LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 8: Modern Recurrent Neural Networks

Mina Rezaei

Department of Statistics – LMU Munich

Winter Semester 2020

LECTURE OUTLINE

Long Short-Term Memory (LSTM)

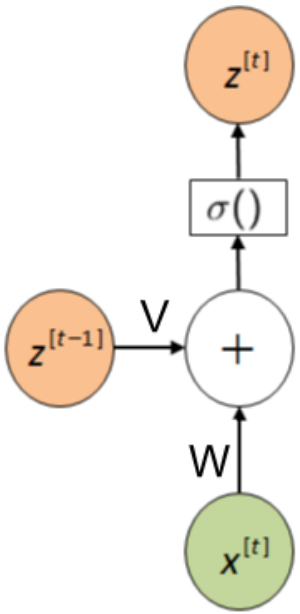
Gated Recurrent Units (GRU)

Bidirectional RNNs

Long Short-Term Memory (LSTM)

LONG SHORT-TERM MEMORY (LSTM)

The LSTM provides a way of dealing with vanishing gradients and modelling long-term dependencies.

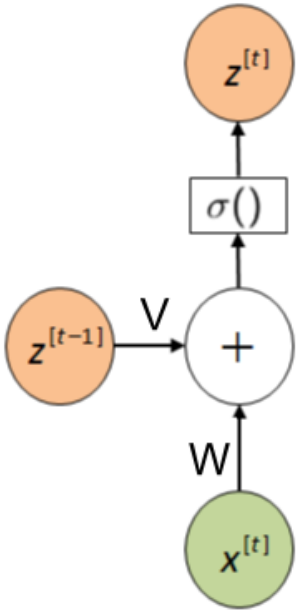


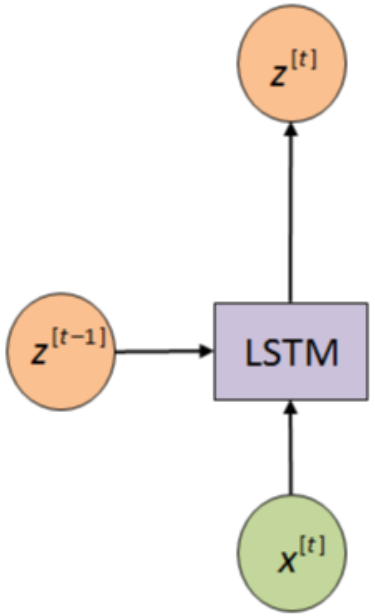
- Until now, we simply computed

$$\boldsymbol{z}^{[t]} = \sigma(\boldsymbol{b} + \boldsymbol{V}^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}^\top \mathbf{x}^{[t]})$$

LONG SHORT-TERM MEMORY (LSTM)

The LSTM provides a way of dealing with vanishing gradients and modelling long-term dependencies.



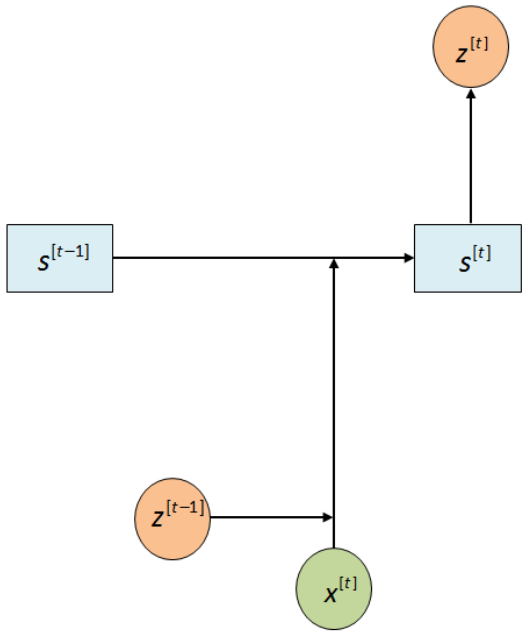


- Until now, we simply computed

$$\boldsymbol{z}^{[t]} = \sigma(\boldsymbol{b} + \boldsymbol{V}^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}^\top \mathbf{x}^{[t]})$$

