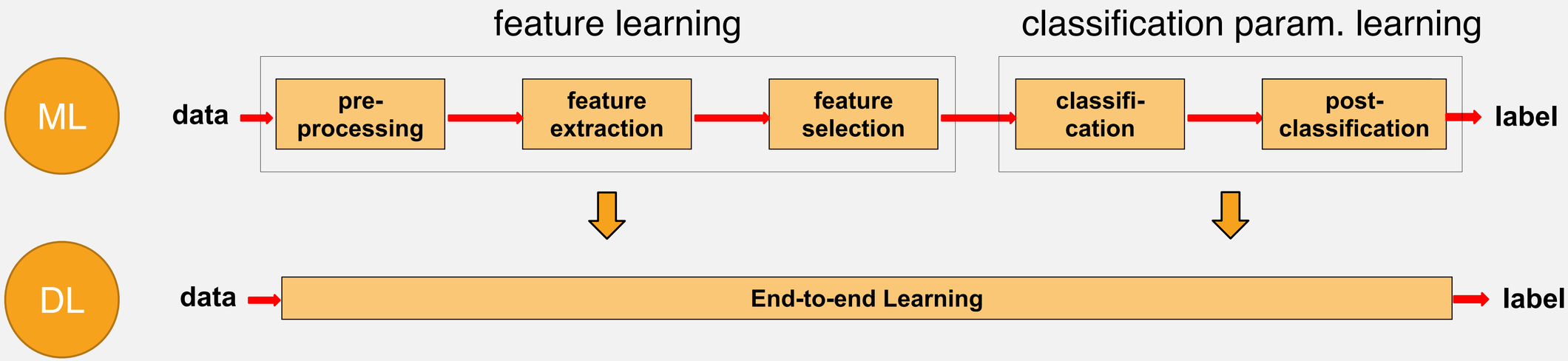
Aprendizado Profundo (Deep Learning)

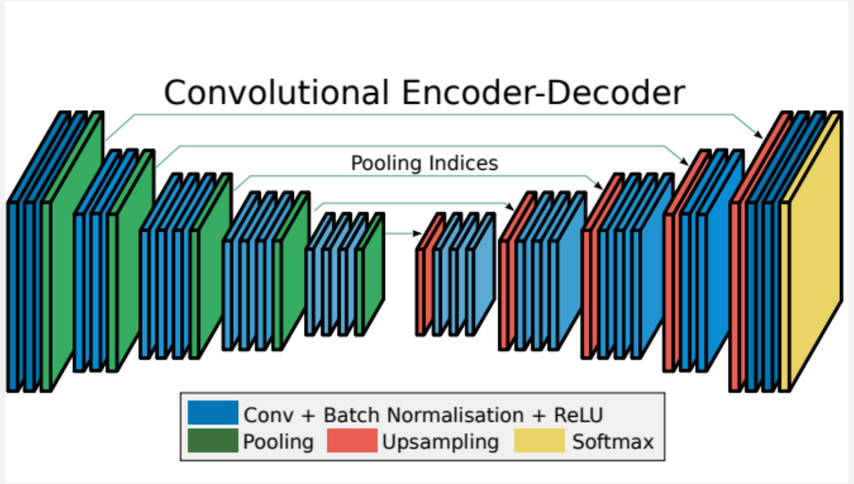
Basics on Neural Networks

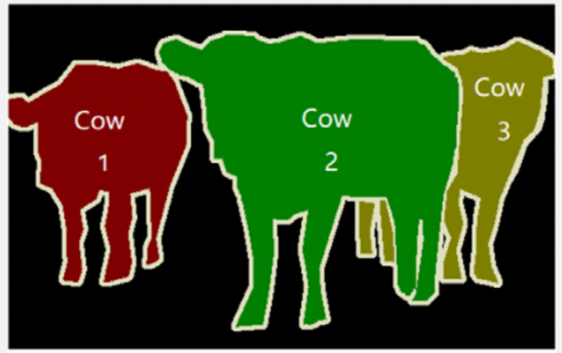
Dario Oliveira (dario.oliveira@fgv.br)

Deep Learning Models



"End-to-end" refers to automatically learning features and classification parameters jointly (vs. step-by-step) straight from the data.



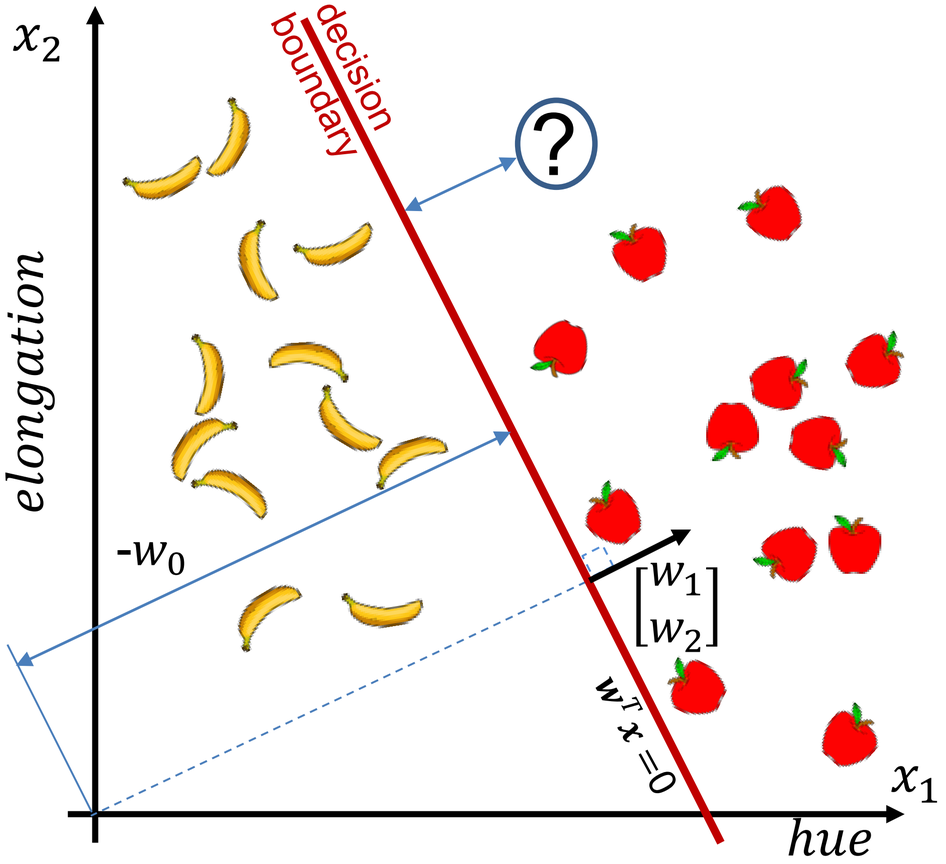


From the beginning

Linear Classifier

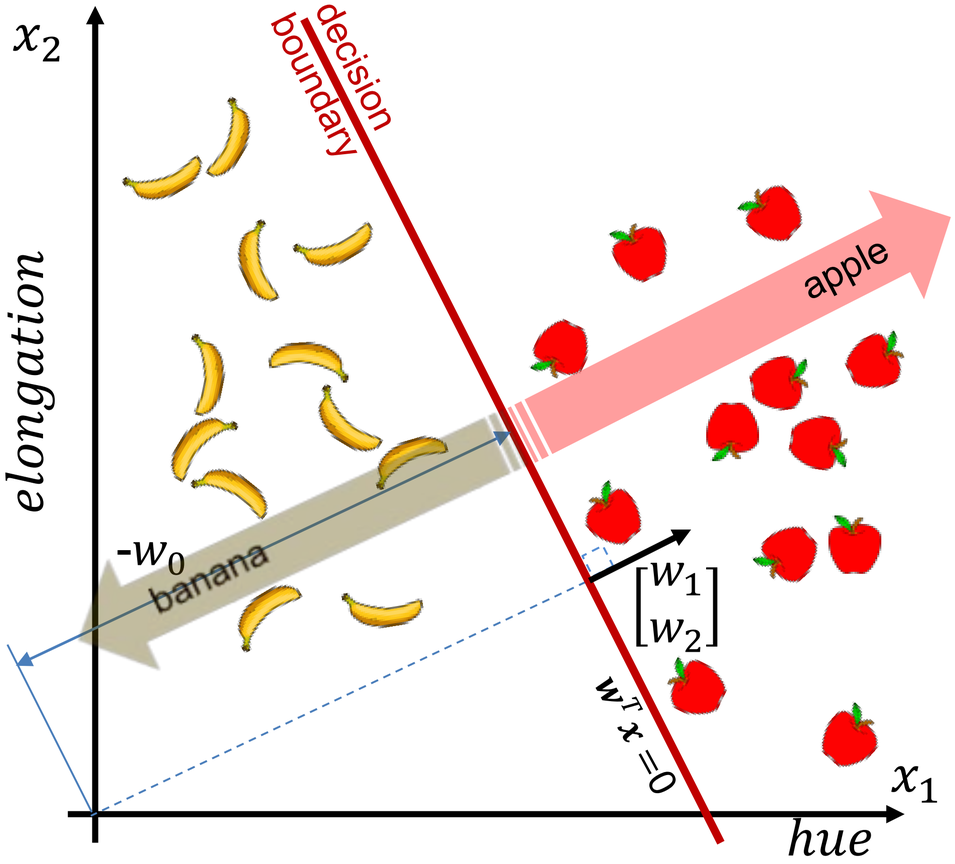
$$\boldsymbol{w}^T\boldsymbol{x} = w_0 + w_1x_1 + w_2x_2$$

distance from the decision boundary



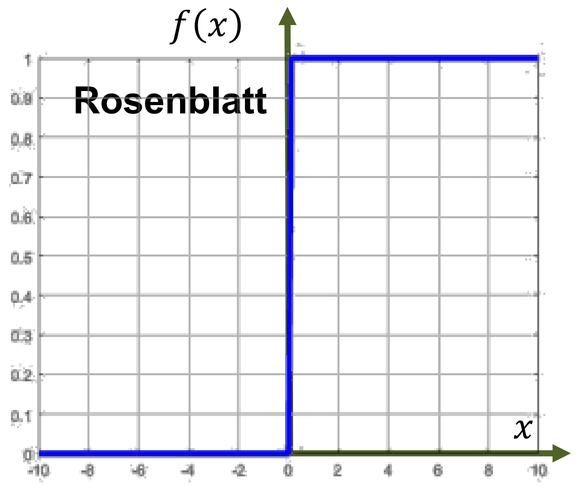
Rosenblatt Perceptron

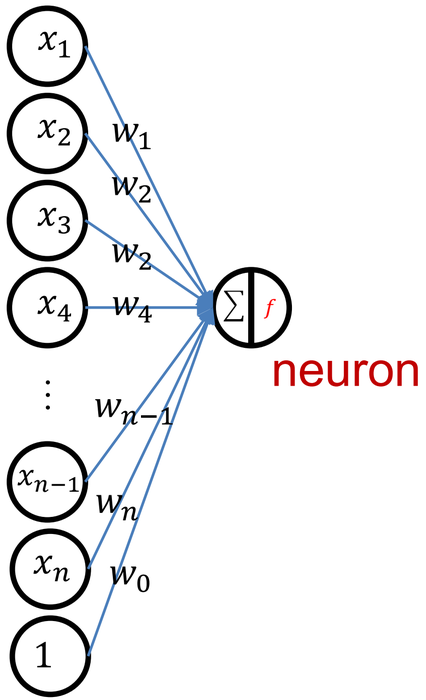
$$f(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} 1 \; if \; \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 \; otherwise \end{cases}$$

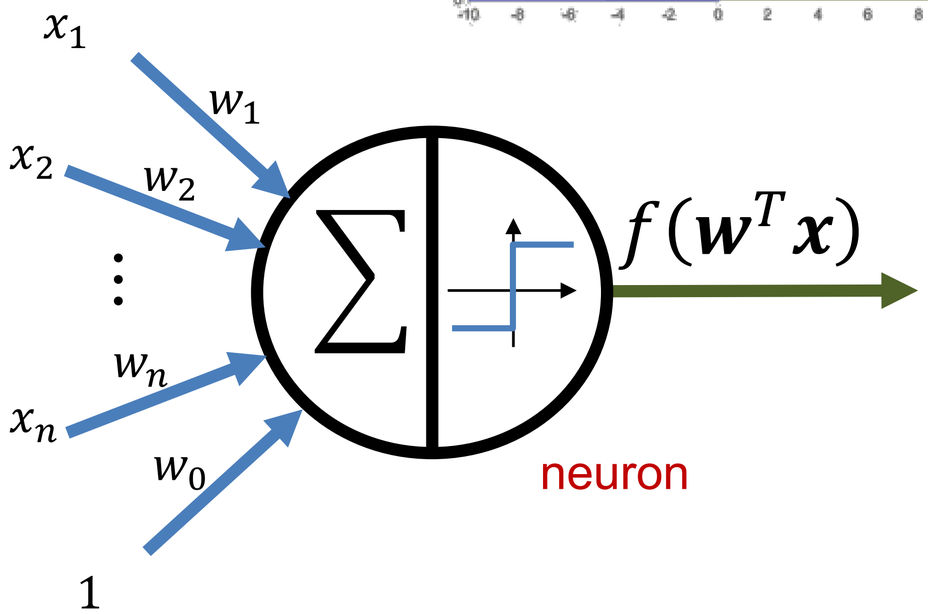


Rosenblatt Perceptron

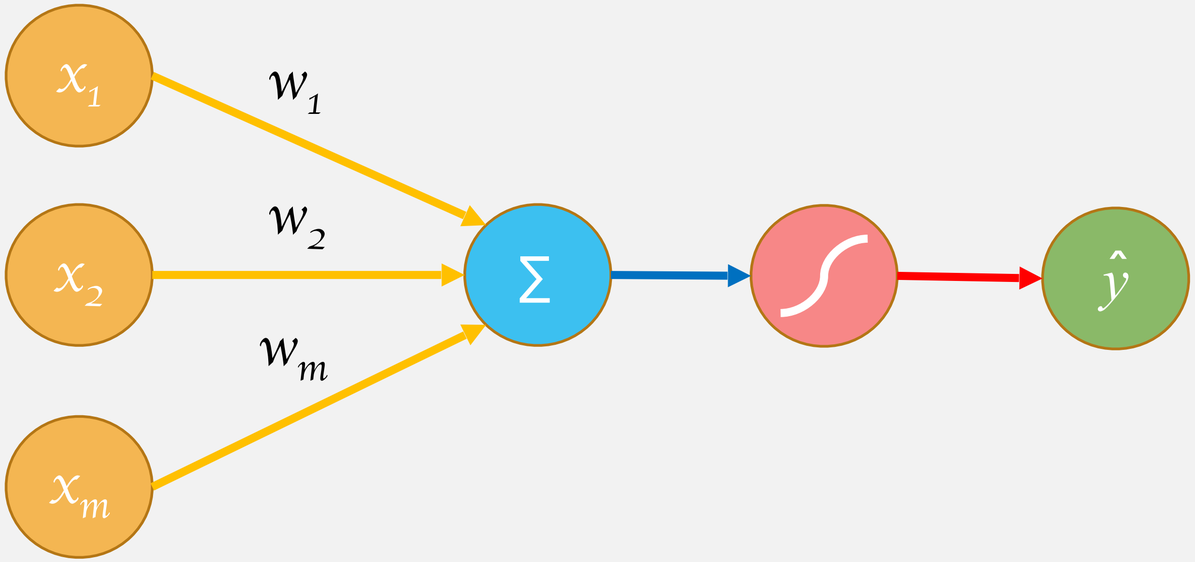
$$f(\boldsymbol{w}^T\boldsymbol{x}) = \begin{cases} 1 \; if \; \boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ 0 \; otherwise \end{cases}$$







