TEK 5040/9040 Memory and Attention in RNNs

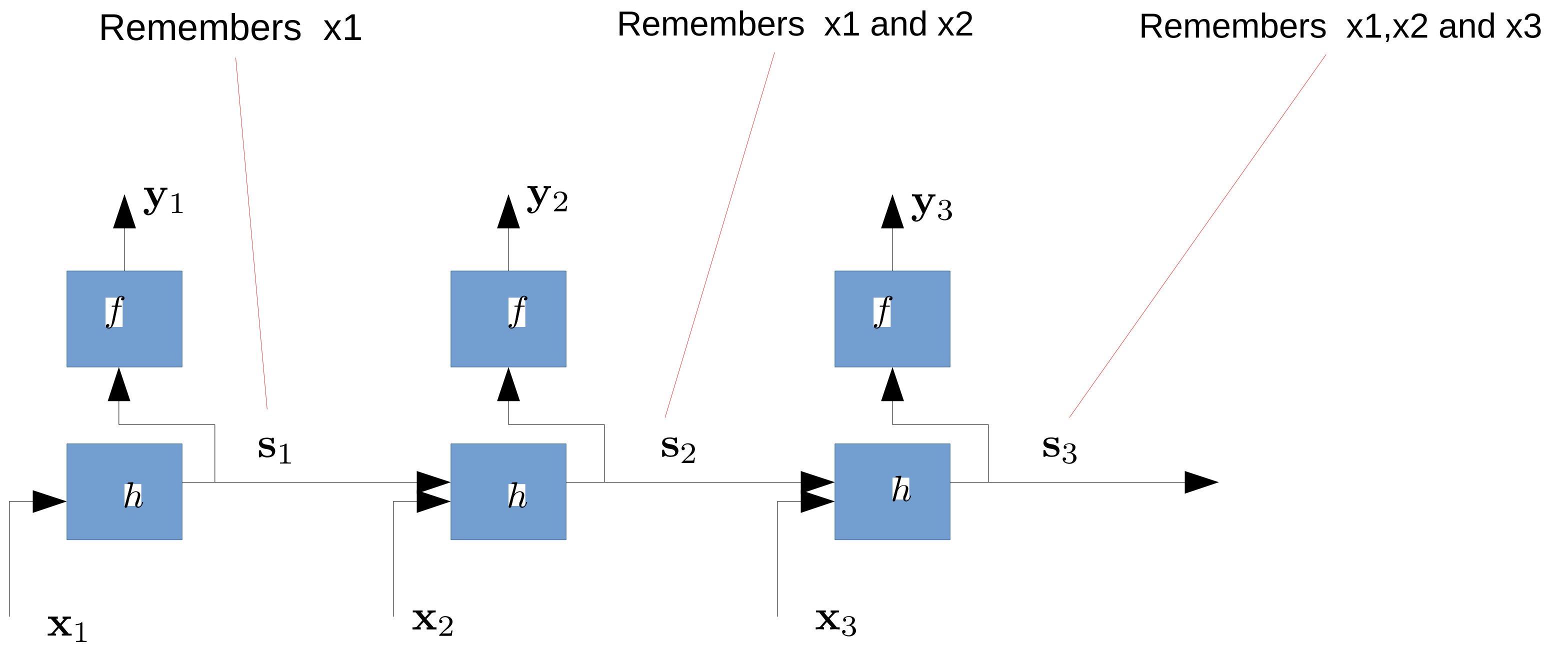
Narada Warakagoda

Sequence processing and "memory"

- Processing inputs such as images does not depend on memory
  - All inputs (pixels) are fed at the same time
- However, sequence processing inherently needs "memory"
  - Need to remember the previous inputs
- Plain RNN provides a simple memory solution!

The memory of plain RNN

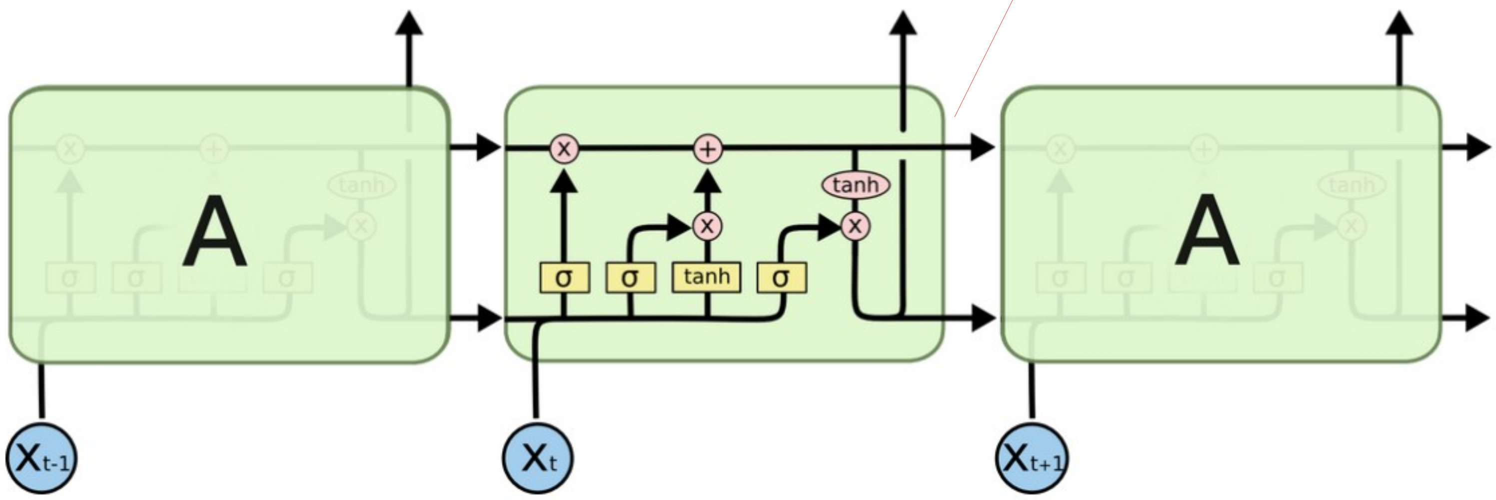
- Cell states are a kind of memory of the previous inputs
- But these memories are highly restricted!



