CS584 Natural Language Processing

Machine Translation, Seq2seq models, Attention models

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **<u>24 hours</u>**.
- **40%** penalty for late submissions within **<u>24-48 hours</u>**.
- After 48 hours, you get **NO** points on the assignment.

Overview

- ❑ Introduce a new task: Machine Translation

↓ is a major use-case of

- ❑ Introduce a new neural architecture: Seq2seq models

↓ is improved by

- ❑ Introduce a new neural technique: Attention

Machine Translation

- Machine Translation (MT) is the task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language).

- x: have a good day (English)
  ↓
- y: bonne journée (French)

Machine Translation

- Translation is easy for (bilingual) people
- Process:
  - Read the text in English
  - Understand it
  - Write it down in French

- Hard tasks for computers
  - The human process is invisible, intangible

History of Machine Translation

- 1950s: Early Machine Translation (Rule-based)
  - Russian → English (motivated by the Cold War!)
  - Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts

- 1990s-2010s: Statistical Machine Translation (Statistics-based)
  - Core idea: Learn a probabilistic model from data
  - The best systems were extremely complex

- 2014- : Neural Machine Translation

Statistical Machine Translation

- Core idea: Learn a probabilistic model from data
- French f → English e
- We want to find best English sentence e, given French sentence f

Mathematically:

$$P(e|f) = \frac{P(e)\,P(f|e)}{P(f)}$$

$$T(f) = \hat{e} = \mathrm{argmax}_{\mathrm{e}}\ P(e|f) = \mathrm{argmax}_{\mathrm{e}}\ P(e)\,P(f|e)$$

Statistical Machine Translation

Components

$$T(f) = \hat{e} = \text{argmax}_e \underline{P(e)} P(f|e)$$

- Language Model
  - Takes care of fluency in the target language
  - Data: corpora in the target language

Statistical Machine Translation

Components

$$T(f) = \hat{e} = \text{argmax}_e P(e)\underline{P(f|e)}$$

- Language Model
  - Takes care of fluency in the target language
  - Data: corpora in the target language

- Translation Model
  - Lexical correspondence between languages
  - Data: aligned corpora in source and target languages

Statistical Machine Translation

Components

$$T(f) = \hat{e} = \underline{\text{argmax}_e} P(e)P(f|e)$$

- Language Model
  - Takes care of fluency in the target language
  - Data: corpora in the target language

- Translation Model
  - Lexical correspondence between languages
  - Data: aligned corpora in source and target languages

- argmax
  - Search done by the *decoder*

The language model P(e)

$$T(f) = \hat{e} = \text{argmax}_e \underline{P(e)} P(f|e)$$

Estimation of how probable a sentence is.

- Refer to the previous slides

The translation model P(f|e)

$$T(f) = \hat{e} = \text{argmax}_e P(e)\underline{P(f|e)}$$

Estimation of the lexical correspondence between languages.

- <u>How to learn translation model P(f|e)?</u>
  - First, need large amount of parallel data (e.g. pairs of human-translated French/English sentences)

The translation model P(f|e)

$$T(f) = \hat{e} = \text{argmax}_e P(e)\underline{P(f|e)}$$

Estimation of the lexical correspondence between languages.

- <u>How to learn translation model P(f|e) from the parallel corpus?</u>
  - Break it down further: we actually want to consider

$$P(f, a|e)$$

where *a* is the **alignment**, i.e. word-level correspondence between French sentence x and English sentence y

What is alignment?

Alignment is the correspondence between particular words in the translated sentence pair.

- Note: Some words have no counterpart

Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993.
http://www.aclweb.org/anthology/J93-2003

Alignment is complex

Alignment can be <u>many-to-one</u>

Alignment is complex

Alignment can be <u>one-to-many</u>

Alignment is complex

Some words are very fertile!

*correction: il m'a entarté

	he	hit	me	with	a	pie
il	■					
a						
m'			■			
entarté		■		■	■	■

Alignment is complex

Alignment can be <u>many-to-many</u> (phrase-level)

Learning alignment for SMT

- We learn $P(f, a|e)$ as a combination of many factors, including:
  - Probability of particular words aligning (also depends on positions in a sentence)
  - Probability of particular words having particular fertility (number of corresponding words)
  - etc.

