Practical AI

Introduction to Neural Networks

Matthias Baumgartner
Abraham Bernstein

DDIS
University of Zürich

April, 2019

University of Zurich[UZH]

Outline

About this lecture

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge:
Image Style Transfer Using Convolutional Neural Networks

- Brief introduction to neural networks along the example of Image Style Transfer.
- Harish Narayanan [1], Goodfellow et al.[2]

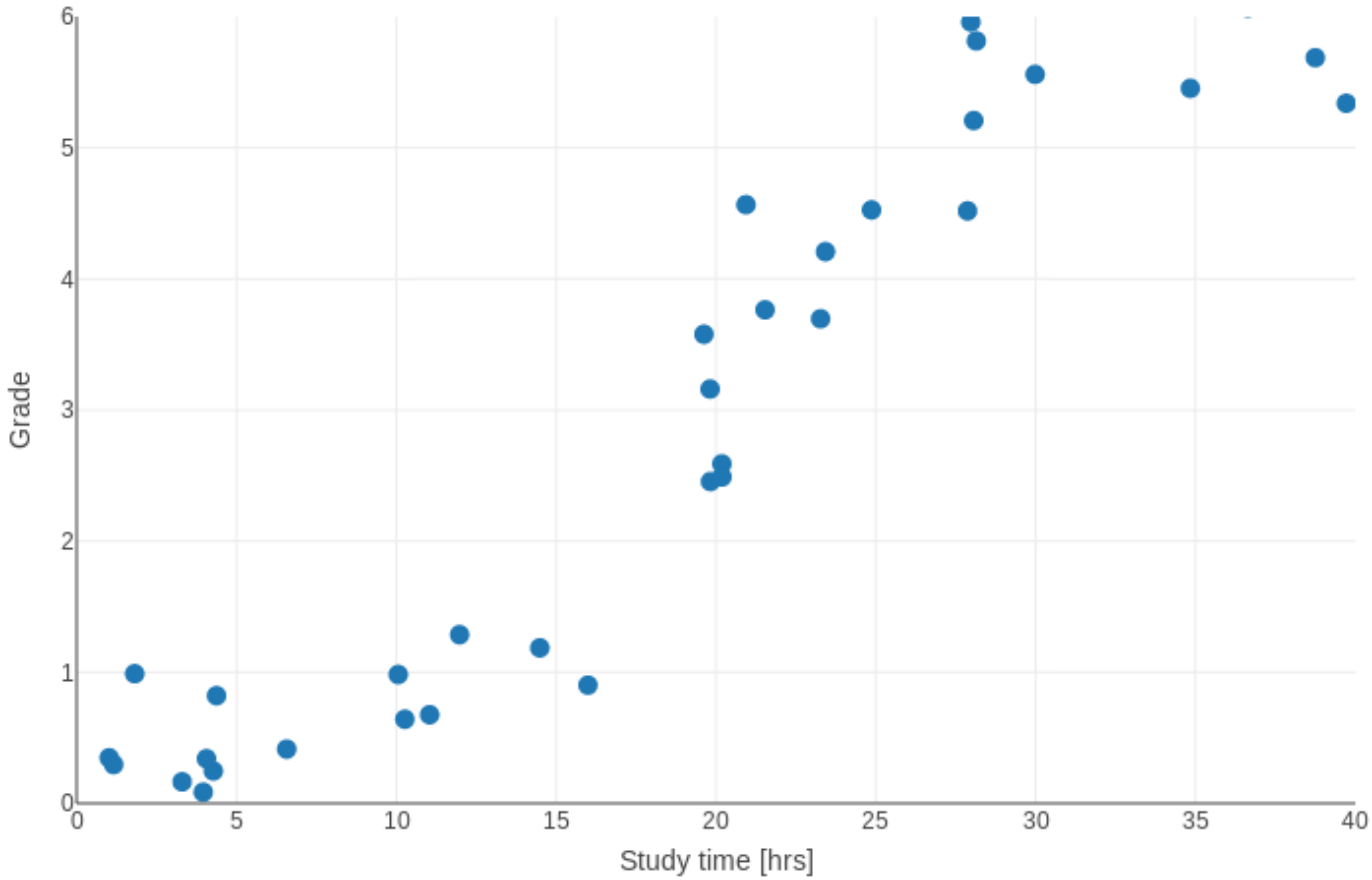
[1] https://harishnarayanan.org/writing/artistic-style-transfer/
[2] https://www.deeplearningbook.org/

Introduction

Introduction

Machine Learning

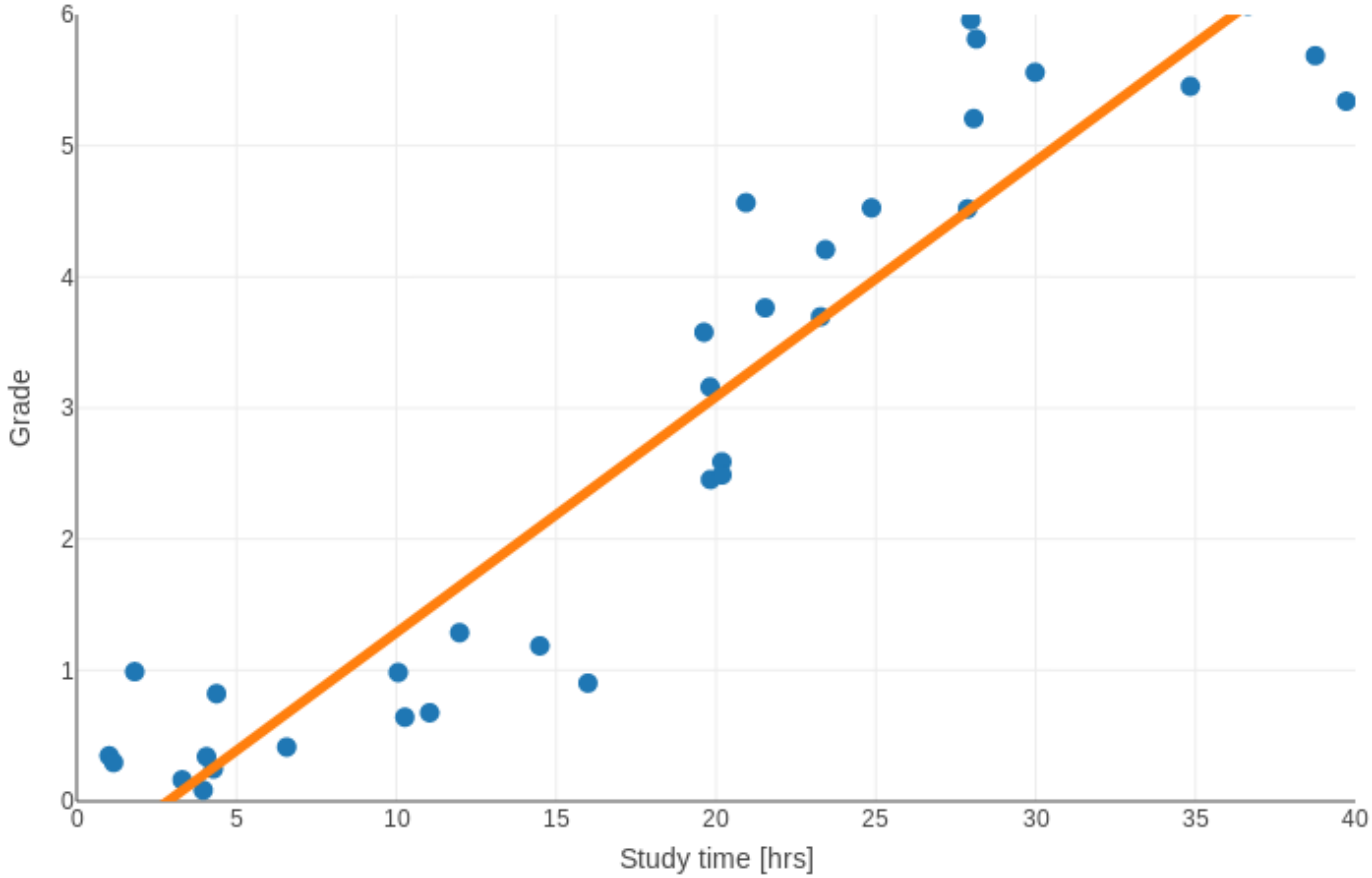
- Explain patterns in data by a model such that we're able to predict unseen data as precise as possible.



Introduction

Machine Learning

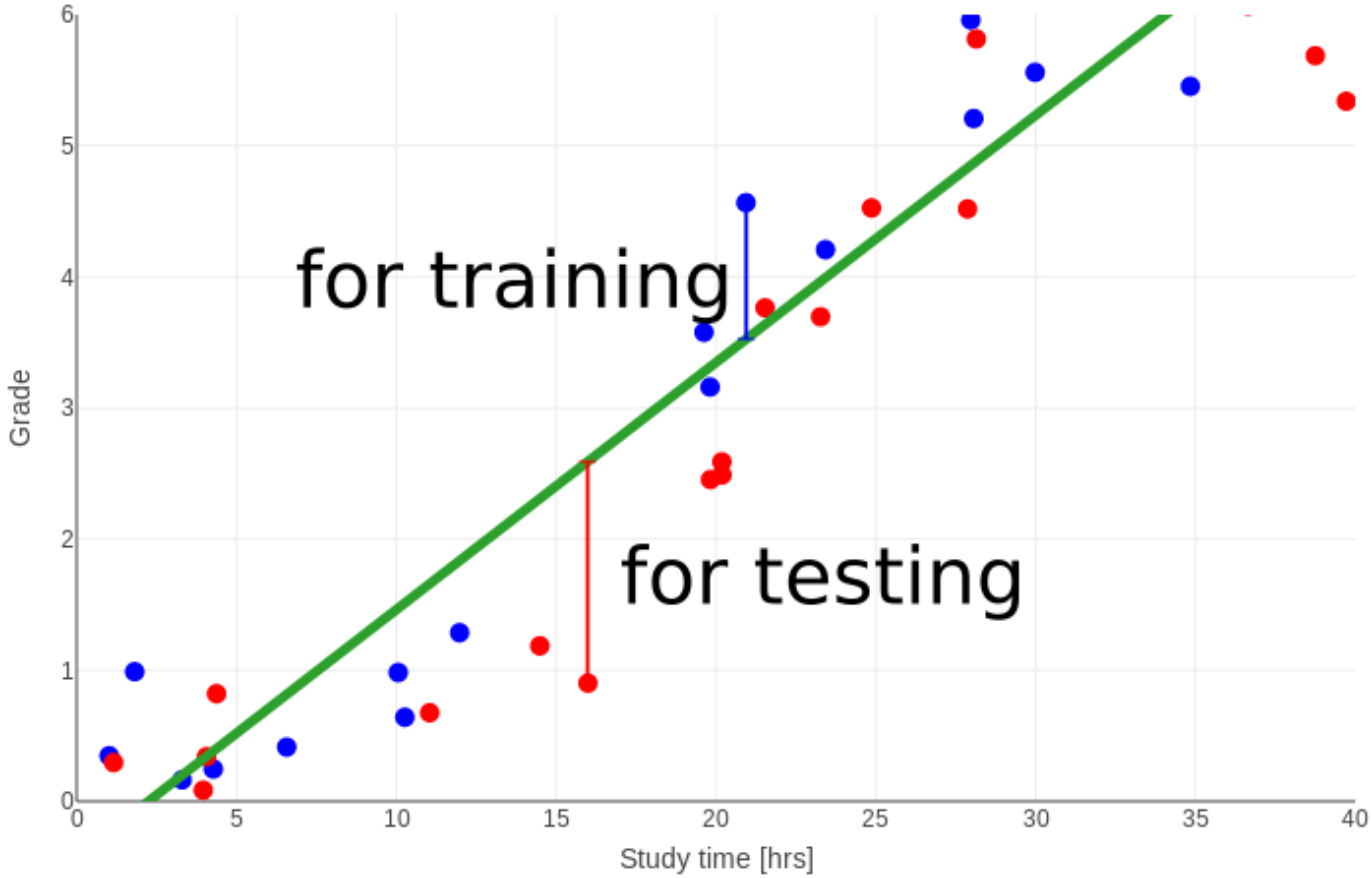
- **Explain patterns in data by a model** such that we're able to predict unseen data as precise as possible.



Introduction

Machine Learning

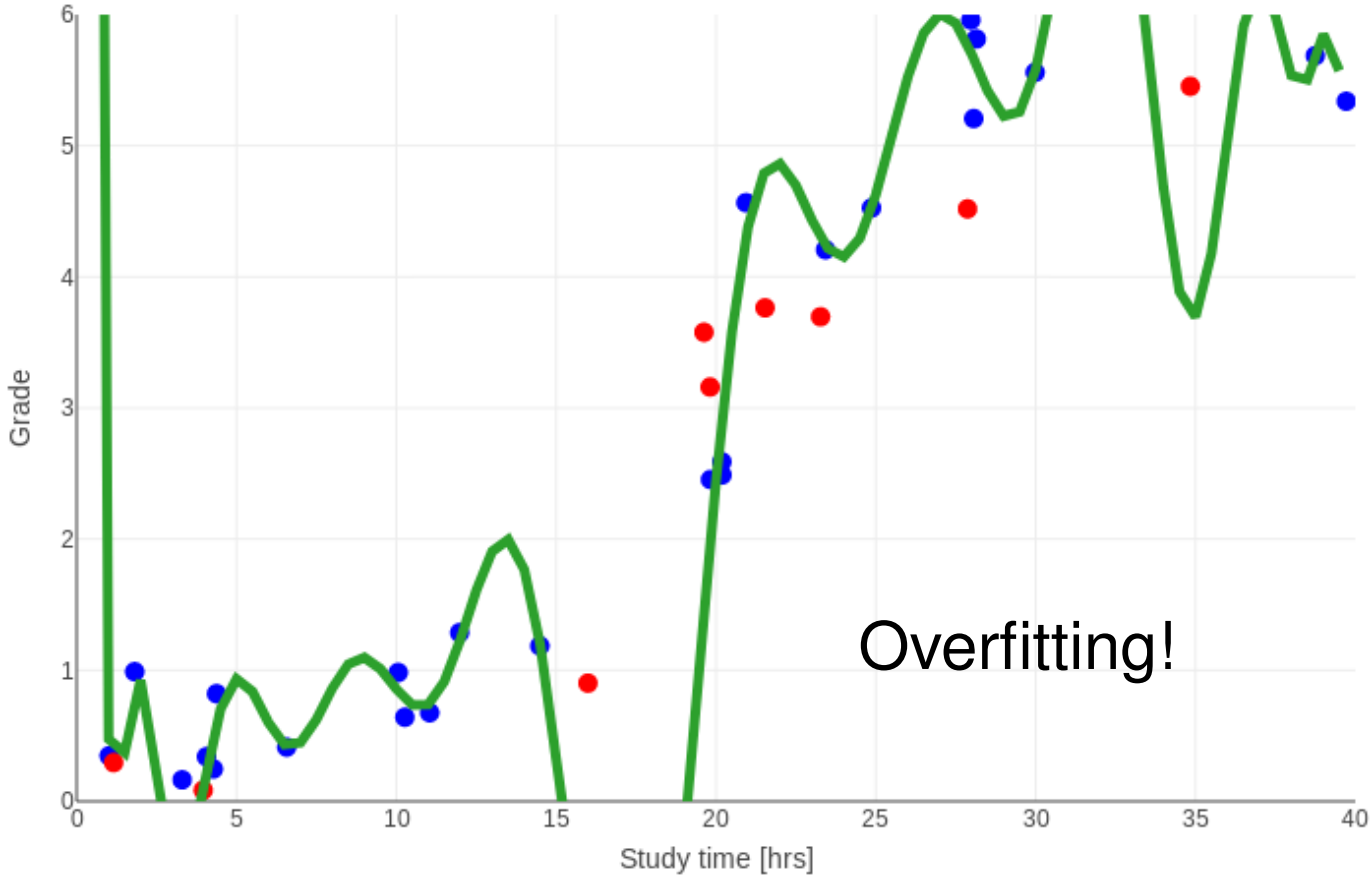
- Explain patterns in data by a model such that we're able to **predict unseen data as precise as possible**.



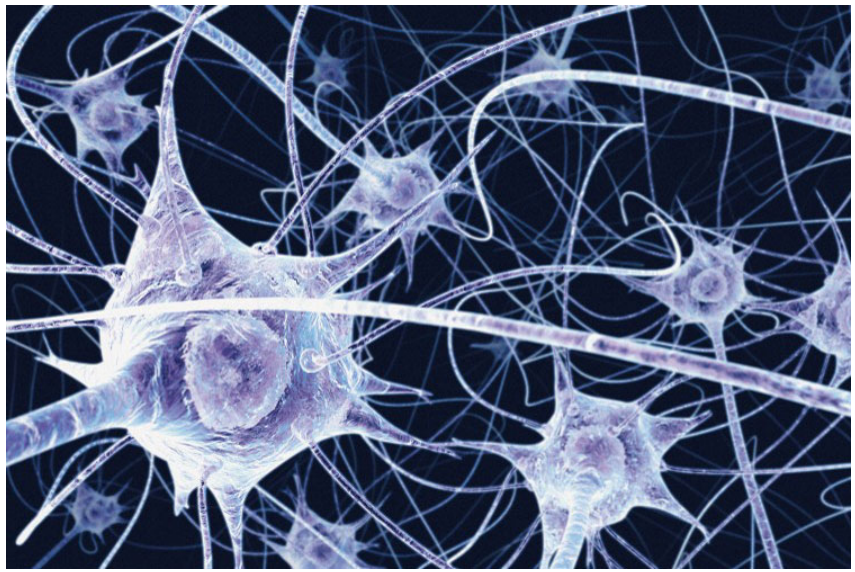
Introduction

Machine Learning

- Explain patterns in data by a model such that we're able to **predict unseen data as precise as possible**.



Introduction



Amy Breau, `https://indianapublicmedia.org/amomentofscience/lose-neurons`

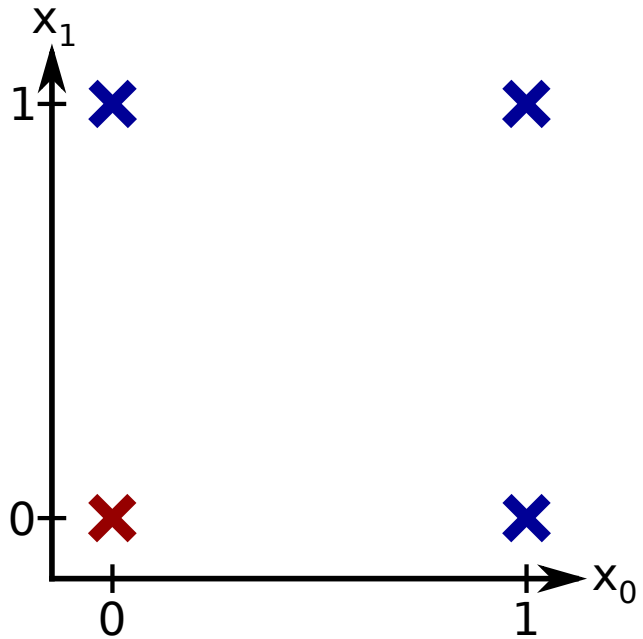
- Black-box function approximation:
  Find function f which explains your data.
- Loosely inspired by brain functionality.

