CONVOLUTIONAL NEURAL NETWORKS

Szabolcs Pável
(Ștefan Máthé)

BOSCH

Outline

- Math Refresher: Calculus
- Optimization for Deep Learning
- Convolutional Neural Networks
- Perspectives

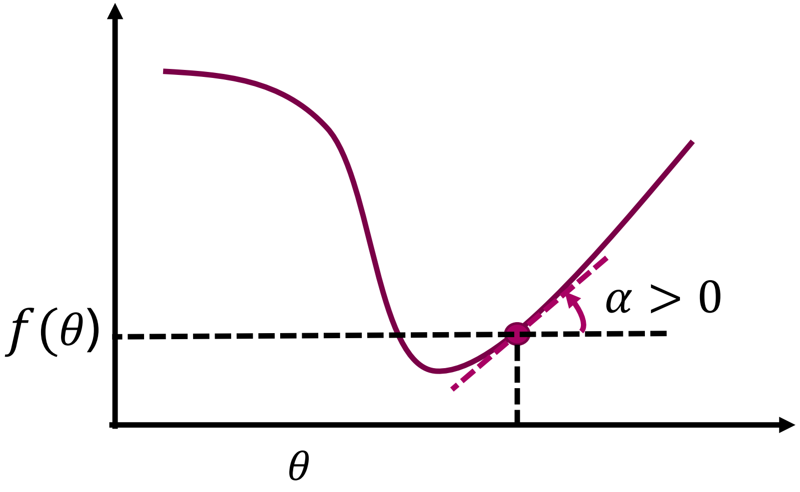
MATH REFRESHER: CALCULUS

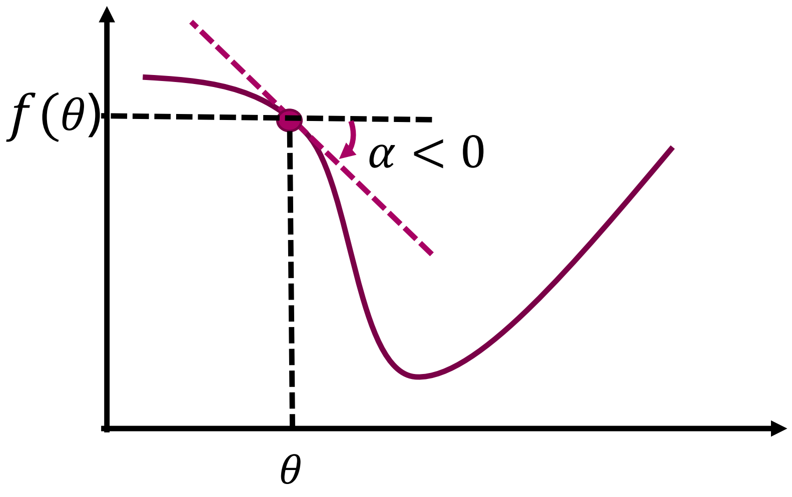
Quick Math Refresher
The Derivative

- Describes the sensitivity of a 1 dimensional scalar function to an infinitesimal change in input

$$f: \mathbb{R} \rightarrow \mathbb{R} \qquad f'(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta + \varepsilon) - f(\theta)}{\varepsilon} = \tan \alpha$$

- The derivative is a scalar
- Interpretation: the tangent of the slope of the function





BOSCH

Quick Math Refresher
The Partial Derivative

- Assume we have a d-dimensional scalar function:

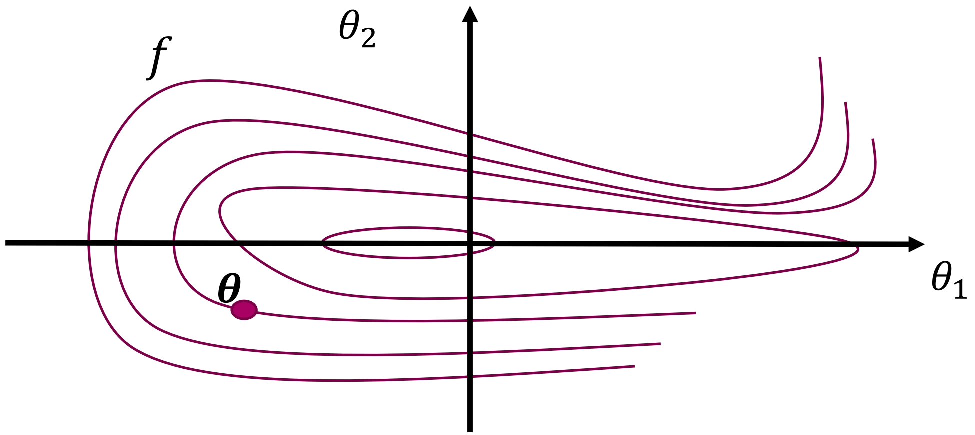
$$f: \mathbb{R}^d \to \mathbb{R}$$

- The partial derivative for dimension i describes how sensitive f is to infinitesimal changes along dimension i

θ_2

$$\frac{\partial f}{\partial \theta_i}(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta + \varepsilon \Delta_i) - f(\theta)}{\varepsilon}$$

where: $\Delta_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ i-th dimension



Quick Math Refresher
The Partial Derivative

- Assume we have a d-dimensional scalar function:

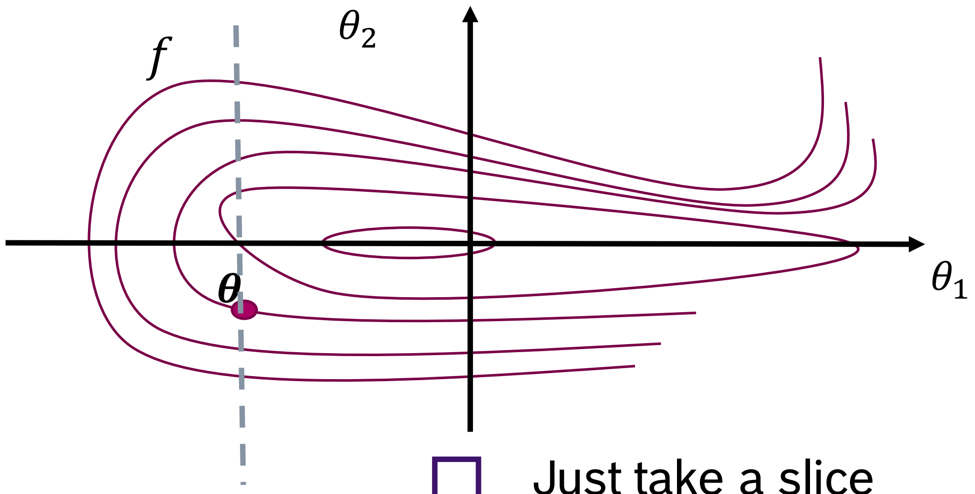
$$f : \mathbb{R}^d \to \mathbb{R}$$

- The partial derivative for dimension i describes how sensitive f is to infinitesimal changes along dimension i

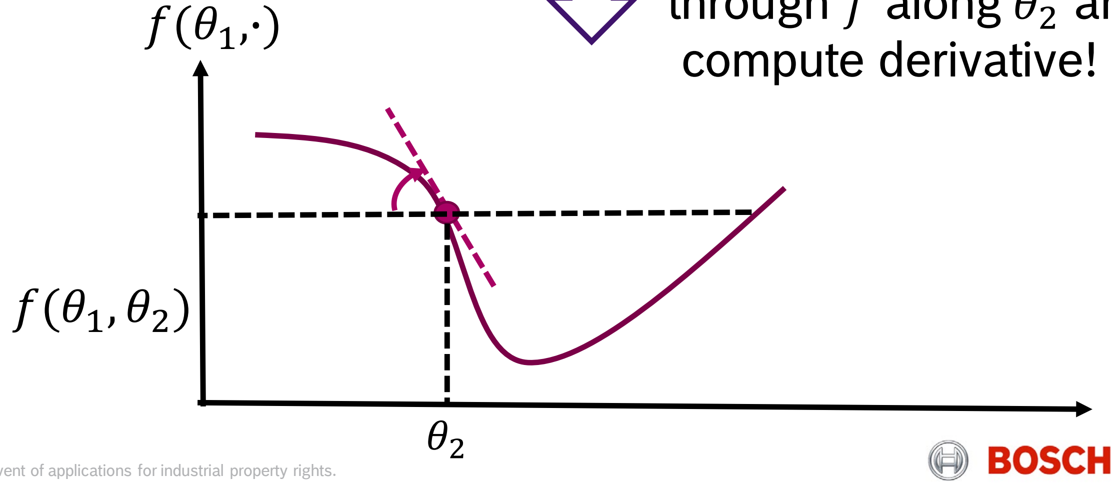
θ_2

$$\frac{\partial f}{\partial \theta_i}(\theta) = \lim_{\varepsilon \to 0} \frac{f(\theta + \varepsilon \Delta_i) - f(\theta)}{\varepsilon}$$

where: $\Delta_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ i-th dimension



Just take a slice through f along θ_2 and compute derivative!

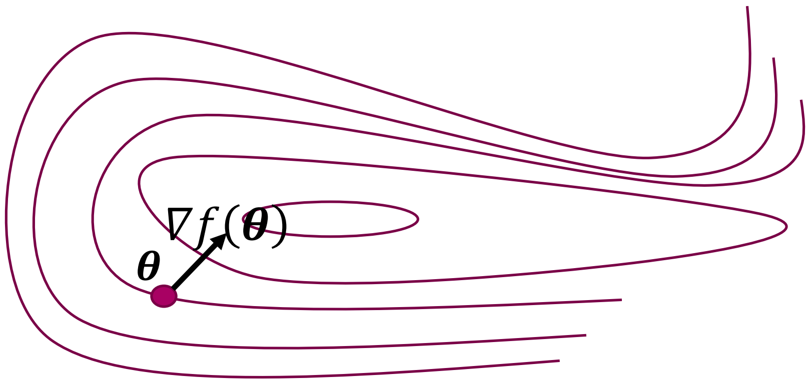


Quick Math Refresher
The Gradient

- Generalizes the gradient to a d-dimensional scalar function
- Just stack together the partial derivatives along each axis
- The gradient is a vector
- Interpretation: the direction of steepest ascent

$$f : \mathbb{R}^d \rightarrow \mathbb{R}$$

$$\nabla f(\theta) = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \\ \vdots \\ \frac{\partial f}{\partial \theta_d} \end{bmatrix}$$



OPTIMIZATION

Optimization
Problem Formulation

- Almost all machine learning problems are a form of (constrained / approximate) minimization of an **objective function** (*e.g.* training negative log likelihood)

$$\boxed{\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$$

$p_{\text{data}}(x, y)$

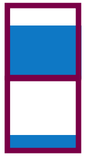
Optimization
Problem Formulation

- Almost all machine learning problems are a form of (constrained / approximate) minimization of an **objective function** (*e.g.* training negative log likelihood)

$$\boxed{\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$$

$p_{\text{data}}(x, y)$

$p_{\text{model}}(y|\boldsymbol{x};\boldsymbol{\theta}) =$



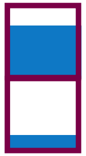
Optimization
Problem Formulation

- Almost all machine learning problems are a form of (constrained / approximate) minimization of an **objective function** (*e.g.* training negative log likelihood)

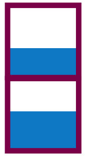
$$\boxed{\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$$

$p_{\text{data}}(x, y)$

$p_{\text{model}}(y|\boldsymbol{x};\boldsymbol{\theta}) =$



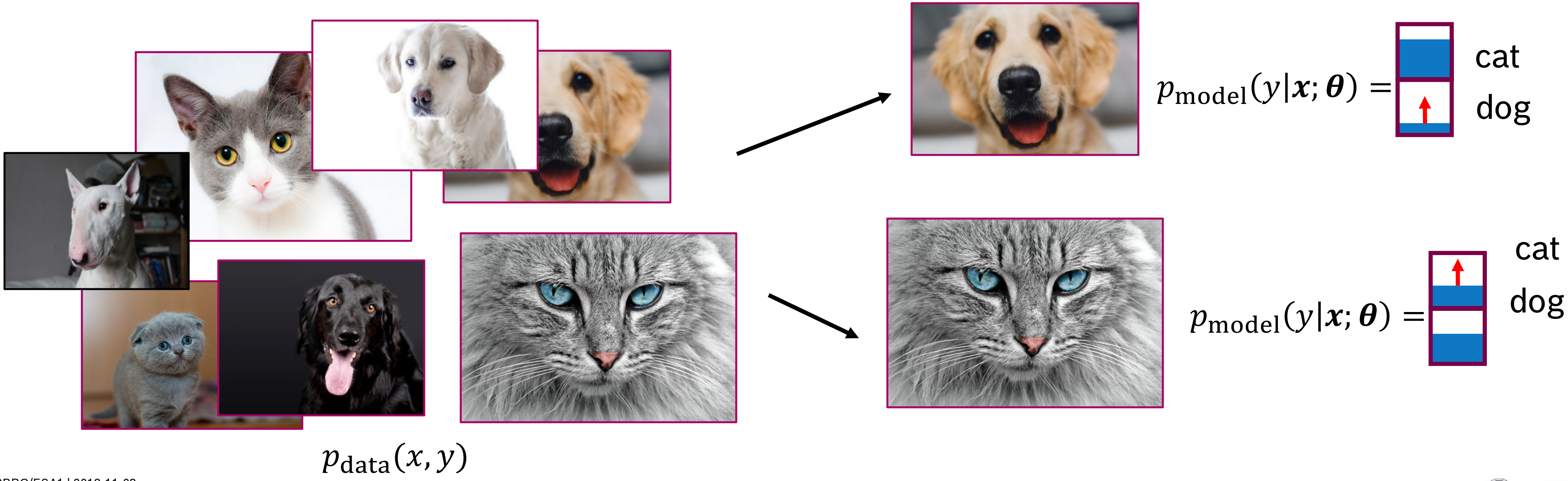
$p_{\text{model}}(y|\boldsymbol{x};\boldsymbol{\theta}) =$



Optimization
Problem Formulation

- Almost all machine learning problems are a form of (constrained / approximate) minimization of an **objective function** (*e.g.* training negative log likelihood)

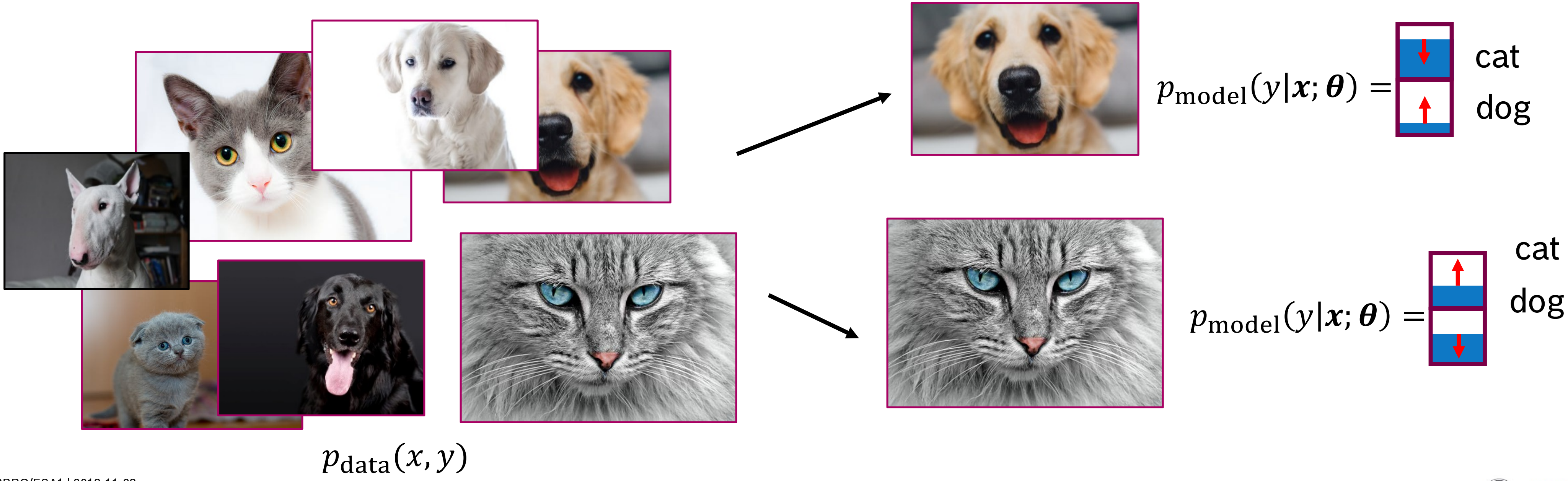
$$\boxed{\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$$



Optimization
Problem Formulation

- Almost all machine learning problems are a form of (constrained / approximate) minimization of an **objective function** (*e.g.* training negative log likelihood)

$$\boxed{\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$$



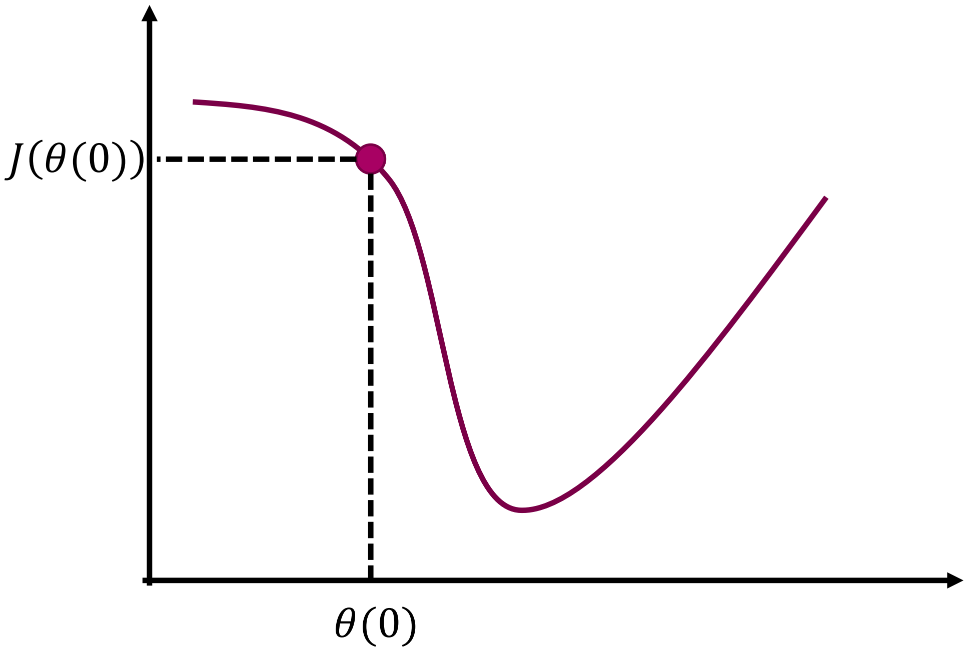
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
$\quad \theta(t+1) \leftarrow \theta(t) - \eta\, J'(\theta(t));$
$\quad t \leftarrow t+1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



RBRO/ESA1 | 2018-11-08

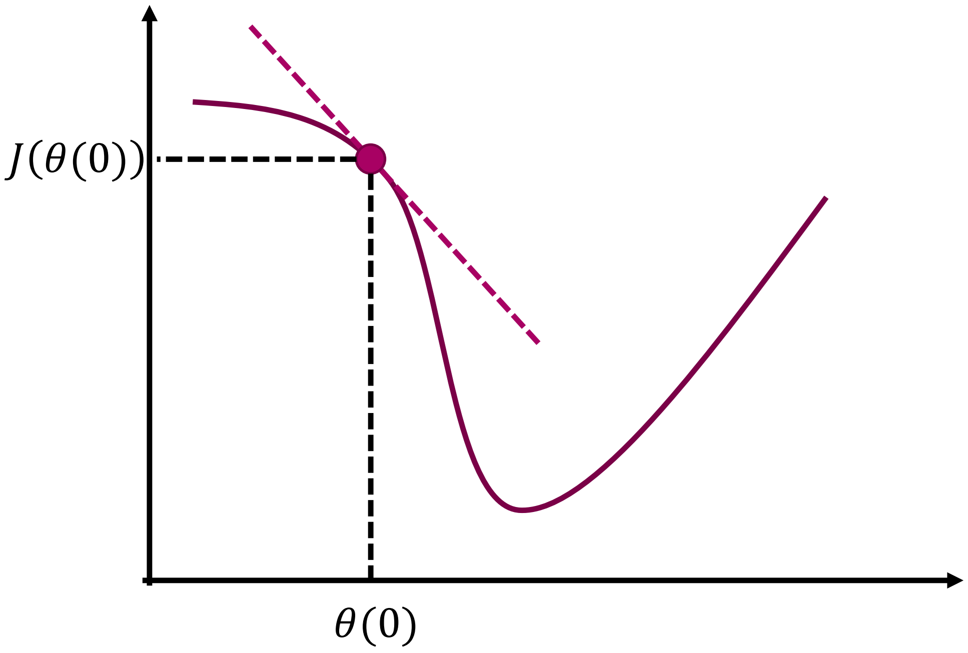
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
    $\theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
    $t \leftarrow t+1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



RBRO/ESA1 | 2018-11-08

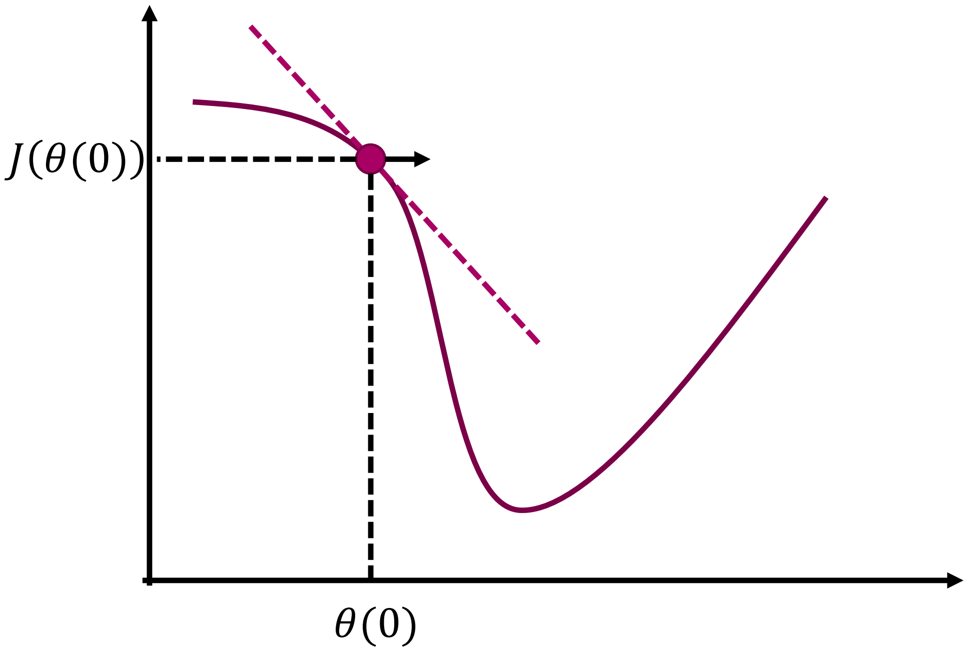
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
    $\theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
    $t \leftarrow t + 1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



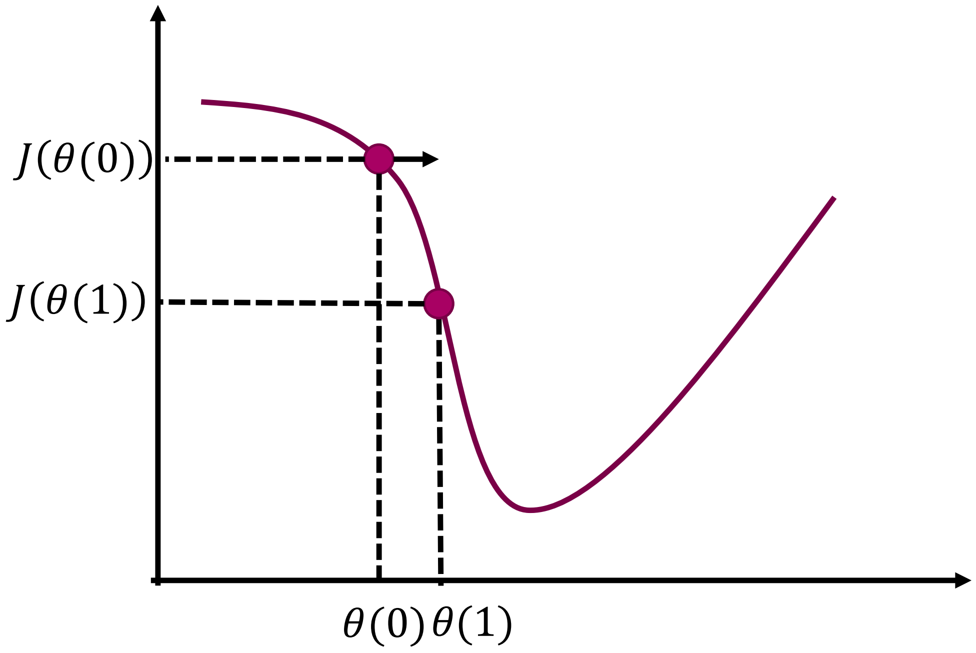
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
$\quad\theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
$\quad t \leftarrow t + 1;$
} **while** ($|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



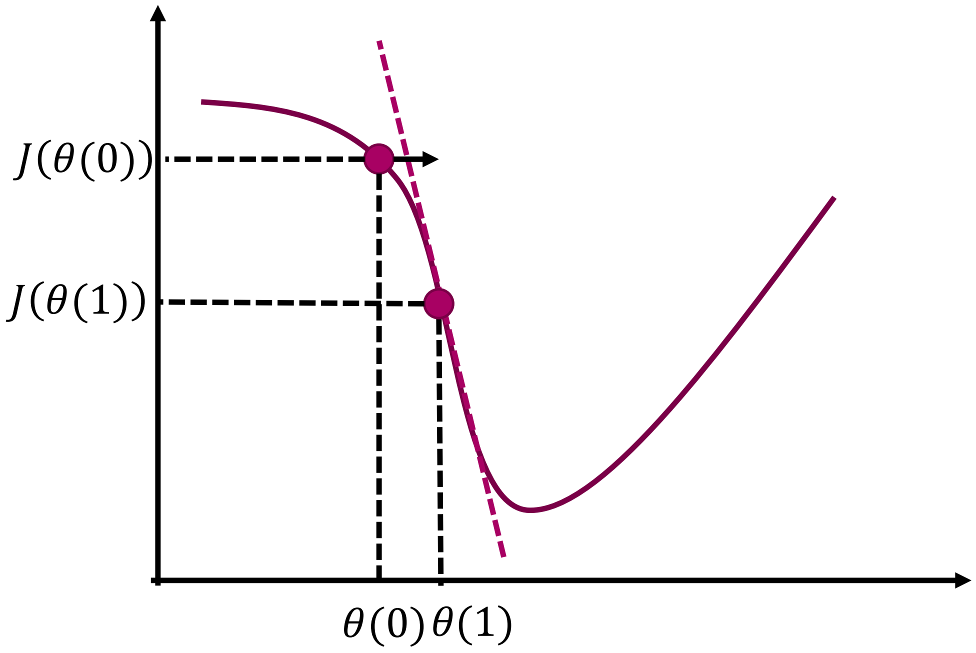
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
$\quad \theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
$\quad t \leftarrow t+1;$
} **while** ($|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



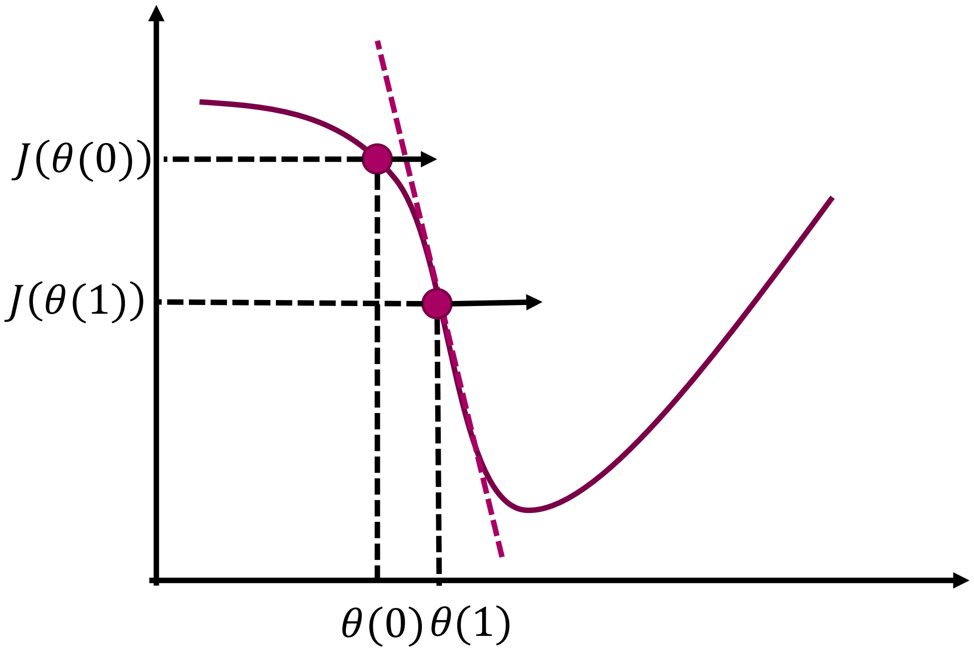
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
  $\theta(t+1) \leftarrow \theta(t) - \eta J'\big(\theta(t)\big);$
  $t \leftarrow t+1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



