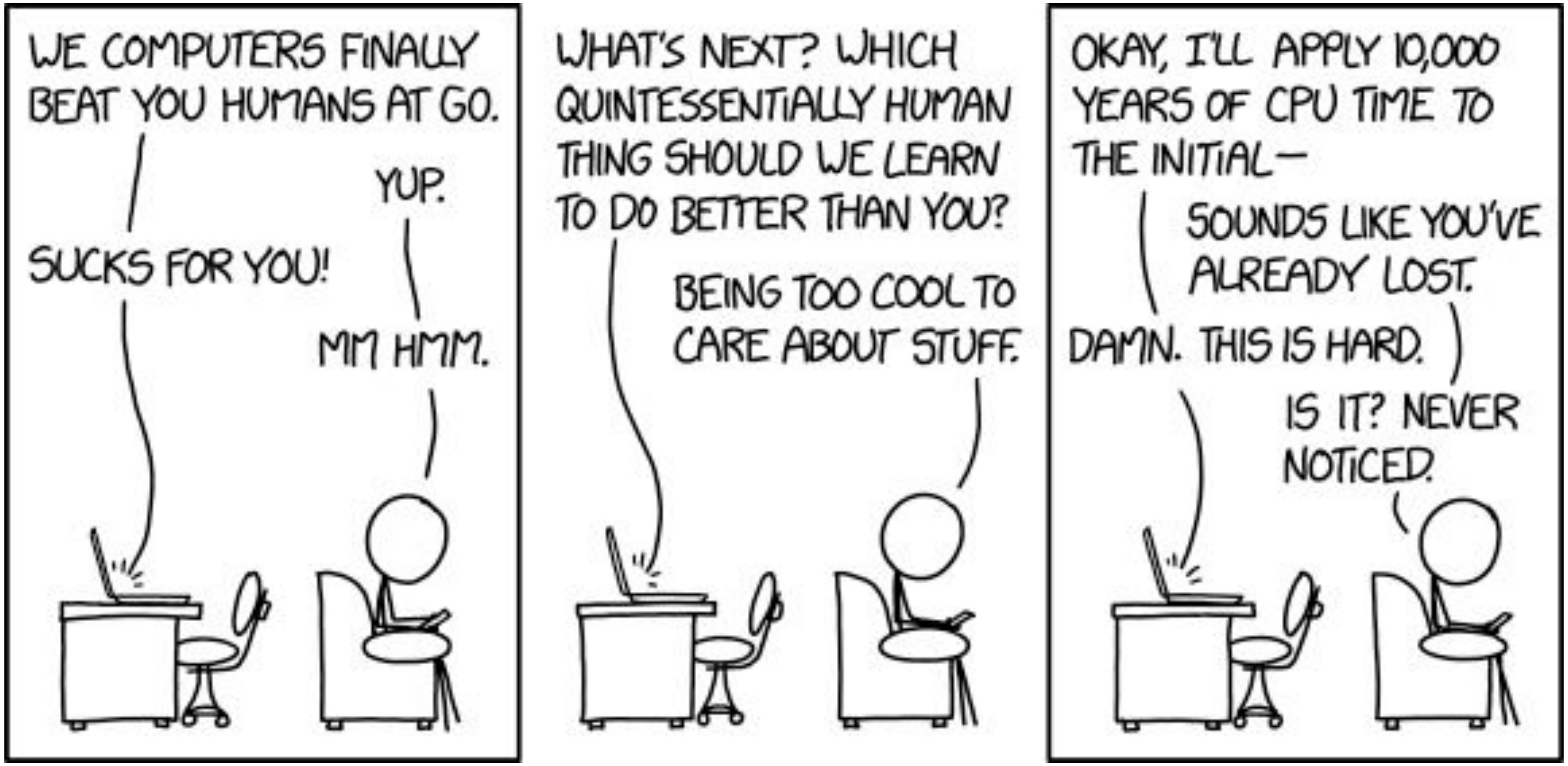
An Introduction to Deep Learning

Hunter Gabbard
University of Glasgow

Outline

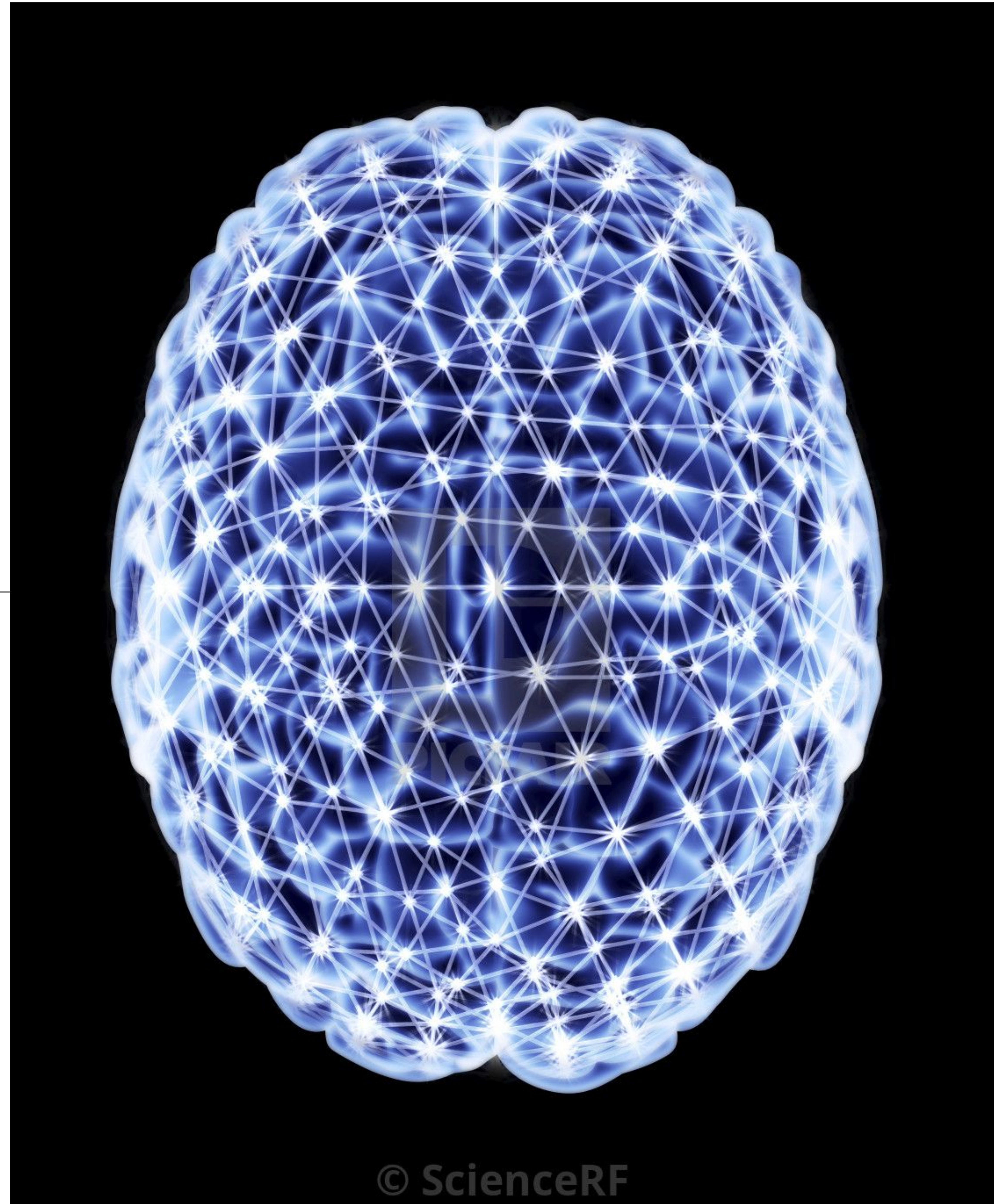
- Introduction
- Fully-Connected Neural Networks
- Convolutional Neural Networks
- LSTM Neural Networks
- GANs



https://www.explainxkcd.com/wiki/index.php/1875:_Computers_vs_Humans

Fully-Connected Neural networks

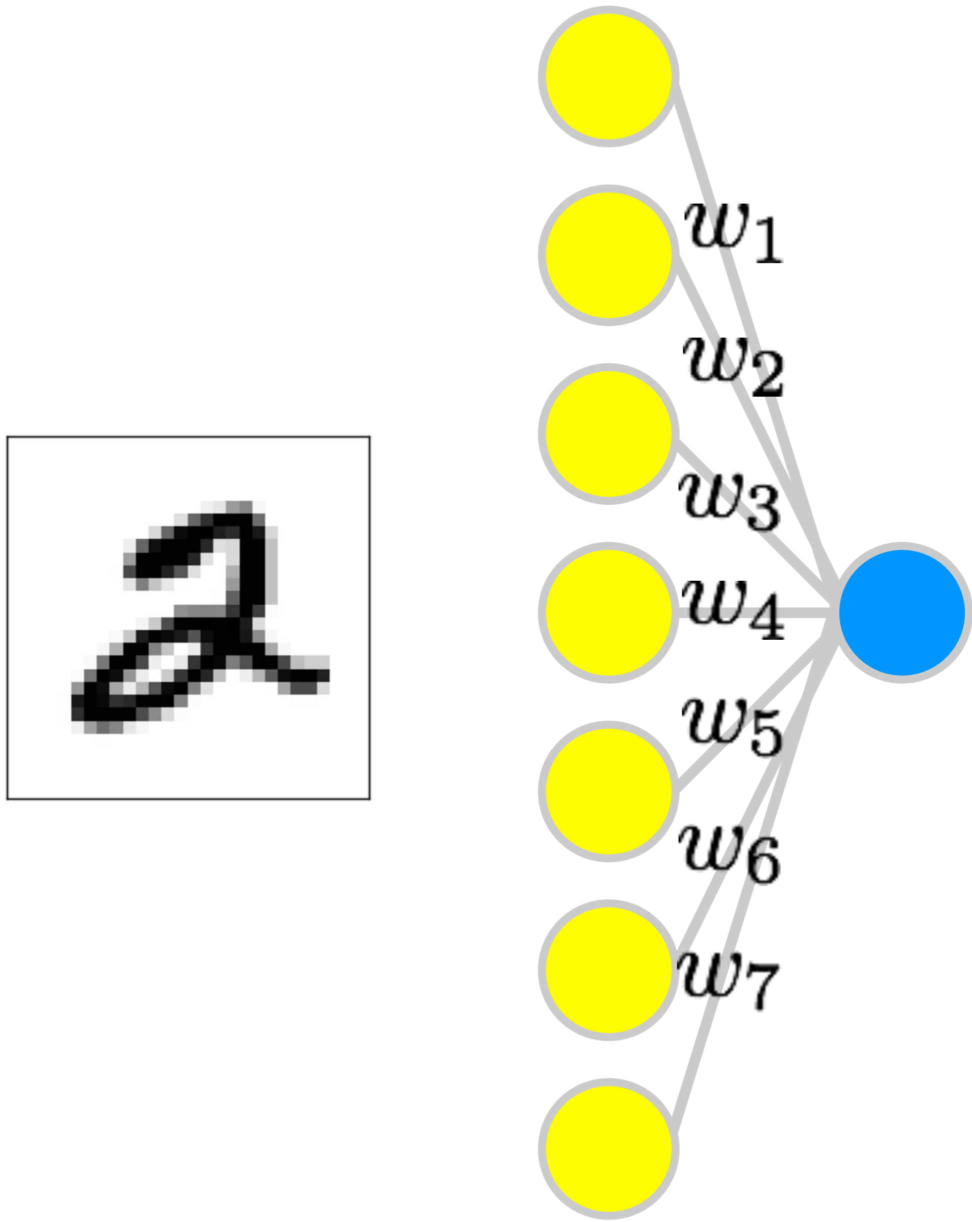
They’re not that hard to understand (honestly)



The MNIST dataset

LeCun et al. (1999)

A single neuron

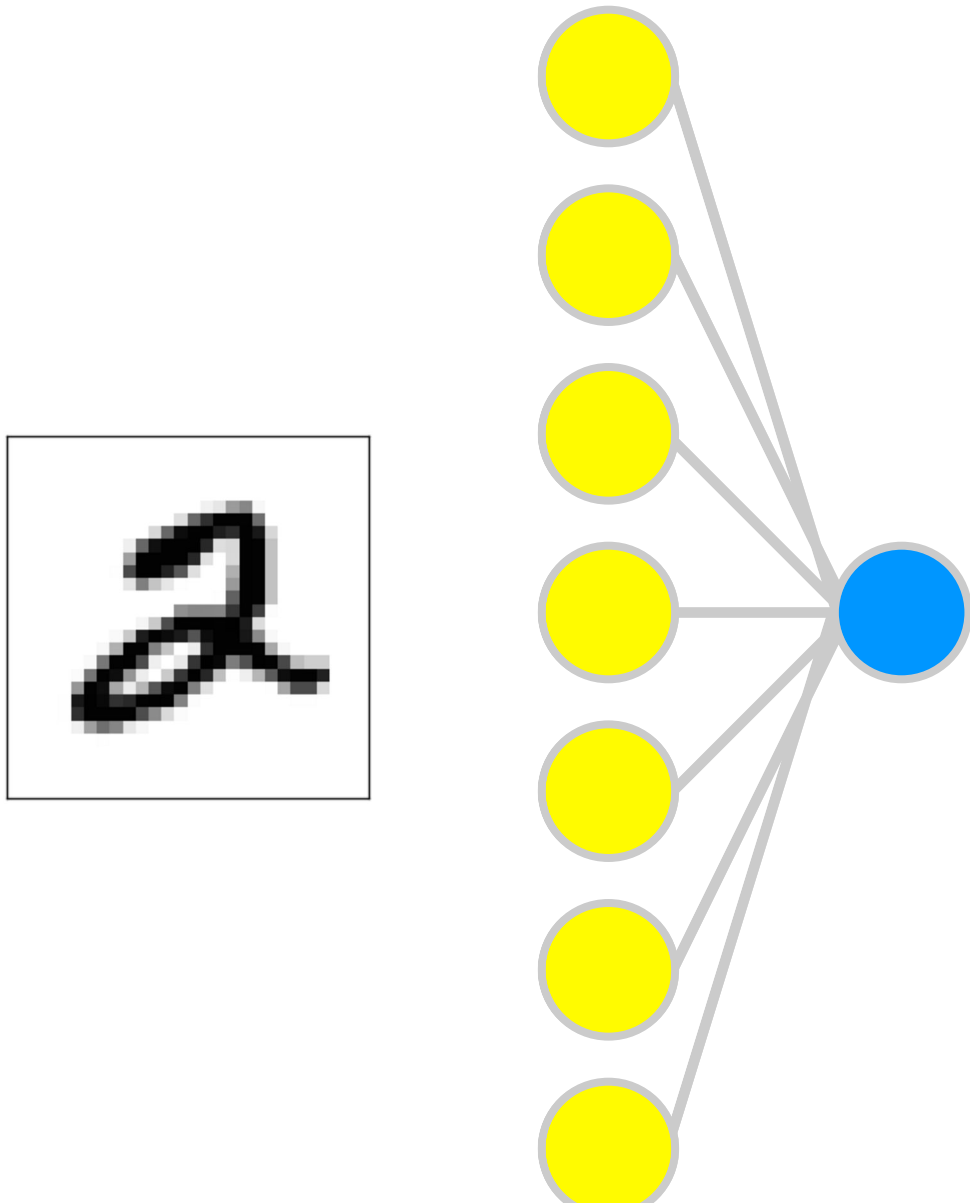


input layer

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

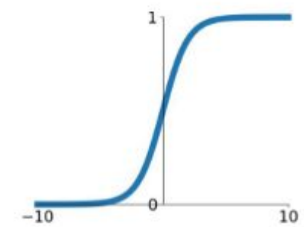
the specific values of the weights and bias are not determined until training is performed

