



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks

**A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle,
K. Packhäuser, M. Zinnen**

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 24, 2023



Outline

Motivation

Simple Recurrent Networks

Long Short-Term Memory Units (LSTMs)

Gated Recurrent Units

Comparison of Simple RNN units, LSTM units and GRUs

Sampling strategies for RNNs

Examples

Summary



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Motivation



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!

OK. Time Context

time dependent
it's

↳ General Context

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, ...) → *stored / batch type*
- 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]



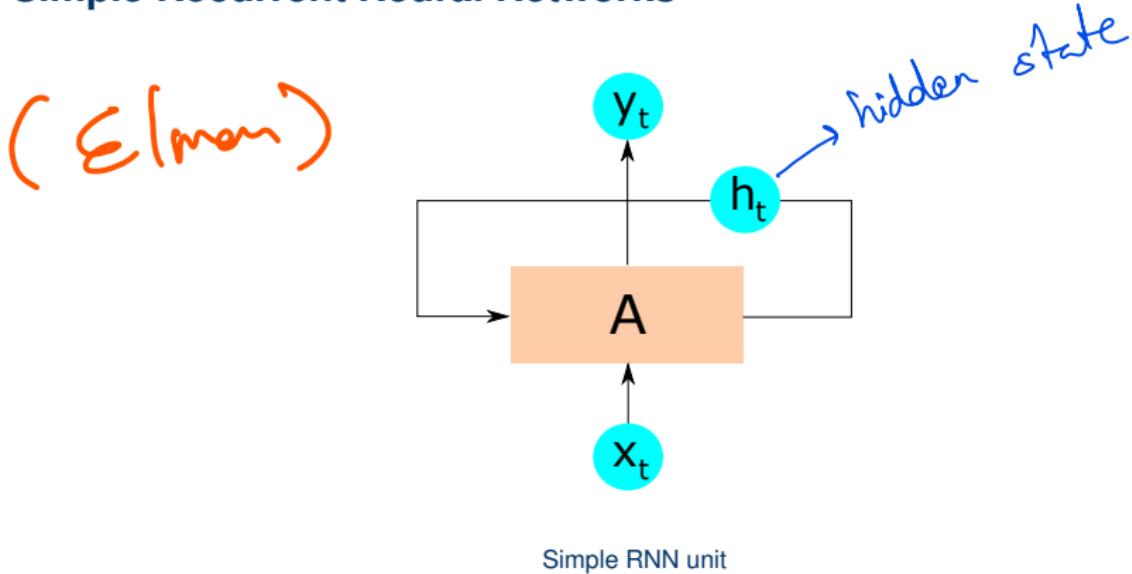
FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Simple Recurrent Networks



Simple Recurrent Neural Networks

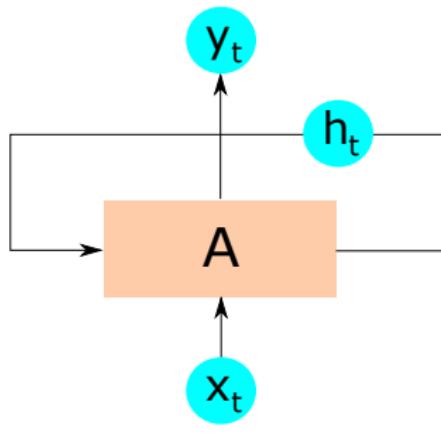


- First models in 1970's [11] and early 1980's [10] (Hopfield Network)
- **Simple recurrent neural network or Elman network** introduced in *Finding Structure in Time* by Jeff Elman in 1990 [5]

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

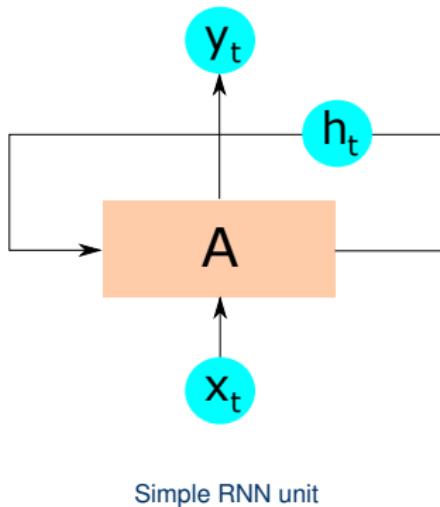
Basic Structure of RNNs



Simple RNN unit

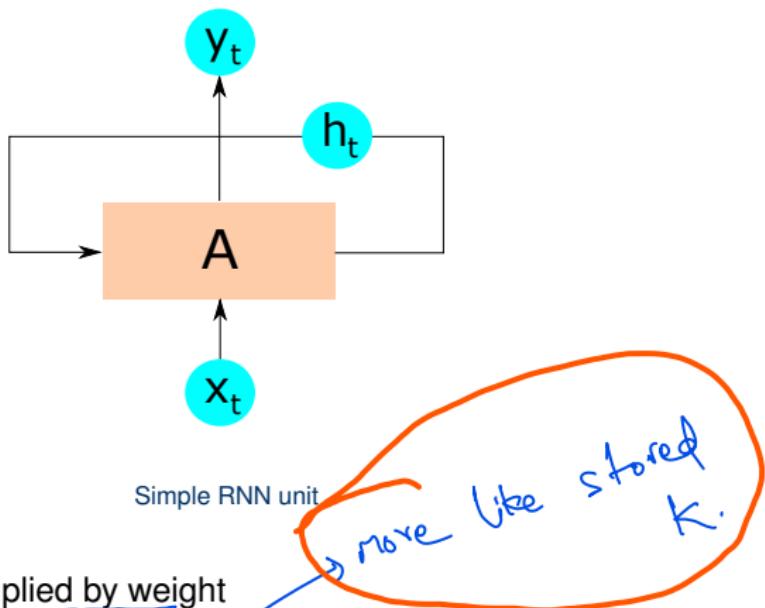
- Current input x_t multiplied by weight

Basic Structure of RNNs



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

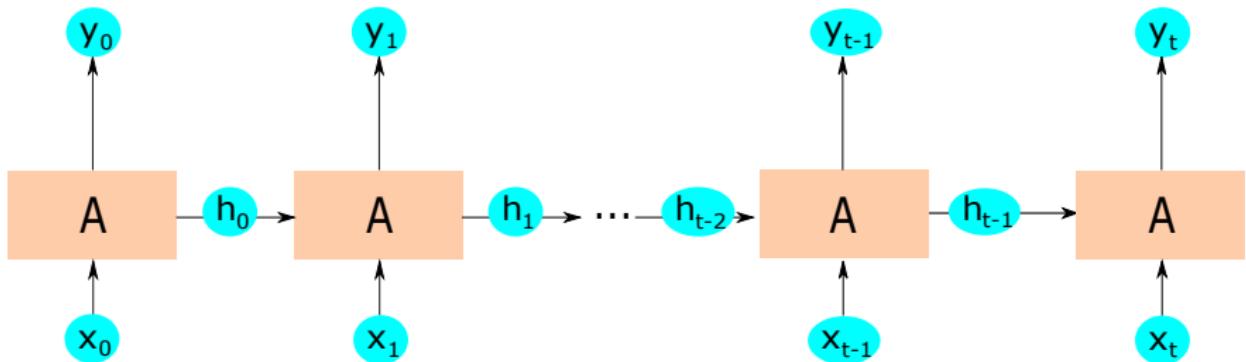
Basic Structure of RNNs



- 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.)

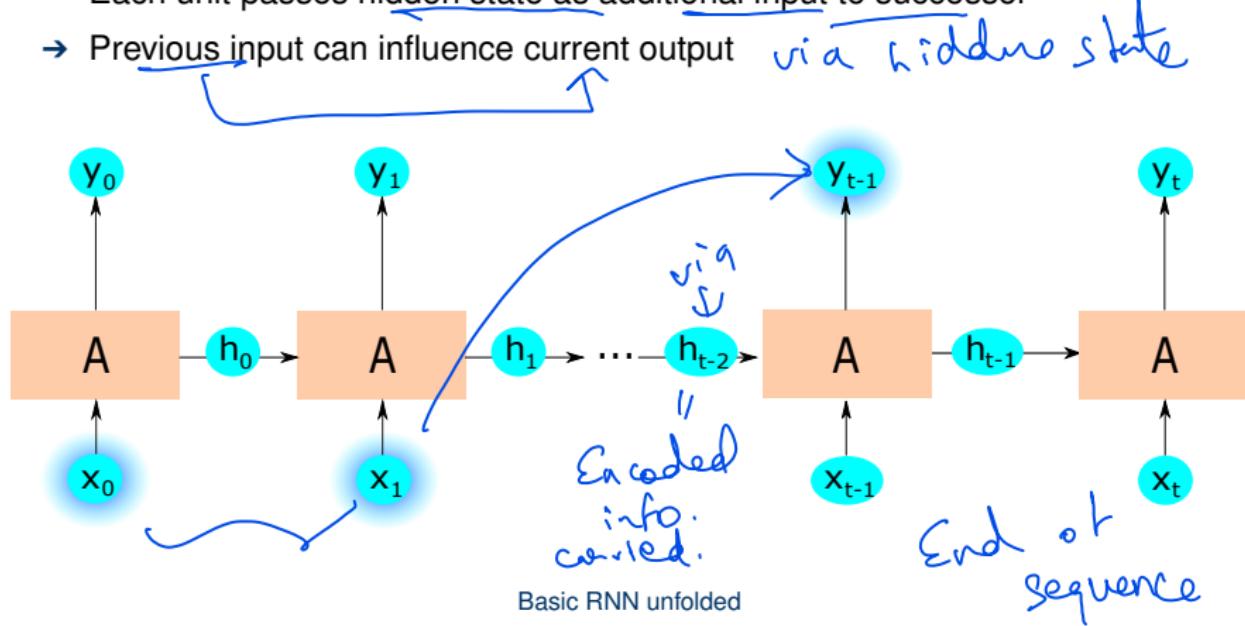
- “Unfolded” RNN unit: sequence of copies of the same unit (= same weights)
- Each unit passes hidden state as additional input to successor



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 *via hidden state*

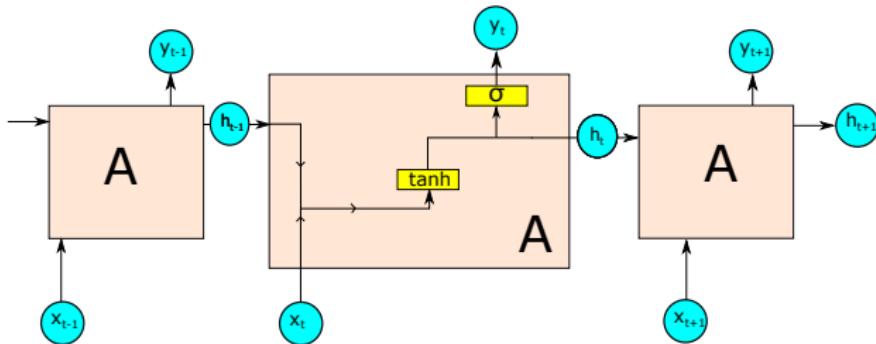


Close-up of a basic RNN unit

- 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?

