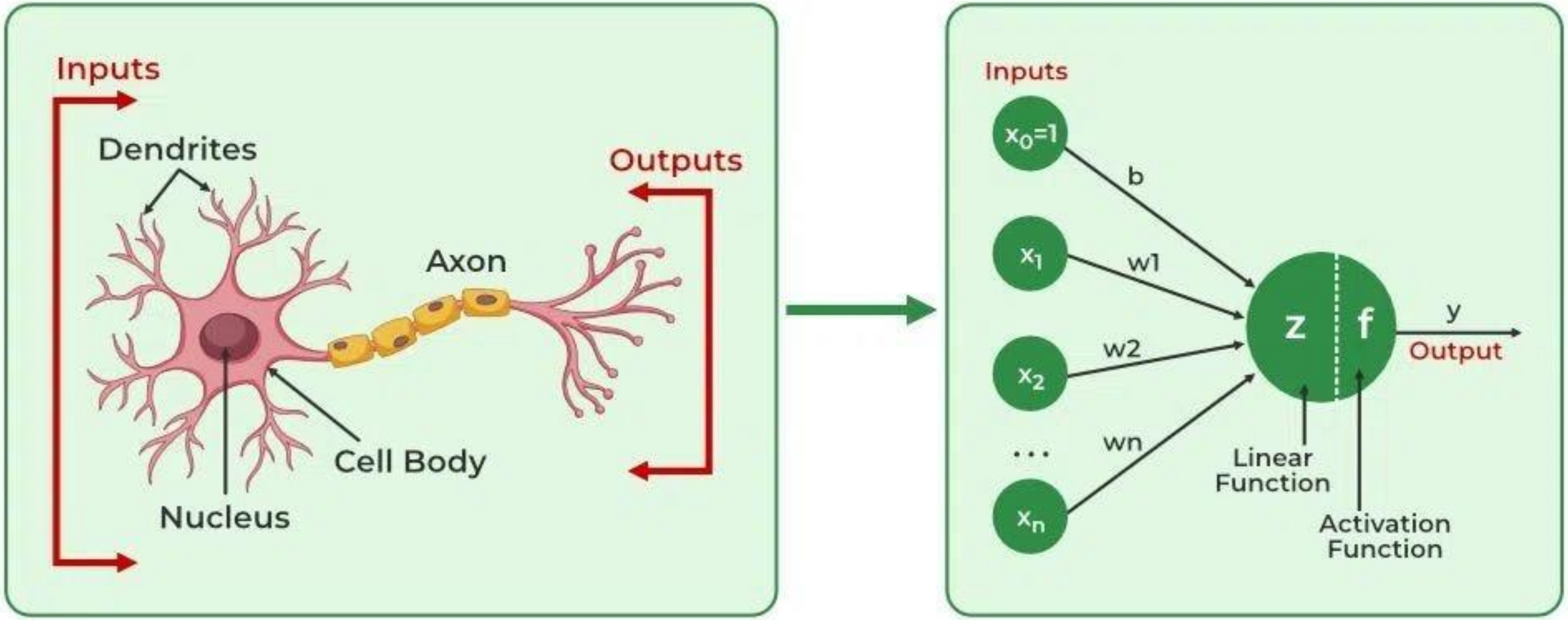
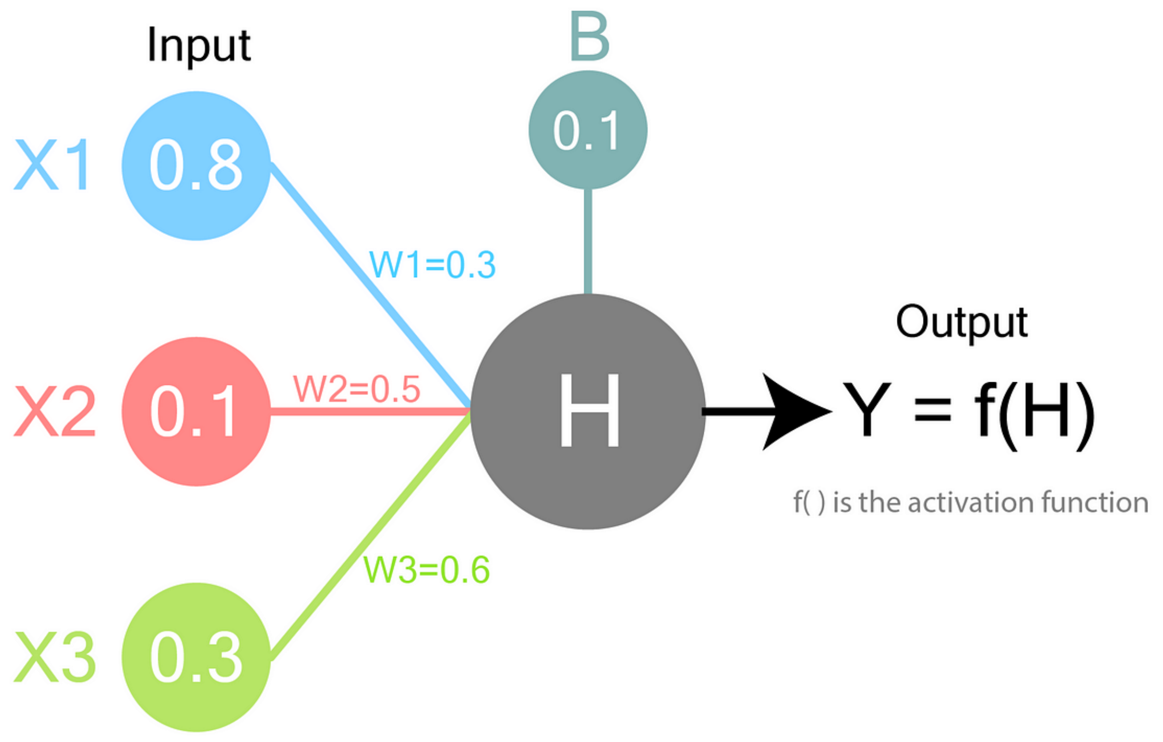
Deep learning

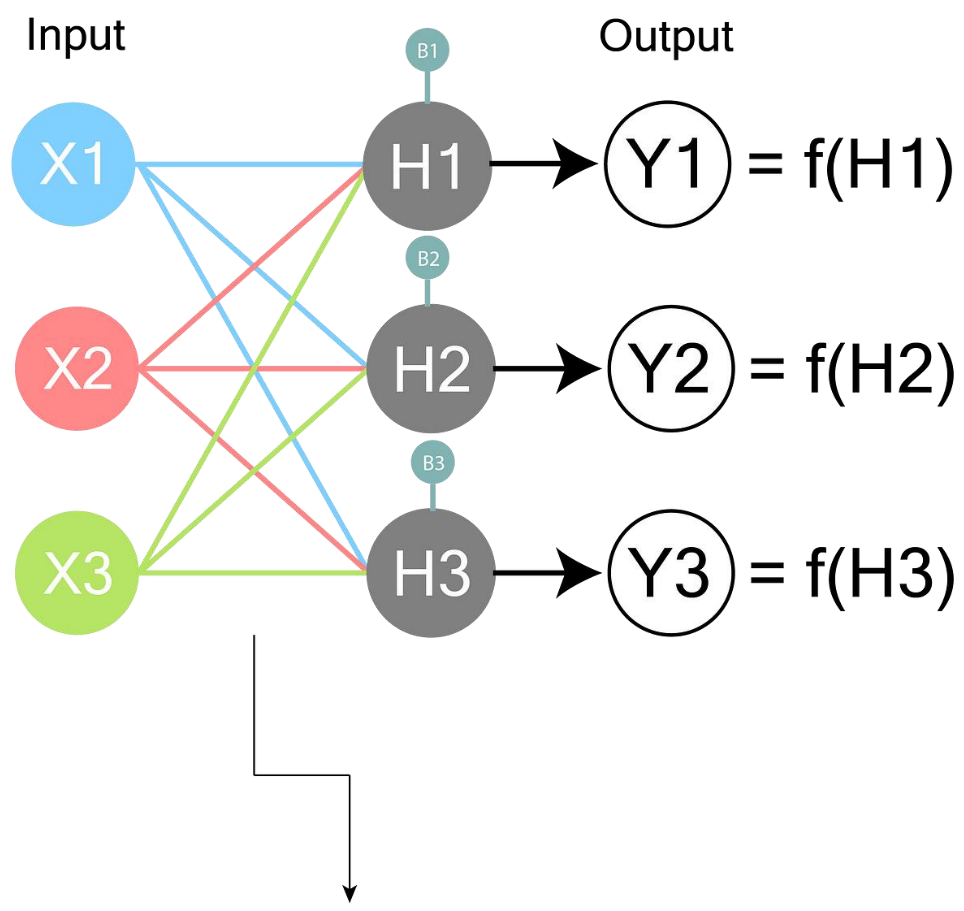

Inputs
Dendrites
Outputs
Axon
Cell Body
Nucleus
Inputs
x_0=1
b
x_1
w1
x_2
w2
...
wn
x_n
z
f
y
Output
Linear Function
Activation Function

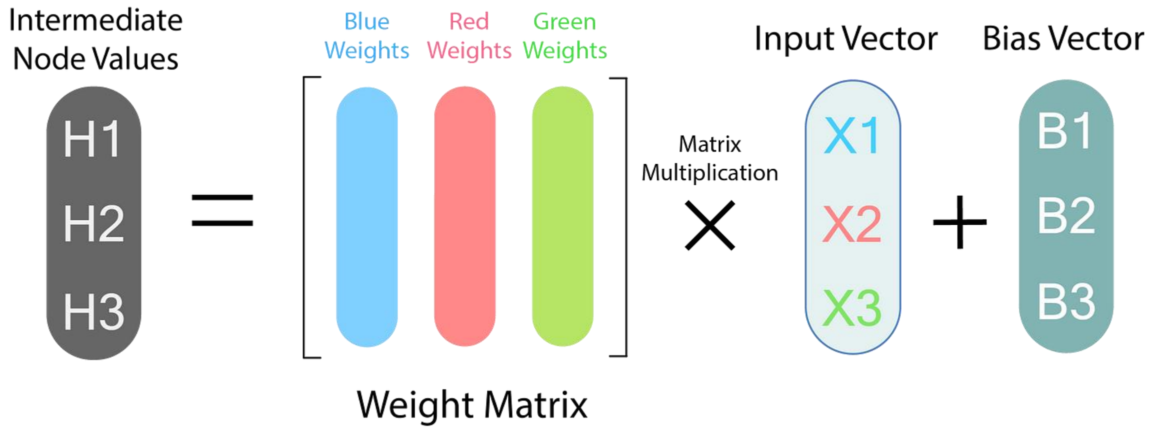
Single Neuron Functionality

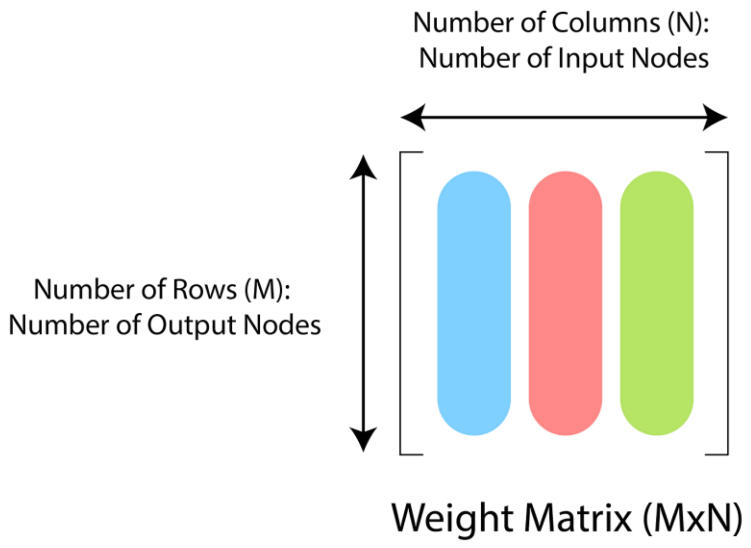


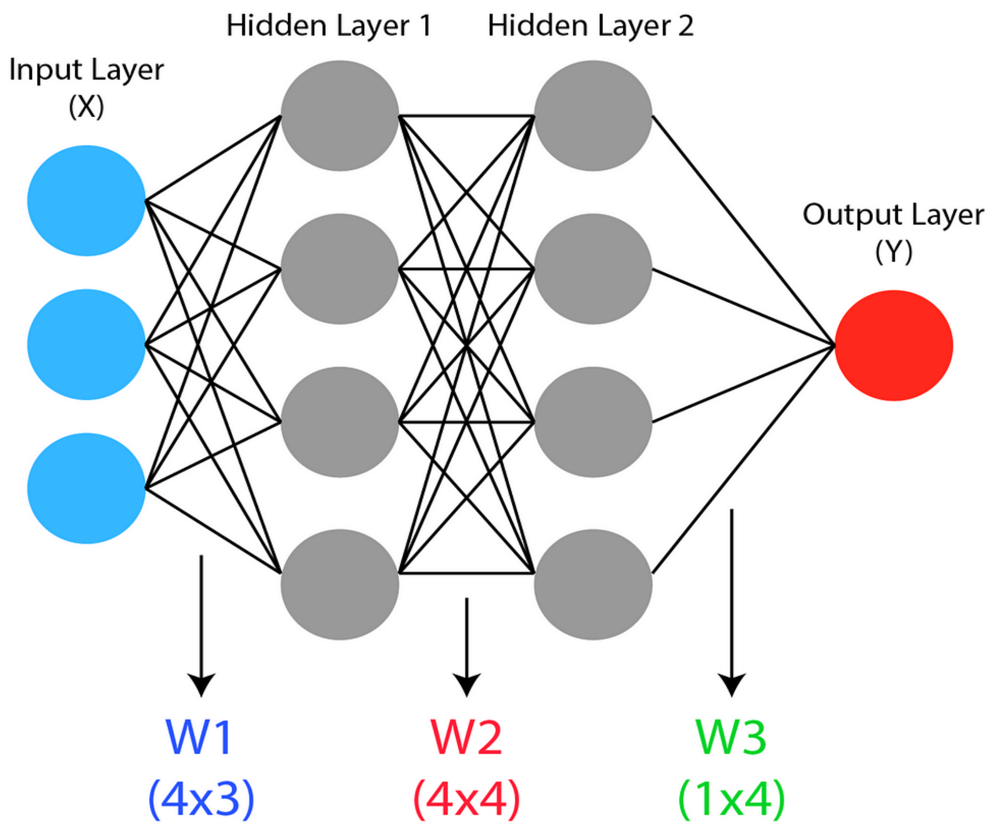
$$H = X1*W1 + X2*W2 + X3*W2 + B$$

$$Y = f(H)$$

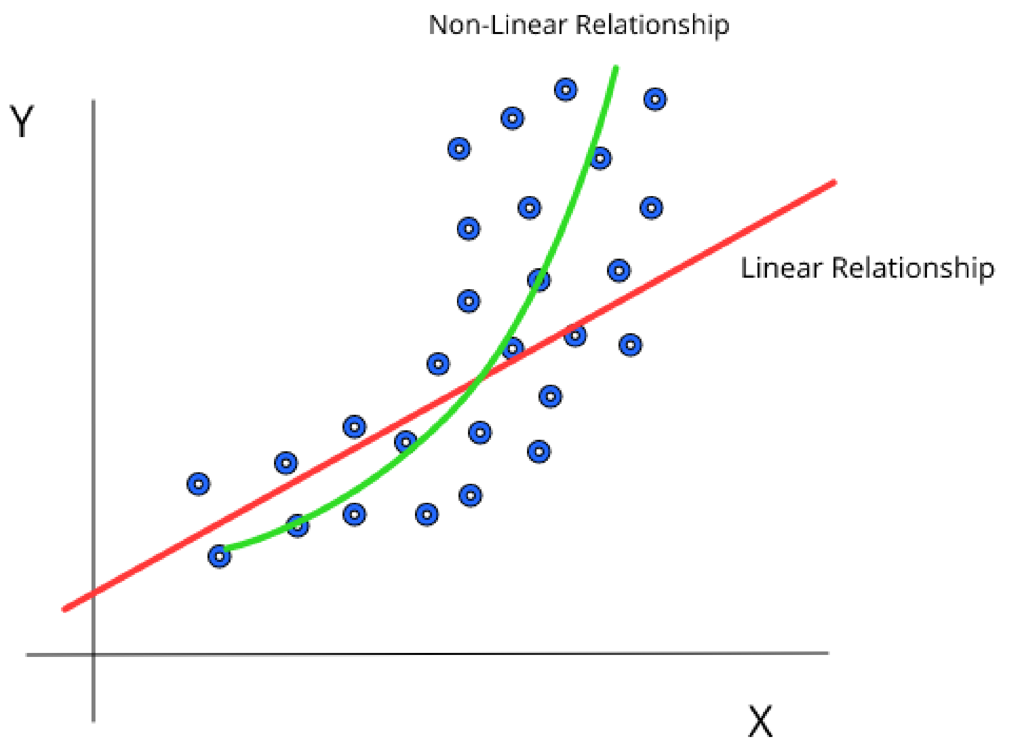

Input
Output
X1
X2
X3
B1
B2
B3
H1
H2
H3
Y1 = f(H1)
Y2 = f(H2)
Y3 = f(H3)


