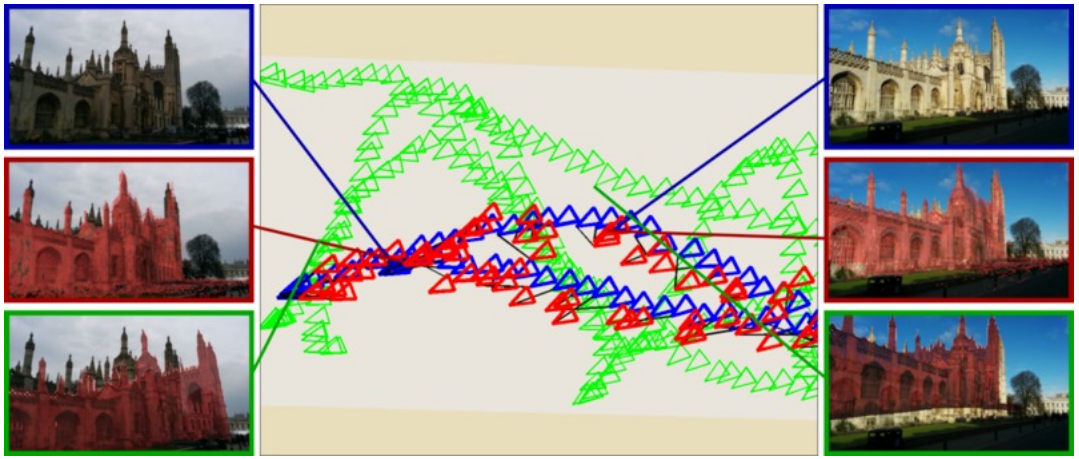
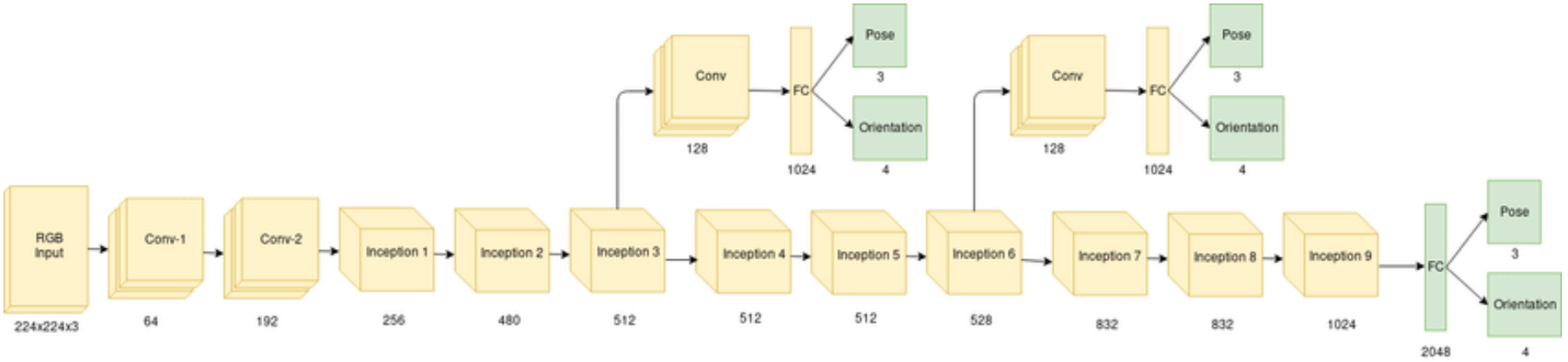
Regression - Application

PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization





https://arxiv.org/pdf/1505.07427.pdf

NEXT VISION

Spinoff of the University of Catania

Thesis in this context are available.

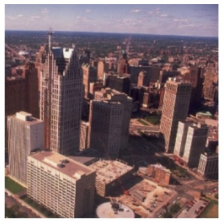
Metric Learning

Metric Learning

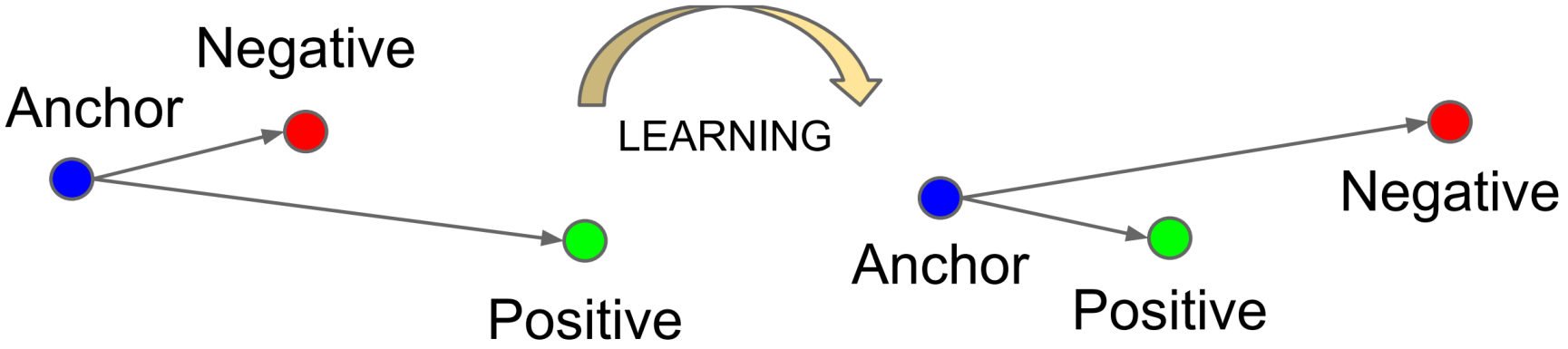
try to fit a metric embedding S so that:

$$S(x, x_1) > S(x, x_2), \quad \forall x, x_1, x_2 \in \mathbb{P} \quad \text{for which } r(x, x_1) > r(x, x_2).$$

where $r(x, x')$ is a rough similarity measure given by an oracle. We focus on finding an L_2 embedding, by learning a function $F(x)$ for which $S(x, x') = \|F(x) - F(x')\|_2$.

$D($  ,  $) < D($  ,  $)$

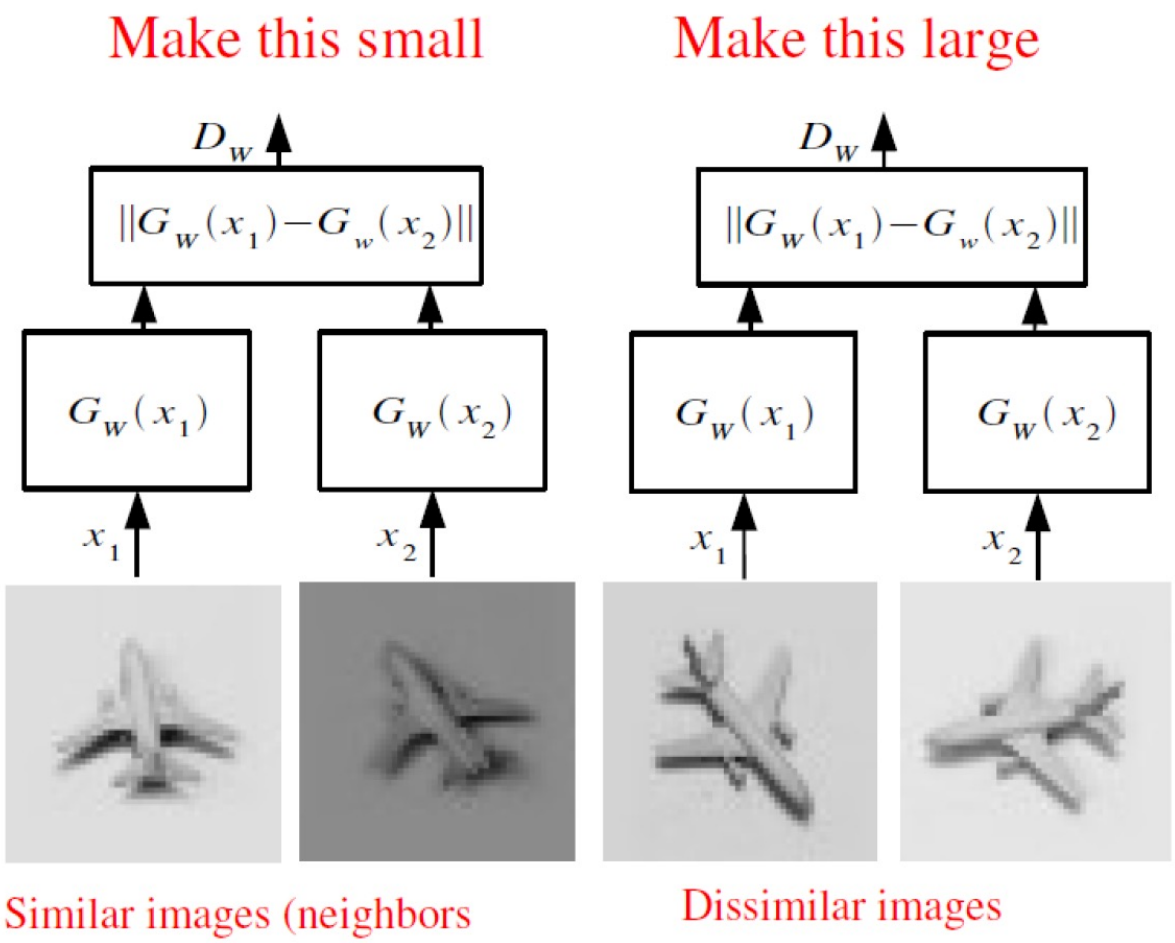
Metric Learning



We need F which minimizes the distance between an anchor and a positive sample, both of which have the same identity/label or that are “similar”, and maximizes the distance between the anchor and a negative sample with a different identity/label or that is “dissimilar”.

Florian Schroff, Dmitry Kalenichenko, James Philbin, FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR 2015

Siamese Network



Labels for pairs:
Similar→0
Dissimilar→1

Here $F=G_w$

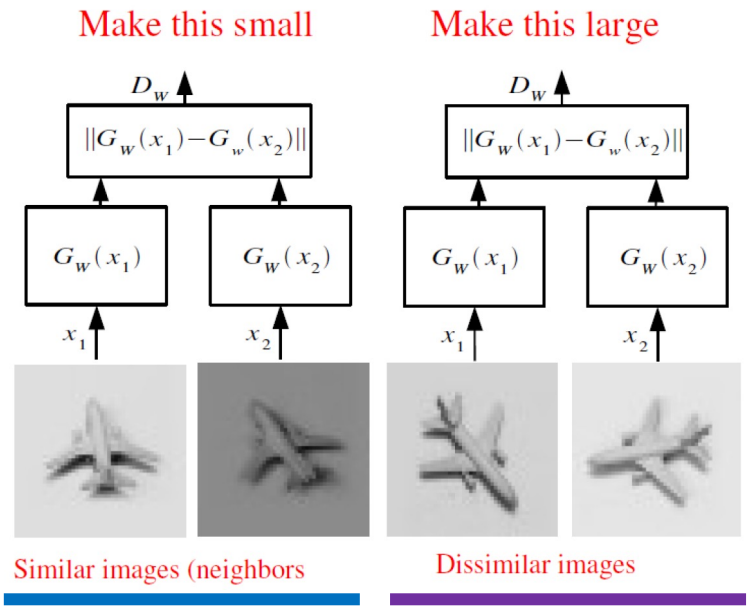
G is our network architecture, and w are the parameters

You can add comparator and also use a softmax as last layer

How to define Loss?

Signature Verification using a "Siamese". Time Delay Neural Network, 1994
https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf

Contrastive Loss



$$y = \begin{cases} 0 & \text{if } (x_1, x_2) \text{ similar pair} \\ 1 & \text{if } (x_1, x_2) \text{ dissimilar pair} \end{cases}$$

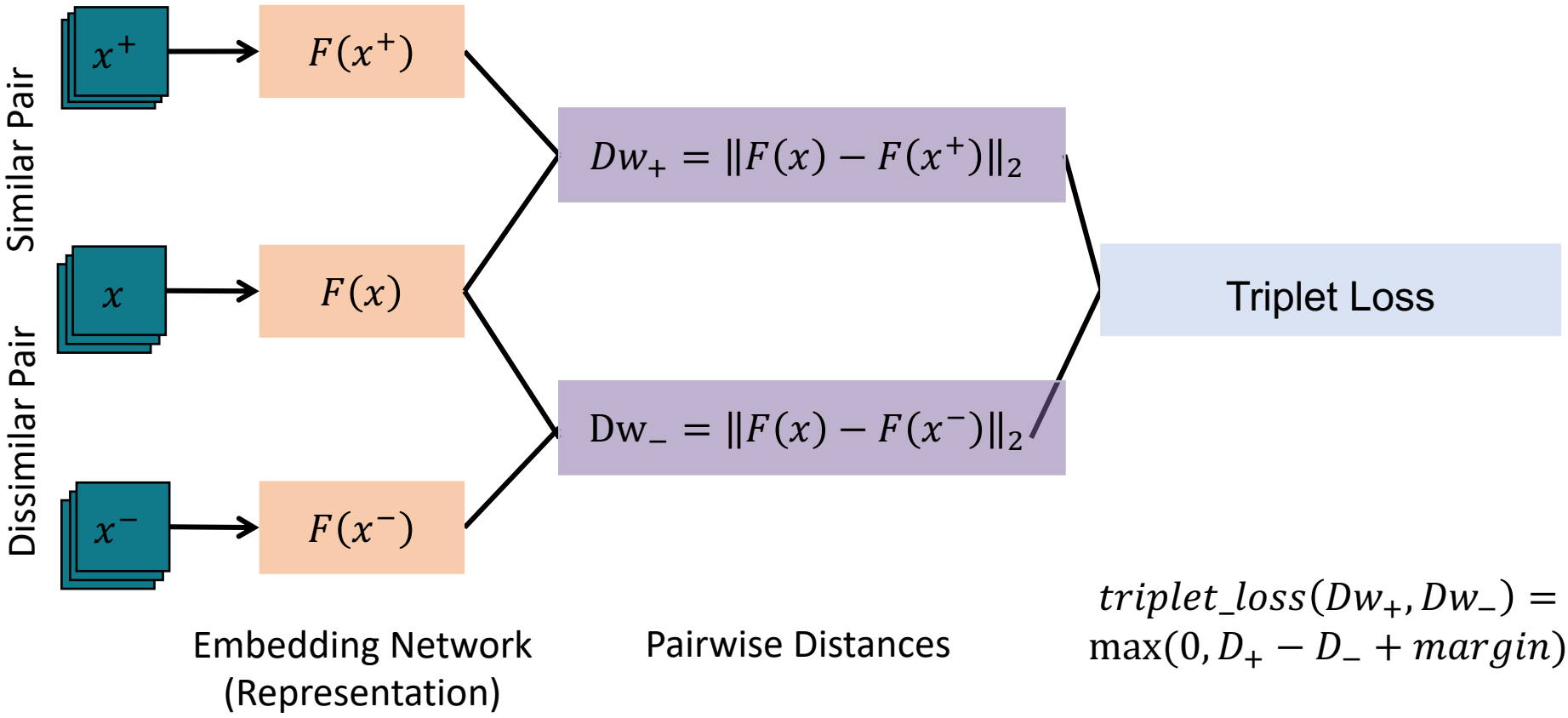
margin m: desirable distance for dissimilar pair (x_1,x_2)

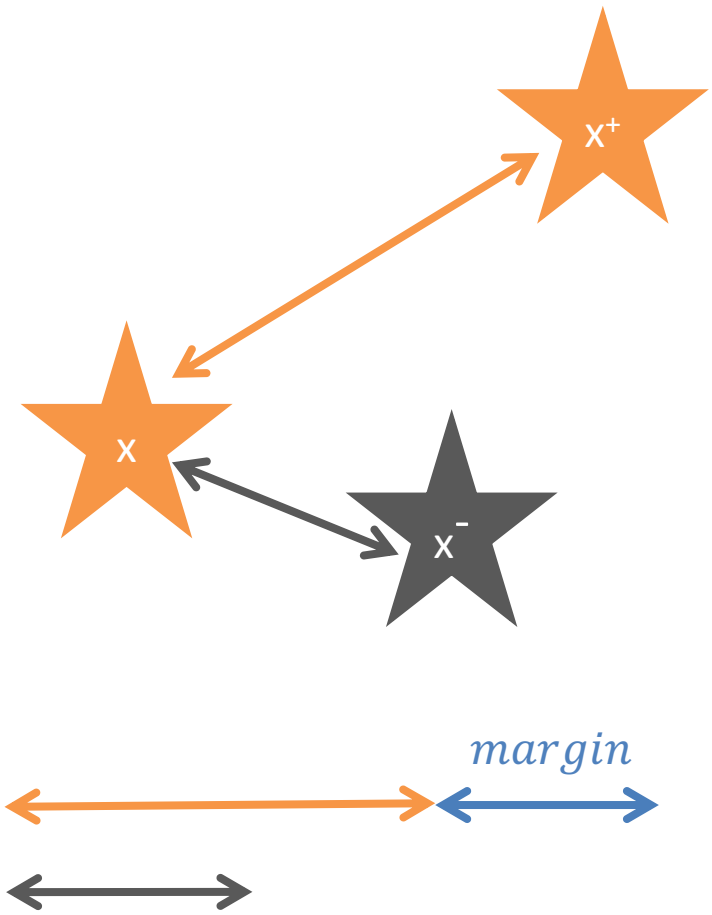
$$contrastive_{loss(W,x_1,x_2)} = \underbrace{\frac{1}{2}(1-y) * D_w^2}_{\text{Pull}} + \underbrace{\frac{1}{2} y * max\{0, m - D_w\}^2}_{\text{Push}}$$

the first term minimizes the Euclidean distance between points when y=0
the second term maximizes the distance between the points when y=1

TRIPLET ARCHITECTURE

[Hoffer & Ailon, Deep metric learning using Triplet network, 2015]

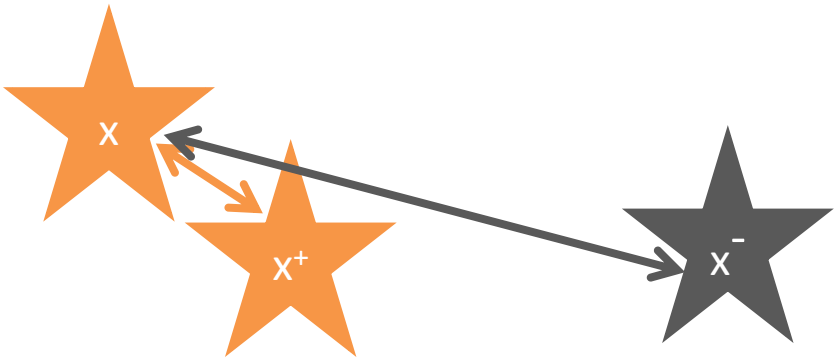




$$triplet\_loss(Dw_+, Dw\_) = \max(0, Dw_+ - Dw\_ + margin)$$

Case 1)

$Dw_+ - Dw\_ + margin$>0 → $Dw_+ + margin > Dw\_$ → we need to adjust of $(Dw_+ - Dw\_) + margin$



