Deep Learning

Dr. José Ramón Iglesias
DSP-ASIC BUILDER GROUP
Director Semillero TRIAC
Ingenieria Electronica
Universidad Popular del Cesar

Summary of this lesson

"What people call *AI* is no more than finding answers to questions we know to ask. Real *AI* is answering questions we haven't dreamed of yet"
-Tom Golway

How deep can we dig in AI?

Content of this lesson

- Recurrent Neural Networks (RNNs)
- Long Short Term Memories (LSTMs)
- Convolutional Neural Networks (CNNs)
- Generative Adversarial Networks (GANs)

Datasets

- Datasets used:

- Deep Learning is the recent evolution of Neural Networks
- It covers:
  - Feedforward networks with many hidden layers (deep ☺)
  - New paradigms, like LSTMs in Recurrent Neural Networks, suitable for time series analysis
  - New topological layers, like convolutional and pooling layers, mainly for image processing
  - New architectures as in Generative Adversarial Networks (GANs)
  - ...
- Improvements are mainly due to:
  - Increased computational power for faster calculations, like GPUs
  - Parallel Computation

Recurrent Neural Networks (RNNs)

What are Recurrent Neural Networks?

- **R**ecurrent **N**eural **N**etworks (RNNs) are a family of neural networks suitable for processing of sequential data
- RNNs include auto and backward connections
- RNNs are used for all sorts of tasks:
  - Language modeling / Text generation
  - Text classification
  - Neural machine translation
  - Image captioning
  - Speech to text
  - Numerical time series data, e.g. sensor data
  - Time series analysis
  - …

Why do we need RNNs for Sequential Data?

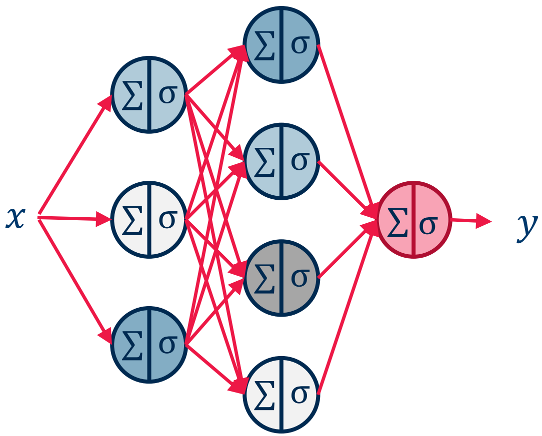
- **Goal**: Translation from German to English

  "Ich mag Schokolade"
  => *"I like chocolate"*

- Option one: Use feed forward network to translate word by word

- But what happens with this question?

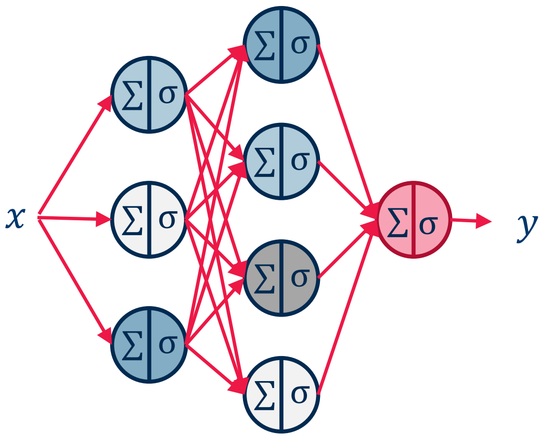
  "Mag ich Schokolade?"
  => *"Do I like chocolate?"*



Input x	Output y
Ich	I
mag	like
Schokolade	chocolate

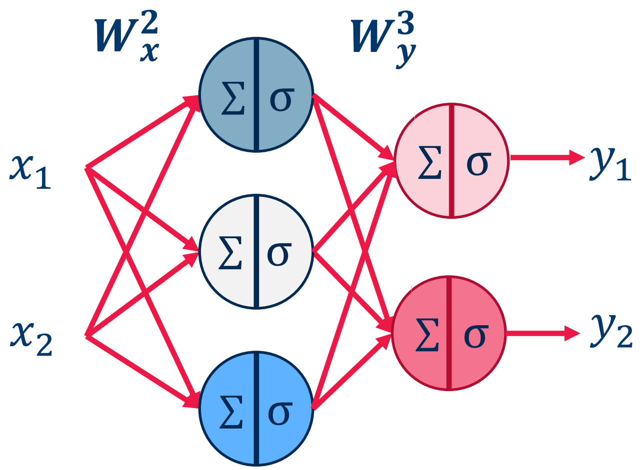
- Problems with FFNN:
  - Each time step is completely independent
  - For translations we need context
    - More general: we need a network that remembers inputs from the past
  - Handle variable sequence length

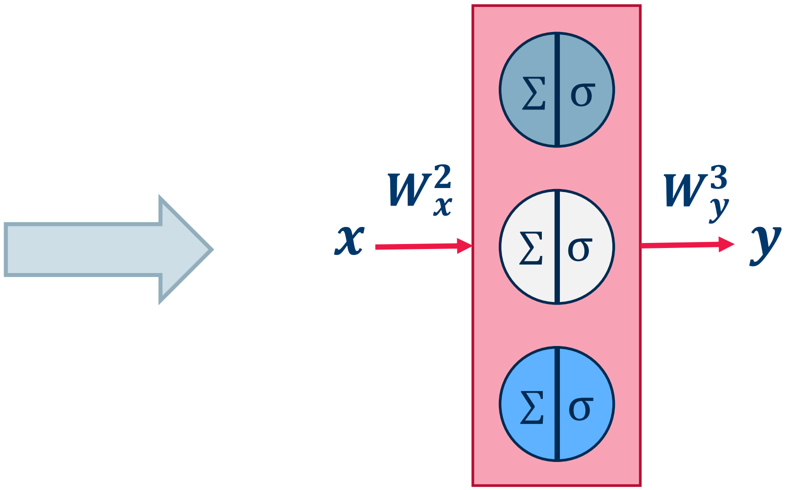
- **Solution**: Recurrent Neural Networks



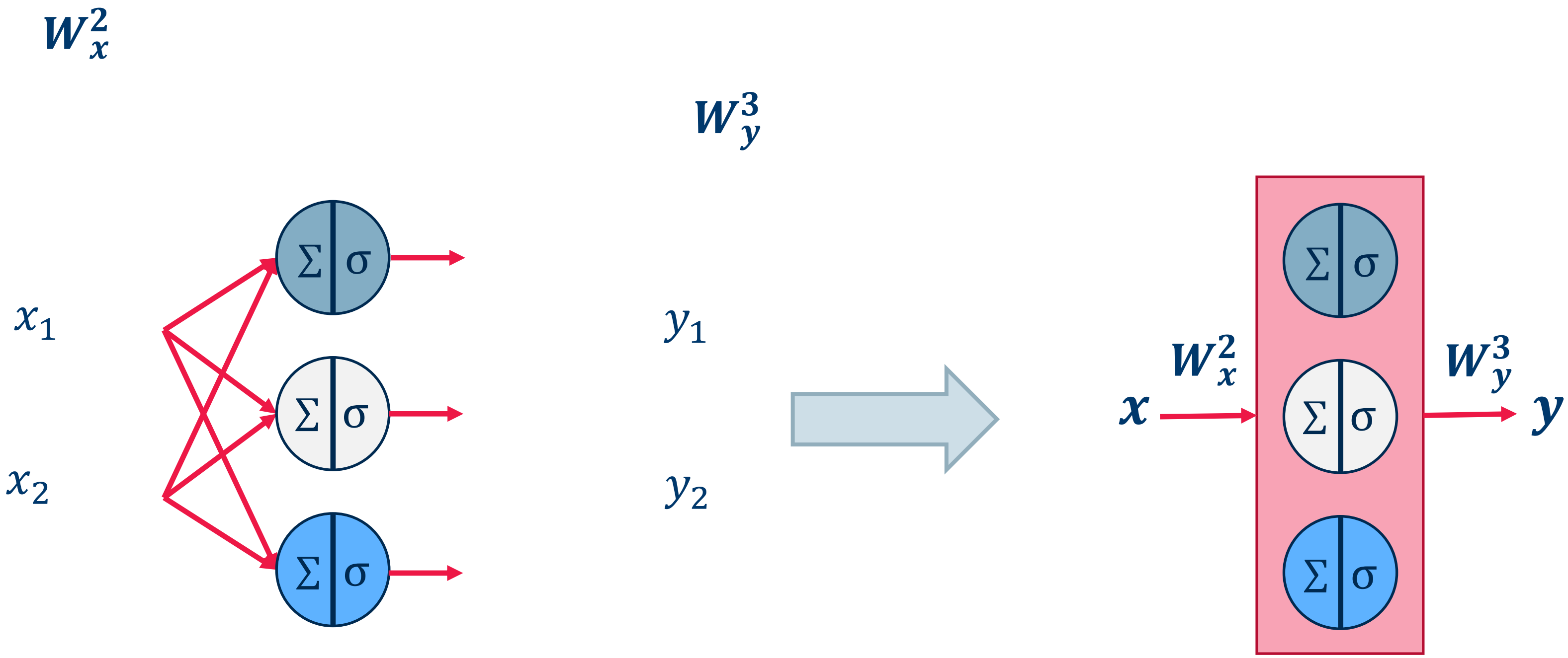
Input x	Output y
Mag	Like
Ich	I
Schokolade	chocolate

From Feed Forward to Recurrent Neural Networks

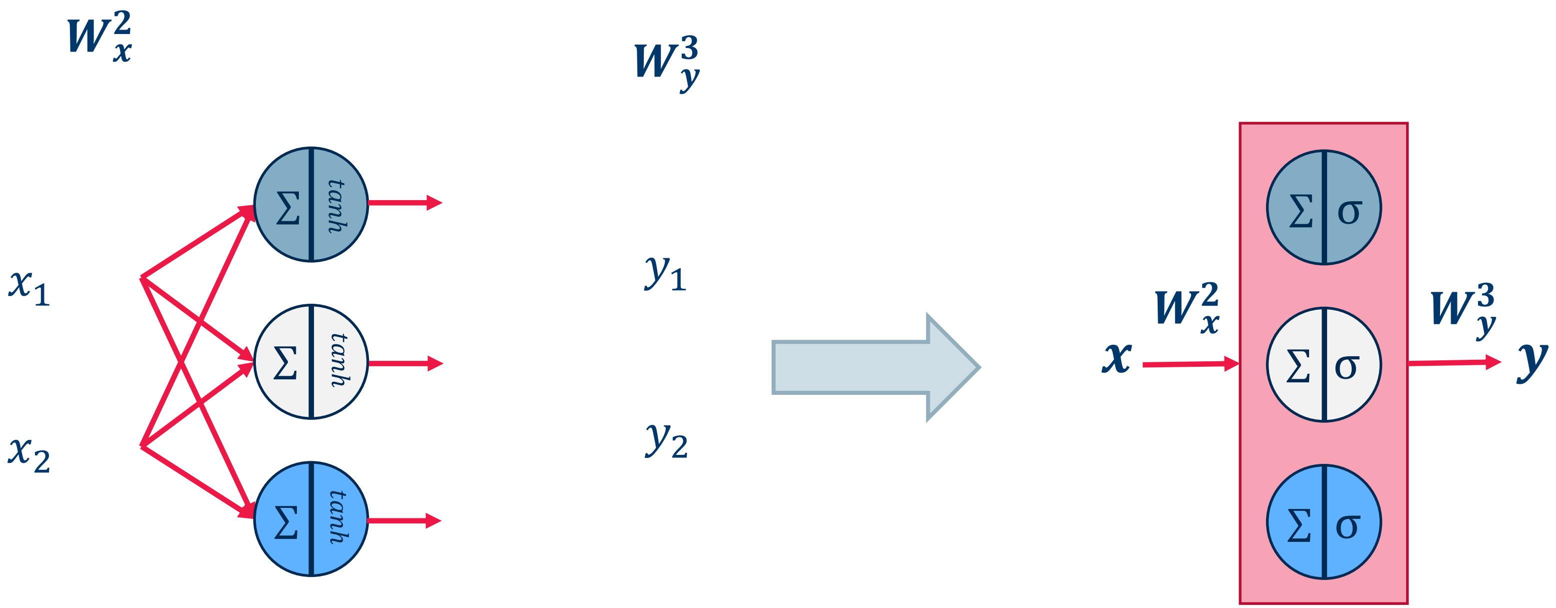




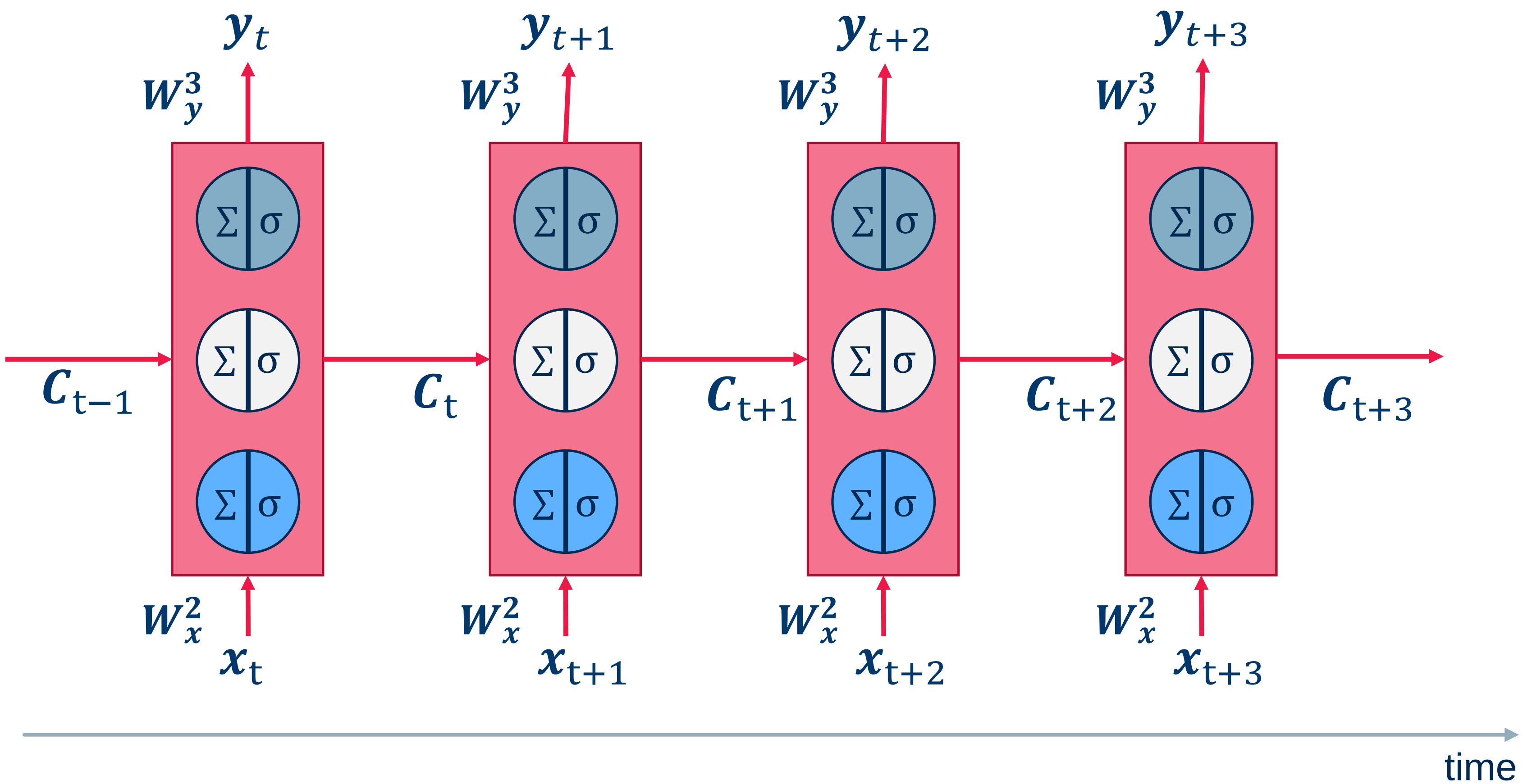
From Feed Forward to Recurrent Neural Networks

