Text Generative Models

CSCI 699
Instructor: Xiang Ren
USC Computer Science
Language Modeling

CSCI 699
Instructor: Xiang Ren
USC Computer Science
Are These Sentences OK?

• Jane went to the store.
• store to Jane went the.
• Jane went store.
• Jane goed to the store.
• The store went to Jane.
• The food truck went to Jane.
Calculating the Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(x_i \mid x_1, \ldots, x_{i-1})$$

Next Word  Context
Calculating the Probability of a Sentence

\[ P(X) = \prod_{i=1}^{I} P(x_i | x_1, \ldots, x_{i-1}) \]

The big problem: How do we predict

\[ P(x_i | x_1, \ldots, x_{i-1}) \]
Review: Count-based Language Models
Count-based Language Models

- Count up the frequency and divide:
  \[ P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})} \]

- Add smoothing, to deal with zero counts:
  \[ P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) = \lambda P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) + (1 - \lambda) P(x_i \mid x_{1-n+2}, \ldots, x_{i-1}) \]

- Modified Kneser-Ney smoothing
A Refresher on Evaluation

- **Log-likelihood:**
  \[
  LL(\mathcal{E}_{\text{test}}) = \sum_{E \in \mathcal{E}_{\text{test}}} \log P(E)
  \]

- **Per-word Log Likelihood:**
  \[
  \text{WLL}(\mathcal{E}_{\text{test}}) = \frac{1}{\sum_{E \in \mathcal{E}_{\text{test}}} |E|} \sum_{E \in \mathcal{E}_{\text{test}}} \log P(E)
  \]

- **Per-word (Cross) Entropy:**
  \[
  H(\mathcal{E}_{\text{test}}) = \frac{1}{\sum_{E \in \mathcal{E}_{\text{test}}} |E|} \sum_{E \in \mathcal{E}_{\text{test}}} - \log_2 P(E)
  \]

- **Perplexity:**
  \[
  \text{ppl}(\mathcal{E}_{\text{test}}) = 2^H(\mathcal{E}_{\text{test}}) = e^{-\text{WLL}(\mathcal{E}_{\text{test}})}
  \]
What Can we Do w/ LMs?

• Score sentences:

  Jane went to the store . → high
  store to Jane went the . → low

  (same as calculating loss for training)
What Can we Do w/ LMs?

• Score sentences:

  Jane went to the store . → high
  store to Jane went the . → low

  (same as calculating loss for training)

• Generate sentences:

  while didn’t choose end-of-sentence symbol:
    calculate probability
    sample a new word from the probability distribution
Problems and Solutions?

• Cannot share strength among similar words

  she bought a car  she bought a bicycle
  she purchased a car  she purchased a bicycle

→ solution: class based language models
Problems and Solutions?

• Cannot share strength among **similar words**
  
  | she bought a car | she bought a bicycle |
  | she purchased a car | she purchased a bicycle |

  → solution: class based language models

• Cannot condition on context with **intervening words**
  
  | Dr. Jane Smith | Dr. Gertrude Smith |

  → solution: skip-gram language models
Problems and Solutions?

• Cannot share strength among similar words
  
  she bought a car  she bought a bicycle  
  she purchased a car  she purchased a bicycle
  
  → solution: class based language models

• Cannot condition on context with intervening words
  
  Dr. Jane Smith  Dr. Gertrude Smith
  
  → solution: skip-gram language models

• Cannot handle long-distance dependencies
  
  for tennis class he wanted to buy his own racquet
  for programming class he wanted to buy his own computer
  
  → solution: cache, trigger, topic, syntactic models, etc.
An Alternative: Featurized Log-Linear Models
An Alternative: Featurized Models

- Calculate features of the context
An Alternative: Featurized Models

- Calculate features of the context
- Based on the features, calculate probabilities
An Alternative: Featurized Models

- Calculate features of the context
- Based on the features, calculate probabilities
- Optimize feature weights using gradient descent, etc.
Example:

Previous words: “giving a"

Words we’re predicting:

a
the
talk
gift
hat
...
Example:

Previous words: “giving a"

\[
b = \begin{bmatrix}
3.0 \\
2.5 \\
-0.2 \\
0.1 \\
1.2 \\
\ldots
\end{bmatrix}
\]

Words we’re predicting

How likely are they?
Example:

Previous words: “giving a”

Words we’re predicting: a, the, talk, gift, hat, ...

