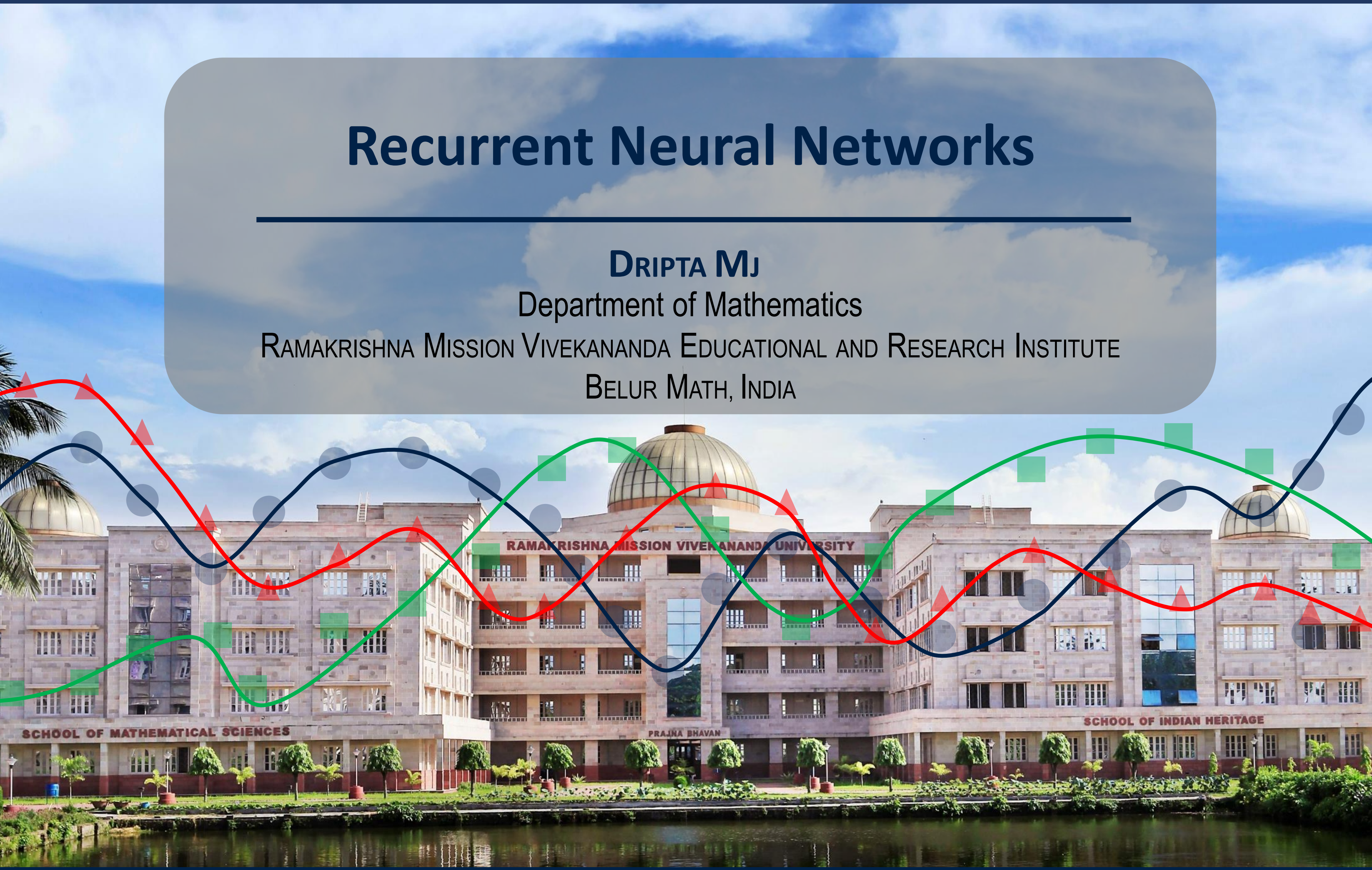
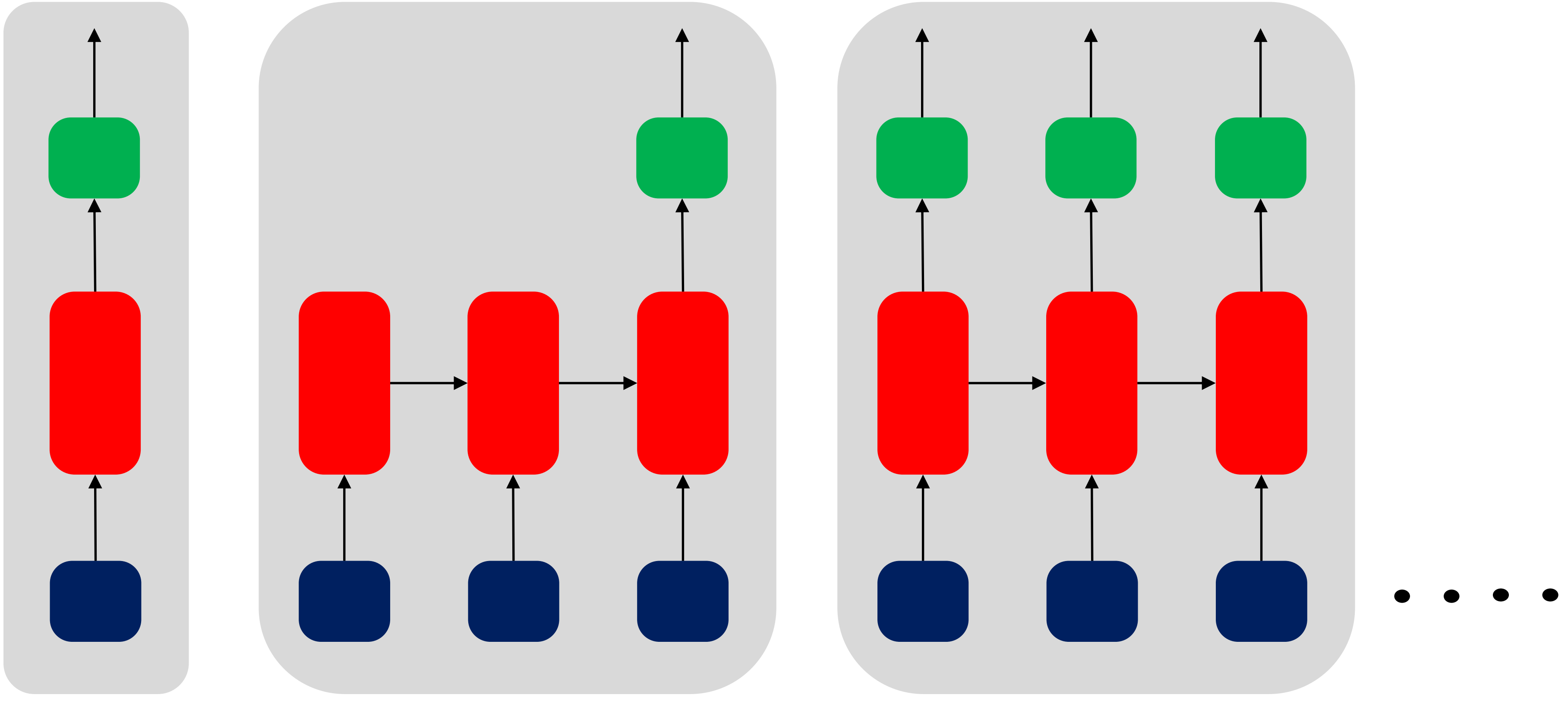
Recurrent Neural Networks

Dripta Mj
Department of Mathematics
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, India



Introduction



*Some slides inspired from Bhiksha Raj (CMU) and Christopher Olah.

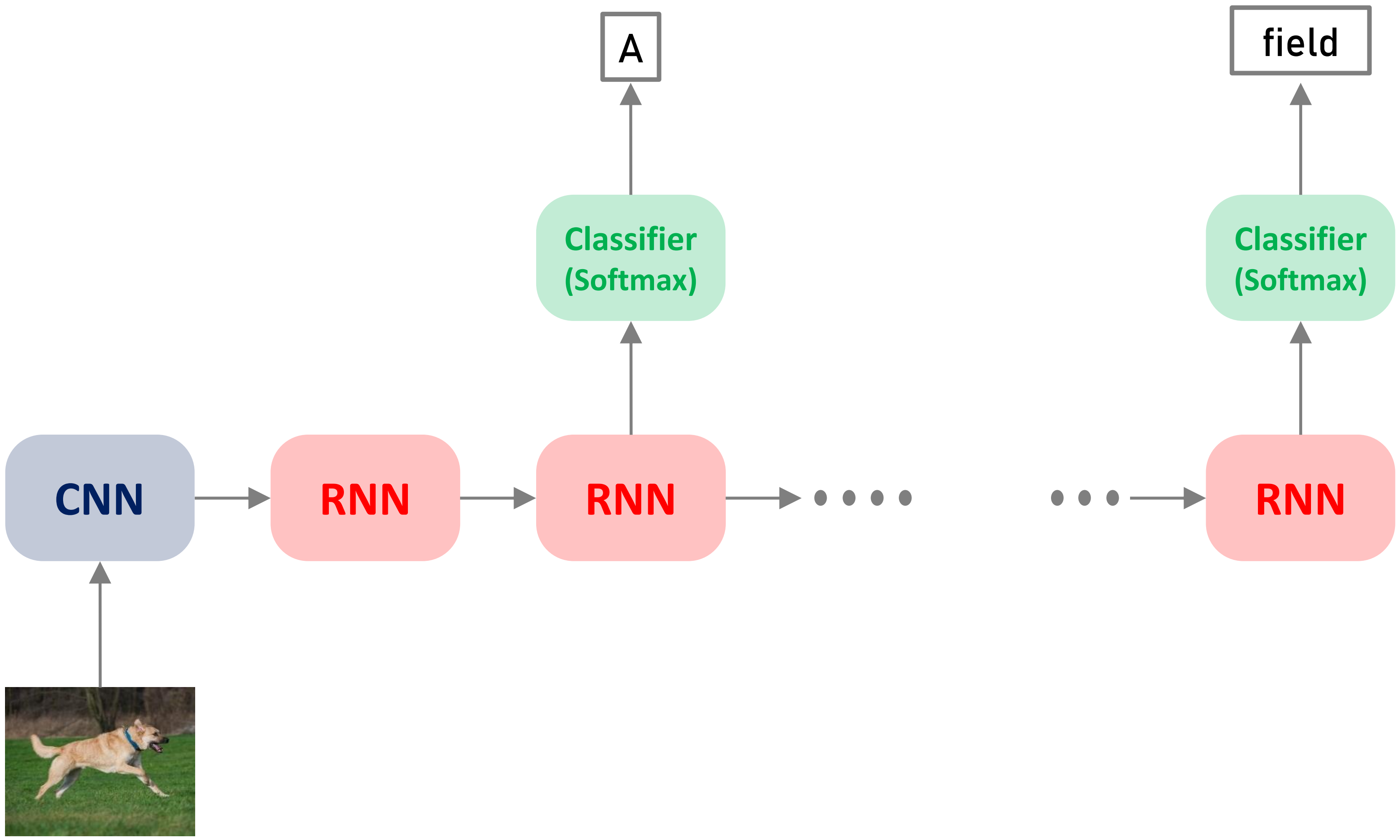
Introduction

- Specialized family of neural networks for processing sequential data.
- In many problems the inputs and outputs are not of fixed length.
- Recurrent neural networks (RNNs) can use their internal state to process sequences of inputs.
- Intermediate states of RNNs store historical information.
- Takes an input or an input sequence and gives an output or an output series. Examples
  - Language translation: Inputs and outputs are both sequences of words.
  - Image captioning.