Activation functions



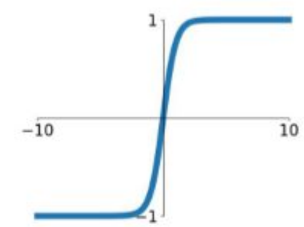
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



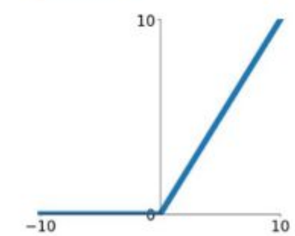
tanh

$\tanh(x)$



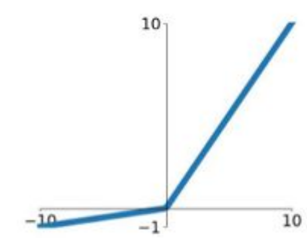
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

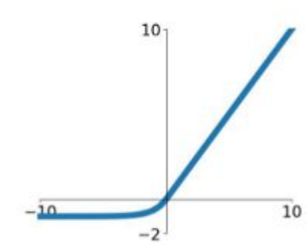


Maxout

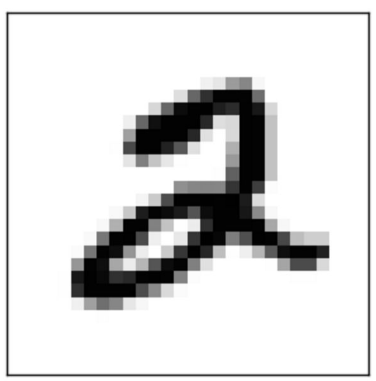
$\max(w_1^T x + b_1, w_2^T x + b_2)$

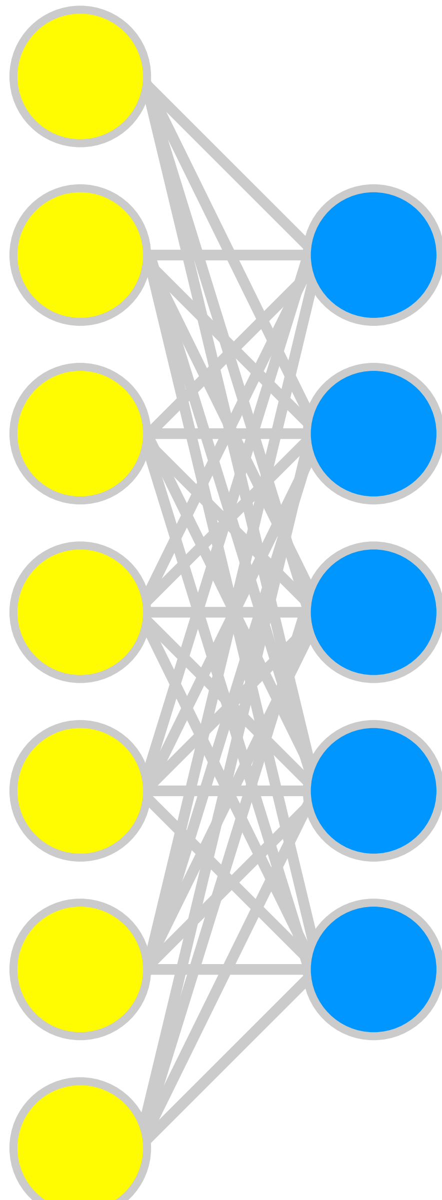
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



A simple network - A layer





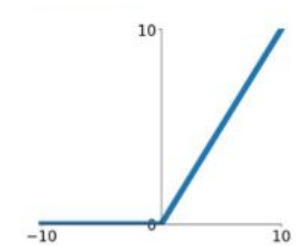
- Fully connected layer- all inputs to all neurons
- Overall result is a big matrix operation

$$\begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & w_{d3} & \dots & w_{dn} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$$

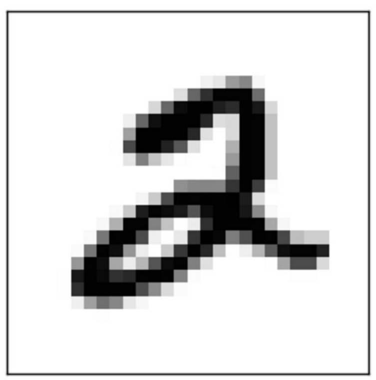
- still processed through a non-linear activation function, e.g.,

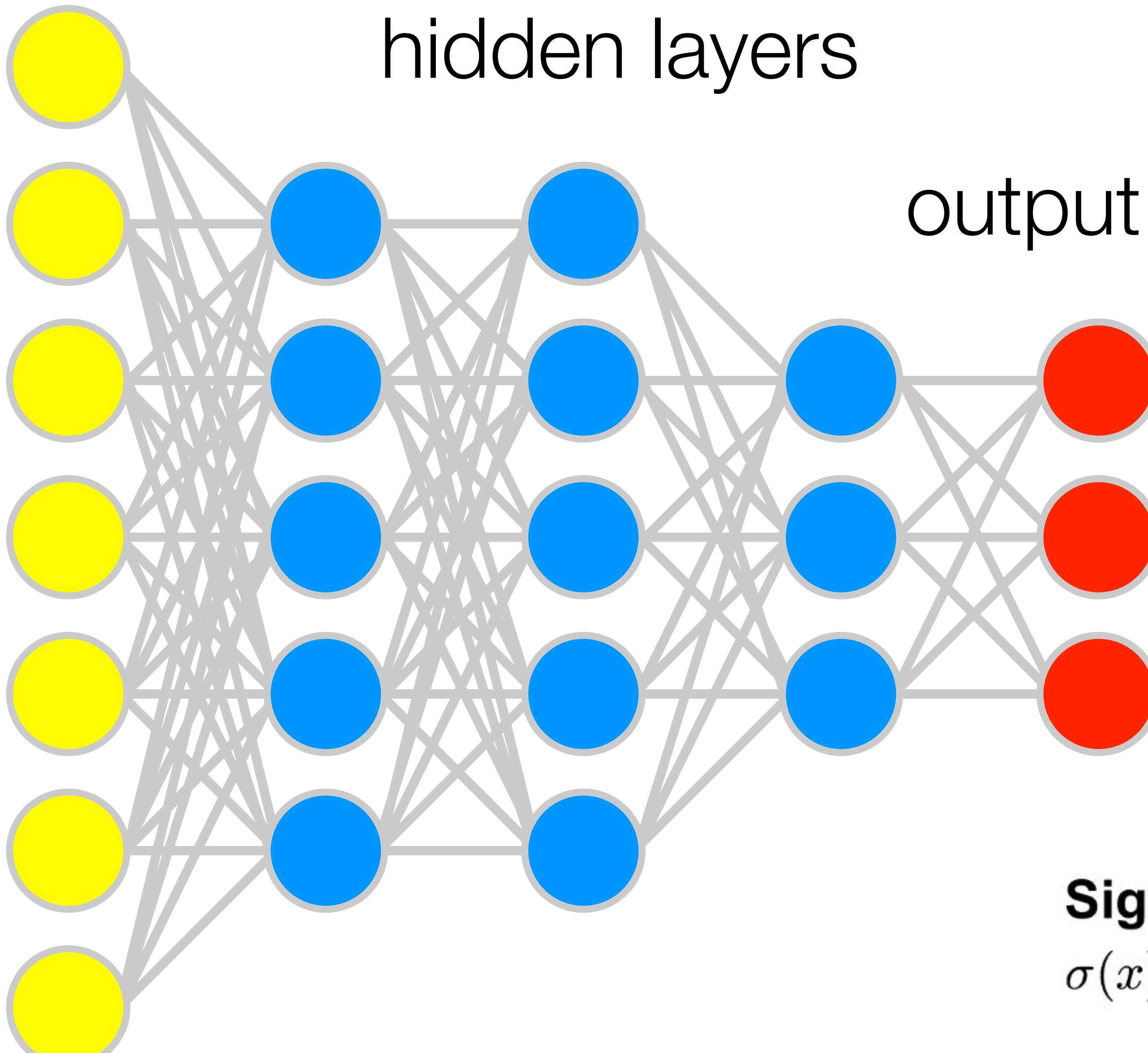
ReLU
$\max(0, x)$



A simple network - Multiple layers

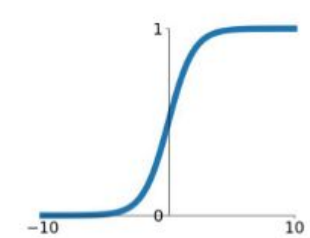
multi-layer perceptron





Sigmoid

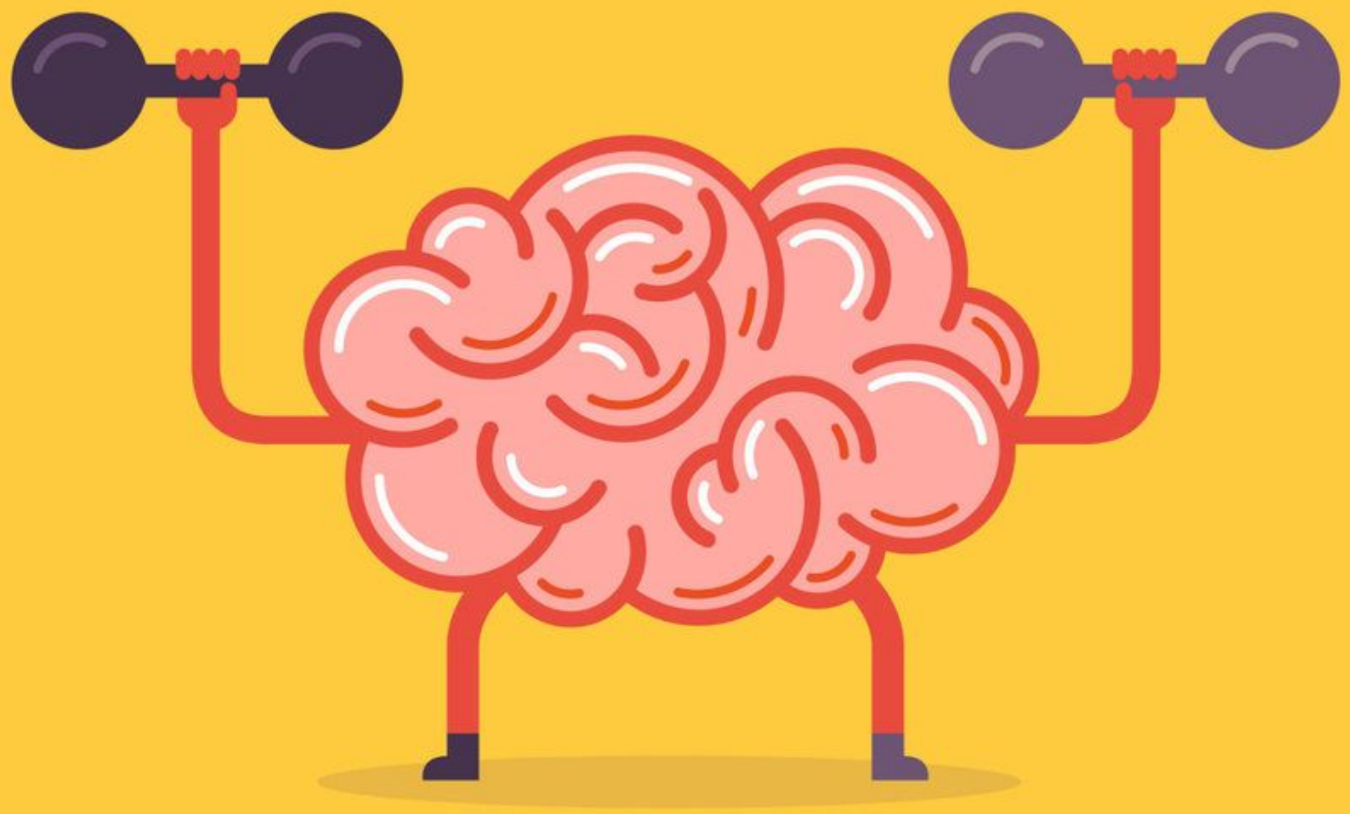
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