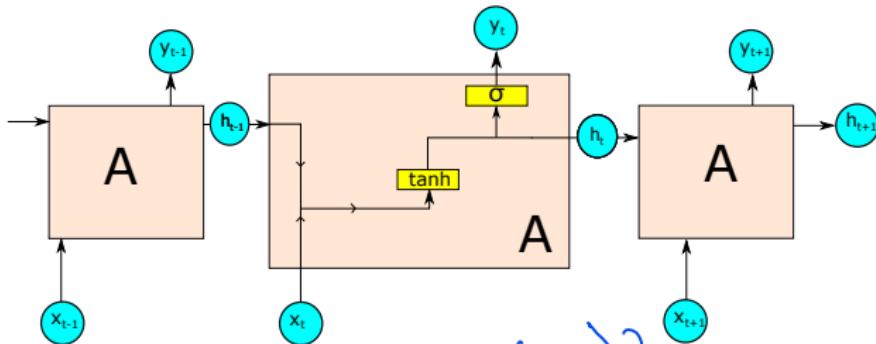


Two activation functions:

- 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?

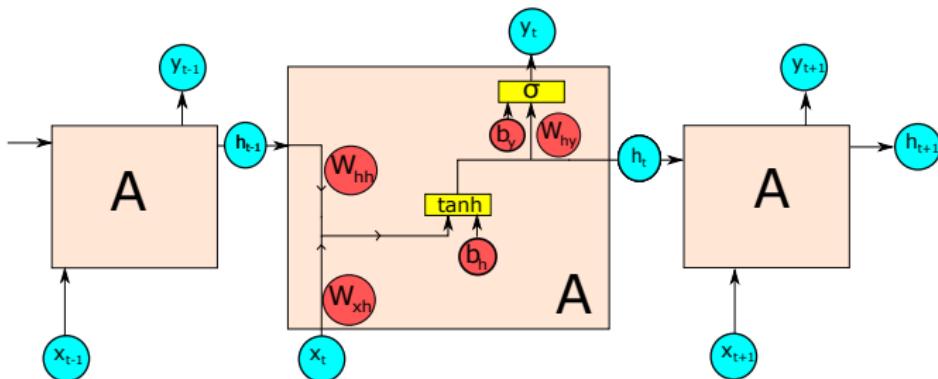


Two activation functions:

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

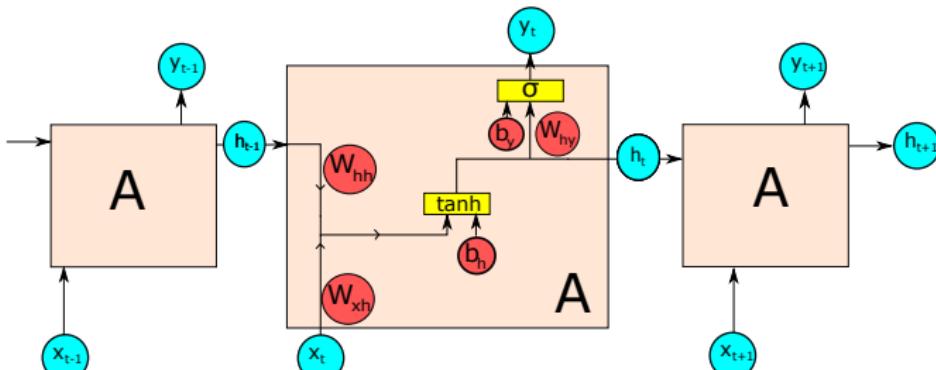
$y_t = \sigma(h_t) ?$
 tanh

Closer-up: How to update the hidden state?



(C.S)

Closer-up: How to update the hidden state?



Update hidden state:

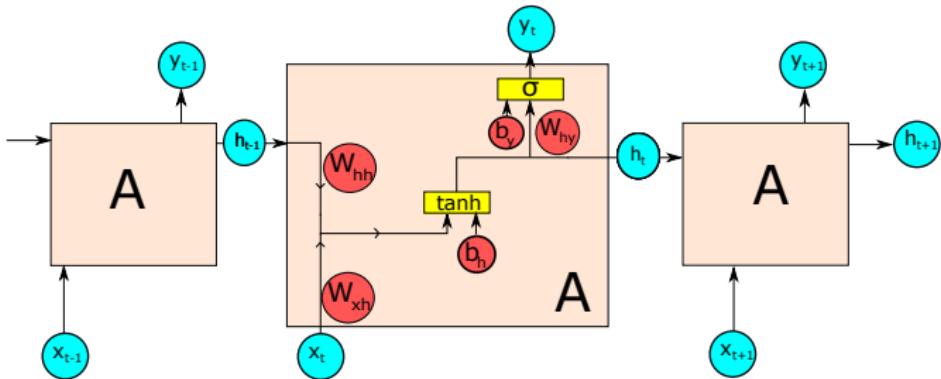
$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_h)$$

W_{hh} : Weight matrix for previous hidden state h_{t-1}

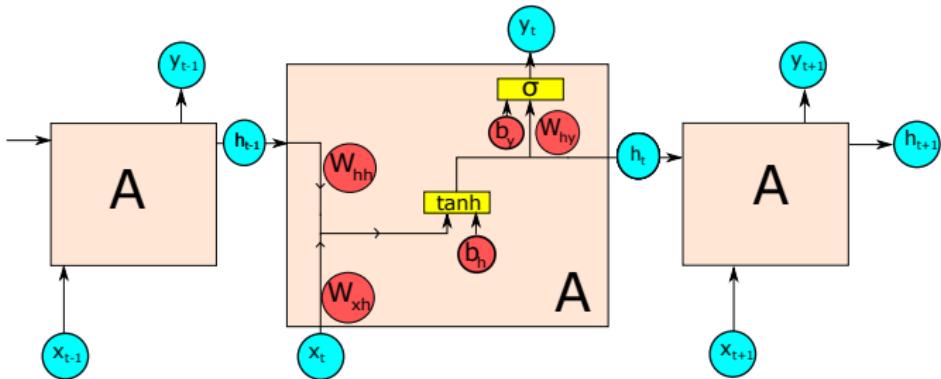
W_{xh} : Weight matrix for current input x_t

b_h : Update bias

Closer-up: How to compute the output?



Closer-up: How to compute the output?



Output formula:

$$\underline{y_t = \sigma (W_{hy} \cdot h_t + b_y)}$$

$\overset{W:\sigma}{\rightarrow}$

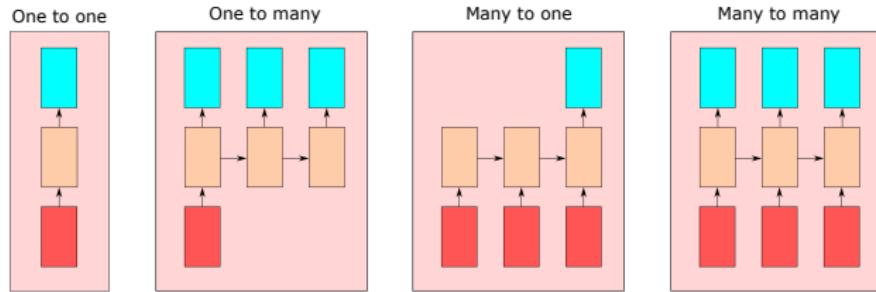
W_{hy} : Weight matrix for current hidden state h_t

b_h : Output bias

b_y . ?

RNN Basic Architectures

Type of sequential relationship \leftrightarrow architecture:

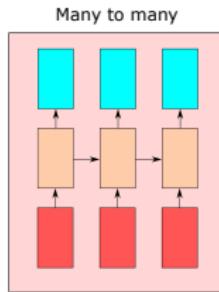
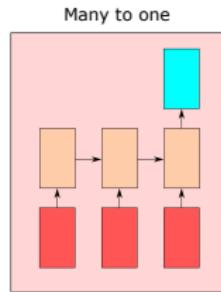
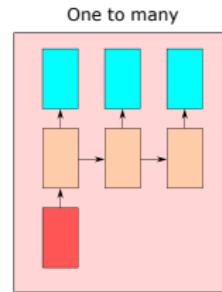
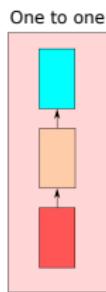


many forms of architecture

RNN Basic Architectures

Type of sequential relationship \leftrightarrow architecture:

o/p



i/p

- Examples:

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

1 image - many words
 one word to describe
 video

Deep RNNs

- So far, only one hidden layer

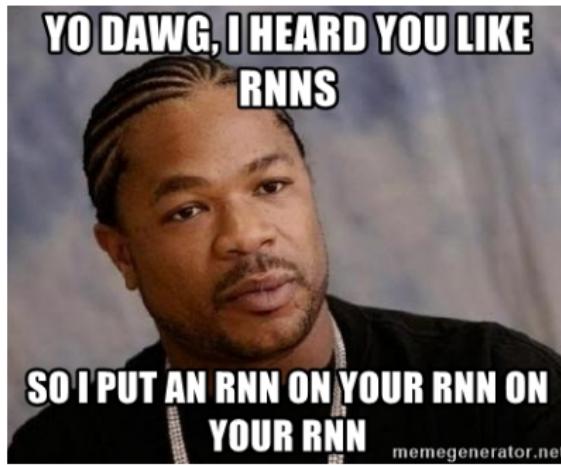
Deep RNNs

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



Deep RNNs

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



Deep RNNs (cont.)

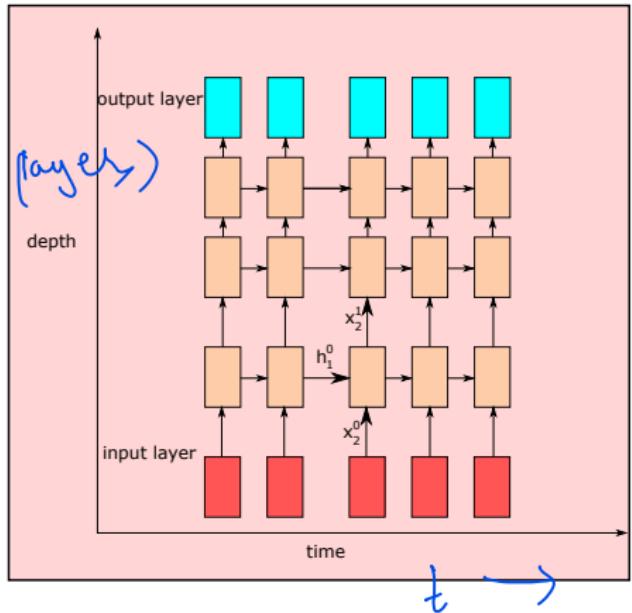
Similar to CNNs, stack multiple units for
deep RNNs

$$h_t^l = \tanh (W_{xh}^l \cdot x_t + W_{hh}^l \cdot h_{t-1}^l + b^l)$$

t: time point

l: layer index

layer layer layer
 $x \rightarrow$ time layer
layer layer layer
 $h_{\text{time}} =$ $+ W_{hh} \cdot h_{t-1}$ $+ \text{bias}$



**NEXT TIME
ON DEEP LEARNING**



FAU

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Recurrent Neural Networks - Part 2

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle,
K. Packhäuser, M. Zinnen

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 24, 2023



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

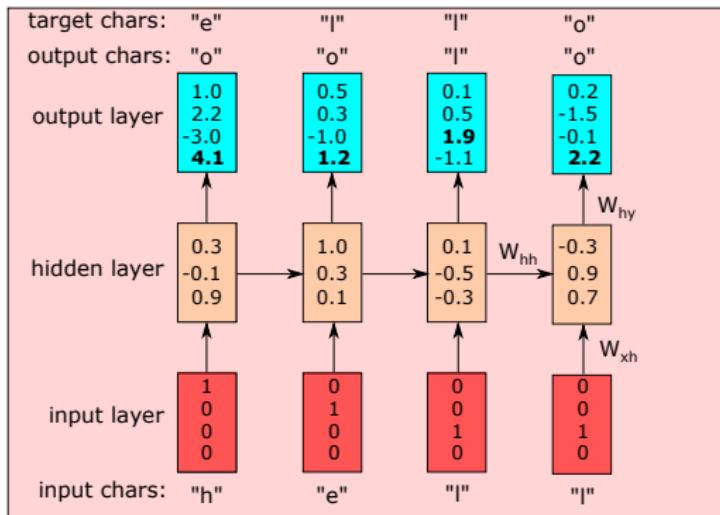
Task: Learn character probability distribution from input text

- 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’?

