Recurrent Neural Networks

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 24, 2020

Outline


FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Motivation


SIGILLUM ACADEMIAE FRIDERICO ALEXANDRINAE

Motivation

- So far: **One** input, e.g., single image
- **Feedforward** neural networks: input ➔ processing ➔ result

Motivation

- So far: **One** input, e.g., single image
- **Feedforward** neural networks: input ➔ processing ➔ result
- But: lots of **sequential** or **time-dependent** signals, e.g.
  - Speech/Music (translation, music classification)
  - Video (object detection/face recognition)
  - Sensor data (speed, temperature, energy consumption, ...)

Motivation

- So far: **One** input, e.g., single image
- **Feedforward** neural networks: input ➔ processing ➔ result
- But: lots of **sequential** or **time-dependent** signals, e.g.
  - Speech/Music (translation, music classification)
  - Video (object detection/face recognition)
  - Sensor data (speed, temperature, energy consumption, ...)
- “Snapshots” often not informative (single word ➔ translation?)

Motivation

- So far: **One** input, e.g., single image
- **Feedforward** neural networks: input ➔ processing ➔ result
- But: lots of **sequential** or **time-dependent** signals, e.g.
  - Speech/Music (translation, music classification)
  - Video (object detection/face recognition)
  - Sensor data (speed, temperature, energy consumption, ...)
- “Snapshots” often not informative (single word ➔ translation?)

➔ **Temporal context** is important!

Motivation (cont.)

- Question: How can we integrate this context in the network?

Motivation (cont.)

- Question: How can we integrate this context in the network?
- Simple approach: Feed the whole sequence to a big network ➔ **Bad idea!**[1]
  - Inefficient memory usage
  - Difficult / impossible to train
  - Difference between spatial and temporal dimensions?

[1] Well... Link: Gehring et al.: A novel approach to neural machine translation [6]

Motivation (cont.)

- Question: How can we integrate this context in the network?
- Simple approach: Feed the whole sequence to a big network ➔ **Bad idea!**[1]
  - Inefficient memory usage
  - Difficult / impossible to train
  - Difference between spatial and temporal dimensions?
  - **Not real-time!** (translation, ...)

[1] Well... Link: Gehring et al.: A novel approach to neural machine translation [6]

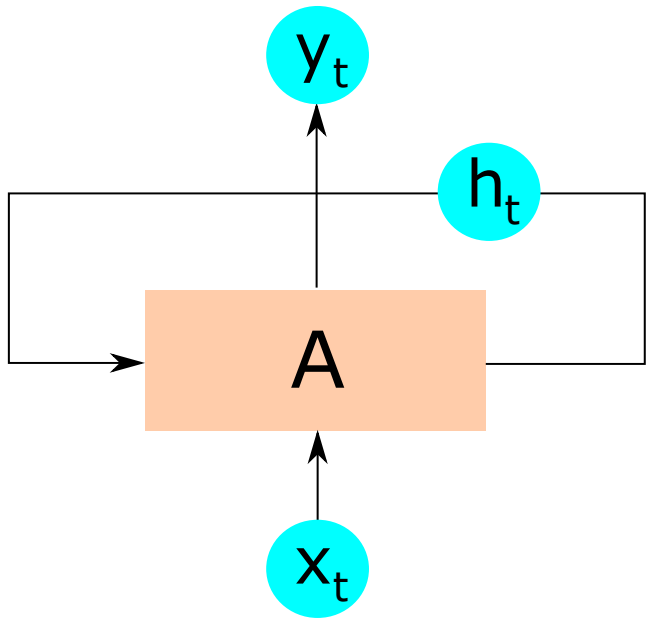
Motivation (cont.)

- Question: How can we integrate this context in the network?
- Simple approach: Feed the whole sequence to a big network ➔ **Bad idea!**[1]
  - Inefficient memory usage
  - Difficult / impossible to train
  - Difference between spatial and temporal dimensions?
  - **Not real-time!** (translation, ...)
- Better approach: Model sequential behavior within the architecture:
  ➔ **Recurrent neural networks (RNNs)**

[1] Well... Link: Gehring et al.: A novel approach to neural machine translation [6]

Simple Recurrent Networks

Simple Recurrent Neural Networks



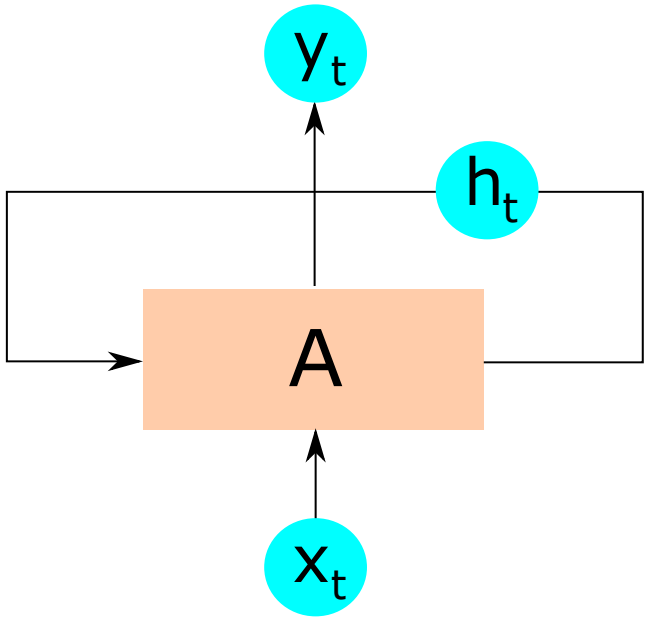
Simple RNN unit

- First models in 1970's [11] and early 1980's [10] (Hopfield Network)
- **Simple recurrent neural network** or **Elman network** introduced in *Finding Structure in Time* by Jeff Elman in 1990 [5]

Difference between Recurrent Neural Networks and Feedforward Neural Networks

- Feedforward networks only **feed** information **forward**
- With recurrent neural networks, we can:
  - model loops
  - model memory and experience
  - learn sequential relationships
  - provide continuous predictions as data comes in ➔ real-time

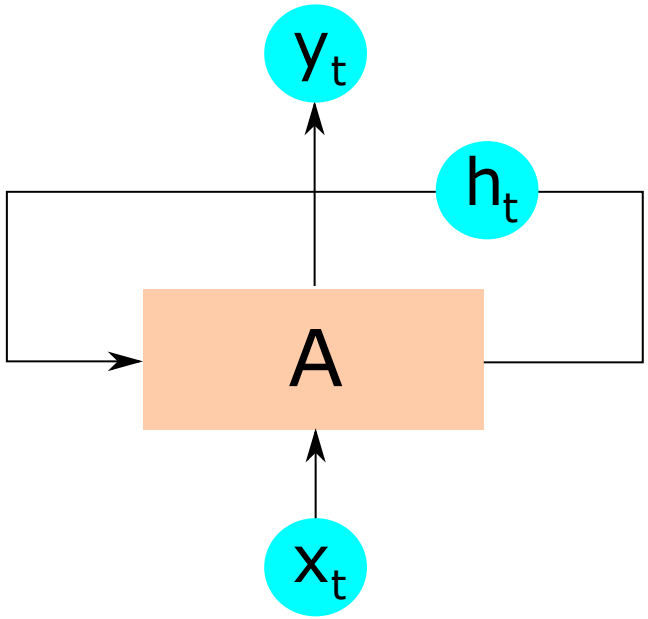
Basic Structure of RNNs



Simple RNN unit

- Current input x_t multiplied by weight

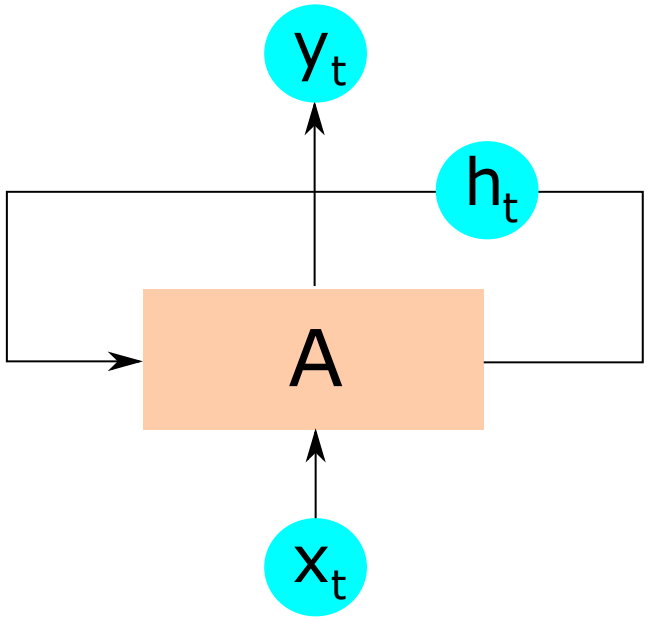
Basic Structure of RNNs



Simple RNN unit

- Current input x_t multiplied by weight
- **Additional input**: **Hidden state** h_{t-1} of the unit

Basic Structure of RNNs



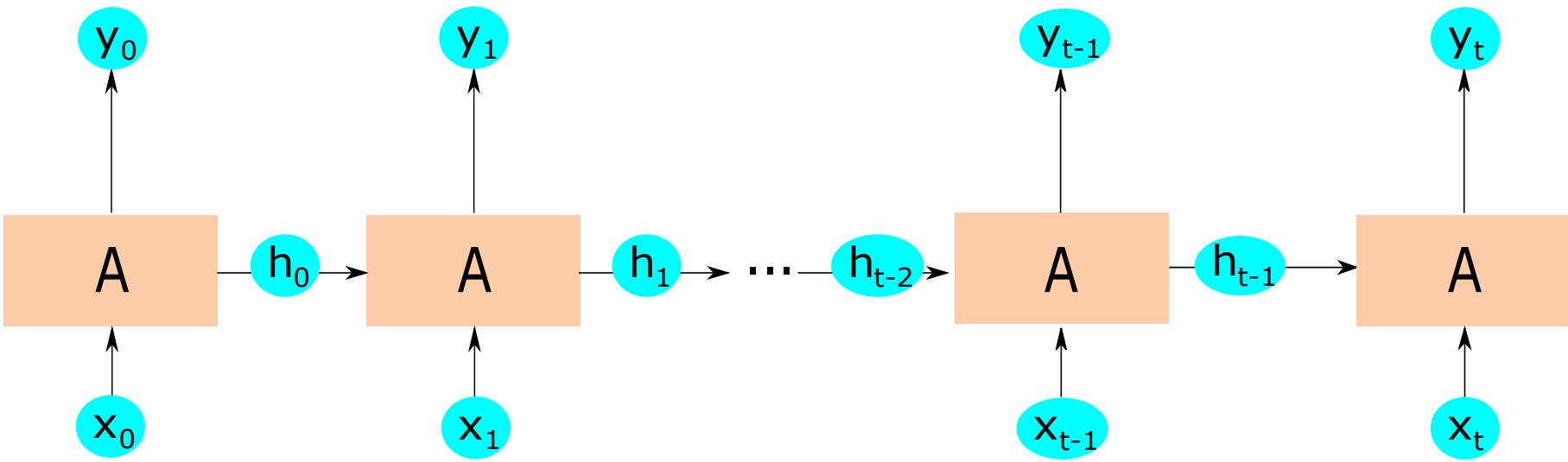
Simple RNN unit

- Current input x_t multiplied by weight
- **Additional input**: **Hidden state** h_{t-1} of the unit

➔ Feedback loop: use information from present *and* recent past to compute output y_t

Basic Structure of RNNs (cont.)

- "Unfolded" RNN unit: sequence of copies of the **same** unit (= same weights)
- Each unit passes hidden state as additional input to successor

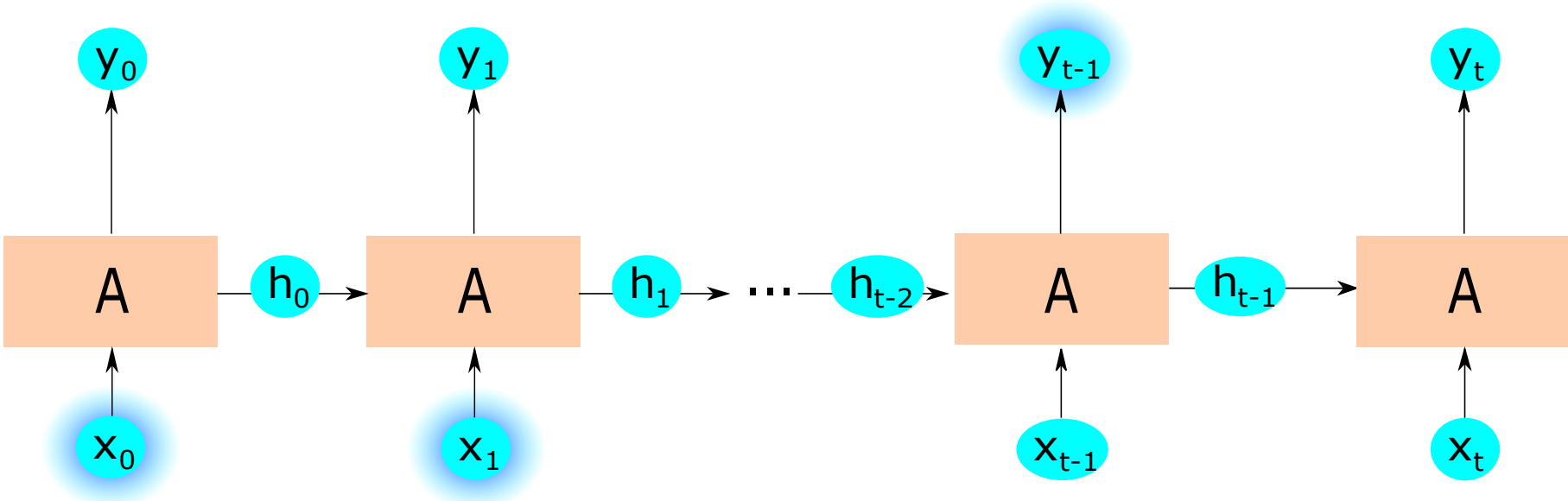


Basic RNN unfolded

Basic Structure of RNNs (cont.)

- "Unfolded" RNN unit: sequence of copies of the **same** unit (= same weights)
- Each unit passes hidden state as additional input to successor

➔ Previous input can influence current output



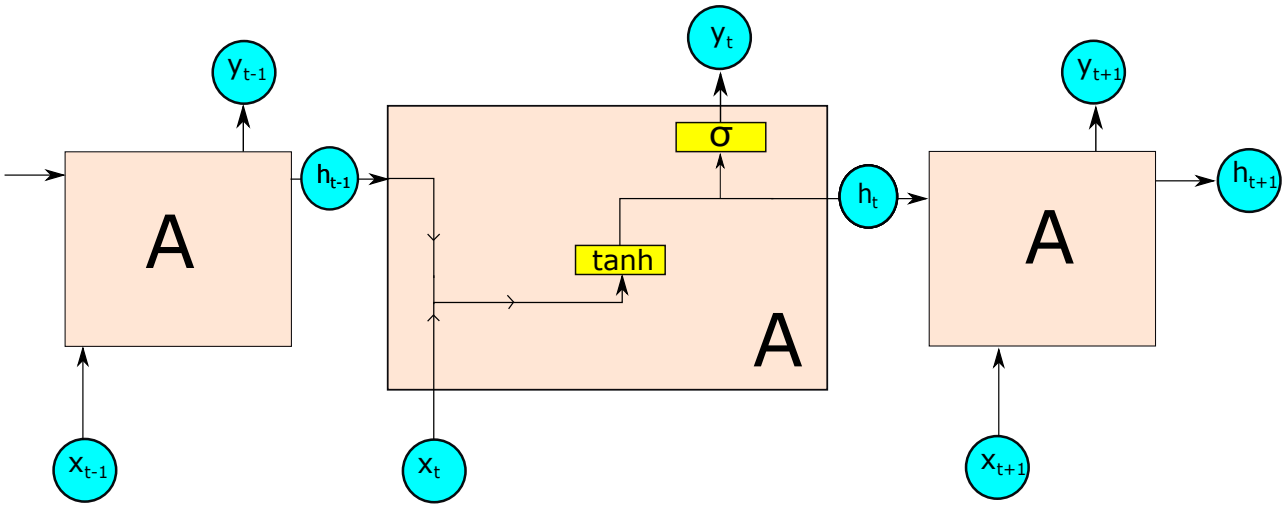
Basic RNN unfolded

Close-up of a basic RNN unit

- Question 1: How do we update the hidden state?
- Question 2: How do we combine input and hidden state to compute output?

Close-up of a basic RNN unit

- Question 1: How do we update the hidden state?
- Question 2: How do we combine input and hidden state to compute output?

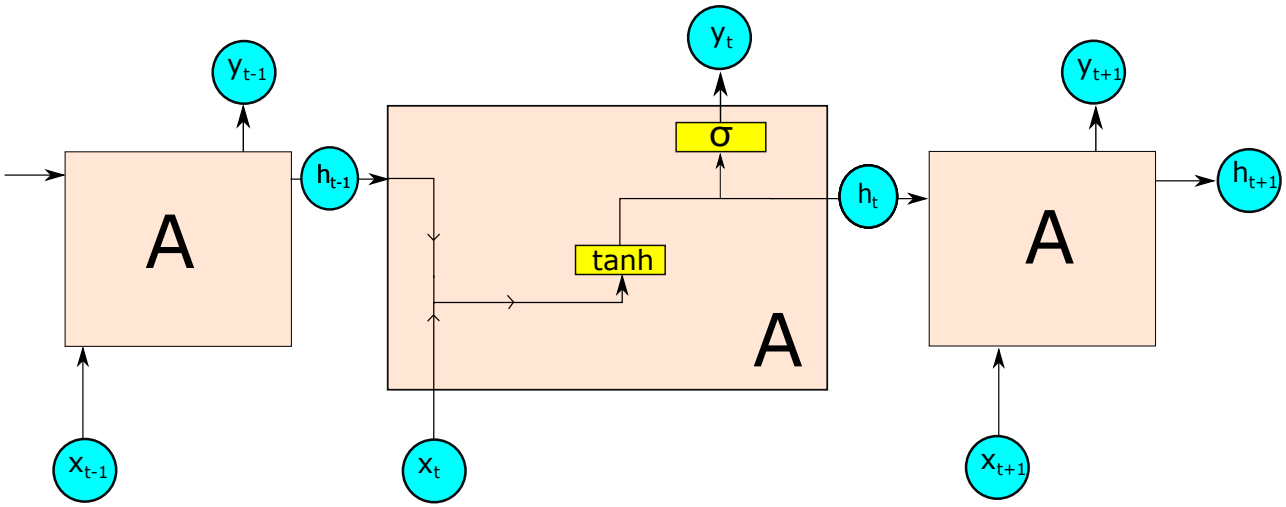


Two activation functions:

- tanh: Combination of previous state and current input

Close-up of a basic RNN unit

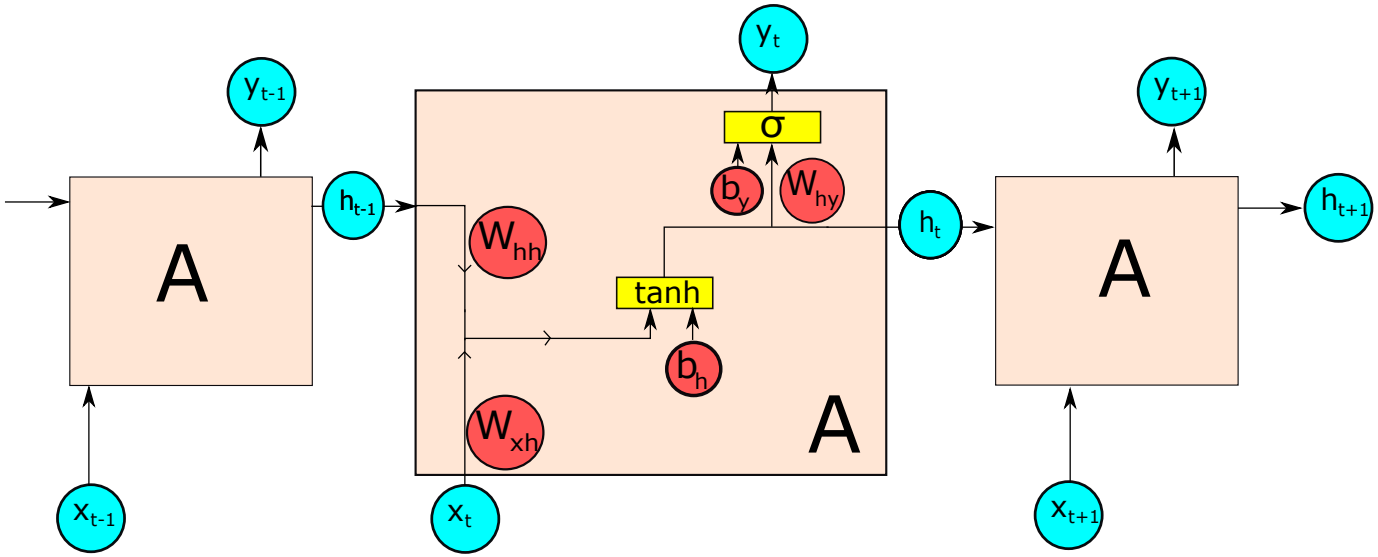
- Question 1: How do we update the hidden state?
- Question 2: How do we combine input and hidden state to compute output?



