Language Modeling

[1] A Neural Probabilistic Language Model
[2] Recurrent Neural Network Based Language Model
Overview

- Language Modeling
- N-gram Models
- A Neural Probabilistic Language Model
- Recurrent Neural Network Based Language Model
Language Modeling

• Language model: Estimated the probability of word sequence
  • Word sequence: $w_1$, $w_2$, $w_3$, ..., $w_t$
  • $P(w_1, w_2, w_3, ..., w_t)$

• Application: speech recognition
  • Different word sequence can have the same pronunciation

recognize speech or wreck a nice beach

• Application: sentence generation

If $P(\text{recognize speech}) > P(\text{wreck a nice beach})$

Output = “recognize speech”
N-gram Models

• How to estimate $P(w_1, w_2, \ldots, w_{t-1}, w_t)$
• Collect a large amount of text data as training data
  • However, the word sequence $w_1, w_2, \ldots, w_t$ may not appear in the training data
• N-gram language model: $P(w_1, w_2, \ldots, w_{t-1}, w_t) = \prod_{t=1}^{T} P(w_t | w_1, w_2, \ldots, w_{t-1})$
  • Learn joint likelihood of training sentences under Markov assumption using n-grams
    $$P(w_1, w_2, \ldots, w_{t-1}, w_t) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}, w_{t-n+2}, \ldots, w_{t-1})$$
  • Calculate from n-gram frequency counts
    $$P(w_t | w_{t-n+1}, \ldots, w_{t-1}) = \frac{\text{count}(w_{t-(n-1)}, \ldots, w_{t-1}, w_t)}{\text{count}(w_{t-(n-1)}, \ldots, w_{t-1})}$$
N-gram Models

• i.e. $P(\text{“wreck a nice beach”})$
  
  $= P(\text{wreck} | \text{START}) P(\text{a} | \text{wreck}) P(\text{nice} | \text{a}) P(\text{beach} | \text{nice})$

  • Estimate $P(\text{beach} | \text{nice})$ from training data

  $$P(\text{beach} | \text{nice}) = \frac{C(\text{nice beach})}{C(\text{nice})}$$

  • It is easy to generalize to 3-gram, 4-gram ......
Limitations of N-gram Models

• It is not taking into account contexts farther than 1 or 2 words
• It is not taking into account the similarity between words
  • “The cat is walking in the bedroom” (training corpus)
  • “A dog was running in a room” (?)
• Curse of Dimensionality
A Neural Probabilistic Language Model (Bengio et al, JMLR 2003)

• Motivation:
  • LM does not take into account contexts farther than 2 words.
  • LM does not take into account the “similarity” between words.

• Idea:
  • A word $w$ is associated with a distributed feature vector (a real-valued vector in $\mathbb{R}^n$ $n$ is much smaller than size of the vocabulary)
  • Express joint probability function $f$ of words sequence in terms of feature vectors
  • Learn simultaneously the word feature vector and the probability function $f$
A Neural Probabilistic Language Model

• Why does it work?
  • If we knew that “dog” and “cat” played similar roles (semantically and syntactically), and similarly for (the, a), (bedroom, room), (is, was), (running, walking), we could naturally generalize from
    • The cat is walking in the bedroom
  • to and likewise to
    • A dog was running in a room
    • The cat is running in a room
    • A dog is walking in a bedroom
    • ....
A Neural Probabilistic Language Model

• Denotations
  • The training set is a sequence $w_1 \ldots w_t$ of words $w_t \in V$, where the vocabulary $V$ is a large but finite set
  • The objective is to learn a good model as below, in the sense that it gives high out-out-sample likelihood
    $$ f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t|w_1^{t-1}) $$
  • The only constraint on model is that for any choice of $w_1^{t-1}$, the sum
    $$ \sum_{i=1}^{|V|} f(i, w_{t-1}, \ldots, w_{t-n+1}) = 1 $$
A Neural Probabilistic Language Model

• Model
  • We decompose the function \( f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t | w_{1}^{t-1}) \) in two parts
    • A mapping \( C \) from any element \( i \) of \( V \) to a real vector \( C(i) \in \mathbb{R}^m \). It represents the distributed feature vectors associated with each word in the vocabulary
  • The probability function over words, expressed with \( C \): a function \( g \) maps an input sequence of feature vectors for words in context, \( (C(w_{t-n+1}), \ldots, C(w_{t-1})) \), to a conditional probability distribution over words in \( V \) for the next word. The output of \( g \) is a vector whose \( i \)-th element estimates the probability
    \[
    f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))
    \]
A Neural Probabilistic Language Model

