Applied Deep Learning for NLP

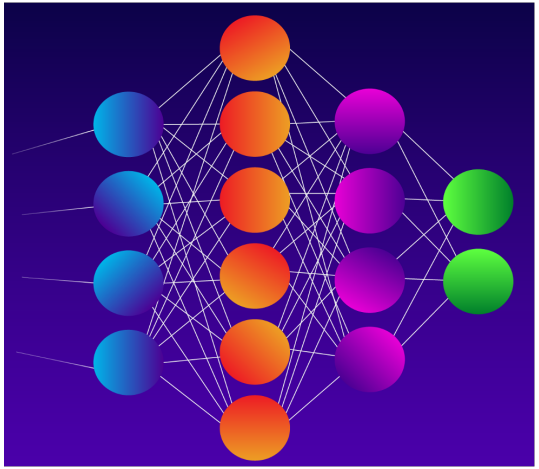
Week 4 - RNNs and LSTMs

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

Munich, 12. November 2020

political
data
science



Task: Text Classification

We have seen examples of text classifications in the coding sessions. It is the most widely used NLP task in the industry.

Binary classification (sigmoid) and **Multiclass** classification (softmax)

Examples: spam detection, misinformation detection, authorship attribution, sentiment analysis, email categorization (personal, social,...), news categorization (sports, politics, entertainment,...)

Loss: Cross-entropy loss (most of the time, sometimes others like Hinge loss)

$$L_{binary} = -(y * log(p) + (1 - y) * log(1 - p))$$

$$L_{multiclass} = -\sum_{c=1}^{M} y_c log(p_c)$$

Key to understand cross-entropy:

- log(1) =0, log(0.5) = -0.3, log(0.2) =-0.7.
- y is a one-hot vector, with only 1 for the correct class and zero for the rest

Task: Text Classification

Positives: Class we care about detecting, for spam, spam is the positive class!
Negatives: The other class, normally the less interesting
True positives, true negatives: Model made the right decision
False positives: Model predict positive, but it was wrong.
False negatives: Model predict negative, but it was wrong.

METRICS:

- **Accuracy** Fraction of times the model makes the correct predictions as compared to the total number of predictions N.

$$Accuracy = \frac{correct}{N}$$

- **Precision**. Shows how precise is the model. Given all the predicted positive cases, how many are really positive.

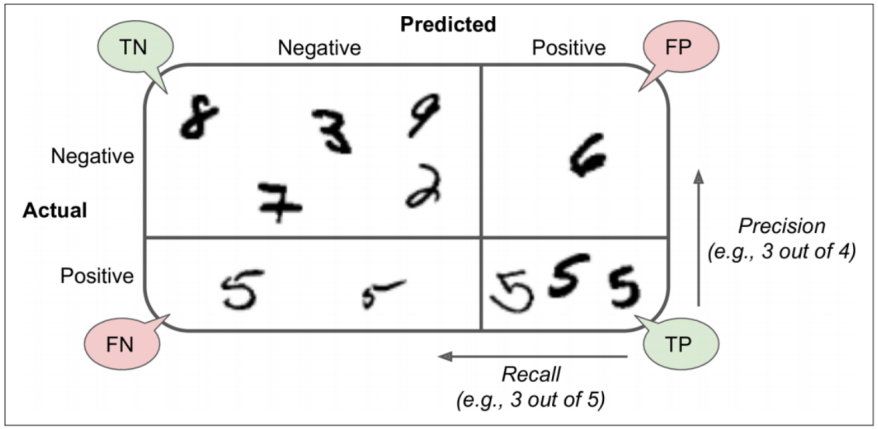
$$Precision = \frac{TP}{TP + FP}$$

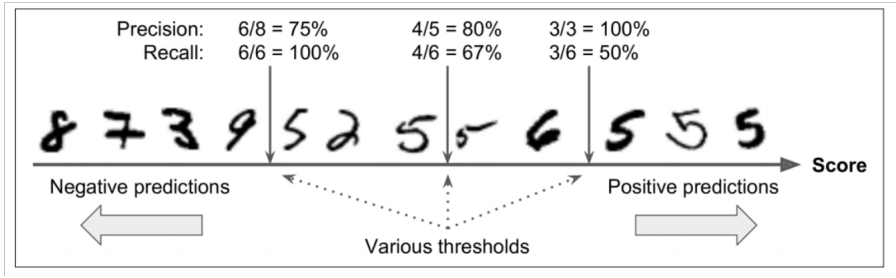
- **Recall**. Shows how many of the positive cases your model can detect from all real positive cases.

$$Recall = \frac{TP}{TP + FN}$$

Task: Text Classification

Confusion Matrix:





Task: Language Modeling

Predict what word comes next given a text corpus.

Given a sequence of words $x^{(1)}, x^{(2)}, ..., x^{(t)}$ compute the probability of the next word $x^{(t+1)}$

$$P(x^{(t+1)}|x^{(1)}, ..., x^{(t)})$$

Can be also interpreted as assigning a probability to a piece of text

$$P(x^{(1)}, ..., x^{(t)}) = P(x^{(1)}) * P(x^{(2)}|x^{(1)}) * ... * P(x^{(t)}|x^{(1)}, ..., x^{(t-1)})$$

Task: Language Modeling

Before neural networks counting n-grams was the main technique. The probability of word w:

$$P(\text{w}|\text{"}Harry \quad opened \quad the\text{"}) = \frac{count(\text{"}Harry \quad opened \quad the \quad \text{w"})}{count(\text{"}Harry \quad opened \quad the\text{"})}$$

Disadvantage: Need to store all n-grams. What about never seen n-grams?

