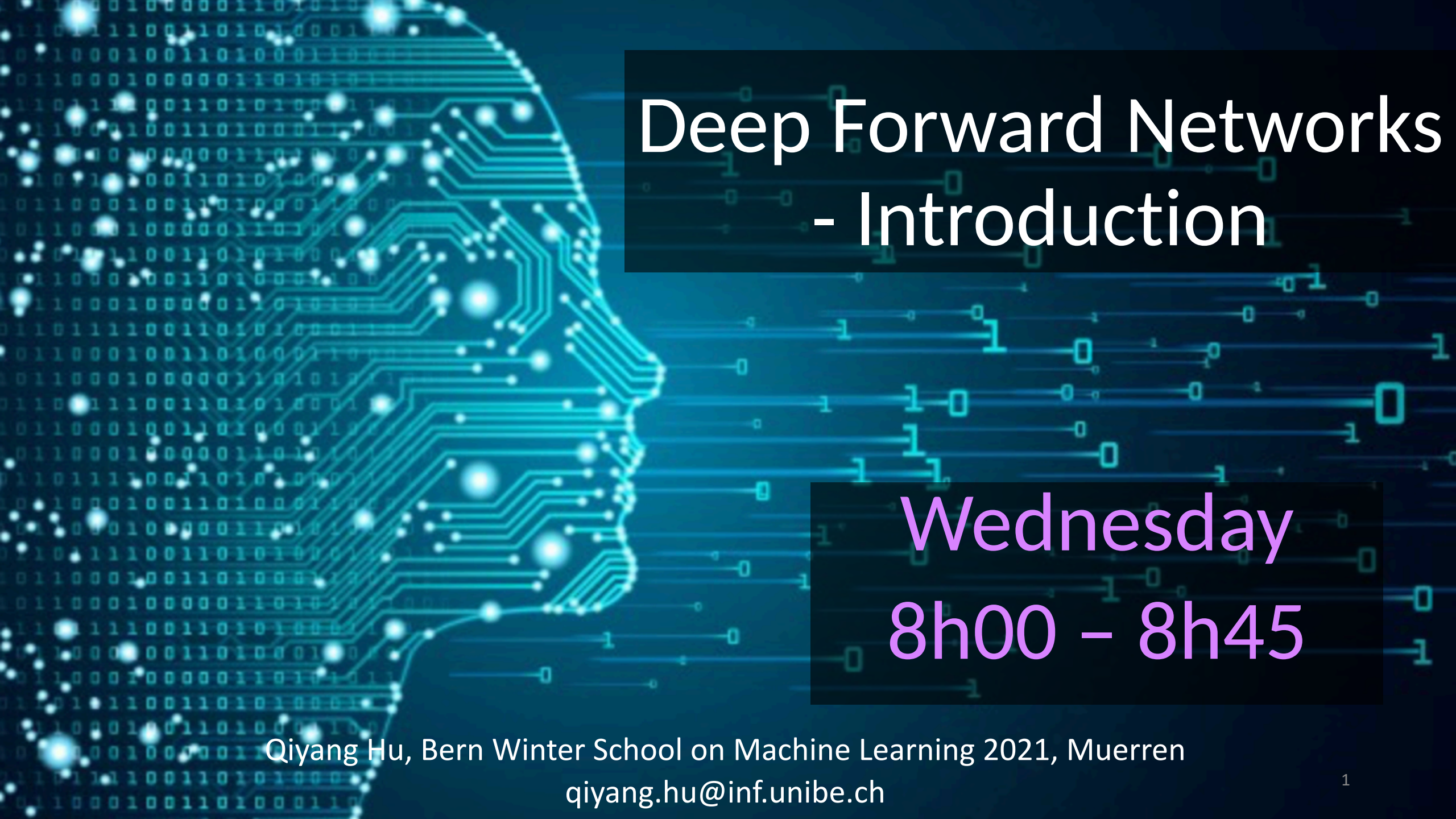


Deep Forward Networks - Introduction

Wednesday
8h00 – 8h45

Qiyang Hu, Bern Winter School on Machine Learning 2021, Muerren
qiyang.hu@inf.unibe.ch

- Continue from the neural network optimisation
  - backpropagation
  - minimize loss function

- Introduction of different layer type
  - conv layer
  - pooling
  - padding

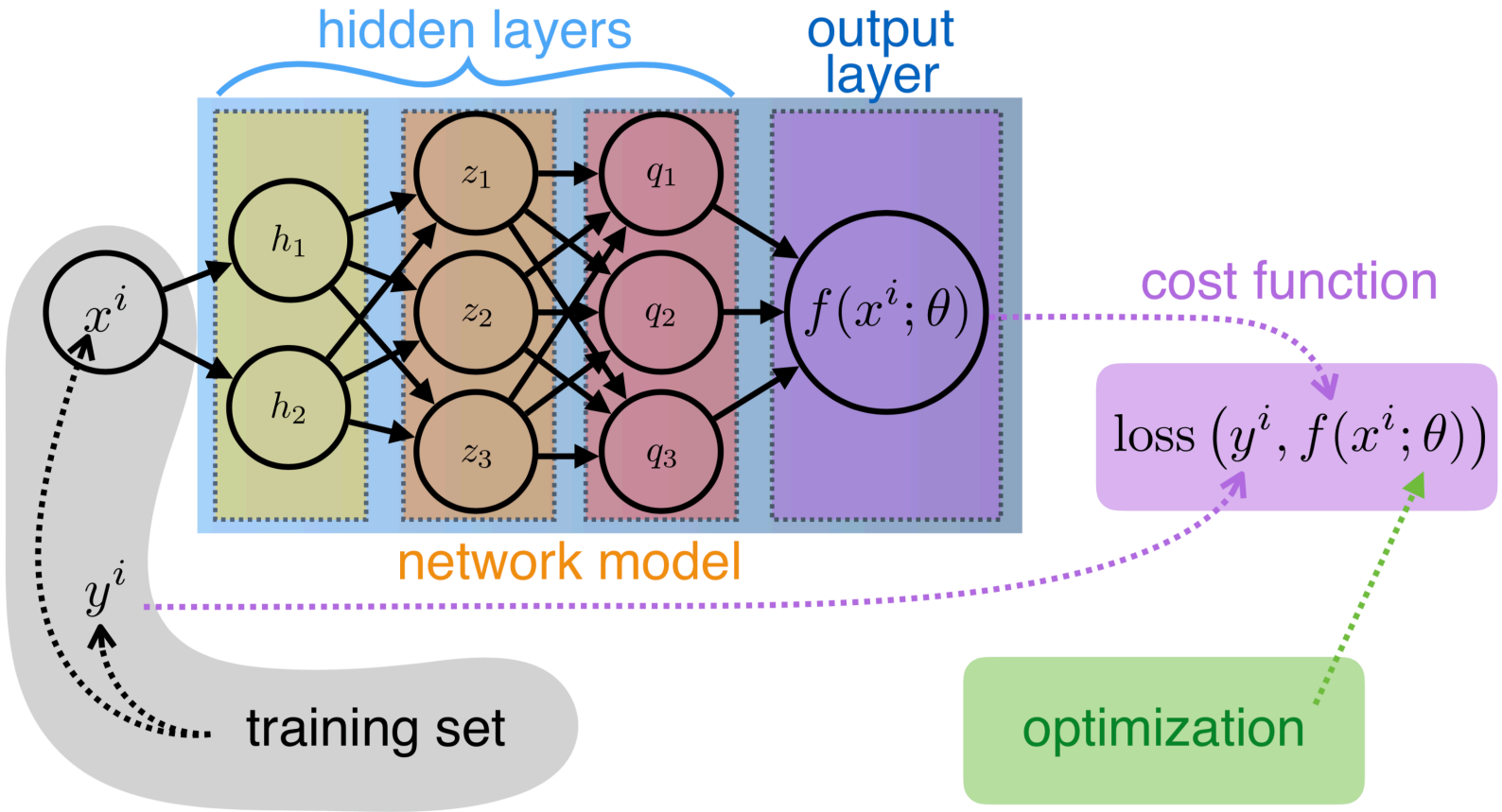
Deploying a Neural Network

Given a task (in terms of **I/O mappings**), we need :

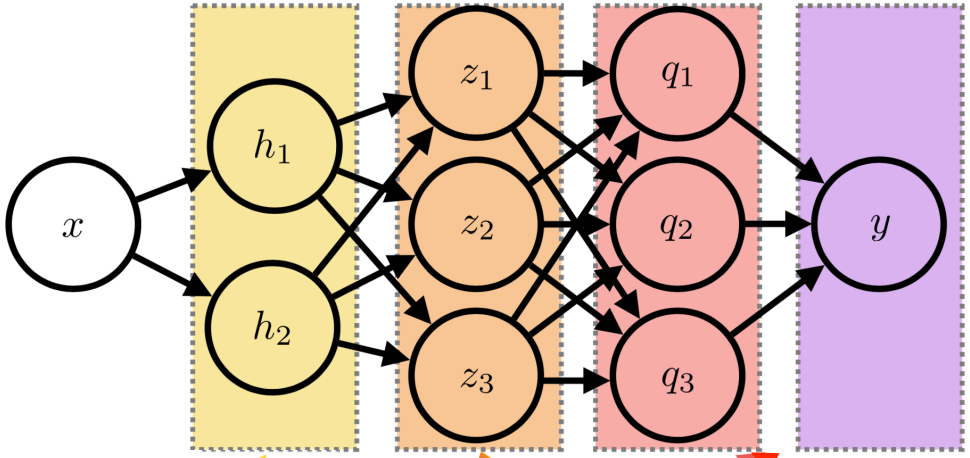
1) **Network model**

2) **Cost function**

3) **Optimization**



Activation functions



$$y = f_4(q_1, q_2, q_3)$$

$$y = f_4(f_{3,1}(f_{2,1}(f_{1,1}(x), f_{1,2}(x)), \dots), \dots)$$

Hierarchical representation

$$h_1 = f_{1,1}(x)$$
$$h_2 = f_{1,2}(x)$$

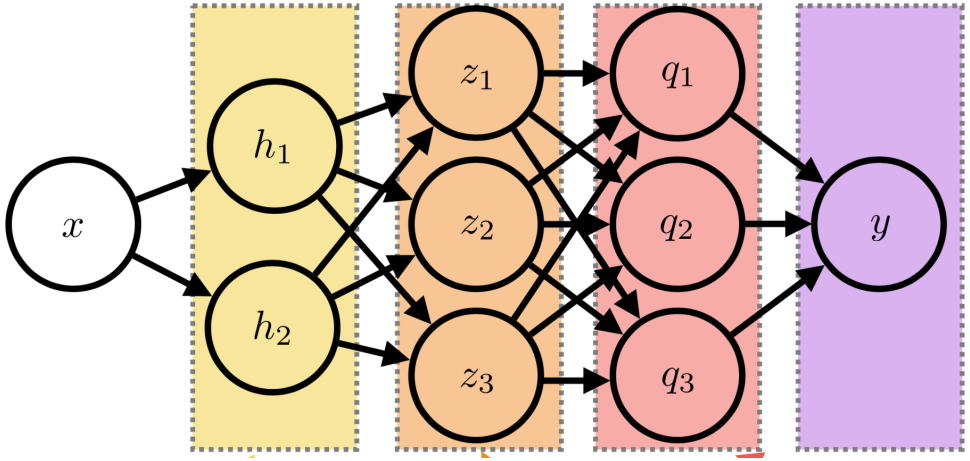
$$z_1 = f_{2,1}(h_1, h_2)$$
$$z_2 = f_{2,2}(h_1, h_2)$$
$$z_3 = f_{2,3}(h_1, h_2)$$

$$q_1 = f_{3,1}(z_1, z_2, z_3)$$
$$q_2 = f_{3,2}(z_1, z_2, z_3)$$
$$q_3 = f_{3,3}(z_1, z_2, z_3)$$

Bias (constant)

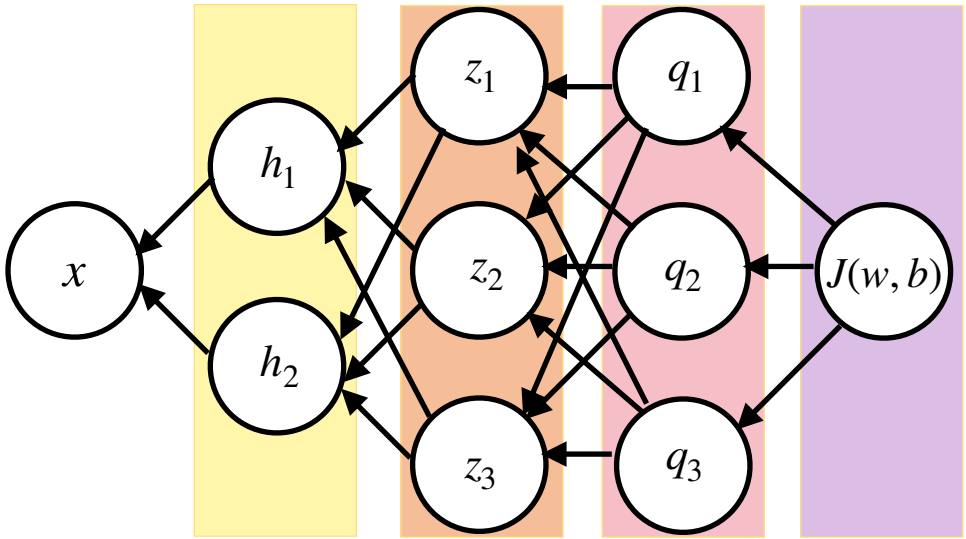
$$f_{2,2}(h_1, h_2) = w_1 h_1 + w_2 h_2 + b_{2,2}$$

3) Optimization



Forward propagation

$$\hat{y} = f_4(f_{3,1}(f_{2,1}(f_{1,1}(x), f_{1,2}(x)), \dots), \dots)$$



$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

Back propagation

Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent.

