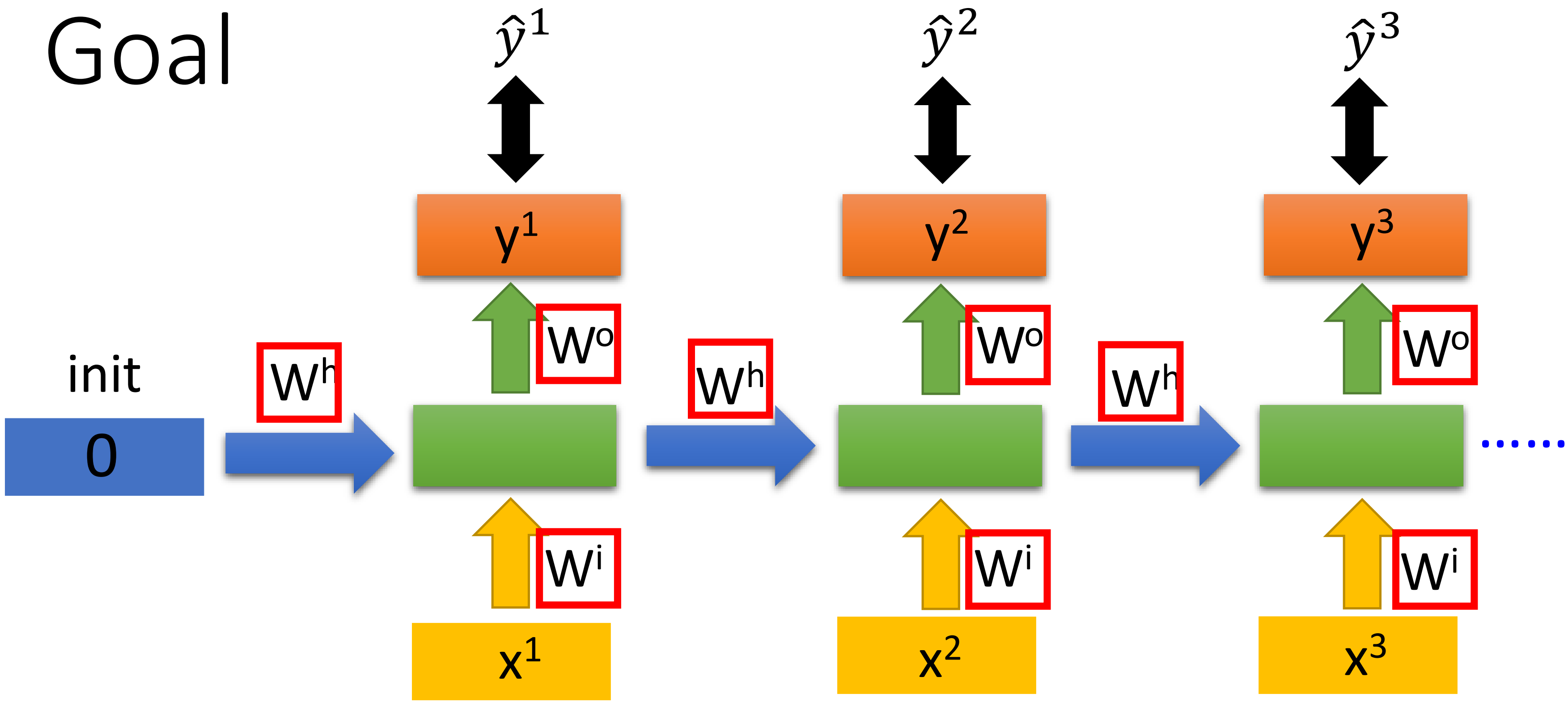
Training Recurrent Neural Network

Hung-yi Lee

Goal



$$C = \frac{1}{2}\sum_{n=1}^{N}\|y^n - \hat{y}^n\|^2$$

$$C^n = \|y^n - \hat{y}^n\|^2$$

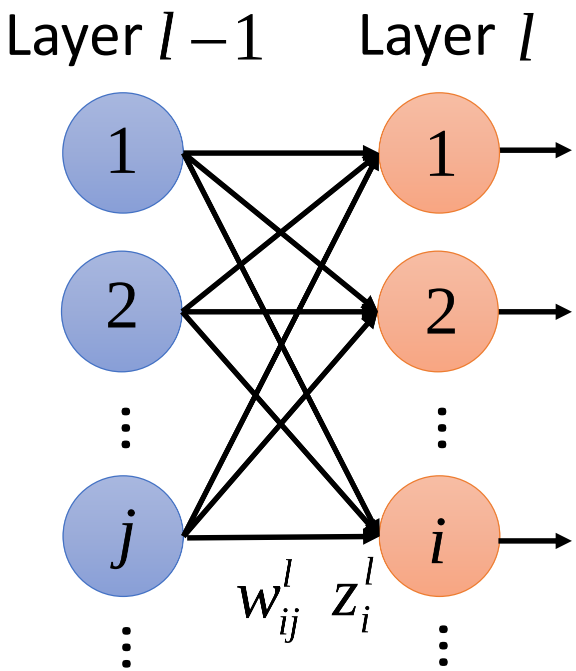
All element w in W^h, W^i or W^o

$\Rightarrow$ $w \leftarrow w - \eta \partial C^n / \partial w$

Backpropagation through time (BPTT)

Review: Backpropagation

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}$$



$$\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}$$

Forward Pass

$$z^1 = W^1 x + b^1$$

$$a^1 = \sigma(z^1)$$

……

$$z^{l-1} = W^{l-1} a^{l-2} + b^{l-1}$$

$$a^{l-1} = \sigma(z^{l-1})$$

Error signal

$$\delta_i^l$$

Backward Pass

$$\delta^L = \sigma'(z^L) \bullet \nabla C_x(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

……

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

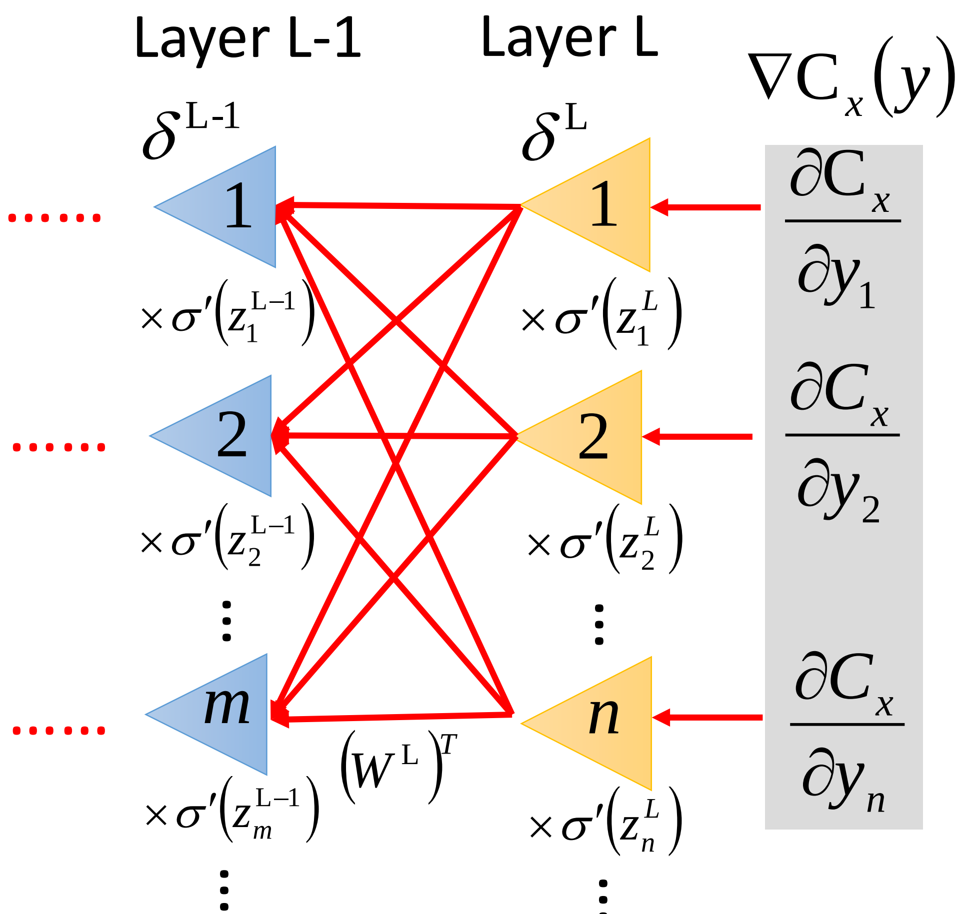
……

Review: Backpropagation

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial z_i^l}{\partial w_{ij}^l} \frac{\partial C_x}{\partial z_i^l}$$

Error signal

$$\delta_i^l$$



Backward Pass

$$\delta^L = \sigma'(z^L) \bullet \nabla C_x(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \bullet (W^L)^T \delta^L$$

