S&DS 365 / 665

Intermediate Machine Learning

Sequence-to-sequence models

November 28

Yale

Welcome back!

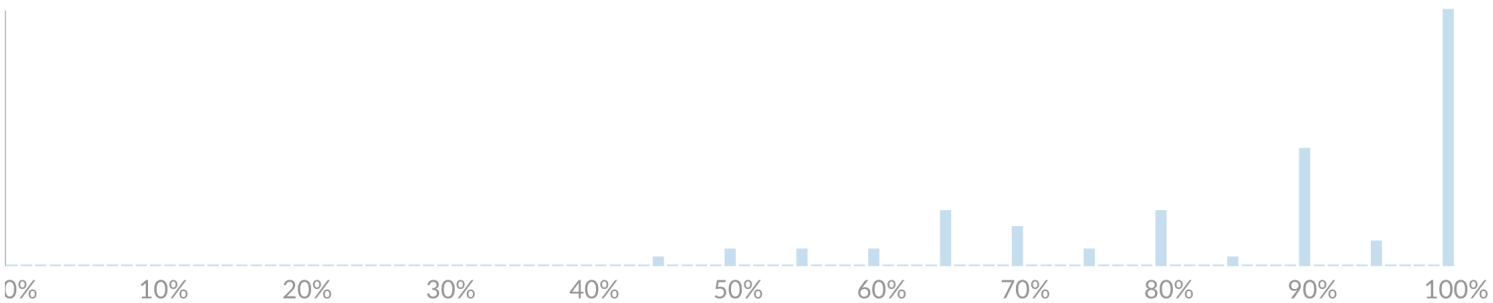
- Quiz 4 grades posted
- Quiz 5 (last!) on Wednesday, Nov 30 (RL, HMMs, RNNs, GRUs)
- Assn 5 out; due next Wednesday
- Final exam: Tuesday Dec 20, 7pm in LC 102
- Practice exams are posted
- Review sessions TBA

Quiz 4

Quiz Summary

Section Filter ▾ | Student Analysis | Item Analysis

Average Score	High Score	Low Score	Standard Deviation	Average Time
86%	100%	45%	1.56	26:08



Notes posted

- HMMs and Kalman filters
- Mixture models

Notes posted

12	Nov 14, 16	Sequential models	vanilla RNN Fakespeare GRU	Nov 14: HMMs and RNNs Nov 16: RNNs, GRUs, LSTMs, and all that	TensorFlow: Text generation Notes on HMMs PML Chapter 15	Nov 16: Assn 4 in Assn 5 out
13	Nov 21, 23	No class, Thanksgiving break				
14	Nov 28, 30	Attention and sequence-to-sequence models	GPT-3 demo Codex demo	Nov 28: Sequence-to-sequence models Nov 30: Attention	Notes on mixtures PML Sections 15.4, 15.5	Nov 30: Quiz 5

For Today

For Today

Sequence models

- Memory circuits: LSTMs and GRUs (recap)
- Sequence to sequence models
- Attention mechanisms
- Next: Transformers

Sequence models

- Generative process, any sequence (of words, characters, stock prices, nucleotides...) is assigned a probability

$$p(x_1, \ldots, x_n)$$

which can be factored as

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2 \mid x_1) \ldots p(x_n \mid x_1, \ldots, x_{n-1})$$

Sequence generation

- Items generated one-by-one
- Output at time t chosen by sampling from a probability distribution
- State h_t encodes "features" of sequence $x_1, x_2, \ldots, x_t$
- We talked about state models: HMMs and RNNs

Hidden Markov Model

In an HMM, current output generated from a latent variable.

- State is chosen stochastically (that is, probabilistically, or randomly)
- As a consequence, the dependence on earlier x_t's extends further back in time than just the previous time (as would be the case for a Markov model)

Recurrent neural network

In an RNN, current output generated from a distributed state—a vector of neurons

- State is a deterministic, a nonlinear function of previous state and current input symbol.
- Dependence on earlier x_t's is encoded in the state

Vanilla RNNs

In principle the state h_t can carry information from far in the past.

In practice, the gradients vanish (or explode) so this doesn't really happen

We need other mechanisms to "remember" information from far away and use it to predict future words

Vanilla RNNs

State is updated according to

$$h_t = \tanh\left(W_{hh}h_{t-1} + W_{hx}x_t + b_h\right)$$

We modify this with two types of “neural circuits” for storing information to be used downstream

Memory circuits

Intuition: In language modeling, may be useful to remember/forget gender or number of subject so that personal pronouns (“he” vs. ”she” vs. ”they”) can be used appropriately.

A variant called “Long Short-Term Memory” (LSTM) RNNs has a special hidden later that “includes” or “forgets” information from the past.

A simpler alternative to the LSTM circuit is called the *Gated Recurrent Unit* (GRU)

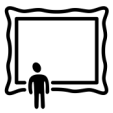
“Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”, Cho et al. 2014
https://arxiv.org/pdf/1406.1078v3.pdf

LSTMs and GRUs

Both LSTMs and GRUs have longer-range dependencies than vanilla RNNs.

We went through this in detail for GRUs, which are simpler, more efficient, and more commonly used

Gated recurrent units (GRUs)



High level idea:

- Learn when to update hidden state to “remember” important pieces of information
- Keep them in memory until they are used
- Reset or “forget” this information when no longer useful

GRUs

GRUs make use of “gates” denoted by Γ (Greek G for “Gate”)

$\Gamma = 1$: “the gate is open” and information flows through

$\Gamma = 0$: “the gate is closed” and information is blocked

GRUs

Two types of gates are used:

Γ^u: When open, information from long-term memory is propagated. When closed, information from local state is used.

Γ^r: When closed, the local state is reset. When open, the state is updated as in a “vanilla” RNN.

Note: These are usually called the “update” and “reset” gates. I find this terminology super confusing. More suggestive names might be the long distance memory gate (Γ^u) and the short distance memory gate (Γ^r).

