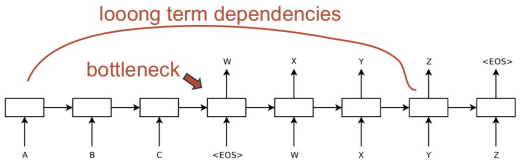
Q) Why Attention

Ans:- In Encoder-decoder the bottle neck context vec might not/cannot represent all the high-dimg inp seq. So the attention idea has came up.

## Problems with Vanilla seq2seq




- training the network to encode 50 words in a vector is hard => very big models are needed
- gradients has to flow for 50 steps back without vanishing => training can be slow and require lots of data

# Soft (Additive) Attention: Math

But where do the weights come from? They are computed by another network!

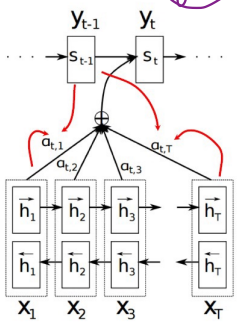
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

softmax

$$e_{ij} = a(s_{i-1}, h_j)$$

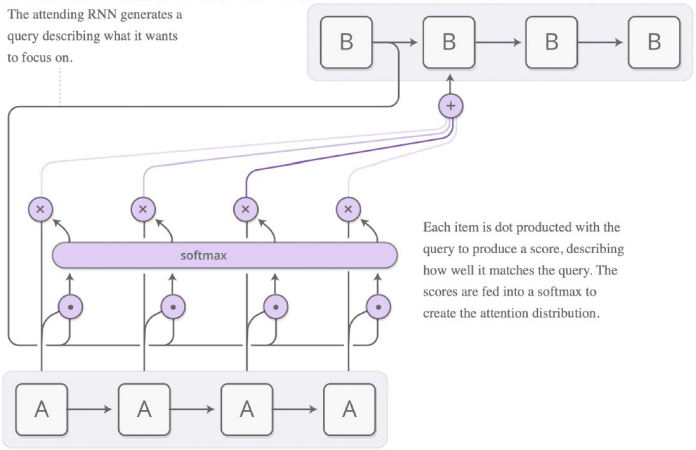
The choice from original paper is 1-layer MLP:

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




# Alternative: Dot-Product Attention

The attending RNN generates a query describing what it wants to focus on.




Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

## Summary

| Name | Alignment score function | Citation |
|---|---|---|
| Content-base attention | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$ | Graves2014 |
| 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 | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$<br>Note: This simplifies the softmax alignment to only depend on the target position. | Luong2015 |
| General | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$<br>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(^) | $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$<br>Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. | Vaswani2017 |

(*) 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.

# Soft (Additive) Attention: Computational Aspects

The computational complexity of using soft attention is quadratic. But it's not slow:

- for each pair of i and j
  - sum two vectors
  - apply tanh
  - compute dot product
- can be done in parallel for all j, i.e.
  - add a vector to a matrix
  - apply tanh
  - compute vector-matrix product
- softmax is cheap
- weighted combination is another vector-matrix product
- in summary: just vector-matrix products = fast!

$$e_{ij} = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j,$$

# Various Soft Attentions

- use dot-product or non-linearity of choice instead of tanh in content-based attention
- use unidirectional RNN instead of bidirectional
- explicitly remember past alignments with an RNN
- use a separate embedding for each of the positions of the input (heavily used in Memory Networks)
- mix content-based and location-based attentions

# Soft vs. Hard Attention

(D)RAM attention mechanism is *hard*
– it outputs a precise location where to look.

Content-based attention from neural MT is *soft*
– it assigns weights to all input locations.

CTC can be interpreted as a hard attention mechanism with tractable gradient.

Diff is in use only $s_{t-1}$, soft attent from decoder we in global we use from $s_{t-1}$..... So attention

### Global vs Local Attention

Luong, et al., 2015 proposed the "global" and "local" attention. The global attention is similar to the soft attention, while the local one is an interesting blend between hard and soft, an improvement over the hard attention to make it differentiable: the model first predicts a single aligned position for the current target word and a window centered around the source position is then used to compute a context vector.

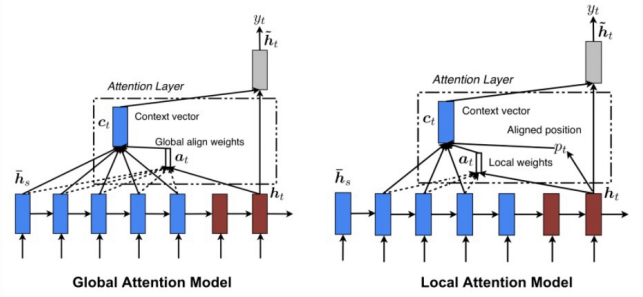



Fig. 8. Global vs local attention (Image source: Fig 2 & 3 in Luong, et al., 2015)

# Soft vs. Hard Attention

| Soft | Hard |
|---|---|
| • deterministic | • stochastic* |
| • exact gradient | • gradient approximation** |
| • O(input size) | • O(1) |
| • typically easy to train | • harder to train |

* deterministic hard attention would not have gradients
** exact gradient can be computed for models with tractable marginalization (e.g. CTC)

## What are the different types of attention?

***1. Self-Attention:***

Self-attention is a type of attention mechanism where the model makes prediction for one part of a data sample using other parts of the observation about the same sample.

