# Gated Recurrent Models

Generative and Deep Learning (GDL)

Davide Bacciu (davide.bacciu@unipi.it)


UNIVERSITÀ DI PISA

# Lecture Outline

- Gated approaches
  - Gating neurons
  - Long-Short Term Memories (LSTM)
  - Gated Recurrent Units (GRU)
- Randomized approaches
  - Controlling memory properties alone
  - Echo state network
- Autoregressive modelling with RNNs

# The naive solution (from previous lecture)

Ideal propagation in recurrent models can be achieved by choosing

- Identity activation function
- Identity weight matrix

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \hat{c}(\boldsymbol{x}_t)$$

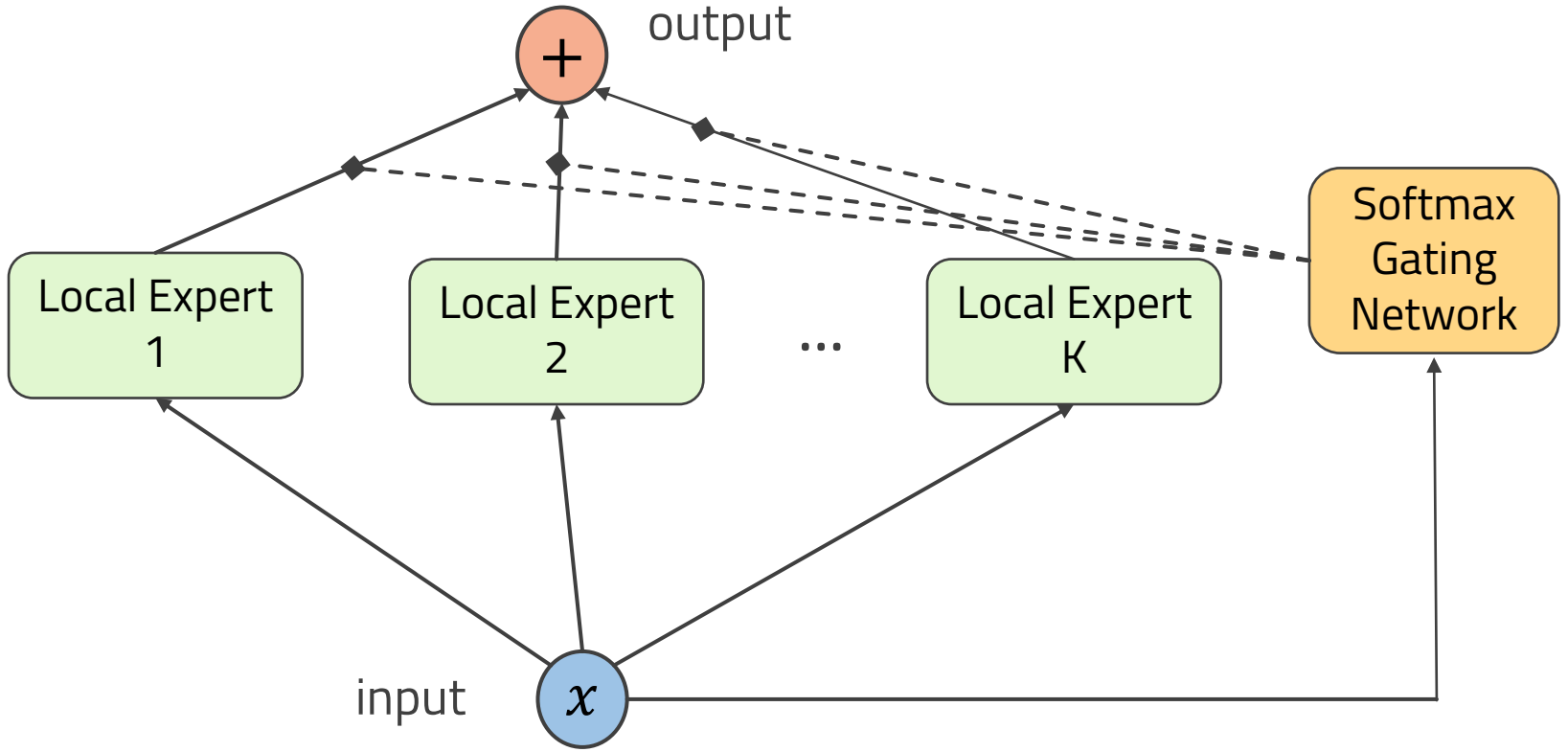
The formulation above has the desired spectral properties but does not work in practice as it quickly saturates memory (e.g. with replicated/non-useful inputs and states).

⇒ We want to be able to "control the forgetting"

# Gated Architectures

# Gating Units

Mixture of experts ⇒ the origin of gating




Jacobs et al (1991), Adaptive Mixtures of Local Experts, ...

# Forget gate

## Constant Error Carousel (CEC)

- Identity activation function
- Identity weight matrix

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \hat{c}(\boldsymbol{x}_t)$$

- No forgetting
- Hidden state saturation

## CEC + forget gate

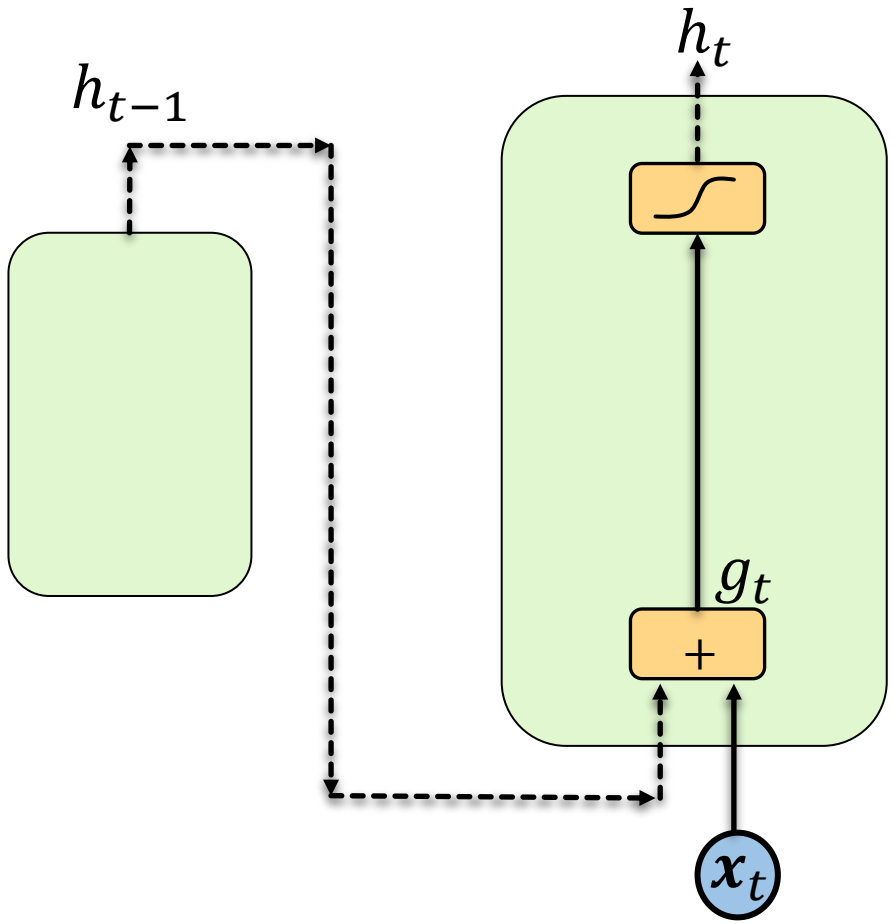
- Forget gate to "soft reset" units

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f)$$

$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \hat{c}(\boldsymbol{x}_t)$$

- Adaptively forgets the past
- Avoid saturation
- No guarantees about constant propagation

