Chapter ML:IX

IX. Deep Learning

- Introduction to Deep Learning
- Autoencoder Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long-Term Dependencies
- RNNs for Machine Translation
- Attention Mechanism
- Self Attention and Transformers
- Transformer Language Models
- Pretraining

Introduction to Deep Learning

History

"Deep Learning" is not a particular method nor a new technology, but an umbrella term under which various developments are collected. The term may be examined both under a historical perspective and a field delineation in AI. [Perceptron Learning]

Cybernetics (1940s–1960s)

1943 Model of the neuron (McCulloch and Pitts)
1958 Perceptron model (Rosenblatt)
1960 Adaptive linear element, ADALINE, a kind of stochastic gradient descent (Widrow and Hoff).

Connectionism / Parallel distributed processing (1980s–1990s)

1986 Paradigm of distributed representation (Hinton et al.)
1986 Back-propagation algorithm (Rumelhart et al., LeCun 1987)
1997 Long short-term memory network, LSTM (Hochreiter and Schmidhuber)

Deep Learning (2006–today)

2006 Strategy of greedy, layer-wise pretraining (Hinton et al., Bengio et al. 2007, Ranzato 2007)
2014 Theory on the importance of depth (Pascanu et al., Montufar et al.)
2017 Transformer network architecture (Vaswani et al.)

[Goodfellow/Bengio/Courville 2016]

Introduction to Deep Learning

History (continued)

"Deep Learning" is not a particular method nor a new technology, but an umbrella term under which various developments are collected. The term may be examined both under a historical perspective and a field delineation in AI. [Perceptron Learning]

Cybernetics (1940s–1960s)

1943 Model of the neuron (McCulloch and Pitts)
1958 Perceptron model (Rosenblatt)
1960 Adaptive linear element, ADALINE, a kind of stochastic gradient descent (Widrow and Hoff).

Connectionism / Parallel distributed processing (1980s–1990s)

1986 Paradigm of distributed representation (Hinton et al.)
1986 Back-propagation algorithm (Rumelhart et al., LeCun 1987)
1997 Long short-term memory network, LSTM (Hochreiter and Schmidhuber)

Deep Learning (2006–today)

2006 Strategy of greedy, layer-wise pretraining (Hinton et al., Bengio et al. 2007, Ranzato 2007)
2014 Theory on the importance of depth (Pascanu et al., Montufar et al.)
2017 Transformer network architecture (Vaswani et al.)

[Goodfellow/Bengio/Courville 2016]

Introduction to Deep Learning

History (continued)

"Deep Learning" is not a particular method nor a new technology, but an umbrella term under which various developments are collected. The term may be examined both under a historical perspective and a field delineation in AI. [Perceptron Learning]

Cybernetics (1940s–1960s)

1943 Model of the neuron (McCulloch and Pitts)
1958 Perceptron model (Rosenblatt)
1960 Adaptive linear element, ADALINE, a kind of stochastic gradient descent (Widrow and Hoff).

Connectionism / Parallel distributed processing (1980s–1990s)

1986 Paradigm of distributed representation (Hinton et al.)
1986 Back-propagation algorithm (Rumelhart et al., LeCun 1987)
1997 Long short-term memory network, LSTM (Hochreiter and Schmidhuber)

Deep Learning (2006–today)

2006 Strategy of greedy, layer-wise pretraining (Hinton et al., Bengio et al. 2007, Ranzato 2007)
2014 Theory on the importance of depth (Pascanu et al., Montufar et al.)
2017 Transformer network architecture (Vaswani et al.)

[Goodfellow/Bengio/Courville 2016]

Introduction to Deep Learning

History (continued)

"Deep Learning" is not a particular method nor a new technology, but an umbrella term under which various developments are collected. The term may be examined both under a historical perspective and a field delineation in AI. [Perceptron Learning]

Cybernetics (1940s–1960s)

1943 Model of the neuron (McCulloch and Pitts)
1958 Perceptron model (Rosenblatt)
1960 Adaptive linear element, ADALINE, a kind of stochastic gradient descent (Widrow and Hoff).

Connectionism / Parallel distributed processing (1980s–1990s)

1986 Paradigm of distributed representation (Hinton et al.)
1986 Back-propagation algorithm (Rumelhart et al., LeCun 1987)
1997 Long short-term memory network, LSTM (Hochreiter and Schmidhuber)

Deep Learning (2006–today)

