Recurrent nets and LSTM

Nando de Freitas

UNIVERSITY OF OXFORD

Outline of the lecture

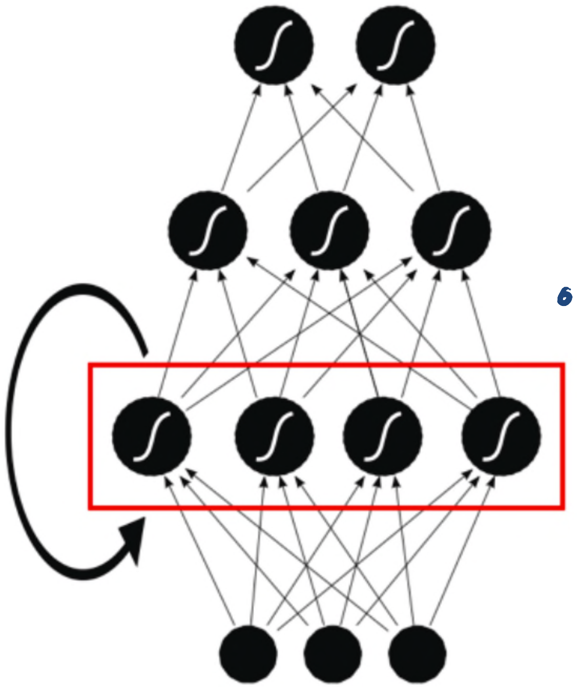
This lecture introduces you sequence models. The goal is for you to learn about:

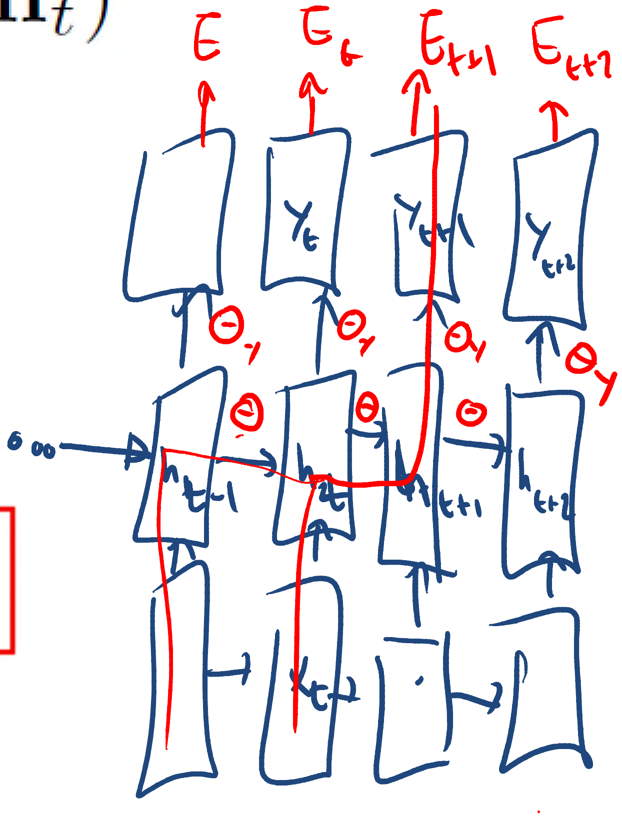
- Recurrent neural networks
- The vanishing and exploding gradients problem
- Long-short term memory (LSTM) networks
- Applications of LSTM networks
    - Language models
    - Translation
    - Caption generation
    - Program execution

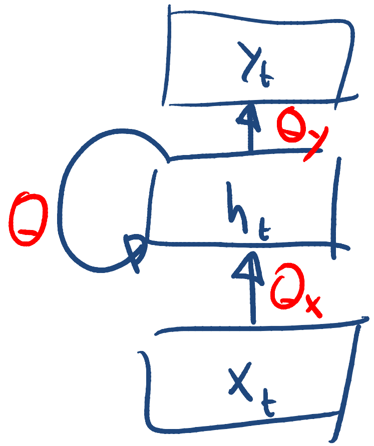
A simple recurrent neural network

$$\mathbf{h}_t = \boldsymbol{\theta}\phi(\mathbf{h}_{t-1}) + \boldsymbol{\theta}_x \mathbf{x}_t$$

$$\mathbf{y}_t = \boldsymbol{\theta}_y \phi(\mathbf{h}_t)$$







[Alex Graves]

Vanishing gradient problem

$$\mathbf{h}_t = \boldsymbol{\theta}\phi(\mathbf{h}_{t-1}) + \boldsymbol{\theta}_x \mathbf{x}_t$$

$$\mathbf{y}_t = \boldsymbol{\theta}_y \phi(\mathbf{h}_t)$$

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{S} \frac{\partial E_t}{\partial \boldsymbol{\theta}}$$

$$\frac{\partial E_t}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \boldsymbol{\theta}}$$

[Yoshua Bengio et al]

Vanishing gradient problem

$$\frac{\partial E_t}{\partial \boldsymbol{\theta}} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial \mathbf{y}_t} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \boldsymbol{\theta}}$$

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{i=k+1}^{t} \boldsymbol{\theta}^T \operatorname{diag}[\boldsymbol{\phi}'(\mathbf{h}_{i-1})]$$

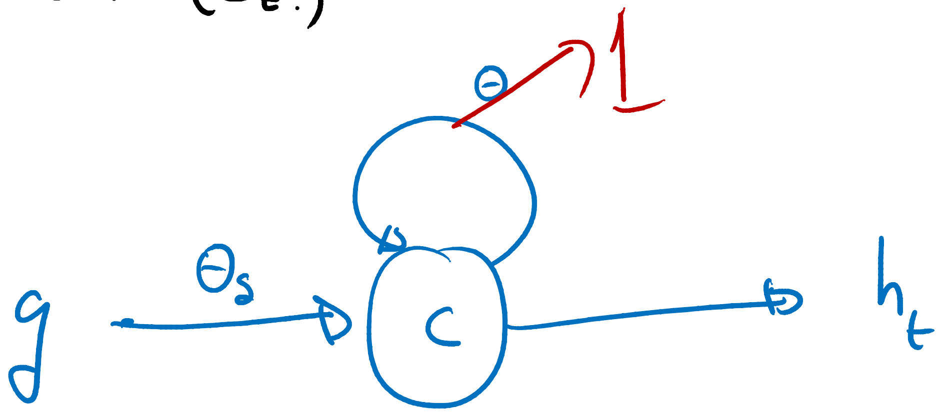
$$\left\| \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \right\| \leq \|\boldsymbol{\theta}^T\| \|\operatorname{diag}[\boldsymbol{\phi}'(\mathbf{h}_{i-1})]\| \leq \gamma_\theta \gamma_\phi$$

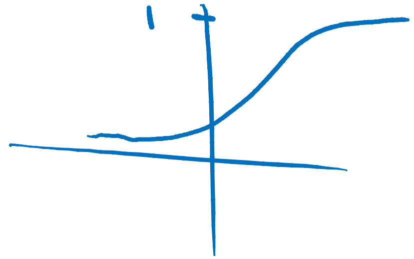
$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \right\| \leq (\gamma_\theta \gamma_\phi)^{t-k}$$

