Artificial Neural Networks - Deep Learning

2- Perceptron and Neural Networks

Unat Teksen

1 Introduction to Deep Learning and AI

1.1 Relationships Among Fields

Deep Learning is often misunderstood in its relation to other fields. The hierarchy is strictly defined as:

- **Artificial Intelligence (AI):** The broad umbrella term for intelligence in machines.
- **Machine Learning (ML):** A subset of AI (systems that learn from data rather than being explicitly programmed).
- **Deep Learning (DL):** A subset of ML (models with multiple layers, specifically neural networks).

Timeline: AI excitement started earlier (1950s), ML flourished later, and the "DL boom" is a more recent phenomenon driven by hardware and data.

Important Statement: "Neural Networks are as old as Artificial Intelligence." Neural nets are not "new"; what is new is the combination of compute power (GPUs), massive data, and training tricks that enable modern Deep Learning.

- **Main takeaway:** Do not equate DL with AI. DL $\subset$ ML $\subset$ AI. The recent popularity is due to data availability, better hardware, and optimization techniques.

1.2 Historical Context and Motivation

- **Inception (1950s):** AI began with the Dartmouth proposal, aiming to create machine intelligence.
- **Two Approaches:** Early AI split into *Symbolic* (logic/rules) and *Connectionist* (neurons). This course focuses on the connectionist line.
- **Limitations of 1940s Computing:** Von Neumann machines were fast at arithmetic and following instructions but were rigid and sequential.
- **Motivation for Neural Networks:** Researchers needed systems capable of:
  - **Robustness:** Handling noisy, messy real-world data.
  - **Parallelism:** Being massively parallel and fault-tolerant (unlike fragile single pipelines).
  - **Adaptability:** Learning and changing behavior based on circumstances.

- **Main takeaway:** Neural networks were developed to overcome the limitations of sequential Von Neumann architecture by offering robustness, parallelism, and learning capabilities.

2 The Biological Inspiration and Artificial Neurons

2.1 The Brain Computational Model

The Perceptron is introduced as a computational model inspired by the brain's hardware. The brain's scale is massive:

- $\sim 10^{11}$ neurons (100 billion).
- $\sim 7,000$ synaptic connections per neuron.
- Total synapses: 10^{14} to 5×10^{14} in adults (up to 10^{15} in children).

Key Properties: Distributed among simple non-linear units, redundant (fault-tolerant), and intrinsically parallel.

- **Main takeaway:** "Distributed + non-linear + parallel + redundant" are the classic features used to justify the neural network approach.

2.2 Computation: Biological vs. Artificial

Biological Neuron: Dendrites receive signals. The neuron integrates them; if the internal potential passes a threshold, it "fires" through the Axon to other neurons via synapses.

Artificial Neuron Model:

- **Inputs:** Dendrites collect charges (inputs $x_1, \dots, x_I$) from synapses (weights), which can be inhibitory or excitatory.
- **Integration:** A cumulative charge is calculated (weighted sum).
- **Firing:** The neuron outputs a value once a threshold is passed.

2.2.1 Mathematical Formulation

The computation involves a weighted sum over inputs followed by an activation function $h_j(\cdot)$. The threshold is typically handled via the **Bias Trick**.

The bias b is represented as $b = -w_0$, allowing us to include it as an extra weight on a constant input $x_0 = 1$.

$$h_j(x; w, b) = h_j\left(\sum_{i=1}^{I} w_i x_i - b\right) = h_j\left(\sum_{i=0}^{I} w_i x_i\right) = h_j(w^T x) \tag{1}$$

Vector Form:

- Inputs: $[1, x_1, \dots, x_I]$
- Weights: $[w_0, w_1, \dots, w_I]$
- Output: $h_j(w^T x)$
- **Main takeaway:** Be comfortable rewriting "threshold b" as "bias weight w_0" with $x_0 = 1$. Be able to express the neuron in compact matrix/vector form.

3 The Perceptron

3.1 Historical Milestones

- **1943 (McCulloch Pitts):** Threshold Logic Unit (activation = threshold function/Heaviside step).
- **1957 (Frank Rosenblatt):** First Perceptron hardware (weights in potentiometers, updates via electric motors).
- **1960 (Bernard Widrow):** Represented threshold as a bias term in ADALINE (Adaptive Linear Neuron).

3.2 Capabilities: Boolean Operators

A perceptron can implement logic gates like OR and AND because they are linearly separable.