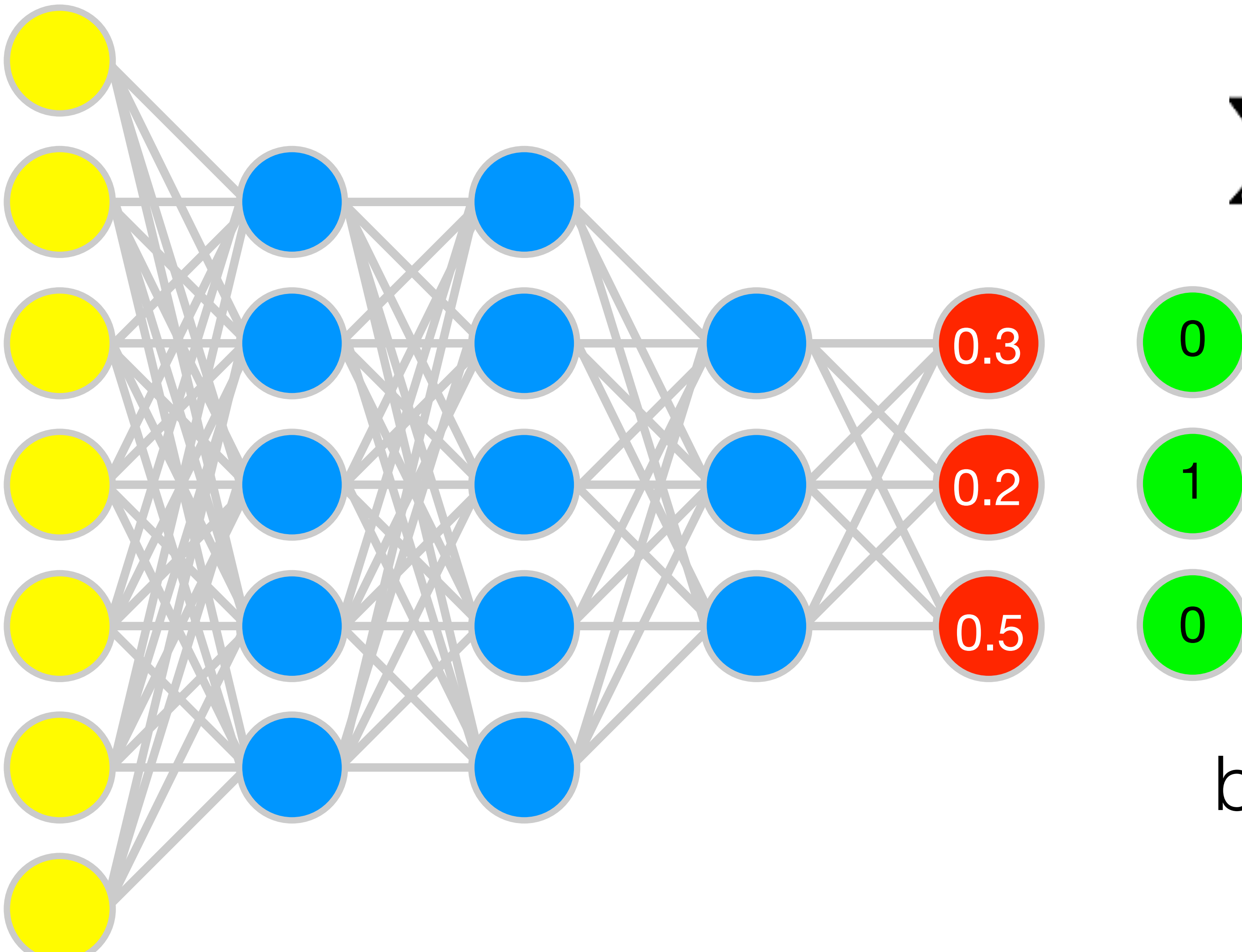
classification

Training/Learning

A brain is useless without input



Loss functions



least squares

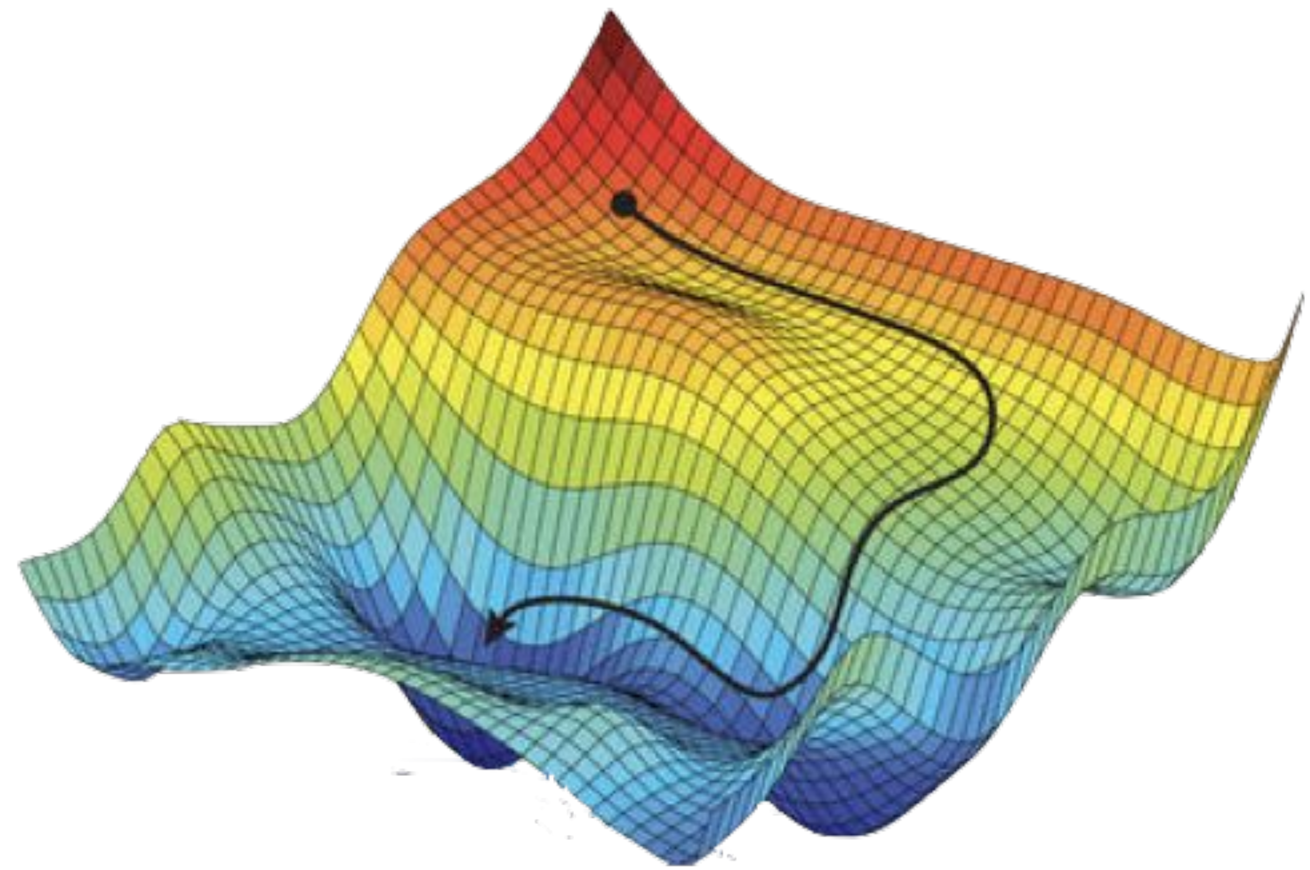
$$\sum_i \left(y_i^{\text{true}} - y_i^{\text{pred}} \right)^2$$

binary cross-entropy

$$-\sum_j^{n \text{ class}} p_j^{\text{true}} \log p_j$$

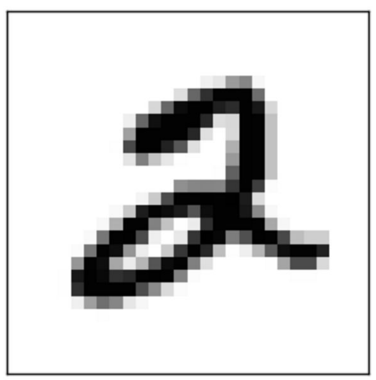
Gradient descent

- Very large number of parameters
- Compute local gradient and move “downhill” in proportion
- Tricks to avoid local minima
  - Work in “batches” - stochastic gradient descent
  - Learning rate
  - Use momentum
- in reality you can have millions of weights and biases