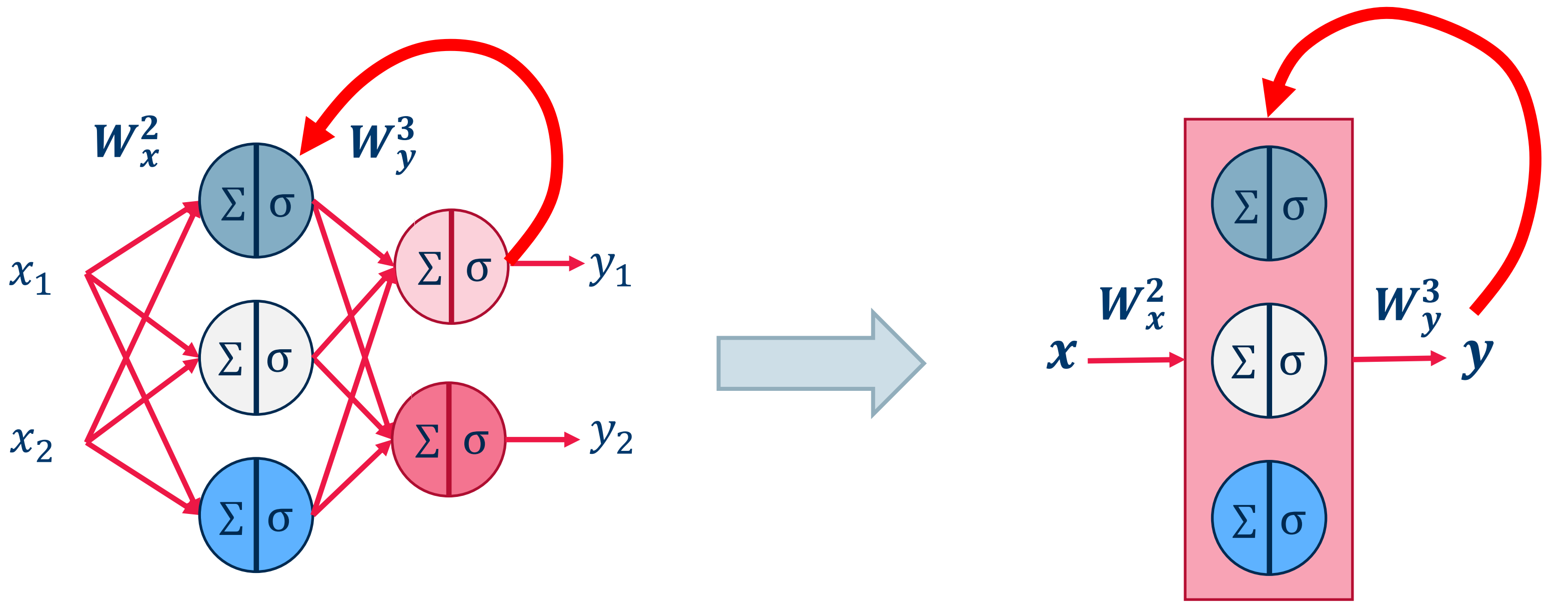


From Feed Forward to Recurrent Neural Networks

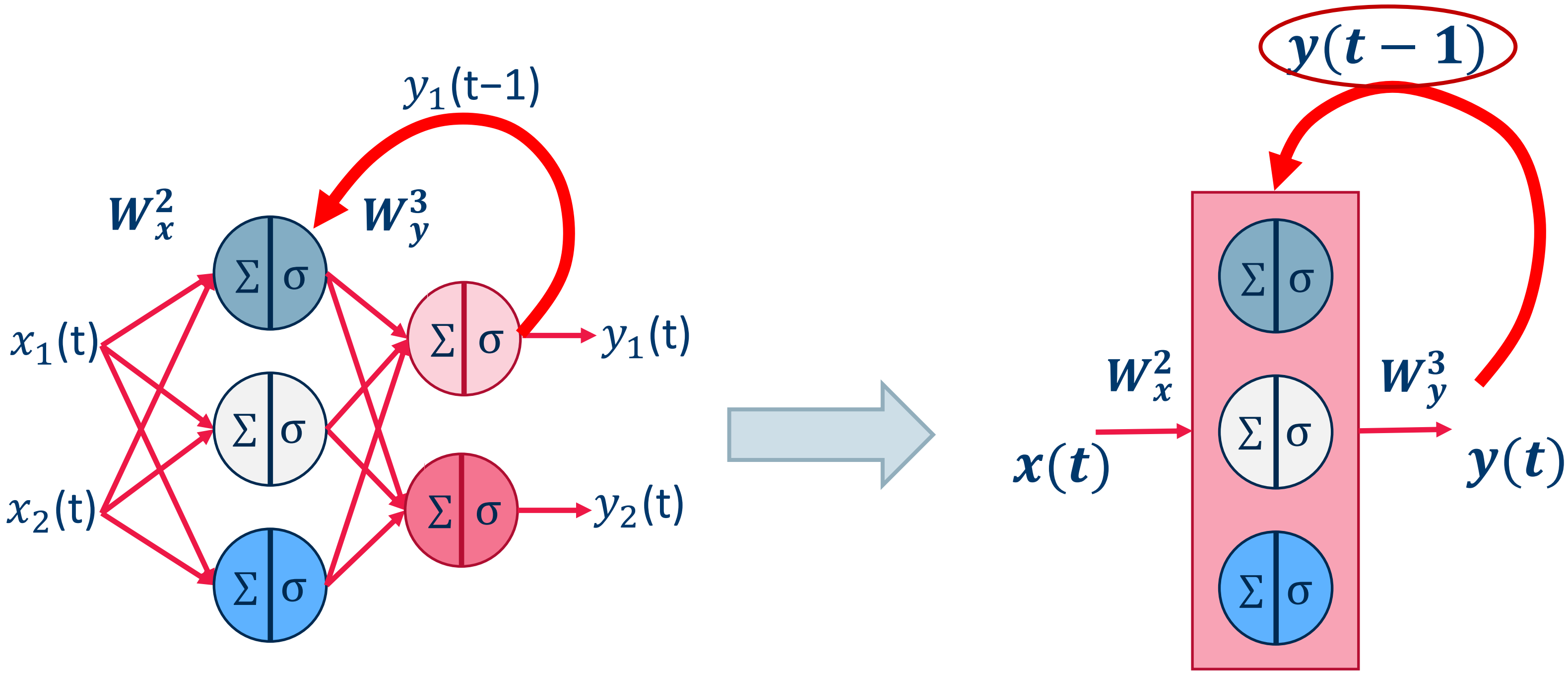


Unrolling of a RNN over time



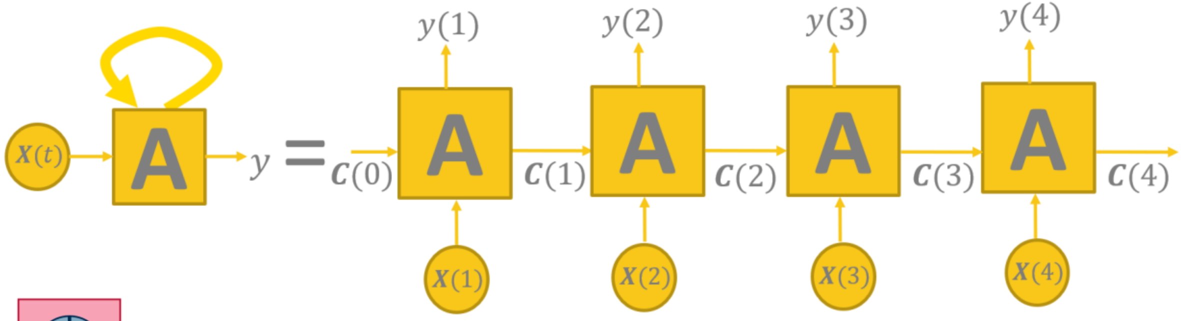


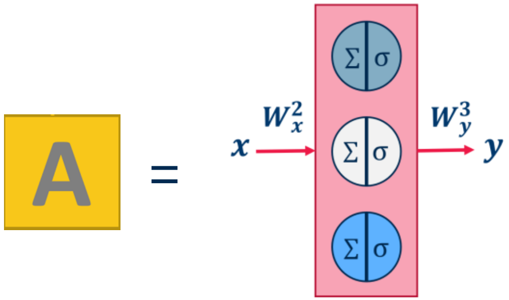
- A Recurrent Neural Network is a FFNN with auto and/or backward connections
- Recurrent connections introduce the concept of **time** in FFNNs



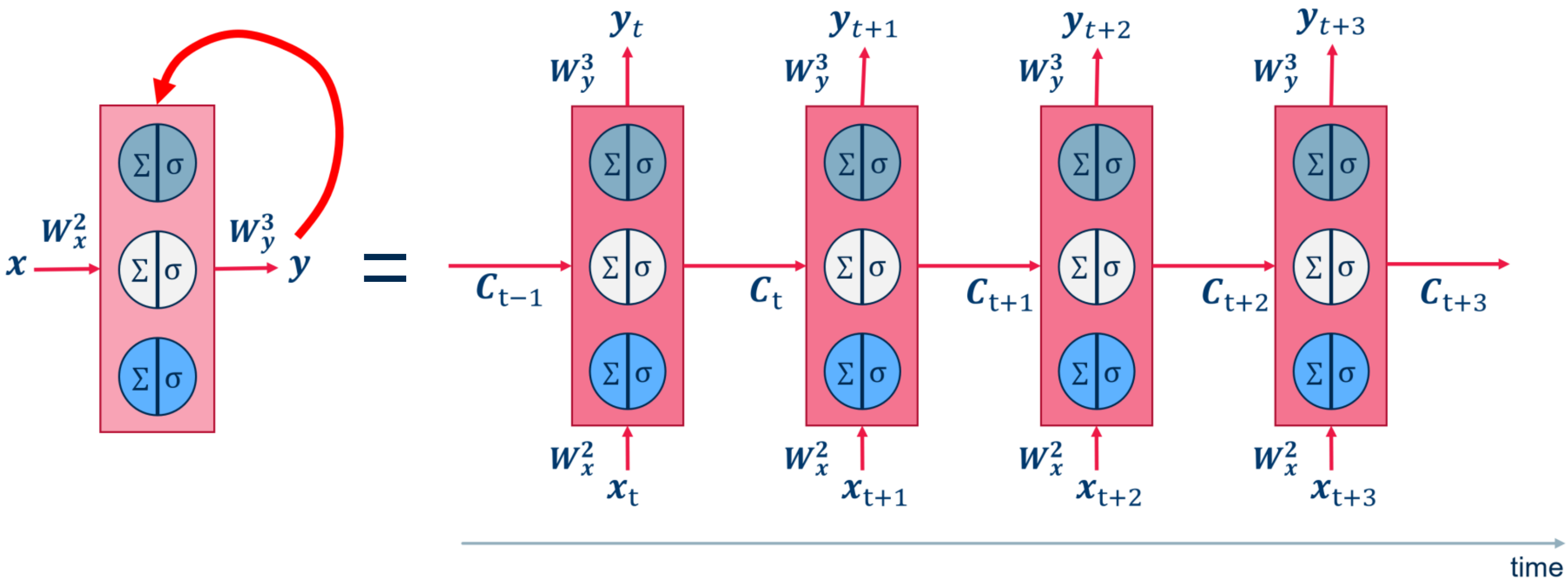
- A Recurrent Neural Network is a FFNN with auto and/or backward connections
- Recurrent connections introduce the concept of **time** in FFNNs

- At every time t, FFNN A has two inputs:
  - **x**(t)
  - some shape of ***y****(t-1)* -> state of network A: ***C****(t-1)*
- The recurrent network can then be **unrolled** over time around **A**





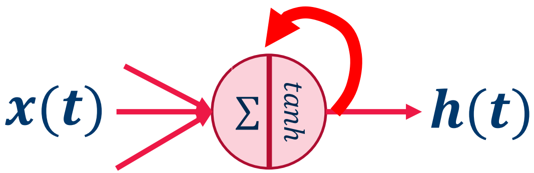
The unrolled version of the original network in *m* intermediate steps becomes a FFNN and can be trained with BackPropagation: **Back-Propagation Through Time (BPTT).**



- Neural network architectures with recurring connections on some units are named Recurrent Neural Networks (RNNs).
- Adding a recurrent connection to one unit might store information about past inputs in the evolving status of the unit.
- An easy trick to represent the recurrent network is to unroll it into *m* copies of the feedforward internal block “A”, each with their set of static weight matrix **W.** Each copy of “A” receives inputs **X**(t) and **C**(t-1) and produces output **y**(t).
- A modified version of the Back-Propagation algorithm is used to train RNNs: Back-Propagation Through Time (BPTT).

Long Short Term Memory

The simplest possible recurrent unit is a single layer with an auto-connection.



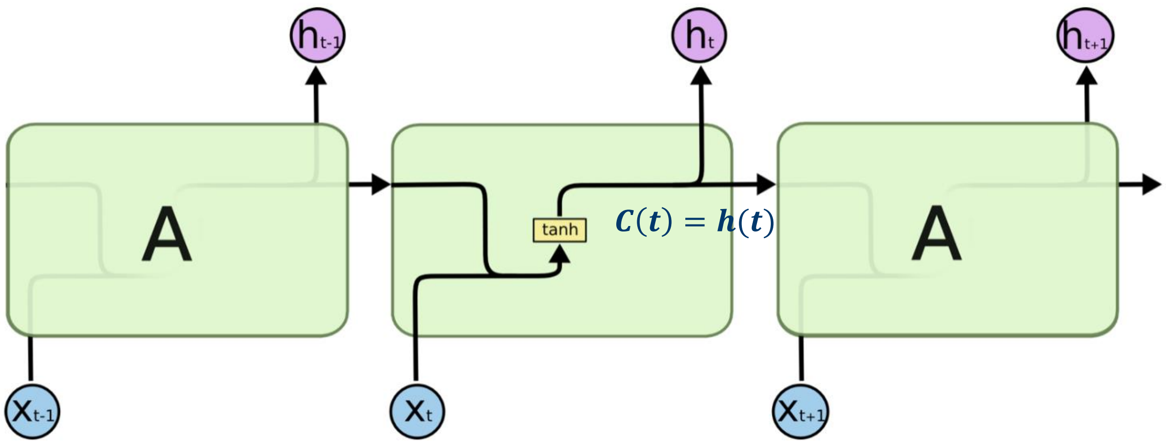


Image Source: Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

The “memory” of simple RNNs is sometimes too limited to be useful:

- *“Cars drive on the ____” (road)*
- *“I love the beach.*
  My favorite sound is the crashing of the ____“ (cars? glass? waves?)
- Sometimes we need to go back deeper in time

LSTM = Long Short Term Memory

Special type of unit with three gates

- Forget gate
- Input gate
- Output gate

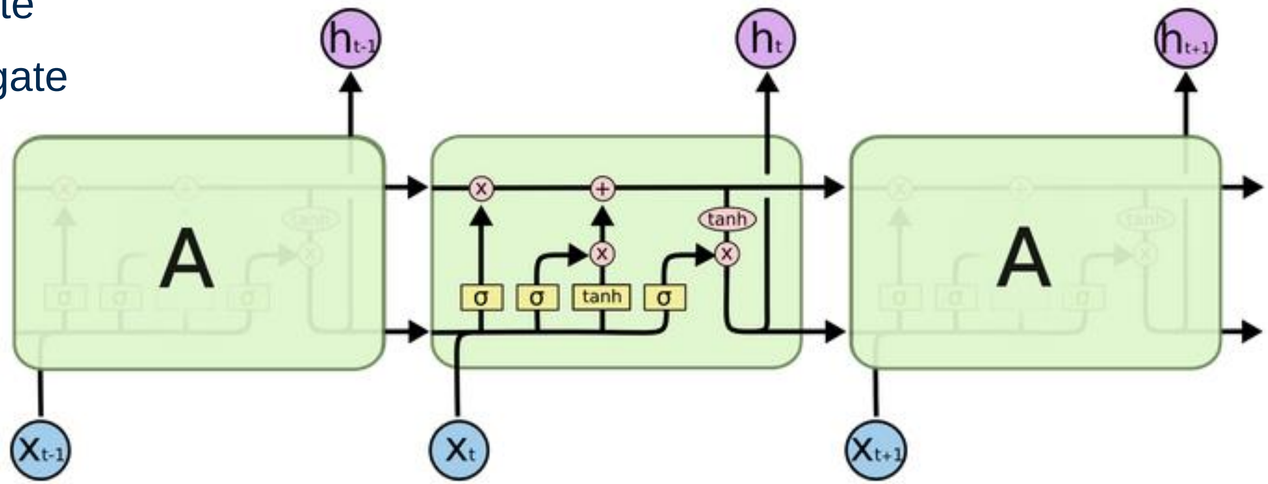
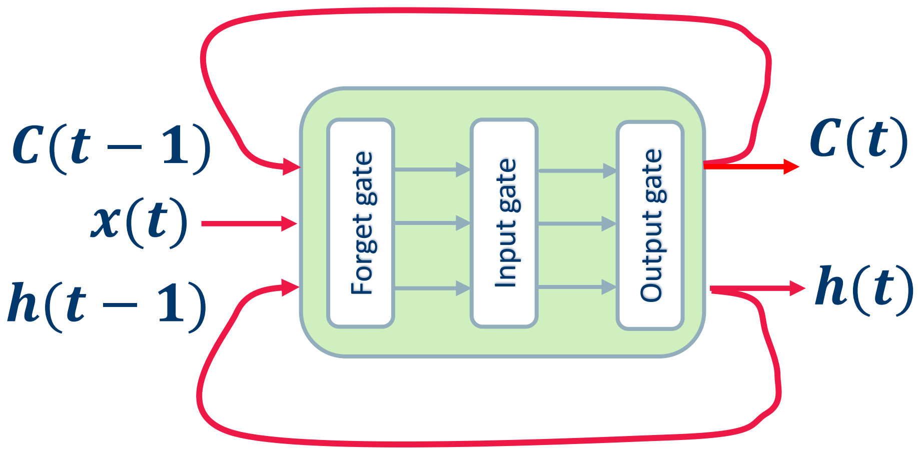


