Neural Networks

Machine Learning 1 UE (INP.33761UF)

Join here: fbr.io/ml1p7

Thomas Wedenig

May 08, 2024

Institute of Theoretical Computer Science
Graz University of Technology, Austria

Hello everyone and welcome to Machine Learning 1! Today, we're diving deep into the fascinating world of Neural Networks. In this session, we'll explore the fundamental concepts behind neural networks, their architecture, and how they enable machines to learn and make decisions akin to human brains.

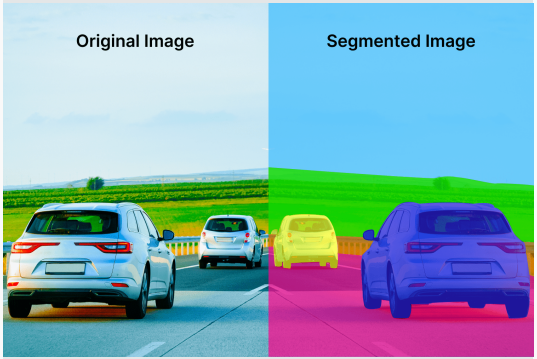
Hello everyone and welcome to Machine Learning 1! Today, we're diving deep into the fascinating world of Neural Networks. In this session, we'll explore the fundamental concepts behind neural networks, their architecture, and how they enable machines to learn and make decisions akin to human brains.

Fun fact: This text was written by a Large Language Model and was read by a voice cloning model that used a two-minute recording of my voice.

Table of contents

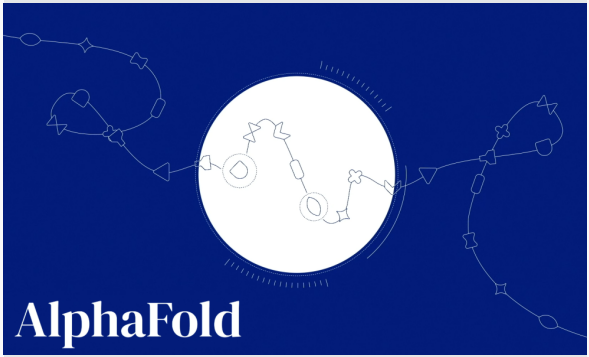
Motivation

Computer Vision



https://deeplobe.ai/
image-segmentation-the-most-interesting-applications/

AI 4 Science



https://techcrunch.com/2021/07/22/
deepmind-puts-the-entire-human-proteome-online-as-folded-by-alphafold

DALL·E 3

OpenAI DALL·E 3, "The inside of a neural network."

thispersondoesnotexist.com

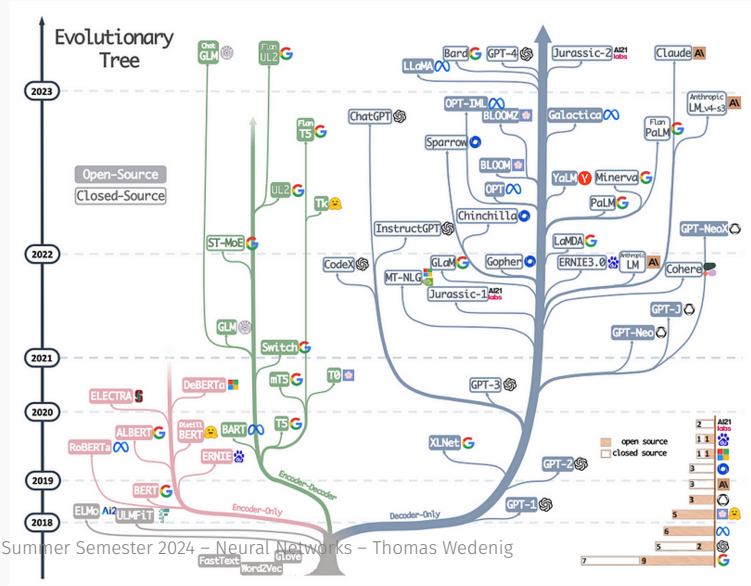
https://thispersondoesnotexist.com

Generative AI - Videos

e.g., https://openai.com/index/sora

Generative AI - Large Language Models

Source: https://abiaryan.com/posts/intro-llms/



Neural Network Size

- Neural Networks are **powerful, flexible, non-linear function approximators** !
- Most of these impressive applications **utilize large (and deep) neural networks**
- In this course, we will work with small and simple networks

Neural Network Size

- Neural Networks are **powerful, flexible, non-linear function approximators** !
- Most of these impressive applications **utilize large (and deep) neural networks**
- In this course, we will work with small and simple networks

NN architectures

- **Feed-forward NNs**
- Recurrent NNs
- Convolutional NNs
- Transformers

Covered in **Deep Learning VO/KU**
(Master's level)

