Universität Konstanz

07 | Recurrent Neural Networks

Giordano De Marzo

https://giordano-demarzo.github.io/

Deep Learning for the Social Sciences

Recap

- Graph Theory
- Machine Learning on Graphs
- Convolution on Graphs
- Graph Convolutional Neural Networks
- Applications

Universität Konstanz

Outline

1. Sequence Data
2. Recurrent Neural Networks
3. Long-Short Term Memories
4. Applications


Sequence Data

Sequence Data

Up to now we considered several types of data: numerical data, images, graph structured data. However many data come in the form of sequences of inputs. Differently from images or graphs, in this case rather than spatial information we have to consider temporal relations. Examples of data structured as sequences are

- language (sequences of words/letters)
- time series (sequences of numbers)
- music (sequences of musical notes)
- videos (sequences of images)

When dealing with sequential data, it is crucial to carry a memory of past data to influence the output.

Time Series Analysis

The forecast of stock prices or market indices is a typical example of sequential data analysis

- is it possible to use one or more time series as input
- also the output can consists in one or more time series
- in this case the number of data points can often be enormous
- capturing both short term and long term trends is a challenge

Image Captioning

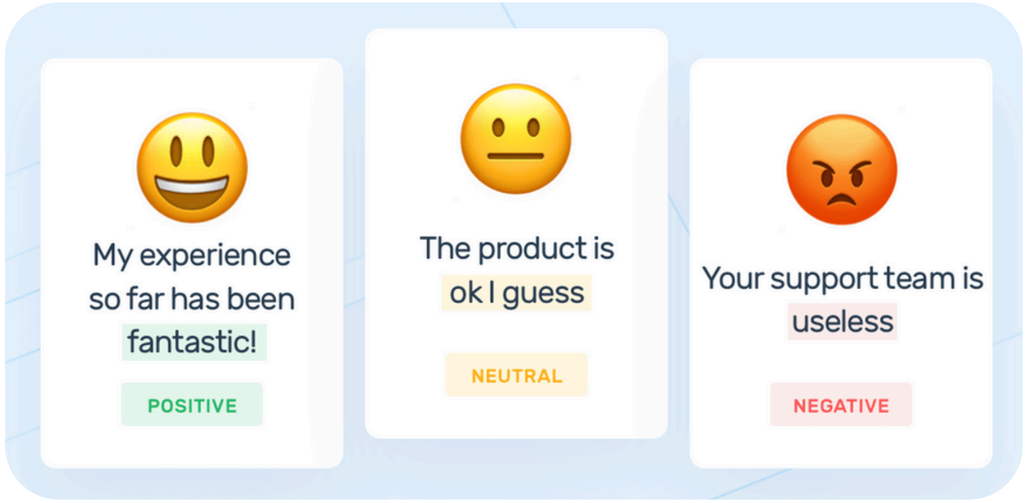
Image captioning consists in associating a short string of text to an image

- the input is an image (typically pre-processed using a CNN)
- the output is a sequence of words
- this is an example of one to many problem
  - the input is a single value (or matrix)
  - the output consists in a whole sequence of variable length

Sentiment Analysis

Sentiment analysis consists in associating a number (sentiment) to a string of text (sequence)

- the input is a string of text, so a sequence of variable length
- the output is the sentiment
- this is an example of many to one problem
    - the input is a sequence of variable lenght
    - the output is a single value (or matrix)



Challenges of Sequence Data

Sequential data pose new challenges that we can not handle with the tools we have

Heterogeneous Length

Similarly to graph, which are characterized by variable dimensions, sequential data will generally have a variable length

Temporal Correlations

Sequential data are dominated by temporal correlations that influence future elements in the sequence

Huge Size

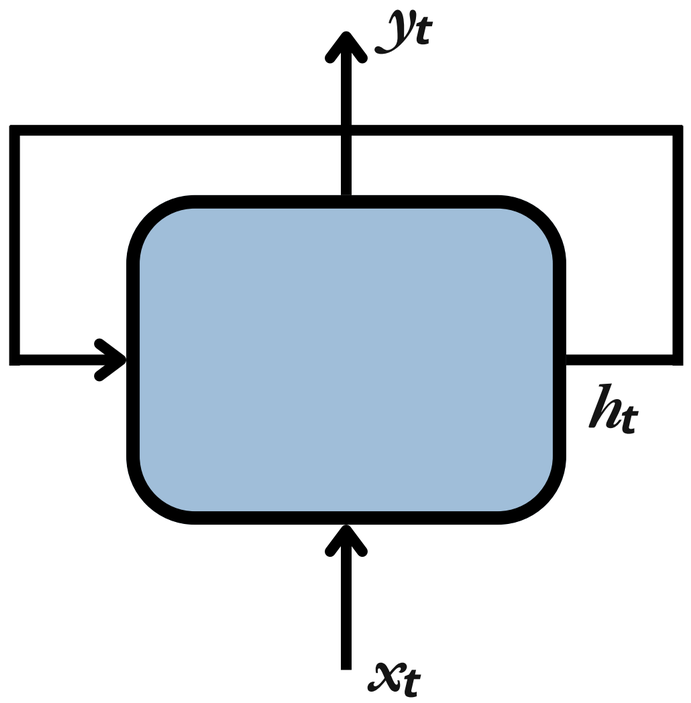
Time series can be extremely long and in areas such as finance we may want to analyze many of them in parallel to determine correlations between different stocks.

Recurrent Neural Networks

Recurrent Connections

In order to capture temporal dependencies in the data we add a feedback loop obtaining a so called Recurrent Neural Network (RNN)

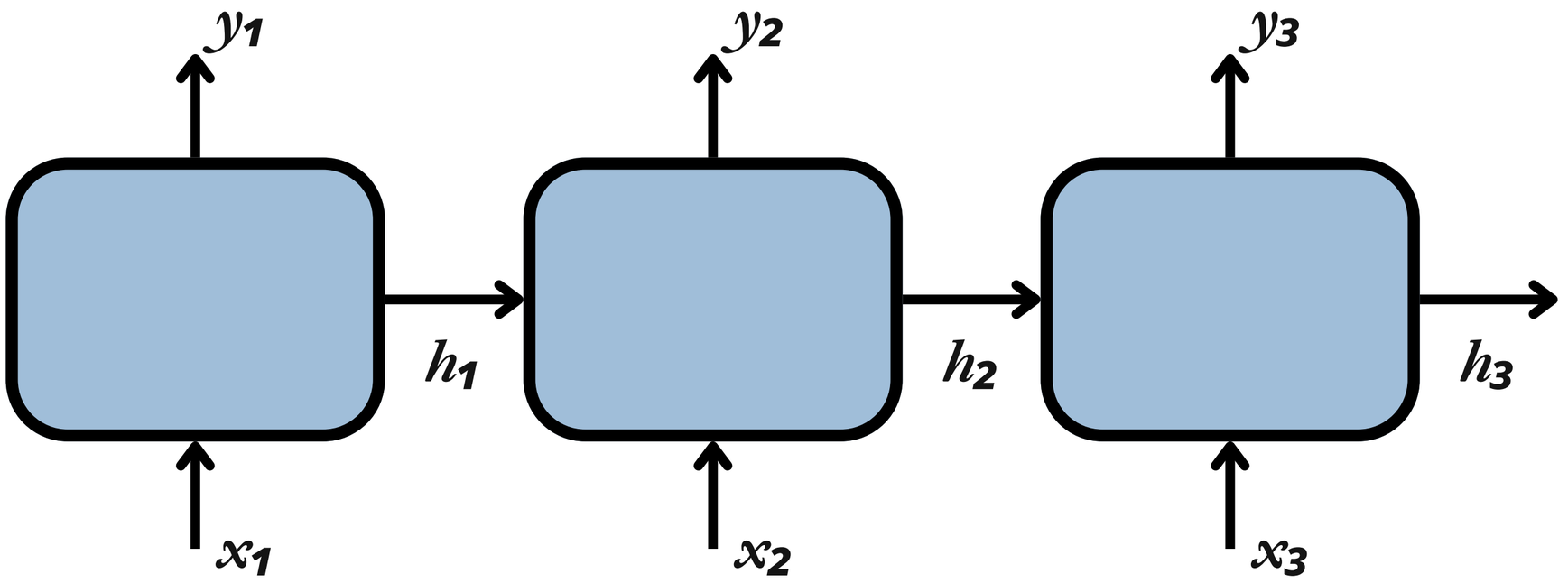
- a RNN is characterized by an hidden state $\mathbf{h_t}$
- at each time step the RNN takes in input the value $\mathbf{x_t}$ from the sequence and the previous hidden state $\mathbf{h_{t-1}}$
- it then produces and output y_t and a new hidden state $\mathbf{h_{t-1}}$



$$h_t = f_W(h_{t-1}, x_t)$$
$$y_t = g_W(h_t)$$

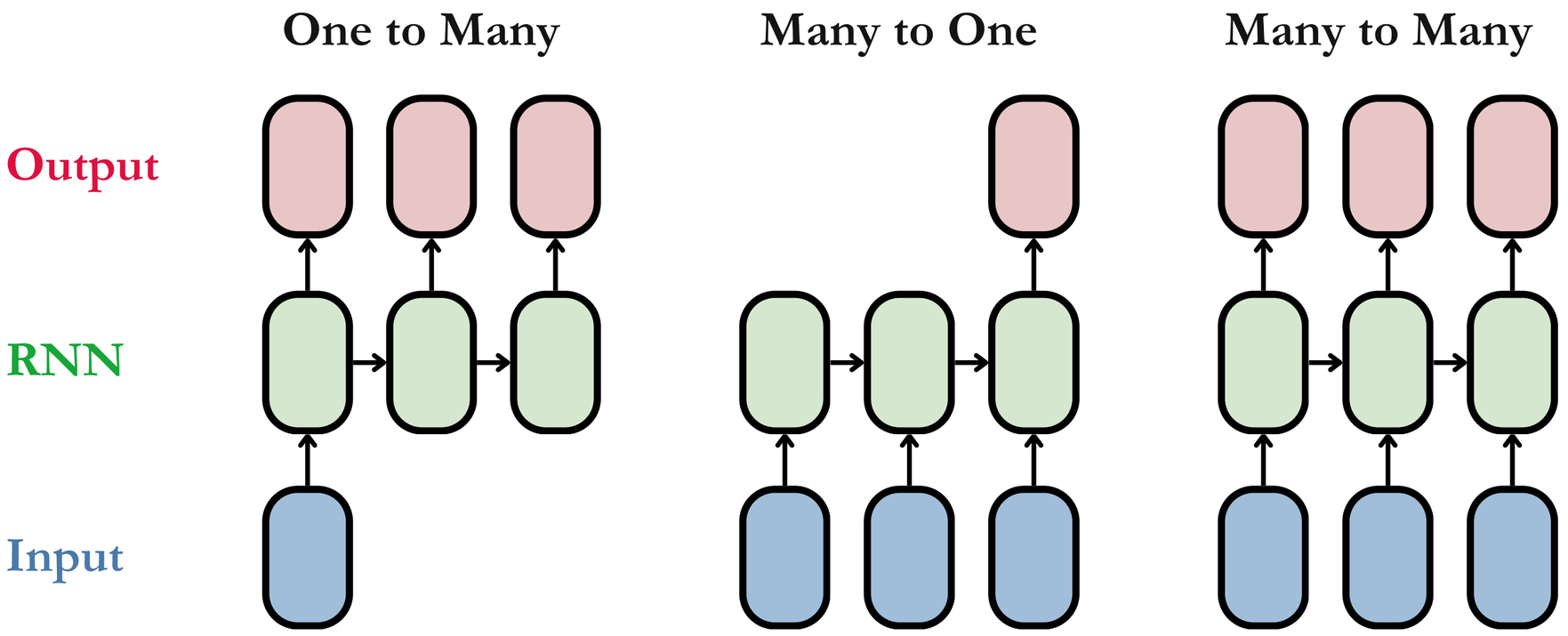
Unfolding a RNN

We can visualize a RNN by unfolding its feedback look. In this way the RNN can be seen as a series of multiple copies of the same block, each taking in input a different value from the input sequence. The parameters are the same for each block.



