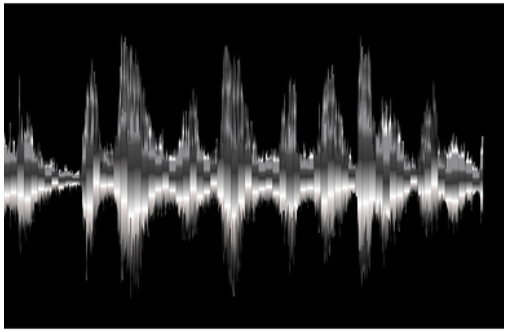
Deep Learning for sequential data

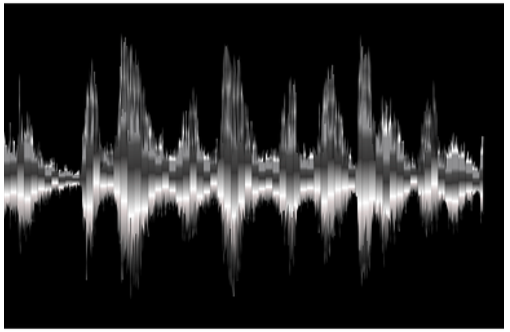
Sequential data

Text, Video, and Audio



Sequential data

Text, Video, and Audio



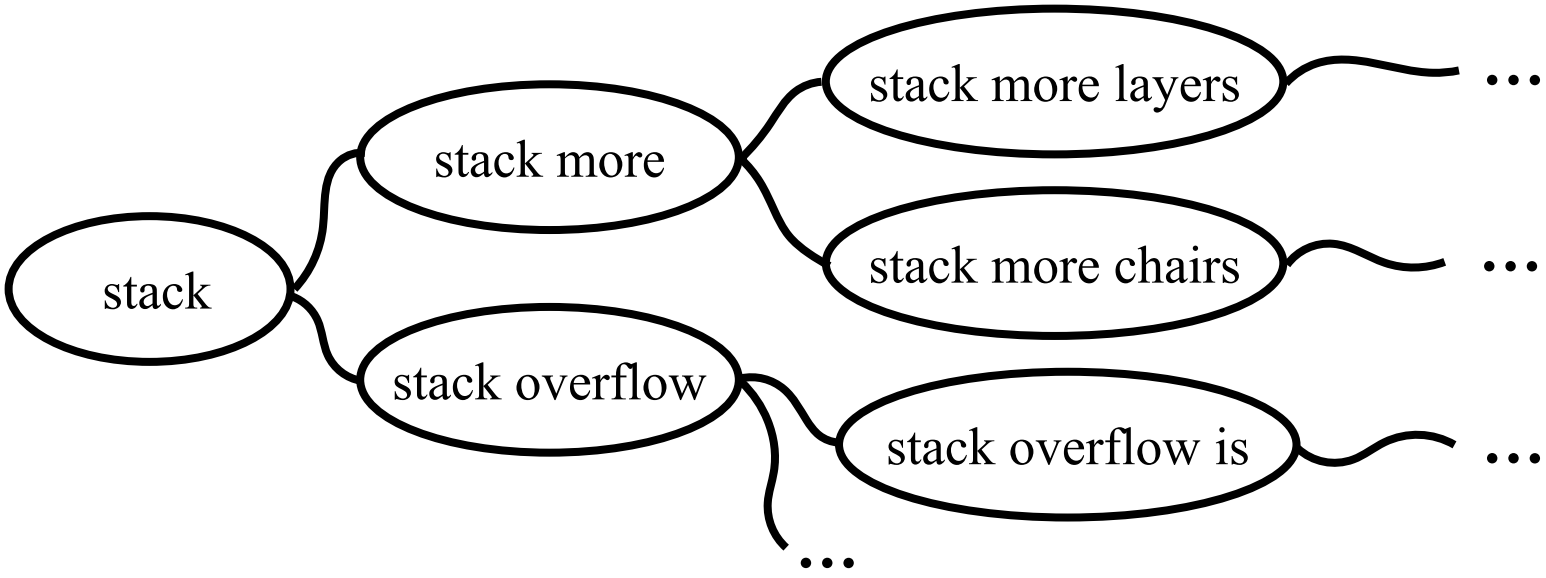
Time series: finance, industry, medicine…

Sequences are everywhere!

Language model

We want to train a generative model of natural language

$$P(text) = P(x_0, \ldots, x_n) =$$
$$= P(x_0)P(x_1|x_0)P(x_2|x_0, x_1) \ldots P(x_n| \ldots)$$



Language model

We want to train a generative model of natural language

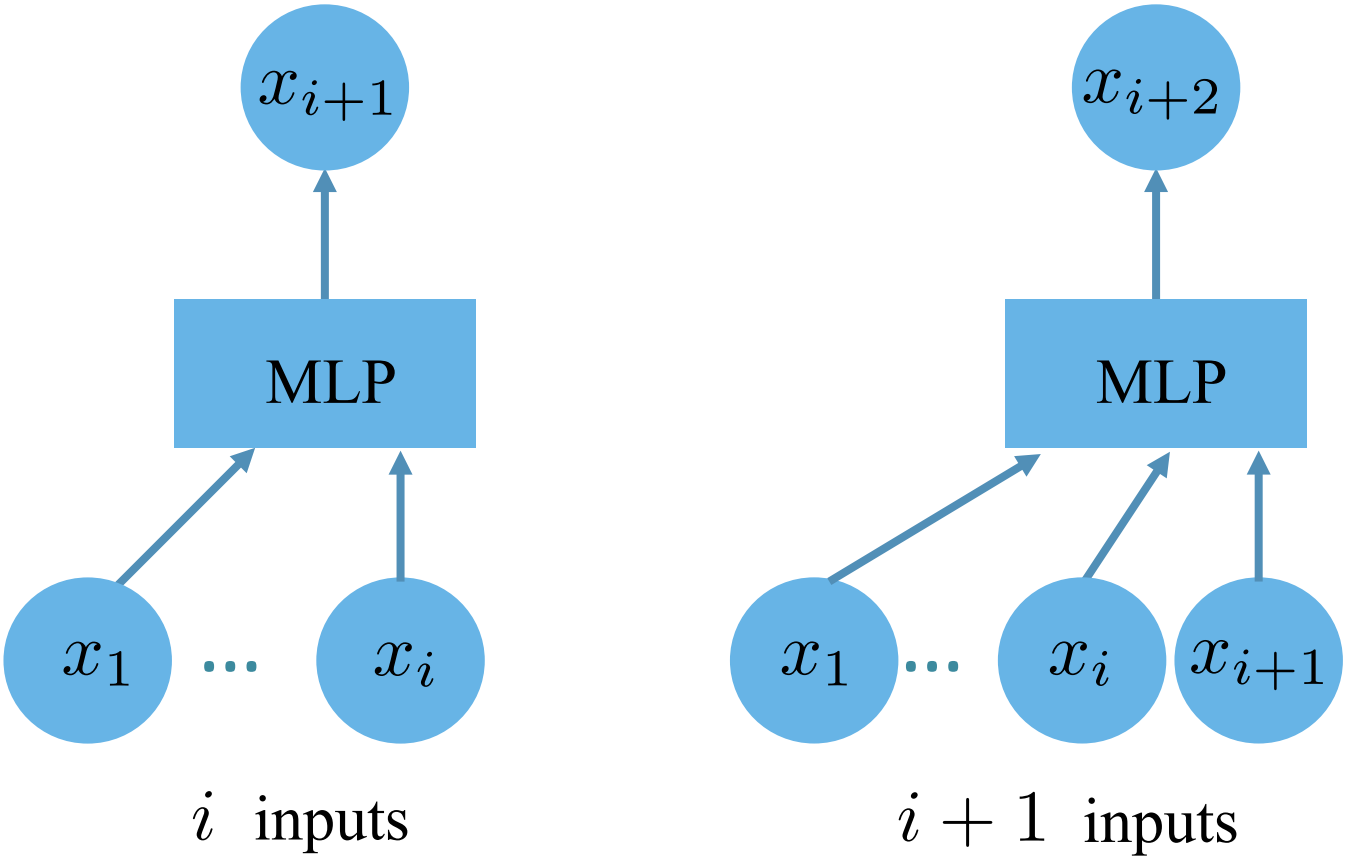
$$P(text) = P(x_0, \ldots, x_n) =$$
$$= P(x_0)P(x_1|x_0)P(x_2|x_0, x_1)\ldots P(x_n|\ldots)$$

Why do we need it?

- Chatbots, question answering
- Machine translation
- Speech recognition
- Any text analysis you can imagine

Why not MLP?

The main problem is arbitrary length of sequences:



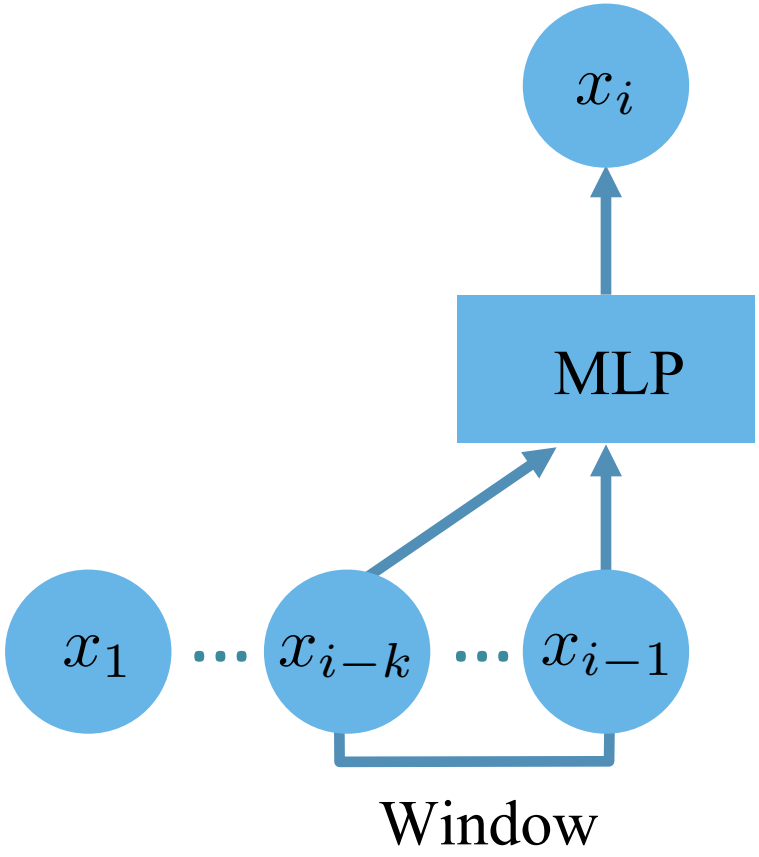
i inputs

$i+1$ inputs

How can we overcome it?

Why not MLP?

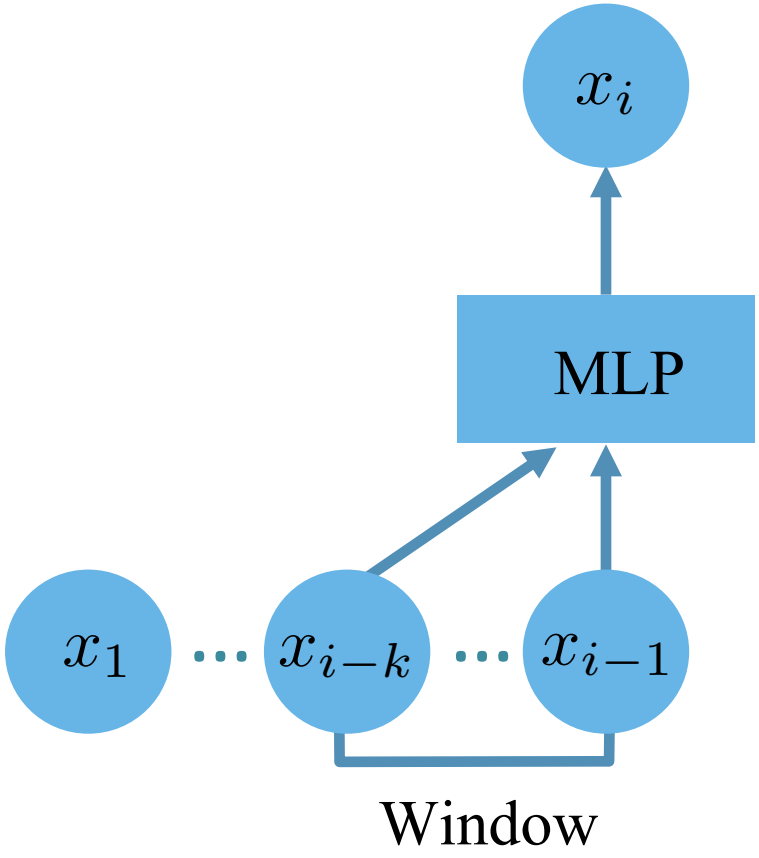
We can use a window of a fixed size as an input.



Why not MLP?

We can use a window of a fixed size as an input.

- This is just a heuristic and it is not clear how to choose the width of the window
- In some tasks we need very wide window therefore there is a problem with the large number of parameters



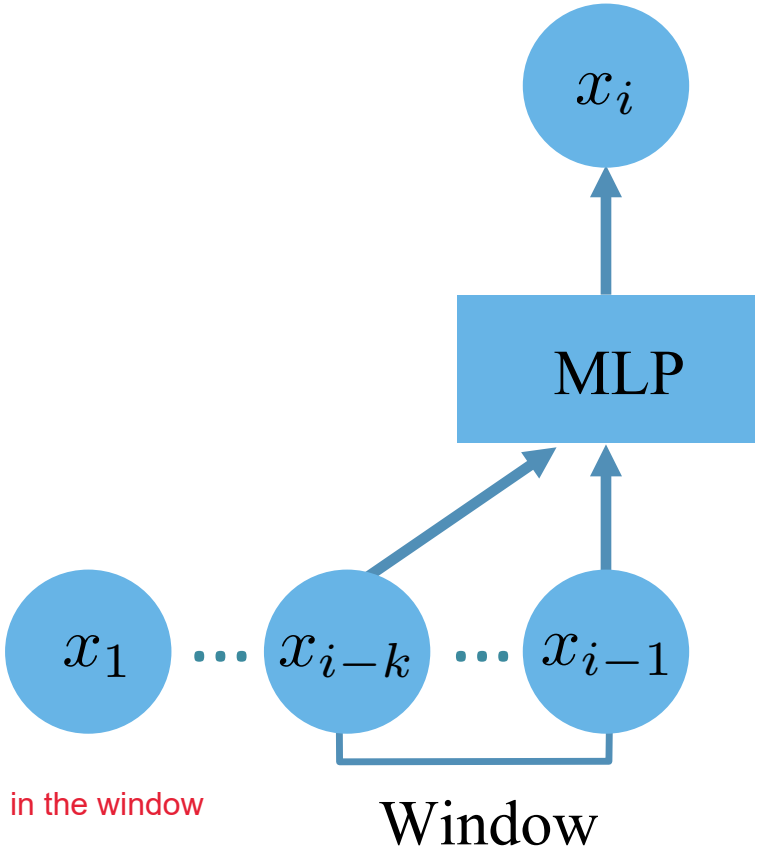
Why not MLP?

We can use a window of a fixed size as an input.

Question

How many weights are there in the first layer of the MLP?

- hidden neurons: 100
- window width: 100
- word embeddings size: 100



100 * 100 inputs to the 1st layer; 100 words for each word in the window
weight matrix contain 1M parameters
in addition to 100 parameters of bias vec

in total = 1,000,100

Why not MLP?

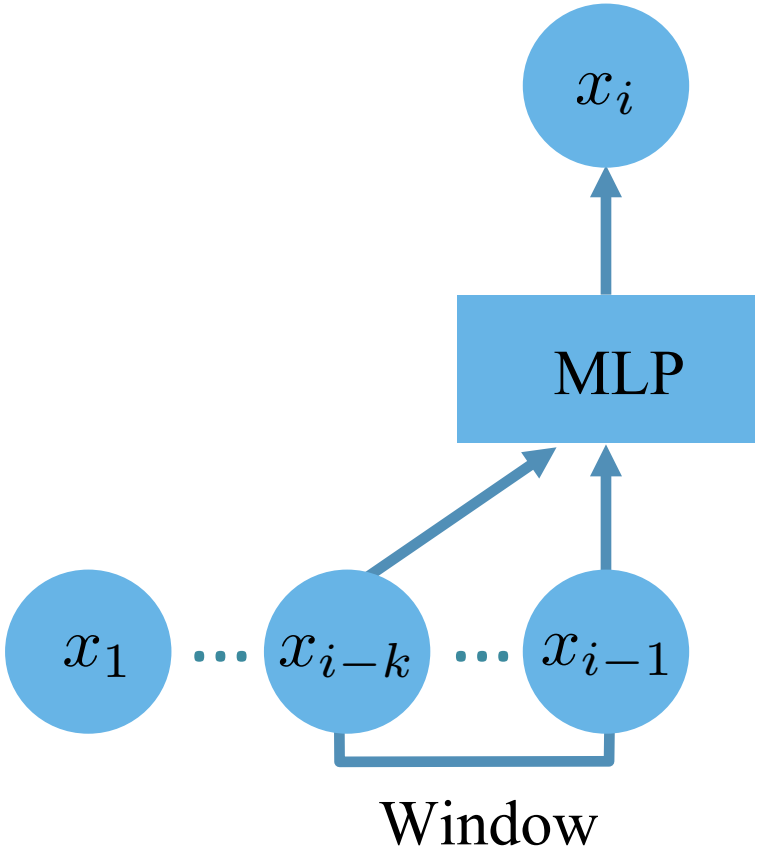
We can use a window of a fixed size as an input.

Question

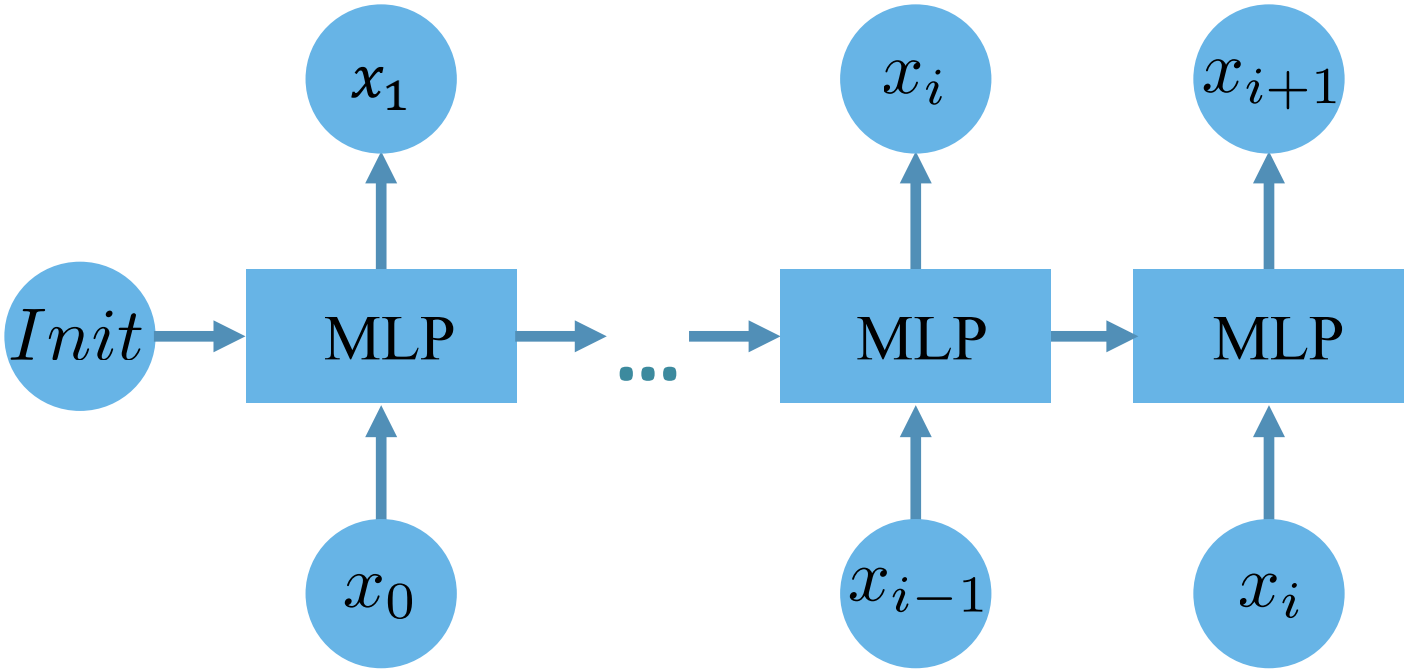
How many weights are there in the first layer of the MLP?

- hidden neurons: 100
- window width: 100
- word embeddings size: 100

More than a million!



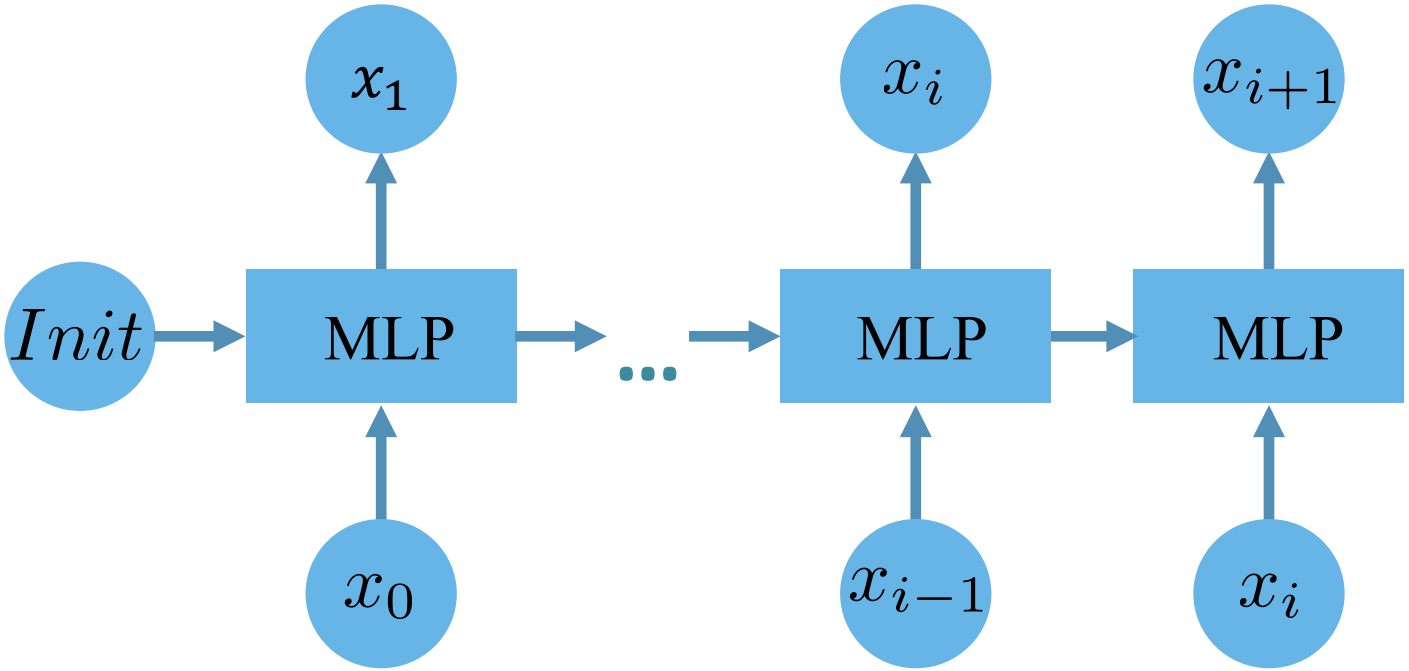
Recurrent Architecture



Problem #1: Arbitrary sequence length

Here: Fixed number of inputs at each time step.
At the first step we use some initial vector as an input from previous time step.

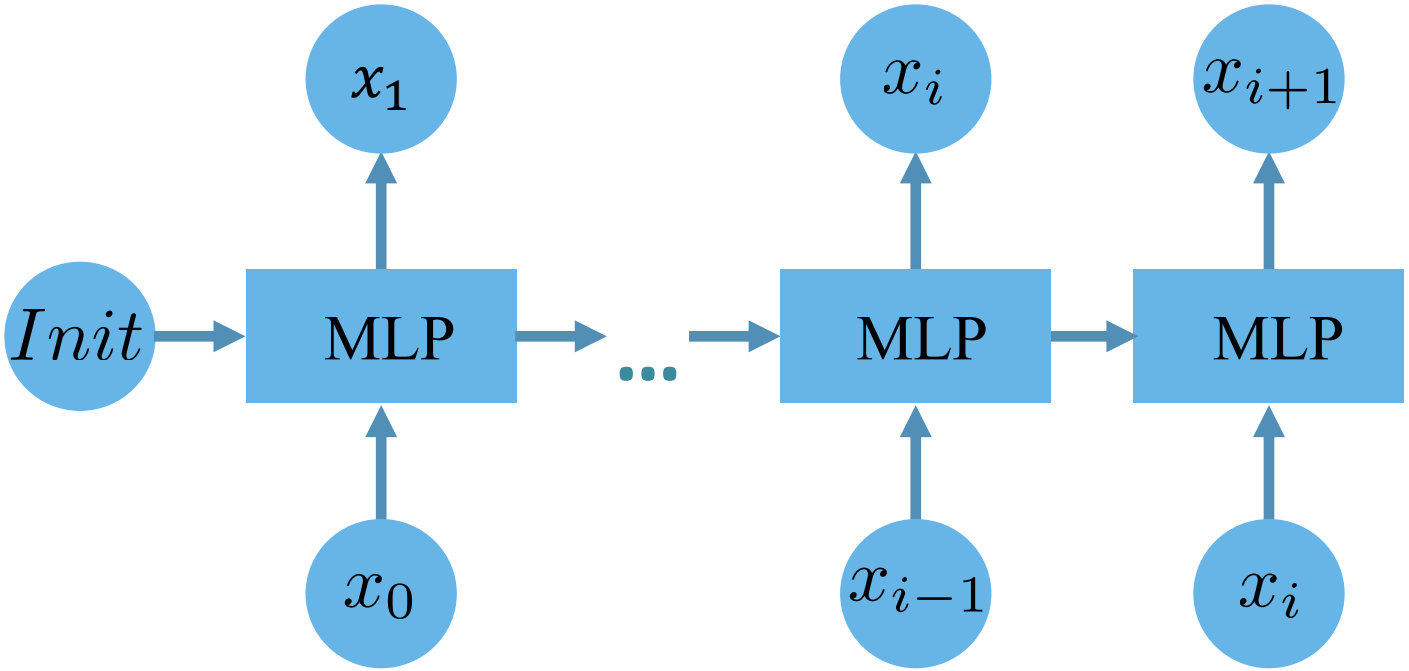
Recurrent Architecture



Problem #2: Large number of parameters

Here: All the parameters of an MLP are shared across the different time steps so we need a much smaller number of parameters.

Recurrent Architecture



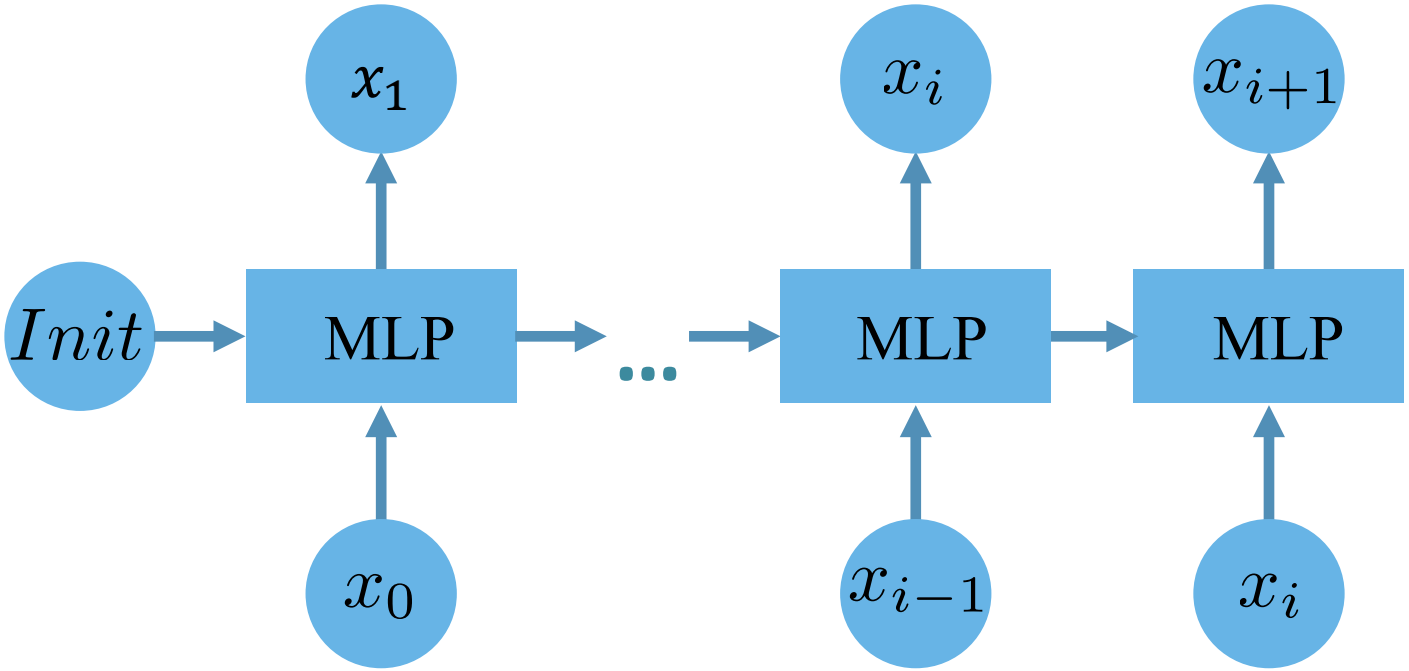
Question: How many weights are there in the first layer of the MLP?

- hidden neurons: 100
- word embeddings size: 100

100 + 100 inputs to the first layer;
100 from current input & 100 from previous hidden state
weight matrix contain 200 * 100 parameters,
in addition to 100 parameters for the bias vec

20 100 in total

Recurrent Architecture



Question: How many weights are there in the first layer of the MLP?

- hidden neurons: 100
- word embeddings size: 100

Only 20100!