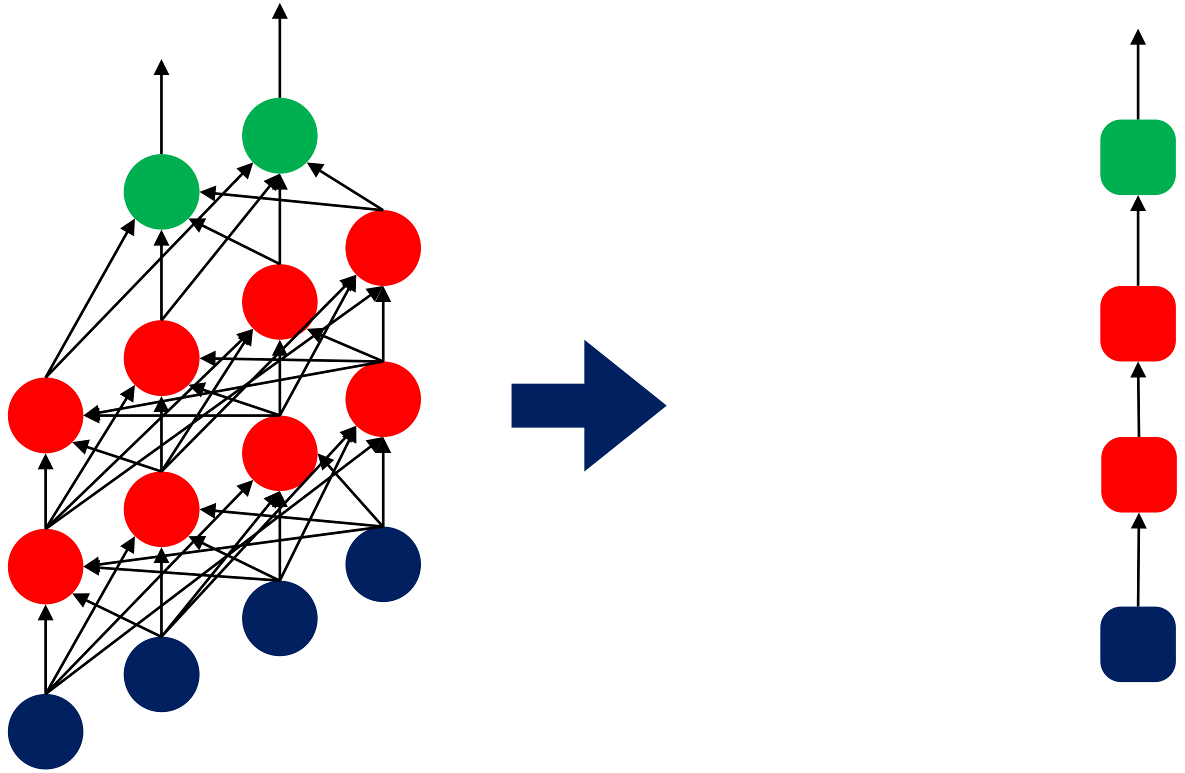
Image captioning

A dog running across the field

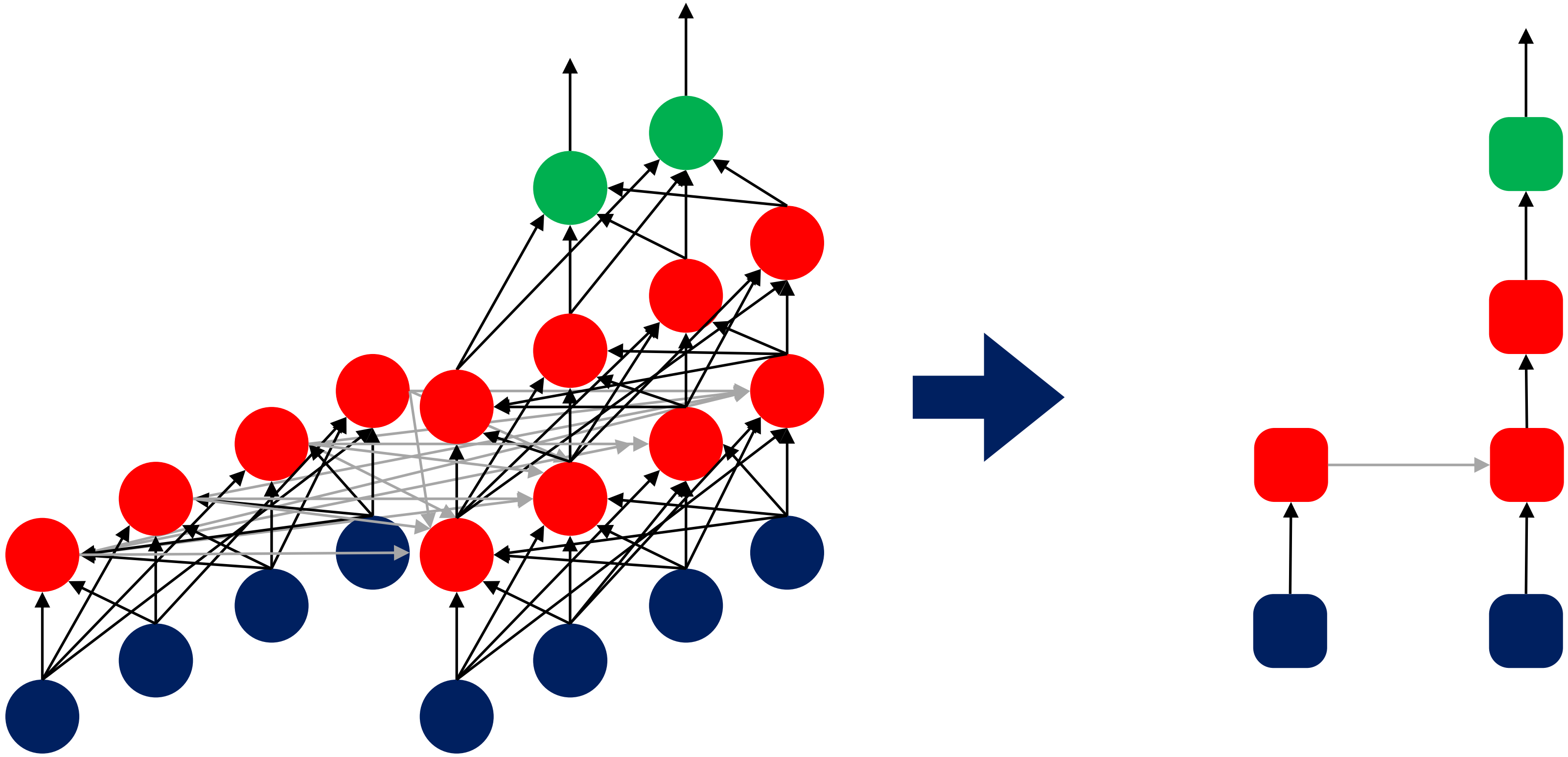
Image captioning



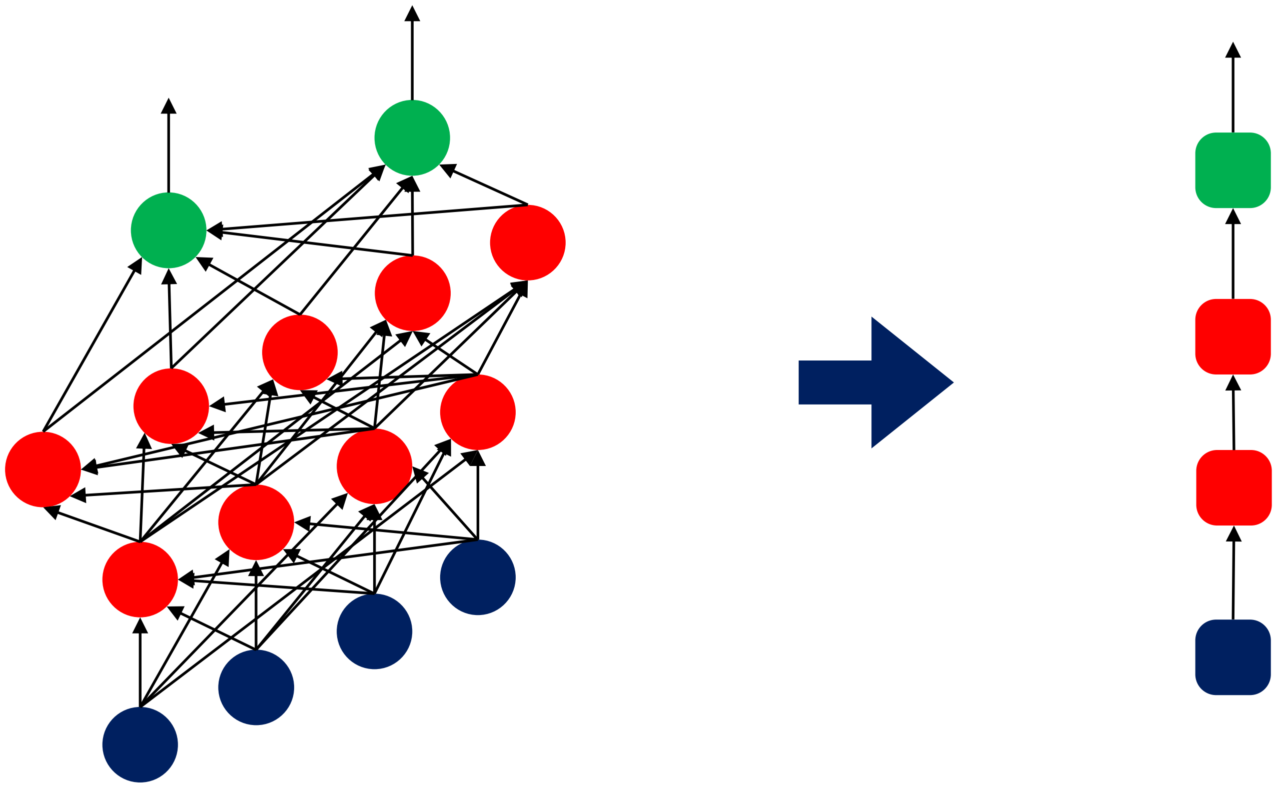
Representation



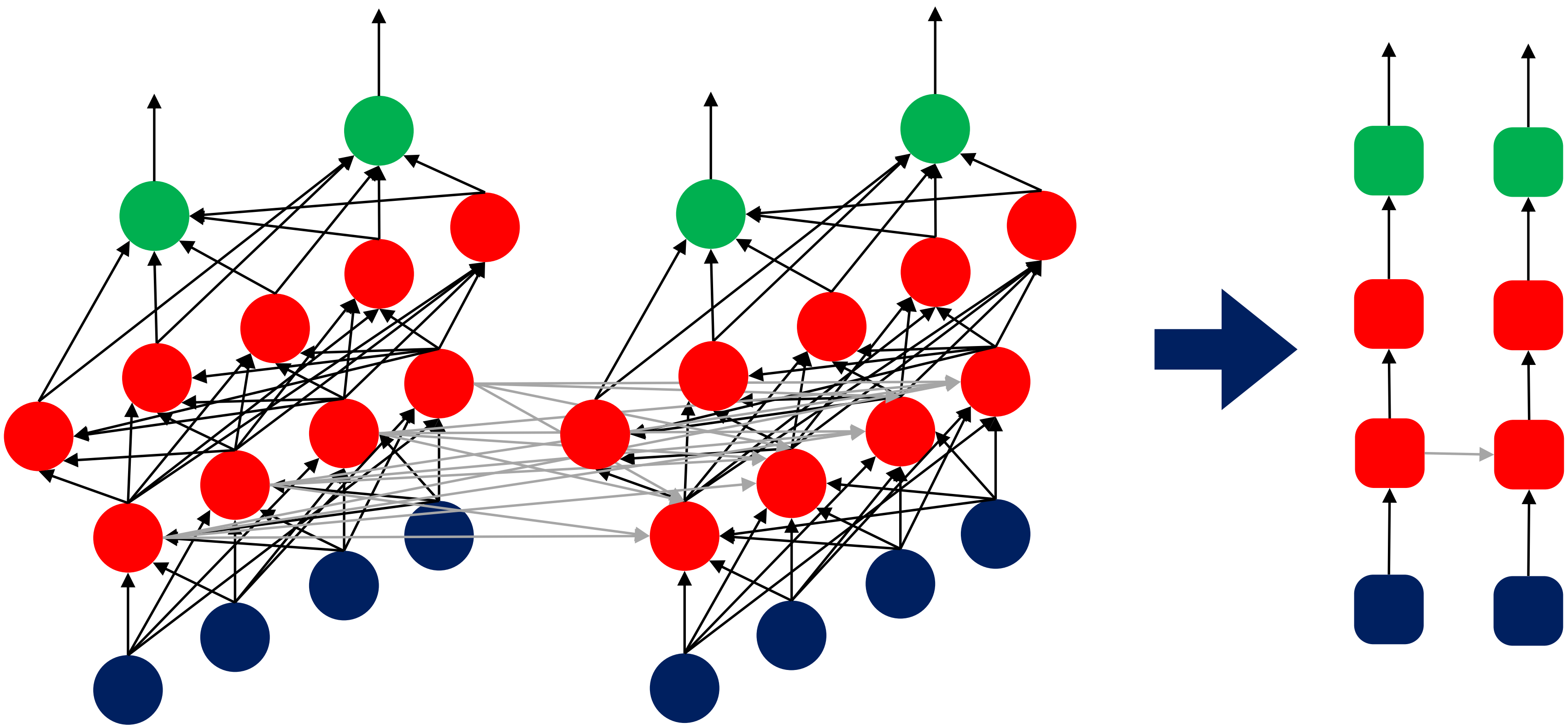
Representation



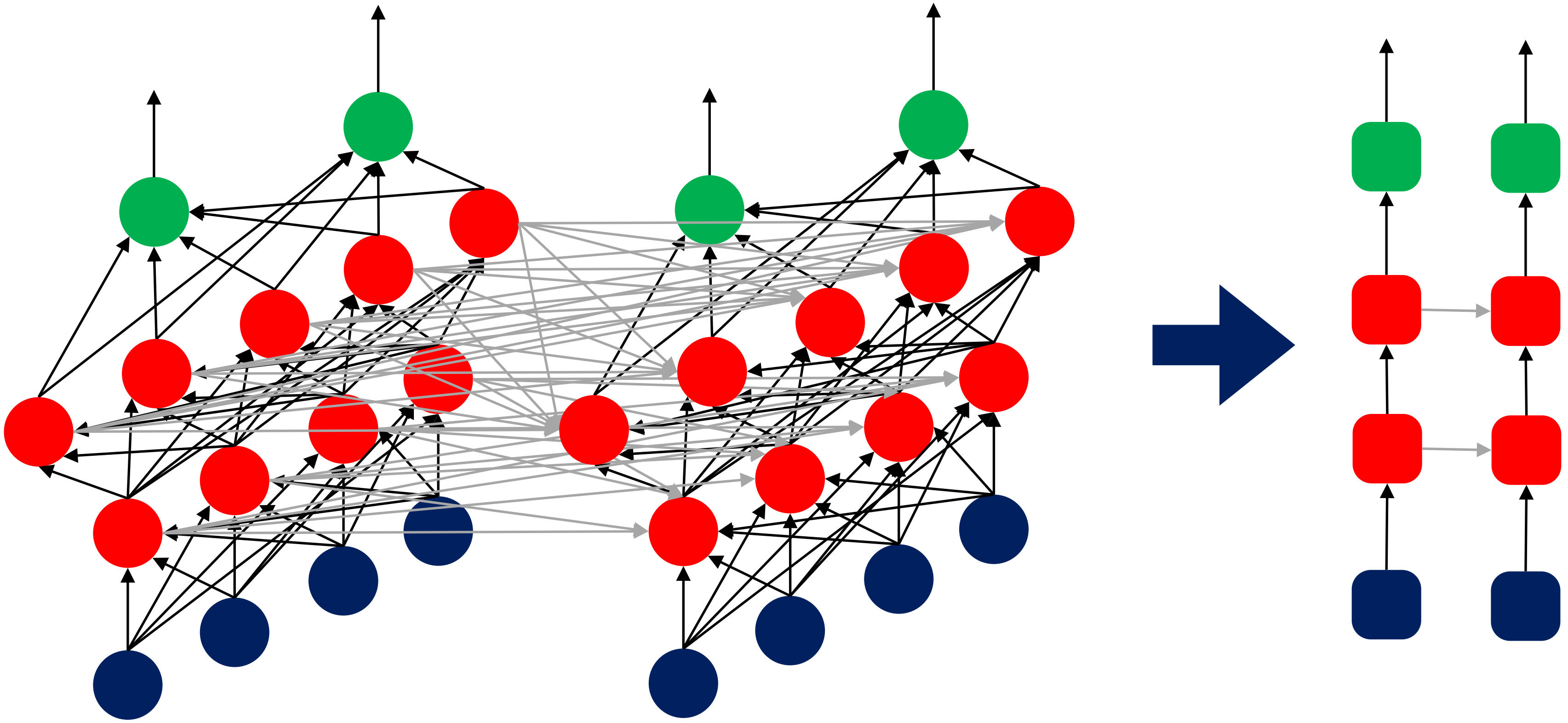
Representation



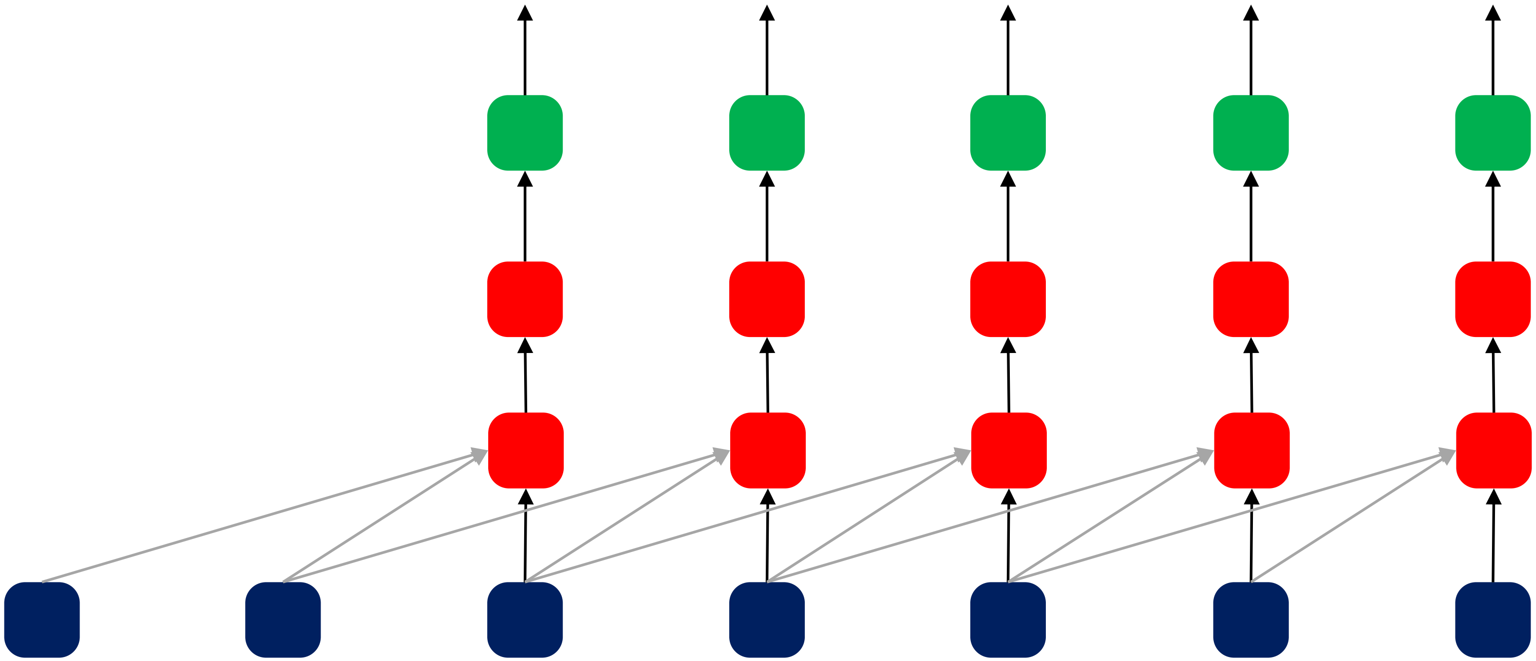
Representation



Representation



Finite response system

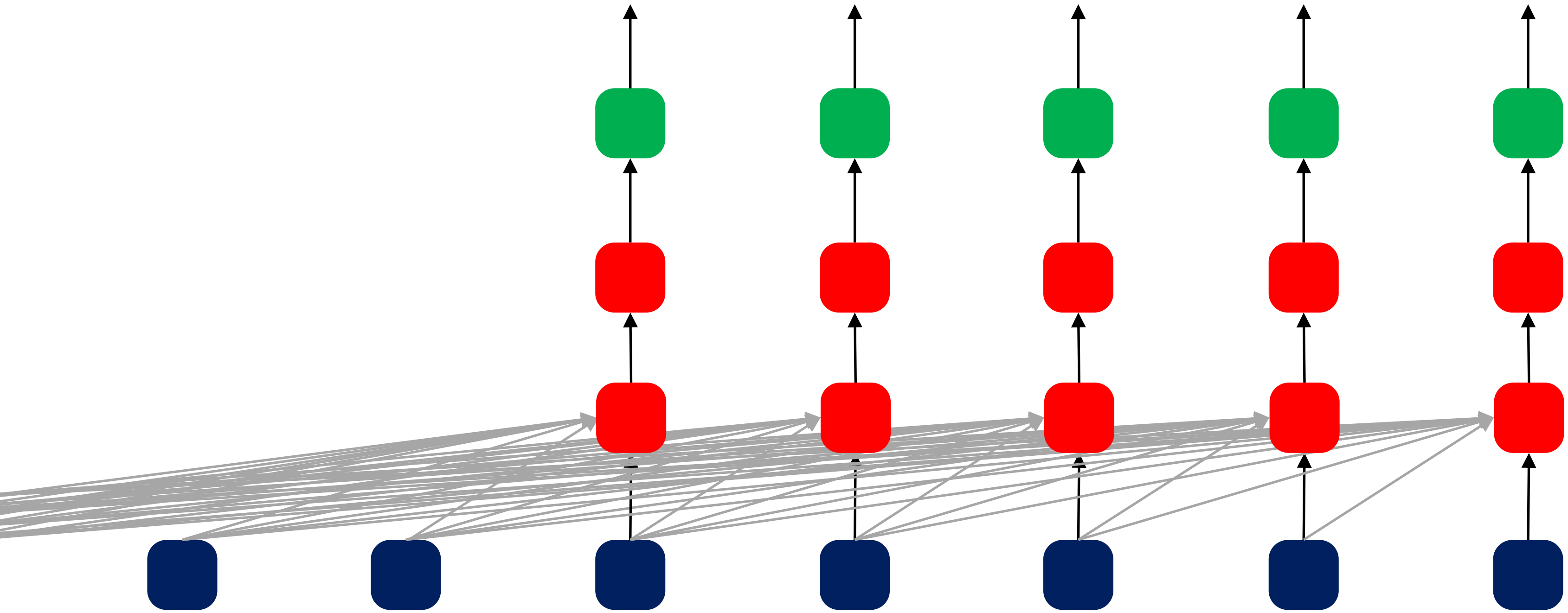


- Output is affected by a finite length past input history. If K is the window of the system, then

$$\mathbf{y}_t^* = f\left(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-K}\right)$$

- Longer trends can be accounted for by increasing the history.
  - But the network becomes more complex.

Infinite response system

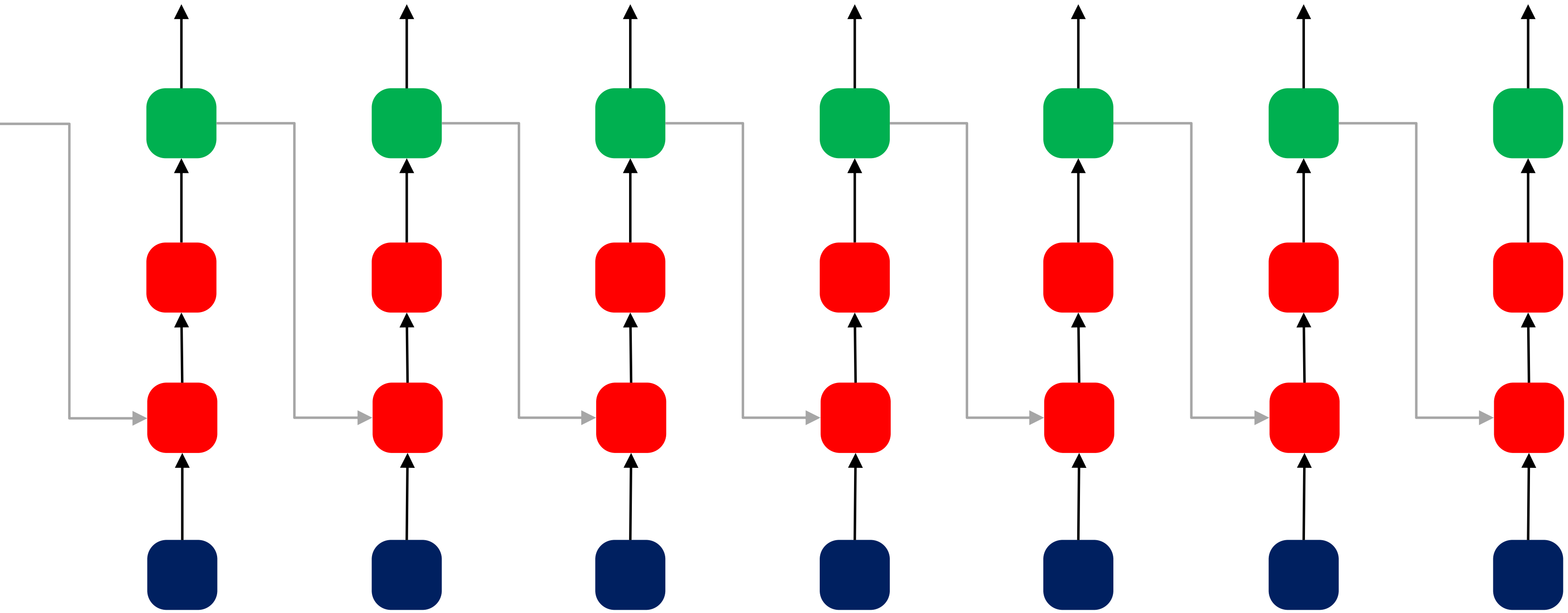


- Output is affected by the entire input history.

$$\mathbf{y}_t^* = f\left(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-\infty}\right)$$

- Number of parameters will blow-up soon.
- Let's change the structure of the network.

Use output instead of input

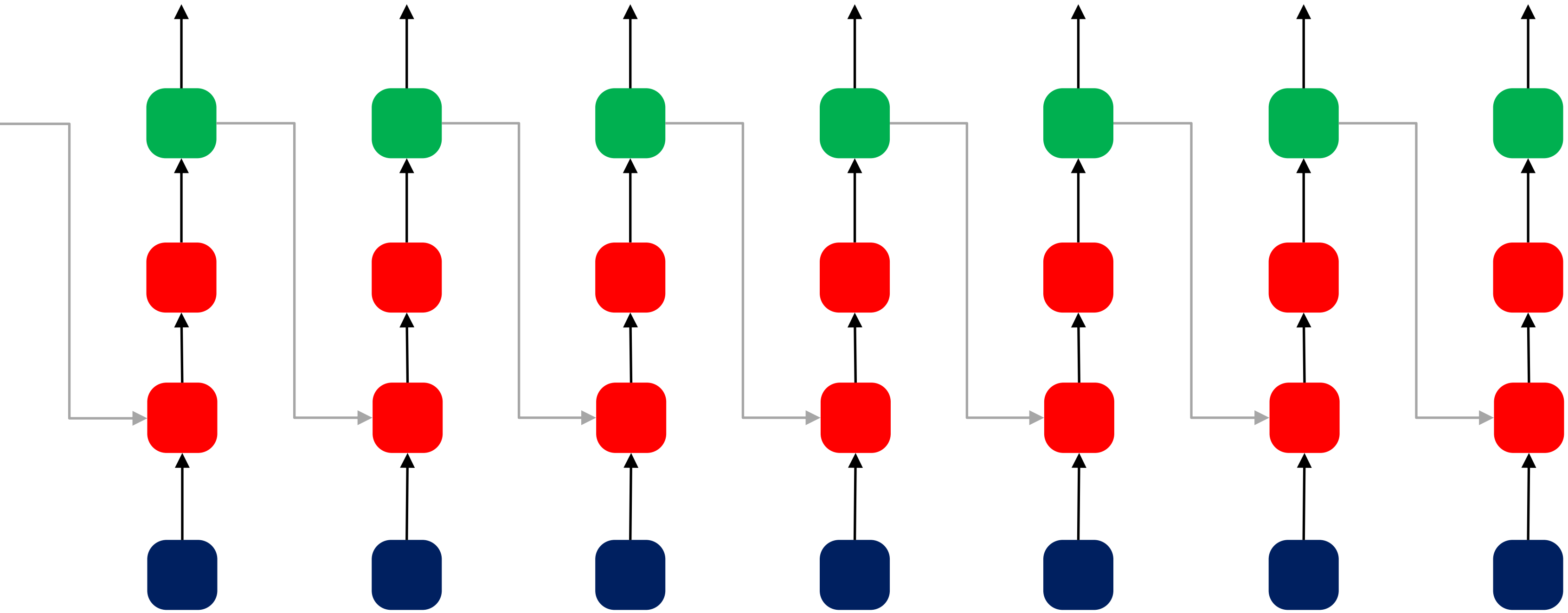


- Instead of using only the input, have the system respond to the previous output.
- The output is then given as

$$\mathbf{y}_t^* = f\left(\mathbf{x}_t, \mathbf{y}_{t-1}^*\right)$$

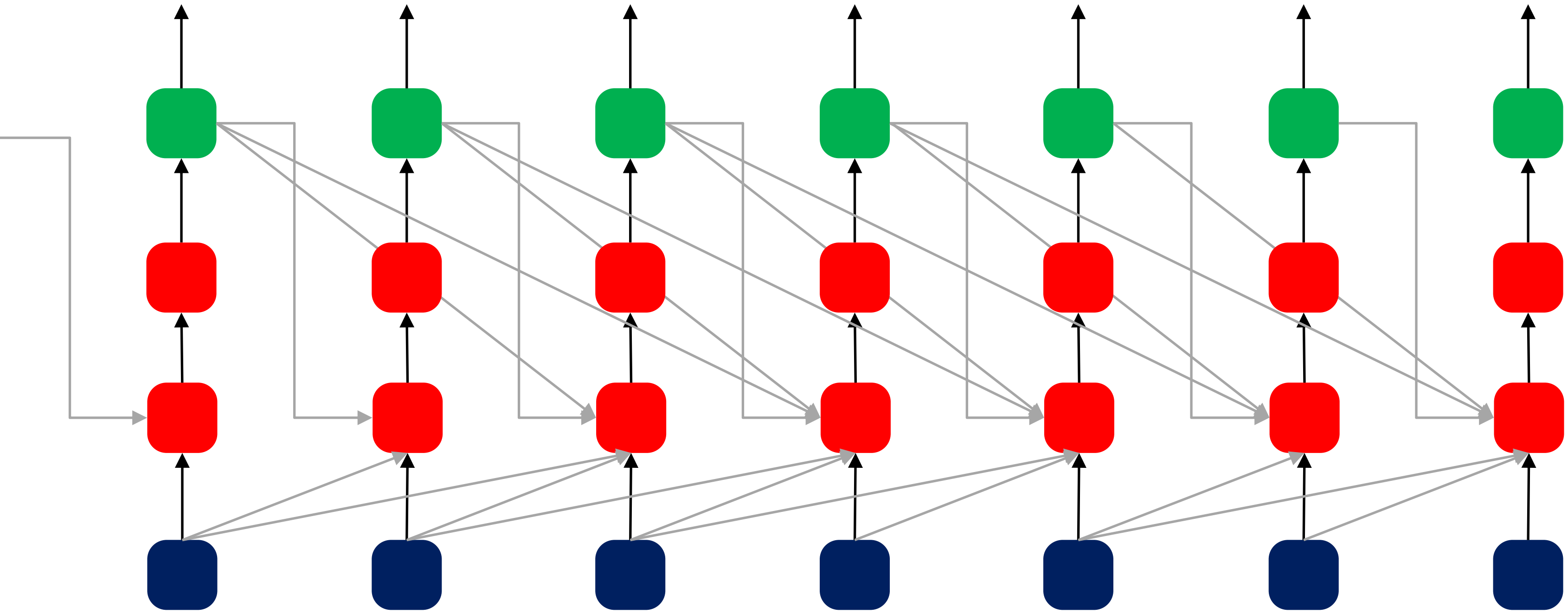
- The above equation is recurrent: The value of a variable (output) at a particular step t is obtained by referring back to same definition of the variable at the previous step.

NARX network



- All columns are identical – the weights in all the columns are the same.
- Define $\mathbf{y}^*_{-1}$ for $t = 0$.
- This is a particular case of a NARX network.
- NARX: Nonlinear autoregressive network with exogenous inputs
- Input at any time affects the outputs forever.