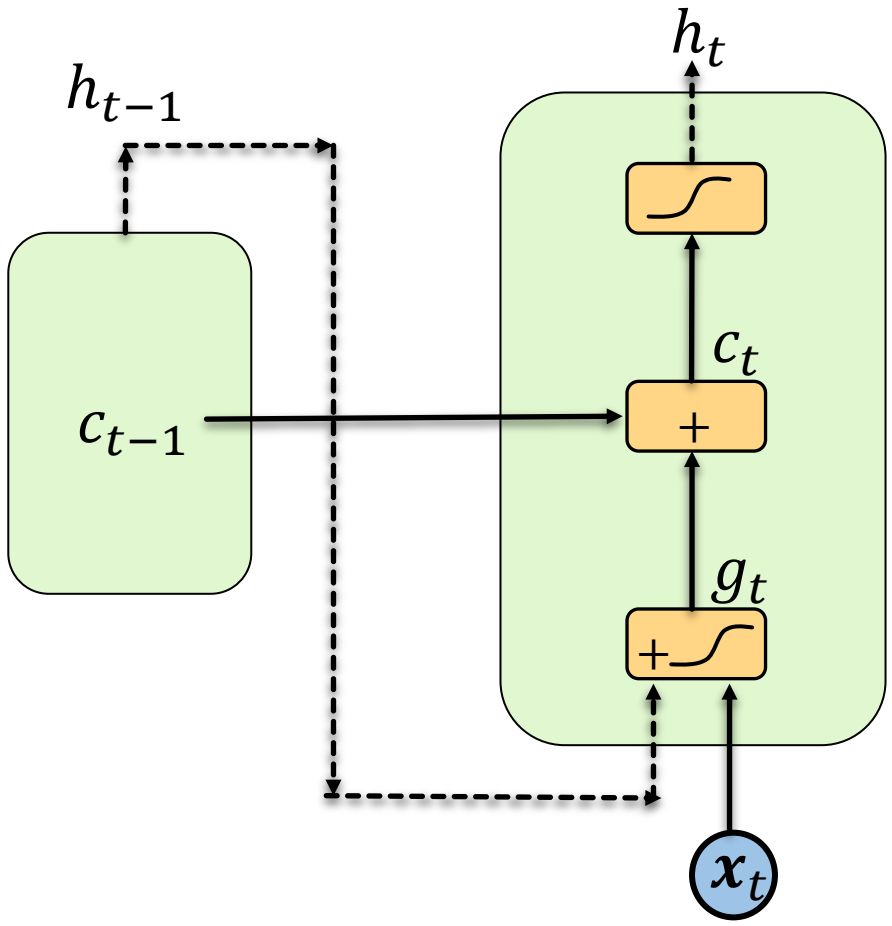
# Long-Short Term Memory (LSTM) Cell




Let's start from the vanilla RNN unit

S. Hochreiter, J. Schmidhuber, Long short-term memory". Neural Computation, Neural Comp. 1997

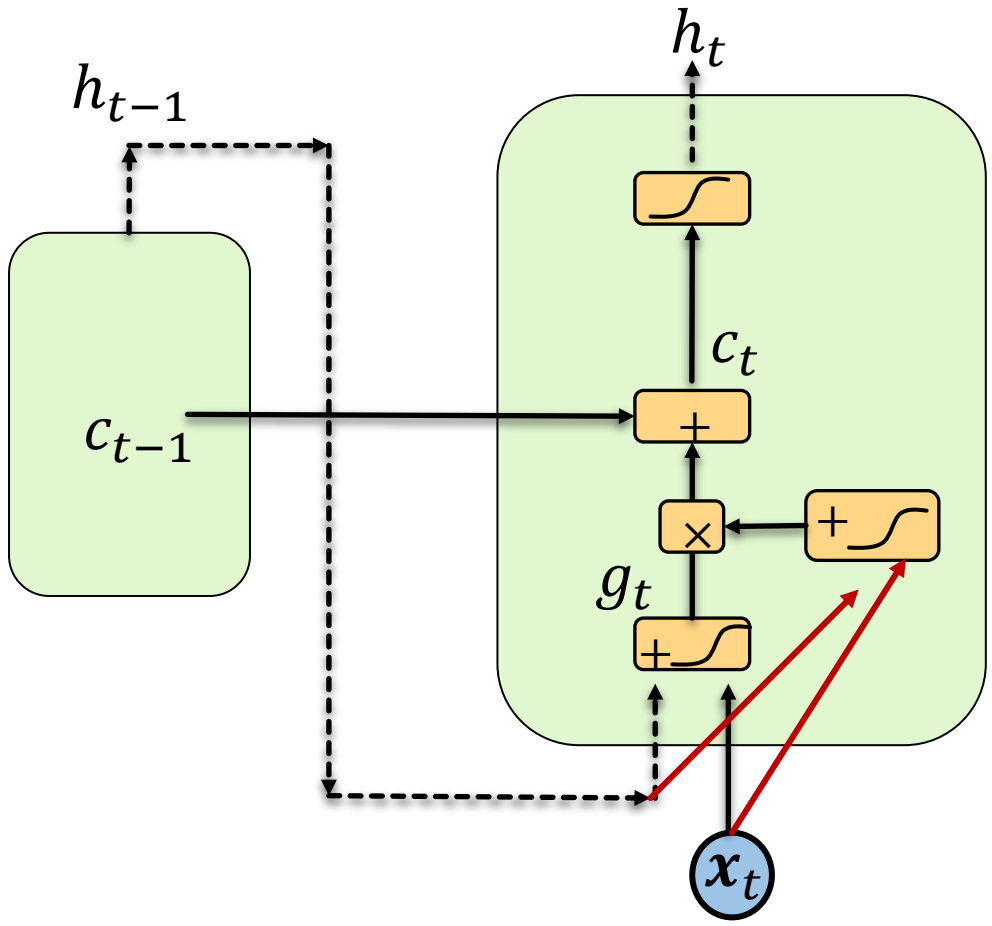
# LSTM Design – Step 1




Introduce a linear/ identity memory $c_t$

Combines past internal state $c_{t-1}$ with current input $\boldsymbol{x}_t$

# LSTM Design – Step 2 (Gates)



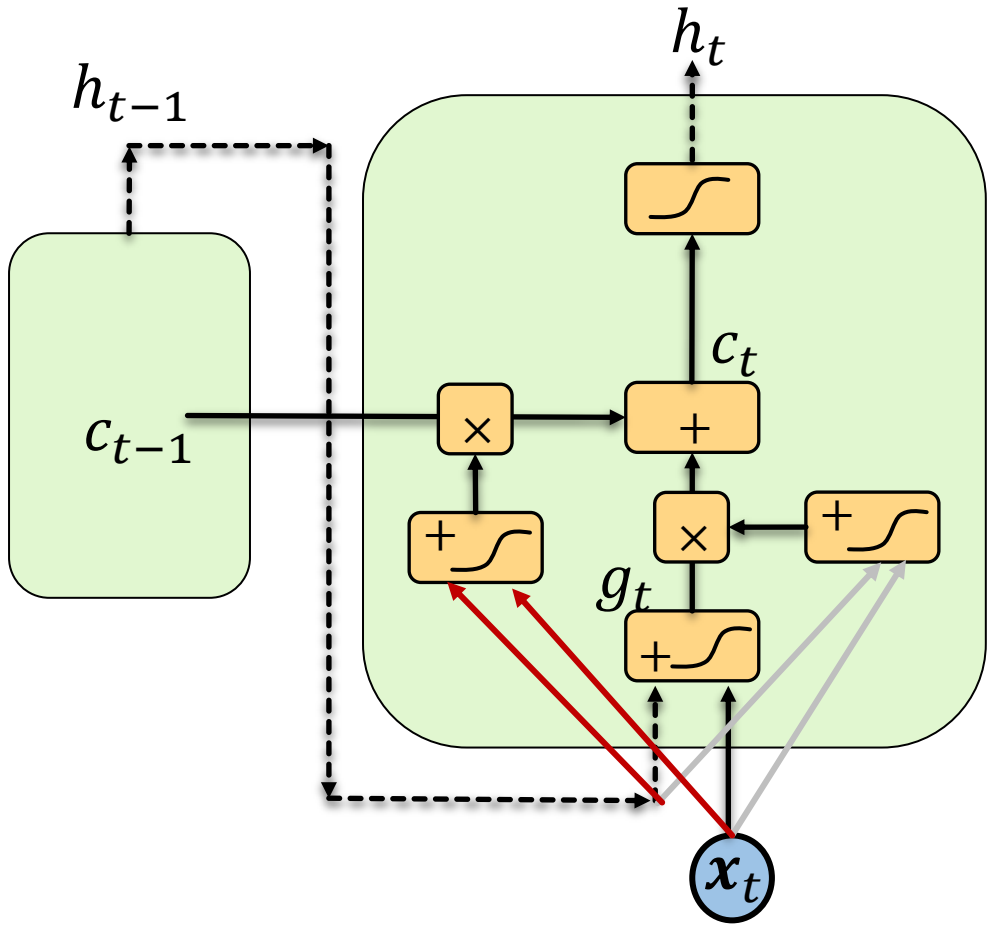

Input gate

Controls how inputs contribute to the internal state

$$I_t(x_t, h_{t-1})$$

Logistic sigmoid

# LSTM Design – Step 2 (Gates)



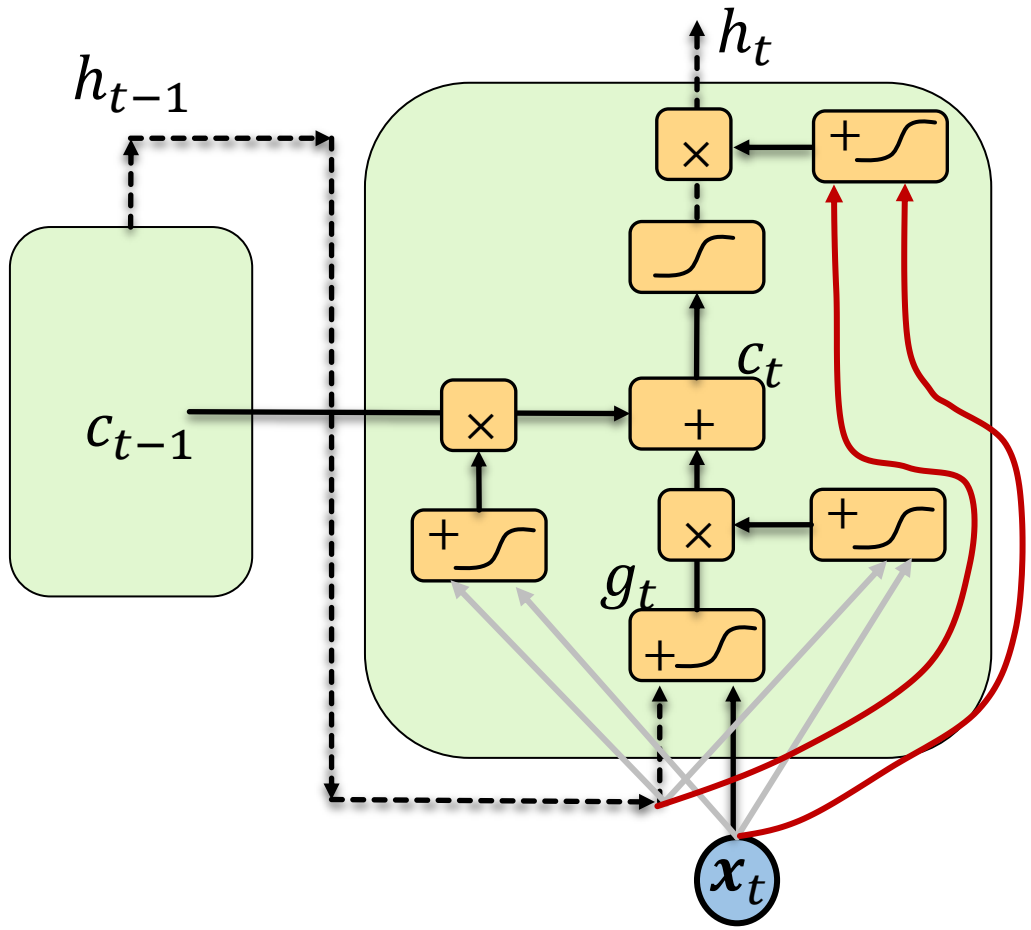

Forget gate

Controls how past internal state $c_{t-1}$ contributes to $c_t$

$$F_t(x_t, h_{t-1})$$

Logistic sigmoid

# LSTM Design – Step 2 (Gates)




## Output gate

Controls what part of the internal state is propagated out of the cell

$$O_t(x_t, h_{t-1})$$

Logistic sigmoid

# LSTM in Equations

1) Compute activation of input and forget gates