Summary

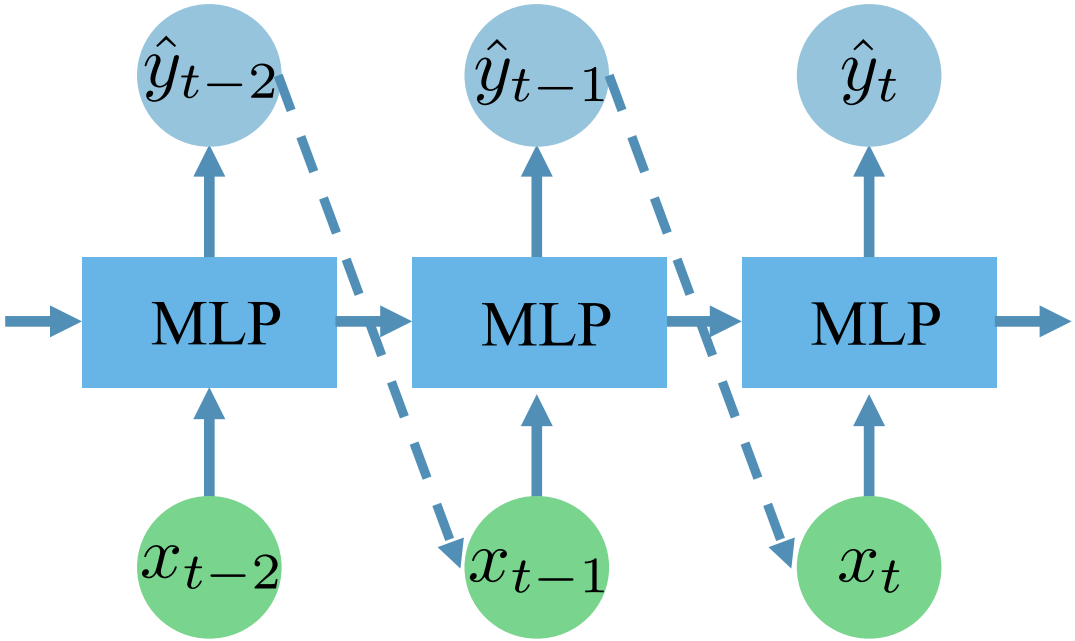
- Sequential data is everywhere!
- Feedforward neural network isn't a very natural choice for such data because of arbitrary sequence length and large number of parameters
- Recurrent architecture is much more useful

In the next video:

Simple Recurrent Neural Network:

what is it and how to train it

Previously on this week: Recurrent Architecture

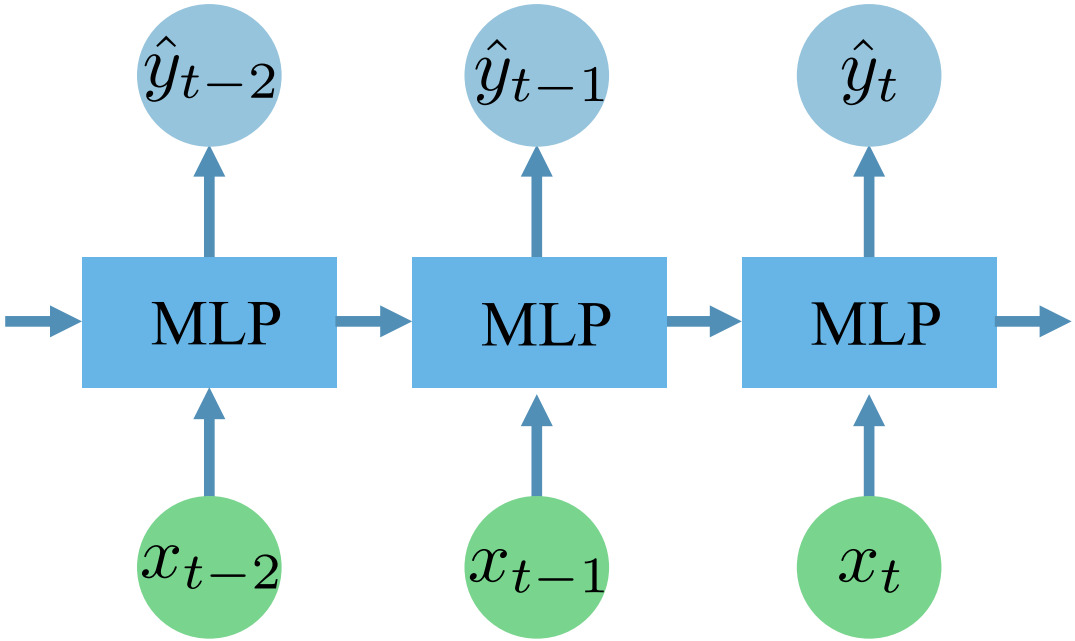


Language model:

x - word embedding

$\hat{y}$ - probability distribution for the next word

Previously on this week: Recurrent Architecture

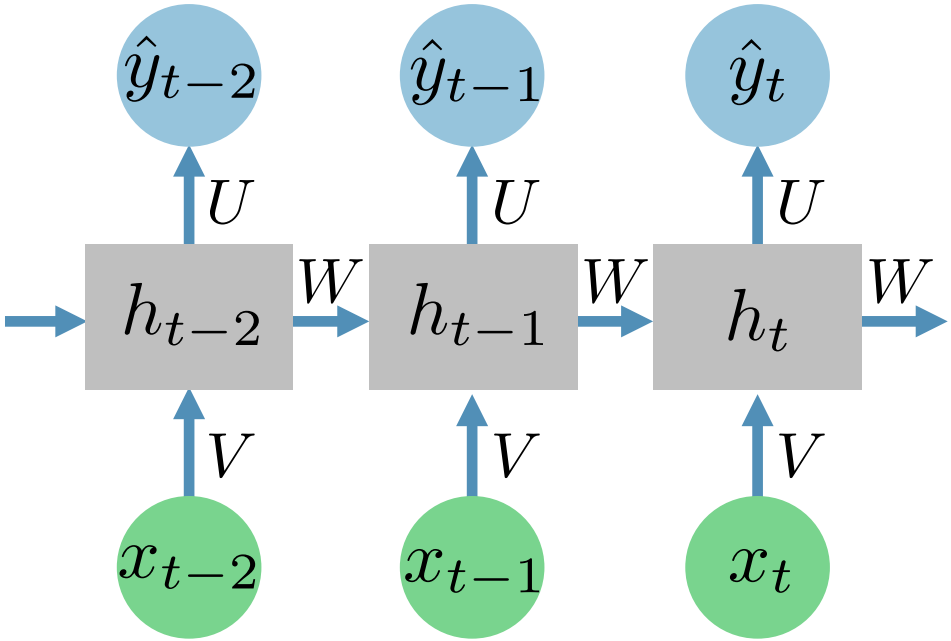


POS tagging:

x - word embedding

$\hat{y}$ - probability distribution for a POS tag of the current word

Recurrent Neural Network (RNN)

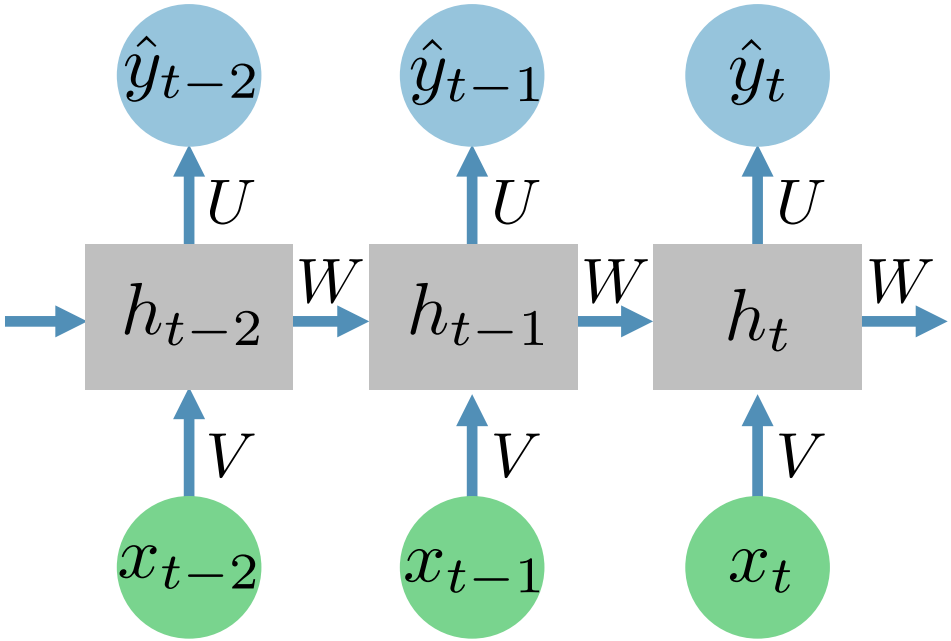


x - input

$\hat{y}$ - output (prediction)

h - hidden state

Recurrent Neural Network (RNN)



weight matrix and bias are shared at each time step

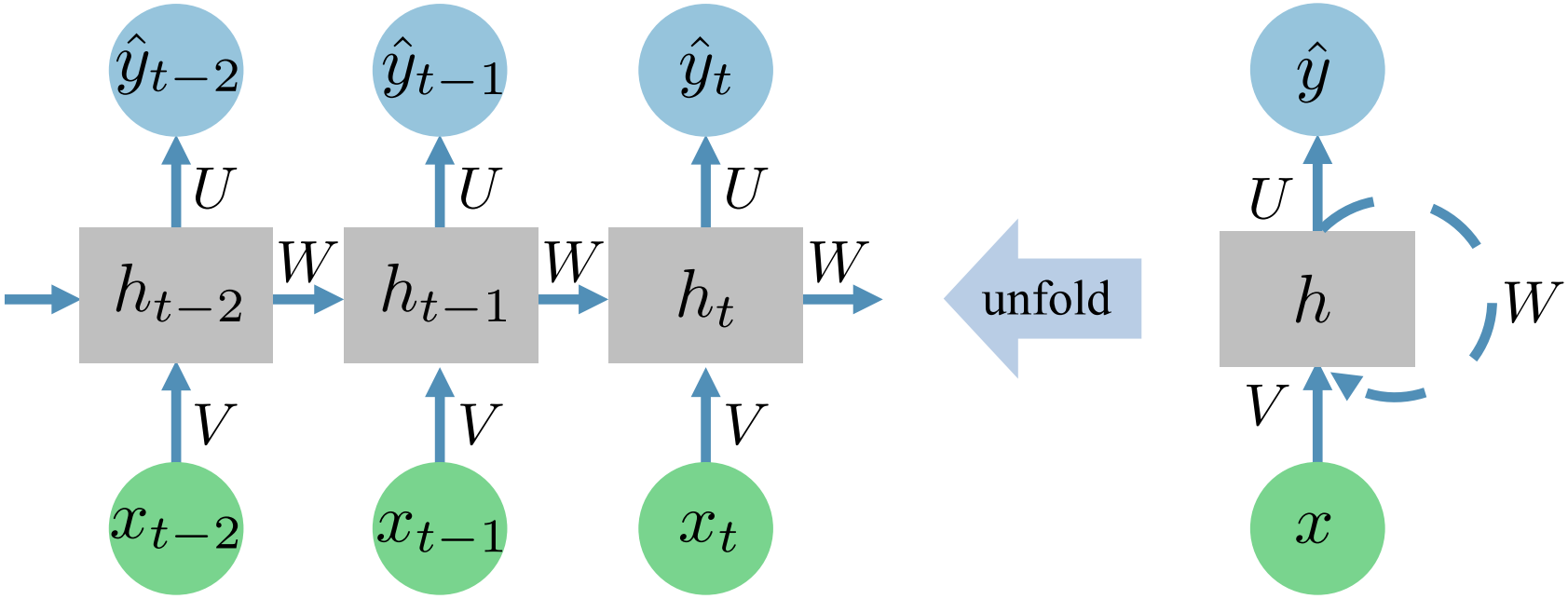
x - input

$\hat{y}$ - output (prediction)

h - hidden state

- $h_t = f_h(Vx_t + Wh_{t-1} + b_h)$
- $\hat{y}_t = f_y(Uh_t + b_y)$

Recurrent Neural Network (RNN)



x - input

$\hat{y}$ - output (prediction)

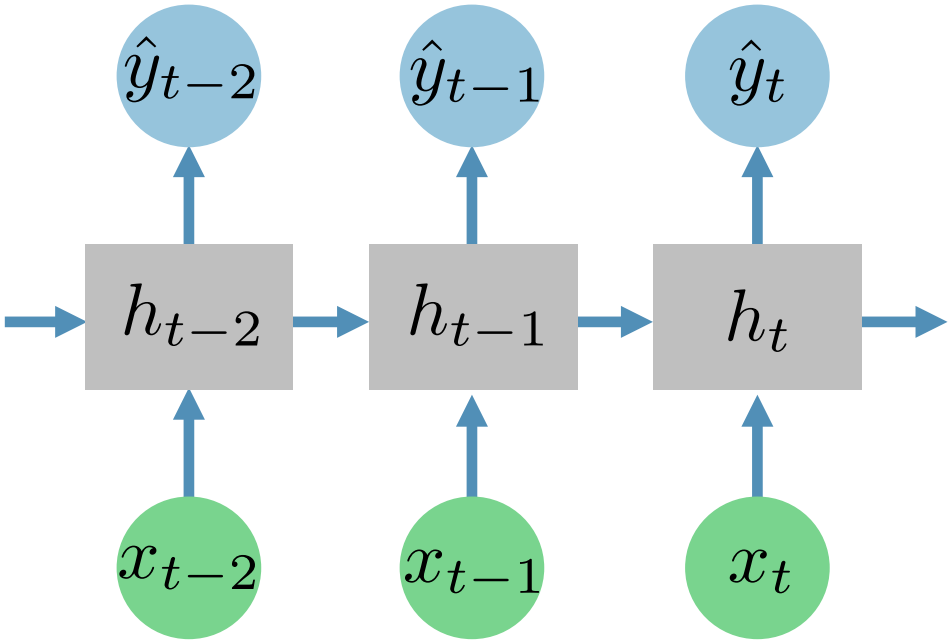
h - hidden state

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

$$\hat{y}_t = f_y(Uh_t + b_y)$$

How to train RNN?

Let's consider an RNN in the unfolded form.

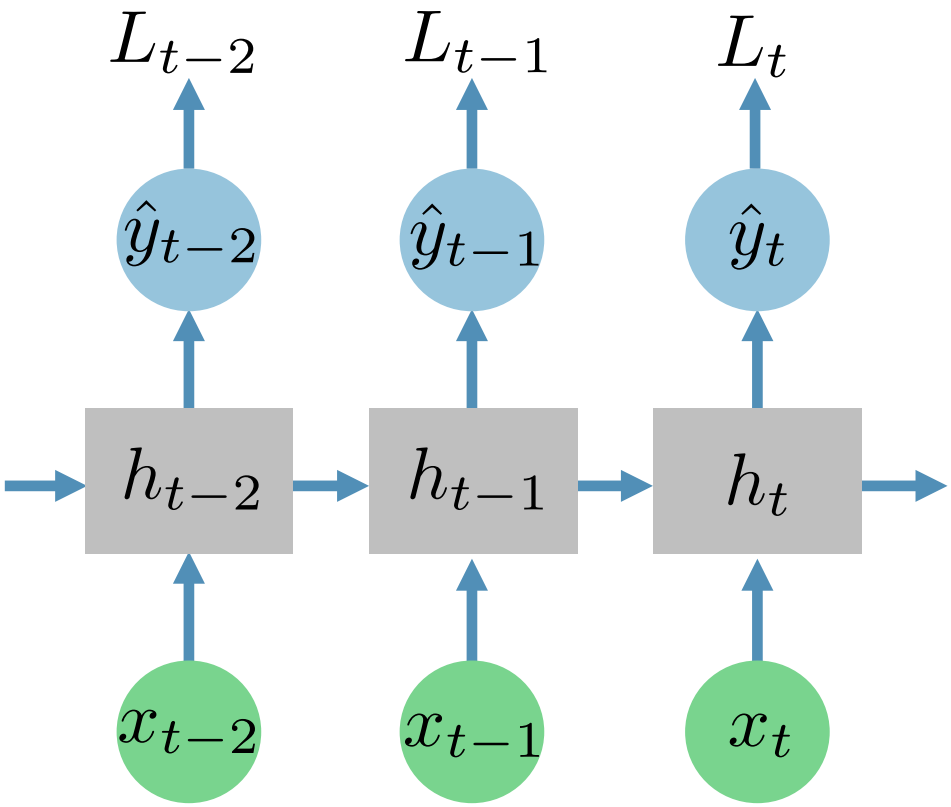


How to train RNN?

Let's consider an RNN in the unfolded form.

At each time step:

- y_t - true label
- $\hat{y}_t$ - prediction
- $L_t(y_t, \hat{y}_t)$ - some loss function



How to train RNN?

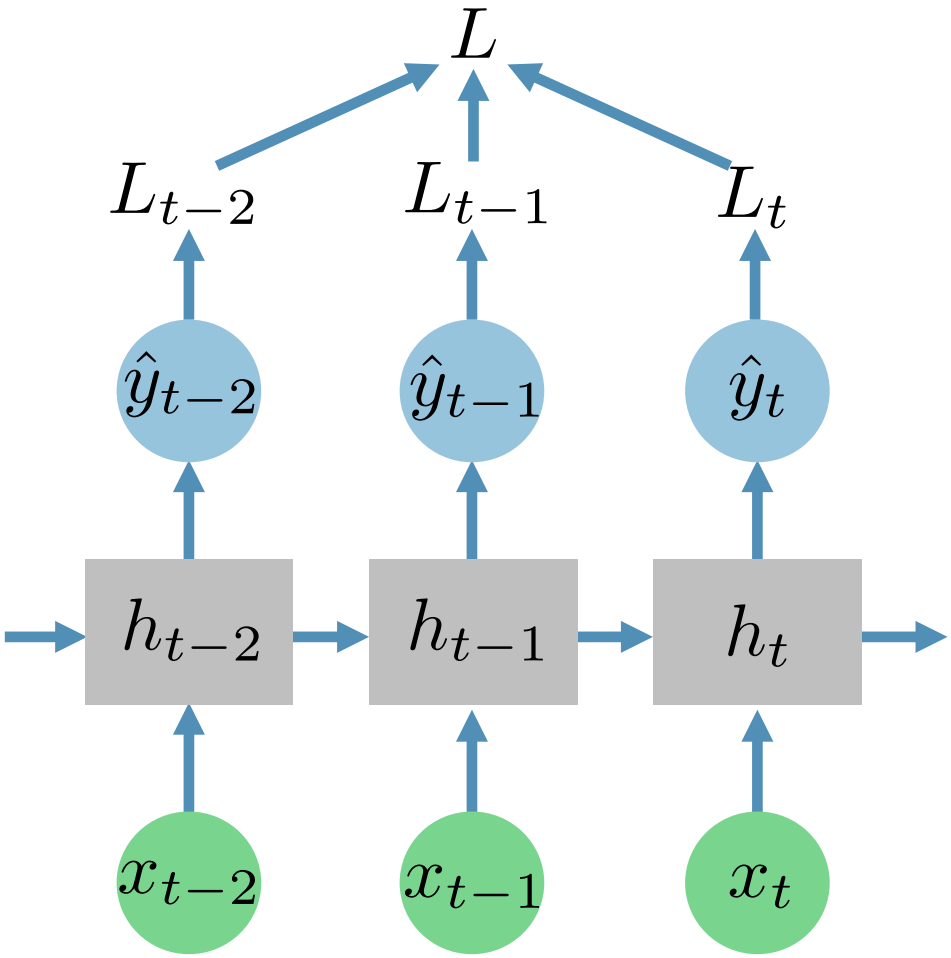
Let's consider an RNN in the unfolded form.

At each time step:

- y_t - true label
- $\hat{y}_t$ - prediction
- $L_t(y_t, \hat{y}_t)$ - some loss function

Loss:

$$L = \sum_i L_i(y_i, \hat{y}_i)$$



How to train RNN?

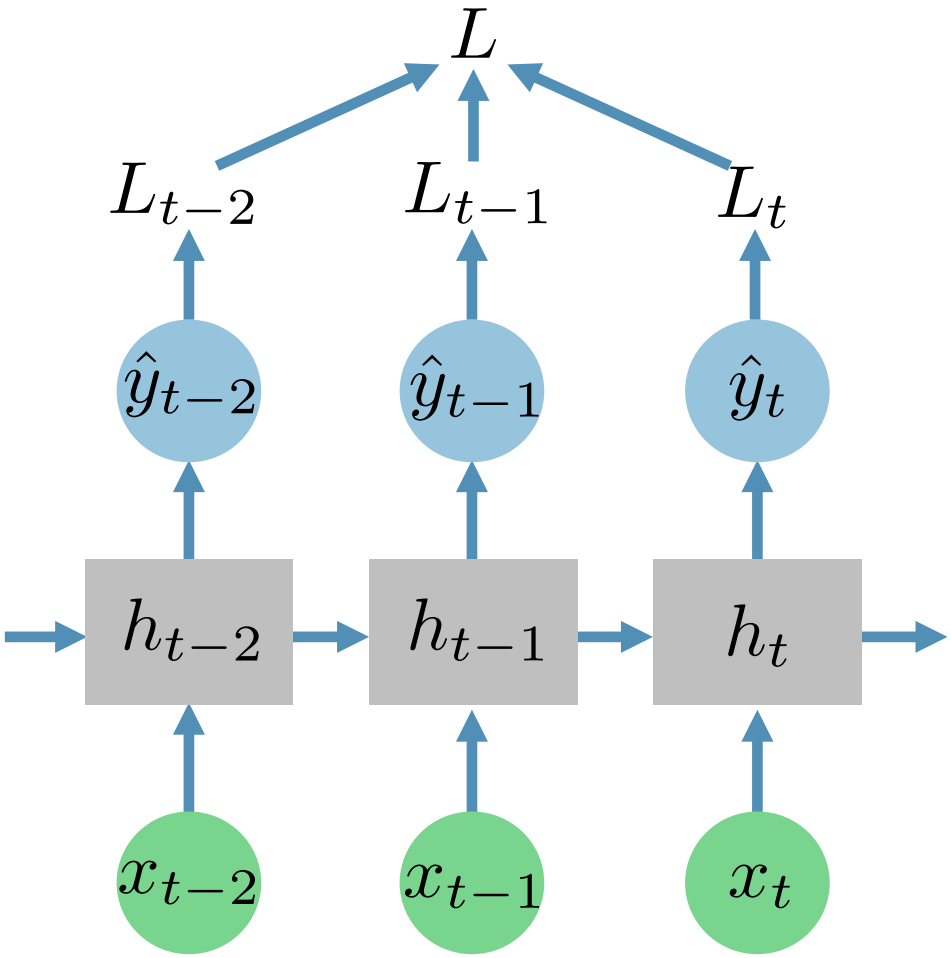
Let's consider an RNN in the unfolded form.

At each time step:

- y_t - true label
- $\hat{y}_t$ - prediction
- $L_t(y_t, \hat{y}_t)$ - some loss function

Loss:

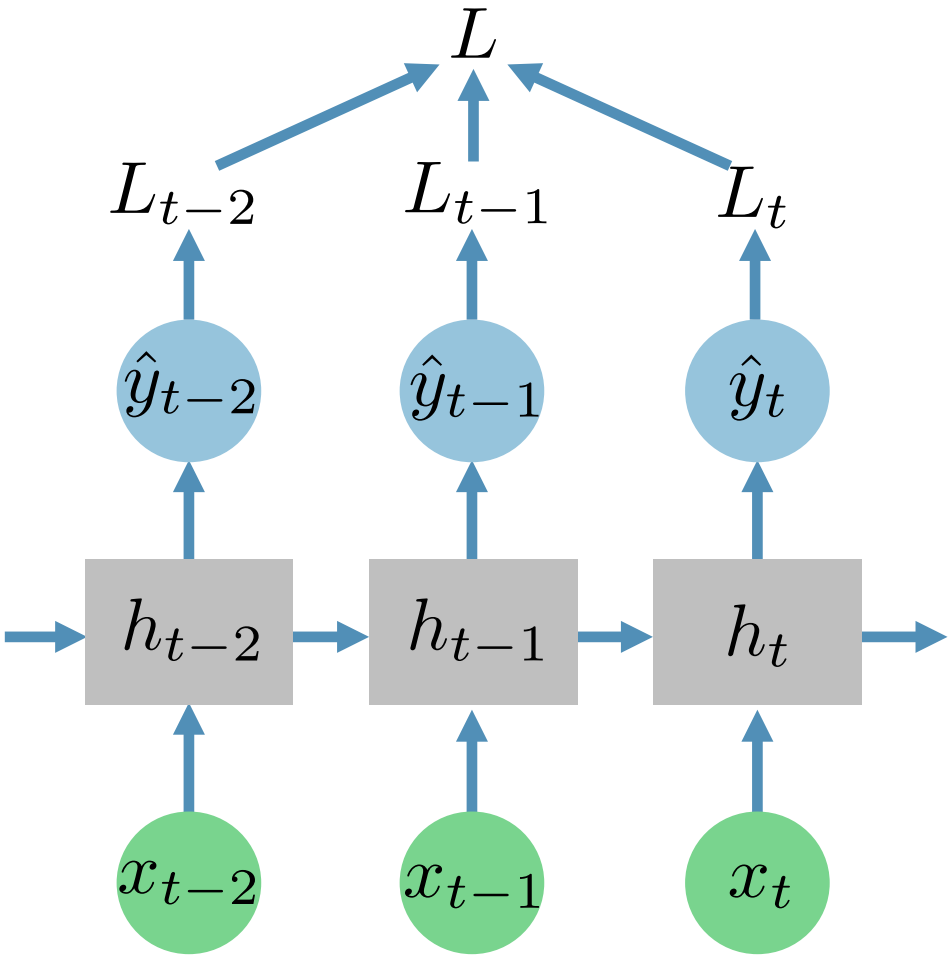
- $L = \sum_i L_i(y_i, \hat{y}_i)$



We can use Backpropagation to train the RNN!

Backpropagation Through Time (BPTT)

As usual we do forward and backward passes.

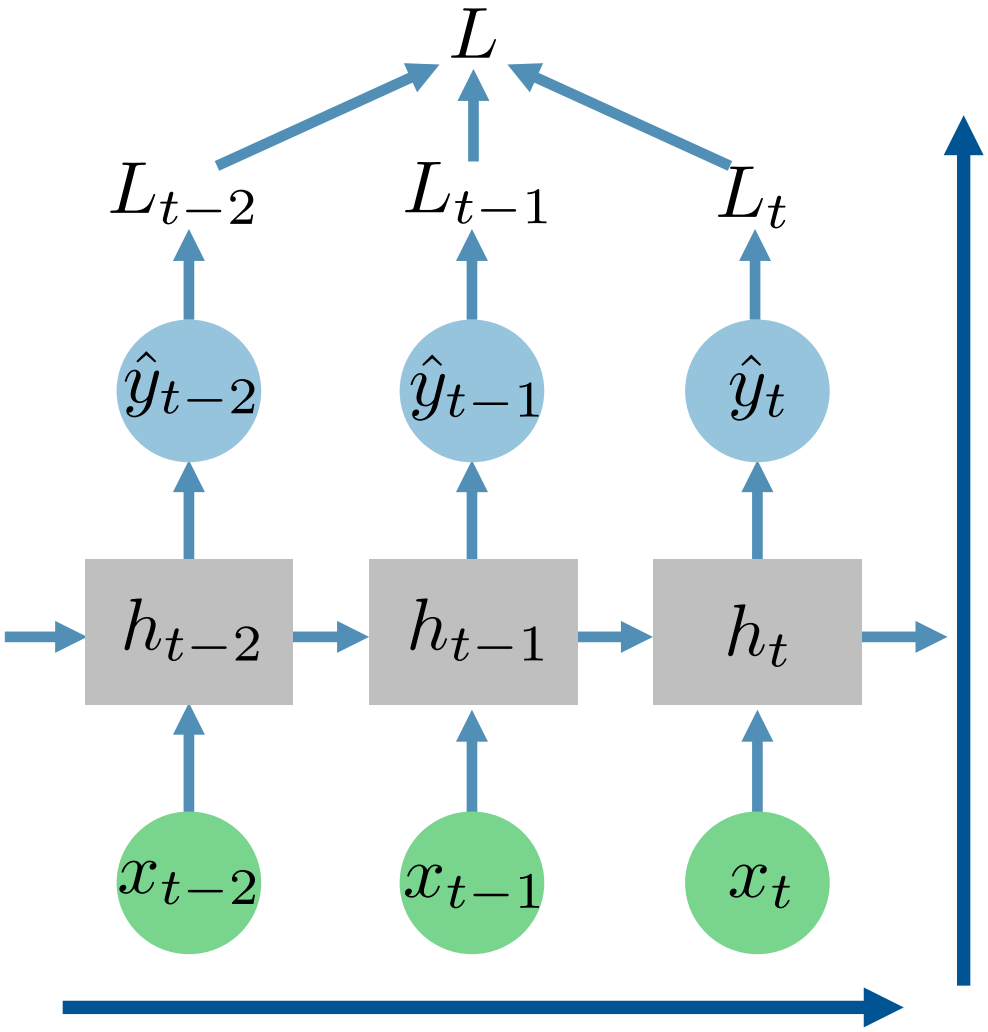


Backpropagation Through Time (BPTT)

As usual we do forward and backward passes.

Forward pass:

$$h_t,\ \hat{y}_t,\ L_t,\ L$$



Backpropagation Through Time (BPTT)

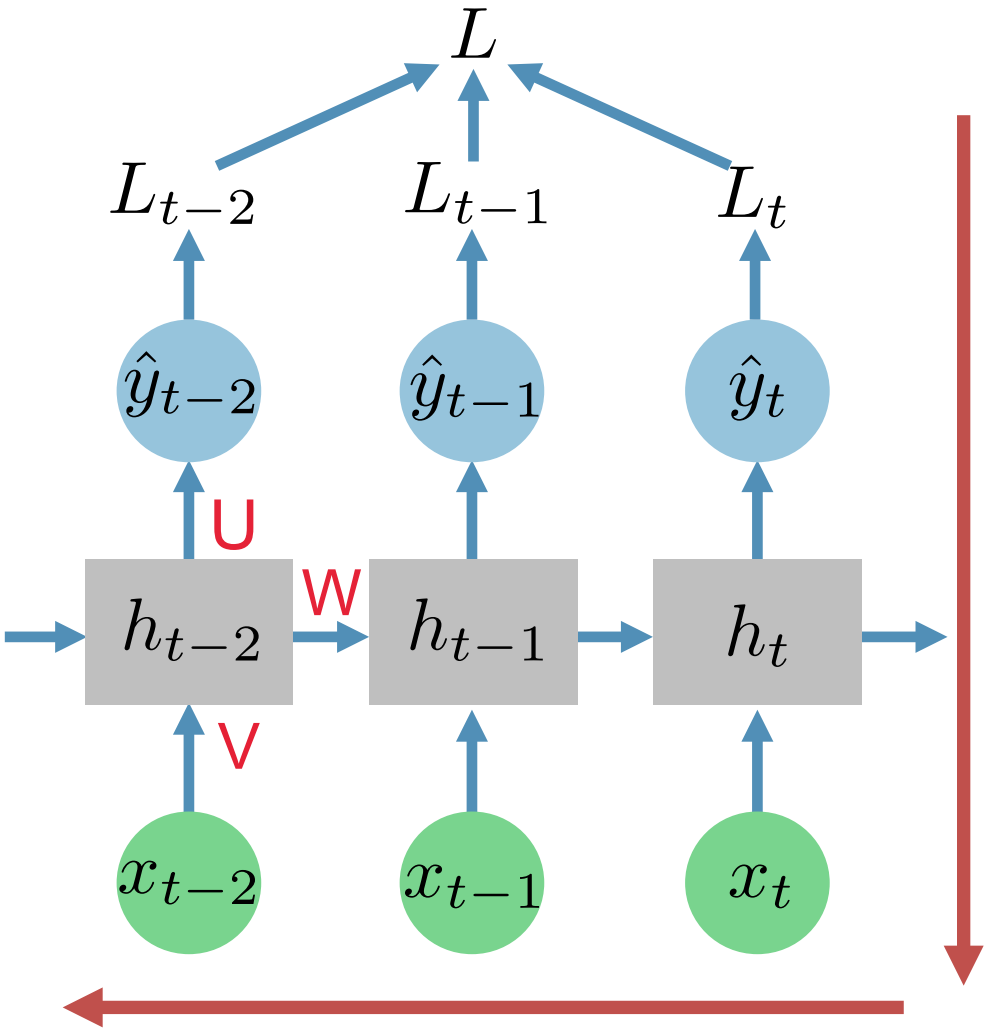
As usual we do forward and backward passes.

Forward pass:

$$h_t,\ \hat{y}_t,\ L_t,\ L$$

Backward pass:

$$\frac{\partial L}{\partial U},\ \frac{\partial L}{\partial V},\ \frac{\partial L}{\partial W},$$
$$\frac{\partial L}{\partial b_x},\ \frac{\partial L}{\partial b_h}$$

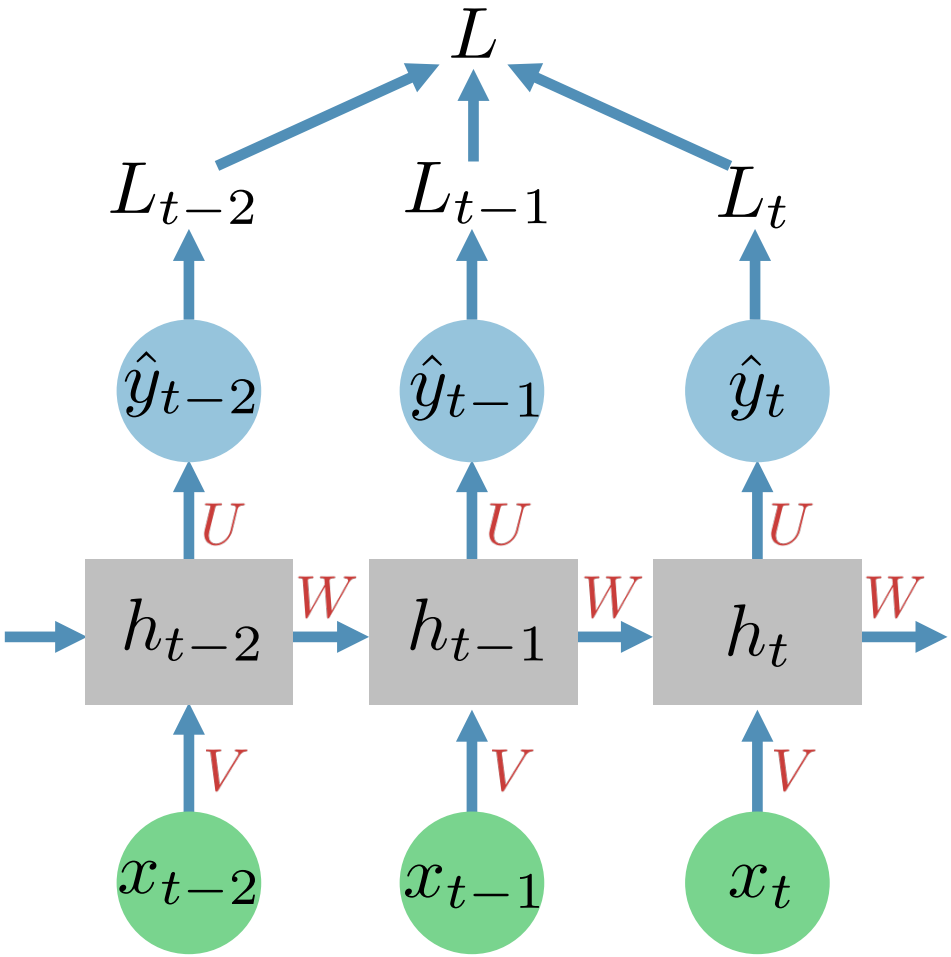


N.B.

We backpropagate through layers and time.

Backpropagation Through Time (BPTT)

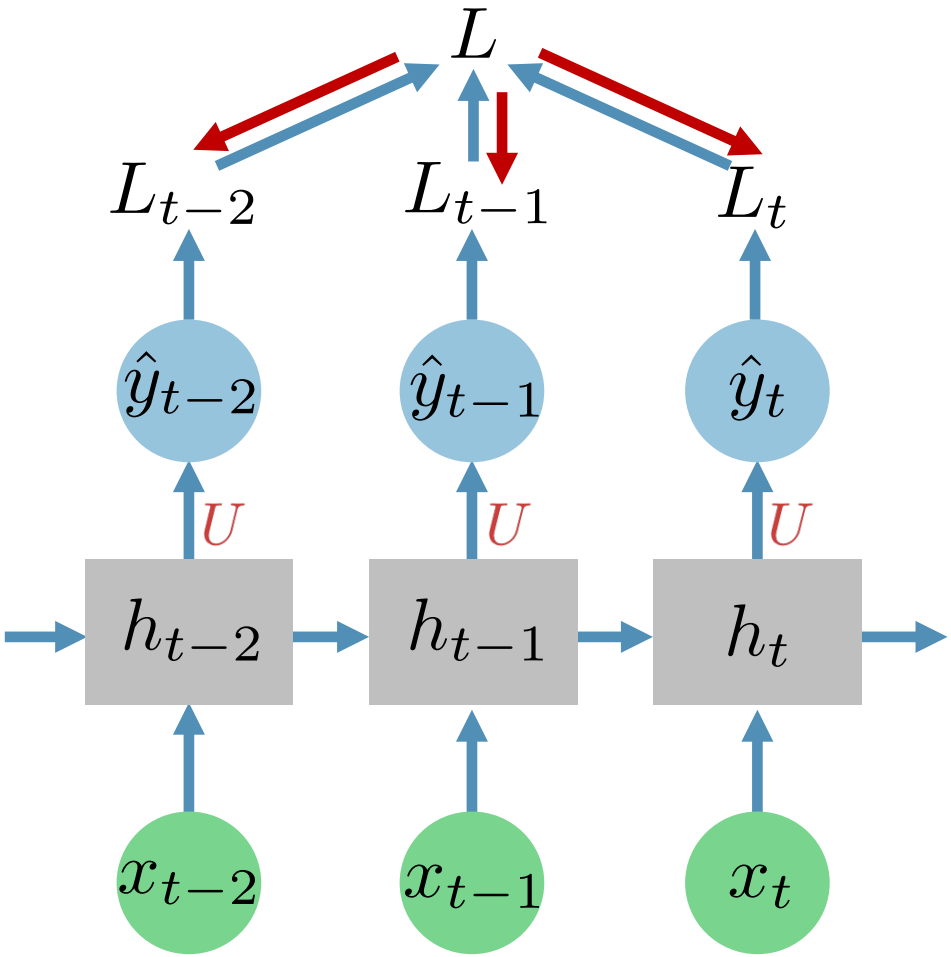
All weights are shared across time steps!



