An Introduction of Artificial Neural Networks

1 Architecture of Artificial Neural Networks

Artificial Neural Networks (ANNs) are composed of interconnected neurons organized in layers. The basic architecture consists of:

- **Input Layer**: Receives the input data.
- **Hidden Layers**: Intermediate layers that perform computations.
- **Output Layer**: Produces the final output.

Each layer is fully connected to the next layer, forming a feedforward network.

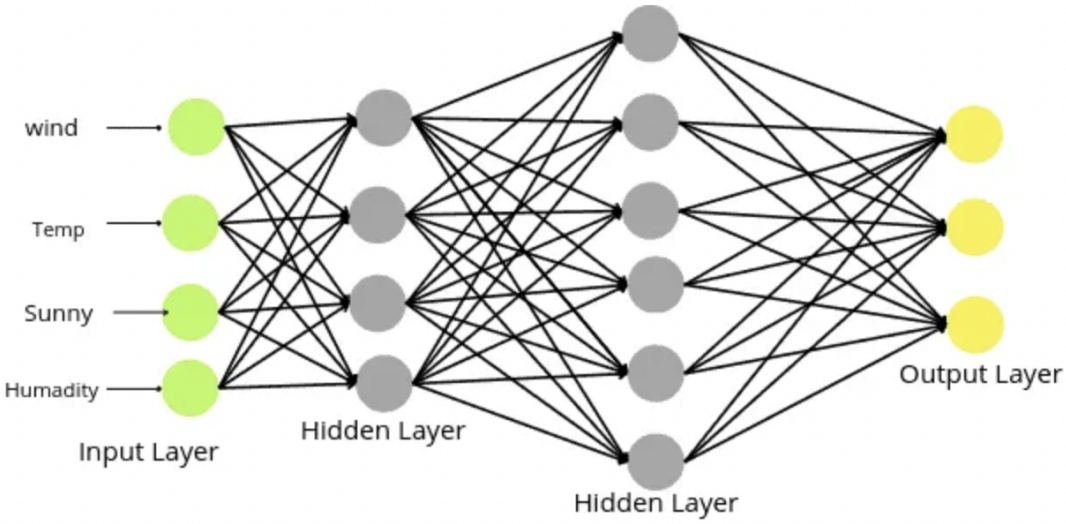


Figure 1: Architecture of ANN with 2 hidden layers.
Figure credit: https://hands-on.cloud/introduction-to-artificial-neural-networks/

Remark 1.1.
- In an ANN, each layer is fully connected.
- Number of hidden layers should be no smaller more than 1.
- Arrows stands for data transfers. The transfer between every two layers is described by a mapping.

2 Mappings in ANNs

ANNs perform mappings from input to output through a series of function compositions:

Input layer $\xrightarrow{g_1}$ Hidden layer 1 $\xrightarrow{g_2}$ Hidden layer 2 $\xrightarrow{g_3} \cdots \xrightarrow{g_k}$ Hidden layer k $\xrightarrow{g_{k+1}}$ Output layer,

where each mapping g_i has the form

$$g = f(\boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{b})$$

- $\boldsymbol{w}$ is the weight matrix.
- $\boldsymbol{x}$ is the input vector.
- $\boldsymbol{b}$ is the bias vector.
- $\boldsymbol{f}$ is the activation function.

Remark 2.1.

- Activation functions introduce non-linearity into the network, enabling it to learn complex patterns. Common activation functions include:
  - **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$
  - **ReLU (Rectified Linear Unit)**: $f(x) = \max(0, x)$
  - **Tanh**: $f(x) = \tanh(x)$
- By convention, the last mapping g_{k+1} is not activated, i.e. a linear mapping of the input.