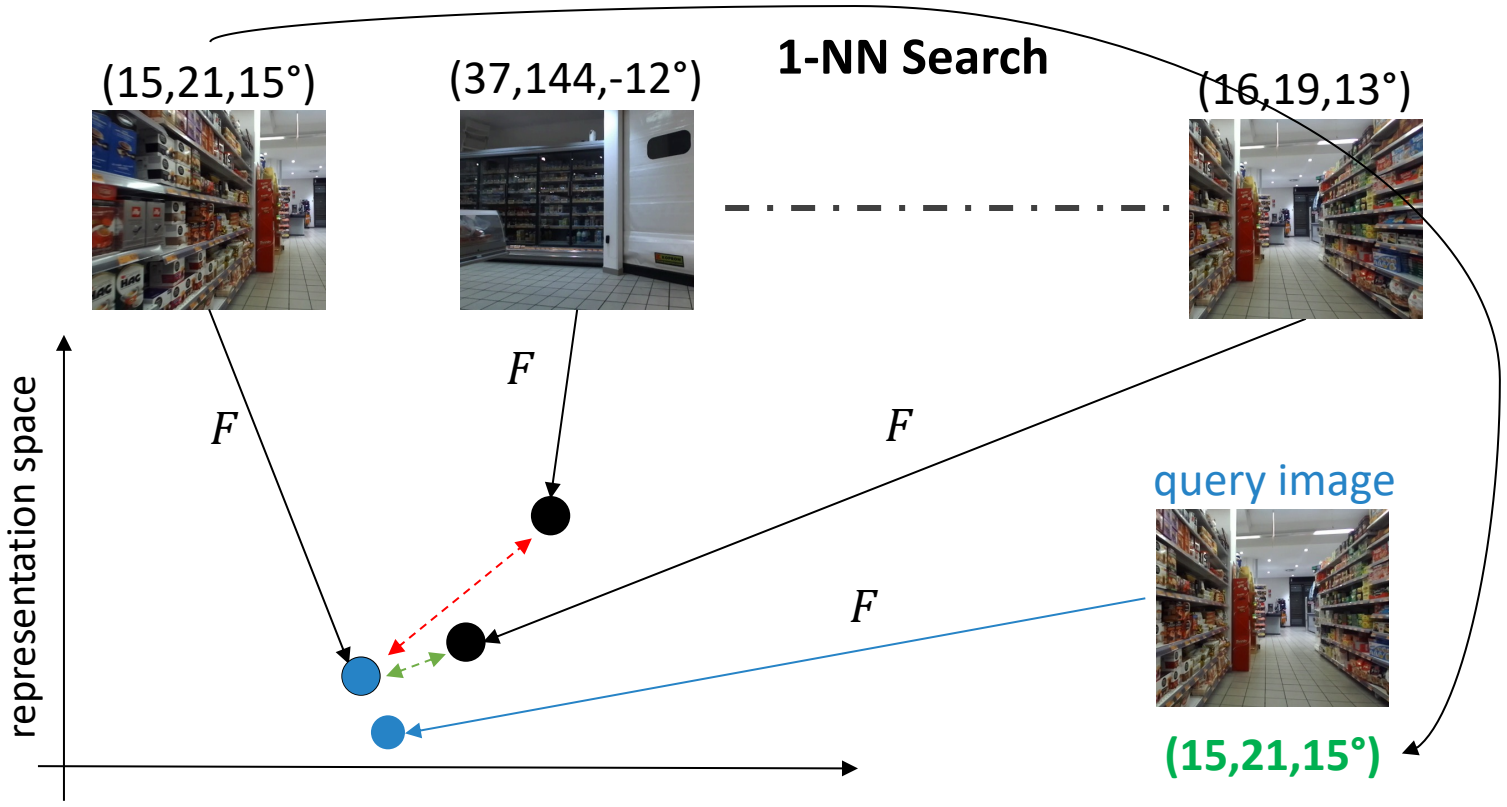
$$triplet\_loss(Dw_+, Dw\_) = \max(0, Dw_+ - Dw\_ + margin)$$

Case 2)

$Dw_+ - Dw\_ + margin \leq 0 \rightarrow Dw_+ + margin \leq Dw\_ \rightarrow$ this is fine. No need adjustment

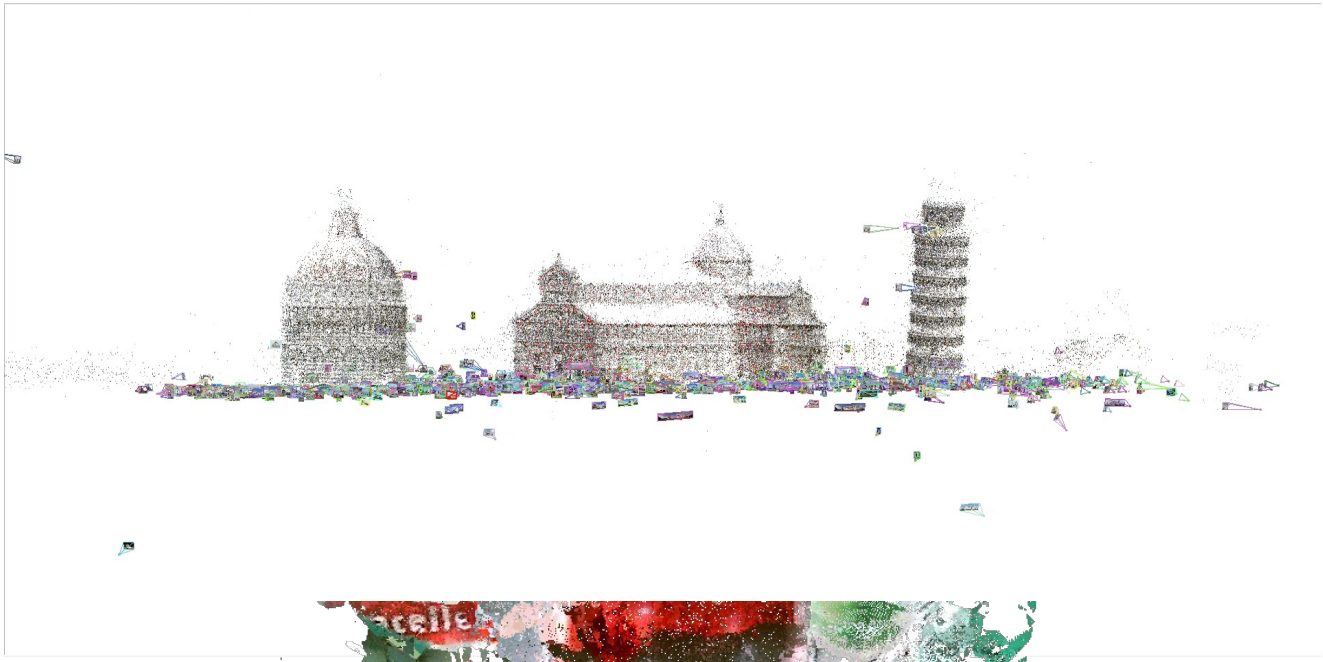
Application Example

Use deep metric learning to learn a representation function F which maps close to each other images of nearby locations



E. Spera, A. Furnari, S. Battiato, G. M. Farinella, Egocentric Shopping Cart Localization, International Conference on Pattern Recognition (ICPR), 2018

Image Labelling – Structure from Motion



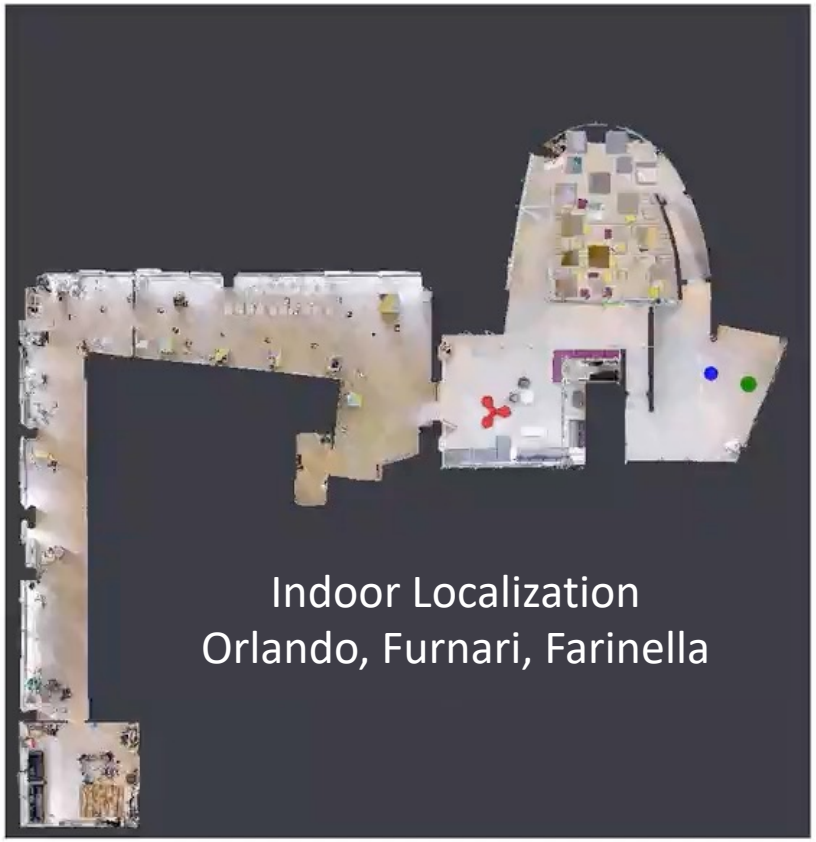
How to form triplets?

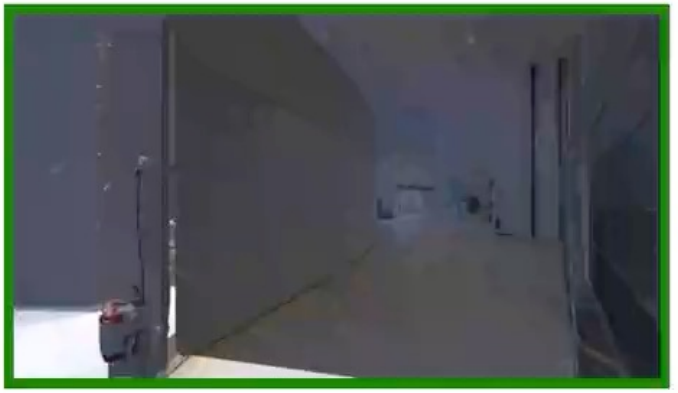
- Considering the localization example (i.e., regression problem)
  - Select a pair of position/orientation thresholds:
    - E.g., $t_p = 0.5m$ and $t_o = 45°$;
  - Given an image x, selects randomly $\overline{x}$ and it:
    - as a positive sample x^+ if:
      - distance between x and $\overline{x}$ is under t_p and
      - angular distance between x and $\overline{x}$ is under t_o;
    - as a negative sample x^- otherwise.
- What about classification?

Xenia® SOFTWARE SOLUTIONS

Image Processing LABoratory

UNIVERSITÀ degli STUDI di CATANIA





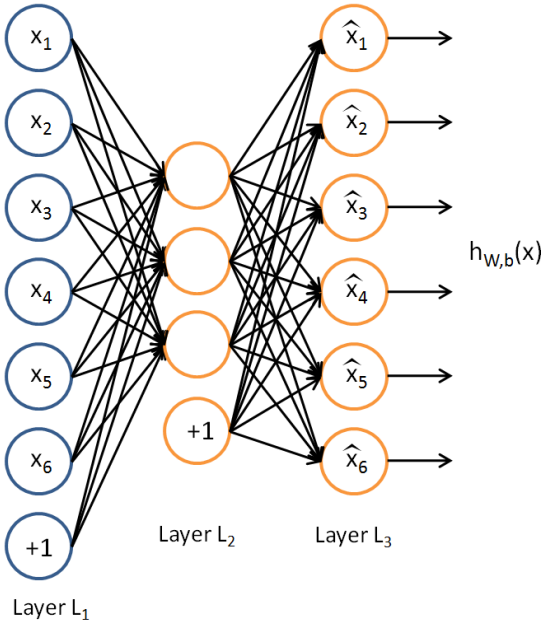
Input Position

Predicted Position

Autoencoders

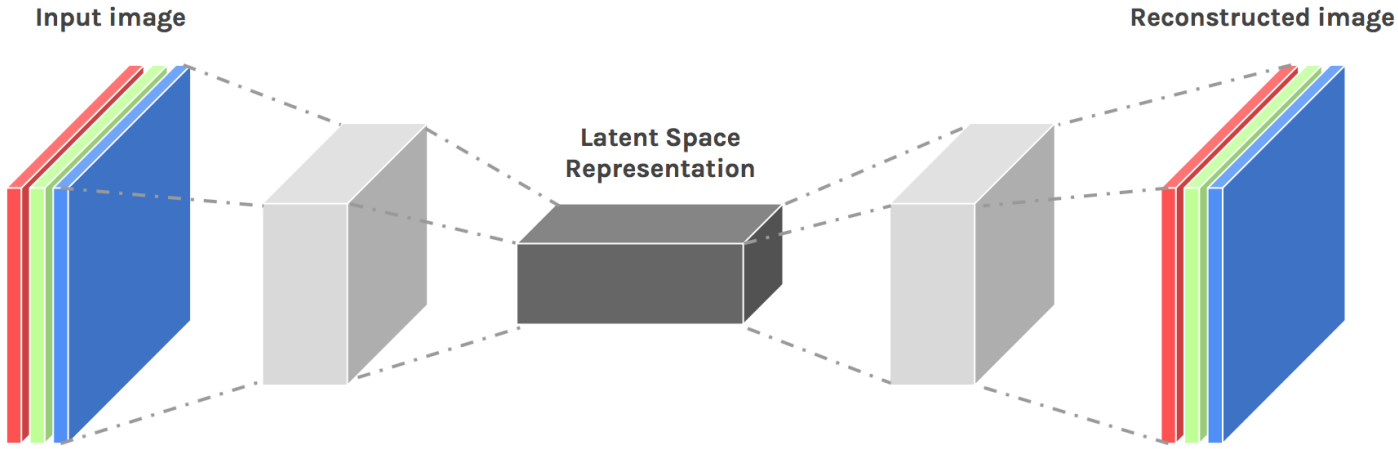
Learning representations in an unsupervised manner.

Autoencoders

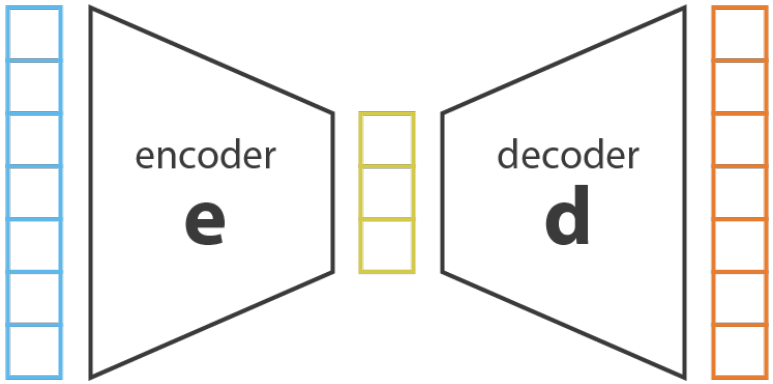


$$J = \sum_{n=1}^{N} |x(n) - \hat{x}(n)|^2$$

One possibility is to use the same objective function of PCA. Note that PCA is restricted to linear mapping.



Can be used for dimensional reduction, compression, feature extraction….



x — **initial data** in space R^n

e(x) — **encoded data** in latent space R^m (with $m<n$)

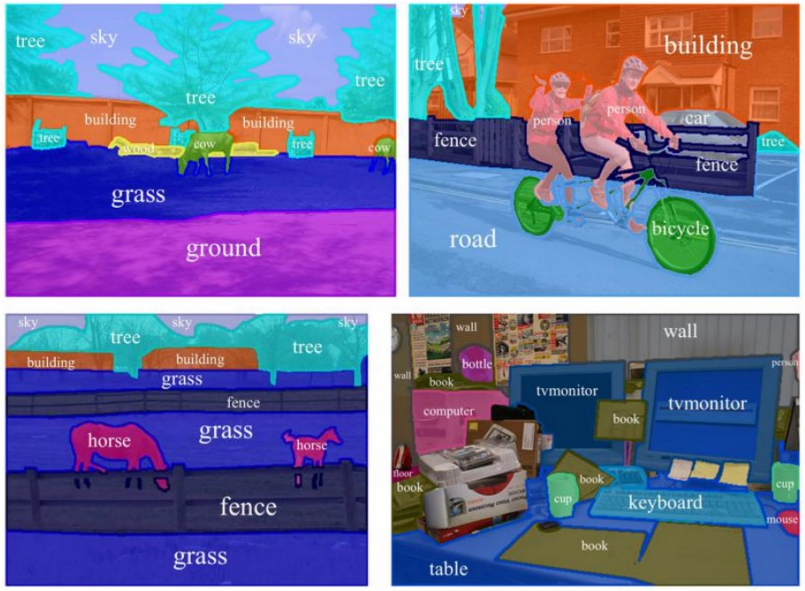
d(e(x)) — **encoded-decoded data** back in the initial space R^n

$x = d(e(x))$ ➡ **lossless encoding**
no information is lost when reducing the number of dimensions

$x \neq d(e(x))$ ➡ **lossy encoding**
some information is lost when reducing the number of dimensions and can't be recovered later

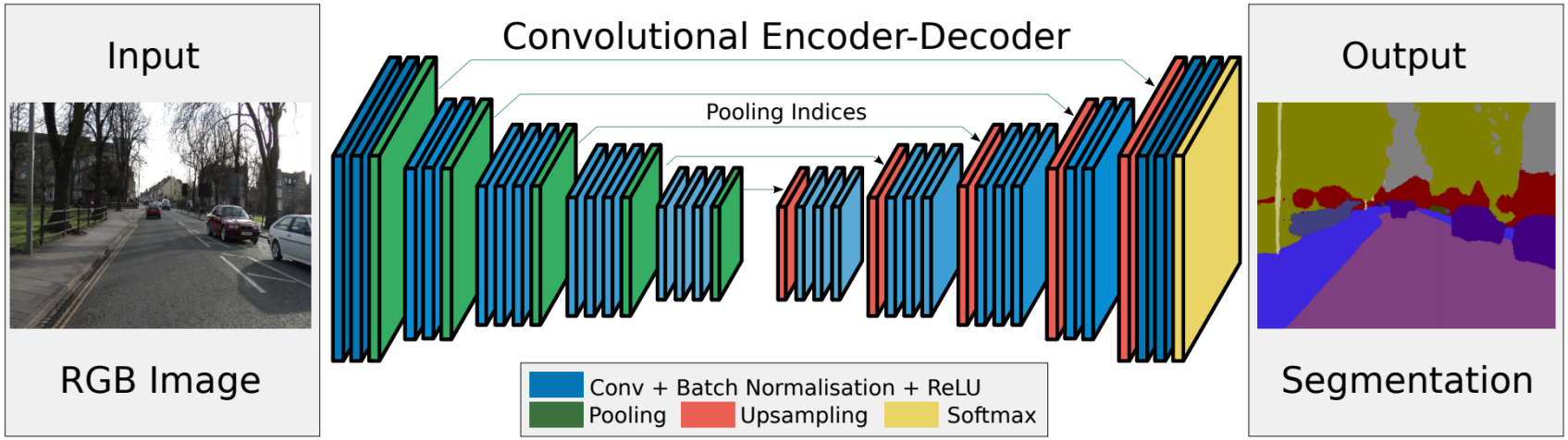
Autoencoders Application Example

Segmentation Problem



Semantic Segmentation Problem

SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

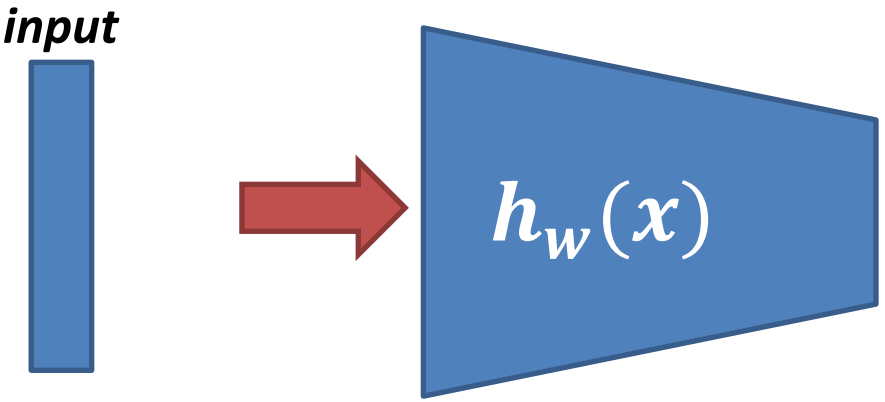


https://arxiv.org/pdf/1511.00561.pdf

Recurrent Neural Network

Book: https://www.deeplearningbook.org/contents/rnn.html

Feedfoward Neural Networks



1 dimensional signal

output:

$h_w(x)$= ***"cat"***

label: y = ''dog''

loss L: $f(\|\text{''dog''} - \text{''cat''}\|)$

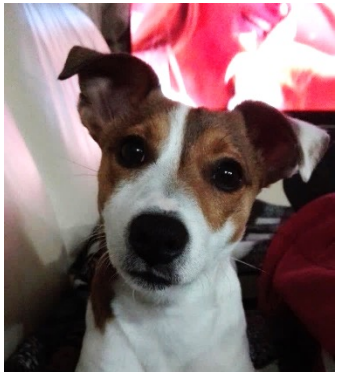
Parameters: w

Each weight contributed to the output → each weight contributed to the error

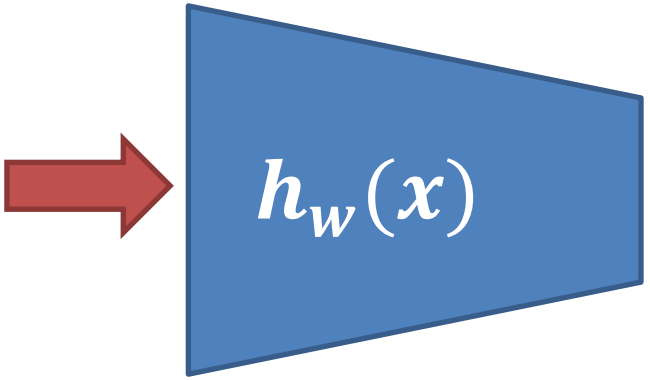
$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

Feedfoward Neural Networks

input



2D/3D signal



output:

$h_w(x)$= ***"cat"***

label: y = "dog"

loss L: $f(\|\text{"dog"} - \text{"cat"}\|)$

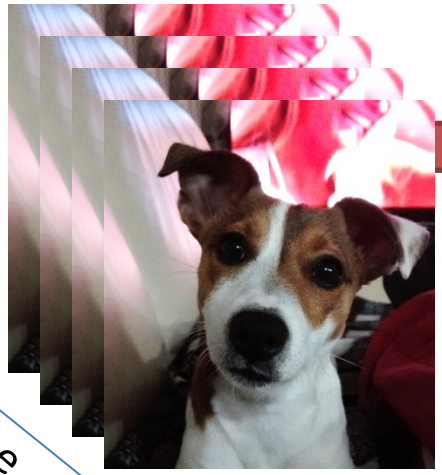
Parameters: w

Each weight contributed to the output → each weight contributed to the error

$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

Feedfoward Neural Networks

input



1D/2D/3D signal + Time (e.g., to capture motion)

$h_w(x)$

output:

$h_w(x)$= ***"cat"***

label: y = ''dog''

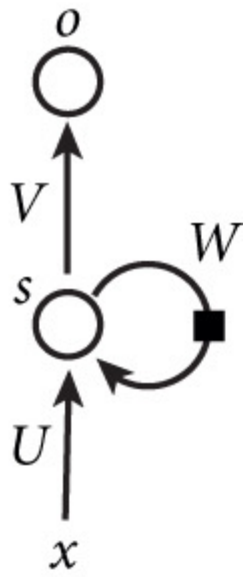
loss L: $f(\|\text{''dog''} - \text{''cat''}\|)$

Parameters: w

Each weight contributed to the output → each weight contributed to the error

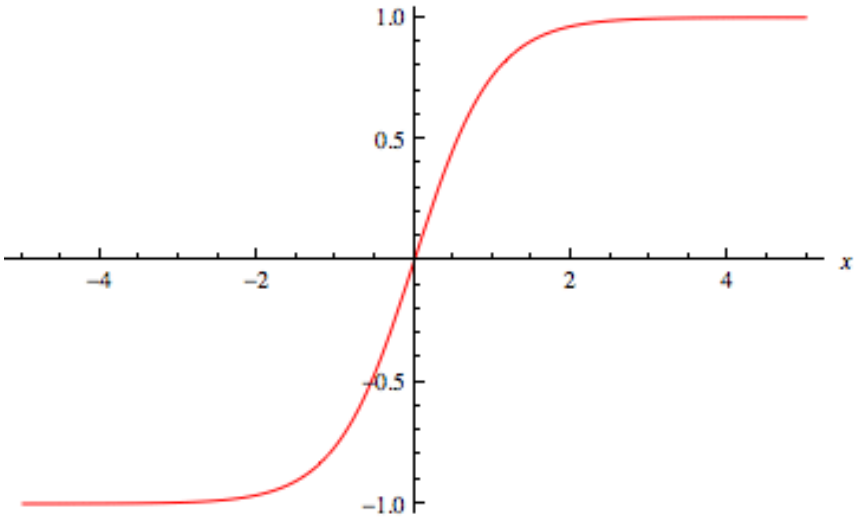
$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

Recurrent Neural Network

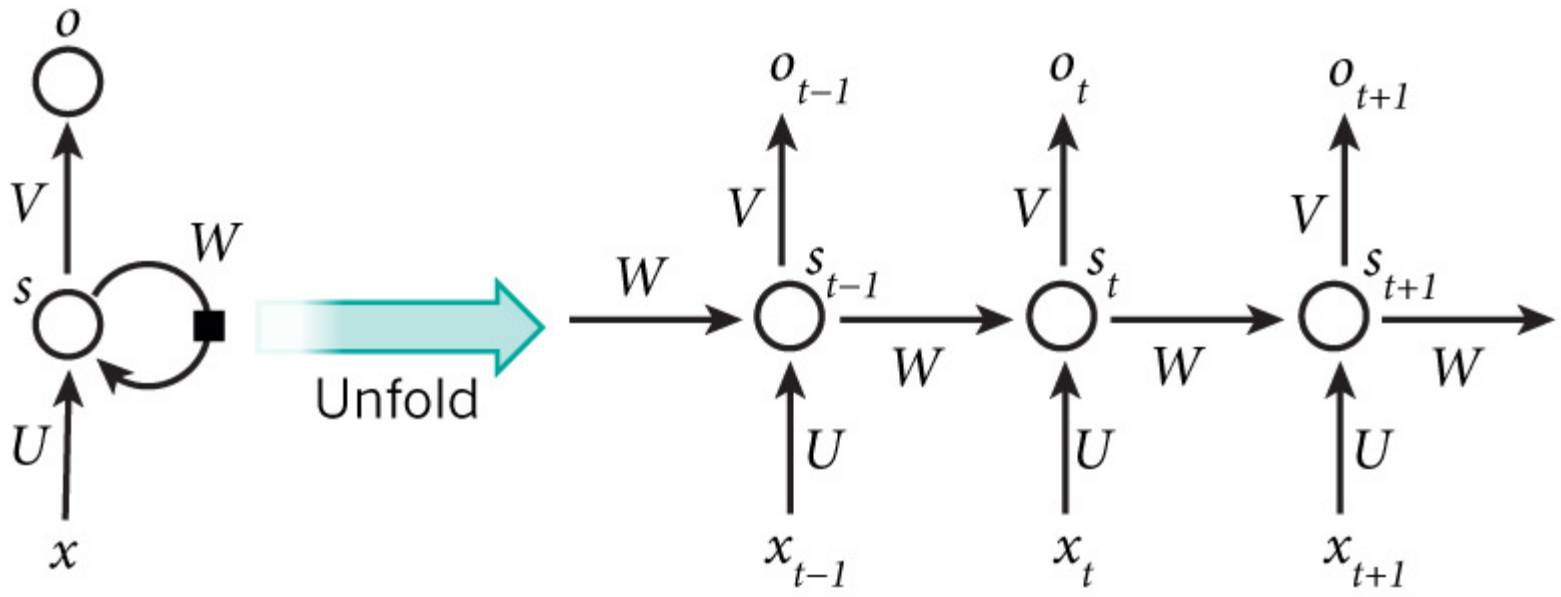


$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$f(x) = tanh(x)$$

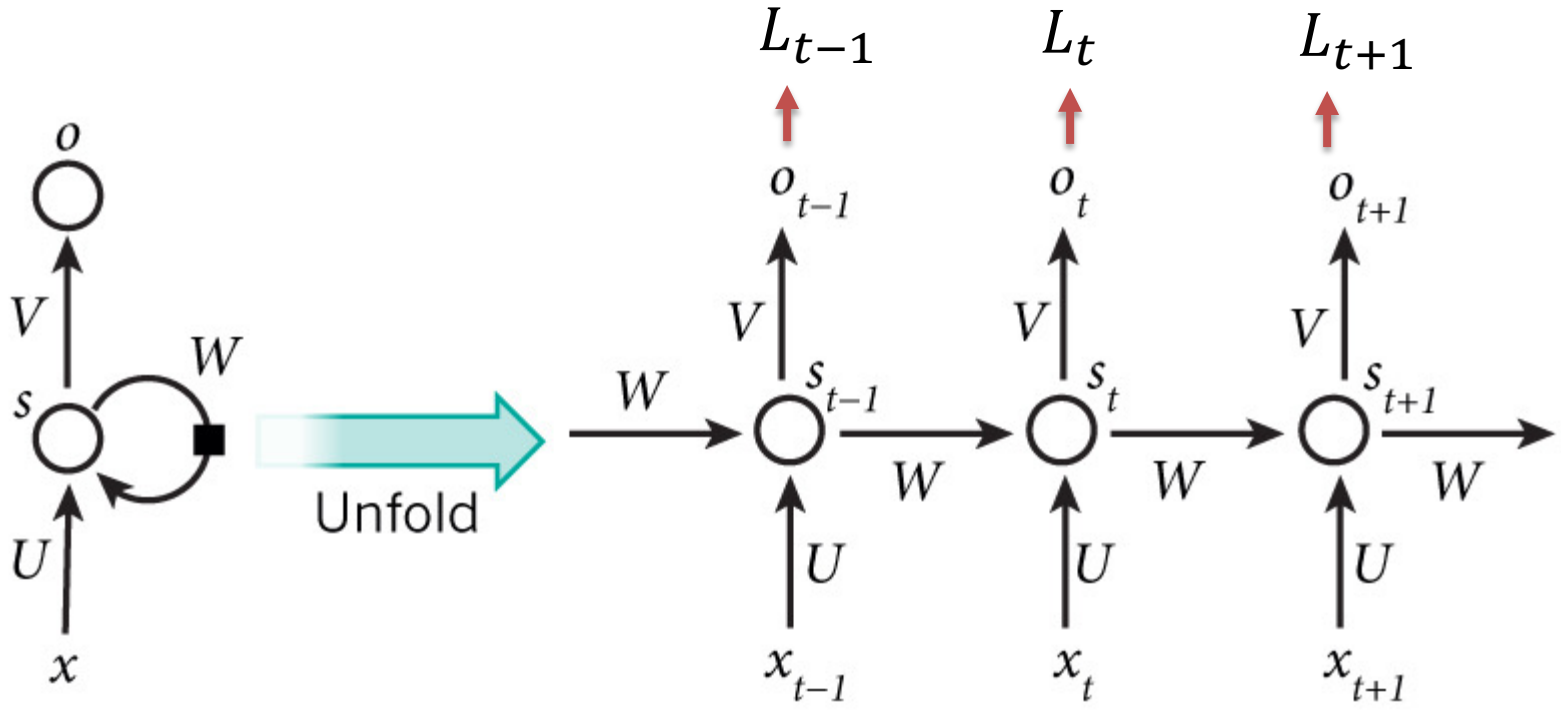


Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

Recurrent Neural Network



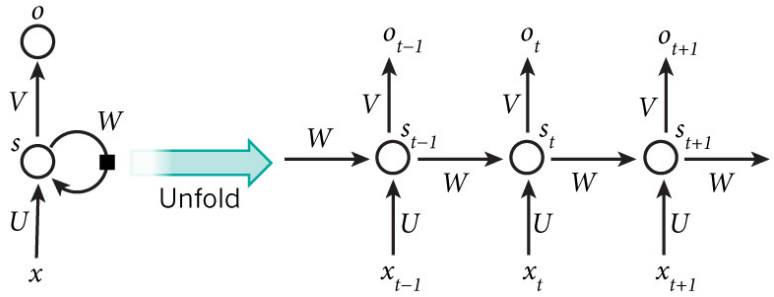
$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

Total Loss: $L = \sum_t^T L_t$

Recurrent Neural Network

(BackPropagation Through Time – BPTT)

1. Unroll the RNN

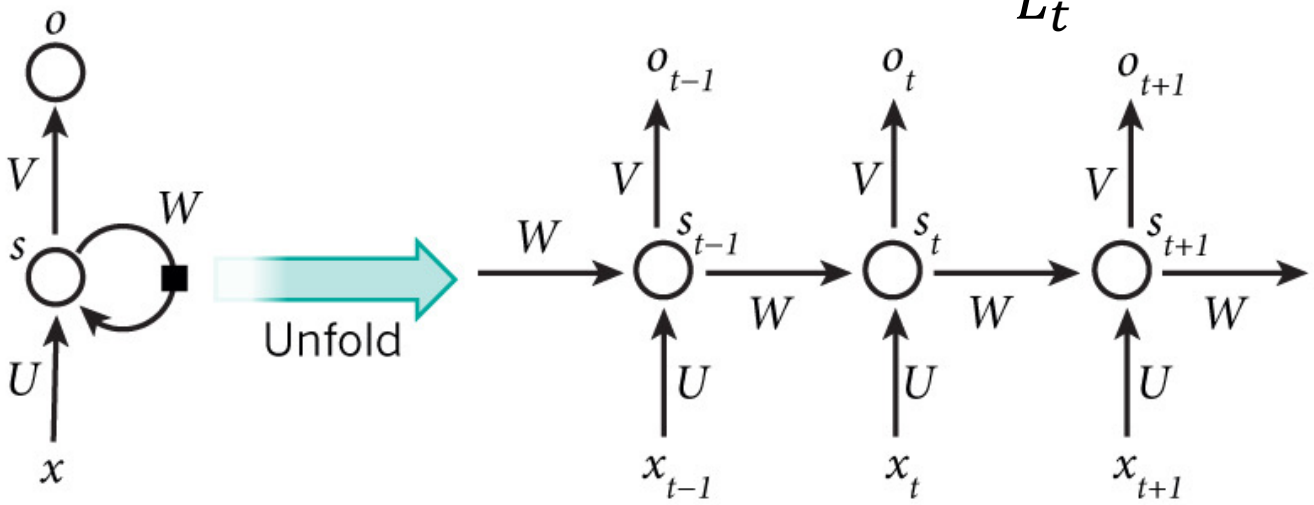


2. Define a Loss function

$$L = \sum_{t}^{T} L_t$$

3. Perform Gradient Descent (i.e., compute gradients wrt all weights)

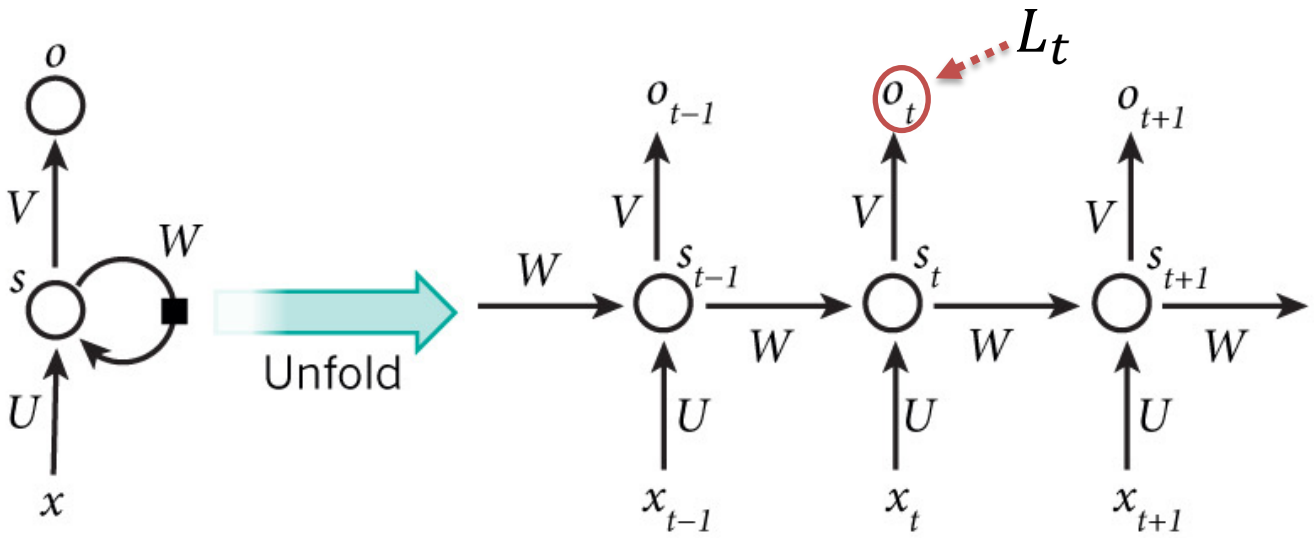
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = ?$$

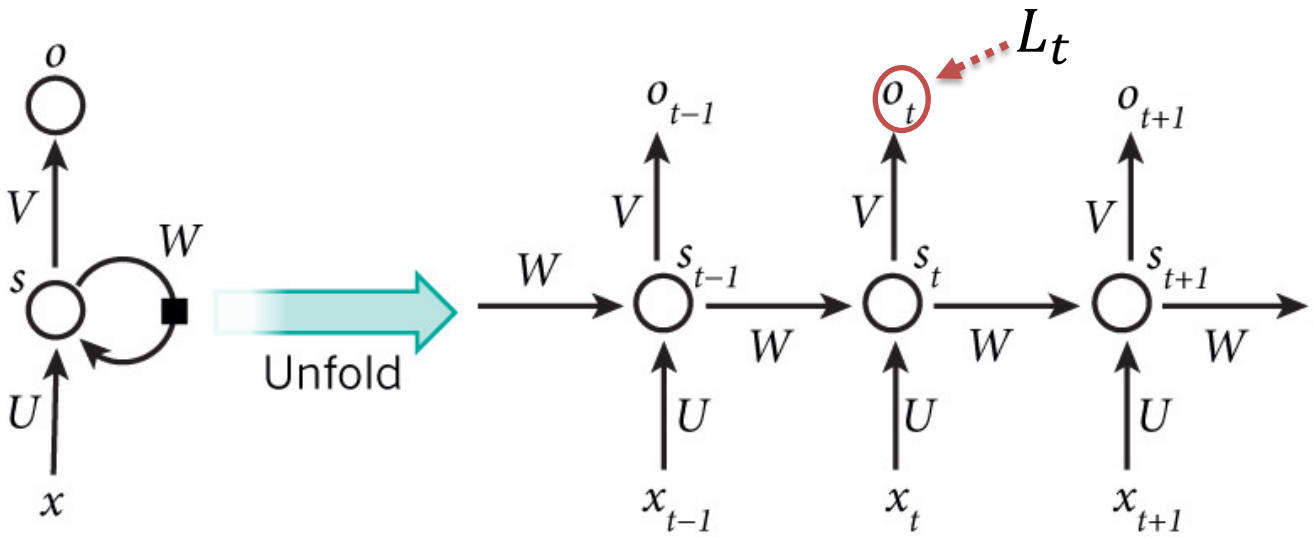
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = ?$$

