

# Deep learning basics

G. Châtel

3 mars 2017

# Machine learning

## Supervised learning

Machine learning is a subfield of artificial intelligence.

**Intuitively** We want to *learn from* and *make predictions on* data.

**Technically** We want to build a model that approximate well (e.g. minimize a loss function) an unknown function.

It is important to note that the function we want to approximate may or may not have a closed form.

# Application examples

## Supervised learning

- Regression

Polynomial  $(x, y, z) \rightarrow f(x, y, z)$

House price (surface, nb rooms, city)  $\rightarrow$  price

- Classification

Image classification pixel values  $\rightarrow$  cat or dog

Text classification list of words  $\rightarrow$  spam or valid email

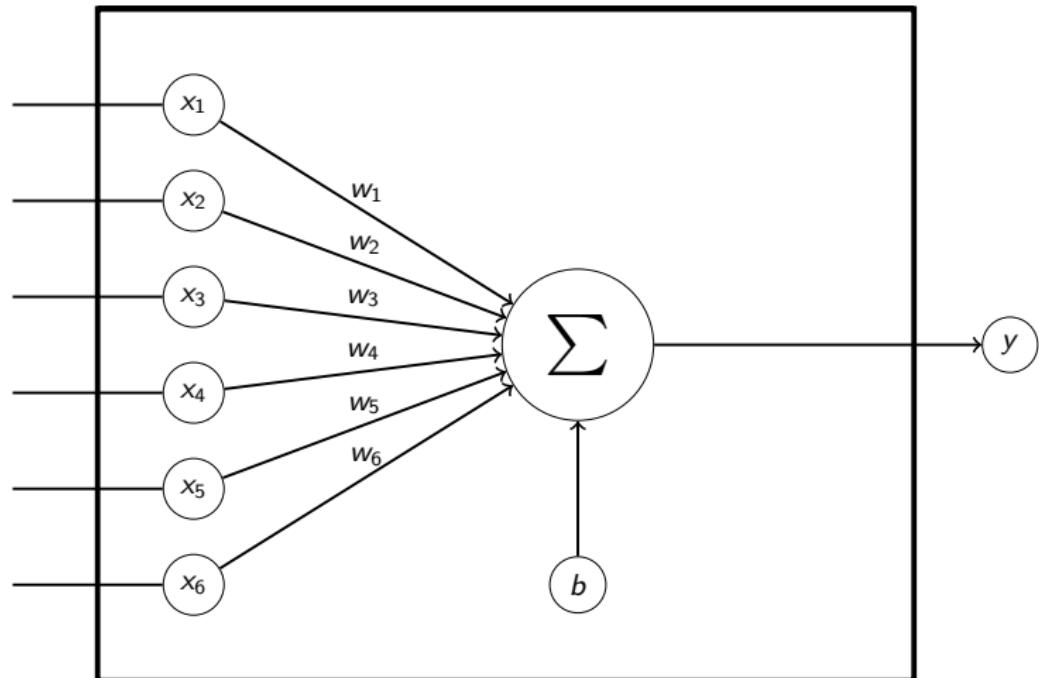
# Deep learning

Deep learning is a subfield of machine learning in which we use artificial neural networks to make predictions.

An artificial neural networks is a computation model loosely based on the human brain. It aims to mimic electric signals travelling through neurons in order to make computations.