RNN
 5 lines
 o.l.
 f send
 cede

Simple Example: Character Level Language Model (cont.)

Prediction with random initialization:

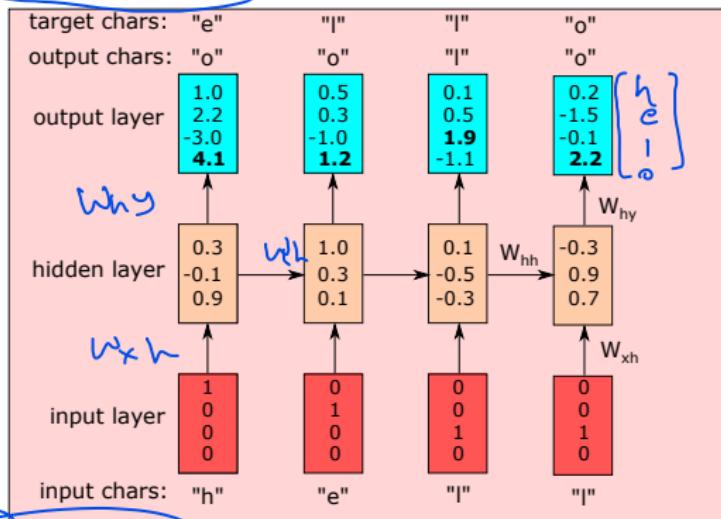


- Goal: Maximize prediction for correct component
- How can we now train this network?

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

Simple Example: Character Level Language Model (cont.)

Prediction with random initialization:



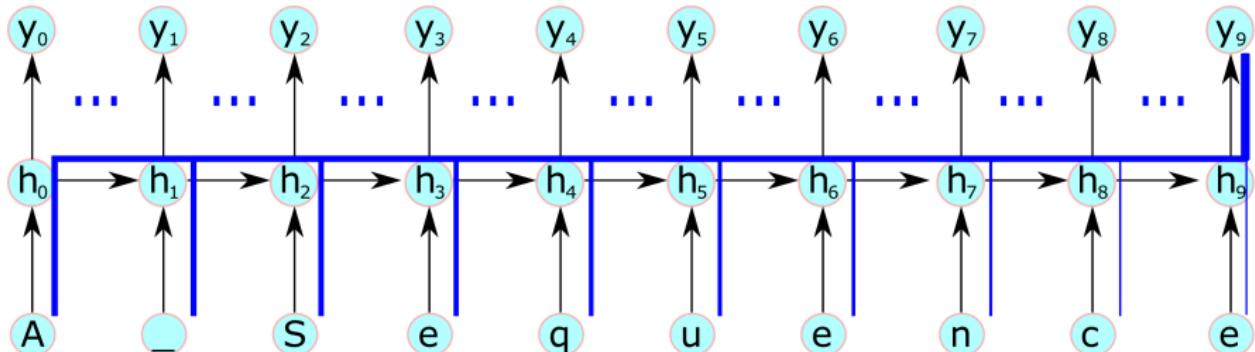
- Goal: Maximize prediction for correct component
- How can we now train this network?
- **Backpropagation through time (BPTT): train “unfolded” network**

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

Backpropagation through Time (BPTT)

Concept: Train the unfolded network

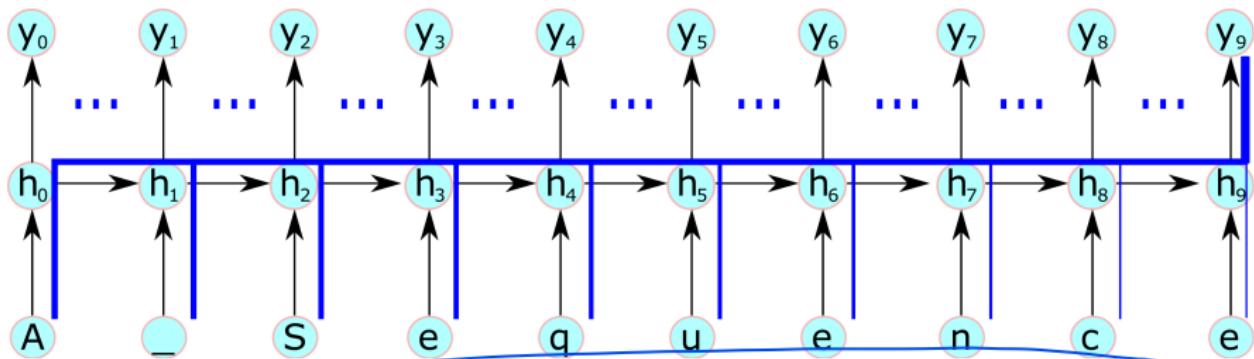
- Compute the forward pass for the full sequence → loss



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

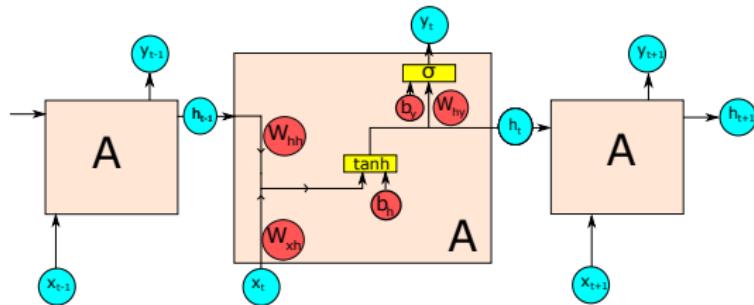


Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output

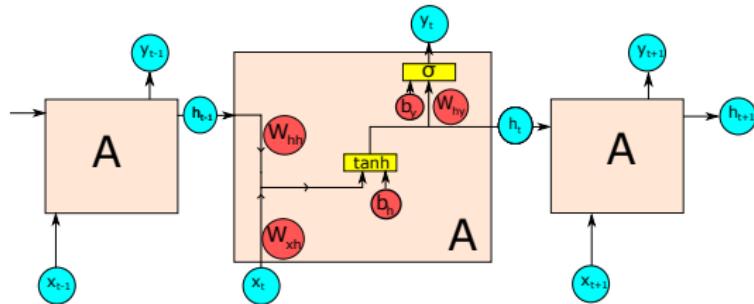


Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$

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

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



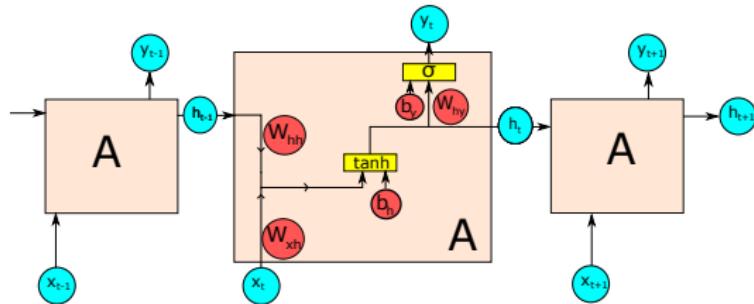
Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$

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

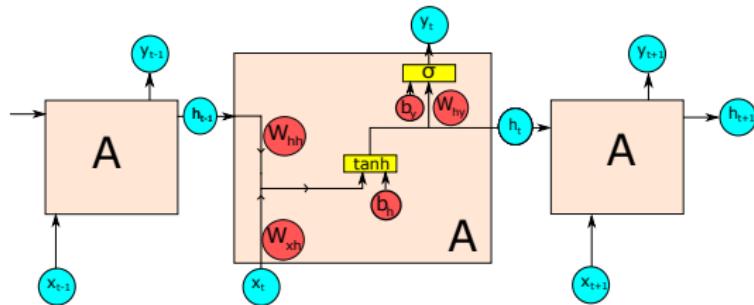


Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$

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

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



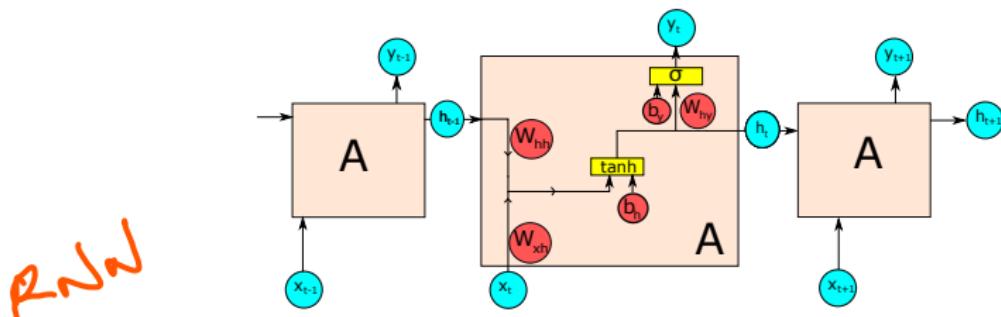
Input sequence: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$

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

(Euh OH)

Backpropagation through Time (BPTT) (cont.)

