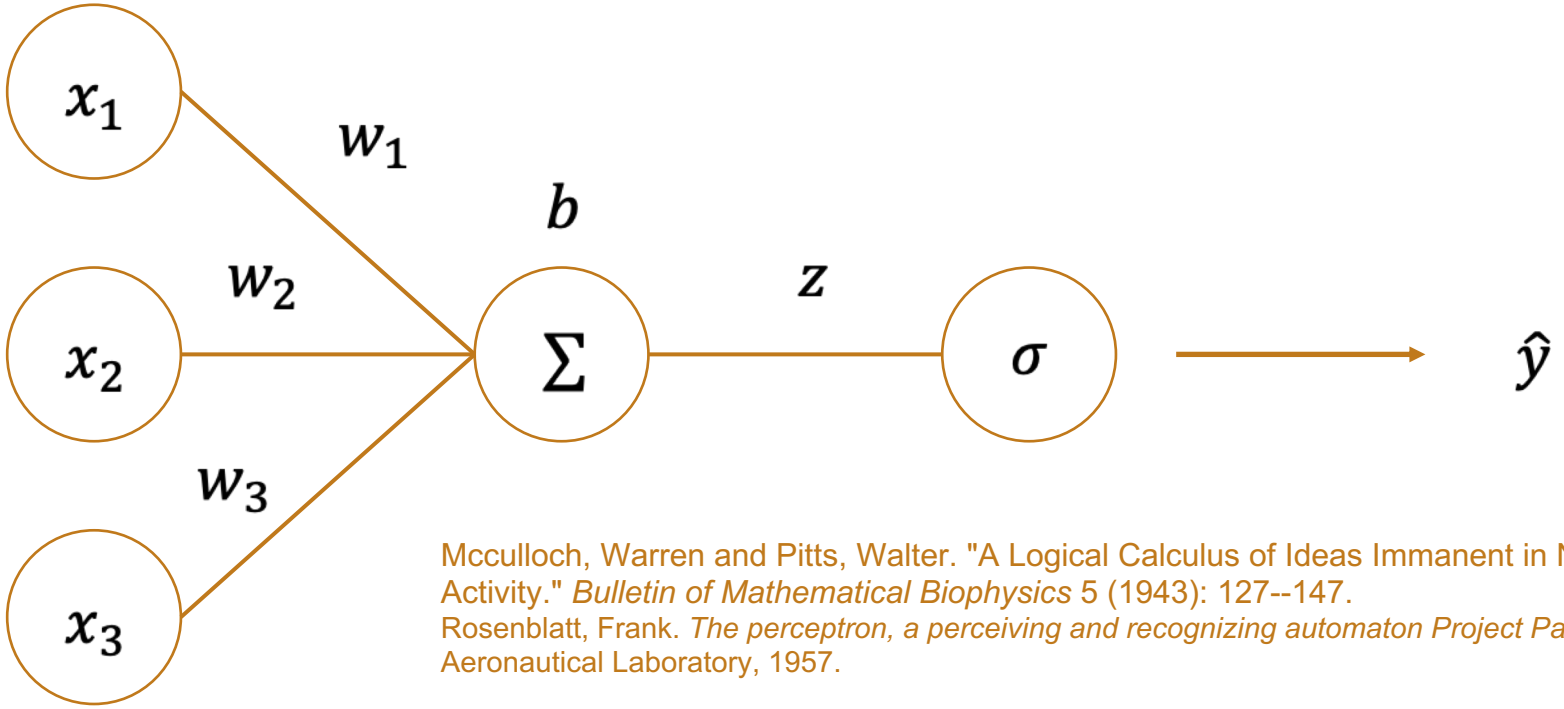
Introduction to Neural Networks: Perceptrons

Guillem & Roderic, Summer 2025

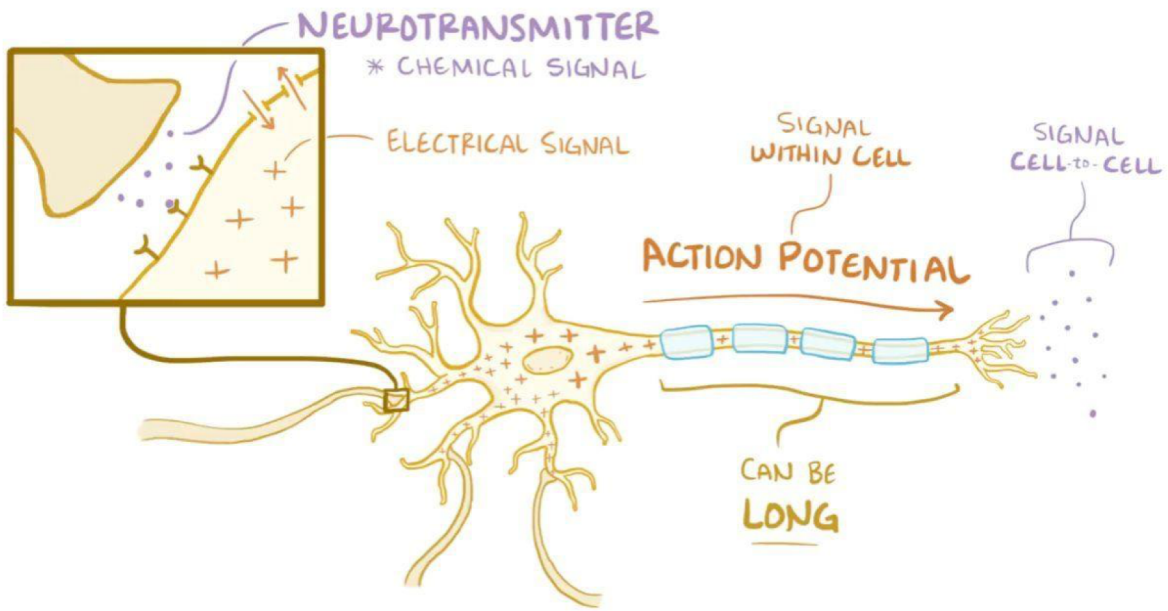
Single neuron model: The Perceptron

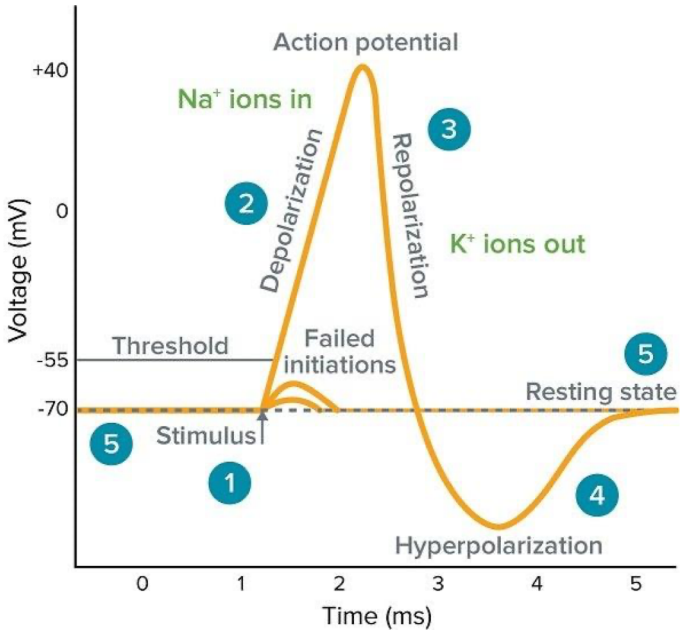
- McCulloch and Pitts (1943) – Description of the model
- Frank Rosenblatt (1957) – Machine implementation



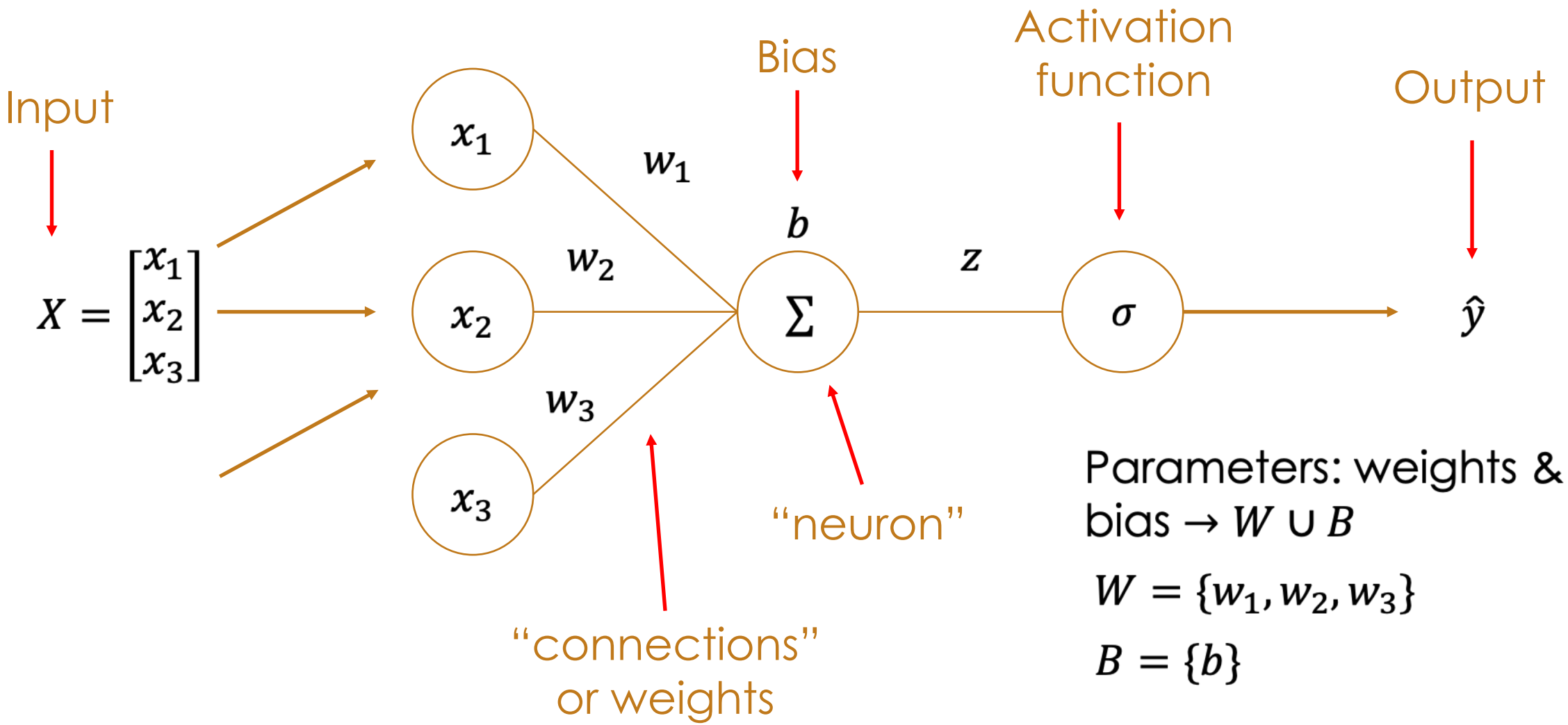
Mcculloch, Warren and Pitts, Walter. "A Logical Calculus of Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5 (1943): 127--147.
Rosenblatt, Frank. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

Neuron model





Single neuron model: The Perceptron

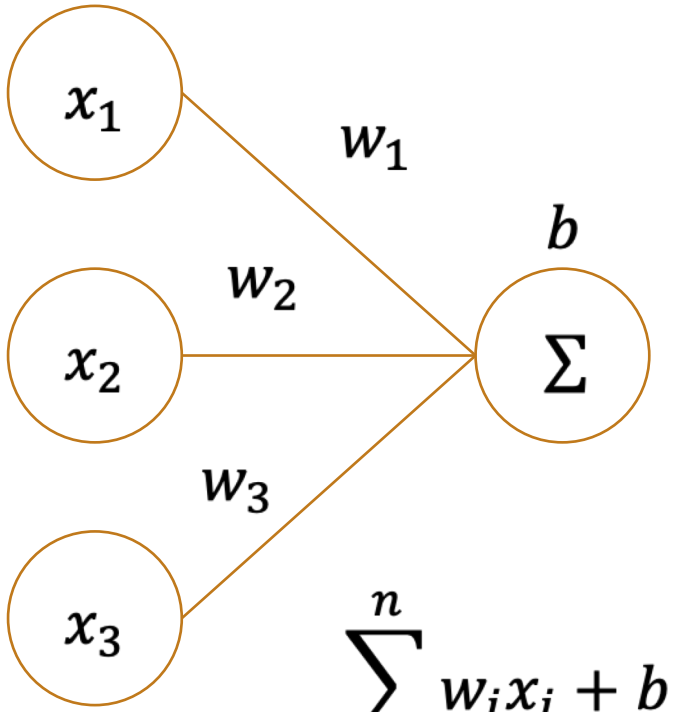


Parameters: weights & bias → $W \cup B$

$W = \{w_1, w_2, w_3\}$

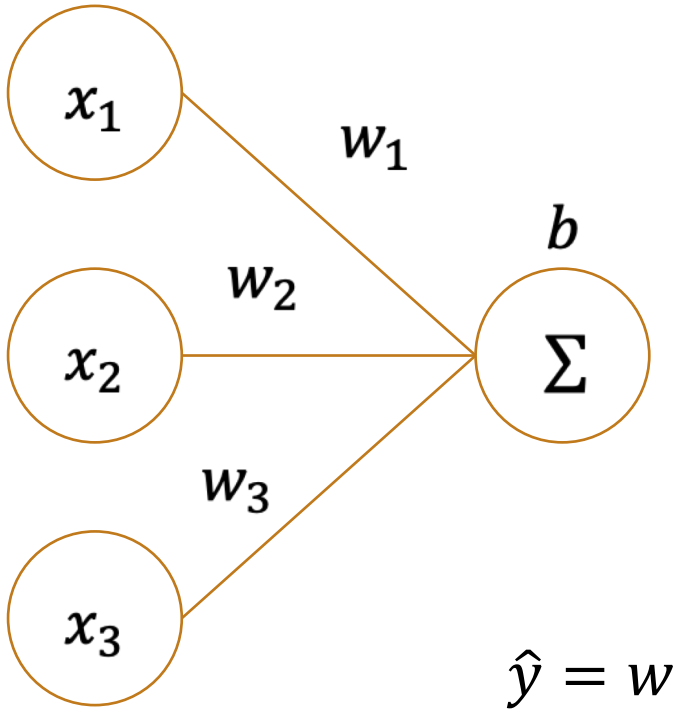
$B = \{b\}$

Single neuron model: The Perceptron



$$\sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Single neuron model: The Perceptron



