Natural Language Processing
Sequence to Sequence Models and Attention

Felipe Bravo-Marquez

June 6, 2023

Language Models and Language Generation

- Language modeling is the task of assigning a probability to sentences in a language.
- Example: what is the probability of seeing the sentence “the lazy dog barked loudly”?
- The task can be formulated as the task of predicting the probability of seeing a word conditioned on previous words:

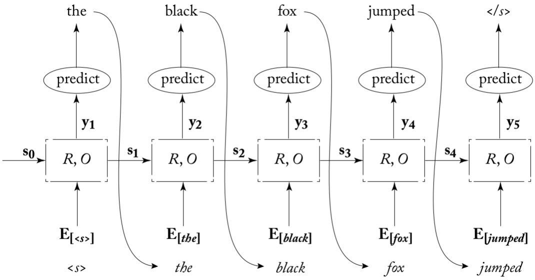
$$P(w_i|w_1, w_2, \cdots, w_{i-1}) = \frac{P(w_1, w_2, \cdots, w_{i-1}, w_i)}{P(w_1, w_2, \cdots, w_{i-1})}$$

Language Models and Language Generation

- RNNs can be used to train language models by tying the output at time *i* with its input at time $i + 1$.
- This network can be used to generate sequences of words or random sentences.
- Generation process: predict a probability distribution over the first word conditioned on the start symbol, and draw a random word according to the predicted distribution.
- Then predict a probability distribution over the second word conditioned on the first, and so on, until predicting the end-of-sequence $< /s>$ symbol.

Language Models and Language Generation

- After predicting a distribution over the next output symbols $P(t_i = k|t_{1:i-1})$, a token t_i is chosen and its corresponding embedding vector is fed as the input to the next step.



- Teacher-forcing: during **training** the generator is fed with the ground-truth previous word even if its own prediction put a small probability mass on it.
- It is likely that the generator would have generated a different word at this state in **test time**.

Sequence to Sequence Problems

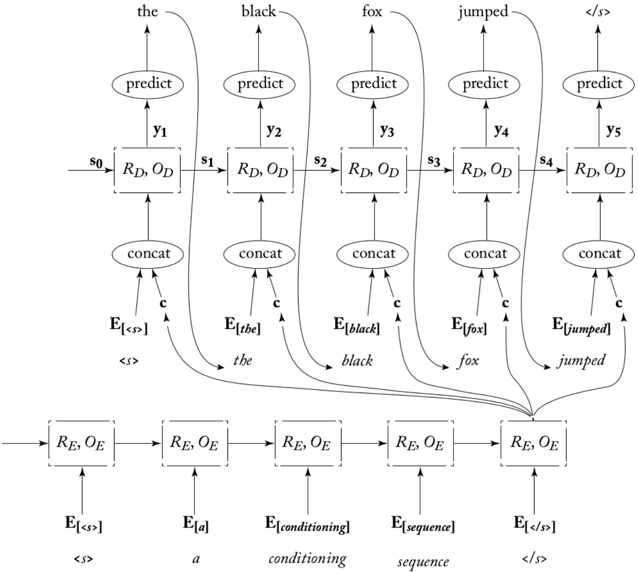
Nearly any task in NLP can be formulated as a sequence to sequence (or conditioned generation) task i.e., generate output sequences from input ones. Input and output sequences can have different lengths.

- Machine Translation: source language to target language.
- Summarization: long text to short text.
- Dialogue (chatbots): previous utterances to next utterance.

Conditioned Generation

- While using the RNN as a generator is a cute exercise for demonstrating its strength, the power of RNN generator is really revealed when moving to a conditioned generation or encoder-decoder framework.
- Core idea: using two RNNs.
- Encoder: One RNN is used to encode the source input into a vector $\vec{c}$.
- Decoder: Another RNN is used to decode the encoder's output and generate the target output.
- At each stage of the generation process the context vector $\vec{c}$ is concatenated to the input $\hat{t}_j$ and the concatenation is fed into the RNN.