Image Source: Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

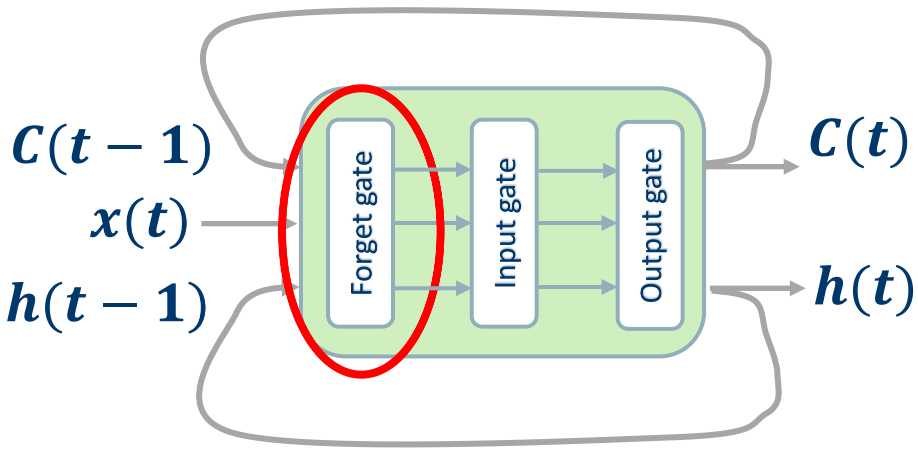
This is an engineered type of unit with three gates:

- Forget gate
- Input gate
- Output gate

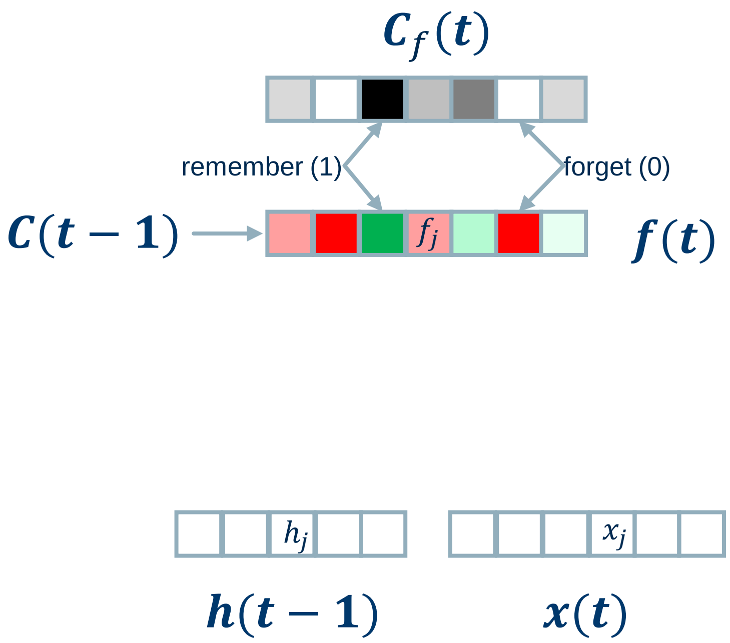


This is an engineered type of unit with three gates:

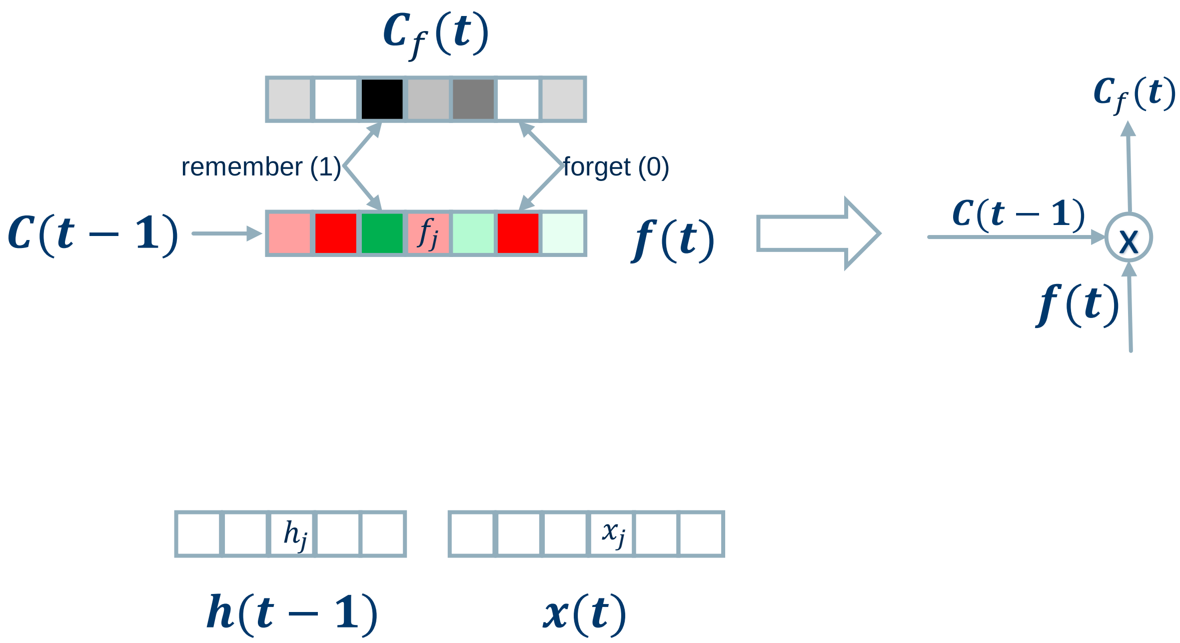
- Forget gate
- Input gate
- Output gate



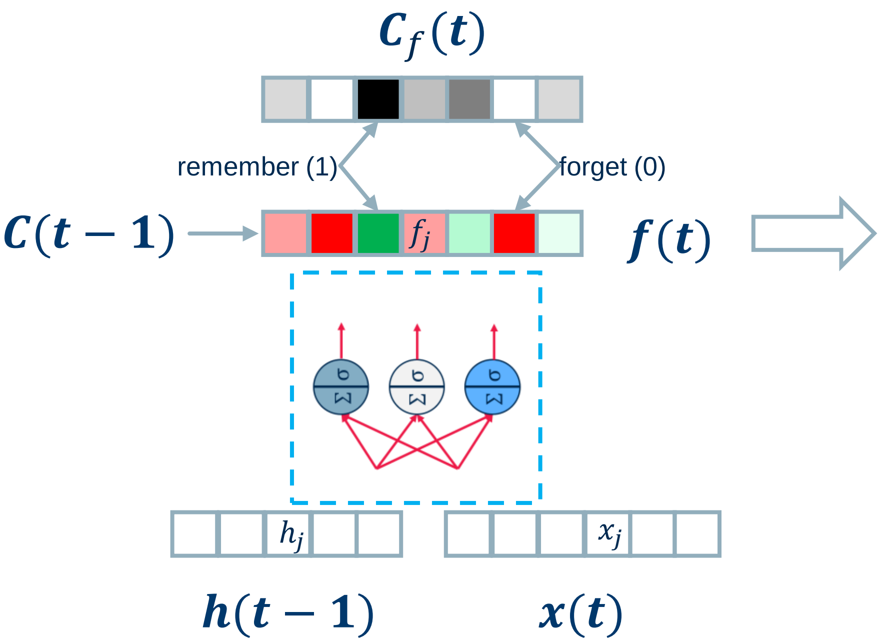
- ***Forget Gate*** is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

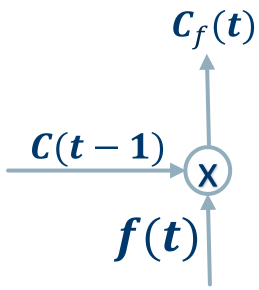


- ***Forget Gate*** is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

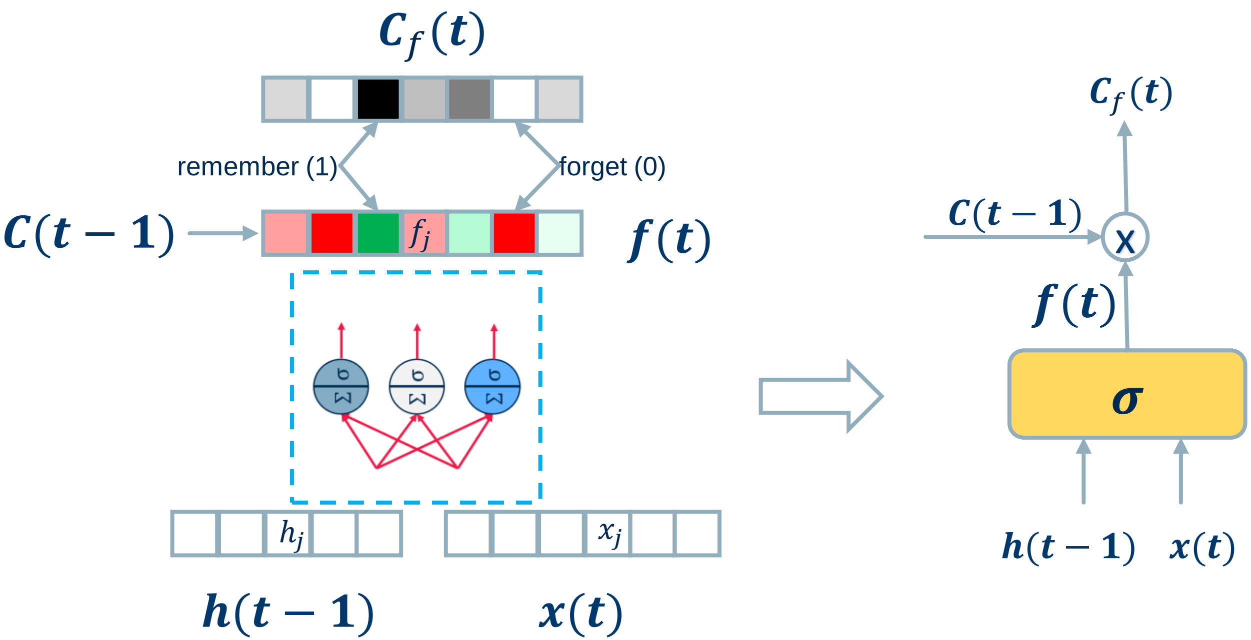


- ***Forget Gate*** is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



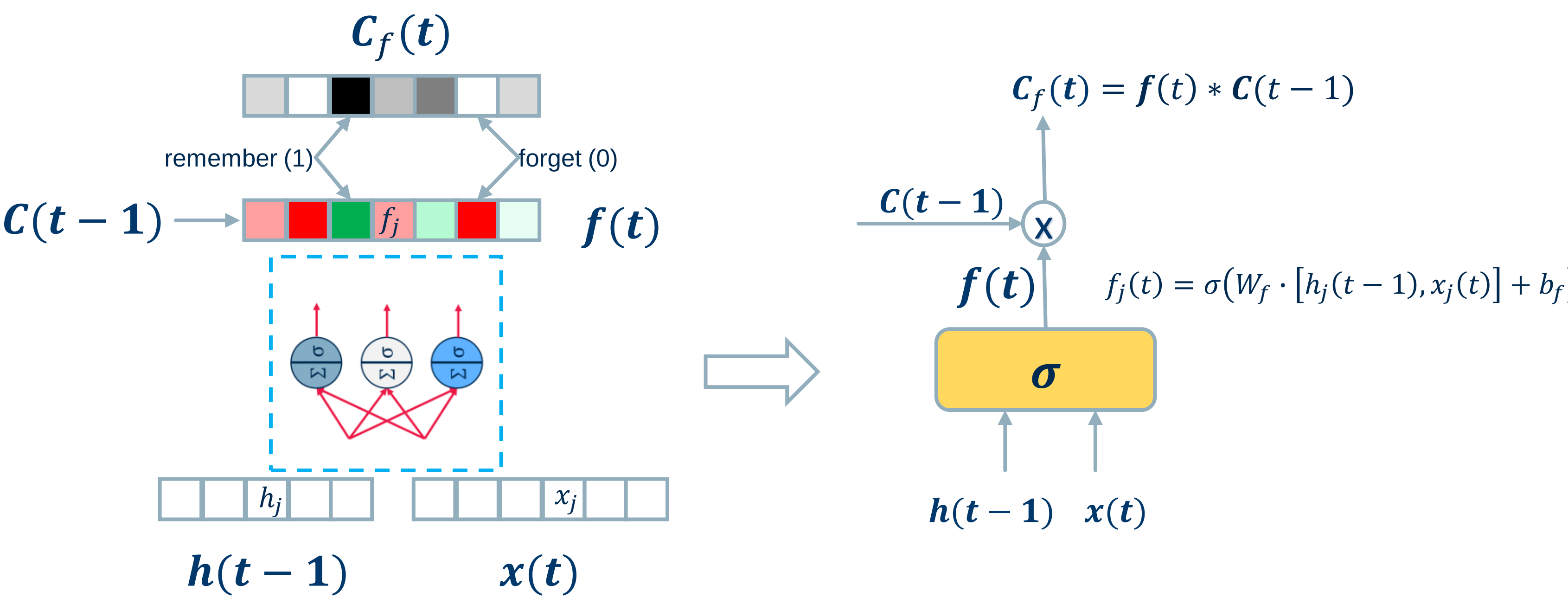


- ***Forget Gate*** is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM: Forget Gate

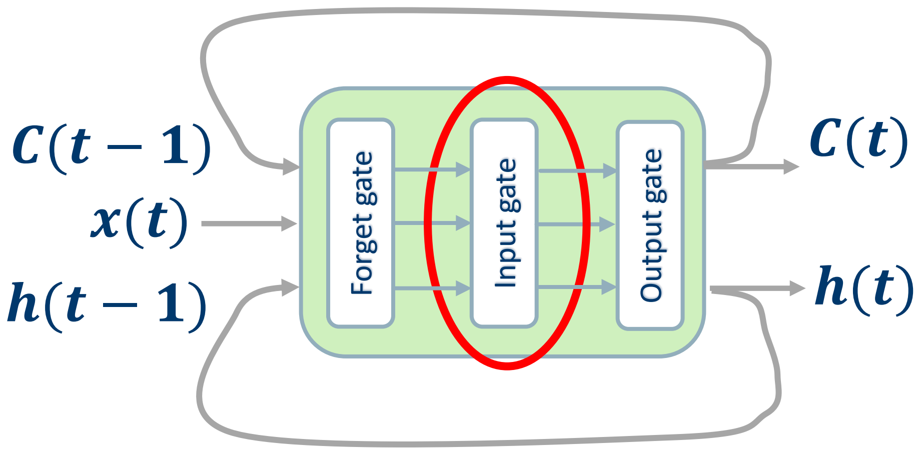
- ***Forget Gate*** is trained to forget parts of the cell state.
- At time t, the forget gate decides which item of $\boldsymbol{C}(t-1)$ to keep (and how much of it) in $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM = Input Gate

This is an engineered type of unit with three gates:

- Forget gate
- Input gate
- Output gate



- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

$\boldsymbol{C}_i(\boldsymbol{t})$

$\widetilde{\boldsymbol{C}}(t)$ → $\boldsymbol{i}(\boldsymbol{t})$

1

Create new cell state candidate

2

Filter cell state candidate

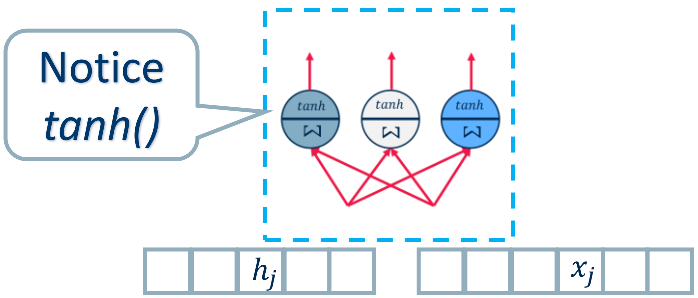
$\boldsymbol{h}(\boldsymbol{t}-\mathbf{1})$ $\boldsymbol{x}(\boldsymbol{t})$ $\boldsymbol{h}(\boldsymbol{t}-\mathbf{1})$ $\boldsymbol{x}(\boldsymbol{t})$

- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

$\boldsymbol{C}_i(\boldsymbol{t})$

$\widetilde{\boldsymbol{C}}(t)$

$\boldsymbol{i}(\boldsymbol{t})$

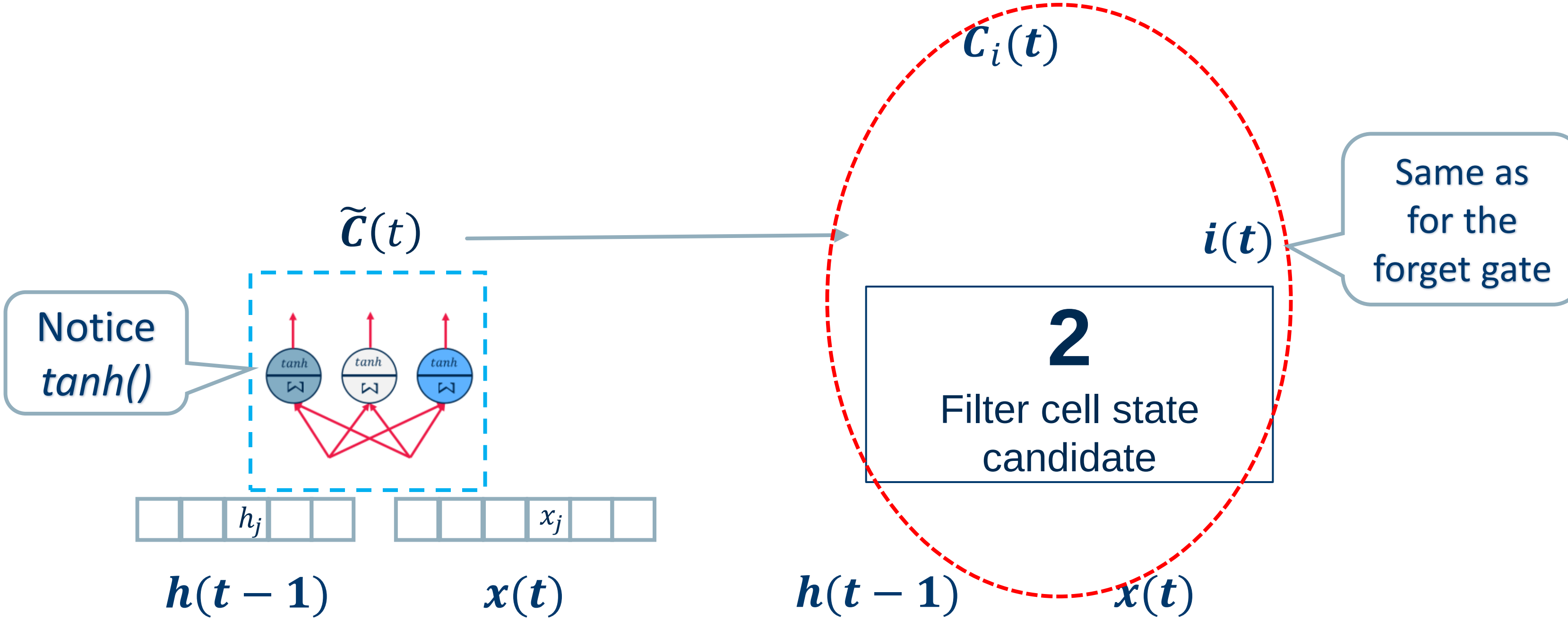


2

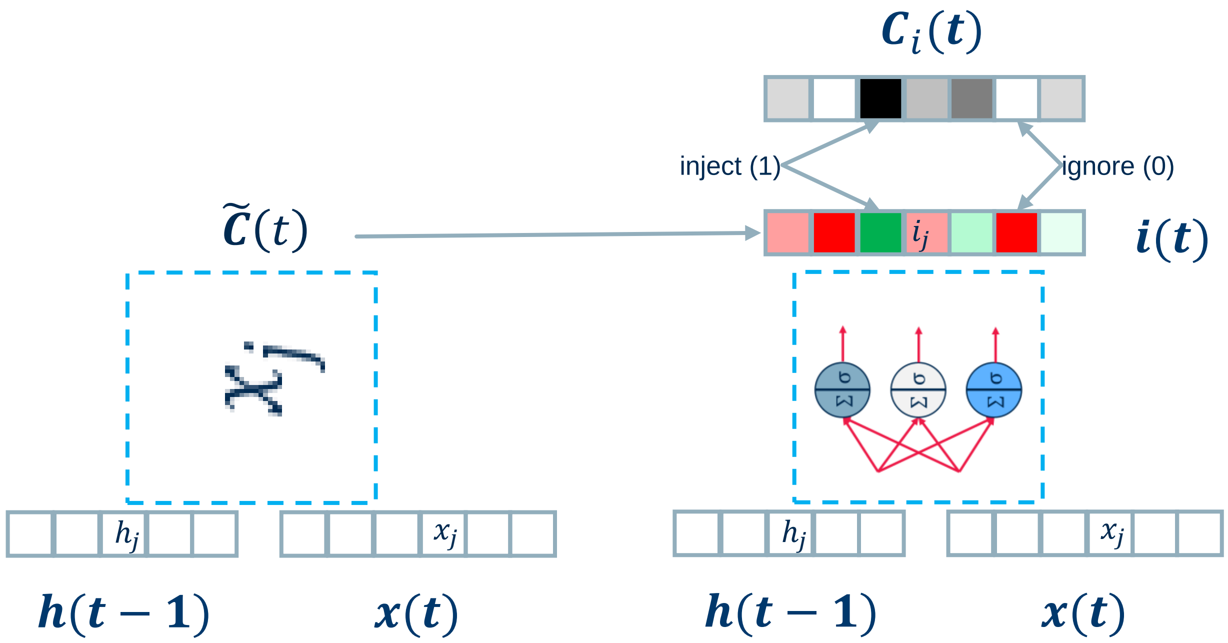
Filter cell state candidate

$\boldsymbol{h}(\boldsymbol{t}-\mathbf{1})$ $\boldsymbol{x}(\boldsymbol{t})$ $\boldsymbol{h}(\boldsymbol{t}-\mathbf{1})$ $\boldsymbol{x}(\boldsymbol{t})$

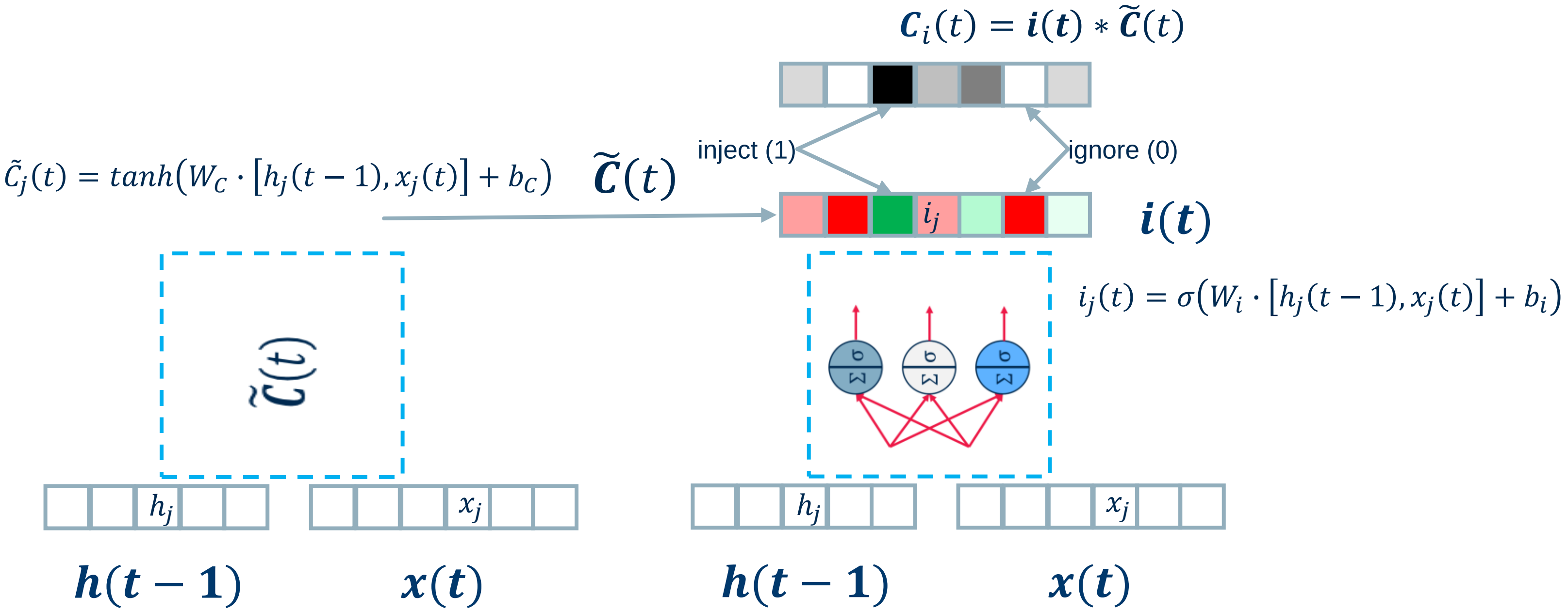
- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

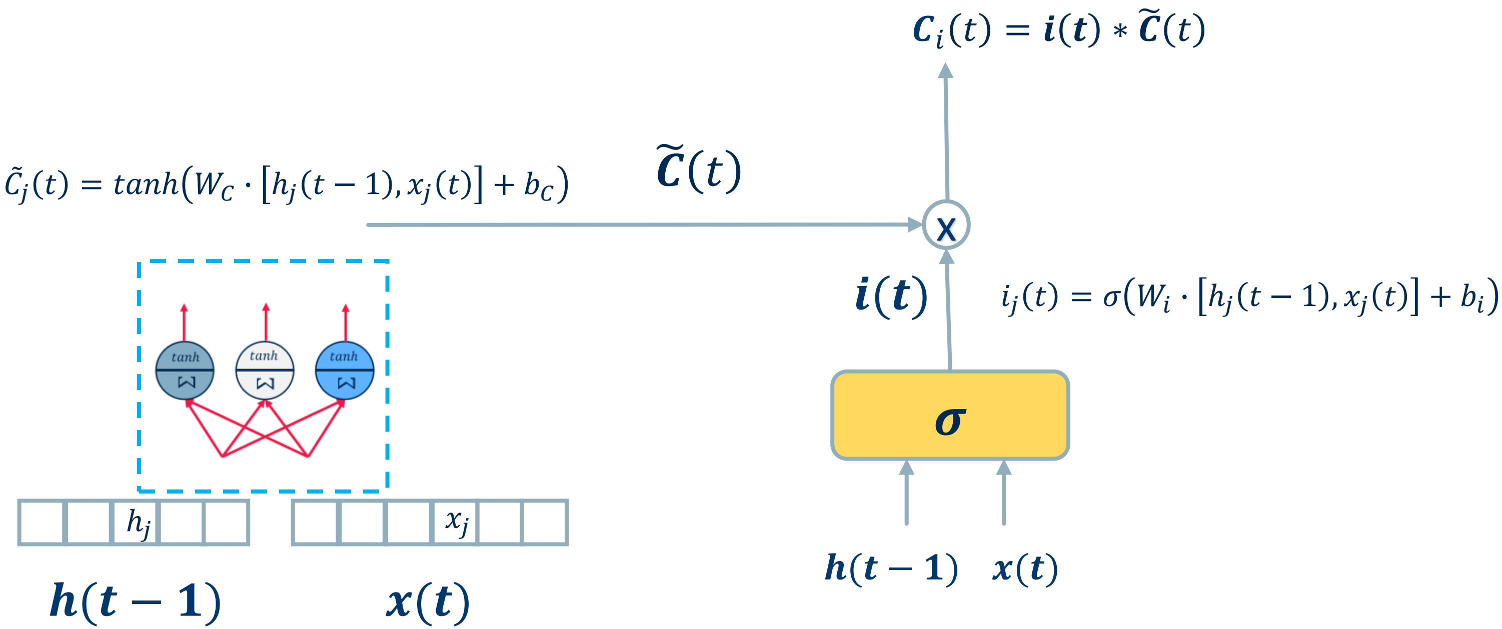


- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

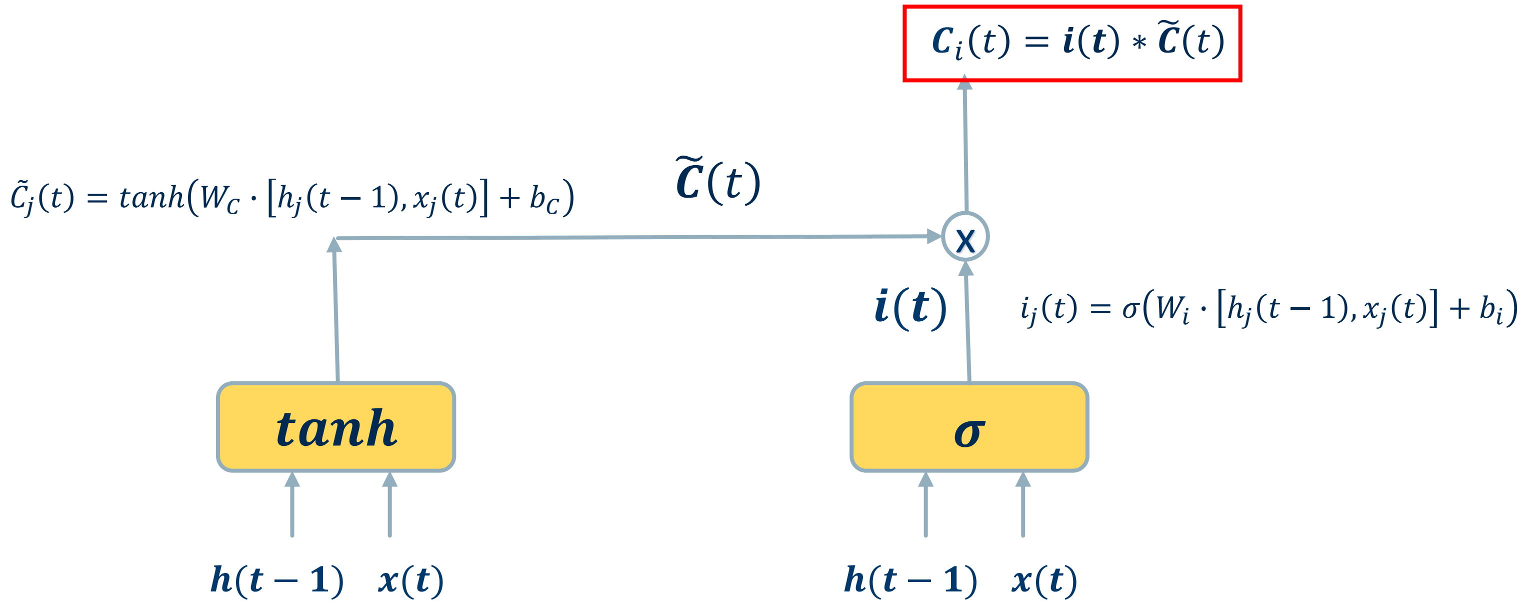


LSTM: Input Gate

- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



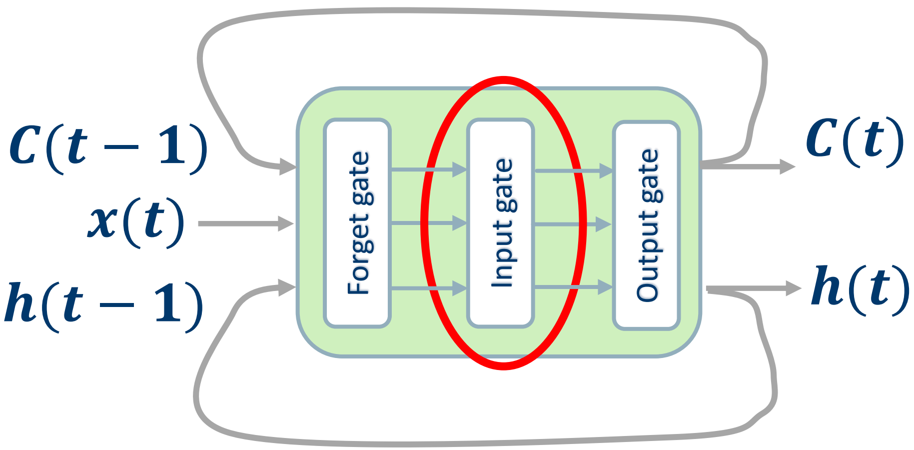
- ***Input Gate*** is trained to inject significant parts of the current input into the cell state.
- At time t, the input gate decides which item of $\boldsymbol{x}(t)$ to inject (and how much of it) into $\boldsymbol{C}(t)$, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM = Input Gate