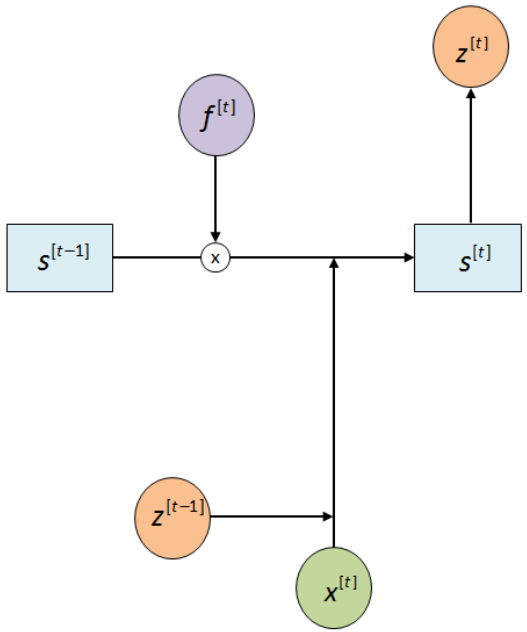
- Now we introduce the LSTM cell (which is a small network on its own).

LONG SHORT-TERM MEMORY (LSTM)



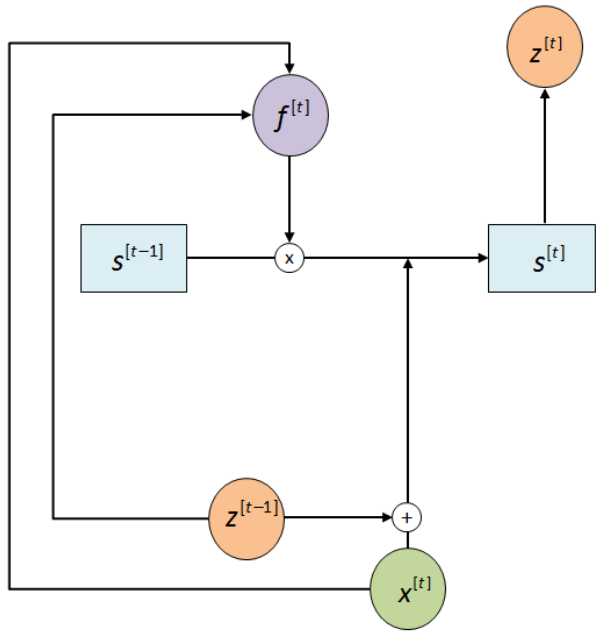
- The key to LSTMs is the **cell state** $\boldsymbol{s}^{[t]}$.
- $\boldsymbol{s}^{[t]}$ can be manipulated by different **gates** to forget old information, add new information, and read information out of it.
- Each gate is a vector of the same size as the $\boldsymbol{s}^{[t]}$ with elements between 0 ("let nothing pass") and 1 ("let everything pass").

LONG SHORT-TERM MEMORY (LSTM)



- **Forget gate $\boldsymbol{f}^{[t]}$**: indicates which information of the old cell state we should forget.
- Intuition: Think of a model trying to predict the next word based on all the previous ones. The cell state might include the gender of the present subject, so that the correct pronouns can be used. When we now see a new subject, we want to forget the gender of the old one.

