# Sequence-to-Sequence Modeling & Attention



Tanmoy Chakraborty Associate Professor, IIT Delhi https://tanmoychak.com/

Slides are adopted from the Stanford course 'NLP with DL' by C. Manning and UMass course 'Advanced NLP' by M Iyyer

### Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a single neural network.
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.







### Neural Machine Translation (NMT)

#### The Sequence-to-Sequence Model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.



**Encoder RNN** produces an encoding of the source sentence.





### Neural Machine Translation (NMT)





#### Sequence-to-Sequence is Versatile!

- The general notion here is an encoder-decoder model
  - One neural network takes input and produces a neural representation
  - Another network produces output based on that neural representation
  - If the input and output are sequences, we call it a seq2seq model
- Sequence-to-sequence is useful for more than just MT
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - **Dialogue** (previous utterances  $\rightarrow$  next utterance)
  - **Parsing** (input text  $\rightarrow$  output parse as sequence)
  - Code generation (natural language  $\rightarrow$  Python code)





### Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model** 
  - Language Model because the decoder is predicting the next word of the target sentence y
  - Conditional because its predictions are also conditioned on the source sentence x
- NMT directly calculates P(y|x)

$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots P(y_T|y_1, \dots, y_{T-1}, x)$$

Probability of next target word, given target words so far and source sentence *x* 

• How to train an NMT system?





### Training an NMT System







### Greedy decoding

• We saw how to generate (or "decode") the target sentence by taking argmax on each step of the decoder.



- This is greedy decoding (take most probable word on each step)
- Problems with this method?





### **Problems With Greedy Decoding**

- Greedy decoding has no way to undo decisions!
- Input: *il a m'entarté* (he hit me with a pie)
- → he \_\_\_\_\_
- → he hit \_\_\_\_\_
- → he hit a \_\_\_\_\_ (whoops! no going back now...)

#### How to fix this?







### **Exhaustive Search Decoding**

• Ideally we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots, P(y_T|y_1, \dots, y_{T-1}, x)$$
$$= \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, x)$$

- We could try computing all possible sequences y
- This means that on each step *t* of the decoder, we're tracking *V*<sup>t</sup> possible partial translations, where *V* is vocab size
- This O(V<sup>T</sup>) complexity is far too expensive!





### **Beam Search Decoding**

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - *k* is the beam size (in practice around 5 to 10)
- A hypothesis  $y_1, \ldots, y_t$  has a score which is its log probability:

score
$$(y_1, \ldots, y_t) = \log P_{LM}(y_1, \ldots, y_t | x) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top *k* on each step
- Beam search is not guaranteed to find optimal solution
  - But much more efficient than exhaustive search!





### **Beam Search Decoding: Example**

#### Beam size = k = 2.



Calculate prob distribution of next word

Beam size = k = 2. Blue numbers = score
$$(y_1, ..., y_t) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$$



Take top *k* words and compute scores

Beam size = k = 2. Blue numbers = score
$$(y_1, ..., y_t) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$$



Beam size = k = 2. Blue numbers = score
$$(y_1, ..., y_t) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$$



Beam size = k = 2. Blue numbers = score
$$(y_1, ..., y_t) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, ..., y_{i-1}, x)$$



Beam size = k = 2. Blue numbers = score
$$(y_1, \ldots, y_t) = \sum_{i=1}^t \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$



Beam size = k = 2. Blue numbers = score
$$(y_1, \ldots, y_t) = \sum_{i=1}^t \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$



For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam size = k = 2. Blue numbers = score
$$(y_1, \ldots, y_t) = \sum_{i=1}^t \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$



Of these *k*<sup>2</sup> hypotheses, just keep *k* with highest scores



For each of the *k* hypotheses, find top *k* next words and calculate scores



Of these *k*<sup>2</sup> hypotheses, just keep *k* with highest scores



For each of the *k* hypotheses, find top *k* next words and calculate scores



This is the top-scoring hypothesis!



Backtrack to obtain the full hypothesis

### **Beam Search Decoding: Stopping Criterion**

- In greedy decoding, usually we decode until the model produces a <END> token
  - For example: <START> he hit me with a pie <END>
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
  - We reach timestep *T* (where *T* is some pre-defined cutoff), or
  - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)





### Beam Search Decoding: Finishing Up

- We have our list of completed hypotheses.
- How to select top one with highest score?
- Each hypothesis  $y_1, \ldots, y_t$  on our list has a score

score
$$(y_1, \dots, y_t) = \log P_{LM}(y_1, \dots, y_t | x) = \sum_{i=1}^t \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$

- **Problem:** longer hypotheses have lower scores
- Fix: Normalize by length. Use this to select the top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\rm LM}(y_i | y_1, \dots, y_{i-1}, x)$$