How to evaluate if a good language model is good?

PERPLEXITY Inverse probability of the corpus of size T

$$\prod_{t=1}^{T}\left(\frac{1}{P(x^{(t+1)}|x^{(1)},...,x^{(t)})}\right)^{\frac{1}{T}}$$

Lower perplexity is better!

Task: Text Generation

Language models are the basic blocs for other NLP tasks.

In text generation, we use the probabilities calculated from the language model to generate new text.

Next word prediction consists of taking a **sample** from the probability distribution (Like when you are texting on the phone, the word suggestions are the ones with the highest probabilities)

Language model with FNN

A **fixed-window** language model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

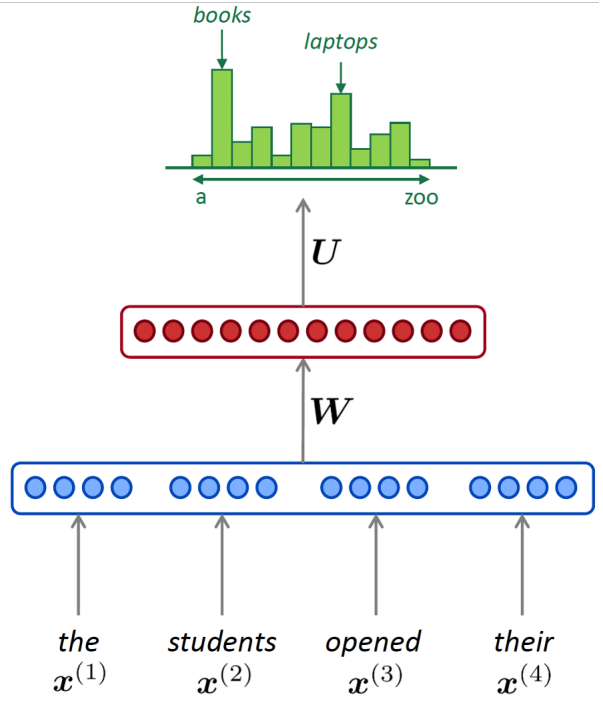
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Slides in this lecture from the amazing Stanford course: https://web.stanford.edu/class/cs224n/

Recurrent Neural Networks (RNNs)

Apply the same weights matrices repeatedly

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

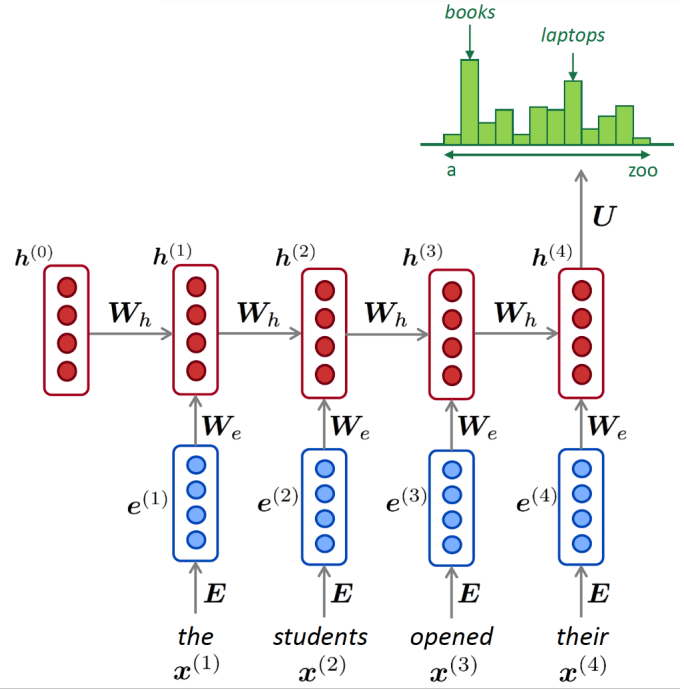
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

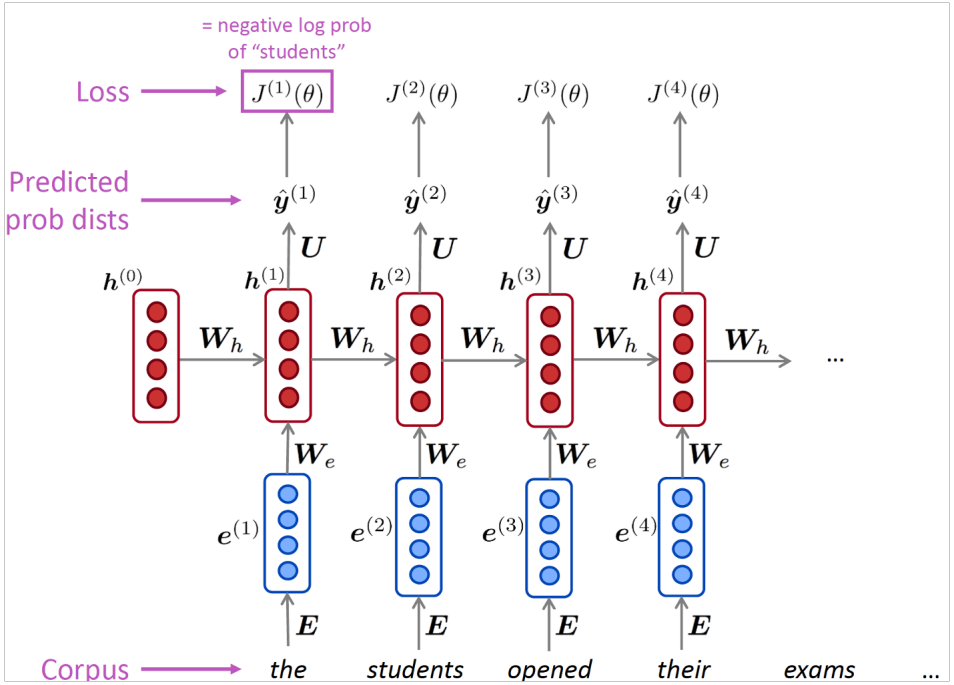
words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Training an RNN

$$Loss = L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{w=1}^{W} -y_w^{(t)} log(\hat{y}_w^{(t)}) = J(\theta)$$



