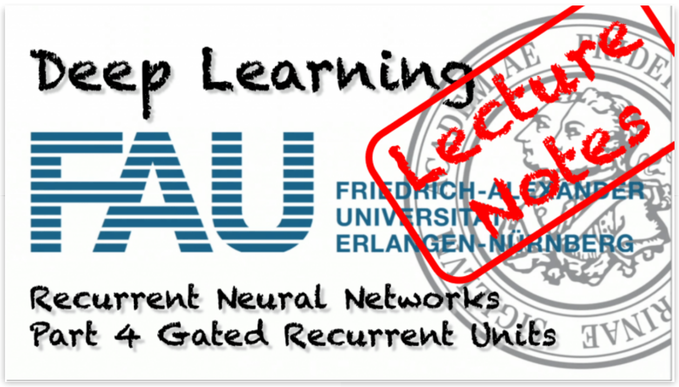
Pattern Recognition Lab

Lecture Notes in Deep Learning: Recurrent Neural Networks – Part 4

August 3, 2020

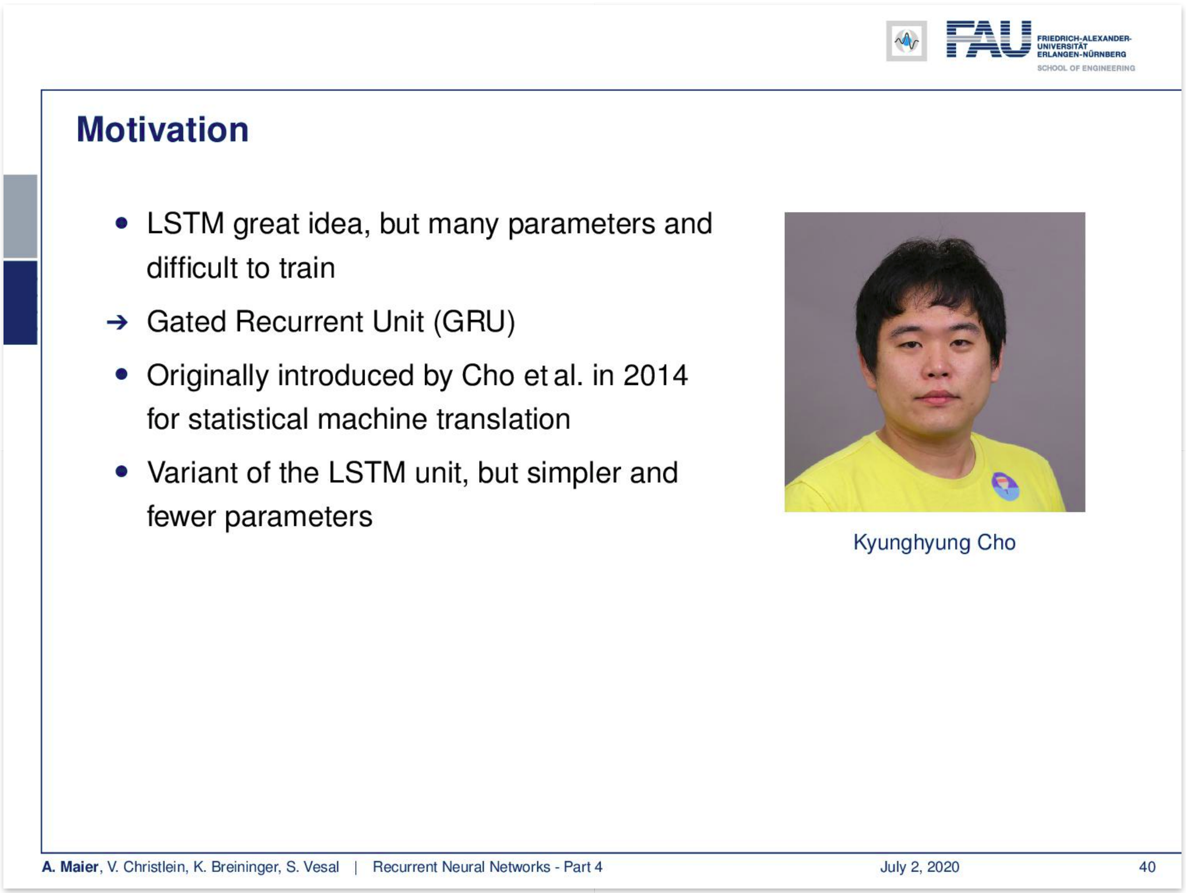

Deep Learning
FAU
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
Lecture Notes
Recurrent Neural Networks
Part 4 Gated Recurrent Units

Gated Recurrent Units

These are the lecture notes for FAU's YouTube Lecture "Deep Learning (https://www.youtube.com/watch?v=p-_Stl0t3kU&list=PLpOGQvPCDQzvgpD3S0vTy7bJe2pf_yJFj&index=1)". This is a full transcript of the lecture video & matching slides. We hope, you enjoy this as much as the videos. Of course, this transcript was created with deep learning techniques largely automatically and only minor manual modifications were performed. If you spot mistakes, please let us know!