Neural Network Size

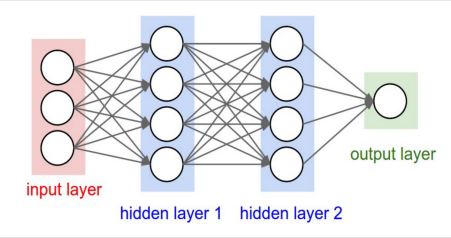
- Neural Networks are **powerful, flexible, non-linear function approximators** !
- Most of these impressive applications **utilize large (and deep) neural networks**
- In this course, we will work with small and simple networks

NN architectures

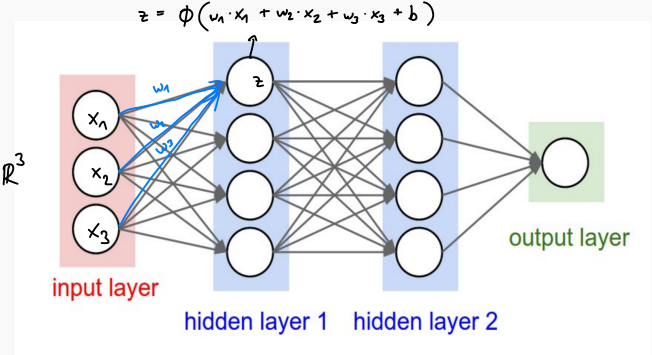
- **Feed-forward NNs**
- Recurrent NNs
- Convolutional NNs
- Transformers

Covered in **Deep Learning VO/KU**
(Master's level)

Feed-forward Neural Network



Feed-Forward Neural Networks



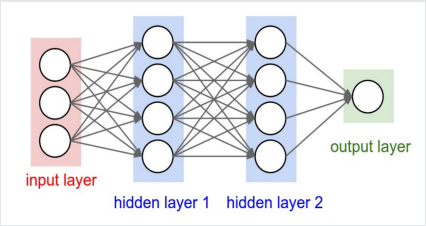
- Let $f_{\theta}(\mathbf{x})$ denote the network output
- Popular **activation function** for hidden units $\phi_h(z) = \text{ReLU}(z) = max(0, z)$
- What about the **activation at the output** ϕ_o ❓

Supervised Learning: Dataset $\mathcal{D} = \left\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(M)}, y^{(M)})\right\}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^N$

Supervised Learning: Dataset $\mathcal{D} = \left\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(M)}, y^{(M)})\right\}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^N$

Regression

$$y \in \mathbb{R} \implies f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}$$

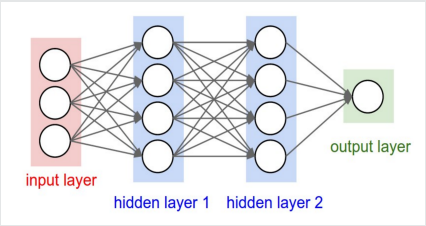


- **No activation** at output
  - "linear" activation: $\phi_o(z) = z$

Supervised Learning: Dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(M)}, y^{(M)})\}$ with $\mathbf{x}^{(i)} \in \mathbb{R}^N$

Regression

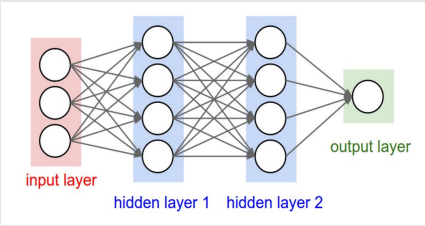
$$y \in \mathbb{R} \implies f_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}$$



- **No activation** at output
  - "linear" activation: $\phi_o(z) = z$

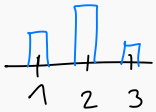
Binary Classification

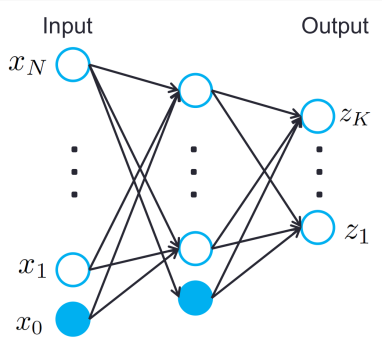
$$y \in \{-1, 1\} \implies f_{\boldsymbol{\theta}}(\mathbf{x}) \in [0, 1]$$



- **Sigmoid activation** at output
  - $\phi_o(z) = \sigma(z) = 1/(1 + \exp(-z))$

- What if $y \in \{1, \dots, K\}$ ❓
  - K-class **classification problem** 🤔

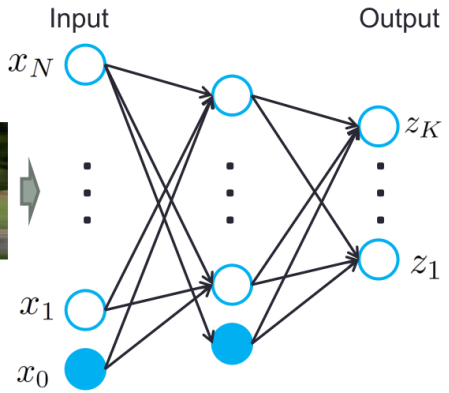




- NN should output a **probability distribution** over K classes
- Let $\mathbf{z} = (z_1, \dots, z_K)^T$ be the output **before applying the output activation**
  - Called "pre-activation" or "logits"
- We use $\phi_o(\mathbf{z}) = \text{softmax}(\mathbf{z})$ with