Deep Learning Basics: The Perceptron Model



inputs weights sum non-linearity output

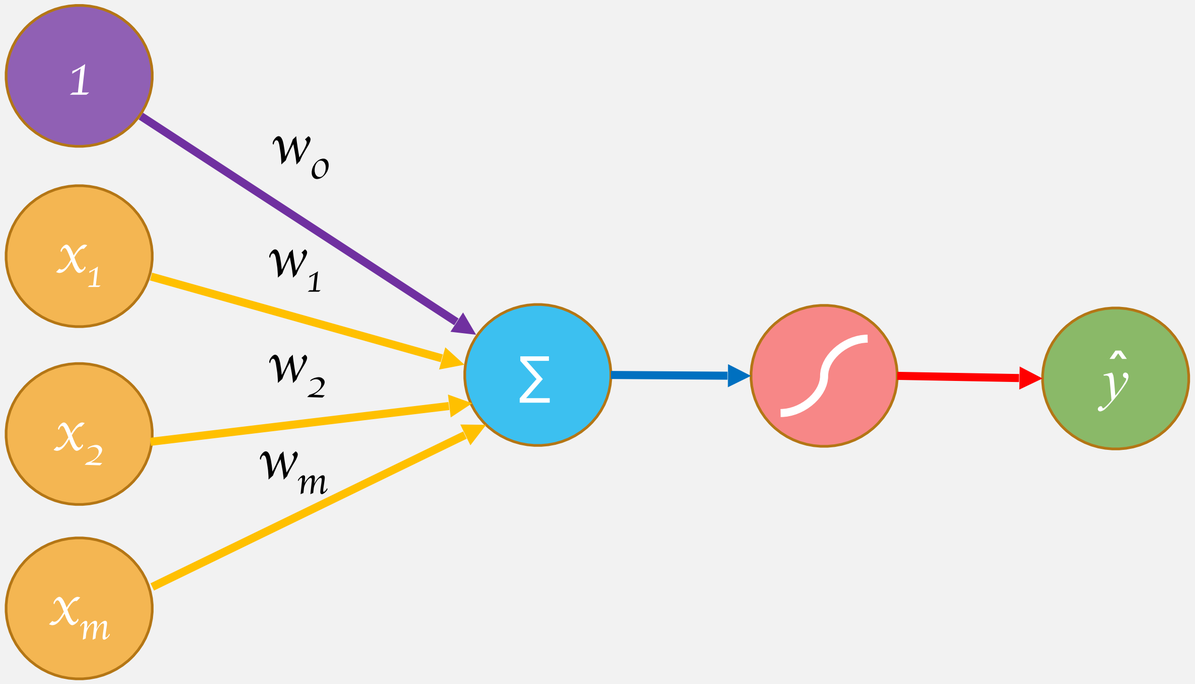
linear combination of inputs

output

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i w_i\right)$$

non-linear activation function

Deep Learning Basics: The Perceptron Model



inputs weights sum non-linearity output

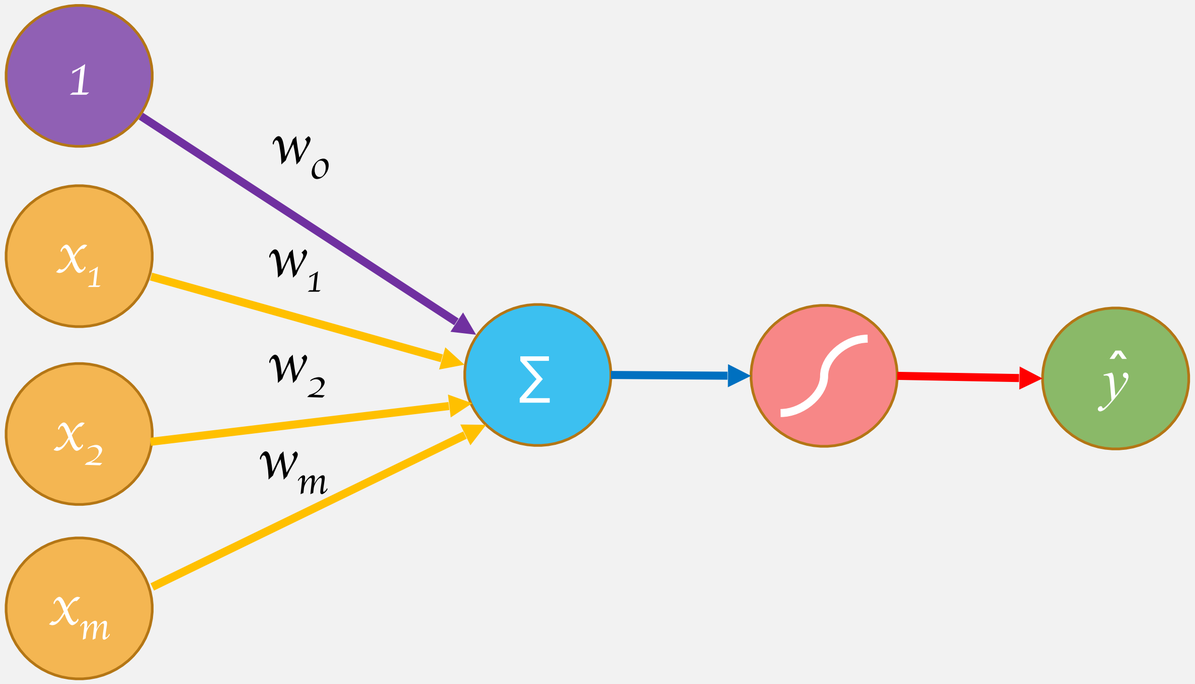
linear combination of inputs

output bias

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

non-linear activation function

Deep Learning Basics: The Perceptron Model



inputs weights sum non-linearity output

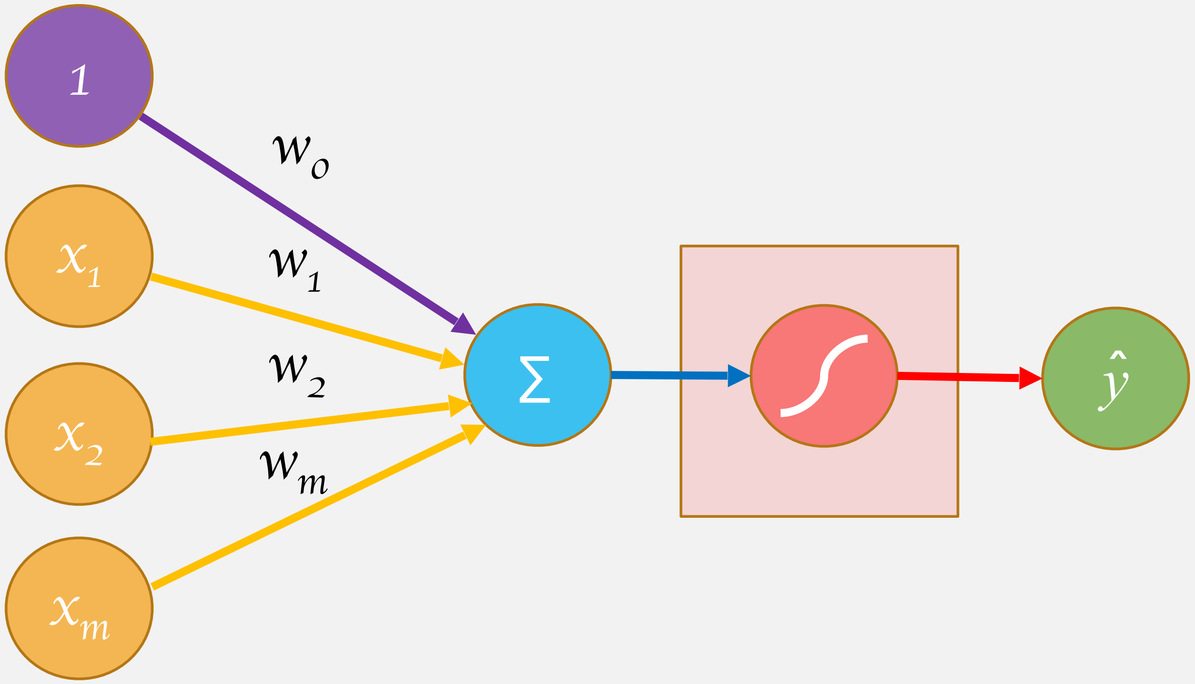
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \cdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \cdots \\ w_m \end{bmatrix}$

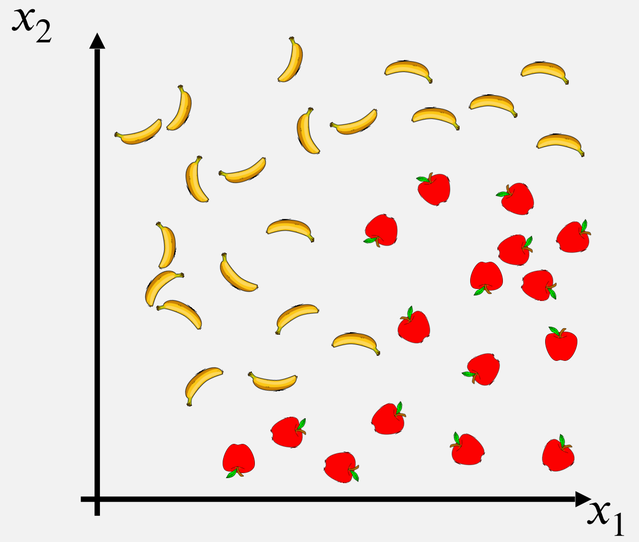
Deep Learning Basics: The Perceptron Model

What is the role of an activation function?



inputs weights sum non-linearity output

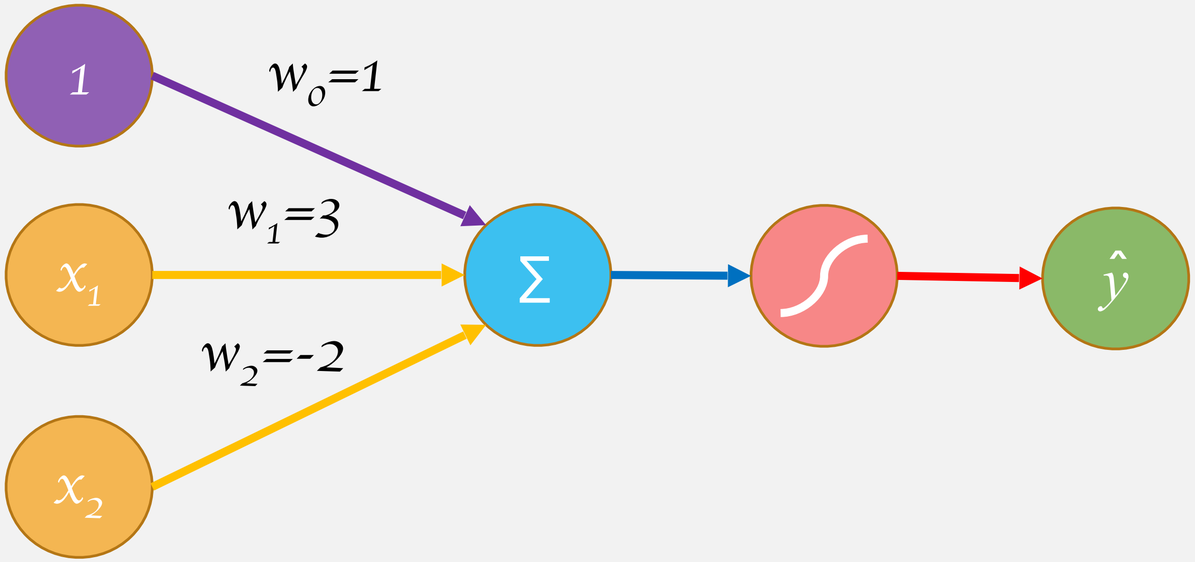
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T\boldsymbol{W}\right)$$



How to define the best boundary to separate apples and bananas?

Deep Learning Basics: The Perceptron Model

How Perceptron works? A simple example



inputs weights sum non-linearity output

$$\hat{y} = g(w_0 + \boldsymbol{X}^T\boldsymbol{W})$$

with $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

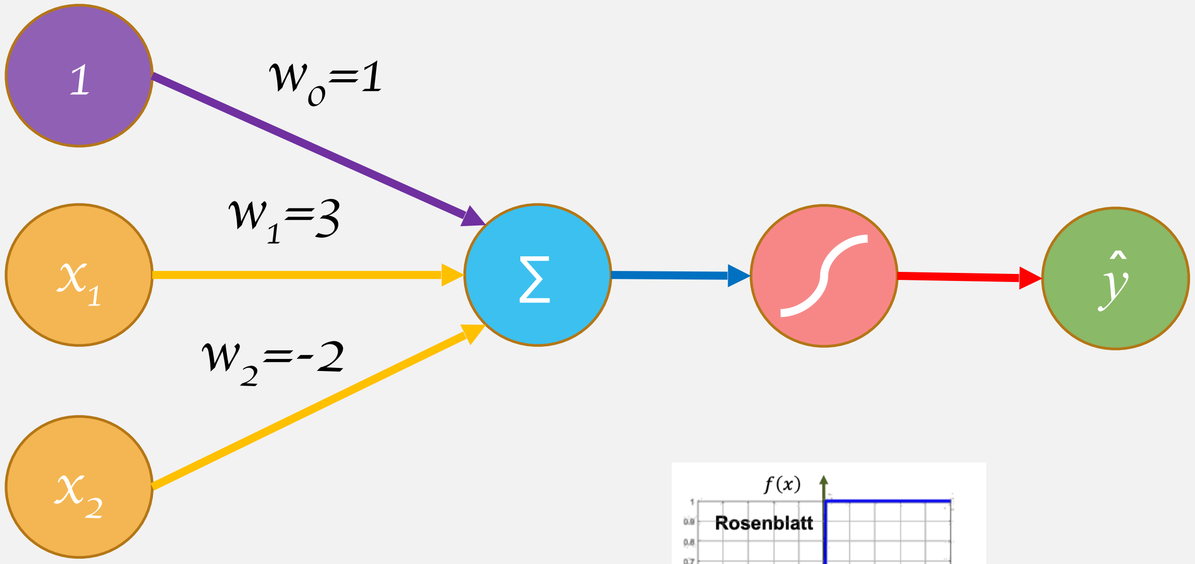
$$\hat{y} = g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

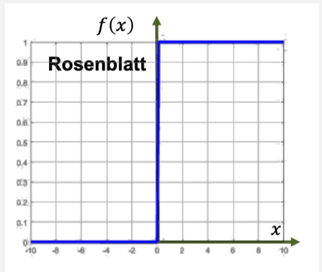
Equation for a 2D line!

Deep Learning Basics: The Perceptron Model

How Perceptron works? A simple example

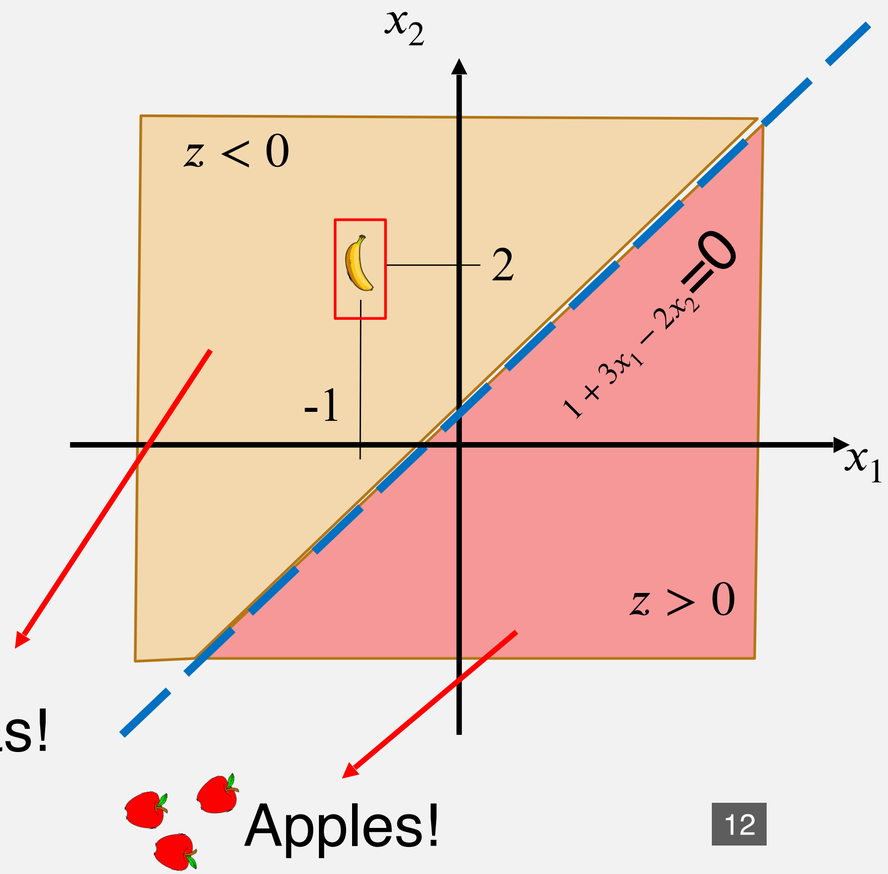


inputs weights sum output



$$z$$

$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$



Bananas!

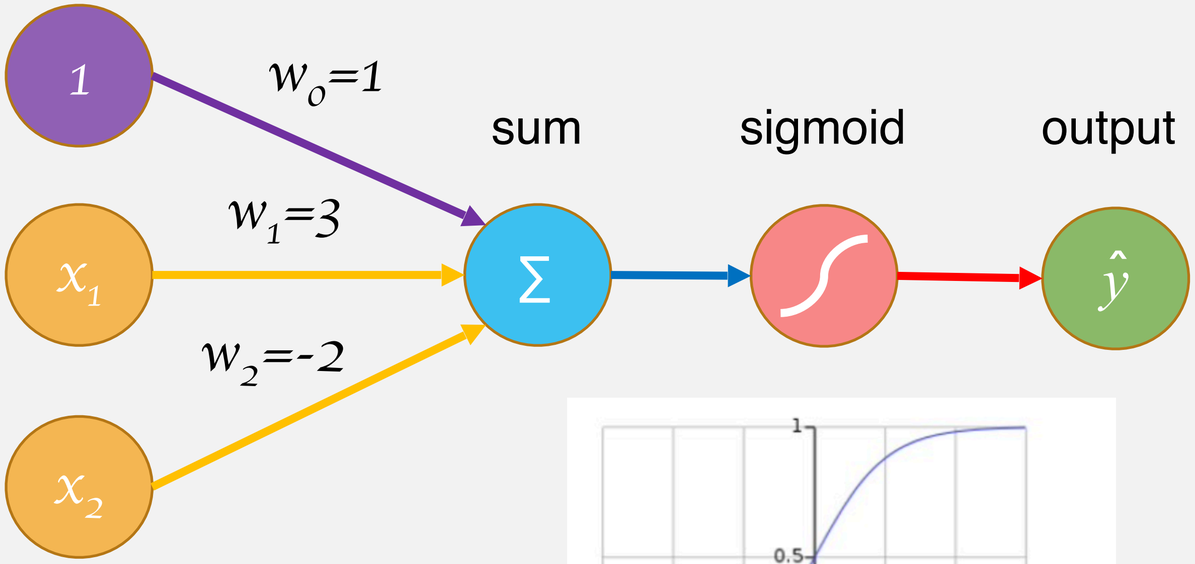
Apples!

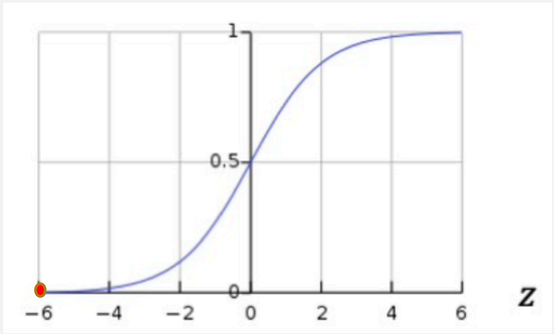
Deep Learning Basics: The Perceptron Model