- Function : $J(\theta)$
- Objective : minimize $J(\theta)$

- Outline:

  - Start with an initial θ (mostly we initialise with $\theta = 0$)
  - Keep updating θ to reduce $J(\theta)$ until we hopefully end up at a minimum

  Criteria to update θ :

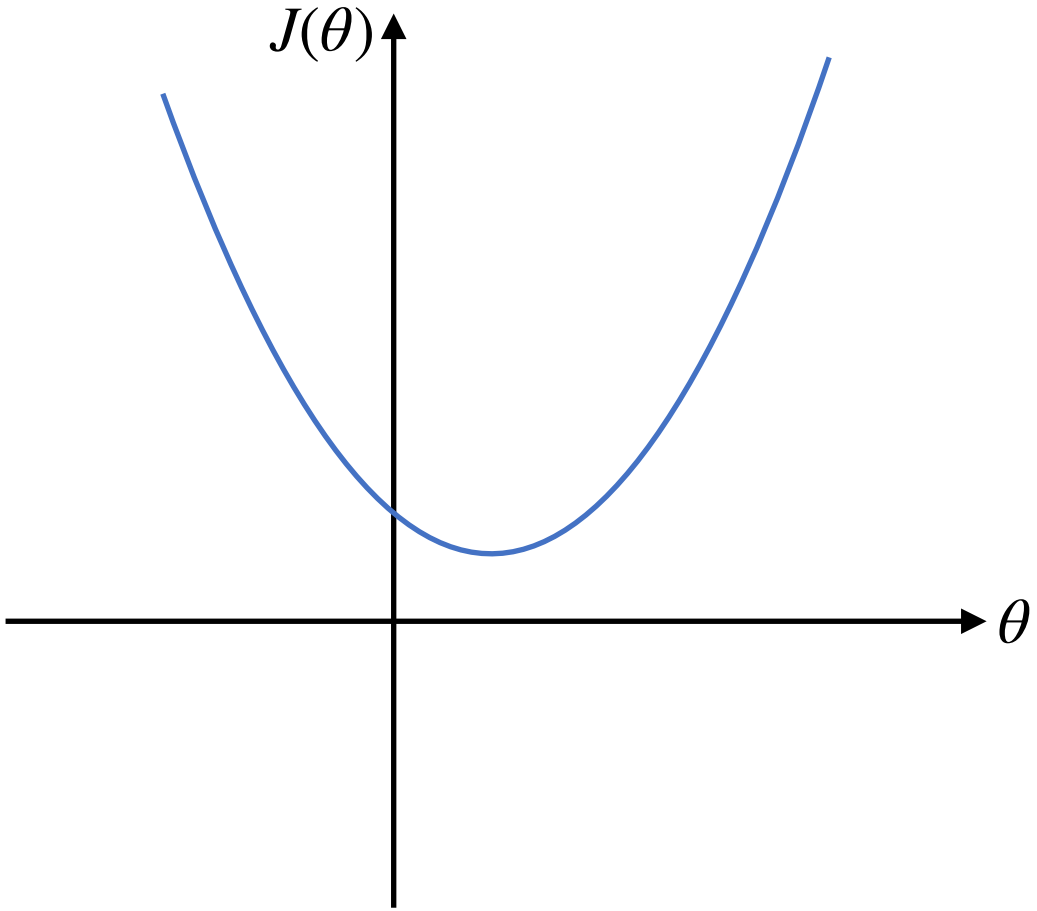
$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

Learning rate (Positive constant)

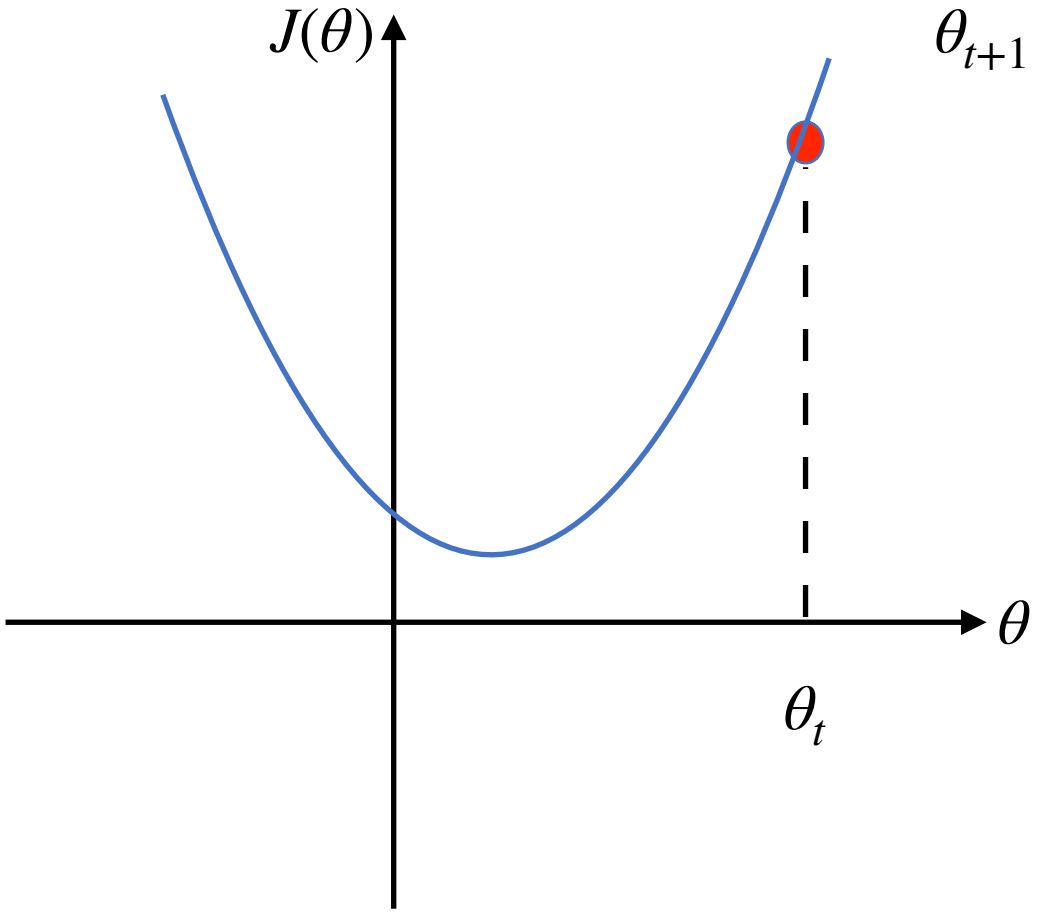
Gradient

The gradient vector can be interpreted as the "direction and rate of fastest increase".

Gradient descent

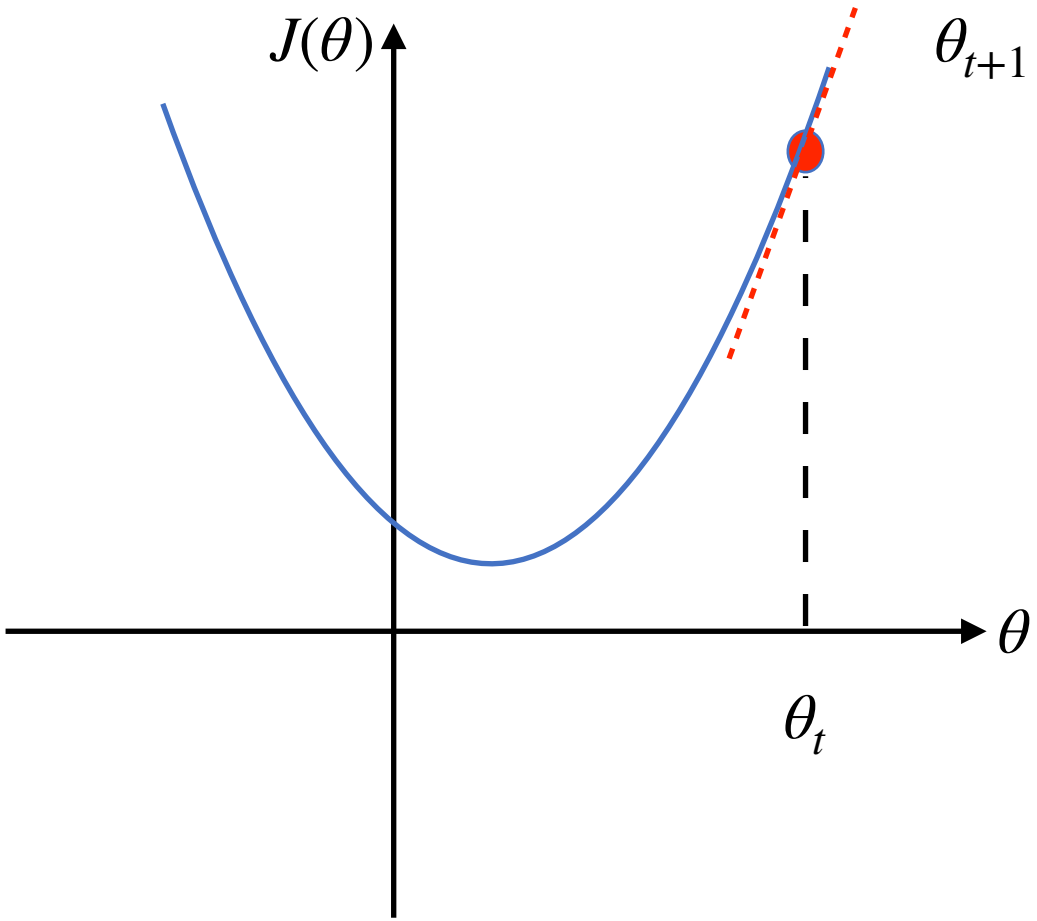


Gradient descent



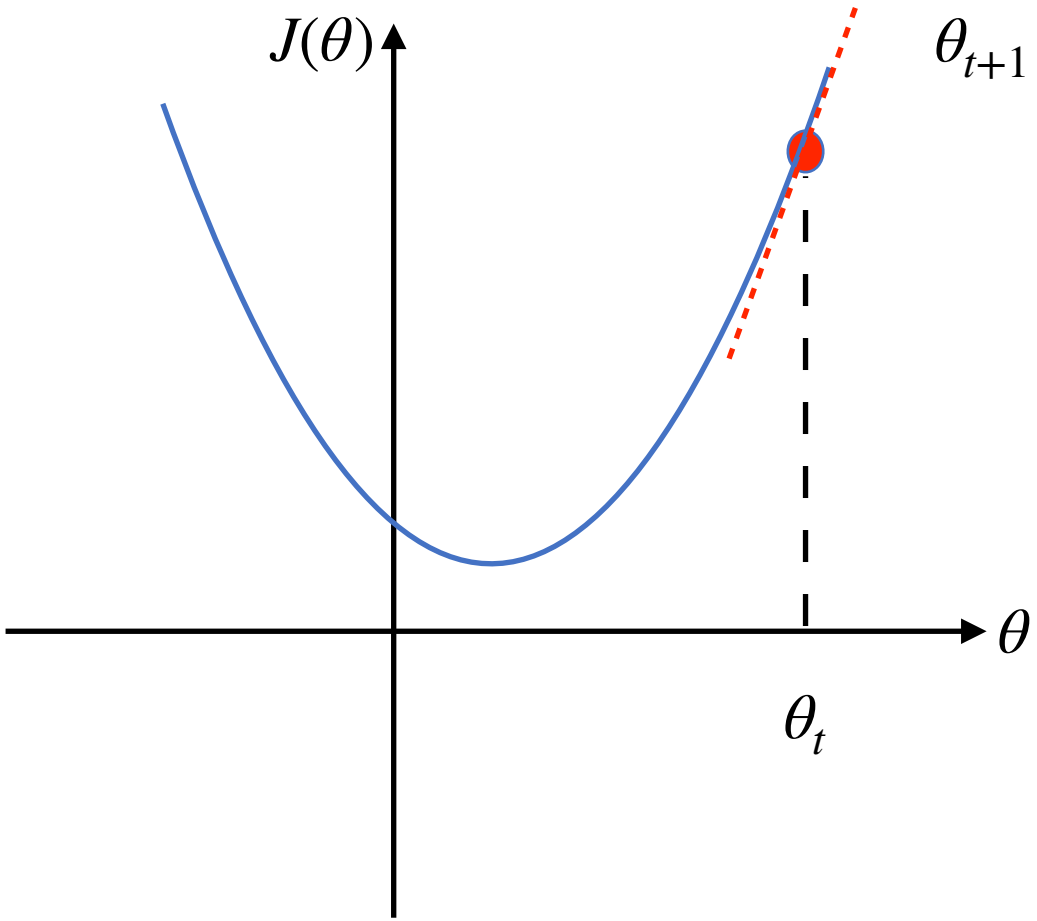
$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

