The slides have been created by Cyrill Stachniss.
5 Minute Preparation for Today

https://www.ipb.uni-bonn.de/5min/

5 Minutes with Cyrill
Neural Networks
Image Classification

input

classifier

output

"cat"

"5"
Semantic Segmentation

“a label for each pixel”
Neural Networks

- Machine learning technique
- Often used for classification, semantic segmentation, and related tasks
- First ideas discussed in the 1950/60ies
- Theory work on NNs in the 1990ies
- Increase in attention from 2000 on
- Deep learning took off around 2010
- CNNs for image tasks from 2012 on
Part 1
Neural Networks Basics
Neural Network

What is a neuron?  What is a network?

fundamental unit (of the brain)  connected elements

neural networks are connected elementary (computing) units
Biological Neurons

Biological neurons are the fundamental units of the brain that

- Receive sensory input from the external world or from other neurons
- Transform and relay signals
- Send signals to other neurons and also motor commands to the muscles
Artificial Neurons

Artificial neurons are the fundamental units of artificial neural networks that
- Receive **inputs**
- **Transform** information
- Create an **output**
Neurons

- Receive **inputs / activations** from sensors or other neurons
- **Combine / transform** information
- Create an **output / activation**
Neurons as Functions

We can see a neuron as a function

- Input given by $\mathbf{x} \in \mathbb{R}^N$
- Transformation of the input data can be described by a function $f$
- Output $f(\mathbf{x}) = \hat{y} \in \mathbb{R}$
Neural Network

- NN is a network/graph of neurons
- Nodes are neurons
- Edges represent input-output connections of the data flow
Neural Network as a Function

- The whole network is again a function
- Multi-layer perceptron or MLP is often seen as the “vanilla” neural network

\[
\begin{bmatrix}
    x_0 \\
    x_1 \\
    x_2 \\
    x_3 \\
\end{bmatrix} \in \mathbb{R}^4 \quad \hat{y} = f_{NN}(x) \quad \begin{bmatrix}
    y_0 \\
    y_1 \\
    y_2 \\
    y_3 \\
\end{bmatrix} \in \mathbb{R}^4
\]
Neural Networks are Functions

- Neural networks are functions
- Consist of connected artificial neurons
- Input layer takes (sensor) data
- Output layer provides the function result (information or command)
- Hidden layers do some computations

\[ \mathbf{x} \in \mathbb{R}^N \quad \xrightarrow{f_{NN}(\mathbf{x}) = \hat{\mathbf{y}}} \quad \hat{\mathbf{y}} \in \mathbb{R}^M \]

input layer | hidden layers | output layer
Different Types of NNs

- Perceptron
- MLP – Multilayer perceptron
- Autoencoder
- CNN – Convolutional NN
- RNN – Recurrent NN
- LSTM – Long/short term memory NN
- GANs – Generative adversarial network
- Graph NN
- Transformer
- ...
A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

- Deep Feed Forward (DFF)
- Perceptron (P)
- Feed Forward (FF)
- Radial Basis Network (RBF)
- Recurrent Neural Network (RNN)
- Long / Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Auto Encoder (AE)
- Variational AE (VAE)
- Denoising AE (DAE)
- Sparse AE (SAE)

- Markov Chain (MC)
- Hopfield Network (HN)
- Boltzmann Machine (BM)
- Restricted BM (RBM)
- Deep Belief Network (DBN)
Multi-layer Perceptron (MLP)
Multi-layer Perceptron
Seen as a Function
Image Classification Example

$\mathcal{X}$

input image

$\hat{y}$

label

$f_{NN}(\cdot)$

function that maps images to labels

\begin{align*}
x_0 \\
x_1 \\
\vdots \\
x_N \\
y_0 \\
y_1 \\
\vdots \\
y_M \\
\end{align*}

"cat"
What is the Network’s Input?

An image consists of individual pixels.
What is the Network’s Input?

An image consists of individual pixels.

Each pixel stores an intensity value.
What is the Network’s Input?

pixel intensities

An image consists of individual pixels.

Each pixel stores an intensity value.
What is the Network’s Input?

An image consists of individual pixels.
Each pixel stores an intensity value.
We have $N+1$ such intensity values.
What is the Network’s Input?

Arrange all the intensity values in a N+1 dim vector.
What is the Network’s Input?