# Artificial neural network

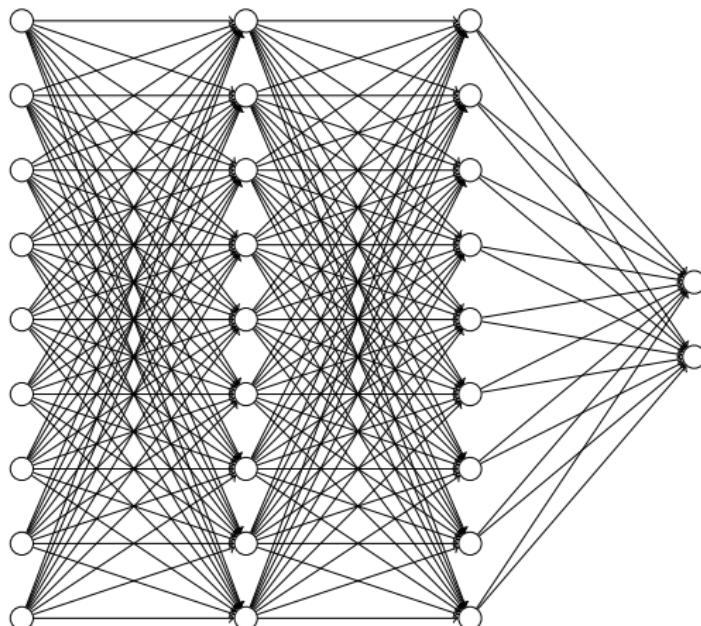
## Neuron



$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + b$$

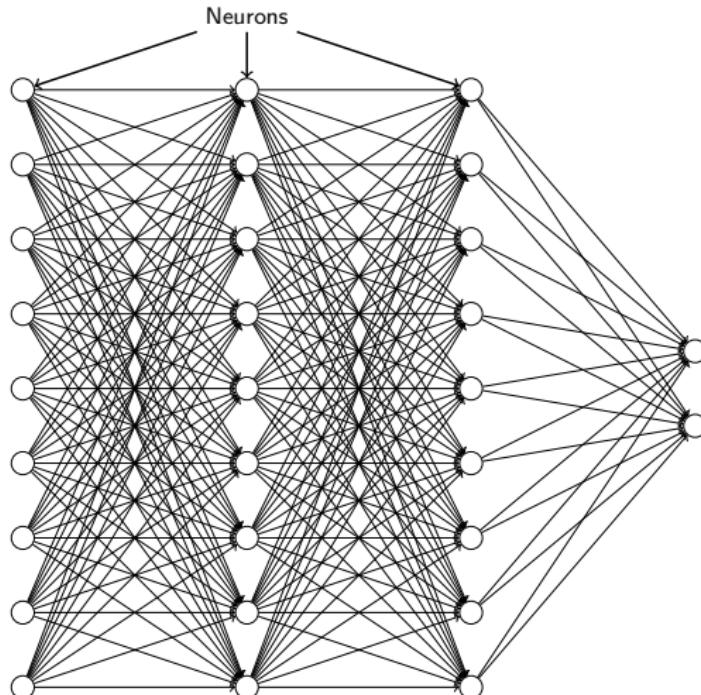
# Artificial neural network

## Network



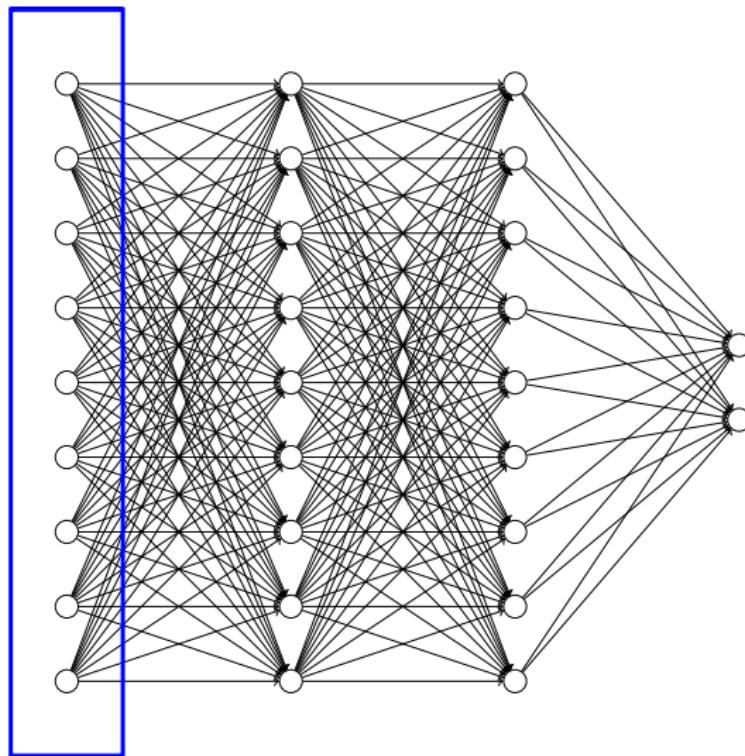
# Artificial neural network

## Network



# Artificial neural network

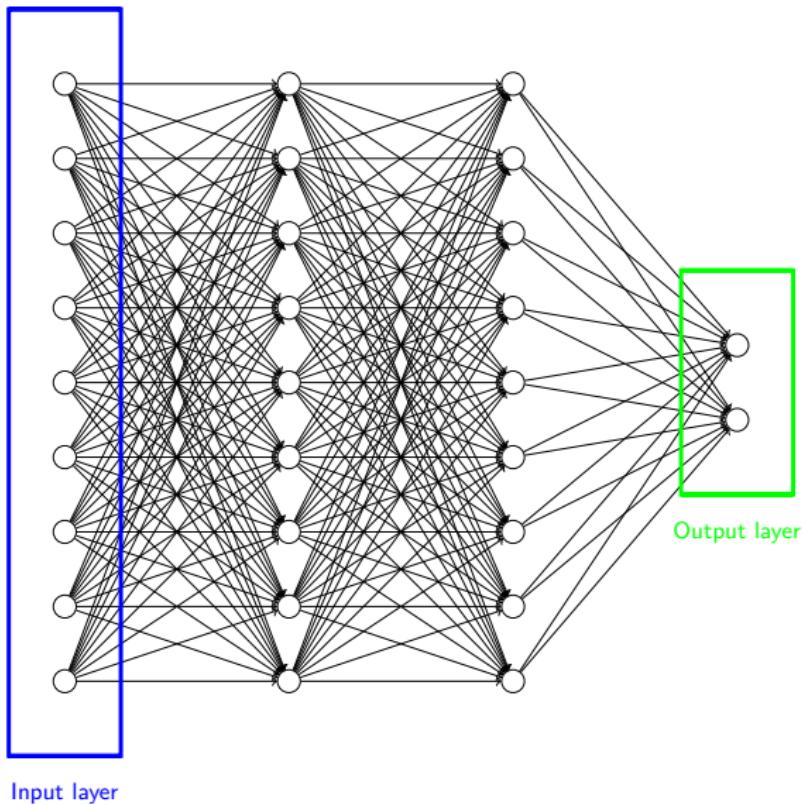
## Network



Input layer

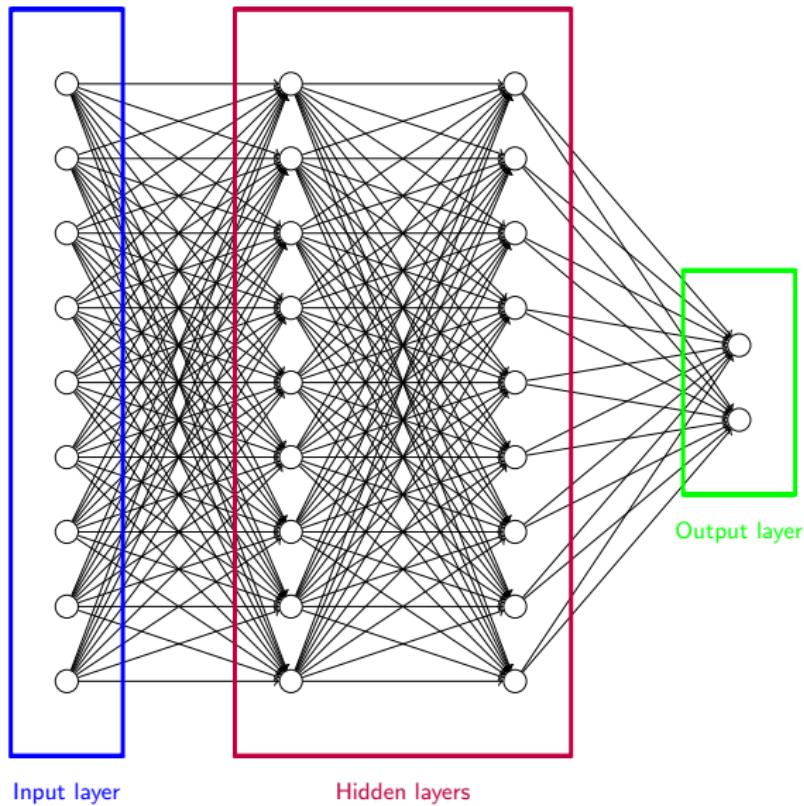
## Artificial neural network

## Network



## Artificial neural network

## Network



## “Problem” with this definition

We now have a quite complicated framework to compute [linear functions](#).

Neuron	linear function
Neural network	linear combination of neuron outputs

To approximate non linear functions, we would like to have a non linear model.

## “Problem” with this definition

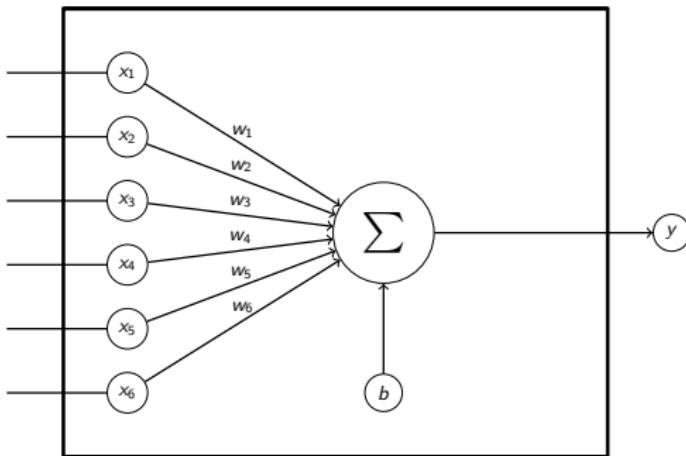
We now have a quite complicated framework to compute [linear functions](#).

Neuron	linear function
Neural network	linear combination of neuron outputs

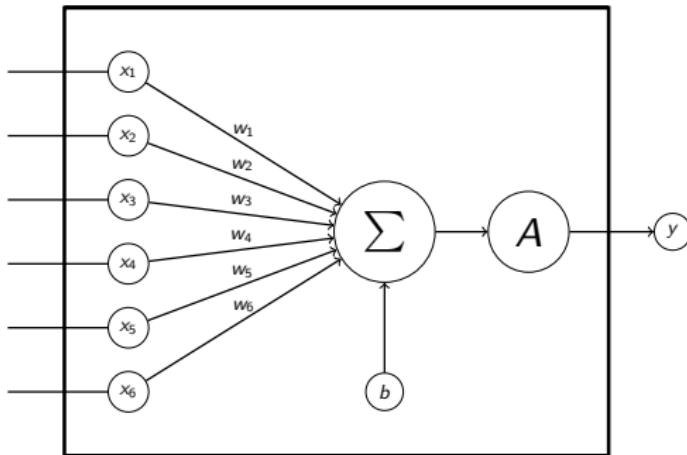
To approximate non linear functions, we would like to have a non linear model.

[Solution](#): add nonlinearity to neurons.

## Neuron with activation

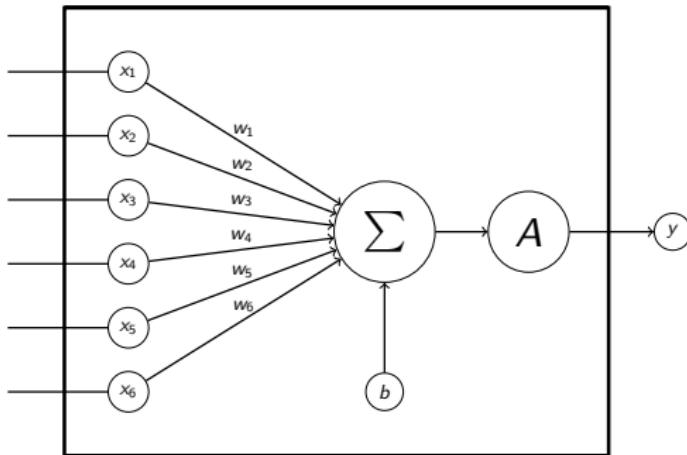


## Neuron with activation



$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

## Neuron with activation

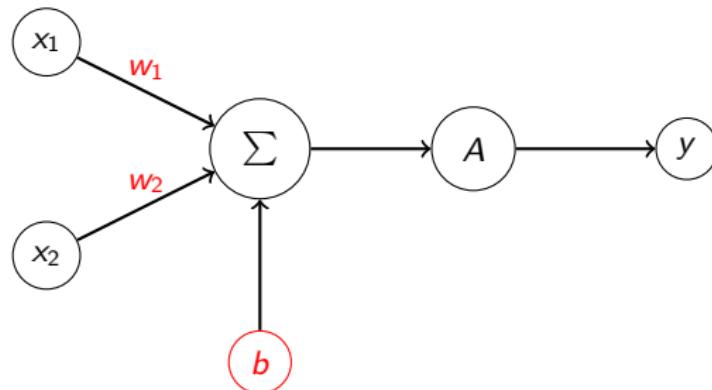


$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$y = A(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6 + b)$$