Welcome back to deep learning! Today we want to talk a bit about gated recurrent units (GRUs), a simplification of the LSTM cell.

So again a neural network: Gated recurrent units. So the idea here is that the LSTM is, of course, great, but it has a lot of parameters and it's kind of difficult to train.



Motivation

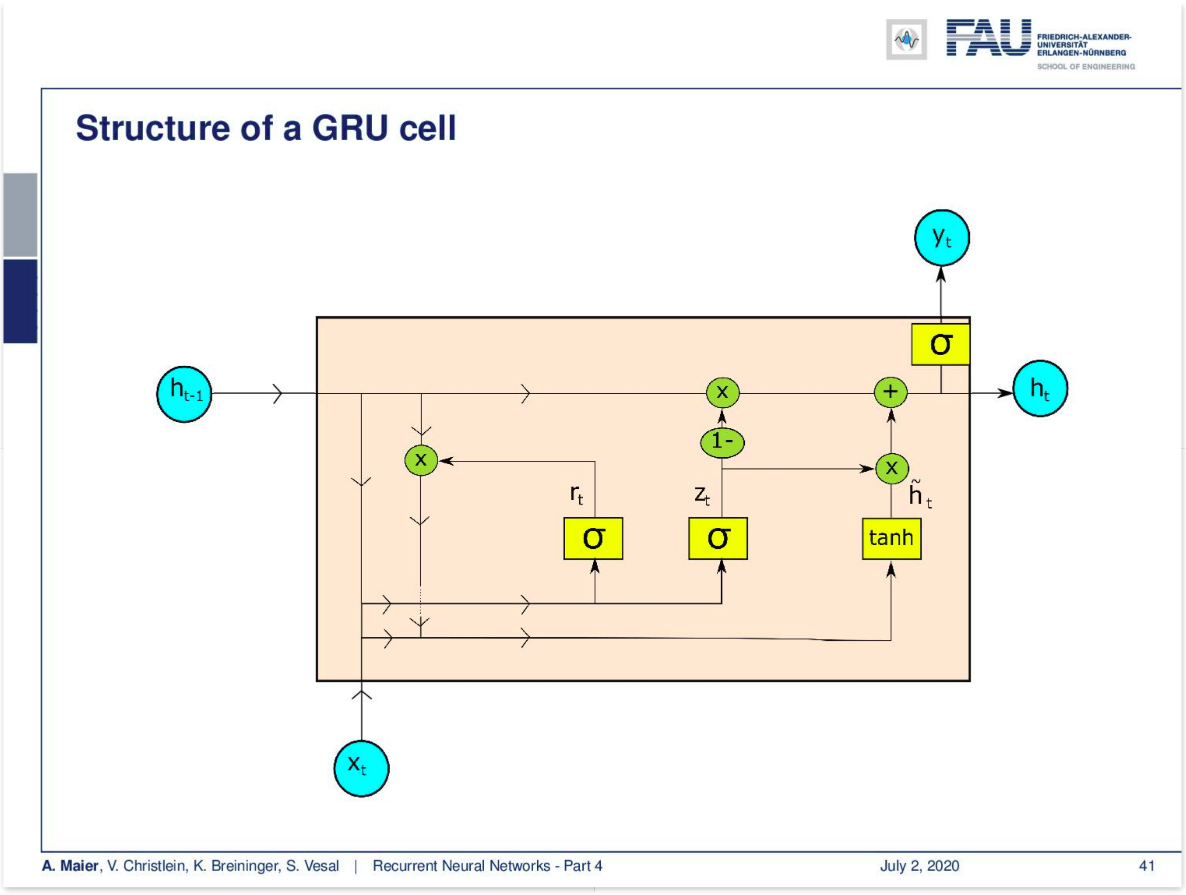
- LSTM great idea, but many parameters and difficult to train
- → Gated Recurrent Unit (GRU)
- Originally introduced by Cho et al. in 2014 for statistical machine translation
- Variant of the LSTM unit, but simpler and fewer parameters

