Statistical Machine Learning

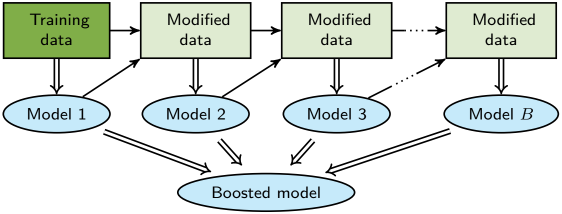
Lecture 8 – Deep learning and neural networks

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

Email: niklas.wahlstrom@it.uu.se

Summary of Lecture 7 (I/III)

Boosting is a sequential ensemble method for regression or classification.



The models are built **sequentially** such that each models tries to **correct the mistakes** made by the previous one!

Contrary to bagging, boosting methods **reduce the bias** of their base models. Works well with e.g. shallow trees.

Summary of Lecture 7 (II/III)

AdaBoost is a boosting method for binary classification. The AdaBoost classifier can be written as

$$\widehat{y}^{B}_{\mathsf{boost}}(\mathbf{x}) = \mathsf{sign}\left(\sum_{b=1}^{B} \alpha^b \widehat{y}^b(\mathbf{x})\right).$$

where $\widehat{y}^b(\mathbf{x})$, the bth base classifier, predicts either -1 or 1, and α^b is a nonnegative real number representing this classifiers "confidence" in its prediction.

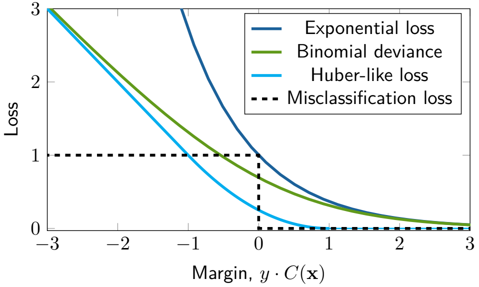
The base classifiers and confidence coefficients are found by sequentially (and greedily) minimizing the **exponential loss** which, for a classifier $\widehat{y}(\mathbf{x}) = \mathsf{sign}\{C(\mathbf{x})\}$, is defined as,

$$L(y, C(\mathbf{x})) = \exp(-y \cdot C(\mathbf{x}))$$

where $y \cdot C(\mathbf{x})$ is referred to the **margin.**

Summary of Lecture 7 (III/III)

The exponential loss function is computationally convenient, but sensitive to **outliers**. In practice it is often preferable to use a **robust loss function** with a more gentle slope for negative margins.



Gradient boosting can be used to learn a boosting ensemble for general differentiable loss functions.

Practical info

1. **The laboratory work**
    - Format: 4h mandatory computer lab about deep learning. Approved on spot, no report.
    - Available slots: March 4, 13:15-17:00, March 7, 8:15-12:00, March 8, 8:15-12:00 and 13:15-17:00.
    - **Sign up at studentportalen**
    - Lab-pm and code available from the homepage. Take a printed lab-pm before you leave and bring it to the lab session.
    - Use the lab computers or bring your own laptop with Tensorflow installed and running.
    - Do the preparatory exercises **before** the lab session.
2. **The exam**
    - You may bring one A4 handwritten sheet of paper, mathematics handbook and a calculator.
    - Questions from any part of the course may be on the exam.
    - **Don't forget to register!**

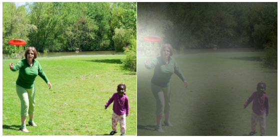
UPPSALA UNIVERSITET

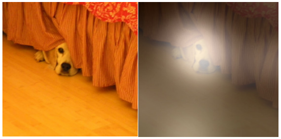
Contents – Lecture 8

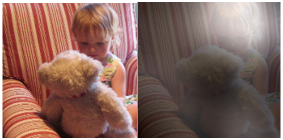
1. **This lecture:** The neural network model
    - Neural network for regression
    - Neural network for classification
    - Convolutional neural network
2. **Next lecture:** How to train a neural network

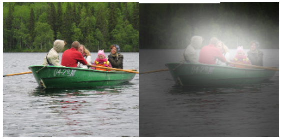
Deep Learning Example: Automatic caption generation

Generate caption automatically from images


A woman is throwing a frisbee in a park.


A dog is standing on a hardwood floor.


A little girl sitting on a bed with a teddy bear.


A group of people sitting on a boat in the water.

Xu, K., Lei Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R. Richard S. Zemel, R. S., and Bengio, Y. **Show, attend and tell: neural image caption generation with visual attention**. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, July, 2015.

UPPSALA UNIVERSITET

Constructing NN for regression

A **neural network (NN)** is a nonlinear function $y = f(\mathbf{x}; \boldsymbol{\theta}) + \epsilon$ from an input $\mathbf{x}$ to an output y parameterized by parameters $\boldsymbol{\theta}$.