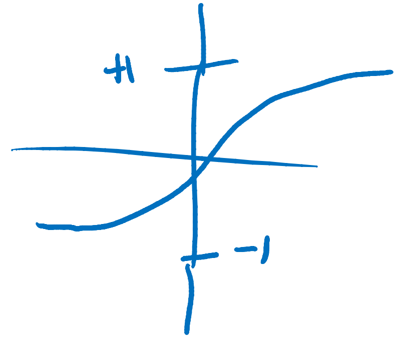
Simple solution

$$c_t = \cancel{\theta}^{\,1}\, c_{t-1} + \theta_g g_t$$

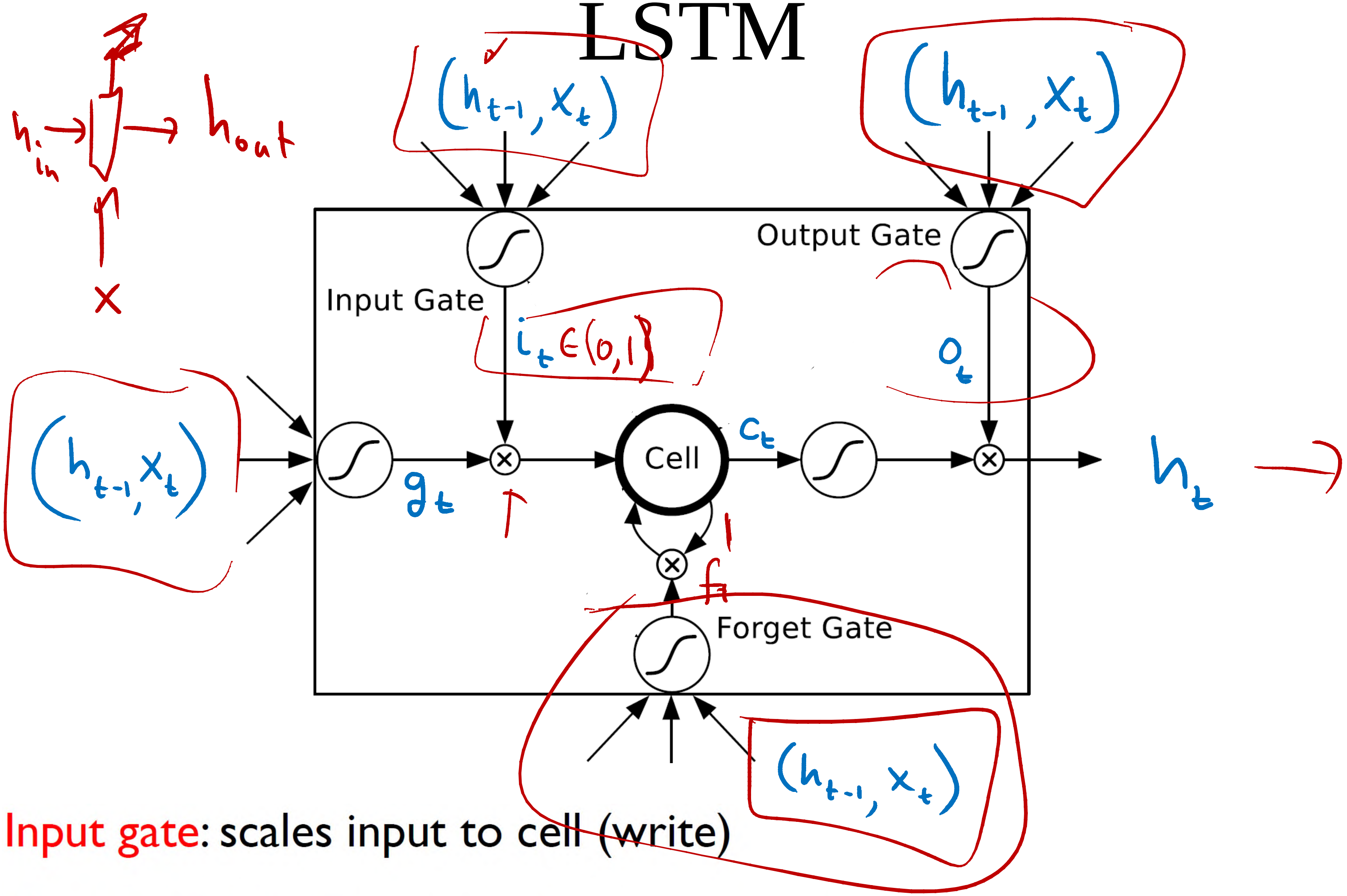
$$h_t = \tanh(c_t)$$







LSTM



Input gate: scales input to cell (write)

Output gate: scales output from cell (read)

Forget gate: scales old cell value (reset)

[Alex Graves]

LSTM

$$\mathbf{i}_t = Sigm(\boldsymbol{\theta}_{xi}\mathbf{x}_t + \boldsymbol{\theta}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = Sigm(\boldsymbol{\theta}_{xf}\mathbf{x}_t + \boldsymbol{\theta}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{o}_t = Sigm(\boldsymbol{\theta}_{xo}\mathbf{x}_t + \boldsymbol{\theta}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\mathbf{g}_t = Tanh(\boldsymbol{\theta}_{xg}\mathbf{x}_t + \boldsymbol{\theta}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} \boxed{+} \mathbf{i}_t \odot \mathbf{g}_t$$

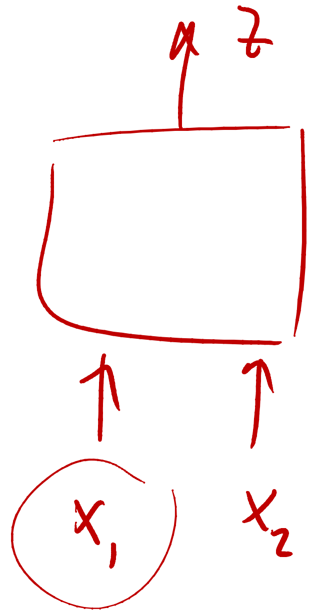
$$\mathbf{h}_t = \mathbf{o}_t \odot Tanh(\mathbf{c}_t)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \odot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 y_1 \\ x_2 y_2 \end{bmatrix}$$

Entry-wise multiplication layer

$$\mathbf{z} = f(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \odot \mathbf{x}_2$$

$$\frac{\partial E}{\partial \mathbf{x}_1} = \frac{\partial E}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}_1} = \frac{\partial E}{\partial \mathbf{z}} \odot \mathbf{x}_2$$

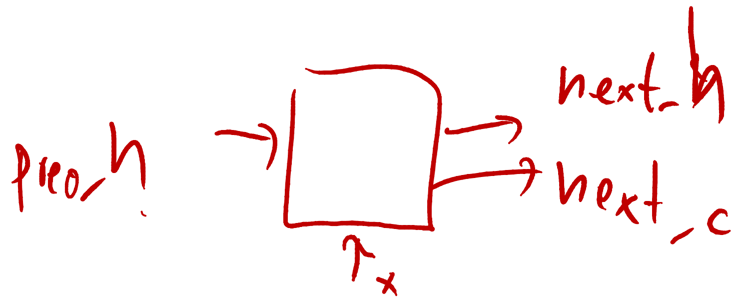


$$z_i = f(x_{1i}, x_{2i}) = x_{1i} x_{2i}$$

$$\frac{\partial E}{\partial x_{1i}} = \sum_j \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial x_{1i}} = \frac{\partial E}{\partial z_i} x_{2i}$$