It can be used to find the correlation between current words and other parts of the same sentence.

***2. Soft vs Hard attention:***

**Soft attention**: The attention weights are learnt and spread out across the entire sequence i.e., attends to the entire input.

**Adv**: The model is smooth and differential

**Disadv**: expensive with large source.

**Hard attention:** The attention weights are learnt for 1 element at a time i.e., attends to only a part of the input.

**Adv**: Less calculation at inference time

**Disadv**: the model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train.

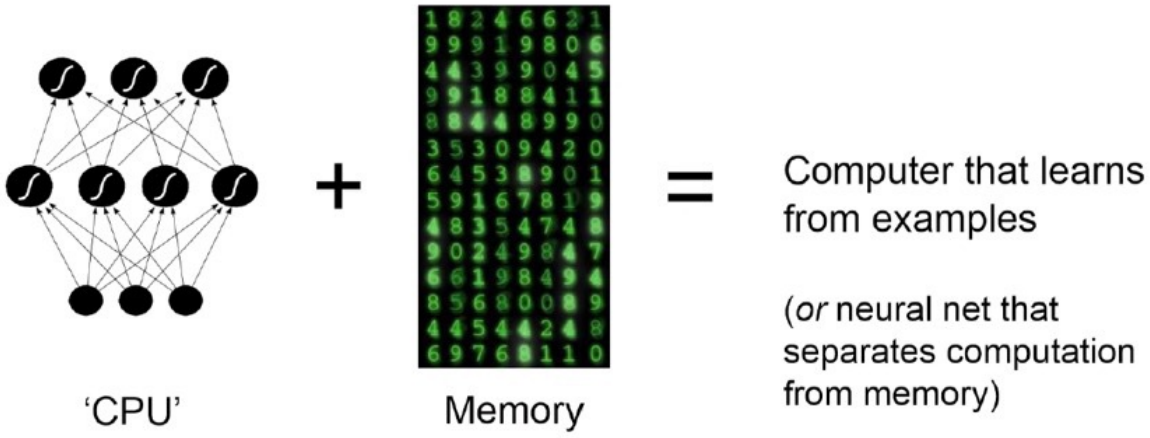
***3. Global vs Local:***

Global attention: Similar to the soft attention i.e., attends to the entire input.

Local attention: Combination of soft and hard attention i.e., attends over a window of fixed size.

**Memory:**

# Why explicit memory?




**List advantages of using explicit/external memory?**

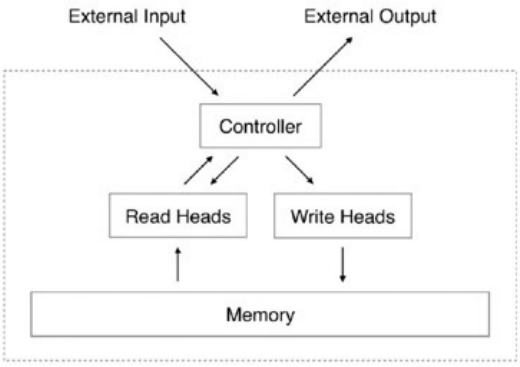
Information and gradients can be propagated for very long duration.

Explicit memory would allow network to not only store and retrieve specific facts but also to sequentially reason with them.

The **Controller** is a **neural network** (recurrent or feedforward)

The **Heads** **select** portions of the memory and **read** or **write** to them

The **Memory** is a real-valued **matrix**




*Everything is differentiable!*

**Explain NTMs?**

**Neural Turing Machine (NTM,** Graves, Wayne & Danihelka, 2014) is a model architecture for coupling a neural network with external memory storage. The memory mimics the Turing machine tape and the neural network controls the operation heads to read from or write to the tape. However, the memory in NTM is finite, and thus it probably looks more like a "Neural von Neumann Machine".

NTM contains two major components, a *controller* neural network and a *memory* bank. Controller: is in charge of executing operations on the memory. It can be any type of neural network, feed-forward or recurrent. Memory: stores processed information. It is a matrix of size $N \times M$, containing N vector rows and each has $M$ dimensions.

In one update iteration, the controller processes the input and interacts with the memory bank accordingly to generate output. The interaction is handled by a set of parallel *read* and *write* heads. Both read and write operations are "blurry" by softly attending to all the memory addresses.

**Content-based addressing**

The content-addressing creates attention vectors based on the similarity between the key vector $\mathbf{k}_t$ extracted by the controller from the input and memory rows. The content-based attention scores are computed as cosine similarity and then normalized by softmax. In addition, NTM adds a strength multiplier $\beta_t$ to amplify or attenuate the focus of the distribution.

$$w_t^c(i) = \text{softmax}(\beta_t \cdot \text{cosine}[\mathbf{k}_t, \mathbf{M}_t(i)]) = \frac{\exp(\beta_t \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\|\mathbf{k}_t\| \cdot \|\mathbf{M}_t(i)\|})}{\sum_{j=1}^{N} \exp(\beta_t \frac{\mathbf{k}_t \cdot \mathbf{M}_t(j)}{\|\mathbf{k}_t\| \cdot \|\mathbf{M}_t(j)\|})}$$

**Interpolation**

Then an interpolation gate scalar $g_t$ is used to blend the newly generated content-based attention vector with the attention weights in the last time step:

$$\mathbf{w}_t^g = g_t \mathbf{w}_t^c + (1 - g_t)\mathbf{w}_{t-1}$$

**Location-based addressing**

The location-based addressing sums up the values at different positions in the attention vector, weighted by a weighting distribution over allowable integer shifts. It is equivalent to a 1-d convolution with a kernel $\mathbf{s}_t(.)$, a function of the position offset. There are multiple ways to define this distribution. See Fig. 11. for inspiration.