Backpropagation Through Time (BPTT)

All weights are shared across time steps!

$$\frac{\partial L}{\partial U} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial U}$$

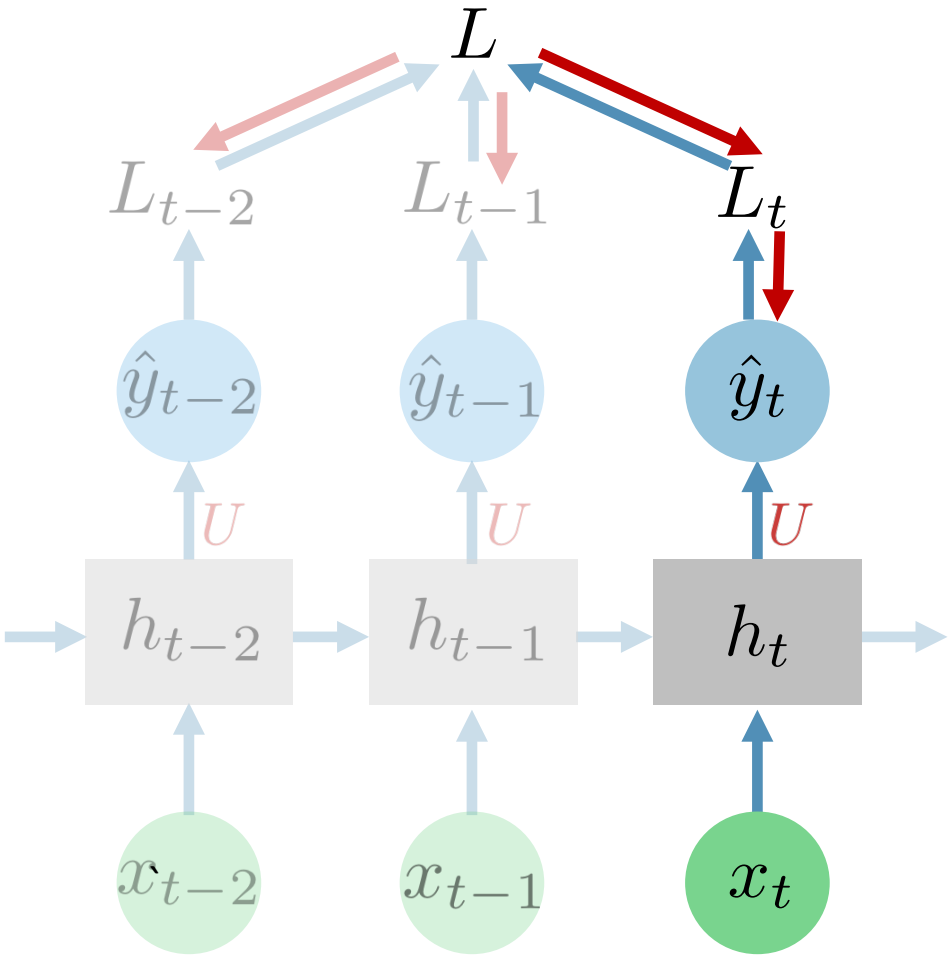


Backpropagation Through Time (BPTT)

All weights are shared across time steps!

$$\frac{\partial L}{\partial U} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial U}$$



Backpropagation Through Time (BPTT)

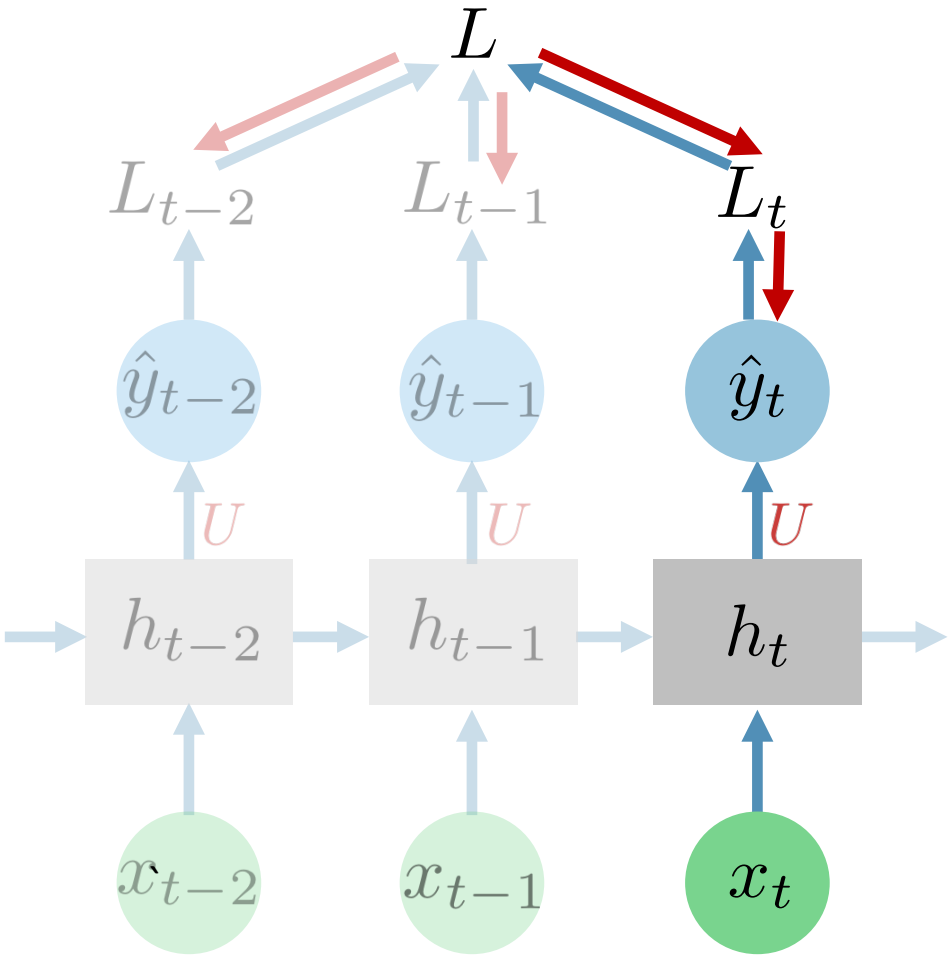
All weights are shared across time steps!

$$\frac{\partial L}{\partial U} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial U}$$

$$\hat{y}_t = f_y(Uh_t + b_y)$$

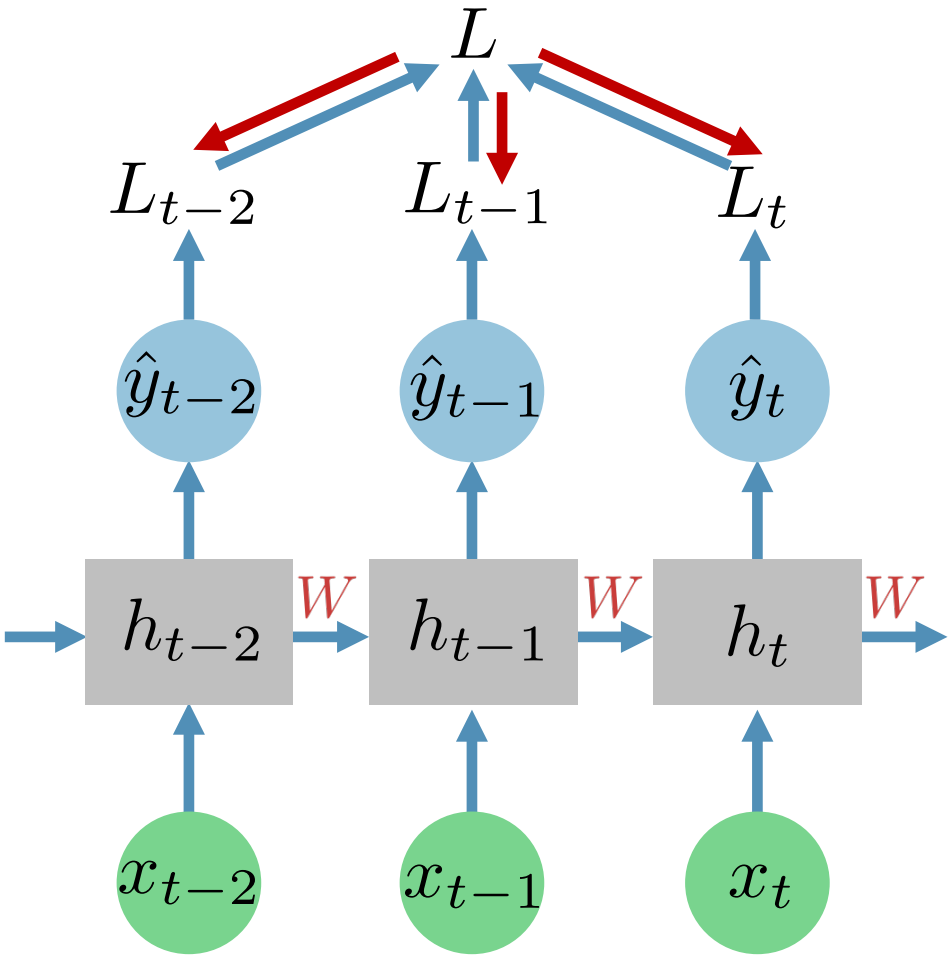
- this is the only dependence



Backpropagation Through Time (BPTT)

All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

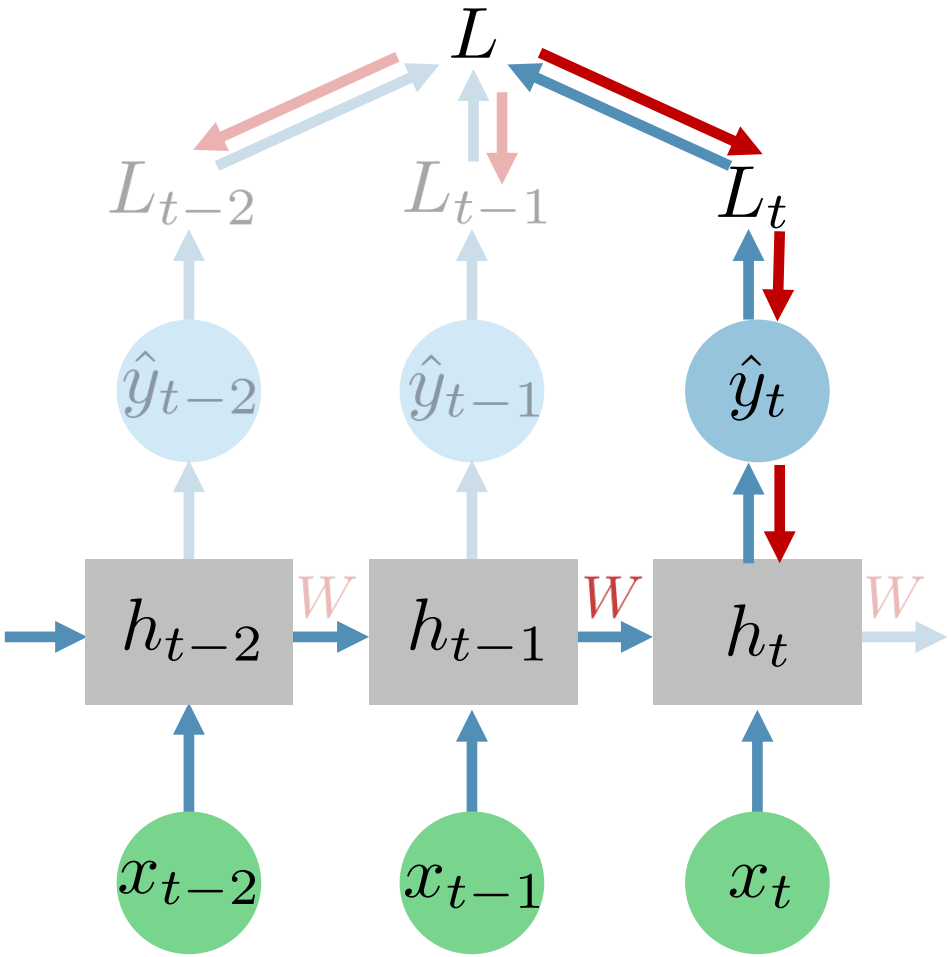


Backpropagation Through Time (BPTT)

All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$



Backpropagation Through Time (BPTT)

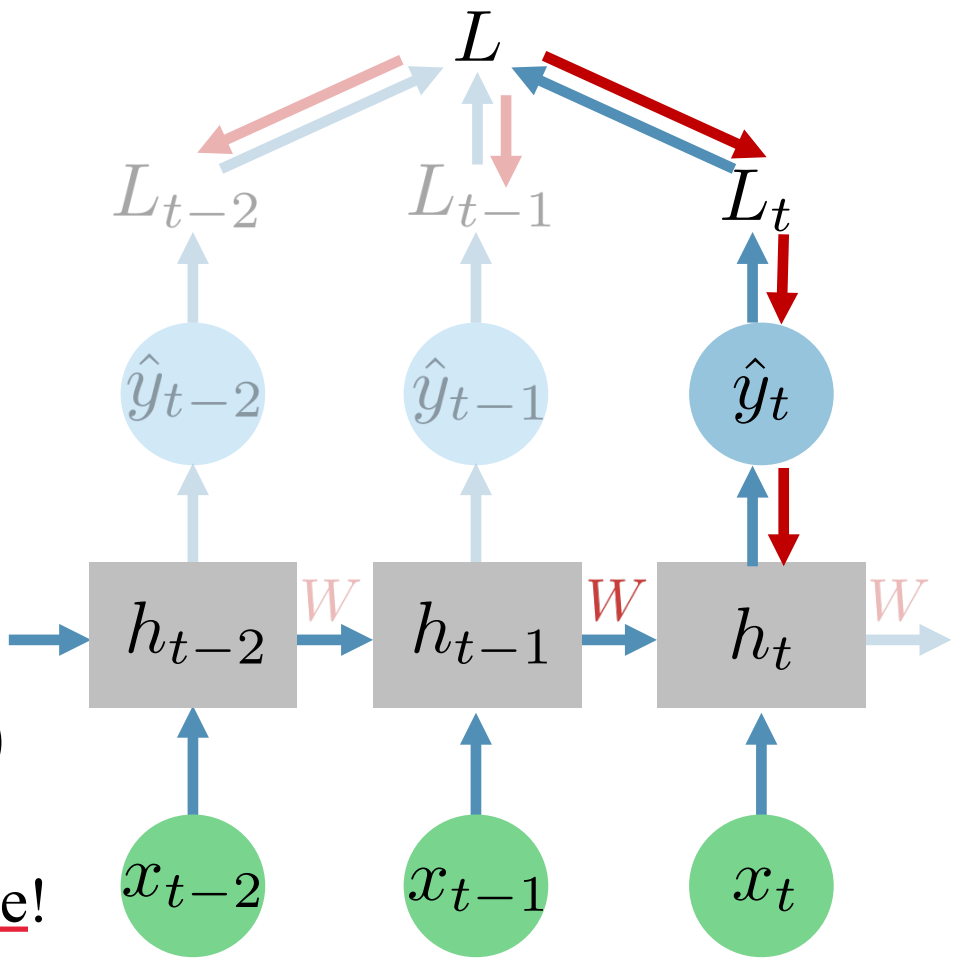
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

This is NOT the only dependence!



Backpropagation Through Time (BPTT)

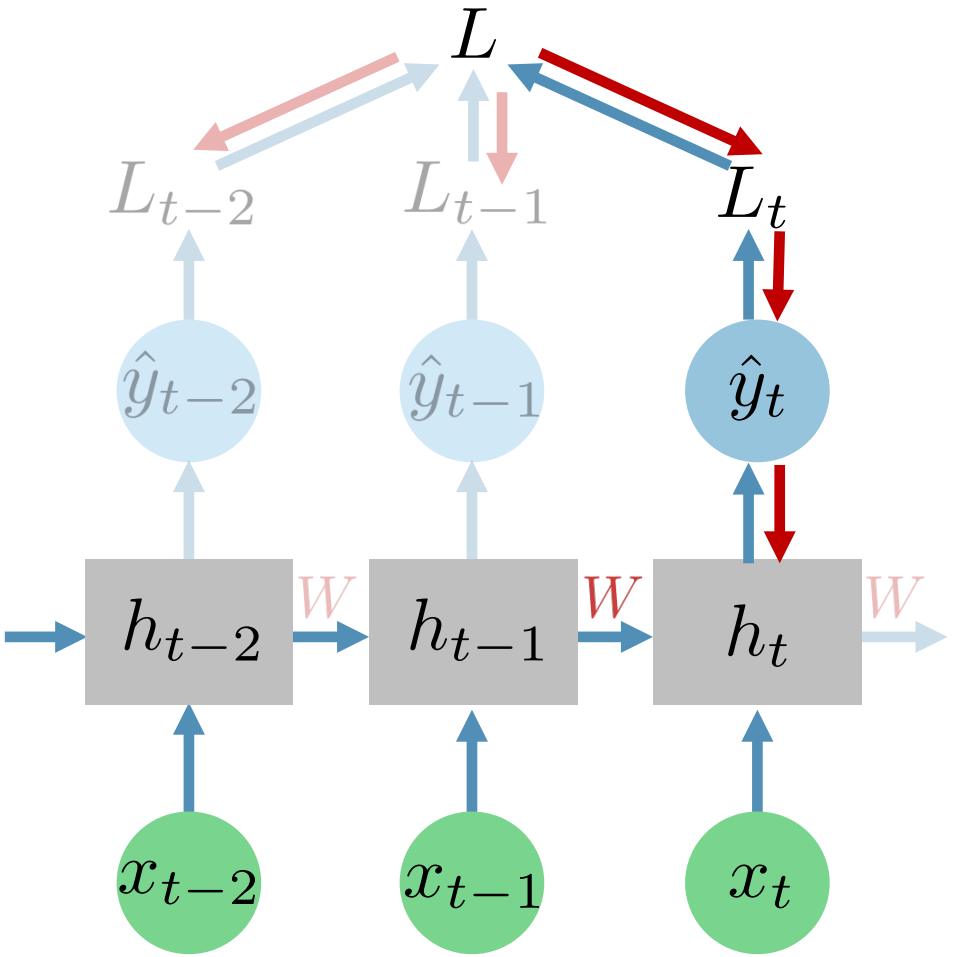
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}$$

$$h_t = f_h(V x_t + W h_{t-1} + b_h)$$

Depends on W too!



Backpropagation Through Time (BPTT)

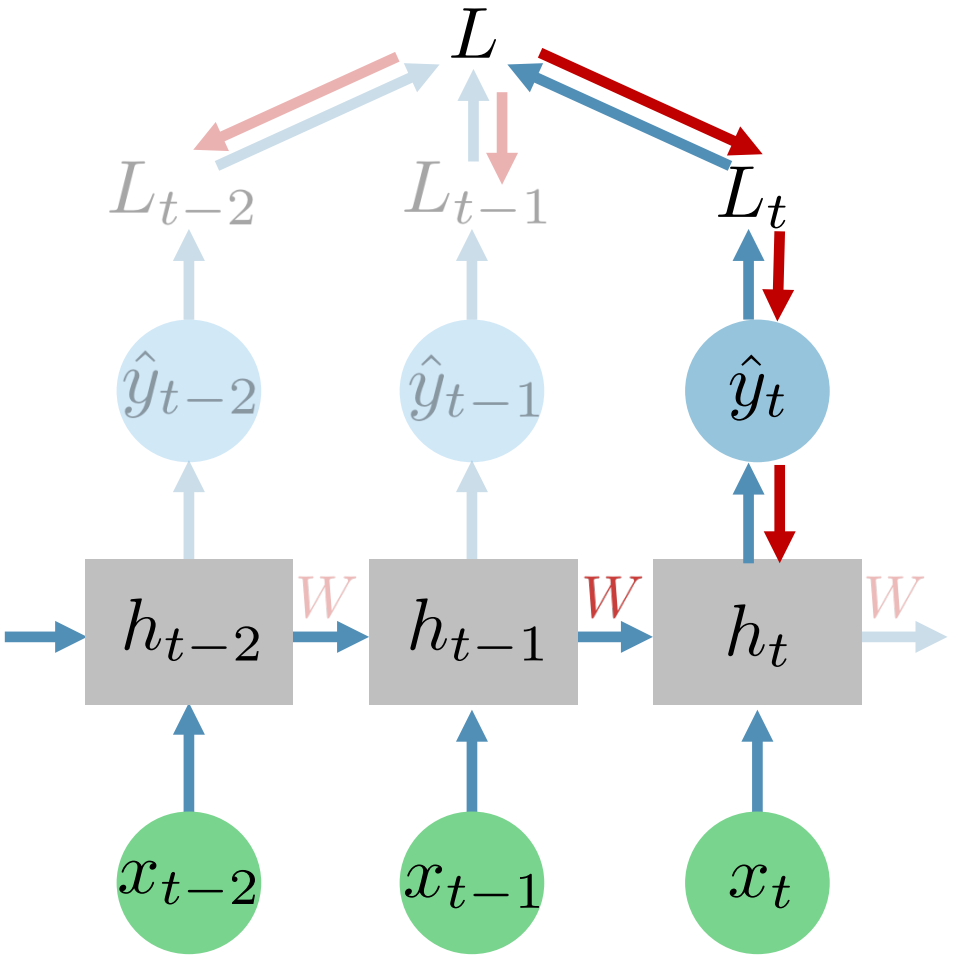
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\cancel{\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}}$$

$$h_t = f_h(V x_t + W h_{t-1} + b_h)$$

Depends on W too!



Backpropagation Through Time (BPTT)

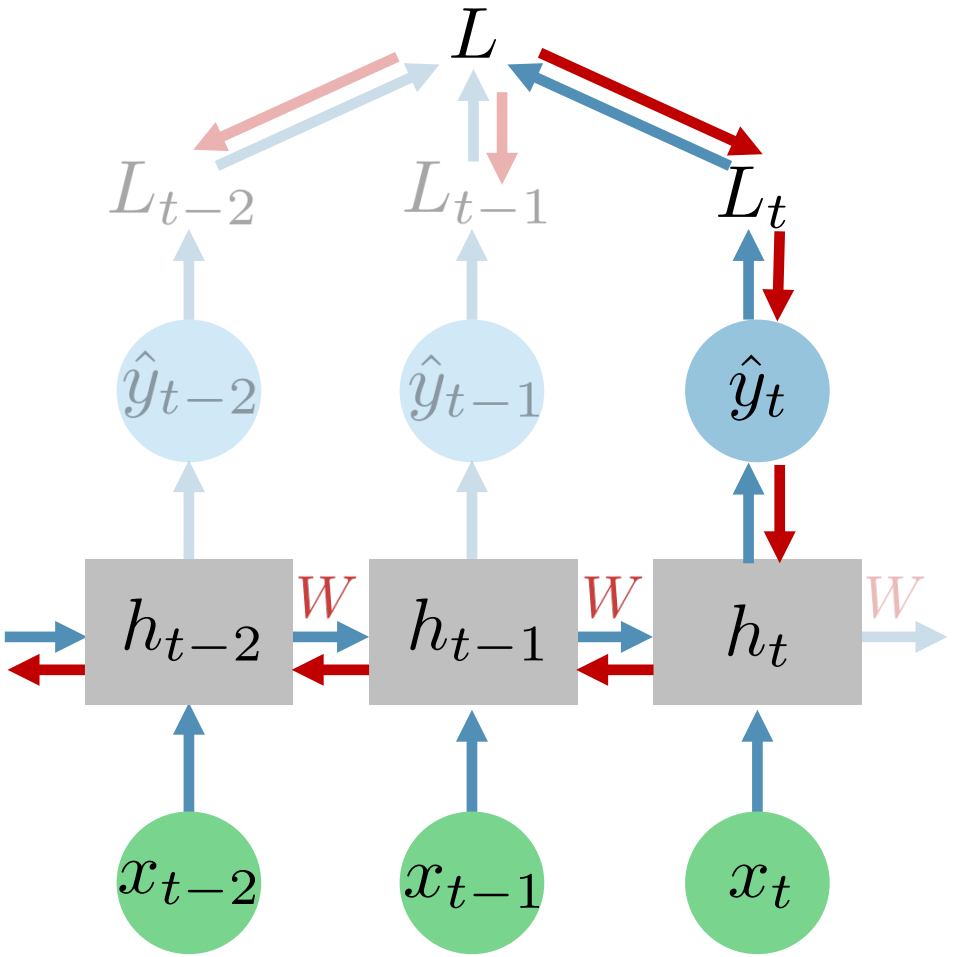
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\cancel{\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}}$$

$$h_t = f_h(V x_t + W h_{t-1} + b_h)$$

Depends on W too!



- $$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \cdots \right)$$

Backpropagation Through Time (BPTT)

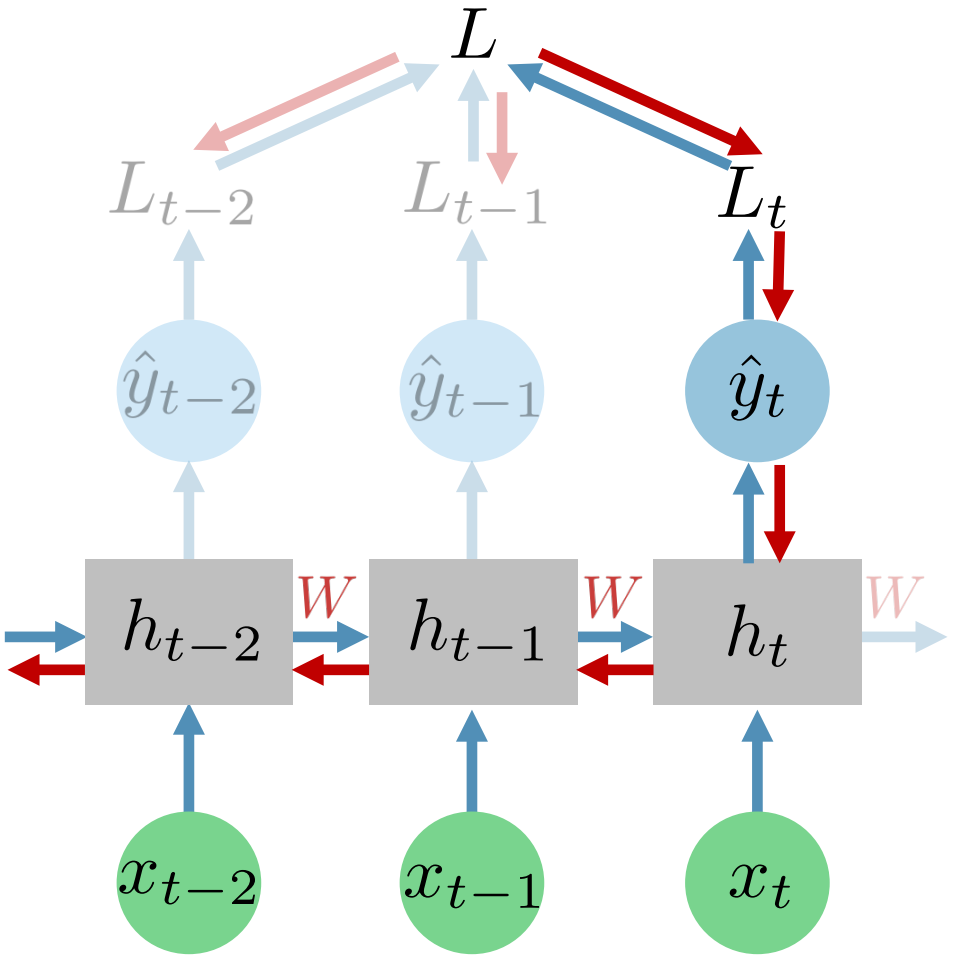
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\cancel{\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}}$$

$$h_t = f_h(V x_t + W h_{t-1} + b_h)$$

Depends on W too!



- $$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \sum_{k=0}^{t} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_{k+1}}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Backpropagation Through Time (BPTT)

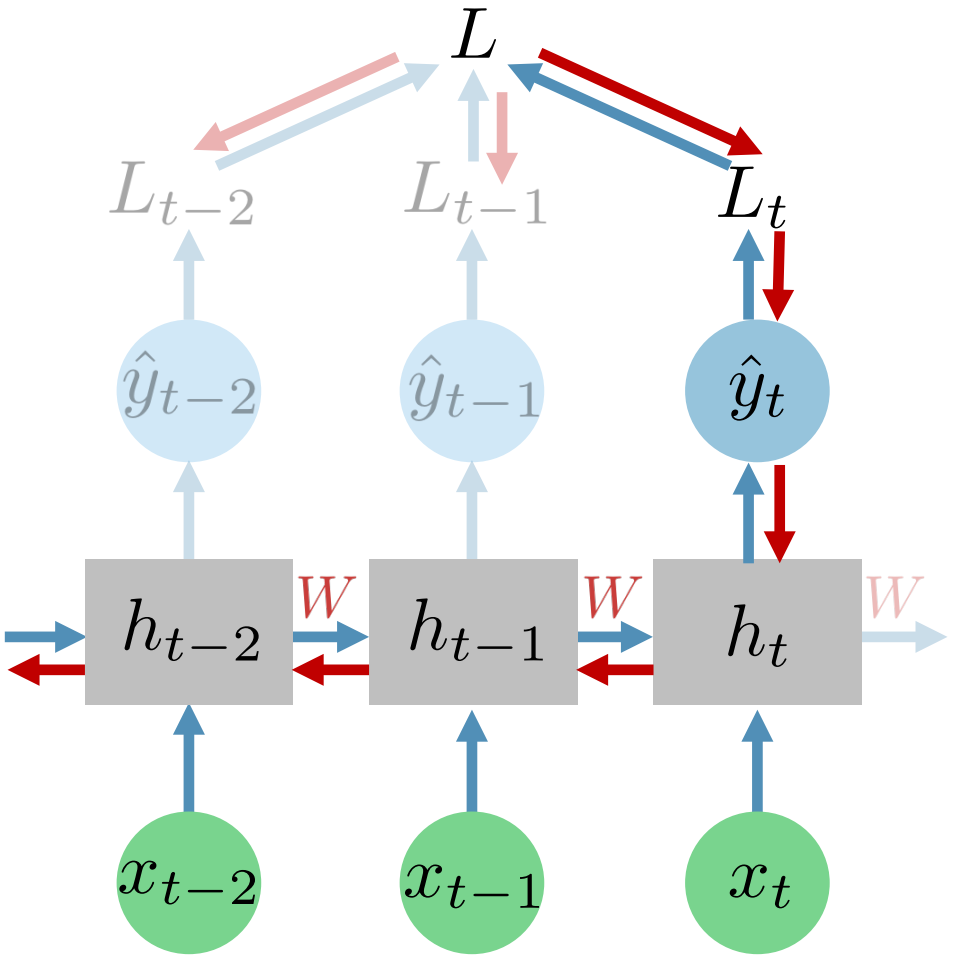
All weights are shared across time steps!

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\cancel{\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W}}$$

$$h_t = f_h(V x_t + W h_{t-1} + b_h)$$

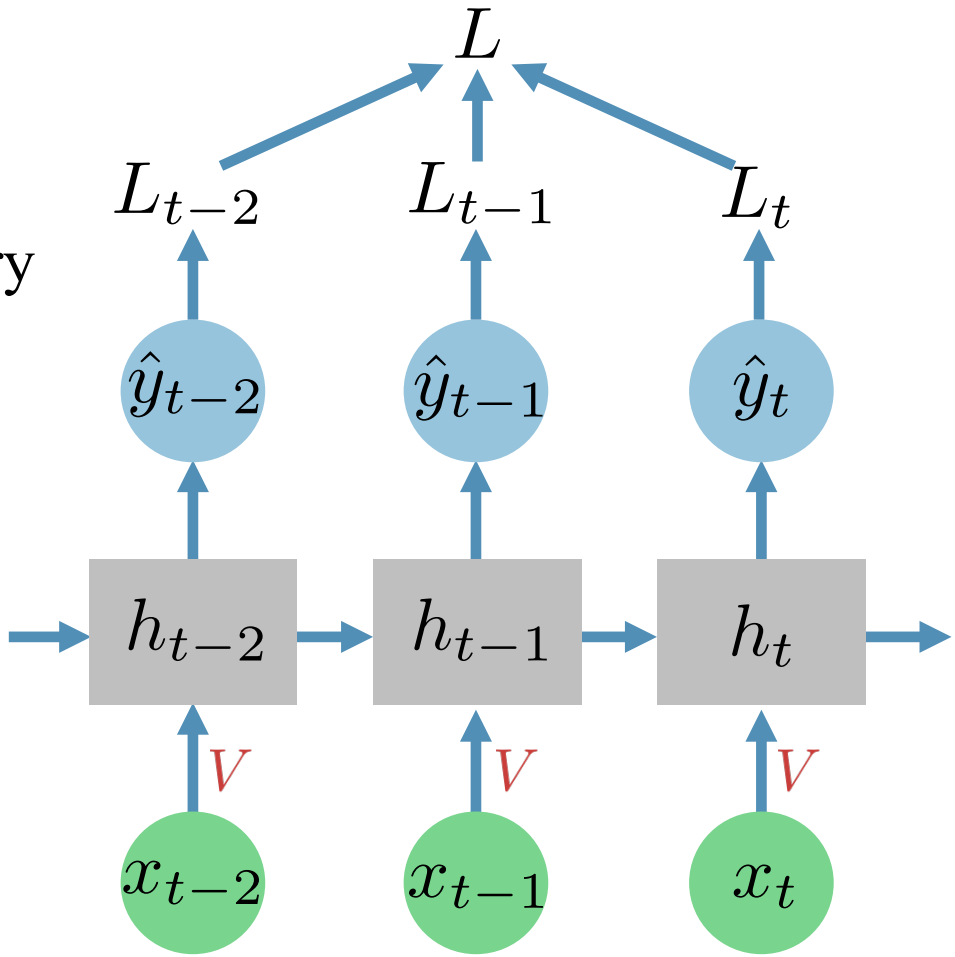
Depends on W too!



- $$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Backpropagation Through Time (BPTT)

And what about the last weight matrix V? Is it necessary to go backwards in time to calculate $\frac{\partial L}{\partial V}$?