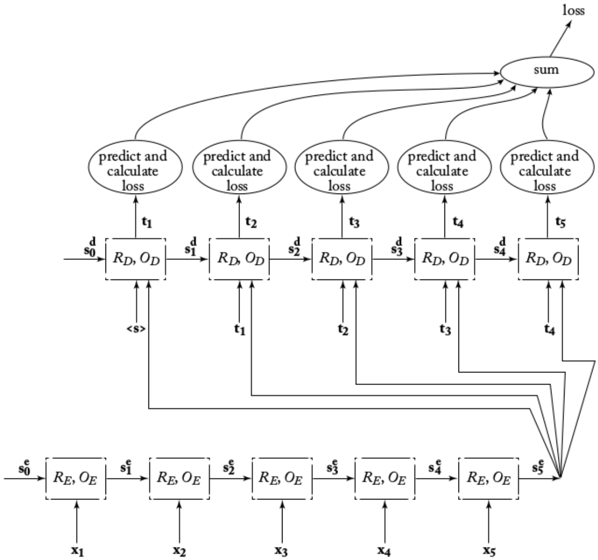
Encoder Decoder Framework



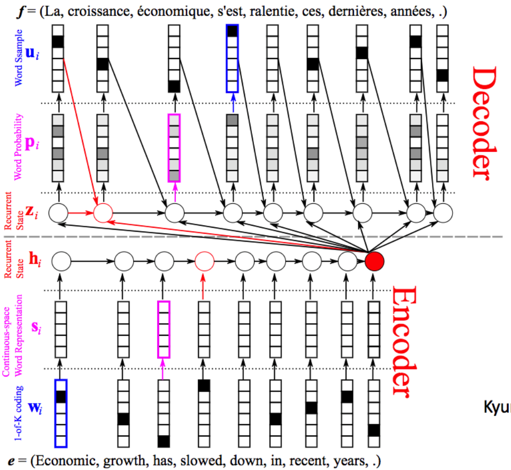
Conditioned Generation

- This setup is useful for mapping sequences of length n to sequences of length m.
- The encoder summarizes the source sentence as a vector $\vec{c}$.
- The decoder RNN is then used to predict (using a language modeling objective) the target sequence words conditioned on the previously predicted words as well as the encoded sentence $\vec{c}$.
- The encoder and decoder RNNs are trained jointly.
- The supervision happens only for the decoder RNN, but the gradients are propagated all the way back to the encoder RNN.

Sequence to Sequence Training Graph

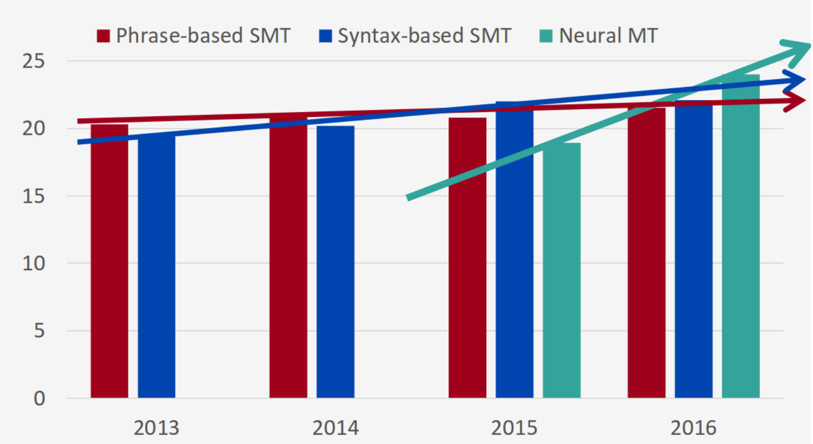


Neural Machine Translation



Kyunghyun Cho et al. 2014

Machine Translation BLEU progress over time



[Edinburgh En-De WMT]

[0]source: `http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf`

Decoding Approaches

- The decoder aims to generate the output sequence with maximal score (or maximal probability), i.e., such that $\sum_{i=1}^{n} P(\hat{t}_i|\hat{t}_{1:i-1})$ is maximized.
- The non-markovian nature of the RNN means that the probability function cannot be decomposed into factors that allow for exact search using standard dynamic programming.
- Exact search: finding the optimum sequence requires evaluating every possible sequence (computationally prohibitive).
- Thus, it only makes sense to solving the optimization problem above approximately.
- Greedy search: choose the highest scoring prediction (word) at each step.
- This may result in sub-optimal overall probability leading to prefixes that are followed by low-probability events.