……

$$\delta^l = \sigma'(z^l) \bullet (W^{l+1})^T \delta^{l+1}$$

……

Backpropagation through Time

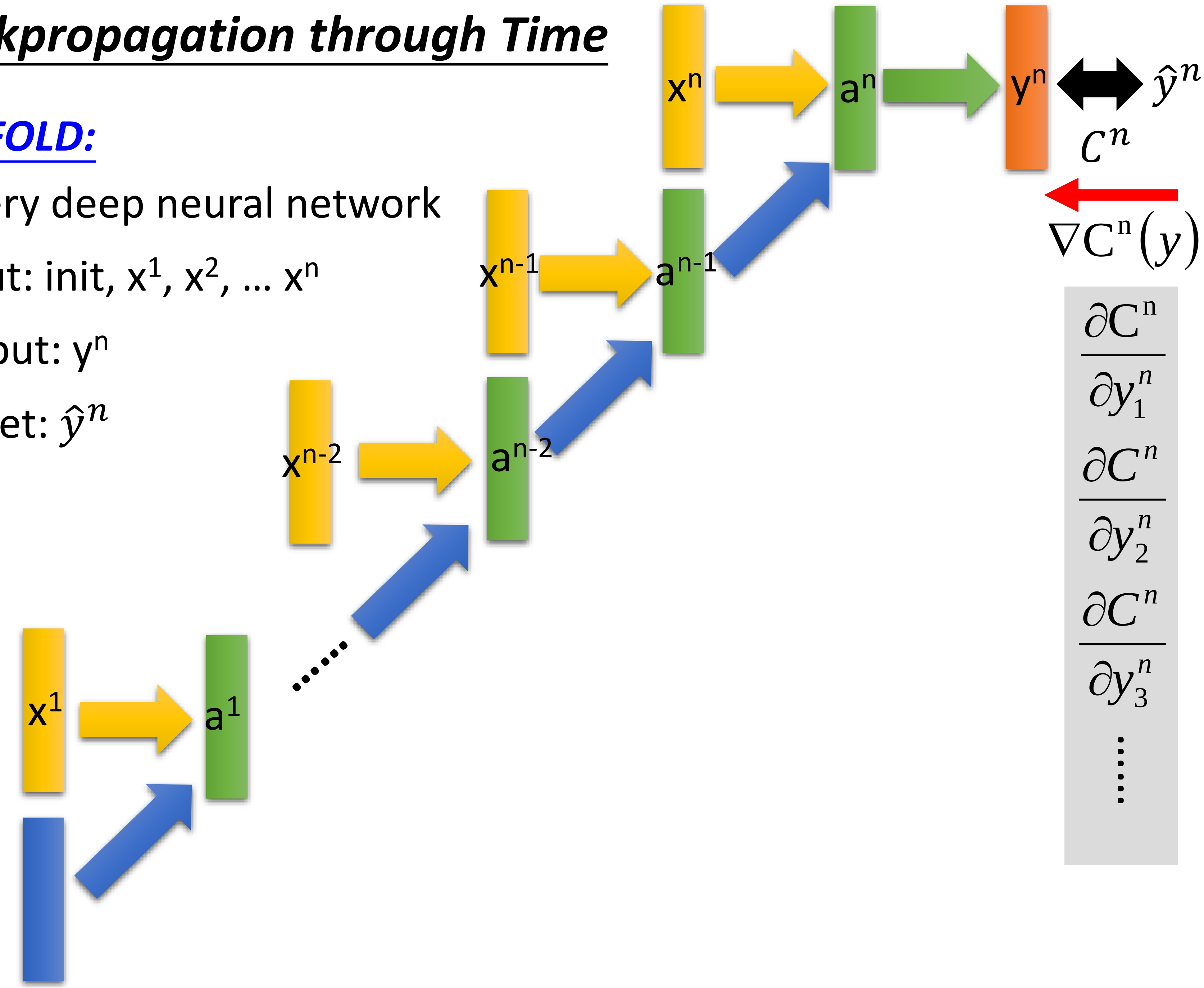
UNFOLD:

A very deep neural network

Input: init, x^1, x^2, … x^n

output: y^n

target: $\hat{y}^n$



Backpropagation through Time

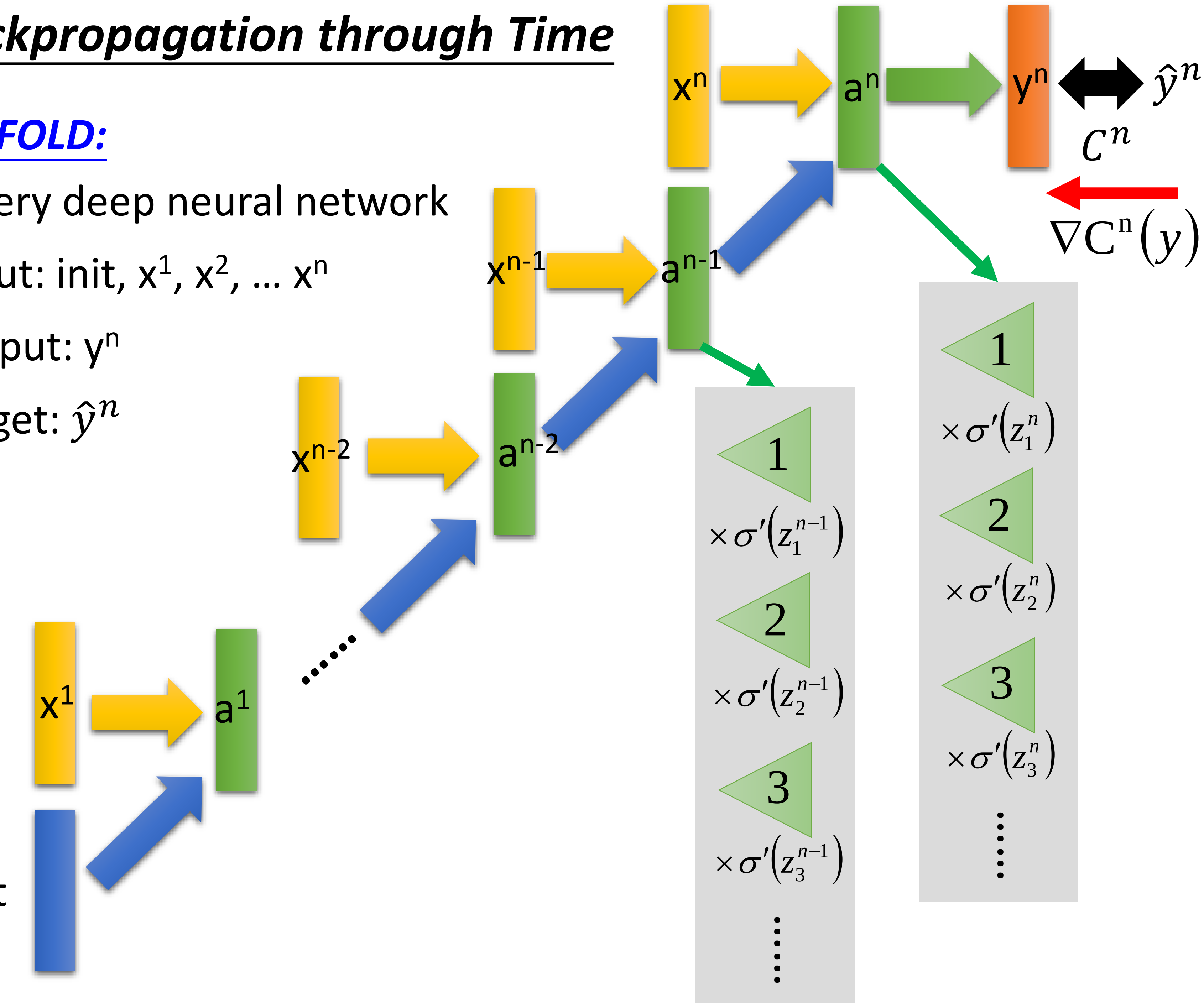
UNFOLD:

A very deep neural network

Input: init, x^1, x^2, … x^n

output: y^n

target: $\hat{y}^n$



Backpropagation through Time

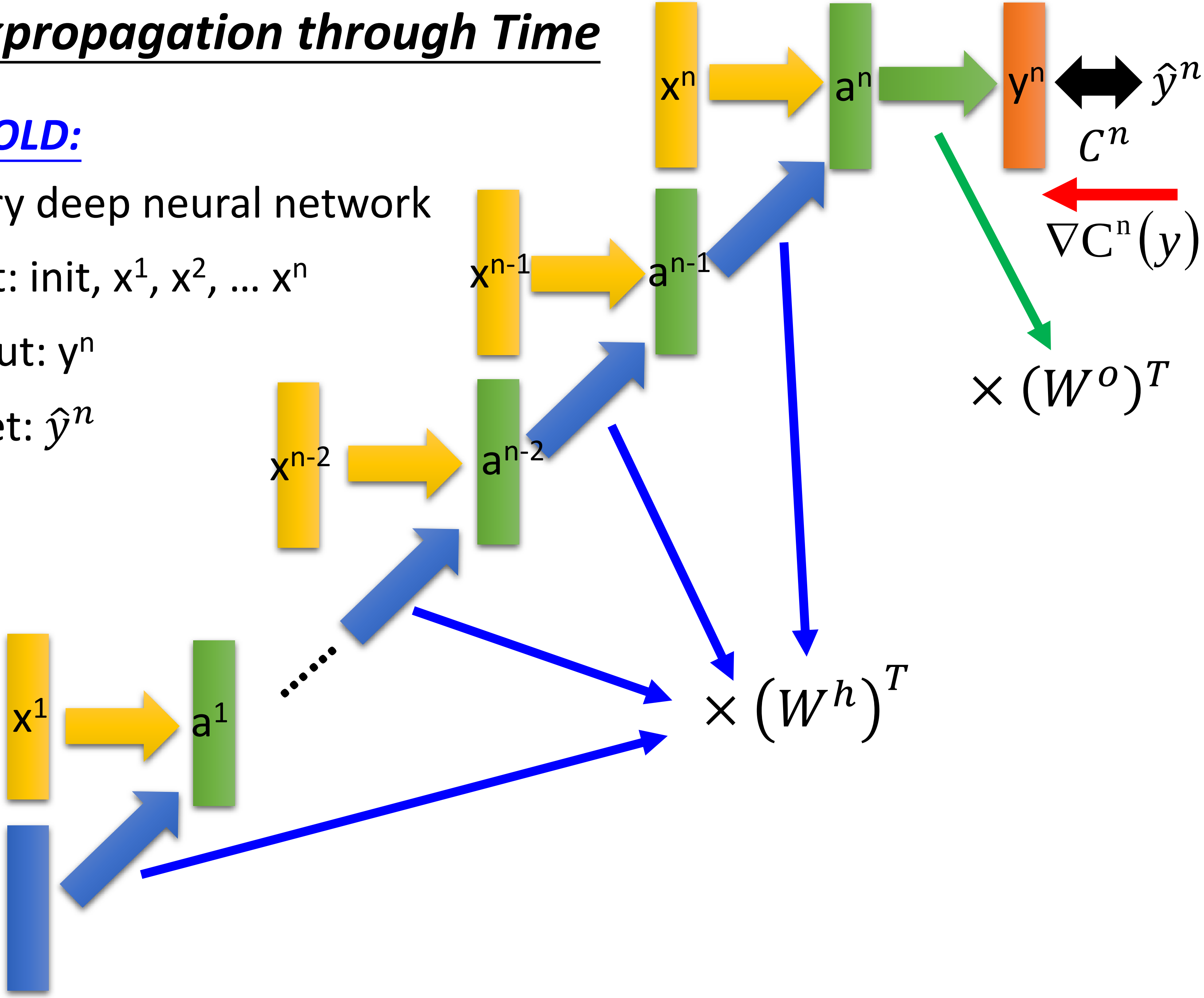
UNFOLD:

A very deep neural network

Input: init, x^1, x^2, … x^n

output: y^n

target: $\hat{y}^n$



Backpropagation through Time

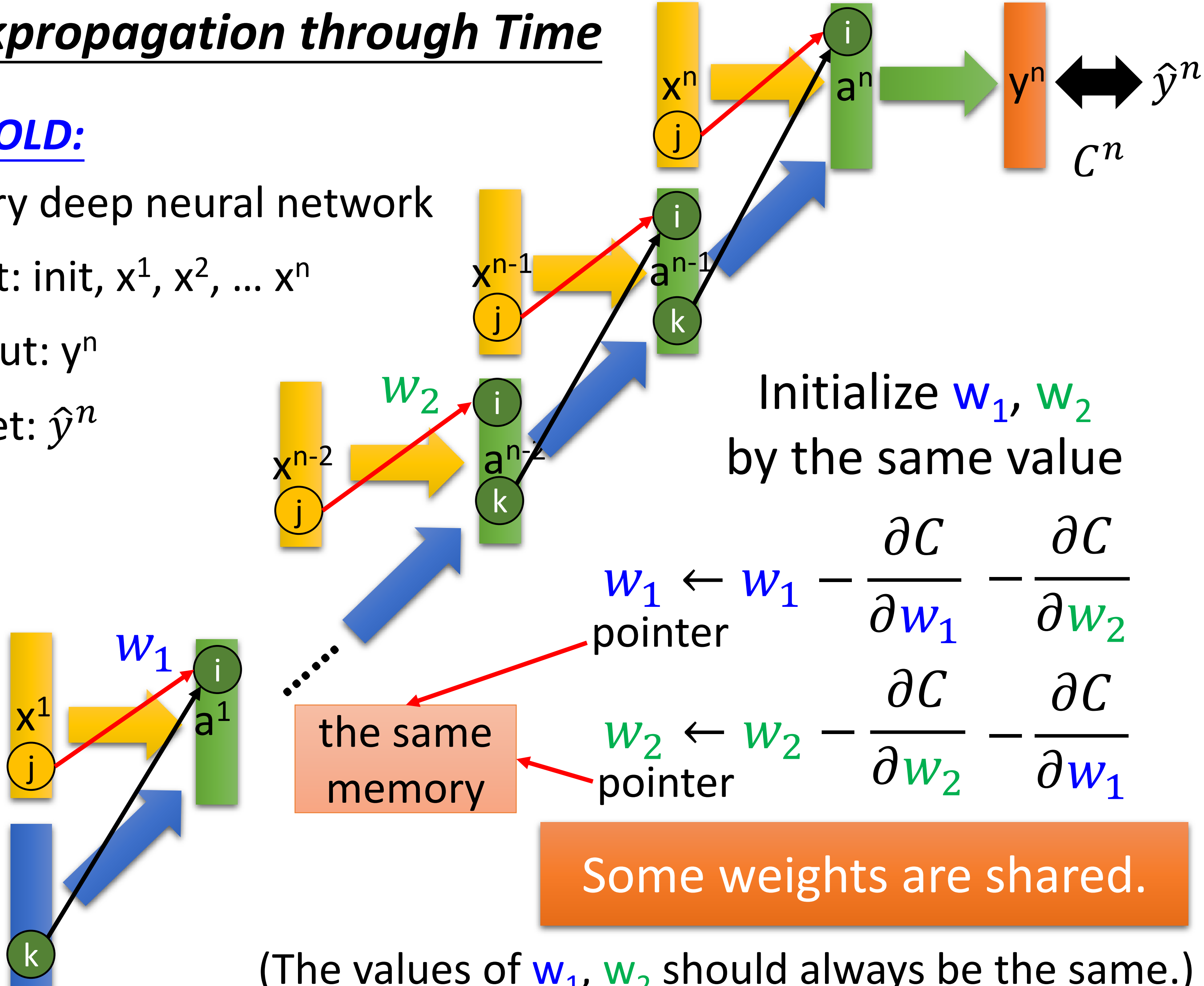
UNFOLD:

A very deep neural network

Input: init, x^1, x^2, … x^n

output: y^n

target: $\hat{y}^n$



Initialize w_1, w_2 by the same value

$$w_1 \leftarrow w_1 - \frac{\partial C}{\partial w_1} - \frac{\partial C}{\partial w_2}$$

$$w_2 \leftarrow w_2 - \frac{\partial C}{\partial w_2} - \frac{\partial C}{\partial w_1}$$

Some weights are shared.

(The values of w_1, w_2 should always be the same.)

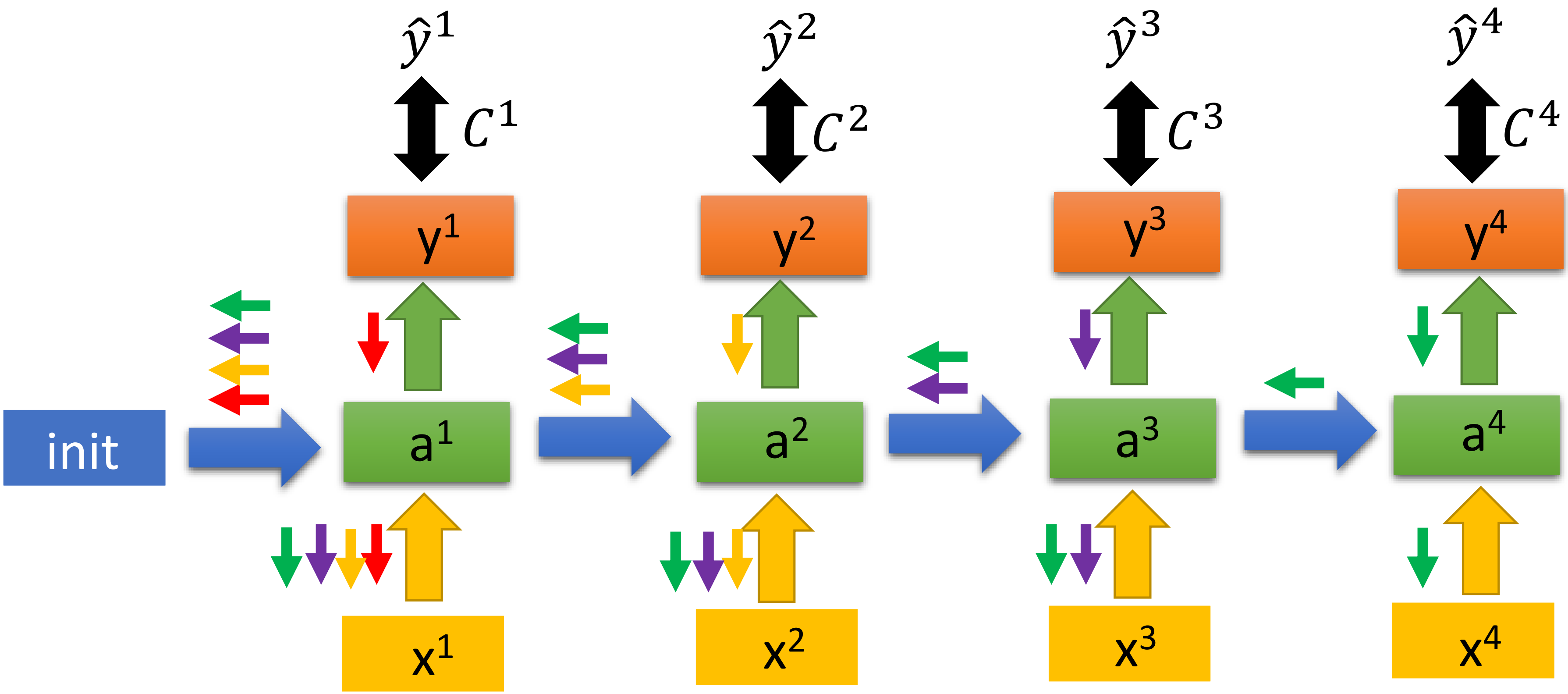
BPTT

Forward Pass: Compute a^1, a^2, a^3, a^4 ......

Backward Pass:

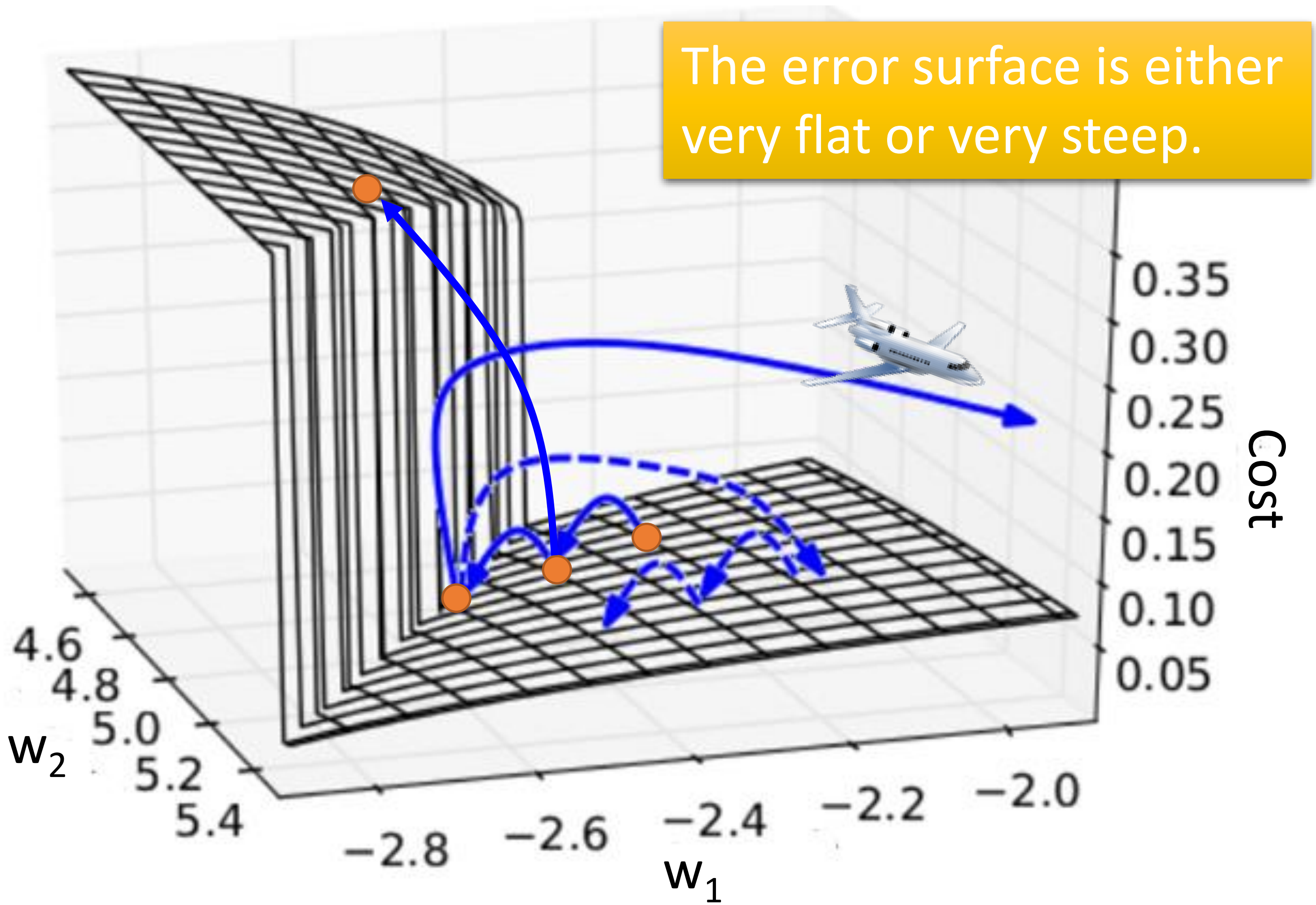
→ For C^4 → For C^3

→ For C^2 → For C^1



Unfortunately, it is not easy to train RNN.