Linear regression models the relationship between a continuous output y and a continuous input $\mathbf{x}$,

$$y = \beta_0 + \sum_{j=1}^{p} x_j \beta_j = \beta^{\mathsf{T}} \mathbf{x} + \epsilon,$$

where β is the parameters composed by the "weights" β_j and the offset ("bias"/"intercept") term β_0,

$$\beta = \begin{pmatrix} \beta_0 & \beta_1 & \beta_2 & \cdots & \beta_p \end{pmatrix}^{\mathsf{T}},$$

$$\mathbf{x} = \begin{pmatrix} 1 & x_1 & x_2 & \cdots & x_p \end{pmatrix}^{\mathsf{T}}.$$

UPPSALA UNIVERSITET

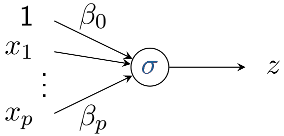
Generalized linear regression

We can generalize this by introducing nonlinear transformations of the predictor $\boldsymbol{\beta}^\mathsf{T}\mathbf{x}$,

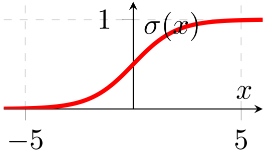
$$y = \sigma(\boldsymbol{\beta}^\mathsf{T}\mathbf{x}) + \epsilon,$$

or equivalently

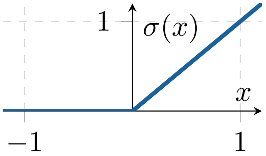
$$y = z + \epsilon, \qquad z = \sigma(\boldsymbol{\beta}^\mathsf{T}\mathbf{x})$$



We call $\sigma(x)$ the **activation function**. Two common choices are:



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

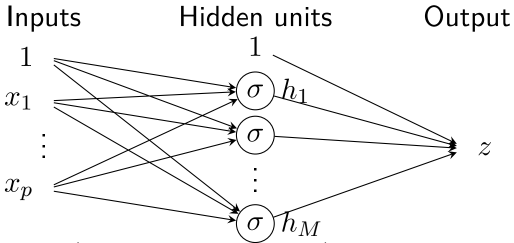


ReLU: $\sigma(x) = \max(0, x)$

Let us consider an example of a **neural network**.

Neural network - construction

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



$$h_1 = \sigma\left(\beta_{01}^{(1)} + \sum\nolimits_{j=1}^{p} \beta_{j1}^{(1)} x_j\right)$$

$$h_2 = \sigma\left(\beta_{02}^{(1)} + \sum\nolimits_{j=1}^{p} \beta_{j2}^{(1)} x_j\right)$$

$$\vdots$$

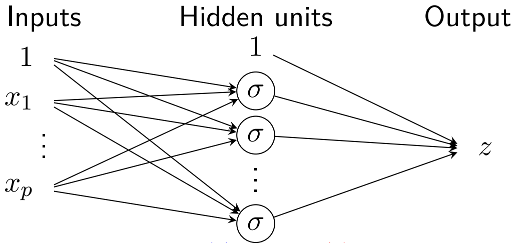
$$h_M = \sigma\left(\beta_{0M}^{(1)} + \sum\nolimits_{j=1}^{p} \beta_{jM}^{(1)} x_j\right)$$

$$z = \beta_0^{(2)} + \sum_{m=1}^{M} \beta_m^{(2)} h_m$$

UPPSALA UNIVERSITET

Neural network - construction

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



$$\mathbf{h} = \sigma(W^{(1)\mathsf{T}}\mathbf{x} + \mathbf{b}^{(1)\mathsf{T}})$$

$$z = \mathbf{W}^{(2)\mathsf{T}}\mathbf{h} + \mathbf{b}^{(2)\mathsf{T}}$$

$$\mathbf{h} = \begin{bmatrix} h_1 \\ \vdots \\ h_M \end{bmatrix}$$

Hidden units

$$\mathbf{b}^{(1)} = \begin{bmatrix} \beta_{01}^{(1)} & \dots & \beta_{0M}^{(1)} \end{bmatrix}$$

offset vector

$$\mathbf{b}^{(2)} = \begin{bmatrix} \beta_{0}^{(2)} \end{bmatrix}$$

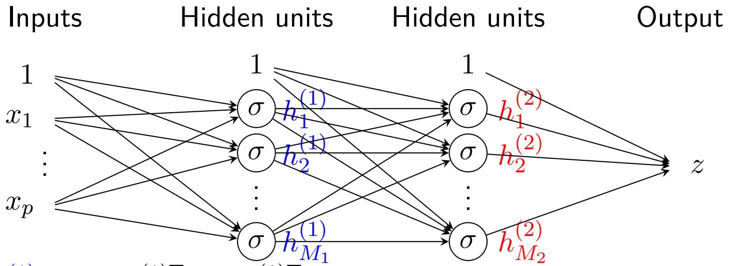
$$\mathbf{W}^{(1)} = \begin{bmatrix} \beta_{11}^{(1)} & \dots & \beta_{1M}^{(1)} \\ \vdots & \dots & \vdots \\ \beta_{p1}^{(1)} & \dots & \beta_{pM}^{(1)} \end{bmatrix}$$

weight matrix

$$\mathbf{W}^{(2)} = \begin{bmatrix} \beta_{1}^{(2)} \\ \vdots \\ \beta_{M}^{(2)} \end{bmatrix}$$