Introduction

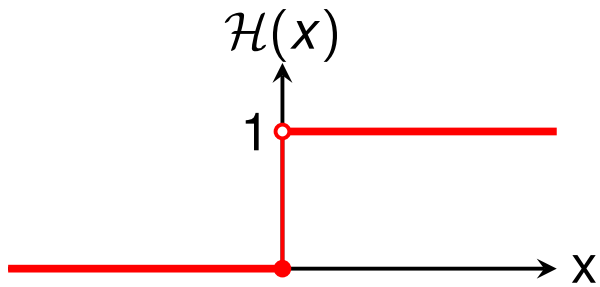
Logical OR

f : with inputs $x_0, x_1 : \{0,1\}^2 \mapsto \{0,1\}$

$$f : \quad x_0 \vee x_1$$
$$f : \quad x_0 + x_1 > 0.5$$
$$f : \quad \mathcal{H}_{0.5}(x_0 + x_1)$$



$$\mathcal{H}_\theta(x) = \begin{cases} 1, x > \theta \\ 0, \text{otherwise} \end{cases}$$

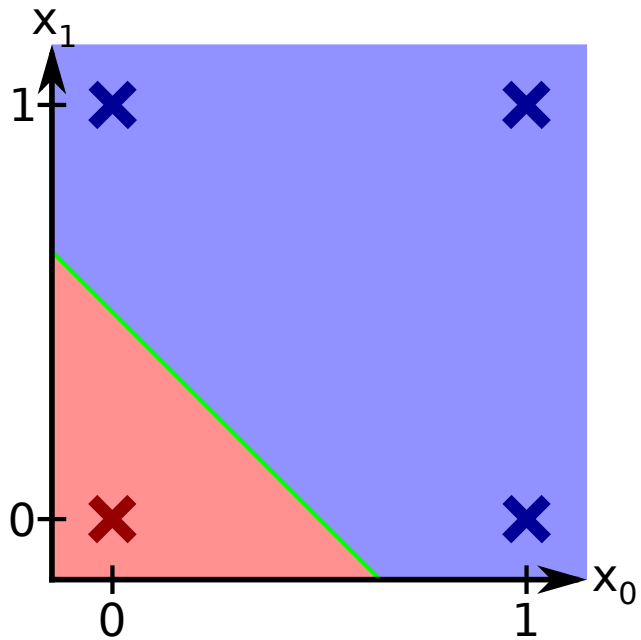


Introduction

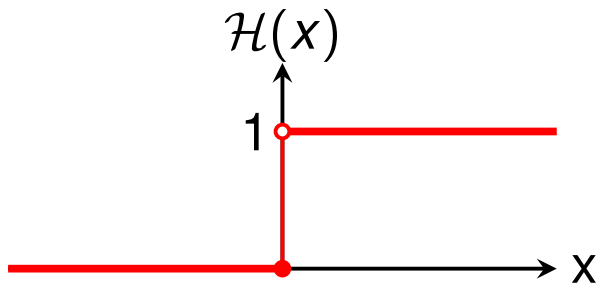
Logical OR

f : with inputs $x_0, x_1 : \{0,1\}^2 \mapsto \{0,1\}$

$$f : \quad x_0 \vee x_1$$
$$f : \quad x_0 + x_1 > 0.5$$
$$f : \quad \mathcal{H}_{0.5}(x_0 + x_1)$$



$$\mathcal{H}_\theta(x) = \begin{cases} 1, x > \theta \\ 0, \text{otherwise} \end{cases}$$

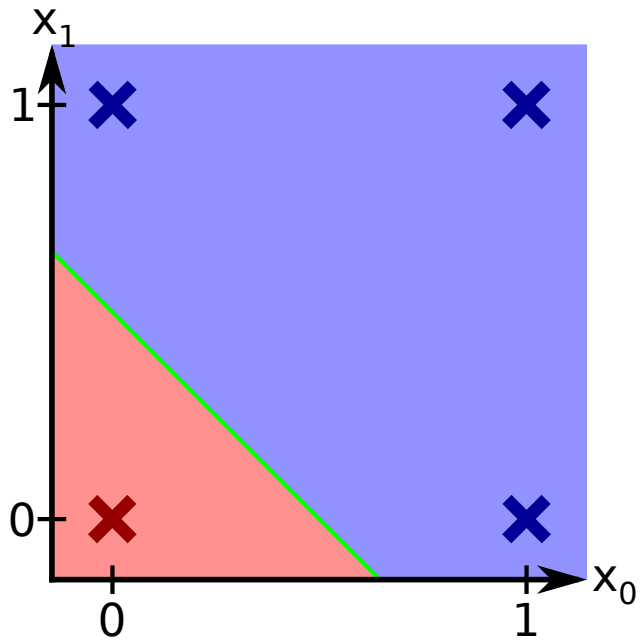


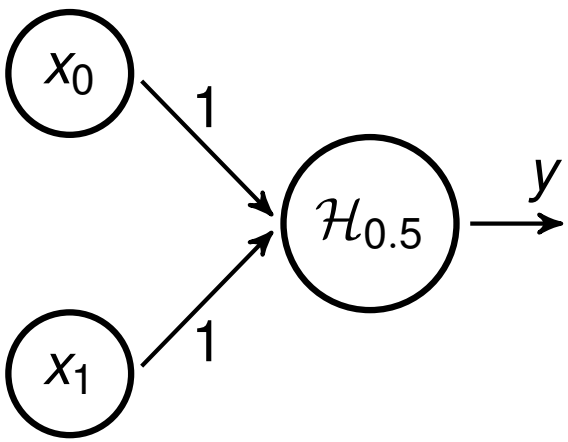
Introduction

Logical OR

f : with inputs $x_0, x_1 : \{0,1\}^2 \mapsto \{0,1\}$

$$
\begin{aligned}
f: &\quad x_0 \vee x_1 \\
f: &\quad x_0 + x_1 > 0.5 \\
f: &\quad \mathcal{H}_{0.5}(x_0 + x_1) \\
f: &\quad \mathcal{H}_{0.5}(1x_0 + 1x_1)
\end{aligned}
$$





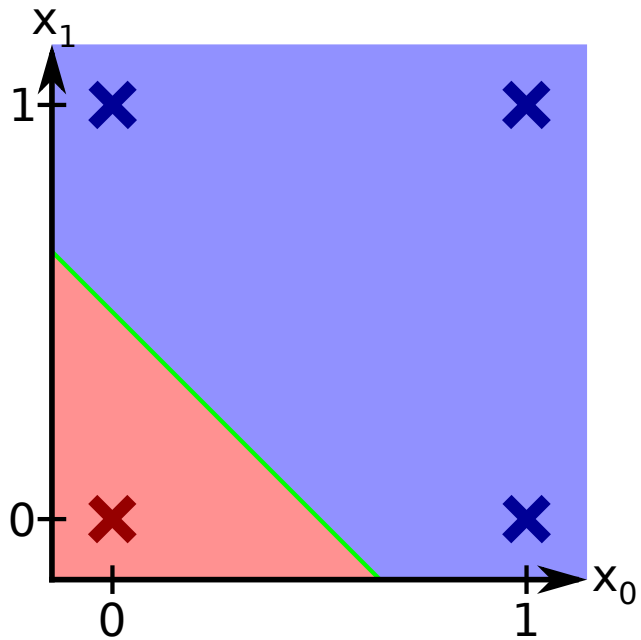
$$
\begin{aligned}
y &= \mathcal{H}_{0.5}\left(1x_0 + 1x_1\right) \\
&= \mathcal{H}_{0.5}\left(\sum_i w_i x_i\right)
\end{aligned}
$$

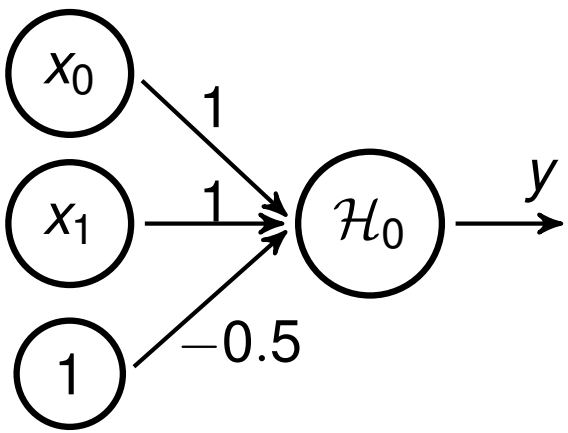
Introduction

Logical OR

f : with inputs $x_0, x_1 : \{0,1\}^2 \mapsto \{0,1\}$

$$
\begin{aligned}
f: &\quad x_0 \vee x_1 \\
f: &\quad x_0 + x_1 > 0.5 \\
f: &\quad \mathcal{H}_{0.5}(x_0 + x_1) \\
f: &\quad \mathcal{H}_{0.5}(1x_0 + 1x_1) \\
f: &\quad \mathcal{H}_0(1x_0 + 1x_1 - 0.5)
\end{aligned}
$$





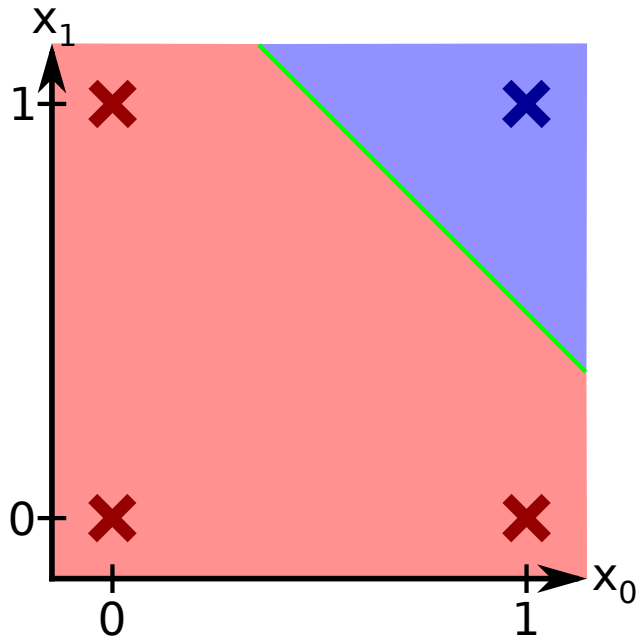
$$
\begin{aligned}
y &= \mathcal{H}_0\left(1x_0 + 1x_1 - 0.5\right) \\
&= \mathcal{H}_0\left(-0.5 + \sum_i w_i x_i\right)
\end{aligned}
$$

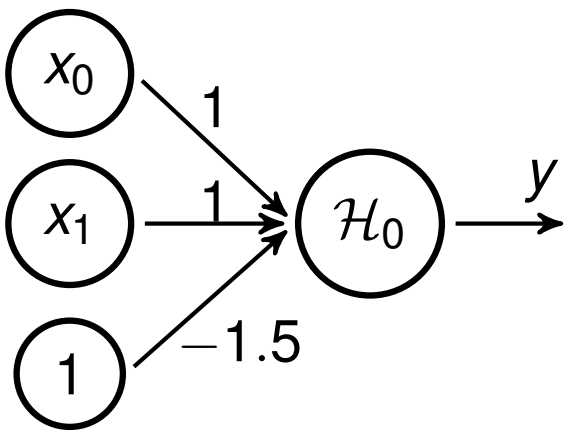
Introduction

Logical AND

f : with inputs $x_0, x_1 : \{0,1\}^2 \mapsto \{0,1\}$

$$
\begin{aligned}
f: &\quad x_0 \wedge x_1 \\
f: &\quad x_0 + x_1 > 1.5 \\
f: &\quad \mathcal{H}_0(1x_0 + 1x_1 - 1.5)
\end{aligned}
$$

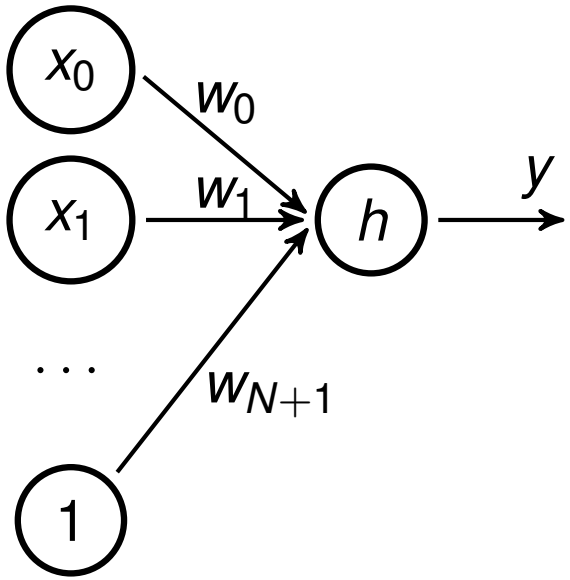




$$
\begin{aligned}
y &= \mathcal{H}_0\left(1x_0 + 1x_1 - 1.5\right) \\
&= \mathcal{H}_0\left(-1.5 + \sum_i w_i x_i\right)
\end{aligned}
$$

Introduction

Perceptron



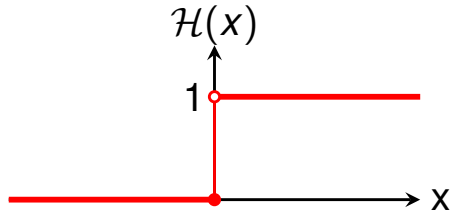
$$
\begin{aligned}
y(\mathbf{x}) &= h\left(w_{N+1} + \sum_{i=0}^{N} w_i x_i\right) \\
&= h\left([\mathbf{x}|1]^T \mathbf{w}\right)
\end{aligned}
$$

- $\mathbf{x}$ Input
- $\mathbf{w}$ Weights
- h Activation function
- y Output

Introduction

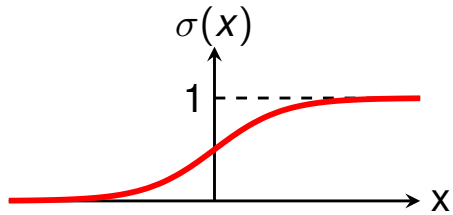
Activation functions

Heavyside



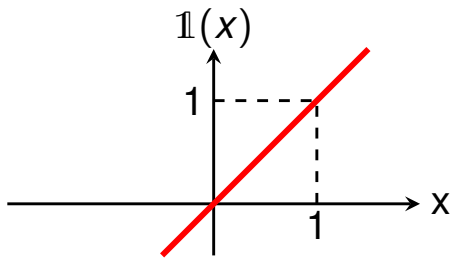
$$\mathcal{H}(x) = \begin{cases} 1, x > 0 \\ 0, \text{otherwise} \end{cases}$$

Sigmoid



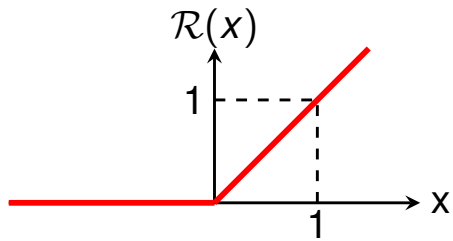
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Identity



$$\mathbb{1}(x) = x$$

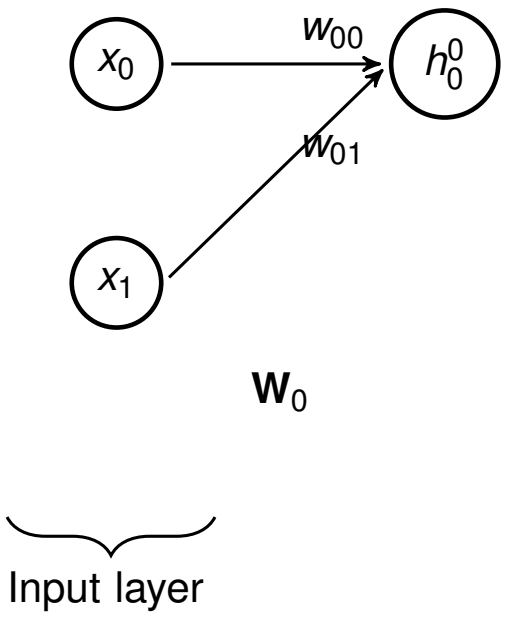
ReLu



$$\mathcal{R}(x) = \begin{cases} 0, x \leq 0 \\ x, \text{otherwise} \end{cases}$$

Introduction

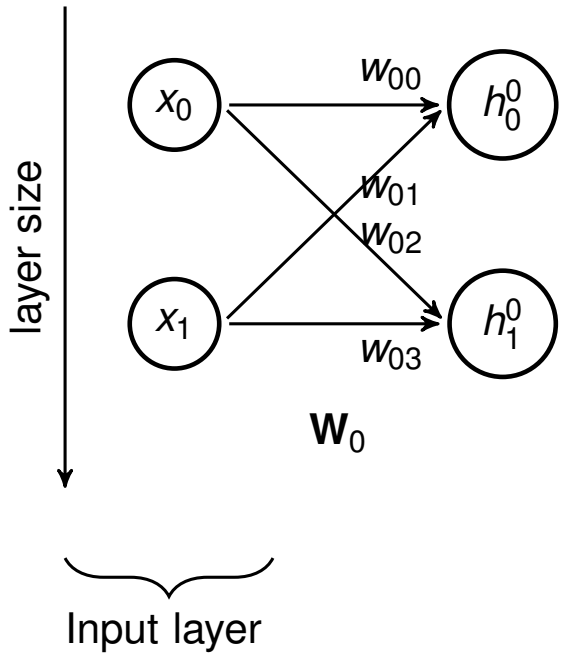
Neural Network



- Specify: activation function

Introduction

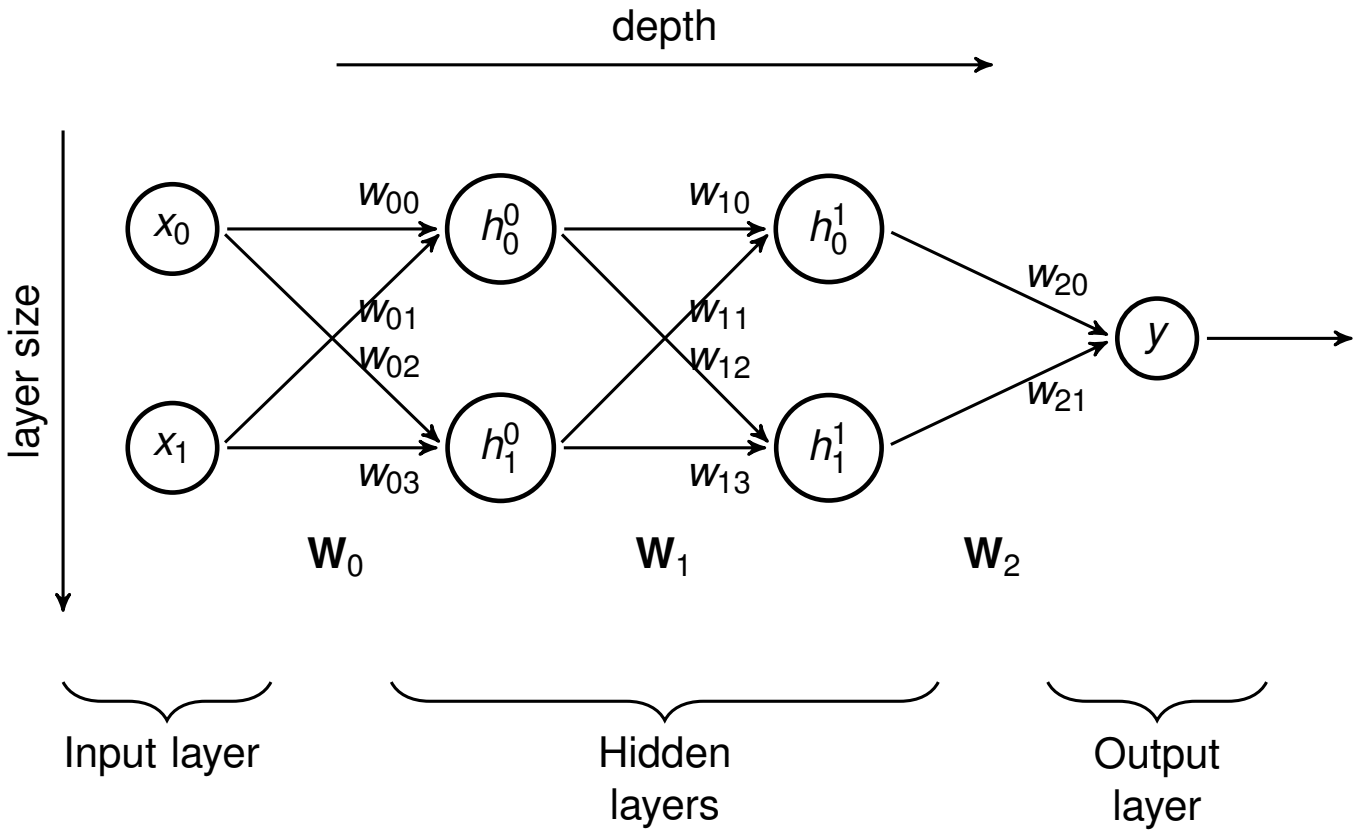
Neural Network



- Specify: activation function, layer size

Introduction

Neural Network



- Specify: activation function, layer size, depth, connectivity

Introduction

Universal Approximation

Goodfellow et al: "the universal approximation theorem (Hornik et al., 1989; Cybenko, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function [...] can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units."