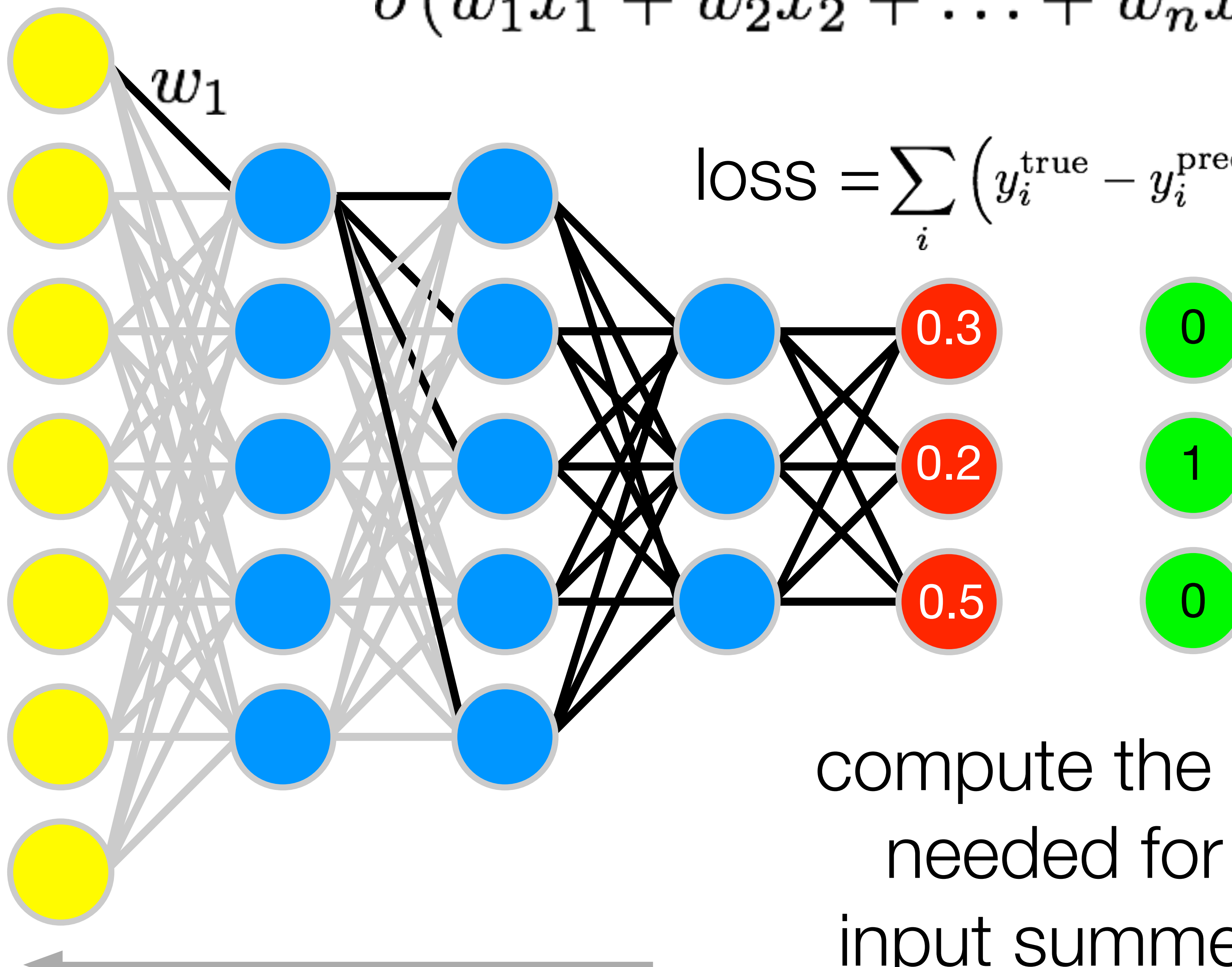


Machine “learning” here is simply minimising a loss function

Back propagation



$$\sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$



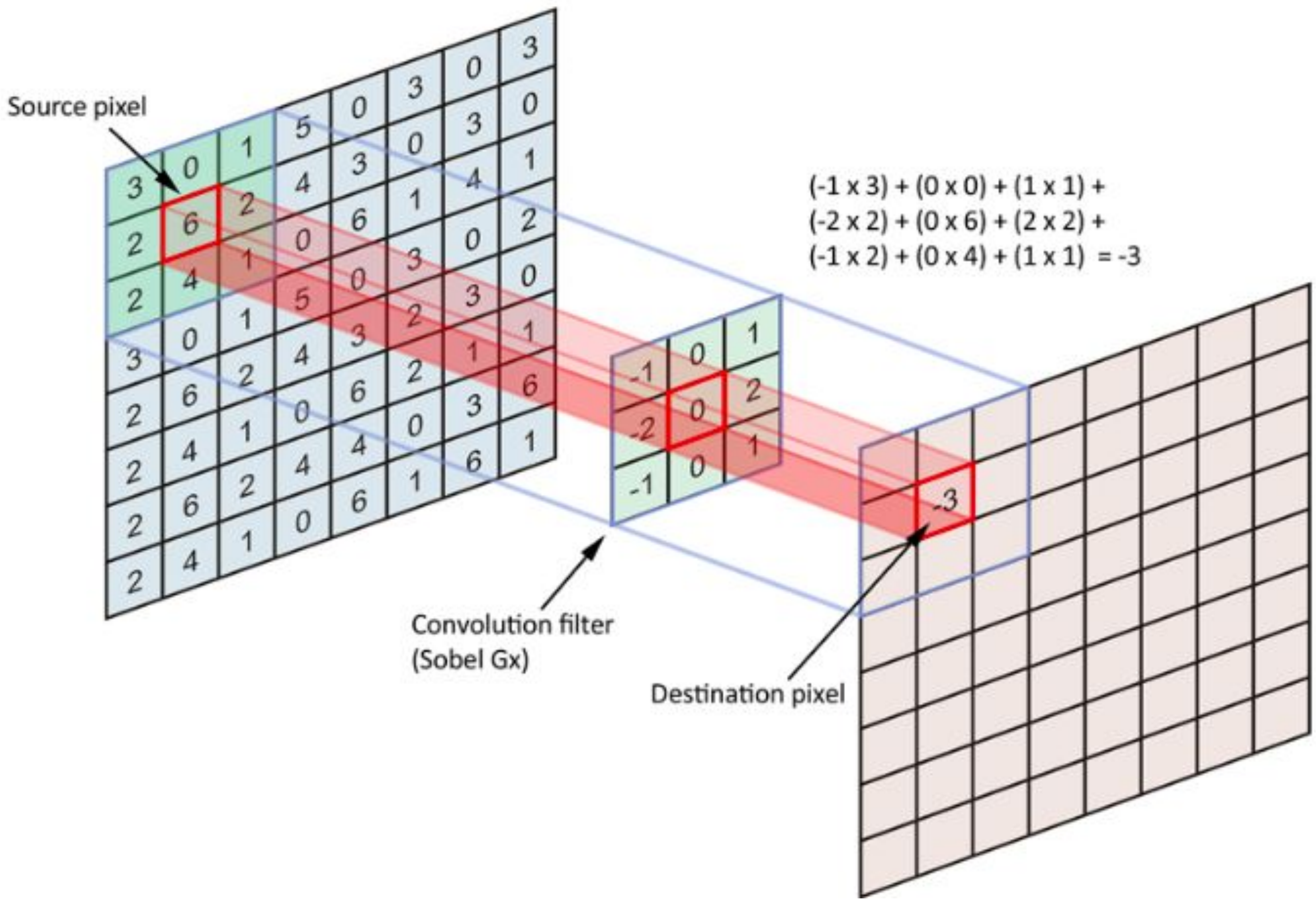
$$\text{loss} = \sum_i \left(y_i^{\text{true}} - y_i^{\text{pred}}\right)^2$$

propagate backwards

compute the nudges needed for each input summed over each output

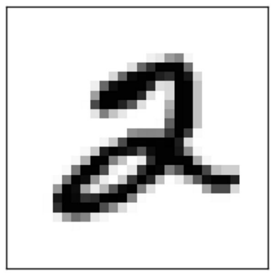
Convolutional Neural Networks

Convolutional Filter

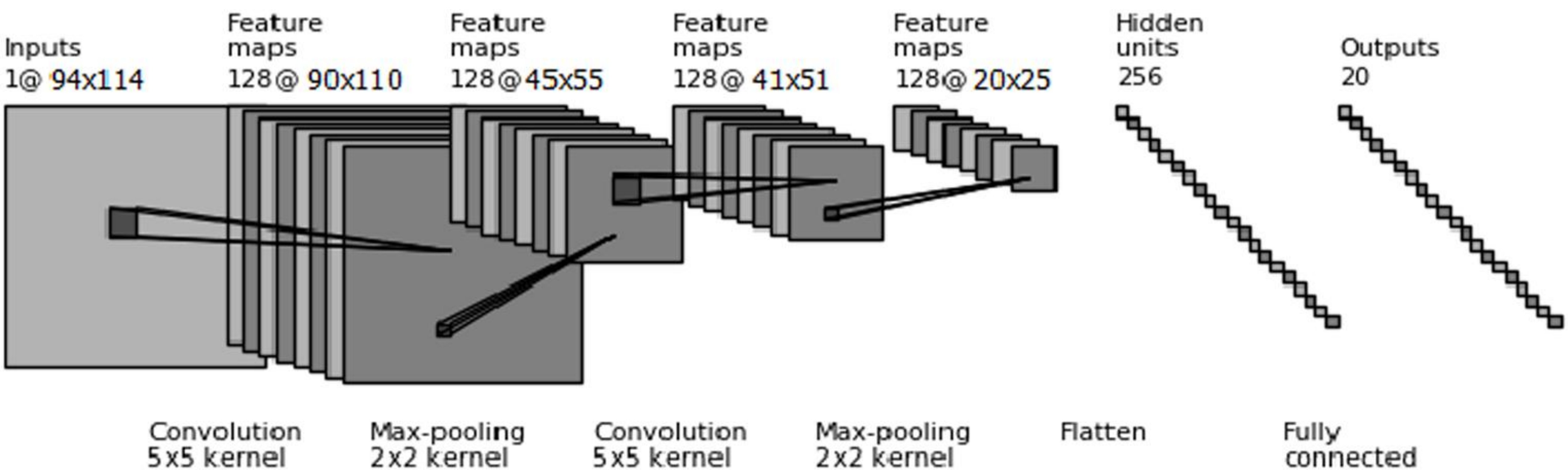


Convolutional Stride

CNN Structure

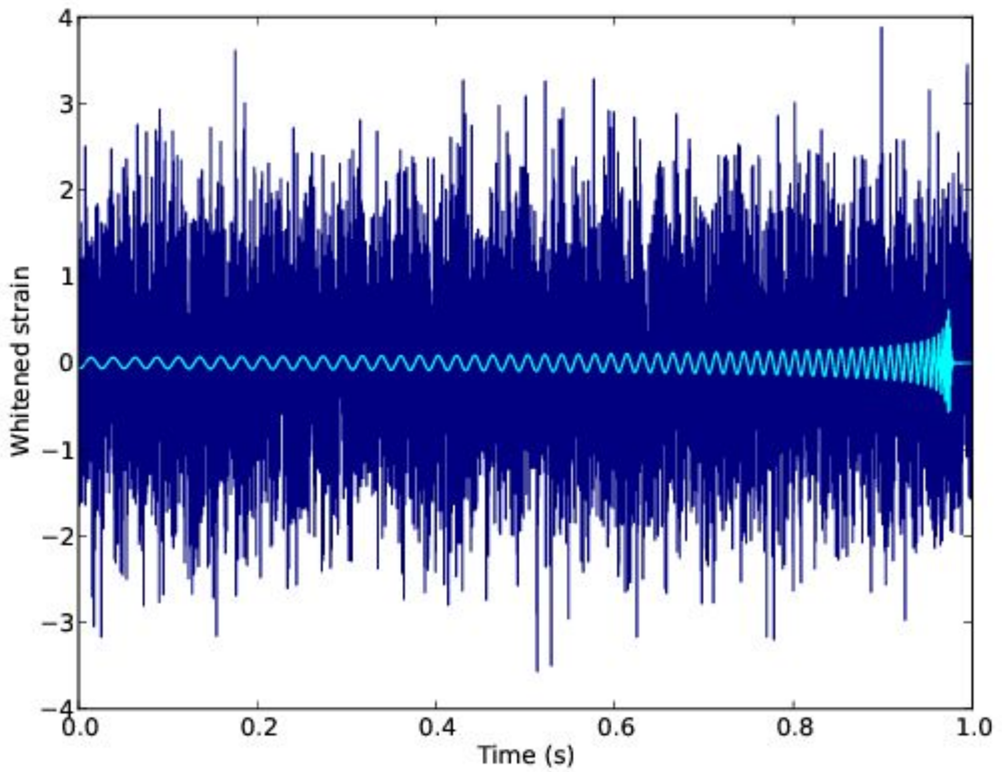


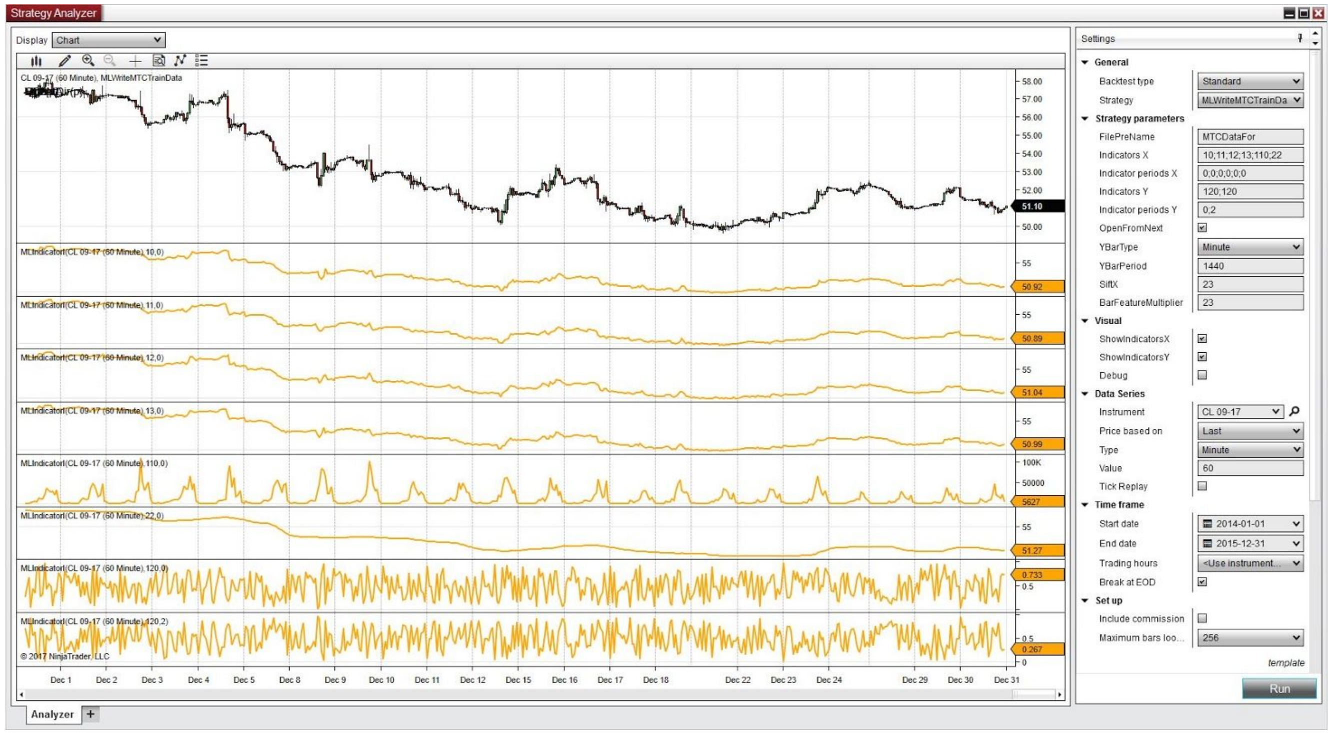
$$\begin{bmatrix}5&2&4&6\\3&9&7&1\\7&5&4&4\\8&7&1&8\end{bmatrix}\otimes\begin{bmatrix}1&0\\0&1\end{bmatrix}=\begin{bmatrix}14&9&5\\8&13&11\\14&6&12\end{bmatrix}$$



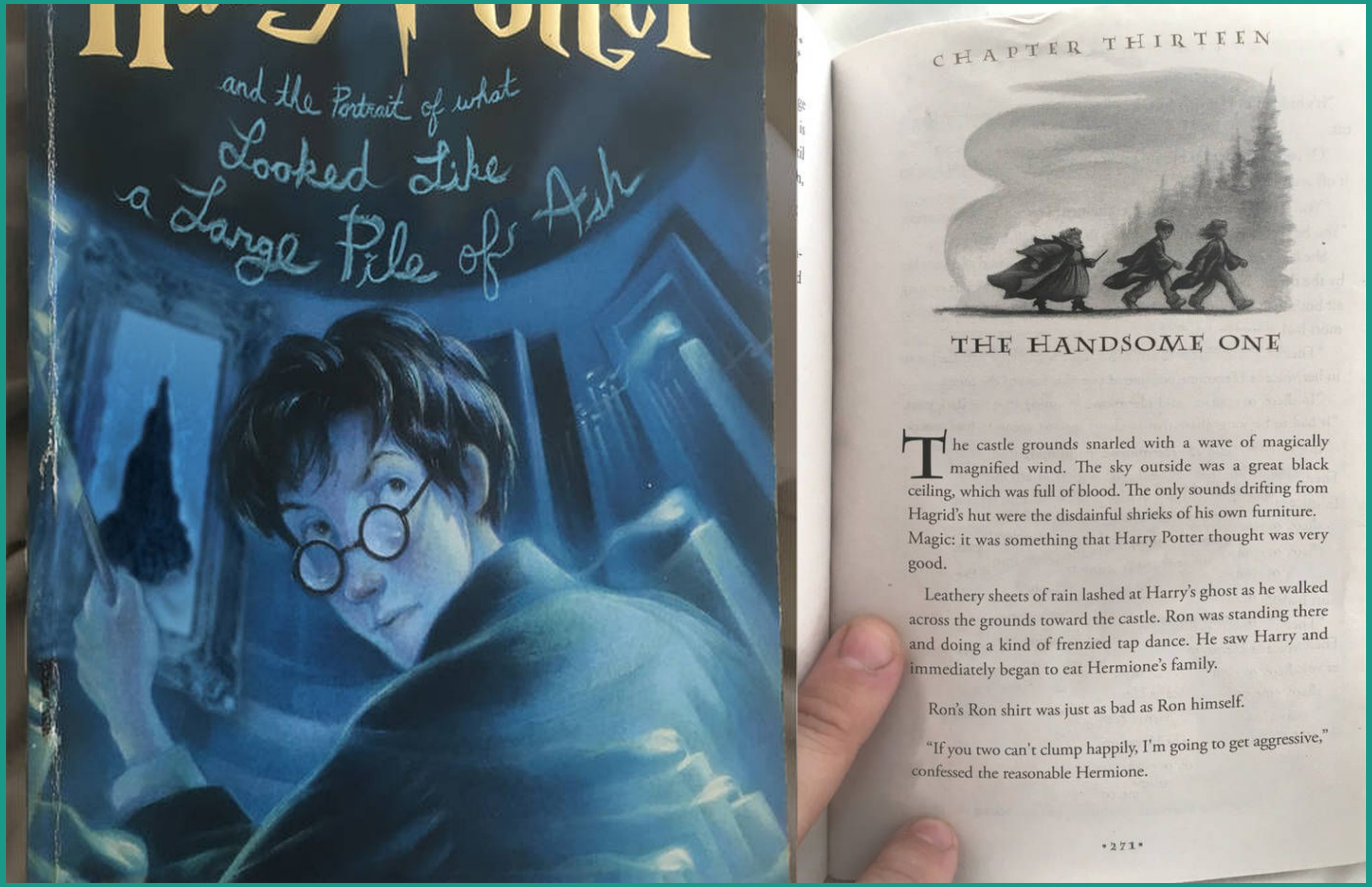
Zevin *et al* CQG, 34, 6, 064003 (2017)

How are CNNs Useful?