# NTM Attention Mechanisms

- content-based addressing
  - similar to Hopfield nets
  - retrieve by query with (approximated) part
- interpolation
  - blend with attention weights from last step
- location-based addressing
  - needed for variables (varying content)
  - support address iteration (rotational weight shift) and random-access jumps

## NTM Read & Write

- soft read (content-location attention)

$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i)$$

- soft erase

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\,[\mathbf{1} - w_t(i)\mathbf{e}_t]$$

- soft write (add)

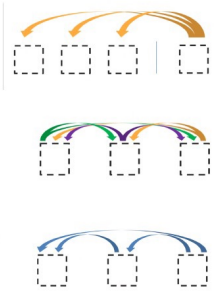
$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t$$

## NTM Attention Mechanisms

- content-based addressing
  - similar to Hopfield nets
  - retrieve by query with (approximated) part
- interpolation
  - blend with attention weights from last step
- location-based addressing
  - needed for variables (varying content)
  - support address iteration (rotational weight shift) and random-access jumps

## 3 Types of Attention in Transformer

- usual attention between encoder and decoder
  Q=[dec state] K=V=[enc states]
- self-attention in encoder
  Q=K=V=[encoder states]
  encoder attends to itself
- masked self-attention in decoder
  Q=K=V=[decoder states]
  decoder attends to itself,
  but a states can only attend previous states



## Other Tricks in Transformer

- allows different processing of information coming from different locations

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- positional embeddings are required to preserve the order information:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_\text{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_\text{model}})$$

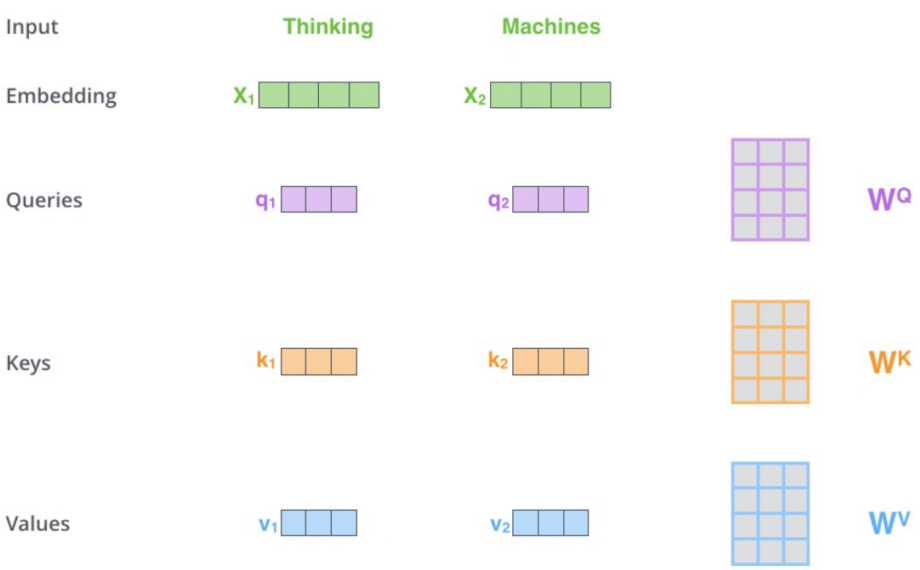
(trainable parameter embeddings also work)