Kyunghyung Cho

A. Maier, V. Christlein, K. Breininger, S. Vesal | Recurrent Neural Networks - Part 4 July 2, 2020 40

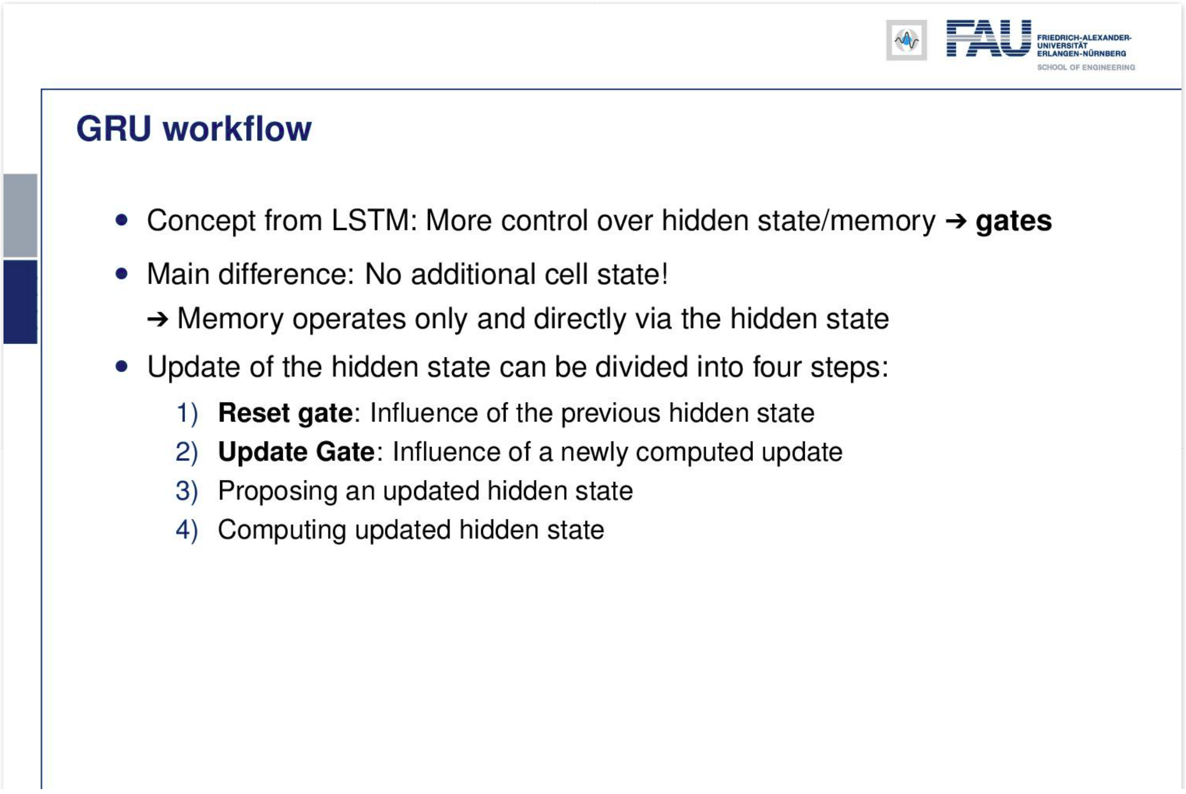
GRUs have comparable performance to LSTMs, but fewer parameters. Image under CC BY 4.0 from the Deep Learning Lecture.

So, Cho came up with the gated recurrent unit and it was introduced in 2014 for statistical machine translation. You could argue it's a kind of LSTM, but it's simpler and it has fewer parameters.



The structure of a GRU cell. Image under CC BY 4.0 from the Deep Learning Lecture.