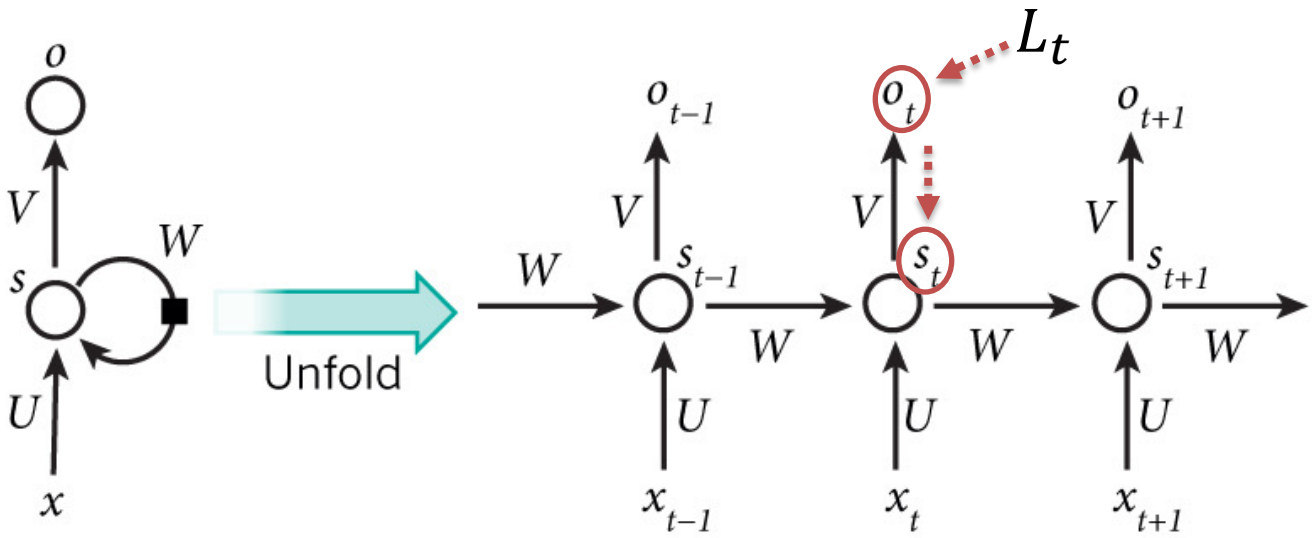
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t} \dots$$

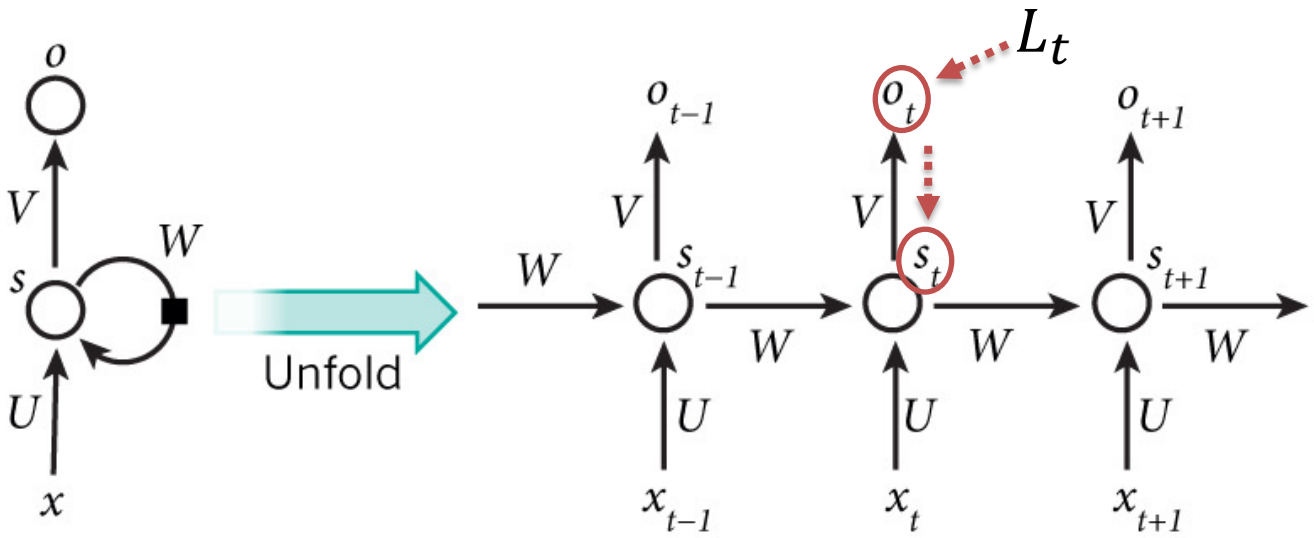
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t} \dots$$

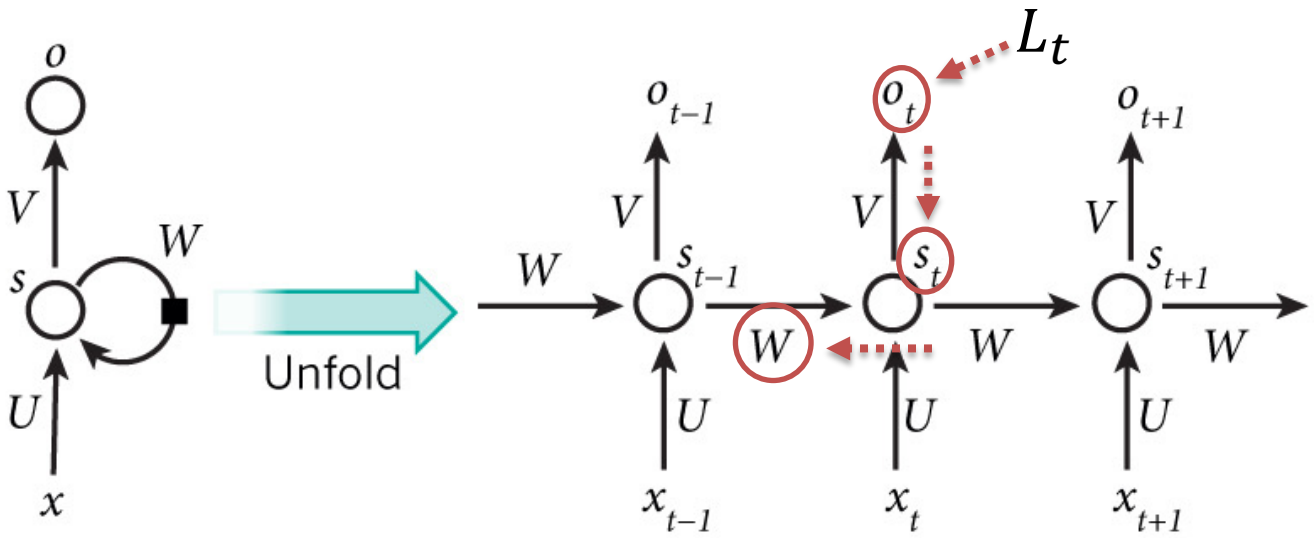
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t}\frac{\delta o_t}{\delta s_t} \dots$$

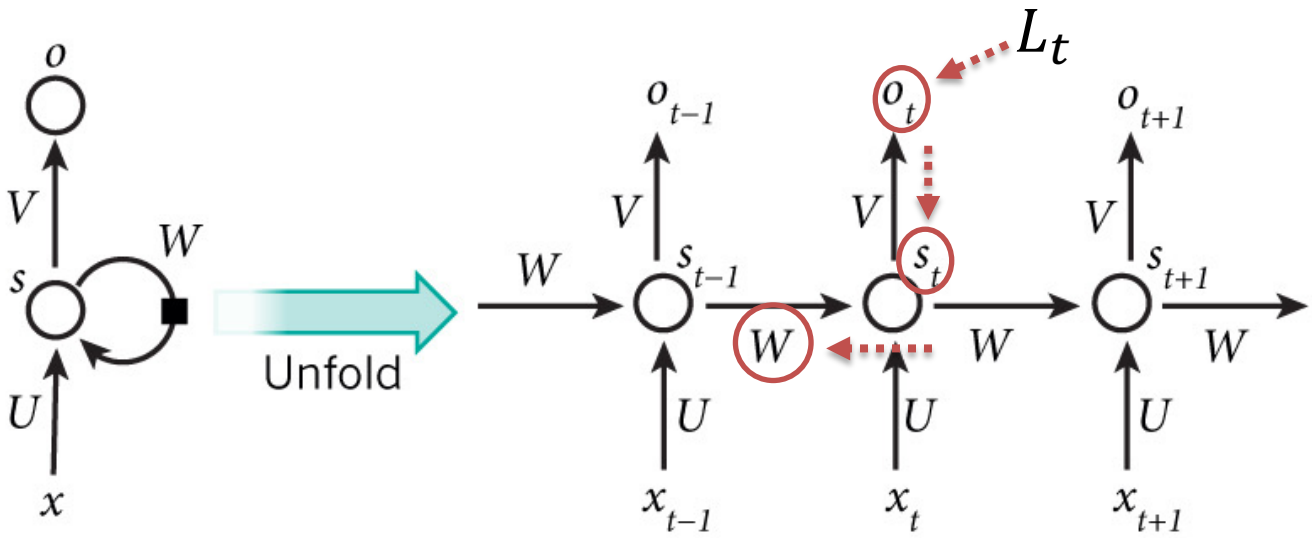
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t}\frac{\delta o_t}{\delta s_t} \dots$$

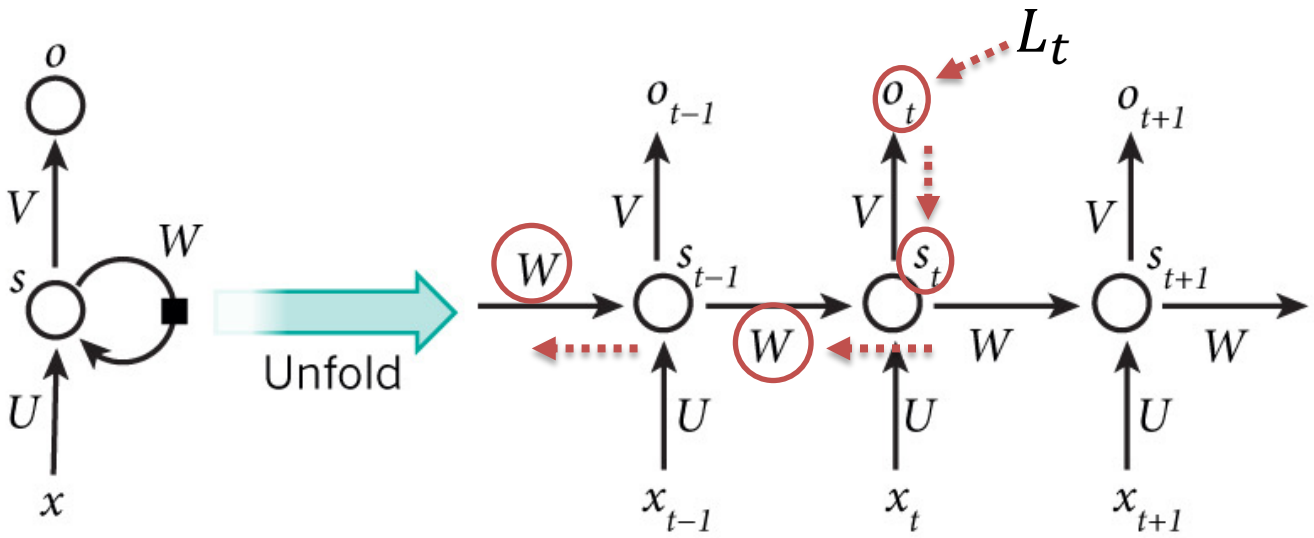
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t}\frac{\delta o_t}{\delta s_t}\frac{\delta s_t}{\delta W}$$

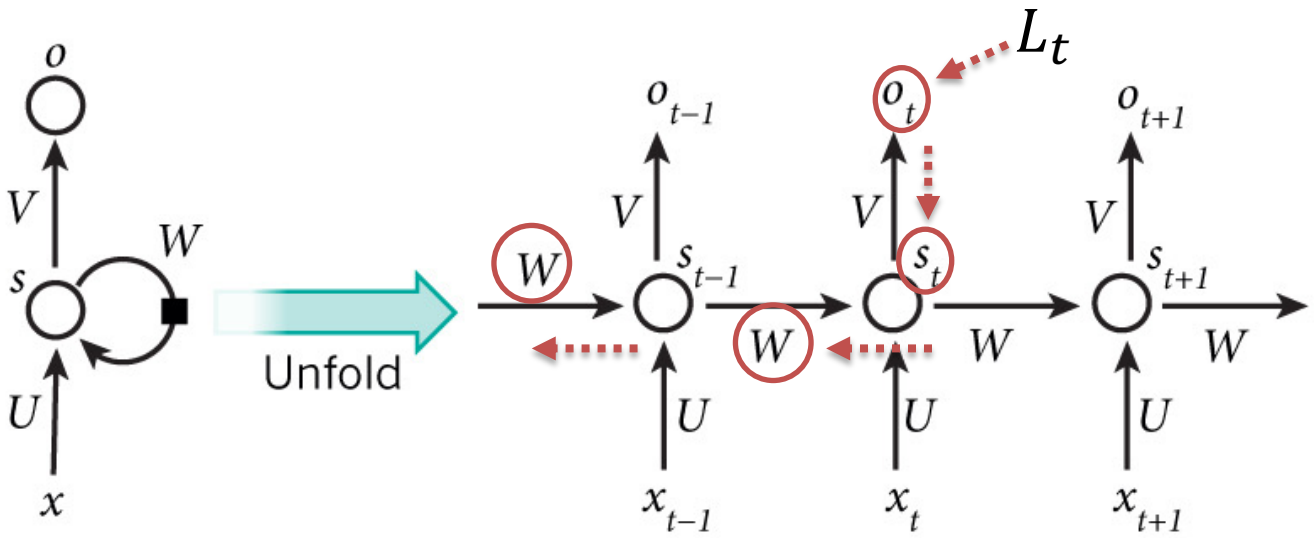
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

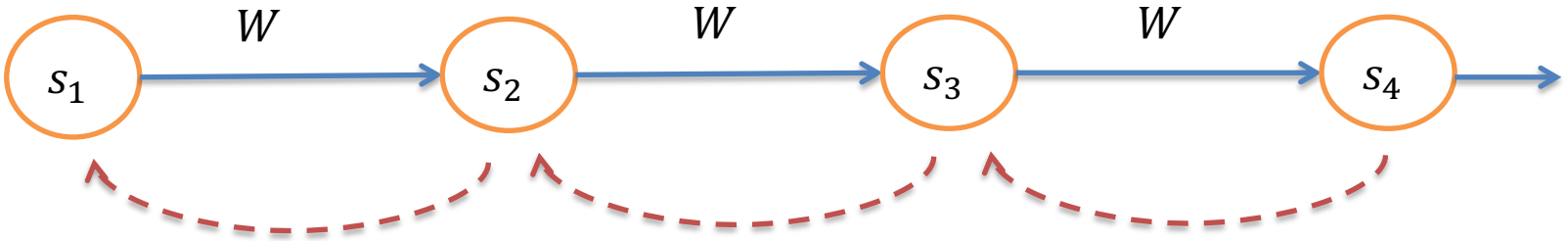
$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t}\frac{\delta o_t}{\delta s_t}\cancel{\frac{\delta s_t}{\delta W}}$$

Recurrent Neural Network

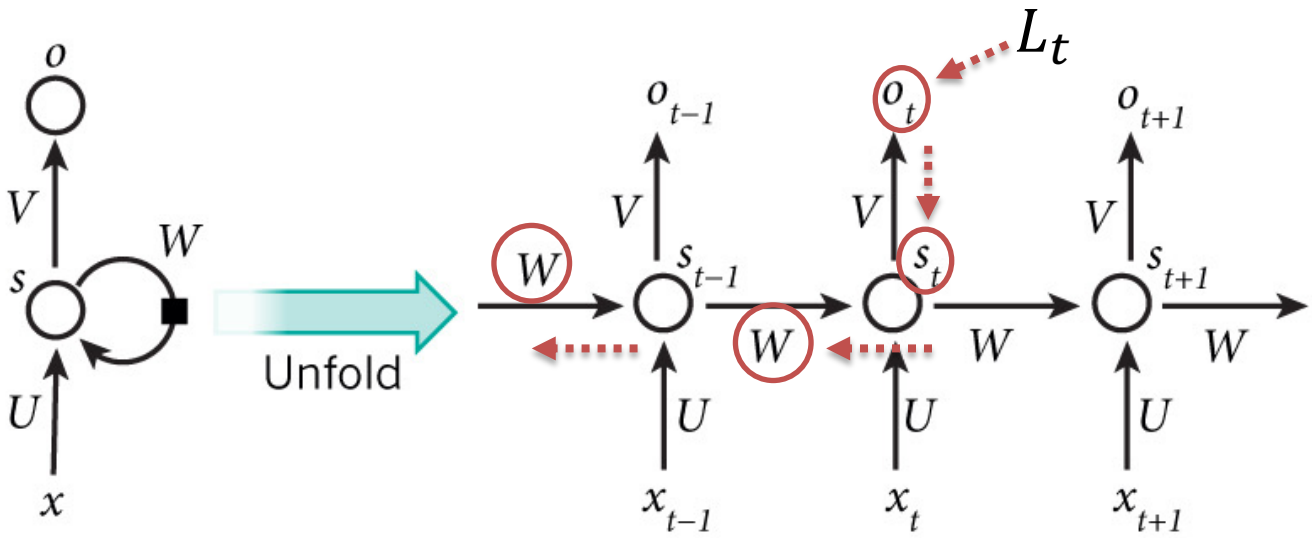


$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \frac{\delta L_t}{\delta o_t}\frac{\delta o_t}{\delta s_t}\cancel{\frac{\delta s_t}{\delta W}}$$



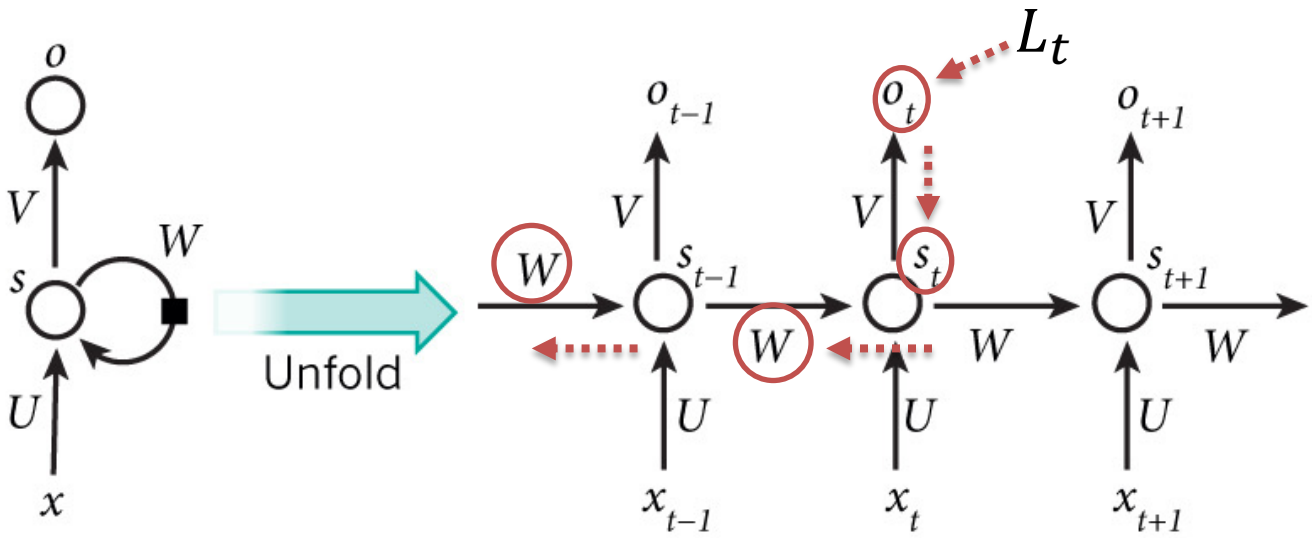
Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