Neural network - construction

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



$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)\mathsf{T}}\mathbf{x} + \mathbf{b}^{(1)\mathsf{T}})$$

$$\mathbf{h}^{(2)} = \sigma(\mathbf{W}^{(2)\mathsf{T}}\mathbf{h}^{(1)} + \mathbf{b}^{(2)\mathsf{T}})$$

$$z = \mathbf{W}^{(3)\mathsf{T}}\mathbf{h}^{(2)} + \mathbf{b}^{(3)\mathsf{T}}$$

The model learns better using a deep network (several layers) instead of a wide and shallow network.

Deep learning

A neural network with L can be written as

$$\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x} \\
\mathbf{h}^{(l)} &= \sigma\left(\mathbf{W}^{(l)\mathsf{T}}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)\mathsf{T}}\right), \quad l = 1, \dots, L-1 \\
z &= \mathbf{W}^{(L)\mathsf{T}}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)\mathsf{T}}
\end{aligned}$$

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

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

which constitutes the parametric model $z = f(\mathbf{x}; \boldsymbol{\theta})$. If L is large we call it a deep neural network.

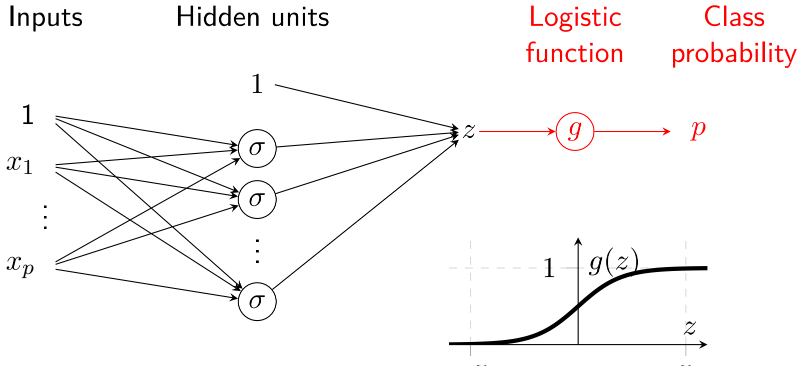
Deep learning is a class of machine learning models and algorithms that use a cascade of multiple layers, each of which is a nonlinear transformation.

UPPSALA UNIVERSITET

NN for classification ($K = 2$ classes)

We can also use neural networks for classification. With $K = 2$ classes we use the logistic function as we did in logistic regression to map $z \in \mathbb{R}$ to a **class probability** $p \in [0, 1]$

$$p = \Pr(y = 1|\mathbf{x}), \qquad p = g(z) = \frac{e^z}{1 + e^z}$$

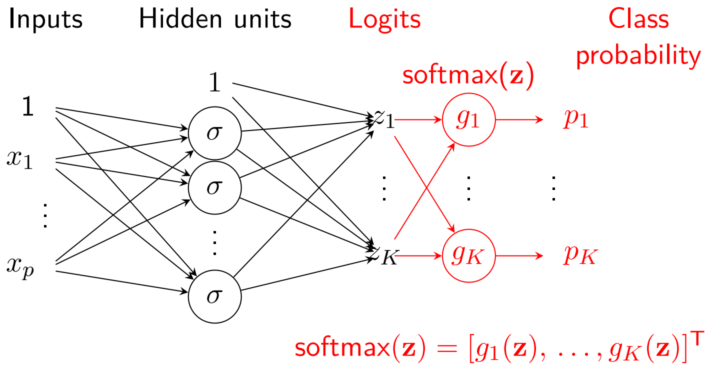


UPPSALA UNIVERSITET

NN for classification ($K > 2$ classes)

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

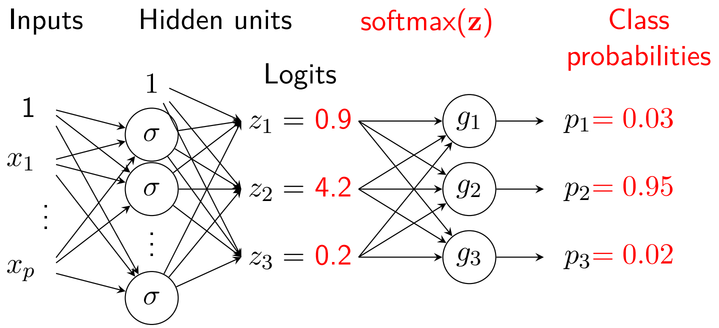
$$p_k = g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}, \qquad k = 1, \dots, K.$$



UPPSALA UNIVERSITET

Example with $K = 3$ classes

Consider an example with three classes $K = 3$.



where

$$g_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}, \qquad \text{softmax}(\mathbf{z}) = [g_1(\mathbf{z}), \dots, g_K(\mathbf{z})]^\mathsf{T}.$$

Softmax activation function

By construction...