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

$-\log(0.9)$

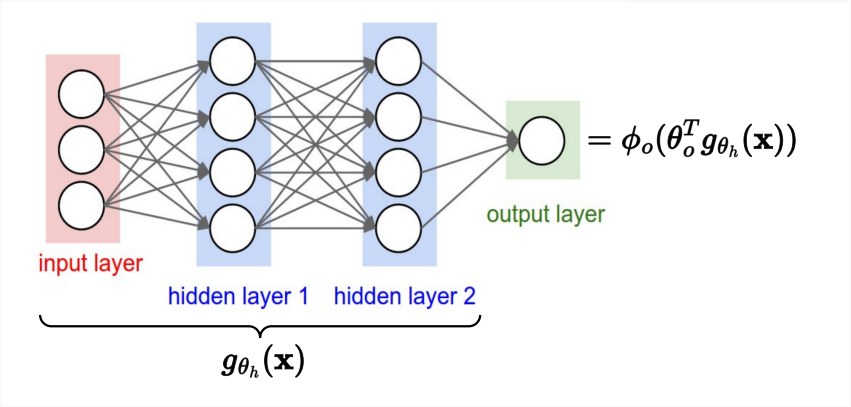


	K categories
0.9	“car”
0.03	“motorbike”
0.07	“cow”

Connection to previous methods

Previously, our prediction function $f_{\boldsymbol{\theta}}(\mathbf{x})$ looked like this:

- **Linear Regression:** $f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$
  - A linear function in $\mathbf{x}$ and $\boldsymbol{\theta}$
- **Linear Regression with non-linear feature transform:** $f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T g(\mathbf{x})$
  - A linear function in $\boldsymbol{\theta}$, but not in $\mathbf{x}$
- **Logistic Regression:** $f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^T g(\mathbf{x})\right)$
  - As above, pushed through a sigmoid non-linearity
- **Note:** Feature transform g has no parameters that are learned



- Non-linear feature transform g **now has learnable parameters θ_h** **!**

Loss Functions and Training

- We use the same loss functions as always:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \ell(y^{(i)}, \underbrace{f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})}_{\hat{y}^{(i)}})$$

  where $\ell(y, \hat{y})$ is a per-example loss

- **Regression** uses **Mean Squared Error**:

$$\ell(y, \hat{y}) = (y - \hat{y})^2$$

- **Binary Classification** uses **Binary Cross-Entropy**:

$$\ell(y, \hat{y}) = \mathbb{1}[y = 1](-\log(\hat{y})) + \mathbb{1}[y = -1](-\log(1 - \hat{y}))$$

- **Multi-class Classification** uses **Multi-class Cross-Entropy**:

$$\ell(y, \hat{\mathbf{y}}) = -\log(\hat{\mathbf{y}}_y), \qquad \hat{\mathbf{y}} \in \mathbb{R}^K, y \in \{1, \ldots, K\}$$

- We want

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

- $\mathcal{L}(\boldsymbol{\theta})$ is **highly non-convex** in $\boldsymbol{\theta}$ 😟
- But it's **differentiable** almost everywhere 😎
- Use **Gradient Descent** (or some variant) to find a local optimum of $\mathcal{L}(\boldsymbol{\theta})$
- Gradients $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ can be computed via **backpropagation**
- However...

- Recall that

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \ell\left(y^{(i)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right) \qquad \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}}\ell\left(y^{(i)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)$$

- What could be the **problem from a computational perspective** ❓

- Recall that

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \ell\left(y^{(i)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right) \qquad \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}}\ell\left(y^{(i)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)$$

- What could be the **problem from a computational perspective** ❓
- In many cases, $M = |\mathcal{D}|$ is **very large**
- We have to loop over our **entire training set** to get **a single gradient** $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$
  - Which amounts to a **single step** in gradient descent 😩

Solution: Mini-Batch Gradient Descent

For each step, use a **small subset** of our training data $\mathcal{B} \subset \mathcal{D}$ to estimate the gradient

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}} \ell \left(y^{(i)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right) \approx \frac{1}{|\mathcal{B}|} \sum_{j=1}^{|\mathcal{B}|} \nabla_{\boldsymbol{\theta}} \ell \left(y^{(j)}, f_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}) \right)$$

where $(\mathbf{x}^{(j)}, y^{(j)}) \in \mathcal{B}$.

- $\mathcal{B}$ is called a **mini-batch**
- This version of Gradient Descent is often called **Mini-Batch Gradient Descent**

Neural Network Regression Demo