Forward pass: Computation of hidden states and output



Input sequence: $\mathbf{X} = \{x_1, x_2, \dots, x_T\}$

- ```

1: for t from 1 to T do:
2: $u_t = W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t + b_u$
3: $h_t = \tanh(u_t)$ → next.
4: $o_t = W_{hy} \cdot h_t + b_y$
5: $\hat{y}_t = \sigma(o_t)$

```

} 5 lines of  
pseudo code

## 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  $\eta$

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

Common weights  
 $\Sigma$   
 biases  
 $\nabla \mathbf{h}_0$   
 $T_{\text{initial}} h_0$

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

loss of all  
 $\hat{\mathbf{y}}$  at time  
 t  
 weights  
 bias  
 initial  
 h
- Update parameters using a learning rate  $\eta$

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

- 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  $t = T, \dots, 1$

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

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

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

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

## Backpropagation through Time (BPTT) (cont.)

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

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

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

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

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

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  $t = T, \dots, 1$

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

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

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

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

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

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

## Backpropagation through Time (BPTT) (cont.)

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

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

*reverse order*

at  
time  
 $t$

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

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

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

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

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

(Chain rule)

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

$h_{t+1}$

$o_t$

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

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

$$+ \left( \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \right)^T \nabla \mathbf{o}_t$$

$$+ \mathbf{W}_{hy}^T \nabla \mathbf{o}_t$$

## 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:

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

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

$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

## Backpropagation through Time (BPTT) (cont.)

Updates for other w. matrices

- 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:



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



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

$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

- 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:

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



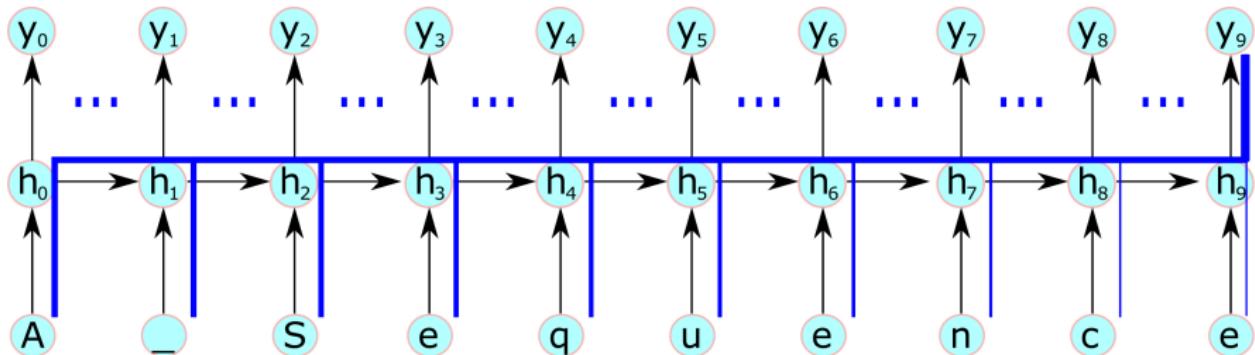
$$\nabla \mathbf{W}_{xh,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t) \cdot \mathbf{x}_t^T$$

$$\nabla \mathbf{b}_{h,t} = \nabla \mathbf{h}_t \cdot \tanh'(\mathbf{u}_t)$$

- 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  $t = \{1, \dots, T\}$ !

sum over

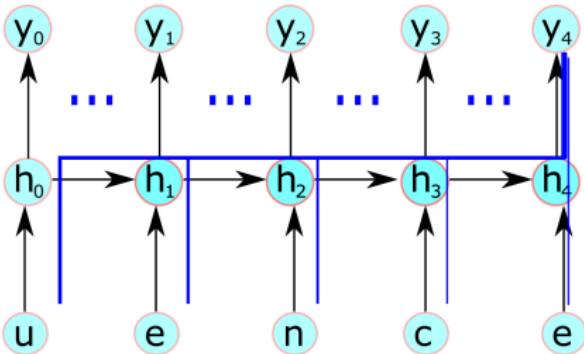
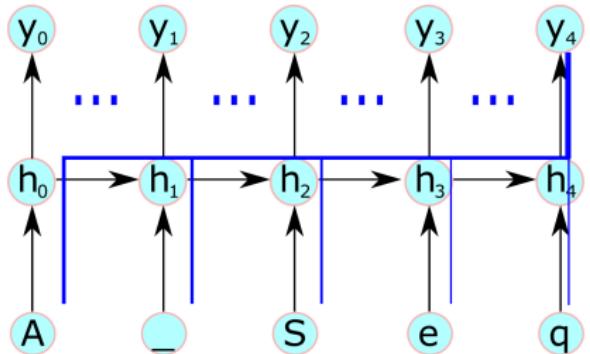
## Normal BPTT



- BPTT: One update requires backpropagation through a **complete sequence**
- Single parameter update is **very expensive!**



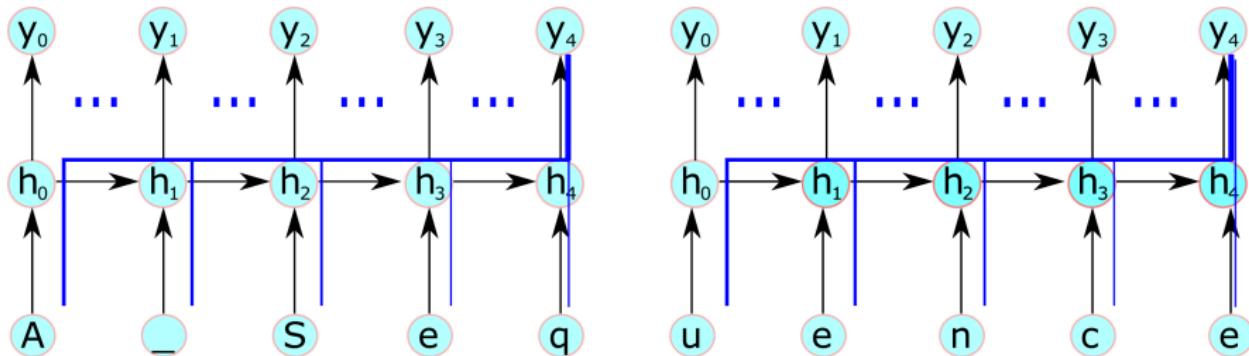
## Naive BPTT



Naive Solution:

- Split long sequences into batches of smaller parts

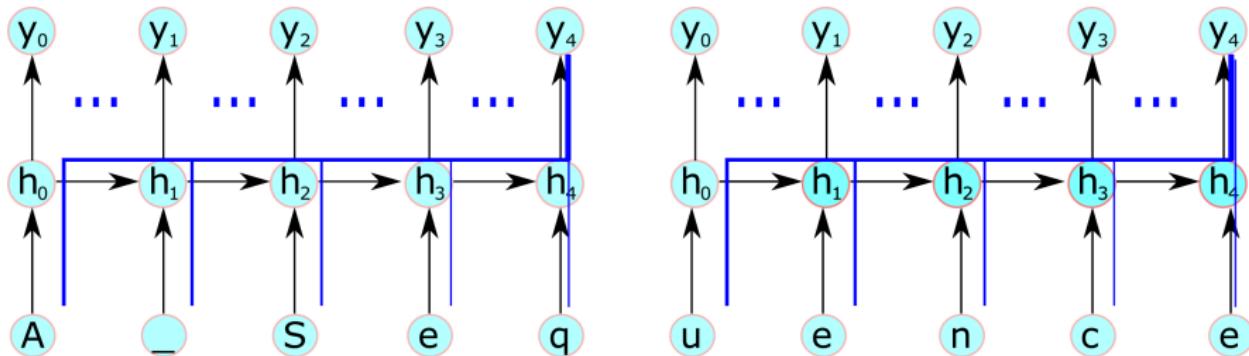
## Naive BPTT



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)**
- Think  
of  
parts
- 1st : lp → not connected  
as part

## 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

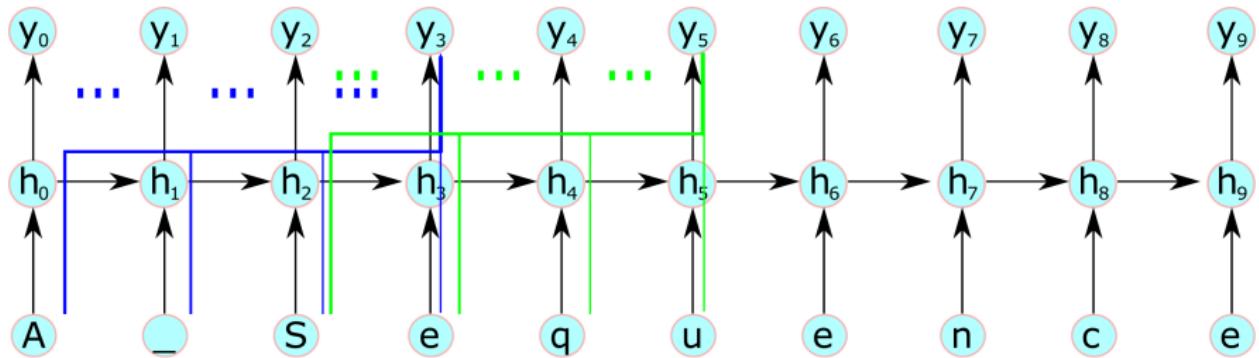
## Truncated Backpropagation through Time (TBPTT)

- 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

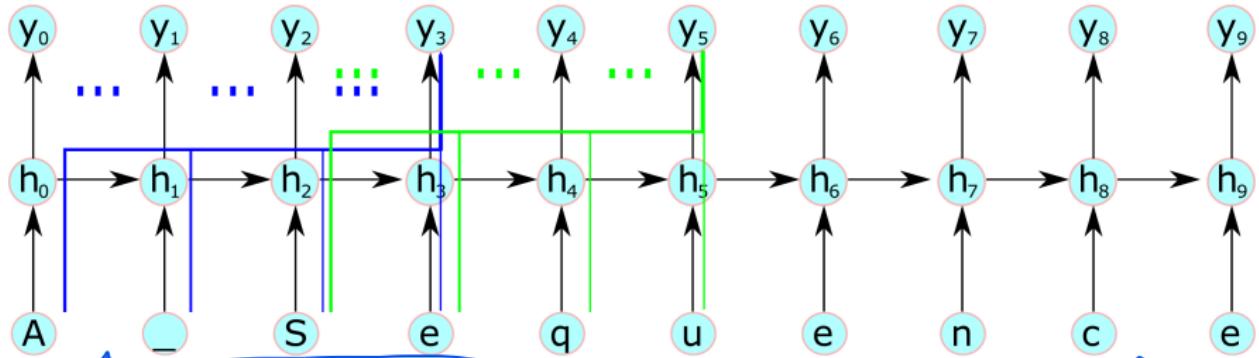
Algorithm:

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

## Truncated BPTT



## Truncated BPTT

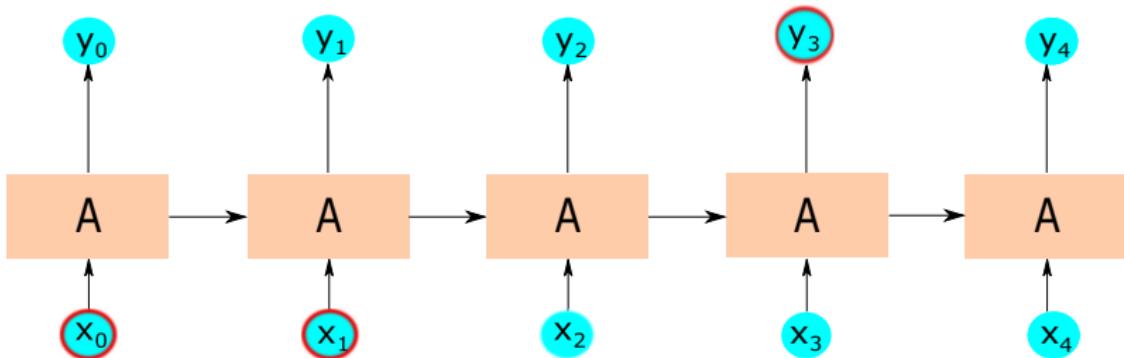


So can we train successful RNNs now? Still no...

long temporal context  
gone.