1. ... each output $g_k(\mathbf{z}),\ k = 1, \dots, K$ is in the range $]0, 1[$ since
   - $g_k(\mathbf{z}) = \frac{e^{z_k}}{e^{z_1}+\dots+e^{z_K}} > 0, \quad$ since $\quad e^{z_k} > 0$
   - $g_k(\mathbf{z}) = \frac{e^{z_k}}{e^{z_1}+\dots+e^{z_K}} < 1, \quad$ since $\quad e^{z_k} < e^{z_1} + \dots + e^{z_K}$
2. ... all elements $g_k(\mathbf{z}),\ k = 1, \dots, K$ sum to 1 since

$$\sum_{k=1}^{K} g_k(\mathbf{z}) = \frac{\sum_{k=1}^{K} e^{z_k}}{\sum_{l=1}^{K} e^{z_l}} = 1$$

Hence, we can use the softmax activation function $\text{softmax}(\mathbf{z}) = [g_1(\mathbf{z}), \dots, g_K(\mathbf{z})]^\mathsf{T}$ to model probabilities.

One-hot encoding (for $K > 2$)

We use **one-hot encoding** for the output $\mathbf{y} = [y_1, \dots, y_K]^\mathsf{T}$, which means

$$y_k = \begin{cases} 1 & \text{if } \mathbf{x} \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases}$$

Example: We have $K = 3$ classes and if $\mathbf{x}$ belongs to ...

- ...class $k = 1$, then $\mathbf{y} = [1,\ 0,\ 0]^\mathsf{T}$.
- ...class $k = 2$, then $\mathbf{y} = [0,\ 1,\ 0]^\mathsf{T}$.
- ...class $k = 3$, then $\mathbf{y} = [0,\ 0,\ 1]^\mathsf{T}$.

One-hot encoding and cross-entropy loss

Example: Consider $K = 3$ classes and the output $\mathbf{y} = [0, \ 1, \ 0]^\mathsf{T}$.

	$k=1$	$k=2$	$k=3$	$L(\mathbf{y}, \mathbf{p})$
y_k	0	**1**	0	
$p_k(\boldsymbol{\theta}_A)$	0.28	**0.68**	0.04	$-\log 0.68 = 0.39$
$p_k(\boldsymbol{\theta}_B)$	0.51	**0.03**	0.46	$-\log 0.03 = 3.51$
$p_k(\boldsymbol{\theta}_C)$	0.03	**0.95**	0.02	$-\log 0.95 =$ **0.05**

How do we compare $y_1, \dots, y_K$ with $p_1, \dots, p_K$?

Cross-entropy loss function: $L(\mathbf{y}, \mathbf{p}) = -\sum_{k=1}^{K} y_k \log p_k.$

Consider n training data points $\mathcal{T} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{n}$. We then train the network by minimizing

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

with numerical optimization methods (next lecture).

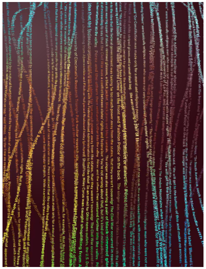
Example: Language models

On very recent result on the use of deep learning for language modeling

Language Models - February 2019

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.
Language Models are Unsupervised Multitask Learners.,

February, 2019.



See also blog post:
`https://blog.openai.com/better-language-models/`

UPPSALA UNIVERSITET

Language models - Background

A language model is a probabilistic model over sequences of words.

Training

$$\underbrace{\text{This is an introductory course to statistical machine}}_{\text{input}}\ \underbrace{\text{learning}}_{\text{output}}$$

The model can be used to hallucinate new texts.

Prediction

$$\underbrace{\text{Machine learning is the scientific study of algorithms and}}_{\text{input}}\ \underbrace{\text{statistical}}_{\text{prediction}}$$

In the paper they trained a huge language model with 1.5 billion (!) parameters using 40 GB of text data from internet.

Can also be used for Reading Comprehension, Language Translation, Question Answering etc...

Language models - Synthetic text (I/IV)

System Prompt (human-written) *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

Model Completion (machine-written)
The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

...

Language models - Synthetic text (II/IV)

...

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

...

Language models - Synthetic text (III/IV)

...

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them - they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

...

Language models - Synthetic text (IV/IV)

...