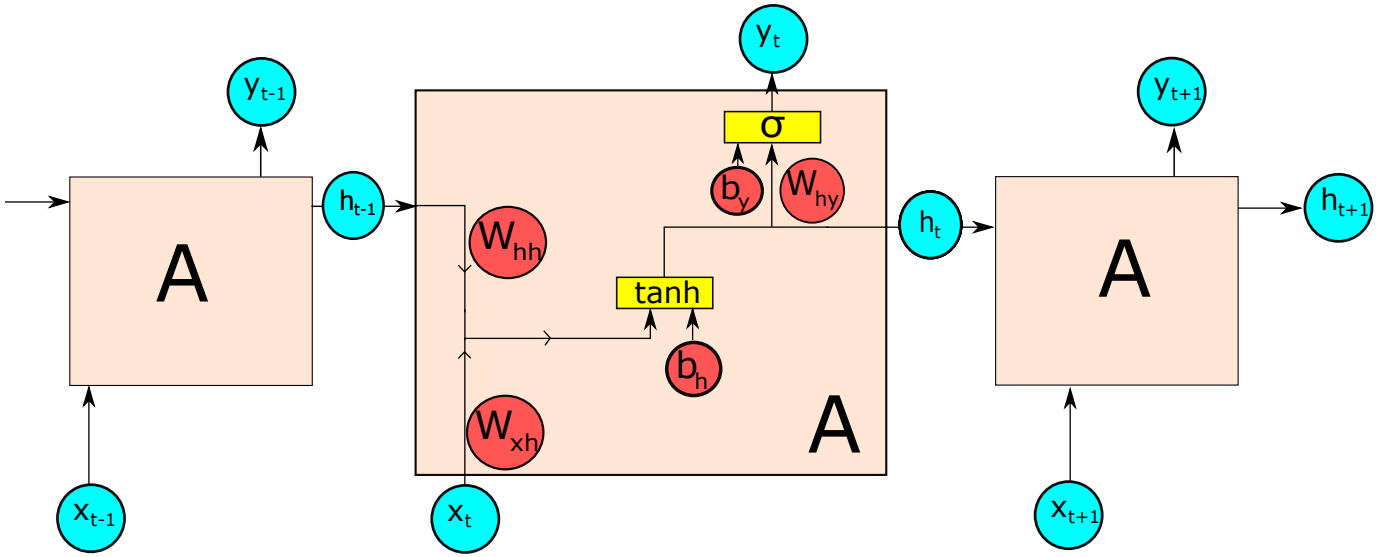
Two activation functions:

- tanh: Combination of previous state and current input
- σ: Additional non-linearity for output

Closer-up: How to update the hidden state?



Closer-up: How to update the hidden state?



Update hidden state:

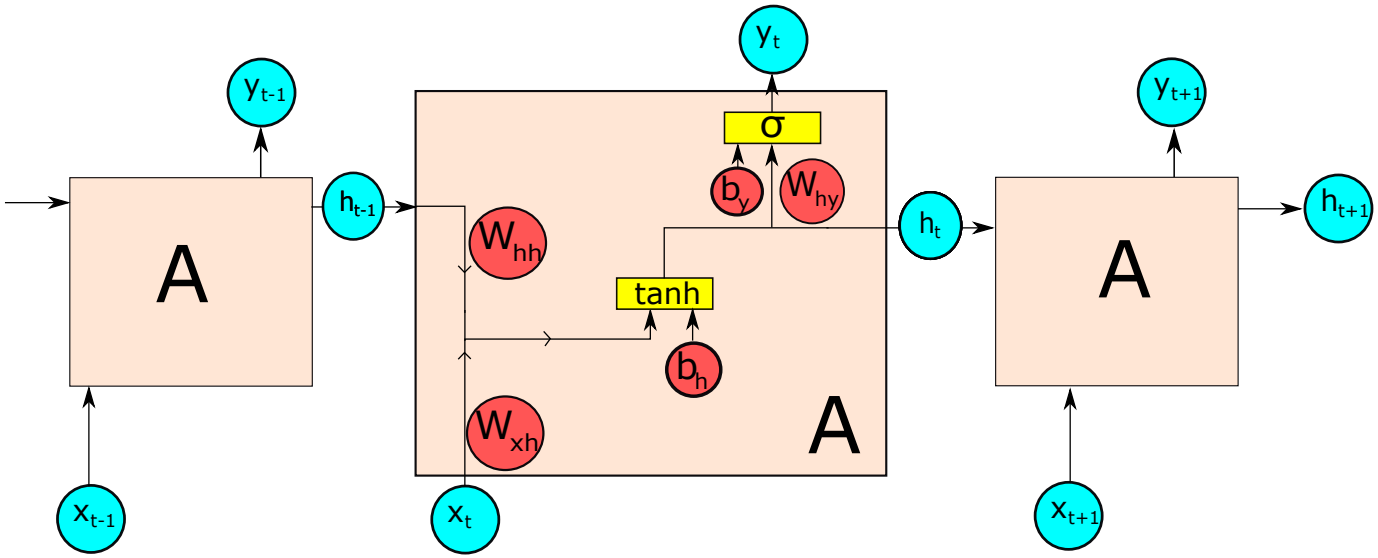
$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h\right)$$

$\mathbf{W}_{hh}$: Weight matrix for previous hidden state $\mathbf{h}_{t-1}$

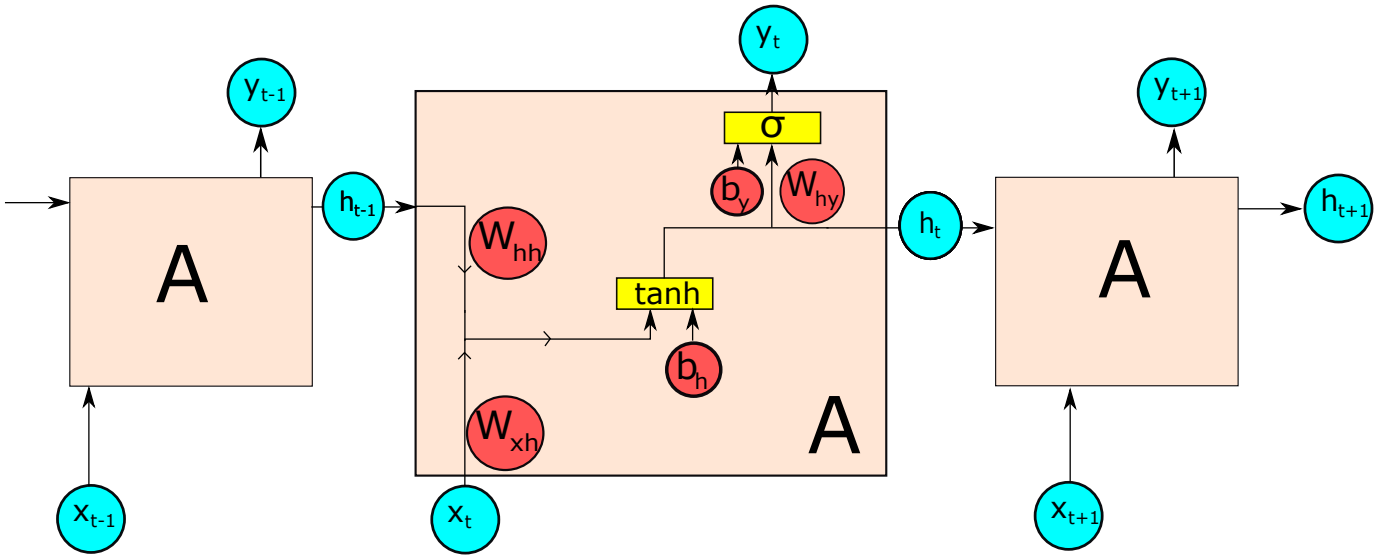
$\mathbf{W}_{xh}$: Weight matrix for current input $\mathbf{x}_t$

$\mathbf{b}_h$: Update bias

Closer-up: How to compute the output?



Closer-up: How to compute the output?



Output formula:

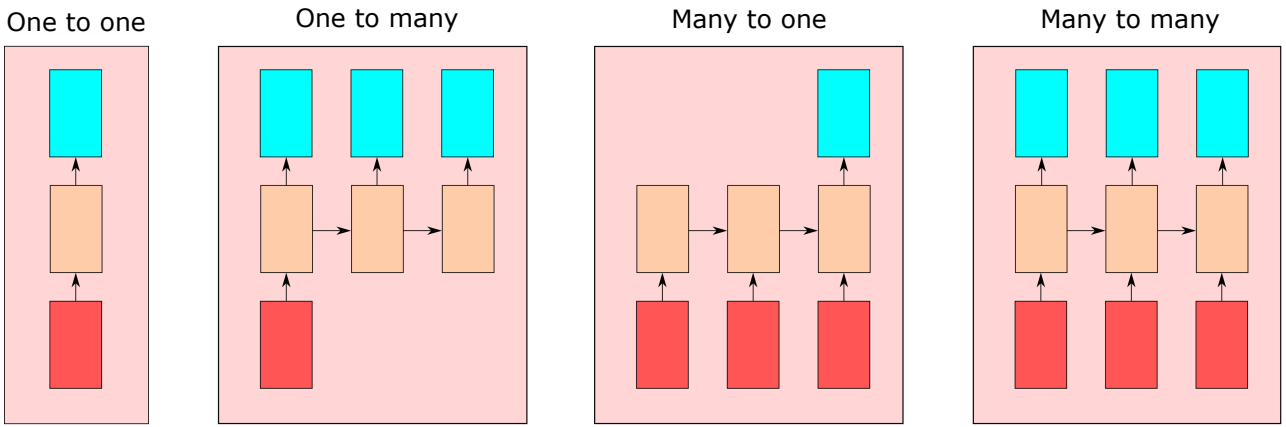
$$\mathbf{y}_t = \sigma\left(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y\right)$$

$\mathbf{W}_{hy}$: Weight matrix for current hidden state $\mathbf{h}_t$

$\mathbf{b}_h$: Output bias

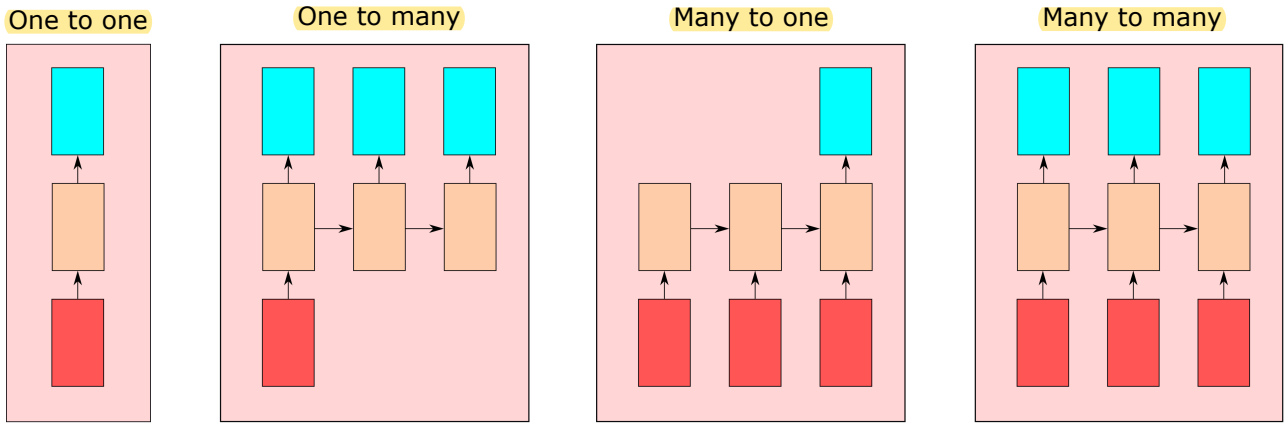
RNN Basic Architectures

Type of sequential relationship $\leftrightarrow$ architecture:



RNN Basic Architectures

Type of sequential relationship $\leftrightarrow$ architecture:



- Examples:
  - One to one: Image classification (classic feed-forward)
  - One to many: Image captioning
  - Many to one: Sentiment analysis
  - Many to many: Video classification

Deep RNNs

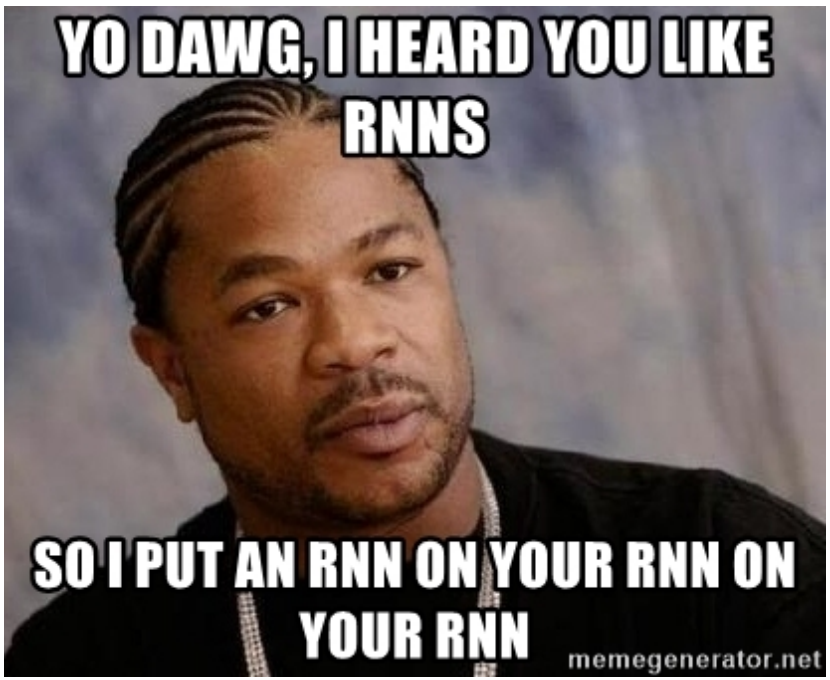
- So far, only one hidden layer

Deep RNNs

- So far, only one hidden layer
- Recurring (ha!) motto:

Deep RNNs

- So far, only one hidden layer
- Recurring (ha!) motto:



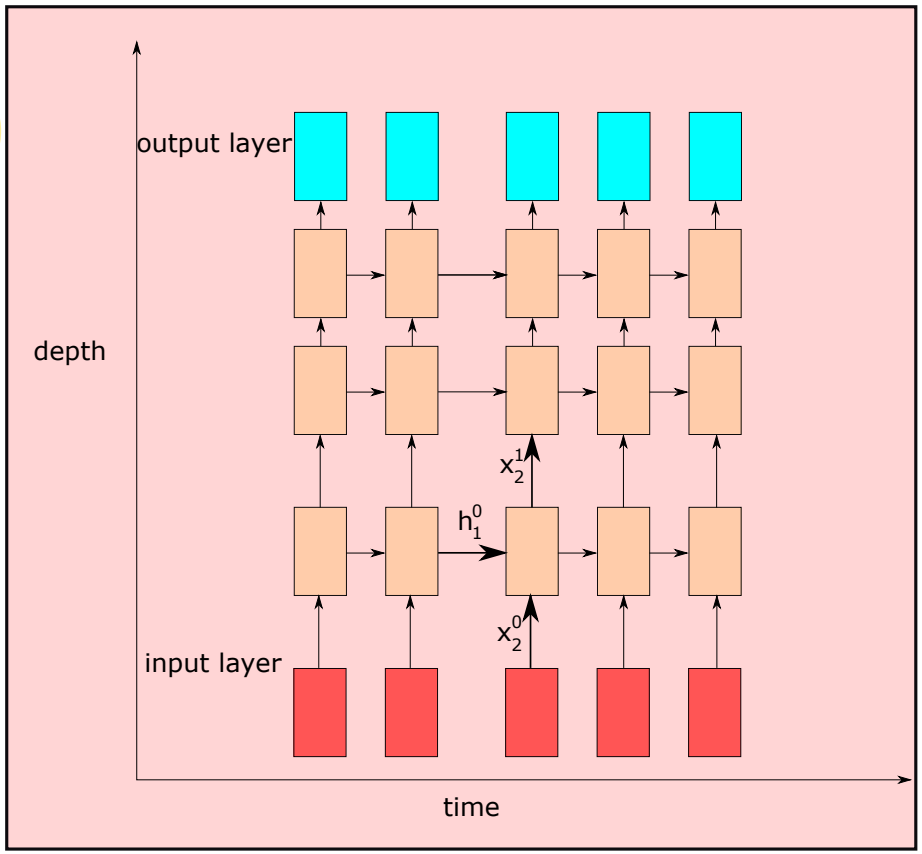
Deep RNNs (cont.)

Similar to CNNs, stack multiple units for **deep RNNs**

$$\mathbf{h}_t^l = \tanh\left(\mathbf{W}_{xh}^l \cdot \mathbf{x}_t^l + \mathbf{W}_{hh}^l \cdot \mathbf{h}_{t-1}^l + \mathbf{b}^l\right)$$

t: time point

l: layer index



NEXT TIME

ON DEEP LEARNING

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 2

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 24, 2020

Simple Example: Character Level Language Model

Task: Learn character probability distribution from input text

- Vocabulary of $\{h, e, l, o\}$
- Characters encoded as one-hot vectors, e.g., $h = (1, 0, 0, 0)$

Simple Example: Character Level Language Model

Task: Learn character probability distribution from input text

- Vocabulary of $\{h, e, l, o\}$
- Characters encoded as one-hot vectors, e.g., $h = (1, 0, 0, 0)$
- Train RNN on the sequence "*hello*":
  Given '*h*' as first input, the network should generate sequence "*hello*"

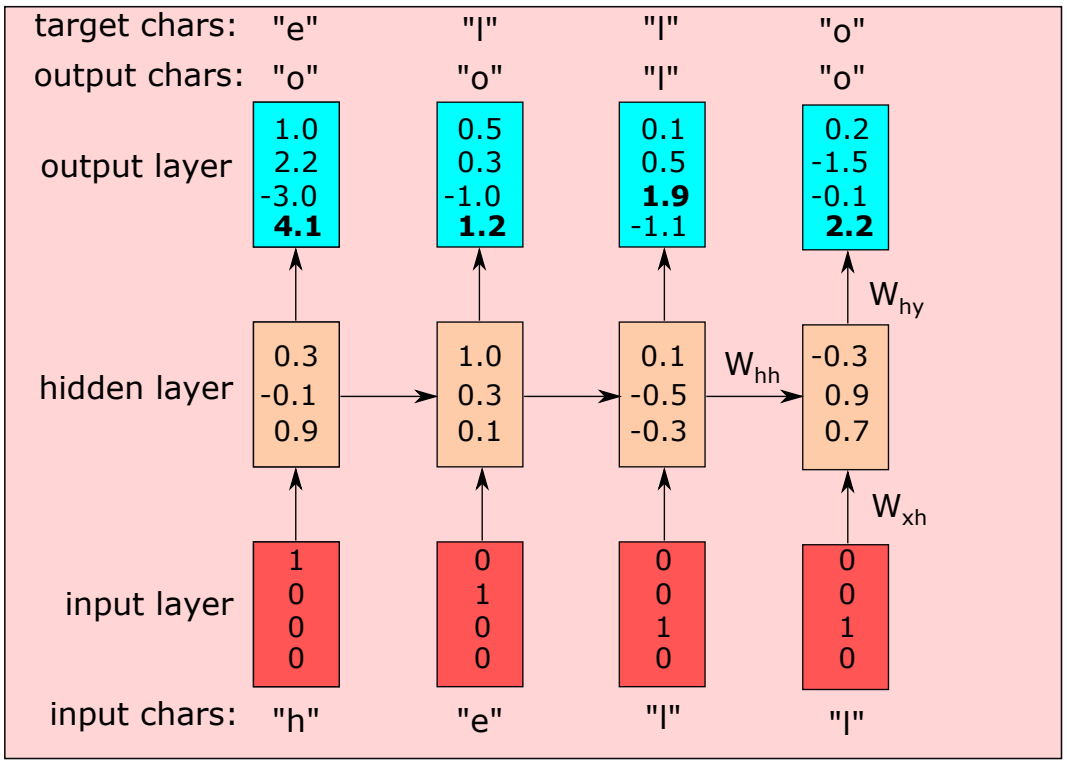
Simple Example: Character Level Language Model

Task: Learn character probability distribution from input text

- Vocabulary of $\{h, e, l, o\}$
- Characters encoded as one-hot vectors, e.g., $h = (1, 0, 0, 0)$
- Train RNN on the sequence "*hello*":
  Given '*h*' as first input, the network should generate sequence "*hello*"
- Network needs to know **previous inputs** when presented with '*l*':
  Do we need another '*l*' or an '*o*'?

Simple Example: Character Level Language Model (cont.)

Prediction with random initialization:

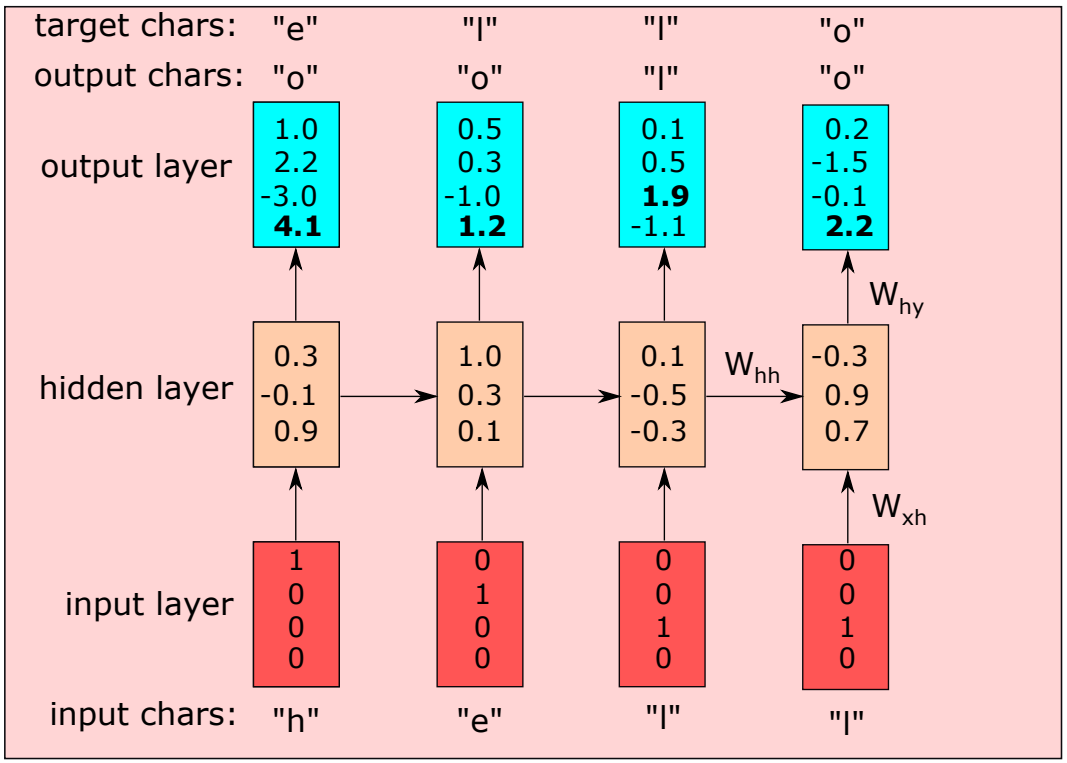


- Goal: Maximize prediction for correct component
- How can we now train this network?