Perceptron as OR:

- Weights: $w_2 = 1, w_1 = 1, w_0 = -0.5$.
- Rule: Output 1 if $-0.5 + x_1 + x_2 > 0$, else 0.

Perceptron as AND:

- Weights: $w_2 = 1, w_1 = 1, w_0 = -1.5$ (or similar scaling like $w_0 = -2, w_1 = 1.5$).
- Rule: Output 1 if bias + sum ¿ 0.
- **Main takeaway:** The value is not just that OR/AND exist, but that a **single trainable hardware unit** can learn different operators simply by adjusting weights.

3.3 Hebbian Learning

"The strength of a synapse increases according to the simultaneous activation of the relative input and the desired target" (Hebb, 1949).

Update Rule:

$$w_i^{k+1} = w_i^k + \Delta w_i^k \tag{2}$$

$$\Delta w_i^k = \eta \, x_i^k \, t^k \tag{3}$$

Where:

- η: Learning rate.
- x_i^k: i-th input at time k.
- t^k: Desired output/target at time k.

The weights are fixed one sample at a time (online) and only updated if the sample is not correctly predicted.

3.4 Convergence and Decision Boundary

The Perceptron computes a weighted sum and returns its Sign (Thresholding). It is a **linear classifier** where the decision boundary is the hyperplane:

$$w_0 + w_1 x_1 + \cdots + w_I x_I = 0 \tag{4}$$

In 2D, this represents a line. Changing w_0 translates the line.

Convergence Properties:

- The procedure converges **if and only if** the data is linearly separable (Perceptron Convergence Theorem).
- The final weights are not unique (they depend on initialization and sample order).

3.5 Limitations: The XOR Problem

What if the dataset does not have a linear separation boundary?

- The Perceptron fails.
- **Example:** XOR (Exclusive OR) with inputs $\{-1, +1\}$. Points $(-1, -1)$ and $(1, 1)$ are one class, while $(-1, 1)$ and $(1, -1)$ are another. No single straight line can separate them.

This limitation (highlighted by Minsky Papert, 1969) necessitates **Multi-Layer Perceptrons (MLP)** to create non-linear boundaries. Hebbian learning is insufficient for hidden layers because hidden neurons do not have direct target labels.

4 Feed Forward Neural Networks

4.1 Architecture

To solve non-linear problems, we arrange neurons in layers:

- **Input Layer:** I inputs (+ bias).
- **Hidden Layers:** Multiple layers ($J_1, J_2, \ldots$ neurons).
- **Output Layer:** K neurons.

Layers are connected through weights $W^{(l)} = w_{ji}^{(l)}$ (weight from neuron i in layer $l-1$ to neuron j in layer l). The property is **Feed-Forward**: output depends only on previous layers.

$$h^{(l)} = h_j^{(l)}(h^{(l-1)}, W^{(l)}) \tag{5}$$

Activation functions must be differentiable to allow training via Backpropagation.

4.2 Activation Functions

- **Linear:** $g(a) = a$. Derivative $g'(a) = 1$. Stacking linear layers collapses the network to a single linear model.
- **Sigmoid:** $g(a) = \frac{1}{1+\exp(-a)}$. Output in $(0, 1)$. Derivative: $g'(a) = g(a)(1 - g(a))$.
- **Tanh:** $g(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$. Output in $(-1, 1)$. Derivative: $g'(a) = 1 - g(a)^2$.

4.3 Output Layer Selection

The choice of output activation depends on the task:

- **Regression:** Linear activation (Output $\in \mathbb{R}$).
- **Binary Classification:**
  - Labels $\{-1, +1\}$: Tanh.
  - Labels $\{0, 1\}$: Sigmoid (interpreted as posterior probability).
- **Multi-class Classification:** Softmax with K output neurons (One-hot encoding).

$$y_k = \frac{\exp(z_k)}{\sum_{k=1}^{K} \exp(z_k)} \tag{6}$$

4.4 Universal Approximation Theorem

A single hidden-layer feedforward net with S-shaped activations can approximate any measurable function on a compact set to any desired accuracy (Hornik, 1991).

- **Main takeaway:** This guarantees representation power, but not learnability. It might require an exponential number of hidden units or fail to generalize.