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

Language models - policy implication

Could be used for

- AI writing assistants
- More capable dialogue agents
- Unsupervised translation between languages
- Better speech recognition systems

Potential malicious purposes

- Generate misleading news articles
- Impersonate others online
- Automate the production of faked content on social media
- Automate the production of spam/phishing content

Due to our concerns about malicious applications of the technology, we are not releasing the trained model.

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)

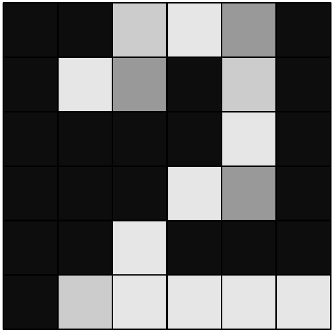
The description here will focus on images.

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



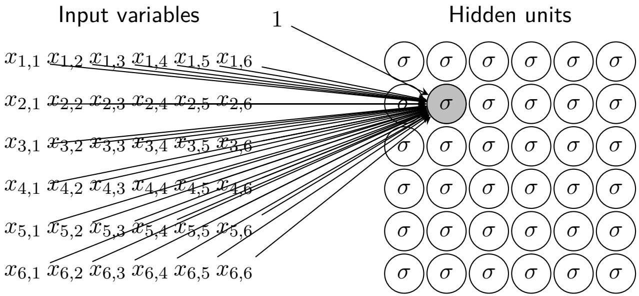
Data representation

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.

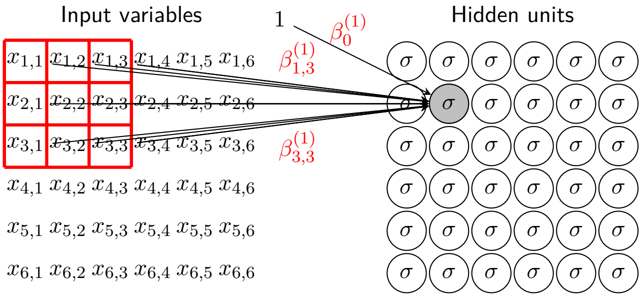
- **Dense layer**: Each hidden unit is connected with **all pixels**. Each pixel-hidden-unit-pair has its own **unique parameter**.


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

The convolutional layer

Consider a hidden layer with 6×6 hidden units.

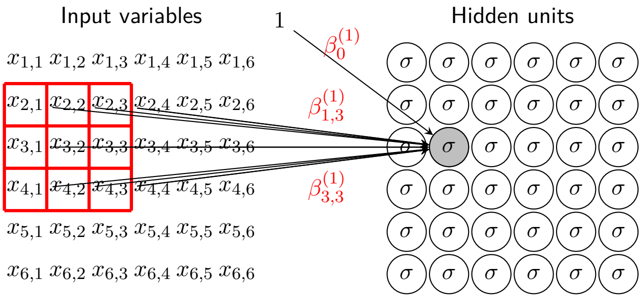
- **Dense layer**: Each hidden unit is connected with **all pixels**. Each pixel-hidden-unit-pair has its own **unique parameter**.
- **Convolutional layer**: Each hidden unit is connected with a **region of pixels** via a set of parameters, so-called **kernel**. Different hidden units have the **same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


Input variables
Hidden units
1
$\beta_0^{(1)}$
$\beta_{1,3}^{(1)}$
$\beta_{3,3}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ

The convolutional layer

Consider a hidden layer with 6×6 hidden units.

- **Dense layer**: 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 **kernel**. Different hidden units have the same **same set of parameters**.

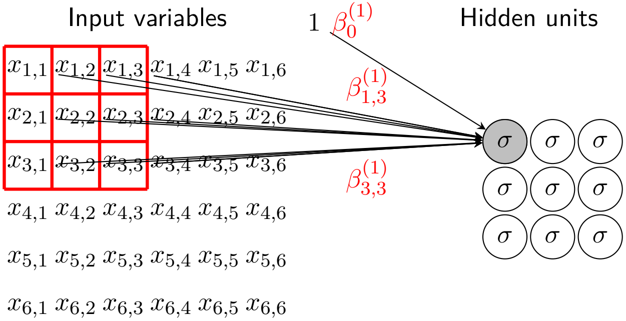


Input variables
Hidden units
1
$\beta_0^{(1)}$
$\beta_{1,3}^{(1)}$
$\beta_{3,3}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ

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

Condensing information with strides

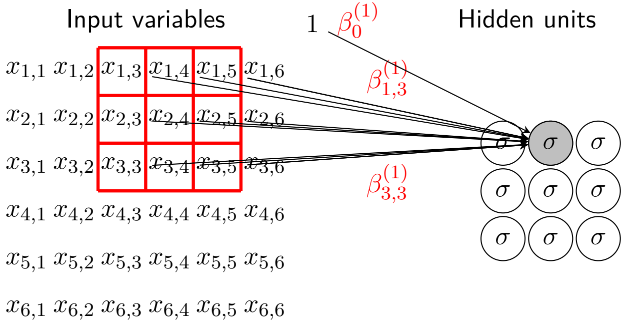
- **Problem**: As we proceed though the network we want to condense the information.
- **Solution**: Apply the kernel to every two pixels. We use a **stride** of 2 (instead of 1).