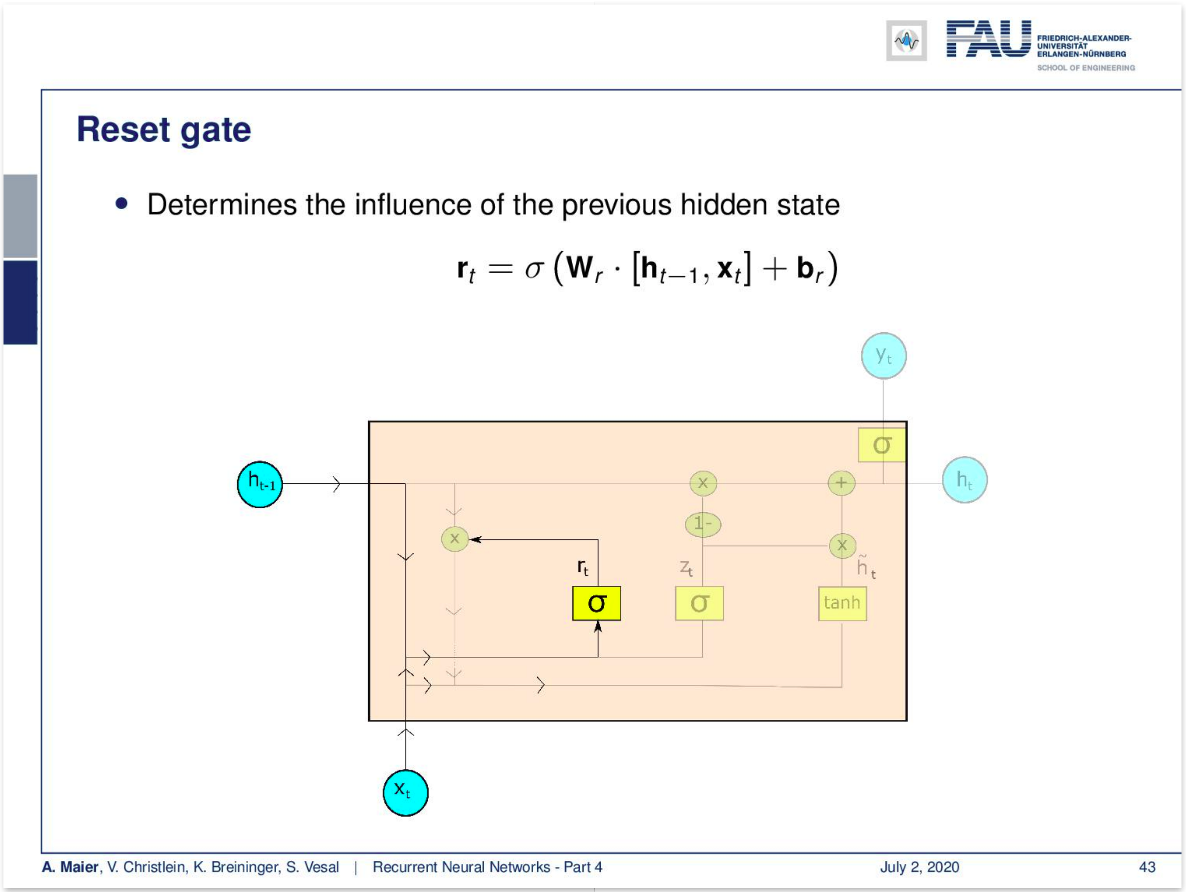
So this is the general setup. You can see we don't have two different memories like in the LSTM. We only have one hidden state. One similarity to the LSTM is the hidden state flows only along a linear process. So, you only see multiplications and additions here. Again, as in the LSTM, we produce from the hidden state the output.



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

So let's have a look into the ideas that Cho had in order to propose this cool cell. Well, it takes the concepts from the LSTM and it controls the memory by gates. The main difference is that there is no additional cell state. So, the memory only operates directly on the hidden state. The update of the state can be divided into four steps: There's a reset gate that is controlling the influence of the previous hidden state. Then there is an update gate that introduces newly computed updates. So, the next step proposes an updated hidden state which is then used to update the hidden state.



