CS11-737 Multilingual NLP
Machine Translation/
Sequence-to-sequence Models

Graham Neubig

Site
http://demo.clab.cs.cmu.edu/11737fa20/
Language Models

- Language models are generative models of text

\[ s \sim P(x) \]

“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.

Text Credit: Max Deutsch (https://medium.com/deep-writing/)
**Conditioned Language Models**

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

<table>
<thead>
<tr>
<th>Input X</th>
<th>Output Y (Text)</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Data</td>
<td>NL Description</td>
<td>NL Generation</td>
</tr>
<tr>
<td>English</td>
<td>Japanese</td>
<td>Translation</td>
</tr>
<tr>
<td>Document</td>
<td>Short Description</td>
<td>Summarization</td>
</tr>
<tr>
<td>Utterance</td>
<td>Response</td>
<td>Response Generation</td>
</tr>
<tr>
<td>Image</td>
<td>Text</td>
<td>Image Captioning</td>
</tr>
<tr>
<td>Speech</td>
<td>Transcript</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

\[ P(Y \mid X) = \prod_{j=1}^{J} P(y_j \mid X, y_1, \ldots, y_{j-1}) \]
(One Type of) **Language Model**

**(Mikolov et al. 2011)**

(One Type of) **Conditional Language Model**
(Sutskever et al. 2014)

---

Encoder

How to Pass Hidden State?

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

• Transform (can be different dimensions)

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

Methods of Generation
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.

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

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

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

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

• 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
Attention
Sentence Representations

**Problem!**

“You can’t cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!”
— Ray Mooney

- But what if we could use multiple vectors, based on the length of the sentence.

  this is an example

  this is an example
Attention: Basic Idea

(Bahdanau et al. 2015)

- Encode each word in the sentence into a vector
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”
- Use this combination in picking the next word

Calculating Attention (1)

- Use “query” vector (decoder state) and “key” vectors (all encoder states)
- For each query-key pair, calculate weight
- Normalize to add to one using softmax

<table>
<thead>
<tr>
<th>Key Vectors</th>
<th>$a_1 = 2.1$</th>
<th>$a_2 = -0.1$</th>
<th>$a_3 = 0.3$</th>
<th>$a_4 = -1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Vector</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\alpha_1 = 0.76$</td>
<td>$\alpha_2 = 0.08$</td>
<td>$\alpha_3 = 0.13$</td>
<td>$\alpha_4 = 0.03$</td>
</tr>
</tbody>
</table>
Calculating Attention (2)

• Combine together value vectors (usually encoder states, like key vectors) by taking the weighted sum

\[ α_1 = 0.76 \quad α_2 = 0.08 \quad α_3 = 0.13 \quad α_4 = 0.03 \]

• Use this in any part of the model you like
A Graphical Example

Image from Bahdanau et al. (2015)
Attention Score Functions (1)

- \( q \) is the query and \( k \) is the key

- **Multi-layer Perceptron** (Bahdanau et al. 2015)
  
  \[
  a(q, k) = w_2^\top \tanh(W_1[q; k])
  \]

  - Flexible, often very good with large data

- **Bilinear** (Luong et al. 2015)
  
  \[
  a(q, k) = q^\top W k
  \]

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." *EMNLP 2015.*
Attention Score Functions (2)

- **Dot Product** (Luong et al. 2015)
  \[ a(q, k) = q^T k \]
  - No parameters! But requires sizes to be the same.

- **Scaled Dot Product** (Vaswani et al. 2017)
  - *Problem*: scale of dot product increases as dimensions get larger
  - *Fix*: scale by size of the vector
    \[ a(q, k) = \frac{q^T k}{\sqrt{|k|}} \]
Attention is not Alignment!
(Koehn and Knowles 2017)

- Attention is often blurred
- Attention is often off by one
- It can even be manipulated to be non-intuitive!
(Pruthi et al. 2020)

Improvements to Attention
Coverage

• **Problem:** Neural models tend to drop or repeat content

• **Solution:** Model how many times words have been covered
  
  • Impose a penalty if attention not approx. 1 over each word (Cohn et al. 2015)
  
  • Add embeddings indicating coverage (Mi et al. 2016)

Multi-headed Attention

- **Idea:** multiple attention “heads” focus on different parts of the sentence.

- e.g. Different heads for “copy” vs regular (Allamanis et al. 2016)

- Or multiple independently learned heads (Vaswani et al. 2017)

---


Supervised Training
(Liu et al. 2016)

- Sometimes we can get “gold standard” alignments \textit{a-priori}
  - Manual alignments
  - Pre-trained with strong alignment model
- \textbf{Train the model to match} these strong alignments

Self Attention/Transformers
Self Attention
(Cheng et al. 2016)

• Each element in the sentence attends to other elements → context sensitive encodings!

this is an example

• Can be used as drop-in replacement for other sequence models, e.g. RNNs, CNNs

Why Self Attention?

- Unlike RNNs, parallelizable -> fast training on GPUs!
- Unlike CNNs, easily capture global context
- In general, high accuracy, although not 100% clear when all things being held equal (Chen et al. 2018)
- *Downside*: quadratic computation time

Summary of the “Transformer” (Vaswani et al. 2017)

- A sequence-to-sequence model based entirely on attention
- Strong results on standard WMT datasets
- Fast: only matrix multiplications
Transformer Attention Tricks

• **Self Attention:** Each layer combines words with others

• **Multi-headed Attention:** 8 attention heads learned independently

• **Normalized Dot-product Attention:** Remove bias in dot product when using large networks

• **Positional Encodings:** Make sure that even if we don’t have RNN, can still distinguish positions
Transformer Training Tricks

- **Layer Normalization**: Help ensure that layers remain in a reasonable range.
- **Specialized Training Schedule**: Adjust the default learning rate of the Adam optimizer.
- **Label Smoothing**: Insert some uncertainty in the training process.
- **Masking for Efficient Training**
Masking for Training

• We want to perform training in as few operations as possible using big matrix multiplies

• We can do so by “masking” the results for the output

\[
\begin{array}{ccccccccc}
 kono & eiga & ga & kirai & I & hate & this & movie & </s>
\end{array}
\]
A Unified View of Sequence-to-sequence Models

- Review: sequence labeling

- Sequence-to-sequence modeling
In-class Assignment
Code Walk

• There will be no graded discussion, but we'll have a code walk through The Annotated Transformer https://nlp.seas.harvard.edu/2018/04/03/attention.html

• We'll go into depth into some of the design decisions, their motivation, etc.