This is an engineered type of unit with three gates:

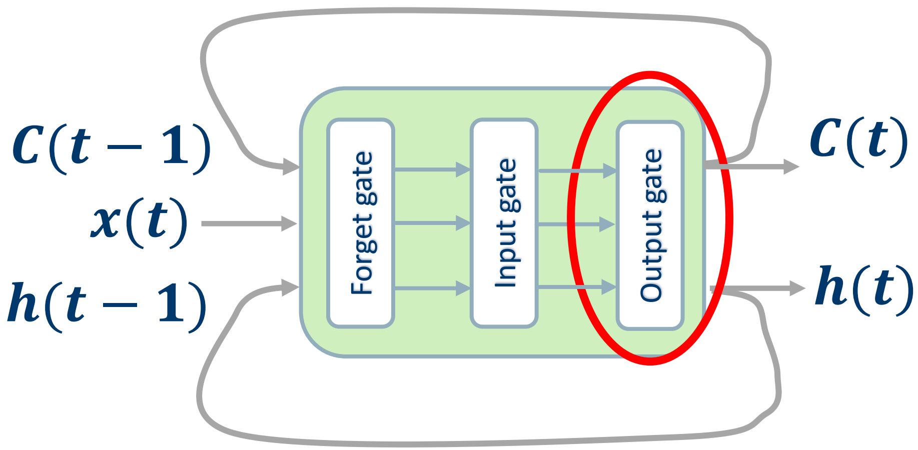
- Forget gate
- Input gate
- Output gate



$$\boldsymbol{C}(t) = \boldsymbol{C}_f(t) + \boldsymbol{C}_i(t) = \boldsymbol{f}(t) * \boldsymbol{C}(t-1) + \boldsymbol{i}(t) * \widetilde{\boldsymbol{C}}(t)$$

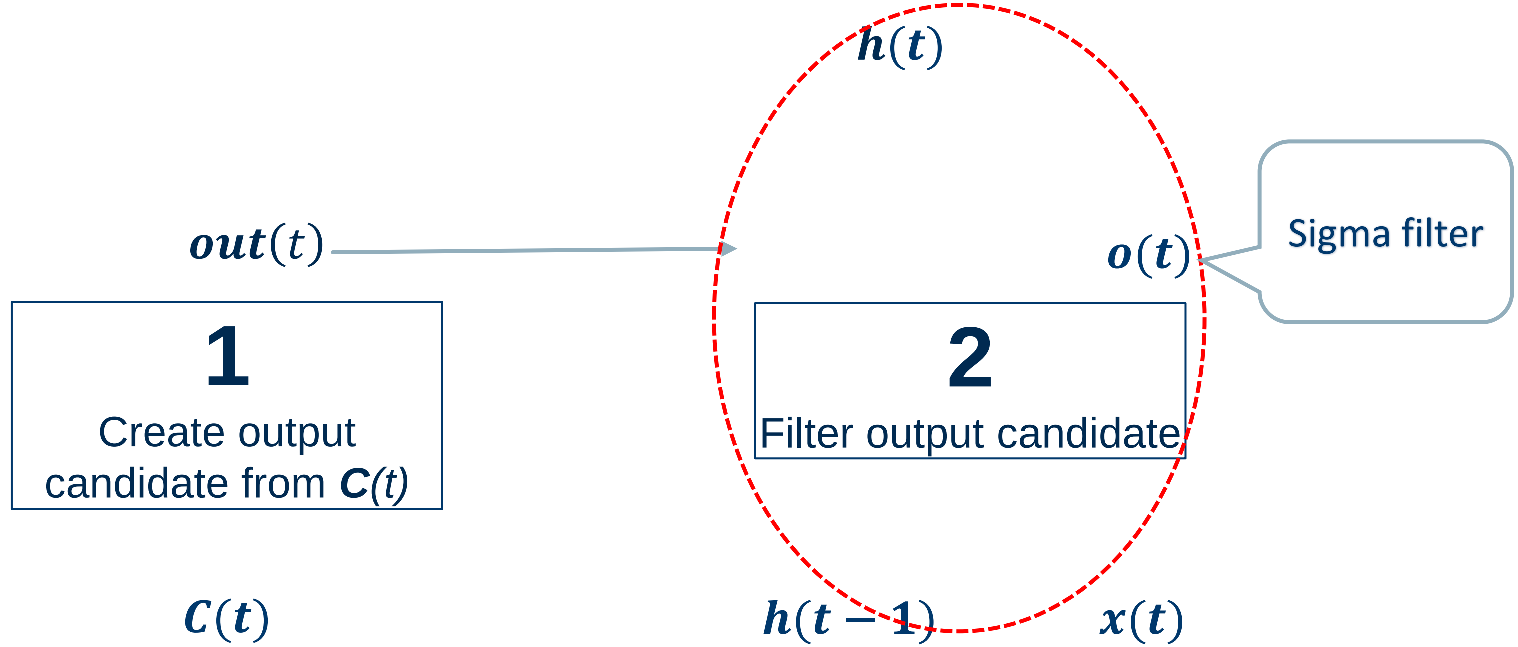
This is an engineered type of unit with three gates:

- Forget gate
- Input gate
- Output gate

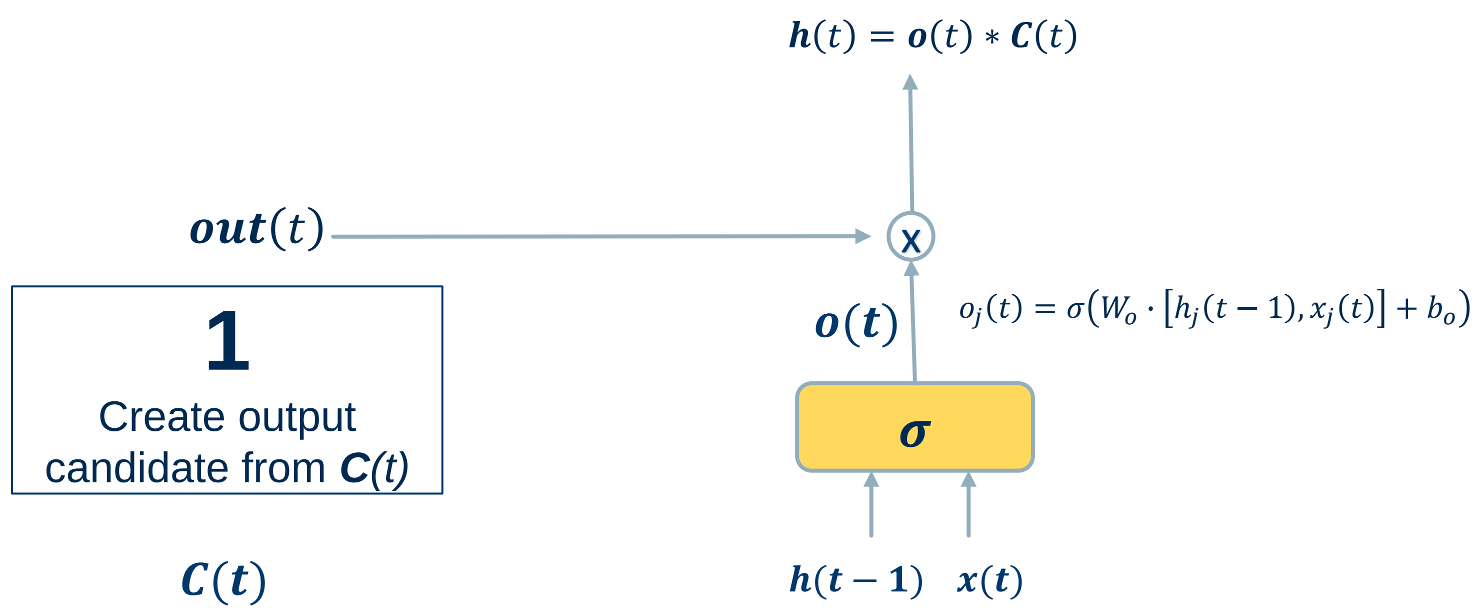


LSTM: Output Gate – input inject into status

- ***Output Gate*** is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

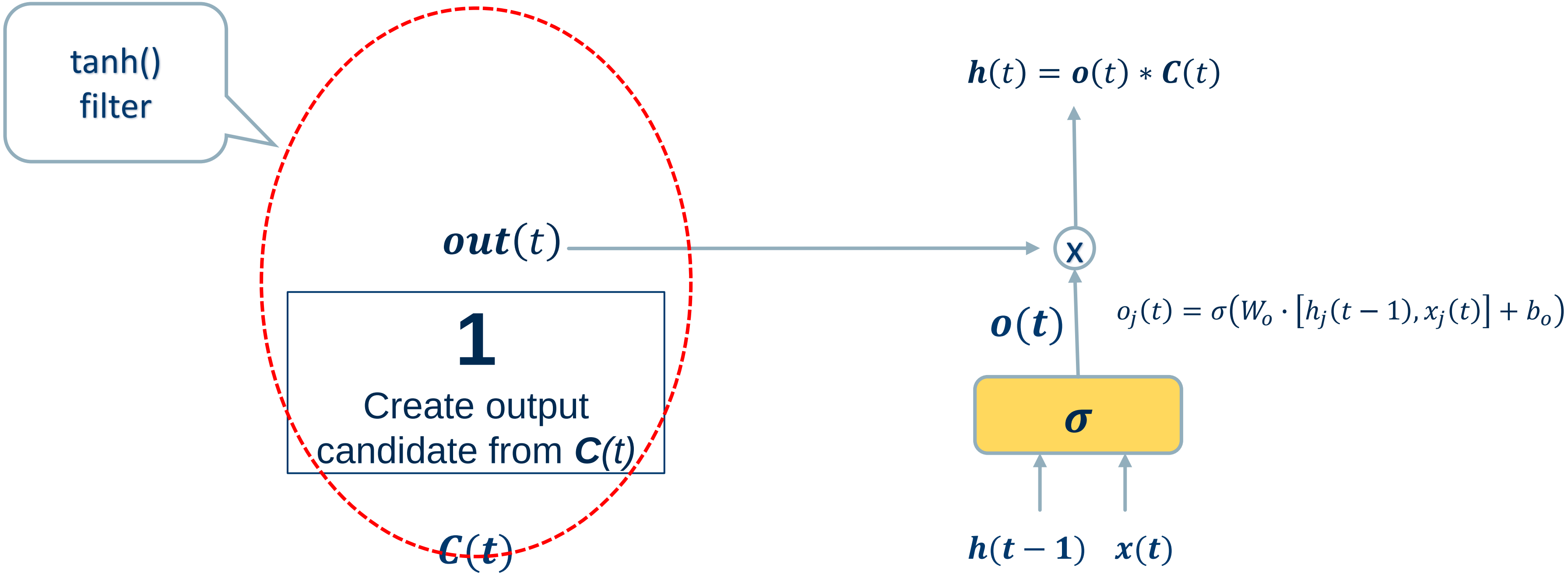


- ***Output Gate*** is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.

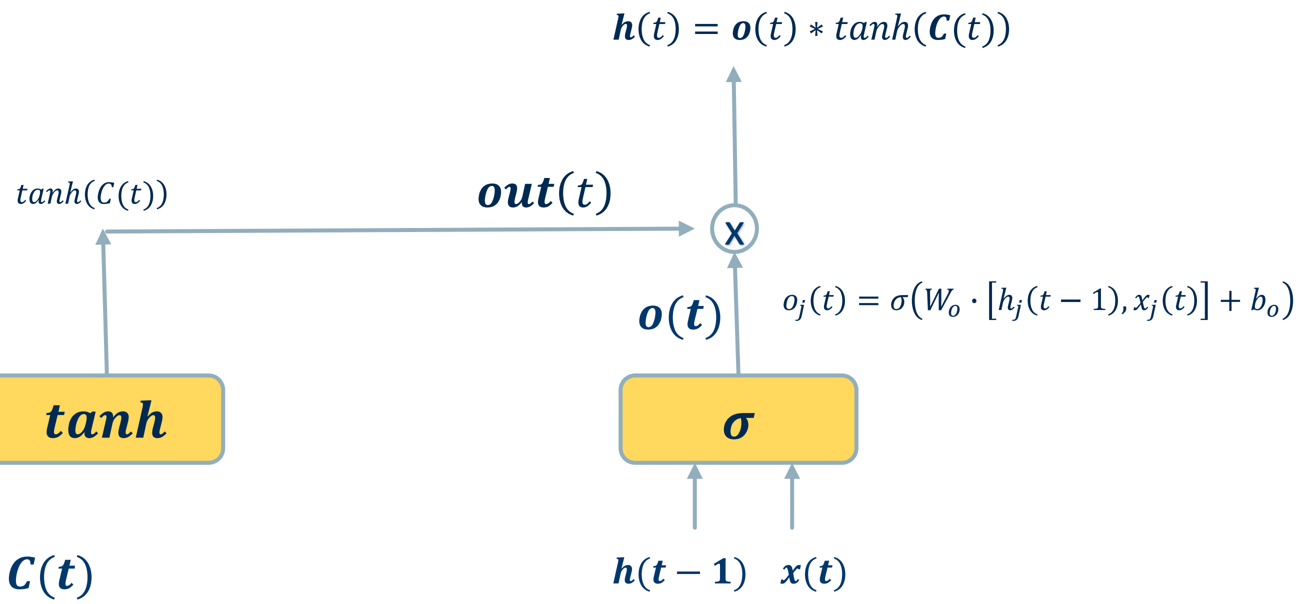


LSTM: Input Gate – prepare output candidate

- ***Output Gate*** is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



- ***Output Gate*** is trained to output a reasonable result.
- At time t, output gate decides which parts of status $\boldsymbol{C}(t)$ (and how much of it) will be output, given input vector $\boldsymbol{x}(t)$ and previous output $\boldsymbol{h}(t-1)$.