Memory of the LSTM

- Control state has a better memory
  - Forgets irrelevant information
  - Remembers important information

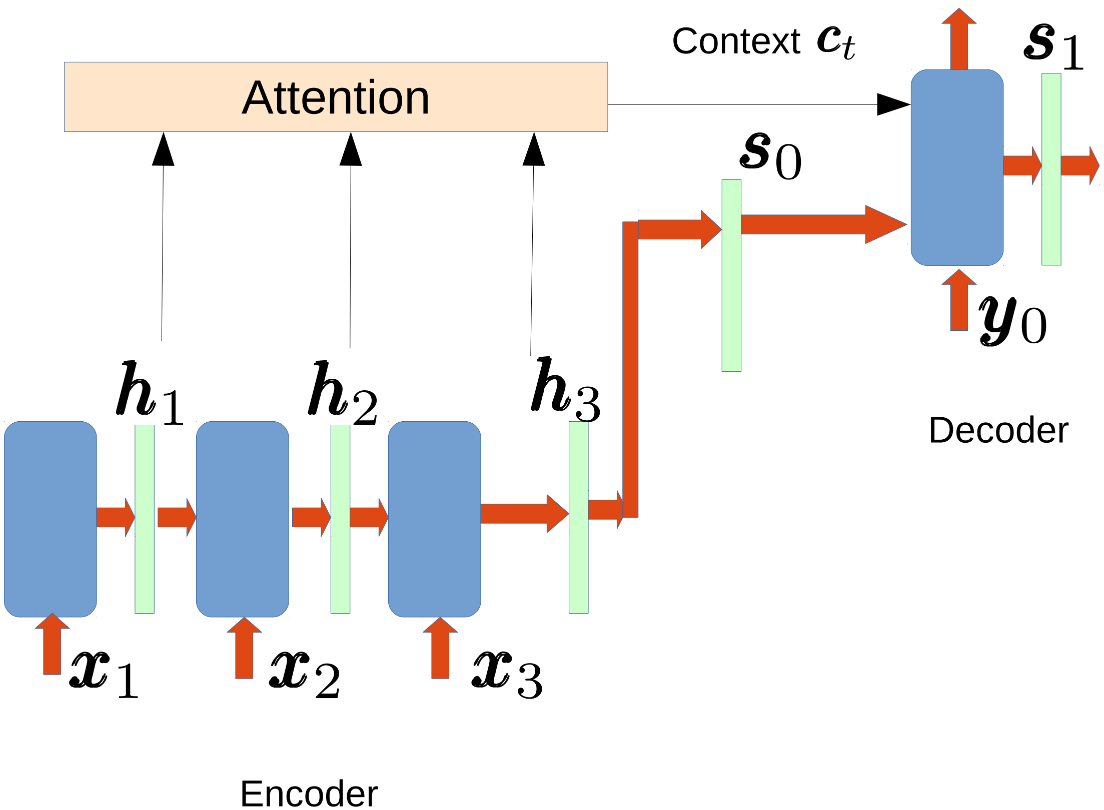
$\mathbf{c}_t$ remembers information from $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$



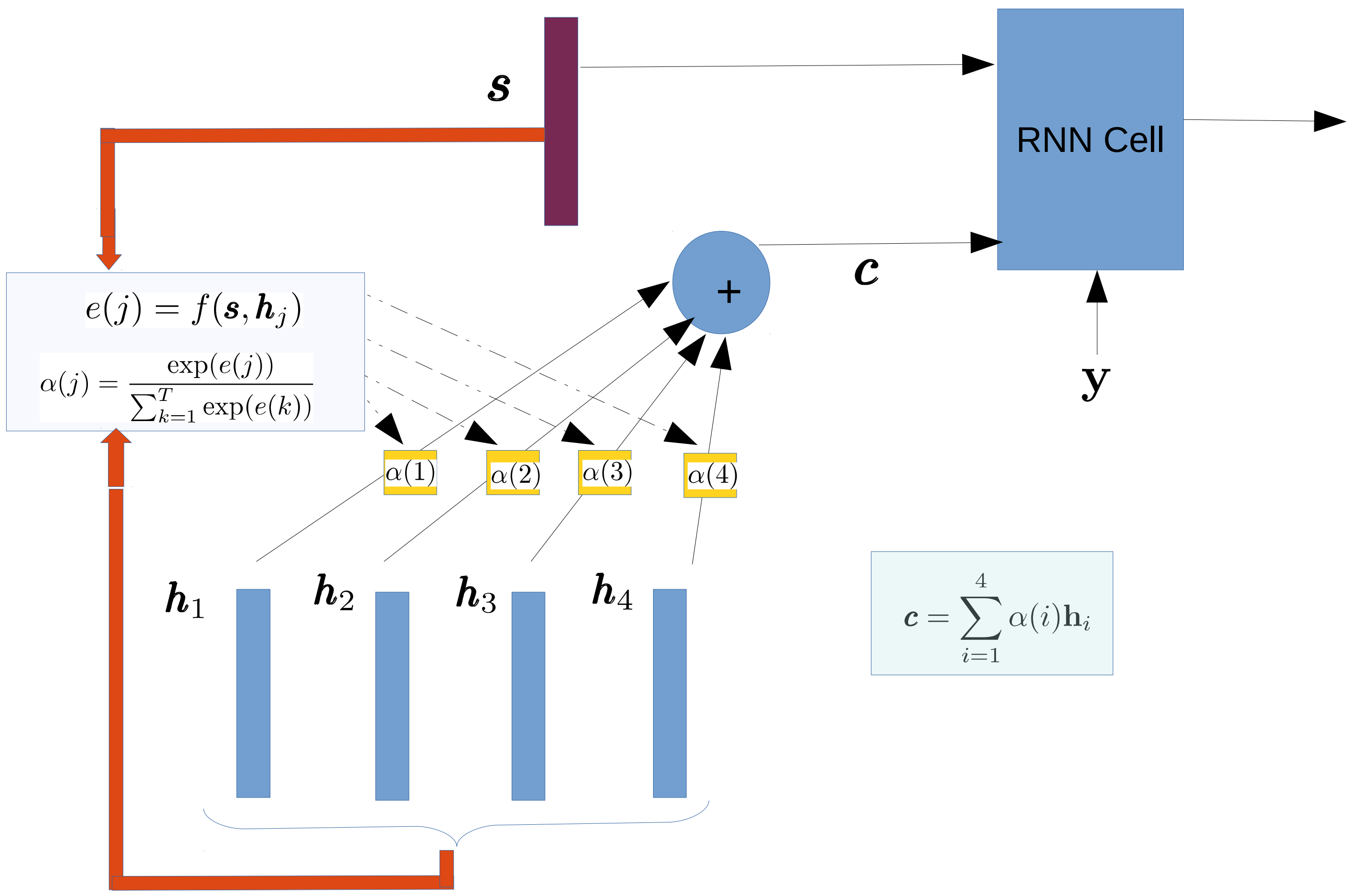
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Attention