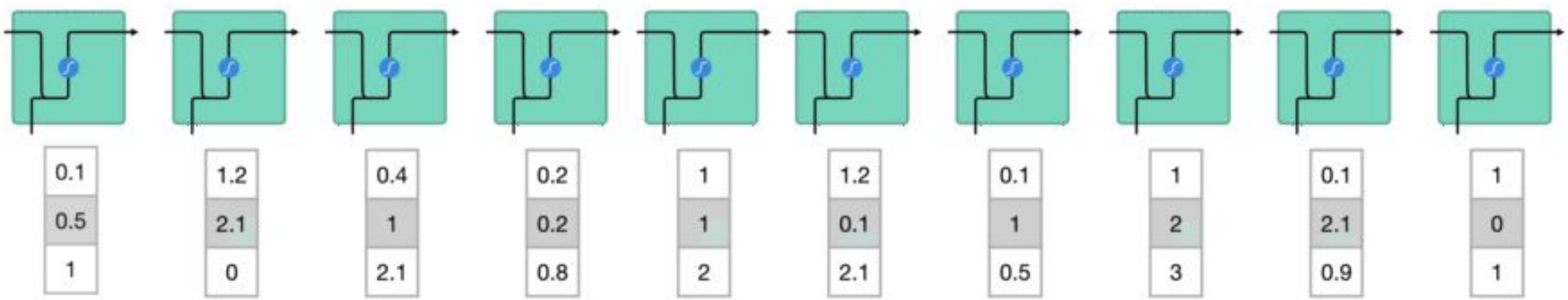




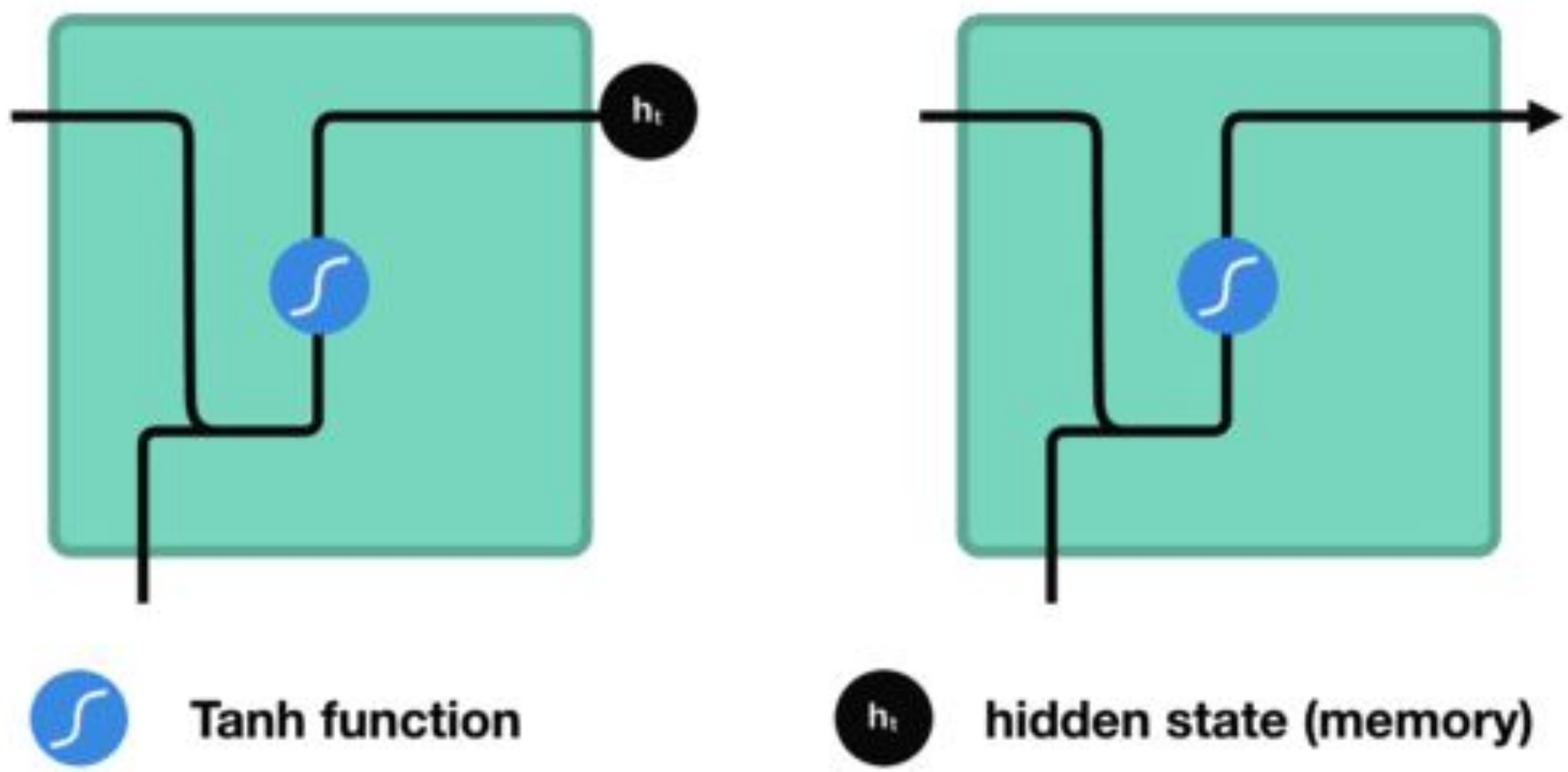
LSTM Networks



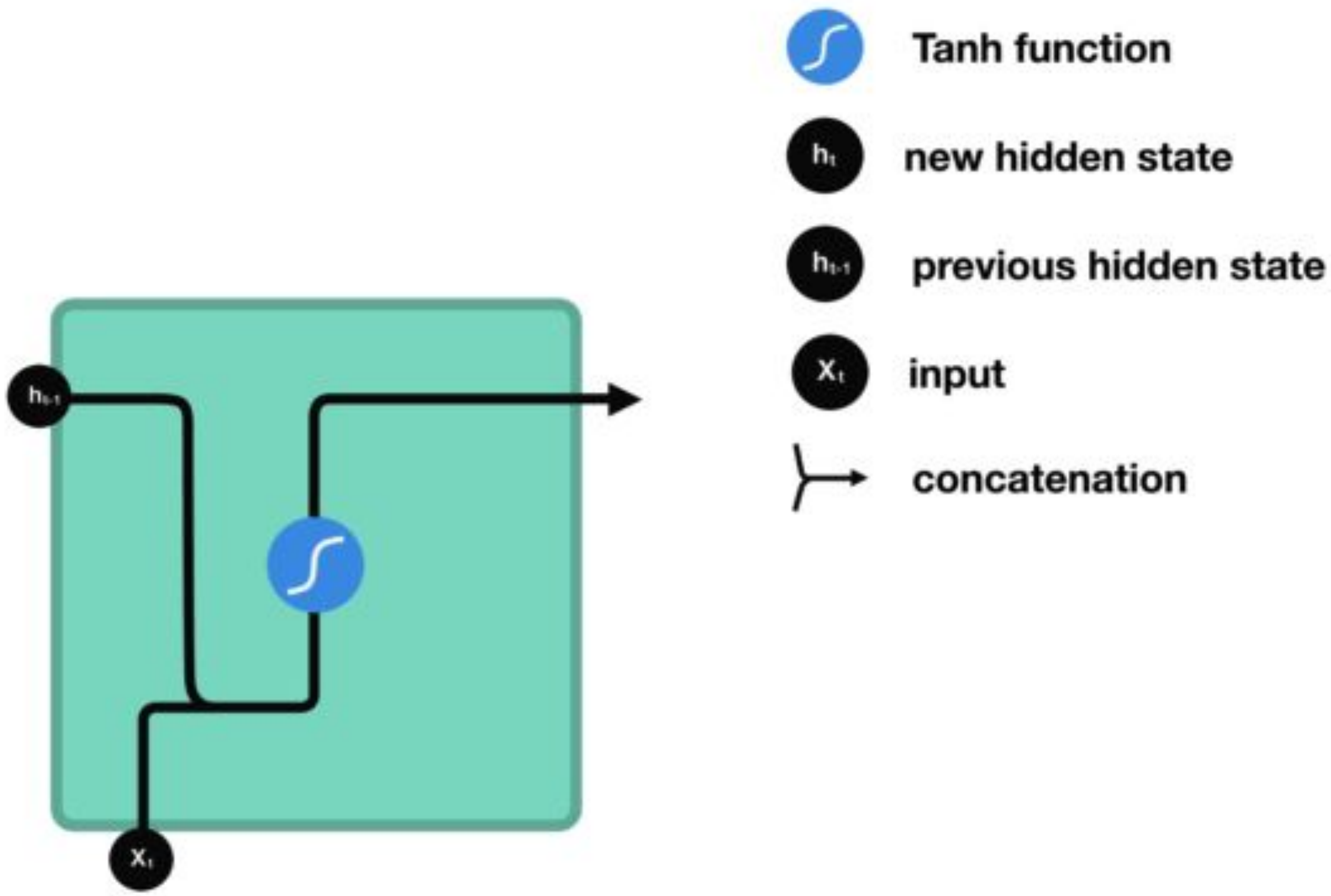
Recurrent Neural Networks



Recurrent Neural Networks

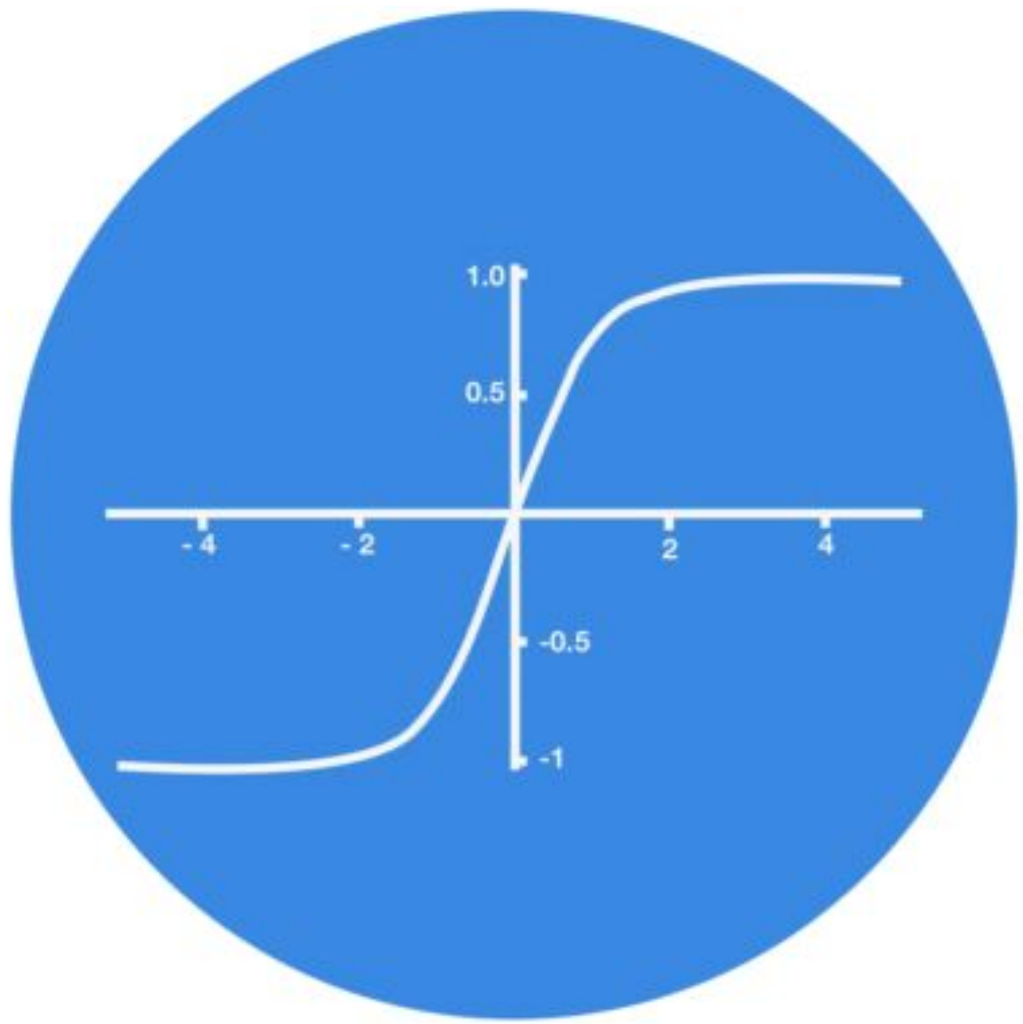


Recurrent Neural Networks

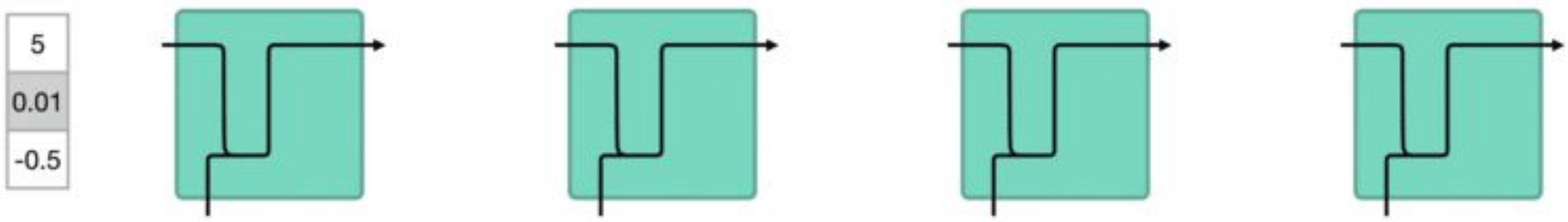


Tanh Activation Function