LSTM cell in Torch

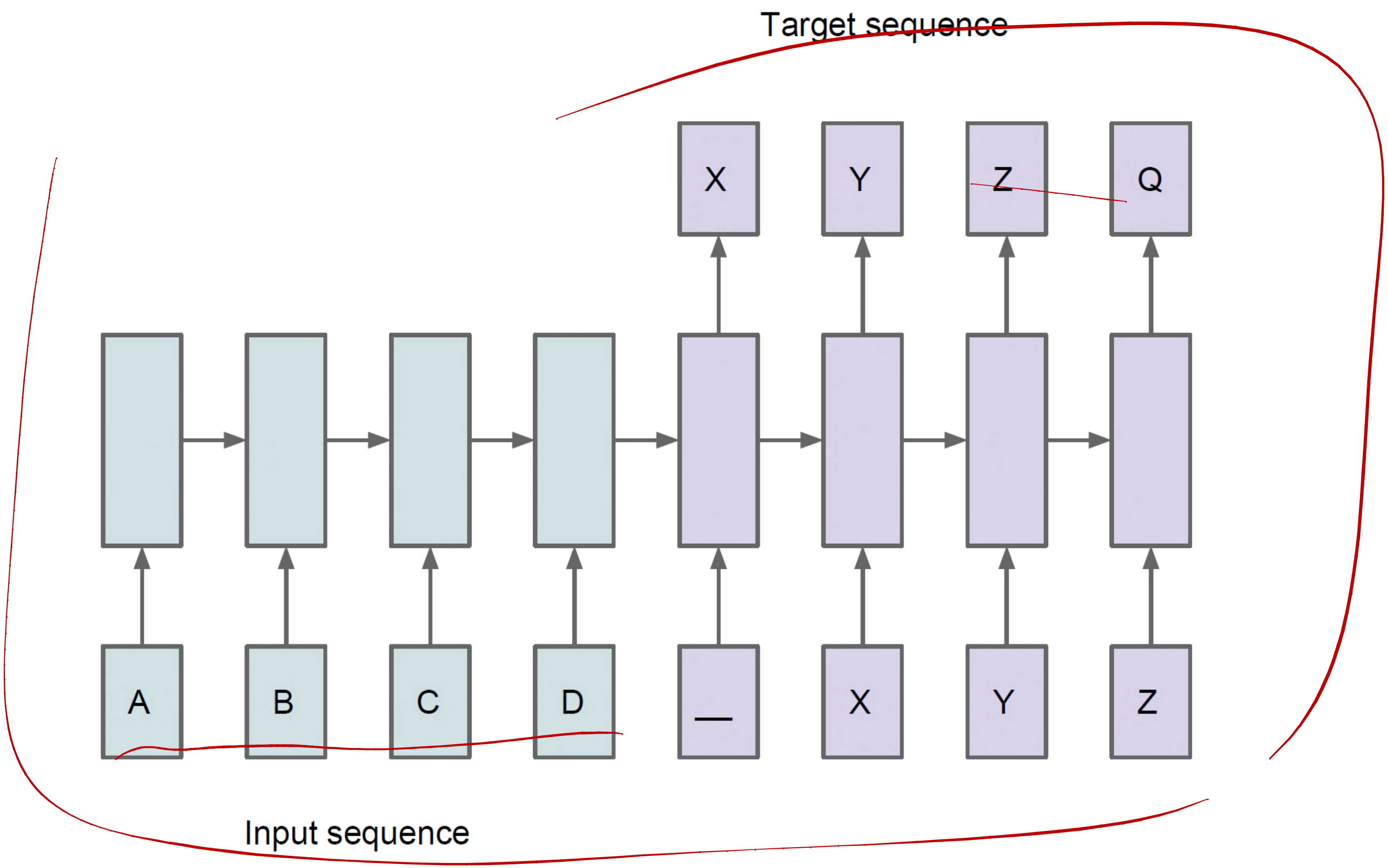
```
local function make_lstm_step(opt, input, prev_h, prev_c)
    local function new_input_sum()
        local x_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
        local h_to_h = nn.Linear(opt.rnn_size, opt.rnn_size)
        return nn.CAddTable()({ x_to_h(input), h_to_h(prev_h)})
    end
    local in_gate = nn.Sigmoid()(new_input_sum())
    local forget_gate = nn.Sigmoid()(new_input_sum())
    local cell_gate = nn.Tanh()(new_input_sum())
    local next_c = nn.CAddTable()({
        nn.CMulTable()({forget_gate, prev_c}),
        nn.CMulTable()({in_gate, cell_gate})})
    local out_gate = nn.Sigmoid()(new_input_sum())
    local next_h = nn.CMulTable()({out_gate, nn.Tanh()(next_c)})
    return next_h, next_c
end
```



LSTM column in Torch

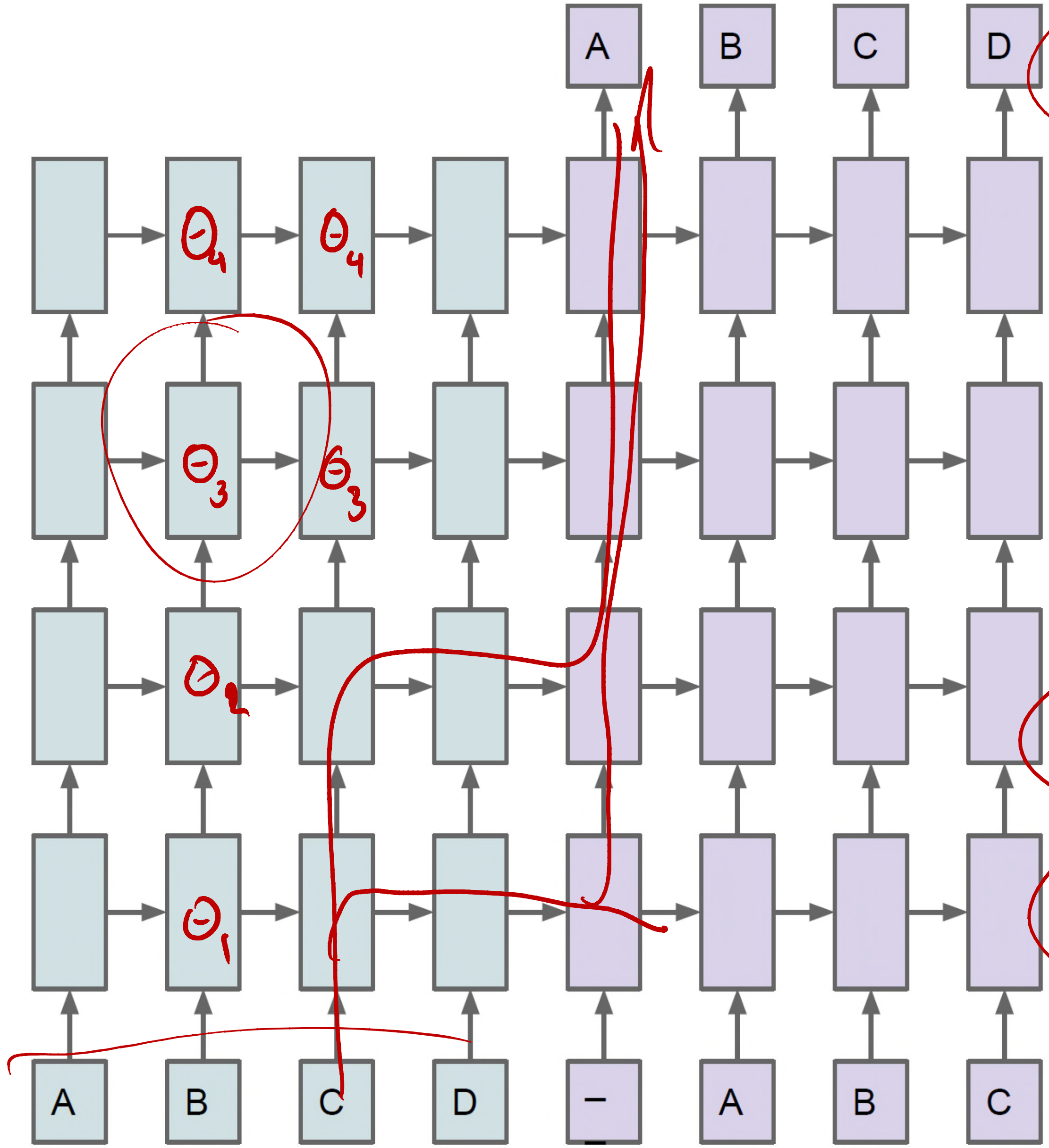
```lua
local function make_lstm_network(opt)
    local n_layers = opt.n_layers or 1

    local x = nn.Identity()()
    local prev_s = nn.Identity()()
    local splitted_s = {prev_s:split(2 * n_layers)}
    local next_s = {}
    local inputs = {[0] = x}
    for i = 1, n_layers do
        local prev_h = splitted_s[2 * i - 1]
        local prev_c = splitted_s[2 * i]
        local next_h, next_c = make_lstm_step(opt, inputs[i - 1], prev_h, prev_c)
        next_s[#next_s + 1] = next_h
        next_s[#next_s + 1] = next_c
        inputs[i] = next_h
    end
    local module = nn.gModule({x, prev_s}, {inputs[n_layers], nn.Identity()(next_s)})
    module:getParameters():uniform(-0.08, 0.08)
    module = cuda(module)
    return module
end
```