Source: Adapted from http://karpathy.github.io/2015/05/21/rnn-effectiveness

Simple Example: Character Level Language Model (cont.)

Prediction with random initialization:



- Goal: Maximize prediction for correct component
- How can we now train this network?

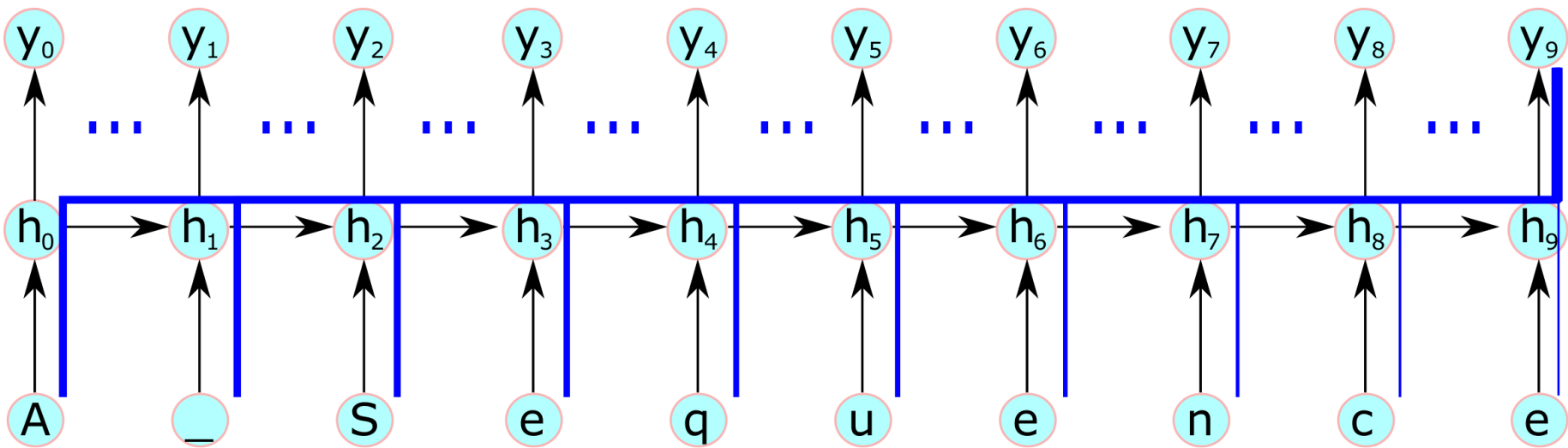
➔ **Backpropagation through time (BPTT)**: train “unfolded” network

Source: Adapted from http://karpathy.github.io/2015/05/21/rnn-effectiveness

Backpropagation through Time (BPTT)

Concept: Train the unfolded network

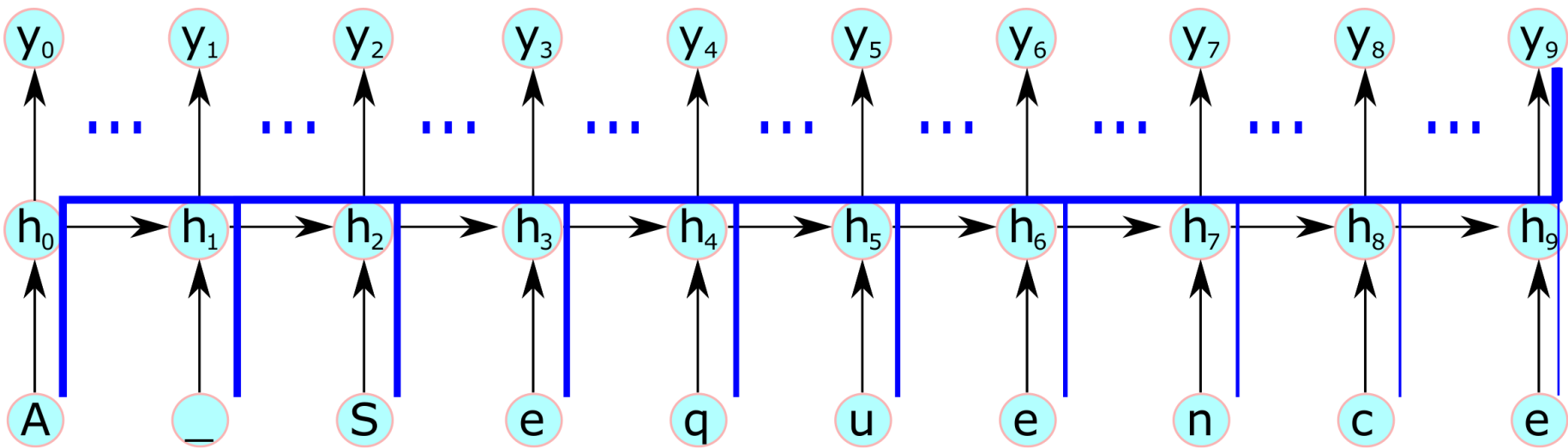
- Compute the forward pass for the full sequence ➔ loss



Backpropagation through Time (BPTT)

Concept: Train the unfolded network

- Compute the forward pass for the full sequence → loss
- Compute backward pass through full sequence to get gradients → weight update

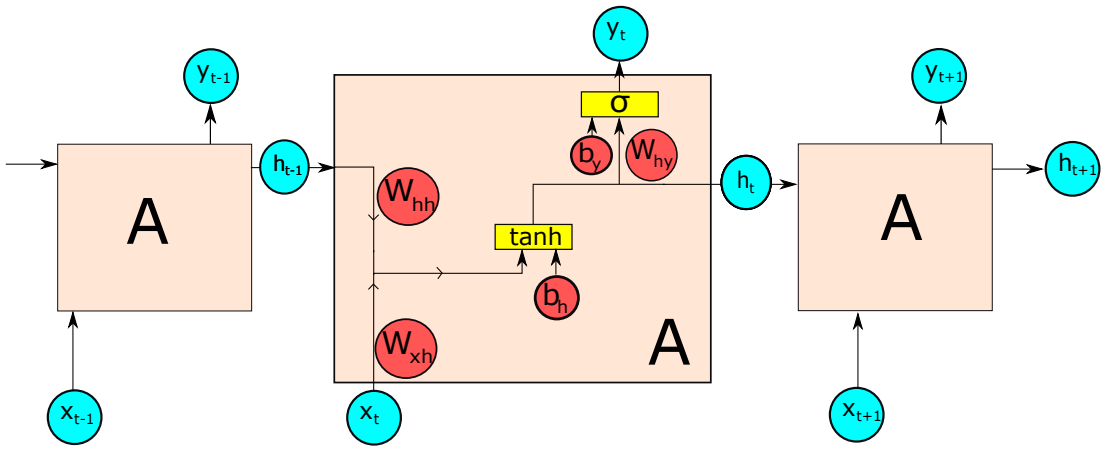


Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output

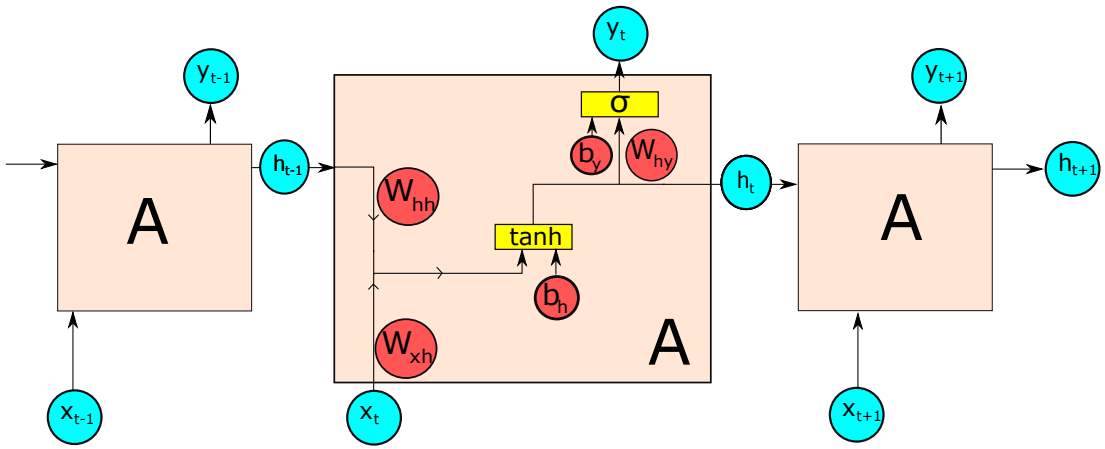


Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$

1: **for** t **from** 1 **to** T **do**:

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



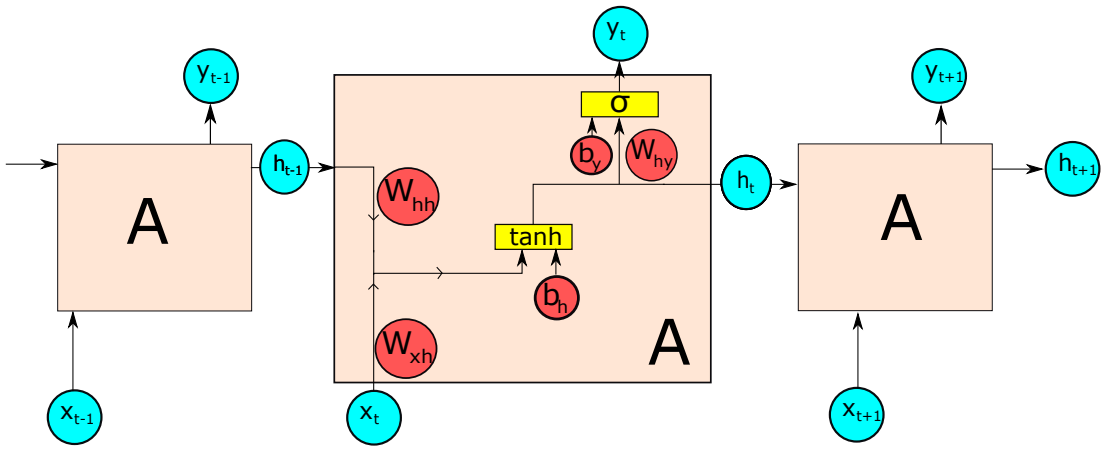
Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$

1: **for** t **from** 1 **to** T **do**:

2: $\quad \mathbf{u}_t = \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h$

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$

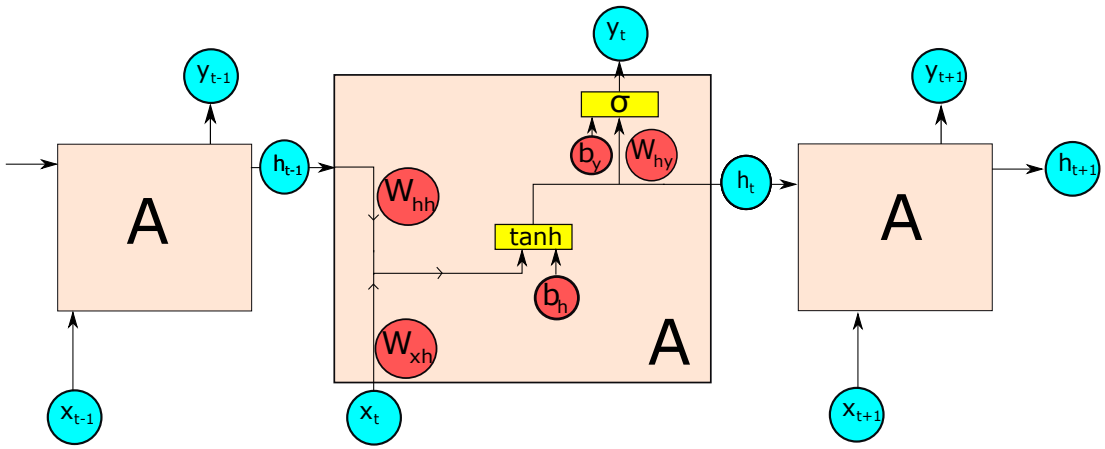
1: **for** t **from** 1 **to** T **do**:

2: $\quad \mathbf{u}_t = \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h$

3: $\quad \mathbf{h}_t = \tanh(\mathbf{u}_t)$

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$

1: **for** t **from** 1 **to** T **do**:

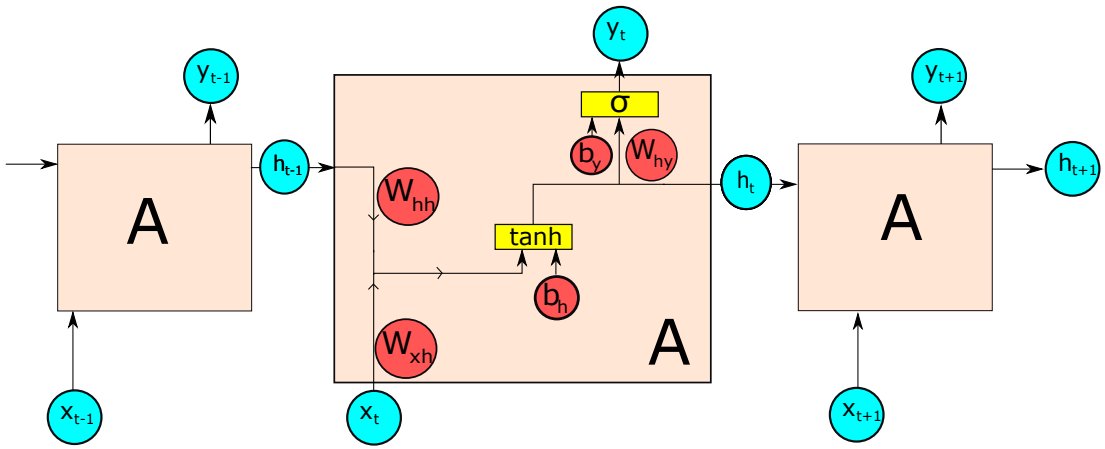
2: $\quad \mathbf{u}_t = \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h$

3: $\quad \mathbf{h}_t = \tanh(\mathbf{u}_t)$

4: $\quad \mathbf{o}_t = \mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y$

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$

1: **for** t **from** 1 **to** T **do**:

2: $\mathbf{u}_t = \mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h$

3: $\mathbf{h}_t = \tanh(\mathbf{u}_t)$

4: $\mathbf{o}_t = \mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y$

5: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t)$

Backpropagation through Time (BPTT) (cont.)

- Loss function, e.g., cross-entropy loss: $L(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$
  - $\hat{\mathbf{y}}$: predicted output
  - $\mathbf{y}$: ground truth
- Compute gradient of the loss function

$$\nabla\theta = [\nabla \mathbf{W}_{xh}, \nabla \mathbf{W}_{hh}, \nabla \mathbf{W}_{hy}, \nabla \mathbf{b}_h, \nabla \mathbf{b}_y, \nabla \mathbf{h}_0]$$

- Update parameters using a learning rate η

$$\theta = \theta - \eta \nabla\theta$$

Backpropagation through Time (BPTT) (cont.)

- Loss function, e.g., cross-entropy loss: $L(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} L(\hat{\mathbf{y}}_t, \mathbf{y}_t)$
  - $\hat{\mathbf{y}}$: predicted output
  - $\mathbf{y}$: ground truth
- Compute gradient of the loss function

$$\nabla\theta = [\nabla \mathbf{W}_{xh}, \nabla \mathbf{W}_{hh}, \nabla \mathbf{W}_{hy}, \nabla \mathbf{b}_h, \nabla \mathbf{b}_y, \nabla \mathbf{h}_0]$$

- Update parameters using a learning rate η

$$\theta = \theta - \eta \nabla\theta$$

- Question: How do we get these derivatives?
- Go “back in time” through the network

Backpropagation through Time (BPTT) (cont.)

Go “backwards” through the unfolded unit, starting at final time step $t = T$ and iteratively compute gradients for $t = T, \ldots, 1$

Reminder: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t) = \sigma(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$

$$\nabla \mathbf{o}_t = \sigma'(\mathbf{o}_t) \cdot \frac{\partial L}{\partial \hat{\mathbf{y}}_t}(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

$$\nabla \mathbf{W}_{hy,t} = \nabla \mathbf{o}_t \mathbf{h}_t^\mathsf{T}$$

$$\nabla \mathbf{b}_{y,t} = \nabla \mathbf{o}_t$$

Backpropagation through Time (BPTT) (cont.)

Go “backwards” through the unfolded unit, starting at final time step $t = T$ and iteratively compute gradients for $t = T, \ldots, 1$

Reminder: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t) = \sigma(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$

$$\nabla \mathbf{o}_t = \sigma'(\mathbf{o}_t) \cdot \frac{\partial L}{\partial \hat{\mathbf{y}}_t}(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

$$\nabla \mathbf{W}_{hy,t} = \nabla \mathbf{o}_t \mathbf{h}_t^\mathsf{T}$$

$$\nabla \mathbf{b}_{y,t} = \nabla \mathbf{o}_t$$

The gradient $\nabla \mathbf{h}_t$ depends on two elements - the hidden state influences $\mathbf{o}_t$ *and* the next hidden state $\mathbf{h}_{t+1}$:

Backpropagation through Time (BPTT) (cont.)

Go “backwards” through the unfolded unit, starting at final time step $t = T$ and iteratively compute gradients for $t = T, \ldots, 1$

Reminder: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t) = \sigma(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$

$$\nabla \mathbf{o}_t = \sigma'(\mathbf{o}_t) \cdot \frac{\partial L}{\partial \hat{\mathbf{y}}_t}(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

$$\nabla \mathbf{W}_{hy,t} = \nabla \mathbf{o}_t \mathbf{h}_t^\mathsf{T}$$

$$\nabla \mathbf{b}_{y,t} = \nabla \mathbf{o}_t$$

The gradient $\nabla \mathbf{h}_t$ depends on two elements - the hidden state influences $\mathbf{o}_t$ *and* the next hidden state $\mathbf{h}_{t+1}$:

$$\nabla \mathbf{h}_t = \left(\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right)^\mathsf{T} \nabla \mathbf{h}_{t+1} + \left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right)^\mathsf{T} \nabla \mathbf{o}_t$$

Backpropagation through Time (BPTT) (cont.)

Go "backwards" through the unfolded unit, starting at final time step $t = T$ and iteratively compute gradients for $t = T, \ldots, 1$

Reminder: $\hat{\mathbf{y}}_t = \sigma(\mathbf{o}_t) = \sigma(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y)$

$$\nabla \mathbf{o}_t = \sigma'(\mathbf{o}_t) \cdot \frac{\partial L}{\partial \hat{\mathbf{y}}_t}(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

$$\nabla \mathbf{W}_{hy,t} = \nabla \mathbf{o}_t \mathbf{h}_t^\mathsf{T}$$

$$\nabla \mathbf{b}_{y,t} = \nabla \mathbf{o}_t$$

The gradient $\nabla \mathbf{h}_t$ depends on two elements - the hidden state influences $\mathbf{o}_t$ *and* the next hidden state $\mathbf{h}_{t+1}$:

$$\nabla \mathbf{h}_t = \left(\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right)^\mathsf{T} \nabla \mathbf{h}_{t+1} + \left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right)^\mathsf{T} \nabla \mathbf{o}_t$$

$$= \mathbf{W}_{hh}^\mathsf{T} \cdot \tanh'(\mathbf{W}_{hh}\mathbf{h}_t + \mathbf{W}_{xh}\mathbf{x}_{t+1} + \mathbf{b}_h) \cdot \nabla \mathbf{h}_{t+1} + \mathbf{W}_{hy}^\mathsf{T} \nabla \mathbf{o}_t$$

Backpropagation through Time (BPTT) (cont.)

- Note: For $t = 0$ and $t = T$, we only need one element of the sum.

Backpropagation through Time (BPTT) (cont.)

- Note: For $t = 0$ and $t = T$, we only need one element of the sum.
- Since we can now compute $\nabla \mathbf{h}_t$, we can get the remaining gradients
- Reminder: $\mathbf{h}_t = \tanh(\mathbf{u}_t) = \tanh(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h)$ Then:

$$\nabla \mathbf{W}_{hh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{h}_{t-1}^\mathsf{T}$$
$$\nabla \mathbf{W}_{xh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{x}_t^\mathsf{T}$$
$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

Backpropagation through Time (BPTT) (cont.)

- Note: For $t = 0$ and $t = T$, we only need one element of the sum.
- Since we can now compute $\nabla \mathbf{h}_t$, we can get the remaining gradients
- Reminder: $\mathbf{h}_t = \tanh(\mathbf{u}_t) = \tanh(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h)$ Then:

$$\nabla \mathbf{W}_{hh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{h}_{t-1}^\mathsf{T}$$
$$\nabla \mathbf{W}_{xh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{x}_t^\mathsf{T}$$
$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

- Currently, gradient depends on t. How do we get the gradient for the sequence?

Backpropagation through Time (BPTT) (cont.)

