Deep Learning
Lecture 6

Thomas Hofmann, ETH Zurich

31 October 2019

Section 11

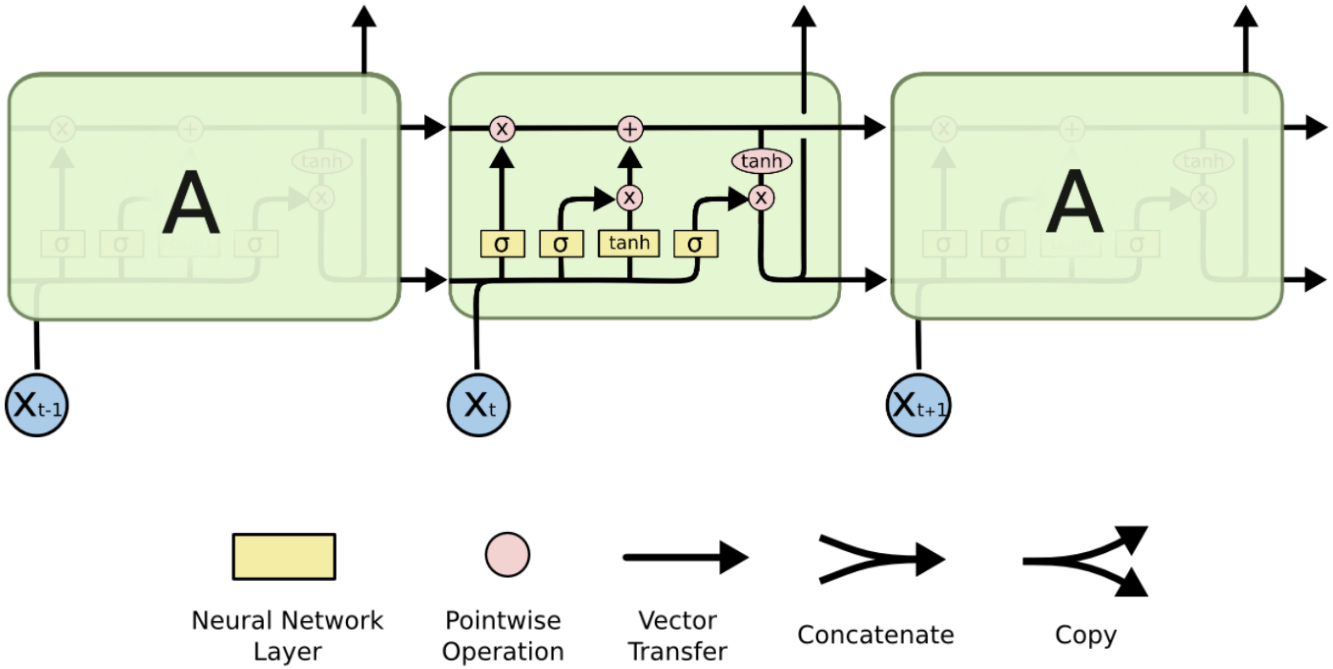
Differentiable Memory

Long-Term Dependencies

1. Sometimes: important to model long-term dependencies $\Longrightarrow$ network needs to memorize features from the distant past
2. Recurrent network: hidden state needs to preserve memory
3. Conflicts with short-term fluctuations and vanishing gradients
4. Conclusion: difficult to learn long-term dependencies with standard recurrent network (see DL Section 10.7)
5. Popular remedy: **gated units**

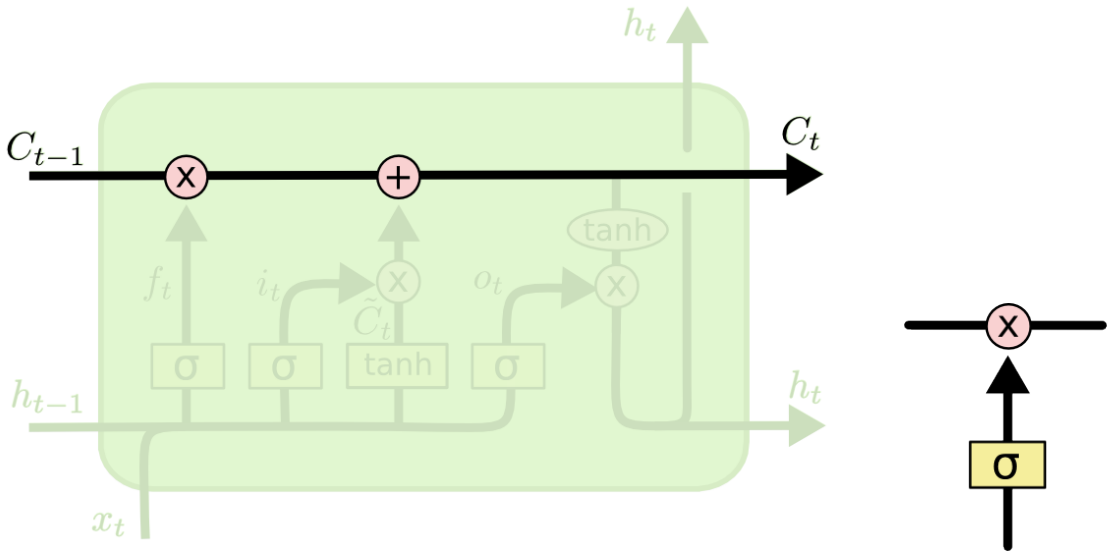
LSTM: Overall Architecture

Long-Short-Term-Memory: unit for memory management



from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

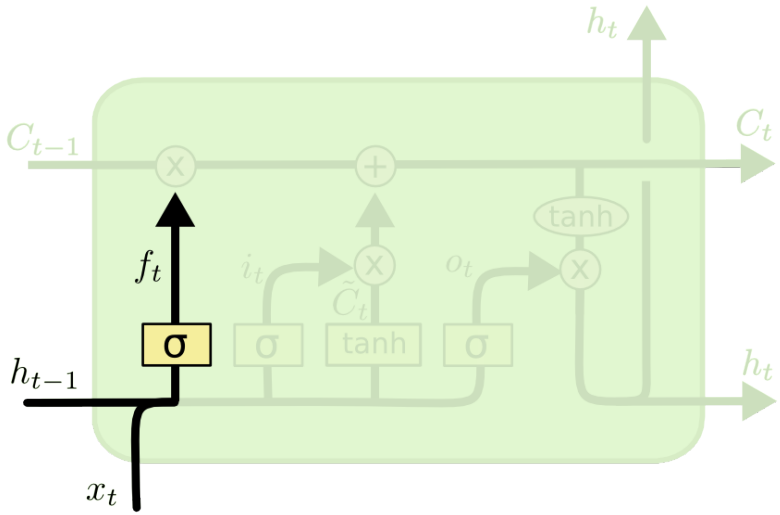
LSTM: Flow of Information



1. Information propagates along the chain like on a conveyor belt.
2. Information can flow unchanged and is only selectively changed (vector addition) by σ-gates.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Forget Gate

