Statistical and Sequential Learning for Time Series Forecasting

Neural Networks, RNN, LSTM etc.

Margaux Brégère

Classification And Regression Tree - CART

- Segmentation criteria for classification
- Segmentation criteria for regression
- Algorithm

Bagging

- Bootstrap
- Prediction error diminution

Random Forest

- Algorithm
- Out of bag error and importance

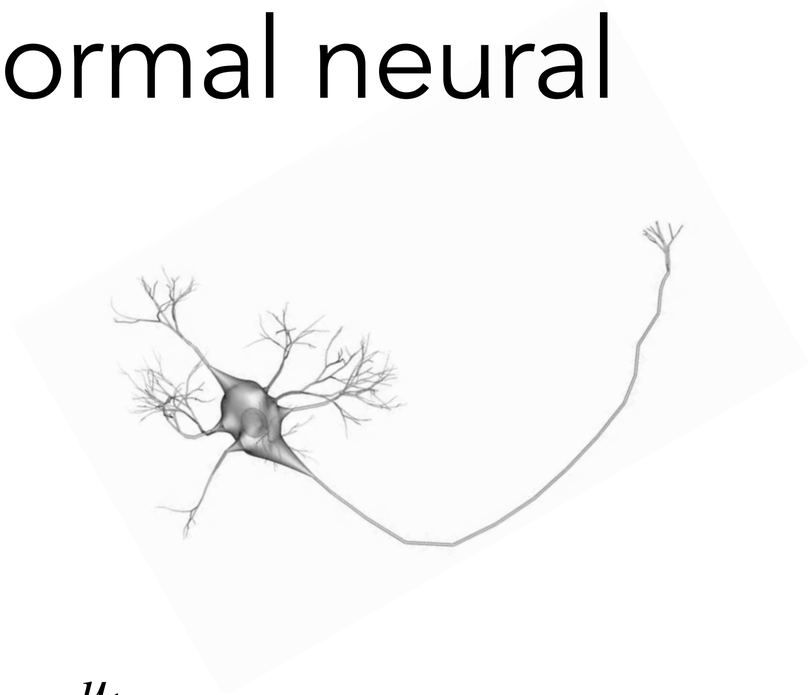
Boosting

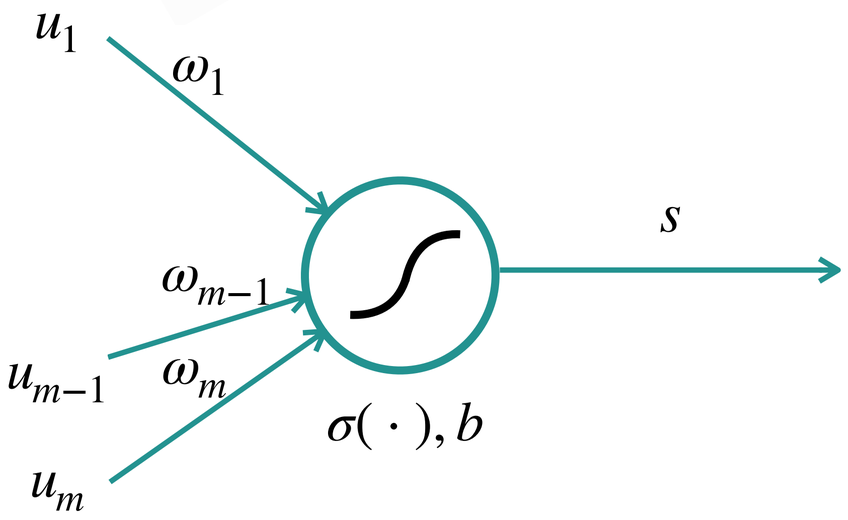
- Adaboost
- Gradient boosting

Online approches

Neural Networks

Formal neural





Inputs:

- Weights $\omega_1, \omega_2, \ldots, \omega_m$
- Bias b
- Non linear activation function $\sigma(\cdot)$