How likely are they? How likely are they given prev. word is “a”?

\[
b = \begin{pmatrix}
3.0 \\
2.5 \\
-0.2 \\
0.1 \\
1.2 \\
\end{pmatrix}
\]

\[
w_{1,a} = \begin{pmatrix}
-6.0 \\
-5.1 \\
0.2 \\
0.1 \\
0.5 \\
\end{pmatrix}
\]
Example:

Previous words: “giving a”

<table>
<thead>
<tr>
<th>Words we’re predicting</th>
<th>How likely are they?</th>
<th>How likely are they given prev. word is “a”?</th>
<th>How likely are they given 2nd prev. word is “giving”?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example:

Previous words: "giving a"

Words we’re predicting

How likely are they?

How likely are they given prev. word is “a”?

How likely are they given 2nd prev. word is “giving”?

Total score
Softmax

- Convert scores into probabilities by taking the exponent and normalizing (softmax)

\[
P(x_i \mid x_{i-n+1}^{i-1}) = \frac{e^{s(x_i \mid x_{i-n+1}^{i-1})}}{\sum_i \tilde{x}_i e^{s(\tilde{x}_i \mid x_{i-n+1}^{i-1})}}
\]

\[
s = \begin{pmatrix}
-3.2 \\
-2.9 \\
1.0 \\
2.2 \\
0.6 \\
\ldots
\end{pmatrix}
\]

\[
p = \begin{pmatrix}
0.002 \\
0.003 \\
0.329 \\
0.444 \\
0.090 \\
\ldots
\end{pmatrix}
\]
A Computation Graph View

giving

lookup2

+ 

lookup1

bias

scores

= 

probs

softmax

Each vector is size of output vocabulary
A Note: “Lookup”

- Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings

![Diagram showing a matrix of word embeddings with `lookup(2)` pointing to a single vector.](image)
A Note: “Lookup”

• Lookup can be viewed as “grabbing” a single vector from a big matrix of word embeddings.

num. words

vector size

lookup(2)

• Similarly, can be viewed as multiplying by a “one-hot” vector.

num. words

vector size

0
1
0
0

• Former tends to be faster.
Training a Model

• **Reminder**: to train, we calculate a “loss function” (a measure of how bad our predictions are), and move the parameters to reduce the loss.
Training a Model

- **Reminder**: to train, we calculate a “loss function” (a measure of how bad our predictions are), and move the parameters to reduce the loss.

- The most common loss function for probabilistic models is “negative log likelihood”:

\[
p = \begin{pmatrix}
0.002 \\
0.003 \\
0.329 \\
0.444 \\
0.090 \\
0.002 \\
0.003 \\
0.329 \\
0.444 \\
0.090 \\
\end{pmatrix}
\]

If element 3 (or zero-indexed, 2) is the correct answer:

\[-\log(0.329) \approx 1.112\]
Parameter Update

- Back propagation allows us to calculate the derivative of the loss with respect to the parameters $\frac{\partial l}{\partial \theta}$.

- Simple stochastic gradient descent optimizes parameters according to the following rule:

$$\theta \leftarrow \theta - \alpha \frac{\partial l}{\partial \theta}$$
Choosing a Vocabulary
Unknown Words

• Necessity for UNK words
  • We won’t have all the words in the world in training data
  • Larger vocabularies require more memory and computation time
Unknown Words

- Necessity for UNK words
  - We won’t have all the words in the world in training data
  - Larger vocabularies require more memory and computation time
- Common ways:
  - Frequency threshold (usually UNK <= 1)
  - Rank threshold
Evaluation and Vocabulary

• **Important**: the vocabulary must be the same over models you compare

• Or more accurately, all models must be able to generate the test set (it’s OK if they can generate more than the test set, but not less)

• e.g. Comparing a character-based model to a word-based model is fair, but not vice-versa
Beyond Linear Models
Linear Models can’t Learn Feature Combinations

- These can’t be expressed by linear features
- What can we do?
  - Remember combinations as features (individual scores for “farmers eat”, “cows eat”)
    → Feature space explosion!
  - Neural nets
Neural Language Models

- (See Bengio et al. 2004)

\[
\begin{align*}
\text{tanh}(W_1 h + b_1) & \\
W & + \text{bias scores} = \text{probs}
\end{align*}
\]
Where is Strength Shared?