- **Borel measurable:** includes all continuous functions on a closed and bounded subset of $\mathbb{R}^n$
- **Squashing function:** Non-decreasing function $\Psi : \mathbb{R} \mapsto [0, 1]$, such that $\lim_{\lambda \to \infty} \Psi(\lambda) = 1$ and $\lim_{\lambda \to -\infty} \Psi(\lambda) = 0$.

Introduction

Universal Approximation - Implications

- A NN can approximate all practically relevant functions
- A wide class of activation functions are admissible
- No statement about the hidden layer size
- Doesn't imply that the function can be learned!

Introduction

Summary

- A neural network consists of:
  - Layers of neurons. At least input and output, usually also (at least one) hidden layer.
  - Activation function per neuron
  - Connection weights between neurons of consecutive layers
  - Typically a bias (a constant input of 1)
- Neural networks can be used for classification as well as regression
- Neural networks can in principle approximate any function. However, there's no guarantee that training the network finds the best fit.

Training

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?
  Network weights **W**

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?
  Network weights **W**
- What is the learning goal?

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?
  Network weights $\mathbf{W}$
- What is the learning goal?

$$J_{MSE}(\mathbf{W};\mathbf{x},\mathbf{d}) = \frac{1}{N}\sum_{i=0}^{N}(d_i - y(x_i;\mathbf{W}))^2$$

$$J_{CE}(\mathbf{W};\mathbf{x},\mathbf{d}) = -\frac{1}{N}\sum_{i=0}^{N} d_i \ln y(x_i;\mathbf{W}) + (1-d_i)\ln\left(1 - y(x_i;\mathbf{W})\right)$$

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?
  Network weights $\mathbf{W}$
- What is the learning goal?

$$J_{MSE}(\mathbf{W};\mathbf{x},\mathbf{d}) = \frac{1}{N}\sum_{i=0}^{N}(d_i - y(x_i;\mathbf{W}))^2$$

$$J_{CE}(\mathbf{W};\mathbf{x},\mathbf{d}) = -\frac{1}{N}\sum_{i=0}^{N} d_i \ln y(x_i;\mathbf{W}) + (1-d_i)\ln(1-y(x_i;\mathbf{W}))$$

- How is the network trained?

Training

Introduction

- Given some data (input and desired output) and a neural network (activation functions, layer sizes, depth, connectivity).
- What is learned through training?
  Network weights $\mathbf{W}$
- What is the learning goal?

$$J_{MSE}(\mathbf{W};\mathbf{x},\mathbf{d}) = \frac{1}{N}\sum_{i=0}^{N}(d_i - y(x_i;\mathbf{W}))^2$$

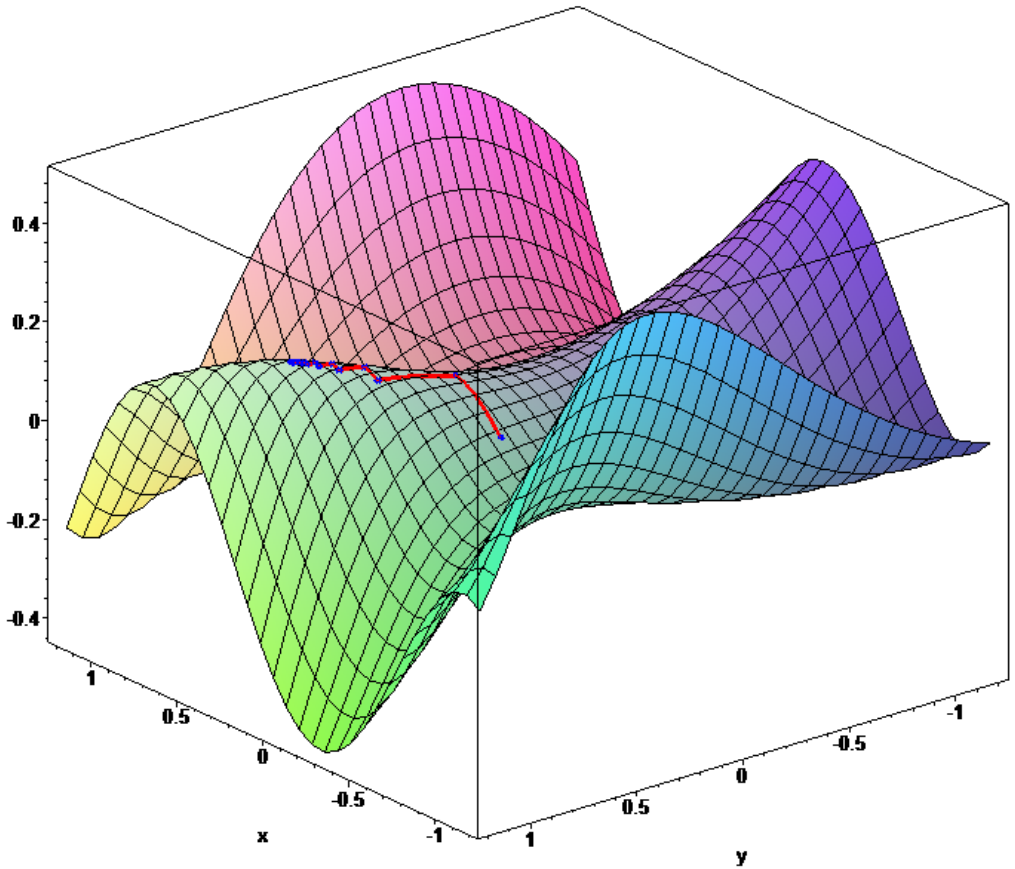
$$J_{CE}(\mathbf{W};\mathbf{x},\mathbf{d}) = -\frac{1}{N}\sum_{i=0}^{N} d_i \ln y(x_i;\mathbf{W}) + (1-d_i)\ln\left(1 - y(x_i;\mathbf{W})\right)$$

- How is the network trained?
  Backpropagation and *gradient descent*

Training

Stochastic Gradient Descent

Parameter update: $\mathbf{W}_{n+1} \leftarrow \mathbf{W}_n - \alpha \nabla J(\mathbf{W}_n; \mathbf{x}, \mathbf{d})$

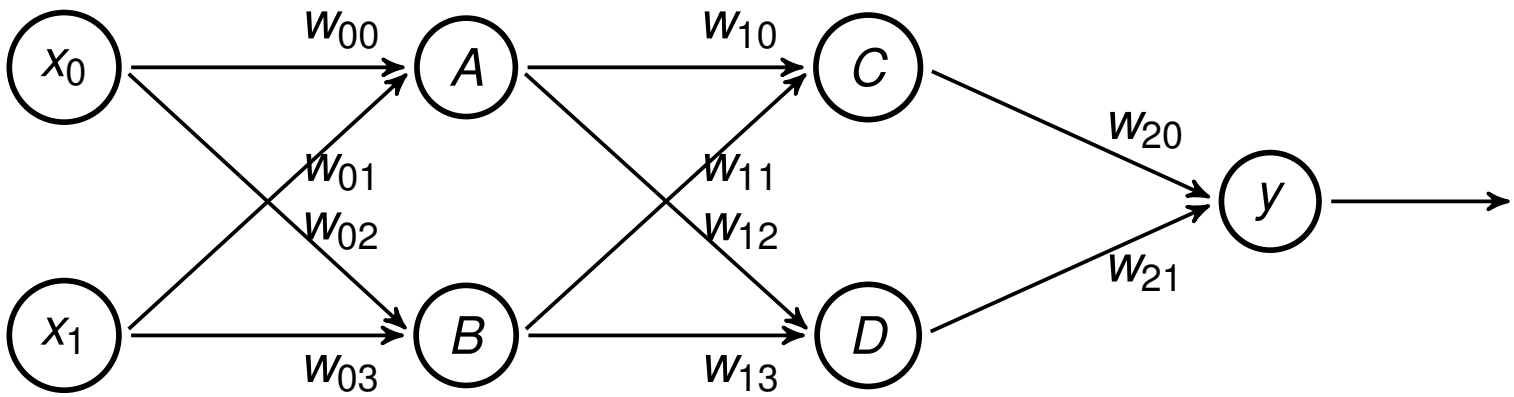


- $\mathbf{W}$ Parameters
- α Learning rate
- J Loss function
- $\mathbf{x}$ Batch samples
- ∇ Gradient $\nabla = \left[\frac{\partial J}{\partial W_0} \quad \frac{\partial J}{\partial W_1} \quad \cdots \quad \frac{\partial J}{\partial W_n} \right]^T$

Joris Gillis, `https://en.wikipedia.org/wiki/File:Gradient_ascent_(surface).png`

Training

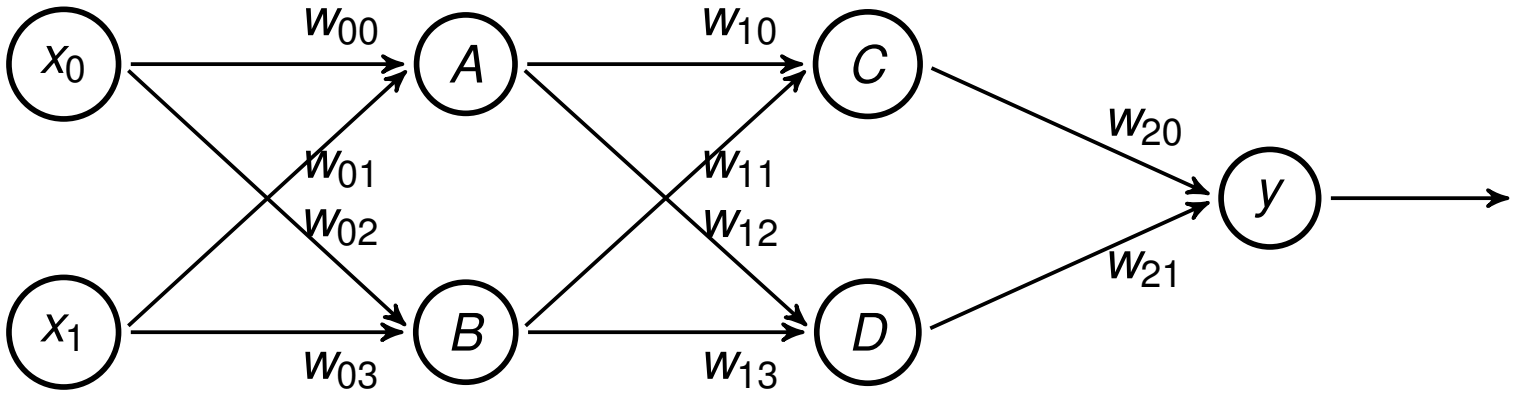
Backpropagation



- A, B, C, D are sigmoid activated: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Output y is identity activated: $\mathbb{1}(x) = x$
- MSE w.r.t. target d: $J \sim \frac{1}{2}(d_i - y_i)^2$

Training

Backpropagation

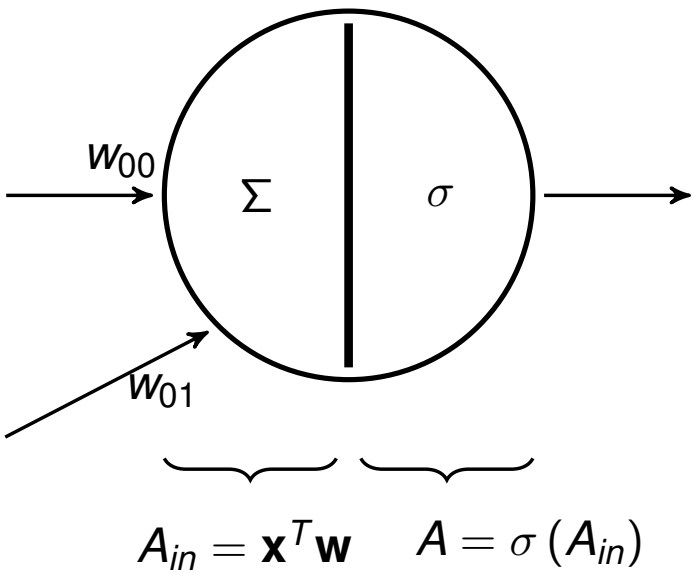


Forward pass:

- Set inputs
- Propagate values through the network.
- Store values of output y and hidden nodes A, B, C, D.

Training

Backpropagation - Preliminaries

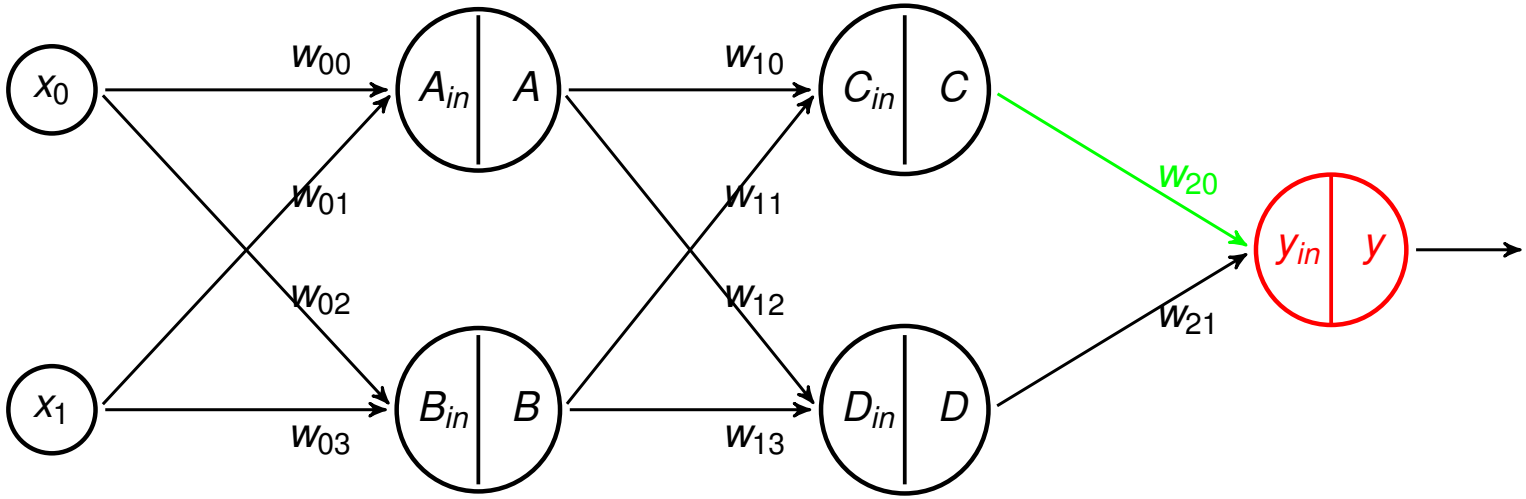


Recall the chain rule from calculus:

$$\begin{aligned} y &= f(g(x)) \\ u &:= g(x) \\ \frac{\partial y}{\partial x} &= \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} \end{aligned}$$