  Sigmoid $\sigma(x) = \dfrac{1}{1+\mathrm{e}^{-x}}$

  Hyperbolic tangent $\sigma(x) = \dfrac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$

  Relu $\sigma(x) = \max(0, x)$ …

Output: $s = \sigma\left(\sum_{k=1}^{m} \omega_k u_k - b\right)$

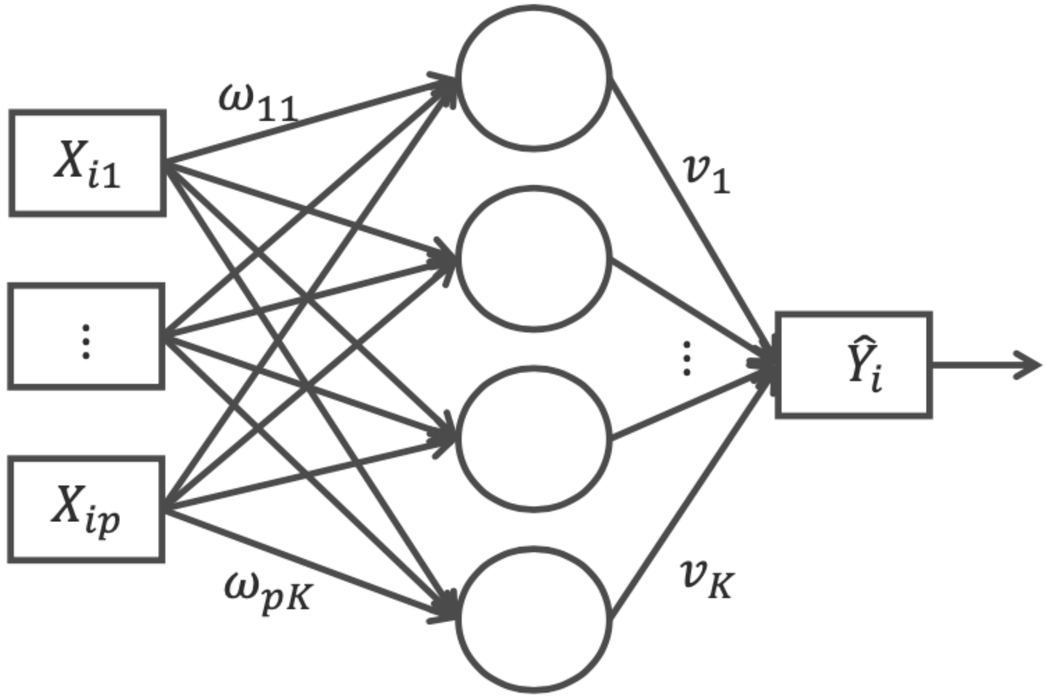
W. McCulloch and W. Pitts (1943)

Neural network with a hidden layer

One-layer perceptron with a continuous non-polynomial activation function $\sigma(\cdot)$ for each node

Output:

$$\hat{f}(X) = \sum_{k=1}^{K} \nu_k \sigma\Big(\sum_{j=1}^{p} \omega_{jk} X_j - b_k \Big) = \sum_{k=1}^{K} \nu_k \sigma\Big(\omega_k^{\mathrm{T}} X - b_k \Big)$$



Universality Approximation Theorem

Theorem (G. Cybenko, 1989)

Let σ be a sigmoidal function $\sigma(x) = \begin{matrix} 1 & \text{for} & x \to +\infty \\ 0 & \text{for} & x \to -\infty \end{matrix}$, the finite sums $s_K : [0,1]^p \to \mathbb{R}$ of the form $s_K(X) = \sum_{k=1}^{K} a_k \, \sigma\left(\omega_k^{\mathrm{T}} X + b_k\right)$ with $K \in \mathbb{N}$, $a_k \in \mathbb{R}$, $\omega_k \in \mathbb{R}^p$ and $b_k \in \mathbb{R}$, are dense in $C\left([0,1]^p\right)$ with respect to the supremum norm