Input variables
Hidden units
1
$\beta_0^{(1)}$
$\beta_{1,3}^{(1)}$
$\beta_{3,3}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ σ σ
σ σ σ
σ σ σ

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 kernel to every two pixels. We use a **stride** of 2 (instead of 1).

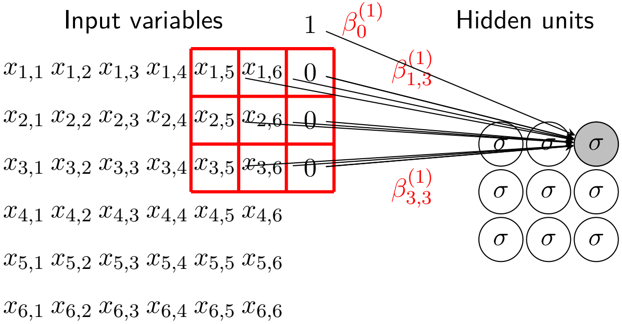


Input variables
Hidden units
1
$\beta_0^{(1)}$
$\beta_{1,3}^{(1)}$
$\beta_{3,3}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ σ σ
σ σ σ
σ σ σ

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 kernel to every two pixels. We use a **stride** of 2 (instead of 1).

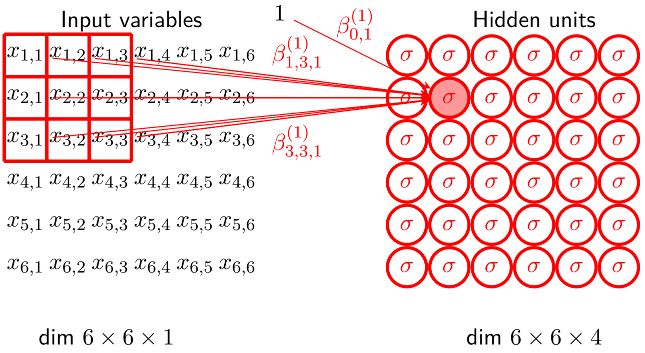


Input variables
Hidden units
1
$\beta_0^{(1)}$
$\beta_{1,3}^{(1)}$
$\beta_{3,3}^{(1)}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$ 0
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$ 0
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$ 0
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ

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

Multiple channels

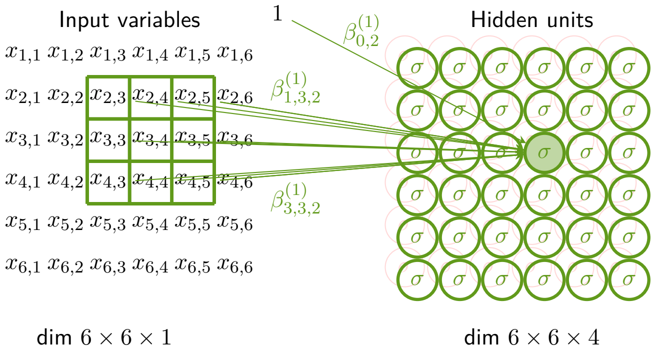
- One kernel per layer does not give enough flexibility. $\Rightarrow$
- We use **multiple kernels** (visualized with different colors).
- Each kernel produces its own set of hidden units – a **channel**.



Input variables
Hidden units
1
$\beta^{(1)}_{0,1}$
$\beta^{(1)}_{1,3,1}$
$\beta^{(1)}_{3,3,1}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

Multiple channels

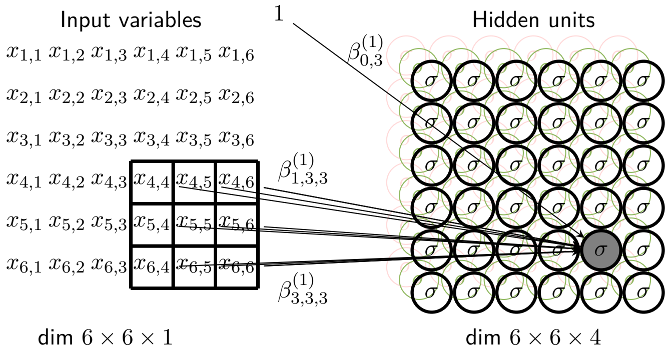
- One kernel per layer does not give enough flexibility. $\Rightarrow$
- We use **multiple kernels** (visualized with different colors).
- Each kernel produces its own set of hidden units – a **channel**.



