Language Modeling

Instructor: Thien Huu Nguyen

Based on slides from: Chris Manning
Language Modeling

- Language Modeling is the task of predicting what word comes next given a sequence of previous words. the students opened their _______

- More formally, given a sequence of words $x_1, x_2, ..., x_i$, compute the probability distribution of the following word:

$$P(x_{i+1}|x_i, x_{i-1}, ..., x_1)$$

where $x_{i+1}$ can be any word in the vocabulary $V = \{w_1, ..., w_{|V|}\}$

- A system that can do this is called a language model
Language Modeling

• A language model can also be viewed as a system that assigns probability to a piece of text (i.e., estimating how likely the piece of text is).

• For instance, given the text $x_1, x_2, \ldots, x_N$, the probability of this text can be computed based on the probabilities from the language model by:

$$P(x_1, x_2, \ldots, x_N) = P(x_1)P(x_2|x_1) \ldots P(x_N|x_{N-1}, \ldots, x_1)$$

$$= \prod_{i=1}^{N} P(x_i|x_{i-1}, \ldots, x_1)$$
Why should we care about language modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language.

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.
Language Modeling

• Remember this?

Google

- how
  - how i met your mother
  - how are you
  - how old are you
  - how to train your dragon 3
  - how are you doing
  - how deep is your love
  - how to basic
  - how to train your dragon
  - how i met your mother season 5
  - how i met your mother season 2
Language Model

• Remember this?

using

using essential oils safely
using giveaway
using linkedin
using innovative products
using technology
using hydraulic equipment
using social media
using
See all results for using
n-gram language models

*the students opened their _______*

- **Question**: How do we learn a language model?
- **Answer** (before deep learning): use n-gram language model

- **Definition**: a n-gram is a sequence of $n$ consecutive words
  - unigrams: “the”, “students”, “opened”, “their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- **Main idea**: collect the statistics about the frequency of the n-gram in some corpus, and use it to estimate the probability
n-gram language models

• The simplifying assumption: \( x_i \) only depends on the \( n - 1 \) preceding words, i.e.,

\[
P(x_i \mid x_{i-1}, \ldots, x_1) = P(x_i \mid x_{i-1}, \ldots, x_{i-n+1}) \quad \text{(assumption)}
\]

\[
= \frac{P(x_{i-n+1}, x_{i-n}, \ldots, x_{i-1})}{P(x_{i-n+1}, x_{i-n}, \ldots, x_i)}
\]

• Question: how to obtain the probabilities of such n-grams?
• Answer: by counting their appearance in some large corpus of text:

\[
= \frac{\text{count}(x_{i-n+1}, x_{i-n}, \ldots, x_{i-1})}{\text{count}(x_{i-n+1}, x_{i-n}, \ldots, x_i)} \quad \text{statistical approximation}
\]
n-gram language models: example

• Suppose we are learning a 4-gram language model
  as the proctor started the clock, the students opened their ______
  discard condition on this

• For example, suppose that in the corpus:
  – “students opened their” occurred 1000 times
  – “students opened their books” occurred 400 times
    \[ P(books \mid \text{students opened their}) = 0.4 \]
  – “students opened their exams” occurred 100 times
    \[ P(exams \mid \text{students opened their}) = 0.1 \]

Should we have discarded the “proctor” context?
Sparsity problems with n-gram language models

- using larger n-grams makes predictions more accurate but causes the sparsity problems

### Sparsity Problem 1

**Problem:** What if “students opened their w” never occurred in data? Then w has probability 0!

**Solution:** Add small δ to the count for every w ∈ V. This is called smoothing.

\[
P(w|\text{students opened their}) = \frac{\text{count(\text{students opened their } w)}}{\text{count(\text{students opened their})}}
\]

### Sparsity Problem 2

**Problem:** What if “students opened their” never occurred in data? Then we can’t calculate probability for any w!

**Solution:** Just condition on “opened their” instead. This is called backoff.

**Note:** Increasing n makes sparsity problems worse. Typically we can’t have n bigger than 5.
Back-off

- In given corpus, we may have never seen
  - Scottish beer drinkers
  - Scottish beer eaters

- We may have seen “beer drinker” more often than “beer eater”
- Better: back-off to bigram