Equivalently, given any continuous function $f : [0,1]^p \to \mathbb{R}$ and $\varepsilon > 0$, $\exists\ K \in \mathbb{N}$, $a_1, \ldots, a_K \in \mathbb{R}$, $\omega_1, \ldots, \omega_K \in \mathbb{R}^p$ and $b_1, \ldots, b_K \in \mathbb{R}$ such that

$$\forall X \in [0,1]^p, \quad \left| f(X) - \sum_{k=1}^{K} a_k \, \sigma\left(\omega_k^{\mathrm{T}} X + b_k\right) \right| < \varepsilon$$

Universality Approximation Theorem

→ a neural network with a hidden layer can approximate with as much precision as desired (layer with $K(\varepsilon)$ neurons and with activation functions σ) any continuous function

Sketch of the proof:

Decompose in Fourier series and filter low frequencies (depends on ε)

Show that sine and cosine decompose on the activation function σ

Universality Approximation Theorem

Theorem (Maiorov, 1999):

If f has m Sobolev derivatives, the total number of neurons to approximate f with a one hidden layer neural network and a precision ε is of the order of

$$K \approx \varepsilon^{-\frac{p-1}{m}}$$

f has m Sobolev derivatives if its Fourier transform $\hat{f}$ satisfies $|\hat{f}(\omega)| = o(|\omega|^{-m}),\ \forall \omega \in \mathbb{Z}^p$ (low frequencies dominate and the function is truncated)

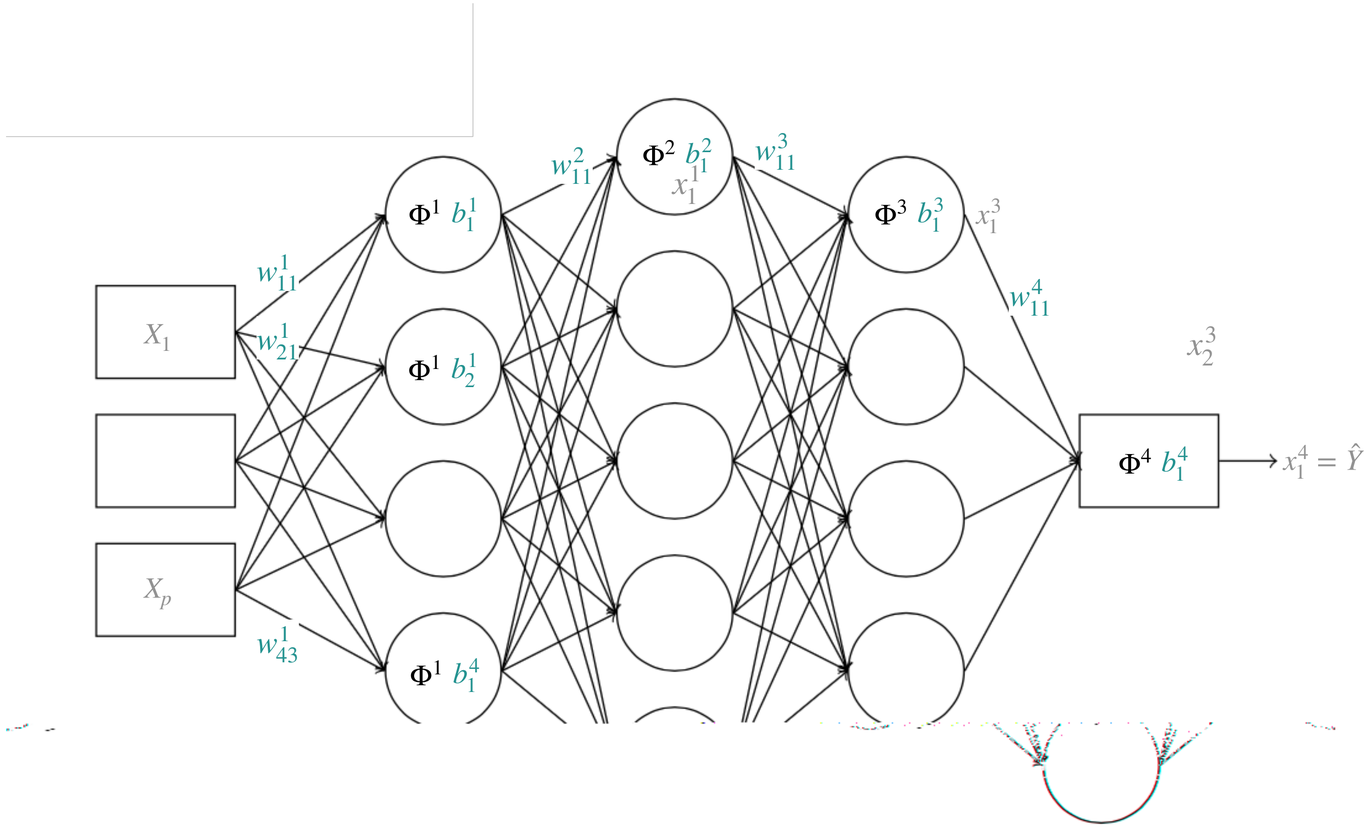
K exploses when $p \nearrow$ (feature number) or $m \searrow$ (non-smooth function)… mathematically elegant but impractical except in low dimension cases!