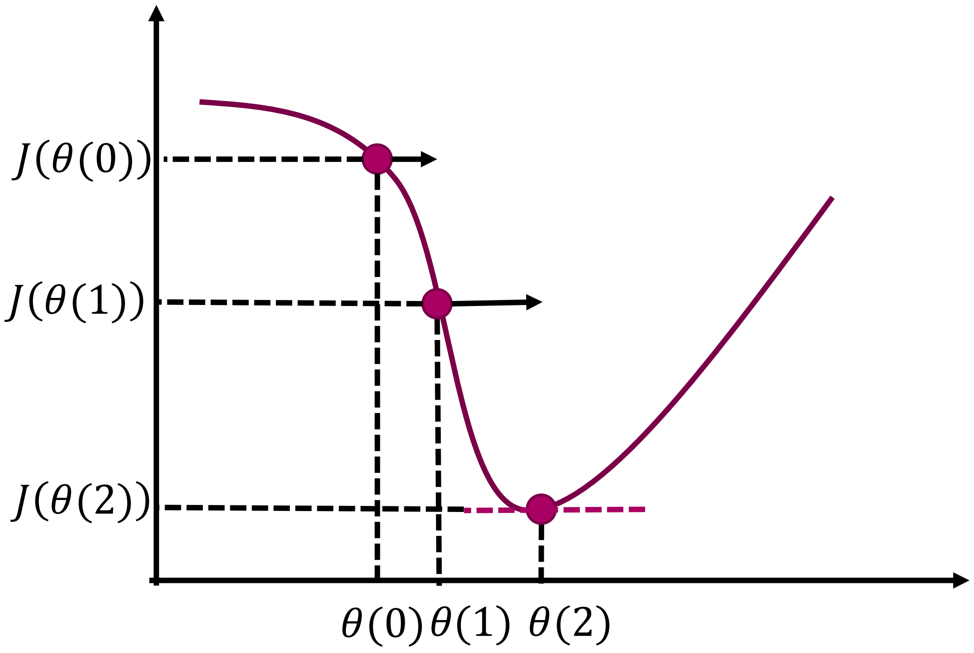
Optimization
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
$\quad\theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
$\quad t \leftarrow t + 1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



RBRO/ESA1 | 2018-11-08

BOSCH

Optimization

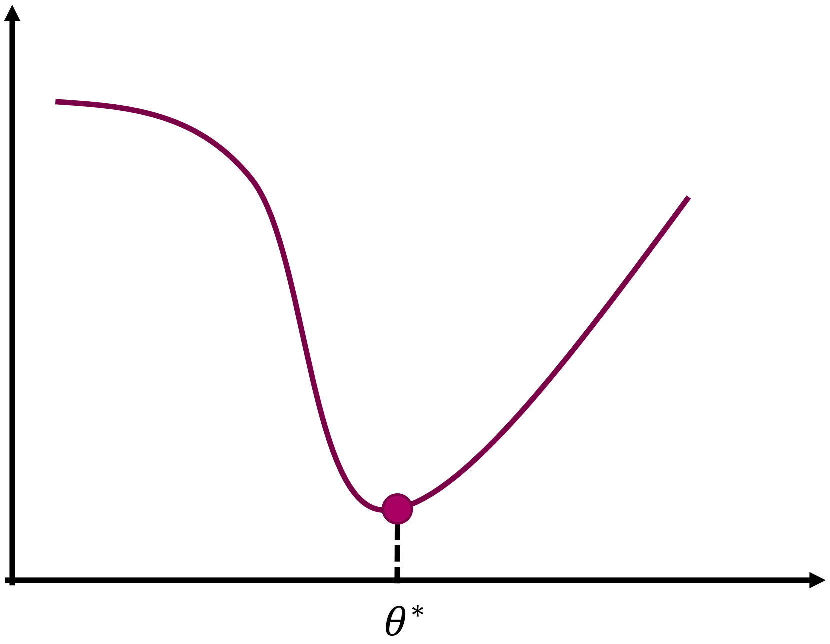
Gradient Descent Algorithm (1D Case)

function *gradient-descent-1d*
input: a function J
output: a local optimum θ^* of J

// Start from a random initial guess.
$\theta(0) \leftarrow rand(\mathbb{R});$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
    $\theta(t+1) \leftarrow \theta(t) - \eta\, J'\big(\theta(t)\big);$
    $t \leftarrow t + 1;$
} **while** $(|\theta(t) \leftarrow \theta(t-1)| > \varepsilon$**;**

return $\theta(t);$



Optimization
Gradient Descent Algorithm (n-D case)

- Use a 1st order Taylor expansion of J:

$$J(\boldsymbol{\theta}) \cong J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla J(\boldsymbol{\theta}_0)$$

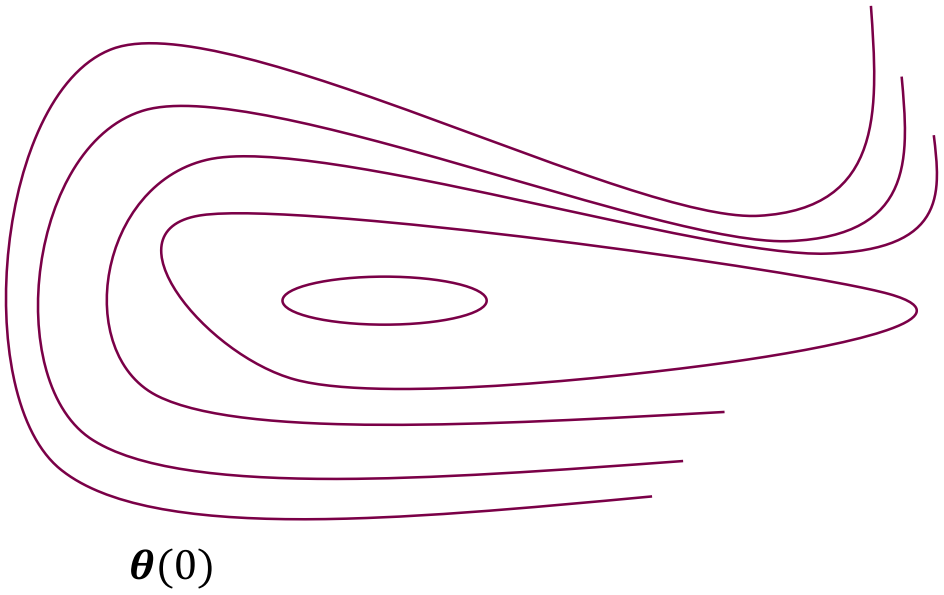
- Move in the direction with negative slope

function *gradient-descent*
input: a function J
output: a local optimum $\boldsymbol{\theta}^*$ of J

// Start from a random initial guess.
$\boldsymbol{\theta}(0) \leftarrow rand(\mathbb{R}^p)$;

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0$;
do {
  $\boldsymbol{\theta}(t+1) \leftarrow \boldsymbol{\theta}(t) - \eta\nabla J\big(\boldsymbol{\theta}(t)\big)$;
  $t \leftarrow t + 1$;
} **while** $(|\boldsymbol{\theta}(t) \leftarrow \boldsymbol{\theta}(t-1)| > \varepsilon$**;**

return $\boldsymbol{\theta}(t)$;



BOSCH

Optimization
Gradient Descent Algorithm (n-D case)

- Use a 1st order Taylor expansion of J:

$$J(\boldsymbol{\theta}) \cong J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla J(\boldsymbol{\theta}_0)$$

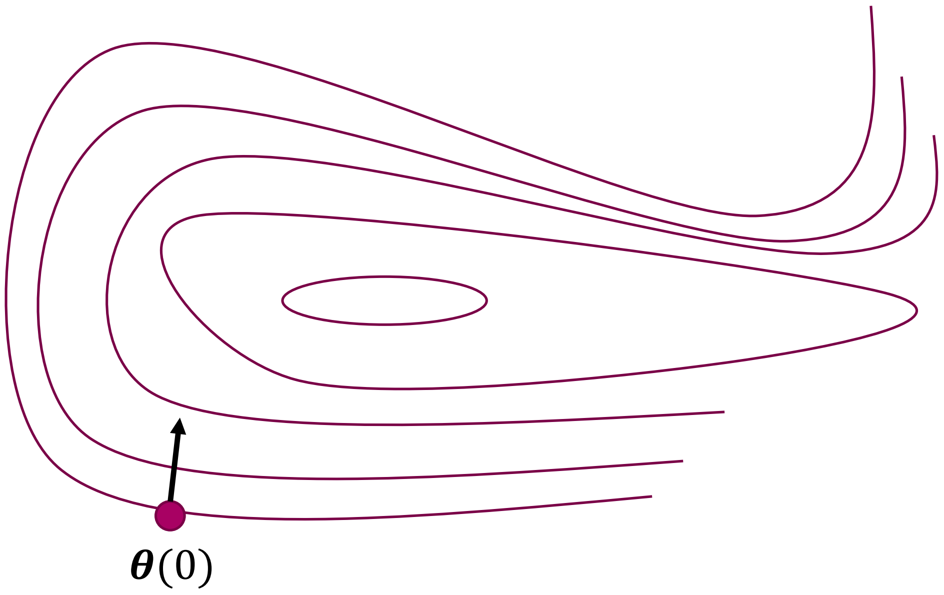
- Move in the direction with negative slope

function *gradient-descent*
input: a function J
output: a local optimum $\boldsymbol{\theta}^*$ of J

// Start from a random initial guess.
$\boldsymbol{\theta}(0) \leftarrow rand(\mathbb{R}^p);$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
    $\boldsymbol{\theta}(t+1) \leftarrow \boldsymbol{\theta}(t) - \eta \nabla J\big(\boldsymbol{\theta}(t)\big);$
    $t \leftarrow t+1;$
} **while** ($|\boldsymbol{\theta}(t) \leftarrow \boldsymbol{\theta}(t-1)| > \varepsilon$**;**

return $\boldsymbol{\theta}(t);$



Optimization
Gradient Descent Algorithm (n-D case)

- Use a 1st order Taylor expansion of J:

$$J(\boldsymbol{\theta}) \cong J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \boldsymbol{\nabla} J(\boldsymbol{\theta}_0)$$

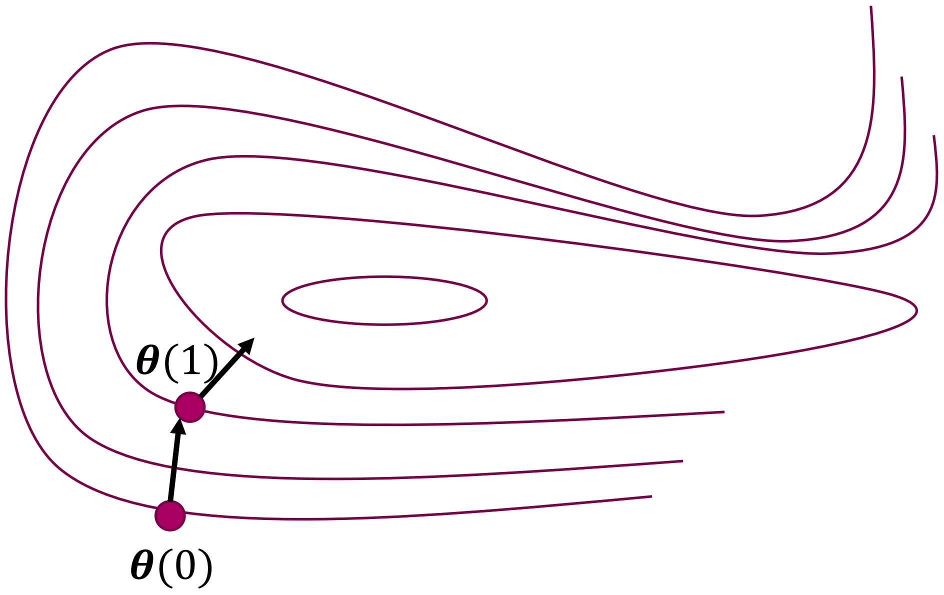
- Move in the direction with negative slope

function *gradient-descent*
input: a function J
output: a local optimum $\boldsymbol{\theta}^*$ of J

// Start from a random initial guess.
$\boldsymbol{\theta}(0) \leftarrow rand(\mathbb{R}^p)$;

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0$;
do {
  $\boldsymbol{\theta}(t+1) \leftarrow \boldsymbol{\theta}(t) - \eta \boldsymbol{\nabla} J(\boldsymbol{\theta}(t))$;
  $t \leftarrow t + 1$;
} **while** $(|\boldsymbol{\theta}(t) \leftarrow \boldsymbol{\theta}(t-1)| > \varepsilon$**;**

return $\boldsymbol{\theta}(t)$;



Optimization
Gradient Descent Algorithm (n-D case)

- Use a 1st order Taylor expansion of J:

$$J(\boldsymbol{\theta}) \cong J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla J(\boldsymbol{\theta}_0)$$

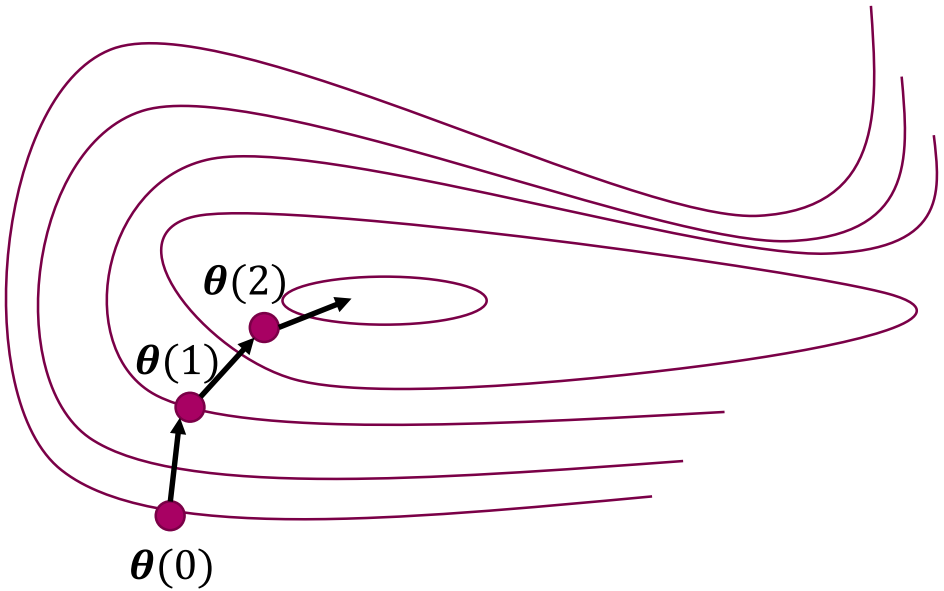
- Move in the direction with negative slope

function *gradient-descent*
input: a function J
output: a local optimum $\boldsymbol{\theta}^*$ of J

// Start from a random initial guess.
$\boldsymbol{\theta}(0) \leftarrow rand(\mathbb{R}^p)$;

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0$;
do {
    $\boldsymbol{\theta}(t+1) \leftarrow \boldsymbol{\theta}(t) - \eta \nabla J\big(\boldsymbol{\theta}(t)\big)$;
    $t \leftarrow t + 1$;
} **while** $(|\boldsymbol{\theta}(t) \leftarrow \boldsymbol{\theta}(t-1)| > \varepsilon$**;**

return $\boldsymbol{\theta}(t)$;



Optimization
Gradient Descent Algorithm (n-D case)

- Use a 1st order Taylor expansion of J:

$$J(\boldsymbol{\theta}) \cong J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top \nabla J(\boldsymbol{\theta}_0)$$

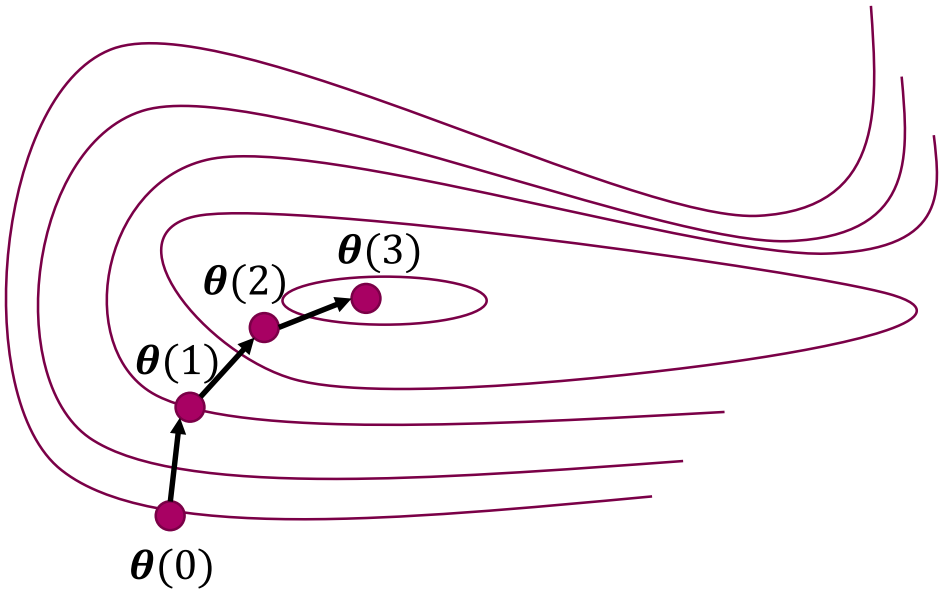
- Move in the direction with negative slope

function *gradient-descent*
input: a function J
output: a local optimum $\boldsymbol{\theta}^*$ of J

// Start from a random initial guess.
$\boldsymbol{\theta}(0) \leftarrow rand(\mathbb{R}^p);$

// Take steps in the direction of negative slope until
// convergence.
$t \leftarrow 0;$
do {
$\quad \boldsymbol{\theta}(t+1) \leftarrow \boldsymbol{\theta}(t) - \eta \nabla J\big(\boldsymbol{\theta}(t)\big);$
$\quad t \leftarrow t+1;$
} **while** $(|\boldsymbol{\theta}(t) \leftarrow \boldsymbol{\theta}(t-1)| > \varepsilon$**;**

return $\boldsymbol{\theta}(t);$

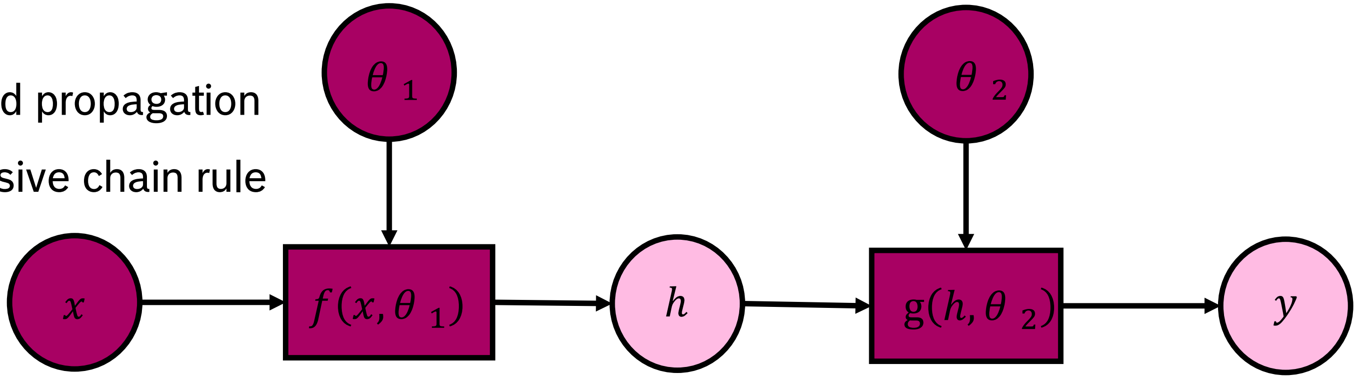


Optimization
Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition

- **Output:** gradient with respect to either:
  - parameter – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis

- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

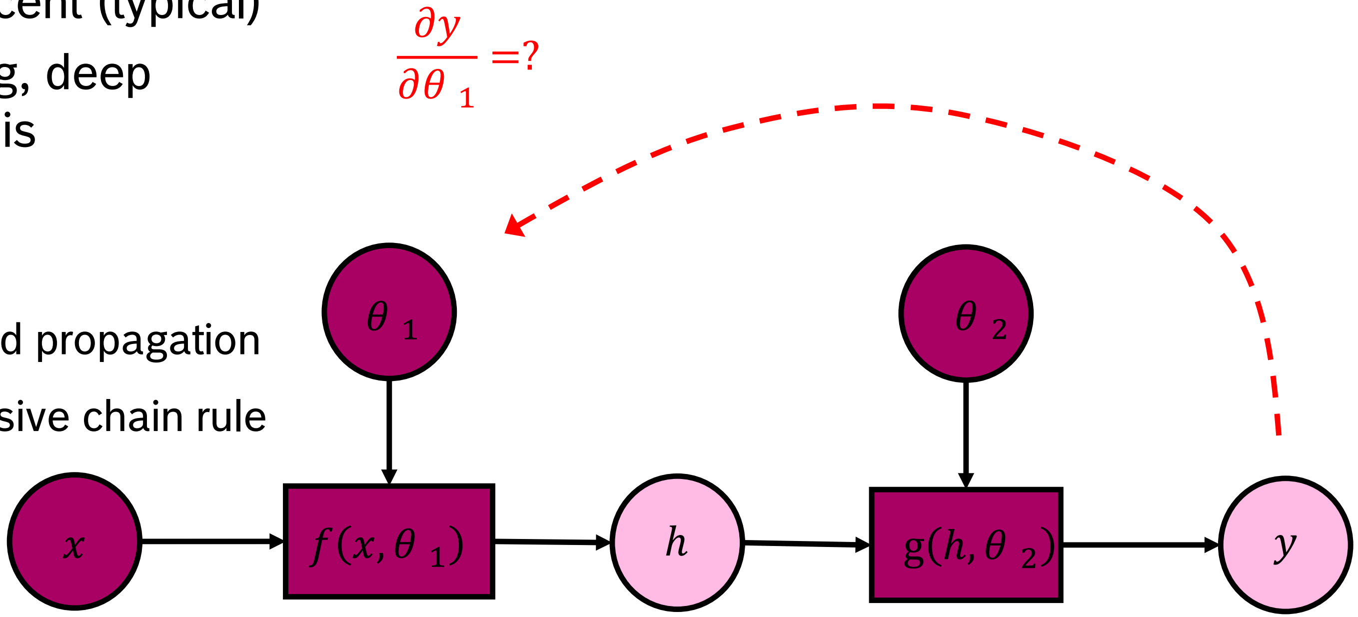


RBRO/ESA1 | 2018-11-08

BOSCH

Optimization
Computing the Gradient: The Backpropagation Algorithm

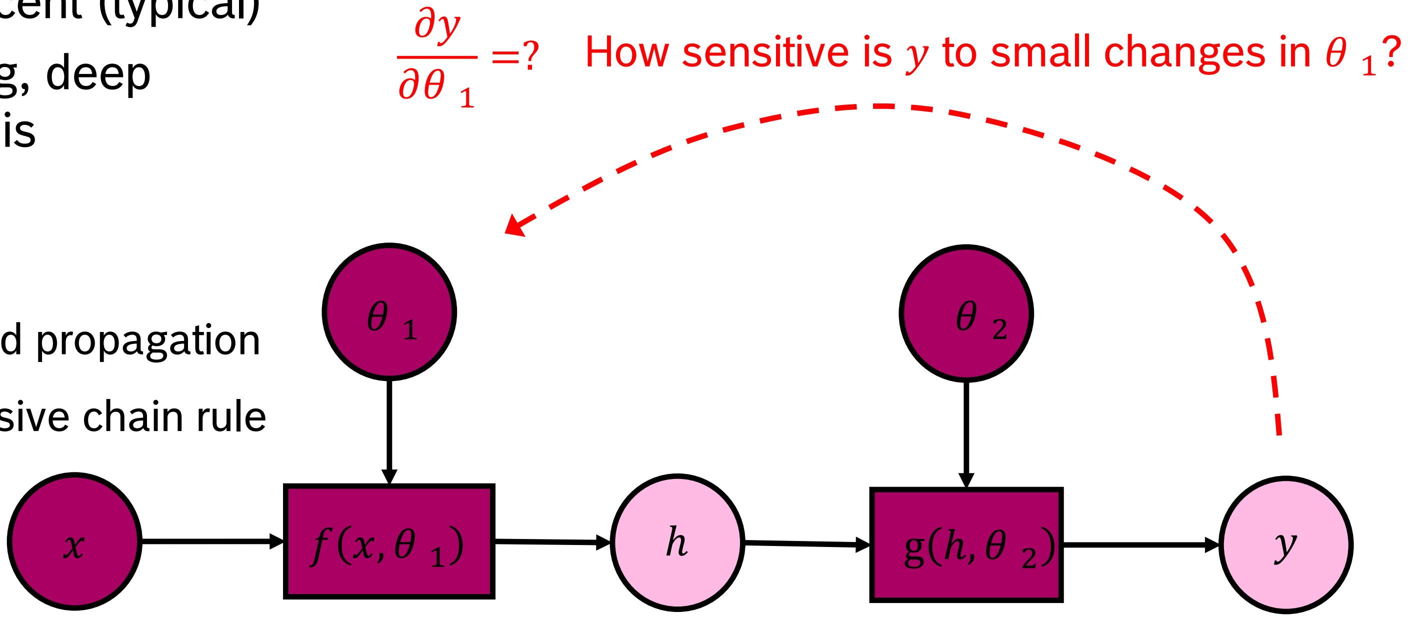
- **Input:** function composition
- **Output:** gradient with respect to either:
  - parameter – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis
- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications



RBRO/ESA1 | 2018-11-08

Optimization
Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition
- **Output:** gradient with respect to either:
  - parameter – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis
- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

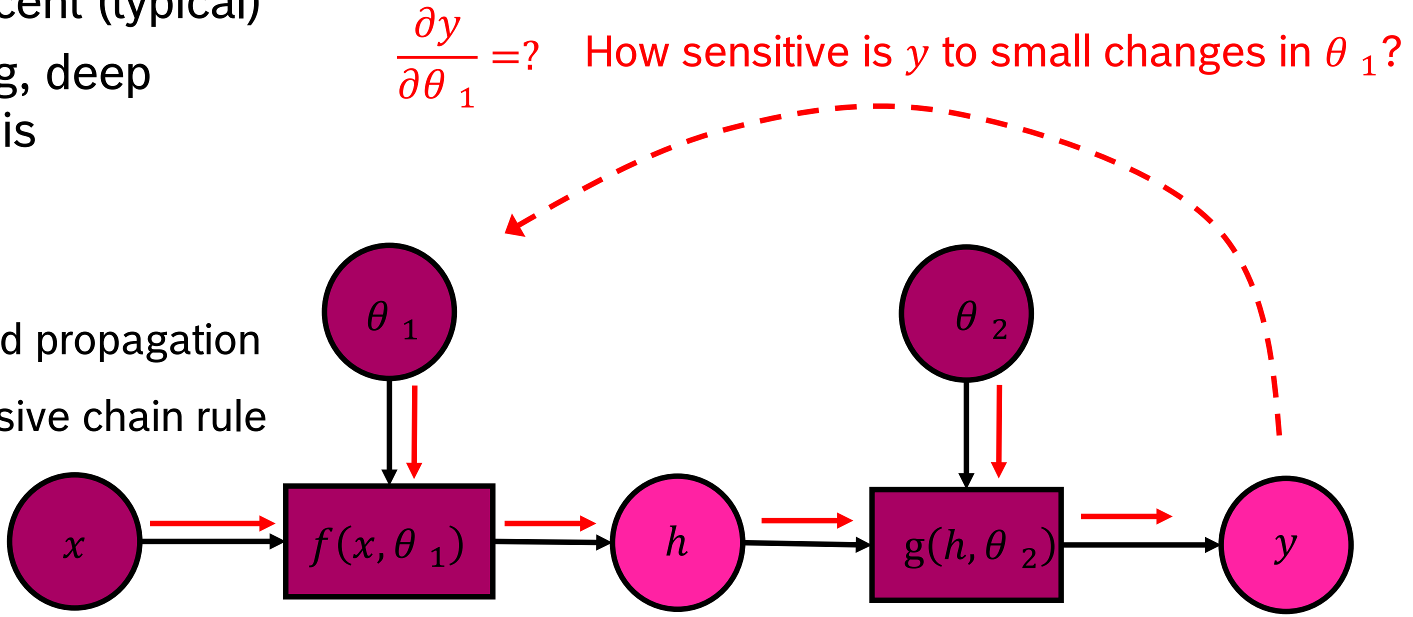


RBRO/ESA1 | 2018-11-08

Optimization

Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition
- **Output:** gradient with respect to either:
  - parameter – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis
- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications



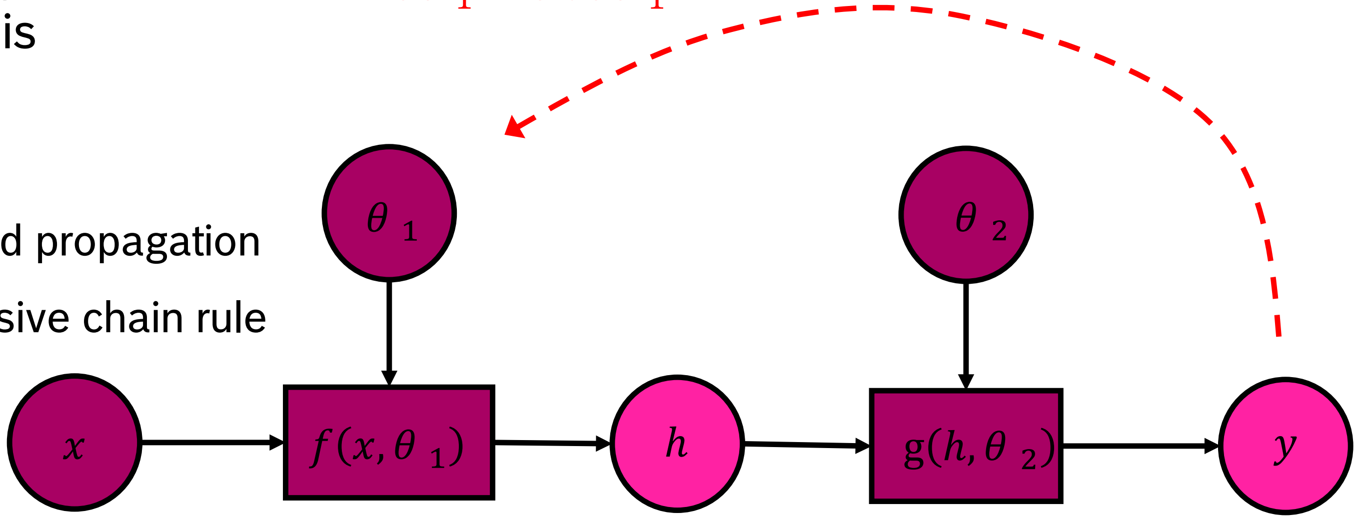
Optimization
Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition

- **Output:** gradient with respect to either:
  - <u>parameter</u> – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis

- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

$$\frac{\partial y}{\partial \theta_1} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial \theta_1}$$

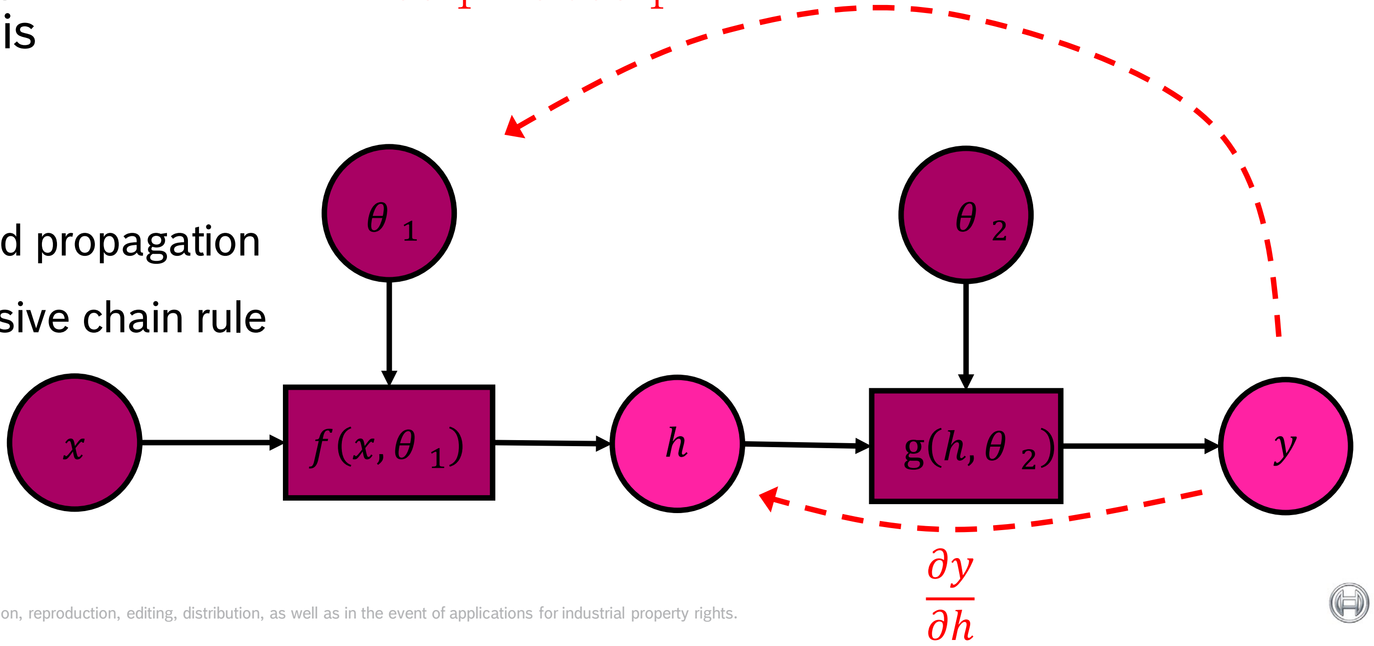


Optimization
Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition
- **Output:** gradient with respect to either:
  - <u>parameter</u> – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis
- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

How sensitive is y to small changes in h?

$$\frac{\partial y}{\partial \theta_1} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial \theta_1}$$



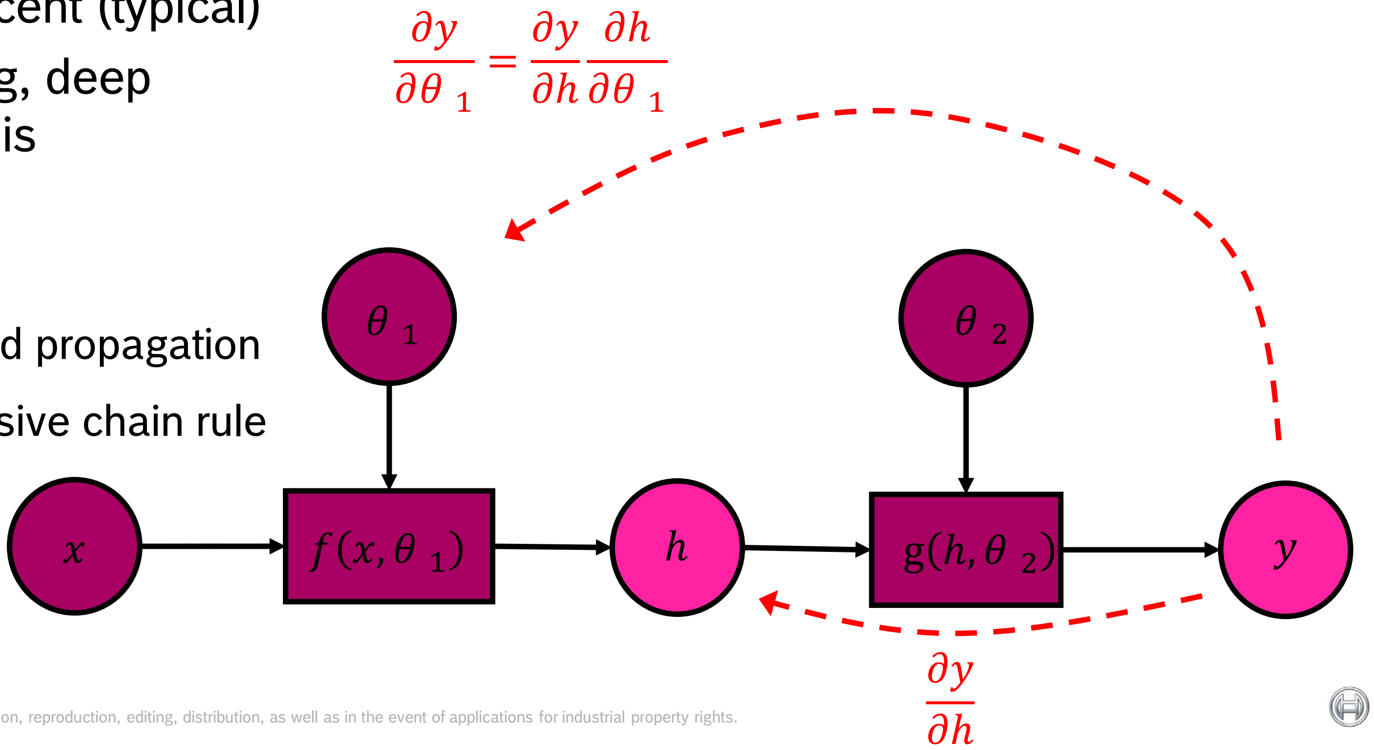
BOSCH

Optimization
Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition

- **Output:** gradient with respect to either:
  - parameter – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis

- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

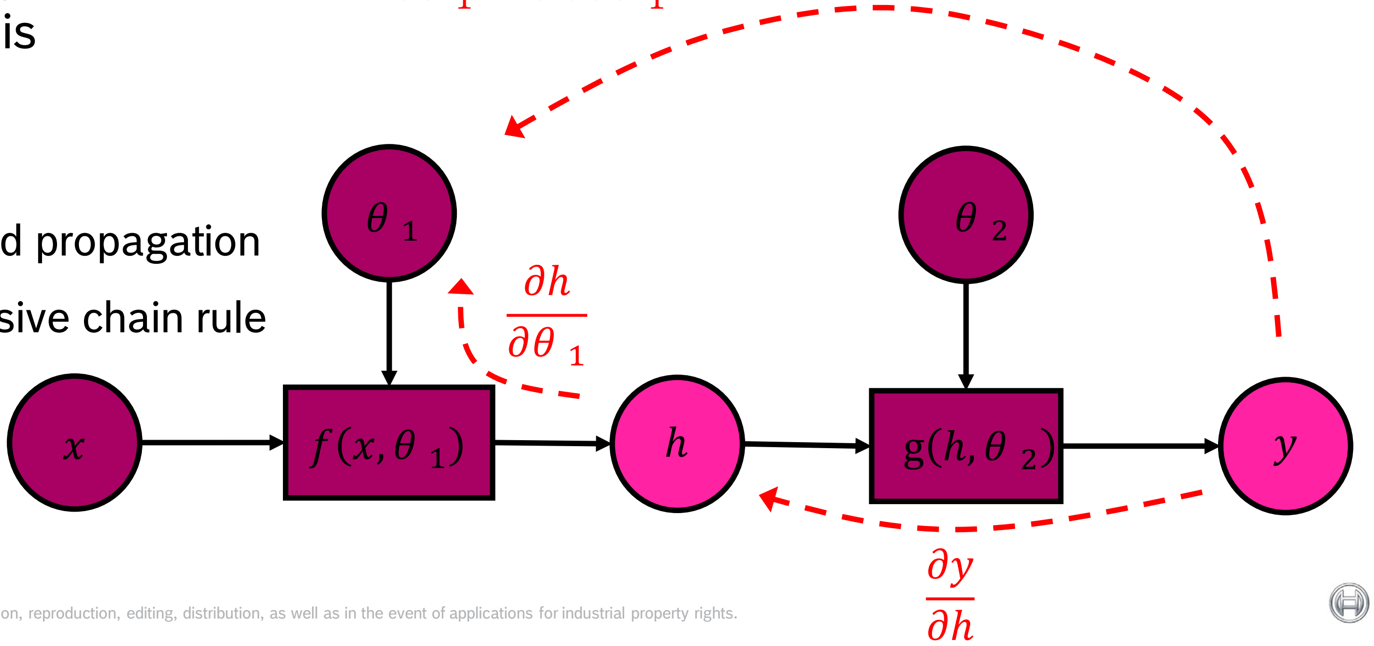


Optimization

Computing the Gradient: The Backpropagation Algorithm

- **Input:** function composition
- **Output:** gradient with respect to either:
  - <u>parameter</u> – gradient descent (typical)
  - input – adversarial training, deep dreaming, network analysis
- **Steps:**
  - Step 1: Compute y by forward propagation
  - Step 2: Compute $\frac{\partial y}{\partial \theta}$ by recursive chain rule applications

$$\frac{\partial y}{\partial \theta_1} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial \theta_1}$$



Optimization
Backpropagation: Extensions

- **Variants:**
  - **symbol-to-number**: nodes compute gradient numerically (Torch, Caffe, Pytorch)
  - **symbol-to-symbol**: augment the graph with nodes that compute the gradient (Theano, Tensorflow) – can simplify symbolic expressions!

- **Efficiency:**
  - Not optimal (special case of the **reverse mode differentiation algorithm**)
  - Finding the optimal formula is NP-complete *[Naumann, 2008]*

- **Implementation:**
  - **automatic differentiation** (transparent gradient computation)
  - new way of writing programs:

“Deep Learning est mort. Vive Differentiable Programming!” [Yann LeCun]

BOSCH

Optimization
Stochastic Gradient Descent

- **Problem:**
  - Empirical risk is too expensive (sums over **all** training samples):

$$\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} \frac{1}{m} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \log p_{\text{model}}(y_i | \boldsymbol{x}_i; \boldsymbol{\theta})$$

Optimization
Stochastic Gradient Descent

- **Problem:**
  - Empirical risk is too expensive (sums over **all** training samples):

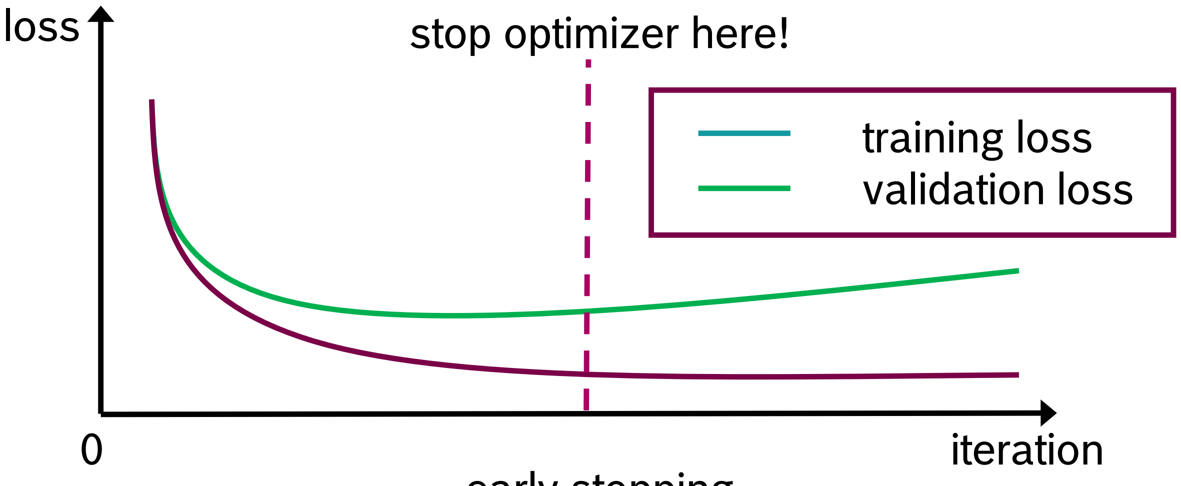
$$\boldsymbol{\theta}^* = \operatorname{argmax}_{\boldsymbol{\theta}} \frac{1}{m} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{n}} \log p_{\text{model}} (y_i | \boldsymbol{x}_i; \boldsymbol{\theta})$$

- **Solution:**
  - Compute gradient on a random set of samples, i.e. a **minibatch**
  - Use this as the approximation of the gradient for the full batch

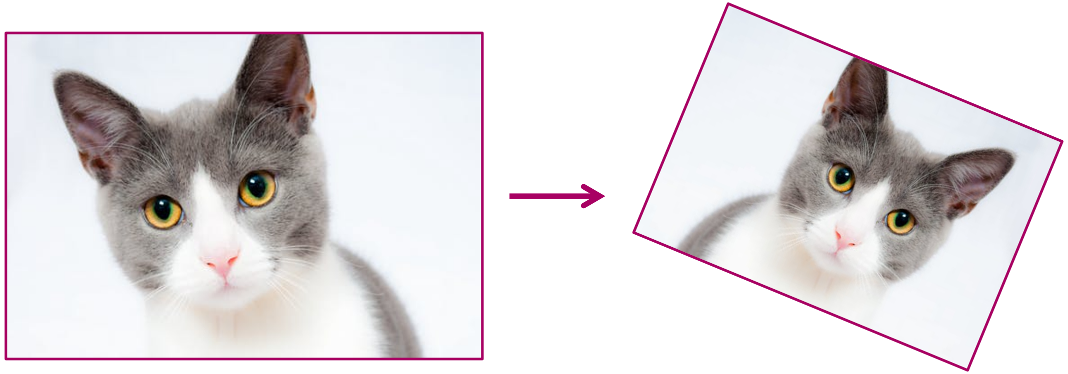
- **Minibatch size:**
  - Increases gradient accuracy (but less than linearly!)
  - Increases parallelism utilization
  - Increases memory utilization (assuming full parallelism)
  - Decreases regularization effect

Optimization
Regularization In Deep Learning (Revisited)

- **Optimizer:**
  - early stopping:
  - minibatch size
  - constrained optimization
- **Data augmentation:**
  - explicit input transformations
  - adversarial training

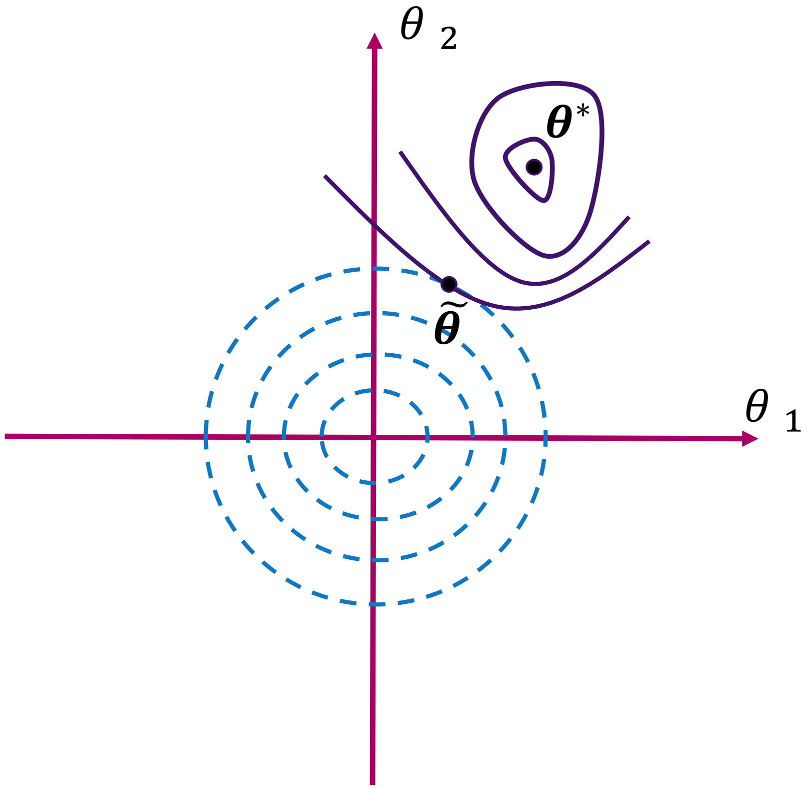


early stopping


explicit input transformation (mirroring, rotation, scaling)

Optimization
Regularization In Deep Learning (Revisited)

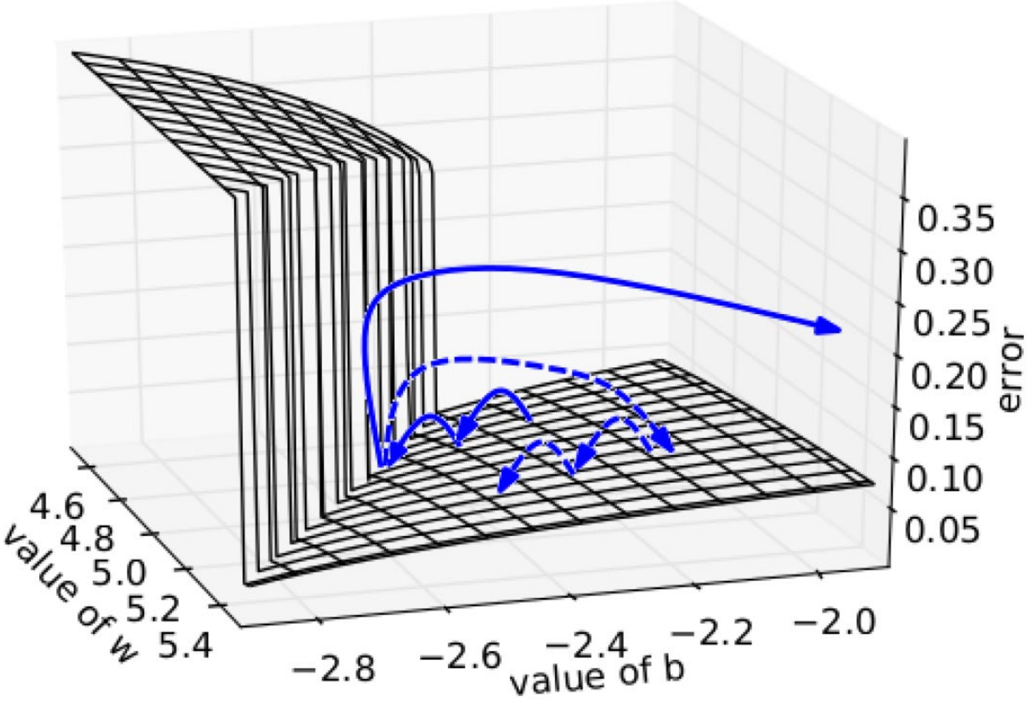
- **Model structure:**
  - sharing parameters
  - trimming parameters
  - multi-task learning
  - dropout
- **Objective function:**
  - weight decay
  - label smoothing



weight decay
(adapted from [1])

Optimization
Meta algorithms: Batch Normalization

- **Problem:**
  - sensitivity to one parameters often depends on others parameters
  - highly nonlinear behavior!
  - dangerous to update parameters at once
- **Solution:**
  - re-parameterize the network
  - add batch normalization layer:
    - computes mean and variance of each feature over the mini-batch
    - for each feature: subtracts mean and divides by variance
  - layer disappears at runtime (collapsed with the conv/fully connected layer)



(adapted from [1])

CONVOLUTIONAL **N**EURAL **N**ETWORKS

Convolutional Neural Networks
Deep Learning Recap

- **Machine Learning:**
  - Search for a program that maps features to correct answers
  - Feature design is hard!

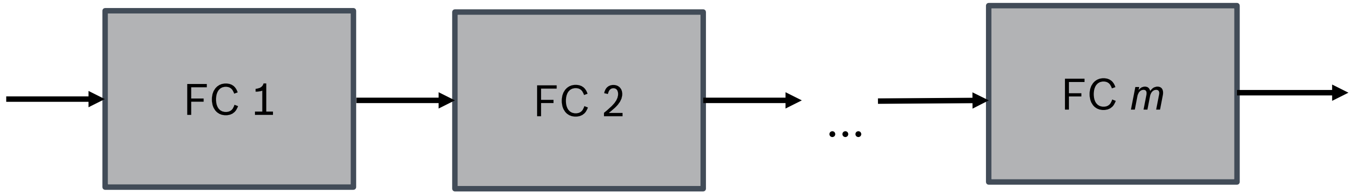
- **End-to-end Learning:**
  - Features are real-world observations
  - Learn several intermediate representation layers

- **Deep Learning:**
  - End-to-end learning with a lot of layers
  - How do we avoid the parameter space explosion? (overfitting)

Convolutional Neural Networks
A First Try: The Multi-Layer Perceptron

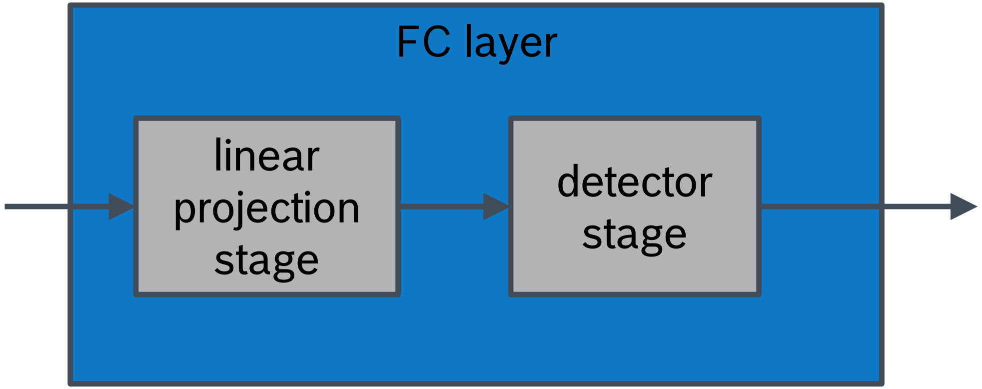
- **Idea:**
  - Level 1: match the input image to a set of patterns
  - Level 2: match the level 1 features to a set of patterns
  - …

- **Implementation:**
  - Fully Connected (FC) Layers



Convolutional Neural Networks
The Stages of an FC Layer

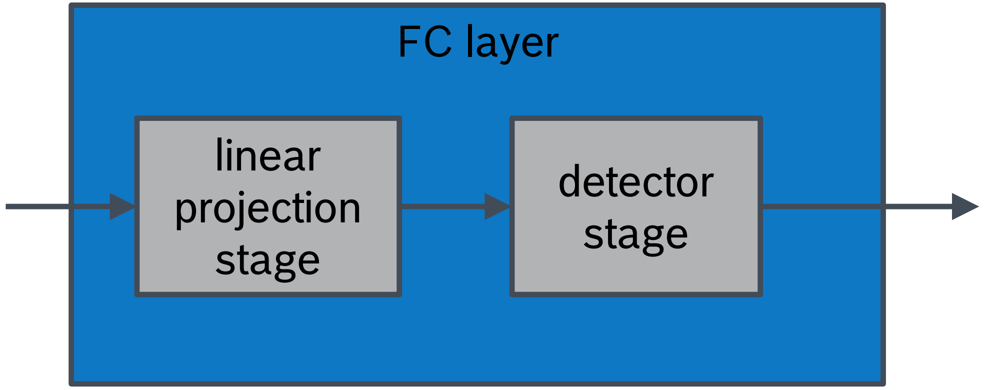
- **Stages:**
  - **Linear projection:** project feature vector $\mathbf{x}$ along a learned direction $\mathbf{w}$
  - **Detector:** is the projection y “strong enough” for a pattern match? (match strength z)

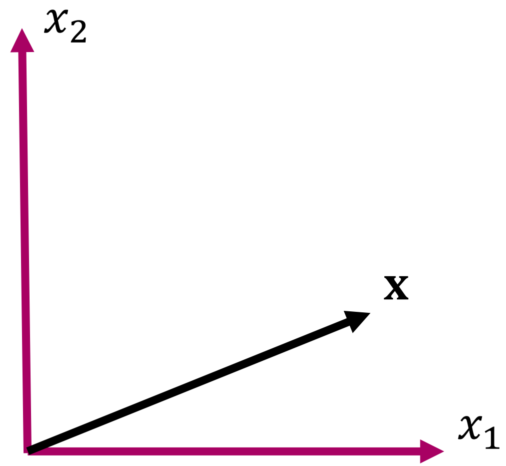


RBRO/ESA2 | 2018-11-08

Convolutional Neural Networks
The Stages of an FC Layer

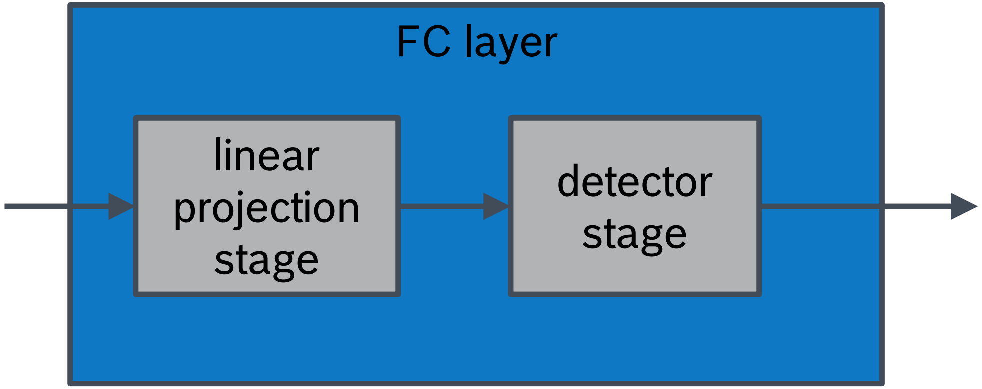
- **Stages:**
  - **Linear projection:** project feature vector $\mathbf{x}$ along a learned direction $\mathbf{w}$
  - **Detector:** is the projection y "strong enough" for a pattern match? (match strength z)

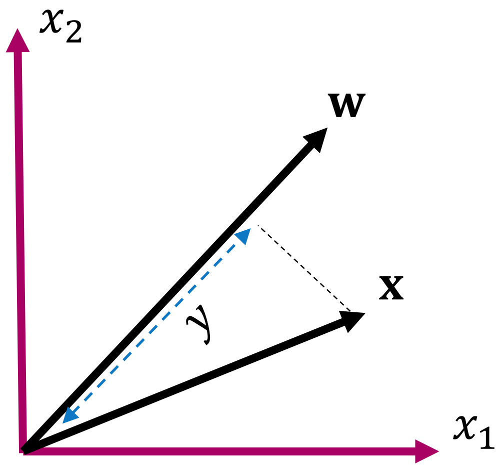




Convolutional Neural Networks
The Stages of an FC Layer

- **Stages:**
  - **Linear projection:** project feature vector $\mathbf{x}$ along a learned direction $\mathbf{w}$
  - **Detector:** is the projection y "strong enough" for a pattern match? (match strength z)

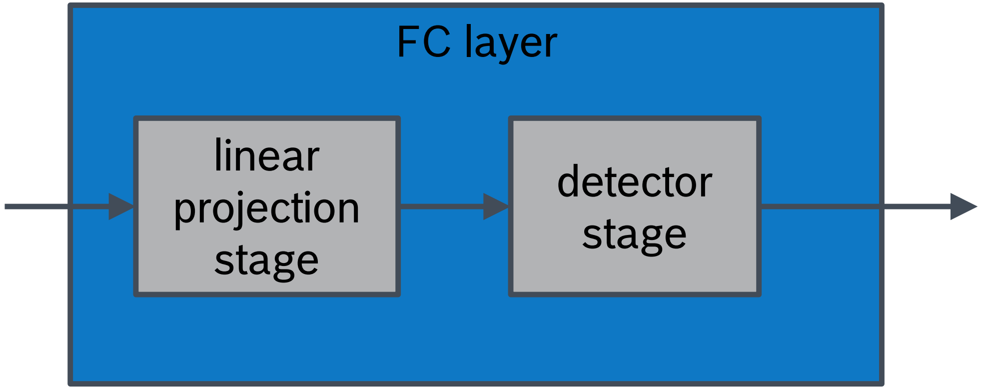


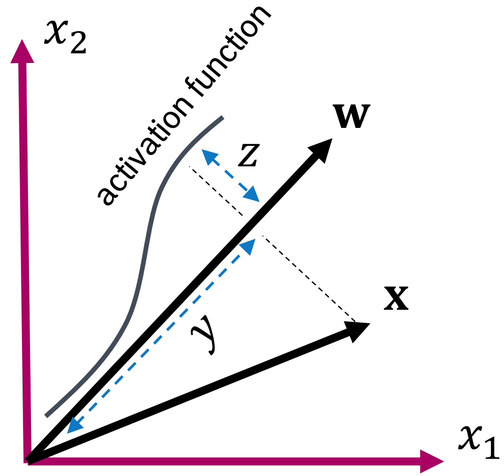


BOSCH

Convolutional Neural Networks
The Stages of an FC Layer

- **Stages:**
  - **Linear projection:** project feature vector $\mathbf{x}$ along a learned direction $\mathbf{w}$
  - **Detector:** is the projection y “strong enough” for a pattern match? (match strength z)

