Outline

Outline

Deep Learning: A Brief History

thws Technical University of Applied Sciences Würzburg-Schweinfurt

The Journey from Perceptrons to Modern AI

- **1958**: Rosenblatt's Perceptron - single layer, limited capacity
- **1969**: Minsky & Papert show perceptron limitations (e.g., XOR problem)
- **1986**: Backpropagation algorithm popularized (Rumelhart et al.)
- **1990s-2000s**: "AI Winter" - neural networks fall out of favor
- **2012**: AlexNet wins ImageNet - deep learning renaissance begins
- **2017-present**: Transformers revolutionize NLP and beyond

Why now?

- Massive datasets (ImageNet, Common Crawl, etc.)
- Computational power (GPUs, TPUs)
- Algorithmic innovations (ReLU, batch normalization, residual connections)

What Can Deep Learning Do?

Technical University
of Applied Sciences
Würzburg-Schweinfurt

Computer Vision

- Image classification, object detection, semantic segmentation
- Face recognition, medical image analysis

Natural Language Processing

- Machine translation, text generation, question answering
- Large language models (GPT, Claude, Gemini)

Other Domains

- Speech recognition and synthesis
- Game playing (AlphaGo, AlphaZero)
- Protein structure prediction (AlphaFold)
- Autonomous driving, robotics

Common thread: Learning hierarchical representations from data

Outline

1. What is Deep Learning?

2. **From Linear Models to Neural Networks**

3. Neural Network Architecture

4. Loss Functions and Learning

5. Course Overview

Linear Models: A Quick Recap

Linear Regression

- Model: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
- Input: $\mathbf{x} \in \mathbb{R}^d$, weights: $\mathbf{w} \in \mathbb{R}^d$, bias: $b \in \mathbb{R}$
- Learns a hyperplane in input space

Linear Classification (e.g., Logistic Regression)

- Model: $f(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$
- Decision boundary is still linear

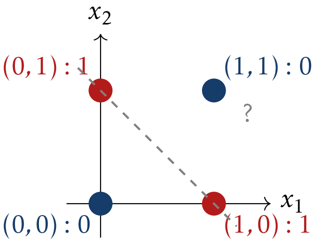
Fundamental Limitation

- Can only learn linear decision boundaries
- Many real-world problems are inherently non-linear
- Example: XOR problem cannot be solved by linear classifier

The XOR Problem

Problem Definition

- XOR (exclusive OR): output is 1 if inputs differ, 0 otherwise



Why Linear Models Fail

- No single line can separate blue points from red points
- We need non-linear decision boundaries
- Solution: multiple layers with non-linear activations

Outline

Technical University
of Applied Sciences
Würzburg-Schweinfurt

The Artificial Neuron

Technical University of Applied Sciences Würzburg-Schweinfurt

Biological Inspiration

- Loosely inspired by biological neurons
- Multiple inputs, single output
- Non-linear activation function

Mathematical Model

- Input: $\mathbf{x} = (x_1, x_2, \ldots, x_d)$
- Weights: $\mathbf{w} = (w_1, w_2, \ldots, w_d)$, bias: b
- Pre-activation: $z = \sum_{i=1}^{d} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$
- Output: $a = \phi(z)$ where ϕ is the activation function

Key insight: Without ϕ, composition of neurons is still just linear!

Activation Functions

Common Choices

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

- Output range: $(0, 1)$
- Historically popular, now less common in hidden layers
- Problem: vanishing gradients for $|z|$ large

Tanh: $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

- Output range: $(-1, 1)$
- Zero-centered (better than sigmoid)
- Still suffers from vanishing gradients

ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$

- Most popular for hidden layers
- Advantages: fast to compute, no vanishing gradient for $z > 0$
- Disadvantage: “dead neurons” when $z \leq 0$ always

Multi-Layer Perceptron (MLP)

Layer-wise Architecture

- **Input layer:** raw features $\mathbf{x} \in \mathbb{R}^{d_0}$
- **Hidden layers:** L layers with dimensions $d_1, d_2, \ldots, d_L$
- **Output layer:** predictions $\hat{\mathbf{y}} \in \mathbb{R}^{d_{L+1}}$

Forward Pass for Layer ℓ

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}$$
$$\mathbf{a}^{(\ell)} = \phi(\mathbf{z}^{(\ell)})$$

where:

- $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix
- $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias vector
- ϕ is applied element-wise
- Convention: $\mathbf{a}^{(0)} = \mathbf{x}$ (input)

Network Depth and Width

Network Capacity

- **Width:** number of neurons per layer
- **Depth:** number of hidden layers
- Both affect the model's representational power

Universal Approximation Theorem (informal)

- A neural network with one hidden layer of sufficient width can approximate any continuous function on a compact domain
- **But:** may require exponentially many neurons!
- Depth is more parameter-efficient than width

Why Deep Networks?

