RNN-based AMs
+ Introduction to Language Modeling

Lecture 9

Instructor: Preethi Jyothi
Recall RNN definition

Two main equations govern RNNs:

\[ h_t = H(Wx_t + Vh_{t-1} + b^{(h)}) \]
\[ y_t = O(Uh_t + b^{(y)}) \]

where \( W, V, U \) are matrices of input-hidden weights, hidden-hidden weights and hidden-output weights resp; \( b^{(h)} \) and \( b^{(y)} \) are bias vectors and \( H \) is the activation function applied to the hidden layer
Training RNNs

• An unrolled RNN is just a very deep feedforward network

• For a given input sequence:
  • create the unrolled network
  • add a loss function node to the network
  • then, use backpropagation to compute the gradients

• This algorithm is known as backpropagation through time (BPTT)
Deep RNNs

- RNNs can be stacked in layers to form deep RNNs
- Empirically shown to perform better than shallow RNNs on ASR [G13]

Vanilla RNN Model

\[ h_t = H(Wx_t + Vh_{t-1} + b^{(h)}) \]

\[ y_t = O(Uh_t + b^{(y)}) \]

H : element wise application of the sigmoid or tanh function

O : the softmax function

Run into problems of exploding and vanishing gradients.
Exploding/Vanishing Gradients

- In deep networks, gradients in early layers are computed as the product of terms from all the later layers

- This leads to unstable gradients:
  - If the terms in later layers are large enough, gradients in early layers (which is the product of these terms) can grow exponentially large: Exploding gradients
  - If the terms are in later layers are small, gradients in early layers will tend to exponentially decrease: Vanishing gradients

- To address this problem in RNNs, Long Short Term Memory (LSTM) units were proposed [HS97]

• Memory cell: Neuron that stores information over long time periods
• Forget gate: When on, memory cell retains previous contents. Otherwise, memory cell forgets contents.
• When input gate is on, write into memory cell
• When output gate is on, read from the memory cell
• BiRNNs process the data in both directions with two separate hidden layers

• Outputs from both hidden layers are concatenated at each position
ASR with RNNs

• We have seen how neural networks can be used for acoustic models in ASR systems

• Main limitation: Frame-level training targets derived from HMM-based alignments

• Goal: Single RNN model that addresses this issues and does not rely on HMM-based alignments [G14]

• $H$ was implemented using LSTMs in [G13]. Input: Acoustic feature vectors, one per frame; Output: Phones + space
• Deep bidirectional LSTM networks were used to do phone recognition on TIMIT
• Trained using the Connectionist Temporal Classification (CTC) loss [covered in later class]

RNN-based Acoustic Model

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>WEIGHTS</th>
<th>EPOCHS</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC-3L-500H-TANH</td>
<td>3.7M</td>
<td>107</td>
<td>37.6%</td>
</tr>
<tr>
<td>CTC-1L-250H</td>
<td>0.8M</td>
<td>82</td>
<td>23.9%</td>
</tr>
<tr>
<td>CTC-1L-622H</td>
<td>3.8M</td>
<td>87</td>
<td>23.0%</td>
</tr>
<tr>
<td>CTC-2L-250H</td>
<td>2.3M</td>
<td>55</td>
<td>21.0%</td>
</tr>
<tr>
<td>CTC-3L-421H-UNI</td>
<td>3.8M</td>
<td>115</td>
<td>19.6%</td>
</tr>
<tr>
<td>CTC-3L-250H</td>
<td>3.8M</td>
<td>124</td>
<td>18.6%</td>
</tr>
<tr>
<td>CTC-5L-250H</td>
<td>6.8M</td>
<td>150</td>
<td>18.4%</td>
</tr>
</tbody>
</table>

TIMIMIT phoneme recognition results

So far, we’ve looked at acoustic models…
Next, language models

• Language models
  • provide information about word reordering
    \[ \text{Pr(“she class taught a”) < Pr(“she taught a class”)} \]
  • provide information about the most likely next word
    \[ \text{Pr(“she taught a class”) > Pr(“she taught a speech”)} \]
Application of language models

- Speech recognition

- $\Pr(\text{“she taught a class”}) > \Pr(\text{“sheet or tuck lass”})$

- Machine translation

- Handwriting recognition/Optical character recognition

- Spelling correction of sentences

- Summarization, dialog generation, information retrieval, etc.
Popular Language Modelling Toolkits

- SRILM Toolkit:
  http://www.speech.sri.com/projects/srilm/

- KenLM Toolkit:
  https://kheafield.com/code/kenlm/

- OpenGrm NGram Library:
  http://opengrm.org/
Introduction to probabilistic LMs
Probabilistic or Statistical Language Models

• Given a word sequence, $W = \{w_1, \ldots, w_n\}$, what is $\Pr(W)$?