Recurrent Neural Network

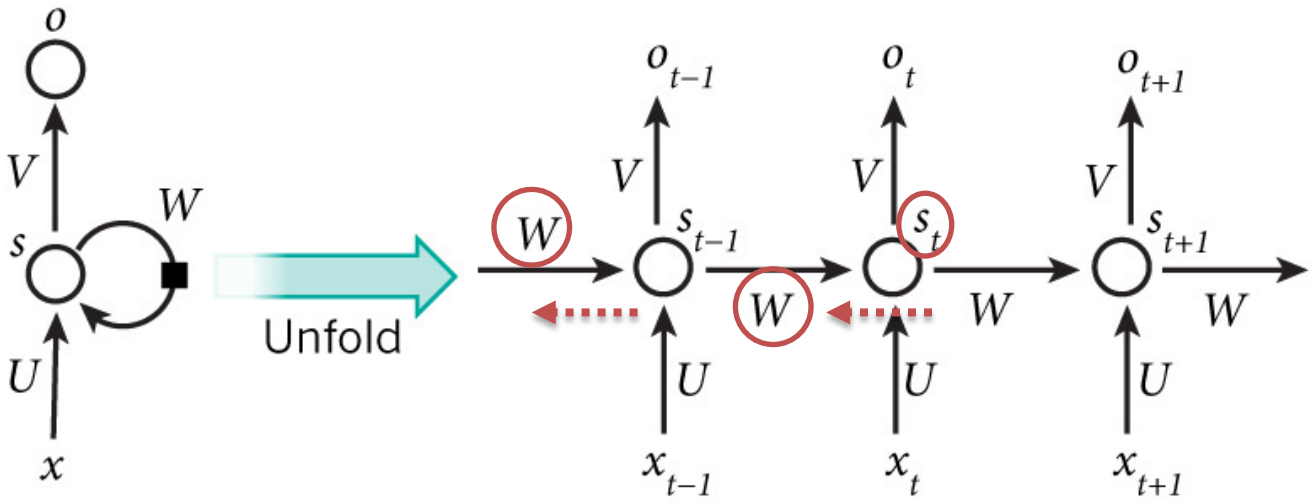


$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

when *t* >> *k* we are modelling a *long-term dependency*

Recurrent Neural Network

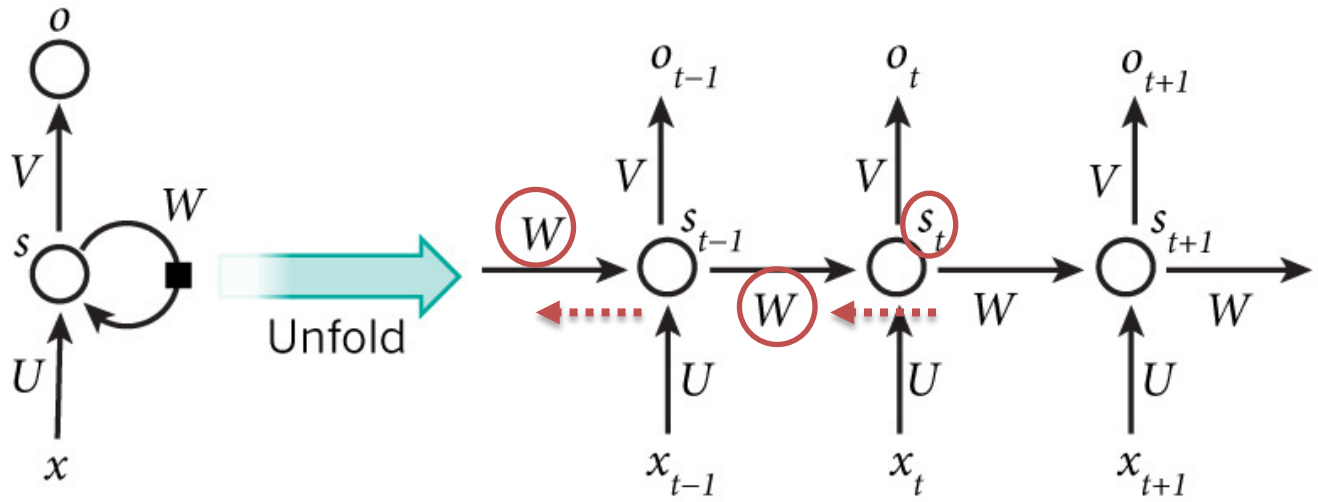


$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

is a chain rule itself!

Recurrent Neural Network

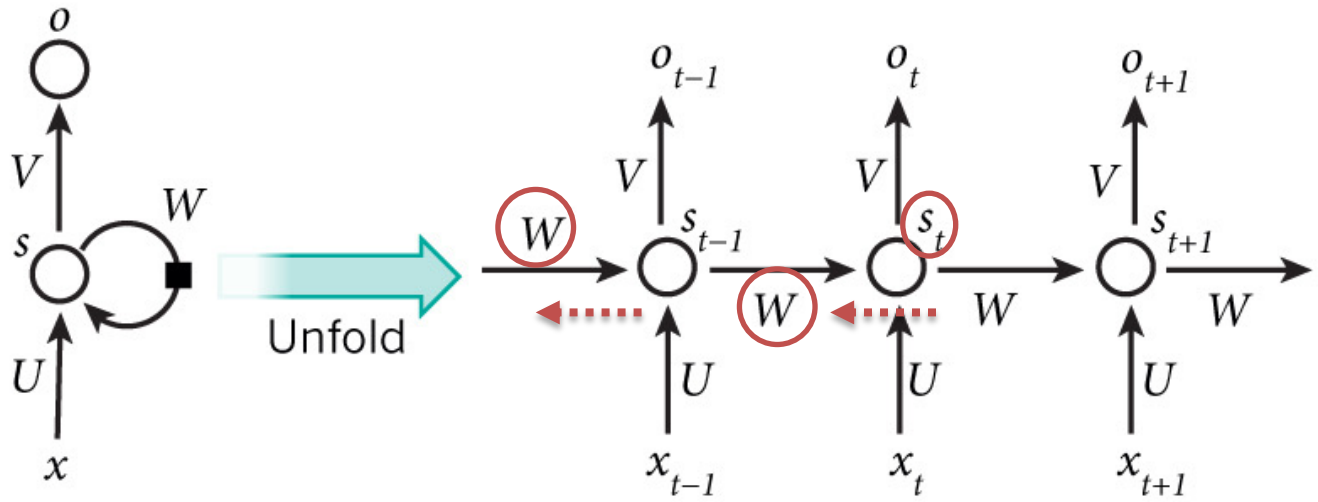


$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

$$\frac{\delta s_t}{\delta s_k} = \prod_{j=k+1}^{t} \frac{\delta s_j}{\delta s_{j-1}}$$

Recurrent Neural Network



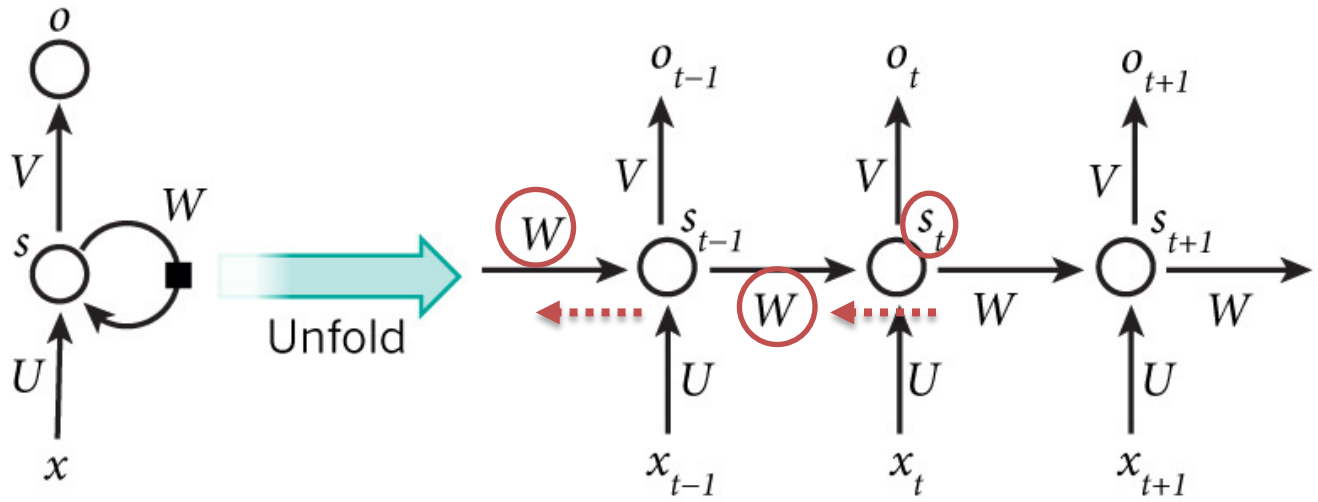
$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

$$\frac{\delta s_t}{\delta s_k} = \prod_{j=k+1}^{t} \frac{\delta s_j}{\delta s_{j-1}}$$

Product of (t-k) matrices whose norm is <1

Recurrent Neural Network



$$s_t = f(Ux_t + Ws_{t-1})$$
$$o_t = Vs_t$$

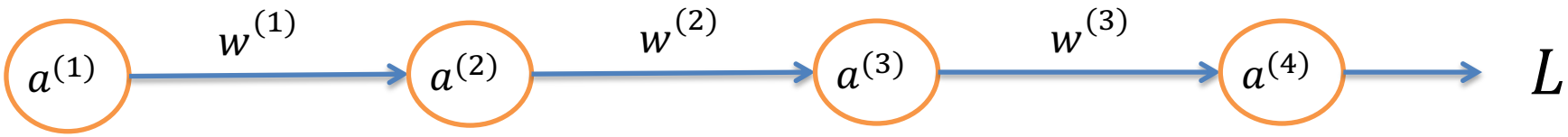
$$\frac{\delta L_t}{\delta W} = \sum_{k=0}^{t} \frac{\delta L_t}{\delta o_t} \frac{\delta o_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W}$$

$$\frac{\delta s_t}{\delta s_k} = \prod_{j=k+1}^{t} \frac{\delta s_j}{\delta s_{j-1}}$$

Product of (t-k) matrices whose norm is <1

Gradient vanishing problem

Gradient vanishing problem Backpropagation

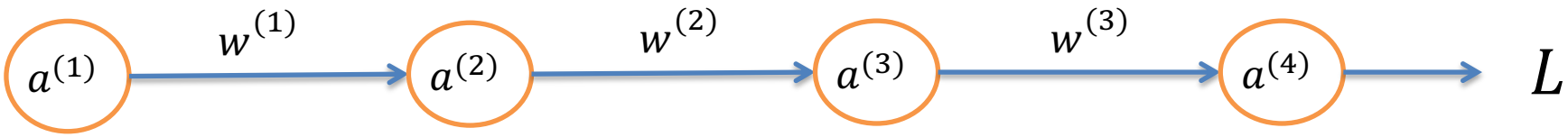


$$\frac{\delta L}{\delta w^{(1)}} = ???$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation



$$\frac{\delta L}{\delta w^{(1)}} = ???$$

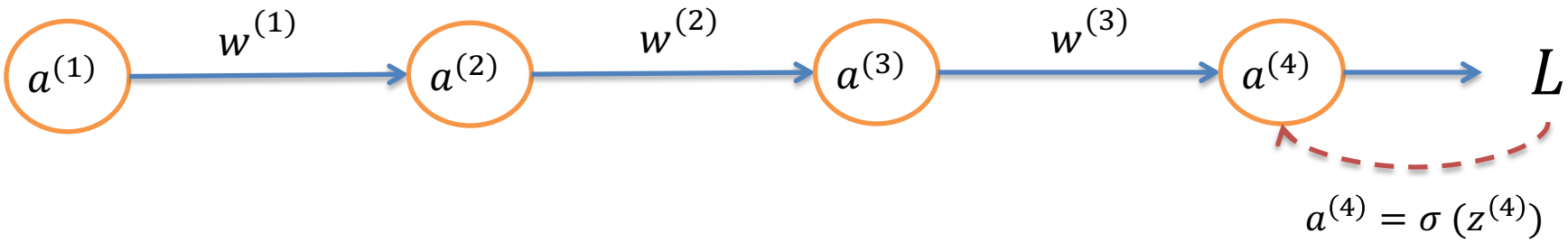
Chain rule

$$(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation



$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \dots$$

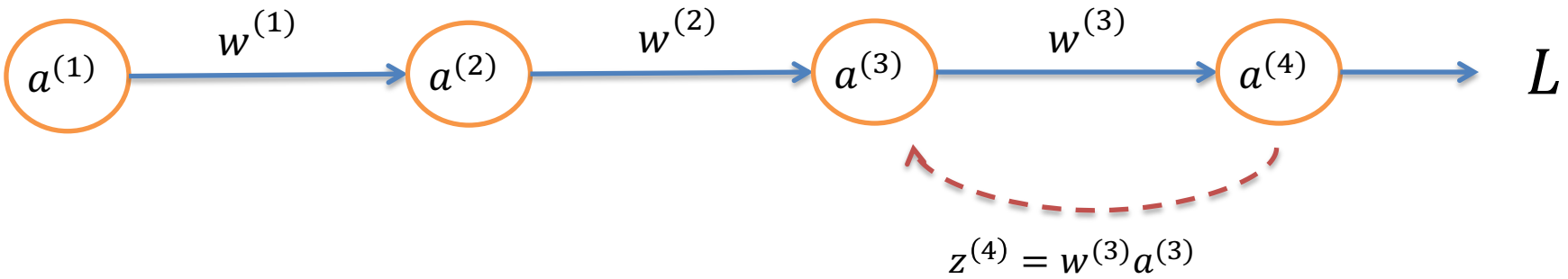
Chain rule

$$(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation



$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \dots$$

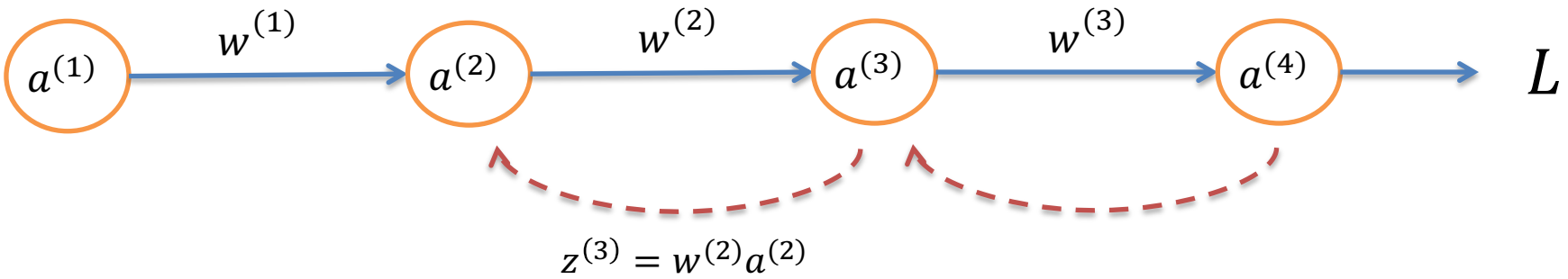
Chain rule

$$(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation



$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \dots$$

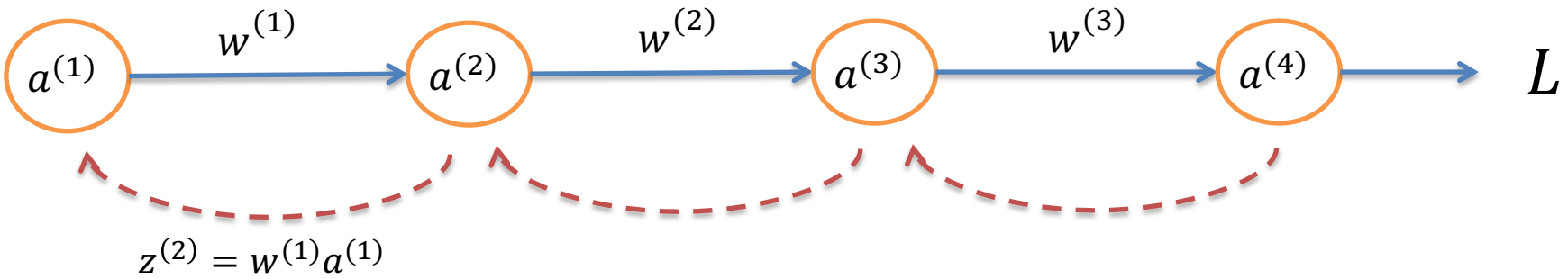
Chain rule

$$(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation

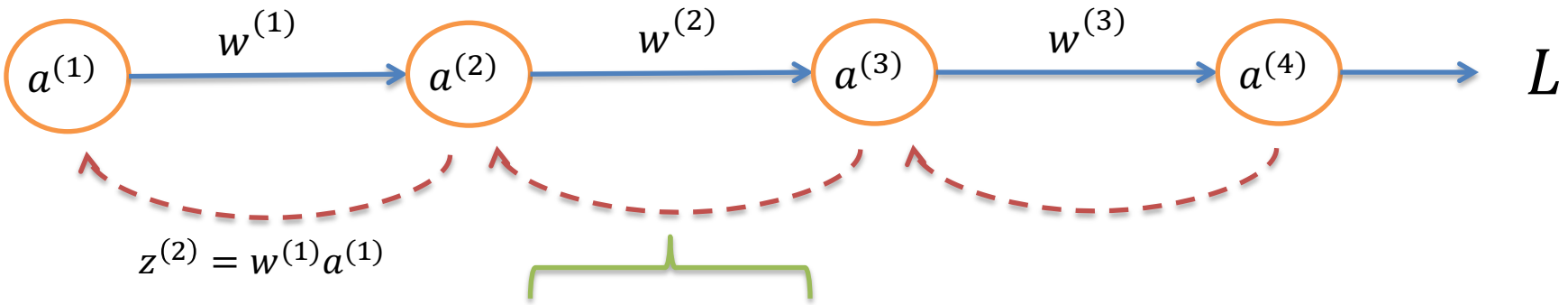


$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w^{(1)}}$$

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

σ is the sigmoid function
w are our parameters

Gradient vanishing problem Backpropagation

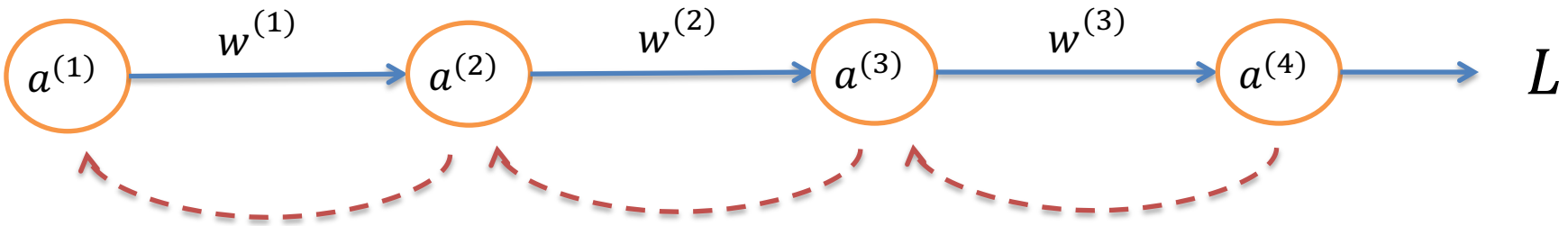


$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w^{(1)}}$$

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

σ is sigmoid function
w are our parameters

Gradient vanishing problem
Backpropagation



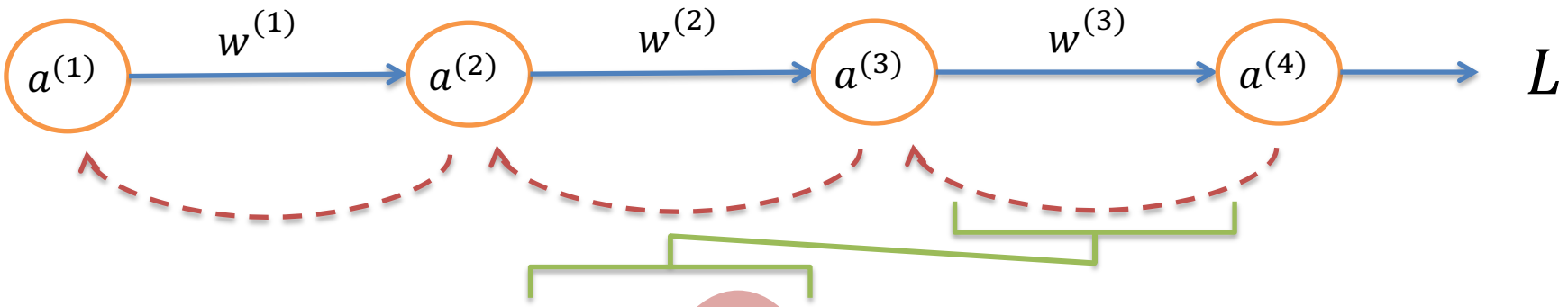
$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w^{(1)}}$$