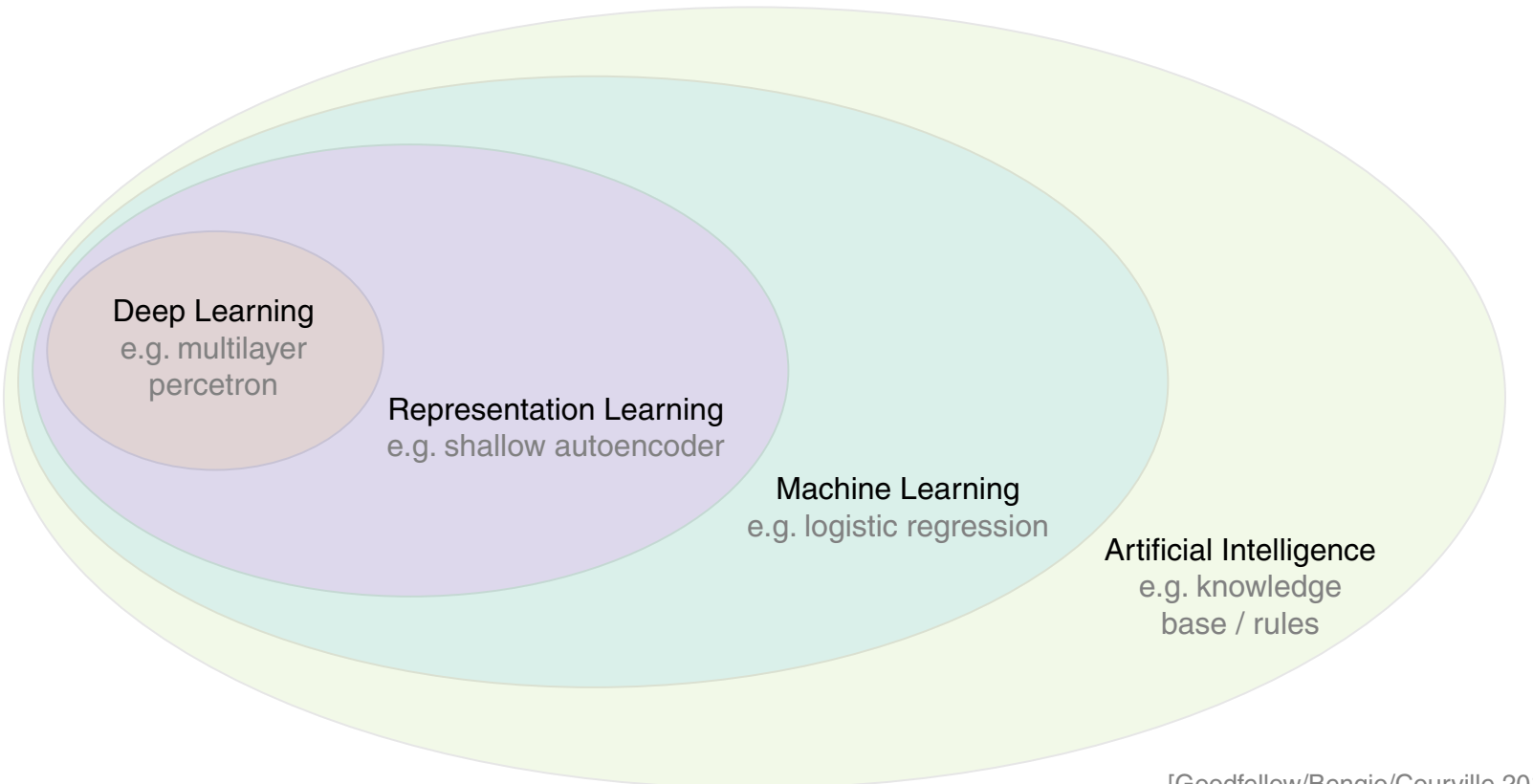
2006 Strategy of greedy, layer-wise pretraining (Hinton et al., Bengio et al. 2007, Ranzato 2007)
2014 Theory on the importance of depth (Pascanu et al., Montufar et al.)
2017 Transformer network architecture (Vaswani et al.)

[Goodfellow/Bengio/Courville 2016]

Introduction to Deep Learning

History (continued)

"Deep Learning" is not a particular method nor a new technology, but an umbrella term under which various developments are collected. The term may be examined both under a historical perspective and a field delineation in AI. [Perceptron Learning]



[Goodfellow/Bengio/Courville 2016]

Introduction to Deep Learning

History (continued)

The history of neural computing came along with a nearly exponential development in computing power, which eventually could be “tamed” to deal with the advent of Big Data and model functions with very lage parameter spaces.

➜ The machine-based solution of “AI problems” reached a new quality.

Characteristica of selected problems and solutions:

	1980s	...	2020s
Computing power (per device)	20 MFLOPs (Fujitsu MB86900)		300 TFLOPs (Nvidia A100)
Memory (per device)	$<$ 128 MB		$>$ 80 GB (GPU)
Devices (per model)	1$\times$ SUN-4/260 (SN, 1988)		$>$ 10,000$\times$ Nvidia V-100 (GPT-3, 2020)
Number of network parameters	$<$ 10,000 (SN, 1988)		175 Billion (GPT-3, 2020)
Speech recognition	DragonDictate		Google Now, MS Cortana, Apple Siri
Translation quality			DeepL, Google Translate

▷ Deep Learning

Introduction to Deep Learning

History (continued)

The history of neural computing came along with a nearly exponential development in computing power, which eventually could be “tamed” to deal with the advent of Big Data and model functions with very lage parameter spaces.

➜ The machine-based solution of “AI problems” reached a new quality.

