FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

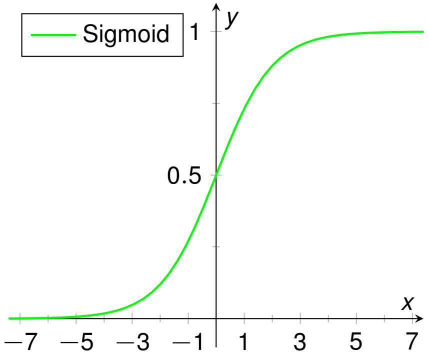
Recurrent Neural Networks

Katharina Breininger, Mingxuan Gu, Noah Maul, Zhaoya Pan, Luca Reeb, Florian Thamm,
Sulaiman Vesal, Tobias Würfl, Zijin Yang
Pattern Recognition Lab, Friedrich-Alexander University of Erlangen-Nürnberg
December 10, 2019

FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Activation Functions

Sigmoid Activation Function



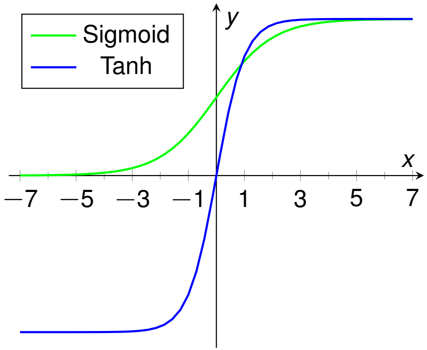
Sigmoid (logistic function)

$$f(x) = \frac{1}{1 + exp(-x)}$$

$$f'(x) = f(x)(1 - f(x))$$

➔ Observe that the derivative can be solely expressed in terms of the activation!

Tanh Activation Function



Tanh

$$f(x) = tanh(x)$$
$$f'(x) = 1 - f(x)^2$$

➔ The derivative is still a function of the activation!

FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Elman Recurrent Neural Network

General strategy

- We interpret the **batch** dimension as **time** dimension now

General strategy

- We interpret the **batch** dimension as **time** dimension now

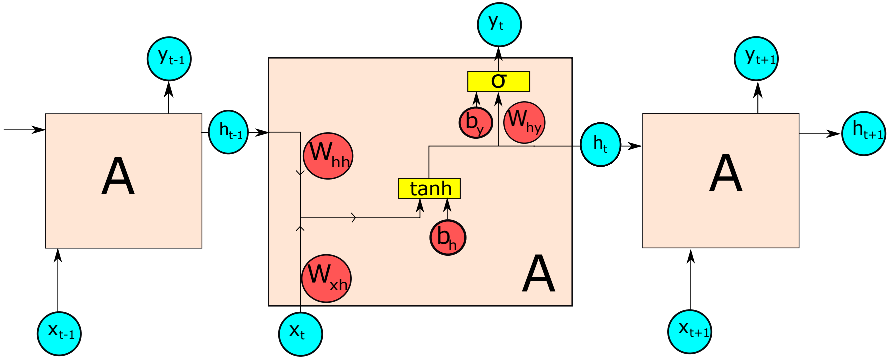
➔ Samples are correlated in this dimension

General strategy

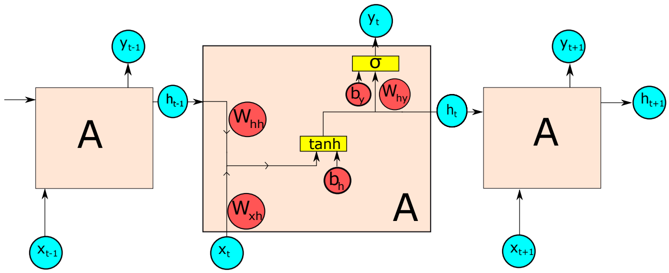
- We interpret the **batch** dimension as **time** dimension now

➔ Samples are correlated in this dimension

- This allows to **reuse** loss functions, optimizers, initializers, activation functions and the Neural Network class


y_{t-1}
y_t
y_{t+1}
h_{t-1}
h_t
h_{t+1}
A
A
A
σ
b_y
W_{hy}
W_{hh}
tanh
b_h
W_{xh}
x_{t-1}
x_t
x_{t+1}

Elman RNN Cell



Output formula:

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

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

$\mathbf{b}_h$: Output bias

A word on software engineering

- In terms of **encapsulation** - how good was the idea to demand exposition of the weights as member?