LSTM = Long Short Term Memory

Special type of unit with three gates:

Forget gate

Input gate

Output gate

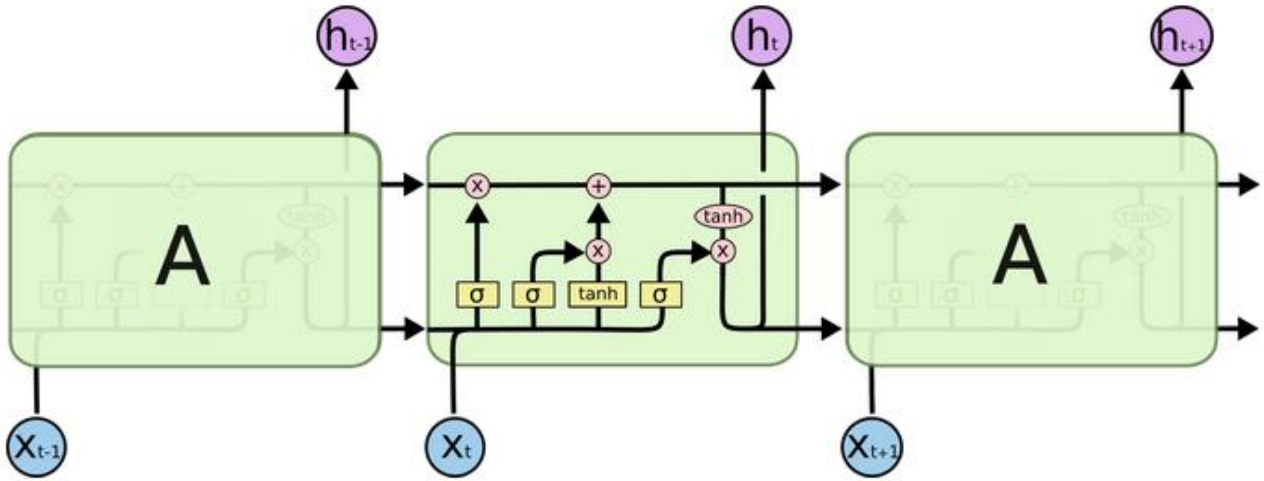
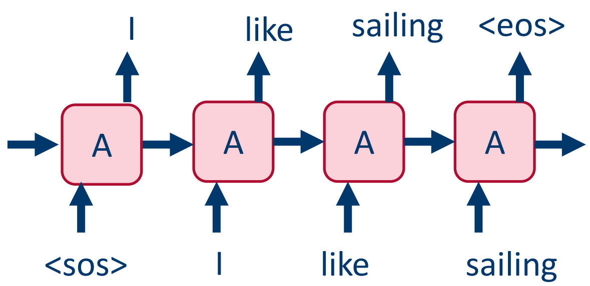
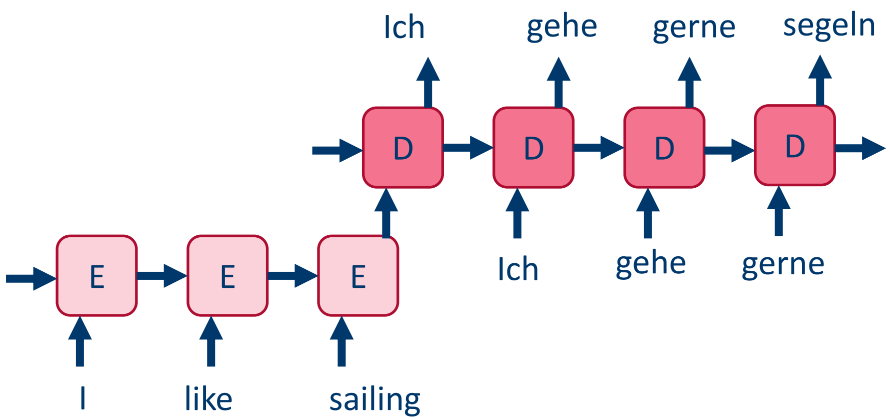


Image Source: Christopher Olah, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Many to Many



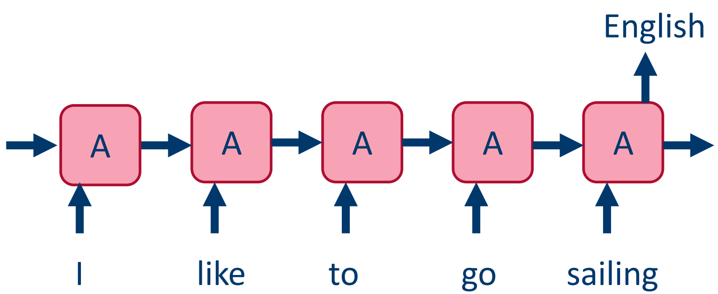
Language model



Neural machine translation

Different Network-Structures and Applications

Many to one



Language classification

Text classification

One to many

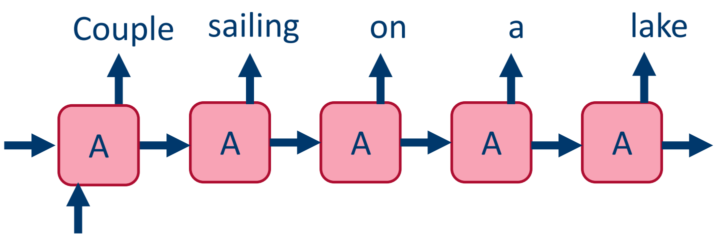
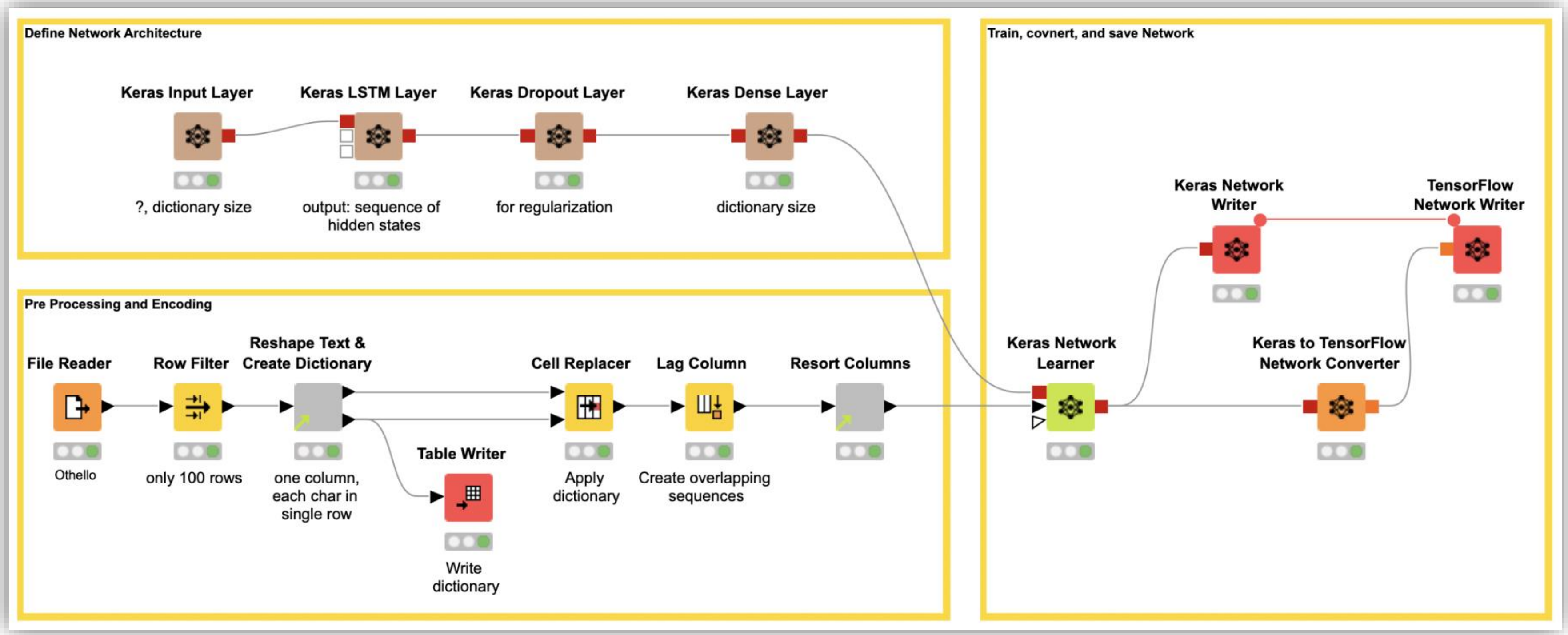
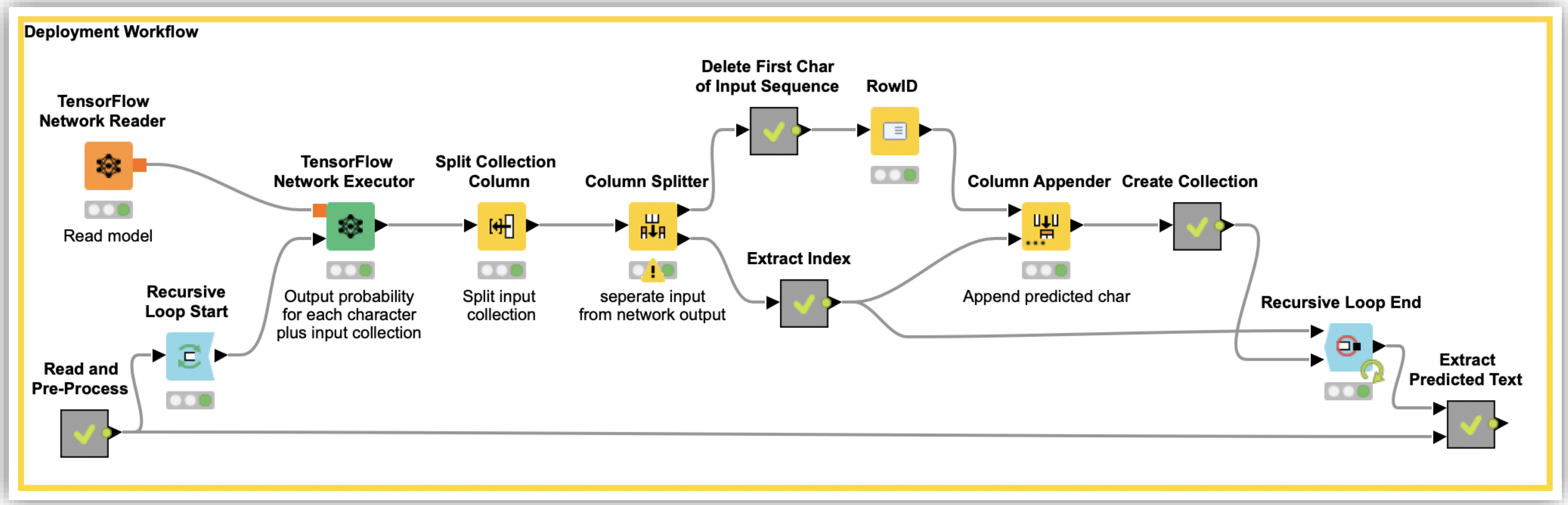


Image captioning

Neural Network: Code-free Example



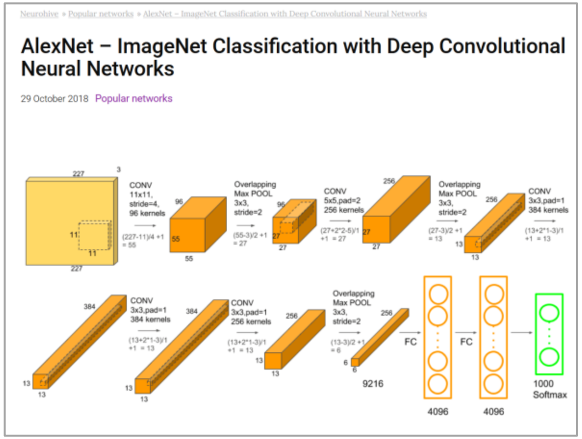
Neural Network: Code-free Example



Convolutional Neural Networks (CNNs)

AlexNet & friends

- The big breakthrough in deep learning happened in 2012 with deep convolutional neural networks
- Here deep learning based AlexNet network won the ImageNet challenge with an unprecedented margin.
- The top-five error rate of AlexNet was 15 percent, while the next best competitor ended up with 26 percent.
- This victory kicked off the surge in deep learning networks.



https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/

- Inspired by the organization of the visual cortex in the human brain, convolutional layers simulate the concept of a receptive field.
- Individual neurons in the convolutional layer respond only when a specific area of the image (the visual field) is active.
- An array of such neurons covers the entire image by responding to slightly overlapping separated areas of the input image.

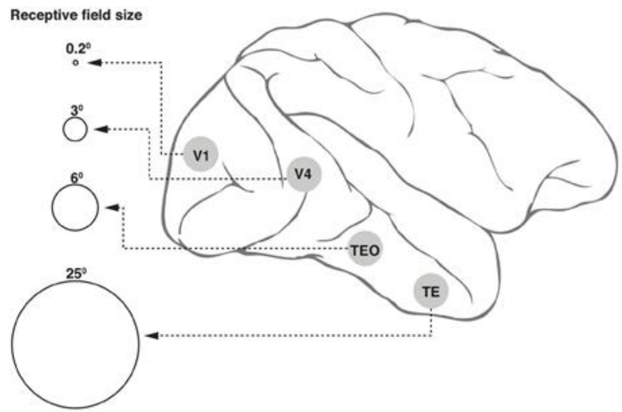
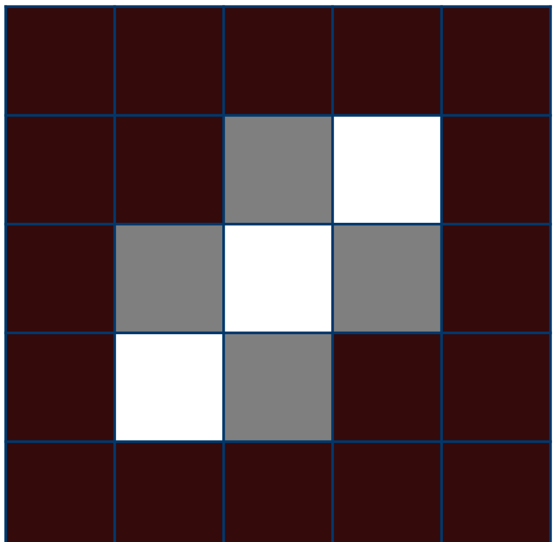


Image from: Wikimedia commons –
https://commons.wikimedia.org/wiki/File:Receptive_field_sizes_along_the_ventral_cortical_stream_in_the_primate.jpg

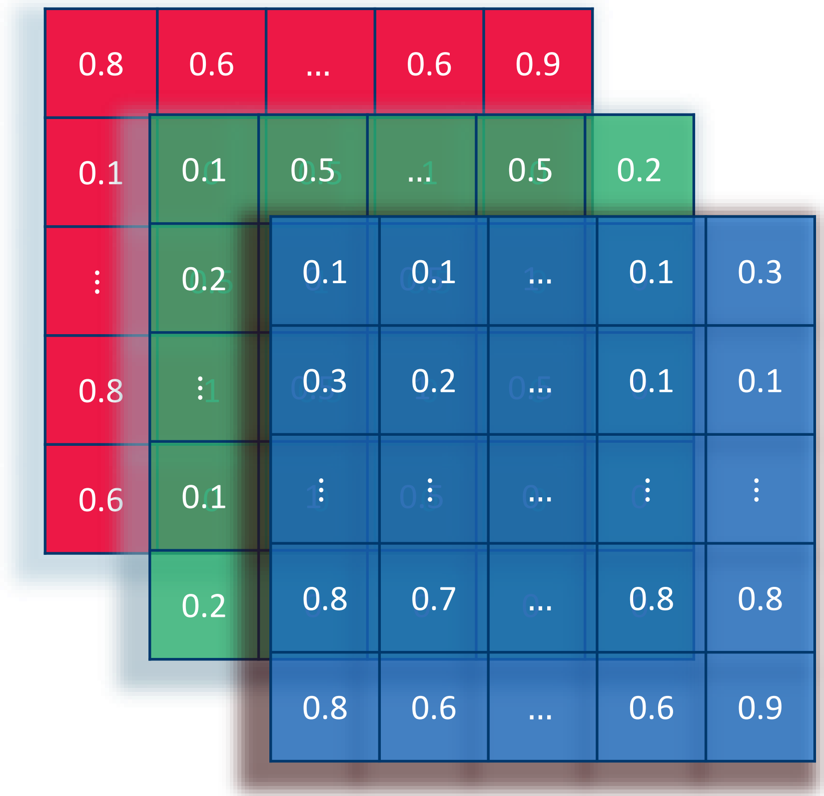
Numerical Representation of a Black and White Image



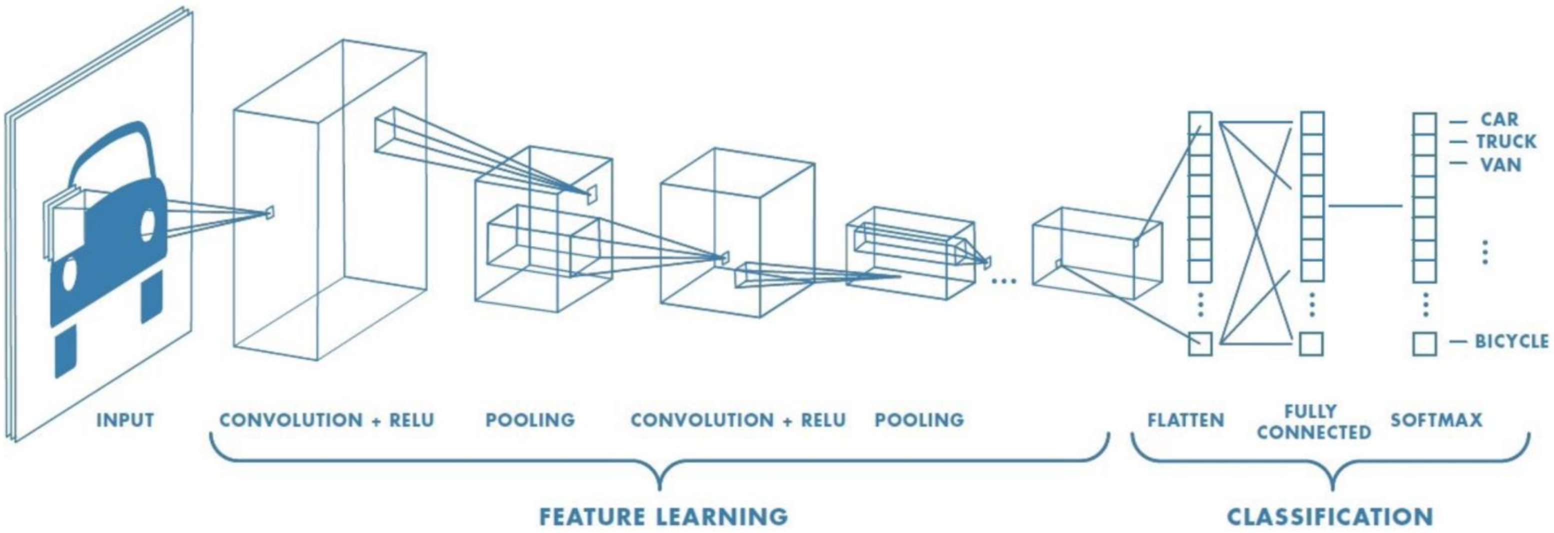
0	0	0	0	0
0	0	0.5	1	0
0	0.5	1	0.5	0
0	1	0.5	0	0
0	0	0	0	0