Characteristica of selected problems and solutions:

	1980s	...	2020s
Computing power (per device)	20 MFLOPs (Fujitsu MB86900)		300 TFLOPs (Nvidia A100)
Memory (per device)	$<$ 128 MB		$>$ 80 GB (GPU)
Devices (per model)	1$\times$ SUN-4/260 (SN, 1988)		$>$ 10,000$\times$ Nvidia V-100 (GPT-3, 2020)
Number of network parameters	$<$ 10,000 (SN, 1988)		175 Billion (GPT-3, 2020)
Speech recognition	DragonDictate		Google Now, MS Cortana, Apple Siri
Translation quality			DeepL, Google Translate

▷ Deep Learning

Introduction to Deep Learning

Types of Learning Tasks

Mapping between constant (“stationary”) input and output:

- vector $\rightarrow$ class — email $\rightarrow \{\text{spam, ham}\}$
- matrix $\rightarrow$ class — image classification
- vector $\rightarrow$ vector — embedding

Mapping tasks involving sequences (“dynamic” input or output) :

(S1) sequence $\rightarrow$ class — sentence $\rightarrow \{\oplus, \ominus\}$

(S2) class $\rightarrow$ sequence — $\{\oplus, \ominus\} \rightarrow$ sentence

(S3) sequence $\rightarrow$ sequence — English sentence $\rightarrow$ German sentence

Solving one of these tasks means to find / construct / parameterize a model function $y()$ that operationalizes the mapping “$\rightarrow$”.

Introduction to Deep Learning

Types of Learning Tasks

Mapping between constant (“stationary”) input and output:

- vector $\rightarrow$ class — email $\rightarrow \{\text{spam, ham}\}$
- matrix $\rightarrow$ class — image classification
- vector $\rightarrow$ vector — embedding

Mapping tasks involving sequences (“dynamic” input or output) :

(S1) sequence $\rightarrow$ class — sentence $\rightarrow \{\oplus, \ominus\}$

(S2) class $\rightarrow$ sequence — $\{\oplus, \ominus\} \rightarrow$ sentence

(S3) sequence $\rightarrow$ sequence — English sentence $\rightarrow$ German sentence

Solving one of these tasks means to find / construct / parameterize a model function $y()$ that operationalizes the mapping “$\rightarrow$”.

Remarks:

- "$\oplus$" and "$\ominus$" denote positive and negative sentiment respectively.
- Here, the model function $y()$ (the mapping which is behind the "$\rightarrow$") is a neural network such as the Multilayer perceptron.
- If the model function is vector-valued, it is written bold, as $\mathbf{y}()$.
- The model function $y()$ (or $\mathbf{y}()$) can take scalars, vectors, matrices, or sequences (typically of vectors) as arguments / input.
- An input vector (scalar, matrix) can encode an entire problem instance or, given sequential information, a certain time step only. An example for the former are all words of an email that is to be classified as spam or ham. An example for the latter is a single word at a certain position in a sentence, where the entire sentence is the instance of a translation problem.

Chapter ML:IX (continued)

IX. Deep Learning

- Introduction to Deep Learning
- Autoencoder Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long-Term Dependencies
- RNNs for Machine Translation
- Attention Mechanism
- Self Attention and Transformers
- Transformer Language Models
- Pretraining

Autoencoder Networks

Representation Learning Task: Word Embedding

[*TODO*]

Autoencoder Networks

Word Embedding with Autoencoders

[*TODO*]

Autoencoder Networks

Representation Learning Task: Co-Occurrence Embedding (Word2Vec)

[*TODO*]

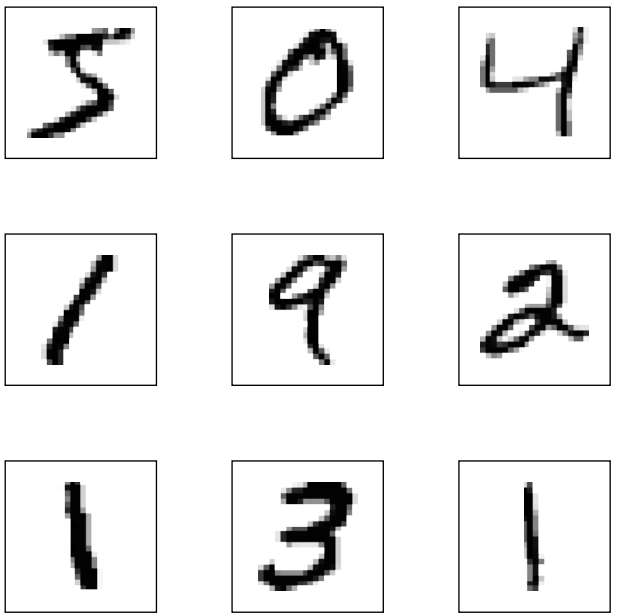
Chapter ML:IX (continued)

IX. Deep Learning

- Introduction to Deep Learning
- Autoencoder Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long-Term Dependencies
- RNNs for Machine Translation
- Attention Mechanism
- Self Attention and Transformers
- Transformer Language Models
- Pretraining