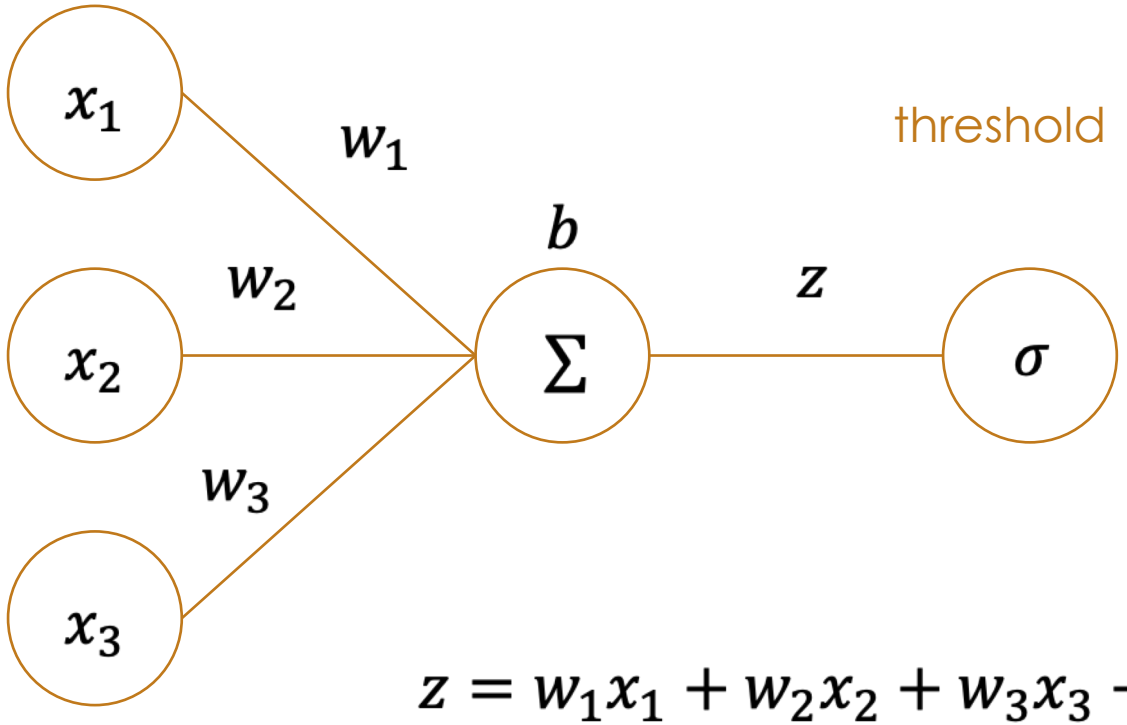
Linear combination of inputs scaled by the weights

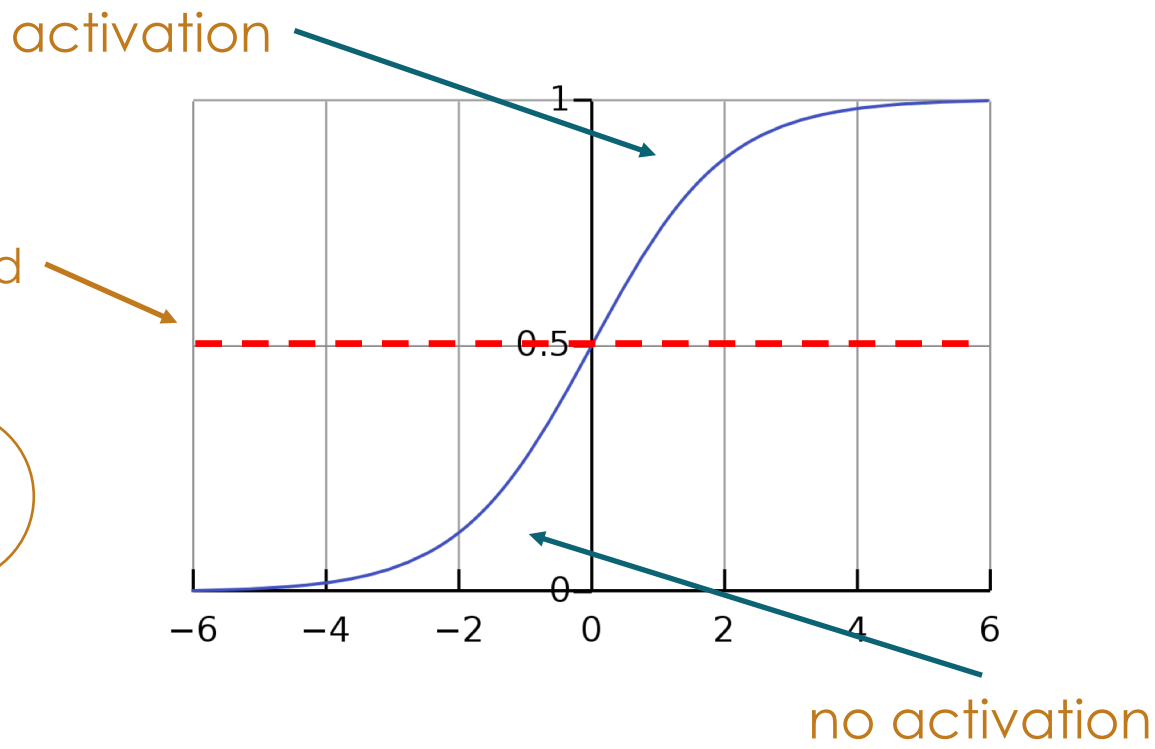
Mathematically equivalent to linear regression

How many trainable parameters does this perceptron have?

$$\hat{y} = w^T x + b \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \qquad b = b$$

Single neuron model: The Perceptron





$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

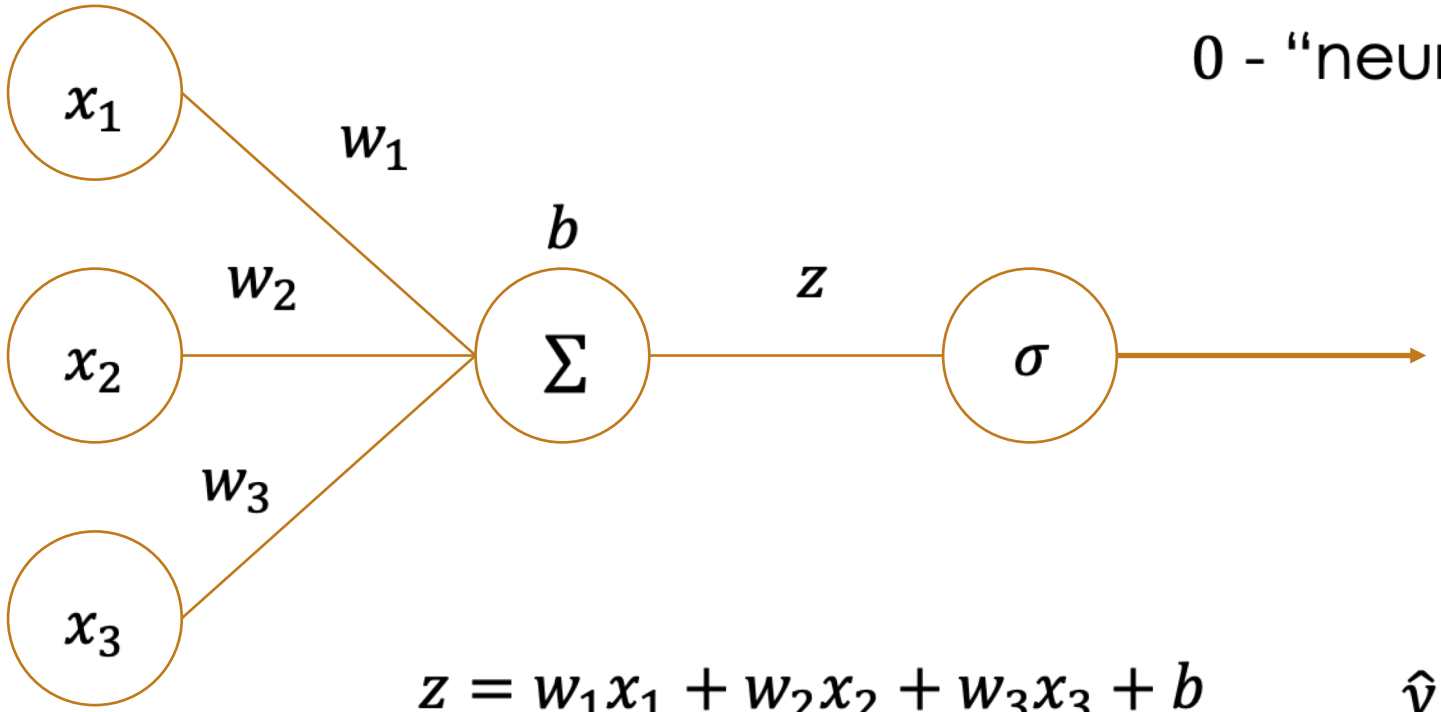
$$\hat{y} = z = \sigma(w_1x_1 + w_2x_2 + w_3x_3)$$

Sigmoid activation

Single neuron model: The Perceptron

1 - “neuron activates”

0 - “neuron doesn’t activate”



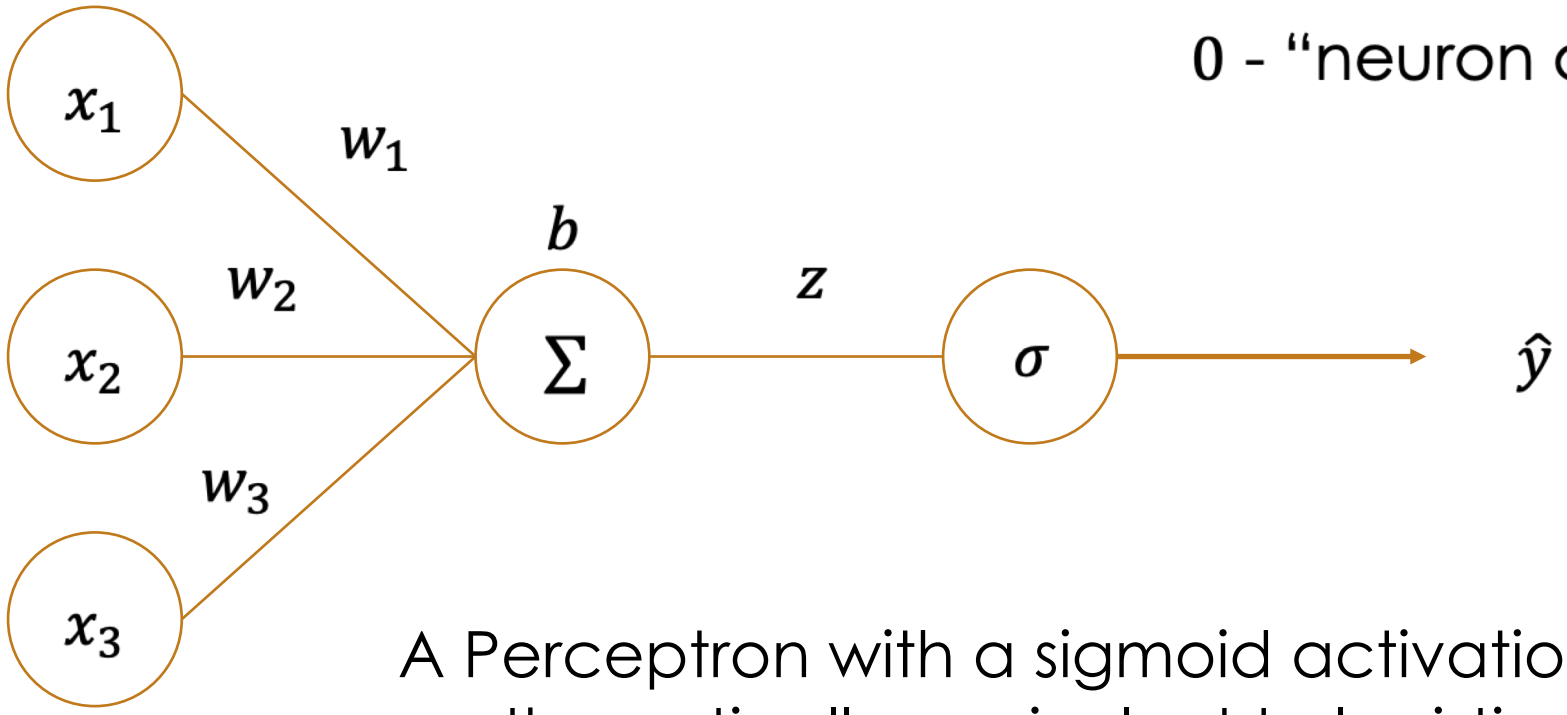
$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Single neuron model: The Perceptron

