Introduction to Neural Networks

Learning from Scratch

by Rishikesh Yadav

Nov 19, 2025

Assistant Professor, School of Mathematical and Statistical Sciences, IIT Mandi, India

Table of Contents

Course Objectives

The objectives of this lecture would be:

- Recap **regression** and **classification** techniques.
- Understand key ideas from scratch in **neural networks**.
- Build and train simple **neural networks** from scratch and also using **TensorFlow**.
- Bridge **traditional statistical regression methods** with **neural network**.

Some Good References

- Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer.
- Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (3rd ed.). O'Reilly. [GitHub companion code]
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. [Free online]
- Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Manning.
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. [Free online book]
- Online learning platforms:
  - Coursera - Deep Learning Specialization
  - fast.ai - Practical Deep Learning for Coders
  - DeepLearning.AI

Recap: Regression and Classification

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - Inputs (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$
  - Outputs (targets): y_i (numeric or categorical)

- **Learning:** Estimate the unknown parameters (weights in machine learning) $\mathbf{w}$ by minimizing a suitable **loss function** $\mathcal{L}(\mathbf{w})$.
- **Prediction:** Once $\hat{\mathbf{w}}$ is obtained, use

$$\hat{y} = f(\mathbf{x}_0; \hat{\mathbf{w}})$$

  for new input $\mathbf{x}_0$.

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.
- **One simpler model (Assume f is linear in w):** Multiple Linear Regression (MLR)

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip} + \varepsilon_i$$

- **Loss function (Mean Squared Error):**

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- **Ordinary Least Squares (OLS) solution:**

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- **Important Note**: For all regression types of problems, we might not get the closed-form expressions of $\hat{\mathbf{w}}$.

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \ldots, K\}$.
- **Model:** Logistic regression (binary or multinomial)

$$P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})}$$

- **Loss function (Cross-Entropy / Log-Loss):**

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}\{y_i = k\} \log P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

- **Estimation:** No closed-form solution; parameters $\mathbf{w}$ are found using iterative optimization methods (e.g., gradient descent, Adam, Newton–Raphson etc.)
- **Prediction:** Assign the class with the highest predicted probability

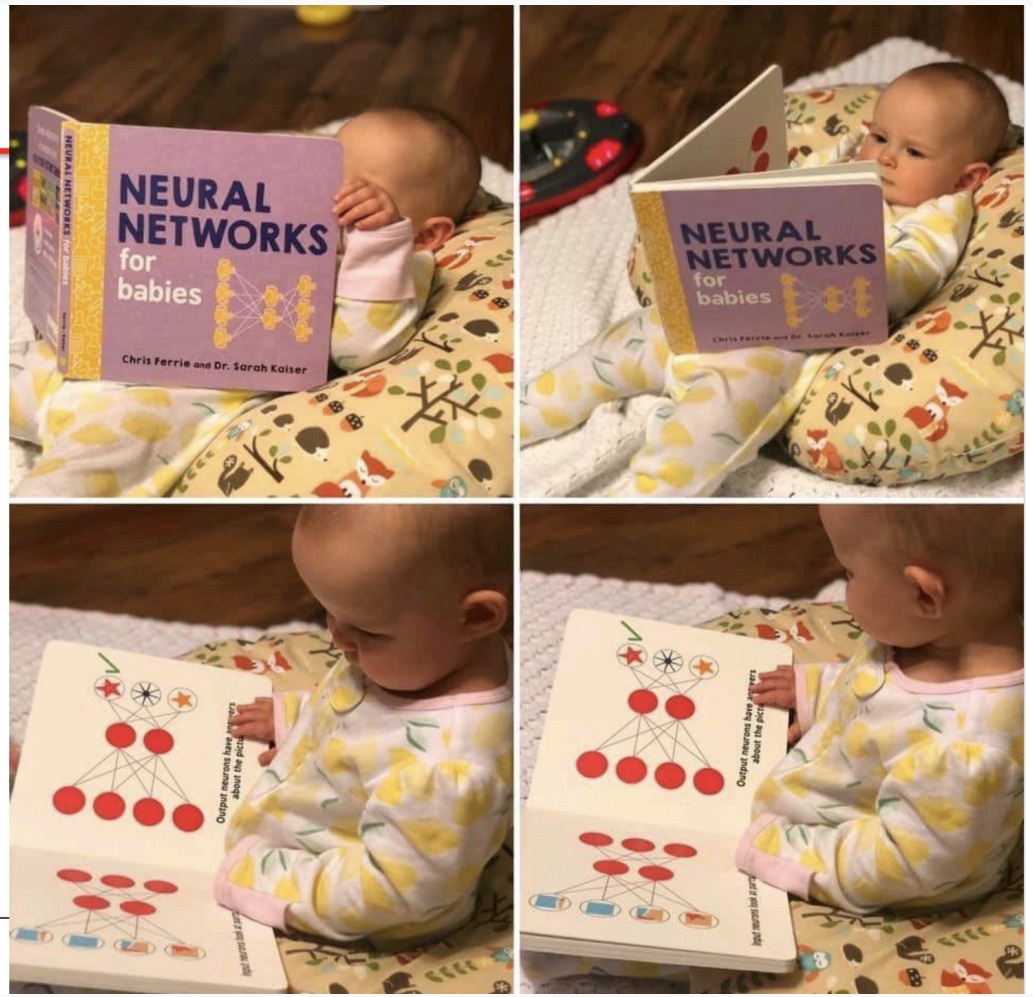
$$\hat{y}_i = \arg \max_{k \in \{1,\ldots,K\}} P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

Neural Networks

Everybody is learning

Never too late

😀



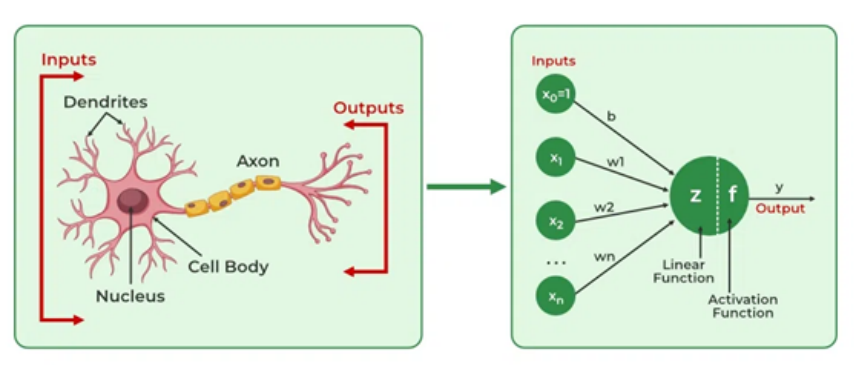
[1]

