• 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

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

• 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 \textit{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 $M$

- 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 \( B \))

\[
v = Ax \\
y = Bv = BAx
\]

• 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

\[ y = \text{activation}(M_1 x_1 + M_2 x_2 + b) \]

• Another view

\[ x = \text{concat}(x_1, x_2) \]
\[ y = \text{activation}(M x + b) \]

• Splitting hairs here, but concatenation useful generally
• Adding vectors: very simplistic, but often done

• Example: compute sentence embeddings $s$ from word embeddings $w_1, \ldots, w_n$

  \[ s = \sum_{i}^{n} w_i \]

• 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
    \[ v \odot u = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \odot \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} v_1 \times u_1 \\ v_2 \times u_2 \end{pmatrix} \]

  - 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
    \[ \text{maxout}(x) = \max(W_1 x + b_1, W_2 x + b_2) \]

- ReLu activation is a maxout layer: maximum of feed-forward layer and 0
  \[ \text{ReLu}(x) = \max(W x + b, 0) \]
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
  → closer to current state

• But computational problematic: very long computation chains
Alternative Sequence Processing

- Convolutional neural networks
- Attention
convolutional neural networks
Convolutional Neural Networks (CNN)

- 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
• 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 $50 \times 50$ 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

⇒ 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: $a(s_{i-1}, h_j) = \frac{1}{\sqrt{|h_j|}} s_{i-1}^T h_j$
  - 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
General View of Dot-Product Attention

- Three element
  - **Query**: decoder state
  - **Key**: encoder state
  - **Value**: encoder state

- Intuition
  - given a query (the decoder state)
  - we check how well it matches keys in the database (the encoder states)
  - and then use the matching score to scale the retrieved value (also the encoder state)

- Computation
  \[
  \text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V
  \]
Scaled Dot-Product Attention

- Refinement of query, key, and value
- Scale it down to lower-dimensional vectors (e.g., 512 from 4096)
- Using a weight matrix for each: $QW^Q$, $KW^K$, $VW^V$
Multi-Head Attention

- Add redundancy
  - say, 16 attention weights
  - each based on its own parameters

- Formally:

\[
\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)
\]

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O
\]

- Multi-head attention is a form of ensembling
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 $H$)

\[
\text{self-attention}(H) = \text{Attention}(HW_i^Q, HW_i^K, HW_i^V)
\]

- Association between every word representation $h_j$ any other context word $h_k$

- Resulting vector of normalized association values used to weigh context words
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
• 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
• Decoder state computed by convolutional layers over previous output words
• Each convolutional state also informed by the input context (using attention)
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)
Self Attention Layer

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

- Self attention
  \[
  \text{self-attention}(H) = \text{MultiHead}(H, H, H)
  \]

- Shortcut connection
  \[
  \text{self-attention}(h_j) + h_j
  \]

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

- Feed-forward step with ReLU activation function
  \[
  \text{relu}(W\hat{h}_j + b)
  \]

- Again, shortcut connection and layer normalization
  \[
  \text{layer-normalization}(\text{relu}(W\hat{h}_j + b) + \hat{h}_j)
  \]
Stacked Self Attention Layers

• Stack several such layers (say, $D = 6$)

• Start with input word embedding

\[ h_{0,j} = Ex_j \]

• Stacked layers

\[ h_{d,j} = \text{self-attention-layer}(h_{d-1,j}) \]
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 $s_i = E y_i$.

• Self attention is computed over previous output words
  - association of a word $s_i$ is limited to words $s_k$ ($k \leq i$)
  - resulting representation $\tilde{s}_i$

$$\text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S})$$
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 $H$

  $$\text{attention}(\tilde{S}, H) = \text{MultiHead}(\tilde{S}, H, H)$$

• Note: attention mechanism formally mirrors self-attention
Full Decoder

Output Word
Output Word Prediction
Decoder Layer
Decoder Layer
Decoder Layer
Decoder Layer
Output Word Embedding
Encoder Layer
Encoder Layer
Encoder Layer
Encoder Layer
Input Word

Argmax
Argmax
Argmax
Argmax
Argmax
Argmax
Argmax

Softmax
Softmax
Softmax
Softmax
Softmax
Softmax
Softmax

Embedding

Input Word

Encoder Layer
Encoder Layer
Encoder Layer
Encoder Layer
Full Decoder

- Self-attention
  \[ \text{self-attention}(\tilde{S}) = \text{MultiHead}(\tilde{S}, \tilde{S}, \tilde{S}) \]
  - shortcut connections
  - layer normalization
  - feed-forward layer

- Attention
  \[ \text{attention}(\tilde{S}, H) = \text{softmaxMultiHead}(\tilde{S}, H, H) \]
  - 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 decoder

• Even better: both self-attention and recurrent neural network, merged at the end