Arrange all the intensity values in a N+1 dim vector.
What is the Network’s Input?

Arrange all the intensity values in a $N+1$ dim vector.
Input Layer of the Network

This vector is the input layer of our network!
What is the Network’s Output?

What is the Network’s Output?

\[ \hat{y} \quad \text{“cat”} \]
What is the Network’s Output?

Is it a...
cat or a
dog or a
human or a

...?
What is the Network’s Output?

Is it a...
- cat or a
dog or a
human or a
...

\[
\begin{bmatrix}
1 \\
0 \\
0 \\
\vdots
\end{bmatrix}
\]

indicator vector
What is the Network’s Output?

Is it a...  
\begin{bmatrix}
1 \\
0 \\
0 \\
\vdots \\
\end{bmatrix}

indica\_vector
What is the Network’s Output?

Is it a...

- cat or a
dog or a
human or a

...?

\[
\begin{bmatrix}
98% \\
1% \\
0.1% \\
\vdots
\end{bmatrix}
\]

we are
never
certain...
Output of the Network

The output layer is a vector indicating an activation/likelihood for each label.
Image Classification

Pixels intensities are the values of the input layer. The output layer is a vector of likelihoods for the possible labels. The largest value determines the class as “cat.”
Multi-layer Perceptron
Let’s Look at a Single Neuron
Multi-layer Perceptron
Let’s Look at a Single Neuron
Perceptron (Single Neuron)

How does this function look like?

\[ a = f(a_0, a_1, a_2, \ldots, a_n) \]
Perceptron (Single Neuron)

$output = f(a_0, a_1, a_2, \ldots, a_n)$

activations from previous layer

weights

output activation for the next layer
Function Behind a Neuron

(input) activations $a_i$

weights $w_i$

bias $b$

activation function $\sigma(\cdot)$

output activation $\alpha$
Function Behind a Neuron

A neuron gets activated ($a$) through

- A weighted sum of input activations $w_i, a_i$
- A bias activation $b$
- An activation function $\sigma(\cdot)$

$$ a = \sigma(w_0a_0 + w_1a_1 + \ldots + w_na_n + b) $$
Similarity to Convolutions?

- A neuron is similar to a convolution
- Remember linear shift-invariant kernels used as local operators

\[ a = \sigma(w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b) \]

This part looks like the convolutions used for defining local operators

Additionally: activation function and bias
Activation Function

- Biological neurons are either active or not active
- We can see this as a step function:

```
\sigma(x)
```

- Bias tells us where the activation happens
Activation Function

We can model this behavior through

\[ a = \begin{cases} 
0 & \sum_i w_i a_i \leq -b \\
1 & \text{otherwise}
\end{cases} \]

Non-smooth functions (e.g., steps) have disadvantages later down the line...
Sigmoid Activation Function

- Common activation function is a sigmoid (also called logistic function)
- Smooth function
- Squeezes values to \([0, 1]\)

\[
\sigma(x) = \frac{1}{1 + \exp(-x)}
\]
ReLU Activation Function

- Most commonly used one is the so-called “rectified linear unit” or ReLU
  \[ \sigma(x) = \text{ReLU}(x) = \max(0, x) \]
- Often advantages for deep networks
Neuron Activation

- A neuron is only activated if $x > 0$

- If $a = \text{ReLU}(w_0a_0 + w_1a_1 + \ldots + w_na_n + b) > 0$

- the weighted activations are larger than the negative bias $-b$
Common Activation Functions

There are different activation functions

- sigmoid()
- ReLU()
- tanh()
- atan()
- softplus()
- identity()
- step-function()
- ...

