Alternative Architectures

Philipp Koehn

15 October 2020



Alternative Architectures



- We introduced one translation model
 - attentional seq2seq model
 - core organizing feature: recurrent neural networks
- Other core neural architectures
 - convolutional neural networks
 - attention
- But first: look at various components of neural architectures



components

Components of Neural Networks



- Neural networks originally inspired by the brain
 - a neuron receives signals from other neurons
 - if sufficiently activated, it sends signals
 - feed-forward layers are roughly based on this
- Computation graph
 - any function possible, as long as it is partially differentiable
 - not limited by appeals to biological validity
- *Deep learning* maybe a better name

Feed-Forward Layer



- Classic neural network component
- Given an input vector *x*, matrix multiplication *M* with adding a bias vector *b*

Mx + b

• Adding a non-linear activation function

 $y = \operatorname{activation}(Mx + b)$

• Notation

$$y = FF_{\text{activation}}(x) = a(Mx + b)$$

Feed-Forward Layer

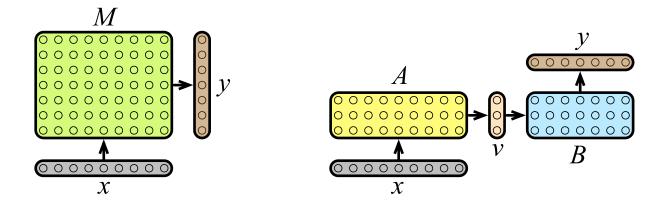


- Historic neural network designs: several feed-forward layers
 - input layer
 - hidden layers
 - output layer
- Powerful tools for a wide range of machine learning problems
- Matrix multiplication also called **affine transforms**
 - appeals to its geometrical properties
 - straight lines in input still straight lines in output

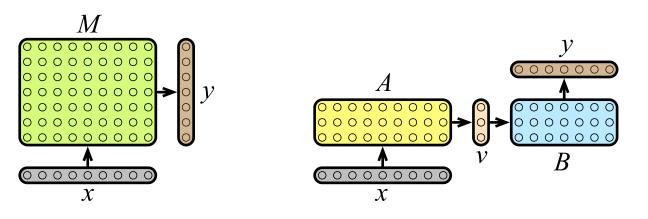
Factored Decomposition



- One challenge: very large input and output vectors
- Number of parameters in matrix $M = |x| \times |y|$
- \Rightarrow Need to reduce size of matrix
 - Solution: first reduce to smaller representation



Factored Decomposition: Math



- Intuition
 - given highly dimension vector x
 - first map to into lower dimensional vector v (matrix A)
 - then map to output vector y (matrix B)

v = Axy = Bv = BAx

- Example
 - $|x| = 20,000, |y| = 50,000 \rightarrow M = 1,000,000,000$
 - $|v| = 100 \rightarrow A = 20,000 \times 100 = 2,000,000, B = 100 \times 50,000 = 5,000,000$
 - reduction from 1,000,000,000 to 7,000,000



Factored Decomposition: Interpretation



- Vector *v* is a bottleneck feature
- Forced to captures salient features
- One example: word embeddings



basic mathematical operations

Concatenation



- Often multiple input vectors to processing step
- For instance recurrent neural network
 - input word
 - previous state
- Combined in feed-forward layer

 $y = \operatorname{activation}(M_1x_1 + M_2x_2 + b)$

• Another view

 $x = \operatorname{concat}(x_1, x_2)$ $y = \operatorname{activation}(Mx + b)$

• Splitting hairs here, but concatenation useful generally

Addition



- Adding vectors: very simplistic, but often done
- Example: compute sentence embeddings s from word embeddings $w_1, ..., w_n$

$$s = \sum_{i}^{n} w_i$$

- Reduces varying length sentence representation into fixed sized vector
- Maybe weight the words, e.g., by attention

Multiplication



- Another elementary mathematical operation
- Three ways to multiply vectors
 - element-wise multiplication

$$v \odot u = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \odot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} v_1 \times u_1 \\ v_2 \times u_2 \end{pmatrix} \blacksquare$$

dot product

$$v \cdot u = v^T u = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}^T \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = v_1 \times u_1 + v_2 \times u_2$$

used for simple version of attention mechanism

– third possibility: vu^T , not commonly done

Maximum



- Goal: reduce the dimensionality of representation
- Example: detect if a face is in image
 - any region of image may have positive match
 - represent different regions with element in a vector
 - maximum value: any region has a face
- Max pooling
 - given: *n* dimensional vector
 - goal: reduce to $\frac{n}{k}$ dimensional vector
 - method: break up vector into blocks of *k* elements, map each into single value