… and yet it works!

→ Increase the number of hidden layers

→ Add filters / convolutions etc.

Gradient Descent



Neural network with L layers,

Output: $\hat{Y}^\theta = \Phi_L\left(\Phi_{L-1}\left(\ldots\Phi_1(W^{1^{\mathrm{T}}}X)\ldots\right)\right)$

where $\theta = \left(W^l\right)_{l=1,\ldots,L}$

The neural network is trained by minimising

$$\ell(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i^\theta - Y_i\right)^2$$

using gradient descent:

$$\theta_{k+1} = \theta_k - \eta\,\nabla\ell(\theta_k)$$

Mini-batch Gradient Descent: at each iteration, the gradient is computed on a (random) sub-sample (a batch) of training data

Backpropagation

Step 1 - Forward: Propagate training data through the model from input to predicted output by computing the successive hidden layers' outputs and finally the final layer's output

Step 2 - Backward: Adjust the weights with gradient step:

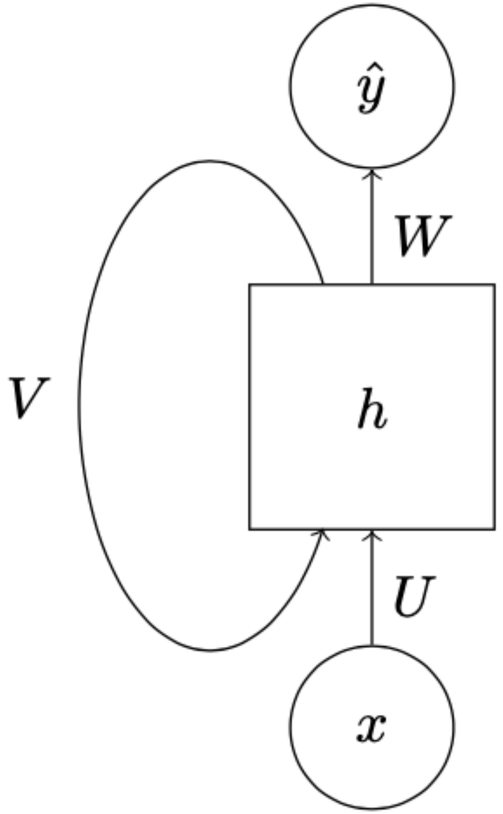
> The derivative is easy to calculate for final layer weights, and possible to calculate for one layer given the next layer's derivatives. Starting at the end, then, the derivatives are calculated layer by layer toward the beginning -- thus « backpropagation »

