Statistical Machine Learning

Lecture 10
Convolutional neural networks
How to train neural networks

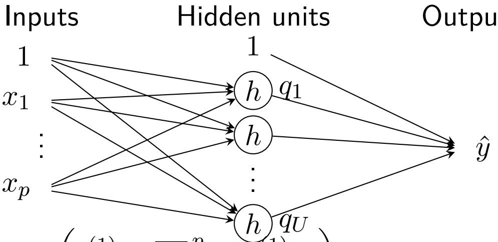
UPPSALA
UNIVERSITET

Niklas Wahlström
Division of Systems and Control
Department of Information Technology
Uppsala University

niklas.wahlstrom@it.uu.se
www.it.uu.se/katalog/nikwa778

Summary of Lecture 9 (I/IV)
Neural network

A neural network is a sequential construction of **several** generalized linear regression models.



$$q_1 = h\left(b_1^{(1)} + \sum_{j=1}^{p} w_{1j}^{(1)} x_j\right)$$

$$q_2 = h\left(b_2^{(1)} + \sum_{j=1}^{p} w_{2j}^{(1)} x_j\right)$$

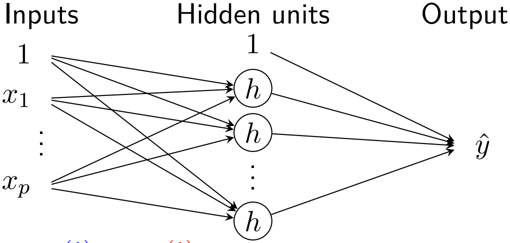
$$\vdots$$

$$q_U = h\left(b_U^{(1)} + \sum_{j=1}^{p} w_{Uj}^{(1)} x_j\right)$$

$$\hat{y} = b^{(2)} + \sum_{i=1}^{U} w_i^{(2)} q_i$$

Summary of Lecture 9 (I/IV)
Neural network

A neural network is a sequential construction of **several** generalized linear regression models.



$$\mathbf{q} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\hat{y} = \mathbf{W}^{(2)}\mathbf{q} + \mathbf{b}^{(2)}$$

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \dots & w_{1p}^{(1)} \\ \vdots & & \vdots \\ w_{U1}^{(1)} & \dots & w_{Up}^{(1)} \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_U^{(1)} \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_U \end{bmatrix}$$

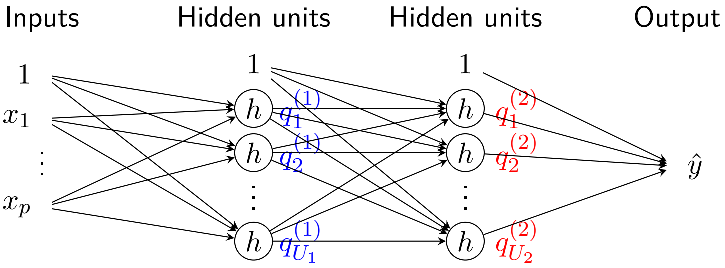
Weight matrix   Offset vector   Hidden units

$$\mathbf{b}^{(2)} = \begin{bmatrix} b^{(2)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_1^{(2)} & \dots & w_U^{(2)} \end{bmatrix}$$

Summary of Lecture 9 (I/IV)
Neural network

A neural network is a **sequential** construction of several generalized linear regression models.



$$\mathbf{q}^{(1)} = h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{q}^{(2)} = h(\mathbf{W}^{(2)}\mathbf{q}^{(1)} + \mathbf{b}^{(2)})$$
$$\hat{y} = \mathbf{W}^{(3)}\mathbf{q}^{(2)} + \mathbf{b}^{(3)}$$

In dense (or fully-connected) layers all input units are connected to all output units.

Summary of Lecture 9 (II/IV)

Parameters = weight matrices and offset vectors
All weight matrices and offset vectors in all layers combined are the parameters of the network

$$\boldsymbol{\theta} = \left[\mathrm{vec}(\mathbf{W}^{(1)})^{\mathsf{T}}, \;\; \mathbf{b}^{(1)\mathsf{T}}, \;\; \dots, \;\; \mathrm{vec}(\mathbf{W}^{(L)})^{\mathsf{T}}, \;\; \mathbf{b}^{(L)\mathsf{T}}\right]^{\mathsf{T}},$$

which constitutes the parametric model $\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$.