• Decompose $\Pr(W)$ using the chain rule:

\[
\Pr(w_1, w_2, \ldots, w_n | w_1, \ldots, w_{n-1}) = \Pr(w_1) \Pr(w_2 | w_1) \Pr(w_3 | w_1, w_2) \ldots \Pr(w_n | w_1, \ldots, w_{n-1})
\]

• Sparse data with long word contexts: How do we estimate the probabilities $\Pr(w_n | w_1, \ldots, w_{n-1})$?
Estimating word probabilities

- Accumulate counts of words and word contexts
- Compute normalised counts to get next-word probabilities
- E.g. \( \Pr(\text{“class I she taught a”}) \)

\[
= \frac{\pi(\text{“she taught a class”})}{\pi(\text{“she taught a”})}
\]

where \( \pi(“…”) \) refers to counts derived from a large English text corpus

- What is the obvious limitation here? We’ll never see enough data
Simplifying Markov Assumption

- Markov chain:
  - Limited memory of previous word history: Only last $m$ words are included

- 1-order language model (or bigram model)

\[
\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) \approx \Pr(w_1 \mid <s>) \Pr(w_2 \mid w_1) \Pr(w_3 \mid w_2) \cdots \Pr(w_n \mid w_{n-1})
\]

- 2-order language model (or trigram model)

\[
\Pr(w_1, w_2, \ldots, w_{n-1}, w_n) \approx \Pr(w_2 \mid w_1, <s>) \Pr(w_3 \mid w_1, w_2) \cdots \Pr(w_n \mid w_{n-2}, w_{n-1})
\]

- Ngram model is an $N$-1th order Markov model
Estimating Ngram Probabilities

• Maximum Likelihood Estimates

• Unigram model

\[
Pr_{ML}(w_1) = \frac{\pi(w_1)}{\sum_i \pi(w_i)}
\]

• Bigram model

\[
Pr_{ML}(w_2|w_1) = \frac{\pi(w_1, w_2)}{\sum_i \pi(w_1, w_i)}
\]
The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is $\Pr(\text{"The cat chased a mouse"})$ using a bigram model?

$$\Pr(\text{"<s> The cat chased a mouse </s>"}) =$$

$$\Pr(\text{"The |<s>"}) \cdot \Pr(\text{"cat |The"}) \cdot \Pr(\text{"chased |cat"}) \cdot \Pr(\text{"a |chased"}) \cdot \Pr(\text{"mouse |a"}) \cdot \Pr(\text{"</s> |mouse"}) =$$

$$\frac{3}{3} \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{48}$$
Example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is $\text{Pr}(\text{"The dog eats cheese"})$ using a bigram model?

$$\text{Pr}(\text{"<s> The dog eats cheese </s>"}) =$$

$$\text{Pr}(\text{"The|<s>"}) \cdot \text{Pr}(\text{"dog|The"}) \cdot \text{Pr}(\text{"eats|dog"}) \cdot \text{Pr}(\text{"cheese|eats"}) \cdot \text{Pr}(\text{"</s>|cheese"}) =$$

$$3/3 \cdot 1/3 \cdot 0/1 \cdot 1/1 \cdot 1/1 = 0! \text{ Due to unseen bigrams}$$

How do we deal with unseen bigrams? We’ll come back to it.
Open vs. closed vocabulary task

- Closed vocabulary task: Use a fixed vocabulary, \( V \). We know all the words in advance.

- More realistic setting, we don’t know all the words in advance. Open vocabulary task. Encounter out-of-vocabulary (OOV) words during test time.

- Create an unknown word: \(<\text{UNK}>\)
  - Estimating \(<\text{UNK}>\) probabilities: Determine a vocabulary \( V \). Change all words in the training set not in \( V \) to \(<\text{UNK}>\)
  - Now train its probabilities like a regular word
  - At test time, use \(<\text{UNK}>\) probabilities for words not in training
Evaluating Language Models

- Extrinsic evaluation:
  - To compare Ngram models A and B, use both within a specific speech recognition system (keeping all other components the same)
  - Compare word error rates (WERs) for A and B
  - Time-consuming process!
Intrinsic Evaluation

- Evaluate the language model in a standalone manner
- How likely does the model consider the text in a test set?
- How closely does the model approximate the actual (test set) distribution?
- Same measure can be used to address both questions — perplexity!
Measures of LM quality

- How likely does the model consider the text in a test set?

- How closely does the model approximate the actual (test set) distribution?

- Same measure can be used to address both questions — perplexity!
Perplexity (I)

- How likely does the model consider the text in a test set?