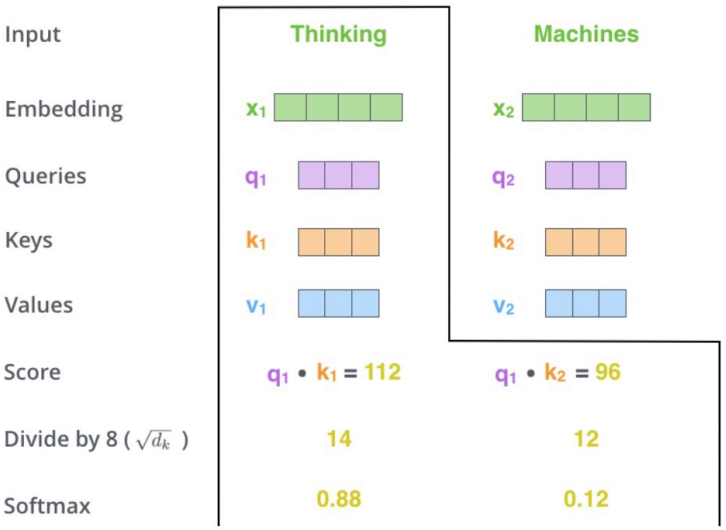
### Key, Value and Query

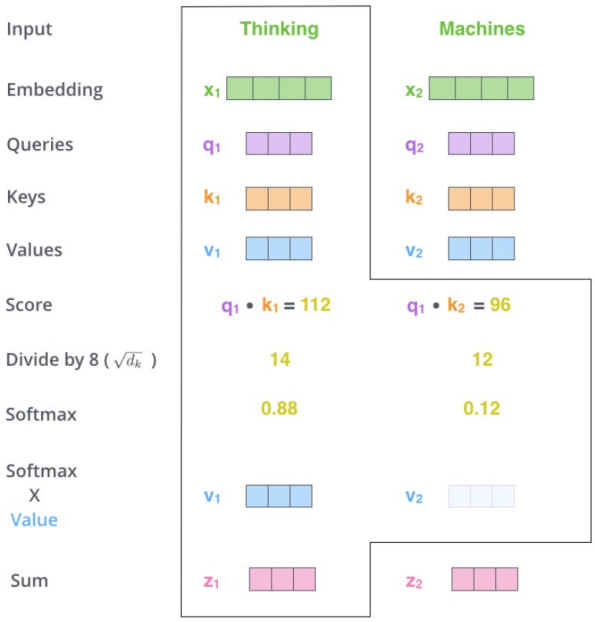
The major component in the transformer is the unit of *multi-head self-attention mechanism*. The transformer views the encoded representation of the input as a set of **key-value** pairs, $(\mathbf{K}, \mathbf{V})$, both of dimension $n$ (input sequence length); in the context of NMT, both the keys and values are the encoder hidden states. In the decoder, the previous output is compressed into a **query** ( $\mathbf{Q}$ of dimension $m$) and the next output is produced by mapping this query and the set of keys and values.

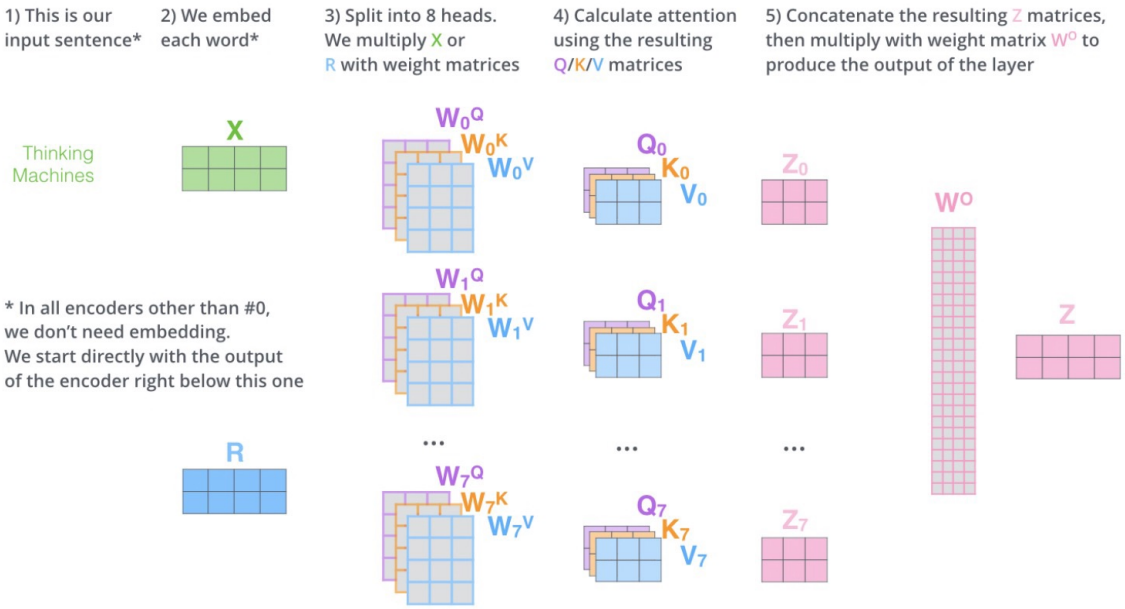
The transformer adopts the scaled dot-product attention: the output is a weighted sum of the values, where the weight assigned to each value is determined by the dot-product of the query with all the keys:

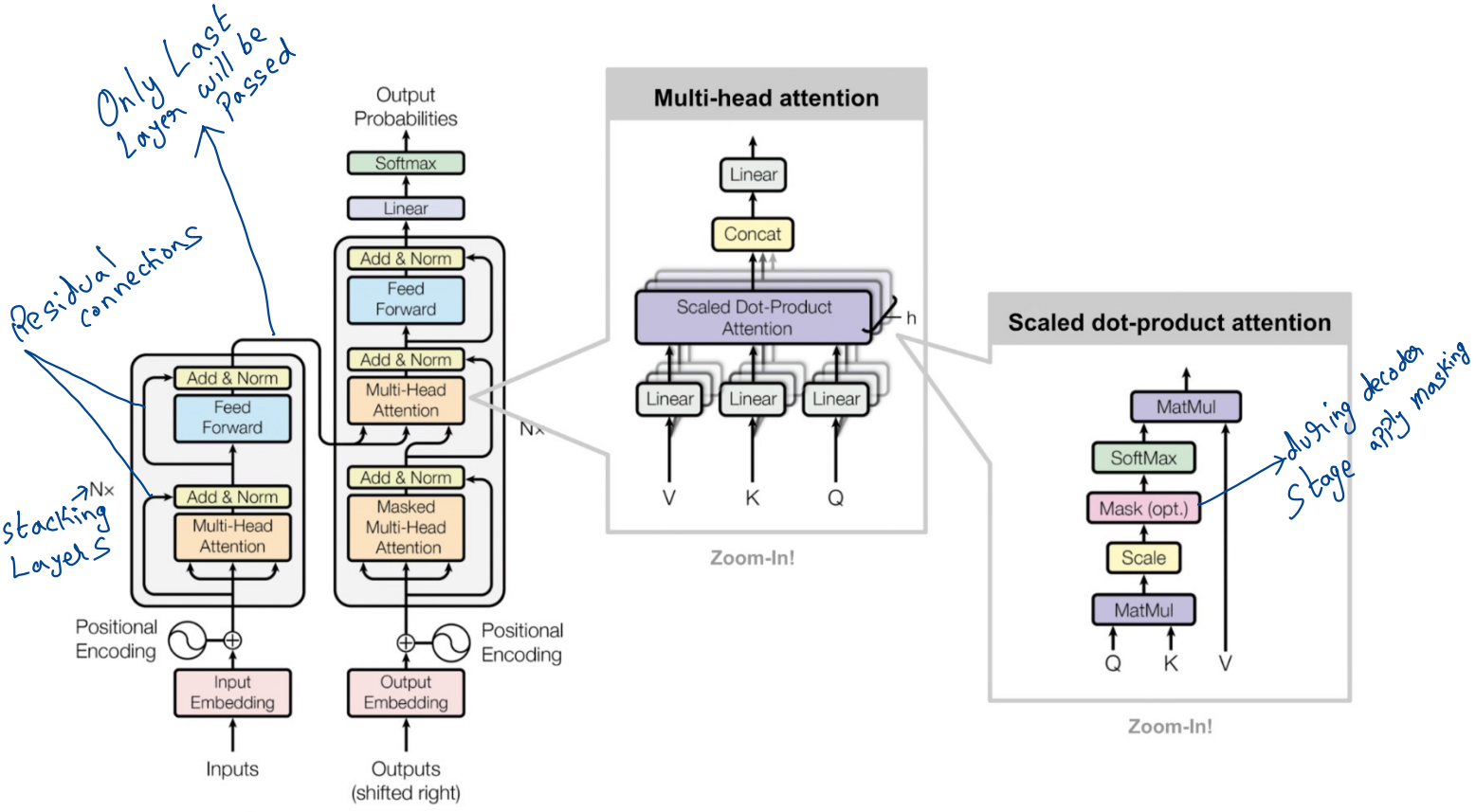
$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$




Input
Thinking
Machines
Embedding
Queries
Keys
Values
Score
Divide by 8 ( $\sqrt{d_k}$ )
Softmax
$x_1$
$x_2$
$q_1$
$q_2$
$k_1$
$k_2$
$v_1$
$v_2$
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
14
12
0.88
0.12

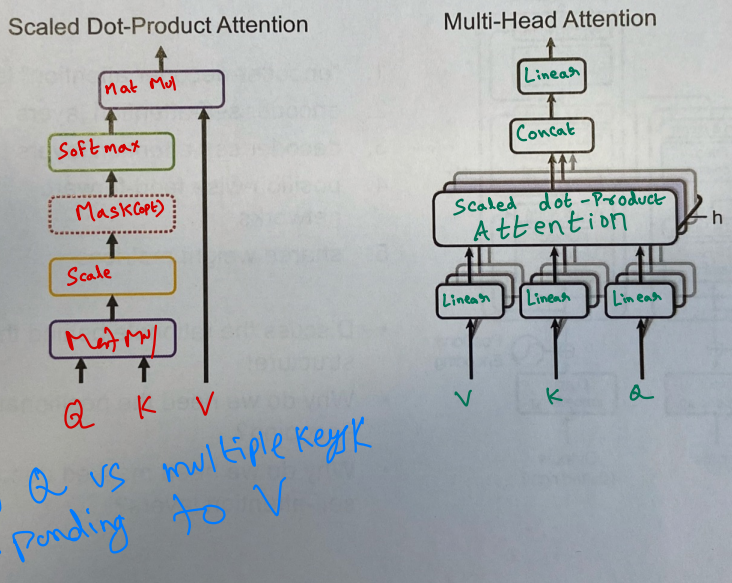

Input
Thinking
Machines
Embedding
Queries
Keys
Values
Score
Divide by 8 ( $\sqrt{d_k}$ )
Softmax
Softmax X Value
Sum
$x_1$
$x_2$
$q_1$
$q_2$
$k_1$
$k_2$
$v_1$
$v_2$
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
14
12
0.88
0.12
$z_1$
$z_2$


1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix $W^O$ to produce the output of the layer
Thinking
Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
$W_0^Q$ $W_0^K$ $W_0^V$
$W_1^Q$ $W_1^K$ $W_1^V$
$W_7^Q$ $W_7^K$ $W_7^V$
$Q_0$ $K_0$ $V_0$
$Q_1$ $K_1$ $V_1$
$Q_7$ $K_7$ $V_7$
$Z_0$
$Z_1$
$Z_7$
$W^O$
Z


Only Last Layer will be Passed
Residual connections
Nx
stacking Layers
Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Multi-head attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q
Zoom-In!
Scaled dot-product attention
MatMul
SoftMax
Mask (opt.)
Scale
MatMul
Q
K
V
during decoder Stage apply masking
Zoom-In!

