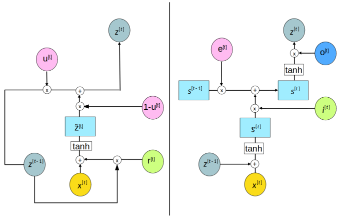
Deep Learning

Modern Recurrent Neural Networks



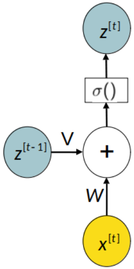
Learning goals

- LSTM cell
- GRU cell
- 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.



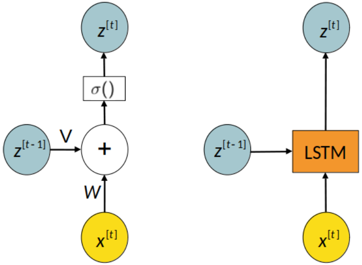
A simple RNN mechanism;

- Until now, we simply computed

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

LONG SHORT-TERM MEMORY (LSTM)

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



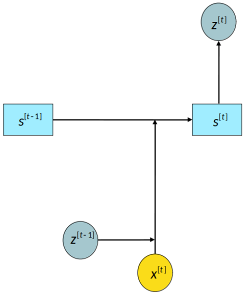
Left: A simple RNN mechanism; Right: An LSTM cell

- Until now, we simply computed

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

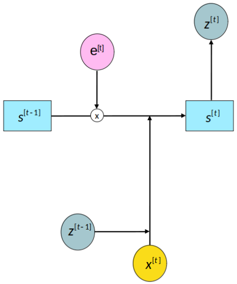
- Now we introduce the LSTM cell, a small network on its own.

LONG SHORT-TERM MEMORY (LSTM)



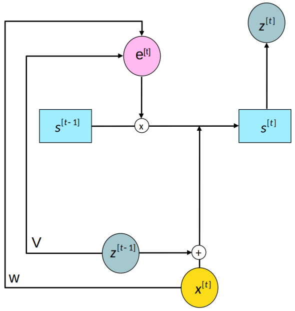
- The key to LSTMs is the **cell state** $\mathbf{s}^{[t]}$.
- $\mathbf{s}^{[t]}$ can be manipulated by different **gates** to forget old information, add new information, and read information out of it.
- Each gate is a vector of the same size as $\mathbf{s}^{[t]}$ with elements between 0 ("let nothing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

LONG SHORT-TERM MEMORY (LSTM)



- **Forget gate $\mathbf{e}^{[t]}$**: 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)

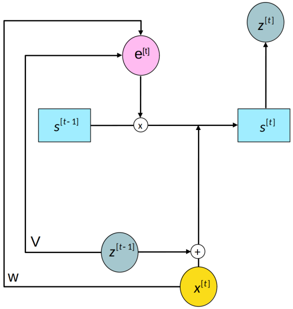


- We obtain the forget gate by computing

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

- $\sigma()$ is a sigmoid and $\mathbf{V}_e, \mathbf{W}_e$ are forget gate specific weights.

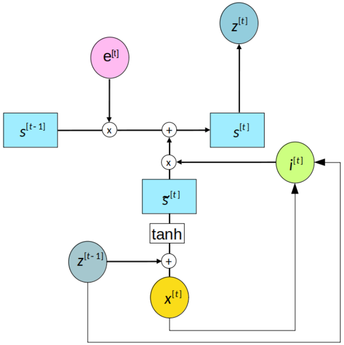
LONG SHORT-TERM MEMORY (LSTM)



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

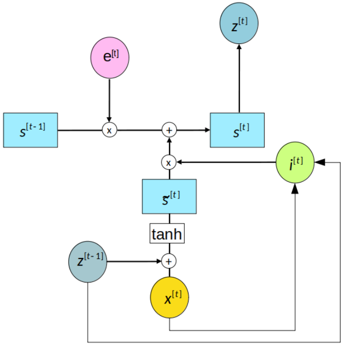
$$\mathbf{e}^{[t]} \odot \mathbf{s}^{[t-1]}, \text{ with } \mathbf{e}^{[t]} \in [0, 1]$$