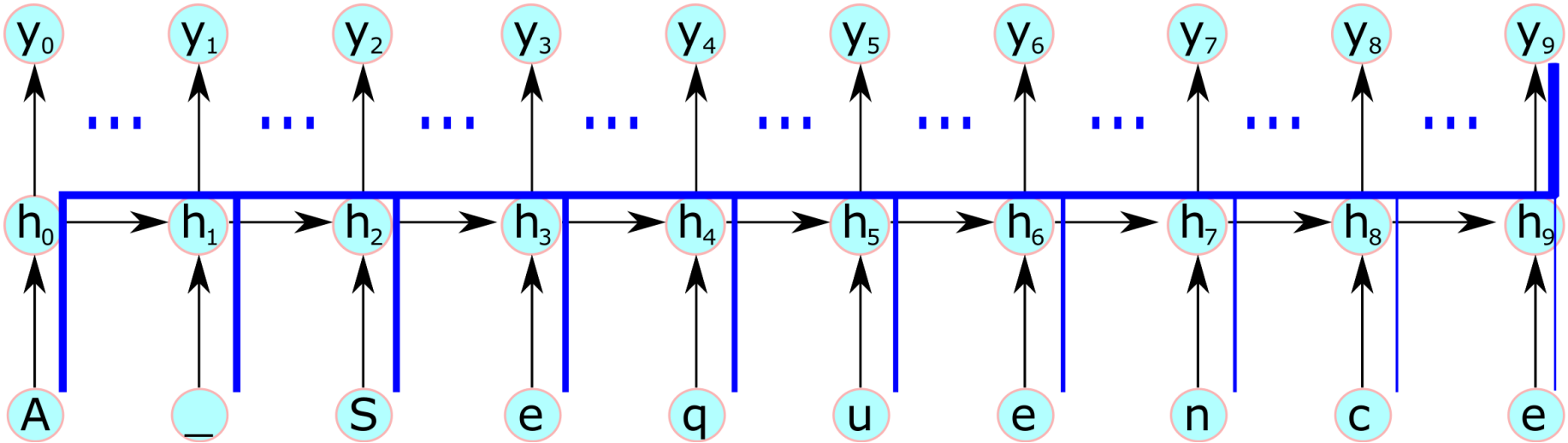
- Note: For $t = 0$ and $t = T$, we only need one element of the sum.
- Since we can now compute $\nabla \mathbf{h}_t$, we can get the remaining gradients
- Reminder: $\mathbf{h}_t = \tanh(\mathbf{u}_t) = \tanh(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h)$ Then:

$$\nabla \mathbf{W}_{hh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{h}_{t-1}^\mathsf{T}$$
$$\nabla \mathbf{W}_{xh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{x}_t^\mathsf{T}$$
$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

- Currently, gradient depends on t. How do we get the gradient for the sequence?
- Unrolled unit is a network with **shared weights**

➔ For each gradient, simply **sum over all time-steps** $t = \{1, \ldots, T\}$!

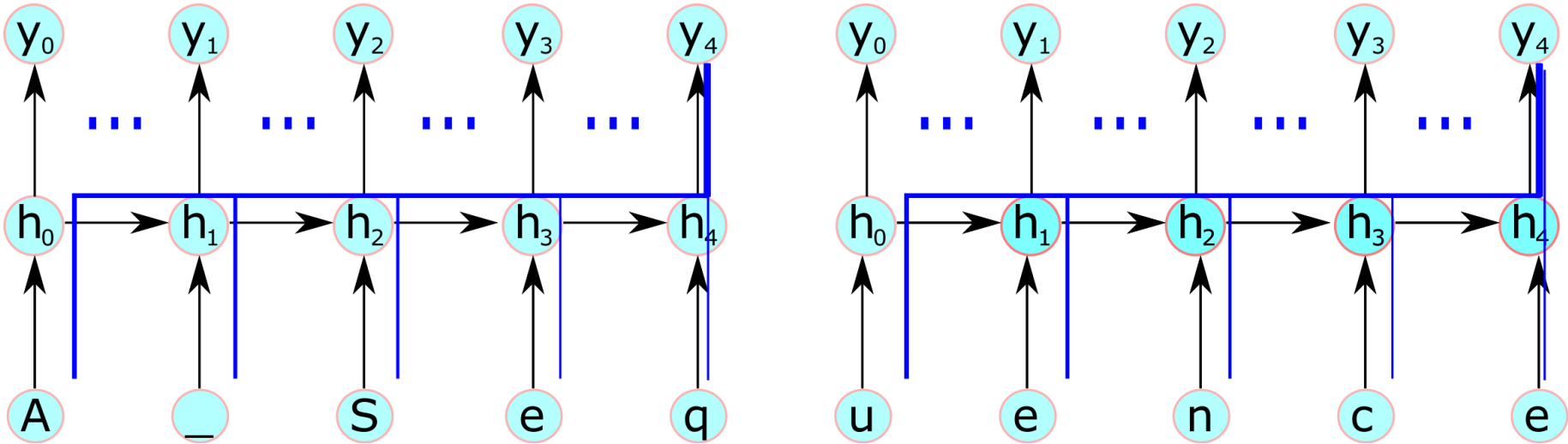
Normal BPTT



- BPTT: One update requires backpropagation through a **complete sequence**

➔ Single parameter update is very expensive!

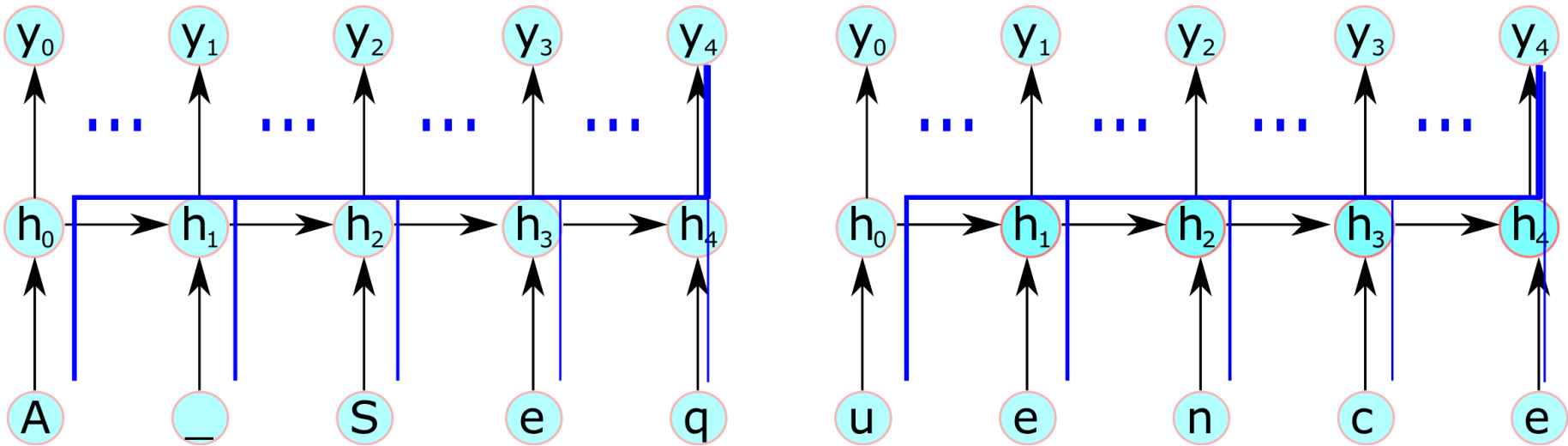
Naive BPTT



Naive Solution:

- Split long sequences into batches of smaller parts

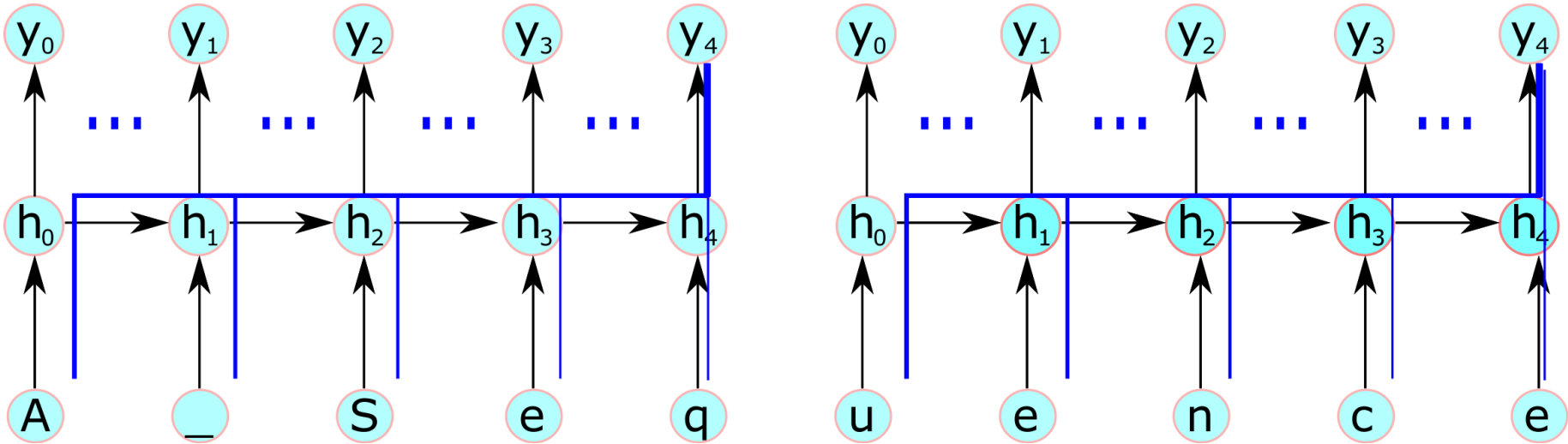
Naive BPTT



Naive Solution:

- Split long sequences into batches of smaller parts
- Might work ok in practice, but blind to long-term dependencies
- Can we do better? Yes!

Naive BPTT



Naive Solution:

- Split long sequences into batches of smaller parts
- Might work ok in practice, but blind to long-term dependencies
- Can we do better? Yes!

➔ **Truncated backpropagation through time (TBPTT)**

Truncated Backpropagation through Time (TBPTT)

- Main idea: Keep processing sequence as a whole
- Adapt frequency and depth of update:
  - Every k_1 time steps, run BPTT for k_2 time steps
  - ➔ Parameter update cheap if k_2 small
- Hidden states are still exposed to many time steps

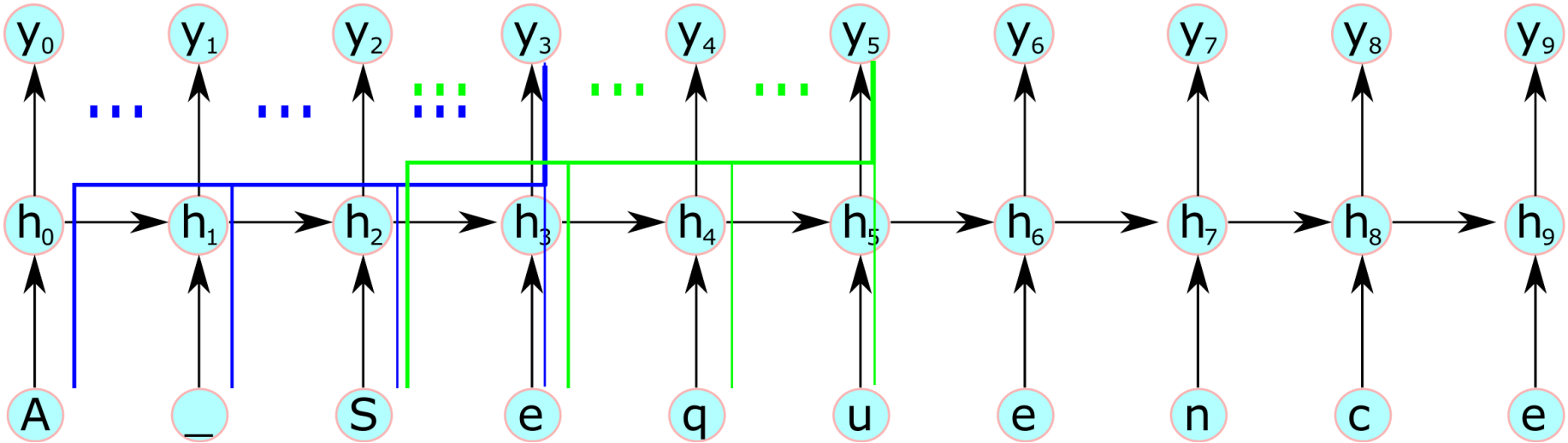
Truncated Backpropagation through Time (TBPTT)

- Main idea: Keep processing sequence as a whole
- Adapt frequency and depth of update:
  - Every k_1 time steps, run BPTT for k_2 time steps
  - ➔ Parameter update cheap if k_2 small
- Hidden states are still exposed to many time steps

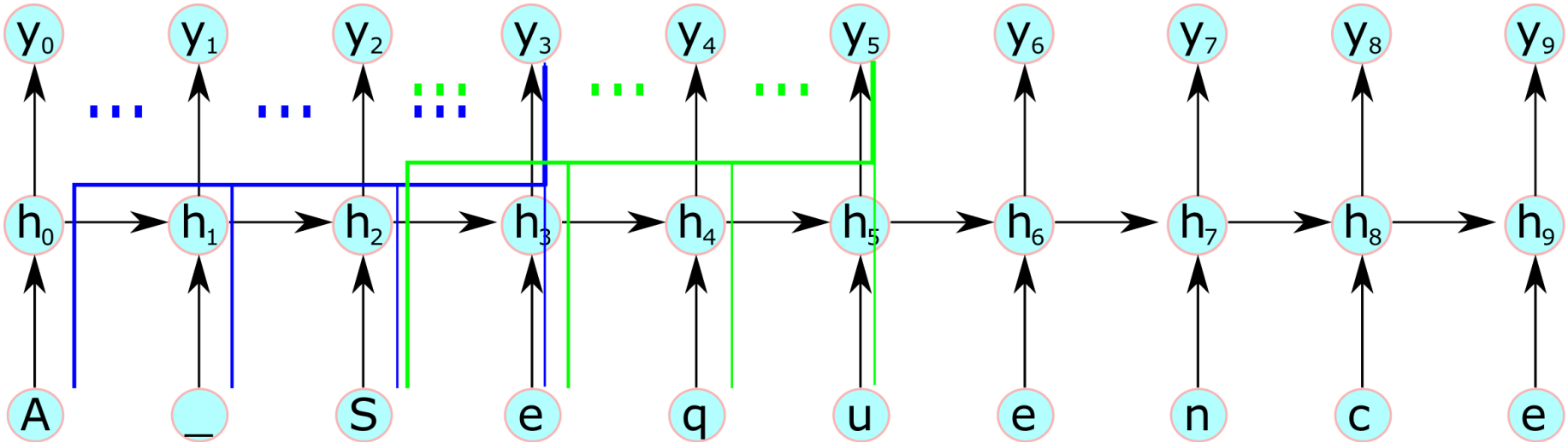
Algorithm:

1: **for** t **from** 1 **to** T **do**:
2: Run RNN for one step, computing h_t and y_t
3: **if** t mod $k_1 == 0$:
4: Run BPTT from t down to $t - k_2$

Truncated BPTT



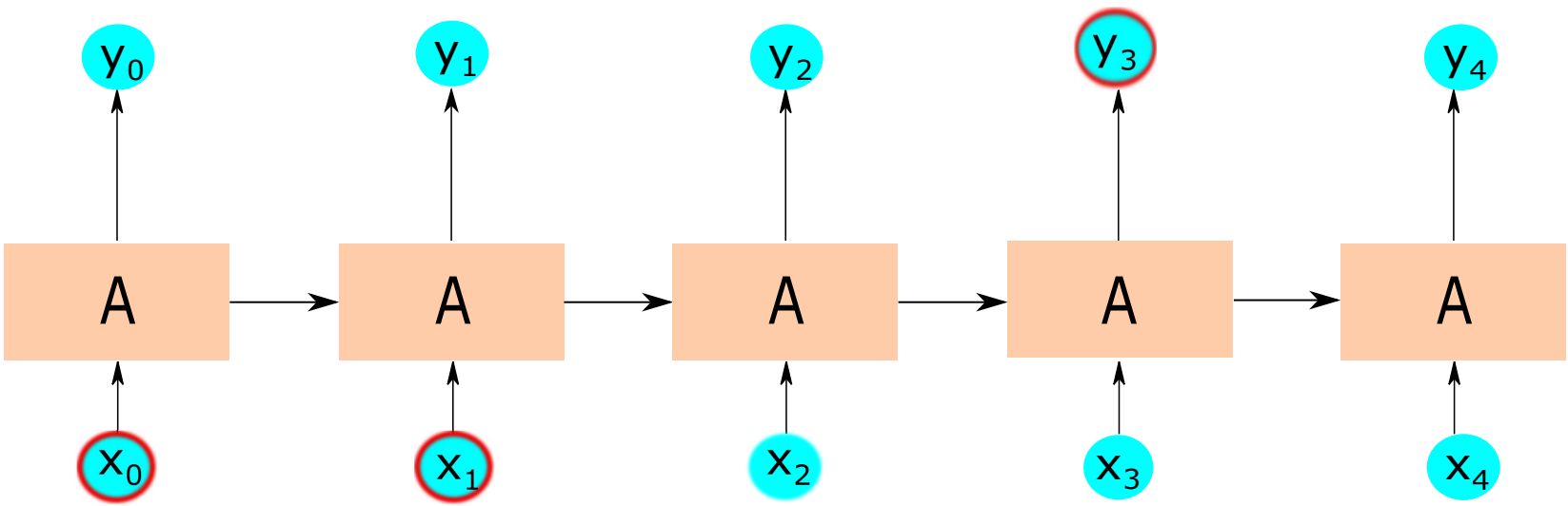
Truncated BPTT



So can we train successful RNNs now? Still no...

The Long-Term Dependency Problem with Basic RNNs

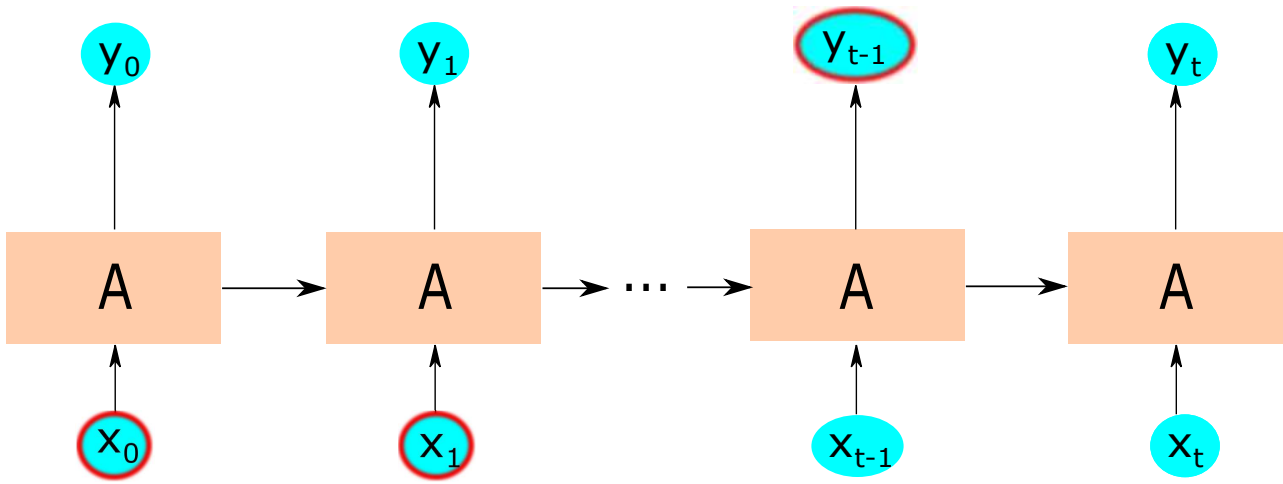
- Short term dependencies work fine
- Example: Predict next word in "the clouds are in the [*sky*]"



- Contextual information nearby ➔ can be encoded in hidden state easily

The Long-Term Dependency Problem with Basic RNNs (cont.)

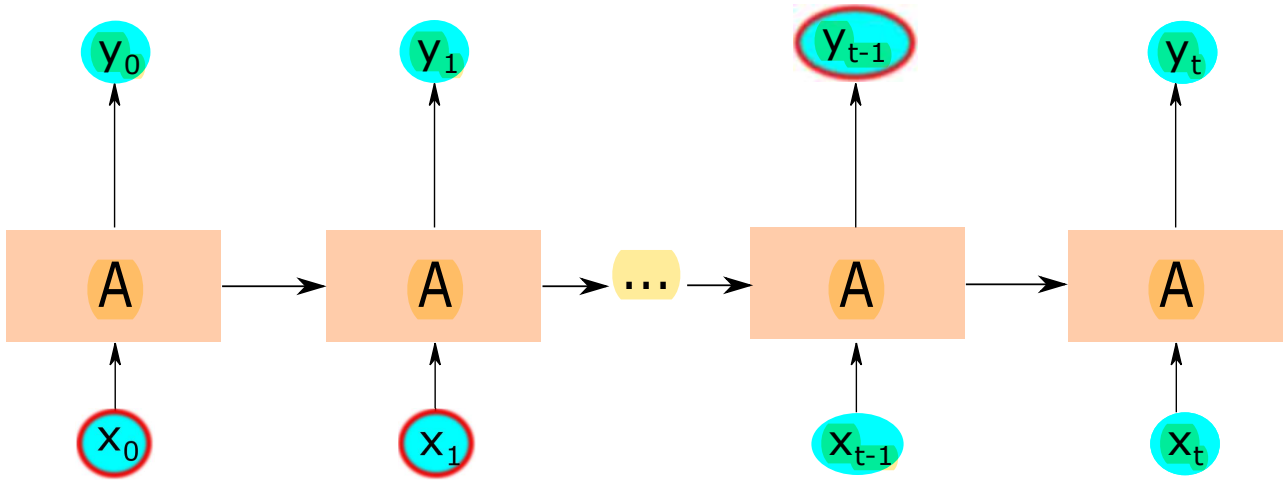
- Harder to connect relevant past and present inputs for longer time spans
- Example: Predict next word in "I grew up in Germany . . . I speak fluent [*German*]"



- Contextual information far away

The Long-Term Dependency Problem with Basic RNNs (cont.)

- Harder to connect relevant past and present inputs for longer time spans
- Example: Predict next word in "I grew up in Germany . . . I speak fluent [*German*]"



- Contextual information far away
- **Why** does this make a difference?

The Long-Term Dependency Problem with Basic RNNs (cont.)

Old acquaintances: vanishing and exploding gradients

- Layers and time steps of deep RNNs are related through multiplication

➔ Gradients prone to vanishing or exploding (Hochreiter and Schmidhuber [12])

The Long-Term Dependency Problem with Basic RNNs (cont.)

Old acquaintances: vanishing and exploding gradients

- Layers and time steps of deep RNNs are related through multiplication

➔ Gradients prone to vanishing or exploding (Hochreiter and Schmidhuber [12])

- **Exploding gradient** relatively easy to solve by truncating gradient
- **Vanishing gradient** harder to solve!

The Long-Term Dependency Problem with Basic RNNs (cont.)

Old acquaintances: vanishing and exploding gradients

- Layers and time steps of deep RNNs are related through multiplication

➔ Gradients prone to vanishing or exploding (Hochreiter and Schmidhuber [12])

- **Exploding gradient** relatively easy to solve by truncating gradient
- **Vanishing gradient** harder to solve!