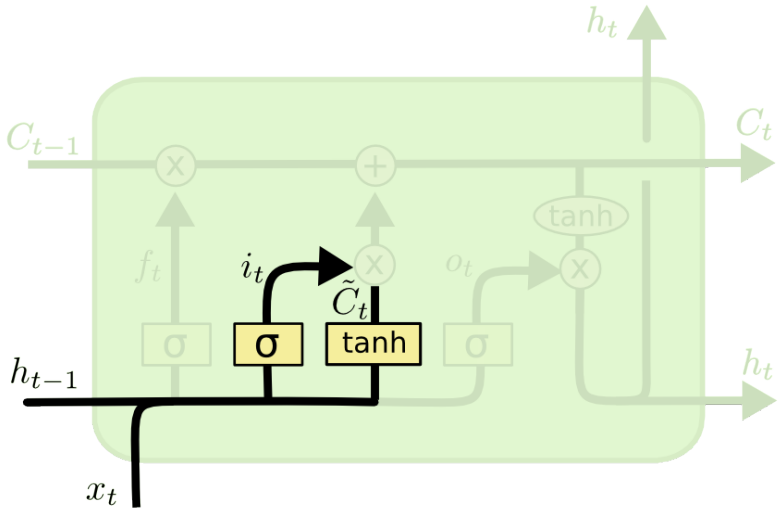


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

1. Keeping or forgetting of stored content?

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Input → Memory Value



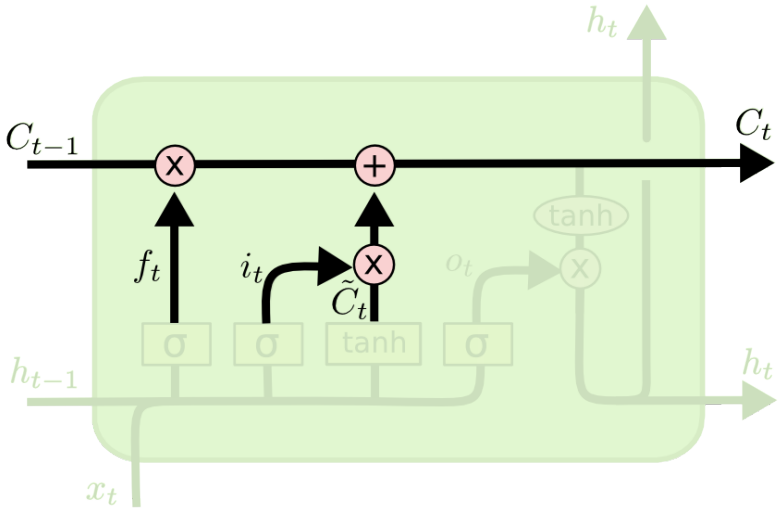
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

1. Preparing new input information to be added to the memory.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Updating Memory

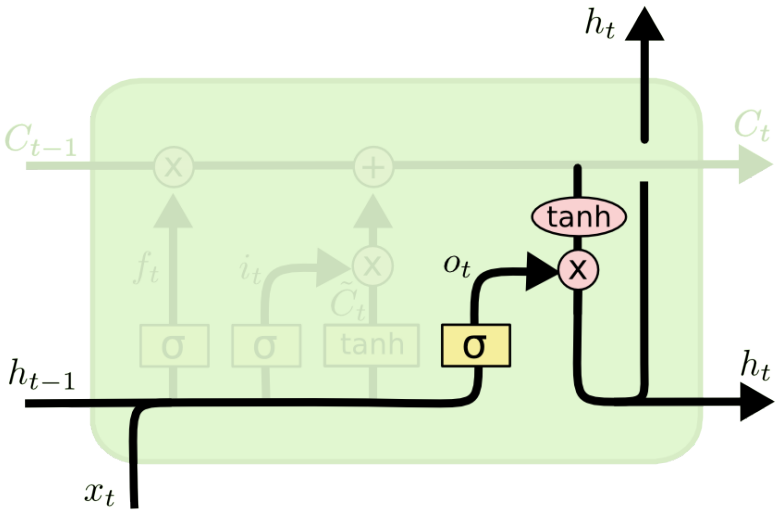


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1. Combining stored and new information.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Output Gate



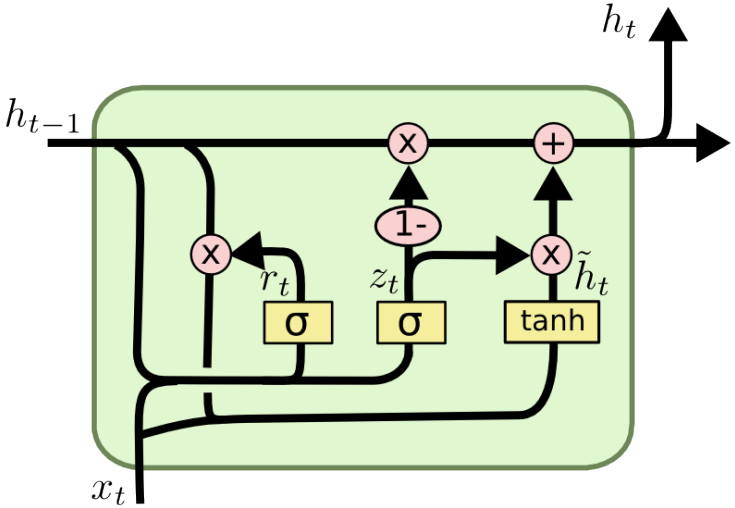
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1. Computing output selectively.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Gated Memory Units



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

1. Memory state = output. Modifications to logic (Cho et al, 2014).
2. Convex combination of old and new information.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Gated Memory Units

1. GRUs and LSTMs can learn active memory strategies: what to memorize, overwrite and recall when.
2. Successful use cases
   - handwriting recognition
   - speech recognition (also: Google)
   - machine translation
   - image captioning
3. Notoriously difficult to understand what units learn...