Intermediate Node Values
H1
H2
H3
=
Blue Weights
Red Weights
Green Weights
Weight Matrix
Matrix Multiplication
×
Input Vector
X1
X2
X3
+
Bias Vector
B1
B2
B3


Number of Columns (N): Number of Input Nodes
Number of Rows (M): Number of Output Nodes
Weight Matrix (MxN)



$$Y = \overbrace{f(W3 * \underbrace{f(W2 * \overbrace{f(W1 * \underbrace{X}_{\text{Input Layer}} + B1)}^{\text{Hidden Layer 1}} + B2)}_{\text{Hidden Layer 2}} + B3)}^{\text{Output layer (Y)}}$$



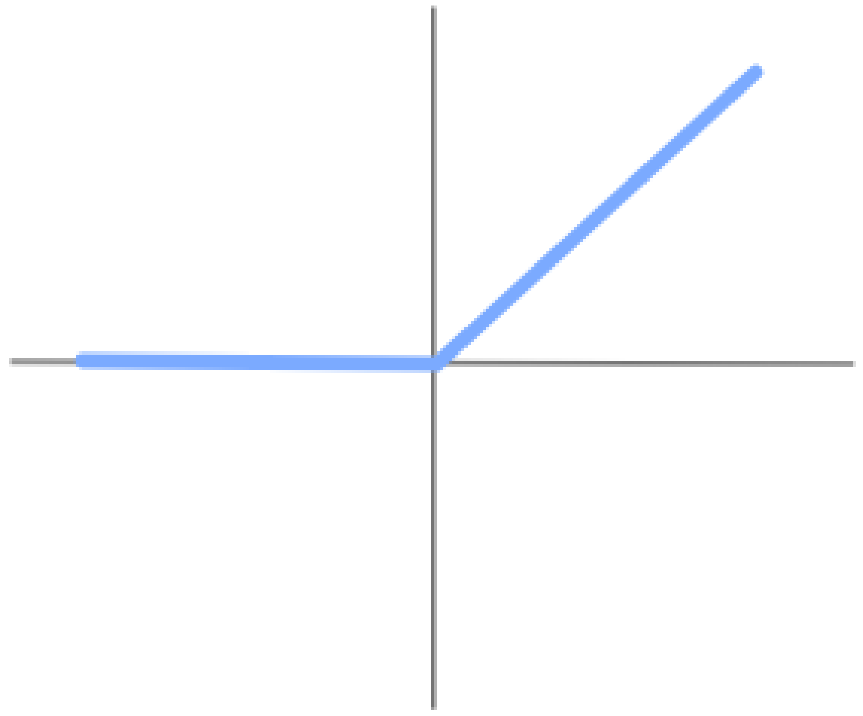
Without Activation Function

$$y = \sum_{i=0}^{n} (W_i * X_i) + B$$

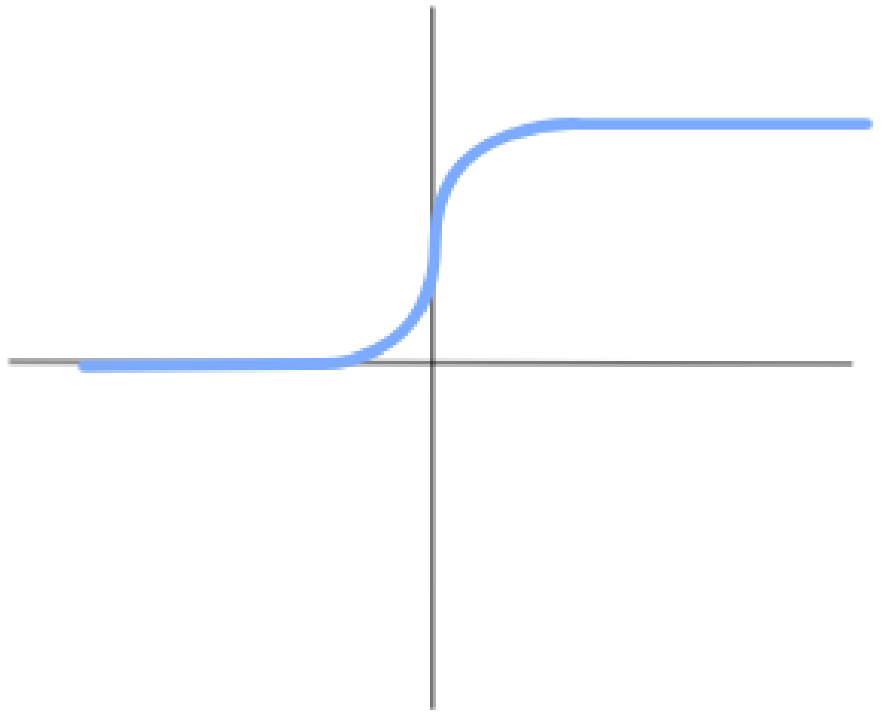
With Activation Function

$$y = f(\sum_{i=0}^{n} (W_i * X_i) + B)$$

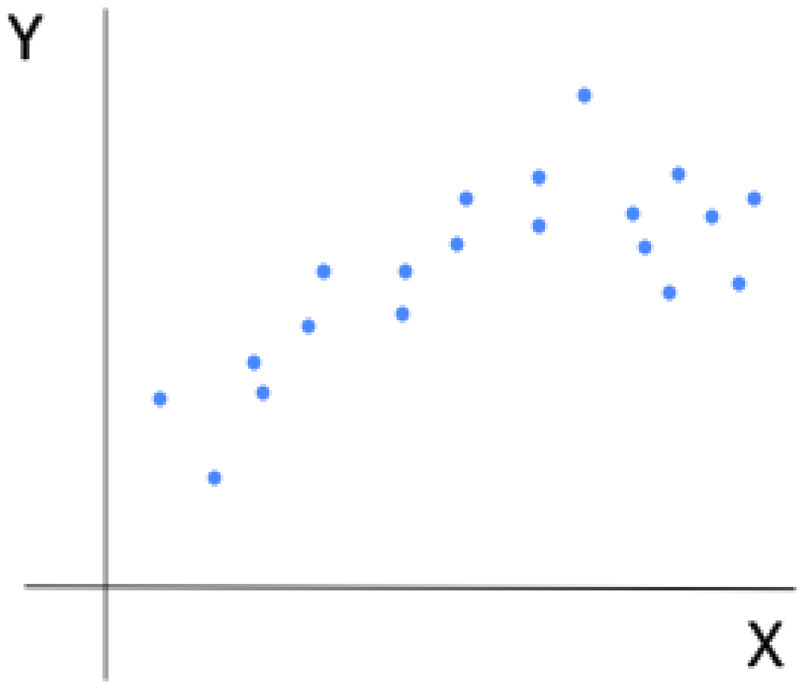
ReLU



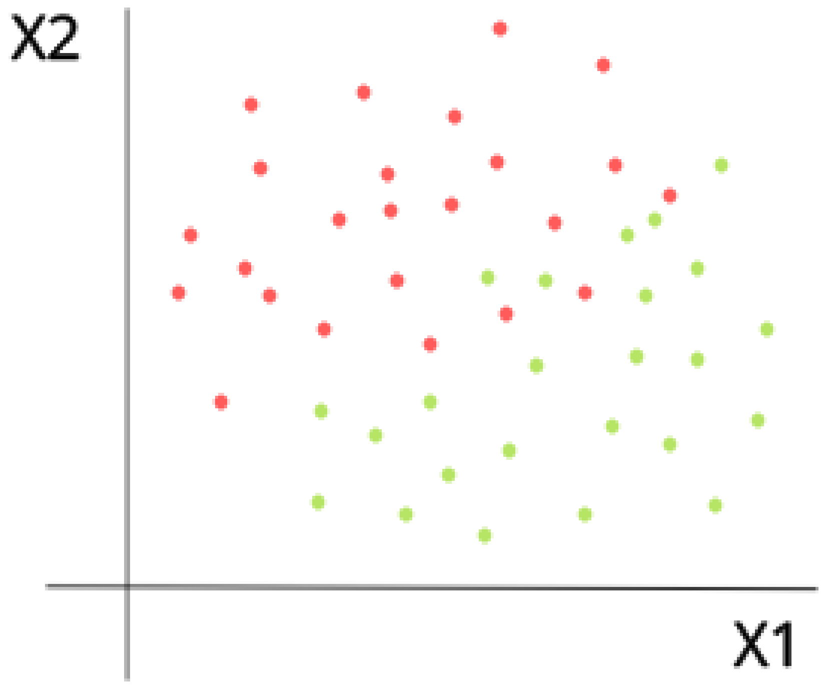
Sigmoid

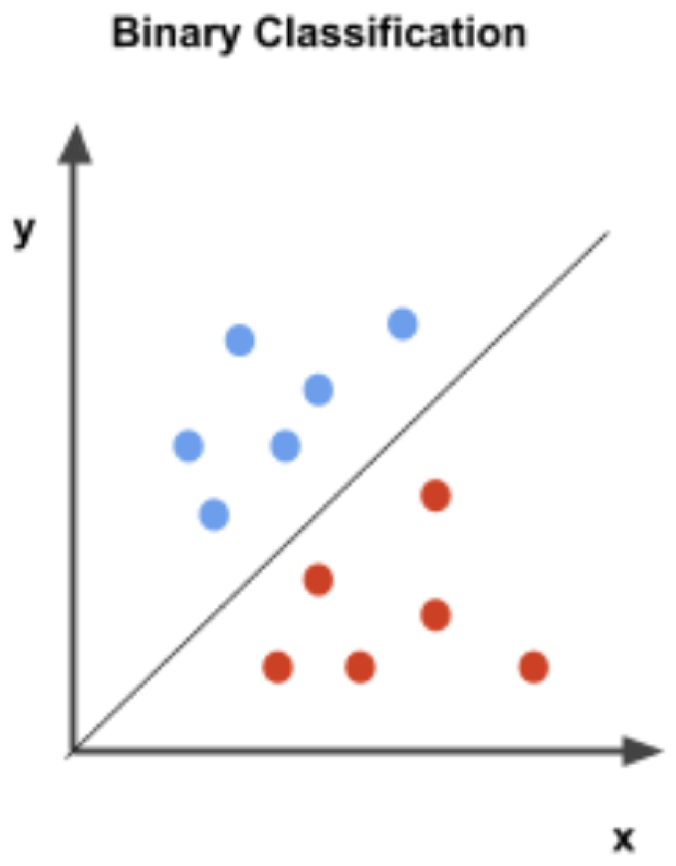
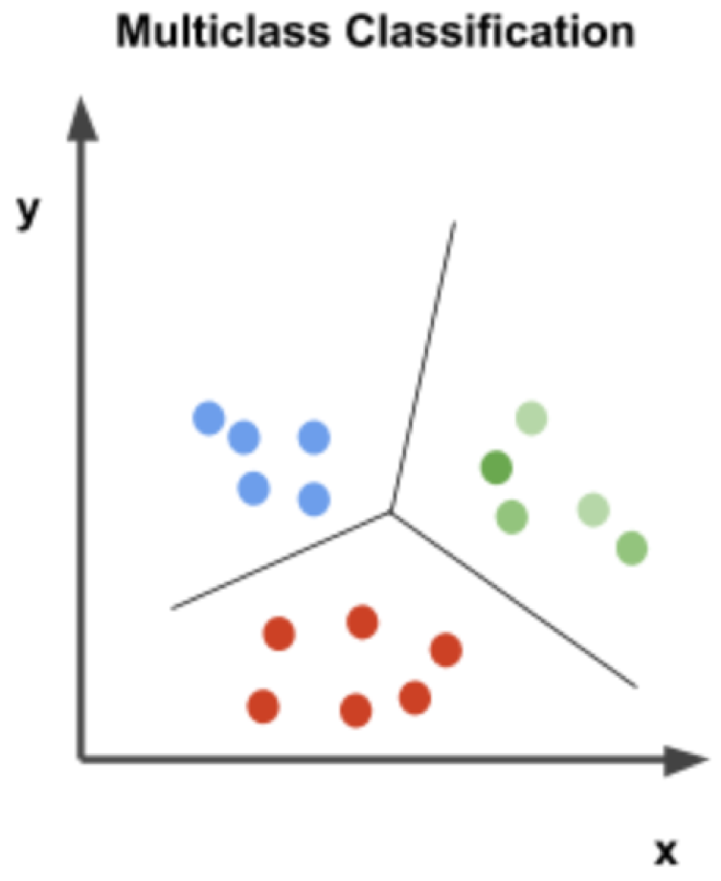