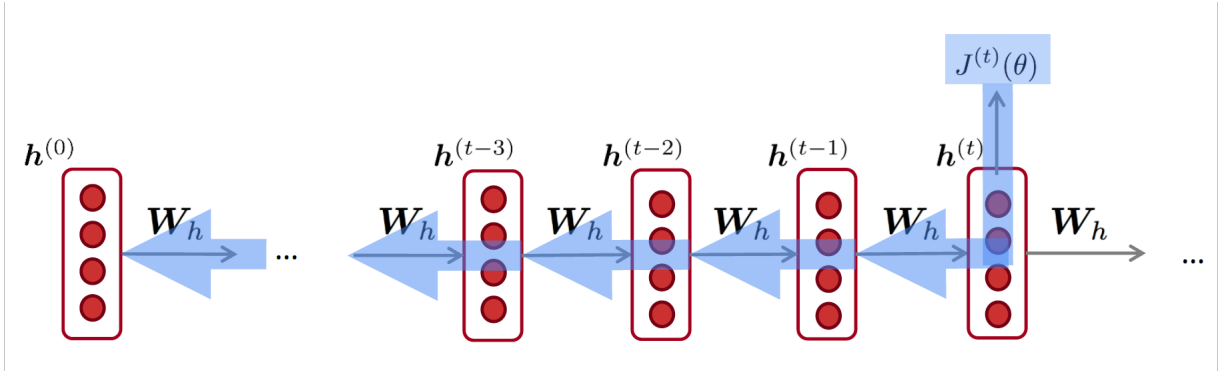
Training an RNN

In practice, we don't calculate the loss of the whole corpus!

We use batches of sentences, compute the loss and update weights

Training an RNN

Backpropagation needs to happen also through time and through the network



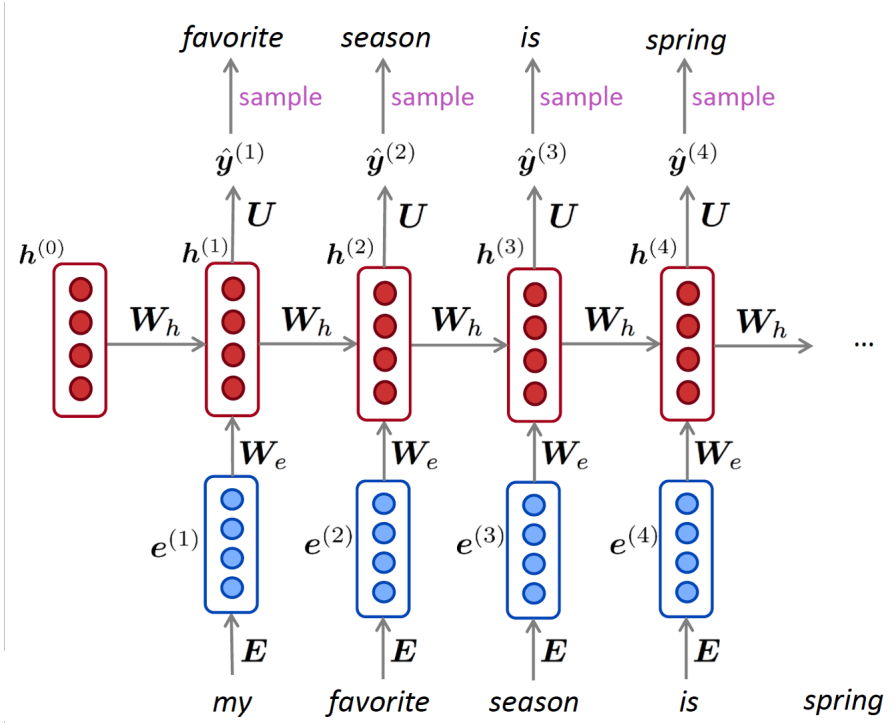
Problems: 1) exploding gradients, 2) vanishing gradients, 3) information is lost after many time steps.

Against exploding gradients, apply **gradient clipping**.

For vanishing gradients? Batch normalization and similar approaches are not as effective

Text generation with RNN

Sampled output is next step's input:

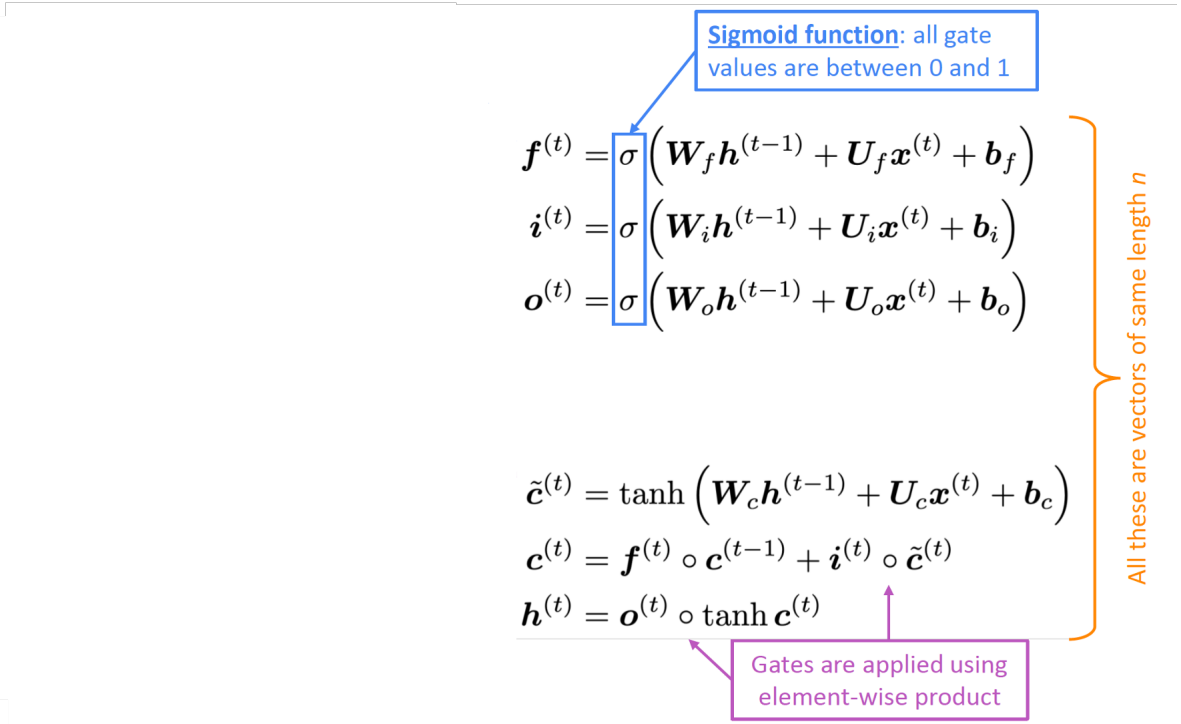


LSTM

Long-short term memory (LSTM) is a cell that improve the simple RNN hidden state layer. We now have a **cell state** to remember information from the past and **gates** to decide what is more important, present or past.

Old information can be kept! Solves also the vanishing gradient problem

LSTM


Sigmoid function: all gate values are between 0 and 1
$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$
$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$
$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$
$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$
$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$
$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$
All these are vectors of same length n
Gates are applied using element-wise product

LSTM

Forget gate: controls what is kept vs forgotten, from previous cell state

Sigmoid function: all gate values are between 0 and 1

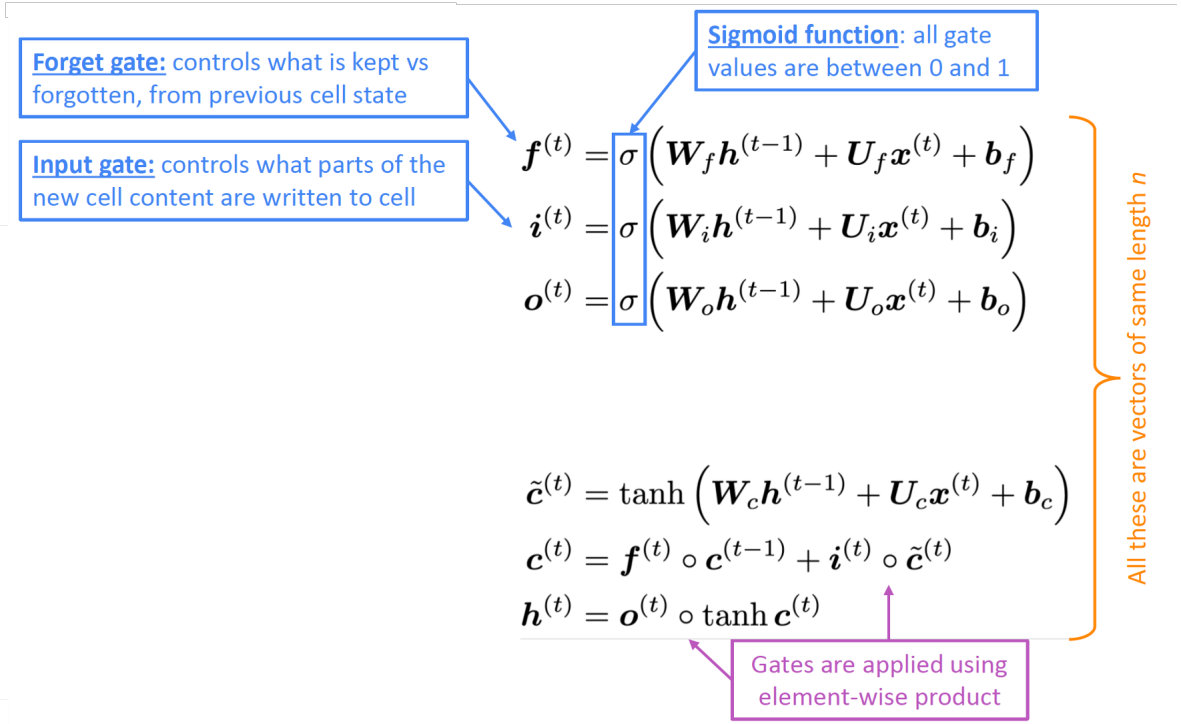
$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