## The Long-Term Dependency Problem with Basic RNNs

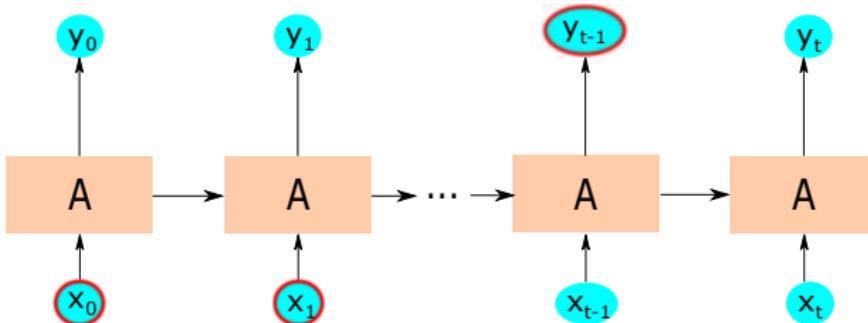
- Short term dependencies work fine
- Example: Predict next word in "the clouds are in the [sky]"



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

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

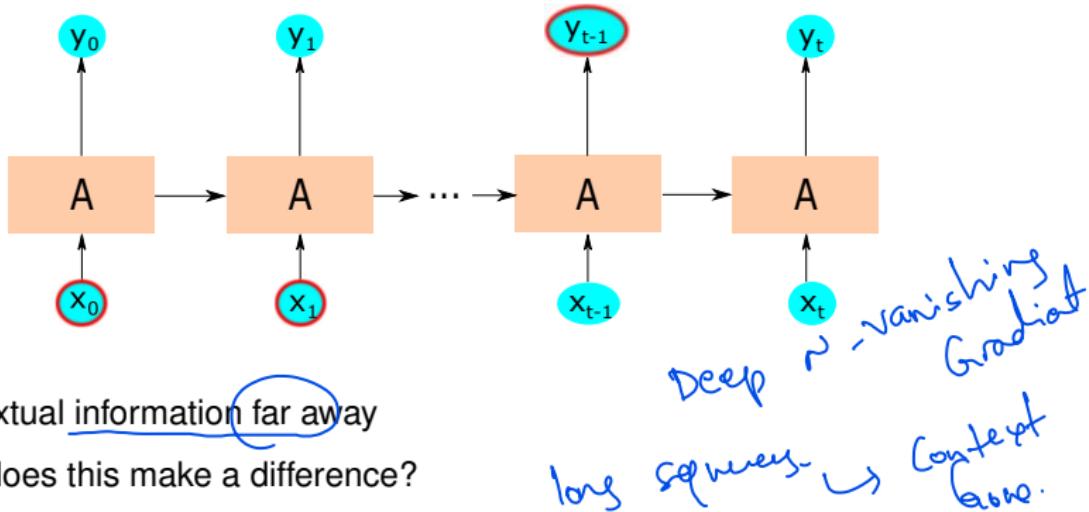
- Harder to connect relevant past and present inputs for longer time spans
- Example: Predict next word in "I grew up in Germany . . . I speak fluent [German]"



- 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

stopping  
it  
influen

- 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**



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Recurrent Neural Networks - Part 3

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle,  
K. Packhäuser, M. Zinnen

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 24, 2023





**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# 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**

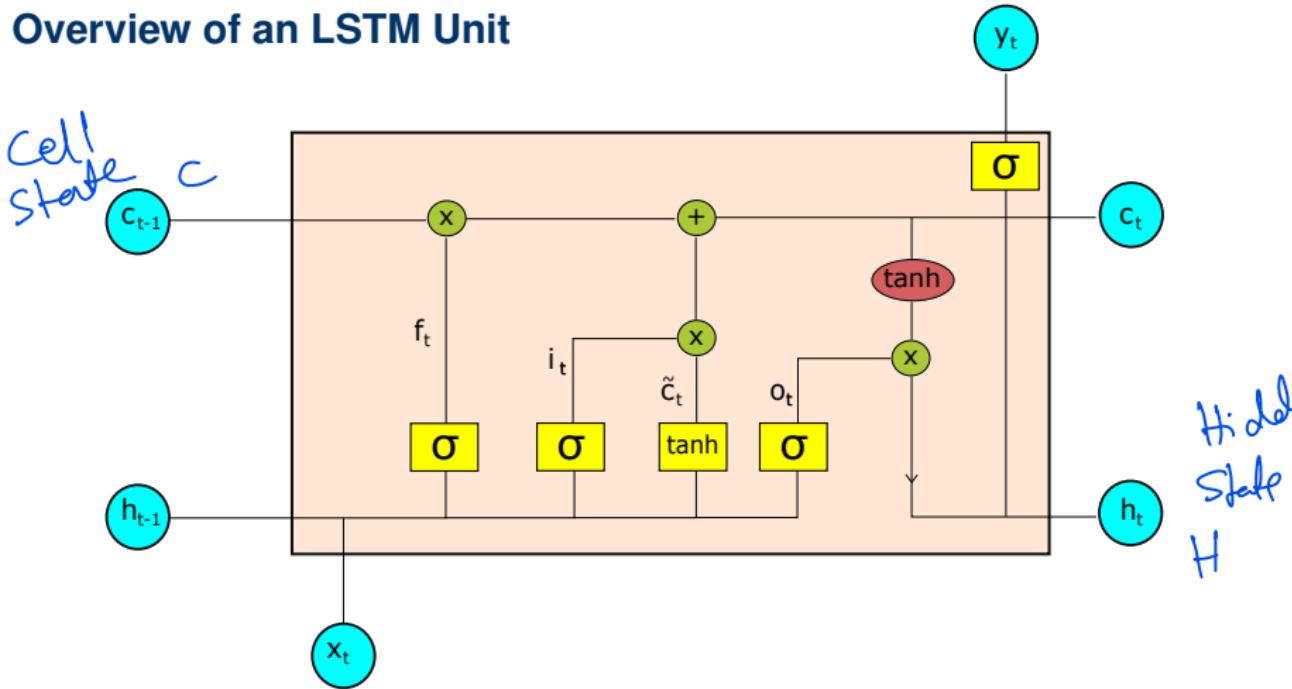


Sepp Hochreiter



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  $\mathbf{y}_t$

→ additional.

Gate:— Control flow of information.

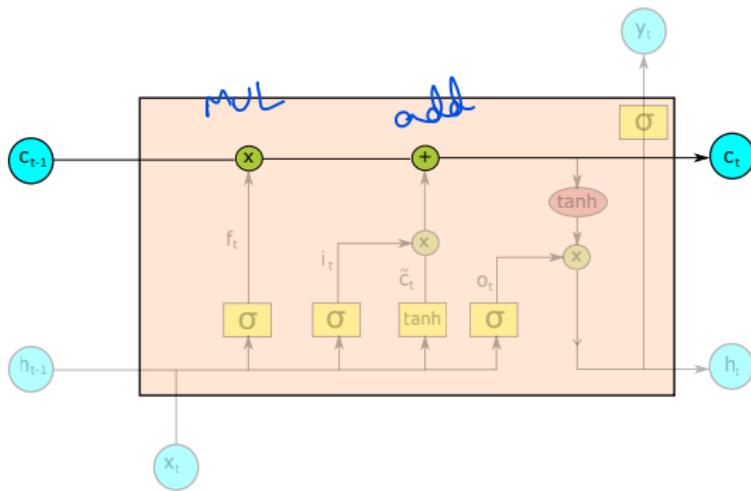
Update of internal states in multiple steps:

- 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

CMAP

## LSTM Cell State

- $c_t$ : **Cell state** after time point  $t$
- Undergoes only **linear changes**; no activation function!
- $c_t$  can flow through a unit unchanged → cell state can be constant for multiple time steps

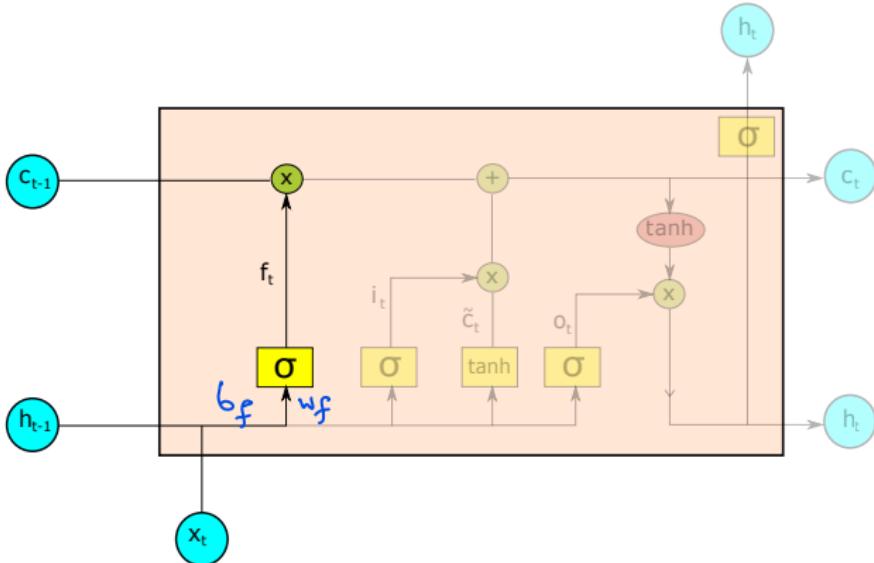


## Forget Gate: Forgetting Old Information

not forget  
= Memorizing

- Key idea: “forgetting” and “memorizing” information in separate steps
- $f_t$  controls how much of the previous cell state is forgotten:

$$f_t = \sigma (\mathbf{w}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$



## Input Gate: Deciding on New Input

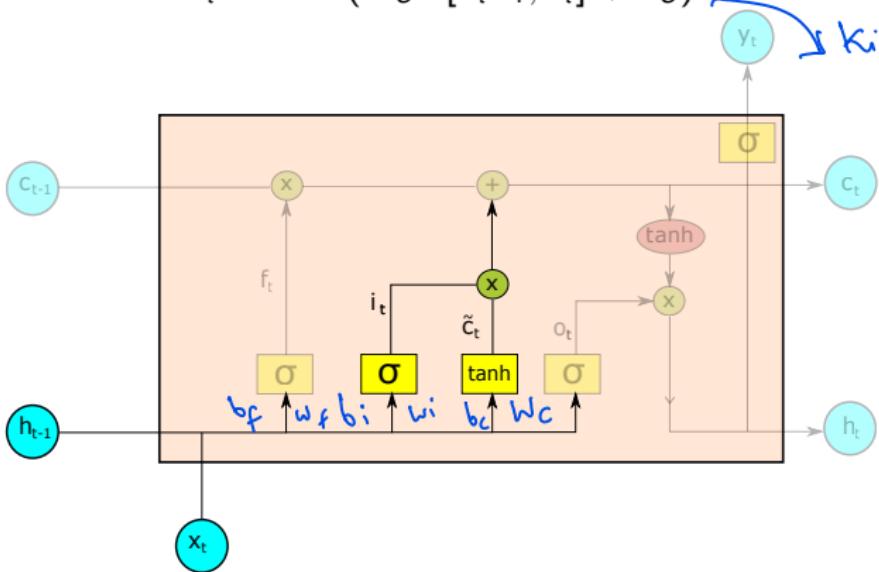
Combination of input and hidden state on two paths:

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh (W_C \cdot [h_{t-1}, x_t] + b_C)$$