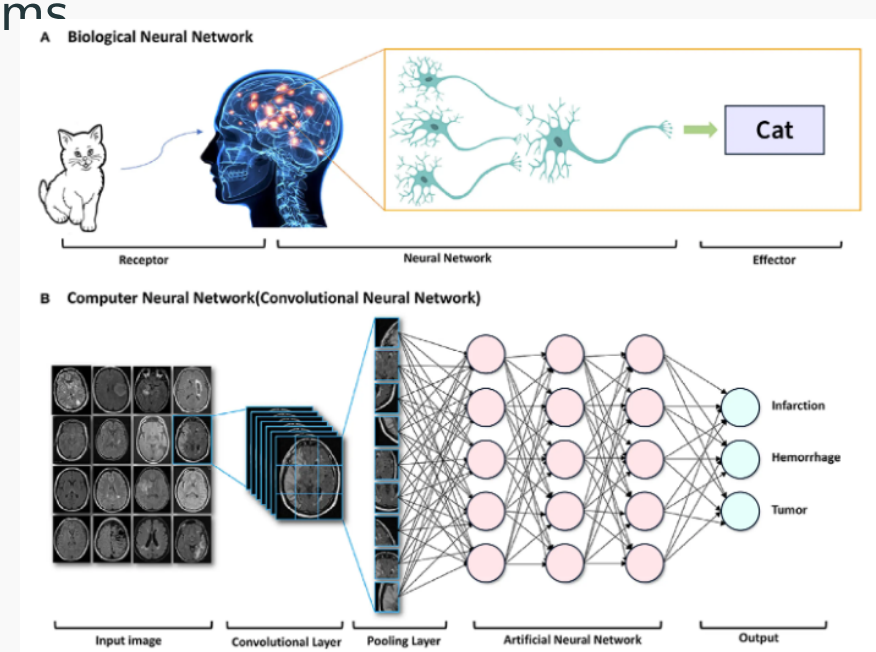
[1] CS 229: Prof. Xiangliang Zhang (KAUST)

Agenda

What is a Neural Network?

- Neural networks mimic how the human brain functions.
- Brain activity occurs when a **stimulus** enters the system; information is processed through the network via neurons that extract relevant information, and this information is passed to another area.
- A **Neural Network** is a computational model inspired by the way biological neural networks in the human brain process information. It consists of **layers** of interconnected **nodes**, or **neurons**, which work together to solve complex problems.





Historical Development of Deep Learning

- **Early Origins (1940s–1960s)**: McCulloch–Pitts (1943): first neuron model, Rosenblatt's Perceptron (1958): first trainable neural network.
- **First AI Winter (1970s)**: Minsky & Papert (1969) show perceptron limits (XOR), Loss of funding & shift to symbolic AI.
- **Backpropagation Revolution (1980s)**: Werbos (1974), Rumelhart–Hinton–Williams (1986) $\rightarrow$ training multi-layer NNs.
- **Second AI Winter (1987–1995)**: Hardware & data insufficient, neural nets sidelined; symbolic/statistical AI rise.
- **Statistical Learning Era (1990s–2000s)**: SVMs (Vapnik), Probabilistic Graphical Models (Pearl), strong math foundations (optimization, RKHS).
- **Deep Learning Boom (2006–2015)**: DBNs (Hinton) in 2006, ImageNet/AlexNet (2012), Word2Vec, GANs.
- **Modern Era (2017–Present)**: Transformers (2017), LLMs (GPT, BERT), AlphaGo, Diffusion Models, Multimodal AI.

Components of Neural Networks

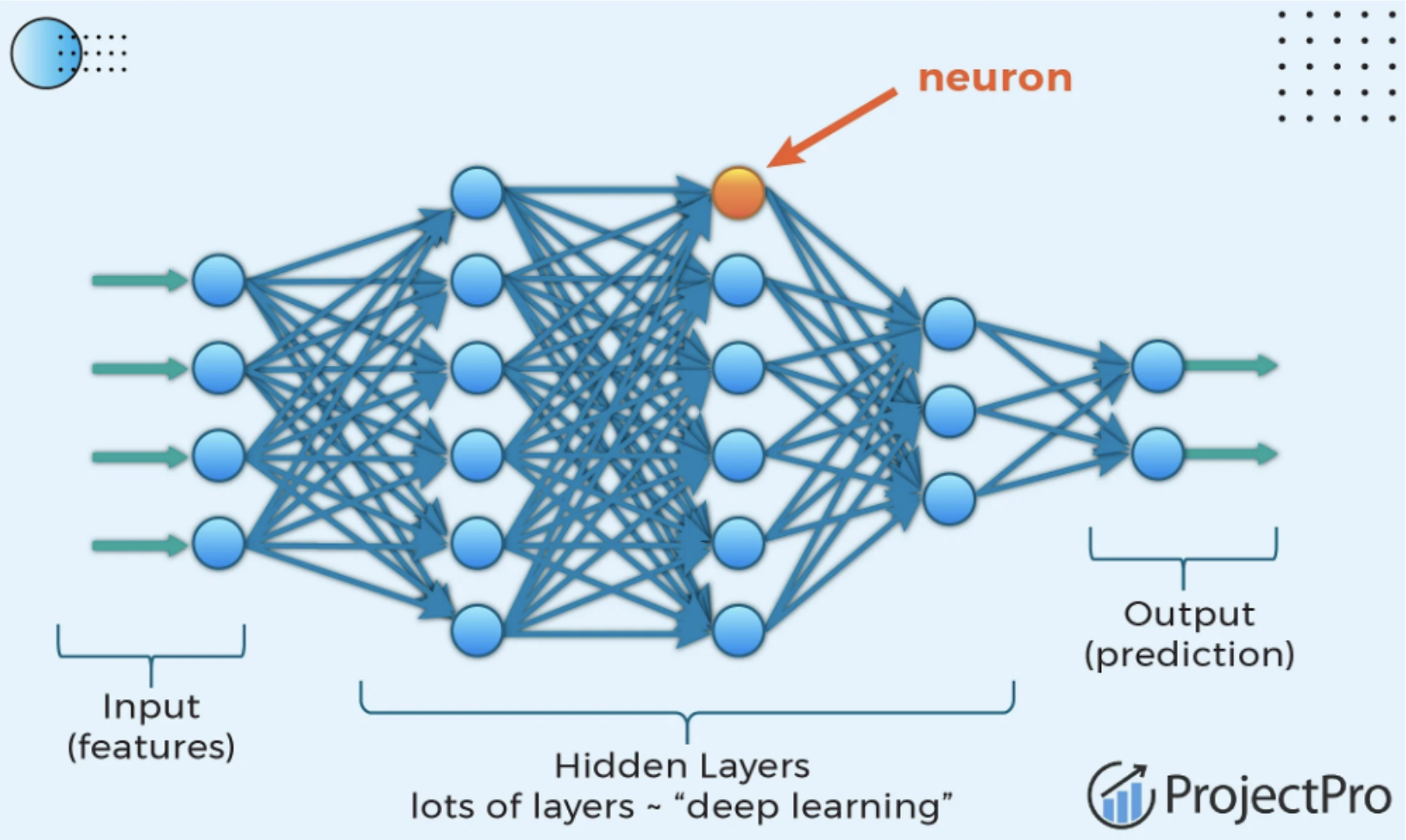
Structural Components

- Layers
- Neurons
- Weights
- Bias
- Activation Functions

Training-Related Components

- Loss Function
- Learning Rate
- Training Data
- Validation and Test Data
- Forward Propagation and Backpropagation

A Basic Architectures of Neural Networks



[2]