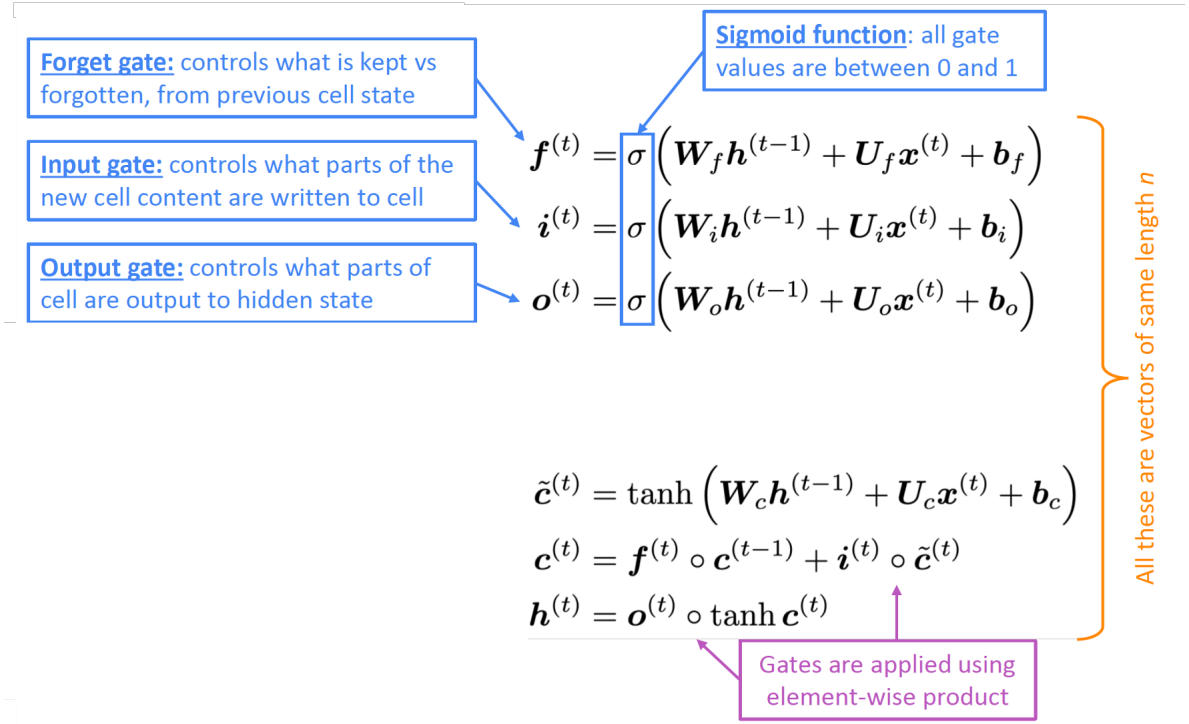
All these are vectors of same length n

Gates are applied using element-wise product

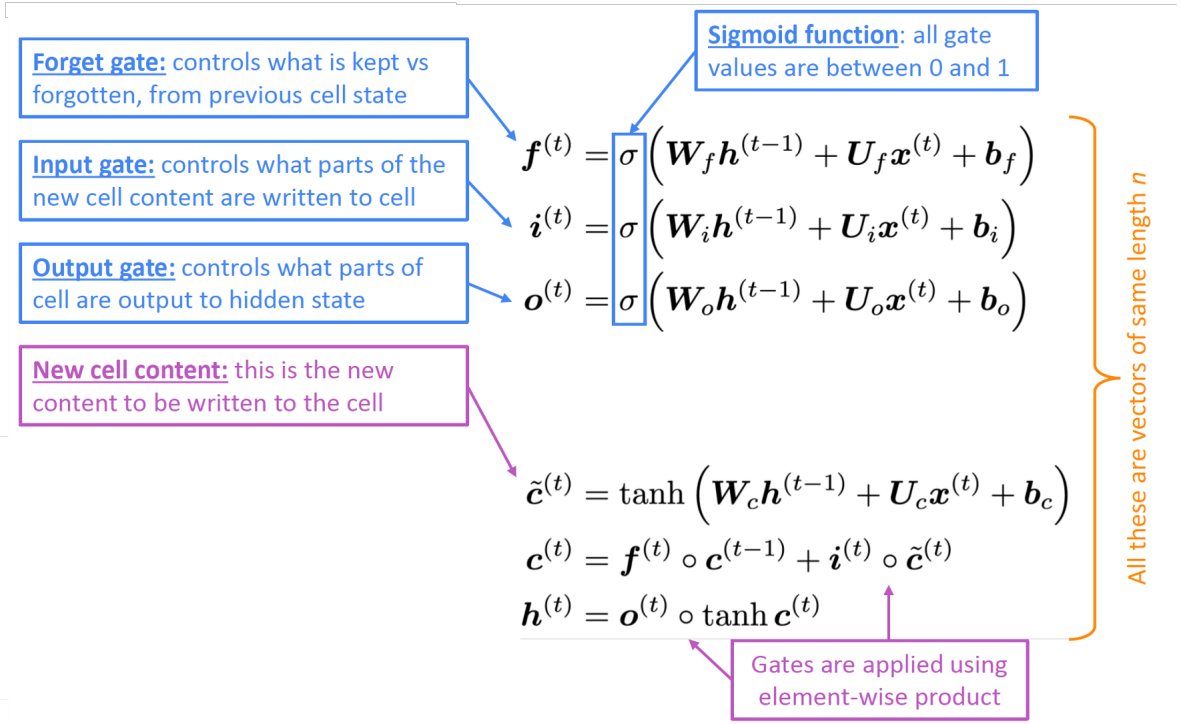
LSTM


Forget gate: controls what is kept vs forgotten, from previous cell state
Input gate: controls what parts of the new cell content are written to cell
Sigmoid function: all gate values are between 0 and 1
$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$
$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$
$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$
$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$
$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$
All these are vectors of same length n
Gates are applied using element-wise product