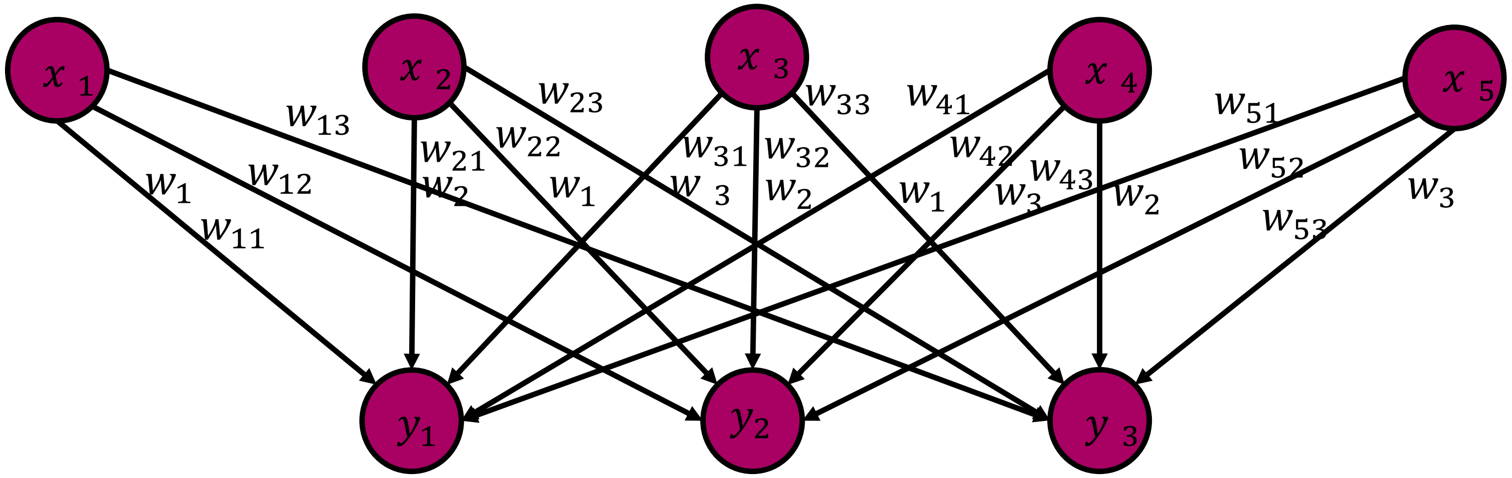


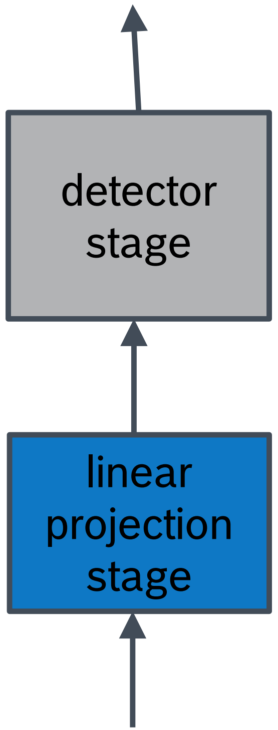


Convolutional Neural Networks
FC Layer: The Linear Projection Stage

- **Input:**
  - plain feature vector
  - do not exploit spatial/temporal structure

- **Output:**
  - each element a different linear combination of the input elements: $y_j = \sum_{i=1}^{N_{\text{in}}} w_{ij} \cdot x_i$

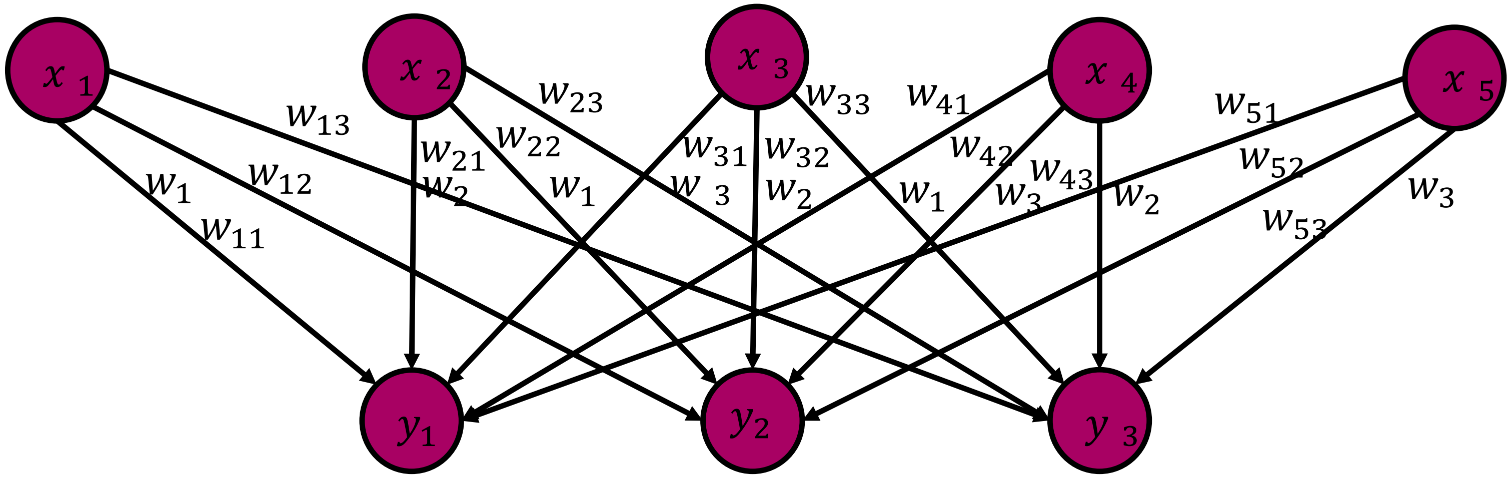


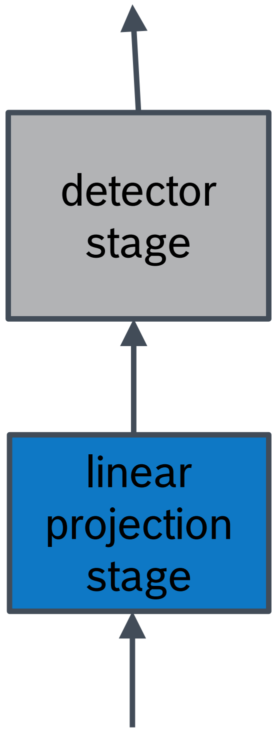


Convolutional Neural Networks
FC Layer: Complexity

- **Parameters:**
  - $N_{\text{in}} \cdot N_{\text{out}}$
  - *Example: 1 million parameters for a 256x256 input image!*

- **Time:**
  - Multiply-and-accumulates (MACs): $N_{\text{in}} \cdot N_{\text{out}}$



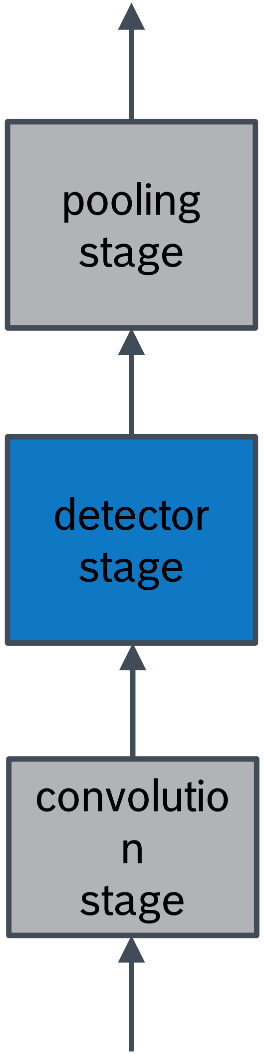


BOSCH

Convolutional Neural Networks
The Detector Stage

- **Requirements:**
  - Highly nonlinear behavior (soft thresholding)
  - Balance modeling power, efficiency, ease of optimization

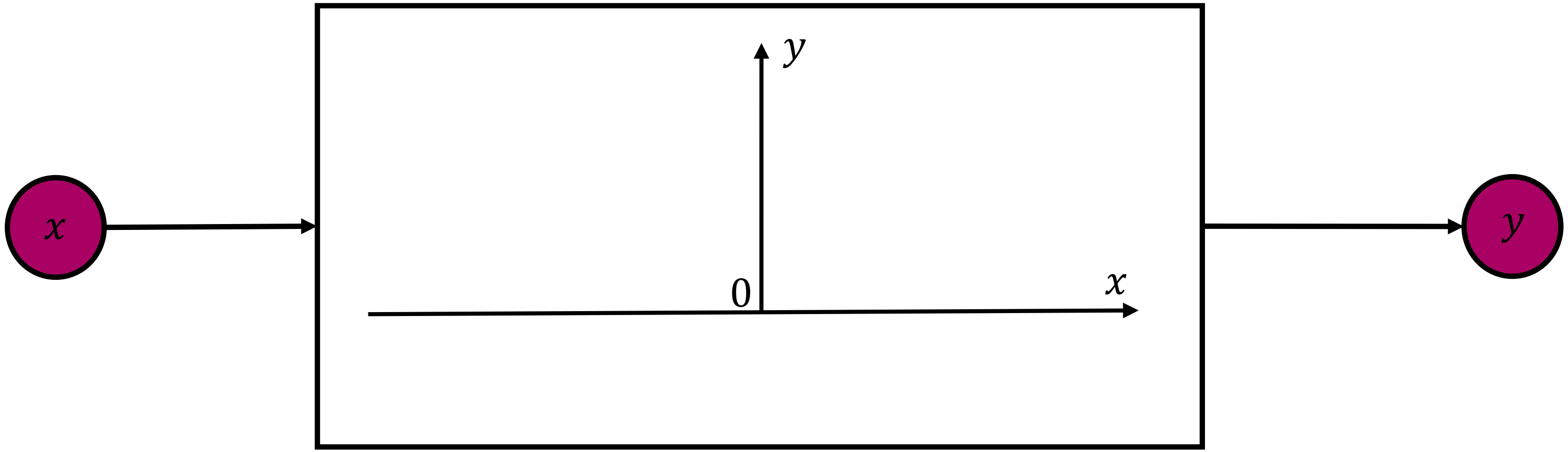
- **State-of-the-art:**
  - Most obvious choices do not work well! (e.g. softplus)
  - Open research field

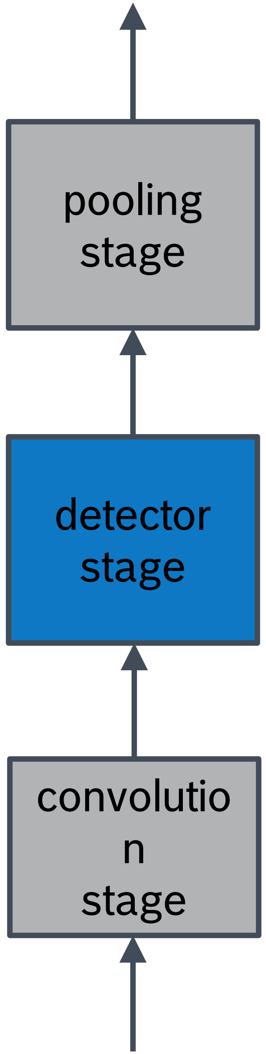


Convolutional Neural Networks
The Detector Stage

- **Requirements:**
  - Highly nonlinear behavior (soft thresholding)
  - Balance modeling power, efficiency, ease of optimization

- **State-of-the-art:**
  - Most obvious choices do not work well! (e.g. softplus)
  - Open research field



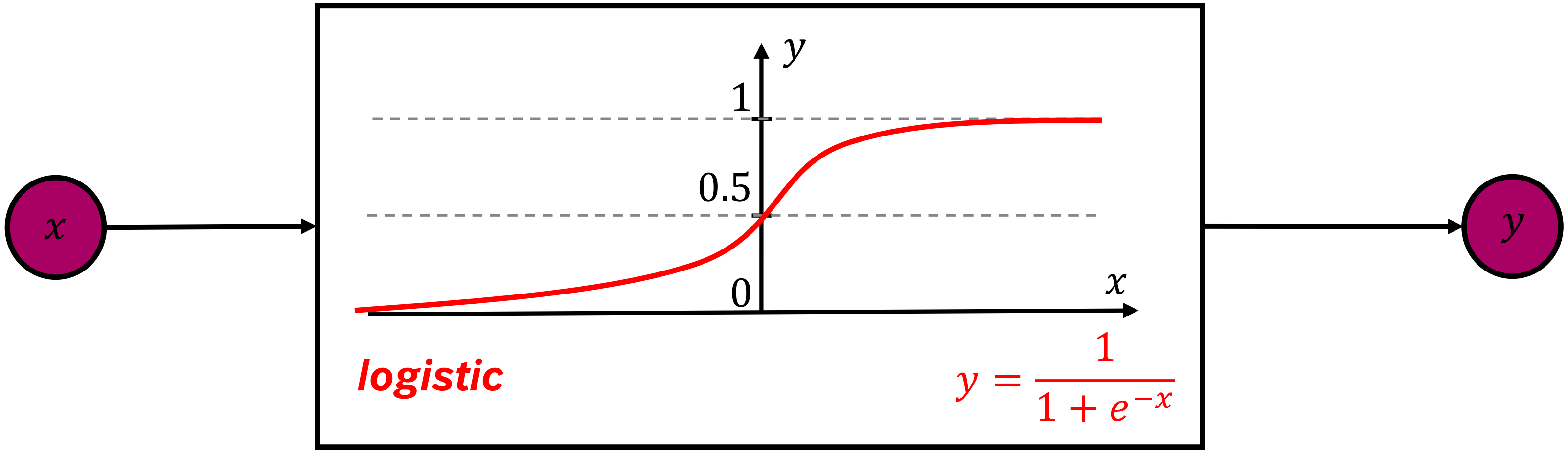


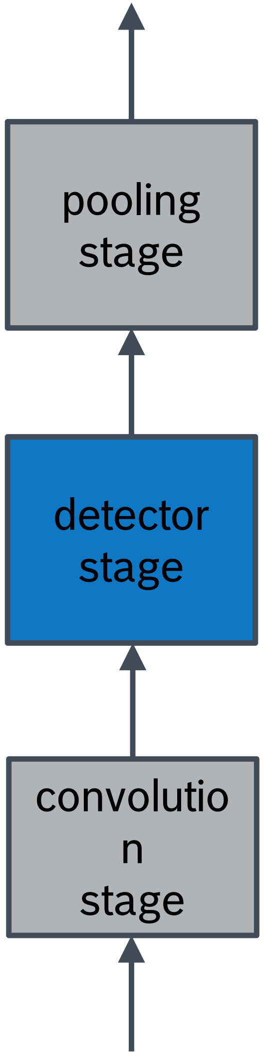
BOSCH

Convolutional Neural Networks
The Detector Stage

- **Requirements:**
  - Highly nonlinear behavior (soft thresholding)
  - Balance modeling power, efficiency, ease of optimization

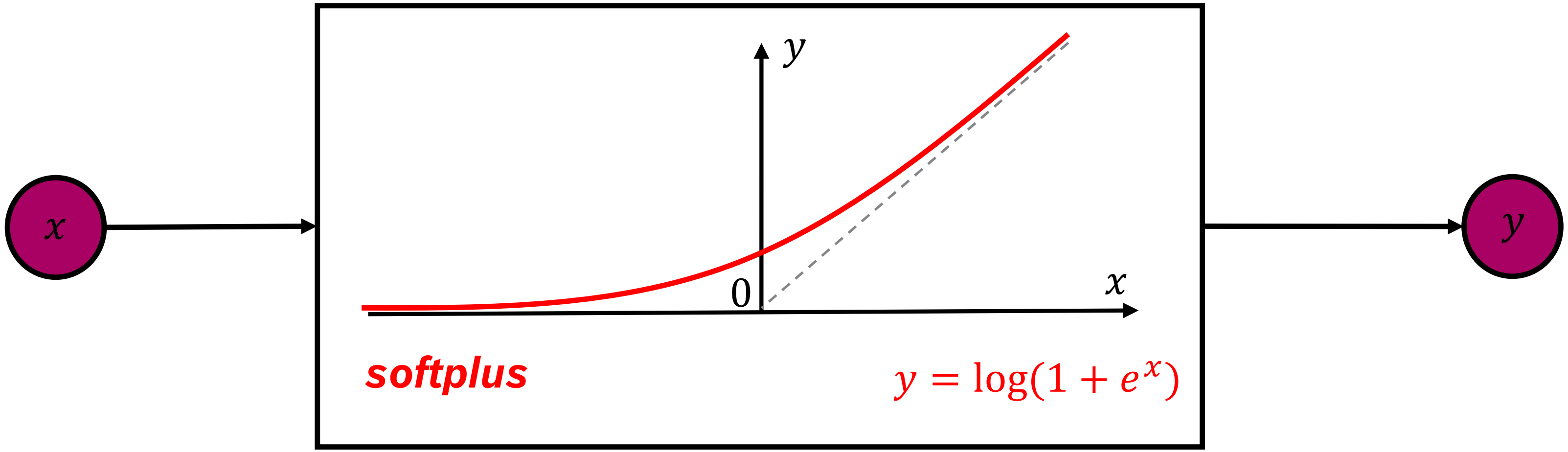
- **State-of-the-art:**
  - Most obvious choices do not work well! (e.g. softplus)
  - Open research field

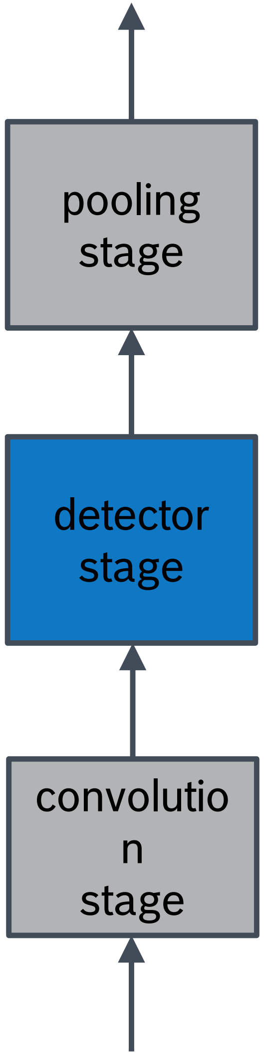




Convolutional Neural Networks
The Detector Stage

- **Requirements:**
  - Highly nonlinear behavior (soft thresholding)
  - Balance modeling power, efficiency, ease of optimization
- **State-of-the-art:**
  - Most obvious choices do not work well! (e.g. softplus)
  - Open research field





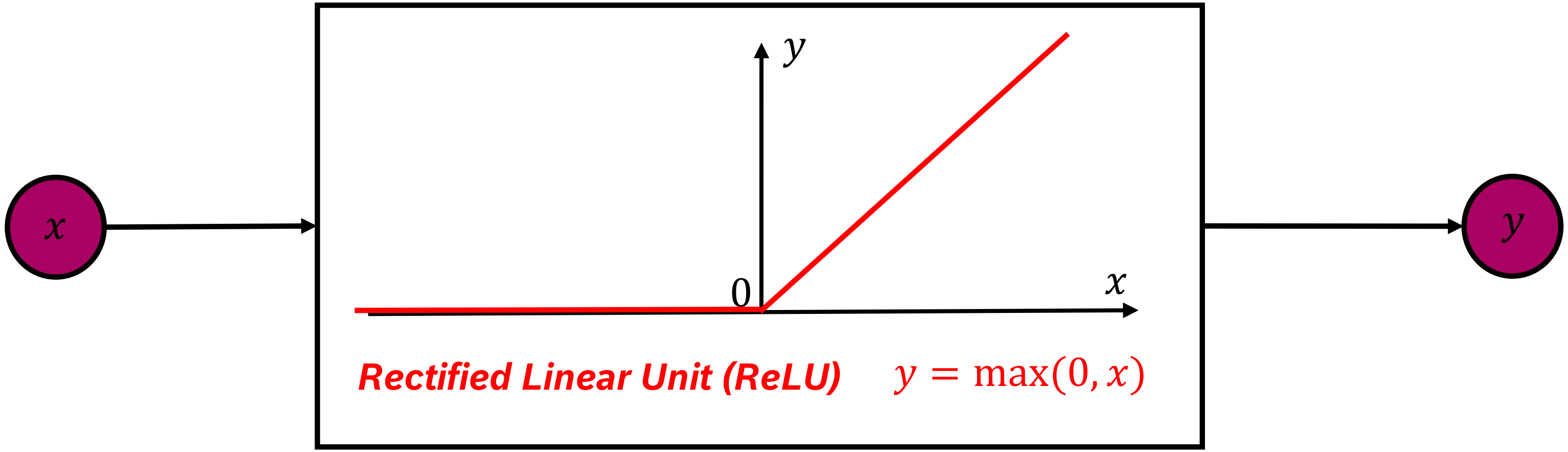
RBRO/ESA2 | 2018-11-08

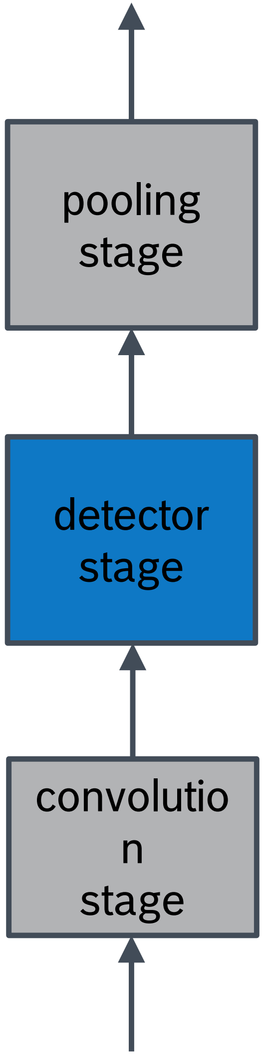
BOSCH

Convolutional Neural Networks
The Detector Stage

- **Requirements:**
  - Highly nonlinear behavior (soft thresholding)
  - Balance modeling power, efficiency, ease of optimization

- **State-of-the-art:**
  - Most obvious choices do not work well! (e.g. softplus)
  - Open research field

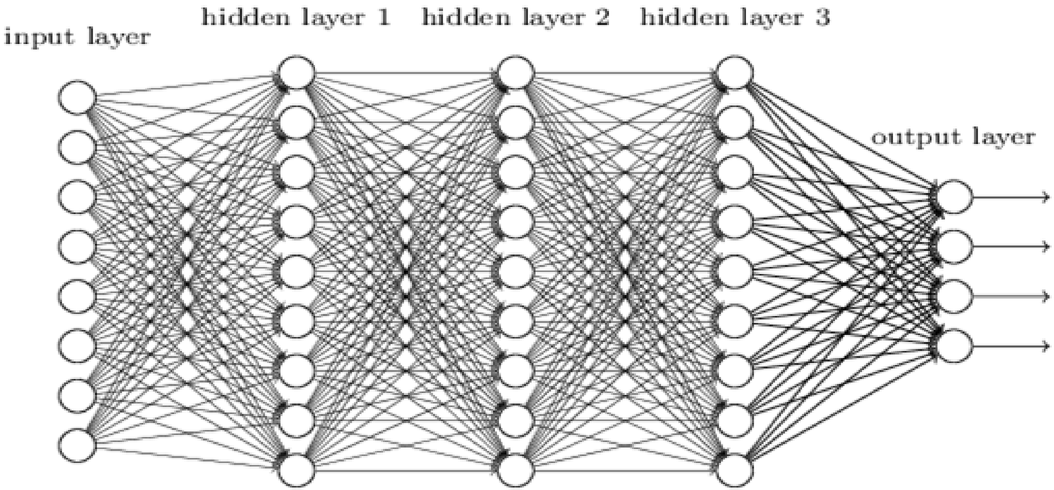




Convolutional Neural Networks
Does the Multi Layer Perceptron Work?

- **Rarely!** ☹

- **Problems:**
  - Huge number of parameters
  - Low spatial invariance

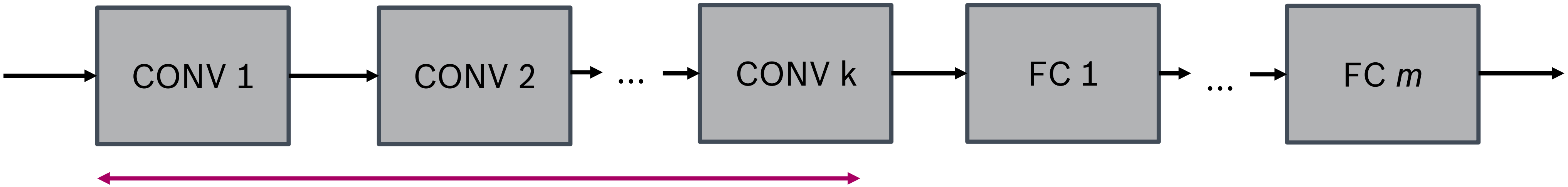


Convolutional Neural Networks
Intuition

- **Solution:**
  - Scan for smaller local patterns and successively group them into larger ones
  - Introduce Convolutional (CONV) Layers

Convolutional Neural Networks
Intuition

- **Solution:**
  - Scan for smaller local patterns and successively group them into larger ones
  - Introduce Convolutional (CONV) Layers



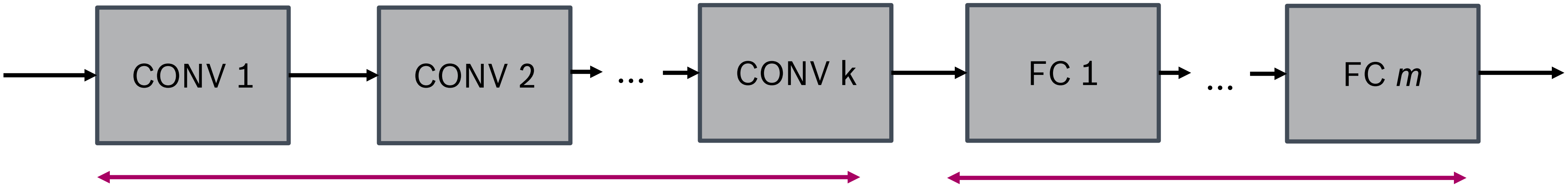
locate spatial/temporal patterns
(edges, circular patterns, nose, mouth, etc.)

many layers but few parameters per layer

BOSCH

Convolutional Neural Networks
Intuition

- **Solution:**
  - Scan for smaller local patterns and successively group them into larger ones
  - Introduce Convolutional (CONV) Layers



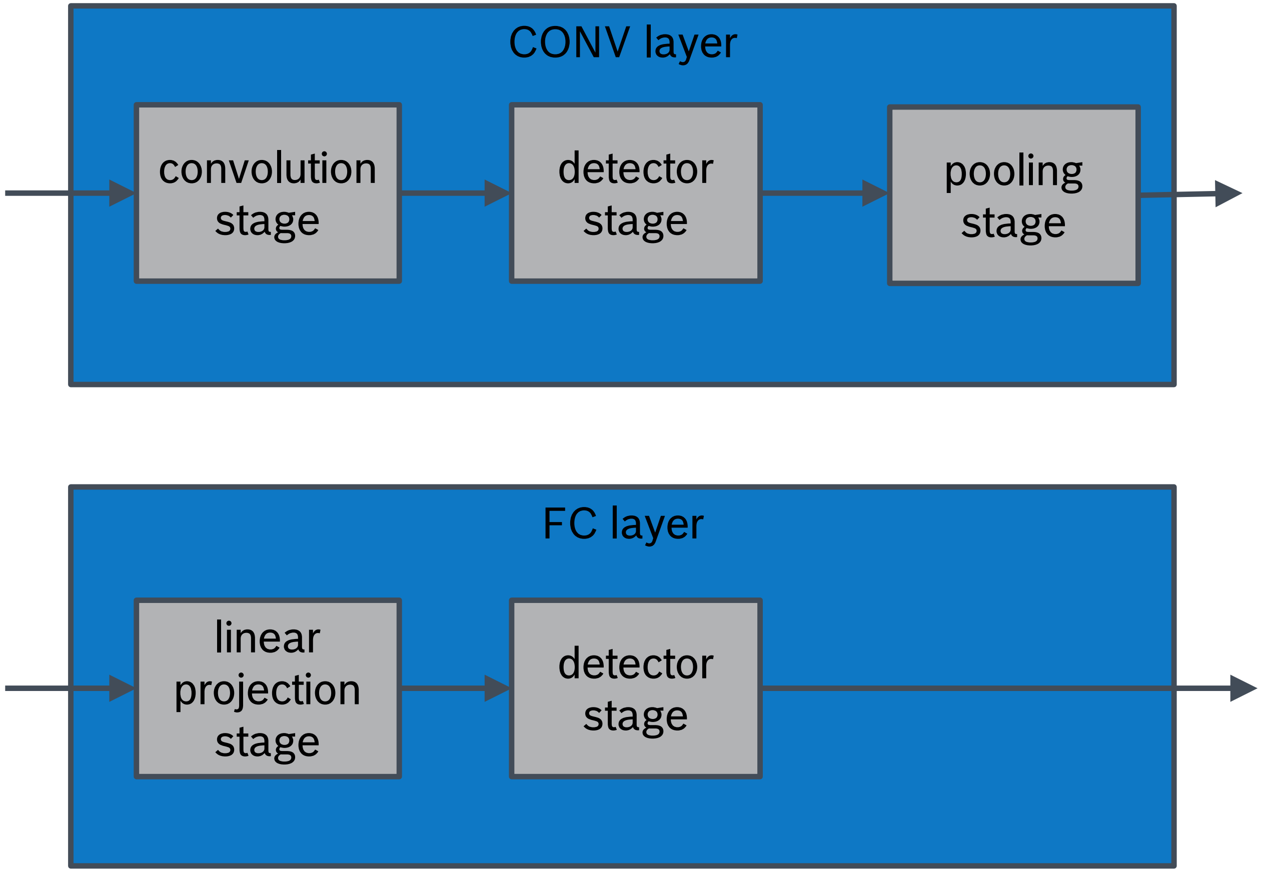
locate spatial/temporal patterns
(edges, circular patterns, nose, mouth, etc.)