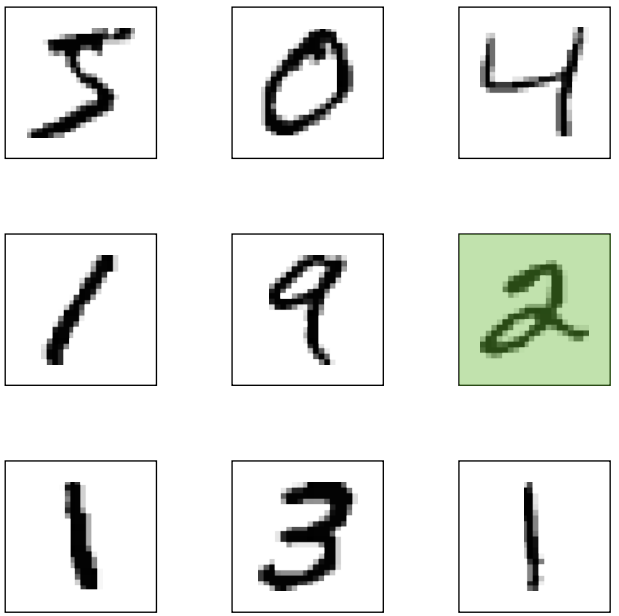
Convolutional Neural Networks

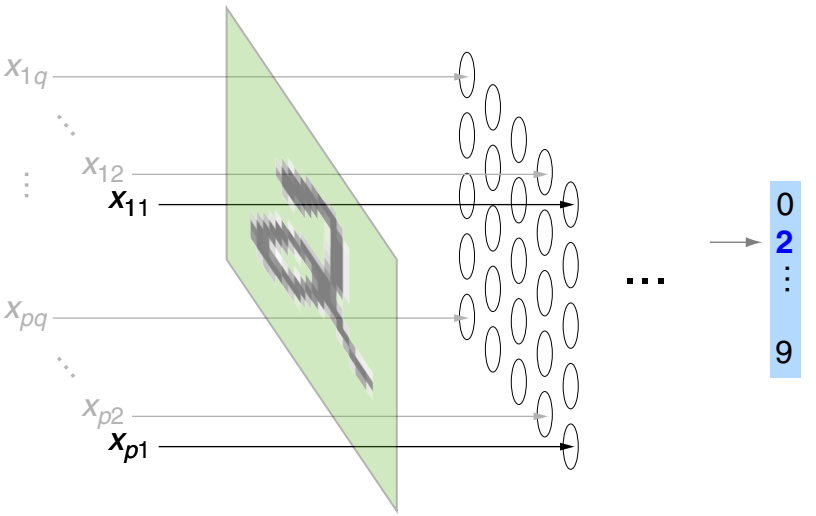
Image Classification Task



Convolutional Neural Networks

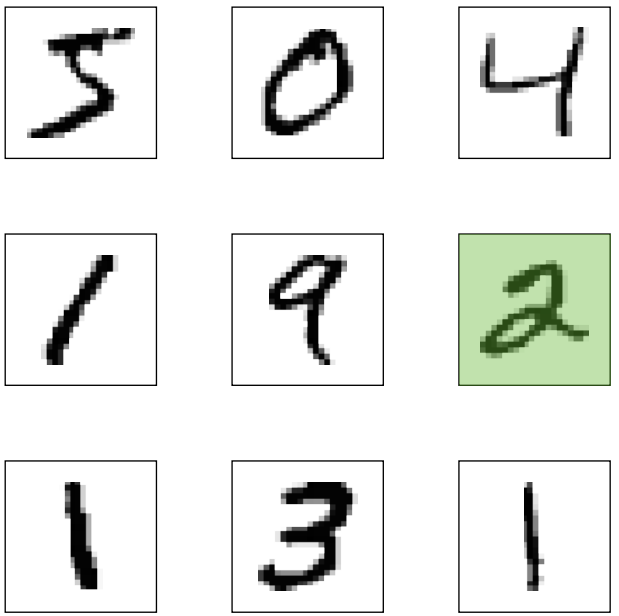
Image Classification Task (continued)

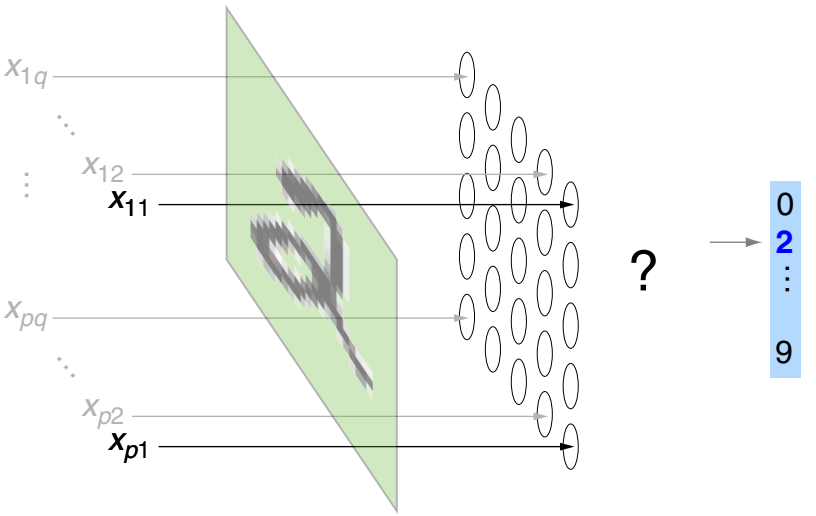


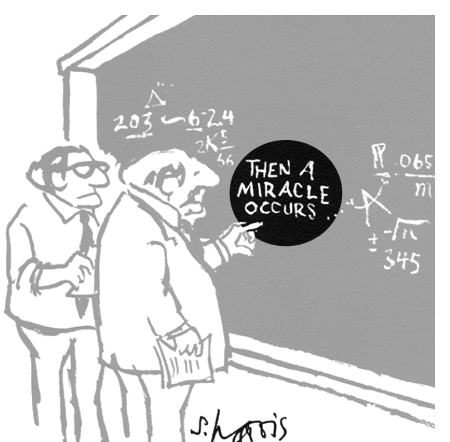


Convolutional Neural Networks

Image Classification Task (continued)