Reset gate

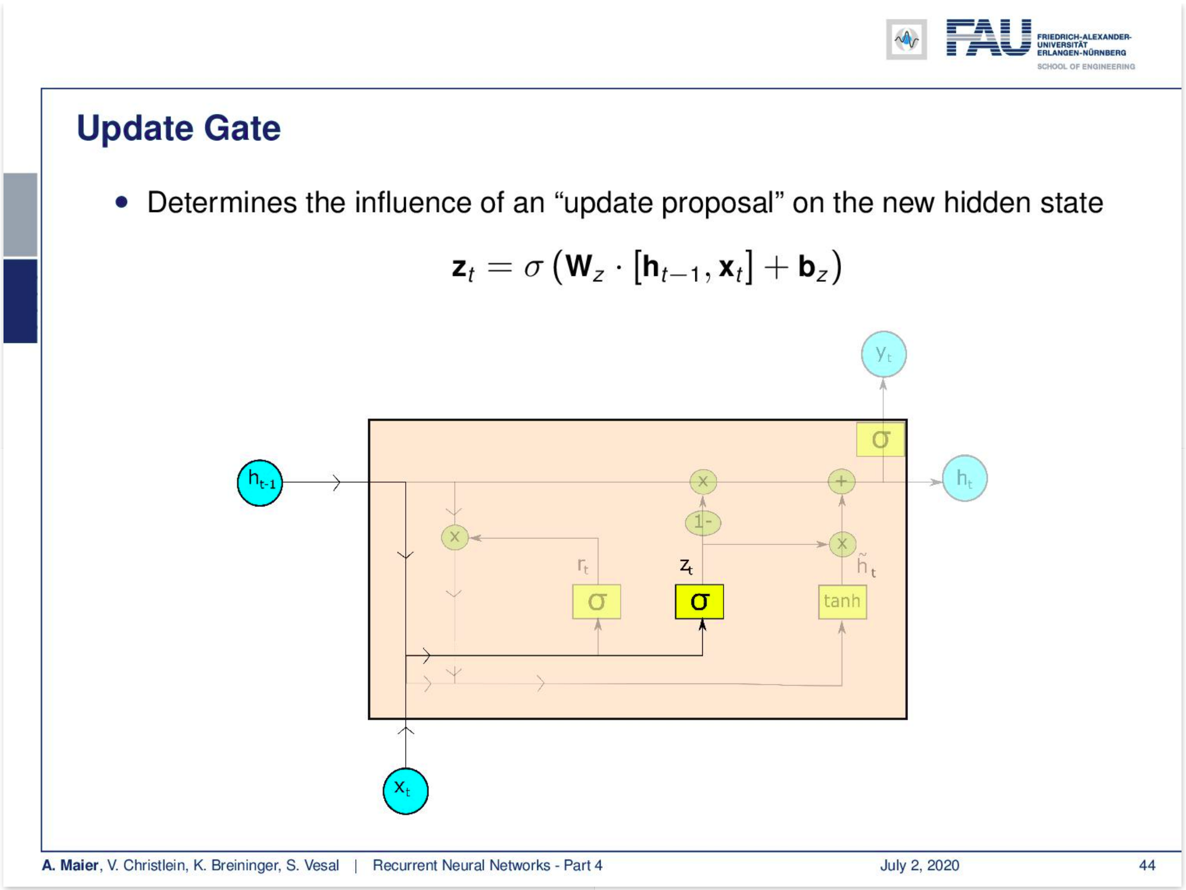
- Determines the influence of the previous hidden state

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

A. Maier, V. Christlein, K. Breininger, S. Vesal | Recurrent Neural Networks - Part 4 July 2, 2020 43

The reset gate. Image under CC BY 4.0 from the Deep Learning Lecture.

So, how does this work? Well, first we determine the influence of the previous hidden state. This is done by a sigmoid activation function. We again have a matrix-type of update. We concatenate the input and the previous hidden state multiplied with a matrix and add some bias. Then, feed it to this sigmoid activation function which produces some reset value **r** subscript t.



Update Gate

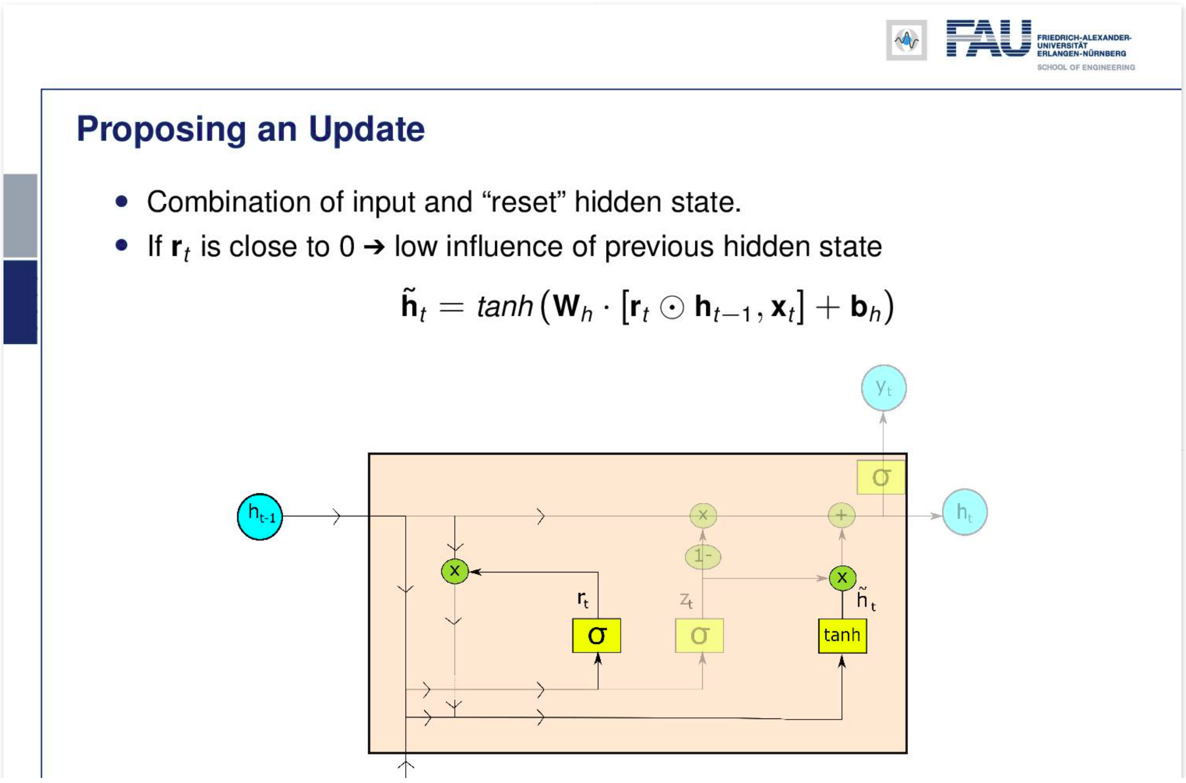
- Determines the influence of an "update proposal" on the new hidden state