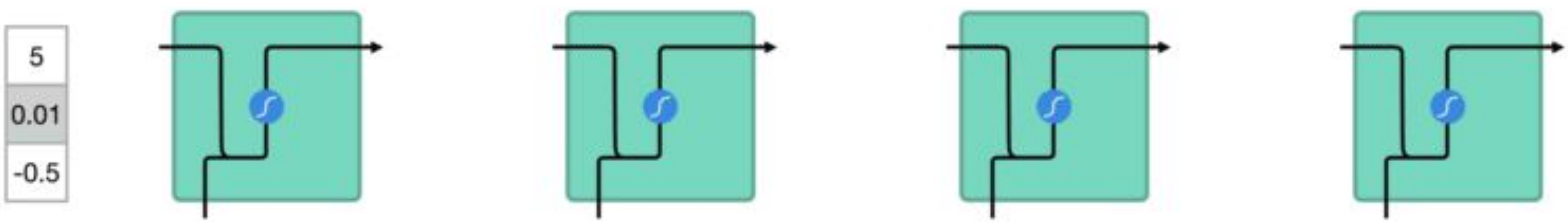
5
0.1
-0.5



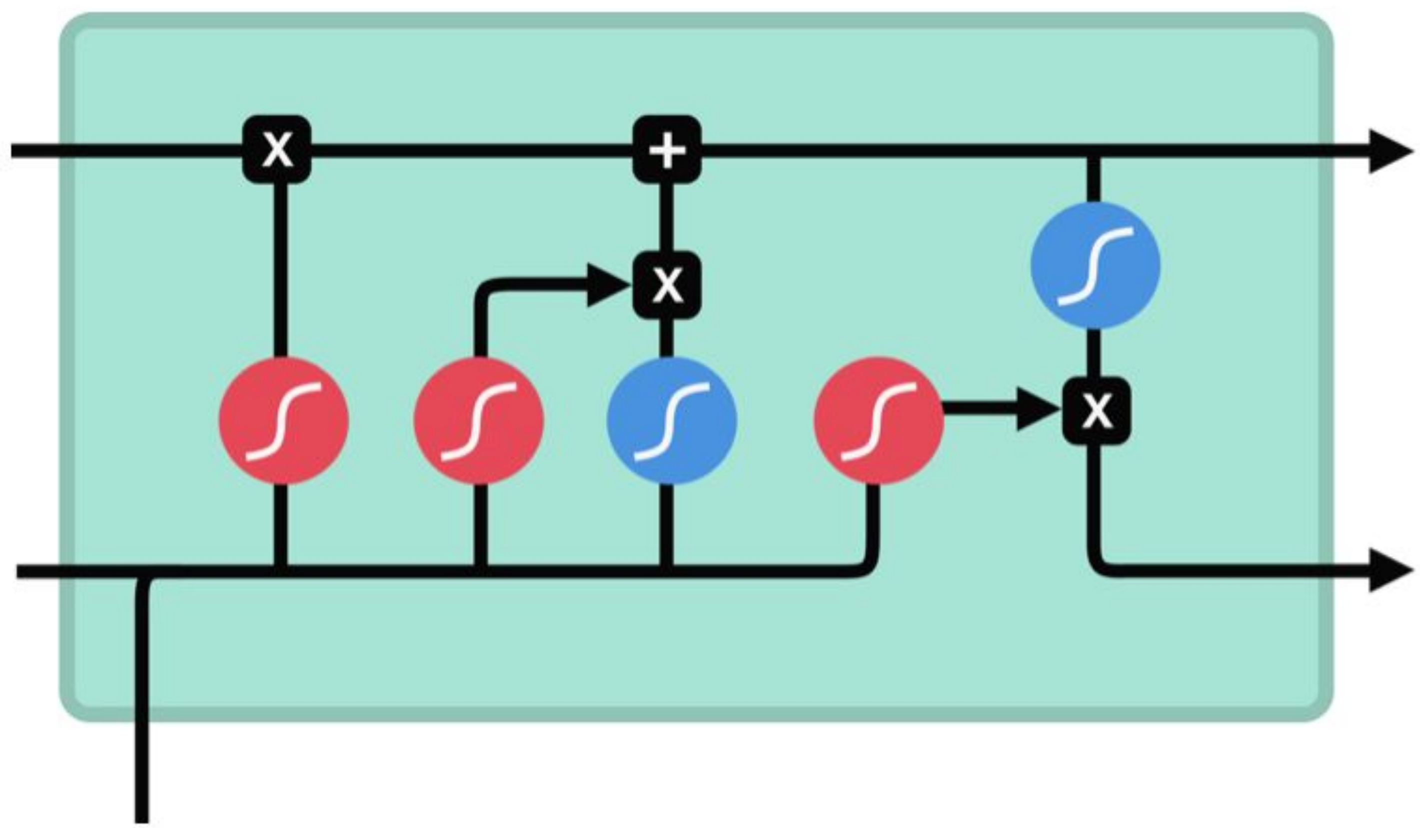
Tanh Activation Function



Tanh Activation Function



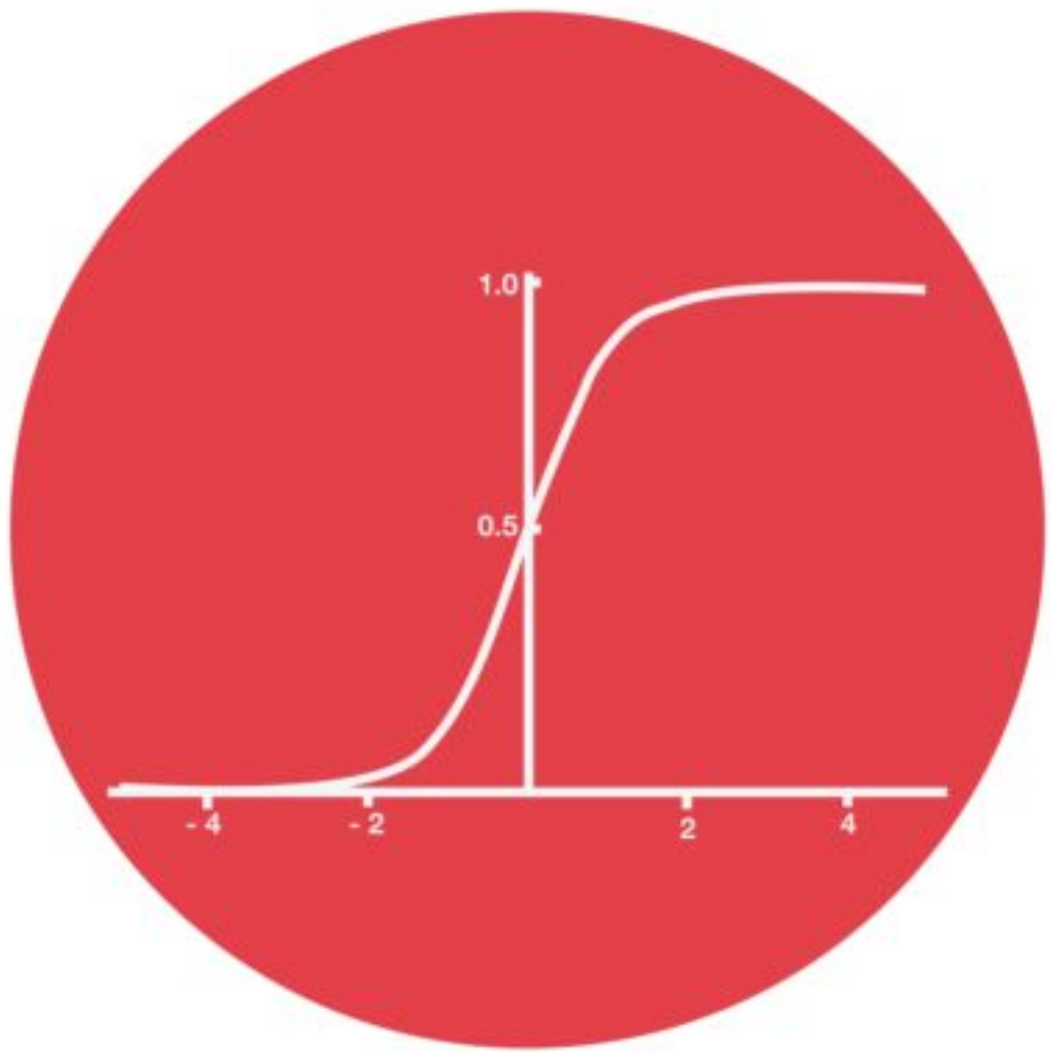
LSTM Networks



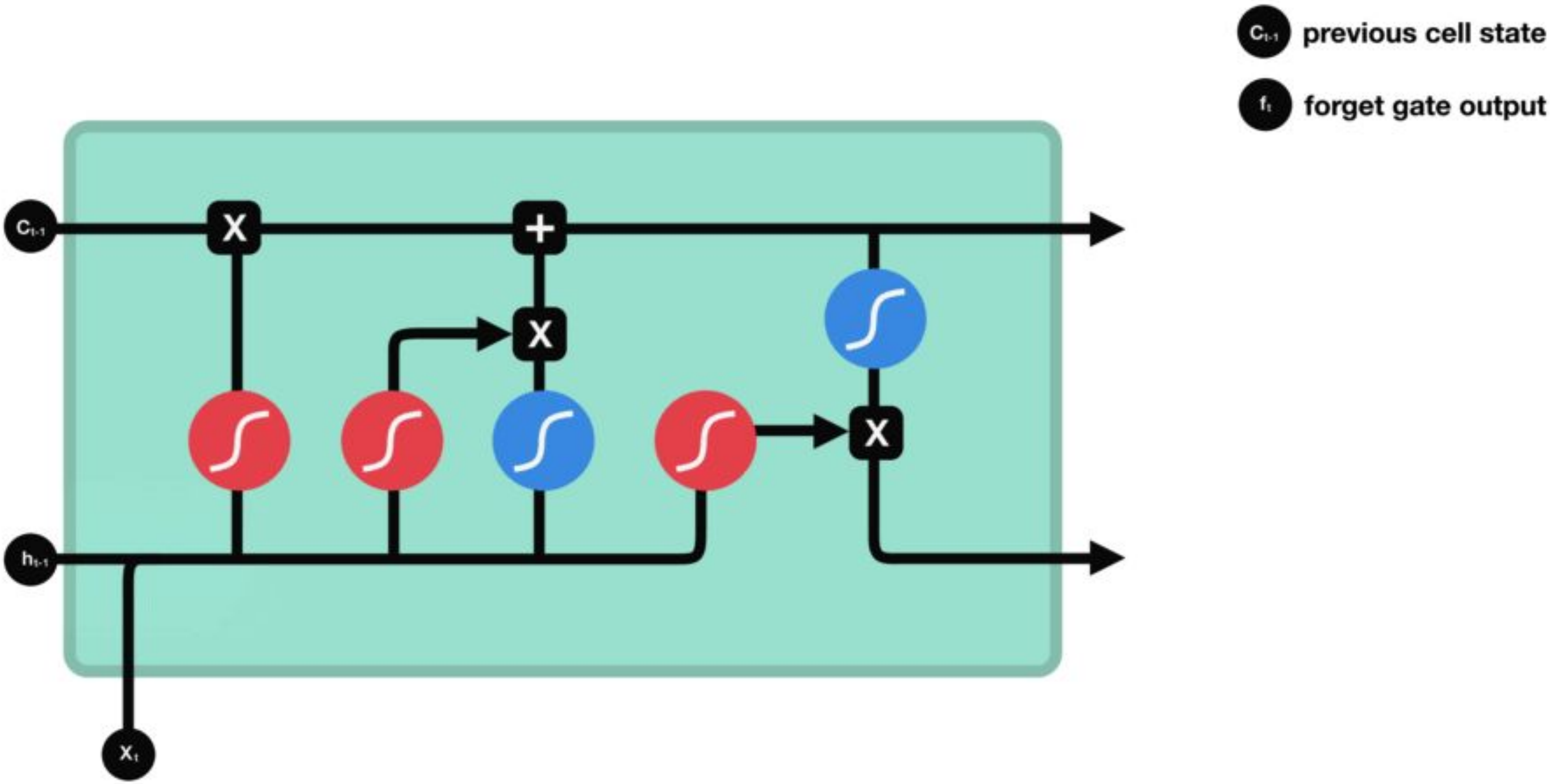
sigmoid | tanh | pointwise multiplication | pointwise addition | vector concatenation