The error surface is rough.



Source: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf

Toy Example

$$\frac{\partial C^n}{\partial w} = \frac{\partial C^n}{\partial y^r}\boxed{\frac{\partial y^n}{\partial w}} \qquad \frac{\partial y^n}{\partial w} \approx \frac{\Delta y^n}{\Delta w}$$

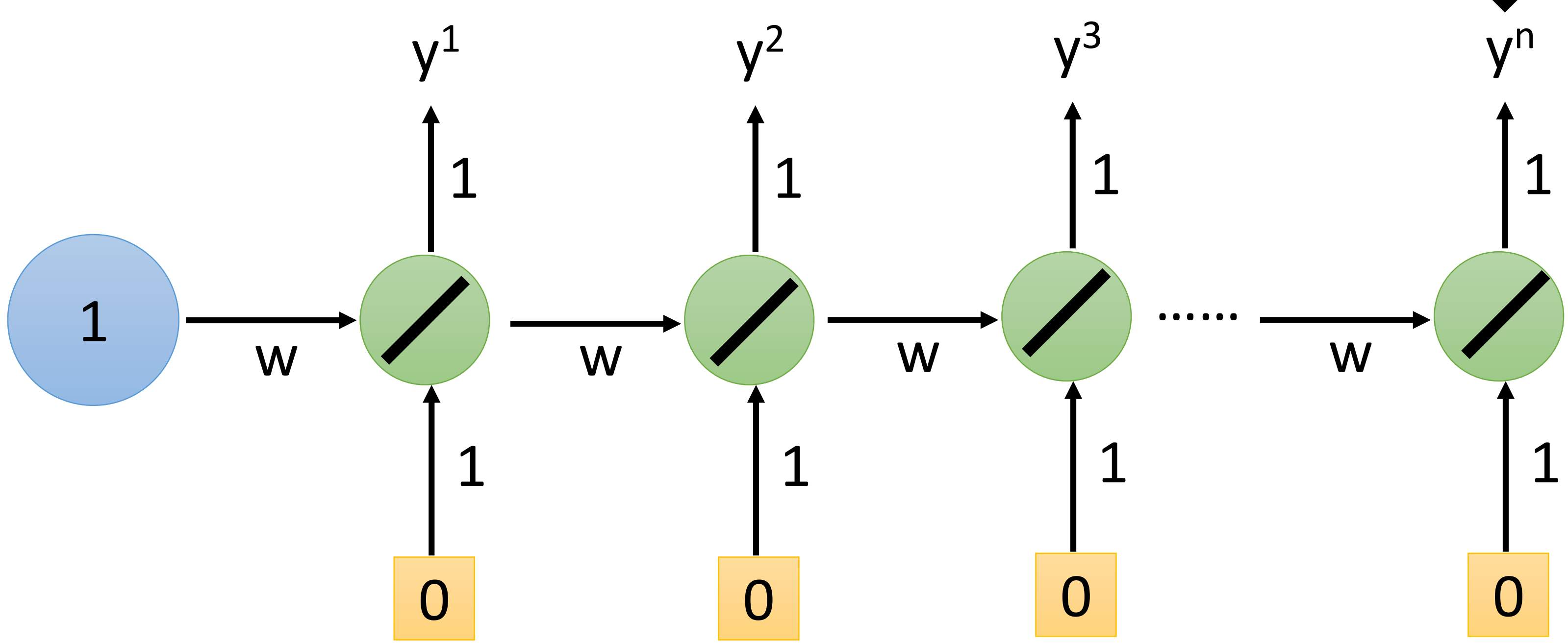
If n = 1000:

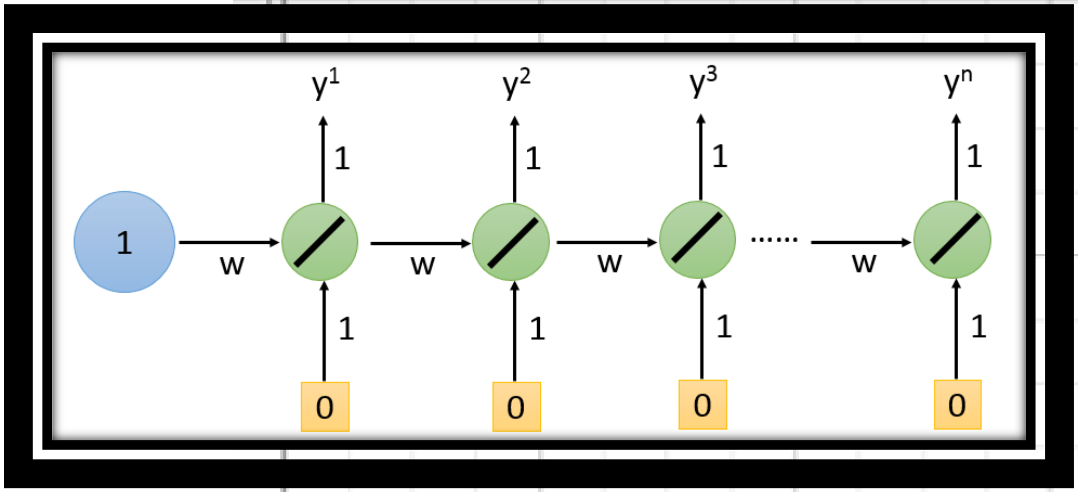
$w = 1$ ⇒ $y^n = 1$

$w = 1.01$ ⇒ $y^n \approx 20000$

$w = 0.99$ ⇒ $y^n \approx 0$

$w = 0.01$ ⇒ $y^n \approx 0$

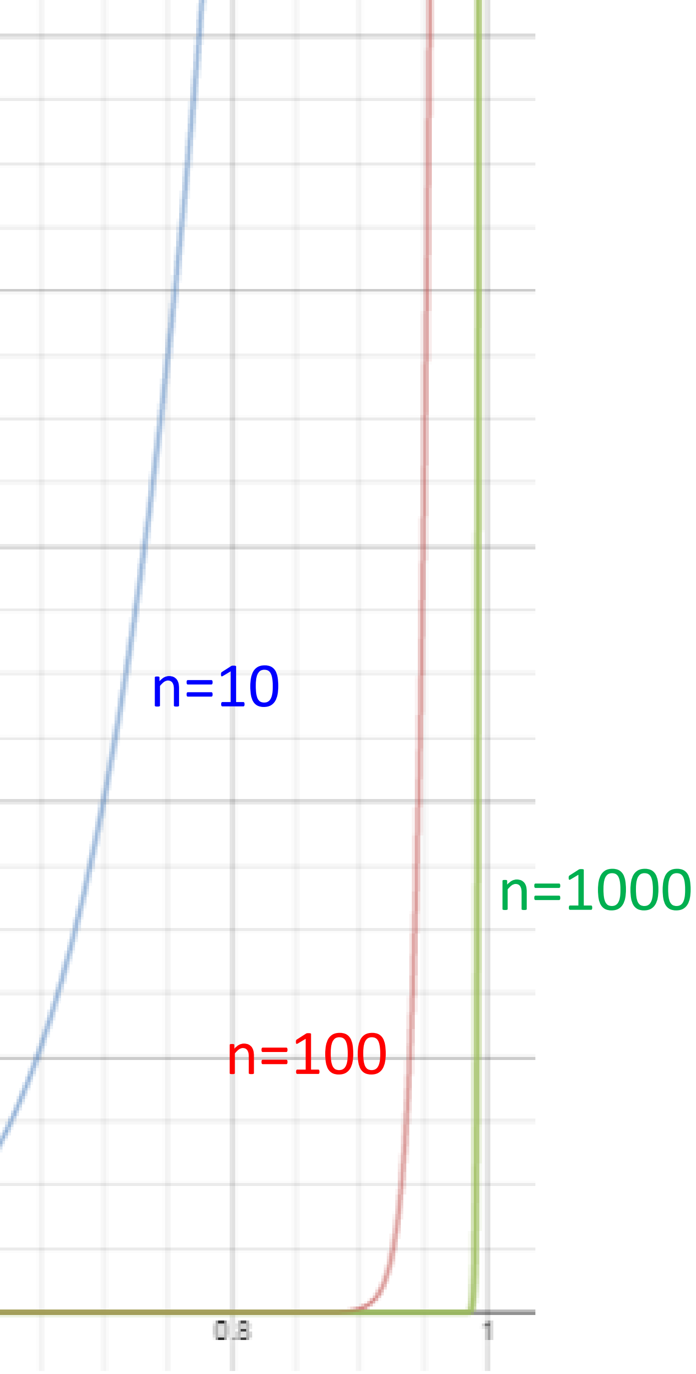




$$y^n = (w)^n$$

$$\frac{\partial y^n}{\partial w} = n(w)^{n-1}$$

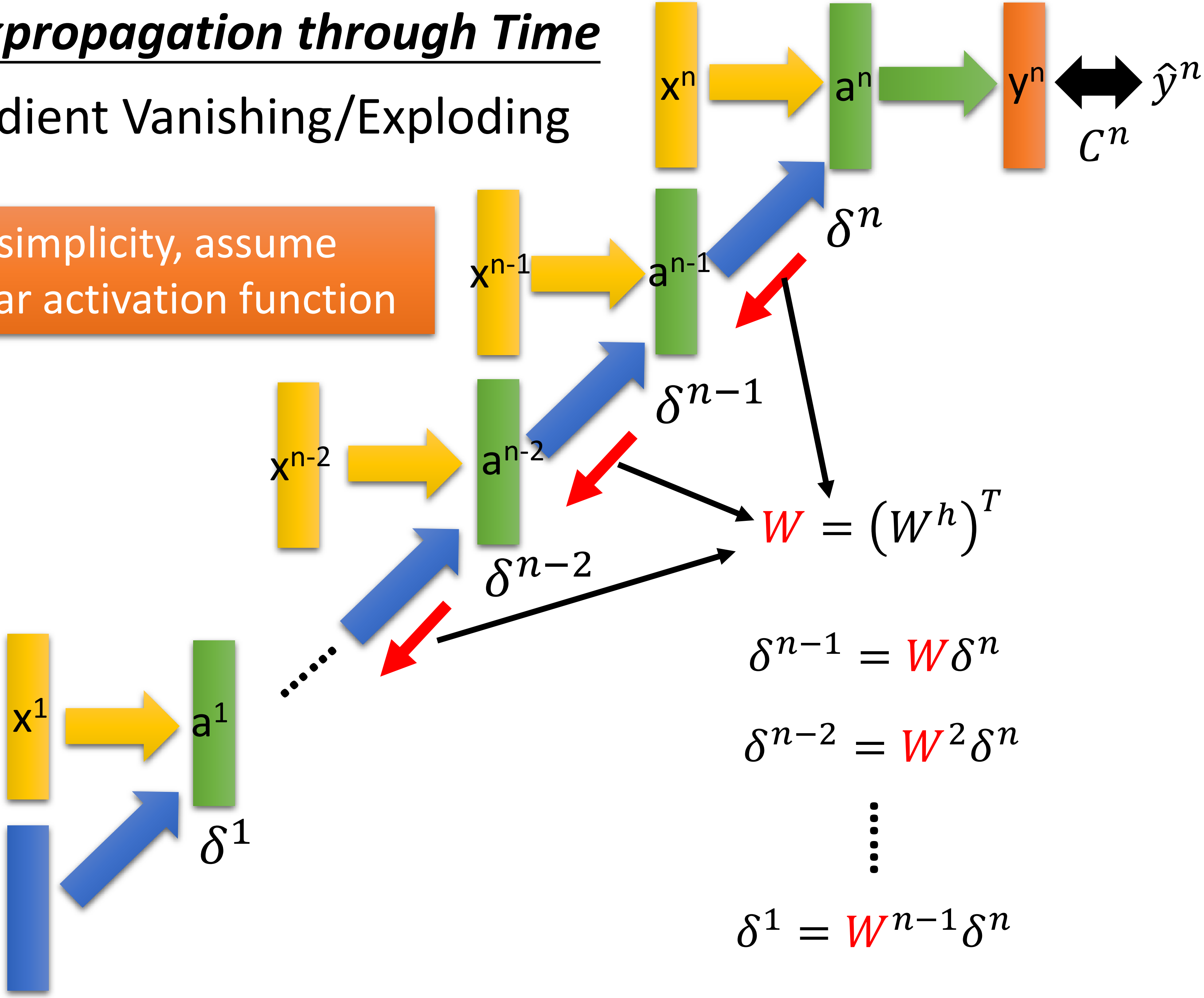
Only extremely large and small value



Backpropagation through Time

Gradient Vanishing/Exploding

For simplicity, assume linear activation function



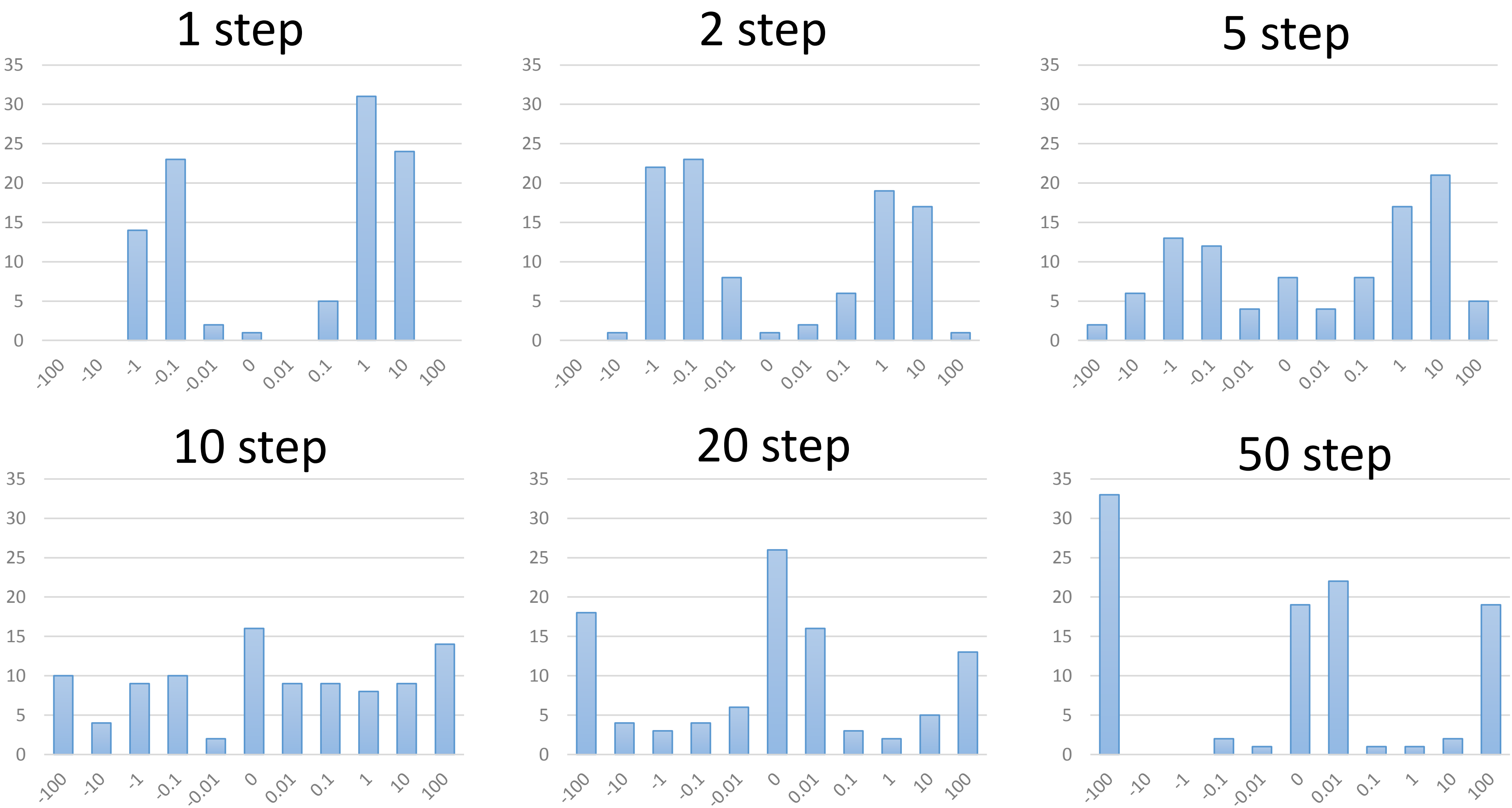
$$\delta^{n-1} = W\delta^n$$

$$\delta^{n-2} = W^2\delta^n$$

$$\vdots$$

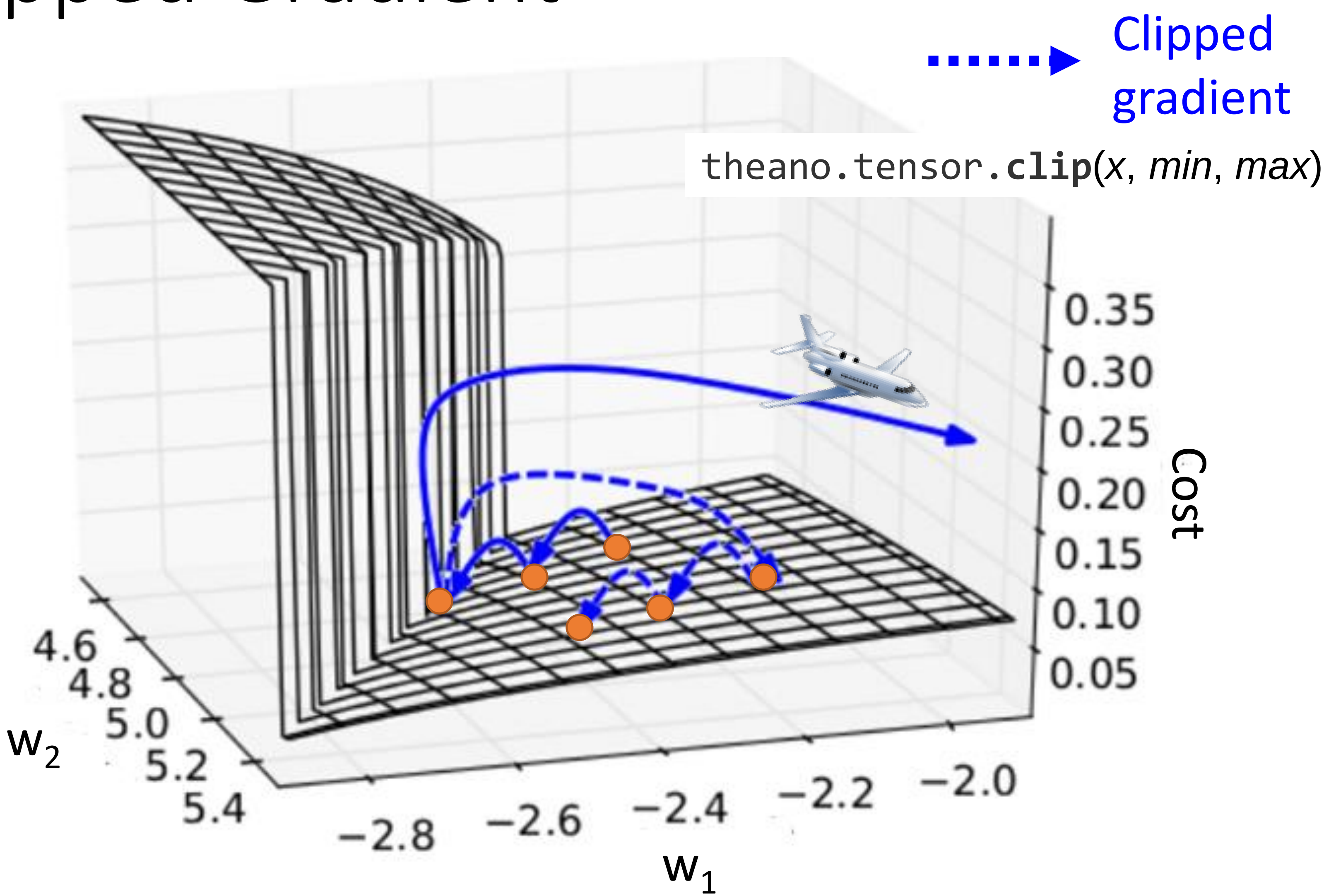