- Perplexity(test) = \(1/Pr_{model}[text]\)

- Normalized by text length:

- Perplexity(test) = \((1/Pr_{model}[text])^{1/N}\) where \(N = \) number of tokens in test

- e.g. If model predicts i.i.d. words from a dictionary of size \(L\), per word perplexity = \((1/(1/L)^N)^{1/N} = L\)
Intuition for Perplexity

• Shannon’s guessing game builds intuition for perplexity

• What is the surprisal factor in predicting the next word?

• At the stall, I had tea and _________  
  biscuits 0.1  
  samosa 0.1  
  coffee 0.01  
  rice 0.001  
  but 0.000000000001

• A better language model would assign a higher probability to the actual word that fills the blank (and hence lead to lesser surprisal/perplexity)
Measures of LM quality

- How likely does the model consider the text in a test set?

- How closely does the model approximate the actual (test set) distribution?

- Same measure can be used to address both questions — perplexity!
Perplexity (II)

- How closely does the model approximate the actual (test set) distribution?
- KL-divergence between two distributions $X$ and $Y$
  \[ D_{KL}(X||Y) = \sum_{\sigma} \Pr_X[\sigma] \log (\Pr_X[\sigma]/\Pr_Y[\sigma]) \]
  - Equals zero iff $X = Y$; Otherwise, positive

- How to measure $D_{KL}(X||Y)$? We don’t know $X$!
  \[ D_{KL}(X||Y) = \sum_{\sigma} \Pr_X[\sigma] \log(1/\Pr_Y[\sigma]) - H(X) \]
  where $H(X) = -\sum_{\sigma} \Pr_X[\sigma] \log \Pr_X[\sigma]$

- Empirical cross entropy:
  \[ \frac{1}{|test|} \sum_{\sigma \in test} \log \left( \frac{1}{\Pr_y[\sigma]} \right) \]
Perplexity vs. Empirical Cross Entropy

- **Empirical Cross Entropy (ECE)**

\[
\frac{1}{|\text{#sents}|} \sum_{\sigma \in \text{test}} \log \left( \frac{1}{\text{Pr}_{\text{model}}[\sigma]} \right)
\]

- **Normalized Empirical Cross Entropy**

\[
\frac{1}{|\text{#words}/\text{#sents}|} \frac{1}{|\text{#sents}|} \sum_{\sigma \in \text{test}} \log \left( \frac{1}{\text{Pr}_{\text{model}}[\sigma]} \right)
\]

\[= \frac{1}{N} \sum_{\sigma} \log \left( \frac{1}{\text{Pr}_{\text{model}}[\sigma]} \right)
\]

where \(N = \text{#words}\)

- How does \(\frac{1}{N} \sum_{\sigma} \log \left( \frac{1}{\text{Pr}_{\text{model}}[\sigma]} \right)\) relate to perplexity?
Perplexity vs. Empirical Cross Entropy

\[
\log(\text{perplexity}) = \frac{1}{N} \log \frac{1}{\Pr[\text{test}]}
\]

\[
= \frac{1}{N} \log \prod_{\sigma} \left( \frac{1}{\Pr_{\text{model}}[\sigma]} \right)
\]

\[
= \frac{1}{N} \sum_{\sigma} \log \left( \frac{1}{\Pr_{\text{model}}[\sigma]} \right)
\]

Thus, perplexity = \(\exp\) (normalized cross entropy)

Example perplexities for Ngram models trained on WSJ (80M words):

Unigram: 962, Bigram: 170, Trigram: 109
Introduction to smoothing of LMs
Recall example

The dog chased a cat
The cat chased away a mouse
The mouse eats cheese

What is \( \Pr(\text{The dog eats cheese}) \)?

\[
\Pr(\text{<s> The dog eats cheese </s>}) = \\
\Pr(\text{The|<s>}) \cdot \Pr(\text{dog|The}) \cdot \Pr(\text{eats|dog}) \cdot \Pr(\text{cheese|eats}) \cdot \Pr(\text{</s>|cheese}) = \\
3/3 \cdot 1/3 \cdot 0/1 \cdot 1/1 \cdot 1/1 = 0! \text{ Due to unseen bigrams}
\]
Unseen Ngrams

- Even with MLE estimates based on counts from large text corpora, there will be many unseen bigrams/trigrams that never appear in the corpus

- If any unseen Ngram appears in a test sentence, the sentence will be assigned probability 0

- Problem with MLE estimates: maximises the likelihood of the observed data by assuming anything unseen cannot happen and overfits to the training data

  - Smoothing methods: Reserve some probability mass to Ngrams that don’t occur in the training corpus
Add-one (Laplace) smoothing

Simple idea: Add one to all bigram counts. That means,

\[
\text{Pr}_{ML}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i)}{\pi(w_{i-1})}
\]

becomes

\[
\text{Pr}_{Lap}(w_i|w_{i-1}) = \frac{\pi(w_{i-1}, w_i) + 1}{\pi(w_{i-1}) + V}
\]

where \( V \) is the vocabulary size