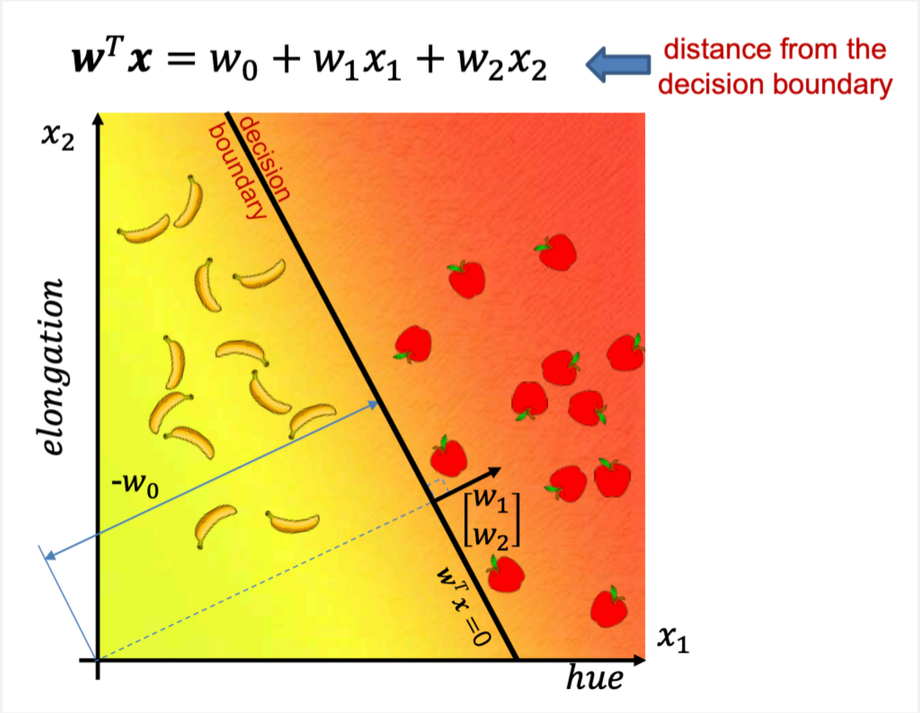
How Perceptron works? A simple example

$$z$$

$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$

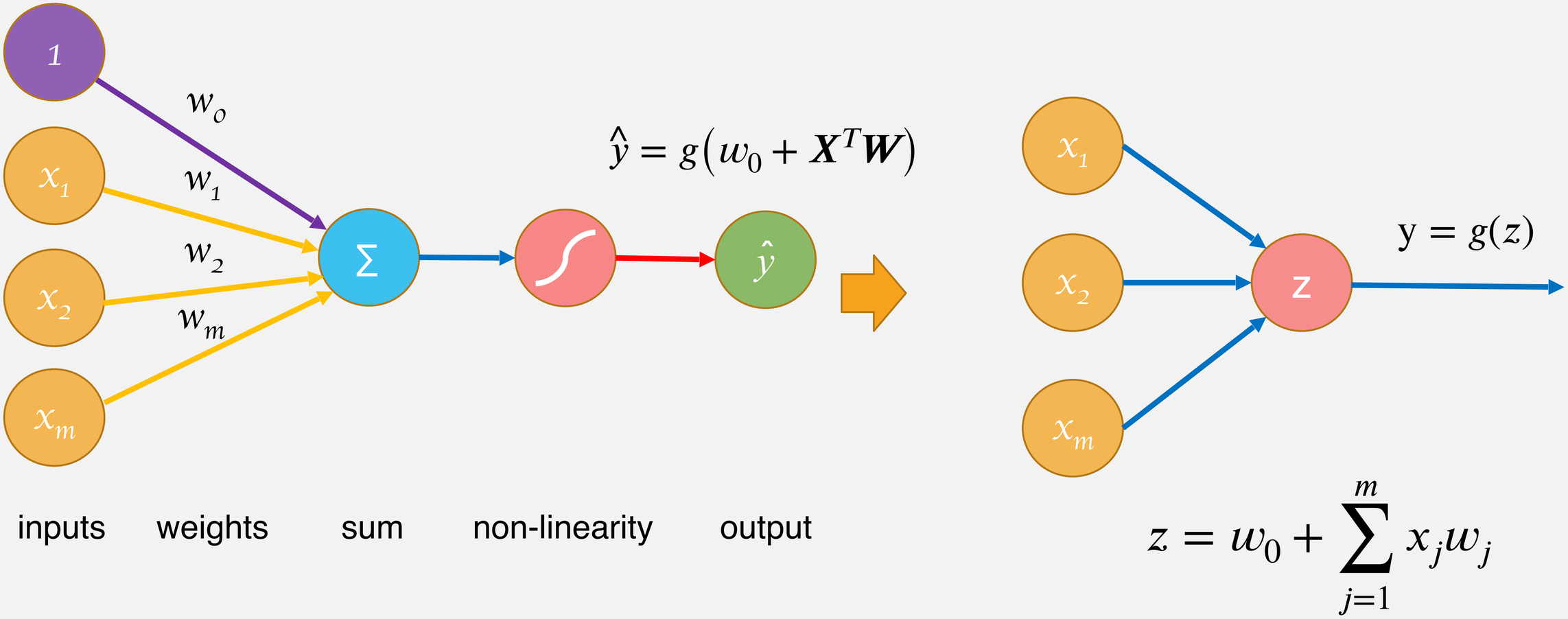




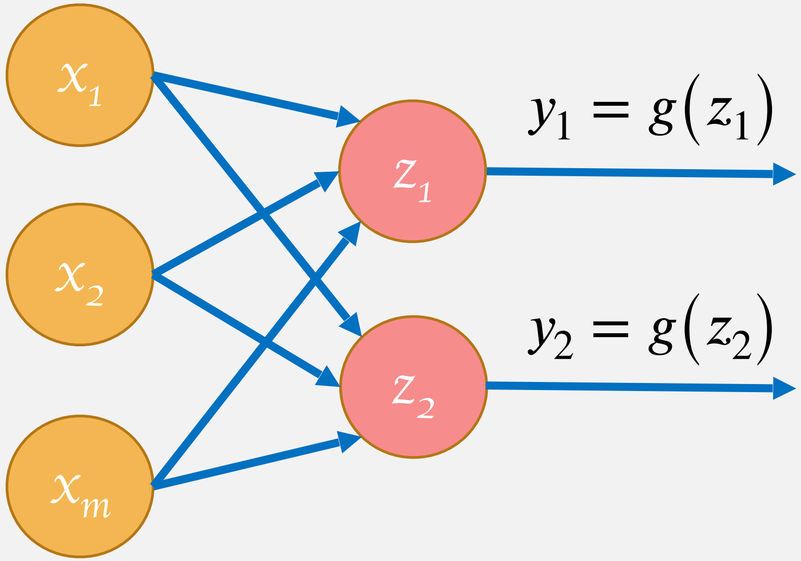


$$\hat{y} = g\left(1 + (3 * -1) - (2 * 2)\right) = g\left(-6\right) \approx 0.002$$

Simplifying the notation!!



Perceptron with Multiple Outputs

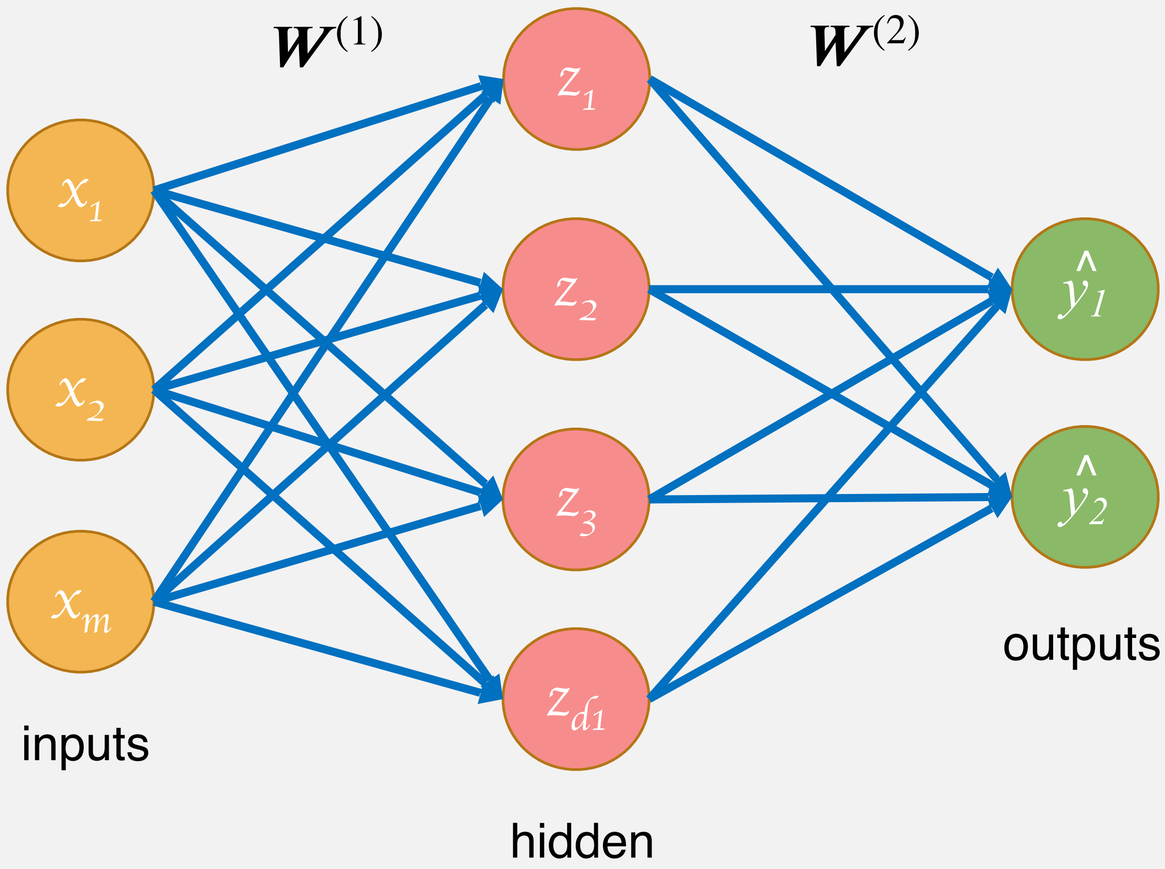


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

i-th output i-th output

Since all inputs are densely connected to all outputs, this structure is called **Dense Layer**!

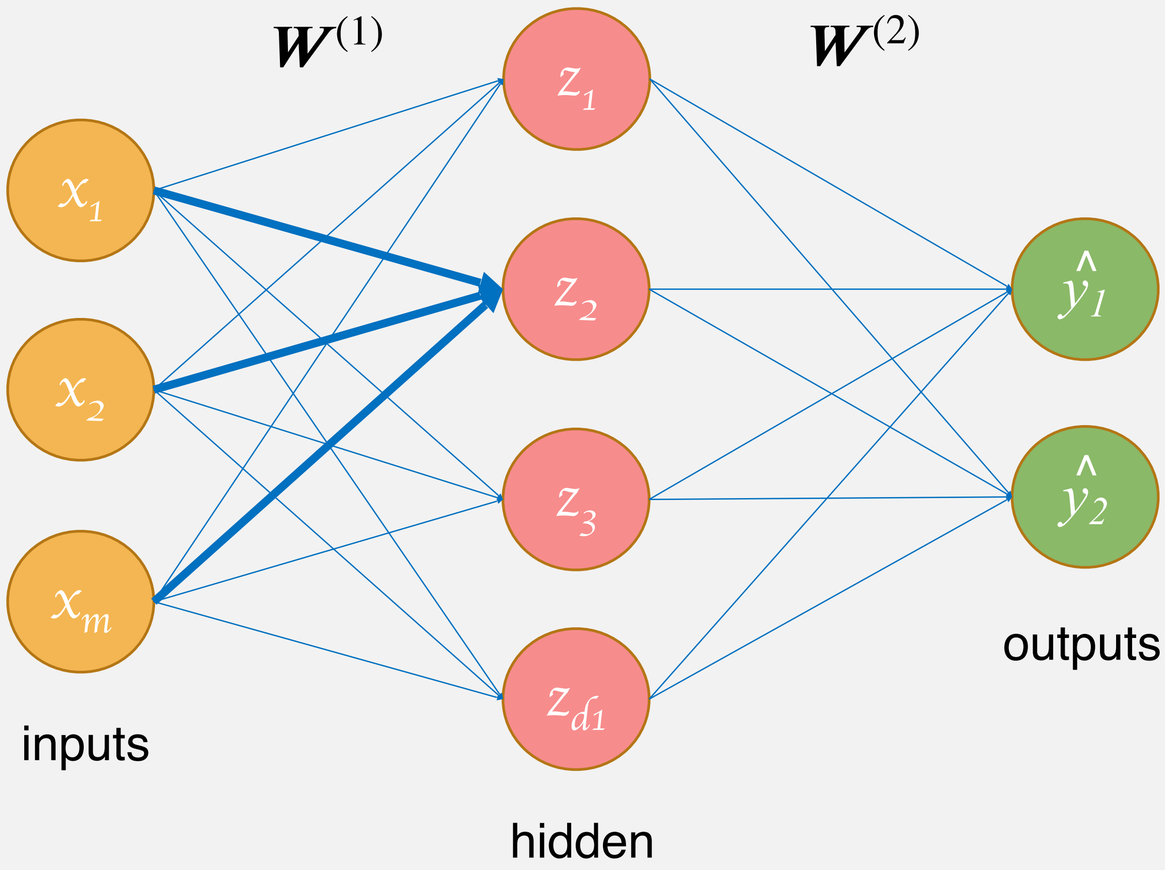
Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}$$

$$\hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j}^{d_1} g(z_j) w_{j,i}^{(2)} \right)$$

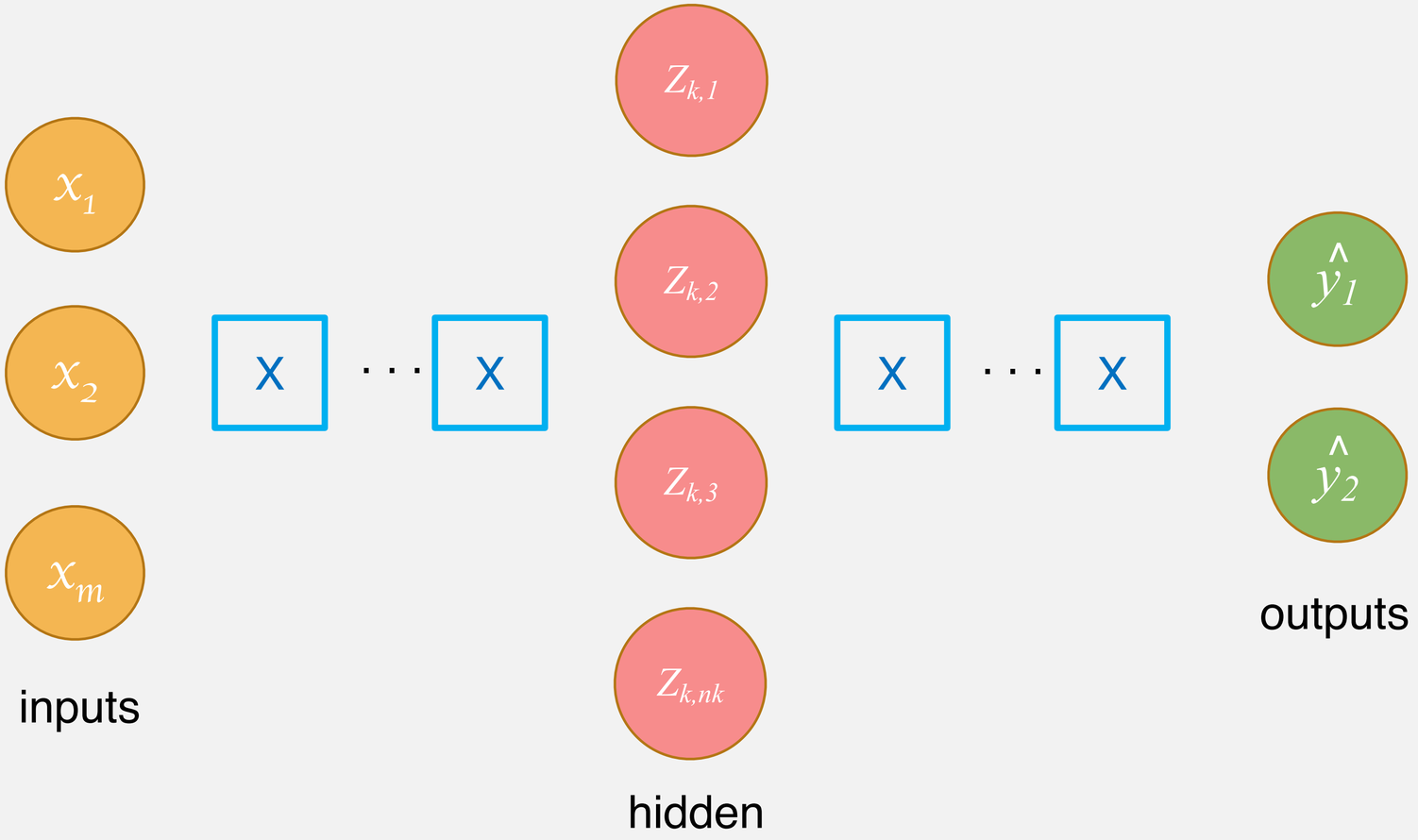
Single Layer Neural Network



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)}$$

$$z_2 = w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)}$$

Deep Neural Network

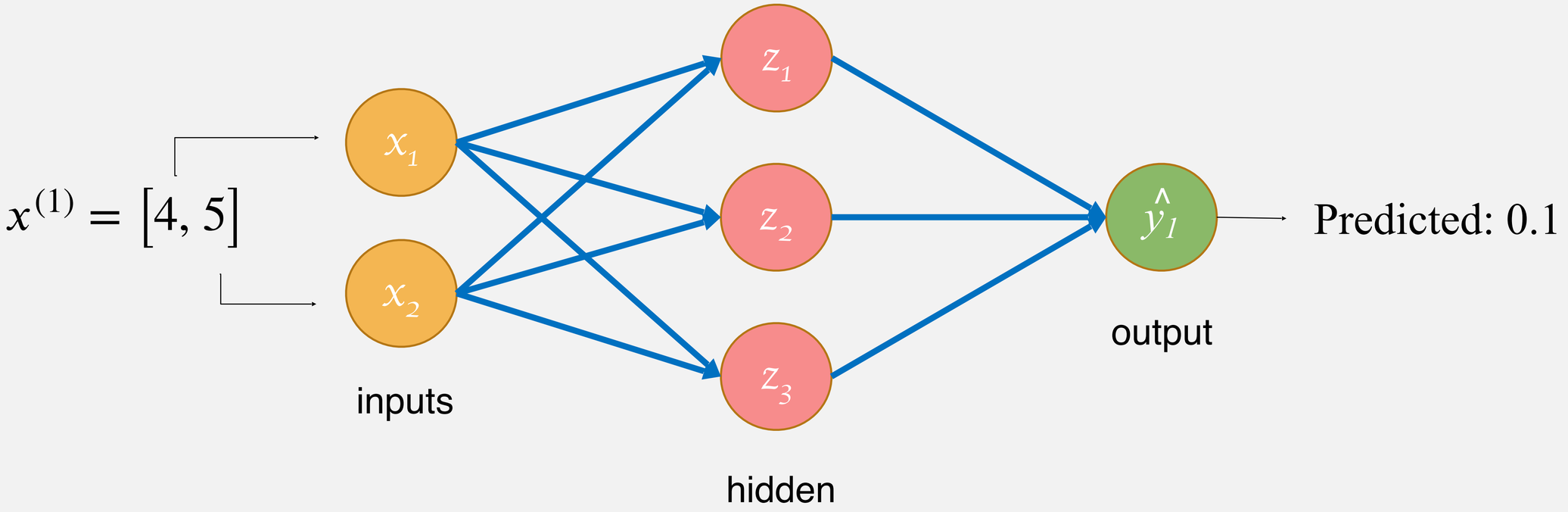


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

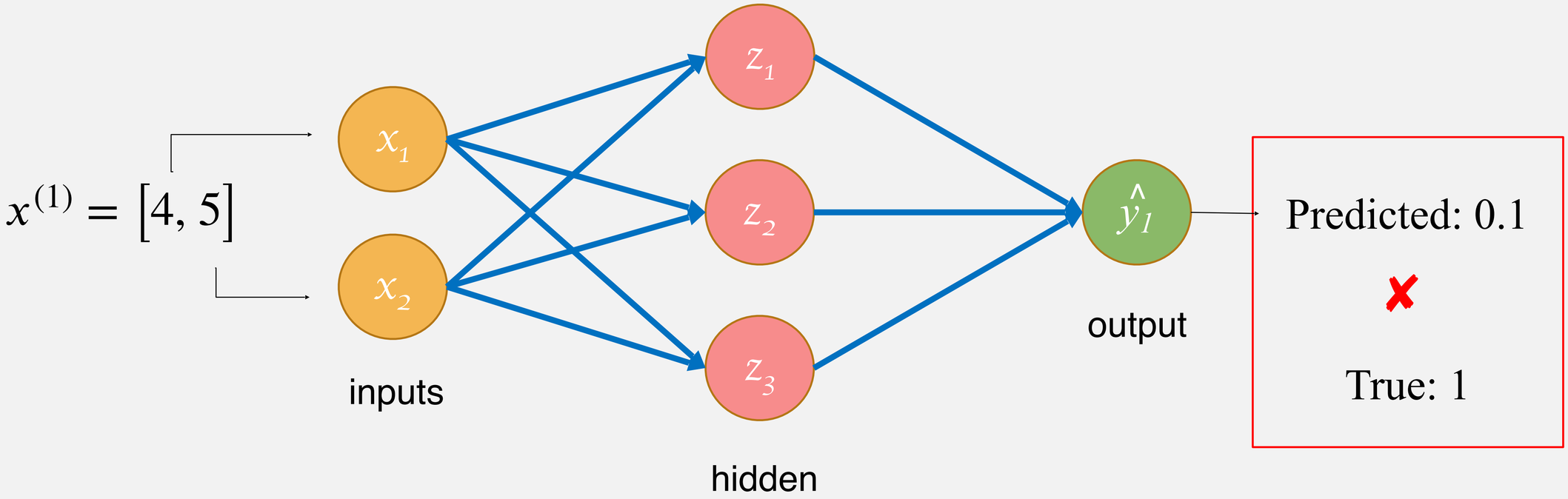
$z_{k,i}$ values at a given layer are computed from the weighted activation outputs from the previous layer $z_{k-1,j}$ values!!