Training

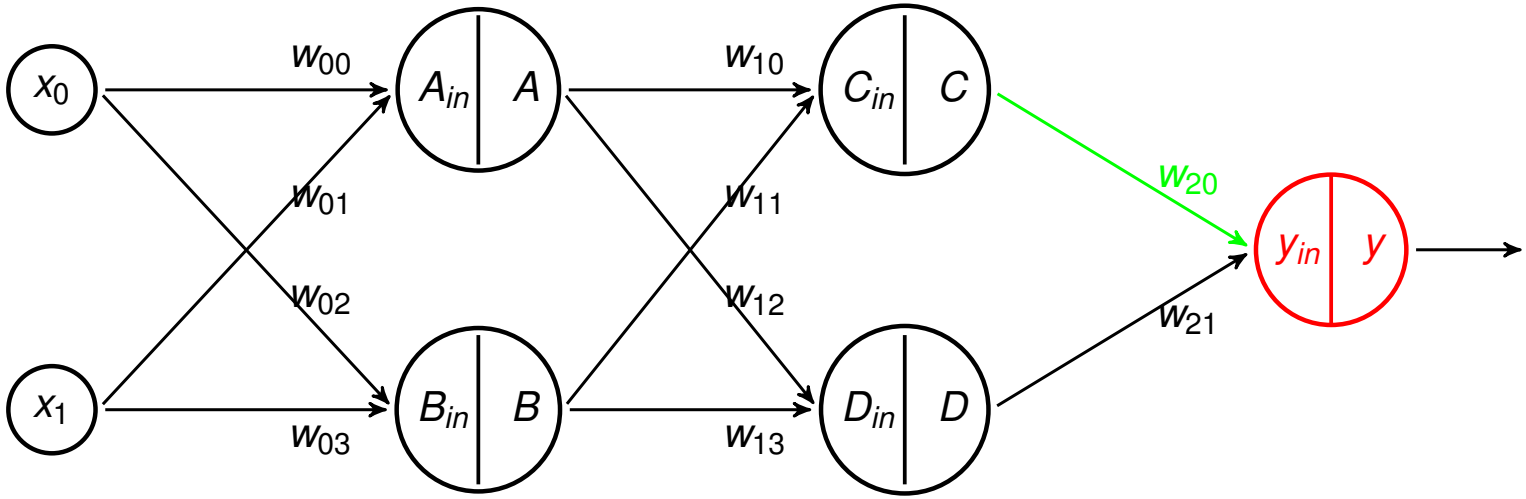
Backpropagation



$$
\begin{aligned}
y &= \mathbb{1}\left(w_{20}C + w_{21}D\right) \\
J &= \frac{1}{2}(d-y)^2 \\
&= \frac{1}{2}\left(d - \mathbb{1}\left(w_{20}C + w_{21}D\right)\right)^2
\end{aligned}
$$

Training

Backpropagation



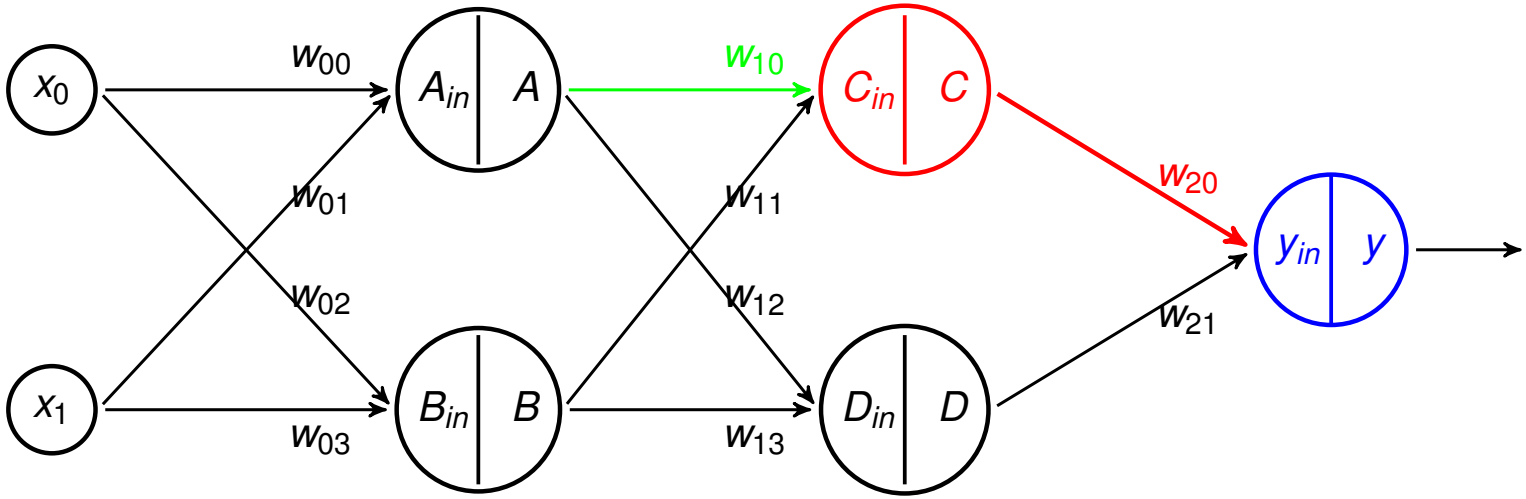
$$
\begin{aligned}
y &= \mathbb{1}\left(w_{20}C + w_{21}D\right) \\
J &= \frac{1}{2}(d-y)^2 \\
&= \frac{1}{2}\left(d - \mathbb{1}\left(w_{20}C + w_{21}D\right)\right)^2
\end{aligned}
$$

$$
\frac{\partial J}{\partial w_{20}} = \frac{\partial J}{\partial y} \underbrace{\frac{\partial y}{\partial y_{in}}}_{1} \underbrace{\frac{\partial y_{in}}{\partial w_{20}}}_{C}
$$

$$
\frac{\partial J}{\partial y} = \underbrace{\frac{\partial}{\partial y}\frac{1}{2}(d-y)^2}_{y-d}
$$

Training

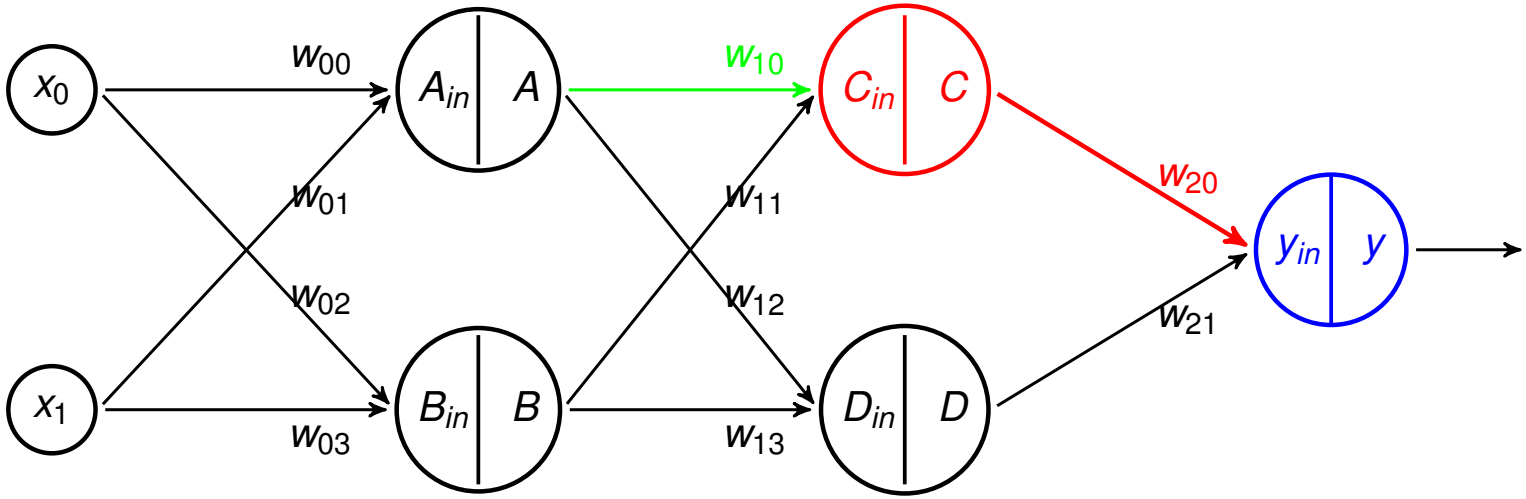
Backpropagation



$$
\begin{aligned}
C &= \sigma\left(w_{10}A + w_{12}B\right) \\
y &= \mathbb{1}\left(w_{20}C + w_{21}D\right) \\
J &= \frac{1}{2}(d-y)^2
\end{aligned}
$$

Training

Backpropagation



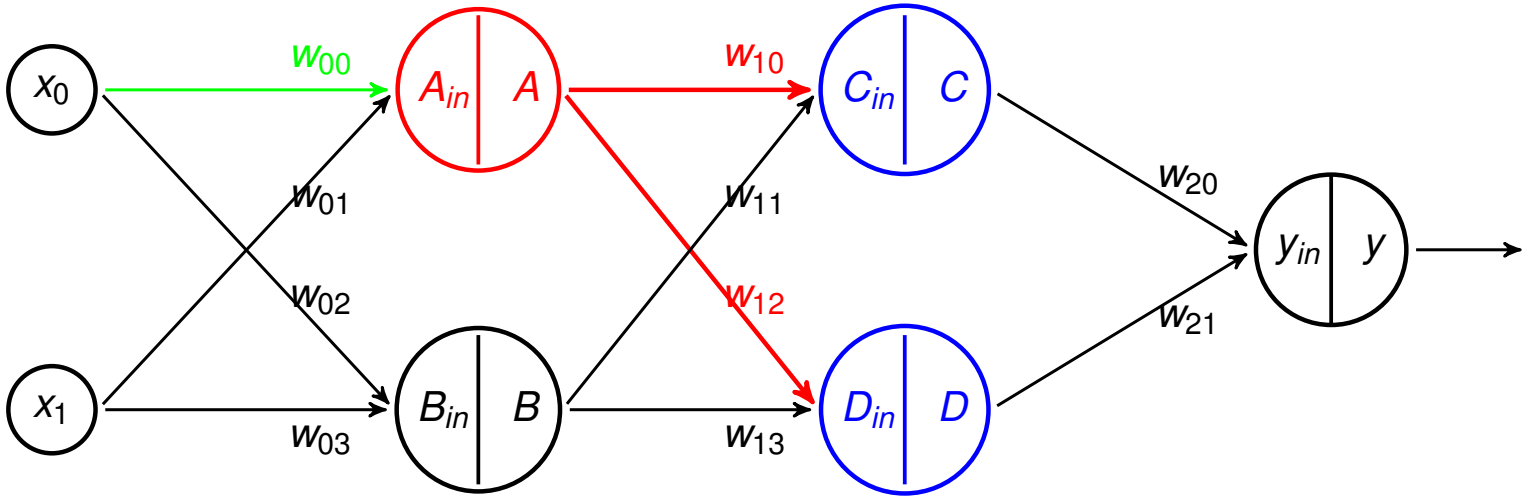
$$
\begin{aligned}
C &= \sigma\left(w_{10}A + w_{12}B\right) \\
y &= \mathbb{1}\left(w_{20}C + w_{21}D\right) \\
J &= \frac{1}{2}(d-y)^2
\end{aligned}
$$

$$
\frac{\partial J}{\partial w_{10}} = \frac{\partial J}{\partial C}\underbrace{\frac{\partial C}{\partial C_{in}}}_{\sigma'(C_{in})}\underbrace{\frac{\partial C_{in}}{\partial w_{10}}}_{A}
$$

$$
\frac{\partial J}{\partial C} = \underbrace{\frac{\partial J}{\partial y}}_{\text{previously computed}}\underbrace{\frac{\partial y}{\partial y_{in}}}_{1}\underbrace{\frac{\partial y_{in}}{\partial C}}_{w_{20}}
$$

Training

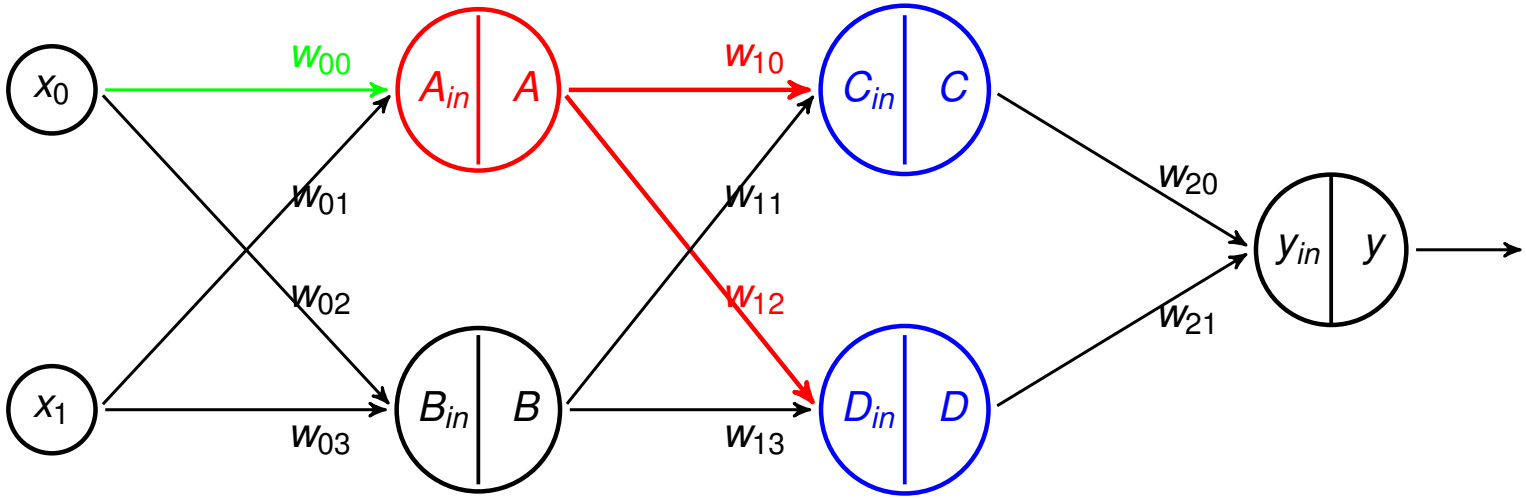
Backpropagation



$$
\begin{aligned}
A &= \sigma\left(w_{00}x_0 + w_{02}x_1\right) \\
C &= \sigma\left(w_{10}A + w_{12}B\right) \\
y &= \mathbb{1}\left(w_{20}C + w_{21}D\right) \\
J &= \frac{1}{2}(d-y)^2
\end{aligned}
$$

Training

Backpropagation

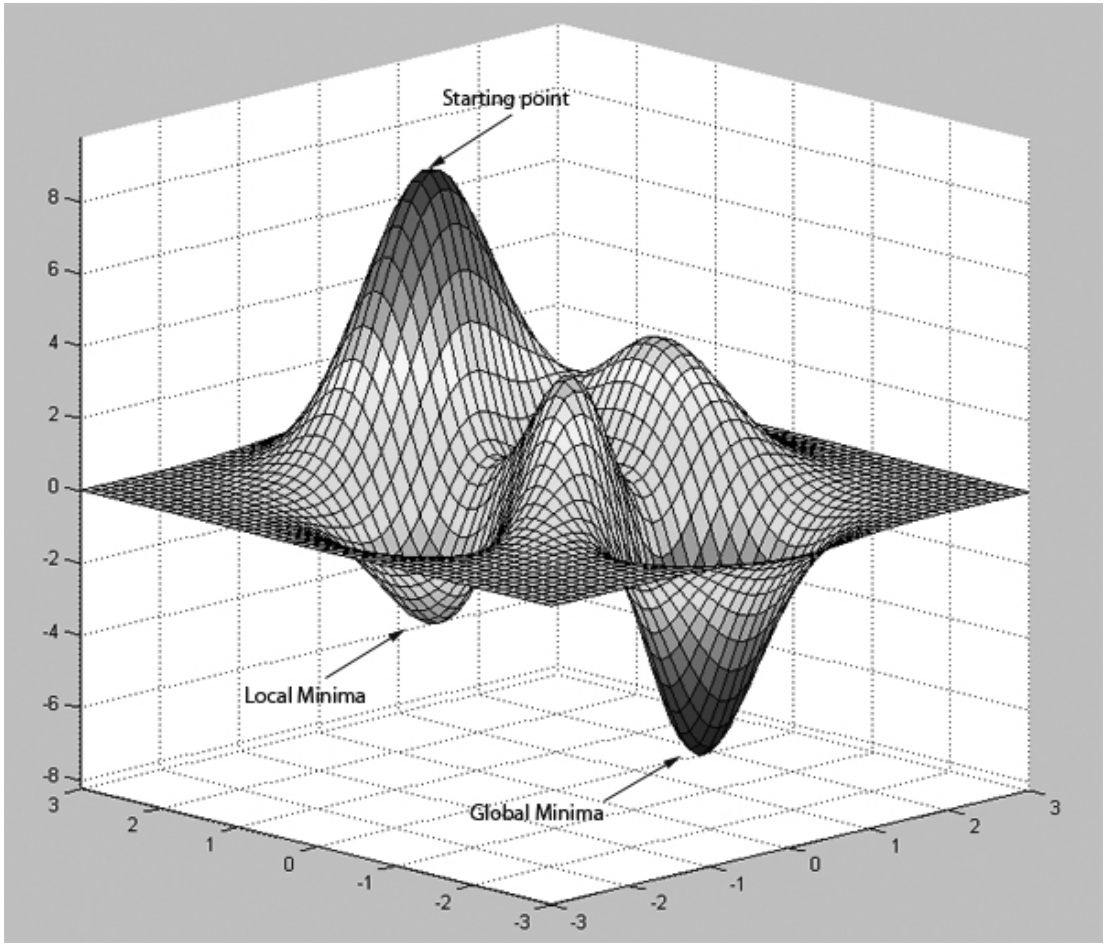


$$\frac{\partial J}{\partial w_{00}} = \frac{\partial J}{\partial A} \underbrace{\frac{\partial A}{\partial A_{in}}}_{\sigma'(A_{in})} \underbrace{\frac{\partial A_{in}}{\partial w_{00}}}_{x_0}$$

$$\frac{\partial J}{\partial A} = \underbrace{\frac{\partial J}{\partial C}}_{\text{previously computed}} \underbrace{\frac{\partial C}{\partial C_{in}}}_{\sigma'(C_{in})} \underbrace{\frac{\partial C_{in}}{\partial A}}_{w_{10}} + \underbrace{\frac{\partial J}{\partial D}}_{\text{previously computed}} \underbrace{\frac{\partial D}{\partial D_{in}}}_{\sigma'(D_{in})} \underbrace{\frac{\partial D_{in}}{\partial A}}_{w_{12}}$$

Optimizations

Introduction



Ciumac Sergiuj, `https://www.codeproject.com/Articles/175777/Financial-predictor-via-neural-network?msg=4290204`

Local minimas, Flat regions, Ill-Conditioning

Optimizations

Improving Gradient Descent

Gradient descent: $\mathbf{W}_{n+1} \leftarrow \mathbf{W}_n - \alpha \nabla J(\mathbf{W}_n; \mathbf{x}, \mathbf{d})$

- **Adaptive Learning Rate**: One learning rate per parameter, *adapted* by considering the parameter's *gradient* and its history (AdaGrad, RMSProp)
- **Momentum**: Moving in the direction of *past gradients* (Adam): $\mathbf{W}_{n+1} = \mathbf{W}_n + v_{n+1}$
  with $v_{n+1} = \epsilon v_n - \alpha \nabla_{\mathbf{W}} J(\mathbf{W}; \mathbf{x}, \mathbf{d})$
- **Initialization**: Ideally: Orthogonal initial weights; Practically: Control the *variance* (Xavier), scale (Saxe), or *sparsity* (Martens) of weights.

Optimizations

Improving the optimization problem

- $\mathcal{L}_\ell$ norm **regularization**: $\tilde{J}(\mathbf{W}; \mathbf{x}, \mathbf{d}) = J(\mathbf{W}; \mathbf{x}, \mathbf{d}) + \lambda ||\mathbf{W}||_\ell$
  - $\mathcal{L}_1$: Favours a sparse weight matrix.
  - $\mathcal{L}_2$: Limits weights by how much they contribute to gradient descent.
- **Noise**: Add noise to *inputs* or *weights*.
- **Early stopping**: Return parameters which perform best on the *test set*.
- **Dropout**: Randomly *remove nodes* from training.