## Computation example

Binary AND gate

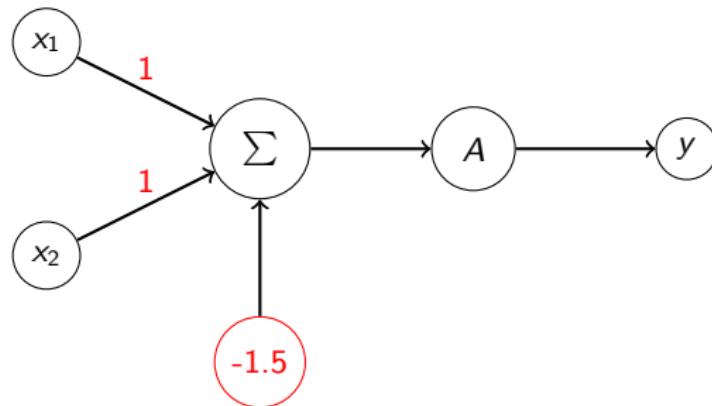


We want to set  $w_1, w_2$  and  $b$  such that:

$$A(w_1x_1 + w_2x_2 + b) = x_1 \text{ AND } x_2$$

## Computation example

Binary AND gate

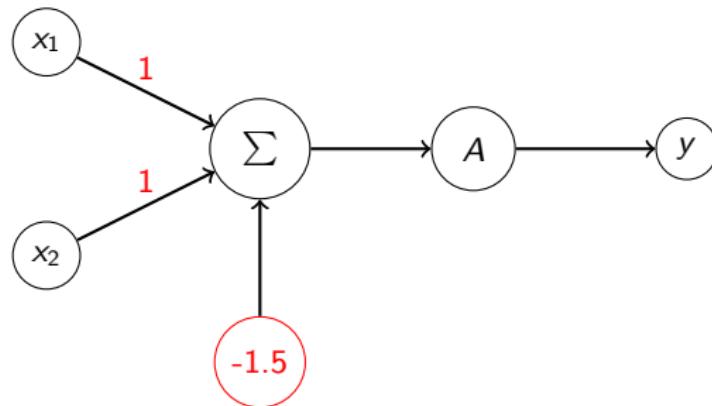


We want to set  $w_1, w_2$  and  $b$  such that:

$$A(w_1x_1 + w_2x_2 + b) = x_1 \text{ AND } x_2$$

## Computation example

Binary AND gate



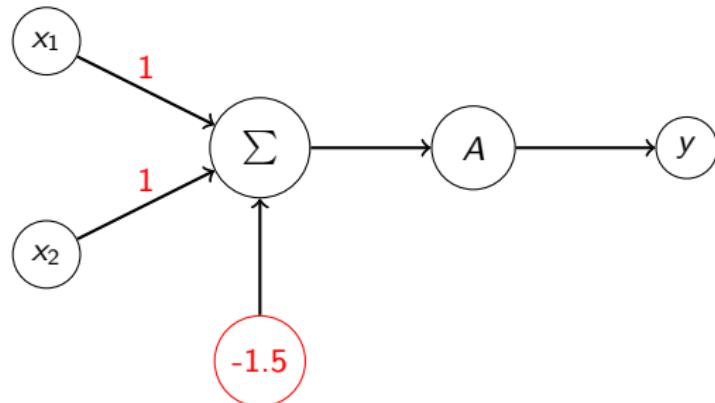
We want to set  $w_1, w_2$  and  $b$  such that:

$$A(w_1x_1 + w_2x_2 + b) = x_1 \text{ AND } x_2$$

$$x_0 = 0, x_1 = 1. \quad y = A(0 + 1 - 1.5) = A(-0.5) = 0$$

## Computation example

Binary AND gate



We want to set  $w_1, w_2$  and  $b$  such that:

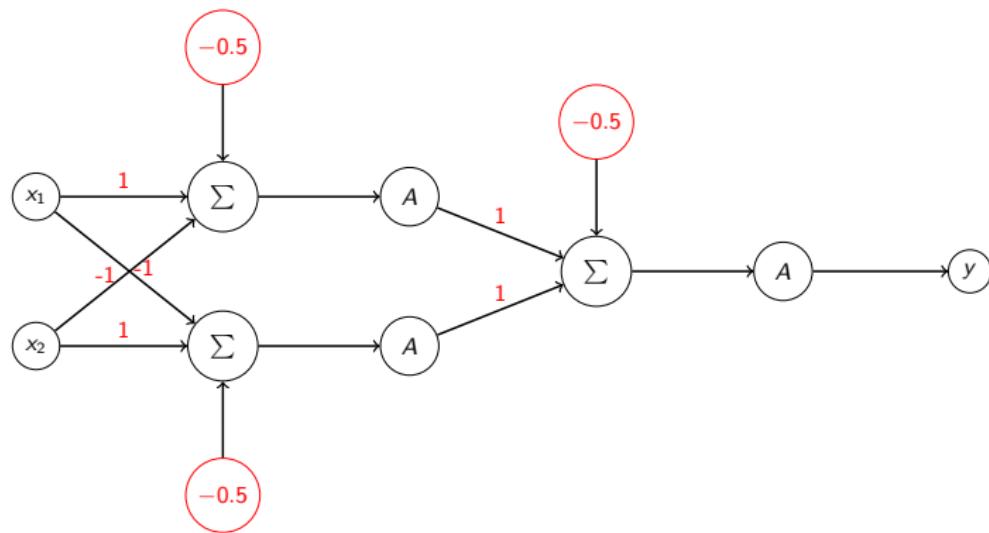
$$A(w_1x_1 + w_2x_2 + b) = x_1 \text{ AND } x_2$$

$$x_0 = 0, x_1 = 1. \quad y = A(0 + 1 - 1.5) = A(-0.5) = 0$$

$$x_0 = 1, x_1 = 1. \quad y = A(1 + 1 - 1.5) = A(0.5) = 1$$

# Computation example

Binary XOR gate



## Model complexity

One way to measure the complexity of a neural network is its number of parameters.

## Model complexity

One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters

## Model complexity

One way to measure the complexity of a neural network is its number of parameters.

- XOR network: 9 parameters
- dogs vs cats pictures (VGG16 network): 138,357,544 parameters

## Model complexity

One way to measure the complexity of a neural network is its number of parameters.

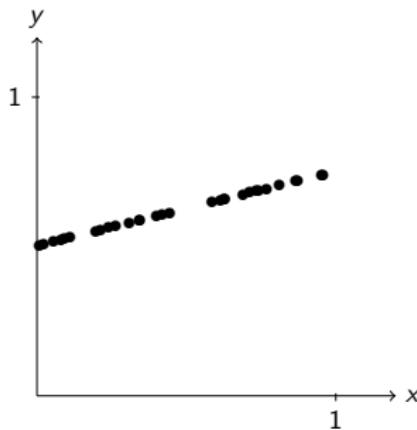
- XOR network: 9 parameters
- dogs vs cats pictures (VGG16 network): 138,357,544 parameters

We should probably look for a way to tune these parameters automatically otherwise it is going to get *really* boring *really* quickly.

## Parameter tuning

### Line approximation

We have a few sample points and we want to find a line that approximate these points.



Our model is just a line

$$y_{pred} = ax + b$$

We want to find  $a$  and  $b$  to best match our samples.

## Parameter tuning

### Gradient descent

First we have to define a loss that measures how good our predictions are.

$$l(x, y, a, b) = (y - y_{pred})^2 = (y - (ax + b))^2$$

and now, we compute how the loss is affected by small changes of  $a$  and  $b$ :

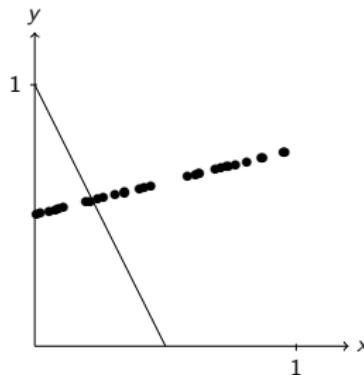
$$\frac{dl}{da} = 2x(ax + b - y)$$

$$\frac{dl}{db} = 2(ax + b - y)$$

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

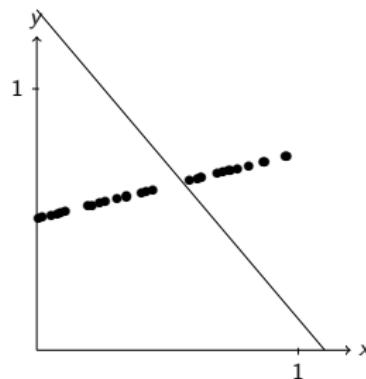
$$a = -2.00 \quad b = 1.00 \quad \overline{\frac{dl}{da}} = -1.07 \quad \overline{\frac{dl}{db}} = -1.37 \quad \overline{l(x, y, a, b)} = 0.860367$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

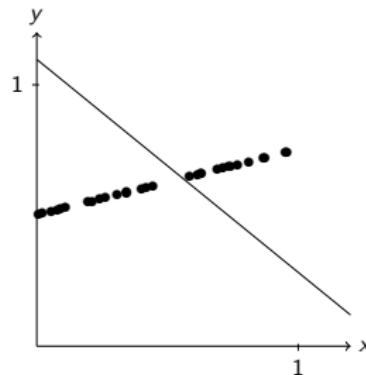
$$a = -1.18 \quad b = 1.30 \quad \overline{\frac{dl}{da}} = -0.17 \quad \overline{\frac{dl}{db}} = 0.09 \quad \overline{l(x, y, a, b)} = 0.159420$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

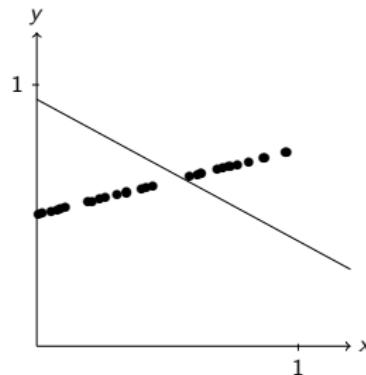
$$a = -0.82 \quad b = 1.10 \quad \overline{\frac{dl}{da}} = -0.13 \quad \overline{\frac{dl}{db}} = 0.07 \quad \overline{l(x, y, a, b)} = 0.088204$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