many layers but few parameters per layer

recognize the presence of abstract patterns
(face-frontal, face-profile-left, face, etc.)

many parameters per layer but few layers

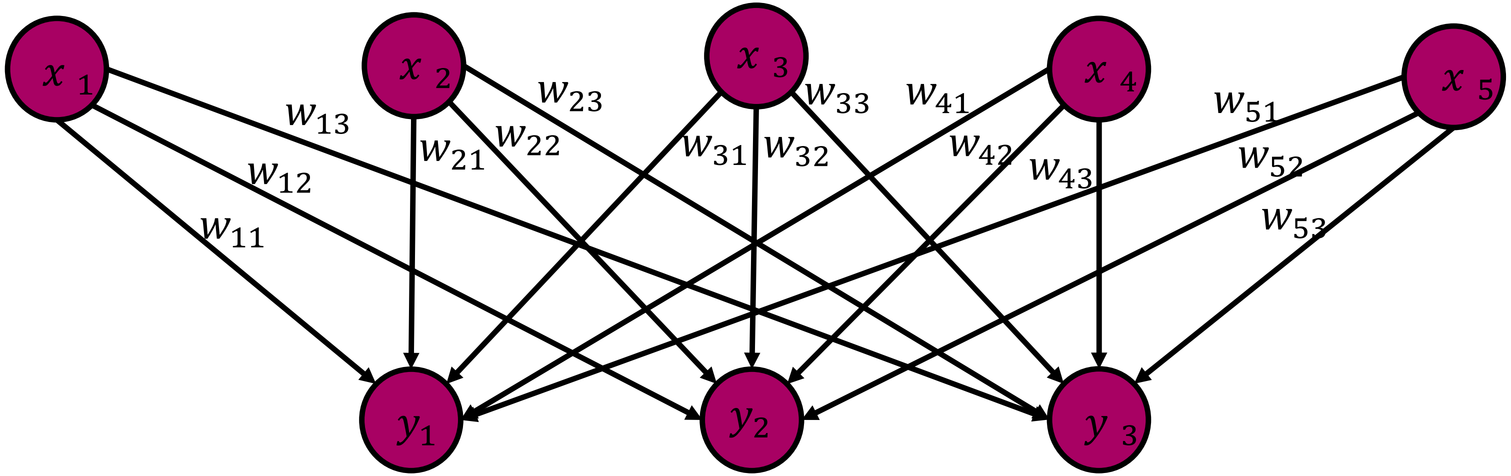
Convolutional Neural Networks
Layer Stages

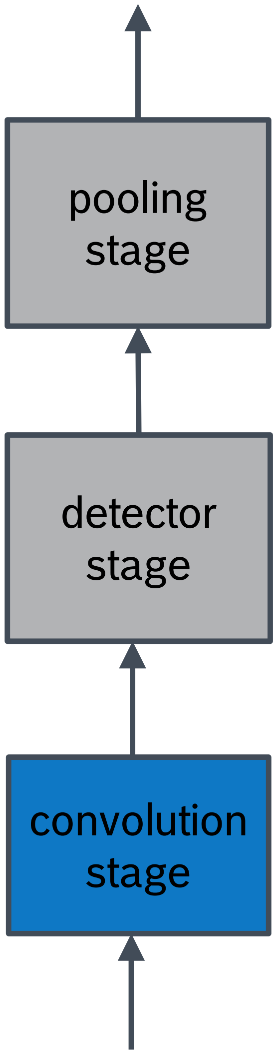


Convolutional Architectures
CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

$$y_j = \sum_{i=1}^{N_\mathrm{w}} w_i \cdot x_{i+j-1}$$

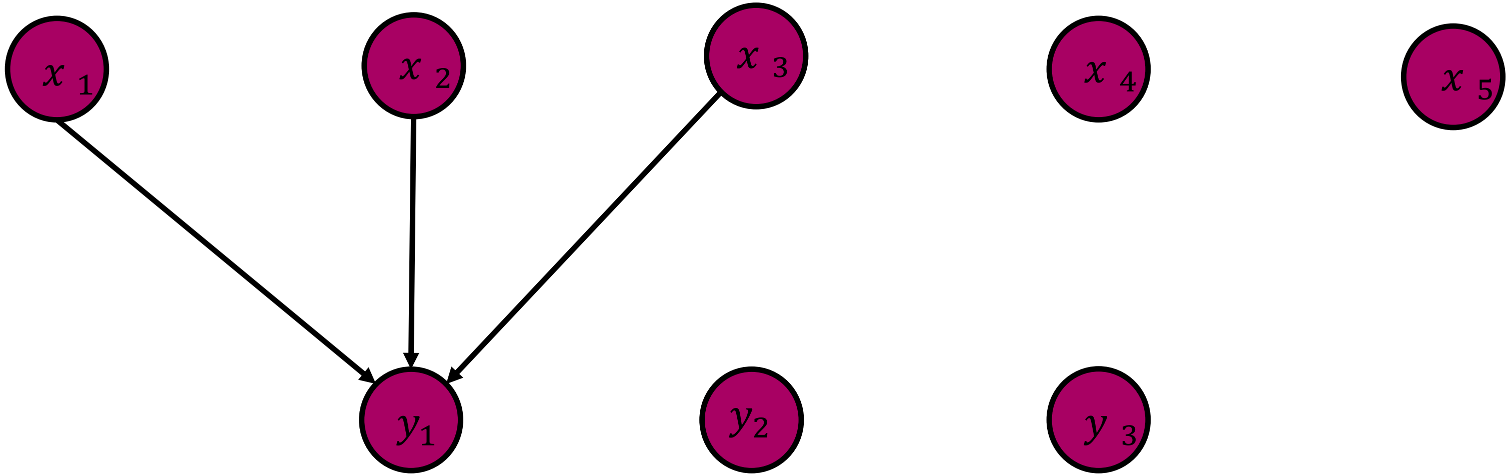


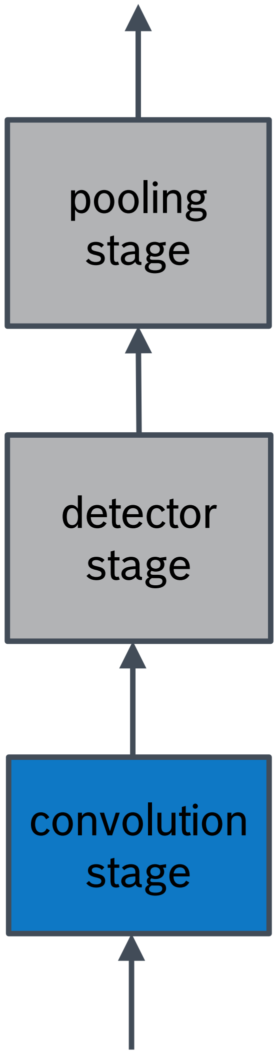


Convolutional Architectures
CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

$$y_j = \sum_{i=1}^{N_w} w_i \cdot x_{i+j-1}$$

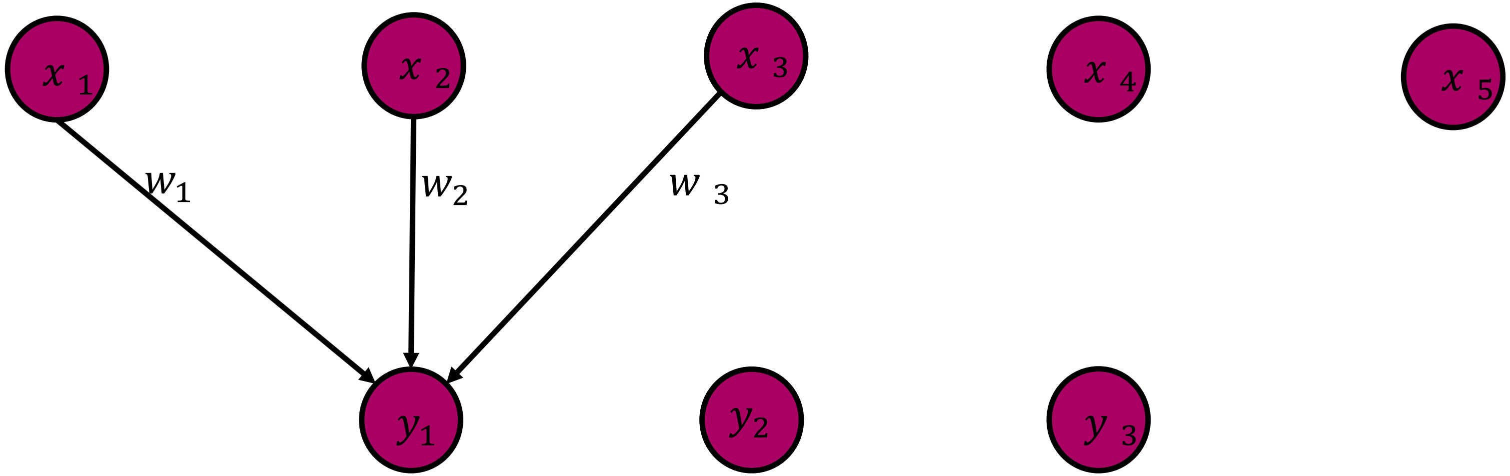


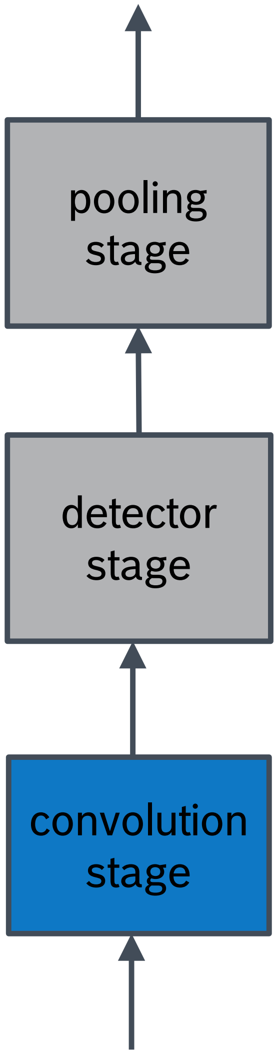


Convolutional Architectures
CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

$$y_j = \sum_{i=1}^{N_w} w_i \cdot x_{i+j-1}$$



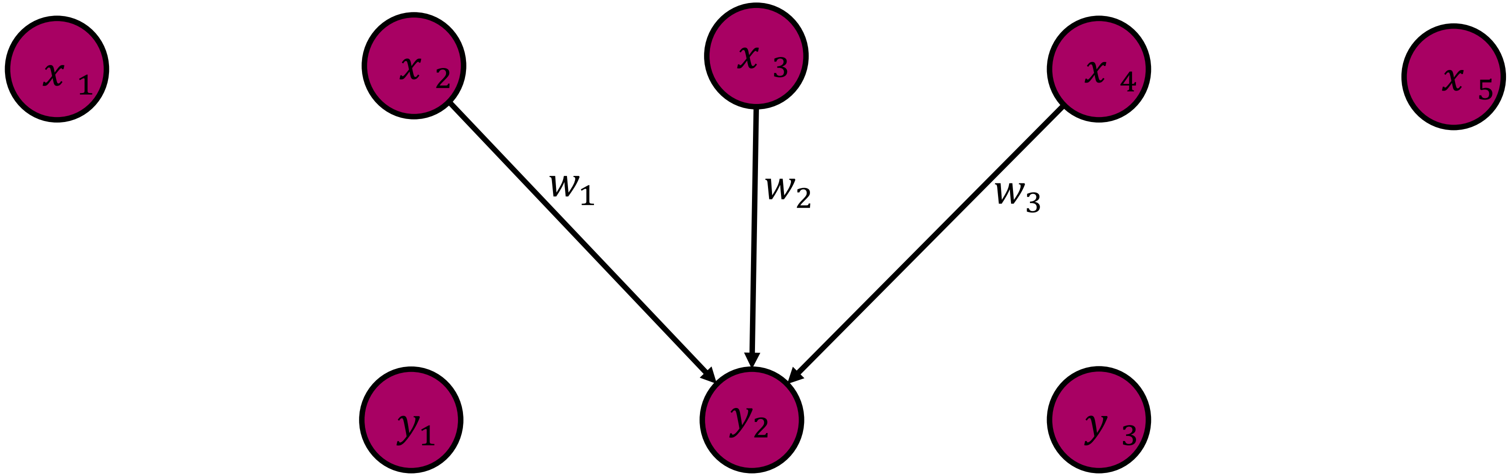


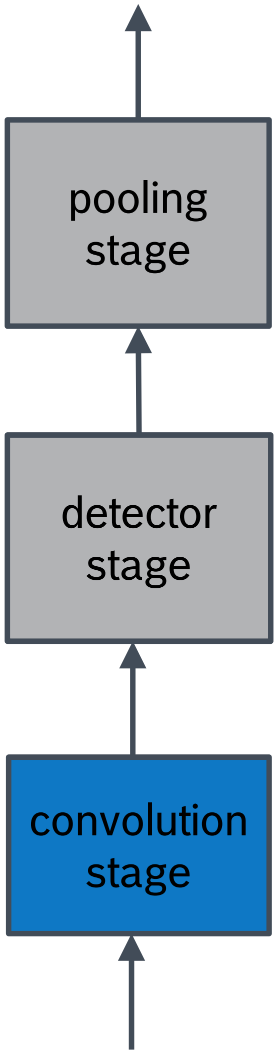
Convolutional Architectures

CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

$$y_j = \sum_{i=1}^{N_\mathrm{w}} w_i \cdot x_{i+j-1}$$

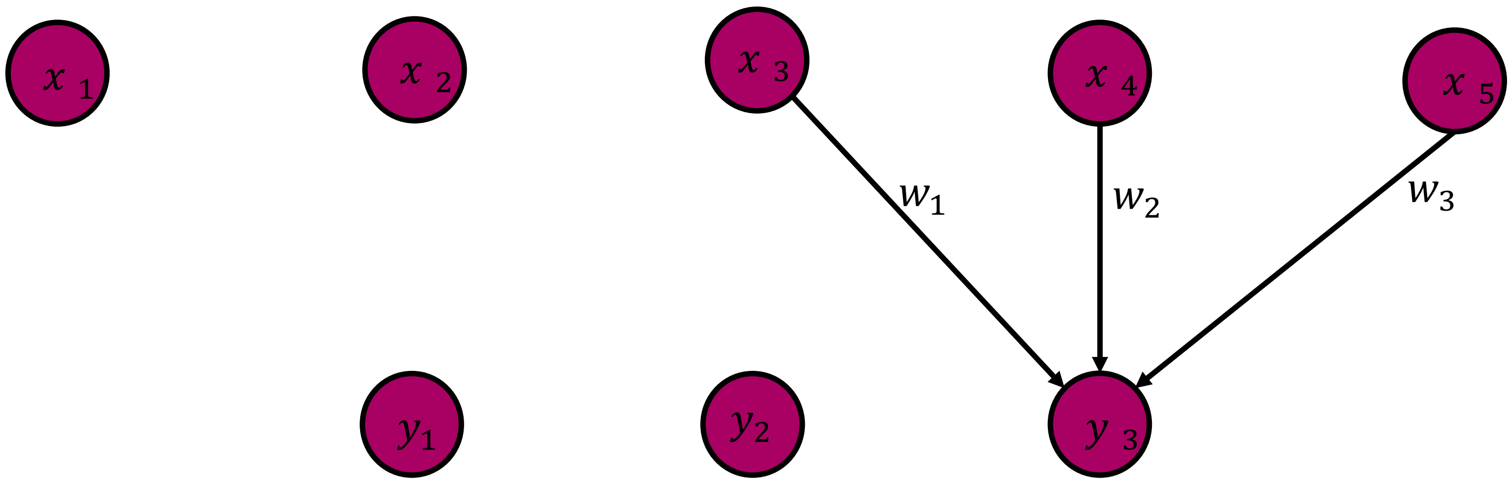


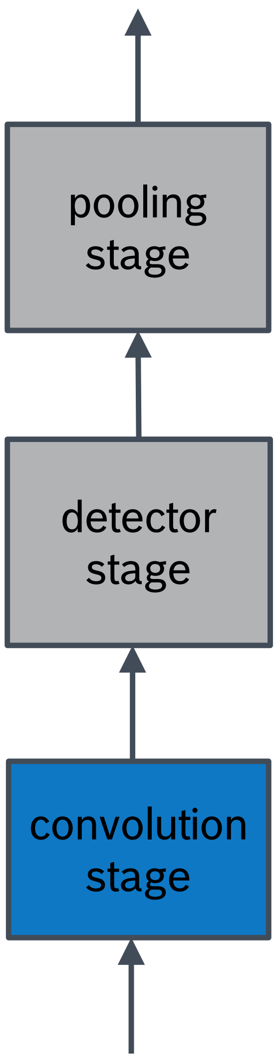


Convolutional Architectures
CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

$$y_j = \sum_{i=1}^{N_{\mathrm{w}}} w_i \cdot x_{i+j-1}$$

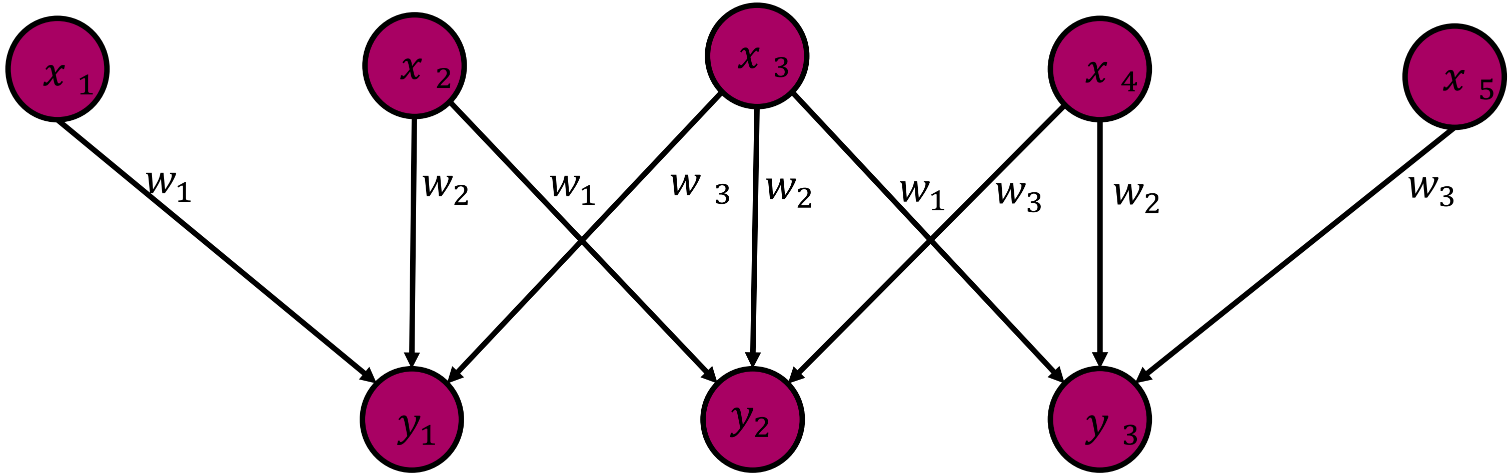


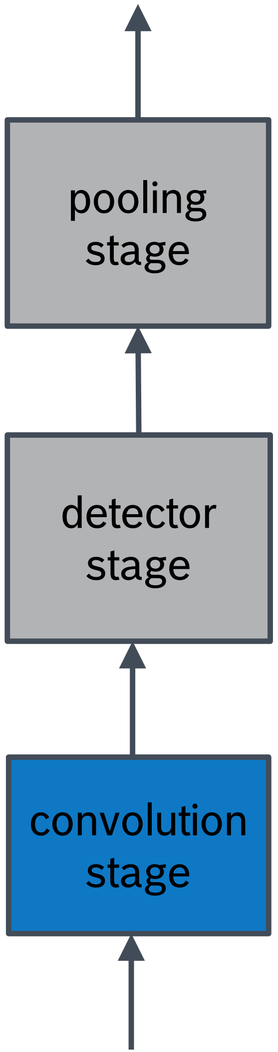


Convolutional Architectures
CONV Layer: The Convolution Stage (1D case)

- **Idea:**
  - Exploit grid-like topology in real data (time series, images)
  - Replace fully connected topology with convolution (or correlation):

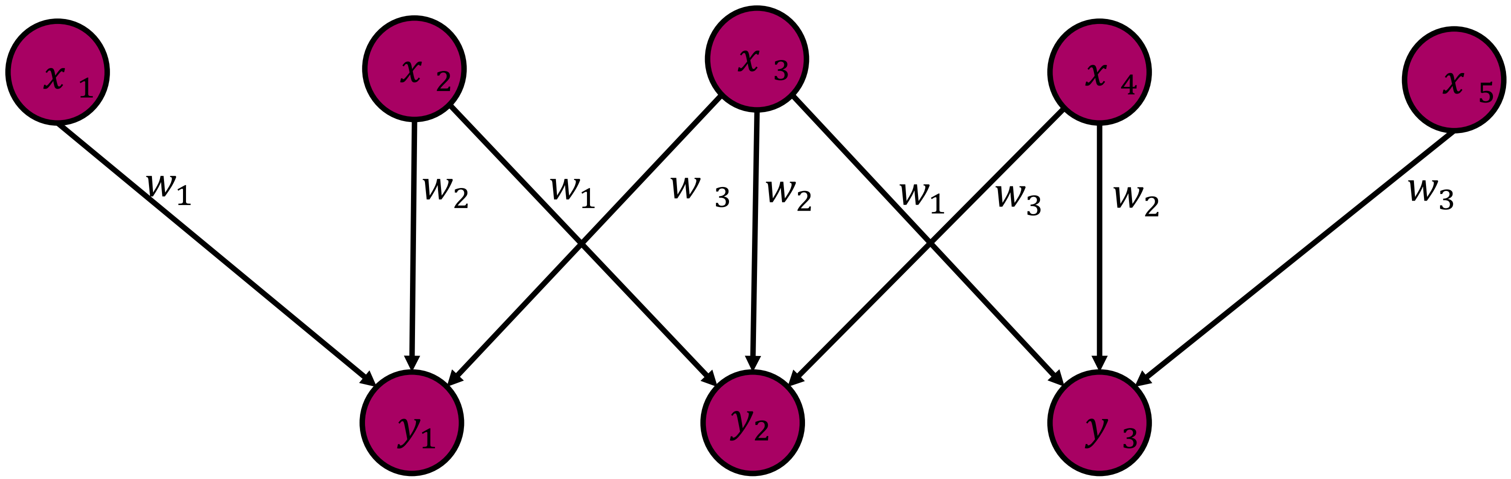
$$y_j = \sum_{i=1}^{N_w} w_i \cdot x_{i+j-1}$$

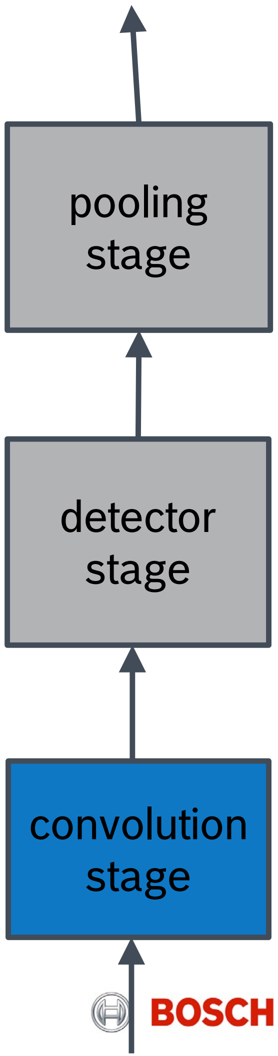




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- **Number of Parameters:** K_w (does not depend on the input size!)
  - We have actually regularized the FC layer!
  - Parameter trimming: $w_{ij} = 0, \text{for } 0 \leq i - j < N_\mathrm{w}$
  - Parameter tying: $w_{ij} = w_{kl}, \text{for } (k - i) = (l - j)$



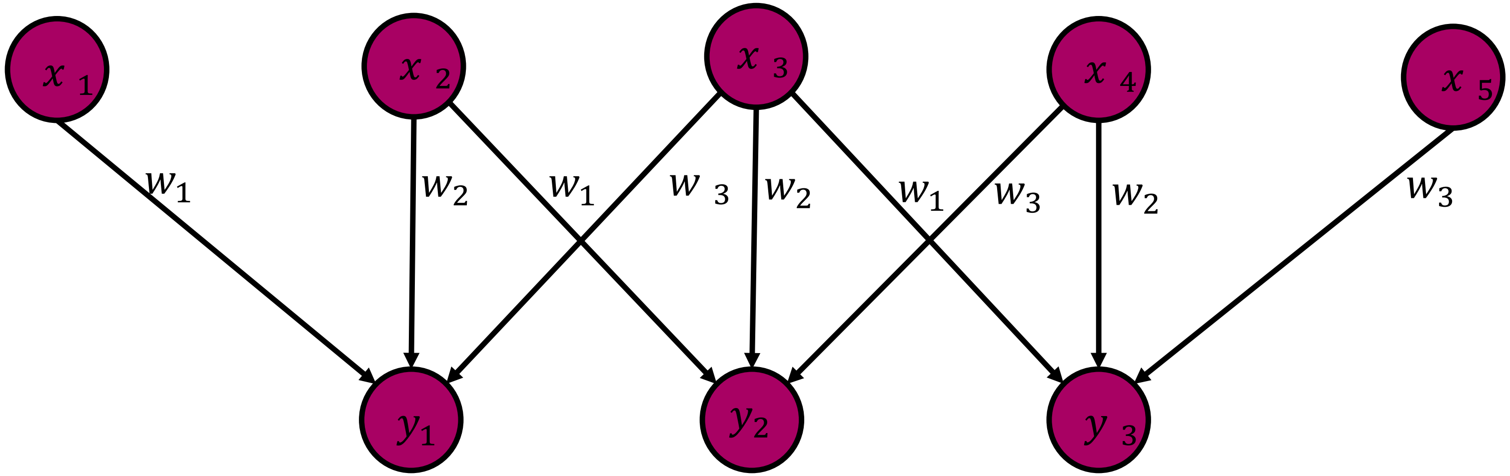


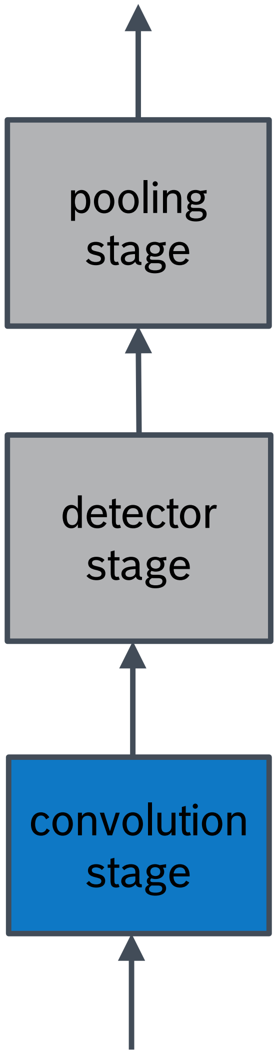
BOSCH

Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- **Time Complexity:** $N_{\mathrm{w}} \cdot N_{\mathrm{out}}$ MACs

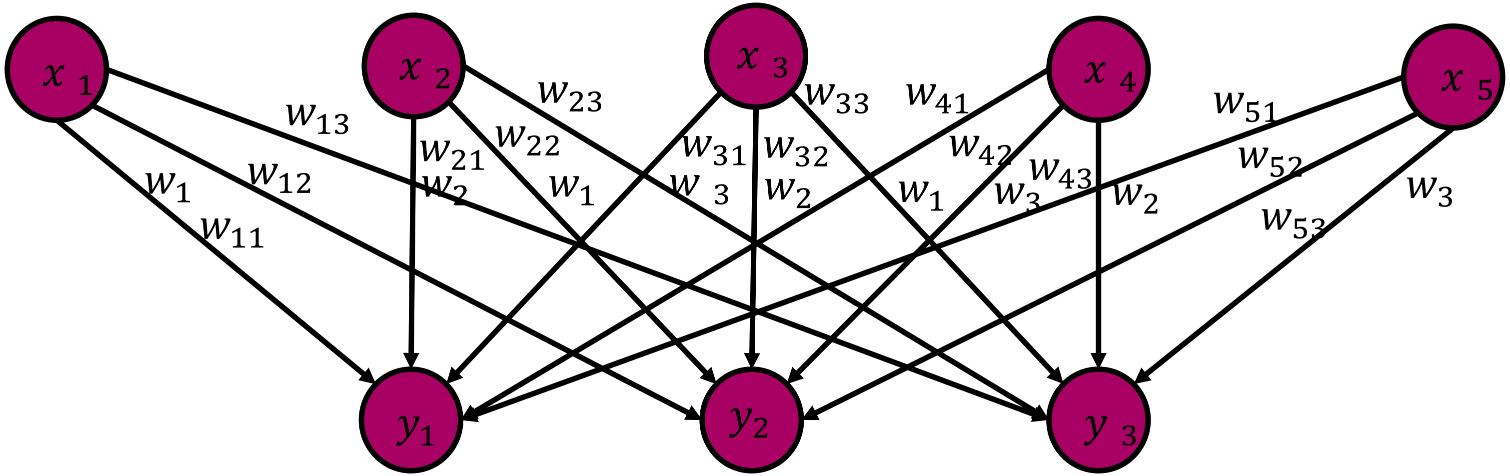
What is the speedup over a 1D Fully connected layer with the same number of inputs?