Training & Regularization

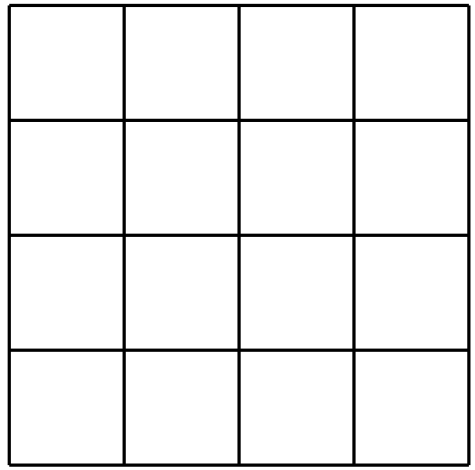
Summary

- Gradient descent incrementally searches parameters that decrease the loss J.
- Derivatives are calculated efficiently with backpropagation.
- Gradient descent can make slow progress or get stuck in local minimas.
- Careful weight initialization, momentum, and adaptive learning rate help to avoid problems with gradient descent.
- Overfitting can be avoided by regularization techniques.

Convolutional Neural Networks

Convolutional Neural Networks

Introduction



- Sequential or spatial structure in the input.
  E.g. images, text, spoken language, etc.
- Captures spatial nature of features (local neighborhood assumption)
- Two ”special” nodes: **Convolution** and **Pooling**

Convolutional Neural Networks

Convolution

0	1	0
1	-4	1
0	1	0

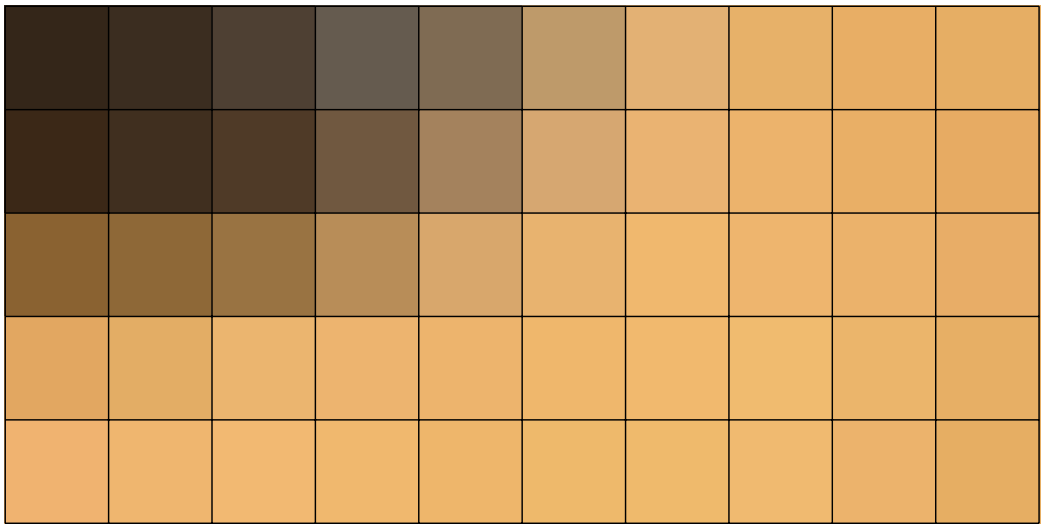
Input

Kernel
(sums to one)

Output
(a.k.a. feature map)

