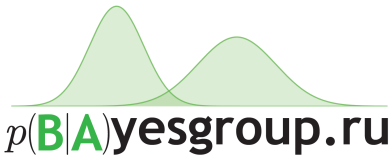
Recurrent neural networks

Ekaterina Lobacheva
lobacheva.tjulja@gmail.com



Deep Learning
CMC MSU, 2017

Memory

LEARNING TO EXECUTE

Can LSTM learn to execute python code?

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

Target: 12184.

[Zaremba, Sutskever, 2014]

LEARNING TO EXECUTE

LSTM reads the entire input one character at a time and produces the output one character at a time.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

Target: 12184.

Input:

```
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
```

Target: hkhpg

[Zaremba, Sutskever, 2014]

LEARNING TO EXECUTE

Training data: Python short code that can be evaluated in O(n) time & O(1) memory

- **length parameter:** constrain the integer in a maximum length.
- **nesting parameter:** constrain the number of times to combine operations.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else 5308)
```

Target: 12184.

an example of length = 4, nesting = 3

[Zaremba, Sutskever, 2014]

Curriculum learning

Idea: use training examples in some meaningful order through learning process

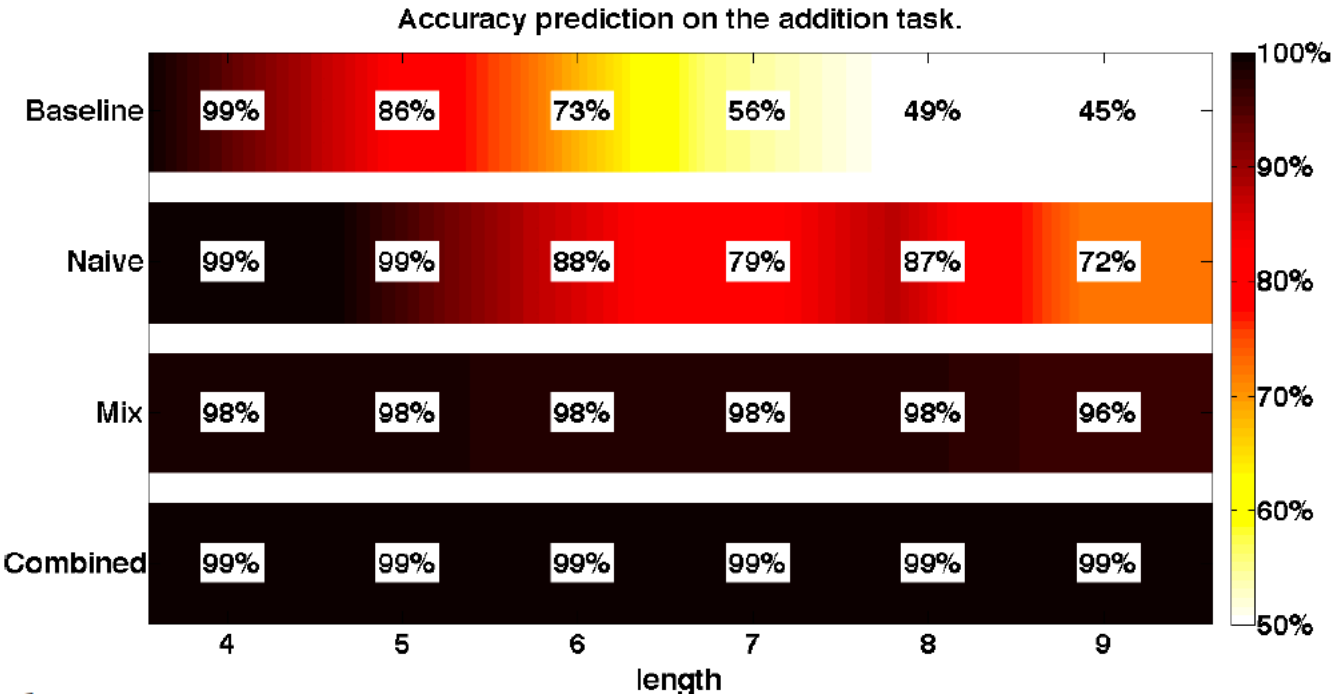
Let's say we want to learn to evaluate programs of length = a, nesting = b.

- **Baseline:** training examples with length = a, nesting = b.
- **Naive:** start with length = 1, nesting = 1 and gradually increase until length = a, nesting = b.
- **Mix:** to generate a example, first pick a random length from [1, a], and a random nesting from [1, b].
- **Combined:** a combination of naive and mix.

[Zaremba, Sutskever, 2014]

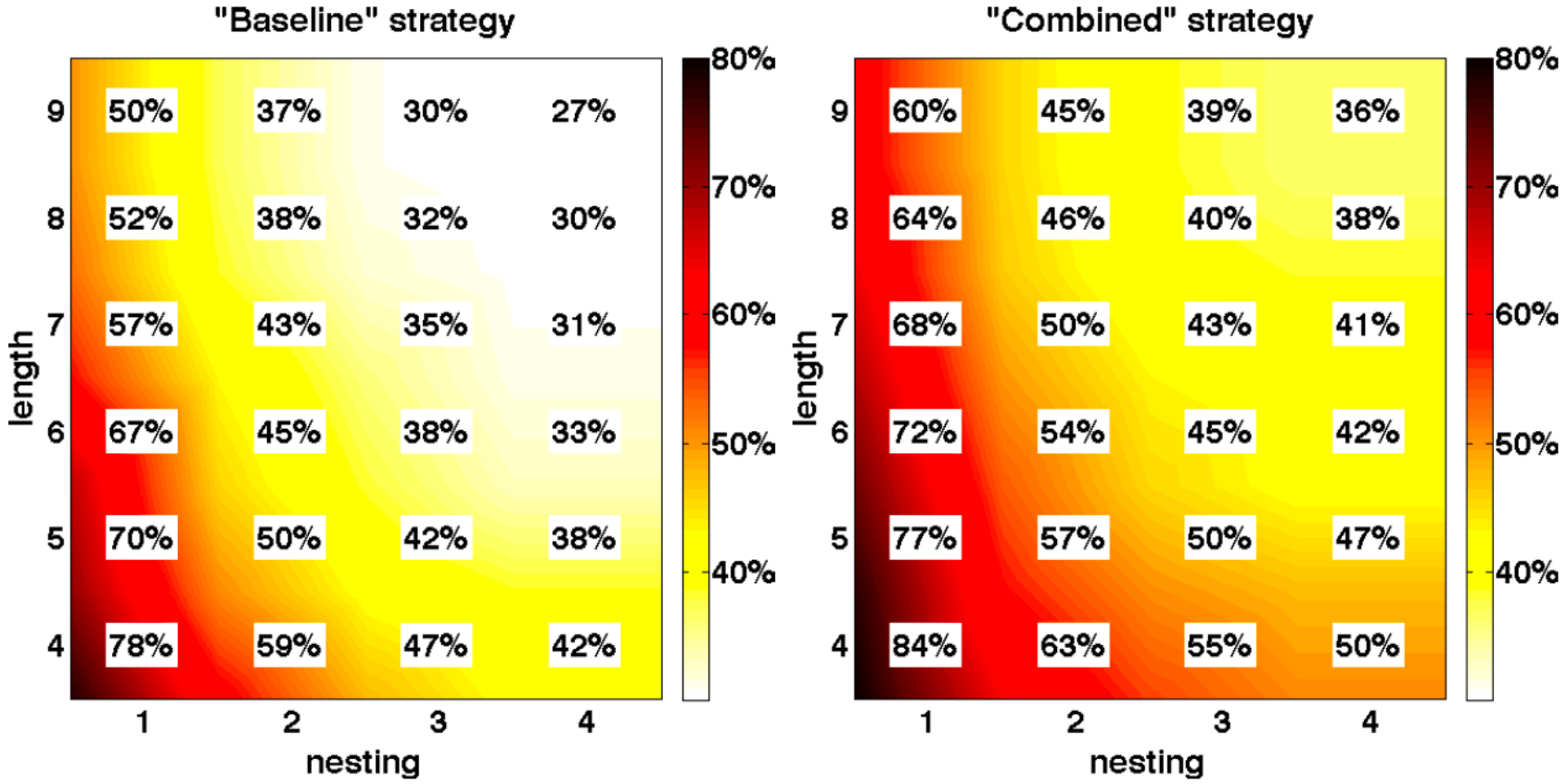
Addition task

Input:
`print(398345+425098)`
Target: 823443



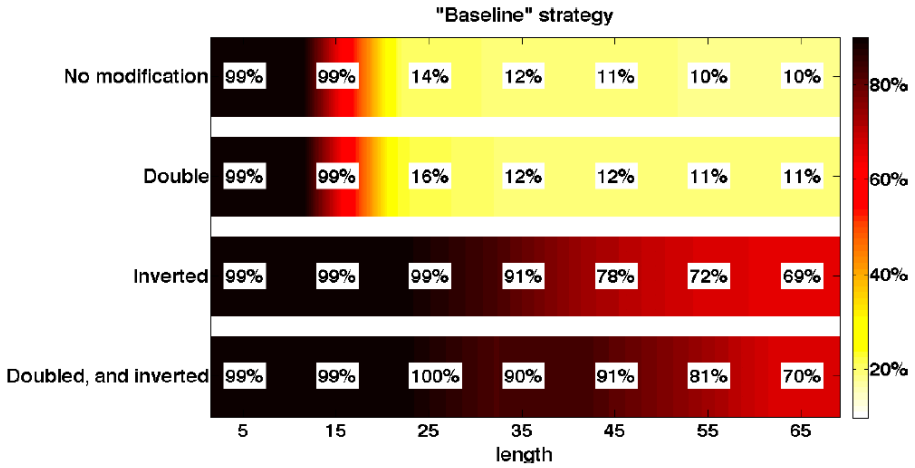
[Zaremba, Sutskever, 2014]