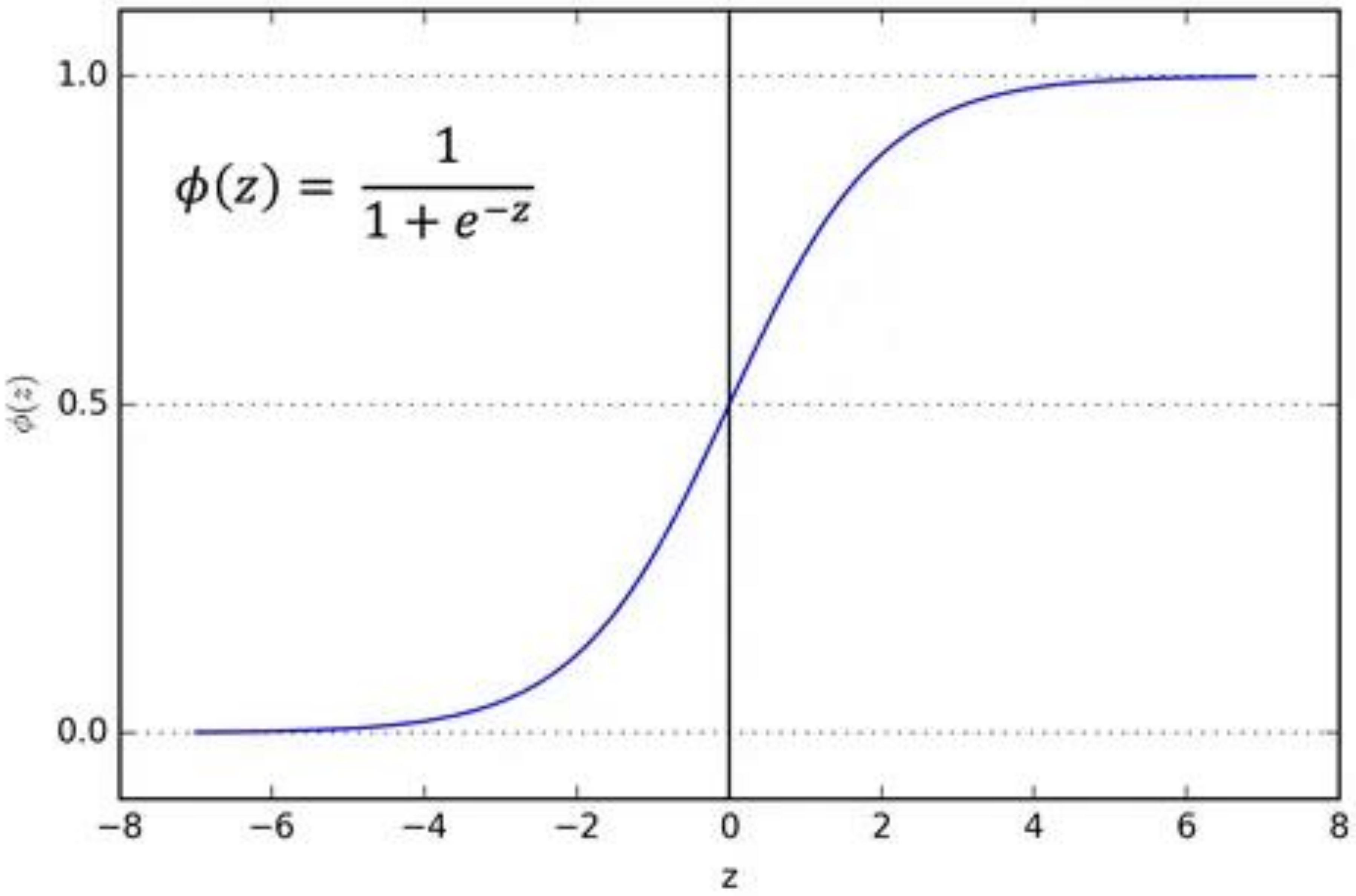
Regression



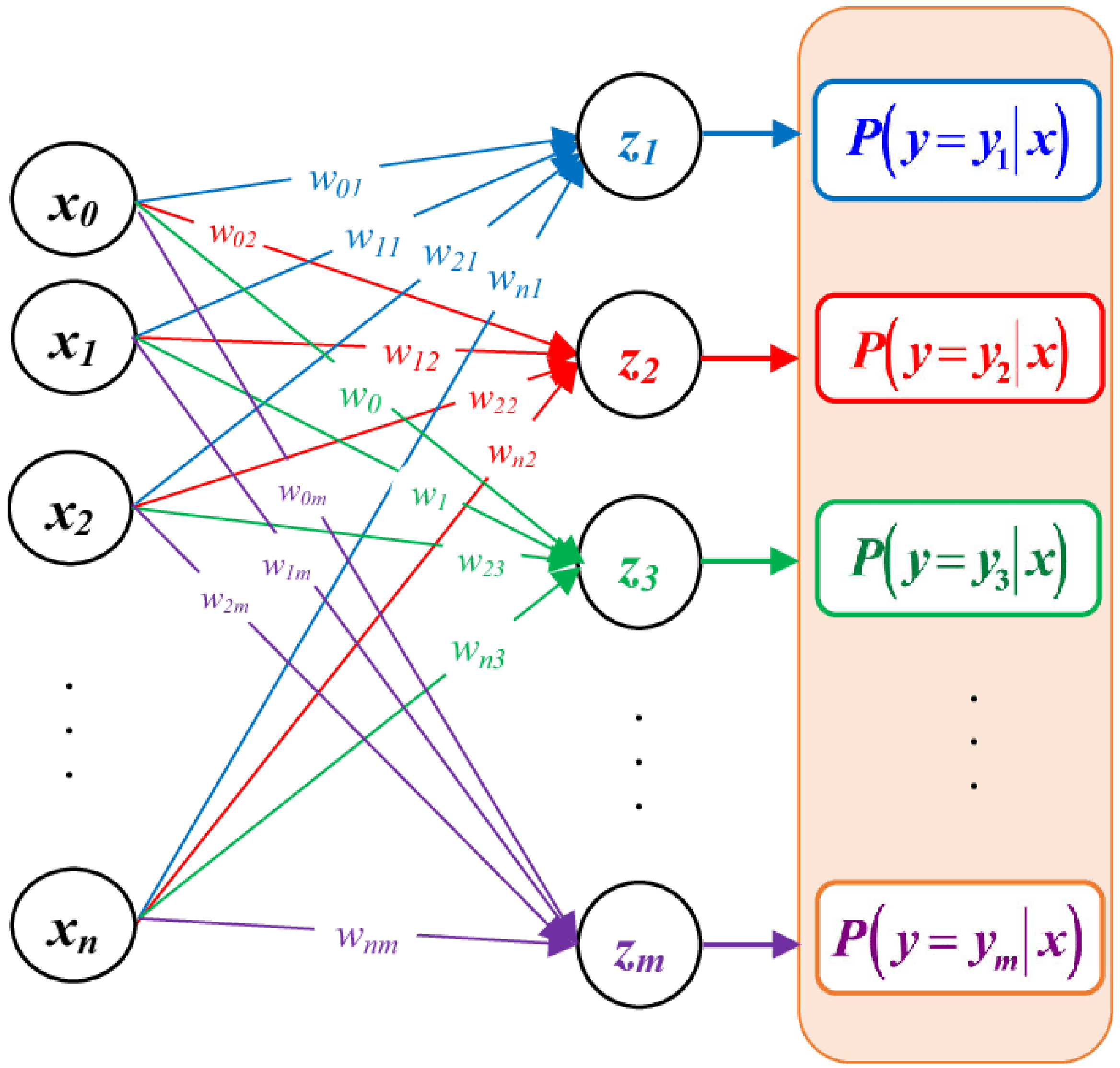
Classification




Binary Classification
y
x

Multiclass Classification
y
x

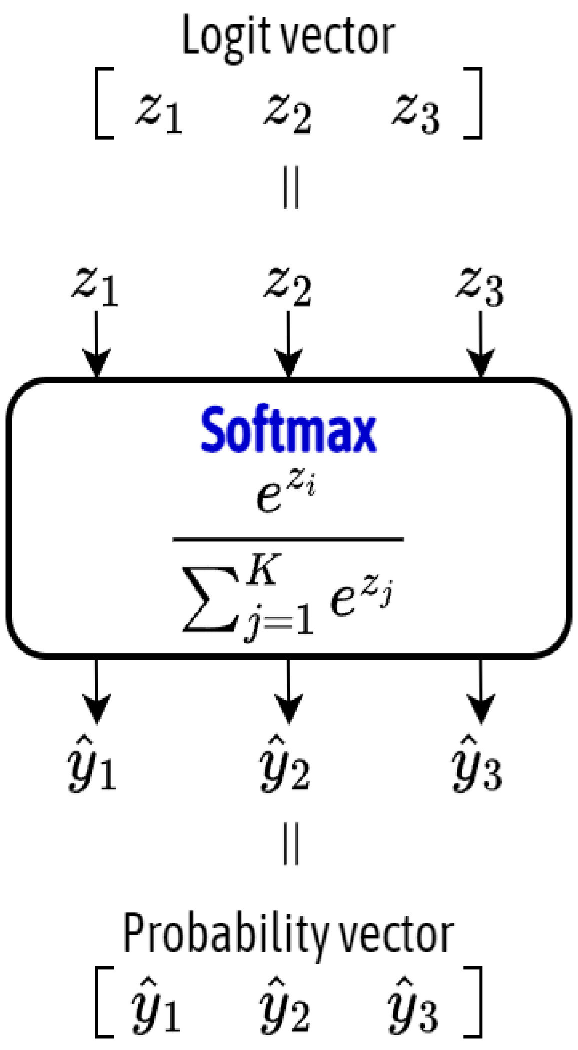


$\phi(z) = \frac{1}{1+e^{-z}}$
1.0
0.5
0.0
$\phi(z)$
−8
−6
−4
−2
0
2
4
6
8
z

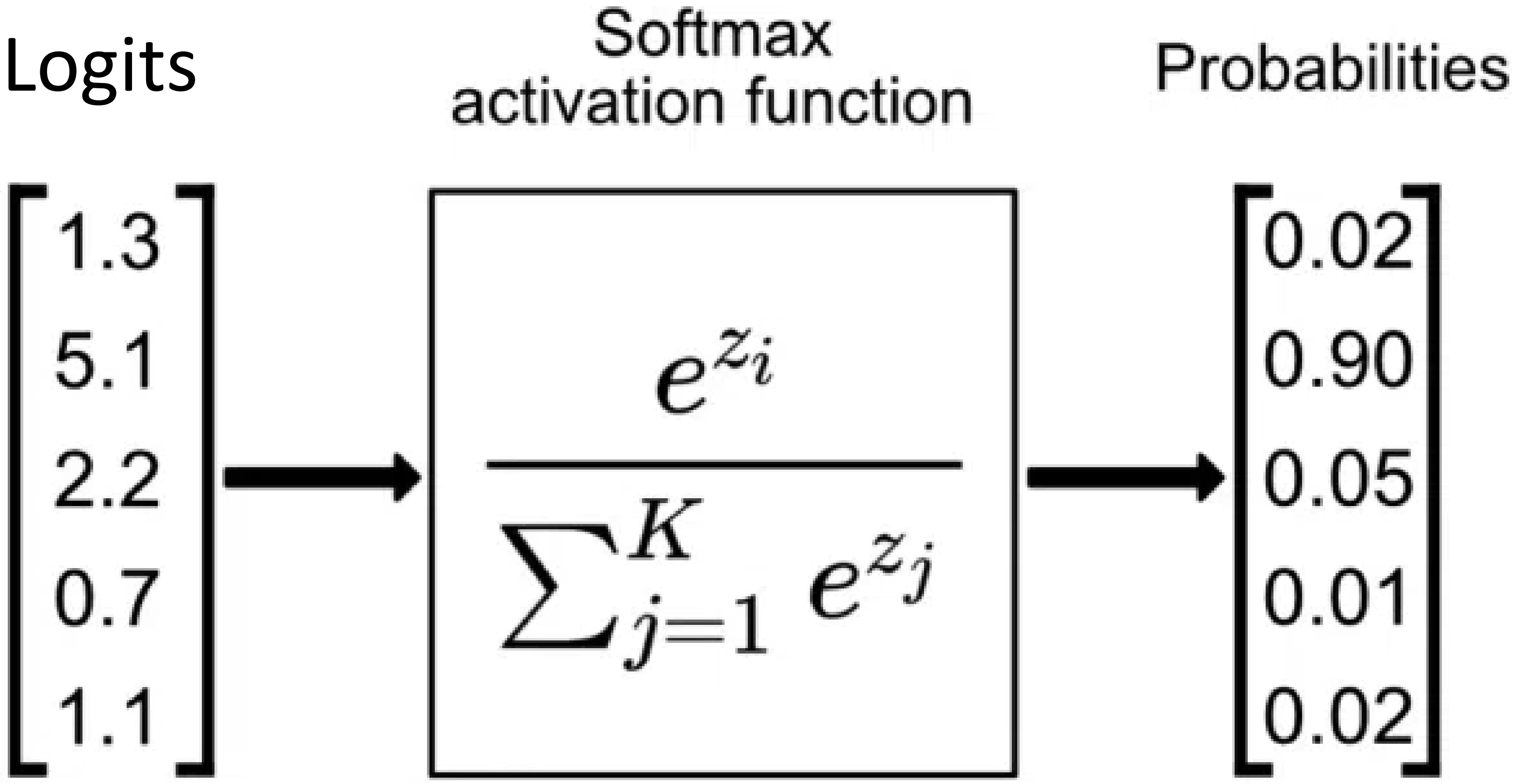


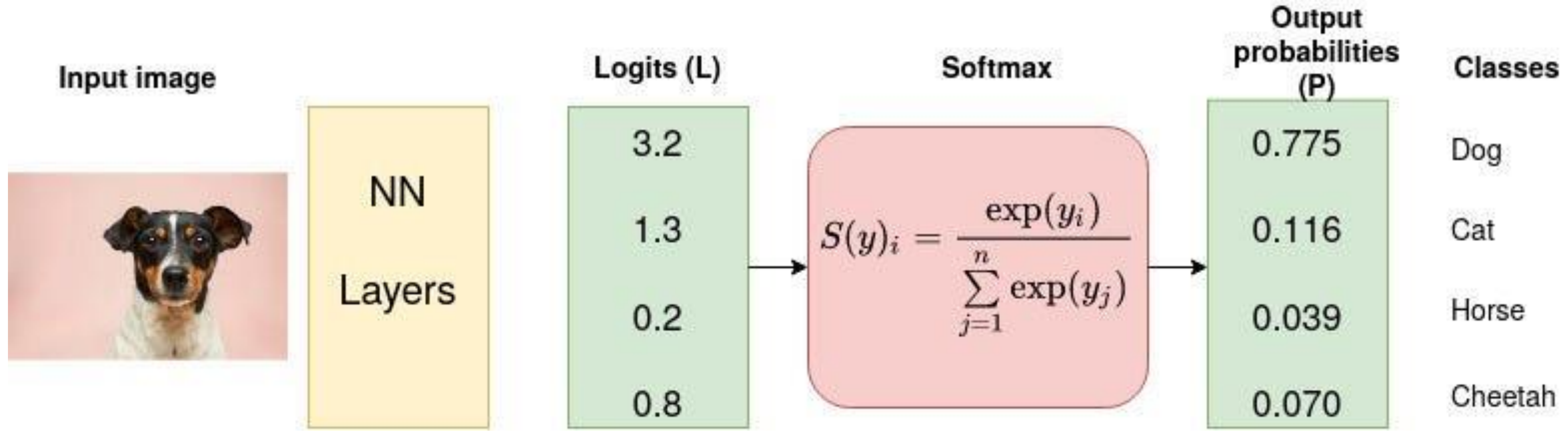
- *In [deep learning](), the term logits layer is popularly used for the last neuron layer of neural network for classification task which produces raw prediction values as real numbers ranging from [-infinity, +infinity].*

- *Logits are the raw scores output by the last layer of a neural network. Before activation takes place.*


Logit vector
$\begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}$
$\|$
z_1
z_2
z_3
Softmax
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\|$
Probability vector
$\begin{bmatrix} \hat{y}_1 & \hat{y}_2 & \hat{y}_3 \end{bmatrix}$

Logits


Softmax
activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02



Softmax *is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.*

It is an activation function that scales logits into probabilities. The output of a Softmax is a vector (say v) with probabilities of each possible outcome. The probabilities in vector v sums to one for all possible outcomes or classes.

$$\exp(3.2) = 24.5325$$
$$\exp(1.3) = 3.6693$$
$$\exp(0.2) = 1.2214$$
$$\exp(0.8) = 2.2255$$

and therefore,

$$\begin{aligned} S(3.2) &= \frac{\exp(3.2)}{\exp(3.2) + \exp(1.3) + \exp(0.2) + \exp(0.8)} \\ &= \frac{24.5325}{24.5325 + 3.6693 + 1.2214 + 2.2255} \\ &= 0.775 \end{aligned}$$

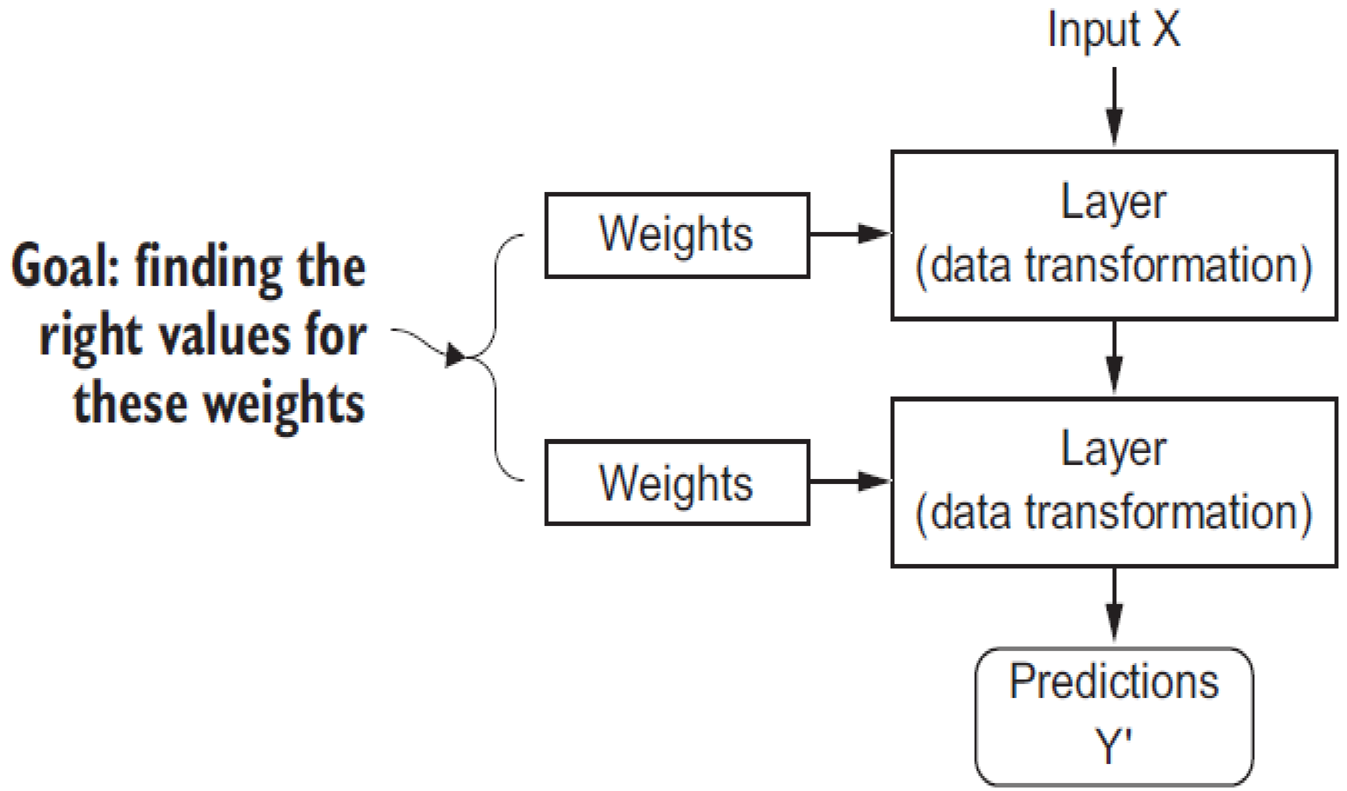


Figure 1.7 A neural network is parameterized by its weights.