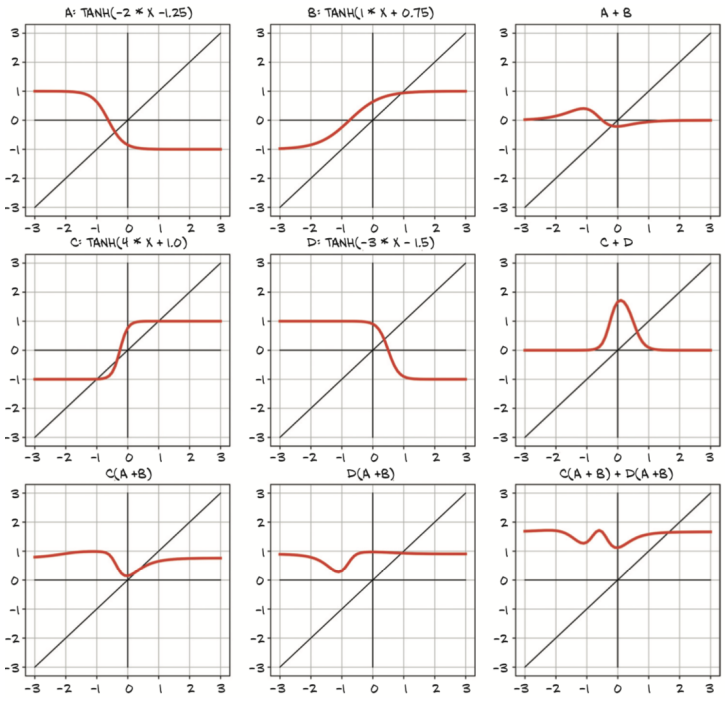


Figure 2: Figure 6.6 in [1]

Theorem 2.2 (Universal approximation: Theorem 3 in [2])**.** *If $f \in C^m(\mathbb{R}^k)$ is nonconstant and bounded, then $\mathcal{R}^k(\psi)$ is uniformly m-dense on compacta in $C^m(\mathbb{R}^k)$ and dense in $C^{m,p}(\mu)$ for all finite measures μ on $\mathbb{R}^k$ with compact support.*

3 Training: a binary classification revisited

3.1 Architecture

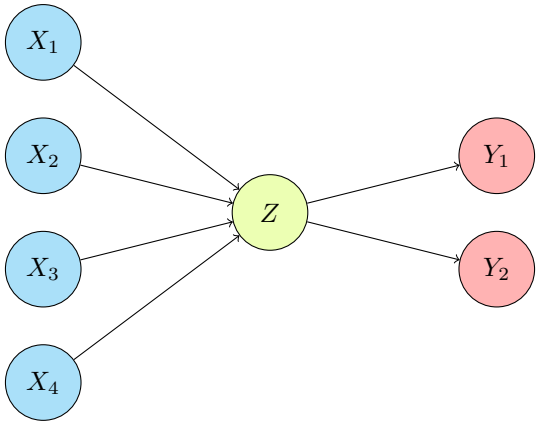


Figure 3: ANN with 4 Inputs, 1 Hidden Neuron, and 2 Outputs, for the a binary classification model

Remark 3.1. For the binary classification example with mammal and non-mammal,

- X_1: Give birth
- X_2: Lay eggs
- X_3: Can fly
- X_4: Have eggs
- Y_1: the probability in Class mammal
- Y_2: the probability in Class non-mammal.

3.2 Mappings

By Section 2, the mapping from the input layer to the hidden layer is

$$Z = f(\boldsymbol{w}_1^\top X_1 + b_1),$$

where $\boldsymbol{w}_1 = (w_{11}, w_{12}, w_{13}, w_{14})'$.

The mapping from the hidden layer to the output layer is

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix} = Z\boldsymbol{w}_2 + \boldsymbol{b}_2 = \begin{pmatrix} Zw_{21} \\ Zw_{22} \end{pmatrix} + \begin{pmatrix} b_{21} \\ b_{22} \end{pmatrix}.$$

We get a mapping from the input $\boldsymbol{X}$ to the output $\boldsymbol{Y}$, denoted by $\boldsymbol{Y} = \boldsymbol{Y}(\boldsymbol{X}; \boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2)$.