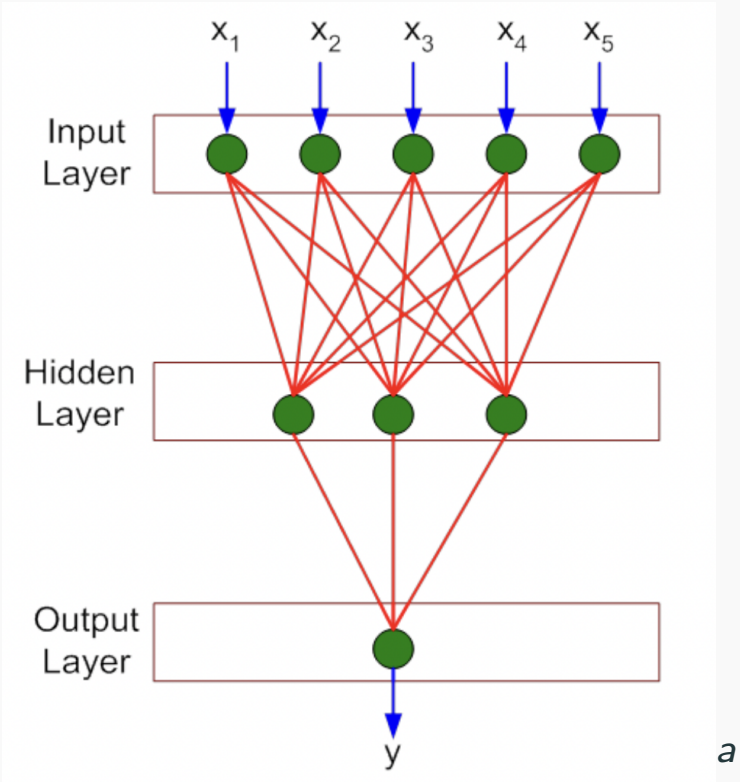
[2] ProjectPro

Agenda

Layers

Neural computing requires a number of neurons, to be connected together into a neural network. Neurons are arranged in layers.

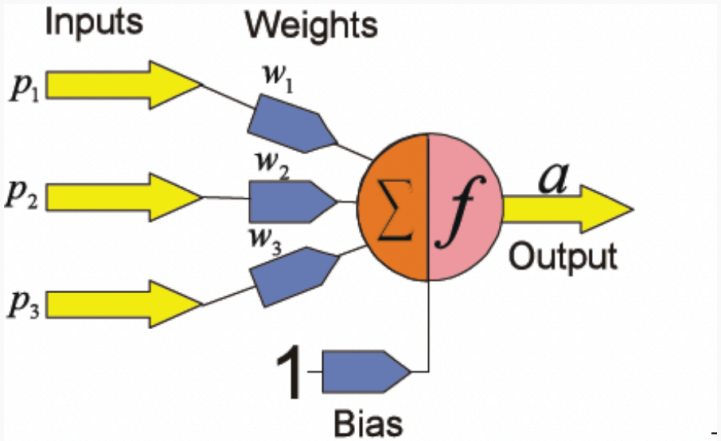
- **Input Layer**
  - First layer of neurons.
  - Receives raw data directly from the input features.
- **Hidden Layers**
  - Layers between the input and output layers.
  - Perform computations.
  - Can have one or many hidden layers.
- **Output Layer**
  - Final layer.
  - Produces the network's output.
  - Structure depends on the specific task (e.g., classification, regression).



[a] CS 229: Prof. Xiangliang Zhang (KAUST)

Neurons, Weights, and Biases

- **Neurons**:
  - Basic units of a neural network.
  - Receive input, process it, and pass it to other neurons.
  - Apply a mathematical function to generate an output.
- **Weights**:
  - Parameters that transform input data within the network.
  - Each connection between neurons has an associated weight.
- **Biases**:
  - Additional parameters in a neuron that allow to fit the data better.
  - Provide ability to each neuron have output not 0 when inputs are 0.



[3]

[3] CS 229: Prof. Xiangliang Zhang (KAUST)

Activation Functions (1)

- Activation functions introduce nonlinearity into the output of a neuron, allowing neural networks to model complex and non-linear relationships.
- While many activation functions exist, they generally satisfy two essential properties:
  - **Nonlinearity**: Without nonlinear activations, the entire network reduces to a pure linear model, regardless of the number of layers.
  - **Continuous differentiability**: Required for gradient-based optimization methods such as backpropagation.
- Activation functions also act similarly to **link functions** in generalized linear models (GLMs), transforming the linear predictor into a scale appropriate for the output.

A single-layer neural network(no hidden layers) is mathematically equivalent to a generalized linear model (GLM).

Activation Functions (2)

Classical Forms

Function	Expression	Pros	Cons	Uses
Sigmoid	$\frac{1}{1+e^{-x}}$	Smooth, bounded, probabilistic output	Vanishing gradients, not zero-centered	Binary classification
Softmax	$\frac{e^{x_i}}{\sum_j e^{x_j}}$	Probabilistic output vector	Sensitive to large logits	Multi-class output
Tanh	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	Zero-centered, stronger gradients	Saturates, vanishing gradients	RNNs, classical MLPs
ReLU	$\max(0, x)$	Fast, simple, no saturation for $x > 0$	Dying ReLU, unbounded output	CNNs, deep MLPs

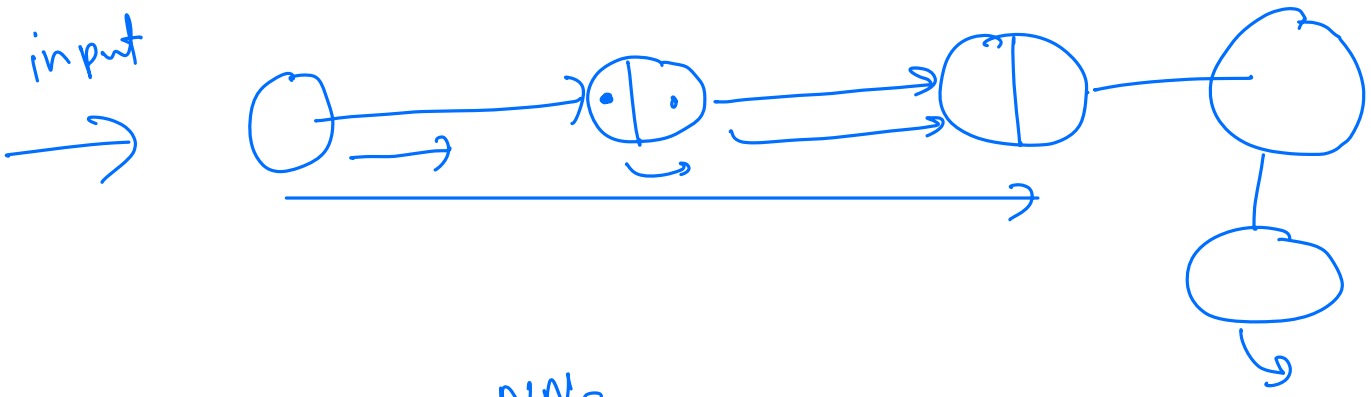
Modern Variants

Function	Expression	Pros	Cons	Uses
Leaky ReLU	$\begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$	Fixes dying ReLU, small negative slope	Slope α arbitrary	CNNs, MLPs
ELU	$\begin{cases} x, & x > 0 \\ \alpha(e^{x}-1), & x \leq 0 \end{cases}$	Negative outputs stabilize training	Expensive, may saturate	Deep CNNs
Swish	$x\,\sigma(x)$	Smooth, strong empirical performance	Slightly slower	Transformers, modern vision