LONG SHORT-TERM MEMORY (LSTM)

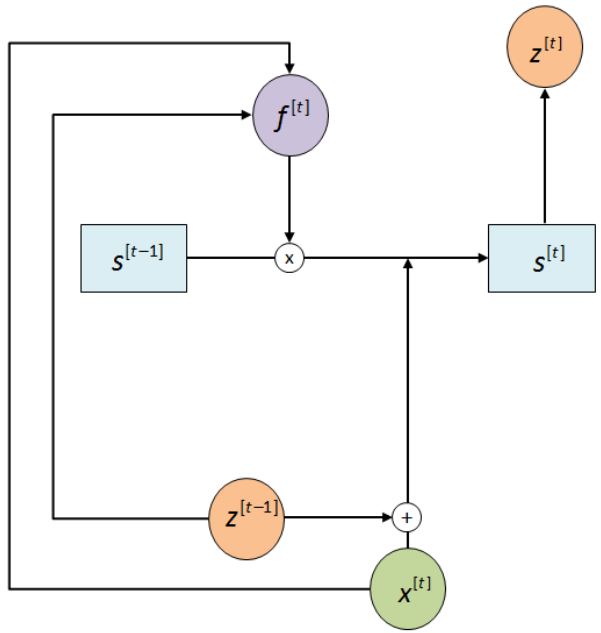


- We obtain the forget gate by computing

$$\boldsymbol{f}^{[t]} = \sigma(\boldsymbol{b}_f + \boldsymbol{V}_f^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}_f^\top \mathbf{x}^{[t]})$$

- $\sigma()$ is a sigmoid, squashing the values to $[0, 1]$, and $\boldsymbol{V}_f$, $\boldsymbol{W}_f$ are forget gate specific weights.

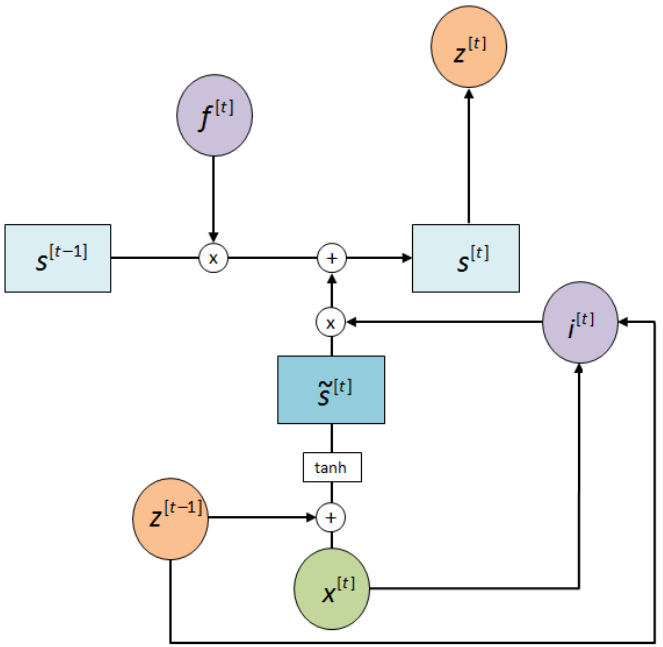
LONG SHORT-TERM MEMORY (LSTM)



- To compute the cell state $\boldsymbol{s}^{[t]}$, the first step is to multiply (element-wise) the previous cell state $\boldsymbol{s}^{[t-1]}$ by the forget gate $\boldsymbol{f}^{[t]}$.

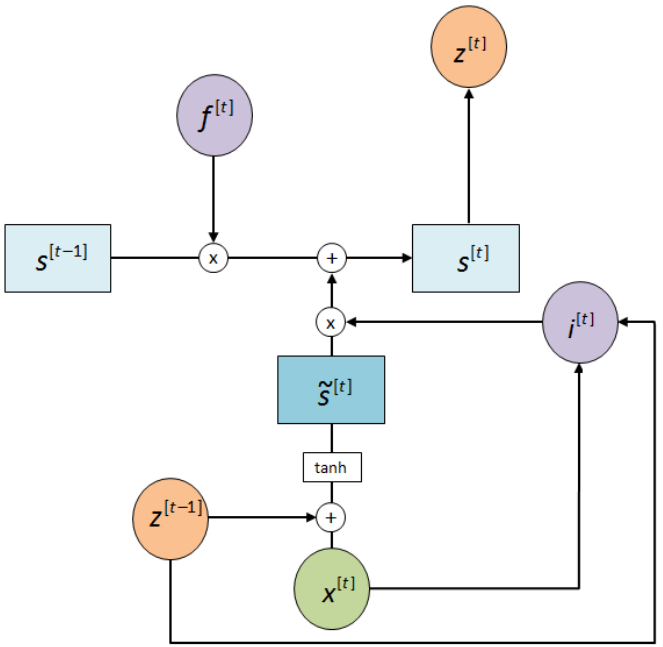
$$\boldsymbol{f}^{[t]} \odot \boldsymbol{s}^{[t-1]}, \text{ with } \boldsymbol{f}^{[t]} \in [0, 1]$$

