VIBOT 2018

Machine learning with Neural Networks

Francesco Ciompi

francesco.ciompi@radboudumc.nl

Radboudumc

Computational Pathology Research Group

The Computational Pathology Group of the Department of Pathology of Radboudumc works in close collaboration with DIAG. The history of the group reaches back to the early 1980s, while the current group has been active since 2013. We have a strong translational focus, owing to our strong embedding in a clinical environment. We have a leading position in our research field, witnessed for instance by the highly successful CAMELYON16 and CAMELYON17 grand challenges which we organized and by our active role in committees and conference organisation (e.g. the Computational Pathology Symposium, held in conjunction with the European Congress of Pathology).

Faculty

Jeroen van der Laak
+31 24 3614367

Geert Litjens
+31 243614322

Francesco Ciompi
+31 24 3614322

Technicians

Karel Gerbrands
+31 24 3652283

Rob van de Loo
+31 24 36 93153

Course overview

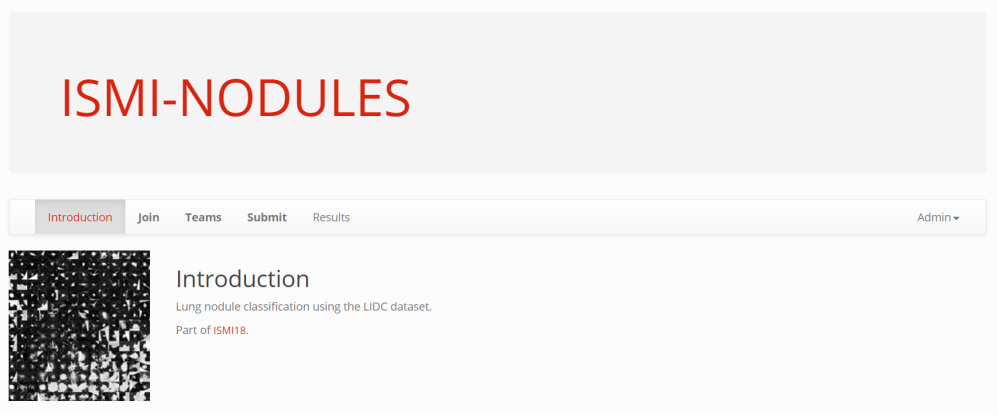
	Theory	Workshop
Monday 9 April	10:00 – 12:00	15:00 – 17:00
Tuesday 10 April	12:00 – 14:00	15:00 – 17:00
Wednesday 11 April	--	12:00 – 14:00

Course material

- **Lecture slides**
  - Robert sends you the slides

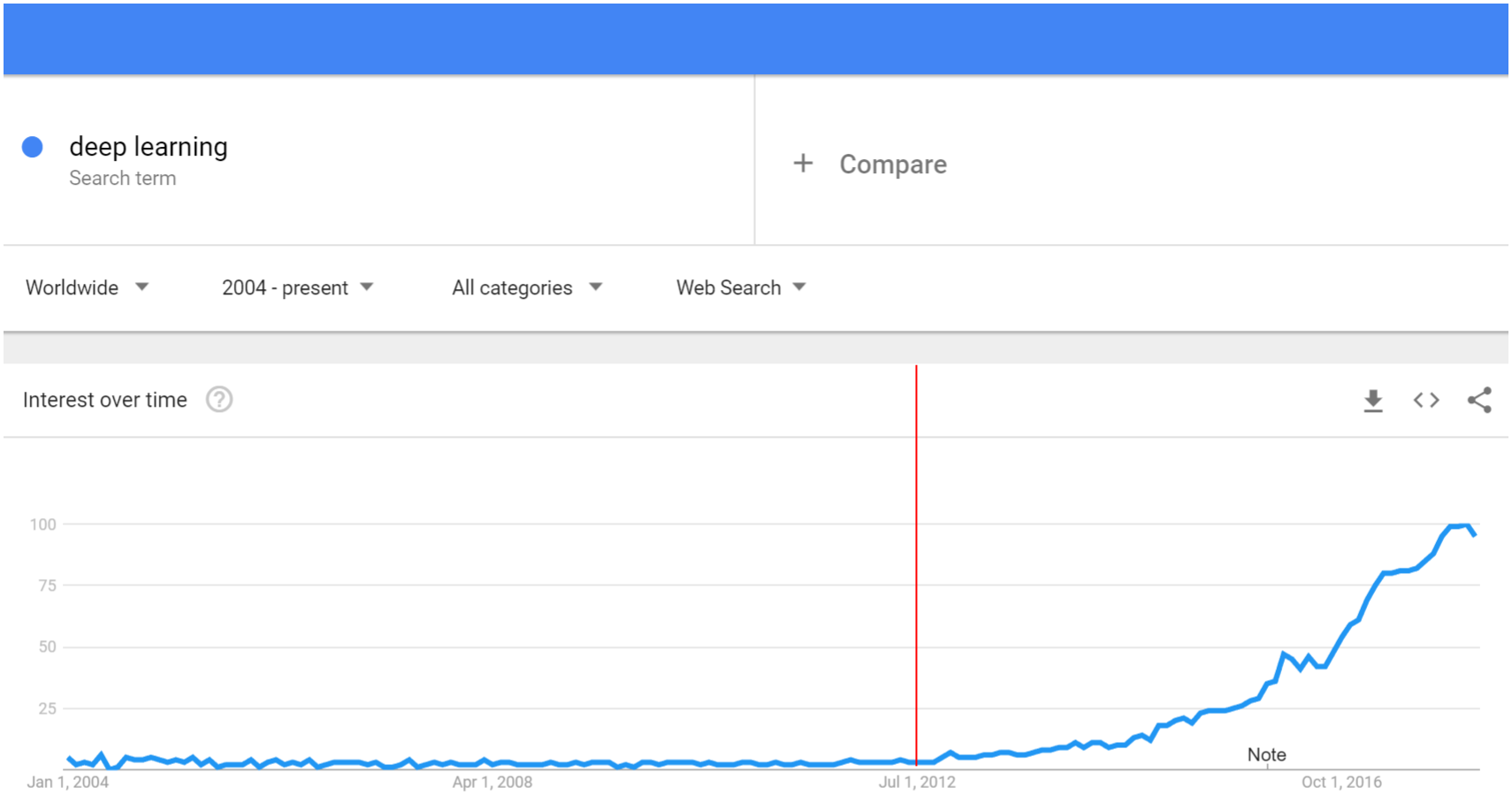
- **Python: Jupyter notebooks**
  - Assignment submission deadline: May 15, 2018

- www.grand-challenge.org

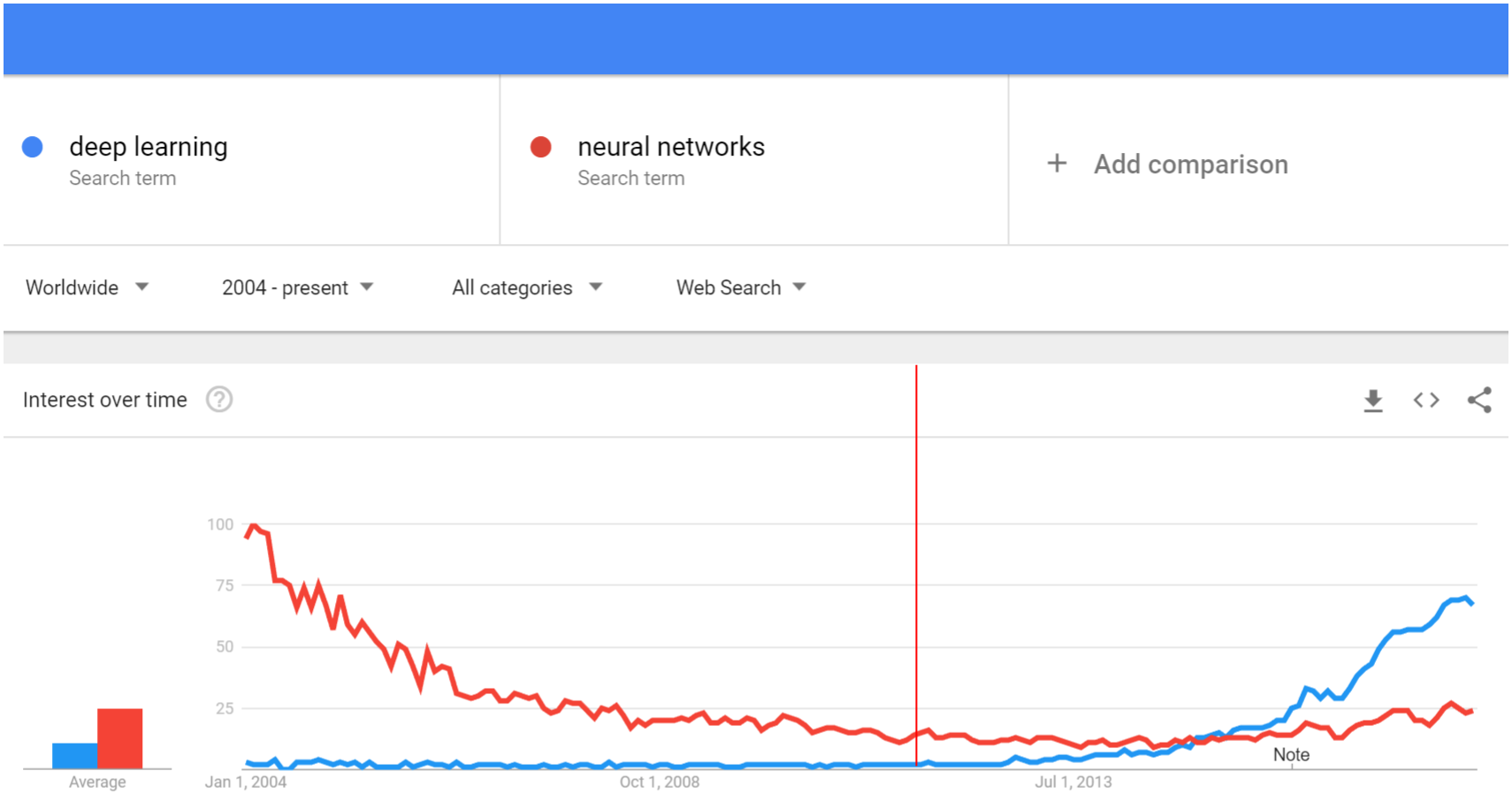


- Access to Linux computer with **3 GPUs**

What happened in 2012?



What was happening before 2012?



Machine Learning

- Our focus will be on:
  - **Supervised** learning
    - *"the truth is given"*
  - **Classification** problems
    - *"we aim at labeling data"*
  - **Discriminative** models
    - *"we aim at finding a decision boundary"*

Data in supervised learning

- We are typically given *N* training samples:

$$(\boldsymbol{x}_i, y_i),\ i = 1, \ldots, N$$

$\boldsymbol{x}_i$: *feature vector for* i^{th} *sample*
y_i: *label for* i^{th} *sample*

Data in supervised learning

- We are typically given *N* training samples:

  $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$

  $\boldsymbol{x}_i$: *feature vector for* i^{th} *sample*
  y_i: *label for* i^{th} *sample*

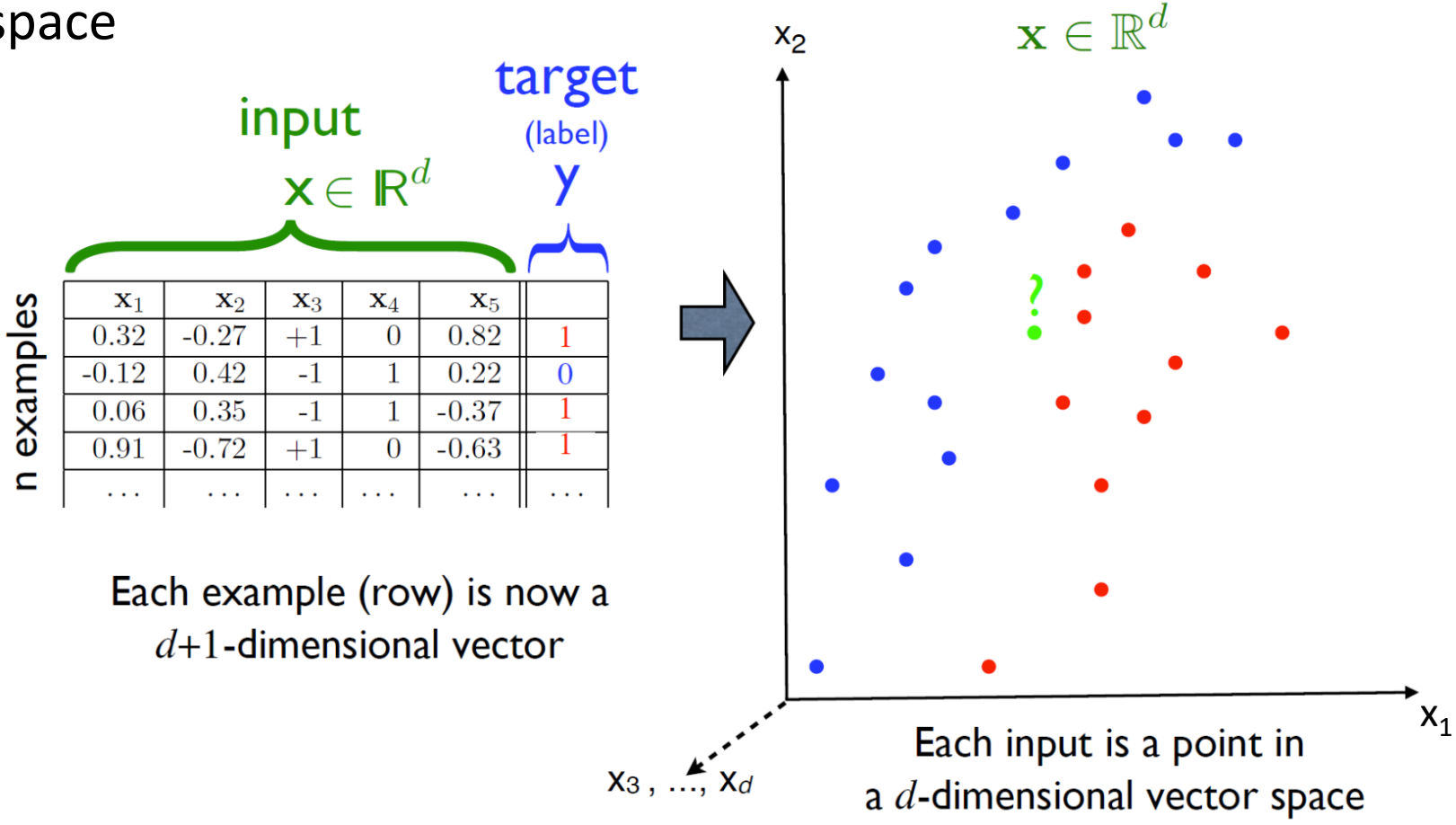
- We want to learn a model (function)

  $f(\boldsymbol{x}) = y$

  that does a good job at predicting the right label *y* for unseen data samples **x**

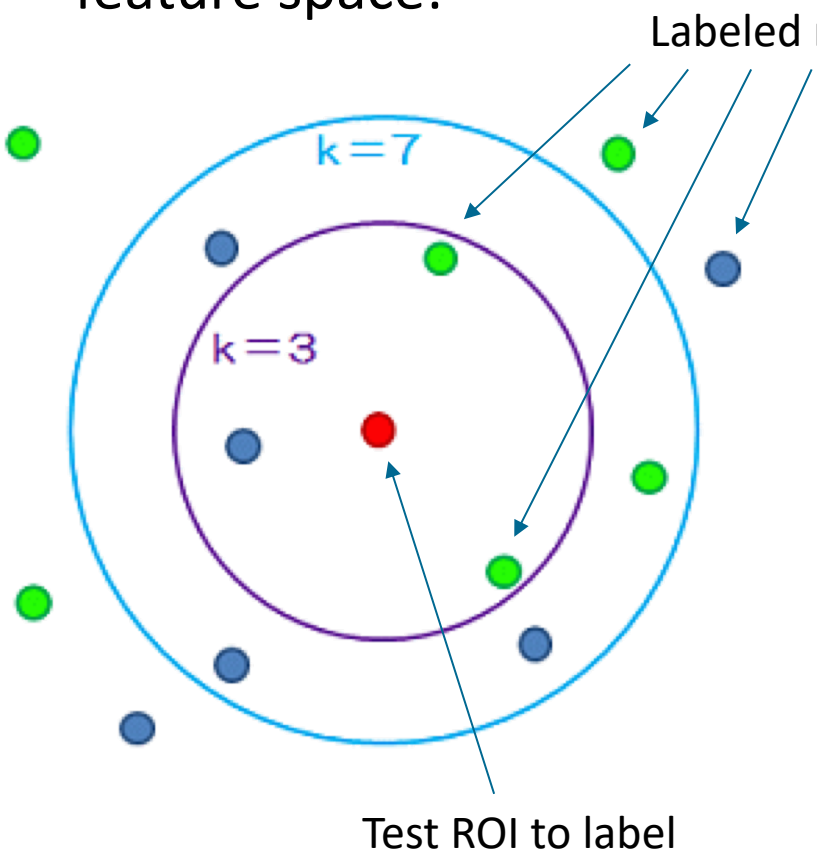
Data in supervised learning

- Dataset imagined as a point cloud in a high-dimensional vector space



Simple classification approach

- The label is assigned based on the labels of neighbors in the feature space:



The parameter **k** has to be tuned based on the data at hand

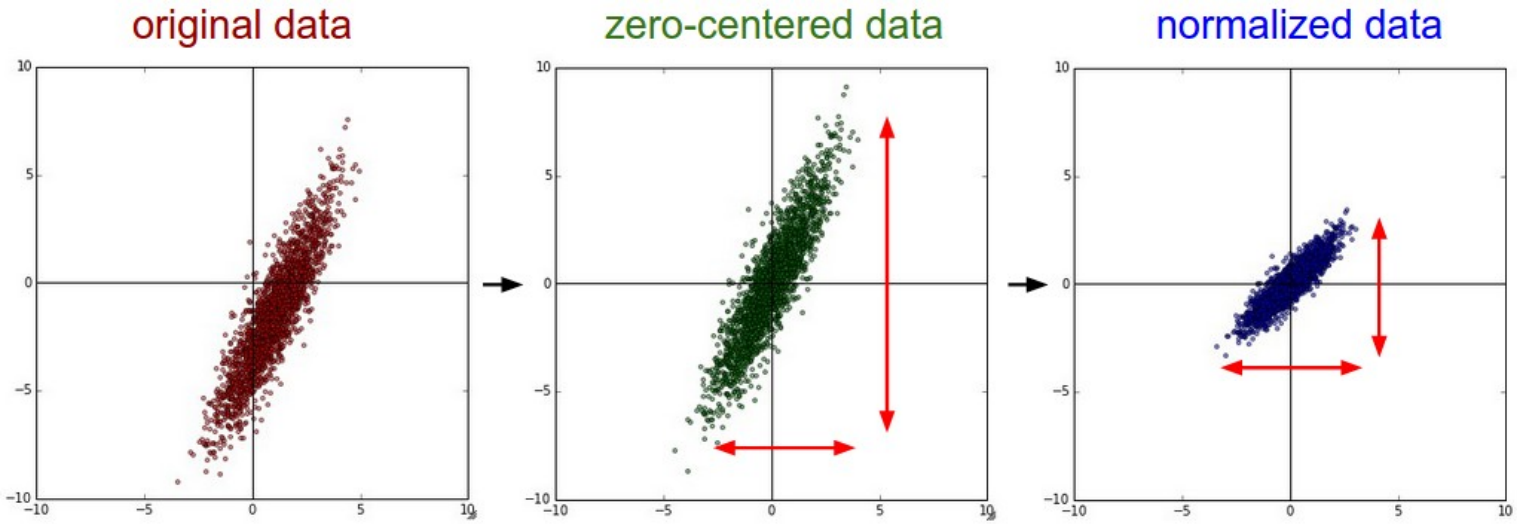
Rule of thumb
K ≈ square root of the number of samples in the training set

No model to train
We have to store all the training samples

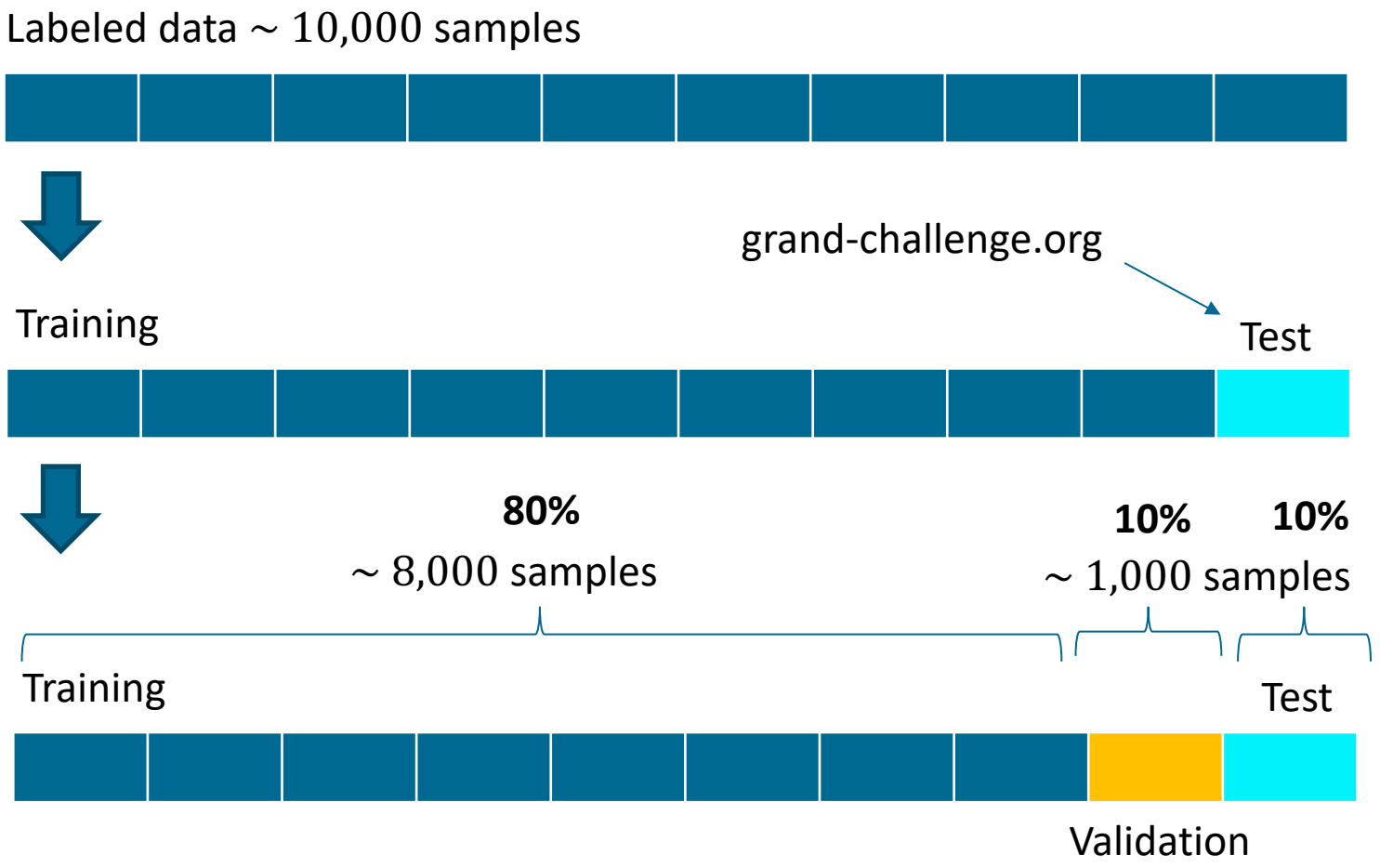
Feature normalization

- Features can have completely **different ranges**
- If you use **distance-based** algorithms, this can be a problem
- **Normalization** is a good idea (not only when kNN is used)
  - Zero mean, unit standard deviation

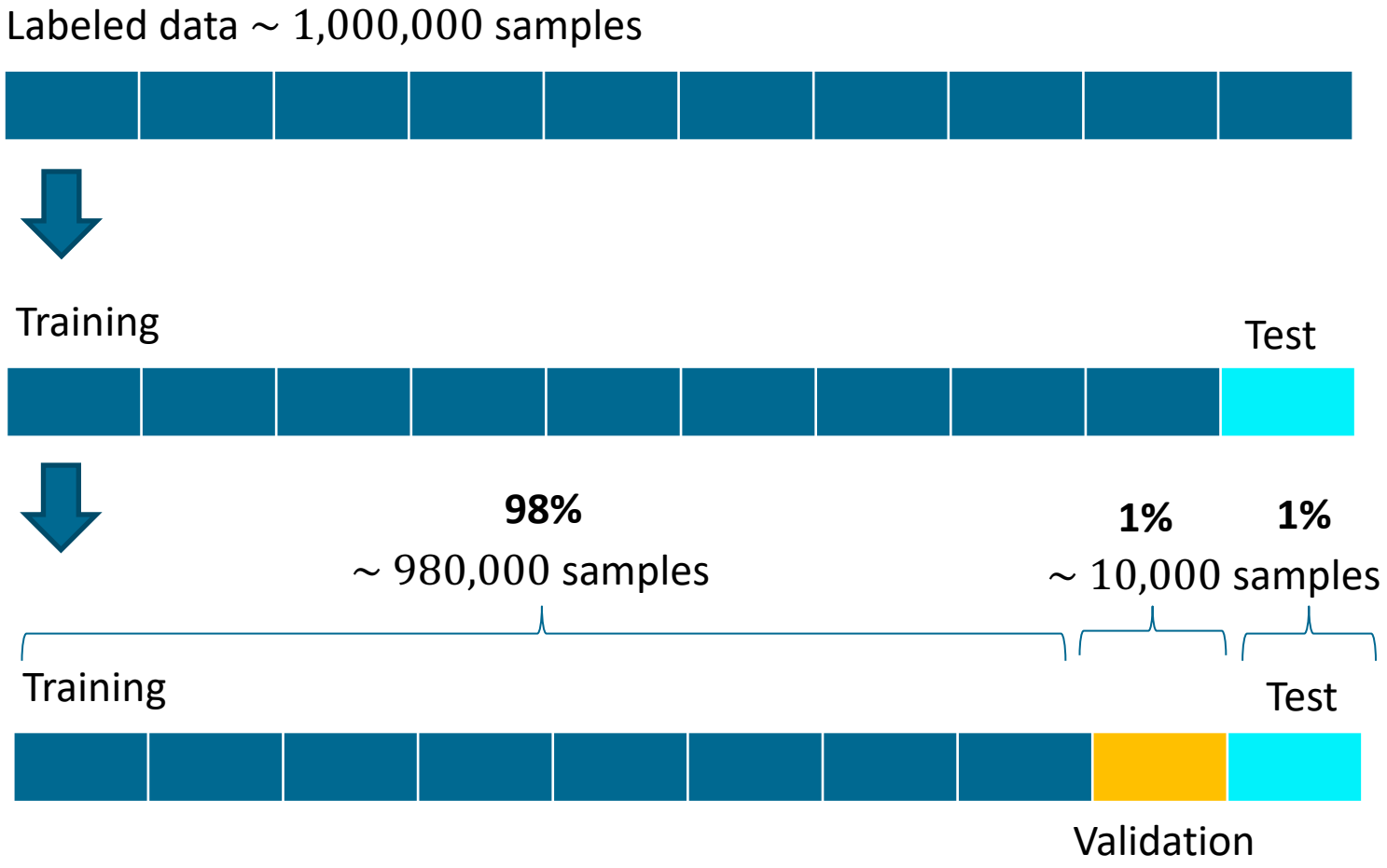
- Normalization is done **per feature**



Supervised learning: how to split data



In the Deep Learning era…



Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 1

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 2

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 3

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

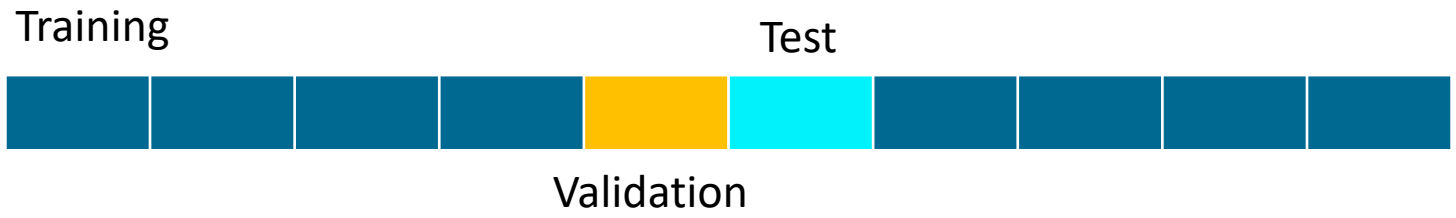
Data tested so far:

iteration: 4

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations



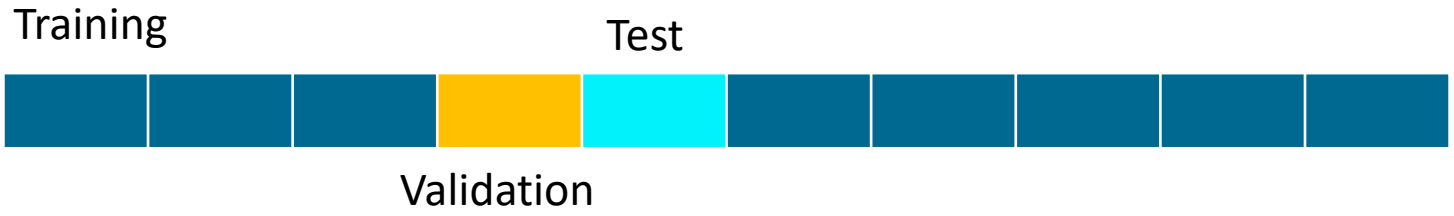
Data tested so far:

iteration: 5

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations



Data tested so far:

iteration: 6

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

iteration: 7

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 8

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 9

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k - 2$ as the **training** set
- Repeat *k* times for all possible combinations

Data tested so far:

iteration: 10

Cross validation

k-fold cross validation:

- Divide the **labeled dataset** into k subsets (e.g., k=10)
- Use one subset as the **validation** set
- Use one subset as the **test** set
- Use the remaining $k-2$ as the **training** set
- Repeat *k* times for all possible combinations

- Called *leave one-out* if k=|trainingData|

In medical imaging, do not mix **patients** in training/ validation / test

NO INTERSECTION OF PATIENT DATA ACROSS FOLDS !