Max Out



- Max out
 - first branch out into multiple feed-forward layers

 $W_1 x + b_1$ $W_2 x + b_2$

– element-wise maximum

$$maxout(x) = max(W_1x + b_1, W_2x + b_2)$$

• ReLu activation is a maxout layer: maximum of feed-forward layer and 0

 $\operatorname{ReLu}(x) = \max(Wx + b, 0)$

15 October 2020

processing sequences



Recurrent Neural Networks



- Already described recurrent neural networks at length
 - propagate state *s*
 - over time steps *t*
 - receiving an input x_t at each turn

$$s_t = f(s_{t-1}, x_t)$$

(state may computed may as a feed-forward layer)

- More successful
 - gated recurrent units (GRU)
 - long short-term memory cells (LSTM)
- Good fit for sequences, like words in a sentence
 - humans also receive word by word
 - most recent words most relevant
 - $\rightarrow\,$ closer to current state
- But computational problematic: very long computation chains

Alternative Sequence Processing

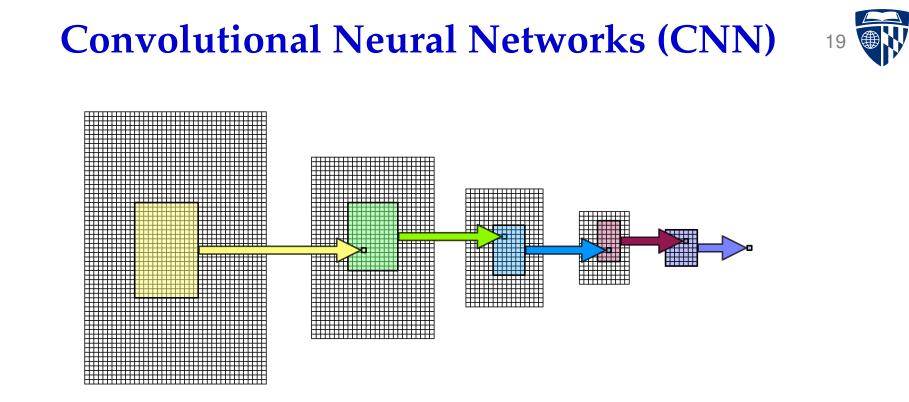


• Convolutional neural networks

• Attention



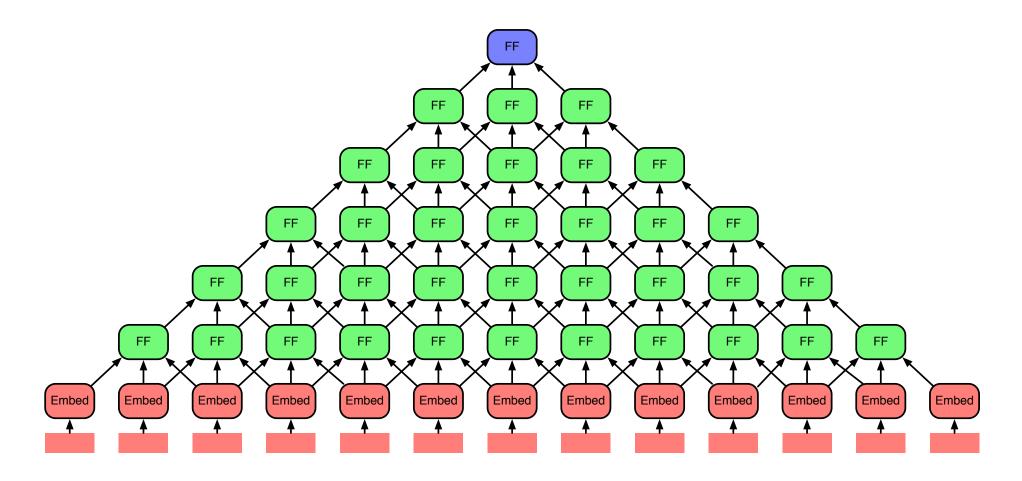
convolutional neural networks



- Popular in image processing
- Regions of an image are reduced into increasingly smaller representation
 - matrix spanning part of image reduced to single value
 - overlapping regions