Examples of Tasks

RNNs find applications in fields ranging from NLP to Computer Vision.



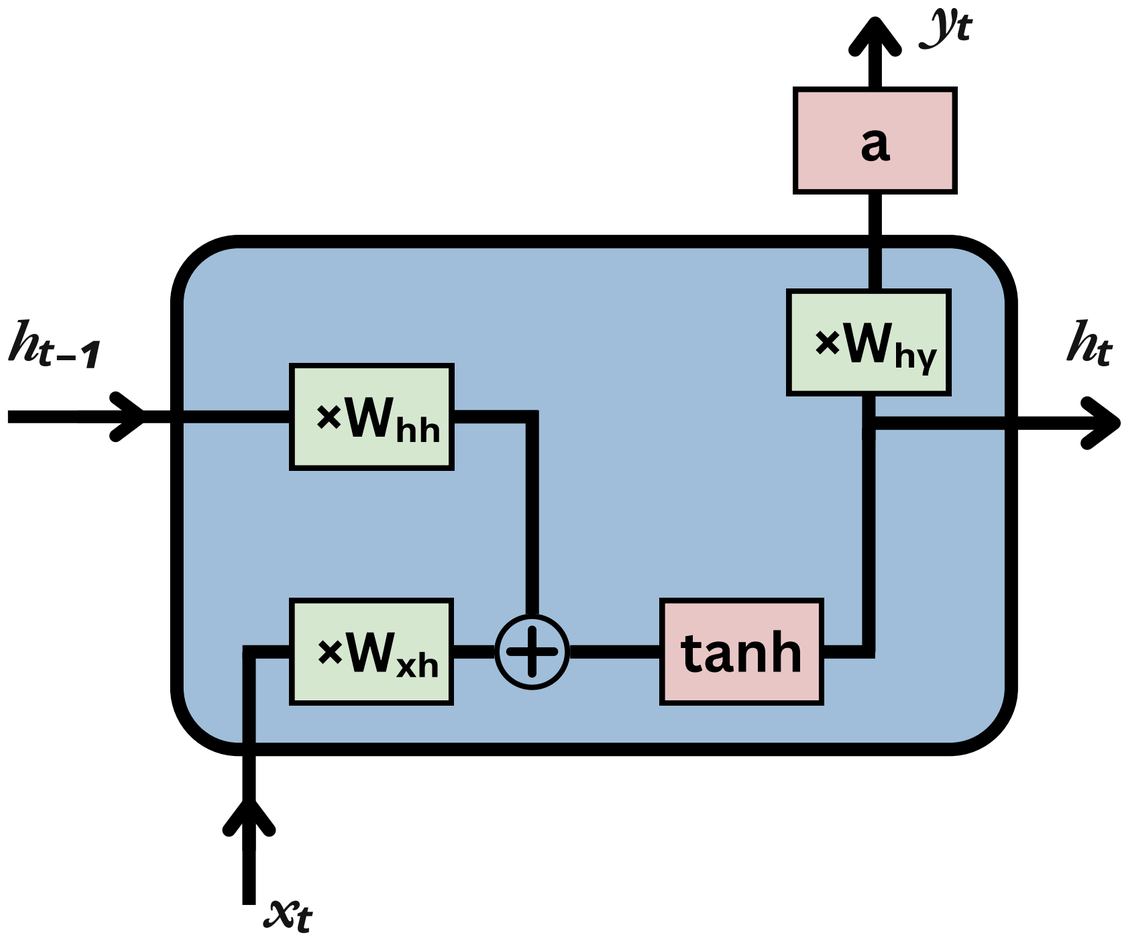
Vanilla RNN

The Vanilla RNN is the most simple approach to the problem

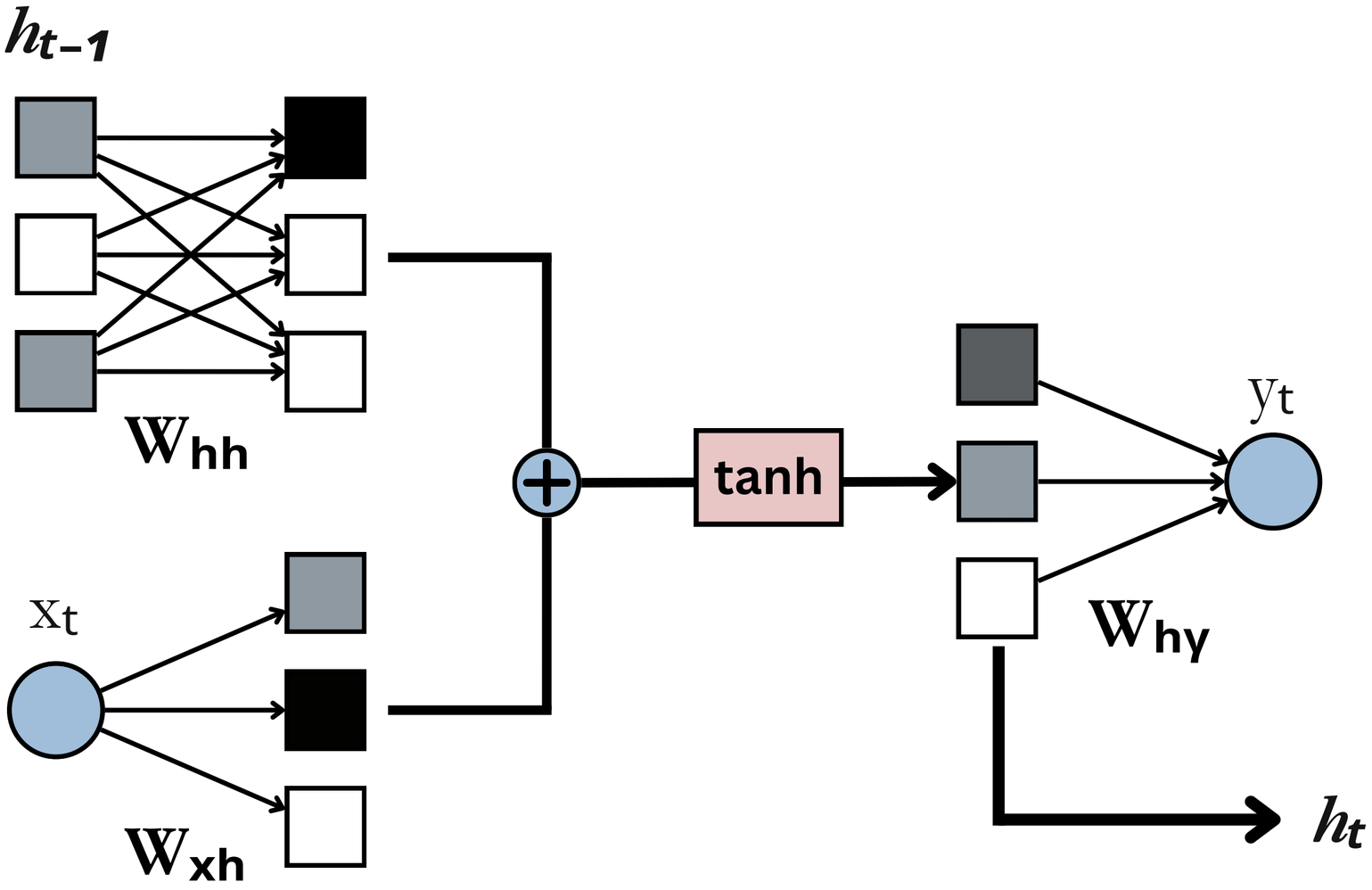
- we have three matrices of learnable parameters
- the output activation a depends on the task
- the Vanilla RNN is defined by the following equations

$$\mathbf{h_t} = \tanh(\mathbf{W_{hh}}\mathbf{h_{t-1}} + \mathbf{W_{xh}}\mathbf{x_t})$$