Gradient descent



$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

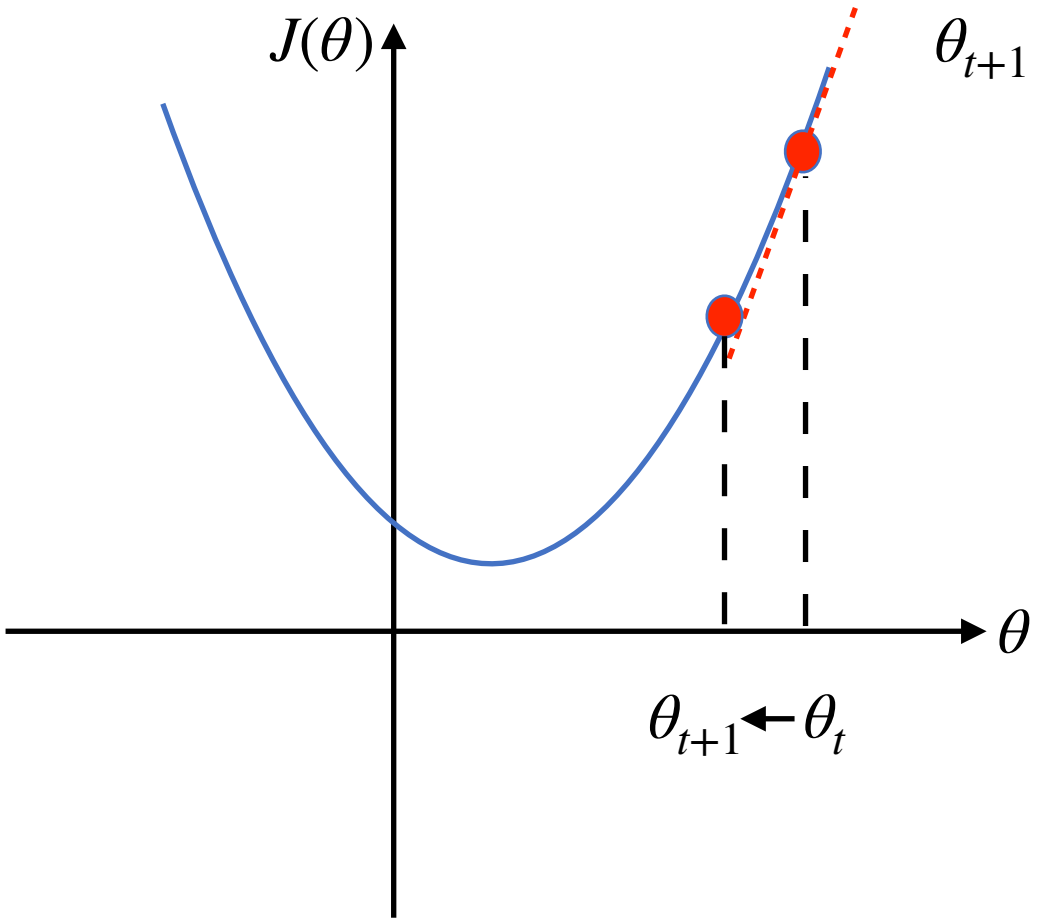
Gradient descent



$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

Positive

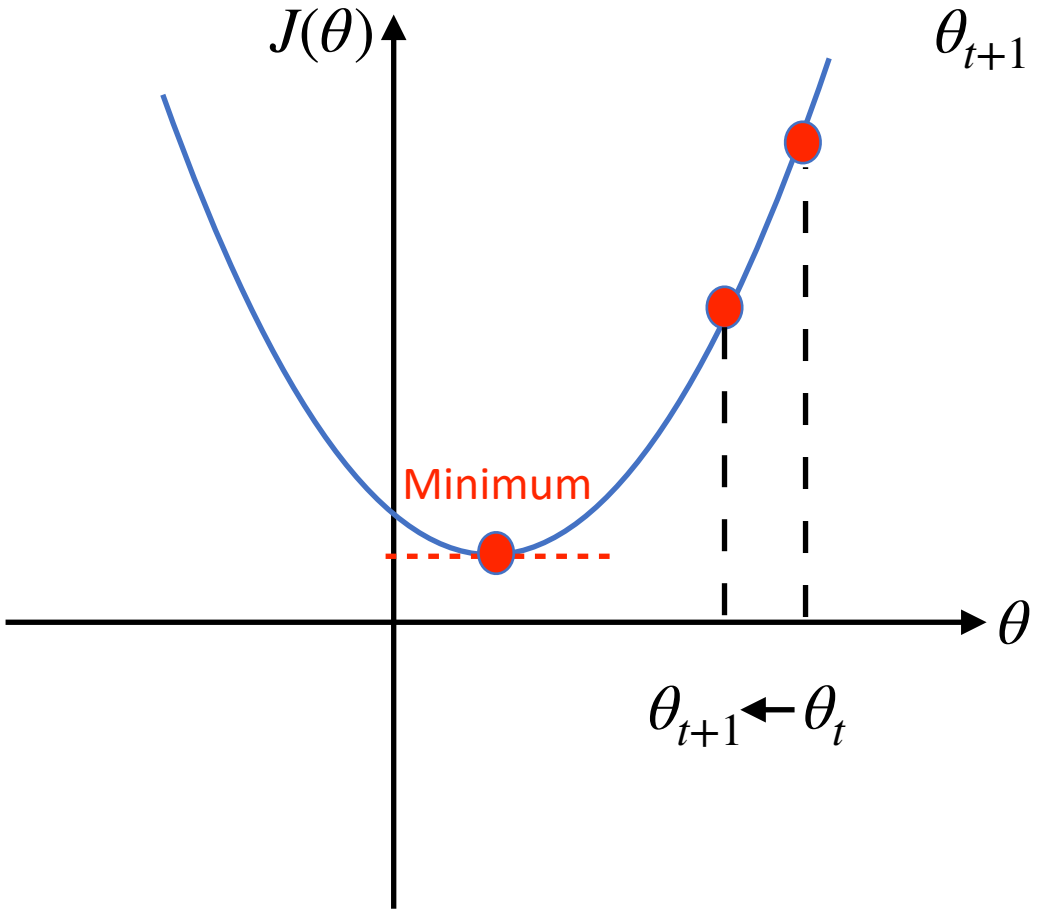
Gradient descent



$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

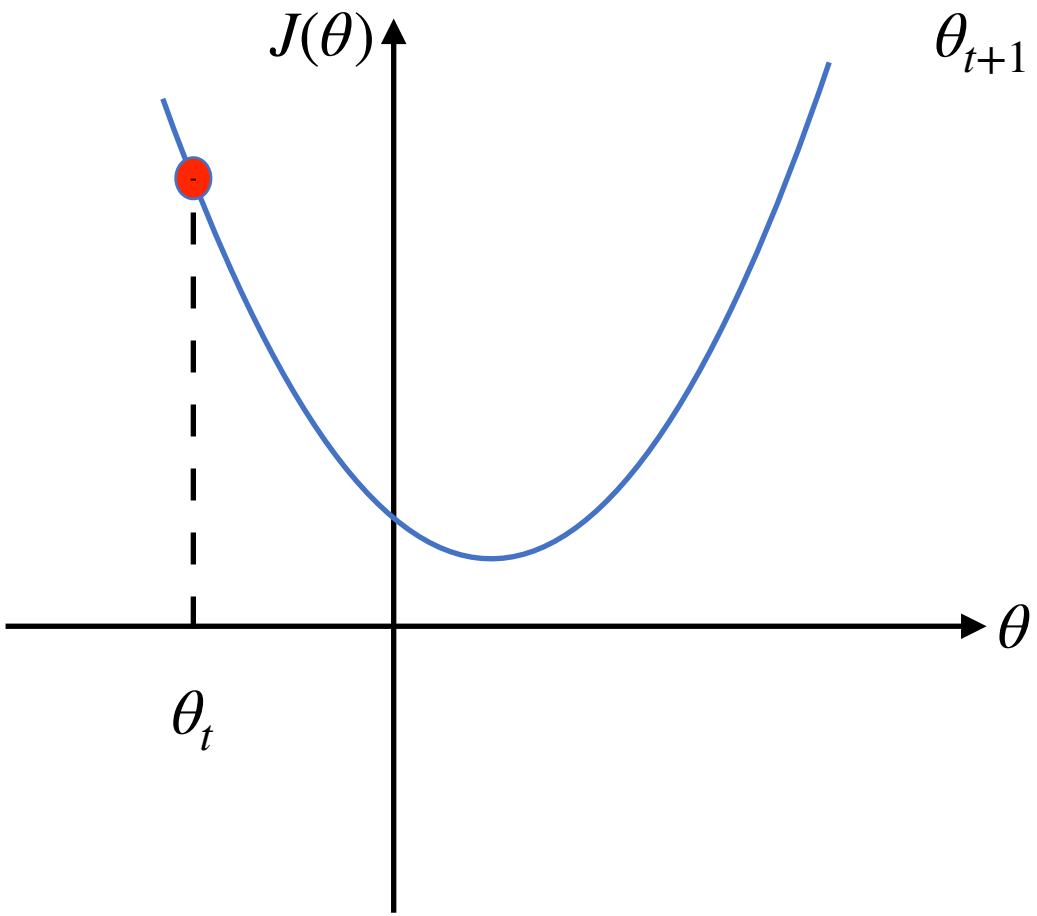
Positive

Gradient descent



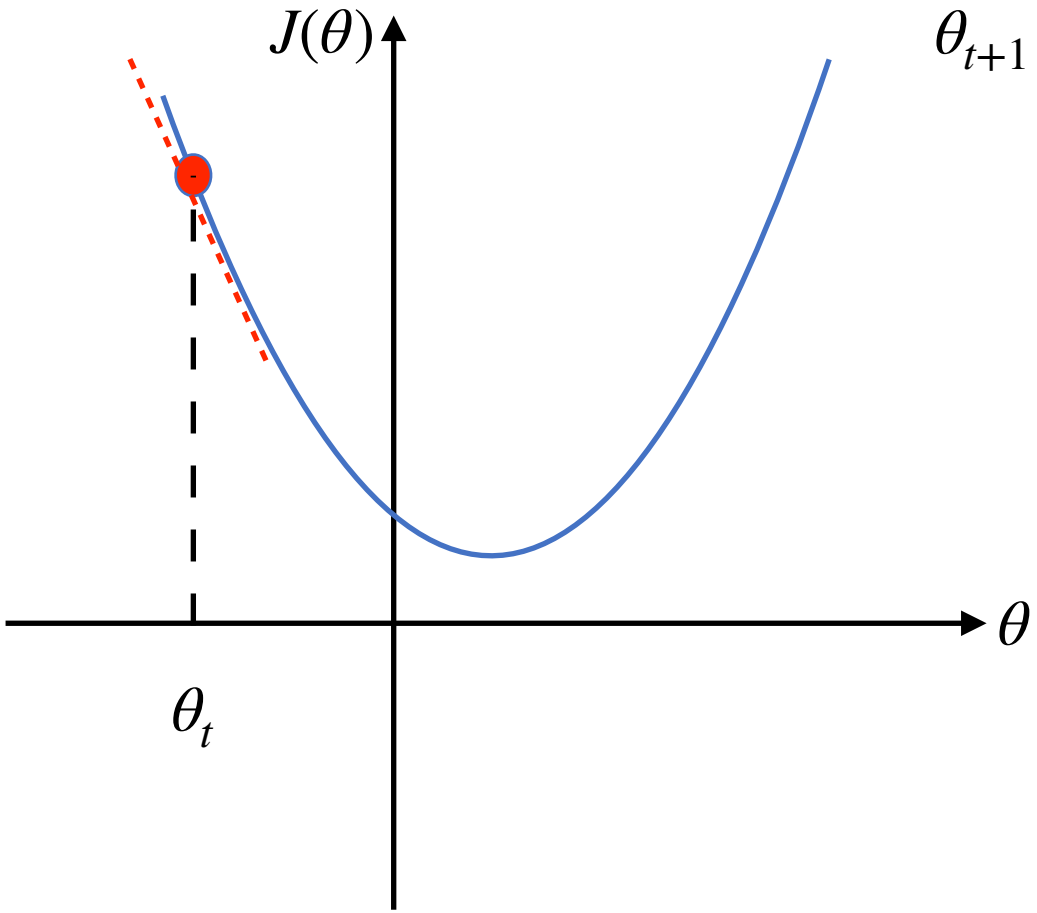
$$\theta_{t+1} := \theta_t - \alpha \underbrace{\triangle J(\theta_t)}_{0}$$