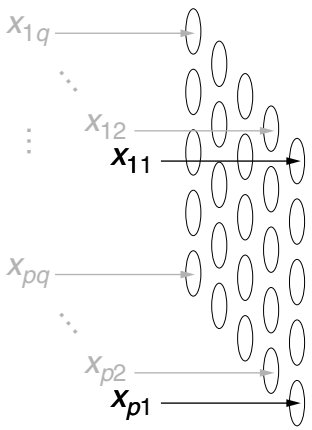






Convolutional Neural Networks

Image Classification with CNNs

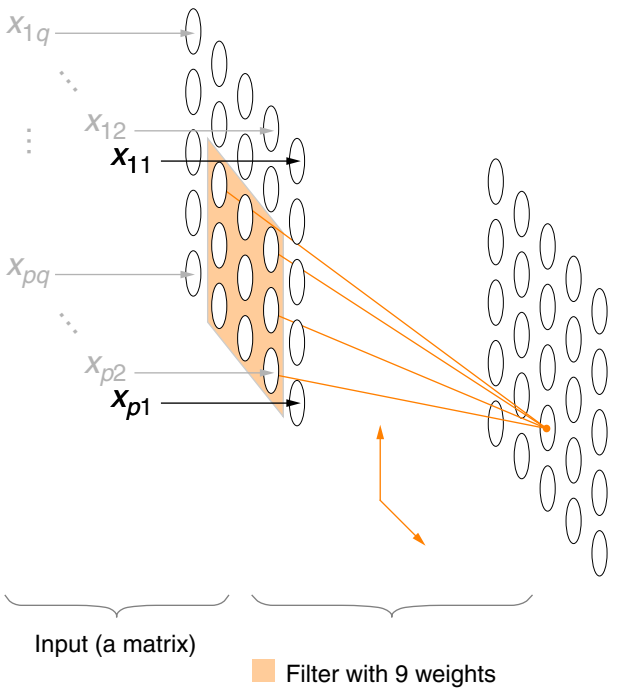


Input (a matrix)

$$\mathrm{y}\left(\begin{pmatrix} x_{11}\ldots x_{1q} \\ \vdots \\ x_{p1}\ldots x_{pq} \end{pmatrix}\right) = \boldsymbol{\sigma}\left(W^{\mathrm{o}}\begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^{\mathrm{h}}\,\mathbf{y}^{\mathrm{c}}\right) \end{pmatrix}\right)$$

Convolutional Neural Networks

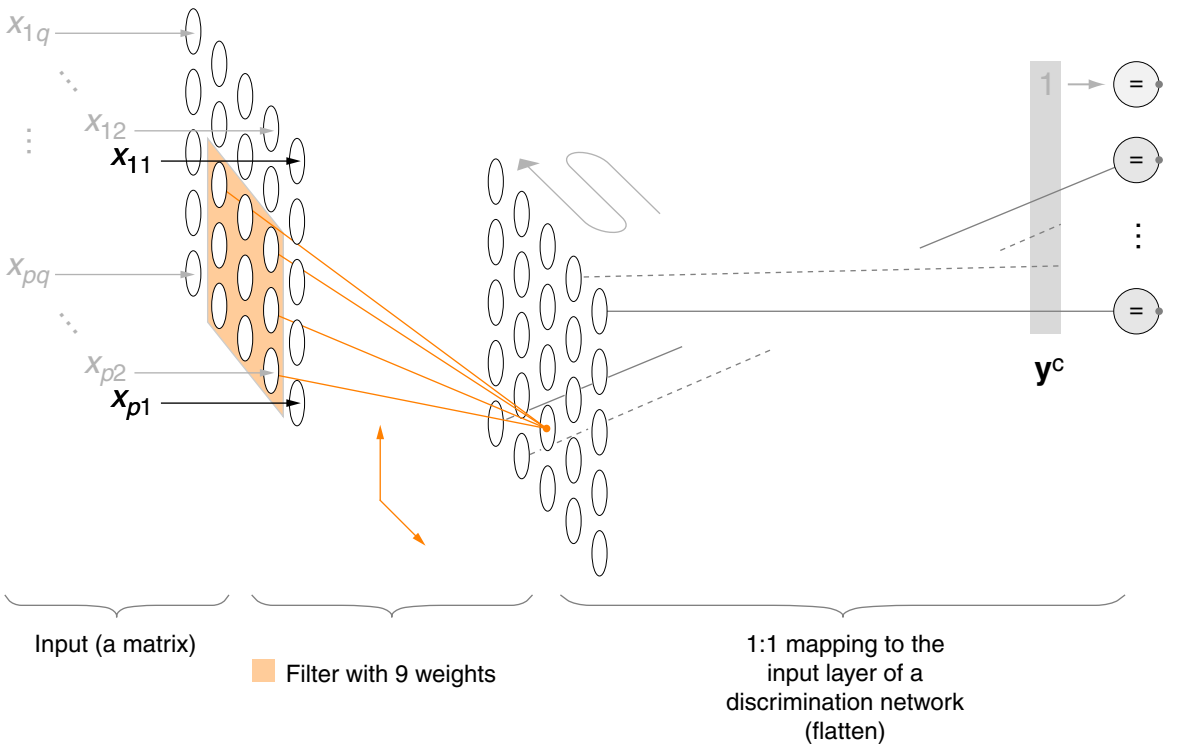
Image Classification with CNNs (continued)