How to find optimal Neural Networks weights?

Deep Learning Basics: Training Deep Neural Networks

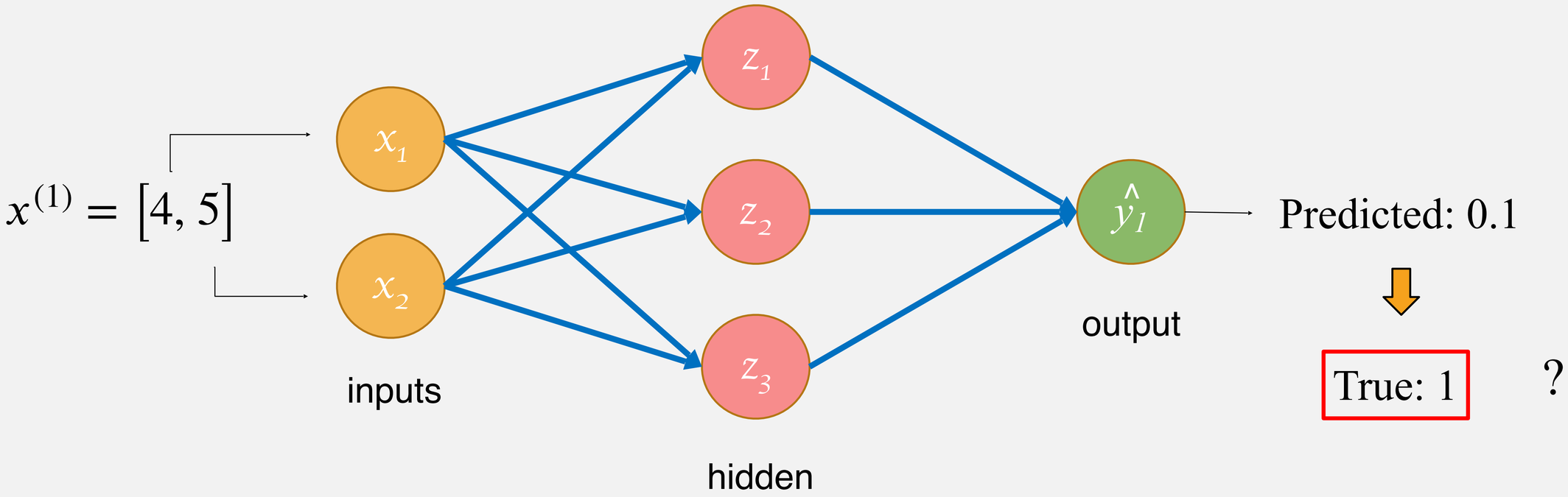


Deep Learning Basics: Training Deep Neural Networks



Not very good weights!

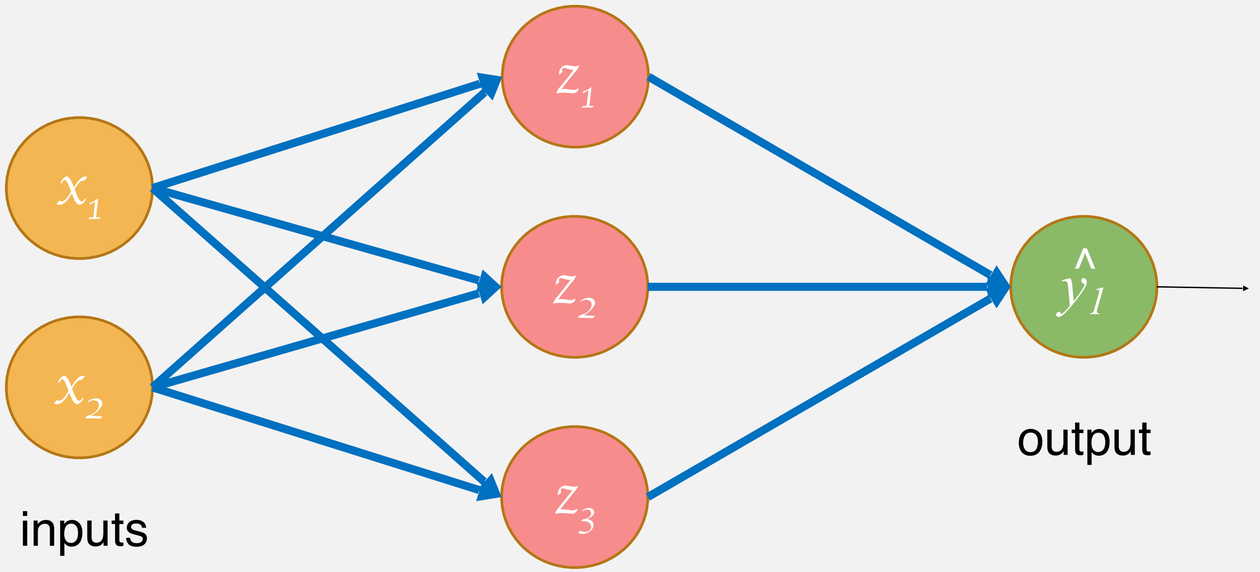
First step: measuring the prediction error or the **loss**



$$\mathscr{L}\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{predicted}}, \underbrace{y^{(i)}}_{\text{true}}\Big)$$

Empirical loss function: computes the total error in a given data **batch**

$$\boldsymbol{X} = \begin{bmatrix} 4 & 5 \\ 2 & 1 \\ 5 & 8 \\ 5 & 5 \\ \cdots & \end{bmatrix}$$



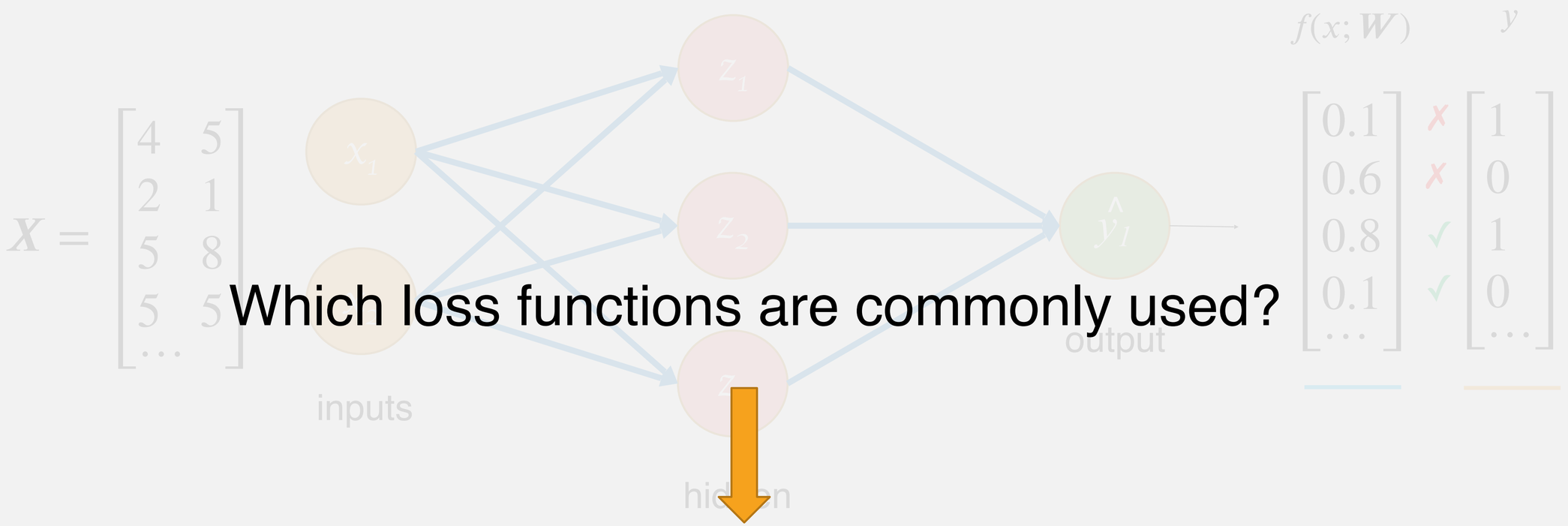
$f(x; \boldsymbol{W})$ y

$$\begin{bmatrix} 0.1 \\ 0.6 \\ 0.8 \\ 0.1 \\ \cdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \checkmark \\ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \cdots \end{bmatrix}$$

$$L(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)} \right)$$

predicted true

Empirical loss function: computes the total error in a given data **batch**



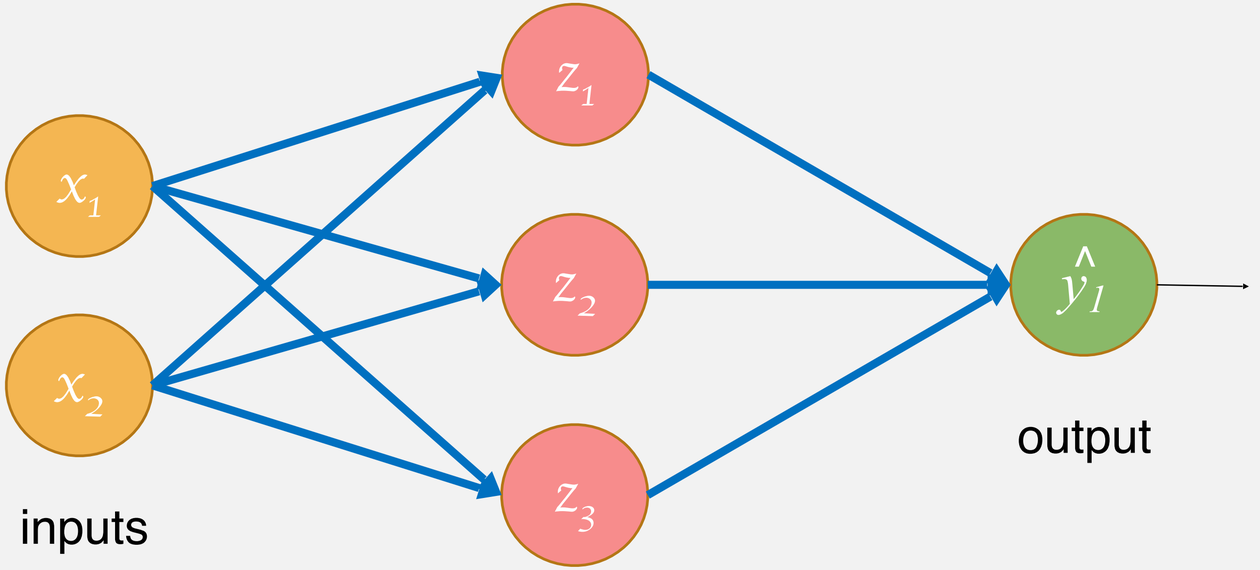
Which loss functions are commonly used?

$$L(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

predicted true

Mean Squared Error loss: computes the error for continuous, real number outputs

$$\boldsymbol{X} = \begin{bmatrix} 4 & 5 \\ 2 & 1 \\ 5 & 8 \\ 5 & 5 \\ \cdots & \end{bmatrix}$$



$f(x; \boldsymbol{W})$ y

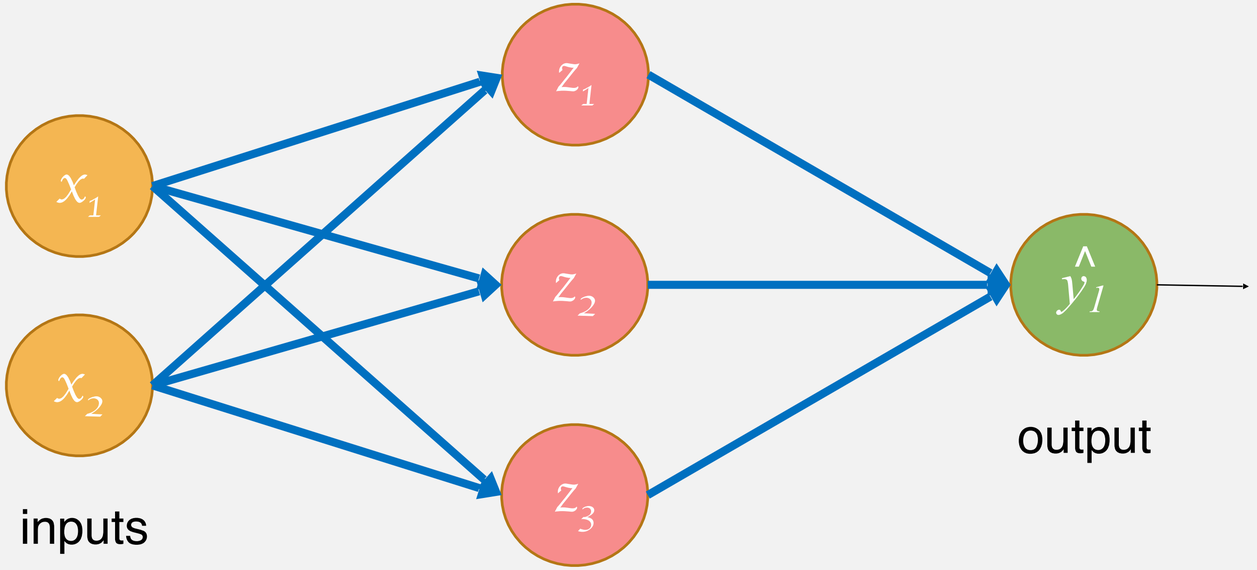
$$\begin{bmatrix} 30 \\ 80 \\ 85 \\ 15 \\ \cdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \checkmark \\ \end{matrix} \begin{bmatrix} 90 \\ 20 \\ 95 \\ 10 \\ \cdots \end{bmatrix}$$

$$L(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W} \right) \right)^2$$

true predicted

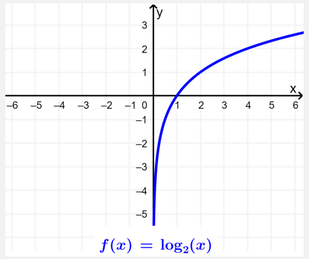
Binary Cross-Entropy loss: computes the error for prob. outputs in [0,1] and a binary truth

$$\boldsymbol{X} = \begin{bmatrix} 4 & 5 \\ 2 & 1 \\ 5 & 8 \\ 5 & 5 \\ \cdots & \end{bmatrix}$$



$f(x; \boldsymbol{W})$ y

$$\begin{bmatrix} 0.1 \\ 0.6 \\ 0.8 \\ 0.1 \\ \cdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \checkmark \\ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \cdots \end{bmatrix}$$



1-errors 0-errors

$$L(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; \boldsymbol{W}\right)\right)$$

true predicted true predicted

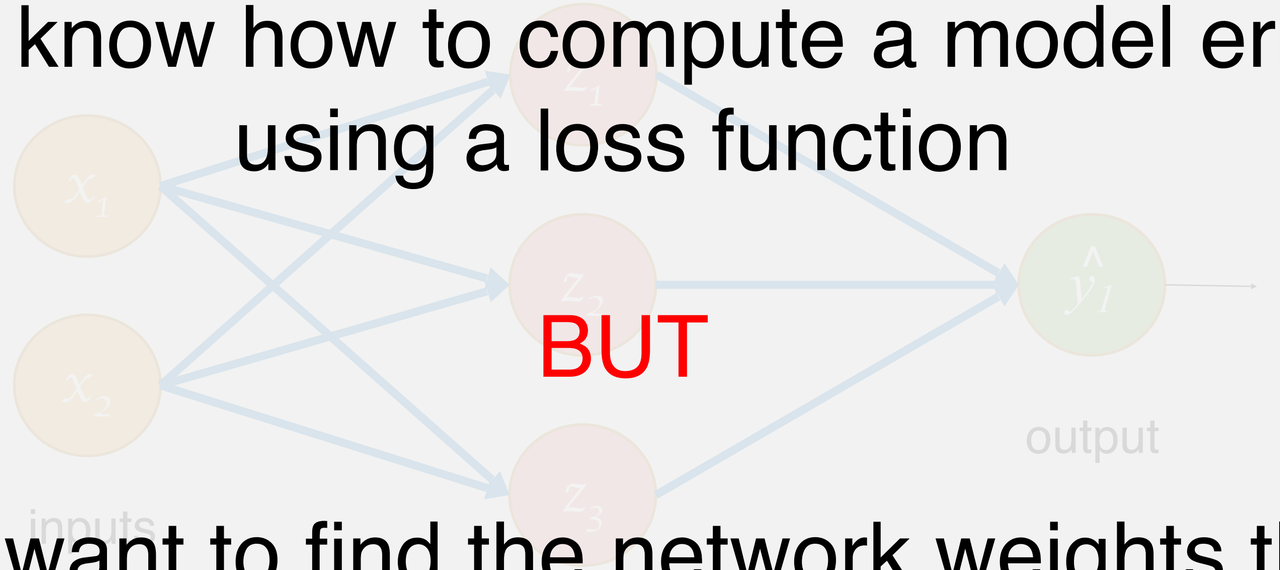
Binary Cross-Entropy loss: computes the error for prob. outputs in [0,1] and a binary truth

We know how to compute a model error using a loss function

BUT

We want to find the network weights that achieves the lowest loss!

$$X = \begin{bmatrix} 4 & 5 \\ 2 & 1 \\ 5 & 8 \\ 5 & 5 \\ \cdots \end{bmatrix}$$



$f(x; W)$ y

$$\begin{bmatrix} 0.1 \\ 0.6 \\ 0.8 \\ 0.1 \\ \cdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \checkmark \\ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ \cdots \end{bmatrix}$$

1-errors 0-errors

$$L(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; \boldsymbol{W}\right)\right)$$

true predicted true predicted

Loss Optimization: finding the optimal network weights that achieve the lowest loss

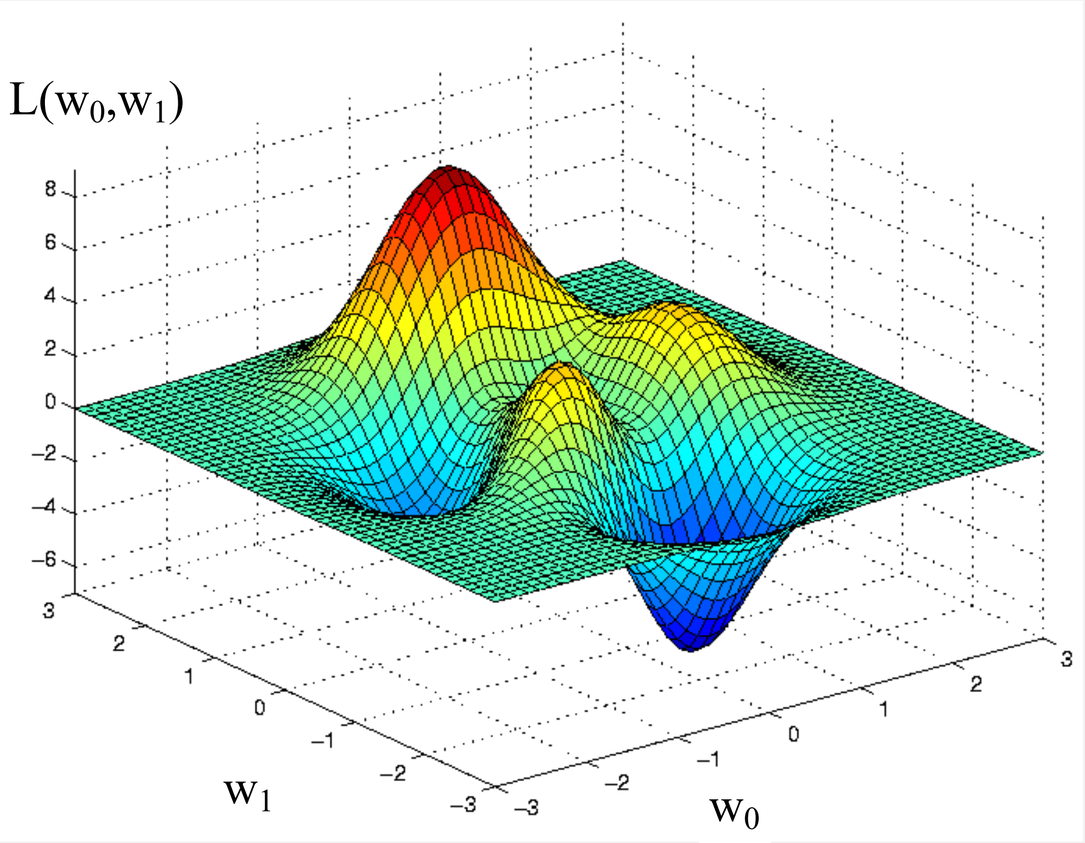
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} L(\boldsymbol{W}) \quad \text{where } \boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \ldots\}$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)} \right)$$