Parametric models

Parametric model

- The result of our **supervised learning** procedure is a **function** *f(x)* that **predicts** the label *y* for a given input ***x***

- Typically, the model we are building has some parameters ***W***

$$y = f(\boldsymbol{x}, \boldsymbol{W})$$

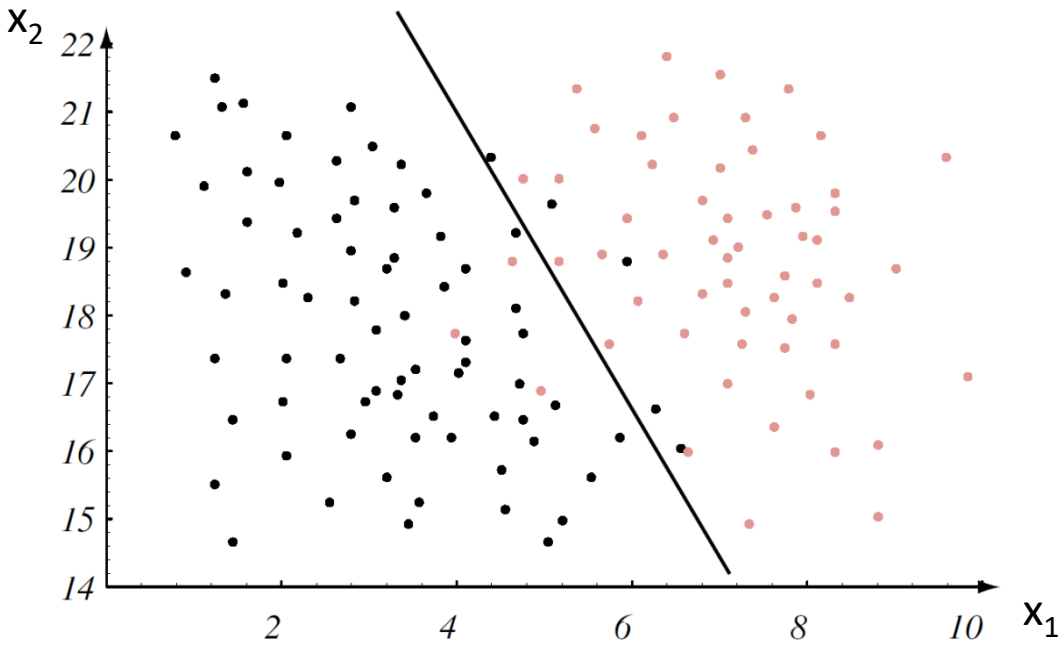
Data
(image, feature vector)

Parameters

- The amount of parameters of the model defines its **capacity**

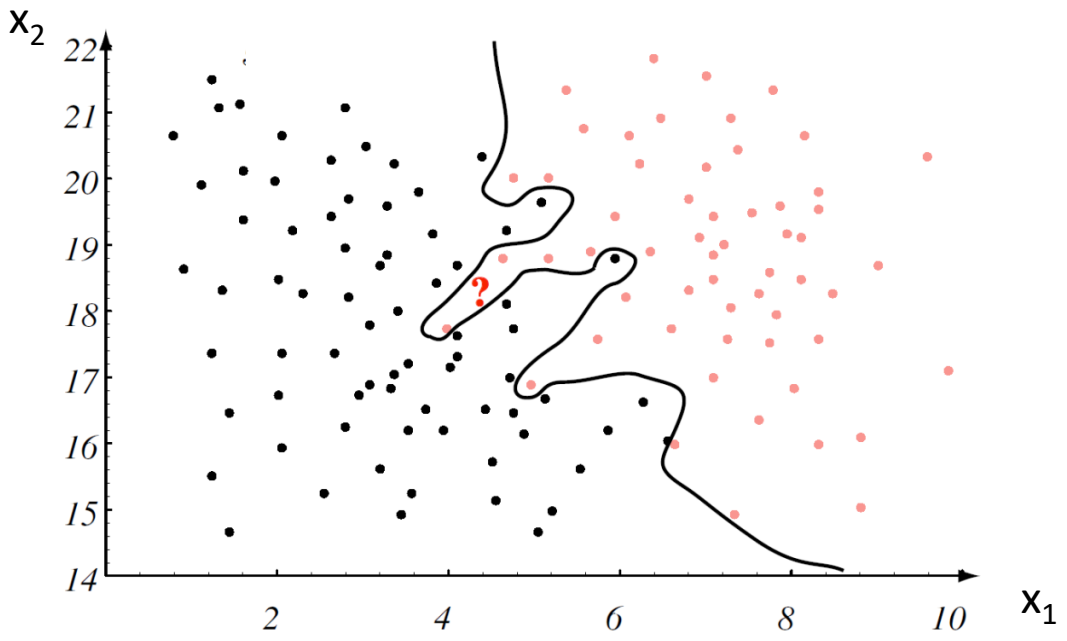
Capacity of a model

- Function family too poor (too inflexible)
- Capacity too low
- Under-fitting



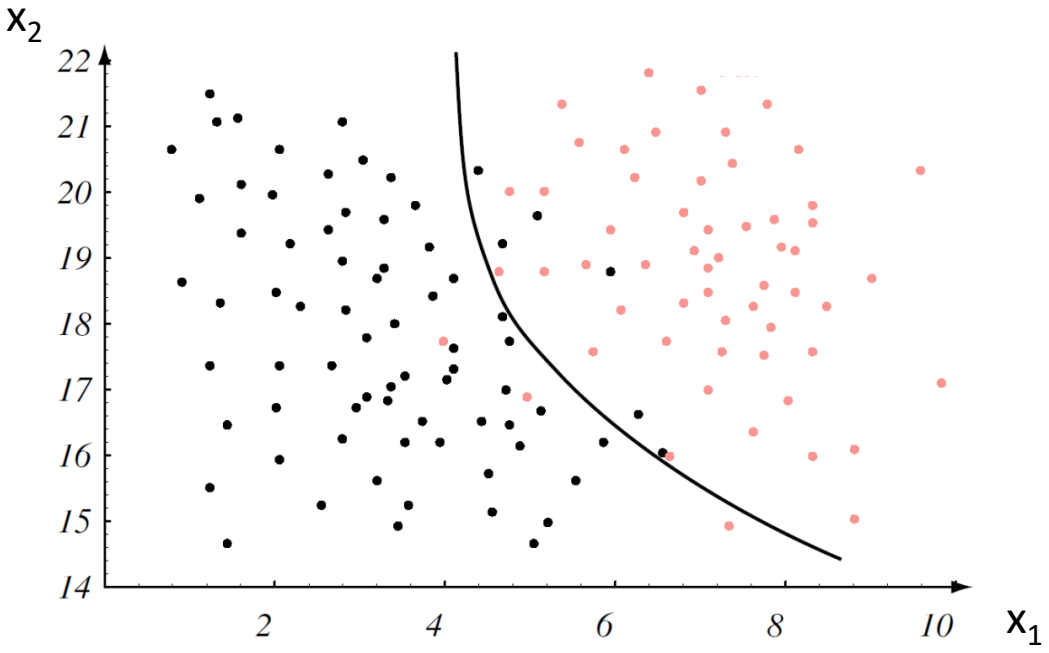
Capacity of a model

- Function family too rich (too flexible)
- Capacity too high (for this problem, relative to the number of examples)
- Over-fitting

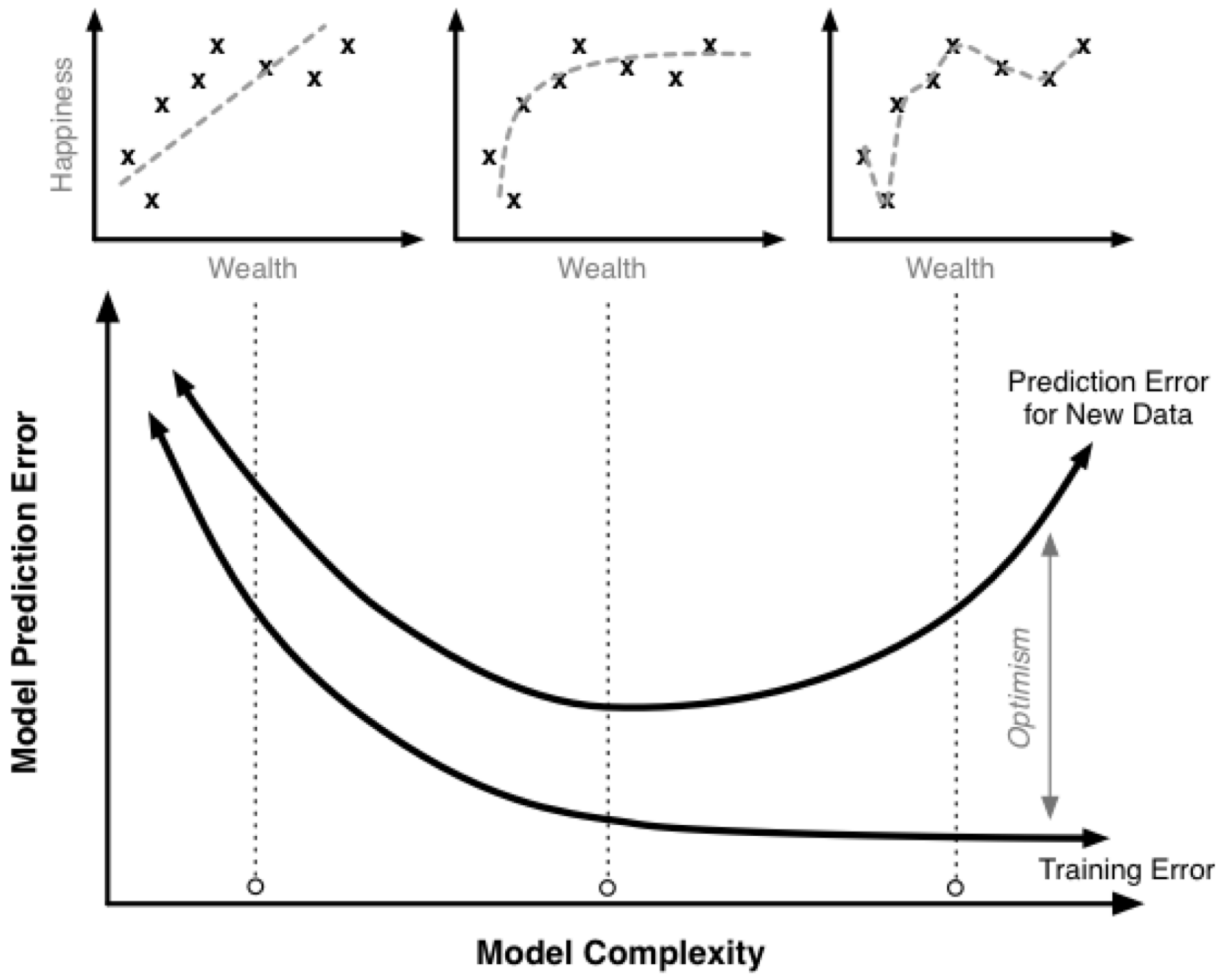


Capacity of a model

- Optimal capacity (for this problem)
- **Best generalization** (on future test points)



Capacity of a model



Training a parametric model

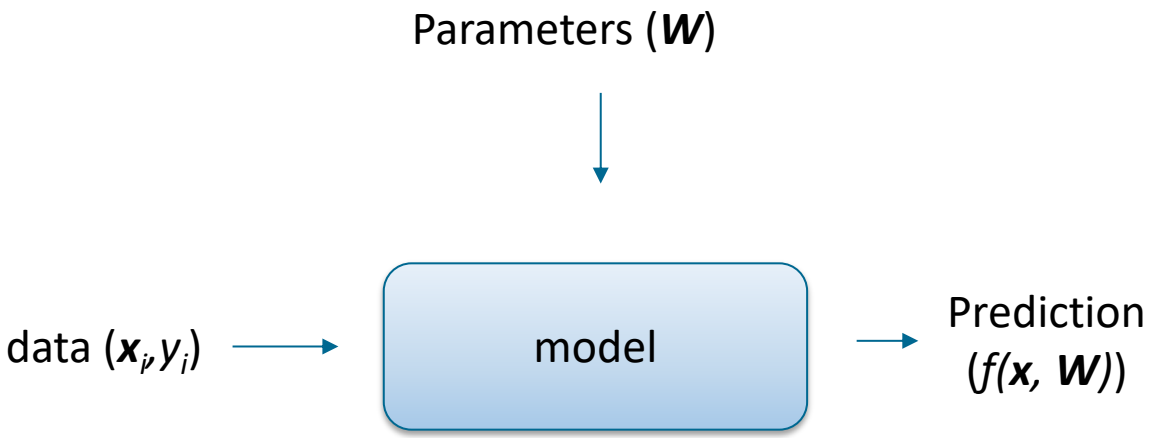
Supervised learning

- The goal of a supervised learning algorithm (**and ours**):

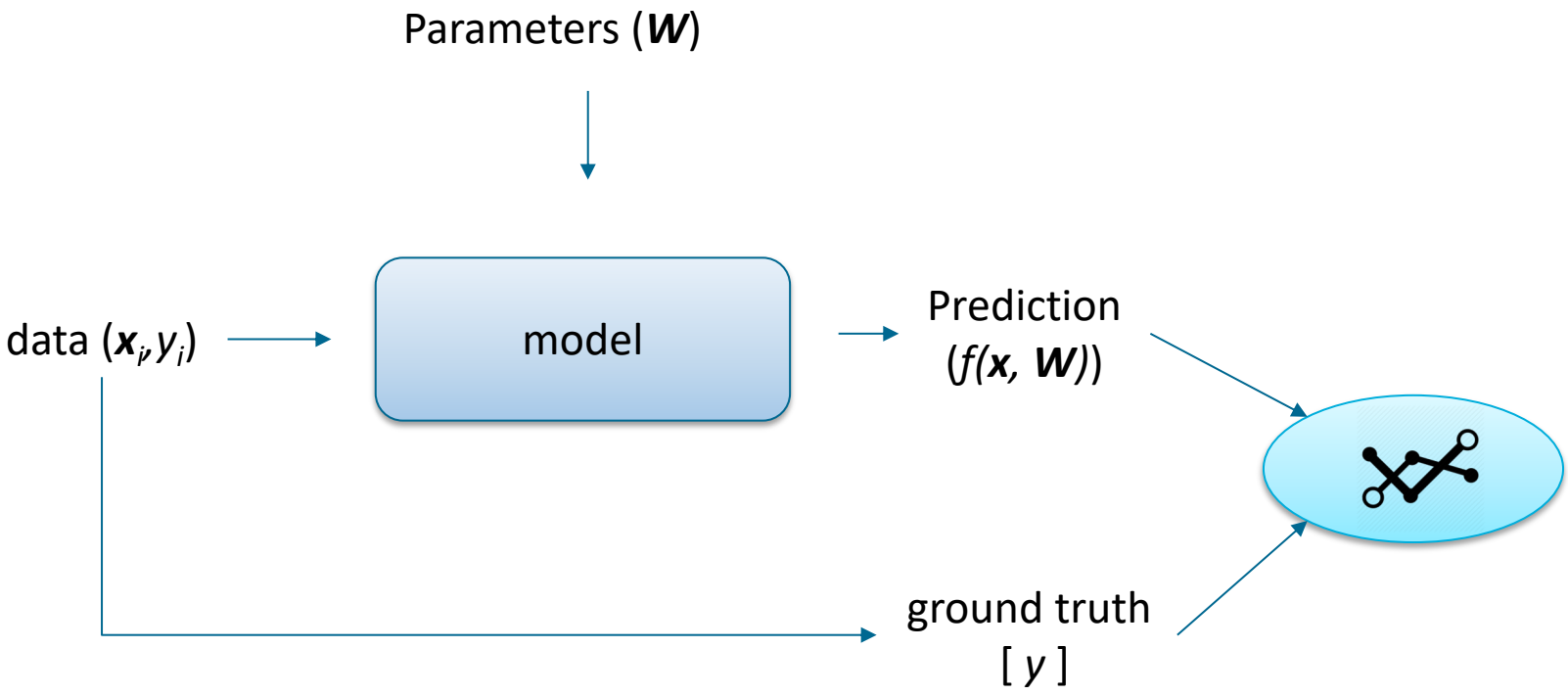
  1. Learn from training data (achieve low training error)
  2. Generalize to unseen data (achieve low validation error)
  3. Avoid overfitting (when both condition (1) and (2) are achieved)

- Let's start from the first question:
  - *"How can we learn from data?"*

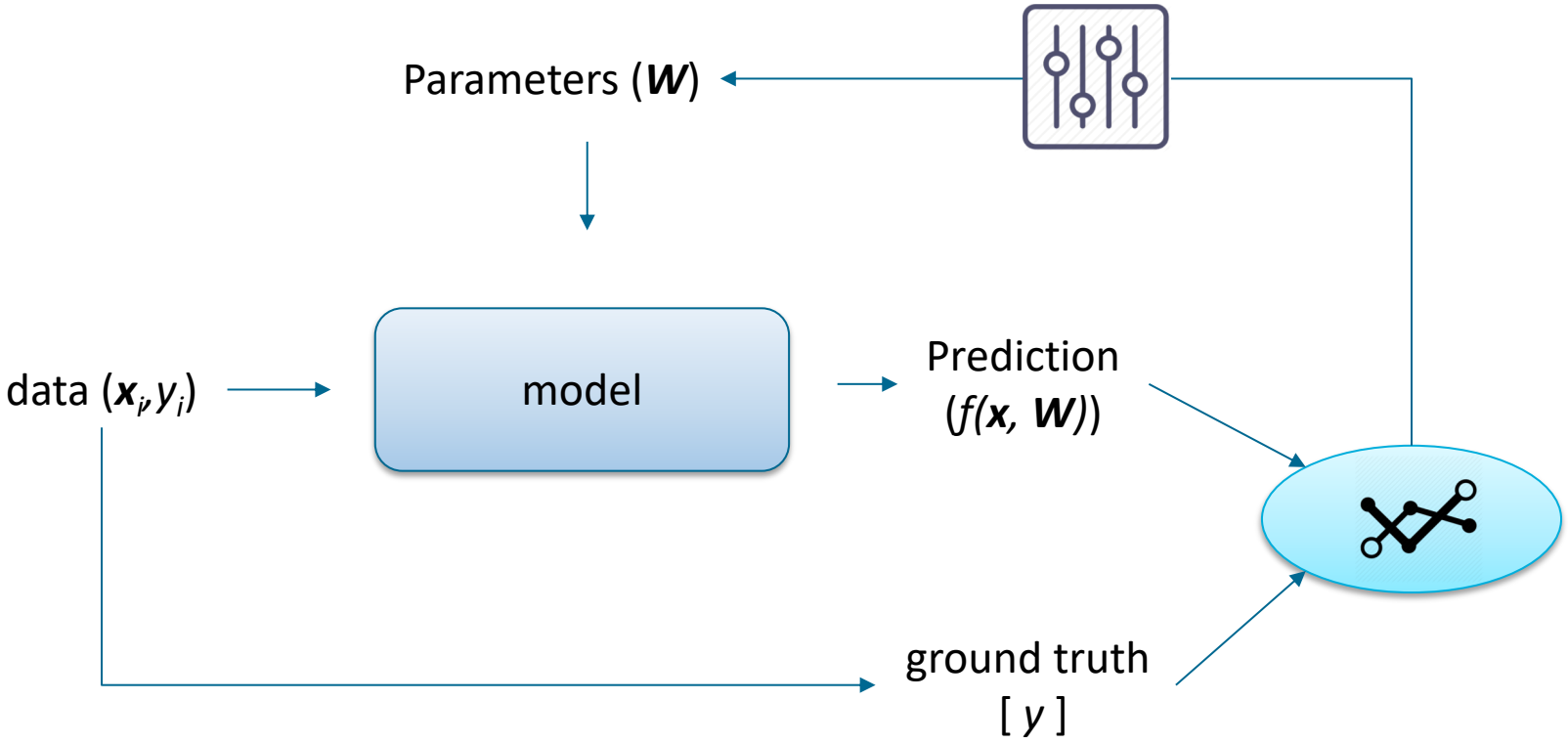
Supervised learning procedure



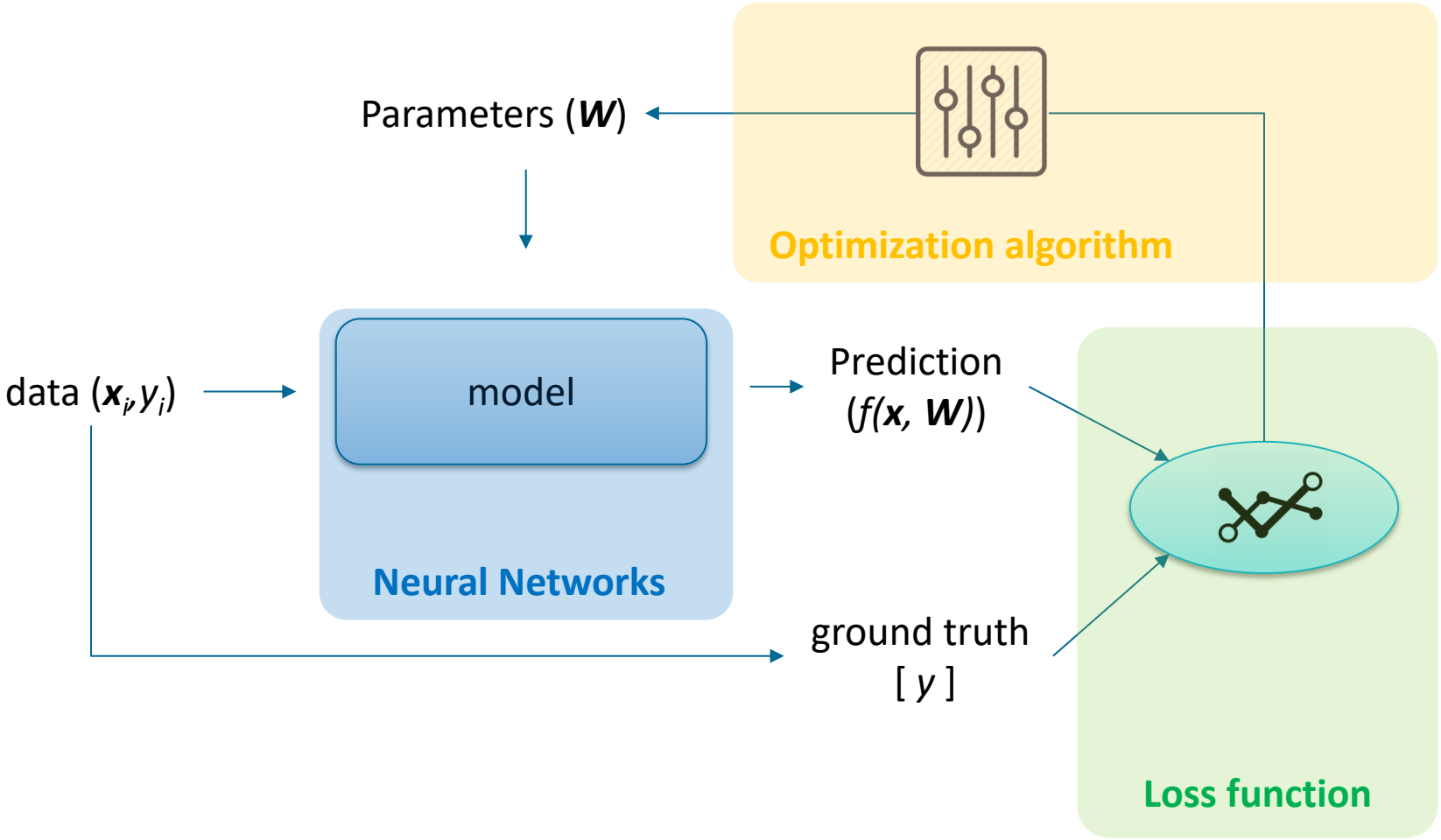
Supervised learning procedure



Supervised learning procedure

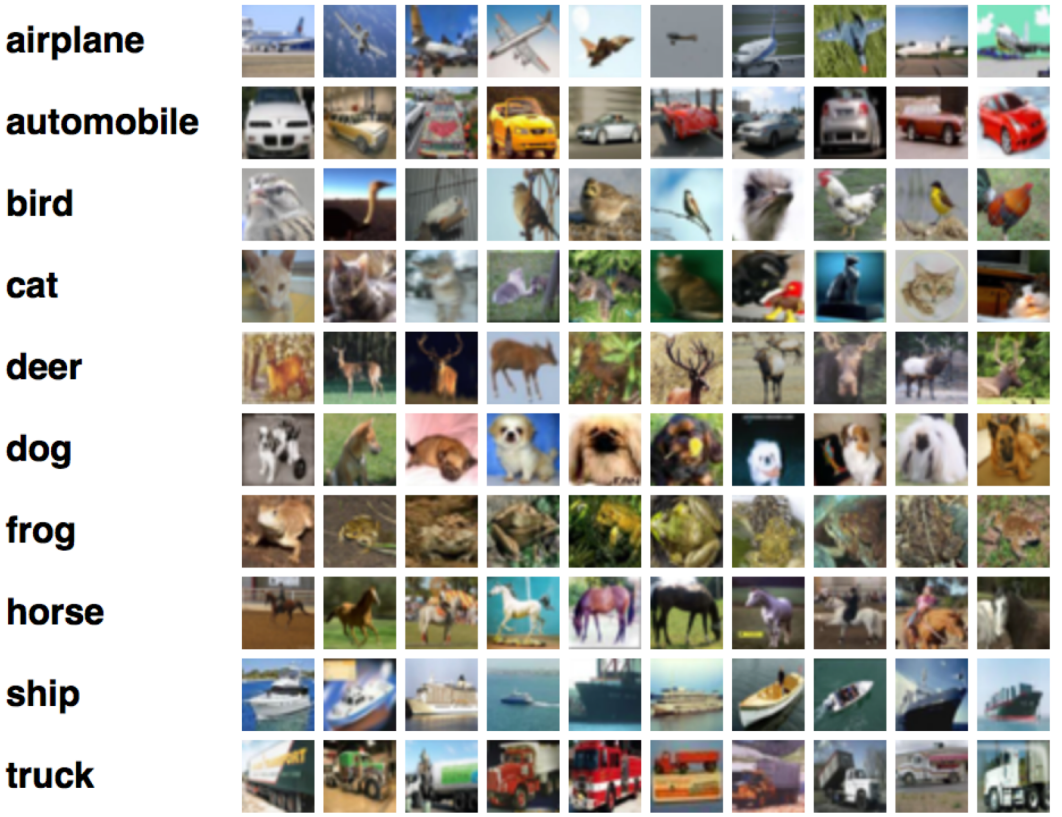


Supervised learning procedure



A simple model: linear classifier

- Suppose we want to classify images in the CIFAR10 dataset



A simple model: linear classifier

$$f(x, W) = Wx$$

3072x1

10x1

10x3072

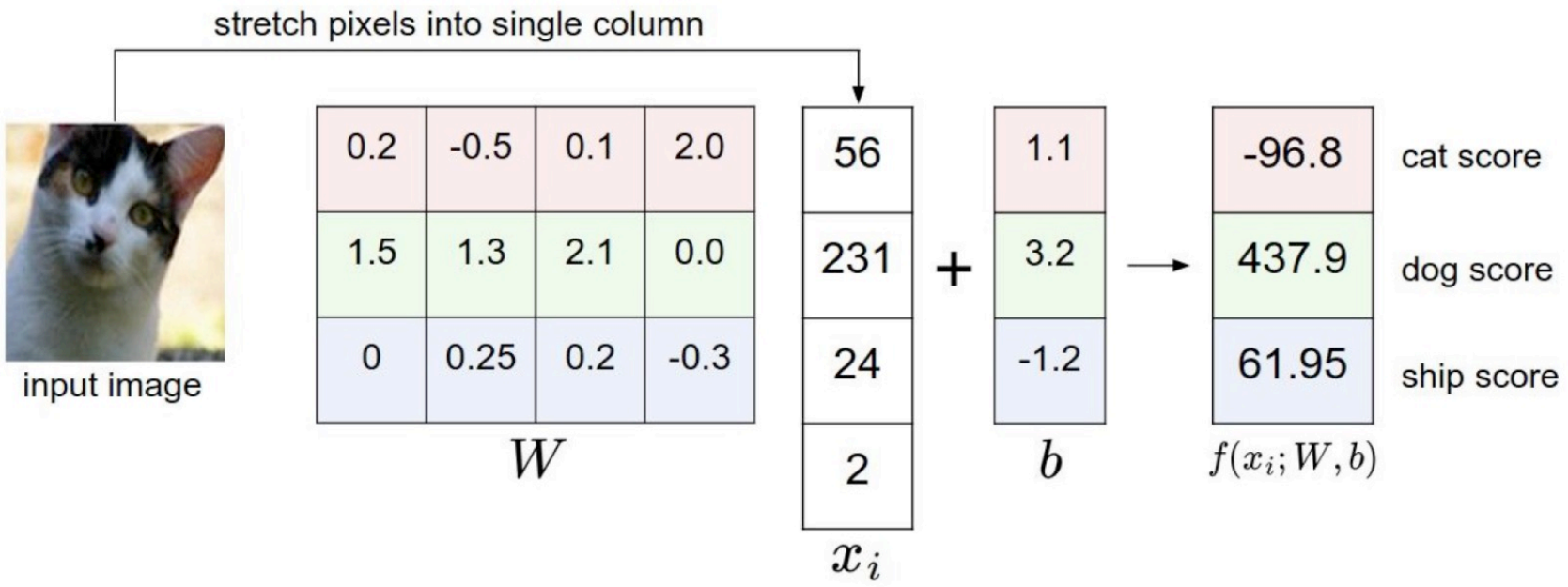
10 numbers, indicating class scores

[32x32x3]
array of numbers 0...1

parameters, or “weights”

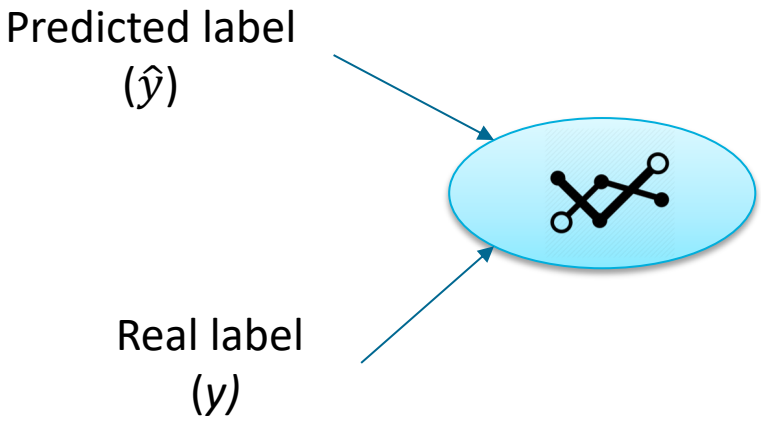
A simple model: linear classifier

- We simplify the example: image with 4 pixels, 3 classes (cat/dog/ship)



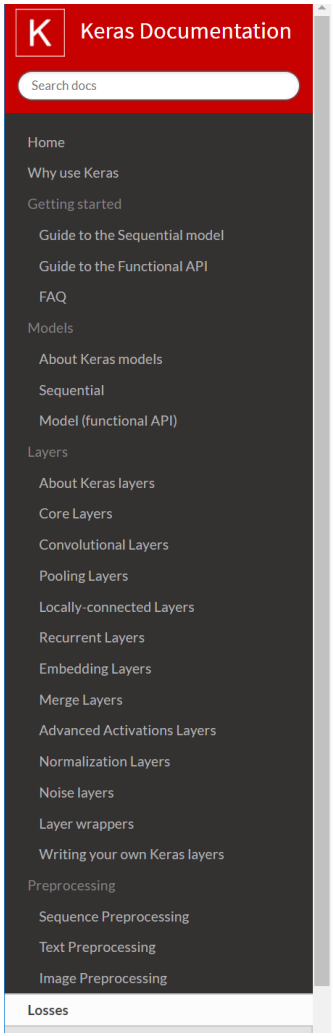
Loss function

- The loss function measures the difference between the **predicted label** and the **real label**



- During training, we want to **minimize the loss**
- It can be an **arbitrary** function, but **differentiable!!!**

Loss function



Arguments

- **y_true**: tensor of true targets.
- **y_pred**: tensor of predicted targets.