$$\mathrm{y}\left(\begin{pmatrix} x_{11} \ldots x_{1q} \\ \vdots \\ x_{p1} \ldots x_{pq} \end{pmatrix}\right) = \boldsymbol{\sigma}\left(W^{\mathrm{o}}\begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^{\mathrm{h}}\,\mathbf{y}^{\mathrm{c}}\right) \end{pmatrix}\right)$$

Convolutional Neural Networks

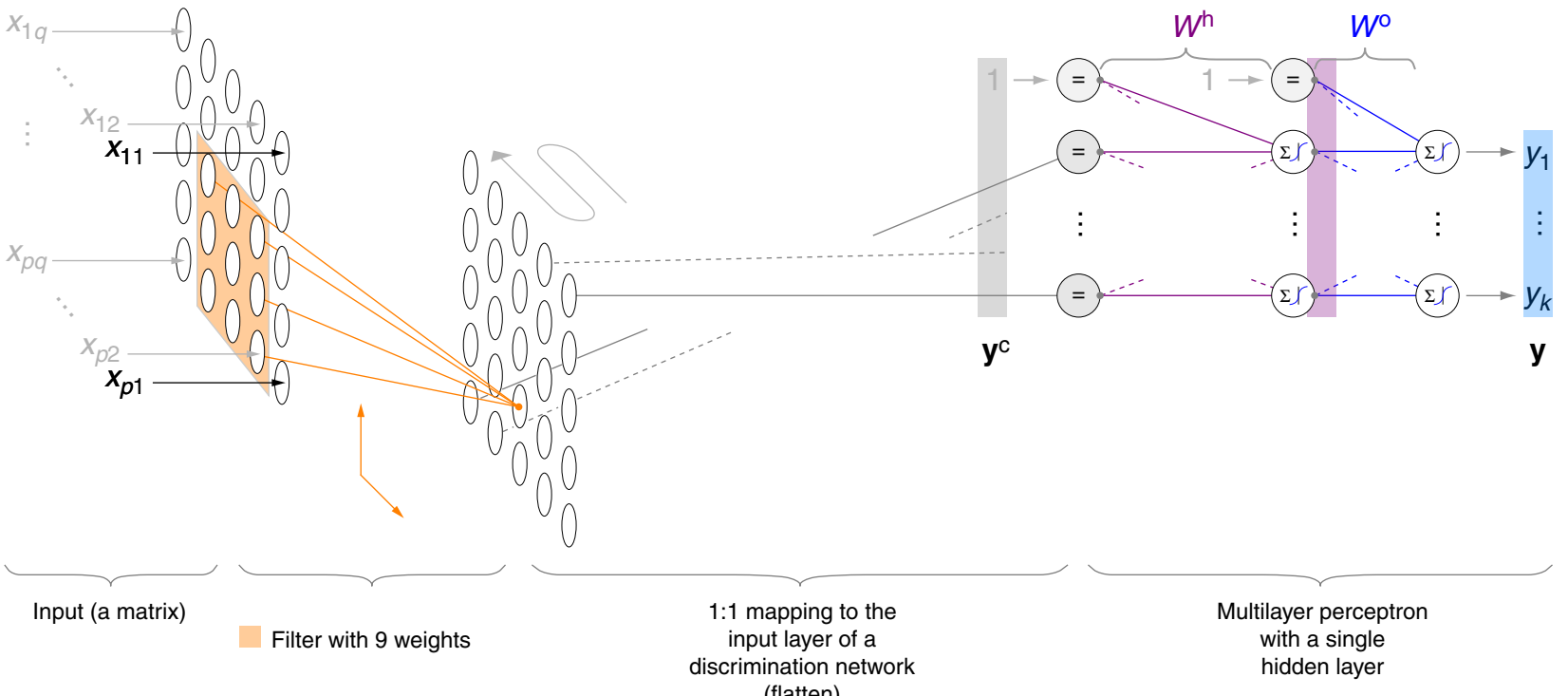
Image Classification with CNNs (continued)



$$\mathrm{y}\left(\begin{pmatrix} x_{11}\ldots x_{1q} \\ \vdots \\ x_{p1}\ldots x_{pq} \end{pmatrix}\right) = \boldsymbol{\sigma}\left(W^{\mathrm{o}}\begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^{\mathrm{h}}\,\mathbf{y}^{\mathrm{c}}\right) \end{pmatrix}\right)$$

