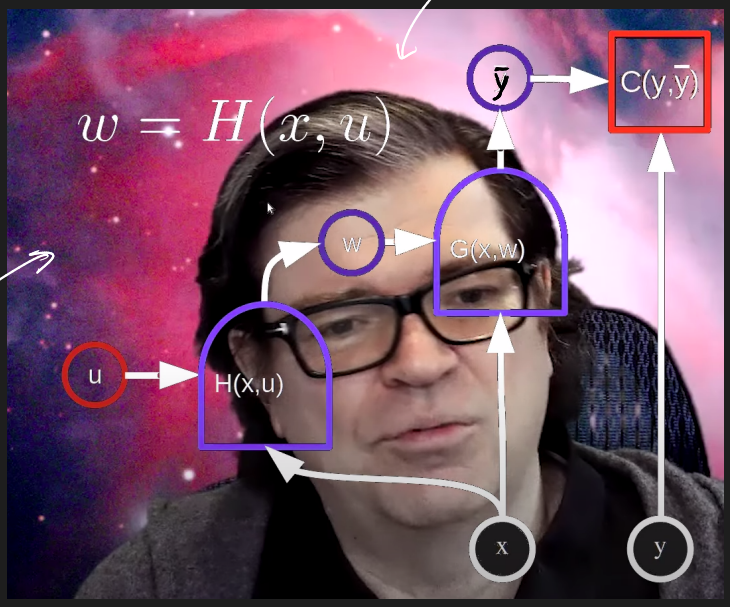
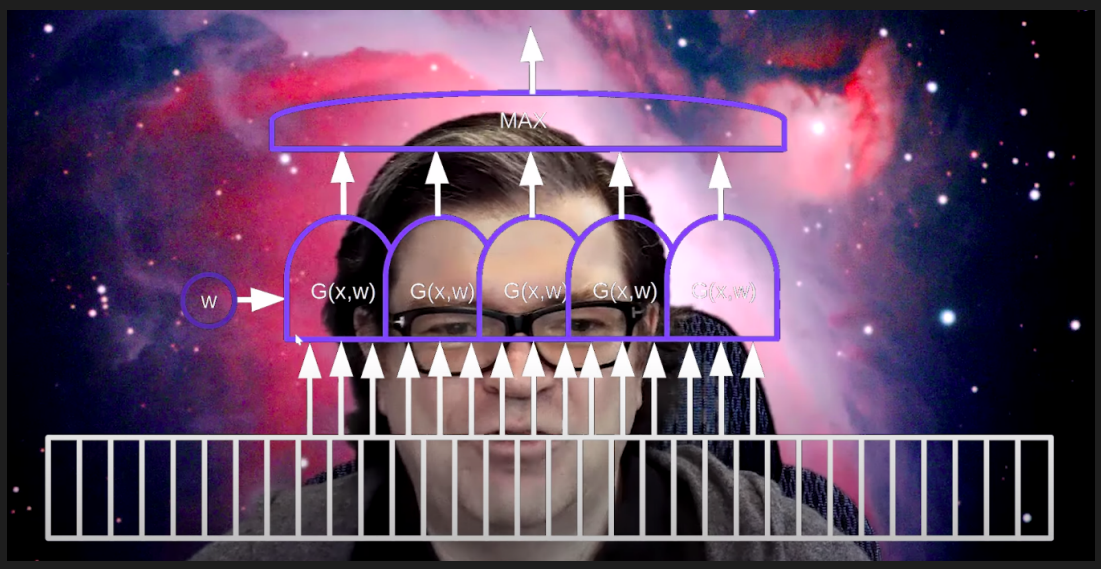
Hyper Networks



- Weights w are the output of another network $H(x, u)$

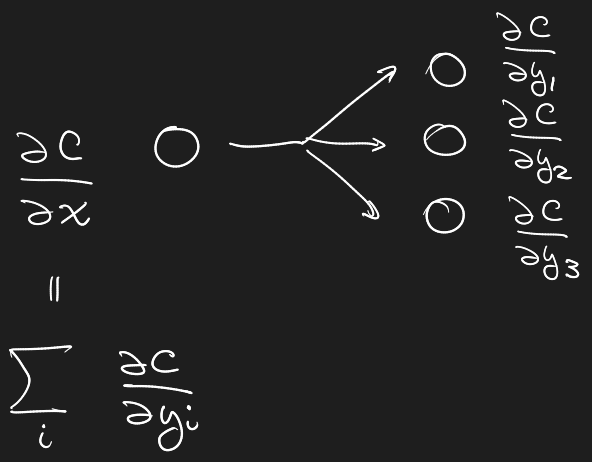
Another idea: Shared weights

- Searching for a motif / pattern.