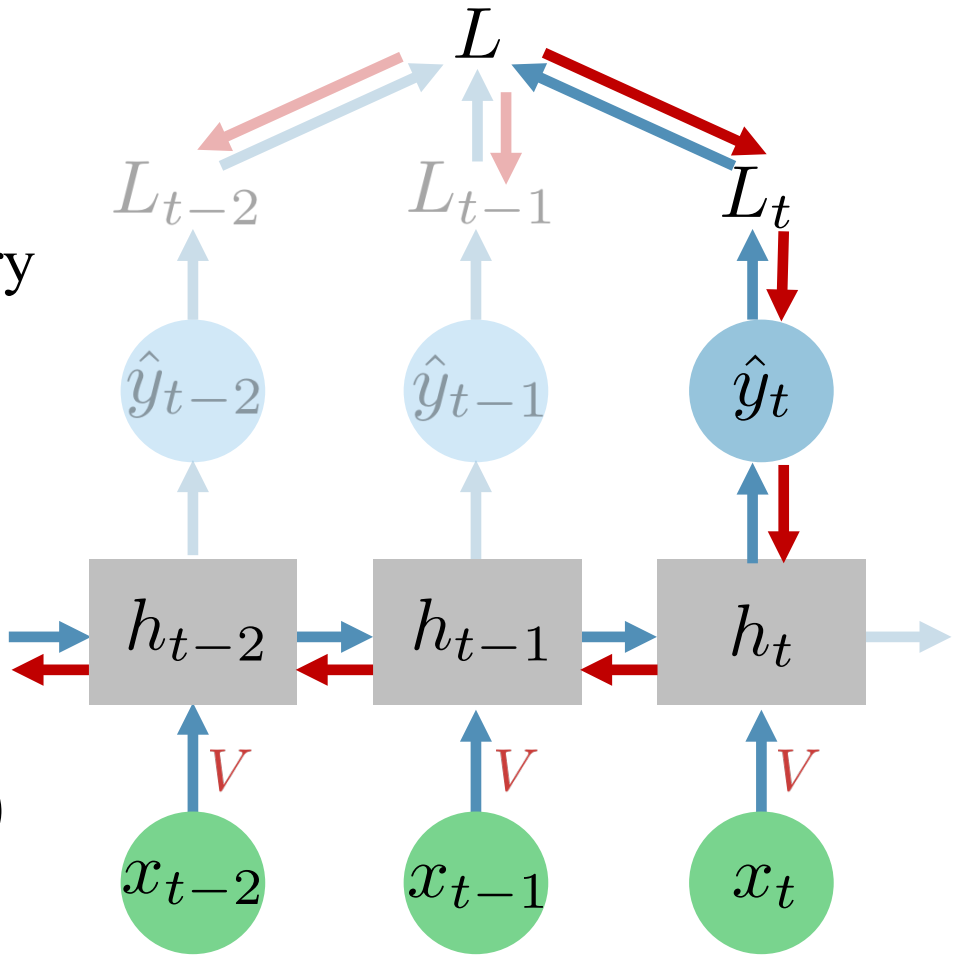
Backpropagation Through Time (BPTT)

And what about the last weight matrix V? Is it necessary to go backwards in time to calculate $\frac{\partial L}{\partial V}$?

Yes! Here we have the same situation as with W:

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

Depends on W too!



Summary

- We have learned what is a simple RNN.
- RNNs are trained using simple Backpropagation. BPTT is just a fancy name for it.

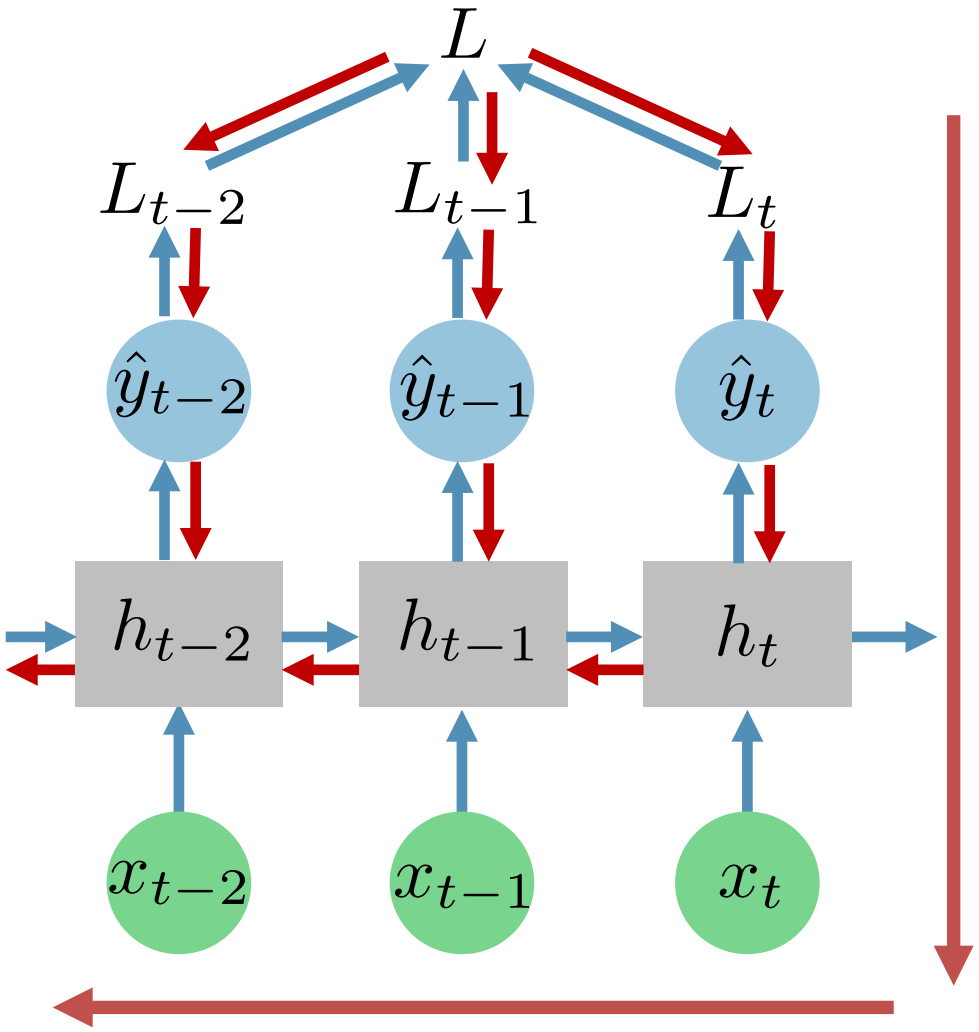
In the next video:

Is it really that simple to train an RNN?

Exploding and vanishing gradients
Problem statement

Previously on this week: BPTT

To train an RNN we need to backpropagate through layers and time



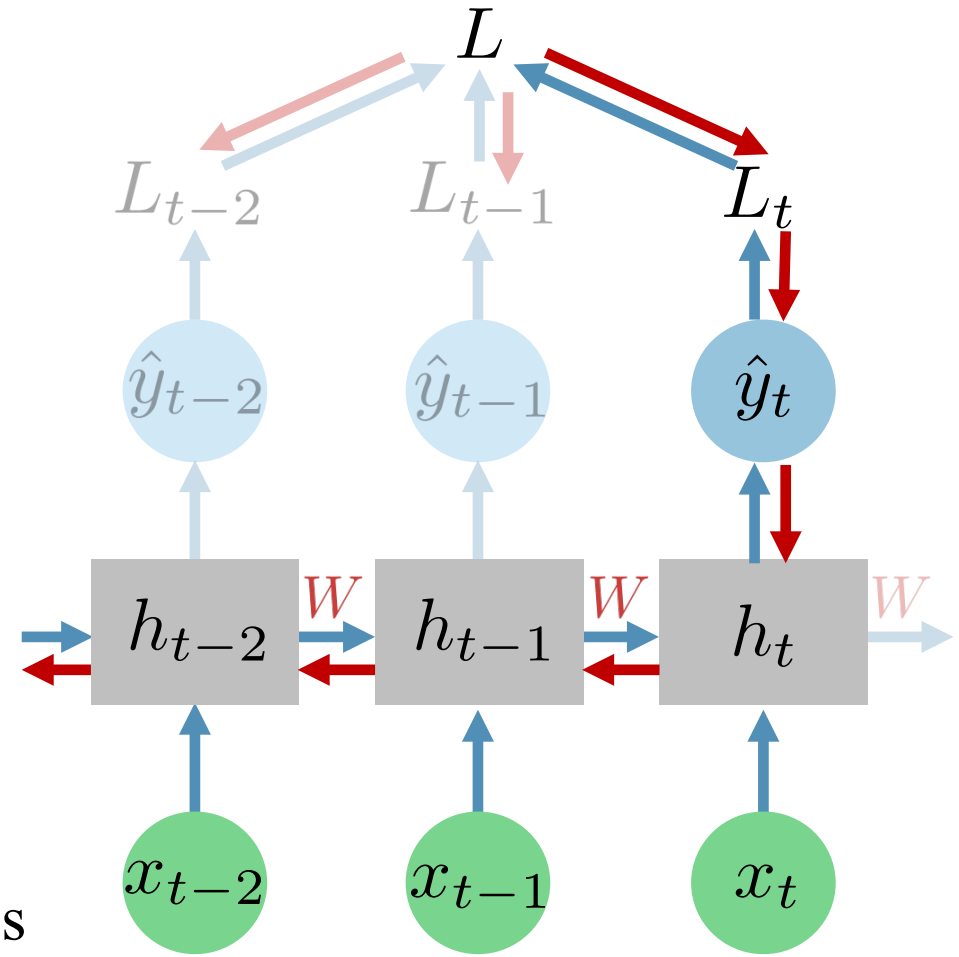
Previously on this week: BPTT

To train an RNN we need to backpropagate through layers and time

$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Contribution of a state at time step ***k*** to the gradient of the loss at time step ***t***



Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

The more steps between the time moments *k* and *t*, the more elements are in this product

In conclusion,

Values of these Jacobian matrices have particularly severe impact on the contributions from faraway steps

Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Let's suppose for a moment that h_i is a scalar and consequently $\frac{\partial h_i}{\partial h_{i-1}}$ is also a scalar

Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Assuming,

Let's suppose for a moment that h_i is a scalar and consequently $\frac{\partial h_i}{\partial h_{i-1}}$ is also a scalar

$\left| \frac{\partial h_i}{\partial h_{i-1}} \right| < 1$ → The product goes to 0 exponentially fast

$\left| \frac{\partial h_i}{\partial h_{i-1}} \right| > 1$ → The product goes to infinity exponentially fast

Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Let's suppose for a moment that h_i is a scalar and consequently $\frac{\partial h_i}{\partial h_{i-1}}$ is also a scalar

$$\left| \frac{\partial h_i}{\partial h_{i-1}} \right| < 1$$

Vanishing gradients

- contributions from faraway steps vanish and don't affect the training
- difficult to learn long-range dependencies

Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Let's suppose for a moment that h_i is a scalar and consequently $\frac{\partial h_i}{\partial h_{i-1}}$ is also a scalar

$$\left| \frac{\partial h_i}{\partial h_{i-1}} \right| > 1$$

Exploding gradients

- make the learning process unstable
- gradient could even become a NaN

Let's look at the gradient

$$\frac{\partial L_t}{\partial W} \propto \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$

Back to matrices

The same is true for matrices but with the spectral matrix norm instead of the absolute value:

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$$ → The product goes to zero-norm matrix exponentially fast

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$$ → The product goes to a matrix of infinite norm exponentially fast

Is it really a problem in practice?

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) = f_h(pr_t)$$

Is it really a problem in practice?

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) = f_h(pr_t)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = diag(f_h'(pr_t)) \cdot \mathbf{?}$$

Is it really a problem in practice?

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) = f_h(pr_t)$$

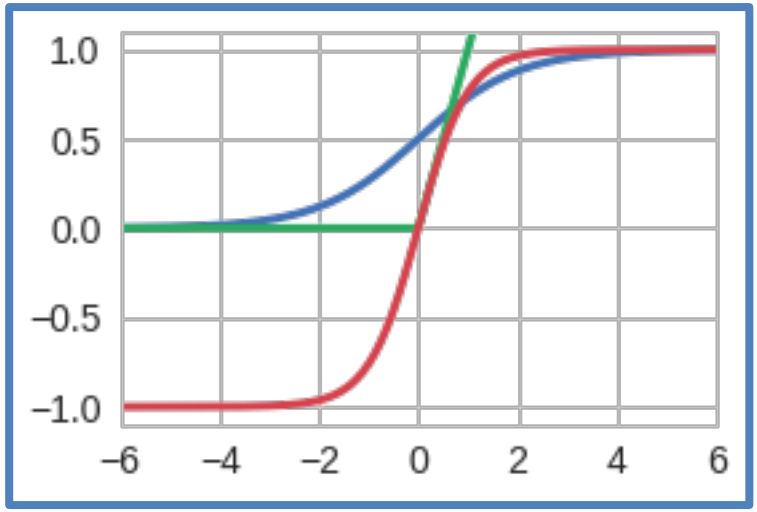
- $$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = diag(f_h'(pr_t)) \cdot W$$

Is it really a problem in practice?

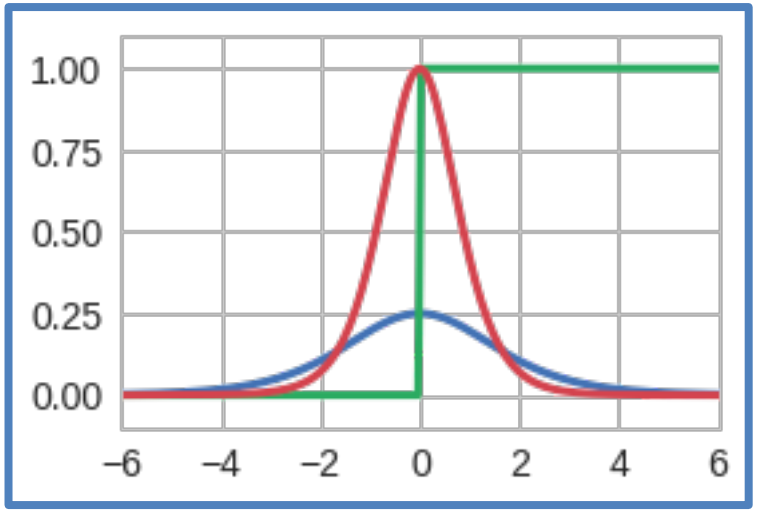
$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) = f_h(pr_t)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = \boxed{diag(f_h'(pr_t))} \cdot W$$

sigmoid, tanh, ReLU



Derivatives



Vanishing gradients are very likely especially with sigmoid and tanh

Is it really a problem in practice?

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) = f_h(pr_t)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = diag(f_h'(pr_t)) \cdot W$$

$||W||$ may be either **small** **or** **large**

Small $||W||$ could aggravate the vanishing gradient problem

Large $||W||$ could cause exploding gradients (especially with ReLU)

Summary

- In practice vanishing and exploding gradients are common for RNNs. These problems also occur in deep Feedforward NNs.
- Vanishing gradients make the learning of long-range dependencies very difficult.
- Exploding gradients make the learning process very unstable and may even crash it.

In the next video:

How to deal with these issues?

Exploding and vanishing gradients Solutions

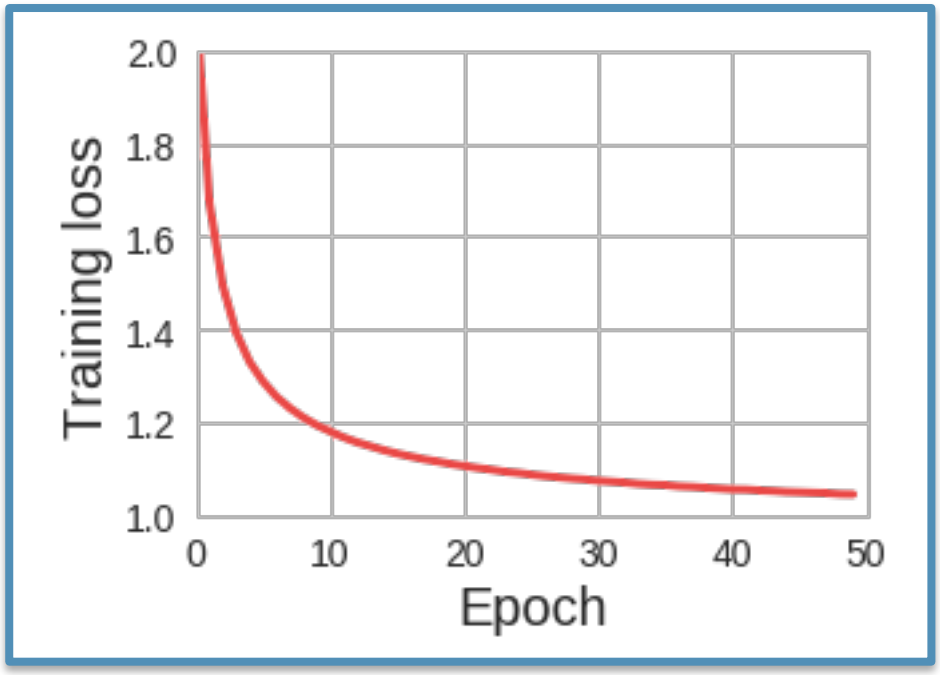
Previously on this week: exploding gradients

- Gradient norms may become very large or even NaNs in the worst case
- Exploding gradients make the learning process unstable

Exploding gradients: detection

Exploding gradients are easy to detect

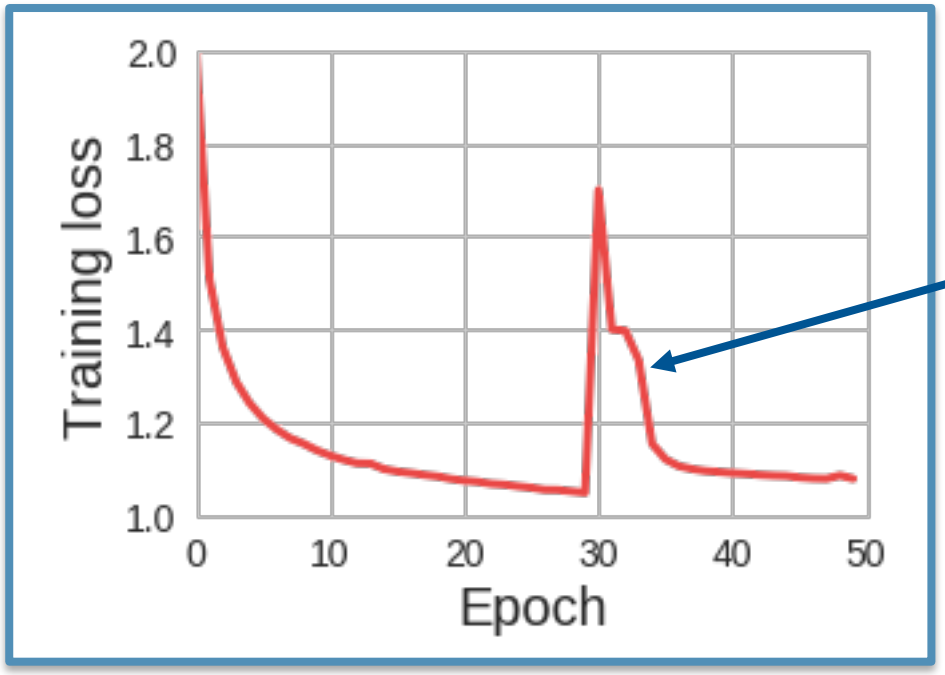
Stable learning curve



Exploding gradients: detection

Exploding gradients are easy to detect

Unstable learning curve



This is it!

If the gradients contain NaNs you end up with NaNs in the weights

Gradient clipping

Exploding gradients are easy to detect

Gradient $g = \frac{\partial L}{\partial \theta}$,θ - all the network parameters

If $||g|| >$ threshold:

$$g \leftarrow \frac{threshold}{\|g\|} g$$

Simple but still very effective!

Gradient clipping

Exploding gradients are easy to detect

Gradient $g = \frac{\partial L}{\partial \theta}$,θ - all the network parameters

If $||g|| >$ threshold:

$$g \leftarrow \frac{threshold}{\|g\|} g$$

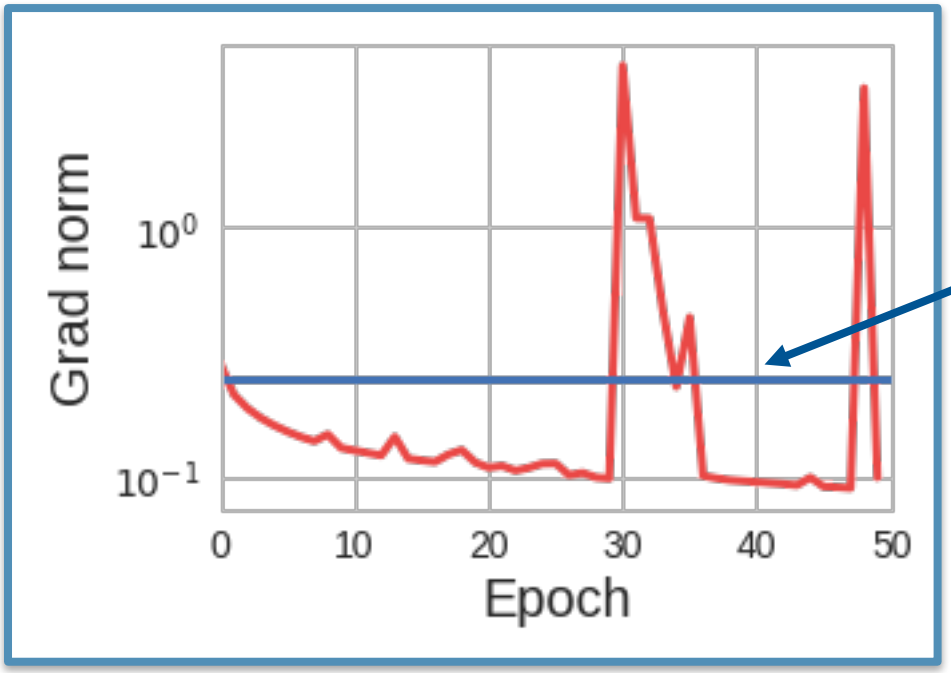
Simple but still very effective!

It is enough to clip $\frac{\partial h_t}{\partial h_{t-1}}$

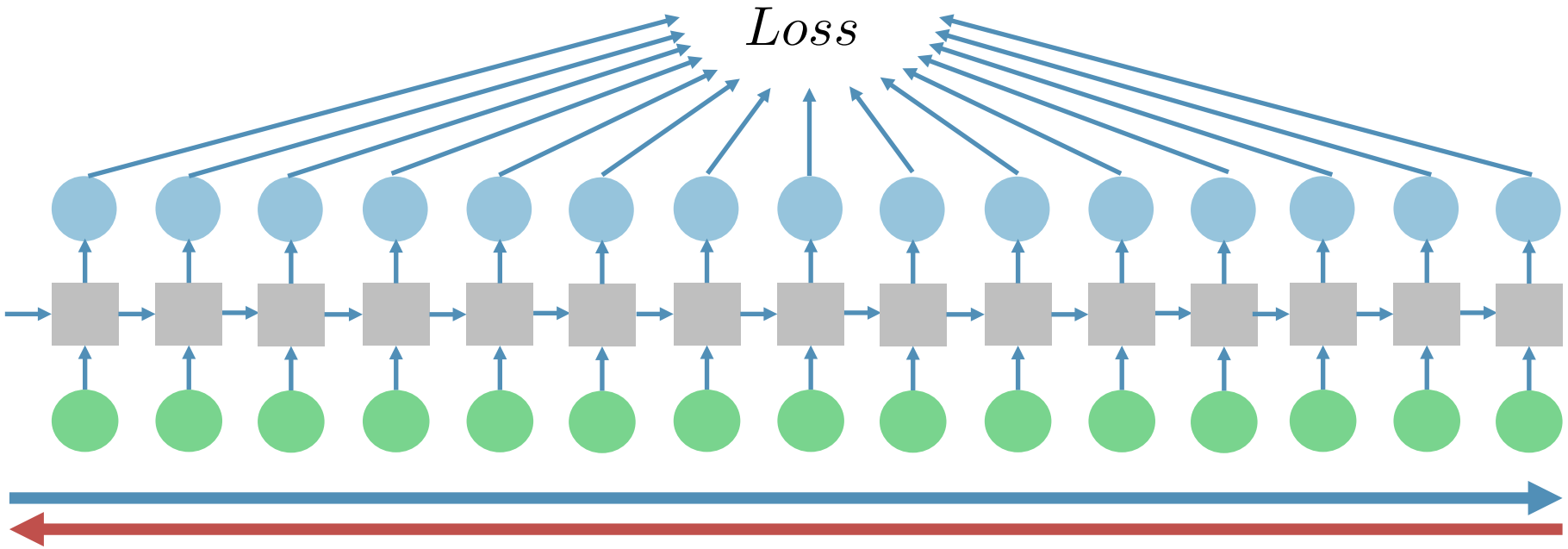
Gradient clipping: threshold

Choose the highest threshold which helps to overcome the exploding gradient problem

Curve of the gradient norm



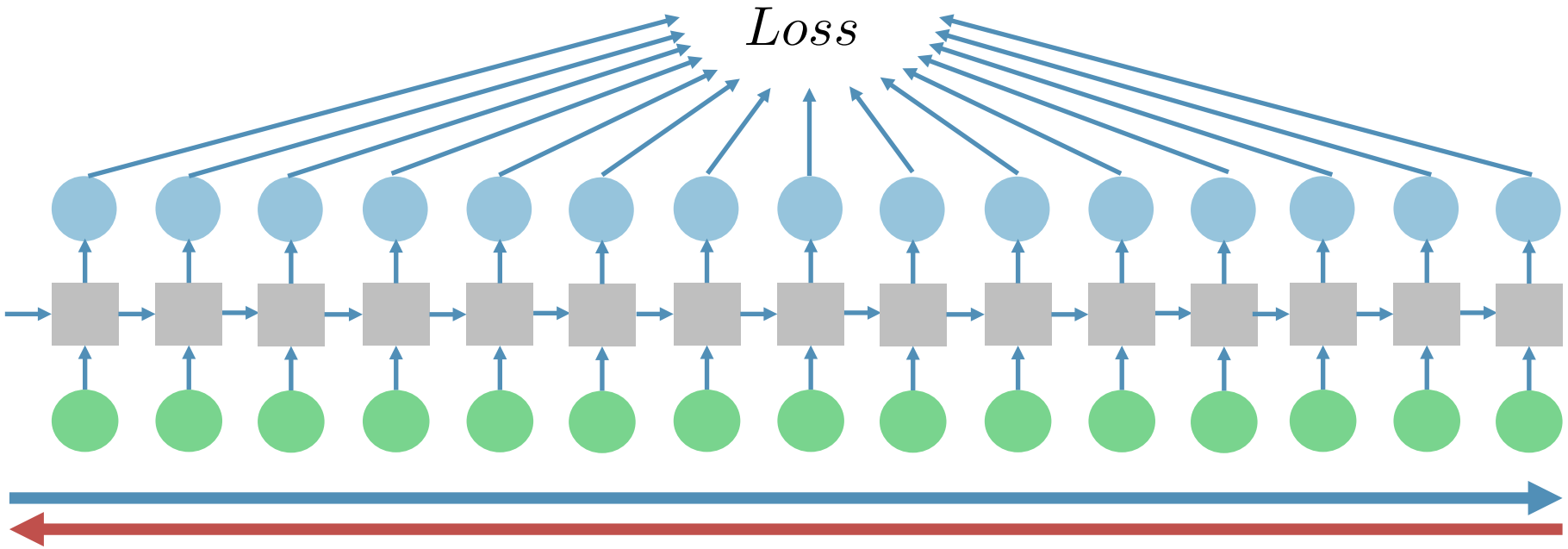
Previously on this week: BPTT



Forward pass through the entire sequence to compute the loss

Backward pass through the entire sequence to compute the gradient

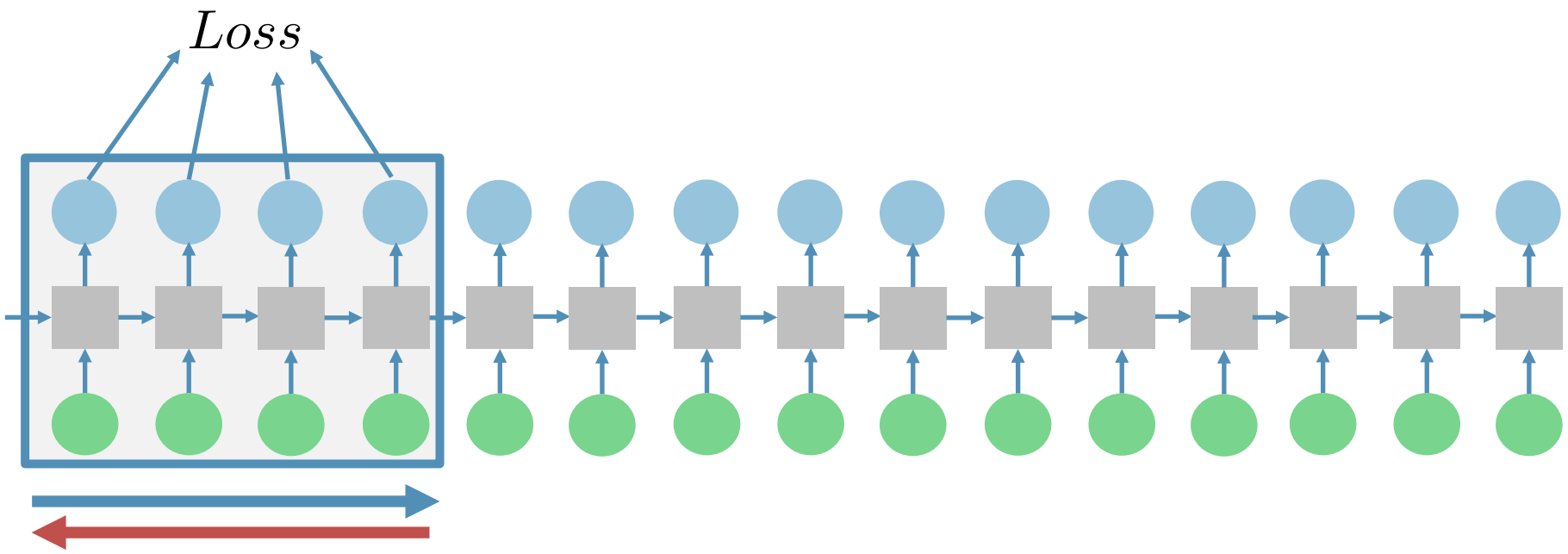
Previously on this week: BPTT



And what if we have very long training sequences?

Way too expensive + exploding gradients

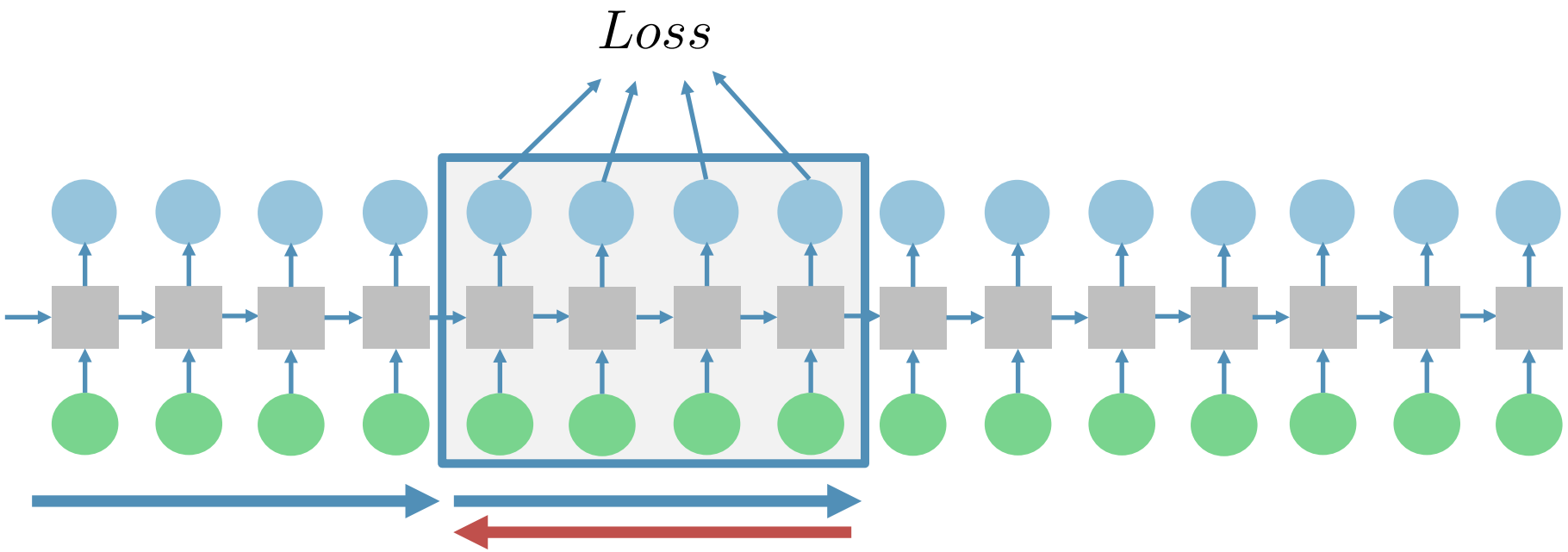
Truncated BPTT



Let's run forward and backward passes through the chunks of the sequence instead of the whole sequence.

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.

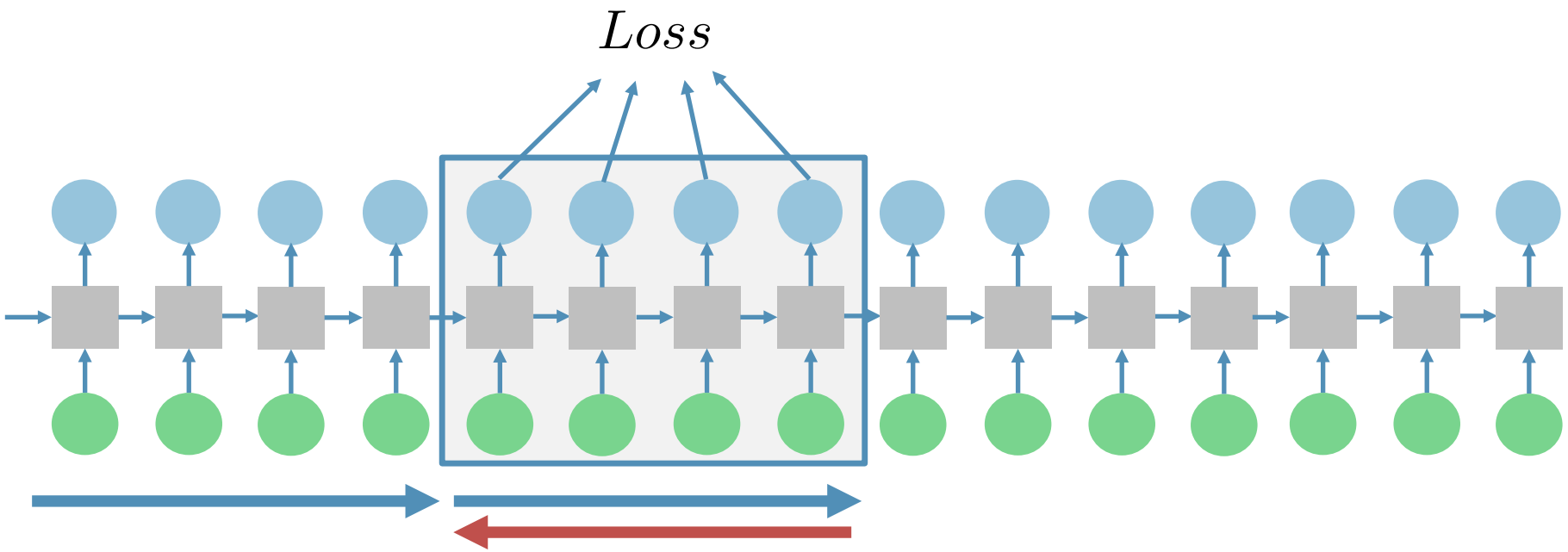
Truncated BPTT



Let's run forward and backward passes through the chunks of the sequence instead of the whole sequence.