GRUs

The state evolves according to

$$c_t = \tanh\left(W_{hx} x_t + W_{hh}\left(\Gamma_t^r \odot h_{t-1}\right) + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

GRUs

The state evolves according to

$$c_t = \tanh\left(W_{hx}x_t + W_{hh}\left(\Gamma_t^r \odot h_{t-1}\right) + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

- c_t is the "candidate state" computed using the usual "vanilla RNN" state, after possibly resetting some components.
- When the long-term memory gate is open ($\Gamma^u = 1$), the information gets sent through directly. *This deals with vanishing gradients*.
- The gates are multi-dimensional, applied componentwise
- Prediction of the next word is made using h_t.

GRUs

Everything needs to be differentiable, so the gate is actually "soft" and between zero and one.

The gates are computed as

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

where σ is the sigmoid function.

Putting it together

GRU state update equations

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

$$c_t = \tanh\left(W_{hx} x_t + W_{hh}\left(\Gamma_t^r \odot h_{t-1}\right) + b_h\right)$$

$$h_t = \left(1 - \Gamma_t^u\right) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

Putting it together

The reset gate is sometimes moved outside the linear map (Assn 5):

GRU state update equations

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

$$c_t = \tanh\left(W_{hx} x_t + \Gamma_t^r \odot W_{hh} h_{t-1} + b_h\right)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

GRUs: Example

Example:

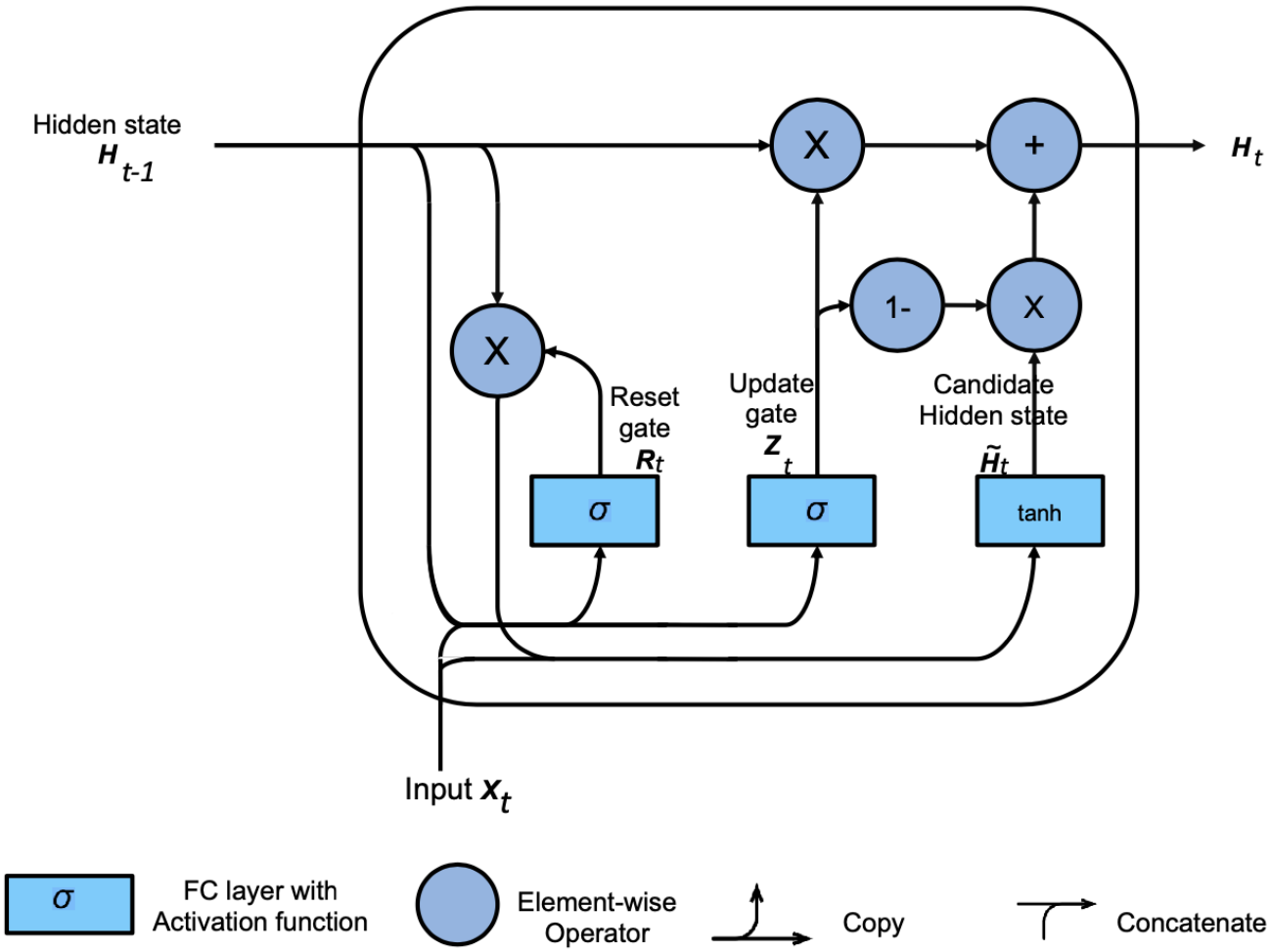
`The leaves, as the weather turned cold in New Haven, fell silently from the trees in November.`

We want to keep `leaves` in memory. It's the subject of the sentence, and plural — syntax

It also has a "foliage" meaning that is relevant when we predict the words `trees` and `November` — semantics