$$\mathbf{y_t} = a(\mathbf{W_{hy}}\mathbf{h_t})$$

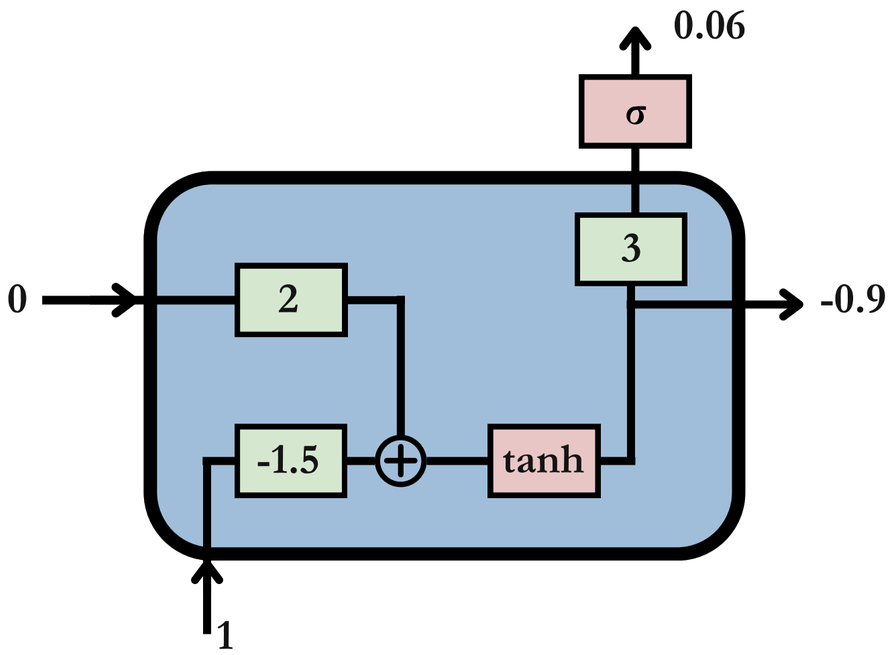


Understading the Vanilla RNN



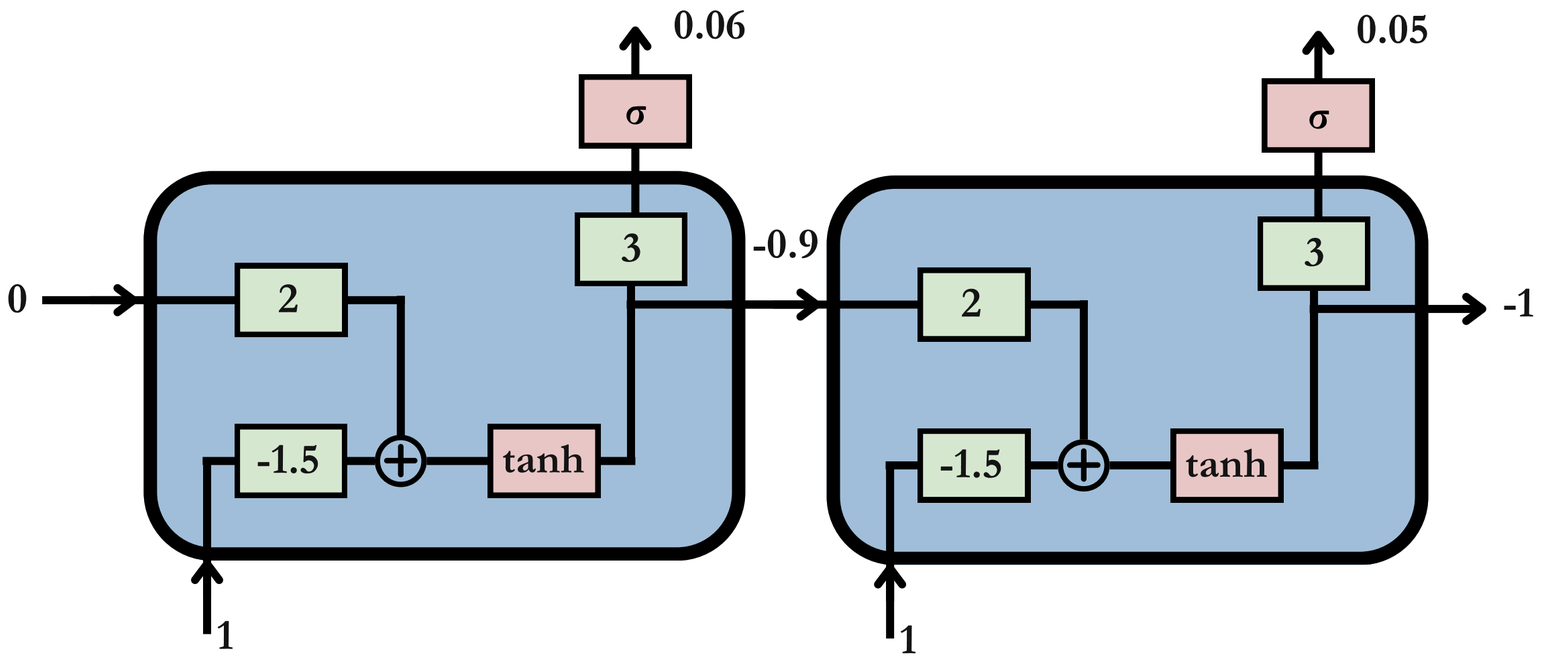
Vanilla RNN Example

For simplicity we consider monodimensional input and hidden vectors and we set $W_{hh}=2$, $W_x=-1.5$ and $W_{hy}=3$. The input sequence is 1, 1, -2.



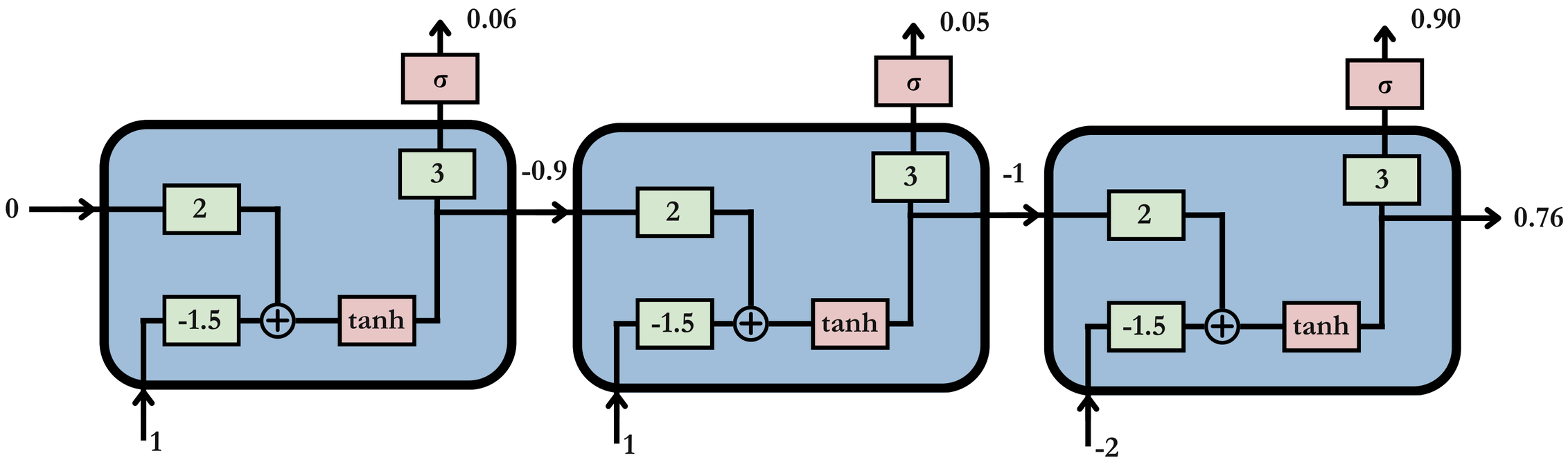
Vanilla RNN Example

For simplicity we consider monodimensional input and hidden vectors and we set $W_{hh}=2$, $W_x=-1.5$ and $W_{hy}=3$. The input sequence is 1, 1, -2.



Vanilla RNN Example

For simplicity we consider monodimensional input and hidden vectors and we set $W_{hh}=2$, $W_x=-1.5$ and $W_{hy}=3$. The input sequence is 1, 1, -2.



Parameters Sharing

One of the key properties of RNN is parameter sharing:

- similar to CNN and GNN
- the same weight matrices are used for all time steps
- this adds robustness, but relies on the stationariety assumption

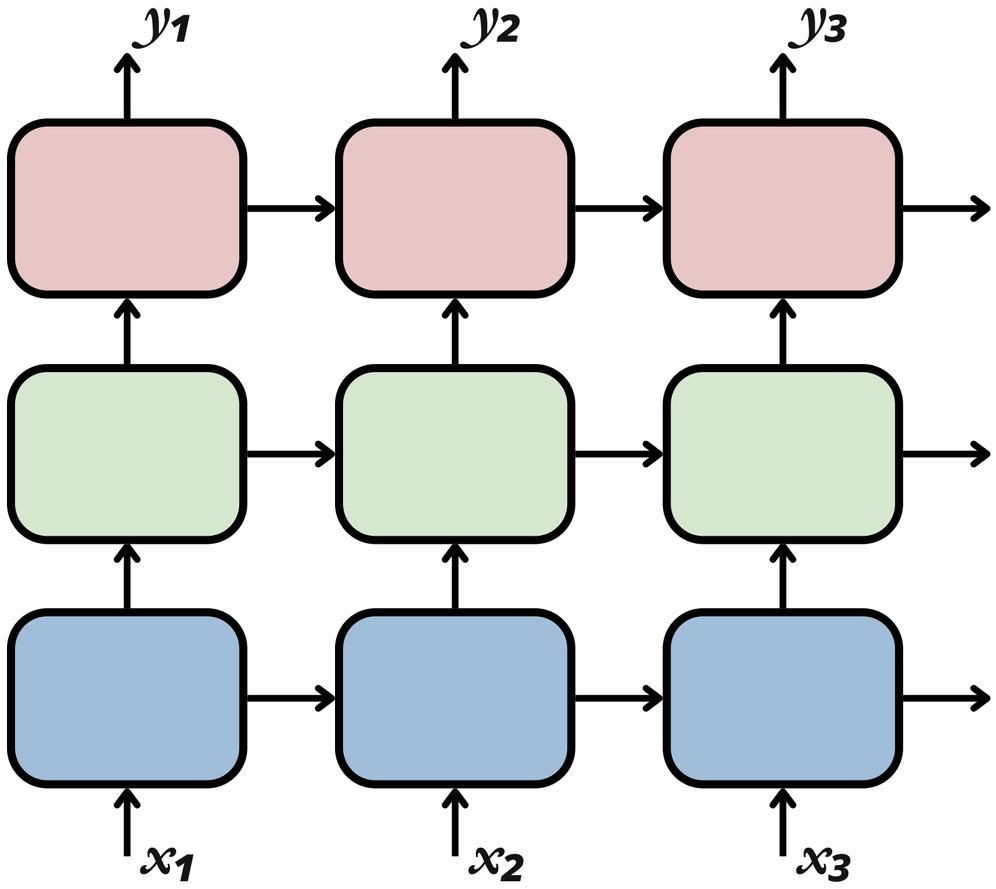
In practice a RNN is like a MLP, but with the same weights matrices used over and over again

- in a MLP the optimization algorithm updates each matrix independently
- in a RNN, the updates for each layer are averaged so to get a global update for the shared weight matrix

Multilayer RNN

Like for standard NN, we can capture more complex features by stacking multiple RRNs

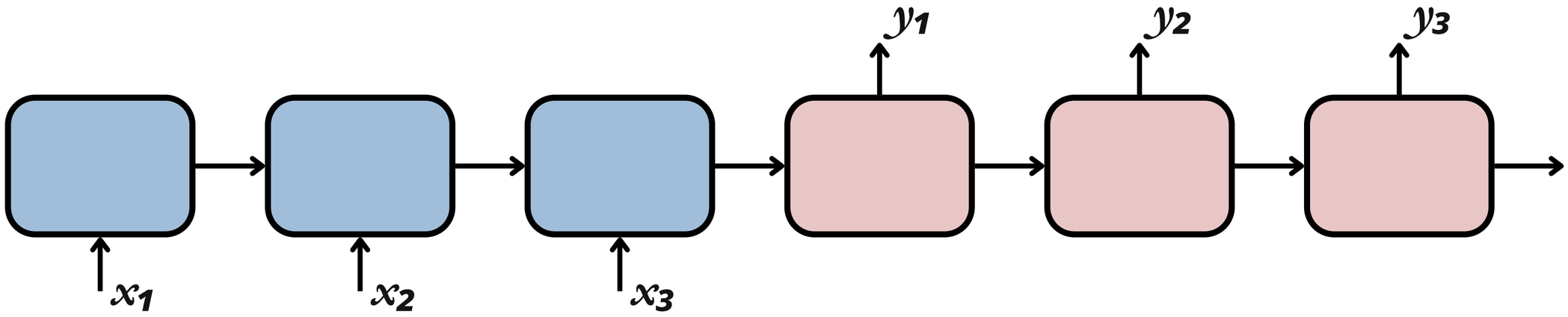
- we add hidden layers to the network
- the hidden state of a layer is feed as input in the following layer
- typically 2 or 3 layers are a reasonable choice



Encoder-Decoder RNNs

NLP is one of the field where RNNs have been applied the most. The Encoder-Decoded RNN is an architecture that allows to perform machine translation

- it uses a first RNN with matrix W_1 to get the hidden representation of a sequence (h_3 in the figure)
- this hidden representation is feed into another RNN with a different weight matrix W_2 whose output is a new sequence



Gradiente Vanishing and Explosion

Despite their versatility and their ability to handle temporal data, Vanilla RNNs are characterized by a fatal flaw that makes it very hard to apply these neural networks to very long sequences

- at each time step we use the same weight matrix to update the hidden state and produce an output
- this implies that when we use the backpropagation to compute the gradient, we will have to multiply many time the same matrix
- this is a problem because this will cause the gradient either to explode or to vanish
    - if the gradient explodes we make jumps that are to large to find the minimum
    - if the gradient vanishes we need too many steps to reach the minimum


Universität
Konstanz
Long Short Term Memories

Limits of Vanilla RNNs

As we already saw Vanilla RNNs have important limits

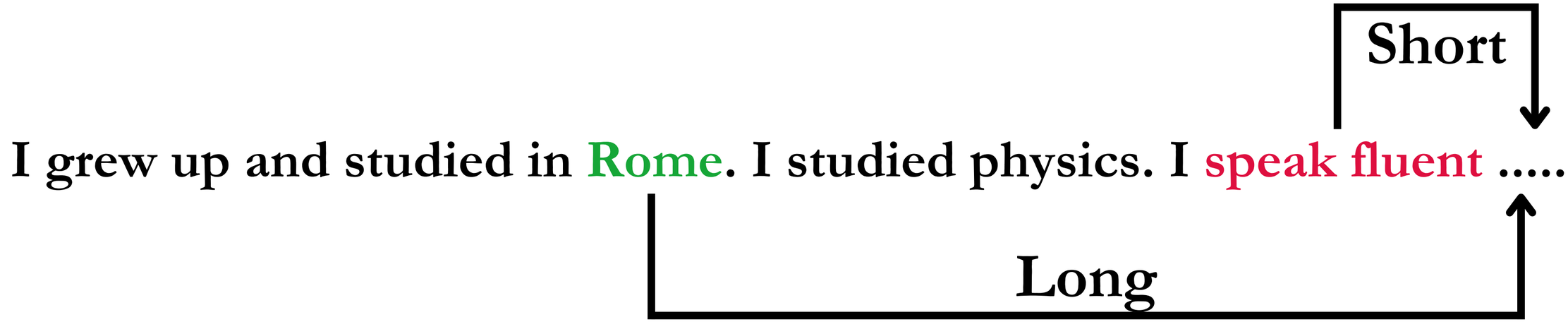
- gradient vanishing and explosions
- can be applied only to short sequences (~20 time steps)
- they struggle in capturing long term dependencies in the data

In order to improve the performances of Vanilla RNNs we have to

- use more complex units that mitigate the gradient issues
- handle longer sequences
- capture long time dependencies

Long and Short Dependencies

A relevant aspect we want our RNNs to capture is the presence of both short term (previous words) and long term (beginning of the paragraph) dependencies in sequence data. A RNN must handle both, since the latter is crucial for capturing the context.