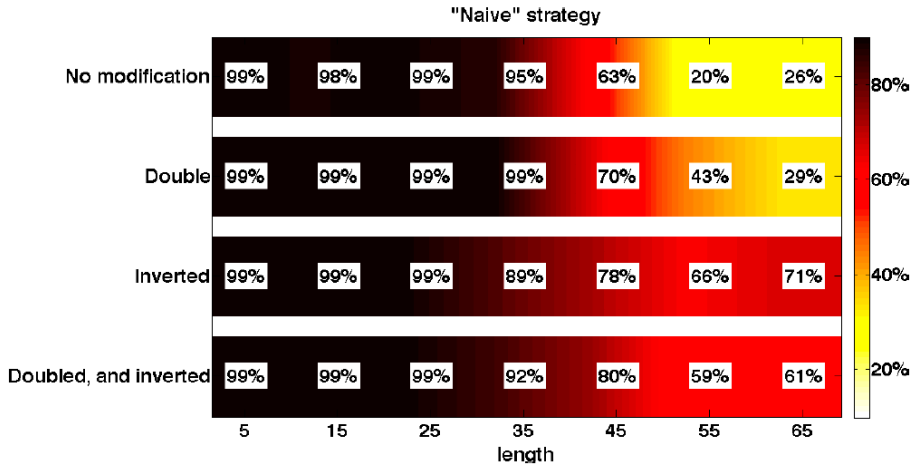
Program execution task

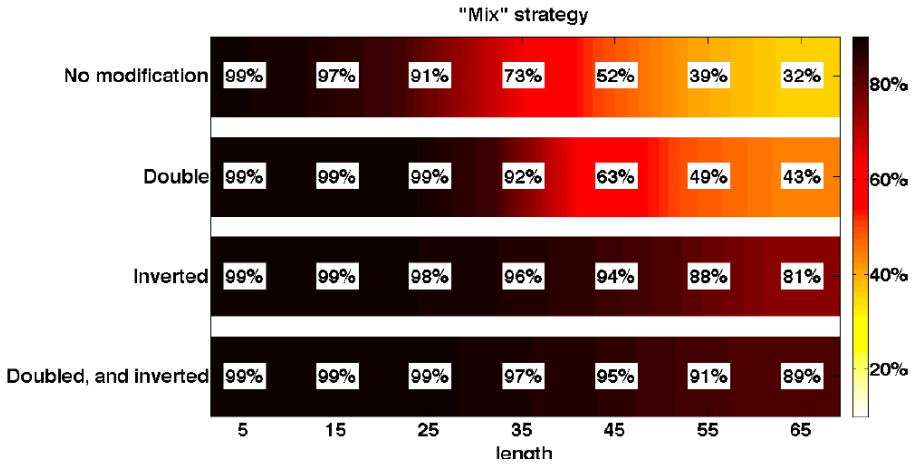


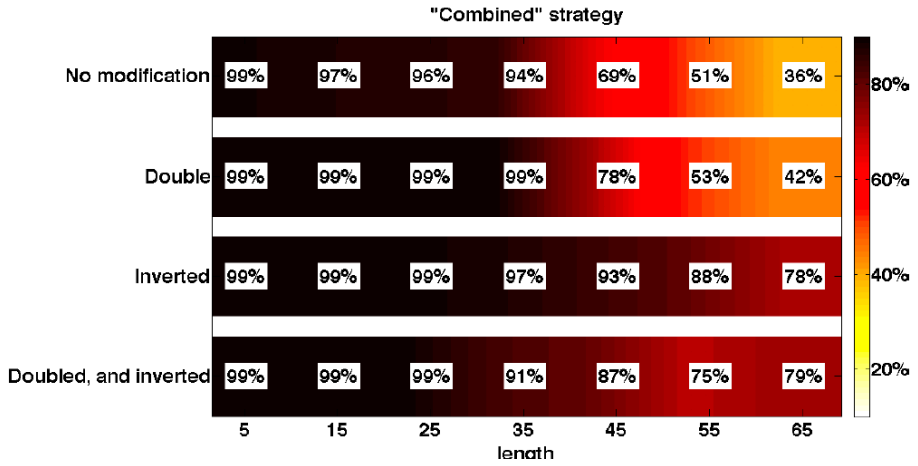
[Zaremba, Sutskever, 2014]

Memorization task



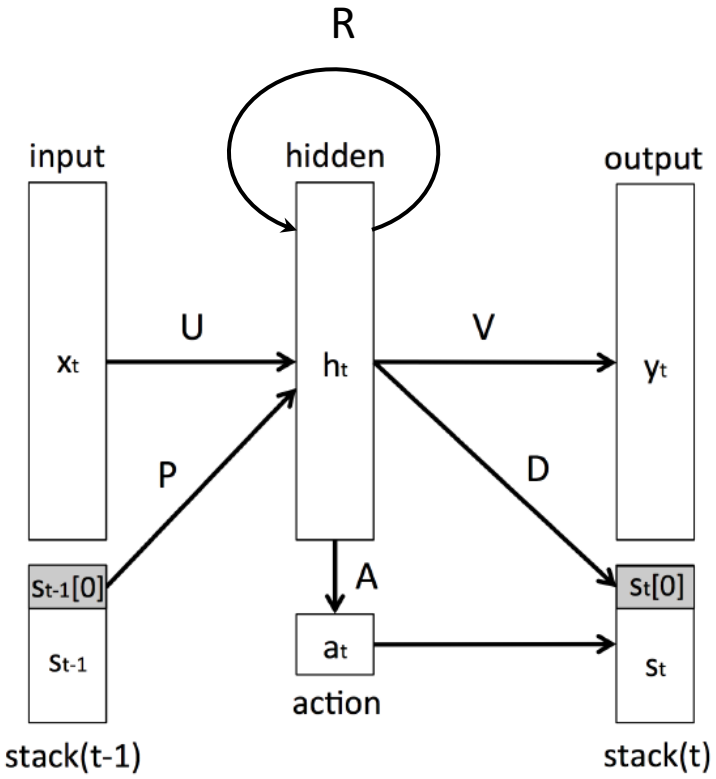






[Zaremba, Sutskever, 2014]

Stack-Augmented Recurrent Nets



$$h_t = \sigma\left(Ux_t + Rh_{t-1} + Ps^k_{t-1}\right)$$

$$a_t = f\left(Ah_t\right)$$

f is a softmax function.

$$s_t[0] = a_t[\text{PUSH}]\sigma(Dh_t) + a_t[\text{POP}]s_{t-1}[1] + a_t[\text{NO-OP}]s_{t-1}[0].$$

$$s_t[i] = a_t[PUSH]s_{t-1}[i-1] + a_t[POP]s_{t-1}[i+1] + a_t[NO-OP]s_{t-1}[i]$$

[Joulin, Mikolov,2015]

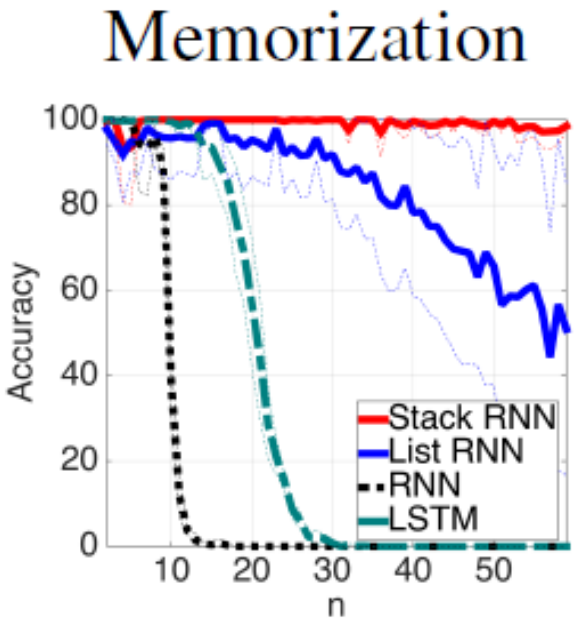
Stack-Augmented Recurrent Nets

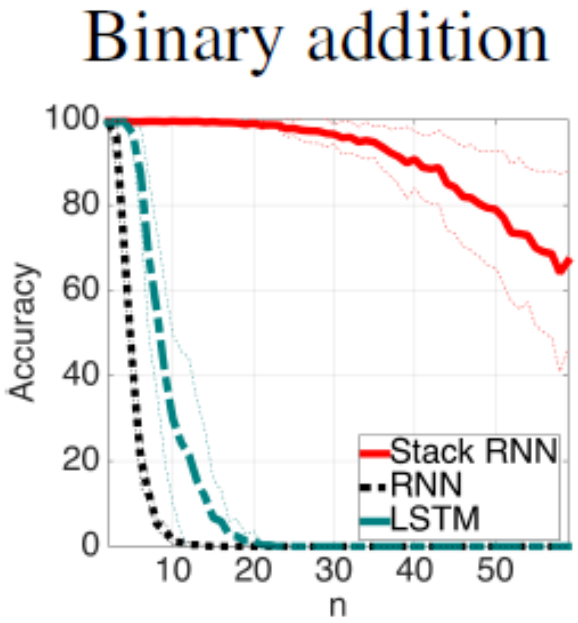
- Multiple stacks (10 in the experiments)
- Doubly-linked lists
- Discretizing the controllers at test time

[Joulin, Mikolov,2015]

Stack-Augmented Recurrent Nets

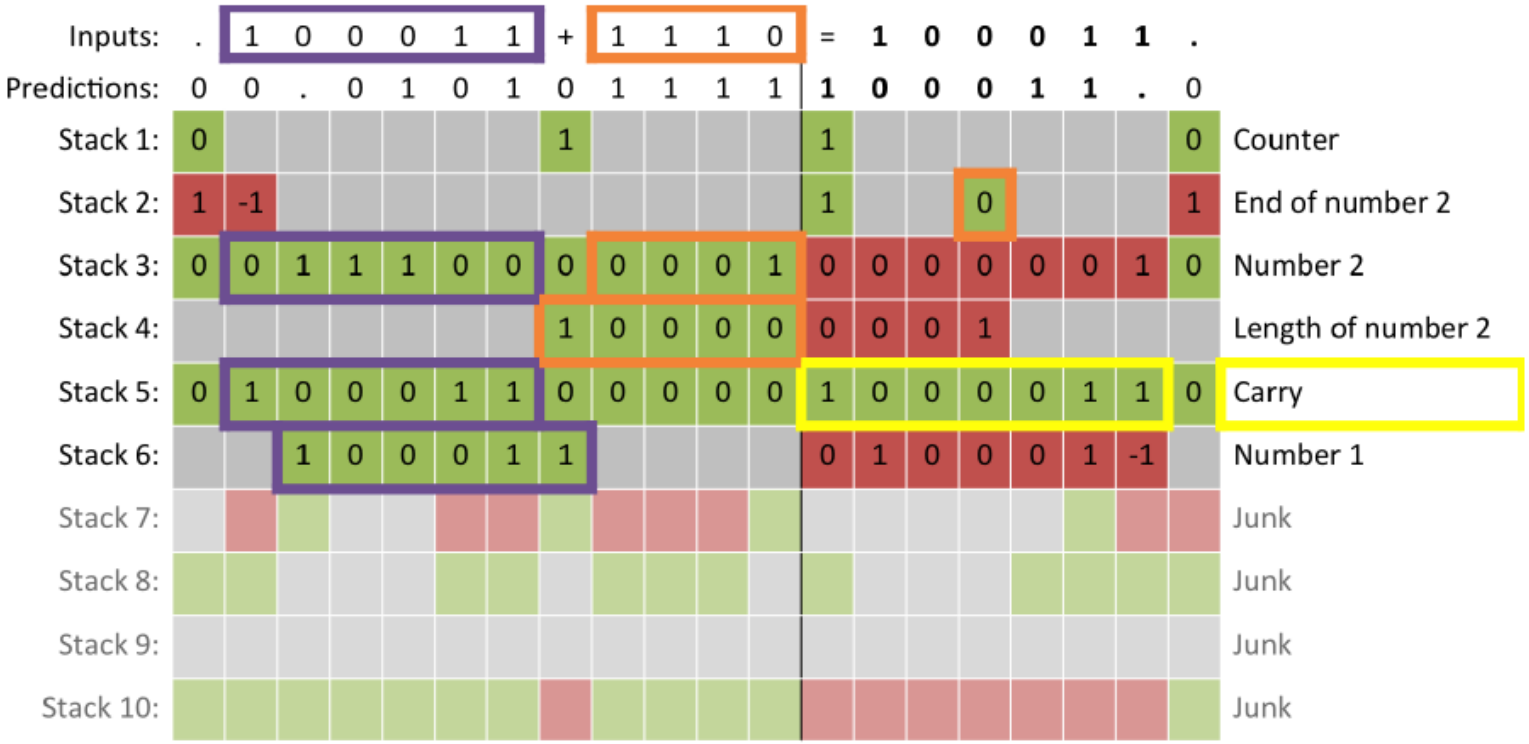
Train: n up to 20. Test: n up to 60.





[Joulin, Mikolov,2015]

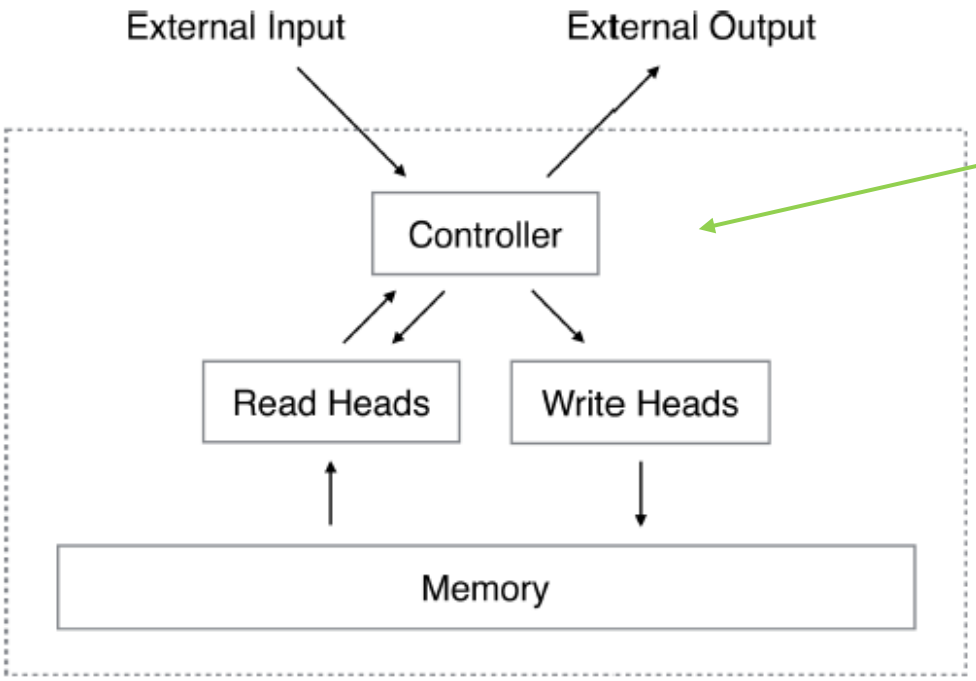
Stack-Augmented Recurrent Nets



Predict result in reverse

[Joulin, Mikolov,2015]

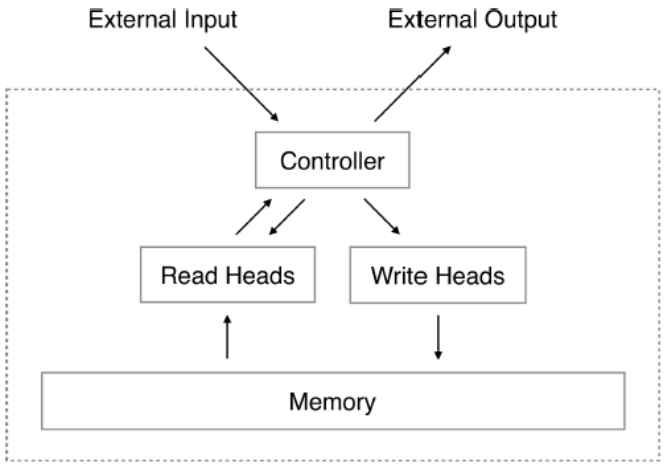
Neural Turing Machines



RNN, LSTM,
Feedforward NN

[Graves et al., 2014]

Neural Turing Machines



M_t - memory matrix at time t of size NxM

- N is the number of memory locations
- M is the vector size at each location

Reading

$$\sum_i w_t(i) = 1, \qquad 0 \le w_t(i) \le 1, \forall i.$$

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

Writing

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

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

[Graves et al., 2014]

Neural Turing Machines

Addressing Mechanisms

- Focusing by Content: use memory cells similar to key vector