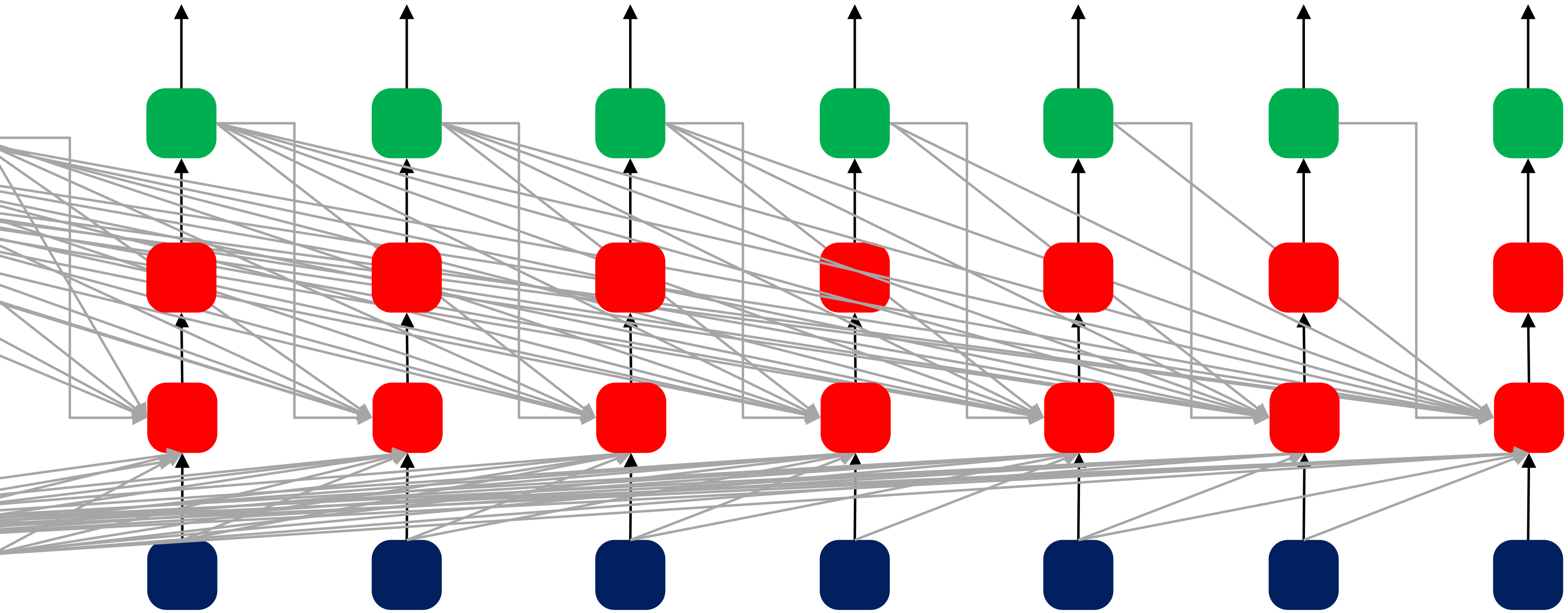
Generic NARX network



- The output $\mathbf{y}_t^*$ is a function of previous inputs over some window L along with the current input, and previous outputs over some window M:

$$\mathbf{y}_t^* = f\left(\mathbf{x}_t, \mathbf{x}_{t-1}, ..., \mathbf{x}_{t-L}, \mathbf{y}_{t-L}^*, \mathbf{y}_{t-2}^*, ..., \mathbf{y}_{t-M}^*\right)$$

Complete NARX network



- Can think of extending the network to incorporate all past inputs and outputs.
- Model not practical.

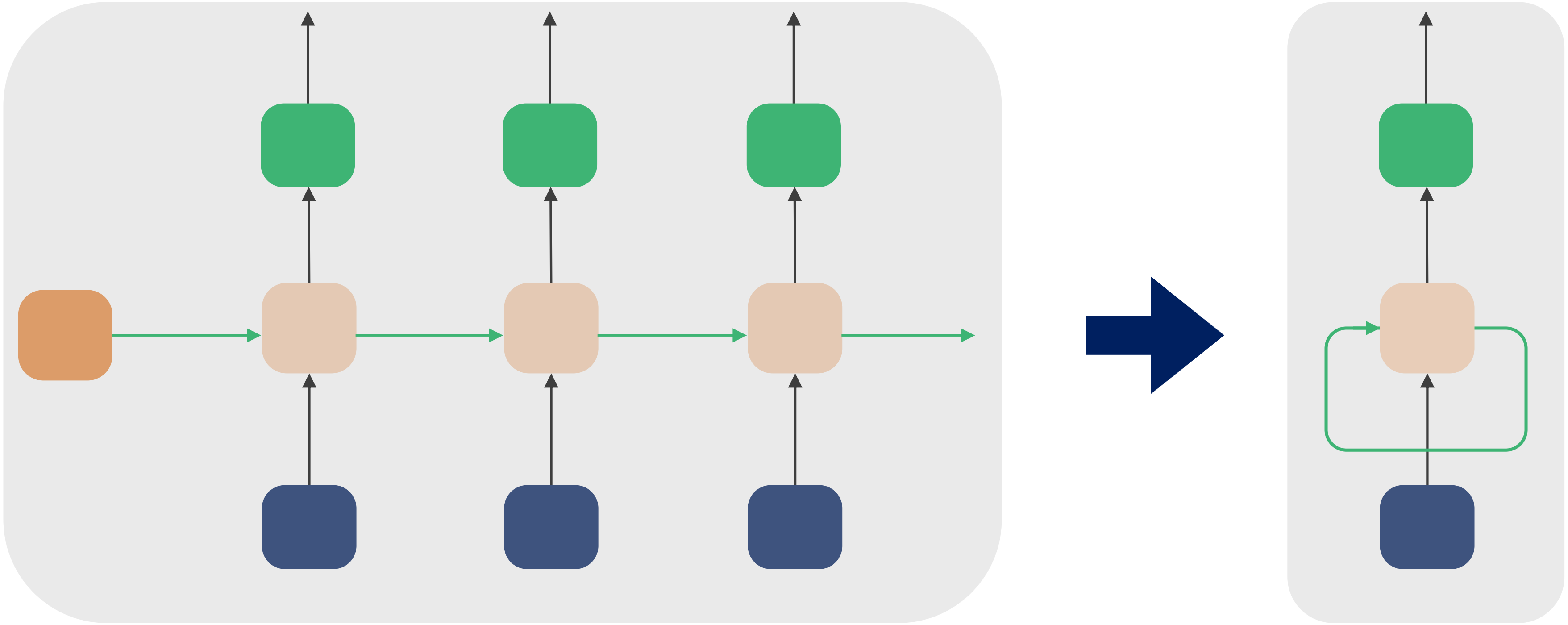
State-space model

- Let $\mathbf{h}_t$ be the state of a network.
- The state of a dynamical system with "external" input can be expressed as

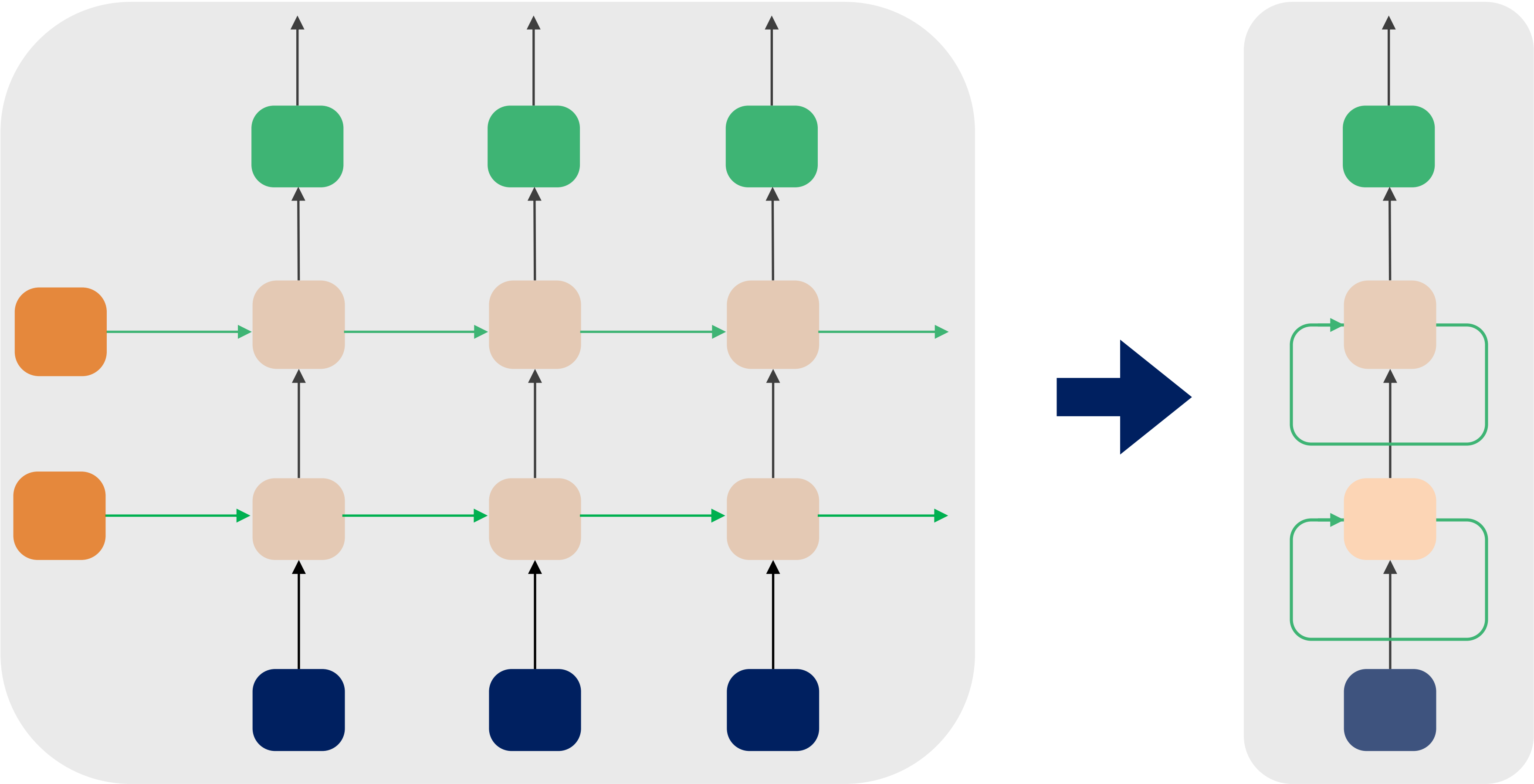
$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$

- The state summarizes the information about the entire past sequence.
- This is a recurrent neural network (RNN).
- RNNs use the same functions and parameters for all time-steps of a sequence.
- Use of shared functions and parameters is good for generalization.

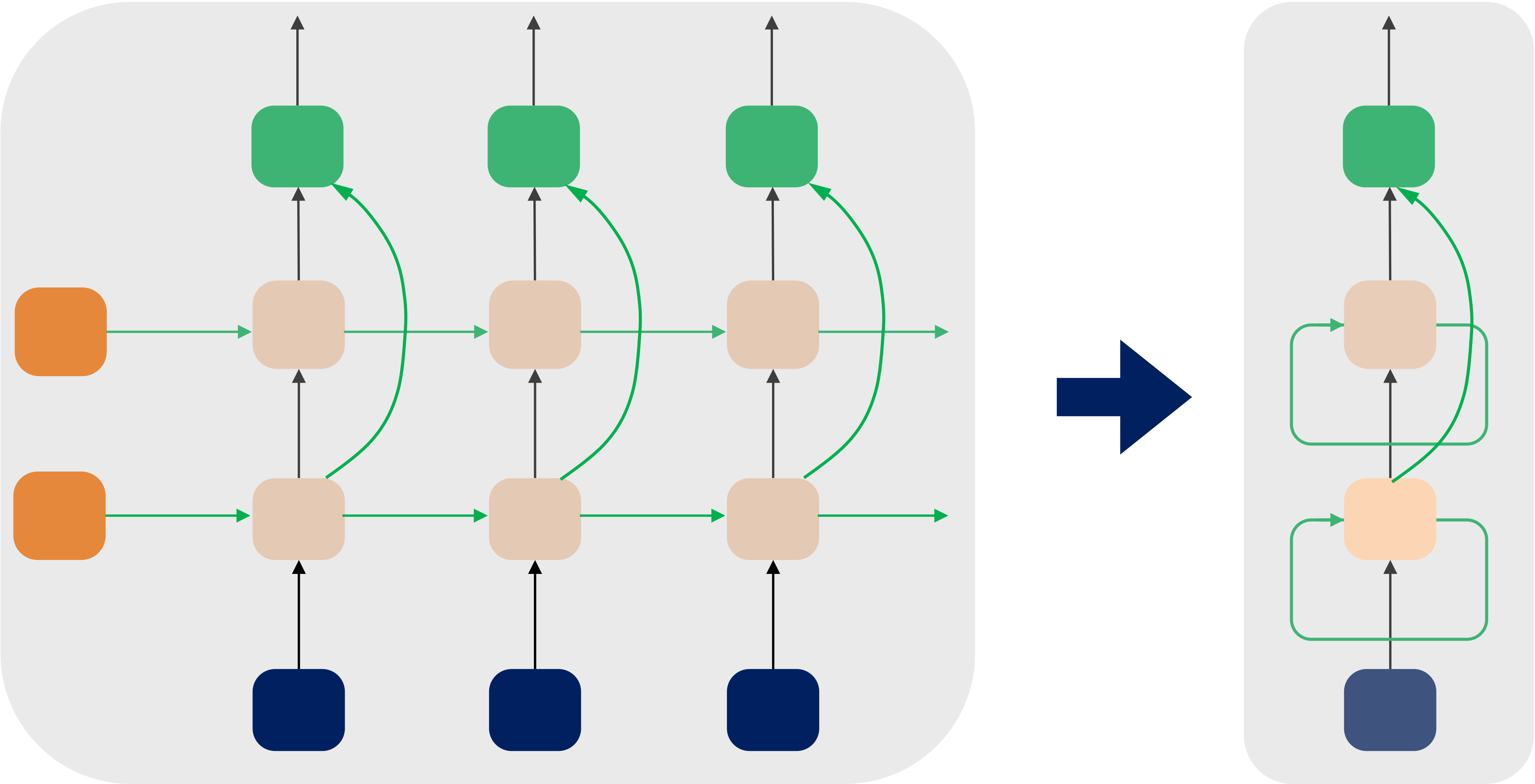
Simple RNN



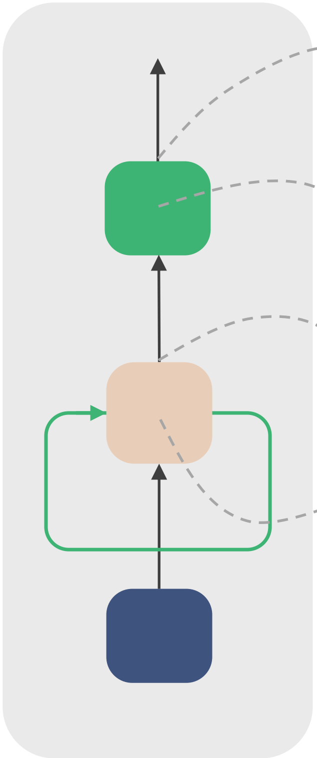
RNN with multiple recurrent layers



RNN with multiple recurrent layers



Mathematics



$$\mathbf{y}^{*(n)} = \mathcal{A}_2\left(\mathbf{z}_{2,t}^{(n)}\right)$$

$$\mathbf{z}_{2,t}^{(n)} = \mathbf{w}_2\mathbf{h}_t^{(n)} + \mathbf{w}_{20}$$

$$\mathbf{h}_t^{(n)} = \mathcal{A}_1\left(\mathbf{z}_{1,t}^{(n)}\right)$$

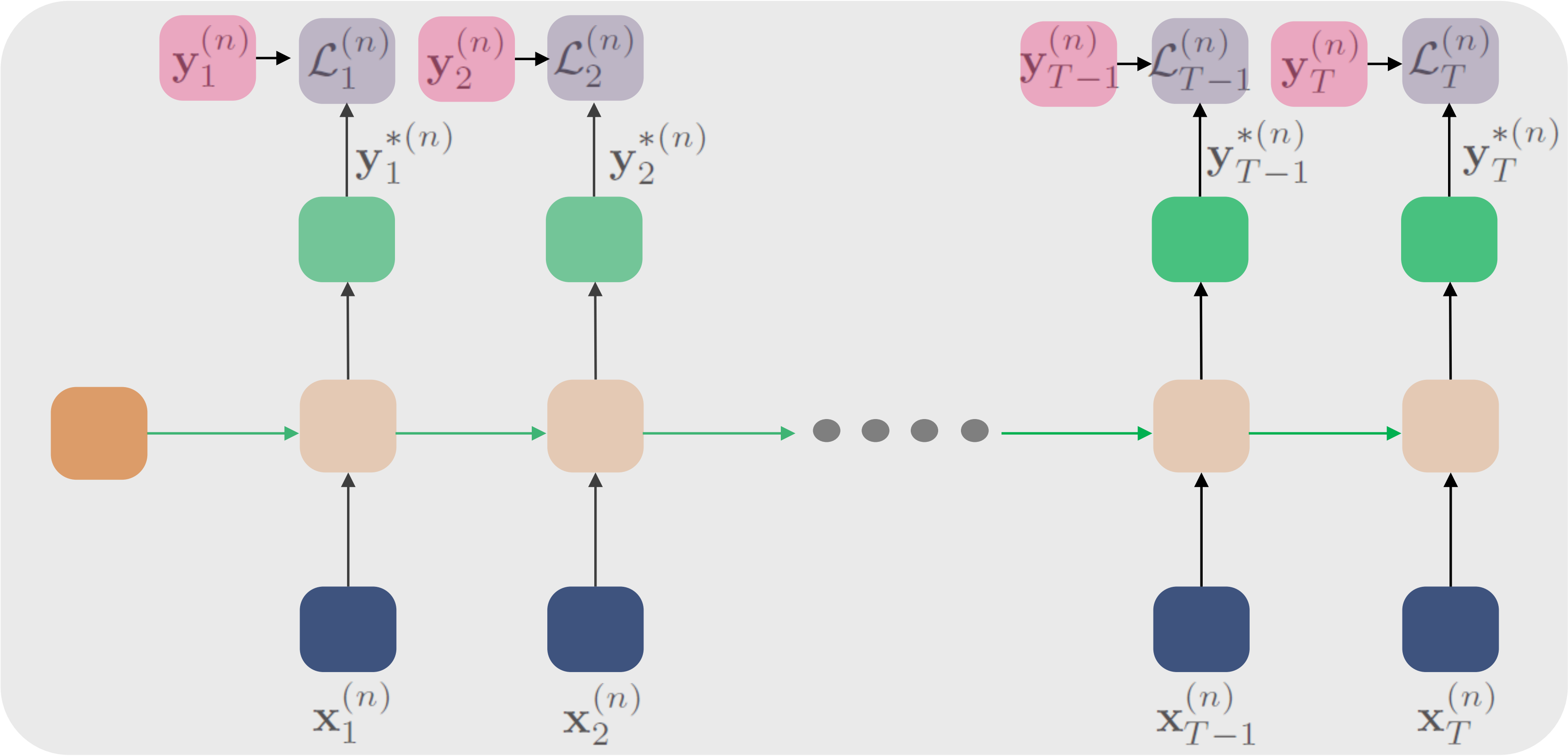
$$\mathbf{z}_{1,t}^{(n)} = \mathbf{w}_1\mathbf{x}_t^{(n)} + \mathbf{w}_{11}\mathbf{h}_{t-1}^{(n)} + \mathbf{w}_{10}$$

Training

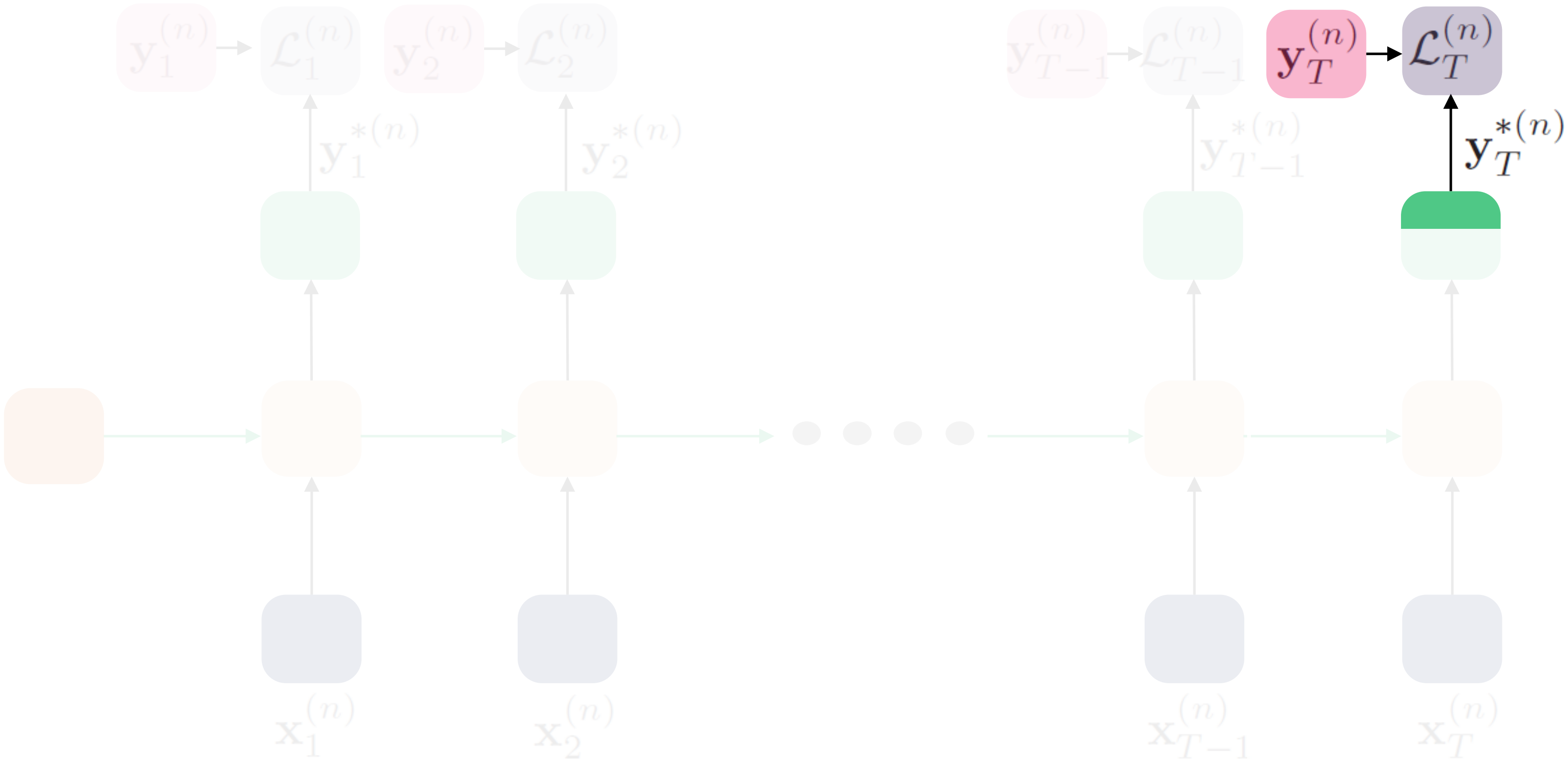
- Use a form of back-propagation algorithm called the back-propagation through time (BPTT) algorithm.
- Facilitates application of gradient based techniques. Backpropagation involves computation of gradients.
- For the nth sequence, we have
  - Inputs: $\mathbf{x}_1^{(n)}, \mathbf{x}_2^{(n)}, \ldots.., \mathbf{x}_{T-1}^{(n)}, \mathbf{x}_T^{(n)}$
  - Outputs from network: $\mathbf{y}_1^{*(n)}, \mathbf{y}_2^{*(n)}, \ldots.., \mathbf{y}_{T-1}^{*(n)}, \mathbf{y}_T^{*(n)}$
  - Given outputs: $\mathbf{y}_1^{(n)}, \mathbf{y}_2^{(n)}, \ldots.., \mathbf{y}_{T-1}^{(n)}, \mathbf{y}_T^{(n)}$
  - Loss at different time-steps: $\mathcal{L}_1^{(n)}, \mathcal{L}_2^{(n)}, \ldots.., \mathcal{L}_{T-1}^{(n)}, \mathcal{L}_T^{(n)}$
  - Loss corresponding to the nth sequence:

$$\mathcal{L}^{(n)} = \sum_{t=1}^{T} \mathcal{L}_t^{(n)}$$

Backpropagation through time (BPTT)