3.3 Input, output and loss function

Assume there are m samples. The input of ANN is $(\boldsymbol{X}^i, \boldsymbol{Y}^i)_{i=1,\cdots,m}$, where

$$\boldsymbol{X}^i = \begin{pmatrix} X_1^i \\ X_2^i \\ X_3^i \\ X_4^i \end{pmatrix}, \qquad \boldsymbol{Y}^i = \begin{pmatrix} Y_1^i \\ Y_2^i \end{pmatrix}.$$

Here $\boldsymbol{X}^i$ is the vector of features and $\boldsymbol{Y}^i$ is the corresponding vector of label: If the animal is mammal, $\boldsymbol{Y}^i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, otherwise $\boldsymbol{Y}^i = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

The output of ANN is $\boldsymbol{Y}(\boldsymbol{X}^i; \boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2)$ for $i = 1, \cdots, m$.

Goal: Compare prediction $\boldsymbol{Y}(\boldsymbol{X}^i; \boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2)$ and observation $\boldsymbol{Y}^i$, $i = 1, \cdots, m$, by minimizing the empirical risk function or the *total* loss function:

$$L(\boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2) = \frac{1}{m} \sum_{j=1}^{m} V\Big(\boldsymbol{Y}^j, \boldsymbol{Y}(\boldsymbol{X}^j; \boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2)\Big),$$

where the function $V(\cdot, \cdot)$ is a loss function measuring the "distance" between $\boldsymbol{Y}^j$ and $\boldsymbol{Y}(\boldsymbol{X}^j; \boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2)$.

Here are some popular loss functions. Let $\boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$ and $\boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}$.

- least-squares (ℓ_2) loss

$$V(\alpha, \beta) = (\alpha_1 - \beta_1)^2 + (\alpha_2 - \beta_2)^2 := \|\alpha - \beta\|_2^2.$$

- cross entropy

$$V(\alpha, \beta) = -\alpha_1 \log \beta_1 - \alpha_2 \log \beta_2.$$

- least absolute (ℓ_1) value

$$V(\alpha, \beta) = |\alpha_1 - \beta_1| + |\alpha_2 - \beta_2| := \|\alpha - \beta\|_1.$$

3.4 Training process

We minimize $L(w_1, b_1, w_2, b_2)$ to get the optimal parameters

$$(\boldsymbol{w}_1^*, b_1^*, \boldsymbol{w}_2^*, \boldsymbol{b}_2^*) = \arg \min_{\boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2} L(\boldsymbol{w}_1, b_1, \boldsymbol{w}_2, \boldsymbol{b}_2),$$

which is a complicated optimization problem, due to

- composition of several (or many) NNs.
- Nonconvexity.

A minimization problem is solved by gradient descent method:

$$p_{n+1} = p_n - \gamma_n \cdot \nabla L(p_n),$$

where

- γ_n is called *stride* or *learning rate*; either fixed or adaptive.
- $p_n := (\boldsymbol{w}_1^n, b_1^n, \boldsymbol{w}_2^n, \boldsymbol{b}_2^n)'$ is the vector of parameters to be updated, at step n
- The algorithm to find the gradient is called backpropagation.

4 Computation graph and backpropagation

Backpropagation is the process (or algorithm) of updating the weights by calculating the gradient $\nabla L(p)$, or simply $\frac{\partial L}{\partial p}$. The gradient measures how fast the cost changes when we change the weights and biases.

• Formulate a (composite) function as a graph

• **In the forward pass**, the function is evaluated.

• **In the backward pass**, the gradient (or derivative) is updated.