- Focusing by Location: use memory cells which were used on the previous step
- + Shifting
- + Sharpening

[Graves et al., 2014]

Neural Turing Machines

Addressing Mechanisms

Focusing by Content: use memory cells similar to key vector

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(i)\right]\right)}{\sum_j \exp\left(\beta_t K\left[\mathbf{k}_t, \mathbf{M}_t(j)\right]\right)}.$$

- k_t - key vector of size M
- β_t - positive key strength
- K - cosine similarity

[Graves et al., 2014]

Neural Turing Machines

Addressing Mechanisms

Focusing by Location: use memory cells which were used on the previous step

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

g_t - scalar interpolation gate

[Graves et al., 2014]

Neural Turing Machines

Addressing Mechanisms

Shifting

$$\tilde{w}_t(i) \longleftarrow \sum_{j=0}^{N-1} w_t^g(j)\, s_t(i-j) \qquad \sum_i s_t(i) = 1$$

s_t - shift weighting

[Graves et al., 2014]

Neural Turing Machines

Addressing Mechanisms

Sharpening

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

γ_t - scalar sharpening parameter

[Graves et al., 2014]

Neural Turing Machines

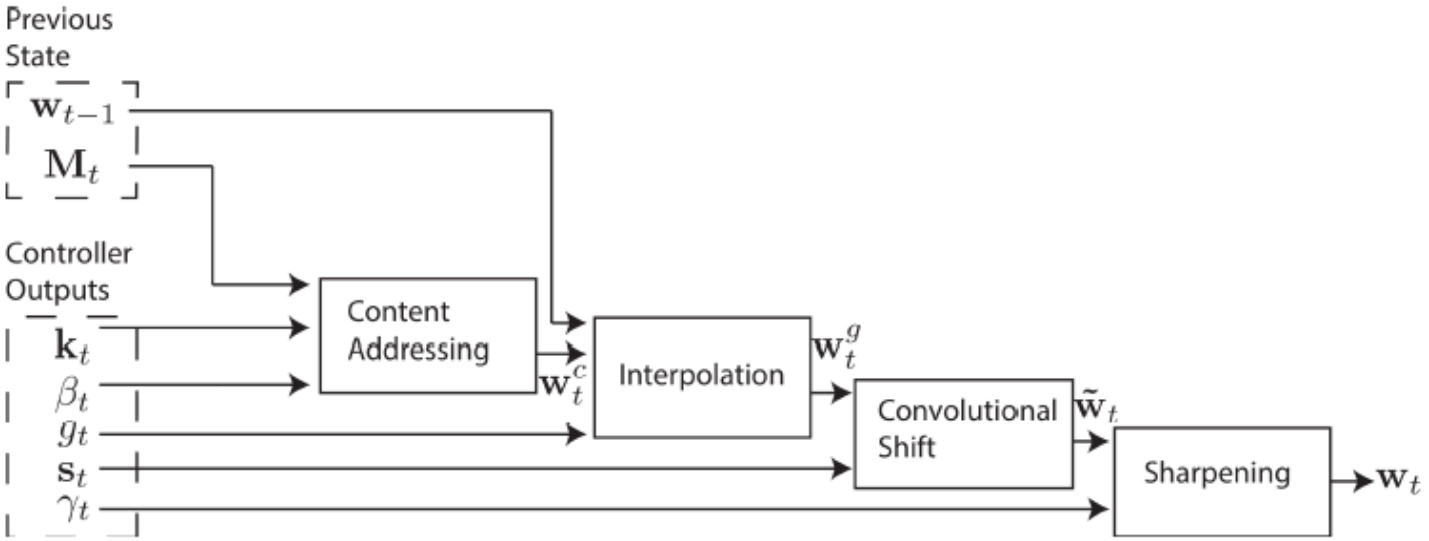


Figure 2: Flow Diagram of the Addressing Mechanism. The *key vector*, $\mathbf{k}_t$, and *key strength*, β_t, are used to perform content-based addressing of the memory matrix, $\mathbf{M}_t$. The resulting content-based weighting is interpolated with the weighting from the previous time step based on the value of the *interpolation gate*, g_t. The *shift weighting*, $\mathbf{s}_t$, determines whether and by how much the weighting is rotated. Finally, depending on γ_t, the weighting is sharpened and used for memory access.

[Graves et al., 2014]

Neural Turing Machines