[4] https://www.analytixlabs.co.in/blog/activation-function-in-neural-network/

Recall

NNs → Brain

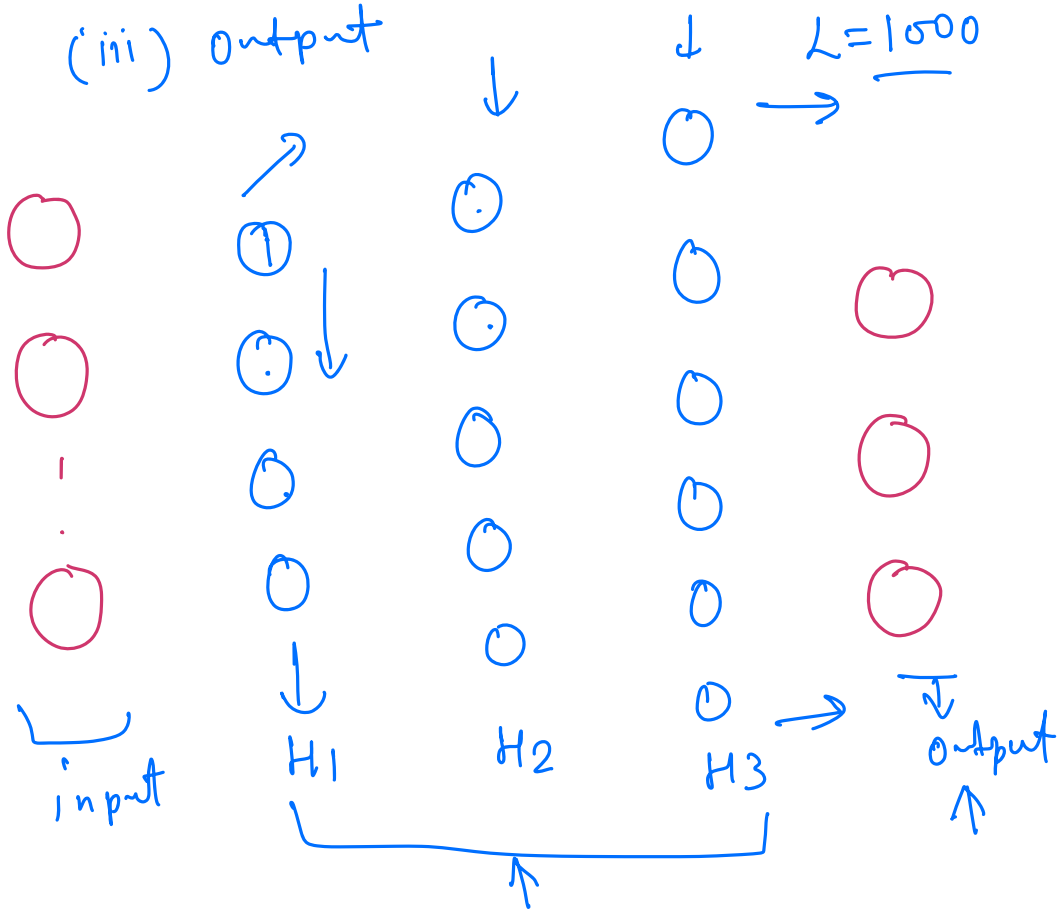


NNs

(i) input Layers

(ii) Hidden L

(iii) output



detect Image

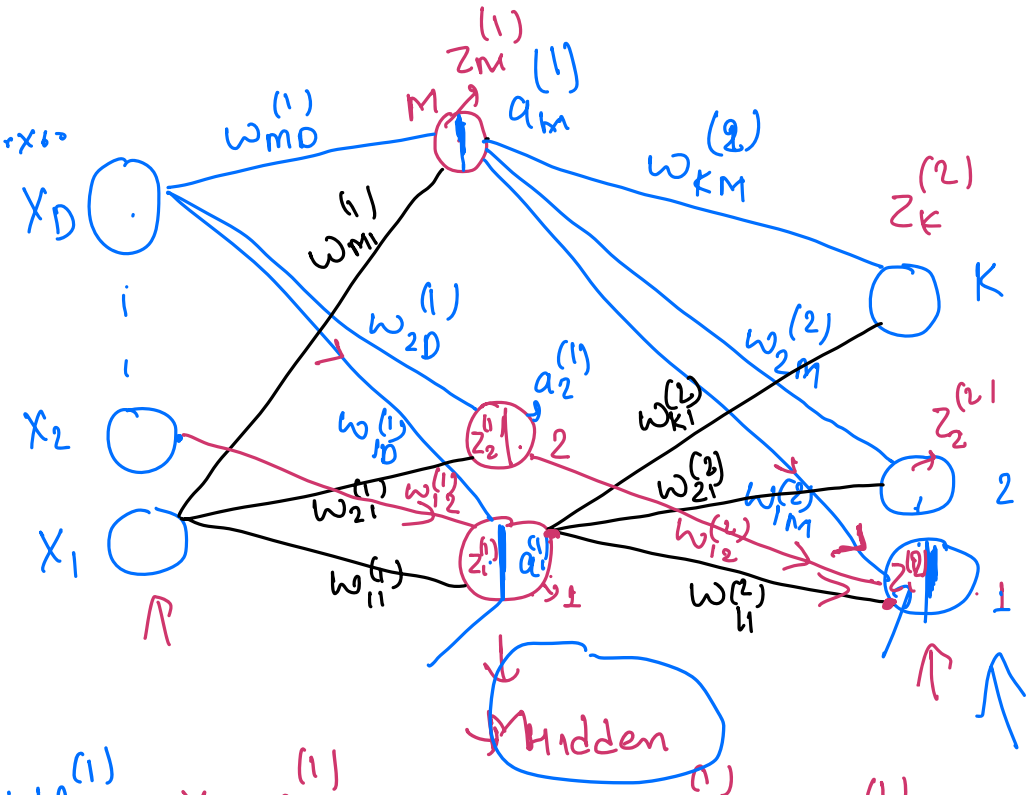
$D = 3600$

x_1: some

x_2: →

x_3: →

!



pre-activation $z^{(1)}$

$$\Rightarrow z_1 = x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + \cdots x_D w_{1D}^{(1)} + w_{10}^{(1)}$$

$$z_1^{(1)} = \sum_{i=0}^{D} x_i \cdot w_{1i}^{(1)} \qquad x_0 = 1$$

pre-activation

$$z_J^{(1)} = \sum_{i=0}^{D} x_i \cdot w_{Ji}^{(1)} \qquad x_0 = 1$$

post-activation → h

pre-activation → $a_J^{(1)} = h\left(z_J^{(1)}\right), \quad J = 1 \cdots M$

$$z_1^{(2)} = a_1^{(1)} w_{11}^{(2)} + a_2^{(1)} w_{12}^{(2)} + \cdots + a_M^{(1)} w_{1M} + w_{10}^{(2)}$$

$$= \sum_{J=0}^{M} a_J^{(1)} w_{1J}^{(2)} \qquad a_0^{(1)} = 1$$

$$z_K^{(2)} = \sum_{J=0}^{M} a_J^{(1)} w_{KJ}^{(2)}, \quad K = 1 \cdots k$$

Hidden

$$\Rightarrow z_1^{(1)} = x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + \cdots + x_D w_{1D}^{(1)} + w_{10}^{(1)}$$

$$z_1^{(1)} = \sum_{i=0}^{D} x_i w_{1i}^{(1)} \qquad x_0 = 1$$

pre-activation

$$z_j^{(1)} = \sum_{i=0}^{D} x_i w_{ji}^{(1)} \qquad x_0 = 1$$

$a_i^{(0)}$

post-activation → h

pre-activation → $a_j^{(1)} = h(z_j^{(1)})$, $j = 1 \cdots M$

$$z_1^{(2)} = a_1^{(1)} w_{11}^{(2)} + a_2^{(1)} w_{12}^{(2)} + \cdots + a_M^{(1)} w_{1n}^{(2)}$$

$$= \sum_{j=0}^{M} a_j^{(1)} w_{1j}^{(2)} \qquad a_0^{(1)} = 1$$

$$z_k^{(2)} = \sum_{j=0}^{M} a_j^{(1)} w_{kj}^{(2)}, \quad k = 1 \cdots K$$

post-activation:

$$y_k = a_k^{(2)} = \sigma(z_k^{(2)}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}\right)$$

$$= \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h(z_j^{(1)})\right)$$

$$y_k = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} x_i w_{ji}^{(1)}\right)\right)$$

Agenda

Notation

- $\mathbf{x} \in \mathbb{R}^d$: d-dimensional input vector (features)
- $\mathbf{W}^{(l)}$: Weight matrix for layer l
- $\mathbf{b}^{(l)}$: Bias vector for layer l
- $\mathbf{z}^{(l)}$: Pre-activation (linear part)
- $\mathbf{a}^{(l)}$: Activation (output) of layer l
- $\sigma(\cdot)$: Activation function (ReLU, sigmoid, etc.)
- Total number of hidden layers will be denoted by H, and dimensions of output would be K.
- $\mathbf{t}$ is the target variables of interest.