Returns

Tensor with one scalar loss entry per sample.

categorical_crossentropy

```
categorical_crossentropy(y_true, y_pred)
```

sparse_categorical_crossentropy

```
sparse_categorical_crossentropy(y_true, y_pred)
```

binary_crossentropy

```
binary_crossentropy(y_true, y_pred)
```

kullback_leibler_divergence

```
kullback_leibler_divergence(y_true, y_pred)
```

poisson

```
poisson(y_true, y_pred)
```

cosine_proximity

```
cosine_proximity(y_true, y_pred)
```

Loss function

- **Binary cross-entropy**
  - Real label: $y_i \in [0,1]$
  - Predicted label: $\widehat{y_i} \in [0,1]$

$$L(y_i, \hat{y}_i) = -yi \log(\hat{y}_i) - (1 - yi)\log(1 - \hat{y}_i)$$

Loss function

- **Binary cross-entropy**
  - Real label: $y_i \in [0,1]$
  - Predicted label: $\widehat{y_i} \in [0,1]$

$$L(y_i, \hat{y}_i) = -yi \log(\hat{y}_i) - (1 - yi)\log(1 - \hat{y}_i)$$

Intuition

- If y = 1: your loss function will force $\hat{y}$ to be **large**
- If y = 0: your loss function will force $\hat{y}$ to be **small**

Loss function

- **Categorical cross-entropy loss**
  - Real label: $y \in [0,1]$
  - Predicted label: $\hat{y} \in [0,1]$

$$L_i = -\sum_j y_{i,j} \log(\hat{y}_{i,j})$$

Loss function

- **Categorical cross-entropy loss**
  - Real label: $y \in [0,1]$
  - Predicted label: $\hat{y} \in [0,1]$

$$L_i = -\sum_j y_{i,j} \log(\hat{y}_{i,j})$$

$$L(y_i, \hat{y}_i) = -yi \log(\hat{y}_i) - (1 - yi)\log(1 - \hat{y}_i)$$

  - Extension of the binary case to multi-class

One-hot representation

- **Categorical cross-entropy for multi-class problems**
- How can we have multiple labels with only $y \in [0,1]$?

One-hot representation

- **Categorical cross-entropy for multi-class problems**
- How can we have multiple labels with only $y \in [0,1]$?
- Suppose we have 4 classes, for each sample, the format of the target will be actually a vector:
  - If y = 1

1	0	0	0

  - If y = 2

0	1	0	0

  - If y = 3

0	0	1	0

  - If y = 4

0	0	0	1

One-hot representation

- **Categorical cross-entropy for multi-class problems**
- How can we have multiple labels with only $y \in [0,1]$?
- Suppose we have 4 classes, for each sample, the format of the target will be actually a vector:
  - If y = 1 | 1 | 0 | 0 | 0 |
  - If y = 2 | 0 | 1 | 0 | 0 |
  - If y = 3 | 0 | 0 | 1 | 0 |
  - If y = 4 | 0 | 0 | 0 | 1 |

One-hot representation

Cost function

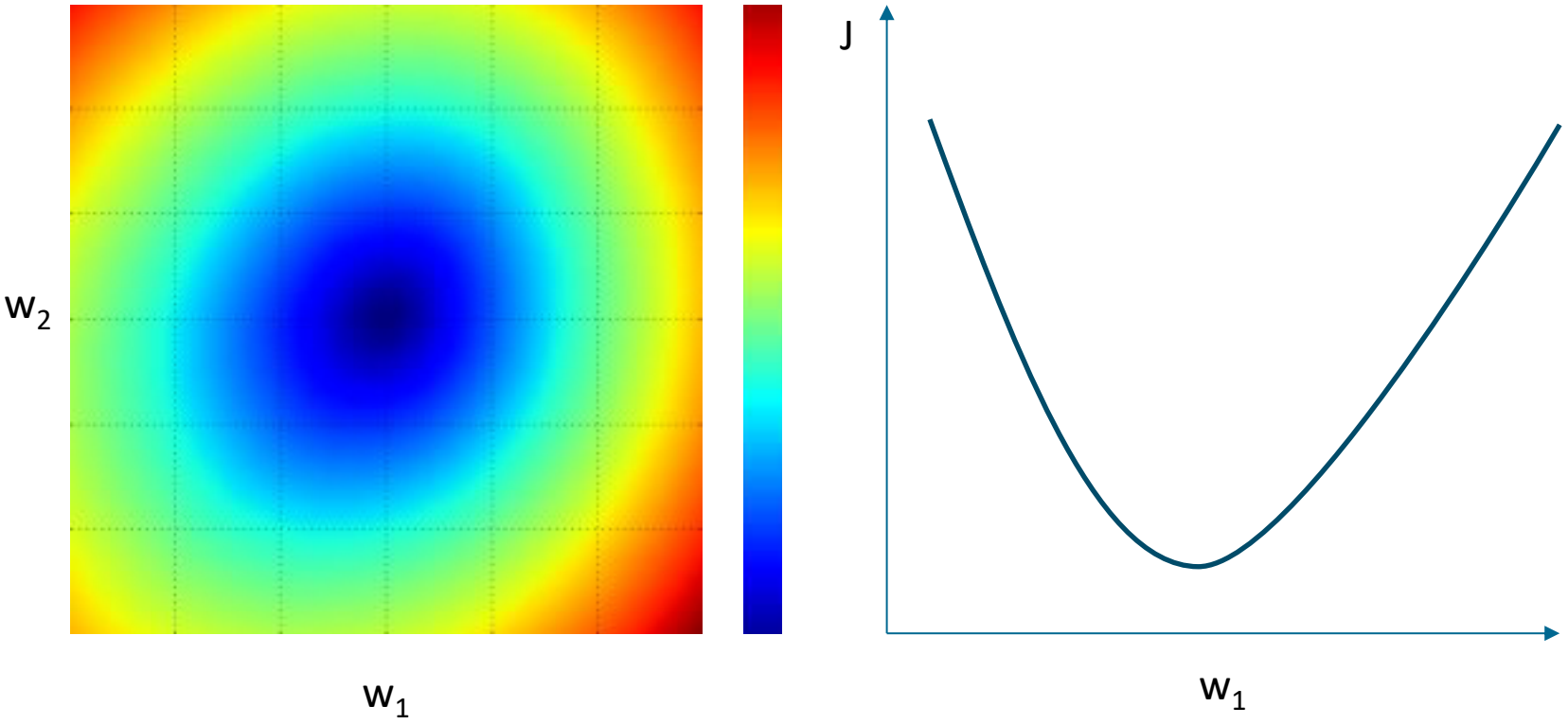
- Given a training set $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$:
  - We can define a **loss** function L_i for each training sample
  - We can define a **cost** function *J* for the entire training set

$$J = \frac{1}{N} \sum_{i=1}^{N} L_i$$

- The final cost function is the **average across the training set**

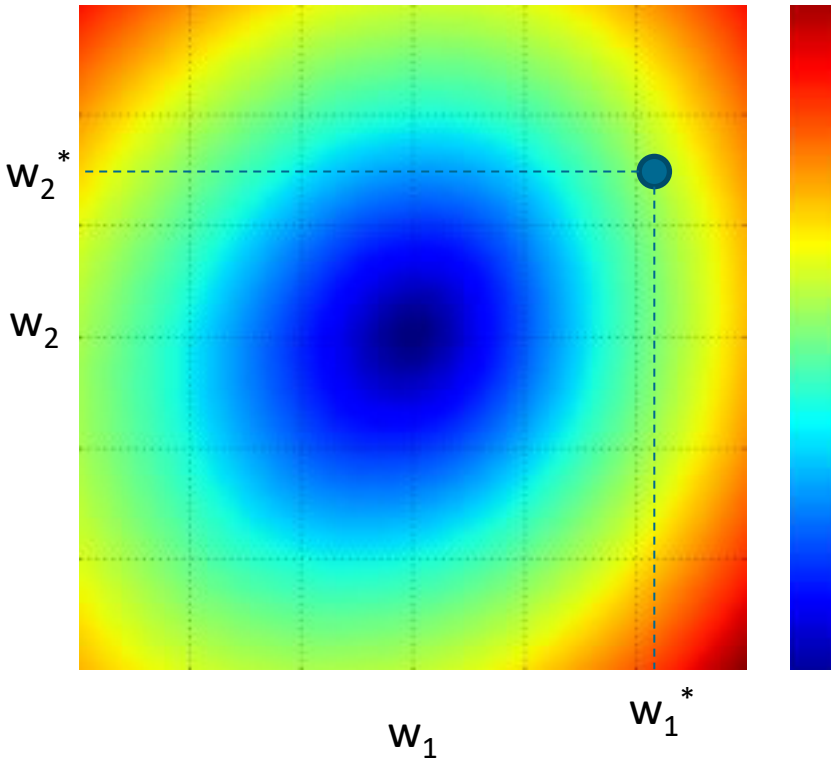
Cost function

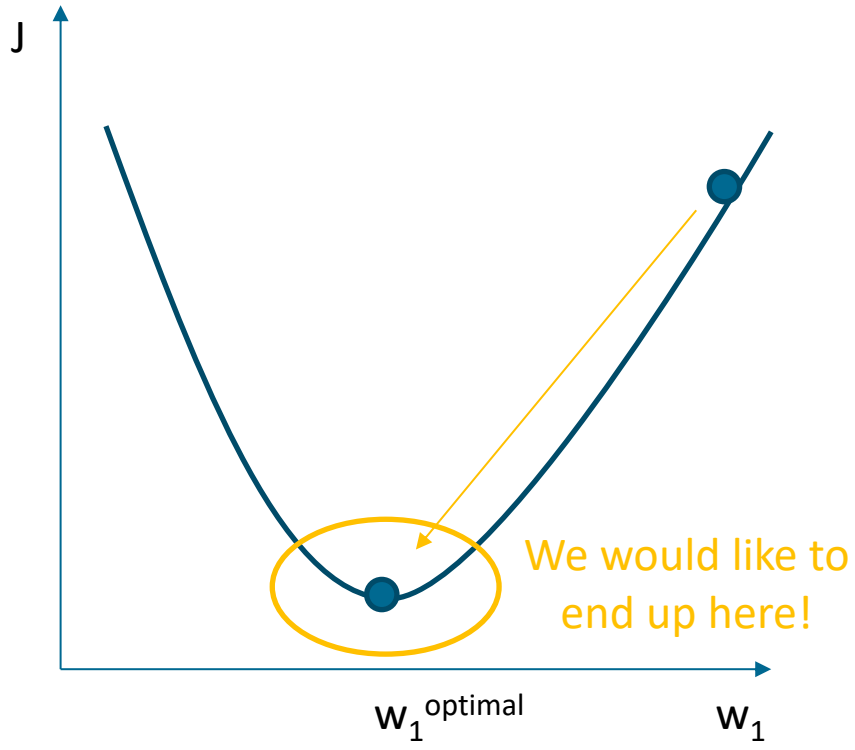
- The shape of the cost function could be something like this:



Cost function

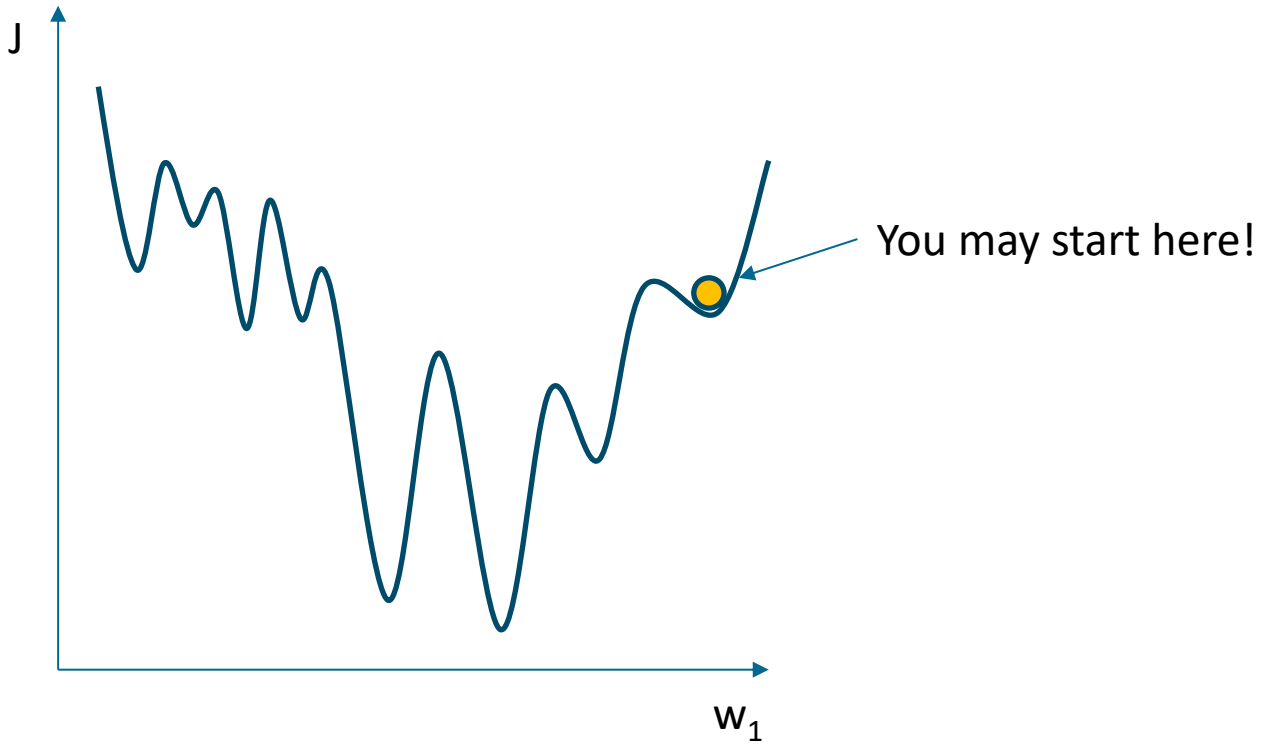
- When we initialize our model (typically with random parameters), we are in a random position of this bowl:





Cost function

- How can we get to the minimum?
- Consider that the typical profile of cost function is like this:



Gradient

- Given the initial position, we have to find the **best direction** along which we should step to change our weights
- The best direction is given by the **gradient** of the cost function
- Computing the gradient is like computing the "**slope"** of the function given a certain position
  - **Slope** → one-dimensional function, single number
  - **Gradient** → vector of numbers (parameters)

Gradient descent

- Let us define the **gradient** of the loss function *L* as:

$$\nabla_{\mathbf{W}} L(\mathbf{W}) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_N}\right]$$

Vector of partial derivatives

- We can use the gradient to update the parameters of the model in an iterative way:

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J(\mathbf{W})\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

step size
a.k.a. **learning rate**

Gradient descent

- Let us define the **gradient** of the loss function *L* as:

$$\nabla_{\boldsymbol{W}} L(\mathbf{W}) = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_N}\right]$$

Vector of partial derivatives

- We can use the gradient to update the parameters of the model in an iterative way:

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J(\mathbf{W})\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

???

Loss/cost function

- Do we apply gradient descent to the loss function or to the cost function?

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J(\mathbf{W})\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

- Remember that: $J = \frac{1}{N}\sum_{i=1}^{N} L_i$
- In practice, we compute $\nabla_{\boldsymbol{W}}$ for each sample and cumulate it
- In the end, we update by $\frac{\nabla_{\boldsymbol{W}}}{N}$ (linear operations)

Gradient descent

- **Batch gradient descent**

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J(\mathbf{W})\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

- **Stochastic gradient descent** (online, 1 random sample)

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J\left(\mathbf{W}, \boldsymbol{x}_{i,}\, y_i\right)\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

- **Mini-batch gradient descent** (mini-batches of *n* random samples)

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta \nabla_{\boldsymbol{W}} J\left(\mathbf{W}, \boldsymbol{x}_{i:i+n,}\, y_{i:i+n}\right)\Big|_{\boldsymbol{W}=\boldsymbol{W}_{t-1}}$$

What people now call “stochastic gradient descent” (SGD) is actually a “mini-batch gradient descent”

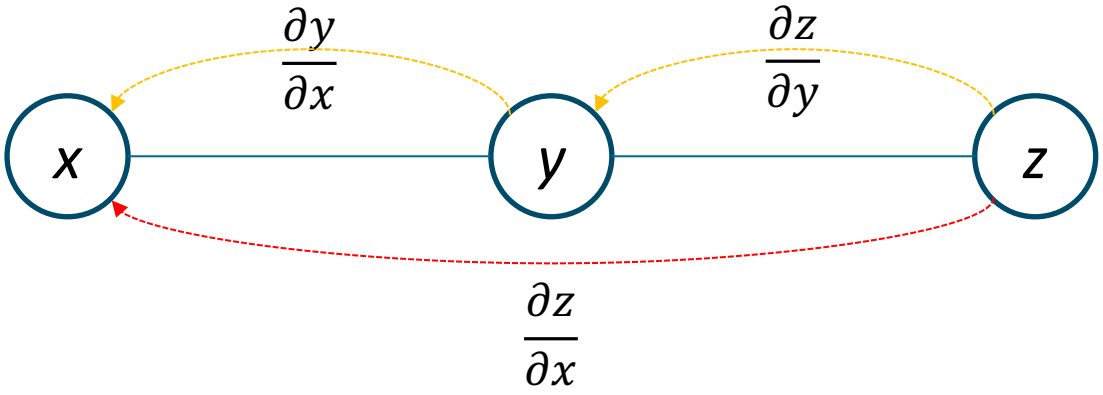
Backpropagation

- Algorithm to **compute gradients** of expression through recursive application of the chain rule

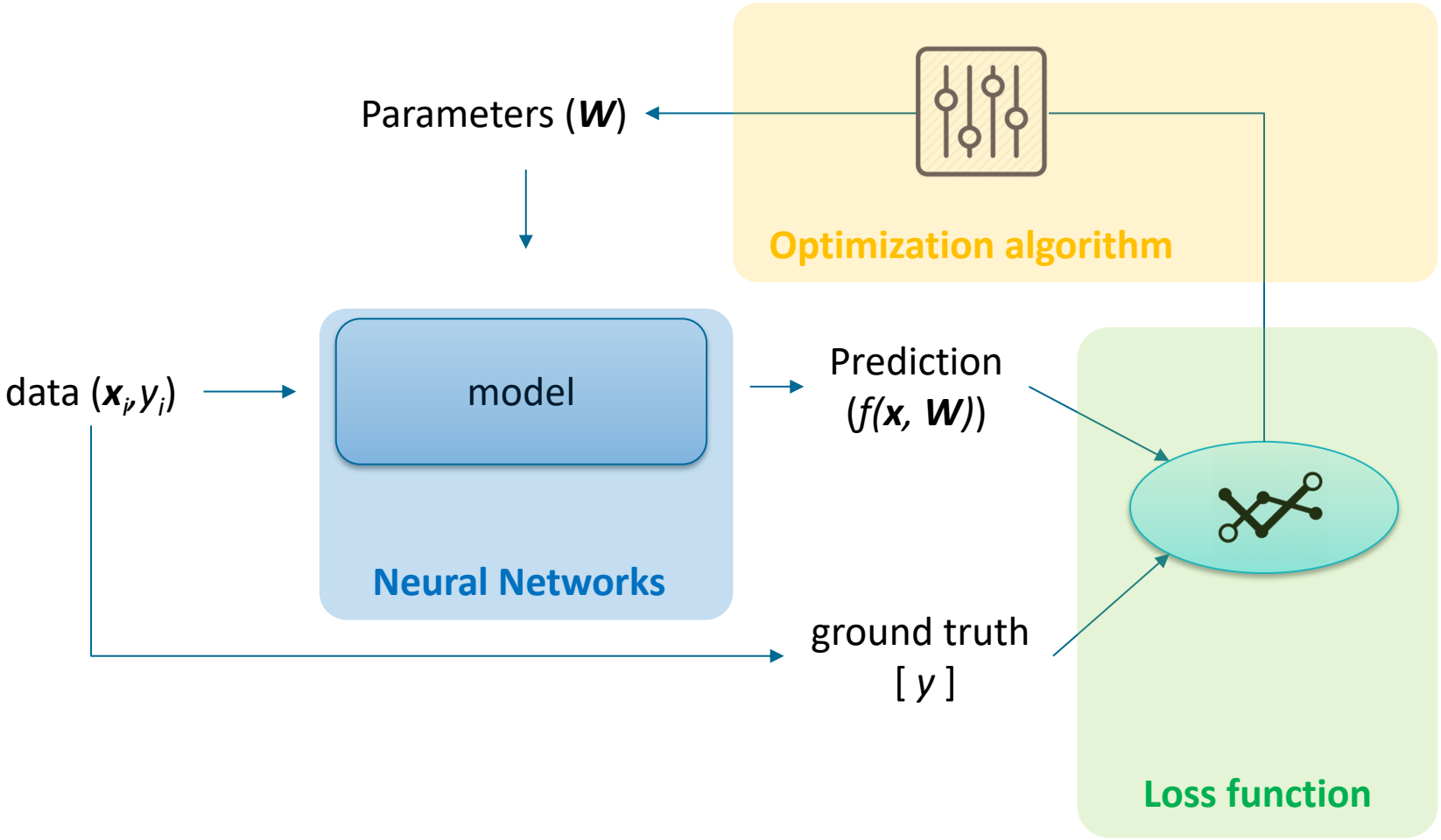
CHAIN RULE

- Given three variables, (*x,y,z*), such that *z = f(y)* and *y = f(x)*, it is possible to compute the derivative of *z* with respect to *x* as:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$



Supervised learning procedure



Training using mini-batches

- We select a subset of training samples
  - Mini-batch

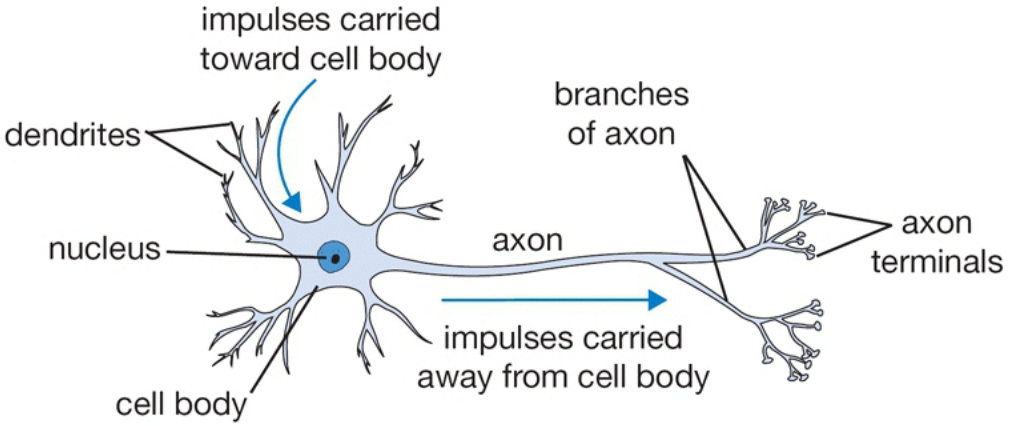
- Forward pass
  - Compute predictions
  - Compute loss
  - Compute cost as a combination of the losses (e.g., average)

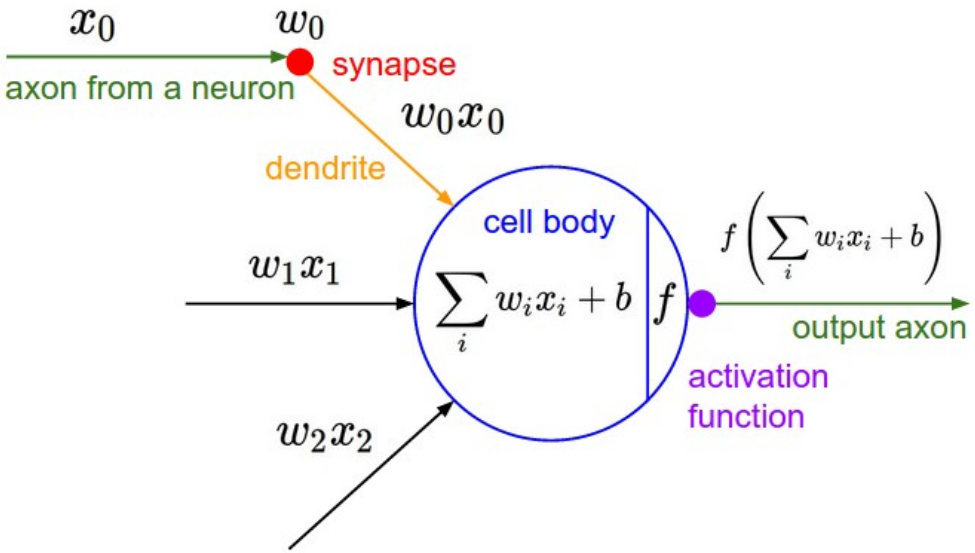
- Backpropagation
  - Compute derivatives

Break?