CNNs for Language





• Map words into fixed-sized sentence representation

Hierarchical Structure and Language



- Syntactic and semantic theories of language
 - language is recursive
 - central: verb
 - dependents: subject, objects, adjuncts
 - their dependents: adjectives, determiners
 - also nested: relative clauses
- How to compute sentence embeddings active research topic

Convolutional Neural Networks



- Key step
 - take a high dimensional input representation
 - map to lower dimensional representation
- Several repetitions of this step
- Examples
 - map 50×50 pixel area into scalar value
 - combine 3 or more neighboring words into a single vector
- Machine translation
 - encode input sentence into single vector
 - decode this vector into a sentence in the output language



attention

Attention



- Machine translation is a structured prediction task
 - output is not a single label
 - output structure needs to be built, word by word
- Relevant information for each word prediction varies
- Human translators pay attention to different parts of the input sentence when translating
- \Rightarrow Attention mechanism

Computing Attention



- Attention mechanism in neural translation model (Bahdanau et al., 2015)
 - previous hidden state s_{i-1}
 - input word embedding h_j
 - trainable parameters b, W_a , U_a , v_a

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j + b)$$

- Other ways to compute attention
 - Dot product: $a(s_{i-1}, h_j) = s_{i-1}^T h_j$
 - Scaled dot product: $a(s_{i-1}, h_j) = \frac{1}{\sqrt{|h_i|}} s_{i-1}^T h_j$
 - General: $a(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$
 - Local: $a(s_{i-1}) = W_a s_{i-1}$

Attention of Luong et al. (2015)



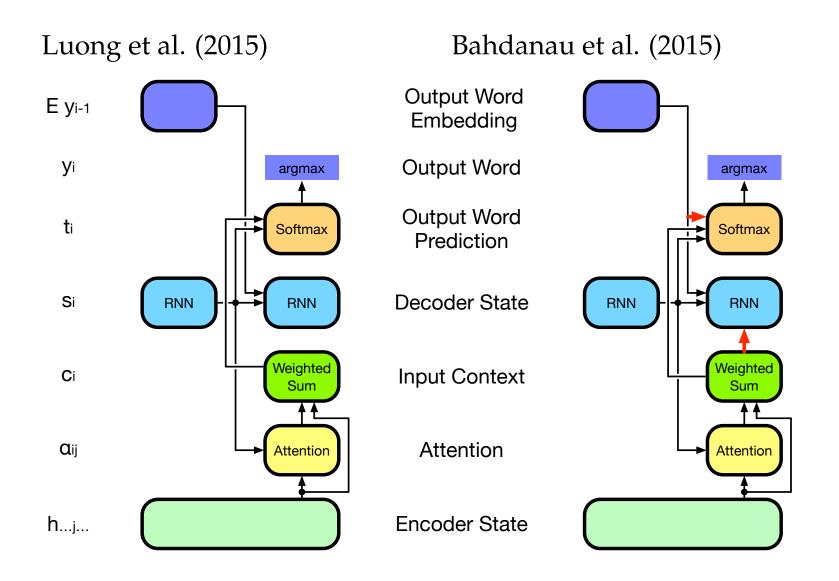
• Luong et al. (2015) demonstrate good results with the dot product

 $a(s_{i-1}, h_j) = s_{i-1}^T h_j$

- No trainable parameters
- Additional changes
- Currently more popular

Attention of Luong et al. (2015)





Attention of Luong et al. (2015)



Luong et al. (2015)

Attention $\alpha_{ij} = \text{softmax FF}(s_{i-1}, h_j)$

Input context $c_i = \sum_j \alpha_{ij} h_j$

Output word $p(y_t|y_{< t}, x) =$ $softmax(W FF_{tanh}(s_{i-1}, c_i))$

Decoder state $s_i = FF_{tanh}(s_{i-1}, Ey_{i-1})$ Bahdanau et al. (2015)