Additional problem: memory overwriting

- Hidden state is overwritten each time step

  ➔ Detecting long-term dependencies even more difficult

The Long-Term Dependency Problem with Basic RNNs (cont.)

Old acquaintances: vanishing and exploding gradients

- Layers and time steps of deep RNNs are related through multiplication

➔ Gradients prone to vanishing or exploding (Hochreiter and Schmidhuber [12])

- **Exploding gradient** relatively easy to solve by truncating gradient
- **Vanishing gradient** harder to solve!

Additional problem: memory overwriting

- Hidden state is overwritten each time step

  ➔ Detecting long-term dependencies even more difficult
- Can we do better? Again, yes!

NEXT TIME

ON DEEP LEARNING

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 3

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 24, 2020

Long Short-Term Memory Units (LSTMs)

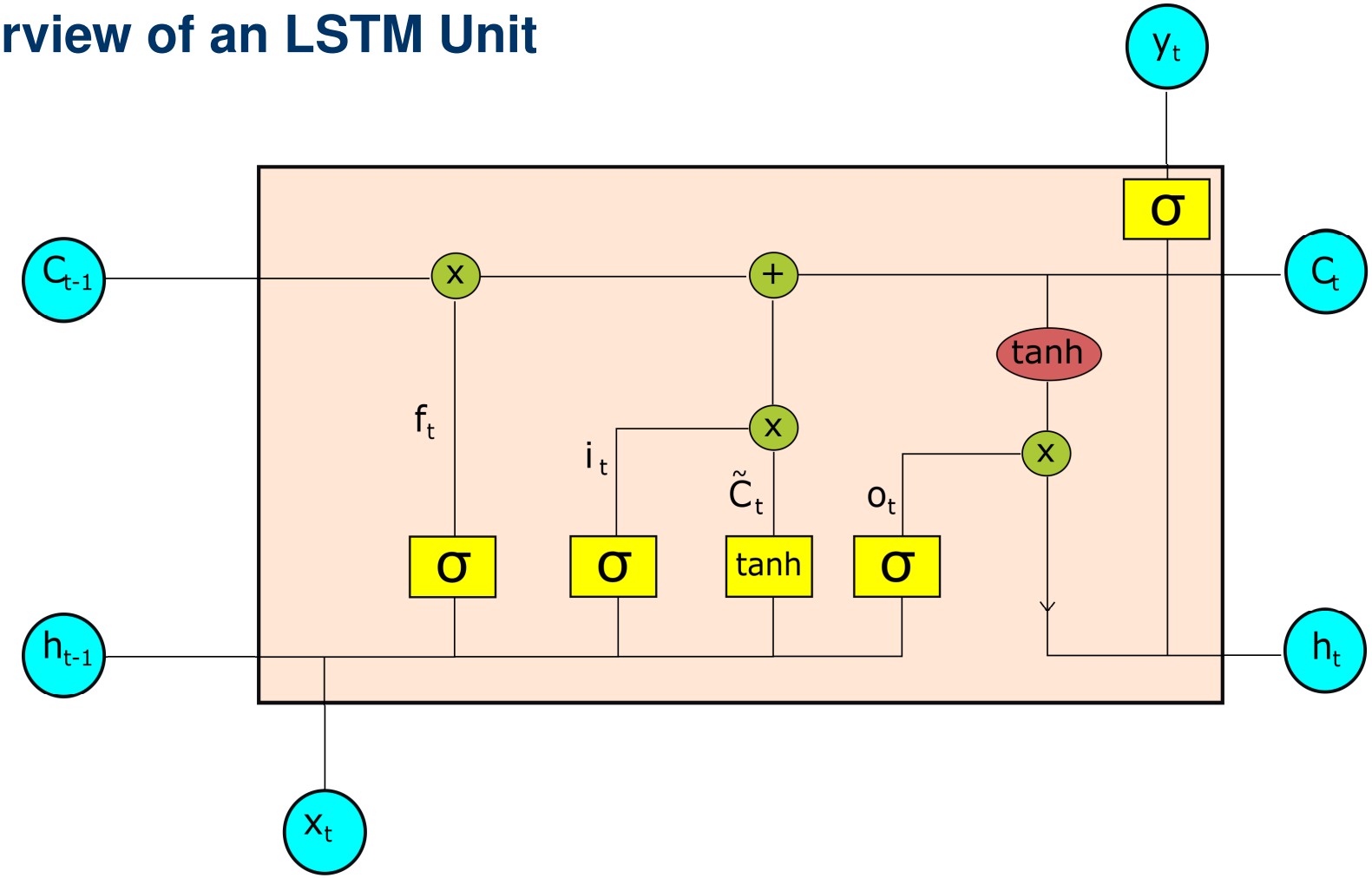
Background

- **Long Short-Term Memory Units (LSTMs)** introduced by Hochreiter & Schmidhuber in 1997
- Designed to solve vanishing gradient and learning long-term dependencies
- Main idea: introduction of **gates** that control writing and accessing “memory” in additional **cell state**

Sepp Hochreiter

Jürgen Schmidhuber

Overview of an LSTM Unit



LSTM unit workflow

Elements of LSTM units:

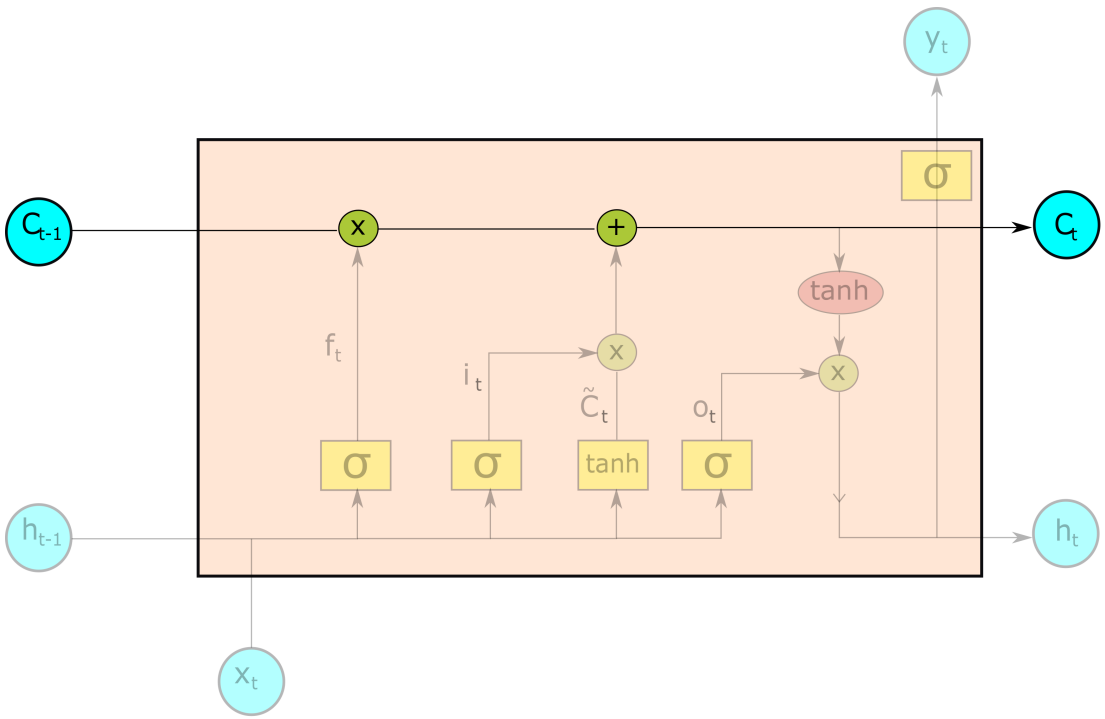
- Input $\mathbf{x}_t$
- Hidden state $\mathbf{h}_{t-1}/\mathbf{h}_t$
- Cell state $\mathbf{C}_{t-1}/\mathbf{C}_t$
- Output $\mathbf{y}_t$

Update of internal states in multiple steps:

1) **Forget gate**: Forgetting old information in cell state
2) **Input gate**: Deciding on new input for cell state
3) Computing the updated cell state
4) Computing the updated hidden state

LSTM Cell State

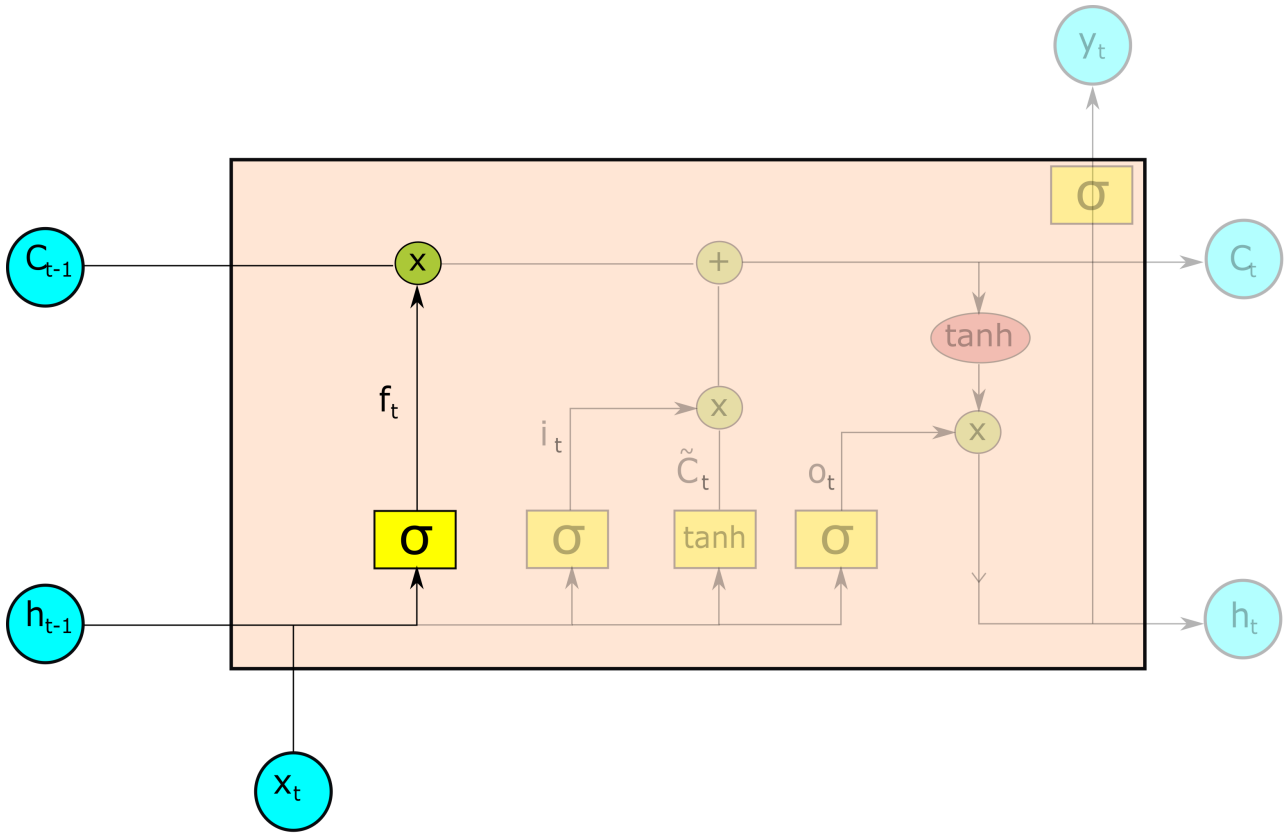
- $\mathbf{C_t}$: **Cell state** after time point *t*
- Undergoes only **linear changes**: no activation function!
- $\mathbf{C_t}$ can flow through a unit unchanged ➔ cell state can be constant for multiple time steps



Forget Gate: Forgetting Old Information

- Key idea: “forgetting” and “memorizing” information in separate steps
- $\mathbf{f}_t$ controls how much of the previous cell state is forgotten:

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f \cdot \left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_f\right)$$

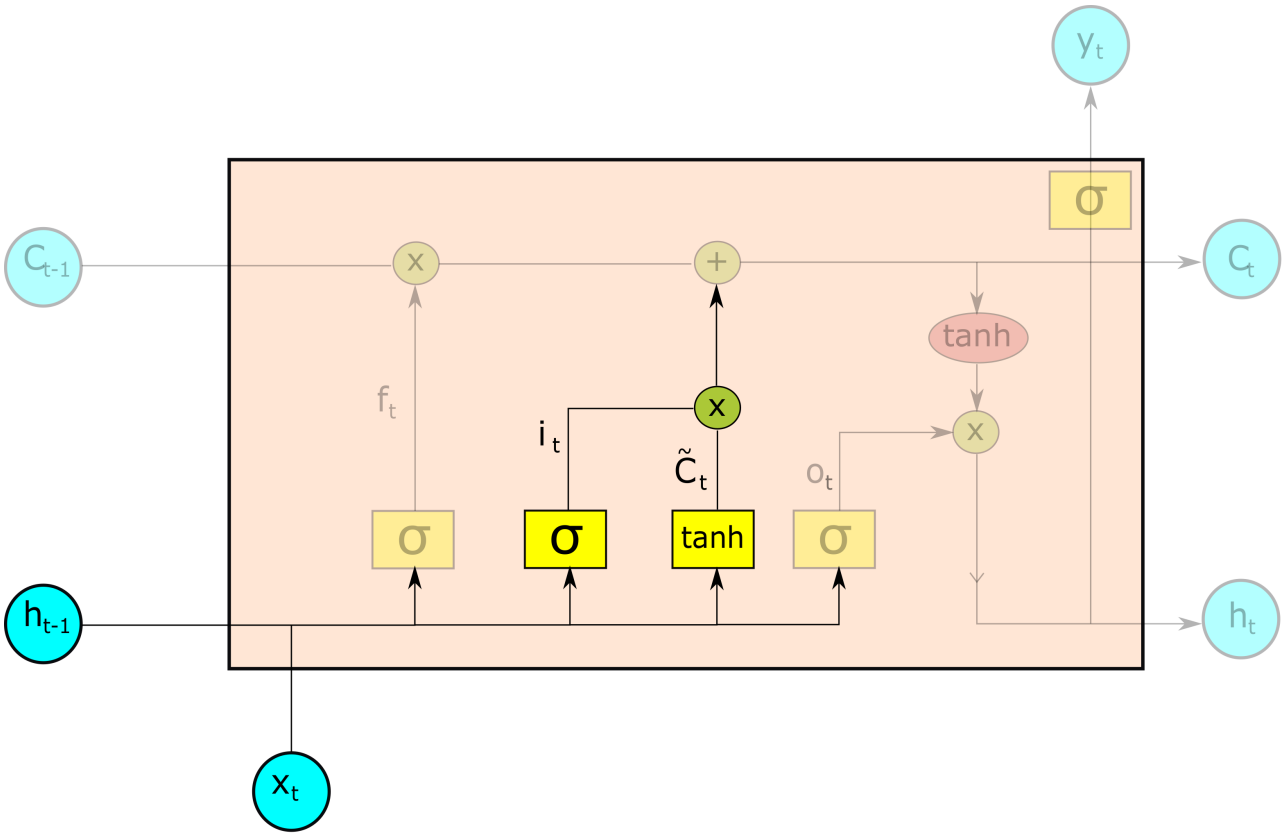


Input Gate: Deciding on New Input

Combination of input and hidden state on two paths:

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i \cdot \left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_i\right)$$

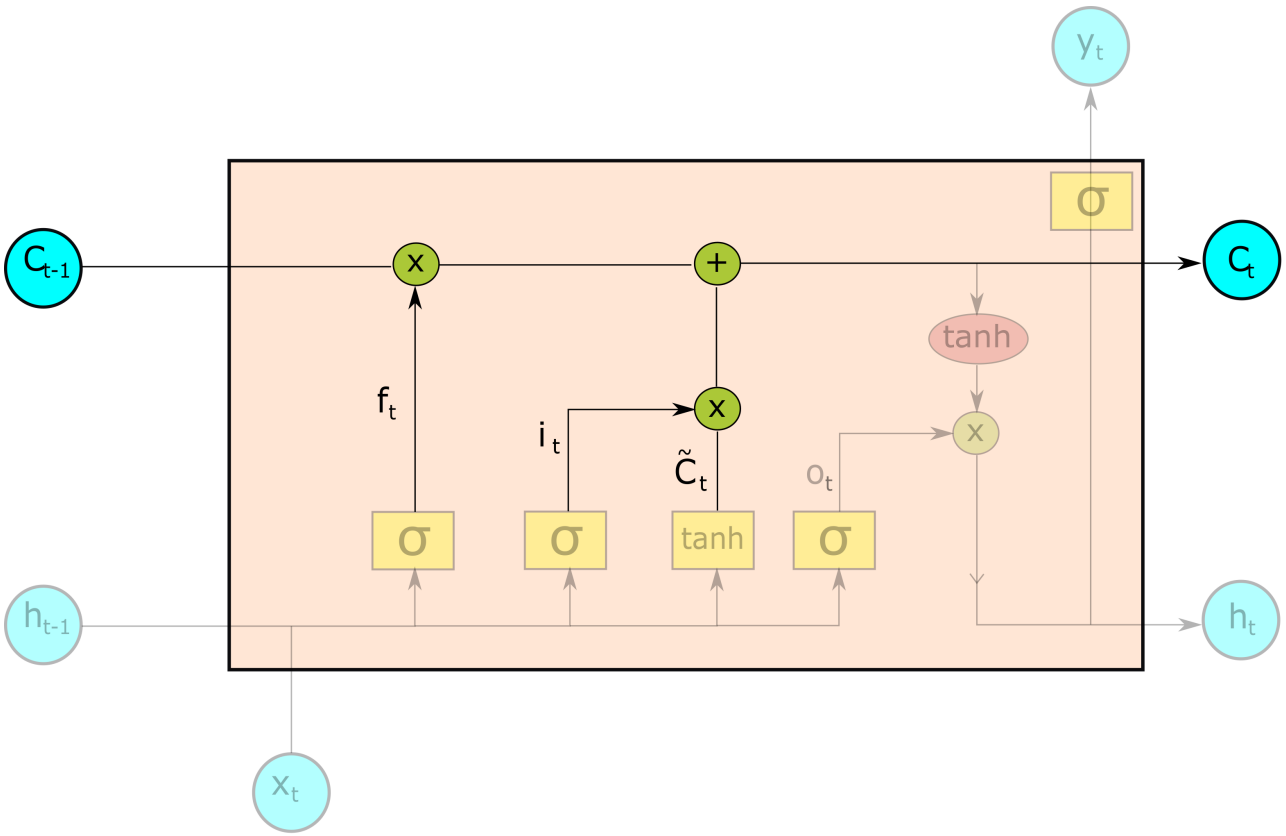
$$\tilde{\mathbf{C}}_t = tanh\left(\mathbf{W}_C \cdot \left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_C\right)$$



Updating the Cell State

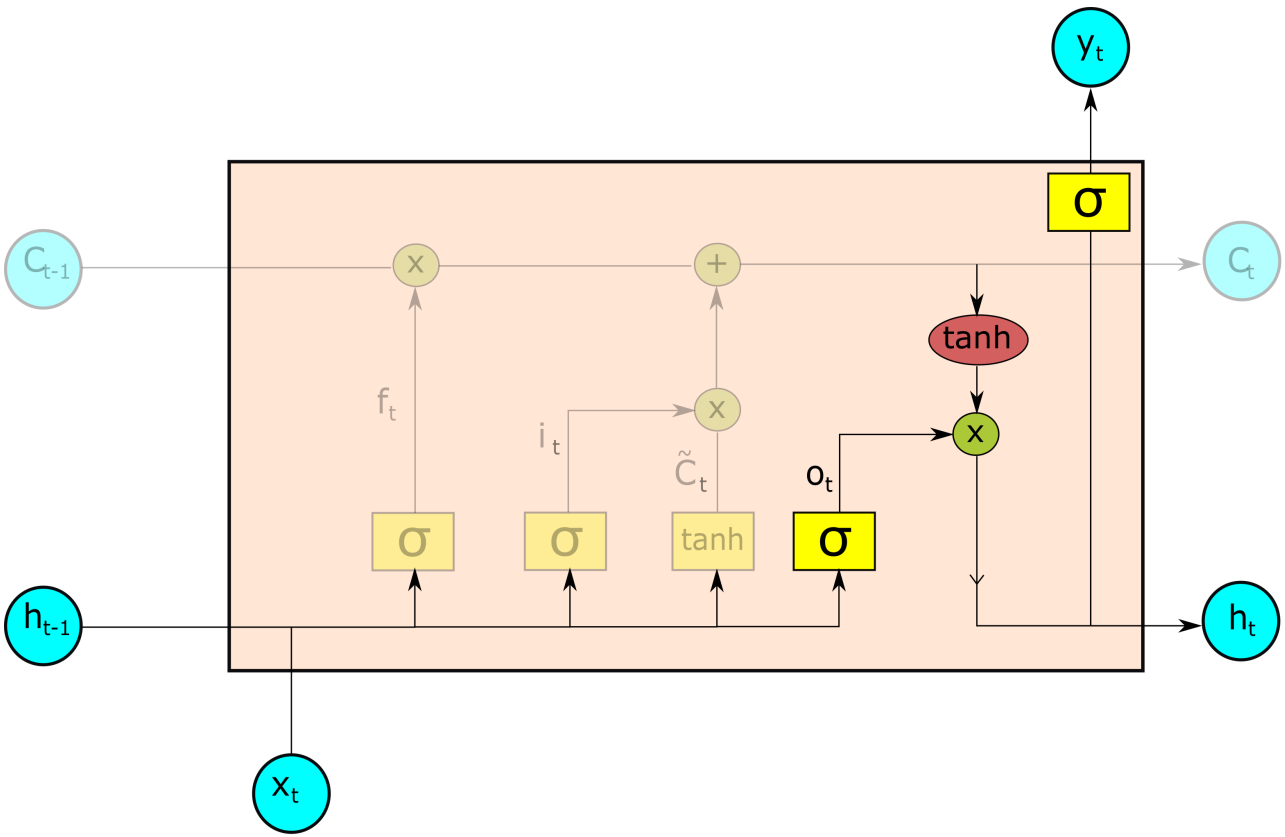
- New cell state: Sum of "remaining information" from $\mathbf{C}_{t-1}$ and new information from input and hidden state ($\odot$: element-wise multiplication)