Because the weights are shared, while doing backprop, the contribution of each replica of the network (each gradient) is added, to update the only weights w

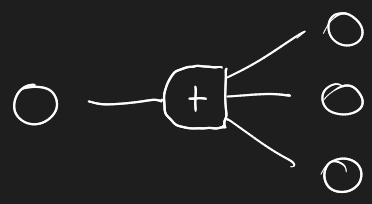
Backward pass of replicated variable



$$\frac{\partial C}{\partial x} = \sum_i \frac{\partial C}{\partial y_i}$$

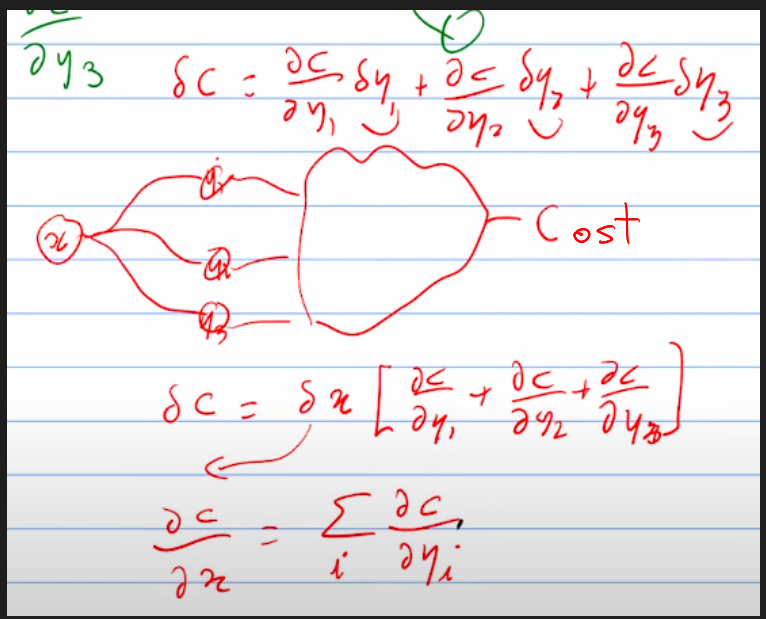
Branching up in the forward prop.

correspond to a sum in the backward prop:

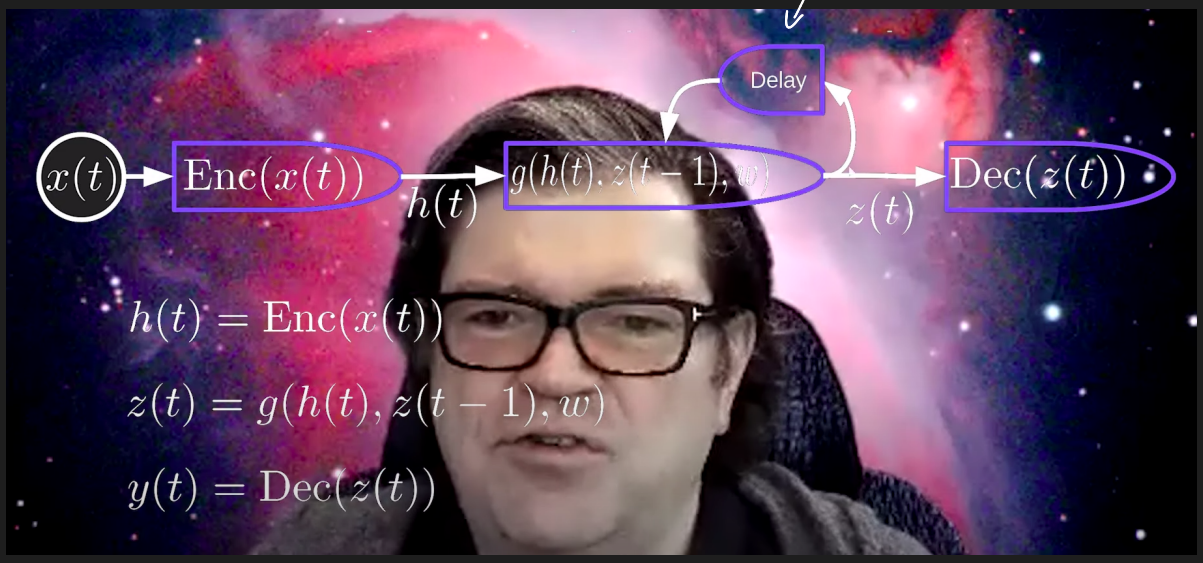


And vice versa.

- Q: "Why we sum the gradients?"