Training
We train a network on training data $\{\mathbf{x}_i, y_i\}_{i=1}^n$ by considering the optimization problem

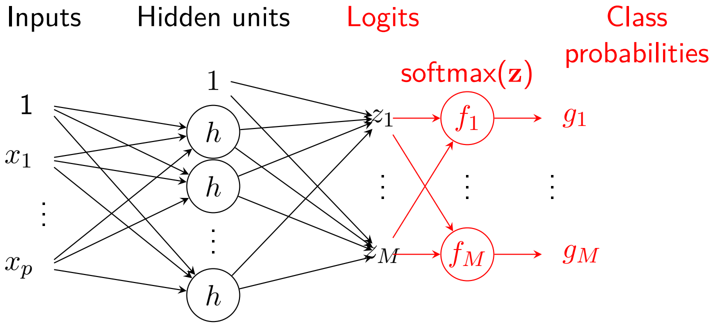
$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

where $\hat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$

Summary of Lecture 9 (III/IV) NN for classification ($M > 2$ classes)

For $M > 2$ classes we want to predict the class probability for all M classes $g_m = p(y = m|\mathbf{x})$. We extend the logistic function to the **softmax activation function**

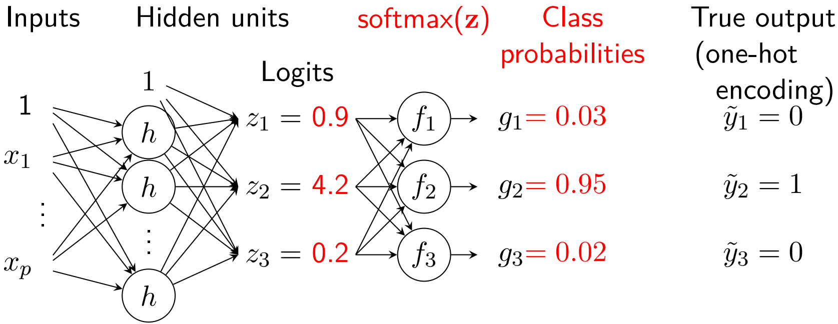
$$g_m = f_m(\mathbf{z}) = \frac{e^{z_m}}{\sum_{l=1}^{M} e^{z_l}}, \qquad m = 1, \dots, M.$$



softmax($\mathbf{z}$) $= [f_1(\mathbf{z}), \dots, f_M(\mathbf{z})]^\mathsf{T}$

Summary of Lecture 9 (IV/IV)
Example $M = 3$ classes

Consider an example with three classes $M = 3$ and where $y = 2$.



Inputs

Hidden units

Logits

softmax($\mathbf{z}$)

Class probabilities

True output (one-hot encoding)

$z_1 = 0.9$ | $g_1 = 0.03$ | $\tilde{y}_1 = 0$

$z_2 = 4.2$ | $g_2 = 0.95$ | $\tilde{y}_2 = 1$

$z_3 = 0.2$ | $g_3 = 0.02$ | $\tilde{y}_3 = 0$

The network is trained by minimizing the **cross-entropy**

$$L(\tilde{\mathbf{y}}, \mathbf{g}) = -\sum_{m=1}^{M} \tilde{y}_m \ln(g_m) = -\ln 0.95 = 0.05$$

Outline

1. **Previous lecture** The neural network model
   - Neural network for regression
   - Neural network for classification
2. **This lecture**
   - Convolutional neural network
   - How to train a neural network

Convolutional neural networks

Convolutional neural networks (CNN) are a special kind neural networks tailored for problems where the input data has a grid-like structure.

Examples

- Digital images (2D grid of pixels)
- Audio waveform data (1D grid, times series)
- Volumetric data e.g. CT scans (3D grid)

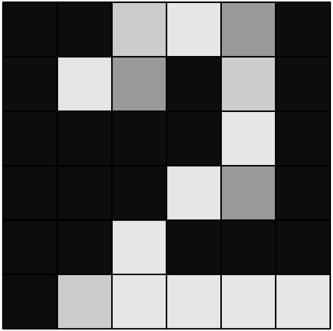
The description here will focus on images.

Data representation of images

Consider a grayscale image of 6×6 **pixels**.

- Each pixel value represents the color. The value ranges from 0 (total absence, black) to 1 (total presence, white)
- The pixels are the input variables $x_{1,1}, x_{1,2}, \dots, x_{6,6}$.

Image



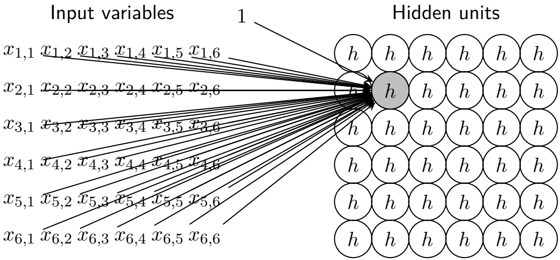
Data representation

0.0	0.0	0.8	0.9	0.6	0.0
0.0	0.9	0.6	0.0	0.8	0.0
0.0	0.0	0.0	0.0	0.9	0.0
0.0	0.0	0.0	0.9	0.6	0.0
0.0	0.0	0.9	0.0	0.0	0.0
0.0	0.8	0.9	0.9	0.9	0.9

The convolutional layer

Consider a hidden layer with 6×6 hidden units.