Repeatedly update the weights until they converge or the model has undergone enough iterations

Recurrent Neural Networks

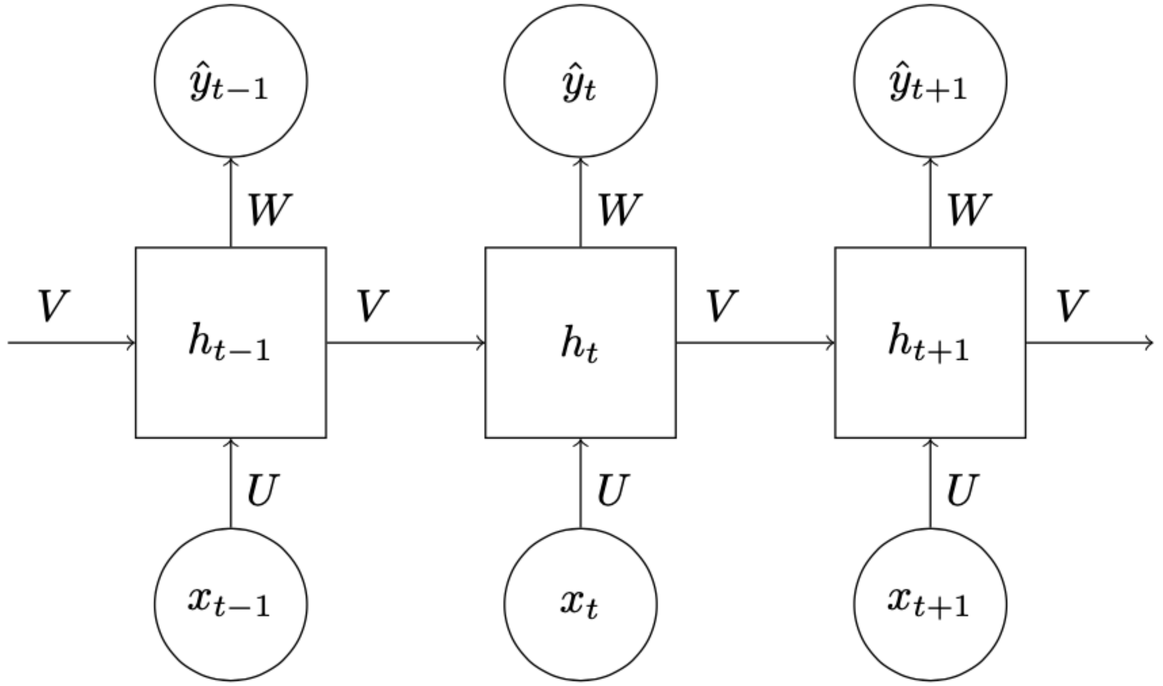
Sequential data: Time series - Text - Audio

Recurrent Neural Network (RNN)