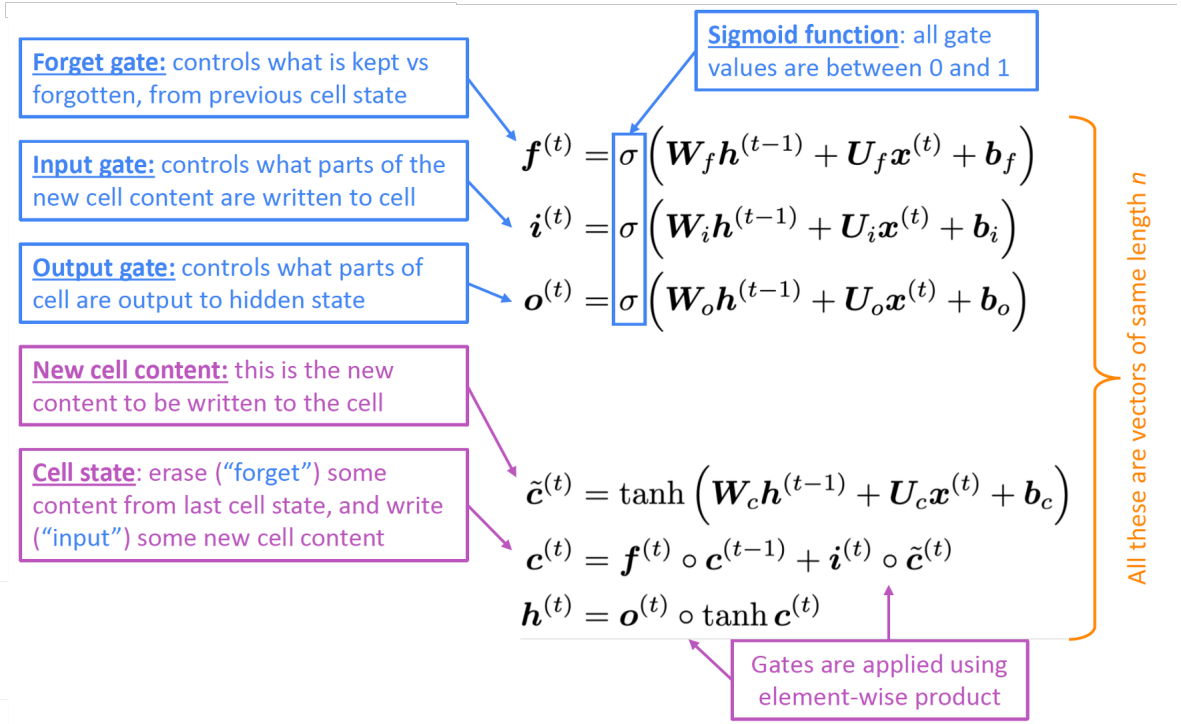
LSTM


Forget gate: controls what is kept vs forgotten, from previous cell state
Input gate: controls what parts of the new cell content are written to cell
Output gate: controls what parts of cell are output to hidden state
Sigmoid function: all gate values are between 0 and 1
$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$
$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$
$o^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$
$\tilde{c}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$
$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$
$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$
All these are vectors of same length n
Gates are applied using element-wise product