   Resource-hungry. Slow in learning.

Language Modeling

Model	Test Perplexity	Number of Params [billions]
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3	4.1
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6	1.76
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9	33
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (No Dropout)	37.9	3.3
LSTM-8192-2048 (50% Dropout)	32.2	3.3
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN Inputs	**30.0**	**1.04**
BIG LSTM+CNN Inputs + CNN Softmax	39.8	**0.29**
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8	**0.39**
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9	**0.23**

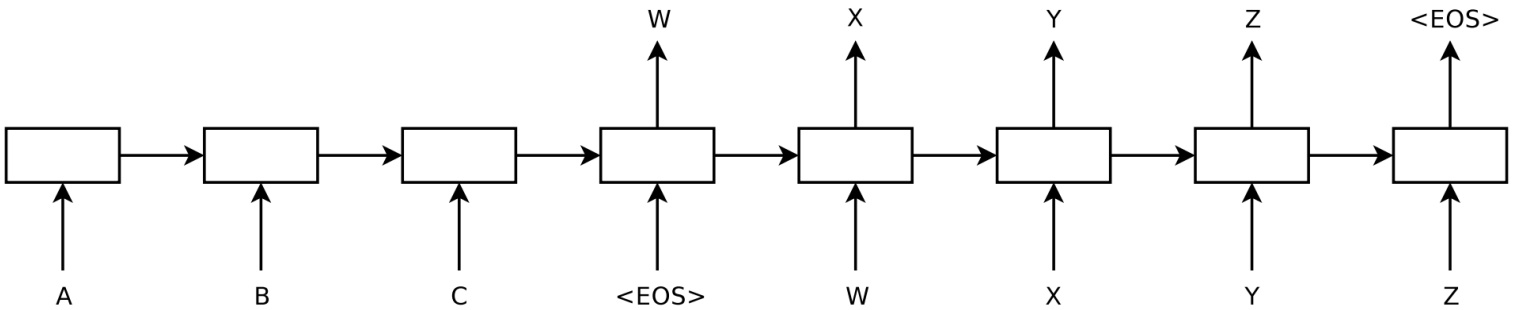
from Jozefowicz et al, 2016

1. evaluation on corpus w/ 1B words
2. number of parameters can be in the 100Ms or even Bs!
3. ensembles can reduce perplexity to ≈ 23 (best result 06/2016)

Sequence to Sequence Learning

1. Important use of memory units: **sequence to sequence learning**. Seminal paper: Sutskever, Vinyals & Le, 2014

2. **Encoder-decoder architecture**



Encode sequence (e.g. sentence) into vector

Decode sequence (e.g. translate) from vector (w/ autoregressive output feedback)

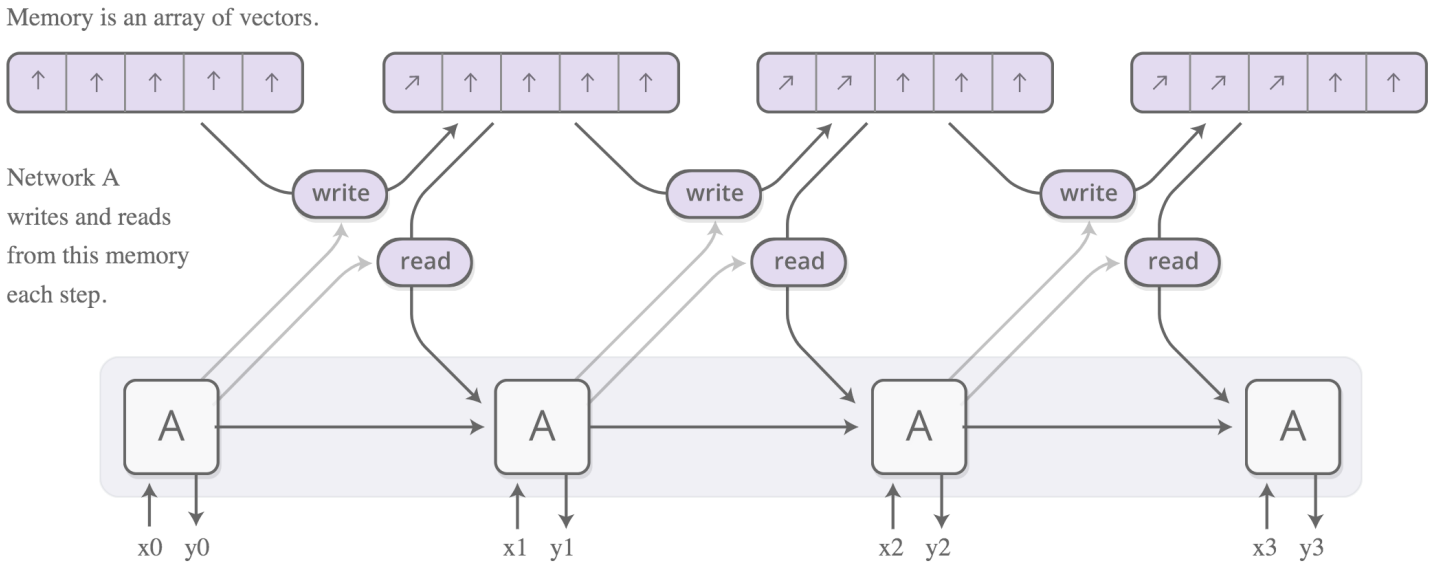
RNN Encoder/Decoder

How to make this work? (Sutskever, Vinyals & Le, 2014)

1. Deep LSTMs (multiple layers, e.g. 4)
2. Different RNNs for encoding and decoding
3. Teacher forcing (maximum likelihood) during training
4. Beam search for decoding at test time
5. Reverse order of source sequence
6. Ensemble-ing
7. $\Longrightarrow$ State-of-the art results on WMT benchmarks at the time