Neural Networks

Neural Networks - Inspiration

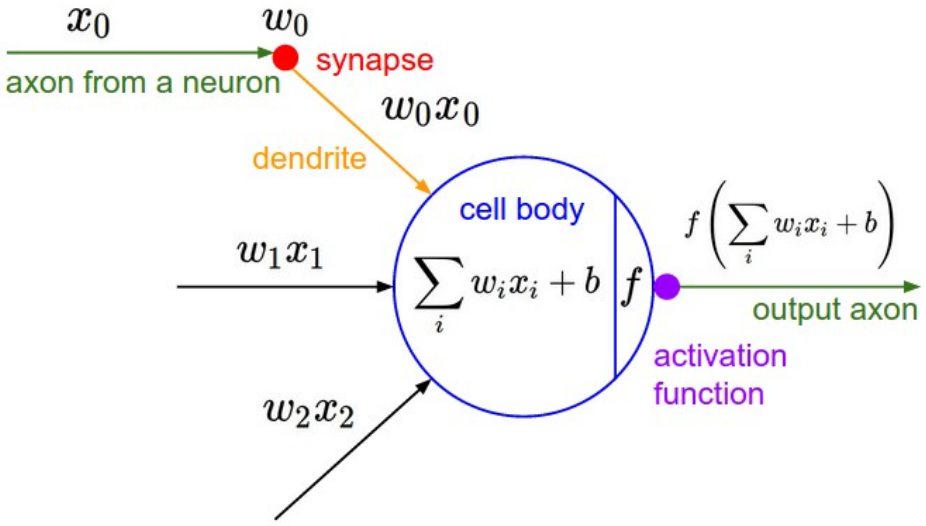




Neural Networks

- **Artificial Neuron**

$$y = f\left(\sum_i w_i x_i + b\right)$$



Neural Networks

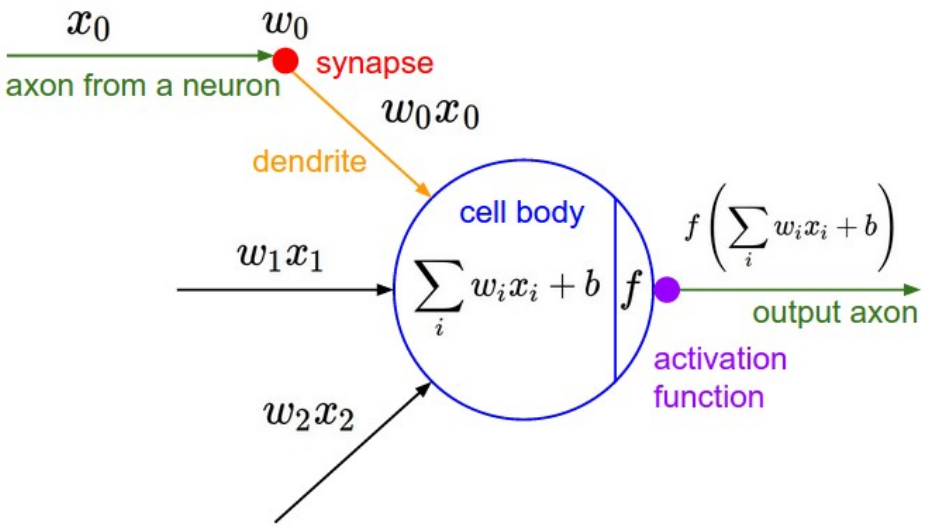
- **Artificial Neuron**

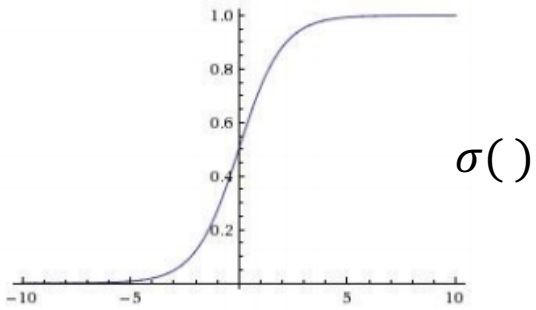
$$y = f\left(\sum_i w_i x_i + b\right)$$

weights bias

simplified

$$y = f(\boldsymbol{W}\boldsymbol{x})$$

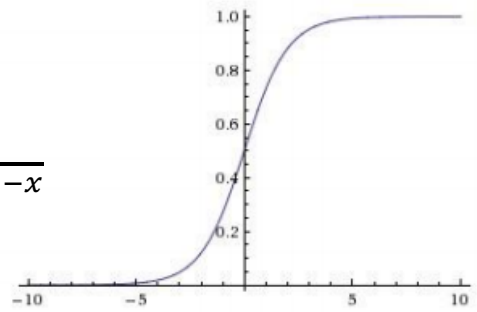




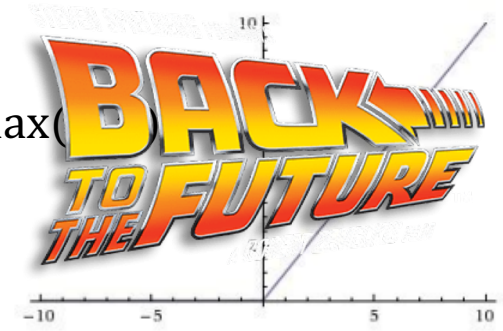
$\sigma(\,)$

Non linearity

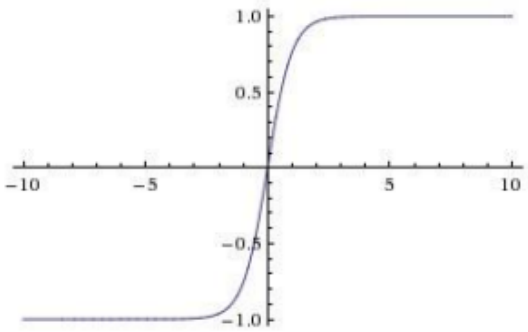
Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$



ReLu max(

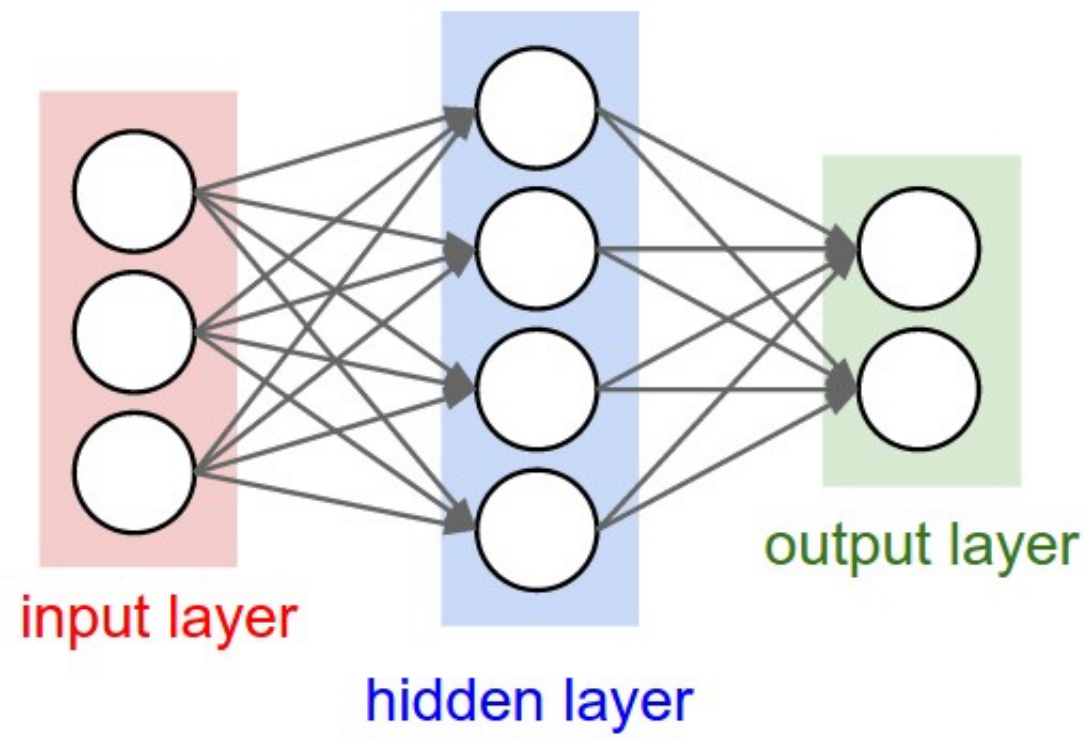


Tanh $Tanh(x)$



Neural Networks

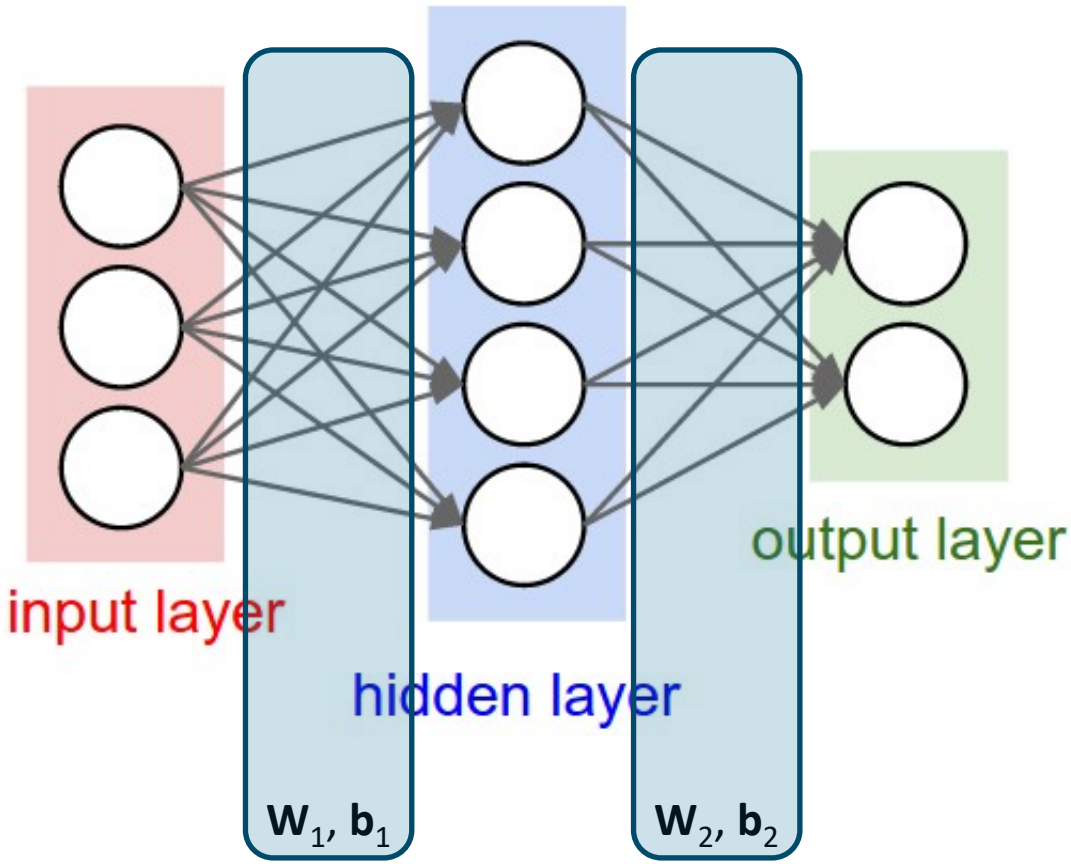
- More than one neuron and more than one layer



Two-layer
neural network

Neural Networks

- More than one neuron and more than one layer

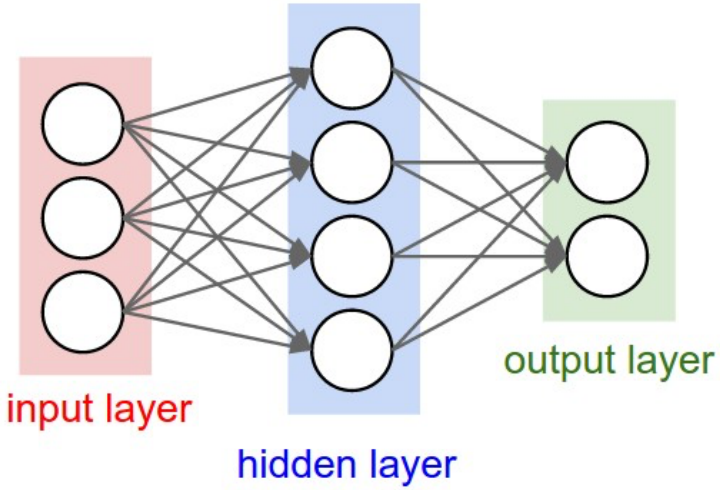


Two-layer neural network

$$f = \sigma(W2\sigma(W1x))$$

Neural Networks

- How many parameters?



Hidden layer

- 4 neurons connected to 3 inputs
- 4x3 weights + 4 biases = 16

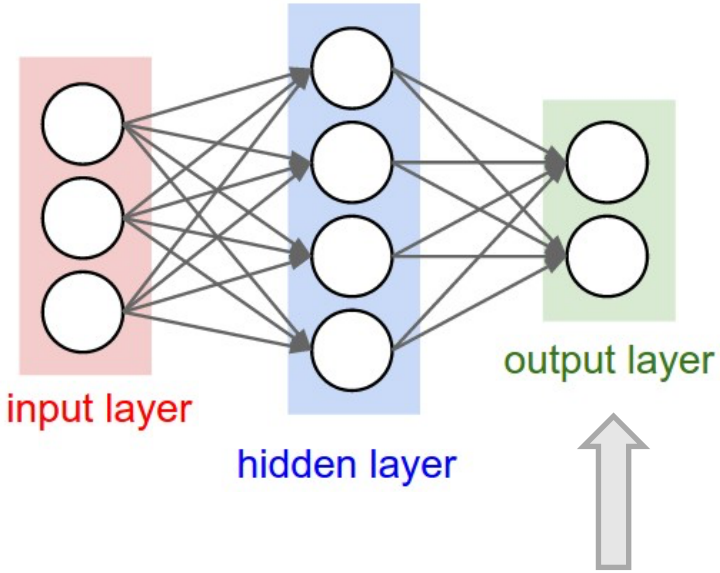
Output layer

- 2 neurons connected to 4 neurons
- 2x4 weights + 2 biases = 10

TOTAL = 26 parameters

Neural Networks

- How many parameters?



The number of neurons in the output layer depends on the classification problem (# classes)

Hidden layer

- 4 neurons connected to 3 inputs
- 4x3 weights + 4 biases = 16

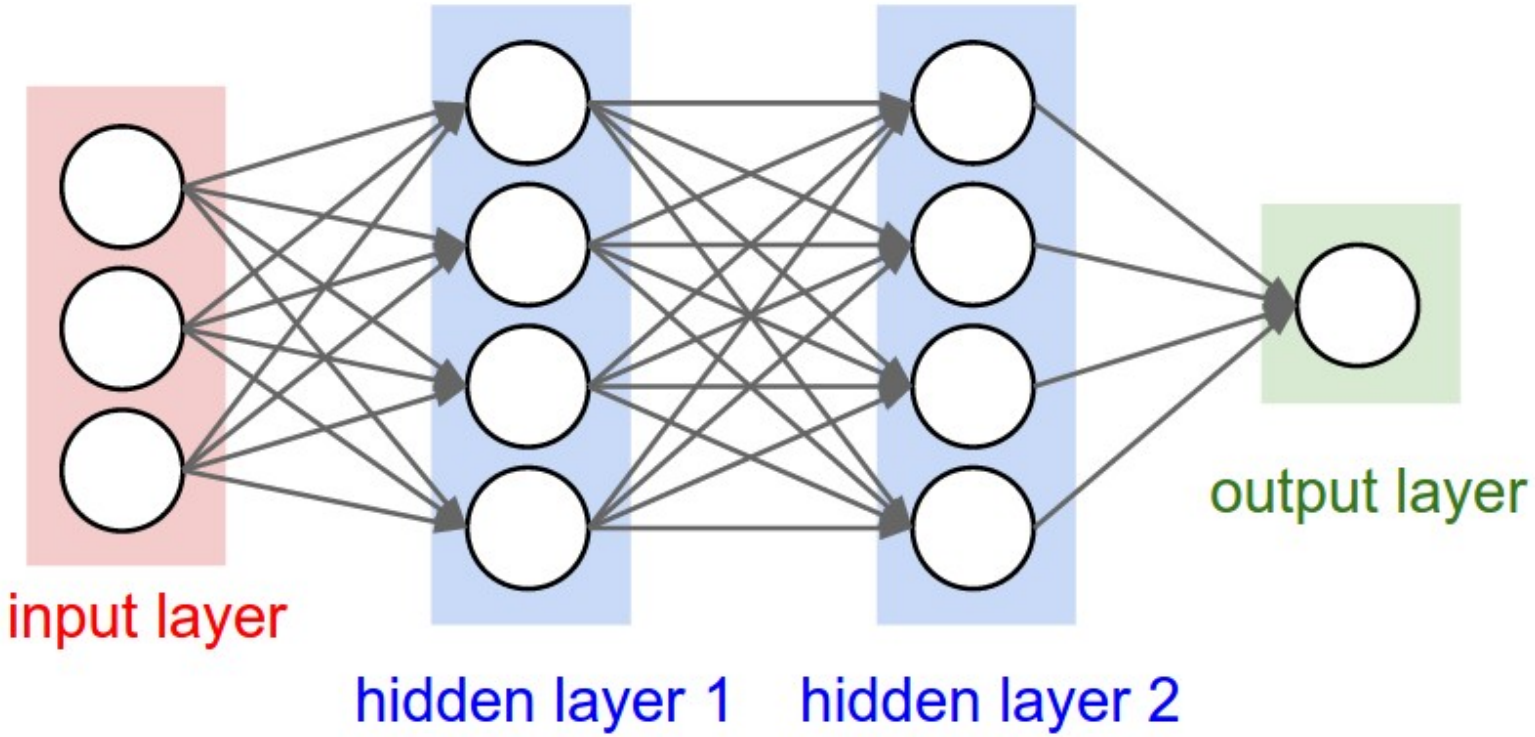
Output layer

- 2 neurons connected to 4 neurons
- 2x4 weights + 2 biases = 10

TOTAL = 26 parameters

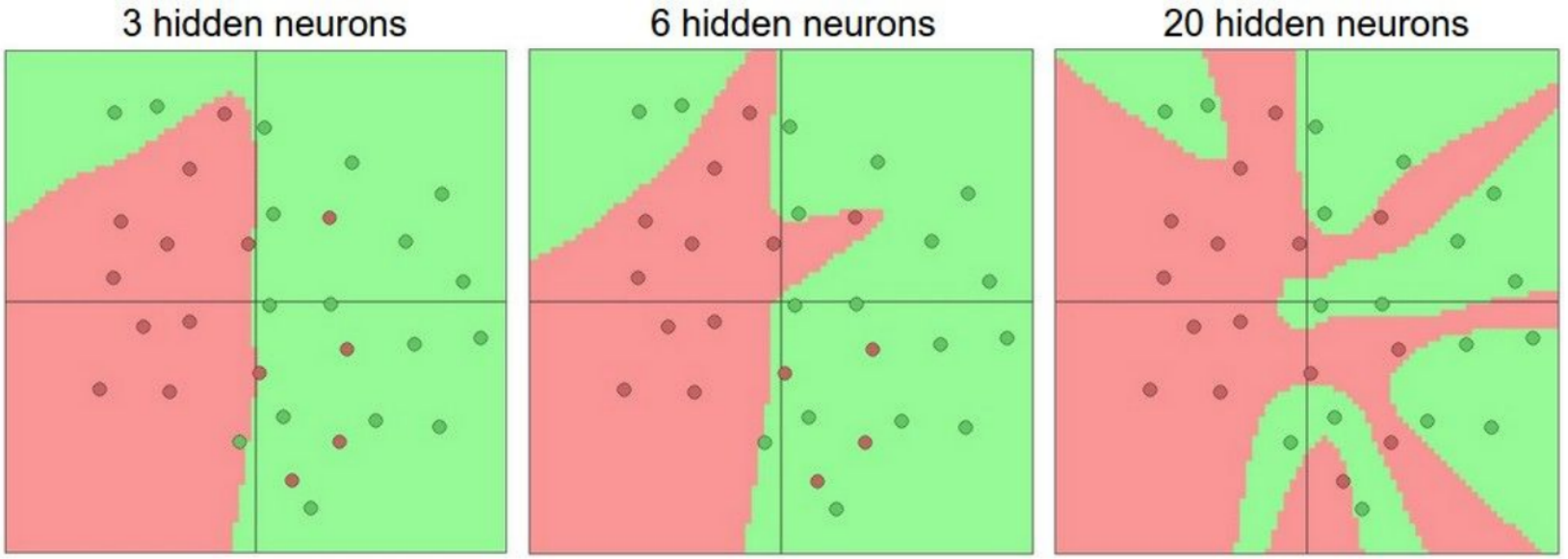
Neural Networks

- $f = \sigma(W3\sigma(W2\sigma(W1x)))$



Neurons in hidden layer

Demo = http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html



more neurons = more capacity

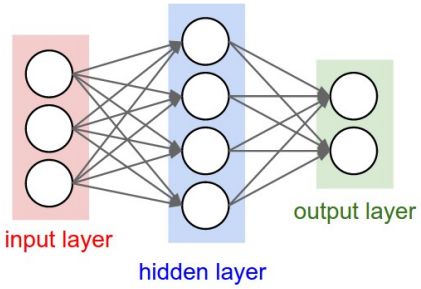
Non-linearity

- What if we do not incorporate non-linearity?

- $f = \underbrace{W_2(\overbrace{W_1 x}^{1^{st}\text{ layer}})}_{2^{nd}\text{ Layer}} = (W_2 W_1)x = W'x$ Still a linear transformation!!

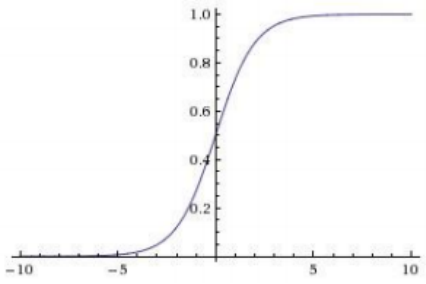
Output layer

- The desired output for a multi-class classification problem is:
  - One classification score per class



Achieved by using one neuron per class

  - Each score indicates the probability for its class ($P \in [0,1]$)



A sigmoid function guarantees output in the range [0,1]

Softmax

- What if we have more than one class? We use Softmax!
- Softmax is a generalization of logistic regression (sigmoid function) to multiple classes
- If we have K neurons in the output layer, the probability for class with index *i* is, based on softmax:

$$P_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Always positive

Does the normalization

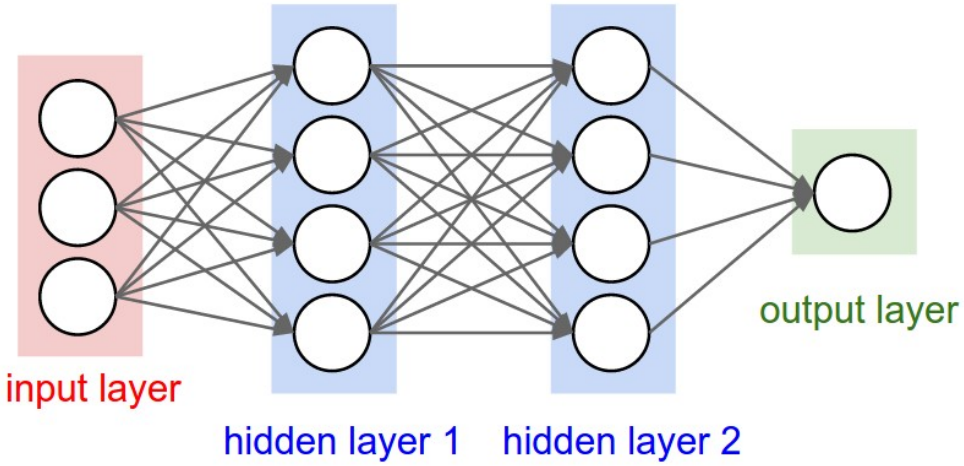
Training a Neural Networks

How to

Radboudumc

1. Define an architecture

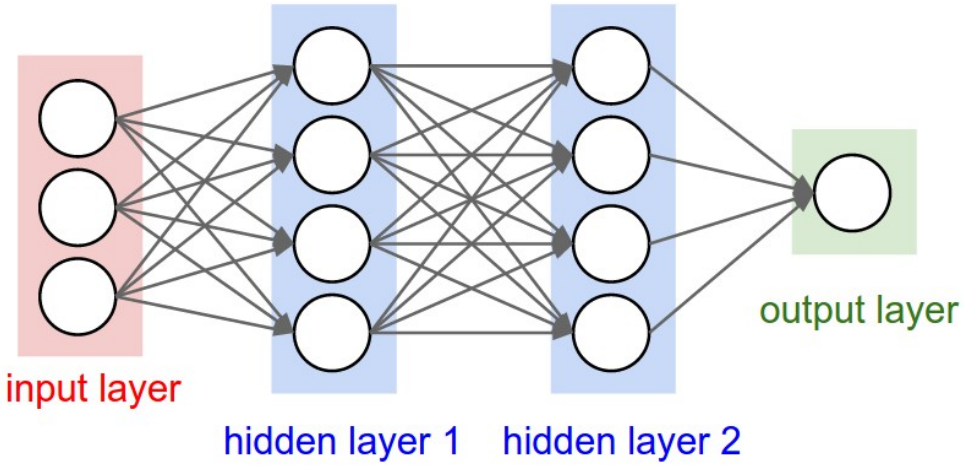
- Let's pick this architecture and count its parameters



Layer	# input connections	# output connections	# parameters
Input layer	N/A	3	0
Hidden layer 1	4 * 3	3	12 + 4 = 16
Hidden layer 2	4 * 4	4	16 + 4 = 20
Output layer	4	1	4 + 1 = 5

2. Initialization

- How to initialize the parameters that have to be trained?



Layer	# input connections	# output connections	# parameters	Initial values
Input layer	N/A	3	0	N/A
Hidden layer 1	4 * 3	3	12 + 4 = 16	???
Hidden layer 2	4 * 4	4	16 + 4 = 20	???
Output layer	4	1	4 + 1 = 5	???

Initialization

- How do we initialize **weights** and **biases**?

1. **All parameters set to zero**
   - When we compute the gradient, all parameters will get the same update!

2. **Random initialization with normal distribution**
   - Zero mean and unit standard deviation

The variance of the output of a neuron increases with the number of inputs → we can normalize the variance accordingly:

```
w = np.random.randn(n) / sqrt(n)
```

number of inputs for that neuron

Initialization

- This idea has been further developed, even recently:
  - Glorot10
  - He15

glorot_normal

```
glorot_normal(seed=None)
```

Glorot normal initializer, also called Xavier normal initializer.

It draws samples from a truncated normal distribution centered on 0 with `stddev = sqrt(2 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

Arguments

- **seed**: A Python integer. Used to seed the random generator.

Returns

An initializer.

References

Glorot & Bengio, AISTATS 2010 - **http**://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

he_normal

```
he_normal(seed=None)
```

He normal initializer.

It draws samples from a [illegible] with `stddev = sqrt(2 / fan_in)` where `fan_in` is the number [illegible]

Arguments

- **seed**: A Python integer. Used to seed the [illegible]

Returns

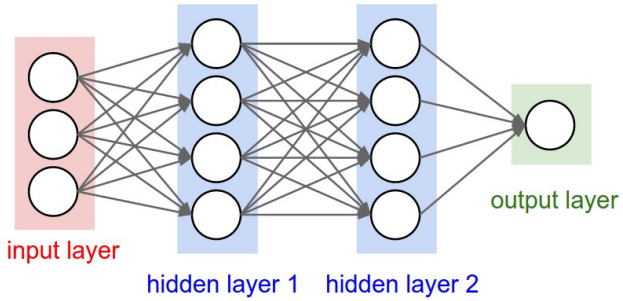
An initializer.

References

He et al., http://arxiv.org/abs/1502.01852

3. Training and validation set

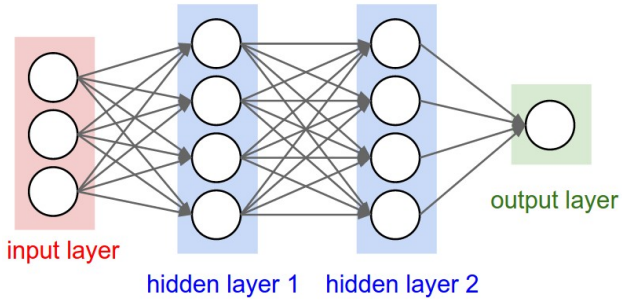
- We need data to train our network -> training set
- We need data to assess its performance -> validation set

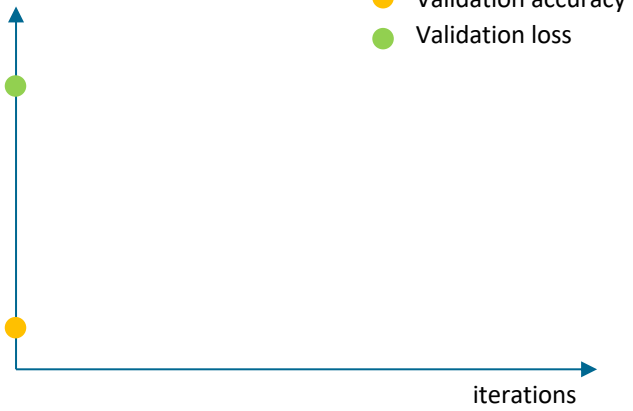


SHUFFLE YOUR TRAINING SET BEFORE TRAINING!

3b. Initial validation performance

- We need data to train our network -> training set
- We need data to assess its performance -> validation set

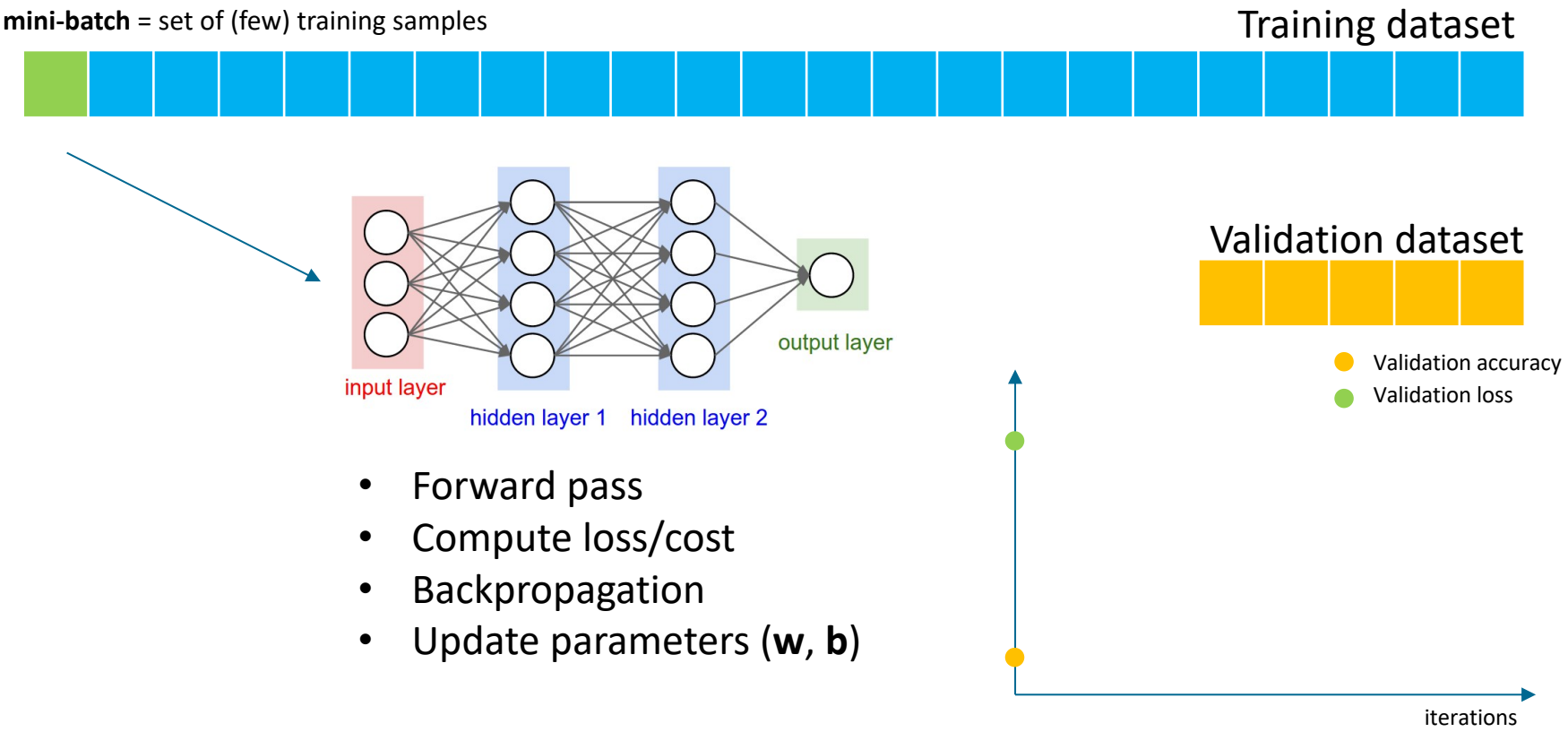




Computing **performance** on the validation set (before training) gives the performance of random parameters!