$$\delta^1 = W^{n-1}\delta^n$$

Gradient Vanishing/Exploding



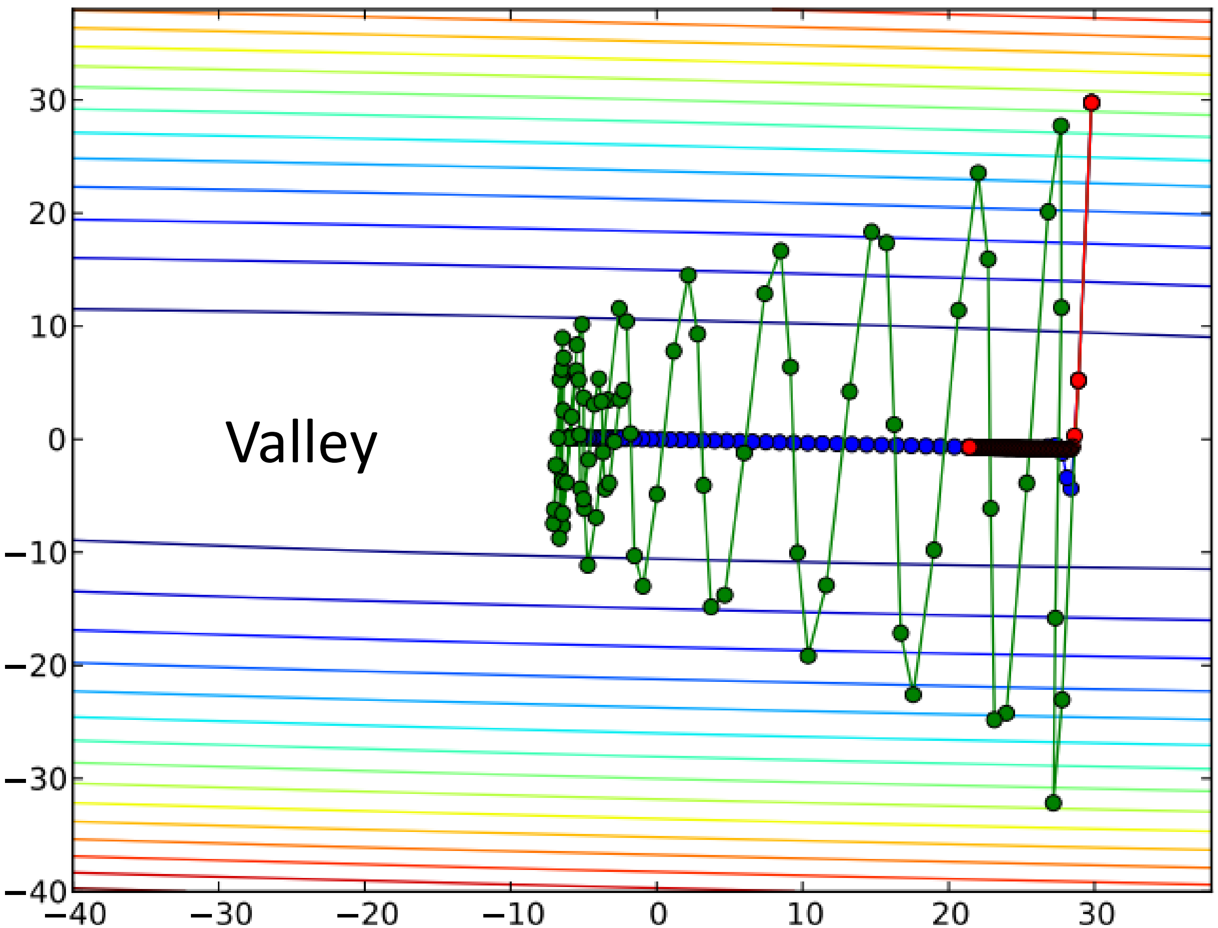
Possible Solutions

Clipped Gradient



Source: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf

NAG

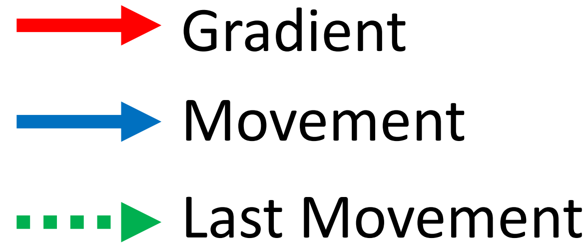


Methods:

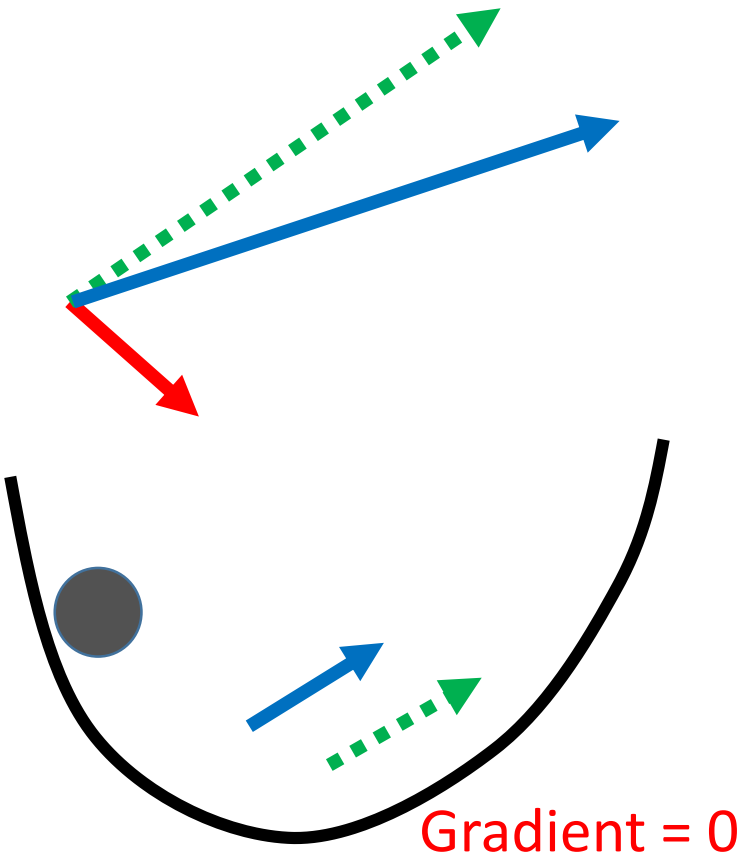
- Gradient descent
- Momentum
- Nesterov's Accelerated Gradient (NAG)

Source: http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

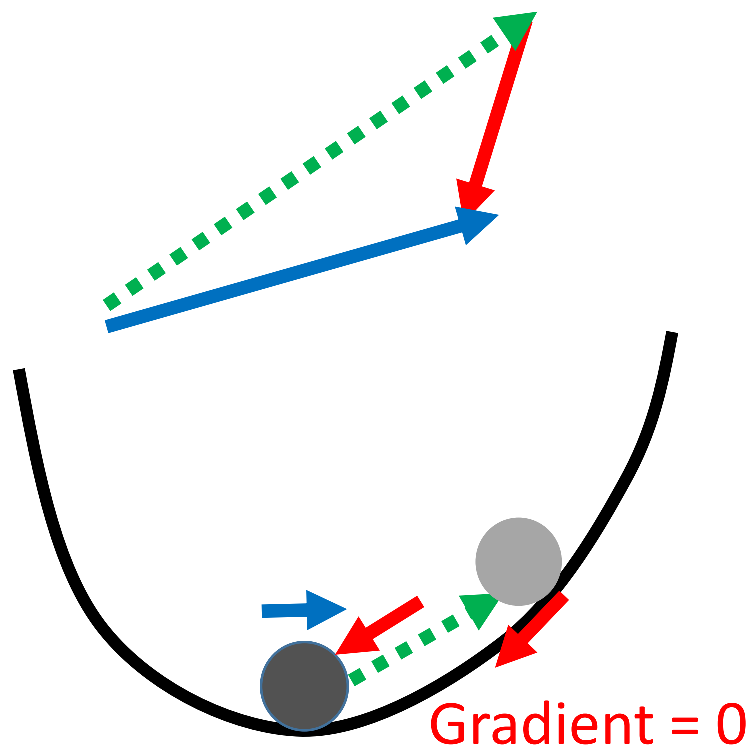
NAG



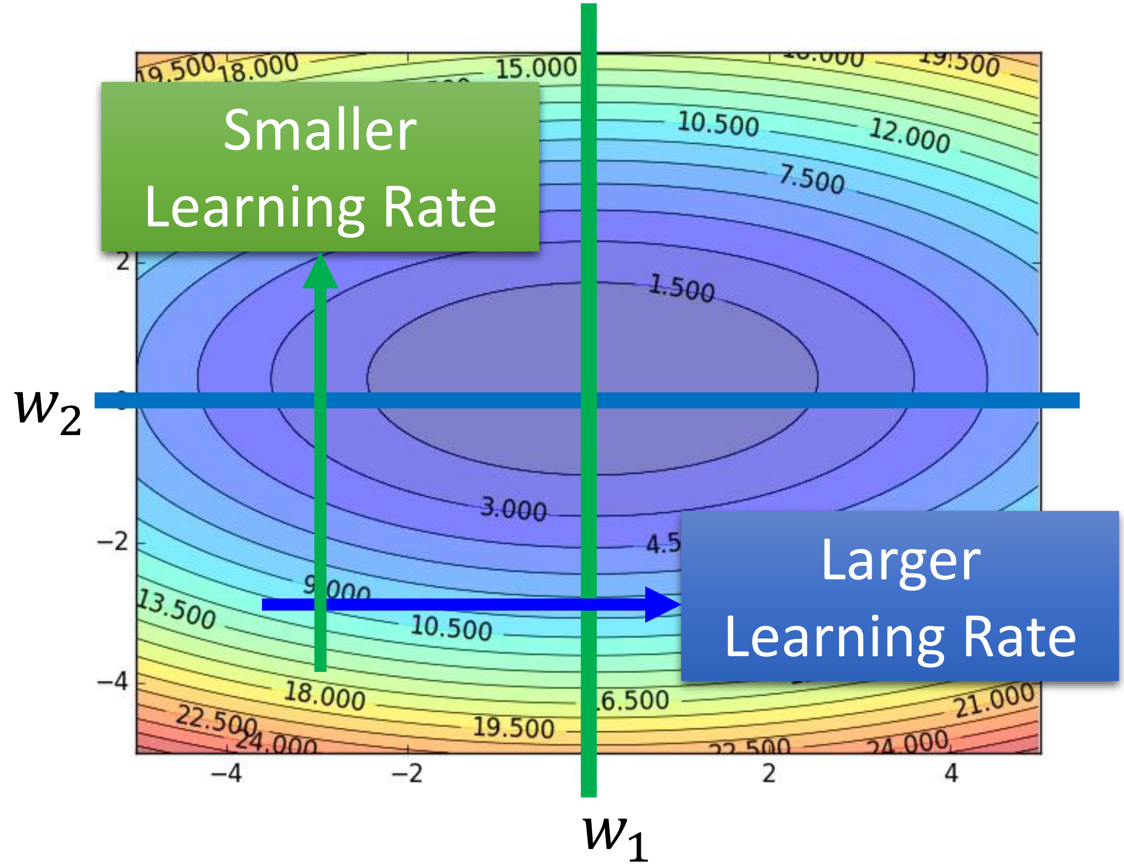
- Momentum



- Nesterov's Accelerated Gradient (NAG)