Gradient descent



$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

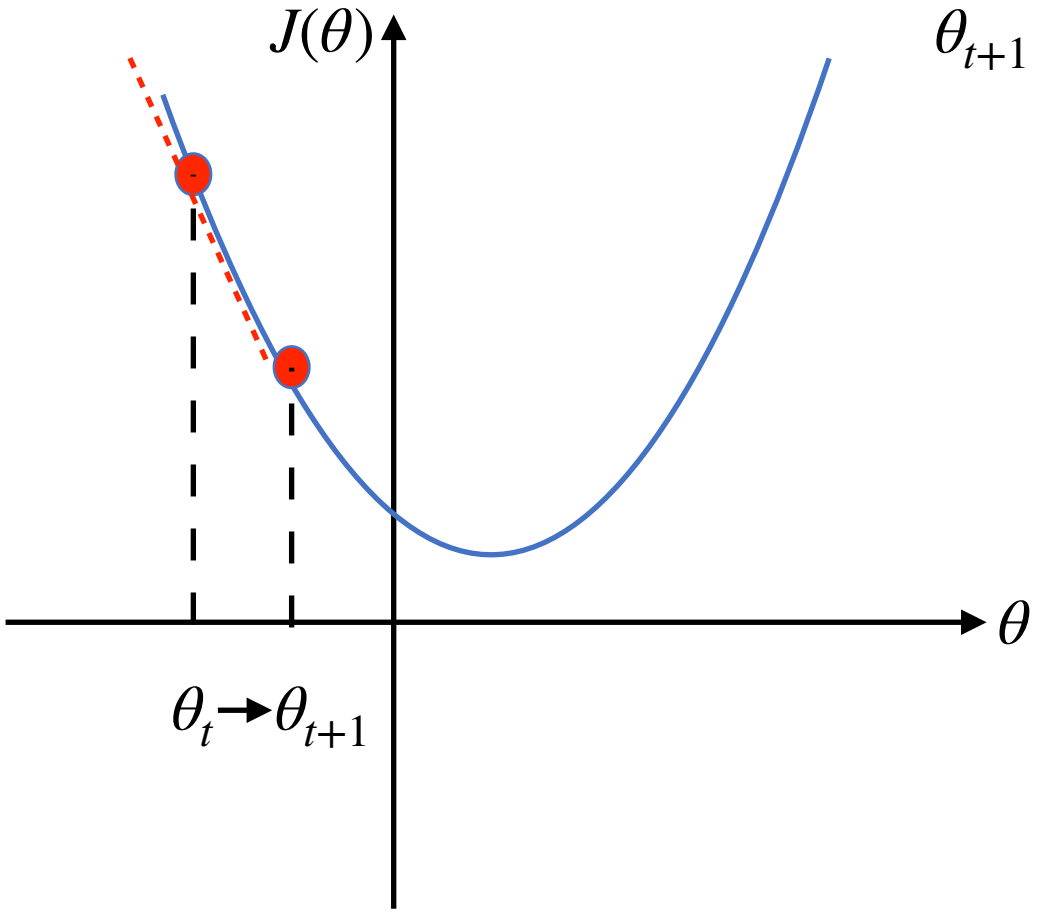
Gradient descent



$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

Negative

Gradient descent

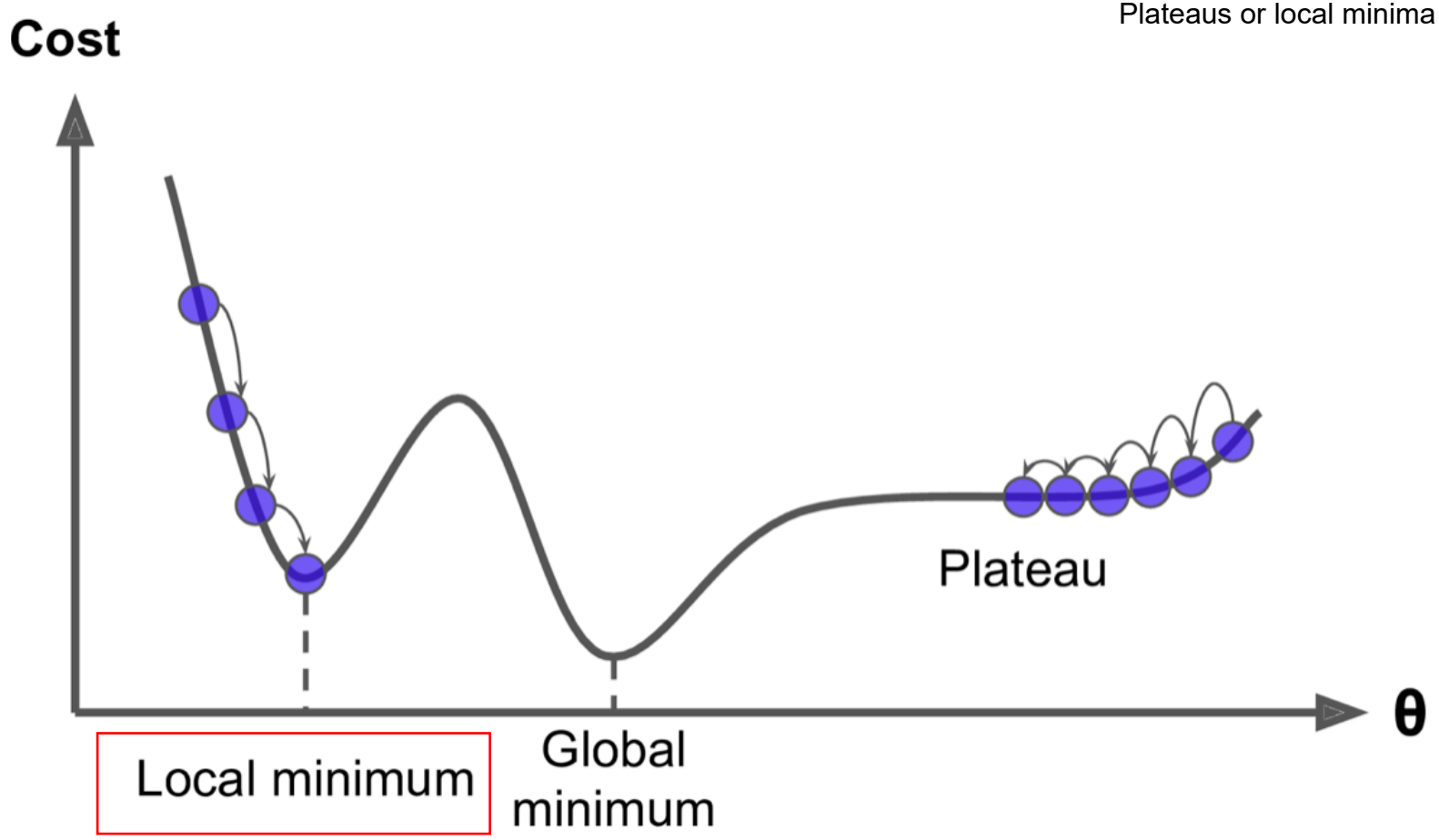


$$\theta_{t+1} := \theta_t - \alpha \triangle J(\theta_t)$$

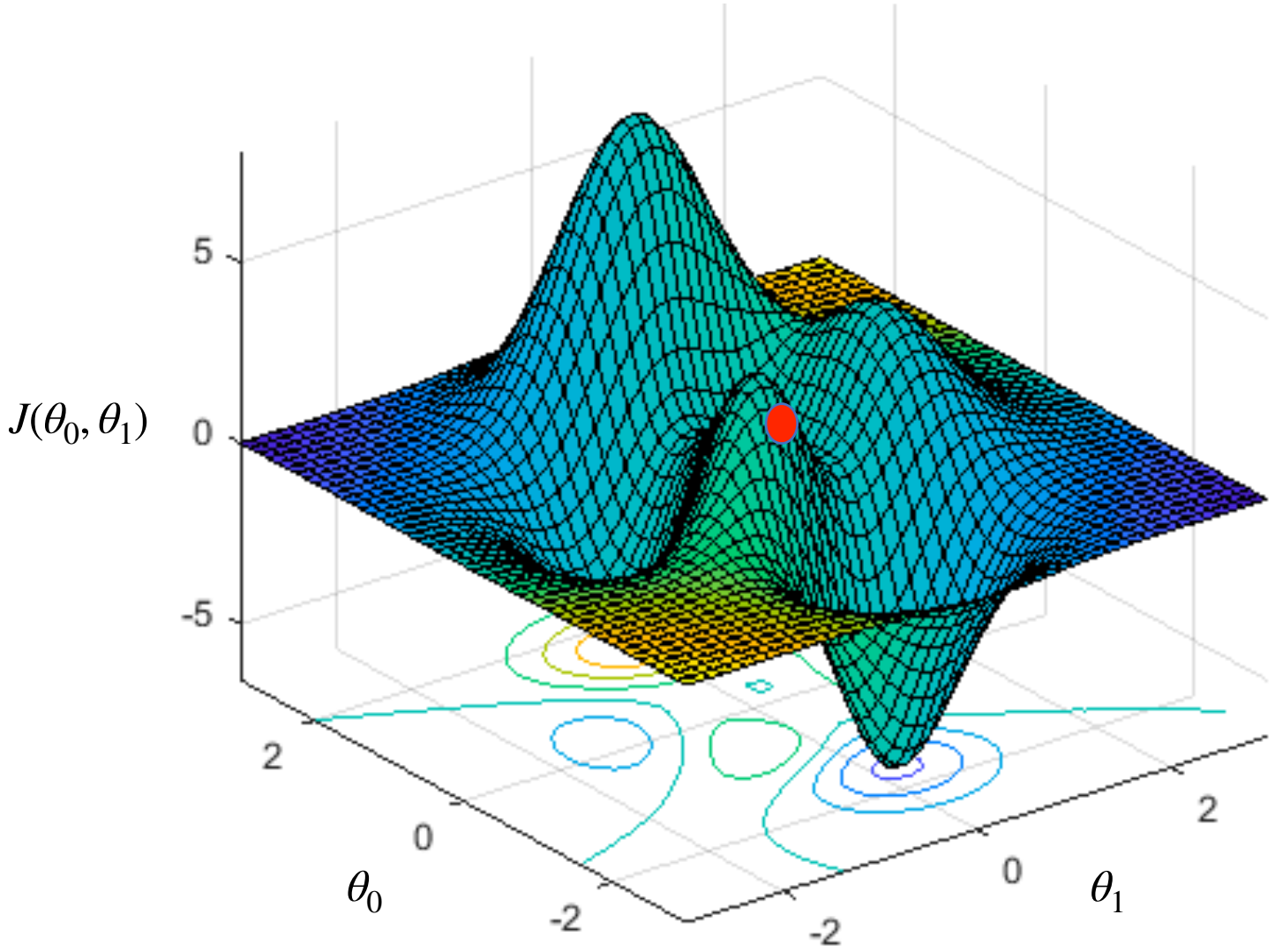
Negative

Optimization pitfalls

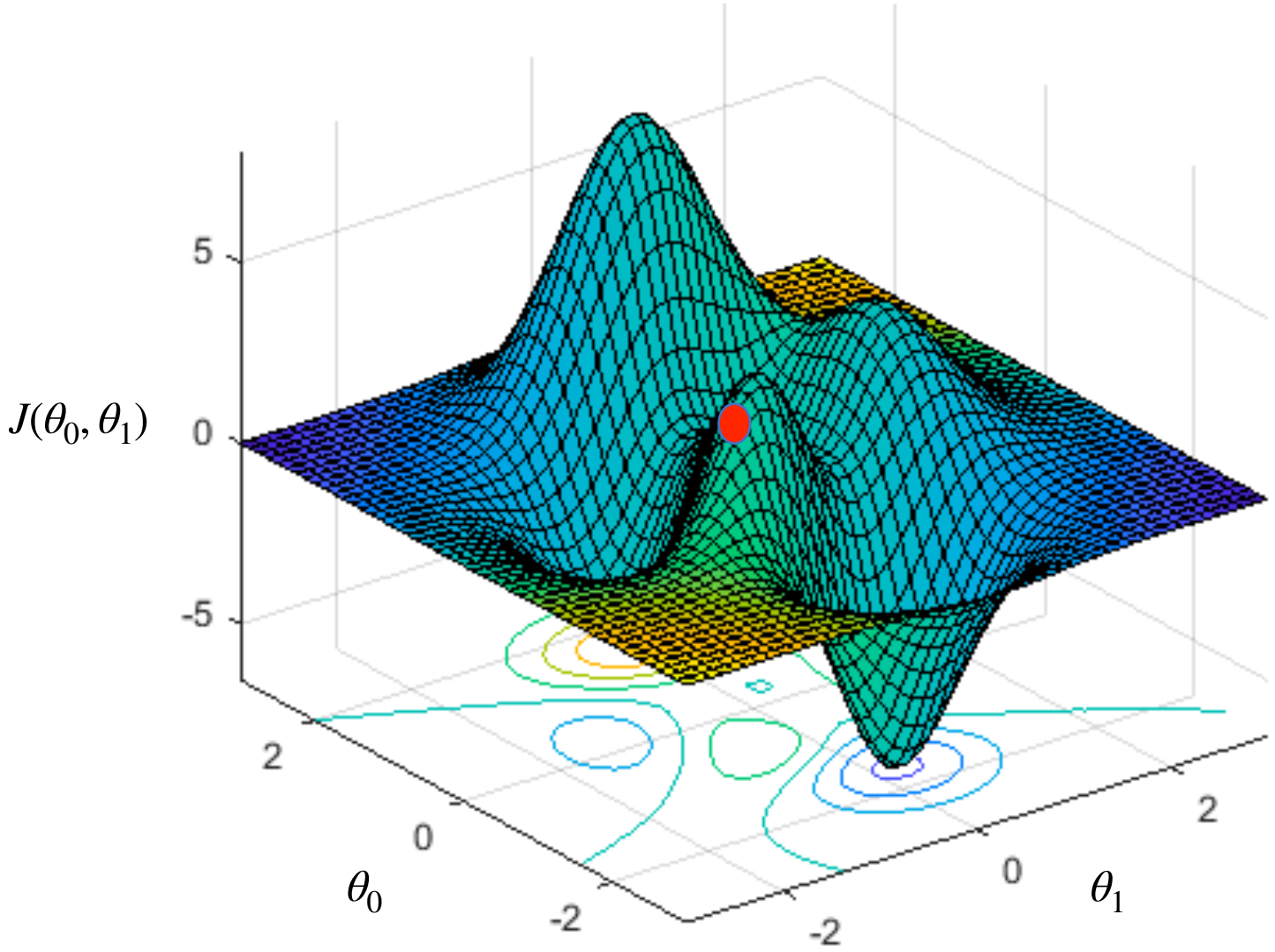
Plateaus or local minima might prevent reaching a global minimum.



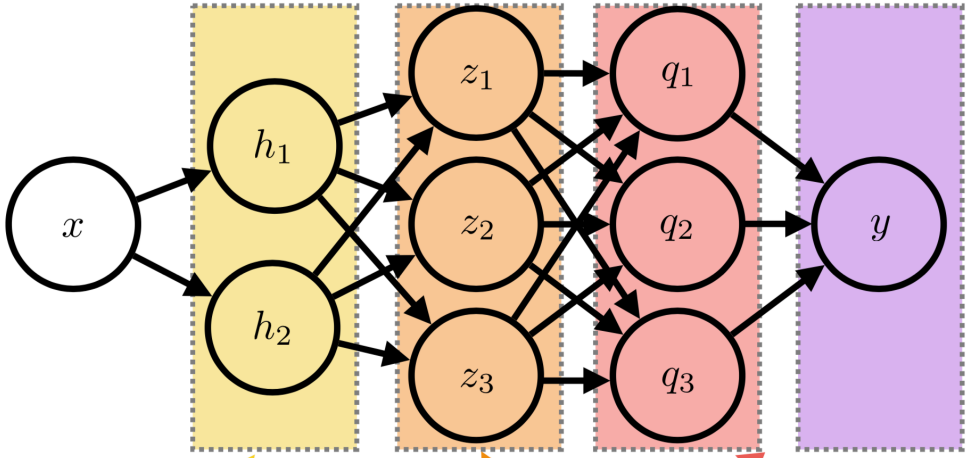
Optimization pitfalls



Optimization pitfalls

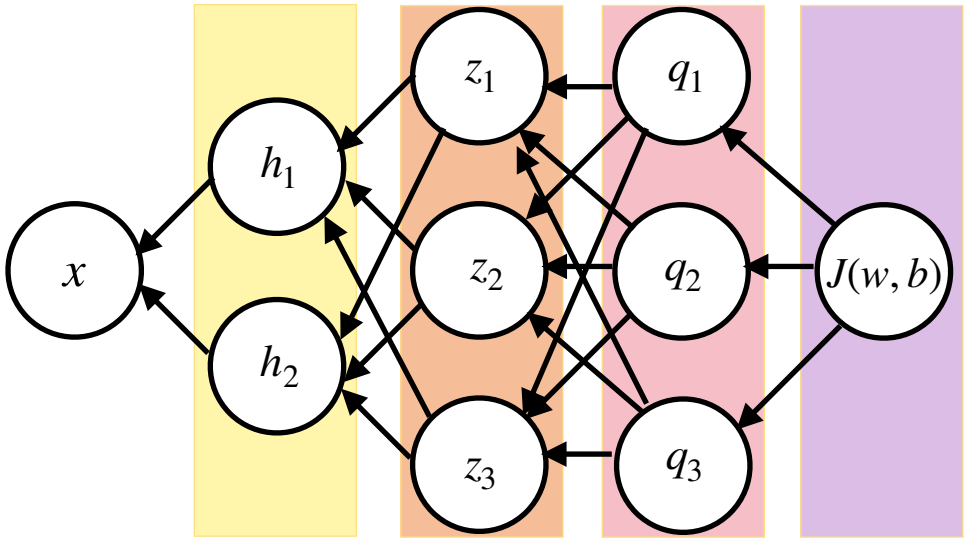


Backpropagation



Forward propagation

$$\hat{y} = f_4(f_{3,1}(f_{2,1}(f_{1,1}(x), f_{1,2}(x)), \dots), \dots)$$



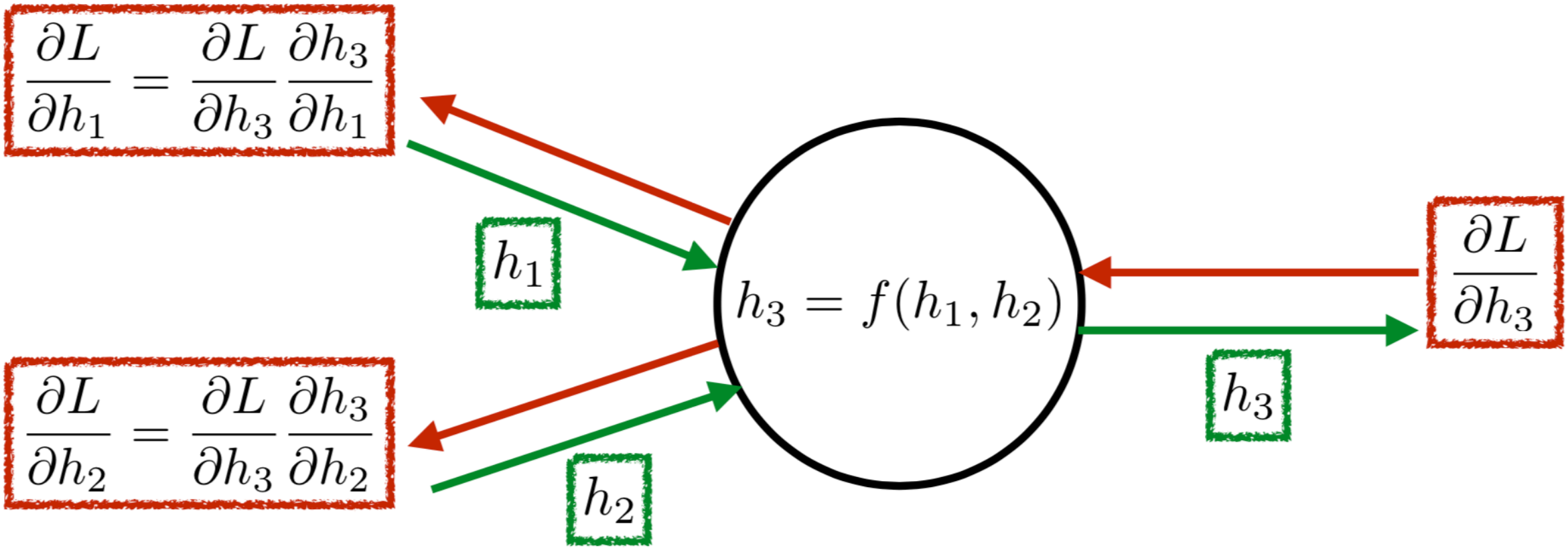
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

Back propagation

Backpropagation

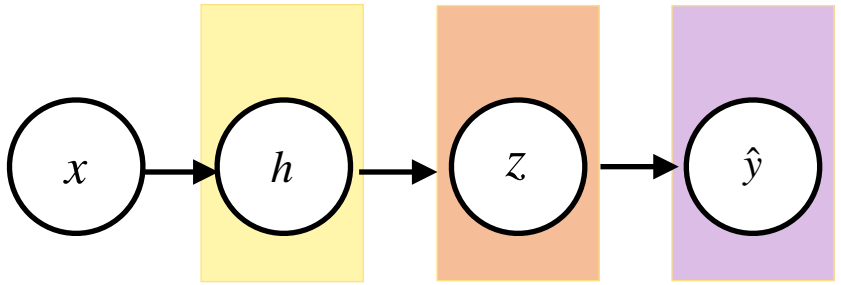
- Efficient implementation of the chain-rule to compute derivatives with respect to network weights

Calculating gradient and optimizing weights with chain rule.