Numerical Representation of a Color Image

=

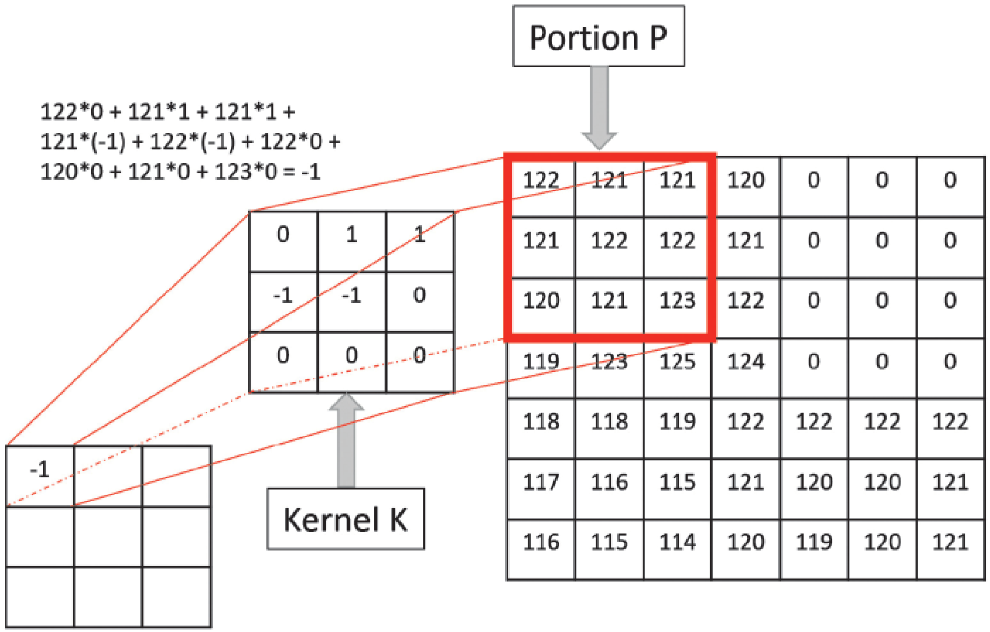


Convolutional Neural Networks



- The idea of convolution relies on a kernel *K*, a mask to overlap onto a portion *P* of the image pixels for the convolution operation.
- From the product of the kernel *K* and the pixels in portion *P* we get a number, which will be the output of the first neuron in the convolutional layer.
- Then the kernel *K* moves *n* steps on the right and goes to cover another portion *P* of the image possibly slightly overlapping with the previous one; the output for the second unit of the convolutional layer is generated.
- And so on till the whole image has been covered by the kernel *K* and convoluted into output values.
- The distance in number of pixels *n* between two adjacent portions *P* is called *stride*.

Convolutional Neurons: Example



- Zero padding
  - Artificially increases the input at the boundary
  - Helps with preserving the spatial resolution and alignment

- Stride
  - The *jump* the kernel makes when moving over the input
  - Reduces the spatial resolution

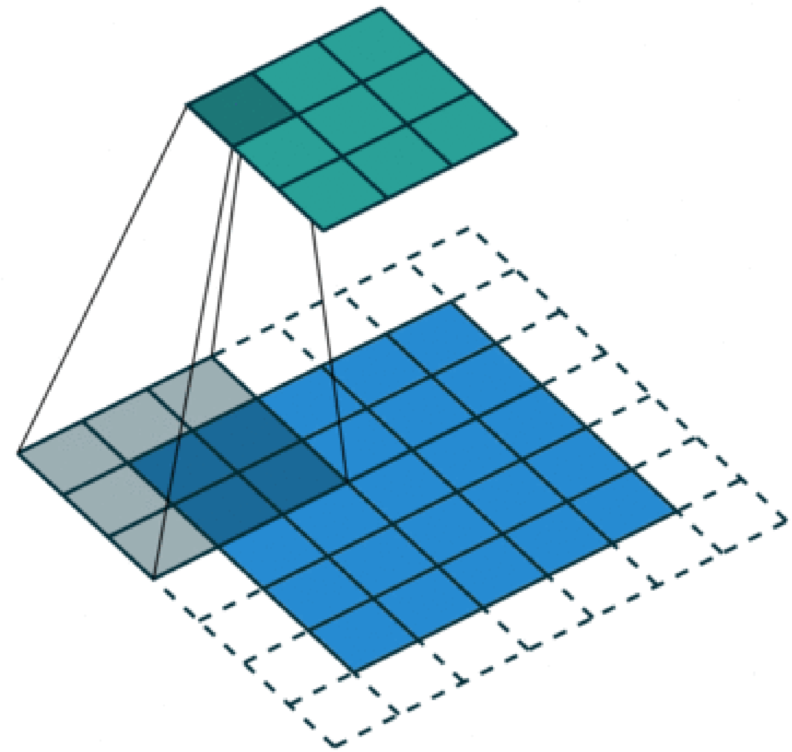
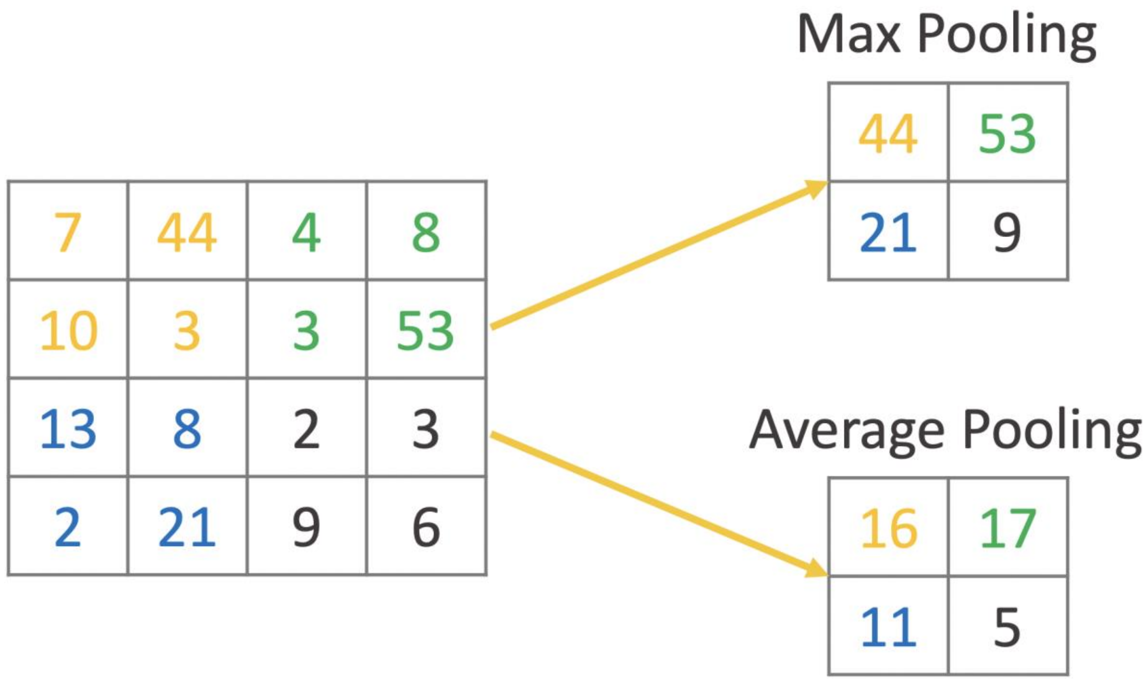


Image from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Pooling Layers

- Usually a number of convolutional layers are used.
- Each layer provides one further step in the process of extracting high-level features from the input image (colors, edges, entities, …).
- *Pooling layers* are often used to reduce the spatial resolution in between convolutional layers to
  - Increase the *receptive field* of the following layers
  - Reduce computational complexity
- Two types of Pooling
  - **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel.
  - **Average Pooling** returns the **average of all values** from the portion of the image covered by the Kernel.

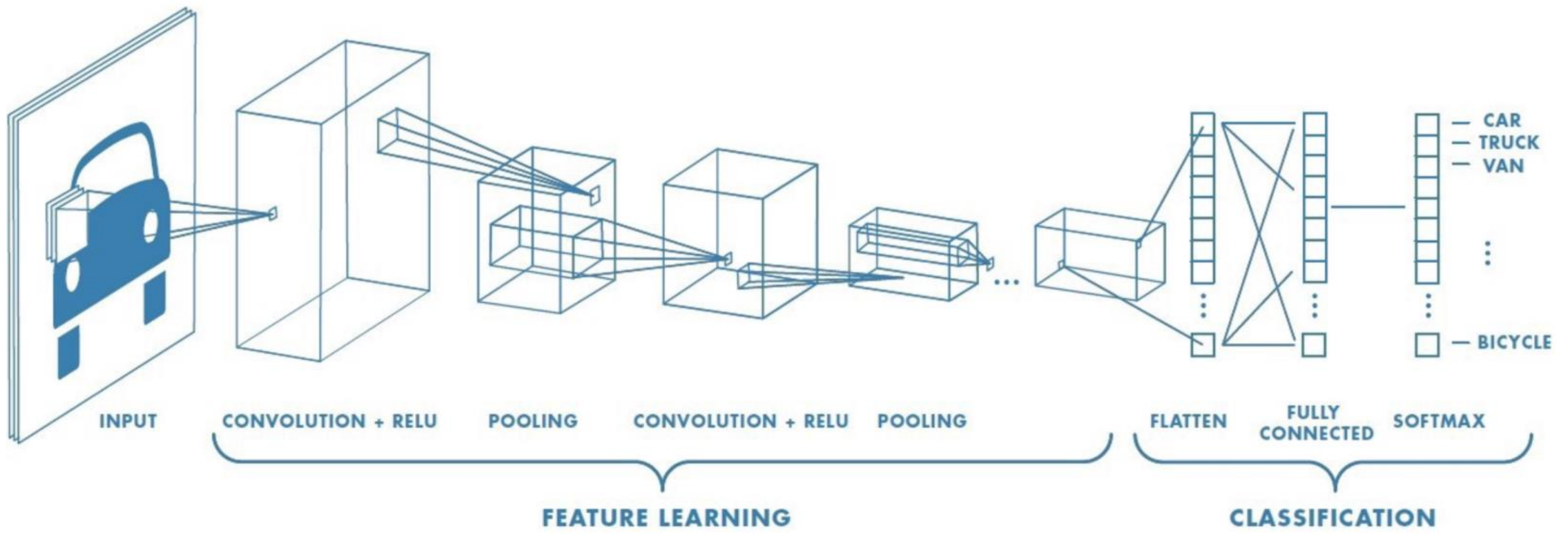
Example Pooling



Classification Layers

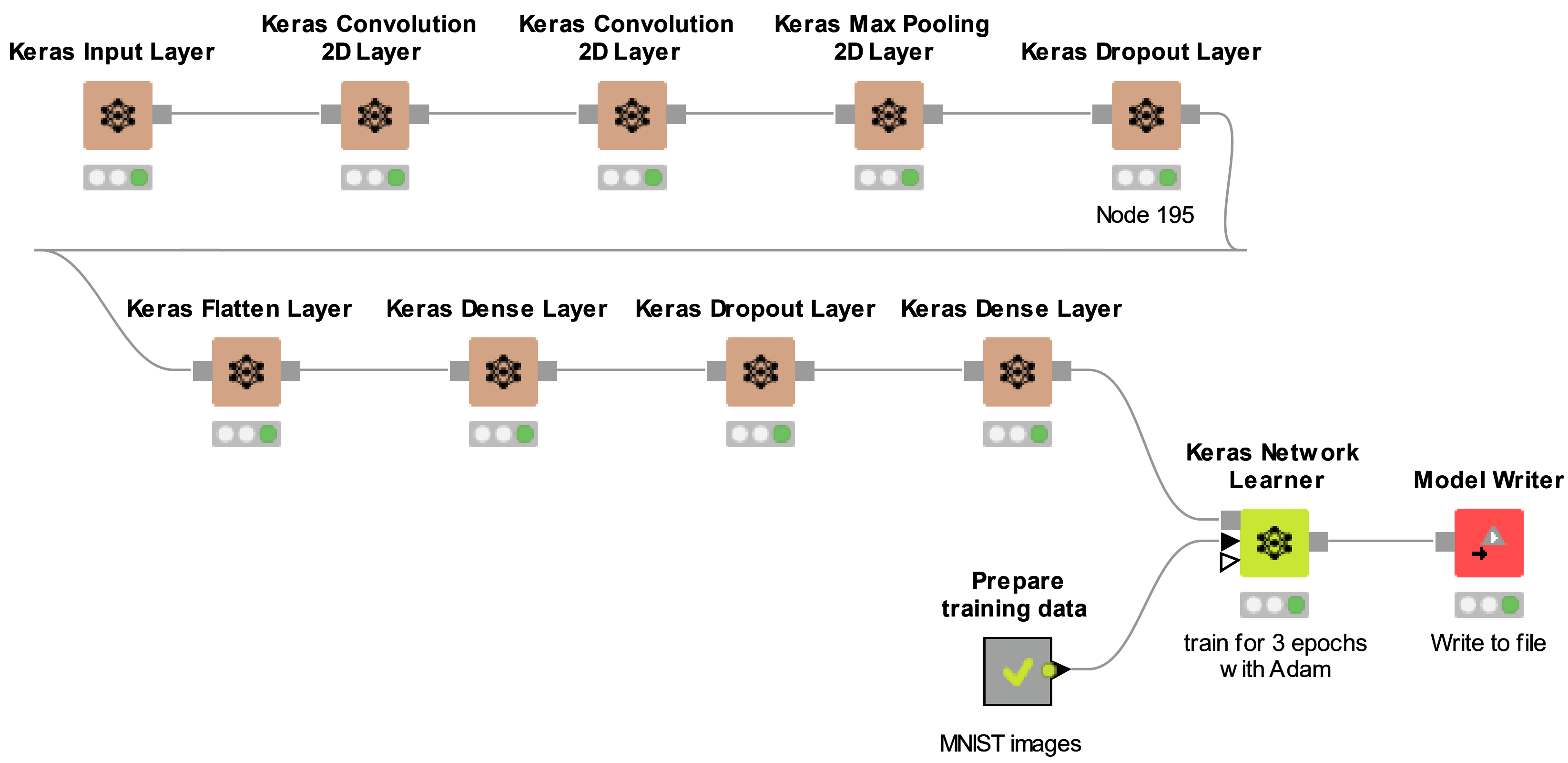
– After the sequence of convolutional + pooling layers, a classic feedforward multilayer Perceptron network is applied to carry out the classification process.

– Successful examples of CNNs for image recognition : LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, ZFNet.



- Training such networks is a long and complex process, requiring very powerful machines.
- Instead of retraining a new network completely from scratch, we could recycle existing networks, already built and trained by others on **similar** data.
- This technique is called ***Transfer Learning***.
- In Transfer Learning a model developed for a task is reused as the starting point for another model on a second task.
- On top of a previously trained network we add one or more neural layers
- We freeze all or some of the previously trained layers
- And we retrain only the remaining part of the whole network on our new task

Building CNNs with KNIME



Generative-Adversarial Networks (GANs)

- So far: RNNs and CNNs
- Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) represent probably the biggest contribution of deep learning to the field of neural networks.
- However, deep learning is responsible for other innovations, such as for example Generative Adversarial Networks (GANs).