4. Training

- We train neural networks with **mini-batch** gradient descent

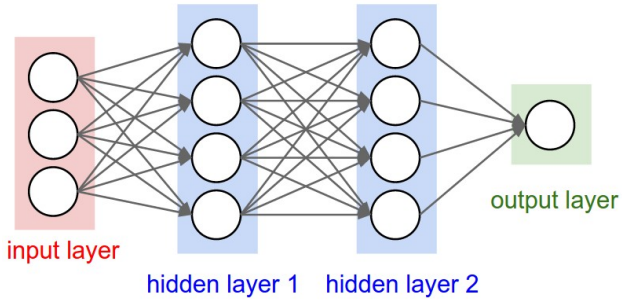


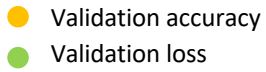
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)

4. Training

- We train neural networks with **mini-batch** gradient descent

mini-batch = set of (few) training samples





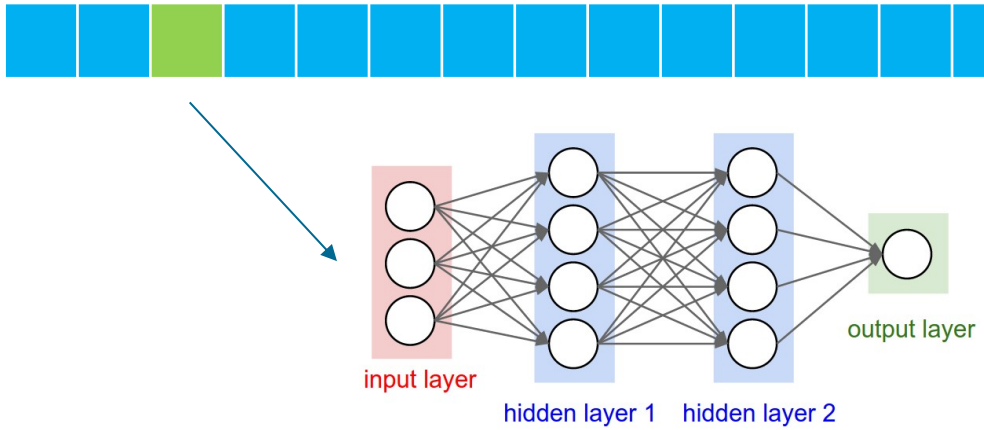
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)

4. Training

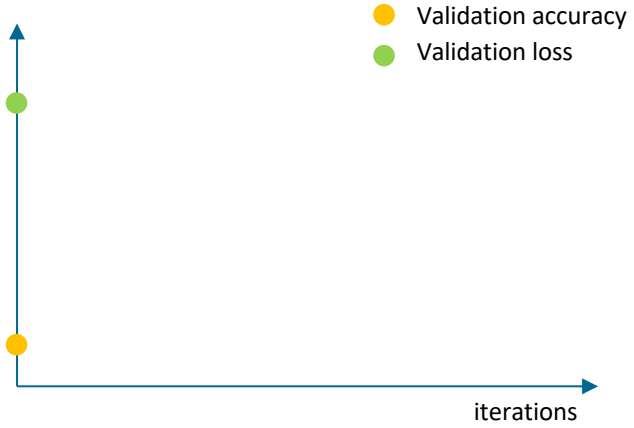
- We train neural networks with **mini-batch** gradient descent

mini-batch = set of (few) training samples

Training dataset



Validation dataset

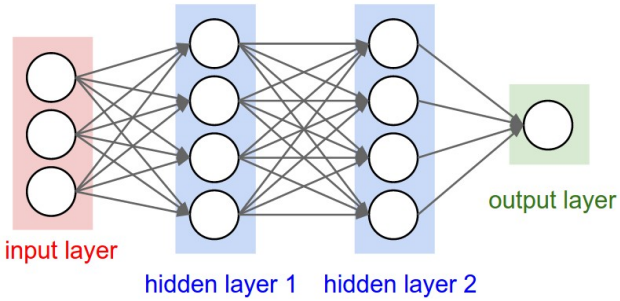


- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)

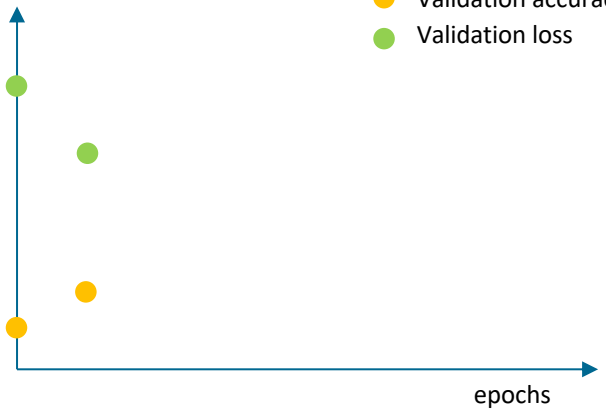
4. Training

- When the network has seen all the mini-batches, it has done an **epoch**

mini-batch = set of (few) training samples



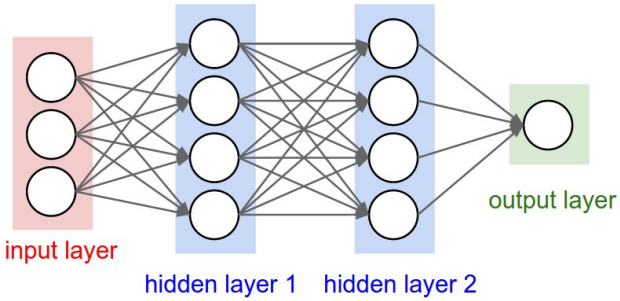
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)
- Apply to validation set

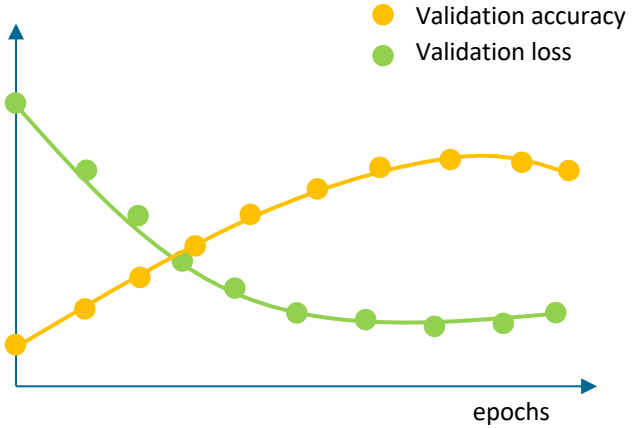


4. Training

- Repeat this several epochs

mini-batch = set of (few) training samples



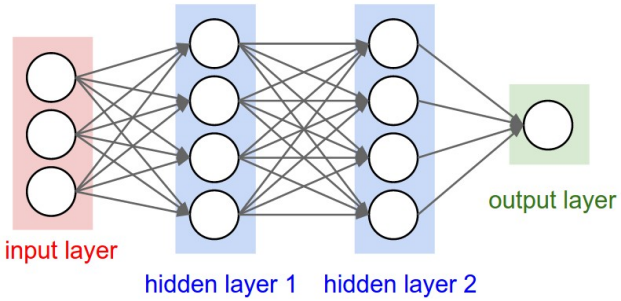


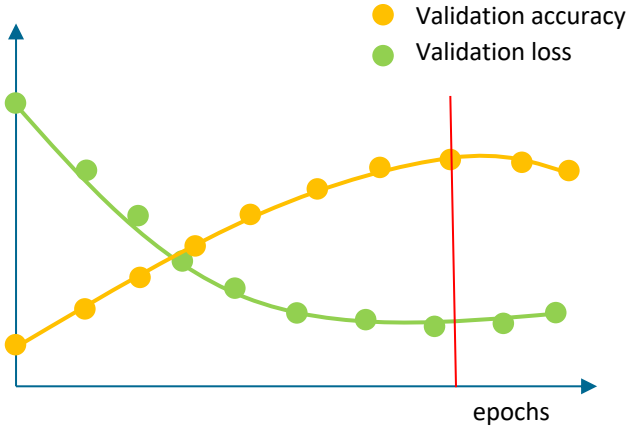
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)
- Apply to validation set

4. Training

- Save the parameters of the network at each iteration
- Pick the ones that maximize the accuracy (for example)

mini-batch = set of (few) training samples



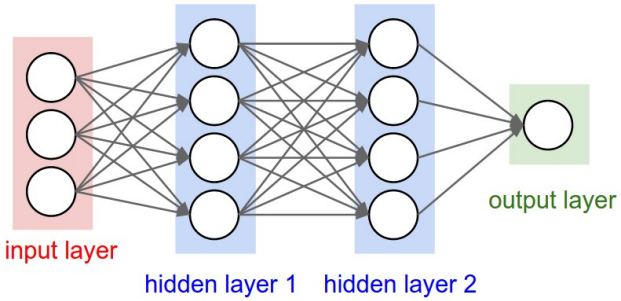


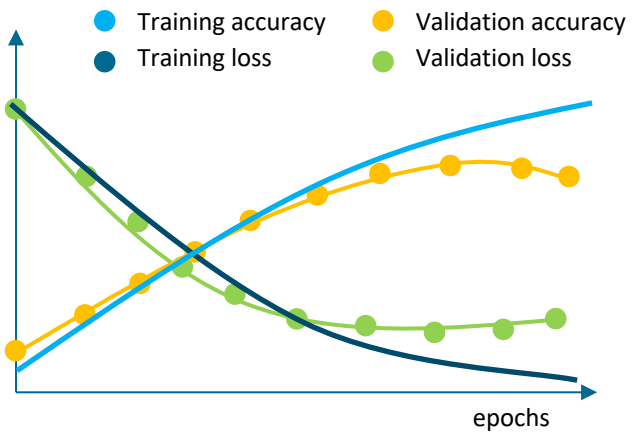
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)
- Apply to validation set

4. Training

- During training you also monitor **training loss** and **accuracy**

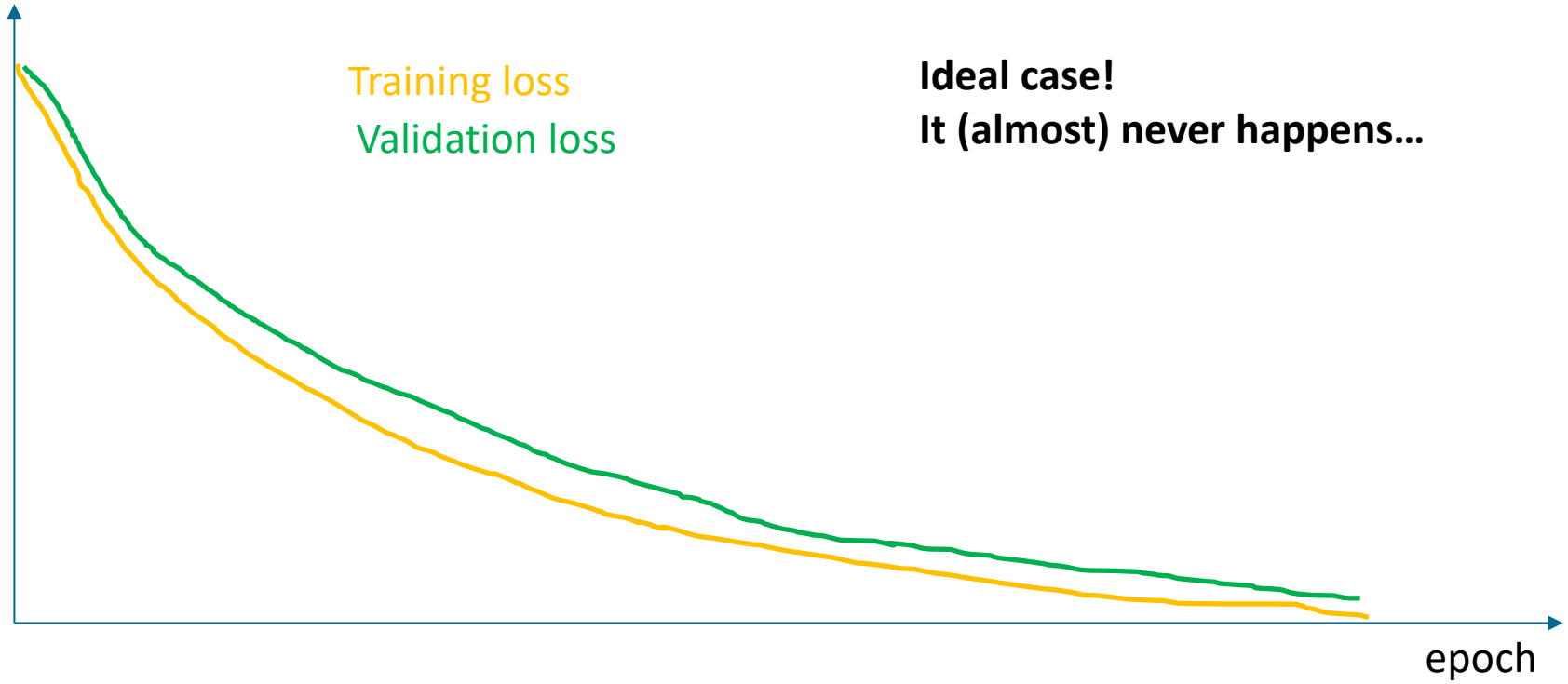
mini-batch = set of (few) training samples





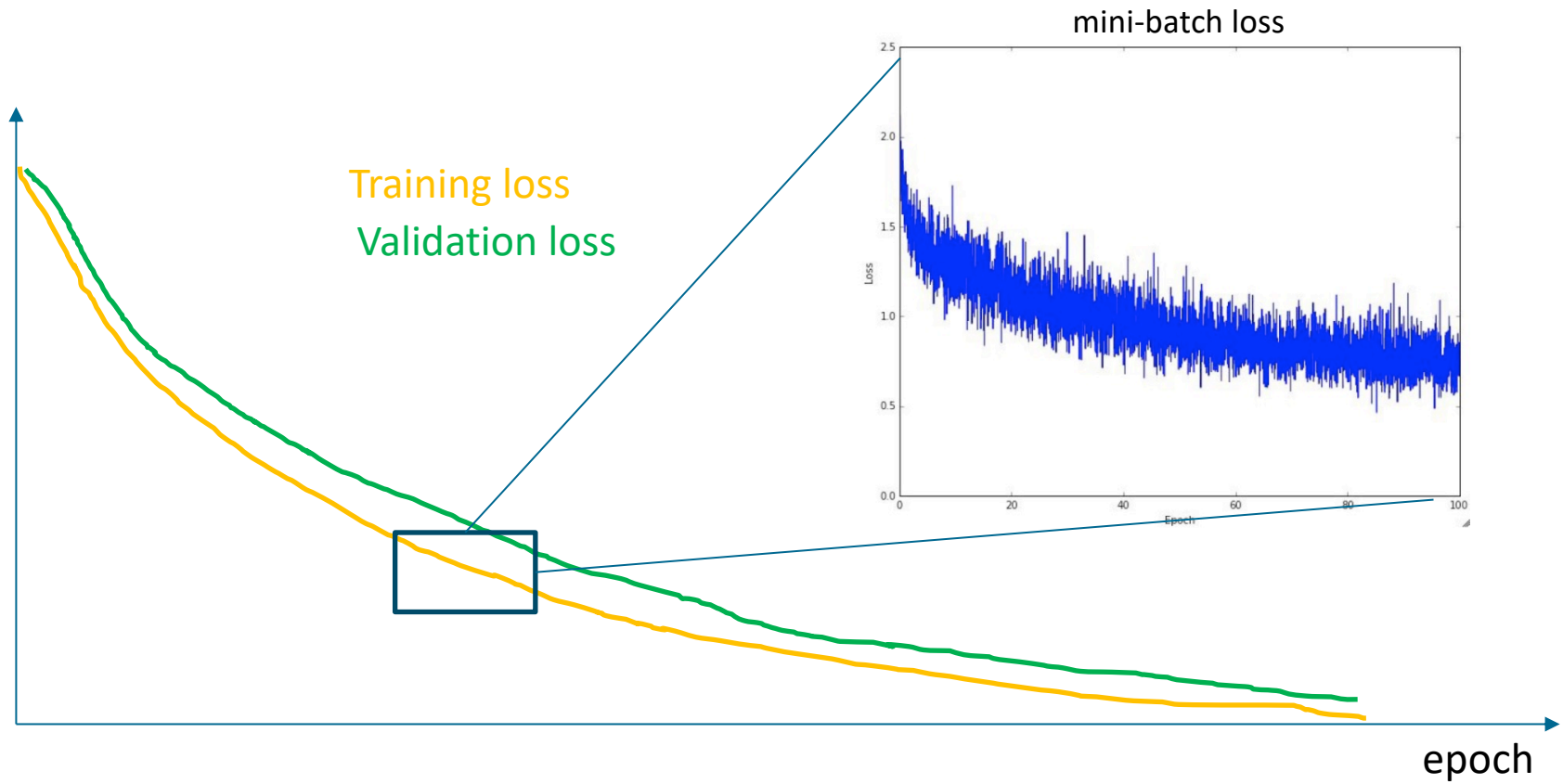
- Forward pass
- Compute loss/cost
- Backpropagation
- Update parameters (**w**, **b**)
- Apply to validation set

Learning curves

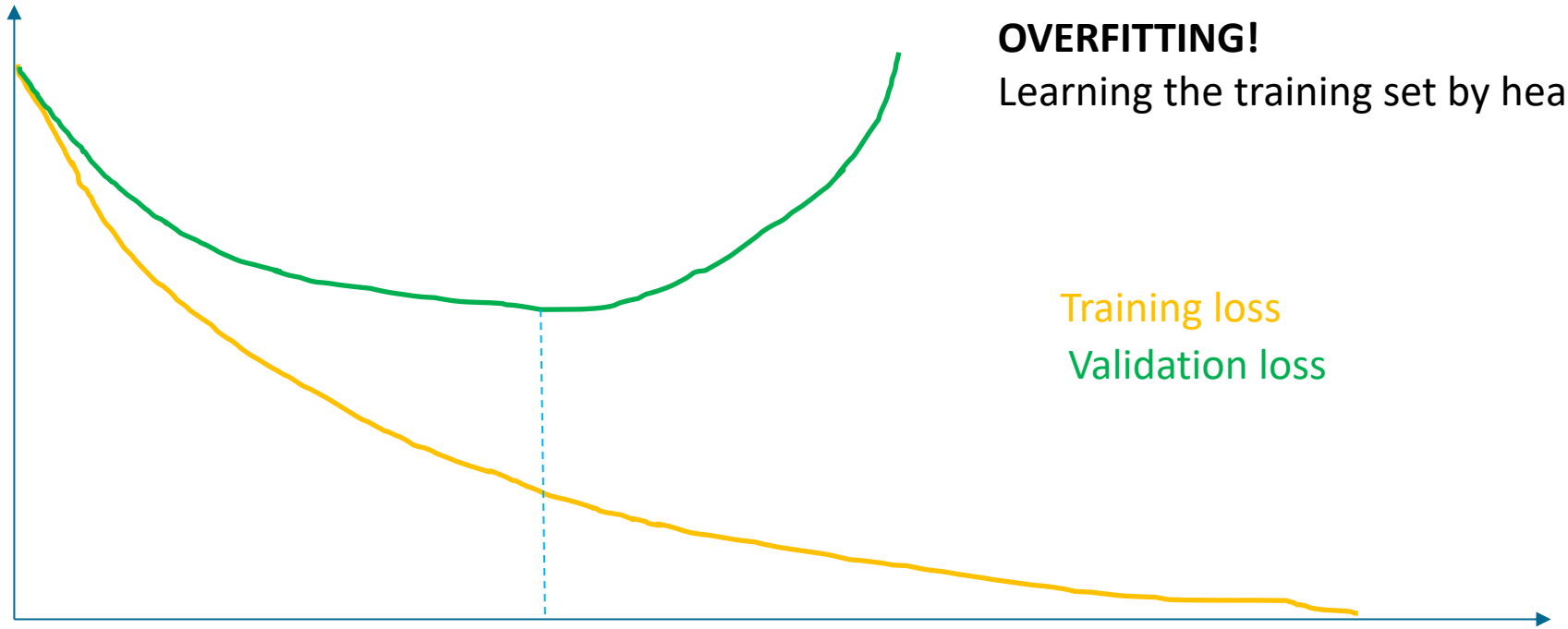


Ideal case!
It (almost) never happens…

Learning curves



Learning curves

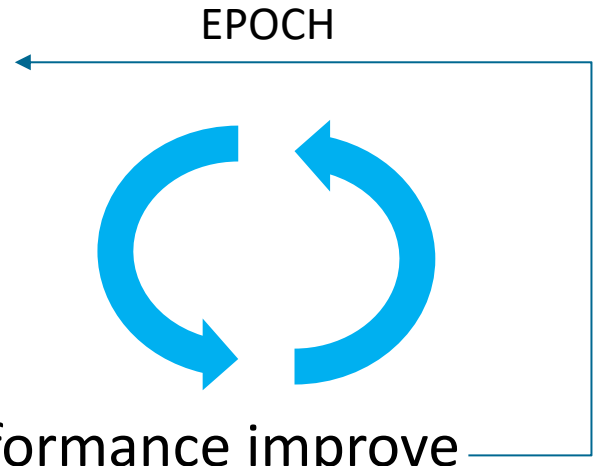


OVERFITTING!
Learning the training set by heart

Save the parameters that you have in this epoch
It's the best you can do with these settings!

Training overview

1. Define network architecture
2. Initialize parameters
3. Define training and validation sets
4. Do training
    a. Sample training mini-batches
    b. Forward pass
    c. Backpropagation
    d. Update parameters
    e. Apply on validation set
    f. Save network parameters if performance improve

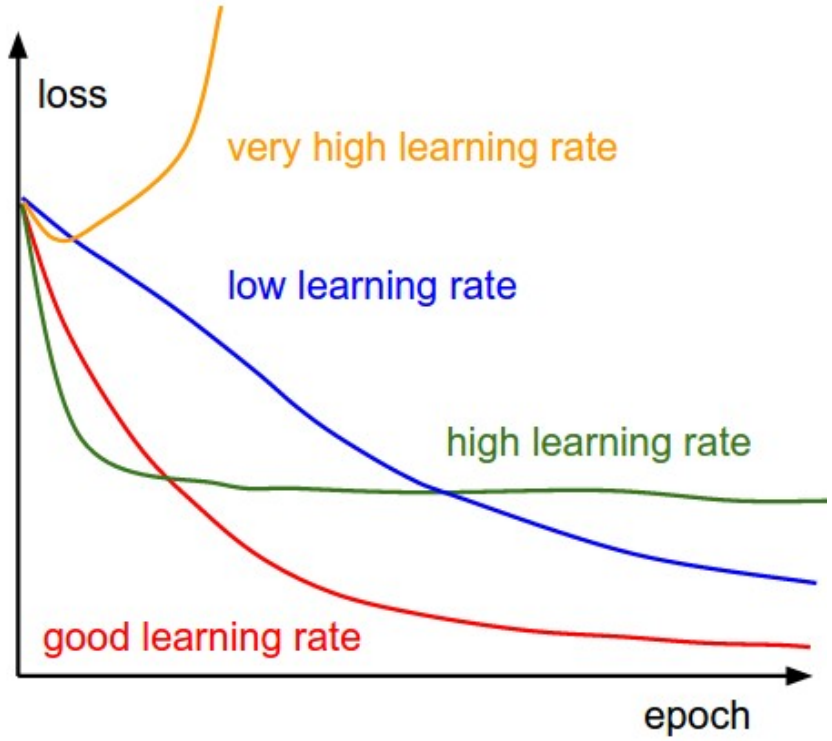


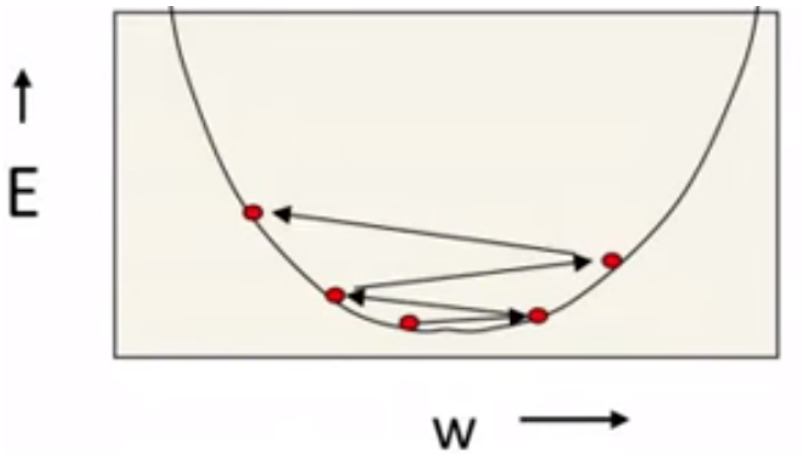
Design choices

1. Define network architecture ← # layers, # neurons
2. Initialize parameters ← algorithm
3. Define training and validation sets ← data split
4. Do training
   a. Sample training mini-batches ← mini-batch size
   b. Forward pass
   c. Backpropagation ← loss function, regularization
   d. Update parameters ← Learning rate, update rule (SGD)
   e. Apply on validation set
   f. Save network parameters if performance improve

Learning rate

- Learning rate is an important parameter
  - **Too low: very slow convergence**
  - **Too high: No convergence!**





Learning rate

- How do we pick a good learning rate?
- In practice, we only pick the **initial one**, then we implement a strategy to decrease it over the epochs
  - Linear decrease
  - Exponential decrease
  - …
- You can decrease it every M epochs, or when the performance of the network doesn't improve for N epochs

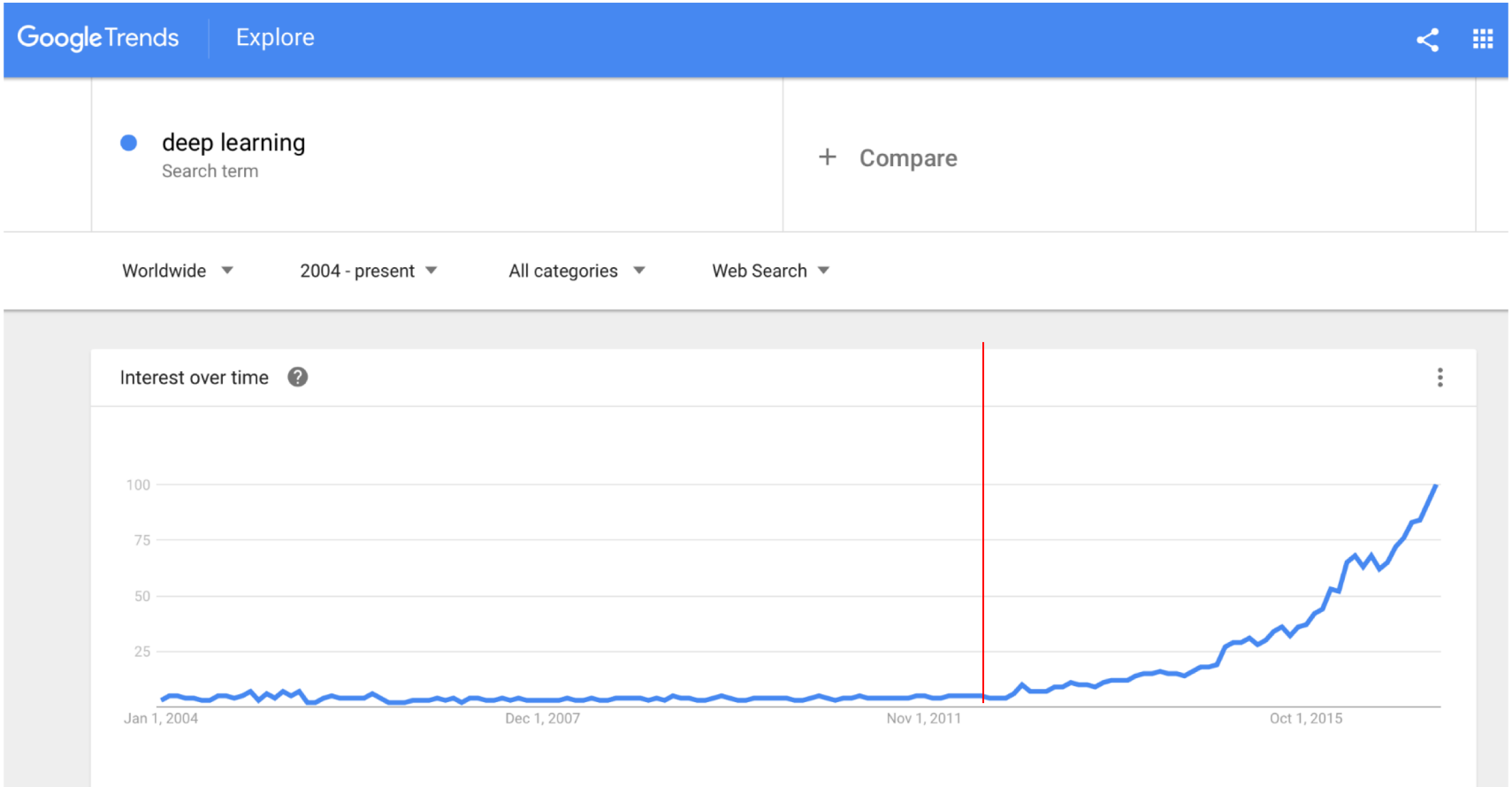
Useful tricks and sanity checks

- **Sanity checks**
  - Train and validate on your (small) training set -> you should get loss ≈ 0 and accuracy ≈ 1 (over-fit training set)
  - Run your network just initialized on the validation set. If balanced, you should know the accuracy that you get

- **Tricks**
  - Make sure your batches are balanced
  - Make a balanced validation set, if accuracy is the target
  - **Shuffle your dataset**!
  - Penalize samples in the loss function in a different way

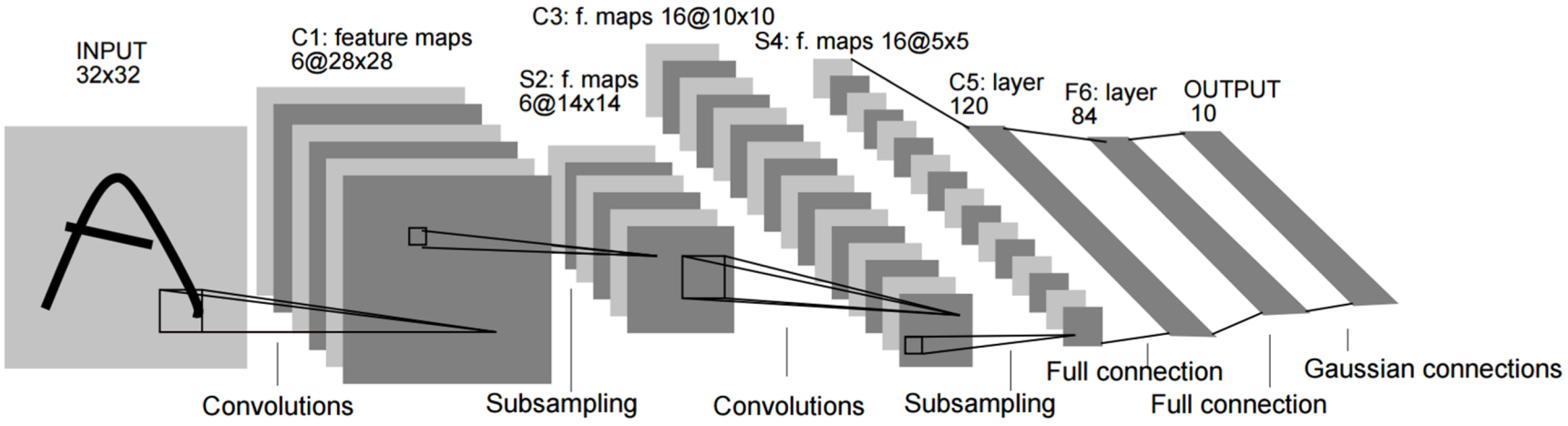
Convolutional Neural Networks

What happened in 2012?