1 - "neuron activates"

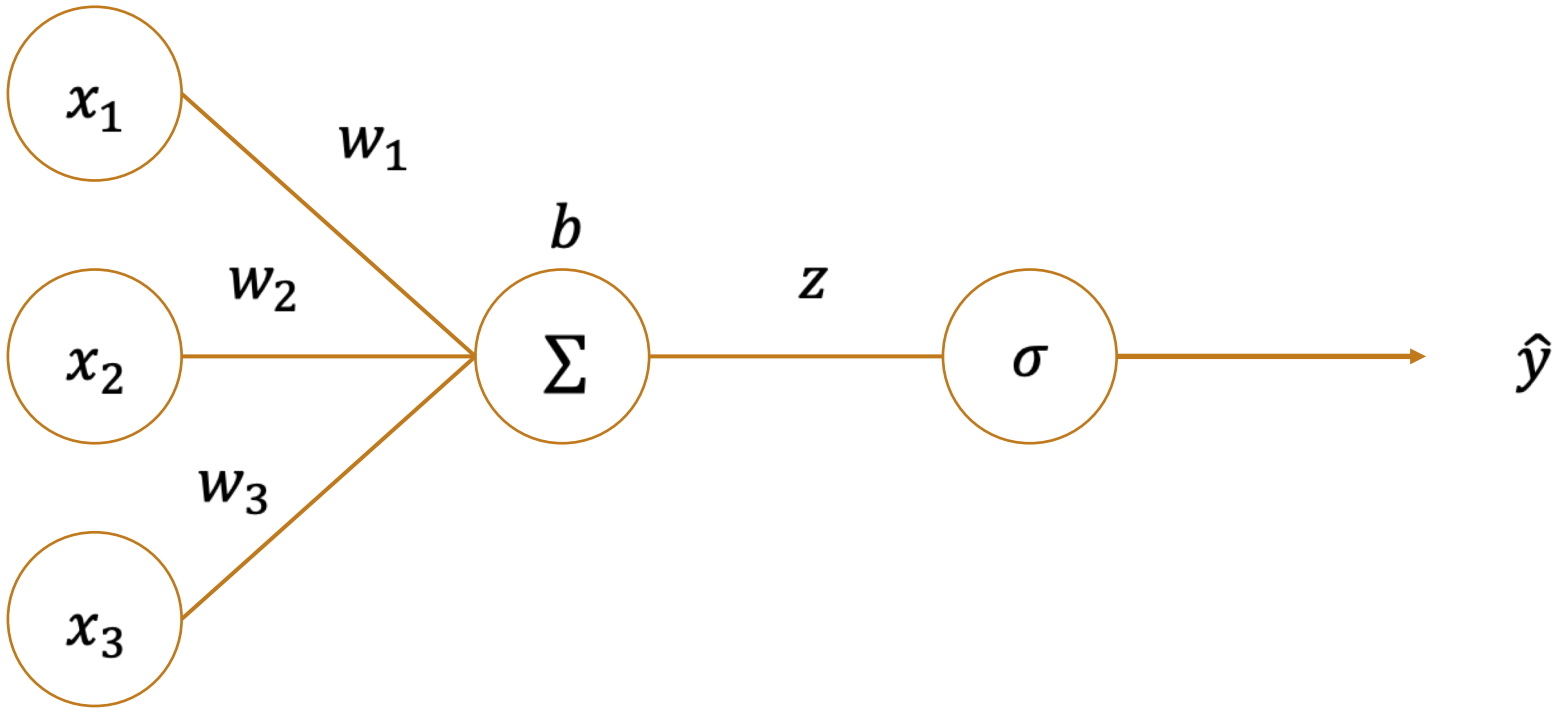
0 - "neuron doesn't activate"



A Perceptron with a sigmoid activation function is mathematically equivalent to logistic regression

Single neuron model: The Perceptron

- Example: predicting whether someone has a sleep disorder given the quality of the sleep, the daily amount of exercise and the stress level

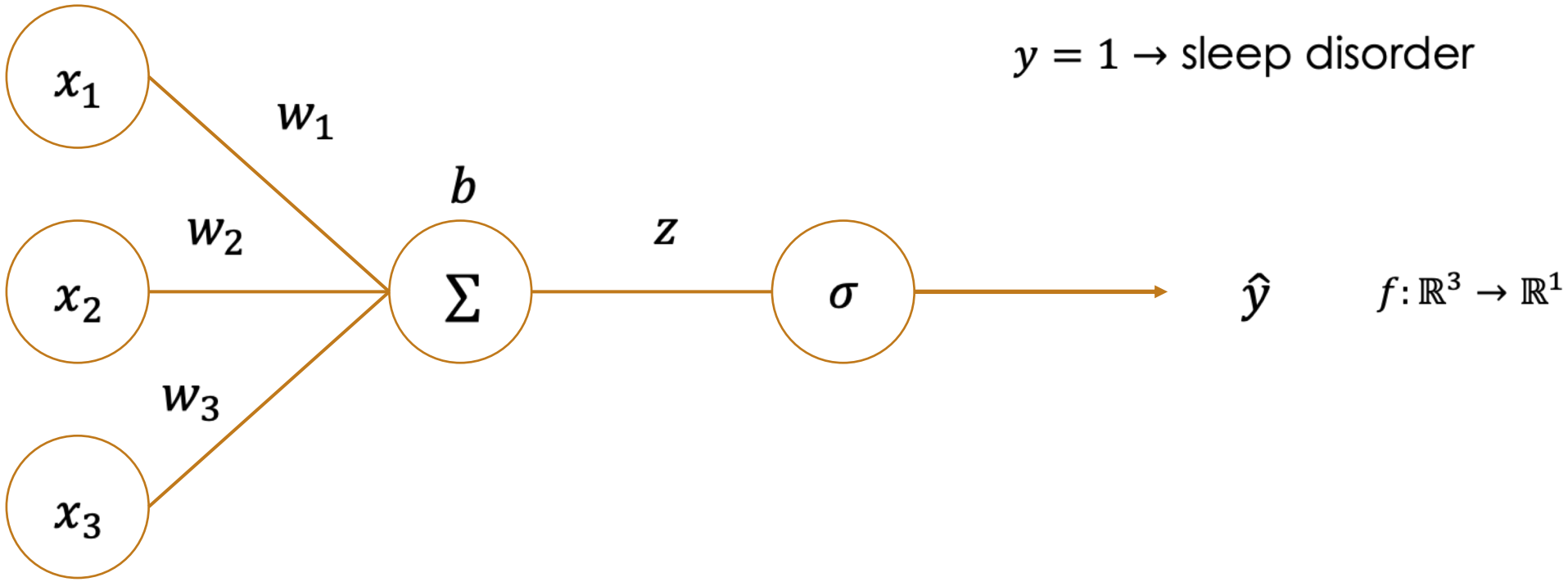


Single neuron model: The Perceptron

x_1 = sleep quality (1 - 10) x_2 = exercise (min/daily) x_3 = stress level (1 - 10)

$y = 0 \rightarrow$ normal

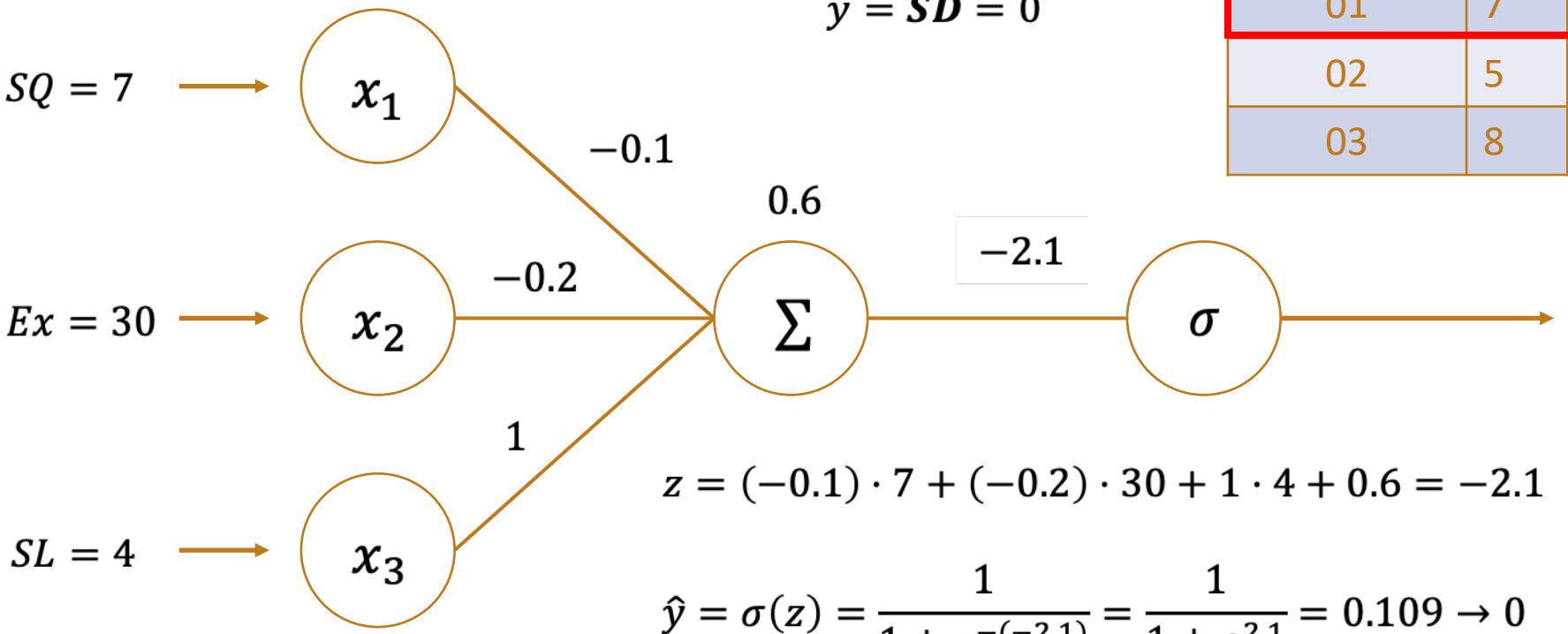
$y = 1 \rightarrow$ sleep disorder



Single neuron model: The Perceptron

x_1 = sleep quality (1 - 10) x_2 = exercise (min/daily) x_3 = stress level (1 - 10)