Input (fed into the controller):

$$\mathbf{i}_t, \mathbf{r}_t \in \mathbb{R}^N$$

Output with which we use to manipulate M_t:

$$\mathbf{e}_t, \mathbf{a}_t, \mathbf{k}_t \in (0,1)^N$$

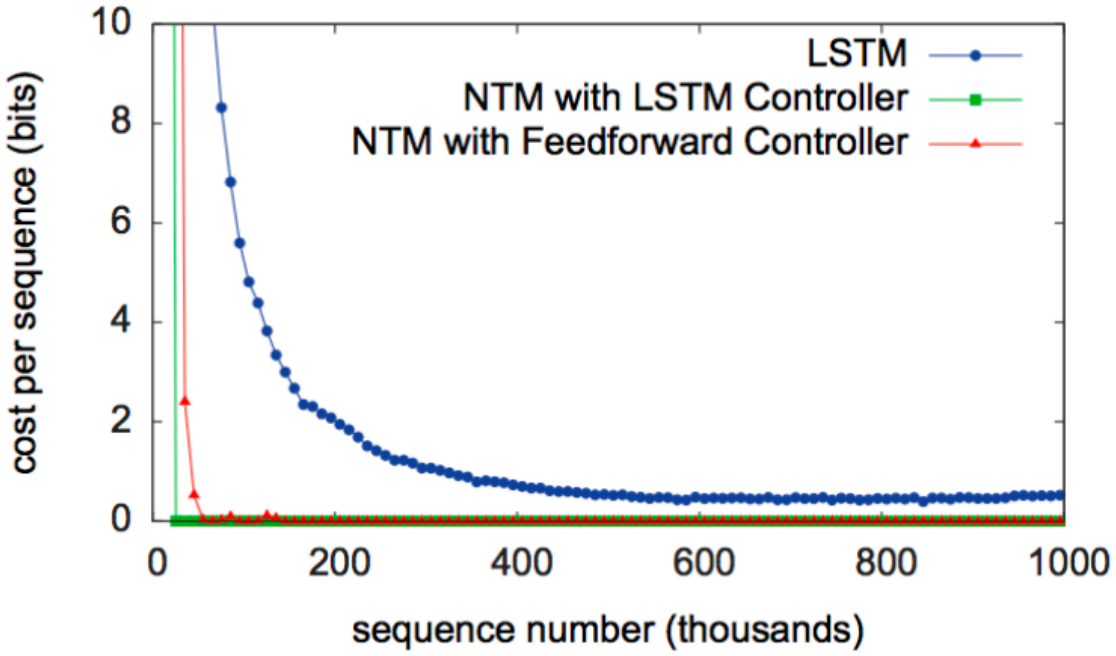
$$\mathbf{s}_t \in (0,1)^M, \qquad \sum_i s_t(i) = 1$$

$$\beta_t \in \mathbb{R}^+ \qquad \gamma_t \in \mathbb{R}^{\geq 1}$$

$$g_t \in (0,1)$$

[Graves et al., 2014]

NTM: Copy task

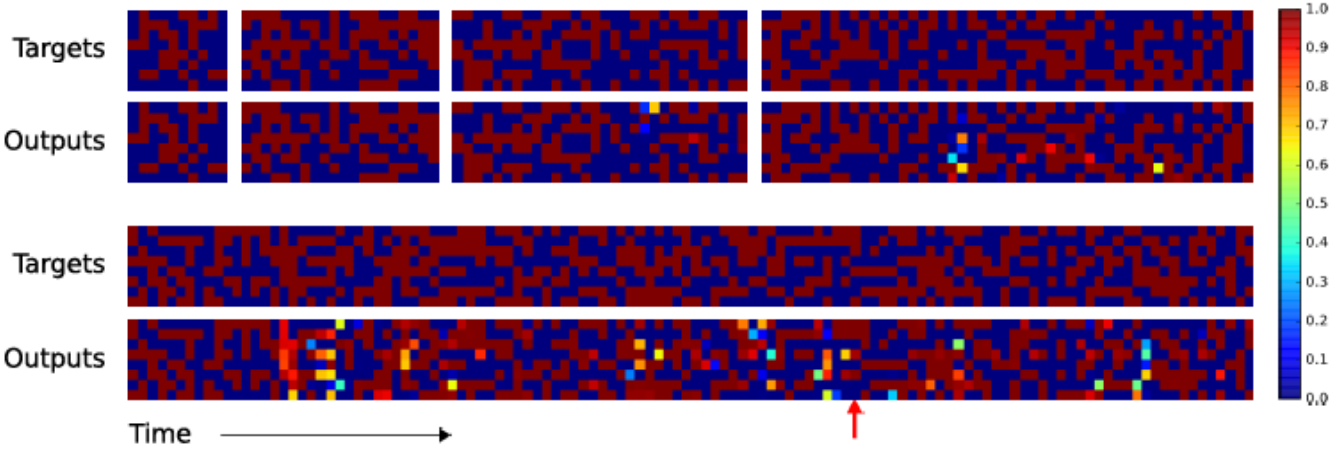


[Graves et al., 2014]

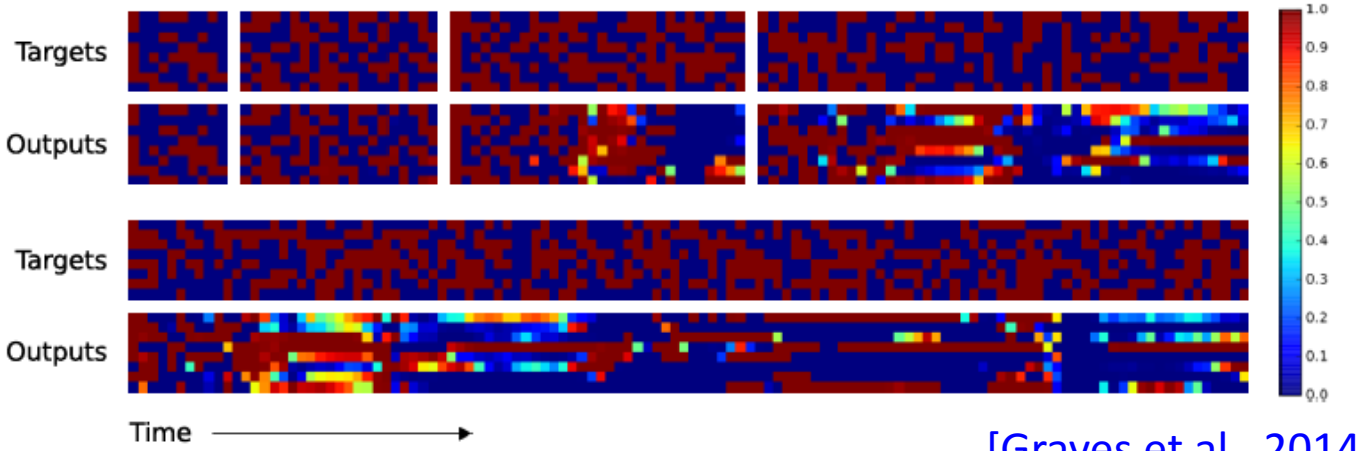
NTM: Copy task

Copy task. Train: n up to 20. Test: n up to 120.

NMT



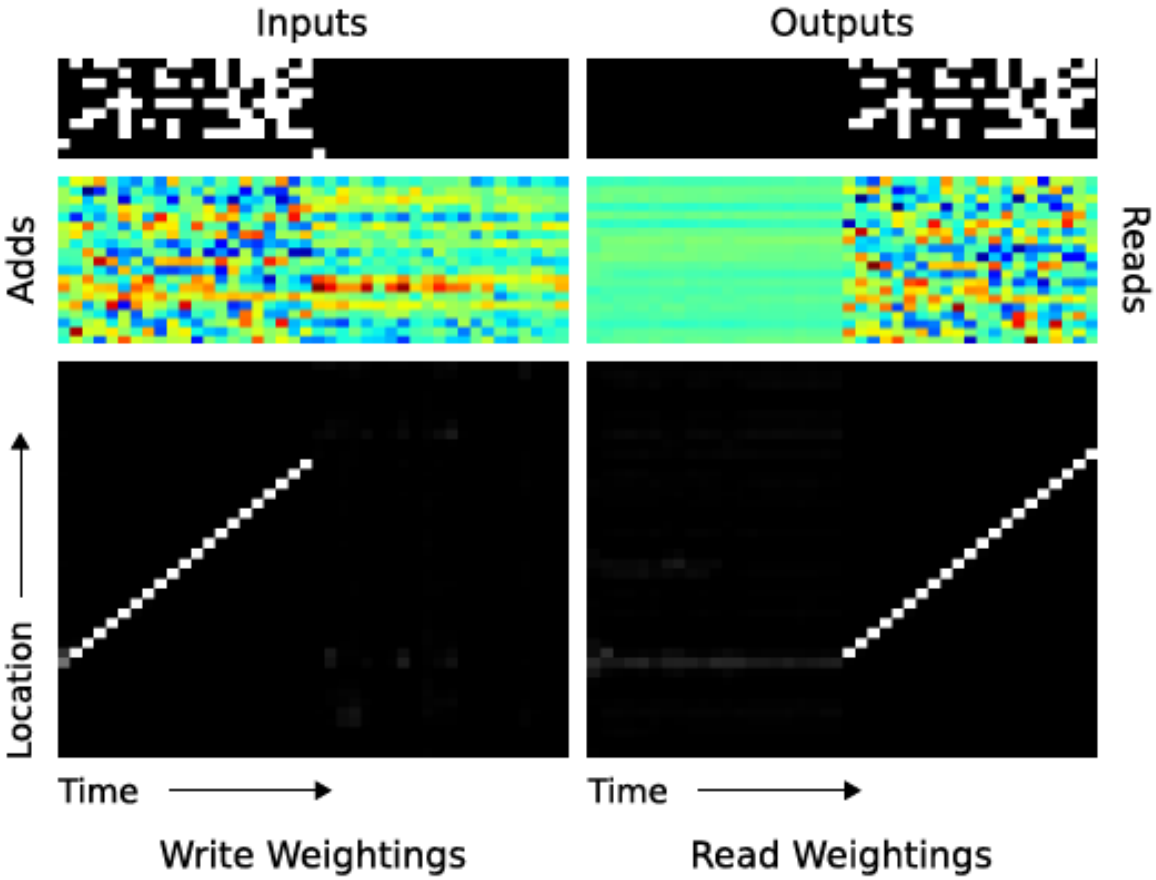
LSTM



[Graves et al., 2014]

NTM: Copy task

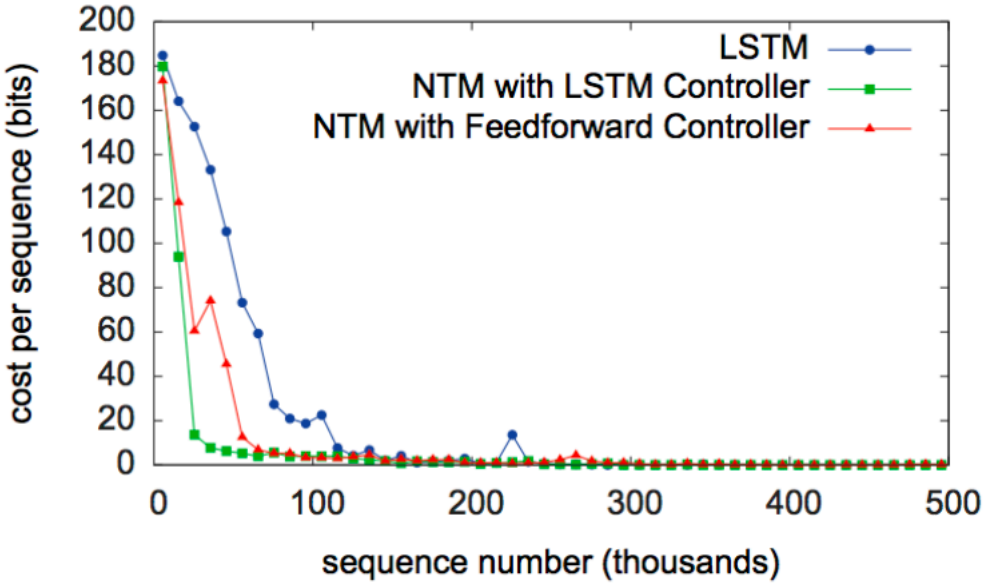
initialise: move head to start location
while input delimiter not seen **do**
  receive input vector
  write input to head location
  increment head location by 1
end while
return head to start location
while true **do**
  read output vector from head location
  emit output
  increment head location by 1
end while



[Graves et al., 2014]

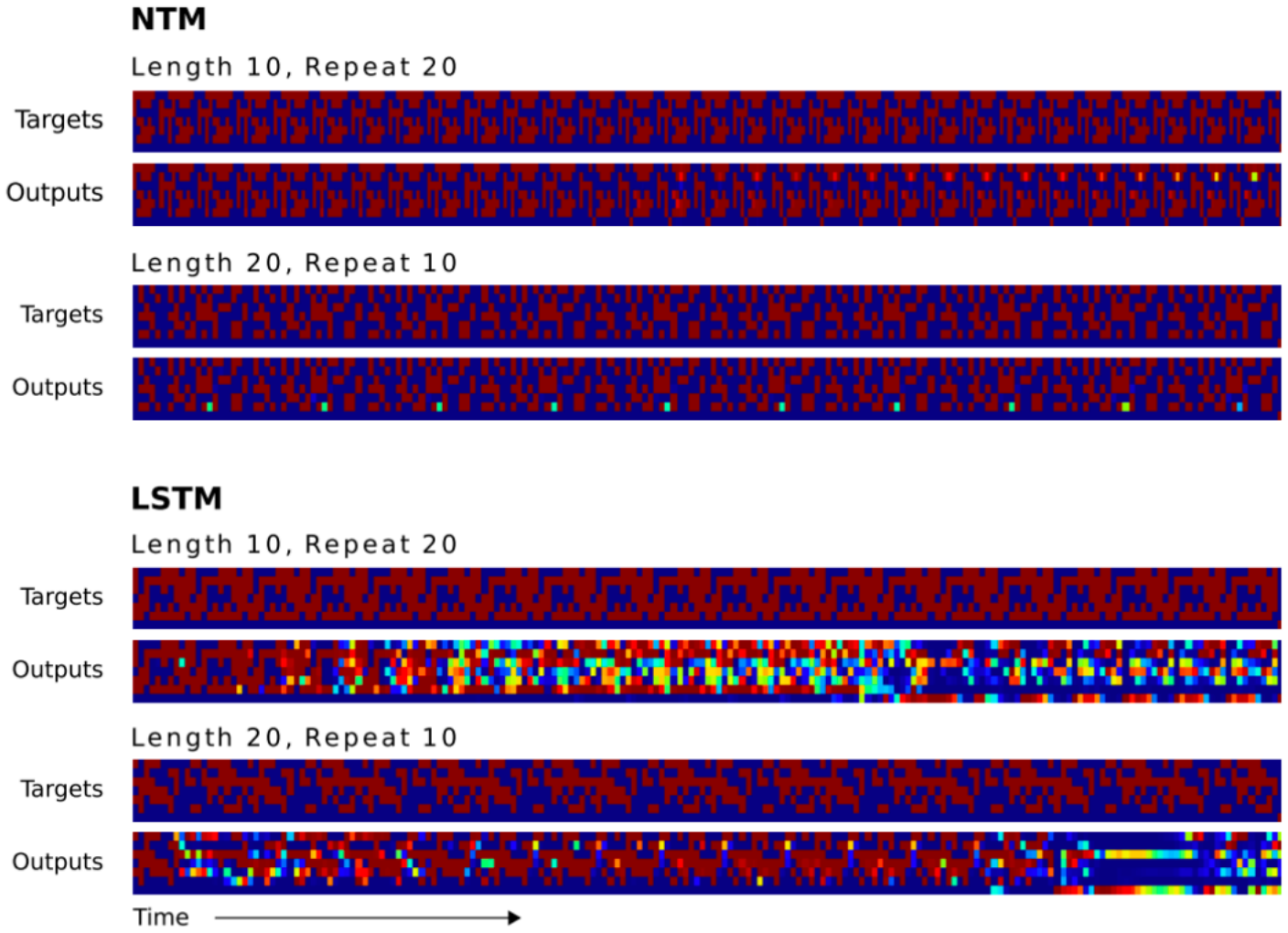
NTM: Repeat Copy task

Tests whether NTM can learn simple nested "for loop" function



[Graves et al., 2014]

NTM: Repeat Copy task

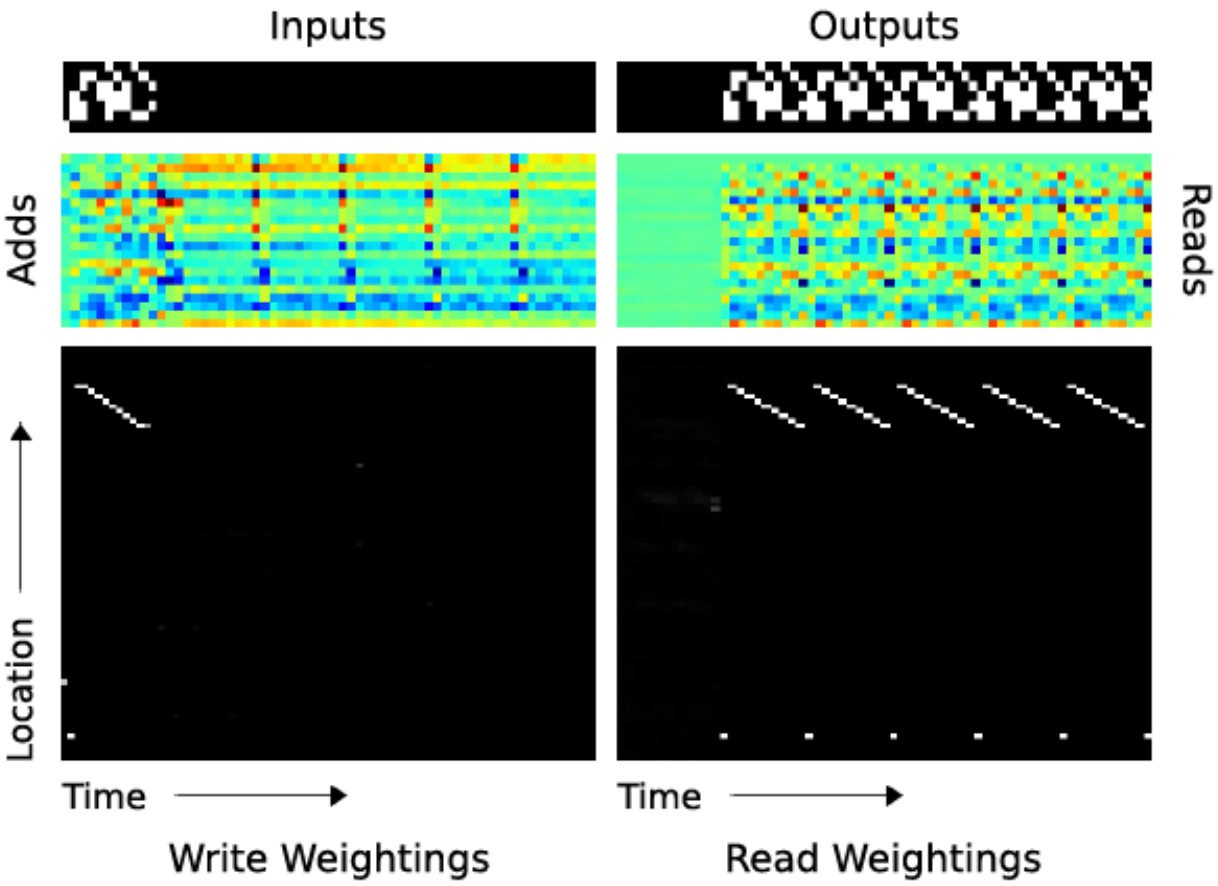


[Graves et al., 2014]

NTM: Repeat Copy task

Fails to figure out where to end. Unable to keep count of how many repeats it has completed.

Use another memory location to help switch back the pointer to the start.



[Graves et al., 2014]

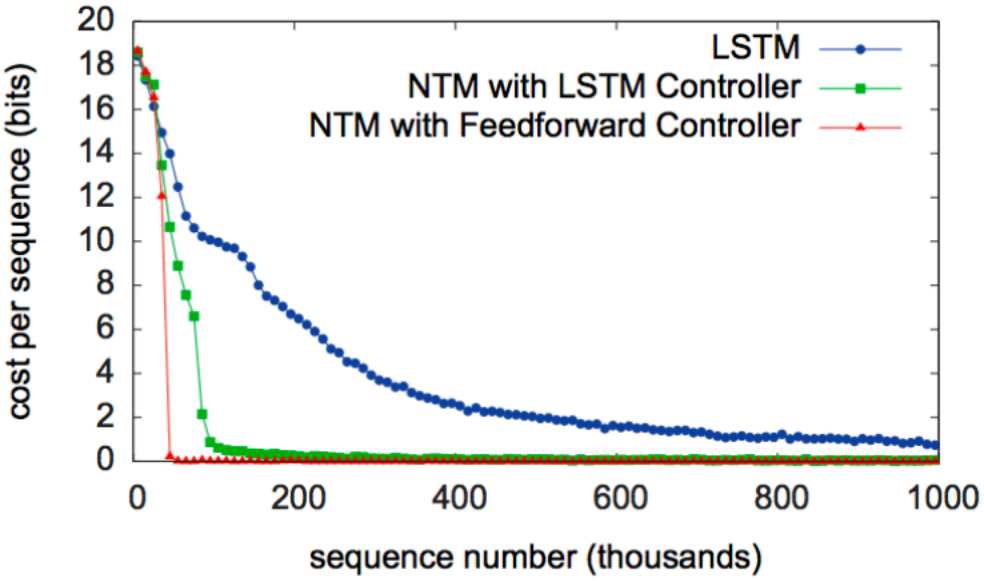
NTM: Associative Recall

- Training input is list of items, followed by a query item
- Output is subsequent item in list
- Each item is a three sequence 6---bit binary vector
- Each ‘episode’ has between two and six items