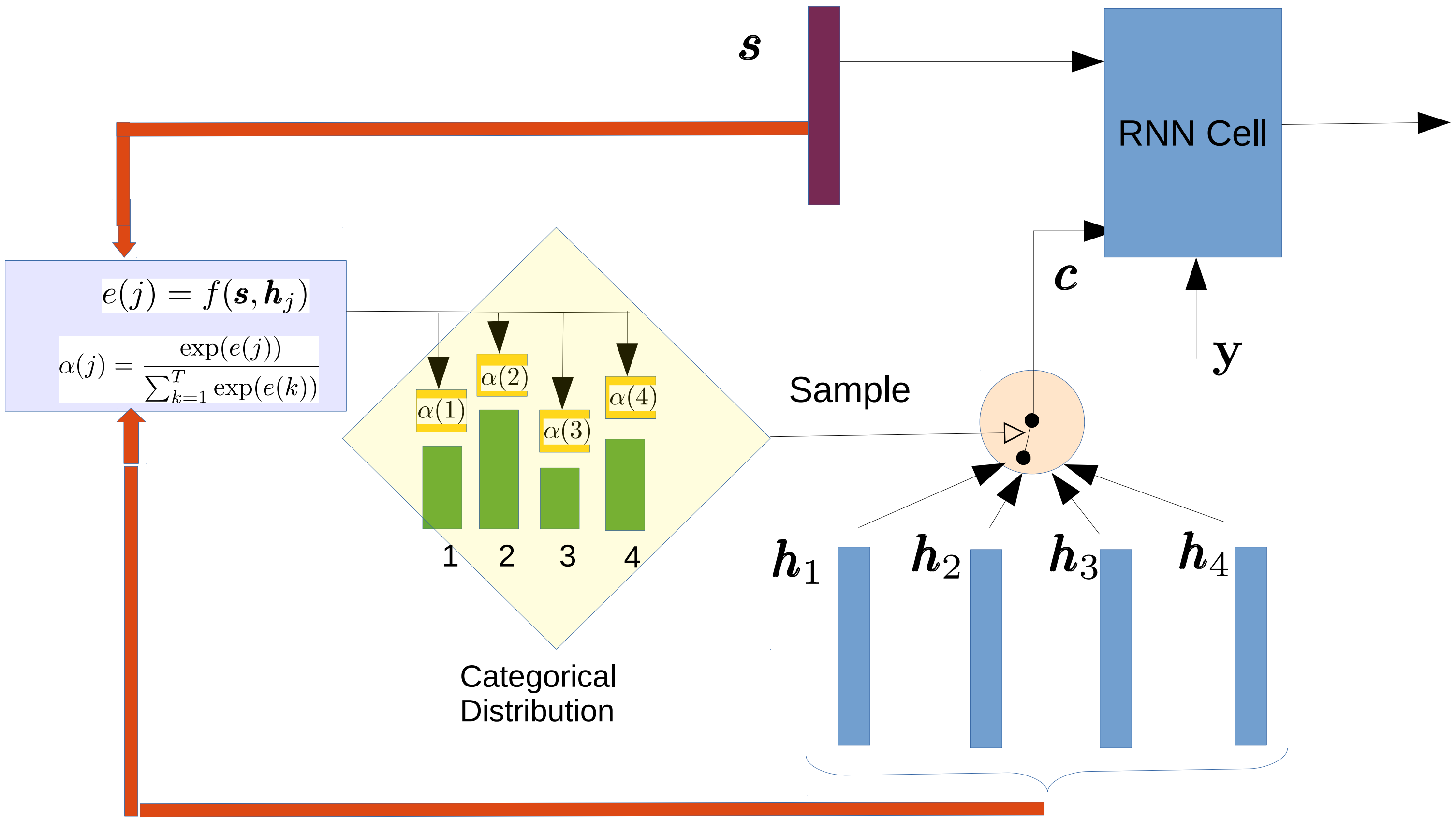
- Direct access to all the previous states
- But not all previous states are not important
  - Pick the states which are important



Soft attention



Hard Attention



Examples of attention

- Machine translation
  - Sequence to sequence
- Image captioning
  - Vector to sequence