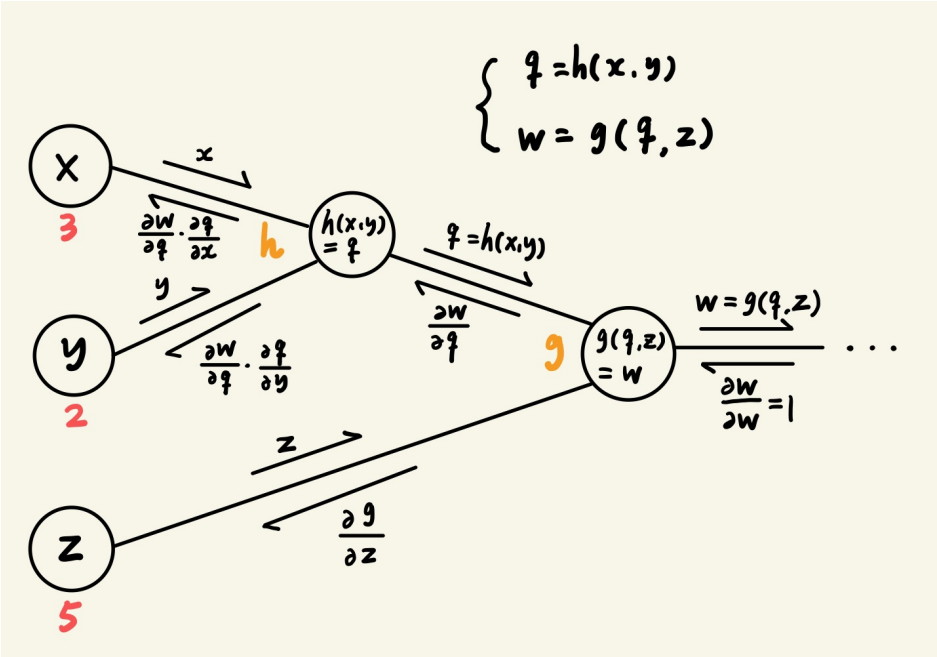


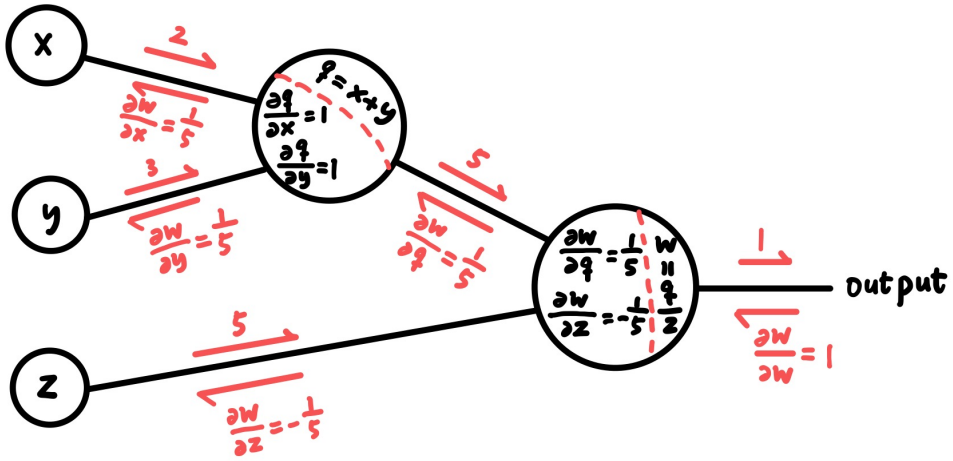
Figure: computation graph of $f(x, y, z) = g(h(x, y), z)$

• From the computation graph(backpropagation), gradient of f is

$$Df = \left(\frac{\partial w}{\partial x}, \frac{\partial w}{\partial y}, \frac{\partial w}{\partial z}\right)^\top.$$

Example 4.1 (Computation graph of $w = \frac{x+y}{z}$)**.**

- Define the operation "+" and its derivatives: $x, y \mapsto x + y$, $\frac{\partial "+"}{\partial x} : x, y \mapsto 1$, $\frac{\partial "+"}{\partial y} : x, y \mapsto 1$.
- Define the operation "/" and its derivatives: $q, z \mapsto \frac{q}{z}$, $\frac{\partial "/"}{\partial q} : q, z \mapsto \frac{1}{z}$, $\frac{\partial "/"}{\partial z} : q, z \mapsto -\frac{q}{z^2}$.
- Let $q = x + y$ and $w = \frac{q}{z}$.
- q can be evalued when the value of x, y are passed in. They are stored for future use; forward: pass q to the next layer, backward: compute derivative of the former layer.



References

[1] Stevens, E., Antiga, L. and Viehmann, T. (2020). Deep learning with PyTorch.

[2] Author, B. (1991). Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4, 251-257.