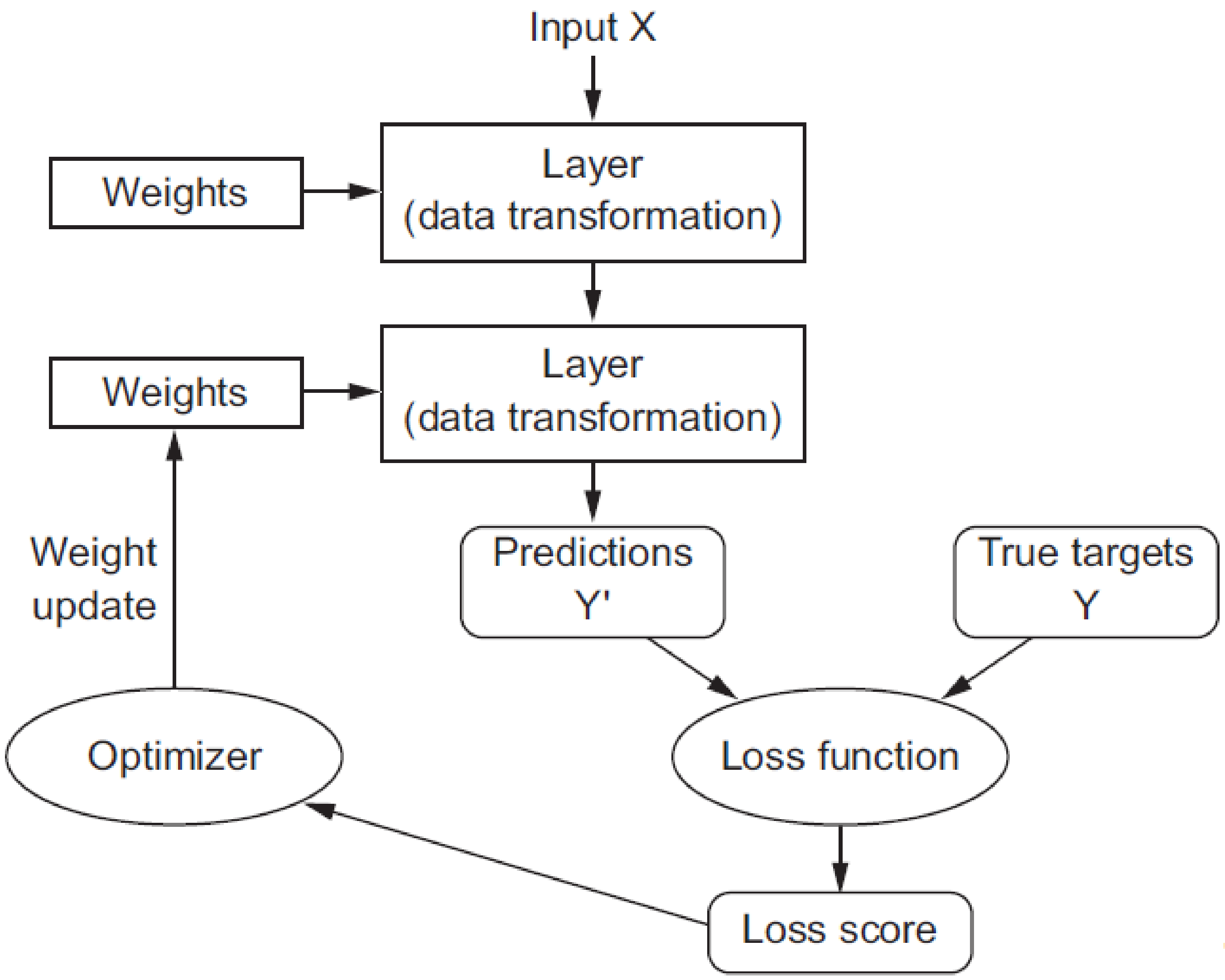
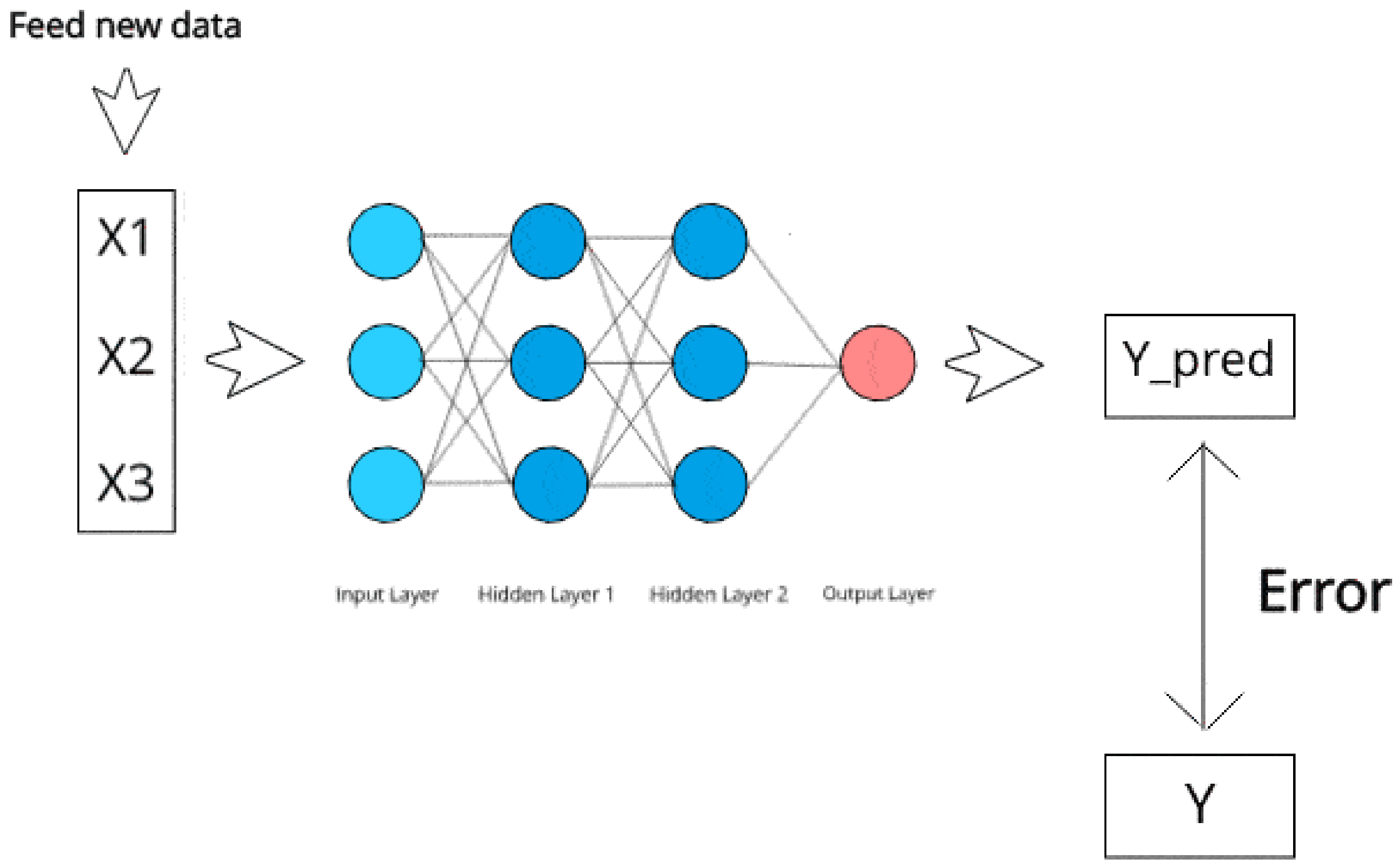


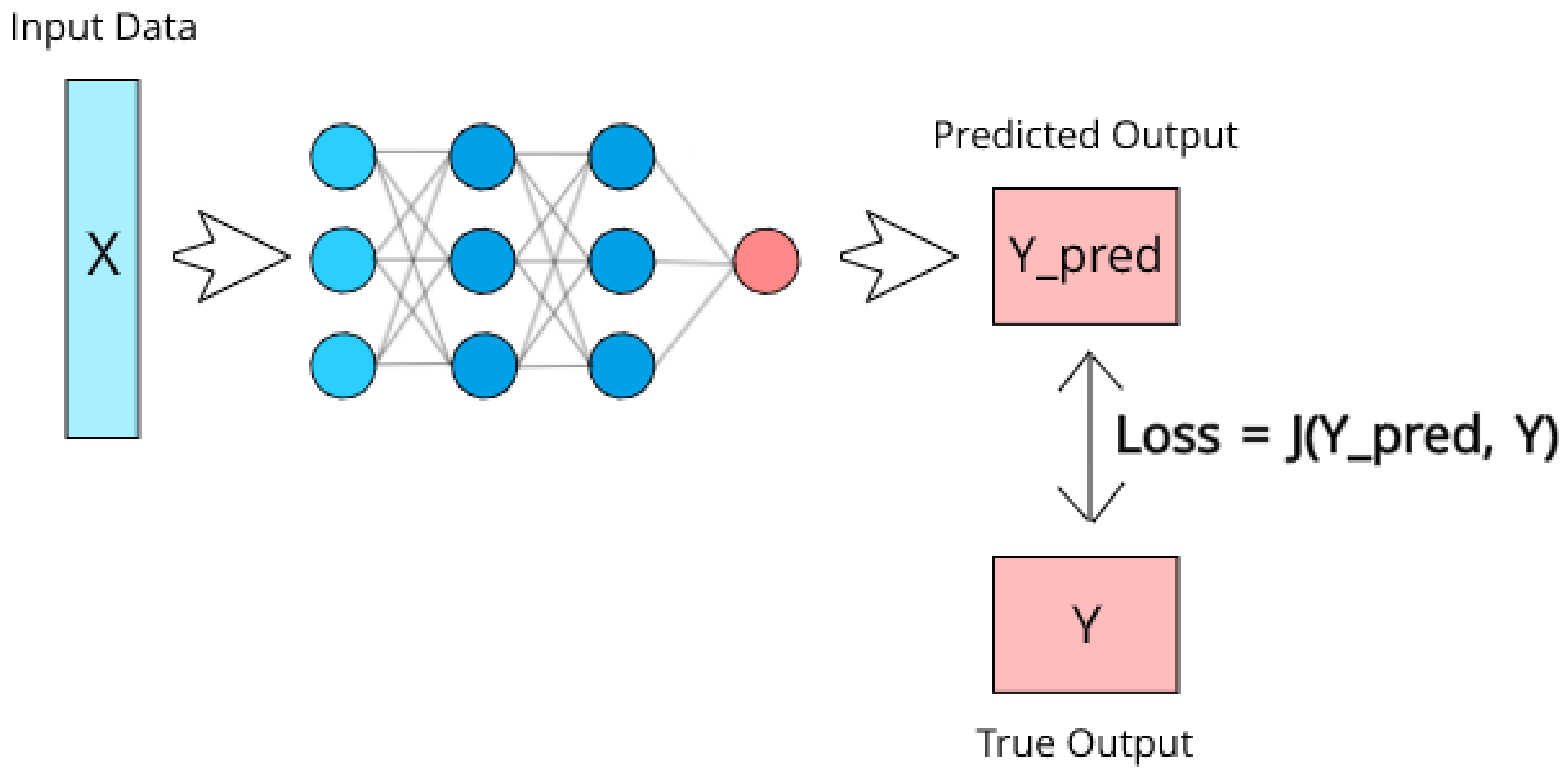
Figure 1.9 The loss score is used as a feedback signal to adjust the weights.



- Loss function will essentially calculate how **poorly/Good,** our model is performing by comparing what the model is predicting with the actual value (ground truth) it is supposed to output.

- If Y_pred is very far off from Y, the Loss value will be **very high**. However if both values are almost similar, the Loss value will be **very low.**

The objective is almost always to minimize the loss function. The lower the loss the better the model.



Regression Loss

In regression, our model is trying to predict a **continuous value.** Some examples of regression models are:

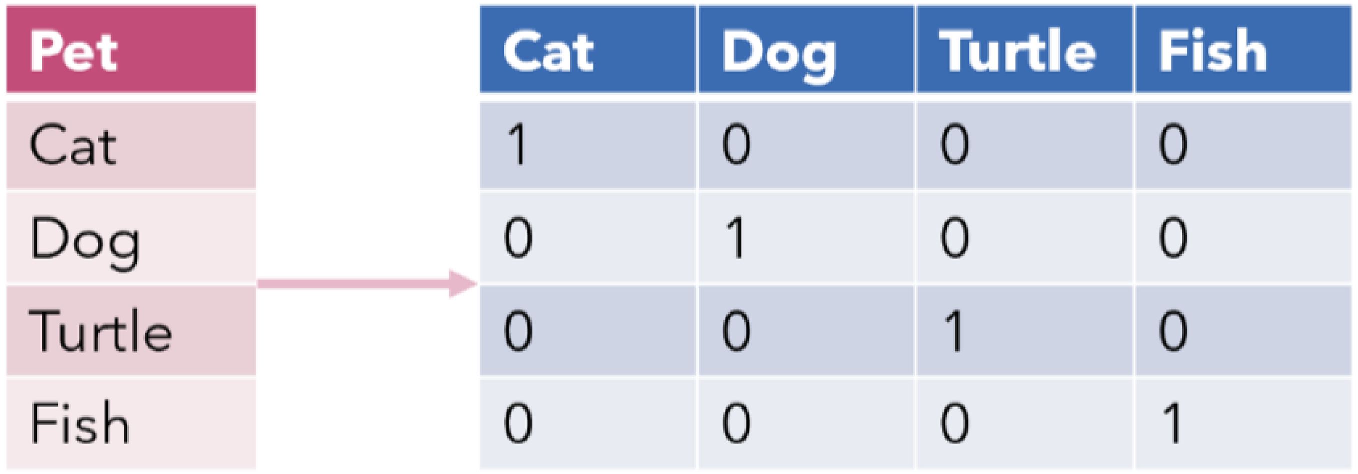
- House price prediction
- Person Age prediction

$$Loss = \frac{1}{n} * \sum_{i=0}^{n} (Y_i - Y_{pred_i})^2$$

Mean Squared Error Loss Function

Classification

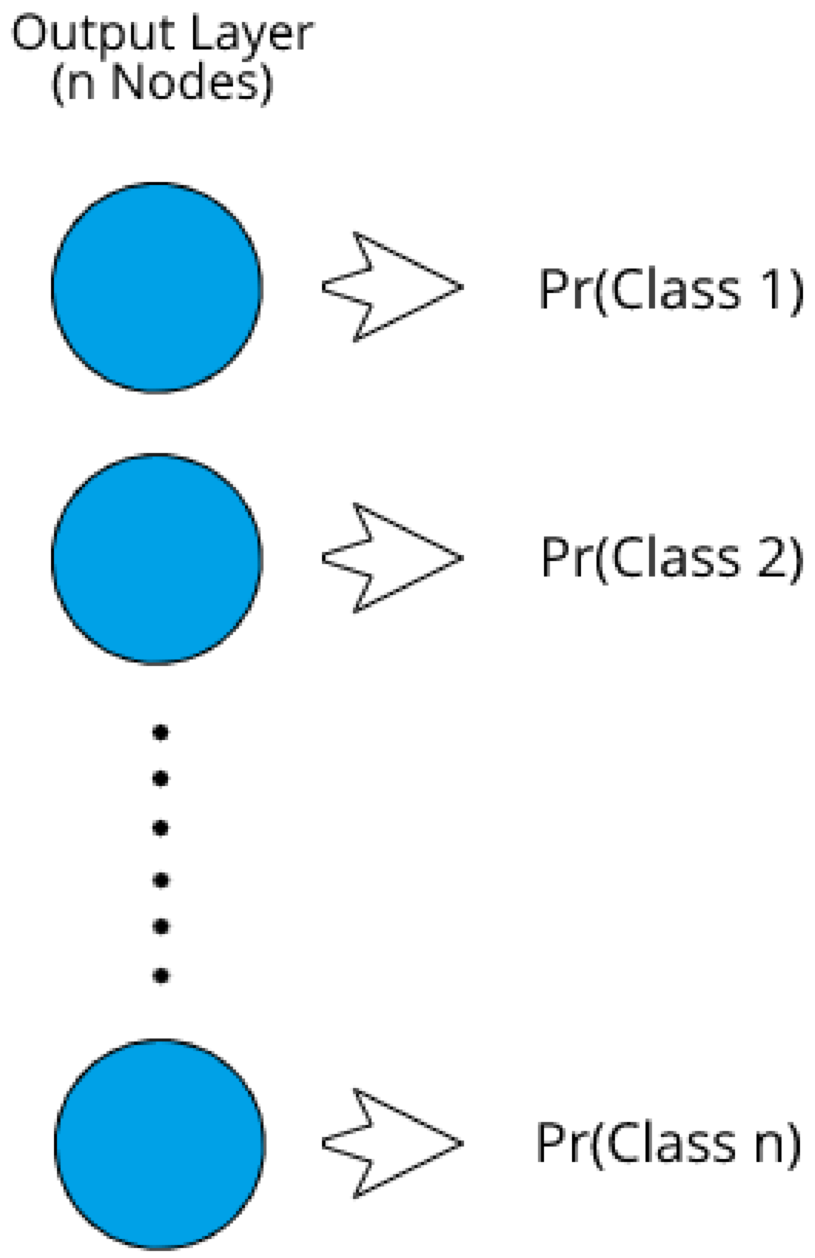
Target data representation: One hot encoding



Pet
Cat
Dog
Turtle
Fish

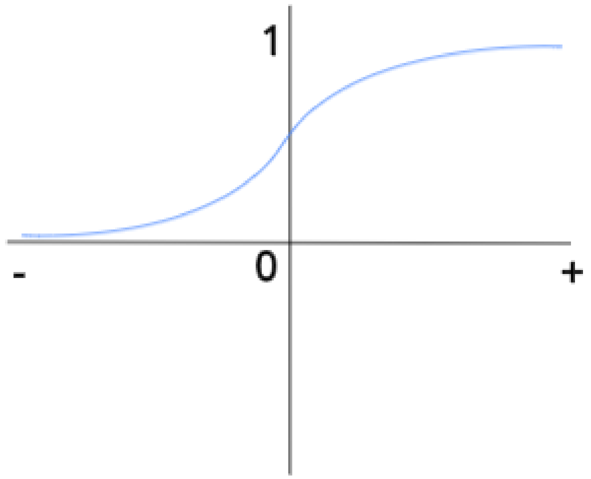
Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

One Hot Encoding is a process in the data processing that is applied to categorical data, to convert it into **a binary vector representation** for use in machine learning algorithms