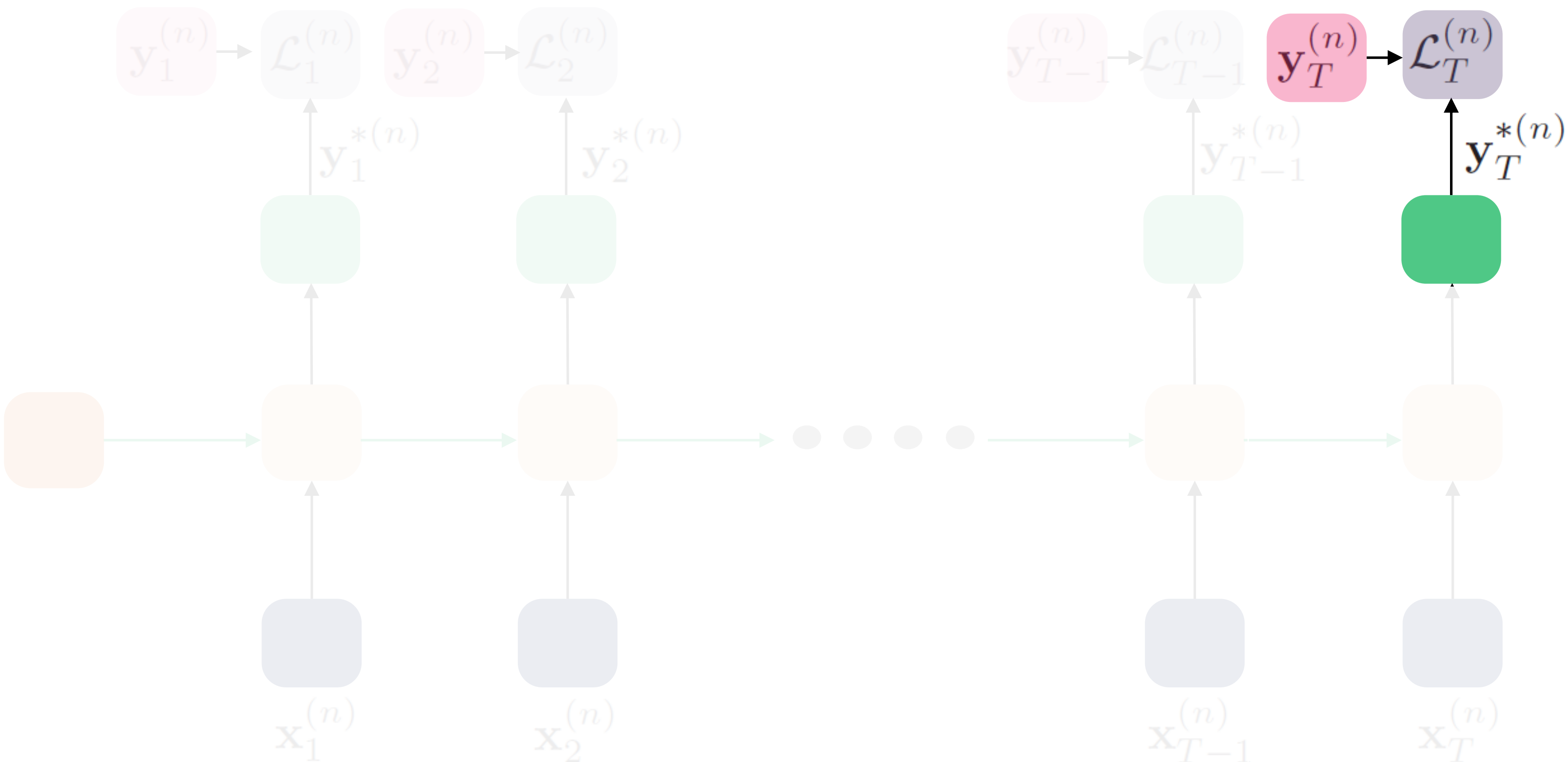


Backpropagation through time (BPTT)



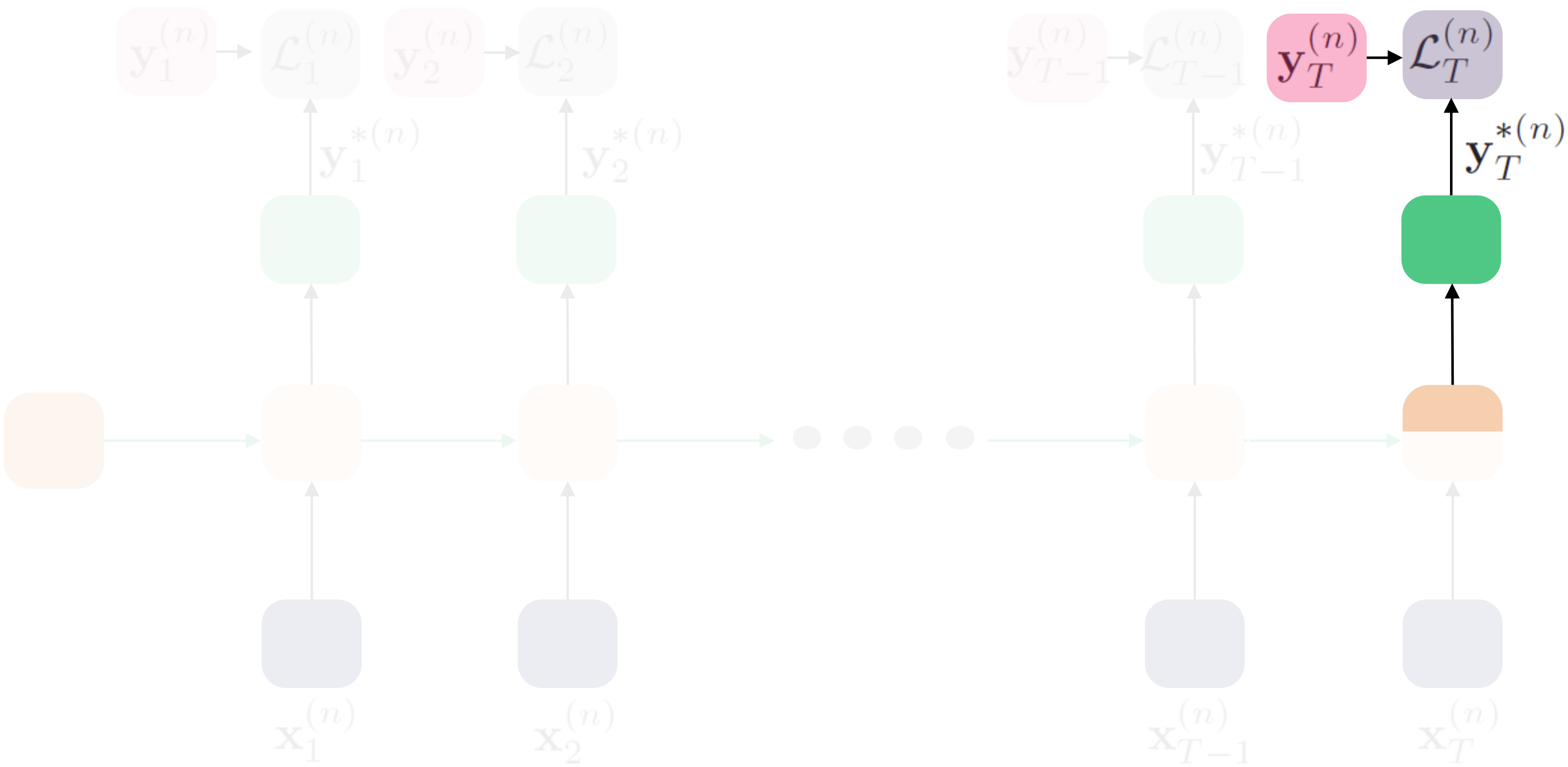
$$\frac{\partial \mathcal{L}^{(n)}}{\partial y_{T,i}^{*(n)}}$$

Backpropagation through time (BPTT)



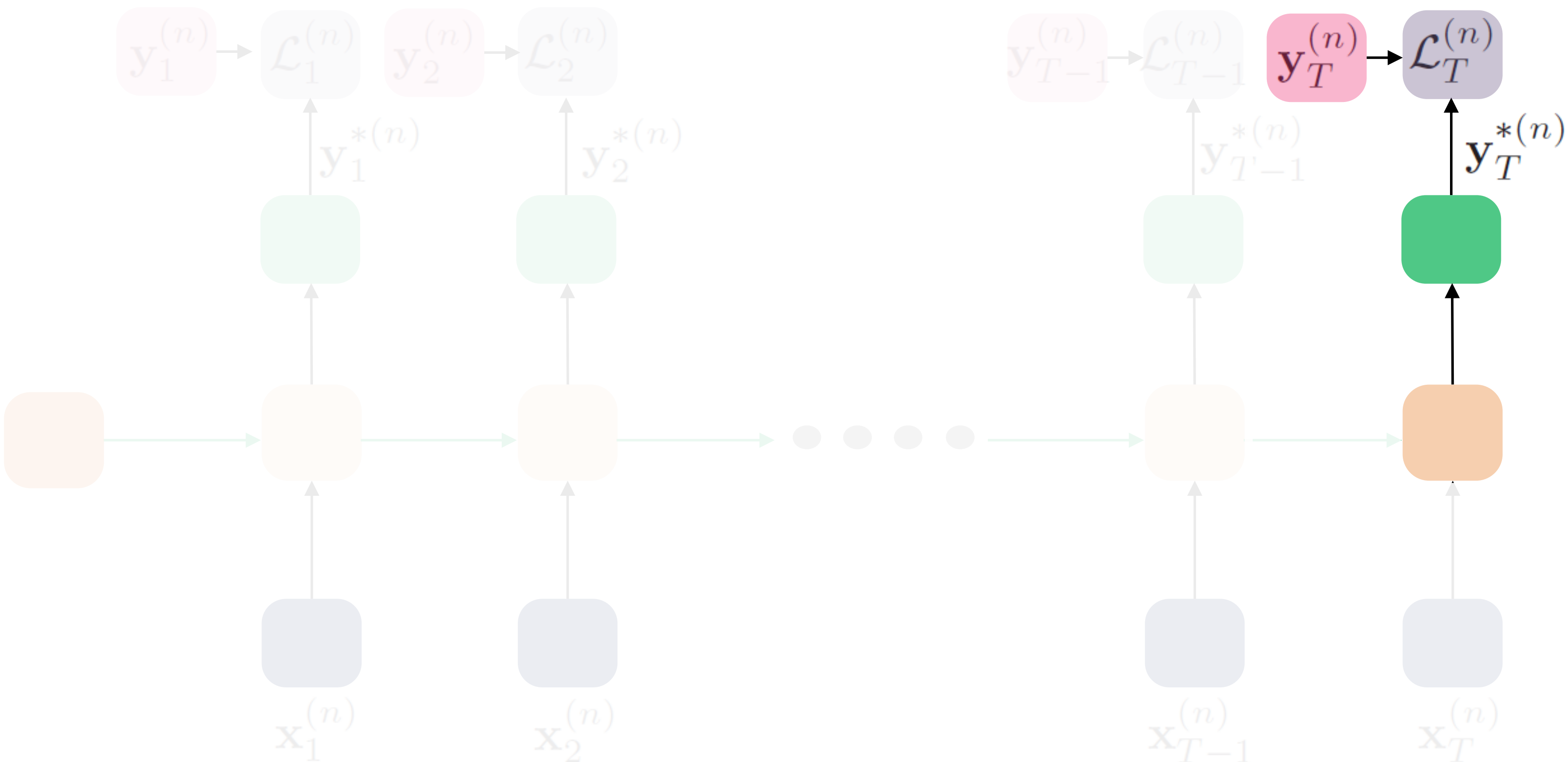
$$\frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T,i}^{(n)}} = \frac{\partial \mathcal{L}^{(n)}}{\partial y_{T,i}^{*(n)}} \frac{\partial y_{T,i}^{*(n)}}{\partial z_{2,T,i}^{(n)}}$$

Backpropagation through time (BPTT)



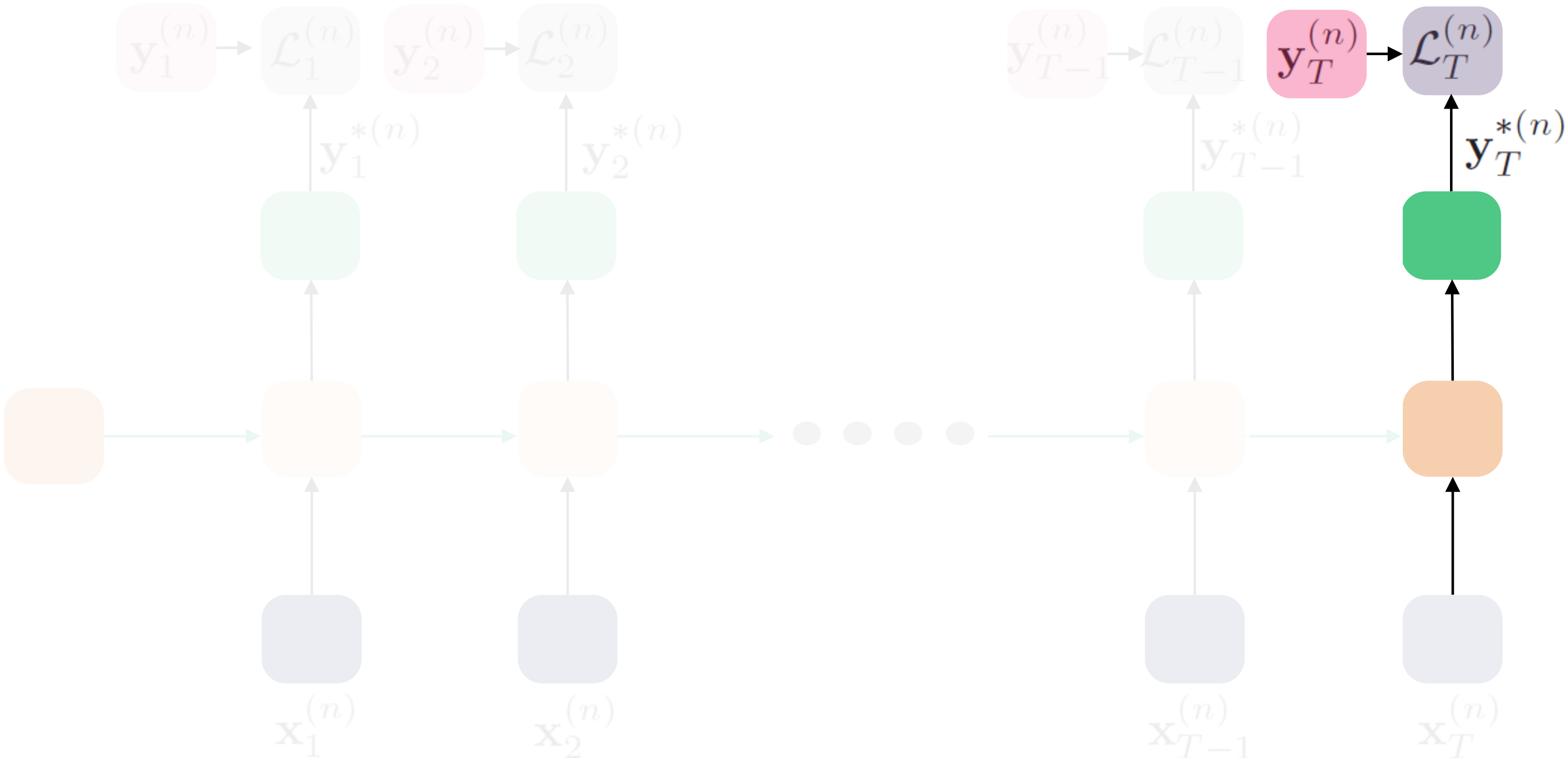
$$\frac{\partial \mathcal{L}^{(n)}}{\partial h_{T,i}^{(n)}} = \sum_j \frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T,j}^{(n)}} \frac{\partial z_{2,T,j}^{(n)}}{\partial h_{T,i}^{(n)}}$$

Backpropagation through time (BPTT)



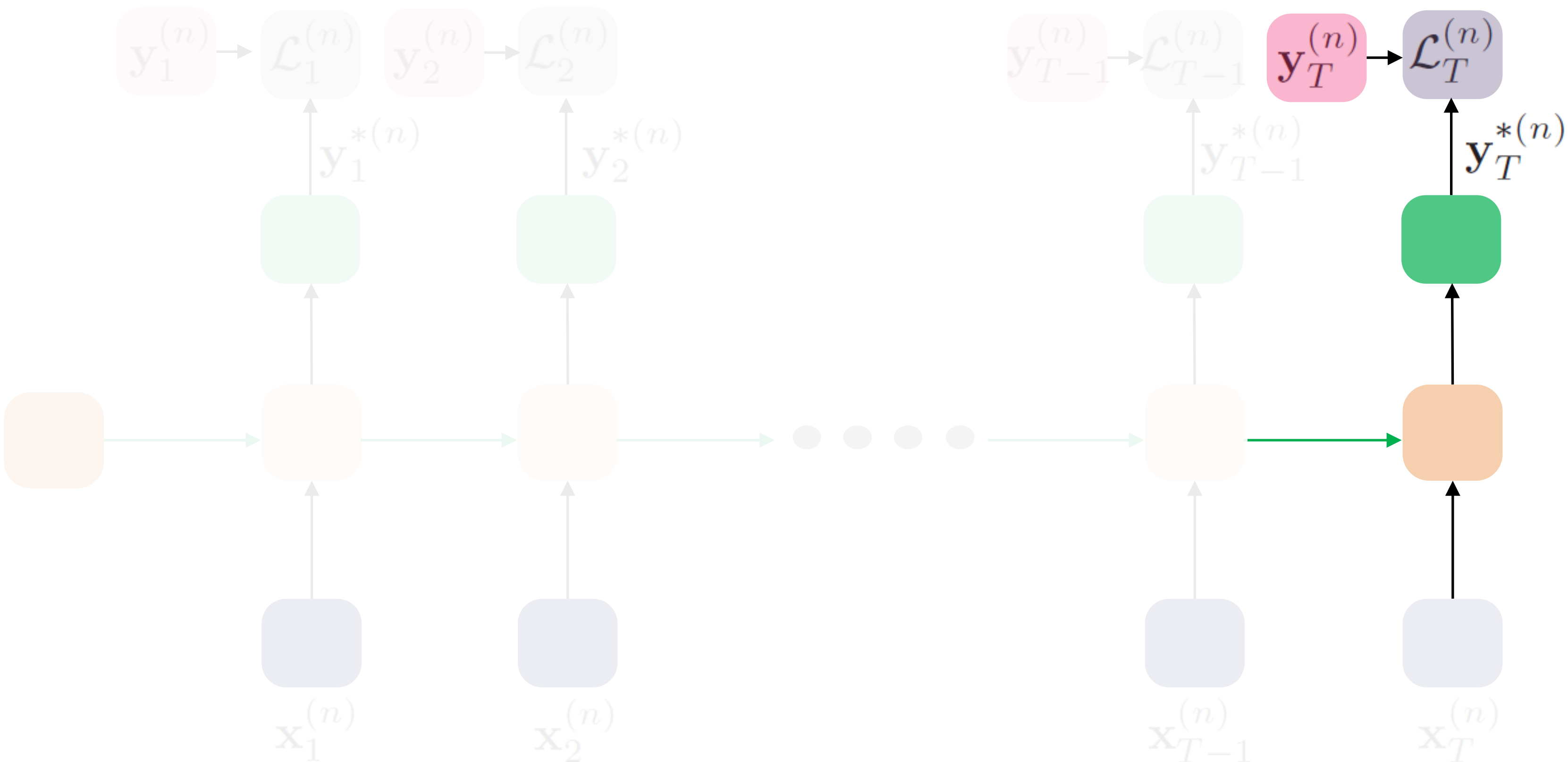
$$\frac{\partial \mathcal{L}^{(n)}}{\partial z_{1,T,i}^{(n)}} = \frac{\partial \mathcal{L}^{(n)}}{\partial h_{T,i}^{(n)}} \frac{\partial h_{T,i}^{(n)}}{\partial z_{1,T,i}^{(n)}}$$

Backpropagation through time (BPTT)



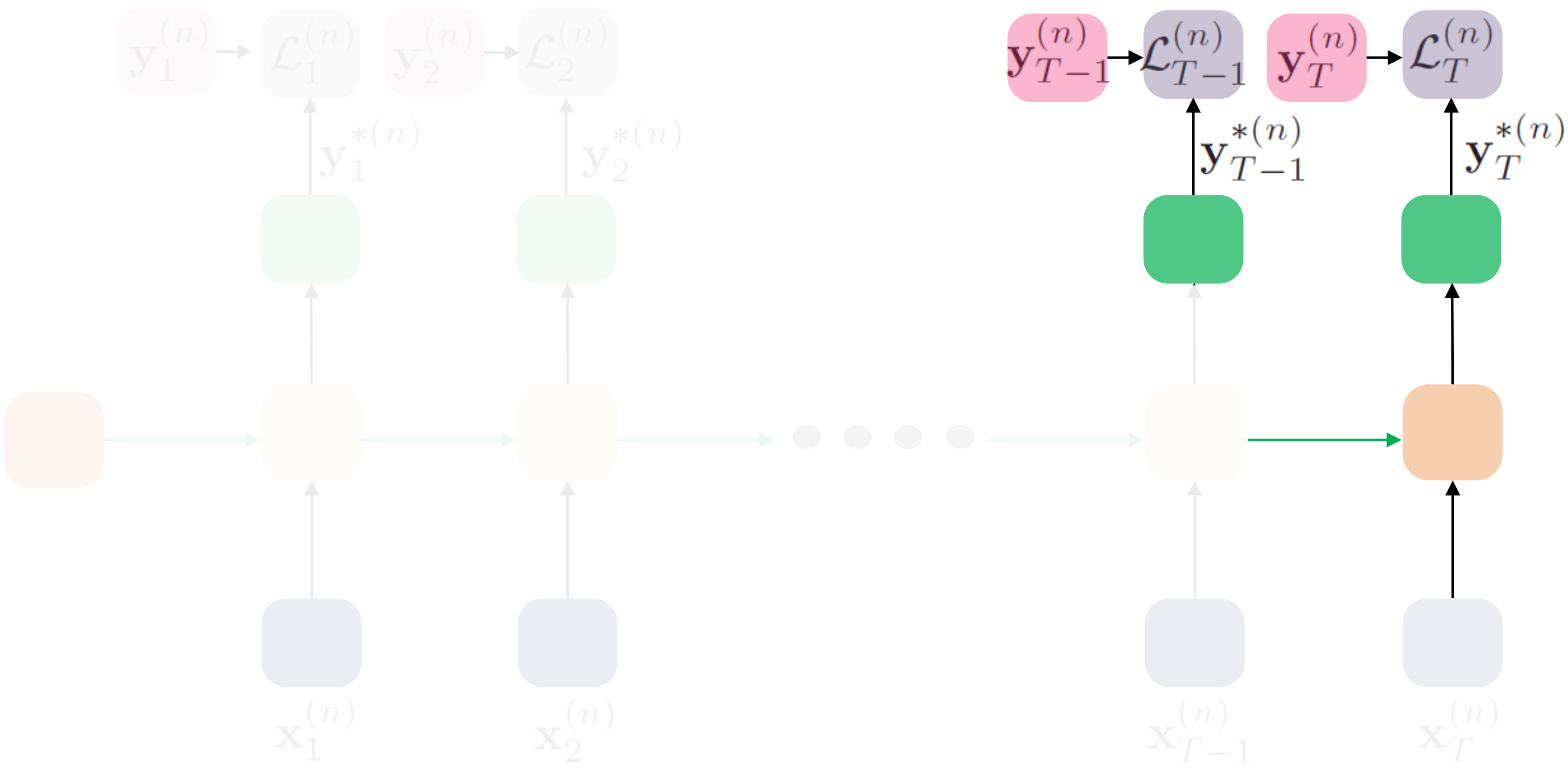
$$\frac{\partial \mathcal{L}^{(n)}}{\partial w_{1,ij}} = \frac{\partial \mathcal{L}^{(n)}}{\partial z_{1,T,j}^{(n)}} x_{T,i}^{(n)}$$