LONG SHORT-TERM MEMORY (LSTM)



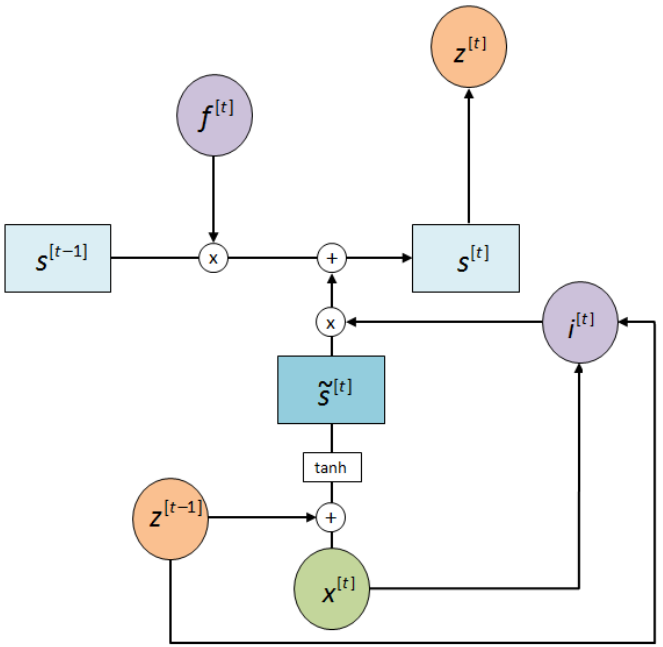
- **Input gate** $i^{[t]}$: indicates which new information should be added to $\boldsymbol{s}^{[t]}$.
- Intuition: In our example, this is where we add the new information about the gender of the new subject.

LONG SHORT-TERM MEMORY (LSTM)



- The new information is given by $\tilde{\boldsymbol{s}}^{[t]} = \tanh(\boldsymbol{b} + \boldsymbol{V}^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}^\top \mathbf{x}^{[t]}) \in [-1, 1]$.
- The input gate is given by $\boldsymbol{i}^{[t]} = \sigma(\boldsymbol{b}_i + \boldsymbol{V}_i^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}_i^\top \mathbf{x}^{[t]}) \in [0, 1]$.
- $\boldsymbol{W}$ and $\boldsymbol{V}$ are weights of the new information, $\boldsymbol{W}_i$ and $\boldsymbol{V}_i$ the weights of the input gate.