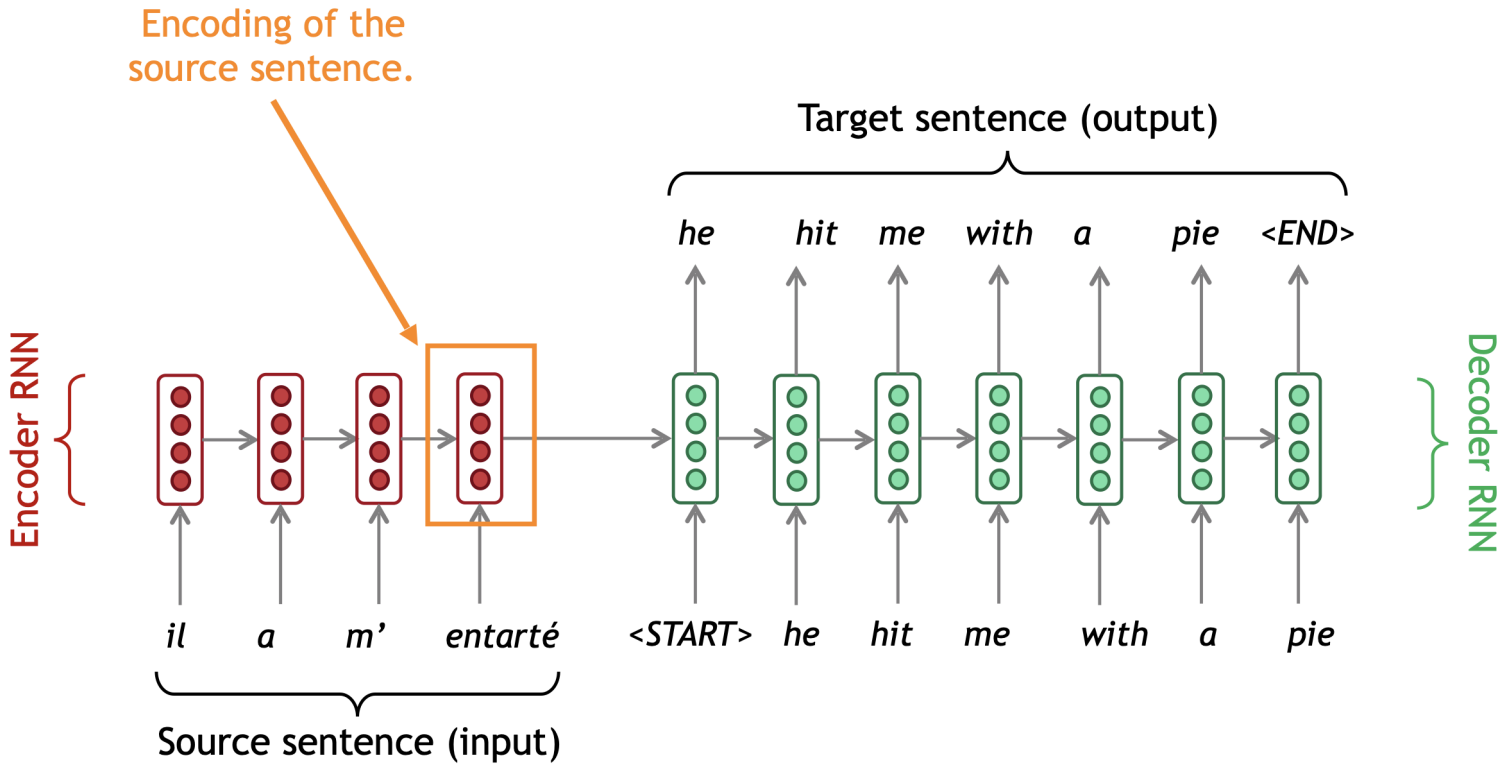
Let's step through on the blackboard how the GRU handles this.

GRU Diagram (using slightly different notation)



Sequence-to-sequence models

- Important in translation
- Uses two RNNs (GRUs or LSTMs): Encoder and Decoder



"Sequence to sequence learning with neural networks," Sutskever et al. 2014, https://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf. Figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Sequence-to-sequence models

- The goal of Seq2seq is to estimate the conditional probability

$$p(y_1, \ldots, y_T \mid x_1, \ldots, x_S)$$

- Encoder RNN computes the fixed dimensional representation $h(x)$; decoder RNN then computes

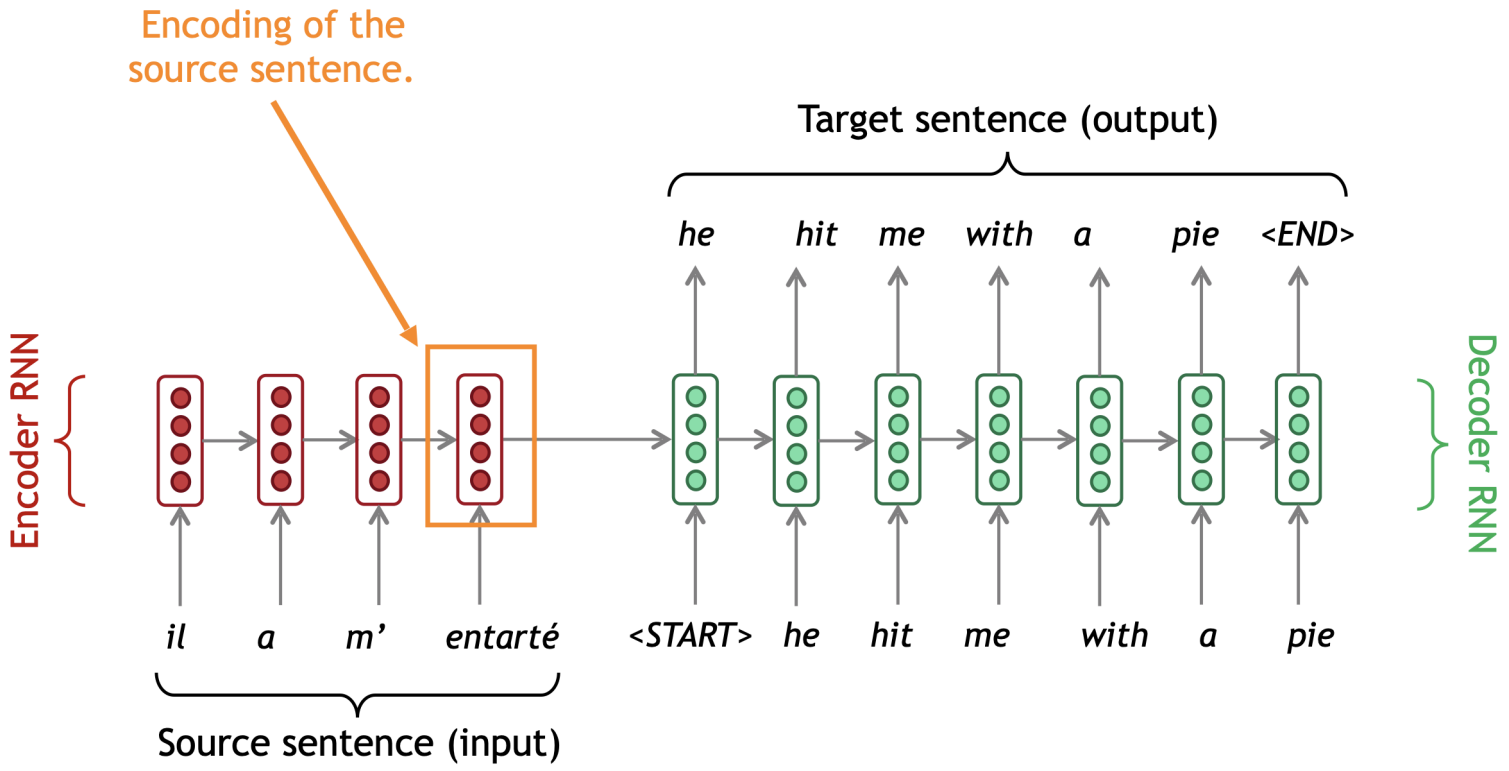
$$\prod_{t=1}^{T} p(y_t \mid h, y_1, \ldots, y_{t-1})$$