5 Optimization and Learning

5.1 Supervised Learning Framework

We aim to learn parameters w such that for a dataset $D = \langle x_n, t_n \rangle$, the model output $g(x_n, w)$ approximates t_n. We minimize the Sum of Squared Errors (SSE):

$$E = \sum_{n=1}^{N} (t_n - g(x_n, w))^2 \tag{7}$$

For linear models, this is convex. For Neural Networks, this is a **non-linear optimization** problem due to activation functions.

5.2 Gradient Descent

Since closed-form solutions are practically never available for NNs, we use iterative Gradient Descent.

$$w^{k+1} = w^k - \eta \left. \frac{\partial E(w)}{\partial w} \right|_{w^k} \tag{8}$$

Momentum: Adds "inertia" to avoid oscillation and escape shallow local minima.

$$w^{k+1} = w^k - \eta \frac{\partial E}{\partial w} - \alpha \left. \frac{\partial E}{\partial w} \right|_{w^{k-1}} \tag{9}$$

5.3 Variations of Gradient Descent

1. **Batch GD:** Uses all N data points. Precise but computationally expensive.
$$\frac{\partial E}{\partial w} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial E(x_n, w)}{\partial w} \tag{10}$$
2. **Stochastic GD (SGD):** Uses a single sample. Unbiased but high variance.
3. **Mini-batch GD:** Uses a subset $M < N$. Good trade-off between variance and computation.

6 Backpropagation

Manually deriving gradients for every weight in a deep network is impractical ("Can I make it automatic?"). Backpropagation provides a systematic way to compute gradients.

6.1 The Chain Rule

Neural networks are compositions of functions. For $y = g(x)$ and $z = f(y)$, the derivative is:
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = f'(y)g'(x) \tag{11}$$
Backpropagation applies this recursively. Weights update can be done in parallel, locally, and requires just two passes:

- **Forward Pass:** Compute activations h_j and outputs g.
- **Backward Pass:** Propagate gradients $\frac{\partial E}{\partial w}$ from output to input.

6.2 Derivative Example

For a weight $w_{ji}^{(1)}$ (hidden layer), the gradient involves the error term, output derivative, downstream weights, hidden derivative, and input:
$$\frac{\partial E}{\partial w_{ji}^{(1)}} \propto \sum_{n=1}^{N} (t_n - y_n) \cdot g'(\cdot) \cdot w_{kj}^{(2)} \cdot h_j'(\cdot) \cdot x_{i,n} \tag{12}$$
Main takeaway: Recognize the Backprop pattern: Error $\times$ Derivative $\times$ Weight $\times$ Derivative $\times$ Input.

7 Statistical Learning Perspective

7.1 Maximum Likelihood Estimation (MLE)

MLE chooses parameters that maximize the probability of the observed data. **The Recipe:**

1. Define likelihood $L = P(\text{Data}|\theta)$.
2. Take Log-likelihood $l = \log P(\text{Data}|\theta)$ (turns products to sums, easier to derive).
3. Compute derivatives and solve $\frac{\partial l}{\partial \theta} = 0$.

7.2 Gaussian Likelihood and SSE

Consider regression where targets t_n have Gaussian noise:

$$t_n \sim \mathcal{N}(g(x_n|w), \sigma^2) \tag{13}$$

The likelihood function is:

$$L(w) = \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t_n - g(x_n|w))^2}{2\sigma^2}\right) \tag{14}$$

Taking the log-likelihood:

$$l(w) \propto -\sum_{n=1}^{N} (t_n - g(x_n|w))^2 \tag{15}$$

- **Main takeaway:** Maximizing the Likelihood (MLE) under a Gaussian noise assumption is mathematically equivalent to **minimizing the Sum of Squared Errors (SSE)**. This justifies why SSE is the natural loss function for regression.

8 Exam Cheat Sheet and Practice Questions

8.1 Exam Cheat Sheet

8.1.1 Core Hierarchy and History

- **Hierarchy:** AI $\supset$ ML $\supset$ Deep Learning (DL is a subset of ML, which is a subset of AI).

8.1.2 Artificial Neuron / Perceptron

- Computes a weighted sum + activation:

$$a = w^T x, \quad y = h(a) \tag{16}$$

- **Bias Trick:** Include $x_0 = 1$ and weight $w_0 = -b$, so:

$$a = \sum_{i=0}^{I} w_i x_i \tag{17}$$

- **Decision Boundary:** $w^T x + w_0 = 0$. In 2D:

$$x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} x_1 \tag{18}$$

- **capabilities:** OR and AND are linearly separable (Perceptron works). XOR is **not** linearly separable (Single Perceptron fails).