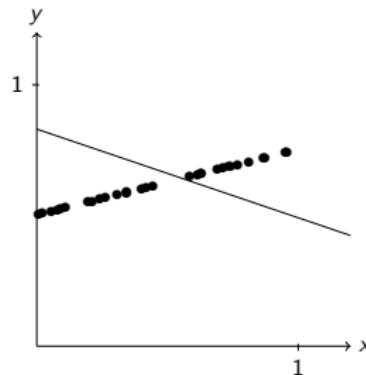
$$a = -0.54 \quad b = 0.94 \quad \overline{\frac{dl}{da}} = -0.09 \quad \overline{\frac{dl}{db}} = 0.05 \quad l(x, y, a, b) = 0.048802$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

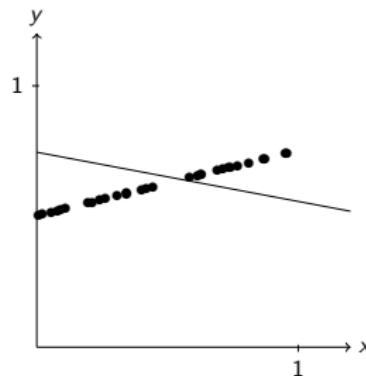
$$a = -0.34 \quad b = 0.83 \quad \overline{\frac{dl}{da}} = -0.07 \quad \overline{\frac{dl}{db}} = 0.04 \quad l(x, y, a, b) = 0.027001$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

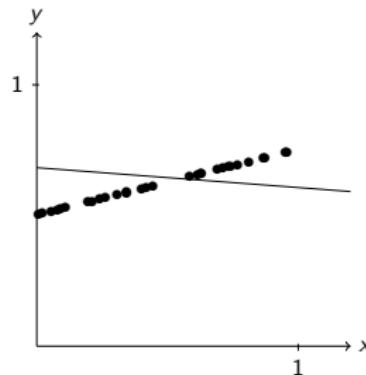
$$a = -0.19 \quad b = 0.75 \quad \overline{\frac{dl}{da}} = -0.05 \quad \overline{\frac{dl}{db}} = 0.03 \quad l(x, y, a, b) = 0.014939$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

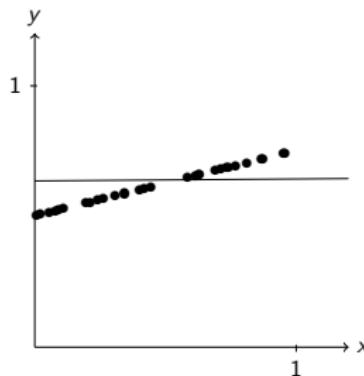
$$a = -0.08 \quad b = 0.68 \quad \overline{\frac{dl}{da}} = -0.04 \quad \overline{\frac{dl}{db}} = 0.02 \quad \overline{l(x, y, a, b)} = 0.008266$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

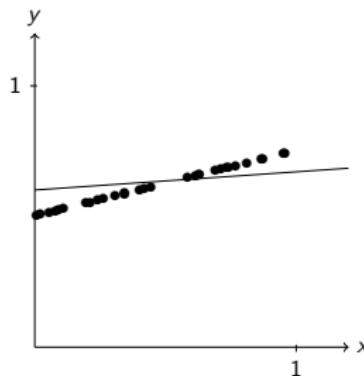
$$a = 0.01 \quad b = 0.64 \quad \overline{\frac{dl}{da}} = -0.03 \quad \overline{\frac{dl}{db}} = 0.02 \quad \overline{l(x, y, a, b)} = 0.004573$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

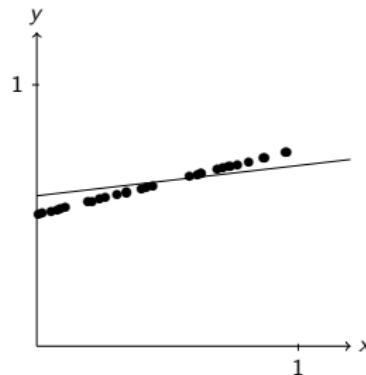
$$a = 0.07 \quad b = 0.60 \quad \overline{\frac{dl}{da}} = -0.02 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.002530$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

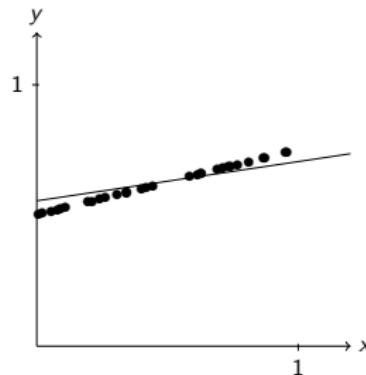
$$a = 0.12 \quad b = 0.58 \quad \overline{\frac{dl}{da}} = -0.02 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.001400$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

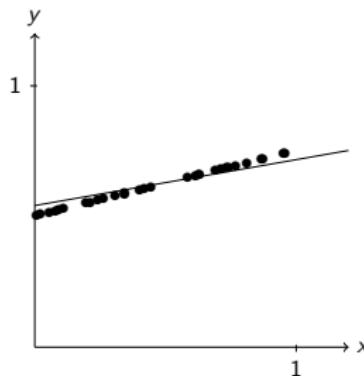
$$a = 0.15 \quad b = 0.56 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.01 \quad \overline{l(x, y, a, b)} = 0.000775$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

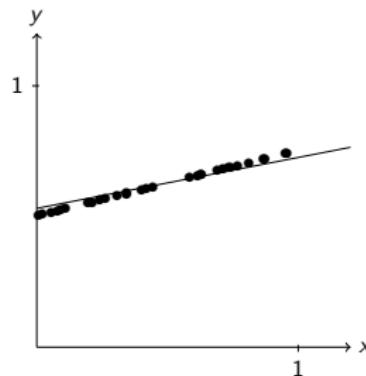
$$a = 0.18 \quad b = 0.54 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000429$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

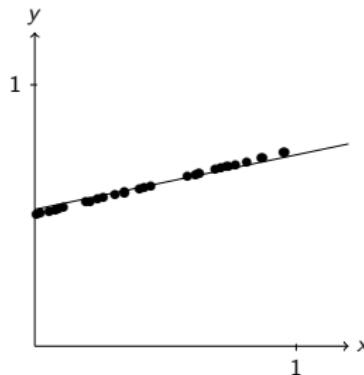
$$a = 0.19 \quad b = 0.53 \quad \overline{\frac{dl}{da}} = -0.01 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000237$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

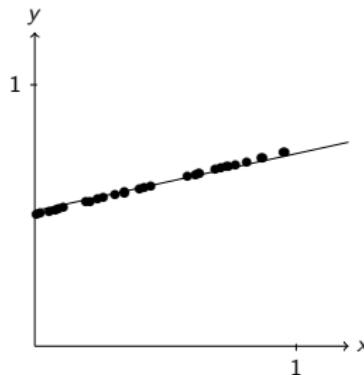
$$a = 0.21 \quad b = 0.52 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000131$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

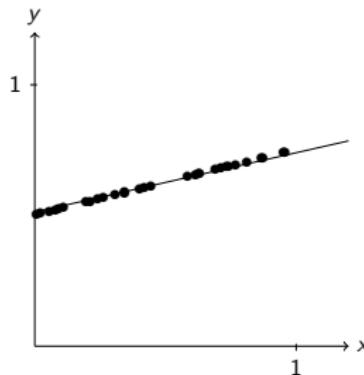
$$a = 0.22 \quad b = 0.52 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000073$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

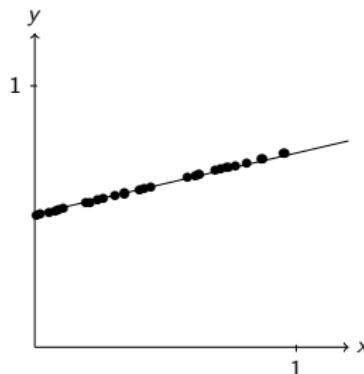
$$a = 0.23 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000040$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

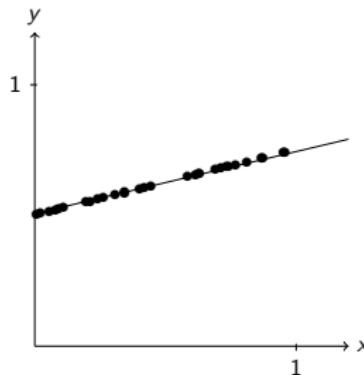
$$a = 0.23 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000022$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

# Parameter tuning

## Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

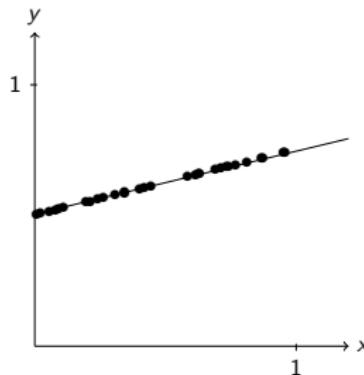
$$a = 0.24 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000012$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

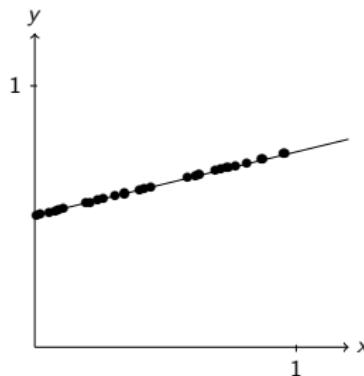
$$a = 0.24 \quad b = 0.51 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000007$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Gradient descent

We start with random values:  $a = -2$  and  $b = 1$



Then, we compute the average of the gradient along all  $(x, y)$  couples

$$a = 0.24 \quad b = 0.50 \quad \overline{\frac{dl}{da}} = 0.00 \quad \overline{\frac{dl}{db}} = 0.00 \quad \overline{l(x, y, a, b)} = 0.000004$$

And update  $a$  and  $b$  by subtracting a small proportion of their gradient.