Convolutional Neural Networks

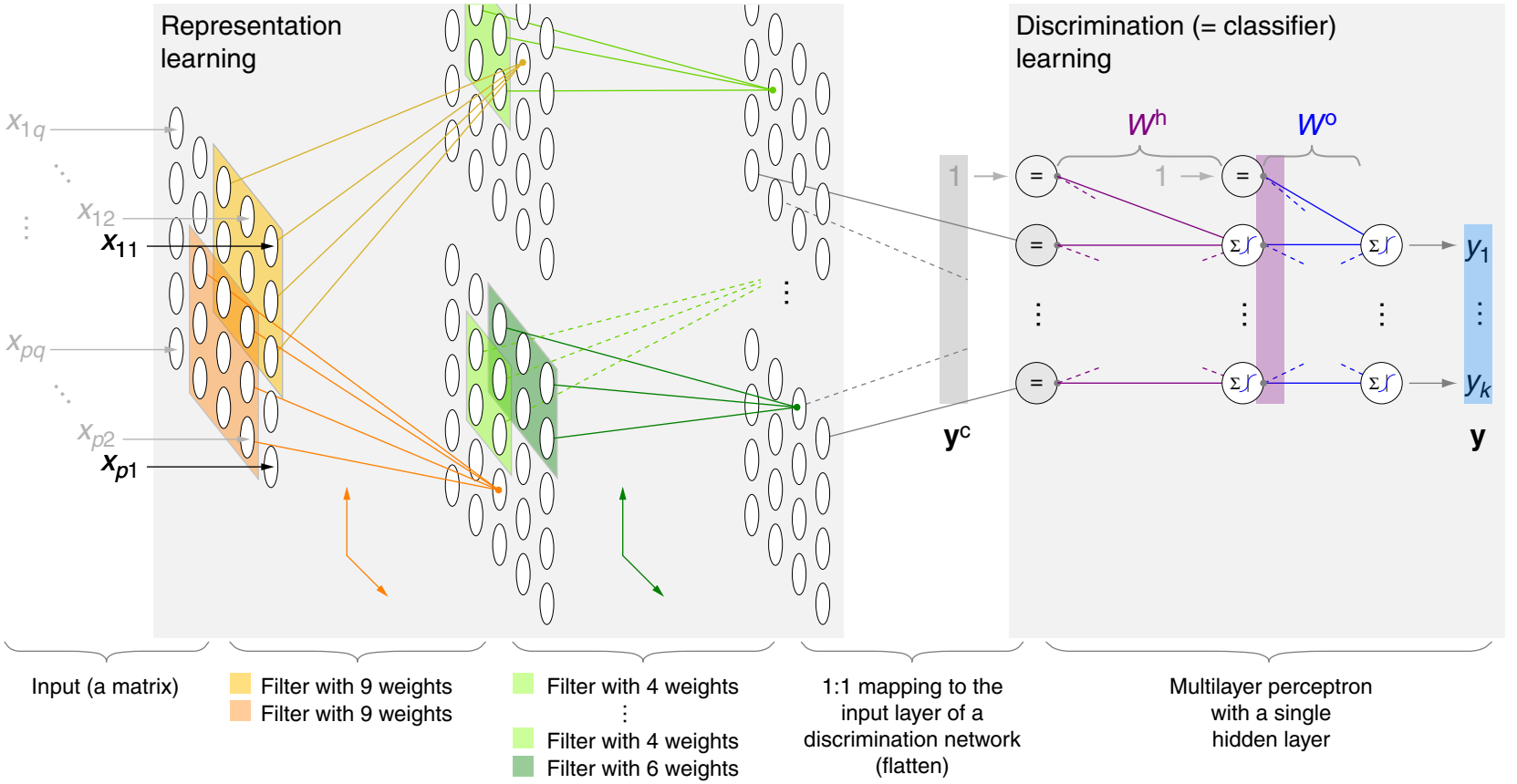
Image Classification with CNNs (continued)



$$\mathbf{y}\left(\begin{pmatrix} x_{11}\ldots x_{1q} \\ \vdots \\ x_{p1}\ldots x_{pq} \end{pmatrix}\right) = \boldsymbol{\sigma}\left(W^{\mathrm{o}}\begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^{\mathrm{h}}\,\mathbf{y}^{\mathrm{c}}\right) \end{pmatrix}\right)$$

Convolutional Neural Networks

Image Classification with CNNs (continued)

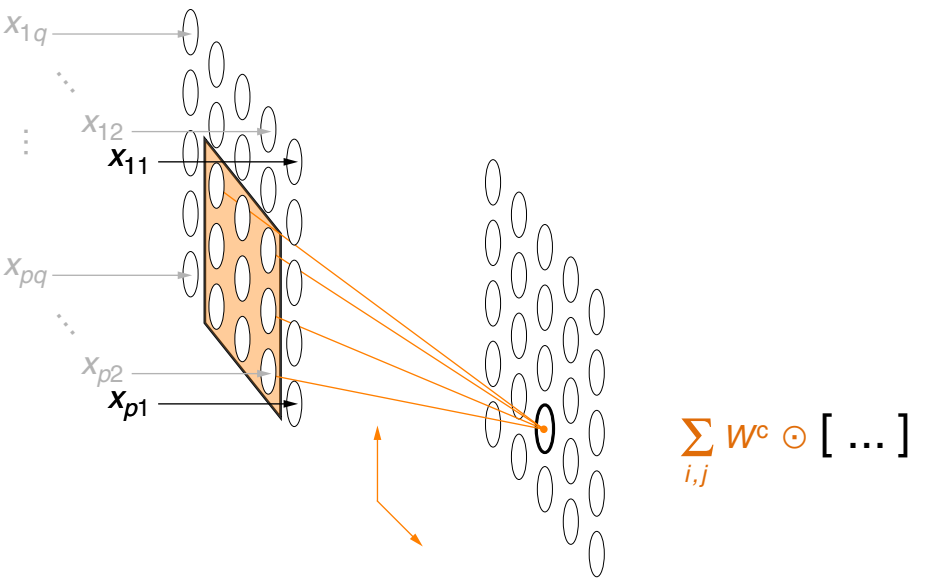


$$\mathbf{y}\left(\begin{pmatrix} x_{11}\ldots x_{1q} \\ \vdots \\ x_{p1}\ldots x_{pq} \end{pmatrix}\right) = \boldsymbol{\sigma}\left(W^o\begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^h\,\mathbf{y}^c\right) \end{pmatrix}\right)$$