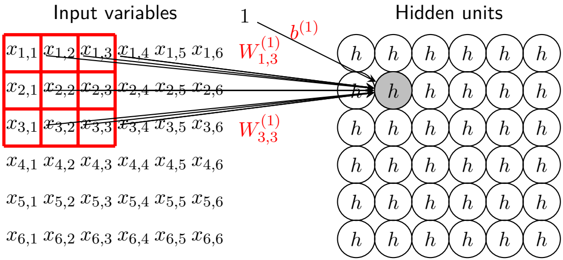
- **Dense layer** (previous lecture): Each hidden unit is connected with **all pixels**. Each pixel-hidden-unit-pair has its own **unique parameter**.


Input variables
1
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
h

The convolutional layer

Consider a hidden layer with 6×6 hidden units.

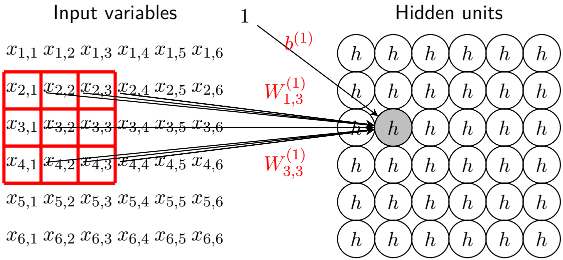
- **Dense layer** (previous lecture): Each hidden unit is connected with **all pixels**. Each pixel-hidden-unit-pair has its own **unique parameter**.
- **Convolutional layer**: Each hidden unit is connected with a **region of pixels** via a set of parameters, so-called **filter**. Different hidden units have the **same set of parameters**.


Input variables
1
$b^{(1)}$
Hidden units
$W^{(1)}_{1,3}$
$W^{(1)}_{3,3}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
h

The convolutional layer

Consider a hidden layer with 6×6 hidden units.

- **Dense layer** (previous lecture): Each hidden unit is connected with **all pixels**. Each pixel-hidden-unit-pair has its own **unique parameter**.
- **Convolutional layer**: Each hidden unit is connected with a **region of pixels** via a set of parameters, so-called **filter**. Different hidden units have the **same set of parameters**.



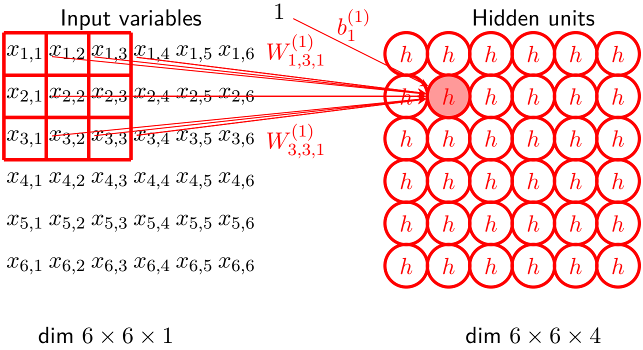
Input variables

$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$

Conv. layer uses **sparse interactions** and **parameter sharing**

Multiple filters

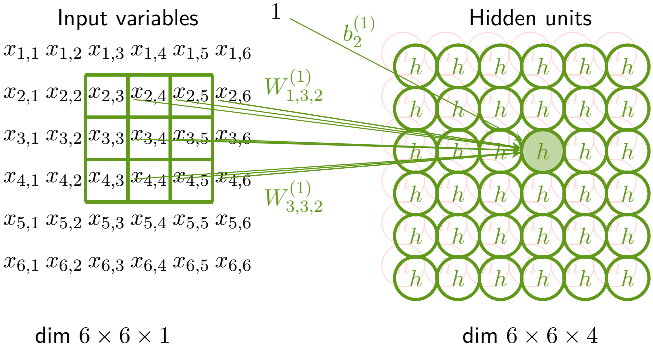
- One filter per layer does not give enough flexibility. ⇒
- We use **multiple filters** (visualized with different colors).
- Each filter produces its own set of hidden units – a **channel**.



Input variables
Hidden units
1
$b_1^{(1)}$
$W_{1,3,1}^{(1)}$
$W_{3,3,1}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
h
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

Multiple filters

- One filter per layer does not give enough flexibility. ⇒
- We use **multiple filters** (visualized with different colors).
- Each filter produces its own set of hidden units – a **channel**.