Backpropagation through time (BPTT)



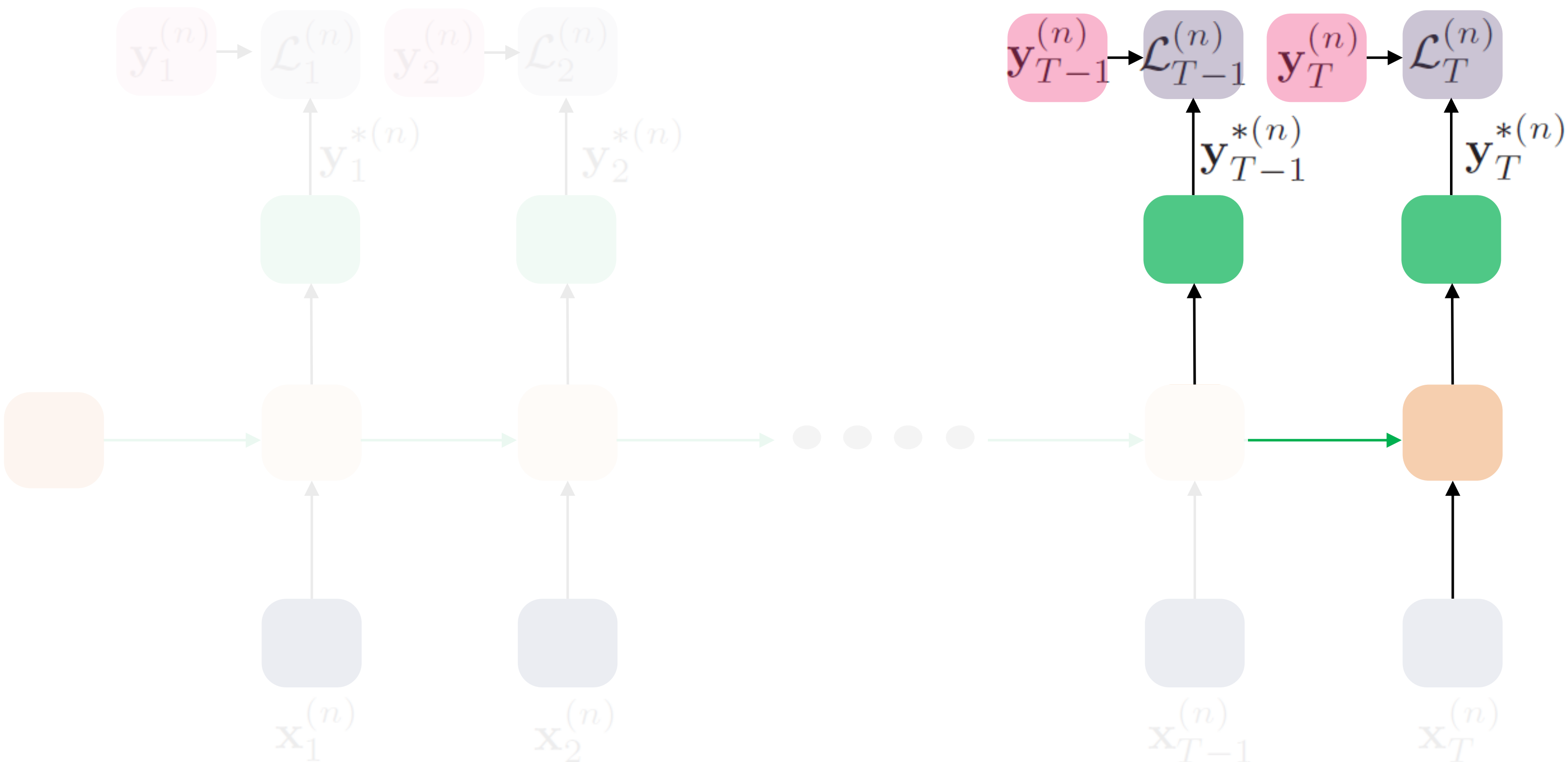
$$\frac{\partial \mathcal{L}^{(n)}}{\partial w_{11,ij}} = \frac{\partial \mathcal{L}^{(n)}}{\partial z_{1,T,j}^{(n)}} h_{T-1,i}^{(n)}$$

Backpropagation through time (BPTT)



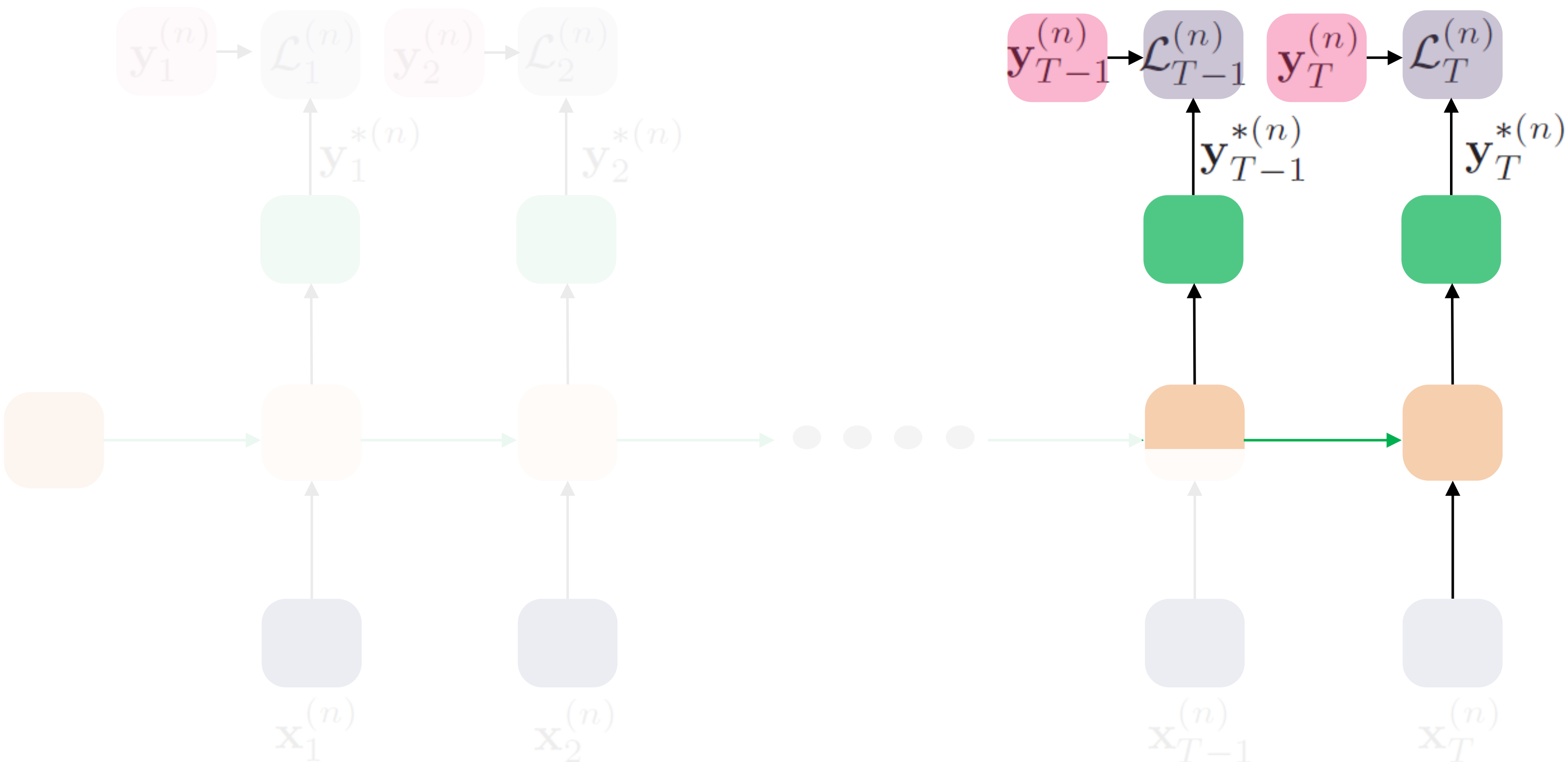
$$\frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T-1,i}^{(n)}} = \frac{\partial \mathcal{L}^{(n)}}{\partial y_{T-1,i}^{*(n)}} \frac{\partial y_{T-1,i}^{*(n)}}{\partial z_{2,T-1,i}^{(n)}}$$

Backpropagation through time (BPTT)



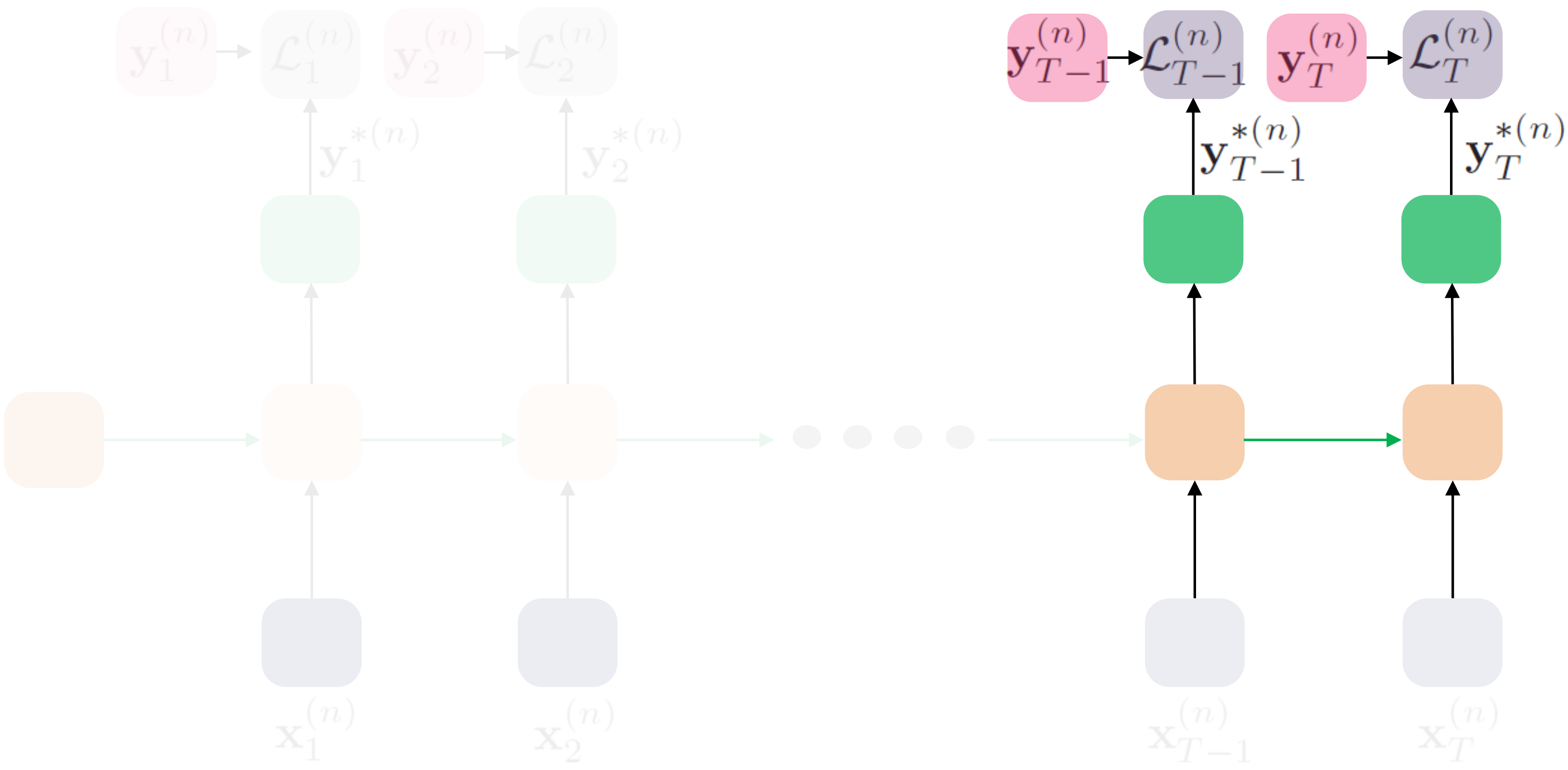
$$\frac{\partial \mathcal{L}^{(n)}}{\partial w_{2,ij}} += \frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T-1,j}^{(n)}} h_{T-1,i}^{(n)}$$

Backpropagation through time (BPTT)



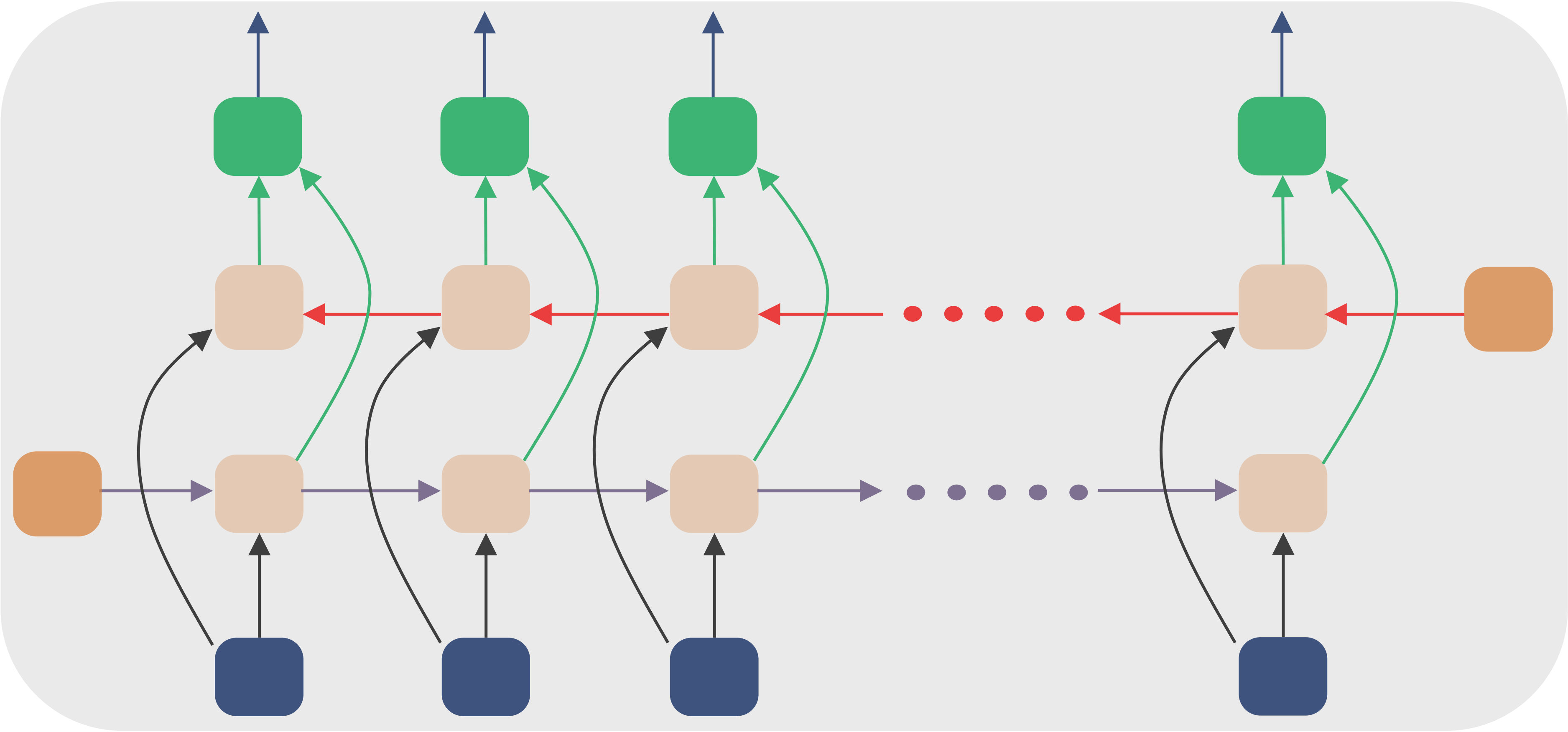
$$\frac{\partial \mathcal{L}^{(n)}}{\partial h_{T-1,i}^{(n)}} = \sum_j w_{2,ij} \frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T-1,j}^{(n)}} + \sum_j w_{11,ij} \frac{\partial \mathcal{L}^{(n)}}{\partial z_{2,T,j}^{(n)}}$$

Backpropagation through time (BPTT)



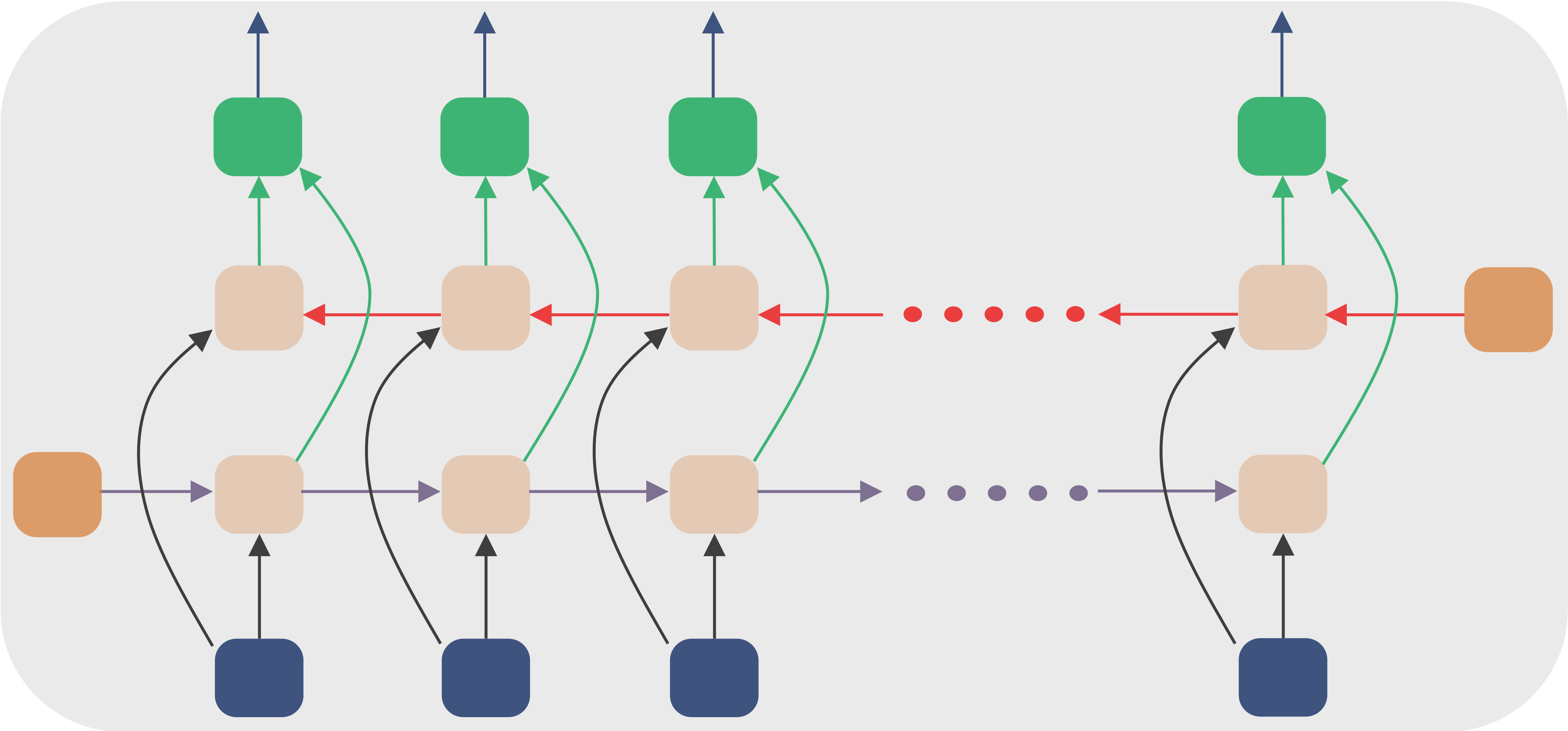
$$\frac{\partial \mathcal{L}^{(n)}}{\partial w_{1,ij}} += \frac{\partial \mathcal{L}^{(n)}}{\partial z_{1,T-1,j}^{(n)}} x_{T-1,i}^{(n)}$$

Bidirectional RNN (BRNN)



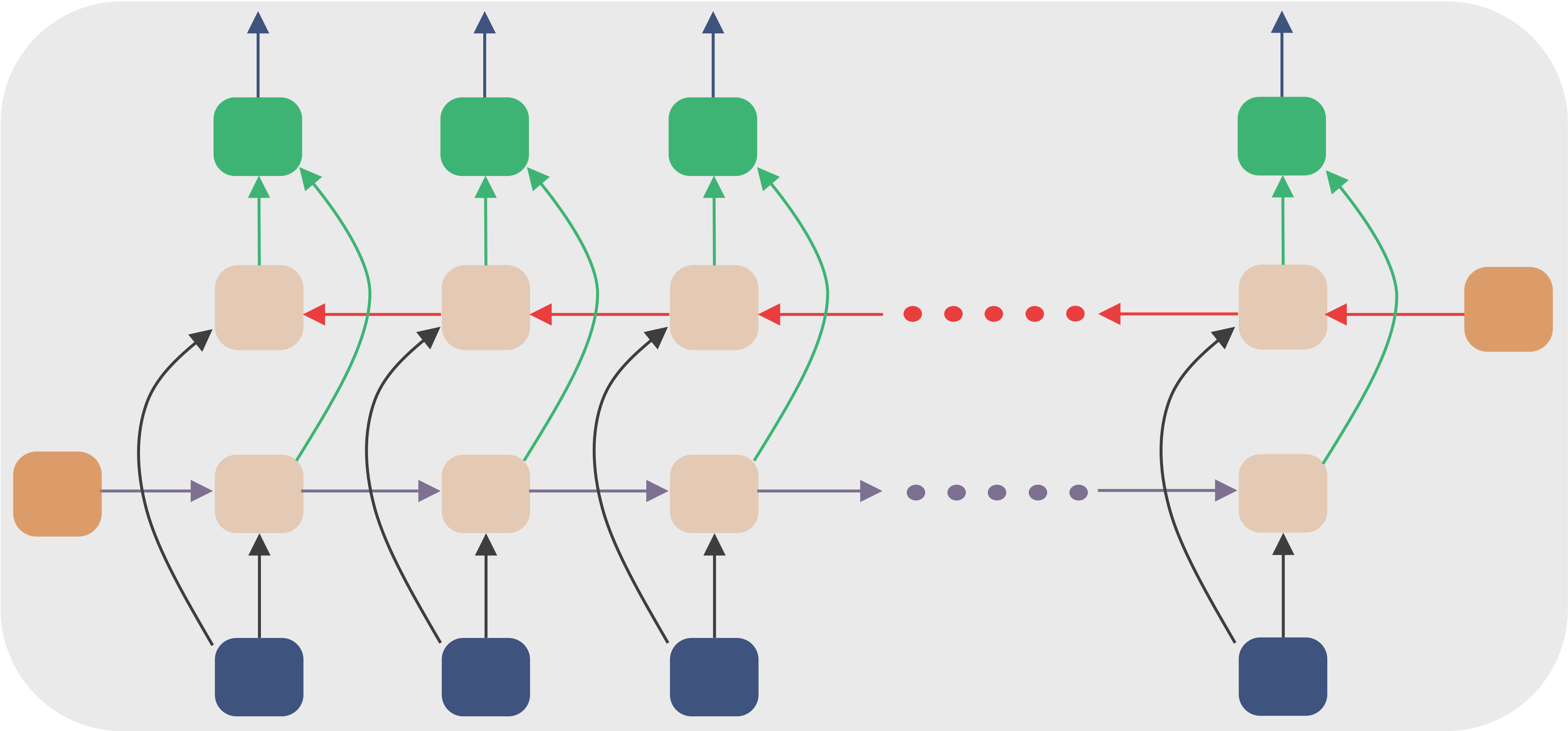
- The t-th output depends on both the past and the future elements of the sequence.
- In standard RNNs, knowledge of future inputs cannot be obtained from the present state.