Convolutional Neural Neural

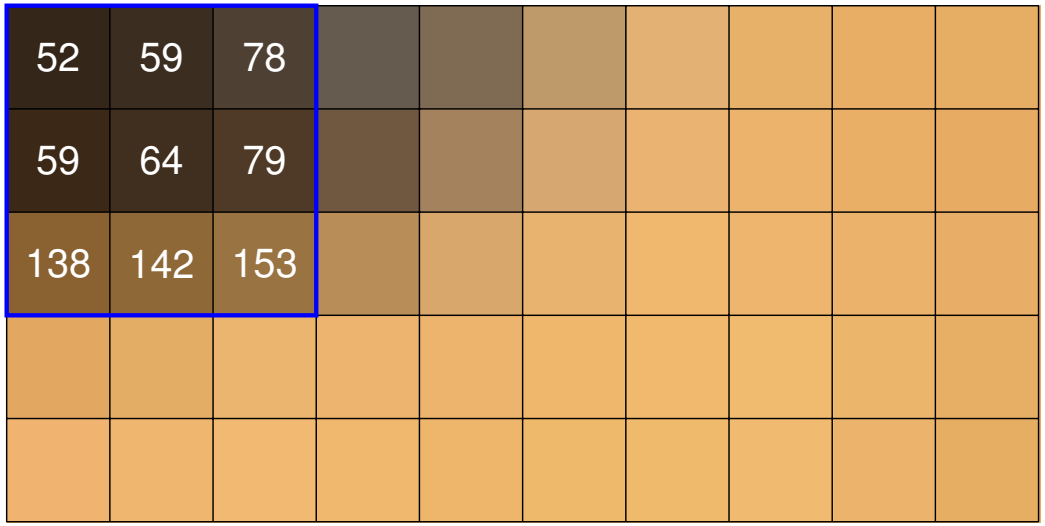
Convolution operation



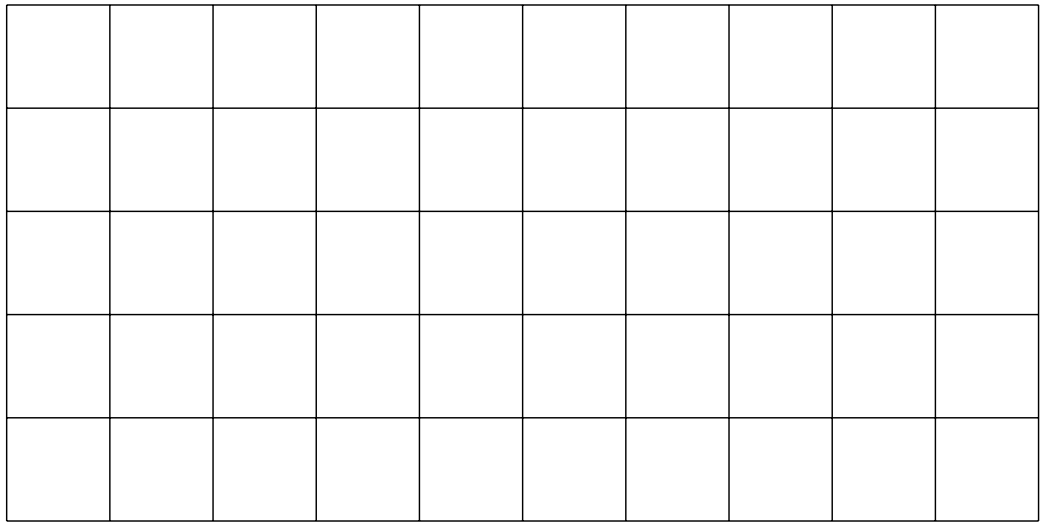
0	1	0
1	-4	1
0	1	0

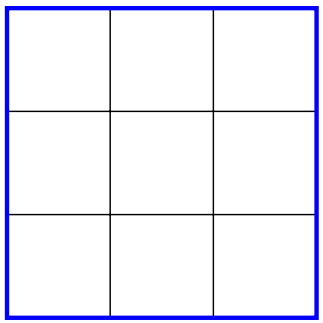
Convolutional Neural Neural

Convolution operation



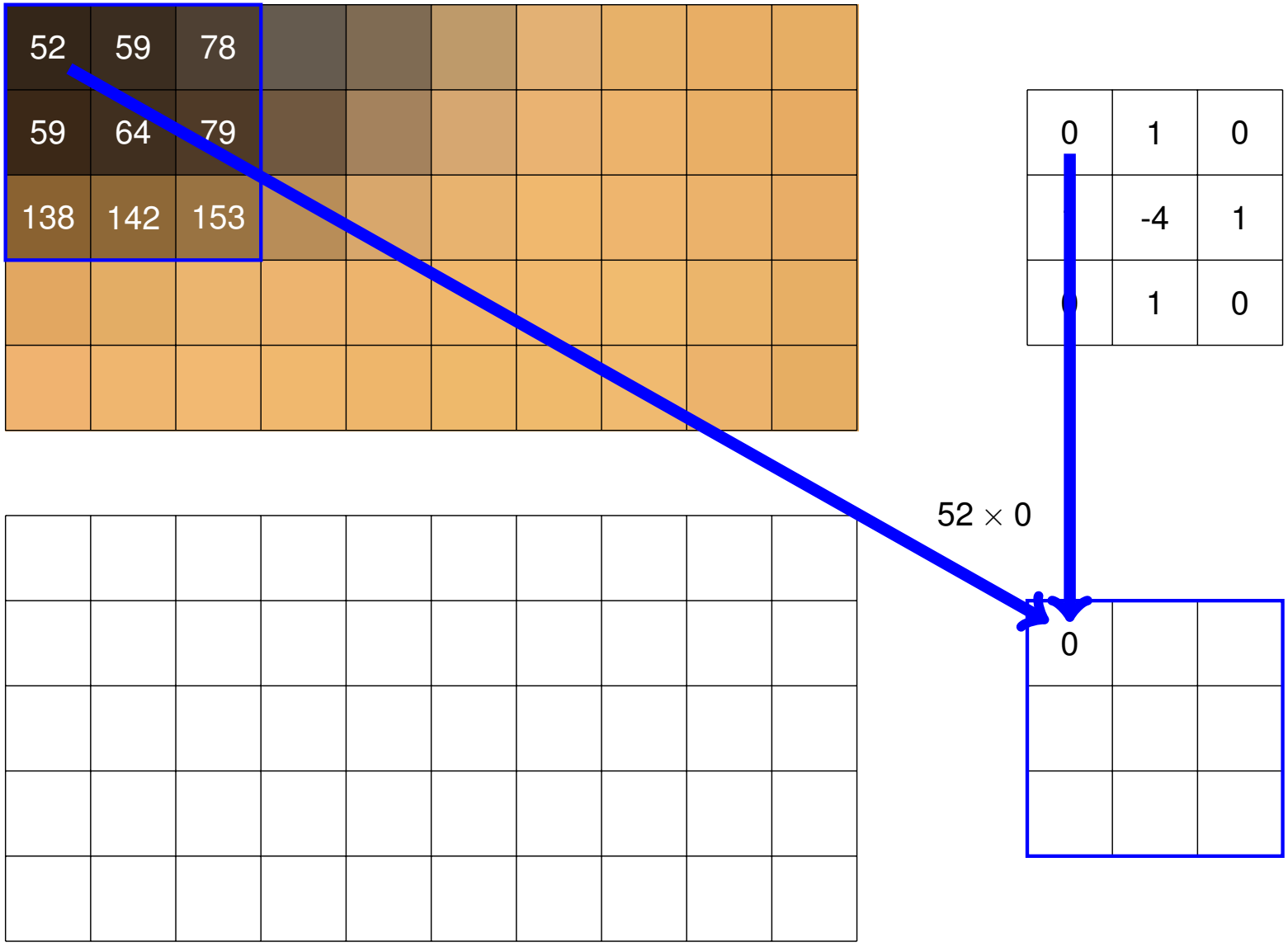
0	1	0
1	-4	1
0	1	0





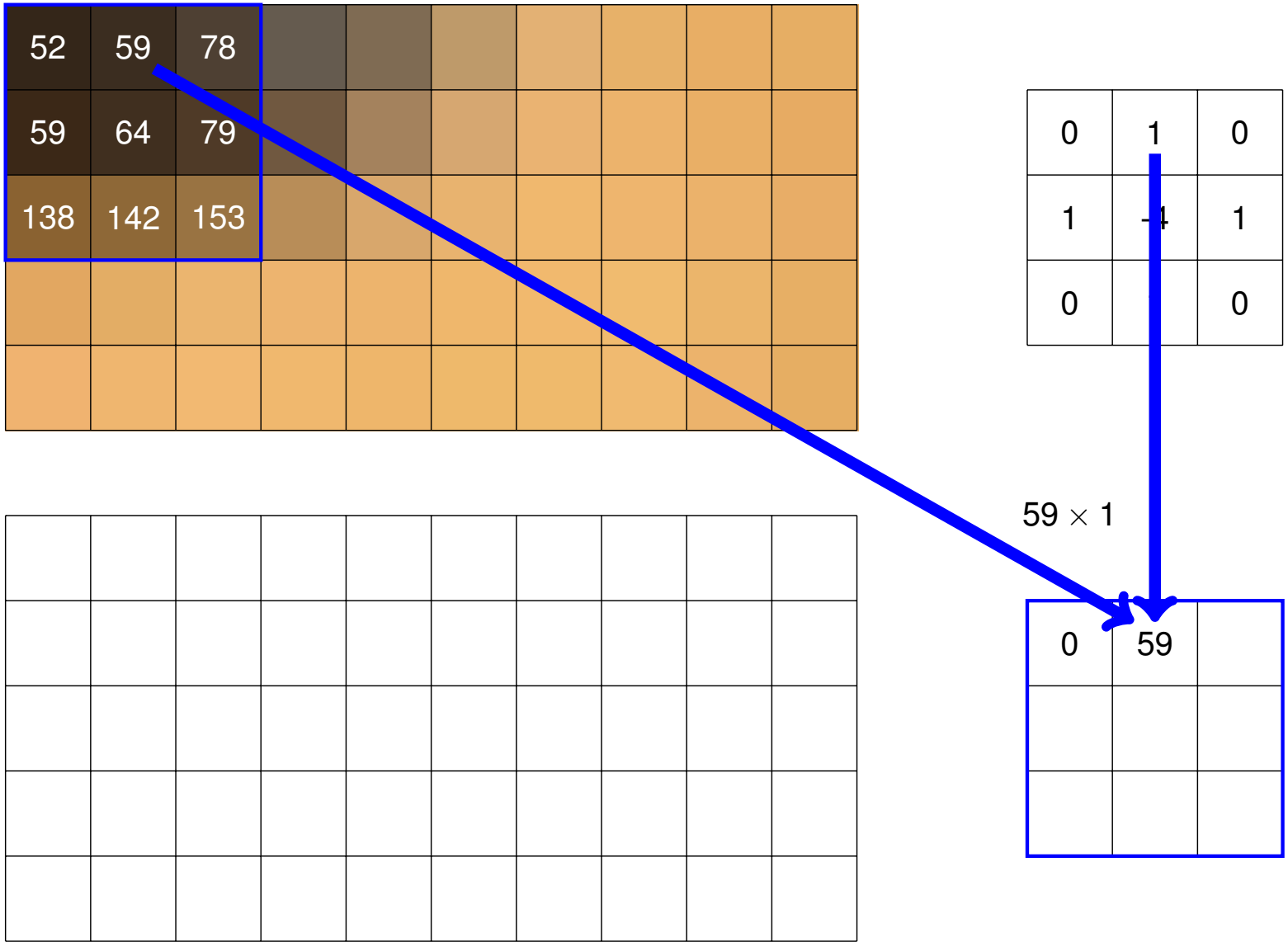
Convolutional Neural Neural

Convolution operation



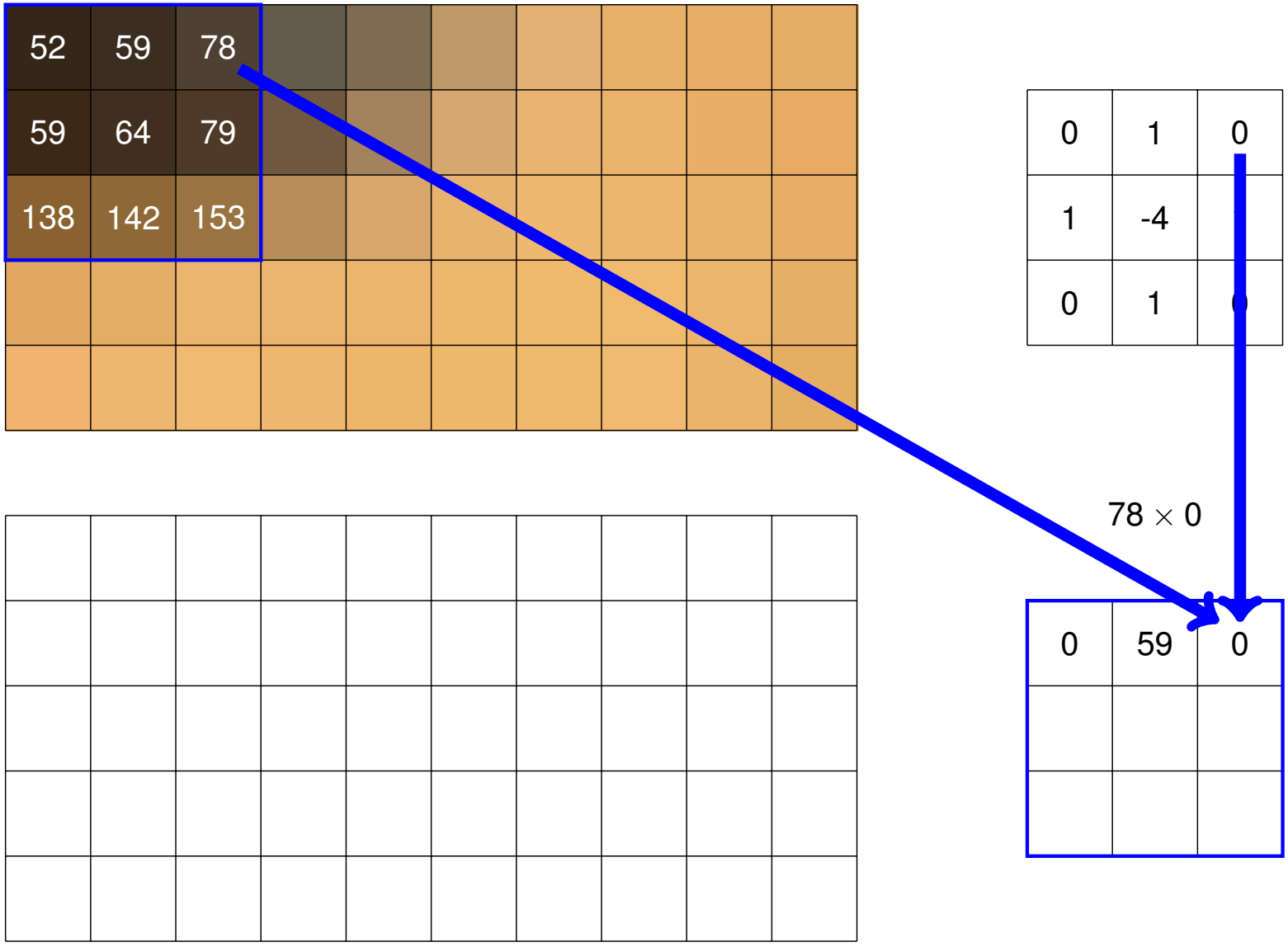
Convolutional Neural Neural

Convolution operation



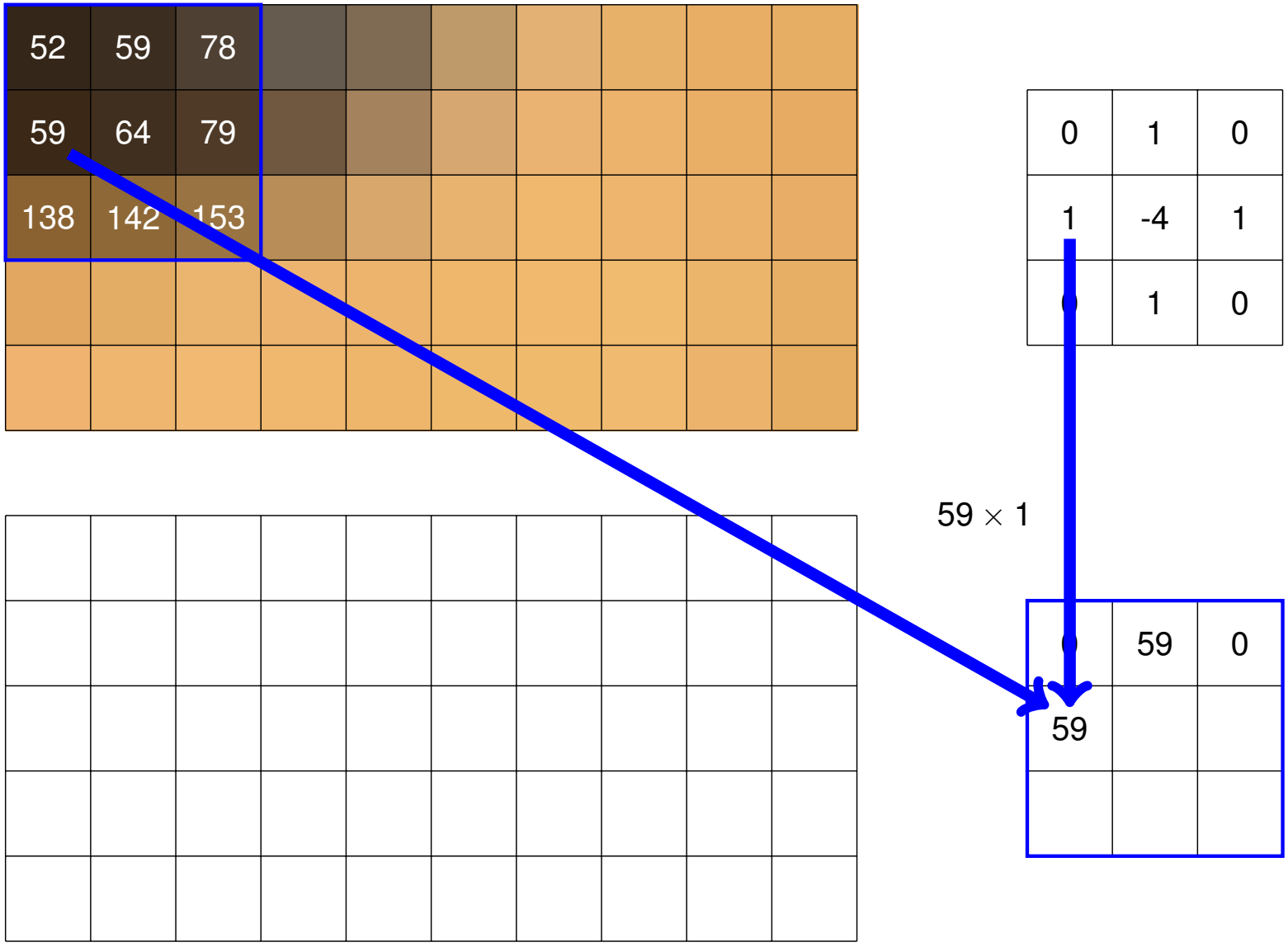
Convolutional Neural Neural

Convolution operation



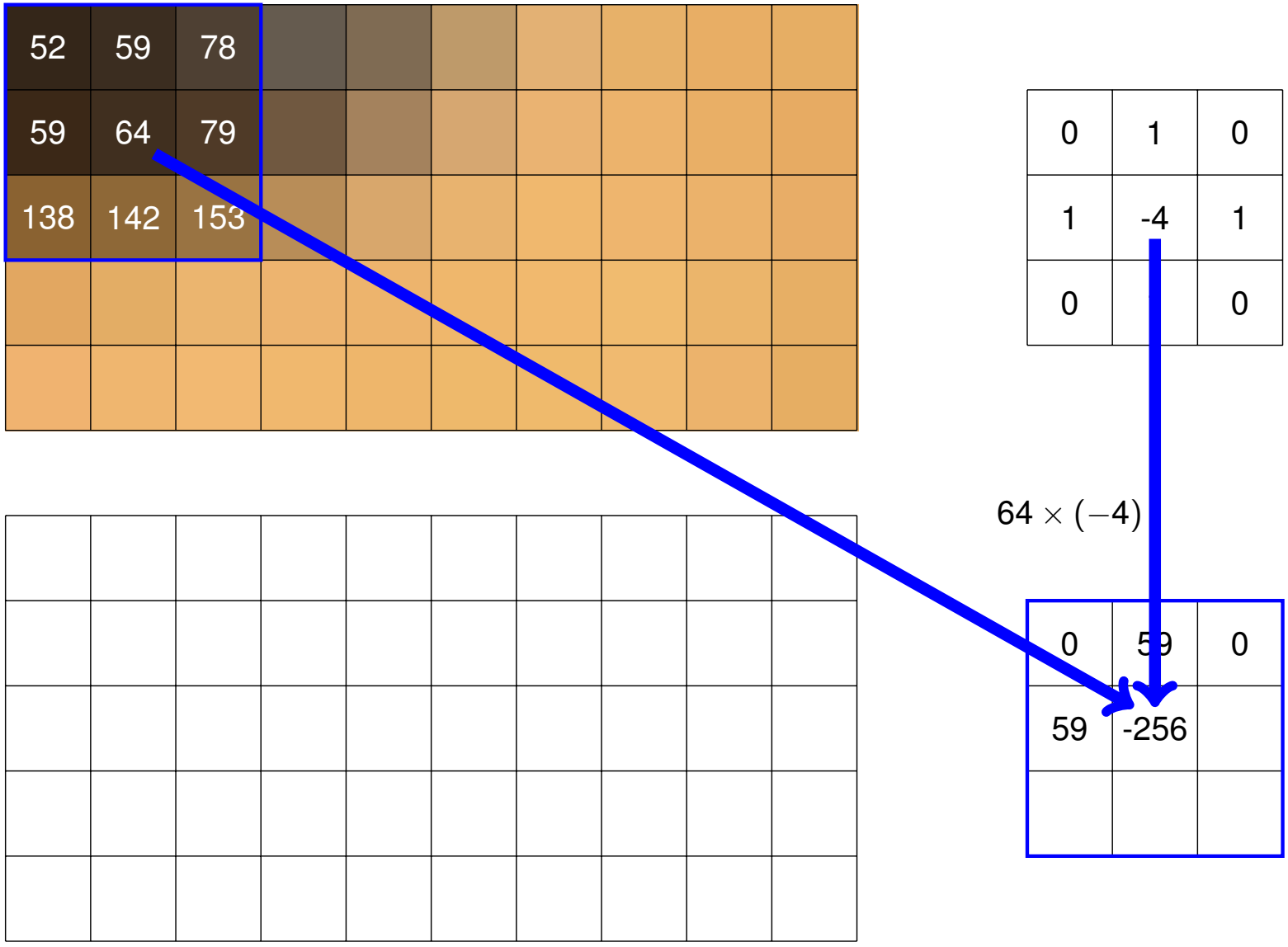
Convolutional Neural Neural

Convolution operation



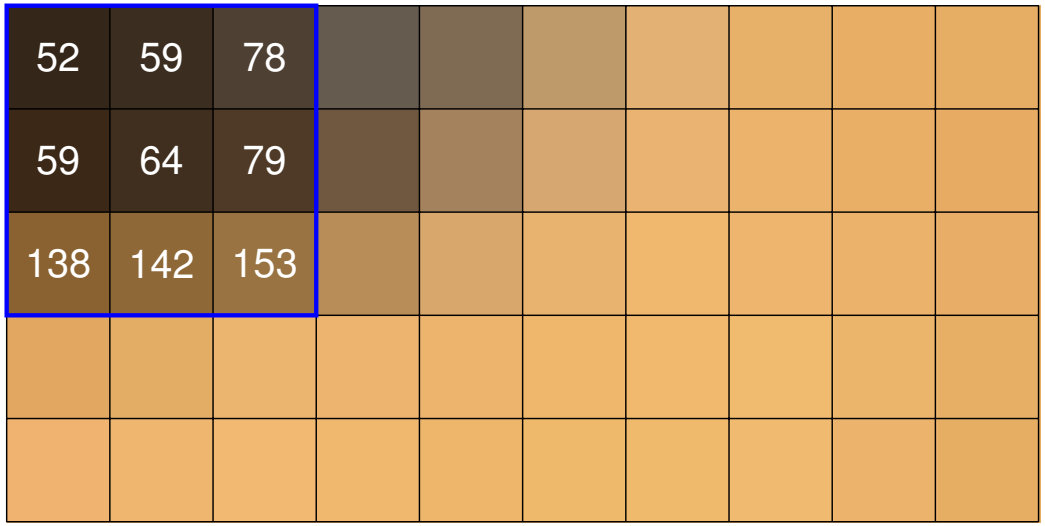
Convolutional Neural Neural

Convolution operation



Convolutional Neural Neural

Convolution operation

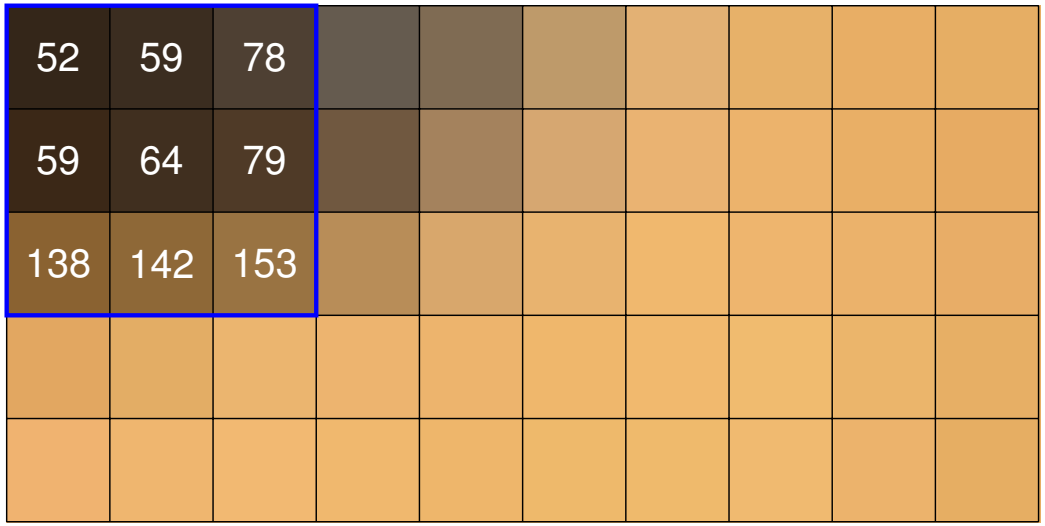


0	1	0
1	-4	1
0	1	0

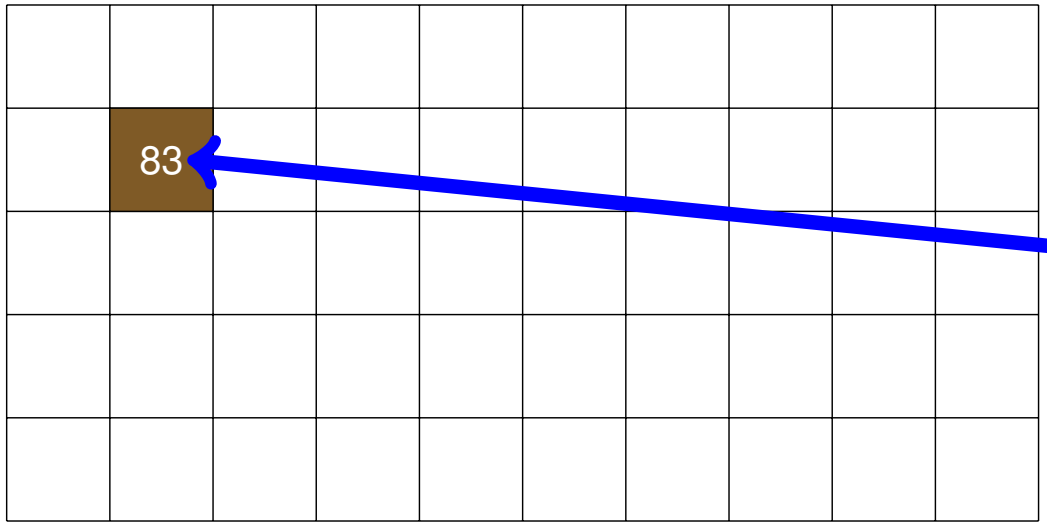
0	59	0
59	-256	79
0	142	0

Convolutional Neural Neural

Convolution operation



0	1	0
1	-4	1
0	1	0

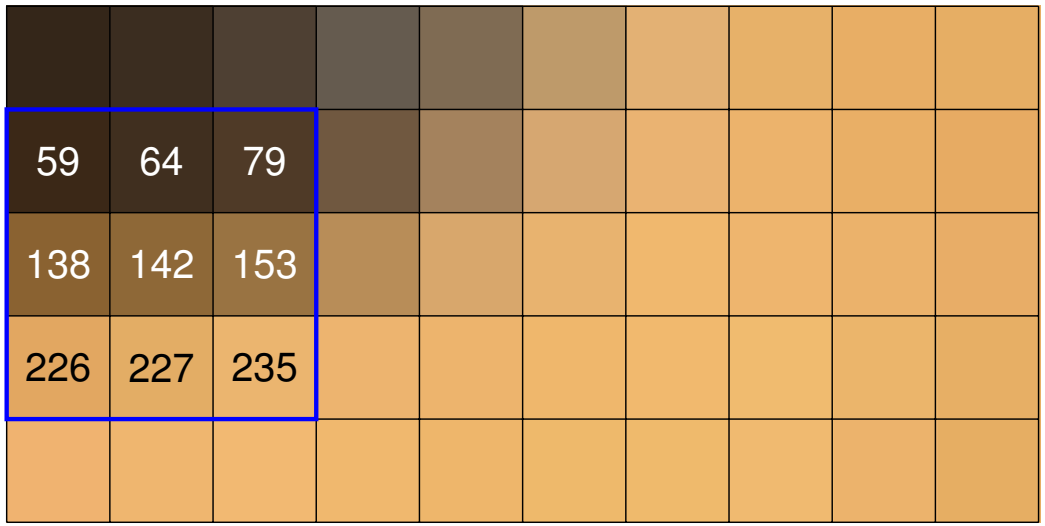


Σ

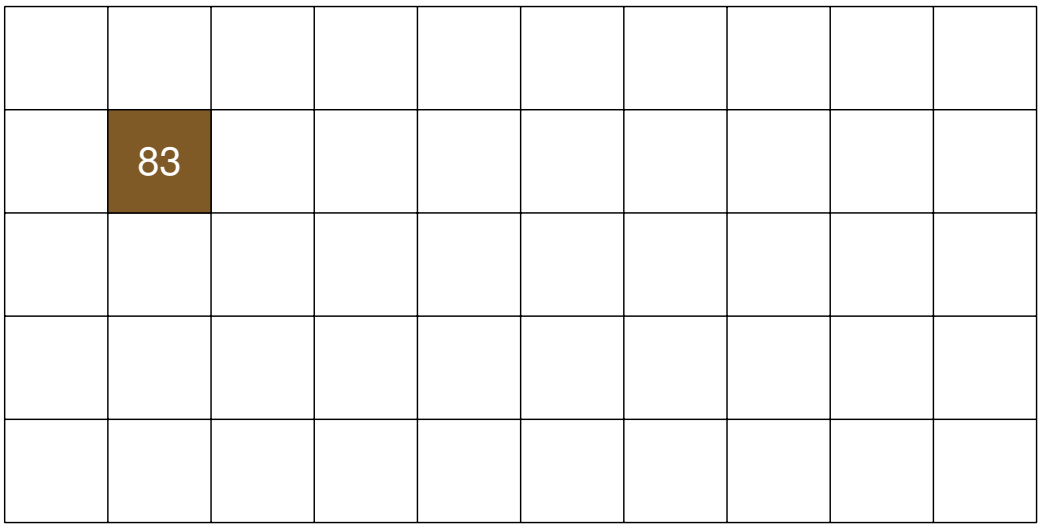
0	59	0
59	-256	79
0	142	0

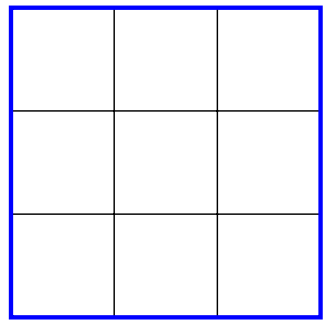
Convolutional Neural Neural

Convolution operation



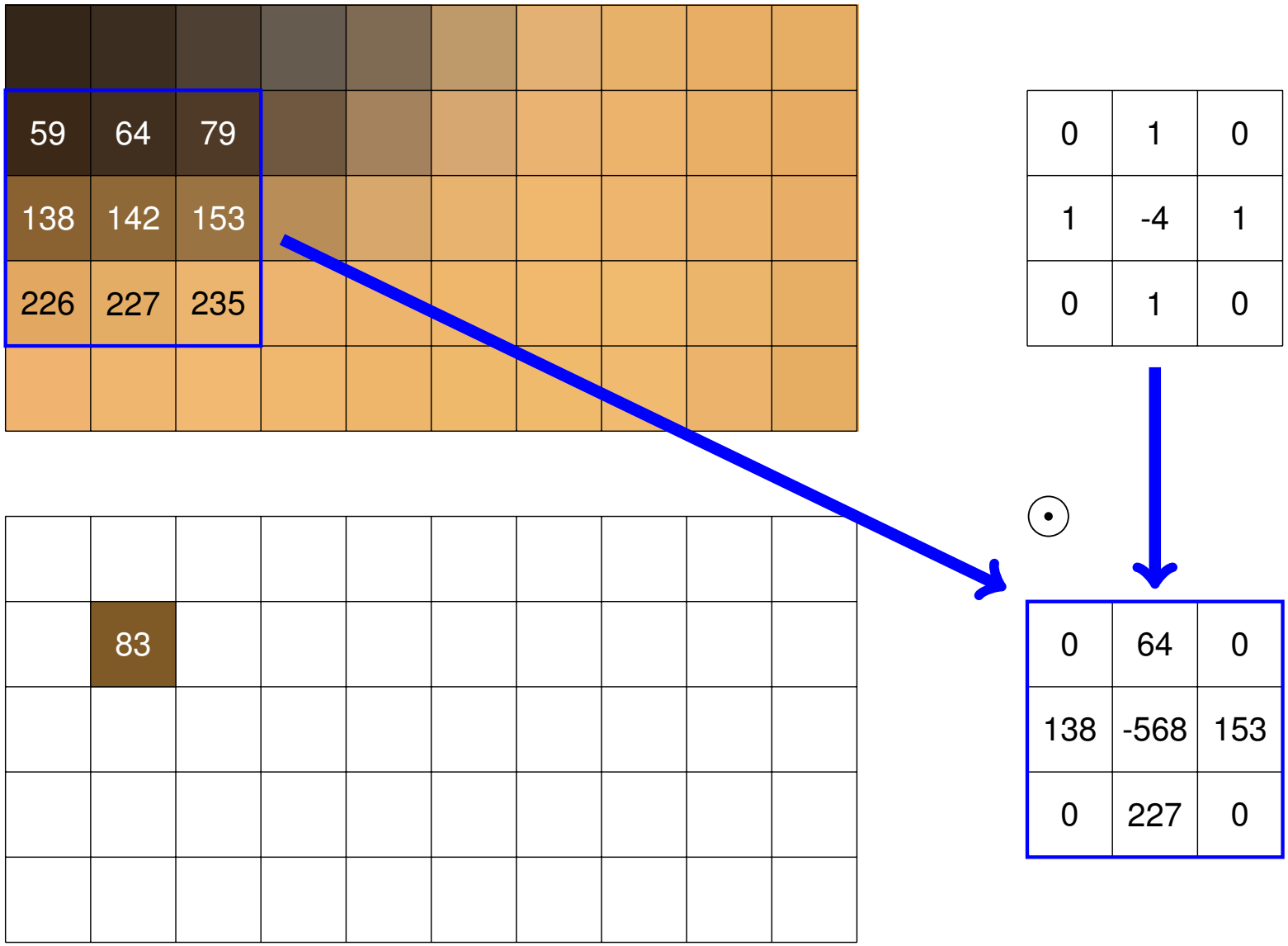
0	1	0
1	-4	1
0	1	0





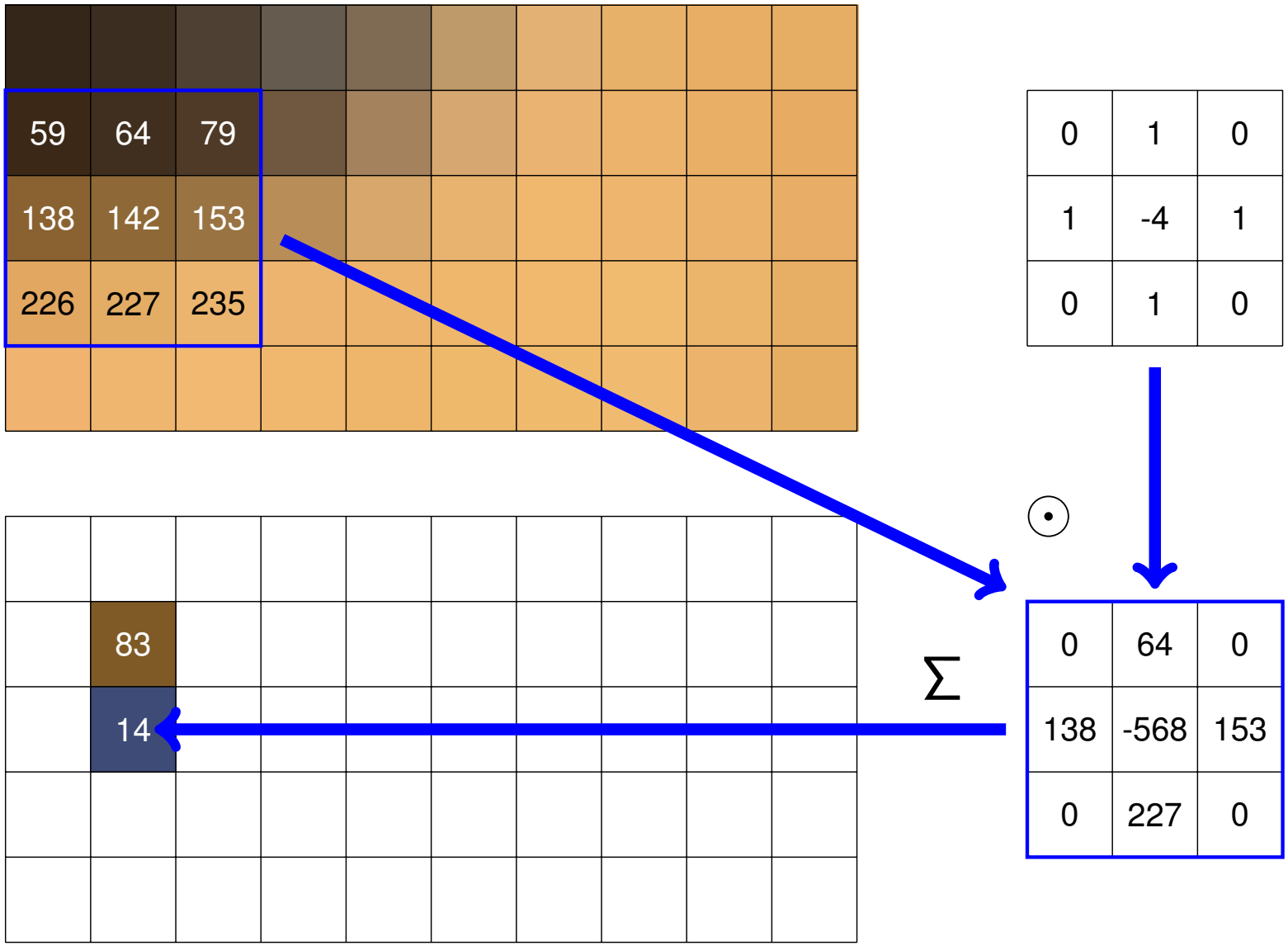
Convolutional Neural Neural

Convolution operation



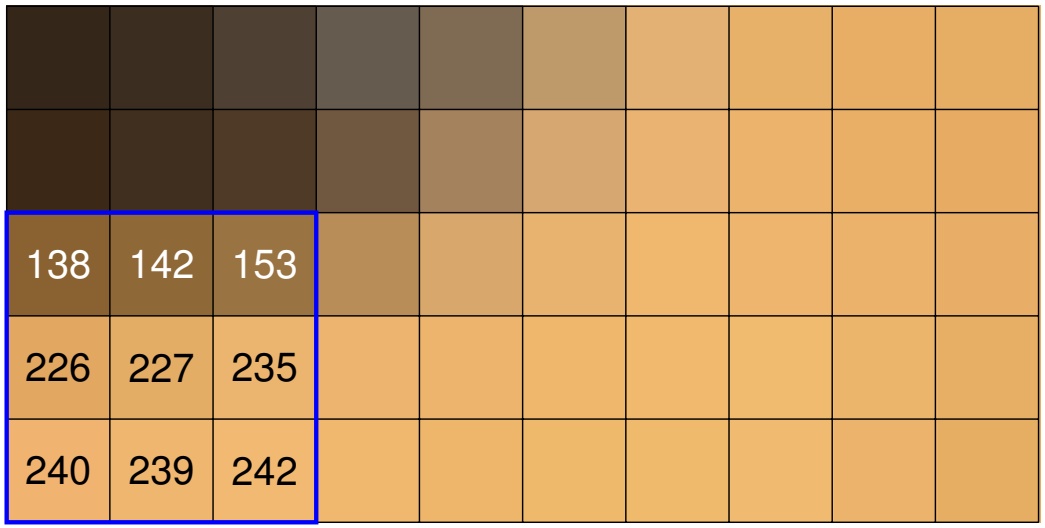
Convolutional Neural Neural

Convolution operation

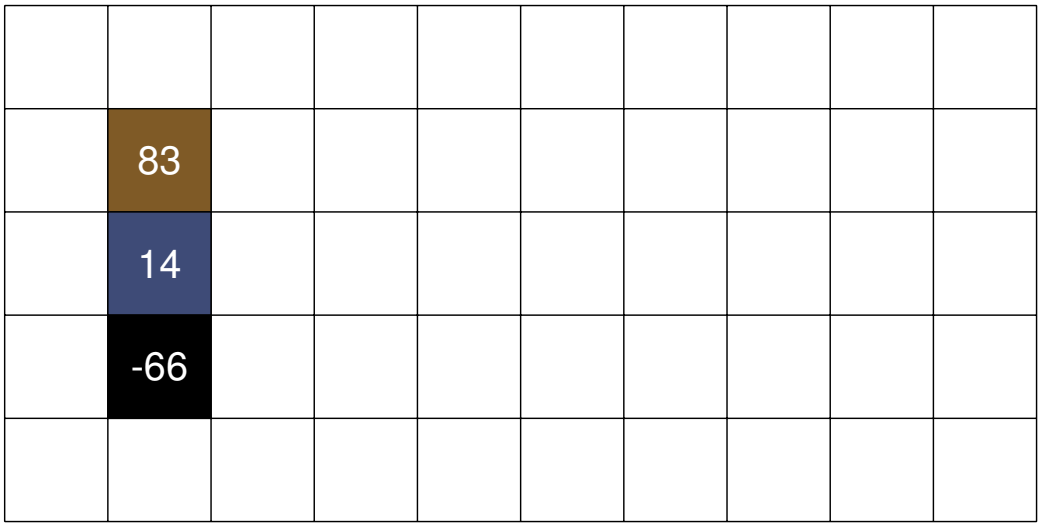


Convolutional Neural Neural

Convolution operation



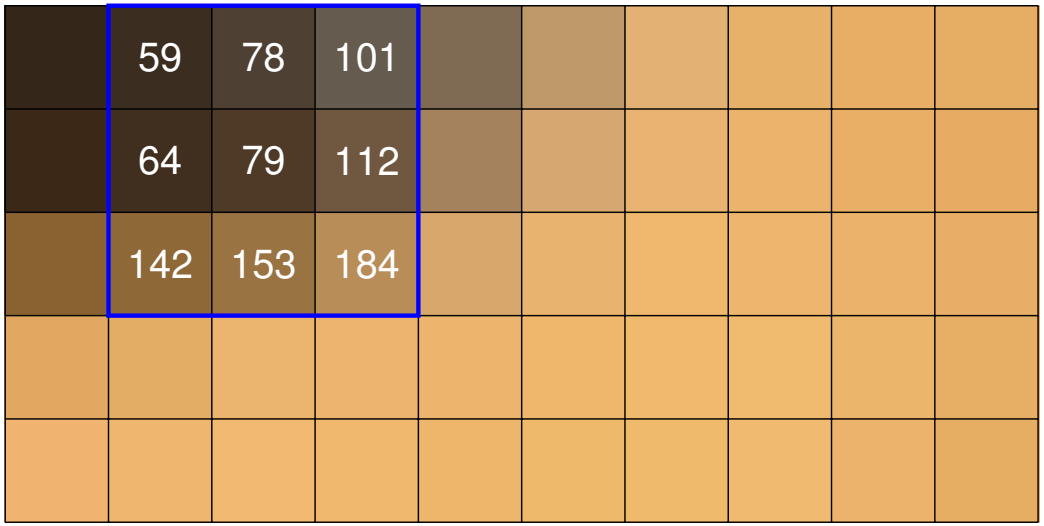
0	1	0
1	-4	1
0	1	0



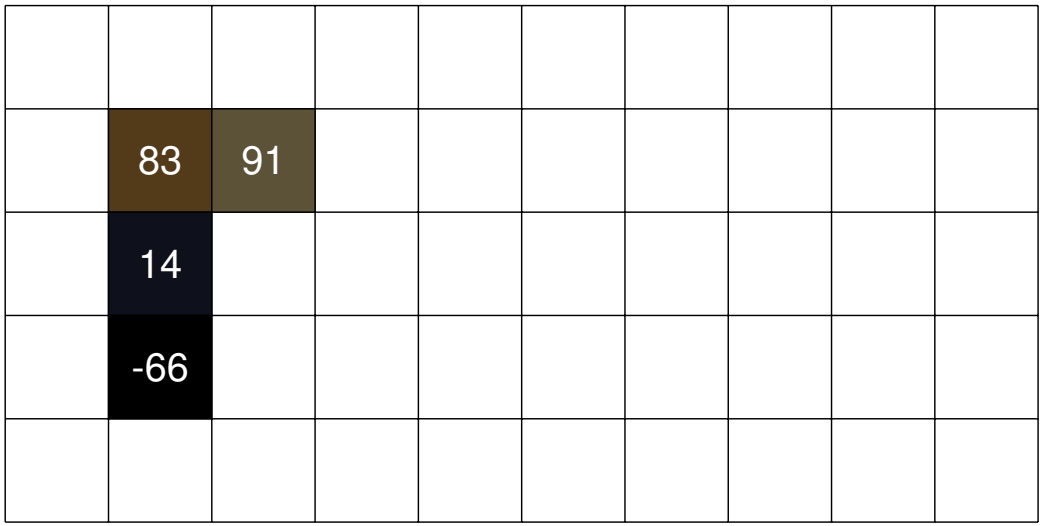
0	142	0
226	-908	235
0	239	0

Convolutional Neural Neural

Convolution operation



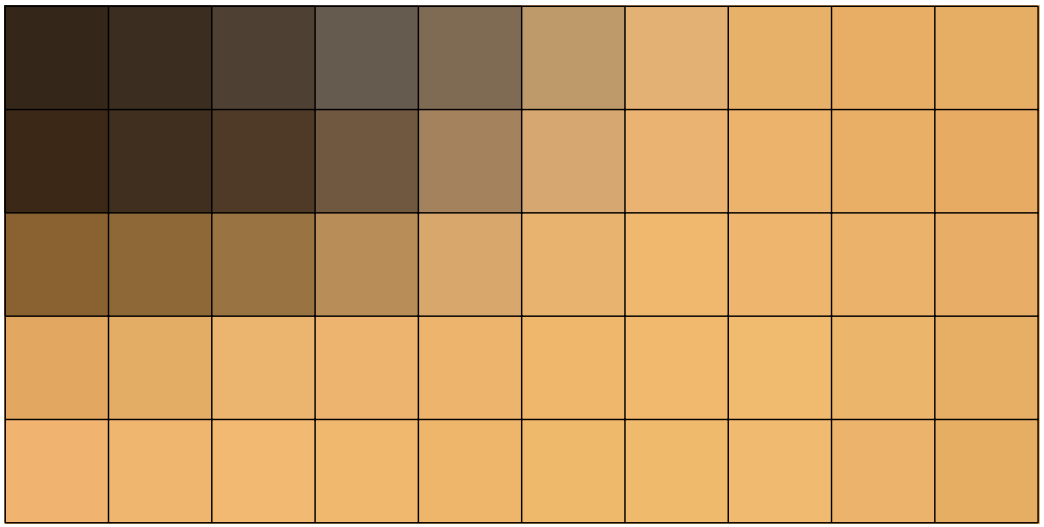
0	1	0
1	-4	1
0	1	0



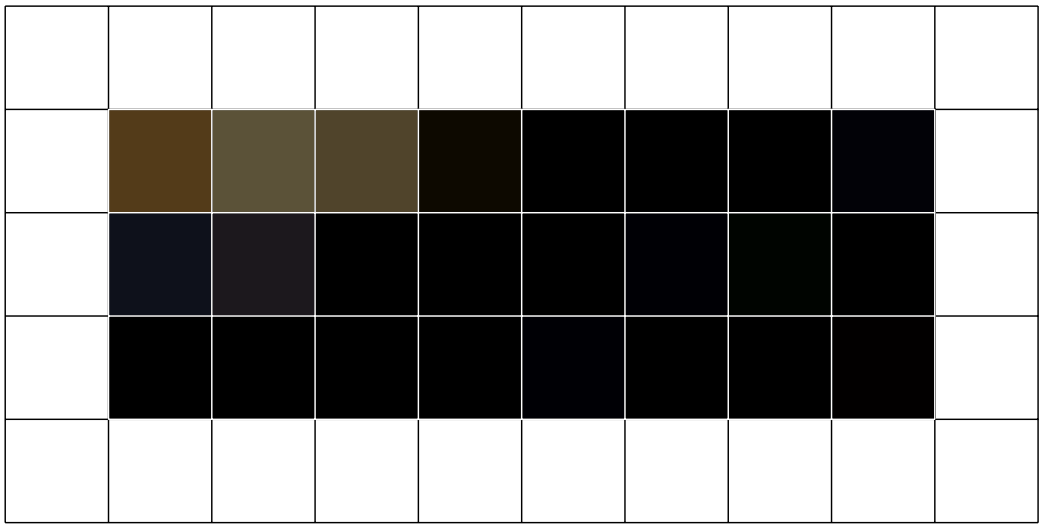
0	78	0
64	-316	112
0	153	0

Convolutional Neural Neural

Convolution operation

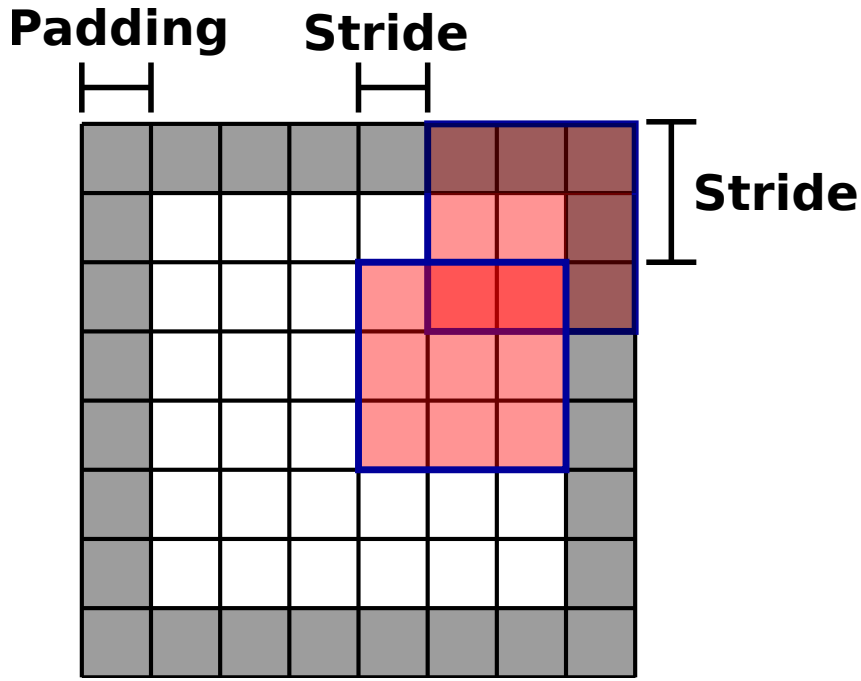


0	1	0
1	-4	1
0	1	0



Convolutional Neural Networks