$$\boldsymbol{I}_t = \sigma(\boldsymbol{W}_{Ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Iin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{I})$$

$$\boldsymbol{F}_t = \sigma(\boldsymbol{W}_{Fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Fin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{F})$$

2) Compute input potential and internal state

$$\boldsymbol{g}_t = tanh(\boldsymbol{W}_{h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{x}_t + \mathbf{b}_\mathrm{h})$$

$$\boldsymbol{c}_t = \boldsymbol{F}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{I}_t \odot \boldsymbol{g}_t$$

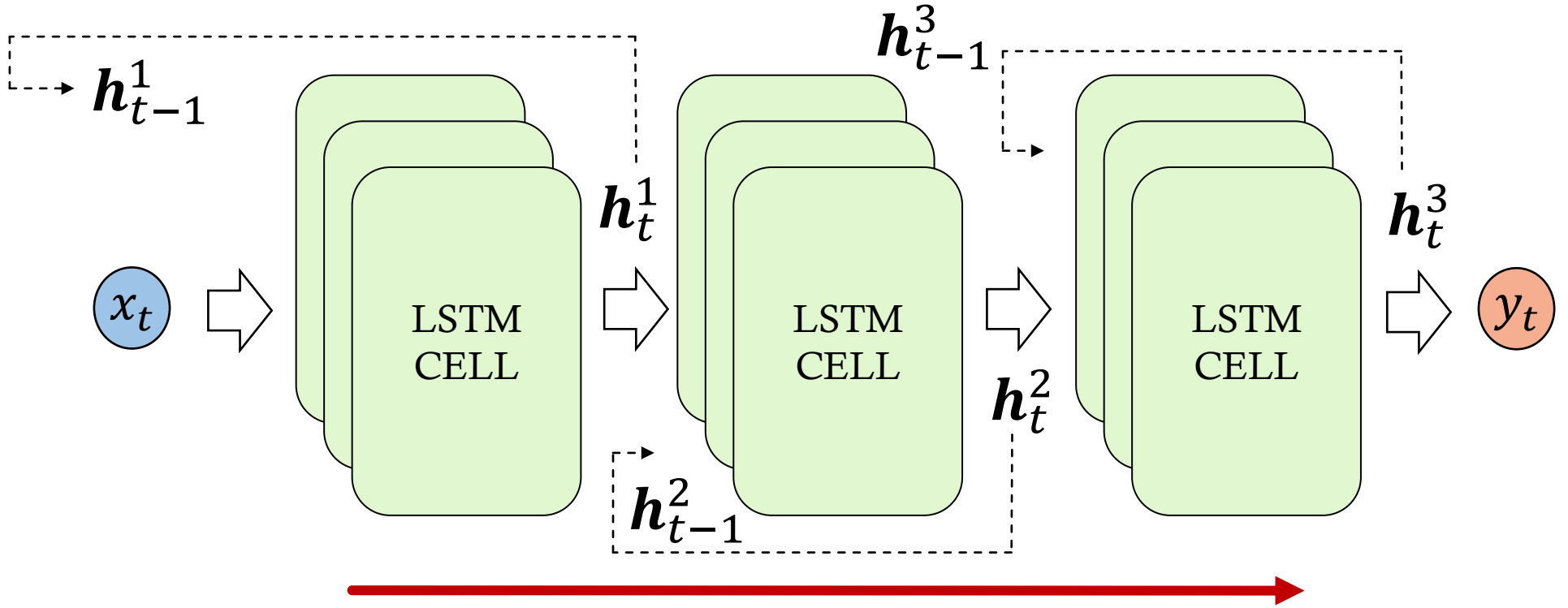
3) Compute output gate and output state

$$\boldsymbol{O}_t = \sigma(\boldsymbol{W}_{Oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Oin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{O})$$

$$\boldsymbol{h}_t = \boldsymbol{O}_t \odot tanh(\boldsymbol{c}_t)$$

⊙ element-wise multiplication

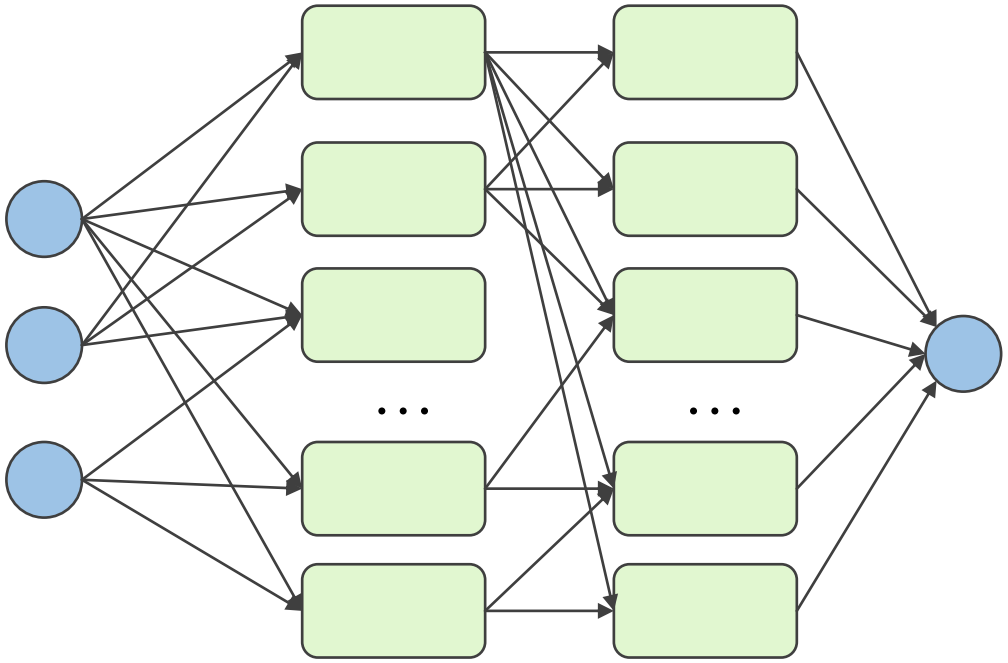
# Deep LSTM




LSTM layers extract information at increasing levels of abstraction (enlarging context)