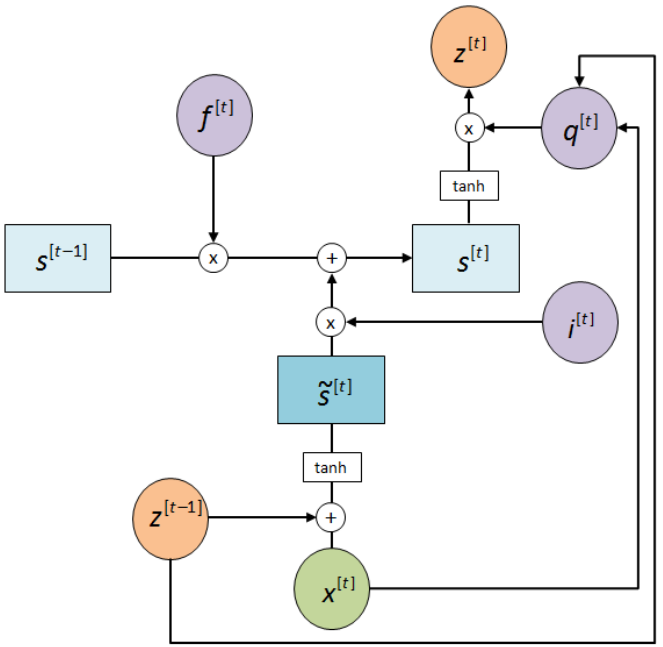
LONG SHORT-TERM MEMORY (LSTM)



- Now we can finally compute the cell state $\boldsymbol{s}^{[t]}$:

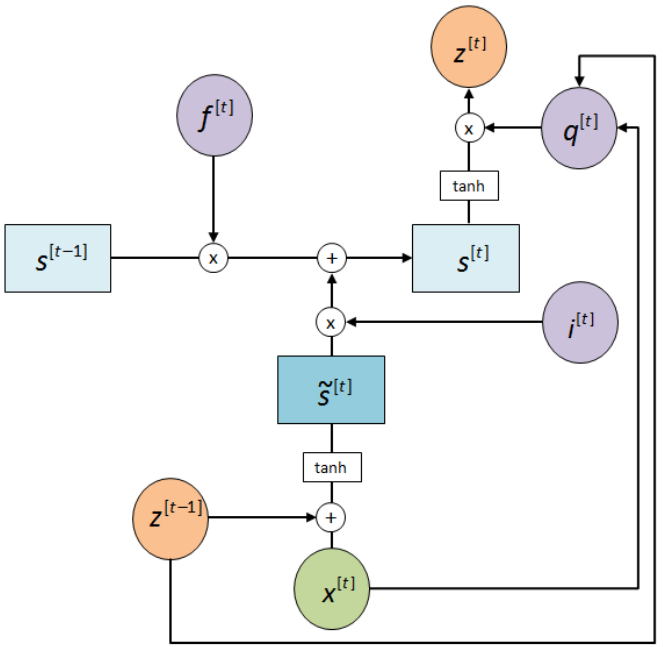
$$\boldsymbol{s}^{[t]} = \boldsymbol{f}^{[t]} \odot \boldsymbol{s}^{[t-1]} + \boldsymbol{i}^{[t]} \odot \tilde{\boldsymbol{s}}^{[t]}$$

LONG SHORT-TERM MEMORY (LSTM)



- **Output gate $\boldsymbol{q}^{[t]}$**: Indicates which information form the cell state is filtered.
- It is given by $\boldsymbol{q}^{[t]} = \sigma(\boldsymbol{b}_q + \boldsymbol{V}_q^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}_q^\top \mathbf{y}^{[t]})$, with specific weights $\boldsymbol{W}_q$, $\boldsymbol{V}_q$.

LONG SHORT-TERM MEMORY (LSTM)



- Finally, the new state $\boldsymbol{z}^{[t]}$ of the LSTM is a function of the cell state, multiplied by the output gate:

$$\boldsymbol{z}^{[t]} = \boldsymbol{q}^{[t]} \odot \tanh(\boldsymbol{s}^{[t]})$$

Gated Recurrent Units (GRU)

GATED RECURRENT UNITS (GRU)

- The key distinction between regular RNNs and GRUs is that the latter support gating of the hidden state.
- Here, we have dedicated mechanisms for when a hidden state should be updated and also when it should be reset.
- These mechanisms are learned and they address the listed concerns:
  - avoid the vanishing/exploding gradient problem which comes with a standard recurrent neural network.
  - GRU's are able to solve the vanishing gradient problem by using an update gate and a reset gate.
  - the update gate controls information that flows into memory, and the reset gate controls the information that flows out of memory.