## Parameter tuning

### Problem with ANN

Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

## Parameter tuning

### Problem with ANN

Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

**Problem:** The function computed by our neural network is not differentiable because of the activation function.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

## Parameter tuning

### Problem with ANN

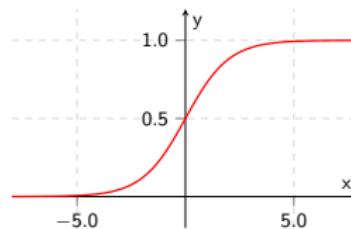
Using gradient descent, we know how to minimize (or at least reach a local minima) a differentiable function.

**Problem:** The function computed by our neural network is not differentiable because of the activation function.

$$A(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

**Solution:** We replace it by a differentiable function that does the same job.

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



# Convolution

## Motivation

For now, our networks computes a nonlinear function of the inputs. If we work with images, it has to learn the *spacial structure* of the data by itself, which takes a long time.

A nice way to help it is to build **convolution layers** into the network.

# Convolution

## Kernel

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.

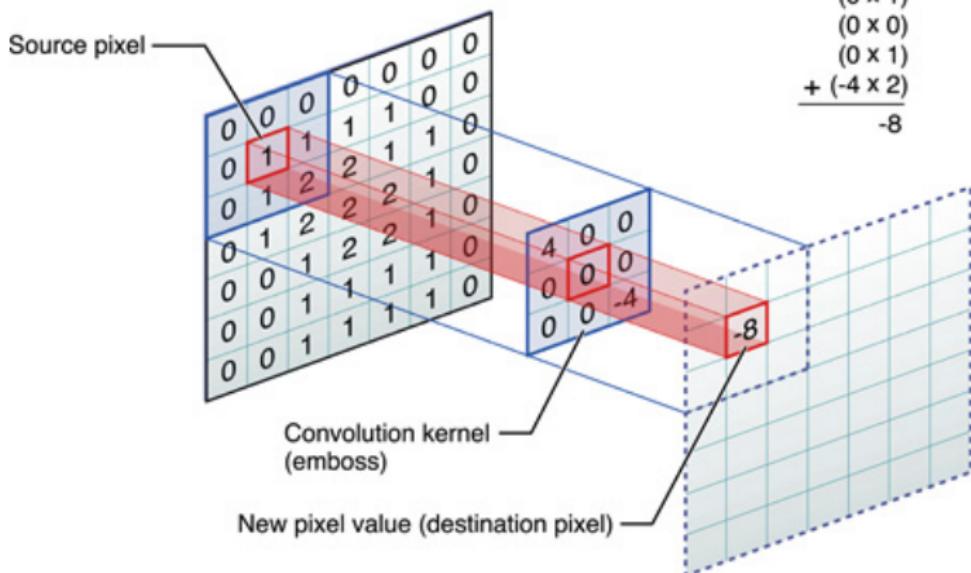
$$\begin{array}{r} (4 \times 0) \\ (0 \times 0) \\ (0 \times 0) \\ (0 \times 0) \\ (0 \times 1) \\ (0 \times 1) \\ (0 \times 0) \\ (0 \times 1) \\ + (-4 \times 2) \\ \hline -8 \end{array}$$


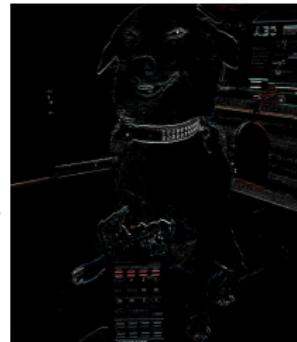
Image from <http://stats.stackexchange.com/>

# Convolution

Example: Sobel operators



$$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$



$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$



Image from <http://www.reddit.com/>

# Convolutional neural network

VGG network (2014)

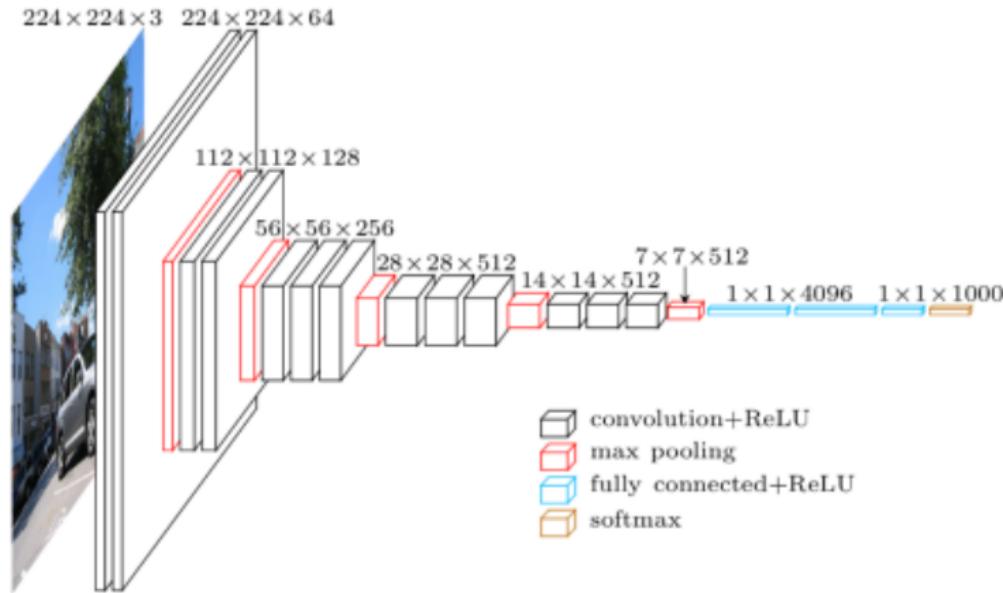


Image from <https://www.cs.toronto.edu/~frossard/post/vgg16/>

# Convolution

What do filter recognize?

Image from <http://www.matthewzeiler.com/pubs/arxive2013/eccv2014.pdf>



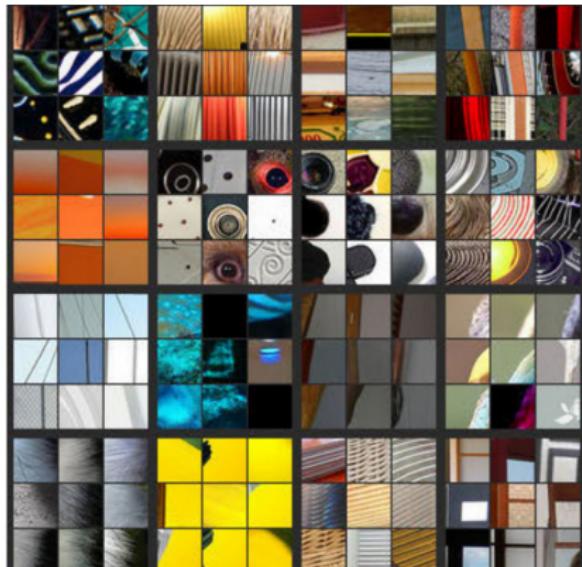
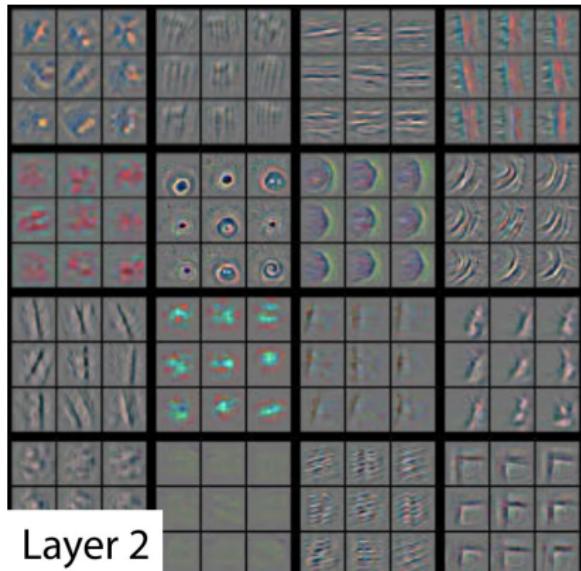
Layer 1



# Convolution

What do filter recognize?