Greedy Search

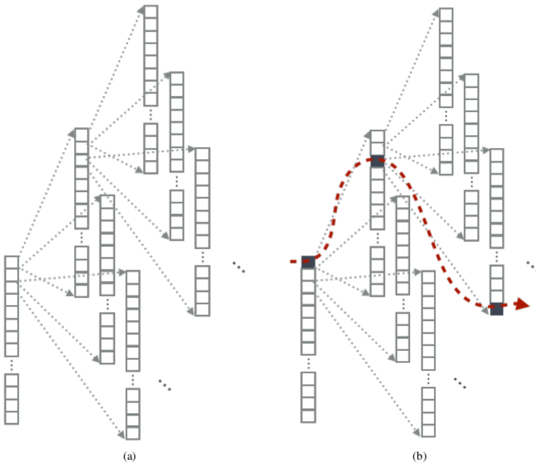


Figure 6.4: (a) Search space depicted as a tree. (b) Greedy search.

[0] Source: [Cho, 2015]

Beam Search

- Beam search interpolates between the exact search and the greedy search by changing the size K of hypotheses maintained throughout the search procedure [Cho, 2015].
- The Beam search algorithm works in stages.
- We first pick the K starting words with the highest probability
- At each step, each candidate sequence is expanded with all possible next steps.
- Each candidate step is scored.
- The K sequences with the most likely probabilities are retained and all other candidates are pruned.
- The search process can halt for each candidate separately either by reaching a maximum length, by reaching an end-of-sequence token, or by reaching a threshold likelihood.
- The sentence with the highest overall probability is selected.

[0] More info at: `https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/`

Conditioned Generation with Attention

- In the encoder-decoder networks the input sentence is encoded into a single vector, which is then used as a conditioning context for an RNN-generator.
- This architectures forces the encoded vector $\vec{c}$ to contain all the information required for generation.
- It doesn't work well for long sentences!
- It also requires the generator to be able to extract this information from the fixed-length vector.
- "You can't cram the meaning of a whole %&!$# sentence into a single $&!#* vector!" -Raymond Mooney
- This architecture can be can be substantially improved (in many cases) by the addition of an attention mechanism.
- The attention mechanism attempts to solve this problem by allowing the decoder to "look back" at the encoder's hidden states based on its current state.

Conditioned Generation with Attention

- The input sentence (a length n input sequence $\vec{x}_{1:n}$) is encoded using a biRNN as a sequence of vectors $\vec{c}_{1:n}$.
- The decoder uses a soft attention mechanism in order to decide on which parts of the encoding input it should focus.
- At each stage j the decoder sees a weighted average of the vectors $\vec{c}_{1:n}$, where the attention weights ($\vec{\alpha}^j$) are chosen by the attention mechanism.

$$\vec{c}^j = \sum_{i=1}^{n} \vec{\alpha}^j_{[i]} \cdot \vec{c}_i$$

- The elements of $\vec{\alpha}^j$ are all positive and sum to one.

Conditioned Generation with Attention

- Unnormalized attention weights ($\bar{\alpha}^{j}_{[i]}$) are produced taking into account the decoder state at time j ($\vec{s}_j$) and each of the vectors $\vec{c}_i$.
- They can be obtained in various ways, basically any differentiable function returning a scalar out of two vectors $\vec{s}_j$ and $\vec{c}_i$ could be employed.
- The simplest approach is a dot product: $\bar{\alpha}^{j}_{[i]} = \vec{s}_j \cdot \vec{c}_i$.
- The one we will use in these slides is Additive attention, which uses a Multilayer Perceptron: $\bar{\alpha}^{j}_{[i]} = MLP^{att}([\vec{s}_j; \vec{c}_i]) = \vec{v} \cdot \mathtt{tanh}([\vec{s}_j; \vec{c}_i]U + \vec{b})$