Smoothing gives same probability
Interpolation

• Higher and lower order n-gram models have different strengths and weaknesses
  – high-order n-grams are sensitive to more context, but have sparse counts
  – low-order n-grams consider only very limited context, but have robust counts

• Combine them:

\[ p_I(w_3|w_1, w_2) = \lambda_1 p_1(w_3) \times \lambda_2 p_2(w_3|w_2) \times \lambda_3 p_3(w_3|w_1, w_2) \]
Storage problems with n-gram language models

**Storage**: Need to store count for all n-grams you saw in the corpus.

\[
P(w|\text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}
\]

Increasing nor increasing corpus increases model size!
n-gram language models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop.

```
today the _______
```

generate probability distribution

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>company</td>
<td>0.153</td>
</tr>
<tr>
<td>bank</td>
<td>0.153</td>
</tr>
<tr>
<td>price</td>
<td>0.077</td>
</tr>
<tr>
<td>italian</td>
<td>0.039</td>
</tr>
<tr>
<td>emirate</td>
<td>0.039</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Sparsity problem: not much granularity in the probability distribution.
Generating text with a n-gram language model

- You can also use a Language Model to generate text.

```
today the ______
```

Condition on this:

- Get probability distribution:
  - company: 0.153
  - bank: 0.153
  - price: 0.077
  - italian: 0.039
  - emirate: 0.039
  - ...

Sample:
Generating text with a n-gram language model

- You can also use a Language Model to generate text.

```
today the price_______
```

condition on this

get probability distribution

<table>
<thead>
<tr>
<th>of</th>
<th>0.308</th>
</tr>
</thead>
<tbody>
<tr>
<td>for</td>
<td>0.050</td>
</tr>
<tr>
<td>it</td>
<td>0.046</td>
</tr>
<tr>
<td>to</td>
<td>0.046</td>
</tr>
<tr>
<td>is</td>
<td>0.031</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Generating text with a n-gram language model

- You can also use a Language Model to generate text.

\[ today \ the \ price \ of \ \underline{\text{______}} \]

condition on this

get probability distribution

<table>
<thead>
<tr>
<th>Word</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0.072</td>
</tr>
<tr>
<td>18</td>
<td>0.043</td>
</tr>
<tr>
<td>oil</td>
<td>0.043</td>
</tr>
<tr>
<td>its</td>
<td>0.036</td>
</tr>
<tr>
<td>gold</td>
<td>0.018</td>
</tr>
</tbody>
</table>

...
Generating text with a n-gram language model

- You can also use a Language Model to generate text.

  today the price of gold per ton, while production of shoe lasts and shoe industry, the bank intervened just after it considered and rejected an imf demand to rebuild depleted European stocks, sept 30 end primary 76 cts a share.

  Surprisingly grammatical!

- ...but incoherent. We need to consider more than three words at a time if we want to model language well.

- But increasing n worsens sparsity problem, and increases model size...
Neural language model

• Recall:
  – Given the previous words $x_{i-1}, \ldots, x_1$
  – We need to compute $P(x_i|x_{i-1}, \ldots, x_1)$

• How about a window-based neural model?

as the proctor started the clock, the students opened their ______

discard

fixed-window
A fixed-window neural language model

\[ \hat{y}_4 = P(x_5 | \text{the students opened their}) \]

Output distribution
\[ y_i = \text{softmax}(U h + b_2) \]

Hidden layer
\[ h = f(W e + b_1) \]

Concatenated word embeddings:
\[ e = [e_1, e_2, e_3, e_4] \]

Word embeddings

Words/one-hot vectors
\[ x_i \in \{0,1\}^{|V|} \]
A fixed-window neural language model

\[ \hat{y}_4 = P(x_5 | \text{the students opened their}) \]

- **Improvements over n-gram LM:**
  - No sparsity problem
  - Don’t need to store all observed n-grams

- **Remaining problems**
  - Fixed window is too small
  - Enlarging window enlarges \( W \)
  - Window can never be large enough!
  - \( x_1 \) and \( x_2 \) are multiplied by completely different weights in \( W \).
  - No symmetry in how the inputs are processed.