RESET GATES AND UPDATE GATES

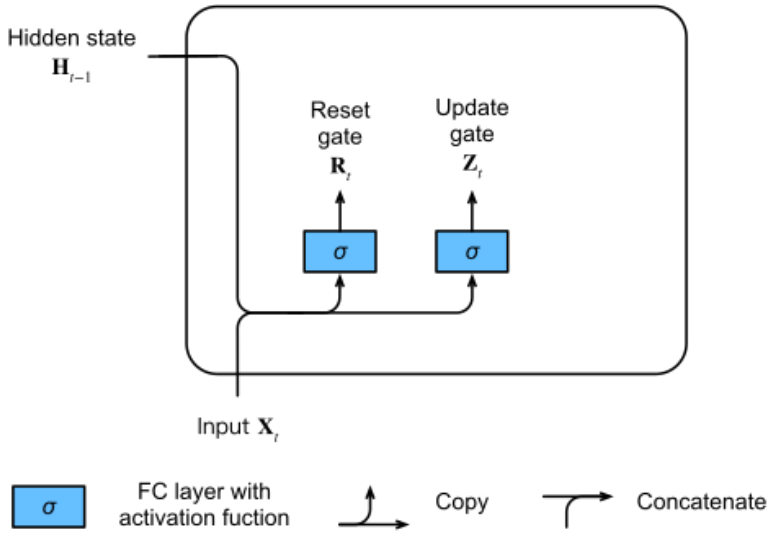


Figure: Reset and update gate in a GRU.

- as shown by Figure1, given the current time step input X_t and the hidden state of the previous time step H_{t-1}. The output is given by a fully connected layer with a sigmoid as its activation function.

RESET GATES AND UPDATE GATES

- For a given time step t , the minibatch input is $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ (number of examples:n, number of inputs:d) and the hidden state of the last time step is $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times h}$(number of hidden states:h). Then, the reset gate $\mathbf{R}_t \in \mathbb{R}^{n \times h}$ and update gate $\mathbf{Z}_t \in \mathbb{R}^{n \times h}$ are computed as follows:
- $\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r)$
- $\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z)$
- Here,$\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d \times h}$ and $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h \times h}$ are weight parameters and $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1 \times h}$ are biases. We use a sigmoid function to transform input values to the interval $(0, 1)$.

RESET GATES IN ACTION

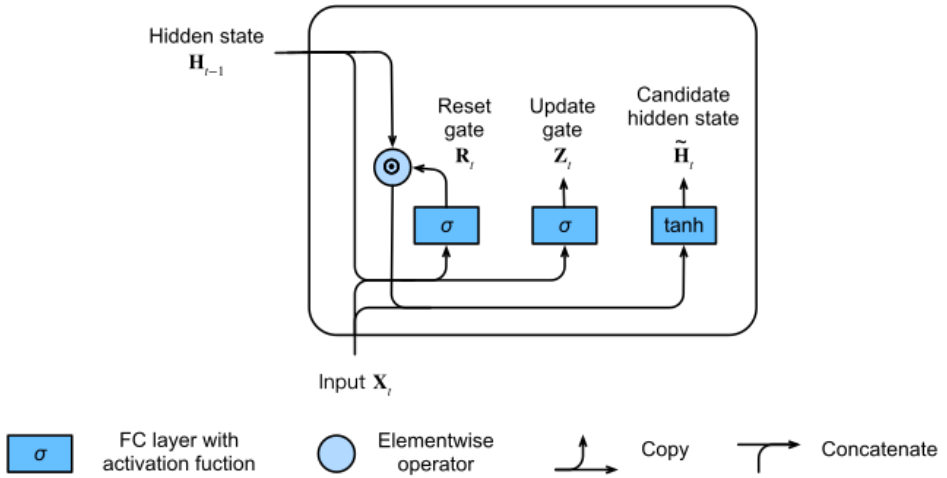


Figure: Candidate hidden state computation in a GRU. The multiplication is carried out elementwise.