$$\sigma'(z^{(4)})$$

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

$$\frac{\delta a^{(l)}}{\delta z^{(l)}} = \sigma'(z^{(l)})$$

Gradient vanishing problem Backpropagation



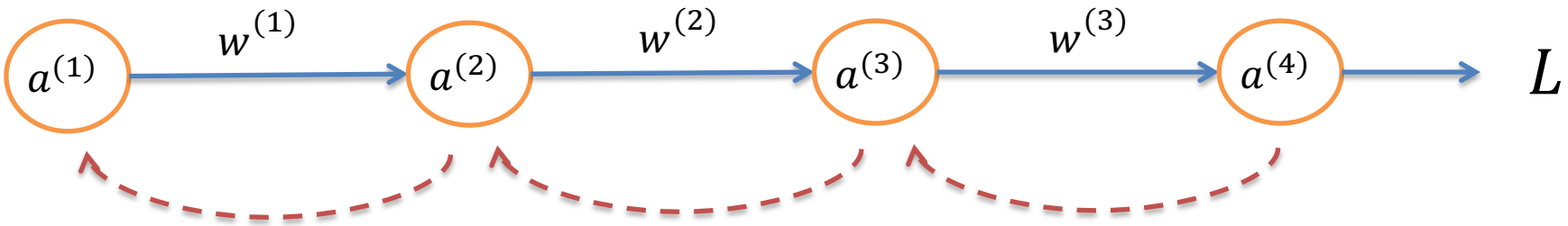
$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w^{(1)}}$$

$$\sigma'(z^{(4)}) \times w^{(3)}$$

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

$$\frac{\delta z^{(l)}}{\delta a^{(l-1)}} = w^{(l-1)}$$

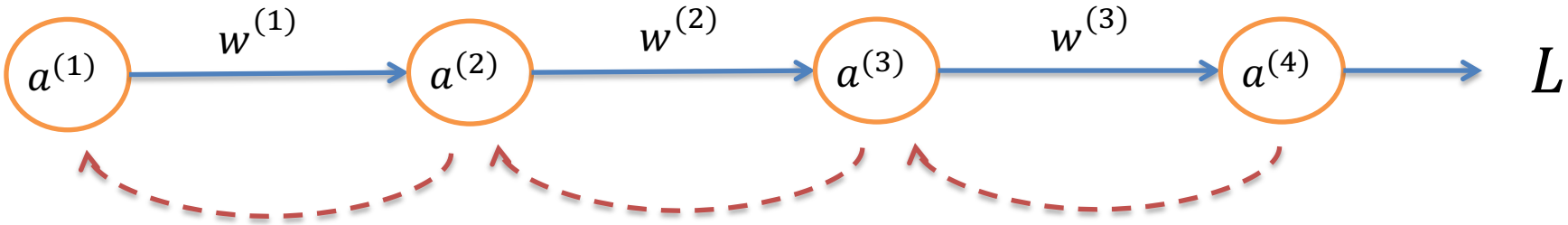
Gradient vanishing problem Backpropagation



$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \frac{\delta a^{(4)}}{\delta z^{(4)}} \frac{\delta z^{(4)}}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta w^{(1)}}$$

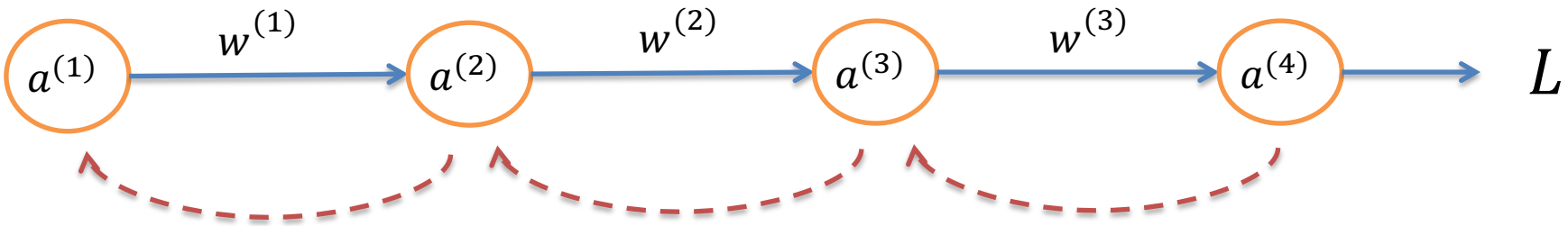
$$\frac{\delta L}{\delta a^{(4)}} \times \sigma'(z^{(4)}) \times w^{(3)} \times \sigma'(z^{(3)}) \times w^{(2)} \times \sigma'(z^{(2)}) \times a^{(1)}$$

Gradient vanishing problem Backpropagation

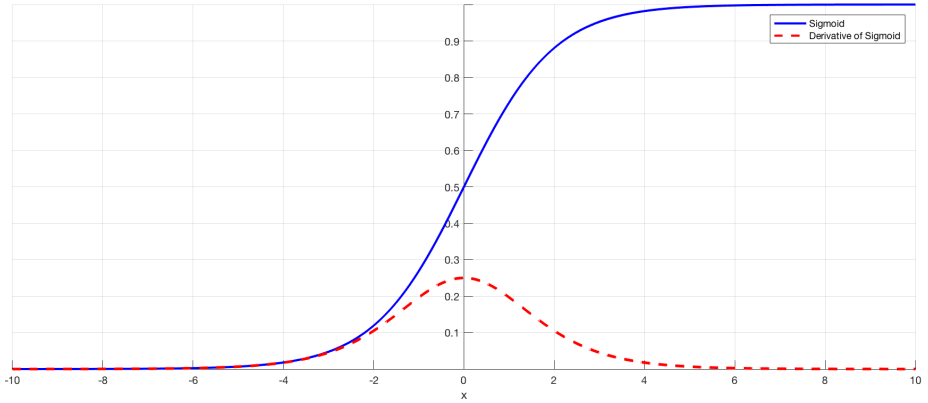


$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \times \sigma'(z^{(4)}) \times w^{(3)} \times \sigma'(z^{(3)}) \times w^{(2)} \times \sigma'(z^{(2)}) \times a^{(1)}$$

Gradient vanishing problem Backpropagation

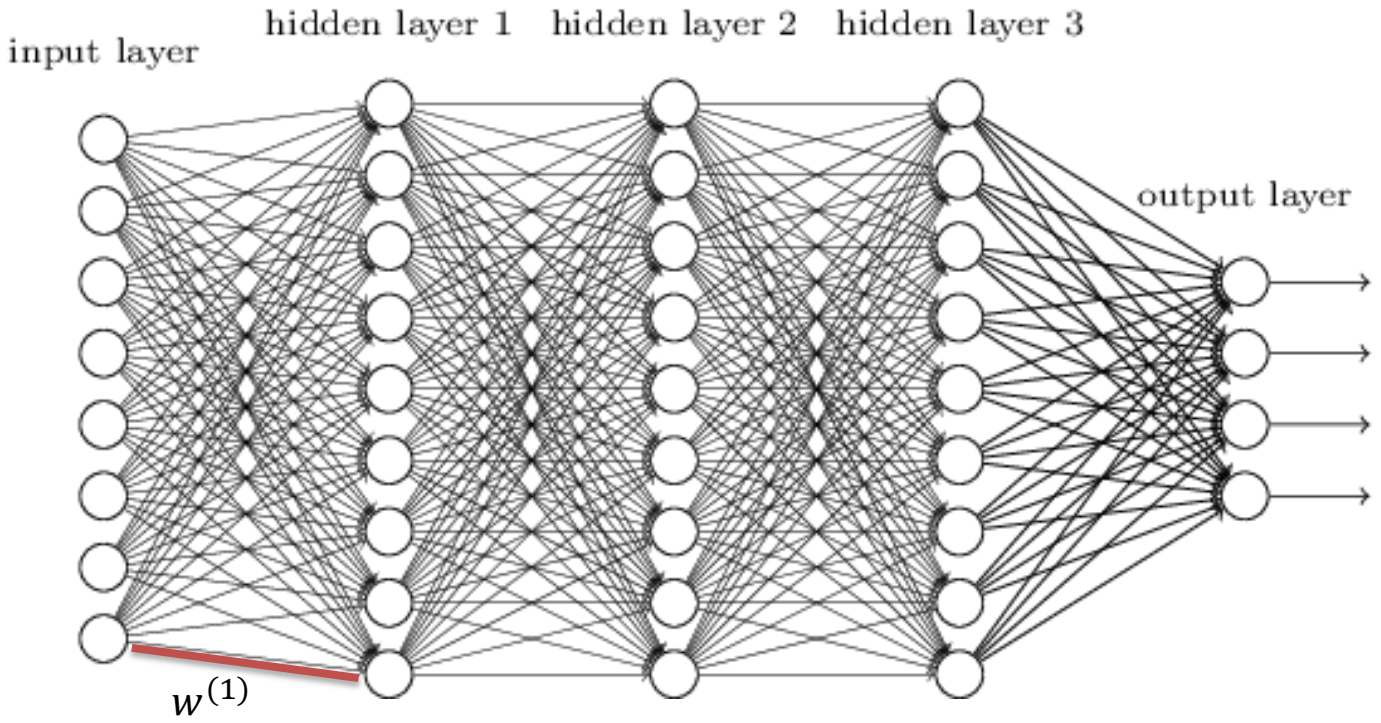


$$\frac{\delta L}{\delta w^{(1)}} = \frac{\delta L}{\delta a^{(4)}} \times \underbrace{\sigma'(z^{(4)})}_{\leq 0.25} \times w^{(3)} \times \underbrace{\sigma'(z^{(3)})}_{\leq 0.25} \times w^{(2)} \times \underbrace{\sigma'(z^{(2)})}_{\leq 0.25} \times a^{(1)}$$



$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

Gradient vanishing problem Backpropagation

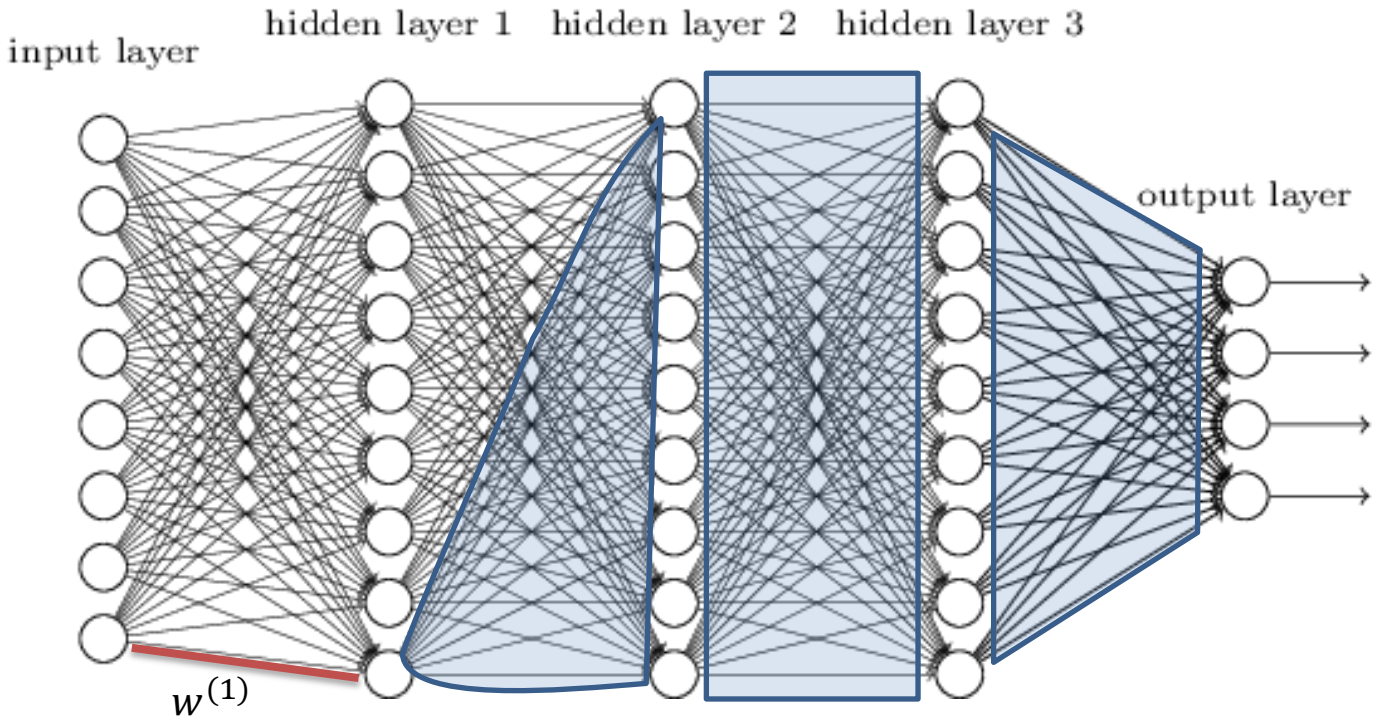


L

$$\frac{\delta L}{\delta w^{(1)}} = ???$$

$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

Gradient vanishing problem Backpropagation



L

$$\frac{\delta L}{\delta w^{(1)}} = ???$$

$$w_{new} = w_{old} - \alpha \frac{\delta L}{\delta w}$$

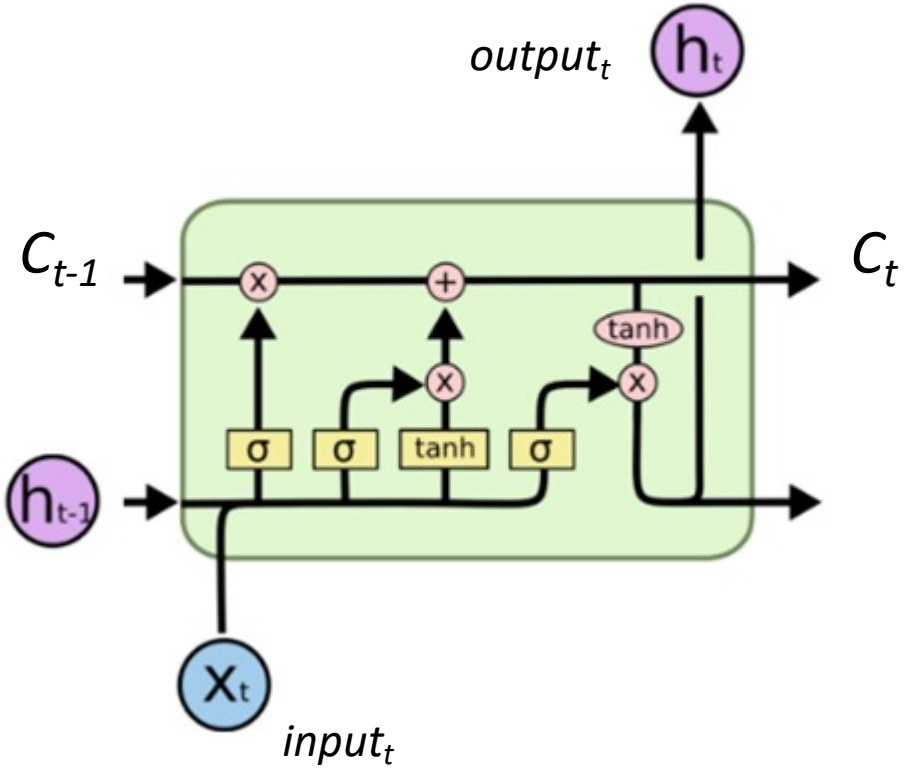
Long-Short Term Memory Networks

Dealing with Gradient Vanishing Problem

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780. (The original paper on LSTMs; the forget gate was added later)
https://direct.mit.edu/neco/article/9/8/1735/6109/Long-Short-Term-Memory

Long-Short Term Memory (LSTM)

LSTM are a special kind of RNN, capable of learning long-term dependencies and explicitly designed to avoid the long-term dependency problem, remembering information for long periods of time.

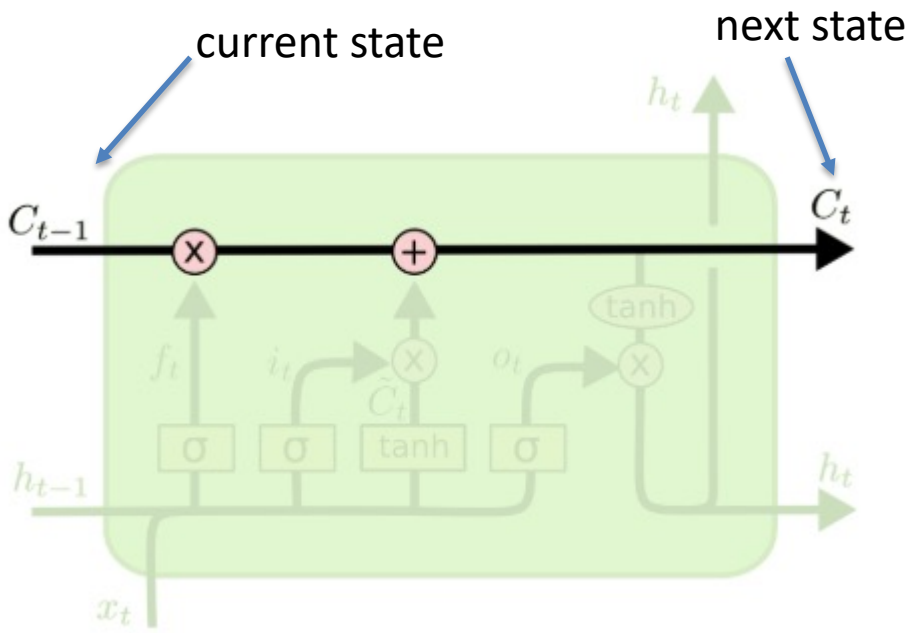


LSTM block:

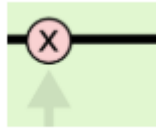
- memory **cell state** C_t
- gate layers
- non-linear layers
- pointwise operations

Long-Short Term Memory (LSTM)

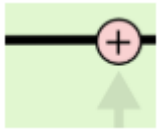
Cell state: the key to LSTMs is the cell state C_t.
The cell state **is passed from a timestep to the following one** after only some minor linear interactions. It's very easy for information to just flow along it unchanged.



Determines what to forget or keep from the current state

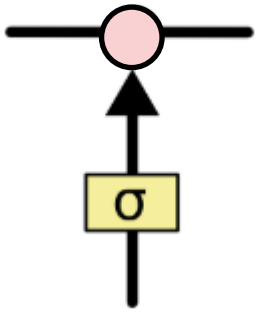


Determines how to update the current state given a new input



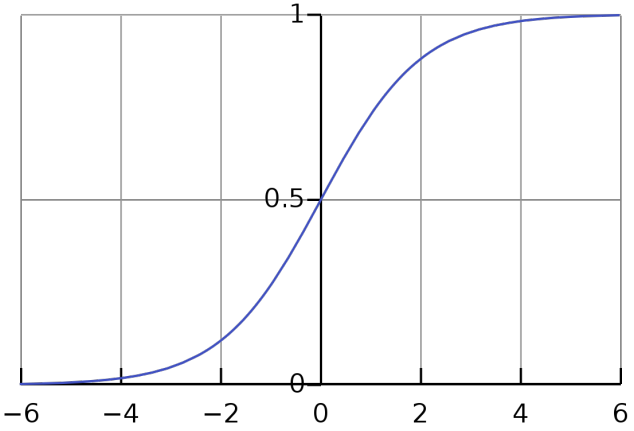
Long-Short Term Memory (LSTM)

Gate units: LSTM have the ability to **remove or add information to the cell state, carefully regulated by structures called gates**.
Gates are a way to optionally let information through. They are composed out of a sigmoid layer and a pointwise multiplication operation.