Yann LeCun, 1998

- **LeNet5**
  - Used backpropagation
  - Used stochastic gradient descent



Lots of data

IMAGENET

14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications CoolStuff About

Not logged in. Login | Signup

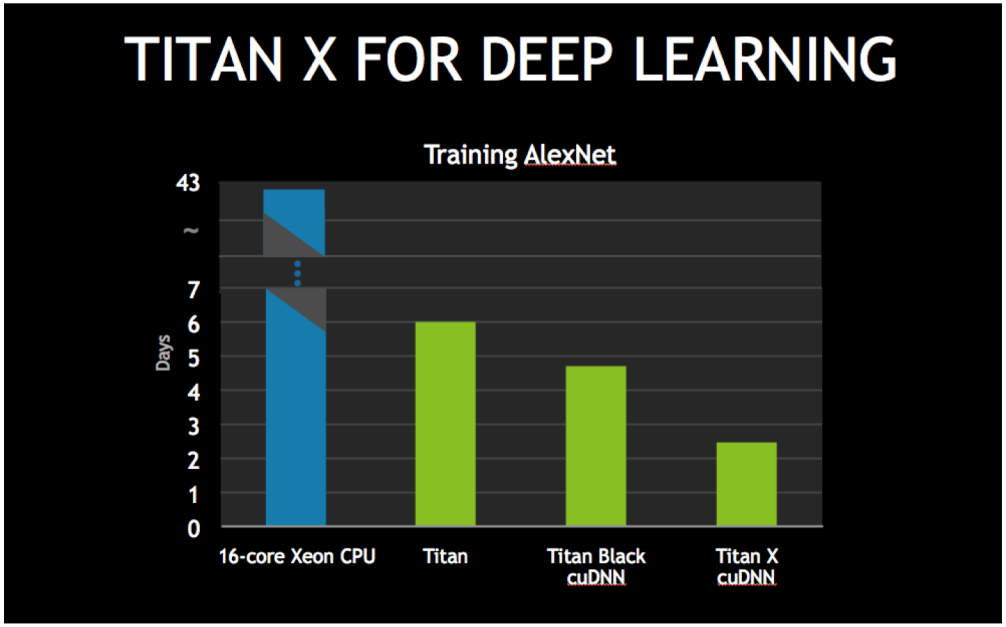
ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.

Click here to learn more about ImageNet, Click here to join the ImageNet mailing list.

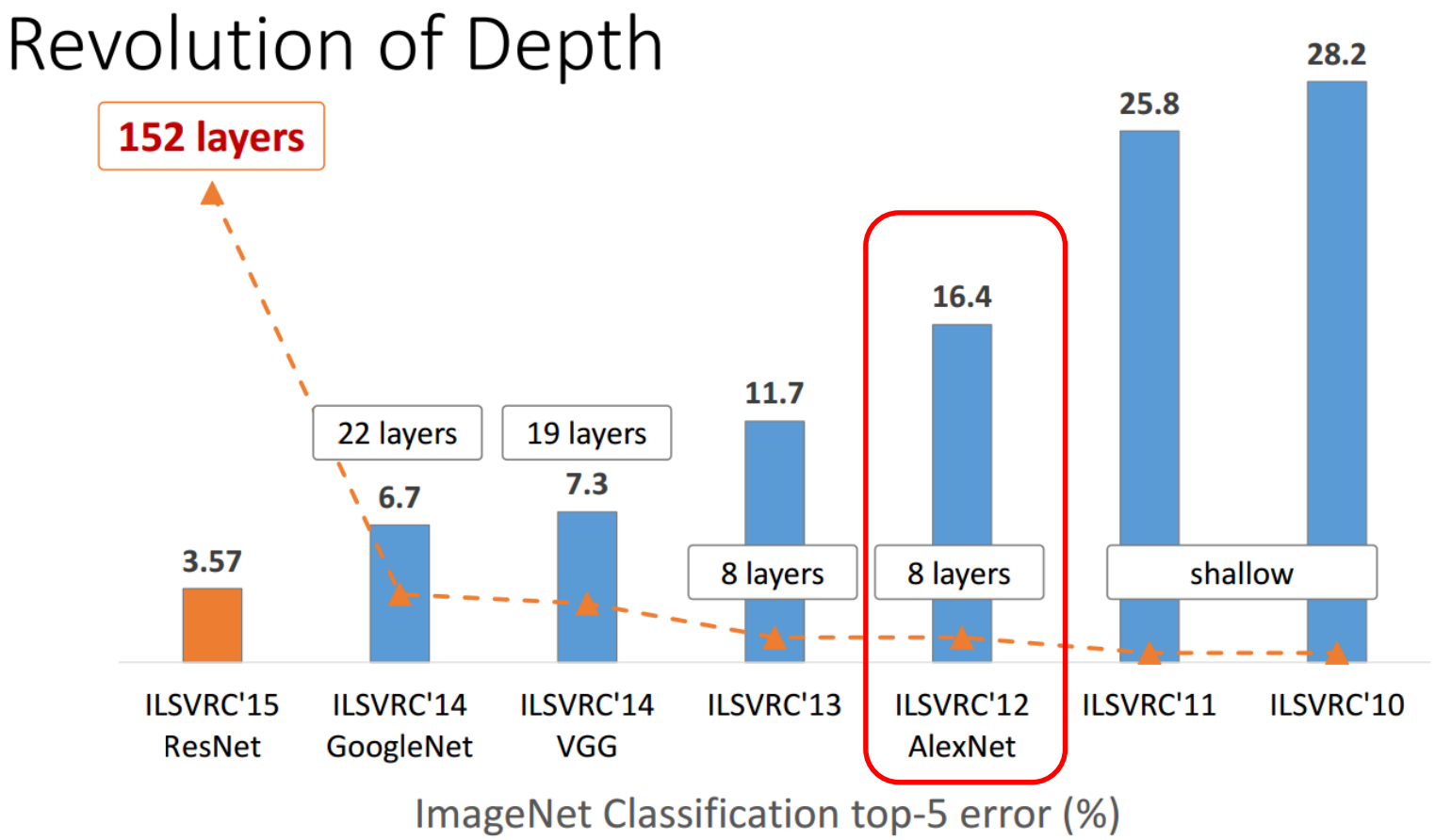
What do these images have in common? *Find out!*

GPUs


GEFORCE GTX
TITAN

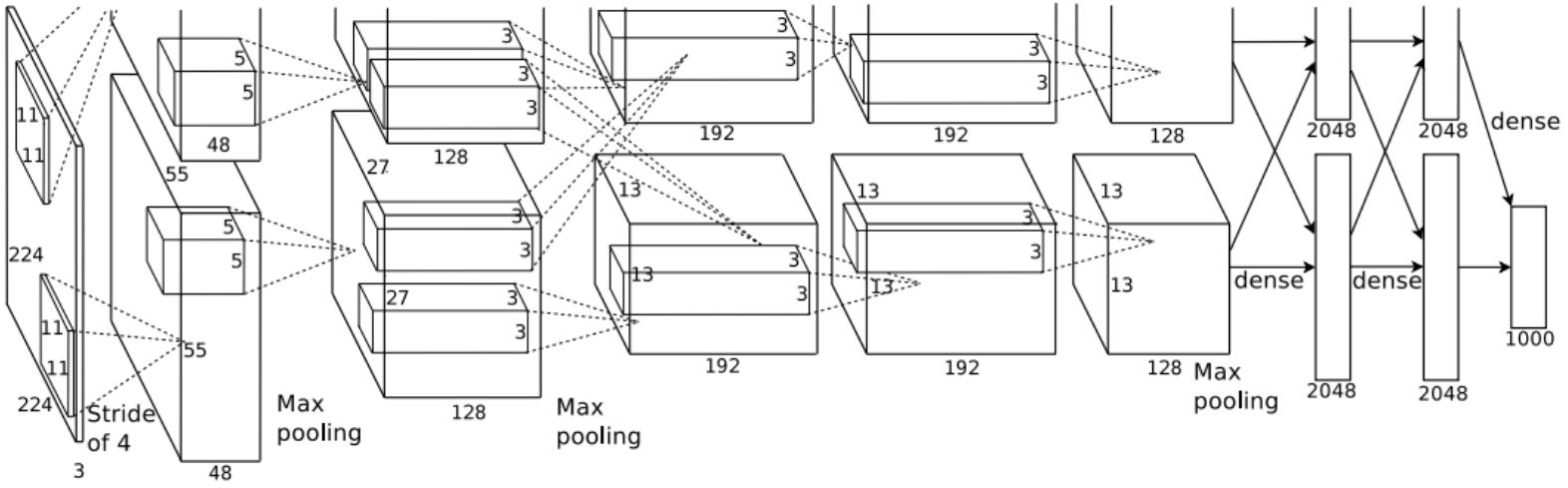

TITAN X FOR DEEP LEARNING
Training AlexNet
Days
43
~
7
6
5
4
3
2
1
0
16-core Xeon CPU
Titan
Titan Black cuDNN
Titan X cuDNN

In 2012 (and later…)



Alex Khrizevsky, 2012

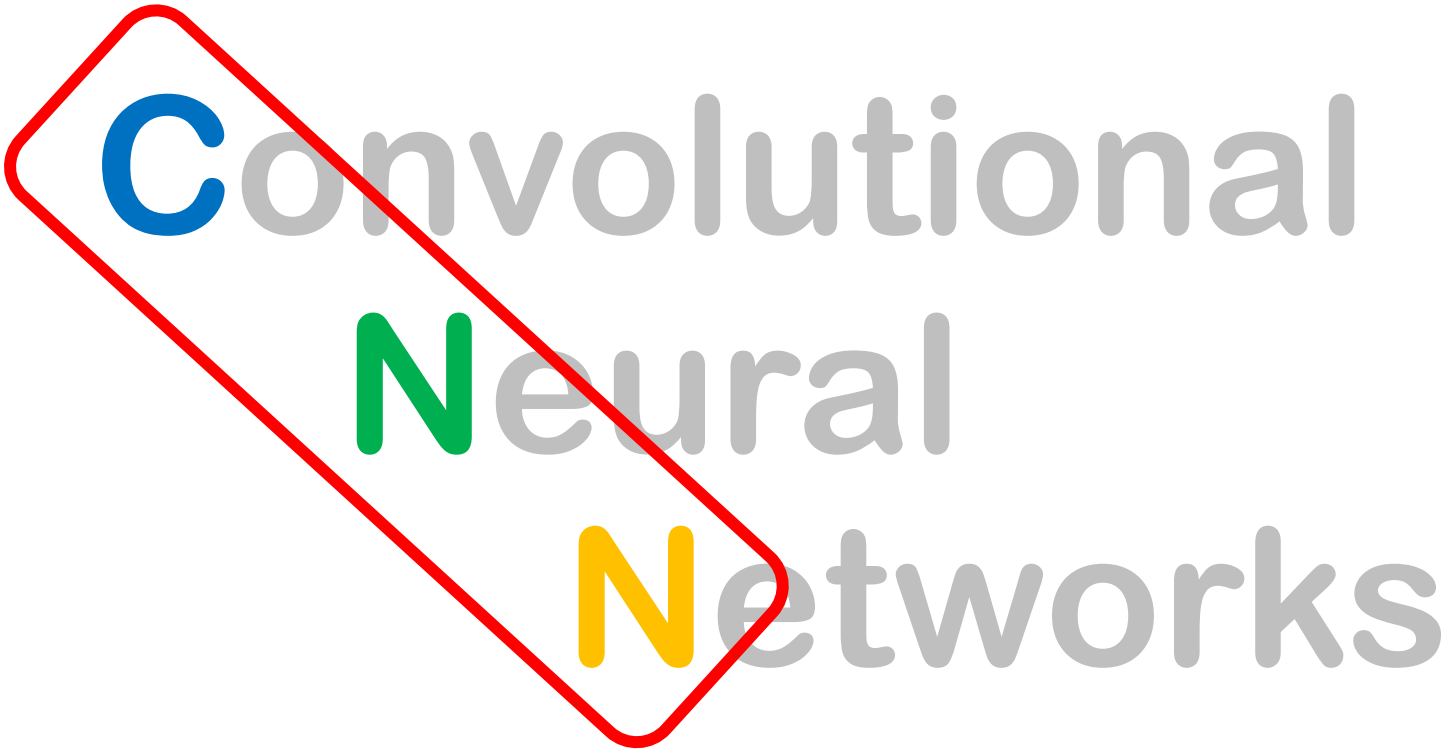
- **AlexNet**
  - Used backpropagation, SGD, ReLU
  - Used Dropout
  - Used (2x)GPU
  - Used 7 layers
  - Used ImageNet data (1.5M images, 1k classes)



Convolutional

Convolutional Neural

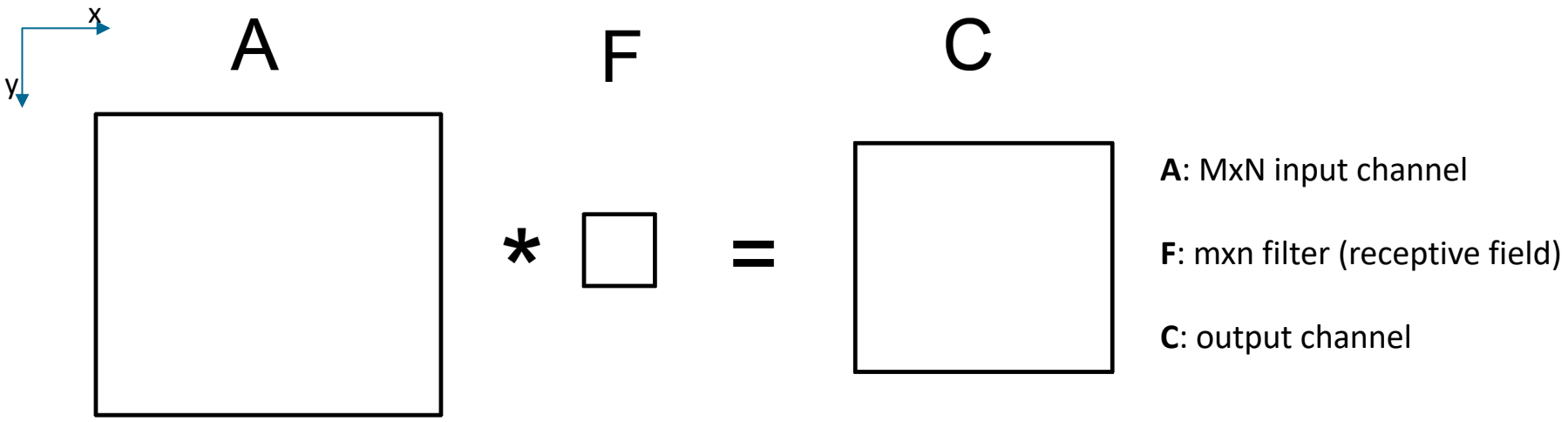
Convolutional Neural Networks



CNN ≠ CNN ConvNets

Convolutional Neural Networks

Convolutional Neural Network

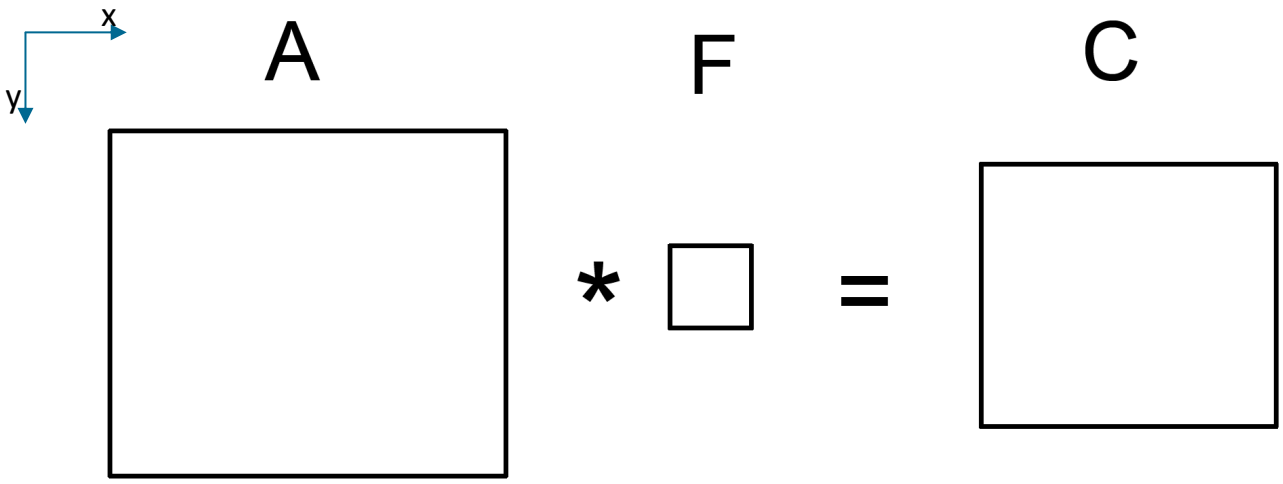


Convolution: C = A*F

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network



A: MxN input channel

F: mxn filter (receptive field)

C: output channel

Convolution: C = A*F

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y) F(x,y)$$

associative: (A*(B*F) = (A*B)*F)

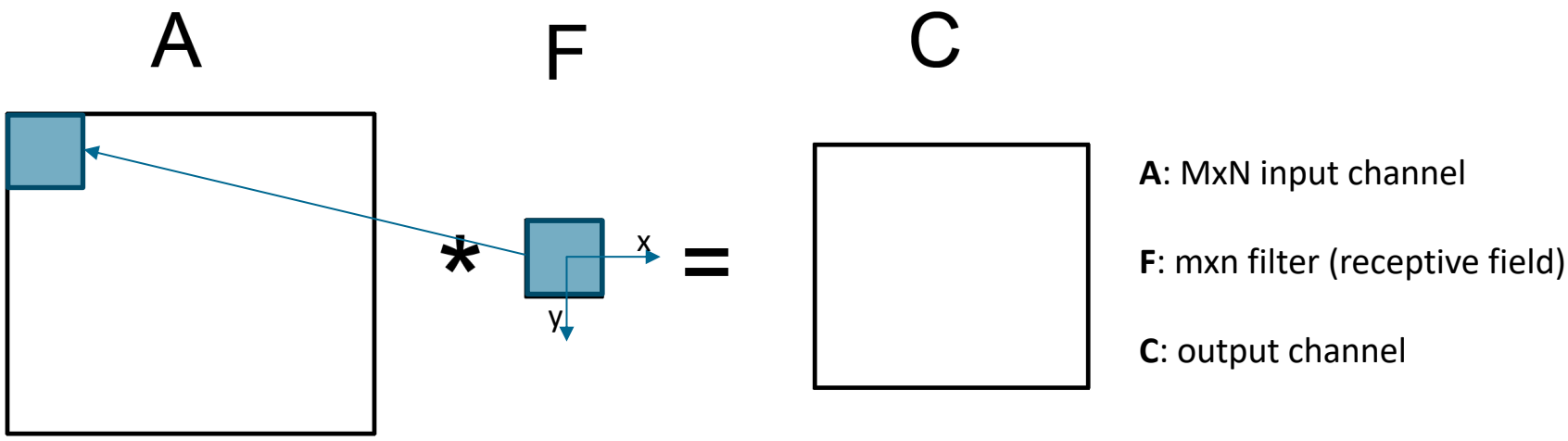
Correlation

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i+x, j+y) F(x,y)$$

not associative

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

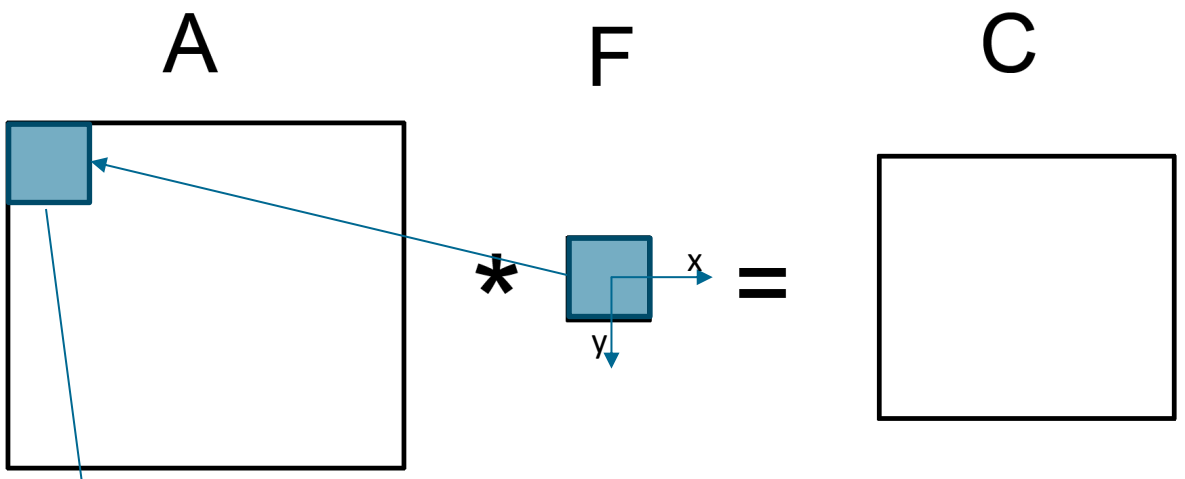


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network



A: MxN input channel

F: mxn filter (receptive field)