Patient ID	SQ	Ex	SL	**SD**
01	7	30	4	0
02	5	20	8	1
03	8	60	8	0



$$z = (-0.1) \cdot 7 + (-0.2) \cdot 30 + 1 \cdot 4 + 0.6 = -2.1$$

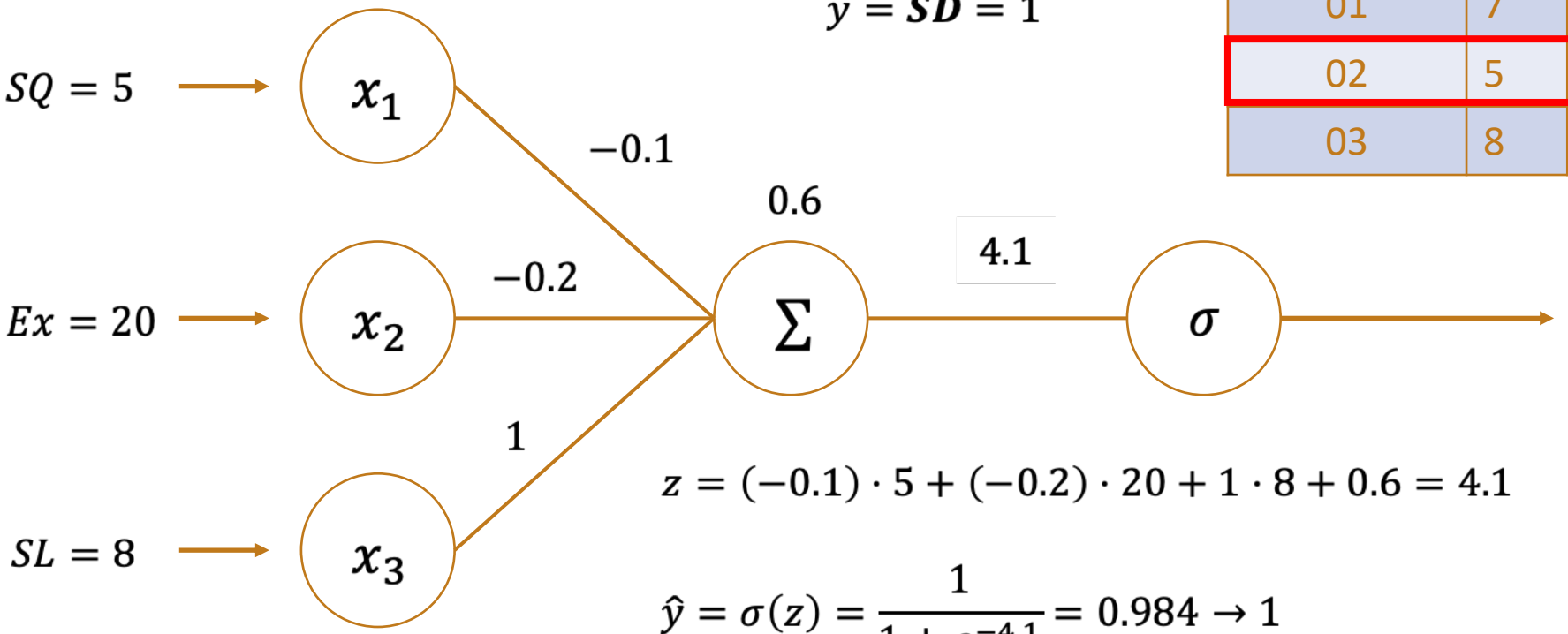
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-(-2.1)}} = \frac{1}{1 + e^{2.1}} = 0.109 \rightarrow 0$$

$w_1 = -0.1$
$w_2 = -0.2$
$w_3 = 1$
$b = 0.6$

Single neuron model: The Perceptron

x_1 = sleep quality (1 - 10) x_2 = exercise (min/daily) x_3 = stress level (1 - 10)

Patient ID	SQ	Ex	SL	**SD**
01	7	30	4	0
02	5	20	8	1
03	8	60	8	0



$$z = (-0.1) \cdot 5 + (-0.2) \cdot 20 + 1 \cdot 8 + 0.6 = 4.1$$

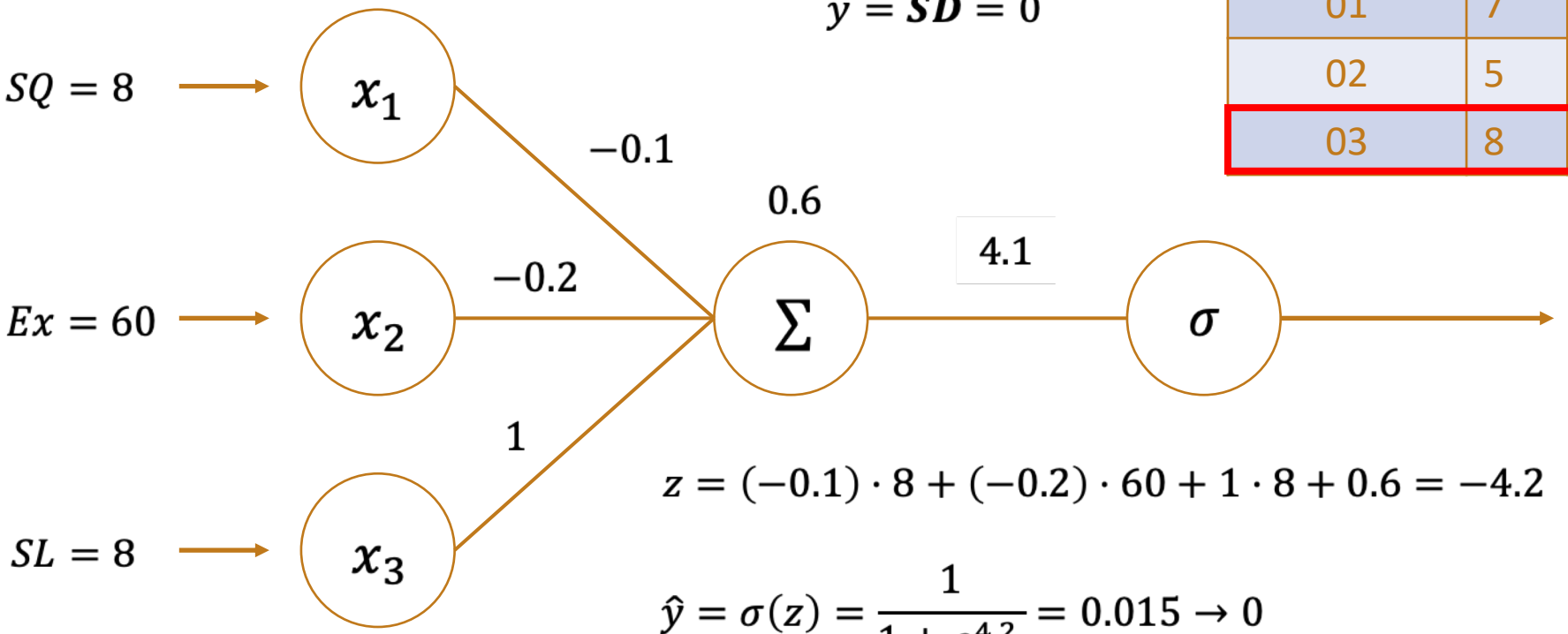
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-4.1}} = 0.984 \rightarrow 1$$

$w_1 = -0.1$
$w_2 = -0.2$
$w_3 = 1$
$b = 0.6$

Single neuron model: The Perceptron

x_1 = sleep quality (1 - 10) x_2 = exercise (min/daily) x_3 = stress level (1 - 10)

Patient ID	SQ	Ex	SL	**SD**
01	7	30	4	0
02	5	20	8	1
03	8	60	8	0



$$z = (-0.1) \cdot 8 + (-0.2) \cdot 60 + 1 \cdot 8 + 0.6 = -4.2$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{4.2.}} = 0.015 \rightarrow 0$$

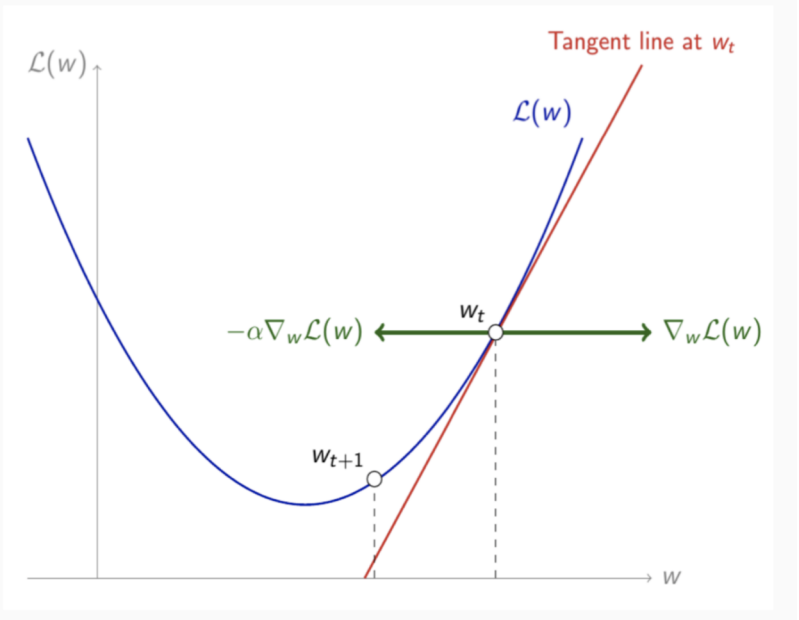
$w_1 = -0.1$
$w_2 = -0.2$
$w_3 = 1$
$b = 0.6$

Training a Perceptron

- How do we find w_1, w_2, w_3 and b?

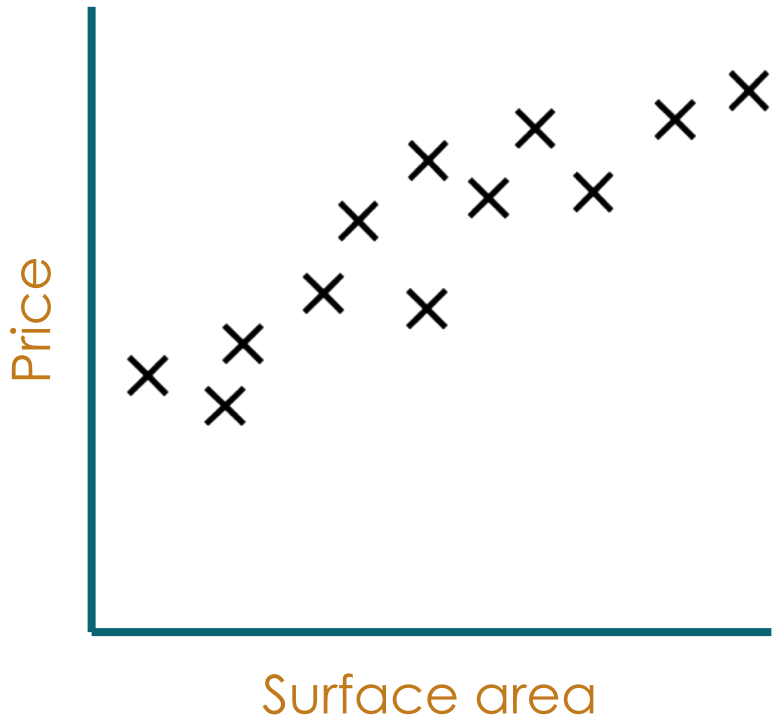
- Iterative process: using the training data, we will gradually adjust the values of the parameters such that we minimize the error in the predictions

- Gradient descent



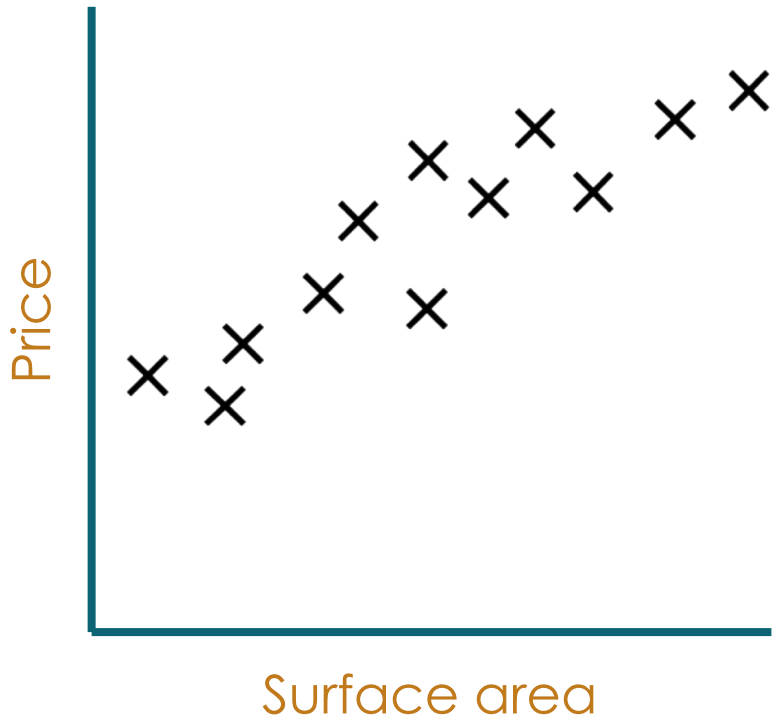
House price model

- A model to predict the price of a house based on its surface area in square meters
- Linear model: $y = ax + b$
- How do we find a and b ?
- We will use the observations
- Iterative process