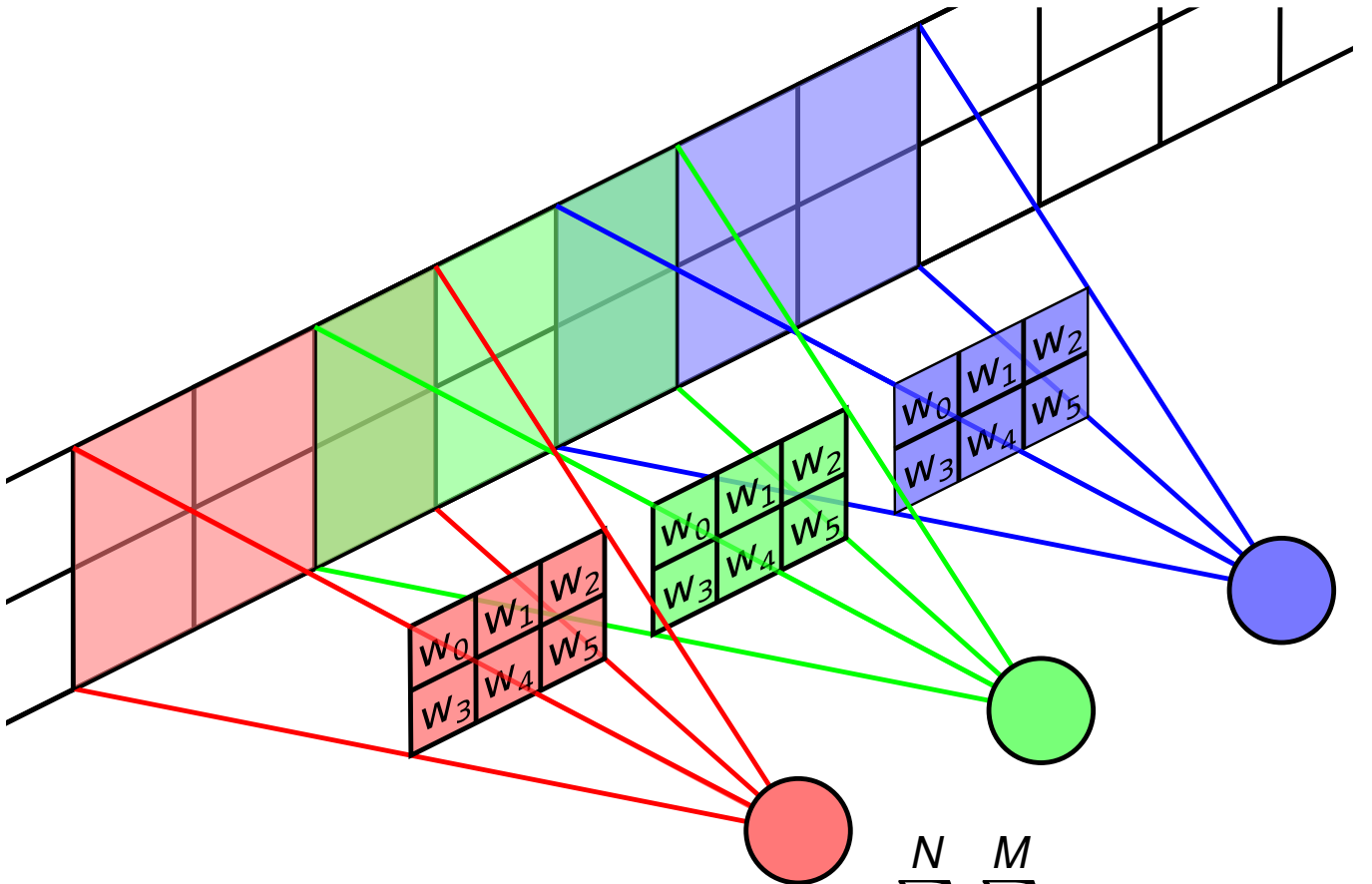
Borders & shift parameters



- **Stride**: Shift between perception fields of neurons.
- **Padding**: Additional border around input, with value zero.
- Output size = 1 + (Input + 2 Padding - Filter size) / Stride

Convolutional Neural Networks

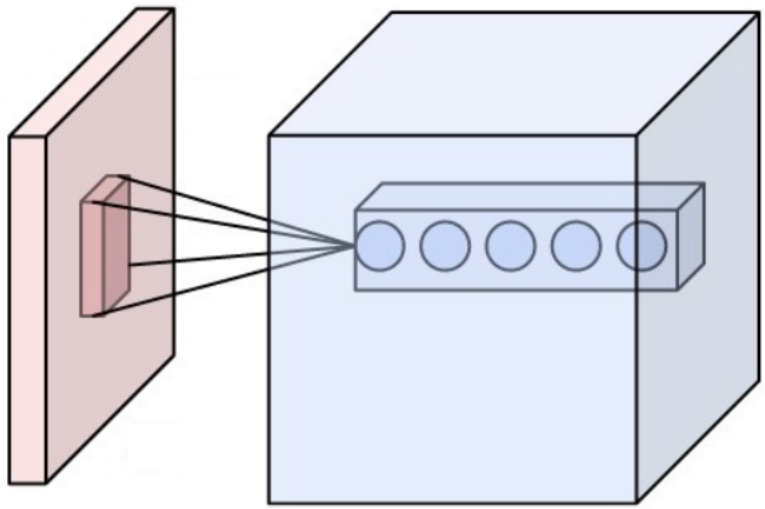
Convolution



$$\sum_{i=0}^{N}\sum_{j=0}^{M} w_{i,j}x_{i+o_x,j+o_y}$$

Convolutional Neural Networks

Convolution

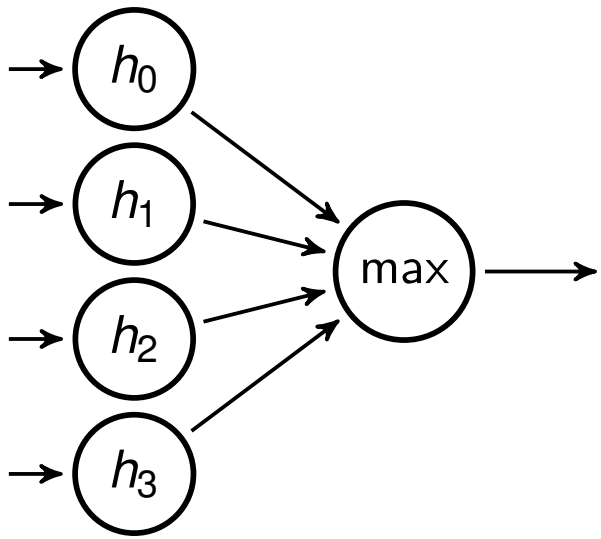


Aphex34, `https://commons.wikimedia.org/wiki/File:Conv_layer.png`

Convolutional Neural Networks

Pooling

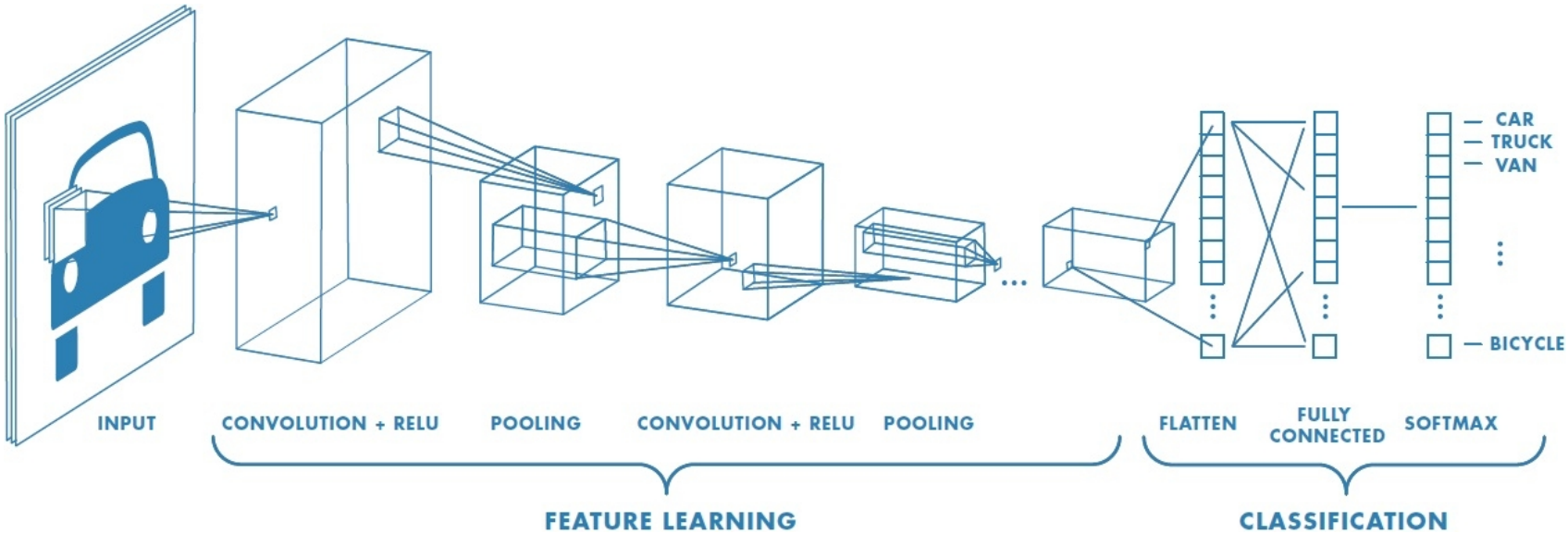
Summary over neighborhood, e.g. maximum, average, $\mathcal{L}_2$, etc.



- Pooling over spatial region: Translation invariance.

Convolutional Neural Networks

Topology



MathWorks, `https://de.mathworks.com/discovery/convolutional-neural-network.html`

1. Convolution
2. Activation
3. Pooling

Convolutional Neural Networks

Research example

Leon A. Gatys, Alexander S. Ecker, Matthias Bethge:
Image Style Transfer Using Convolutional Neural Networks

Figures on the following slides are taken from this paper

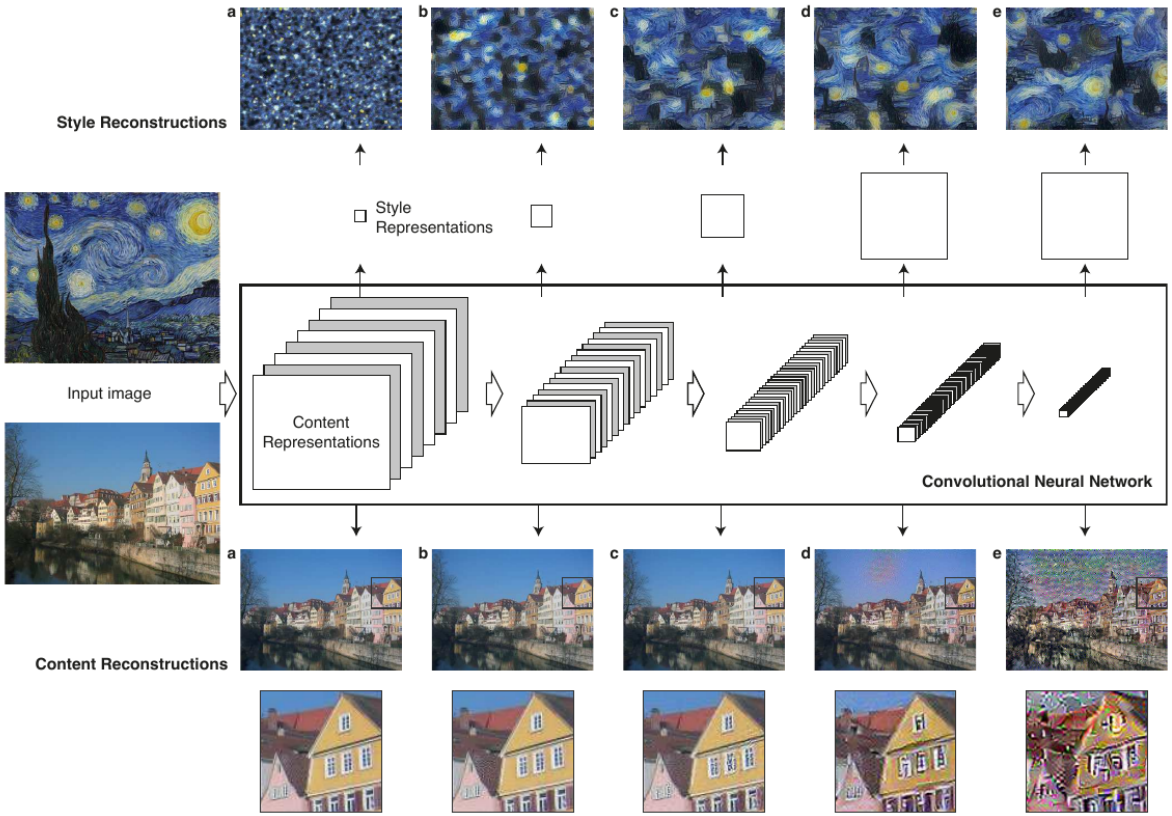
Convolutional Neural Networks

Research example

- Transfer Van Gogh's painting style from "The starry night" over to an image of the river of Tübingen
- Assumes that style and content of an image can be seperated.

Convolutional Neural Networks

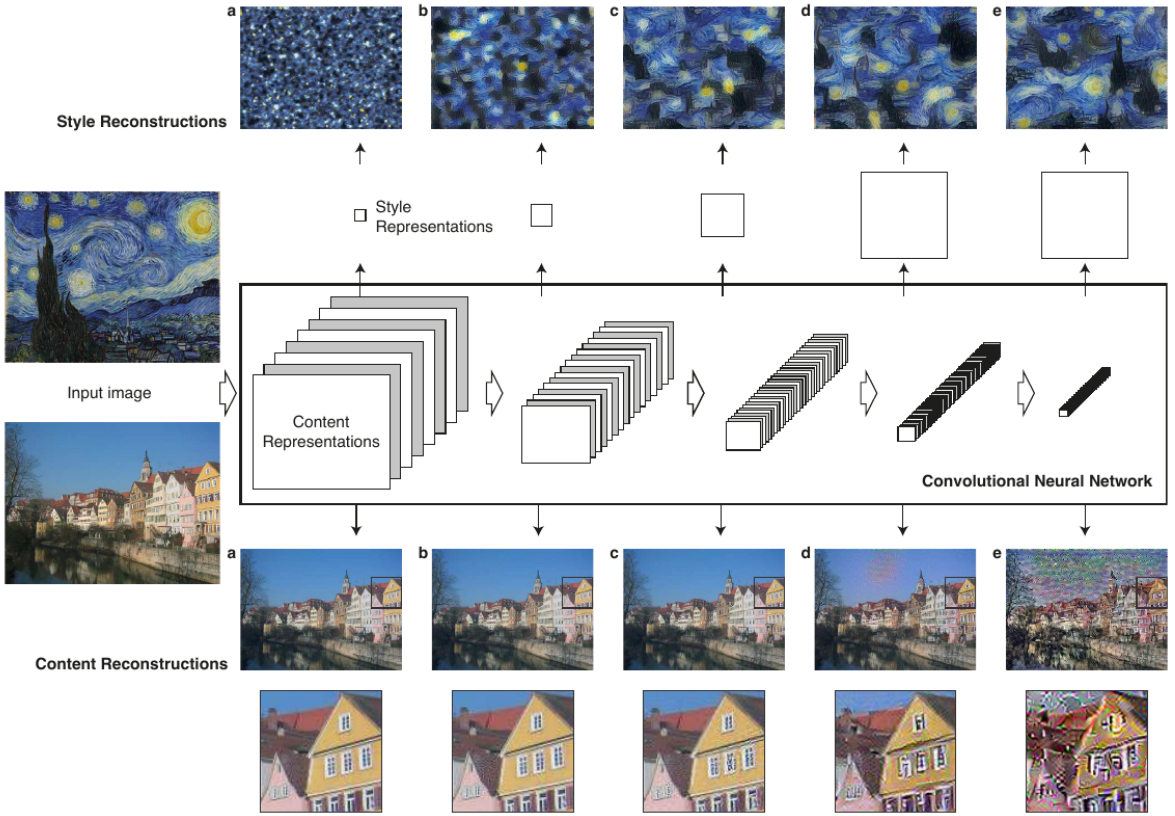
Research example



- Based on VGG network (object recognition and localization)
- Layers increasingly capture high-level features (i.e. content) and become invariant to precise pixel values.

Convolutional Neural Networks

Research example



- **Content representation**: Content image and generated image have same feature maps at layer ℓ.
- **Style representation**: Style image and generated image have same correlations between feature maps, on average.

Convolutional Neural Networks

Research example

- Representations are expressed as losses which takes the difference between source (content or style) and generated image into account.