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.

Truncated BPTT



Truncated BPTT is much faster but it doesn't come without a price! Dependencies longer than the chunk size don't affect the training but at least they still work at forward pass.

long range dependencies can not be ranked properly

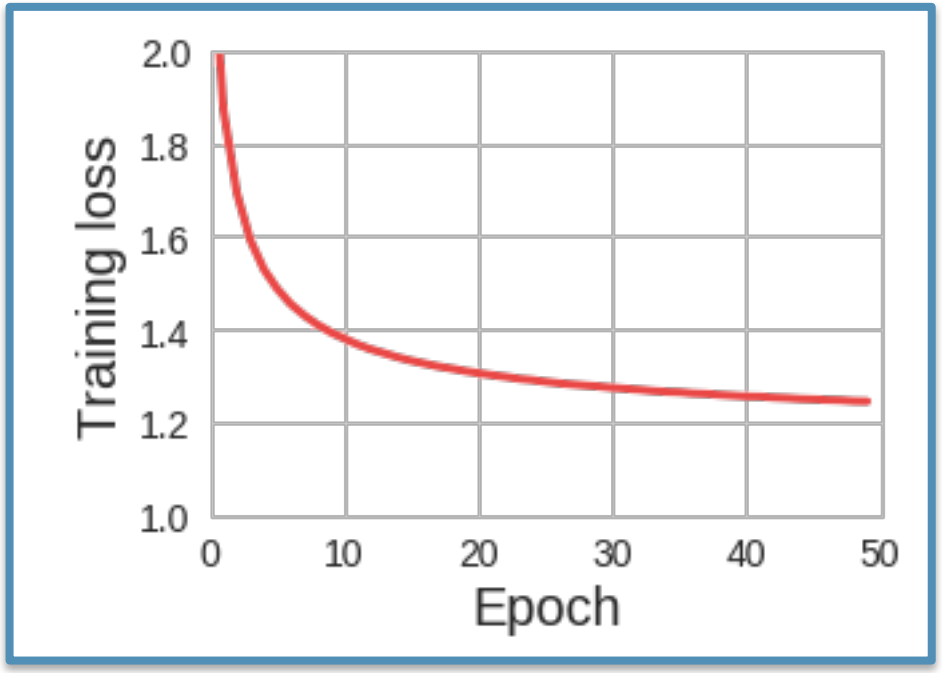
Previously on this week: vanishing gradients

- Contributions from faraway steps vanish and don't affect the training
- It is difficult to learn long-range dependencies

Vanishing gradients: detection

It is not clear how to detect vanishing gradients

Learning curve



Does the gradient vanish or the task is difficult?

Vanishing gradients: detection

It is not clear how to detect vanishing gradients

in case of...

Gradient norm

$$\left\| \frac{\partial L_t}{\partial h_{t-100}} \right\|_2$$ is small

Does the gradient vanish or there are no long-range dependencies in the data?

Vanishing gradients: how to deal with them?

- LSTM, GRU
- ReLU activation function
- Initialization of the recurrent weight matrix
- Skip connections
- …

ReLU activation function

obviously the jacobian matrix depends on the choice of the activation function & weights W

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} \overset{1}{=} diag(f_h'(pr_t)) \cdot W$$

Let's use the ReLU which is much more resistant to the vanishing gradient problem.

Initialization of the recurrent weight matrix

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = diag(f_h'(pr_t)) \cdot \boxed{W}^{2}$$

Q is orthogonal if $Q^T = Q^{-1}$ =>

$\prod_i Q_i$ doesn't explode or vanish

Initialization of the recurrent weight matrix

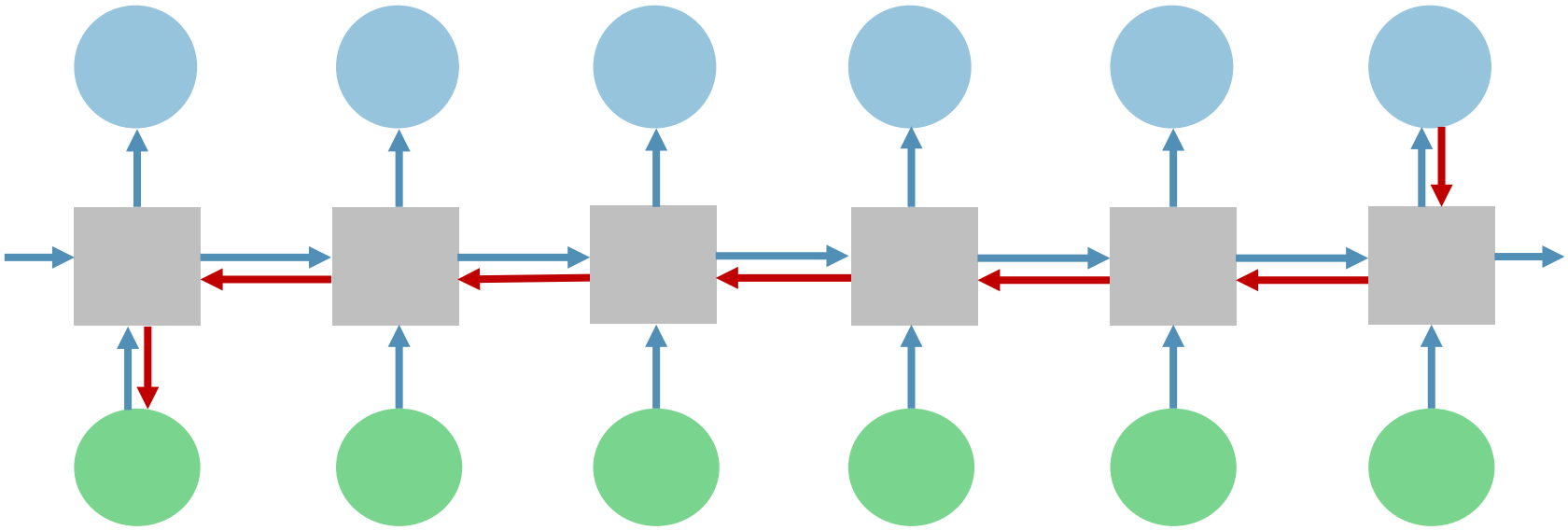
$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial h_t}{\partial pr_t}\frac{\partial pr_t}{\partial h_{t-1}} = diag(f_h'(pr_t)) \cdot W$$

Q is orthogonal if $Q^T = Q^{-1}$ =>

$\prod_i Q_i$ doesn't explode or vanish

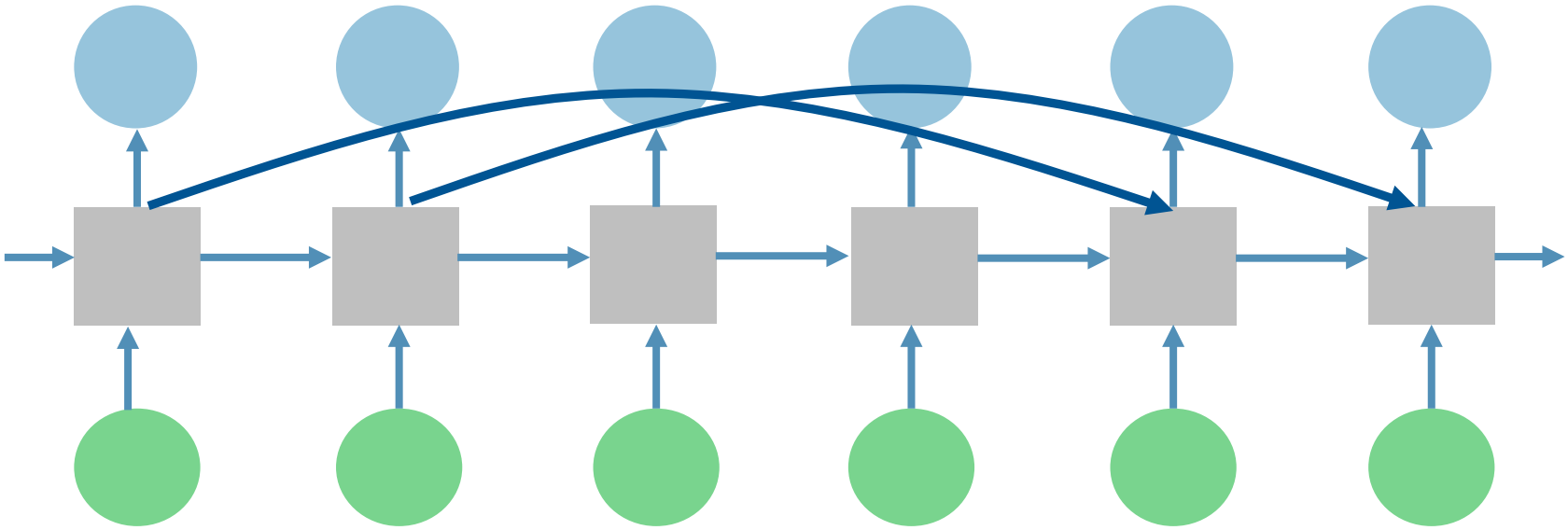
- Initialise W with an orthogonal matrix
- Use orthogonal W through the whole training

Skip connections



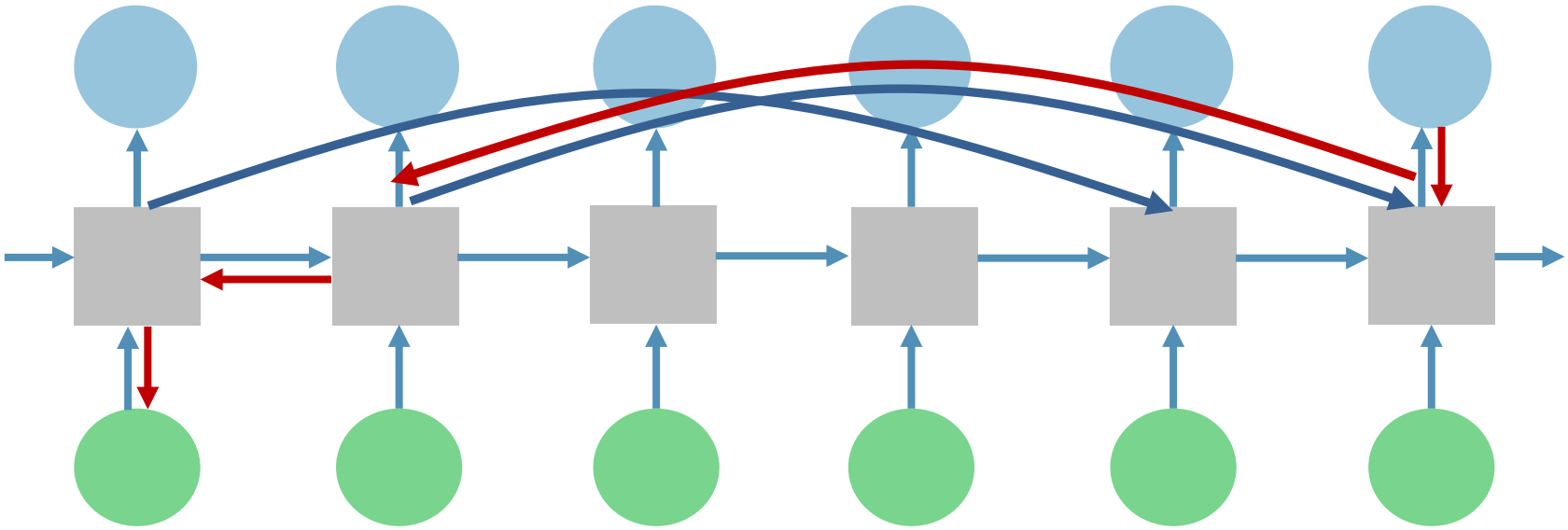
Very long ways for the gradients => vanishing gradients

Skip connections



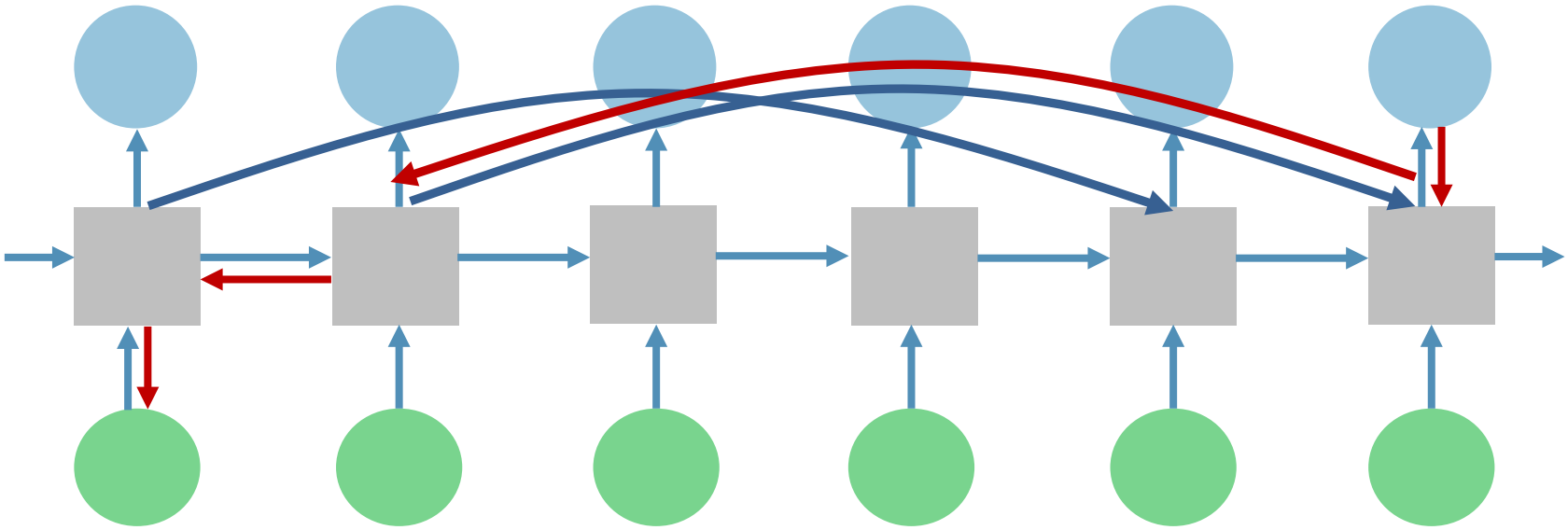
Let's add shortcuts!

Skip connections



Add shortcuts => shorter ways for the gradients =>
learn longer dependencies

Skip connections



Add shortcuts => shorter ways for the gradients => learn longer dependencies

The idea is similar to the residual connections in the ResNet

Summary

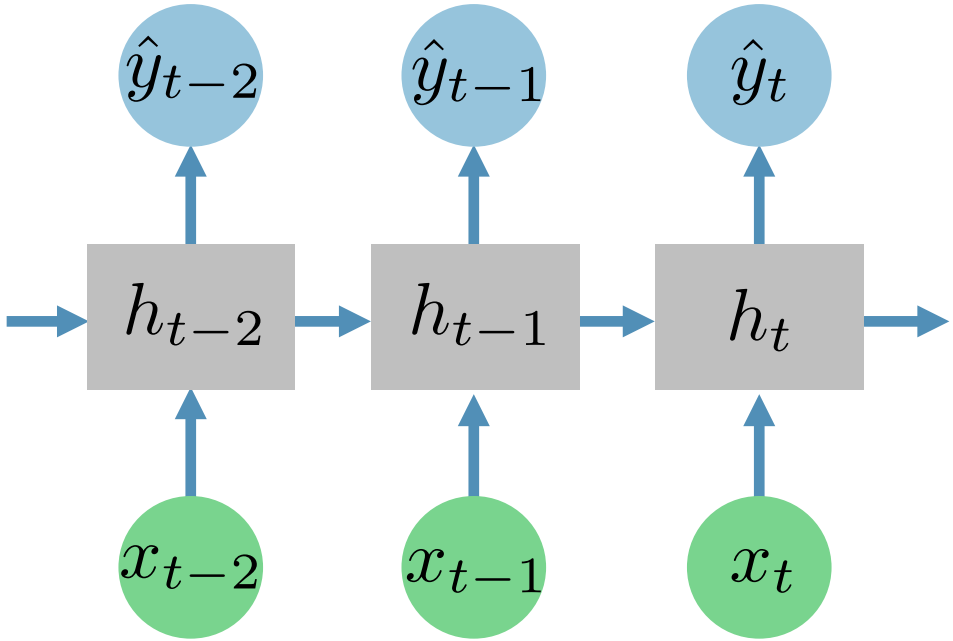
- Exploding gradients are easy to detect but it is not clear how to detect vanishing gradients.
- Exploding gradients: gradient clipping and Truncated BPTT
- Vanishing gradients: ReLU nonlinearity, orthogonal initialisation of the recurrent weights, skip connections.

In the next video:

LSTM and GRU

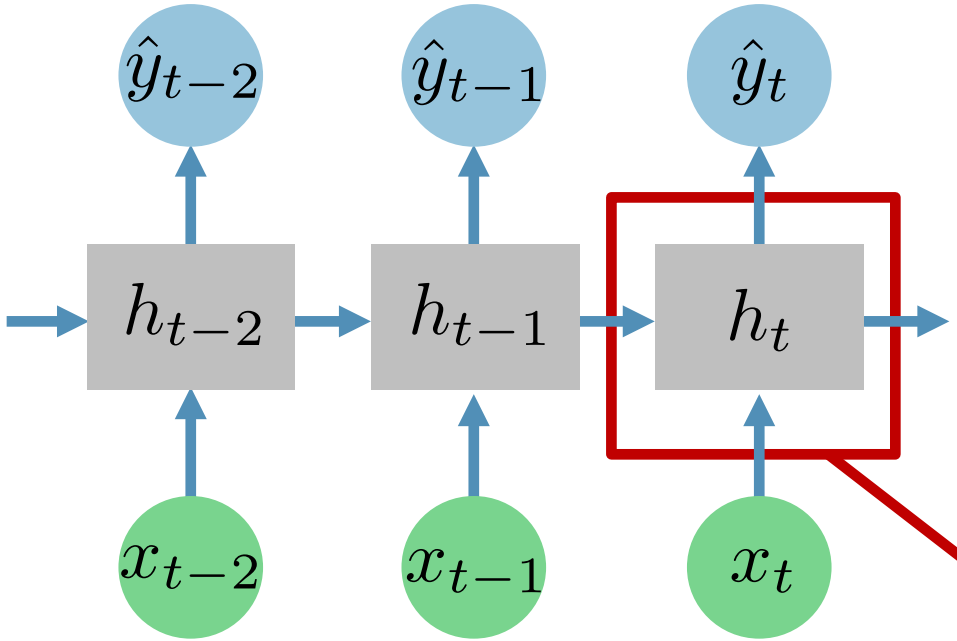
LSTM and GRU

Previously on this week: Simple RNN



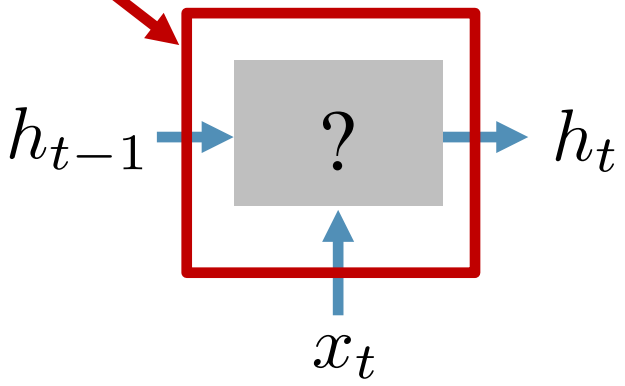
$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

Previously on this week: Simple RNN

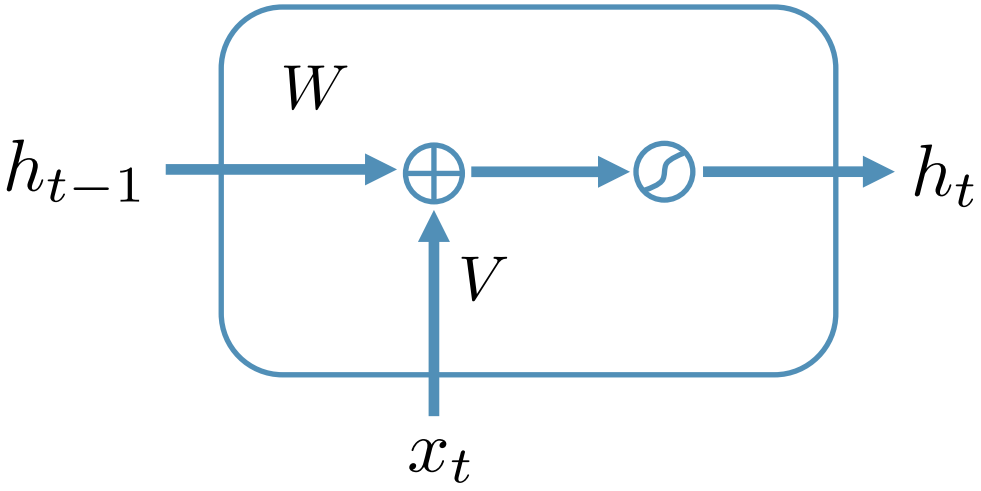


More sophisticated function

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h)$$

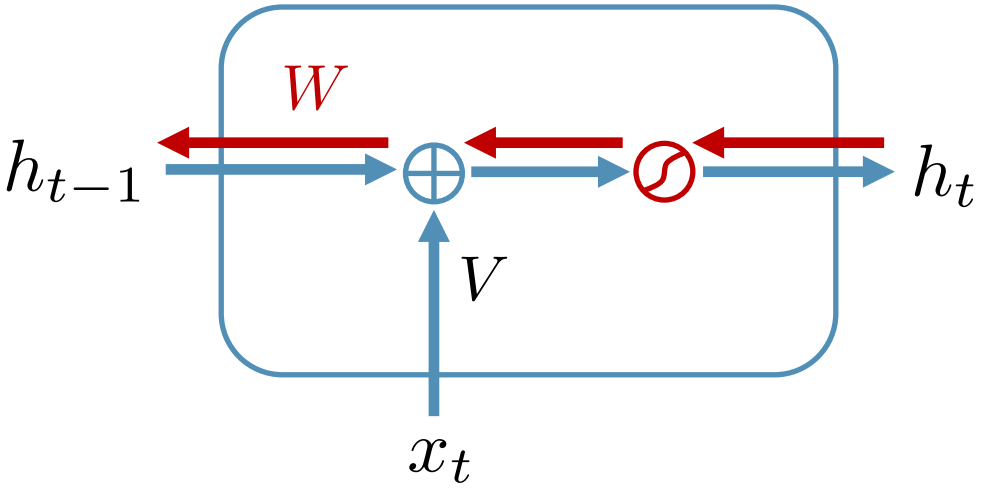


Simple RNN



$$h_t = \tilde{f}(Vx_t + Wh_{t-1} + b_h)$$

Simple RNN



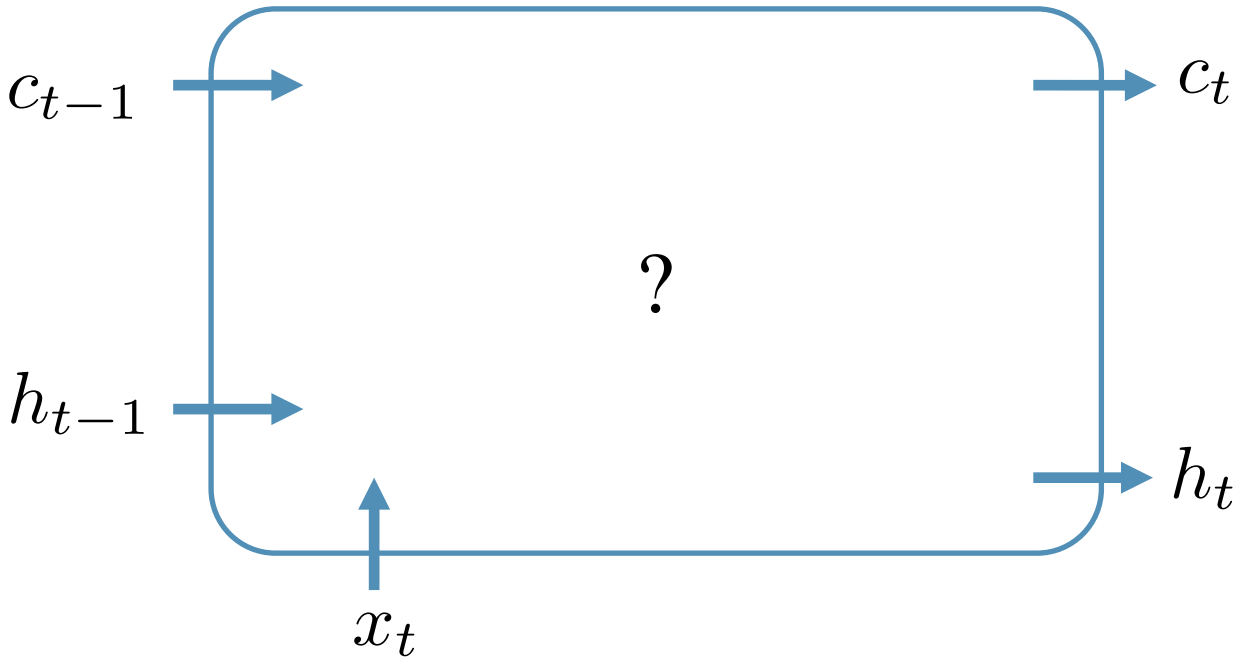
$$h_t = \tilde{f}(Vx_t + Wh_{t-1} + b_h)$$

Backward pass

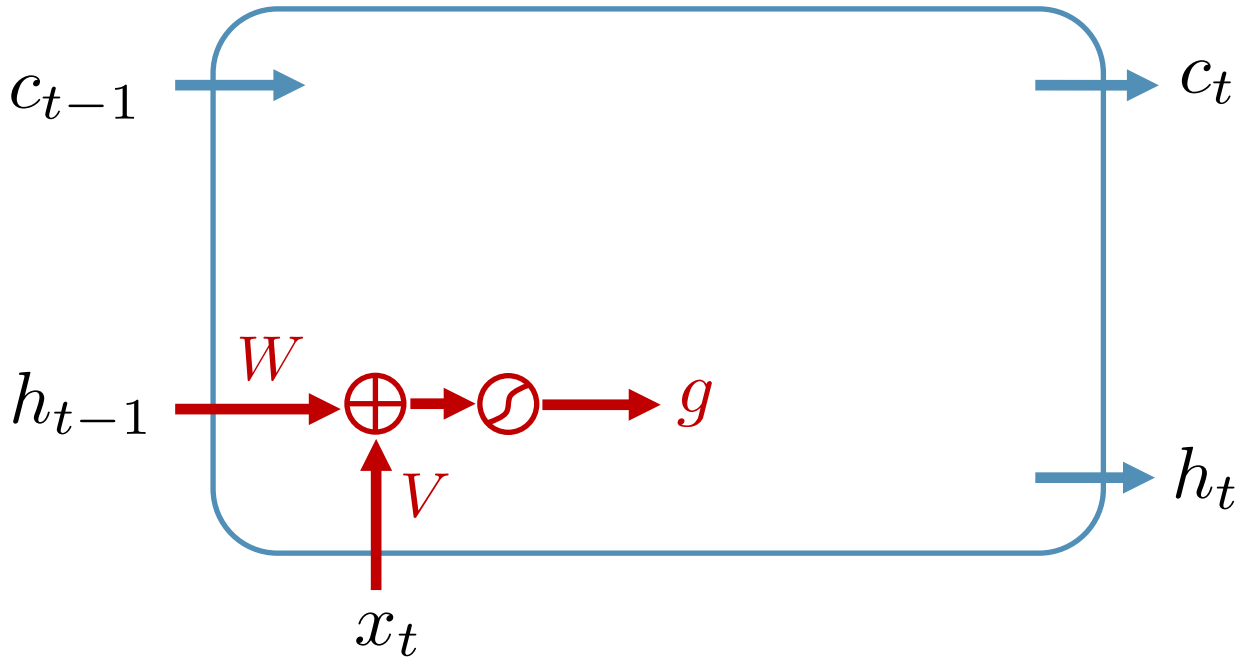
W and nolinearity ⇒ vanishing gradients

We need a short way for the gradients!

LSTM: version 0

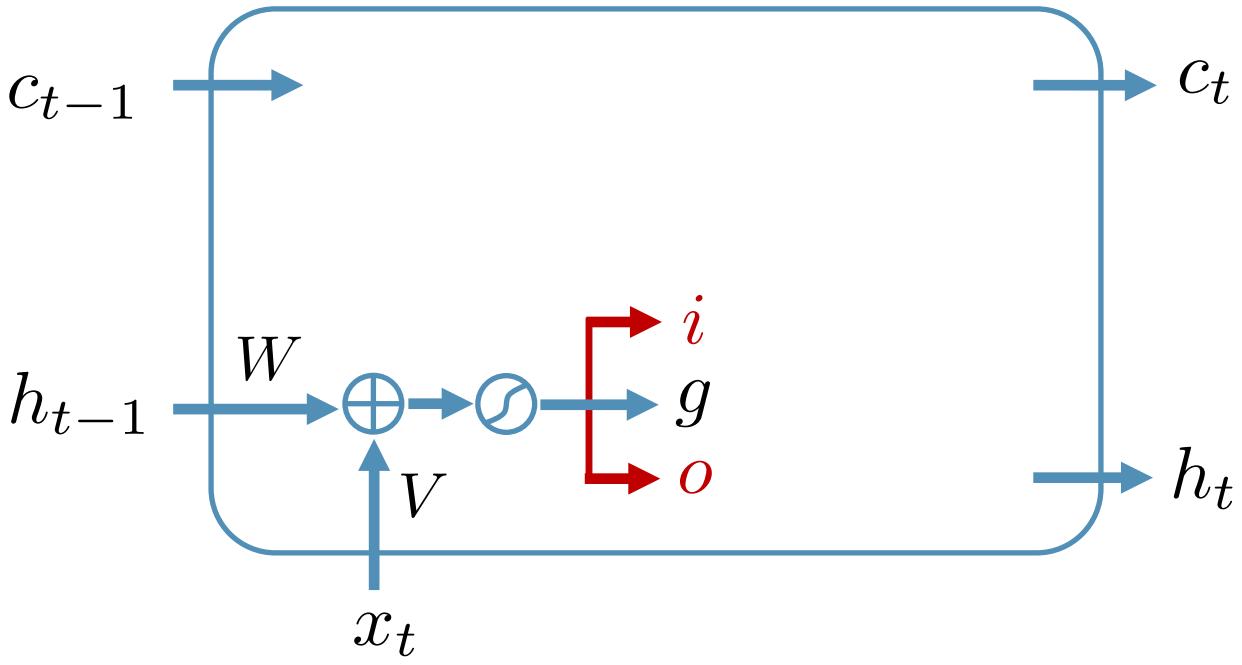


LSTM: version 0



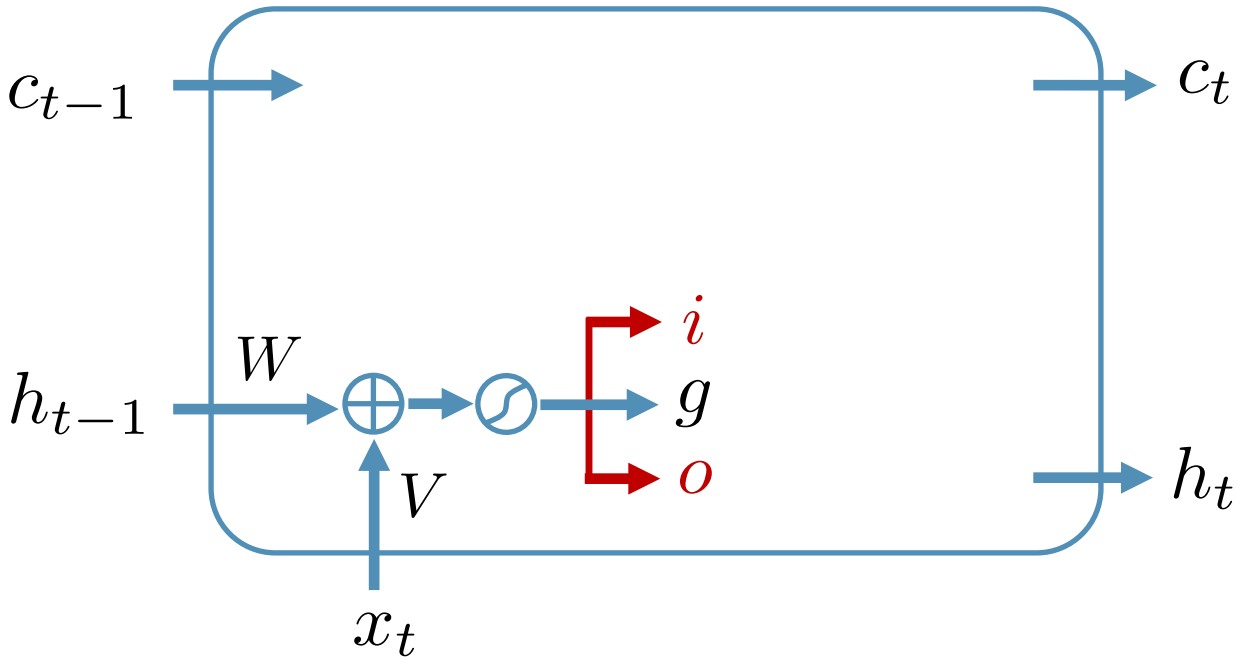
$$g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$$

LSTM: version 0



- $g_t = \tilde{f}(V_g x_t + W_g h_{t-1} + b_g)$
- $i_t = \sigma(V_i x_t + W_i h_{t-1} + b_i)$
- $o_t = \sigma(V_o x_t + W_o h_{t-1} + b_o)$