### **Other Decoding Techniques**







#### Deterministic vs. Stochastic Decoding Strategies

- The decoding methods which we discussed in the previous lecture greedy decoding, exhaustive search and beam search – are all deterministic, i.e., given the same model and the same input context, they always generate the same output sequence.
- However, to ensure diversity of the generated responses we need stochastic decoding strategies.
  - The stochastic decoding strategies generate diverse responses even for the same model and same input context.
- In this lecture, we will look into three stochastic decoding strategies:
  - 1. Top-k Sampling
  - 2. Top-p / Nucleus Sampling
  - 3. Temperature Sampling







### Random Sampling

- Before moving onto those three stochastic decoding strategies, let's first look at simple random sampling.
- If we want to generate a sequence of N words  $w_1, ..., w_N$ , then random sampling can be written formally as the following algorithm:

```
i \in 1
w_i \sim P(w_i \mid c)
while w_i != EOS \text{ or } i <= N:
i \in i + 1
w_i \sim P(w_i \mid c, w_{< i})
```

Intuitively, if say P(cheese | c='I like pizza with loads of') = 0.35, then on giving c as input 100 times, on random sampling, 'cheese' is expected to be generated approximately 35 times as the next token, though the exact count may vary due to randomness.





### Top-k Sampling

- In greedy decoding, the token with the highest probability was chosen at each step, making it deterministic. Top-k sampling is a stochastic generalization of greedy decoding.
- Top-k sampling involves the following steps:
  - 1. Choose in advance a number of tokens *k*
  - 2. Given the probability  $P(w_t | c, w_{<t})$  for all tokens in the vocabulary V, as generated by the model, sort the tokens by their likelihood, and throw away any token that is not one of the top k most probable tokens.
  - 3. Renormalize the scores of the *k* tokens to be a legitimate probability distribution.
  - 4. Randomly sample a token from within these remaining *k* most-probable tokens according to its probability.
- When k = 1, top-k sampling is identical to greedy decoding.



### Top-k Sampling: Working Example



- Let's take **k=3**.
- <u>Step-1</u>: Sort the tokens by their likelihood, and throw away any token that is not one of the top *k* most probable tokens.
  - In our example, top-3 tokens with P(w|c) are: cheese (0.35), toppings (0.20), pepperoni (0.12)





### Top-k Sampling: Working Example



- <u>Step-2:</u> Renormalize the scores of the *k* tokens to be a legitimate probability distribution.
- After renormalization,  $P_{renorm}(w|c) = \frac{P(w|c)}{\sum_{x \in top-k} P(x|c)}$ .
  - Renormalized probabilities: cheese (0.522), toppings (0.299), pepperoni (0.179)





### Top-k Sampling: Working Example





- <u>Step-3</u>: Randomly sample a token from within these remaining *k* most-probable tokens according to its probability after renorm.
- For generating the next token,  $w_t \sim P_{renorm}(w_t | c)$





### Problem With Top-k Sampling

- In top-k sampling *k* is fixed, but the shape of the probability distribution over tokens differs in different contexts.
- For example, if we set k = 10:
  - Sometimes the top 10 tokens will be very likely and include most of the probability mass.
  - But other times the probability distribution will be flatter and the top 10 tokens will only include a small part of the probability mass.

#### **Solution:** Top-p / Nucleus Sampling

Image source: https://towardsdatascience.com/how-to-sample-from-language-models-682bceb97277



Introduction to LLMs



Tanmoy Chakraborty

### Top-p / Nucleus Sampling

- The goal of nucleus sampling is the same as top-k sampling, which is to truncate the distribution to remove the very unlikely tokens.
- Here, we keep the top *p* percent of the probability mass.
  - By measuring probability rather than the number of tokens, the hope is that the measure will be more robust in very different contexts, dynamically increasing and decreasing the pool of token candidates.
- Formally, given a distribution  $P(w_t | c, w_{< t})$ , the top-p vocabulary  $V^{(p)}$  is the smallest set of tokens such that:

$$\sum_{w \in V^{(p)}} P(w|c, w_{< t}) \ge p$$


# Top-p Sampling: Working Example



ICS





Probability Distribution for Next Token

# Top-p Sampling: Working Example



• <u>Step-2:</u> Renormalize the scores of the selected tokens (same procedure as top-k).

Renormalized probabilities: cheese (
$$\frac{0.35}{0.35+0.20} = 0.64$$
), toppings ( $\frac{0.20}{0.35+0.20} = 0.36$ )



Introduction to LLMs

# Top-p Sampling: Working Example







Tanmoy Chakraborty

# **Temperature Sampling**

- In temperature sampling, we don't truncate the distribution but instead reshape it.
- The **temperature parameter**  $(\tau)$  is incorporated while computing **softmax** over the logits of the tokens (for next token prediction).

#### **Normal Softmax**

$$P(t_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- $P(t_i)$ : Probability assigned to token  $t_i$
- $z_i$ : Logit for token  $t_i$

#### Softmax with Temperature

$$P(t_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

- τ : Temperature parameter
  - τ > 0







#### **Effect of Temperature**

I like pizza with loads of

Probability Distribution for Next Token Prediction Over V



Probability Distribution for Next Token Prediction Over V

1

0.8

0.4

Probability Distribution for Next Token Prediction Over V





# Effect of Temperature



- A low temperature sharpens the probabilities, making the model confident and more deterministic.
  - As τ approaches 0 the probability of the most likely word approaches 1.
  - Lower temperatures make the model more "confident" which can be useful in applications like question answering.
- A high temperature flattens the probabilities, encouraging diversity and creativity in outputs but with a risk of incoherence.
  - Thus, higher temperatures make the model more "creative" which can be useful when generating prose, for example.