Input variables

$$\begin{matrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} \\ x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} & x_{5,6} \\ x_{6,1} & x_{6,2} & x_{6,3} & x_{6,4} & x_{6,5} & x_{6,6} \end{matrix}$$

1

$b_2^{(1)}$

$W_{1,3,2}^{(1)}$

$W_{3,3,2}^{(1)}$

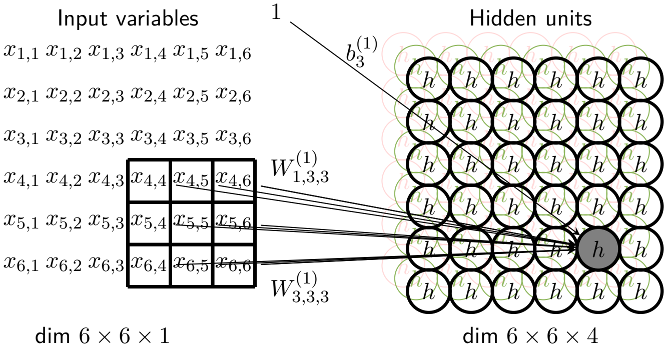
Hidden units

dim $6 \times 6 \times 1$

dim $6 \times 6 \times 4$

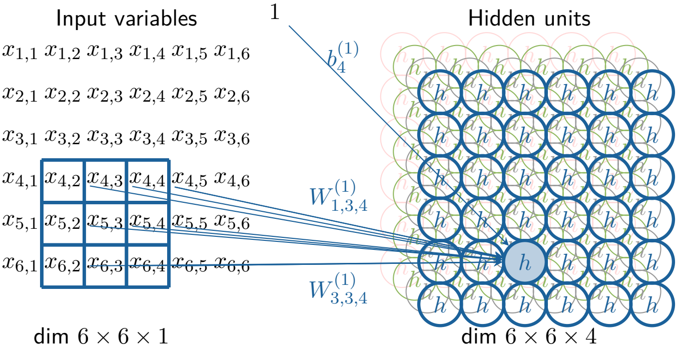
Multiple filters

- One filter per layer does not give enough flexibility. ⇒
- We use **multiple filters** (visualized with different colors).
- Each filter produces its own set of hidden units – a **channel**.


Input variables
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
dim $6 \times 6 \times 1$
1
$b_3^{(1)}$
$W_{1,3,3}^{(1)}$
$W_{3,3,3}^{(1)}$
Hidden units
h
dim $6 \times 6 \times 4$

Multiple filters

- One filter per layer does not give enough flexibility. ⇒
- We use **multiple filters** (visualized with different colors).
- Each filter produces its own set of hidden units – a **channel**.



Input variables

$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$
$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$
$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$
$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,4}$	$x_{4,5}$	$x_{4,6}$
$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,4}$	$x_{5,5}$	$x_{5,6}$
$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,4}$	$x_{6,5}$	$x_{6,6}$

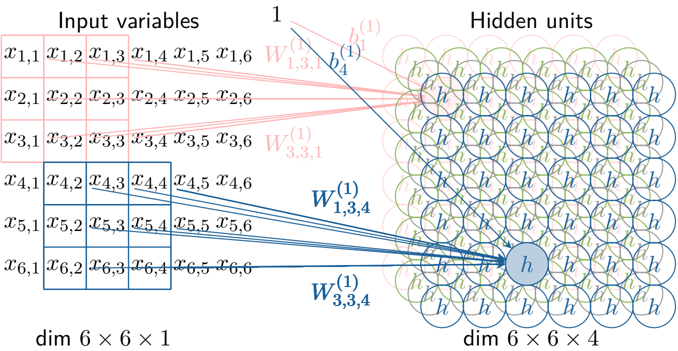
dim $6 \times 6 \times 1$

Hidden units

dim $6 \times 6 \times 4$

Multiple filters

- One filter per layer does not give enough flexibility. ⇒
- We use **multiple filters** (visualized with different colors).
- Each filter produces its own set of hidden units – a **channel**.


Input variables
1
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$W^{(1)}_{1,3,1}$
$b^{(1)}_1$
$b^{(1)}_4$
$W^{(1)}_{3,3,1}$
$\boldsymbol{W^{(1)}_{1,3,4}}$
$\boldsymbol{W^{(1)}_{3,3,4}}$
h
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

Hidden layers are organized in **tensors** of size (rows × columns × channels).

What is a tensor?

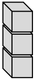
A **tensor** is a generalization of scalar, vector and matrix to arbitrary **order**.