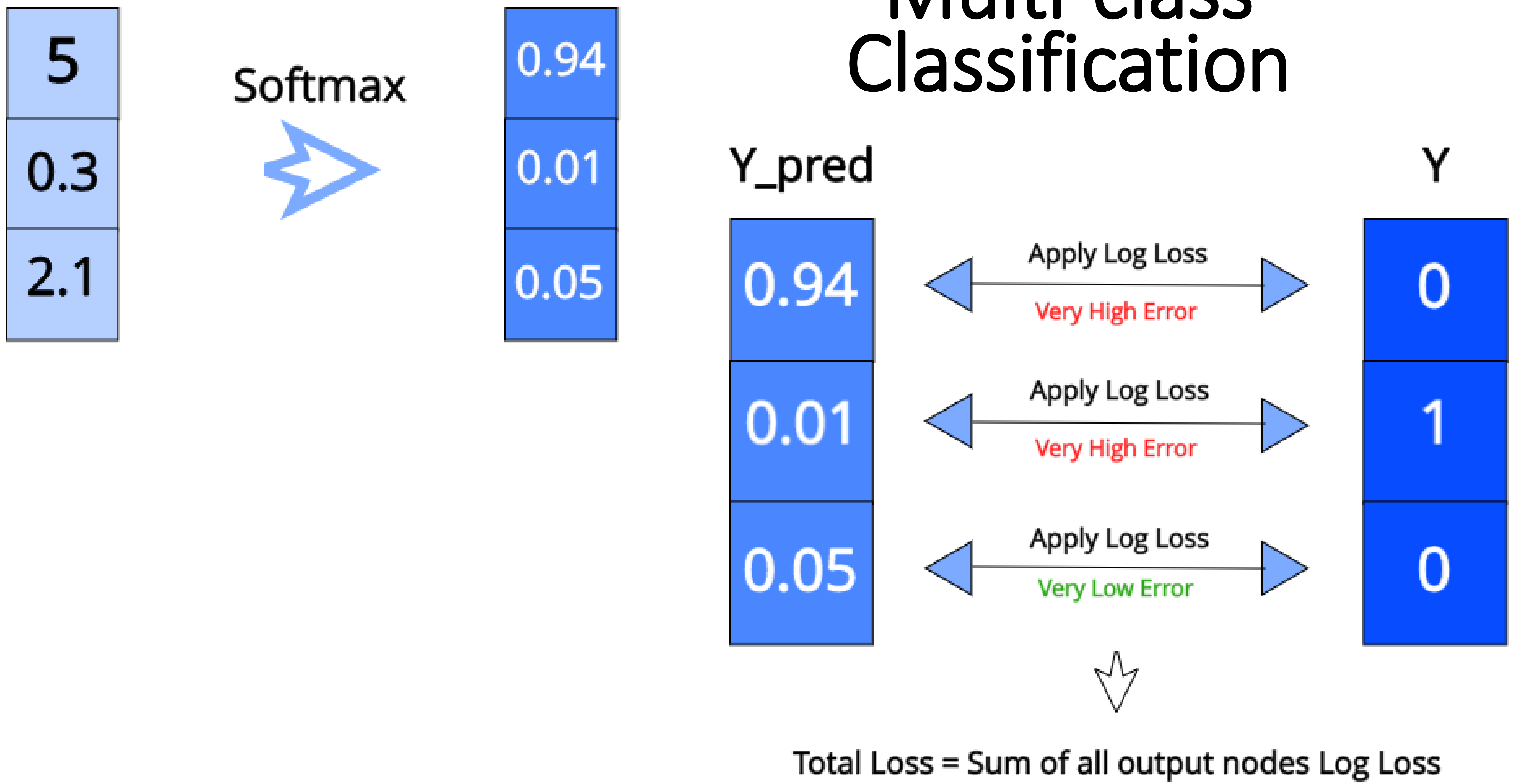
Binary Classification

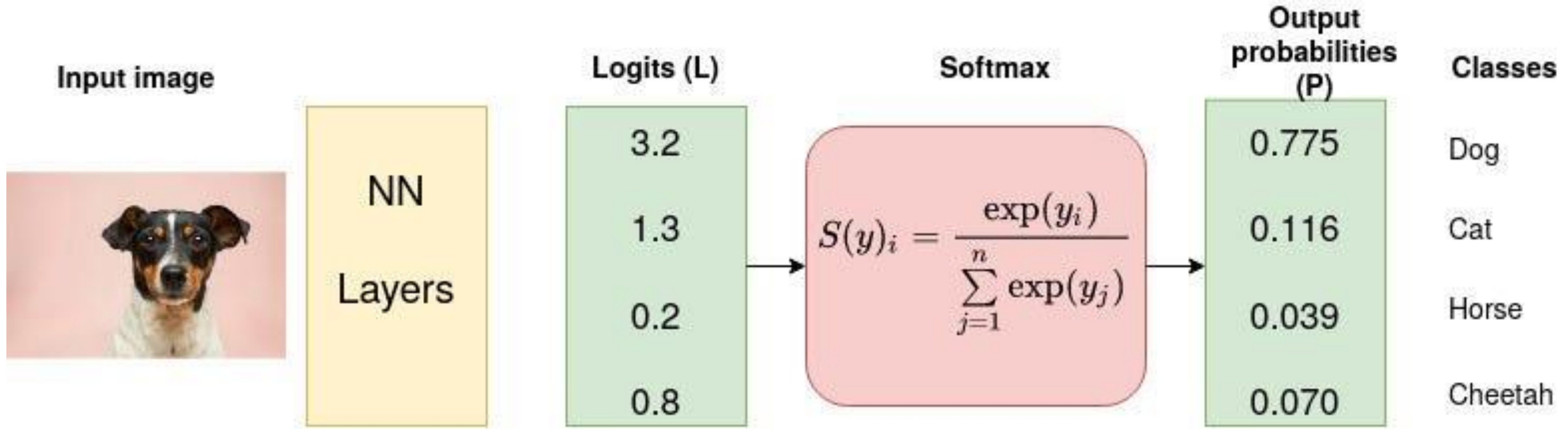
Sigmoid



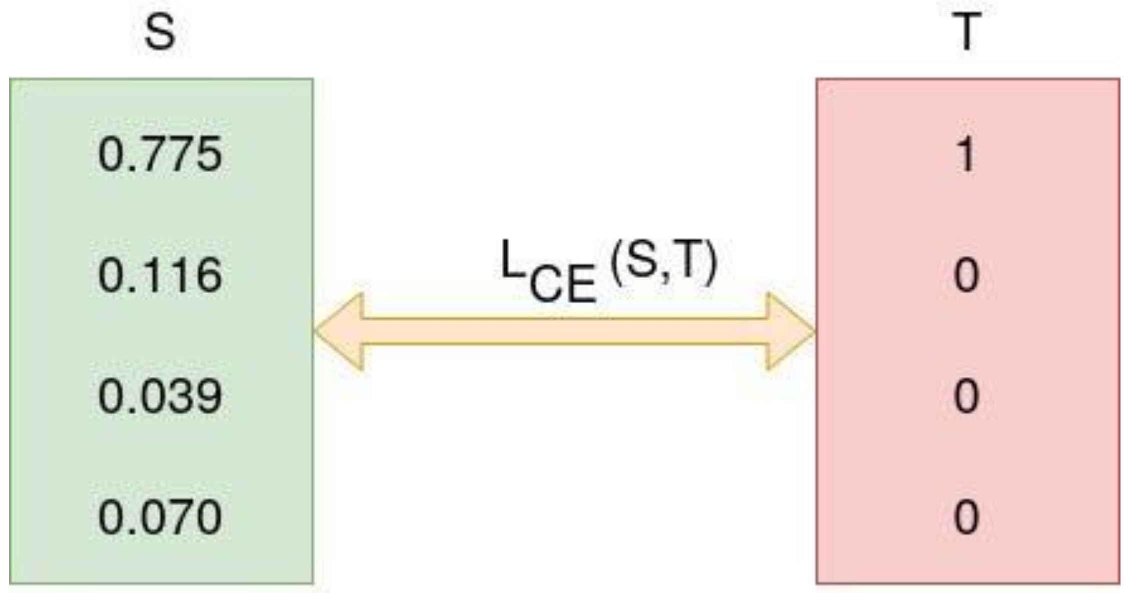
Y	Y_pred		
1	1	➤	Loss is 0
1	0	➤	Loss is very high
0	1	➤	Loss is very high
0	0	➤	Loss is 0

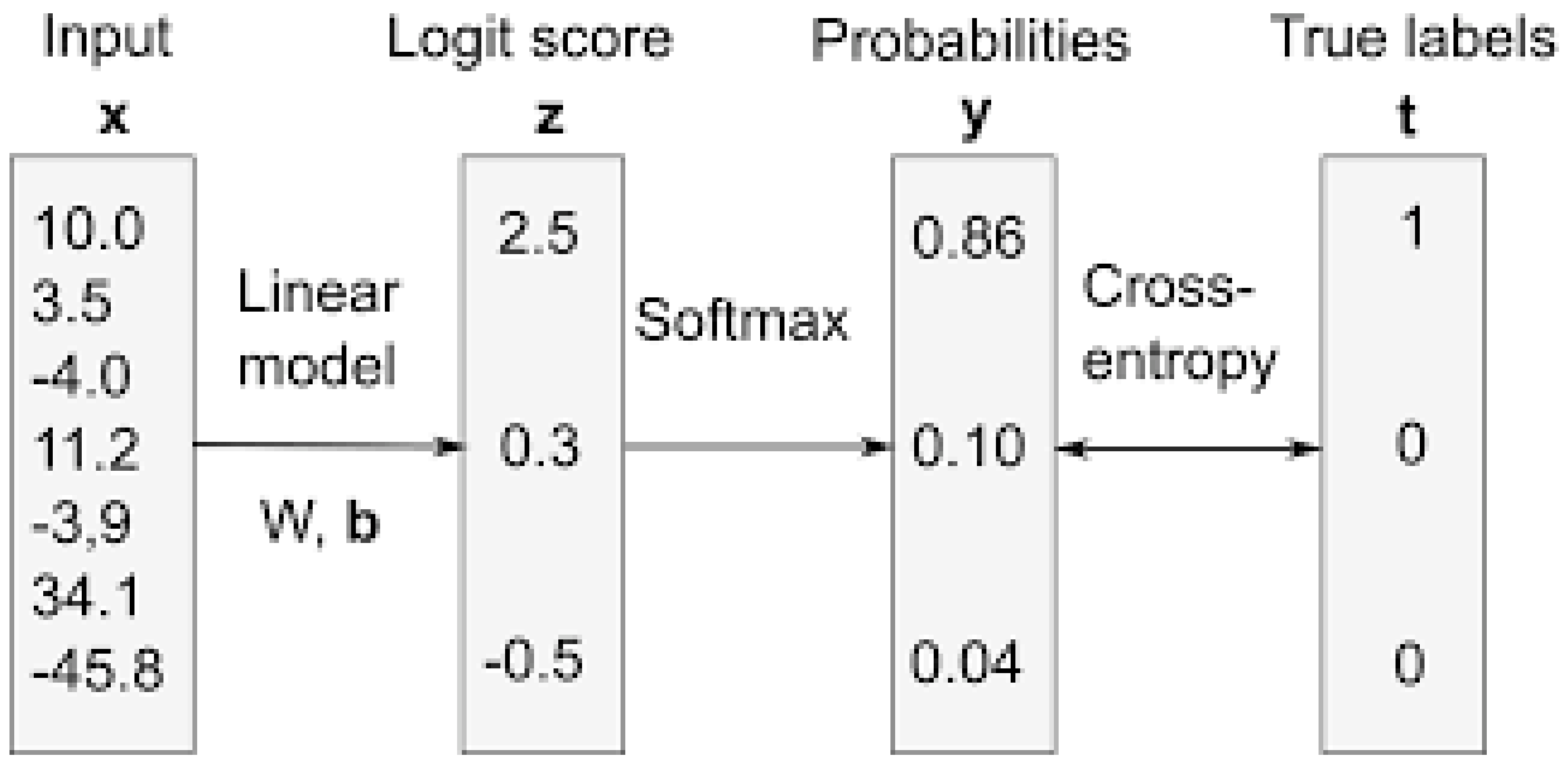
Multi-class Classification



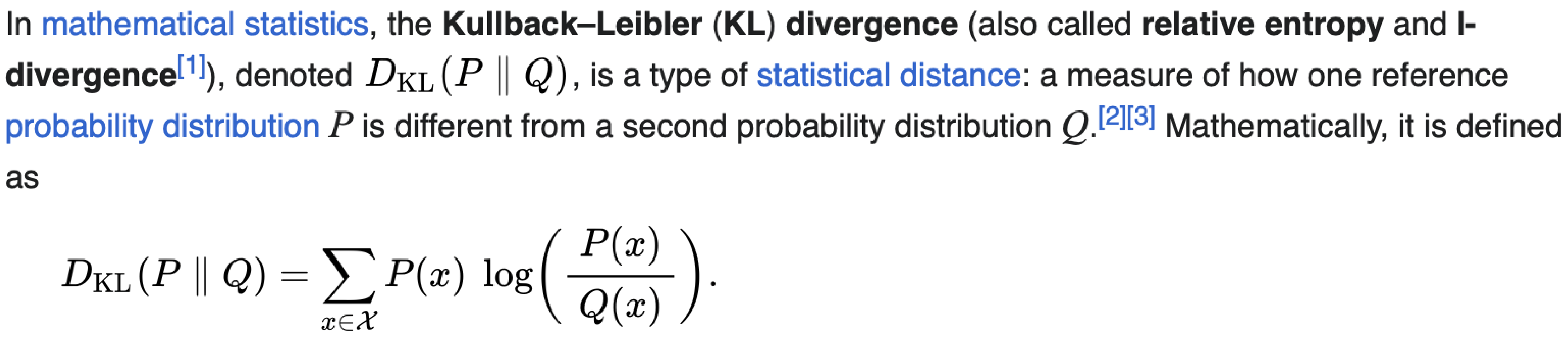


We need a metric that can be used to measure the difference between two probability distributions.



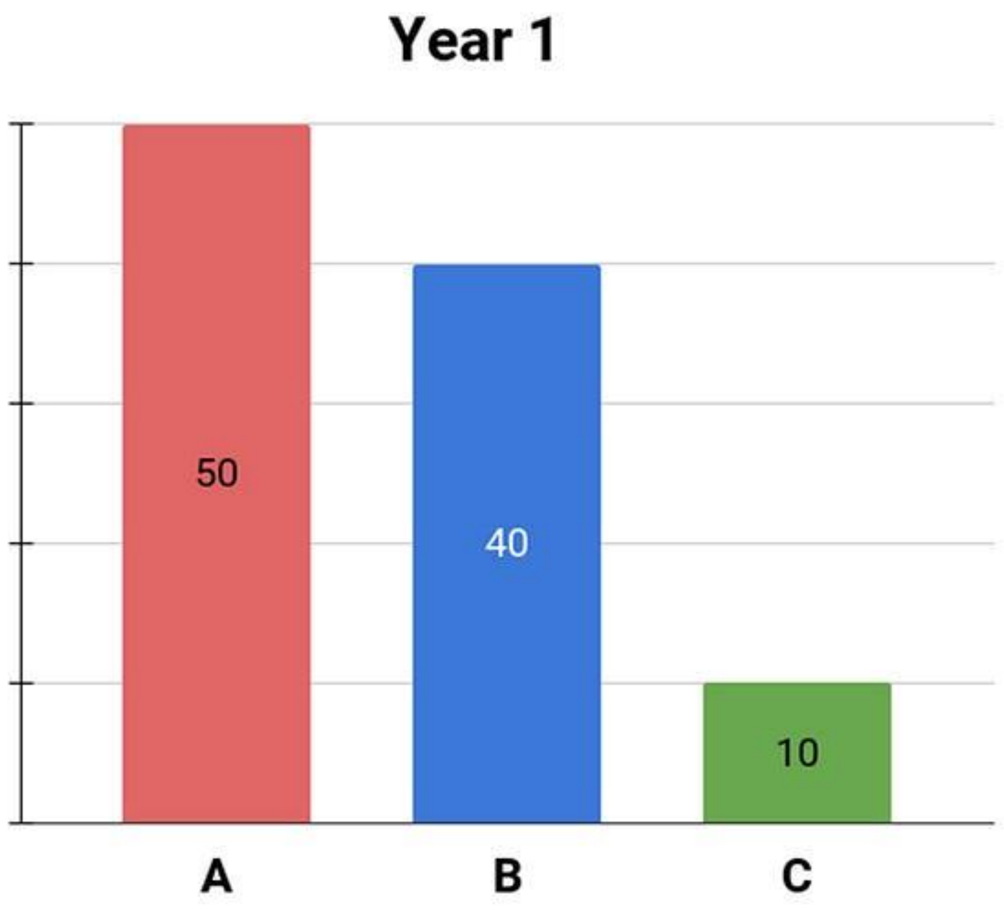

Input
x
10.0
3.5
-4.0
11.2
-3,9
34.1
-45.8
Linear model
W, b
Logit score
z
2.5
0.3
-0.5
Softmax
Probabilities
y
0.86
0.10
0.04
Cross-entropy
True labels
t
1
0
0