LSTMs for sequence to sequence prediction

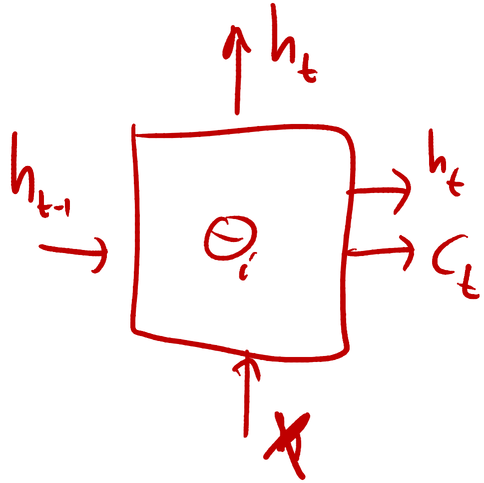


[Ilya Sutskever et al]

LSTMs for sequence to sequence prediction



80k softmax by
1000 dims
This is very big!



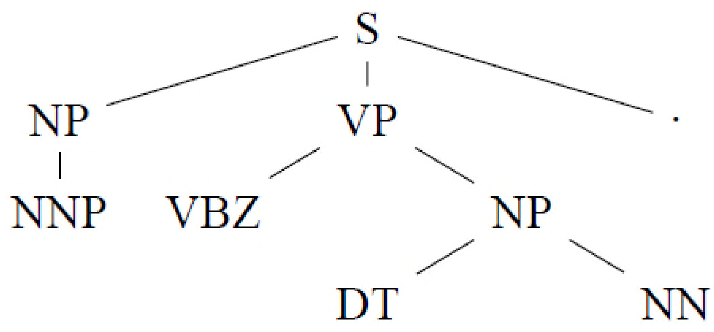
1000 LSTM cells
2000 dims per
timestep

2000 x 4 =
8k dims per
sentence

160k vocab in
input language

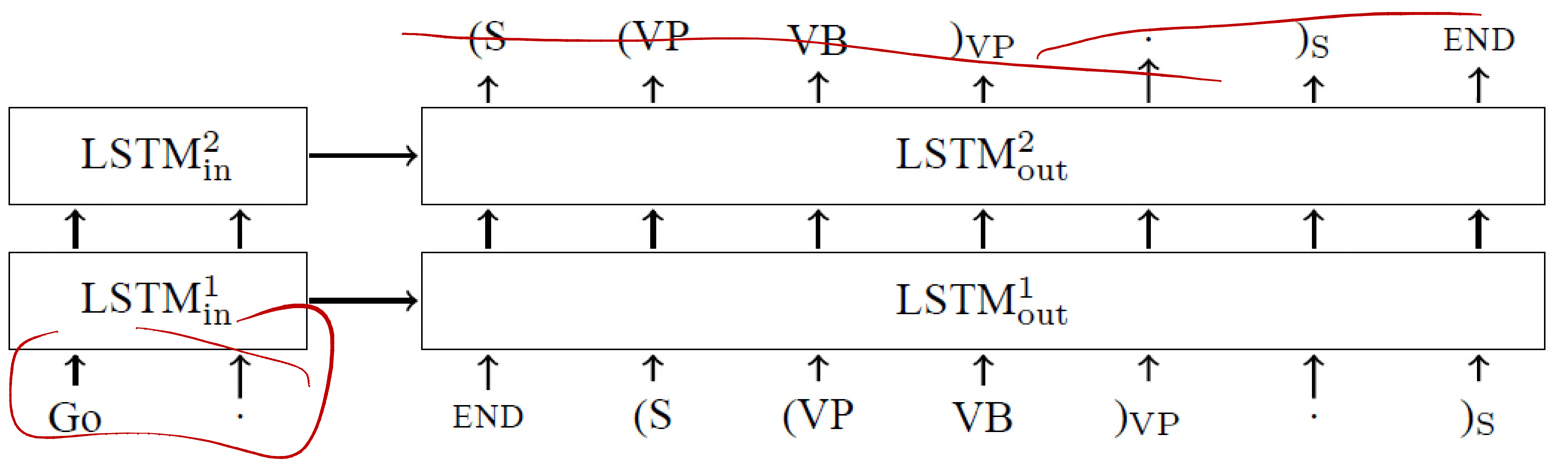
Learning to parse

John has a dog . $\rightarrow$



John has a dog . $\rightarrow$ (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$

John has a dog . $\rightarrow$ (S (NP NNP $)_{NP}$ $\perp$ (VP VBZ $\perp$ (NP DT $\perp$ NN $)_{NP}$ $)_{VP}$ $\perp$. $)_{S}$ $\perp$



[Oriol Vinyals et al]

Learning to execute

Input:

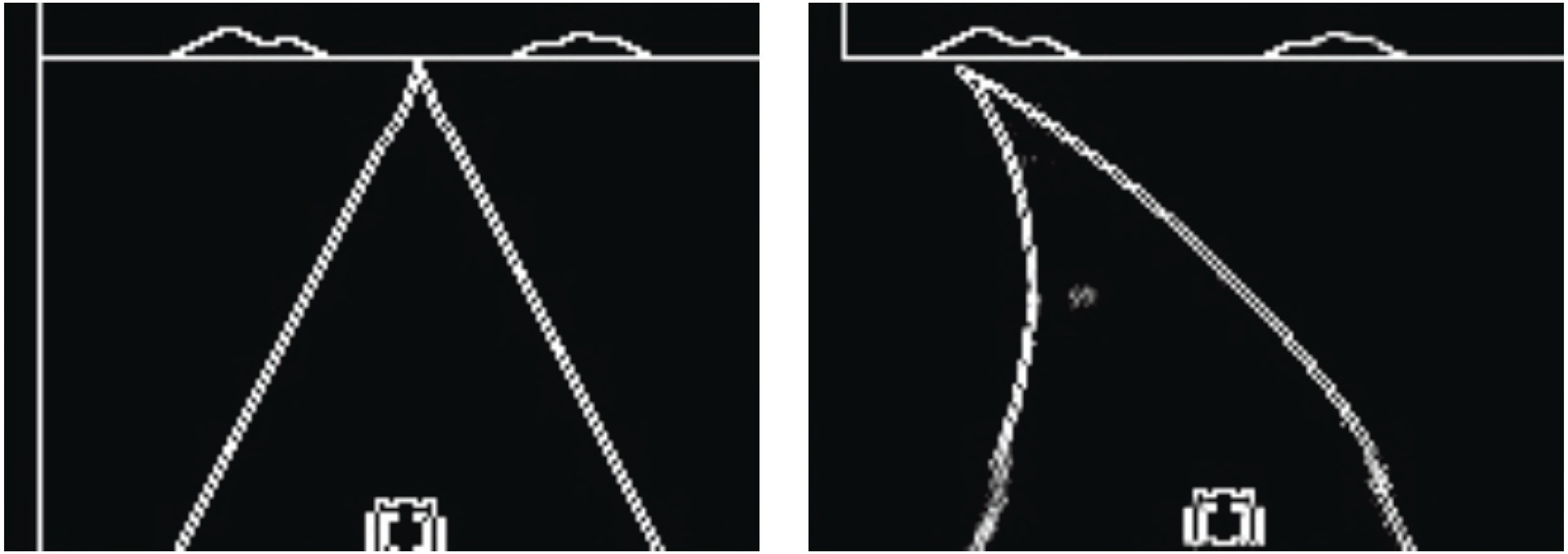
```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

[Wojciech Zaremba and Ilya Sutskever]

Video prediction

Real Generated

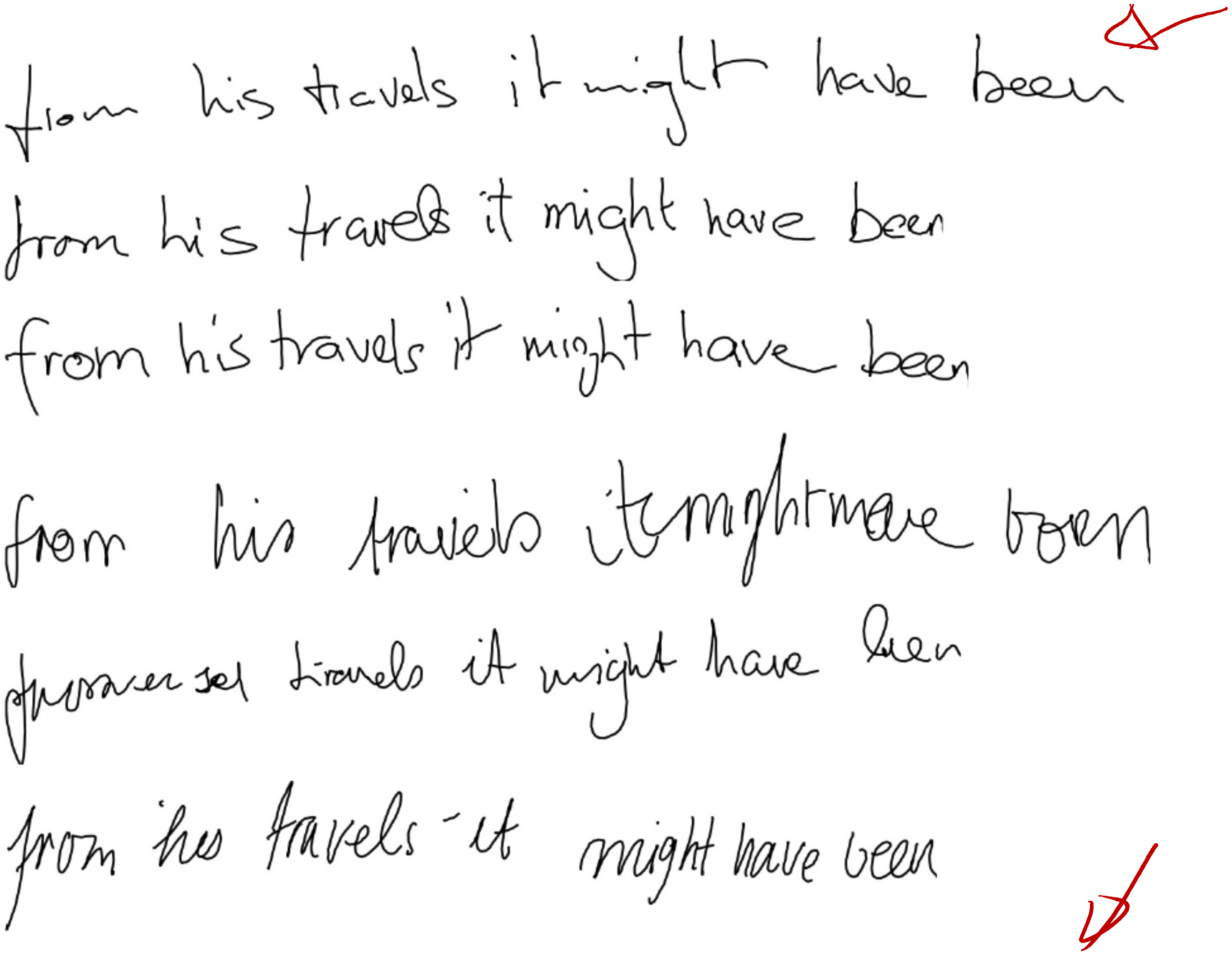


Karol Gregor, Ivo Danihelka, Andriy Mnih, Daan Wierstra…

Google DeepMind

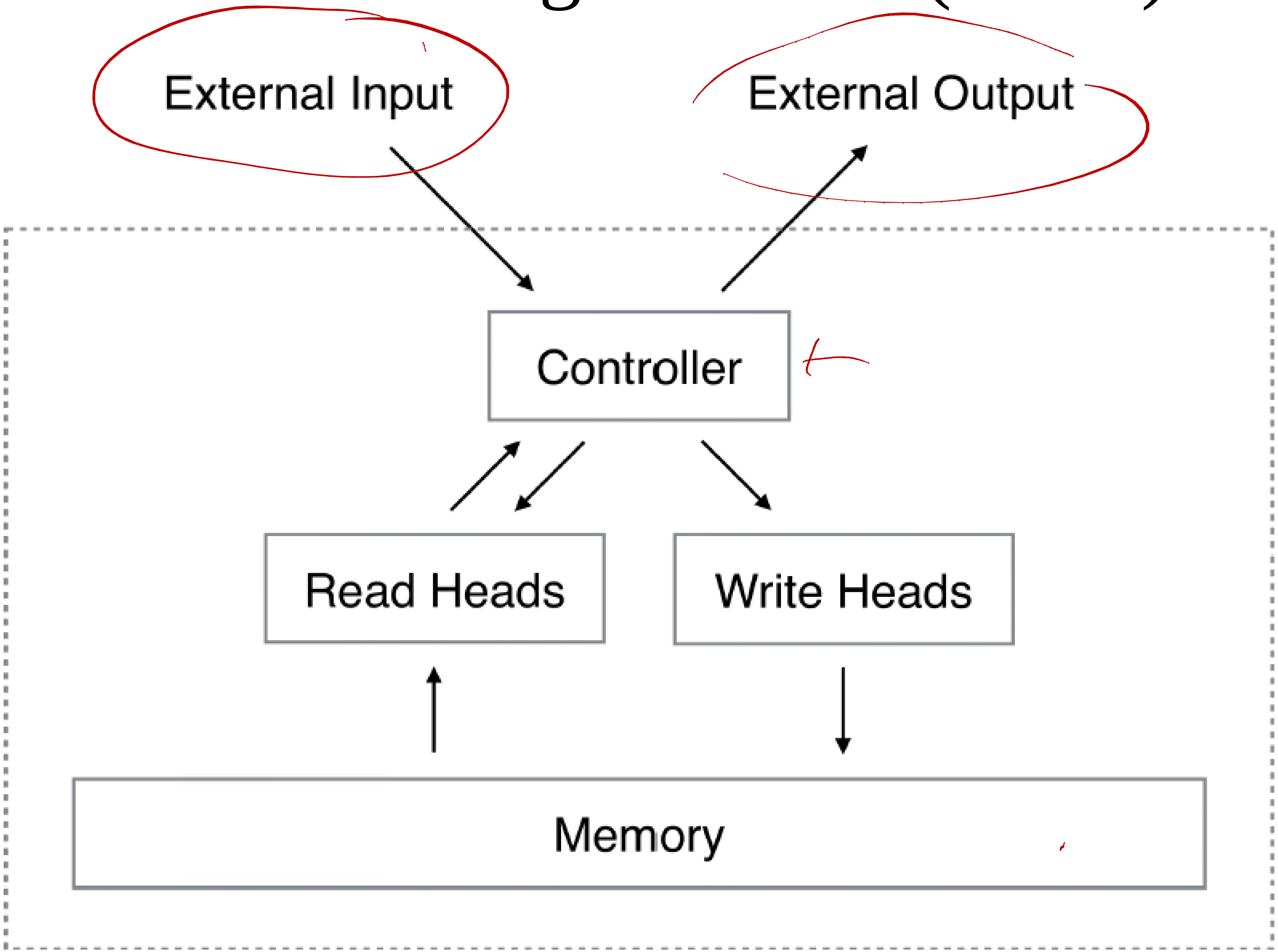
Hand-writing recognition and synthesis

Which is Real?



[Alex Graves]

Neural Turing Machine (NTM)



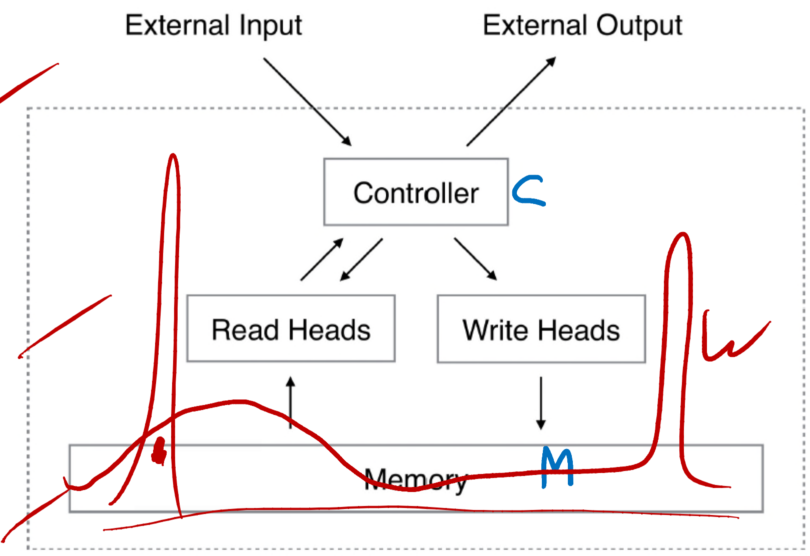
[Alex Graves, Greg Wayne, Ivo Danihelka]

Neural Turing Machine (NTM)