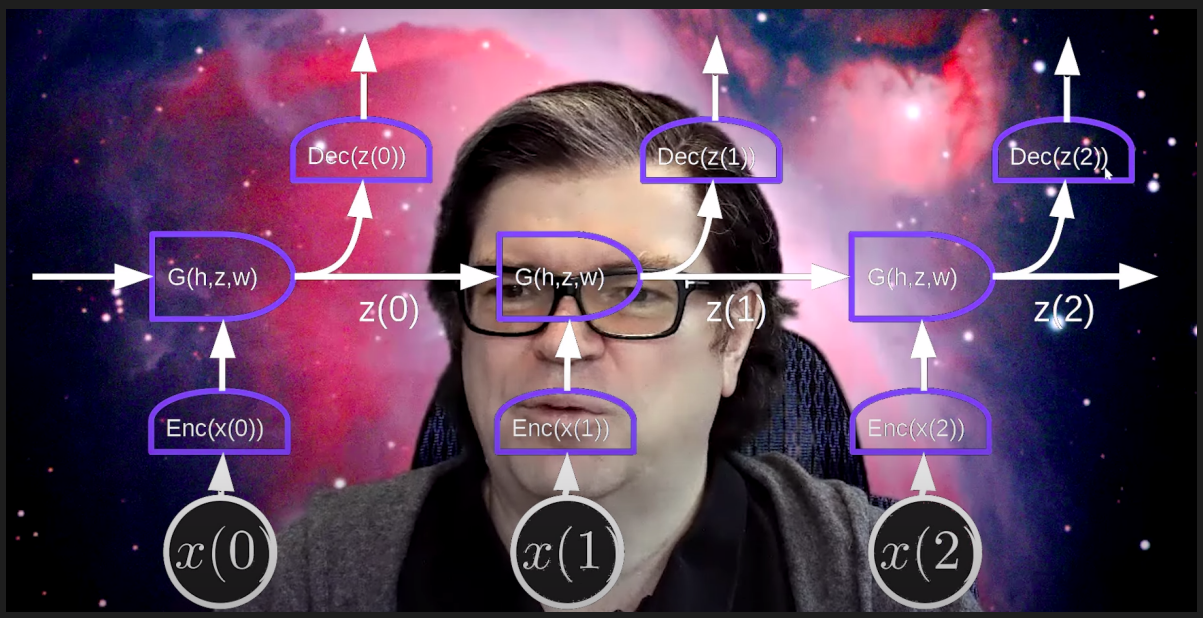


Loops : RNNs



Allows the network to have a memory.

Unrolling the network.



Difficult to train.

↳ Long sequences : Vanishing/exploding gradients.

- Exploding : easy to solve (ie : add a sigmoid)
- Vanishing : not so easy.

Example

$$z_t \boxed{W} z_{t+1} = W z_t$$

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

eigen values of 1

$$W = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

↑ Rotation matrix

They do not change the norm after doing the prod

$$\| z_t \| = \| W z_t \| = \| z_{t+1} \|$$

Now, if $W = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

$$\Downarrow$$

$$\| z_{t+1} \| = 2 \cdot \| z_t \|$$

↑ explosion

$$W = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

$$\Downarrow$$

$$\| z_{t+1} \| = \frac{1}{2} \| z_t \|$$

↑ Vanishing (shrinks)