Loss Optimization: finding the optimal network weights that achieve the lowest loss

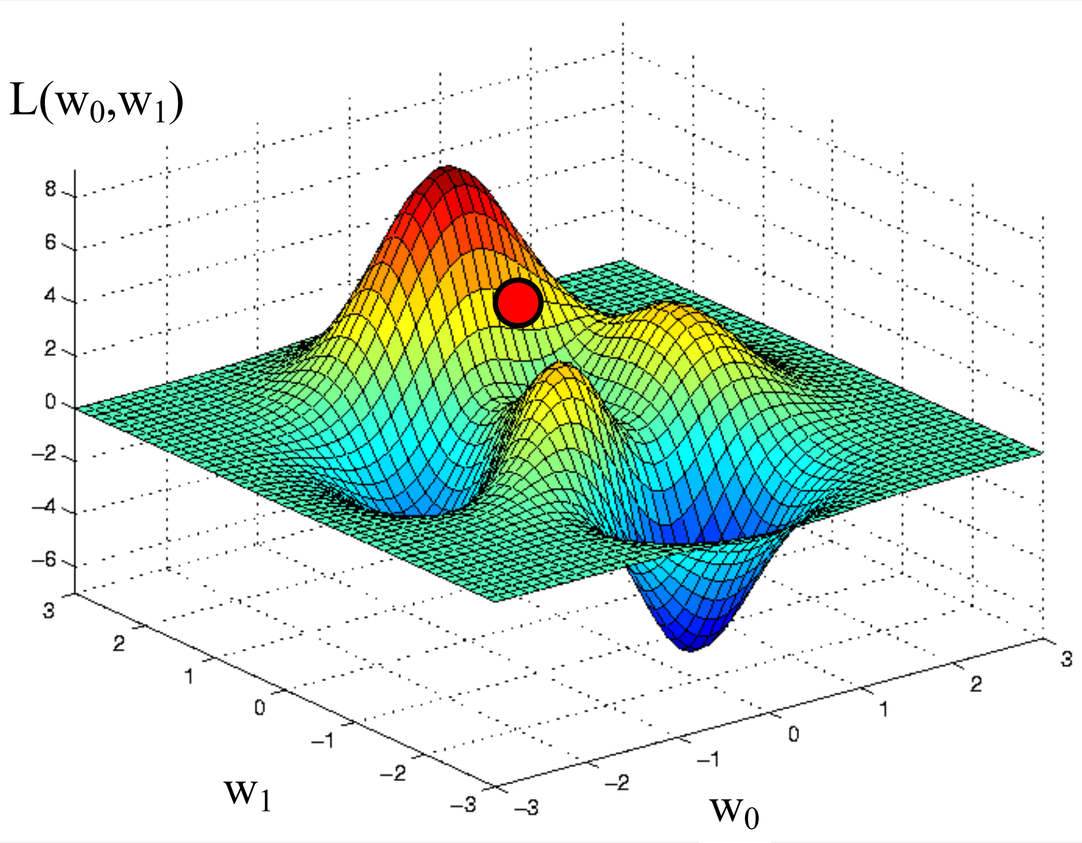
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} L(\boldsymbol{W})$$

L is a function of network weights!



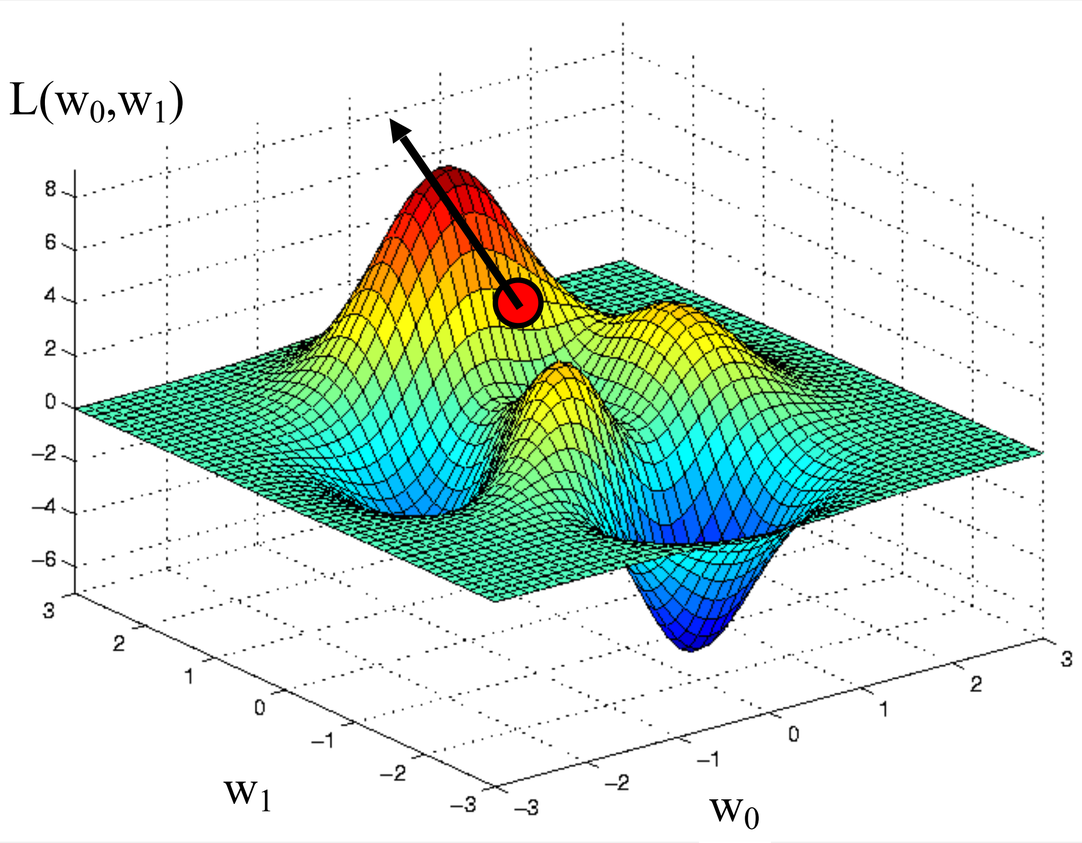
Loss Optimization using the **gradient descent** algorithm

1. Randomly pick an initial (w_0, w_1) point



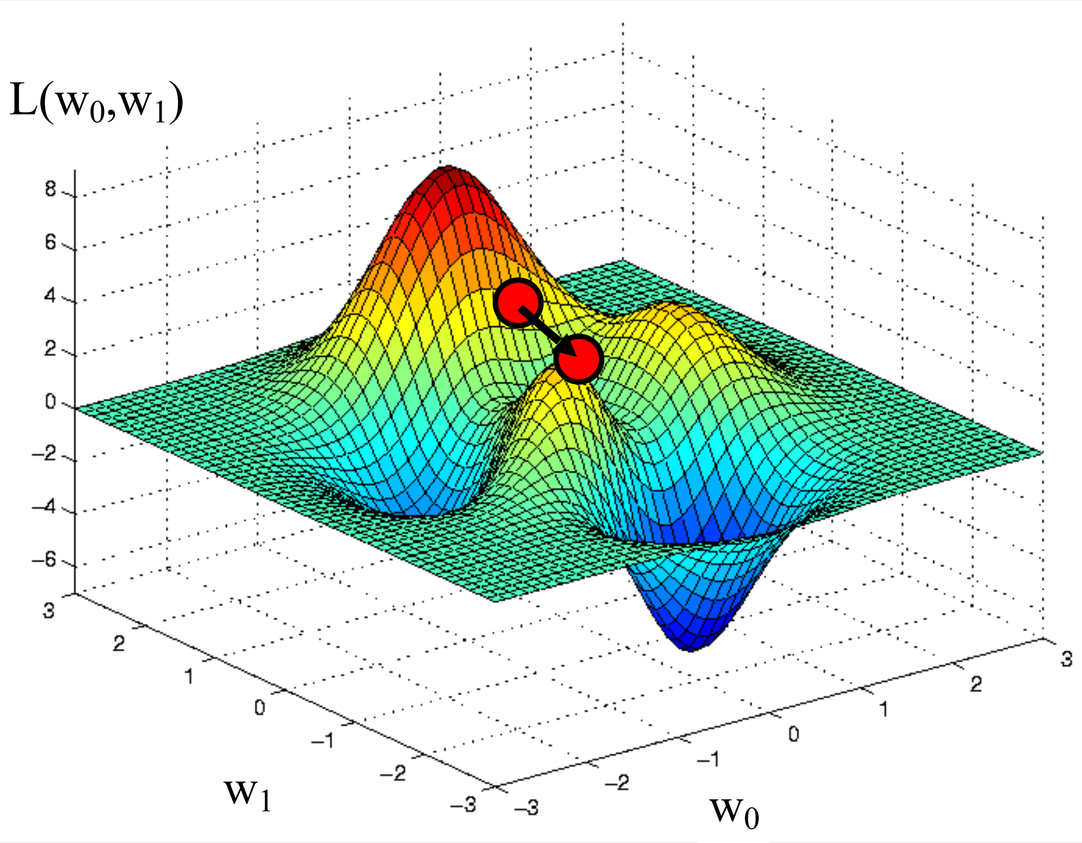
Loss Optimization using the **gradient descent** algorithm

2. Compute the gradient $\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$



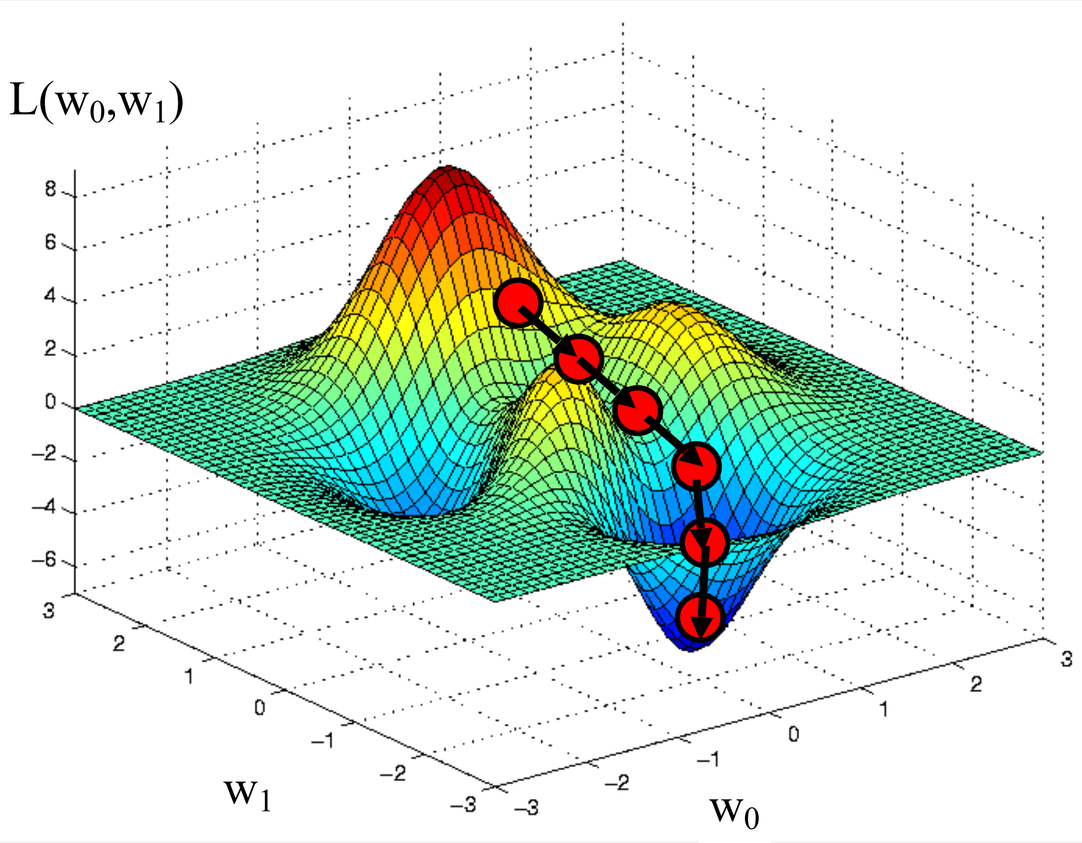
Loss Optimization using the **gradient descent** algorithm

3. Take a small step in the opposite direction of the gradient



Loss Optimization using the **gradient descent** algorithm

4. Repeat until convergence



Gradient descent algorithm

1. Initialize weights randomly $\sim N(0,\sigma^2)$
2. Loop until convergence
3. Compute gradient $\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update network weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning rate

Gradient descent algorithm

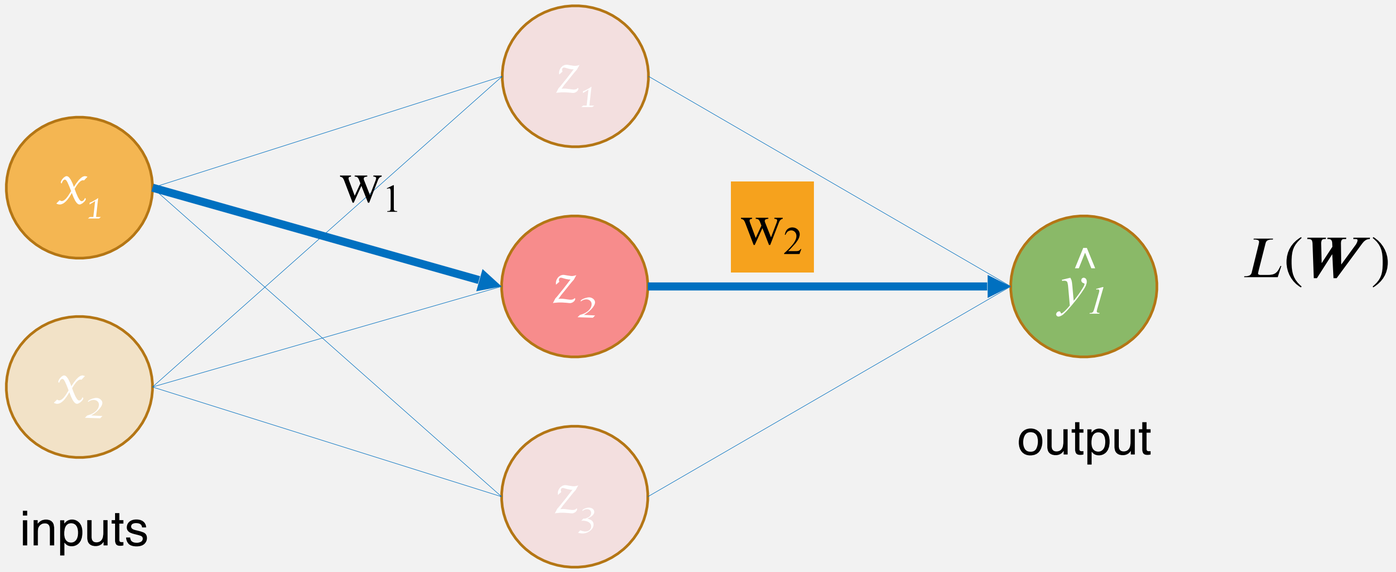
1. Initialize weights randomly $\sim N(0, \sigma^2)$
2. Loop until convergence
3. Compute gradient $\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update network weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

How to compute gradients?

Learning rate

Back-propagation algorithm

How does a **small change in** w_2 affect the output and the loss function?

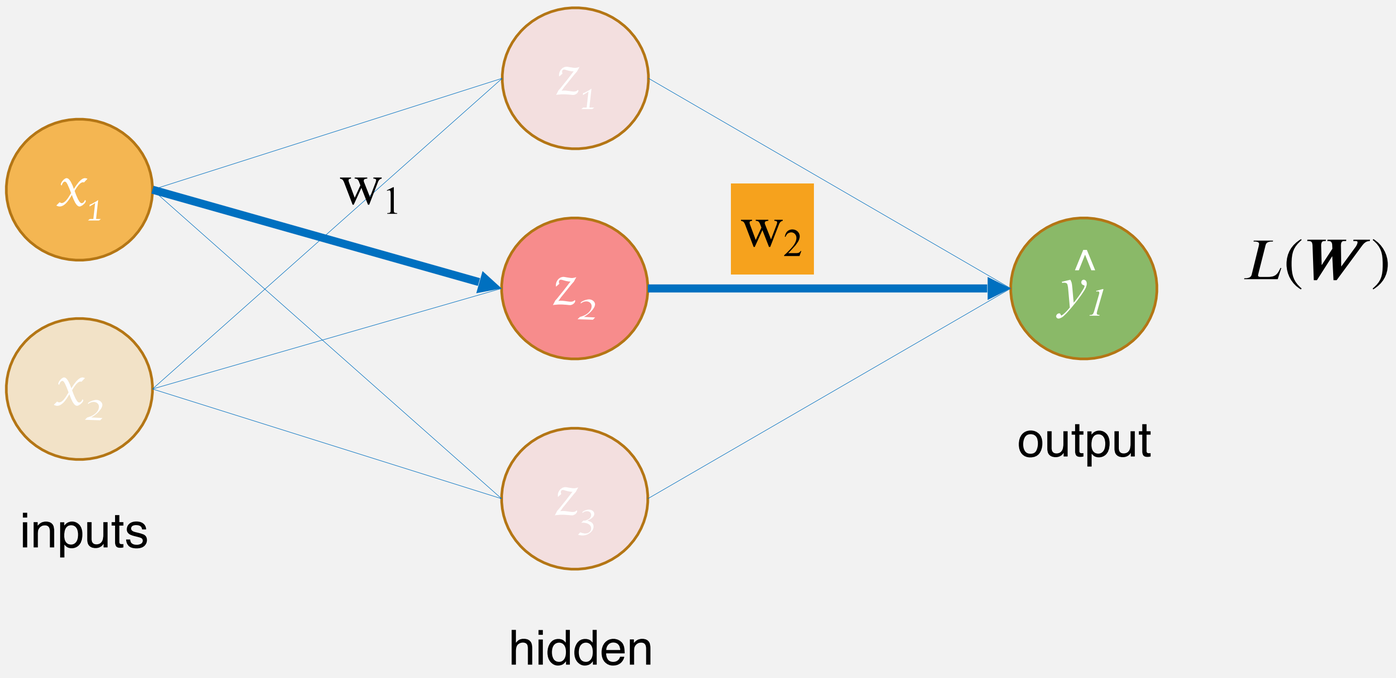


$$\frac{\partial L(\boldsymbol{W})}{\partial w_2}$$

(let's apply the chain rule!!)