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

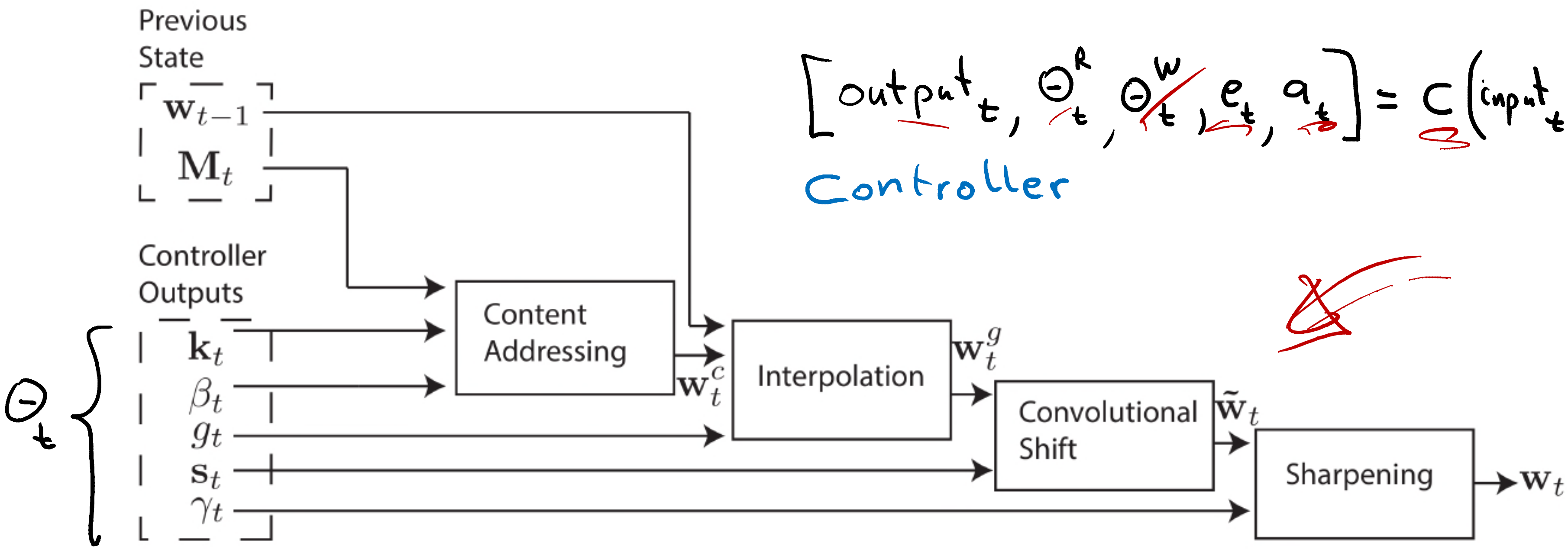
$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t^{W}(i)\mathbf{e}_t\right]$$ Erase

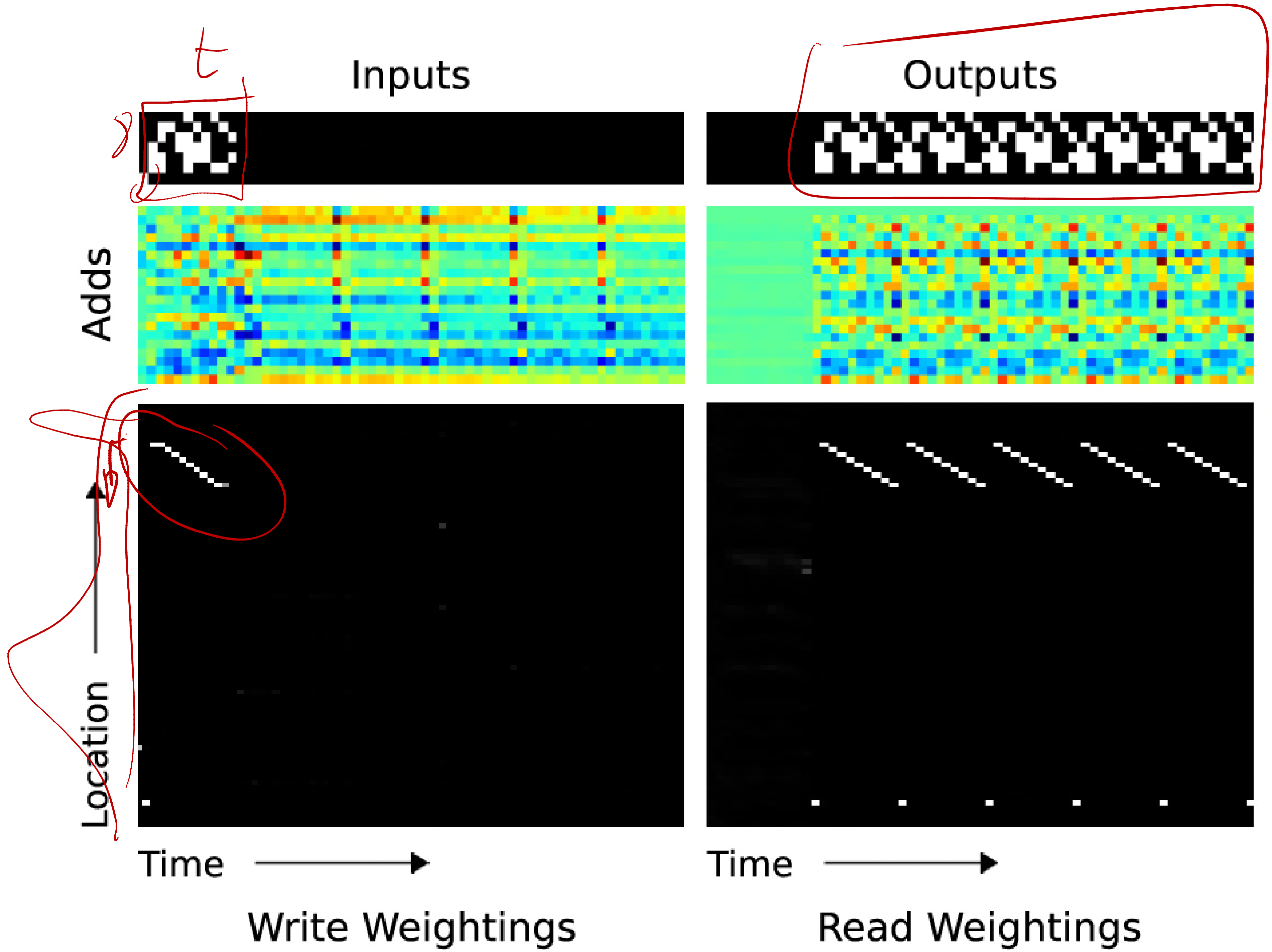
$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t^{W}(i)\,\mathbf{a}_t$$ Write



$$\left[\mathrm{output}_t,\ \Theta_t^{R},\ \Theta_t^{W},\ e_t,\ a_t\right] = C\left(\mathrm{input}_t,\ r_t\right)$$

Controller





t
Inputs
Outputs
Adds
Location
Time
Time
Write Weightings
Read Weightings

Translation with alignment (Bahdanau et al)