# Regularizing LSTM - Dropout

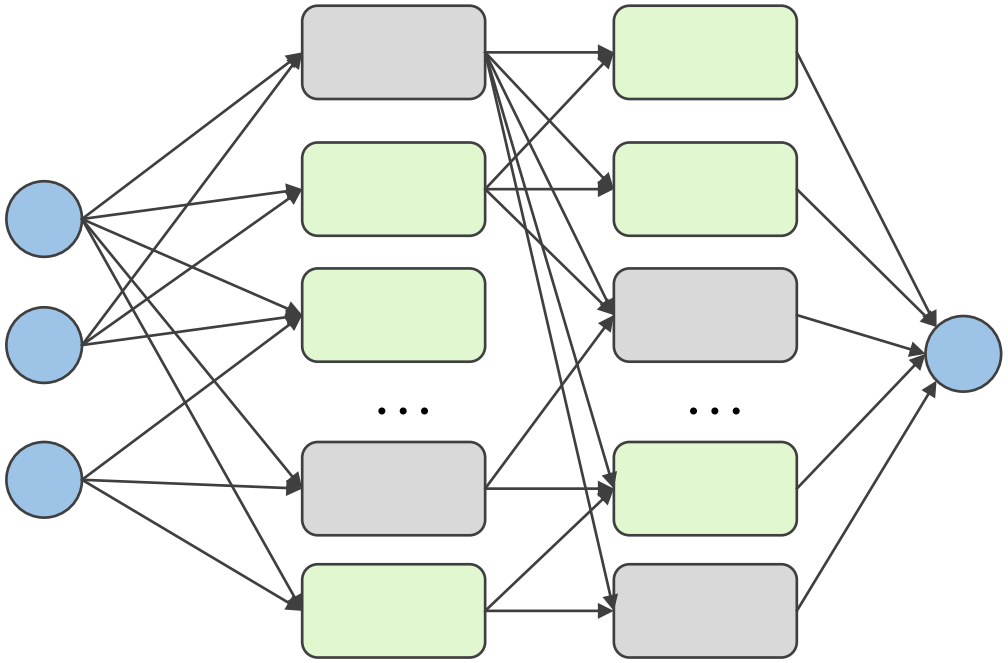
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

# Regularizing LSTM - Dropout

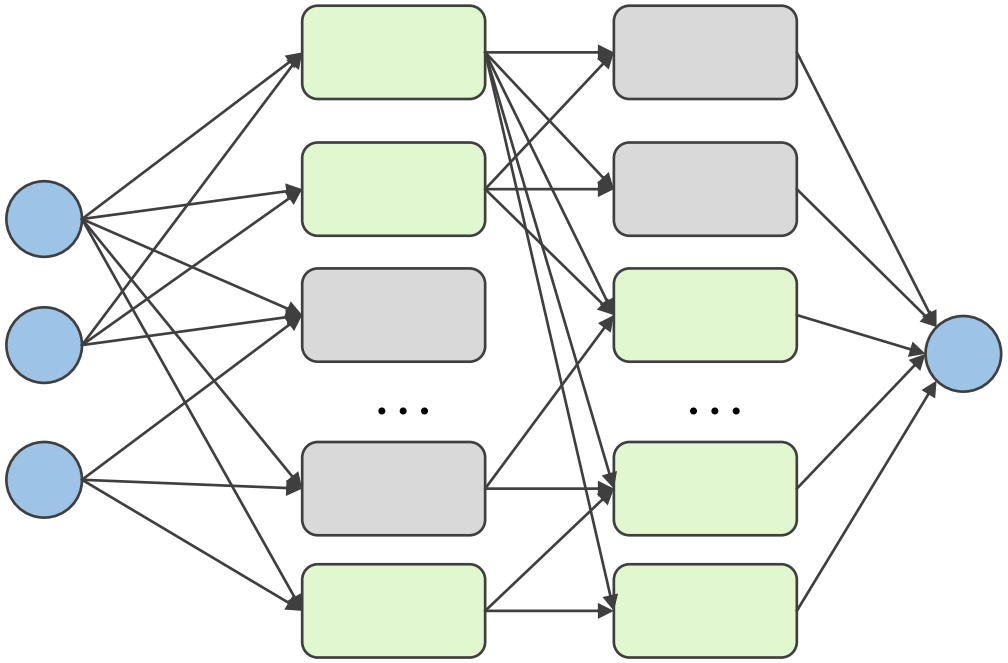
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

# Regularizing LSTM - Dropout

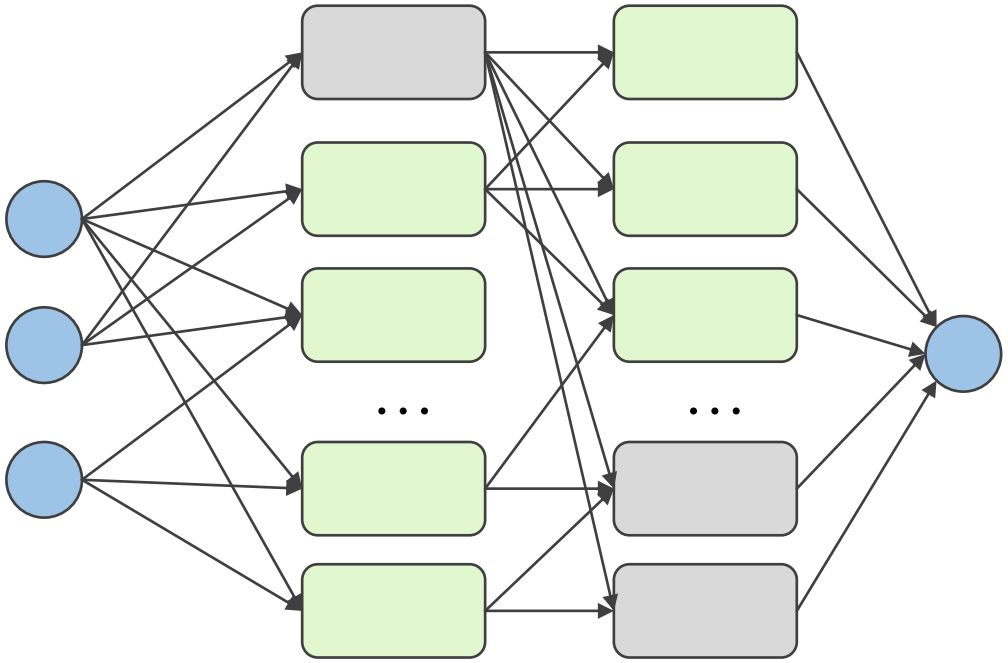
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

# Regularizing LSTM - Dropout

Randomly disconnect units from the network during training

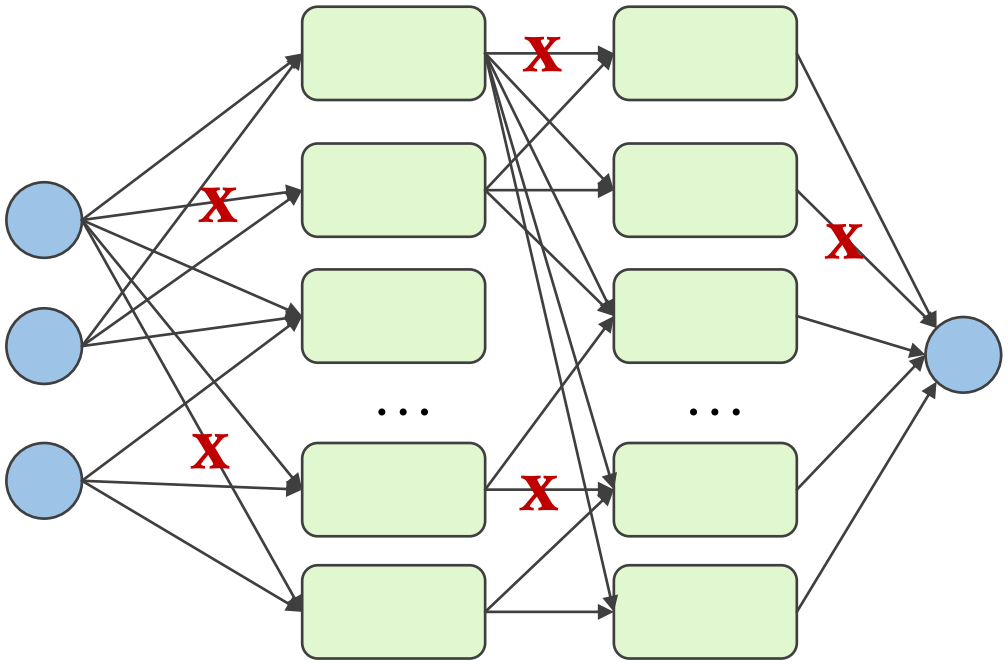


- Regulated by unit dropping hyperparameter
- Prevents unit coadaptation
- Committee machine effect
- Need to adapt prediction phase
- Drop units for the whole sequence!