- Learn hierarchical representations (edges $\rightarrow$ shapes $\rightarrow$ objects)
- More parameter-efficient for complex functions
- Empirically perform better on real tasks

Outline

1. What is Deep Learning?
2. From Linear Models to Neural Networks
3. Neural Network Architecture
4. **Loss Functions and Learning**
5. Course Overview

The Learning Problem

Supervised Learning Setup

- Training data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$
- Goal: find parameters $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$ that minimize prediction error

Loss Function

- Measures how well the network fits the data
- For a single example: $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
- Total loss (empirical risk):

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i)$$

Optimization Goal

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$$

Common Loss Functions

Technical University of Applied Sciences Würzburg-Schweinfurt

Regression: Mean Squared Error (MSE)

$$\mathcal{L}_{\text{MSE}}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \frac{1}{2}\sum_{j=1}^{d}(\hat{y}_j - y_j)^2$$

Probabilistic interpretation: assumes $y \sim \mathcal{N}(\hat{y}, \sigma^2)$, equivalent to maximum likelihood

Binary Classification: Binary Cross-Entropy

$$\mathcal{L}_{\text{BCE}}(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

where $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) \in (0, 1)$ is predicted probability
Probabilistic interpretation: negative log-likelihood for Bernoulli distribution

Multi-Class Classification: Cross-Entropy

thws
Technical University
of Applied Sciences
Würzburg-Schweinfurt

Softmax Activation

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}} \quad \text{for } j = 1, \ldots, C$$

- Converts logits $\mathbf{z}$ to probability distribution
- $\sum_{j=1}^{C} \hat{y}_j = 1$ and $\hat{y}_j \in (0, 1)$

Cross-Entropy Loss

$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{j=1}^{C} y_j \log(\hat{y}_j)$$

For one-hot encoded labels (e.g., $\mathbf{y} = [0, 1, 0, 0]$ for class 2):

$$\mathcal{L}_{\text{CE}} = -\log(\hat{y}_c)$$

where c is the true class

Probabilistic interpretation: negative log-likelihood for categorical distribution

Gradient Descent Preview

The Optimization Strategy

- We need to minimize $\mathcal{L}(\boldsymbol{\theta})$ over all parameters
- Problem: high-dimensional, non-convex, no closed-form solution
- Solution: iterative gradient-based optimization

Gradient Descent (simplified)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)$$

- $\eta > 0$ is the learning rate (step size)
- Move in direction opposite to gradient (steepest descent)
- Repeat until convergence

Key Challenge How do we compute $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ efficiently for deep networks?
Answer: Backpropagation (next lecture!)

Outline

Course Structure

Weeks 1-3: Foundations

- Neural network basics, backpropagation, computational graphs
- Optimization algorithms (SGD, momentum, Adam)
- Regularization techniques (dropout, batch normalization, weight decay)

Weeks 4-6: Convolutional Networks

- CNN architectures and design principles
- Transfer learning and data augmentation

Weeks 7-9: Sequential Models

- RNNs, LSTMs, GRUs
- Attention mechanisms and Transformers

Weeks 10-14: Advanced Topics

- Generative models, self-supervised learning
- Practical considerations: training at scale, deployment
- Ethics and fairness

Logistics

Course Components

- **Lectures:** 1.5 hours per week (theory and concepts)
- **Exercises:** 1.5 hours per week (implementation and problem-solving)
- **Semester Project:** Train models on real datasets
- **Parallel course:** Basic ML (covers general ML foundations)

Prerequisites

- Programming: Python, NumPy (we'll use PyTorch or TensorFlow)
- Math: Linear algebra, calculus, basic probability
- We'll provide probability refreshers just-in-time as needed

Recommended Reading

- Simon J.D. Prince: *Understanding Deep Learning* (2023)
- Free online: `https://udlbook.github.io/udlbook/`

Next Week: Backpropagation and Optimization

What We'll Cover

- Computational graphs and automatic differentiation
- Backpropagation algorithm in detail
- Stochastic gradient descent and mini-batches
- Common optimization algorithms (momentum, Adam)

Preparation

- Review: chain rule from calculus
- Review: matrix calculus (gradients, Jacobians)
- Reading: Prince Chapter 6 (Fitting Models), Chapter 7 (Gradients and Initialization)

Exercise Session This Week

- Python/NumPy refresher
- Implementing activation functions
- Forward propagation from scratch