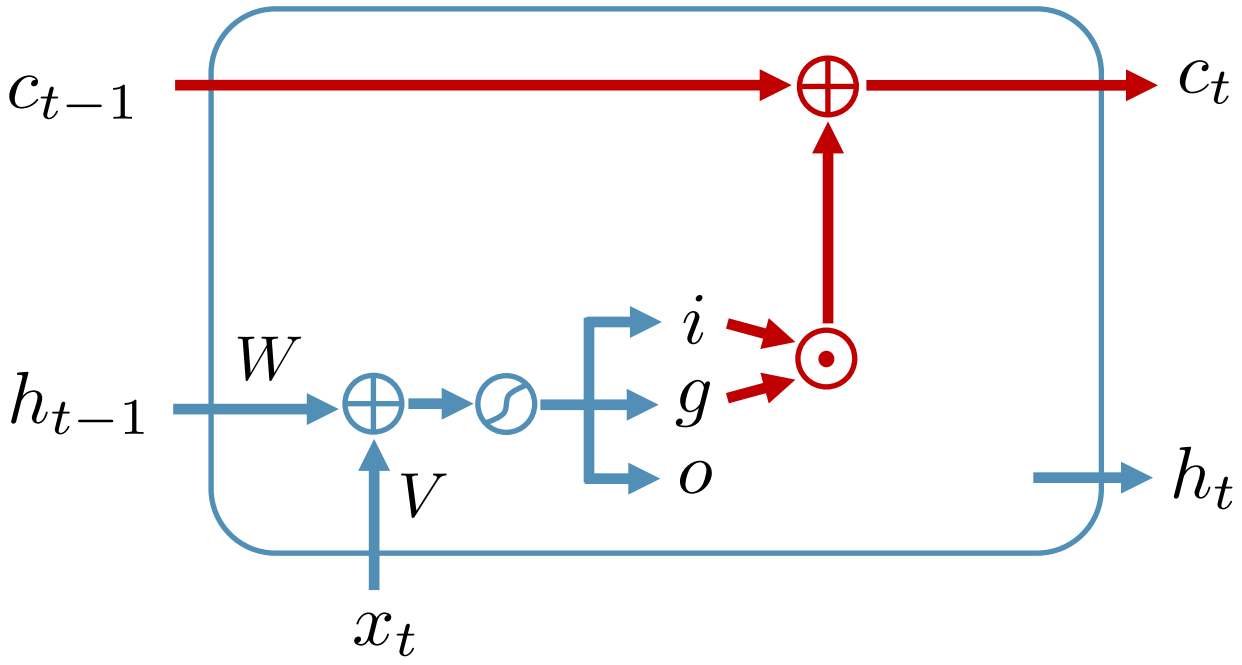
LSTM: version 0



$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b)$$

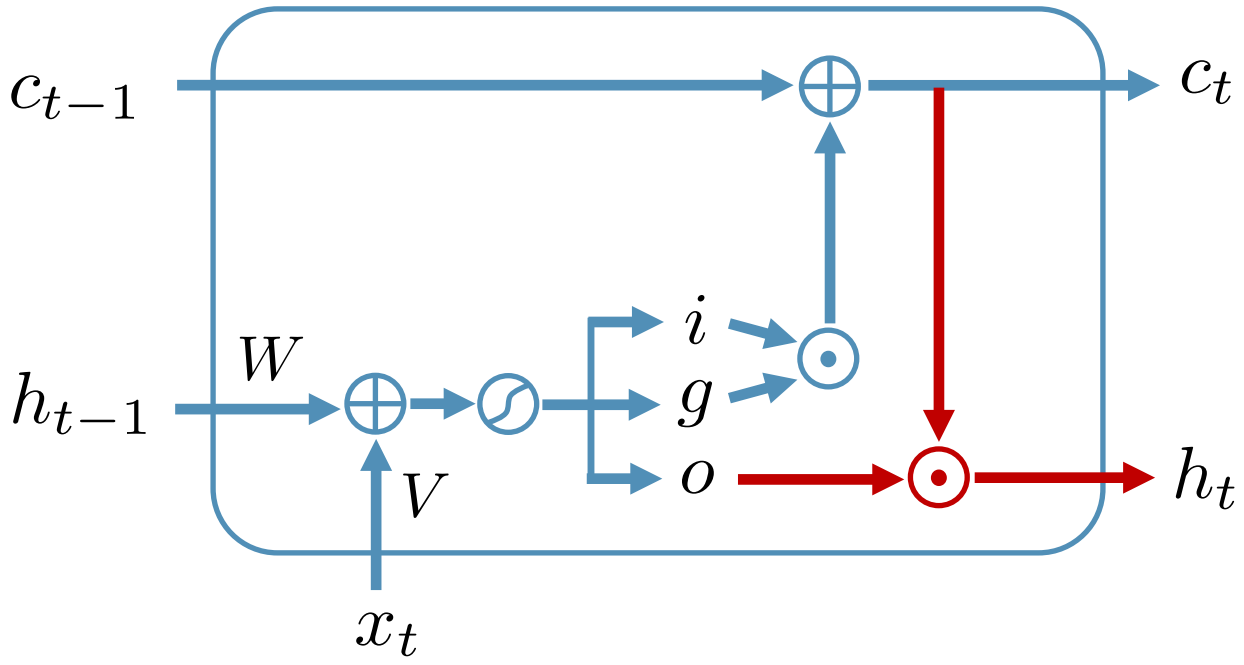
activation functions

LSTM: version 0



$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b) \qquad c_t = c_{t-1} + i_t \cdot g_t$$

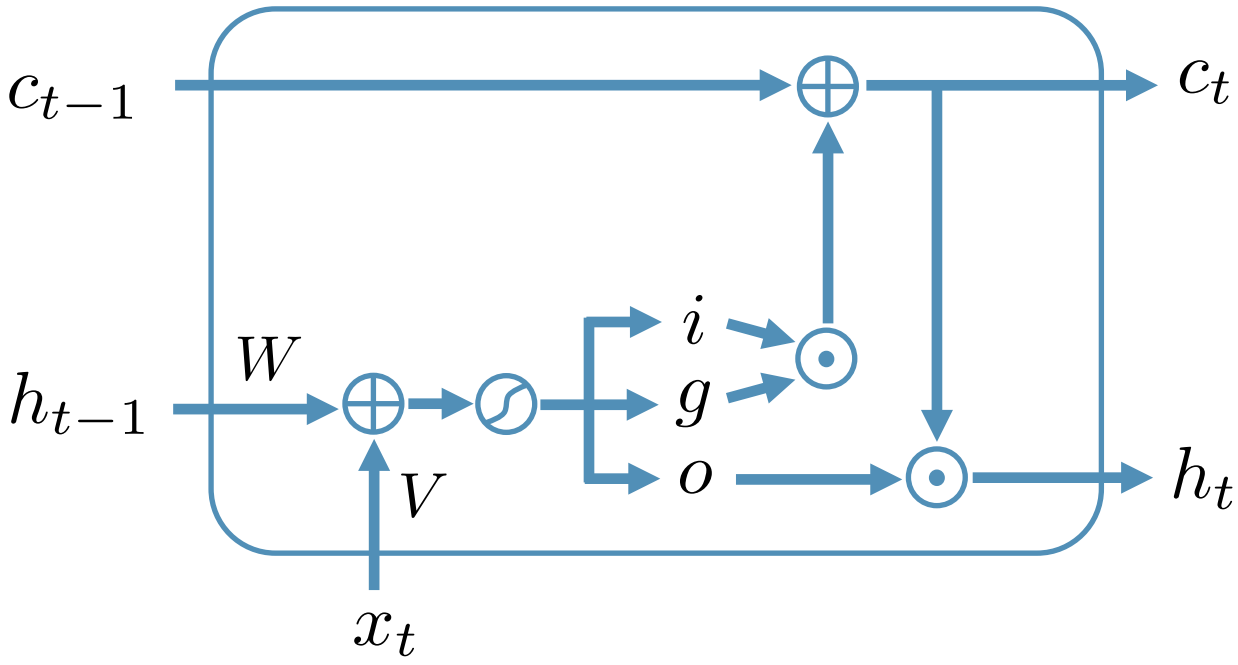
LSTM: version 0



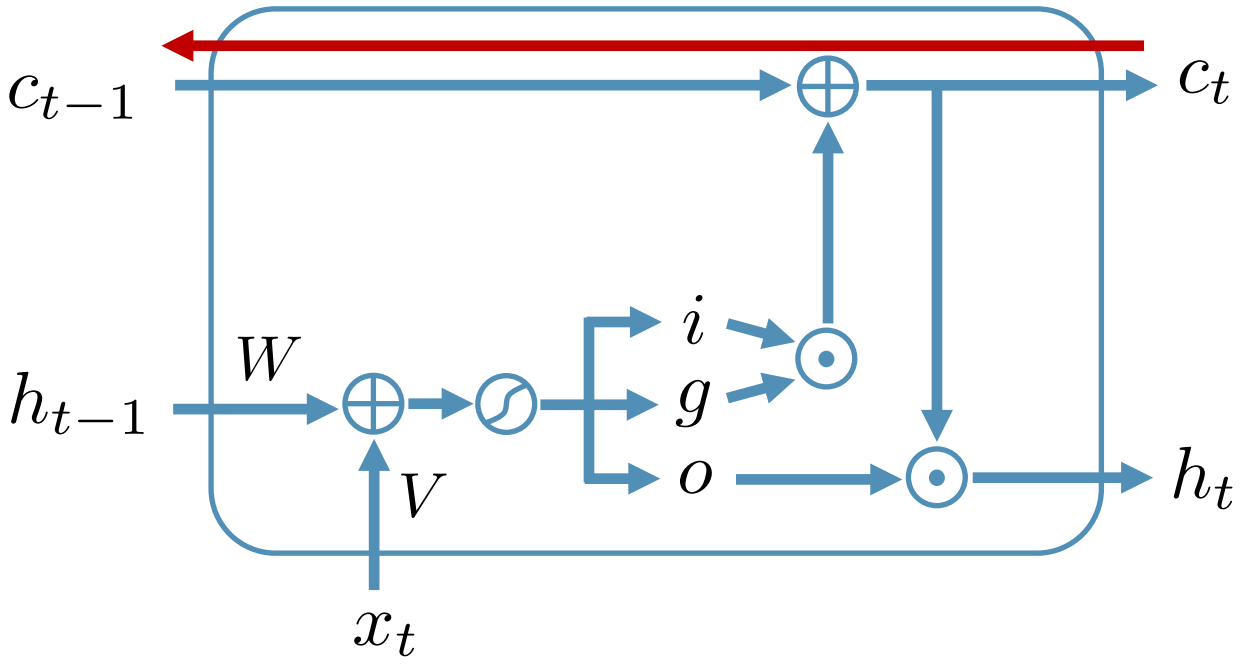
$$\begin{pmatrix} g_t \\ i_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b)$$

- $c_t = c_{t-1} + i_t \cdot g_t$
- $h_t = o_t \cdot \tilde{f}(c_t)$

LSTM: vanishing gradients



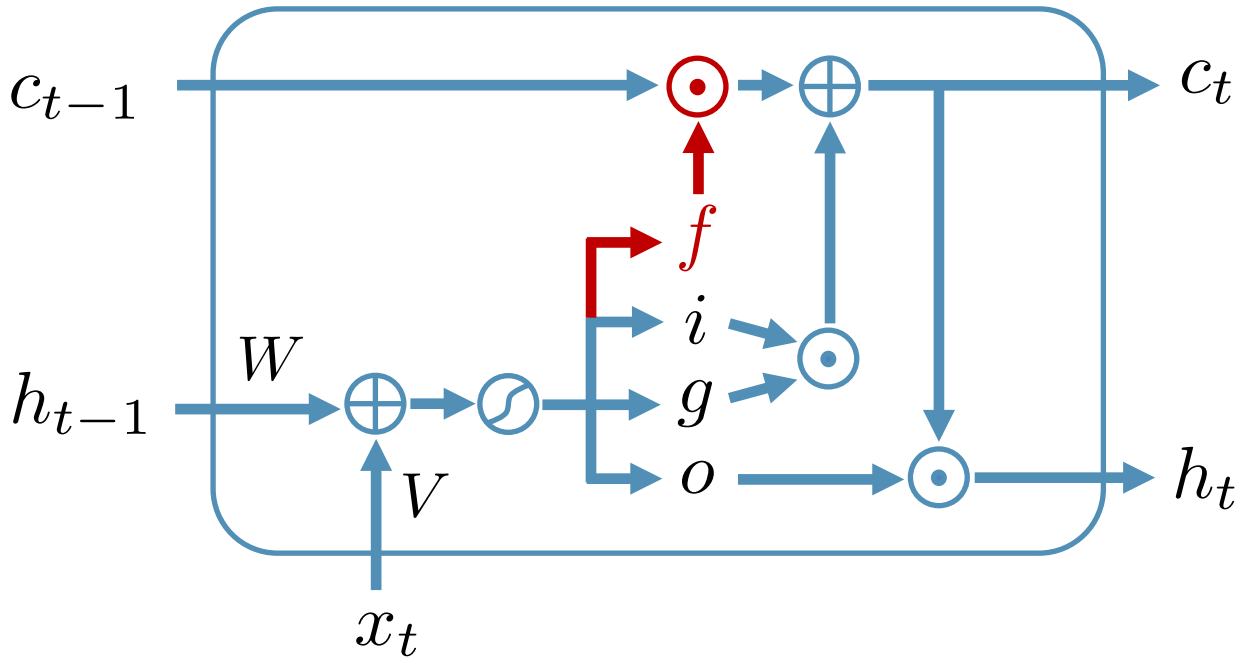
LSTM: vanishing gradients



$$c_t = c_{t-1} + i_t \cdot g_t \qquad \frac{\partial h_t}{\partial h_{t-1}} \Rightarrow \frac{\partial c_t}{\partial c_{t-1}} = diag(1)$$

Gradients do not vanish!

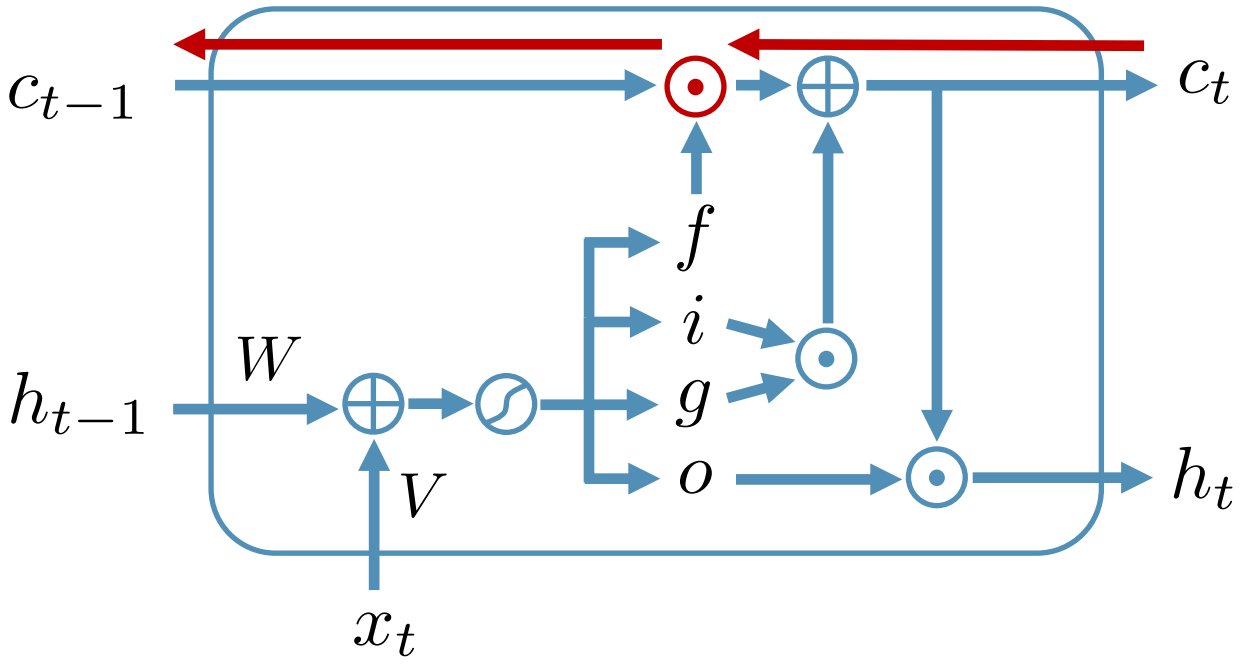
LSTM: forget sometimes



$$\begin{pmatrix} g_t \\ i_t \\ o_t \\ f_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b) \qquad c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$

$$h_t = o_t \cdot \tilde{f}(c_t)$$

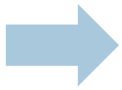
LSTM: forget sometimes



$$f_t = \sigma(V_f x_t + W_f h_{t-1} + b_f)$$

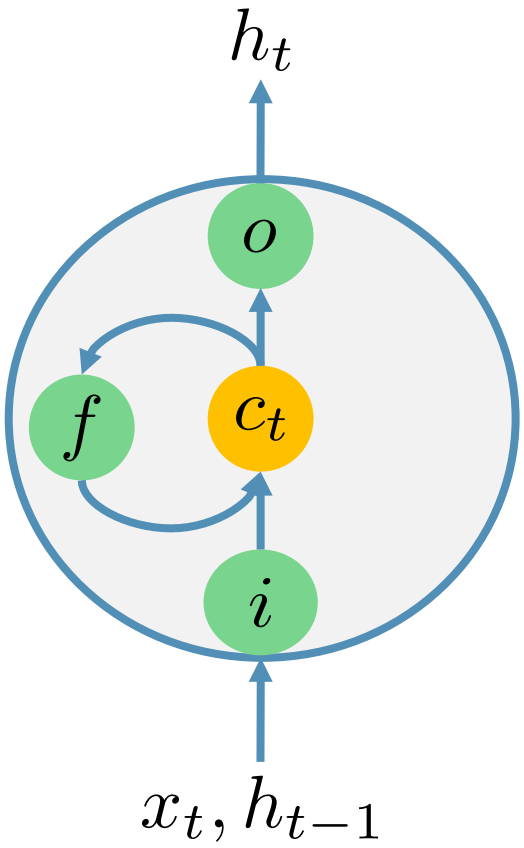
$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = diag(f_t)$$



High initial b_f

LSTM: extreme regimes

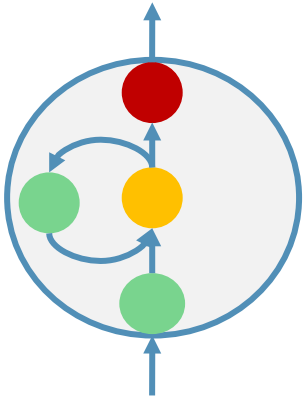


LSTM: extreme regimes

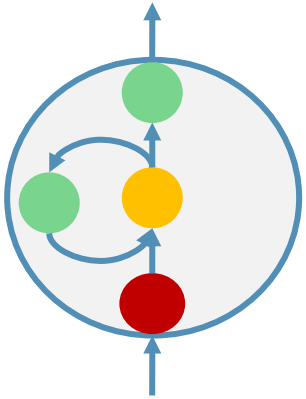
- gate is close

- gate is open

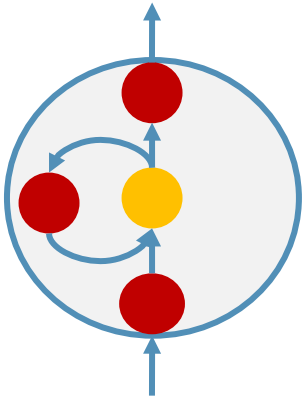
Captures info



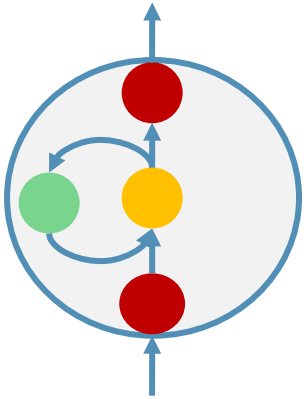
Releases info



Erases info



Keeps info



= RNN

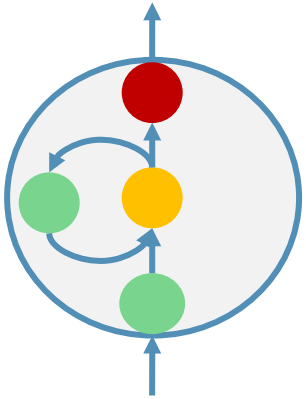
?

LSTM: extreme regimes

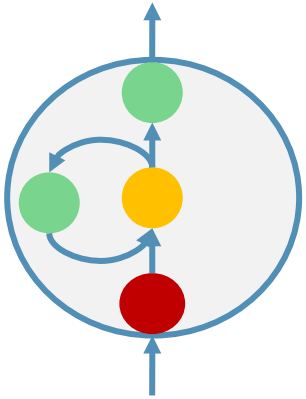
- gate is close

- gate is open

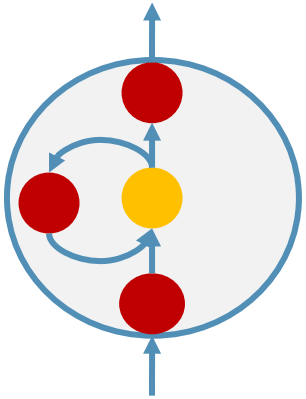
Captures info



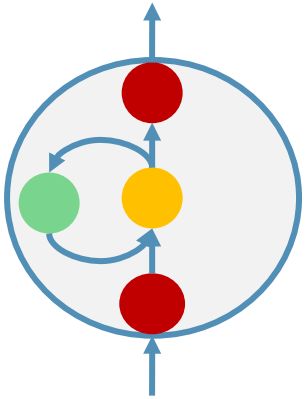
Releases info



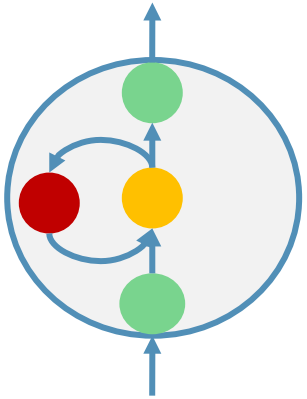
Erases info



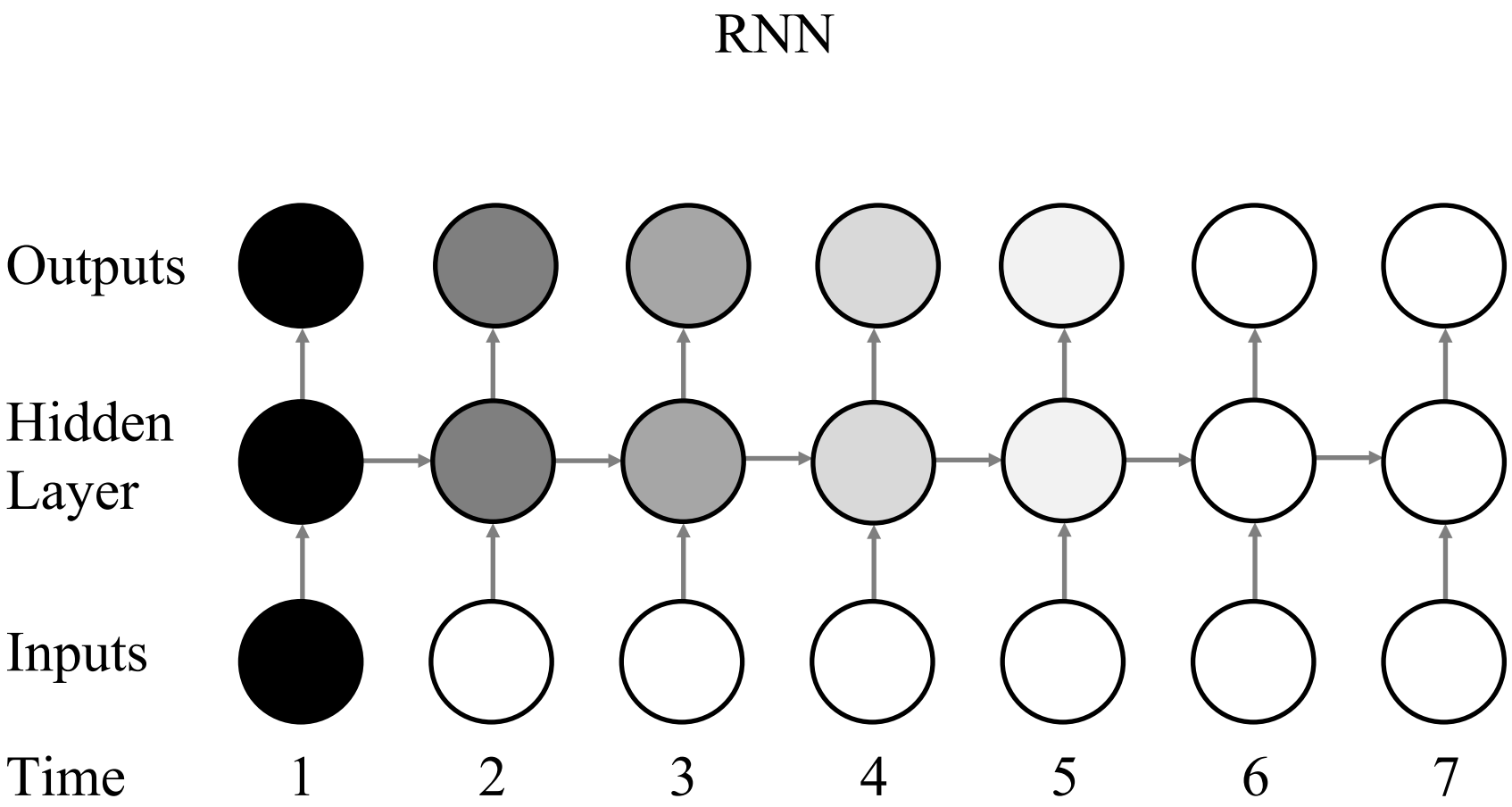
Keeps info



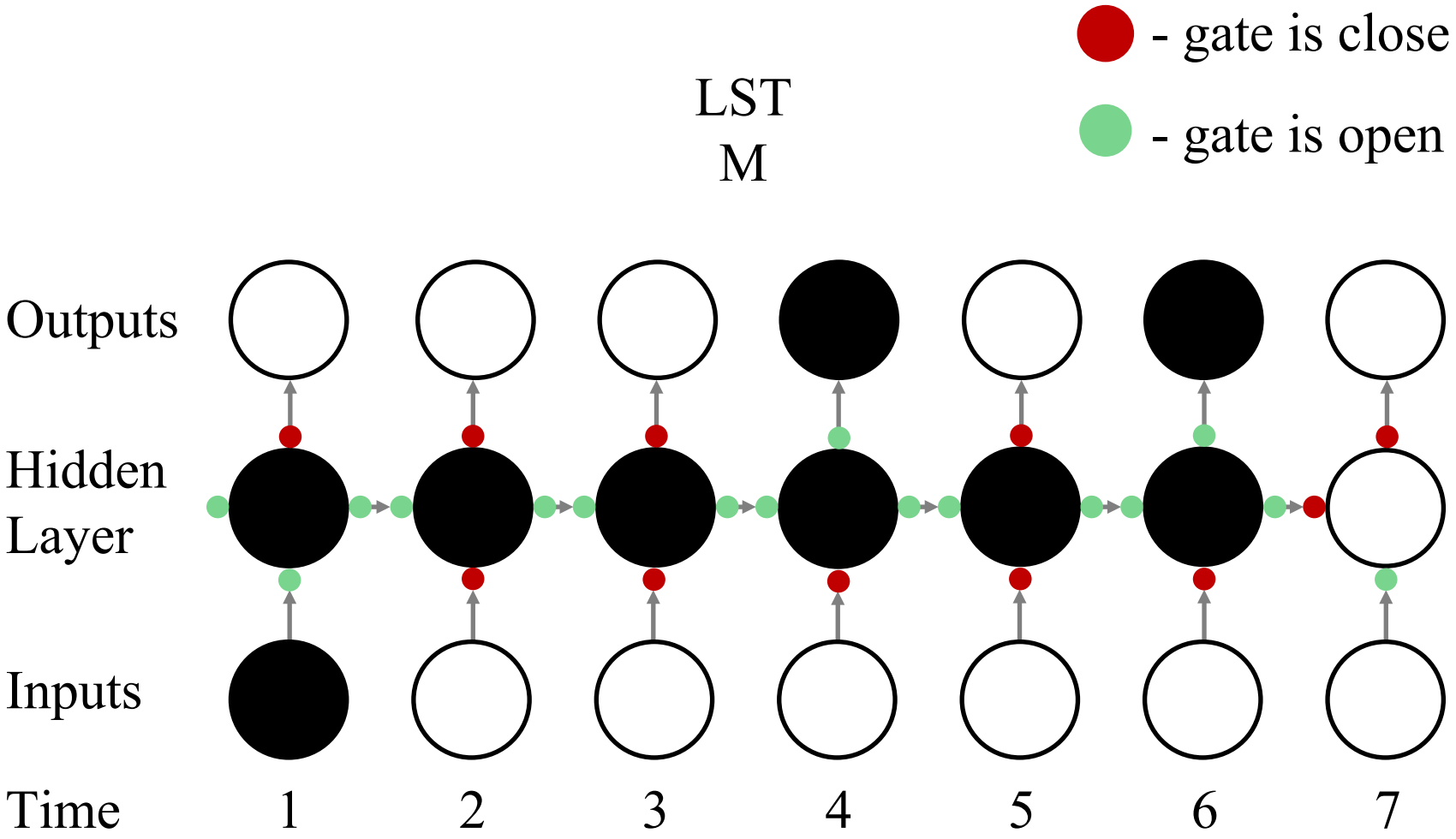
= RNN



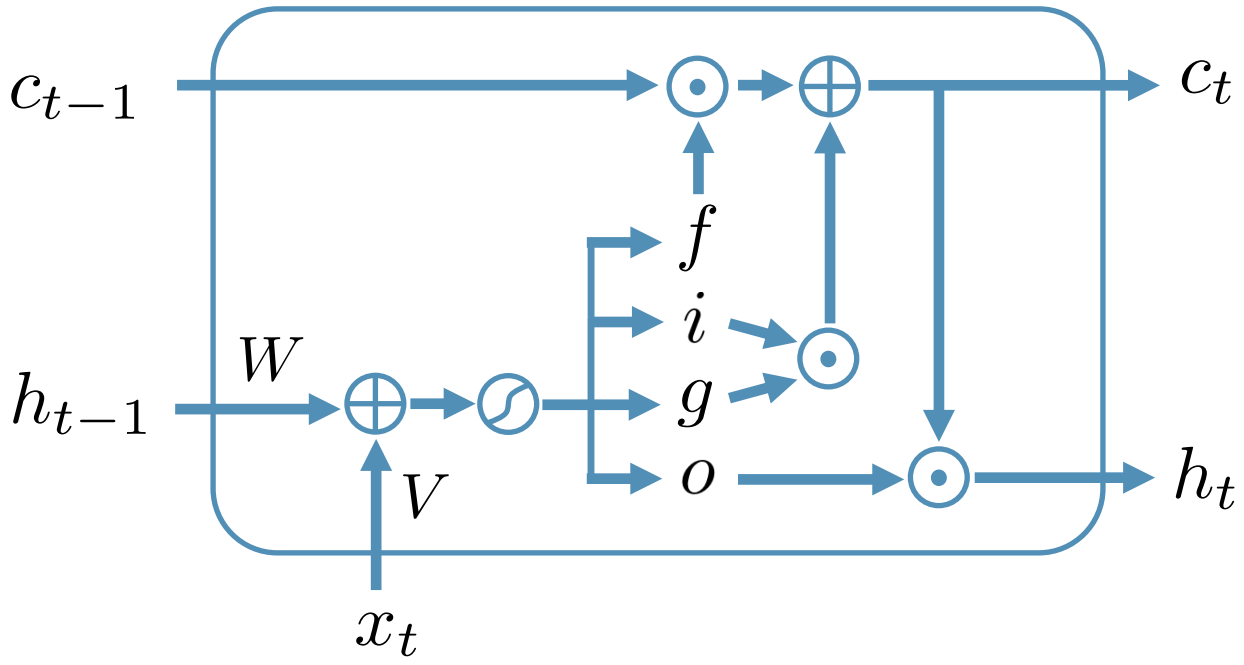
LSTM: information flow



LSTM: information flow



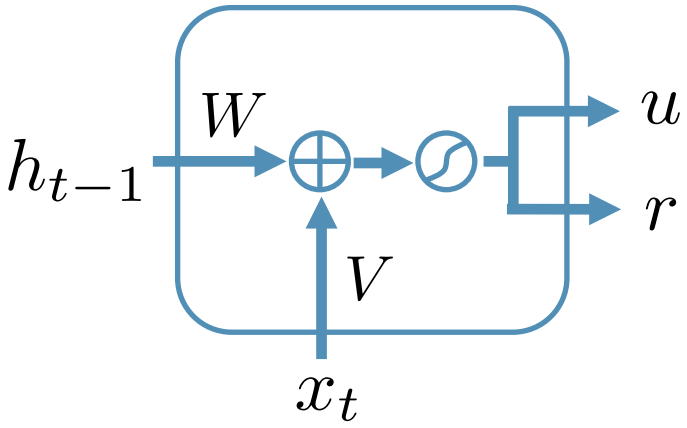
LSTM

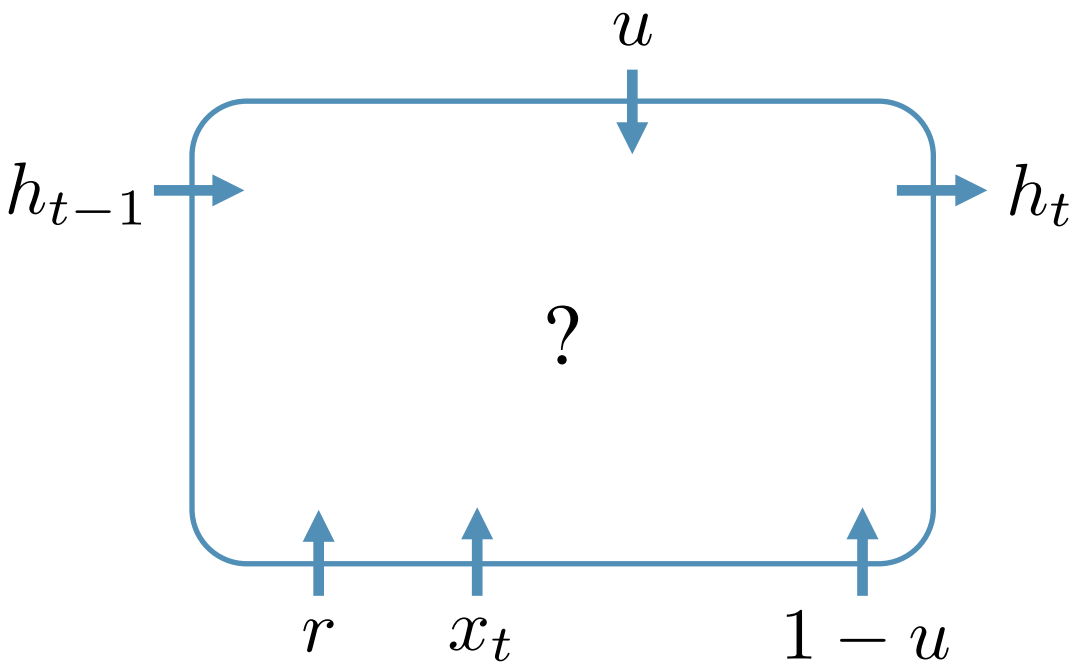


- $$\begin{pmatrix} g_t \\ i_t \\ o_t \\ f_t \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} (Vx_t + Wh_{t-1} + b)$$