We need a neural architecture that can process any length input.
A RNN language model

output distribution
\[ y_i = \text{softmax}(U h_t + b_2) \]

hidden states
\[ h_i = \sigma(W_h h_{i-1} + W_e e_i + b_1) \]
h_0 is the initial hidden state
using LSTM or GRU is more common

\[ \hat{y}_4 = P(x_5 | \text{the students opened their}) \]

word embeddings
\[ e_i = E x_i \]

words/one-hot vectors
\[ x_i \in \{0,1\}^{|V|} \]
A RNN language model

\[ \hat{y}_4 = P(x_5 | \text{the students opened their}) \]

- **RNN advantages:**
  - Can process any length input
  - Computation for step \( i \) can (in theory) use information from many steps back
  - Model size doesn’t increase for longer input
  - Same weights applied on every timestep, so there is symmetry in how inputs are processed.

- **RNN disadvantages:**
  - Recurrent computation is slow
  - In practice, difficult to access information from many steps back (although LSTM can help a bit)
Training a RNN language model

• Get a big corpus of text which is a sequence of words \( x_1, ..., x_T \)
• Feed into RNN-LM; compute output distribution \( \hat{y}_i \) for every step \( i \)
  — i.e., predict probability distribution of every word, given words so far

• Loss function on step \( i \) is cross-entropy between predicted probability distribution \( \hat{y}_i \), and the true next word \( y_i \) (one-hot distribution for \( x_{i+1} \)):

\[
J_i(\theta) = CE(y_i, \hat{y}_i) = - \sum_{w \in V} y_i(w) \log \hat{y}_i(w) = - \log \hat{y}_i(x_{i+1})
\]

• Average this to get overall loss for entire training set:

\[
J(\theta) = \frac{1}{T} \sum_{i=1}^{T} J_i(\theta) = \frac{1}{T} \sum_{i=1}^{T} - \log \hat{y}_i(x_{i+1})
\]
A RNN language model

Loss

Predicted prob

dists

Corpus

the

\( x_1 \)

students

\( x_2 \)

opened

\( x_3 \)

their

\( x_4 \)

exams

Loss = \( J_1(\theta) \) = negative log probability of “students”

The diagram shows a recurrent neural network (RNN) model for language prediction. The RNN architecture includes hidden states \( h_t \) and input states \( e_t \) connected through weight matrices \( W_h \) and \( W_e \). The predicted probabilities \( \hat{y}_t \) are generated from the hidden states. The loss function for each prediction step is the negative log probability of the correct word, contributing to the overall loss \( J_1(\theta) \).
A RNN language model

Loss: $J_1(\theta)$

Predicted prob dists: $J_2(\theta) = -\text{negative log probability of “opened”}$

$J_3(\theta)$

$J_4(\theta)$

$h_0$

$w_h$

$h_1$

$y_1$

$U$

$h_2$

$y_2$

$U$

$h_3$

$y_3$

$U$

$h_4$

$y_4$

$U$

$\cdots$

$e_1$

$e_2$

$e_3$

$e_4$

$W_e$

$W_e$

$W_e$

$W_e$

$E$

$E$

$E$

$E$

Corpus

the $x_1$

students $x_2$

opened $x_3$

their $x_4$

exams

CIS 410/510: Natural Language Processing
A RNN language model

Loss → $J_1(\theta) → J_2(\theta) → J_3(\theta) → J_4(\theta) → U$

Predicted prob dists

$h_0 \rightarrow W_h \rightarrow U \rightarrow h_1 \rightarrow W_h \rightarrow U \rightarrow h_2 \rightarrow W_h \rightarrow U \rightarrow h_3 \rightarrow W_h \rightarrow U \rightarrow h_4 \rightarrow W_h \rightarrow \ldots$

$\bar{y}_1 \rightarrow W_e \rightarrow e_1 \rightarrow E \rightarrow \text{the } x_1$

$\bar{y}_2 \rightarrow W_e \rightarrow e_2 \rightarrow E \rightarrow \text{students } x_2$

$\bar{y}_3 \rightarrow W_e \rightarrow e_3 \rightarrow E \rightarrow \text{opened } x_3$

$\bar{y}_4 \rightarrow W_e \rightarrow e_4 \rightarrow E \rightarrow \text{their } x_4 \rightarrow \text{exams}$

$\text{=negative log probability of “their”}$

Corpus
A RNN language model

![Diagram of RNN language model]

- **Corpus**
  - the
  - students
  - opened
  - their

- **Predicted prob dists**
  - $J_1(\theta)$
  - $J_2(\theta)$
  - $J_3(\theta)$
  - $J_4(\theta)$