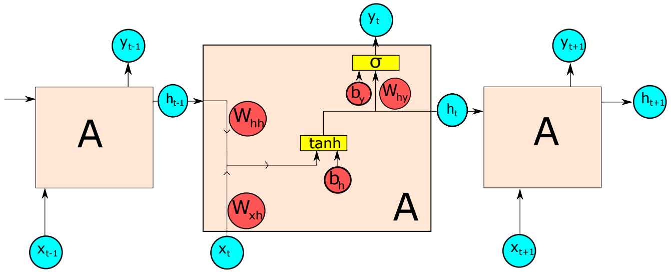
A word on software engineering

- In terms of **encapsulation** - how good was the idea to demand exposition of the weights as member?
- Suppose we implement the RNN cell as **composite** structure
- **Getters** and **Setters** provide us the flexibility to do so

A word on software engineering

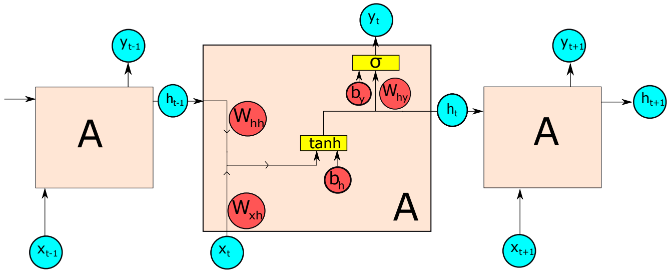
- In terms of **encapsulation** - how good was the idea to demand exposition of the weights as member?
- Suppose we implement the RNN cell as **composite** structure
- **Getters** and **Setters** provide us the flexibility to do so
- Takeaway? Not doing **proper software engineering** most of the time will demand a price at some point.

Elman RNN Cell



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

Elman RNN Cell



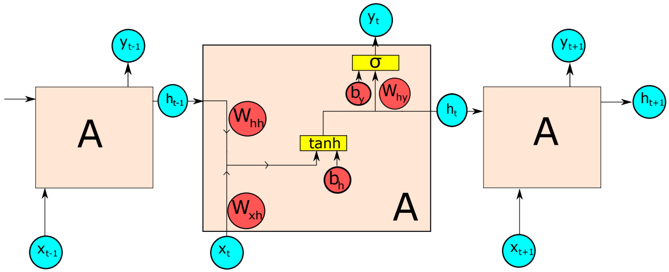
$$\mathbf{h}_t = \tanh\left(\mathbf{h}_{t-1} \cdot \mathbf{W}_{hh} + \mathbf{x}_t \cdot \mathbf{W}_{xh} + \mathbf{b}_h\right)$$

$\mathbf{W}_{hh}$: 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

Elman RNN Cell

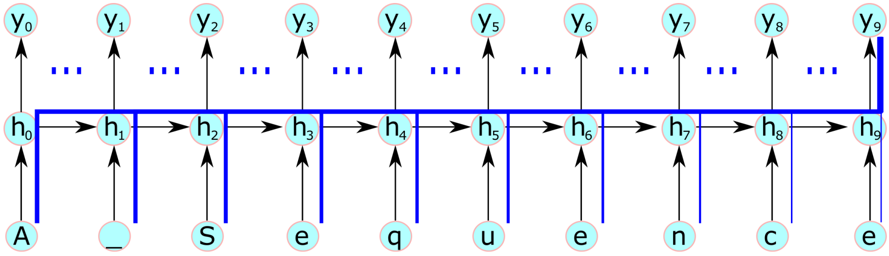


$$\mathbf{h}_t = \tanh\left(\tilde{\mathbf{x}}_t \cdot \mathbf{W}_h\right)$$

$\mathbf{W}_h$: Weight matrix of a fully connected layer

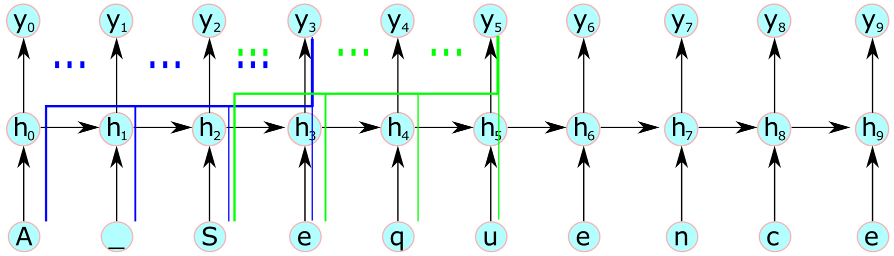
$\tilde{\mathbf{x}}_t$: Concatenation of $\mathbf{x}_t$, $\mathbf{h}_{t-1}$ and a 1

Different from output: Not processed independently!



y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9
...
h_0 h_1 h_2 h_3 h_4 h_5 h_6 h_7 h_8 h_9
A _ S e q u e n c e