- Next
  - A basic example of learning alignment

Word Alignment - Vector Representation

- Assumption: each French word f is aligned to exactly one English word e
  - Including NULL

- Alignment vector a = [2,3,4,5,6,6,6]
  - length of a = length of sentence f
  - a_i = j if French position i is aligned to English position j

- How many possible alignments in A?

$$(l+1)^m$$

Formalizing the connection between word alignments & the translation model

$$
\begin{aligned}
& p(f_1, f_2, \ldots, f_m \mid e_1, e_2, \ldots, e_l, m) \\
&= \sum_{a \in A} p(f_1, \ldots, f_m, a_1, \ldots, a_m \mid e_1, \ldots, e_l, m)
\end{aligned}
$$

- We define a conditional model
  - Projecting word translations
  - Through alignment links

IBM Model 1

- Input
  - an English sentence of length l
  - a length m

- For each French position i in 1..m
  - Pick an English source index j
  - Choose a translation

Alignment probabilities are UNIFORM

$$q(j \mid i, l, m) = \frac{1}{l+1}$$

$$t(f_i \mid e_{a_i})$$

Words are translated independently

More on IBM model 1:http://mt-class.org/jhu/slides/lecture-ibm-model1.pdf

t(f|e)

- t(f|e)
  - Word translation probability table
  - for all words in French & English vocab

f	e	p(f | e)
le	the	0.42
la	the	0.4
programme	the	0.001
a	has	0.78
…	…	…

IBM Model 1

- Input
  - an English sentence of length l
  - a length m

- For each French position i in 1..m
  - Pick an English source index j
  - Choose a translation

$$q(j \mid i, l, m) = \frac{1}{l+1}$$

$$t(f_i \mid e_{a_i})$$

$$\boxed{p(f_1 \ldots f_m, a_1 \ldots a_m | e_1 \ldots e_l, m) = \prod_{i=1}^{m} q(a_i | i, l, m) t(f_i | e_{a_i})}$$

- More: IBM Model 2
  - Remove assumption that q is uniform

Decoding for SMT

language model

$$T(f) = \hat{e} = \underline{\text{argmax}_e} P(e) P(f|e)$$

translation model

Responsible for the search in the space of possible translations.

- How to compute this argmax?
  - We could enumerate every possible *e* and calculate the probability? → Too expensive!
  - Answer: Use a heuristic search algorithm to search for the best translation, discarding hypotheses that are too low-probability
  - This process is called decoding

Decoding for SMT

phrase to phrase alignment

German:

Decoding for SMT

Decoding for SMT

Source: ”Statistical Machine Translation", Chapter 6, Koehn, 2009

Statistical Machine Translation

- SMT was a huge research field
- The best systems were extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems had many separately-designed subcomponents
  - Lots of feature engineering
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining extra resources
    - Like tables of equivalent phrases
  - Lots of human effort to maintain
    - Repeated effort for each language pair!

Neural Machine Translation

- A way to do Machine Translation with a single neural network

- The neural network architecture is called sequence-to-sequence (seq2seq) and it involves two RNNs

Seq2Seq Basics

- A relatively new paradigm, with its first published usage in 2014 for English-French translation (Sutskever et al. 2014).

- At a high level, a seq2seq model is an end-to-end model made up of two RNNs:
  - an *encoder*, which takes the model's input sequence as input and encodes it into a fixed-size "context vector"
  - a *decoder*, which uses the context vector from above as a "seed" from which to generate an output sequence.

- Seq2Seq models are often referred to as "encoder-decoder" models.

Seq2Seq

- 2 parts: encoder and decoder.

- The decoder is a text generator.
- Difference from the simple text generator: the initial states are determined by the encoder.

Seq2Seq - Encoder

- **Goal**: read the input sequence to our Seq2Seq model and generate a fixed-dimensional context vector *C* for the sequence.
- The encoder will use a recurrent neural network cell (usually an LSTM or GRU) to read the input tokens one at a time. The final hidden state of the cell will then become *C*.

output the final hidden states
(Context vector *C*)

Source sentence (input)

Seq2Seq - Encoder

- However, it's so difficult to compress an arbitrary-length sequence into a single fixed-size vector (especially for difficult tasks like translation)
- The encoder will usually consist of stacked RNNs

output the final layer's hidden states **(Context vector *C*)**

Seq2Seq - Decoder

- Initialize the hidden state with the context vector
- A stack of several recurrent units (one or more layers) where each predicts an output y_t at a time step t.
- the decoder will literally use the context of the input to generate an output.

Seq2Seq - Training

- Define a **loss**, the cross entropy on the prediction sequence
- minimize it with a gradient descent algorithm and **back-propagation**.
- Both the encoder and decoder are trained at the same time, so that they both learn the same context vector representation.

Neural Machine Translation - Seq2Seq

Source sentence (input)

Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Neural Machine Translation - Seq2Seq

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

Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a Language Model that generates target sentence, conditioned on encoding.

Seq2Seq is versatile!

- Seq2seq 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 → next utterance)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

- The seq2seq 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

Two components in SMT!

- NMT directly calculates P(y|x):

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

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

- How to train a NMT system?
  - Get a big parallel corpus…

Training a Neural Machine Translation system

I am a student

Source sentence (from corpus)

<START> je suis un étudiant

Translated sentence (from corpus)

Training a Neural Machine Translation system

I am a student

Source sentence (from corpus)

<START> je suis un étudiant

Translated sentence (from corpus)

Training a Neural Machine Translation system

Training a Neural Machine Translation system

Training a Neural Machine Translation system

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J1} + J2 + \boxed{J3} + J4 + \boxed{J5}$$

= negative log prob of "je"

= negative log prob of "un"

= negative log prob of "<END>"

Training a Neural Machine Translation system

Seq2seq is optimized as a <u>single system</u>.