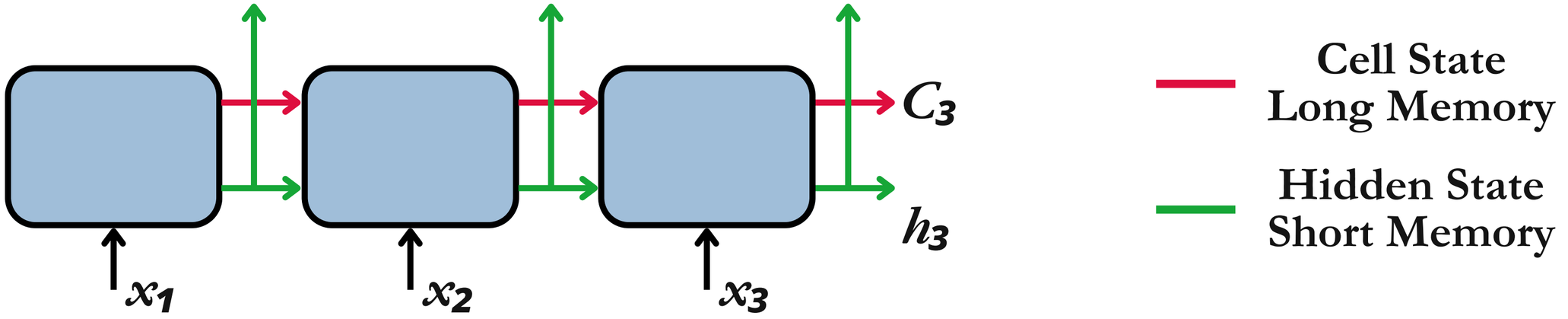


Long Short Term Memories

Long Short Term Memories (LSTM) are a more powerful type of RNN that handle both long and short dependencies:

- long term memories are stored in the cell state ***C***
- short term memories are stored in the hidden state ***h***

The output of the network is computed only using the hidden state ***h***

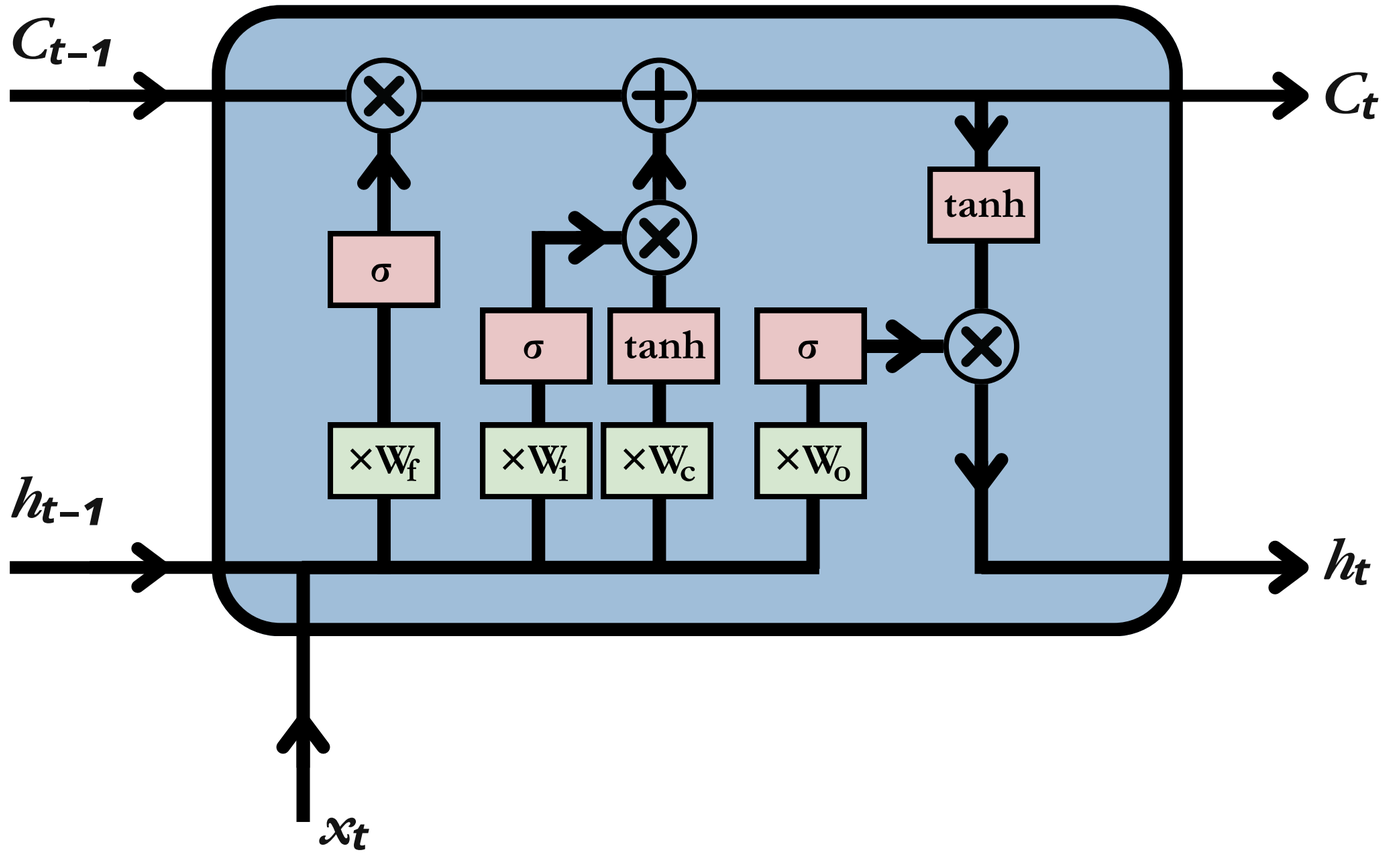


LSTM Unit

The LSTM is composed of

- 4 weight matrices
- 4 gates

Gates are handles reading and writing to and from the hidden and cell state. Like any RNN, the state from the previous time step is combined with the input to update the state



LSTM Mathematical Representation

We stack the 4 weight matrices in a big matrix **W**

- we have 4 gates
  - input i
  - forget f
  - output o
  - cell activation g
- ⊙ denotes elementwise multiplication
- the 4 gates are used to update h and c

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

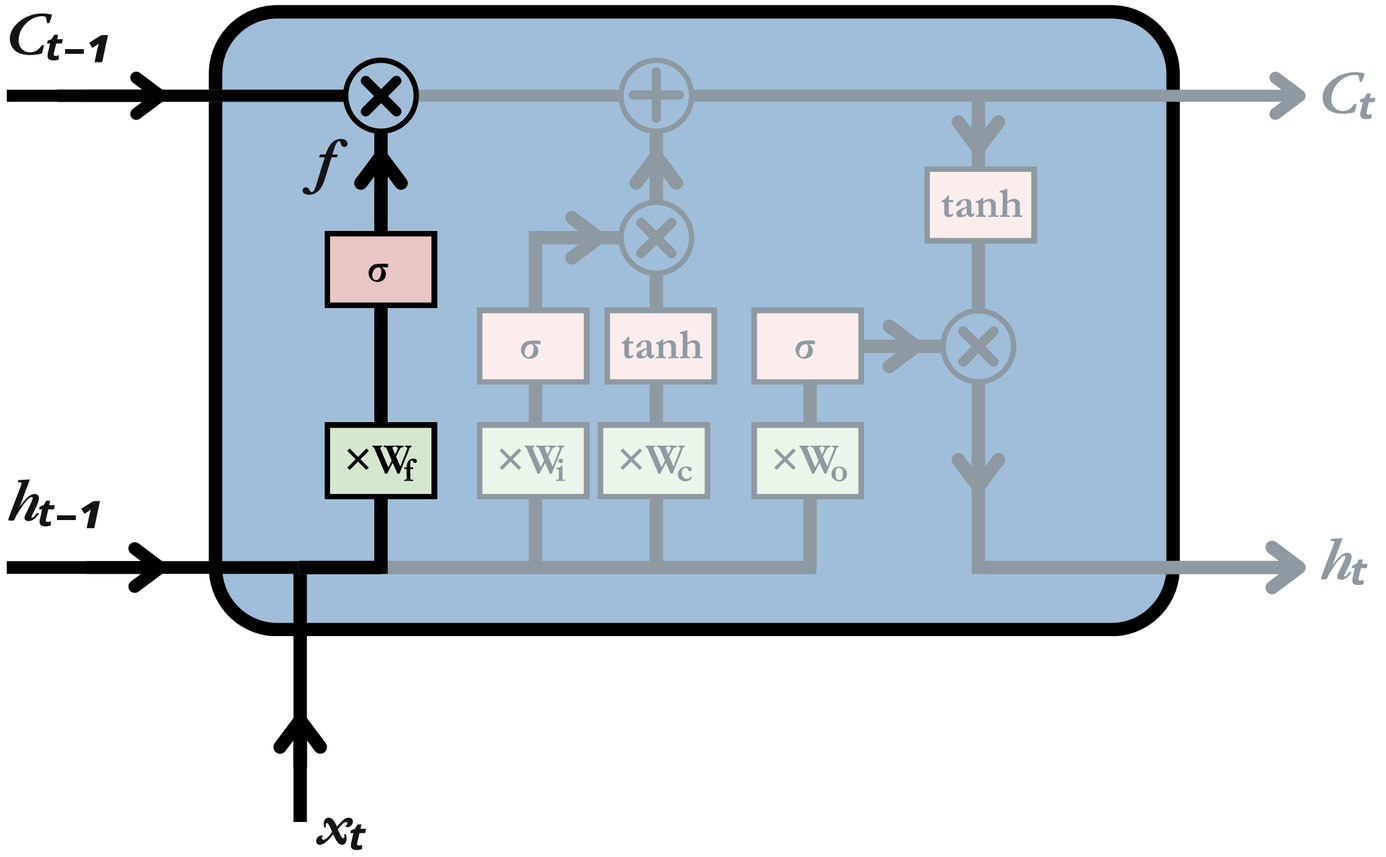
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Forget Gate

The forget gates determines which percentage of the cell state C_{t-1} the network should keep in memory