LONG SHORT-TERM MEMORY (LSTM)



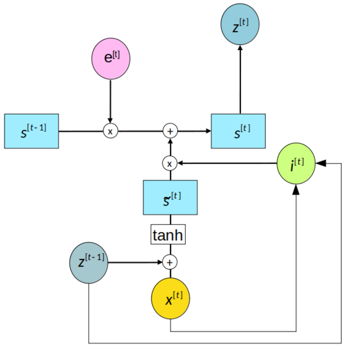
- **Input gate $\mathbf{i}^{[t]}$**: indicates which new information should be added to $\mathbf{s}^{[t]}$.
- Intuition: In our example, this is where we add the new information about the gender of the new subject.

LONG SHORT-TERM MEMORY (LSTM)



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

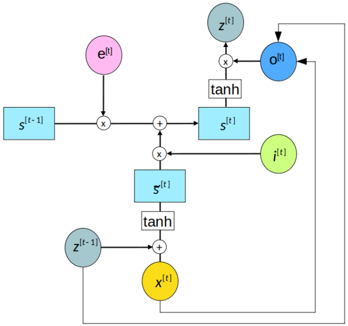
LONG SHORT-TERM MEMORY (LSTM)



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

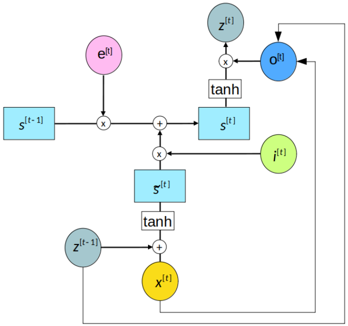
$$\mathbf{s}^{[t]} = \mathbf{e}^{[t]} \odot \mathbf{s}^{[t-1]} + \mathbf{i}^{[t]} \odot \tilde{\mathbf{s}}^{[t]}$$

LONG SHORT-TERM MEMORY (LSTM)



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

LONG SHORT-TERM MEMORY (LSTM)



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

$$\mathbf{z}^{[t]} = \mathbf{o}^{[t]} \odot \tanh(\mathbf{s}^{[t]})$$

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 to:
    - avoid the vanishing/exploding gradient problem which comes with a standard recurrent neural network.
    - solve the vanishing gradient problem by using an update gate and a reset gate.
    - control the information that flows into (update gate) and out of (reset gate) memory.

GATED RECURRENT UNITS (GRU)

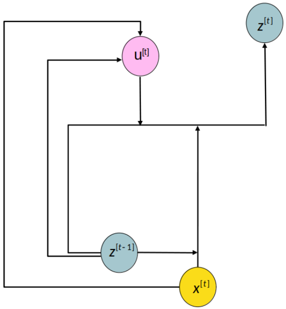


Figure: Update gate in a GRU.

- For a given time step t, the hidden state of the last time step is $\mathbf{z}^{[t-1]}$. The update gate $\mathbf{u}^{[t]}$ is computed as follows:
- $\mathbf{u}^{[t]} = \sigma(\mathbf{W}_u^\top \mathbf{x}^{[t]} + \mathbf{V}_u^\top \mathbf{z}^{[t-1]} + \mathbf{b}_u)$
- We use a sigmoid to transform input values to $(0, 1)$.

GATED RECURRENT UNITS (GRU)

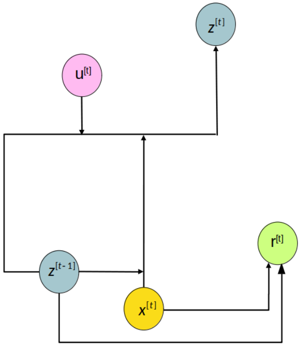


Figure: Reset gate in a GRU.