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

A. Maier, V. Christlein, K. Breininger, S. Vesal | Recurrent Neural Networks - Part 4 July 2, 2020 44

The update gate first computes **z** subscript t. Image under CC BY 4.0 from the Deep Learning Lecture.

Next, we produce some **z**. This is essentially the update proposal on the new hidden state. So, this is again produced by a sigmoid function where we concatenate the last hidden state, and the input vector multiplied with a matrix **W** subscript z and add some bias.

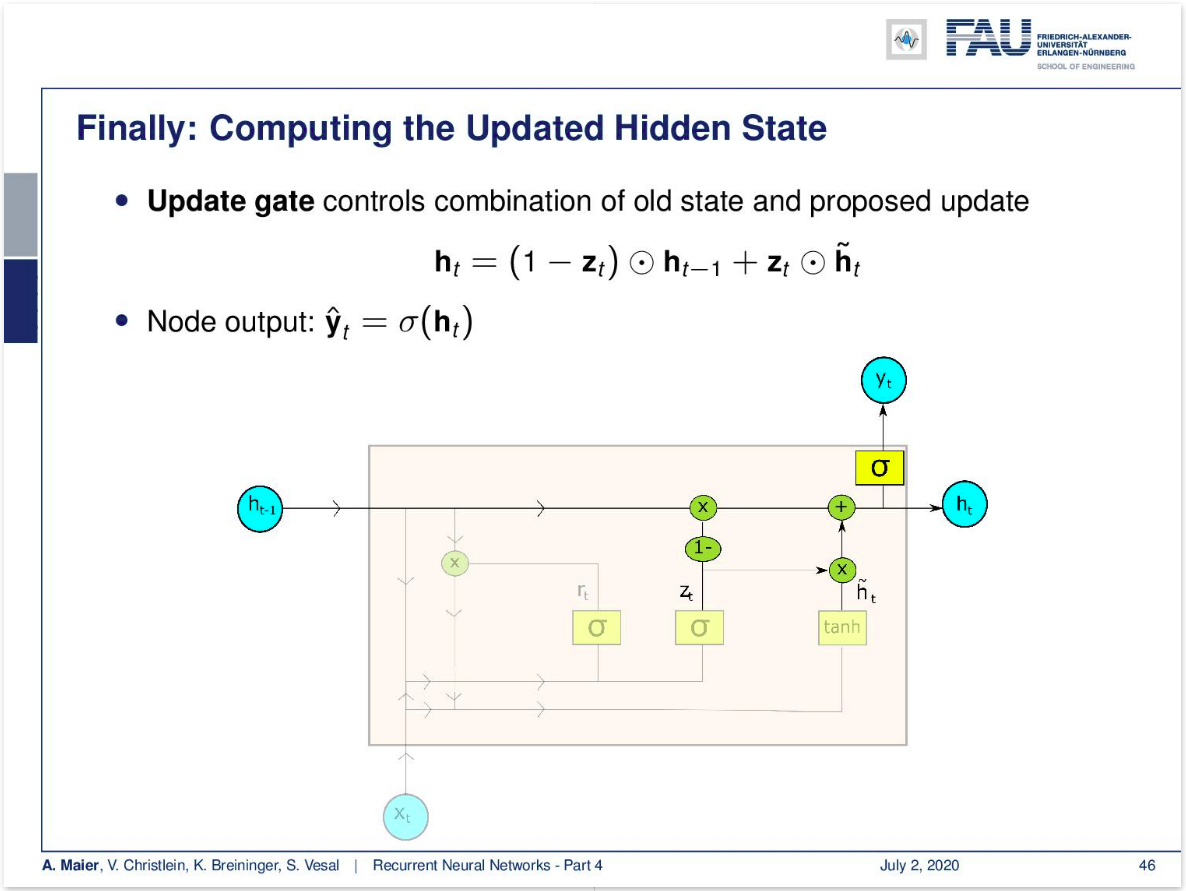


Proposing an Update

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

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

Next, we propose the update. So we combine the input and the reset state. This is done in the following manner: So, the update proposal **h** tilde is