Scalar
order **0**

$$a = 3$$

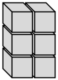
Vector
order **1**

$$\mathbf{b} = \begin{bmatrix} 3 \\ -2 \\ -1 \end{bmatrix}$$



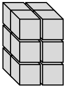
Matrix
order **2**

$$W = \begin{bmatrix} 3 & 2 \\ -2 & 1 \\ -1 & 2 \end{bmatrix}$$



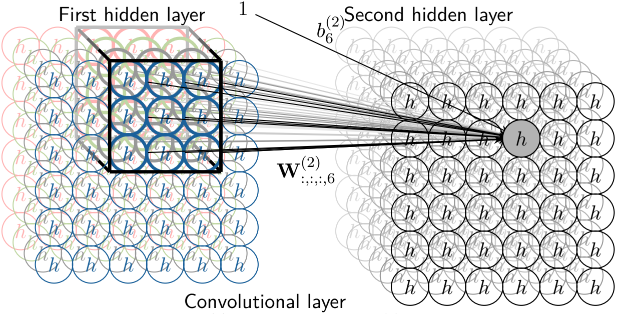
Tensor
any order
(here order **3**)

$$\mathbf{T}_{:,:,1} = \begin{bmatrix} 3 & 2 \\ -2 & 1 \\ -1 & 2 \end{bmatrix}, \quad \mathbf{T}_{:,:,2} = \begin{bmatrix} -1 & 4 \\ 1 & 2 \\ -5 & 3 \end{bmatrix}$$



Multiple filters (cont.)

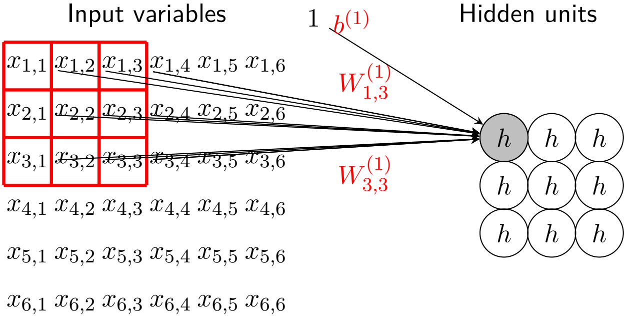
- A filter operates on **all channels** in a hidden layer.
- Each filter has the dimension (filter rows × filter colomns × input channels), here (3 × 3 × 4).
- We stack all filter parameters in a **weight tensor** with dimensions (filter rows × filter columns × input channels × output channels), here (3 × 3 × 4 × 6)



Convolutional layer
$\mathbf{W}^{(2)} \in \mathbb{R}^{3\times3\times4\times6}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{6}$

Condensing information with strides

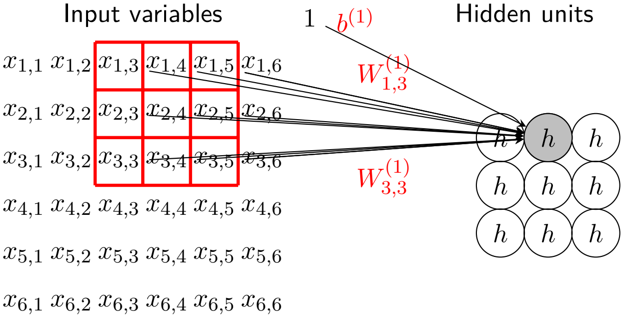
- **Problem**: As we proceed though the network we want to condense the information.
- **Solution**: Apply the filter to every second pixel. We use a **stride** of 2 (instead of 1).


Input variables
1
$b^{(1)}$
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$W^{(1)}_{1,3}$
$W^{(1)}_{3,3}$
h h h
h h h
h h h

With stride 2 we get half the number of rows and columns in the hidden layer.

Condensing information with strides

- **Problem**: As we proceed though the network we want to condense the information.
- **Solution**: Apply the filter to every second pixel. We use a **stride** of 2 (instead of 1).

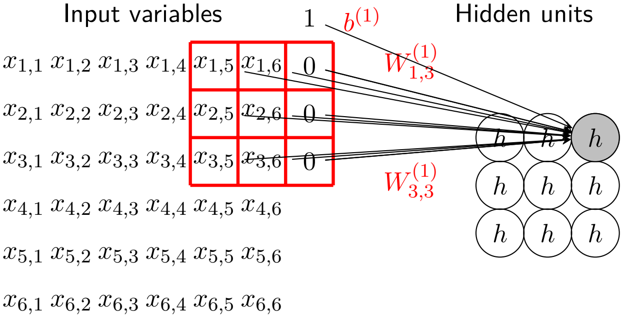


Input variables
1
$b^{(1)}$
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$W^{(1)}_{1,3}$
$W^{(1)}_{3,3}$
h h h
h h h
h h h

With stride 2 we get half the number of rows and columns in the hidden layer.

Condensing information with strides

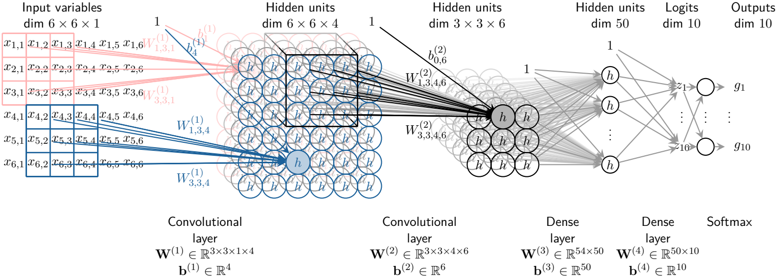
- **Problem**: As we proceed though the network we want to condense the information.
- **Solution**: Apply the filter to every second pixel. We use a **stride** of 2 (instead of 1).