- We begin by integrating the reset gate with a regular latent state updating mechanism. In a conventional RNN, we would have an hidden state update of the form:
- $\mathbf{H}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h)$.
- $\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h)$.

UPDATE GATES IN ACTION

- Next we need to incorporate the effect of the update gate Z_t.
- The gating variable Z_t can be used for this purpose, simply by taking elementwise convex combinations between both candidates.
- This leads to the final update equation for the GRU.
- $\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$.

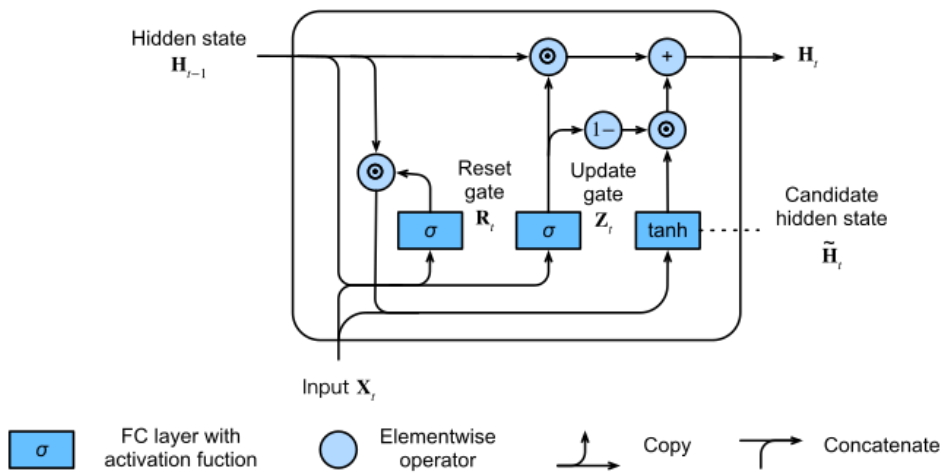


Figure: Hidden state computation in a GRU. As before, the multiplication is carried out elementwise.

UPDATE GATES IN ACTION

These designs can help us to eleminate the vanishing gradient problem in RNNs and capture better dependencies for time series with large time step distances. In summary, GRUs have the following two distinguishing features:

- Reset gates help capture short-term dependencies in time series.
- Update gates help capture long-term dependencies in time series.