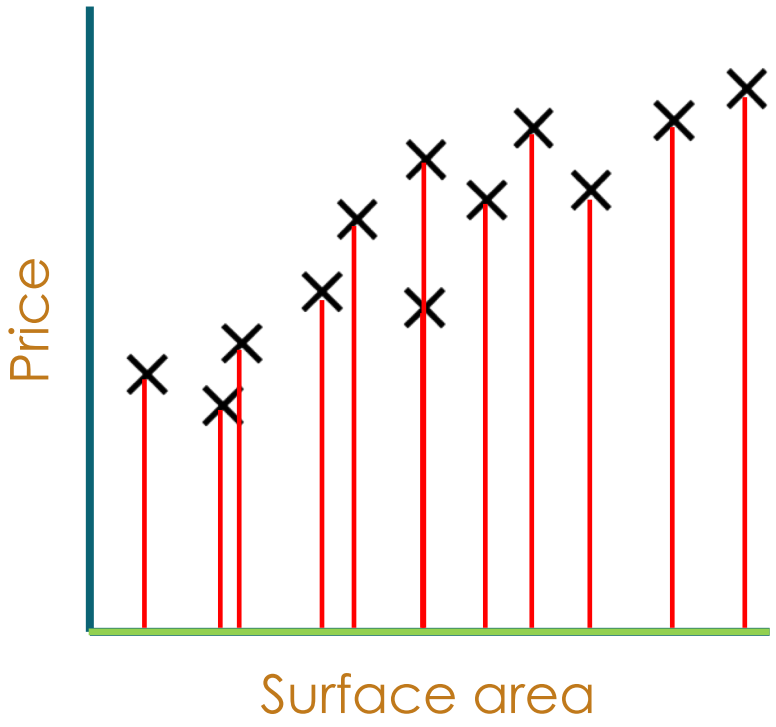


House price model

1. Initial guess: $a = 0$, $b = 0$
2. Compute the error of the model
3. Slightly increase or decrease the value of a such that the error of the new model is lower
4. Do the same for b
5. Repeat step 2 until the error no longer decreases

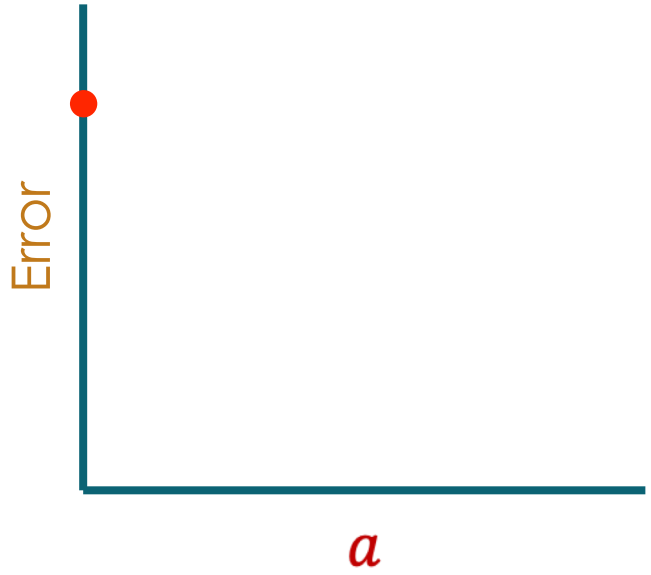


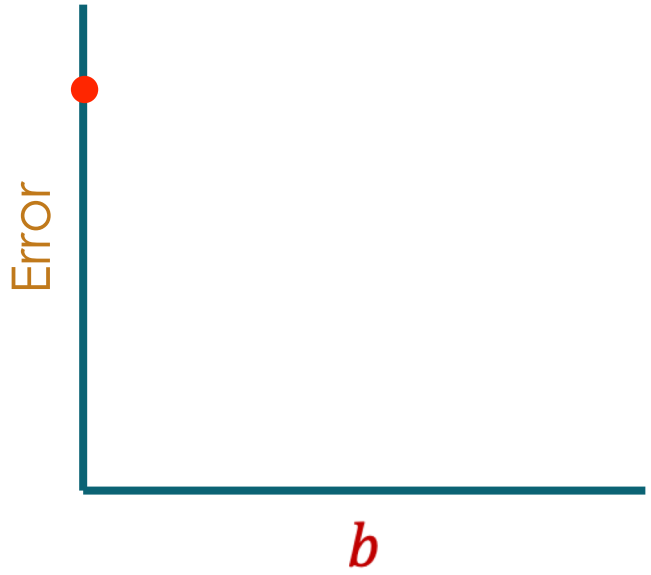
House price model



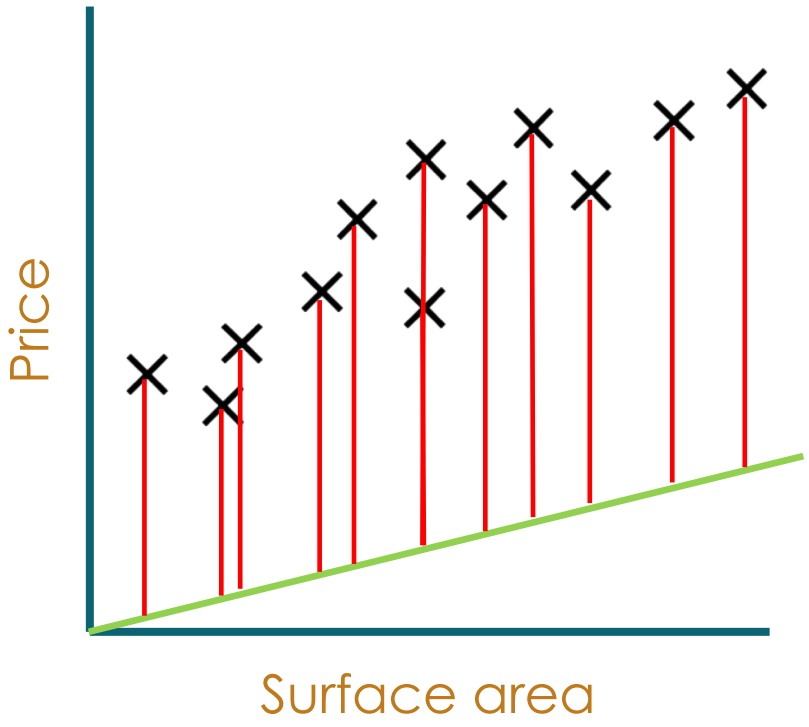
$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



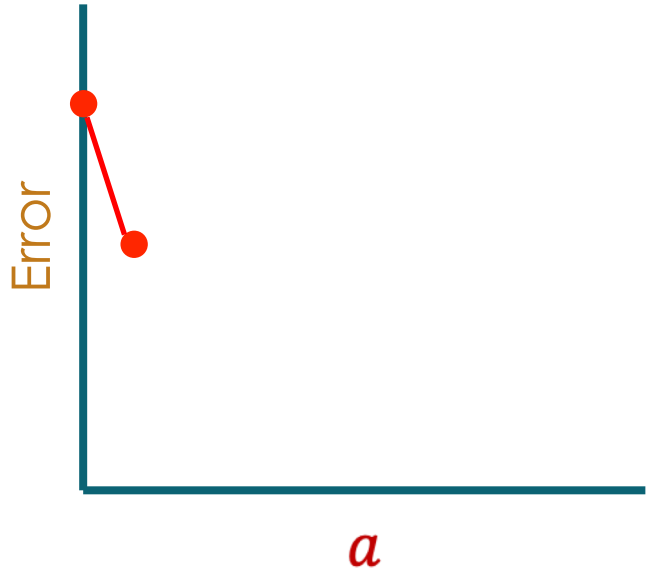


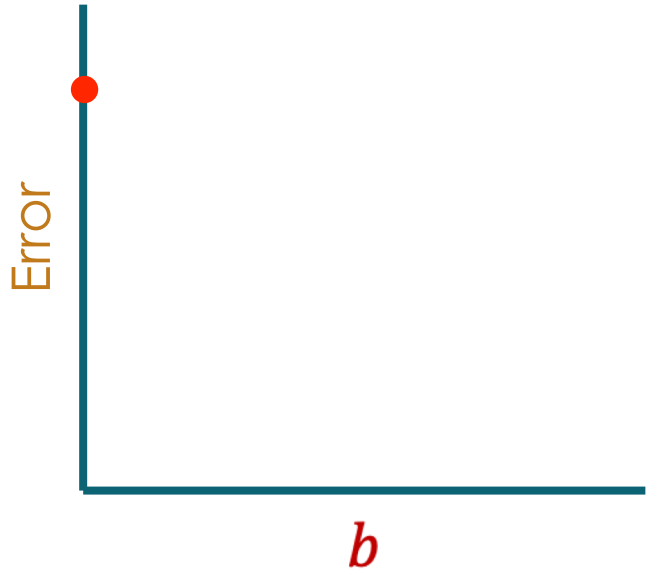
House price model



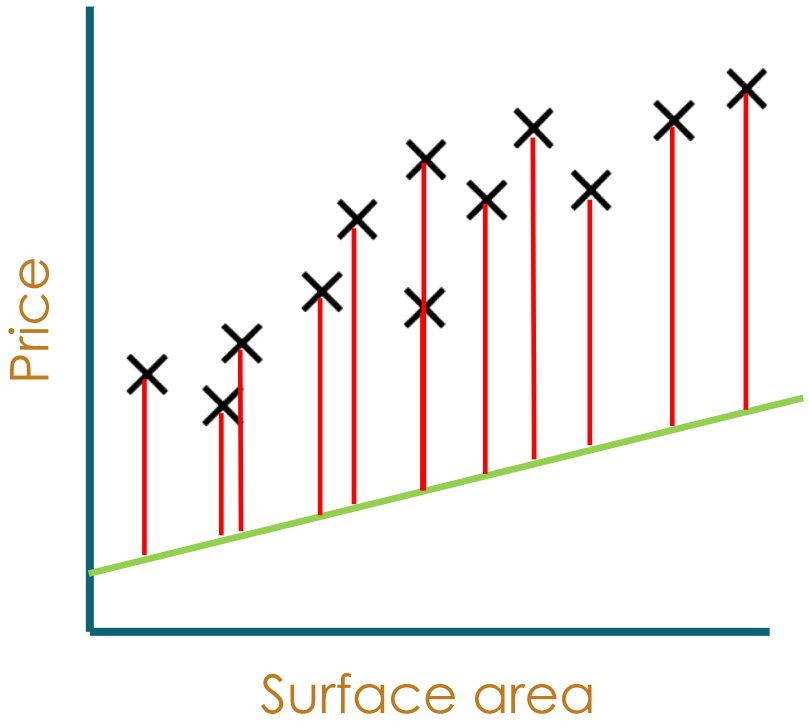
$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



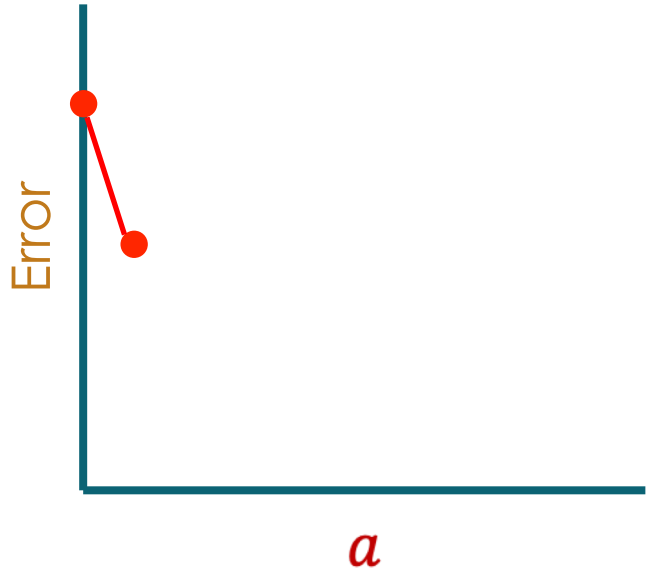


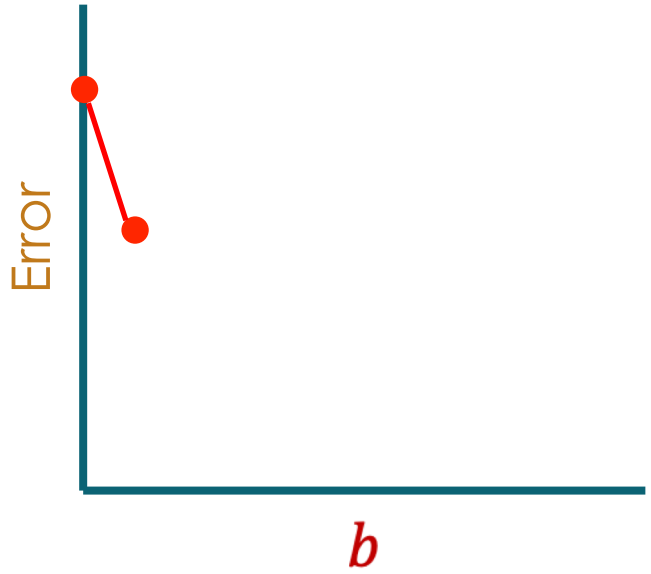
House price model



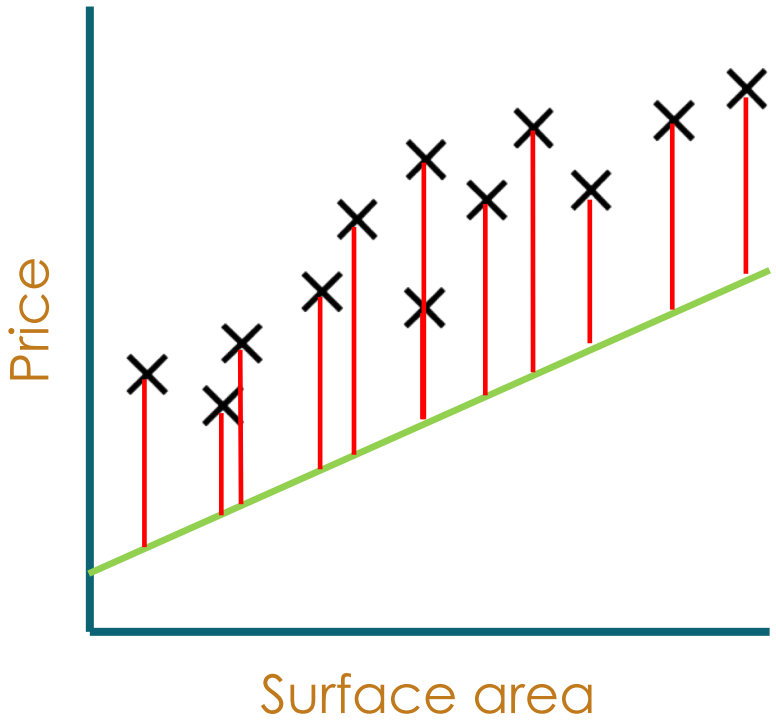
$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