Attention $\alpha_{ij} = \text{softmax FF}(s_{i-1}, h_j)$

Input context $c_i = \sum_j \alpha_{ij} h_j$

Output word $p(y_t|y_{< t}, x) =$ $softmax(W FF_{tanh}(s_{i-1}, Ey_{i-1}, c_i))$

Decoder state $s_i = FF_{tanh}(s_{i-1}, Ey_{i-1}, c_i)$

Multi-Head Attention



- Add redundancy
 - say, 16 attention weights
 - each based on its own parameters
- Formally, for each head *k* compute an associated between
 - decoder state s_{i-1} at time step i
 - encoder state h_j for the *j*th input word
 - using the softmax of some parameterized function a^k

$$\alpha_{ij}^k = \text{softmax } a^k(s_{i-1}, h_j)$$

• Average the attention weights

$$\alpha_{ij} = \frac{1}{k} \sum_{k} \alpha_{ij}^{k}$$

• Multi-head attention is a form of ensembling

Fine-Grained Attention



- Why just use a single scalar value to weight entire vectors?
 - learn weights for each element
 - computation of attention values returns vector instead of scalar
- Architecturally, still a feed-forward neural network (or any of variants)

 $a(s_{i-1}, h_j) = \mathbf{F}\mathbf{F}^k(s_{i-1}, h_j)$

• Softmax is now applied over each dimension *d*

$$\alpha_{ij}^d = \frac{\exp a^d(s_{i-1}, h_j)}{\sum_k a^d(s_{i-1}, h_k)}$$

• Input context is now computed by a element-wise multiplication

$$c_i = \sum_j \alpha_{ij} \times h_j$$

Self Attention



- Finally, a very different take at attention
- Motivation so far: need for alignment between input words and output words
- Now: refine representation of input words in the encoder
 - representation of an input word mostly depends on itself
 - but also informed by the surrounding context
 - previously: recurrent neural networks (considers left or right context)
 - now: attention mechanism
- Self attention:

Which of the surrounding words is most relevant to refine representation?

Self Attention



• Formal definition (based on sequence of vectors h_j , packed into matrix H

self-attention(H) = softmax
$$\left(\frac{HH^T}{\sqrt{|h|}}\right)H$$

- Association between every word representation h_j any other context word h_k
 - computed by dot product
 - results in a vector of raw association values

HH^T

• Scaled by the size of the word representation vectors |h|, and softmax

softmax $\left(\frac{HH^T}{\sqrt{|h|}}\right)$

• Resulting vector of normalized association values used to weigh context words

Self Attention



- More familiar math, using word representation vectors h_j
- Raw association $\frac{HH^T}{\sqrt{|h|}}$ $a_{jk} = \frac{1}{|h|}h_jh_k^T$
- Normalized association (softmax)

$$\alpha_{jk} = \frac{\exp(a_{jk})}{\sum_{\kappa} \exp(a_{j\kappa})}$$

• Weighted sum

self-attention
$$(h_j) = \sum_k \alpha_{j\kappa} h_k$$

• More on this later (Transformer)

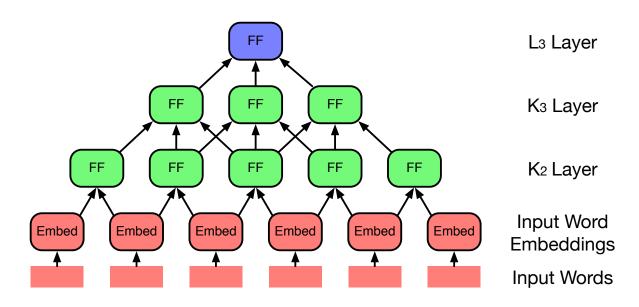


convolutional machine translation

Convolutional Machine Translation



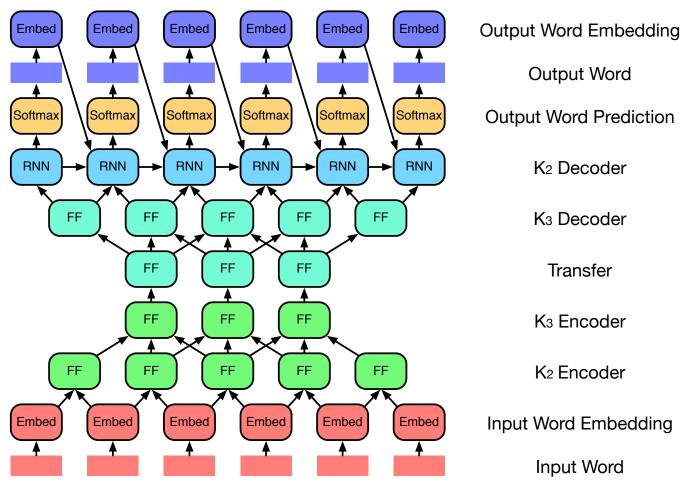
- First end-to-end neural machine translation model of the modern era [Kalchbrenner and Blunsom, 2013]
- Encoder