## 2. Task: Complete the Sketch!

Highlight any trainable components!




## Attention Summary

- used to focus on parts of inputs / outputs
- content vs. location based (Dot Product) (Recurrent Attent) NTM
- hard vs. soft

- main distinct uses:
  - connect encoder and decoder in sequence-to-sequence task
  - achieve scale-invariance and focus in image processing
  - self-attention as basic building block for neural nets, often replacing RNNs and CNNs

**What are the different types of attention?**

*1. Self Attention:*

Self-attention is a type of attention mechanism where the model makes prediction for one part of a data sample using other parts of the observation about the same sample.

It can be used to find the correlation between current words and other parts of the same sentence.

*2. Soft vs Hard attention:*

Soft attention: The attention weights are learnt and spread out across the entire sequence i.e attends to the entire input.

Adv: The model is smooth and differential

Disadv: expensive with large source.

Hard attention: The attention weights are learnt for 1 element at a time i.e attends to only a part of the input.

Adv: Less calculation at inference time

Disadv: the model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train.

*3. Global vs Local:*

Global attention: Similar to the soft attention i.e attends to the entire input.

Local attention: Combination of soft and hard attention i.e attends over a window of fixed size.

# 1. Questions

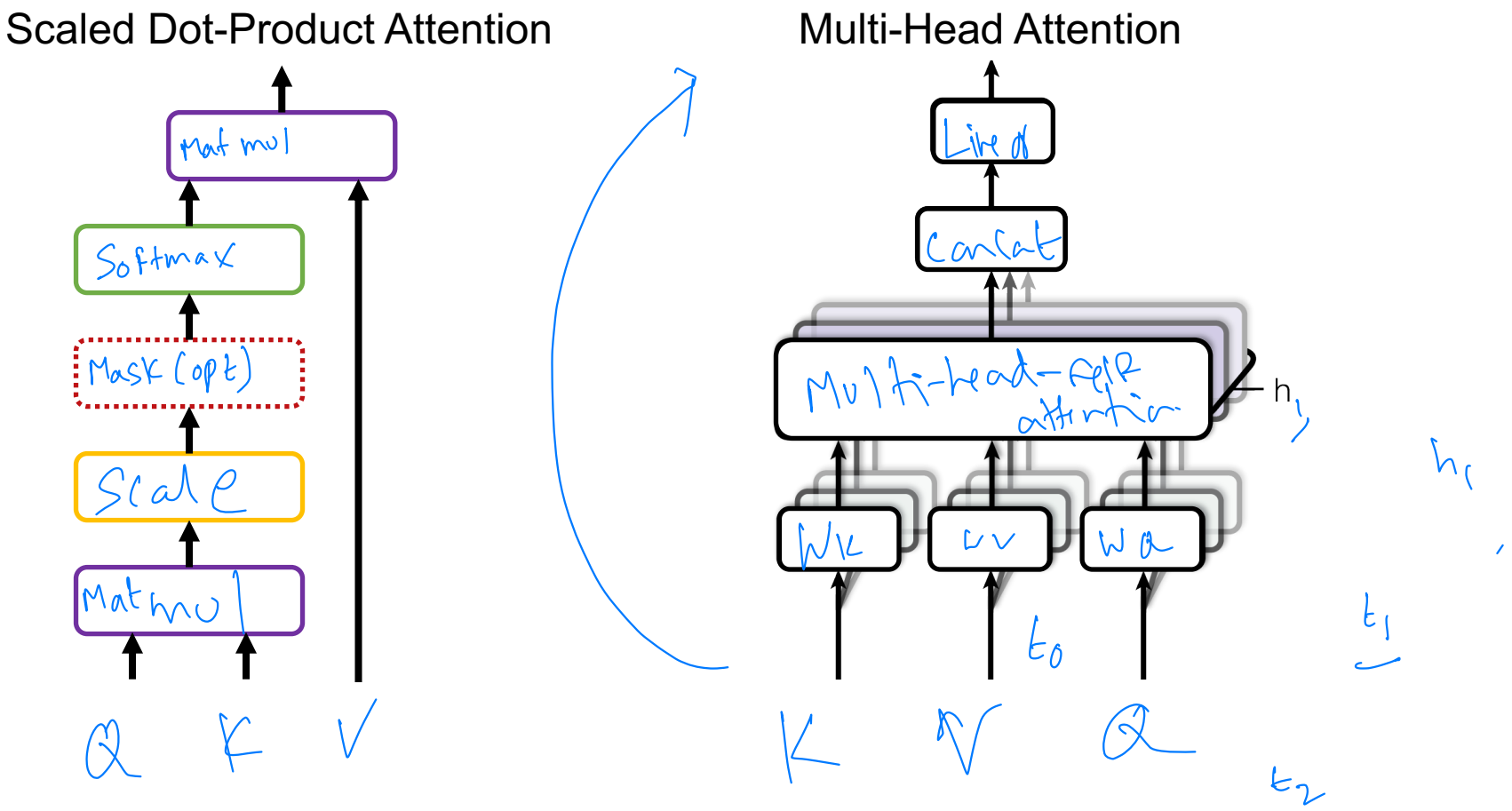
* Long-term dependencies,
* when there is no hidden-2-hidden connections, if neighbouring val is dependent on prev (a) rex

