Imperial College Business School

Imperial means Intelligent Business

Machine Learning in Finance

Lecture 4
Introduction to Deep Learning

Arnaud de Servigny & Jeremy Chichportich

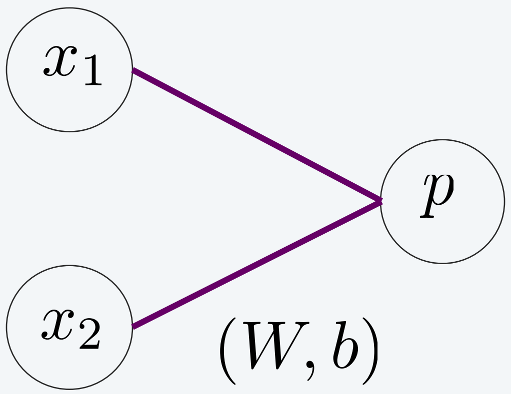
Outline:

- From Logistic Regression to Shallow Neural Networks
- Deep Neural Networks and Loss functions
- Deep Learning Techniques
- Programming Session

Part 1 : From Logistic Regression to Shallow Neural Networks

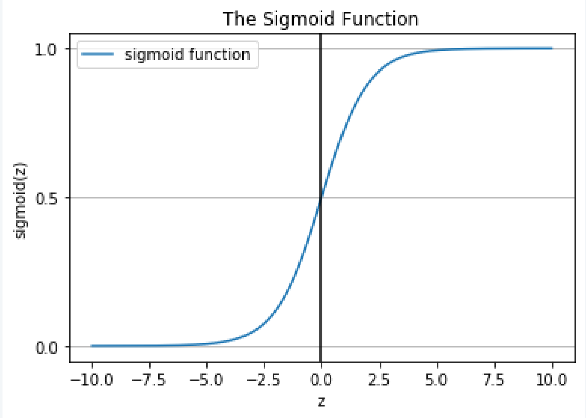
Logistic Regression

- The Logistic Regression Model predicts the probability of the positive class using the combination of linear decision function and a sigmoid function

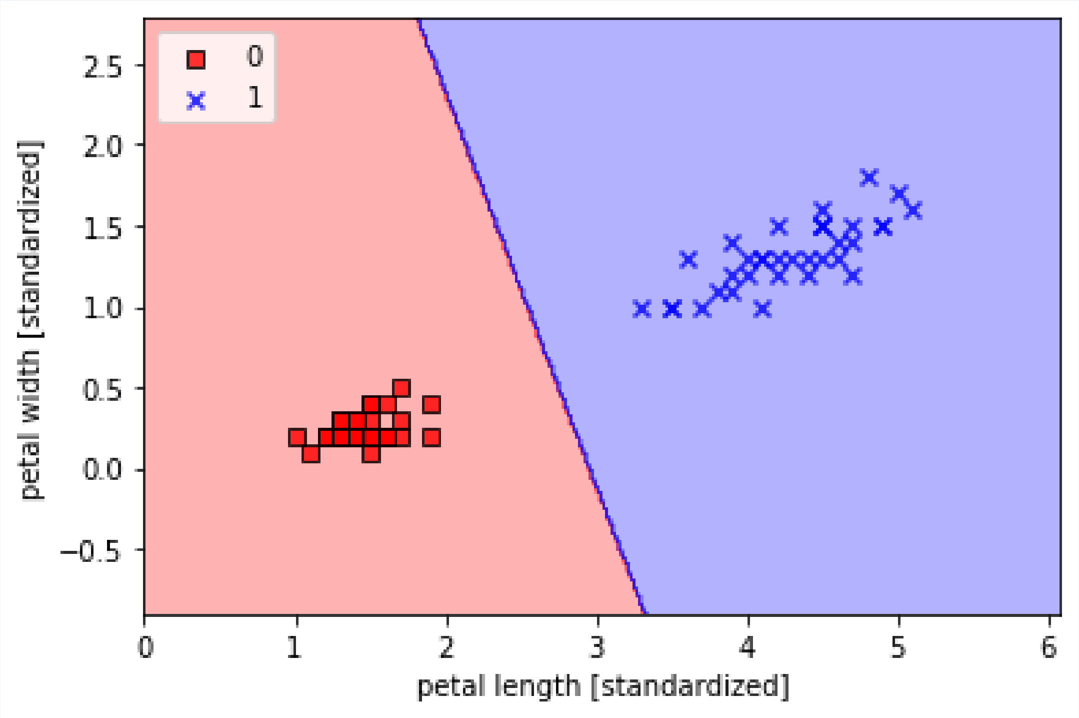


$$p = \mathbb{P}(Y = 1 | X_1 = x_1, X_2 = x_2)$$
$$= \sigma(W_1 x_1 + W_2 x_2 + b)$$

$$\sigma : z \mapsto \frac{1}{1 + e^{-z}}$$



- As the decision boundary is a **hyperplane,** the Logistic Regression model performs well on linearly separable classes.



Linearly inseparable data

- The basic idea to deal with linearly inseparable data is to create nonlinear combinations of the original features.
- We can then transform a two-dimensional dataset onto a three-dimensional feature space where the classes become linearly separable.

Original features:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

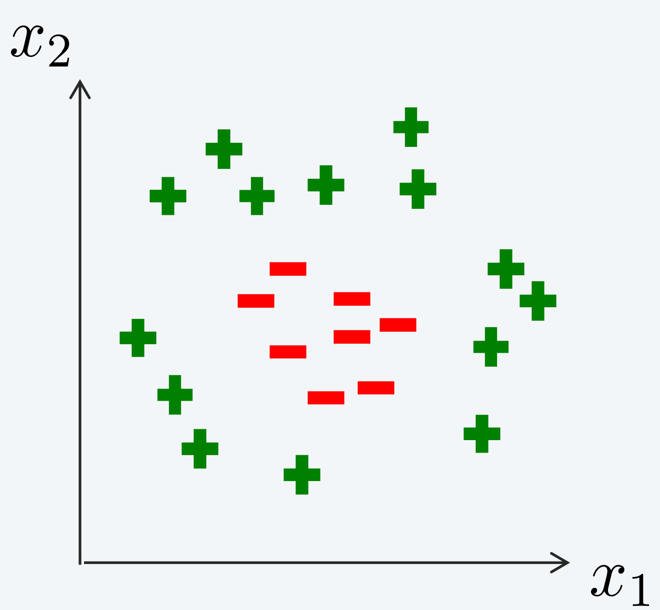
Transformation of the features:

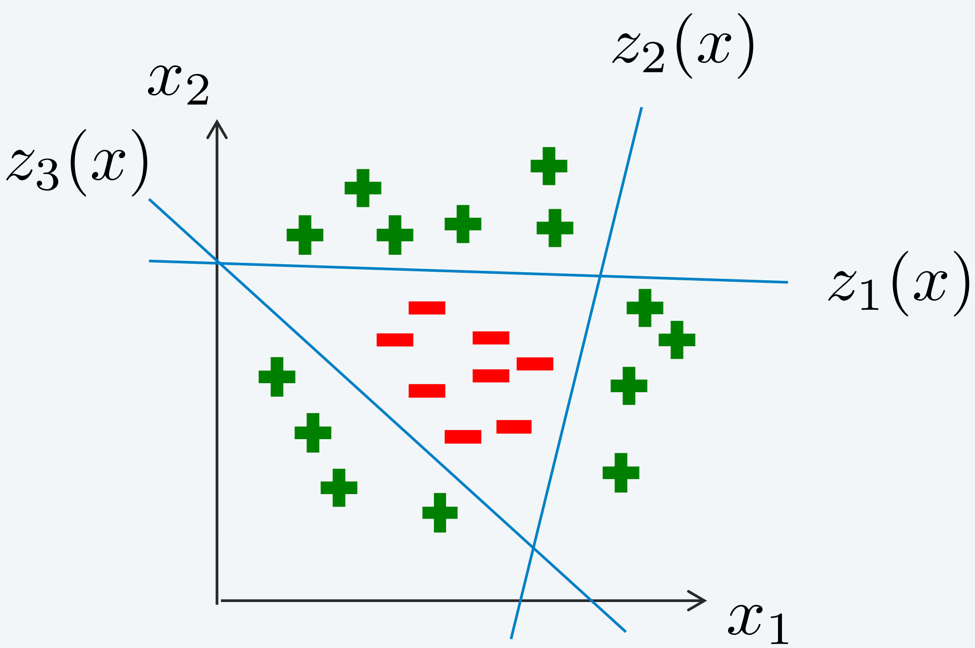
$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