- $c_t = f_t \cdot c_{t-1} + i_t \cdot g_t$
- $h_t = o_t \cdot \tilde{f}(c_t)$

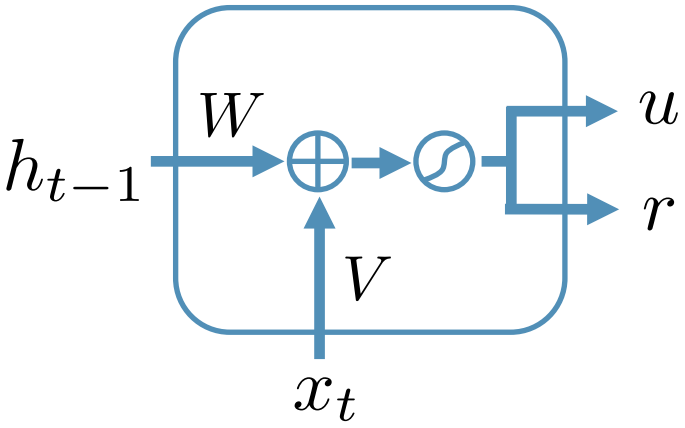
GRU

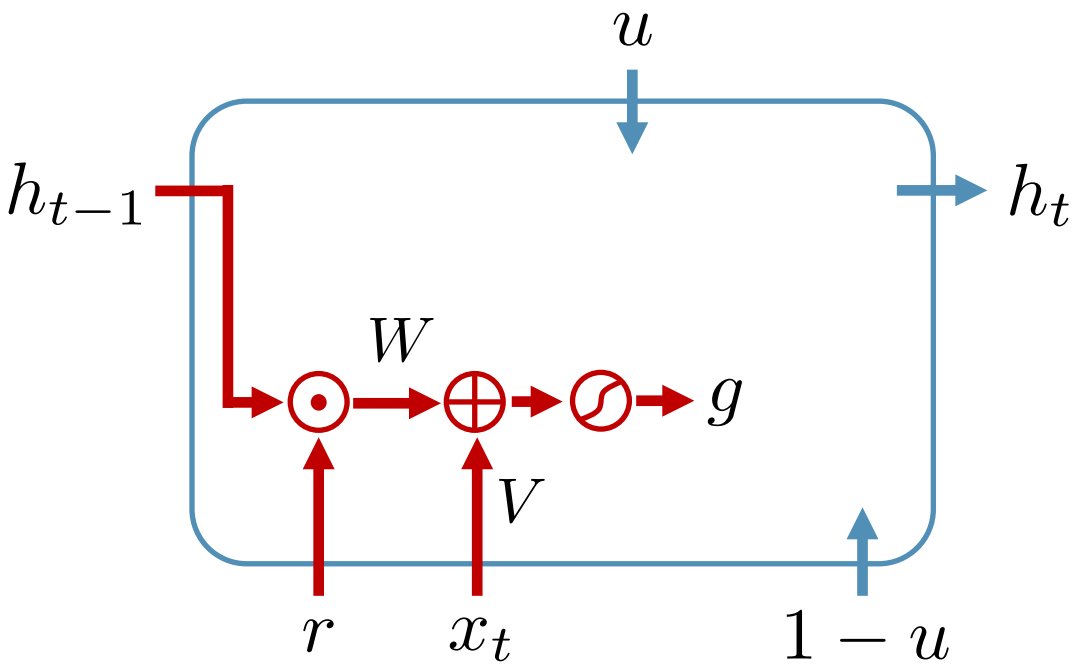




$$\begin{pmatrix} r_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b)$$

GRU

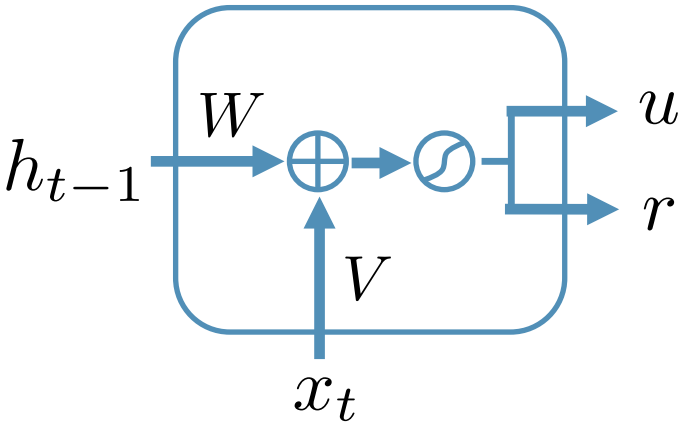


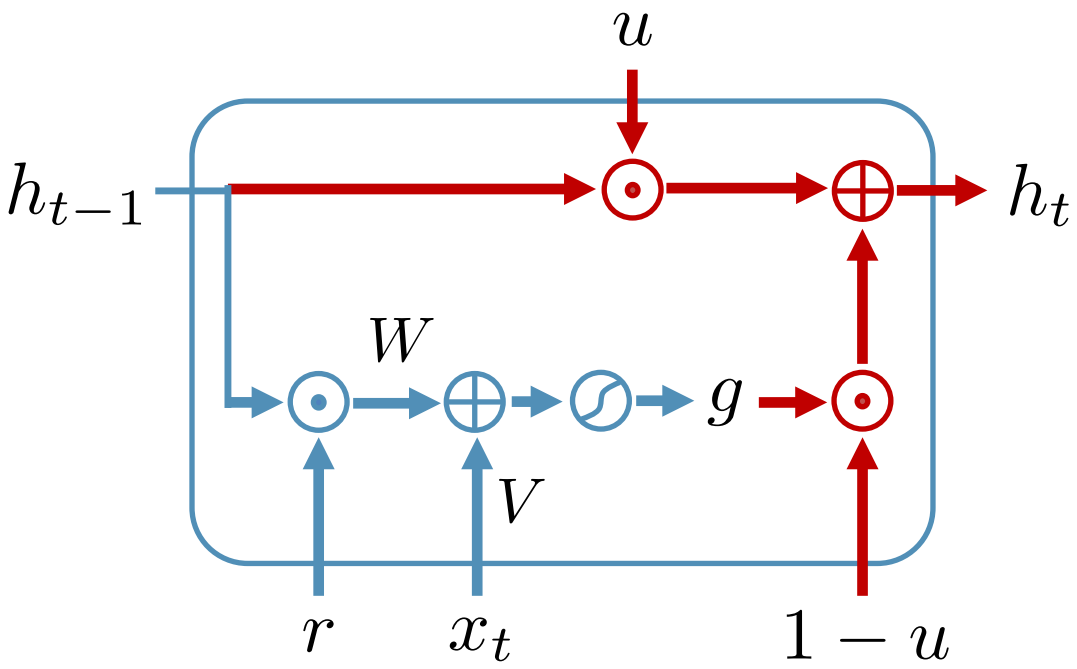


$$\begin{pmatrix} r_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b) \qquad g_t = \tilde{f}\left(V_g x_t + W_g(h_{t-1} \cdot r_t) + b_g\right)$$

GRU

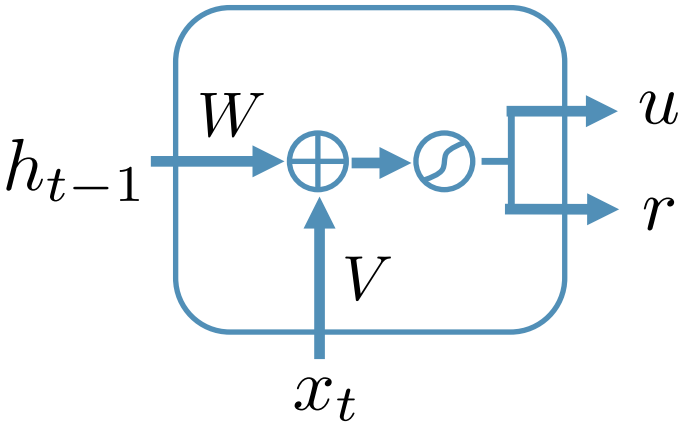
Gated Recurrent Unit

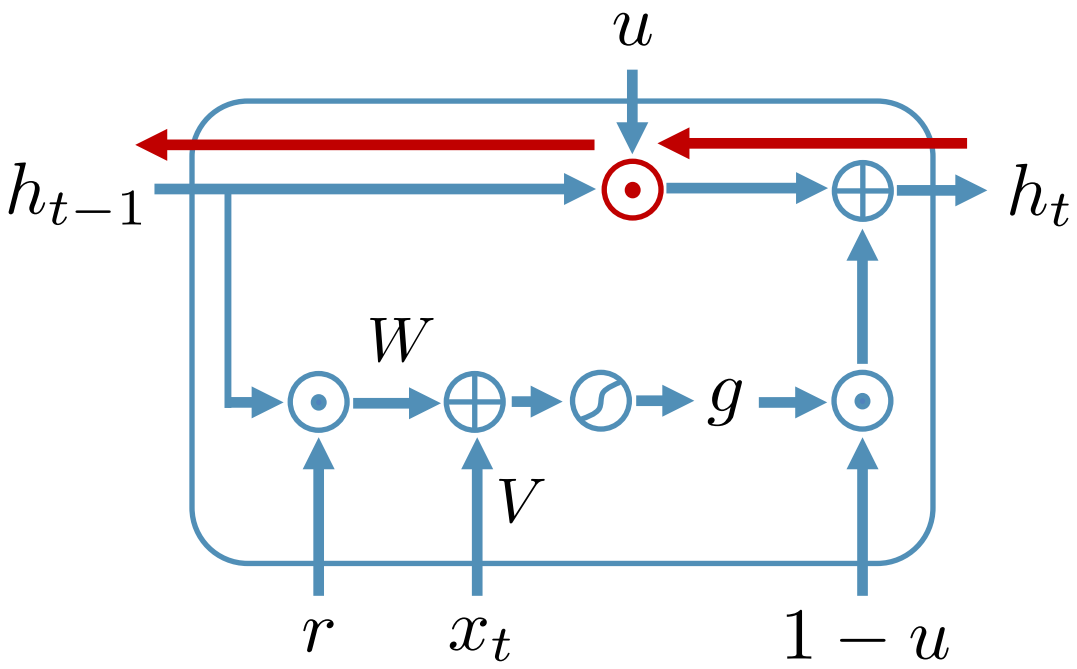




- $\begin{pmatrix} r_t \\ u_t \end{pmatrix} = \sigma(Vx_t + Wh_{t-1} + b)$
- $g_t = \tilde{f}\left(V_g x_t + W_g(h_{t-1} \cdot r_t) + b_g\right)$
- $h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$

GRU: vanishing gradients

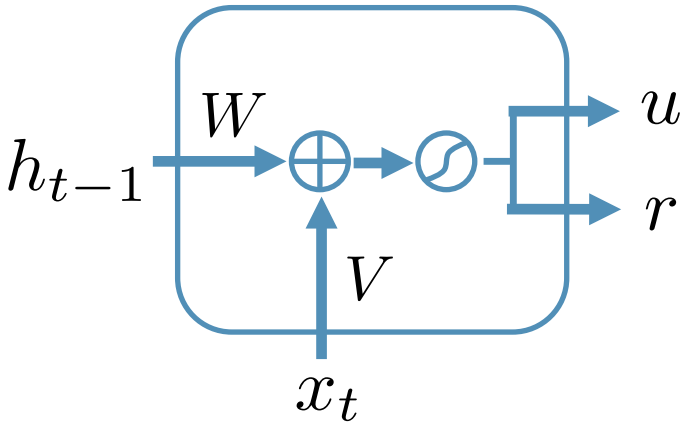


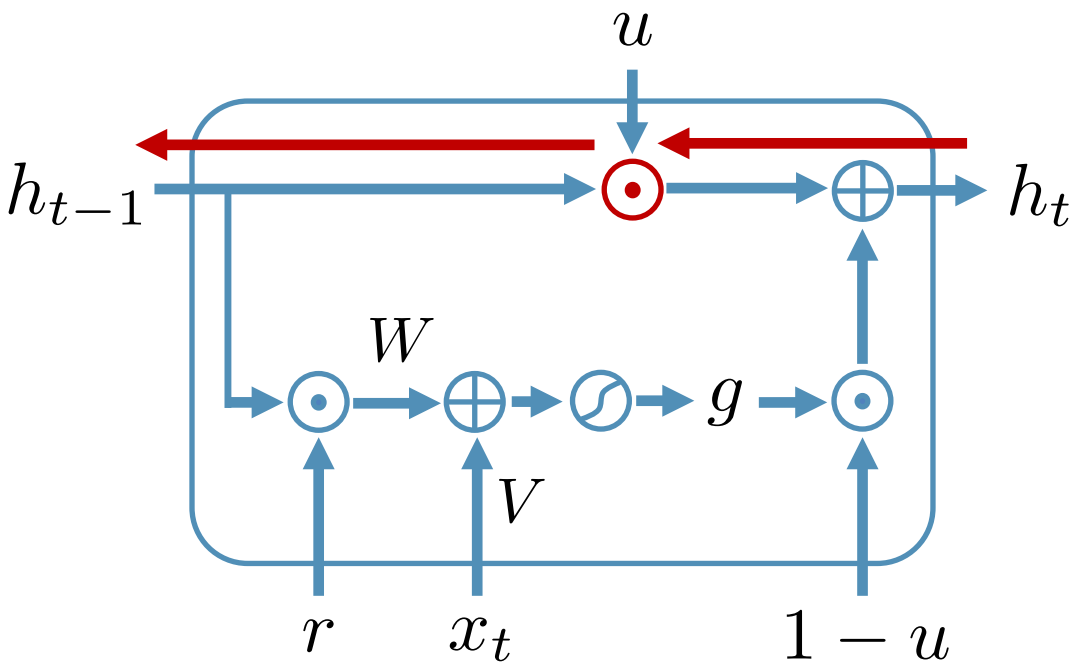


$$u_t = \sigma(V_u x_t + W_u h_{t-1} + b_u)$$

$$h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$$

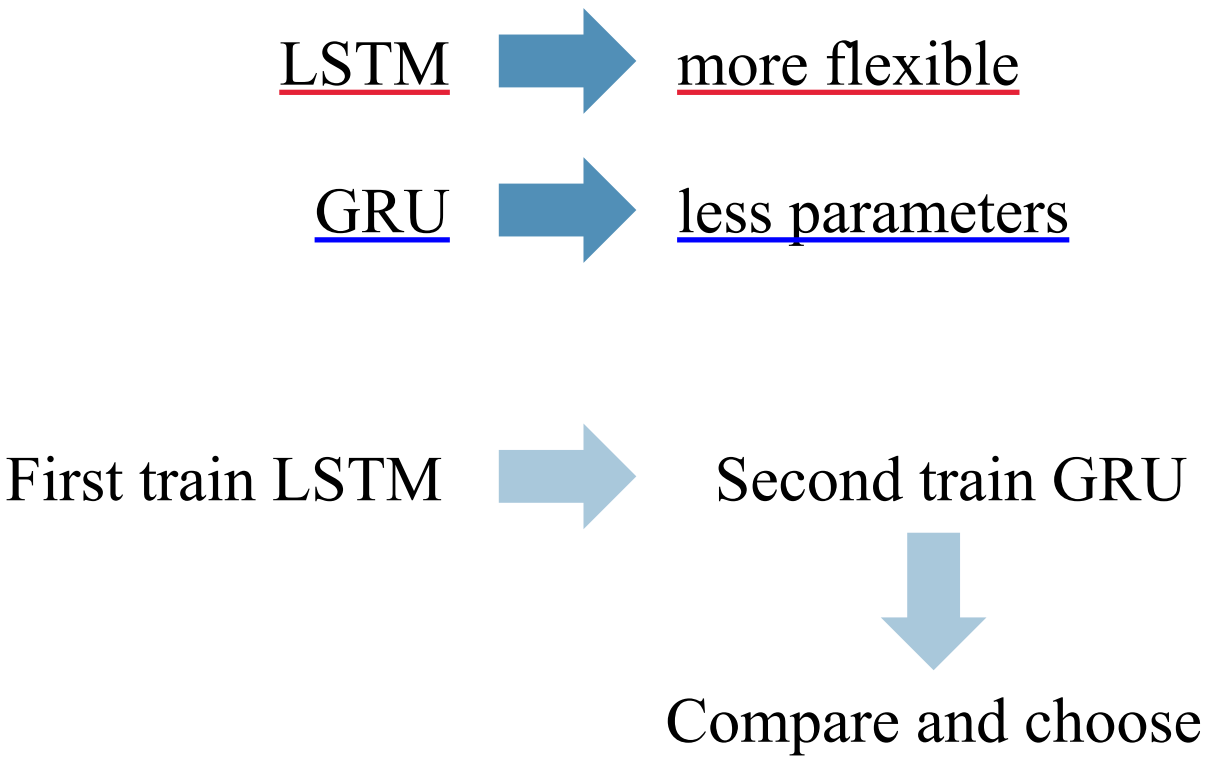
GRU: vanishing gradients



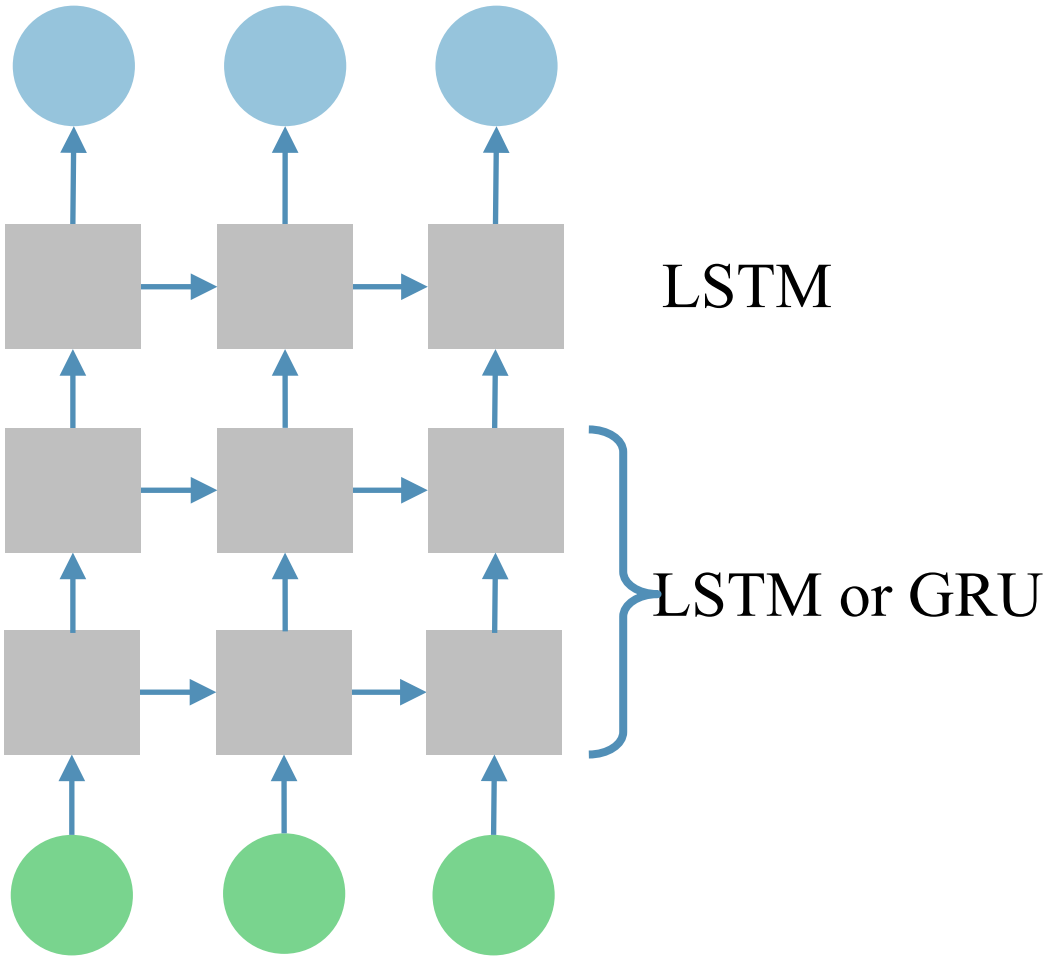


- $u_t = \sigma(V_u x_t + W_u h_{t-1} + b_u)$
- $h_t = (1 - u_t) \cdot g_t + u_t \cdot h_{t-1}$
- $\frac{\partial h_t}{\partial h_{t-1}} = diag(1 - u_h) \cdot \frac{\partial g_h}{\partial h_{h-1}} + diag(u_h)$ ⇒ High initial b_u

LSTM or GRU?


LSTM
more flexible
GRU
less parameters
First train LSTM
Second train GRU
Compare and choose

LSTM or GRU: stack more layers



Summary

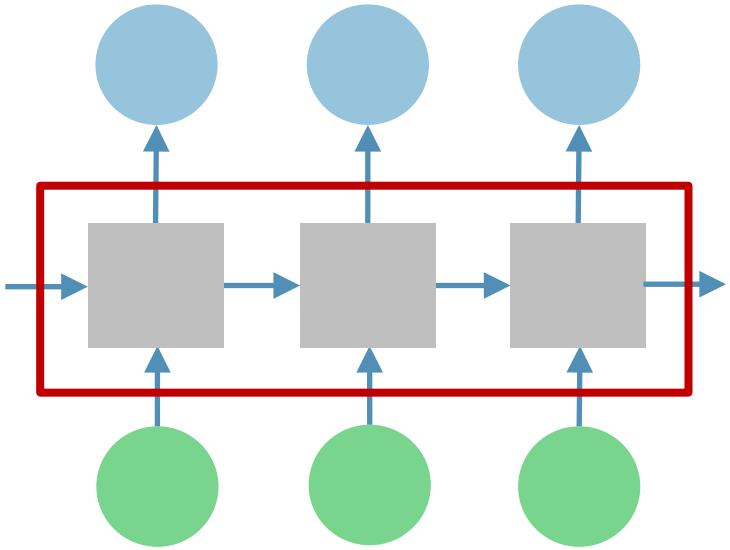
- Gated recurrent architectures: LSTM and GRU.
- They do not suffer from vanishing gradients that much because there is an additional short way for the gradients through them

In the next video:

How to use RNNs to solve different practical tasks

Practical use cases

Previously on this week: Recurrent architecture

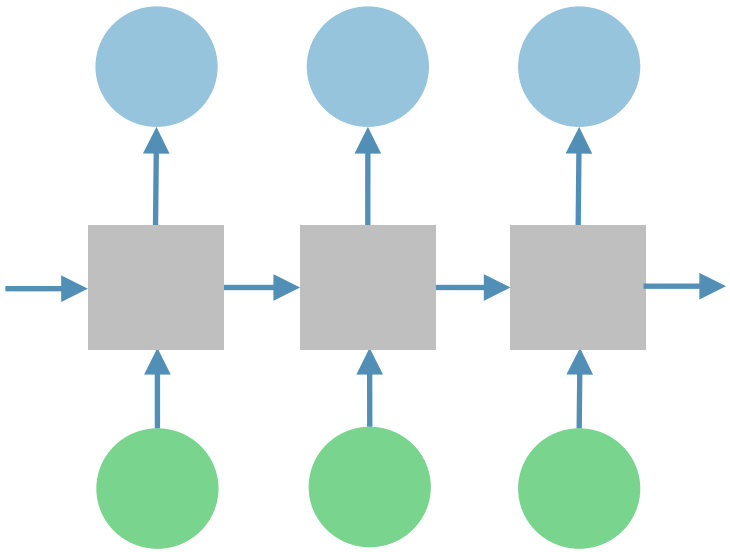


One or more layers:

- Simple RNN
- LSTM
- GRU
- …

Elements-wise classification

1



Input sequence

Output sequence

Input and output are synchronized

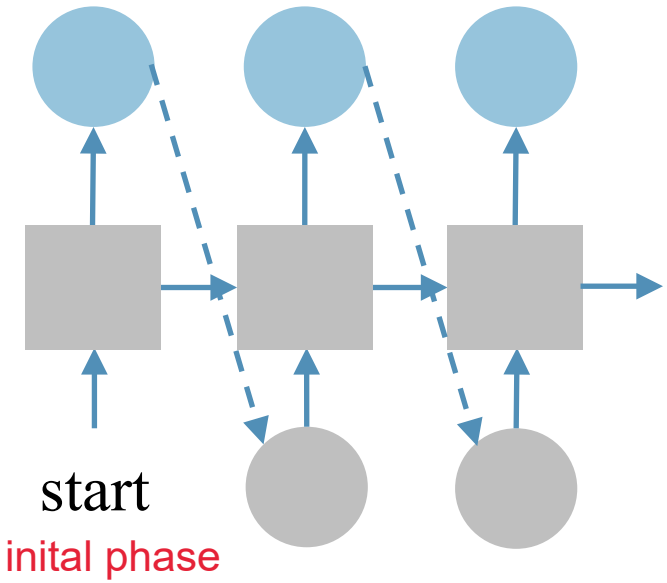
one output for each input

Tasks

- POS tagging
- Video frames classification

Sequence generation 2

Unconditional



Input ---

Output sequence

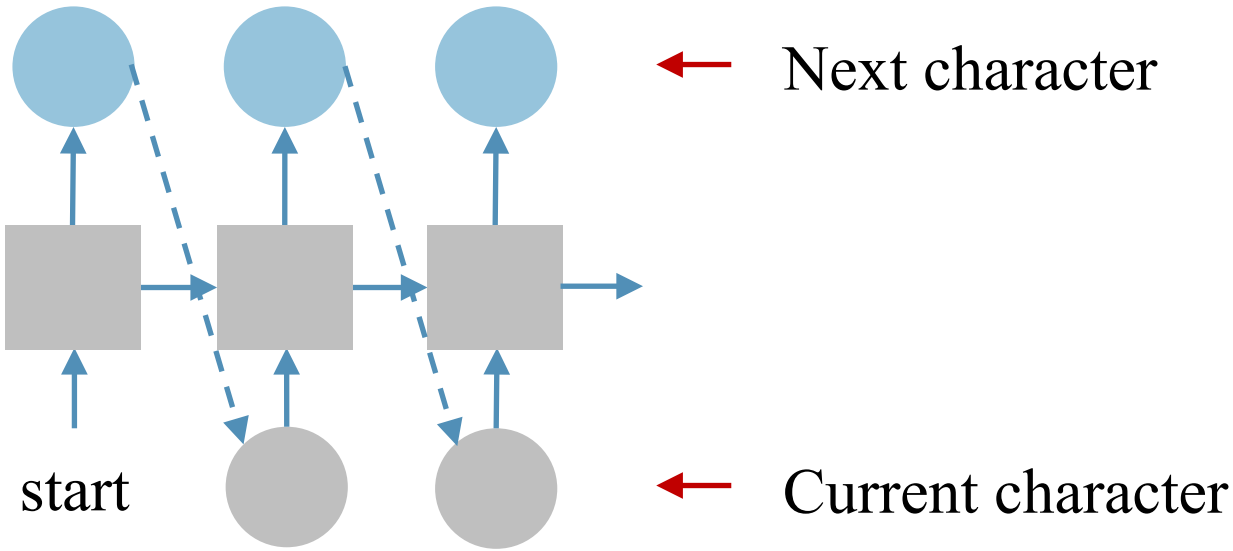
Tasks

- Character-based language model
- Word-based language model
- Music generation
- Speech generation
- Handwriting generation
- …

Char-based language model: Shakespeare

2.1

Model	3 layer LSTM of 512 units
Training data	all the works of Shakespeare



Char-based language model: Shakespeare

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

one character at each time step, not even a word

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

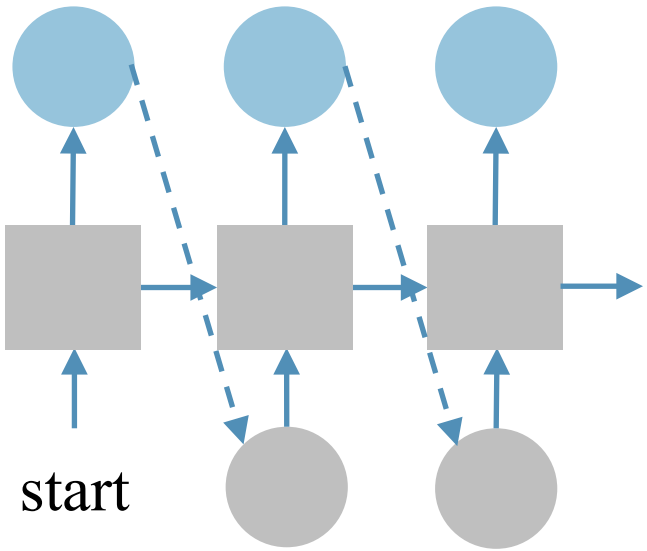
Clown:
Come, sir, I will make did behold your worship.

Andrej Karpathy, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Handwriting generation

2.2

We predict handwriting point by point



← Next pen position:

x, y – mixture of Gaussians

z – Bernoulli distribution

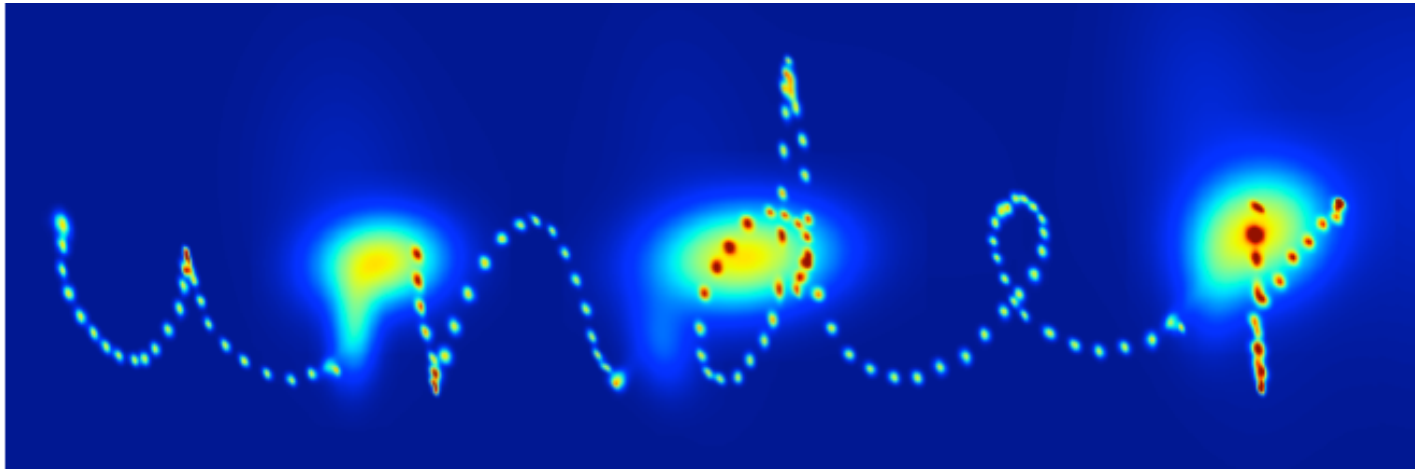
← Current pen position:

x, y – pen offset

z – do we end a stroke here?