- always two convolutional layers, with different size
- here: K_2 and K_3
- Decoder similar

Refinement





- Convolutions do not result in a single sentence embedding but a sequence
- Decoder is also informed by a recurrent neural network

CNNs With Attention

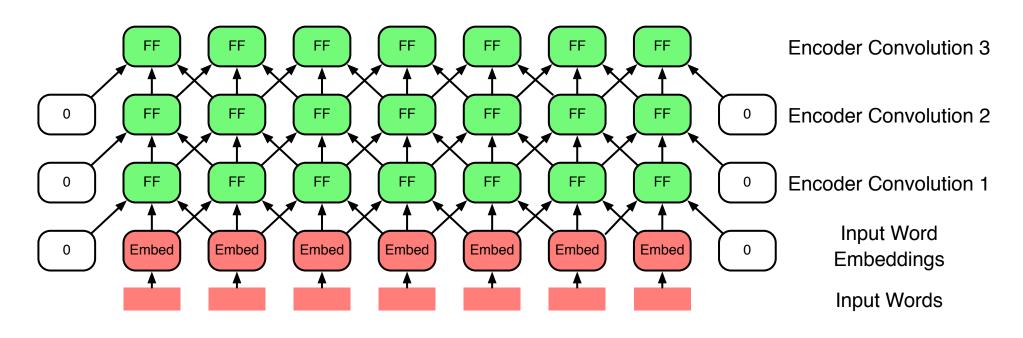


[Gehring et al. 2017]

- Combination of
 - convolutional neural networks
 - attention
- Sequence-to-sequence attention, mainly as before
- Recurrent neural networks replaced by convolutional layers

Encoder





- Stacked encoder convolutions
- Not shortening representations
- But: faster processing due to more parallelism

Encoder: Math



• Start with input word embeddings Ex_j

$$h_{0,j} = E x_j$$

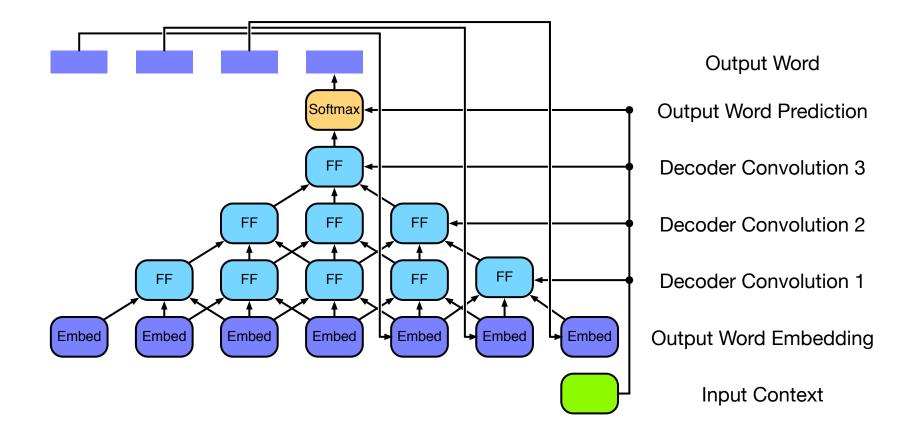
- Progress through
 - sequence of layer encodings $h_{d,j}$
 - at different depth \boldsymbol{d}
 - until maximum depth *D*

$$h_{d,j} = f(h_{d-1,j-k}, ..., h_{d-1,j+k})$$

- Details
 - function *f* is feed-forward layer with shortcut connection
 - final representation $h_{D,j}$ may only be informed by partial sentence context
 - all words at one depth can be processed in parallel \rightarrow fast

Decoder





- Decoder state computed by convolutional layers over previous output words
- Each convolutional state also informed by the input context (using attention)