Convolutional Neural Networks

Image Classification with CNNs (continued)

A convolutional filter computes of a fixed number of input “pixels” a weighted sum:



The vector $\mathbf{y}^c$ results from concatenating the outputs of the final layer of filters.

Remarks (computation of $\mathbf{y}^{\mathsf{c}}$) :

- For a CNN with input size $p \times q$ and a single filter with weights W^{c} of size $p_c \times q_c$,

$$\mathbf{y}^{\mathsf{c}} = \begin{pmatrix} & & 1 & & \\ \text{flatten}\left(\begin{pmatrix} \sum_{i,j} W^{\mathsf{c}} \odot \begin{pmatrix} x_{1\,1} & \cdots & x_{1\,q_c} \\ & \vdots & \\ x_{p_c\,1} & \cdots & x_{p_c\,q_c} \end{pmatrix} & \cdots & \sum_{i,j} W^{\mathsf{c}} \odot \begin{pmatrix} x_{1\,(q-q_c)} & \cdots & x_{1\,q} \\ & \vdots & \\ x_{p_c\,(q-q_c)} & \cdots & x_{p_c\,q} \end{pmatrix} \\ \vdots & & \\ \sum_{i,j} W^{\mathsf{c}} \odot \begin{pmatrix} x_{(p-p_c)\,1} & \cdots & x_{(p-p_c)\,q_c} \\ & \vdots & \\ x_{p\,1} & \cdots & x_{p\,q_c} \end{pmatrix} & \cdots & \sum_{i,j} W^{\mathsf{c}} \odot \begin{pmatrix} x_{(p-p_c)\,(q-q_c)} & \cdots & x_{(p-p_c)\,q} \\ & \vdots & \\ x_{p\,(q-q_c)} & \cdots & x_{p\,q} \end{pmatrix} \end{pmatrix}\right) \end{pmatrix},$$

  where

$$\text{flatten}\left(\begin{pmatrix} z_{1\,1} & \cdots & z_{1\,l} \\ & \vdots & \\ z_{k\,1} & \cdots & z_{k\,l} \end{pmatrix}\right) = (z_{11}, z_{12}, \ldots, z_{k\,l-1}, z_{kl})^T$$

- Recap. The symbol »$\odot$« denotes the Hadamard product, also known as the element-wise or the Schur product. It is a binary operation that takes two matrices of the same dimensions and produces another matrix of the same dimension as the operands, where each element is the product of the respective elements of the two original matrices. [Wikipedia]

Remarks (technical variants and improvements) :

- Each convolutional filter in a CNN has several hyperparameters that can be adjusted independently, such as:
  - the number of the weights (= number of inputs on the previous layer),
  - how the boundary of the input is handled (e.g. padding),
  - the step size (“stride”) between adjacent input regions mapped by the filter.
- In CNN architectures, convolutional layers are commonly interleaved with downsampling layers which, instead of applying a filter with learned parameters, map an input region to its maximum (“max-pooling”) or average (“average-pooling”).
- Modern CNN architectures commonly use skip connections, which create additional, shorter paths through the network to help alleviate the vanishing gradient problem. This is implemented by adding or concatenating the activations of a given layer to several layers deeper in the network.
- The best-performing CNN models for image classification tasks can be more than 100 layers deep, with hundreds of millions of parameters. However, current state of the art in image classification tends to be dominated by transformer-based architectures, many of which do not use convolutional filters explicitly, but learn similar operations from large amounts of training data. [PapersWithCode: Classification on ImageNet]

Remarks (learning strategies) :

- Representation and discrimination learning can follow two strategies:
  1. Decoupled. Learn [the filter weights of] a representation that maps matrices on vectors $\mathbf{y}^c$, also called an “embedding”. Afterwards, learn [the weights of] a classifier that maps the vectors $\mathbf{y}^c$ to the possible classes. Autoencoders are one possibility to learn a representation.
  2. Combined. Learn [the filter weights of] a representation along with [the weights of] a classifier.

- The “decoupled” approach is particularly suited if the set of examples, X, is large enough to create a suitable embedding (= to compute the filter weights), but the number of labeled examples thereof is rather small. In its extreme form an existing embedding from a different dataset is used instead of computing a specific embedding from the actual data set X. Keywords: few-shot learning, one-shot learning

- In the “combined” approach the classification errors (or losses) are propagated back into the representation learning layers as well. Hence the embedding will encode information about (= will be tailored to) the task. This kind of “end-to-end” learning is very effective but requires sufficient of both ground truth data (labeled examples) and computing resources.