- Original paper uses 4-layer LSTMs with 1000 neurons in each layer; trained for 10 days.

Recall: This is very similar to what we did with LaTeX equation models conditioned on topics.

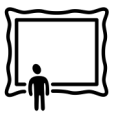
Sequence-to-sequence models

This results in a “bottleneck” problem—all the information needs to be funneled through that final state.

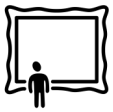


Important modification: Attention

Attention/Alignment

 On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

Attention/Alignment



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

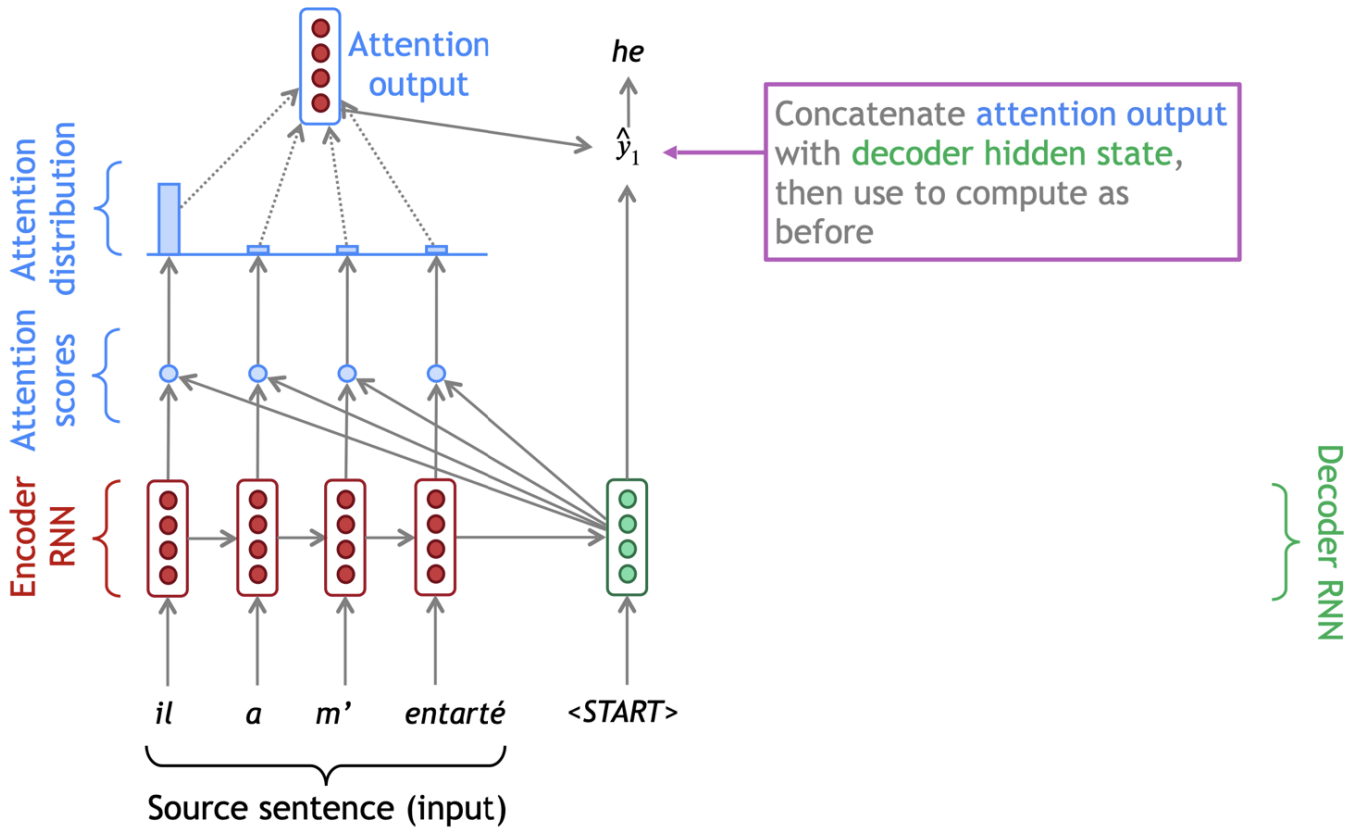
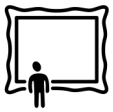