Word embeddings: Similar input words get similar vectors

giving a lookup lookup

tanh(\(W_1 * h + b_1\))

Similar output words get similar rows in in the softmax matrix

Similar contexts get similar hidden states

W + bias scores = softmax probs
What Problems are Handled?

- Cannot share strength among similar words

  - she bought a **car**  
  - she purchased a **car**
  - she bought a **bicycle**
  - she purchased a **bicycle**

→ solved, and similar contexts as well! 😊
What Problems are Handled?

• Cannot share strength among similar words

<table>
<thead>
<tr>
<th>she bought a car</th>
<th>she bought a bicycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>she purchased a car</td>
<td>she purchased a bicycle</td>
</tr>
</tbody>
</table>

→ solved, and similar contexts as well! 😊

• Cannot condition on context with intervening words

| Dr. Jane Smith       | Dr. Gertrude Smith   |

→ solved! 😊
What Problems are Handled?

- Cannot share strength among **similar words**

  | she bought a car | she bought a bicycle |
  | she purchased a car | she purchased a bicycle |

  → solved, and similar contexts as well! 😊

- Cannot condition on context with **intervening words**

  | Dr. Jane Smith | Dr. Gertrude Smith |

  → solved! 😊

- Cannot handle **long-distance dependencies**

  | for tennis class he wanted to buy his own racquet |
  | for programming class he wanted to buy his own computer |

  → not solved yet 😞
Training Tricks
Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time

  - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?
Shuffling the Training Data

- Stochastic gradient methods update the parameters a little bit at a time
  - What if we have the sentence “I love this sentence so much!” at the end of the training data 50 times?
  - To train correctly, we should randomly shuffle the order at each time step
Other Optimization Options

- **SGD with Momentum**: Remember gradients from past time steps to prevent sudden changes

- **Adagrad**: Adapt the learning rate to reduce learning rate for frequently updated parameters (as measured by the variance of the gradient)

- **Adam**: Like Adagrad, but keeps a running average of momentum and gradient variance

- **Many others**: RMSProp, Adadelta, etc. (See Ruder 2016 reference for more details)
Early Stopping, Learning Rate Decay

• Neural nets have tons of parameters: we want to prevent them from over-fitting
Early Stopping, Learning Rate Decay

- Neural nets have tons of parameters: we want to prevent them from over-fitting.
- We can do this by monitoring our performance on held-out development data and stopping training when it starts to get worse.
- It also sometimes helps to reduce the learning rate and continue training.
Dropout

- Neural nets have lots of parameters, and are prone to overfitting

- Dropout: randomly zero-out nodes in the hidden layer with probability $p$ at **training time only**

- Because the number of nodes at training/test is different, scaling is necessary:
  - Standard dropout: scale by $p$ at test time
  - Inverted dropout: scale by $1/(1-p)$ at training time
Efficiency Tricks: Operation Batching
Efficiency Tricks: Mini-batching

- On modern hardware, 10 operations of size 1 is much slower than 1 operation of size 10
- Minibatching combines together smaller operations into one big one
Minibatching
Conditional Generation

CSCI 699
Instructor: Xiang Ren
USC Computer Science
“The Malfoys!” said Hermione.

Harry was watching him. He looked like Madame Maxime. When she strode up the wrong staircase to visit himself.

“I’m afraid I’ve definitely been suspended from power, no chance—indeed?” said Snape. He put his head back behind them and read groups as they crossed a corner and fluttered down onto their ink lamp, and picked up his spoon. The doorbell rang. It was a lot cleaner down in London.
Conditioned Language Models