$$\forall i \in \{1, 2, 3\} :$$

$$z_i = \sigma(b_i^{(1)} + W_{1,i}^{(1)} x_1 + W_{2,i}^{(1)} x_2)$$

where $\sigma : z \mapsto \dfrac{1}{1 + e^{-z}}$

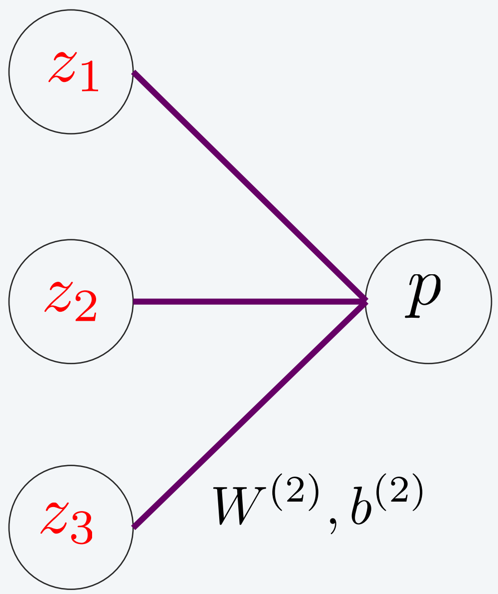




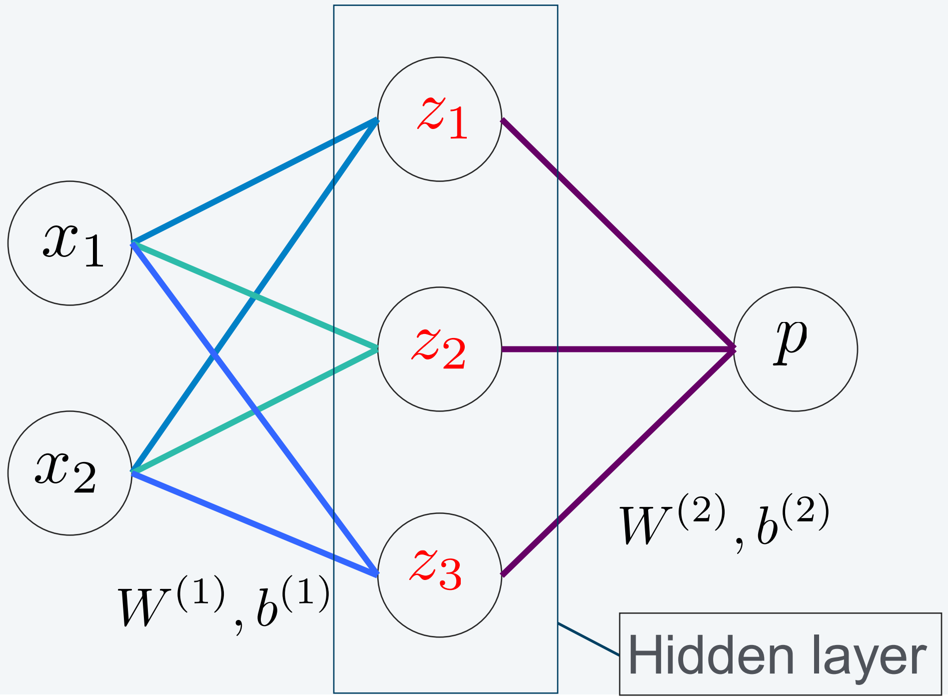
Shallow Neural Network

- After transforming the features, we can build a linear model on top of the new features:

$$p = \sigma(b^{(2)} + W_1^{(2)} z_1 + W_2^{(2)} z_2 + W_3^{(2)} z_3)$$



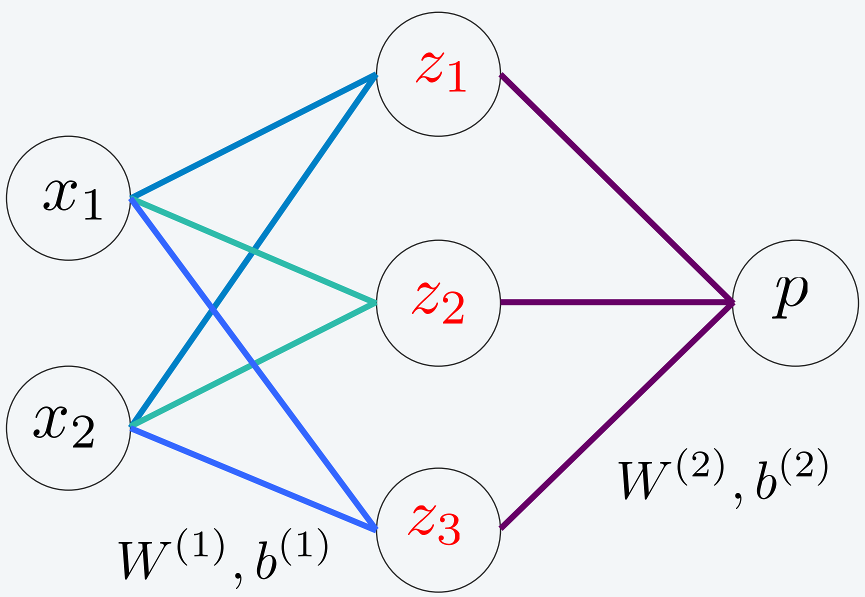
- This model is a basic example of an artificial neural network with **one hidden layer** containing **3 neurons**.



$$\forall i \in \{1, 2, 3\} \quad z_i = \sigma(b_i^{(1)} + W_{1,i}^{(1)} x_1 + W_{2,i}^{(1)} x_2)$$

$$p = \sigma(b^{(2)} + W_1^{(2)} z_1 + W_2^{(2)} z_2 + W_3^{(2)} z_3)$$

Interactive Session



- This is our model :

$$\forall i \in \{1, 2, 3\} \quad z_i = \sigma(b_i^{(1)} + W_{1,i}^{(1)} x_1 + W_{2,i}^{(1)} x_2)$$
$$p = \sigma(b^{(2)} + W_1^{(2)} z_1 + W_2^{(2)} z_2 + W_3^{(2)} z_3)$$

- Prove that without the activation function, the model becomes a simple **linear model**

$$p = \mathbf{U} x_1 + \mathbf{V} x_2 + \mathbf{b}$$

Training the Shallow Neural Network – Part 1 –

- We used Gradient Descent to learn the parameters of Logistic Regression, we can do the same for this shallow neural network model:

$$\forall i \in \{1,2,3\} \quad z_i = \sigma(b_i^{(1)} + W_{1,i}^{(1)} x_1 + W_{2,i}^{(1)} x_2)$$

$$p = \sigma(b^{(2)} + W_1^{(2)} z_1 + W_2^{(2)} z_2 + W_3^{(2)} z_3)$$

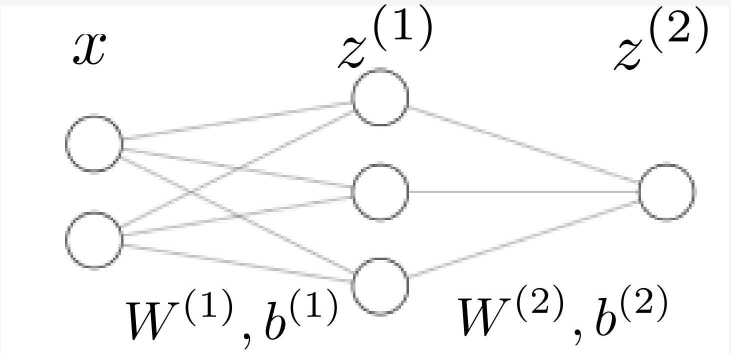
- The parameters θ of the model can be summerized as follows:

$$(W^{(1)} \in \mathbb{R}^{2\times 3}, b^{(1)} \in \mathbb{R}^3) \quad \text{and} \quad (W^{(2)} \in \mathbb{R}^{3\times 1}, b^{(2)} \in \mathbb{R})$$

- The Gradient Descent algorithm requires the use of **backpropagation.**

- **Backpropagation** consists in computing the gradient of the loss function J (that will be detailed later) with respect to each weight by the chain rule, iterating backward from the last layer.

Backpropagation



Chain rule

$$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial W^{(1)}}$$

$$\frac{\partial J}{\partial W^{(2)}} = \frac{\partial J}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial W^{(2)}}$$

Training the Shallow Neural Network – Part 2 –

- The Gradient Descent (Batch) algorithm consists in the following steps:
  - Initilize randomly θ_0
  - Fix a number of iterations K and a learning rate η and repeat K times:

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta J(\theta_k)$$

- J represents the loss function. We will detail later how to choose the appropriate one.

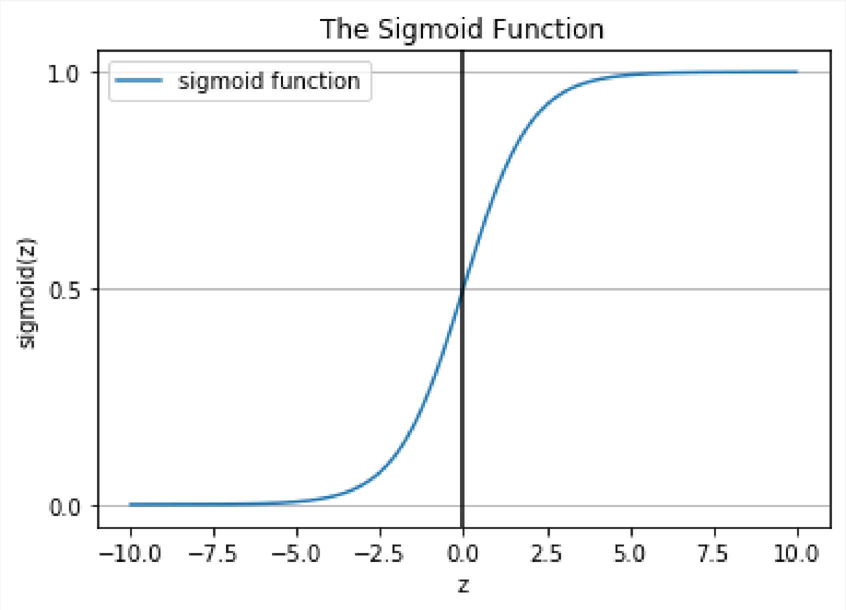
- We will also see in Lecture 7 several ways to improve the learning process:
  - By using Momentum.
  - By using adaptive learning rate.