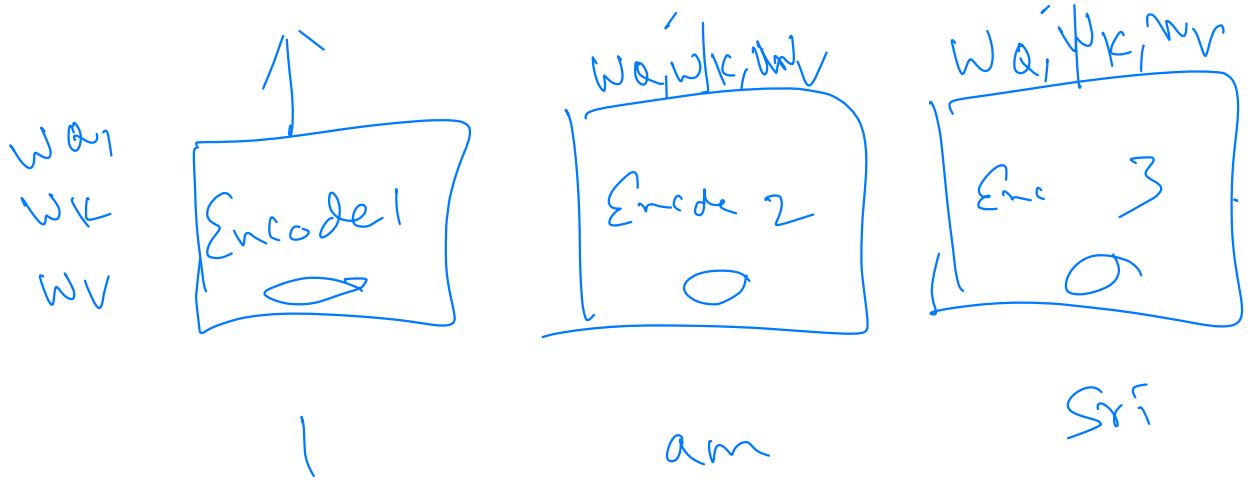
1. When can attentional interfaces be used?
2. Compare a sequence-to-sequence encoder-decoder RNN with and without attention! What is the advantage of adding the attention mechanism? Bottleneck
3. How could including an attention mechanism into the CharRNN improve the generated text? – bear, beat
4. What is self-attention?
5. List advantages of using explicit / external memory!
6. Which ways of addressing memory could be implemented?

→ Knowing the context of current word at any time step with all its neighbors.

# 2. Task: Complete the Sketch!

Highlight any trainable components!