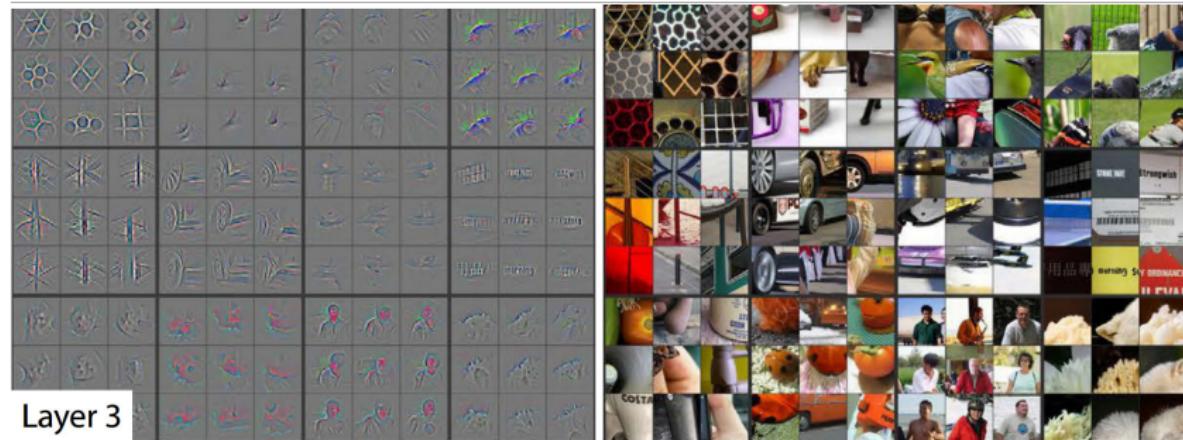
Image from <http://www.matthewzeiler.com/pubs/arxive2013/eccv2014.pdf>



## Convolution

## What do filter recognize?

Image from <http://www.matthewzeiler.com/pubs/arxive2013/eccv2014.pdf>



# Convolution

What do filter recognize?

Image from <http://www.matthewzeiler.com/pubs/arxive2013/eccv2014.pdf>

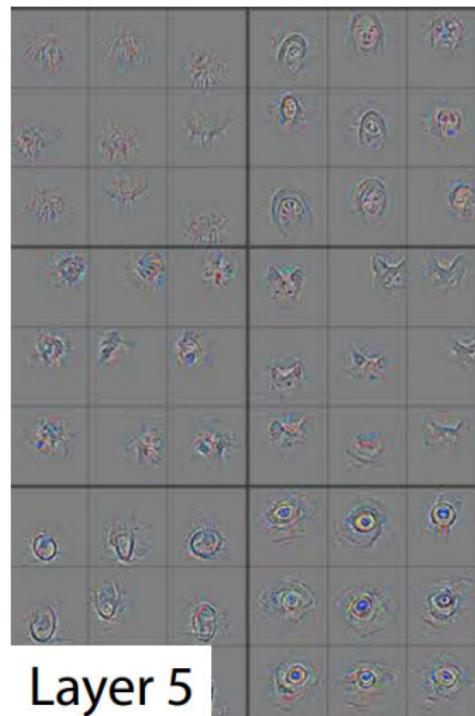


Layer 4

# Convolution

What do filter recognize?

Image from <http://www.matthewzeiler.com/pubs/arxive2013/eccv2014.pdf>



Layer 5

# Convolutional neural network

VGG network (2014)

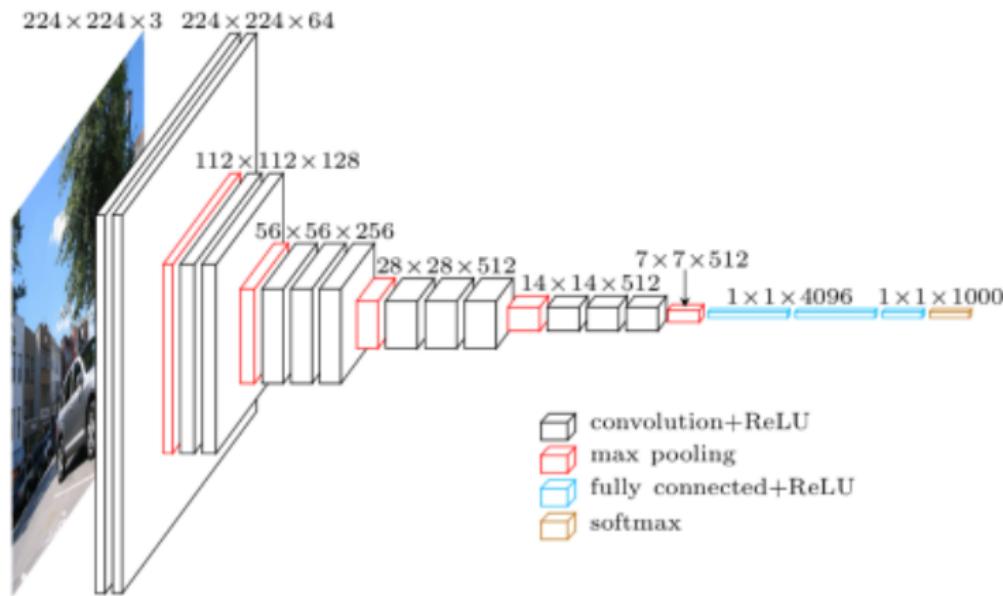
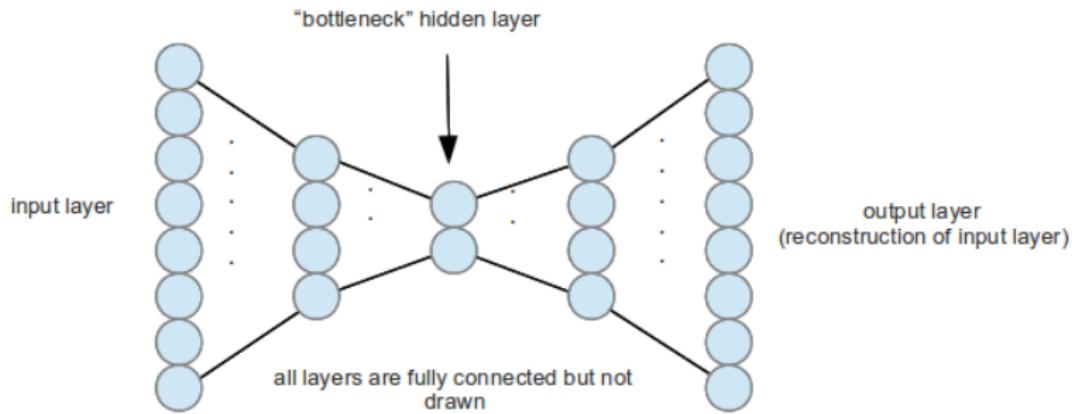


Image from <https://www.cs.toronto.edu/~frossard/post/vgg16/>

# Architecture examples

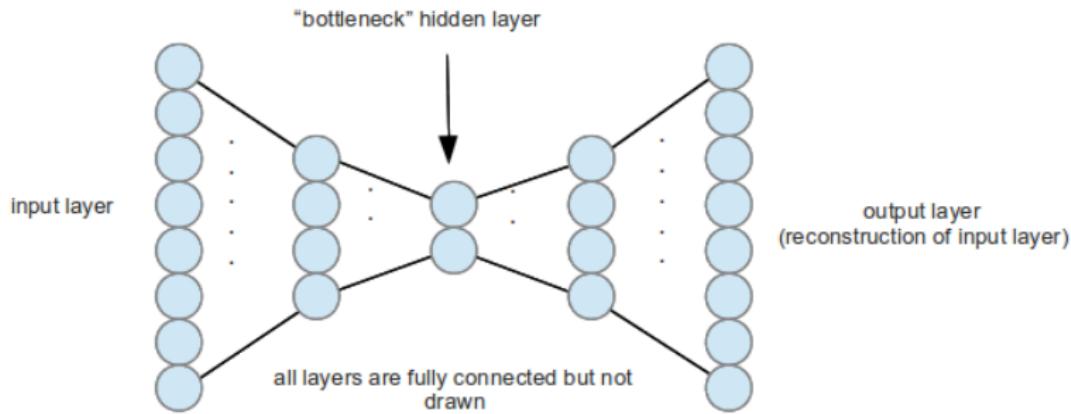
## Autoencoder: data encoding



Hinton, Salakhutdinov (2006)

# Architecture examples

## Autoencoder: data encoding

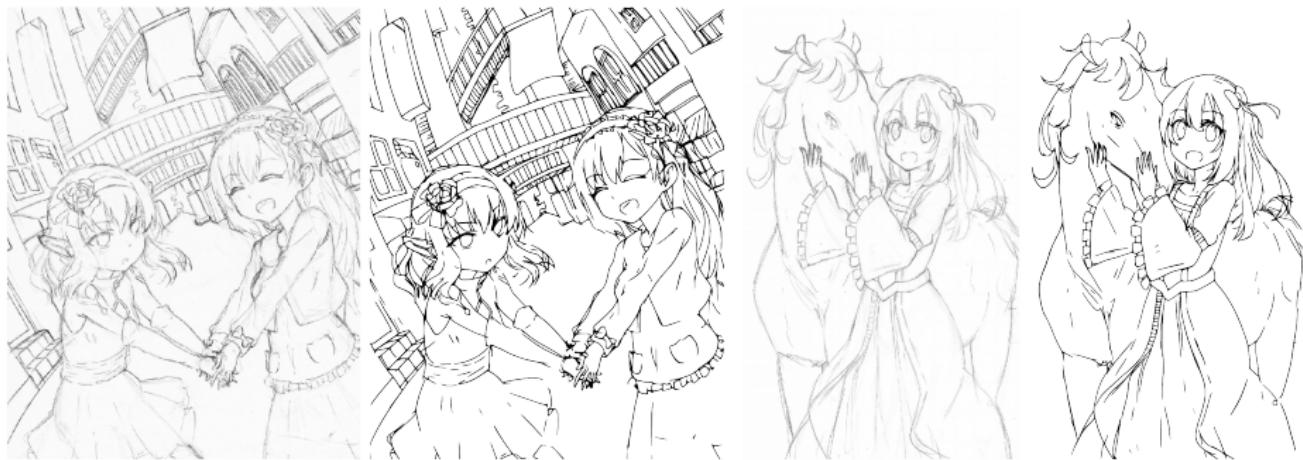


Hinton, Salakhutdinov (2006)

If we cut this autoencoder at the bottleneck, we get two parts: an encoder and a decoder. The encoder is an encoder highly specific to the content the network has been trained with.