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

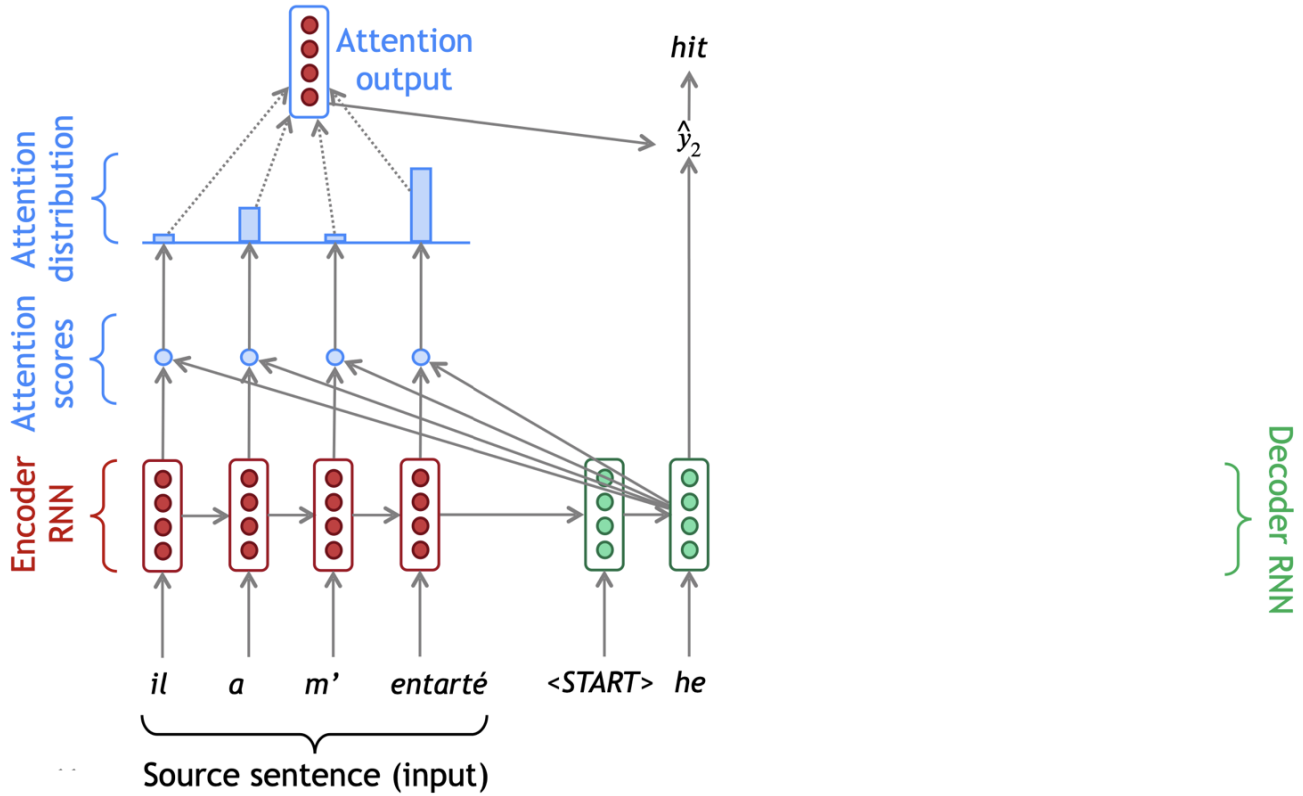
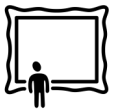


figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

Attention/Alignment



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

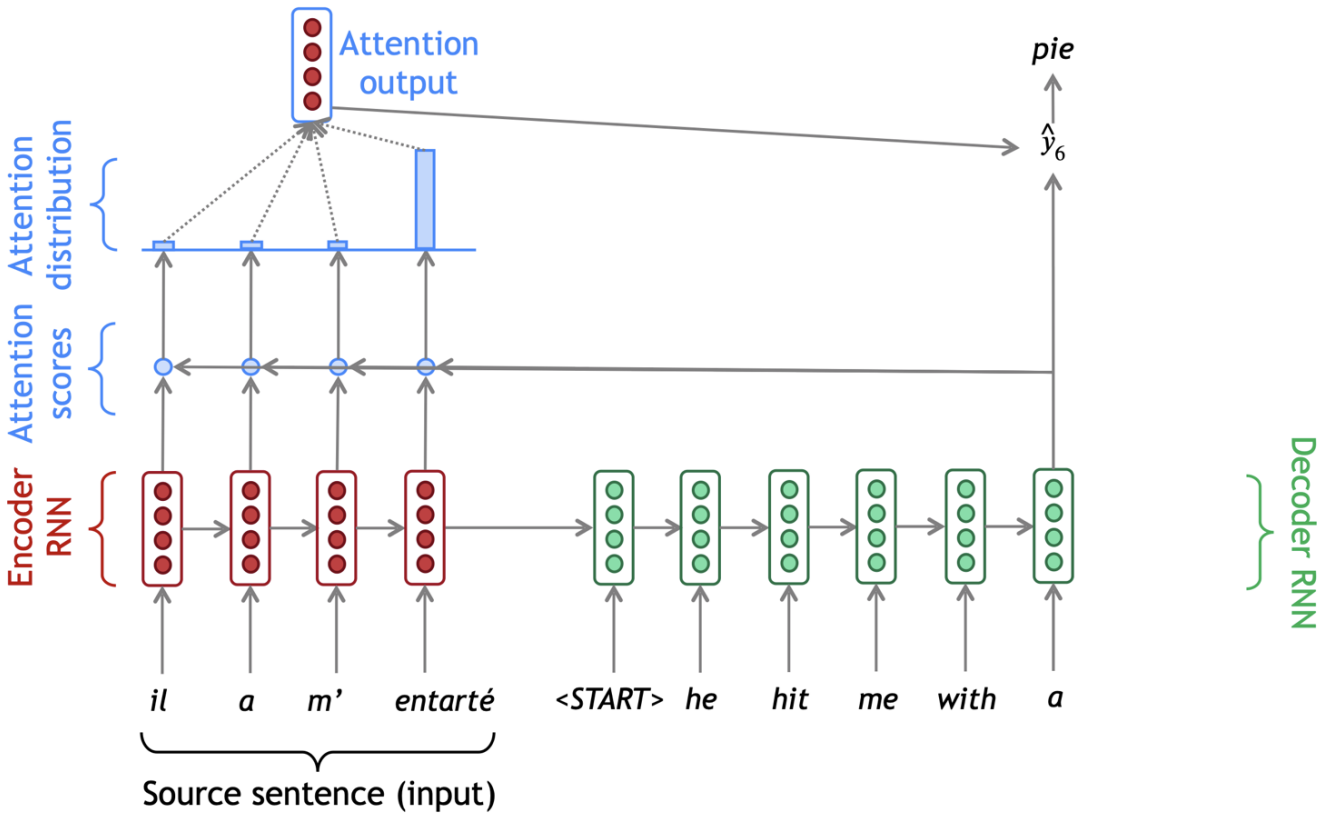
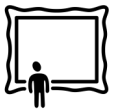