$$\frac{\partial Loss}{\partial W_Q}$$

(13) $\frac{\partial Loss}{\partial W_Q}$ =

I. H $y$ Loss

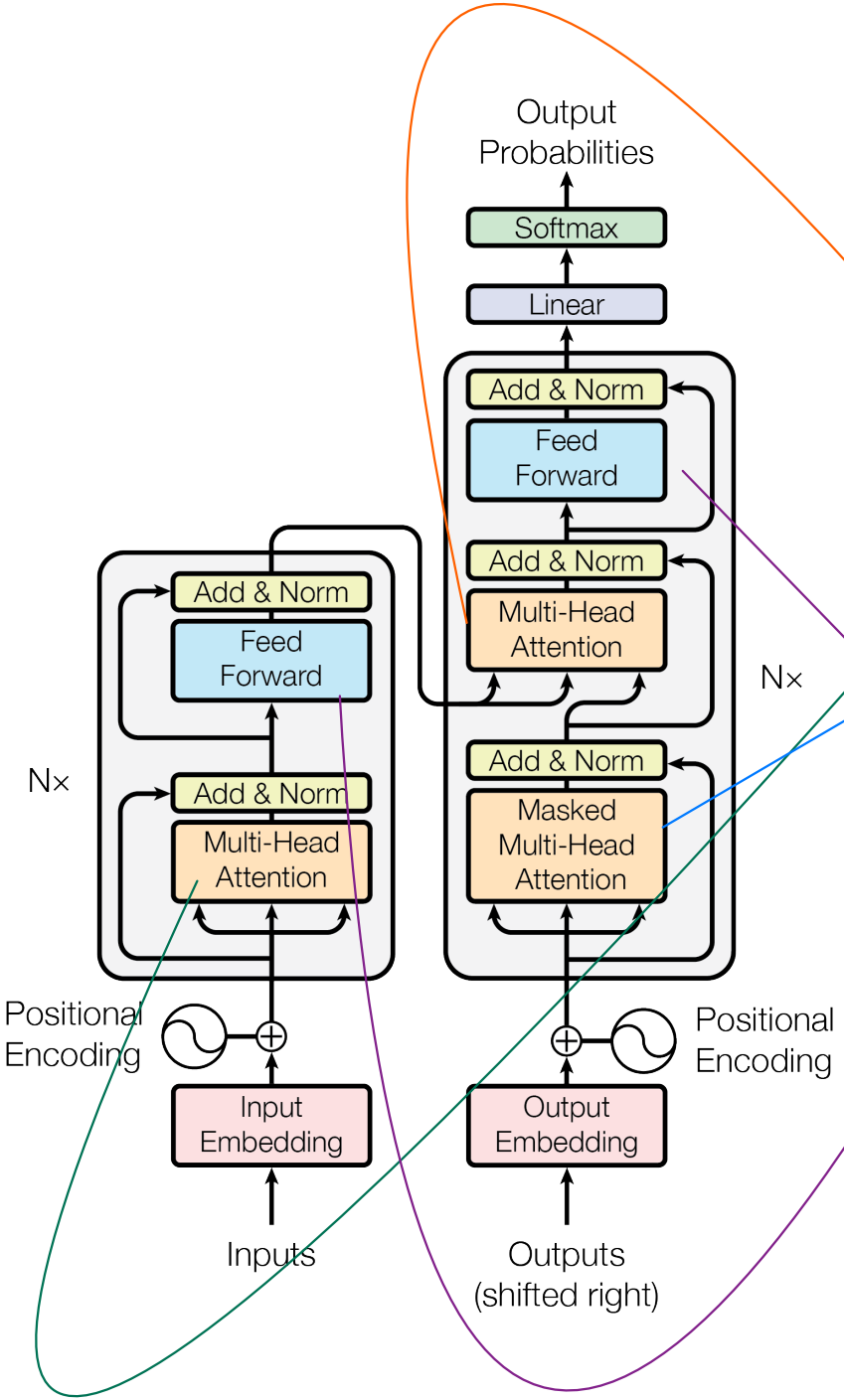
$W_1$ $W_2$ $W_Q$

$t=0$

$$\frac{\partial Loss}{\partial y_0} \times \frac{\partial y_0}{\partial W_Q}$$

$$\frac{\partial Loss}{\partial y_1} + \frac{\partial y_1}{\partial W_Q}$$

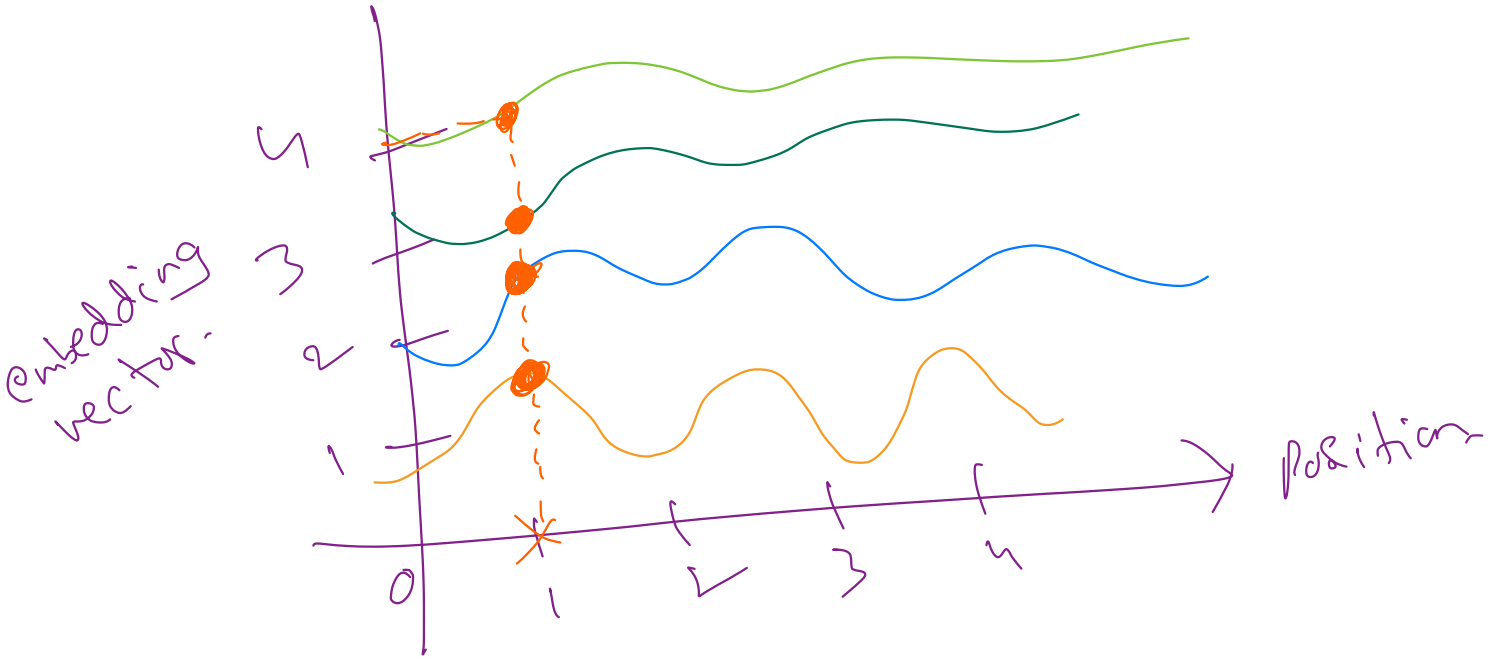
$W_{Q_1}$ =




# 3. Task: Annotate the Transformer!

1. "encoder-decoder attention" layers
2. encoder self-attention layers
3. decoder self-attention layers
4. position-wise feed-forward networks
5. shared weight matrices (across time $W_Q, W_K, W_V$)

- Discuss the rationale behind this structure!
- Why do we need the positional encoding? (for ordering)
- Why do we need masked decoder self-attention layers?


Embedding vector.
4
3
2
1
0
1
2
3
4
Position