Tests NTM’s ability to associate data references

[Graves et al., 2014]

NTM: Associative Recall



[Graves et al., 2014]

NTM: Associative Recall

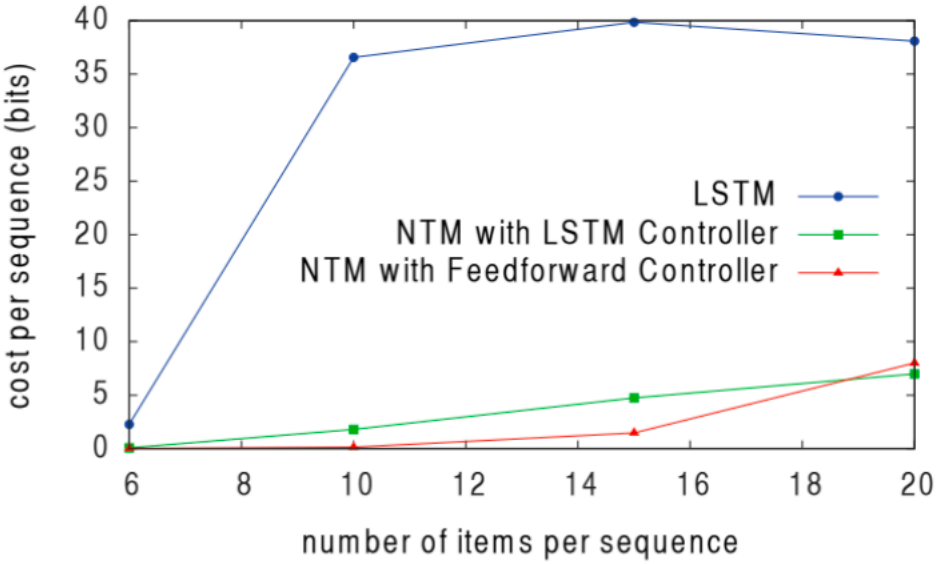


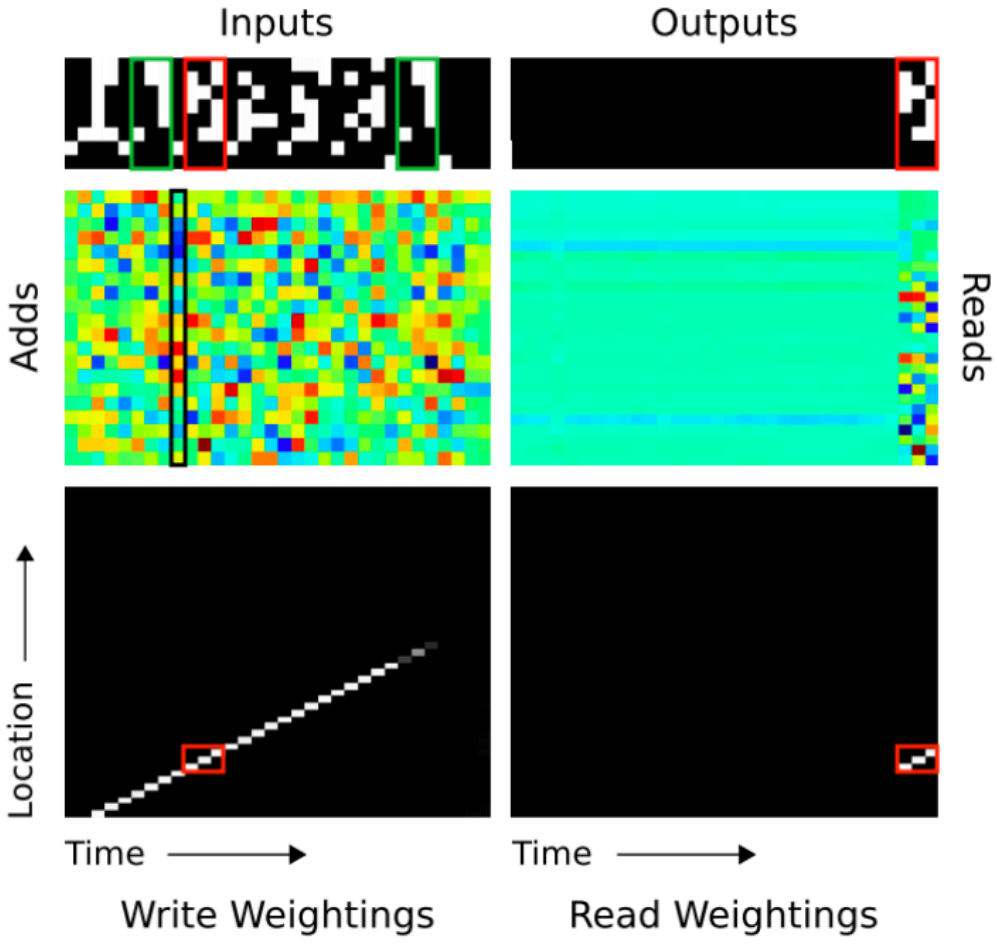
Figure 11: Generalisation Performance on Associative Recall for Longer Item Sequences. The NTM with either a feedforward or LSTM controller generalises to much longer sequences of items than the LSTM alone. In particular, the NTM with a feedforward controller is nearly perfect for item sequences of twice the length of sequences in its training set.

[Graves et al., 2014]

NTM: Associative Recall

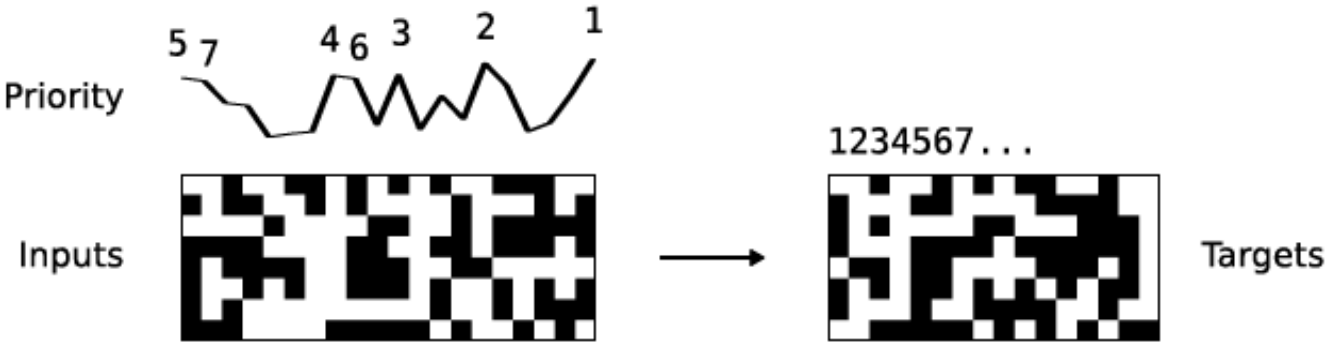
When each item delimiter is presented, the controller writes a compressed representation of the previous three time slices of the item.

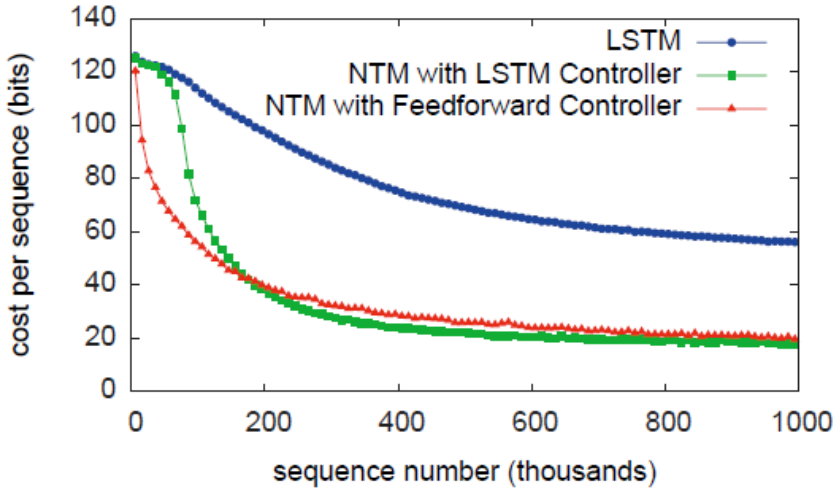
After the query arrives, the controller recomputes the same compressed representation of the query item, uses a content-based lookup to find the location where it wrote the first representation, and then shifts by one to produce the subsequent item in the sequence.



[Graves et al., 2014]

NTM: Sort task





[Graves et al., 2014]

Neural Turing Machines

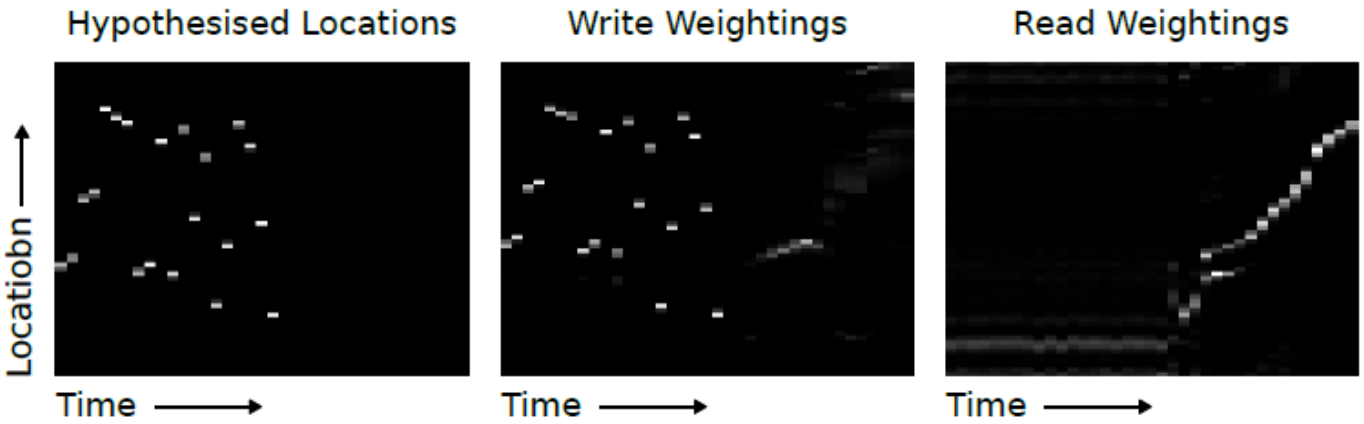


Figure 17: NTM Memory Use During the Priority Sort Task. Left: Write locations returned by fitting a linear function of the priorities to the observed write locations. Middle: Observed write locations. Right: Read locations.

[Graves et al., 2014]

Attention

Why attention in deep learning?

- **Better** interpretability
- **Faster** models
- **Stronger** results

Hard & soft attention

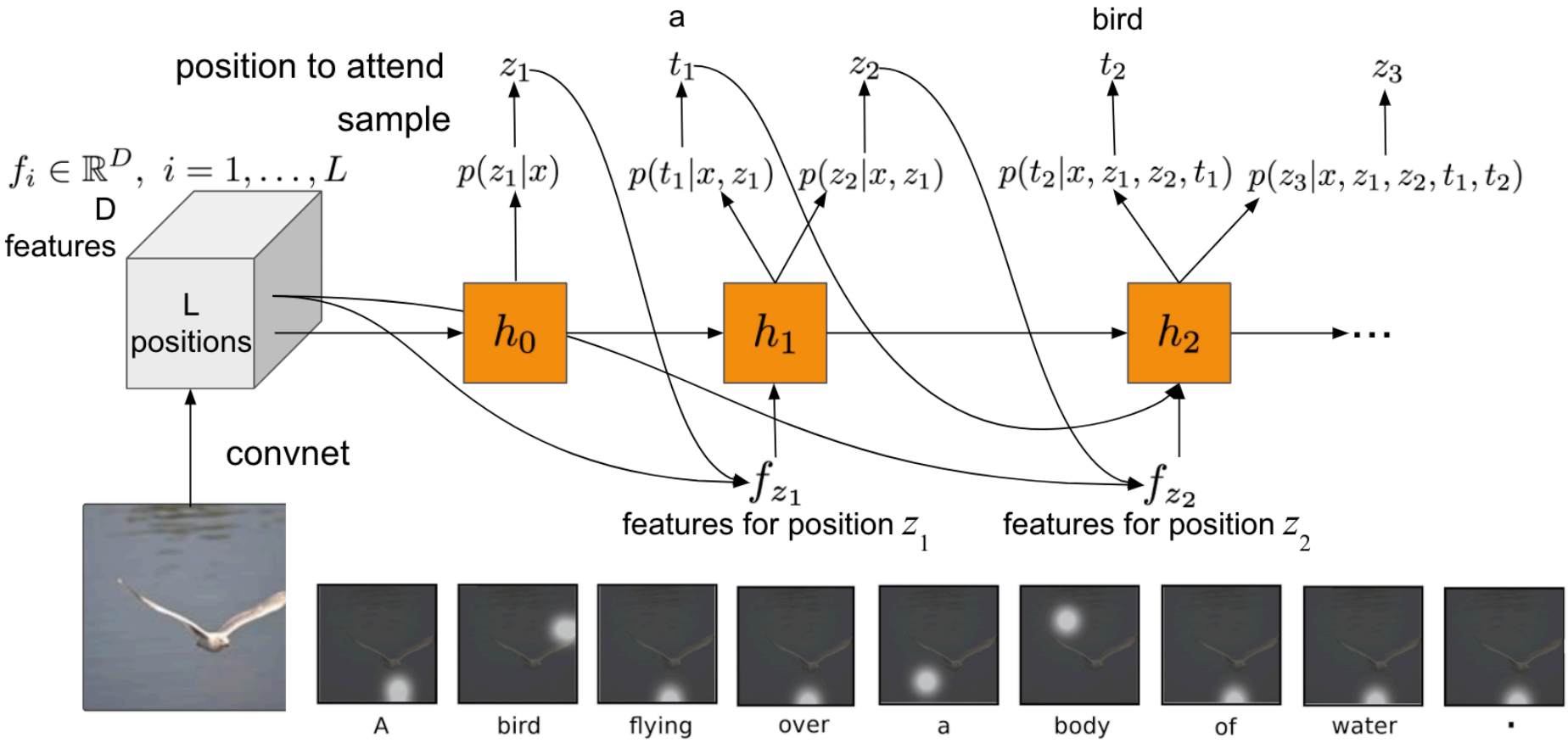
Hard

- Attention as stochastic process; sampling