8.1.3 Multi-layer Perceptron (Feed Forward NN)

- Layers with weights $W^{(l)}$; output depends only on previous layers.
- Hidden layers + non-linear activations allow modeling non-linear boundaries (fixes XOR).

8.1.4 Activations and Derivatives (Must Know)

- **Linear:** $g(a) = a \implies g'(a) = 1$.
- **Sigmoid:** $g(a) = \frac{1}{1+e^{-a}} \implies g'(a) = g(a)(1 - g(a))$.
- **Tanh:** Output in $(-1, 1) \implies g'(a) = 1 - g(a)^2$.

8.1.5 Output Layer & Loss Choice

- **Regression:** Linear output → Minimize SSE (Assumption: Gaussian Noise).

$$E = \sum_{n=1}^{N} (t_n - g(x_n, w))^2 \tag{19}$$

- **Binary Classification:**
  - Labels $\{0, 1\}$ → Sigmoid → Minimize Binary Cross-Entropy (Assumption: Bernoulli).
  - Labels $\{-1, +1\}$ → Tanh.
- **Multi-class:** Softmax + One-hot encoding.

8.1.6 Learning and Optimization

- **Gradient Descent Update:** $w^{k+1} = w^k - \eta \nabla_w E$.
- **Batch vs. SGD vs. Mini-batch:**
  - **Batch:** Full dataset per update (Slow).
  - **SGD:** 1 sample per update (Noisy/High Variance).
  - **Mini-batch:** Best tradeoff (Used in practice).
- **Backpropagation:** Application of the chain rule in 2 passes (Forward to compute activations, Backward to propagate gradients).

8.1.7 Perceptron Update Rule

For a misclassified point (x_i, t_i):

$$w \leftarrow w + \eta t_i x_i, \quad w_0 \leftarrow w_0 + \eta t_i \tag{20}$$

8.2 Practice Questions

1. **Bias Trick:** Show how a threshold b is absorbed into weights.
   Answer: Write $a = \sum_{i=1}^{I} w_i x_i - b$. Let $x_0 = 1$ and $w_0 = -b$. Then $a = \sum_{i=0}^{I} w_i x_i = w^T x$.
2. **Decision Boundary:** A perceptron has $w_0 = -2, w_1 = 1.5, w_2 = 1$. Write the boundary line x_2 as a function of x_1.
   Answer: $w_0 + w_1 x_1 + w_2 x_2 = 0 \implies -2 + 1.5x_1 + x_2 = 0 \implies x_2 = 2 - 1.5x_1$.

3. **OR Perceptron Check:** Using $w_0 = -0.5, w_1 = 1, w_2 = 1$ (inputs 0/1), check outputs.
   Answer: $a = -0.5 + x_1 + x_2$.
   - (0,0): $a = -0.5 \to 0$
   - (1,0): $a = 0.5 \to 1$
   - (0,1): $a = 0.5 \to 1$
   - (1,1): $a = 1.5 \to 1$
4. **XOR Impossibility:** Why can't a single perceptron solve XOR?
   Answer: XOR labels opposite corners with the same class. No single line/hyperplane can separate them (not linearly separable).
5. **Derivatives:** Give $g'(a)$ for Sigmoid and Tanh.
   Answer: Sigmoid: $g'(a) = g(a)(1 - g(a))$. Tanh: $g'(a) = 1 - g(a)^2$.
6. **Output Activation Choice:** What to use for (i) Regression, (ii) Binary $\{0, 1\}$, (iii) K-class?
   Answer: (i) Linear, (ii) Sigmoid, (iii) Softmax.
7. **MLE to SSE:** Under Gaussian noise $t \sim \mathcal{N}(g(x|w), \sigma^2)$, what loss maximizes likelihood?
   Answer: Minimize SSE $\sum(t_n - g(x_n|w))^2$.
8. **MLE to Cross-Entropy:** Under Bernoulli assumption, what is the loss?
   Answer: Negative Log-Likelihood:
   $$E = -\sum_n [t_n \log g(x_n) + (1 - t_n) \log(1 - g(x_n))]$$
9. **Gradient Variations:** Which has the highest variance? Which is used in practice?
   Answer: Highest variance: SGD (1 sample). Used in practice: Mini-batch.
10. **Perceptron Update:** Update rule for misclassified (x, t) where $t \in \{-1, +1\}$.
    Answer: $w \leftarrow w + \eta t x$.