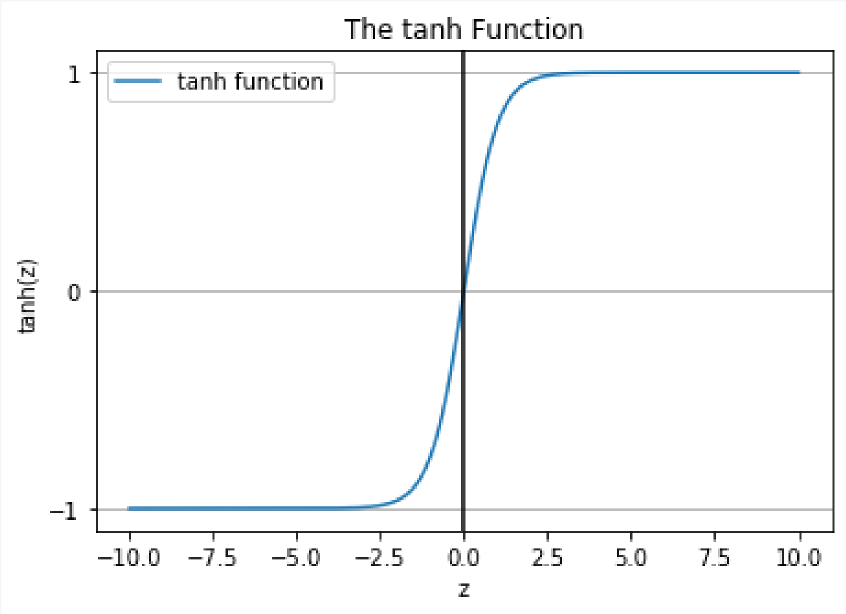
The activation functions – Part 1 –

- In the previous example, the non linearity results from the sigmoid function.
- The sigmoid function takes values in $[0,1]$, which is convenient when we want to output a probability.
- However, we can still use other functions to introduce non linearity in the intermediate layers.
- For instance, the hyperbolic tangent can also be used. It is just a scaled and shifted version of the sigmoid, so it has the same shape of the sigmoid with the advantage of taking values in $[-1,1]$.

$$\text{sigmoid}: \quad z \mapsto \frac{1}{1+\exp(-z)}$$

$$\tanh: \quad z \mapsto \frac{\exp(2z)-1}{\exp(2z)+1}$$





The activation functions – Part 2 –

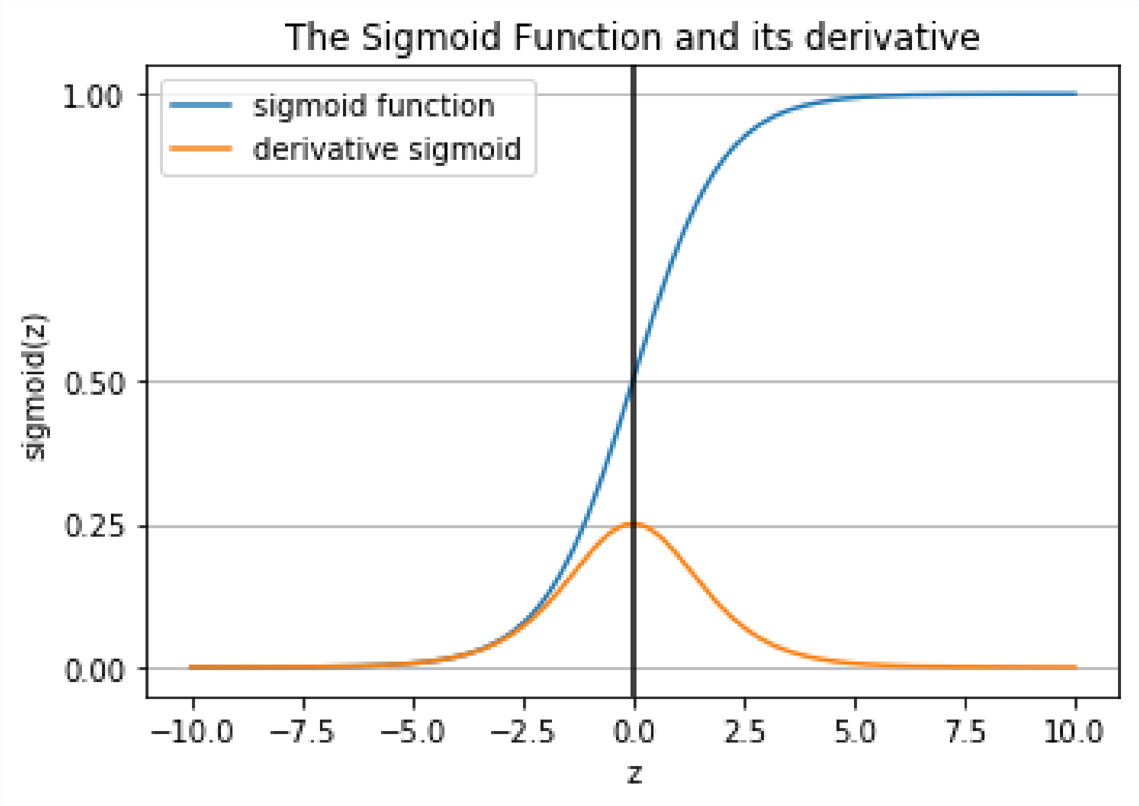
- Although the sigmoid and tanh functions have good theoratical properties (both smooth, both diffentiable everywhere), they suffer from the **vanishig gradient problem**.

- To train the model we use **backpropagation.** However, the deeper the neural network is, the more terms have to be multipled due to the chain rule.

- For instance, with L layers: $$\frac{\partial J}{\partial W^{(1)}} = \frac{\partial J}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial z^{(L-1)}} \cdots \frac{\partial z^{(2)}}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial W^{(1)}}$$

- The problem with the sigmoid function is that its derivative has a maximum value of 0.25.

- If we employ deep neural networks with sigmoid activation functions, we'll encounter the issue of multiplying numerous small numbers, resulting in the vanishing gradient problem.



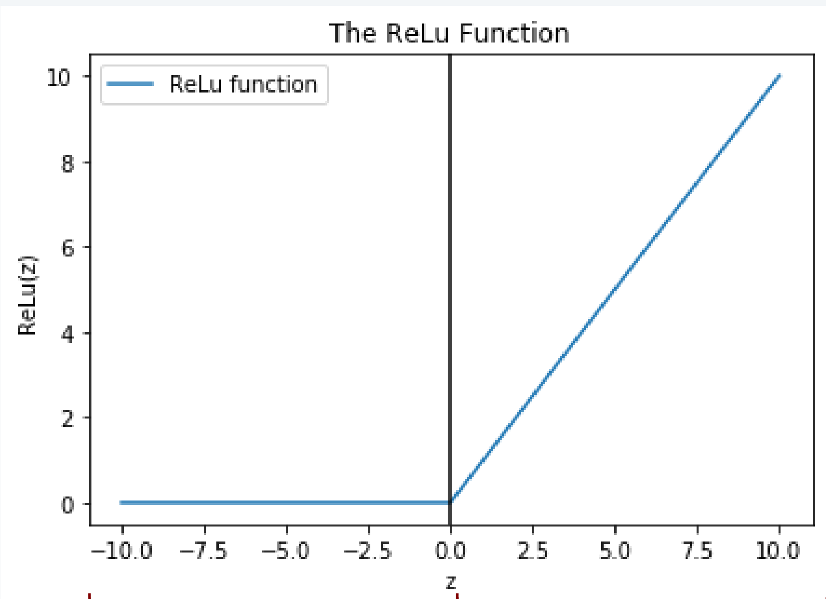
The activation functions – Part 3 –

- A simple solution to the vanishing gradient problem is to use the following activations functions:

$$\text{ReLu} : z \mapsto \max(0, z)$$

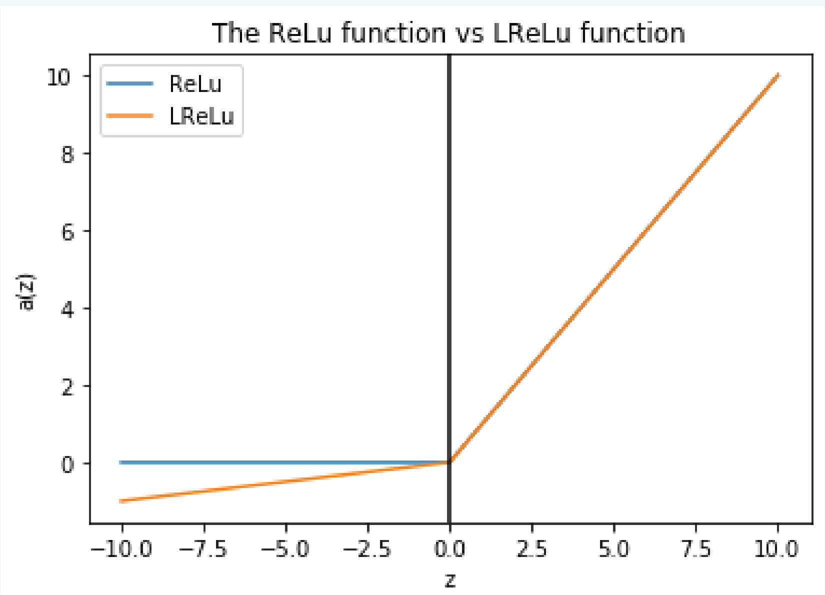
$$\text{Leaky Relu} : z \mapsto \begin{cases} z & \text{if} \quad z \neq 0 \\ \alpha z & \text{if} \quad z < 0 \end{cases}$$