Input variables
1
$b^{(1)}$
Hidden units
$W^{(1)}_{1,3}$
$W^{(1)}_{3,3}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ 0
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$ 0
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$ 0
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
h h h
h h h
h h h

With stride 2 we get half the number of rows and columns in the hidden layer.

Full CNN architecture

- A full CNN usually consist of multiple convolutional layers (here two) and a few final dense layers (here two).
- If we have a classification problem at hand, we end with a softmax activation function to produce class probabilities.


Input variables
dim $6 \times 6 \times 1$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
1
$b_1^{(1)}$
$W_{1,3,1}^{(1)}$
$b_4^{(1)}$
$W_{3,3,1}^{(1)}$
$W_{1,3,4}^{(1)}$
$W_{3,3,4}^{(1)}$
Hidden units
dim $6 \times 6 \times 4$
h
1
$b_{0,6}^{(2)}$
$W_{1,3,4,6}^{(2)}$
$W_{3,3,4,6}^{(2)}$
Hidden units
dim $3 \times 3 \times 6$
Hidden units
dim 50
1
Logits
dim 10
z_1
z_{10}
Outputs
dim 10
g_1
g_{10}
Convolutional
layer
$\mathbf{W}^{(1)} \in \mathbb{R}^{3\times3\times1\times4}$
$\mathbf{b}^{(1)} \in \mathbb{R}^{4}$
Convolutional
layer
$\mathbf{W}^{(2)} \in \mathbb{R}^{3\times3\times4\times6}$
$\mathbf{b}^{(2)} \in \mathbb{R}^{6}$
Dense
layer
$\mathbf{W}^{(3)} \in \mathbb{R}^{54\times50}$
$\mathbf{b}^{(3)} \in \mathbb{R}^{50}$
Dense
layer
$\mathbf{W}^{(4)} \in \mathbb{R}^{50\times10}$
$\mathbf{b}^{(4)} \in \mathbb{R}^{10}$
Softmax

Here we use 50 hidden units in the last hidden layer and consider a classification problem with $M = 10$ classes.

Skin cancer – background

One result on the use of deep learning in medicine - Detecting skin cancer (February 2017)
Andre Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M. and Thrun, S. **Dermatologist-level classification of skin cancer with deep neural networks**. *Nature*, 542, 115–118, February, 2017.

Some background figures (from the US) on skin cancer:

- Melanomas represents less than 5% of all skin cancers, **but** accounts for 75% of all skin-cancer-related deaths.
- Early detection absolutely critical. Estimated 5-year survival rate for melanoma: Over 99% if detected in its earlier stages and 14% is detected in its later stages.

Skin cancer – task

Image copyright Nature (doi:10.1038/nature21056)

Skin cancer – solution (ultrabrief)

In the paper they used the following network architecture

Image copyright Nature doi:10.1038/nature21056)

- Initialize all parameters from a neural network trained on 1.28 million images (**transfer learning**).
- From this initialization we learn new model parameters using $129\,450$ clinical images (~ 100 times more images than any previous study).
- Use the model to predict class based on unseen data.

Skin cancer – indication of the results

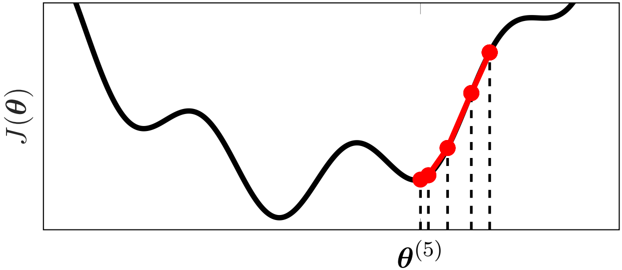
$$\text{sensitivity} = \frac{\text{true positive}}{\text{positive}} \qquad \text{specificity} = \frac{\text{true negative}}{\text{negative}}$$

Image copyright Nature (doi:10.1038/nature21056)

Unconstrained numerical optimization

We train a network by considering the optimization problem

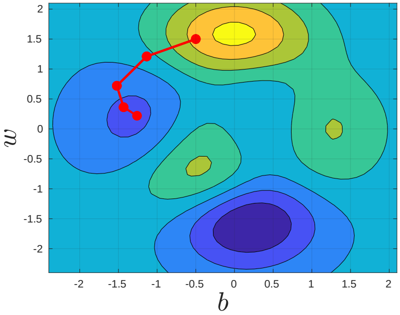
$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \qquad J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$$



We solve the optimization problem by

- … making an initial guess of $\boldsymbol{\theta}$…
- … and updating $\boldsymbol{\theta}$ iteratively.