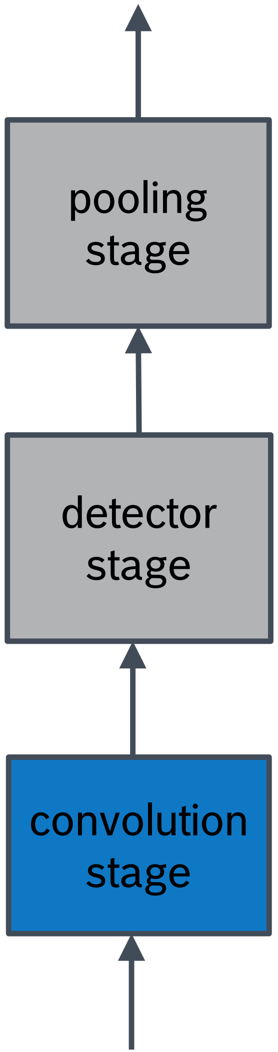




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

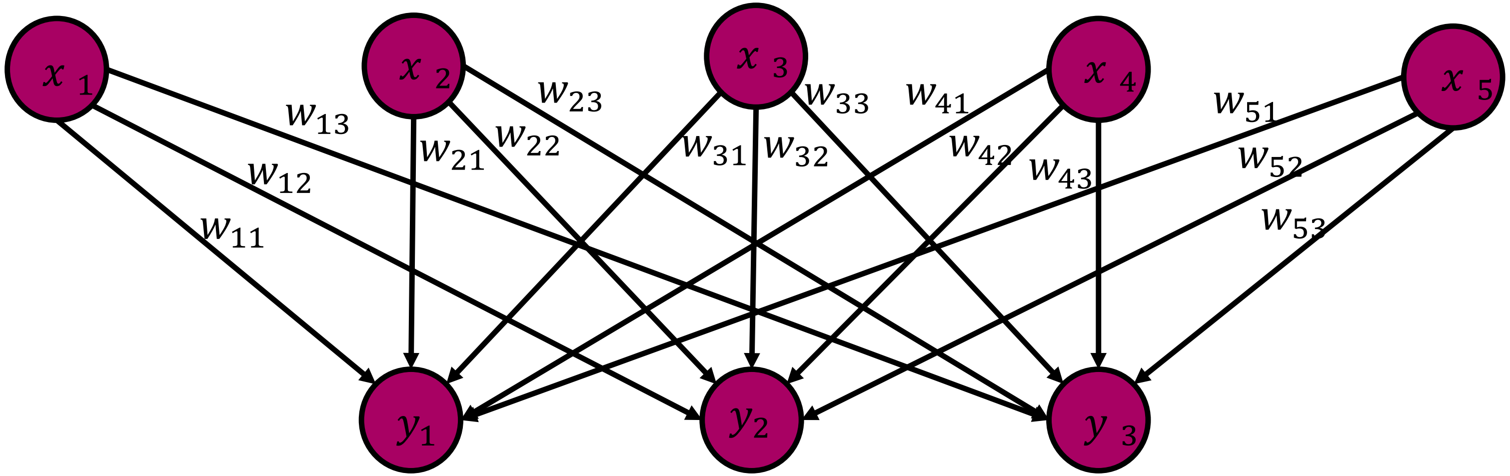


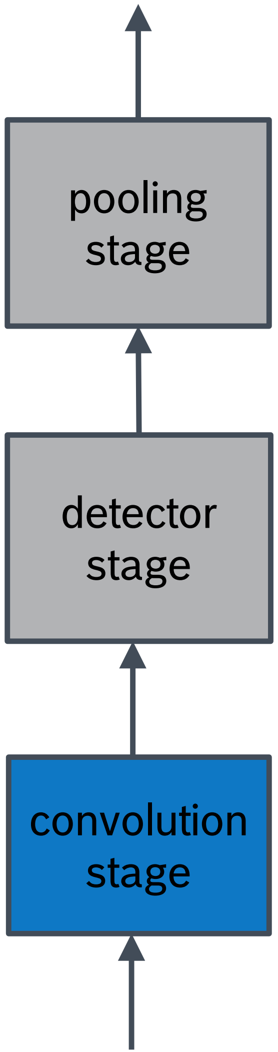


- Equivariant to translation, but not to scale and rotation
- Regularization: parameter trimming and tying (infinite prior)
- Biologically inspired (one of the few stories of success)

Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

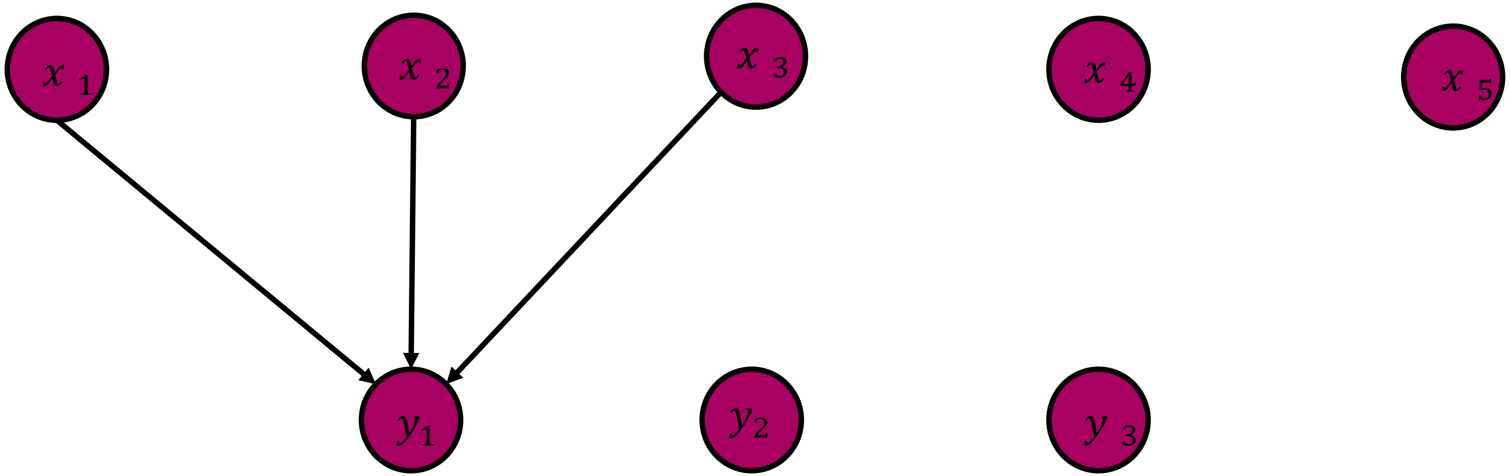
- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

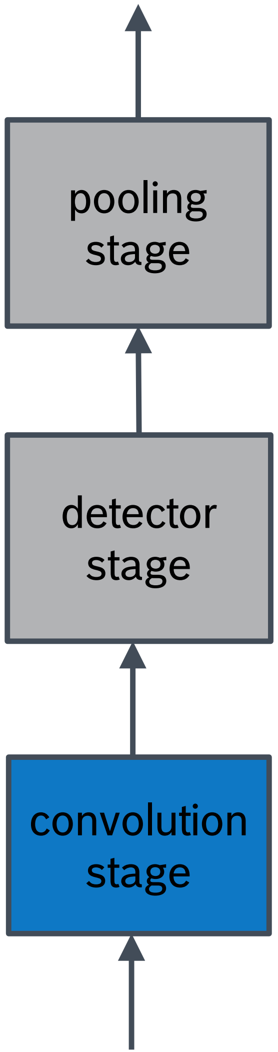




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

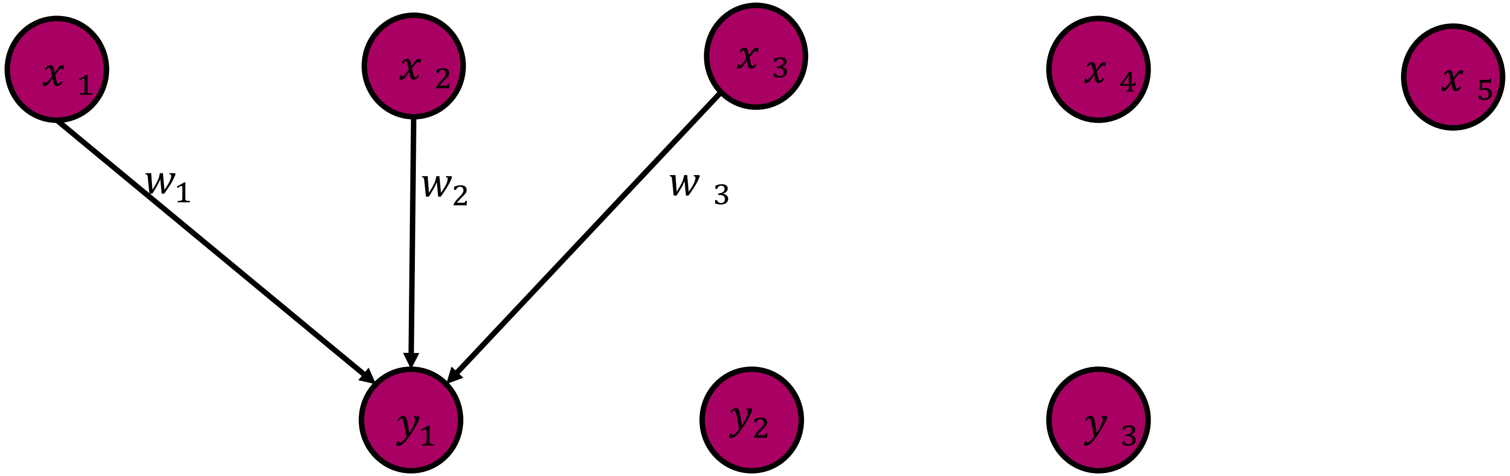


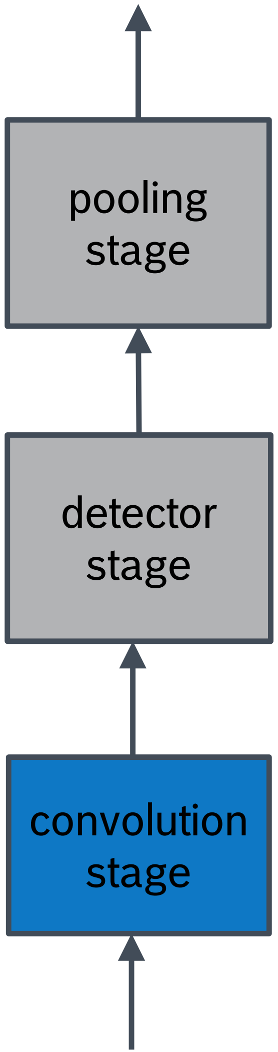


Convolutional Neural Networks

CONV Layer: The Convolution Stage (1D case)

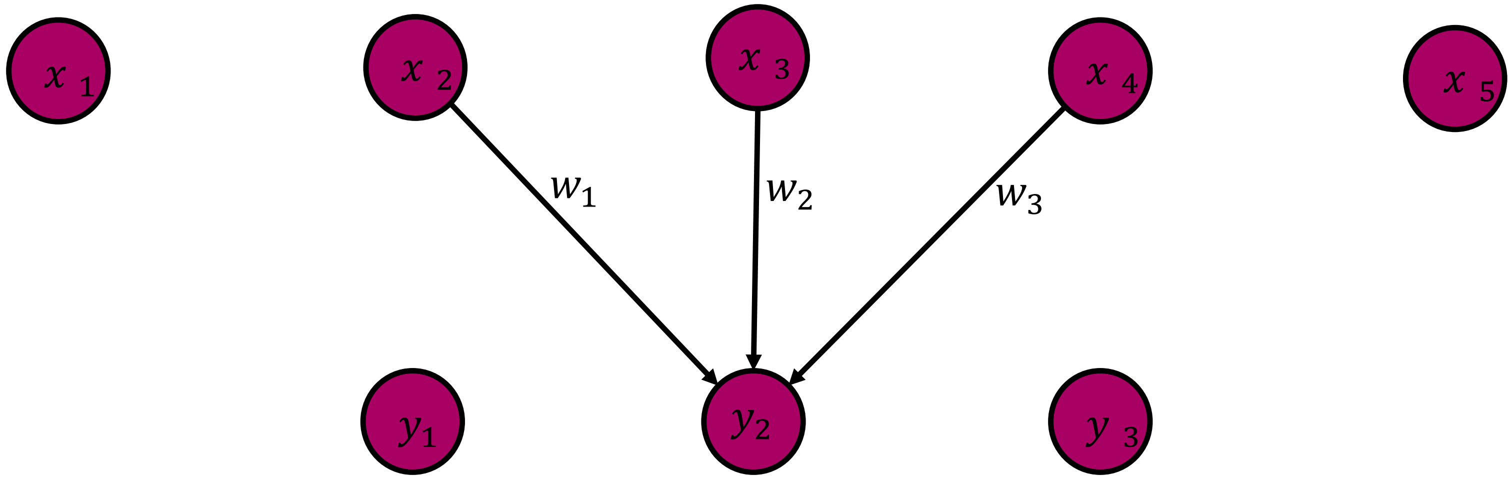
- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

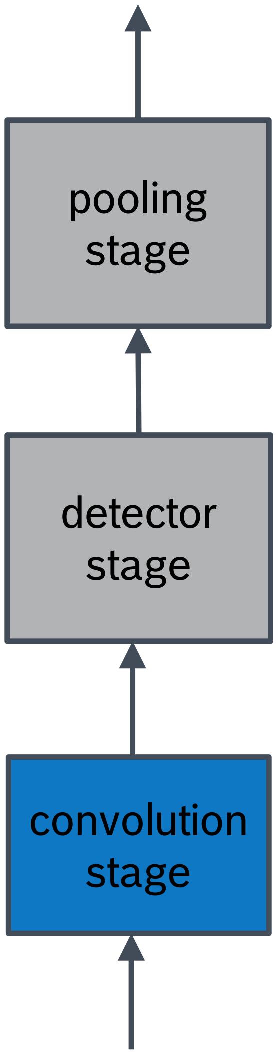




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

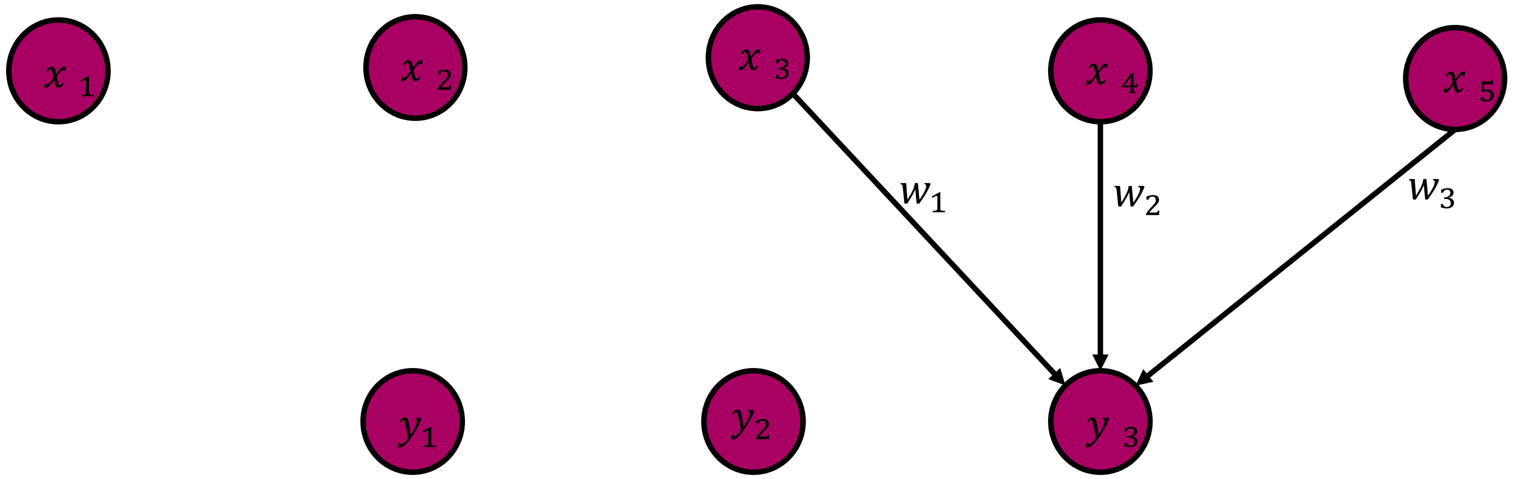
- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

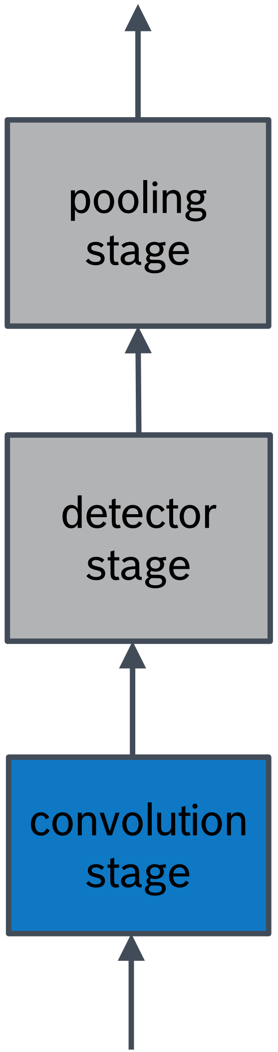




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)



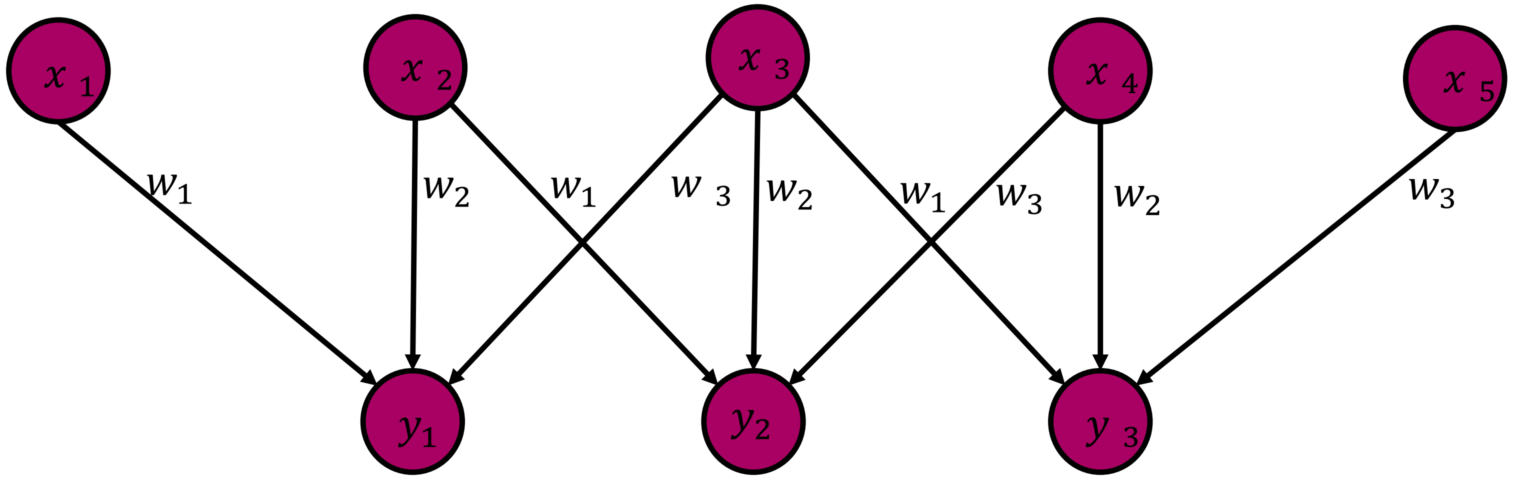


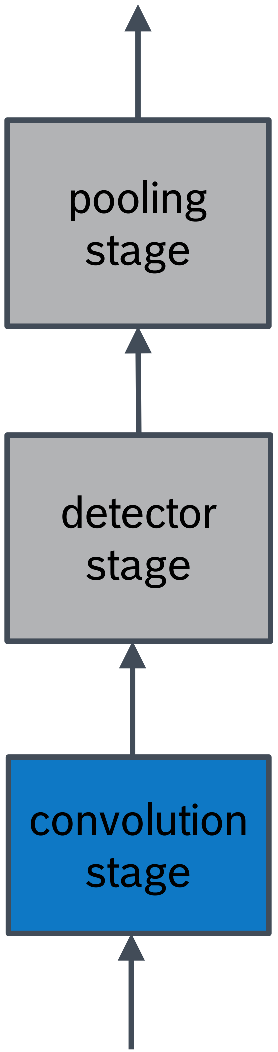
BOSCH

Convolutional Neural Networks

CONV Layer: The Convolution Stage (1D case)

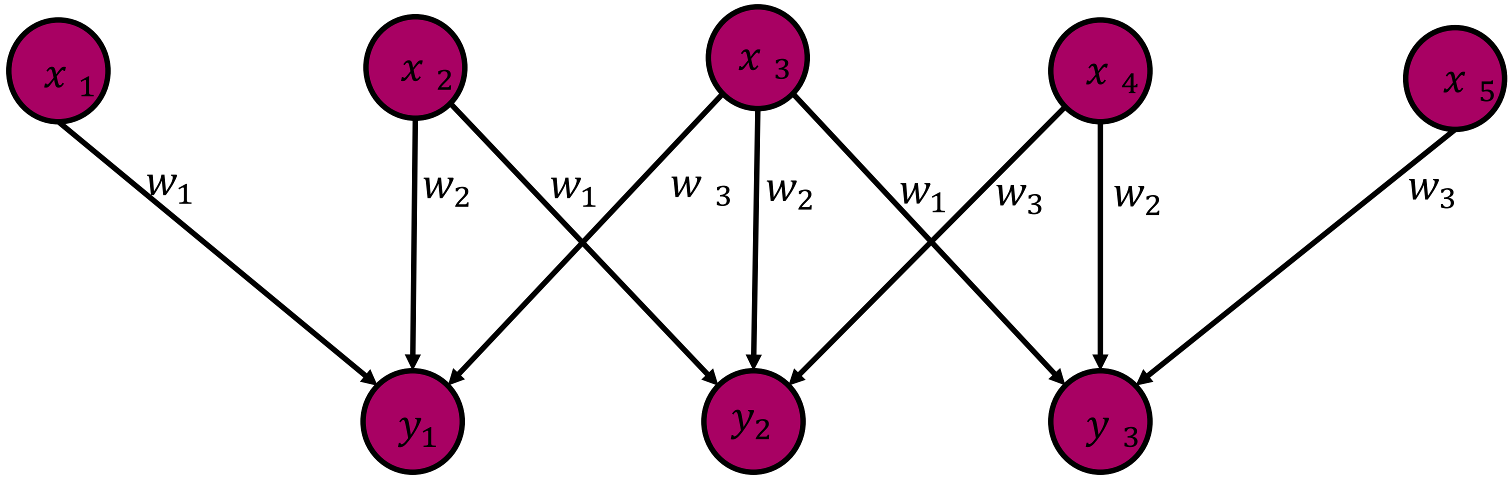
- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

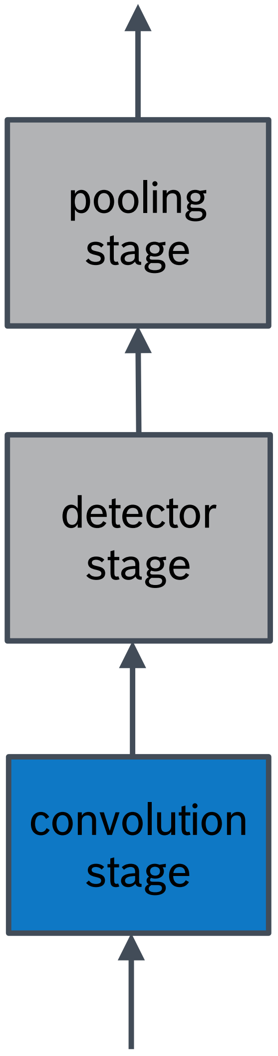




Convolutional Neural Networks
CONV Layer: The Convolution Stage (1D case)

- Exploit grid-like topology in real data (time series, images)
- Replace fully connected topology with convolution (or correlation)

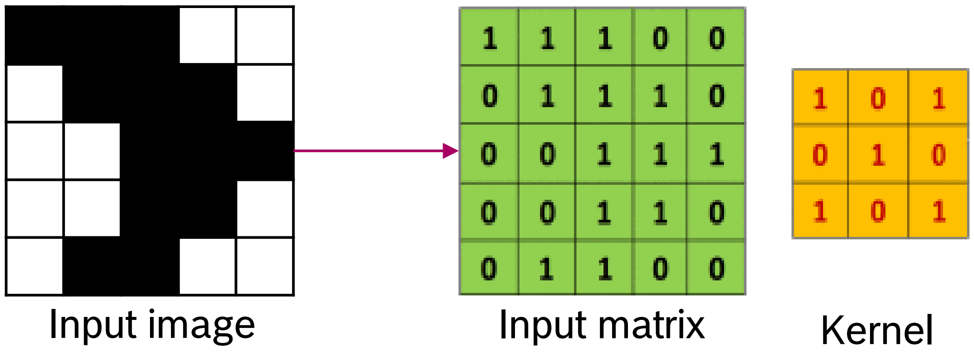


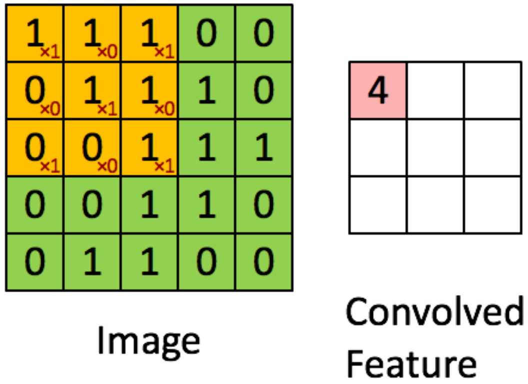


- Equivariant to translation, but not to scale and rotation
- Regularization: parameter trimming and tying (infinite prior)
- Biologically inspired (one of the few stories of success)

Convolutional Neural Networks
CONV Layer: The Convolution Stage (2D case)

- **Input:** $W_{\mathrm{in}} \times H_{\mathrm{in}}$ matrix (*e.g.* binary image)
- **Parameters:** $W_{\mathrm{k}} \times H_{\mathrm{k}}$ matrix (filter)
- **Operation:**
  - Slide the kernel over the input matrix
  - At each position calculate element wise multiplication
  - Calculate the sum of products
- **Output:**
  - $W_{\mathrm{out}} \times H_{\mathrm{out}}$ matrix (activation map)

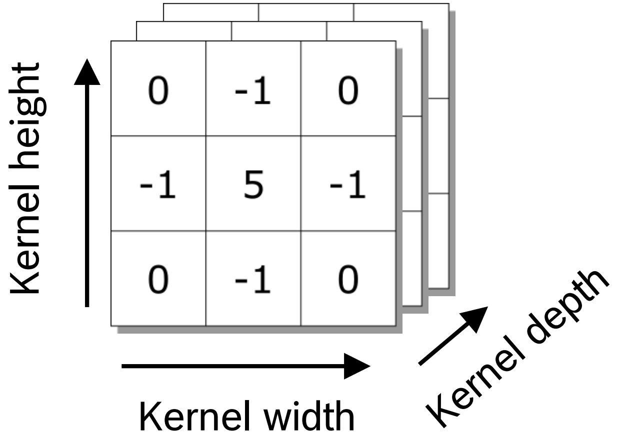


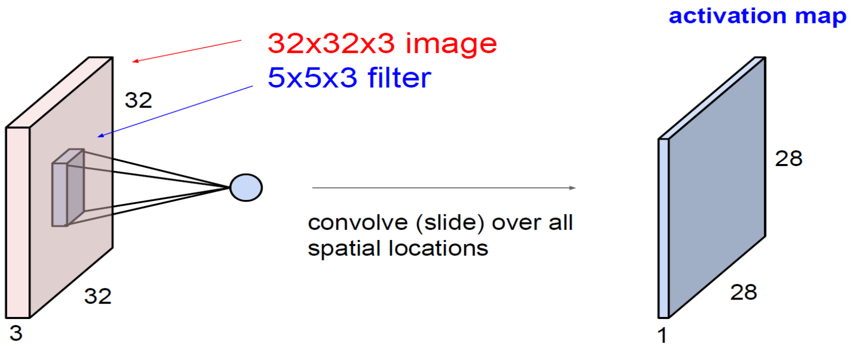


Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Networks
CONV Layer: The Convolution Stage (3D input)

- **Input:** $D_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}$ tensor (D_{in} stacked matrices)
  - D_{in} is the depth of the input (*e.g.* 3 for an RGB image)
- **Parameters:** $D_{\text{in}} \times W_{\text{k}} \times H_{\text{k}}$ matrix (kernel)
  - Must have the same depth as the input
- **Operation:**
  - Slide the kernel over the width and height of the input
  - At each position calculate element wise multiplication
  - Calculate the sum of products
  - This is still 2D convolution (we do **not** slide along depth!)
- **Output:** $W_{\text{out}} \times H_{\text{out}}$ matrix (depth 1)