- Supports discrete decisions (number of steps)
- Training with REINFORCE

Soft

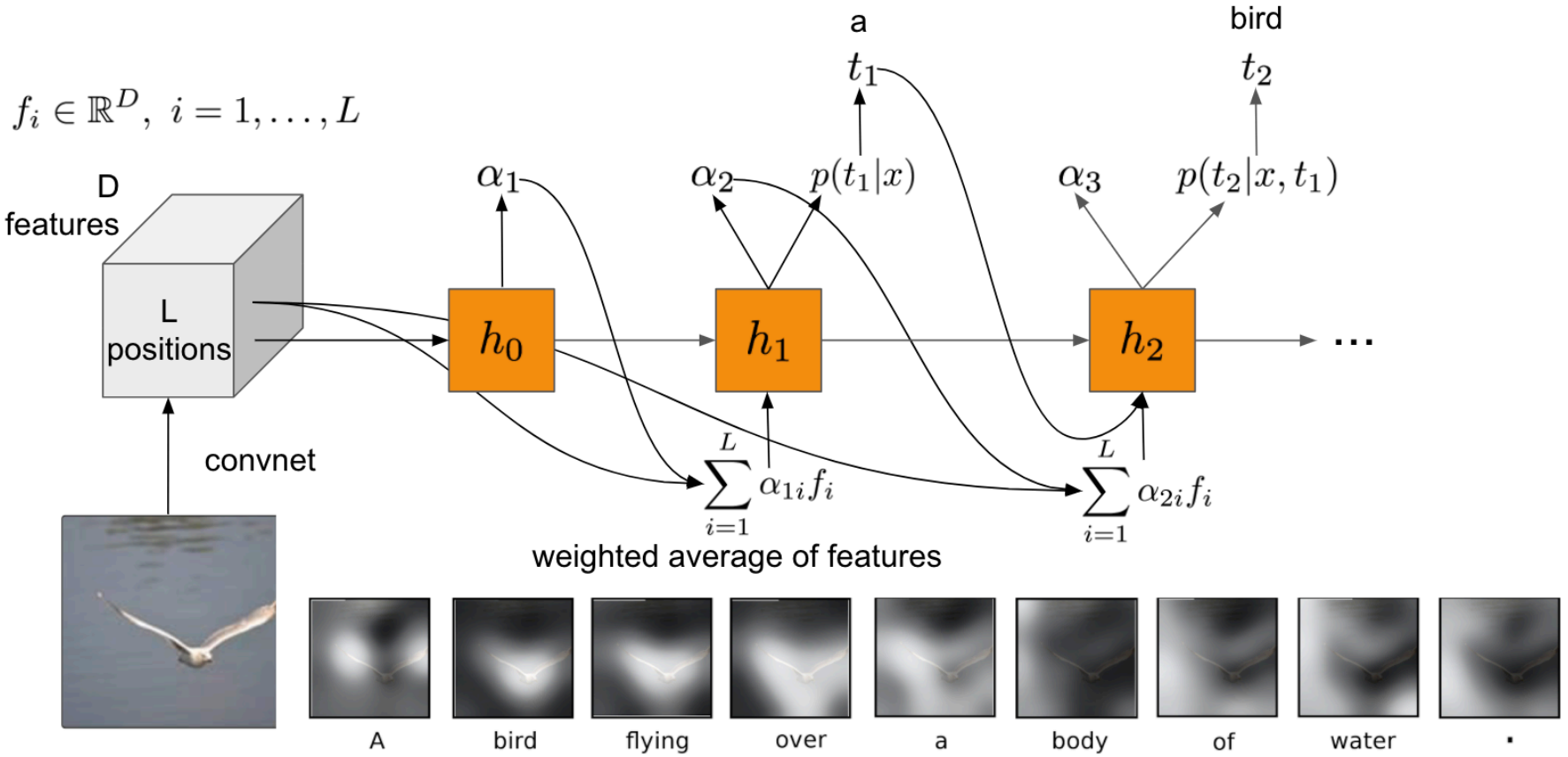
- Attention as differentiable layer
- No sampling
- Training with backprop

Hard attention for image captioning



[Xu et al. 2015]

Soft attention for image captioning



[Xu et al. 2015]

Hard & soft attention

$$z \sim \text{Discrete}(\alpha_1, \ldots, \alpha_L)$$

$$\alpha_i = \frac{\exp(\text{score}(h, f_i))}{\sum_{j=1}^{L} \exp(\text{score}(h, f_j))}$$

1 - additive attention
dim(*h*) != dim(*f*)

2,3 - multiplicative attention
dim(*h*) = dim(*f*)

Many ways to compute the score

- $\text{score}(h, f) = \text{NN}(h, f) = w_3^T \tanh(W_1 h + W_2 f)$ [1]
- $\text{score}(h, f) = h^T f$ [2]
- $\text{score}(h, f) = h^T f / \sqrt{d},\ d = \dim(h)$ [3]

[1] Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014
[2] Luong et al. "Effective approaches to attention-based neural machine translation", 2015
[3] Vaswani et al. "Attention Is All You Need", 2017

Introspection via soft attention

A stop sign is on a road with a mountain in the background.

A man is talking on his cell phone while another man watches.

[Xu et al. 2015]

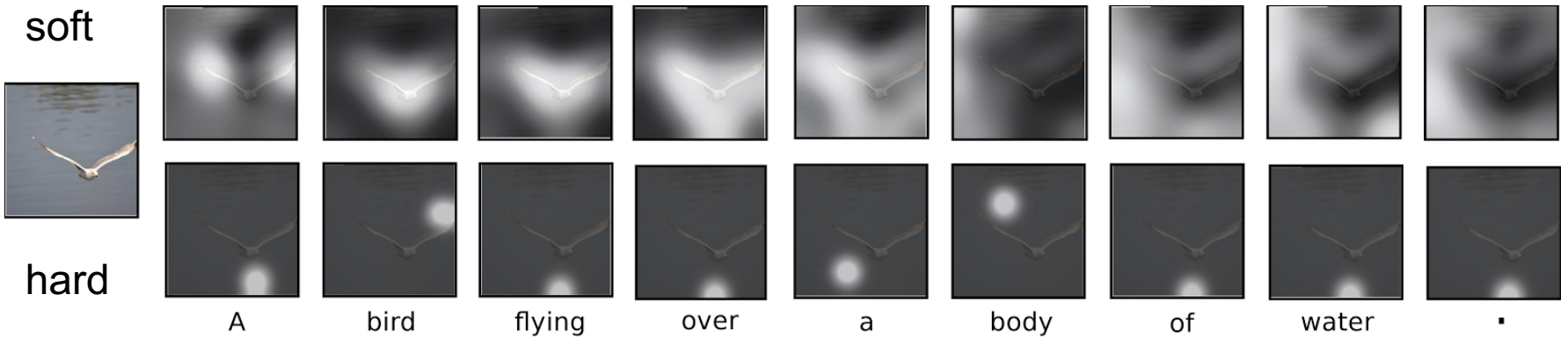
Soft vs. hard attention

Hard attention

- Better results (sometimes?)
- More principled
- Hard to train

Soft attention

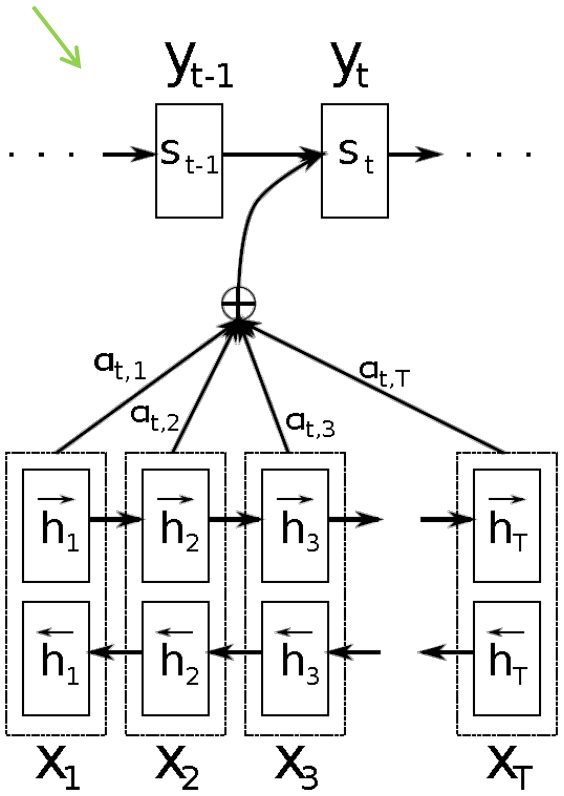
- Not always applicable
- Easy to train



[Xu et al. 2015]

Translation with attention

decoder RNN



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

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

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

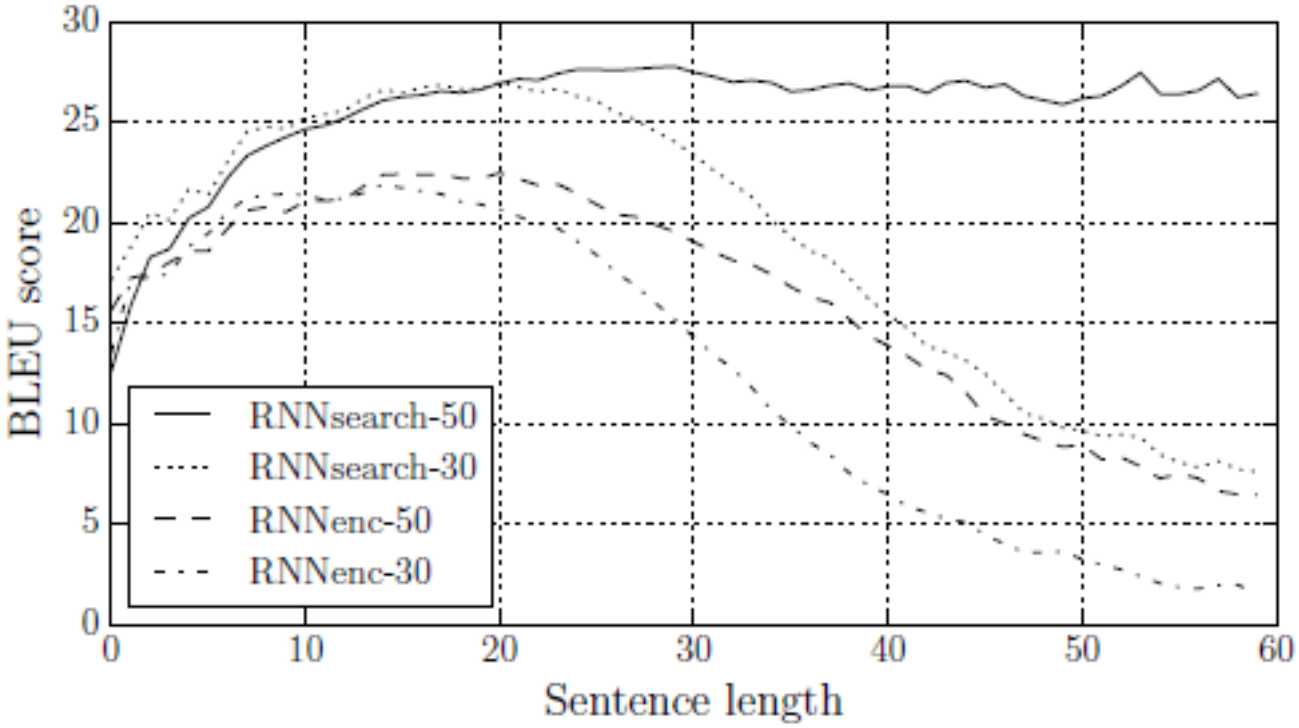
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

bidirectional RNN

[Bahdanau et al. 2015]