Decoder: Math



• Recall: decoder recurrent neural network decoder

$$s_i = f(s_{i-1}, Ey_{i-1}, c_i)$$

- encoder state s_i
- embedding of previous output word Ey_{i-1}
- input context c_i
- Now
 - state computation not depending on previous state s_{i-1} (not recurrent)
 - conditioned on the sequence of the κ most recent previous words

$$s_i = f(Ey_{i-\kappa}, ..., Ey_{i-1}, c_i)$$

• Stacked convolutions

$$s_{1,i} = f(Ey_{i-\kappa}, ..., Ey_{i-1}, c_i)$$

$$s_{d,i} = f(s_{d-1,i-\kappa-1}, ..., s_{d-1,i}, c_i) \text{ for } d > 0, d \le \hat{D}$$

Attention



- Attention mechanism fundamentally unchanged
- Input context c_i computed based on association $a(s_{i-1}, h_j)$ between
 - encoder state h_j
 - decoder state s_{i-1}
- Now
 - encoder state $h_{D,j}$
 - decoder state $s_{\hat{D},i-1}$
- Refinement when computing the context vector c_i: shortcut connection between encoder state h_{D,j} and input word embedding x_j



transformer

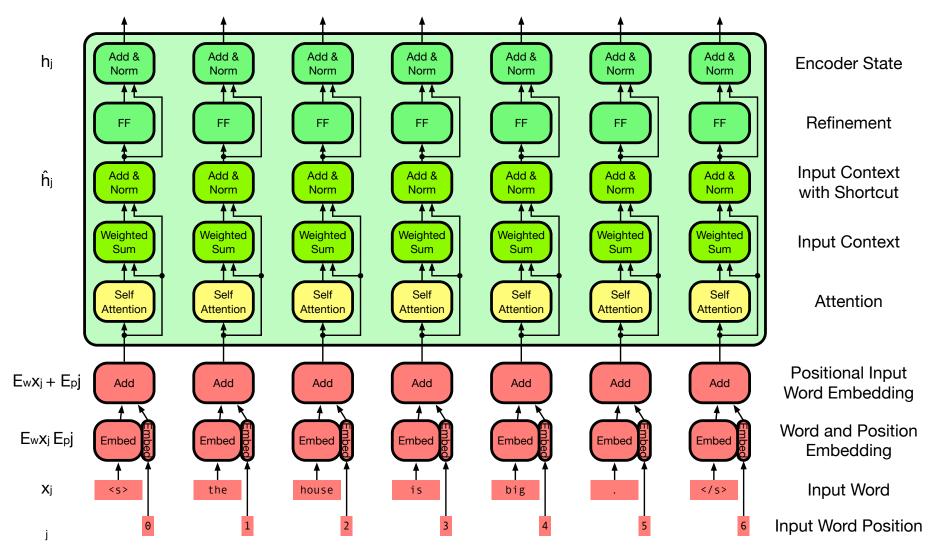
Self Attention: Transformer



- Self-attention in encoder
 - refine word representation based on relevant context words
 - relevance determined by self attention
- Self-attention in decoder
 - refine output word predictions based on relevant previous output words
 - relevance determined by self attention
- Also regular attention to encoder states in decoder
- Currently most successful model (maybe only with self attention in decoder, but regular recurrent decoder)

Encoder





Sequence of self-attention layers

Self Attention Layer



- Given: input word representations h_j , packed into a matrix $H = (h_1, ..., h_j)$
- Self attention

self-attention(H) = softmax
$$\left(\frac{HH^T}{\sqrt{|h|}}\right)H$$

• Shortcut connection

self-attention $(h_j) + h_j$

• Layer normalization

 $\hat{h}_j = \text{layer-normalization}(\text{self-attention}(h_j) + h_j)$

- Feed-forward step with ReLU activation function $\operatorname{relu}(W\hat{h}_i + b)$
- Again, shortcut connection and layer normalization

layer-normalization(relu($W\hat{h}_j + b$) + \hat{h}_j)