Machine translation

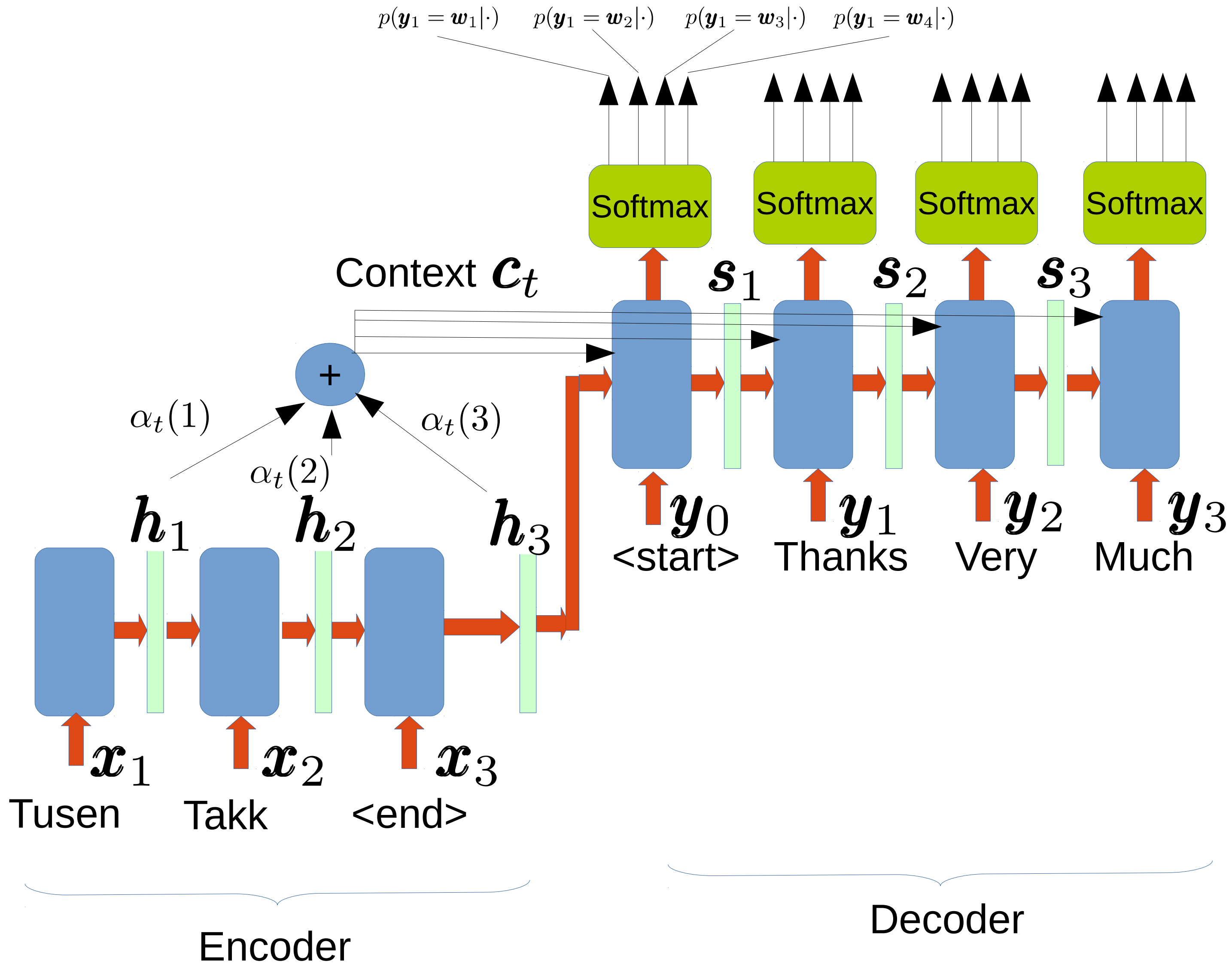
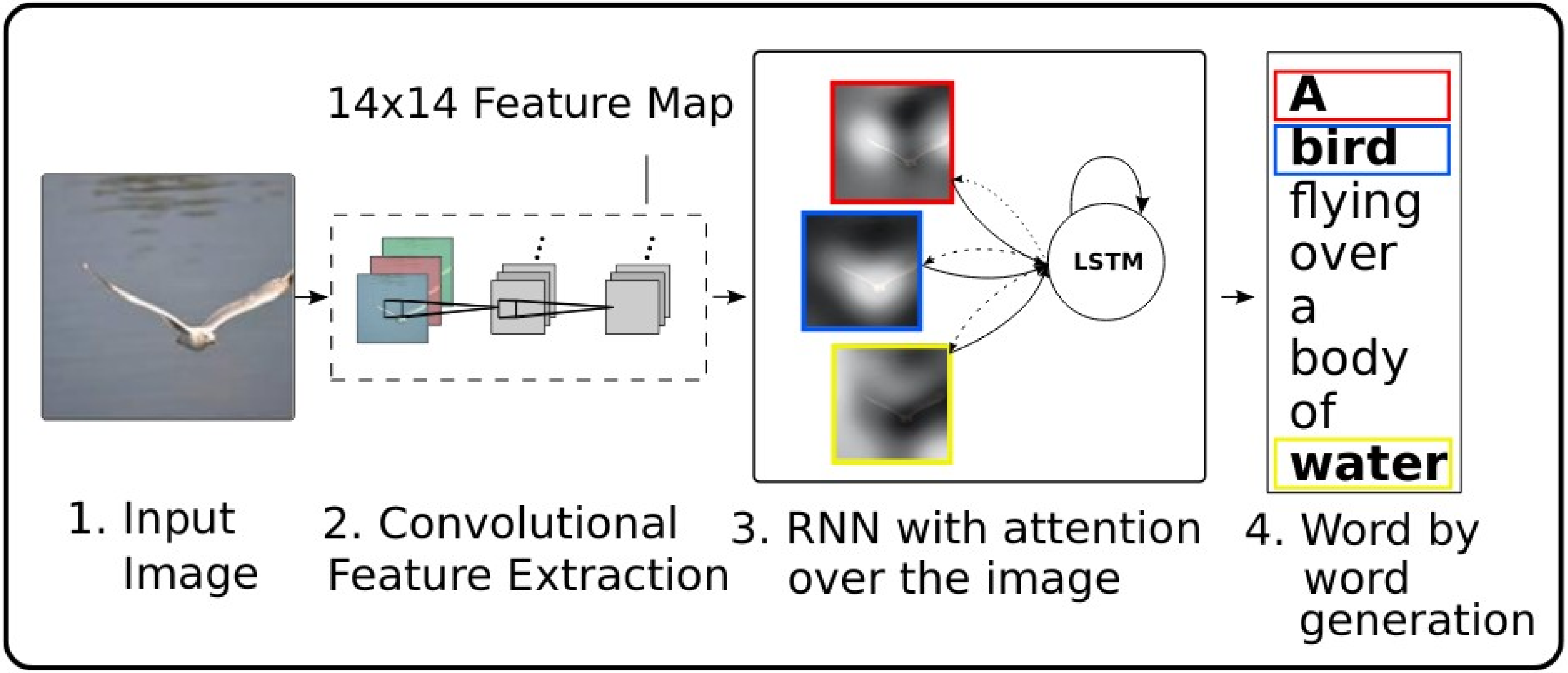
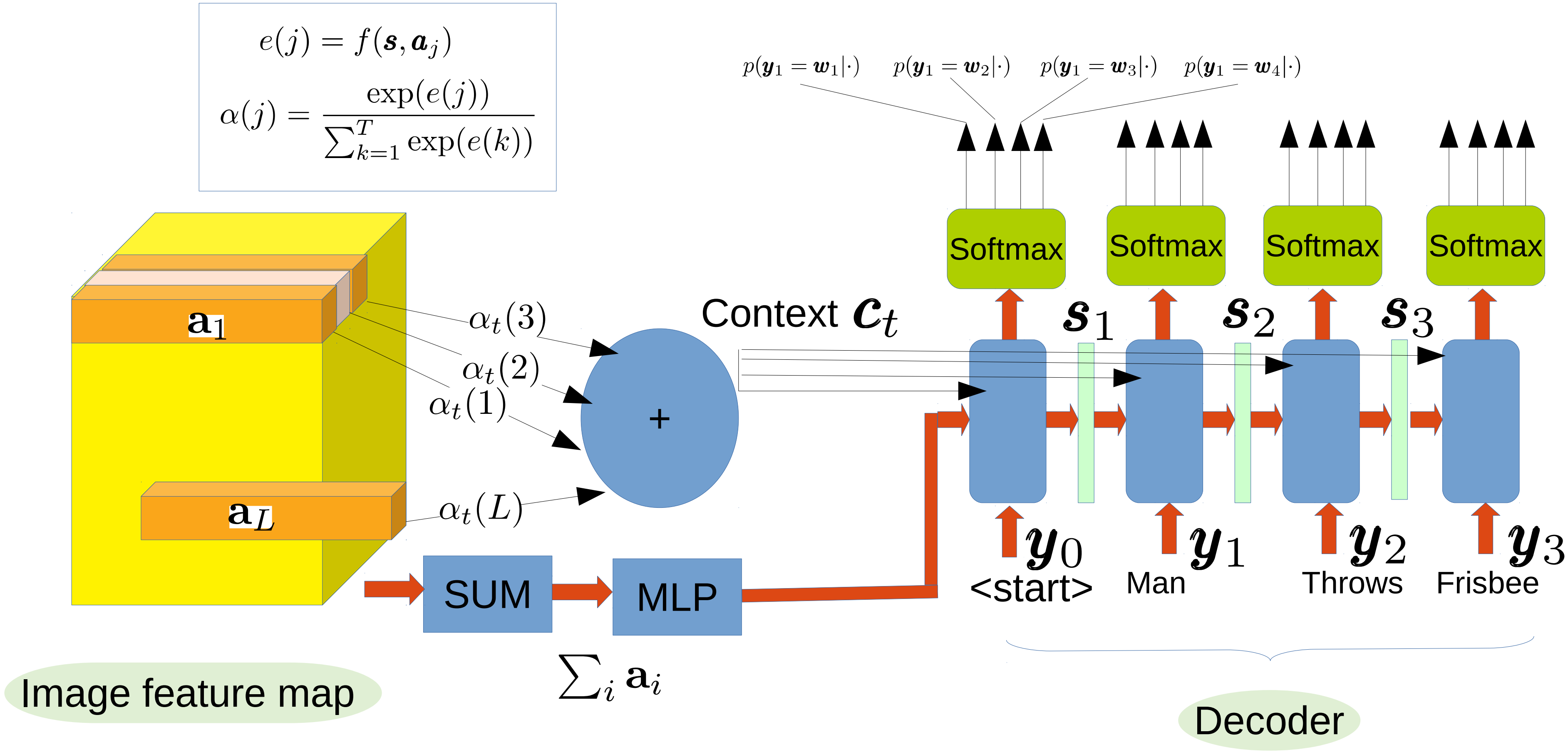


Image captioning



Attend to locations



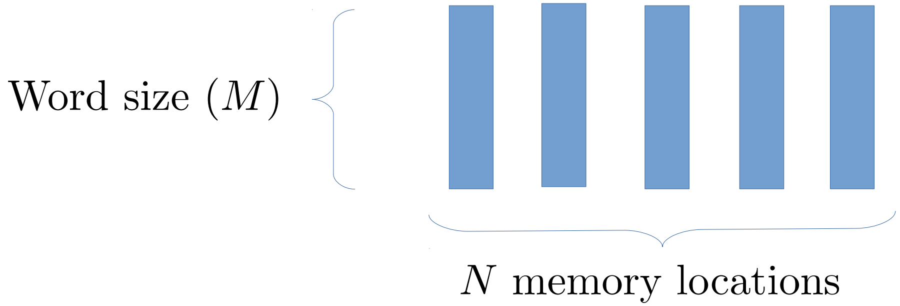
Attention visualization

Pros & cons of attention

- Pros:
  - Direct access to previous states (memory of previous inputs)
- Cons:
  - Memory = states (Not a general layout)
  - Writing to memory = generating states one by one