$$\text{Softplus} : z \mapsto \log(1 + \exp(z))$$

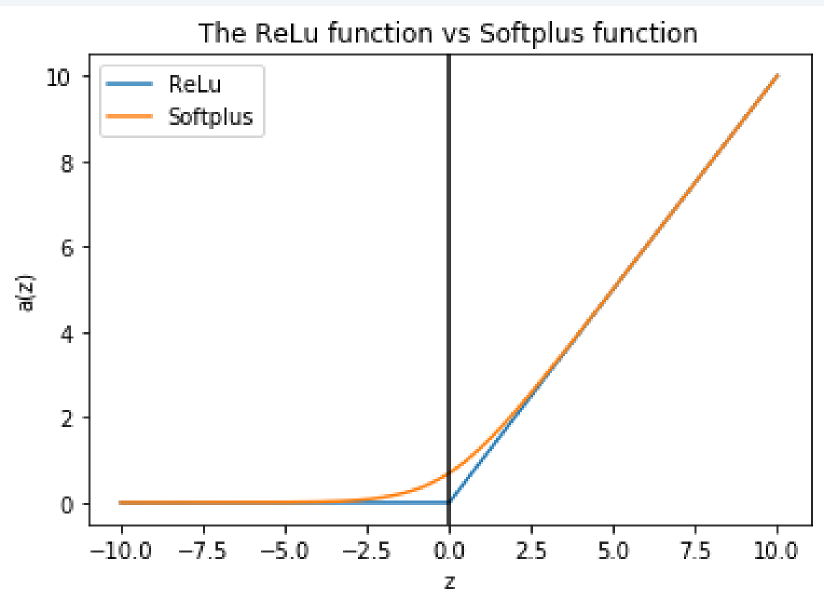


the gradient in the left side is vanished

The right side is not vanishing



Fixing the left side of the ReLu



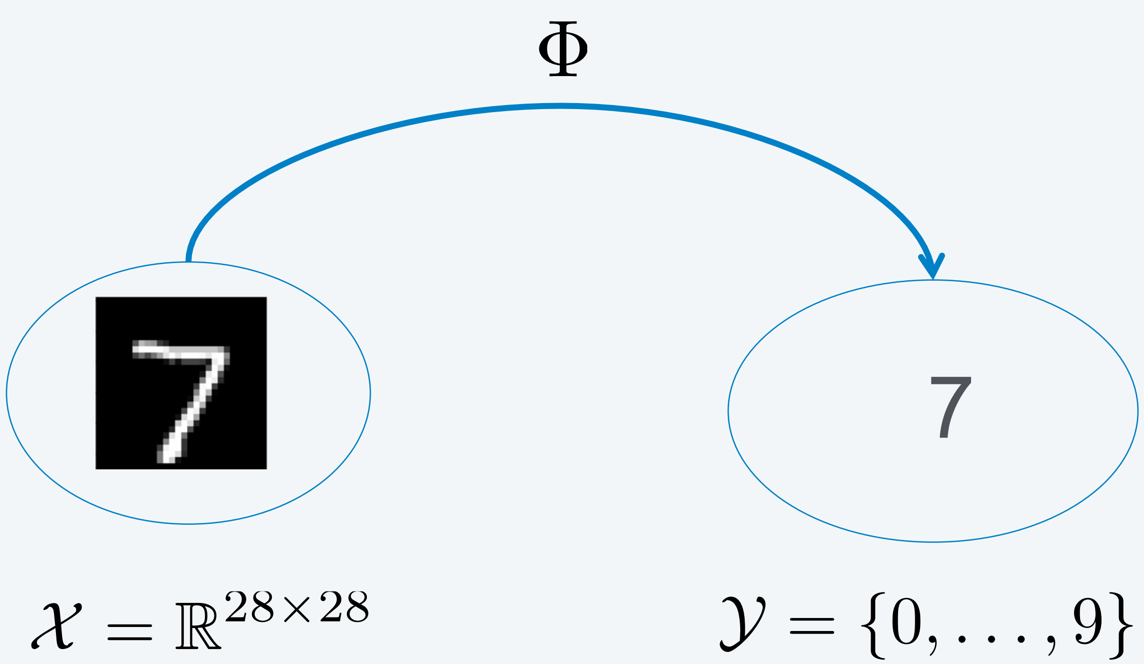
Looks linear on the right side

- Aside from selecting the appropriate activation function for the final layer based on the problem at hand (we'll explore later why sigmoid is suitable for binary classification, softmax for multiclass classification, and no activation function for regression), it's crucial to compare various activation functions for the remaining layers, just as we would with any other hyperparameter.

Universal Approximation Theory

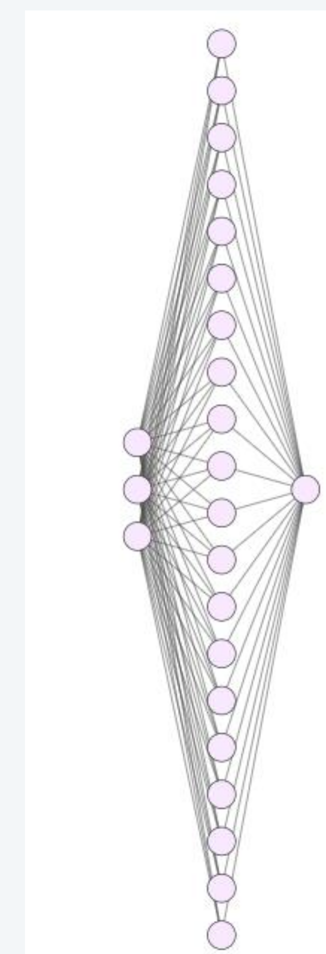
- The Universal Approximation Theory demonstrates that any continuous function Φ can be closely approximated by a shallow neural network under certain regularity conditions. However, achieving this may necessitate exponentially increasing the number of neurons relative to the problem's dimension.

- Reference : [Pinkus 1999 : Approximation theory of the MLP model in neural networks]



The classification problem is of dimension:
d = 784

Theoratically, it should require exp(d) neurons in the hidden layer

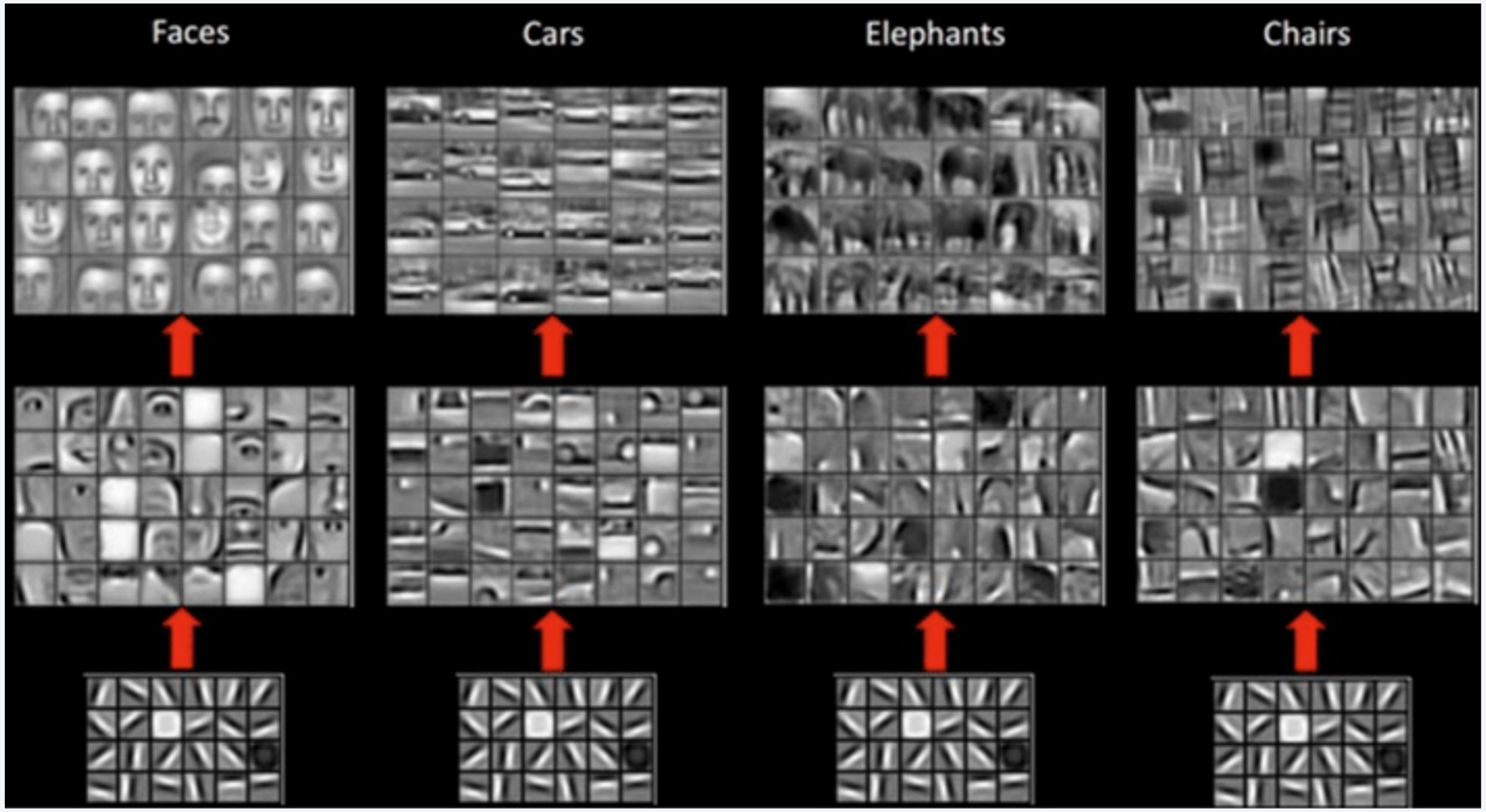


- While Multilayer Networks are less theoretically understood compared to Shallow Networks, in practice, it's been observed that deeper networks outperform shallow ones for a given number of parameters.