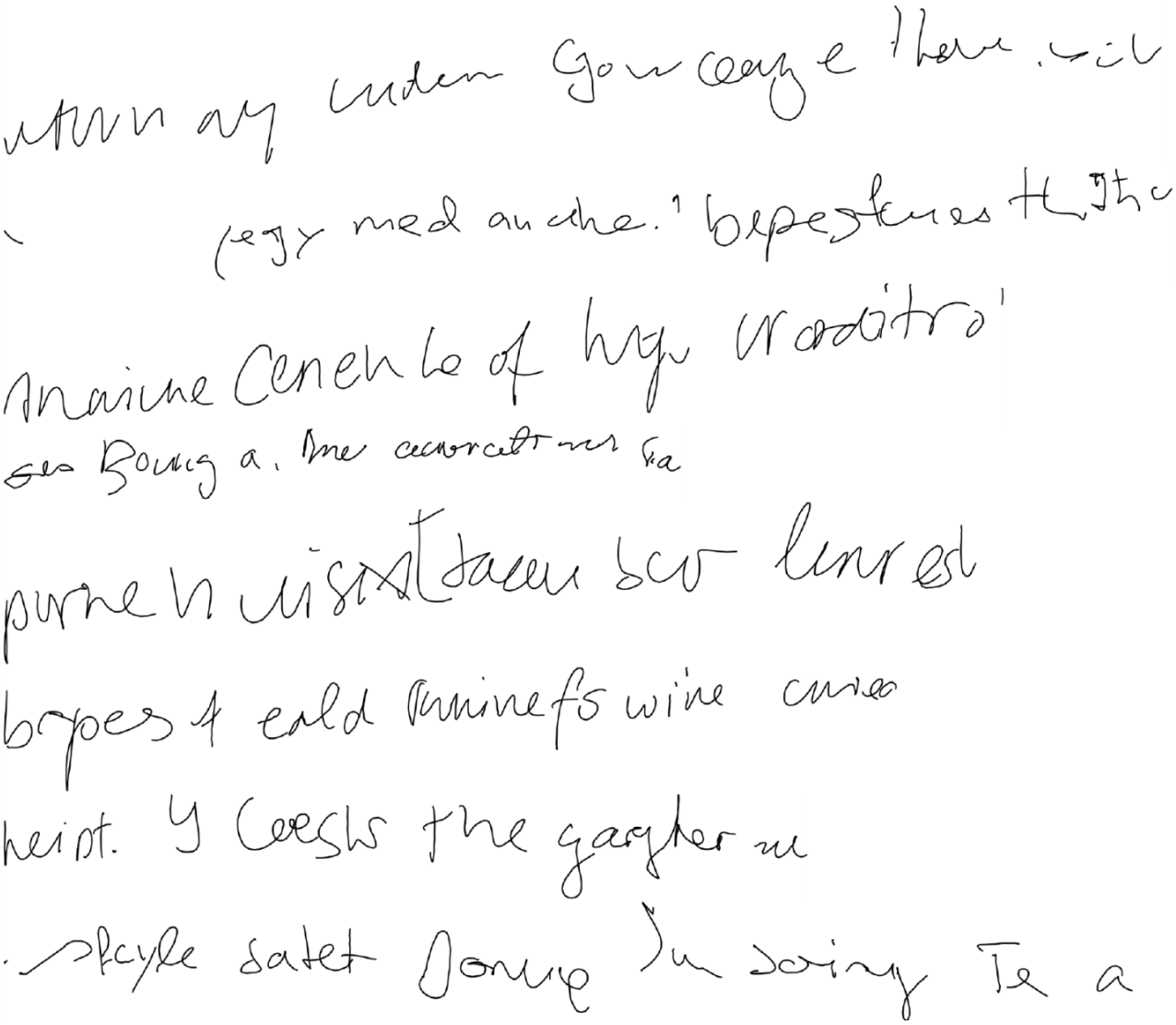
Handwriting generation

We predict handwriting point by point



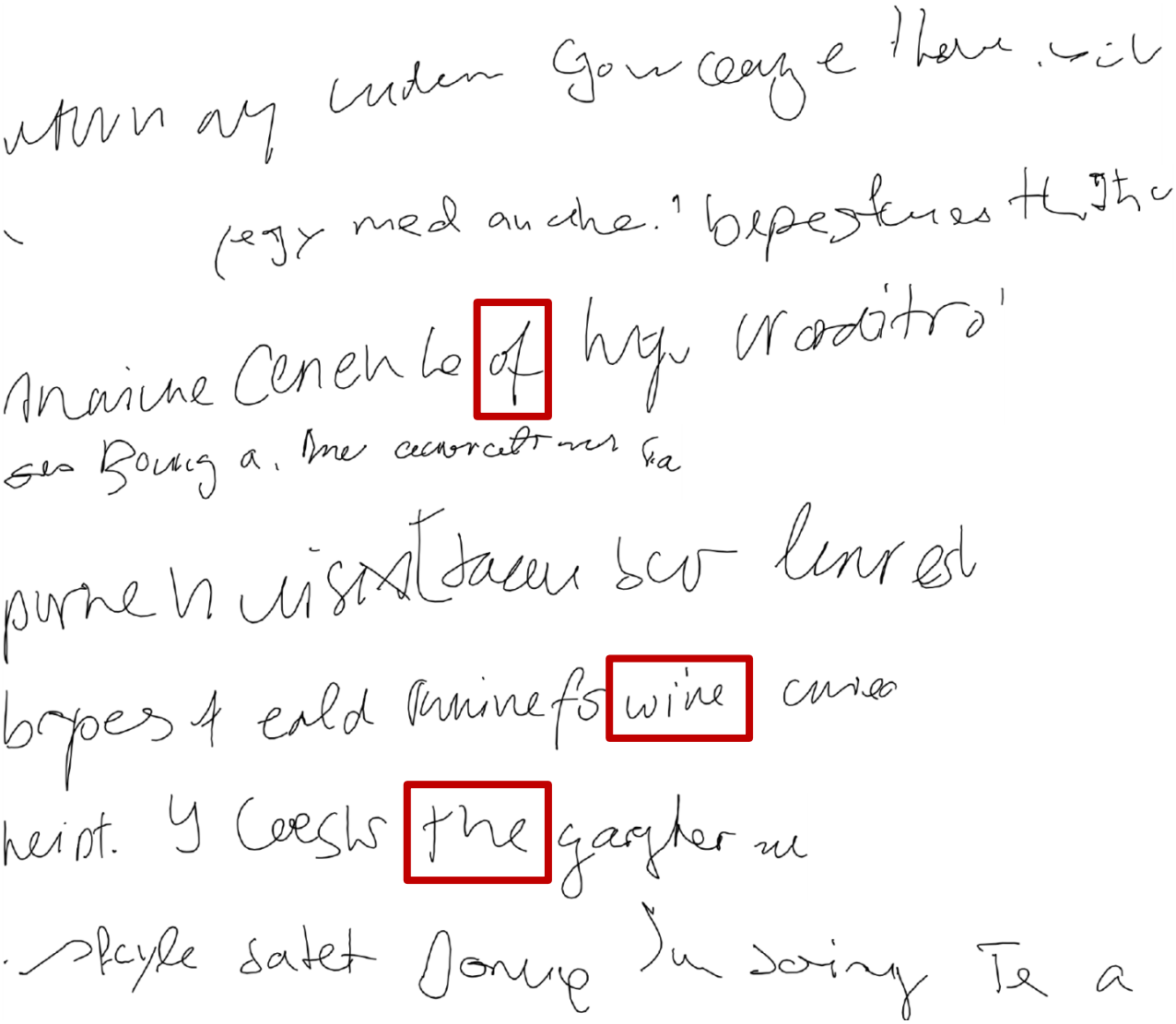
Alex Graves, https://arxiv.org/pdf/1308.0850.pdf

Handwriting generation



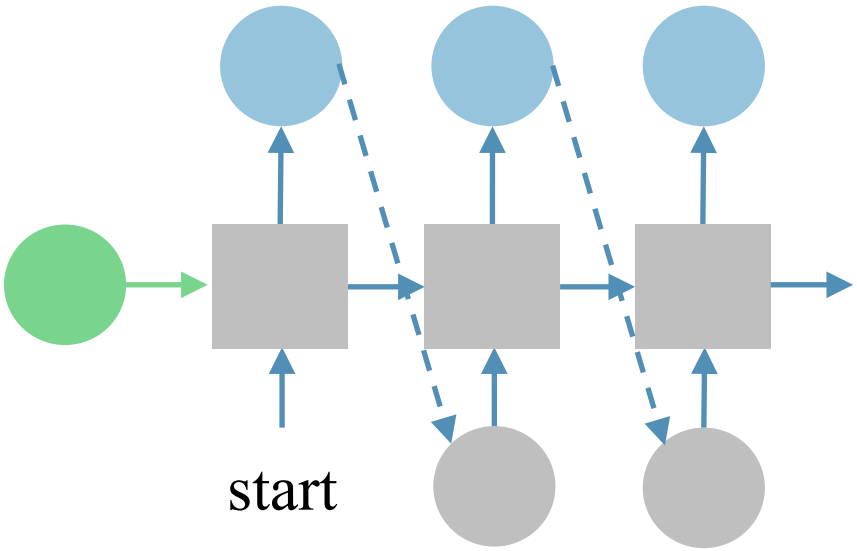
Alex Graves, https://arxiv.org/pdf/1308.0850.pdf

Handwriting generation



Alex Graves, https://arxiv.org/pdf/1308.0850.pdf

Conditional sequence generation



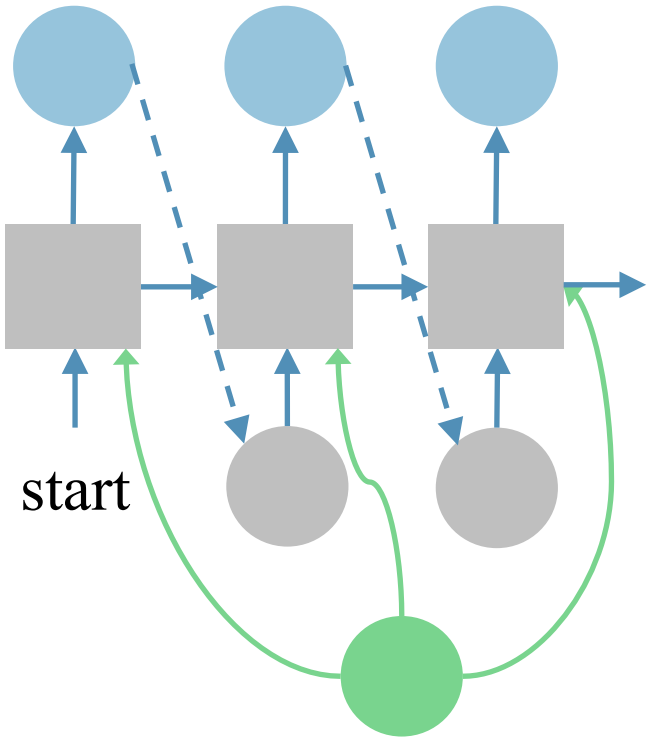
Input some object

Output sequence

Tasks

- Speech generation
- Handwriting generation
- Image captioning
- …

Conditional sequence generation



info about object can be used at each time step as well

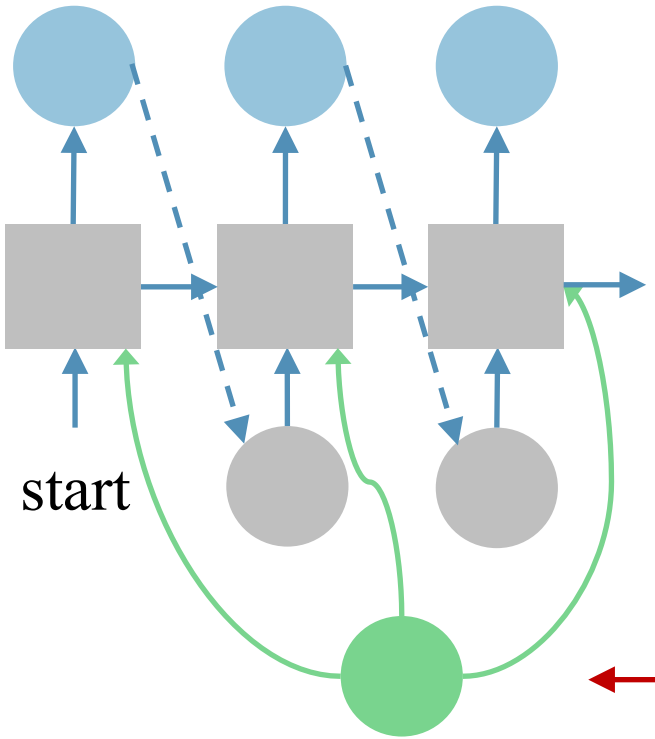
Input some object

Output sequence

Tasks

- Speech generation
- Handwriting generation
- Image captioning
- …

Conditional handwriting generation



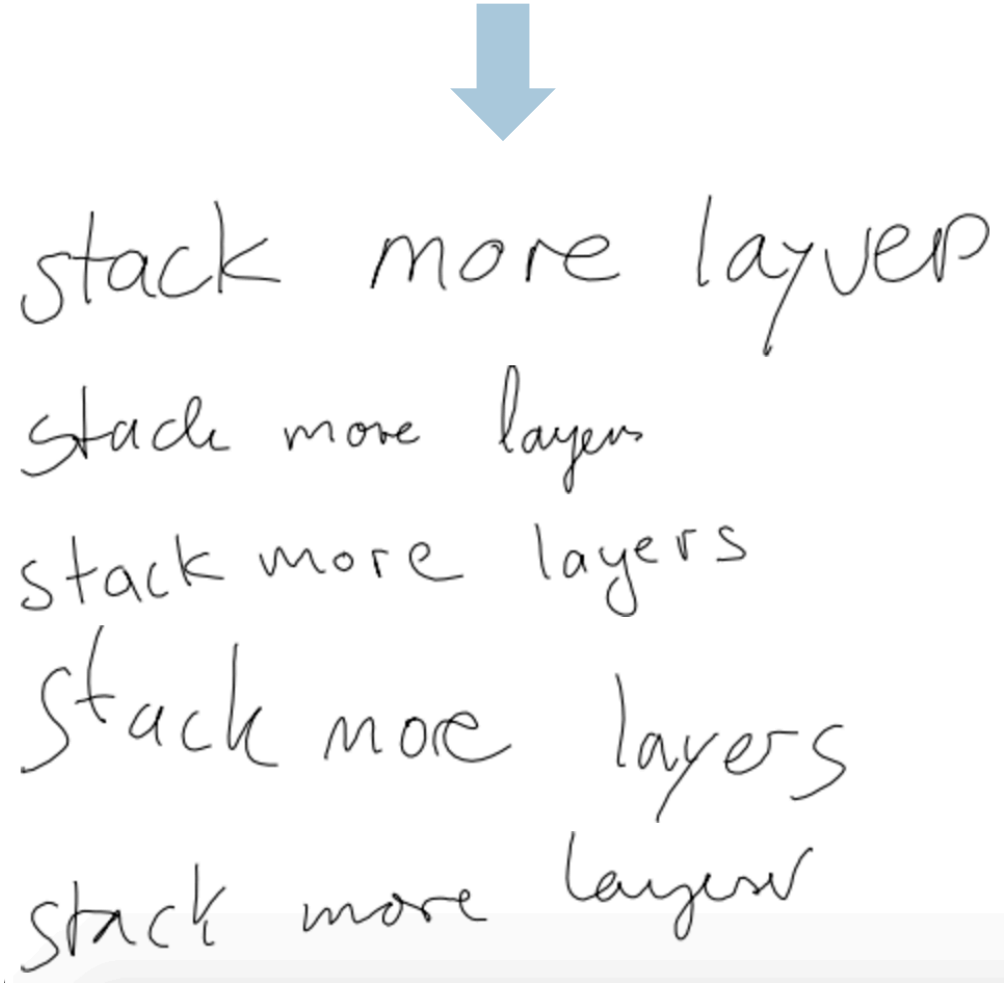
Next pen position

Current pen position

Window: which part of text we are writing now

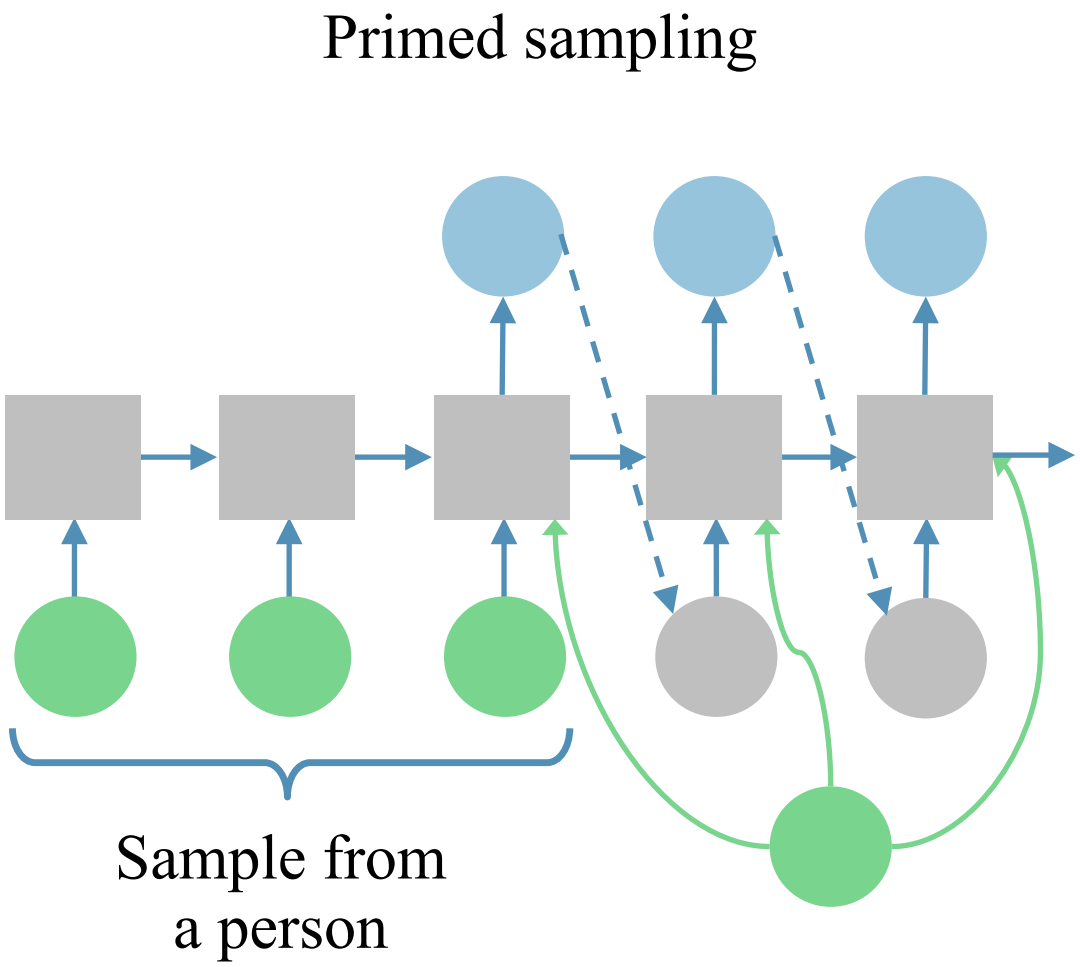
Conditional handwriting generation

stack more layers



Alex Graves, http://www.cs.toronto.edu/~graves/handwriting.html

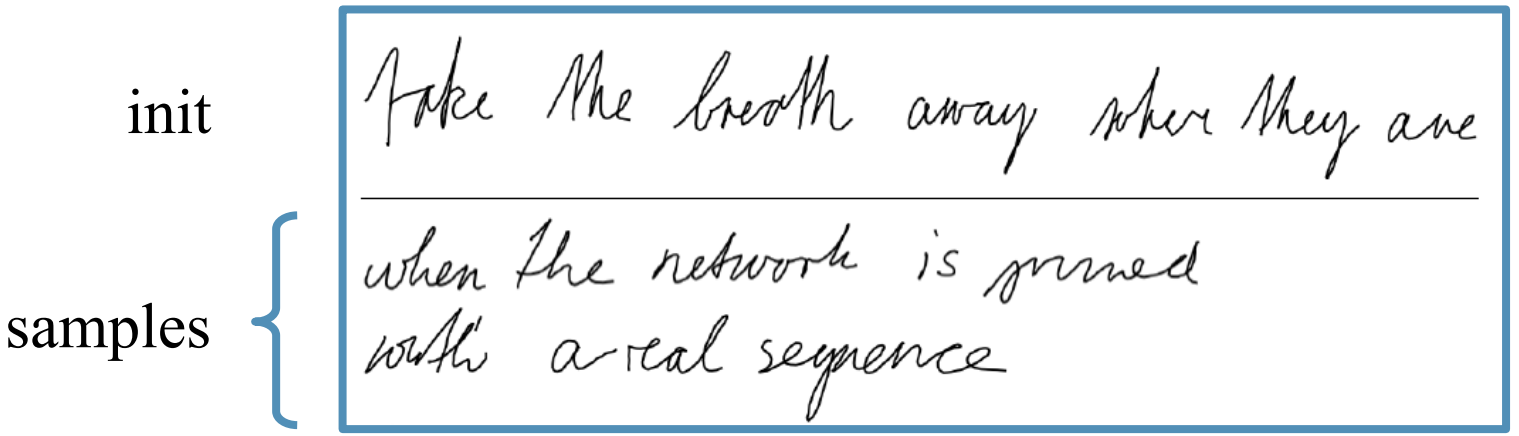
Conditional handwriting generation



to be confined to a particular style,
not just random

Conditional handwriting generation

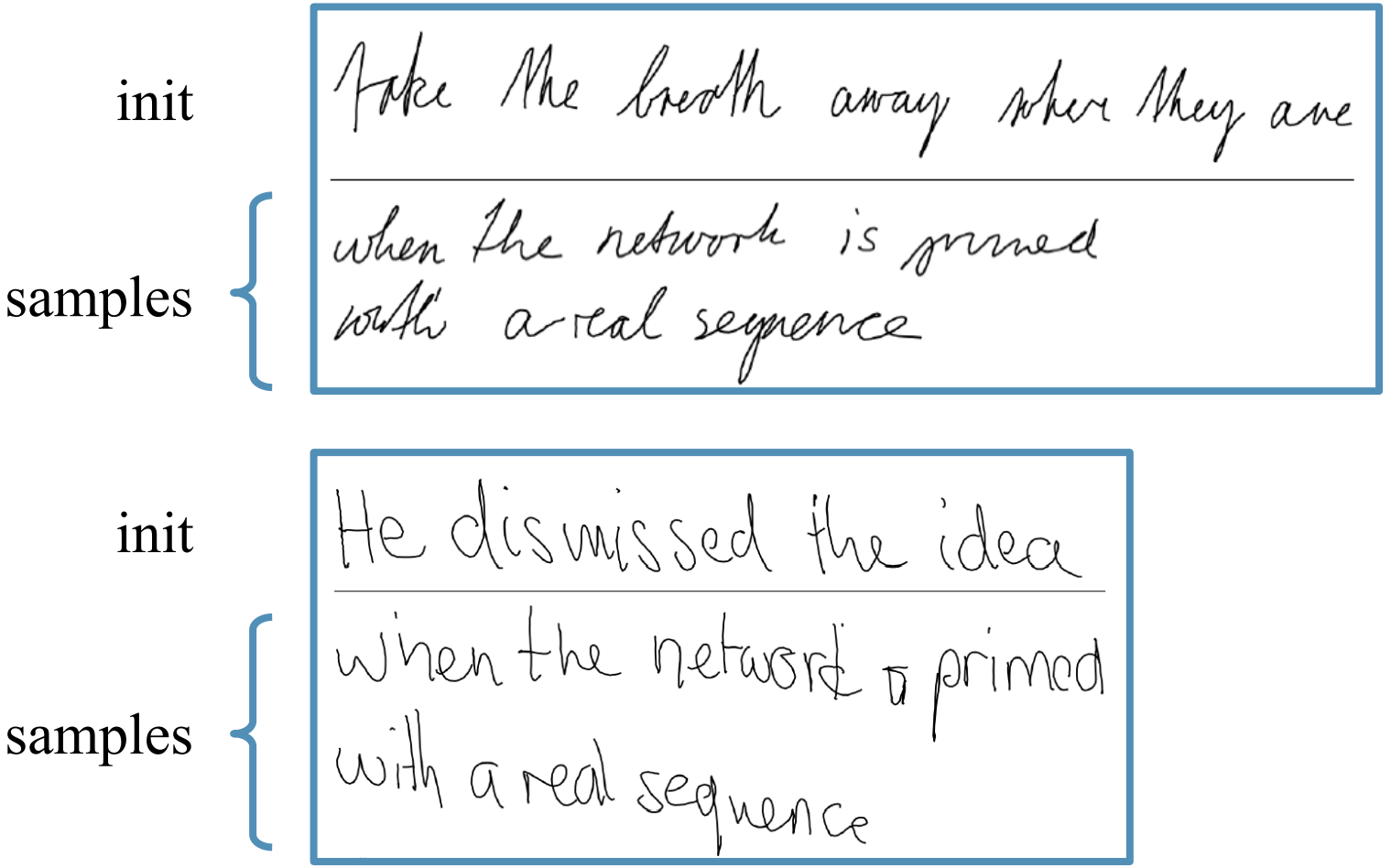
Primed sampling



Alex Graves, https://arxiv.org/pdf/1308.0850.pdf

Conditional handwriting generation

Primed sampling



Alex Graves, https://arxiv.org/pdf/1308.0850.pdf

Image Captioning

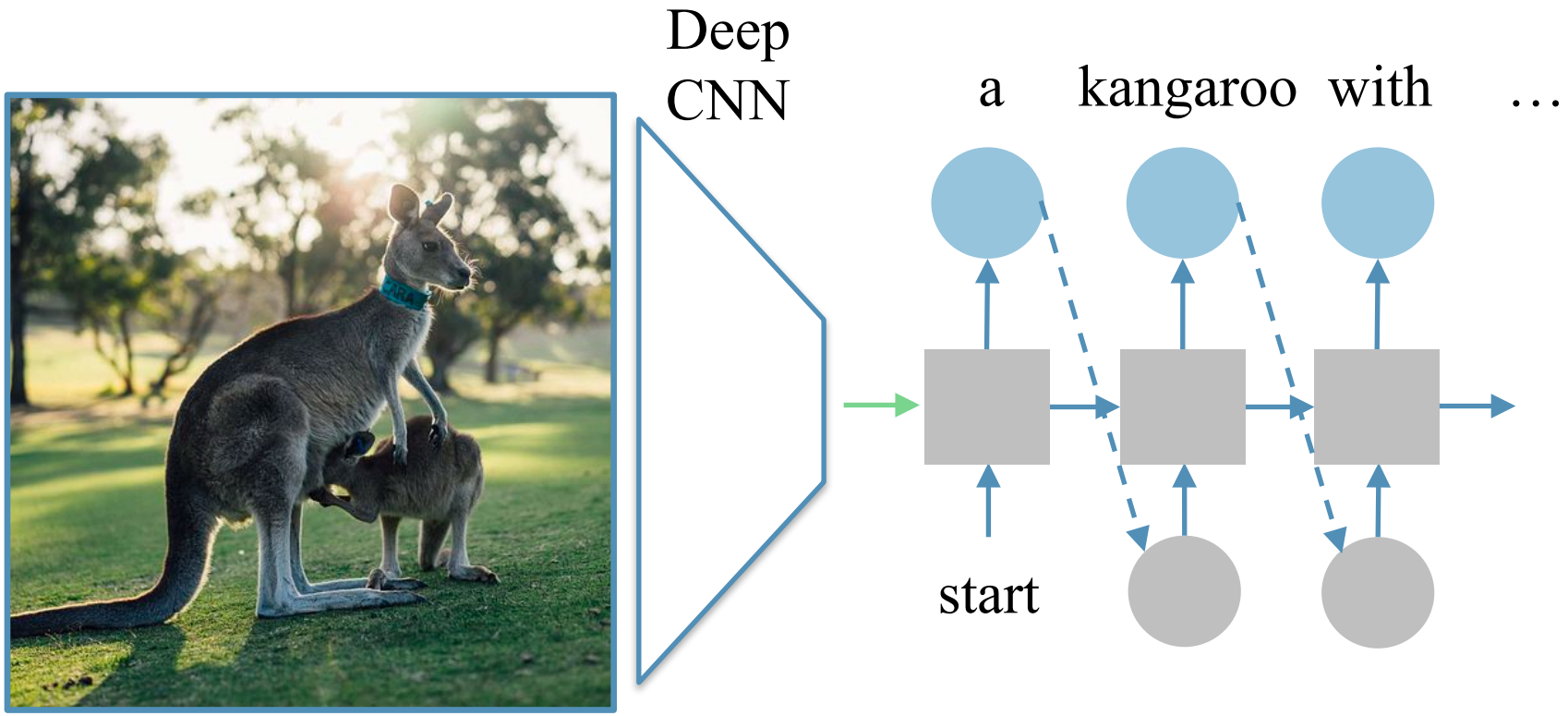


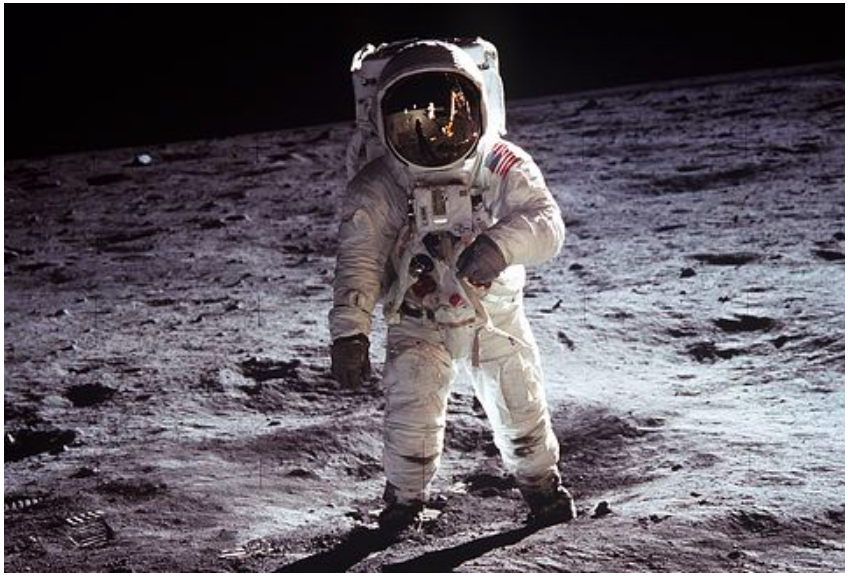
Image Captioning: good examples

a man riding a wave on a surfboard

a large brown bear walking across a river

Image Captioning: bad examples

a man riding on the back of a boat



a man riding a snowboard

Image Captioning: bad examples

?

a man is holding a kite in the air

Image Captioning: bad examples

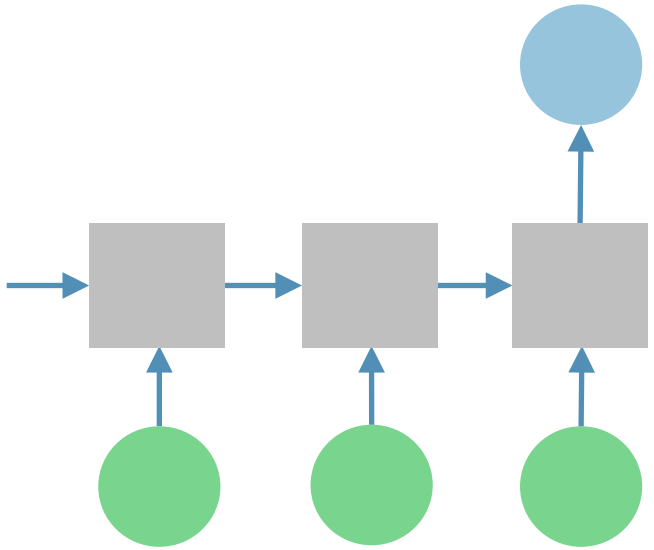
a man is holding a tennis racket on a tennis court

a man is holding a kite in the air

Sequence classification

Input sequence

Output one label



← One output at the end

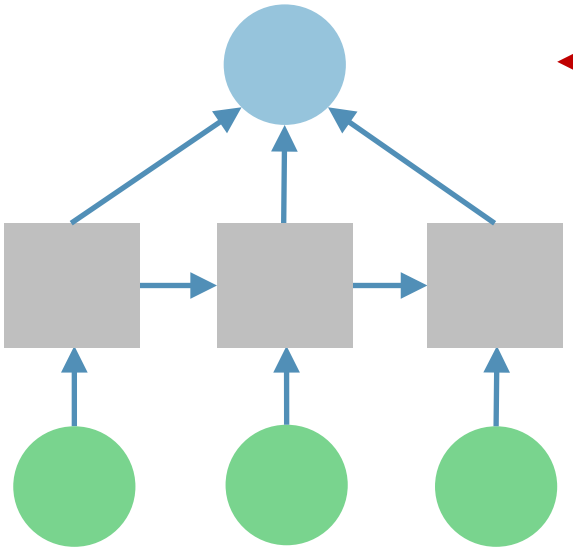
loss is calculated at the last time step

Tasks

- Sentiment analysis
- …

Sequence classification

Input sequence
Output one label



Dynamic mean or max pooling
+ Attention mechanism

Tasks

- Sentiment analysis
- …

Sequence translation

"Sequence-to-sequence"

Input sequence

Output sequence

Tasks

- Handwriting to text / text to handwriting
- Speech to text / text to speech
- Machine translation

Sequence translation

Input sequence
Output sequence

Tasks

- Handwriting to text / text to handwriting
- Speech to text / text to speech
- Machine translation

Input and output …

- are NOT synchronized
- may have different length
- may have different order

Sequence translation

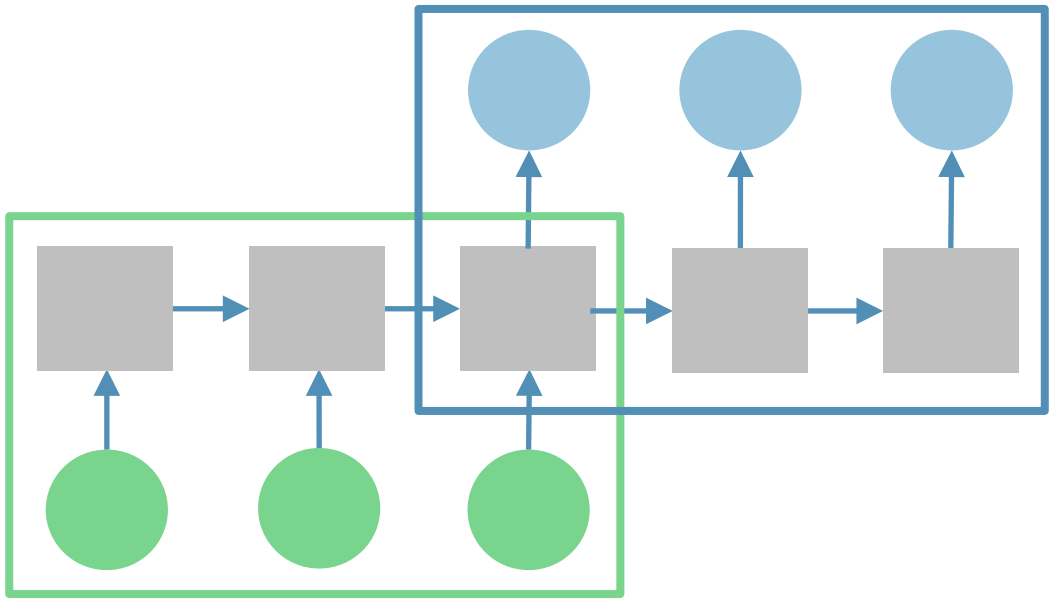
Input sequence

Output sequence

Tasks

- Machine translation

Input and output have different order



Sequence translation

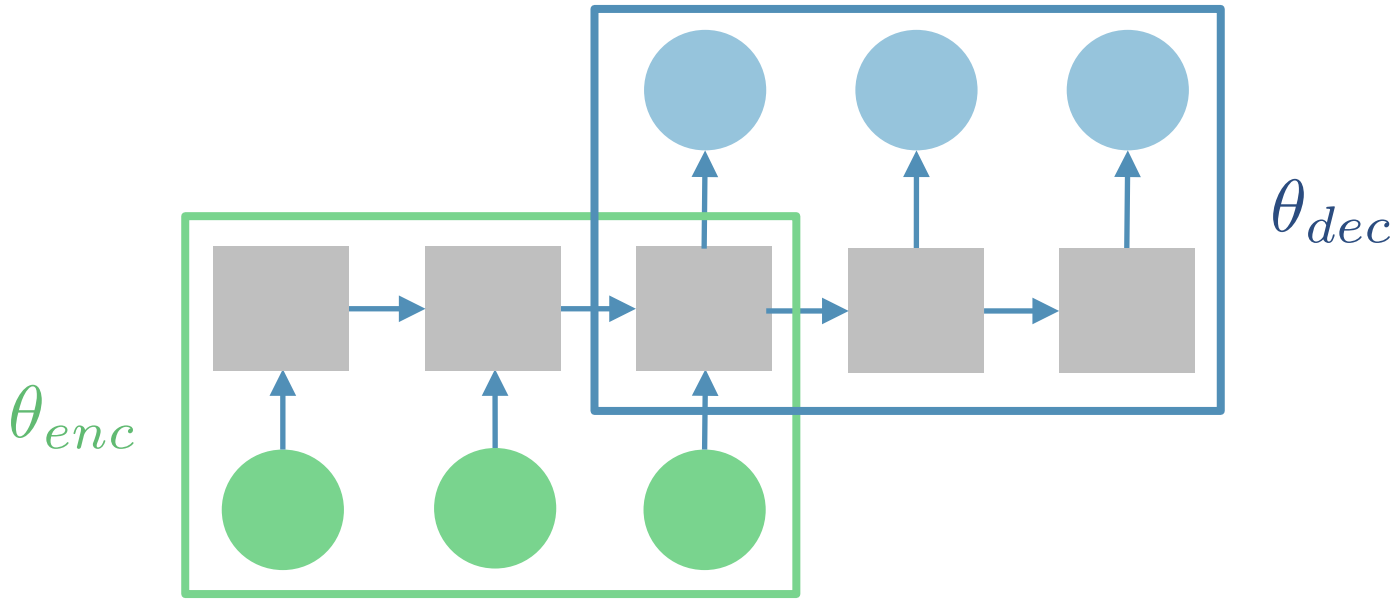
Input sequence

Output sequence

Tasks

- Machine translation

Input and output have different order



Summary

We have learned how to use recurrent networks for:

- Element-wise sequence classification
- Sequence generation: unconditional and conditional
- Sequence classification
- Sequence translation

Good luck with the final project!