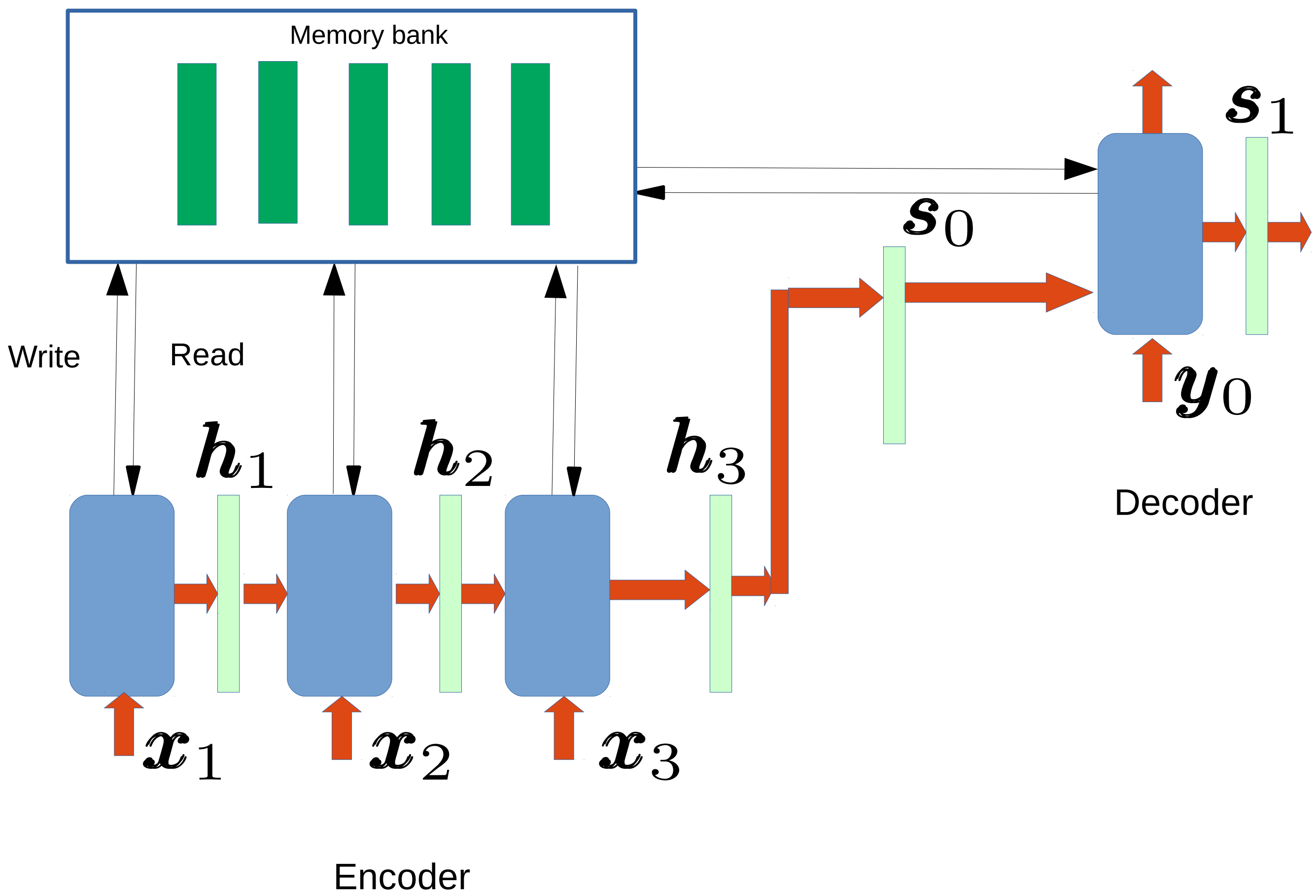
Memory Augmented Neural Network (MANN)

- Provides a general memory layout
  - (independent of the RNN architecture and time-steps)

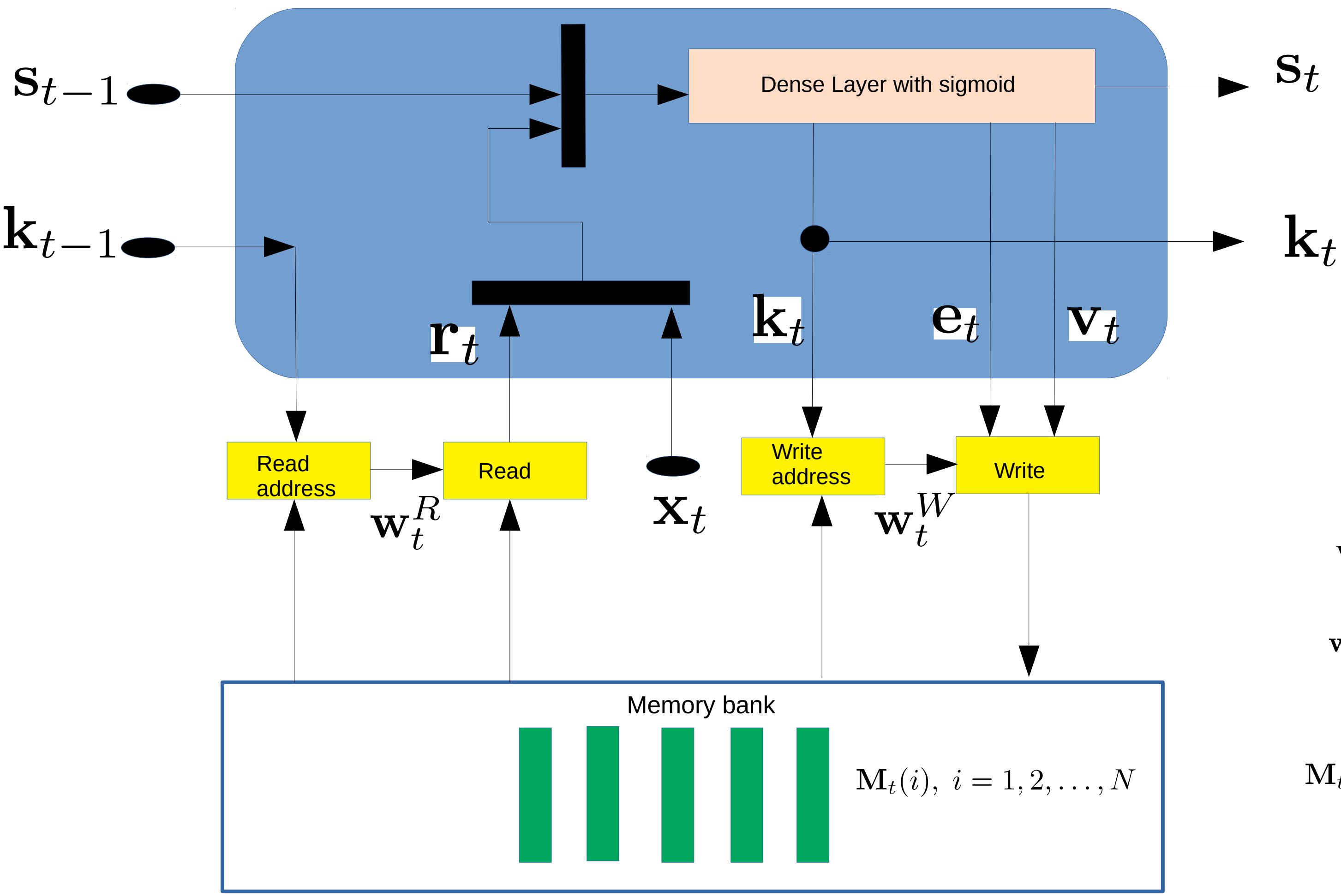


- Provides a more general reading/writing mechanism
- Choice in reading/writing memory
  - Content based addressing
  - Location based addressing