Today: use of attention-based models (see next section)

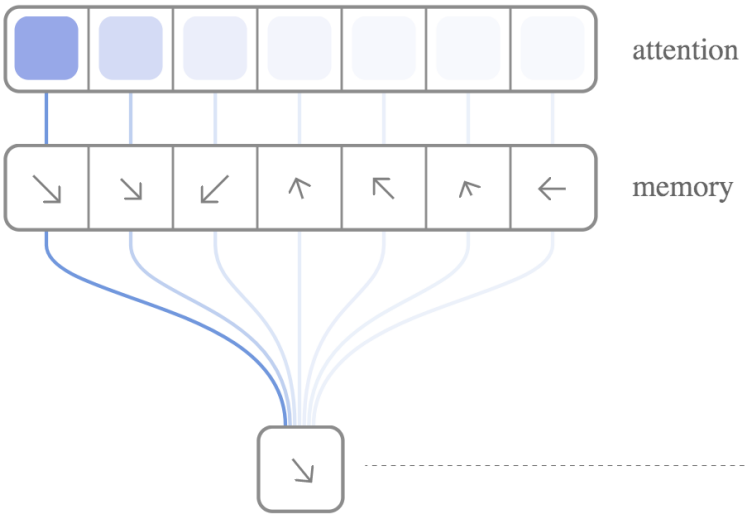
Neural Turing Machine: Architecture

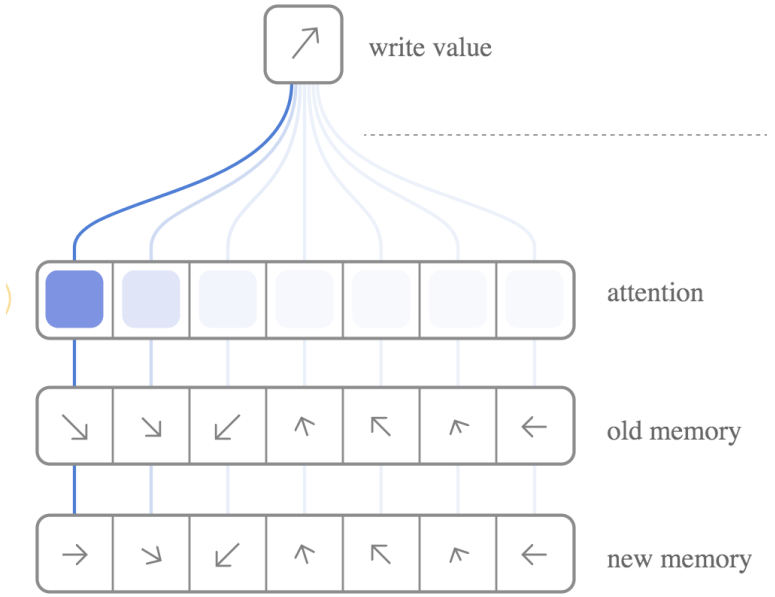


1. RNN controls an external memory bank
2. Reminiscent of Turing machine, but: each cell $M_i \in \mathbb{R}^d$

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Neural Turing Machine: Differentiable Memory

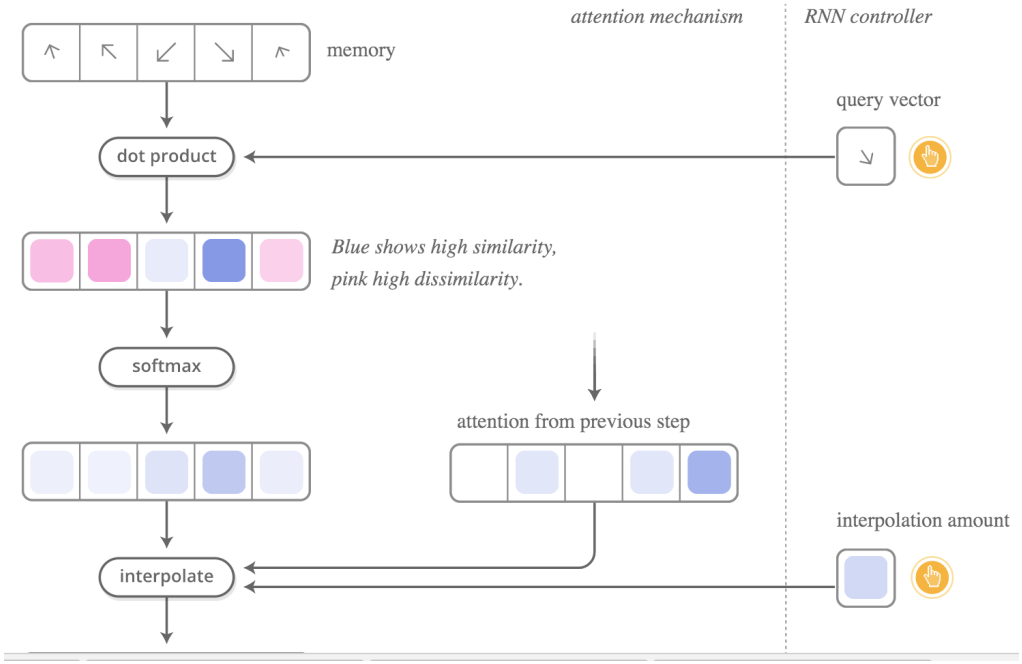




1. Compute attention distribution $(\alpha_i)_i$, $\alpha_i \geq 0$ s.t. $\sum_i \alpha_i = 1$.
2. Read out expected memory content: $r \leftarrow \sum_i \alpha_i M_i$.
3. Write uses weights $(\beta_i)_i$, $\beta_i \in [0;1]$, $M_i \leftarrow (1-\beta_i)M_i + \beta_i w$.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