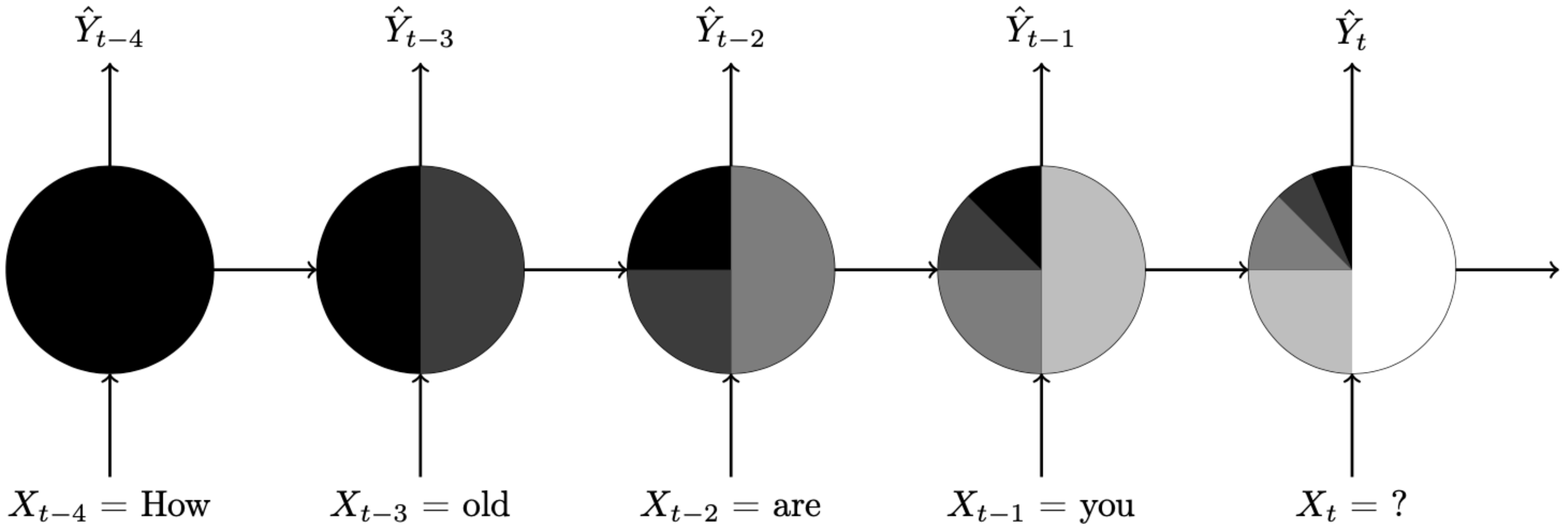


$$\hat{y}_t = \text{softmax}(c + Wh_t)$$

$$h_t = f(b + Ux_t + Vh_{t-1}) \text{, with } f = \tanh \text{ or ReLU}.$$



Recurrent Neural Network (RNN)



Training RNN with backpropagation:

gradients vanish (derivative of the tanh activation function which is smaller than 1)