Backpropagation

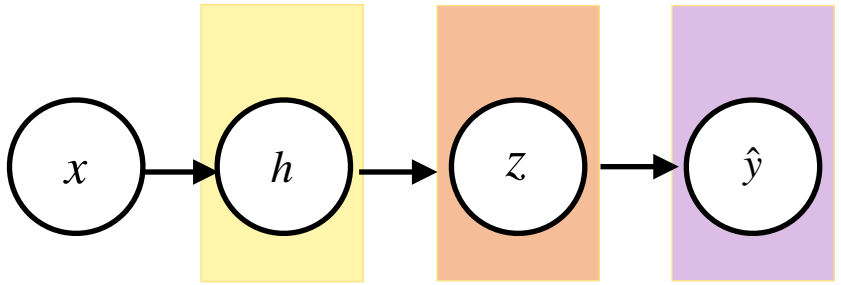
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$



$$h = f_1(x) \qquad z = f_2(h) \qquad \hat{y} = f_3(z)$$

Backpropagation

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

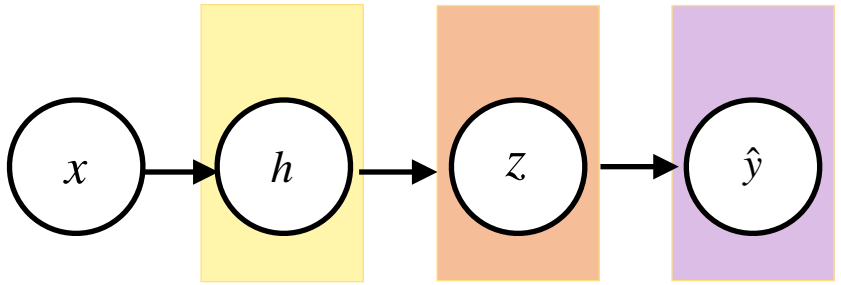


$$h = f_1(x) \qquad z = f_2(h) \qquad \hat{y} = f_3(z)$$
$$= w_1 x + b_1 \quad = w_2 x + b_2 \quad = w_3 x + b_3$$

$$f = wx + b$$

Backpropagation

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$



$$J(w_1, b_1, w_2, b_2, w_3, b_3)$$

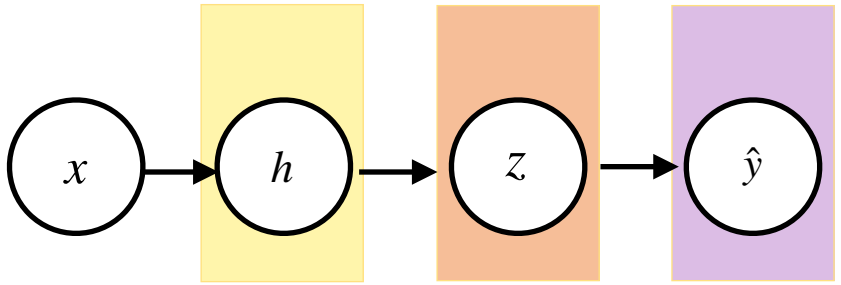
$$\triangle J(w_1, b_1, w_2, b_2, w_3, b_3) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial b_2}, \frac{\partial J}{\partial w_3}, \frac{\partial J}{\partial b_3}\right)$$

$$h = f_1(x) \qquad z = f_2(h) \qquad \hat{y} = f_3(z)$$
$$= w_1 x + b_1 \quad = w_2 x + b_2 \quad = w_3 x + b_3$$

$$f = wx + b$$

Backpropagation

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$



$$J(w_1, b_1, w_2, b_2, w_3, b_3)$$

$$\triangle J(w_1, b_1, w_2, b_2, w_3, b_3) = (\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial b_2}, \frac{\partial J}{\partial w_3}, \frac{\partial J}{\partial b_3})$$

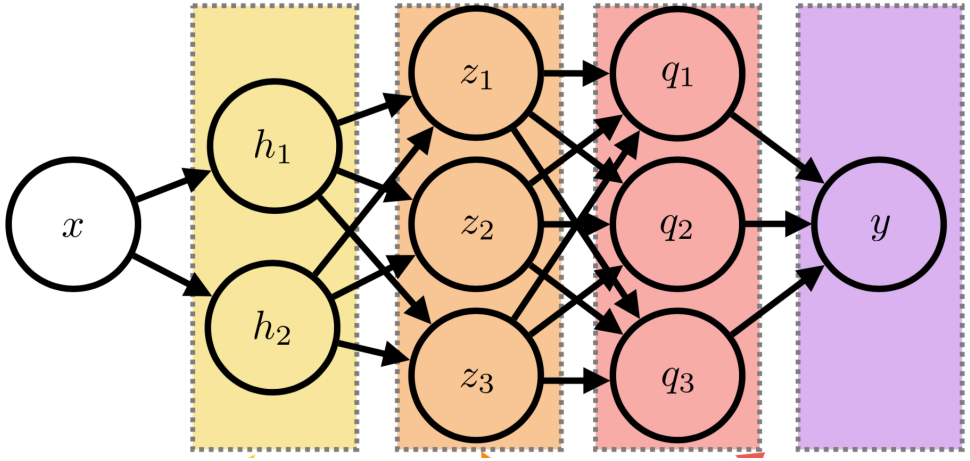
$$h = f_1(x) \qquad z = f_2(h) \qquad \hat{y} = f_3(z)$$

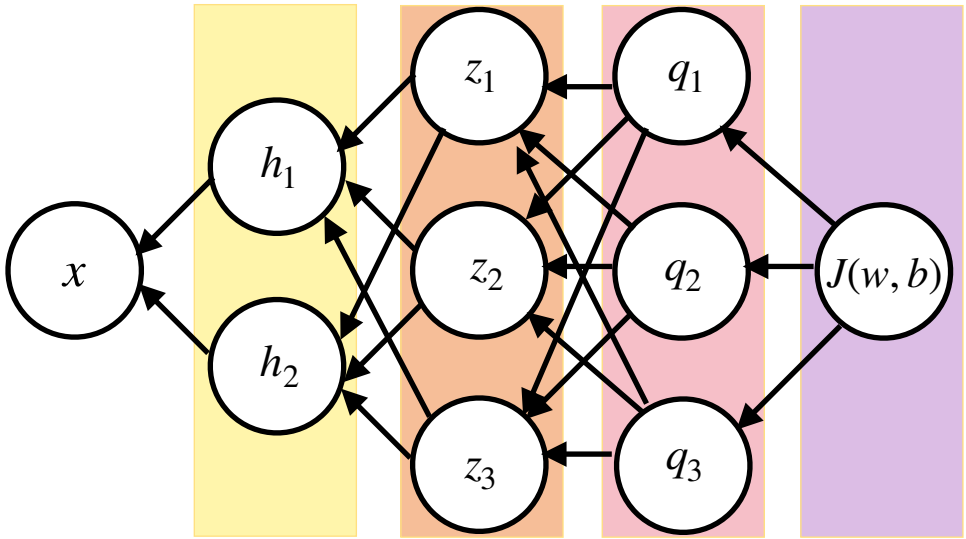
$$= w_1 x + b_1 \qquad = w_2 x + b_2 \qquad = w_3 x + b_3$$

$$\frac{\partial J}{\partial w_1} = (\frac{\partial J}{\partial f_3}\frac{\partial f_3}{\partial f_2}\frac{\partial f_2}{\partial f_1}\frac{\partial f_1}{\partial w_1})$$

$$f = wx + b$$

Training iteration





1. Forward propagation, get output
2. Compute loss
3. Back propagation, update parameters
4. Forward propagation again with updated parameters, get new output
5. Repeat step 1-4 until converge

Loss function choice

Loss function choice depends on what you want to achieve with the model.

- Choice determined by the output representation
  - Probability vector (**classification**) : Cross-entropy

$$\hat{y} = \sigma(w^\top h + b) \qquad p\left(y \middle| \hat{y}\right) = \hat{y}^y \left(1 - \hat{y}\right)^{(1-y)}$$

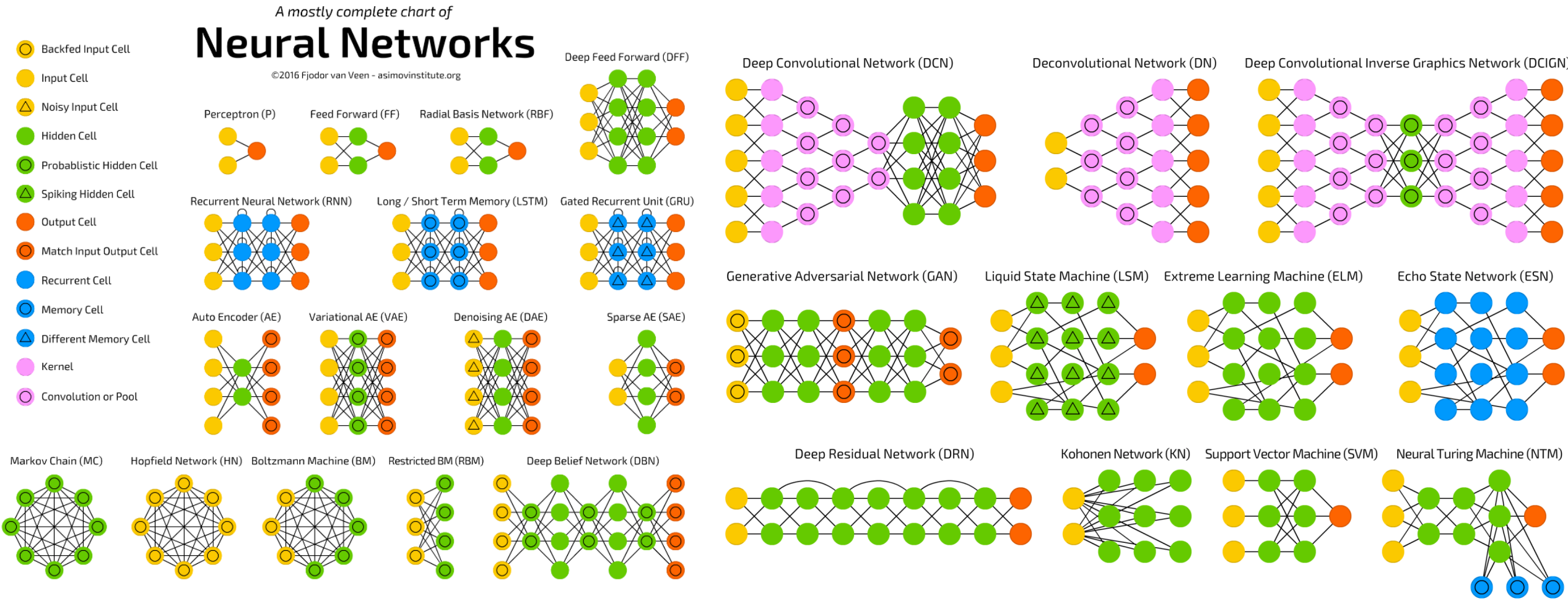
$$L\left(\hat{y}, y\right) = -\log p\left(y \middle| \hat{y}\right) = -\left(y \log\left(\hat{y}\right) + (1-y)\log\left(1 - \hat{y}\right)\right)$$ **(binary classification)**

  - Mean estimate (**regression**) : Mean Squared Error, L2 loss

$$\hat{y} = W^\top h + b \qquad p\left(y \middle| \hat{y}\right) = \mathrm{N}(y; \hat{y})$$

$$L_2\left(\hat{y}, y\right) = -\log p\left(y \middle| \hat{y}\right) = \sum_{i=0}^{m} \left(y^i - \hat{y}^i\right)^2$$

1) Network Model

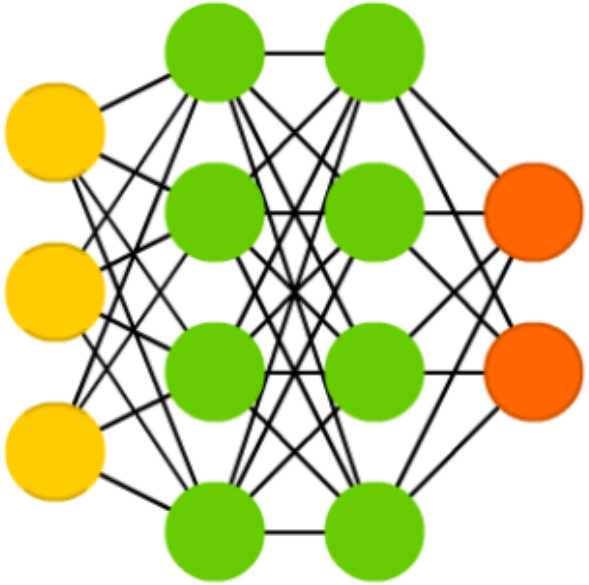


(Deep) Feedforward NN (DFF)

- the simplest type of neural network
- All units are fully connected
- information flows from input to output layer without back loops
- The first single-neuron network was proposed already in 1958 by AI pioneer Frank Rosenblatt
- Deep for “more than 1 hidden layer”

Feed Forward (FF)

Deep Feed Forward (DFF)



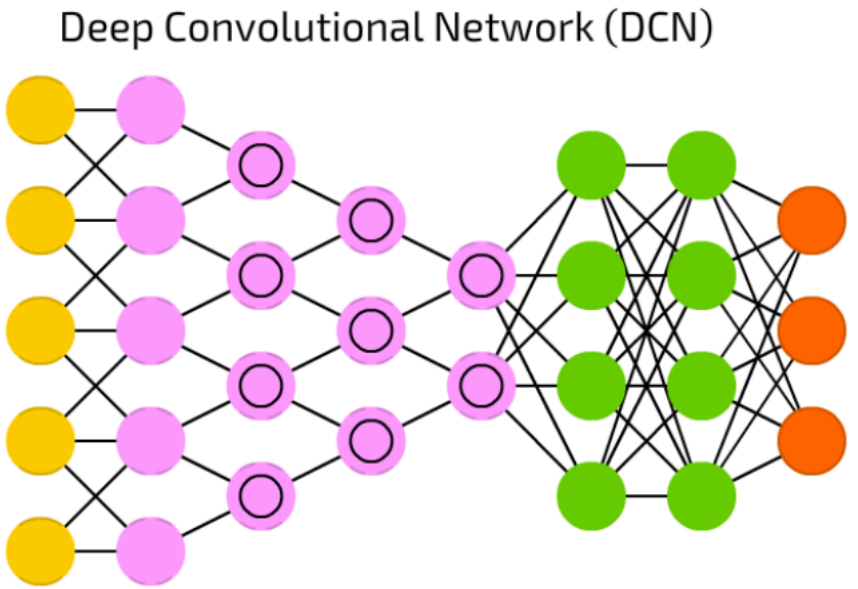
1 hidden layer networks are simply called "feedforward NN", with more than one hidden layer we call it a "deep feedforward NN"

Convolutional Neural Networks (CNN)

- inspired by the organization of the animal visual cortex

- Kernel and convolution or pool cells used to process and simplify input data
  - Weight sharing between local regions

- well suited for computer vision tasks
  - Image classification
  - Object detection