MANN architecture



Cell architecture



$$\begin{aligned}
\mathbf{x}_t &= \text{input} \\
\mathbf{s}_t &= \text{cell state} \\
\mathbf{k}_t &= \text{key} \\
\mathbf{w}_t^R &= \text{read weights} \\
\mathbf{r}_t &= \text{read vector} \\
\mathbf{w}_t^W &= \text{write weights} \\
\mathbf{e}_t &= \text{erase vector} \\
\mathbf{v}_t &= \text{write vector} \\
\mathbf{M}_t(i) &= \text{memory vector}
\end{aligned}$$

Reading Memory

- Generate read weights

$$\alpha_j = \mathbf{k} \cdot \mathbf{M}(j) \qquad \text{(Similarity measure)}$$

$$w^R(j) = \text{softmax}(\alpha_j) = \frac{\exp(\alpha_j)}{\sum_k \exp(\alpha_k)}$$

- Generate the read vector

$$\mathbf{r} = \sum_j w_j^R \mathbf{M}(j)$$

- "Soft" operation and hence differentiable
- Interpreted as "attention" to memory

Writing to memory

- Generate write weights

$$\alpha_j = \mathbf{k} \cdot \mathbf{M}^t(j) \qquad \text{(Similarity measure)}$$

$$w^W(j) = \text{softmax}(\alpha_j) = \frac{\exp(\alpha_j)}{\sum_k \exp(\alpha_k)}$$

- Perform the write operation (update memory)

$$\mathbf{M}^{t+1}(j) = \mathbf{M}^t(j) \circ \left(\mathbf{1} - w^W(j)\mathbf{e}\right) + w^W(j)\mathbf{v}$$

Elementwise multiplication

Erase vector

Write vector

Addressing

- Content based addressing (previous slides)
  - Find the similarity between memory contents and the key
  - Locations with high similarity
    - contributes more to the read vector (in reading)
    - are written with “more” information (in writing)
  - Locations with low similarity
    - Do the opposite

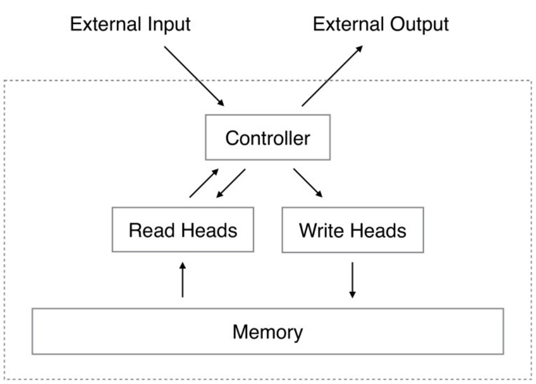
- Location based addressing
  - Read from (write to) a specified location
  - Common in regular computer systems
  - Differentiable location based addressing is useful in some cases.

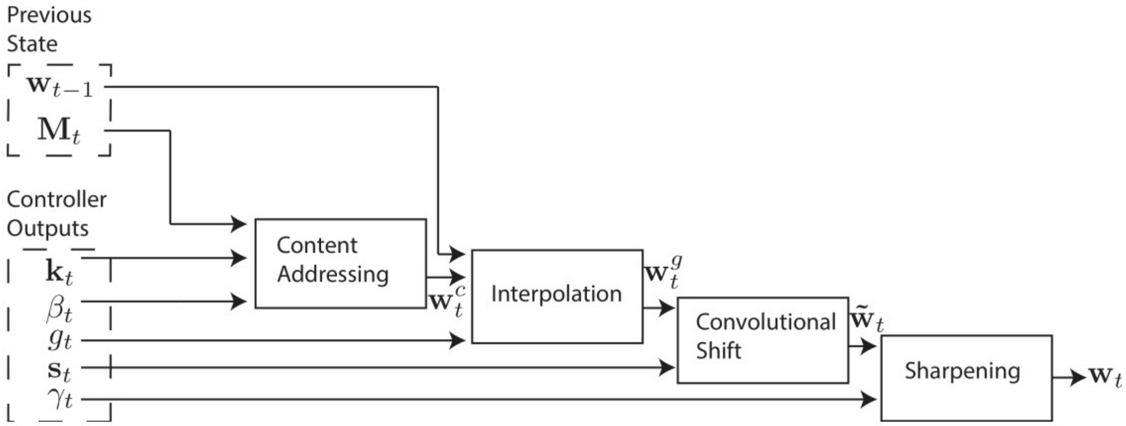
MANN examples

- Neural Turing Machine
- Differentiable Neural Computer

Neural Turing Machine

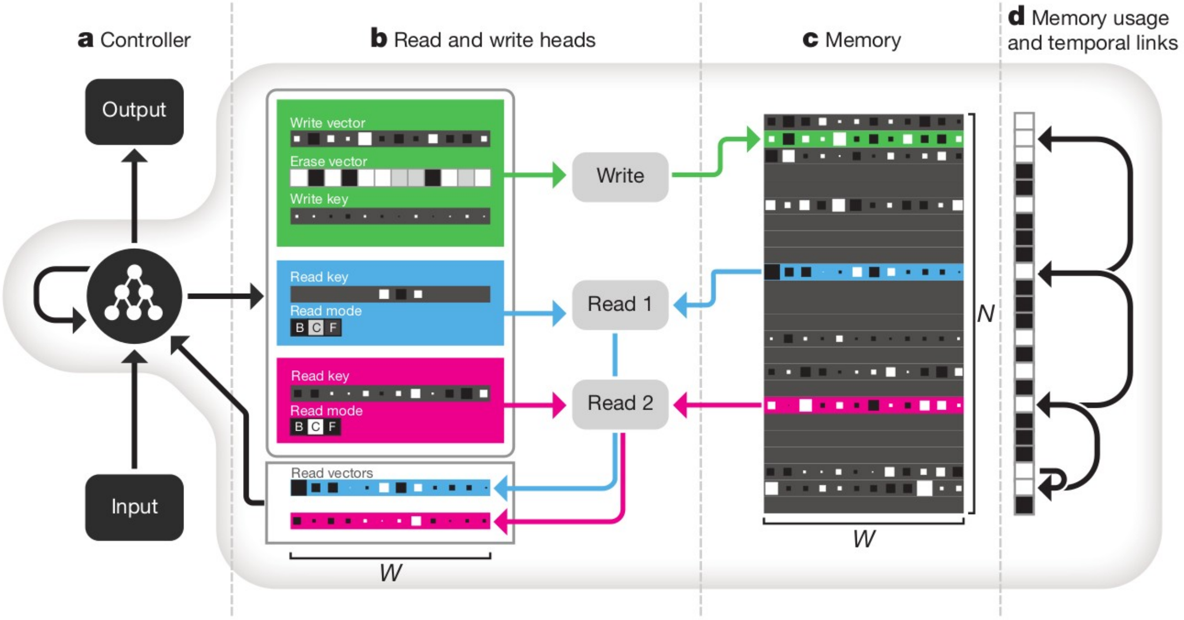
- Sophisticated weight generaion mechansim
- Supports
  - Content based addressing
  - Location based addressing (through address shifting distribution s_t)
  - Both addressing mechanisms are blended