Bidirectional RNN (BRNN)



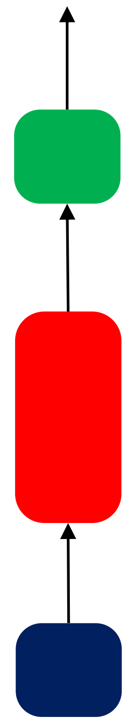
- Comprise two RNNs:
  - Forward RNN processes data from $t = 0$ to $t = T$.
  - Backward RNN processes data from $t = T$ to $t = 0$.
- The output is computed using the hidden state of forward and backward RNNs.

Bidirectional RNN (BRNN)



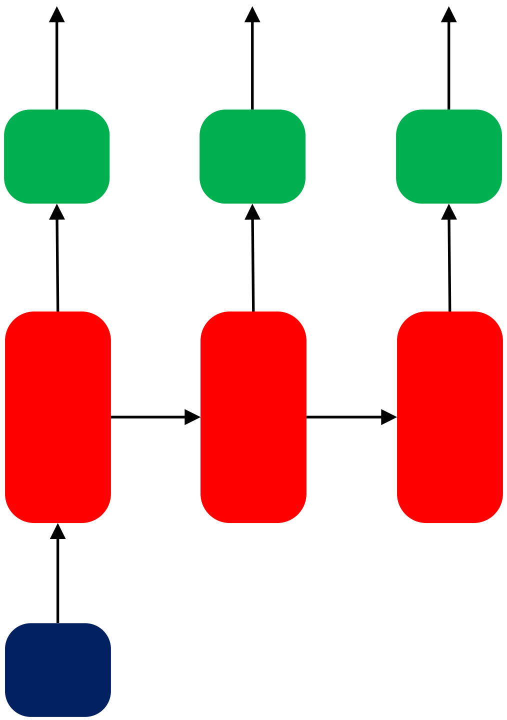
- Examples:
  - In handwriting recognition, better performance can be achieved using knowledge of letters on the two sides of the present letter.
  - Predict the missing word in a sequence.

RNN paradigms – one to one



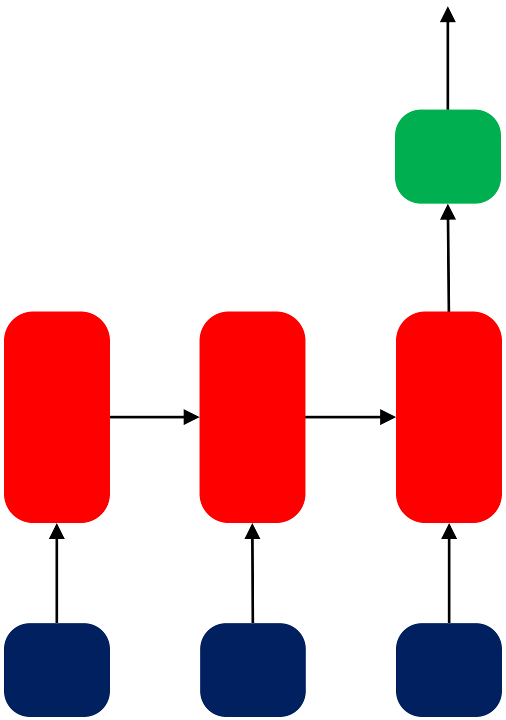
- Single input, singe output system
- A feed-forward neural network

RNN paradigms – one to many



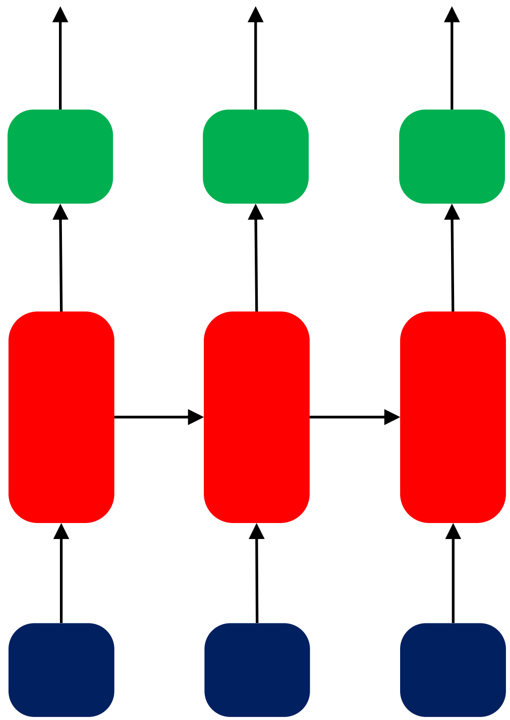
- Takes a single input to predict a sequence of outputs
- Application – Image captioning

RNN paradigms – many to one



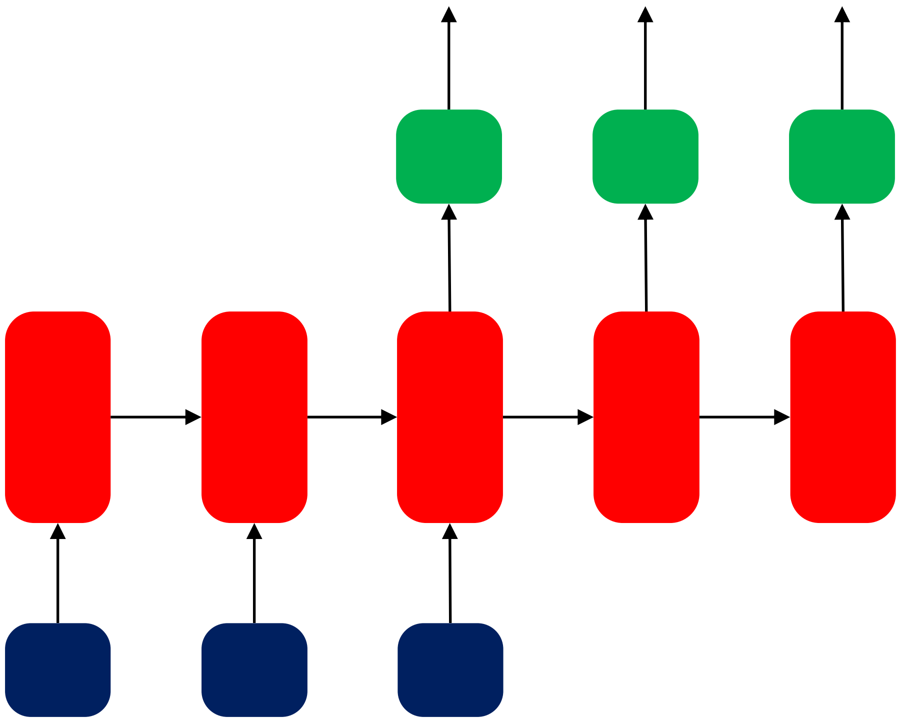
- Takes a sequence of inputs to predict an output
- Application – Sentiment Analysis

RNN paradigms – many to many



- Takes a sequence of inputs to predict a sequence of outputs
- Application – Part of speech tagging

RNN paradigms – many to many



- Takes a sequence of inputs to predict a sequence of outputs
- Application – Language translation

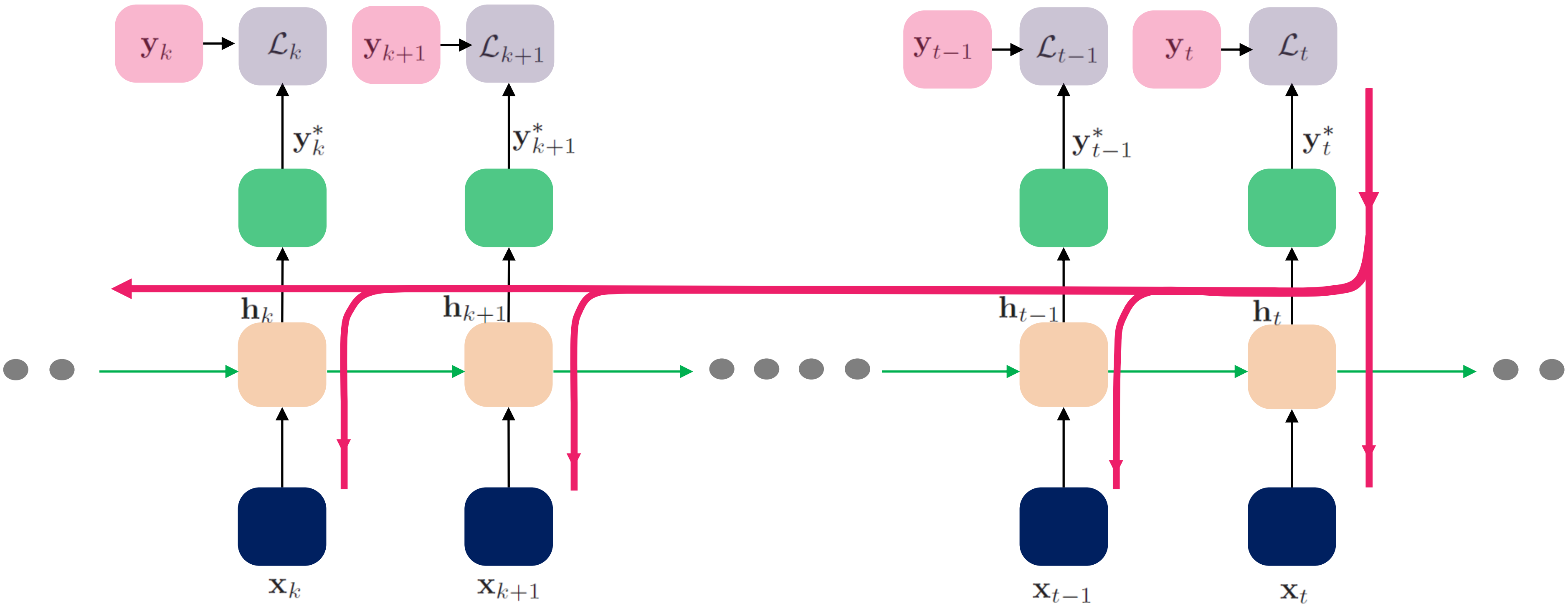
Shortcomings of RNN

- The "memory" of RNNs used in practice are poor.
- The memory is affected by the activation function of the hidden layers:
  - Sigmoid activation saturate fast, so "memory" is lost very quickly.
  - RELU activation functions are prone to blowing up.
  - tanh(.) activation function is more effective than others, but cannot retain information for long.
- The outputs of the hidden layers are also affected by the recurrent weights matrix:
  - The network response may blow up if the largest eigenvalue of the matrix is > 1.
  - The response can attenuate if the largest eigenvalue is < 1.

Shortcomings of RNN

- Problem of vanishing/exploding gradients – a typical problem of deep networks where the gradients in the initial layers can "vanish" or "explode".
- Derivative of the loss function at a particular layer depends on the weight matrices and Jacobians of activation functions in earlier layers.
- The derivatives of the activation functions such as $\sigma(.)$, $\tanh(.)$, RELU(.) are always ≤ 1.
  - During backpropagation, as one moves towards the beginning of the network, the Jacobians of the activation functions shrink the derivative of the loss function.
  - The derivative of the loss function becomes negligible after a few layers.

Vanishing/exploding gradients in RNN



- Suppose want to compute the the partial derivative of the loss function value at time-step t w.r.t. weights connecting the inputs with the hidden units

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{w}_1} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{y}^*_t} \frac{\partial \mathbf{y}^*_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{w}_1}$$

where $$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \cdots \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \prod_{m=k+1}^{t} \frac{\partial \mathbf{h}_m}{\partial \mathbf{h}_{m-1}}$$

Vanishing/exploding gradients in RNN

$$\prod_{m=k+1}^{t} \left\| \frac{\partial \mathbf{h}_m}{\partial \mathbf{h}_{m-1}} \right\| = \prod_{m=k+1}^{t} \left\| \mathrm{diag}\left[\mathcal{A}'\left(\mathbf{w}_{11}^{\mathrm{T}} \mathbf{h}_{m-1}\right)\right] \mathbf{w}_{11}^{\mathrm{T}} \right\|$$

$$\leq \prod_{m=k+1}^{t} \left\| \mathrm{diag}\left[\mathcal{A}'\left(\mathbf{w}_{11}^{\mathrm{T}} \mathbf{h}_{m-1}\right)\right] \right\| \; \left\| \mathbf{w}_{11}^{\mathrm{T}} \right\|$$

- Let γ be the largest singular value of $\mathrm{diag}\left[\mathcal{A}'\left(\mathbf{w}_{11}^{\mathrm{T}} \mathbf{h}_{m-1}\right)\right]$.
- Let α be the largest singular value of $\mathbf{w}_{11}^{\mathrm{T}}$.
- Let $\beta = \alpha\gamma$

Vanishing/exploding gradients in RNN

- Then we have

$$\prod_{m=k+1}^{t} \left\| \frac{\partial \mathbf{h}_m}{\partial \mathbf{h}_{m-1}} \right\| \leq \prod_{m=k+1}^{t} \left\| \mathrm{diag}\left[\mathcal{A}'\left(\mathbf{w}_{11}^{\mathrm{T}} \mathbf{h}_{m-1}\right)\right] \right\| \; \left\| \mathbf{w}_{11}^{\mathrm{T}} \right\|$$
$$\leq \prod_{m=k+1}^{t} \gamma\alpha$$
$$\leq \beta^{t-k}$$

- If $\beta < 1$, then long-term contributions (i.e. $t-k$ is large) tend to 0 very fast.

- If $\beta > 1$ (i.e. $\alpha > \frac{1}{\gamma}$), then we can have exploding gradients.

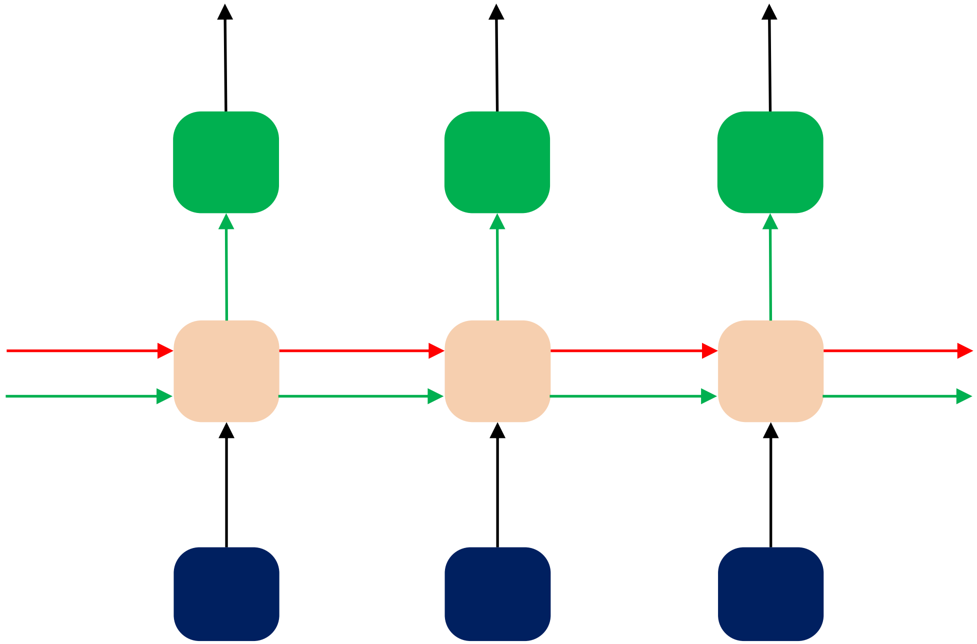
Shortcomings of RNN

- Problem of vanishing/exploding gradients – a typical problem of deep networks where the gradients in the initial layers can "vanish" or "explode".
- Derivative of the loss function at a particular layer depends on the weight matrices and Jacobians of activation functions in earlier layers.
- The weight matrices affect the derivative of the loss function by
  - Increasing it in directions where the singular values of the weight matrices are greater than 1.
  - Shrinking it in directions where the singular values of the weight matrices are less than 1.
  - Thus repeated multiplication of the weight matrices leads to exploding or vanishing gradients.

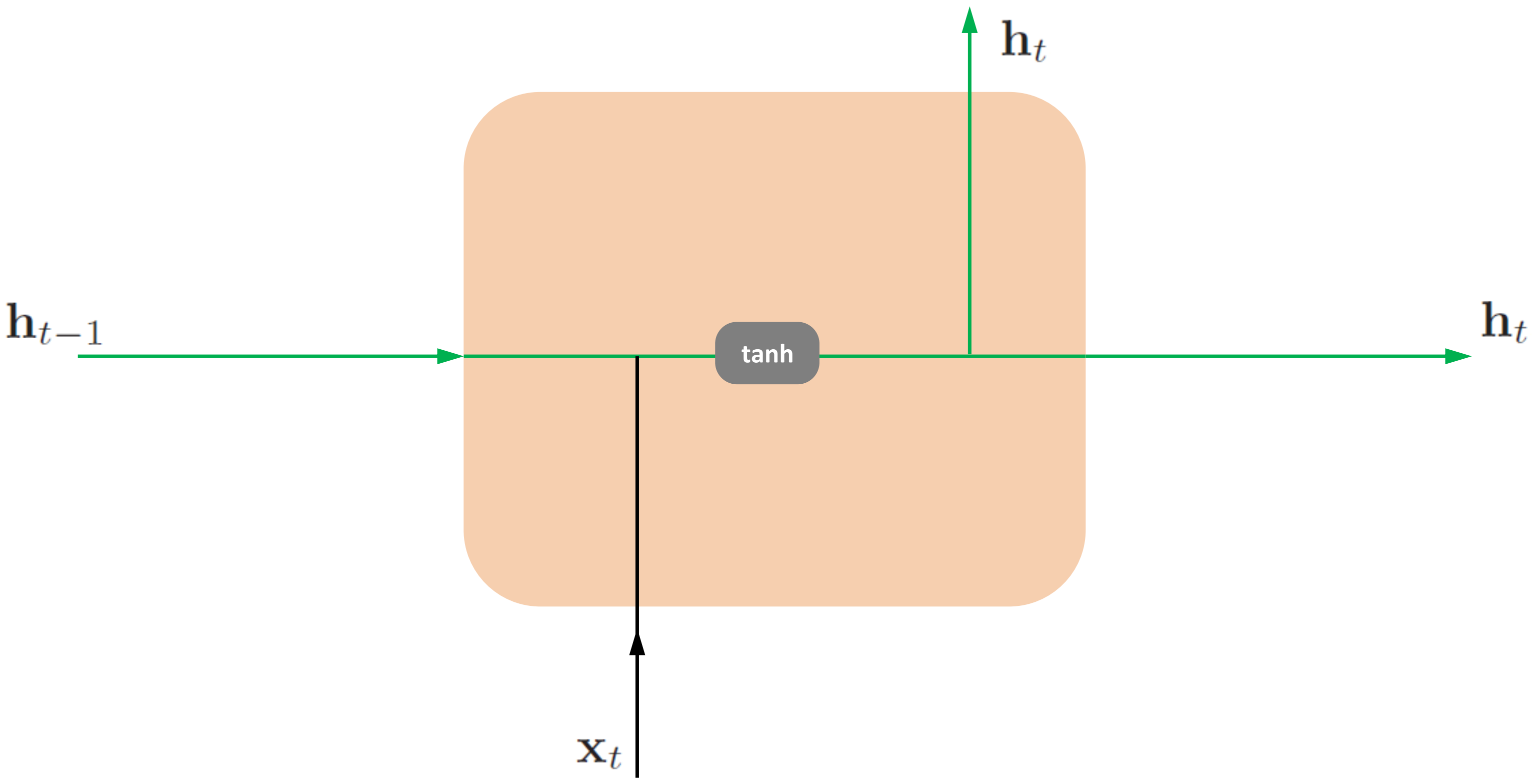
Long short term memory networks

- Long short term memory networks (LSTMs) are a specialized type of RNN.
- LSTMs are capable of capturing long-term dependencies.
- Structure of the cell-state in LSTMs different from that in standard RNNs.
  - The structure facilitates relatively easy learning of long-term dependencies.
- There are four interacting neural network layers in a state cell.