Figure 1: Neural architecture: \( f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1})) \) where \( g \) is the neural network and \( C(i) \) is the \( i \)-th word feature vector.
A Neural Probabilistic Language Model

• softmax output layer:
  \[ P(w_t|w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{y_{wt}}}{\sum_i e^{y_i}} \]

• \( y_i \) unnormalized log-probabilities for each output word \( i \)
  \[ y = b + Utanh(d + Hx) \]

• \( x \) is the word features layer activation vector
  \[ x = (C(w_{t-1}), \ldots, C(w_{t-n+1})) \]

• The free parameters of the model are:
  \( \theta = (b, d, W, U, H, C) \)

  • \( b \) output biases (|\( V \)|)
  • \( d \) the hidden layer biases (\( h \))
  • \( U \) the hidden-to-output weights (|\( V \)|\( \times h \))
  • \( H \) the hidden layer weights (\( h \times (n - 1)m \))
  • \( C \) word features (|\( V \)|\( \times m \))

4 weeks of training (40 CPUs) on 14,000,000 words training set
|\( V \)|=17964
A Neural Probabilistic Language Model

- Objective function
  - Training is achieved by looking for $\theta$ that maximizes the training corpus penalized log-likelihood, where $R(\theta)$ is a regularization term:

  $$L = \frac{1}{T} \sum_{t} \log f(w_t, \ldots, w_{t-n+1}; \theta) + R(\theta)$$

- Stochastic gradient ascent

  $$\theta = (b, d, W, U, H, C)$$

  $$\theta \leftarrow \theta + \varepsilon \frac{\partial \log P(w_t|w_{t-1}, \ldots, w_{t-n+1})}{\partial \theta}$$

- Note that a large fraction of the parameters needs not be updated or visited after each example: the word feature $C(j)$ of all words $j$ that do not occur in the input window
A Neural Probabilistic Language Model

<table>
<thead>
<tr>
<th>Model</th>
<th>n</th>
<th>h</th>
<th>m</th>
<th>direct</th>
<th>mix</th>
<th>train.</th>
<th>valid.</th>
<th>test.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP1</td>
<td>5</td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>182</td>
<td>284</td>
<td>268</td>
</tr>
<tr>
<td>MLP2</td>
<td>5</td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td>275</td>
<td>257</td>
<td></td>
</tr>
<tr>
<td>MLP3</td>
<td>5</td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>201</td>
<td>327</td>
<td>310</td>
</tr>
<tr>
<td>MLP4</td>
<td>5</td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td>286</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>MLP5</td>
<td>5</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td>209</td>
<td>296</td>
<td>279</td>
</tr>
<tr>
<td>MLP6</td>
<td>5</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td>273</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>MLP7</td>
<td>3</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td>210</td>
<td>309</td>
<td>293</td>
</tr>
<tr>
<td>MLP8</td>
<td>3</td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td>284</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>MLP9</td>
<td>5</td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>no</td>
<td>175</td>
<td>280</td>
<td>276</td>
</tr>
<tr>
<td>MLP10</td>
<td>5</td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>yes</td>
<td>265</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Del. Int.</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>352</td>
<td>336</td>
</tr>
<tr>
<td>Kneser-Ney back-off</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>334</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>Kneser-Ney back-off</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td>348</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td>354</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td>326</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td>335</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>3</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td>343</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>4</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>class-based back-off</td>
<td>5</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
</tbody>
</table>

Perplexity: the lower the better

\[
ppx = \exp \left( -\frac{1}{T} \log \prod_{t=1}^{T} P(w_t | w_{t-1}^{t-1}) \right)
\]
A Neural Probabilistic Language Model

http://metaoptimize.com/projects/wordreprs/
Recurrent Neural Network Based Language Model (Mikolov et al, INTERSPEECH 2010)

• Motivation
  • A major deficiency of Bengio’s approach is that a feedforward network has to use **fixed length** context that needs to be specified *ad hoc* before training. Usually this means that neural networks see only five to ten preceding words when predicting the next one. It is well known that humans can exploit longer context with great success.