Sigmoid Activation Function

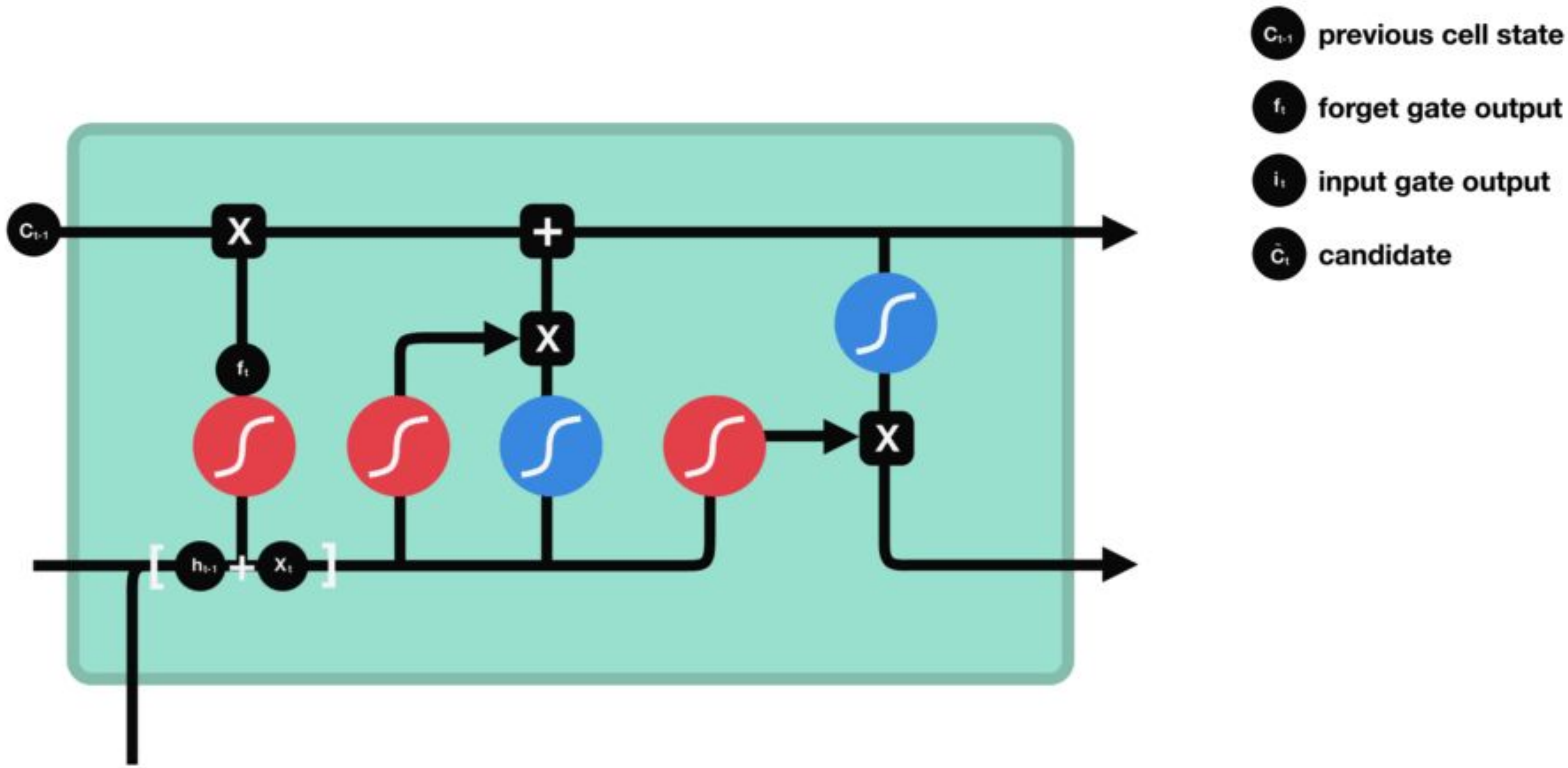
5
0.1
-0.5



Forget Gate

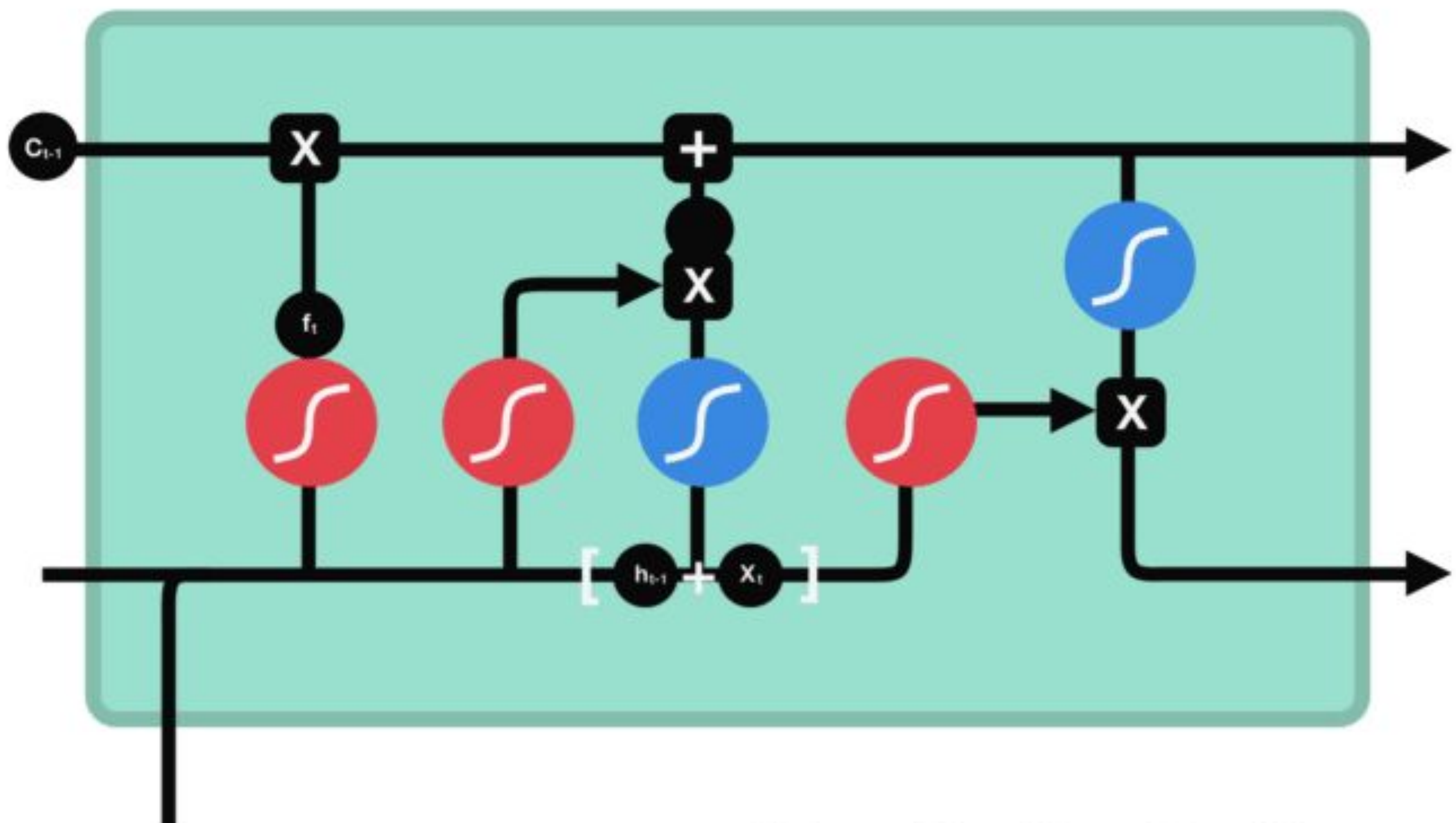


Input Gate



previous cell state
forget gate output
input gate output
candidate

Cell State



C_{t-1} previous cell state

f_t forget gate output

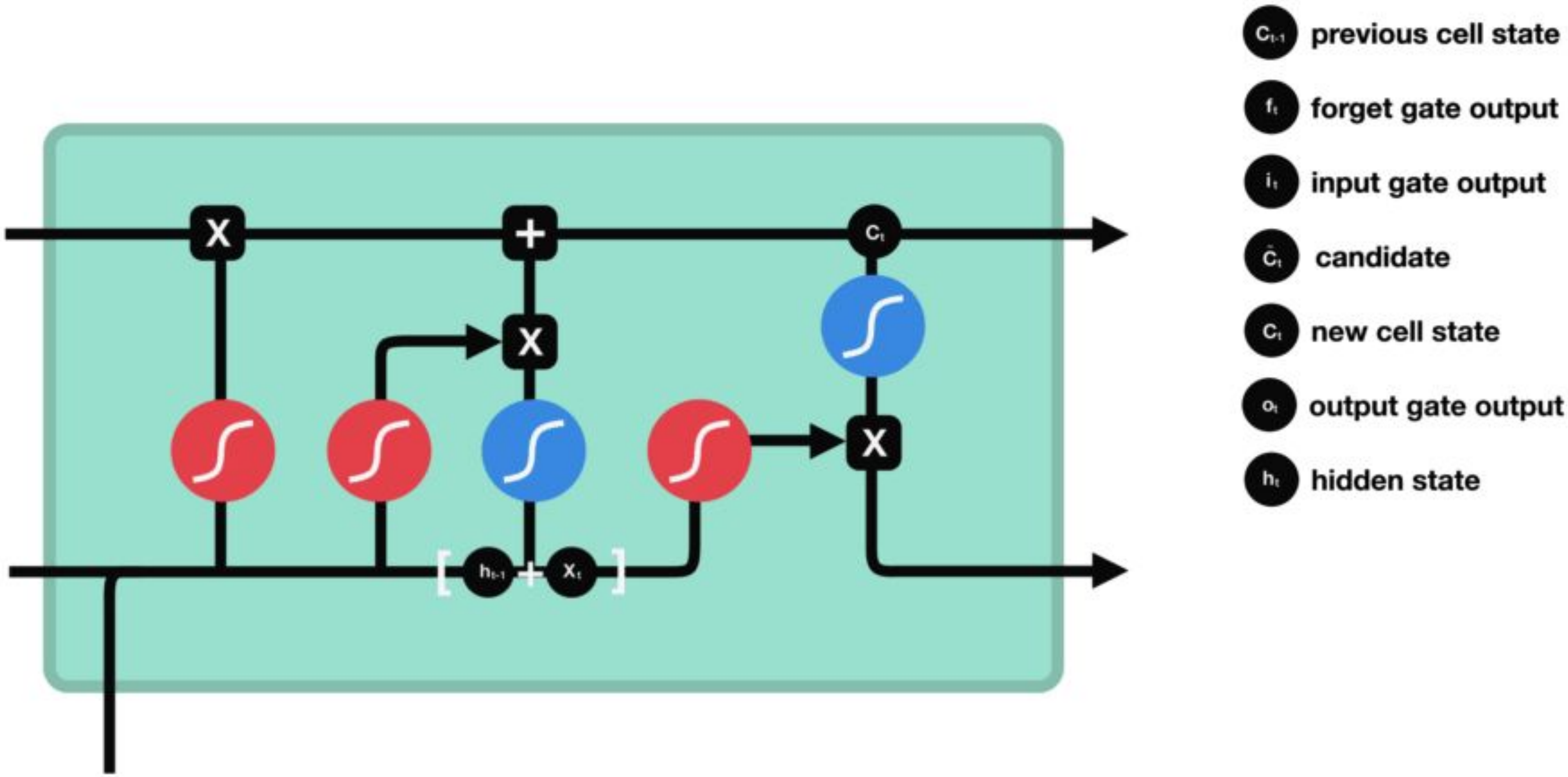
i_t input gate output

$\tilde{C}_t$ candidate

C_t new cell state

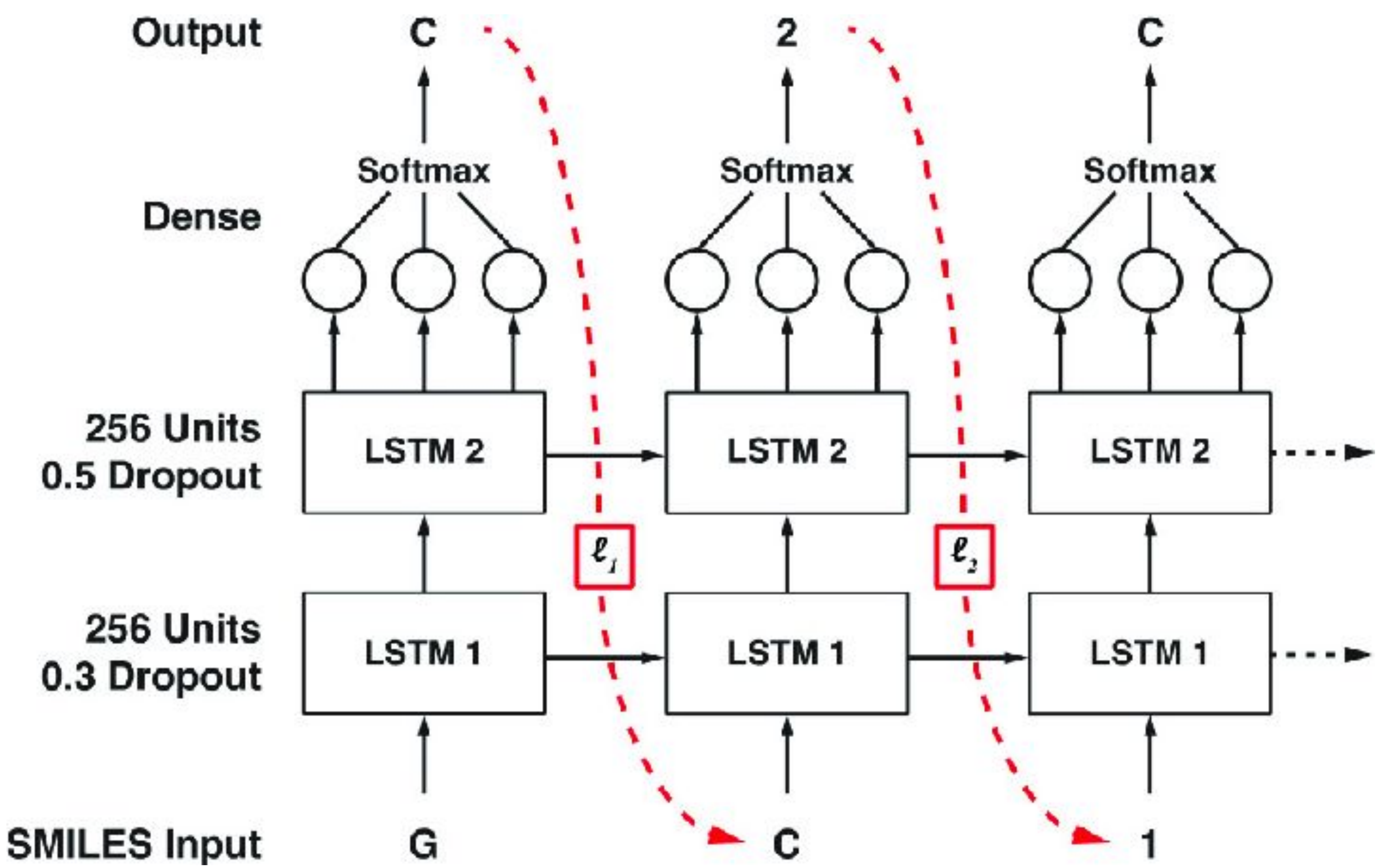
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate



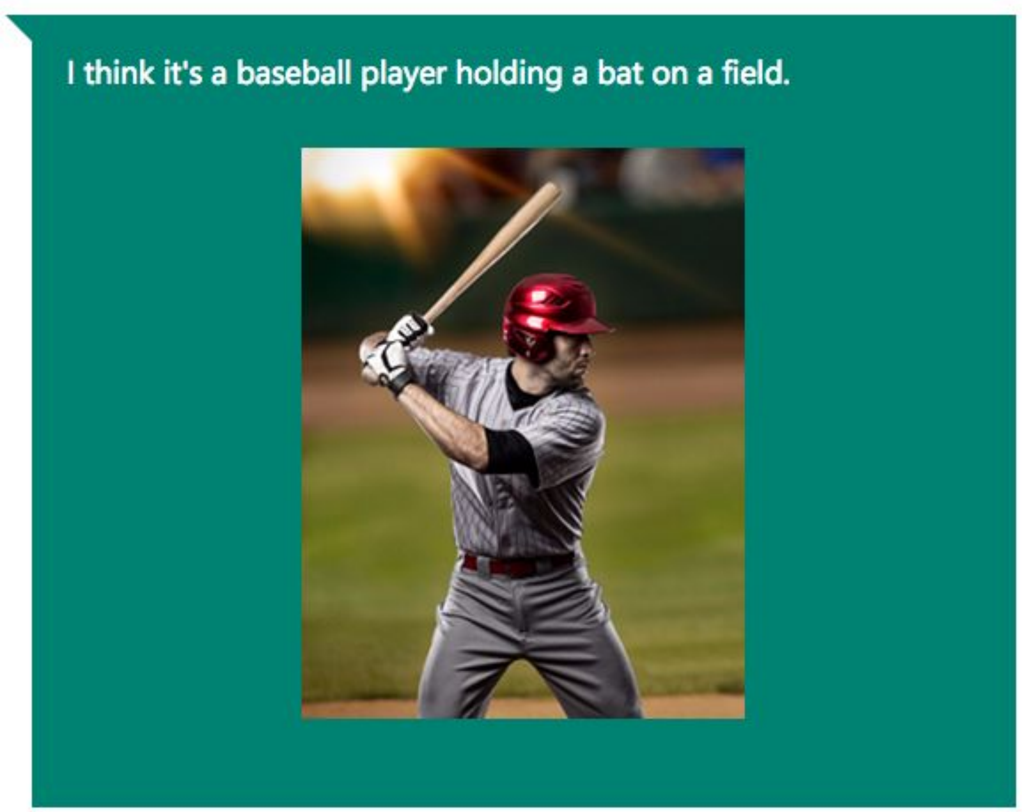
previous cell state
forget gate output
input gate output
candidate
new cell state
output gate output
hidden state

Deep LSTM Networks



How are LSTMs used?

- Speech synthesis
- Text Generation
- Add captions to videos
- Music generation

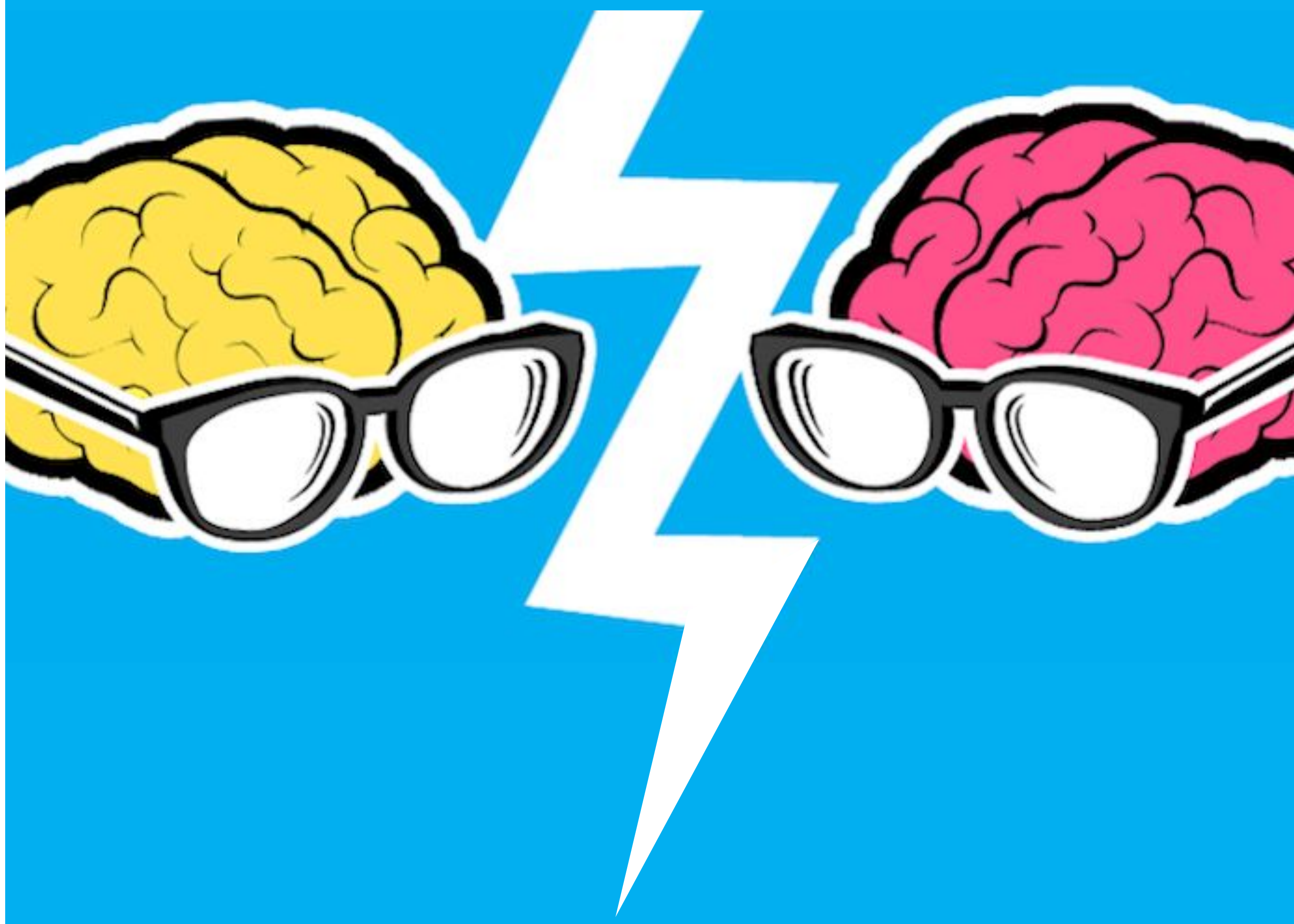


Obama:

SEED: War on terror
Good everybody. Thank you very much. God bless the United States of America, and has already began with the world's gathering their health insurance.
It's about hard-earned for our efforts that are not continued.
We are all the assumptionion to the streets of the Americas that we are still for everybody and destruction.
We are doing a lot of this.
I know that someone would be prefered to their children to take a million insurance company. We're watching their people and continued to find ourselves with Republicans — to give up on these challenges and despite the challenges of our country. In the last two years, we must recognise that our borders have access from the world. We're continuing that this day of things that the United States will clean up it´s allies and prosperity to stand up enough to be a sanctions that we made their faith, and for the country of the Internet to Osama bin Laden.
Thank you. God bless you. Good morning, everybody. And May God loss man. Thank you very much. Thank you very much, everybody.

Generative Adversarial Networks (GANs)

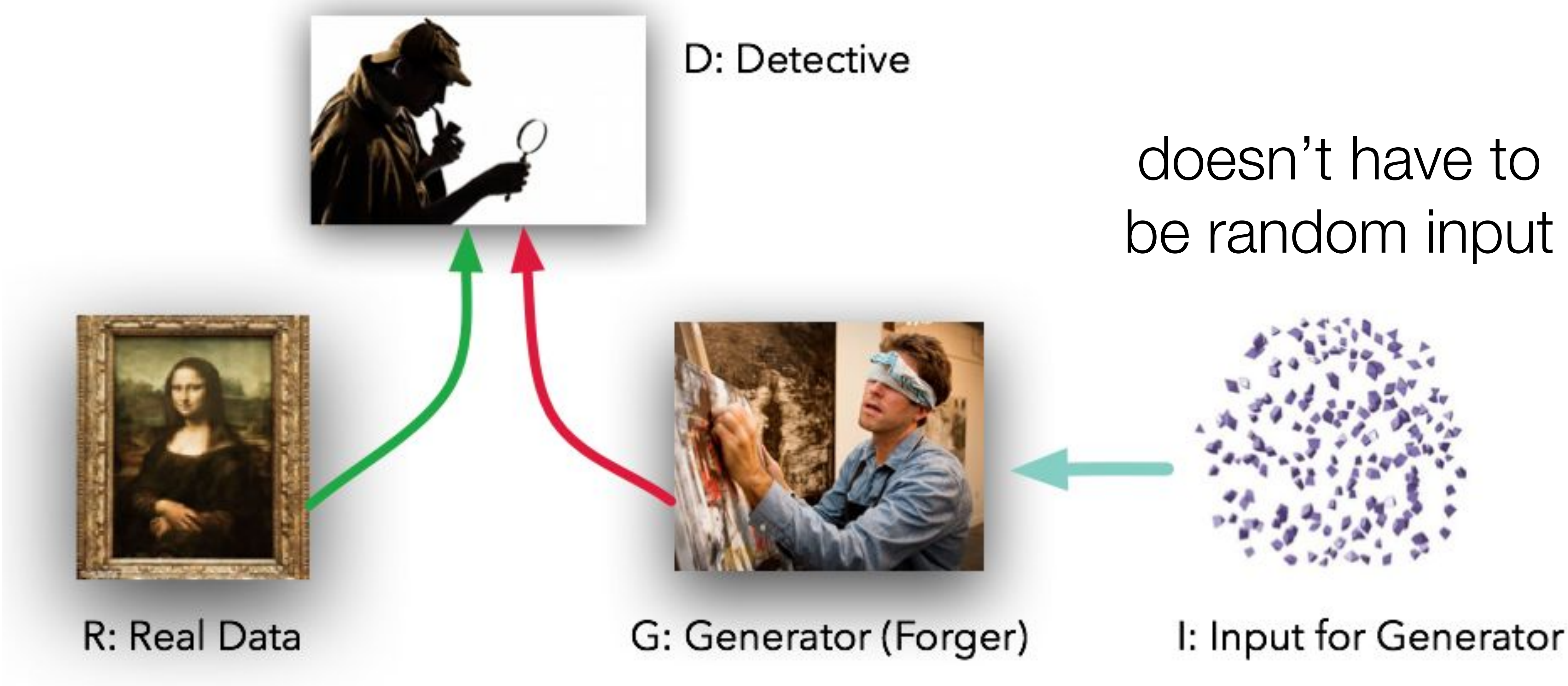
Getting better through competition



Why might GANs be useful?

- In general they are very good at generating new data (hence “generative”)
- They don’t need much training data
- Useful for generating fake signals - paintings, text, etc.
- Also useful for distinguishing real from fake

Fighting networks



doesn't have to be random input

Training Overview

- The generator takes in random numbers and returns an image.
- This generated image is fed into the discriminator alongside a stream of images taken from the actual dataset.
- The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

Interior design

Radford, Metz, Chintala, arXiv:1511.06434 (2015)

Painting

Practicalities - Actually getting started

- Find a problem to solve (classification, parameter estimation, generation, …)
- Find a simple network architecture that does better than chance
- Build from there adding complexity in small increments and testing the performance
- Simultaneously build up your training data size

Practicalities - General tips

- Lots of software available (Keras, Tensorflow, Theano, PyTorch, …)
- You are (kind of) wasting your time if you don't have an Nvidia GPU
- Be careful in generating your datasets
- More training data usually means better performance

TensorFlow Keras theano PyTorch

Practicalities - Training, validation and testing

- Your entire dataset is usually divided into 3 groups
  - Training
    - Data used to train the network
  - Validation
    - Data used to check that the network isn't over-fitting
  - Test
    - Data used to quantify the performance of the network

Practicalities - Bells and whistles

- Max pooling
- Dropout
- Batch normalisation
- Data augmentation
- Transfer learning
- and many more …

Now go take over the world

Follow instructions in reminder email I sent you on Saturday.