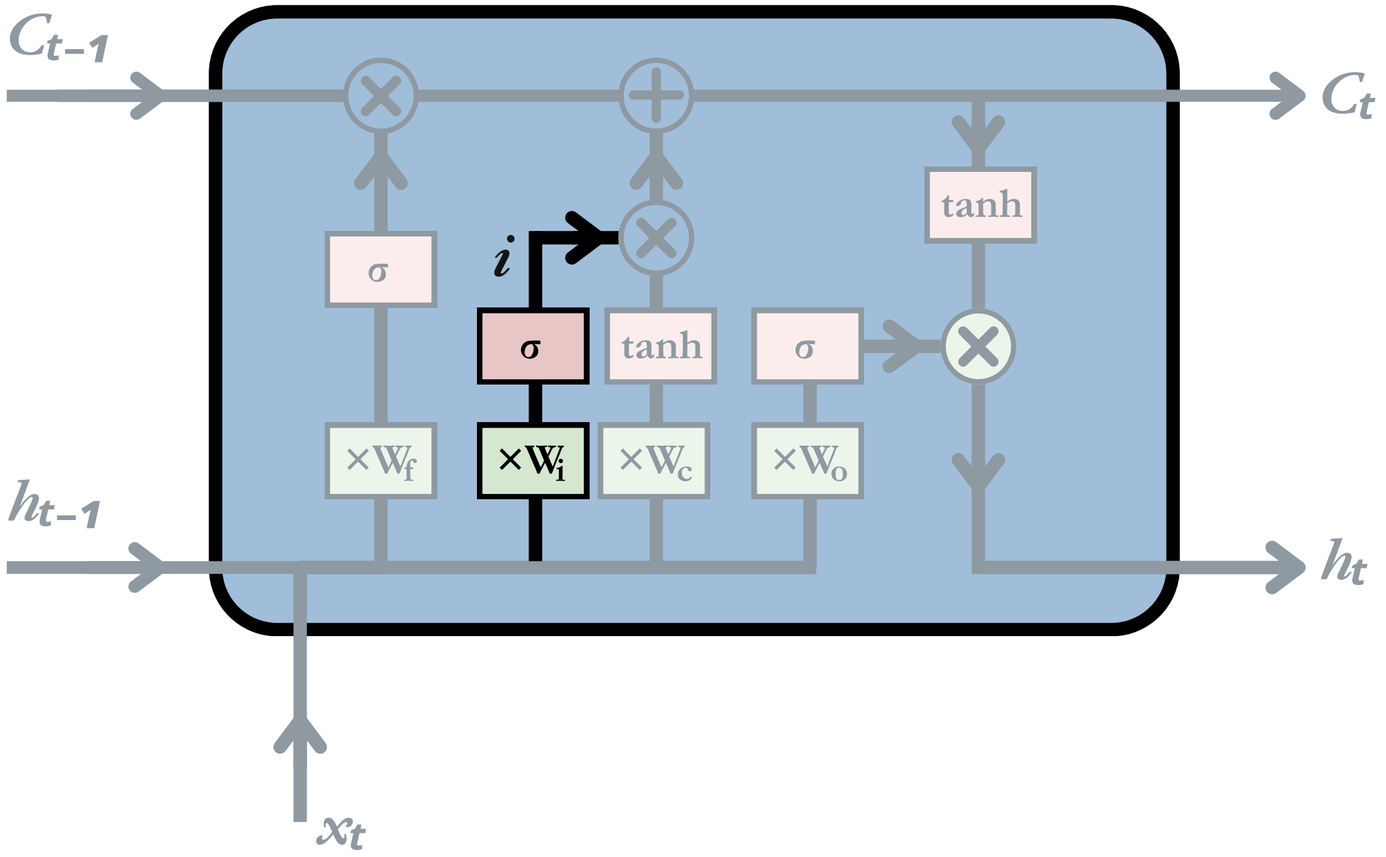
- W_f transforms (h_{t-1}, x_t)
- the sigmoid transforms each element in a percentage producing f
- C_{t-1} is multiplied by f



Input Gate

The forget gate determines which percentage of the cell activation gate ***g*** to add to the cell state

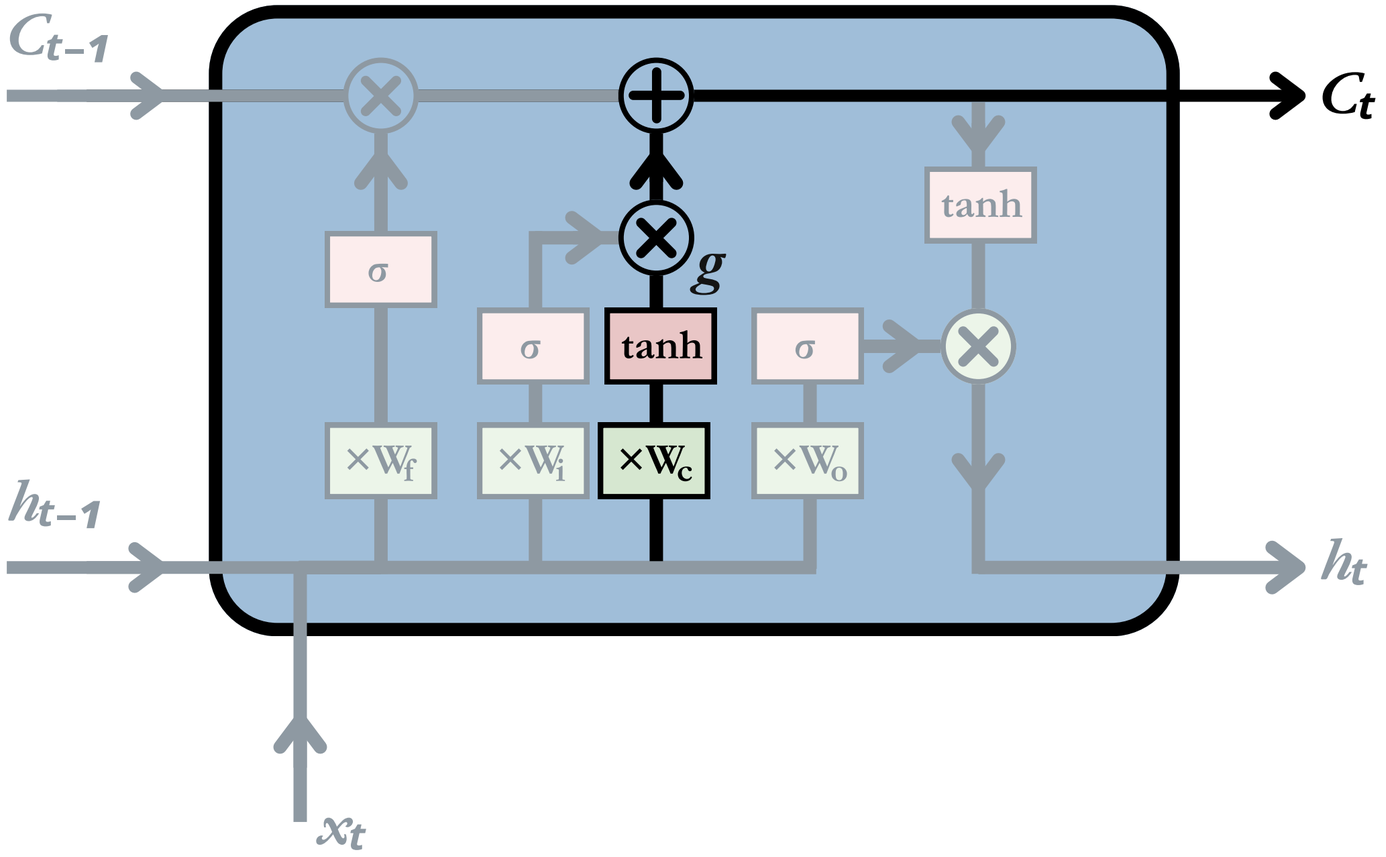
- $\mathbf{W_i}$ transforms $(\mathbf{h_{t-1}}, \mathbf{x_t})$
- the sigmoid transforms each element in a percentage producing ***i***
- ***g*** is multiplied by ***i***



Cell Activation Gate

The cell activation gate determines what to write on the cell state

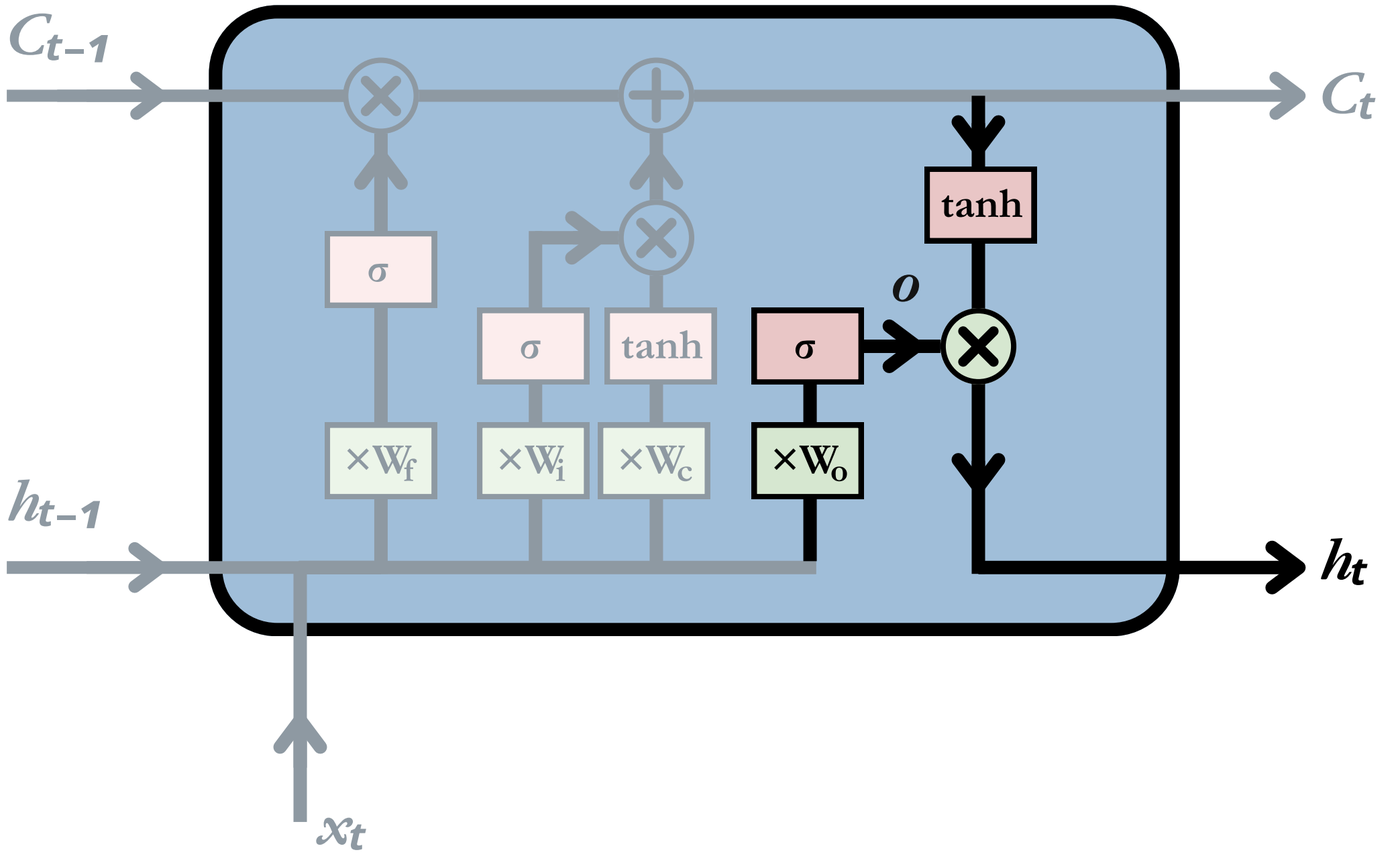
- $\mathbf{W_c}$ transforms $(\mathbf{h_{t-1}}, \mathbf{x_t})$
- the tanh transforms each element in a number in [-1,1] producing $\boldsymbol{g}$
- $\boldsymbol{g}$ is multiplied by $\boldsymbol{i}$ and added to the cell state producing $\boldsymbol{C_t}$