Back-propagation algorithm

How does a **small change in** w_2 affect the output and the loss function?

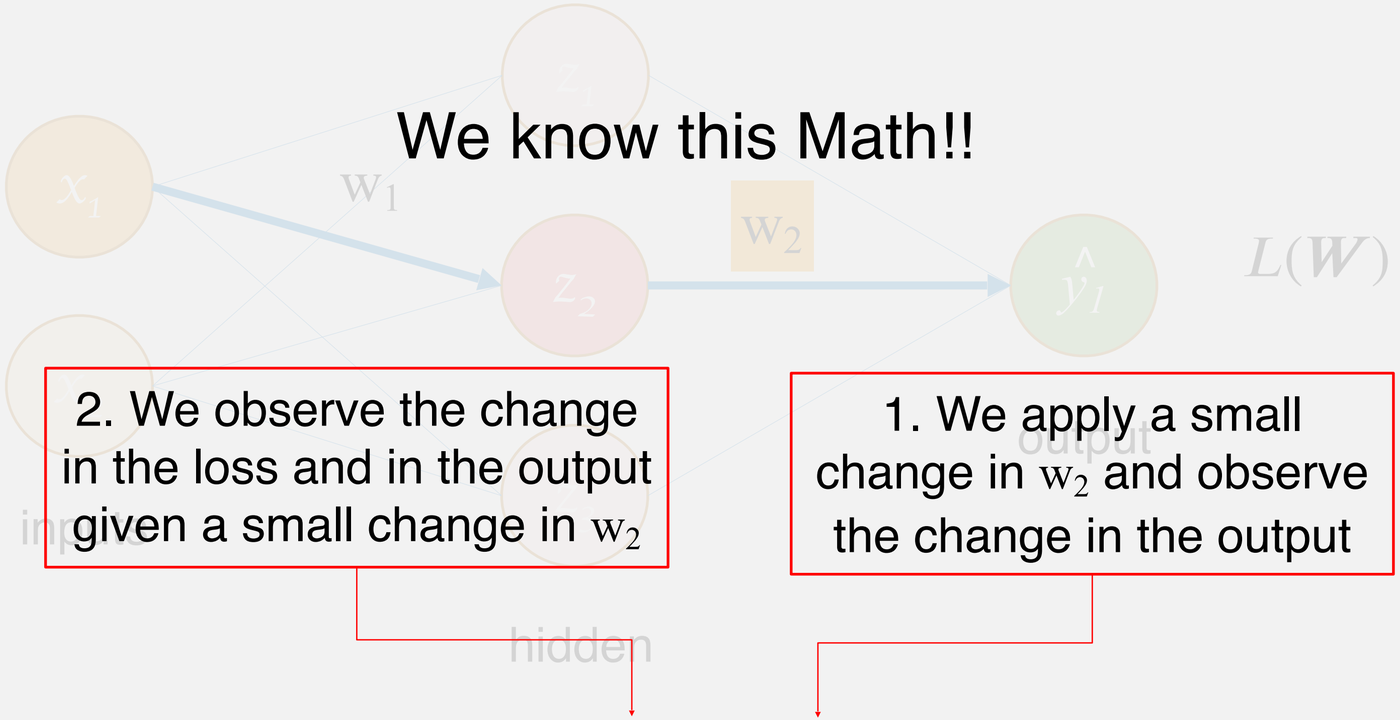


$$\frac{\partial L(\boldsymbol{W})}{\partial w_2} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial w_2}$$

(chain rule!!)

Back-propagation algorithm

How does a **small change in w_2** affect the output and the loss function?

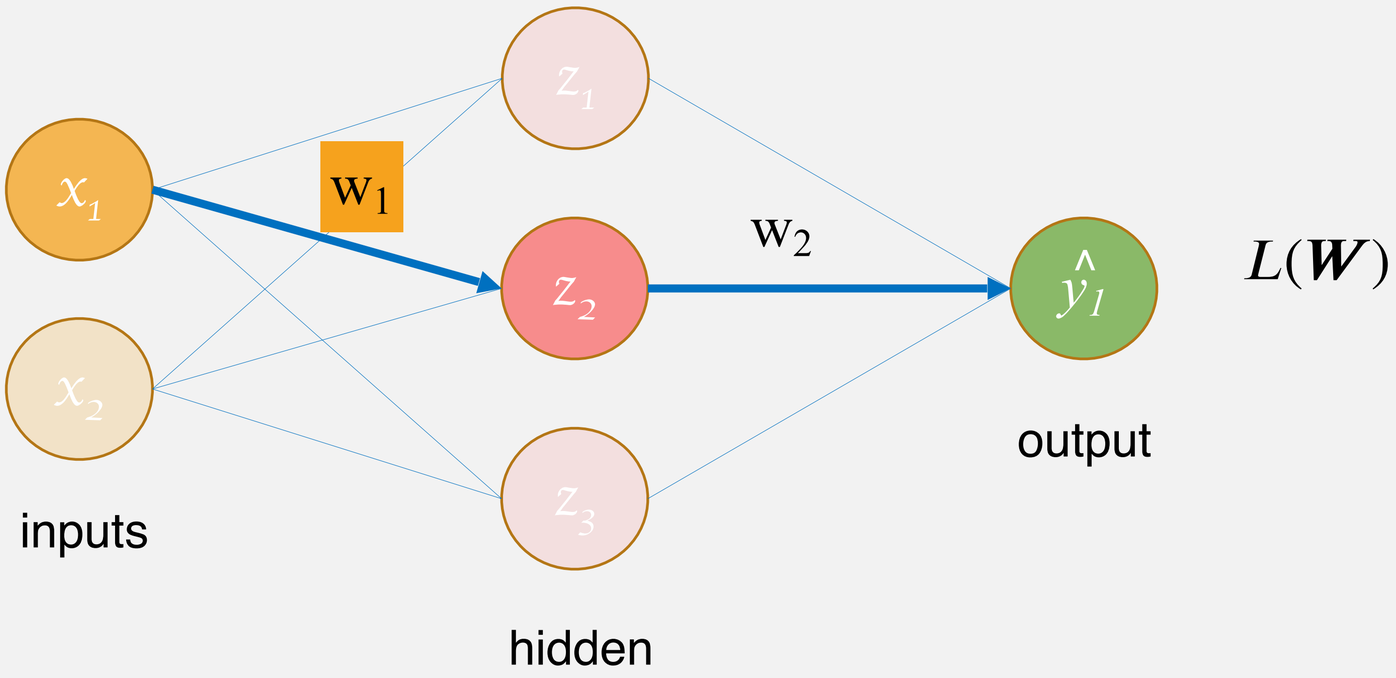


$$\frac{\partial L(\boldsymbol{W})}{\partial w_2} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial w_2}$$

(chain rule!!)

Back-propagation algorithm

How does a **small change in** w_1 affect the output and the loss function?

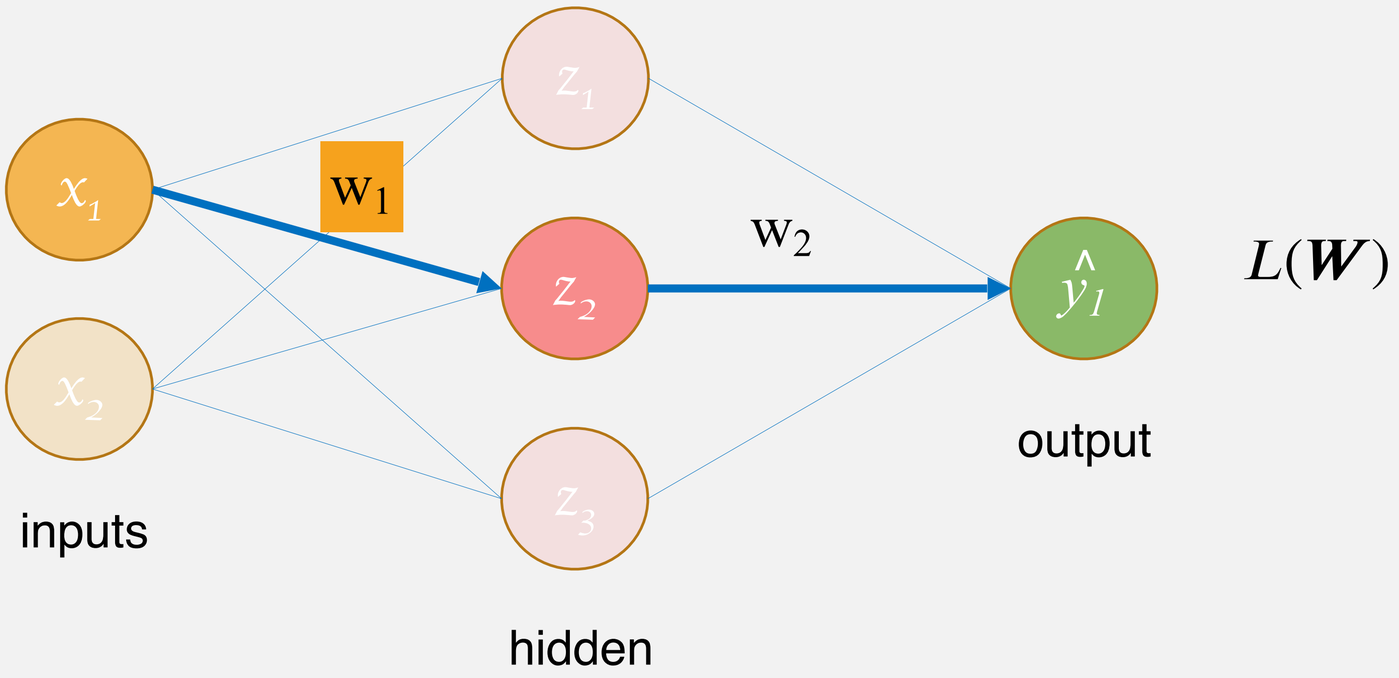


$$\frac{\partial L(\boldsymbol{W})}{\partial w_1} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial w_1}$$

(chain rule!!)

Back-propagation algorithm

How does a **small change in** w_1 affect the output and the loss function?

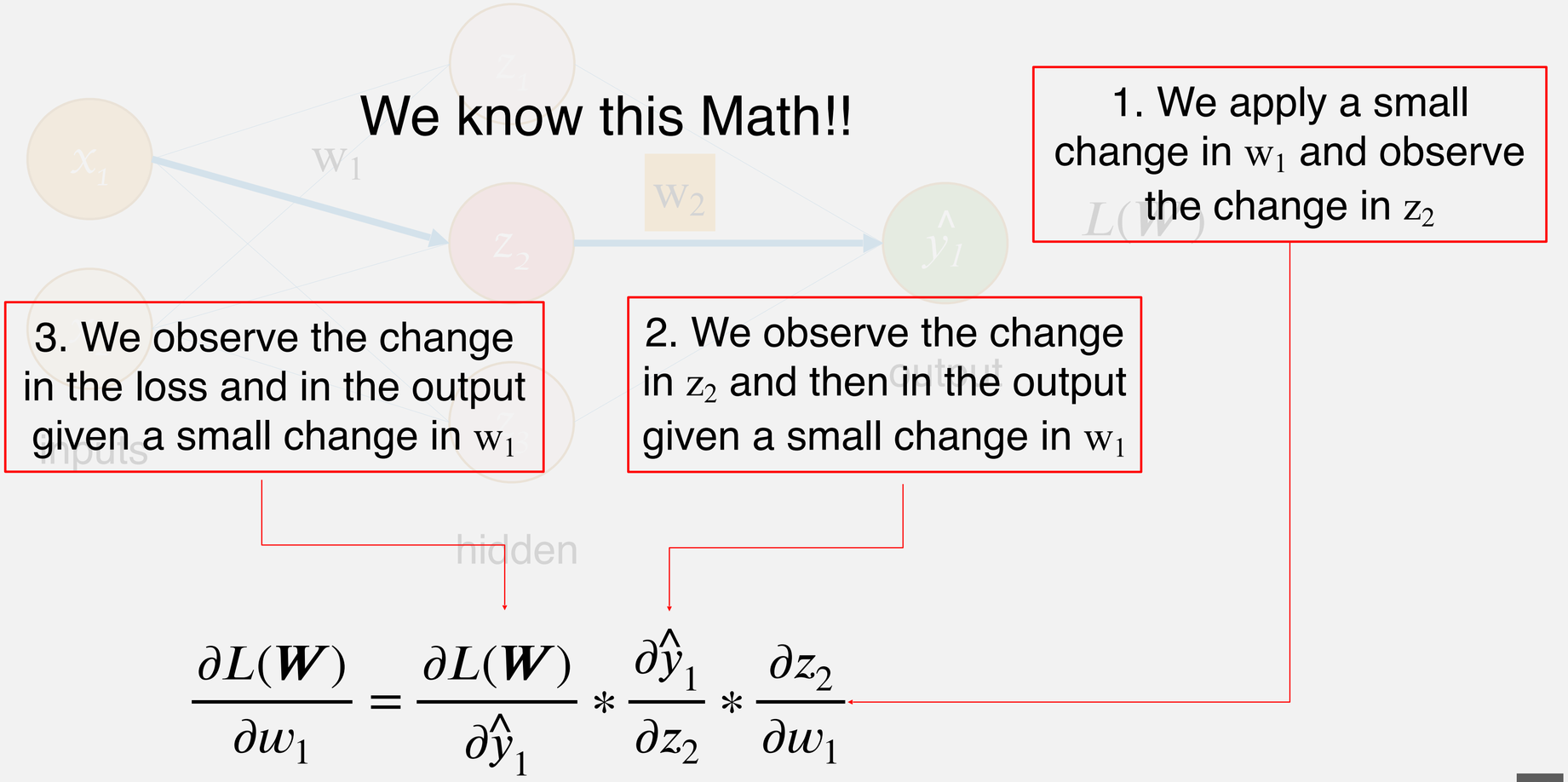


$$\frac{\partial L(\boldsymbol{W})}{\partial w_1} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial z_2} * \frac{\partial z_2}{\partial w_1}$$

(and chain rule again!!)

Back-propagation algorithm

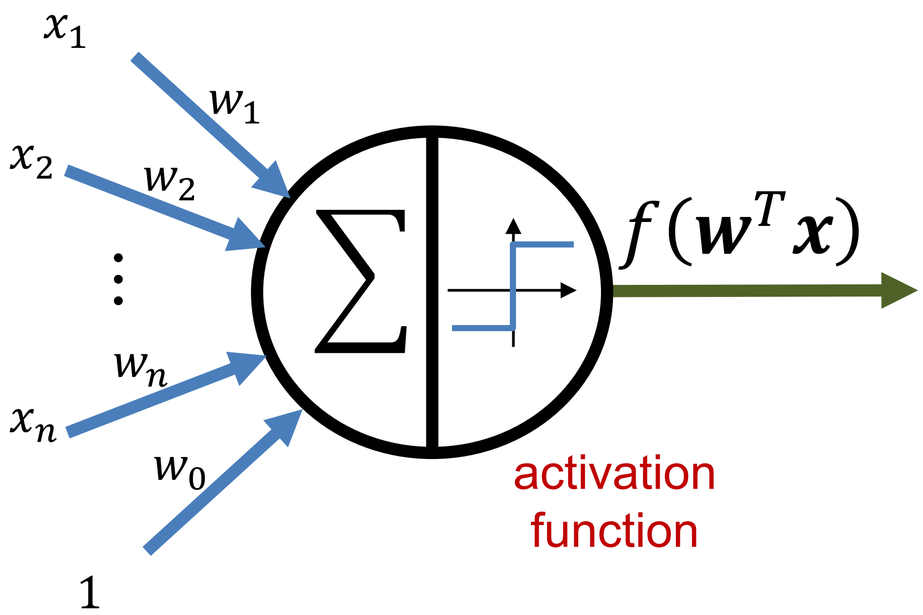
How does a **small change in w_1** affect the output and the loss function?



Digging deeper on details

Rosenblatt Perceptron

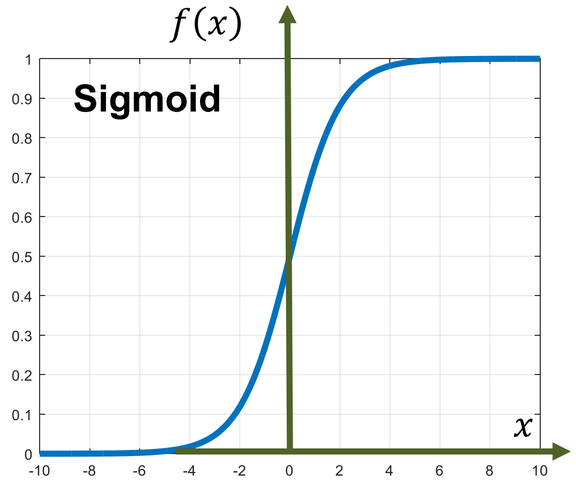
- Problem: the derivative of step function used in Rosemblatt perceptron is equal to zero!
- SGD relies on derivatives: the zero derivative behavior will imply in a zero update.