RMSProp



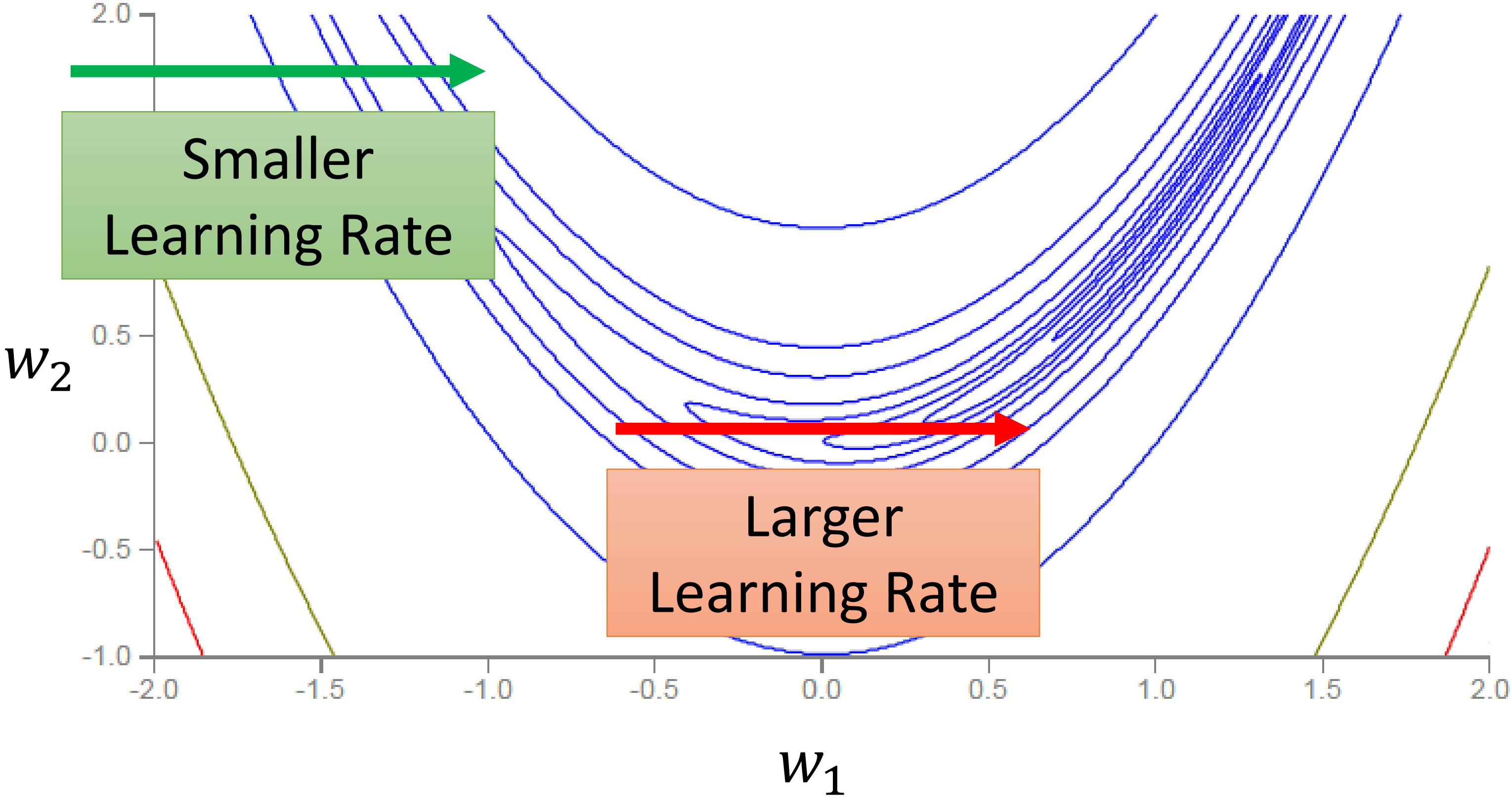
Review: Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}} g^t$$

Use first derivative to estimate second derivative

RMSProp

Error Surface can be even more complex when training RNN.



RMSProp

$$w^1 \leftarrow w^0 - \frac{\eta}{\sigma^0} g^0 \qquad \sigma^0 = g^0$$

$$w^2 \leftarrow w^1 - \frac{\eta}{\sigma^1} g^1 \qquad \sigma^1 = \sqrt{\alpha(\sigma^0)^2 + (1-\alpha)(g^1)^2}$$

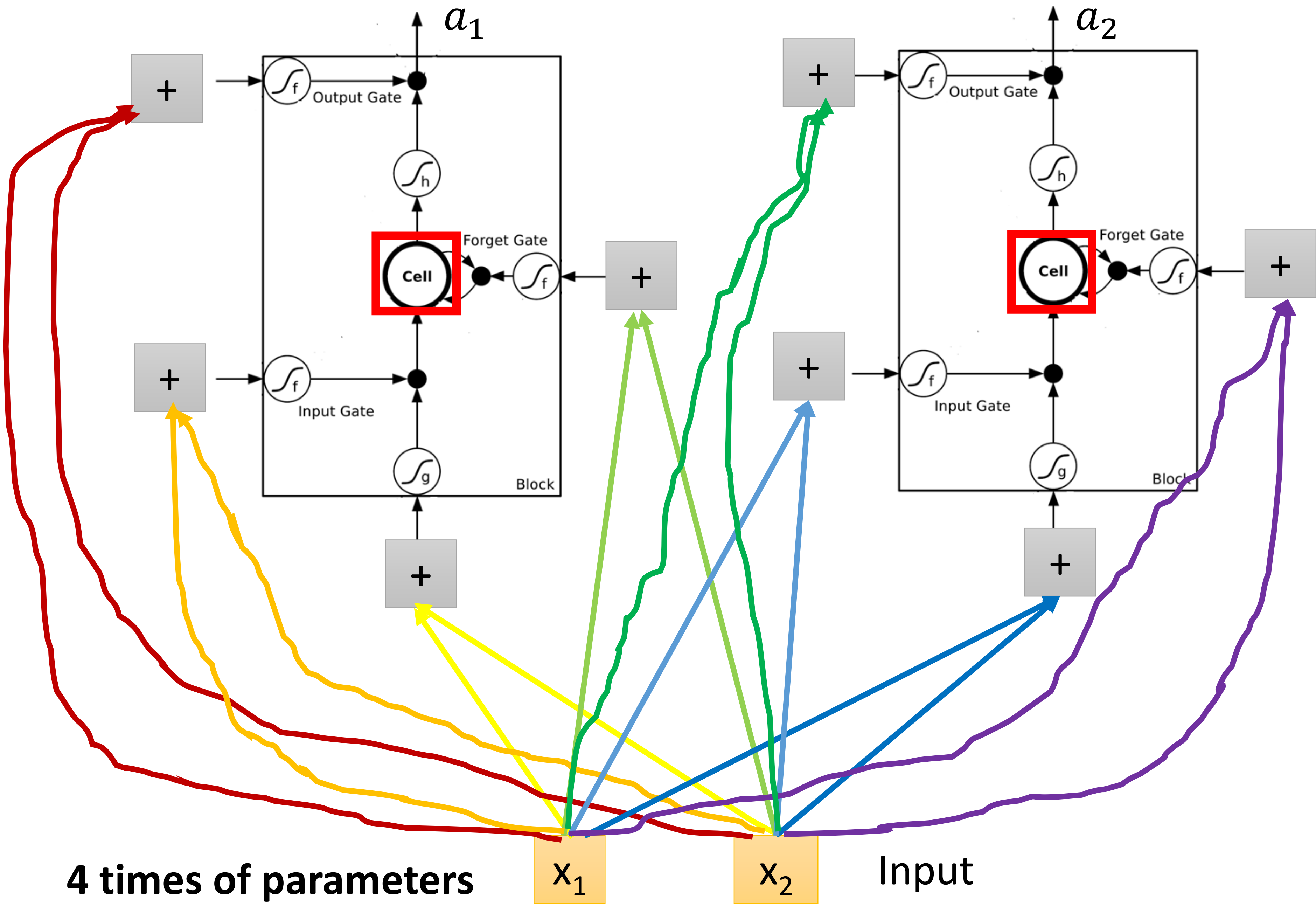
$$w^3 \leftarrow w^2 - \frac{\eta}{\sigma^2} g^2 \qquad \sigma^2 = \sqrt{\alpha(\sigma^1)^2 + (1-\alpha)(g^2)^2}$$

⋮

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \qquad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$

Root Mean Square of the gradients
with previous gradients being decayed

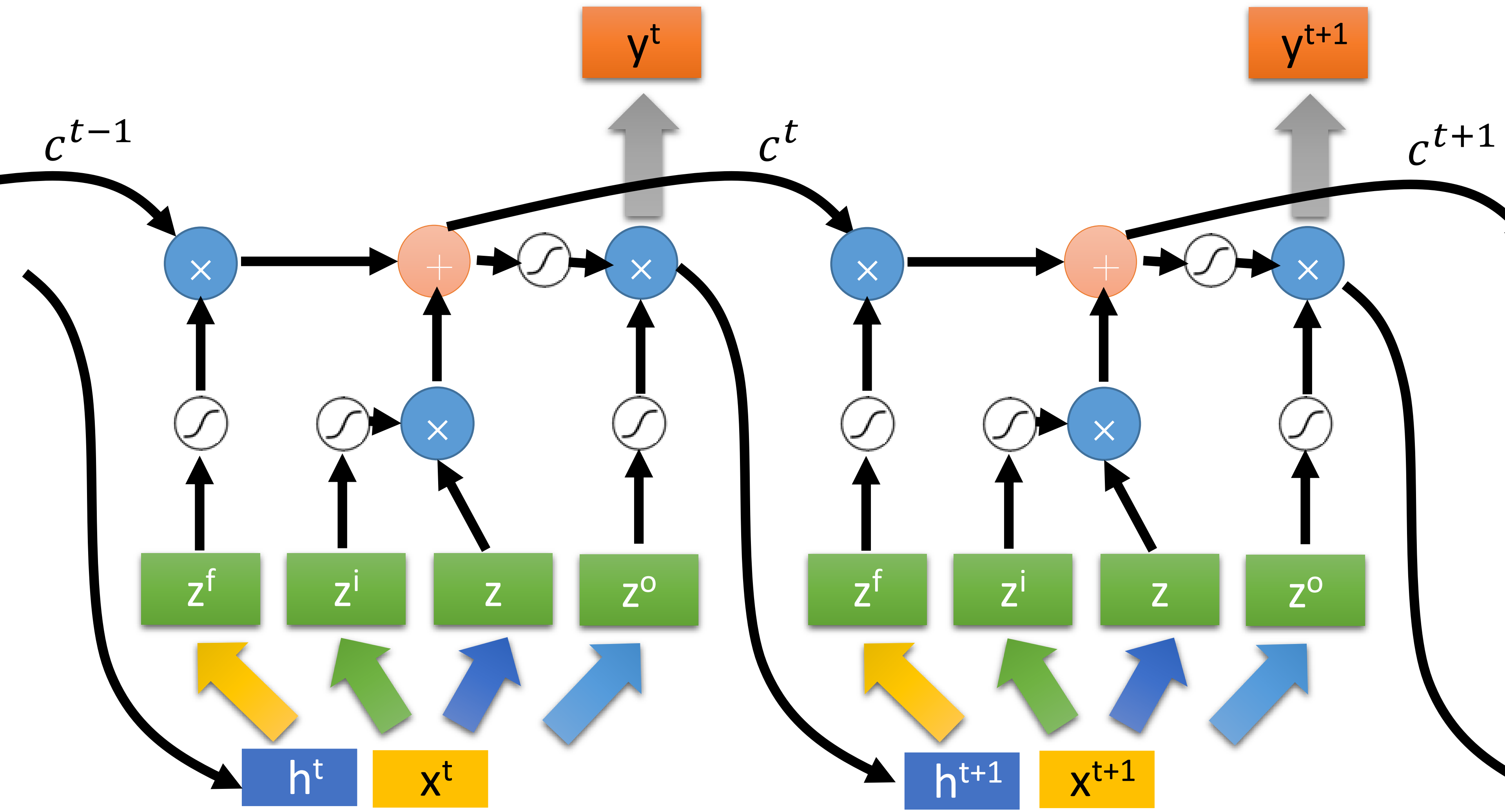
LSTM can address the gradient vanishing problem.