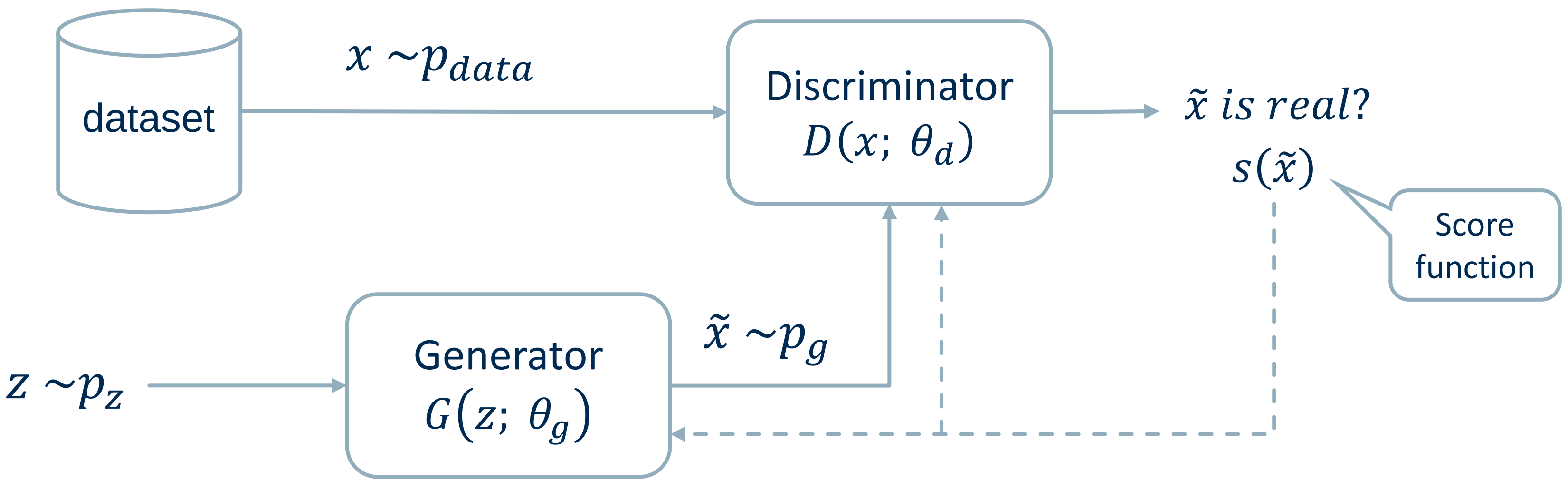
Can You Tell Real from Fake?

Source: https://thispersondoesnotexist.com

- GANs include two neural networks competing with each other: the generator and the discriminator.
- A **generator *G*** is a transformation that transforms the input noise *z* into a tensor – usually an image – *x* (*x*=*G(z)*). The generated image *x* is then fed into the discriminator network *D*.
- The **discriminator network D** compares the real images in the training set and the image generated by the generator network and produces an output D(x), which is the probability that image *x* is real.

- Both generator and discriminator are trained using the backpropagation and gradient descent.
- Both networks are trained in alternating steps, competing with each other to improve themselves.
  - The objective of the generator is to fool the discriminator i.e. $D\big(G(z)\big) = 1$
  - The objective of the discriminator is to output $D\big(G(z)\big) = 0$ and $D(x_{real}) = 1$
- The GAN model eventually converges and produces images that look real.
- Given a training set, this technique learns to generate new data under the same statistics as the training set.

GAN: Architecture



GANs

- For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

- GANs have been successfully applied to image tensors to create anime, human figures, and even van Gogh-like masterpieces.

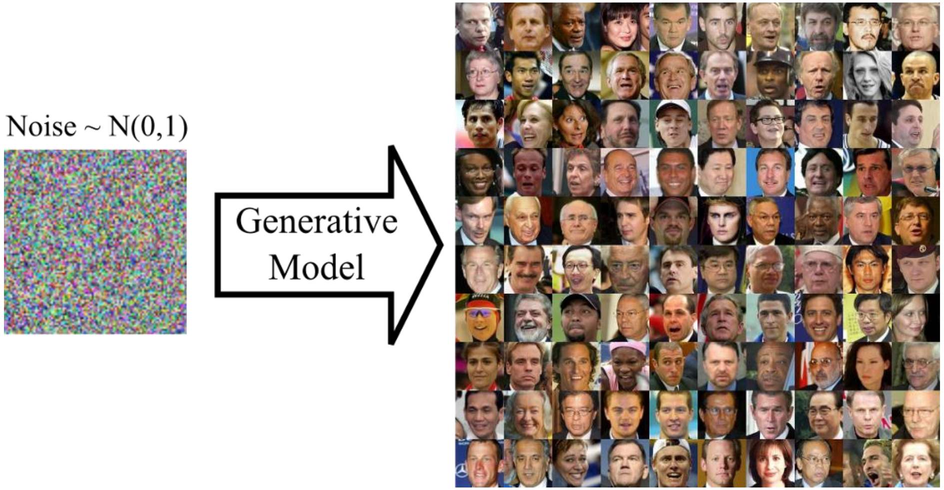


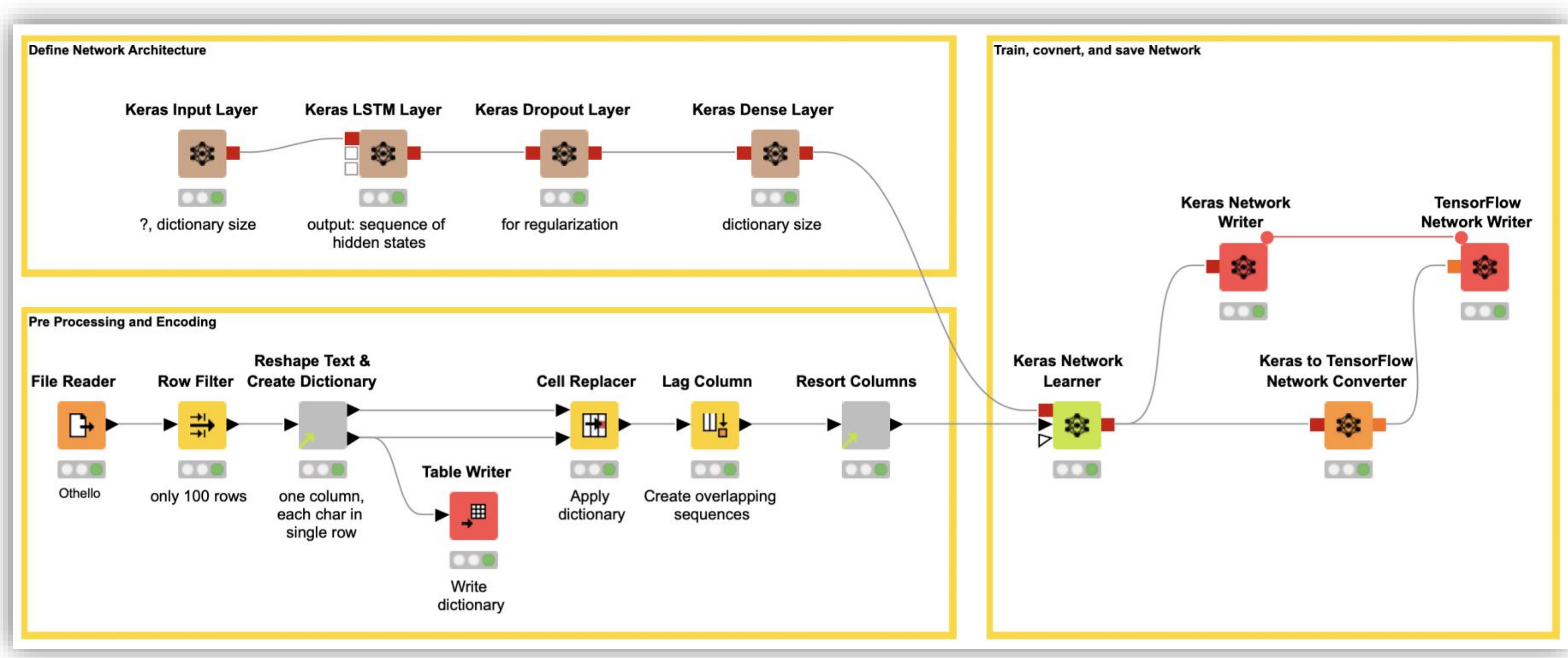
Image from: Pankaj Kishore, Towards data Science
https://towardsdatascience.com/art-of-generative-adversarial-networks-gan-62e96a21bc35

Summary

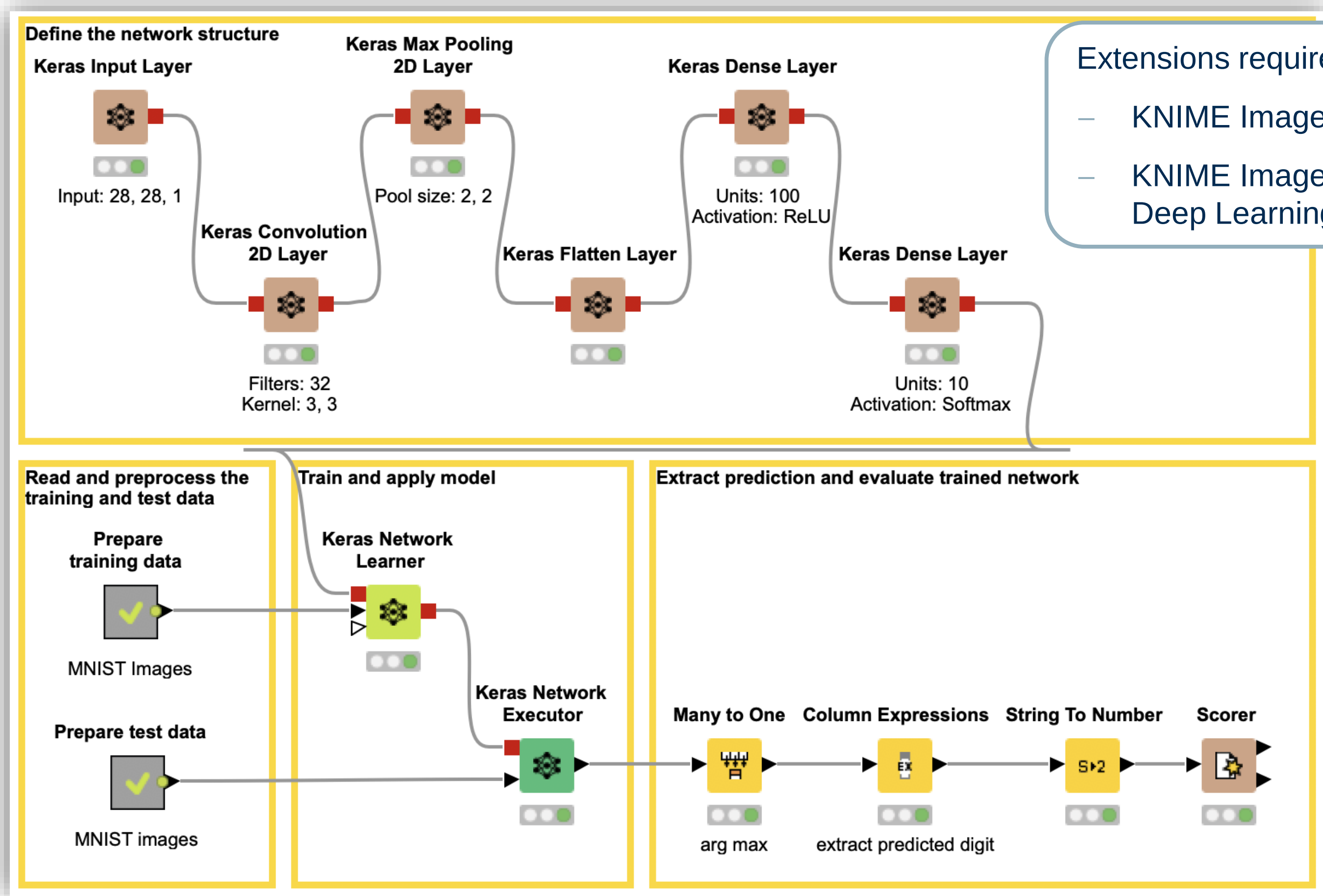
- Recurrent Neural Networks (RNNs)
- Long Short Term Memories (LSTMs)
- Convolutional Neural Networks (CNNs)
- Generative Adversarial Networks (GANs)

Practical Examples with KNIME Analytics Platform

RNN Workflow: Text Generation



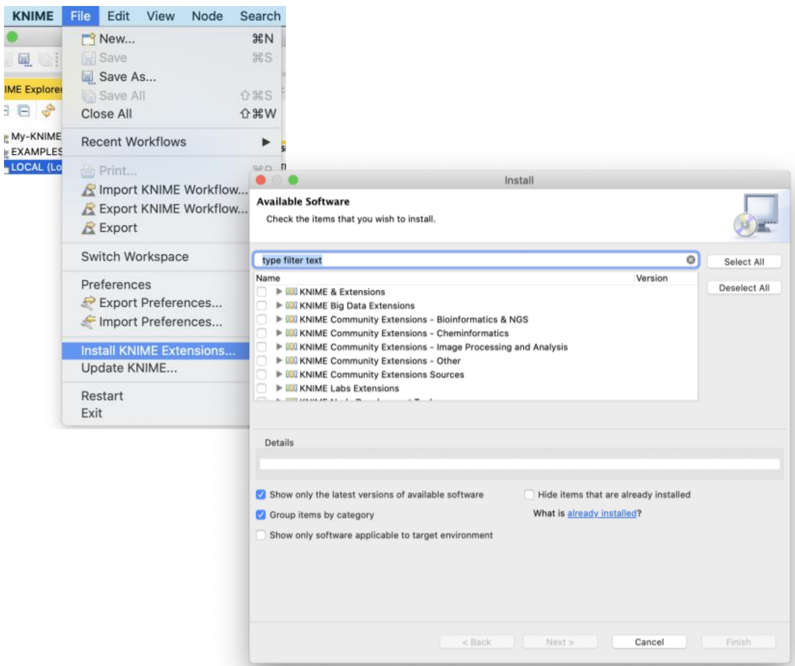
CNN Workflow: Image Classification using MNIST

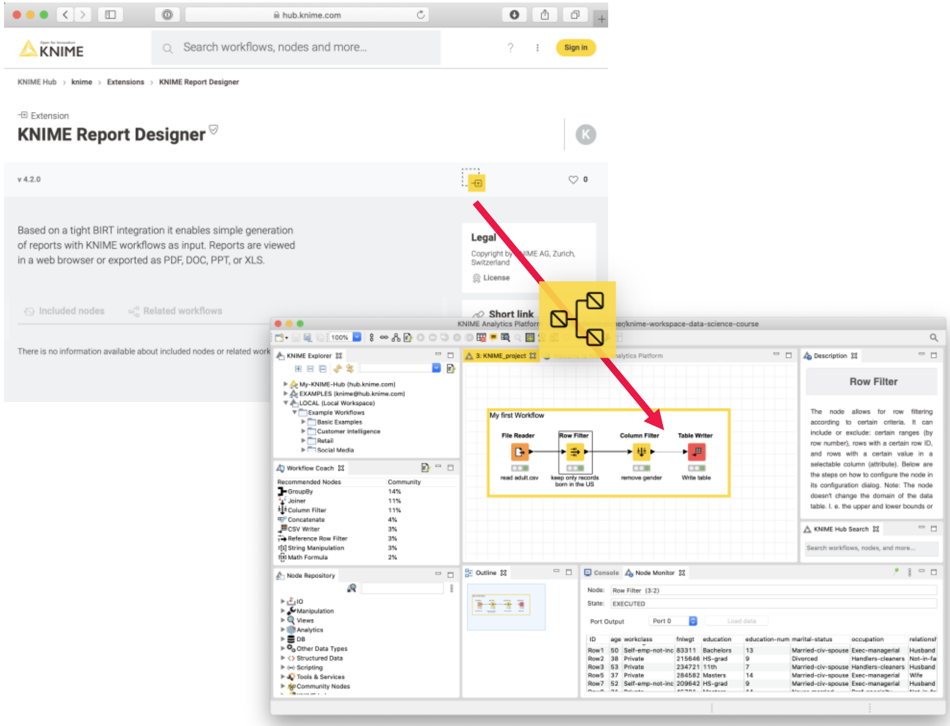


Extensions required:

- KNIME Image Processing
- KNIME Image Processing – Deep Learning Extension

– Install extension by going to File -> Install KNIME Extension or via Drag & Drop from the KNIME Community Hub





Thank you