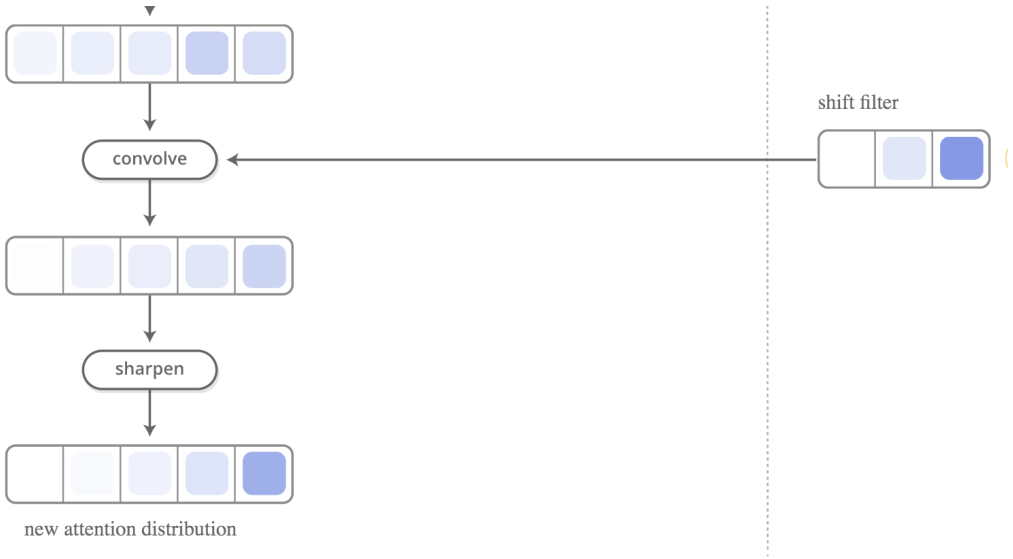
Neural Turing Machine: Memory Controller



1. Associative memory access with query (key) $q \in \mathbb{R}^d$.
2. Cells are scored via softmax.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Neural Turing Machine: Memory Controller



1. Ability to shift relative to content-selected locations (convolution).
2. Additional accentuation to sharpen attention distribution.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Differentiable Memory: Discussion

1. NTM architectures can learn loops and simple programs. However, no real-world applications.
2. Other architectures:
   - Neural random access machines (Kurach et al, 2015)
   - Differentiable data structures like stacks and queues (Grefenstette et al., 2015; Joulin & Mikolov, 2015)
3. Direction of future research

Section 12

Attention

Attention Mechanisms

1. Simple way to overcome some challenges of RNN-based memorization: **attention mechanism**

   Selectively attend to **inputs** or **feature representations** computed from inputs.

2. **RNNs**: learn to encode information relevant for the future.

   vs.

   Attention: select what is relevant from the past in hindsight!

   Both ideas can be combined

Gating Function

Definition (Softmax Gating Function)

A softmax gating function f_ϕ takes as input a query vector $\boldsymbol{\xi} \in \mathbb{R}^n$ as well as a set of values $\mathbf{x}^t \in \mathbb{R}^m$ $(t = 1, \ldots, s)$ and is defined as

$$f_\phi(\boldsymbol{\xi}, (\mathbf{x}^1, \ldots, \mathbf{x}^s)) = \frac{1}{\sum_j \exp\left[\phi(\boldsymbol{\xi}, \mathbf{x}^j)\right]} \begin{pmatrix} \exp\left[\phi(\boldsymbol{\xi}, \mathbf{x}^1)\right] \\ \vdots \\ \exp\left[\phi(\boldsymbol{\xi}, \mathbf{x}^s)\right] \end{pmatrix}$$

for some similarity or compatibility function $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$.

1. ϕ can often be learned in a black-box manner via a MLP
2. simplest choice for $n = m$: $\phi(\boldsymbol{\xi}, \mathbf{x}) = \boldsymbol{\xi}^\top \mathbf{x}$ (inner product)
3. every restriction $f_\phi(\boldsymbol{\xi}, \cdot)$ maps to the interior of a simplex

Self-Gated Attention

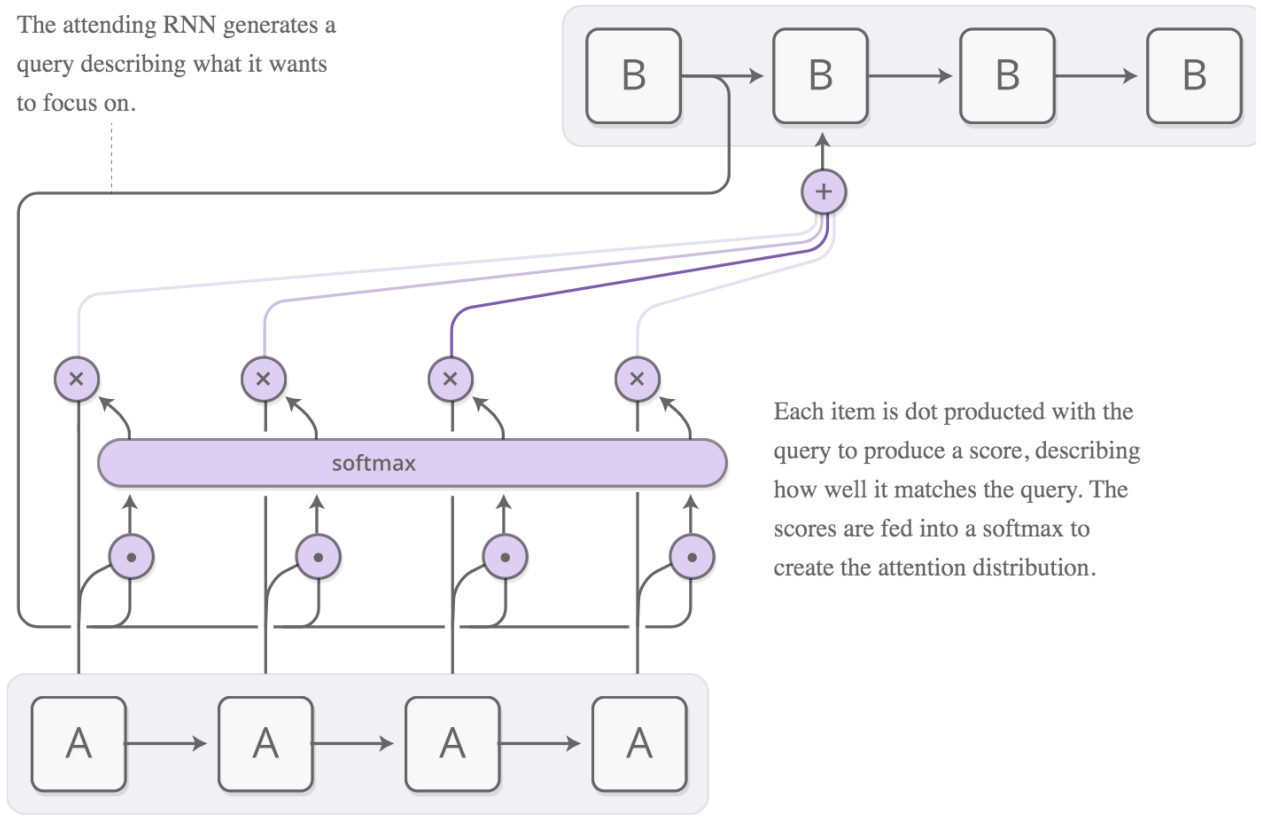
Definition (Self-Gated Attention)