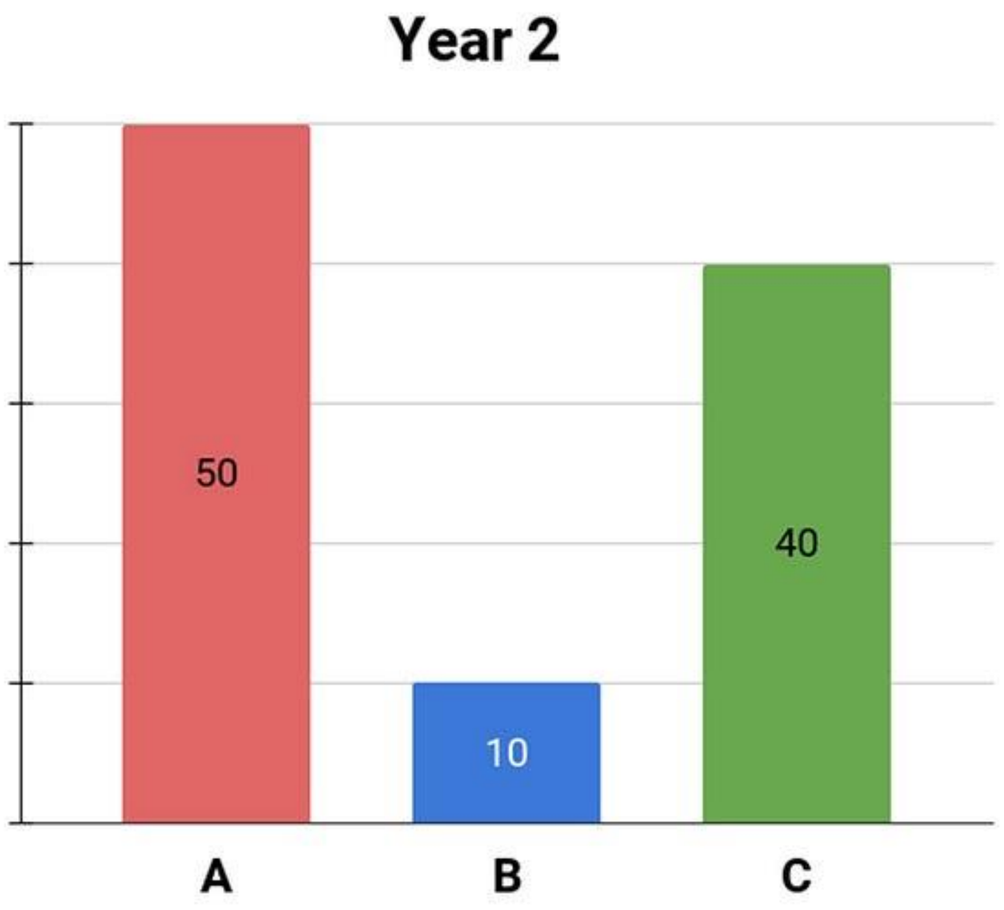
Kullback–Leibler divergence



In mathematical statistics, the **Kullback–Leibler** (**KL**) **divergence** (also called **relative entropy** and **I-divergence**[1]), denoted $D_{\text{KL}}(P \parallel Q)$, is a type of statistical distance: a measure of how one reference probability distribution P is different from a second probability distribution Q.[2][3] Mathematically, it is defined as

$$D_{\text{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x)\ \log\left(\frac{P(x)}{Q(x)}\right).$$


Year 1
50
40
10
A
B
C

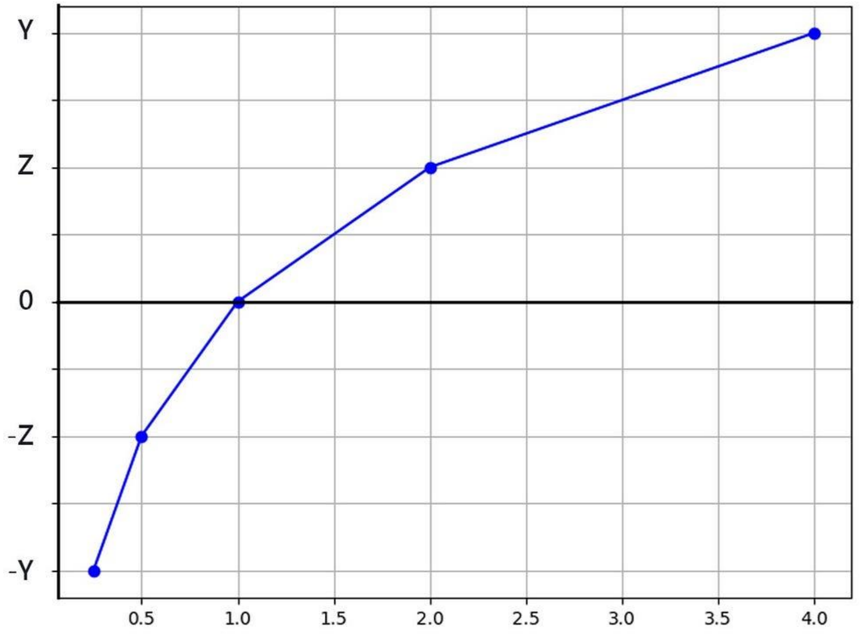

Year 2
50
10
40
A
B
C

A flaw introduced by this method.

$$\frac{(1 + 1/4 + 4)}{3} = 1.75$$

We know that averages can be skewed by large numbers. In our case, the ratios ¼ and 4 represent opposing yet equal influences. However, when averaged, the influence of 4 dominates, potentially inflating our metric. Thus, a simple average might not be the ideal solution.

To rectify this, let’s explore a transformation. Can we find a function, denoted as F, to apply to these ratios (1, ¼, 4) that satisfies the requirement of treating opposing influences equally? We seek a function where, if we input 4, we obtain a certain value (y), and if we input 1/4, we get (-y).



$$\frac{\sum_x \log\left(\frac{P(x)}{Q(x)}\right)}{n}$$

$$\frac{\sum_x \log\left(\frac{P(x)}{Q(x)}\right)}{n}$$

Instead of this equal weighting, let’s use a probabilistic weighting based on the proportion of students that like a particular subject in the current distribution, denoted by ***P(x).***

$$\sum_x P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

$$D_{\mathrm{KL}}(p(x) \,||\, q(x)) = \sum_{x \in X} p(x) ln \quad \frac{p(x)}{q(x)}$$

The metric is asymmetric. Did we satisfy that? Switching P and Q in the formula yields different results, aligning with our requirement for an asymmetric metric.

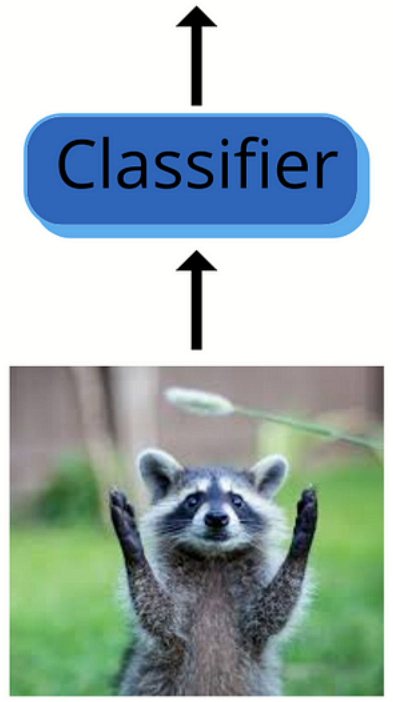
$$CrossEntropy, H(p, q) = -\sum p(i) \log(q(i))$$

$$CrossEntropy = Entropy + KL - Divergence$$

$$D_{KL}(p||q) = H(p,q) - H(p) = -\sum_i p_i \log(q_i) - (-\sum_i p_i \log(p_i))$$

$$D_{KL}(p||q) = -\sum_i p_i \log(q_i) + \sum_i p_i \log(p_i) = \sum_i p_i \log \frac{p_i}{q_i}$$

	CAT	DOG	FOX	RED PANDA	RACCOON	COW	BEAR
True Distribution:	0%	0%	0%	0%	100%	0%	0%
Predicted Distribution:	5%	2%	45%	10%	30%	1%	2%

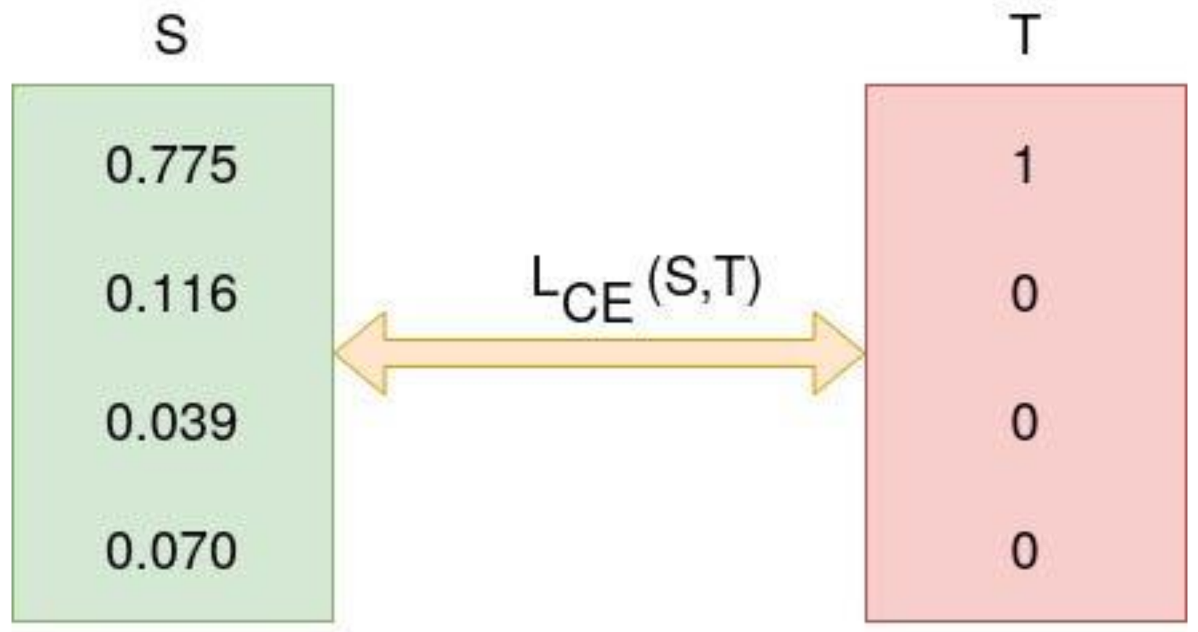


Cross-Entropy is -1*log(0.3) = — log(0.3) = 1.203

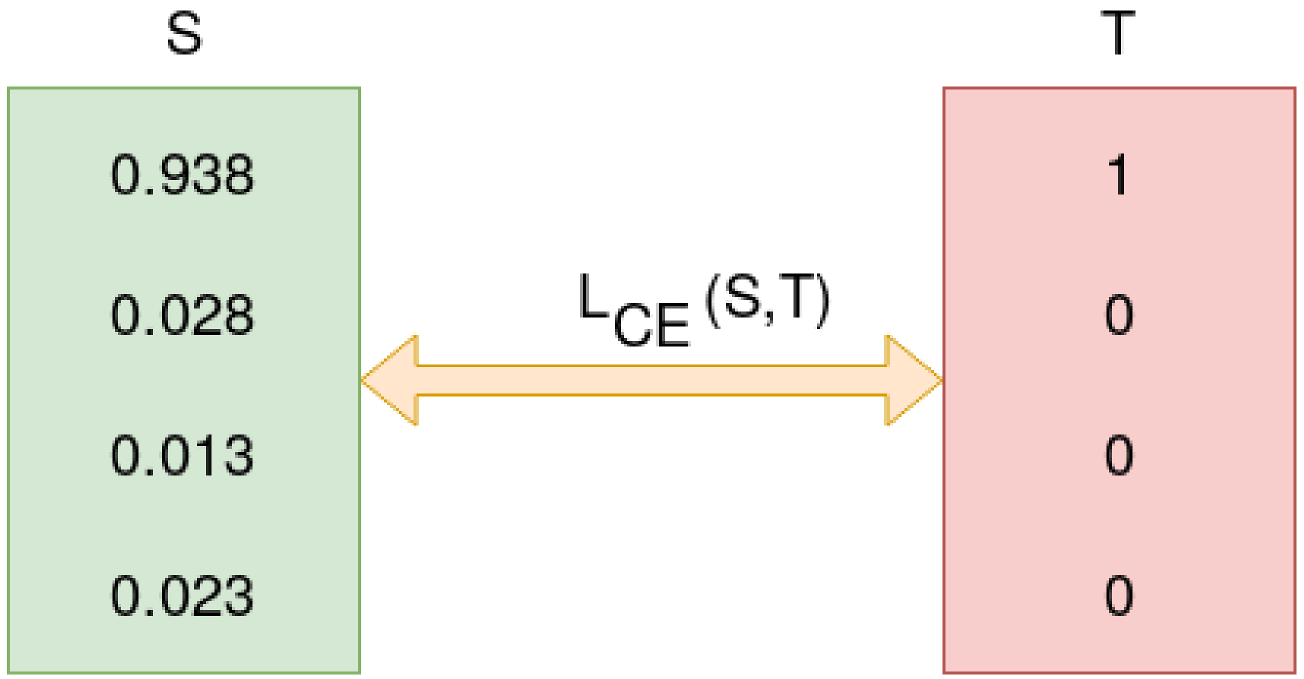
Cross-Entropy Loss Function

$$L_{\text{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.



$$
\begin{aligned}
L_{CE} &= -\sum_{i=1} T_i \log(S_i) \\
&= -\left[1\log_2(0.775) + 0\log_2(0.126) + 0\log_2(0.039) + 0\log_2(0.070)\right] \\
&= -\log_2(0.775) \\
&= 0.3677
\end{aligned}
$$



$$\begin{aligned} L_{CE} &= -1\log_2(0.936) + 0 + 0 + 0 \\ &= 0.095 \end{aligned}$$

Binary Cross-Entropy Loss

$$\begin{aligned}
L &= -\sum_{i=1}^{2} t_i \log(p_i) \\
&= -[t_1 \log(p_1) + t_2 \log(p_2)] \\
&= -[t \log(p) + (1-t) \log(1-p)]
\end{aligned}$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the i^{th} class. Since we have two classes 1 and 0 we can have $t_1 = 1$ and $t_2 = 0$ and since $p's$ are probabilities then $p_1 + p_2 = 1 \implies p_1 = 1 - p_2$. For the convenience of notation, we can then let $t_1 = t, t_2 = 1 - t, p_1 = p$ and $p_2 = 1 - p$.

Binary Cross-Entropy Cost function

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}[t_j \log(p_j) + (1 - t_j)\log(1 - p_j)]\right]$$

Calculated as the average cross-entropy across all data examples

Anatomy of a neural network

- **Architecture (layer)**
- **Input data presentation**
- **Loss function**
- **Optimizer**