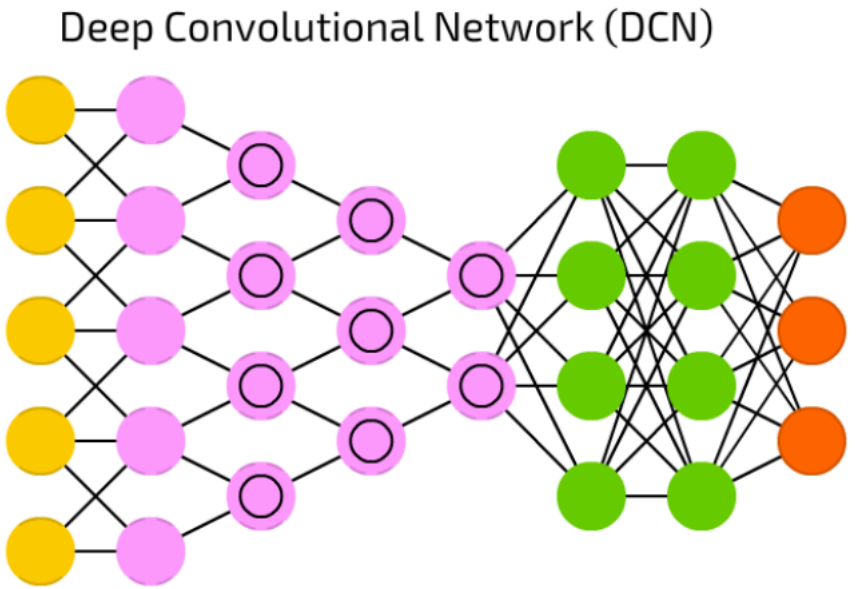


Convolutional Neural Networks (CNN)

- inspired by the organization of the animal visual cortex

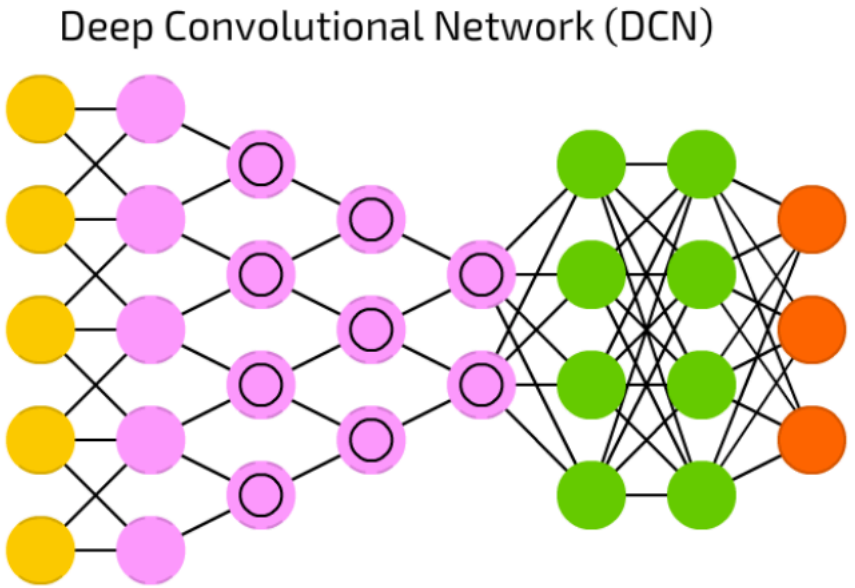
- **Kernel and convolution or pool cells** used to process and simplify input data
  - Weight sharing between *local regions*

- well suited for computer vision tasks
  - Image classification
  - Object detection



Convolutional Neural Networks (CNN)

- inspired by the organization of the animal visual cortex
- **Kernel and convolution or pool cells** used to process and simplify input data
  - Weight sharing between *local regions*
- well suited for computer vision tasks
  - Image classification
  - Object detection



In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

Types of layer in CNN

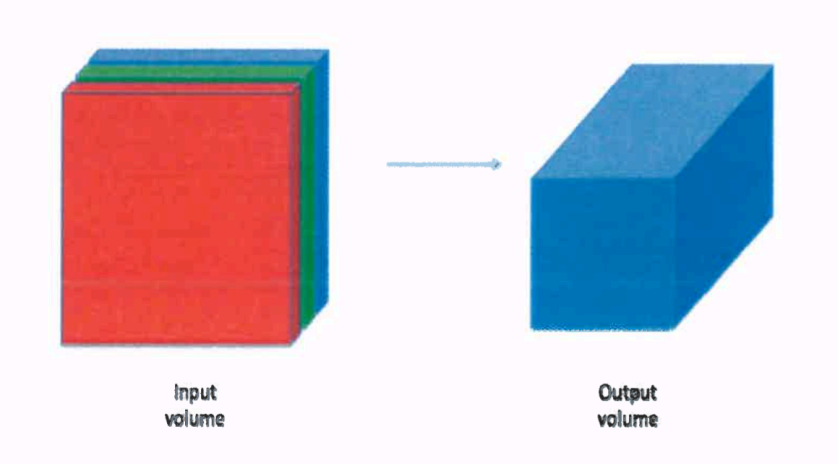
- Convolution (CONV)
- Pooling (POOL)
- Fully connected (FC)
- *In feature learning tasks, usually multiple CONV layers followed by a POOL layer, and FC layers in the last few layers*

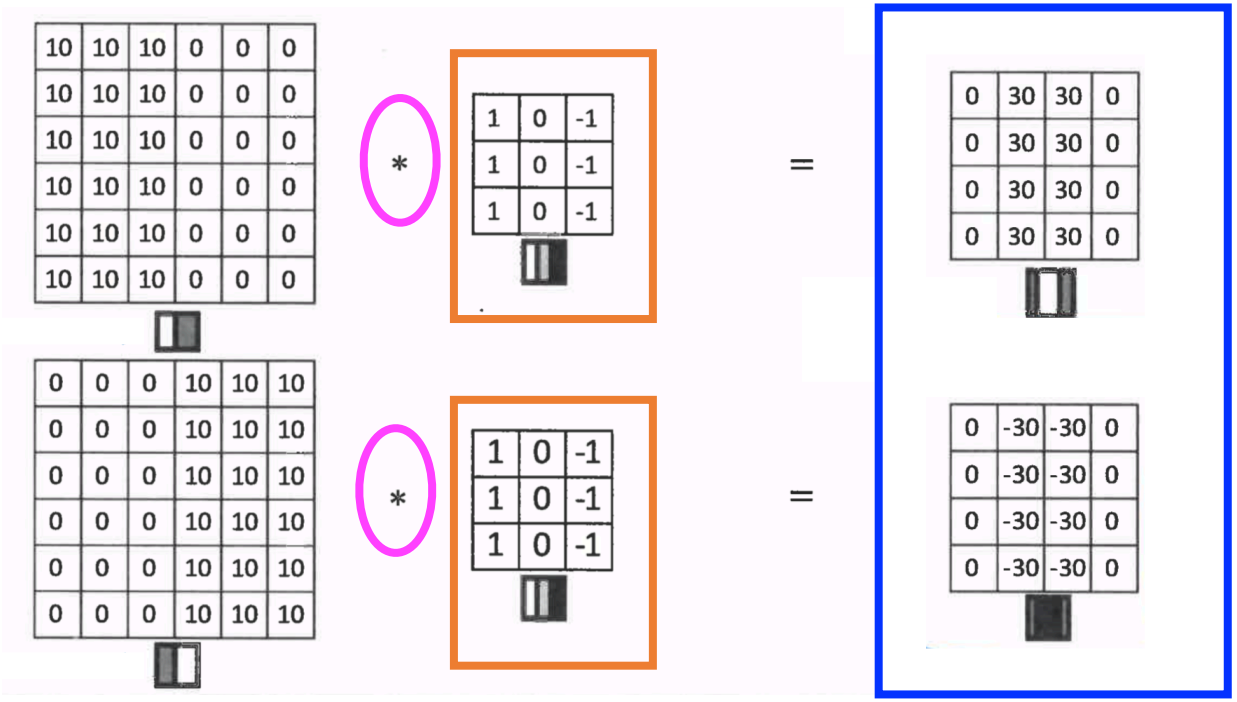
Convolution Layer (CONV)

- **Convolution** transforms an input volume into an output volume of different size, also called feature map

- **Filter kernels** are used to detect features (for example, edge detection in 1st hidden layer)

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image.





$n \times n$ image $\quad f \times f$ filter $\quad \text{out} = n - f + 1$

Padding

Adds zeros around the border of an image

This can be used if the dimension of the input data should remain the same.

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

Animation from :https://stackoverflow.com/questions/52067833/how-to-plot-an-animated-matrix-in-matplotlib

Padding

Adds zeros around the border of an image

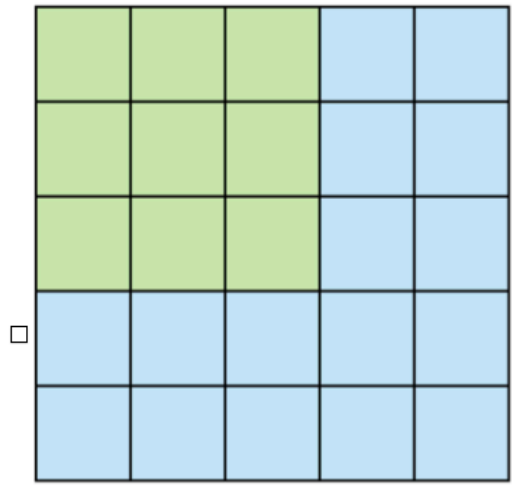
Use cases :

- Keeps more information at the border of an image
- Allows to *use a CONV layer without shrinking* the height and width of the volumes (important for deeper networks)

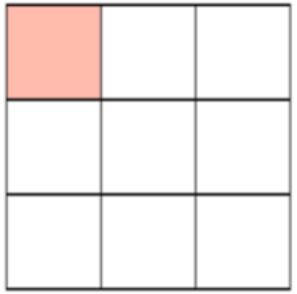
Strided convolutions

The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we can use a stride of 2 for downsampling an image similar to MaxPooling.

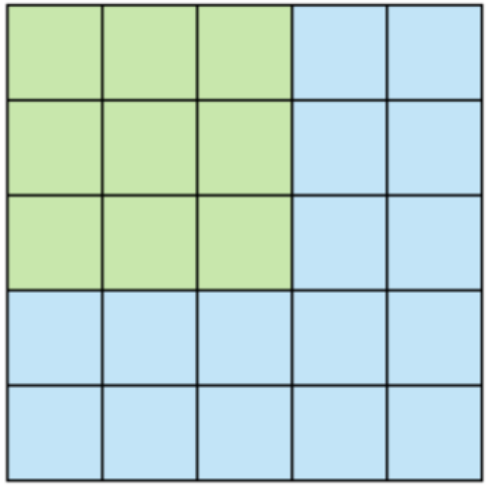
By how much you move the filter



Stride 1



Feature Map



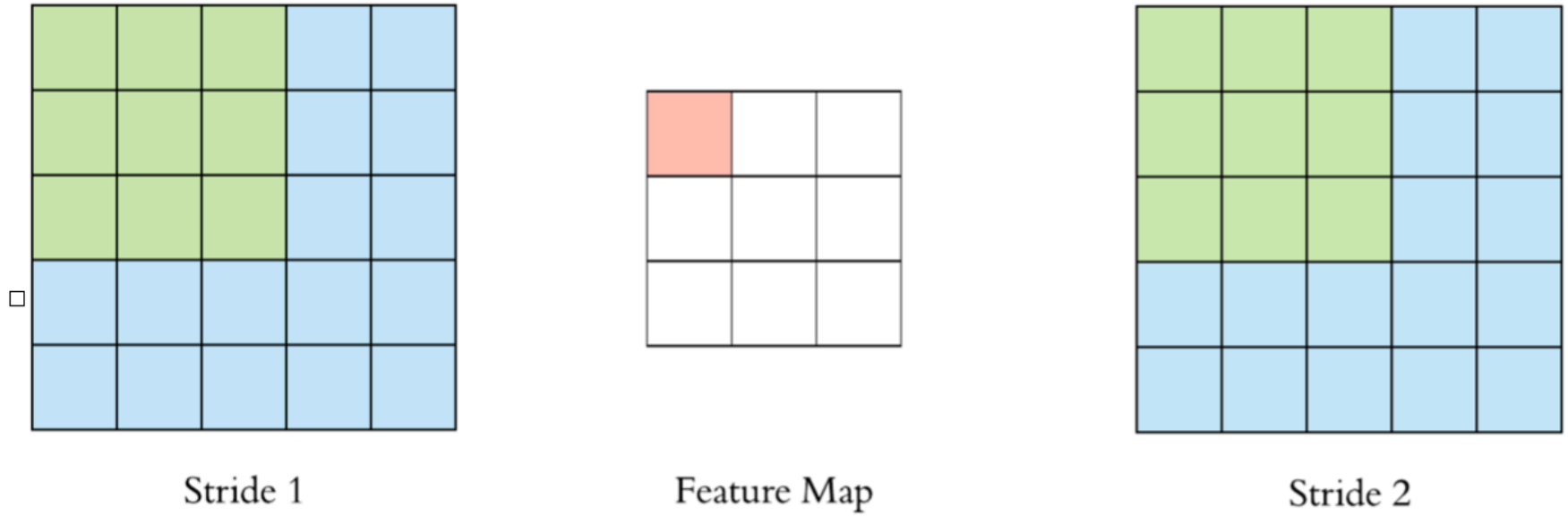
Stride 2

?

Increasing stride from 1 to 2

Strided convolutions

By how much you move the filter

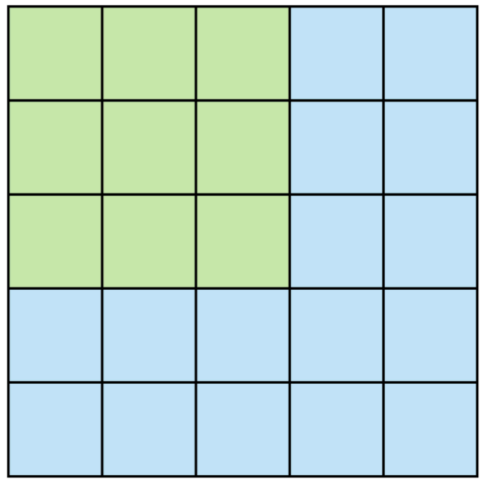


Increasing stride from 1 to 2

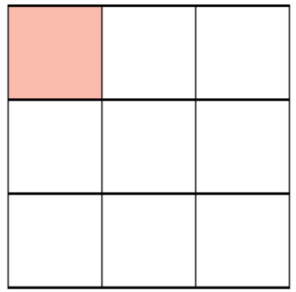
Strided convolutions

By how much you move the filter

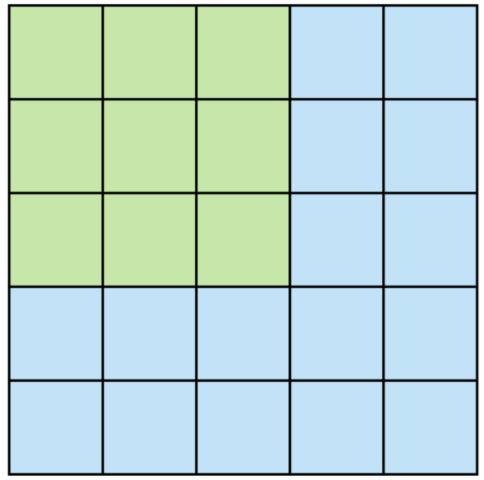
The stride defines the step size of the kernel when traversing the image. While its default is usually 1, we can use a stride of 2 for downsampling an image similar to MaxPooling.