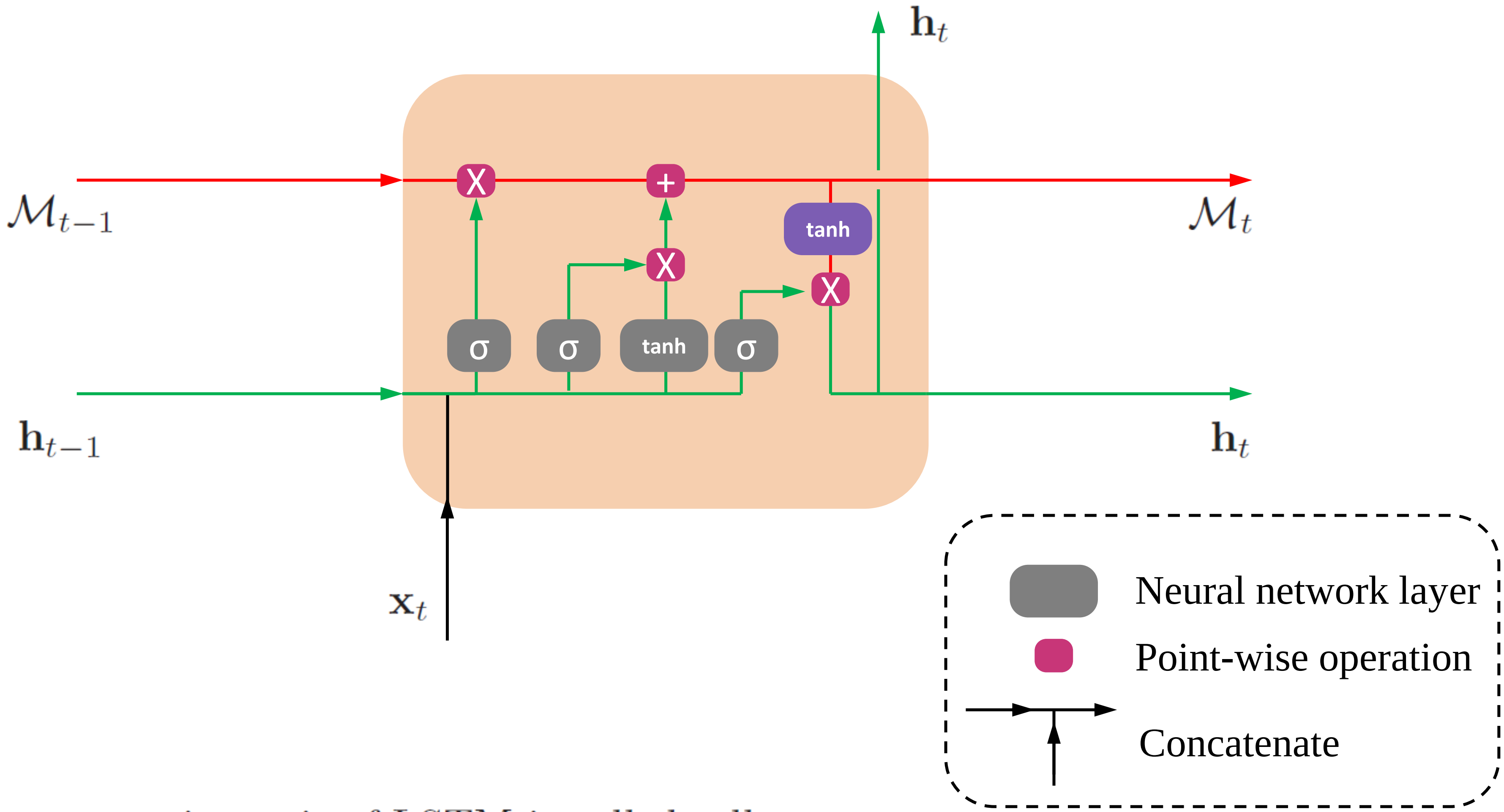
LSTM



RNN hidden unit

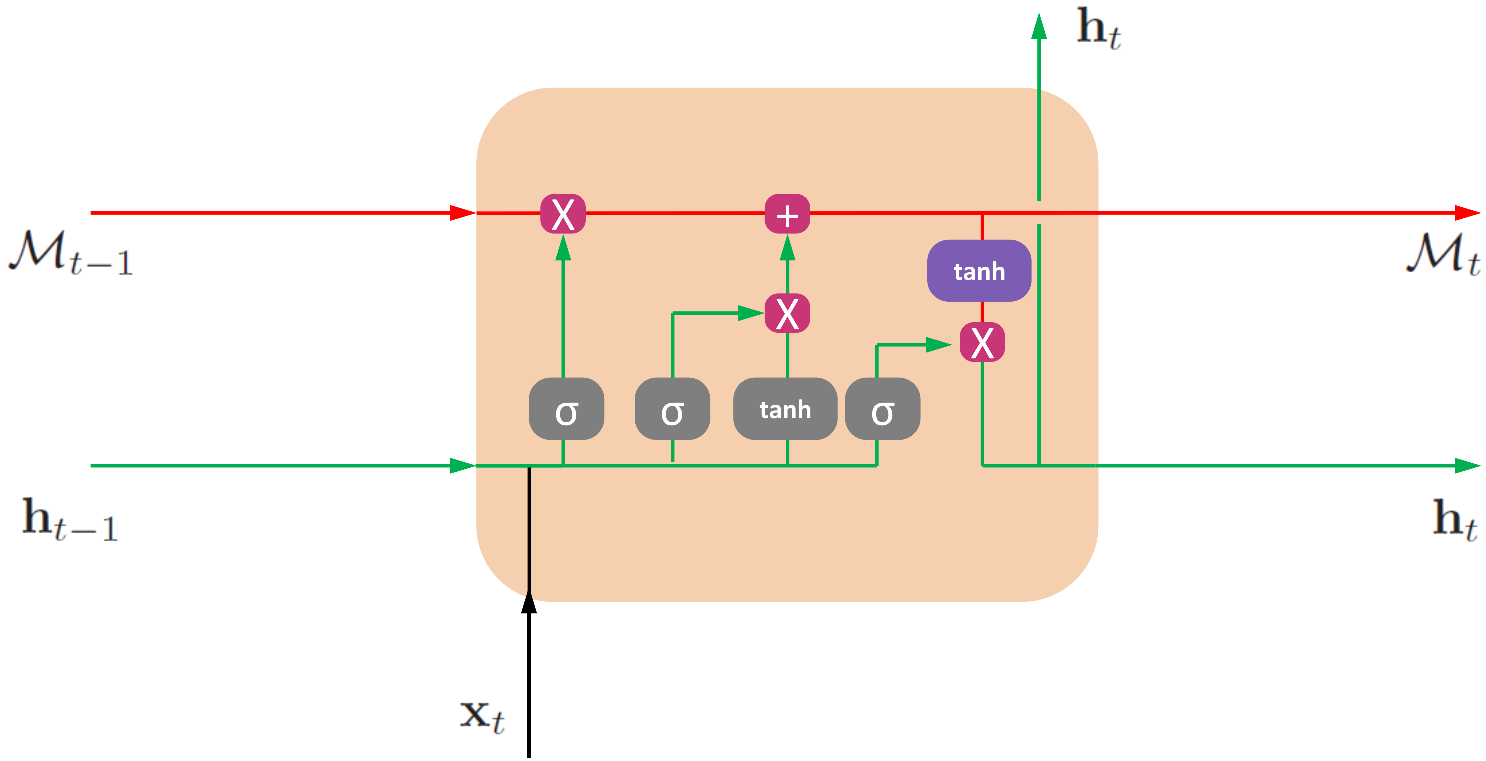


LSTM cell



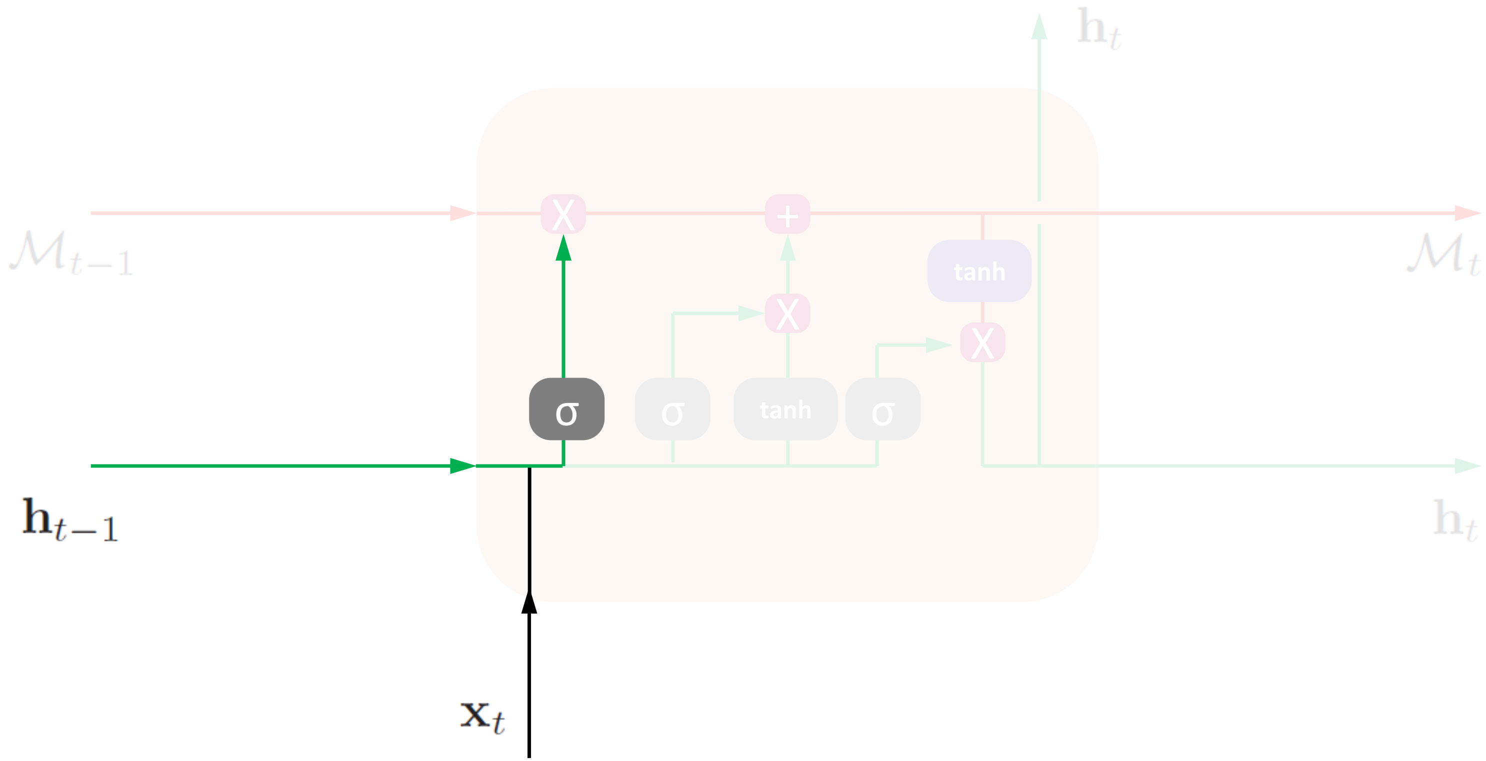
- The processing unit of LSTM is called cell.
- In RNNs hidden units are connected recurrently. In LSTMs cells are connected recurrently.

LSTM cell



- The cell state $\mathcal{M}_t$ is the long-term memory.
  - Carries information through with minor linear interactions.
  - Information is carefully added and deleted from the cell state using structures called gates.
- The hidden state $\mathbf{h}_t$ is the working memory.

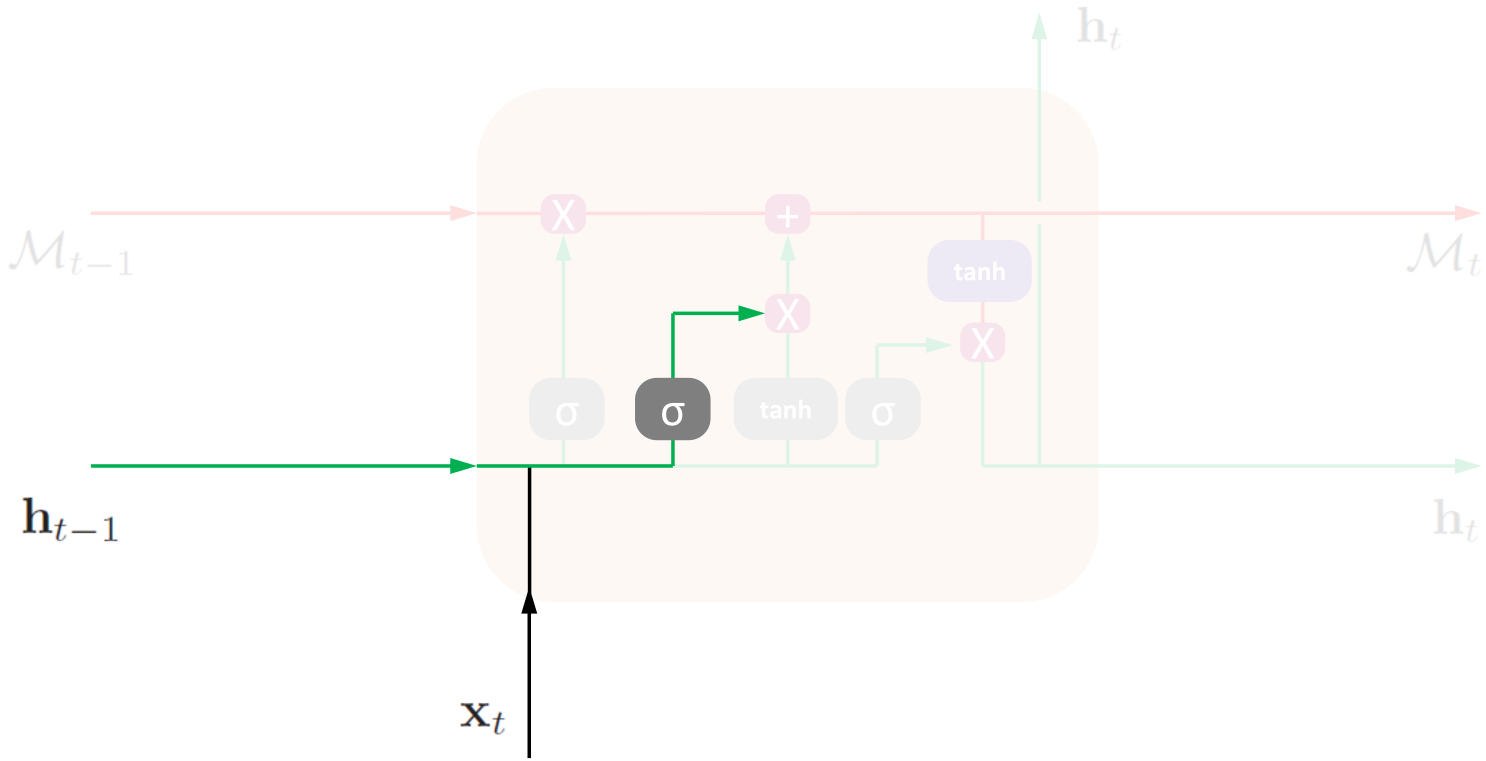
Forget gate layer



$$\mathcal{F}_t = \sigma\left(\mathbf{w}_{\mathcal{F}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{F}0}\right)$$

- Decides what information coming from the previous memory cell is to be carried forward or deleted.
- The layer outputs a value between 0 and 1.
  - 0 means "fully delete"
  - 1 means "fully reatain"

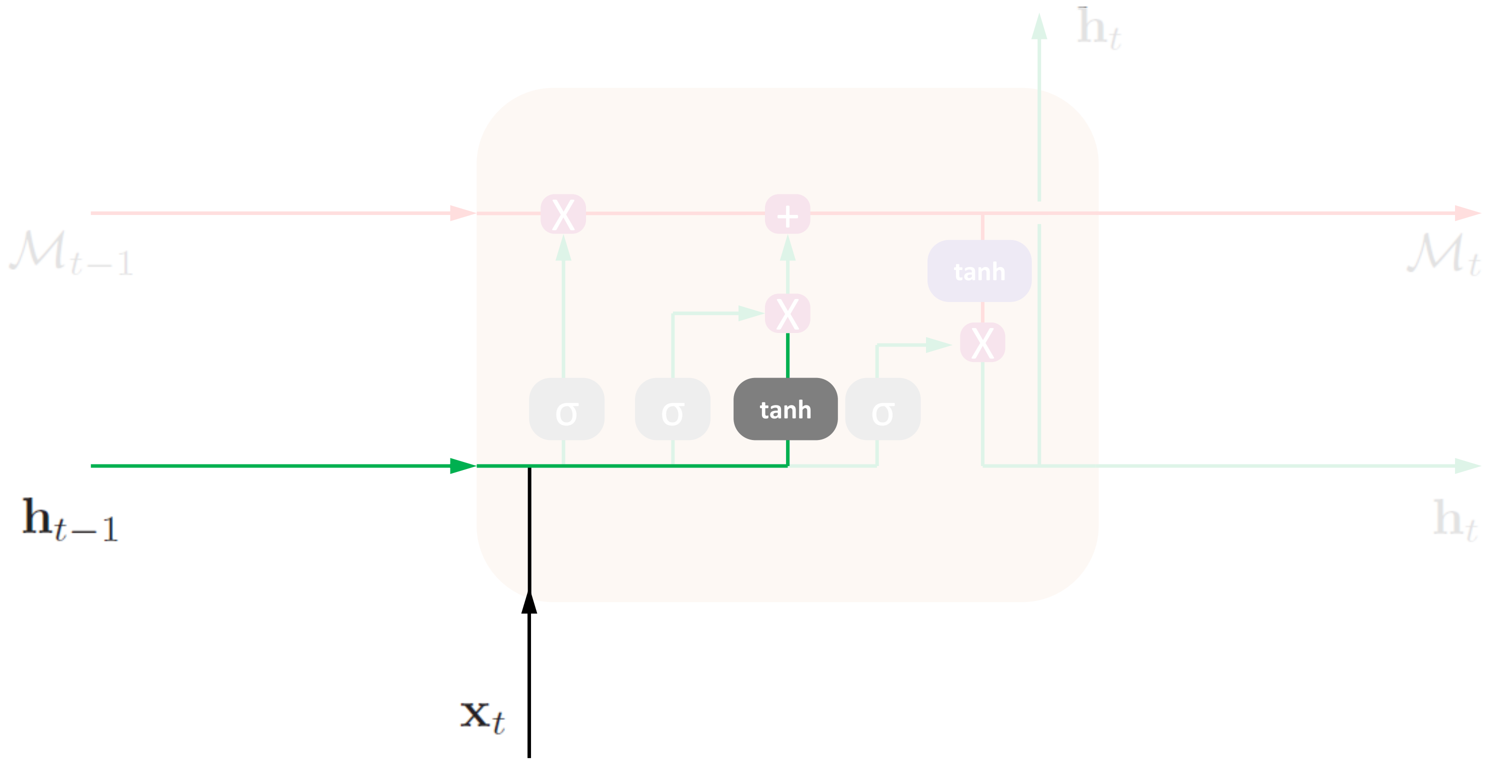
Input gate layer – 1



$$\mathcal{I}_t = \sigma\left(\mathbf{w}_{\mathcal{I}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{I}0}\right)$$

- Decides the values in the memory cell that are to be updated.

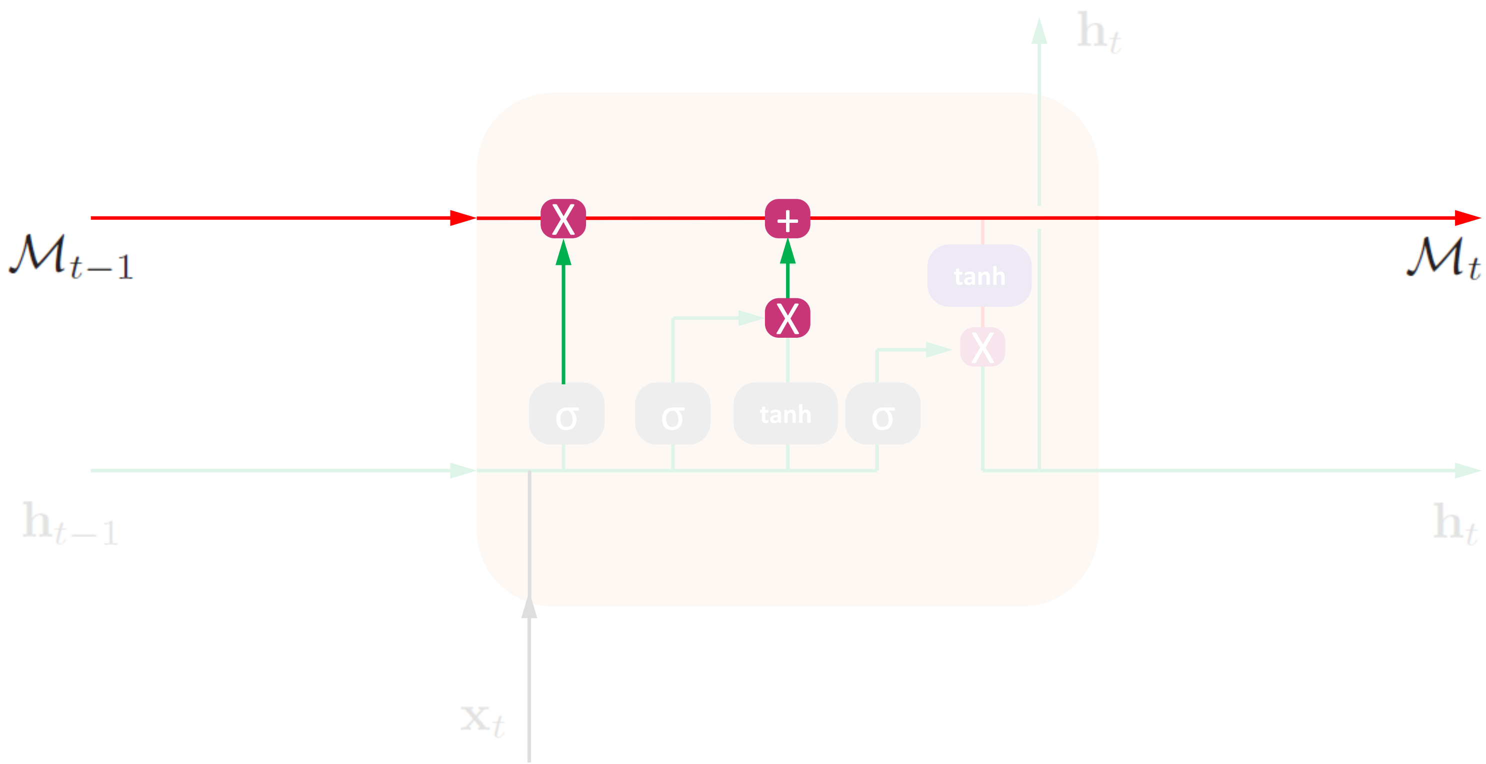
Input gate layer – 2



$$\overline{\mathcal{M}}_t = \tanh\left(\mathbf{w}_{\mathcal{M}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{M}0}\right)$$

- Generates new values with the potential to be added to the memory cell.