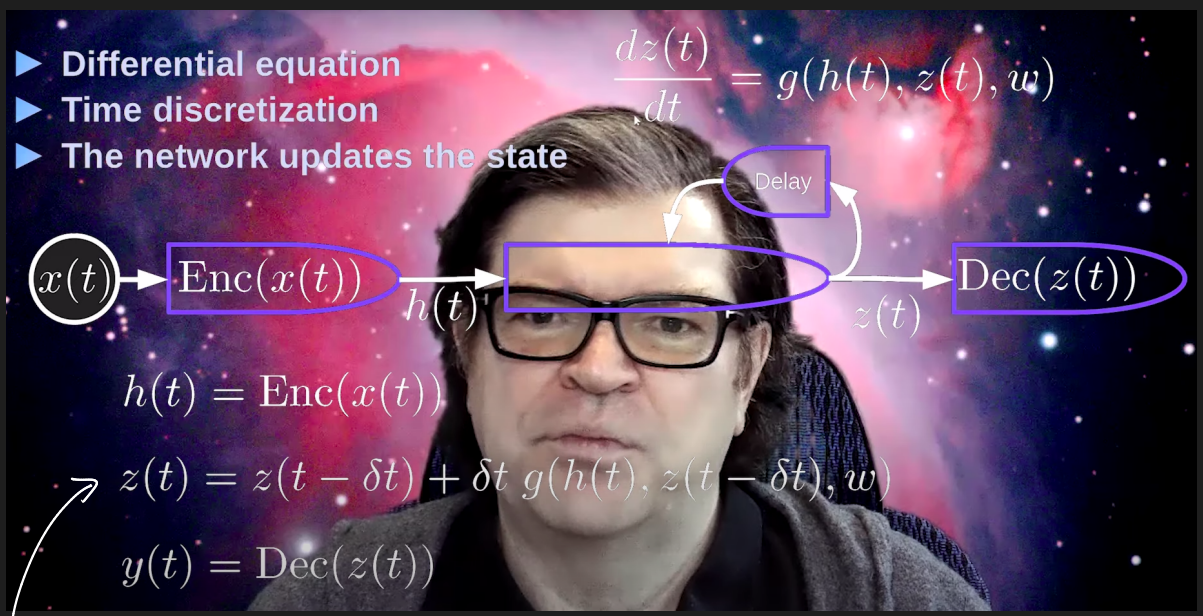
RNN tricks

- [Pascanu, Mikolov, Bengio, ICML 2013; Bengio, Boulanger & Pascanu, ICASSP 2013]

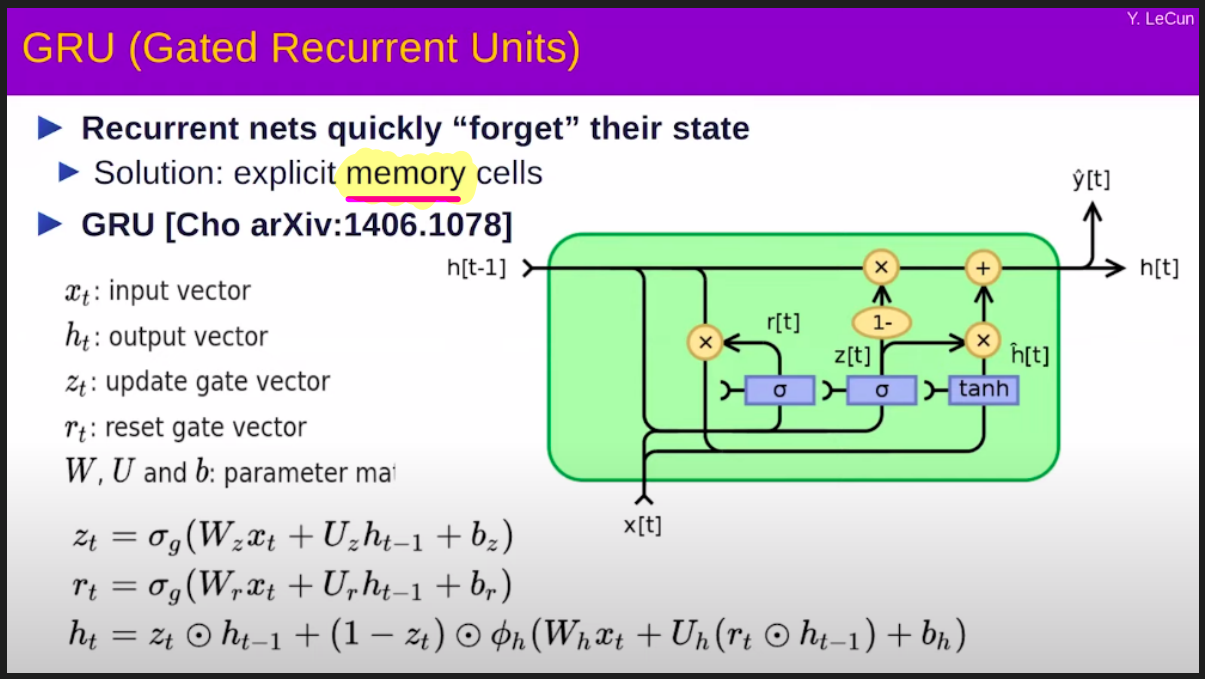
- Clipping gradients (avoid exploding gradients)
- Leaky integration (propagate long-term dependencies)
- Momentum (cheap 2nd order)
- Initialization (start in right ballpark avoids exploding/vanishing)
- Sparse Gradients (symmetry breaking)
- Gradient propagation regularizer (avoid vanishing gradient)
- LSTM self-loops (avoid vanishing gradient)