N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

# Regularizing LSTM - Dropout

Randomly disconnect units from the network during training



- Regulated by unit dropping hyperparameter
- Prevents unit coadaptation
- Committee machine effect
- Need to adapt prediction phase
- Drop units for the whole sequence!

You can also drop single connections (dropconnect)

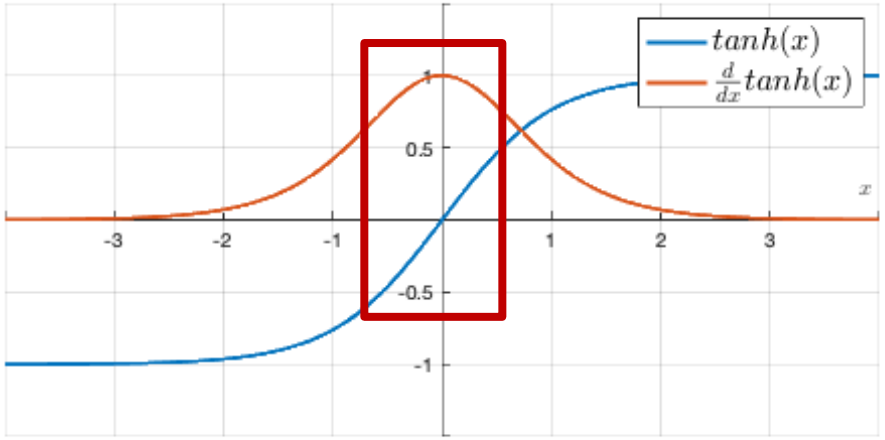
# Activity Regularization

- Penalize the model's activations
- High or low activations saturate the tanh function (zero gradient)
- L2 activity regularization

$$\alpha \| \mathbf{M} \odot \boldsymbol{h}^t \|_2^2$$

- Temporal activity regularization

$$\beta \| \boldsymbol{h}^t - \boldsymbol{h}^{t+1} \|_2^2$$



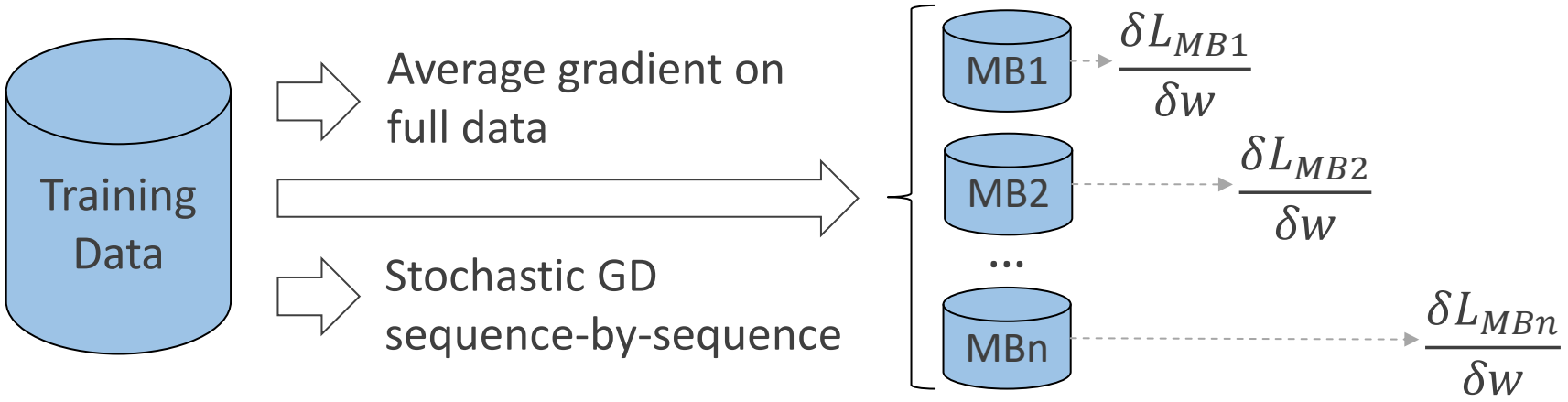

Applies L2 regularization only to non-dropped out neurons to keep activations low

Encourages temporal consistency

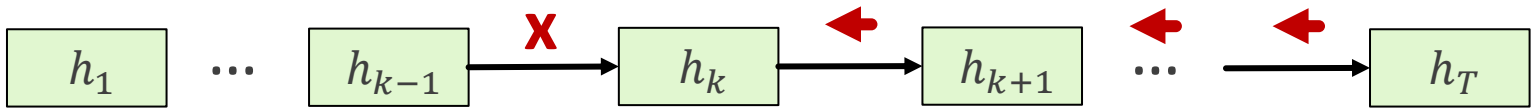
Revisiting Activation Regularization for Language RNNs, https://arxiv.org/pdf/1708.01009.pdf

# Practicalities – Minibatch and Truncated BP

## Minibatch (MB)




## Truncated gradient propagation



# Gated Recurrent Unit (GRU)

Reset acts directly on output state (no internal state and no output gate)

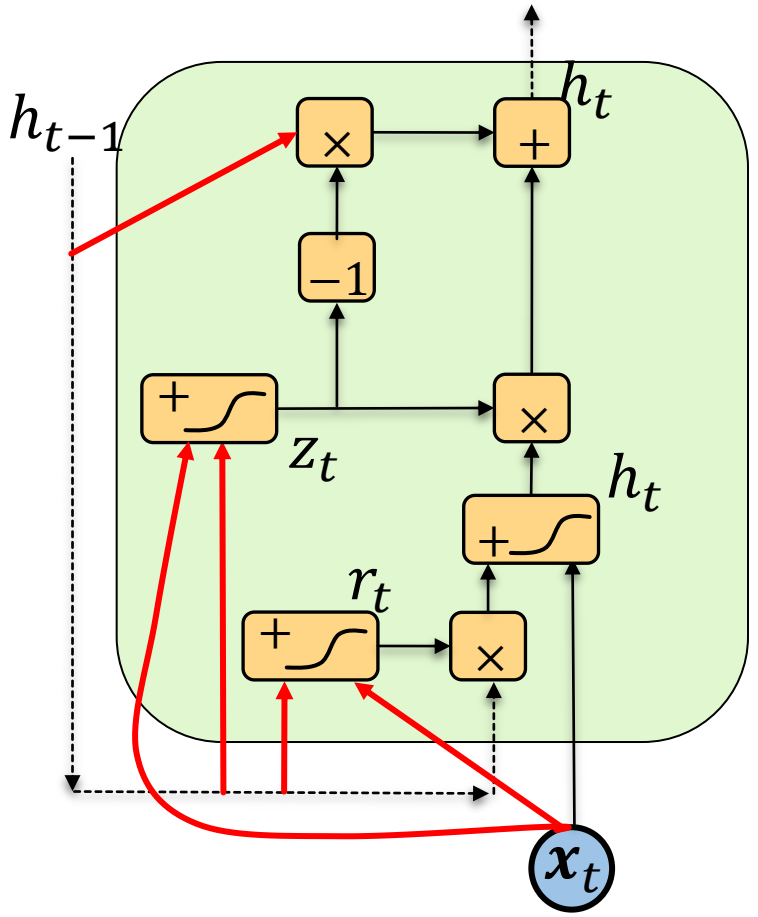
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \boldsymbol{h}_t$$

$$\boldsymbol{h}_t = tanh(\boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_{hin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{h})$$

Reset and update gates when coupled act as input and forget gates

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_{zh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{zin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{z})$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{rh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{rin}\boldsymbol{x}_t + \mathbf{b}_\mathrm{r})$$




*C. Kyunghyun et al, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, EMNLP 2014*