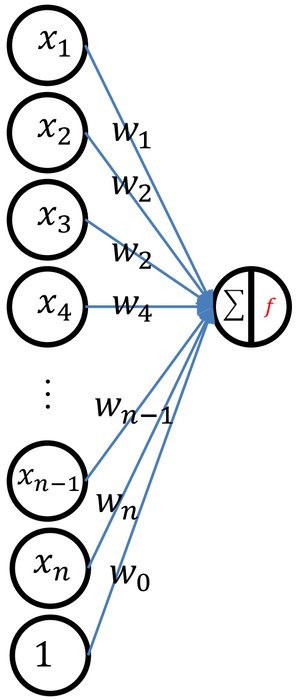


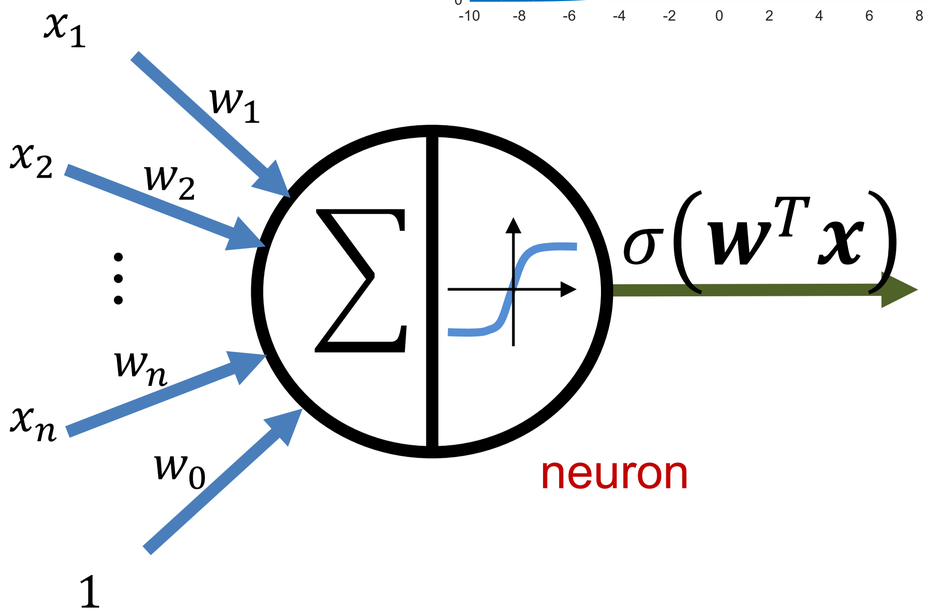
Logistic Regression

$$f(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}} = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

Posterior probability for a class, given the data $\boldsymbol{x}$

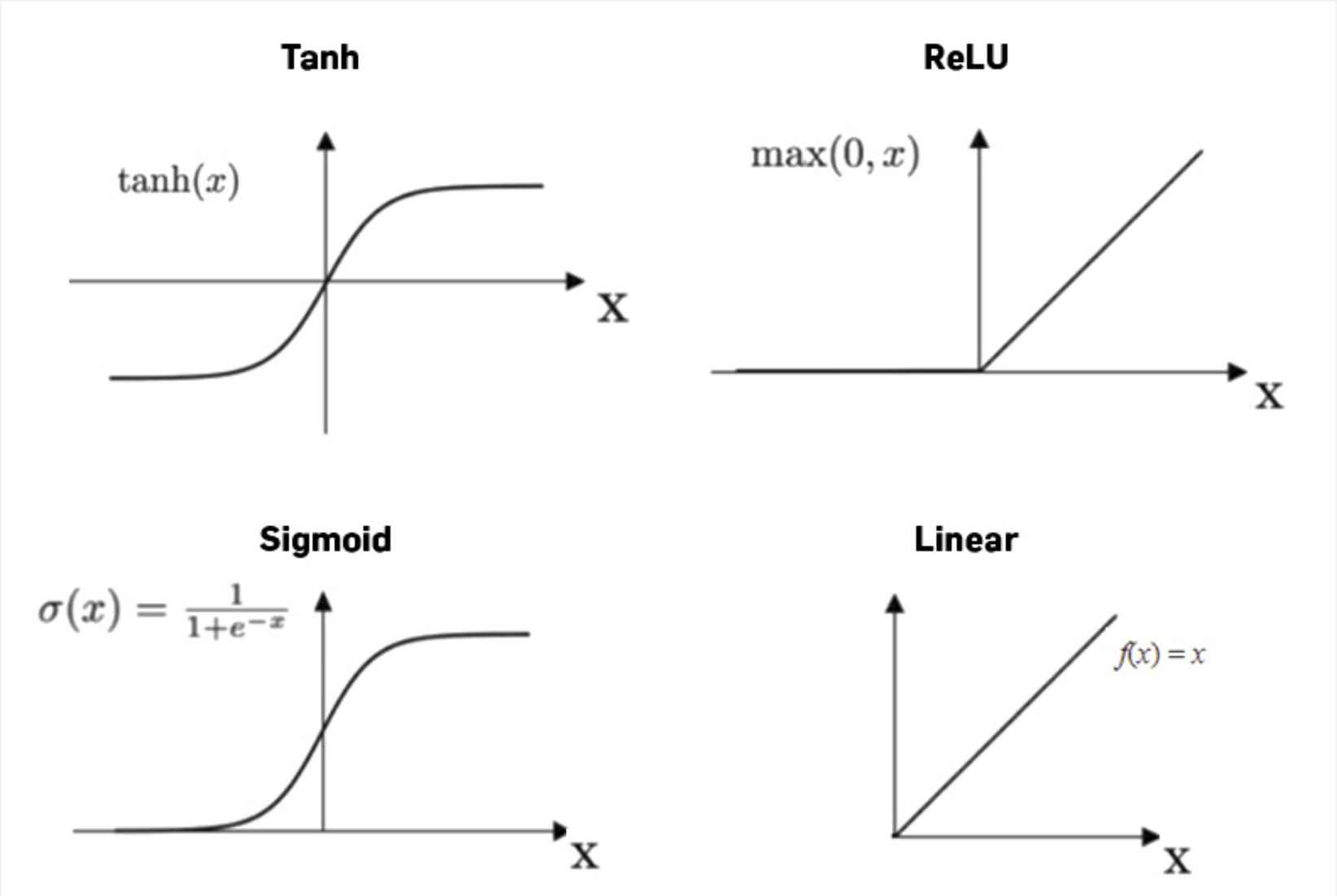




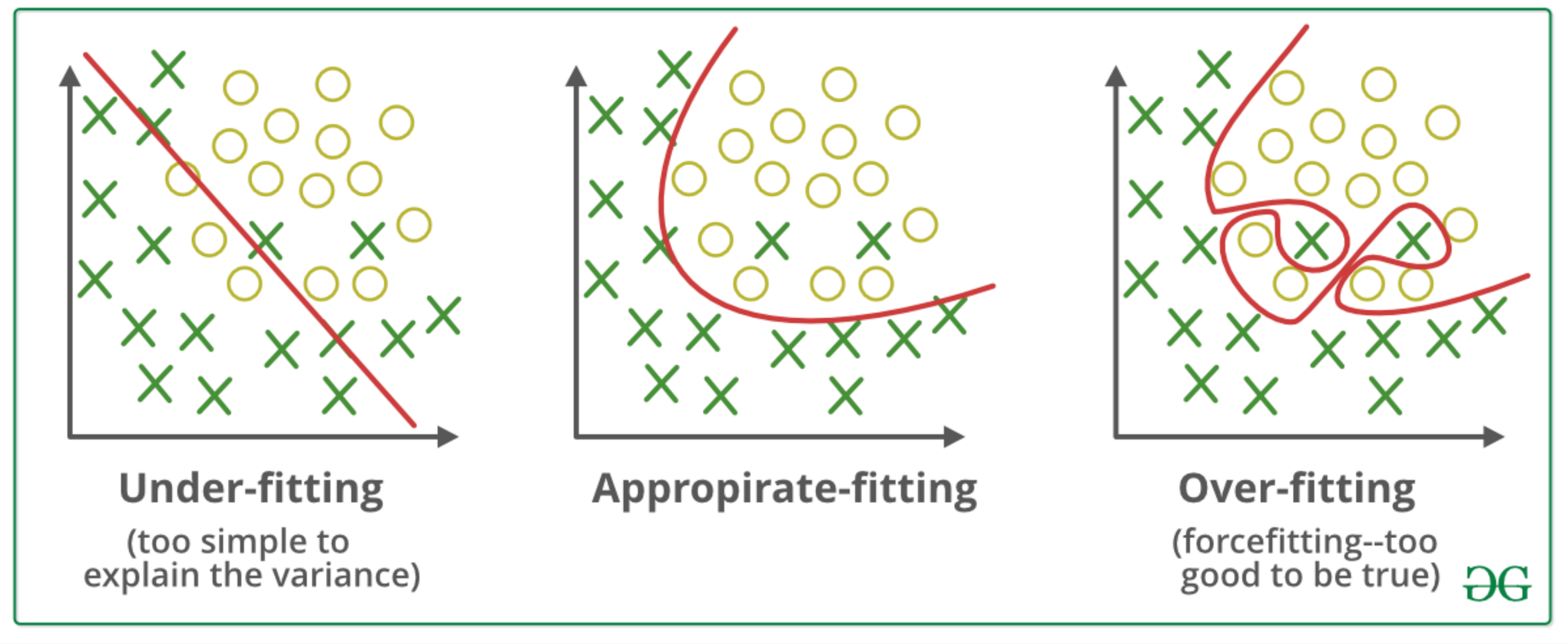


Deep Learning Basics: The Perceptron Model

Common activation functions



Overfitting: when the model learns “too much”



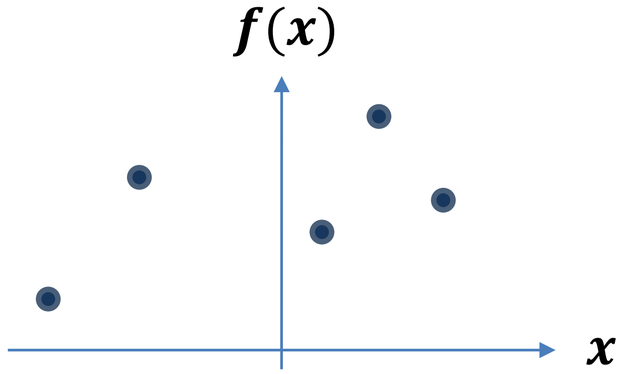
* from https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

Regularization

- In practice we are not concerned with the performance of the classifier for the training data, but rather on how the classifier will behave with samples not seen during training.
- This is called **generalization**.

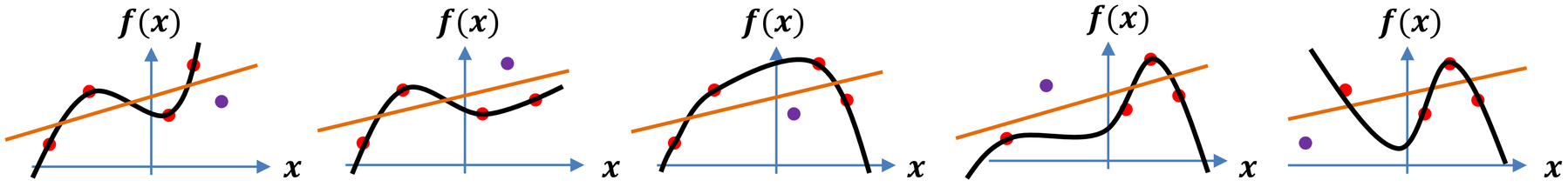
Regularization

- This is easier to illustrate with a regression problem.
- Consider we have 5 points of known coordinates, and wish from 4 of these points to estimate the curve that best fits all points.
- Once the curve is estimated, if we enter an unknown value of x we can estimate the value of $f(x)$.



Regularization

- Suppose we try to fit both a grade 3 polynomial and a straight line on the (training) 4 points.
- Clearly, the simpler function is closer to the point that did not participate in the training (it generalizes better).



Occam's Razor: "Among competing hypotheses, the simplest is the best", William of Ockham, 1285-1347

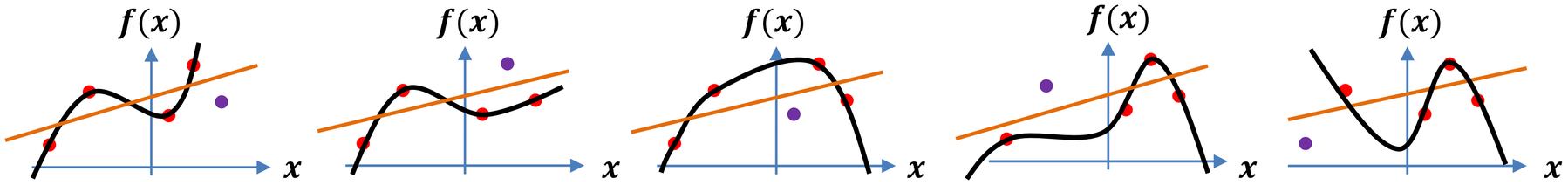
Regularization

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_i L_i[f(\boldsymbol{W}, \boldsymbol{x}_i), y_i] + \lambda R(\boldsymbol{W})$$

regulation weight

Data loss
match between model and data

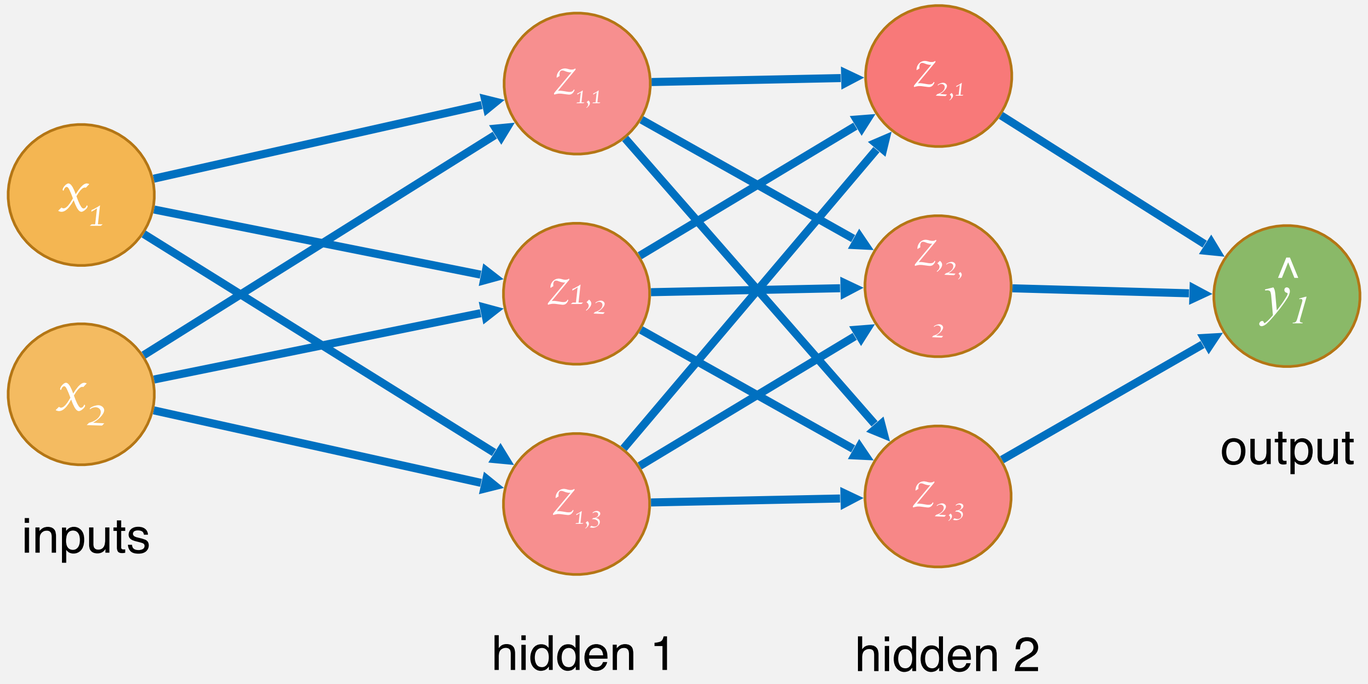
Regularization
model complexity



Occam's Razor: "Among competing hypotheses, the simplest is the best", William of Ockham, 1285-1347

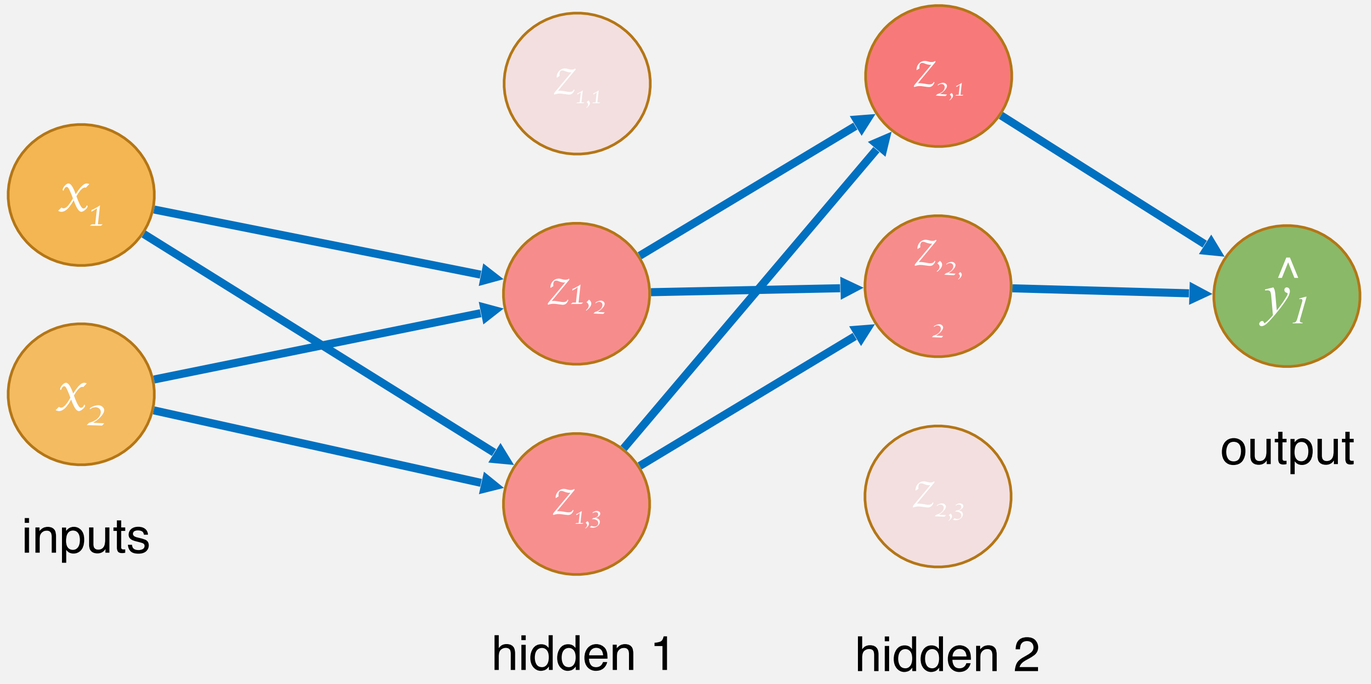
Dropout

Erasing some connections to improve generalisation on unseen data



Dropout

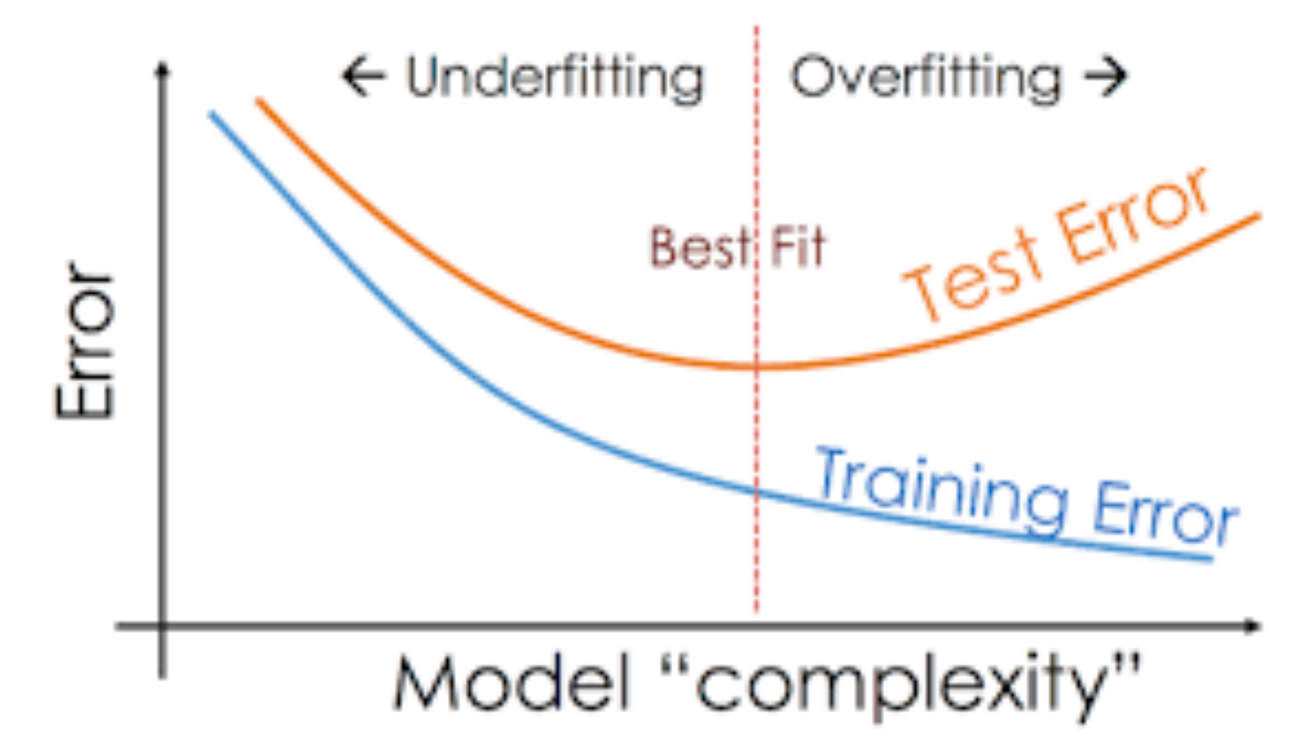
Erasing some connections to improve generalization on unseen data



it learns not to trust in specific nodes!