We define $a^{(0)} = x$ as the input layer.

General Overview

- Forward propagation computes the output of the neural network by passing inputs through each layer.
- Each layer performs:

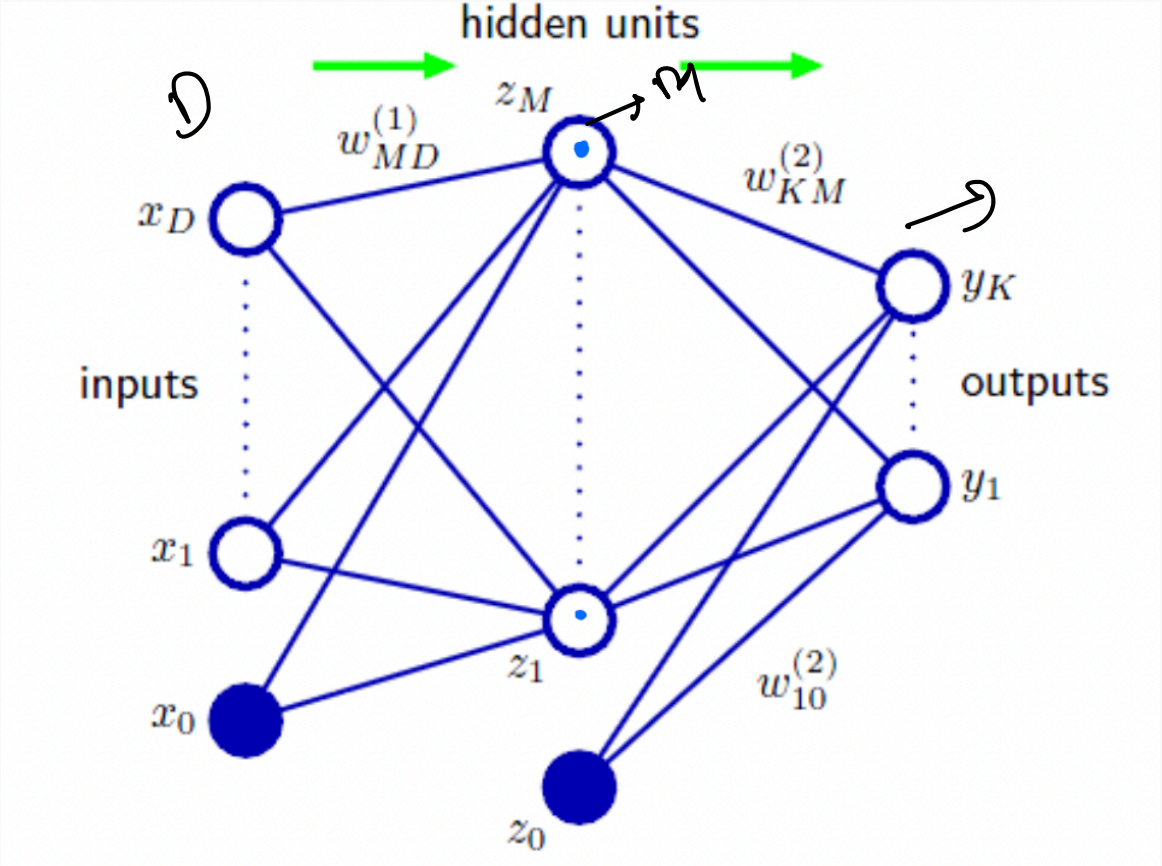
$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

  where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the learnable parameters.
- The activation function transforms the linear output:

$$\mathbf{a}^{(l)} = \sigma\left(\mathbf{z}^{(l)}\right)$$

- This process continues layer by layer until the final output is produced.

Example: Unfolding a Single-Hidden-Layer (Two-Layer) Neural Network



- D input variables
- M number of hidden neurons
- K outputs
- Use the two activation functions onthe weights sum on hidden layers and output layers

A Two-Layer Neural Network (2)

Construct M linear combinations of the inputs $x_1, \ldots, x_d$

$$z_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \qquad x_0 = 1.$$

- $z_j^{(1)}$ are the pre-activations on j^{th} neurons of hidden layers, $j = 1, \ldots, M$.
- $w_{ji}^{(1)}$ are the layer one weights, $i = 1, \ldots, D$.
- $w_{j0}^{(1)}$ are the layer one biases.

Each linear combination $z_j^{(1)}$ is transformed by a (non-linear differentiable) activation function,

$$a_j^{(1)} = h(z_j^{(1)}),$$

which will be the activation now at the j^{th} neurons of the hidden layer and act as an input for the next layers (here output).

A Two-Layer Neural Network (3)

The hidden outputs $a_j^{(1)} = h(z_j^{(1)})$ are linearly combined in layer two:

$$z_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}, \qquad z_0 = 1.$$

- $z_k^{(2)}$ are the output pre-activations at the k^{th} output neurons, $k = 1, \ldots, K$.
- $w_{kj}^{(2)}$ are the layer two weights, $j = 1, \ldots, M$.
- $w_{k0}^{(2)}$ are the layer one biases.

The output pre-activations $z_k^{(2)}$ are transformed by the output (non-linear differentiable) activation function

$$y_k = \sigma(z_k^{(2)}).$$

- y_k are the final outputs.
- $\sigma(\cdot)$ is like $h(\cdot)$, but often sigmoid function for classification and linear function for Regressions.

A Two-Layer Neural Network (4)

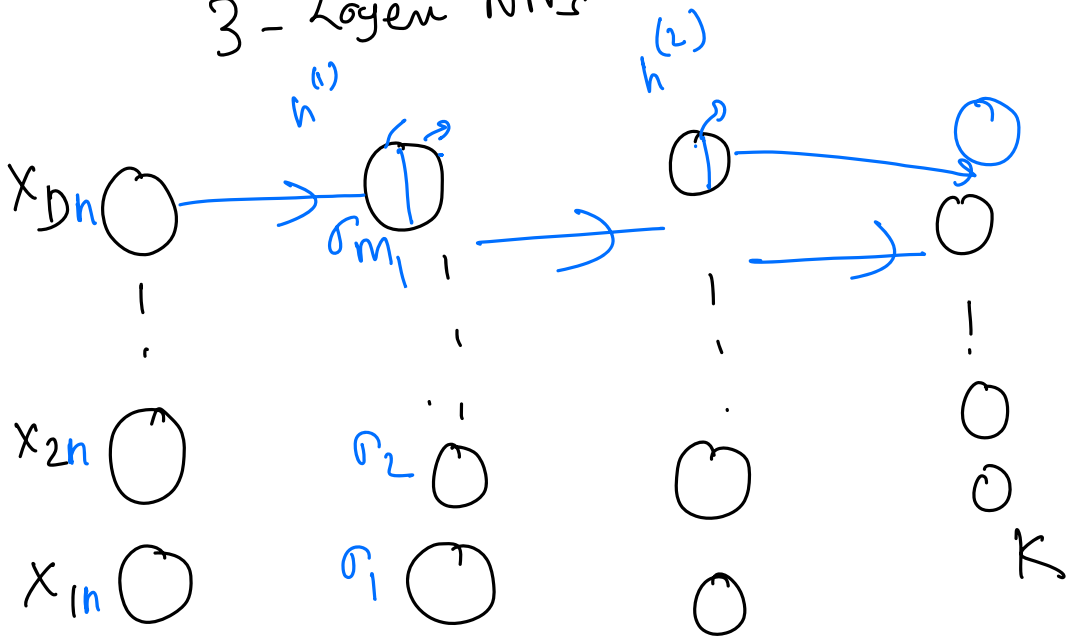
After substituting $y_k = \sigma(z_k^{(2)})$;

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right).$$

Evaluation of this is called **forward propagation**.