- Similarly, the reset gate $\mathbf{r}^{[t]}$ is computed as follows:
- $\mathbf{r}^{[t]} = \sigma(\mathbf{W}_r^\top \mathbf{x}^{[t]} + \mathbf{V}_r^\top \mathbf{z}^{[t-1]} + \mathbf{b}_r)$

GATED RECURRENT UNITS (GRU)

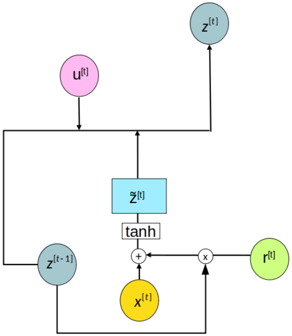


Figure: Hidden state computation in GRU. Multiplication is carried out elementwise.

- $\tilde{\mathbf{z}}^{[t]} = \tanh(\mathbf{W}_z^\top \mathbf{x}^{[t]} + \mathbf{V}_z^\top \left(\mathbf{r}^{[t]} \odot \mathbf{z}^{[t-1]}\right) + \mathbf{b}_z)$.
- In a conventional RNN, we would have an hidden state update of the form: $\mathbf{z}^{[t]} = \tanh(\mathbf{W}_z^\top \mathbf{x}^{[t]} + \mathbf{V}_z^\top \mathbf{z}^{[t-1]} + \mathbf{b}_z)$.

GATED RECURRENT UNITS (GRU)

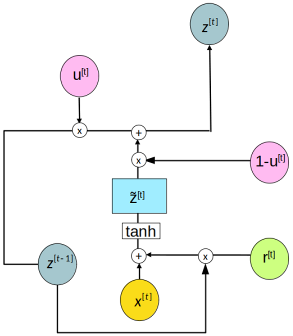


Figure: Update gate in a GRU. The multiplication is carried out elementwise.

- The update gate $\mathbf{u}^{[t]}$ determines how much the old state $\mathbf{z}^{[t-1]}$ and the new candidate state $\tilde{\mathbf{z}}^{[t]}$ is used.
- $\mathbf{z}^{[t]} = \mathbf{u}^{[t]} \odot \mathbf{z}^{[t-1]} + (1 - \mathbf{u}^{[t]}) \odot \tilde{\mathbf{z}}^{[t]}$.

GATED RECURRENT UNITS (GRU)

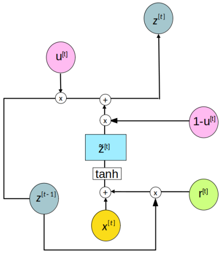


Figure: GRU

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.

GRU VS LSTM

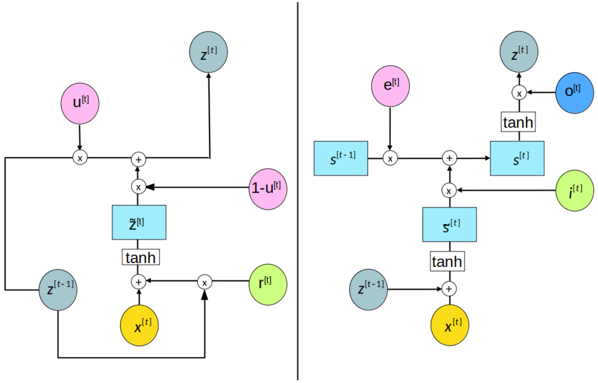


Figure: GRU vs LSTM

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 $\mathbf{x}^{[1]}$ to $\mathbf{x}^{[T]}$ computing a sequence of hidden states $(\mathbf{z}^{[1]}, \dots, \mathbf{z}^{(T)})$, another RNN in the backward direction from $\mathbf{x}^{[T]}$ to $\mathbf{x}^{[1]}$ computing hidden states $(\mathbf{g}^{[T]}, \dots, \mathbf{g}^{[1]})$
- 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 a bidirectional RNN:

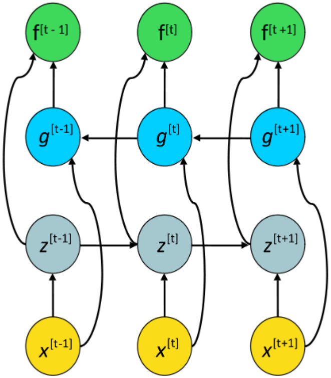


Figure: A bidirectional RNN consists of a forward RNN processing inputs from left to right and a backward RNN processing inputs backwards in time.