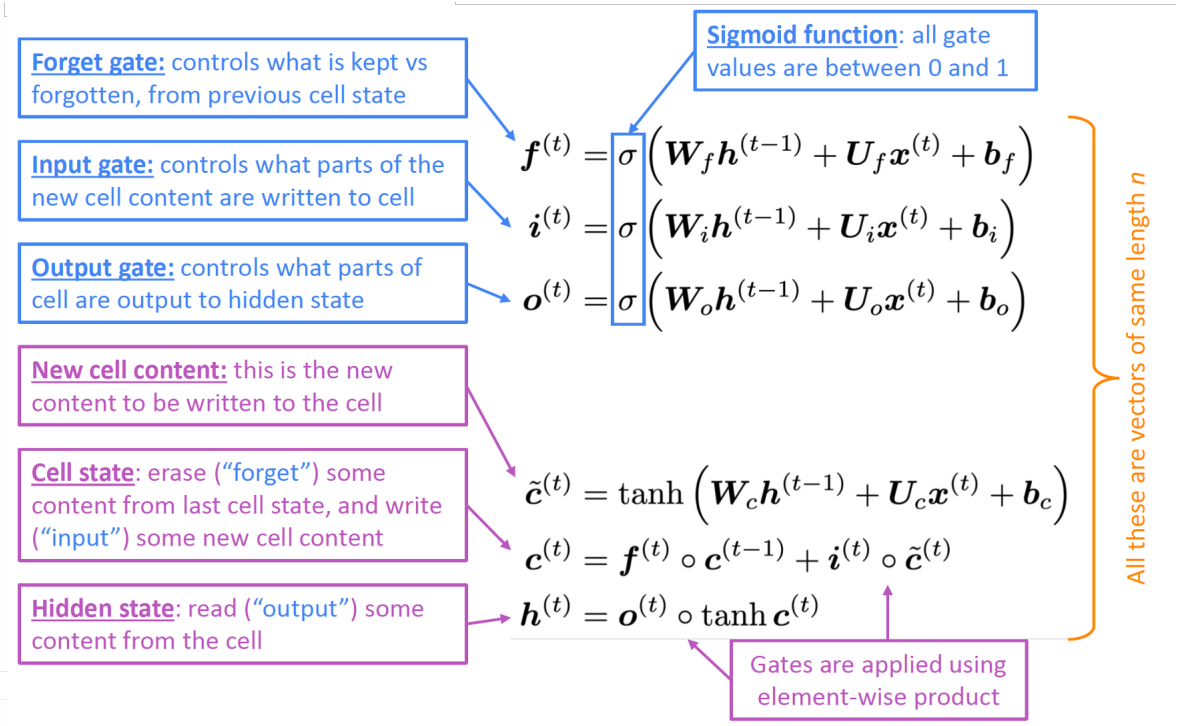
LSTM


Forget gate: controls what is kept vs forgotten, from previous cell state
Input gate: controls what parts of the new cell content are written to cell
Output gate: controls what parts of cell are output to hidden state
New cell content: this is the new content to be written to the cell
Sigmoid function: all gate values are between 0 and 1
$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$
$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$
$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$
$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$
$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$
$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$
All these are vectors of same length n
Gates are applied using element-wise product

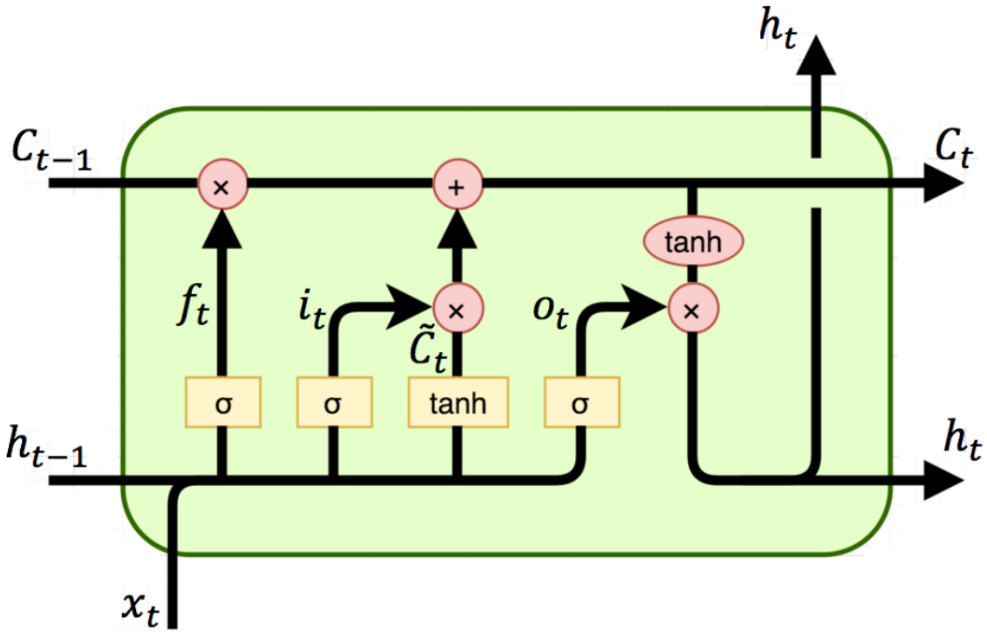
LSTM


Sigmoid function: all gate values are between 0 and 1
Forget gate: controls what is kept vs forgotten, from previous cell state
Input gate: controls what parts of the new cell content are written to cell
Output gate: controls what parts of cell are output to hidden state
New cell content: this is the new content to be written to the cell
Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content
$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$
$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$
$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$
$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$
$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$
$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$
All these are vectors of same length n
Gates are applied using element-wise product

LSTM


Forget gate: controls what is kept vs forgotten, from previous cell state
Input gate: controls what parts of the new cell content are written to cell
Output gate: controls what parts of cell are output to hidden state
New cell content: this is the new content to be written to the cell
Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content
Hidden state: read ("output") some content from the cell
Sigmoid function: all gate values are between 0 and 1
$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$
$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$
$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$
$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$
$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$
$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$
All these are vectors of same length n
Gates are applied using element-wise product

LSTM



RNN Cell

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

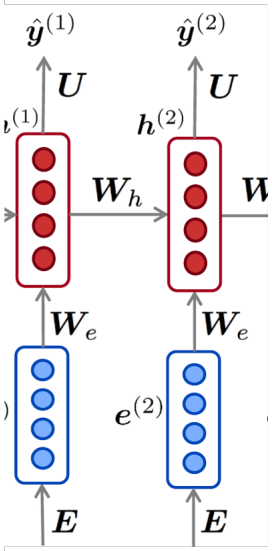
$\boldsymbol{h}^{(0)}$ is the initial hidden state

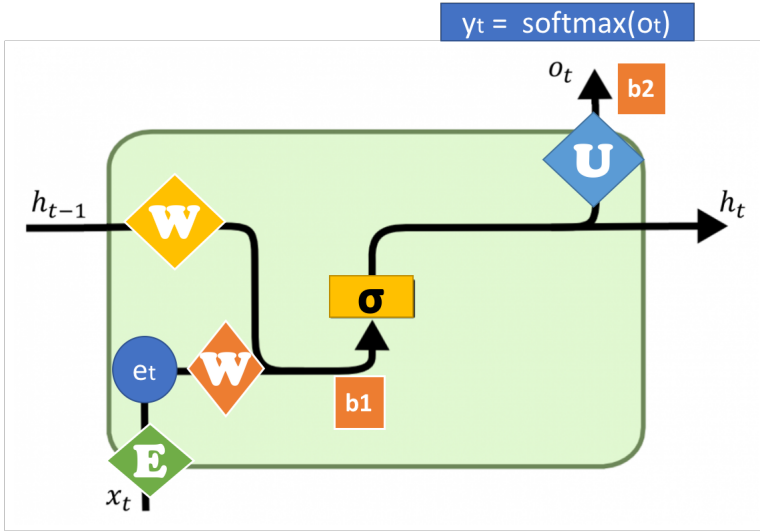
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