Other type of RNNs

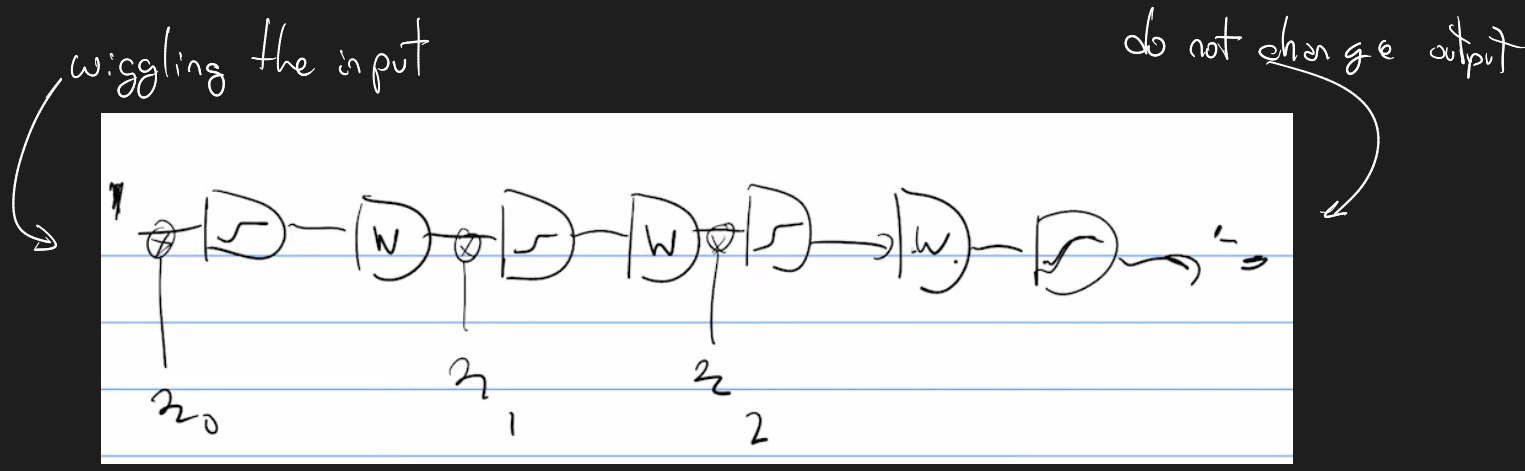
- Neural ODEs



discretization of the differential equation.



On RNNs

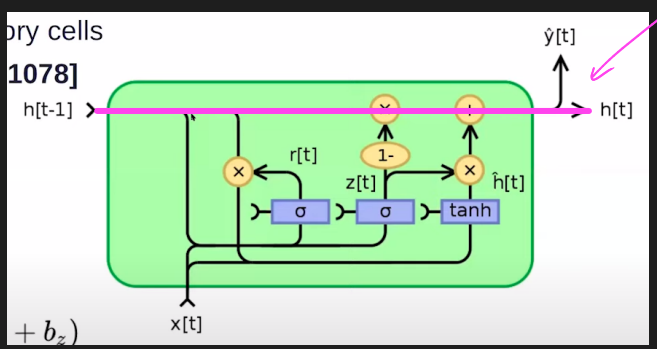


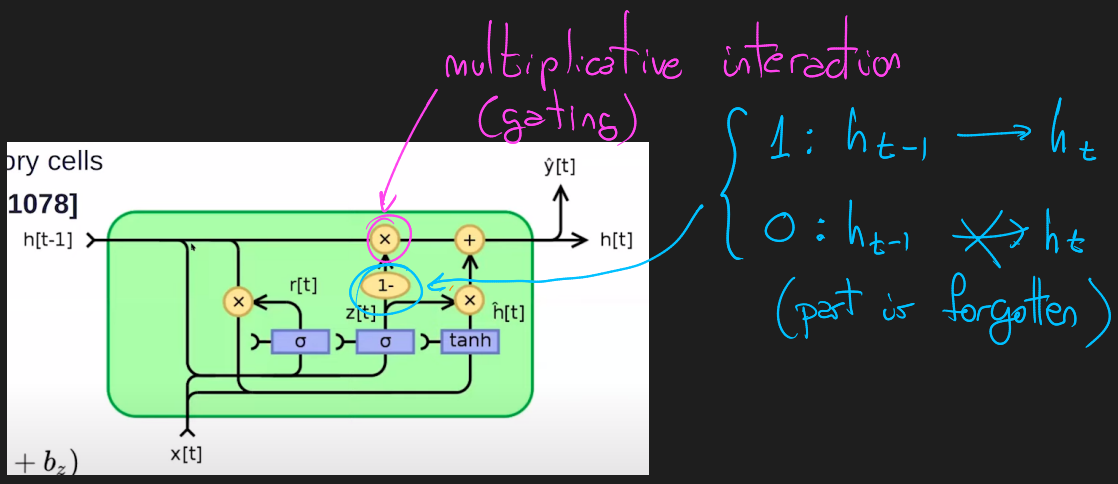
if the network state is saturated.