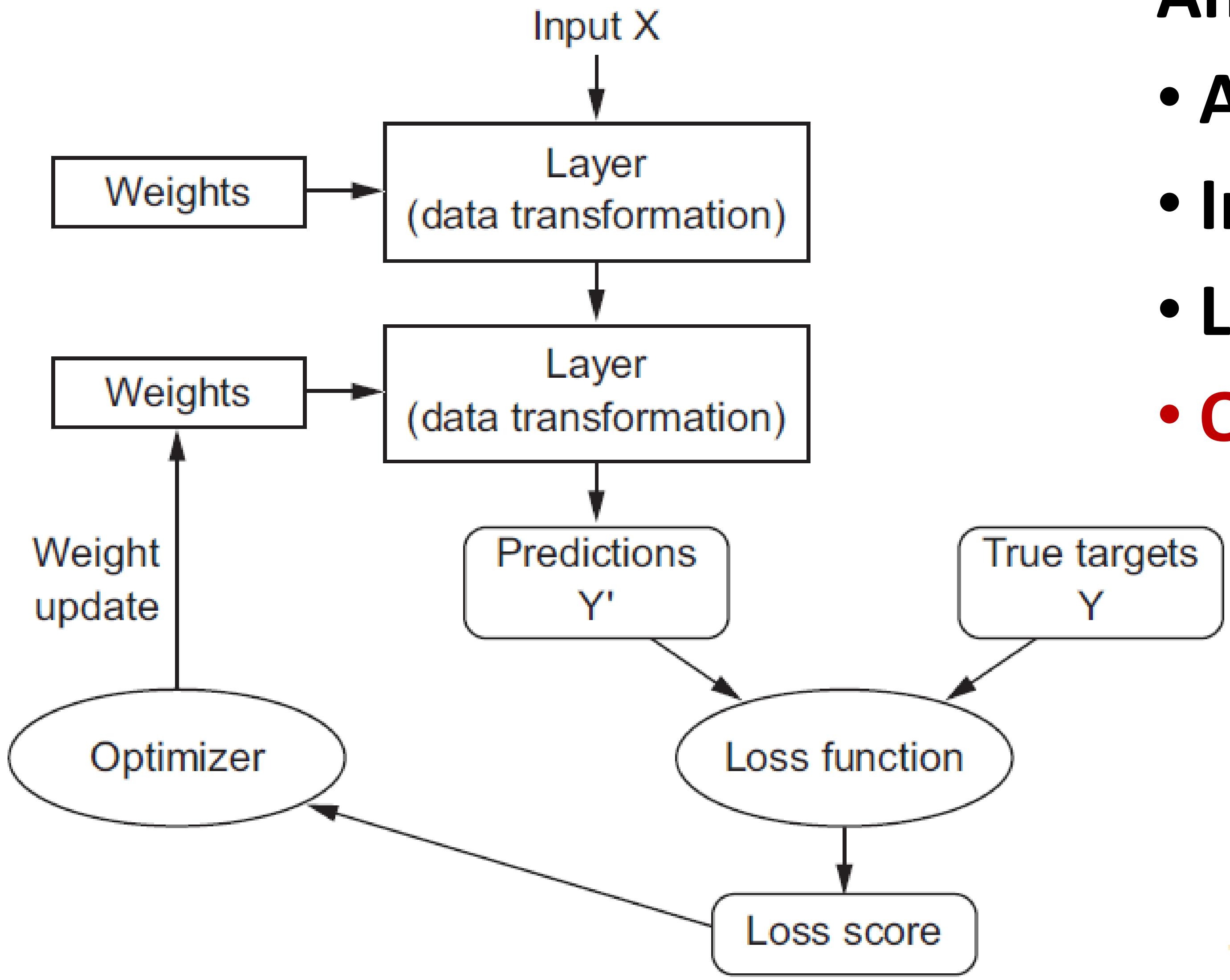


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

Steps to train deep learning model

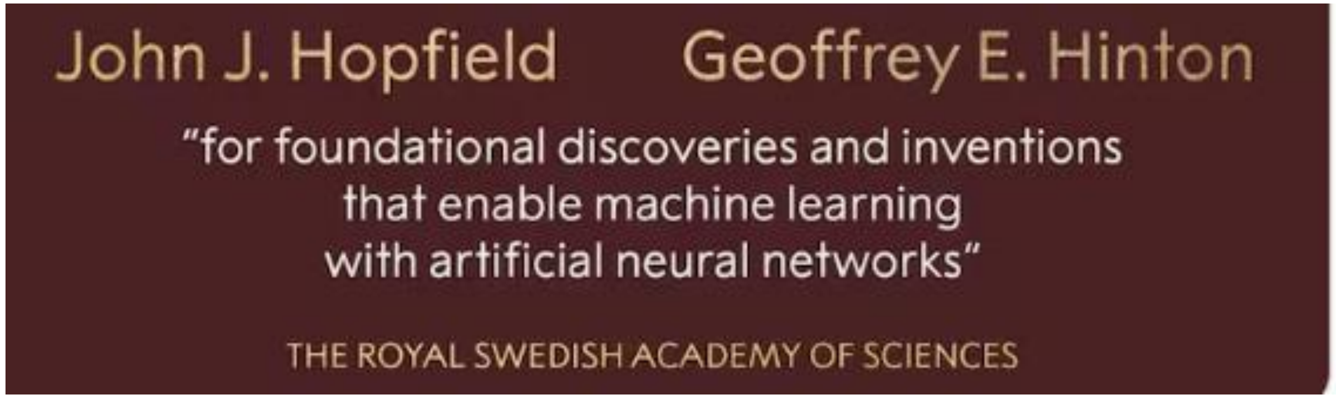
1. Weight and bias initialization
2. Forward pass the data points
3. Calculate the total cost (error) using a suitable loss function
4. Use the backpropagation algorithm to compute the error gradient to all the parameters starting from the output layer back to the input layer
5. Update the parameters using Gradient Descent based optimization algorithm
6. Repeat step 2 to step 5 until the convergence

Backpropagation Algorithm Steps:

1. Initialize the weights of the network randomly.
2. Forward propagate an input through the network to get the predicted output.
3. Compute the error between the predicted output and the actual output.
4. Backward propagate the error through the network to compute the gradient of the loss function with respect to each weight.
5. Update the weights in the opposite direction of the gradient using an optimization algorithm such as Stochastic Gradient Descent (SGD).
6. Repeat steps 2–5 for multiple iterations until the weights converge.

Geoffrey Hinton

Emeritus Prof. Computer Science, University of Toronto



John J. Hopfield Geoffrey E. Hinton

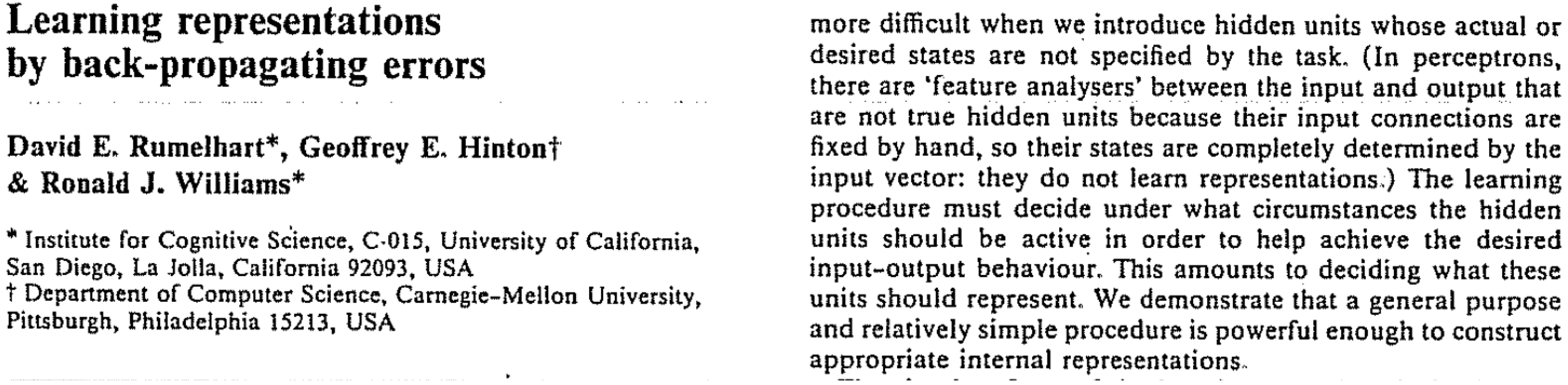
"for foundational discoveries and inventions that enable machine learning with artificial neural networks"

THE ROYAL SWEDISH ACADEMY OF SCIENCES

The basics of backpropagation were derived in the context of control by multiple researchers in the early. The term and its general use in neural networks were proposed in the 1986 Nature paper Learning Representations by Backpropagating Errors, co-authored by David Rumelhart, Hinton, and Ronald Williams.

He was the person who demonstrated that backpropagation could learn interesting internal representations and that this is what made it popular.

1986



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

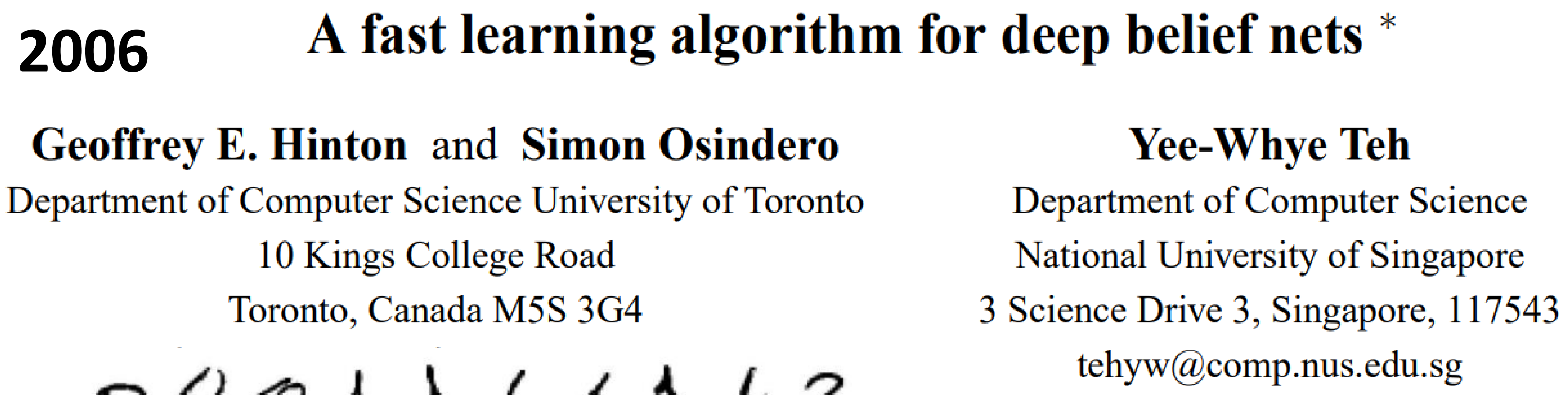
* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

AI winter

- Hardware support
- Data scarcity
- Weight initialization

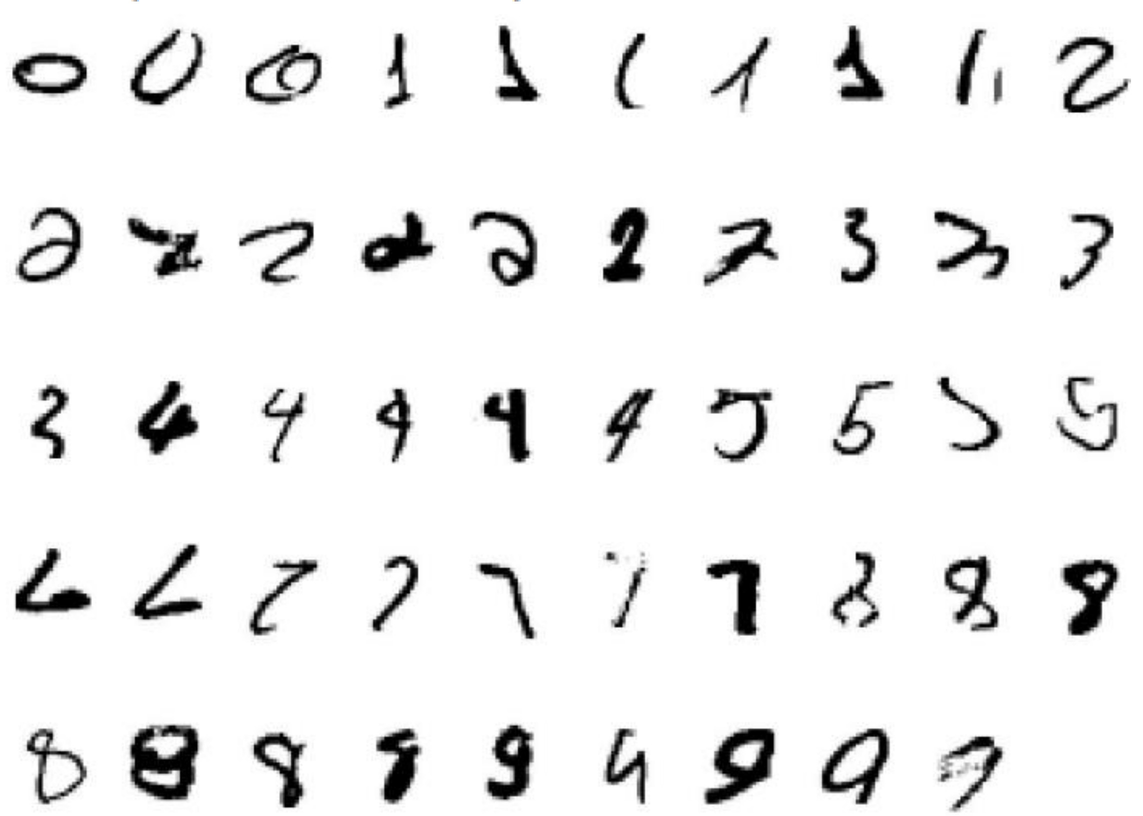
2006



A fast learning algorithm for deep belief nets *

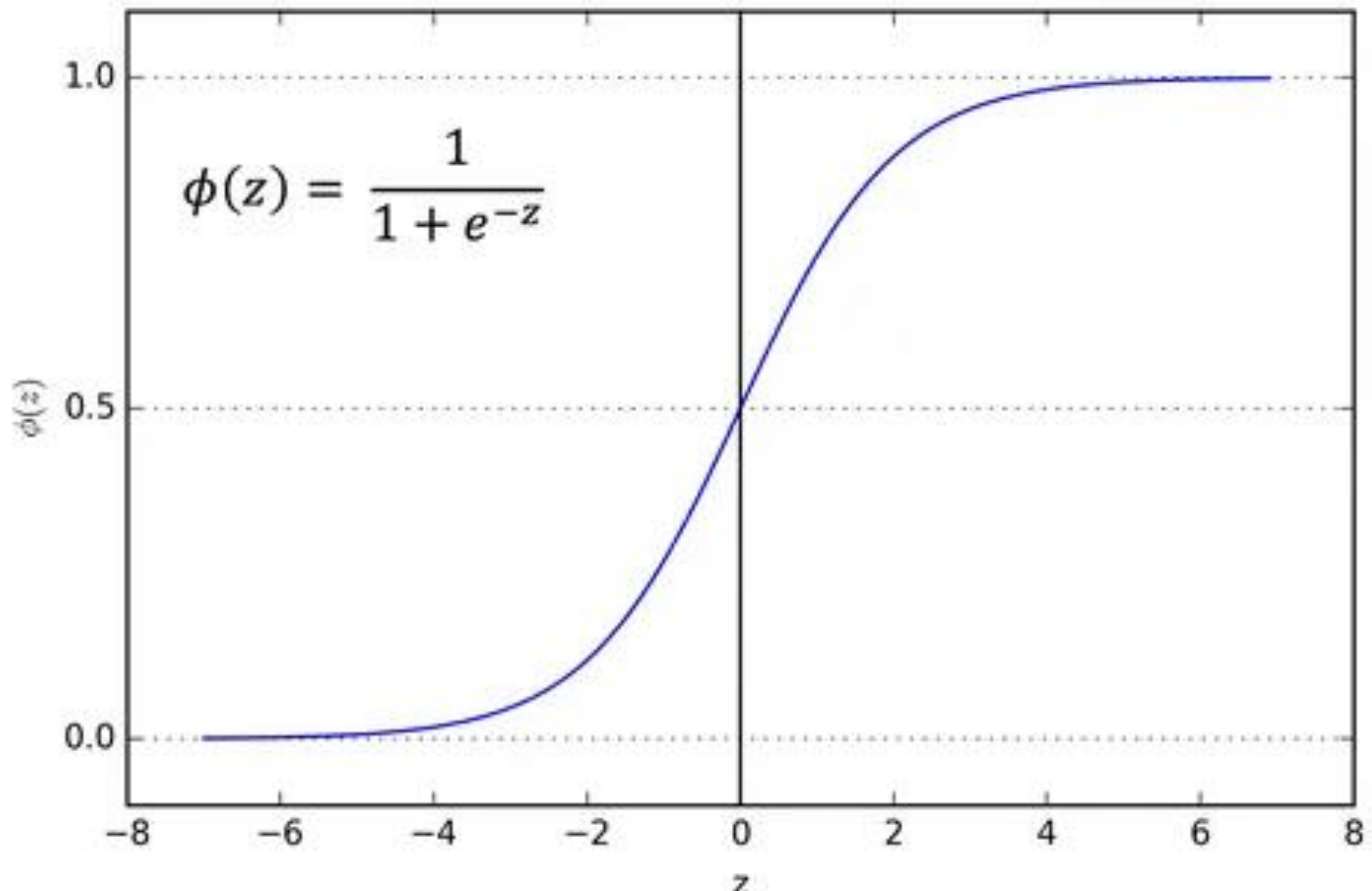
Geoffrey E. Hinton and **Simon Osindero**
Department of Computer Science University of Toronto
10 Kings College Road
Toronto, Canada M5S 3G4

Yee-Whye Teh
Department of Computer Science
National University of Singapore
3 Science Drive 3, Singapore, 117543
tehyw@comp.nus.edu.sg