Input variables
1
$\beta^{(1)}_{0,2}$
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$\beta^{(1)}_{1,3,2}$
$\beta^{(1)}_{3,3,2}$
σ
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

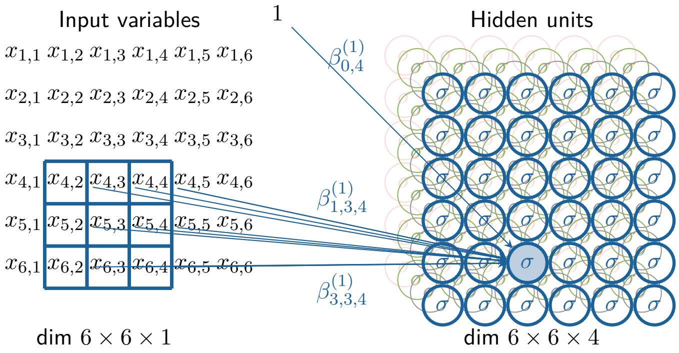
Multiple channels

- One kernel per layer does not give enough flexibility. $\Rightarrow$
- We use **multiple kernels** (visualized with different colors).
- Each kernel produces its own set of hidden units – a **channel**.


Input variables
Hidden units
1
$\beta^{(1)}_{0,3}$
$\beta^{(1)}_{1,3,3}$
$\beta^{(1)}_{3,3,3}$
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
σ
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

Multiple channels

- One kernel per layer does not give enough flexibility. $\Rightarrow$
- We use **multiple kernels** (visualized with different colors).
- Each kernel produces its own set of hidden units – a **channel**.

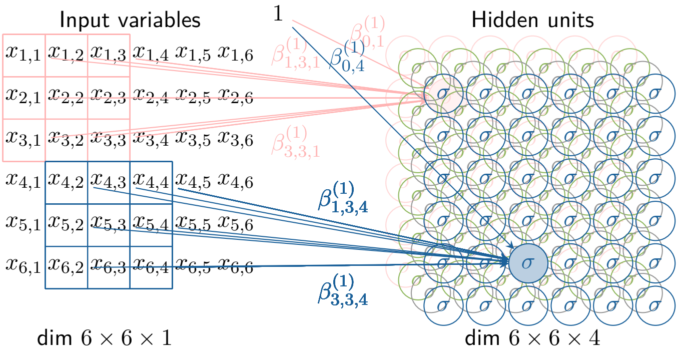


Input variables
1
Hidden units
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$\beta^{(1)}_{0,4}$
$\beta^{(1)}_{1,3,4}$
$\beta^{(1)}_{3,3,4}$
σ
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

UPPSALA UNIVERSITET

Multiple channels

- One kernel per layer does not give enough flexibility. $\Rightarrow$
- We use **multiple kernels** (visualized with different colors).
- Each kernel produces its own set of hidden units – a **channel**.



Input variables
Hidden units
1
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$ $x_{1,5}$ $x_{1,6}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$ $x_{2,5}$ $x_{2,6}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$ $x_{3,5}$ $x_{3,6}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$ $x_{4,5}$ $x_{4,6}$
$x_{5,1}$ $x_{5,2}$ $x_{5,3}$ $x_{5,4}$ $x_{5,5}$ $x_{5,6}$
$x_{6,1}$ $x_{6,2}$ $x_{6,3}$ $x_{6,4}$ $x_{6,5}$ $x_{6,6}$
$\beta^{(1)}_{0,1}$
$\beta^{(1)}_{1,3,1}$
$\beta^{(1)}_{0,4}$
$\beta^{(1)}_{3,3,1}$
$\beta^{(1)}_{1,3,4}$
$\beta^{(1)}_{3,3,4}$
σ
dim $6 \times 6 \times 1$
dim $6 \times 6 \times 4$

Hidden layers are organized in **tensors** of size (rows $\times$ columns $\times$ channels).

What is a tensor?

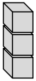
A **tensor** is a generalization of scalar, vector and matrix to arbitrary **order**.

Scalar
order **0**

$$a = 3$$

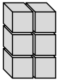
Vector
order **1**

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



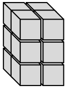
Matrix
order **2**

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



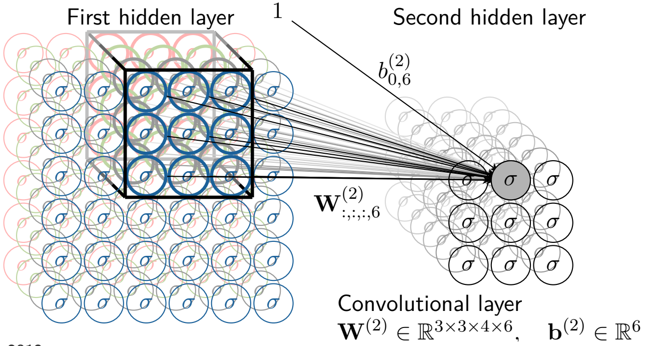
Tensor
any order
(here order **3**)