Memory cell update

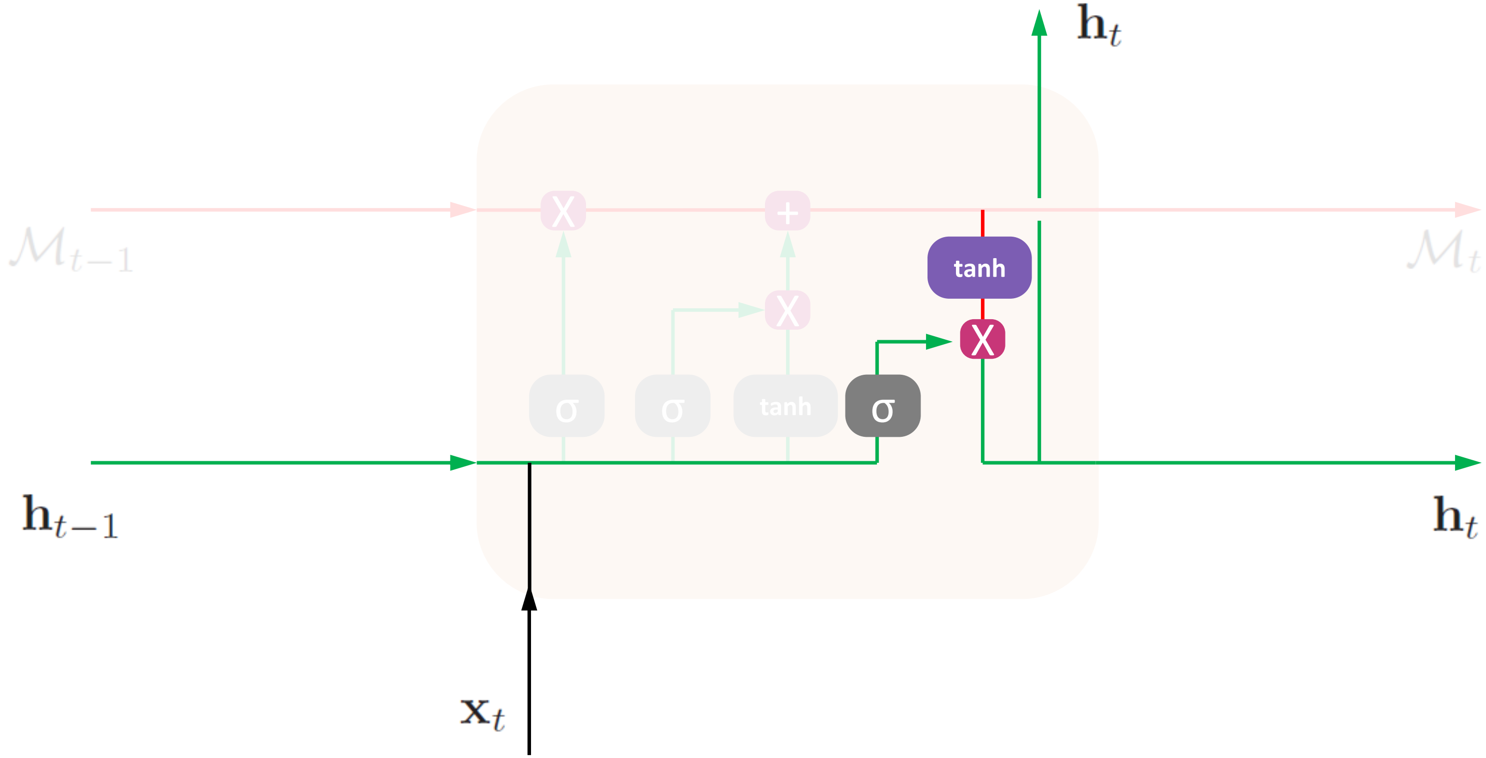


$$\mathcal{M}_t = \mathcal{F}_t \circ \mathcal{M}_{t-1} + \mathcal{I}_t \circ \overline{\mathcal{M}}_t$$

where $\circ$ denotes element-wise multiplication.

- The old memory cell is multiplied by $\mathcal{F}_t$ to forget values that are irrelevant.
- New updates are then made to the memory cell through $\mathcal{I}_t \circ \overline{\mathcal{M}}_t$.

Output



- The current memory cell is passed through tanh so that the output values lie between -1 and 1.

$$h_t = \mathcal{O}_t \circ \tanh(\mathcal{M}_t)$$

- The sigmoid layer decides the components of cell state that are going to be passed

$$\mathcal{O}_t = \sigma\big(\mathbf{w}_{\mathcal{O}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{O}0}\big)$$

Shortcomings of LSTM

- The contents get corrupted after sufficient time as the output of the gates are not exactly 0 or 1.

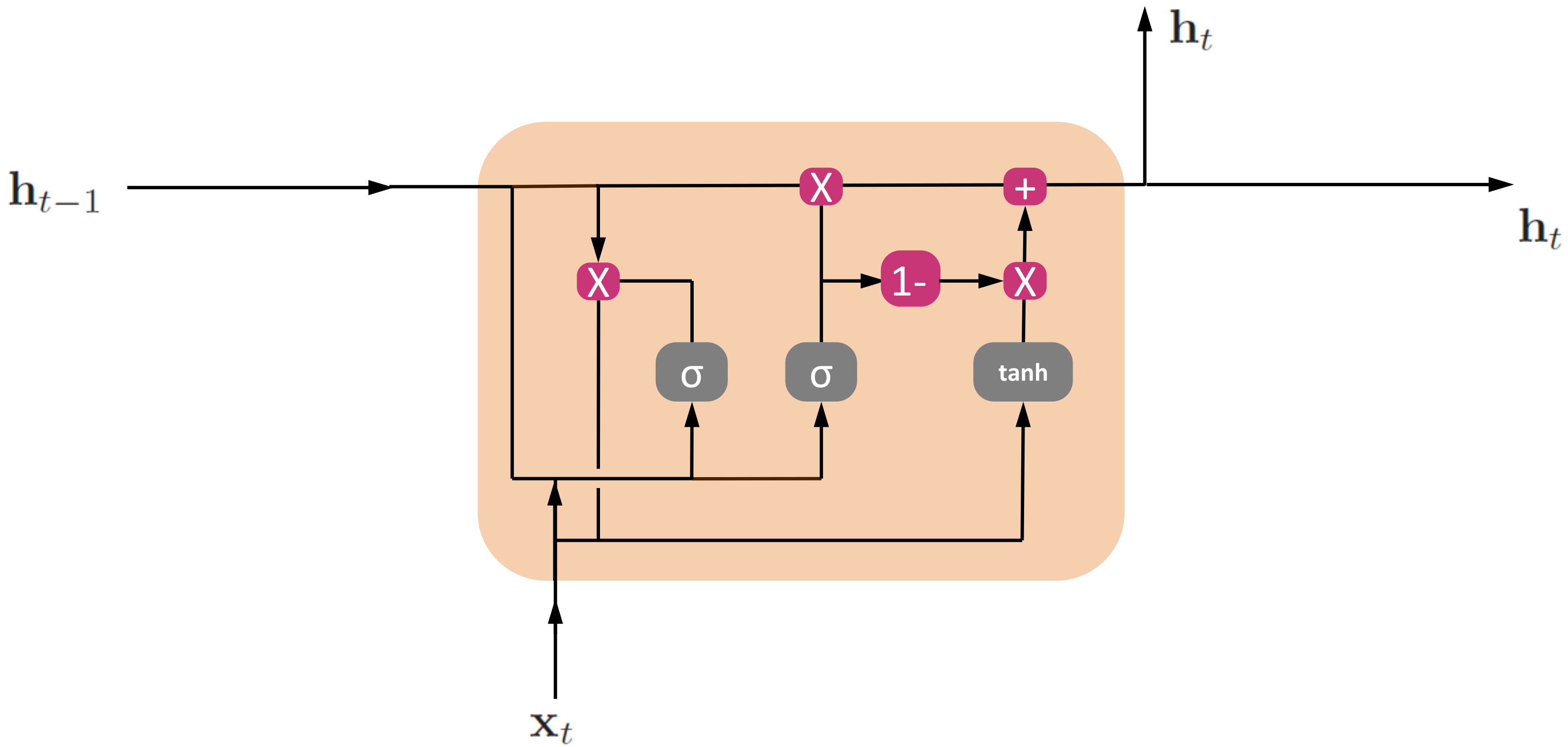
- LSTMs are much better than RNNs, but still has difficulties in capturing very long term dependencies.

- Large number of parameters.
  - The size of the training dataset needs to be large as well.

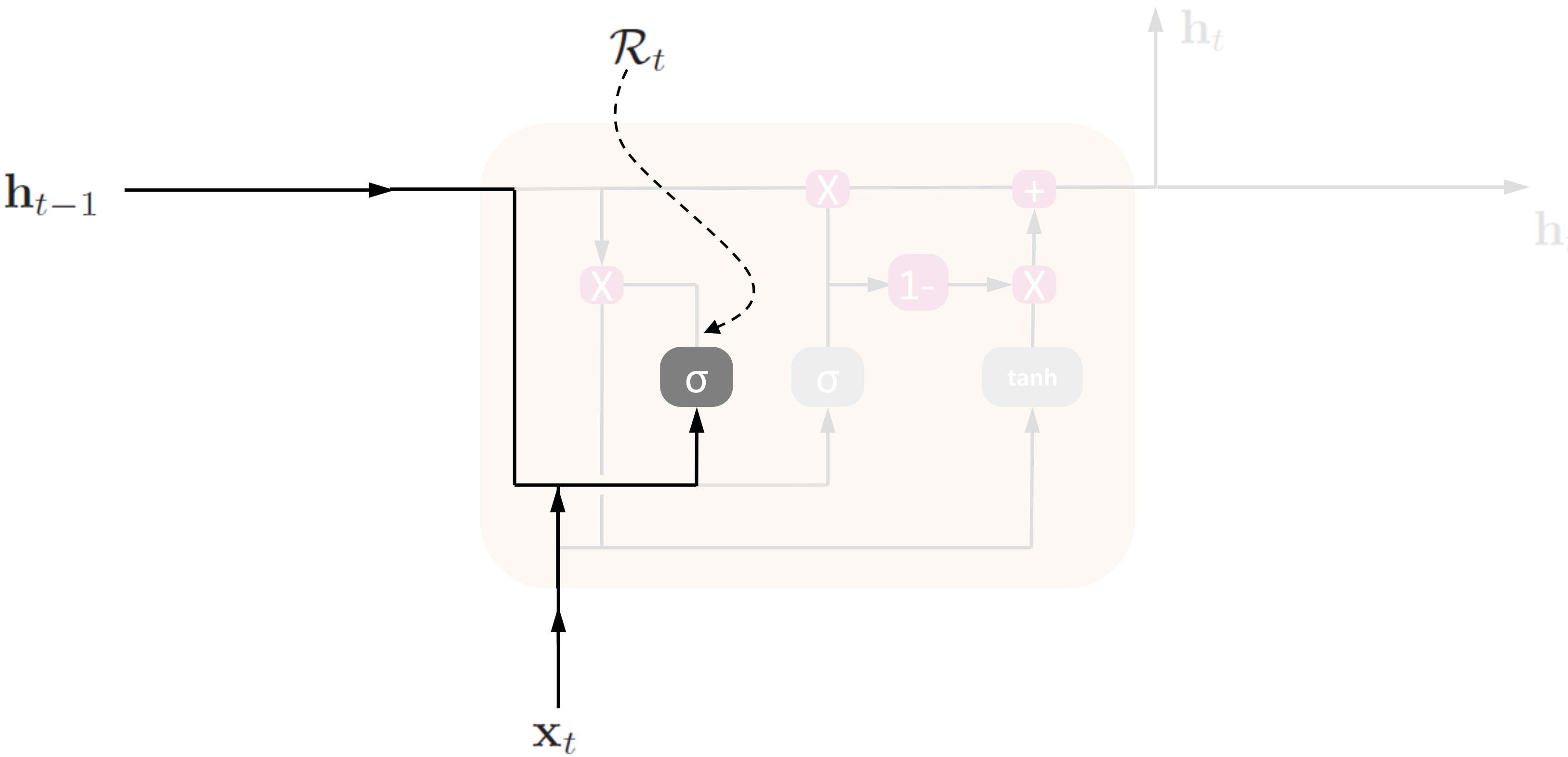
- More time needed for training.

Gated Recurrent Units

GRU cell



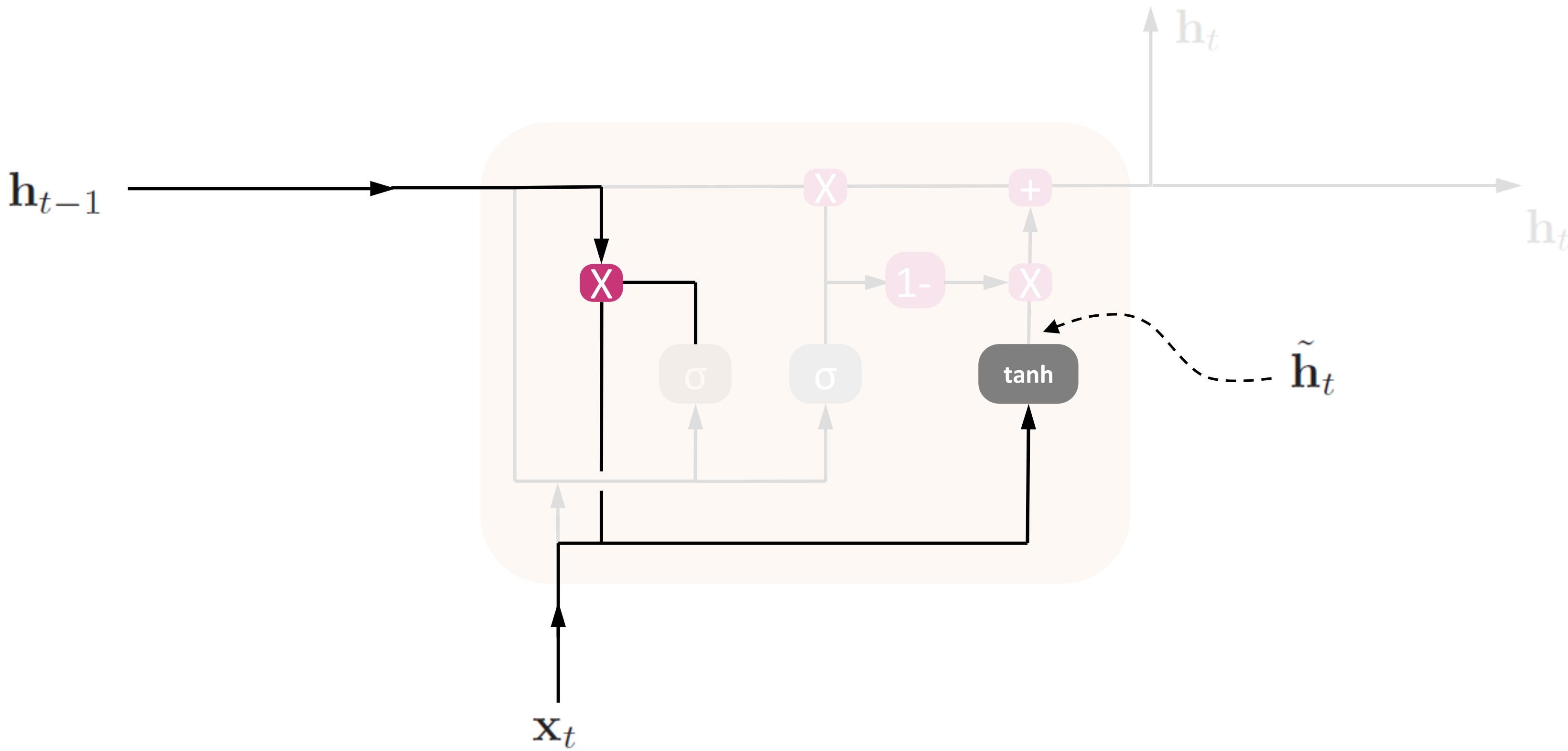
Reset gate



$$\mathcal{R}_t = \sigma\big(\mathbf{w}_{\mathcal{R}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{R}0}\big)$$

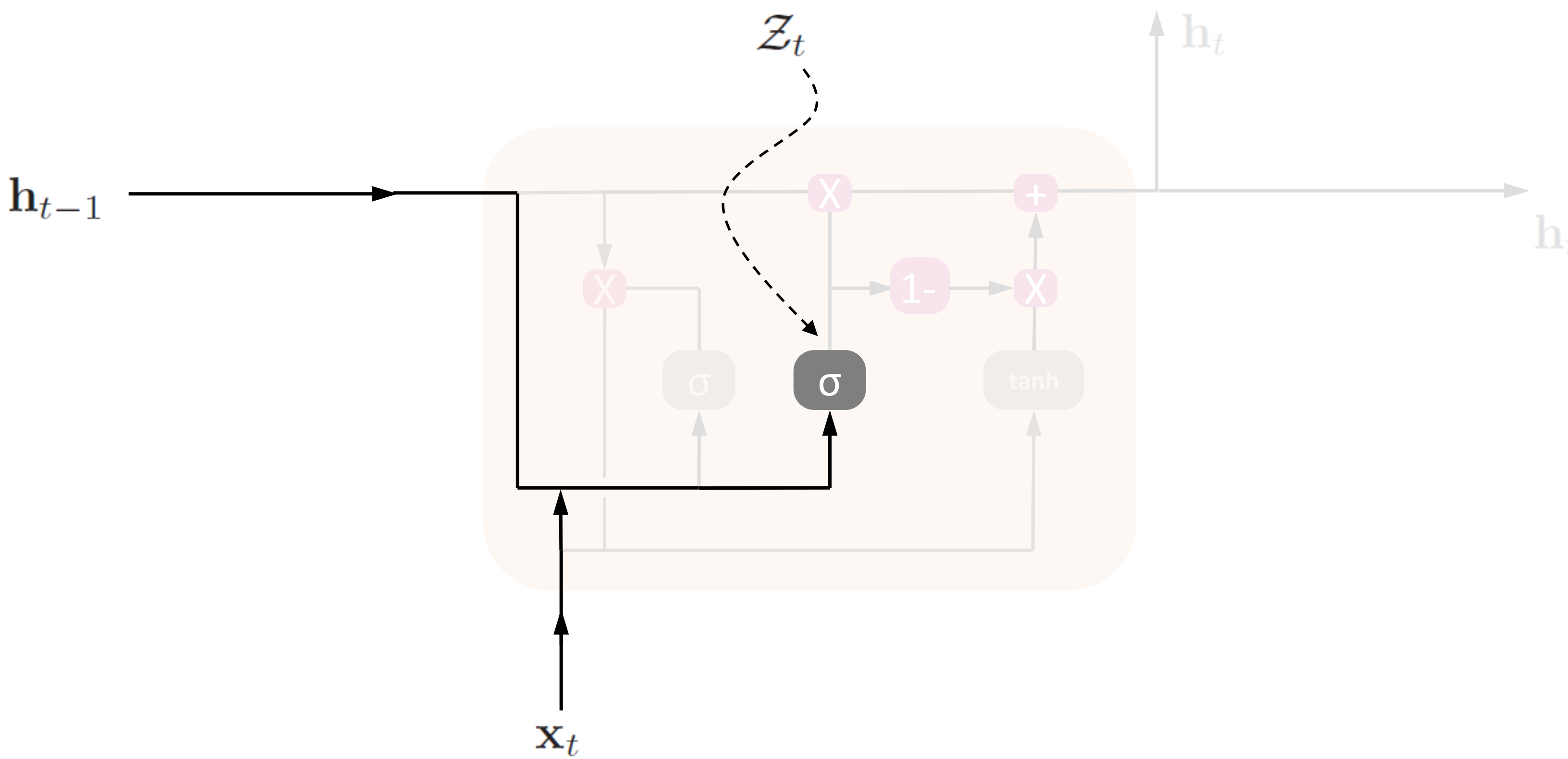
- Reset gate controls how much of the previous state we may want to remember.
  - $\mathcal{R}_t \to 1$: conventional RNN
  - $\mathcal{R}_t \to 0$: MLP

Candidate hidden state



$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{w}_h . \left[\mathcal{R}_t \circ \mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{h0}\right)$$

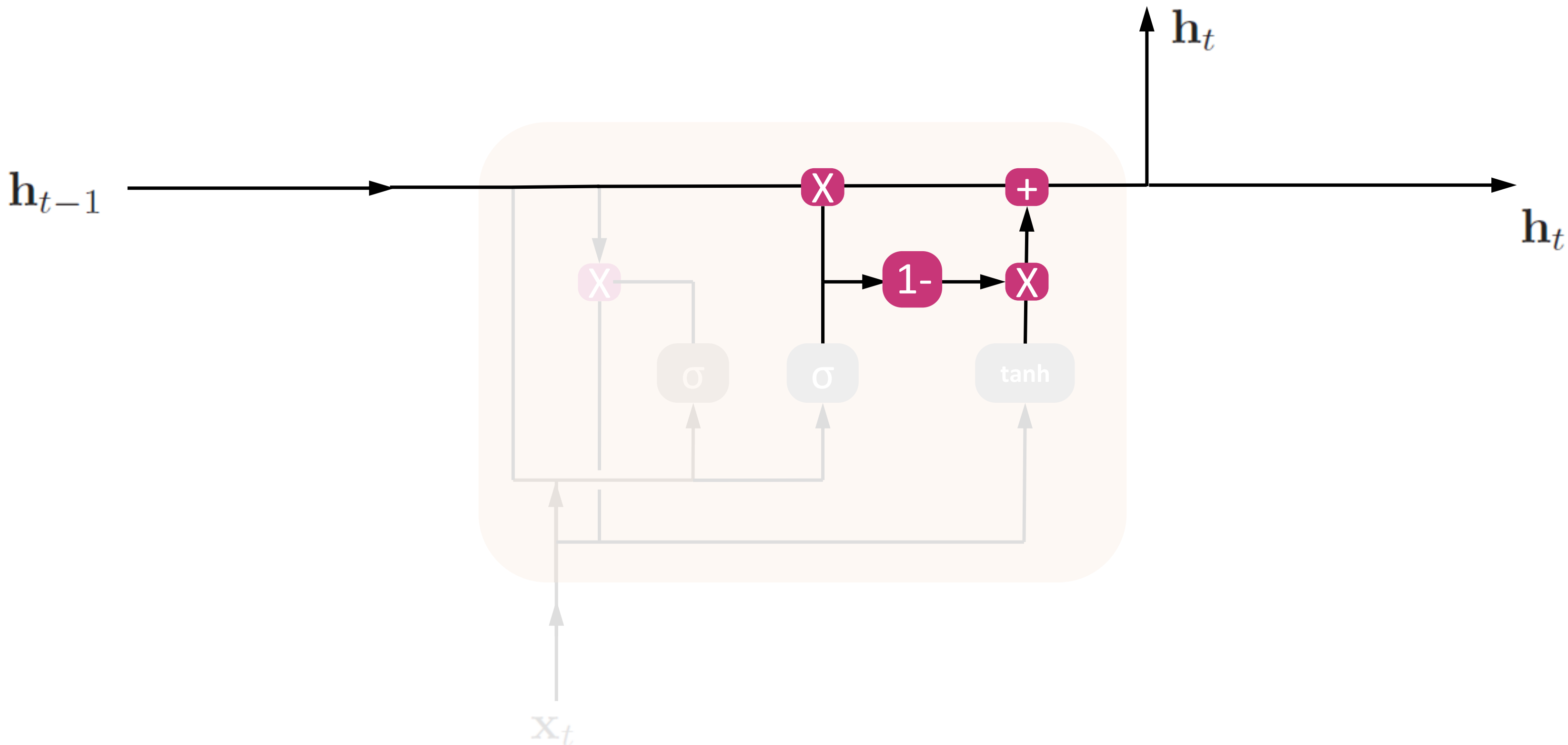
Update gate



$$\mathcal{Z}_t = \sigma\left(\mathbf{w}_{\mathcal{Z}}.\left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{w}_{\mathcal{Z}0}\right)$$

- Determines how much the old state $\mathbf{h}_{t-1}$ is modified and how much the new candidate state $\tilde{\mathbf{h}}_t$ is used

New hidden state



$$\mathbf{h}_t = \mathcal{Z}_t \circ \mathbf{h}_{t-1} + (1 - \mathcal{Z}_t) \circ \tilde{\mathbf{h}}_t$$