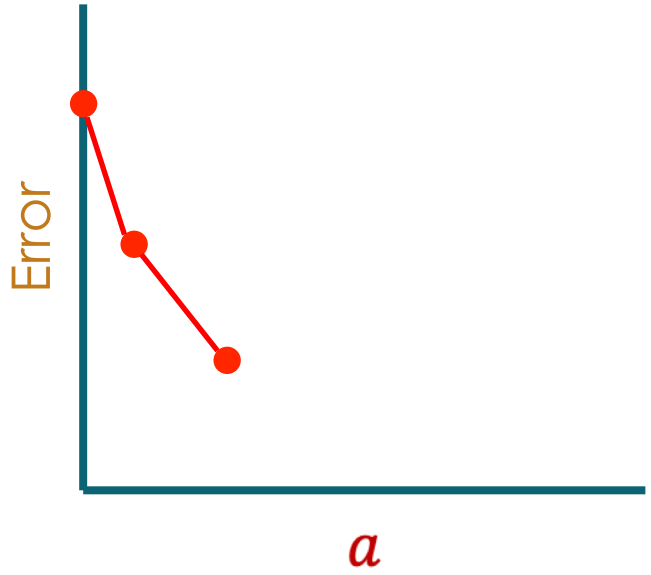


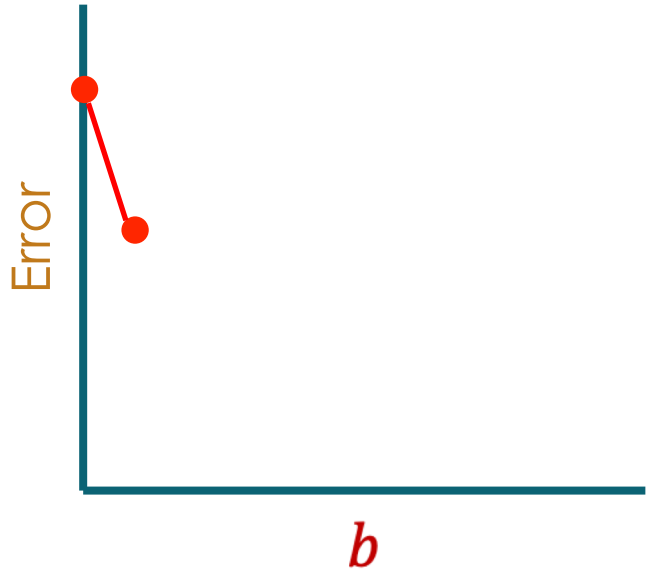
House price model



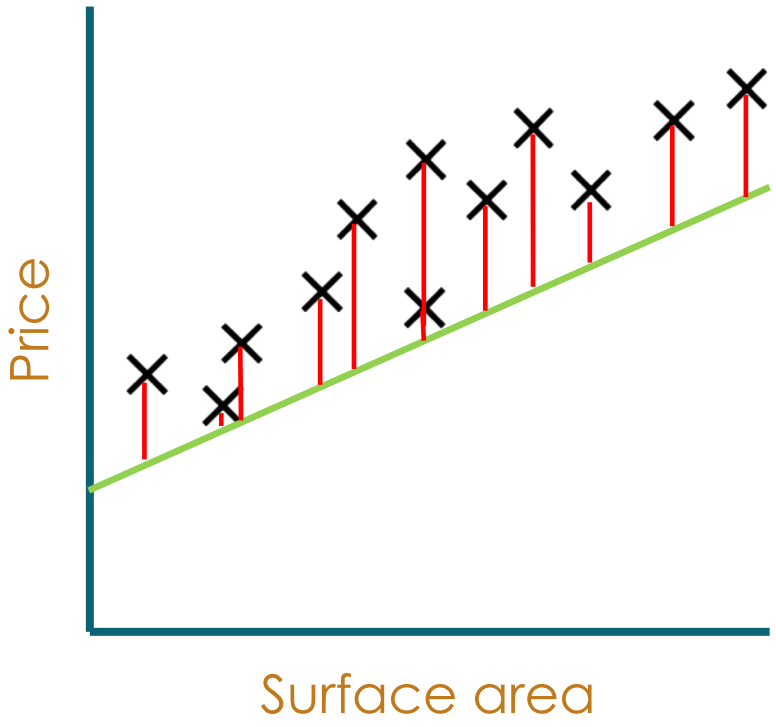
$$\hat{y} = ax + b$$

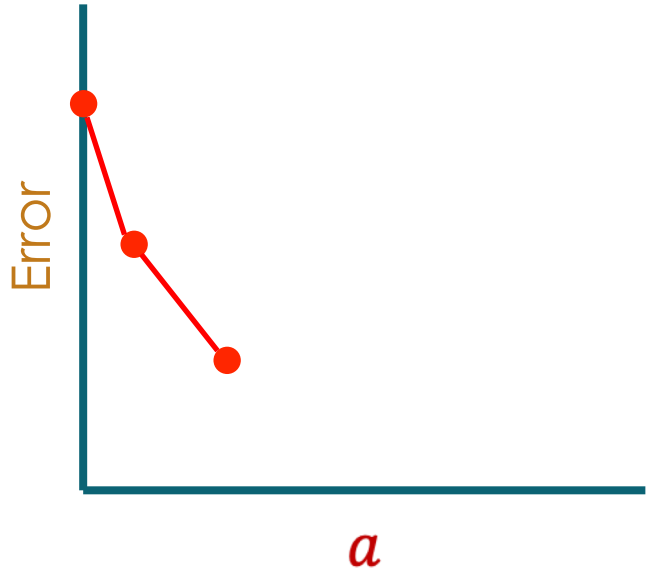
$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

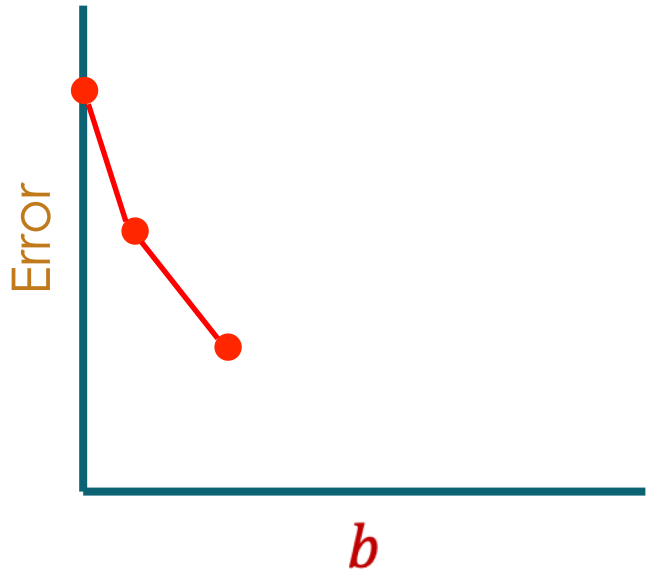




House price model



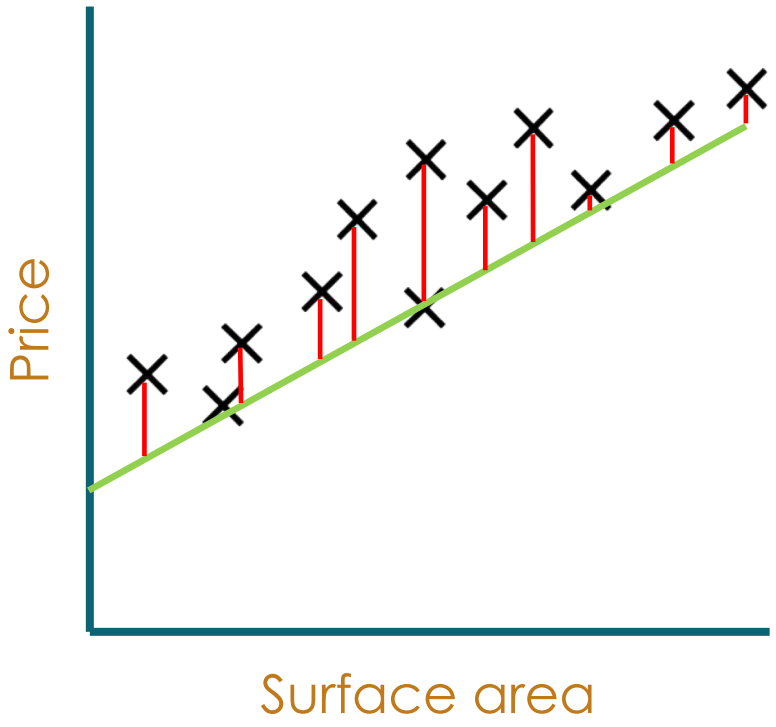


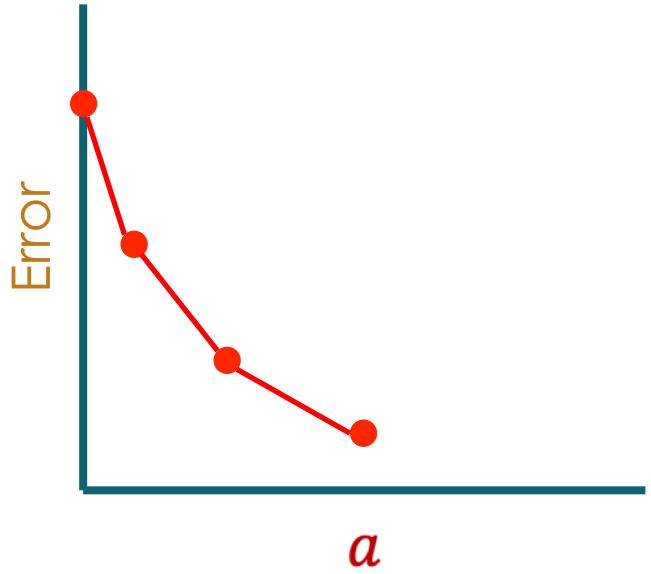


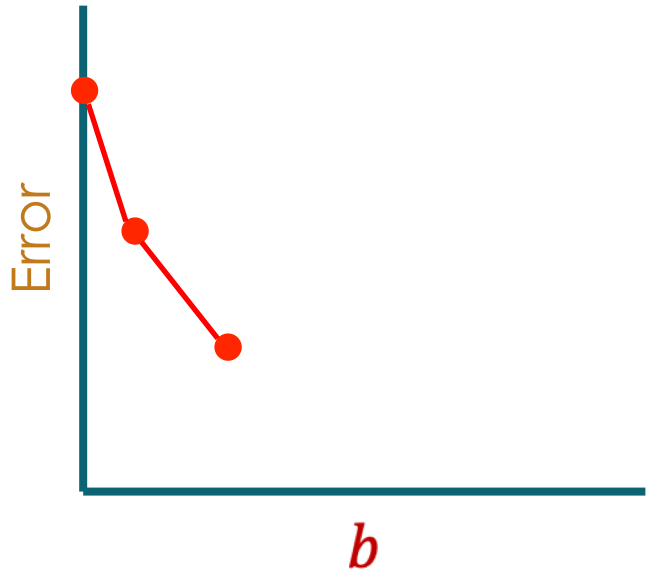
$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