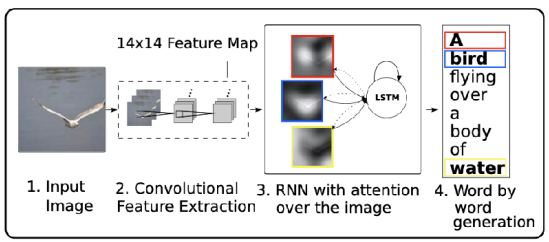
figure source: http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture08-nmt.pdf

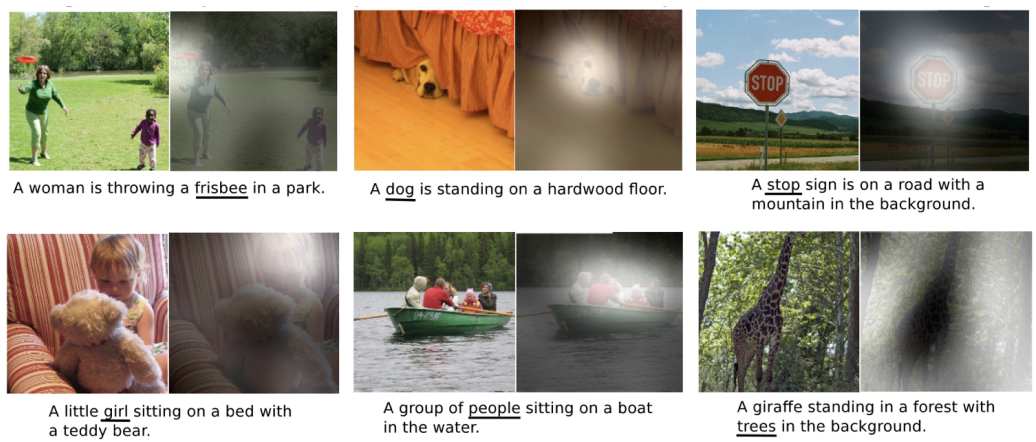
Attention/Alignment



On each step of decoding, directly connect to the encoder, and focus on a particular part of the source sequence

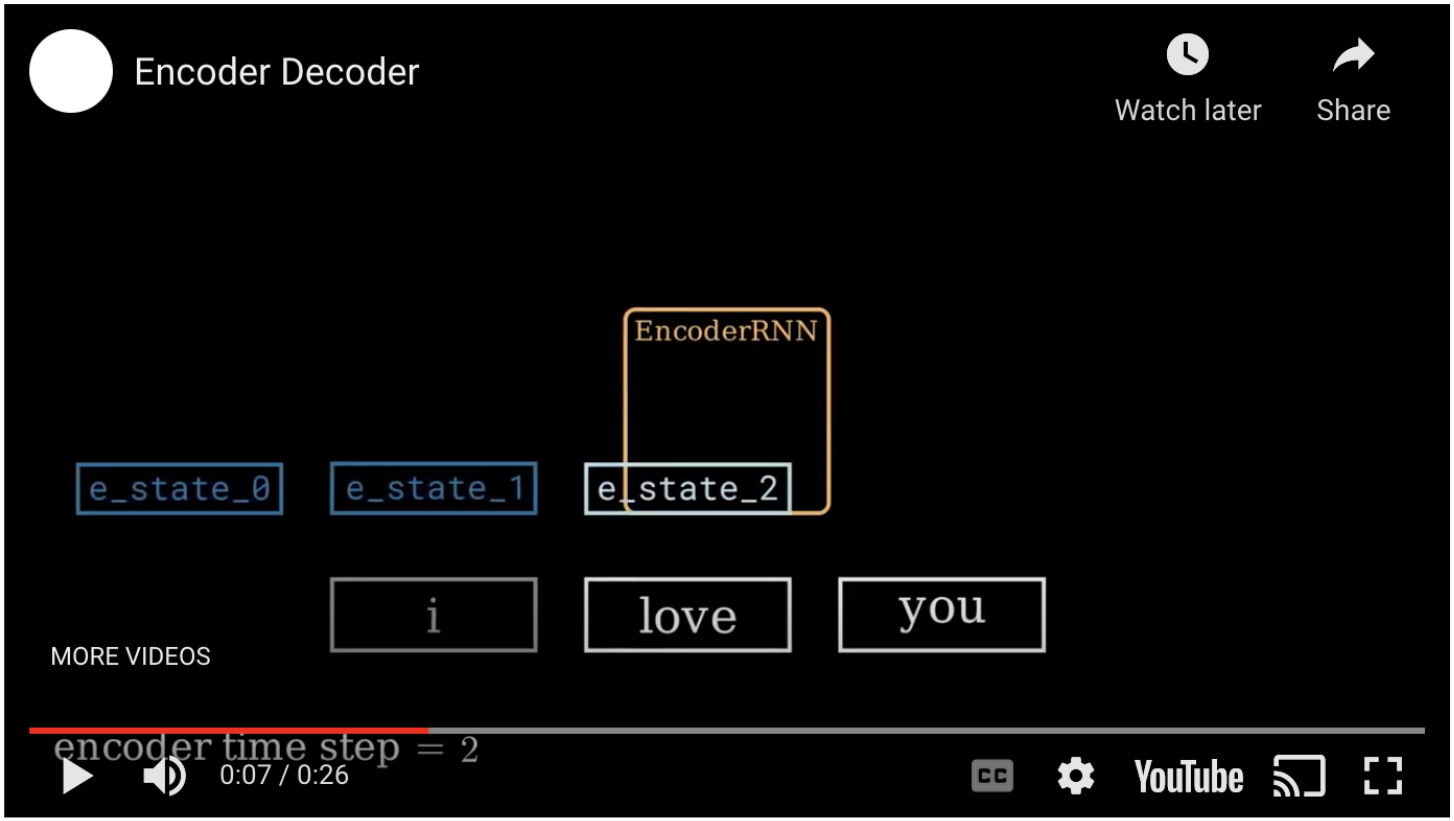
Attention can also be used for other generative models