ReLU is often used
# Illustration

<table>
<thead>
<tr>
<th>Name</th>
<th>Plot</th>
<th>Equation</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity</td>
<td><img src="image" alt="Identity Plot" /></td>
<td>$f(x) = x$</td>
<td>$f'(x) = 1$</td>
</tr>
<tr>
<td>Binary step</td>
<td><img src="image" alt="Binary Step Plot" /></td>
<td>$f(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$f'(x) = \begin{cases} 0 &amp; \text{for } x \neq 0 \ ? &amp; \text{for } x = 0 \end{cases}$</td>
</tr>
<tr>
<td>Logistic (a.k.a Soft step)</td>
<td><img src="image" alt="Logistic Plot" /></td>
<td>$f(x) = \frac{1}{1 + e^{-x}}$</td>
<td>$f'(x) = f(x)(1 - f(x))$</td>
</tr>
<tr>
<td>TanH</td>
<td><img src="image" alt="TanH Plot" /></td>
<td>$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$</td>
<td>$f'(x) = 1 - f(x)^2$</td>
</tr>
<tr>
<td>ArcTan</td>
<td><img src="image" alt="ArcTan Plot" /></td>
<td>$f(x) = \tan^{-1}(x)$</td>
<td>$f'(x) = \frac{1}{x^2 + 1}$</td>
</tr>
<tr>
<td>Rectified Linear Unit (ReLU)</td>
<td><img src="image" alt="Rectified Linear Unit Plot" /></td>
<td>$f(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$f'(x) = \begin{cases} 0 &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases}$</td>
</tr>
<tr>
<td>Parameteric Rectified Linear Unit (PRelu)</td>
<td><img src="image" alt="Parameteric Rectified Linear Unit Plot" /></td>
<td>$f(x) = \begin{cases} \alpha x &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$f'(x) = \begin{cases} \alpha &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases}$</td>
</tr>
<tr>
<td>Exponential Linear Unit (ELU)</td>
<td><img src="image" alt="Exponential Linear Unit Plot" /></td>
<td>$f(x) = \begin{cases} \alpha(e^x - 1) &amp; \text{for } x &lt; 0 \ x &amp; \text{for } x \geq 0 \end{cases}$</td>
<td>$f'(x) = \begin{cases} f(x) + \alpha &amp; \text{for } x &lt; 0 \ 1 &amp; \text{for } x \geq 0 \end{cases}$</td>
</tr>
<tr>
<td>SoftPlus</td>
<td><img src="image" alt="SoftPlus Plot" /></td>
<td>$f(x) = \log_e(1 + e^x)$</td>
<td>$f'(x) = \frac{1}{1 + e^{-x}}$</td>
</tr>
</tbody>
</table>
Function Behind a Neuron

- Neuron gets activated if the weighted sum of input activations is large enough (larger than the negative bias)

\[ a = \sigma (w_0 a_0 + w_1 a_1 + \ldots + w_n a_n + b) \]

- This is the case for all neurons in the neural network
For All Neurons...

These are a lot of values!
Let's Use a Matrix Notation

\[
\begin{bmatrix}
    a_0^{(1)} \\
    a_1^{(1)} \\
    \vdots \\
    a_n^{(1)}
\end{bmatrix} = \sigma \left( \begin{bmatrix}
    w_{00} & w_{01} & \cdots & w_{0n} \\
    w_{10} & w_{11} & \cdots & w_{1n} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{k0} & w_{k1} & \cdots & w_{kn}
\end{bmatrix} \begin{bmatrix}
    a_0^{(0)} \\
    a_1^{(0)} \\
    \vdots \\
    a_n^{(0)}
\end{bmatrix} + \begin{bmatrix}
    b_0^{(1)} \\
    b_1^{(1)} \\
    \vdots \\
    b_n^{(1)}
\end{bmatrix} \right)
\]
Each Layer Can Be Expressed Through Matrix Multiplications

\[
\begin{align*}
\text{layer 1} & \quad \begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_n^{(1)} \end{bmatrix} = \sigma \left( \begin{bmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \cdots & \vdots \\ w_{k0} & w_{k1} & \cdots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \right) \\
\text{layer 0} & \\
\end{align*}
\]

\[
\begin{align*}
\mathbf{a}^{(1)} &= \sigma \left( \mathbf{W} \mathbf{a}^{(0)} + \mathbf{b}^{(1)} \right)
\end{align*}
\]
Do It Layer by Layer...

\[ \sigma \left( W^{(1)} a^{(0)} + b^{(1)} \right) = a^{(1)} \]

\[ \cdots \sigma \left( W^{(k)} a^{(k-1)} + b^{(k)} \right) = a^{(k)} \]
Do It Layer by Layer...

\[ \textbf{input} = \text{layer 0} \quad x = a^{(0)} \]

layer 1

layer 2

\[ \vdots \]

layer \( k = \textbf{output} \quad \sigma \left( W^{(k)} a^{(k-1)} + b^{(k)} \right) = a^{(k)} = \hat{y} \]