House price model



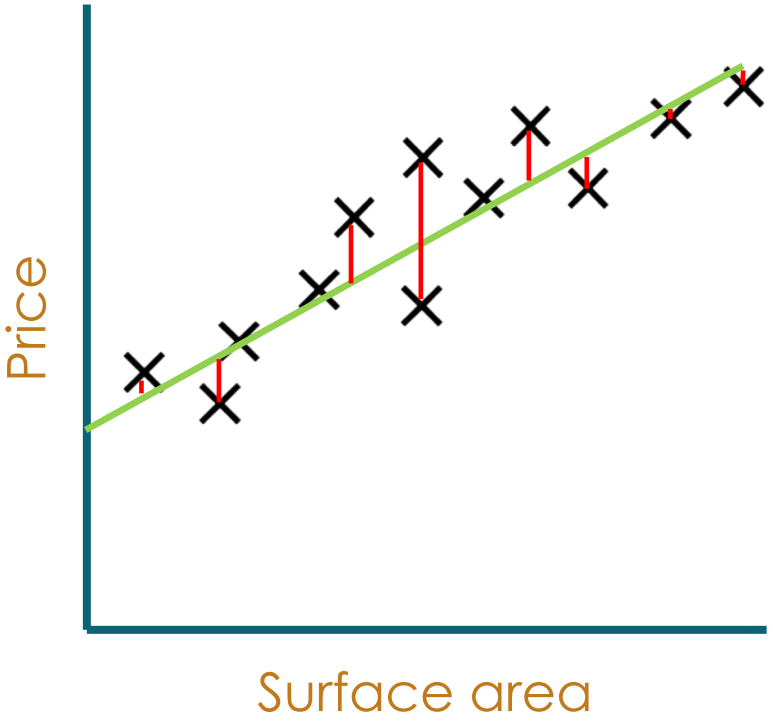


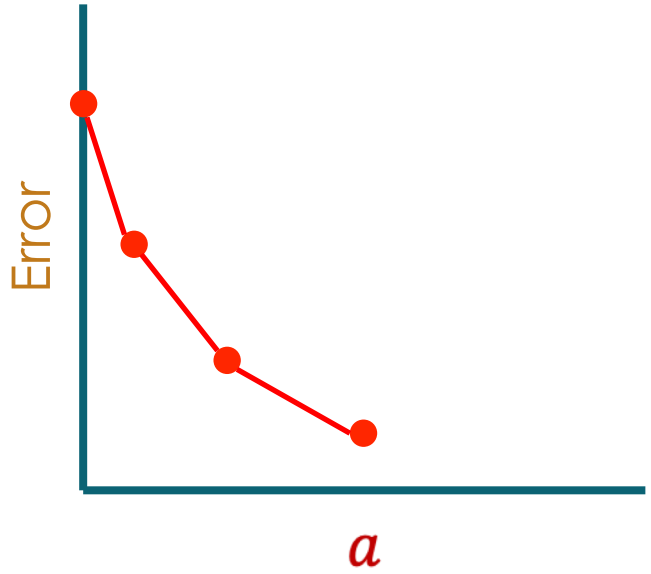


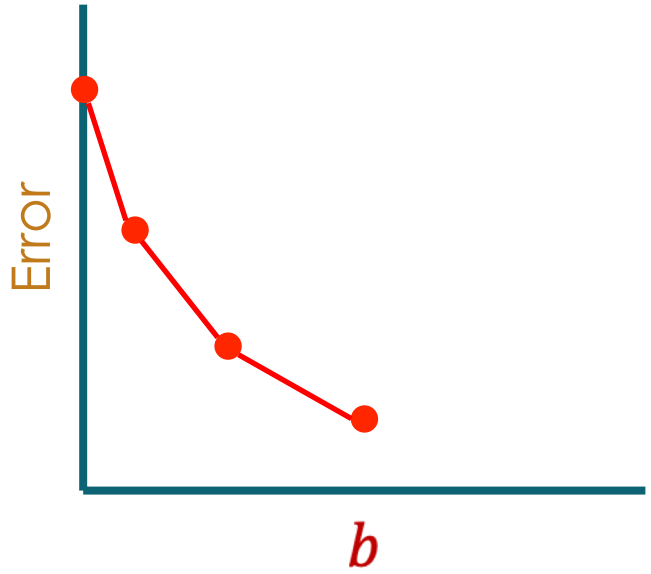
$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

House price model





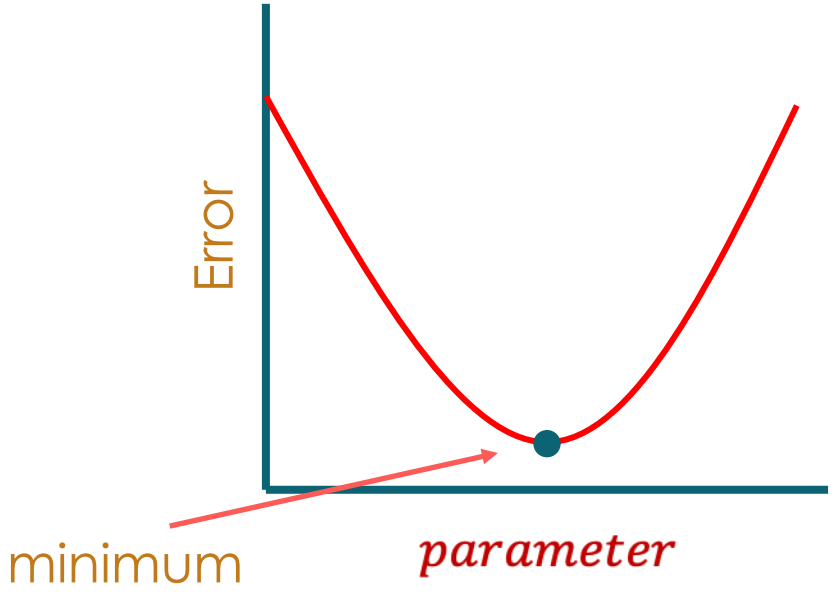


$$\hat{y} = ax + b$$

$$Error(MSE) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

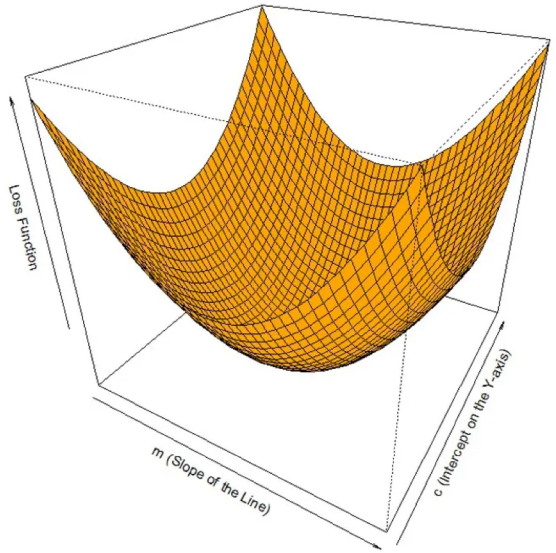
House price model

- Optimization problem: minimize the loss function (MSE)
- Simple linear regression with ordinary least squares (OLS)
- Unique global minimum



$$\widehat{\alpha} = \bar{y} - (\widehat{\beta}\,\bar{x}),$$

$$\widehat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

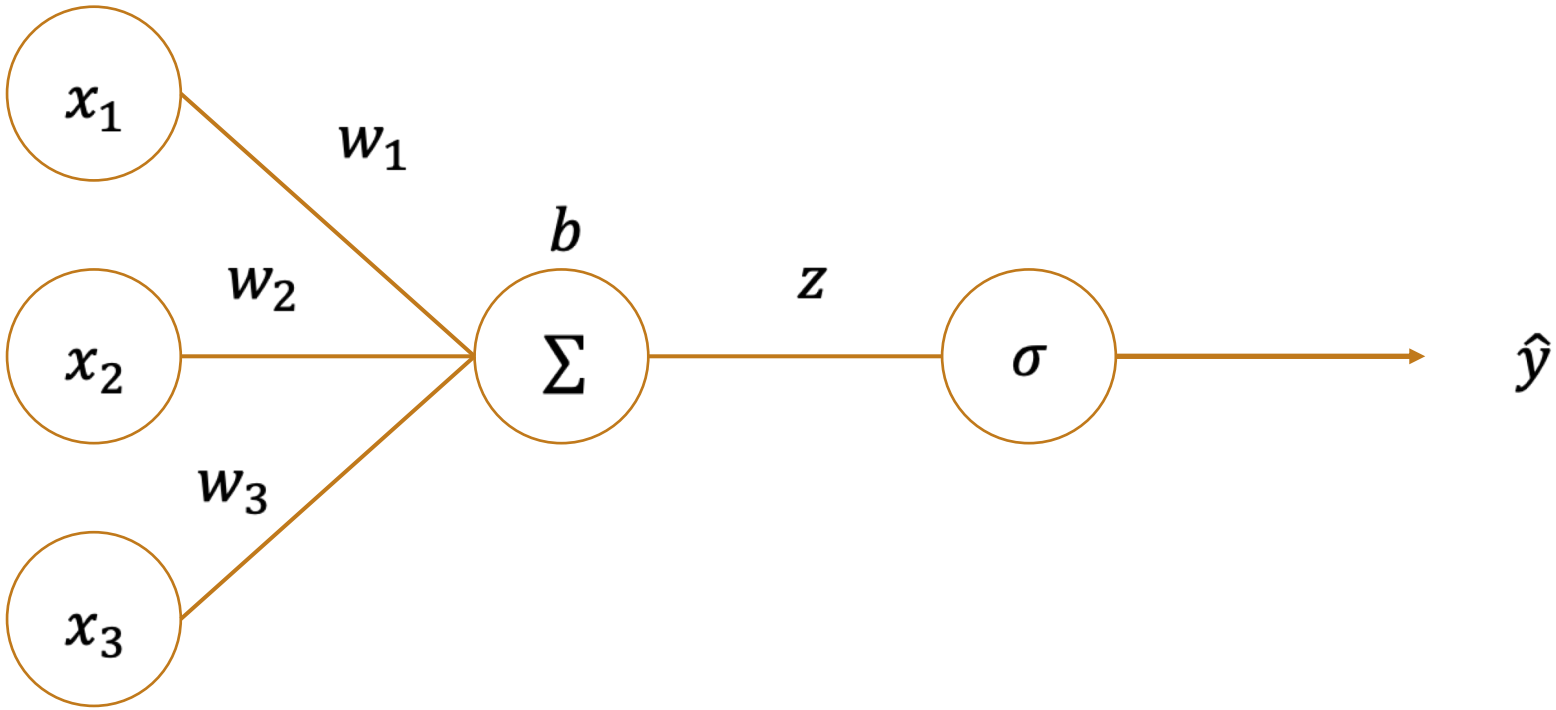


Training algorithm

- Randomly initialize the weights and bias of the Perceptron
- Select a random sample from your training set and use it as input to your model
- Compare the output of your model to the known target value
- Modify the parameters of the model such that the output gets closer to the target value
- Repeat step two with a new sample until all the samples in the training set have been seen by the model
- Repeat steps 2-5 until the error does not decrease anymore or a maximum number of iterations is reached
- Profit!

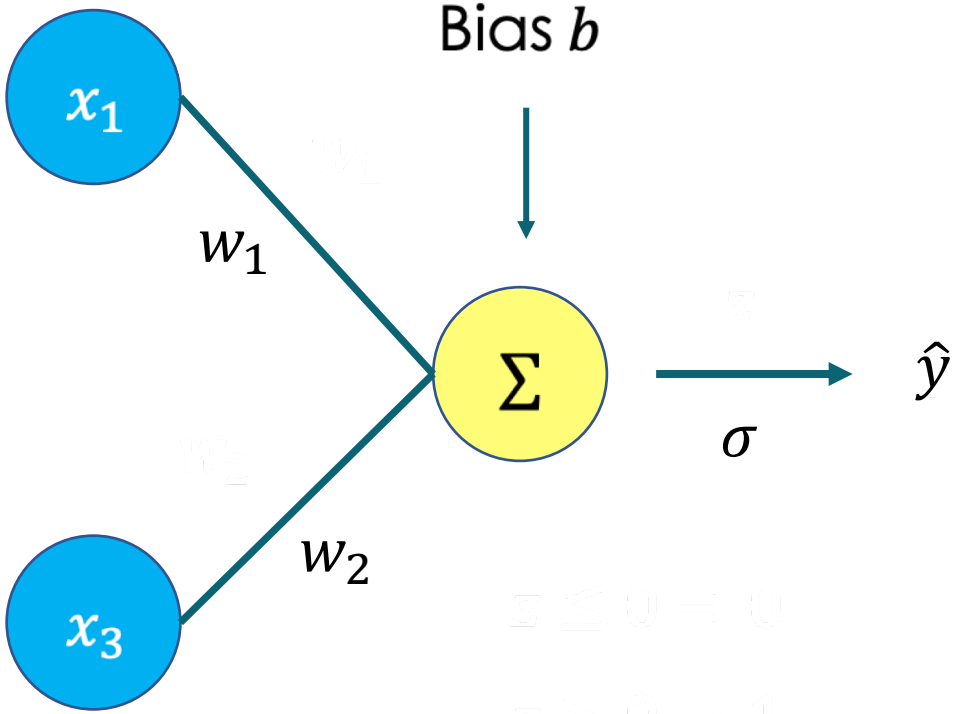
Training a perceptron

- Once our model is trained, we will have found the set of parameters conformed by the weights and bias
- We can then make predictions on new data



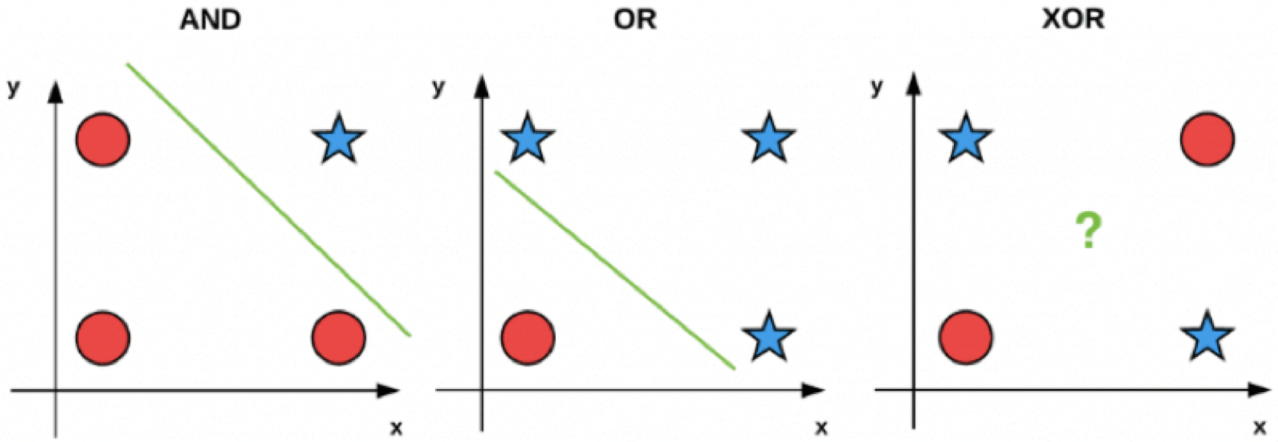
Multilayer Preceptrons (aka Neural Networks!)