Stacked Self Attention Layers



- Stack several such layers (say, D = 6)
- Start with input word embedding

 $h_{0,j} = Ex_j$

• Stacked layers

 $h_{d,j} =$ self-attention-layer $(h_{d-1,j})$



Decoder Add & Norm Add & Decoder State Si Norm 1 t **≜**≜ 1 t 1 t 1 t 1 t _**≜** € FF FF FF FF FF FF Refinement Add & Norm Normalization Add & Ŝi Norm with Shortcut **↑**€ **↑**€ 1 t **↑**€ 1 t eight Sum Context Sum Sum Sum Sum Sum Sum **↑**€ 1 t Attention Attentio Attentio Attentio Attentio Attentio Encoder State Add & Norm Add & Norm Add & Norm Add & Norm Add & Add & Norm Add & **Output State** Norm Norm 44 4 4 **+ +** 44 **+ +** . FF FF FF FF Refinement FF Add & Norm Add & Norm Add & Norm Normalization Add & Norm Add & Norm Add & Add & Norm Norm with Shortcut 1 t 1 t 1 t 1 t 1 t 1 1 1 t Weighte Sum Weighted Sum Veighted Sum Veightee Sum Weighted Sum Weightee Sum Weighted Sum **Output Context** 1 t 1 t 1 t *** *** Self Self Self Self Self Self Self Self-Attention Attentior Attentio Attentio Attentior Attentio ttenti ttentio Positional Output Si Add Add Add Word Embedding 4 Word and Position Embe mbe Embeddina big Уi the is </s> Output Word <s> house Output Word Position

Decoder computes attention-based representations of the output in several layers, initialized with the embeddings of the previous output words

Self-Attention in the Decoder



- Same idea as in the encoder
- Output words are initially encoded by word embeddings $s_i = Ey_i$.
- Self attention is computed over previous output words
 - association of a word s_i is limited to words s_k ($k \le i$)
 - resulting representation $\tilde{s_i}$

self-attention(
$$\tilde{S}$$
) = softmax $\left(\frac{SS^T}{\sqrt{|h|}}\right)S$

Attention in the Decoder



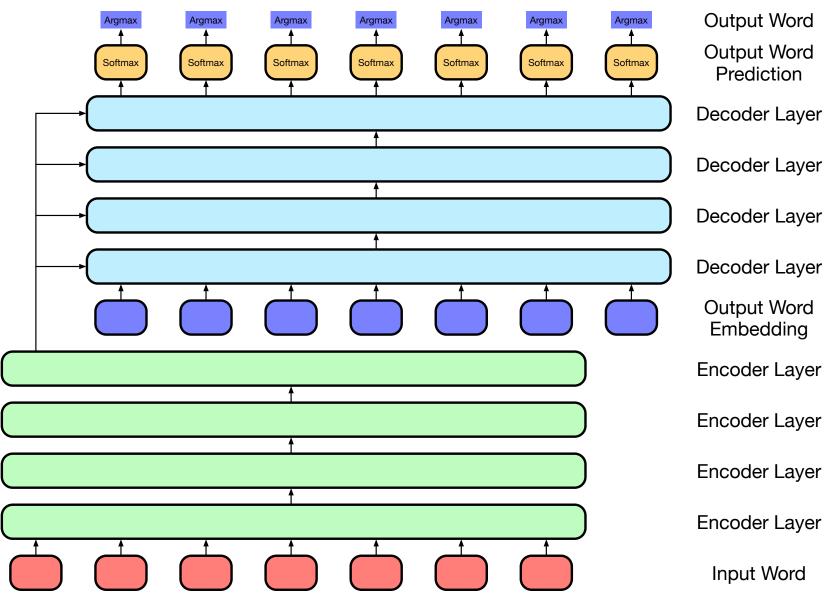
- Original intuition of attention mechanism: focus on relevant input words
- Computed with dot product $\tilde{S}H^T$
- Compute attention between the decoder states \tilde{S} and the final encoder states H

$$\operatorname{attention}(\tilde{S}, H) = \operatorname{softmax}\Big(\frac{\tilde{S}H^T}{\sqrt{|h|}}\Big)H$$

• Note: attention mechanism formally mirrors self-attention

Full Decoder





Full Decoder



• Self-attention

self-attention(
$$\tilde{S}$$
) = softmax $\left(\frac{SS^T}{\sqrt{|h|}}\right)S$

- shortcut connections
- layer normalization
- feed-forward layer
- Attention

attention
$$(\tilde{S}, H) = \operatorname{softmax}\left(\frac{\tilde{S}H^T}{\sqrt{|h|}}\right)H$$

- shortcut connections
- layer normalization
- feed-forward layer
- Multiple stacked layers

Mix and Match



- Encoder may be multiple layers of either
 - recurrent neural networks
 - self-attention layers
- Decoder may be multiple layers of either
 - recurrent neural networks
 - self-attention layers
- Also possible: self-attention encoder, recurrent neural network deocder
- Even better: both self-attention and recurrent neural network, merged at the end