Iterative solution (gradient descent) - Example 2D



$$\boldsymbol{\theta} = [b,\ w]^{\mathsf{T}} \in \mathbb{R}^2$$

1. Pick a $\boldsymbol{\theta}^{(0)}$
2. while(*not converged*)
   - Update $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma \mathbf{d}^{(t)}$, where $\mathbf{d}^{(t)} = \boldsymbol{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
   - Update $t := t + 1$

We call $\gamma \in \mathbb{R}$ the **step length** or **learning rate**.

Computational challenge 1 - dim(θ) is big

At each optimization step we need to compute the gradient

$$\mathbf{d}^{(t)} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}^{(t)}).$$

Computational challenge 1 - dim(θ) big: A neural network contains a lot of parameters. Computing the gradient is costly.

Solution: A NN is a composition of multiple layers. Hence, each term $\nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})$ can be computed efficiently by repeatedly applying the chain rule. This is called the **back-propagation algorithm**. Not part of the course.

Computational challenge 2 - n is big

At each optimization step we need to compute the gradient

$$\mathbf{d}^{(t)} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}^{(t)}).$$

Computational challenge 2 - n big: We typically use a lot of training data n for training the neural netowork. Computing the gradient is costly.

Solution: For each iteration, we only use a small part of the data set to compute the gradient $\mathbf{d}^{(t)}$. This is called the **stochastic gradient descent**.

Stochastic gradient descent

A big data set is often redundant = many data points are similar.

$$\overbrace{\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \mathbf{x}_5 & \mathbf{x}_6 & \mathbf{x}_7 & \mathbf{x}_8 & \mathbf{x}_9 & \mathbf{x}_{10} & \mathbf{x}_{11} & \mathbf{x}_{12} & \mathbf{x}_{13} & \mathbf{x}_{14} & \mathbf{x}_{15} & \mathbf{x}_{16} & \mathbf{x}_{17} & \mathbf{x}_{18} & \mathbf{x}_{19} & \mathbf{x}_{20} \\ \mathbf{y}_1 & \mathbf{y}_2 & \mathbf{y}_3 & \mathbf{y}_4 & \mathbf{y}_5 & \mathbf{y}_6 & \mathbf{y}_7 & \mathbf{y}_8 & \mathbf{y}_9 & \mathbf{y}_{10} & \mathbf{y}_{11} & \mathbf{y}_{12} & \mathbf{y}_{13} & \mathbf{y}_{14} & \mathbf{y}_{15} & \mathbf{y}_{16} & \mathbf{y}_{17} & \mathbf{y}_{18} & \mathbf{y}_{19} & \mathbf{y}_{20} \\ \hline\end{array}}^{\text{Training data}}$$

If the training data is big

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \textstyle\sum_{i=1}^{\frac{n}{2}} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}) \qquad \text{and}$$

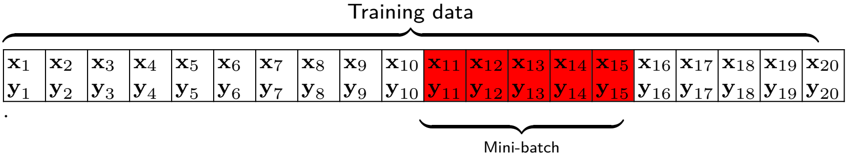
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \textstyle\sum_{i=\frac{n}{2}+1}^{n} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}).$$

We can do the update with only half the computation cost!

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma \frac{1}{n/2} \sum_{i=1}^{\frac{n}{2}} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}^{(t)}),$$

$$\boldsymbol{\theta}^{(t+2)} = \boldsymbol{\theta}^{(t+1)} - \gamma \frac{1}{n/2} \sum_{i=\frac{n}{2}+1}^{n} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}^{(t+1)}).$$

Stochastic gradient descent



Training data

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$	$\mathbf{x}_7$	$\mathbf{x}_8$	$\mathbf{x}_9$	$\mathbf{x}_{10}$	$\mathbf{x}_{11}$	$\mathbf{x}_{12}$	$\mathbf{x}_{13}$	$\mathbf{x}_{14}$	$\mathbf{x}_{15}$	$\mathbf{x}_{16}$	$\mathbf{x}_{17}$	$\mathbf{x}_{18}$	$\mathbf{x}_{19}$	$\mathbf{x}_{20}$
$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_4$	$\mathbf{y}_5$	$\mathbf{y}_6$	$\mathbf{y}_7$	$\mathbf{y}_8$	$\mathbf{y}_9$	$\mathbf{y}_{10}$	$\mathbf{y}_{11}$	$\mathbf{y}_{12}$	$\mathbf{y}_{13}$	$\mathbf{y}_{14}$	$\mathbf{y}_{15}$	$\mathbf{y}_{16}$	$\mathbf{y}_{17}$	$\mathbf{y}_{18}$	$\mathbf{y}_{19}$	$\mathbf{y}_{20}$