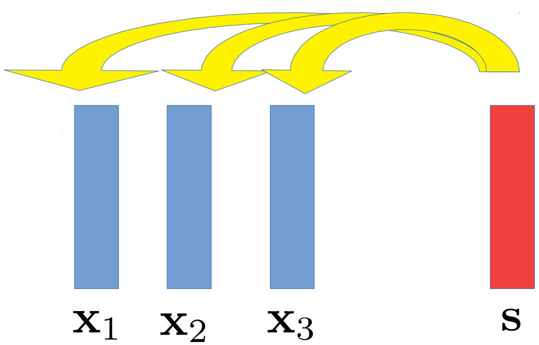
Differentiable Neural Computer (DNC)

- Newer variant of NTM
- Main difference: Memory usage and temporal links.

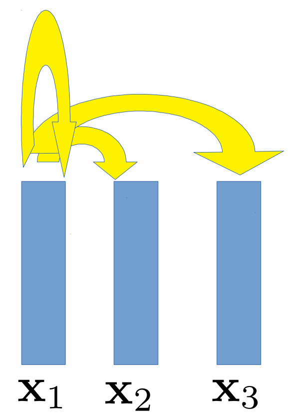


Self attention

- An attention mechanism which considers its own elements of a sequence
- Different to the attention mechanisms related to RNNs
- Forms the basis of Transformers