Output Gate

The output gate determines the new hidden state

- C_t is normalized to [-1,1] using a tanh
- W_o transforms (h_{t-1}, x_t)
- the sigmoid transforms each element into a percentage producing o
- o determines which percentage of the normalized C_t goes into h_t

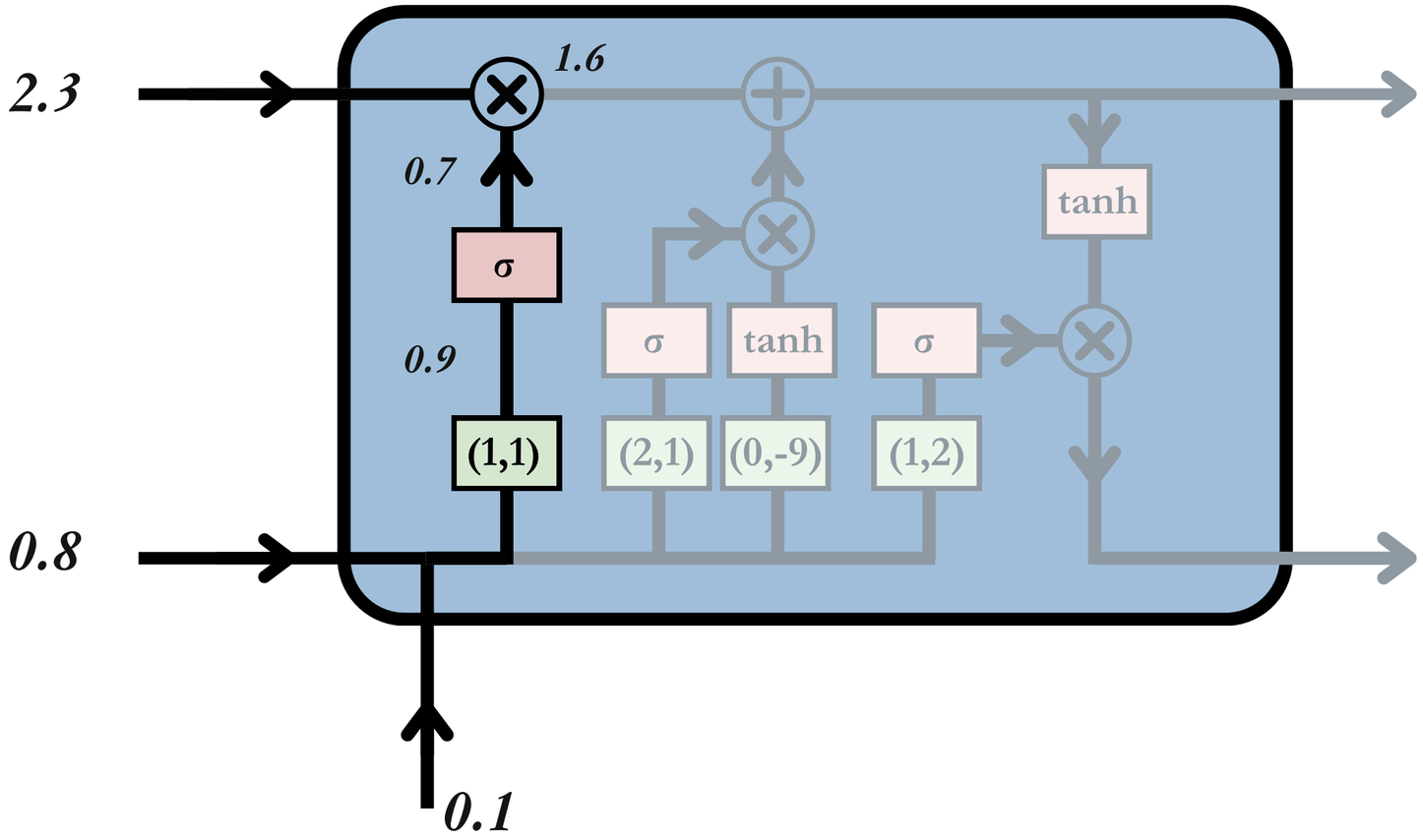


LSTM Example: Forget Gate

We consider a simple monodimensional example

- $C_{t-1}=2.3$
- $h_{t-1}=0.8$
- $x_t=0.1$

The 4 matrices are now vectors with two components.

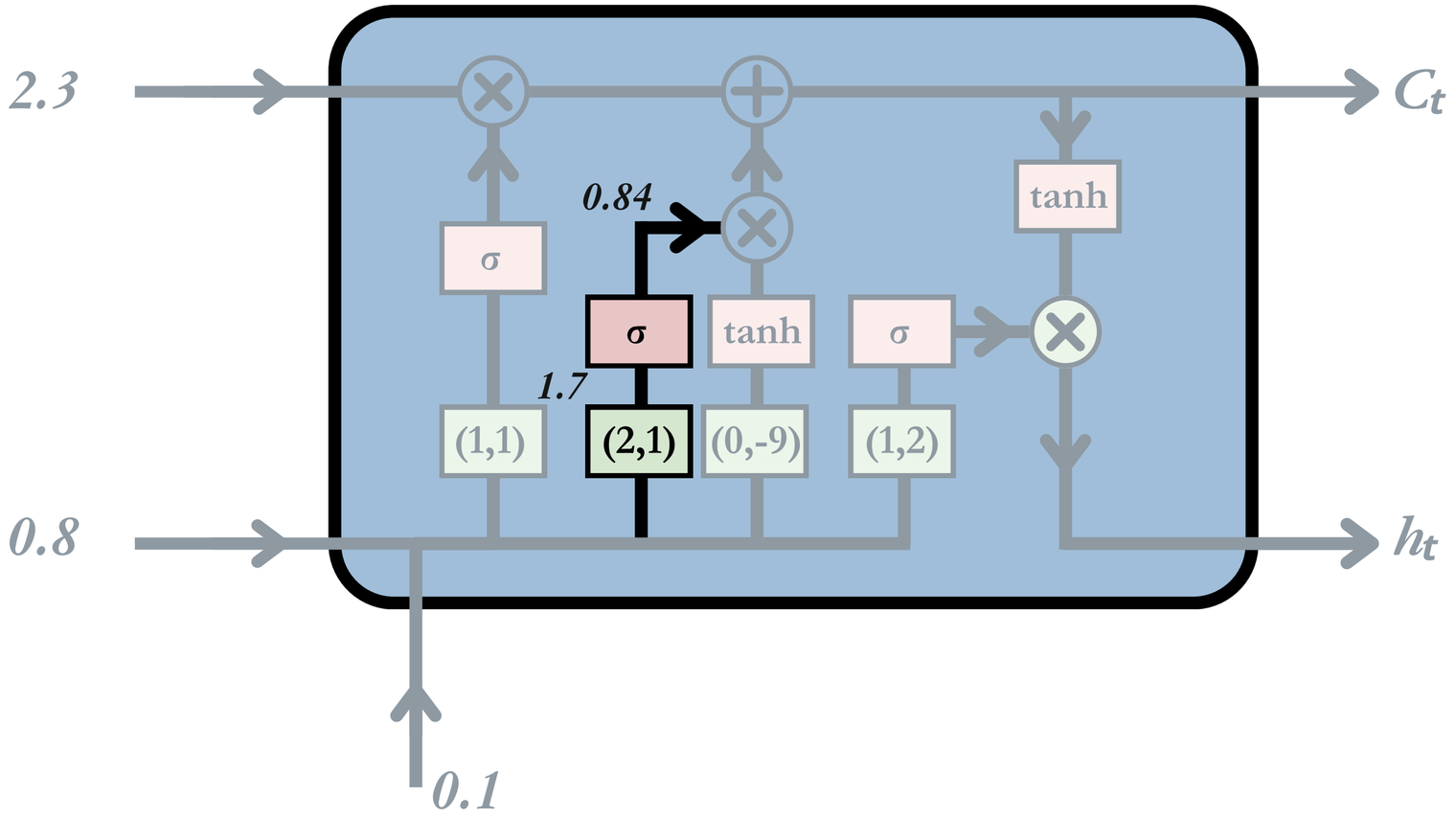


LSTM Example: Input Gate

We consider a simple monodimensional example

- $C_{t-1}=2.3$
- $h_{t-1}=0.8$
- $x_t=0.1$

The 4 matrices are now vectors with two components.

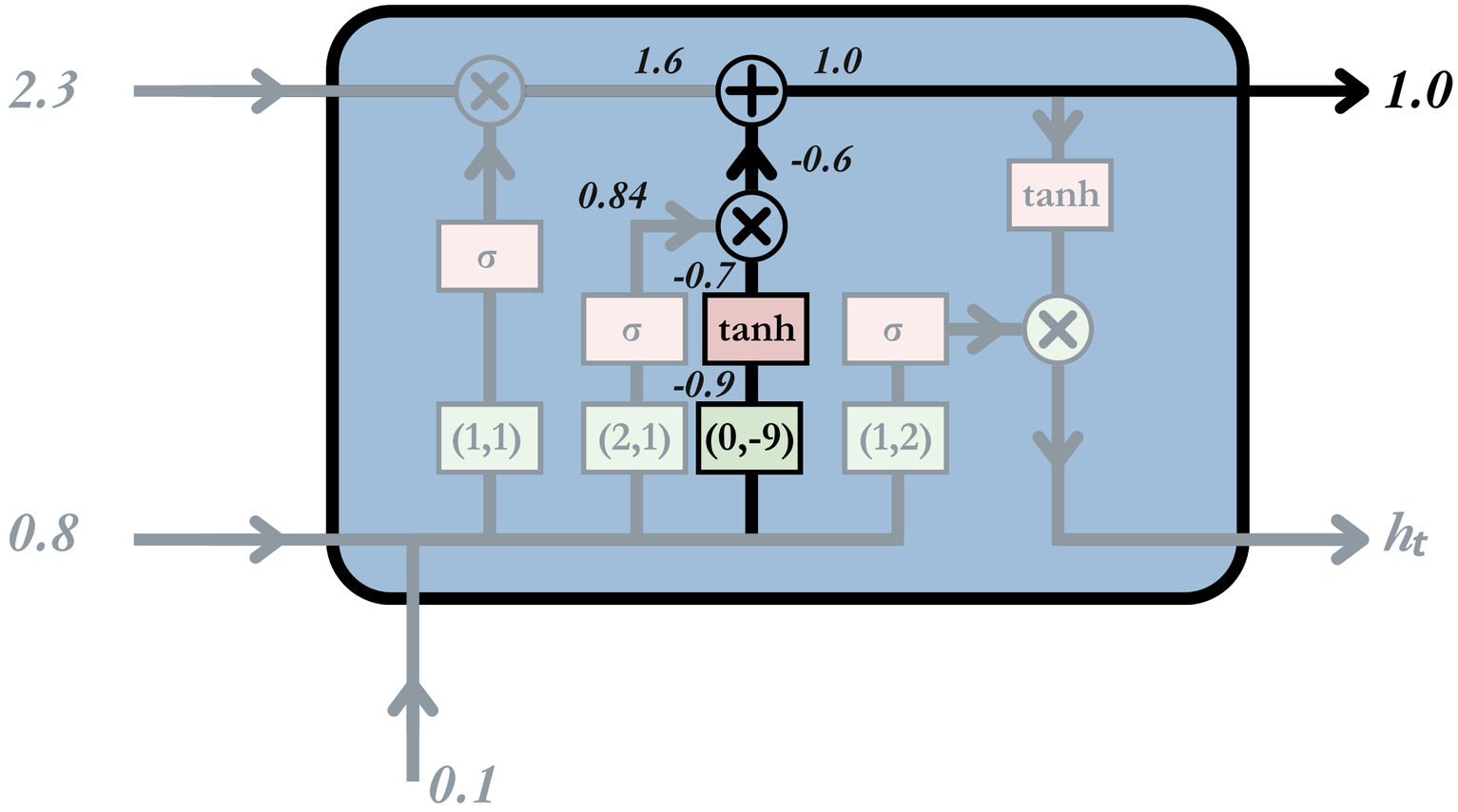


LSTM Example: Cell State Update

We consider a simple monodimensional example

- $C_{t-1}=2.3$
- $h_{t-1}=0.8$
- $x_t=0.1$

The 4 matrices are now vectors with two components.

