td>Italy: Rome</td>
<td>Japan: Tokyo</td>
<td>Florida: Tallahassee</td>
</tr>
<tr>
<td>big - bigger</td>
<td>small: larger</td>
<td>cold: colder</td>
<td>quick: quicker</td>
</tr>
<tr>
<td>Miami - Florida</td>
<td>Baltimore: Maryland</td>
<td>Dallas: Texas</td>
<td>Kona: Hawaii</td>
</tr>
<tr>
<td>Einstein - scientist</td>
<td>Messi: midfielder</td>
<td>Mozart: violinist</td>
<td>Picasso: painter</td>
</tr>
<tr>
<td>Sarkozy - France</td>
<td>Berlusconi: Italy</td>
<td>Merkel: Germany</td>
<td>Koizumi: Japan</td>
</tr>
<tr>
<td>copper - Cu</td>
<td>zinc: Zn</td>
<td>gold: Au</td>
<td>uranium: plutonium</td>
</tr>
<tr>
<td>Berlusconi - Silvio</td>
<td>Sarkozy: Nicolas</td>
<td>Putin: Medvedev</td>
<td>Obama: Barack</td>
</tr>
<tr>
<td>Microsoft - Windows</td>
<td>Google: Android</td>
<td>IBM: Linux</td>
<td>Apple: iPhone</td>
</tr>
<tr>
<td>Microsoft - Ballmer</td>
<td>Google: Yahoo</td>
<td>IBM: McNealy</td>
<td>Apple: Jobs</td>
</tr>
<tr>
<td>Japan - sushi</td>
<td>Germany: bratwurst</td>
<td>France: tapas</td>
<td>USA: pizza</td>
</tr>
</tbody>
</table>

[M13]: Mikolov et al., 13
Bilingual embeddings

[S13]: Socher et al., 13
Word embeddings

• These distributed representations in a continuous space are also referred to as “word embeddings”
  • Low dimensional
  • Similar words will have similar vectors

• Word embeddings capture semantic properties (such as man is to woman as boy is to girl, etc.) and morphological properties (glad is similar to gladly, etc.)

• The word embeddings could be learned via the first layer of a neural network [B03].

[B03]: Bengio et al., “A neural probabilistic LM”, JMLR, 03
Continuous space language models

The idea behind short-lists is to use the neural network to predict the LM probabilities of a subset of the hypotheses, which are then used to create a lattice as input to the translation decoder. A direct search method is used to perform this search. A lattice is created that has one node per word for all words. The neural network is used to assign a probability to each node in the lattice. The neural network is trained to minimize the perplexity on the training data. The complexity to calculate one probability is $O(P^n)$, where $P$ is the dimension of the vector and $n$ is the number of hidden neurons. Note also that the gradient is back-propagated through the projection-layer, which corresponds directly to the word-based architecture.

The complexity to calculate one probability is $O(P^n)$, where $P$ is the dimension of the vector and $n$ is the number of hidden neurons. Note also that the gradient is back-propagated through the projection-layer, which corresponds directly to the word-based architecture.

For all words, the statistical machine translation (SMT) decoder generates a model. The neural network is then used to score the lattice as input to the decoding engine. It can be shown that the output of a neural network is a continuous representation of the words onto the continuous space that is well suited for SMT.
NN language model

- Project all the words of the context $h_j = w_{j-n+1}, \ldots, w_{j-1}$ to their dense forms

- Then, calculate the language model probability $Pr(w_j = i | h_j)$ for the given context $h_j$
NN language model

- Dense vectors of all the words in context are concatenated forming the first hidden layer of the neural network.
- Second hidden layer:
  \[ d_k = \tanh(\sum m_{kj}c_j + b_k) \forall k = 1, ..., H \]
- Output layer:
  \[ o_i = \sum v_{ik}d_k + b_i \forall i = 1, ..., N \]
- \( p_i \rightarrow \text{softmax output from the ith neuron} \rightarrow \Pr(w_j = i|h_j) \)
NN language model

• Model is trained to minimise the following loss function:

\[ L = \sum_{i=1}^{N} t_i \log p_i + \epsilon \left( \sum_{kl} m_{kl}^2 + \sum_{ik} v_{ik}^2 \right) \]

• Here, \( t_i \) is the target output 1-hot vector (1 for next word in the training instance, 0 elsewhere)

• First part: Cross-entropy between the target distribution and the distribution estimated by the NN

• Second part: Regularization term
Decoding with NN LMs

- Two main techniques used to make the NN LM tractable for large vocabulary ASR systems:
  1. Lattice rescoring
  2. Shortlists
Use NN language model via lattice rescoring

- Lattice — Graph of possible word sequences from the ASR system using an Ngram backoff LM
- Each lattice arc has both acoustic/language model scores.
- LM scores on the arcs are replaced by scores from the NN LM
Decoding with NN LMs

- Two main techniques used to make the NN LM tractable for large vocabulary ASR systems:
  1. Lattice rescoring
  2. Shortlists
Shortlist

- Softmax normalization of the output layer is an expensive operation esp. for large vocabularies

- Solution: Limit the output to the $s$ most frequent words.
  