Early stopping

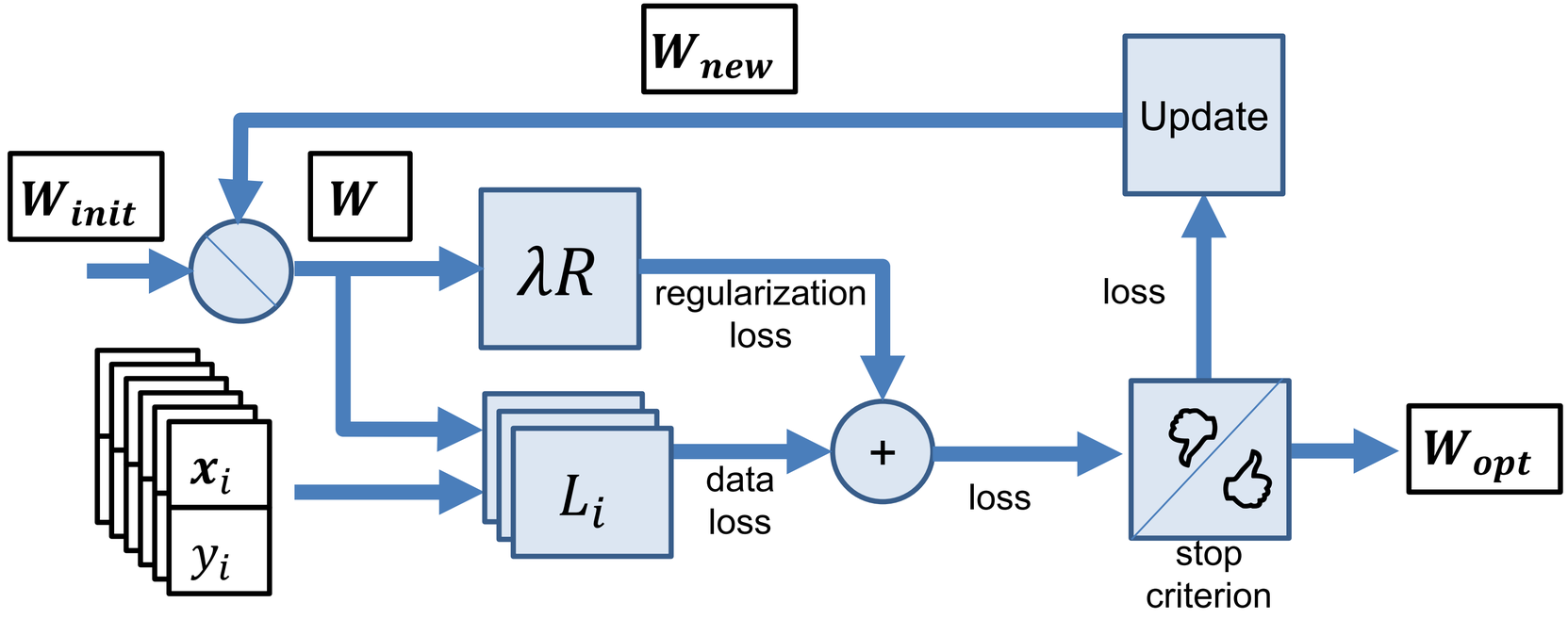
Using validation set to identify when the model losses generalization capacity



Looking for the best $\boldsymbol{W}$

Iterative training procedure:

$$\boldsymbol{W}_{opt} = \underset{\boldsymbol{W}}{\operatorname{argmin}} \left\{ \frac{1}{N} \sum_i L_i[f(\boldsymbol{W}, \boldsymbol{x}_i), y_i] + \lambda\, R(\boldsymbol{W}) \right\}$$



Stochastic Gradient Descent

Often large training sets are required to tune a classifier.

$$\nabla_{\boldsymbol{W}} L(\boldsymbol{W}) = \frac{1}{N} \sum_{i} \nabla_{\boldsymbol{W}} L_i[f(\boldsymbol{W}, \boldsymbol{x}_i), y_i] + \lambda \nabla_{\boldsymbol{W}} R(\boldsymbol{W})$$

Computing the gradient over all samples might be extremely expensive $(O(N))$!

Stochastic Gradient Descent

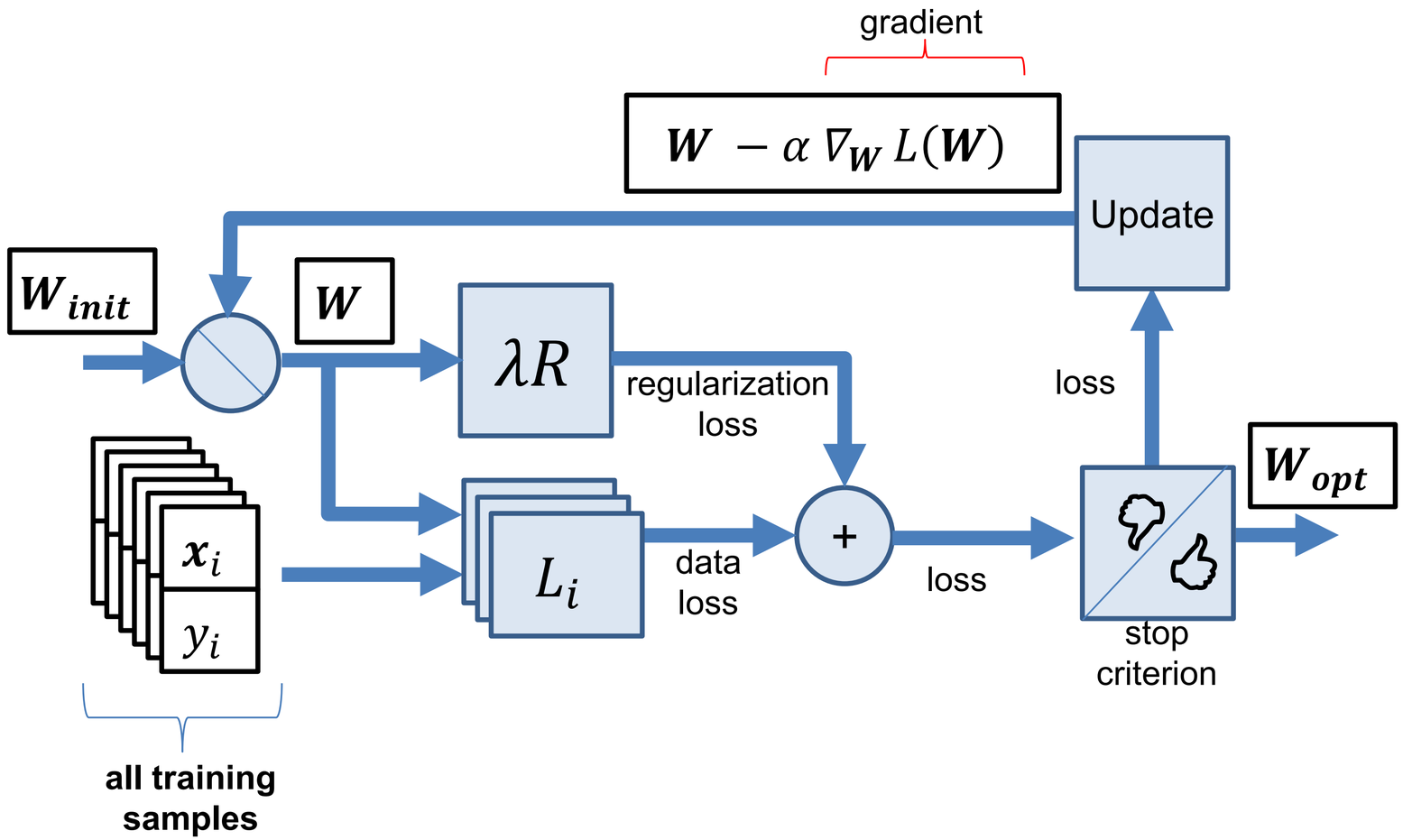
Sample **minibatches** of samples $\mathbb{B} = \{x^{(1)}, \ldots, x^{(m)}\}$, for a "small" m.

At each iteration we estimate the gradient on a minibatch:

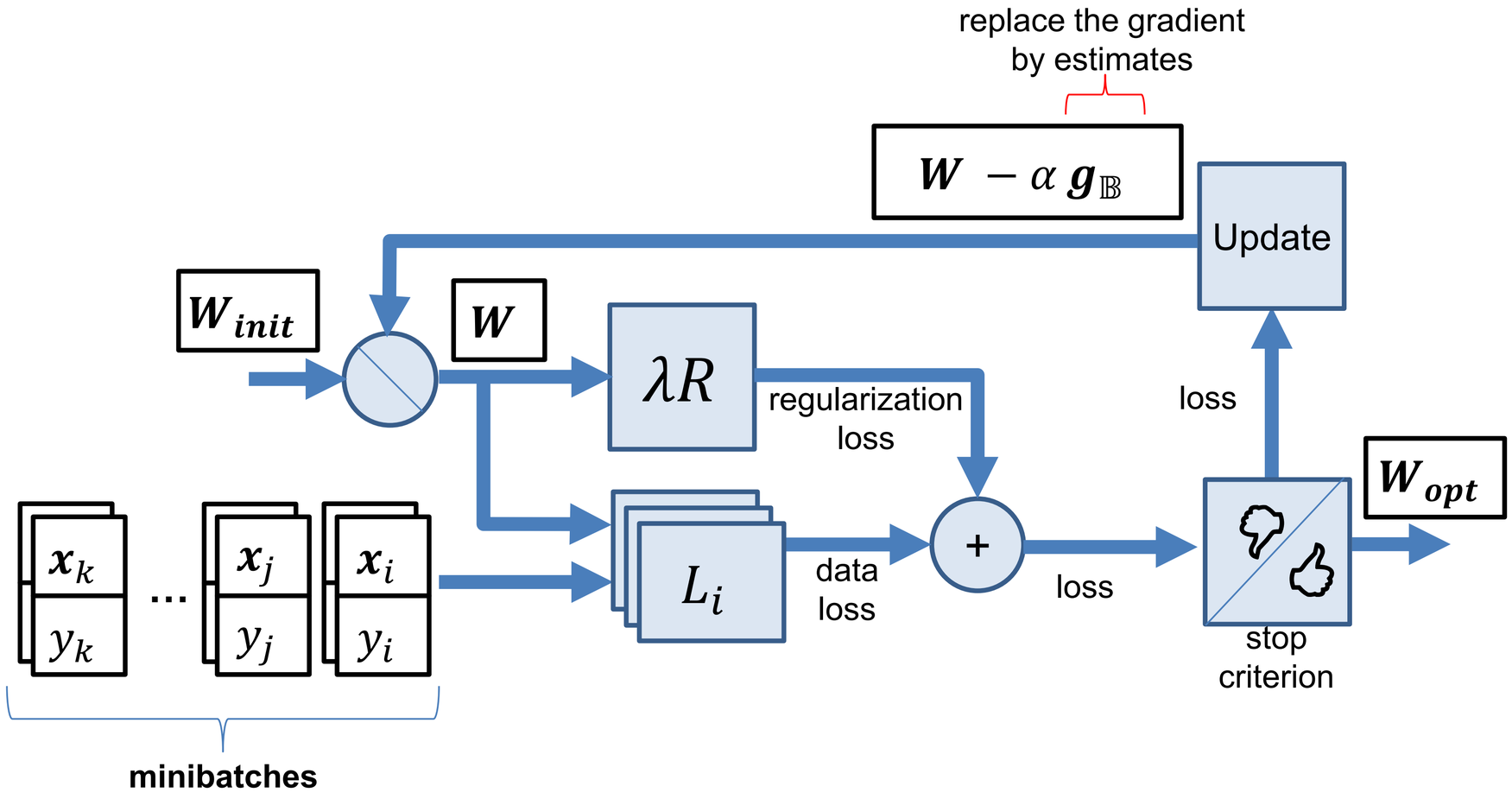
$$\boldsymbol{g}_{\mathbb{B}} = \frac{1}{m} \sum_{\boldsymbol{x}_i \in \mathbb{B}} \nabla_{\boldsymbol{W}} L_i[f(\boldsymbol{W}, \boldsymbol{x}_i), y_i] + \lambda \nabla_{\boldsymbol{W}} R(\boldsymbol{W})$$

and updated $\boldsymbol{W}$ as $\boldsymbol{W} \longleftarrow \boldsymbol{W} - \alpha \boldsymbol{g}_{\mathbb{B}}$

Gradient Descent



Stochastic Gradient Descent



Problems

What if it is not enough?

… if f does not delineate the classes properly?

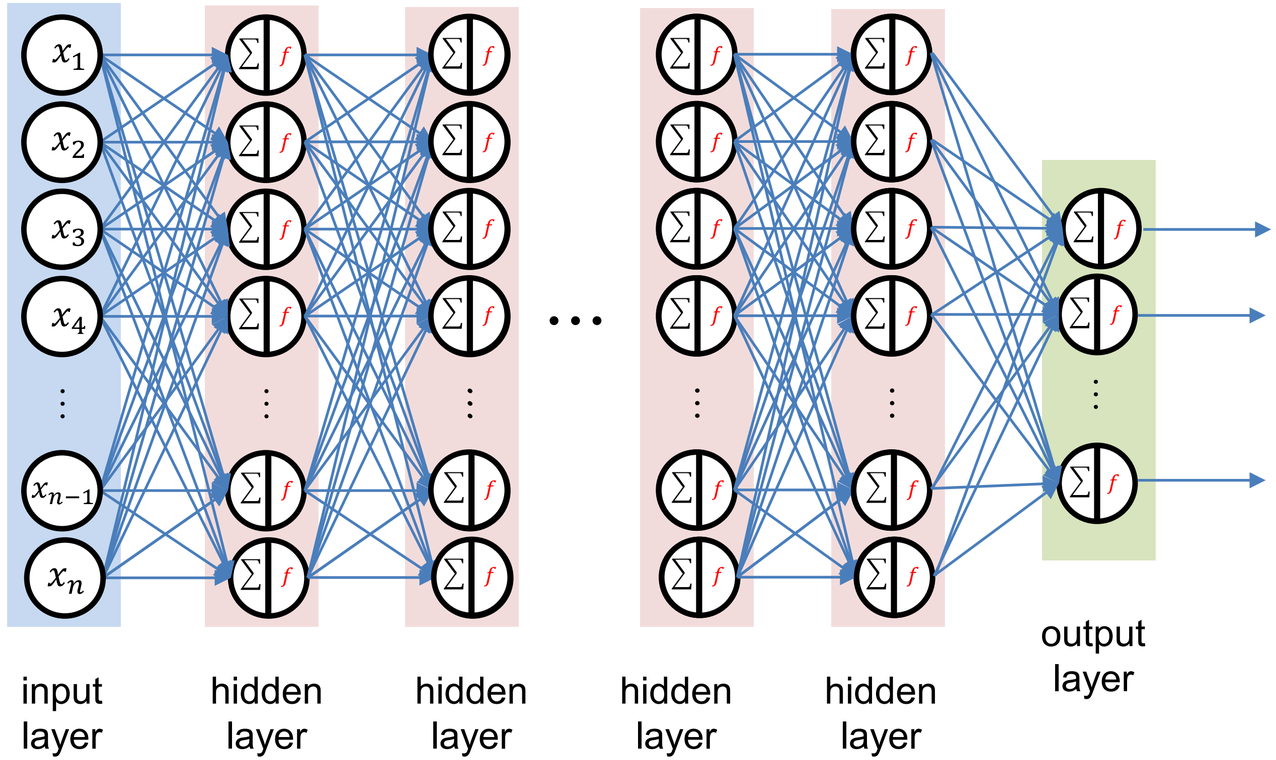
What if it is not enough?

… if f does not delineate the classes properly?

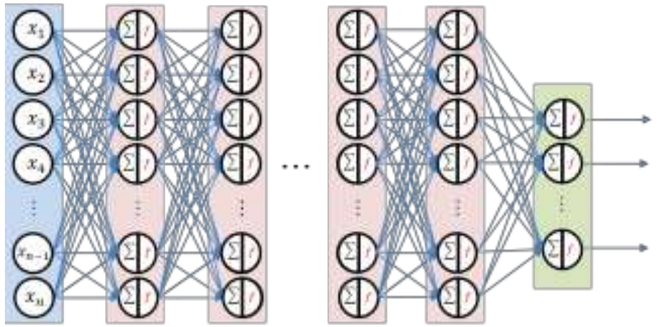
- By cascading multiple non linear layers we are capable of learning increasingly complex functions.
- So the depth of the network is related to the **model capacity**, i.e., the level of complexity that can be captured by the model.

Multilayer Perceptron

$$f(\boldsymbol{W}^T\,\boldsymbol{x}) = f_d\left(\boldsymbol{W}_d^T f_{d-1}\left(\boldsymbol{W}_{d-1}^T f_{d-2}\left(\ldots \boldsymbol{W}_2^T f_1(\boldsymbol{W}_1^T \boldsymbol{x})\right)\right)\right)$$

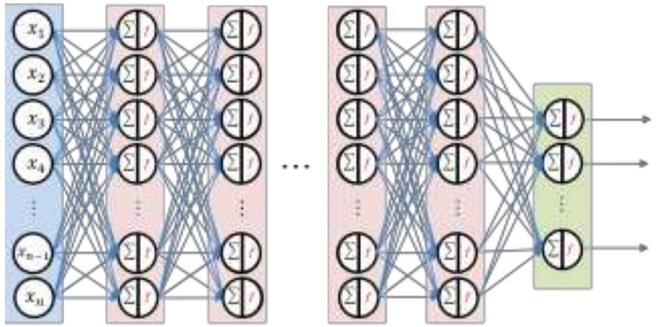


What is a MLP?



- Feed forward neural network.
- Neural networks “inspired by biology“.
- Core idea: concatenate multiple simple mappings to get one powerful mapping.
- Multiple simple steps more powerful than one complex step.
- Keep everything (mostly) differentiable.
- Train by performing gradient descent on classification error.

Problem with the MLP



- Each neuron is **fully connected** to all neurons in the previous layer.
- Neurons in a single layer are independent.
- Neurons in a single layer do not share any connections.
- Consider a MLP for classifying a 200×200 RGB image.
- The number of parameters in a single fully connected layer will be:

$$14.4 \times 10^9$$

Next Lecture

Lecture 3

Convolutional Neural Networks

See you next class!