- Not just generate text, generate text according to some specification

**Input X**
- Structured Data
- English Document
- Utterance
- Image
- Speech

**Output Y(\text{Text})**
- NL Description
- Japanese
- Short Description
- Response
- Text
- Transcript

**Task**
**Conditioned Language Models**

- Not just generate text, generate text according to some specification

<table>
<thead>
<tr>
<th><strong>Input X</strong></th>
<th><strong>Output Y(\text{Text})</strong></th>
<th><strong>Task</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Data</td>
<td>NL Description</td>
<td>NL Generation</td>
</tr>
<tr>
<td>English Document</td>
<td>Japanese Short Description</td>
<td>Translation</td>
</tr>
<tr>
<td>Utterance</td>
<td>Response Text</td>
<td>Summarization</td>
</tr>
<tr>
<td>Image</td>
<td>Transcript</td>
<td>Response Generation</td>
</tr>
<tr>
<td>Speech</td>
<td></td>
<td>Image Captioning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Speech Recognition</td>
</tr>
</tbody>
</table>
Formulation and Modeling
Calculating the Probability of a Sentence

\[ P(X) = \prod_{i=1}^{I} P(x_i \mid x_1, \ldots, x_{i-1}) \]
Conditional Language Models

\[ P(Y|X) = \prod_{j=1}^{J} P(y_j | X, y_1, \ldots, y_{j-1}) \]

Added Context!
(One Type of) Language Model (Mikolov et al. 2011)

I hate this movie
(One Type of) Conditional Language Model
(Sutskever et al. 2014)

Encoder

Decoder

I hate this movie

kono eiga ga kirai

I hate this movie

I hate this movie

I hate this movie

argmax argmax argmax argmax argmax

argmax argmax argmax argmax argmax

argmax argmax argmax argmax argmax
How to Pass Hidden State?

- Initialize decoder w/ encoder (Sutskever et al. 2014)
How to Pass Hidden State?

• Initialize decoder w/ encoder (Sutskever et al. 2014)

• Transform (can be different dimensions)
How to Pass Hidden State?

- Initialize decoder w/ encoder (Sutskever et al. 2014)

![Diagram of decoder initialization with encoder]

- Transform (can be different dimensions)

![Diagram of encoder, transform, decoder]

- Input at every time step (Kalchbrenner & Blunsom 2013)

![Diagram of decoder at every time step]

encoder $\rightarrow$ decoder

encoder $\rightarrow$ transformer $\rightarrow$ decoder

encoder $\rightarrow$ decoder $\rightarrow$ decoder

encoder $\rightarrow$ decoder $\rightarrow$ decoder
Methods of Generation
The Generation Problem

• We have a model of $P(Y|X)$, how do we use it to generate a sentence?
The Generation Problem

• We have a model of $P(Y|X)$, how do we use it to generate a sentence?

• Two methods:
  
  • **Sampling**: Try to generate a *random* sentence according to the probability distribution.

  • **Argmax**: Try to generate the sentence with the *highest* probability.
Ancestral Sampling

- Randomly generate words one-by-one.

\[
\text{while } y_{j-1} \neq "\langle/s\rangle": \quad y_j \sim P(y_j | X, y_1, \ldots, y_{j-1})
\]

- An exact method for sampling from \( P(X) \), no further work needed.
Greedy Search

• One by one, pick the single highest-probability word

\[
\text{while } y_{j-1} \neq "</s>": \\
y_j = \text{argmax } P(y_j | X, y_1, ..., y_{j-1})
\]

• Not exact, real problems:
  • Will often generate the “easy” words first
  • Will prefer multiple common words to one rare word
Beam Search

• Instead of picking one high-probability word, maintain several paths

• Some in reading materials, more in a later class
Model Ensembling
Ensembling

• Combine predictions from multiple models

![Diagram of ensembling]

• Why?
  • Multiple models make somewhat uncorrelated errors
  • Models tend to be more uncertain when they are about to make errors
  • Smooths over idiosyncrasies of the model
Linear Interpolation

- Take a weighted average of the $M$ model probabilities

$$P(y_j \mid X, y_1, \ldots, y_{j-1}) = \frac{\sum_{m=1}^{M} P_m(y_j \mid X, y_1, \ldots, y_{j-1}) P(m \mid X, y_1, \ldots, y_{j-1})}{\text{Probability according to model } m} \text{ Probability of model } m$$

- Second term often set to uniform distribution $1/M$
Log-linear Interpolation

- Weighted combination of log probabilities, normalize

\[ P(y_j \mid X, y_1, \ldots, y_{j-1}) = \]

\[
\text{softmax} \left( \sum_{m=1}^{M} \lambda_m(X, y_1, \ldots, y_{j-1}) \log P_m(y_j \mid X, y_1, \ldots, y_{j-1}) \right)
\]