- **Loss**

- **Equation**
  
  

= negative log probability of “exams”
A RNN language model

\[
J_1(\theta) + J_2(\theta) + J_3(\theta) + J_4(\theta) + \ldots = J(\theta) = \frac{1}{T} \sum_{i=1}^{T} J_i(\theta)
\]
Training a RNN language model

- However: Computing loss and gradients across entire corpus \( x_1, \ldots, x_T \) is too expensive!

\[
J(\theta) = \frac{1}{T} \sum_{i=1}^{T} J_i(\theta)
\]

- In practice, consider \( x_1, \ldots, x_T \) as a sentence (or a document)

- Using Stochastic Gradient Descent: compute the loss \( J(\theta) \) and gradients for a sentence (usually a batch of sentences), update the weights. Repeat.
Generating text with a RNN language model

• Just like a n-gram Language Model, you can use a RNN language model to generate text by repeated sampling. Sampled output is next step’s input.
Generating text with a RNN language model

- You can train a RNN-LM on any kind of text, then generate text in that style.
- N-LM trained on Obama speeches:

**SEED: Democracy**

*Good morning. One of the borders will be able to continue to be here today. We have to say that the partnership was a partnership with the American people and the street continually progress that is a process and distant lasting peace and support that they were supporting the work of concern in the world. They were in the streets and communities that could have to provide steps to the people of the United States and Afghanistan. In the streets — the final decade of the country that will include the people of the United States of America. Now, humanitarian crisis has already rightly achieved the first American future in the same financial crisis that they can find reason to invest in the world.*

*Thank you very much. God bless you. God bless you. Thank you.*

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0
Generating text with a RNN language model

• You can train a RNN-LM on any kind of text, then generate text in that style.
• N-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.
Generating text with a RNN language model

• You can train a RNN-LM on any kind of text, then generate text in that style.
• N-LM trained on Harry Potter:

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.
Generating text with a RNN language model

- You can train a RNN-LM on any kind of text, then generate text in that style.
- N-LM trained on recipes:

  Title: CHOCOLATE RANCH BARBECUE
  Categories: Game, Casseroles, Cookies, Cookies
  Yield: 6 Servings

  2 tb Parmesan cheese -- chopped
  1 c  Coconut milk
  3    Eggs, beaten

  Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

  Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.

https://gist.github.com/nylki/1efbaa36635956d35bcc
Generating text with a RNN language model

• You can train a RNN-LM on any kind of text, then generate text in that style.
• N-LM trained on paint color names:

Evaluating language models

- The standard evaluation metric for Language Models is perplexity.

\[
\text{perplexity} = \prod_{i=1}^{T} \left( \frac{1}{P_{LM}(x_{i+1}|x_i, \ldots, x_1)} \right)^{1/T}
\]

Inverse probability of corpus, according to language model

- This is equal to the exponential of the cross-entropy loss \( J(\theta) \)

\[
\prod_{i=1}^{T} \left( \frac{1}{\hat{y}_i(x_{i+1})} \right)^{1/T} = \exp \left( \frac{1}{T} \sum_{i=1}^{T} - \log \hat{y}_i(x_{i+1}) \right) = \exp(J(\theta))
\]

Lower perplexity is better!
RNNs have greatly improved perplexity

<table>
<thead>
<tr>
<th>Model</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)</td>
<td>67.6</td>
</tr>
<tr>
<td>RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)</td>
<td>51.3</td>
</tr>
<tr>
<td>RNN-2048 + BlackOut sampling (Ji et al., 2015)</td>
<td>68.3</td>
</tr>
<tr>
<td>Sparse Non-negative Matrix factorization (Shazeer et al., 2015)</td>
<td>52.9</td>
</tr>
<tr>
<td>LSTM-2048 (Jozefowicz et al., 2016)</td>
<td>43.7</td>
</tr>
<tr>
<td>2-layer LSTM-8192 (Jozefowicz et al., 2016)</td>
<td>30.0</td>
</tr>
<tr>
<td><strong>Ours small</strong> (LSTM-2048)</td>
<td>43.9</td>
</tr>
<tr>
<td><strong>Ours large</strong> (2-layer LSTM-2048)</td>
<td>39.8</td>
</tr>
</tbody>
</table>

Perplexity improves (lower is better)

https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/