UPDATE GATES IN ACTION

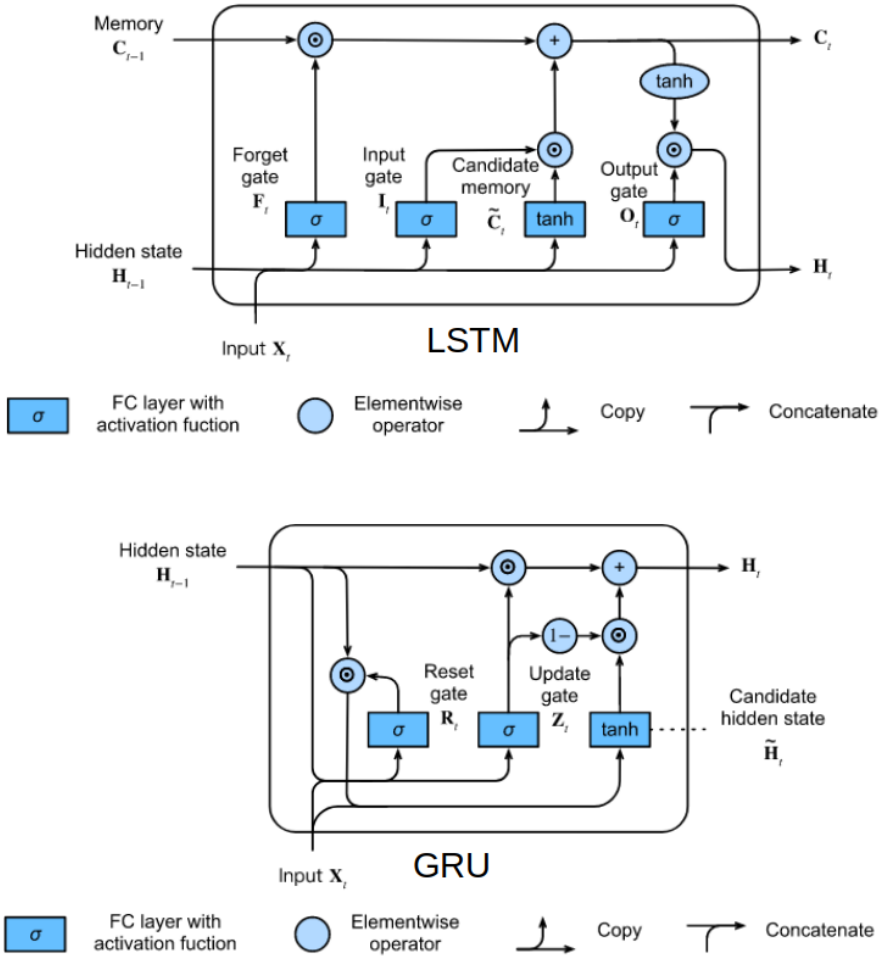


Figure: LSTM vs GRU

Bidirectional RNNs

BIDIRECTIONAL RNNS

- Another generalization of the simple RNN are bidirectional RNNs.
- These allow us to process sequential data depending on both past and future inputs, e.g. an application predicting missing words, which probably depend on both preceding and following words.
- One RNN processes inputs in the forward direction from $x^{[1]}$ to $x^{[T]}$ computing a sequence of hidden states $(z^{[1]}, \ldots, z^{(T)})$, another RNN in the backward direction from $x^{[T]}$ to $x^{[1]}$ computing hidden states $(g^{[T]}, \ldots, g^{[1]})$
- Predictions are then based on both hidden states, which could be **concatenated**.
- With connections going back in time, the whole input sequence must be known in advance to train and infer from the model.
- Bidirectional RNNs are often used for the encoding of a sequence in machine translation.

BIDIRECTIONAL RNNS

Computational graph of an bidirectional RNN:

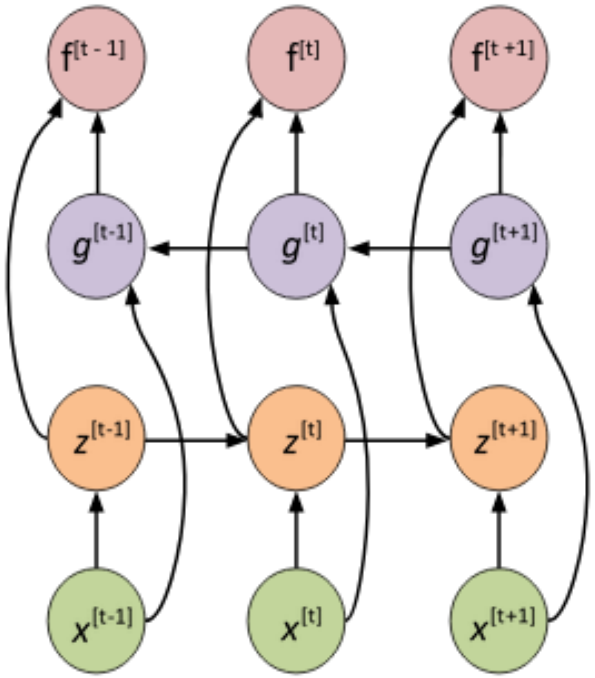


Figure: A bidirectional RNN consists of a forward RNN processing inputs from left to right and a backward RNN processing inputs backwards in time.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Michael Nguyen (2018)

Illustrated Guide to LSTM's and GRU's: A step by step explanation newblock
https://towardsdatascience.com/
illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21