produced by a hyperbolic tangent where we take the reset gate times the last hidden state. So, we essentially remove entries that we don't want to see from the last hidden state and concatenate **x** subscript t multiplied with some matrix **W** subscript h and add some bias **b** subscript h. This is then fed to a tanh to produce the update proposal.



Finally: Computing the Updated Hidden State

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

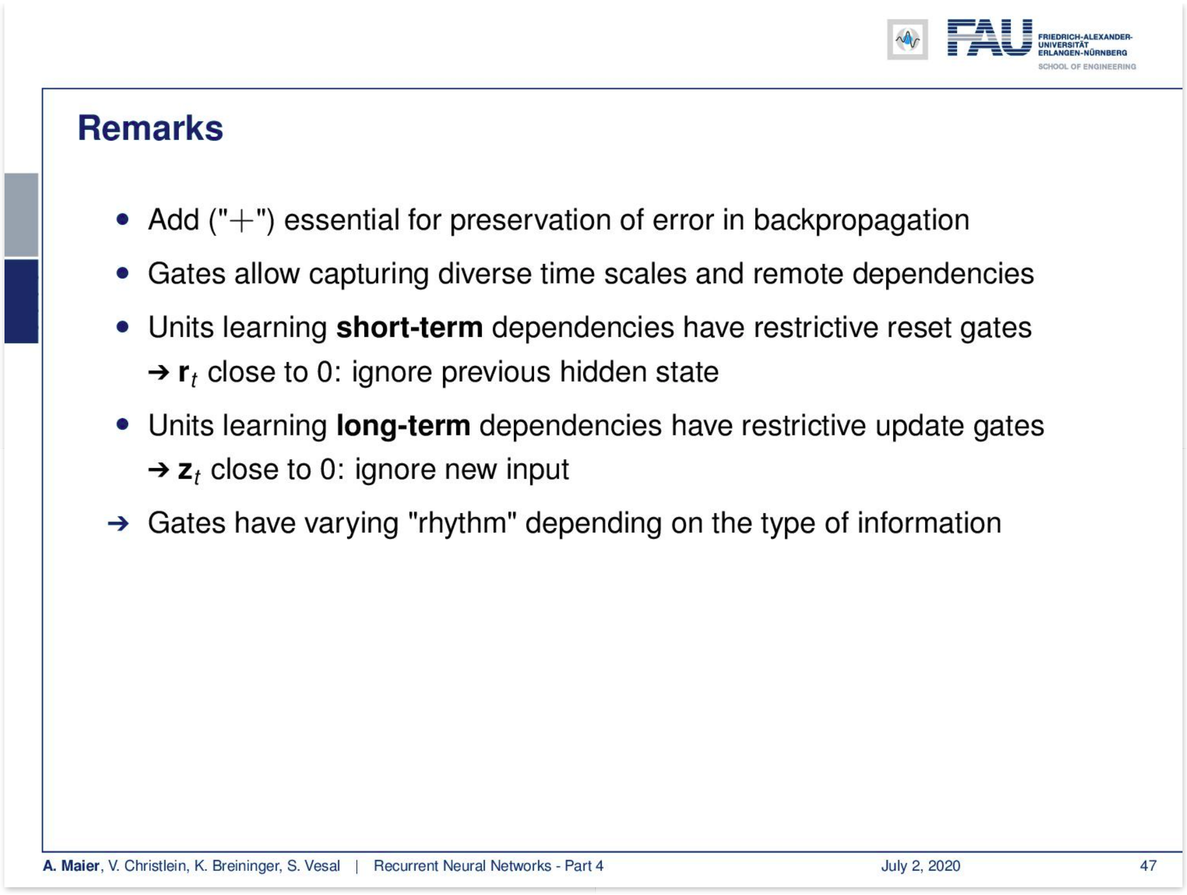
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

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

A. Maier, V. Christlein, K. Breininger, S. Vesal | Recurrent Neural Networks - Part 4 July 2, 2020 46

Finally, the new state is computed as a mixture between the old state and the update proposal. Image under CC BY 4.0 from the Deep Learning Lecture.