Backpropagation operates "end-to-end".

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t = \boxed{J1} + J2 + \boxed{J3} + J4 + \boxed{J5}$$

= negative log prob of "je"

= negative log prob of "un"

= negative log prob of "<END>"

Source sentence (from corpus)

Translated sentence (from corpus)

Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder.
- Greedy decoding takes most probable word on each step

- Problems?
  - Greedy decoding has no way to undo decisions!

- How to fix this?
  - Exhaustive search decoding
  - Beam search decoding

Exhaustive Search Decoding

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

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

- We could try computing <u>all possible sequences</u> y
  - on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
  - This $O(V^T)$ 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:

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

  - Scores are all negative, and higher score is better
  - 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. Blue numbers $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

Calculate prob dist of next word

Beam Search Decoding: example

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

Beam Search Decoding: example

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

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

Beam Search Decoding: example

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

Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: example

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

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

Beam Search Decoding: example

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

Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: example

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

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

Beam Search Decoding: example

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

Of these k^2 hypotheses,
just keep k with highest scores

Beam Search Decoding: example

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

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

Beam Search Decoding: example

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

Of these k^2 hypotheses, just keep k with highest scores

Beam Search Decoding: example

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

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

Beam Search Decoding: example

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

This is the top-scoring hypothesis!

Beam Search Decoding: example

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

Backtrack to obtain the full hypothesis

Beam Search Decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a <END> token
  - e.g. <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.
- <u>How to select top one with highest score?</u>

- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

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

- <u>Problem</u>: longer hypotheses have lower scores

- <u>Fix</u>: Normalize by length. Use this to select top one instead:

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

Advantages of NMT

- Compared to SMT, NMT has many advantages:
  - Better performance
    - More fluent
    - Better use of context
    - Better use of phrase similarities
  - A single neural network to be optimized end-to-end
    - No subcomponents to be individually optimized
  - Requires much less human engineering effort
    - No feature engineering
    - Same method for all language pairs

Disadvantages of NMT

- Compared to SMT:
  - NMT is less interpretable
    - Hard to debug
  - NMT is difficult to control
    - For example, can’t easily specify rules or guidelines for translation

How do we evaluate Machine Translation?

- BLEU (**B**ilingual **E**valuation **U**nderstudy)

- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:
  - n-gram precision (usually for 1, 2, 3 and 4-grams)
  - Plus a penalty for too-short system translations

- BLEU is useful but imperfect
  - There are many valid ways to translate a sentence
  - So a good translation can get a poor BLEU score because it has low n-gram overlap with the human translation

MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]

source: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf

So is Machine Translation solved?

- Nope!

- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs

Further reading: [“Has AI surpassed humans at translation? Not even close!”]

NMT research continues

- NMT is the flagship task for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning

- In 2019: NMT research continues to thrive
  - Researchers have found many, many improvements to the “vanilla” seq2seq NMT system we’ve presented today
  - But one improvement is so integral that it is the new vanilla…

Attention

Seq2Seq: the bottleneck problem

Problems with this architecture?

Seq2Seq: the bottleneck problem

Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

Problems with this architecture?

Attention - Motivation

example

Attention

- 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
- There are many types of attention. We'll examine the encoder mechanisms introduced by Huong et al.[1]
- First we will show via diagram (no equations), then we will show with equations

[1]. Huong et al. 2015, Effective Approaches to Attention-based Neural Machine Translation.

Seq2Seq with attention

Seq2Seq with attention

Seq2Seq with attention

Seq2Seq with attention

Seq2Seq with attention

Seq2Seq with attention

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**.

Seq2Seq with attention

Concatenate **attention output** with **decoder hidden state**, then use to compute y_hat1 as before

Seq2Seq with attention

Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

Seq2Seq with attention

Source sentence (input)

Seq2Seq with attention

Source sentence (input)

Seq2Seq with attention

Source sentence (input)

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^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution a^t for this step (this is a probability distribution and sums to 1)

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

Attention: in equations

- We use a^t to take a weighted sum of the encoder hidden states to get the attention output $\mathbf{a}_t$

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

- Finally we concatenate the attention output $\mathbf{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

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

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
  - Allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to far away 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

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

- You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

Attention is a general Deep Learning technique

- 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).

Attention is a general Deep Learning technique

- 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.

- 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).

Attention variants

- We have some values $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a query $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the attention scores $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get attention distribution α:

There are multiple ways to do this

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

3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the attention output **a** (sometimes called the context vector)

Attention variants

There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.

Summary

- History of Machine Translation (MT)
  - Statistical Machine Translation (SMT)

- Seq2Seq is the architecture for NMT (uses 2 RNNs)

- Attention is a way to focus on particular parts of the input
  - Improves seq2seq a lot!

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You