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t$$



Updating the Hidden State and Computing the Output

- Important: Cell state and hidden state are updated **separately**
- Output $\mathbf{y}_t$ directly depends on the hidden state $\mathbf{h}_t$

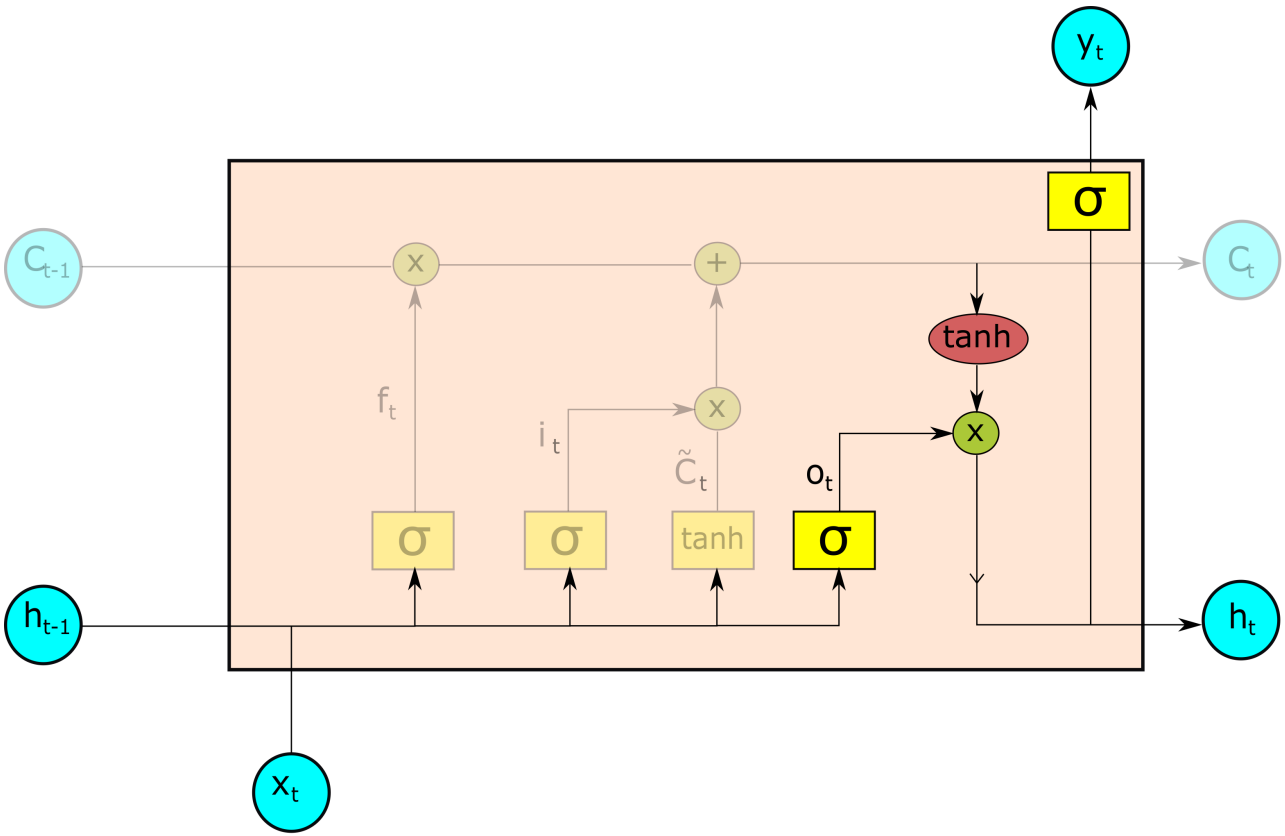


Updating the Hidden State and Computing the Output (cont.)

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot tanh\left(\mathbf{C}_t\right)$$

$$\mathbf{y}_t = \sigma\left(\mathbf{h}_t\right)$$



Gated Recurrent Units

Motivation

- LSTM great idea, but many parameters and difficult to train

Motivation

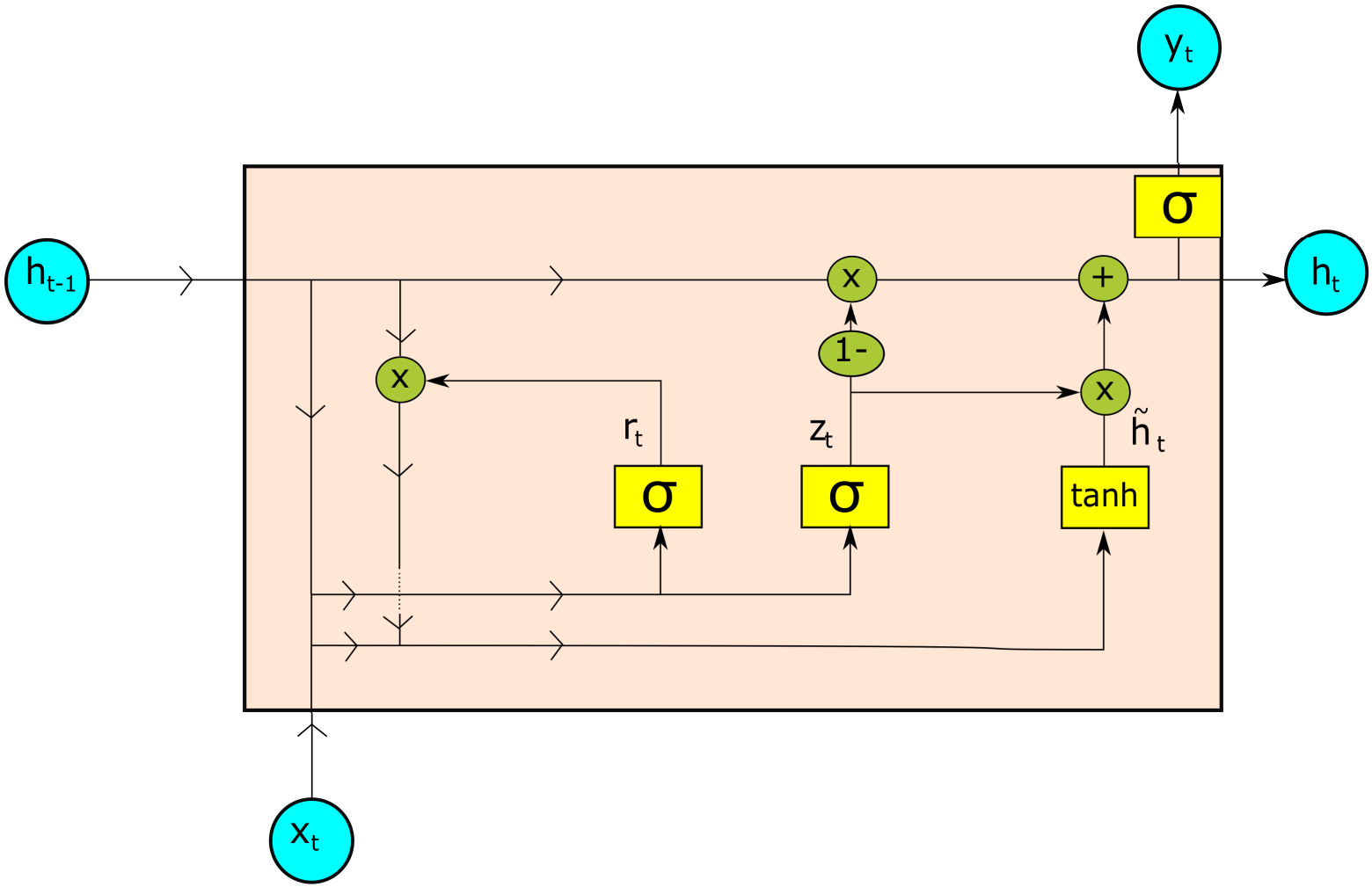
- LSTM great idea, but many parameters and difficult to train

➔ Gated Recurrent Unit (GRU)

- Originally introduced by Cho et al. in 2014 for statistical machine translation
- Variant of the LSTM unit, but simpler and fewer parameters

Kyunghyung Cho

Structure of a GRU cell



GRU workflow

- Concept from LSTM: More control over hidden state/memory ➔ **gates**

GRU workflow

- Concept from LSTM: More control over hidden state/memory ➔ **gates**
- Main difference: No additional cell state!

  ➔ Memory operates only and directly via the hidden state

GRU workflow

- Concept from LSTM: More control over hidden state/memory ➔ **gates**
- Main difference: No additional cell state!

  ➔ Memory operates only and directly via the hidden state
- Update of the hidden state can be divided into four steps:
  1) **Reset gate**: Influence of the previous hidden state

GRU workflow

- Concept from LSTM: More control over hidden state/memory ➔ **gates**
- Main difference: No additional cell state!

  ➔ Memory operates only and directly via the hidden state
- Update of the hidden state can be divided into four steps:
  1) **Reset gate**: Influence of the previous hidden state
  2) **Update Gate**: Influence of a newly computed update

GRU workflow

- Concept from LSTM: More control over hidden state/memory ➔ **gates**
- Main difference: No additional cell state!

  ➔ Memory operates only and directly via the hidden state
- Update of the hidden state can be divided into four steps:
  1) **Reset gate**: Influence of the previous hidden state
  2) **Update Gate**: Influence of a newly computed update
  3) Proposing an updated hidden state

GRU workflow

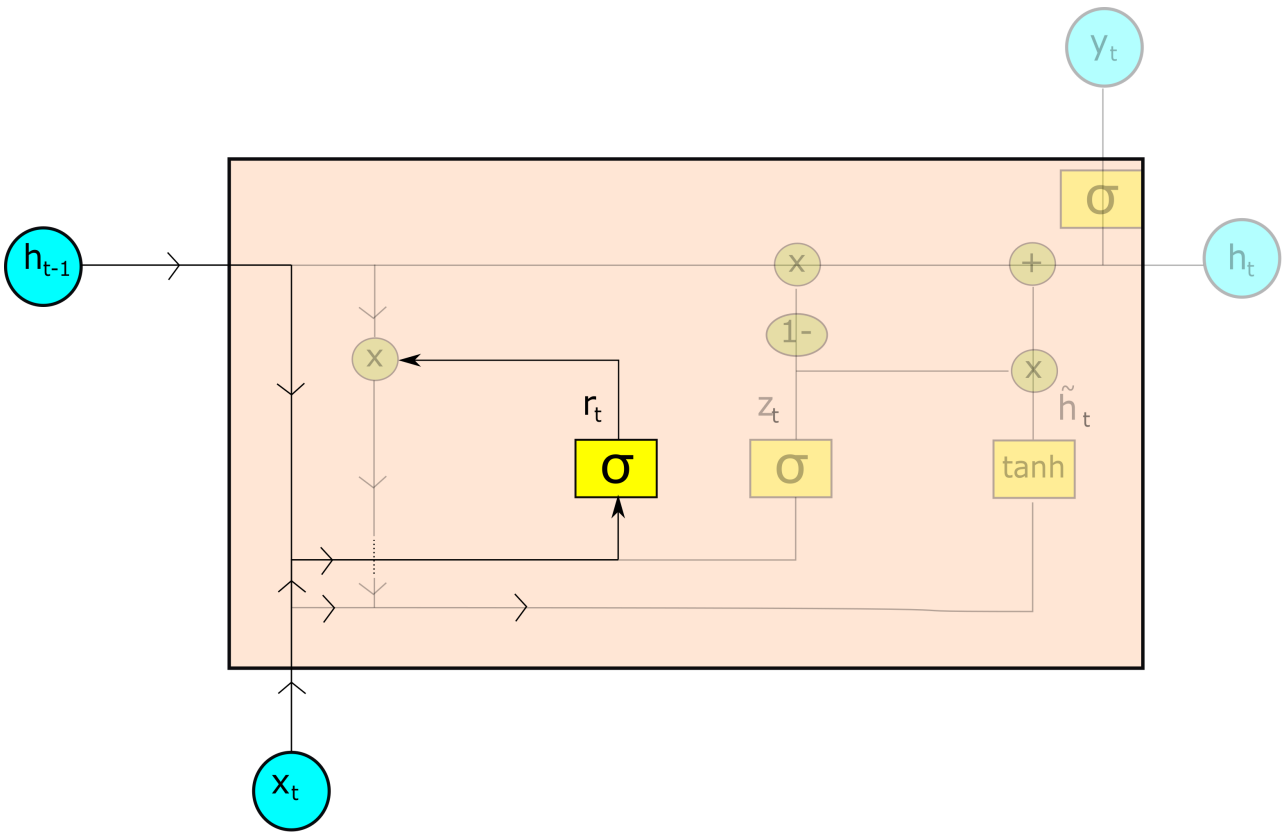
- Concept from LSTM: More control over hidden state/memory ➔ **gates**
- Main difference: No additional cell state!

  ➔ Memory operates only and directly via the hidden state
- Update of the hidden state can be divided into four steps:
  1) **Reset gate**: Influence of the previous hidden state
  2) **Update Gate**: Influence of a newly computed update
  3) Proposing an updated hidden state
  4) Computing updated hidden state

Reset gate

- Determines the influence of the previous hidden state

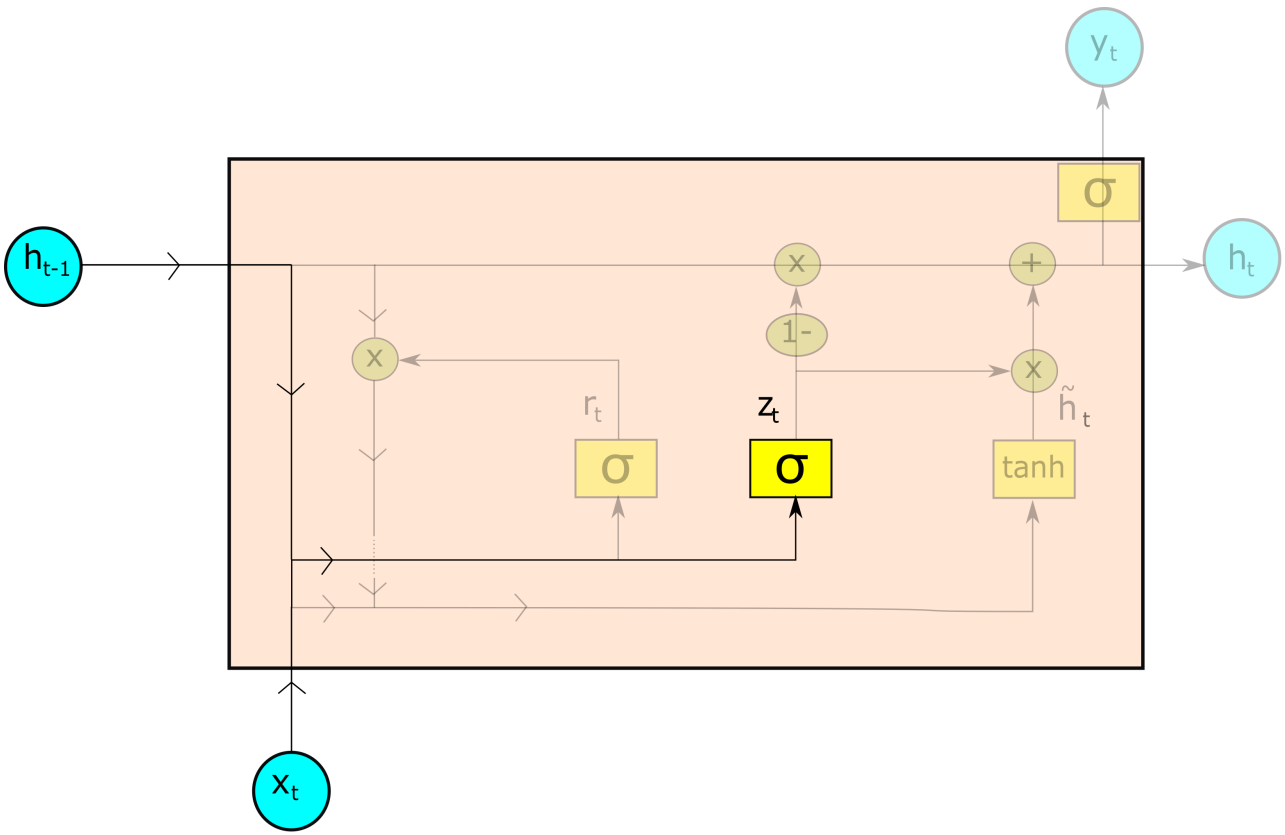
$$\mathbf{r}_t = \sigma\left(\mathbf{W}_r \cdot \left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_r\right)$$



Update Gate

- Determines the influence of an “update proposal” on the new hidden state

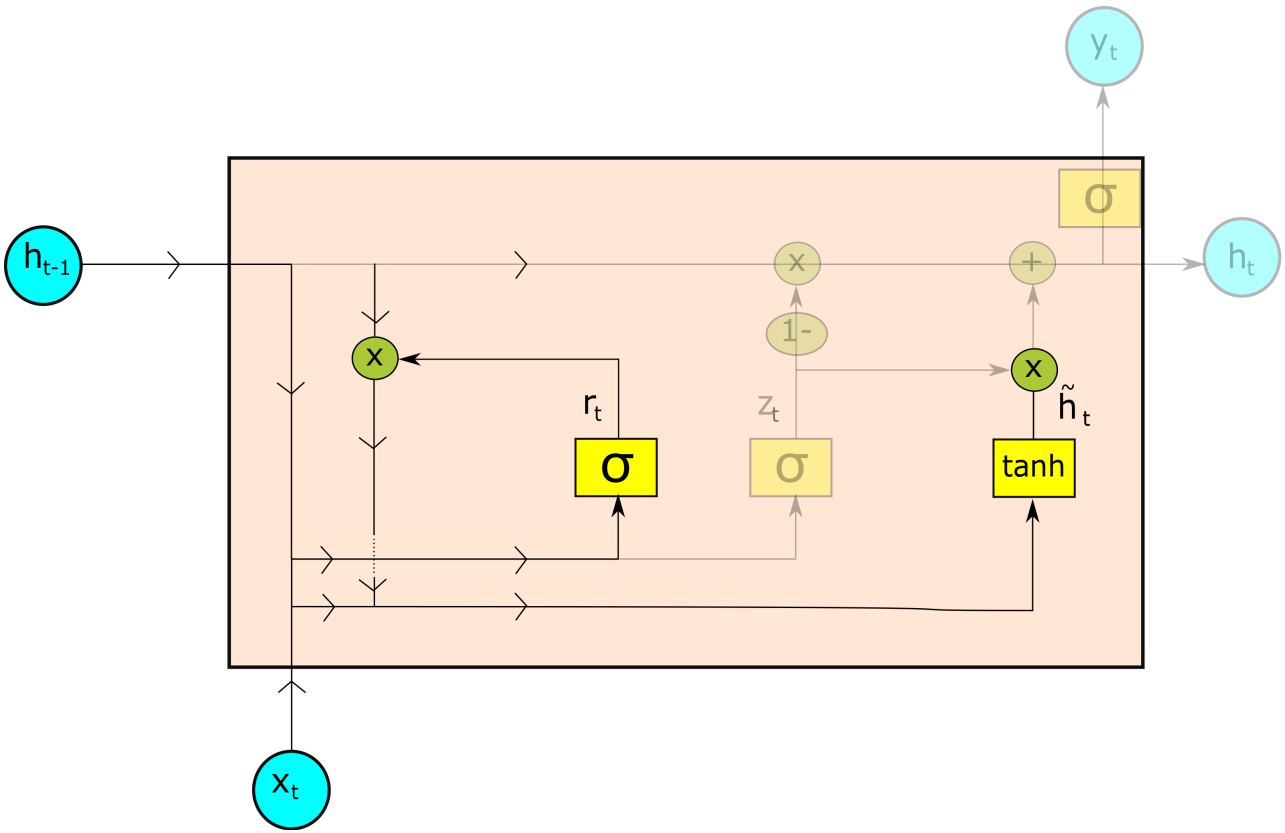
$$\mathbf{z}_t = \sigma\left(\mathbf{W}_z \cdot \left[\mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_z\right)$$



Proposing an Update

- Combination of input and “reset” hidden state.
- If $\mathbf{r}_t$ is close to 0 ➔ low influence of previous hidden state

$$\tilde{\mathbf{h}}_t = tanh\left(\mathbf{W}_h \cdot \left[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t\right] + \mathbf{b}_h\right)$$

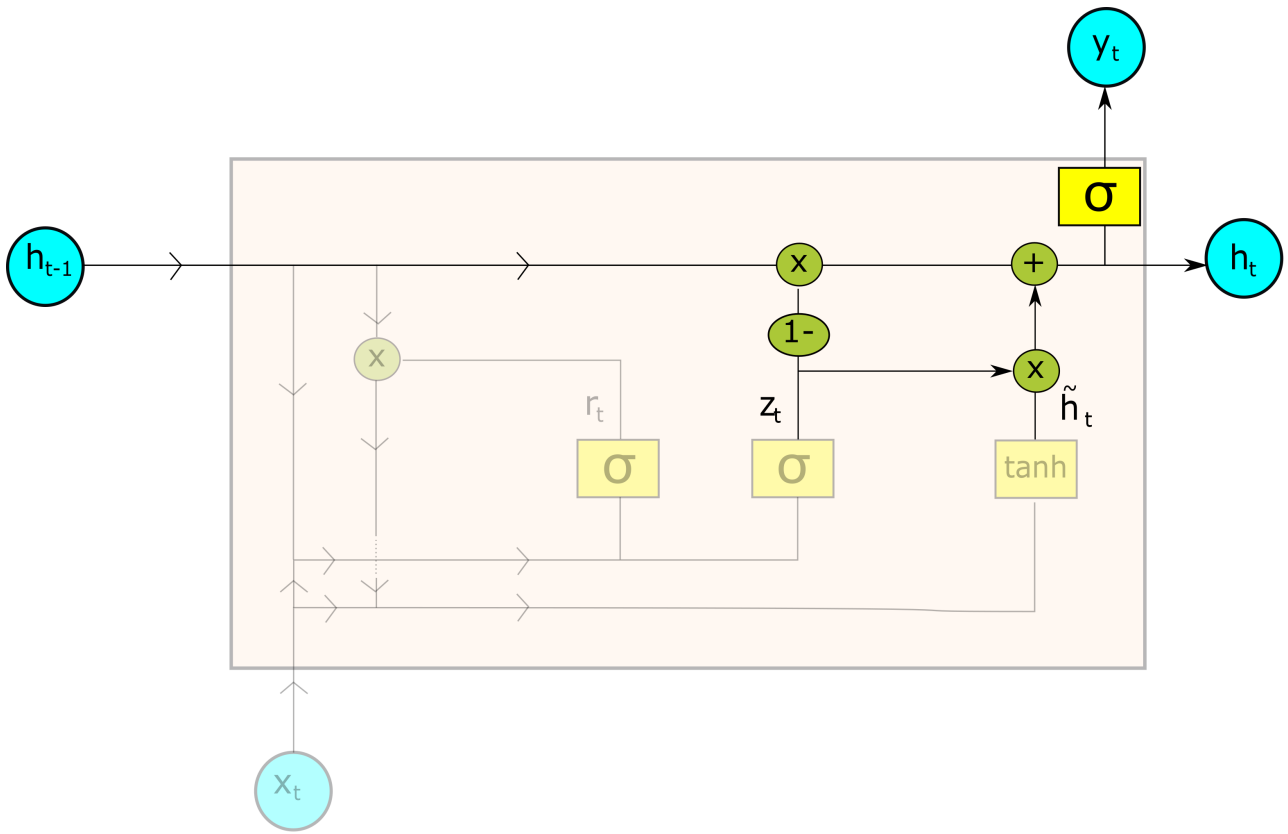


Finally: Computing the Updated Hidden State

- **Update gate** controls combination of old state and proposed update

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

- Node output: $\hat{\mathbf{y}}_t = \sigma(\mathbf{h}_t)$



Remarks

- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies

Remarks

- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies
- Units learning **short-term** dependencies have restrictive reset gates
  ➔ $\mathbf{r}_t$ close to 0: ignore previous hidden state