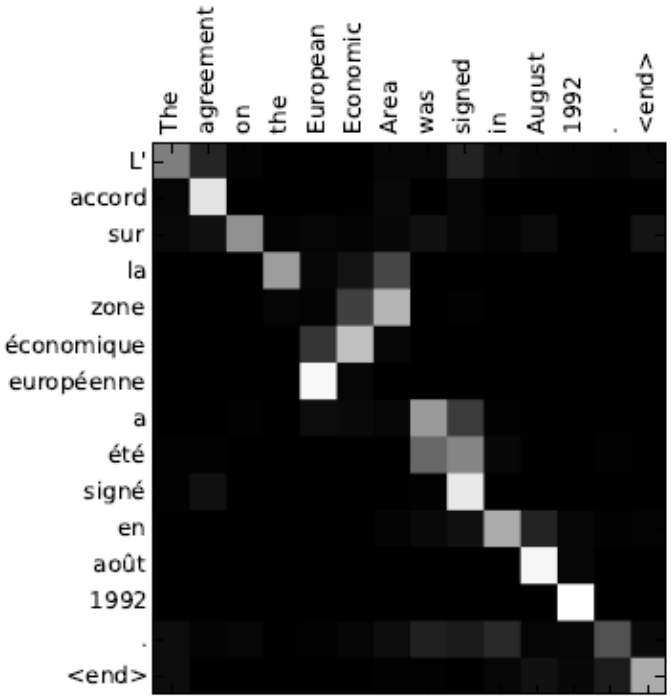
Demo

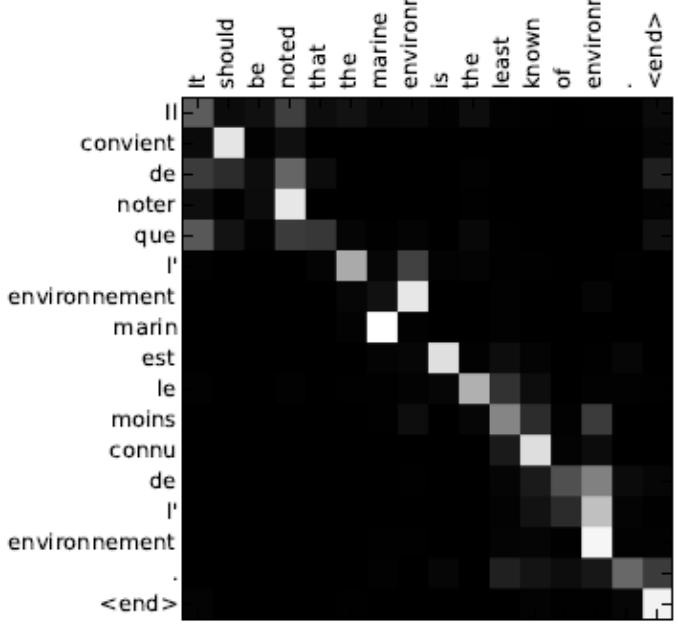
Translation with attention



[Bahdanau et al. 2015]

Translation with attention





[Bahdanau et al. 2015]

Attention Is All You Need

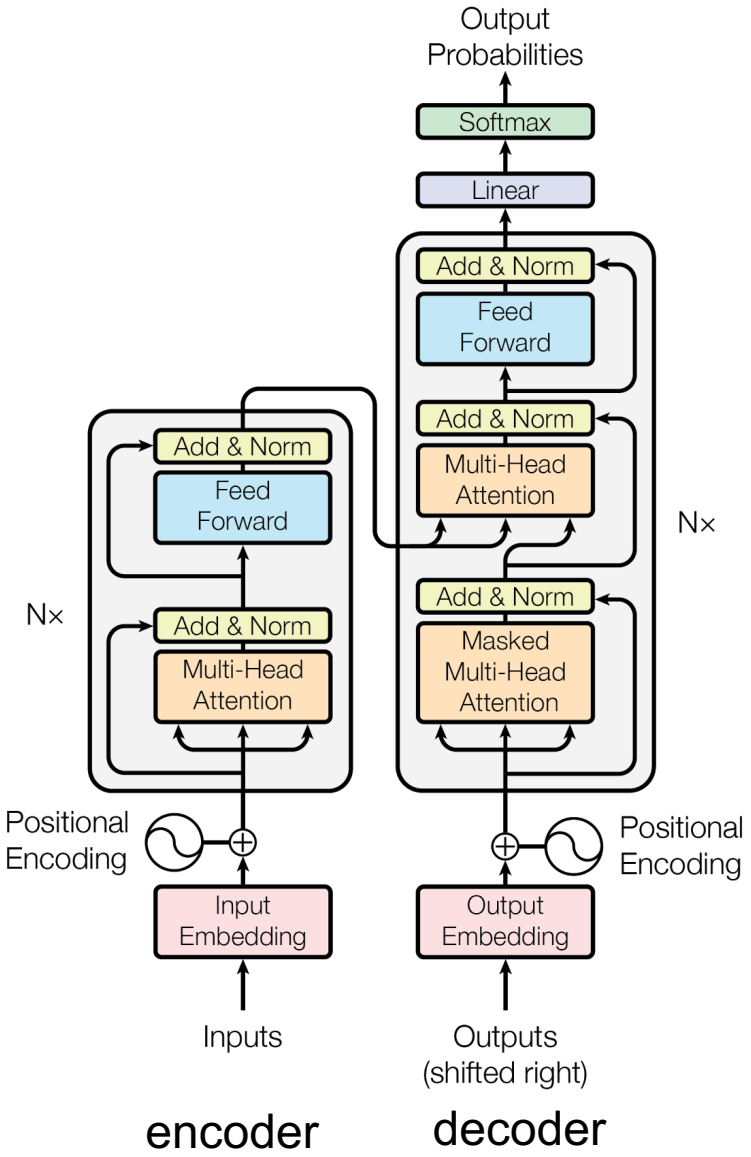
Replace LSTMs with **a lot** of attention! Apply to neural machine translation

- State-of-the art results
- Much less computation for training

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

[Vaswani et al. 2017]

Attention Is All You Need



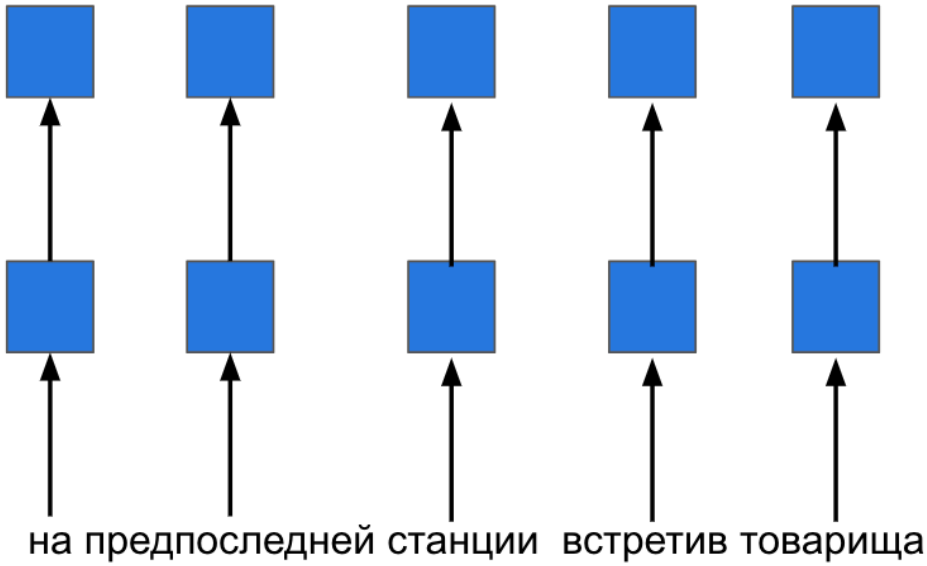
- Input layer
- Self-attention
- Per-word feedforward
- Attention over encoder outputs

[Vaswani et al. 2017]

Input layer

add positional encoding
encode offsets between words
not used in LSTMs

embedding
pick a vector for every word



[Vaswani et al. 2017]

Self-attention

every word attends to features of all words

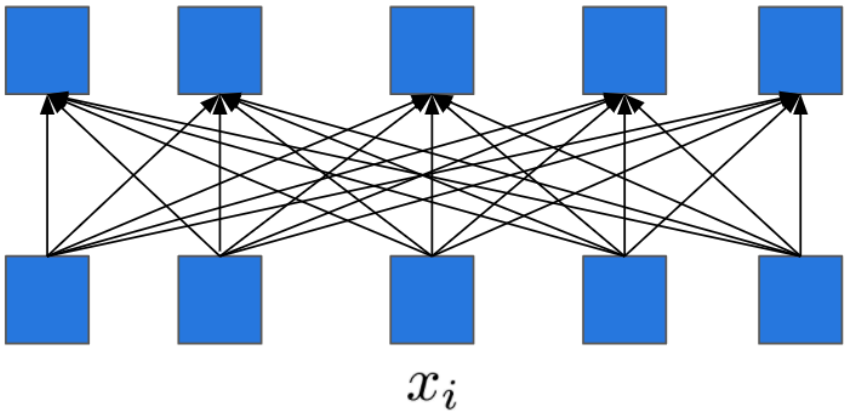
replaces recurrence

$$\alpha_{ki} = \frac{\exp(\text{score}(x_k, x_i))}{\sum_{j=1}^{L} \exp(\text{score}(x_k, x_j))}$$

$$\text{score}(x_k, x_i) = x_k^T x_i / \sqrt{d}, \; d = \dim(x_k)$$

Multi-head attention: apply attention in K feature spaces; concatenate results

$$y_k = \sum_{i=1}^{L} \alpha_{ki} x_i$$



x_i

...

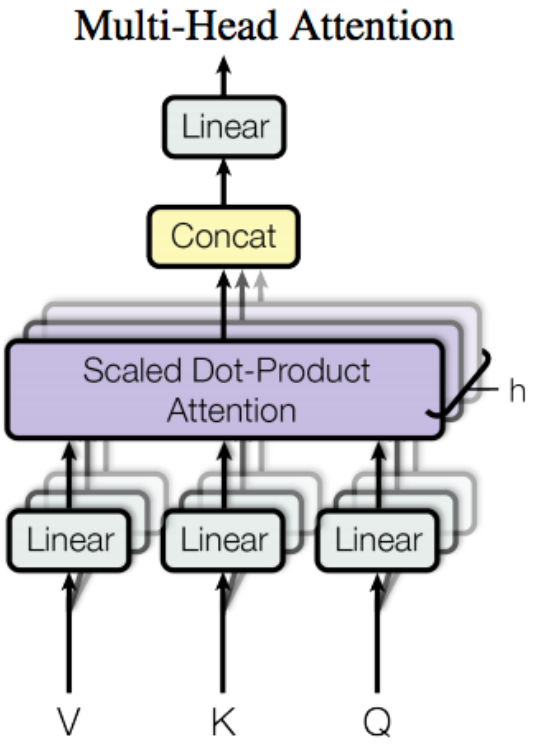
на предпоследней станции встретив товарища

[Vaswani et al. 2017]

Multi-head attention

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

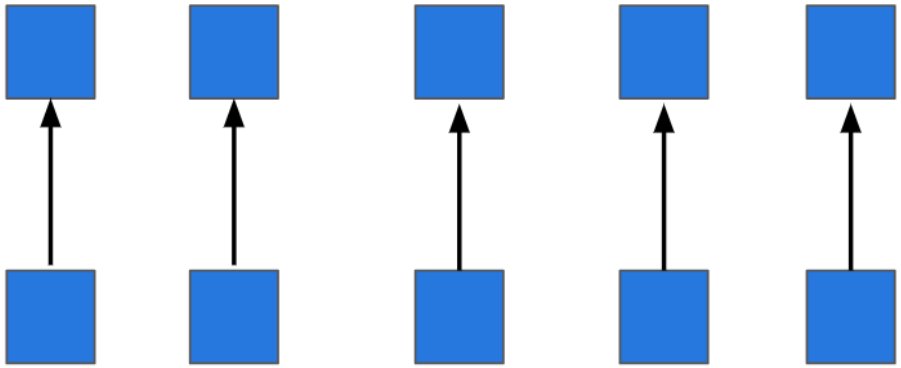
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



[Vaswani et al. 2017]

Per-word feedforward

fully-connected network
same for every word
G(x) = Dense(ReLU(Dense(x)))



...

на предпоследней станции встретив товарища

[Vaswani et al. 2017]

Attention over encoder outputs

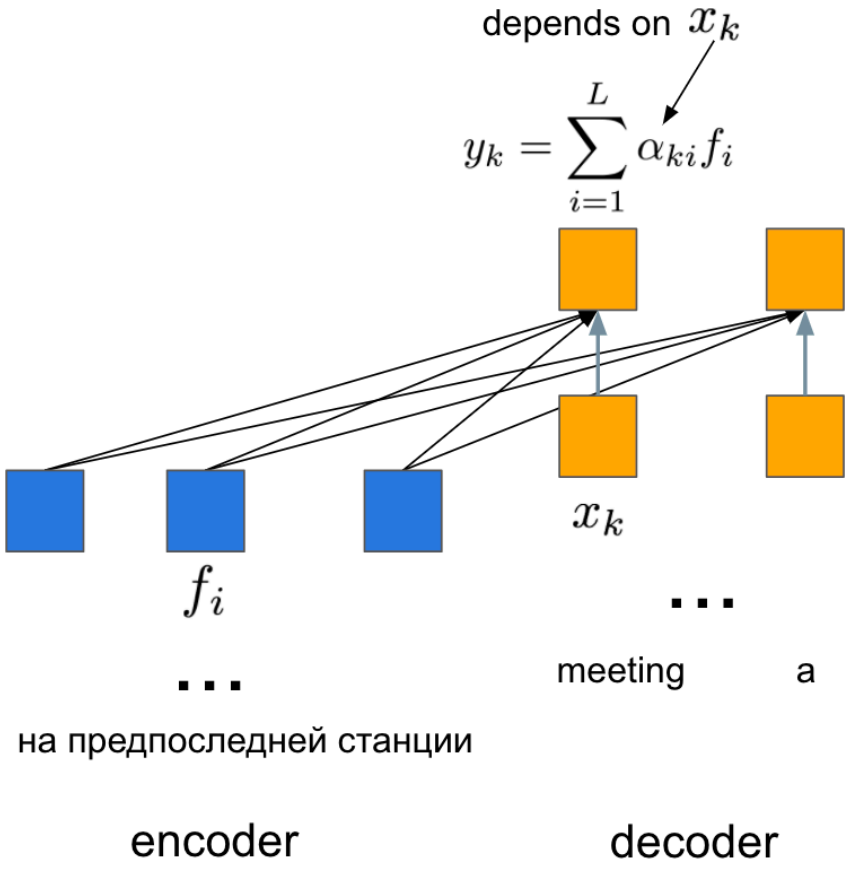