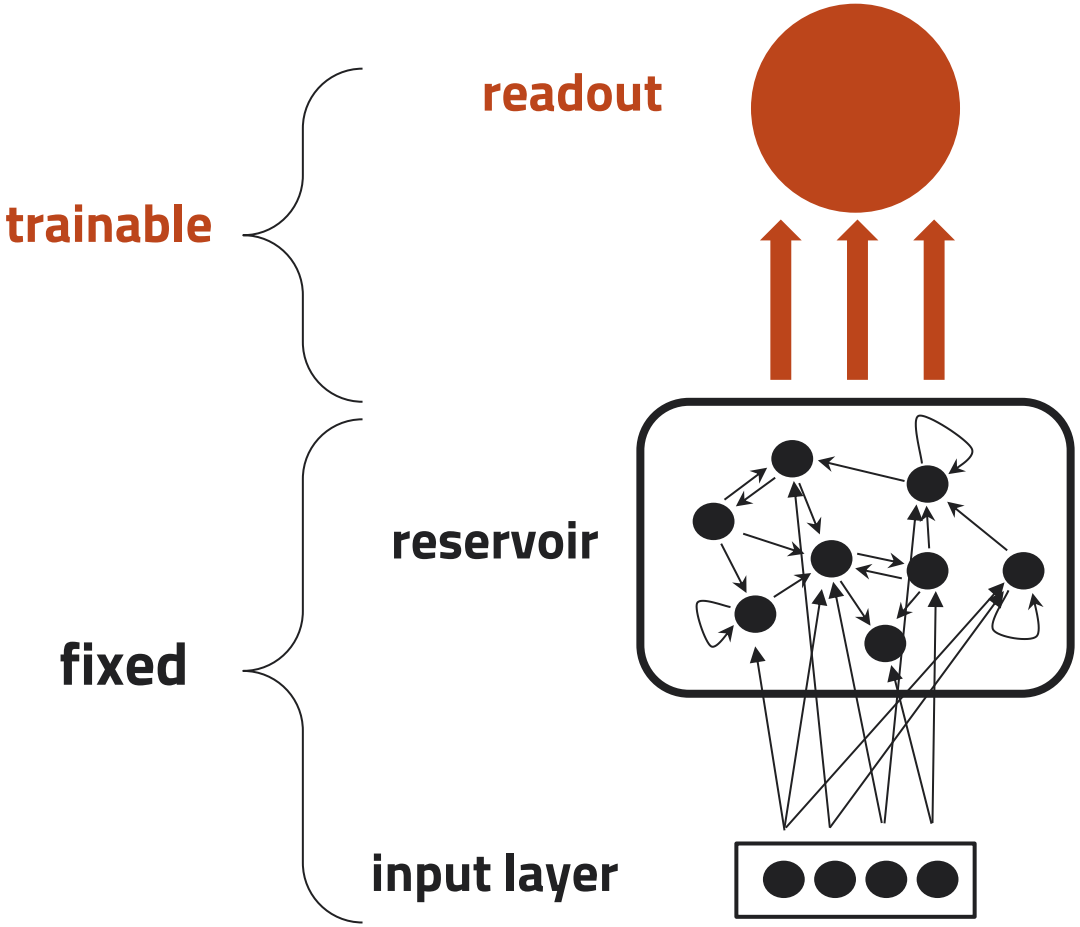
# Randomized Architectures

# Tackling the issue at its core

Deep neural networks find useful representations of data by applying multiple non-linear levels of transformation

Deep Learning = Architectural Biases + Learning Algorithms

# Reservoir Computing: focus on the shaping memory properties




randomized

$$\boldsymbol{h}_t = \tanh(\boldsymbol{x}_t W_{in} + \boldsymbol{h}_{t-1} W_h)$$

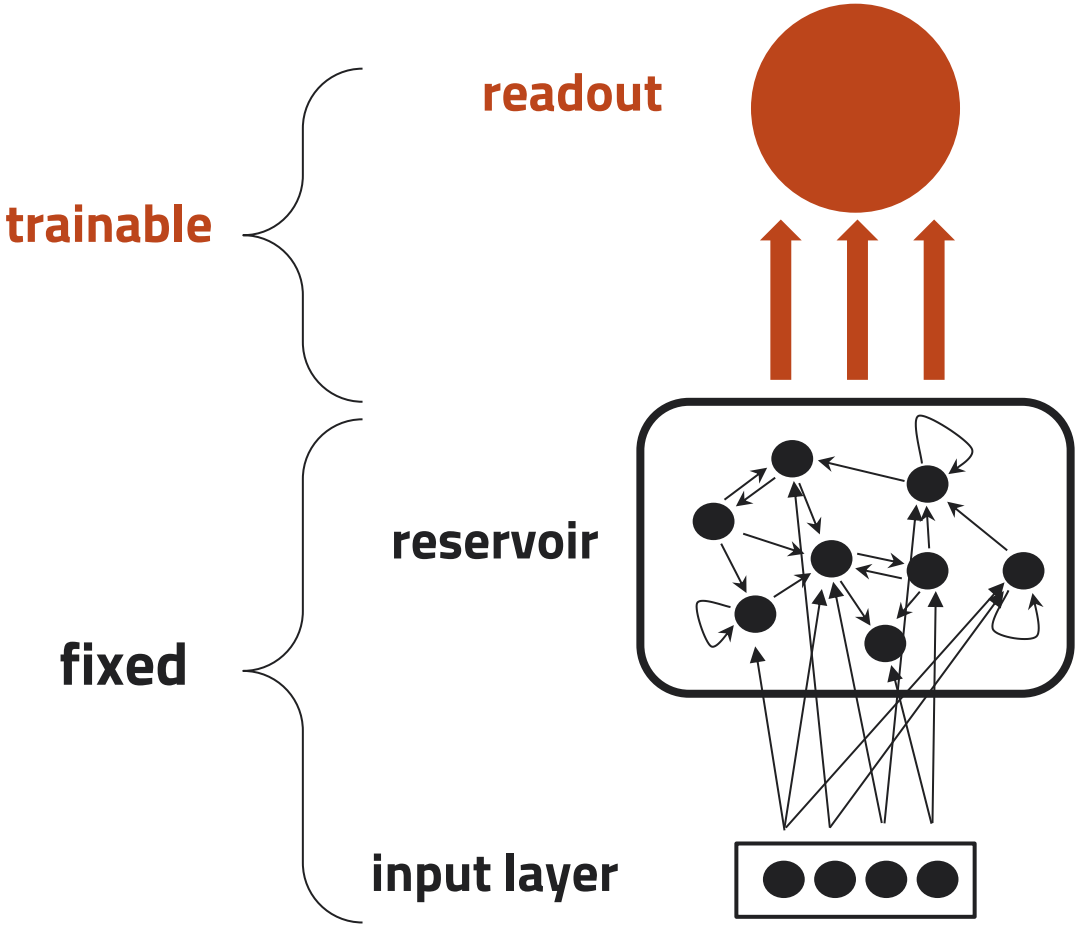
Key difference from standard RNNs:

- $W_h$ and $W_{in}$ are fixed (randomly initialized)
- Only output weights $W_{out}$ are trained
- (Often) Simple linear readout

$$\boldsymbol{y}_t = W_{out} \boldsymbol{h}_t$$

Verstraeten, David, et al. *Neural networks* 20.3 (2007).
Lukoševičius, Mantas, and Herbert Jaeger. *Computer Science Review* 3.3 (2009).

# Reservoir Computing: focus on the shaping memory properties




randomized

$$\boldsymbol{h}_t = \tanh(\boldsymbol{x}_t W_{in} + \boldsymbol{h}_{t-1} W_h)$$

Randomly initialized under stability conditions on the dynamic memory

Stable dynamics => Echo State Property

Verstraeten, David, et al. *Neural networks* 20.3 (2007).
Lukoševičius, Mantas, and Herbert Jaeger. *Computer Science Review* 3.3 (2009).