# NMT: The First Big Success Story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in 2014 to the leading standard method in 2016

- 2014: First seq2seq paper published [Sutskever et al. 2014]
- 2016: Google Translate switches from SMT to NMT and by 2018 everyone had
  - https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html



- This was amazing!
  - SMT systems, built by hundreds of engineers over many years, were outperformed by NMT systems trained by small groups of engineers in a few months





# **Issues With RNN**

- Linear interaction distance
- Bottleneck problem
- Lack of parallelizability

# ATTENTION









A non-LLM breaking news for today

Announced on Mar 5: https://awards.acm.org /about/2024-turing

#### ACM has named **Andrew G. Barto** and **Richard S. Sutton** as the recipients of the **2024 ACM A.M. Turing Award** !



An Introduction second edition

Richard S. Sutton and Andrew G. Barto

They are also the authors of the 'bible of RL'.



LMs: Introduction and Recent Advances



Andrew Barto

**Richard Sutton** 

RL-based techniques also serve as critical steps during post-training of LLMs. We will discuss a few of them (hopefully) by the end of this month. Barto and Sutton, jointly and with others, developed many of the basic algorithmic approaches for RL. These include their foremost contribution, **Temporal Difference Learning**, which made an important advance in solving reward prediction problems, as well as **Policy-Gradient Methods** and the **use of neural networks as a tool to represent learned functions**.



# Sequence-to-Sequence: The Bottleneck Problem







# Sequence-to-Sequence: The Bottleneck Problem







- Attention provides a solution to the bottleneck problem.
- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence
- Let's start with the visualization of the attention mechanism.

















RNN

LLMs: Introduction and Recent Advances





Tanmoy Chakraborty



























Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.







Tanmoy Chakraborty









LLMs: Introduction and Recent Advances



Tanmoy Chakraborty





LLMs: Introduction and Recent Advances





Tanmoy Chakraborty





LLMs: Introduction and Recent Advances









Decoder RNN

# **Attention: In Equations**

- We have encoder hidden states  $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state  $s_t \in \mathbb{R}^h$
- We get the attention scores e<sup>t</sup> for this step:

$$oldsymbol{e}^t = [oldsymbol{s}_t^Toldsymbol{h}_1, \dots, oldsymbol{s}_t^Toldsymbol{h}_N] \in \mathbb{R}^N$$

• We take softmax to get the attention distribution  $\alpha^t$  for this step (this is a probability distribution, sums to 1)

$$\alpha^t = \operatorname{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

• We use  $\alpha^t$  to take a weighted sum of the encoder hidden states to get the attention output  $a_t$ 

$$\boldsymbol{a}_t = \sum_{i=1}^N \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

 Finally we concatenate the attention output a<sub>t</sub> with the decoder hidden state s<sub>t</sub> and proceed as in the non-attention seq2seq model

$$[oldsymbol{a}_t;oldsymbol{s}_t]\in\mathbb{R}^{2h}$$



# Attention is Great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself









### Seq2Seq+Attention for LM



Concatenate (or otherwise compose) the attention output with the current hidden state, then pass through a softmax layer to predict the next word





# Attention is a General Deep Learning Technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- <u>However</u>: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)
- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.
- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values).
- Intuition:
  - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
  - Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).





























The output of attention mechanism is the **weighted sum** of hidden vectors.

Instead of simply summing up the hidden vectors, we can transform them using a learned function to generate **value vectors** and then compute a weighted sum.

Decoding



# Variants of Attention

- Original formulation:  $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product:  $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$

Luong et al., 2015

• Dot product:  $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ 

Luong et al., 2015

$$a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$$

Vaswani et al., 2017

More information:

"Deep Learning for NLP Best Practices", Ruder, 2017. <u>http://ruder.io/deep-learning-nlp-best-practices/index.html#attention</u> "Massive Exploration of Neural Machine Translation Architectures" Britz et al. 2017

"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf







# Self-Attention
## **Positional Encoding**

## Positional Encoding - Motivation

- **Problem :** Self-attention processes all the elements of a sequence in parallel without any regard for their order.
  - Example : the sun rises in the east
  - Permuted version : rises in the sun the east
    - the east rises in the sun



- Self-attention is permutation invariant.
- In natural language, it is important to take into account the order of words in a sentence.
- **Solution :** Explicitly add positional information to indicate where a word appears in a sequence







## **Frequency-based Positional Encoding**

• Helps it determine the position of each word (absolute positional information), or the distance between different words in the sequence(relative positional information)

 $PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$ 

 $PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$ 



The frequency decreases along the encoding dimension.

• We will discuss about positional encoding in detail in the next module.