every word attends to features of words from the original sentence

same as soft attention in Neural Machine Translation with LSTM

$$\alpha_{ki} = \frac{\exp(\text{score}(x_k, f_i))}{\sum_{j=1}^{L} \exp(\text{score}(x_k, f_j))}$$

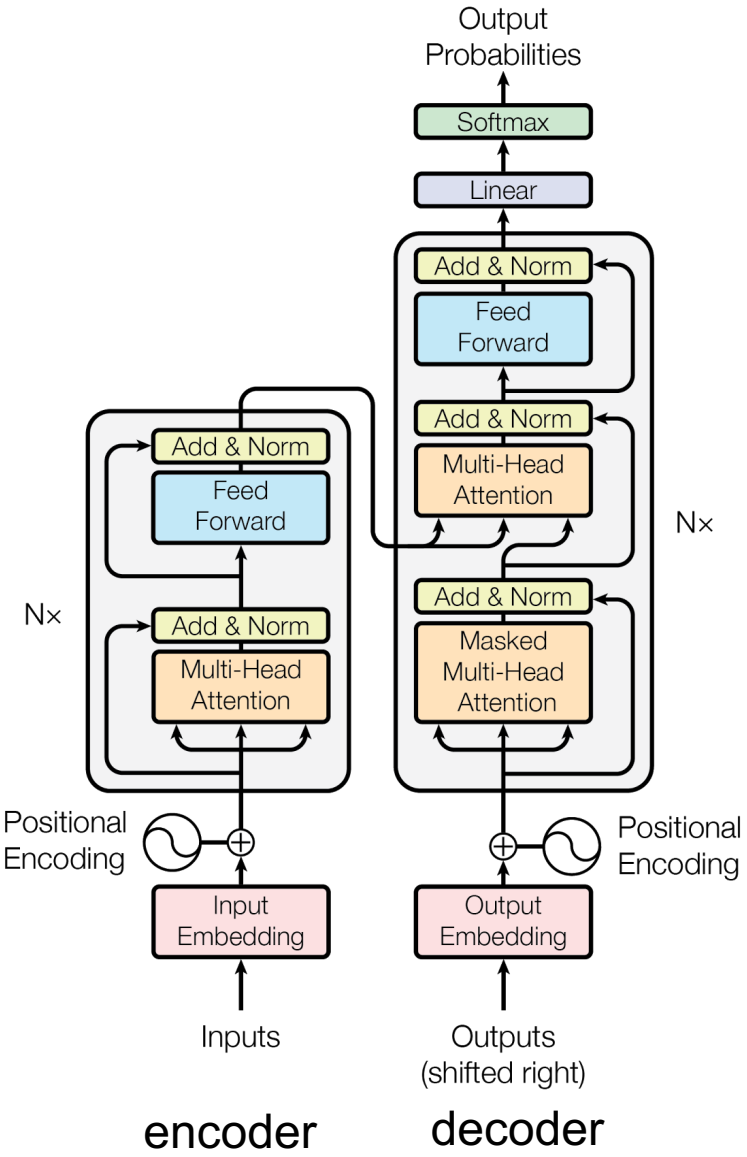
$$\text{score}(x_k, f_i) = x_k^T f_i / \sqrt{d}, \; d = \dim(x_k)$$

Multi-head attention: apply attention in K feature spaces; concatenate results

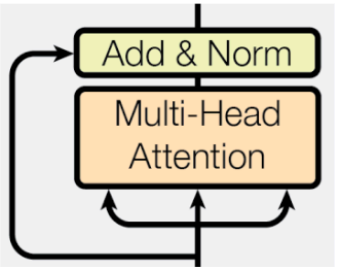


[Vaswani et al. 2017]

Attention Is All You Need

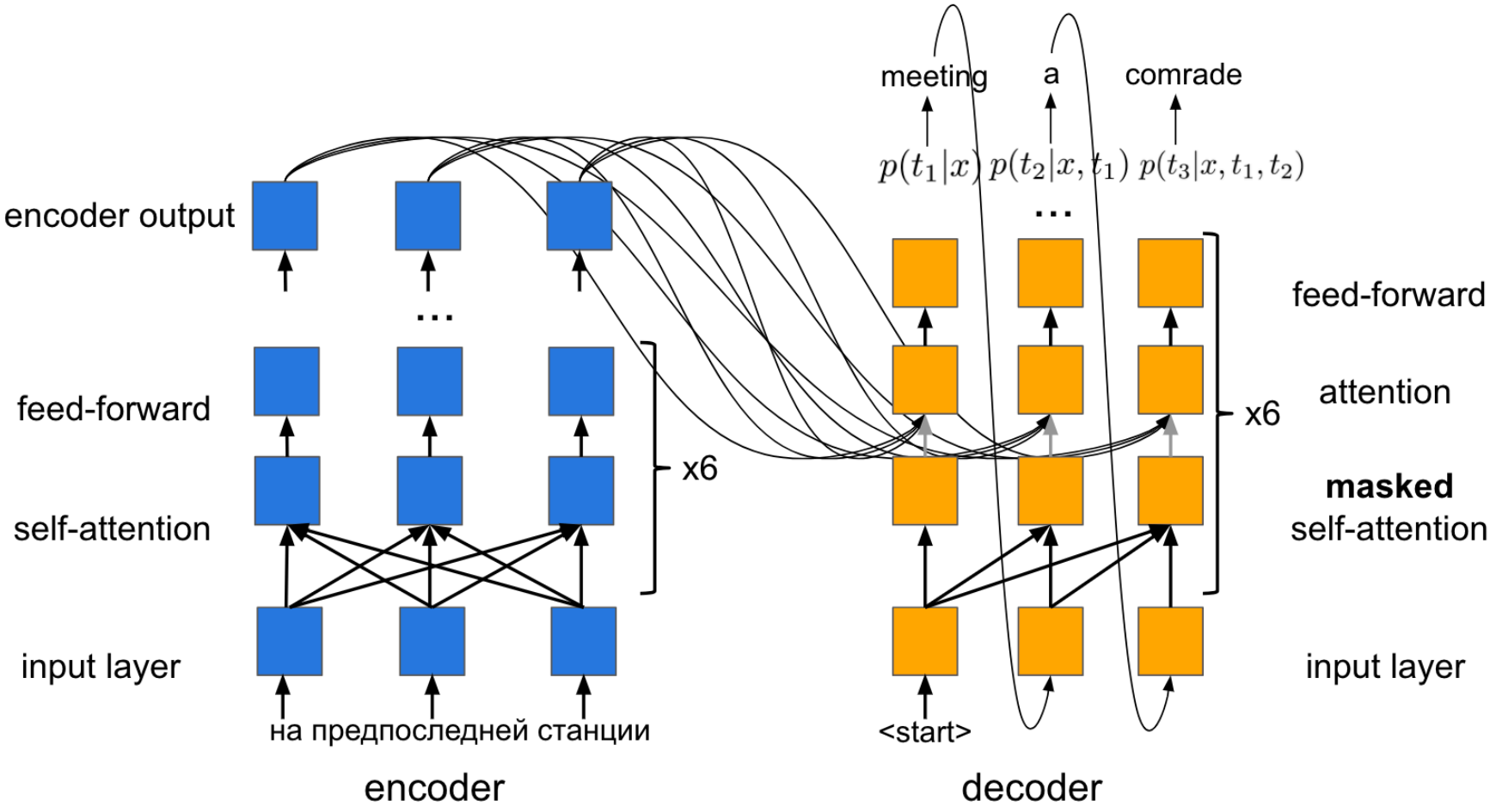


Residual connection + layer normalization



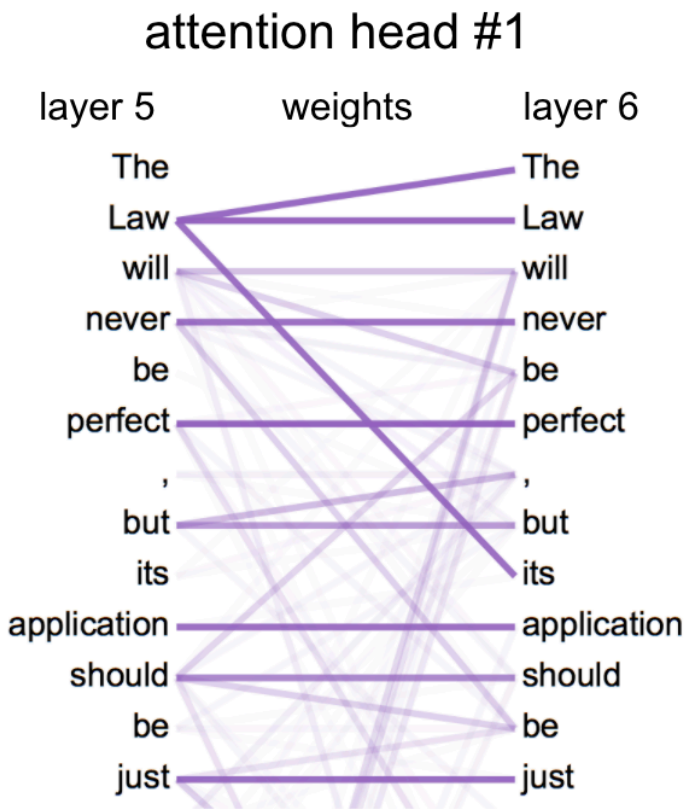
[Vaswani et al. 2017]

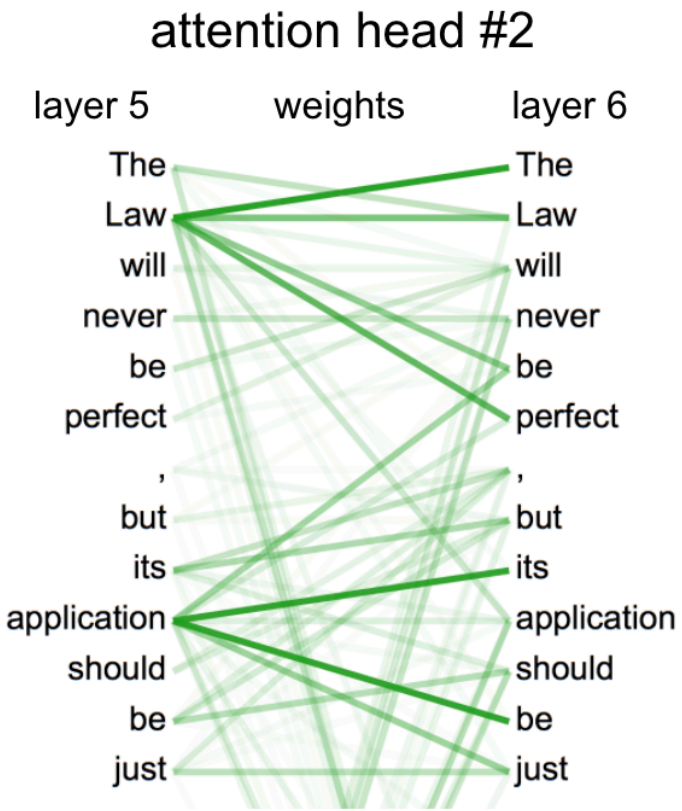
Attention Is All You Need



[Vaswani et al. 2017]

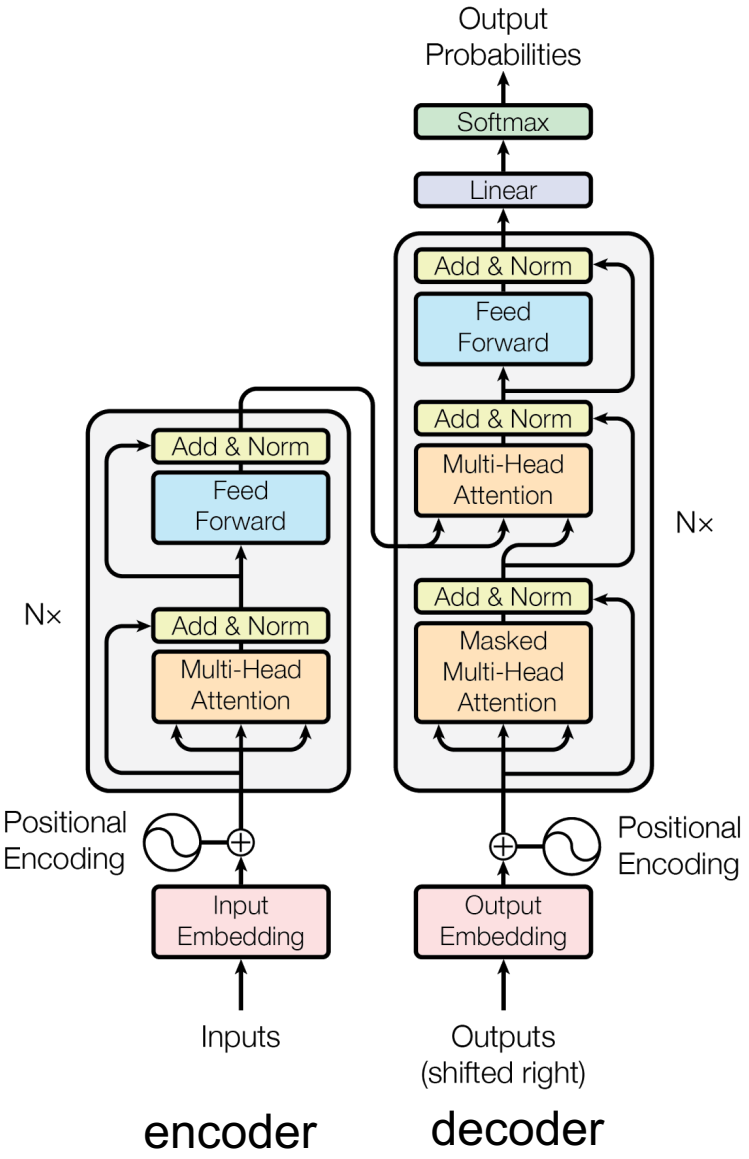
What do the self-attention heads look at?





[Vaswani et al. 2017]

Attention Is All You Need



- All positions are connected
- Maximum Path Length is O(1)
- Parallel computations

[Vaswani et al. 2017]

References

Memory

Presentation by Kira Radinsky

Wojciech Zaremba, Ilya Sutskever. Learning to Execute // arXiv, 2014.

Armand Joulin, Tomas Mikolov. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets // NIPS, 2015.

Alex Graves, Greg Wayne, Ivo Danihelka. Neural Turing Machines // arXiv, 2014.

Alex Graves at al. Hybrid computing using a neural network with dynamic external memory // Nature, 2016.

Attention

Presentation by Michael Figurnov

Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention // ICML 2015

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate // ICLR 2015

Luong et al. Effective approaches to attention-based neural machine translation // EMNLP 2015

Vaswani et al. Attention Is All You Need // 2017