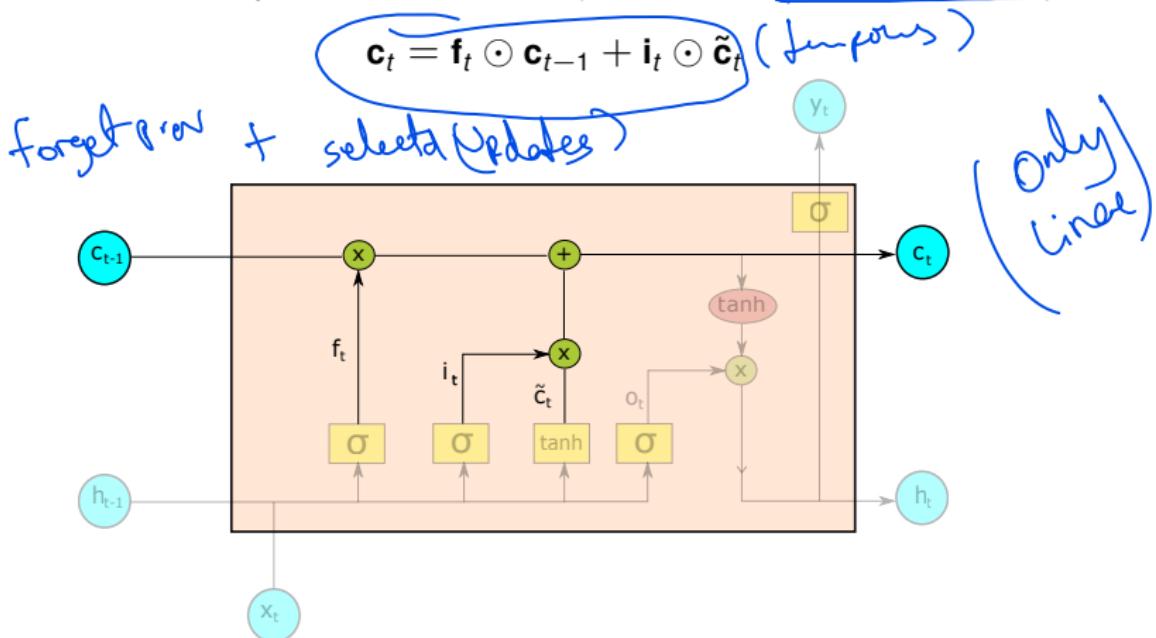
$[0, 1]$  → Sort of selecting.

kind of update state



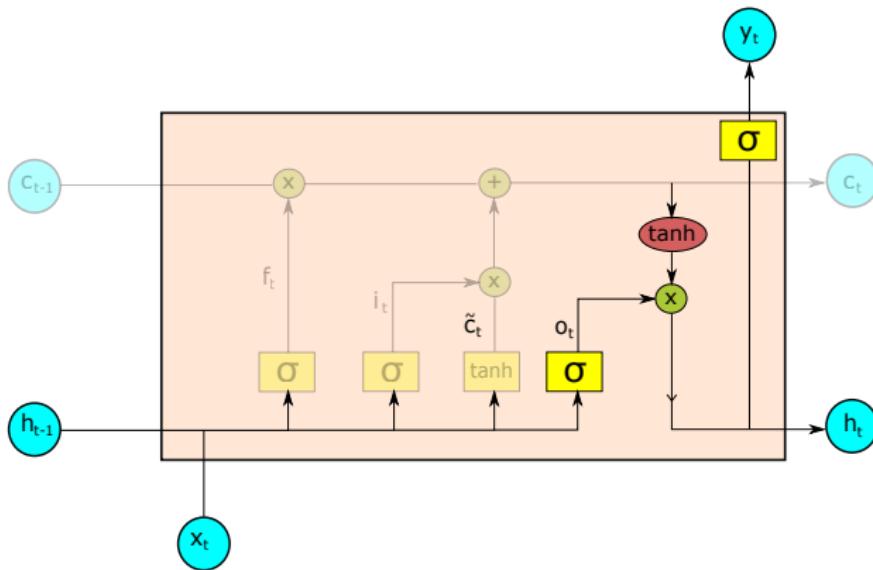
## Updating the Cell State

- New cell state: Sum of “remaining information” from  $c_{t-1}$  and new information from input and hidden state ( $\odot$ : element-wise multiplication)



## Updating the Hidden State and Computing the Output

- Important: Cell state and hidden state are updated **separately**
- Output  $y_t$  directly depends on the hidden state  $h_t$



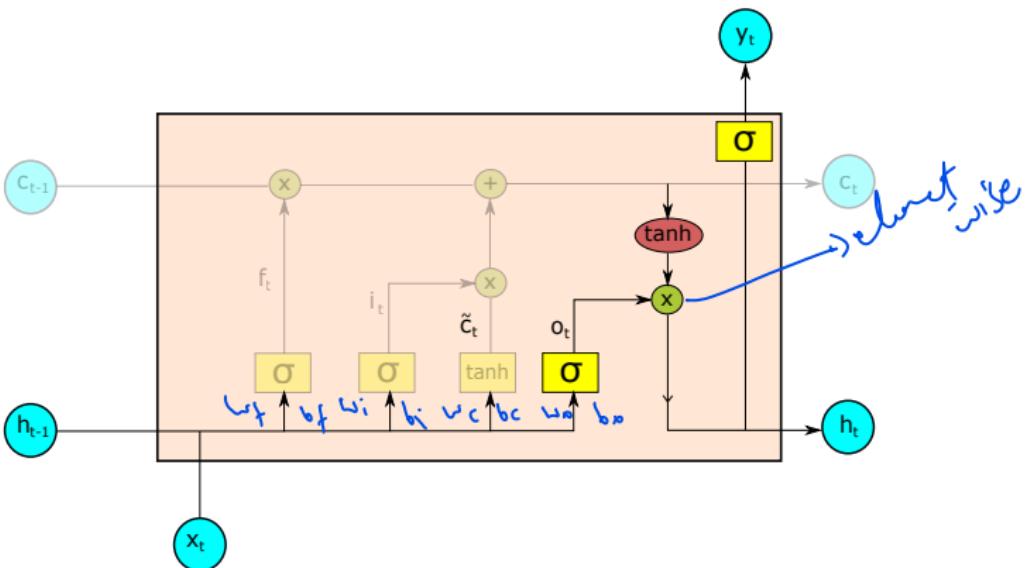
## Updating the Hidden State and Computing the Output (cont.)

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

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

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

Omitted forgetnet or  
graves.



**NEXT TIME  
ON DEEP LEARNING**



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Recurrent Neural Networks - Part 4

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle,  
K. Packhäuser, M. Zinnen

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 24, 2023





**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# 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



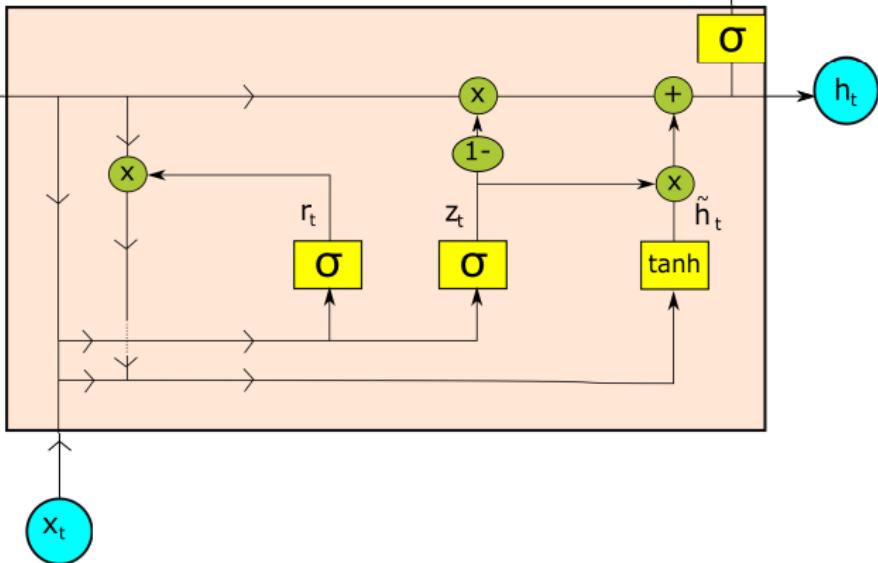
Kyunghyun Cho

## Structure of a GRU cell

Only linear flows

NO

two diff h or C  
variants of 'h'



## 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

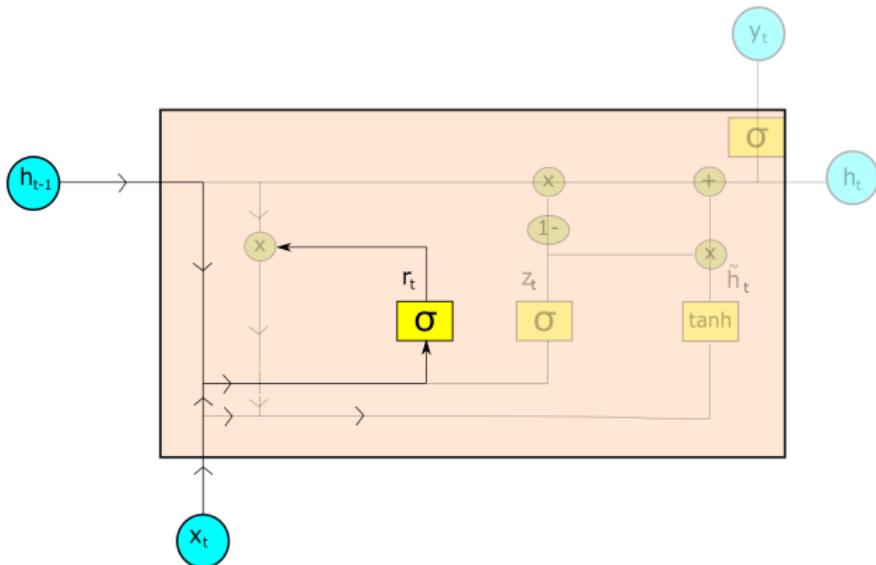
## GRU workflow

- 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

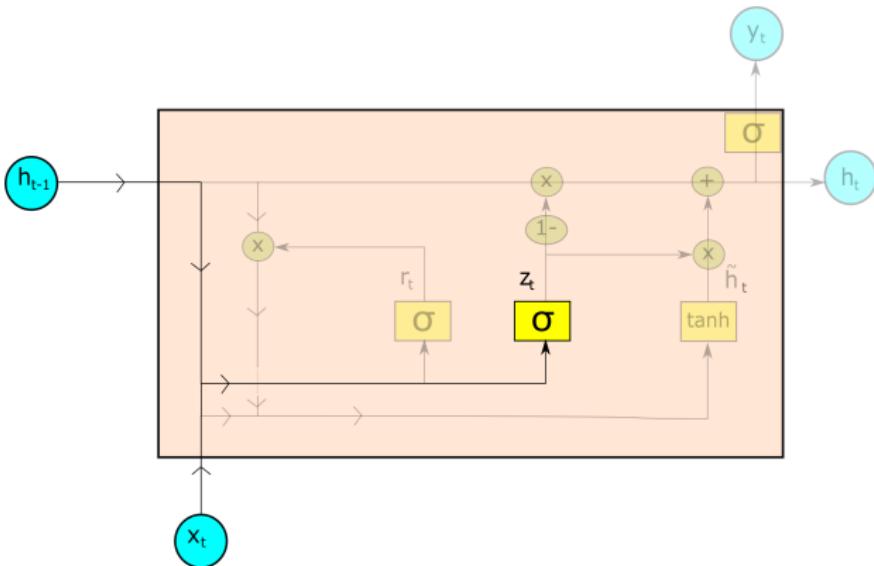
$$r_t = \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r)$$