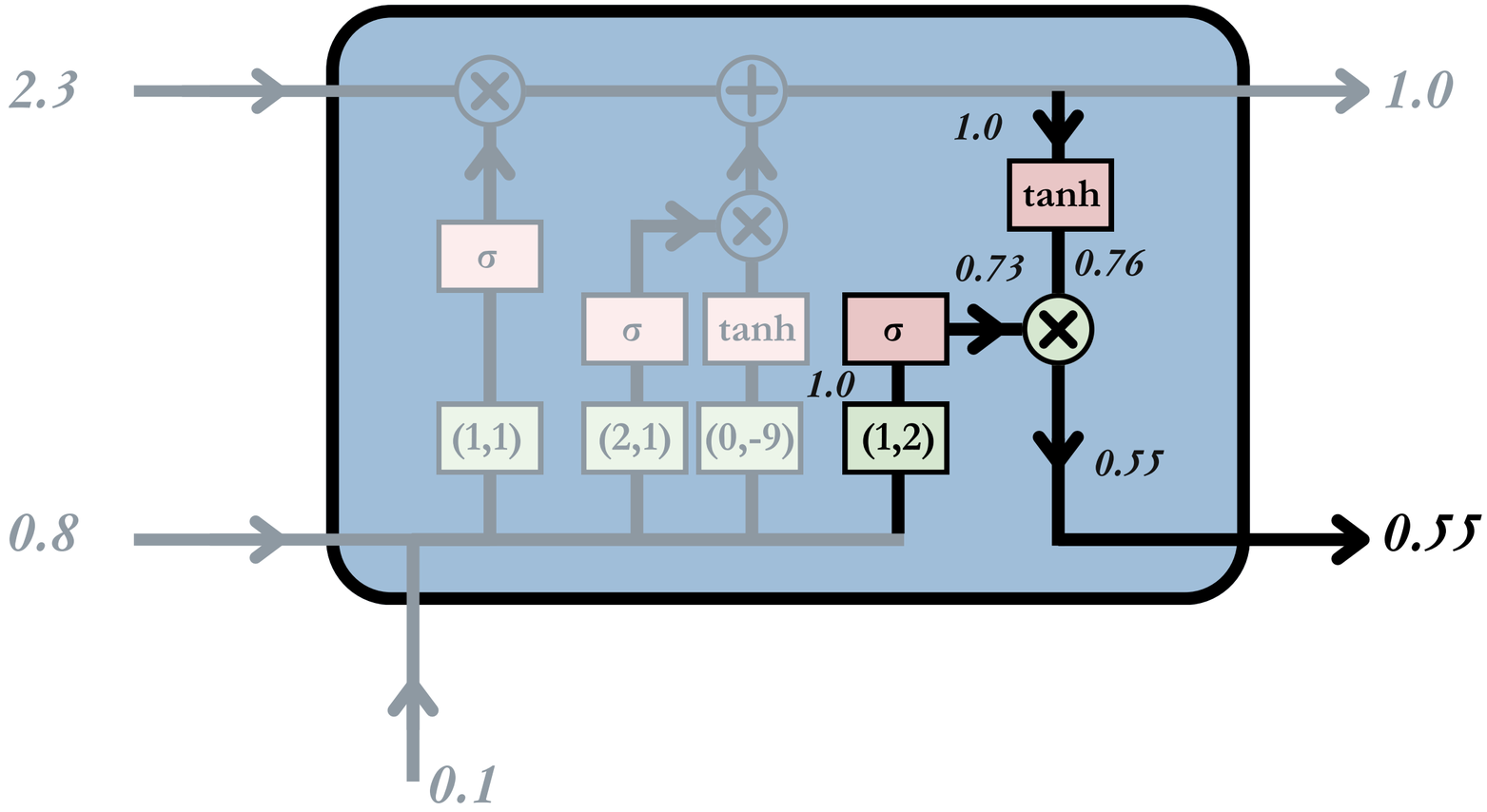


LSTM Example: Output Gate

We consider a simple monodimensional example

- $C_{t-1}=2.3$
- $h_{t-1}=0.8$
- $x_t=0.1$

The 4 matrices are now vectors with two components.



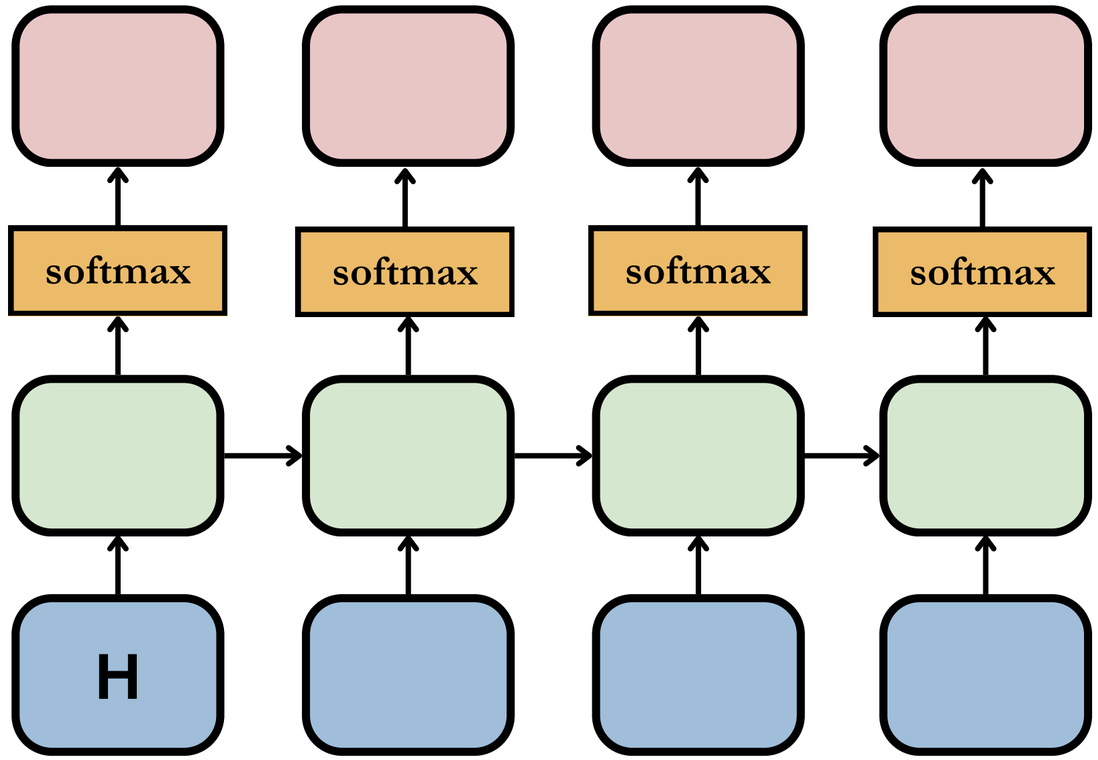
Universität
Konstanz

Text Generation with RNNs

Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

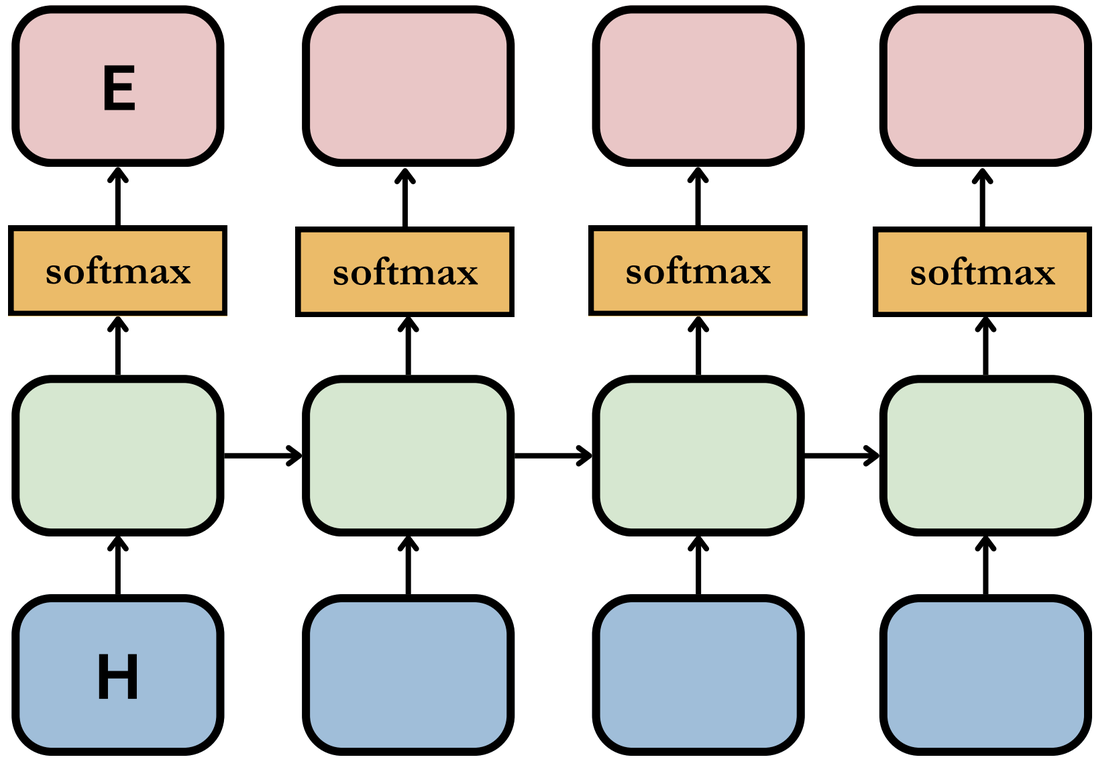
- we input a letter
- the output of the unit is passed through a softmax



Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

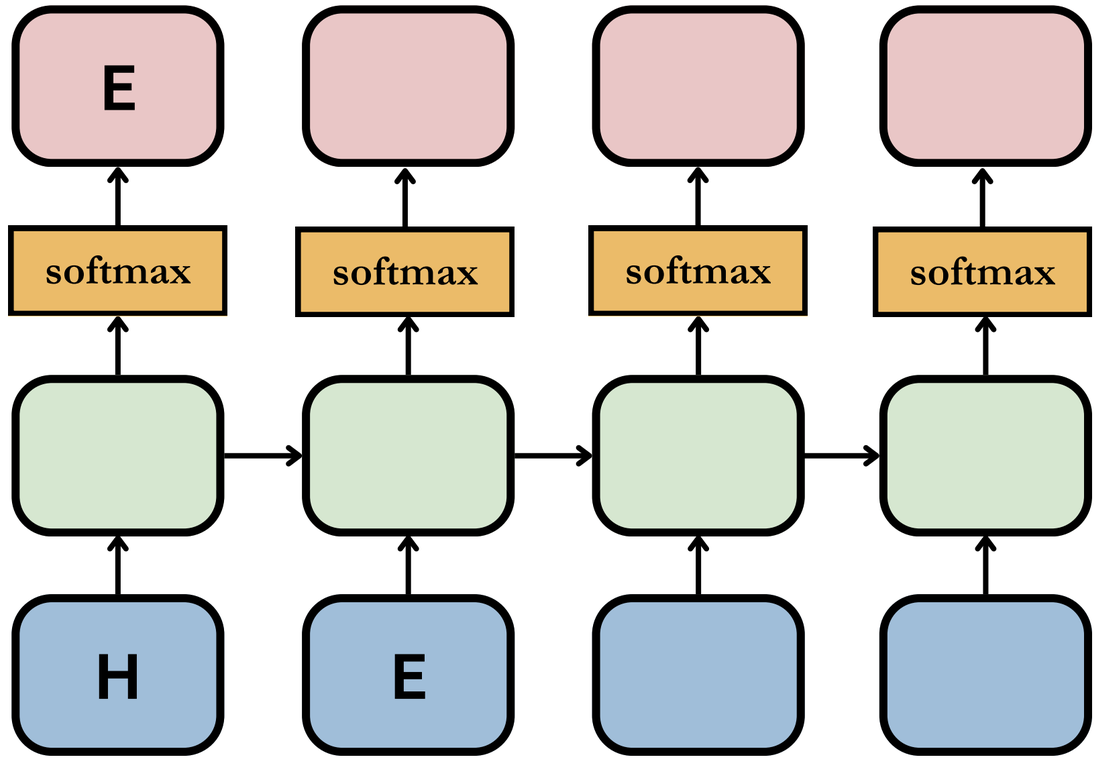
- we input a letter
- the output of the unit is passed through a softmax
- the next character is sampled from the probability distribution



Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

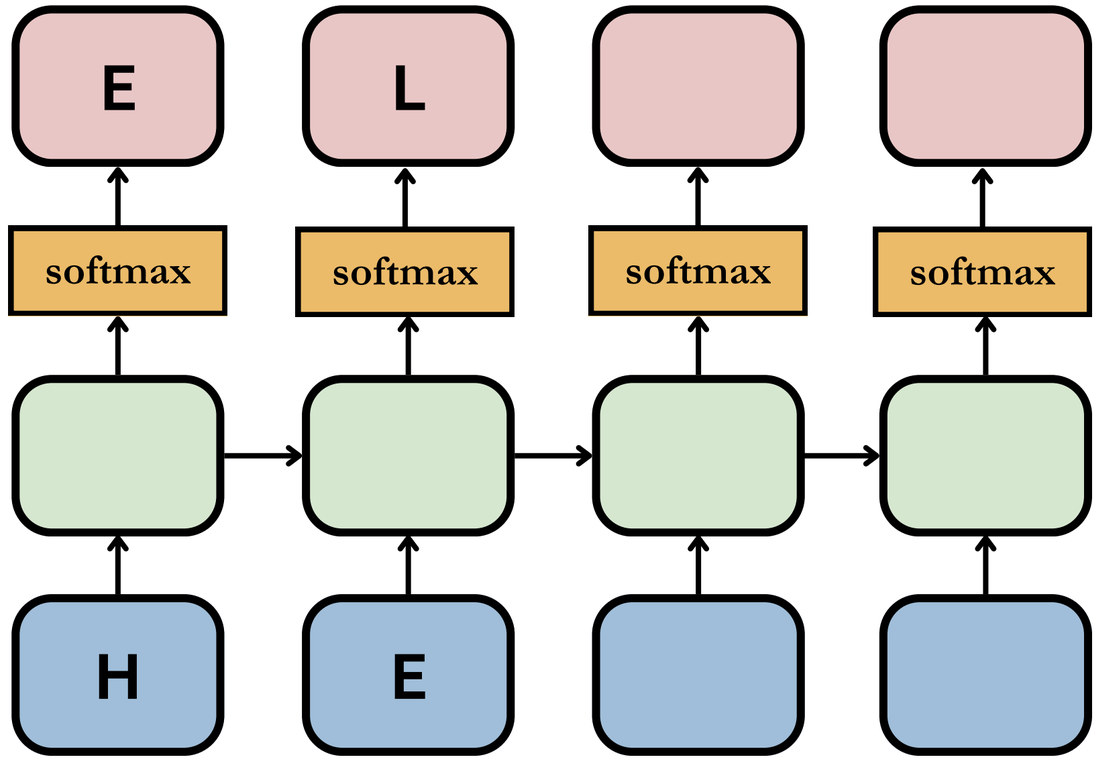
- we input a letter
- the output of the unit is passed through a softmax
- the next character is sampled from the probability distribution
- the output character is used as the next input



Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

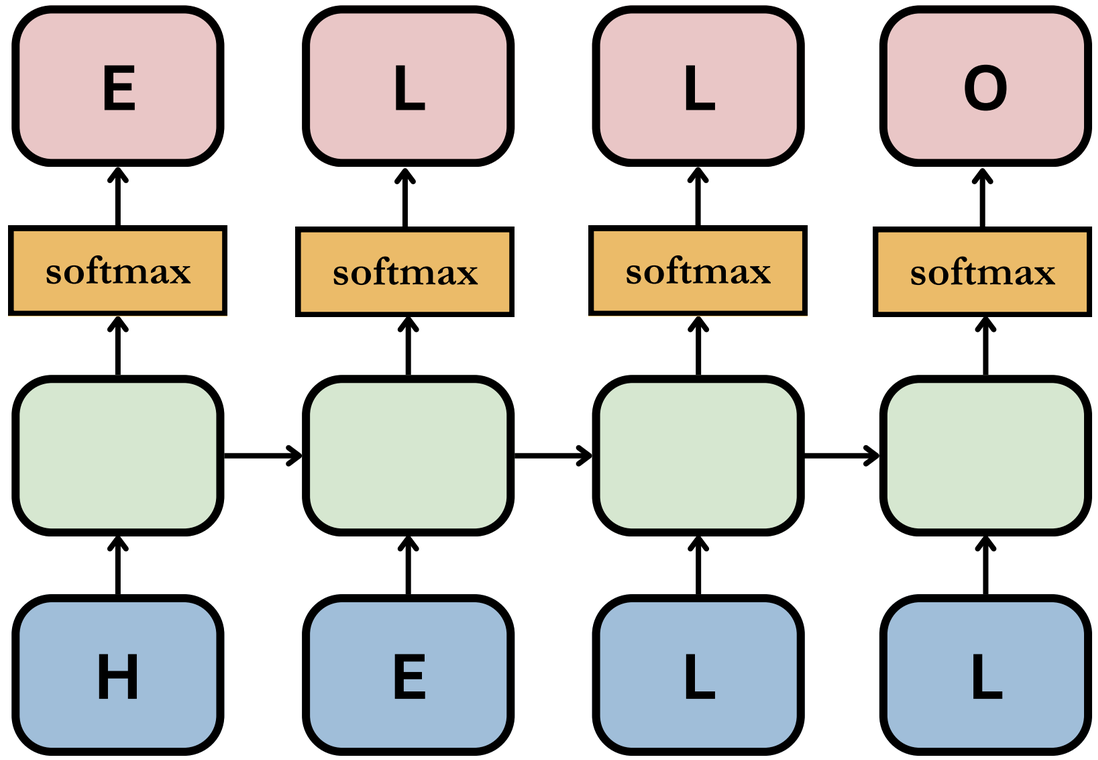
- we input a letter
- the output of the unit is passed through a softmax
- the next character is sampled from the probability distribution
- the output character is used as the next input
- the process is iterated



Text Generation with RNNs

RNNs can be used to generate text by training them to forecast the next character (or word)

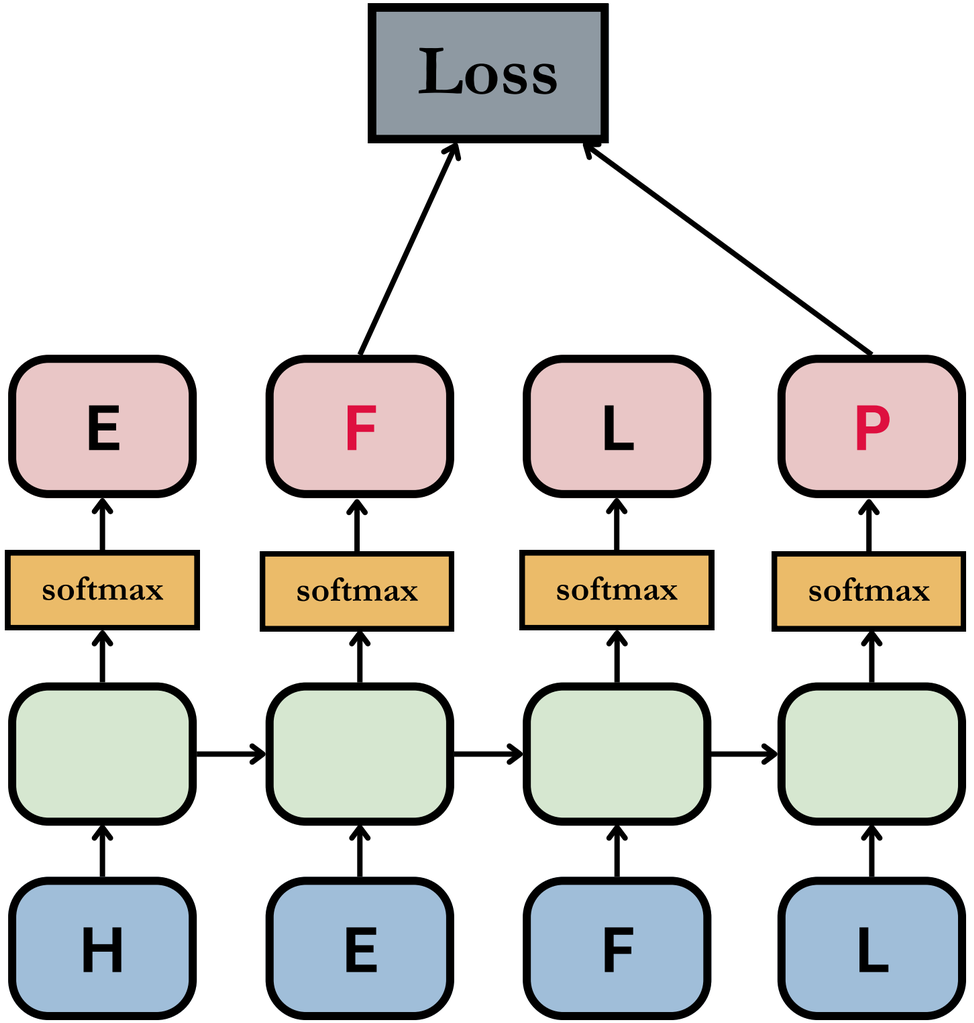
- we input a letter
- the output of the unit is passed through a softmax
- the next character is sampled from the probability distribution
- the output character is used as the next input
- the process is iterated



Loss Function

In this case the loss function is computed summing all the errors over all time steps

- this process can become very demanding for long texts
- when performing the backpropagation, gradients must be propagated up to the beginning of the text
- to solve this problem generally the loss is computed only over the last ~100 elements in the sequence



Example of Text Generation

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Example of Code Generation

```
static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
  return 0;
}

static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Summary

Sequence Data

Sequence data are ubiquitous. Language, music, videos and time series are just few examples. In order to analyze these data we need neural networks that remember past data.

Recurrent Neural Network

We can give neural networks a memory by using feedback loops. Vanilla RNNs are the most simple example of this approach. They are versatile, but suffer from different problems.

Long Short Term Memories

LSTM are more advanced RNNs that store both a short term and a long term memory. This allows to process much longer time sequences.

Text Generation with RNNs

RNNs can be used to generate text by a next character/word prediction framework. The idea is similar to LLMs, but performances are not that good.