Mini-batch

$$\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} - \gamma \frac{1}{5} \sum_{i=11}^{15} \nabla_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta}^{(2)})$$

- The extreme version of this strategy is to use only one data point at each training step (called **online learning**)
- We typically do something in between (not one data point, and not all data). We use a smaller set called **mini-batch**.
- One pass through the training data is called an **epoch**.

Stochastic gradient descent

$\mathbf{x}_7$	$\mathbf{x}_{10}$	$\mathbf{x}_3$	$\mathbf{x}_{20}$	$\mathbf{x}_{16}$	$\mathbf{x}_2$	$\mathbf{x}_1$	$\mathbf{x}_{18}$	$\mathbf{x}_{19}$	$\mathbf{x}_{12}$	$\mathbf{x}_6$	$\mathbf{x}_{11}$	$\mathbf{x}_{17}$	$\mathbf{x}_{15}$	$\mathbf{x}_5$	$\mathbf{x}_{14}$	$\mathbf{x}_4$	$\mathbf{x}_9$	$\mathbf{x}_{13}$	$\mathbf{x}_8$
$\mathbf{y}_7$	$\mathbf{y}_{10}$	$\mathbf{y}_3$	$\mathbf{y}_{20}$	$\mathbf{y}_{16}$	$\mathbf{y}_2$	$\mathbf{y}_1$	$\mathbf{y}_{18}$	$\mathbf{y}_{19}$	$\mathbf{y}_{12}$	$\mathbf{y}_6$	$\mathbf{y}_{11}$	$\mathbf{y}_{17}$	$\mathbf{y}_{15}$	$\mathbf{y}_5$	$\mathbf{y}_{14}$	$\mathbf{y}_4$	$\mathbf{y}_9$	$\mathbf{y}_{13}$	$\mathbf{y}_8$

Iteration: 3

Epoch: 1

- If we pick the mini-batches in order, they might be unbalanced and not representative for the whole data set.
- Therefore, we pick data points **at random** from the training data to form a mini-batch.
- One implementation is to randomly reshuffle the data before dividing it into mini-batches.
- After each epoch we do another reshuffling and another pass through the data set.

Mini-batch gradient descent

The full **stochastic gradient descent** algorithm (a.k.a **mini-batch gradient descent**) is as follows

1. Initialize $\boldsymbol{\theta}^{(0)}$, set $t \leftarrow 1$, choose batch size n_b and number of epochs E.
2. For $i = 1$ to E
   (a) Randomly shuffle the training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$.
   (b) For $j = 1$ to $\frac{n}{n_b}$
      (i) Approximate the gradient of the loss function using the mini-batch $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=(j-1)n_b+1}^{jn_b}$,
      $$\hat{\mathbf{d}}^{(t)} = \frac{1}{n_b} \sum_{i=(j-1)n_b+1}^{jn_b} \boldsymbol{\nabla}_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}.$$
      (ii) Do a gradient step $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \gamma \hat{\mathbf{d}}^{(t)}$.
      (iii) Update the iteration index $t \leftarrow t + 1$.

At each time we get a stochastic approximation of the true gradient $\hat{\mathbf{d}}^{(t)} \approx \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{\nabla}_{\boldsymbol{\theta}} L(\mathbf{x}_i, \mathbf{y}_i, \boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$, hence the name.

Summary

1. **Previous lecture** The neural network model
   - Neural network for regression
   - Neural network for classification
2. **This lecture**
   - Convolutional neural network
   - How to train a neural network

A few concepts to summarize lecture 9

Convolutional neural network (CNN): A NN with a particular structure tailored for input data with a grid-like structure, like for example images.

Filter: (a.k.a kernel) A set of parameters that is convolved with a hidden layer. Each filter produces a new channel.

Channel: A set of hidden units produced by the same filter. Each hidden layer consists of one or more channels.

Stride: A positive integer deciding how many steps to move the filter during the convolution.

Tensor: A generalization of matrices to arbitrary order.

Gradient descent: An iterative optimization algorithm where we at iteration take a step proportional to the negative gradient.

Learning rate: (a.k.a step length). A scalar tuning parameter deciding the length of each gradient step in gradient descent.

Stochastic gradient descent (SGD): A version of gradient descent where we at each iteration only use a small part of the training data (a mini-batch).

Mini-batch: The group of training data that we use at each iteration in SG

Batch size: The number of data points in one mini-batch