- Here, $h(\cdot)$ and $\sigma(\cdot)$ are activation functions.
- There could be several possibilities of activation function; for example:
  - If If $\sigma(\cdot)$ is the sigmoid function, then a binary classification model is obtained, despite any choice of activation $h(\cdot)$
  - If $\sigma(\cdot)$ is the identity function, then a regression model is obtained.
  - If $\sigma(\cdot)$ $h(\cdot)$ both are identity functions, then we get a Linear regression; for example, MLR

3 - Layer NNs



1000 images

$\rightarrow N = 1000$

$n = 1$

$$\downarrow \quad \begin{matrix} x_{11} \\ x_{21} \\ \vdots \\ x_{D1} \end{matrix} \begin{bmatrix} \quad \end{bmatrix} \rightarrow$$

M_1 $\quad$ M_2

$w^{(1)}, w^{(2)}, w^{(3)}$ $\rightarrow$

$n = 2 =$ $\begin{bmatrix} \quad \end{bmatrix}$ $w^{(1)}, w^{(2)}, w^{(3)}$

$w^{(1)}, w^{(2)}, w^{(3)}$

$n = N \rightarrow$ $\begin{bmatrix} \quad \end{bmatrix}$

Exercise: Three-Layer Neural Network (Forward Propagation)

Similar to two-layer neural network show that the network output for a three layer-neural network would be

$$\hat{y}_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{m=0}^{M_2} w_{km}^{(3)}\, h^{(2)}\left(\sum_{j=0}^{M_1} w_{mj}^{(2)}\, h^{(1)}\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)\right).$$

Here:

- $w_{ji}^{(1)}$ are weights from input components x_i to first hidden layer neuron j (with $i = 0$ representing bias and $x_0 = 1$).
- $h^{(1)}$ and $h^{(2)}$ are the activation functions for hidden layer 1 and hidden layer 2 respectively.
- $w_{mj}^{(2)}$ are weights from first hidden layer (j) to second hidden layer (m); $w_{km}^{(3)}$ are weights from second hidden layer (m) to output unit k (again $m = 0$ may be bias).
- D = input dimension, M_1 = number of units in first hidden layer, M_2 = number of units in second hidden layer.

Special cases (Think about these as well):

- If $h^{(1)}, h^{(2)}, \sigma$ are *identity*, the whole map reduces to a single affine function of $\mathbf{x}$ (linear regression).
- If σ is softmax and the hidden activations are non-linear, this is a multi-class classification model.

$\underline{x}_n = (x_{1n}, x_{2n} \dots x_{Dn})$ ↓

$y_k(x_n, w) =$ ------ x_{in}

$y_k(x_1, w)$ — - - - x_{1n}

$y_k(x_2, w)$ → - - . x_{2n}

$y_k(x_N, w)$ — - x_{Nn}

↳ **Regression :-** → $K = 1$

$w = (w^{(1)}, w^{(2)}, w^{(3)})$

$\hat{y}(\underline{x}_1, w) \longrightarrow t_1$

$\hat{y}(\underline{x}_2, w) \longrightarrow t_2$

$\hat{y}(\underline{x}_3, w) \longrightarrow t_3$

$\hat{y}(\underline{x}_N, w) - - - t_N$

MSE

$$\mathcal{L}(w) = \frac{1}{2n} \sum_{n=1}^{N} \left(\hat{y}(x_n, w) - t_n \right)^2$$

for 2-layer NN

$$\mathcal{L}(w) = -\frac{1}{2n} \sum_{n=1}^{N} \left(\sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_{in} \right) \right) - t_n \right)^2$$

$w = (w^{(1)}, w^{(2)})$

↳ **Recall:**

→ gradient D

→ NR

→ Adam

→ Rmsprop

$\sigma(z_F^{(2)})$

NN

$$L(w) = -\frac{1}{2n} \sum_{n=1}^{N} \left(\sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)}\, h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_{in} \right) \right) - t_n \right)^2$$

y_n

$w = (w^{(1)}, w^{(2)})$

↳ Adam

(i) momentum

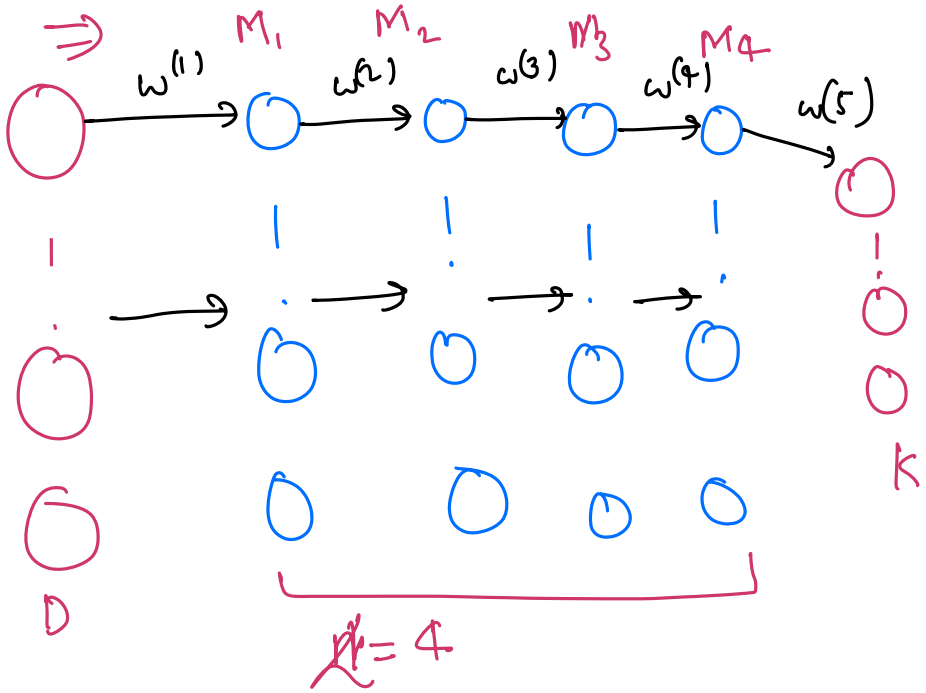
$$\frac{\partial L(w)}{\partial w^{(2)}} \rightarrow \frac{\partial L(w)}{\partial w^{(1)}}$$

$$\frac{\partial L(w)}{\partial w_{kj}^{(2)}} = \frac{\partial L}{\partial \sigma} \times \frac{\partial \sigma}{z_k^{(2)}} \times \frac{\partial z_k^{(2)}}{\partial w_{kj}^{(2)}} \quad ③$$