Part 2 : Deep Neural Networks and Loss functions

Why deeper Networks ?

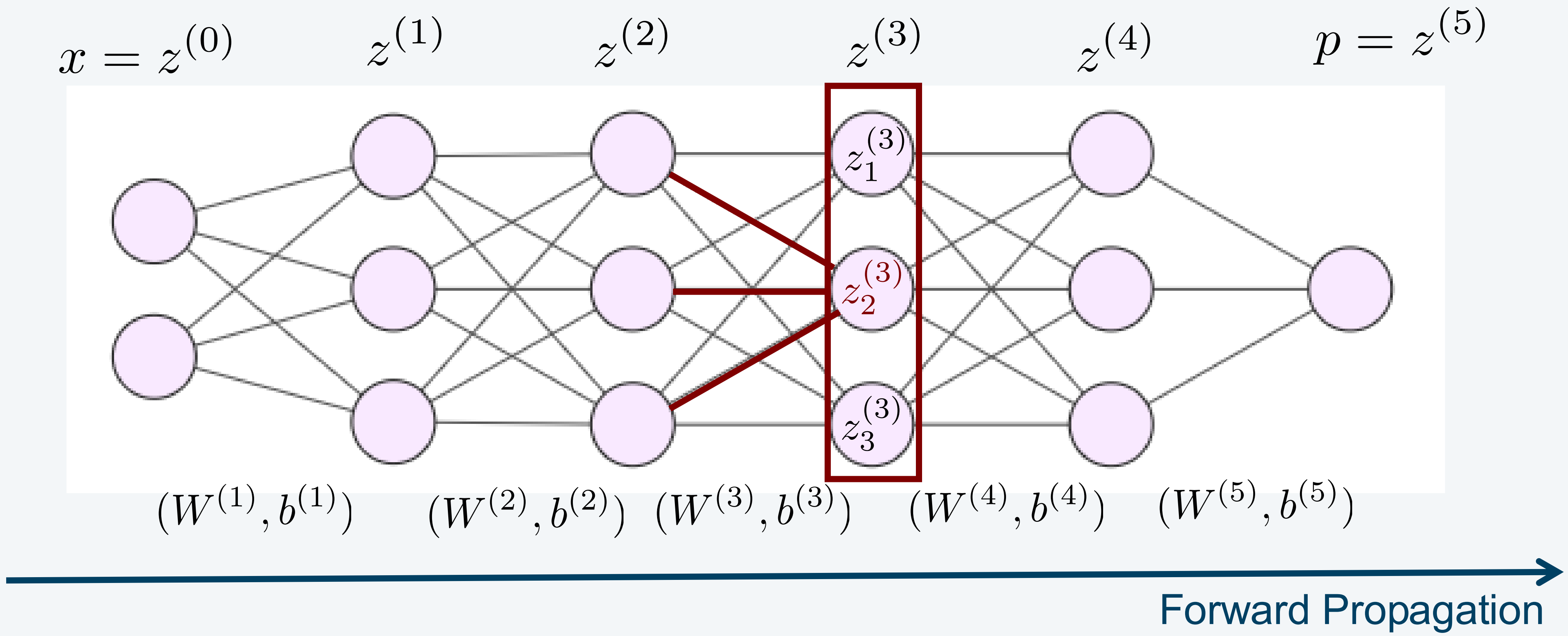
- Each neural network layer is a feature transformation.
- Deeper layers learn increasingly complex features.
- We've covered a particular type of layer known as **Dense Layers**. In future lectures, we'll explore additional types of transformations. For example, Convolutional layers are commonly employed for processing images.



- The first convolution layer will learn small local patterns such as edges.
- Deeper layers will learn larger patterns made of the features of the previous layers, and so on.

Forward Propagation for Binary Classification – Part 1 –

- Let's keep the example of binary classification, but with a deeper model (incorporating multiple layers and multiple neurons per layer).



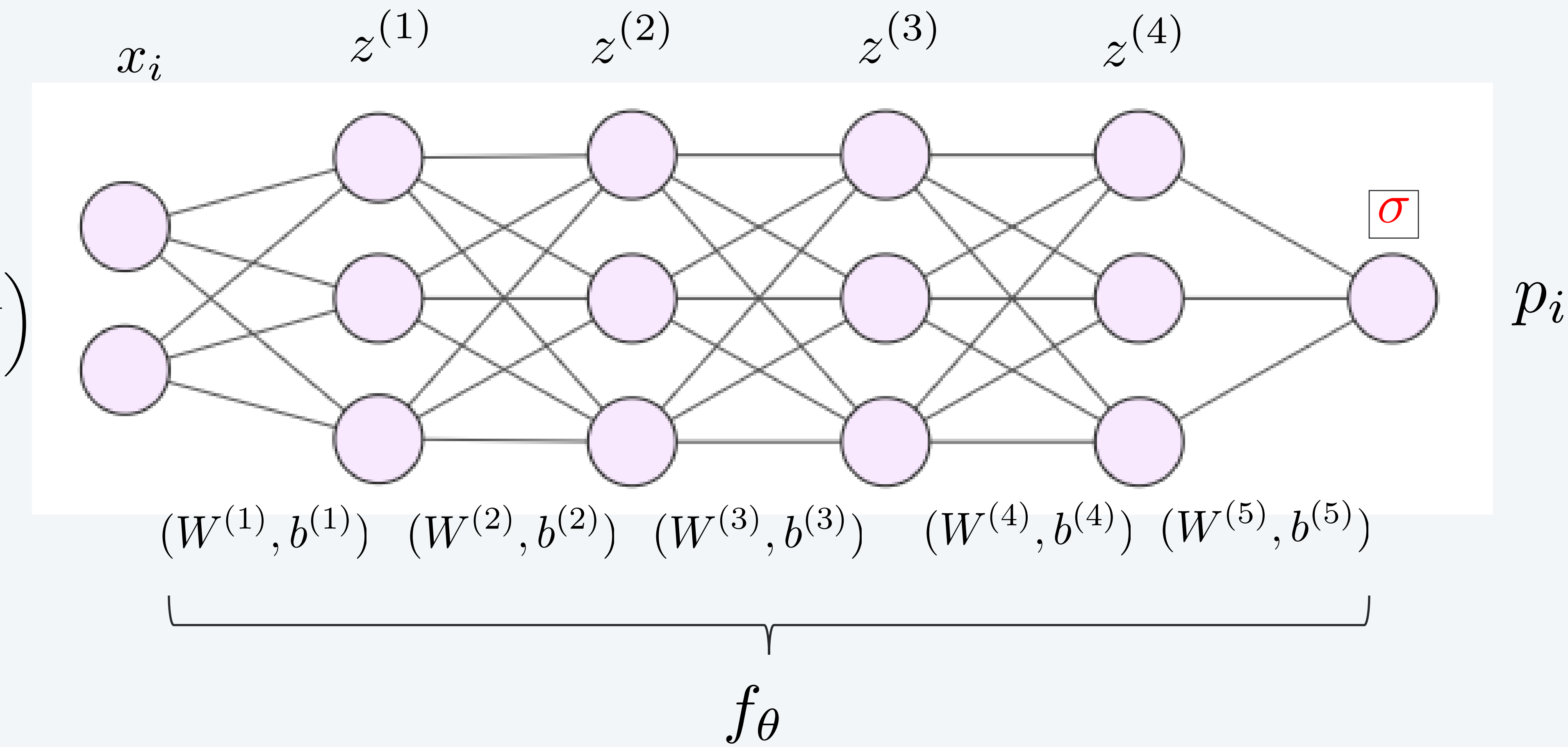
- On a neuron level : $\boxed{z_2^{(3)} = \sigma(W_{12}^{(3)} z_1^{(2)} + W_{22}^{(3)} z_2^{(2)} + W_{32}^{(3)} z_3^{(2)} + b^{(3)})}$

- On a layer level:

$$
\left.\begin{array}{l}
{W^{(3)}}^T = \begin{pmatrix} W_{11}^{(3)} & W_{21}^{(3)} & W_{31}^{(3)} \\ W_{12}^{(3)} & W_{22}^{(3)} & W_{32}^{(3)} \\ W_{13}^{(3)} & W_{23}^{(3)} & W_{33}^{(3)} \end{pmatrix} \qquad \begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix} = z^{(2)}
\end{array}\right\} \Rightarrow \boxed{\begin{array}{c} \forall l \in \{1, \ldots, 5\} \\ z^{(l)} = \sigma({W^{(l)}}^T z^{(l-1)} + b^{(l)}) \end{array}}
$$

Forward Propagation for Binary Classification – Part 2 –

$x_i = \begin{pmatrix} \text{length} \\ \text{width} \end{pmatrix}$



- In the previous example, the forward propagation can be summerized as follows:

$$\forall i \in \{1, \ldots, N\} \quad p_i = f_\theta(x_i) \quad \text{where} \quad \theta = \left\{(W^{(i)}, b^{(i)}); i \in \{1, \ldots, 5\}\right\}$$

- The forward propagation outputs the probability of the positive class: $p_i = \mathbb{P}(Y = 1 | X = x_i)$

- Since forward propagation produces a probability within the range $[0, 1]$ of the final activation function should be a **sigmoid function**.