Given a query $\boldsymbol{\xi} \in \mathbb{R}^m$ and a set of values $\mathbf{x}_i \in \mathbb{R}^n$ $(i = 1, \ldots, k)$. The self-gated attention is defined as

$$\underbrace{F(\boldsymbol{\xi}, (\mathbf{x}_1, \ldots, \mathbf{x}_k))}_{\in \mathbb{R}^k} = \underbrace{\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_k \end{bmatrix}}_{\in \mathbb{R}^{k \times n}} \cdot \underbrace{f_\phi(\boldsymbol{\xi}, (\mathbf{x}_1, \ldots, \mathbf{x}_k)))}_{\in \mathbb{R}^n}$$

where f_ϕ is a gating function.

Seq2seq with Attention: Schematic



from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Seq2seq with Attention

1. Attend to the hidden state of the encoding RNN, i.e. values $(\mathbf{h}_e^1, \ldots, \mathbf{h}_e^s)$.
2. Decoding RNN produces query at each time, i.e. $(\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^{s'})$.
3. Self-gated attention produces "read-out" $\mathbf{z}^t$ from encoder sequence
4. Used as input to the decoding RNN: $(\mathbf{h}_d^t, \mathbf{z}^t) \mapsto \mathbf{h}_d^{t+1}$

Seq2seq with Attention: MT Example

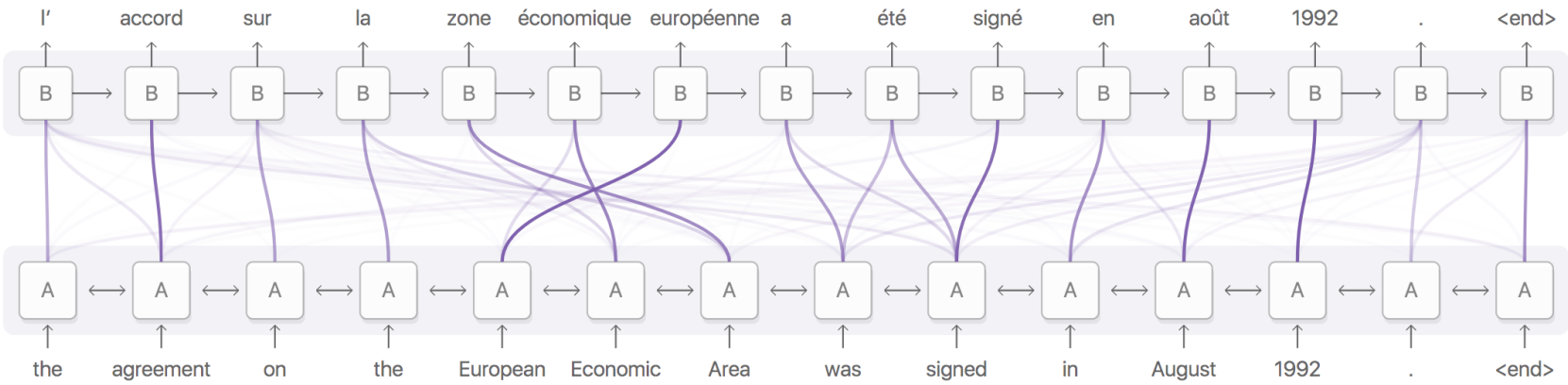
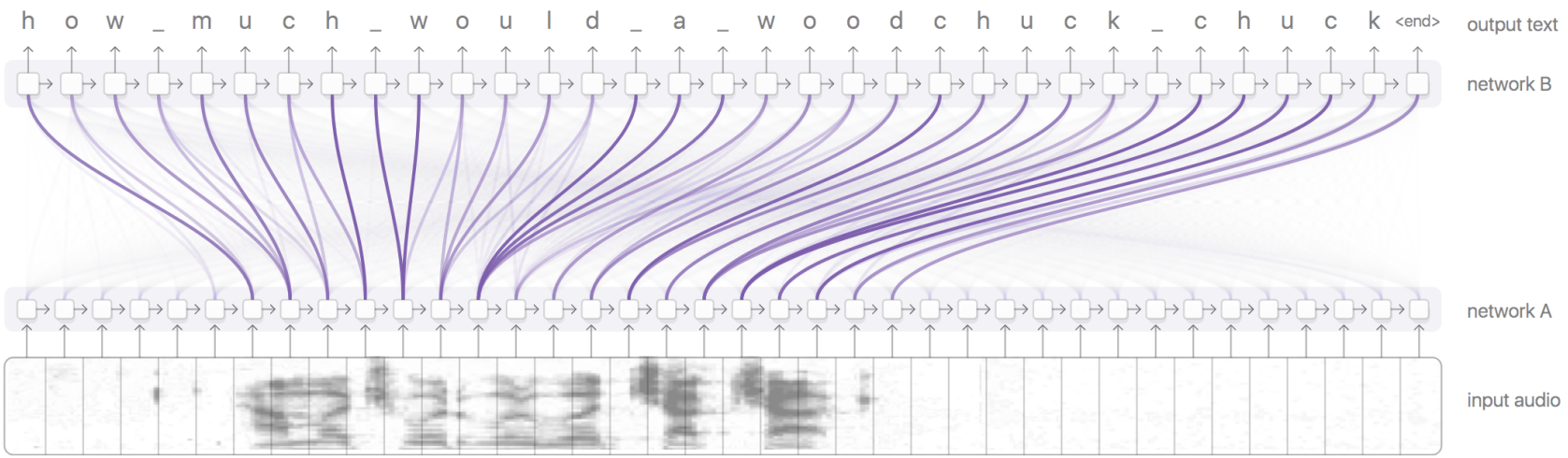


Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

1. Interpretable attention model (akin to alignments)
   Bahdanau, Cho & Bengio, 2014
2. Bi-directional GRU encoder, left-to-right GRU decoder

Seq2seq with Attention: Speech Recognition



from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

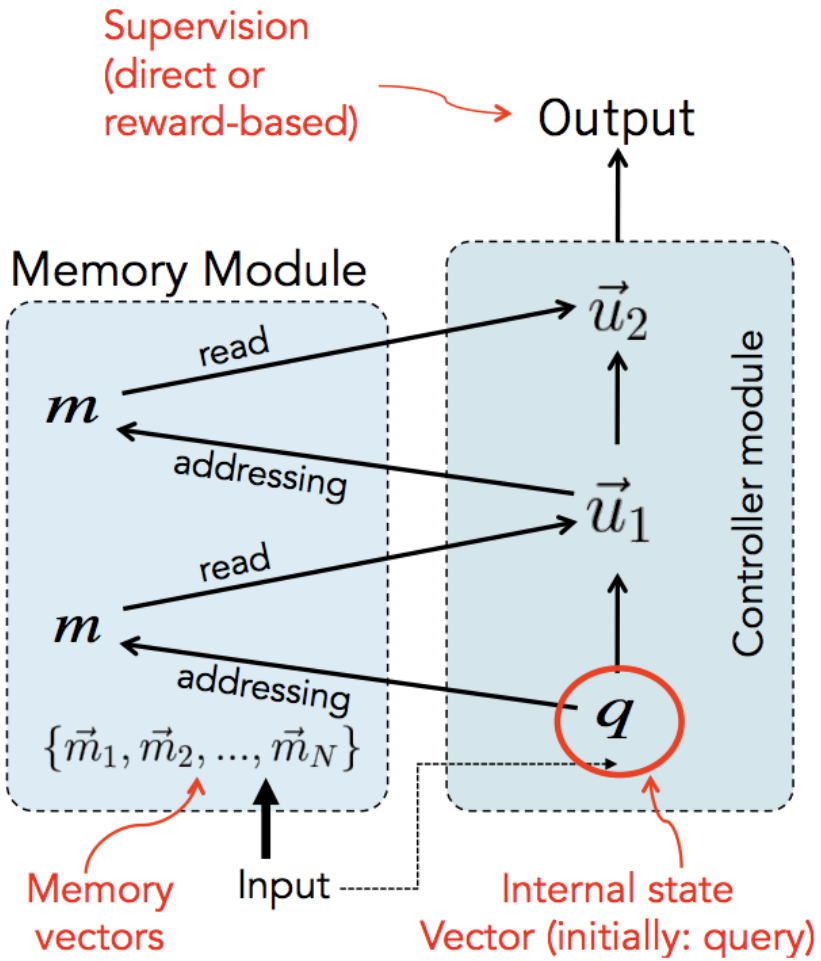
1. Listen, Attend and Spell Modell (Chan et al, 2015)
2. Bi-directional, pyramidal LSTM encoder

Memory Networks

Memory networks (Weston et al, 2014; Kumar et al, 2015)

1. operate over large data corpus (e.g. text documents)
2. given question, learn to infer answers (QA retrieval)
3. simplest form: memory is not altered
4. recursive associative recall: given query q, find best matching memory cell i; use M_i and $\mathbf{x}$ as new key; repeat

Memory Networks



from Weston, ICML Tutorial 2016: `http://www.thespermwhale.com/jaseweston/icml2016/`

Key-Value Attention

Definition (Key-Value Attention)

Given a query $\boldsymbol{\xi} \in \mathbb{R}^n$, key-value pairs $(\mathbf{x}^t, \mathbf{z}^t) \in \mathbb{R}^n \times \mathbb{R}^m$, $t = 1 \ldots, s$ and a gating function f. The (n, m)-dimensional key-value attention map is defined as

$$F(\boldsymbol{\xi}, ((\mathbf{x}^1, \mathbf{z}^1) \ldots, (\mathbf{x}^s, \mathbf{z}^s))) = \begin{bmatrix} \mathbf{z}^1 & \mathbf{z}^2 & \ldots & \mathbf{z}^s \end{bmatrix} \cdot f(\boldsymbol{\xi}, (\mathbf{x}^1, \ldots, \mathbf{x}^s))$$

1. attention weights are computed based on keys
2. produced value is linear (or convex) combination of values
3. keys determine where to look, values determine what features get extracted

Dot-Product Attention

Definition (Scaled Dot-Product Attention)

The attention map induced by

$$f(\boldsymbol{\xi}, \mathbf{x}) = \frac{\boldsymbol{\xi}^\top \mathbf{x}}{\sqrt{n}}$$

is called scaled dot-product attention.

1. simple dot-product similarity between query and key, not necessarily convex (soft-max)
2. motivation for normalization: assume $\boldsymbol{\xi}, \mathbf{x}$ are random n-vectors with zero mean and unit variances, then

$$\mathbf{E}\left[\boldsymbol{\xi}^\top \mathbf{x}\right] = 0 \quad \text{and} \quad \mathbf{E}\left[(\boldsymbol{\xi}^\top \mathbf{x})^2\right] = n$$