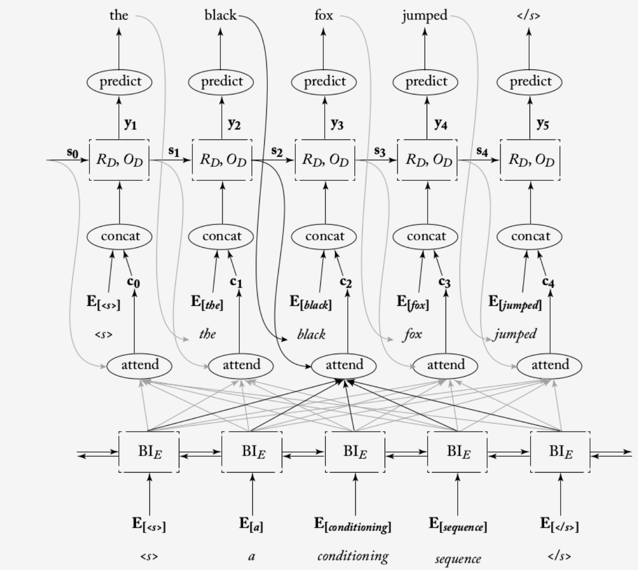
Conditioned Generation with Attention

- These unnormalized weights are then normalized into a probability distribution using the softmax function.

$$
\begin{aligned}
\operatorname{attend}(\boldsymbol{c}_{\mathbf{1}:\boldsymbol{n}}, \hat{t}_{1:j}) &= \boldsymbol{c}^{\boldsymbol{j}} \\
\boldsymbol{c}^{\boldsymbol{j}} &= \sum_{i=1}^{n} \boldsymbol{\alpha}^{\boldsymbol{j}}_{[\boldsymbol{i}]} \cdot \boldsymbol{c}_{\boldsymbol{i}} \\
\boldsymbol{\alpha}^{\boldsymbol{j}} &= \operatorname{softmax}(\bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\mathbf{1}]}, \ldots, \bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\boldsymbol{n}]}) \\
\bar{\boldsymbol{\alpha}}^{\boldsymbol{j}}_{[\boldsymbol{i}]} &= \mathrm{MLP}^{\mathrm{att}}([\boldsymbol{s}_{\boldsymbol{j}}; \boldsymbol{c}_{\boldsymbol{i}}]),
\end{aligned}
$$

- The encoder, decoder, and attention mechanism are all trained jointly in order to play well with each other.

Attention



the
black
fox
jumped
</s>
predict
predict
predict
predict
predict
$\mathbf{y}_1$
$\mathbf{y}_2$
$\mathbf{y}_3$
$\mathbf{y}_4$
$\mathbf{y}_5$
$\mathbf{s}_0$
R_D, O_D
$\mathbf{s}_1$
R_D, O_D
$\mathbf{s}_2$
R_D, O_D
$\mathbf{s}_3$
R_D, O_D
$\mathbf{s}_4$
R_D, O_D
concat
concat
concat
concat
concat
$\mathbf{c}_0$
$\mathbf{c}_1$
$\mathbf{c}_2$
$\mathbf{c}_3$
$\mathbf{c}_4$
$\mathbf{E}_{[<s>]}$
$\mathbf{E}_{[the]}$
$\mathbf{E}_{[black]}$
$\mathbf{E}_{[fox]}$
$\mathbf{E}_{[jumped]}$
<s>
the
black
fox
jumped
attend
attend
attend
attend
attend
BI_E
BI_E
BI_E
BI_E
BI_E
$\mathbf{E}_{[<s>]}$
$\mathbf{E}_{[a]}$
$\mathbf{E}_{[conditioning]}$
$\mathbf{E}_{[sequence]}$
$\mathbf{E}_{[</s>]}$
<s>
a
conditioning
sequence
</s>