# Architecture examples

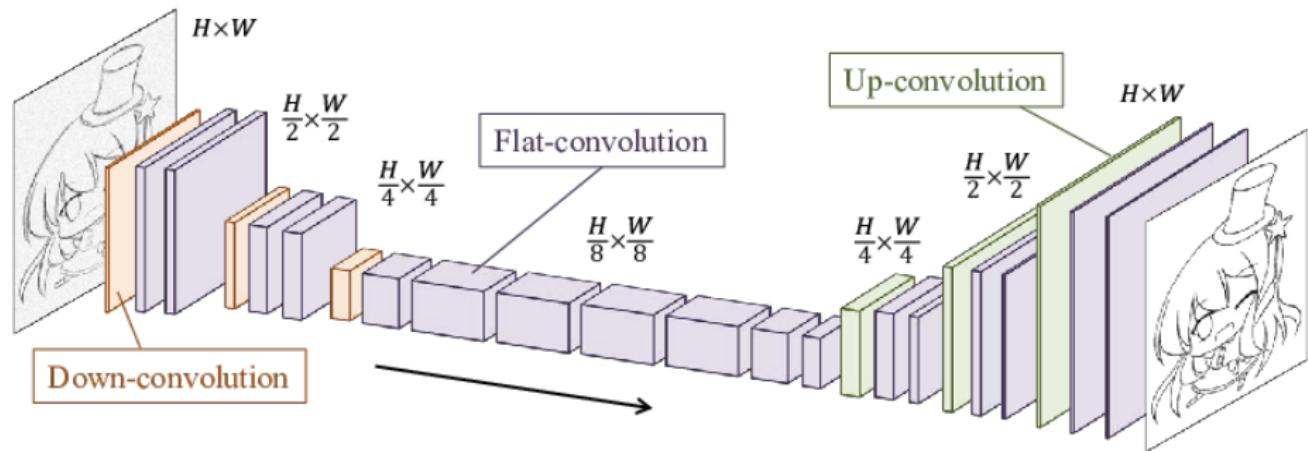
CNN + Autoencoder network: drawing simplification



Simo-Serra, Iizuka, Sasaki, Ishikawa (2016)

# Architecture examples

CNN + Autoencoder network: drawing simplification



Simo-Serra, Iizuka, Sasaki, Ishikawa (2016)

## Architecture examples

### Neural style

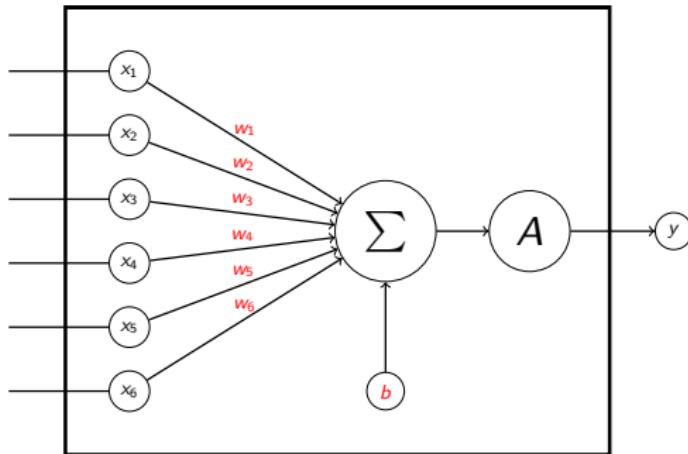
Consider an image  $I$  and its image by the convolution layers of the VGG network  $\text{Conv}(I)$ .

## Architecture examples

### Neural style

Consider an image  $I$  and its image by the convolution layers of the VGG network  $\text{Conv}(I)$ .

We can reconstruct a picture  $I'$  with the same content as  $I$  by applying gradient descent on the pixels in order to obtain  $\text{Conv}(I)$  as output, starting with white noise.

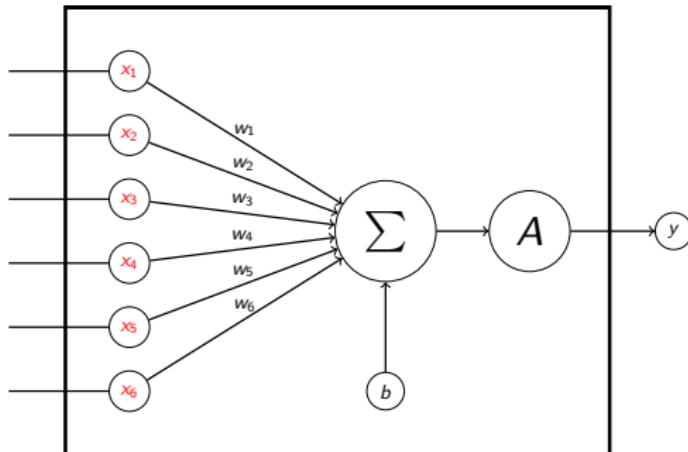


## Architecture examples

### Neural style

Consider an image  $I$  and its image by the convolution layers of the VGG network  $\text{Conv}(I)$ .

We can reconstruct a picture  $I'$  with the same content as  $I$  by applying gradient descent on the pixels in order to obtain  $\text{Conv}(I)$  as output, starting with white noise.



## Architecture examples

### Neural style

Gatys, Ecker and Bethge built a formula to extract the *style* of an image using the convolutional layers of the VGG network (2015).

They build the following Gram matrix:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

where  $F_{ij}^l$  is the  $j$ -th output of the  $i$ -th filter of the  $l$ -th convolutional layer.

# Architecture examples

## Neural style

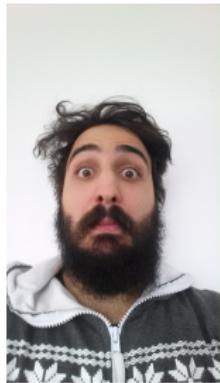
Let's take a random image from the internet.



# Architecture examples

## Neural style

$\alpha$  content(



) +  $\beta$  style(



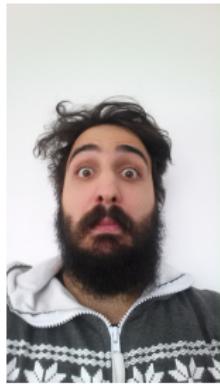
) =



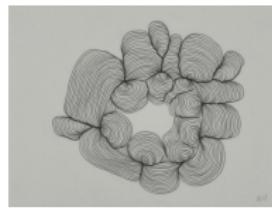
# Architecture examples

## Neural style

$\alpha$  content(



) +  $\beta$  style(



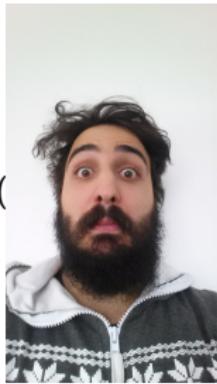
) =



# Architecture examples

## Neural style

$\alpha$  content(



) +  $\beta$  style(



) =



# Architecture examples

## Neural style

$\alpha$  content(



) +  $\beta$  style(



) =



# Architecture examples

## Recurrent neural network

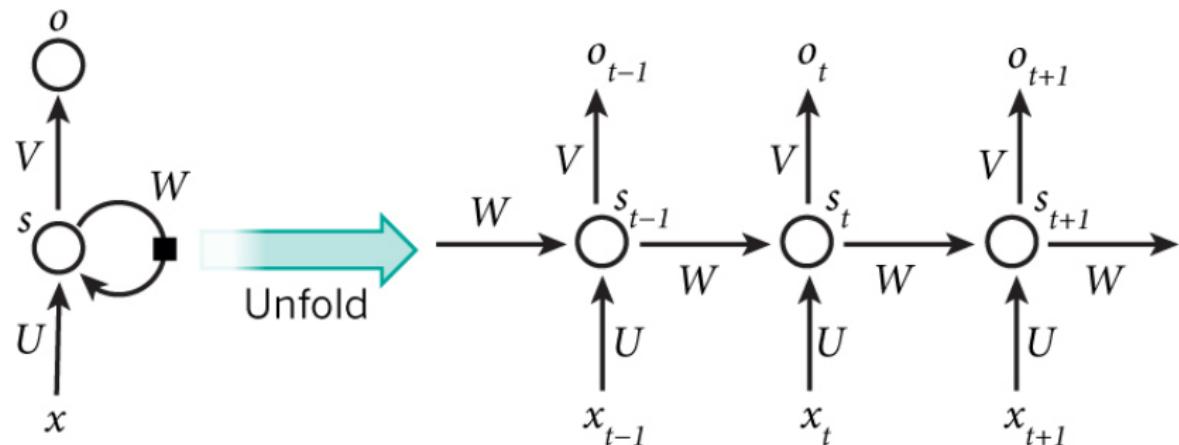
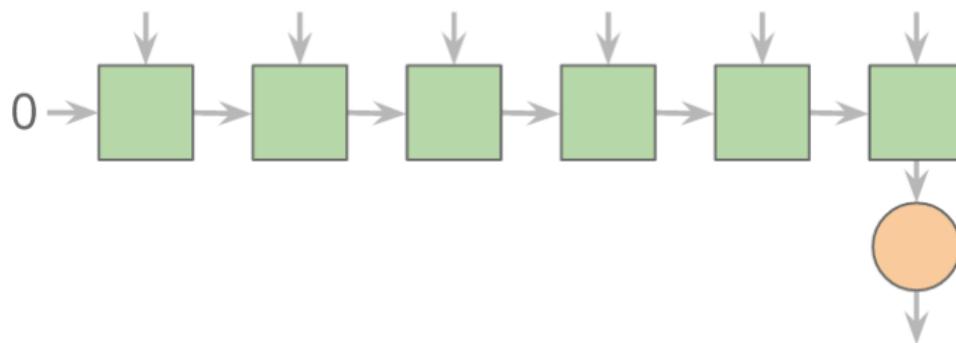