Normalize \quad \text{Interpolation coefficient for model } m \quad \text{Log probability of model } m

- Interpolation coefficient often set to uniform distribution \(1/M\)
Problem: Ensembling means we have to use $M$ models at test time, increasing our time/memory complexity

Parameter averaging is a cheap way to get some good effects of ensembling

Basically, write out models several times near the end of training, and take the average of parameters
Ensemble Distillation
(e.g. Kim et al. 2016)

- **Problem:** parameter averaging only works for models within the same run

- Knowledge distillation trains a model to *copy the ensemble*
  - Specifically, it tries to match the description over predicted words
  - Why? We want the model to make the same mistakes as an ensemble
  - Shown to increase accuracy notably
Stacking

• What if we have two very different models where prediction of outputs is done in very different ways?

• e.g. a word-by-word translation model and character-by-character translation model

• Stacking uses the output of one system in calculating features for another system
How do we Evaluate?
Basic Evaluation Paradigm

• Use parallel test set
• Use system to generate translations
• Compare target translations w/ reference
Human Evaluation

• Ask a human to do evaluation

太郎が花子を訪れた

Taro visited Hanako  the Taro visited the Hanako  Hanako visited Taro

Adequate?  Yes  Yes  No
Fluent?  Yes  No  Yes
Better?  1  2  3

• Final goal, but slow, expensive, and sometimes inconsistent
BLEU

• Works by comparing n-gram overlap w/ reference

Reference: Taro visited Hanako
System: the Taro visited the Hanako

\[
\text{1-gram: } \frac{3}{5} \\
\text{2-gram: } \frac{1}{4}
\]

\[
\text{Brevity: } \min(1, \frac{|System|}{|Reference|}) = \min(1, \frac{5}{3})
\]

\[
\text{Brevity penalty } = 1.0
\]

\[
\text{BLEU-2 } = \left( \frac{3}{5} \times \frac{1}{4} \right)^{\frac{1}{2}} \times 1.0
\]

\[
= 0.387
\]

• **Pros:** Easy to use, good for measuring system improvement

• **Cons:** Often doesn’t match human eval, bad for comparing very different systems
METEOR

- Like BLEU in overall principle, with many other tricks: consider paraphrases, reordering, and function word/content word difference

- **Pros:** Generally significantly better than BLEU, esp. for high-resource languages

- **Cons:** Requires extra resources for new languages (although these can be made automatically), and more complicated
Perplexity

• Calculate the perplexity of the words in the held-out set *without* doing generation

• **Pros:** Naturally solves multiple-reference problem!

• **Cons:** Doesn’t consider decoding or actually generating output.

• May be reasonable for problems with lots of ambiguity.
What Do We Condition On?
From Structured Data
(e.g. Wen et al 2015)

• When you say “Natural Language Generation” to an old-school NLPer, it means this

<table>
<thead>
<tr>
<th>act type</th>
<th>SF Restaurant</th>
<th>SF Hotel</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform,</td>
<td>inform_only, reject,</td>
<td></td>
</tr>
<tr>
<td>confirm,</td>
<td>select, request, reqmore, goodbye</td>
<td></td>
</tr>
<tr>
<td>name,</td>
<td>type, *pricerange, price, phone, address,</td>
<td></td>
</tr>
<tr>
<td>shared</td>
<td>postcode, *area, *near</td>
<td></td>
</tr>
<tr>
<td>*food</td>
<td></td>
<td>*hasinternet</td>
</tr>
<tr>
<td>*goodformeal</td>
<td></td>
<td>*acceptscards</td>
</tr>
<tr>
<td>*kids-allowed</td>
<td></td>
<td>*dogs-allowed</td>
</tr>
</tbody>
</table>

**bold**=binary slots, **=slots can take “don’t care” value
From Input + Labels
(e.g. Zhou and Neubig 2017)

- For example, word + morphological tags -> inflected word

- Other options: politeness/gender in translation, etc.
From Images
(e.g. Karpathy et al. 2015)

- Input is image features, output is text

"A Tabby cat is leaning on a wooden table, with one paw on a laser mouse and the other on a black laptop"
Other Auxiliary Information

• Name of a recipe + ingredients -> recipe (Kiddon et al. 2016)

• TED talk description -> TED talk (Hoang et al. 2016)

• etc. etc.
Questions?