With GRUs

we can make the system remember by default (system=identity function by default: new state is equal to old state)

and have a neural network that slightly modifies it

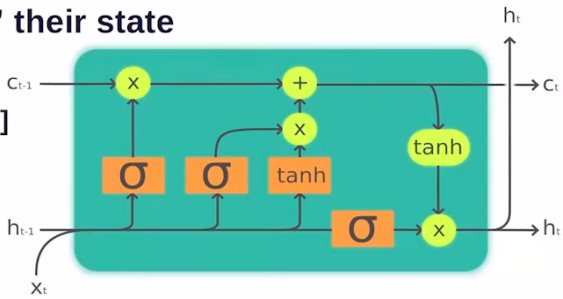




LSTM (Long Short-Term Memory)

Y. LeCun

- **Recurrent nets quickly "forget" their state**
  - Solution: explicit memory cells
- **LSTM [Hochreiter & Schmidhuber 97]**

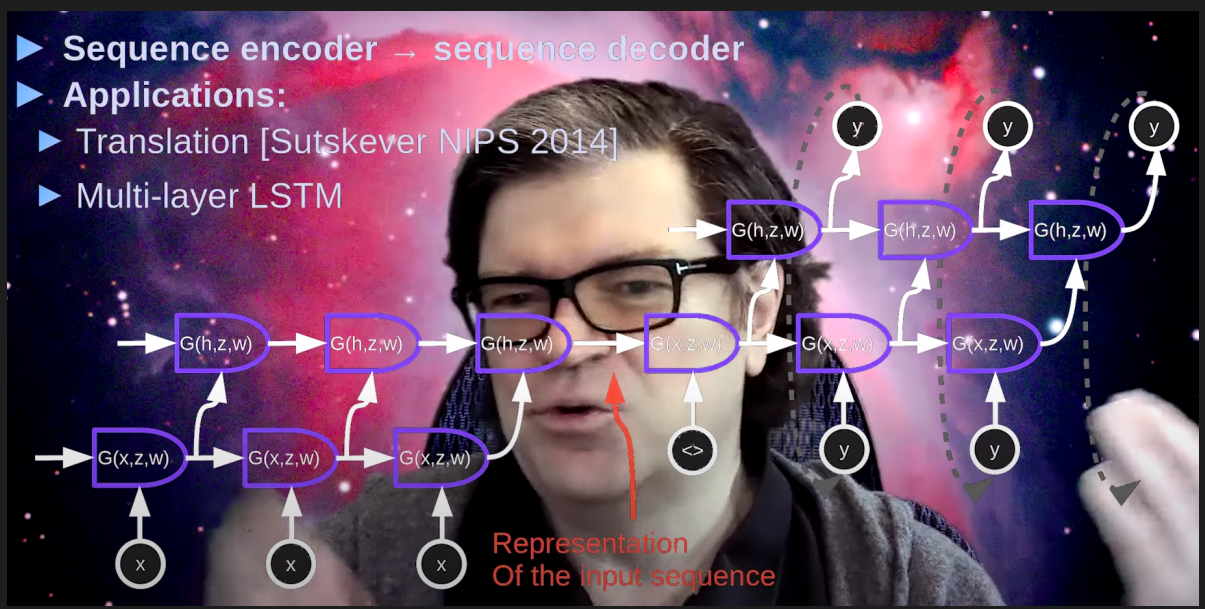


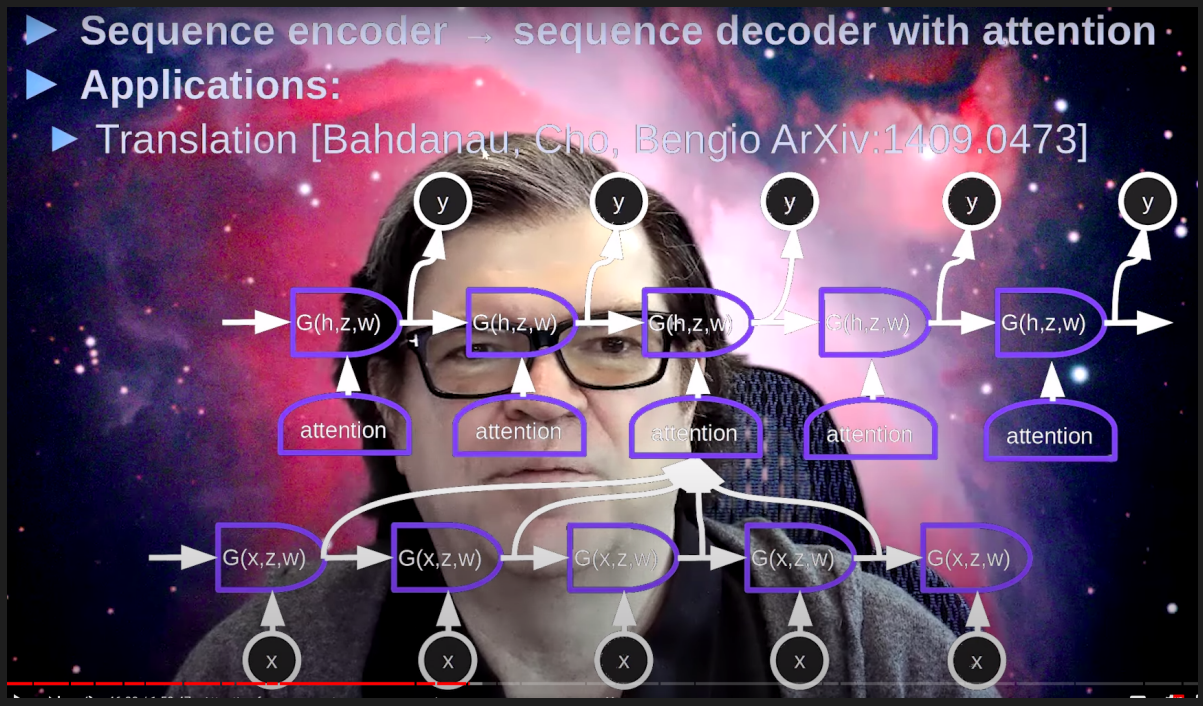
$x_t \in \mathbb{R}^d$: input vector to the LSTM unit
$f_t \in \mathbb{R}^h$: forget gate's activation vector
$i_t \in \mathbb{R}^h$: input/update gate's activation vector
$o_t \in \mathbb{R}^h$: output gate's activation vector
$h_t \in \mathbb{R}^h$: hidden state vector also known as output
$c_t \in \mathbb{R}^h$: cell state vector