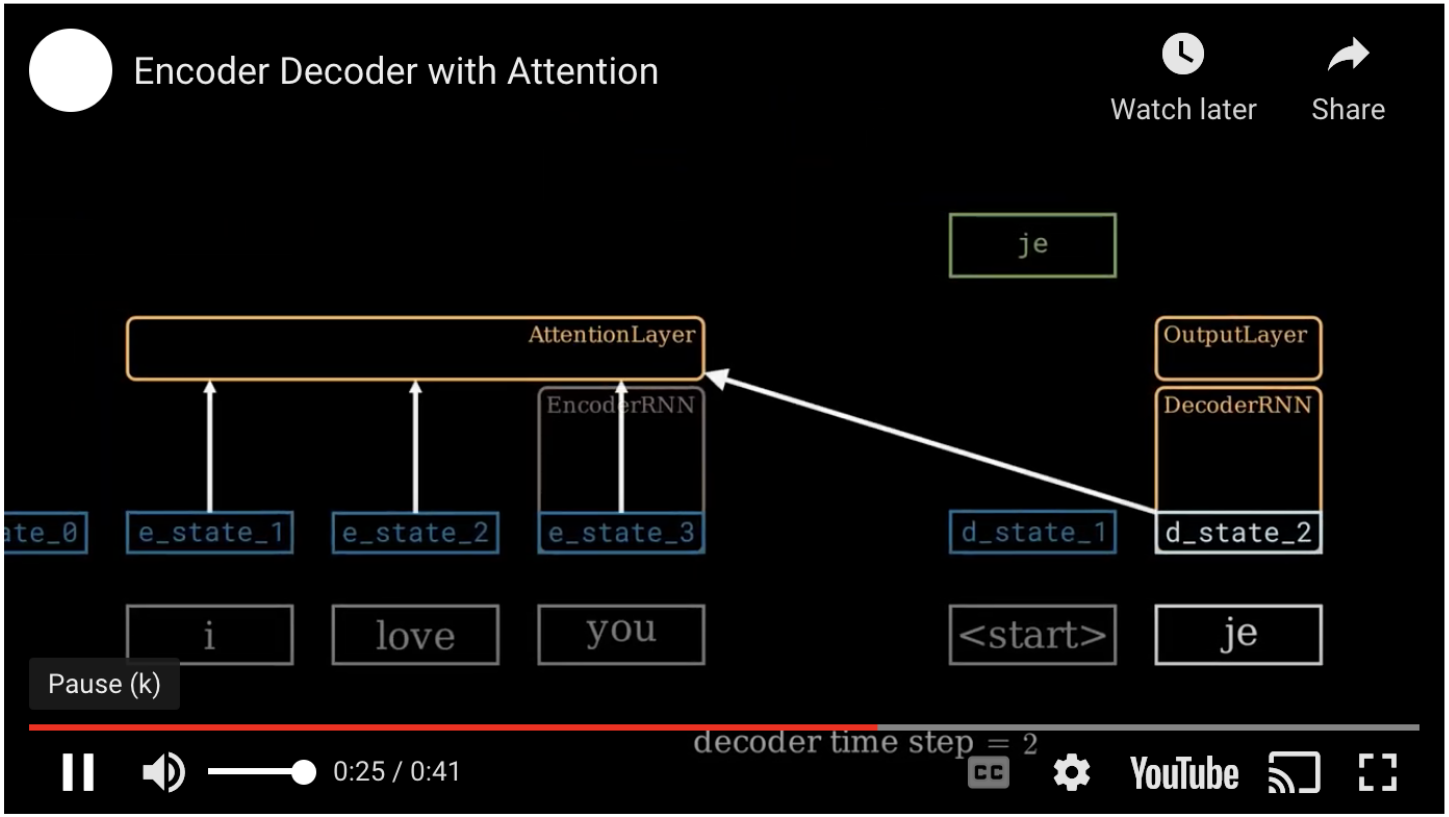
* "Show, attend and tell: neural image caption generation with visual attention", Xu et al. 2016, https://arxiv.org/pdf/1502.03044.pdf

Attention animated



Encoder's hidden state on the last times step becomes the initial state of Decoder. Note: <start> and <end> are special words/tokens to indicate the start and end of the sequence in Decoder. There's nothing special about <start> and <end> tokens other than their role as markers.

Attention animated



We use context vector combined with decoder hidden state to generate the final output in each time step.

https://medium.com/@cakraww/from-recurrent-neural-network-rnn-to-attention-explained-intuitively-7fd7caeef6af