# Echo State Property (ESP)

- The network has the Echo State Property if

$$\lim_{k\to\infty} \left\| \frac{\partial h_t}{\partial h_{t-k}} \right\| = 0$$

  - The effect of initial conditions vanishes
  - The current state depends only on the input history (memory)
- Can again be checked by looking at the state Jacobian $J_i$

$$\frac{\partial h_t}{\partial h_{t-k}} = \prod_{i=t-k+1}^{t} J_i$$

- ESP requires this product to *contract on average*

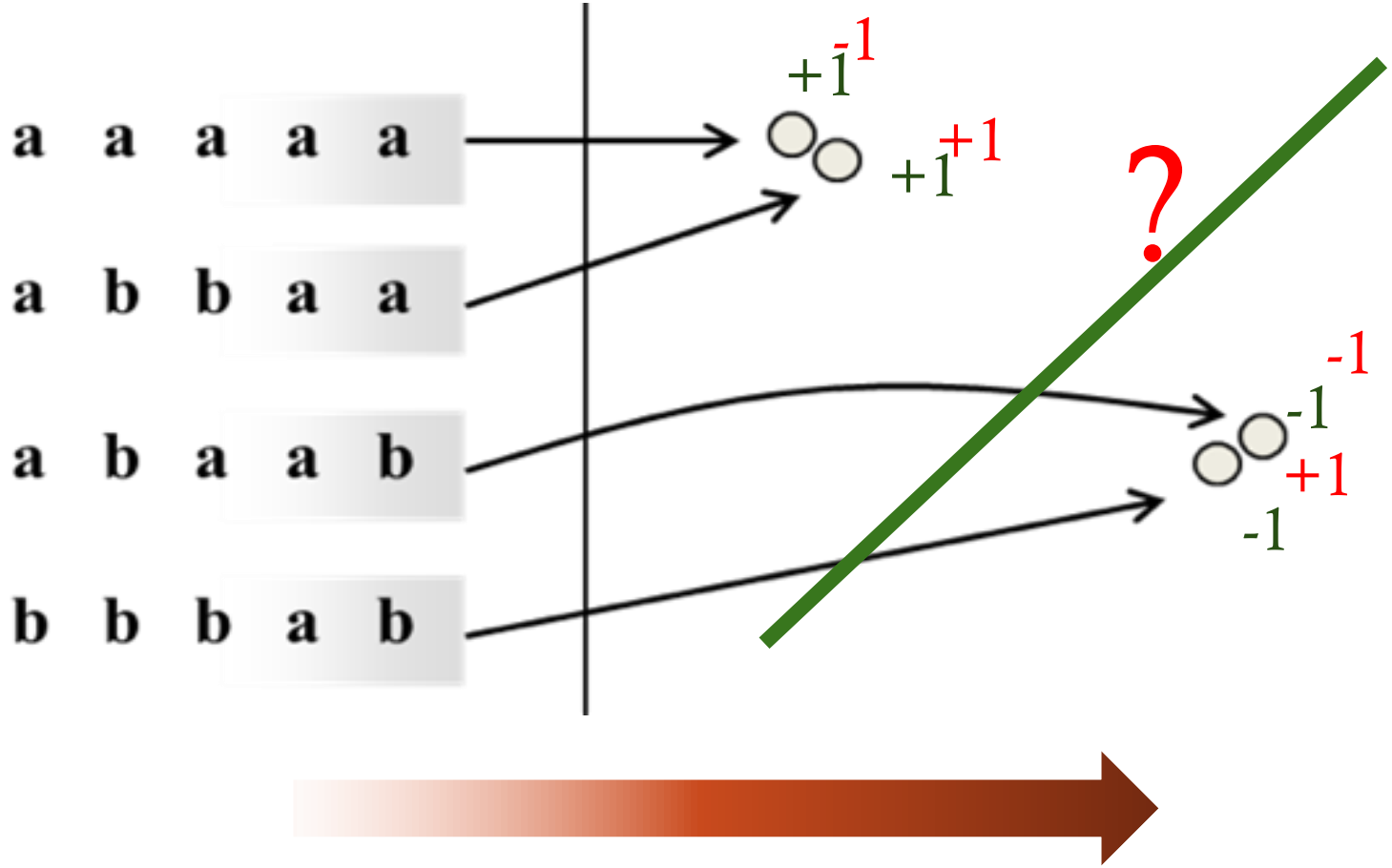
# Controlling memory and the ESP

- The ESP essentially entails that our memory is lossy => forgets about *old* data (initial conditions) to favour remembering *recent* input
- With some simplifications the ESP (and the memory) can be controlled through the spectral radius of $W_h$
  - If $\rho(W_h) \ll 1 \rightarrow$ fast memory decay (contractive behaviour)
  - If $\rho(W_h) \approx 1 \rightarrow$ slow memory decay
  - If $\rho(W_h) > 1 \rightarrow$ instability (chaotic behaviour)
- Echo State Networks (ESNs)– A model of the reservoir computing paradigm using random input and recurrent state matrix, where the latter is initialized to be in the contractive spectrum

# The Randomized Reservoir “catch”

- Non-linearly embed the input into a higher dimensional feature space where the original problem is more likely to be solved linearly (Cover’s Theorem)
- Randomized basis expansion computed by a pool of randomized filters
- Provides a “rich” set of input-driven dynamics

# Why does it work?




Suffix-based Markovian organization of the state space of contraction reservoir mappings even without learning

# ESN – In practice

- Initialize $W_h \sim Sparse(N, N)$, $W_{in} \sim U(-a, a)$ (typically $a \in [0.1, 0.4]$)
- Rescale $W_h$ to desidered spectral radius $\rho_{des}$ (typically $\rho_{des} \in [0.8, 0.99]$)

$$W_h \leftarrow \frac{\rho_{des}}{\rho(W_h)} W_h$$

- Run the network on the entire input sequence and collect the reservoir states

$$\mathbf{H} \leftarrow [\mathbf{h_1}, \dots, \mathbf{h}_T]$$

- Washout the initial transient

$$\mathbf{H} \leftarrow \mathbf{H}(:, T_w : T)$$

- Collect the target data similarly into a matrix

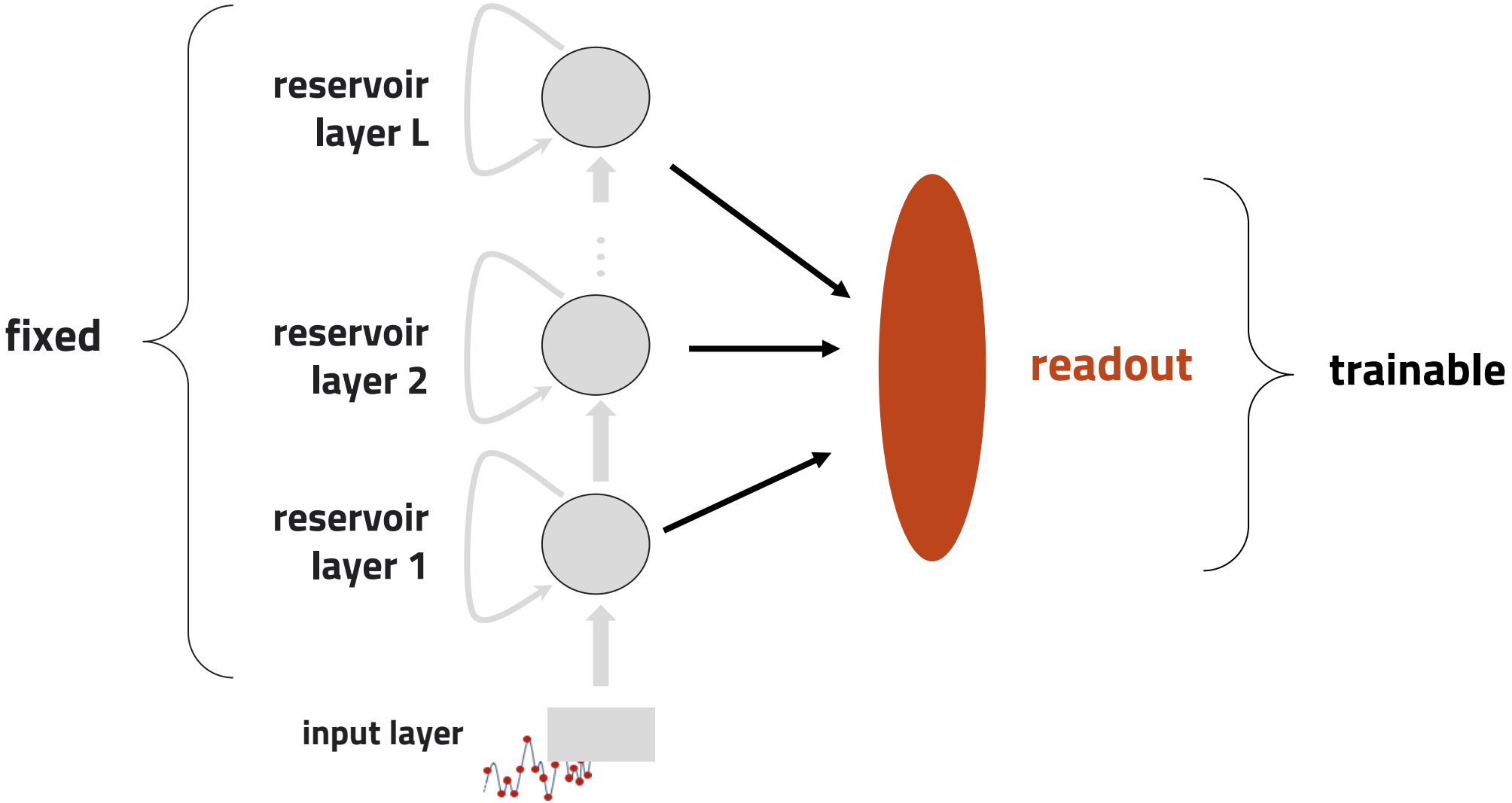
$$\mathbf{Y} \leftarrow [\mathbf{y}_{T_w}, \dots, \mathbf{y}_T]$$