That not much more than linear algebra...
Feedforward Networks

- MLPs are feedforward networks
- The information flows form left to right
- There are no loops

- Such networks are called feedforward networks
- There exist other variants (e.g., RNNs)
Example:
Handwritten Digit Recognition
Handwritten Digit Recognition

28x28 pixel image

504192

= 5

[Image courtesy: Nielsen]
Handwritten Digit Recognition

[Image courtesy: Nielsen/Lecun]
A Basic MLP Recognizing Digits

[Image courtesy: Nielsen]
Images to Digits - A Mapping from 784 to 10 Dimensions

28x28 pixel input images
(784 dim)

[Partial image courtesy: Nielsen]
What Happens in the Layers?
What Happens in the 1\textsuperscript{st} Layer?

784 input activations = pixel intensities
784 weights = weights for pixel intensities
What Happens in the 1\textsuperscript{st} Layer?

784 input activations = pixel intensities
784 weights = weights for pixel intensities

\textbf{treat activations and weights as images}

pixel values $a_i$ weights $w_i$ effect on the weighted sum

\textbf{white black} (rest doesn’t matter)
What Happens in the 1\textsuperscript{st} Layer?

\[ a_i \]

\[ w_i \]

pixel values

weights tell us what matters for activating the neuron!

individual “weight images” for a neuron support individual patterns in the image
Link to Local Operators Defines Through Convolutions

- Direct link to defining image operators through convolutions

Here:
- Global (not local) operators
- Weight matrix does not (yet) “slide over image”
Weights & Bias = Patterns

- **Weights define** the patterns to look for in the image
- **Bias** tells us how well the image must match the pattern
- Activation functions “switches the neuron on” if it matches the pattern
What Happens in the 2\textsuperscript{nd} Layer?

- The weights in layer 2 tell us which 1\textsuperscript{st} layer patterns should be combined
- The deeper we go, the more patterns get arranged and combined
- The last layer decides, which final patterns make up a digit
What Happens in the Layers?

Input Layer 784

Hidden Layer 1 128 (relu)

Hidden Layer 2 64 (relu)

Output Layer 10 (softmax)

raw pixels → simple patterns → combined patterns → patterns to digits

[Image courtesy: Nielsen]
No Manual Features

Compared to several other classifiers, this network also includes the feature computations – it operates directly on the input data, no manual features!

raw simple combined patterns to digits

[Image courtesy: Nielsen]
Classification Performance

Such a simple MLP achieves a correct classification for \( \sim 96\% \) of the examples.

[Partial image courtesy: Nielsen]
Classification Performance

- A simple MLP achieves a classification accuracy of ~96%
- Note that there are tricky cases

That is a good performance for a simple model!
- Improved networks achieve ~99%
How to Design a Neural Network?
How to Make the Network Compute What We Want?

- So far, the network is a recipe for sequentially performing computations
- **Structure** and **parameters** are the design choices
- How to set them?

Learning!
Summary – Part 1

- What are neurons and neural networks
- Lots of different networks exists
- Focus: multi-layer perceptrons (MLP)
- Activations, weights, bias
- Networks have many parameters
- “It’s just a bunch of matrices and vectors”
- MLP for simple image classification
- Part 2: Learning the parameters
Literature & Resources

- Online Book by Michael Nielsen, Chapter 1: http://neuralnetworksanddeeplearning.com/chap1.html
- Nielsen, Chapter 1, Python3 code: https://github.com/MichalDanielDobrzanski/DeepLearningPython
- MNIST database:
  - http://yann.lecun.com/exdb/mnist/
- Grant Sanderson, Neural Networks
  https://www.3blue1brown.com/
- Alpaydin, Introduction to Machine Learning
Slide Information

- The slides have been created by Cyrill Stachniss as part of the photogrammetry and robotics courses.
- **I tried to acknowledge all people from whom I used images or videos. In case I made a mistake or missed someone, please let me know.**
- Huge thank you to Grant Sanderson (3blue1brown) for his great educational videos that influenced this lecture.
- Thanks to Michael Nielsen for his free online book & code
- If you are a university lecturer, feel free to use the course material. If you adapt the course material, please make sure that you keep the acknowledgements to others and please acknowledge me as well. To satisfy my own curiosity, please send me email notice if you use my slides.

Cyrill Stachniss, cyrill.stachniss@igg.uni-bonn.de