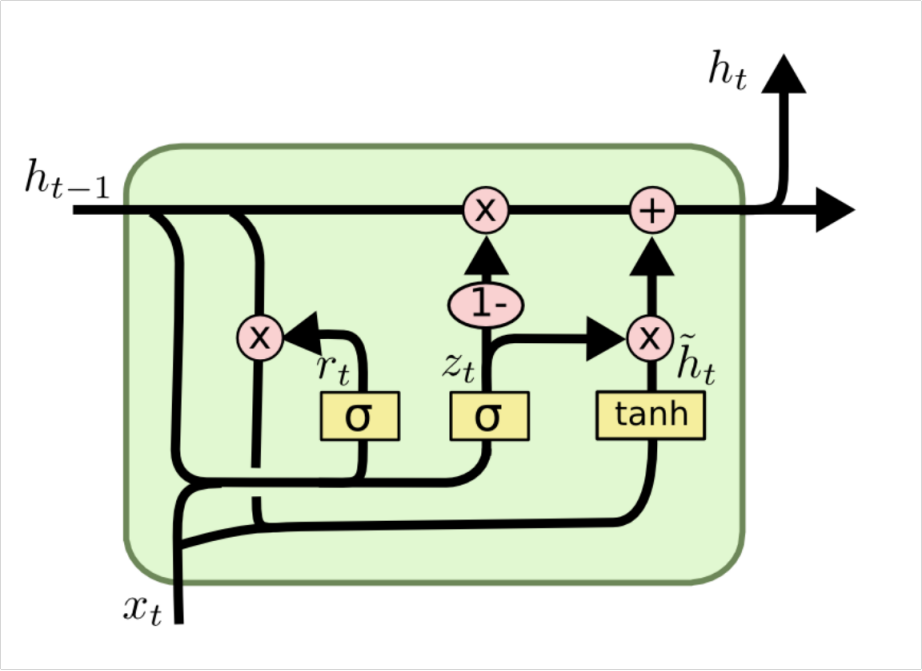
$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$





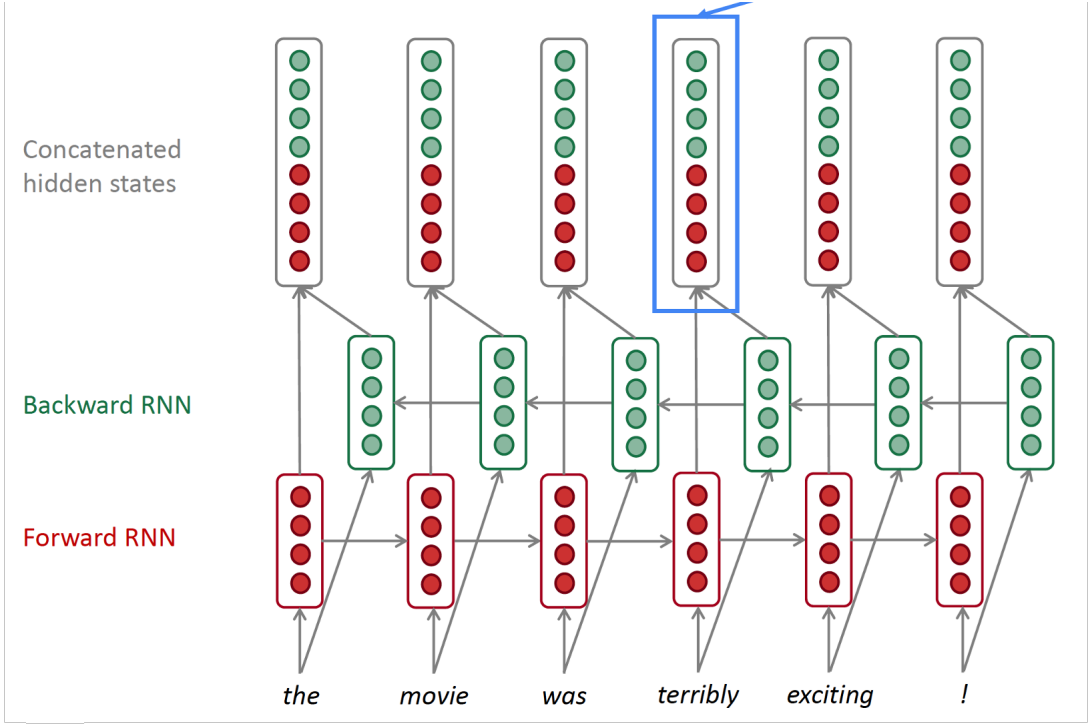
GRU

LSTMs are complicated, with many parameters more than RNNs. Gated Recurrent Unit (GRUs) simplifies the LSTM cell, without taking out its benefits. No more cell state, the hidden state keeps the information from the past



Bidirectional RNNs

Get information from past and future. Only for tasks where the complete sentence is given. (Not to generate text)



Multi-layer RNNs

RNNs can also have more hidden layers! (Normally 2 to 4 are used, otherwise too complex and hard to train)