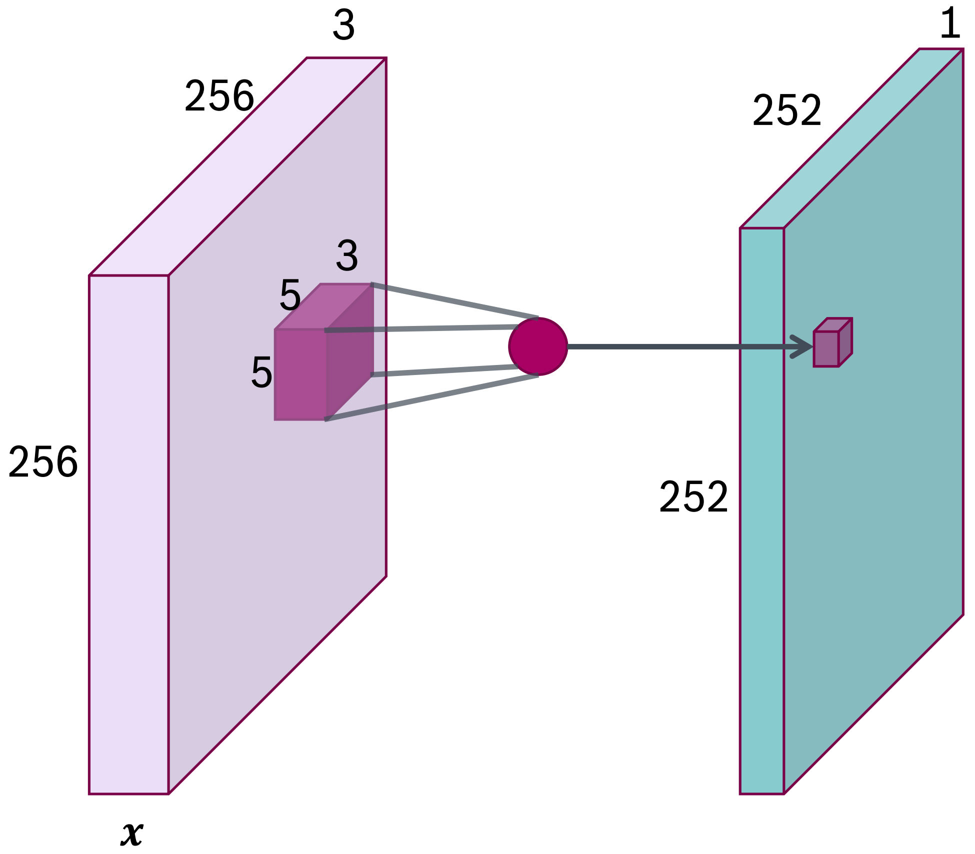


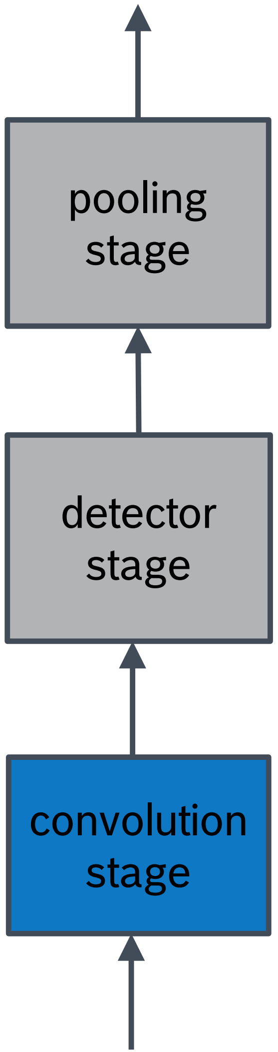


Source: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolutional_neural_networks.html

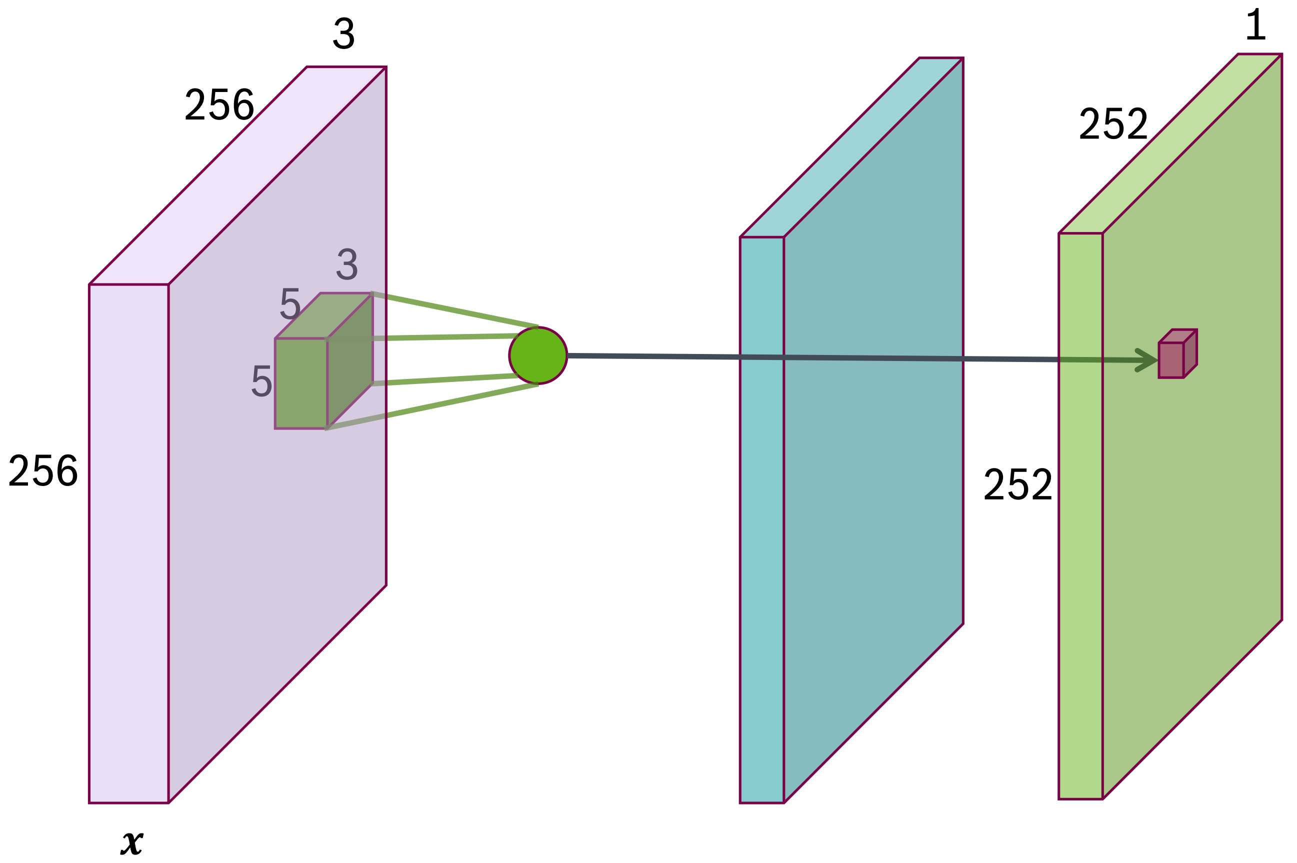
BOSCH

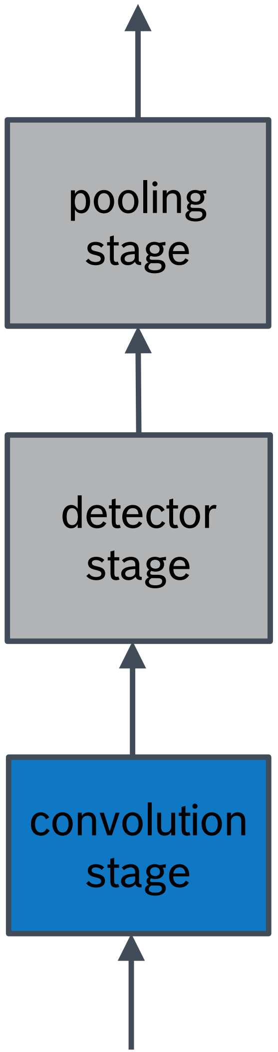
Convolutional Neural Networks
CONV Layer: The Convolution Stage (several kernels)



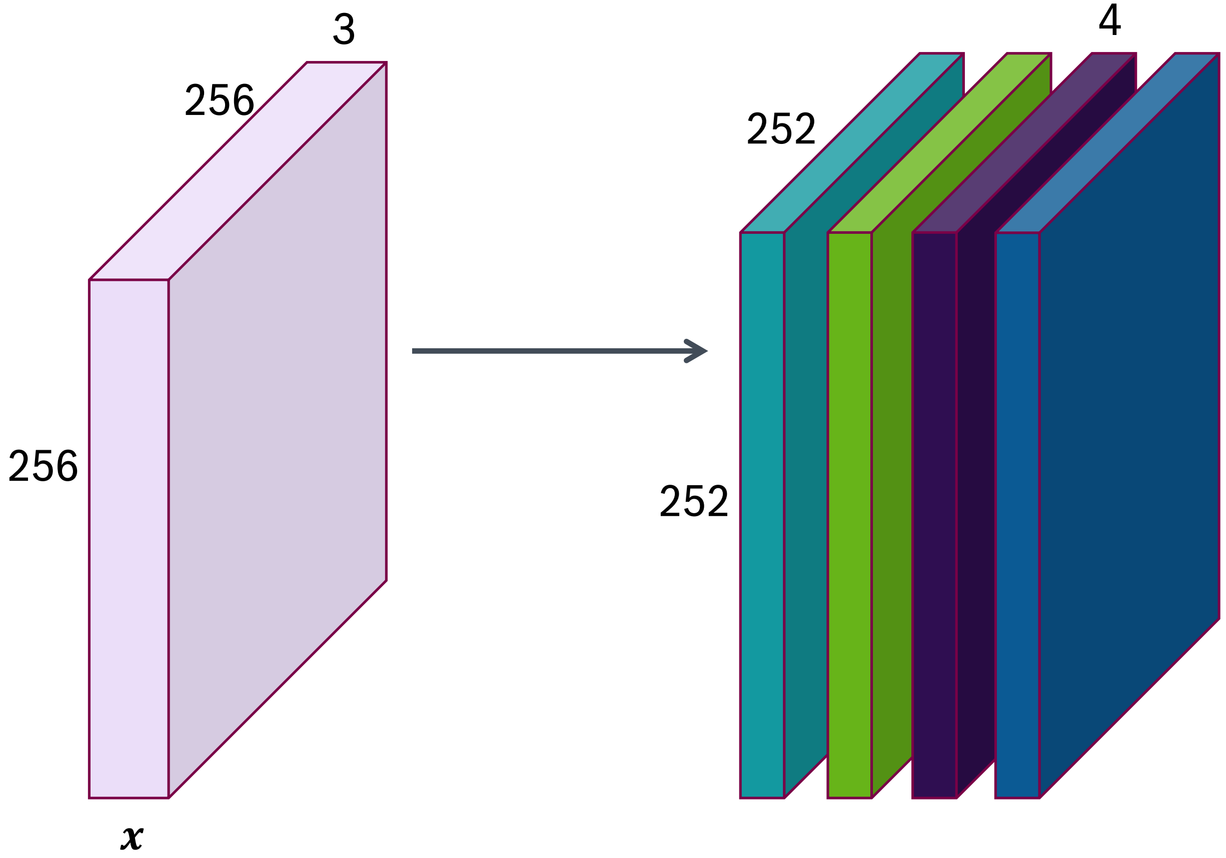


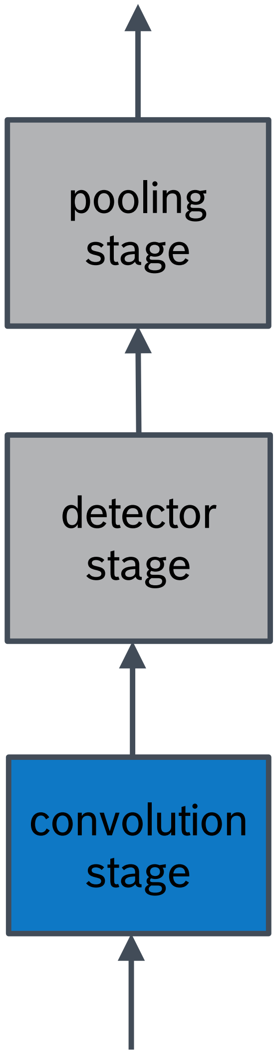
Convolutional Neural Networks
CONV Layer: The Convolution Stage (several kernels)





Convolutional Neural Networks
CONV Layer: The Convolution Stage (several kernels)

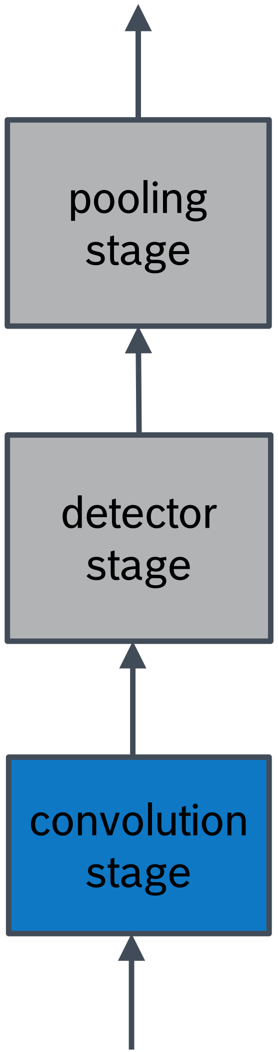




Convolutional Neural Networks
CONV Layer: The Convolution Stage (general case)

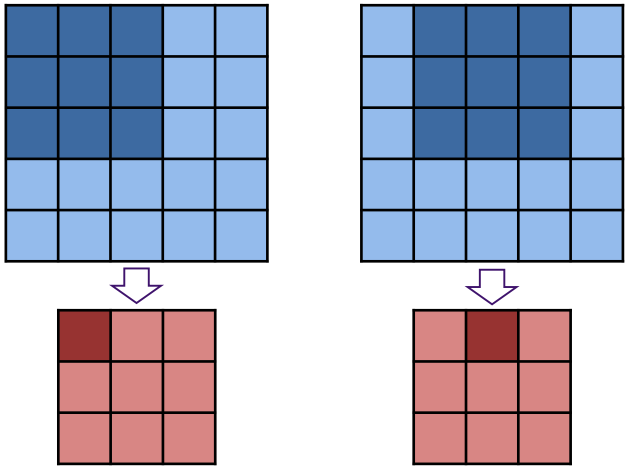
- **Input**: 3D tensor X of size $D_{\mathrm{in}} \times W_{\mathrm{in}} \times H_{\mathrm{in}}$ (D_{in} stacked matrices)
- **Parameters:** 4D tensor W of size $D_{\mathrm{out}} \times D_{\mathrm{in}} \times W_{\mathrm{k}} \times H_{\mathrm{k}}$ - **kernel bank**
  - D_{out} 3D tensors (**kernels**)
  - Each kernel consist of D_{in} 2D matrices (filters)
- **Output:** 3D tensor Y of size $D_{\mathrm{out}} \times (W_{\mathrm{in}} - W_{\mathrm{k}}) \times (H_{\mathrm{in}} - H_{\mathrm{k}})$ (D_{out} stacked matrices)
- **Operation:**
  - Slide each 3D kernel over the input tensor
  - At each position calculate element wise multiplication and sum up the products

$$y_{j,x,y} = \sum_{i=1}^{D_{\mathrm{in}}} \sum_{\Delta y=1}^{H_K} \sum_{\Delta x=1}^{W_K} w_{j,i,\Delta x,\Delta y} \cdot x_{i,x+\Delta x-1,y+\Delta y-1}$$

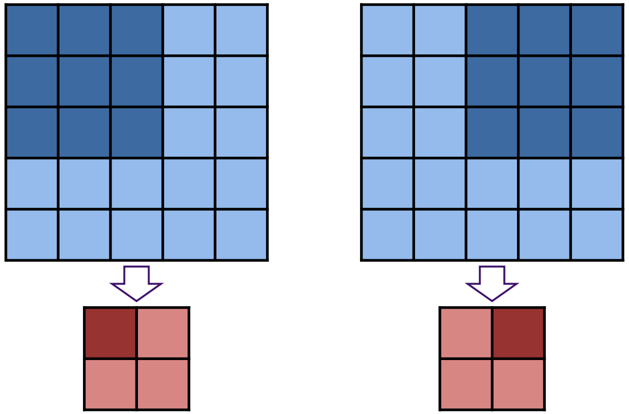


Convolutional Neural Networks
Conv Layer: The Stride Meta-Parameter

- **Motivation:**
  - Reduce the output size
  - Reduce computational complexity
- **Approach:**
  - control the step along each axis when sliding the kernel
- **Meta-parameters:**
  - **horizontal stride**: number of entries we move along the X axis at each step
  - **vertical stride**: number of entries we move along the Y axis at each step



Horizontal stride = 1



Horizontal stride = 2

Convolutional Neural Networks
CONV Layer: The Padding Meta-Parameter

- **Motivation:**
  - Avoid losing a few entries at the border
  - Keep input/output size ratio independent of the input size (1/stride)

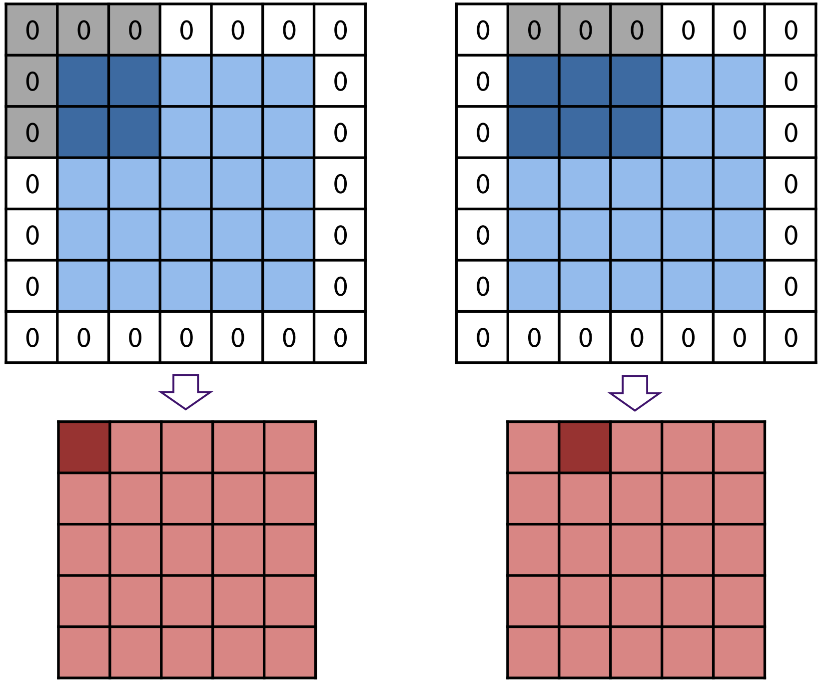
- **Approach:**
  - Add zeros along the border of the input

- **Meta-parameters:**
  - **padding**: number of zeros to add along the border in each slice
  - Common values (assume square kernels of odd size):
    - $\frac{W_{\mathrm{k}}-1}{2}$: **same padding** scheme (preserve input size when stride=1)
    - 0: **valid padding** scheme (only "valid" kernel positions that fit inside the input generate output)

BOSCH

Convolutional Neural Networks
CONV Layer: The Padding Meta-Parameter

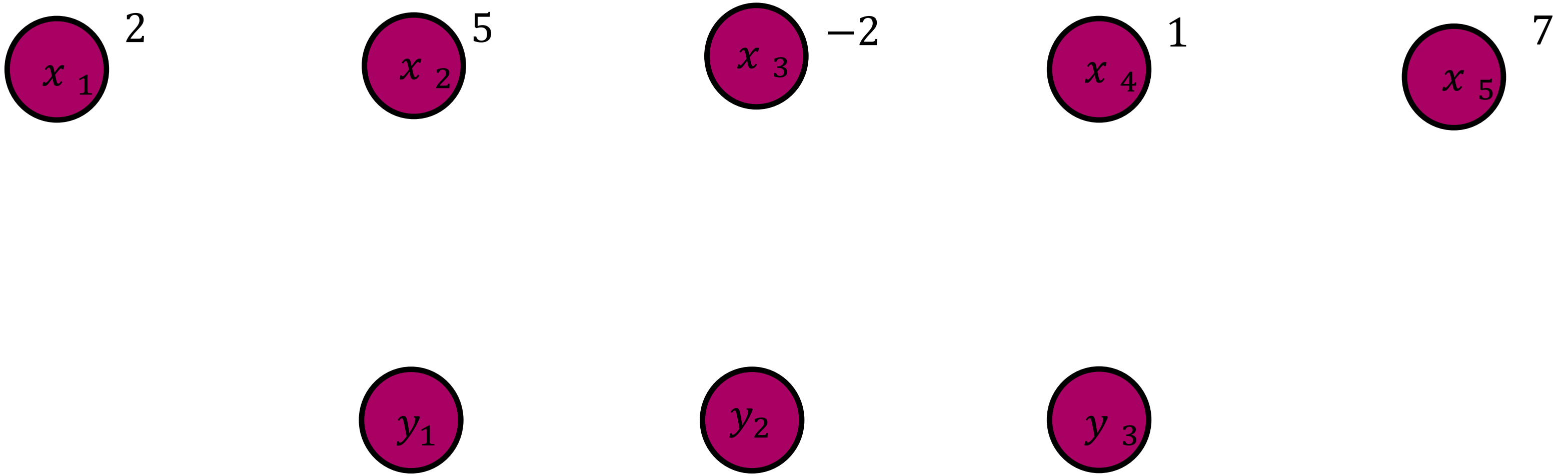
- **Motivation:**
  - Avoid losing a few entries at the border
  - Keep input/output size ratio independent of the input size (1/stride)
- **Approach:**
  - Add zeros along the border of the input
- **Meta-parameters:**
  - **padding**: number of zeros to add along the border in each slice
  - Common values (assume square kernels of odd size):
    - $\frac{W_k - 1}{2}$: **same padding** scheme (preserve input size when stride=1)
    - 0: **valid padding** scheme (only “valid” kernel positions that fit inside the input generate output)

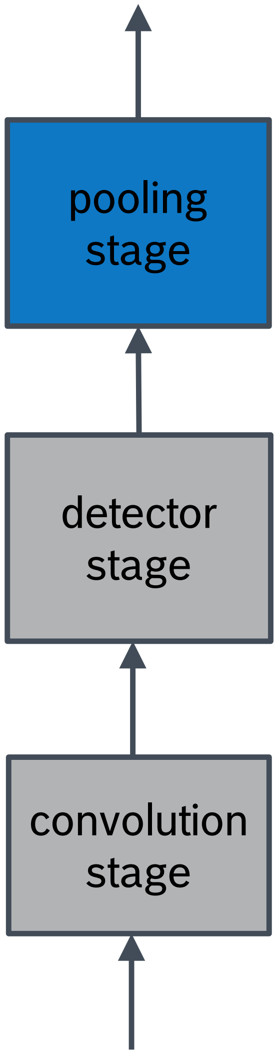


padding = 1

Convolutional Neural Networks
The Pooling Stage (1D case)

- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

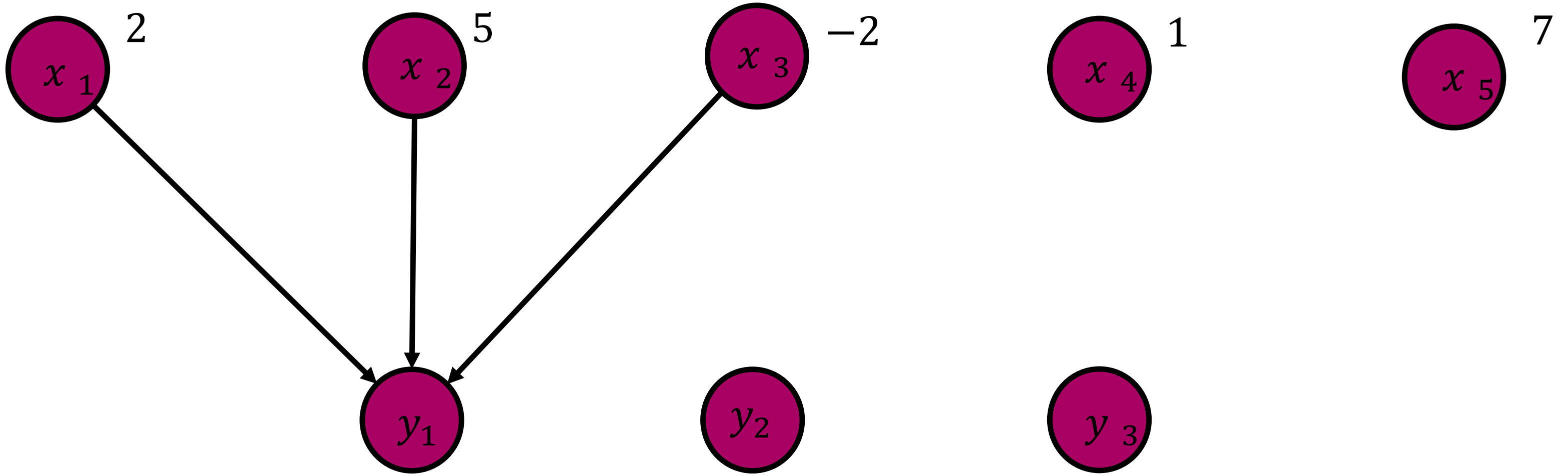


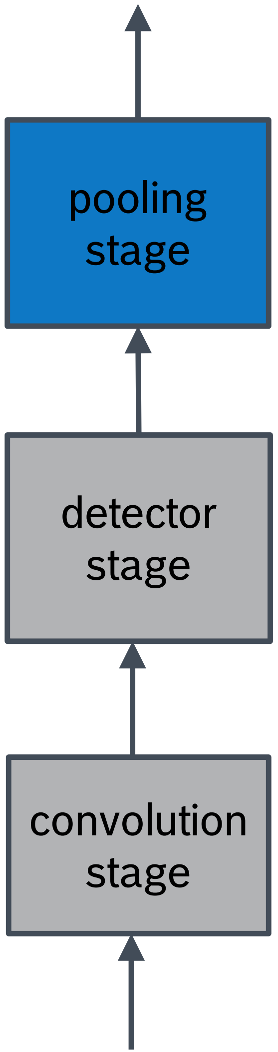


RBRO/ESA2 | 2018-11-08

Convolutional Neural Networks
The Pooling Stage (1D case)

- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

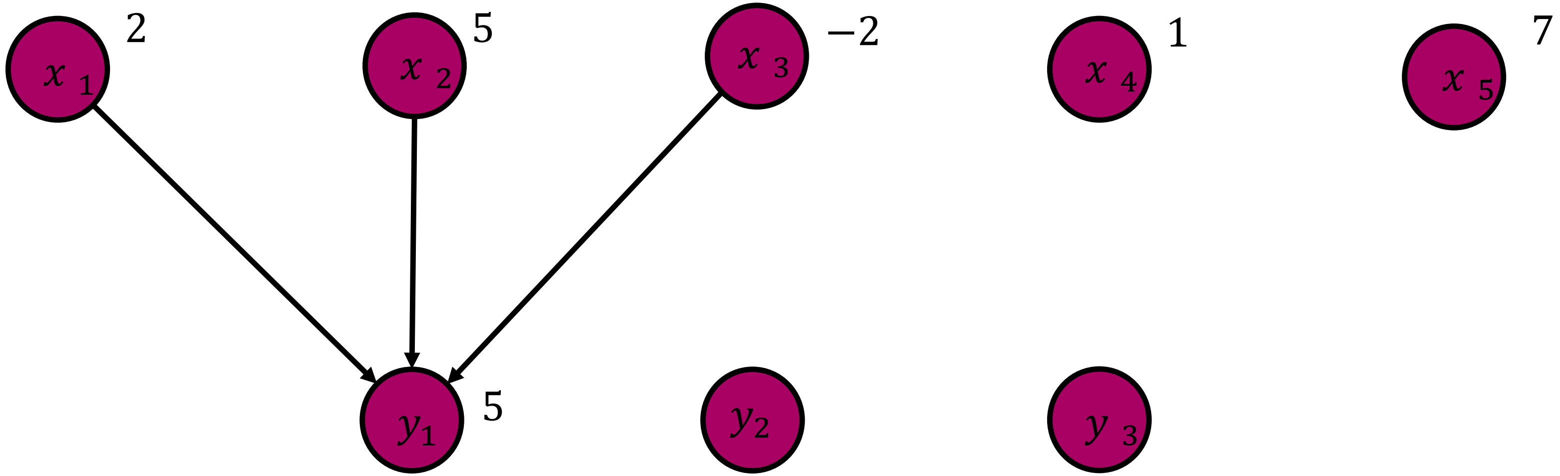


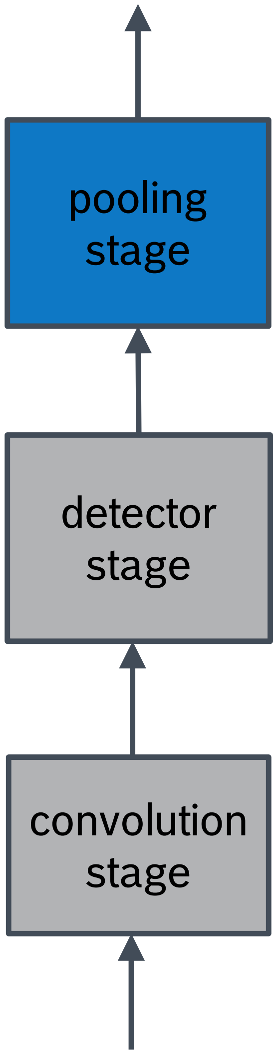


RBRO/ESA2 | 2018-11-08

Convolutional Neural Networks
The Pooling Stage (1D case)

- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

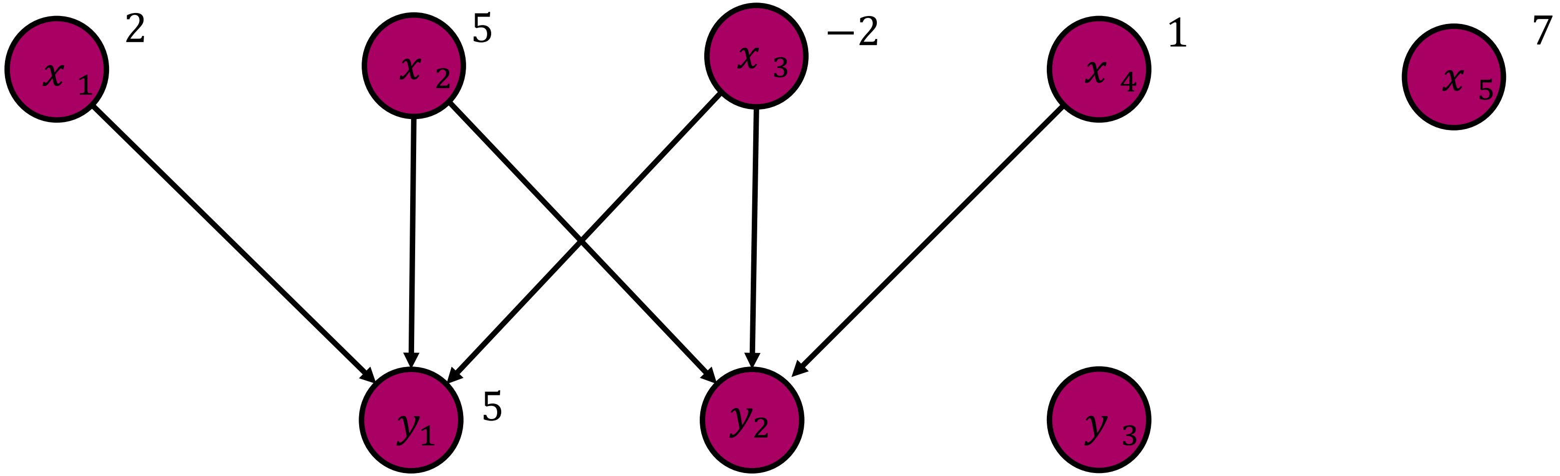


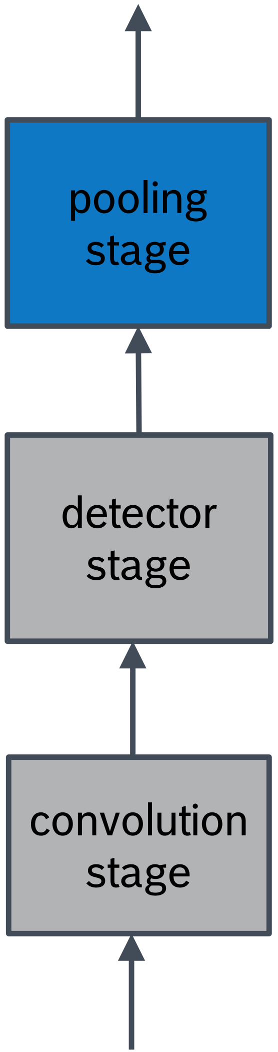


RBRO/ESA2 | 2018-11-08

Convolutional Neural Networks
The Pooling Stage (1D case)

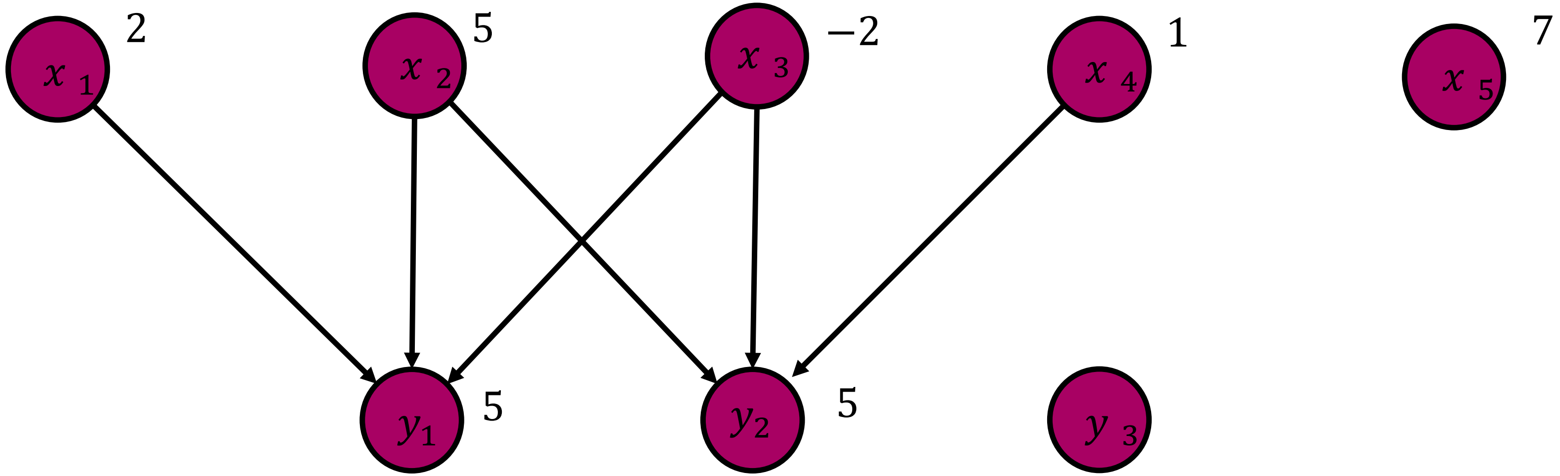
- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

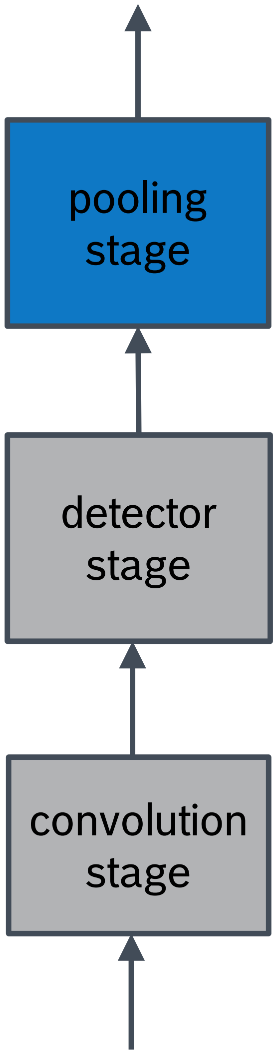




Convolutional Neural Networks
The Pooling Stage (1D case)

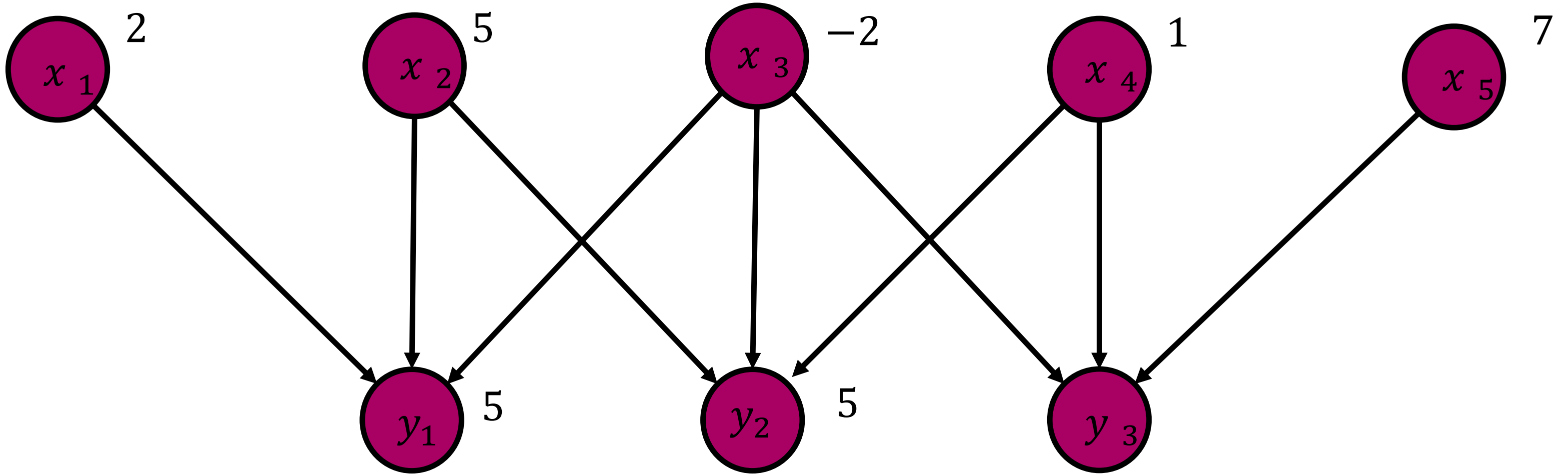
- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

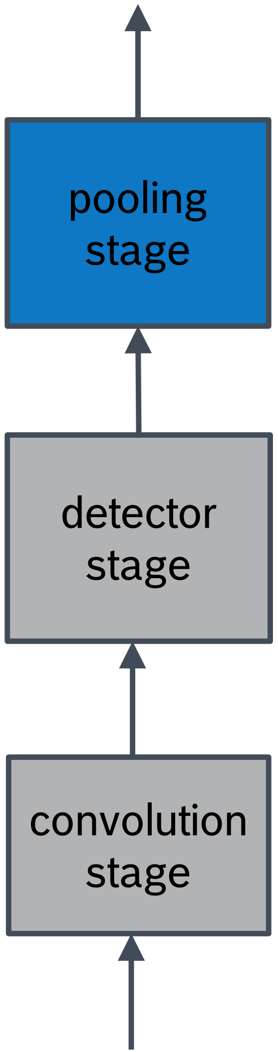




Convolutional Neural Networks
The Pooling Stage (1D case)

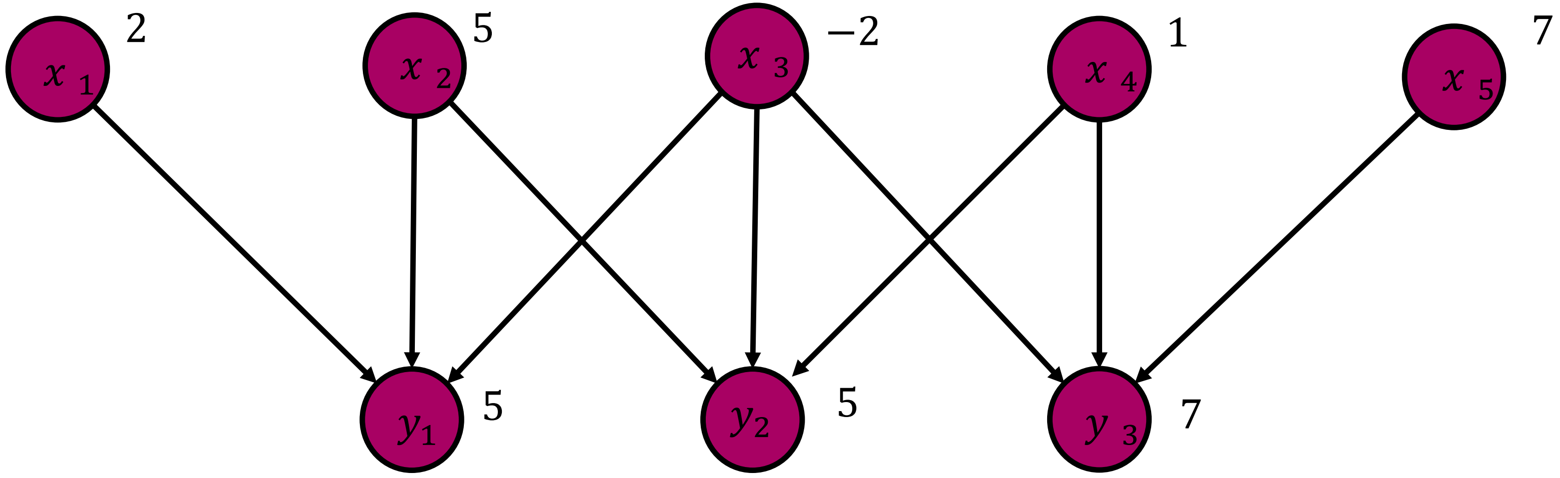
- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

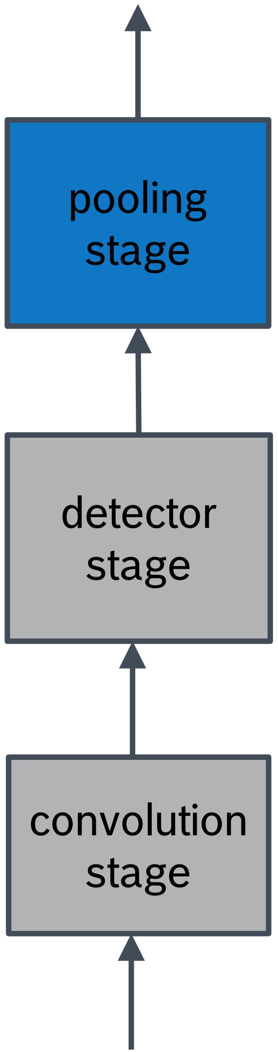




Convolutional Neural Networks
The Pooling Stage (1D case)

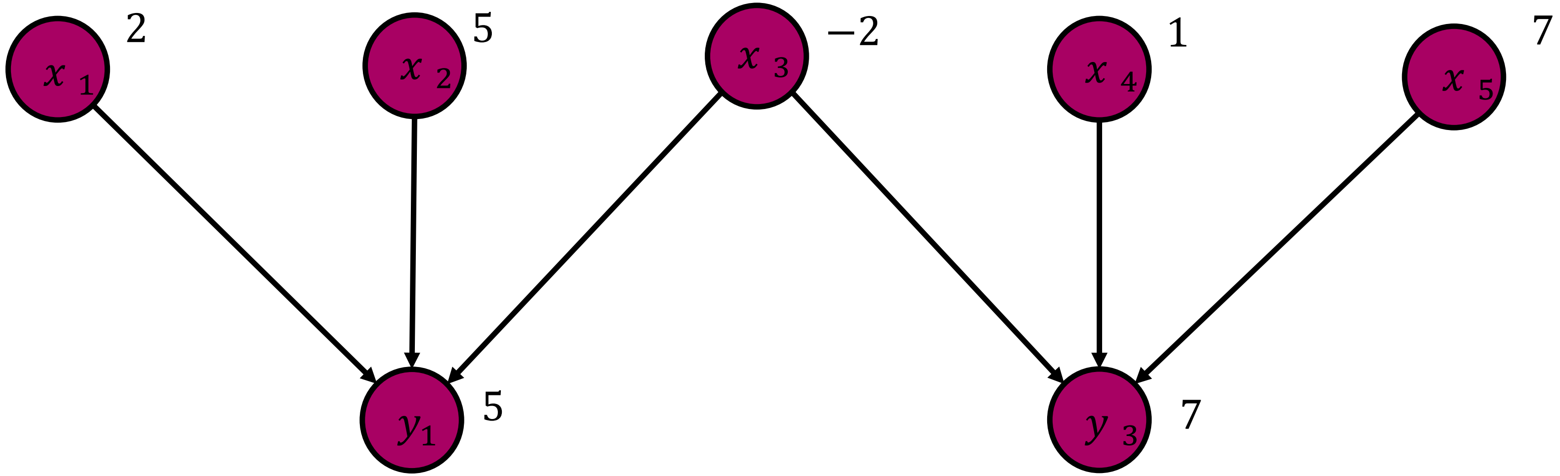
- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

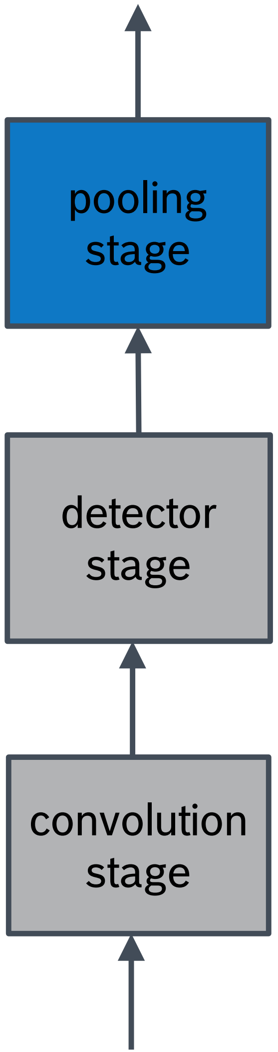




Convolutional Neural Networks
The Pooling Stage (1D case)

- **Idea:**
  - Add invariance to small translations
  - Collect statistics (max, average, etc.) over neighboring features
  - Downsample signal (optionally)
  - Handling inputs of variable size (dynamic pooling regions)

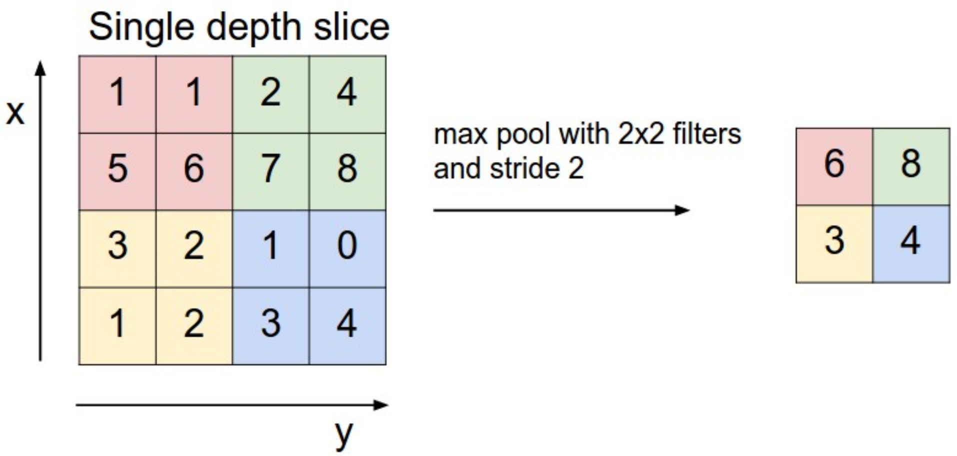


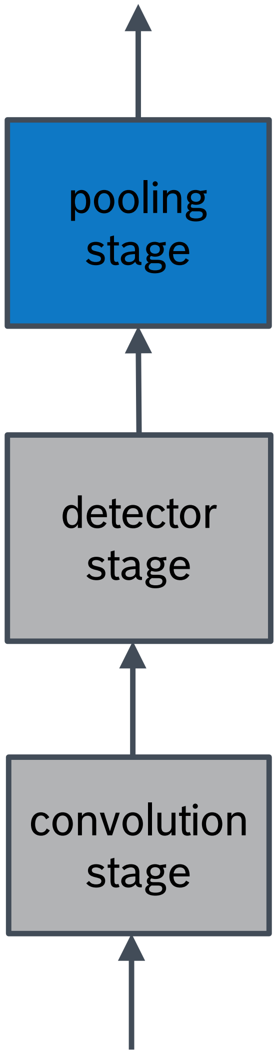


Convolutional Neural Networks
The Pooling Stage (2D case)

- **Operation:**
  - Slide window along width and height (with a given stride)
  - Compute statistics inside the window

- **Meta-Parameters:**
  - **Statistics:**
    - **max**: the winner detection takes all (most common in practice)
    - **average**: smooth out detection noise
  - **window size:** controls spatial invariance
  - **stride:** controls downsampling strength

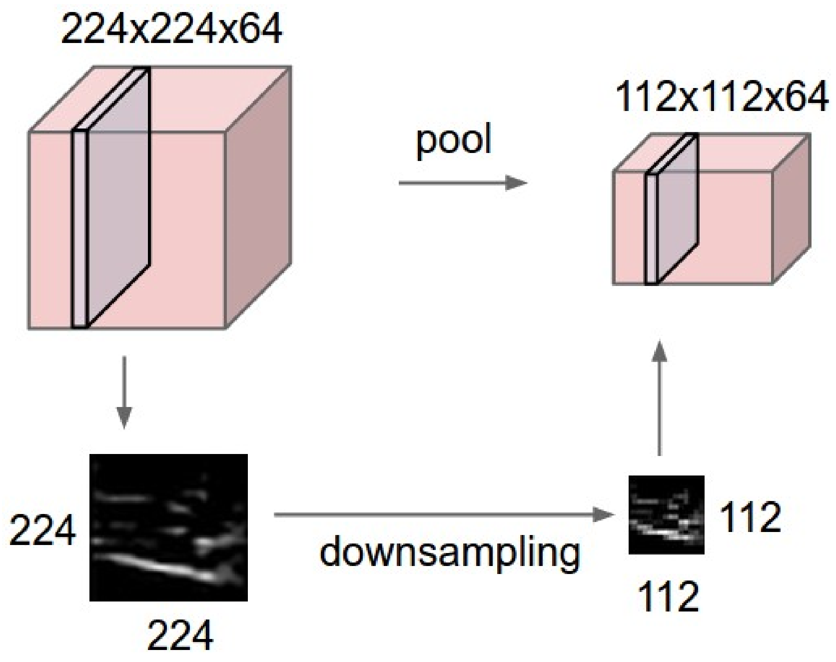




Source: http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks
The Pooling Stage (general case)

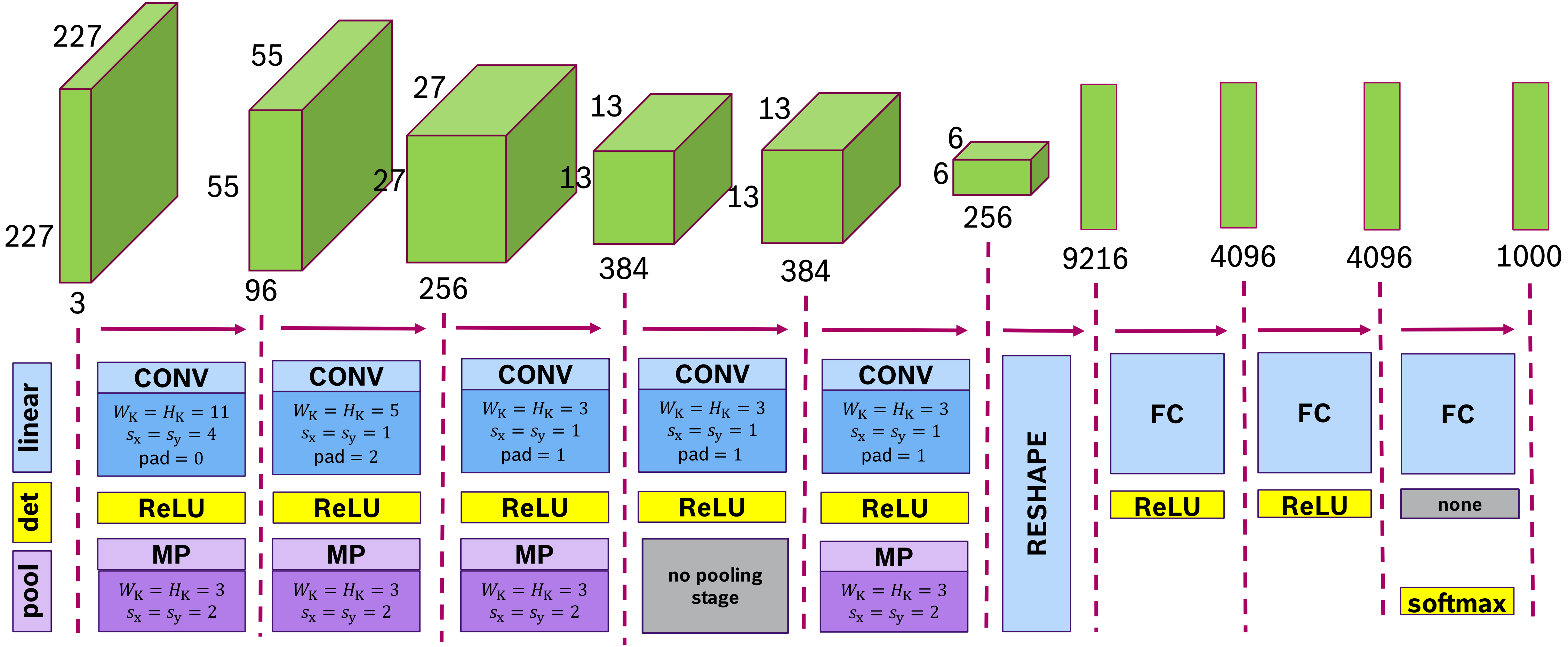
- **Input**: 3D tensor X of size $D_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}$
- **Parameters:** none
- **Meta-parameters:**
  - statistic (max, avg)
  - kernel width/height $(W_{\text{K}}, H_{\text{K}})$
  - horizontal/vertical stride $(s_{\text{x}}, s_{\text{y}})$
- **Output:** 3D tensor Y of size $D_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}$
- **Operation:** perform 2D pooling in each input slice <u>independently</u>



$$y_{j,x,y} = \max_{\substack{0 \le \Delta x < W_{\text{K}} \\ 0 \le \Delta y < H_{\text{K}}}} y_{j, s_{\text{x}} \cdot x + \Delta x, s_{\text{y}} \cdot y + \Delta y}$$

Source: http://cs231n.github.io/convolutional-networks/

Convolutional Neural Networks
Case Study: Image Classification (AlexNet)



BOSCH

Convolutional Neural Networks
The Softmax Layer: Outputting Probability Distributions

- **Issue:**
  - How do we convert the output into a Multinoulli distribution?
  - Condition 1: Entries must be between 0 and 1
  - Condition 2: Must sum up to 1

- **Solutions:**
  - For binary classification: sigmoid for one class (condition 1), implicitly represent the other
  - General case: enforce competition using a **softmax layer**:

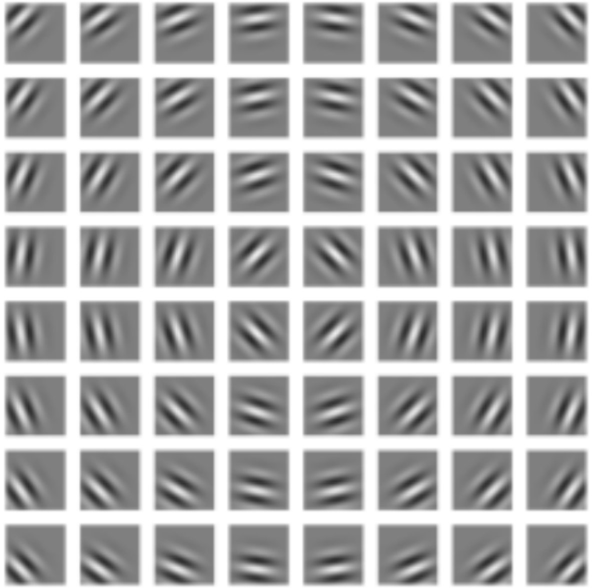
$$y_j = \frac{e^{x_j}}{\sum_{i=1}^{N_{\text{in}}} e^{x_i}}$$

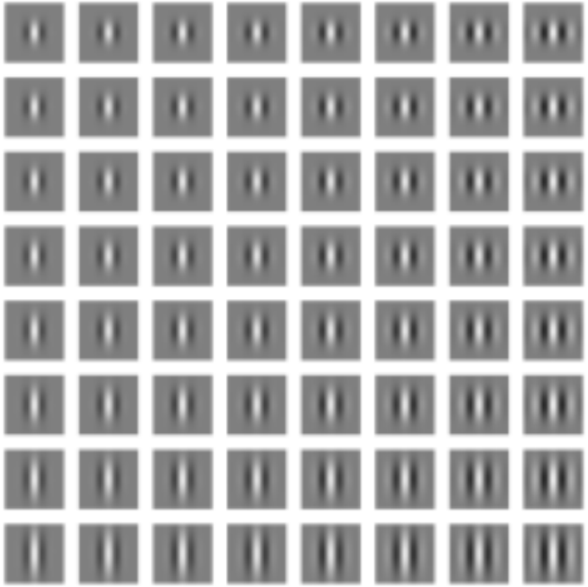
BOSCH

The Path to Deep Learning

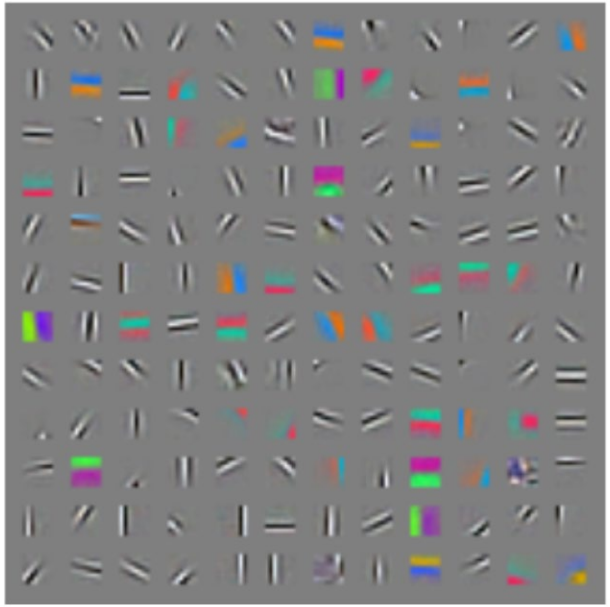
Learned Representations

- Kernels look similar to stimuli for early activation patterns for layers in the brain (V1 area)





Gabor Filters (similar to V1 activators)



Learned kernels in the first convolutional layer

BOSCH

Convolutional Neural Network
Recap

- **Convolution:**
  - Hierarchical local pattern composition (biologically inspired)
  - Uniform equivariant processing across space (and/or time)
  - Can be seen as a strongly regularized fully connected layer
  - Huge reduction in the number of parameters!

- **Spatial pooling:**
  - Add spatial invariance by inserting pooling layers
  - Reduce feature map sizes (computational advantage)

- **Few additional layers (Connectivist Approach):**
  - FC: may be missing altogether, see Fully Convolutional Neural Networks (FCNs)
  - Softmax: convert outputs into probability distributions

BOSCH

PERSPECTIVES

Perspectives
Active Research Areas

- The **statistical** challenge: labeled data is expensive
  - Supervised transfer learning: fueled the breakthrough (ImageNet)
  - Unsupervised pre-training: not that successful! Why?
  - Unsupervised/weakly supervised learning
  - Active learning

- The **computational** challenge: inference is expensive
  - Model compression/speedup (weight/response quantization)
  - Implicit generative models (*e.g.* GANs)

- The **optimization** challenge: slow/lack of convergence needs tuning
  - More or less the same algorithms as 3 decades ago! (SGD family)
  - Second order optimization (*e.g.* Hessian-Free optimization)

REFERENCES

Deep Learning for Computer Vision
References

- [1] Goodfellow, I. and Bengio, Y. and Courville, A., "*Deep Learning*", MIT Press, 2016, http://www.deeplearningbook.org/
- [2] Bishop, C. M. "Pattern Recognition and Machine Learning", Springer, 2006
- [3] Murphy, K.P. "*Machine Learning: a Probabilistic Perspective*", MIT Press, 2012
- [4] *Fei-Fei, Karpathy & Johnson, Lecture Notes, 2016,* http://cs231n.stanford.edu/slides/2017/