  • Use RNN model to encode temporal information implicitly for contexts with **arbitrary lengths**.
Recurrent Neural Network Based Language Model

- $x(t) = w(t) + s(t - 1)$
  - Input layer $x$
- $s_j(t) = f \left( \sum_i x_i(t) u_{ji} \right)$
  - hidden layer $s$
  - $f(z)$ is sigmoid activation function
  - $f(z) = \frac{1}{1+e^{-z}}$
- $y_k(t) = g \left( \sum_j s_j(t) v_{kj} \right)$
  - output layer $y$
  - $g(z)$ is softmax function
  - $g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$

Figure 1: Simple recurrent neural network.
Recurrent Neural Network Based Language Model

• Optimization
  • to improve performance, we merge all words that occur less often than a threshold into a special rare token <rare>.
  
  \[ P(w_i(t + 1)|w(t), s(t - 1)) = \begin{cases} 
  \frac{y_{\text{rare}}(t)}{C_{\text{rare}}} & \text{if } w_i(t+1) \text{ is rare}, \\
  y_i(t) & \text{otherwise} \end{cases} \]

  • where \( C_{\text{rare}} \) is number of words in vocabulary that occur less often than the threshold.
  • All rare words are treated equally.
Recurrent Neural Network Based Language Model

• Recurrent vs feedforward neural networks
  • In feedforward networks, history is represented by context of $N - 1$ words - it is limited in the same way as in N-gram backoff models.
  • In recurrent networks, history is represented by neurons with recurrent connections - history length is unlimited.
  • Also, recurrent networks can learn to compress whole history in low dimensional space, while feedforward networks compress (project) just single word.
  • Recurrent networks have possibility to form short term memory, so they can better deal with position invariance; feedforward networks cannot do that.
Recurrent Neural Network Based Language Model

• Experiements 1 – WSJ
  • Linear interpolation: $0.25 \times \text{KN5} + 0.75 \times \text{RNN}$

• Smaller perplexity, Less error rate

Table 1: Performance of models on WSJ DEV set when increasing size of training data.

<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>
Recurrent Neural Network Based Language Model

• Experiments 2 – RNN params
  • Dynamic model: Continue learning parameters from the test data

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RNN</td>
<td>RNN+KN</td>
</tr>
<tr>
<td>KN5 - baseline</td>
<td>-</td>
<td>221</td>
</tr>
<tr>
<td>RNN 60/20</td>
<td>229</td>
<td>186</td>
</tr>
<tr>
<td>RNN 90/10</td>
<td>202</td>
<td>173</td>
</tr>
<tr>
<td>RNN 250/5</td>
<td>173</td>
<td>155</td>
</tr>
<tr>
<td>RNN 250/2</td>
<td>176</td>
<td>156</td>
</tr>
<tr>
<td>RNN 400/10</td>
<td>171</td>
<td>152</td>
</tr>
<tr>
<td>3xRNN static</td>
<td>151</td>
<td>143</td>
</tr>
<tr>
<td>3xRNN dynamic</td>
<td>128</td>
<td>121</td>
</tr>
</tbody>
</table>
Recurrent Neural Network Based Language Model

- Experiments 3 – data size
  - RNN – 5.4M
  - Back-off: 1.3G

<table>
<thead>
<tr>
<th>Model</th>
<th>WER static</th>
<th>WER dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT05 LM</td>
<td>24.5</td>
<td>-</td>
</tr>
<tr>
<td>RT09 LM - baseline</td>
<td>24.1</td>
<td>-</td>
</tr>
<tr>
<td>KN5 in-domain</td>
<td>25.7</td>
<td>-</td>
</tr>
<tr>
<td>RNN 500/10 in-domain</td>
<td>24.2</td>
<td>24.1</td>
</tr>
<tr>
<td>RNN 500/10 + RT09 LM</td>
<td>23.3</td>
<td>23.2</td>
</tr>
<tr>
<td>RNN 800/10 in-domain</td>
<td>24.3</td>
<td>23.8</td>
</tr>
<tr>
<td>RNN 800/10 + RT09 LM</td>
<td>23.4</td>
<td>23.1</td>
</tr>
<tr>
<td>RNN 1000/5 in-domain</td>
<td>24.2</td>
<td>23.7</td>
</tr>
<tr>
<td>RNN 1000/5 + RT09 LM</td>
<td>23.4</td>
<td>22.9</td>
</tr>
<tr>
<td>3xRNN + RT09 LM</td>
<td><strong>23.3</strong></td>
<td><strong>22.8</strong></td>
</tr>
</tbody>
</table>
Recurrent Neural Network Based Language Model

• Conclusion
  • Arbitrary-length context from the past
  • Outperformance on various tasks with less data
  • Need of improvement on capturing truly long context
Thanks

• Q&A