1st Layer

$$= 2\,(\sigma(\;) - t_n)\; \sigma'(z_F^{(2)})\; h\left(z_j^{(1)}\right)$$

$$\frac{\partial L(w)}{\partial w_{ji}^{(1)}} = \leftrightarrow \quad ④ \quad (\;)\,(1-x)\,(\;) \rightarrow 100$$



(1) forward propagation

$$\hat{y}(x_n, w)$$

$$L(w) = -\frac{1}{2N} \sum_{n=1}^{N} \left(\hat{y}(x_n, w) - t_n \right)^2$$

(2) → calculate gradient

$$\frac{\partial L}{\partial w^{(5)}} \rightarrow \frac{\partial L}{\partial w^{(4)}} \rightarrow \frac{\partial L}{\partial w^{(3)}} \rightarrow \frac{\partial L}{\partial w^{(2)}} \rightarrow \frac{\partial L}{\partial w^{(1)}}$$

↳ update parameter using → Adam

(A1.) $w_0 = \underbrace{w_0^{(1)}, w_0^{(2)}, w_0^{(3)}, w_0^{(4)}, w_0^{(5)}}_{N}$

↳ $L(w_0) = \frac{1}{2} \sum_{n=1} \left(y(x_n, w_0) - t_n \right)$

↳ forward

↳ gradient

$$\frac{\partial L(w_0)}{\partial w_0^{(5)}} \longrightarrow \frac{\partial L(w_0)}{\partial w_0^{(1)}}$$

↳ update parameters

↳ $w_1^{(1)} = w_0^{(1)} - \eta \frac{\partial L(w_0)}{\partial w_0^{(1)}}$

$$w_1^{(5)} = w_0^{(5)} - \eta \frac{\partial L(w_0)}{\partial w_0^{(5)}}$$

↳ Forward propagation → $L(w_1)$

Back-prop ← → gradient

$w_2^{(1)} =$ ⟷

$w_2^{(5)} =$ →

↳ forward → $L(w_2)$ ↳ 1000, 2000

$y \to x_i^2$ ⟷ MLR ⟷ SLR

Agenda

Recall: Error (Loss) Function

Training a neural network involves estimating the weights and biases by optimizing a loss function.

- Multi-class classification: Neural networks with a softmax output layer use the Cross-entropy loss:

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log y_k(\mathbf{x}_n, \mathbf{w}),$$

  where:
  - K = number of classes,
  - $t_{nk} = 1$ if sample n belongs to class k, else 0,
  - $y_k(\mathbf{x}_n, \mathbf{w})$ = softmax probability for class k.

- Regression: Sum of squared

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N} \{y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n\}^2,$$

  where $t_n, n = 1, \ldots, N$ is the observed response. N is the total number of training samples (examples).

Training a Neural Network: Forward and Backward Pass

- **Goal**: Minimize the error/loss function $\mathcal{L}(\boldsymbol{w})$
- **Recall:** We will be using gradient based algorithm for minimizing the loss function which require calculation of gradient of loss function.
- **Each training step has two key phases:**
  - **Forward Pass**: Compute output predictions from input through the layers. (We covered this part previously)
  - **Backward Pass (Backpropagation)**: Compute gradients of loss w.r.t. weights using the chain rule.
- Then update weights using gradient descent (or a variant).

Backpropagation: overview

- **Goal:** compute $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{y}(\mathbf{x}, \mathbf{w}), \mathbf{t})$ efficiently for all weights in the network.
- Use chain rule to propagate "errors" from the output layer to backward through layers.
- For example, for a two-layer NNs (input $\rightarrow$ 1-hidden layer $\rightarrow$ output) we compute:

$$\frac{\partial \mathcal{L}}{\partial w_{kj}^{(2)}}, \qquad \frac{\partial \mathcal{L}}{\partial w_{ji}^{(1)}}$$

where indices follow the earlier slides in case of forward propagation:

$$z_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \quad x_0 = 1, \qquad a_j^{(1)} = h(z_j^{(1)}), \quad j = 1, \ldots, M$$

$$z_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} a_j^{(1)}, \quad a_0^{(1)} = 1, \qquad y_k = \sigma(z_k^{(2)}), \quad k = 1, \ldots, K.$$

Fist-Step: Calculate Output-layer gradients

Let $\mathcal{L}$ be loss. Define

$$\delta_k^{(2)} \equiv \frac{\partial \mathcal{L}}{\partial z_k^{(2)}}.$$

By chain rule,

$$\frac{\partial \mathcal{L}}{\partial w_{kj}^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} \, a_j^{(1)}.$$

Bias gradient:

$$\frac{\partial \mathcal{L}}{\partial w_{k0}^{(2)}} = \delta_k^{(2)}.$$

Key: $\delta_k^{(2)} = \frac{\partial \mathcal{L}}{\partial y_k} \sigma'(z_k^{(2)})$.

Output-layer $\delta_k^{(2)}$ — examples

(1) MSE + output activation σ:

$$\mathcal{L} = \tfrac{1}{2}\sum_k (y_k - t_k)^2 \Rightarrow \frac{\partial \mathcal{L}}{\partial y_k} = y_k - t_k,$$

$$\delta_k^{(2)} = (y_k - t_k)\,\sigma'(z_k^{(2)}).$$

(2) Softmax + CE:

$$\delta_k^{(2)} = y_k - t_k.$$

(Simplifies: no σ' term — standard for classification.)

Hidden-layer gradients: backpropagate deltas

We now propagate to **hidden units**. Define

$$\delta_j^{(1)} \equiv \frac{\partial \mathcal{L}}{\partial z_j^{(1)}}.$$

Using **chain rule** (sum over contributions of hidden unit to all outputs):

$$\delta_j^{(1)} = \sum_{k=1}^{K} \frac{\partial \mathcal{L}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial z_j^{(1)}} = h'(z_j^{(1)}) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_k^{(2)}.$$

Then **gradients for layer-1 weights**:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i, \qquad \frac{\partial \mathcal{L}}{\partial w_{j0}^{(1)}} = \delta_j^{(1)}.$$

Vectorized backpropagation (single example)

Using **column-vector** notation:

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x}, \quad \mathbf{a}^{(1)} = h(\mathbf{z}^{(1)}), \quad \mathbf{z}^{(2)} = W^{(2)}\mathbf{a}^{(1)}, \quad \mathbf{y} = \sigma(\mathbf{z}^{(2)}).$$

Define **error vectors**:

$$\boldsymbol{\delta}^{(2)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \in \mathbb{R}^K, \qquad \boldsymbol{\delta}^{(1)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(1)}} \in \mathbb{R}^M.$$

Then **matrix gradients**:

$$\nabla_{W^{(2)}} \mathcal{L} = \boldsymbol{\delta}^{(2)} \, (\mathbf{a}^{(1)})^\top, \qquad \nabla_{W^{(1)}} \mathcal{L} = \boldsymbol{\delta}^{(1)} \, \mathbf{x}^\top.$$

Backprop relation:

$$\delta^{(1)} = \left((W^{(2)})^\top \boldsymbol{\delta}^{(2)} \right) \odot h'(\mathbf{z}^{(1)}),$$

where $\odot$ = **elementwise** product.