## Update Gate

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

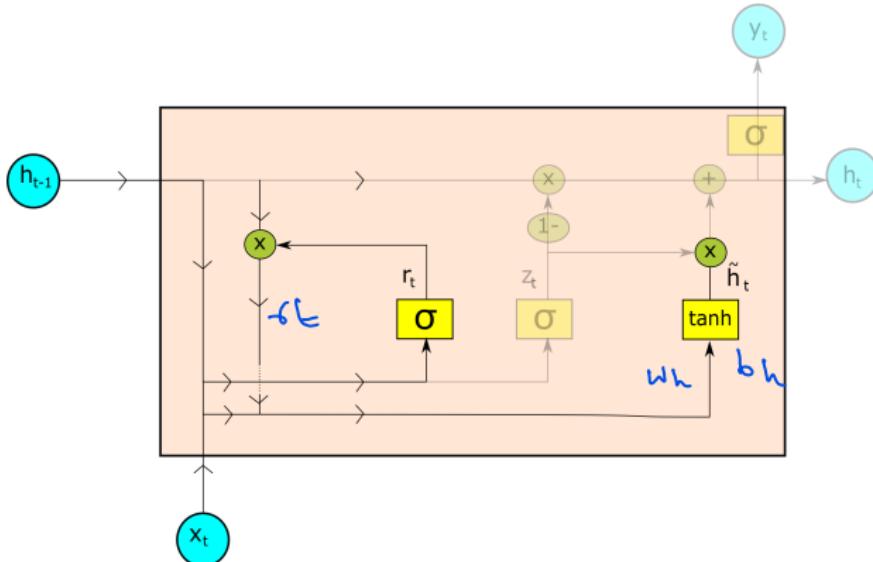
$$z_t = \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z)$$



## Proposing an Update

- Combination of input and “reset” hidden state.
- If  $r_t$  is close to 0 → low influence of previous hidden state

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



## Finally: Computing the Updated Hidden State

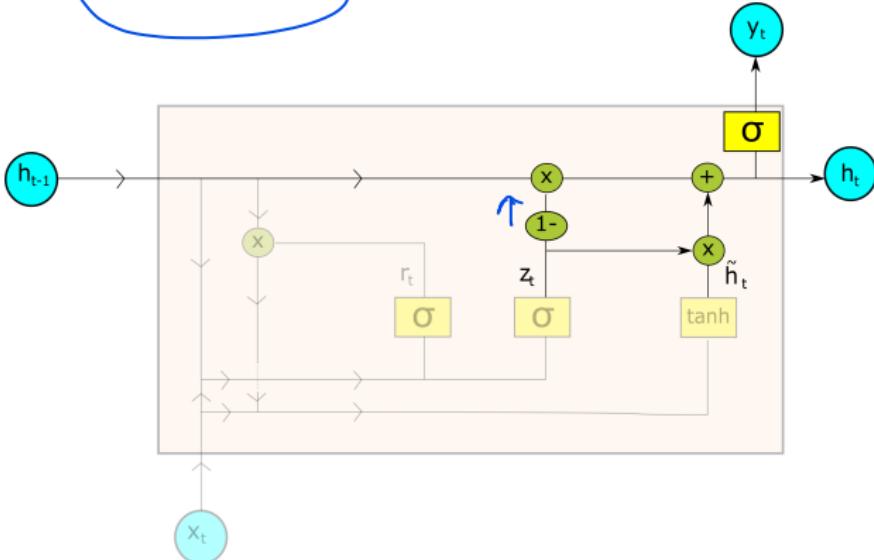
- **Update gate** controls combination of old state and proposed update

( $z_t$ )

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

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

Only  
1 - ADD



## 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  
→  $r_t$  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  
 $\rightarrow r_t$  close to 0: ignore previous hidden state
- Units learning **long-term** dependencies have restrictive update gates  
 $\rightarrow z_t$  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  
→  $r_t$  close to 0: ignore previous hidden state
- Units learning **long term** dependencies have restrictive update gates  
→  $z_t$  close to 0: ignore new input
- Gates have varying "rhythm" depending on the type of information



**FAU**

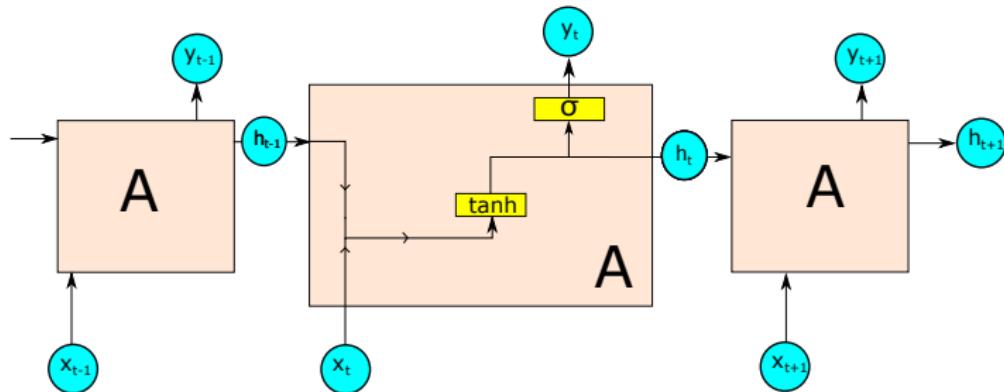
FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Comparison of Simple RNN units, LSTM units and GRUs



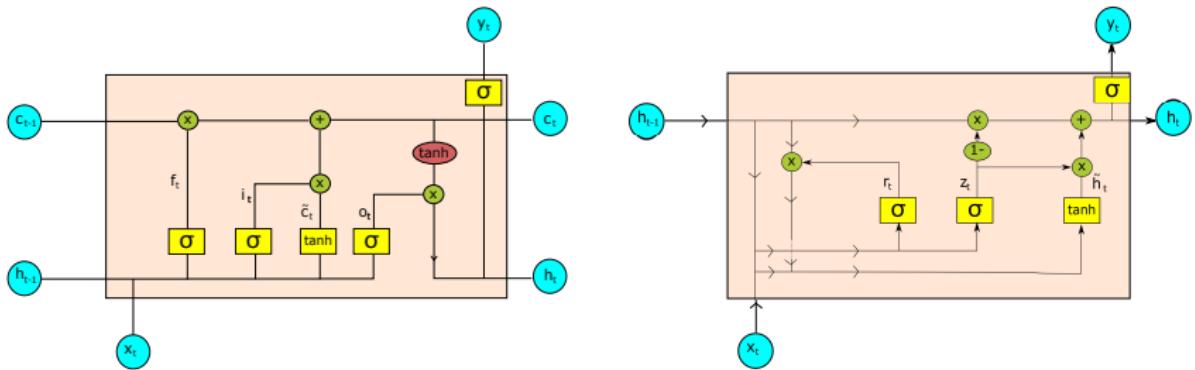
## Recap: Simple RNNs

- 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

Only i add



## 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

[cs]  ≠ = backpropagation

## Advanced structures: LSTM and GRU

### Differences

| LSTM                                                                                                                                              | GRU                                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| Separate 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:<br>$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ <p>→ New memory content independent of current memory</p> | Common update gate:<br>$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ <p>→ New memory content depends on current memory</p> |

## 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**



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# Recurrent Neural Networks - Part 5

A. Maier, V. Christlein, K. Breininger, Z. Yang, L. Rist, M. Nau, S. Jaganathan, C. Liu, N. Maul, L. Folle,  
K. Packhäuser, M. Zinnen

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

April 24, 2023





**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

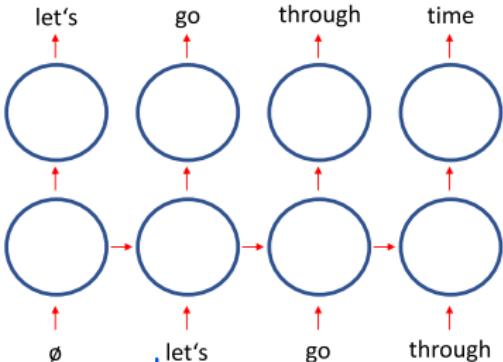
# Sampling strategies for RNNs



*Generate sequences of symbols.*

## Why Sampling Strategies?

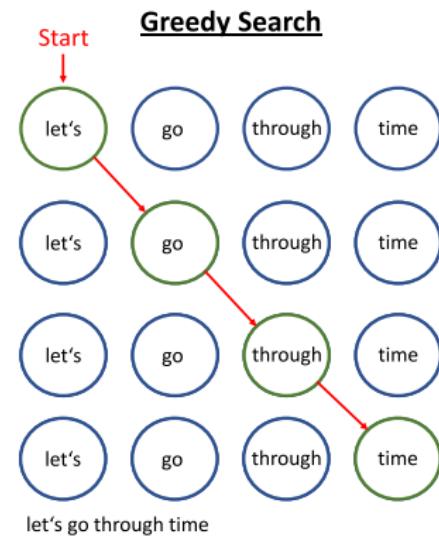
- RNN can generate sequences (words/notes/...)
- RNN actually computes **probability distribution** of the next element
- Question: How do we sample this distribution?



*Ideal sequence - or?*

## Greedy search

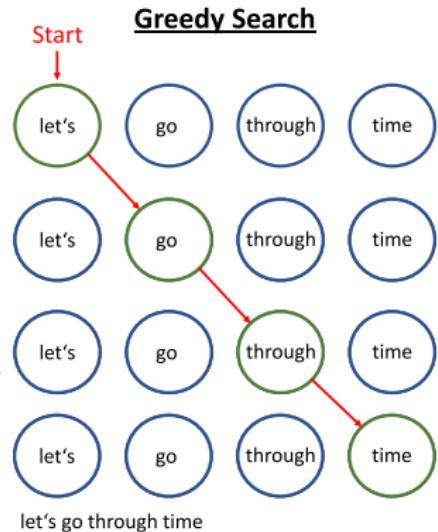
- **Concept:** At each point, pick the most likely element
- Generates exactly one sample sequence per experiment



Greedy search

## Greedy search

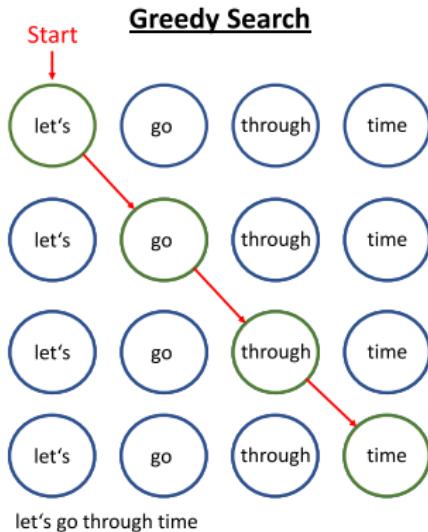
- **Concept:** At each point, pick the most likely element
- Generates exactly one sample sequence per experiment
- **Drawback:** No **lookahead** possible!
  - Example: “let’s” may be most likely after “let’s go” → “let’s go let’s”
  - Cannot detect that “let’s go through time” has a higher total probability