Loss function for Binary Classification – Part 1 –

$$\Rightarrow x_1 = \begin{pmatrix}\text{length}\\ \text{width}\end{pmatrix} \Rightarrow \boxed{f_\theta} \Rightarrow p_1 = f_\theta(x_1) = \mathbb{P}(Y=1|X=x_1)$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$\Rightarrow x_i = \begin{pmatrix}\text{length}\\ \text{width}\end{pmatrix} \Rightarrow \boxed{f_\theta} \Rightarrow p_i = f_\theta(x_i) = \mathbb{P}(Y=1|X=x_i)$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$\Rightarrow x_N = \begin{pmatrix}\text{length}\\ \text{width}\end{pmatrix} \Rightarrow \boxed{f_\theta} \Rightarrow p_N = f_\theta(x_N) = \mathbb{P}(Y=1|X=x_N)$$

- Given the dataset $(x_i, y_i)_{1\leq i\leq N}$ and the model: $Y|X=x_i \sim \mathcal{B}(p_i)$, where $\mathcal{B}$ represents the Bernoulli distribution, our goal is to maximize the following likelihood :

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \mathbb{P}(Y=y_i|X=x_i) = \prod_{i=1}^{N} f_\theta(x_i)^{y_i}(1-f_\theta(x_i))^{1-y_i}$$

Loss function for Binary Classification – Part 2 –

- As usual, instead of maximizing the likelihood, we prefer to minimize the normalized negative log-likelihood known as the loss function J :

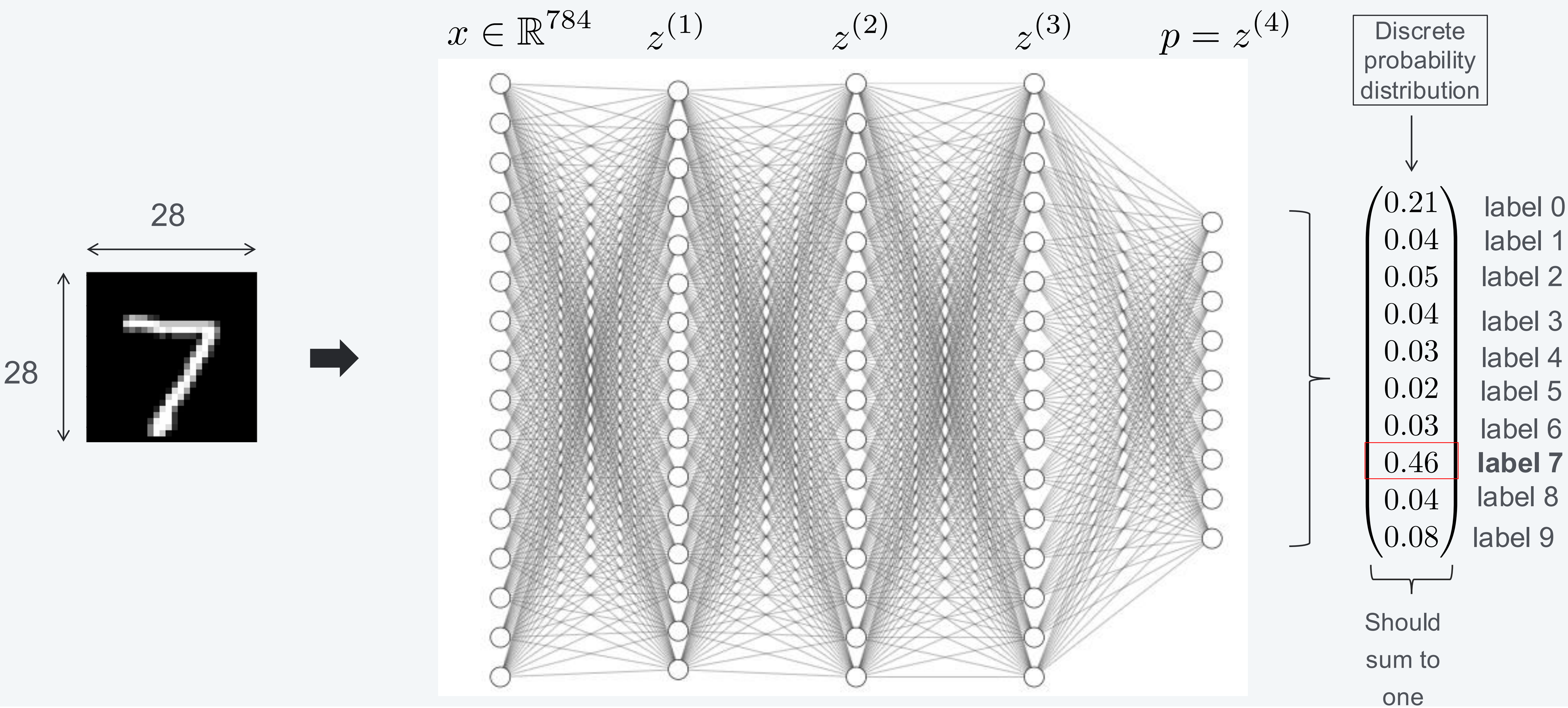
$$
\begin{aligned}
J(\theta) &= -\frac{1}{N}\log(\mathcal{L}(\theta)) \\
&= -\frac{1}{N}\log\left(\prod_{i=1}^{N} f_\theta(x_i)^{y_i}(1-f_\theta(x_i))^{1-y_i}\right) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\log\left(f_\theta(x_i)^{y_i}(1-f_\theta(x_i))^{1-y_i}\right) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\{y_i\log(f_\theta(x_i)) + (1-y_i)\log(1-f_\theta(x_i))\}
\end{aligned}
$$

- Therefore, the loss function for binary classification is the **binary cross-entropy**:

$$\boxed{J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i\log(f_\theta(x_i)) + (1-y_i)\log(1-f_\theta(x_i))\}}$$

Forward Propagation for Multiclass Classification – Part 1 –

- The multiclass classification consists in predicting one of K categories.
- Let's take the example of the MNIST dataset. The objective is to predict a category among the set of numbers $\{0, 1, \ldots, 9\}$ based on an image of shape (28, 28) that we can flatten into a 784 dimensional input vector.



Forward Propagation for Multiclass Classification – Part 2 –

- If we want the output to represent a discrete distribution over all the possible categories (10 in our example), the last layer must contain 10 neurons and the activation function must be the **softmax activation function**.

- The softmax activation converts a vector of size K into a probability distribution. It achieves this by applying the exponential function to convert real numbers into positive values, followed by normalization.

$$\begin{pmatrix} w_1 \\ \vdots \\ w_i \\ \vdots \\ w_K \end{pmatrix} \xrightarrow{\text{Applying the Softmax activation function}} \begin{pmatrix} p_1 \\ \vdots \\ p_i \\ \vdots \\ p_K \end{pmatrix}$$

$$\forall i \in \{1, \ldots, K\} \quad p_i = \frac{e^{w_i}}{\sum_{j=1} e^{w_j}}$$

$$\Downarrow$$

Output of the last layer before applying the activation function

$$\forall i \in \{1, \ldots, K\} \quad p_i \geq 0$$

$$\sum_{i=1}^{K} p_i = 1$$

Output of the last layer after applying the activation function

Categorical Distribution – One hot encoding –

- As we have seen before, the categorical distribution (also called multinomial distribution) models the outcome of a random variable that can take K possible categories.

- Let X be a random variable than can take K possible values, each value k with probability π_k

$$\forall k \in \{1, \ldots, K\} \quad \mathbb{P}(X = k) = \pi_k$$

- We say that X follows a Multinomial distribution:

$$X \sim \mathcal{M}(1, \pi_1, \ldots, \pi_K) \quad \text{where} \quad \forall k \in \{1, \ldots K\} \quad \pi_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} \pi_k$$

- We usually use **one hot encoding** to represent the discrete random variable X, which consists in encoding X with a random variable $Y = (Y_1, \ldots, Y_K)^T$ such that

$$\forall k \in \{1, \ldots, K\} \quad Y_k = 1_{\{X=k\}}$$

- Which means:

$$\{X = k\} \Leftrightarrow Y = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} \leftarrow \boxed{\text{k-th position}}$$