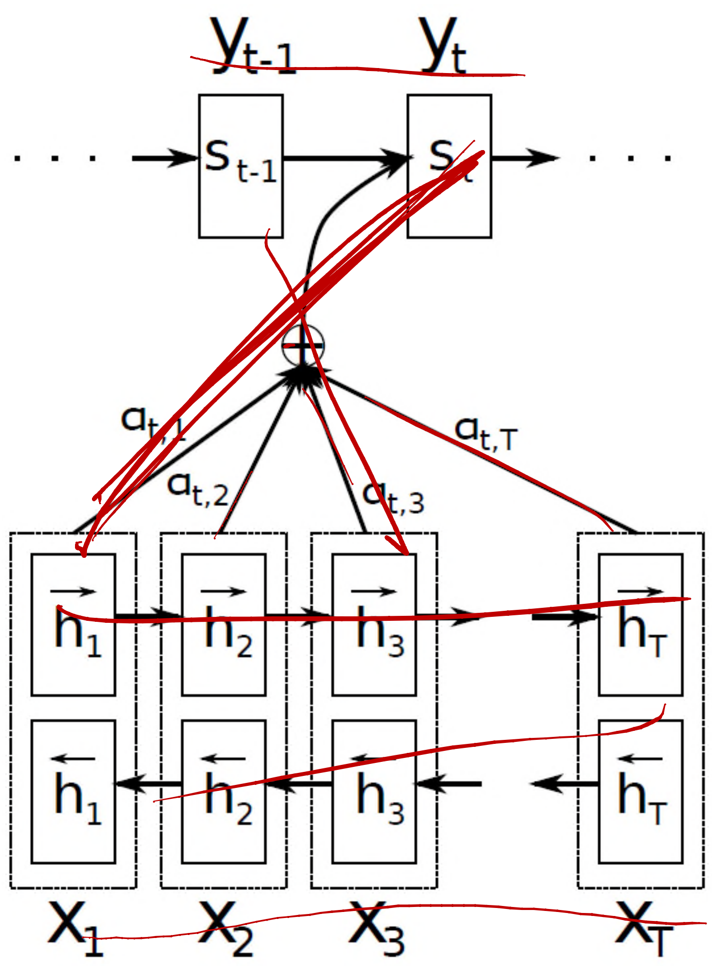
$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

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

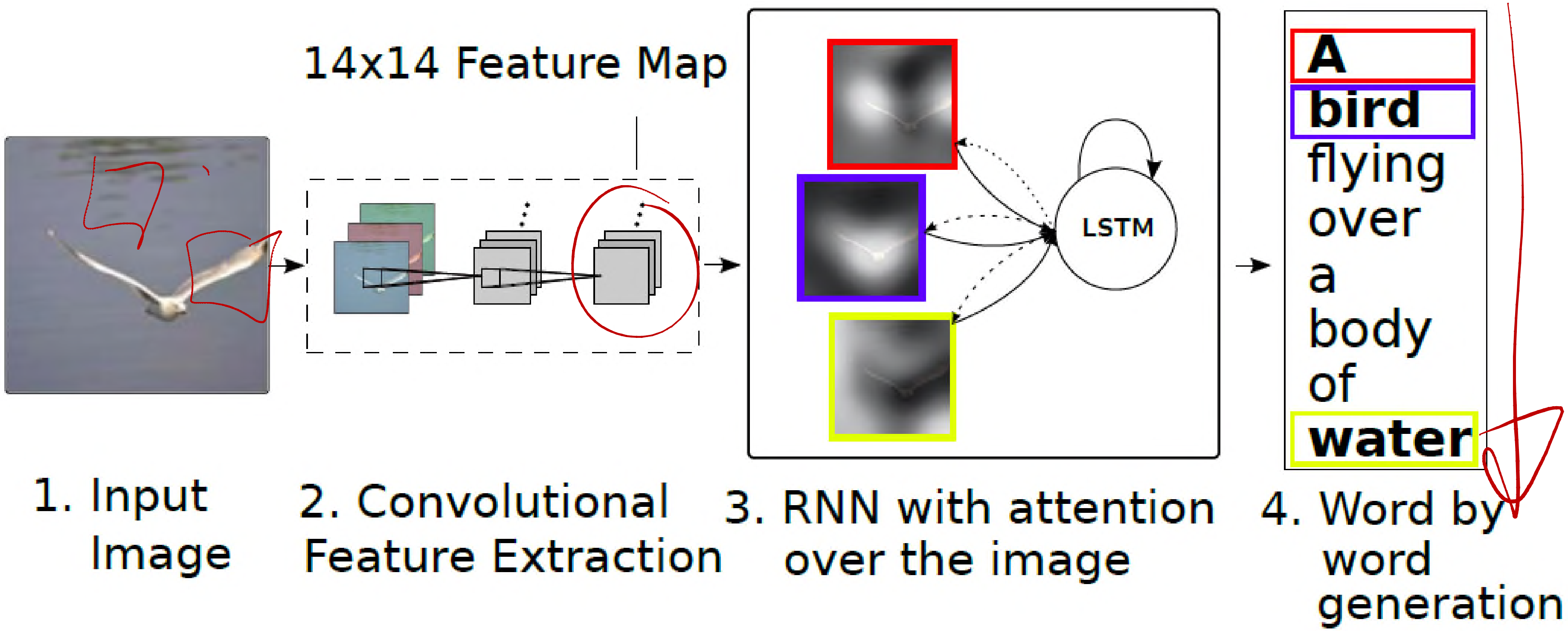
context vector $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$

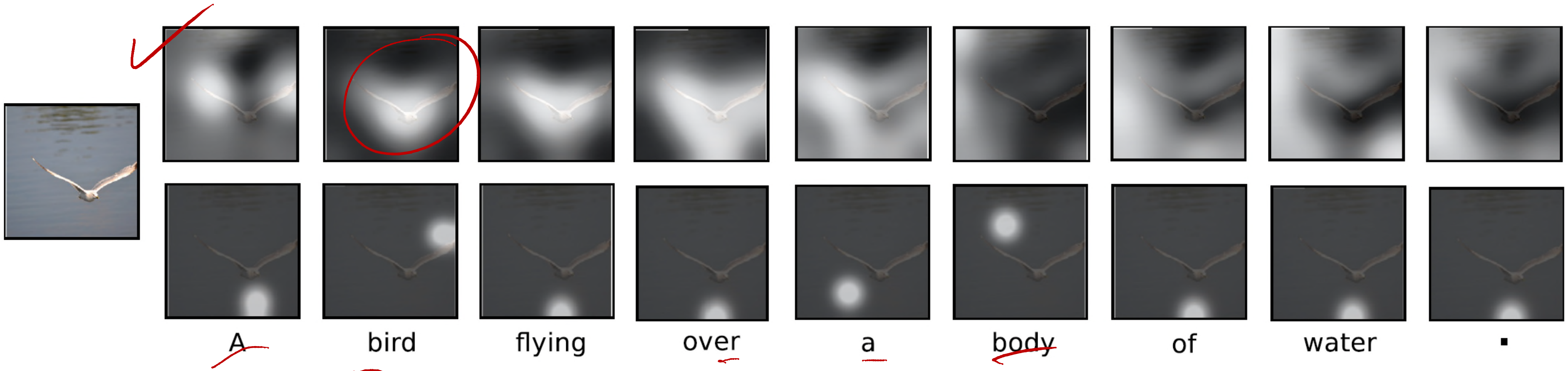
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

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



Show, attend and tell





[Kelvin Xu et al, 2015]

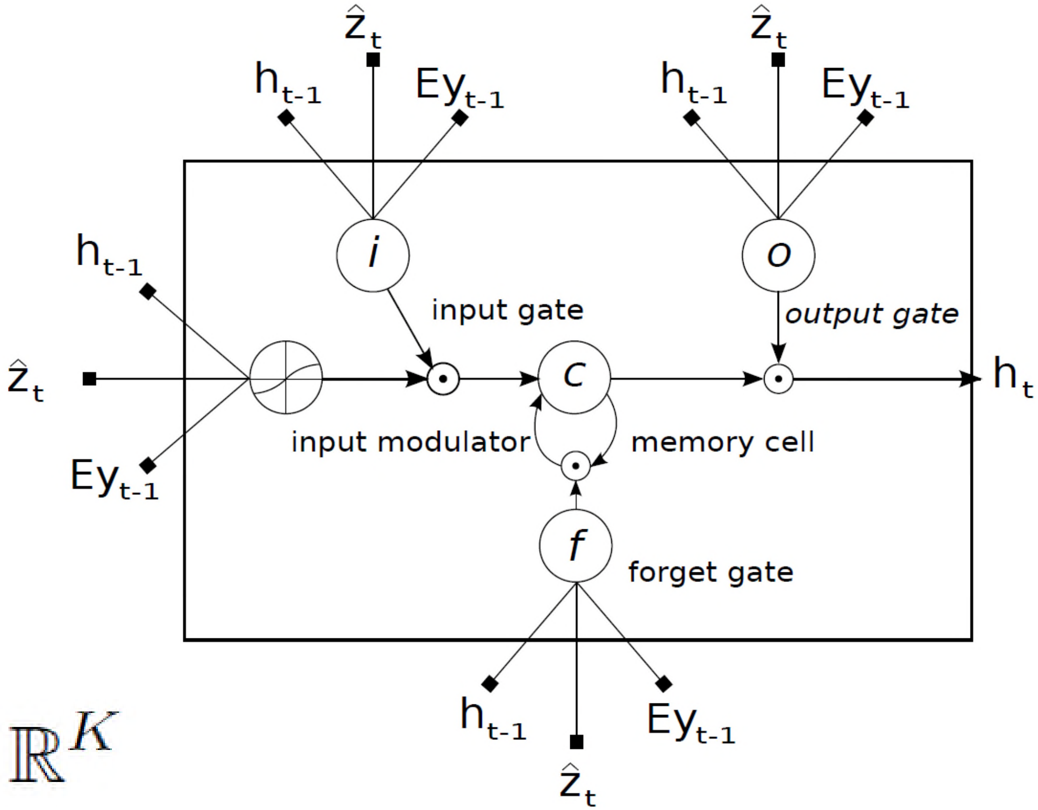
Show, attend and tell

$$a = \{\mathbf{a}_1, \ldots, \mathbf{a}_L\}, \ \mathbf{a}_i \in \mathbb{R}^D$$

$$\hat{\mathbf{z}}_t = \phi\left(\{\mathbf{a}_i\}, \{\alpha_i\}\right) = \sum_i^L \alpha_i \mathbf{a}_i$$

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^L \exp(e_{tk})}$$



$$y = \{\mathbf{y}_1, \ldots, \mathbf{y}_C\}, \ \mathbf{y}_i \in \mathbb{R}^K$$

Next lecture

In the next lecture, we will look techniques for unsupervised learning known as autoencoders. We will also learn about sampling and variational methods.

I **strongly recommend** reading Kevin Murphy's variational inference book chapter prior to the lecture.