- Solve the linear regression problem for the readout

$$\min_{\mathbf{W_o}} \|\mathbf{W_o}\,\mathbf{H} - \mathbf{Y}\|_2^2 \Rightarrow \boldsymbol{W_o} = \boldsymbol{Y H^T (H H^T + \lambda I)^{-1}}$$

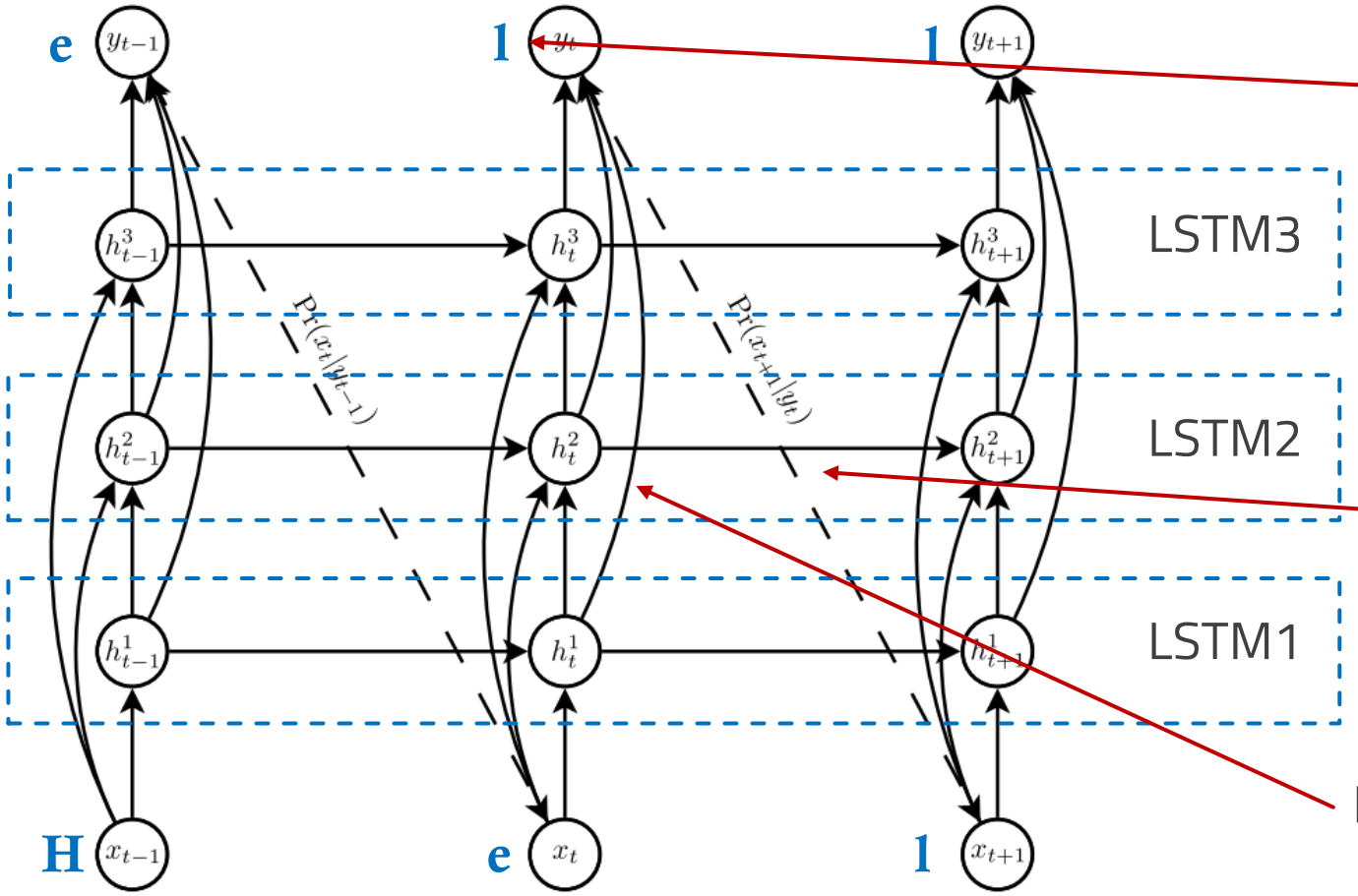
Classical ridge regression solution to least mean square

# Deep Echo State Networks



# Advanced topics and wrap-up

# LSTM – (Primitive) Language Modeling



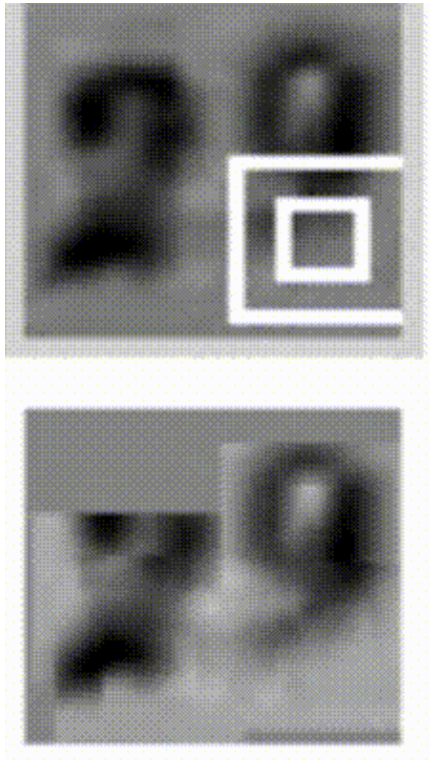

Teacher forcing at training time

Refeeding output at prediction time

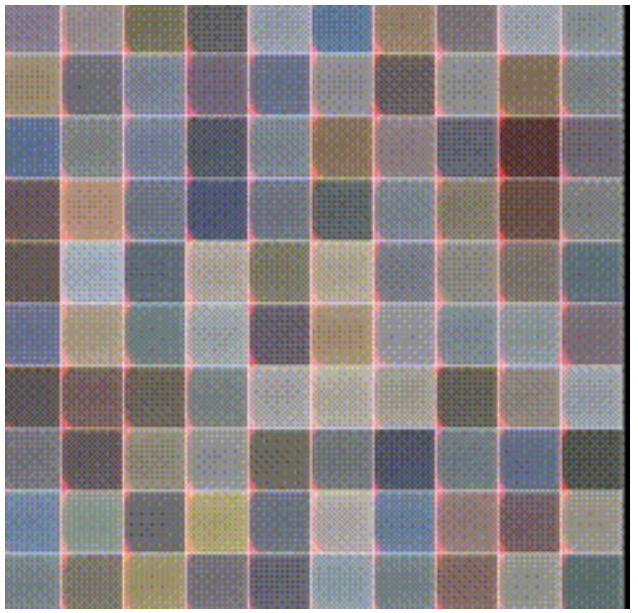
Bypass connections

A. Graves, Generating Sequences With Recurrent Neural Networks, 2013

# Autoregressive RNN – A Broader View



RNN are only for sequential data?



RNN as stateful systems

# Software

Standard LSTM and GRU are available in all deep learning frameworks

- If you want to play with one-element ahead sequence generation, try out char-RNN implementations
  - https://github.com/karpathy/char-rnn (ORIGINAL)
  - https://github.com/sherjilozair/char-rnn-tensorflow
  - https://github.com/crazydonkey200/tensorflow-char-rnn
  - http://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html

Randomized RNNs are less consolidated in code

- No official implementations but a few consolidated Python libraries
  - https://github.com/reservoirpy/reservoirpy
  - https://pypi.org/project/reservoir-computing/
  - https://github.com/lucapedrelli/DeepESN
  - https://github.com/siloekse/PythonESN

# Take Home Messages

- Approaches addressing (circumventing) the difficulty of learning long-term dependencies due to gradient vanish/explosion
- Gated RNN solution
  - Gates are neurons whose output is used to scale another neuron's output
  - Use gates to determine what information can enter (or exit) the internal state
  - Training gated RNN non always straightforward
- Randomized approaches
  - Training-efficient approach to recurrent neural modelling
  - Leverages theoretical properties on memory capacity + high dimensional input projection
- Deep RNN can be used in generative mode
  - Deep RNN as stateful and differentiable machines

# Upcoming Lecture

- Advanced sequential tasks: sequence-to-sequence
- Encoder-decoder recurrent architectures
- Cross-attention mechanism