Remarks

- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies
- Units learning **short-term** dependencies have restrictive reset gates
  ➔ $\mathbf{r}_t$ close to 0: ignore previous hidden state
- Units learning **long-term** dependencies have restrictive update gates
  ➔ $\mathbf{z}_t$ close to 0: ignore new input

Remarks

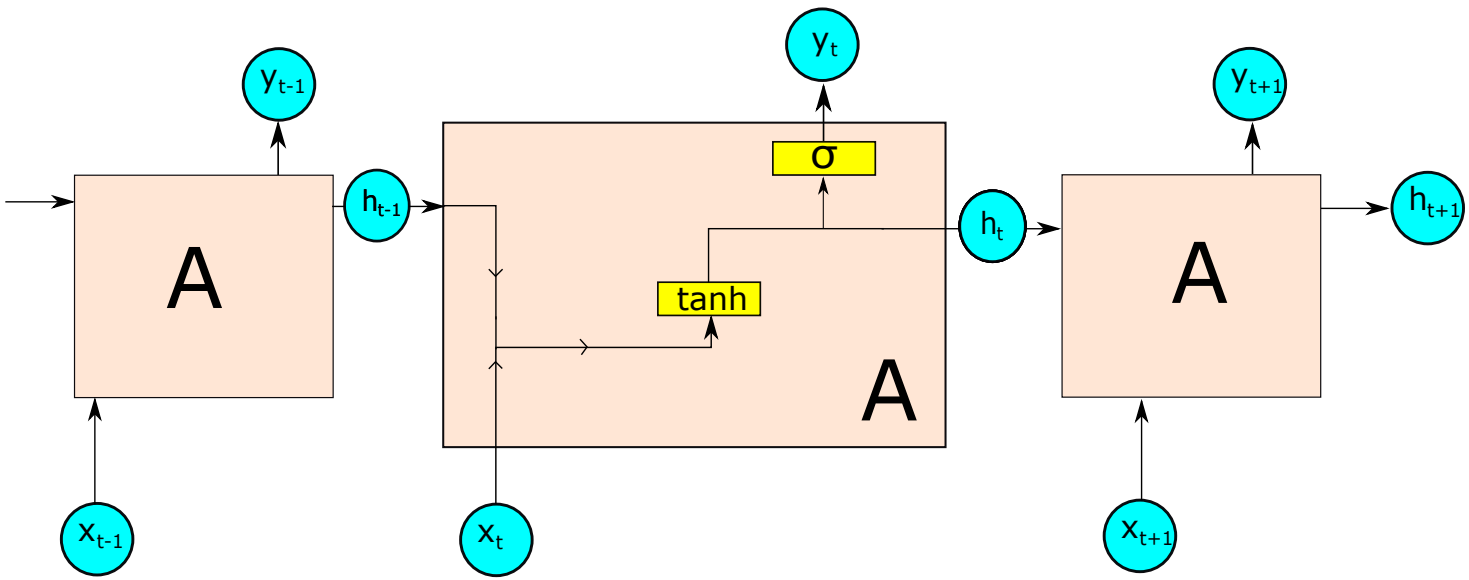
- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies
- Units learning **short-term** dependencies have restrictive reset gates
  → $\mathbf{r}_t$ close to 0: ignore previous hidden state
- Units learning **long-term** dependencies have restrictive update gates
  → $\mathbf{z}_t$ close to 0: ignore new input

→ Gates have varying "rhythm" depending on the type of information

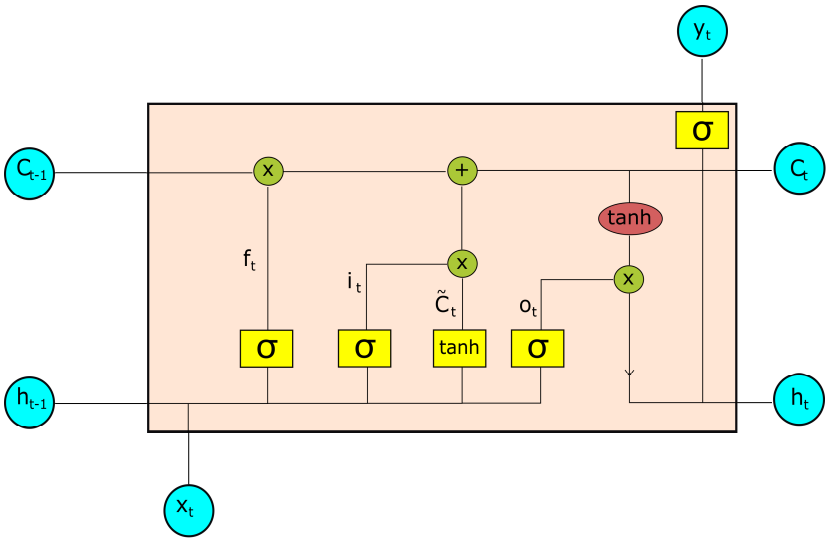
Comparison of Simple RNN units, LSTM units and GRUs

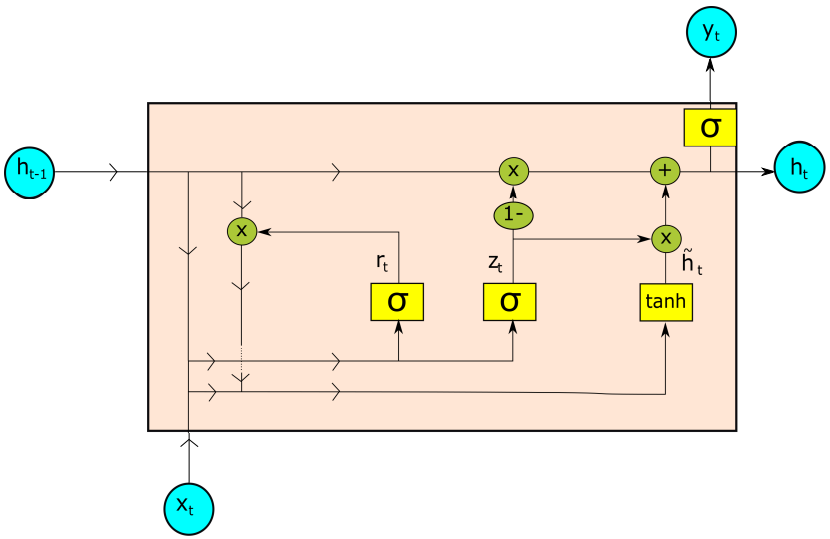
Recap: Simple RNNs

- Gradient-based training difficult (vanishing/exploding gradients)
- Short-term dependencies hide long-term dependencies due to exponentially small gradients
- Hidden state is overwritten in each time step



Advanced structures: LSTM and GRU





Advanced structures: LSTM and GRU

Similarities

- Control information flow via gates
- Ability to capture dependencies of different time scales
- Additive calculation of state preserves error during backpropagation
  ➔ more efficient training possible

Advanced structures: LSTM and GRU

Differences

LSTM	GRU
Seperate hidden and cell state	Combined hidden and cell state
Controlled exposure of memory content through output gate	Full exposure of memory content without control
Independent input and forget gate: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ ➔ New memory content independent of current memory	Common update gate: $h_t = \left(1 - z_t\right) \odot h_{t-1} + z_t \odot \tilde{h}_t)$ ➔ New memory content depends on current memory

Comparison of Recurrent Units: So what should we use?

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [3]

- Comparison of simple RNN, LSTM and GRU networks
- Tasks: Polyphonic music modeling and speech signal modeling

Comparison of Recurrent Units: So what should we use?

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [3]

- Comparison of simple RNN, LSTM and GRU networks
- Tasks: Polyphonic music modeling and speech signal modeling
- Gated recurrent units clearly outperformed regular recurrent unit
- Comparison between GRU and LSTM not conclusive, similar performance

NEXT TIME

ON DEEP LEARNING

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SCHOOL OF ENGINEERING

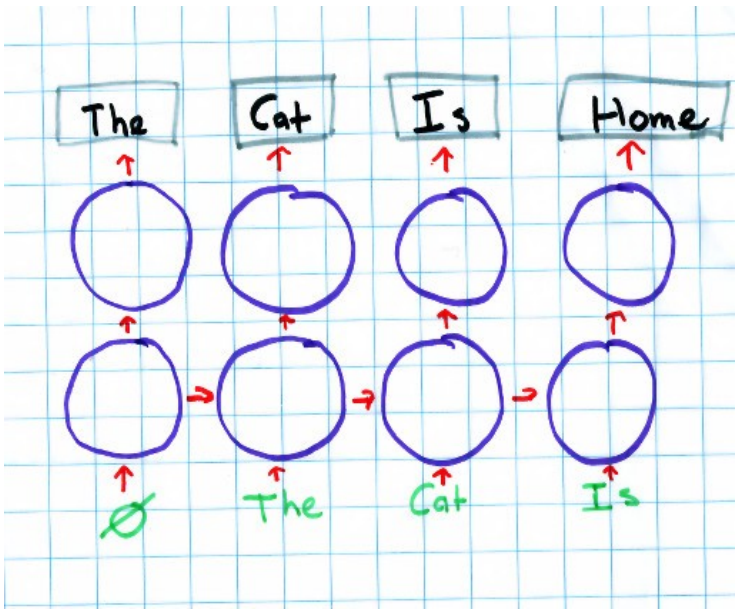
Recurrent Neural Networks - Part 4

A. Maier, V. Christlein, K. Breininger, S. Vesal, F. Meister, C. Liu, S. Gündel, S. Jaganathan, N. Maul, M. Vornehm, L. Reeb, F. Thamm, M. Hoffmann, C. Bergler, F. Denzinger, W. Fu, B. Geissler, Z. Yang
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
May 24, 2020

Sampling strategies for RNNs

Why Sampling Strategies?

- RNN can generate sequences (words/notes/...)
- RNN actually computes **probability distribution** of the next element
- Question: How do we sample this distribution?

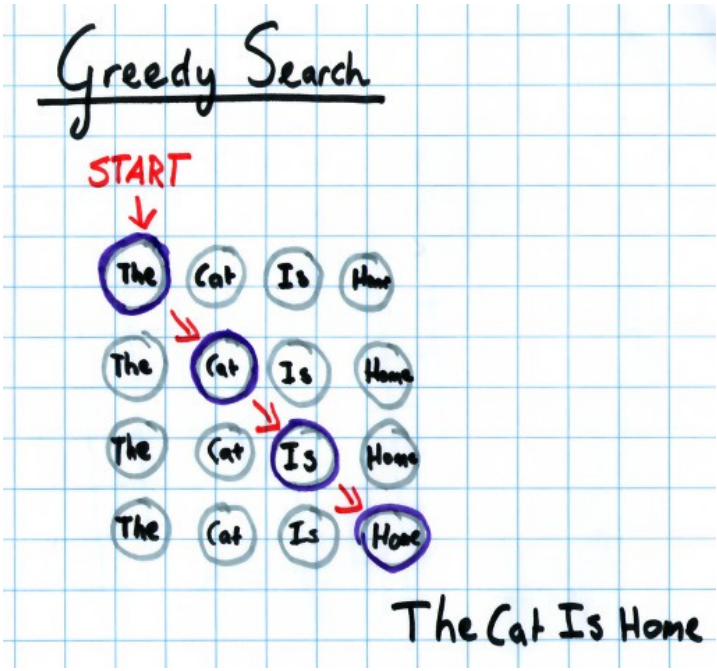


Ideal sequence - or?

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Greedy search

- **Concept**: At each point, pick the most likely element
- Generates exactly one sample sequence per experiment

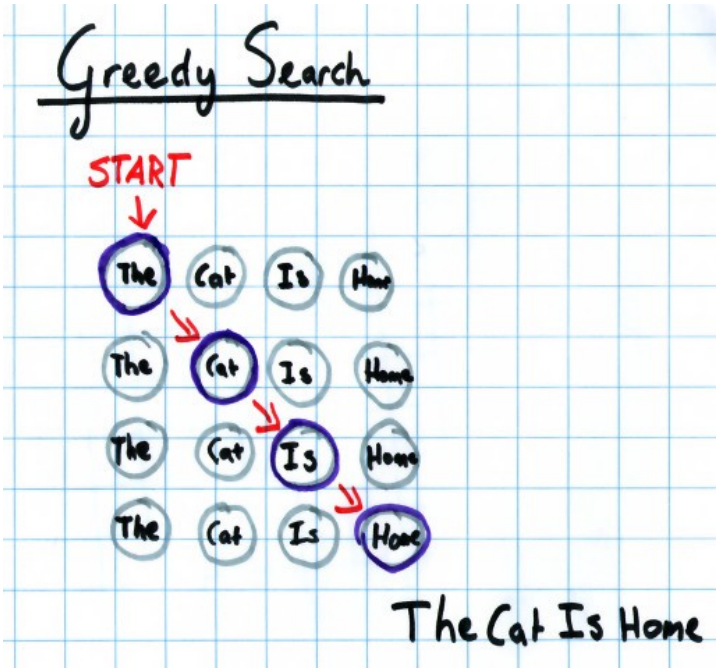


Greedy search

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Greedy search

- **Concept**: At each point, pick the most likely element
- Generates exactly one sample sequence per experiment
- **Drawback**: No **lookahead** possible!
  - Example: “The” may be most likely word after “The Cat” ➔ “The Cat The”
  - Cannot detect that “The Cat Is Home” has a higher total probability

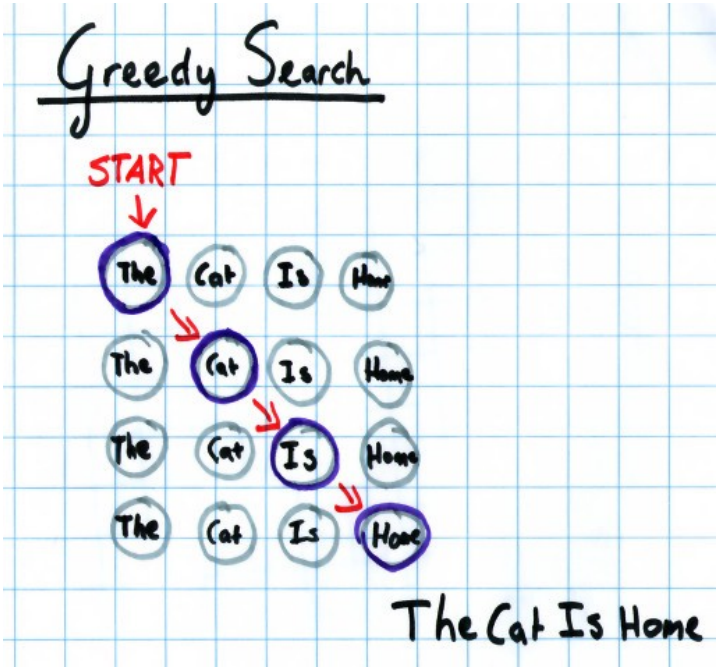


Greedy search

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Greedy search

- **Concept**: At each point, pick the most likely element
- Generates exactly one sample sequence per experiment
- **Drawback**: No **lookahead** possible!
  - Example: “The” may be most likely word after “The Cat” ➔ “The Cat The”
  - Cannot detect that “The Cat Is Home” has a higher total probability
- Tends to repeat sequences of frequent words, e.g., “and”, “the”, “some” in speech

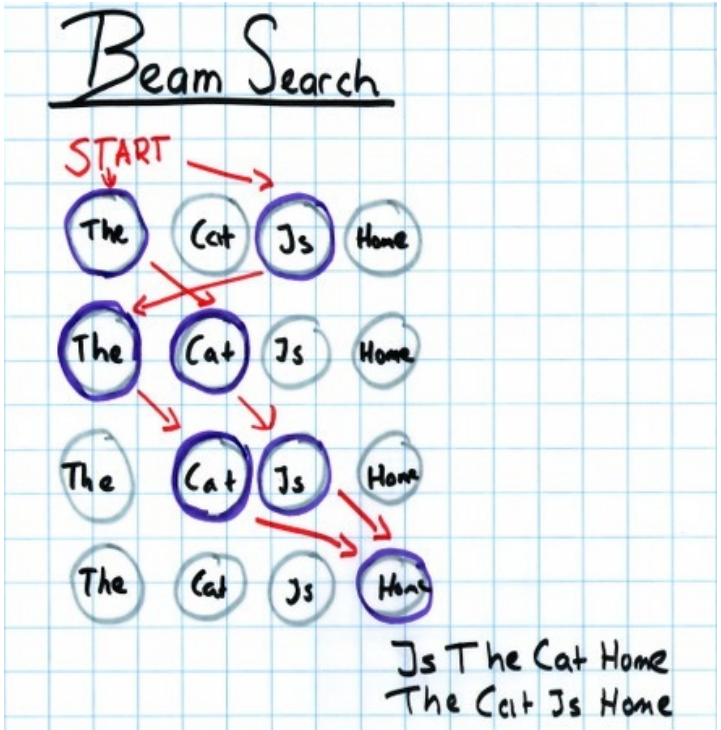


Greedy search

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Beam Search

- **Concept**: Select k most likely words (k: beam width or size)

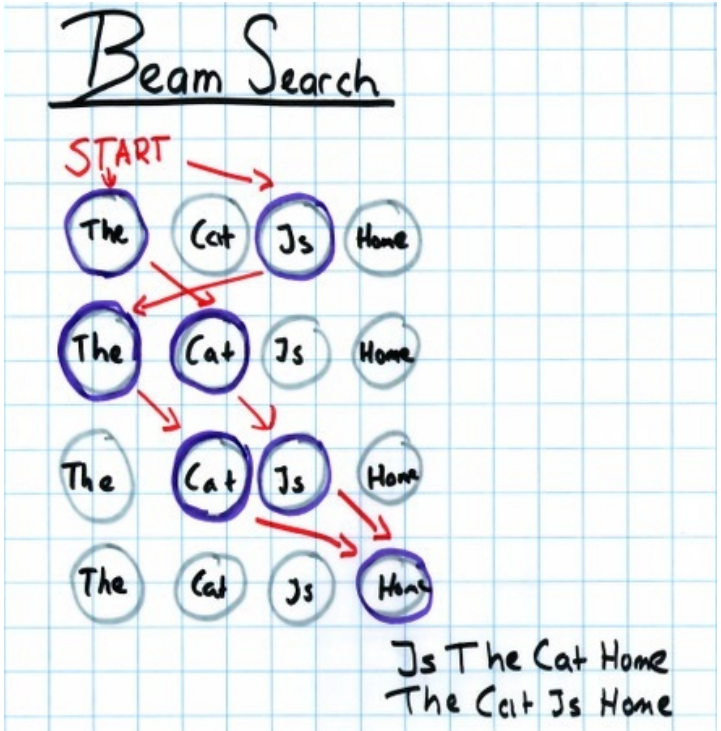


Beam Search with $k = 2$

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Beam Search

- **Concept**: Select k most likely words (k: beam width or size)
- Out of all possible sequences that have one of these k words as a **prefix**, take k most probable ones

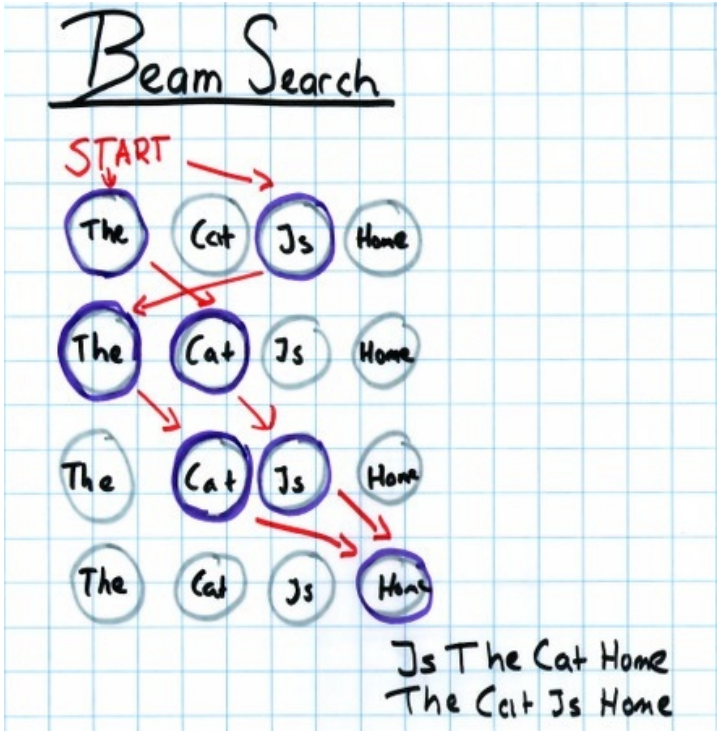


Beam Search with $k = 2$

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Beam Search

- **Concept**: Select k most likely words (k: beam width or size)
- Out of all possible sequences that have one of these k words as a **prefix**, take k most probable ones
- Iterate until end of sequence