Image from <http://www.wildml.com>

## Architecture examples

Sequence to class network: text classifier

*The USA and China have agreed*



*geopolitics*

Image from Martin Gorner

## Architecture examples

Sequence to sequence network: Neural Machine Translation

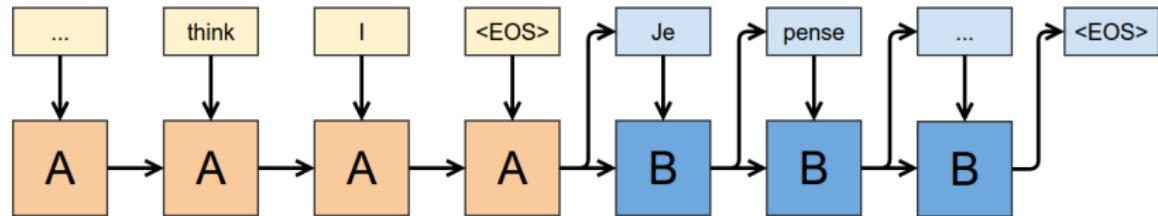


Image from <https://colah.github.io>

Google, September 2016: “The Google Translate mobile and web apps are now using GNMT (Google NMT) for 100% of machine translations from Chinese to English—about 18 million translations per day.”

# Architecture examples

Sequence to sequence network: Neural Conversation Model

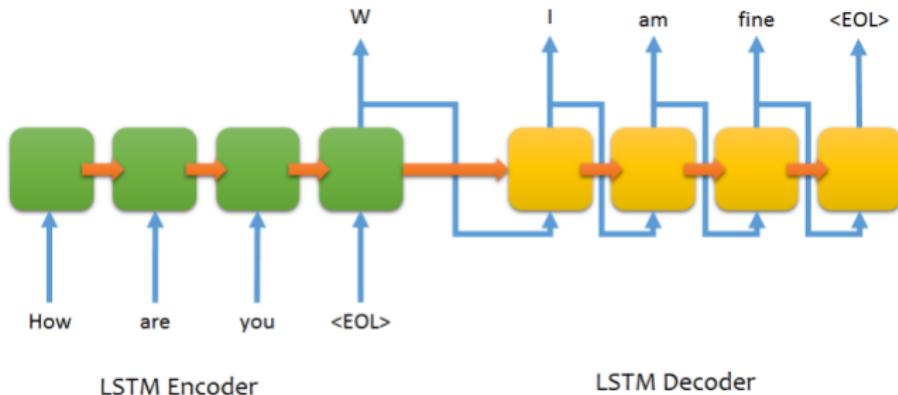


Image from <https://github.com/farizrahman4u/seq2seq>

Really early stage, hard to overcome challenges (context, coherent personality, . . . )

# Architecture examples

Image to sequence: automatic captioning

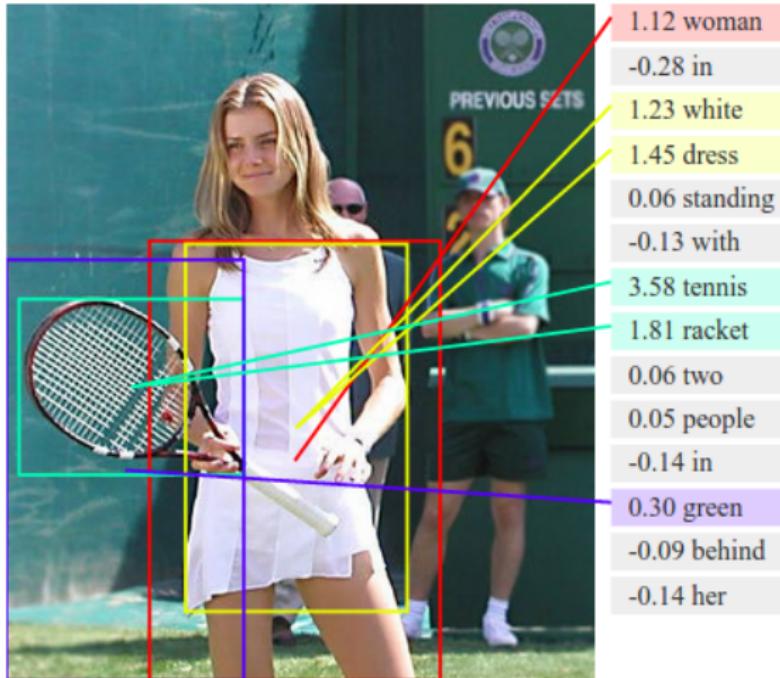


Image from <https://quantumfrontiers.com>

# Architecture examples

Image to sequence: automatic captioning

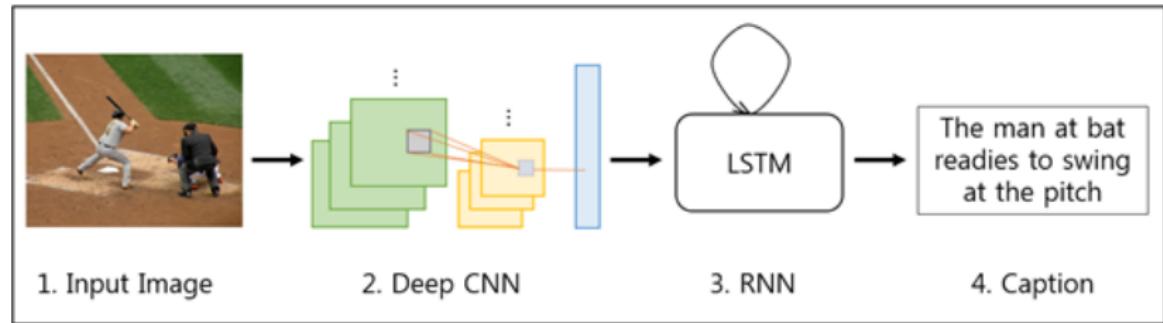


Image from <http://brain.kaist.ac.kr/>

# Architecture examples

## Generative adversarial network

### Generative adversarial networks (conceptual)

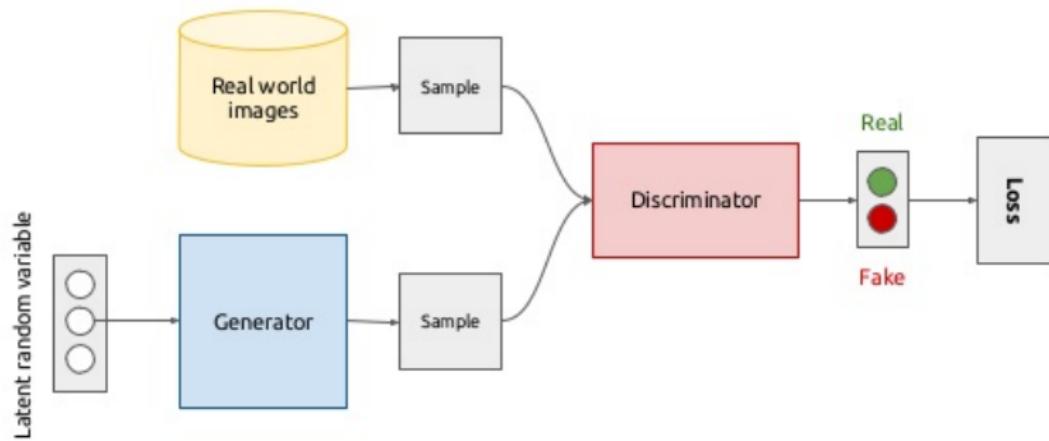


Image from <http://wiki.tum.de/>

# Architecture examples

Generative adversarial network: text to image

Han Zhang et al. (2016)

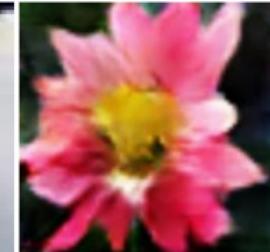
This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face



This bird is white with some black on its head and wings, and has a long orange beak



This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments



(a) Stage-I images



(b) Stage-II images

Image from <https://arxiv.org/pdf/1612.03242.pdf>