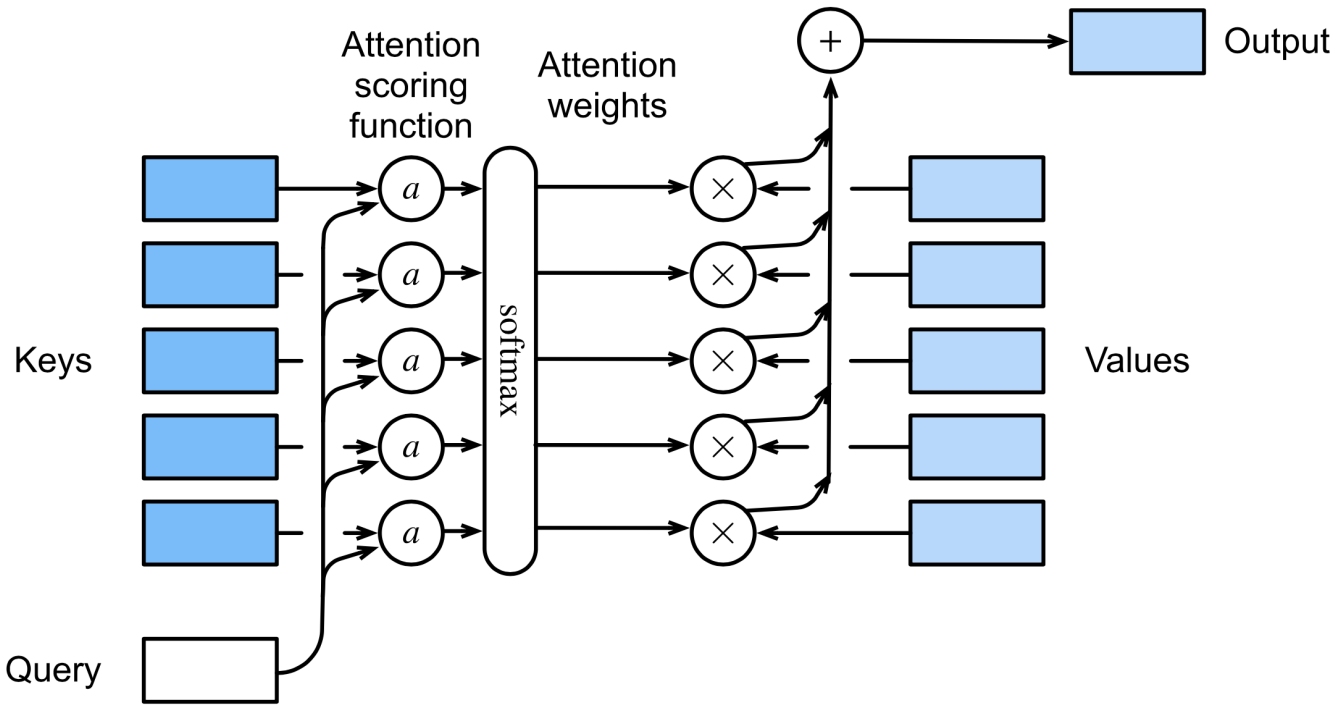
Attention in terms of query/key/values

Basic idea behind attention mechanisms:

- Neural networks so far: Hidden activations are linear combinations of input activations, followed by nonlinearity: $h = \varphi(Wu)$
- A more flexible model:
  - We have a set of m feature vectors or values $V \in \mathbb{R}^{m \times v}$
  - The model dynamically chooses which to use
  - based on how similar a query vector $q \in \mathbb{R}^q$ is to a set of m keys $K \in \mathbb{R}^{m \times k}$.
  - If q is most similar to key i, then we use value (feature) v_i.

Attention in terms of query/key/values

Basic idea behind attention mechanisms:



Attention in terms of query/key/values

Basic idea behind attention mechanisms:

$$\begin{aligned}\text{Attn}(q, \{(k_1, v_1), \ldots (k_m, v_m)\}) &= \text{Attn}(q, (k_{1:m}, v_{1:m})) \\ &= \sum_{i=1}^{m} \alpha(q, k_{1:m}) v_i\end{aligned}$$

where weights α_i are softmax of attention scores:

$$\alpha_i(q, k_{1:m}) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}$$

Kernel regression as attention

Recall the kernel regression estimator:

$$\widehat{m}(x) = \sum_{i=1}^{n} \alpha(x, x_{1:n}) y_i$$

Here the attention scores (for Gaussian kernel) are

$$a(x, x_i) = -\frac{1}{2h^2} \|x - x_i\|^2$$

Dot product attention

Note that if x_i and x have a fixed norm then the attention scores are just scaled dot products:

$$a(x, x_i) = \frac{1}{h^2} x^T x_i$$

If query and keys have same dimension d, dot product attention is

$$a(q, k) = q^T k / \sqrt{d} \in \mathbb{R}$$

and

$$\text{Attn}(Q, K, V) = \text{Softmax}\left(QK^T / \sqrt{d}\right) V$$

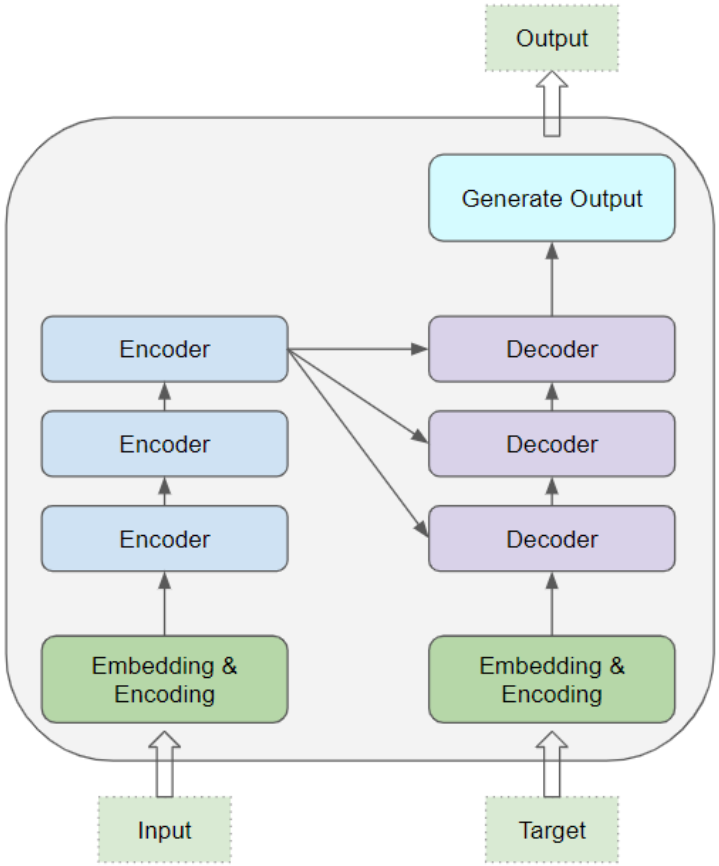
Scaling by $\sqrt{d}$ keeps variance constant

Transformers (next lecture)

The current state-of-the-art is based on *transfomers*

- Attention is the key ingredient
- Rather than processing sequences word-by-word, transformers handle larger chunks of text at once
- The distance between words matters less

Attention: example



https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Attention: example

https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Summary

- Seq2seq pairs together two RNNs, encoding the input sequence as a state, and decoding to generate an output sequence.
- Attention is a type of alignment between related words
- Attention allows dynamic construction of more informative states for predicting the next word
- Transformers process all words together, using layers of encoders and decoders, each with an attention component — details next time