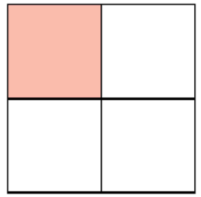
Stride 1



Feature Map



Stride 2



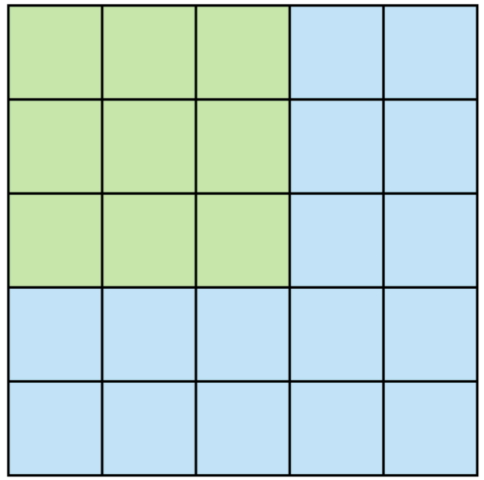
Feature Map

Increasing stride from 1 to 2

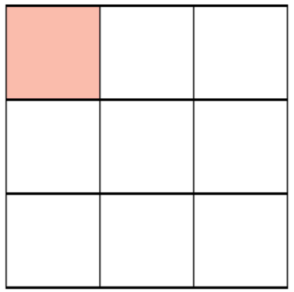
Strided convolutions

By how much you move the filter

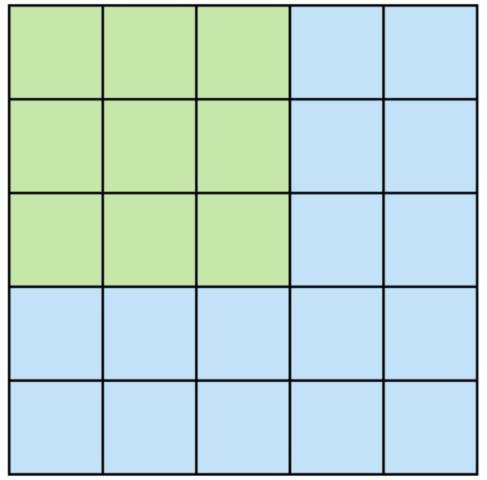
Downsampling



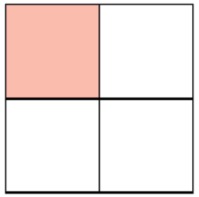
Stride 1



Feature Map



Stride 2



Feature Map

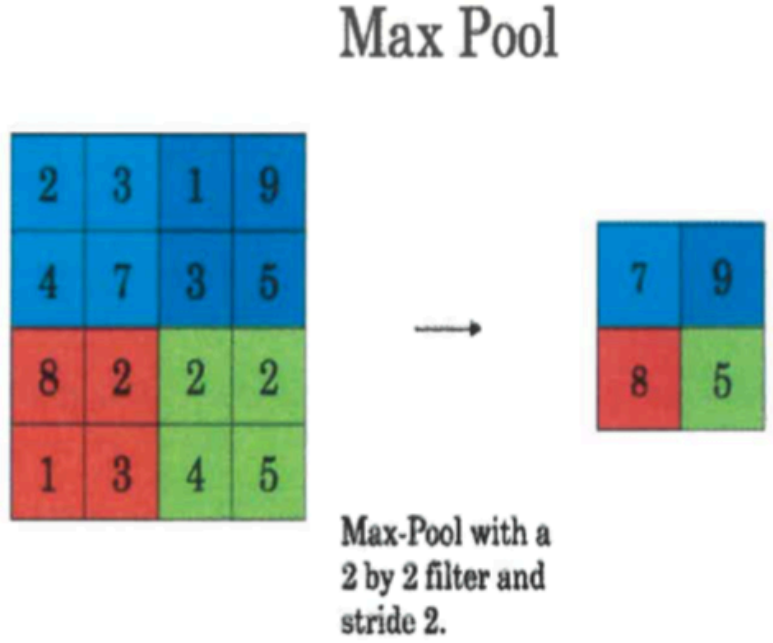
Increasing stride from 1 to 2

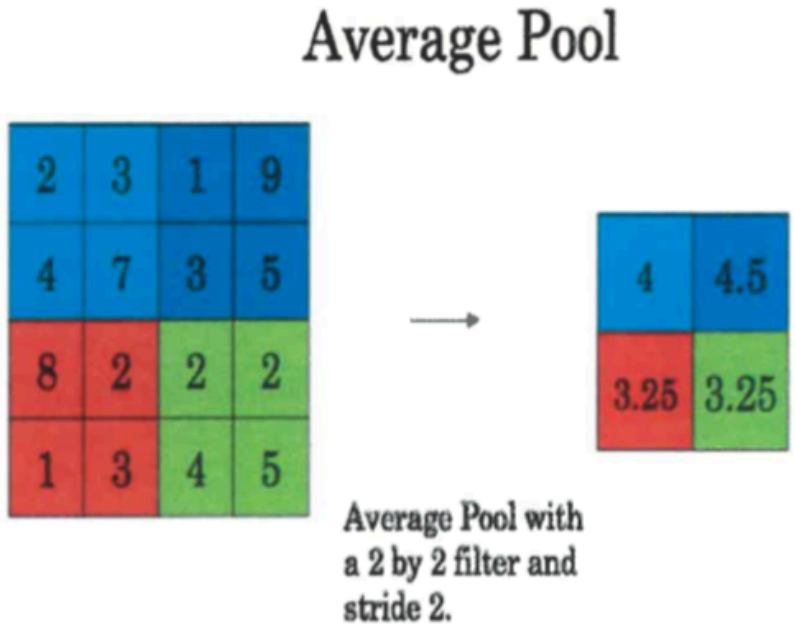
Pooling Layer (POOL)

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

- reduces the spatial dimension (n_H and n_W) to decrease resource (memory) load

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.



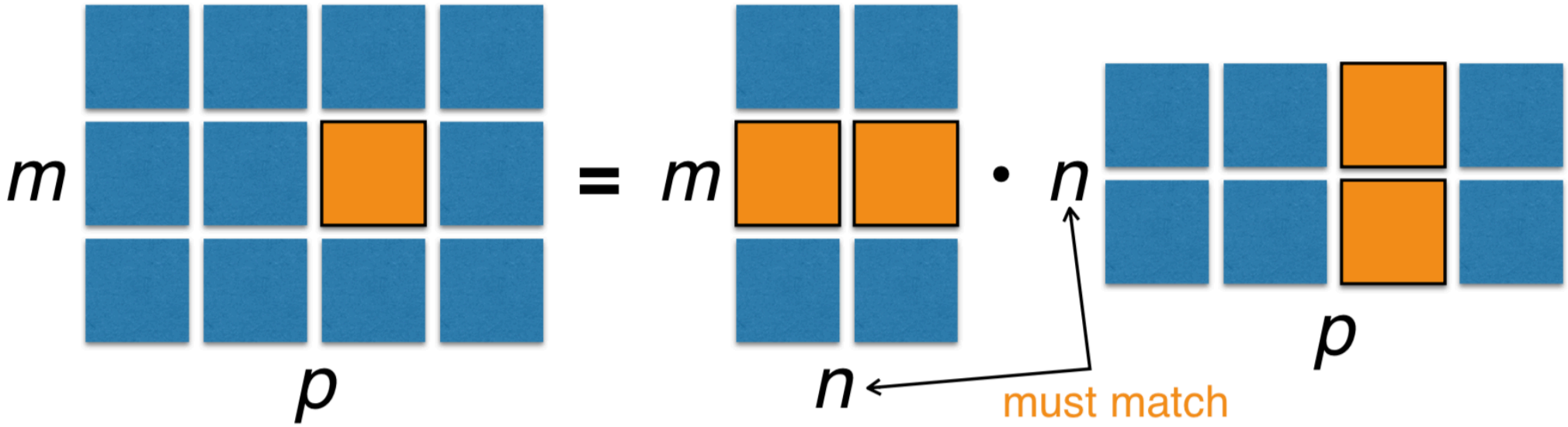


Get the max value

Get the average value

Max pooling also performs as a noise suppressant --> discarding noisy activations --> usually performs better than average pooling, since average pooling just takes the average of all activations.

Fully Connected Layer (FC)

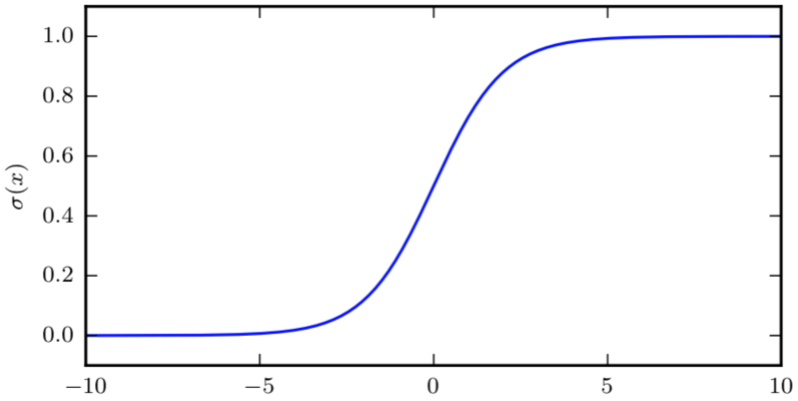


Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

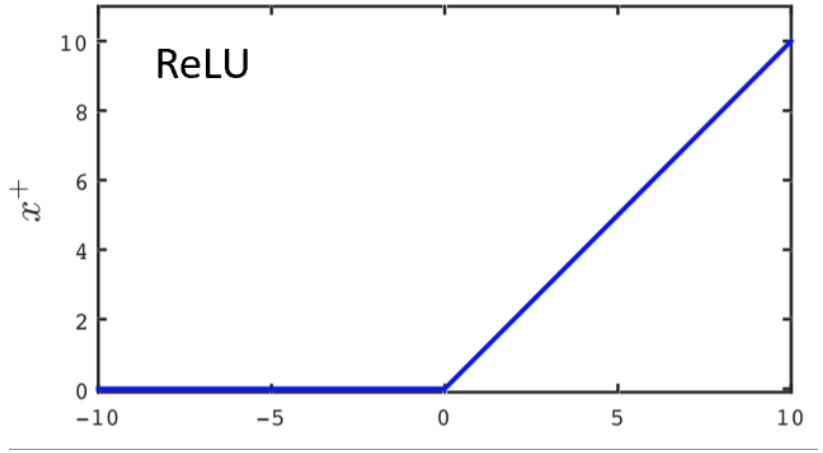
matrix product

Output layer : activation functions

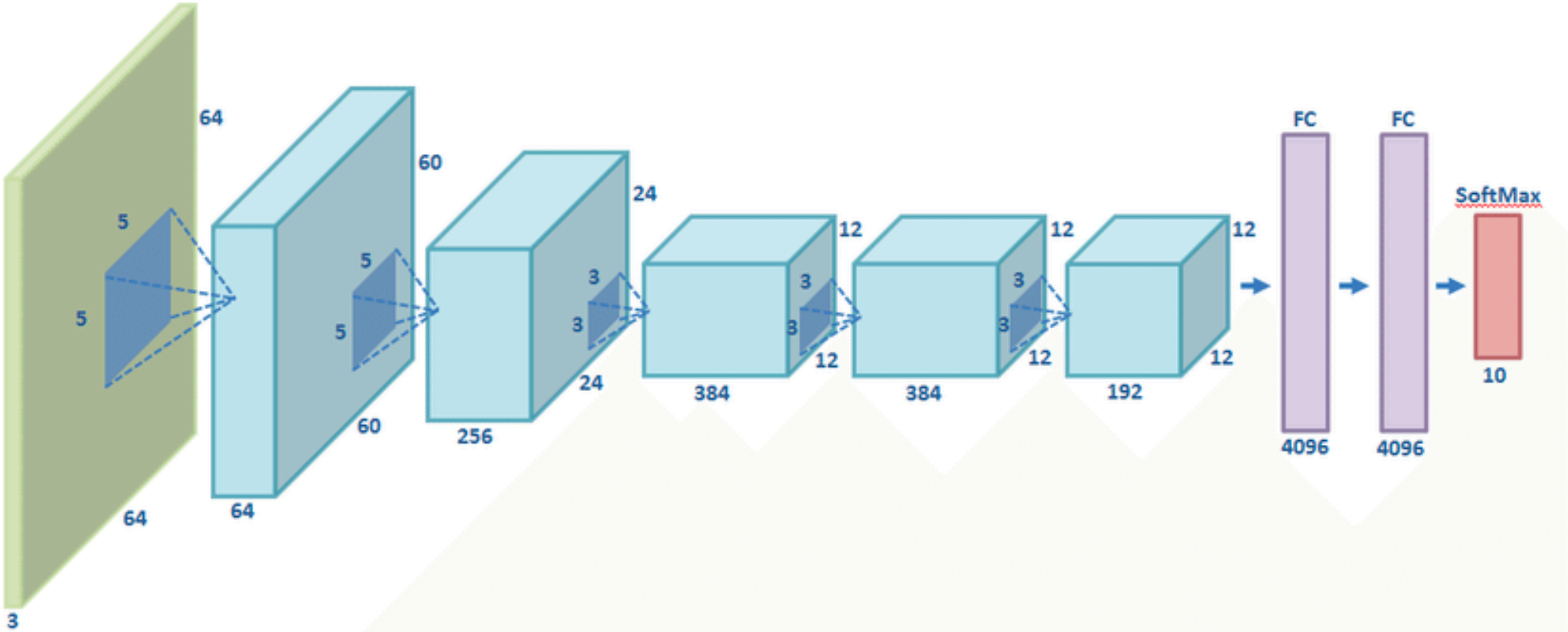
1) **Classification** : probability vector
   - Sigmoid (binary class)
   - Softmax (multiple class)



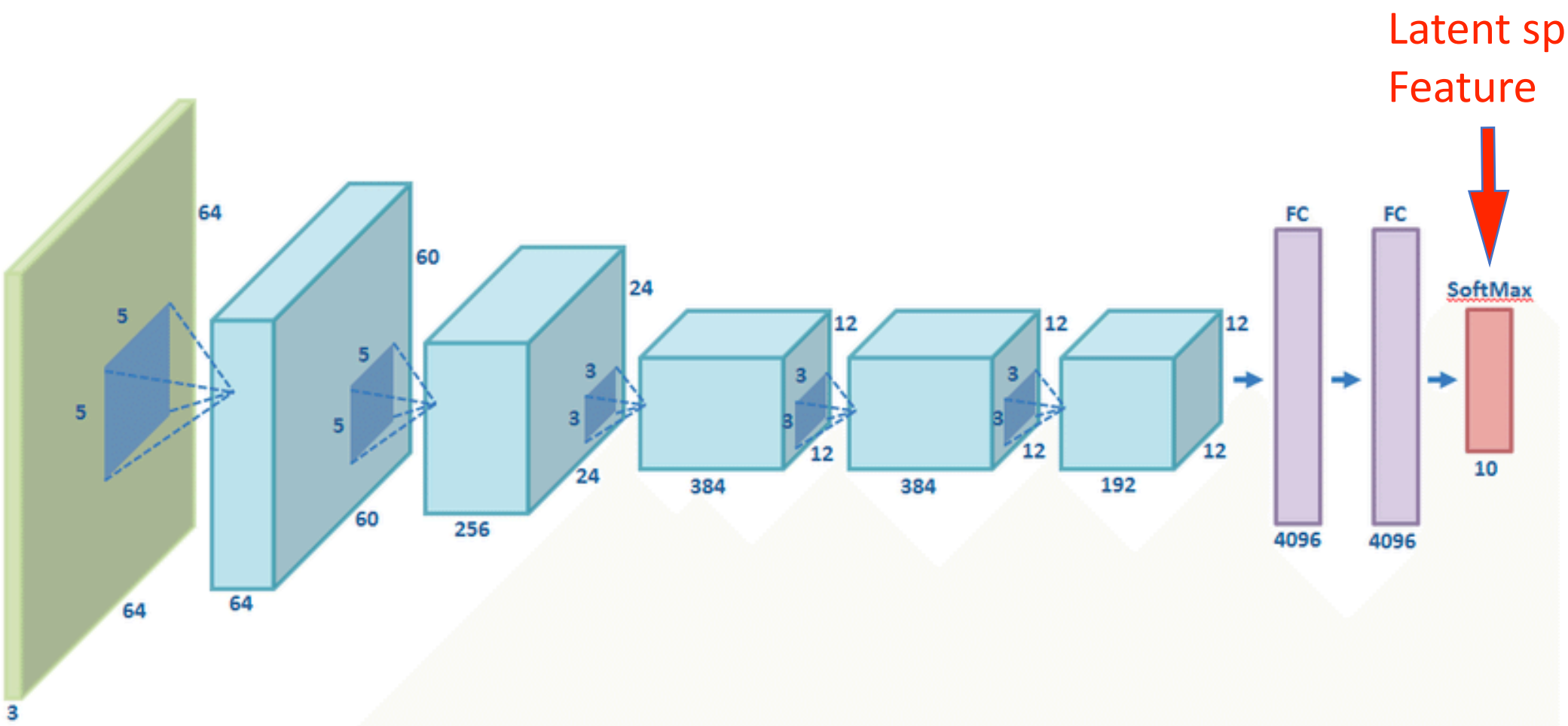
2) **Regression** : mean estimate
   - ReLU
   - Softplus
   - Smoothed max
   - Generalization of ReLU (leaky ReLU,…)