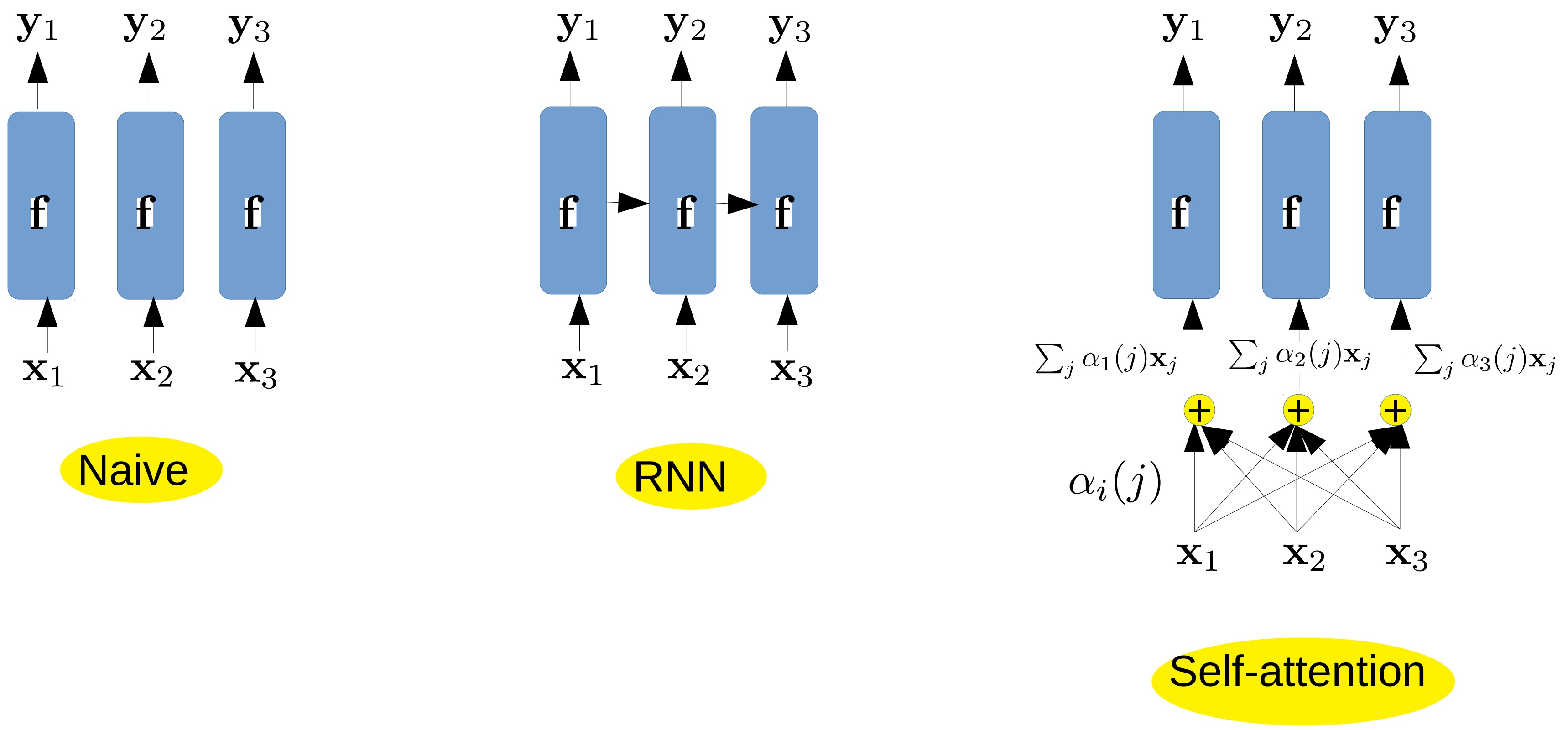
cross-attention



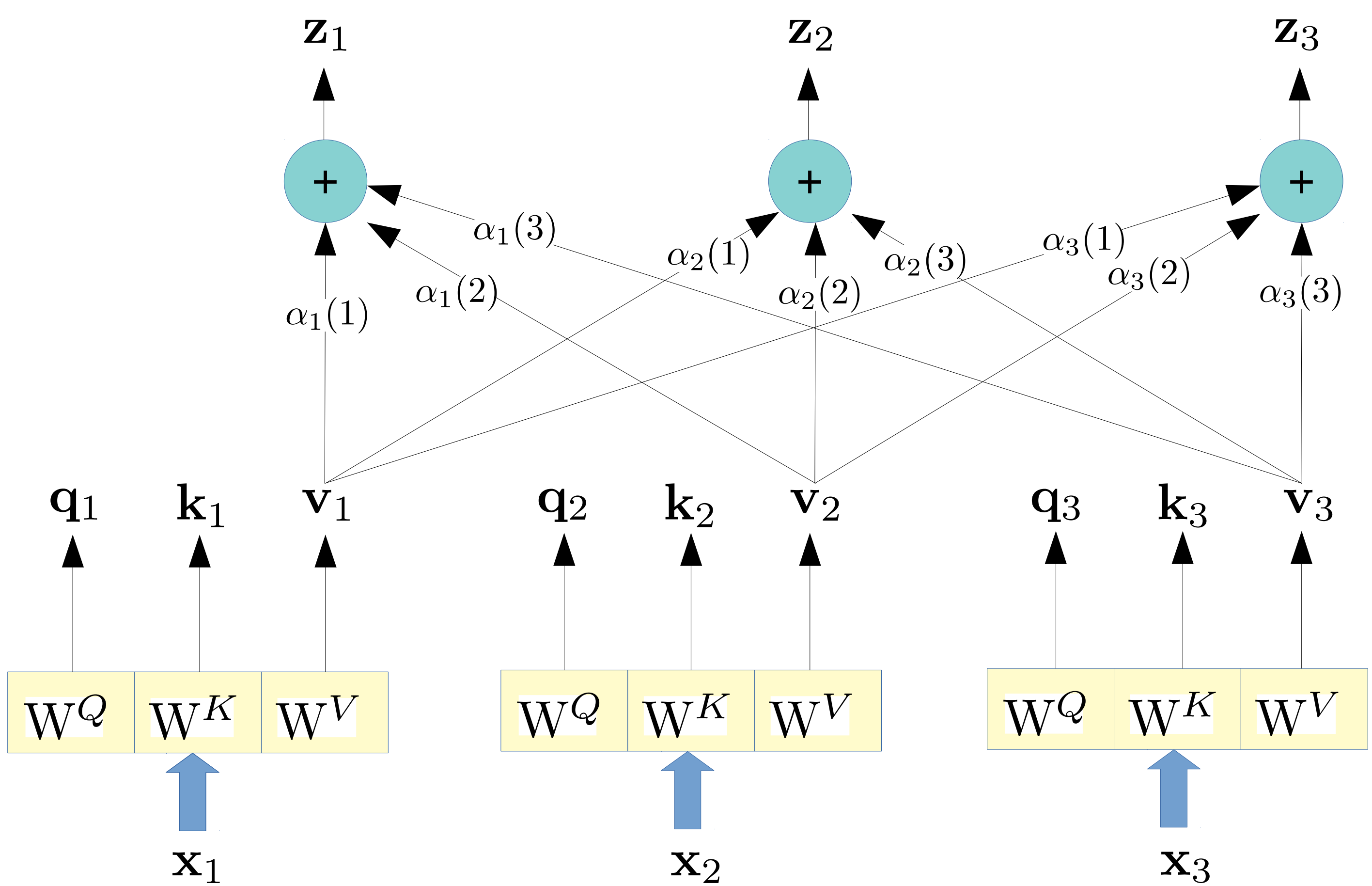
Self-attention

The intuition

- Make the output a function of variable size sequence
- The number of learnable parameters should be a constant.



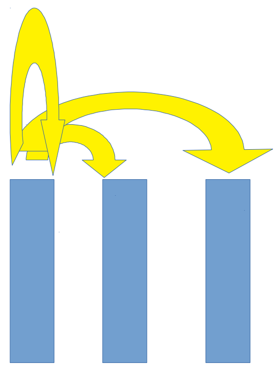
Self-attention mechanics (I)



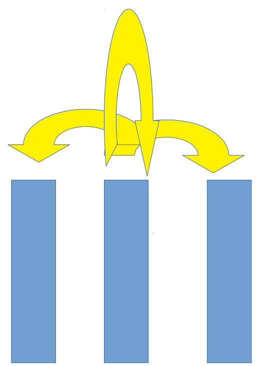
$$\mathbf{q}_i = \mathbf{x}_i W^Q$$
$$\mathbf{k}_i = \mathbf{x}_i W^K$$
$$\mathbf{v}_i = \mathbf{x}_i W^V$$

Self-attention mechanics (II)

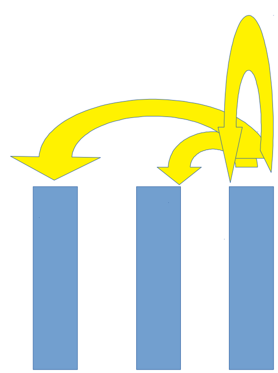
$\boldsymbol{q}_1$	$\boldsymbol{q}_2$	$\boldsymbol{q}_3$
$\boldsymbol{k}_1$	$\boldsymbol{k}_2$	$\boldsymbol{k}_3$
$\boldsymbol{v}_1$	$\boldsymbol{v}_2$	$\boldsymbol{v}_3$
$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$



$$\alpha_1(1) = \frac{\exp(\boldsymbol{q}_1\boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_1\boldsymbol{k}_j^T)} \qquad \alpha_1(2) = \frac{\exp(\boldsymbol{q}_1\boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_1\boldsymbol{k}_j^T)} \qquad \alpha_1(3) = \frac{\exp(\boldsymbol{q}_1\boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_1\boldsymbol{k}_j^T)}$$

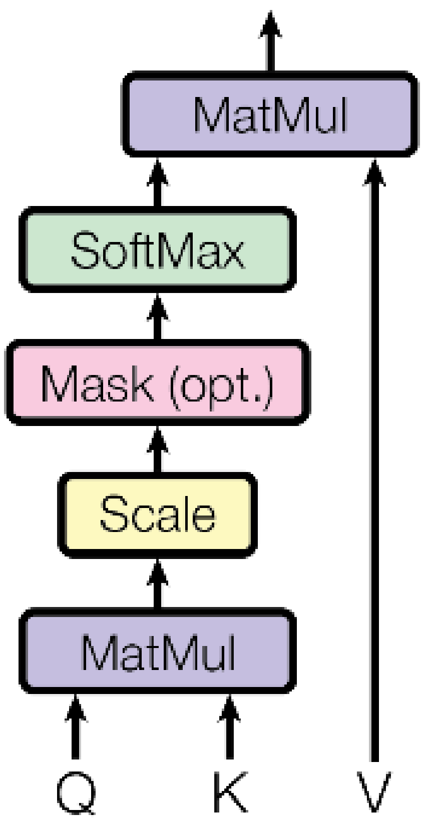


$$\alpha_2(1) = \frac{\exp(\boldsymbol{q}_2\boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_2\boldsymbol{k}_j^T)} \qquad \alpha_2(2) = \frac{\exp(\boldsymbol{q}_2\boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_2\boldsymbol{k}_j^T)} \qquad \alpha_2(3) = \frac{\exp(\boldsymbol{q}_2\boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_2\boldsymbol{k}_j^T)}$$



$$\alpha_3(1) = \frac{\exp(\boldsymbol{q}_3\boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_3\boldsymbol{k}_j^T)} \qquad \alpha_3(2) = \frac{\exp(\boldsymbol{q}_3\boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_3\boldsymbol{k}_j^T)} \qquad \alpha_3(3) = \frac{\exp(\boldsymbol{q}_3\boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_3\boldsymbol{k}_j^T)}$$

Scaled dot product attention



Input word vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]^T$

Query $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_n]^T$

Keys $\boldsymbol{K} = [\boldsymbol{k}_1, \boldsymbol{k}_2, \cdots, \boldsymbol{k}_n]^T$

Values $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n]^T$

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

$\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V$ Trainable weight vectors

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Self-attention Pros and Cons

- Advantages
  - Parallelizable
  - All elements gets equal treatment
  -
- Disadvantages
  - High computational complexity for long sequences
  - Position information gets lost