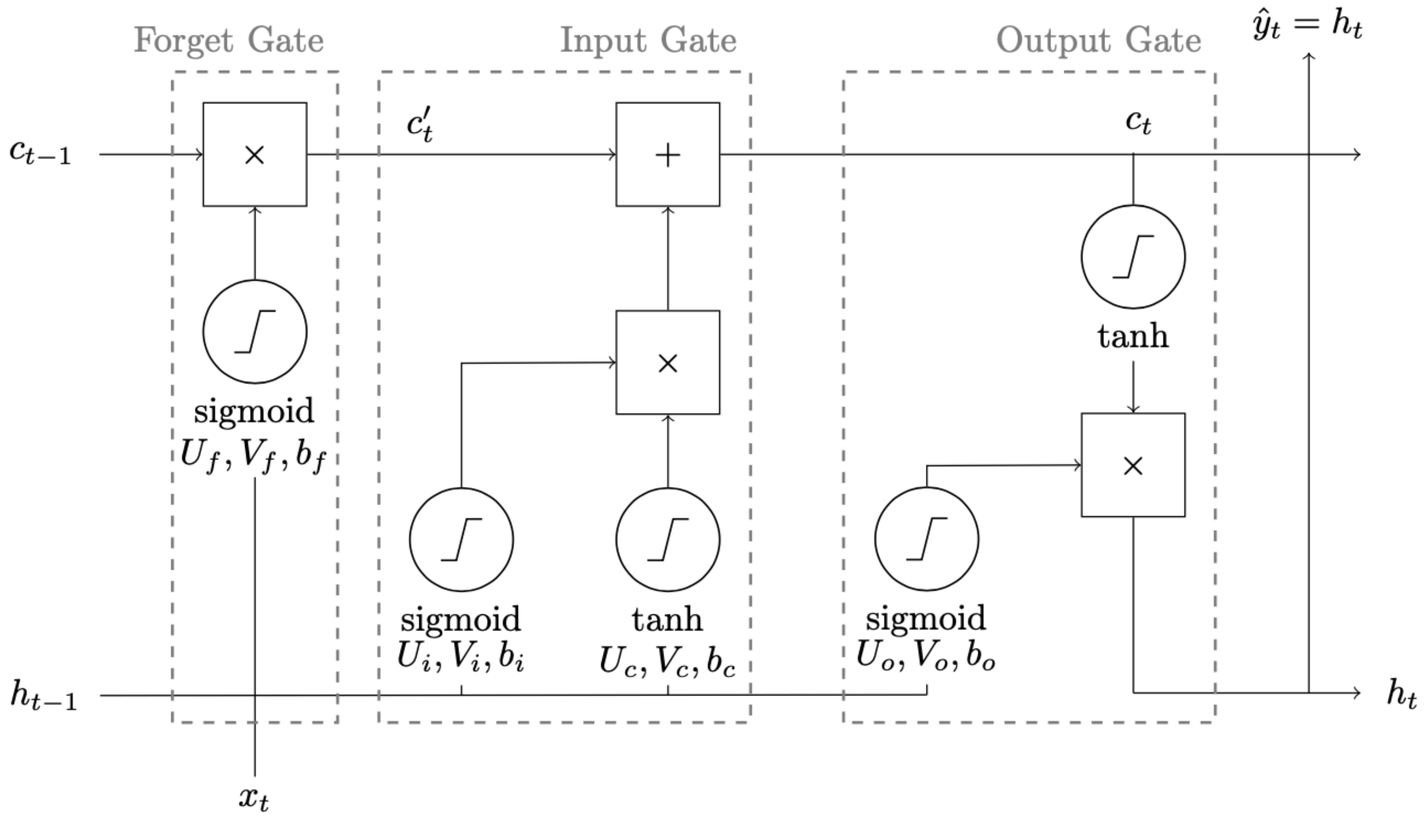
or explose (if weights V are large enough to overpower the smaller tanh derivative)

→ Clipping gradient

If sequences are short the matrix products do not vanish or explode for gradient computations

→ RNNs mainly learn short-term dependencies

Long Short-Term Memory (LSTM)



$$\text{output Forget Gate} = c'_t = c_{t-1} \times \sigma\left(U_f x_t + V_f h_{t-1} + b_f\right)$$

$$\text{output Input Gate} = c_t = c'_t + \sigma\left(U_i x_t + V_i h_{h-1} + b_i\right) \times \tanh\left(U_c x_t + V_c h_{h-1} + b_c\right)$$

$$\text{output Output Gate} = \hat{y}_t = h_t = \sigma\left(U_o x_t + V_o h_{h-1} + b_o\right) \times \tanh(c_t)\,.$$