An LSTM has **three of these gates**, to protect and control the cell state:

- **Forget gate**
- **Input gate**
- **Output gate**

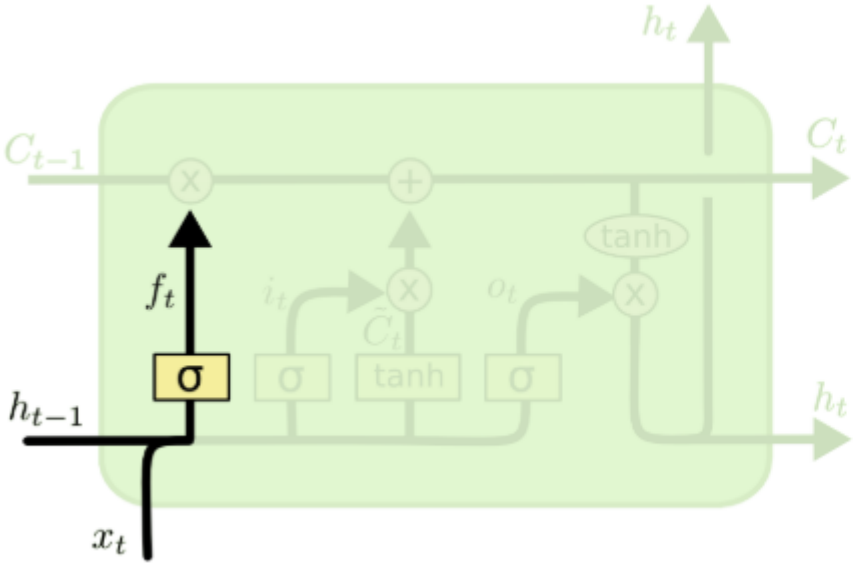


σ *is the sigmoid function*

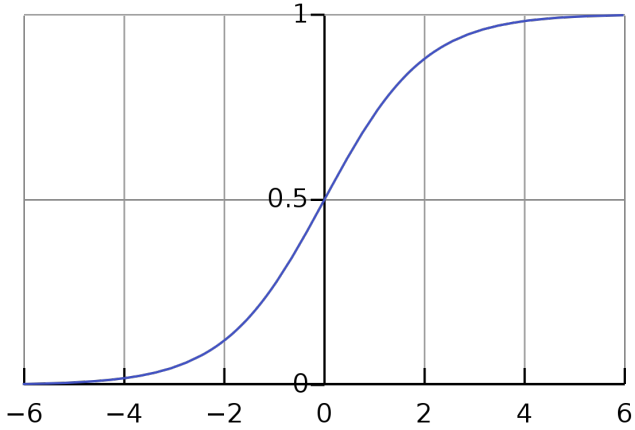
X + pointwise operations

Long-Short Term Memory (LSTM)

Forget gate layer: the first step in our LSTM is to **decide what information we're going to throw away from the cell state**. This decision is made by a sigmoid layer called "forget gate layer." It looks at the previous output h_{t-1} and the current input x_t, and outputs a number between 0 and 1 for any number in the cell state C_{t-1}.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

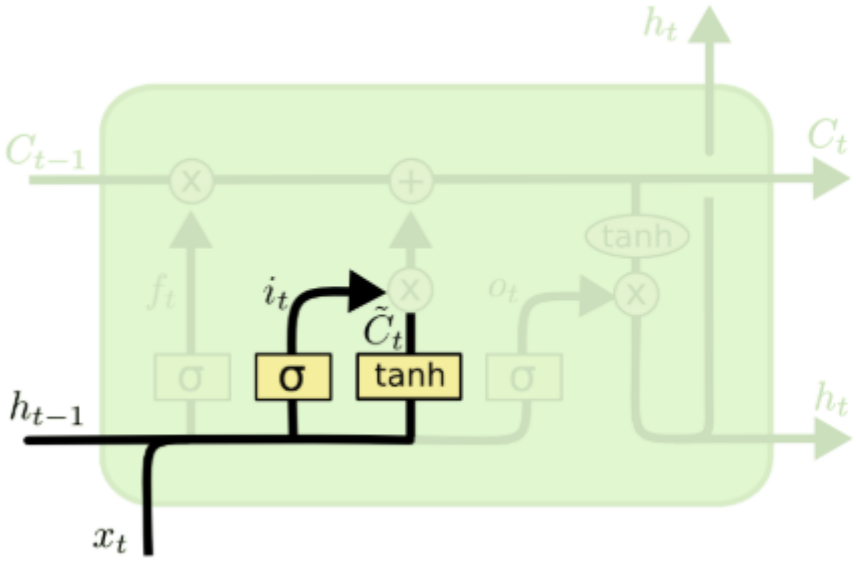


Long-Short Term Memory (LSTM)

Input gate layer: the next step is to **decide what new information (i.e., input) we're going to store in the cell state.**

This has two parts:

1. **a sigmoid layer called the "input gate layer"** decides if there is a need to update.
2. **a tanh layer creates a vector of new candidate representation, $\tilde{C}_t$, that could be added to the state**.

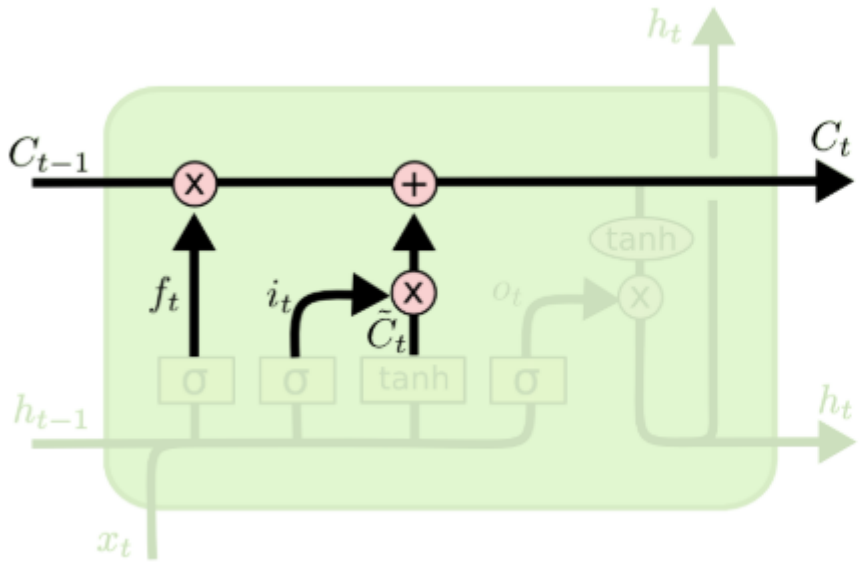


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long-Short Term Memory (LSTM)

Cell update



Cell content

New information

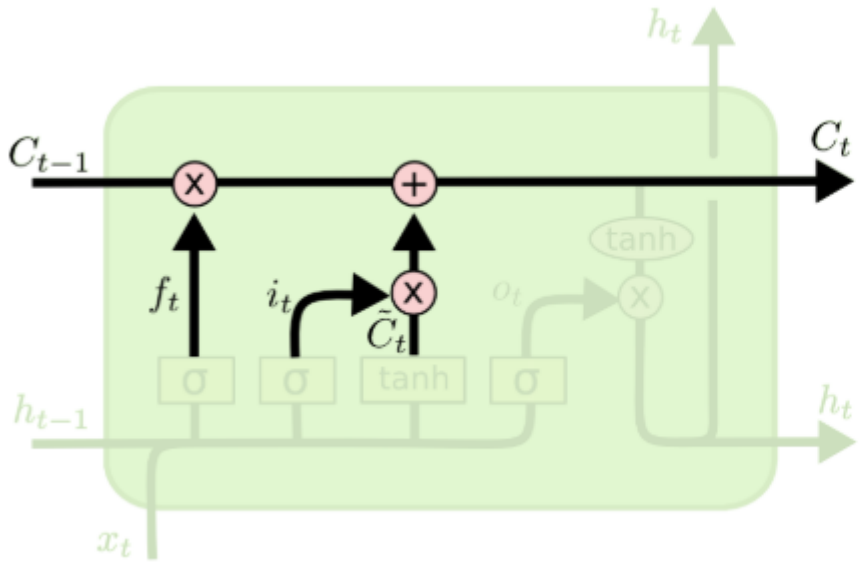
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Forget gate

Input gate

Long-Short Term Memory (LSTM)

Cell update



Cell content

New information

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

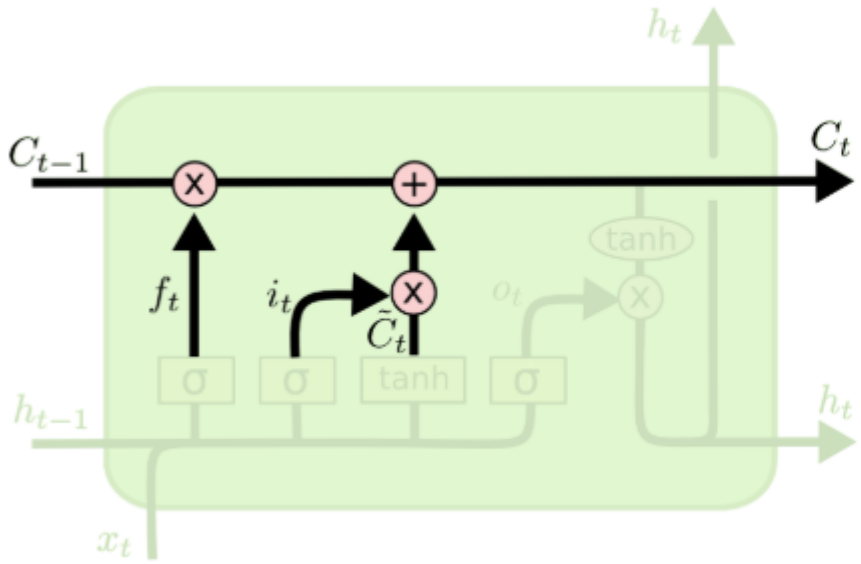
Forget gate

Input gate

Forget irrelevant parts of previous state

Long-Short Term Memory (LSTM)

Cell update



Cell content

New information

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Forget gate

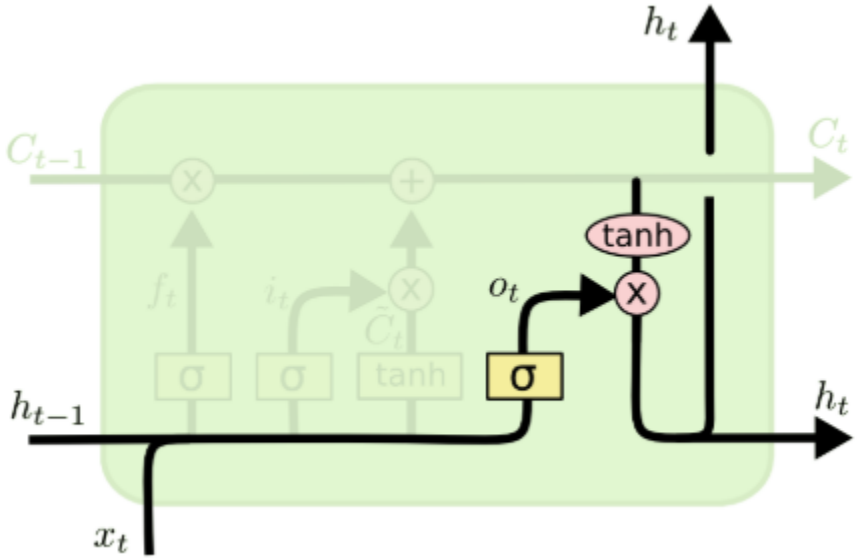
Input gate

Selectively update the cell state value

Long-Short Term Memory (LSTM)

LSTM Output: finally, we need to **decide what we're going to output**. This output will be based on the cell state, but will be a filtered version.

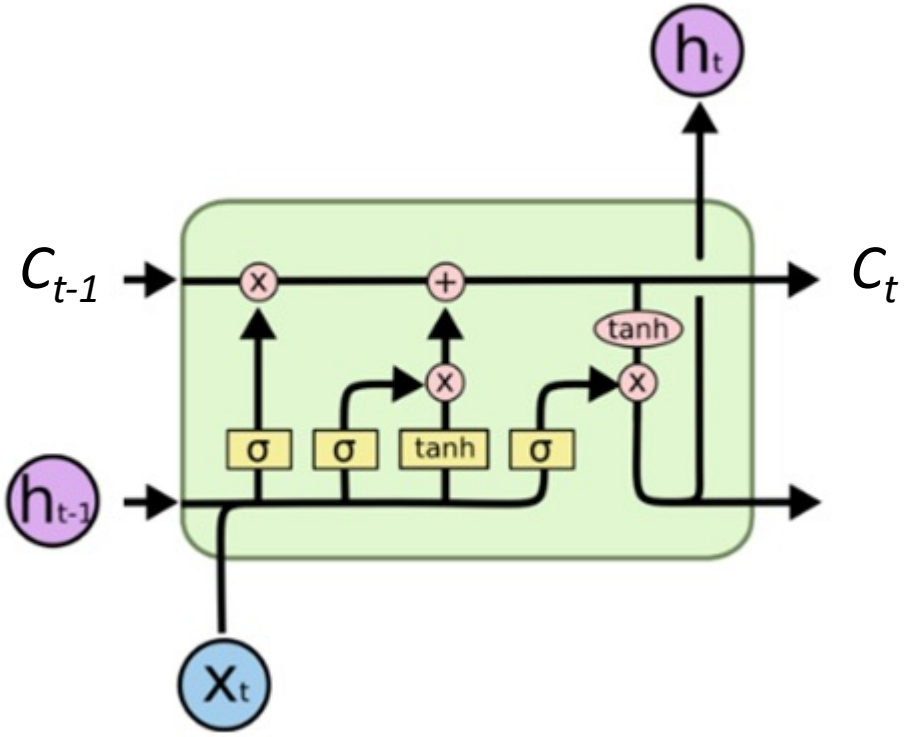
The output gate decides what parts of the cell state we're going to output.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long-Short Term Memory (LSTM)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

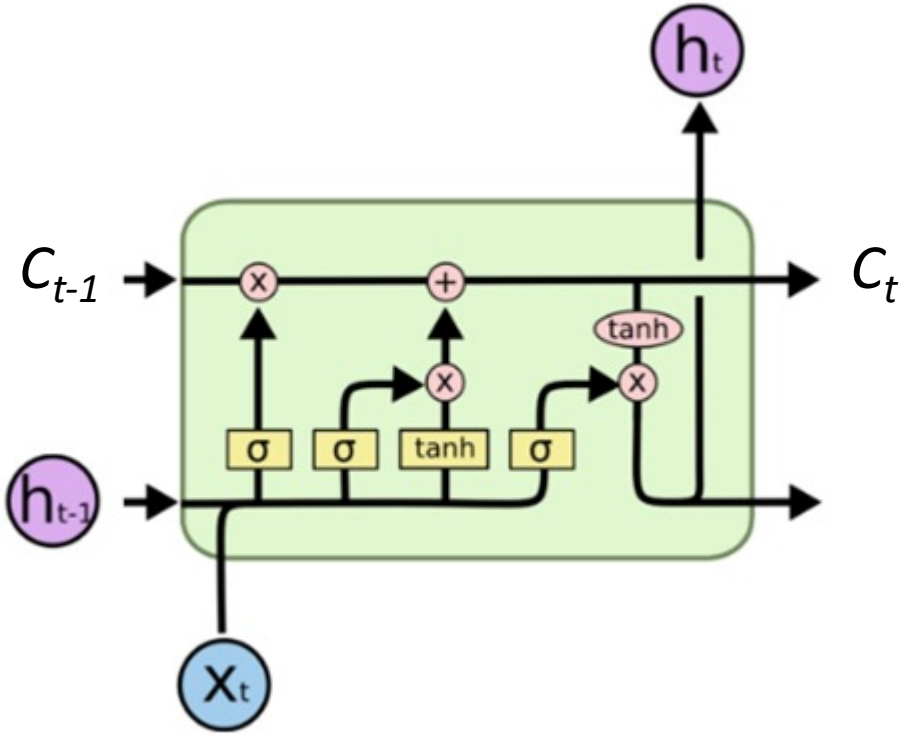
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long-Short Term Memory (LSTM)



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right)$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * \tanh\left(C_t\right)$$

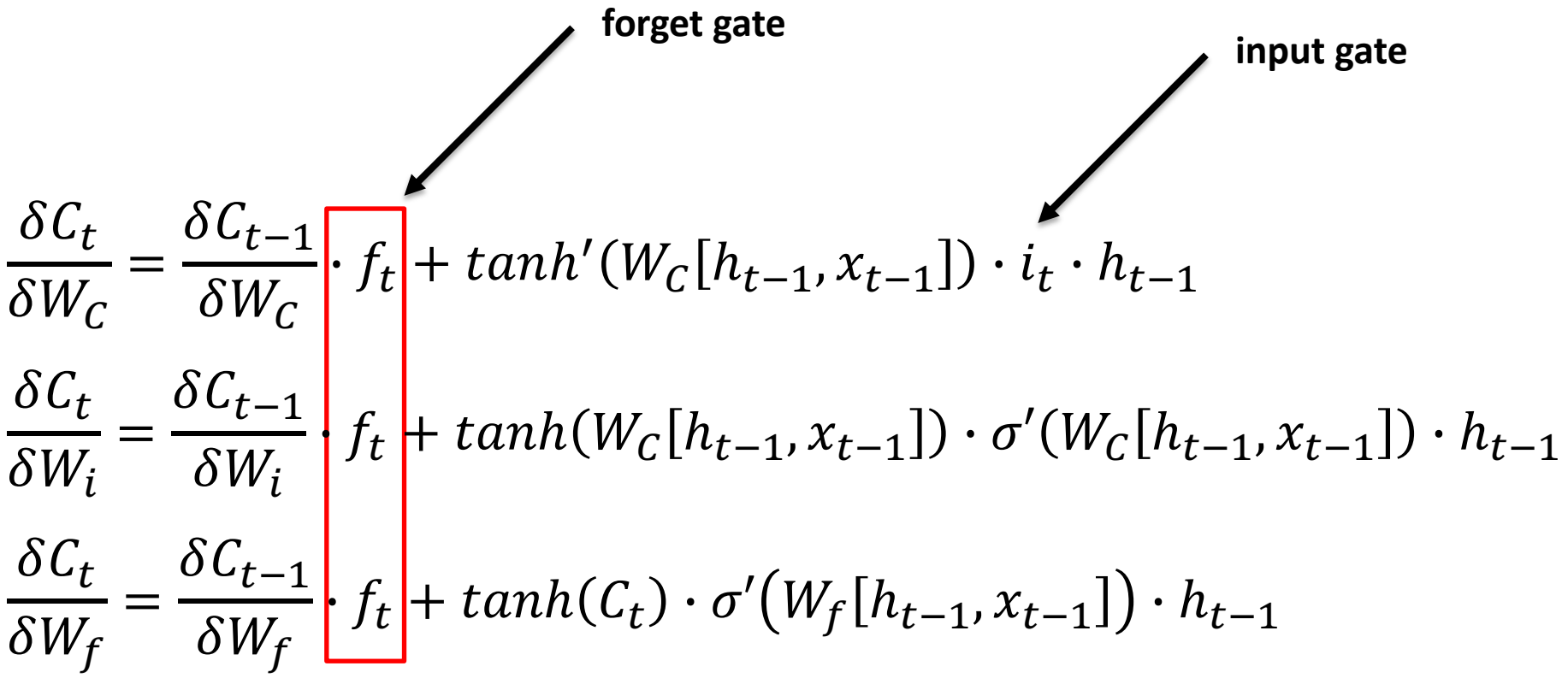
Long-Short Term Memory (LSTM)

$$\frac{\delta C_t}{\delta W_C} = \frac{\delta C_{t-1}}{\delta W_C} \cdot f_t + tanh'(W_C[h_{t-1}, x_{t-1}]) \cdot i_t \cdot h_{t-1}$$