Multi-Headed Attention

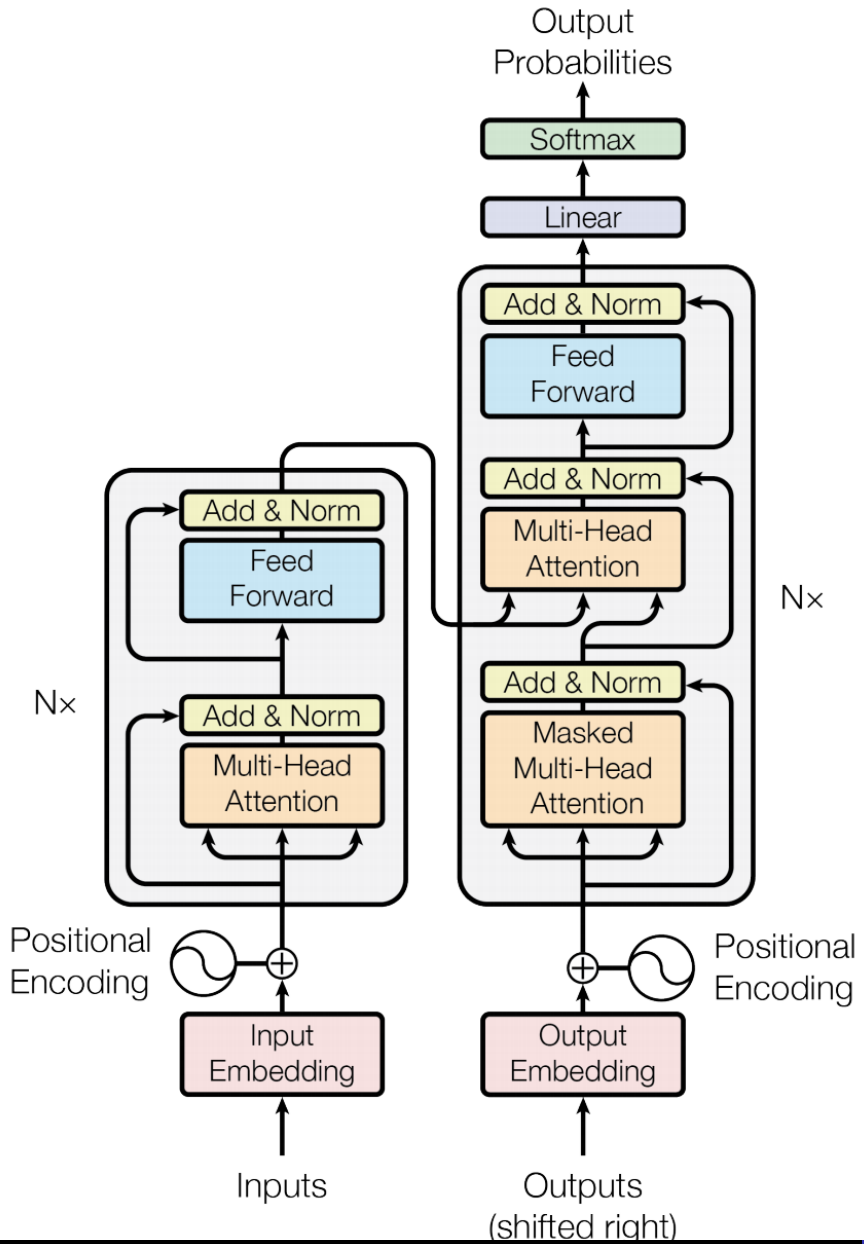
Definition (Multi-Headed Attention)

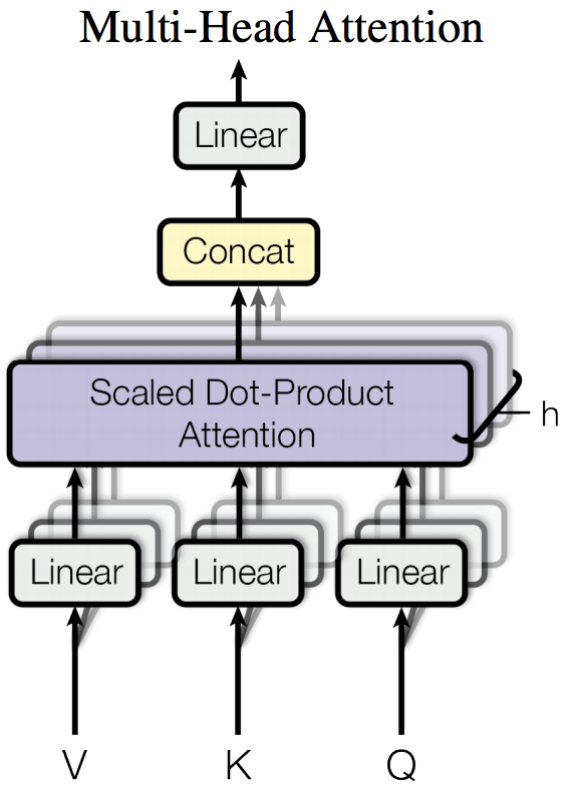
Let F_j, $1 \leq j \leq r$ be (n, m)-dimensional key-value attention map. An r multi-headed (N, M)-dimensional attention map G is defined as follows:

$$G(\boldsymbol{\xi}, (\mathbf{x}^t, \mathbf{z}^t)_{t=1}^s) = \mathbf{W} \begin{bmatrix} F_1(\mathbf{W}_1^q \boldsymbol{\xi}, (\mathbf{W}_1^x \mathbf{x}^t, \mathbf{W}_1^z \mathbf{z}^t)_{t=1}^s) \\ \vdots \\ F_1(\mathbf{W}_1^q \boldsymbol{\xi}, (\mathbf{W}_r^x \mathbf{x}^t, \mathbf{W}_r^z \mathbf{z}^t)_{t=1}^s) \end{bmatrix}$$

1. matrices $\mathbf{W}_i^q, \mathbf{W}_i^x \in \mathbb{R}^{n \times N}$ and $\mathbf{W}_i^z \in \mathbb{R}^{m \times M}$ are linear dimension-reduction matrices (typically: $n < N$ and $m < M$).
2. $\mathbf{W} \in \mathbb{R}^{M \times r \cdot m}$ adjusts the dimension (typically: reduction)
3. example: design choice in (Vaswani et al, 2017): $r = 8$, $n = m = 64$, $N = M = 512$.

Transformer Architecture: Overview





(Vaswani et al, 2017)

Transformer Architecture: Other Design Choices

1. Fully-connected feedforward networks (specifically: ReLU with layer width $512 \mapsto 2048 \mapsto 512$, cf. 1×1 convolution)
2. Positional encoding: learned or fixed (sine-functions of different frequency)
3. Layer normalization (Ba, Krios, Hinton, 2017), cf. later section on activity re-normalization
4. Skip connections with adds (cf. residual layers)