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

Legend: Layer, Pointwize op, Copy

Guillaume Chevalier https://commons.wikimedia.org/w/index.php?curid=71836793



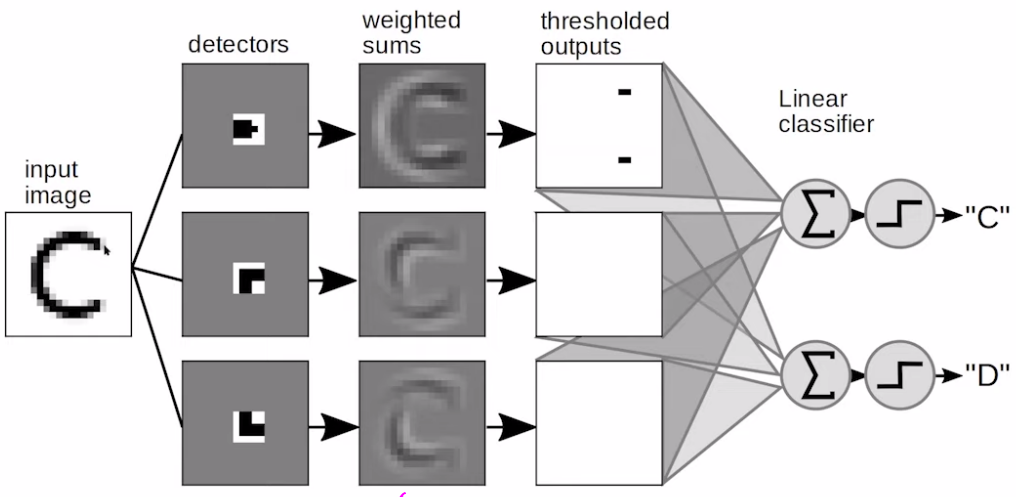


Convolutional Networks

Hierarchical representation of the world

Detecting Motifs in Images

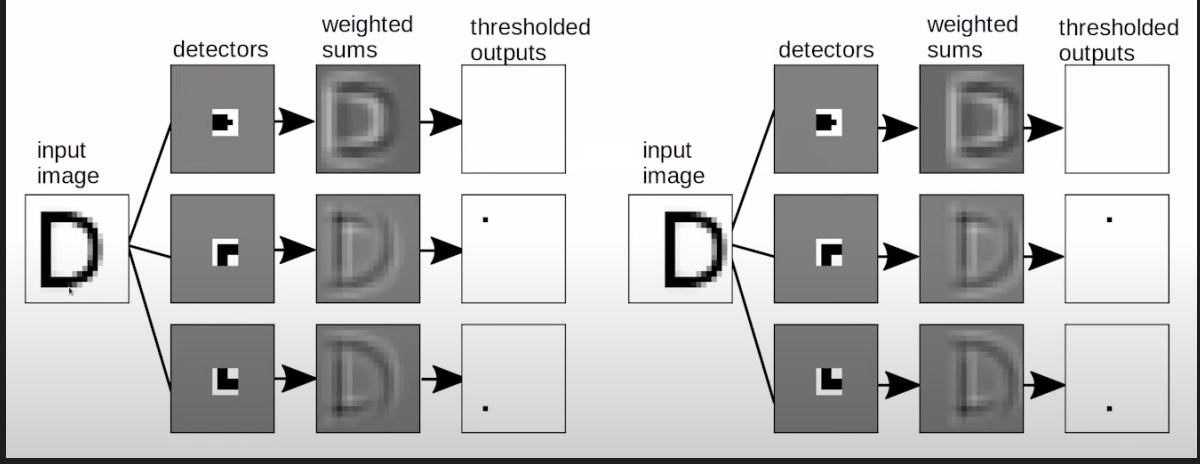
- Swipe “templates” over the image to detect motifs