Conditioned Generation with Attention

The entire sequence-to-sequence generation with attention is given by:

$$
\begin{aligned}
p(t_{j+1} = k \mid \hat{t}_{1:j}, \boldsymbol{x}_{1:n}) &= f(O_{\text{dec}}(\boldsymbol{s}_{j+1})) \\
\boldsymbol{s}_{j+1} &= R_{\text{dec}}(\boldsymbol{s}_j, [\hat{\boldsymbol{t}}_j; \boldsymbol{c}^j]) \\
\boldsymbol{c}^j &= \sum_{i=1}^{n} \boldsymbol{\alpha}^j_{[i]} \cdot \boldsymbol{c}_i \\
\boldsymbol{c}_{1:n} &= \text{biRNN}^{\star}_{\text{enc}}(\boldsymbol{x}_{1:n}) \\
\boldsymbol{\alpha}^j &= \text{softmax}(\bar{\boldsymbol{\alpha}}^j_{[1]}, \ldots, \bar{\boldsymbol{\alpha}}^j_{[n]}) \\
\bar{\boldsymbol{\alpha}}^j_{[i]} &= \text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_i]) \\
\hat{t}_j &\sim p(t_j \mid \hat{t}_{1:j-1}, \boldsymbol{x}_{1:n}) \\
f(\boldsymbol{z}) &= \text{softmax}(\text{MLP}^{\text{out}}(\boldsymbol{z}))
\end{aligned}
$$

$$
\text{MLP}^{\text{att}}([\boldsymbol{s}_j; \boldsymbol{c}_i]) = \boldsymbol{v} \tanh([\boldsymbol{s}_j; \boldsymbol{c}_i]\boldsymbol{U} + \boldsymbol{b}).
$$

Conditioned Generation with Attention

- Why use the biRNN encoder to translate the conditioning sequence $\vec{x}_{1:n}$ into the context vectors $\vec{c}_{1:n}$?
- Why we just don't attend directly on the inputs (word embeddings) $MLP^{att}([\vec{s}_j; \vec{x}_i])$?
- We could, but we get important benefits from the encoding process.
- First, the biRNN vectors $\vec{c}_i$ represent the items $\vec{x}_i$ in their sentential context.
- Sentential context: a window focused around the input item $\vec{x}_i$ and not the item itself.
- Second, by having a trainable encoding component that is trained jointly with the decoder, the encoder and decoder evolve together.
- Hence, the network can learn to encode relevant properties of the input that are useful for decoding, and that may not be present at the source sequence $\vec{x}_{1:n}$ directly.

Attention and Word Alignments

- In the context of machine translation, one can think of MLP^{att} as computing a soft alignment between the current decoder state $\vec{s}_j$ (capturing the recently produced foreign words) and each of the source sentence components $\vec{c}_i$.

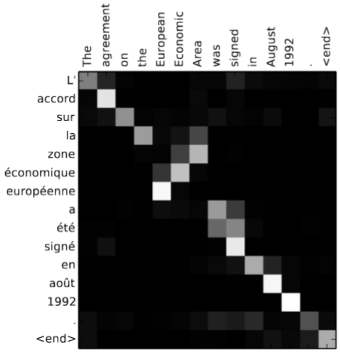


Fig. 2. Visualization of the attention weights α_j^t of the attention-based neural machine translation model [32]. Each row corresponds to the output symbol, and each column the input symbol. Brighter the higher α_j^t.

Figure: Source: [Cho et al., 2015]

Other types of Attention

Summary

Below is a summary table of several popular attention mechanisms (or broader categories of attention mechanisms).

Name	Alignment score function	Citation
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment max to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017
Self-Attention(&)	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.	Cheng2016
Global/Soft	Attending to the entire input state space.	Xu2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu2015; Luong2015

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.
(&) Also, referred to as "intra-attention" in Cheng et al., 2016 and some other papers.

Figure: Source: `https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`

Questions?

Thanks for your Attention!

References I

Cho, K. (2015).
Natural language understanding with distributed representation.
arXiv preprint arXiv:1511.07916.

Cho, K., Courville, A., and Bengio, Y. (2015).
Describing multimedia content using attention-based encoder-decoder networks.
IEEE Transactions on Multimedia, 17(11):1875–1886.