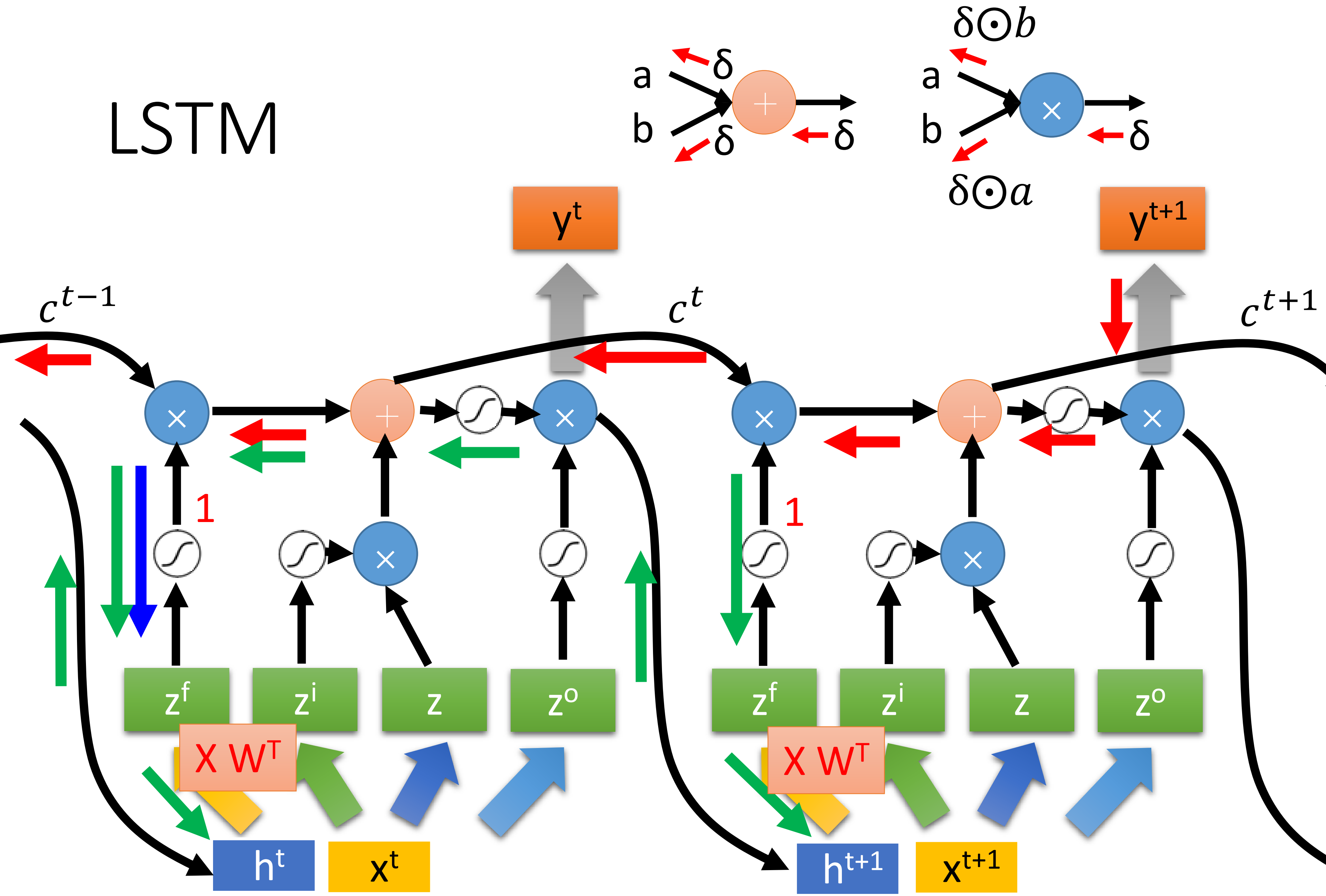
4 times of parameters

LSTM

Extension: "peephole"

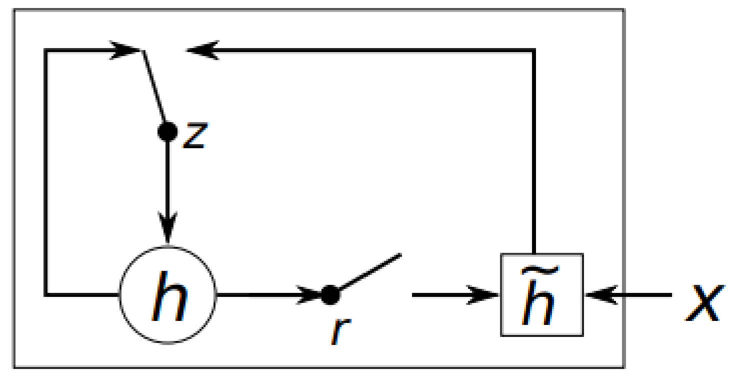


LSTM

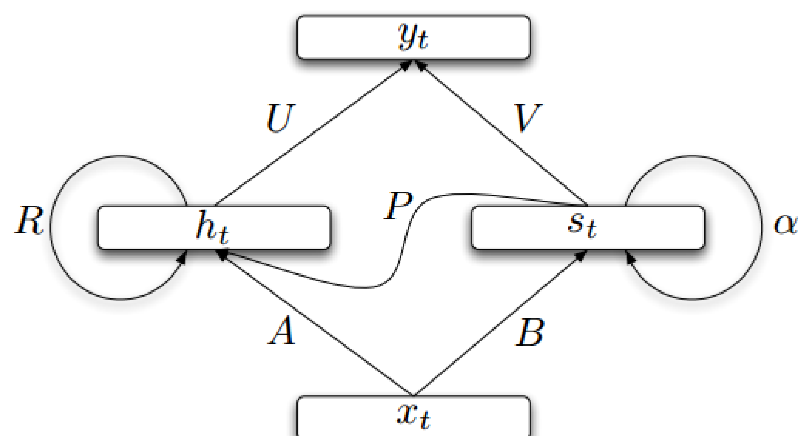


Other Simpler Variants

- GRU: Cho, Kyunghyun, et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation“, EMNLP, 2014



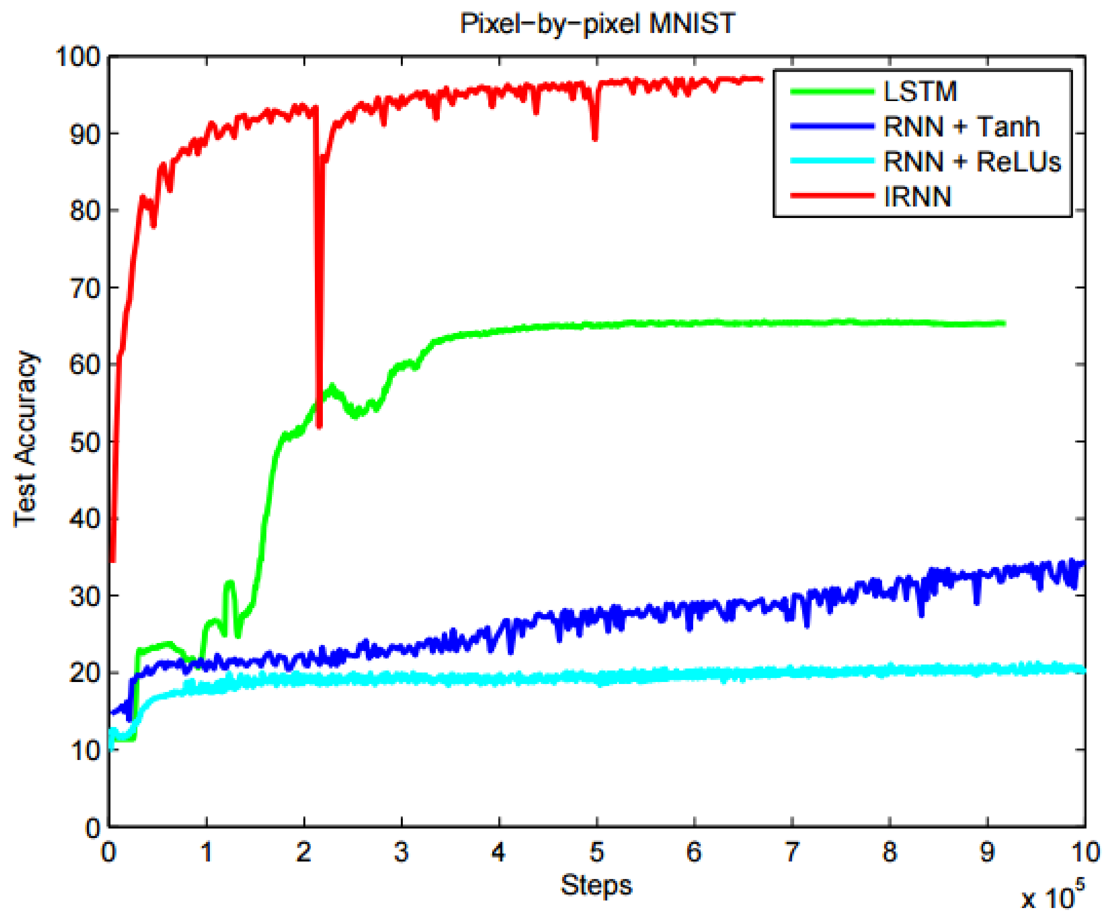
- SCRN: Mikolov, Tomas, et al. "Learning longer memory in recurrent neural networks“, ICLR 2015



Better Initialization

Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units", 2015

- Vanilla RNN: Initialized with Identity matrix + ReLU



Concluding Remarks

- Be careful when training RNN …
- Possible solution:
  - Clipping the gradients
  - Advanced optimization technology
    - NAG
    - RMSprop
  - Try LSTM (or other simpler variants)
  - Better initialization