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



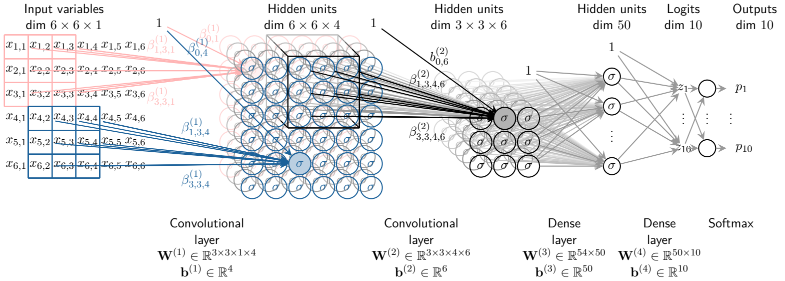
Multiple channels (cont.)

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


First hidden layer
1
Second hidden layer
$b^{(2)}_{0,6}$
$\mathbf{W}^{(2)}_{:,:,:,6}$
σ
Convolutional layer
$\mathbf{W}^{(2)} \in \mathbb{R}^{3\times3\times4\times6}$, $\mathbf{b}^{(2)} \in \mathbb{R}^{6}$

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 $6 \times 6 \times 1$
Hidden units dim $6 \times 6 \times 4$
Hidden units dim $3 \times 3 \times 6$
Hidden units dim 50
Logits dim 10
Outputs dim 10
$\beta^{(1)}_{0,1}$
$\beta^{(1)}_{1,3,1}$
$\beta^{(1)}_{0,4}$
$\beta^{(1)}_{3,3,1}$
$\beta^{(1)}_{1,3,4}$
$\beta^{(1)}_{3,3,4}$
$b^{(2)}_{0,6}$
$\beta^{(2)}_{1,3,4,6}$
$\beta^{(2)}_{3,3,4,6}$
z_1
z_{10}
p_1
p_{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 $K = 10$ classes.

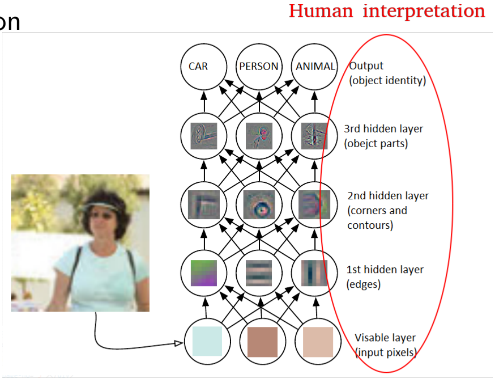
Why deep?

Example: Image classification

Input: pixels of an **image**

Output: **object identity**

Each hidden layer extracts increasingly abstract features.


Human interpretation
CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (obejct parts)
2nd hidden layer (corners and contours)
1st hidden layer (edges)
Visable layer (input pixels)

Zeiler, M. D. and Fergus, R. **Visualizing and understanding convolutional networks** *Computer Vision - ECCV* (2014).

Some comments - Why now?

Neural networks have been around for more than fifty years. Why have they become so popular now (again)?

To solve really interesting problems you need:

1. Efficient learning **algorithms**
2. Efficient computational **hardware**
3. A lot of labeled **data**!

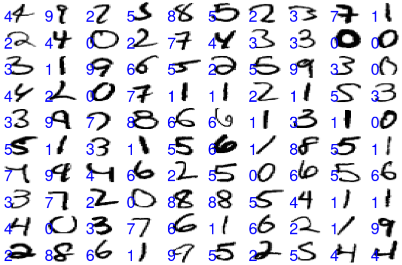
These three factors have not been fulfilled to a satisfactory level until the last 5-10 years.

The lab

Topic: Image classification with neural networks

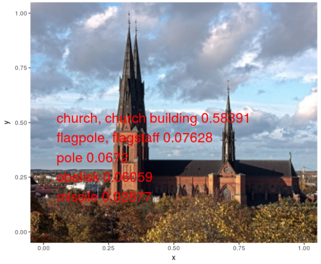
Task 1

Classification of hand-written digits



Task 2

Real world image classification



- Read Section 2 in the lab instruction and do the preparatory exercises in Section 3 **before** the lab

A few concepts to summarize lecture 8

Neural network (NN): A nonlinear parametric model constructed by stacking several linear models with intermediate nonlinear activation functions.

Activation function: (a.k.a squashing function) A nonlinear scalar function applied to each output element of the linear models in a NN.

Hidden units: Intermediate variables in the NN which are not observed, i.e. belongs neither to the input nor output data.

Softmax: A function used to transform the output of a NN to class probabilities.

One-hot encoding: An encoding of the output for training NNs with softmax output.

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

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

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

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

Tensor: A generalization of matrices to arbitrary order.