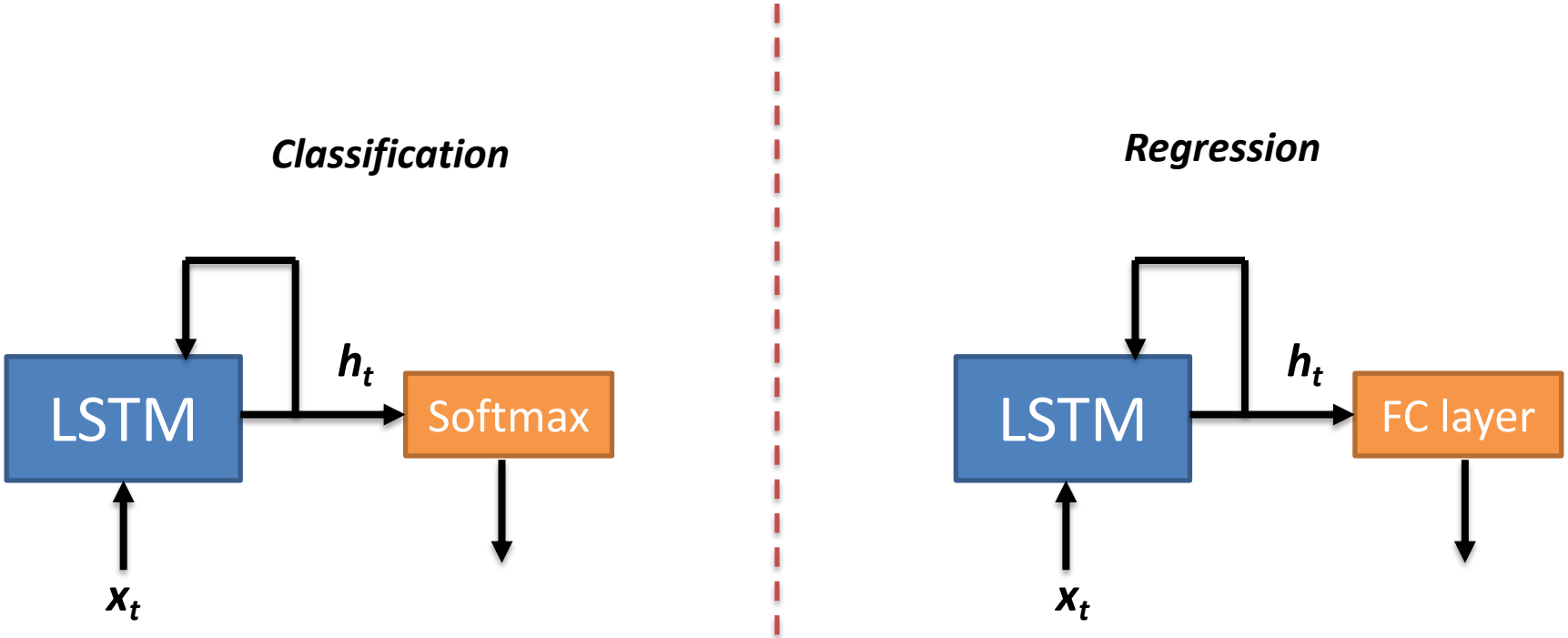
$$\frac{\delta C_t}{\delta W_i} = \frac{\delta C_{t-1}}{\delta W_i} \cdot f_t + tanh(W_C[h_{t-1}, x_{t-1}]) \cdot \sigma'(W_C[h_{t-1}, x_{t-1}]) \cdot h_{t-1}$$

$$\frac{\delta C_t}{\delta W_f} = \frac{\delta C_{t-1}}{\delta W_f} \cdot f_t + tanh(C_t) \cdot \sigma'\left(W_f[h_{t-1}, x_{t-1}]\right) \cdot h_{t-1}$$

Long-Short Term Memory (LSTM)



LSTM Networks

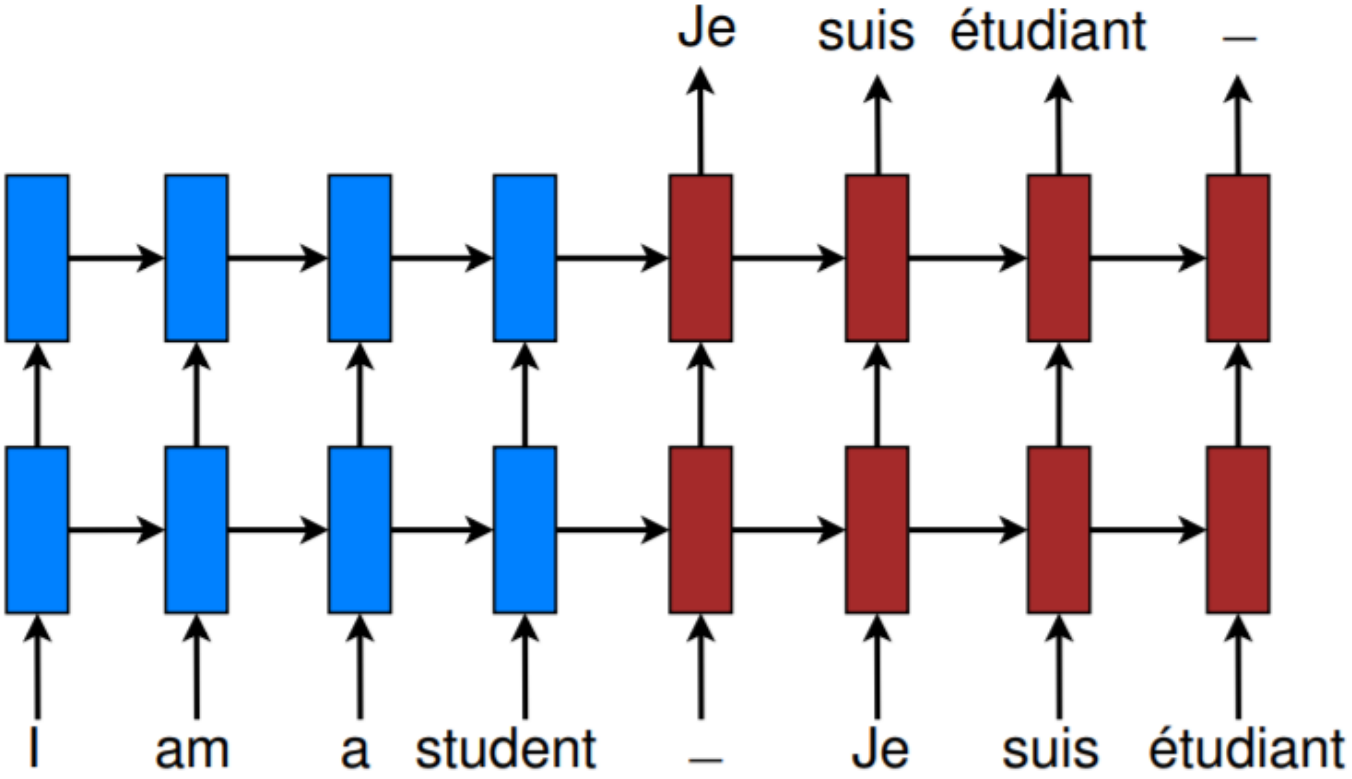


torch.nn.functional.softmax()

torch.nn.Linear()

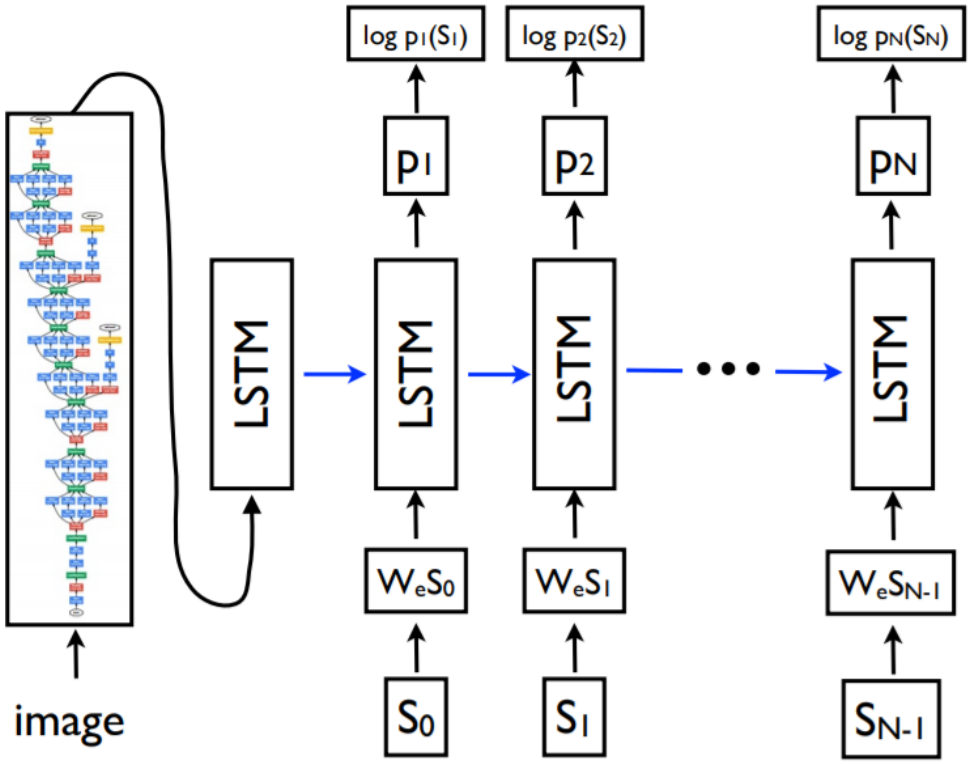
LSTM

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. ***"Sequence to sequence learning with neural networks."*** Advances in neural information processing systems. 2014.



LSTM + CNN

Vinyals, Oriol, et al. ***"Show and tell: A neural image caption generator."*** Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. IEEE, 2015.

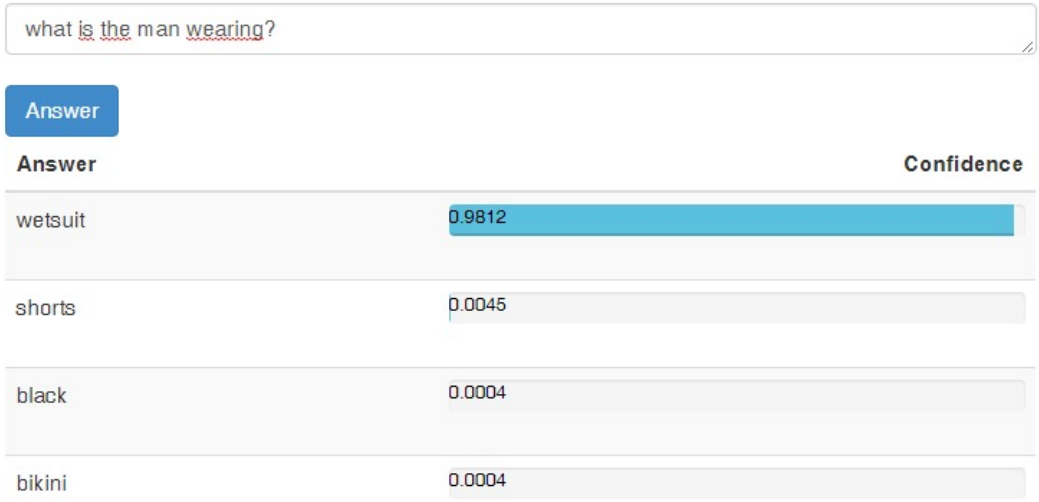


LSTM + CNN

Vinyals, Oriol, et al. ***"Show and tell: A neural image caption generator."*** Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on. IEEE, 2015.

LSTM + CNN

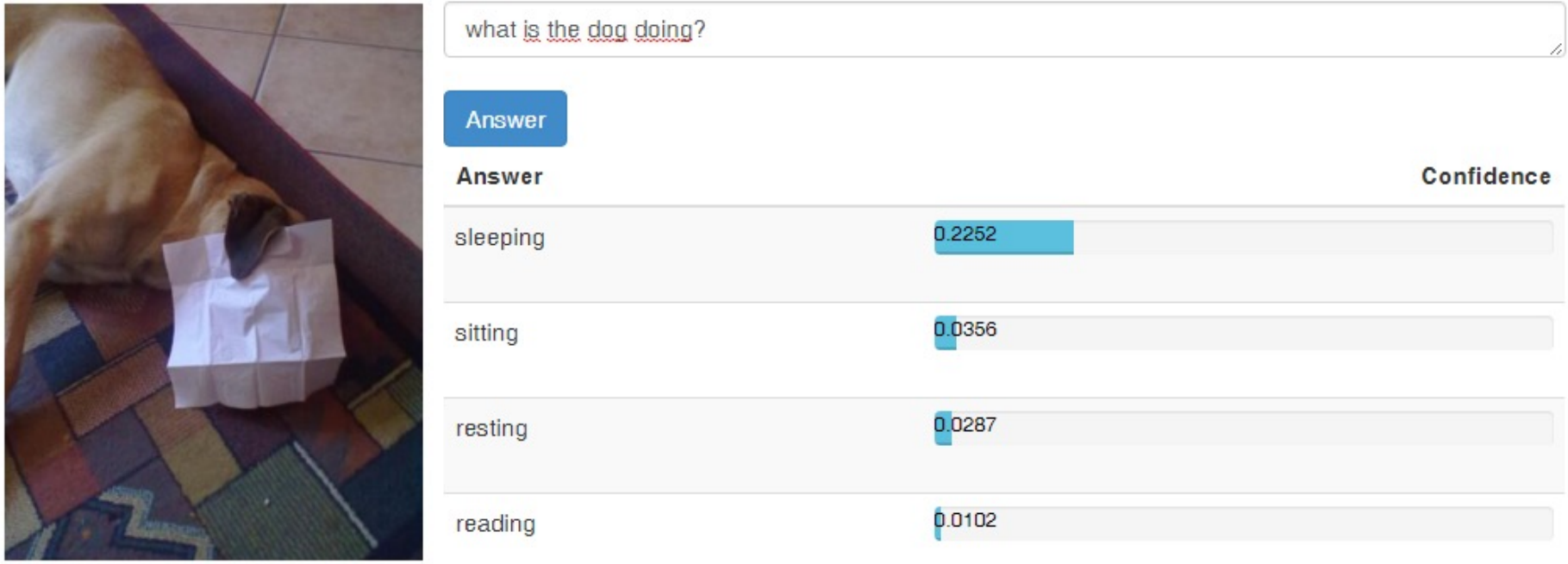
Stanislaw Antol , Aishwarya Agrawal , Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh ***"Visual Question Answering."*** (2015).



VQA demo: http://vqa.cloudcv.org/

LSTM + CNN

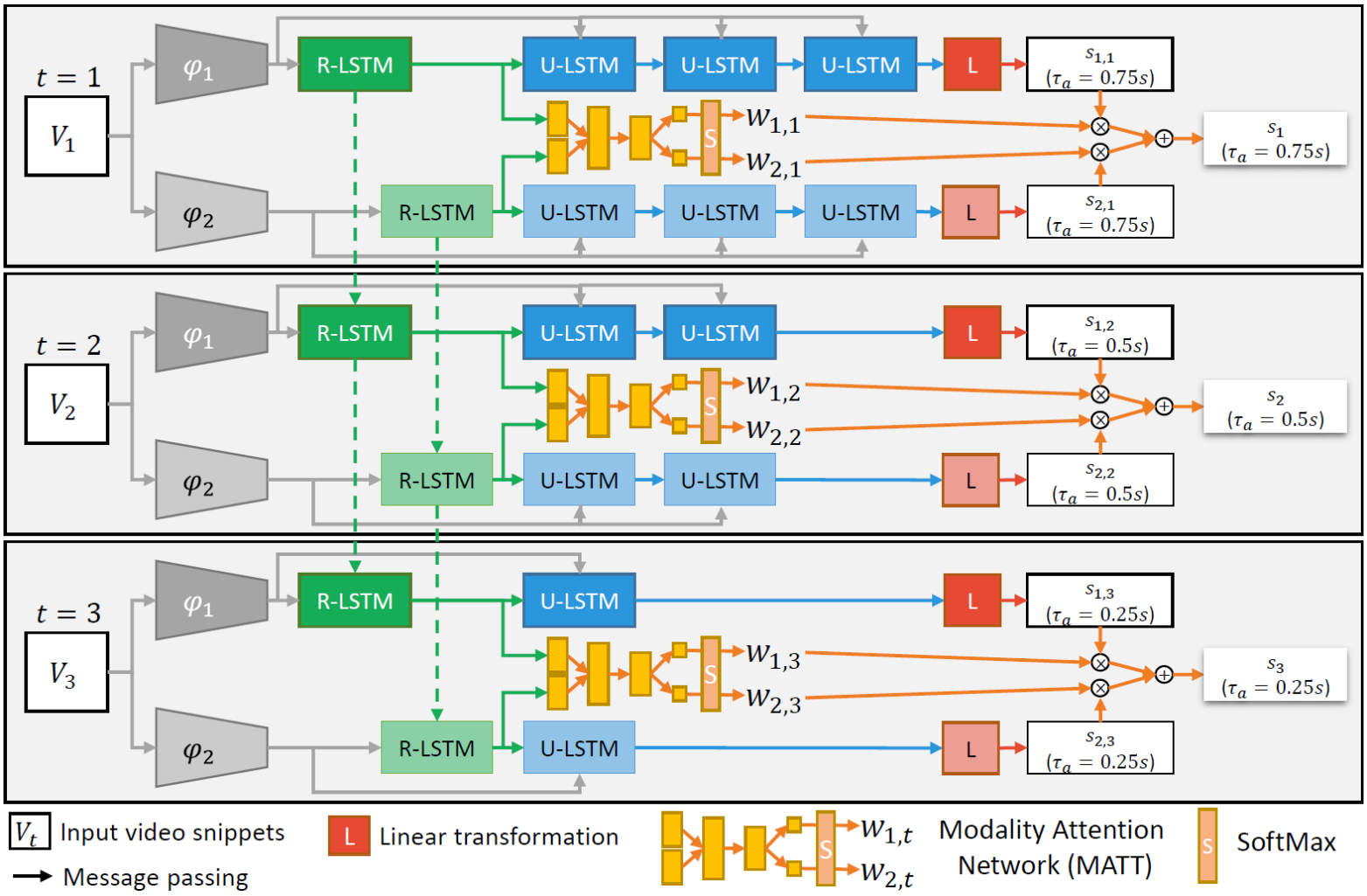
Stanislaw Antol , Aishwarya Agrawal , Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh ***"Visual Question Answering."*** (2015).



VQA demo: http://vqa.cloudcv.org/

What Would You Expect? Anticipating Egocentric Actions with Rolling-Unrolling LSTMs and Modality Attention

Antonino Furnari, Giovanni Maria Farinella



Page - Code: https://iplab.dmi.unict.it/rulstm/

Video Demo: https://youtu.be/buIEKFHTVIg

References LSTM

LSTM papers:

- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

- Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

- Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." *Journal of machine learning research* 3.Aug (2002): 115-143.

- Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International Conference on Machine Learning*. 2013.

Other resources:

- LSTM Pytorch tutorial: https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

- *''Understanding LSTMs''* Colah's blog post: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- VQA demo: http://vqa.cloudcv.org/