Convolutional neural network



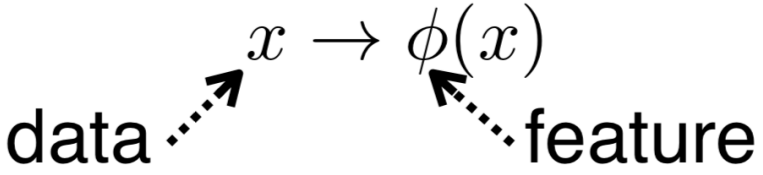
Convolutional neural network

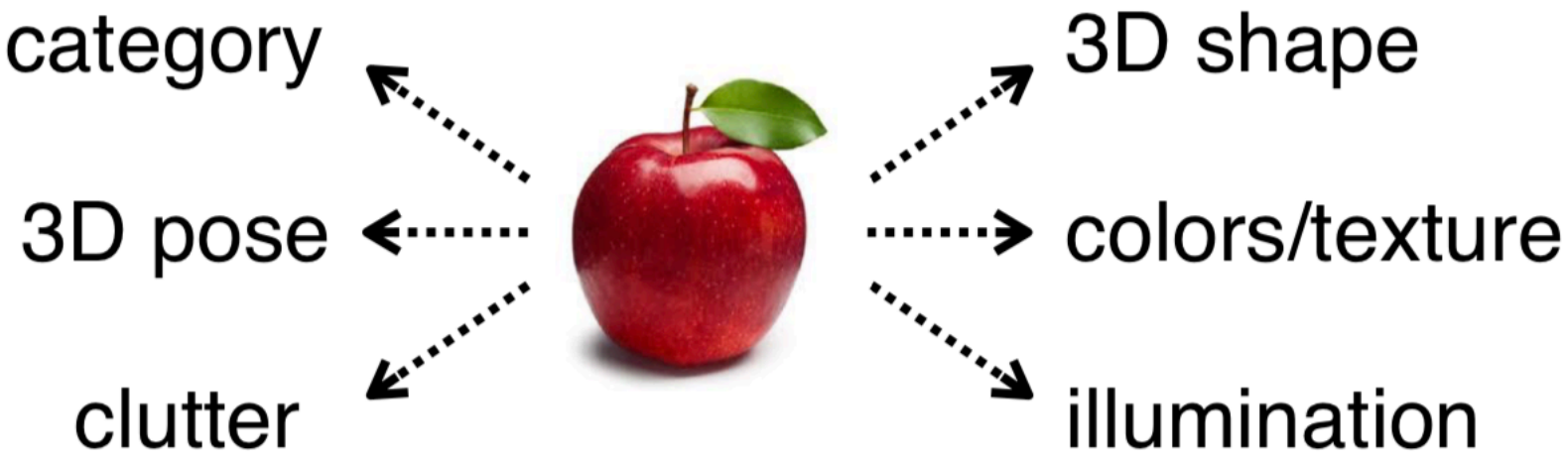


Features

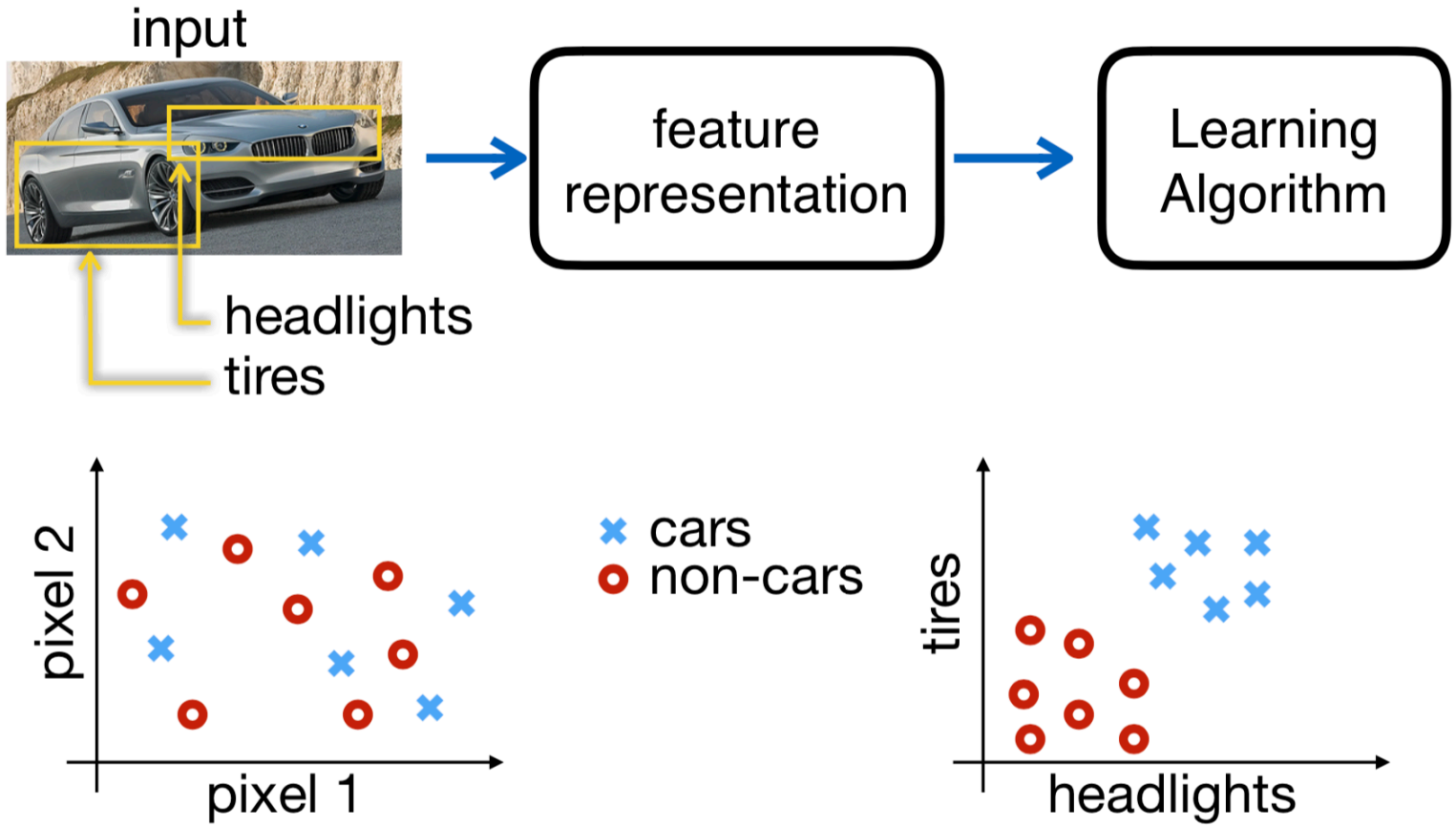
Features are high level data such as edges, color, shape, noise, etc. that can be detected with the right use of CNNs.

- Data often encoded into more focused relevant information (features or internal representation) to simplify the decision





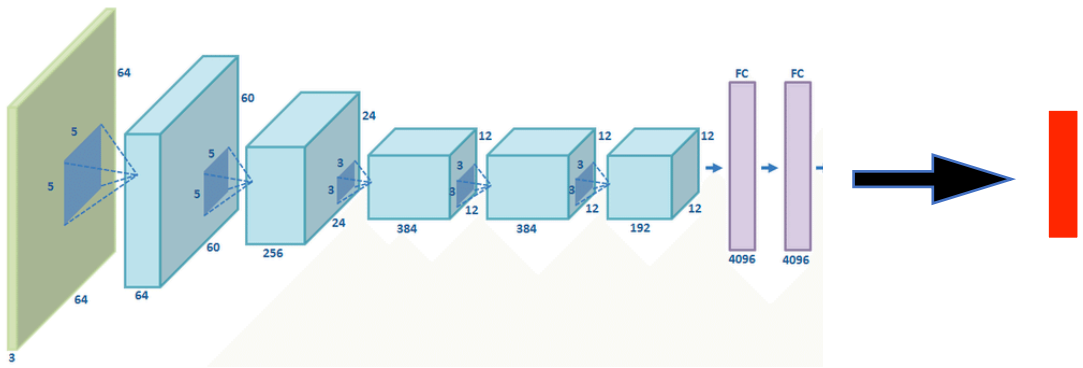
Features Example :Image classification



Convolutional neural network

Input

Latent space/
Feature



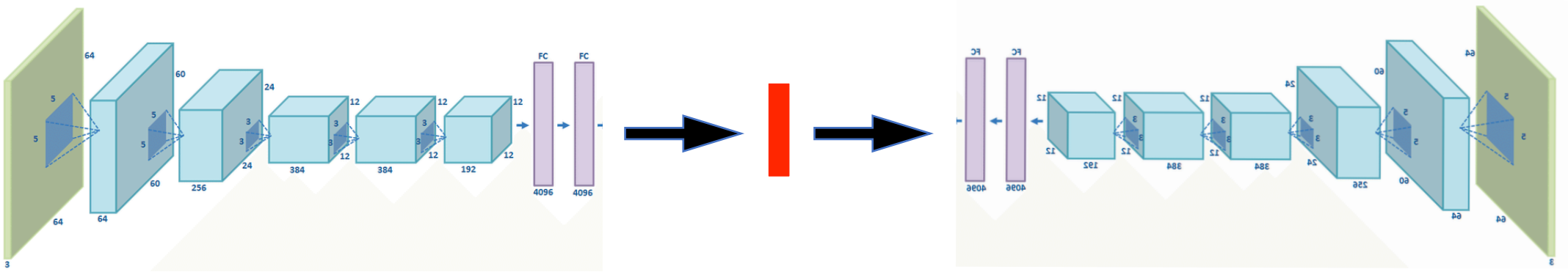
AutoEncoder

Autoencoder is an unsupervised artificial neural network that learns how to efficiently compress and encode data then learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.
Autoencoder, by design, reduces data dimensions by learning how to ignore the noise in the data.

Input

Latent space/
Feature

Reconstruction



1- Encoder: In which the model learns how to reduce the input dimensions and compress the input data into an encoded representation.

2- Bottleneck: which is the layer that contains the compressed representation of the input data. This is the lowest possible dimensions of the input data

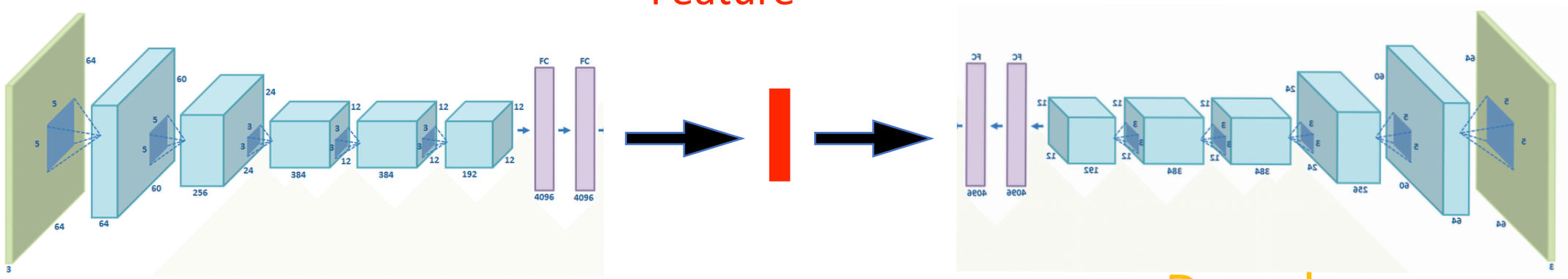
3- Decoder: In which the model learns how to reconstruct the data from the encoded representation to be as close to the original input as possible.

AutoEncoder

Input

Latent space/
Feature

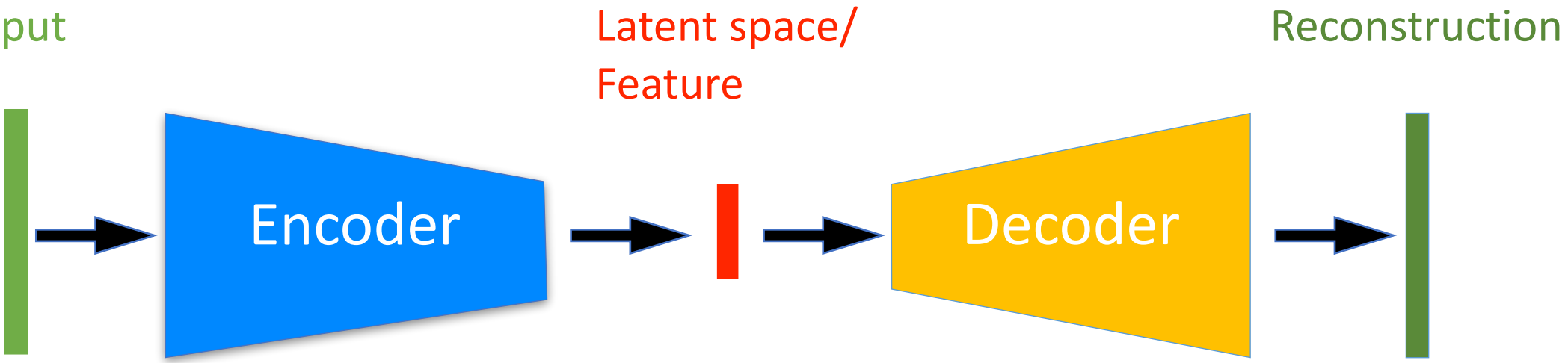
Reconstruction



Encoder

Decoder

AutoEncoder



Tutorial 1 : 11h-12h15:
Introduction to Tensorflow

Tutorial 2: 17h-18h30
Optimization in Tensorflow
and NN introduction