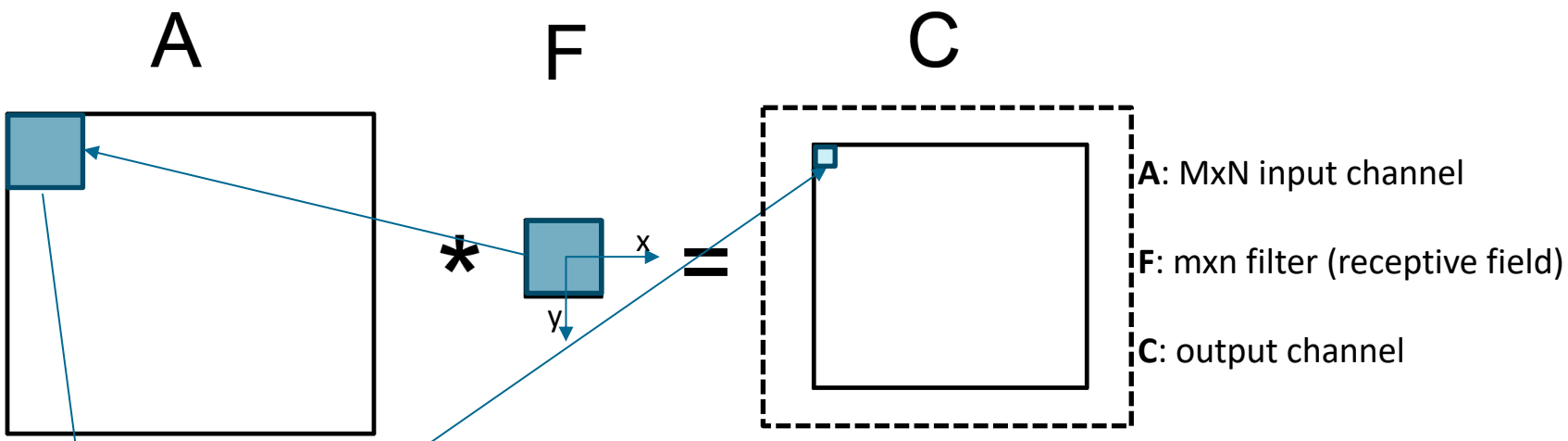
C: output channel

Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

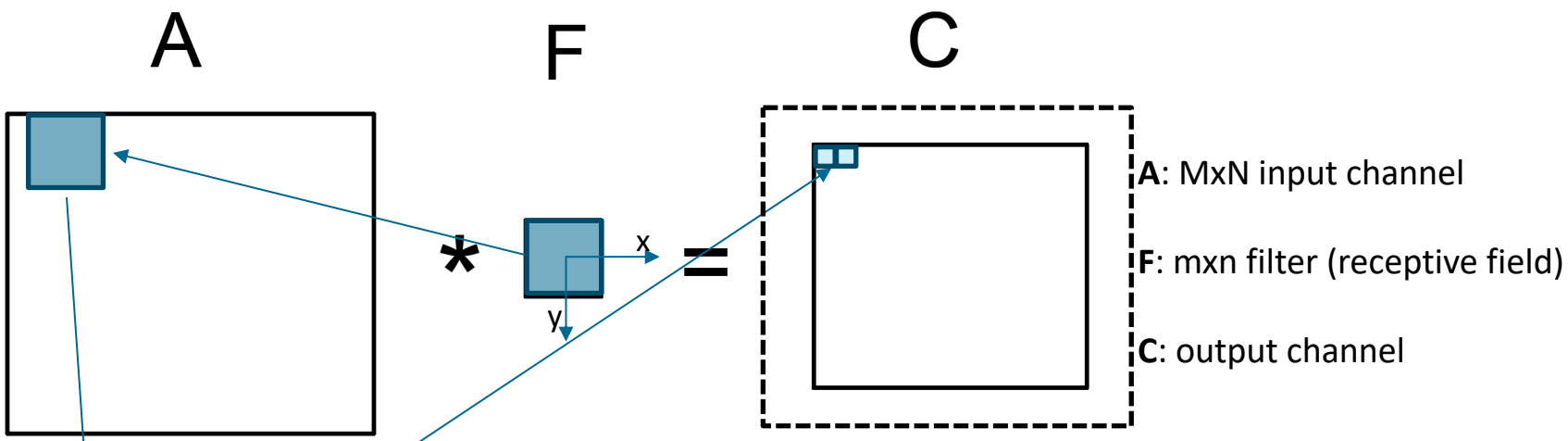


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

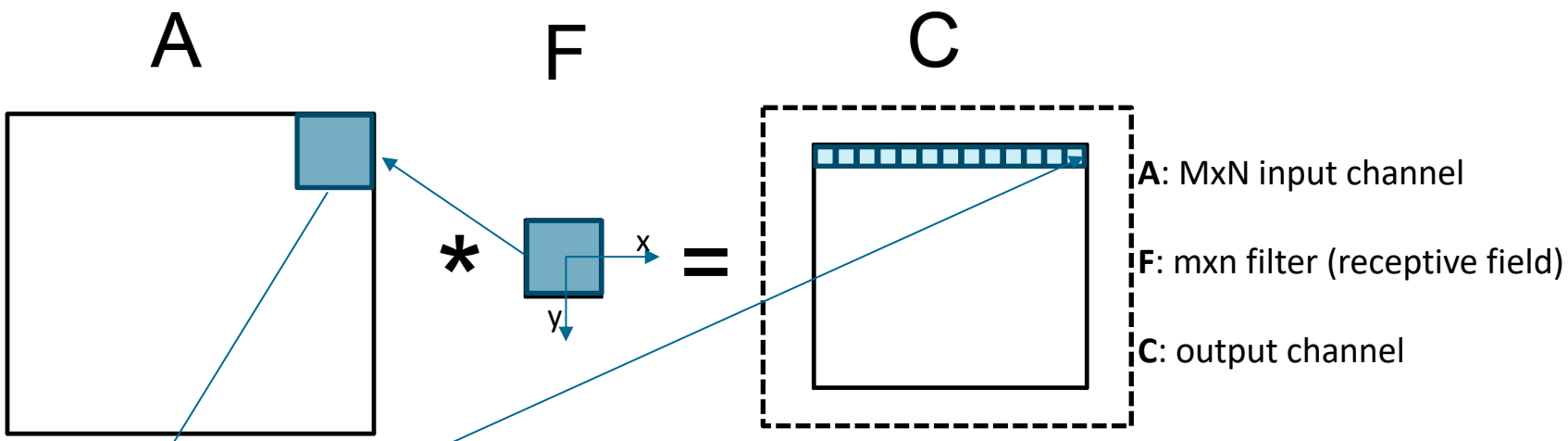


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

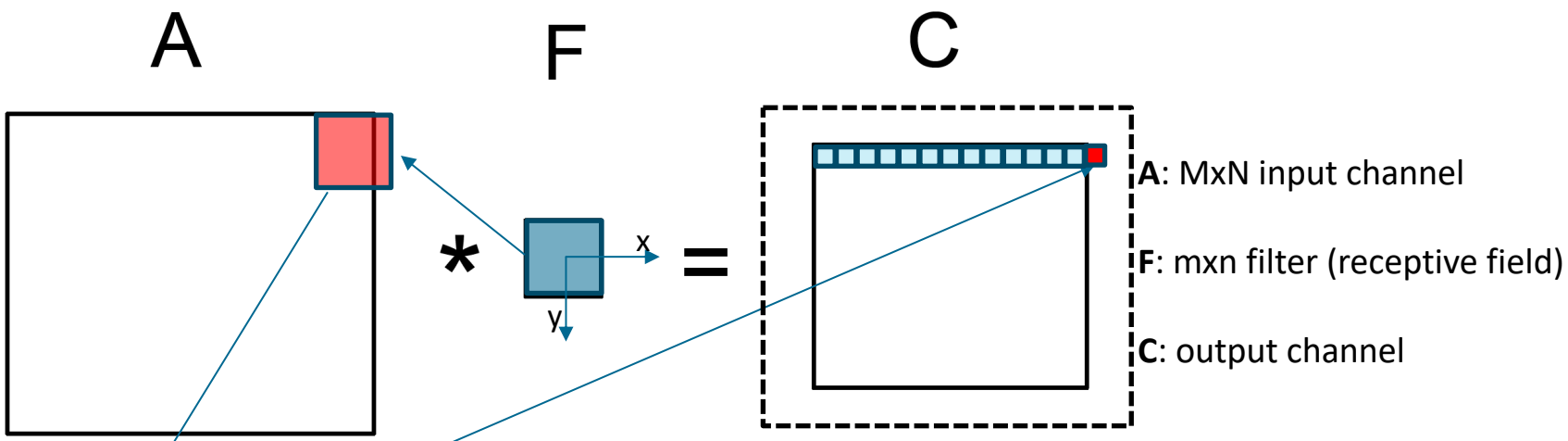


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

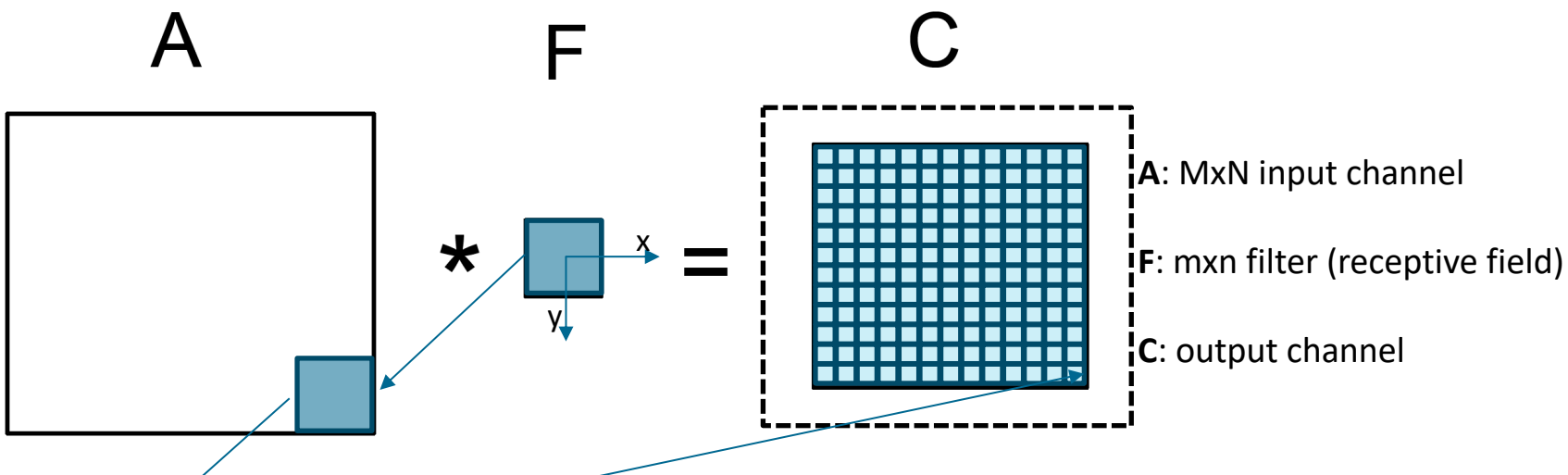


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

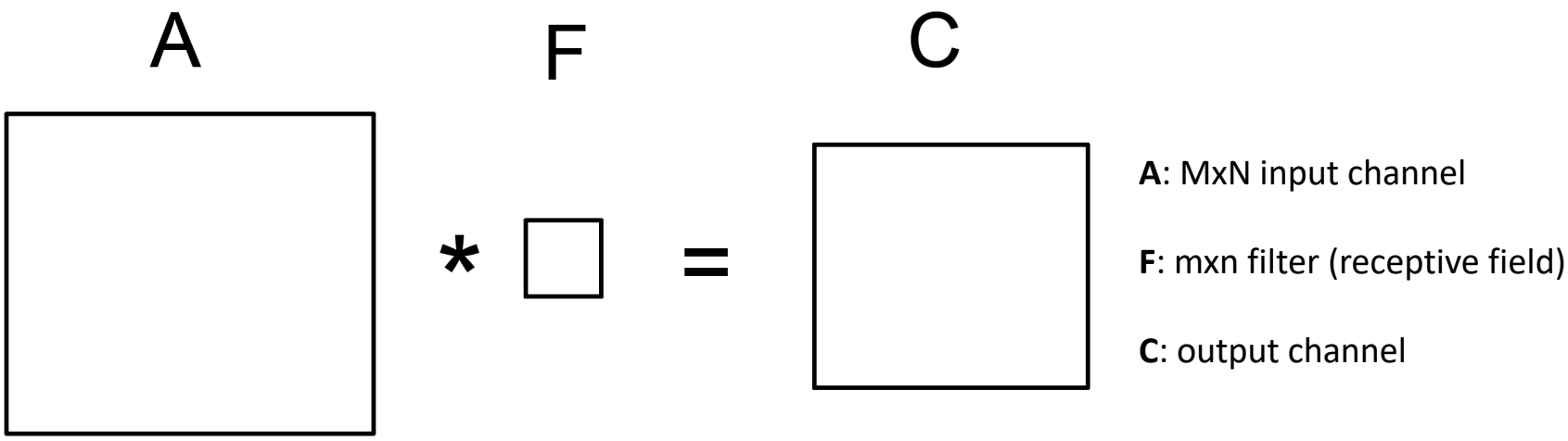


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

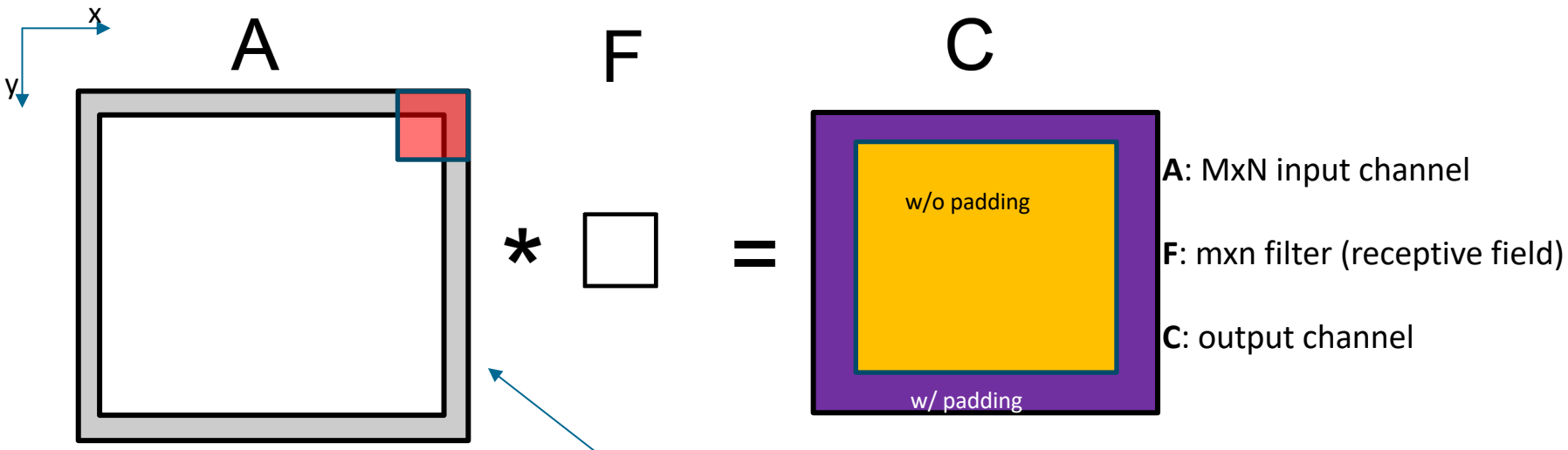


Convolution

$$C(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y)F(x,y)$$

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network



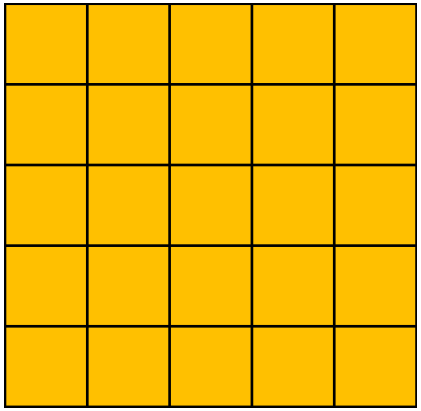
- The size of the output channel of a convolution might be **smaller** than the size of the input channel.
- In order to avoid that, we can apply **padding** to the input channel

[slide adapted from Arnaud Bergeron]

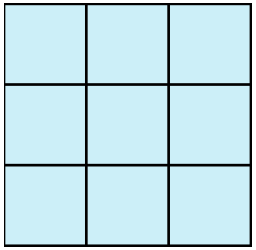
Padding

- How much do we need to pad?

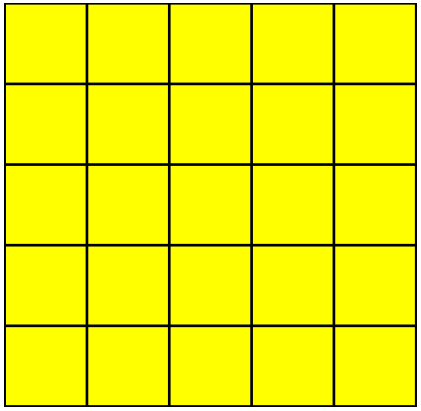
5x5 input image



3x3 filter



5x5 output

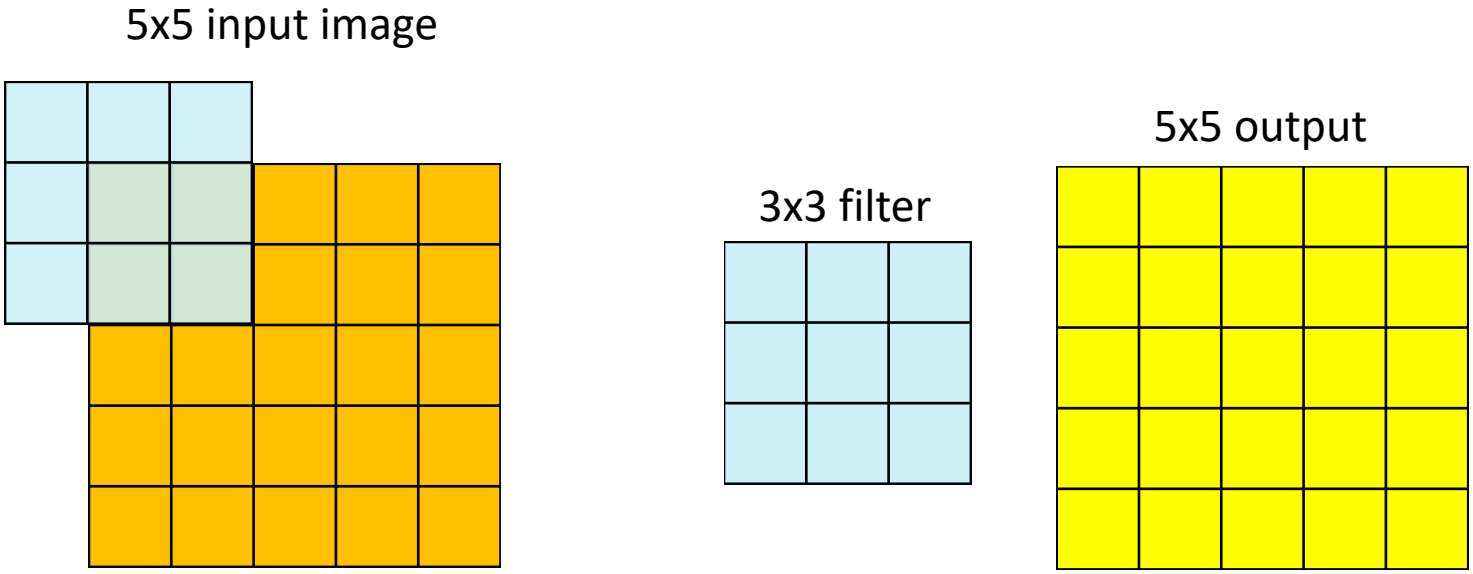


“same” convolution

Padding = (3-1)/2 = 1 px

Padding

- How much do we need to pad?



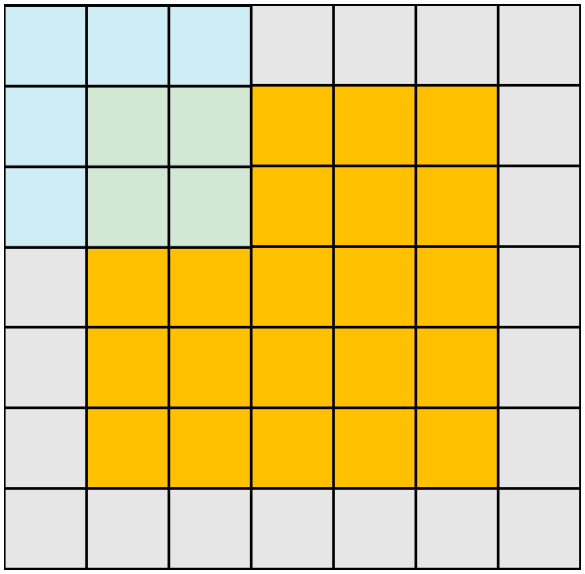
Padding = (3-1)/2 = 1 px (per side)

“same” convolution

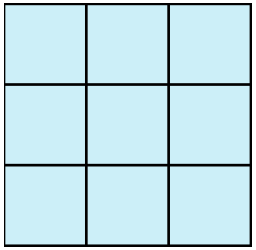
Padding

- How much do we need to pad?

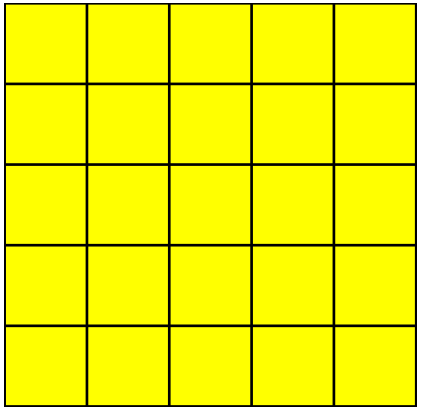
5x5 input image



3x3 filter



5x5 output



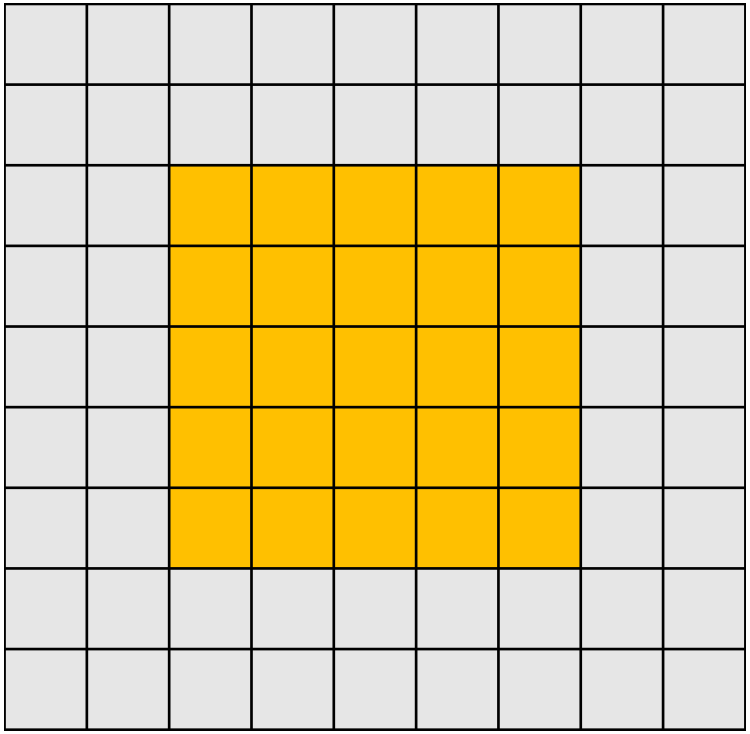
Padding = (3-1)/2 = 1 px (per side)

“same” convolution

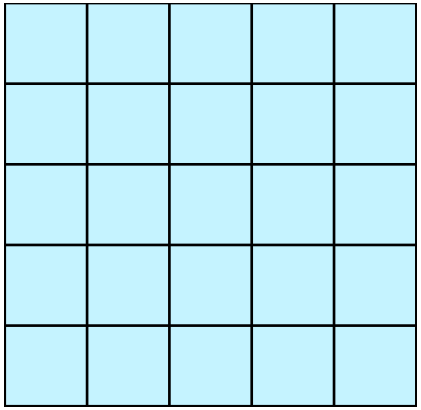
Padding

- How much do we need to pad?

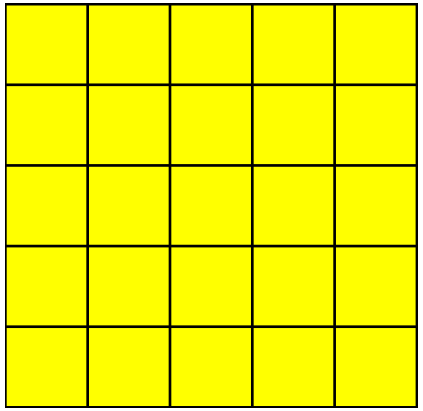
5x5 input image



5x5 filter



5x5 output



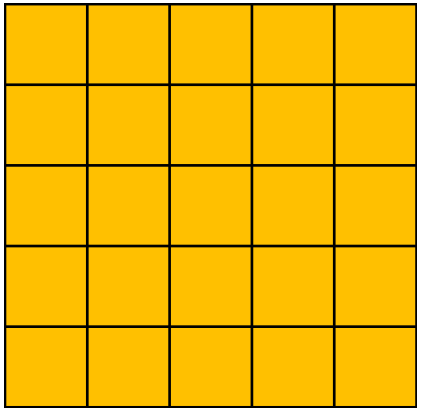
“same” convolution

Padding = (5-1)/2 = 2 px (per side)

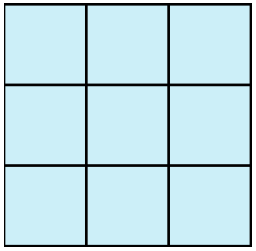
“Valid” convolution

- How much do we need to pad?

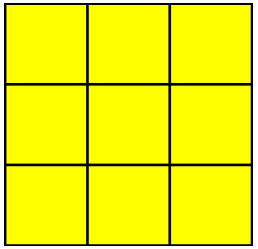
5x5 input image



3x3 filter



3x3 output



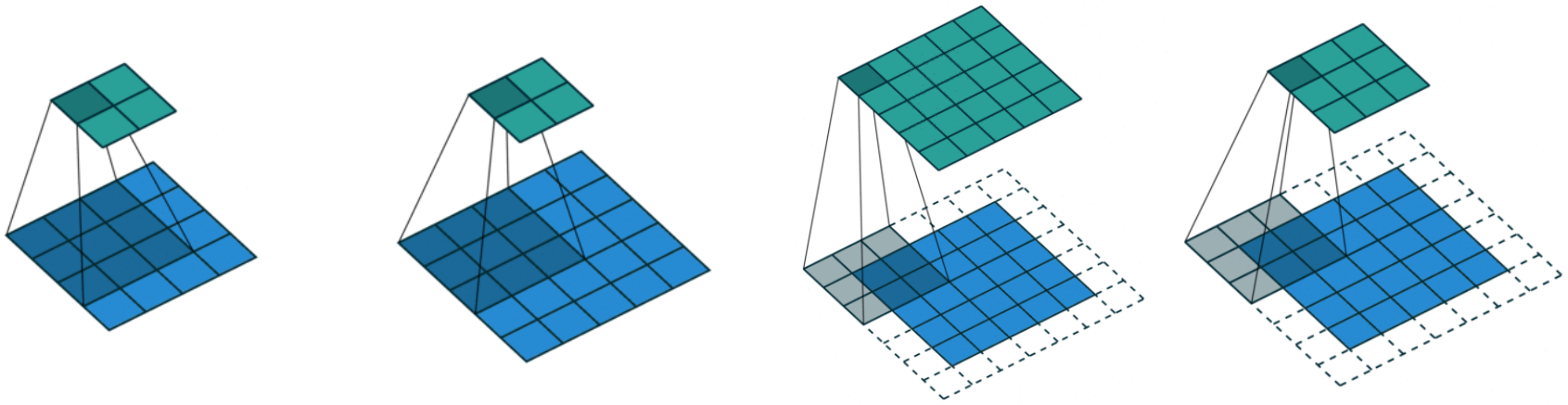
Padding = (3-1)/2 = 1 px

- **padding**: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as the original input. `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.

Stride

- Stride is the **step** that you make when **applying convolutions**

- By changing the stride you can:
  - change the size of the output by keeping the same filter size
  - cover a larger input area with the same convolutions

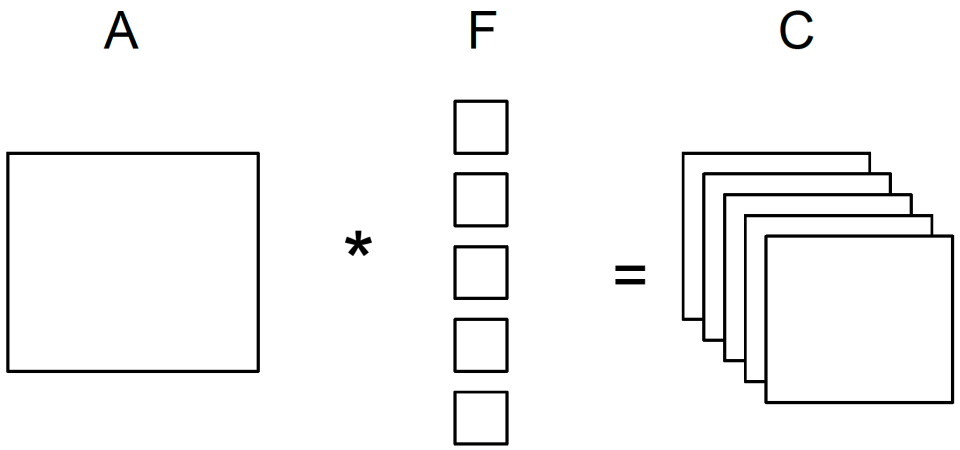


no padding, no stride | no padding, stride | padding, no stride | padding, stride

[animations from https://github.com/vdumoulin/conv_arithmetic]

Convolutional Neural Network

- Applying **multiple filters** we obtain **multiple output** channels



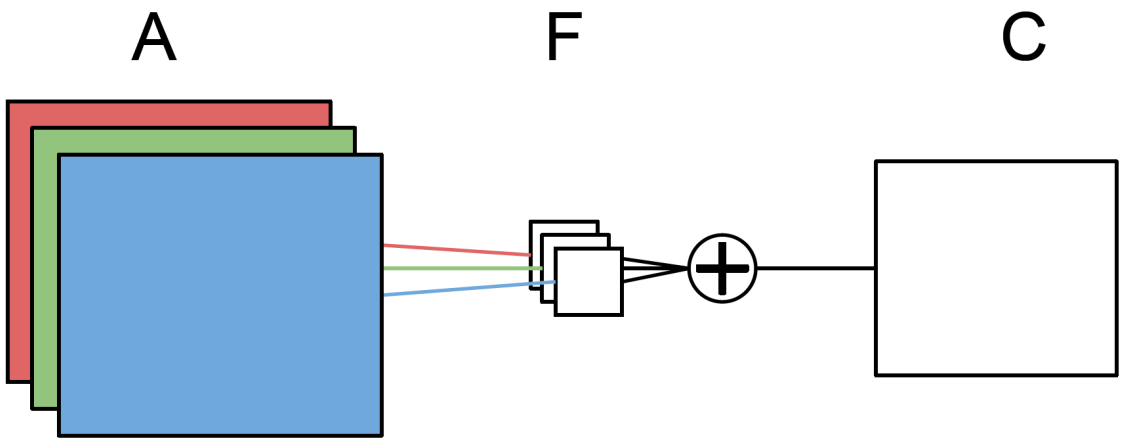
$$C_o(i,j) = \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A(i-x, j-y) F_o(x,y)$$

o is the index of the output channel

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

- Convolution with **multiple input channels** (example: RGB images)



One example of **multiple channels** is also the **output** of the previous slide

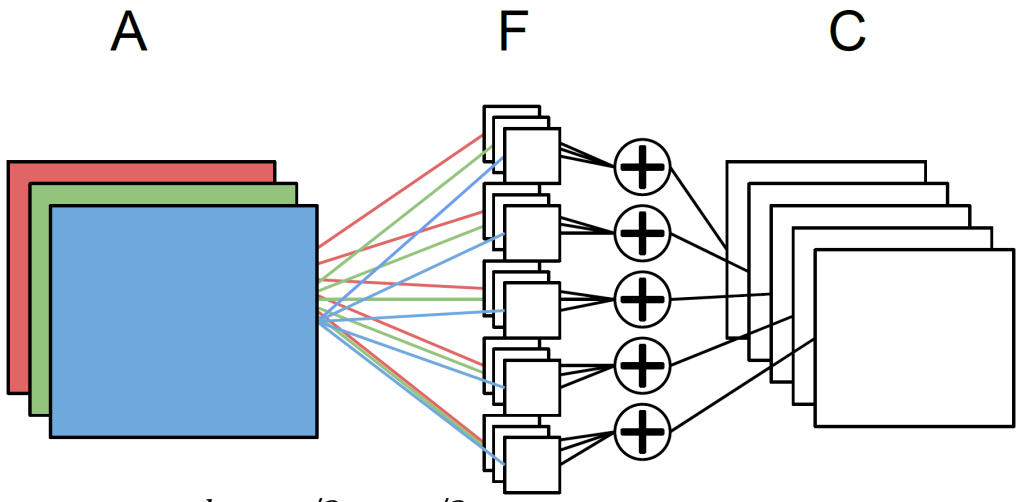
$$C(i,j) = \sum_{k=0}^{l} \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A_k(i-x, j-y) F_k(x,y)$$

F is now a set of filters
k is the index of the input channel

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

- Convolution with **multiple inputs** and **outputs**



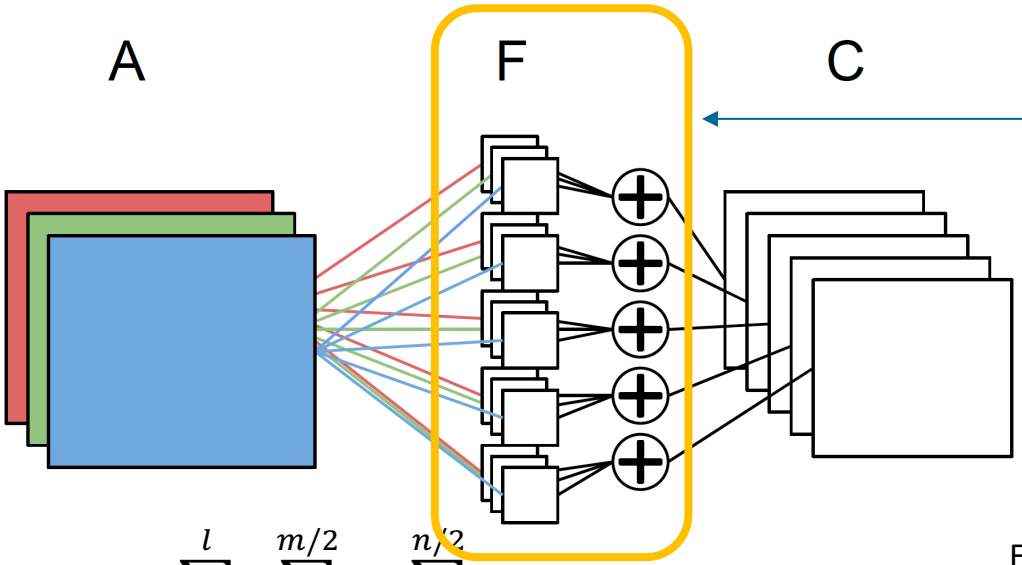
$$C_o(i,j) = \sum_{k=0}^{l} \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A_k(i-x, j-y) F_{ko}(x,y)$$

F is now a set of filters
k is the index of the input channel
o is the index of the output channel