Now, with the update proposal, we go to the update gate. The update gate controls the combination of the old state and the proposed state. So, we compute the new state by multiplying 1 – **z** subscript t. You remember this is the intermediate variable that we computed earlier with the old state. Further, we add **z** subscript t times **h** tilde. This is the proposed update. So, essentially the sigmoid function that produced a **z** subscript t is now used to select whether to keep the old information from the old state or to update it with information from the new state. This gives the new hidden state. With the new hidden state, we produce the new output and notice again that we are omitting the transformation matrices in this step. So, we write this as sigmoid of **h** subscript t, but there's actually the transformation matrices and biases. Noth we are not noting down here. So, this thing gives the final output **y** hat subscript t.



Remarks

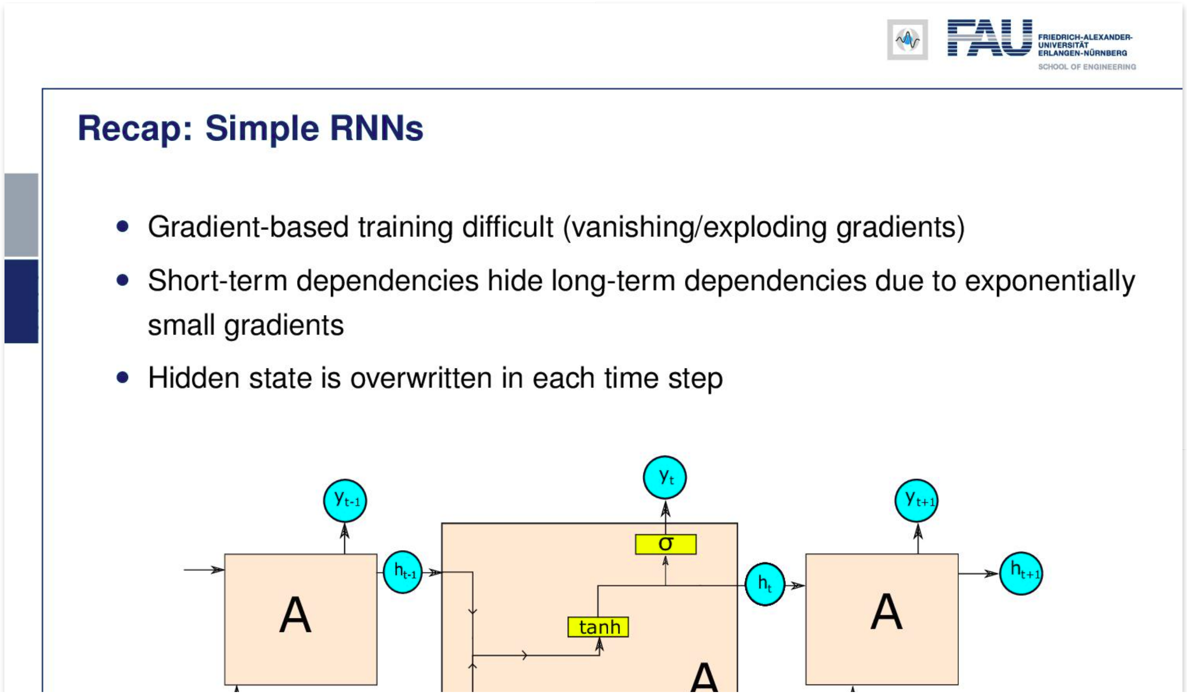
- Add ("+") essential for preservation of error in backpropagation
- Gates allow capturing diverse time scales and remote dependencies
- Units learning **short-term** dependencies have restrictive reset gates
  → $\mathbf{r}_t$ close to 0: ignore previous hidden state
- Units learning **long-term** dependencies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

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

A. Maier, V. Christlein, K. Breininger, S. Vesal | Recurrent Neural Networks - Part 4 July 2, 2020 47

Remarks concerning GRUs. Image under CC BY 4.0 from the Deep Learning Lecture.

Some remarks: The addition is essential for the preservation of the error in the backpropagation. The gates allow capturing diverse timescales and remote dependencies. The units are able to learn short-term dependencies by learning restrictive gates. So, if we have an **r** subscript t close to zero it will ignore the previous hidden state. We can also learn long-term dependencies by having restrictive update gates. So, here we have the z subscript t close to zero which means we ignore new input the gates. Then, a varying rhythm depending on the type of information emerges. Now, you'd say "Ok. now we have our RNN units, we have LSTM units, and GRUs. So, which one should we take?"



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