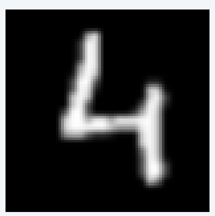
- Thus:

$$\forall k \in \{1, \ldots, K\} \quad \mathbb{P}(X = k) = \mathbb{P}(Y_k = 1)$$

Loss function for Multiclass Classification – Part 1 –

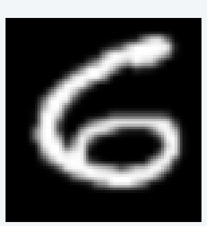
$$x_1 \in \mathbb{R}^{784} \rightarrow \boxed{f_\theta} \rightarrow p_1 = f_\theta(x_1) = \begin{pmatrix} p_1^1 \\ \vdots \\ p_1^k \\ \vdots \\ p_1^K \end{pmatrix} = \begin{pmatrix} \mathbb{P}(Y_1 = 1 | X = x_1) \\ \vdots \\ \mathbb{P}(Y_k = 1 | X = x_1) \\ \vdots \\ \mathbb{P}(Y_K = 1 | X = x_1) \end{pmatrix}$$

$$\vdots$$



$$x_i \in \mathbb{R}^{784} \rightarrow \boxed{f_\theta} \rightarrow p_i = f_\theta(x_i) = \begin{pmatrix} p_i^1 \\ \vdots \\ p_i^k \\ \vdots \\ p_i^K \end{pmatrix} = \begin{pmatrix} \mathbb{P}(Y_1 = 1 | X = x_i) \\ \vdots \\ \mathbb{P}(Y_k = 1 | X = x_i) \\ \vdots \\ \mathbb{P}(Y_K = 1 | X = x_i) \end{pmatrix}$$

$$\vdots$$



$$x_N \in \mathbb{R}^{784} \rightarrow \boxed{f_\theta} \rightarrow p_N = f_\theta(x_N) = \begin{pmatrix} p_N^1 \\ \vdots \\ p_N^k \\ \vdots \\ p_N^K \end{pmatrix} = \begin{pmatrix} \mathbb{P}(Y_1 = 1 | X = x_N) \\ \vdots \\ \mathbb{P}(Y_k = 1 | X = x_N) \\ \vdots \\ \mathbb{P}(Y_K = 1 | X = x_N) \end{pmatrix}$$

Loss function for Multiclass Classification – Part 2 –

- The targets should also be encoded using one-hot encoding.

$$[y_i]_{1\leq i\leq N} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 6 \\ 9 \\ 0 \end{pmatrix} \Rightarrow [\hat{y}_i^k]_{\substack{1\leq i\leq N \\ 1\leq k\leq K}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- Which means: $\forall i \in \{1,\ldots,N\}\ \forall k \in \{1,\ldots,K\} \quad y_i = k \iff \hat{y}_i^k = 1$

- Given the dataset $(x_i, y_i)_{1\leq i\leq N}$ and the model: $Y|X = x_i \sim \mathcal{M}(1, p_i^1, \ldots, p_i^K)$ where $\mathcal{M}$ stands for the Multinomial distribution, our objective is to maximize the following likelihood :

$$\mathcal{L}(\theta) = \prod_{i=1}^{N} \mathbb{P}(Y = y_i | X = x_i) = \prod_{i=1}^{N} \boxed{\prod_{k=1}^{K} {p_i^k}^{\hat{y}_i^k}} \longrightarrow \prod_{k=1}^{K} {p_i^k}^{\hat{y}_i^k} = p_i^l \iff \hat{y}_i^l = 1$$

Loss function for Multiclass Classification – Part 3 –

- As before, instead of maximizing the likelihood, we prefer to minimize the normalized negative log-likelihood known as the loss function J:

$$
\begin{aligned}
J(\theta) &= -\frac{1}{N}\log(\mathcal{L}(\theta)) \\
&= -\frac{1}{N}\log\left(\prod_{i=1}^{N}\prod_{k=1}^{K} {p_i^k}^{\hat{y}_i^k}\right) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\hat{y}_i^k \log(p_i^k)
\end{aligned}
$$

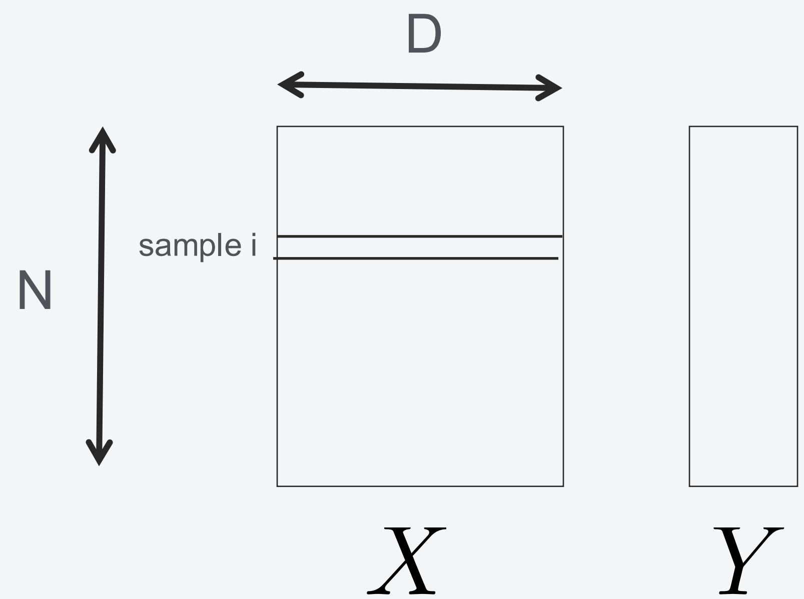
- Therefore, the loss function for multiclass classification is the following **categorical cross-entropy**:

$$
\boxed{J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\hat{y}_i^k \log(p_i^k)}
$$

Part 4 : Deep Learning Techniques

Optimization Techniques: - Batch Gradient Descent -

- Data:



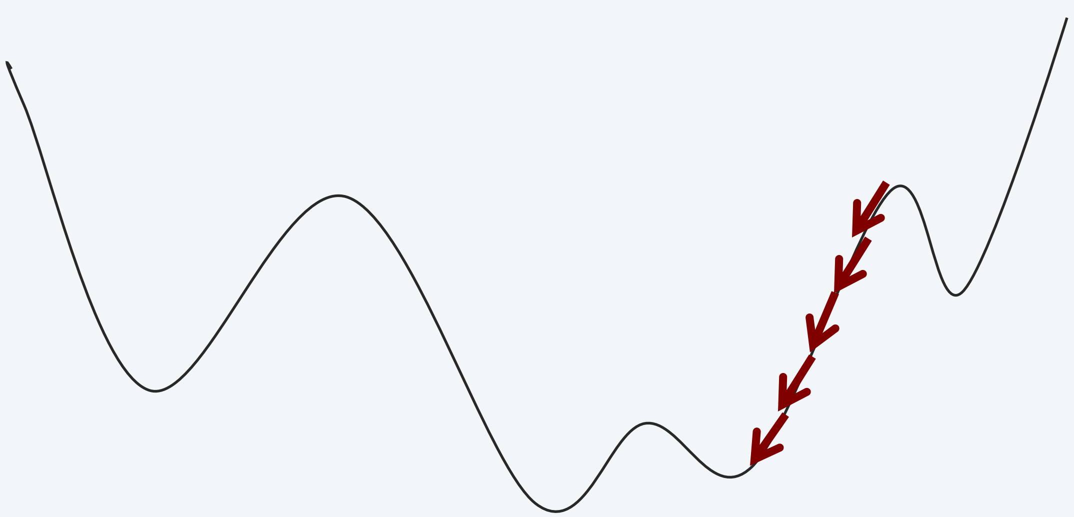
- Parameters: $$\theta = \left\{(W^{(k)}, b^{(k)}); k \in \{1, \ldots, K\}\right\}$$

- Loss function for the dataset: $$\underbrace{J_{\text{dataset}}(\theta)}_{\text{loss of the dataset}} = \frac{1}{N} \sum_{i=1}^{N} \underbrace{J(\theta, i)}_{\text{loss for sample i}}$$

- Algorithm:

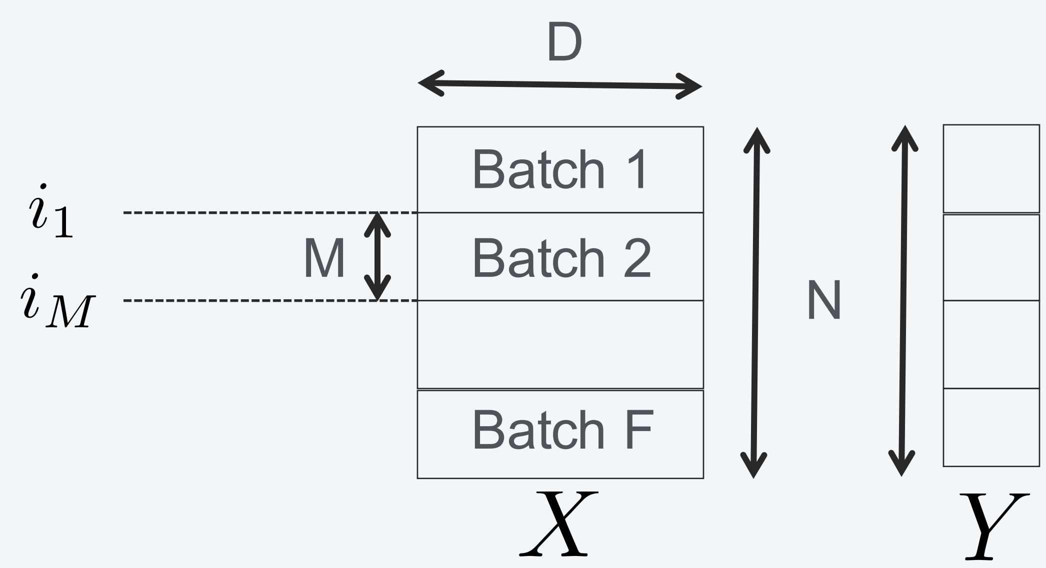
  - Initilize randomly θ_0
  - Fix a number of iterations N_{iter} and a learning rate η and repeat:

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta J_{\text{dataset}}(\theta_k)$$



Optimization Techniques: - Mini Batch Gradient Descent -

- Data:



Number of batches
$F = \text{int}(N/M)$

- Parameters:

$$\theta = \left\{ (W^{(k)}, b^{(k)}); k \in \{1, \dots, K\} \right\}$$

- Loss function for a (small) batch:

$$\underbrace{J_{\text{batch}}(\theta)}_{\text{loss of the batch}} = \frac{1}{M} \sum_{i=i_1}^{i_M} \underbrace{J(\theta, i)}_{\text{loss for sample i}}$$

- Algorithm:

  - Initilize randomly θ_0
  - Repeat N_{epochs} times:
    - Shuffle the data and split it into batches of size M.
    - Update the weights for each batch in $\{1, \dots, F\}$

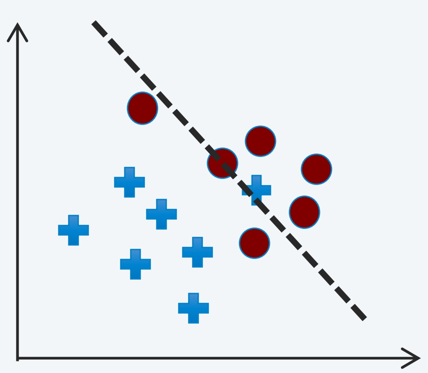
$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta J_{\text{batch}}(\theta_k)$$

Fighting the Overfitting problem – 1 –

- There are a lot of hyperparameters to tune when using Neural Networks:
  - The number of hidden layers, the number of neuros per hidden layer.
  - The activations functions.
  - The number of epochs, the batch size, the learning rate and other hyperparameters for more sophisticated optimization algorithms.

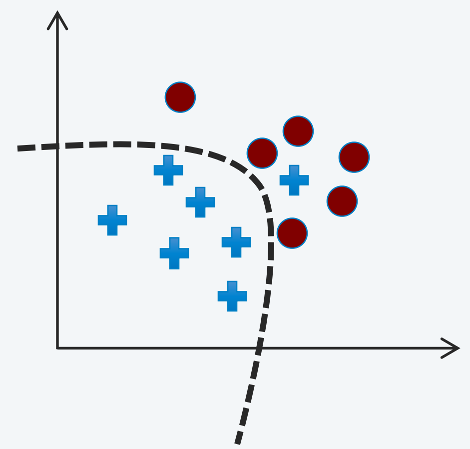
- The primary challenge in designing a neural network architecture, i.e., selecting the hyperparameters, is to ensure a balance between the **optimization** task and the **generalization** objective.

Underfitting



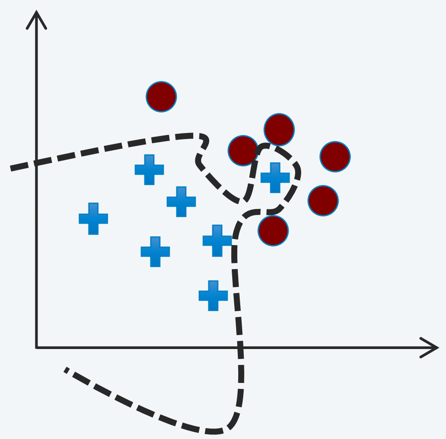
Complexity of $\mathcal{G}$ too small (high bias)

Good Compromise



Complexity of $\mathcal{G}$: neither too big, nor too small.

Overfitting

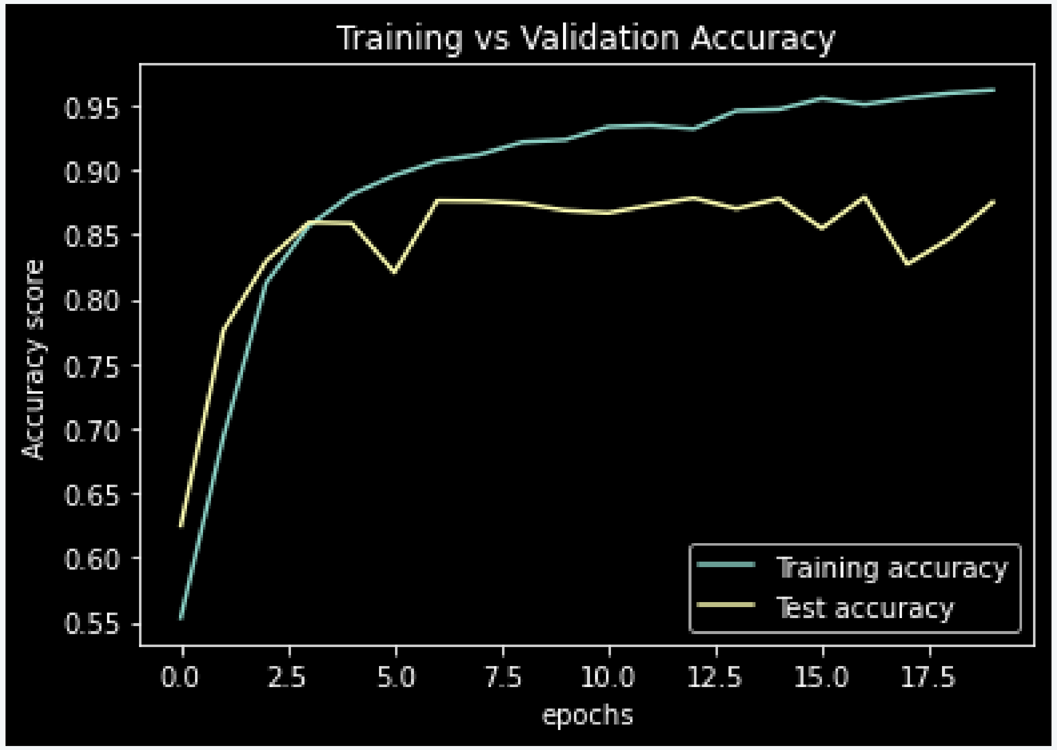


Complexity of $\mathcal{G}$ too big (high variance)

Fighting the Overfitting problem – 2 –

- At the beginning of the training process, **optimization** and **generalization** are correlated. Both the training and the validation metrics are improving.

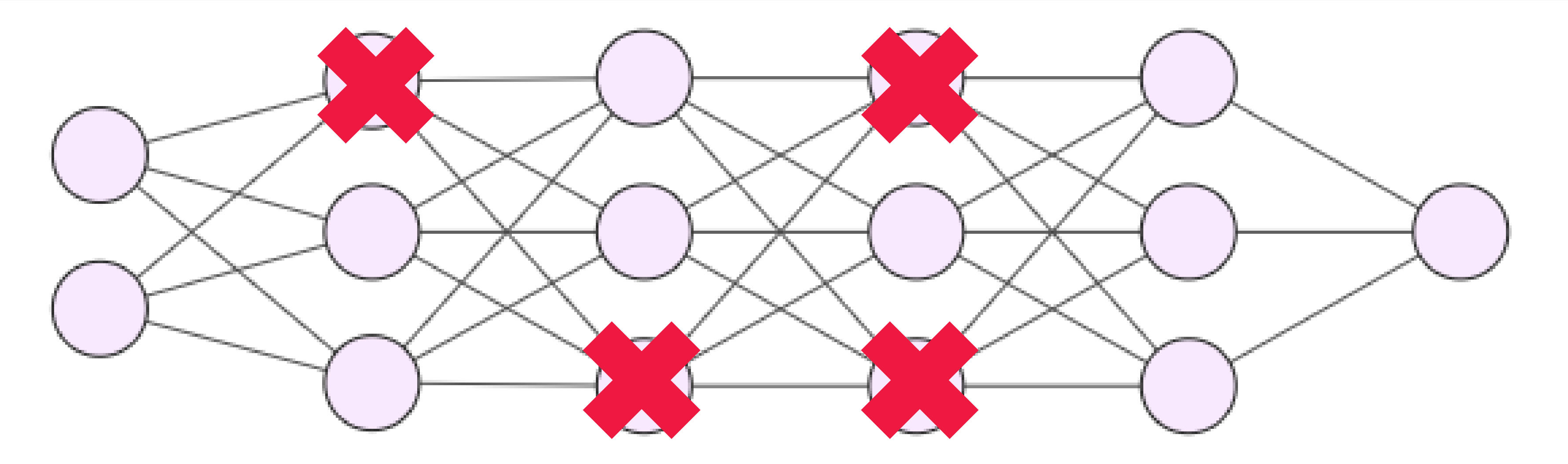
- After several iterations, generalization ceases to improve, and validation metrics begin to degrade. This occurs because the model starts learning patterns specific to the training data that are irrelevant to new data, resulting in **overfitting**.



Fighting the Overfitting problem – 3 –

- To overcome the overfitting problem, we can add more samples or reduce the compexity of the network. We can also test several regularization techniques:

  - **Dropout** applied to a layer involves randomly "dropping out" (i.e., setting to zero) a certain number of output features of the layer during training. The "dropout rate" represents the fraction of features that are being zeroed-out and is typically set between 0.2 and 0.5.



  - **Weight regularization** involves incorporating into the network's loss function a penalty related to having large weights. This penalty can be computed using either the $\mathcal{L}^1$ norm or the $\mathcal{L}^2$ norm.

Programming Session