  - LM probabilities of words in the short-list are calculated by the NN
  
  - LM probabilities of the remaining words are from Ngram backoff models
Results

Table 3
Perplexities on the 2003 evaluation data for the back-off and the hybrid LM as a function of the size of the CTS training data

<table>
<thead>
<tr>
<th>CTS corpus (words)</th>
<th>7.2 M</th>
<th>12.3 M</th>
<th>27.3 M</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In-domain data only</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Back-off LM</td>
<td>62.4</td>
<td>55.9</td>
<td>50.1</td>
</tr>
<tr>
<td>Hybrid LM</td>
<td>57.0</td>
<td>50.6</td>
<td><strong>45.5</strong></td>
</tr>
<tr>
<td><strong>Interpolated with all data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Back-off LM</td>
<td>53.0</td>
<td>51.1</td>
<td><strong>47.5</strong></td>
</tr>
<tr>
<td>Hybrid LM</td>
<td>50.8</td>
<td>48.0</td>
<td>44.2</td>
</tr>
</tbody>
</table>

[S07]: Schwenk et al., “Continuous space language models”, CSL, 07
Longer word context?

- What have we seen so far: A feedforward NN used to compute an Ngram probability \( \text{Pr}(w_j = i | h_j) \) (where \( h_j \) is the Ngram history)

- We know Ngrams are limiting: Alice who had attempted the assignment asked the lecturer

- How can we predict the next word based on the entire sequence of preceding words? Use recurrent neural networks.
Simple RNN language model

- Current word, $x_t$
- Hidden state, $s_t$
- Output, $y_t$

$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = \text{softmax}(Vs_t)$$

- RNN is trained using the cross-entropy criterion
RNN-LMs

- Optimizations used for NNLMs are relevant to RNN-LMs as well (rescoring Nbest lists or lattices, using a shortlist, etc.)

- Perplexity reductions over Kneser-Ney models:

<table>
<thead>
<tr>
<th>Model</th>
<th># words</th>
<th>PPL</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN5 LM</td>
<td>200K</td>
<td>336</td>
<td>16.4</td>
</tr>
<tr>
<td>KN5 LM + RNN 90/2</td>
<td>200K</td>
<td>271</td>
<td>15.4</td>
</tr>
<tr>
<td>KN5 LM</td>
<td>1M</td>
<td>287</td>
<td>15.1</td>
</tr>
<tr>
<td>KN5 LM + RNN 90/2</td>
<td>1M</td>
<td>225</td>
<td>14.0</td>
</tr>
<tr>
<td>KN5 LM</td>
<td>6.4M</td>
<td>221</td>
<td>13.5</td>
</tr>
<tr>
<td>KN5 LM + RNN 250/5</td>
<td>6.4M</td>
<td>156</td>
<td>11.7</td>
</tr>
</tbody>
</table>
Training RNN-LMs

- RNN-LMs are trained using backpropagation through time (BPTT): Unfold the RNN in time + train the unfolded RNN using backpropagation + mini-batch gradient descent
- Main issues with BPTT: Exploding and vanishing gradients
  - Exploding gradients: Gradients can increase exponentially over time during backpropagation. Clip values of gradients to handle this.
  - Vanishing gradients: Magnitude of gradients approach very tiny values as we propagate gradients back in time. Architectures like Long Short Term Memory (LSTMs) networks can handle this.
**LSTM-LMs**

- Vanilla RNN-LMs unlikely to show full potential of recurrent models due to issues like vanishing gradients

- LSTM-LMs: Similar to RNN-LMs except use LSTM units in the 2nd hidden (recurrent) layer

---

Image from: Sundermeyer et al., “LSTM NNs for Language Modeling”, IS 10
Comparing RNN-LMs with LSTM-LMs

Figure 3: Experimental results on the Treebank corpus; for (c) and (d), 200 nodes were used for the hidden layers.

Experiments suggest that the performance of standard recurrent neural network architectures can be improved by about 8% relative in terms of perplexity. Finally, comparatively large improvements were obtained when interpolating an LSTM LM with a huge Kneser-Ney smoothed backing-off model on top of a state-of-the-art French recognition system.

For future work, it seems interesting to analyze the differences between standard and LSTM networks and the impact on the recognition quality of a speech recognizer.

6. Acknowledgment

This work was partly realized as part of the Quaero programme, funded by OSEO, French State agency for innovation.

7. References


Image from: Sundermeyer et al., “LSTM NNs for Language Modeling”, 10
Character-based RNN-LMs

Image from: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Generate text using a trained character-based LSTM-LM

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.