- Implemented by passing the whole sequence as a **batch**

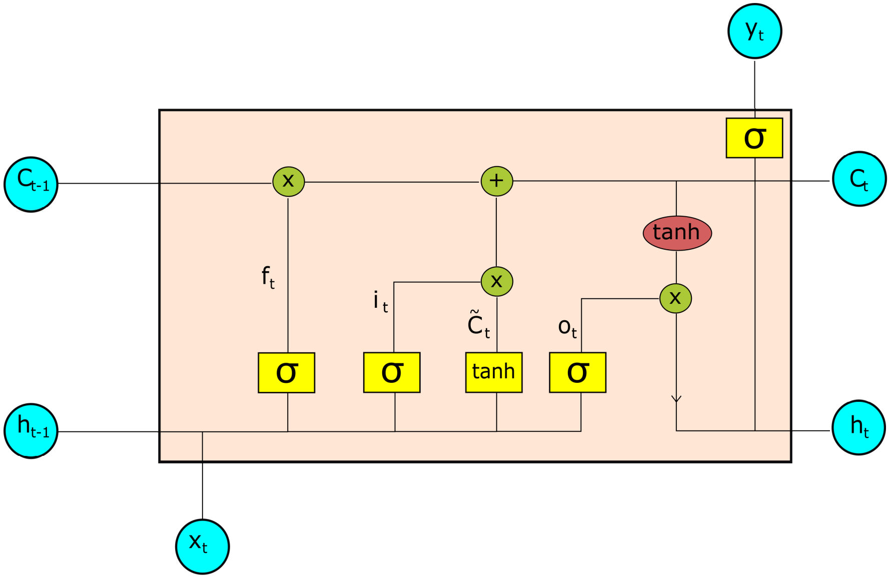


- Implemented by passing **overlapping** parts as a **batch**
- We need to implement memory **Between states**
- Simply store the **last hidden state** and implement a **method** switching whether this state is reused in subsequent forward passes.
- Data has to be fed in **accordingly**!
- Referencing the TBPPT Algorithm presented in the lecture: k_1 is always the sequence length and k_2 is always the TBPTT length.

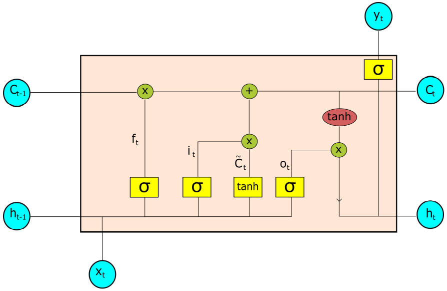
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SCHOOL OF ENGINEERING

Long Short-Term Memory

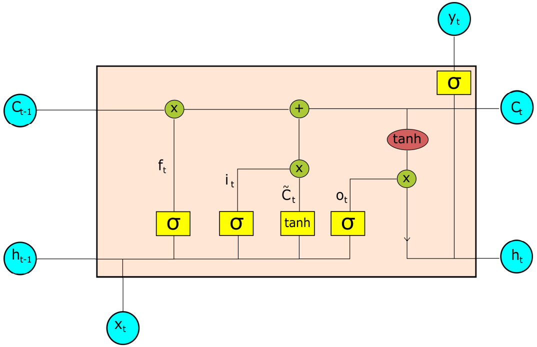

y_t
σ
C_{t-1}
x
+
C_t
tanh
f_t
x
i_t
$\tilde{C}_t$
o_t
x
σ
σ
tanh
σ
h_{t-1}
h_t
x_t

Forward



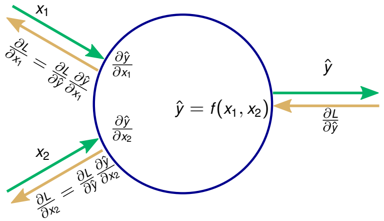
- We can reuse a **fully connected** layer again to for the **output**
- The **concatenation** is also **analogous** to the RNN
- The gates σ and the yellow tanh can be a single **fully connected** layer with an output size of $4 \cdot$ **dim(hidden state)**
- Remember that we have to pass the vectors of the input tensor **sequentially**

Backward



- Most gradients are again handled by the **embedded layers**
- Again **store and feed the values for backprop externally** to the embedded layers because of **multiple calls to forward**
- We need gradients through **summation, multiplication and copying**

Backward



x_1
$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_1}$
$\frac{\partial \hat{y}}{\partial x_1}$
$\frac{\partial \hat{y}}{\partial x_2}$
$\hat{y} = f(x_1, x_2)$
$\hat{y}$
$\frac{\partial L}{\partial \hat{y}}$
x_2
$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x_2}$

Sum

$$f(x_1, x_2) = x_1 + x_2$$
$$\frac{\partial \hat{y}}{\partial x_1} = 1$$

Gradient is **copying** $\frac{\partial L}{\partial \hat{y}}$

Multiply

$$f(x_1, x_2) = x_1 \cdot x_2$$
$$\frac{\partial \hat{y}}{\partial x_1} = x_2$$

Gradient is $\cdot$ with **switched inputs**

Copy

Backward pass of sum

So the gradient is a sum!

Thanks for listening.

Any questions?