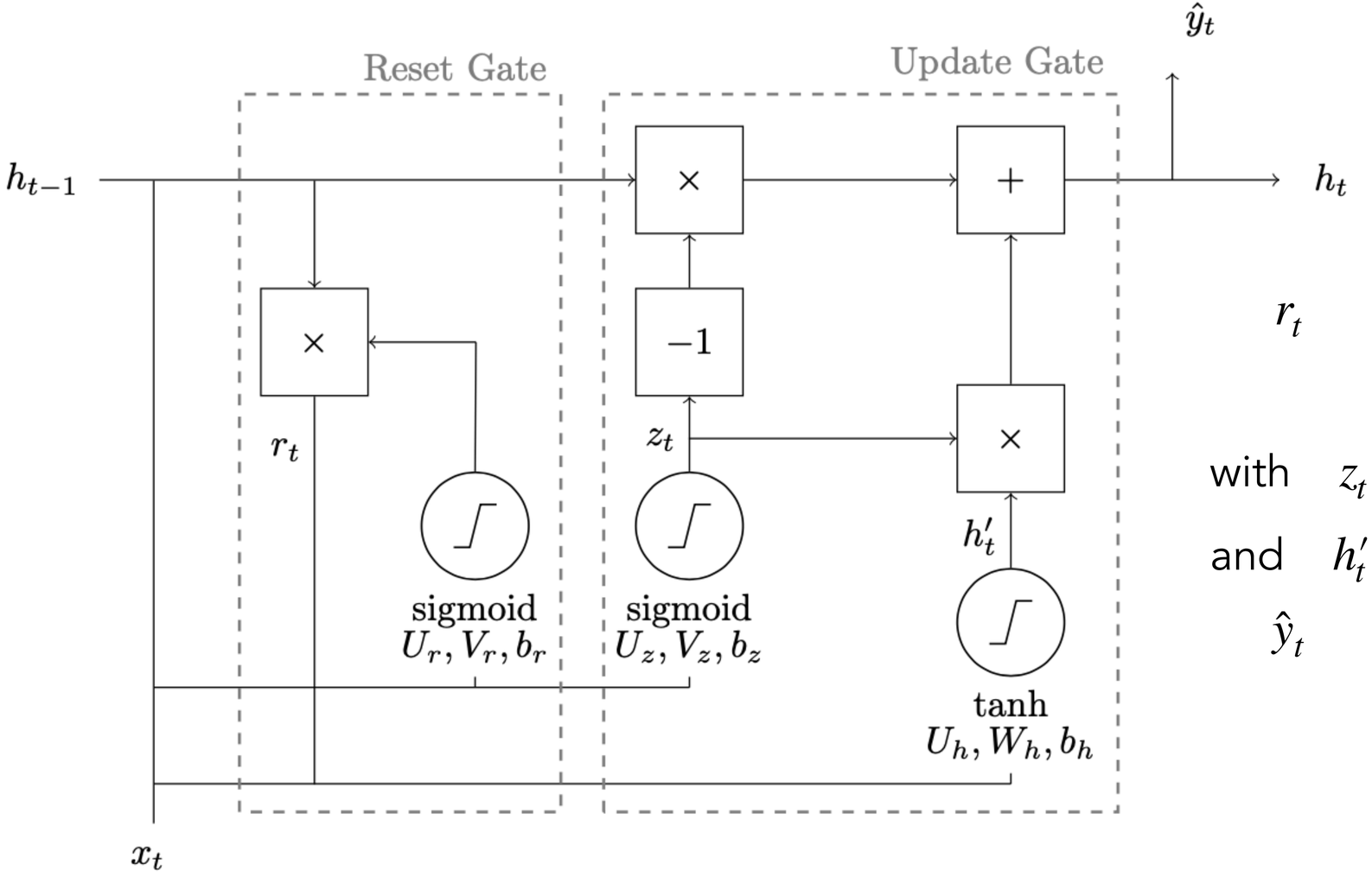
Long Short-Term Memory (LSTM)

The gating mechanism is used for updating and resetting the hidden state accordingly

A memory cell has an internal state is equipped with multiplicative gates which determine:

- Whether a given input should impact the internal state - input gate
- Whether the internal state should be put to 0 - forget gate
- Whether the internal state should impact the cell's output - output gate

Gated recurrent unit (GRU)



$$
\begin{aligned}
r_t &= \text{output Reset Gate} \\
&= h_{t-1} \times \sigma\left(U_r x_t + V_r h_{t-1} + b_r\right) \\
\text{with} \quad z_t &= \sigma\left(U_z x_t + V_z h_{t-1} + b_z\right) \\
\text{and} \quad h'_t &= \sigma\left(U_h x_t + W_h r_t + b_h\right) \\
\hat{y}_t &= \text{output Update Gate} \\
&= h_t = h_{t-1} \times (z_t - 1) + h'_t \times z_t \,.
\end{aligned}
$$

That’s all folks!