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network

- Convolution with **multiple inputs** and **outputs**



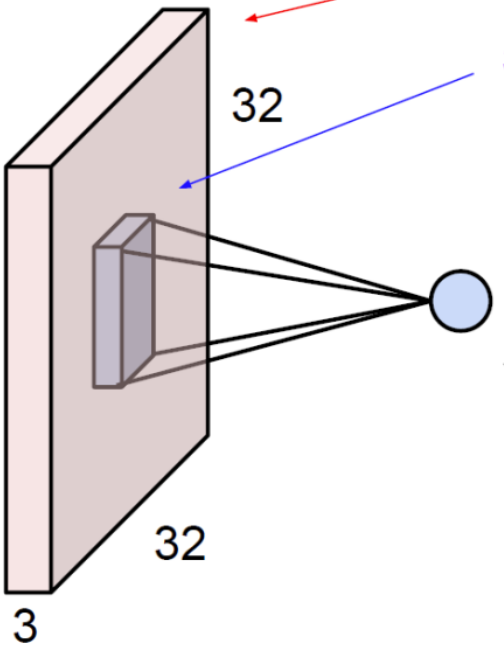
We can call this a
Convolutional layer

$$C_o(i,j) = \sum_{k=0}^{l} \sum_{x=-m/2}^{m/2} \sum_{y=-n/2}^{n/2} A_k(i-x, j-y) F_{ko}(x,y)$$

F is now a set of filters
k is the index of the input channel
o is the index of the output channel

[slide adapted from Arnaud Bergeron]

Convolutional Neural Network



32x32x3 image

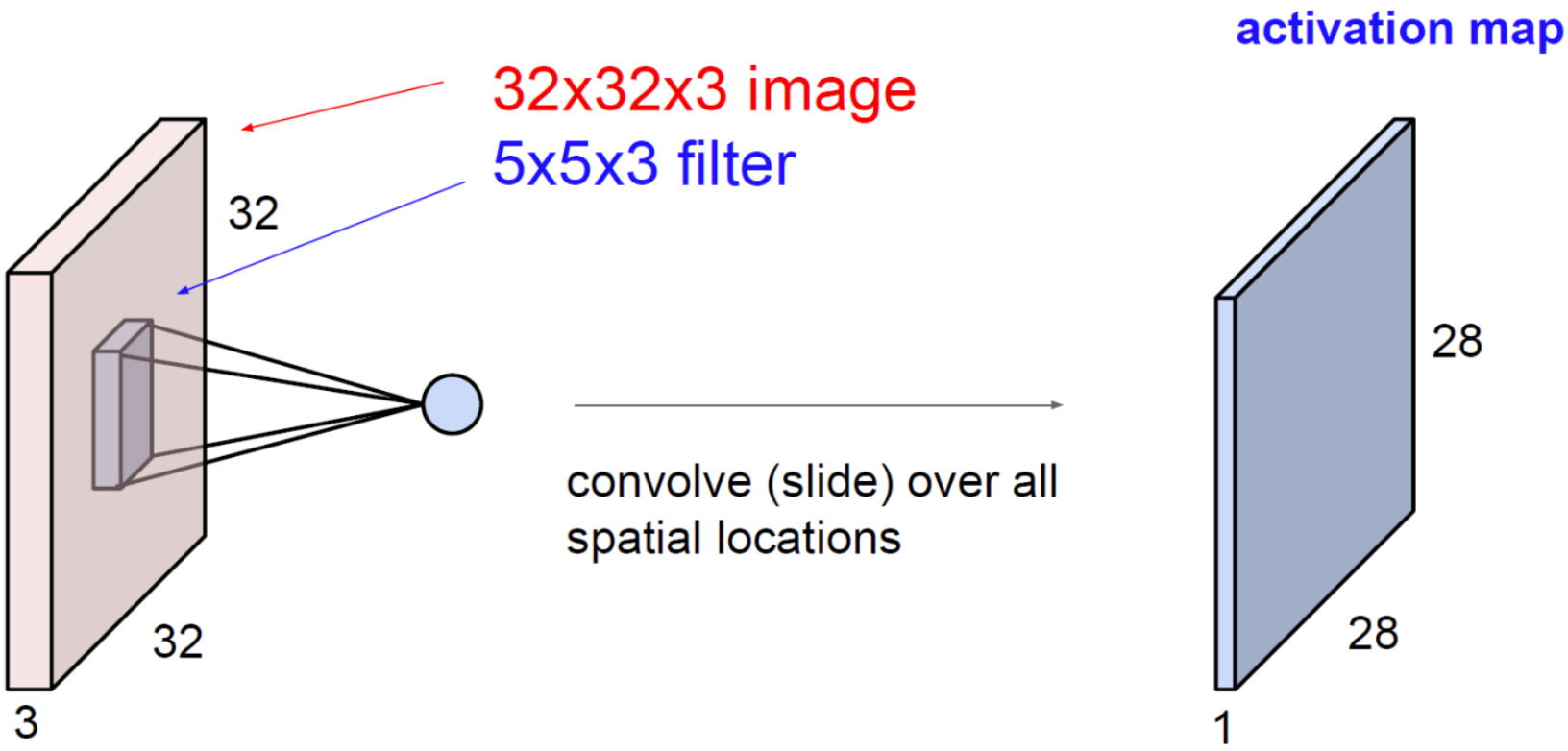
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

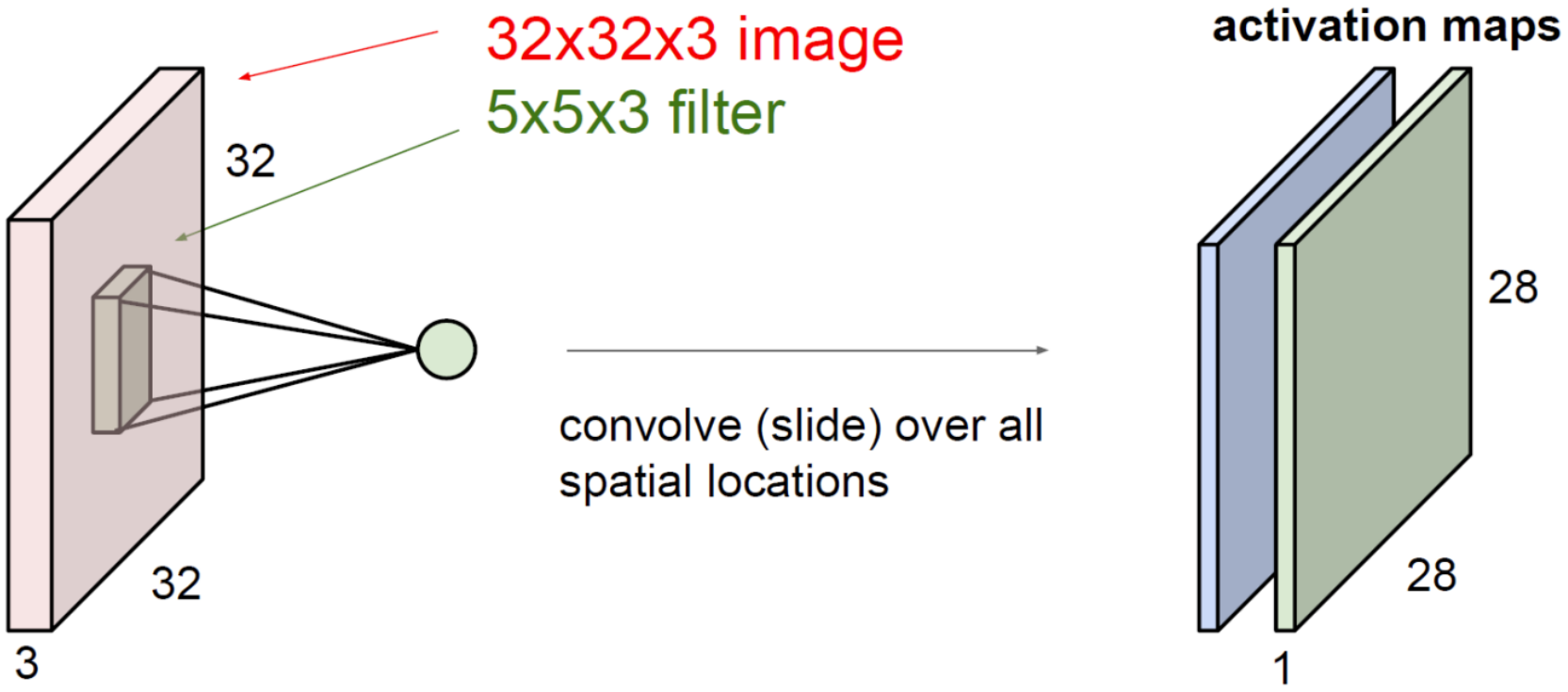
[slide adapted from Andrej Karpathy]

Convolutional Neural Network



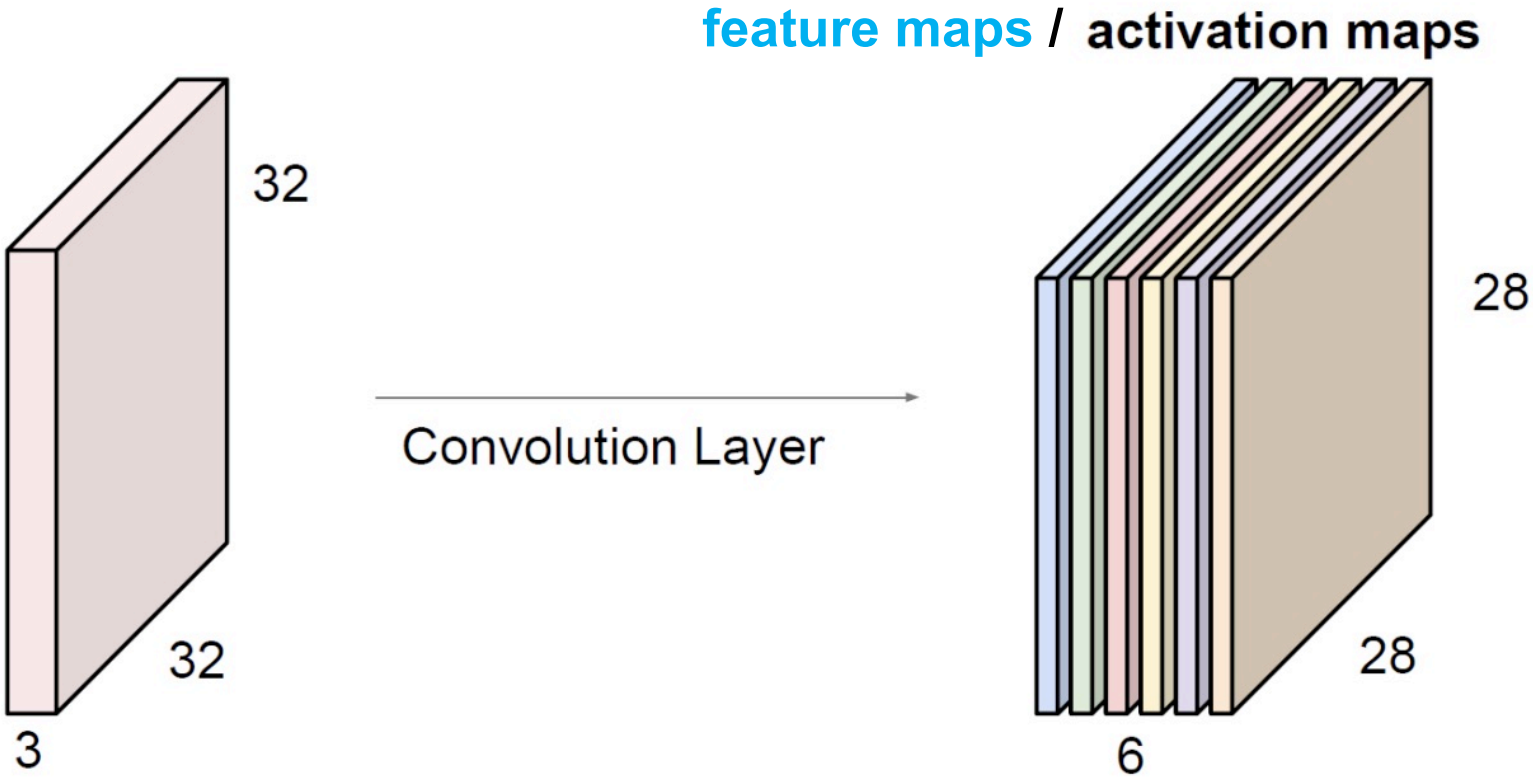
[slide adapted from Andrej Karpathy]

Convolutional Neural Network



[slide adapted from Andrej Karpathy]

Convolutional Neural Network

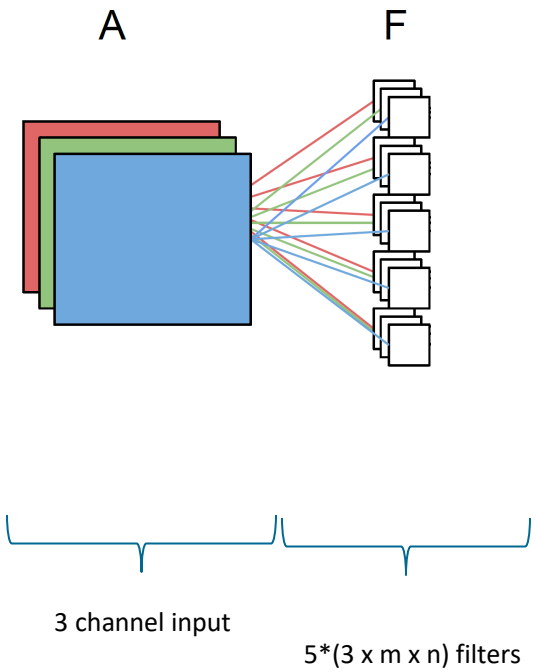


We stack these up to get a “new image” of size 28x28x6!

[slide adapted from Andrej Karpathy]

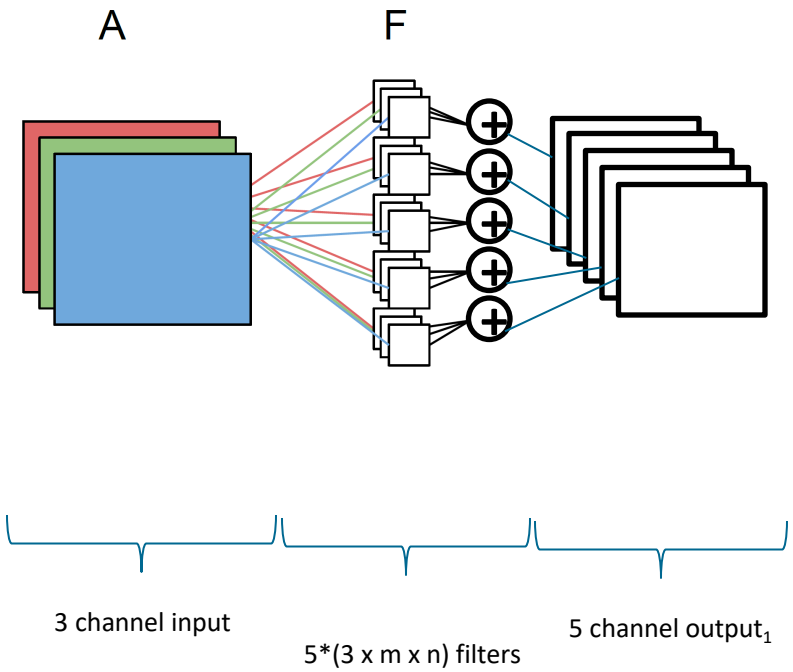
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



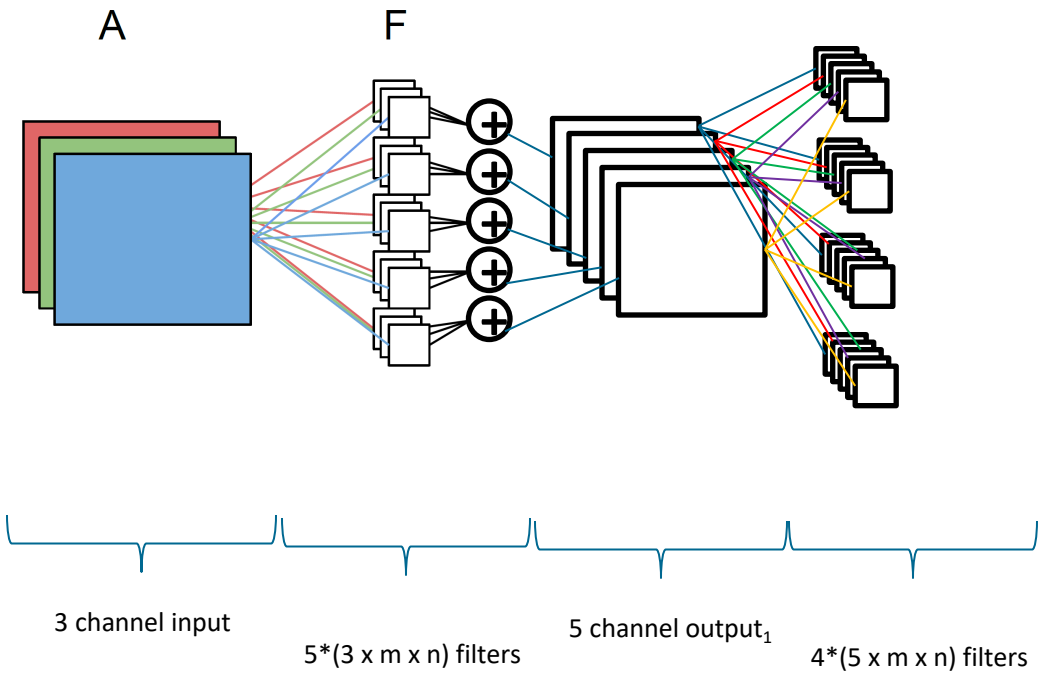
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



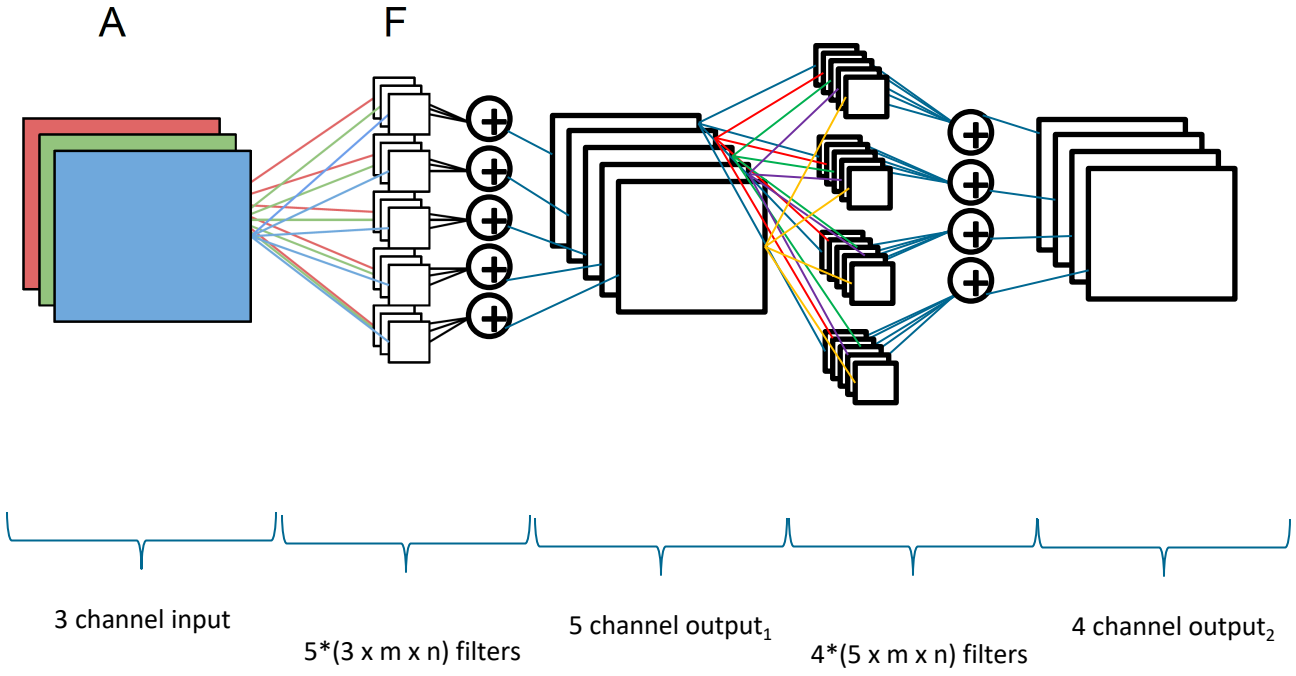
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



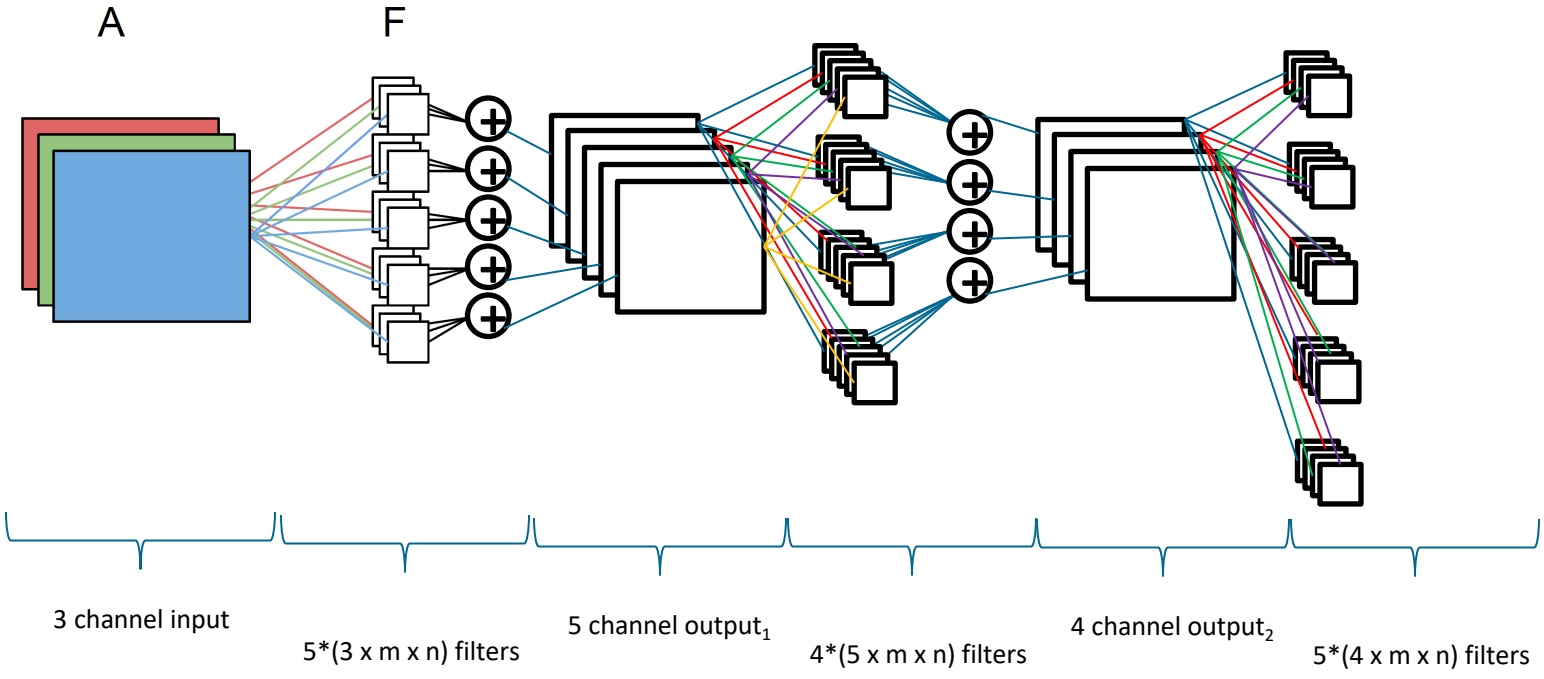
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



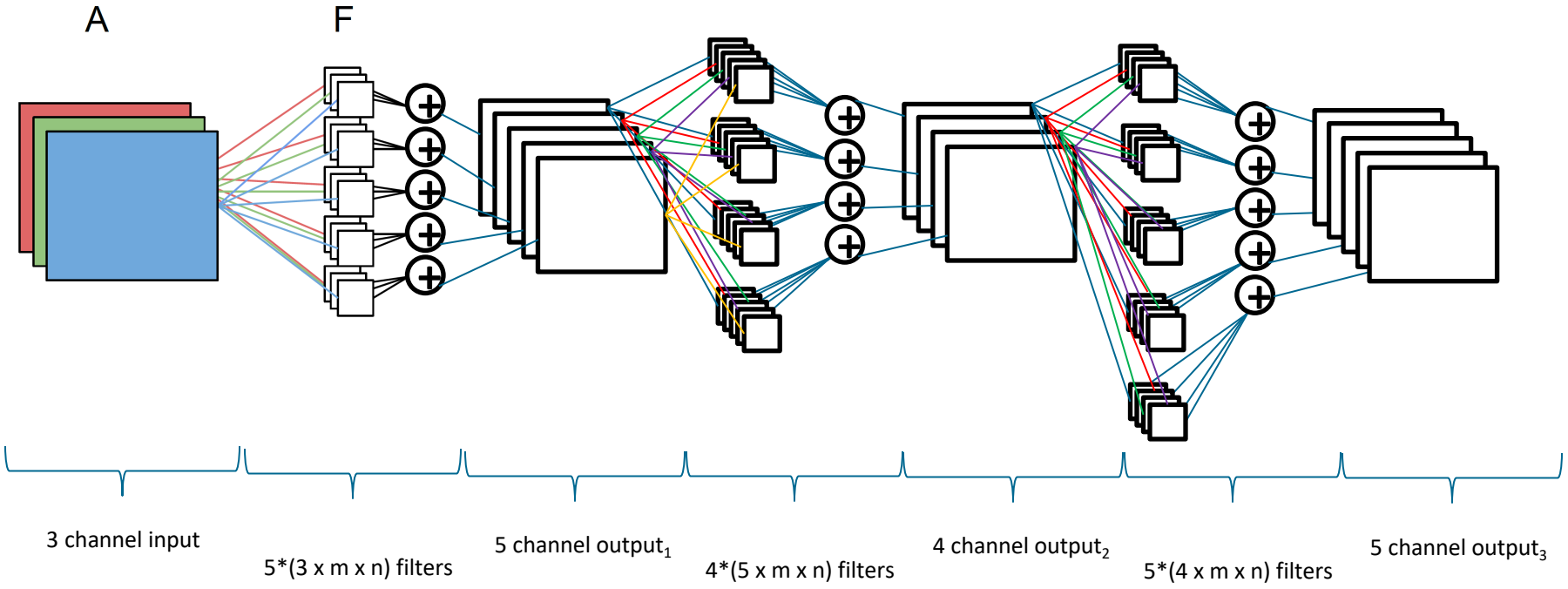
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



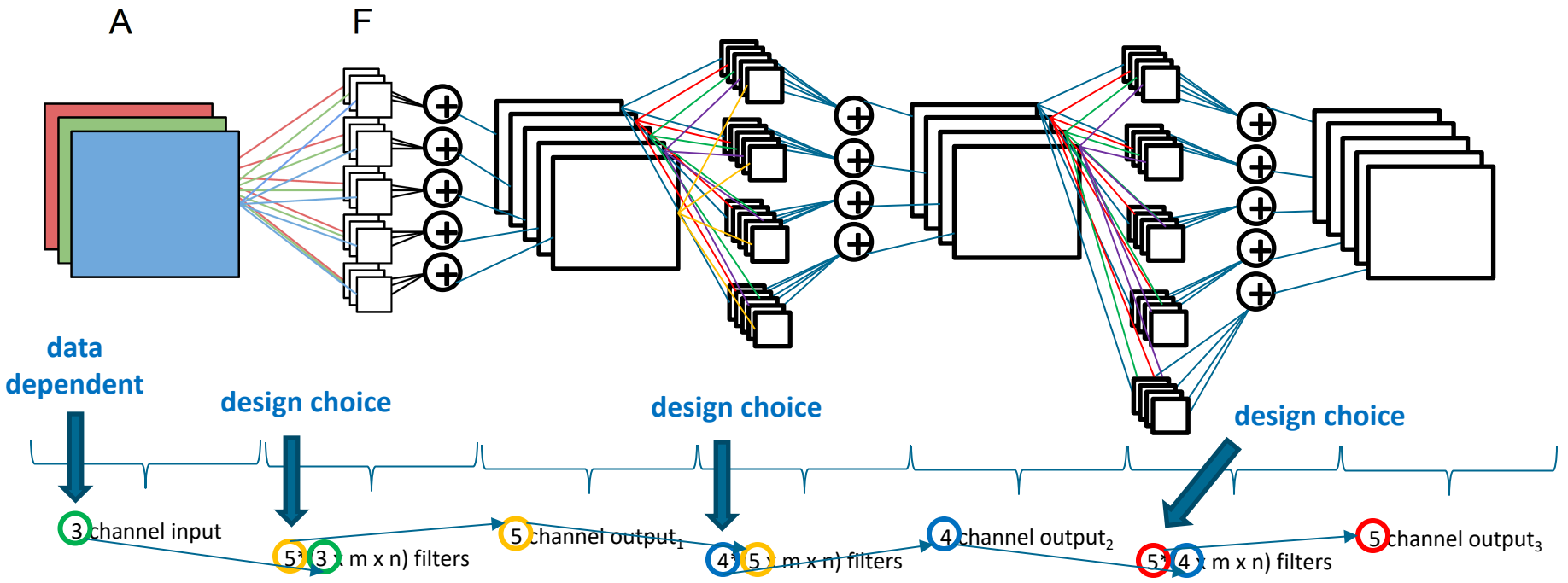
Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



Convolutional Neural Network

- A convolutional network is (basically) a stack of convolutional layers



VIBOT 2018

Machine learning with Neural Networks

Francesco Ciompi

francesco.ciompi@radboudumc.nl

Radboudumc