Mini-batch and averaging

For a **mini-batch** of size N with examples $(\mathbf{x}^{(n)}, \mathbf{t}^{(n)})$ compute per-example deltas and **average**:

$$\nabla_{W^{(2)}} \mathcal{L}_{\text{batch}} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\delta}^{(2),(n)} (\mathbf{a}^{(1),(n)})^{\top}.$$

Same for $W^{(1)}$. In practice use **vectorized** matrix operations: stack activations by columns/rows and compute **matrix products** to get all gradients in one call.

Common activation derivatives

- **Sigmoid** $\sigma(u) = \frac{1}{1+e^{-u}}$: $\sigma'(u) = \sigma(u)(1-\sigma(u))$.
- **Tanh**: $h(u) = \tanh(u)$: $h'(u) = 1 - \tanh^2(u)$.
- **ReLU**: $h(u) = \max(0, u)$: $h'(u) = \begin{cases} 1 & u > 0 \\ 0 & u < 0 \end{cases}$ (undefined at 0, set 0 or 1 in practice).
- **Leaky ReLU**: $h'(u) = 1$ if $u > 0$, else α.

Note: compute derivatives using **pre-activation values** z (numerically stable).

Softmax+Cross-Entropy: derivation (brief)

Softmax $y_k = \frac{e^{s_k}}{\sum_\ell e^{s_\ell}}$, **CE loss**: $\mathcal{L} = -\sum_k t_k \log y_k$. Compute $\frac{\partial \mathcal{L}}{\partial s_j}$:

$$\frac{\partial \mathcal{L}}{\partial s_j} = \sum_k -t_k \frac{1}{y_k} \frac{\partial y_k}{\partial s_j}.$$

Using $\frac{\partial y_k}{\partial s_j} = y_k(\delta_{kj} - y_j)$ leads to:

$$\frac{\partial \mathcal{L}}{\partial s_j} = y_j - t_j.$$

Thus **output delta:** $\boldsymbol{\delta}^{(2)} = \mathbf{y} - \mathbf{t}$ (key simplification!)

Add L2 regularization (weight decay)

If the training objective is:

$$\mathcal{J}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \frac{\lambda}{2} \sum_{\ell} \| W^{(\ell)} \|_F^2,$$

then gradients get an **extra term**:

$$\nabla_{W^{(\ell)}} \mathcal{J} = \nabla_{W^{(\ell)}} \mathcal{L} + \lambda W^{(\ell)}.$$

Biases are not regularized.

Backpropagation algorithm (single minibatch) — pseudocode

1. **Forward pass**: compute $\mathbf{z}^{(1)}$, $\mathbf{a}^{(1)}$, $\mathbf{z}^{(2)}$, $\mathbf{y}$.
2. **Output error**: set $\boldsymbol{\delta}^{(2)} = \partial \mathcal{L} / \partial \mathbf{z}^{(2)}$ (e.g. $\mathbf{y} - \mathbf{t}$).
3. Compute **output weight** gradient:

$$G^{(2)} = \frac{1}{N} \sum_n \boldsymbol{\delta}^{(2),(n)} (\mathbf{a}^{(1),(n)})^\top$$

4. **Backpropagate** to hidden:

$$\boldsymbol{\delta}^{(1),(n)} = ((W^{(2)})^\top \boldsymbol{\delta}^{(2),(n)}) \odot h'(\mathbf{z}^{(1),(n)})$$

5. Compute **input weight** gradient:

$$G^{(1)} = \frac{1}{N} \sum_n \boldsymbol{\delta}^{(1),(n)} (\mathbf{x}^{(n)})^\top$$

6. If **L2 regularization**:

$$G^{(\ell)} \leftarrow G^{(\ell)} + \lambda W^{(\ell)}$$

7. **Update weights** (SGD, Adam, etc.)

Worked scalar example (1 hidden neuron)

Network: $z^{(1)} = w^{(1)}x + b^{(1)}$, $a^{(1)} = h(z^{(1)})$, $s = z^{(2)} = w^{(2)}a^{(1)} + b^{(2)}$, $y = \sigma(s)$, loss $\mathcal{L} = \frac{1}{2}(y - t)^2$.

$$\delta^{(2)} = (y - t)\sigma'(s)$$

$$\frac{\partial \mathcal{L}}{\partial w^{(2)}} = \delta^{(2)} a^{(1)}, \qquad \frac{\partial \mathcal{L}}{\partial b^{(2)}} = \delta^{(2)}$$

$$\delta^{(1)} = h'(z^{(1)})\, w^{(2)}\, \delta^{(2)}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \delta^{(1)} x, \qquad \frac{\partial \mathcal{L}}{\partial b^{(1)}} = \delta^{(1)}.$$

Forward Propagation Through L Hidden Layers

Input Layer

$$a^{(0)} = \mathbf{x}$$

Hidden Layers:

$$z^{(l)} = W^{(l)}\, a^{(l-1)} + b^{(l)}, \qquad l = 1, 2, \ldots, L$$

$$a^{(l)} = \sigma\left(z^{(l)}\right)$$

Output Layer:

$$z^{(L+1)} = W^{(L+1)}\, a^{(L)} + b^{(L+1)}$$

$$\hat{y} = \sigma\left(z^{(L+1)}\right)$$

Loss:

$$\mathcal{L} = \mathcal{L}(\hat{y}, y)$$

General Backpropagation in Deep Networks

Given: A network with L hidden layers

$$a^{(0)} = x, \quad z^{(l)} = W^{(l)}\, a^{(l-1)} + b^{(l)}, \quad a^{(l)} = \sigma(z^{(l)}), \quad l = 1, 2, ..., L$$

Output Layer (layer $L+1$):

$$z^{(L+1)} = W^{(L+1)} a^{(L)} + b^{(L+1)}, \quad \hat{y} = a^{(L+1)}$$

Loss Function:

$$\mathcal{L}(y, \hat{y})$$

Goal: $\dfrac{\partial \mathcal{L}}{\partial W^{(l)}}, \quad \dfrac{\partial \mathcal{L}}{\partial b^{(l)}} \quad \forall\, l = 1, \ldots, L+1$

Backprop Through L Hidden Layers

Step 1: Error at Output Layer

$$\delta^{(L+1)} = \frac{\partial \mathcal{L}}{\partial z^{(L+1)}} = (\hat{y} - y) \odot \sigma'(z^{(L+1)})$$

Step 2: Error Propagation Backwards

$$\delta^{(l)} = \left(W^{(l+1)}\right)^{\top} \delta^{(l+1)} \odot \sigma'(z^{(l)}) \quad (\textit{ChainRule})$$

$$\text{for } l = L, L-1, \ldots, 1$$

Step 3: Gradients for Each Layer

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} \left(a^{(l-1)}\right)^{\top}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)}$$

$$\textit{Update } W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}}, \quad b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(l)}}$$

Practical notes & numerical stability

- For **softmax**: use log-sum-exp trick $\rightarrow s' = s - \max_k s_k$
- Sigmoid: avoid saturation $\rightarrow$ proper **initialization** (Xavier/He)
- ReLU: **subgradient** at 0
- **Use GPUs + batching** $\rightarrow$ matrix ops
- Combine **softmax + CE** as single operation $\rightarrow$ stable and efficient $(y - t)$

Summary (what to take away)

- **Backprop = chain rule**: compute δ, then gradients layer-wise
- **Key identities:**

$$\nabla_{W^{(2)}} = \delta^{(2)}(\mathbf{a}^{(1)})^\top, \quad \delta^{(1)} = ((W^{(2)})^\top \delta^{(2)}) \odot h'(\mathbf{z}^{(1)})$$

- **Softmax + CE**: $\delta^{(2)} = \mathbf{y} - \mathbf{t}$
- **Regularization** adds λW