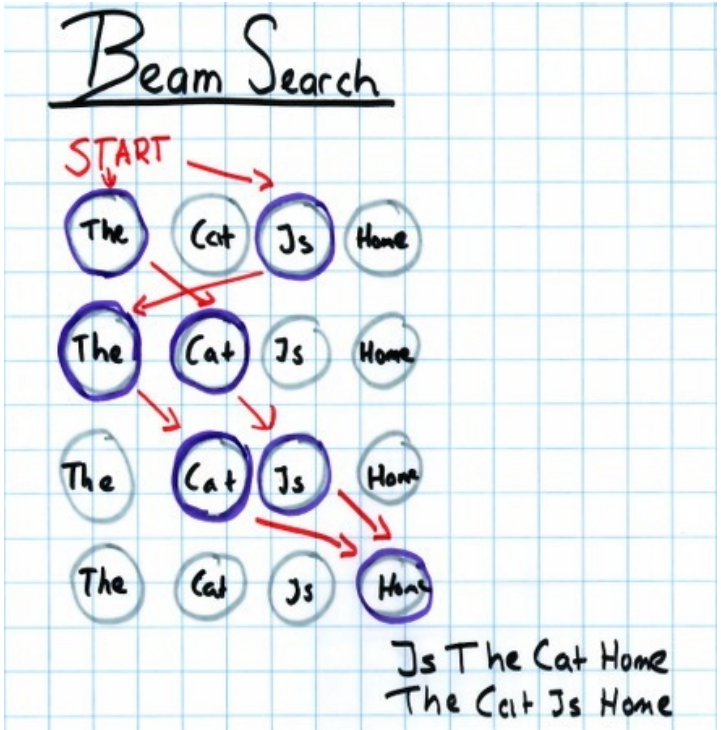


Beam Search with $k = 2$

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Beam Search

- **Concept**: Select k most likely words (k: beam width or size)
- Out of all possible sequences that have one of these k words as a **prefix**, take k most probable ones
- Iterate until end of sequence
- Can generate k sequences in one go
  ➔ usually better than greedy search

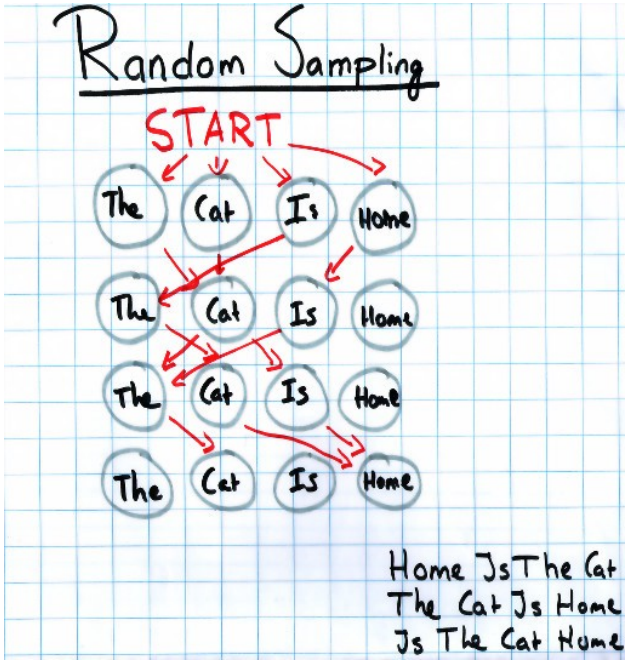


Beam Search with $k = 2$

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Random Sampling

- **Idea**: Sample next word according to output probability distribution

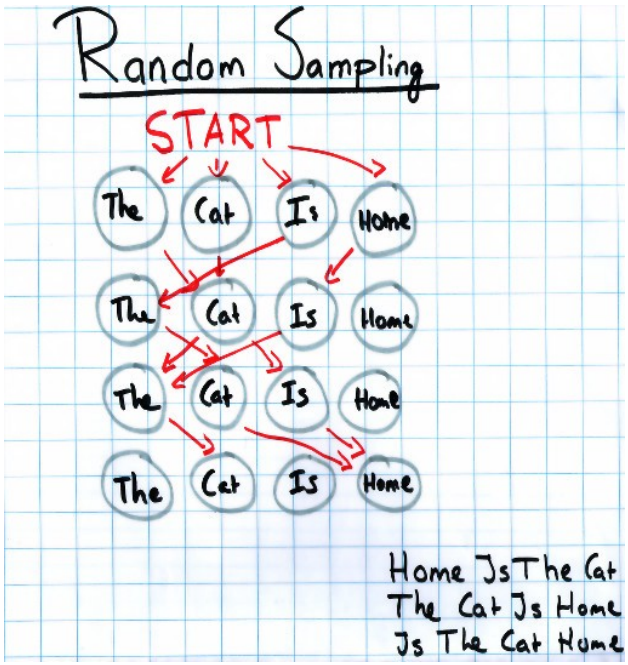


Random Sampling

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Random Sampling

- **Idea**: Sample next word according to output probability distribution
- Example: If “The” has output probability 0.8, it is sampled 8 out of 10 times as the next word

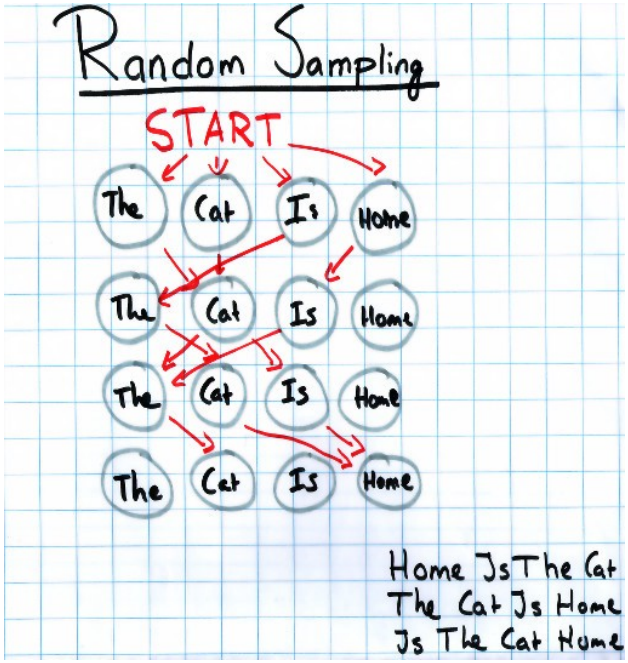


Random Sampling

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Random Sampling

- **Idea**: Sample next word according to output probability distribution
- Example: If “The” has output probability 0.8, it is sampled 8 out of 10 times as the next word
- Creates very diverse results, can look too random



Random Sampling

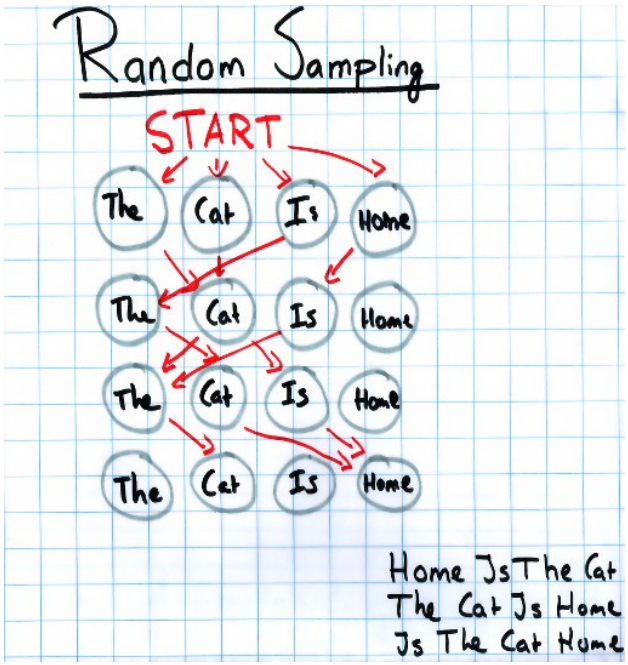
Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Random Sampling

- **Idea**: Sample next word according to output probability distribution
- Example: If “The” has output probability 0.8, it is sampled 8 out of 10 times as the next word
- Creates very diverse results, can look too random
- To reduce randomness, increase/decrease probability of probable/less probable words
  ➔ **Temperature sampling**

$$\tilde{p}_i = f_\tau(p_i) = \frac{p_i^{1/\tau}}{\sum_j p_j^{1/\tau}}$$

p_i: output probability for word i, τ: temperature



Random Sampling

Source: https://medium.com/machine-learning-at-petiteprogrammer/sampling-strategies-for-recurrent-neural-networks-9aea02a6616f

Examples

Character-based Language Modeling with RNNs

- Great blog post (link) by Andrej Karpathy (now director of AI at Tesla)
- Character-level RNN for text generation trained on Shakespeare

Character-based Language Modeling with RNNs

- Great blog post (link) by Andrej Karpathy (now director of AI at Tesla)
- Character-level RNN for text generation trained on Shakespeare
- Example for generated text:

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

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.

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.

VIOLA:
I'll drink it.
```

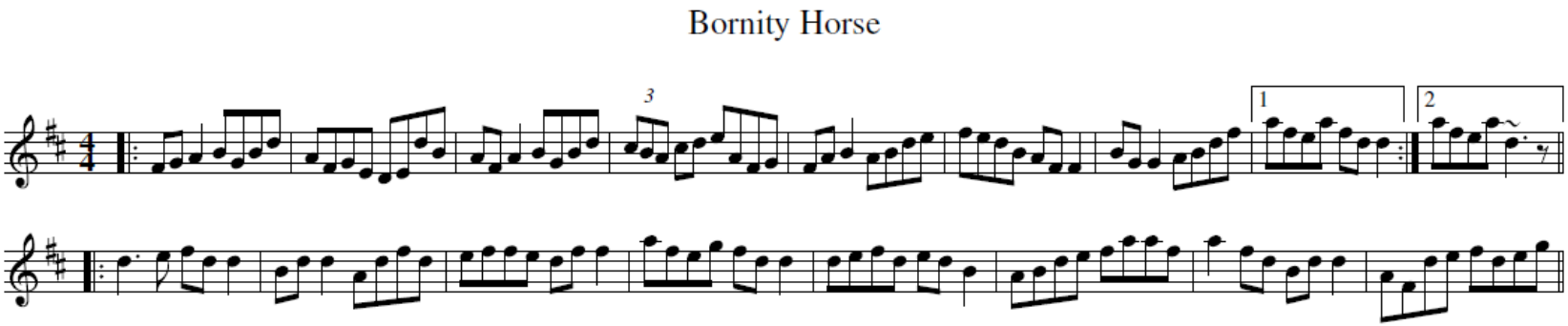
- Experiments with LaTeX, Linux code and Wikipedia entries in the blog post

Composing Folk Music

- Music composition tackled frequently with RNNs (e.g. 1989 by Todd [16], 2002 by Eck and Schmidhuber [4], ...)
- Sturm and Ben-Tal [14] use bigger/deeper networks to generate Folk music
- Character-level RNN using ABC format, including generating title

Composing Folk Music

- Music composition tackled frequently with RNNs (e.g. 1989 by Todd [16], 2002 by Eck and Schmidhuber [4], ...)
- Sturm and Ben-Tal [14] use bigger/deeper networks to generate Folk music
- Character-level RNN using ABC format, including generating title
- Example:



- Audio examples online, e.g., click here

Source: [14]

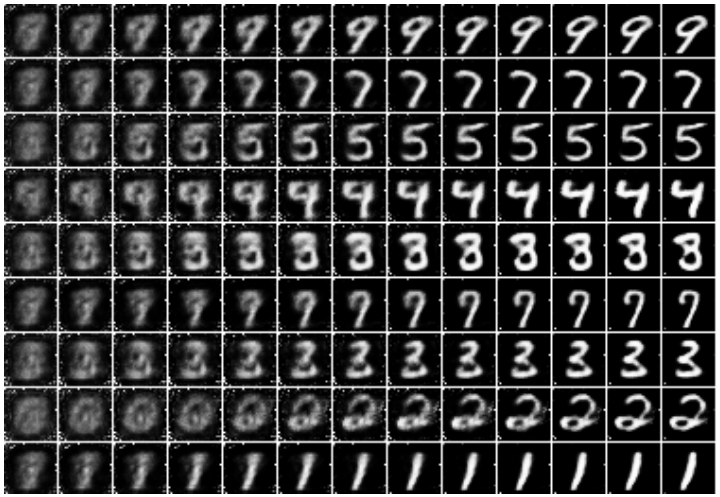
RNNs for Non-Sequencial Tasks

- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image

Source: Adapted from [8]

RNNs for Non-Sequencial Tasks

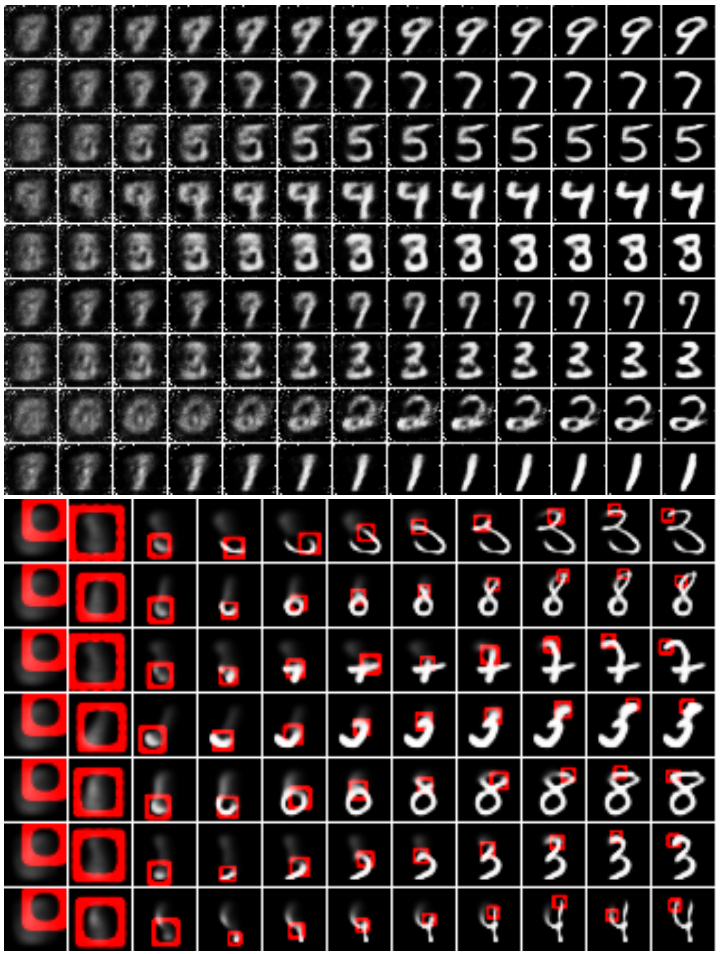
- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image
- Gregor et al. [8]: Drawing numbers
  ➔ from blurry to sharp



Source: Adapted from [8]

RNNs for Non-Sequencial Tasks

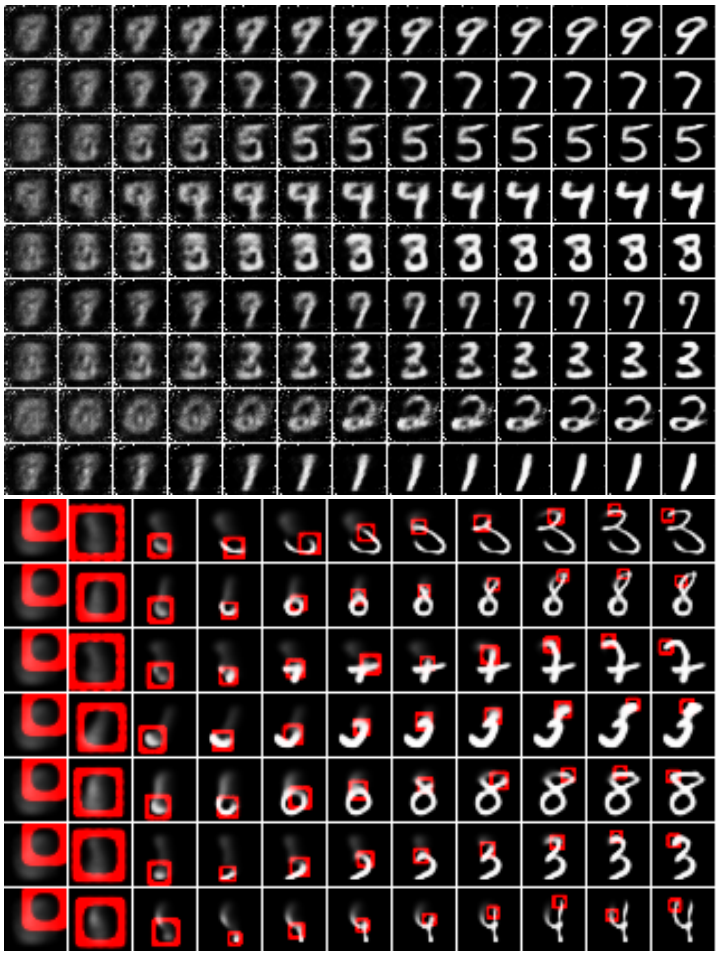
- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image
- Gregor et al. [8]: Drawing numbers
  ➔ from blurry to sharp
- Additional “attention mechanism” telling the network where to look ➔ like brushstrokes



Source: Adapted from [8]

RNNs for Non-Sequencial Tasks

- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image
- Gregor et al. [8]: Drawing numbers
  ➔ from blurry to sharp
- Additional “attention mechanism” telling the network where to look ➔ like brushstrokes
- Uses (variational) **autoencoder** ➔ Lecture 10: Unsupervised Deep Learning



Source: Adapted from [8]

Summary

Summary

- Recurrent neural networks are able to directly model sequential algorithms
- Training via (truncated) backpropagation through time
- Simple units suffer extremely from exploding/vanishing gradients
- LSTM & GRU as improved RNN units that explicitly model “forgetting” and “remembering”

Summary

- Recurrent neural networks are able to directly model sequential algorithms
- Training via (truncated) backpropagation through time
- Simple units suffer extremely from exploding/vanishing gradients
- LSTM & GRU as improved RNN units that explicitly model “forgetting” and “remembering”

We haven’t talked about:

- Memory networks [15], [19]
- Neural turing machines [7]
- Only grazed: Attention + recurrent networks [1], [13]
- ...

NEXT TIME

ON DEEP LEARNING

Coming Up: Visualization

Visualization of ...

- network architecture
- the training process
- the “inner workings” of a network
- neural network “art”.

Attention mechanisms

Comprehensive Questions

- What is the strength of RNNs compared to feed-forward networks?
- What role does the hidden state play in RNNs?
- How do you train RNNs? What are the challenges?
- What is the main idea behind LSTMs and what is the main difference to simple RNN units?
- What are the differences between LSTMs and GRUs?
- In which scenarios would LSTMs be beneficial compared to GRUs?
- Name three applications where many-to-one and one-to-many RNNs would be beneficial.

Further Reading

- Again, the great blog post by Andrej Karpathy on RNNs:
  The Unreasonable Effectiveness of Recurrent Neural Networks
- Blog post by facebook on CNNs for machine translation:
  A novel approach to machine translation
- Blog post on music generation:
  Composing Music With Recurrent Neural Networks

References

References I

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: CoRR abs/1409.0473 (2014). arXiv: 1409.0473.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: IEEE transactions on neural networks 5.2 (1994), pp. 157–166.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: arXiv preprint arXiv:1412.3555 (2014).

[4] Douglas Eck and Jürgen Schmidhuber. “Learning the Long-Term Structure of the Blues”. In: Artificial Neural Networks — ICANN 2002. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 284–289.

References II

[5] Jeffrey L Elman. “Finding structure in time”. In: Cognitive science 14.2 (1990), pp. 179–211.

[6] Jonas Gehring, Michael Auli, David Grangier, et al. “Convolutional Sequence to Sequence Learning”. In: CoRR abs/1705.03122 (2017). arXiv: 1705.03122.

[7] Alex Graves, Greg Wayne, and Ivo Danihelka. “Neural Turing Machines”. In: CoRR abs/1410.5401 (2014). arXiv: 1410.5401.

[8] Karol Gregor, Ivo Danihelka, Alex Graves, et al. “DRAW: A Recurrent Neural Network For Image Generation”. In: Proceedings of the 32nd International Conference on Machine Learning. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1462–1471.

References III

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: arXiv preprint arXiv:1406.1078 (2014).

[10] J J Hopfield. “Neural networks and physical systems with emergent collective computational abilities”. In: Proceedings of the National Academy of Sciences 79.8 (1982), pp. 2554–2558. eprint: http://www.pnas.org/content/79/8/2554.full.pdf.

[11] W.A. Little. “The existence of persistent states in the brain”. In: Mathematical Biosciences 19.1 (1974), pp. 101–120.

[12] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: Neural computation 9.8 (1997), pp. 1735–1780.

References IV

[13] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. “Recurrent Models of Visual Attention”. In: CoRR abs/1406.6247 (2014). arXiv: 1406.6247.

[14] Bob Sturm, João Felipe Santos, and Iryna Korshunova. “Folk music style modelling by recurrent neural networks with long short term memory units”. eng. In: 16th International Society for Music Information Retrieval Conference, late-brea Malaga, Spain, 2015, p. 2.

[15] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, et al. “End-to-End Memory Networks”. In: CoRR abs/1503.08895 (2015). arXiv: 1503.08895.

[16] Peter M. Todd. “A Connectionist Approach to Algorithmic Composition”. In: 13 (Dec. 1989).

[17] Ilya Sutskever. “Training recurrent neural networks”. In: University of Toronto, Toronto, Ont., Canada (2013).

References V

[18] Andrej Karpathy. “The unreasonable effectiveness of recurrent neural networks”. In: Andrej Karpathy blog (2015).

[19] Jason Weston, Sumit Chopra, and Antoine Bordes. “Memory Networks”. In: CoRR abs/1410.3916 (2014). arXiv: 1410.3916.