Greedy search

## Greedy search

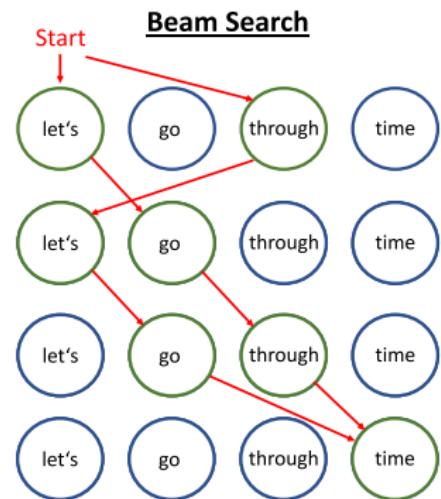
- **Concept:** At each point, pick the most likely element
- Generates exactly one sample sequence per experiment
- **Drawback:** No lookahead possible!
  - Example: "let's" may be most likely after "let's go" → "let's go let's"    *let's so; let's go*
  - Cannot detect that "let's go through time" has a higher total probability
- Tends to repeat sequences of frequent words, e.g., "and", "the", "some" in speech



Greedy search

# Beam Search

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

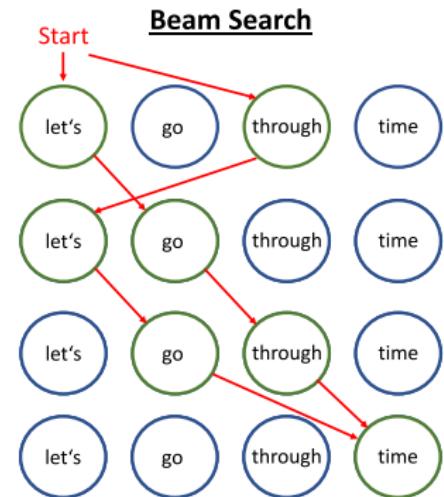


let's go through time  
 through let's go time

Beam Search with  $k = 2$

## Beam Search

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

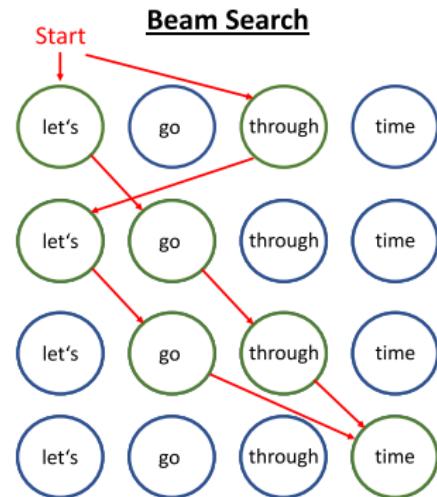


let's go through time  
 through let's go time

Beam Search with  $k = 2$

## Beam Search

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

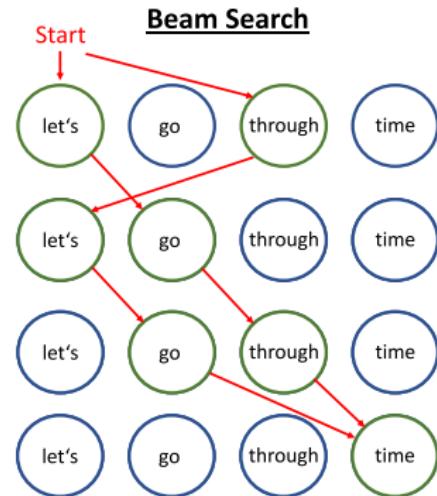


let's go through time  
through let's go time

Beam Search with  $k = 2$

## Beam Search

- **Concept:** Select  $k$  most likely elements ( $k$ : beam width or size)
- Out of all possible sequences that have one of these  $k$  elements 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

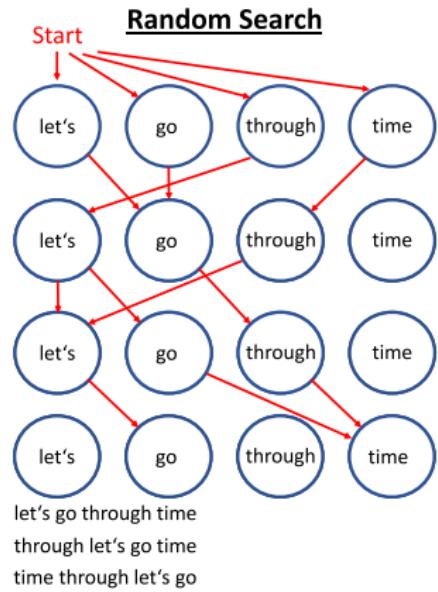


let's go through time  
through let's go time

Beam Search with  $k = 2$

# Random Sampling

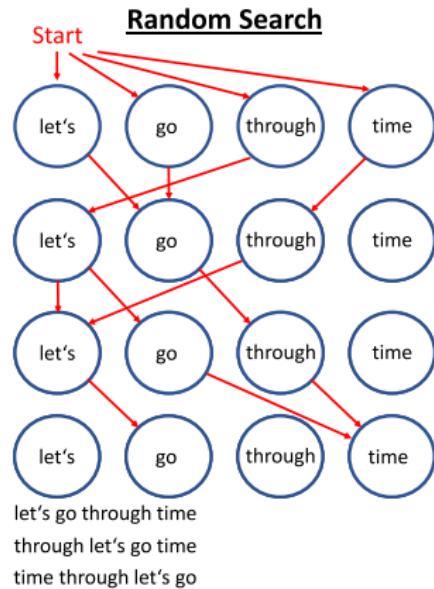
- **Idea:** Sample next word according to output probability distribution



Random Sampling

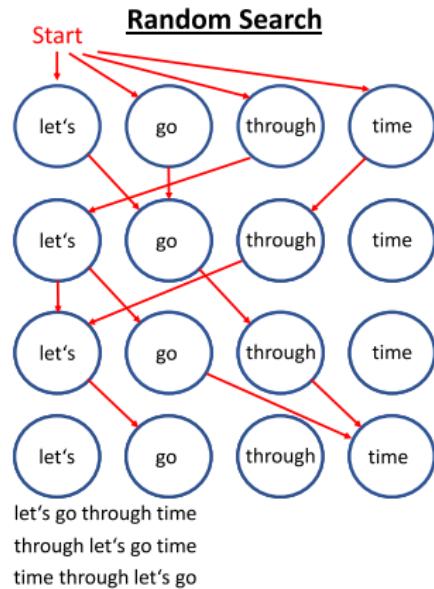
# Random Sampling

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



## Random Sampling

- **Idea:** Sample next word according to output probability distribution
- Example: If “let’s” 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



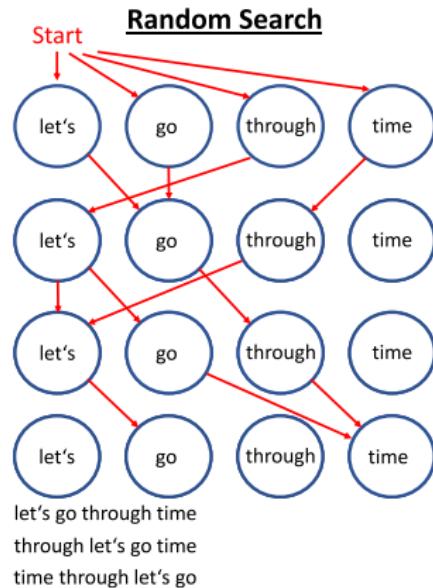
## Random Sampling

- Idea: Sample next word according to output probability distribution
- Example: If "let's" 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  $i$ ,  $\tau$ : temperature



Random Sampling



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# 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 strain 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.

fore ; but still  
non-sense

- Experiments with  $\text{\LaTeX}$ , 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:

Bornity Horse



- 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

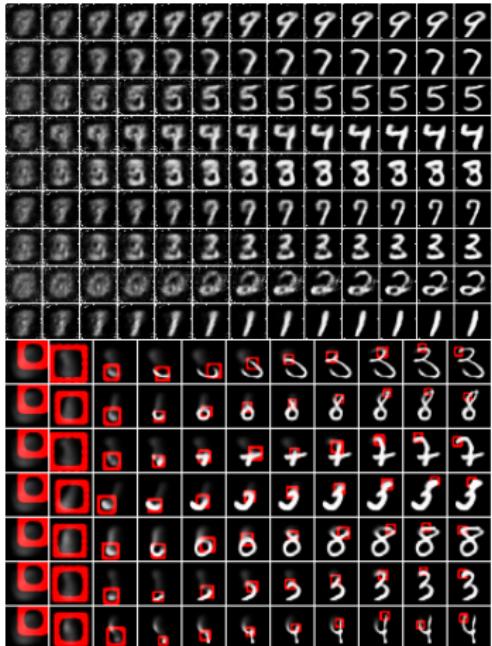
- RNNs can also be used for stationary inputs, e.g., image generation
- Idea: Model progress from rough sketch to final image
- Gregor et al. [8]: Drawing numbers  
→ from blurry to sharp



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

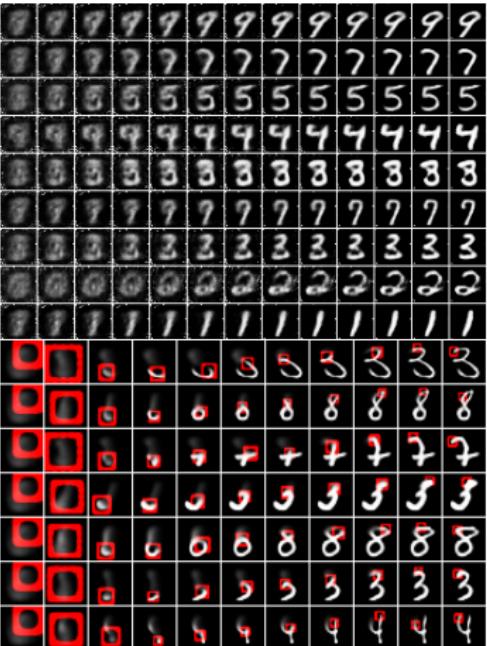


Source: Adapted from [8]

Low feature  
 ↪ High features

## 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]



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# 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? *BPTT* *(par → TBPTT)* → *Model Memory*
- How do you train RNNs? What are the challenges? *i memory* *successive steps*
- What is the main idea behind LSTMs and what is the main difference to simple RNN units? *{CS}* *(STD problem)* *Memory override*
- 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.

*C LSTM*  
*More Control*  
*over memory*  
*longer time, less dependence*

*Text Sentiment analysis:* *→ Image Captioning*  
*video sentiment analysis.* *→ Music*  
*Audio subtit.* *Lyrics.* *Genre*

## 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](#)



**FAU**

FRIEDRICH-ALEXANDER-  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG  
SCHOOL OF ENGINEERING

# 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](https://arxiv.org/abs/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](https://www.pnas.org/content/79/8/2554.full.pdf) 79.8 (1982),  
pp. 2554–2558. eprint:  
[http://www.pnas.org/content/79/8/2554.full.pdf](https://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-breaker](#)  
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.