masks

feature maps

Shift invariance



Discrete Convolution: Swipe an small pattern of coefficient over an image

Def: Convolution (1D)

$$y_i = \sum_j w_j x_{i-j}$$

In practice: Cross-correlation

(Pytorch)

$$y_i = \sum_j w_j x_{i+j}$$

In 2D

$$y_{ij} = \sum_{kl} w_{kl} x_{i+k,j+l}$$

Backpropagating through convolutions

Y. LeCun

- **Convolution** $y_i = \sum_j w_j x_{i+j}$
  - (really: cross-correlation)
- **Backprop to input** $\frac{\partial C}{\partial x_j} = \sum_k w_k \frac{\partial C}{\partial y_{j-k}}$
  - Sometimes called "back-convolution"
- **Backprop to weights** $\frac{\partial C}{\partial w_j} = \sum_i \frac{\partial C}{\partial y_i} x_{i+j}$

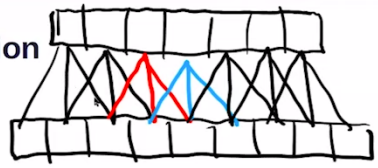
To Recheck!

Stride and Skip: subsampling and convolution “à trous”

Y. LeCun

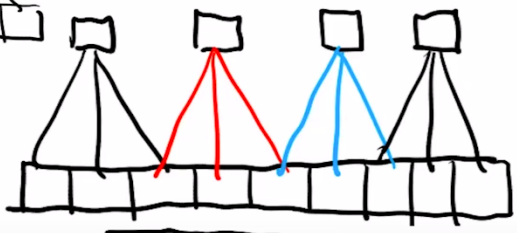
- **Regular convolution**
  - “dense”

kernel = 3
stride = 1

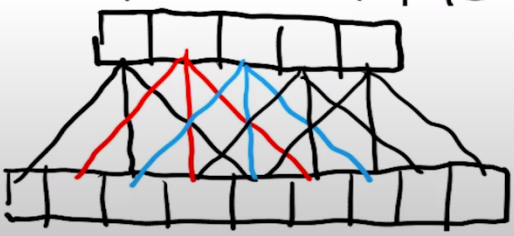


- **Stride**
  - subsampling convolution
  - Reduces spatial resolution

kernel = 3, stride = 2



- **Skip, convolution “à trous”**
  - pronounced “ah troo” (means “with holes”)
  - Dimensionality reduction without loss of resolution



window of 5, but with holes

kernel = 3

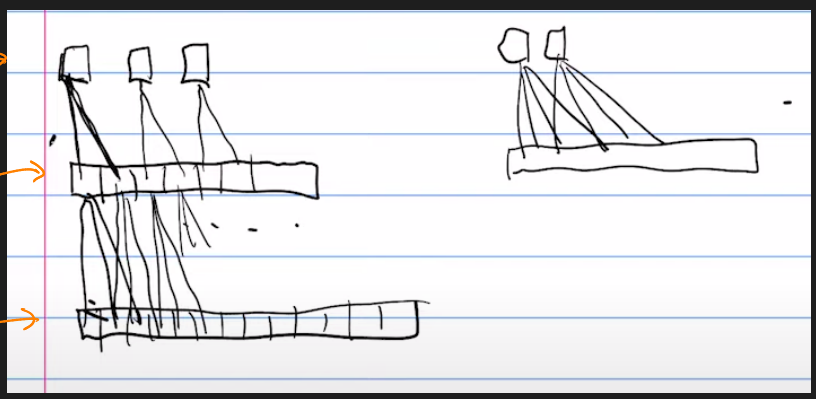
skip = 1

stride = 1

Pooling →

Convolution layer →

Input →



Pooling

↳ Aggregation

↳ Permutation invariant (f = sum, avg, max, L_p,

most popular: p=2

log sum exp

$y = \left(\sum_k x_k^p\right)^{1/p}$ L_p pooling

$y = \max_k(x_k)$ max pooling

$y = \frac{1}{\beta} \log \sum_k e^{\beta x_k}$ log sumexp pooling

f invariant to T : $f(T(x)) = f(x)$

f equivariant to T : $f(T(x)) = T'(f(x))$

Convolution op. is equiv. to translation

$$T(x)_i = x_{i+T}$$

Pooling is locally invariant to translation