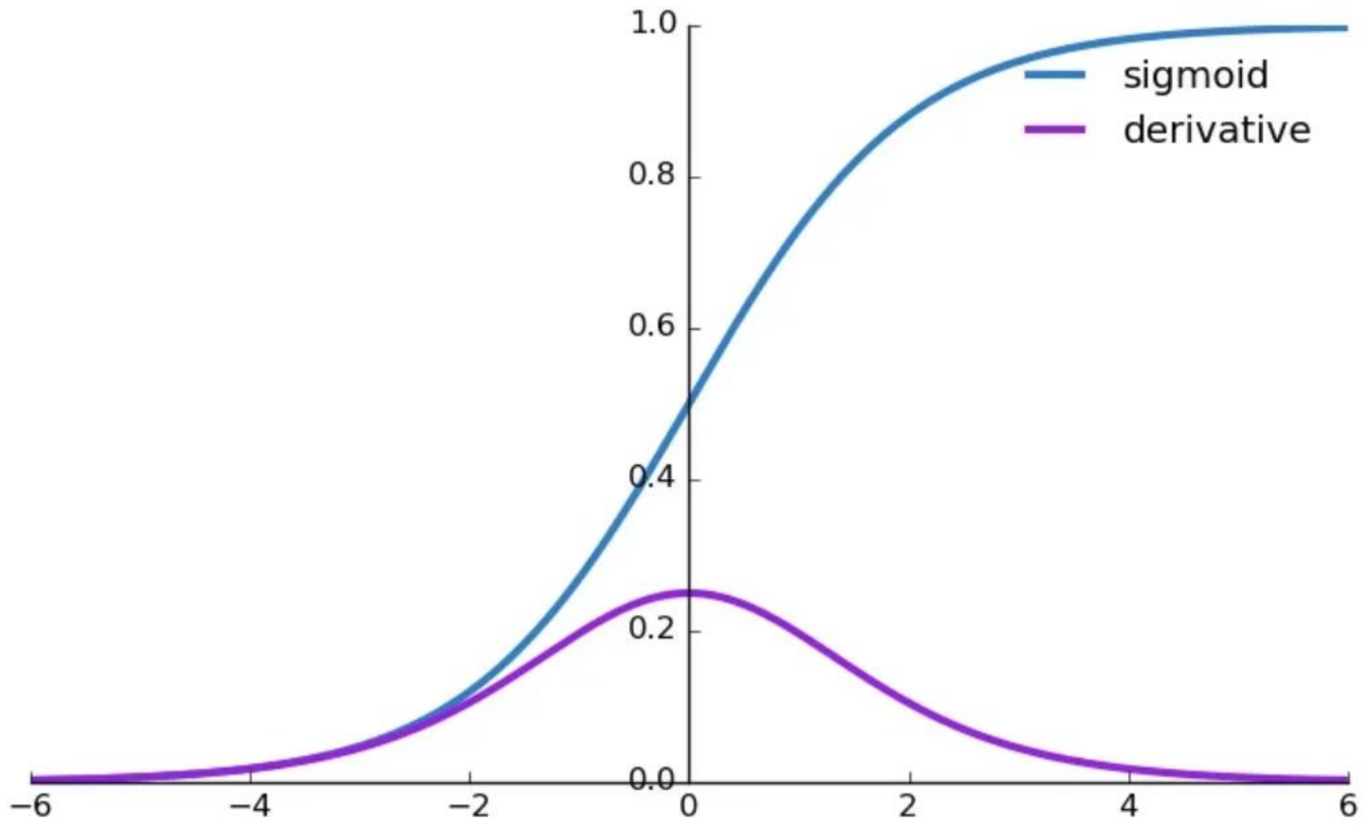


MNIST dataset

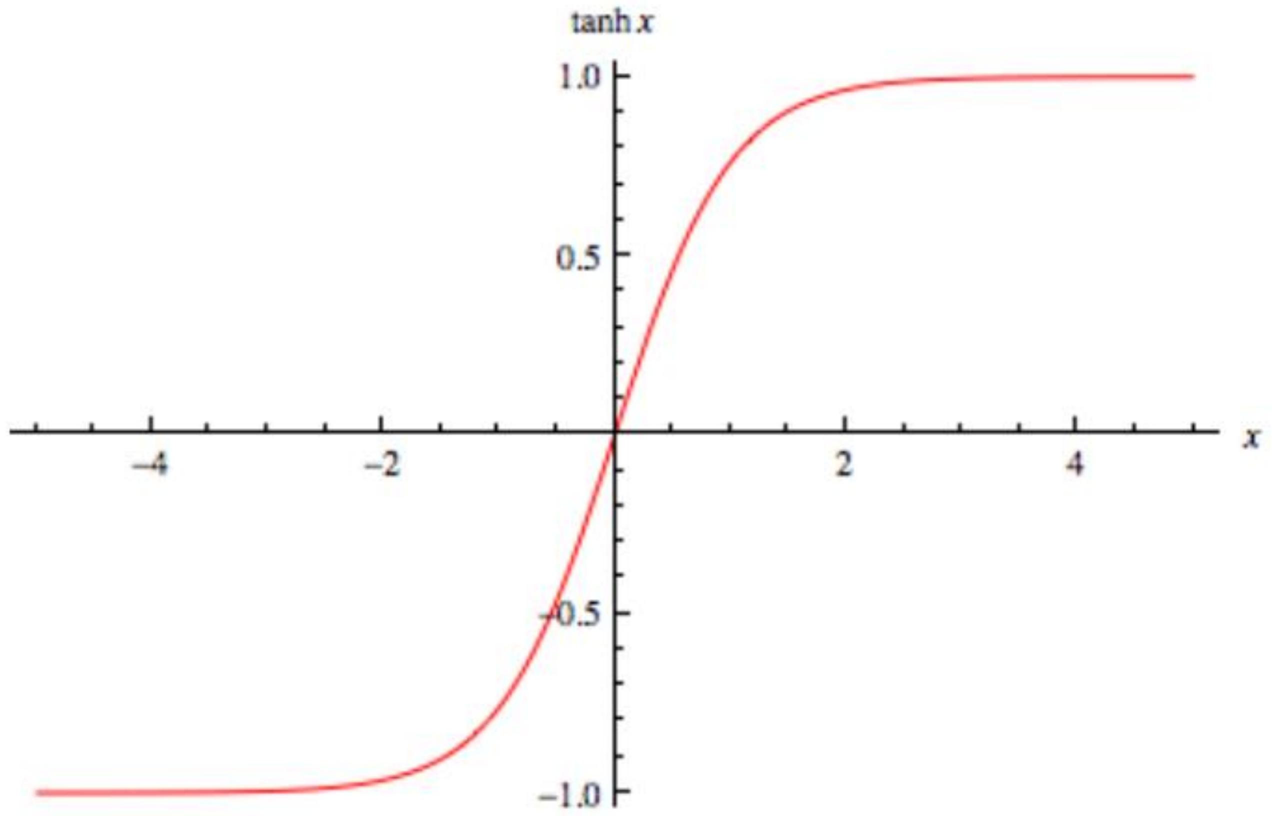
Sigmoid



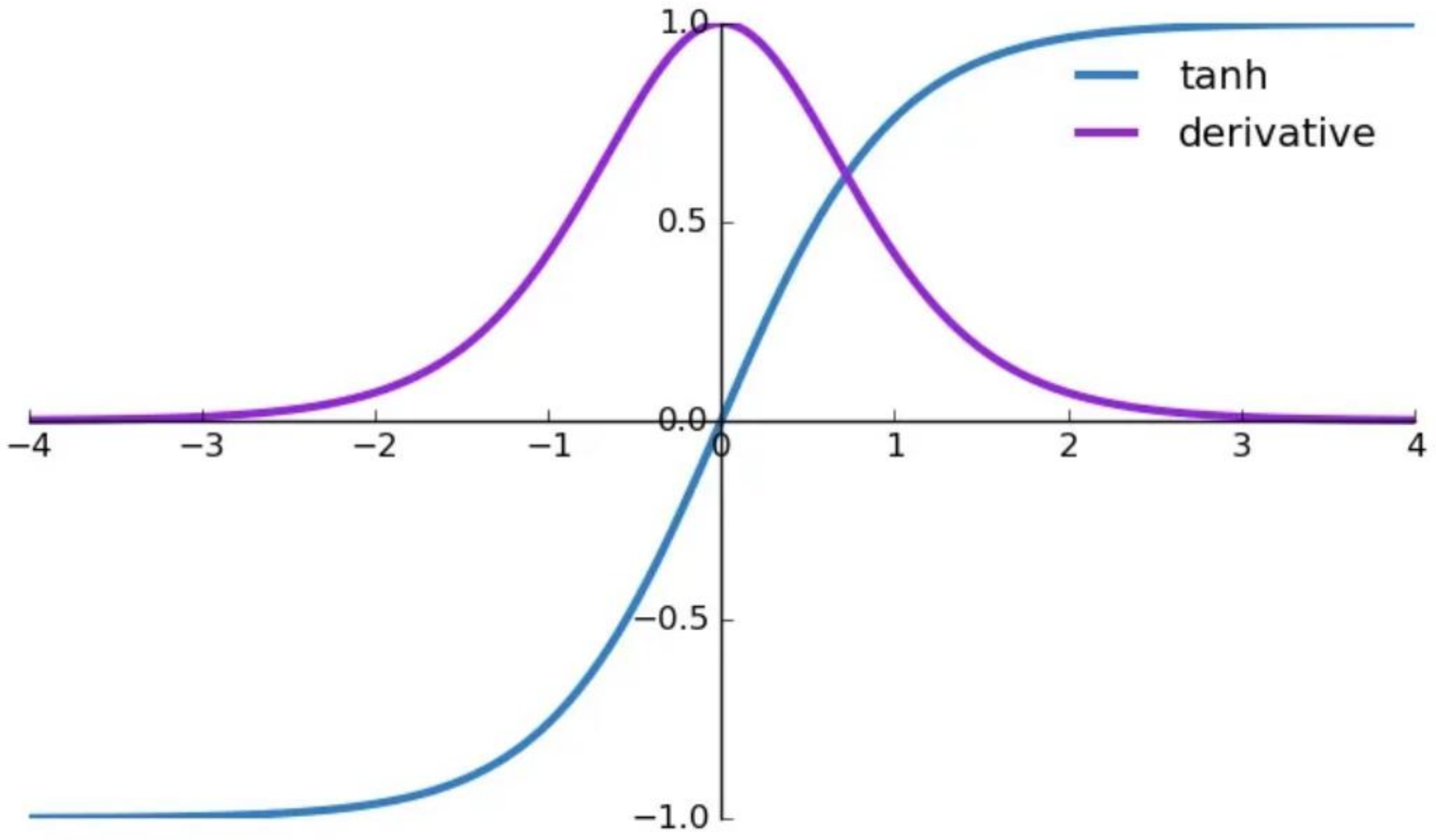
Graph of sigmoid derivative



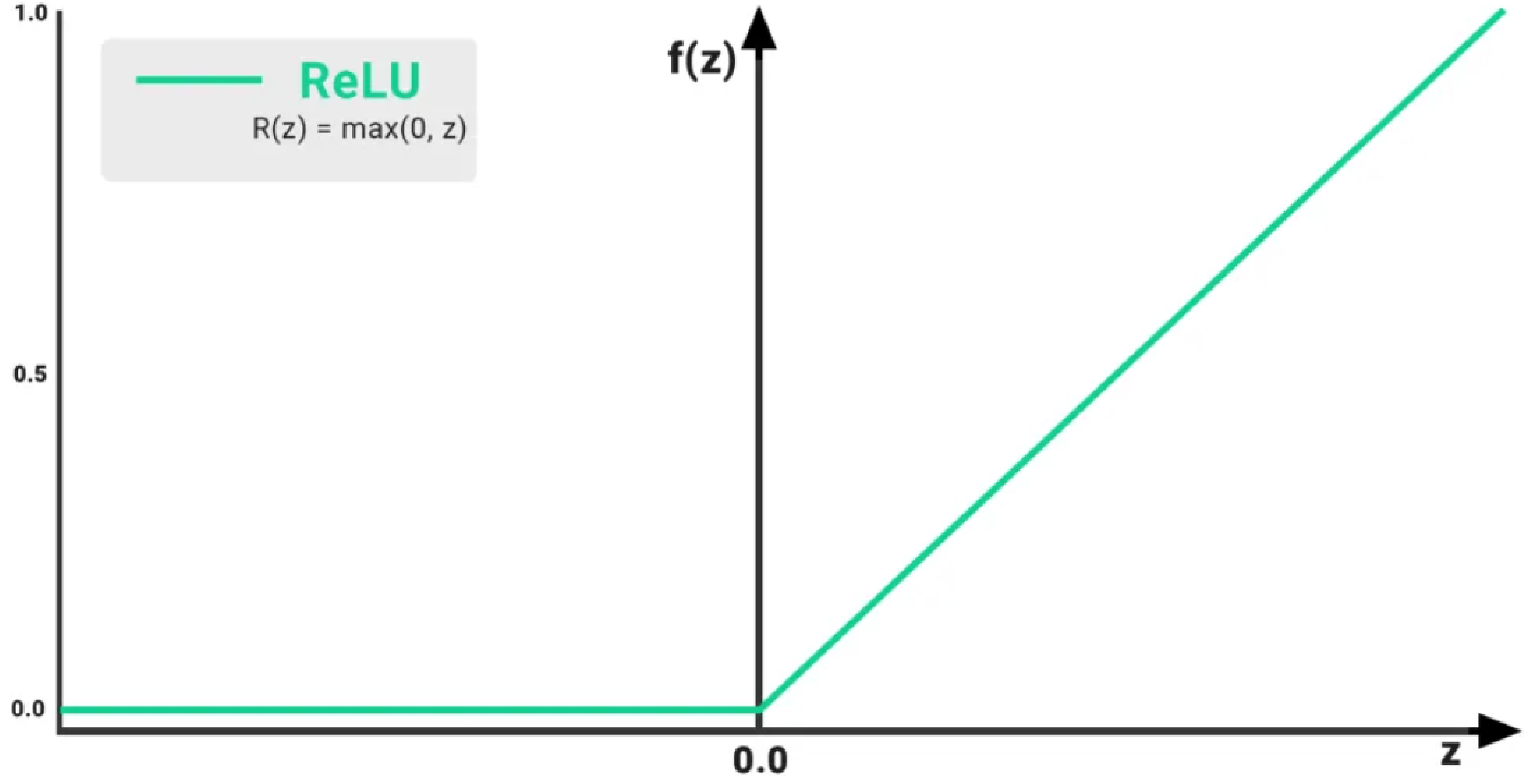
Graph of Tanh function



Graph of derivative of tanh function



Graph for ReLU function



Equation of ReLU function

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

Equation of derivative of ReLU function

$$R'(z) = \left\{ \begin{array}{ll} 1 & z > 0 \\ 0 & z < 0 \end{array} \right\}$$

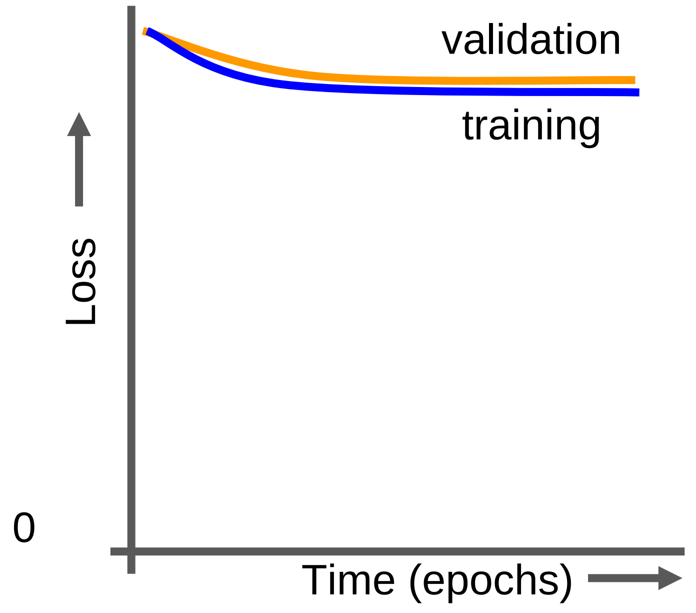
Recap

- ANN → FCN (Dense)
- Forward pass
- Backpropagation
- Loss functions

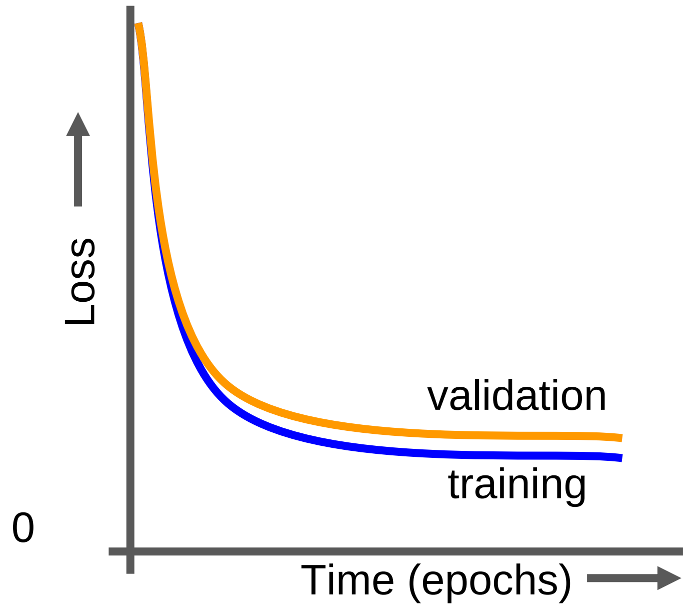
Important factors:

1. Weight Initialization → xavier, lecun, etc.
2. Optimization → Adam, RMSProp, Adagrad, etc
3. Hyperparameter Tuning
4. Tackling overfitting → Dropout
5. Solving gradient problem → proper activation function,
   Batch-Normalization,
   Gradient clipping

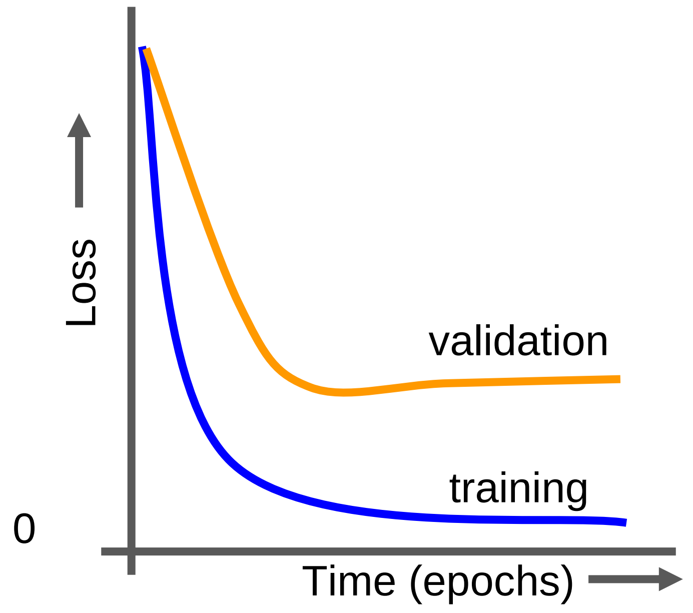
Spotting Underfitting and Overfitting



Underfit: Model performs poorly on training and validation data



Good fit: Model generalizes well from training to validation data



Overfit: Model predicts training data well but fails to generalize to validation data