- $\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{content}} + \beta \mathcal{L}_{\text{style}}$
- Generate image that minimizes $\mathcal{L}_{\text{total}}$
- Parameters: Ratio $\frac{\alpha}{\beta}$, Content layer ℓ

Convolutional Neural Networks

Research example — Ratio $\frac{\alpha}{\beta}$

10^{-4}

10^{-3}

10^{-2}

10^{-1}

Convolutional Neural Networks

Research example — Content layer

Convolutional Neural Networks

Research example

Convolutional Neural Networks

Research example

Convolutional Neural Networks

Research example

Convolutional Neural Networks

Research example

Convolutional Neural Networks

Research example

Conclusion

Summary

- Neural network: Layers size and depth, activation functions, connectivity, shared parameters.
- Has the capability of representing all practically relevant functions.
- Learn connection weights $\mathbf{W}$ via gradient descent and backpropagation.
- Improve learning speed by careful initialization, adaptive learning rate, and momentum
- Improve generalization by regularization
- Specialized models for certain input types
  - Sequential or spatial input: Convolutional Neural Networks (Convolution and Pooling nodes)
  - Sequential input: Recurrent Neural Networks (BPTT, Echo-State Networks, LSTM)

Summary

Why so popular?

- Parallelizable!
- Universal approximation
- Each layer identifies patterns in its input.
- Deeper layers identify more abstract patterns.
- Removes the need for hand-crafted features.

Further material

- Goodfellow et al.: Deep Learning
- Russell & Norvig: Artificial Intelligence: A Modern Approach

- `https://harishnarayanan.org/writing/artistic-style-transfer/`
- `https://github.com/keras-team/keras/blob/master/examples/deep_dream.py`

Sources

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio.
Deep learning, volume 1.
MIT press Cambridge, 2016.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.
Image Style Transfer Using Convolutional Neural Networks.
pages 2414–2423. IEEE, June 2016.

University of Zurich[UZH]