The perceptron and the XOR problem

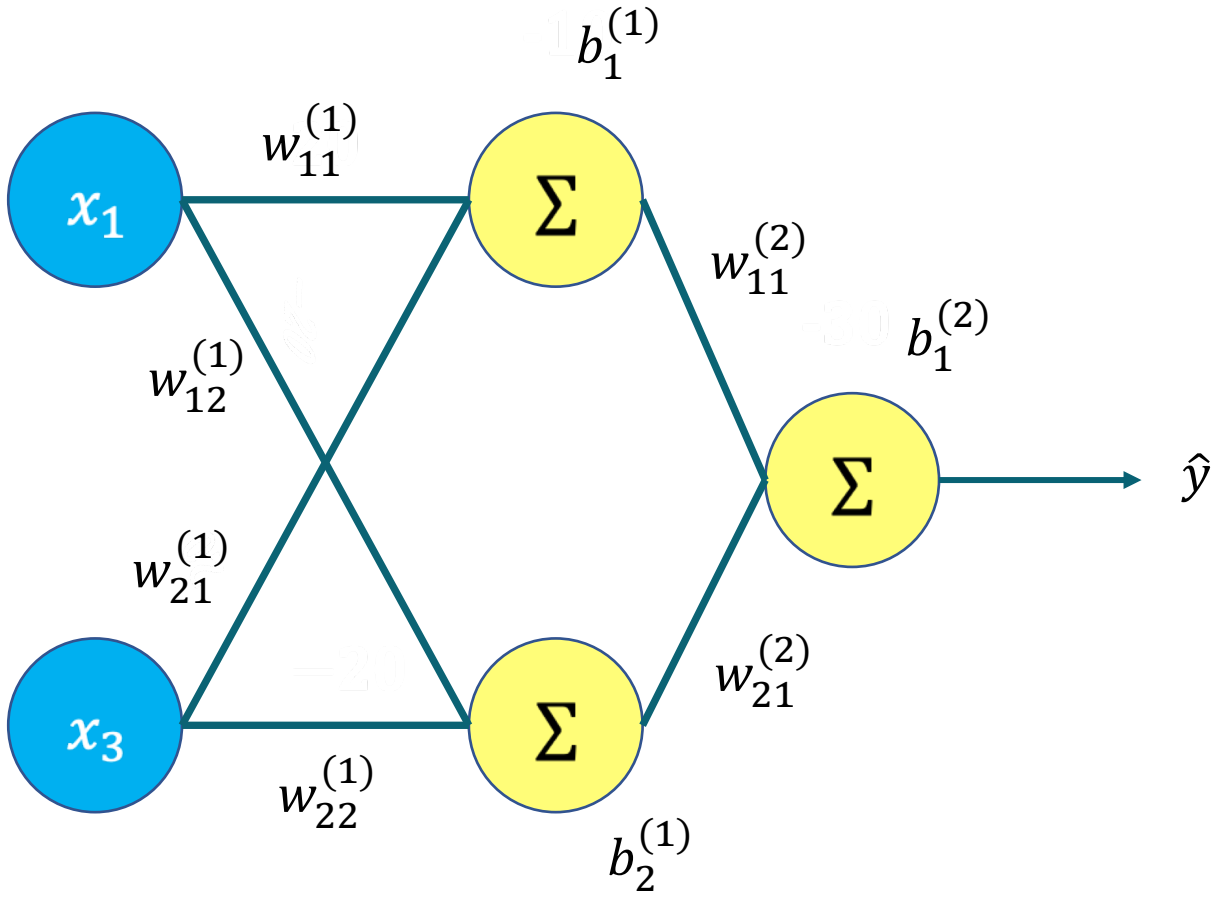


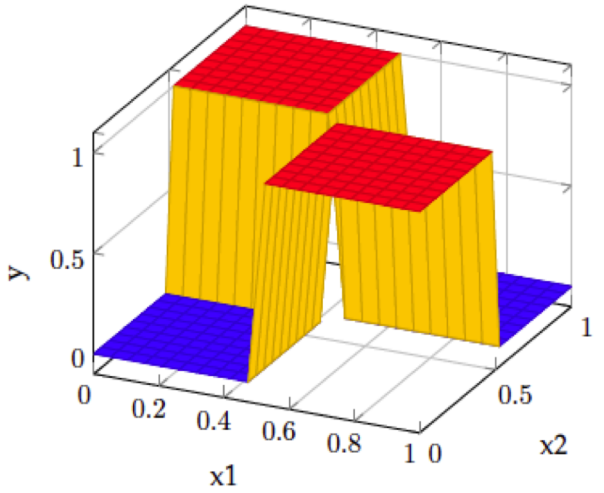
$w_1, w_2, b?$

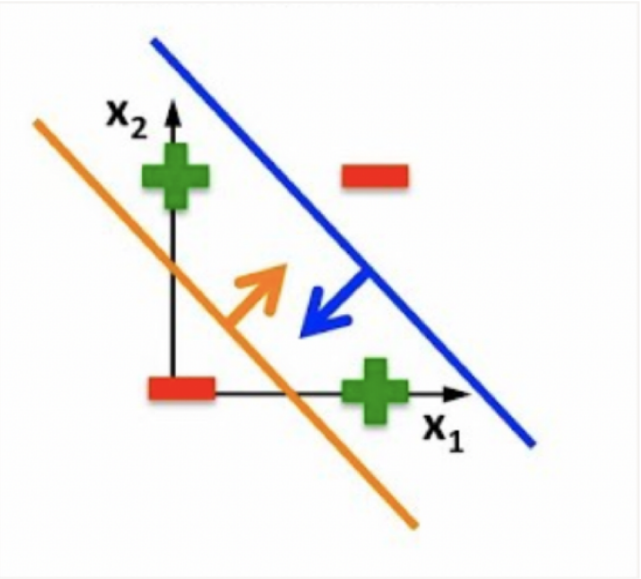
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



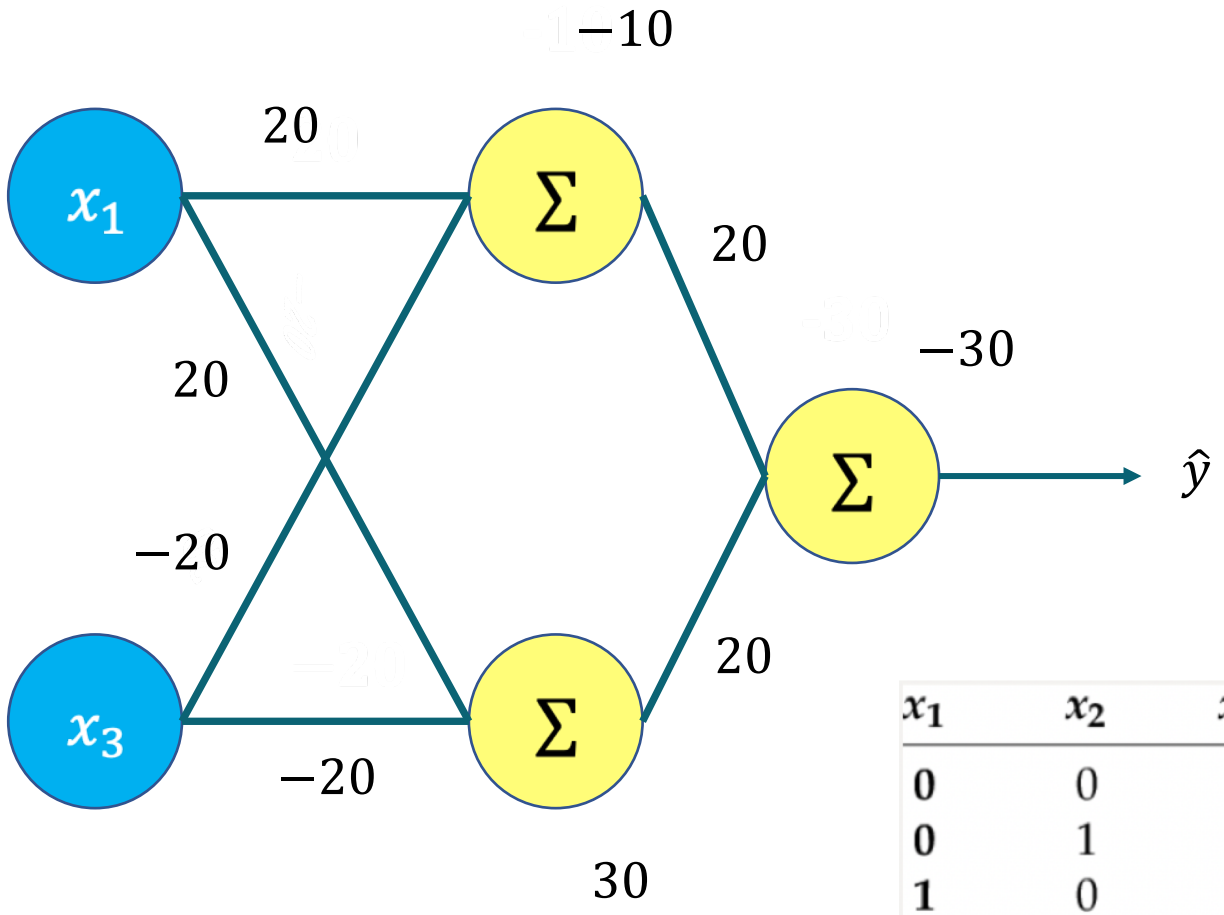
The perceptron and the XOR problem

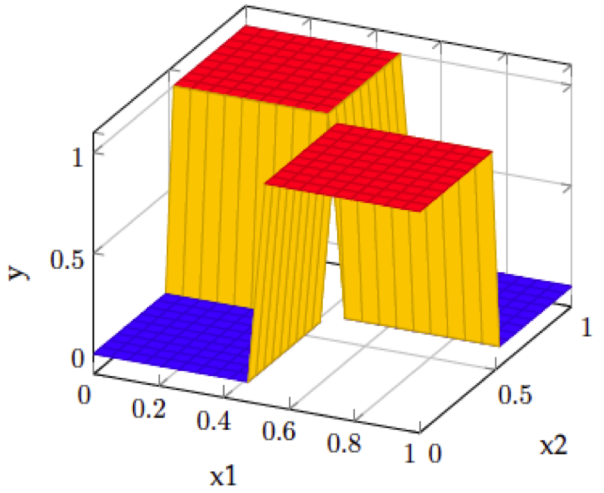




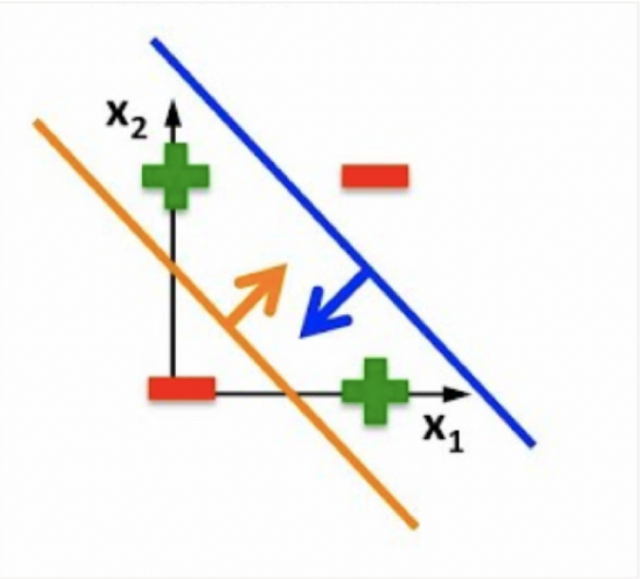


The perceptron and the XOR problem

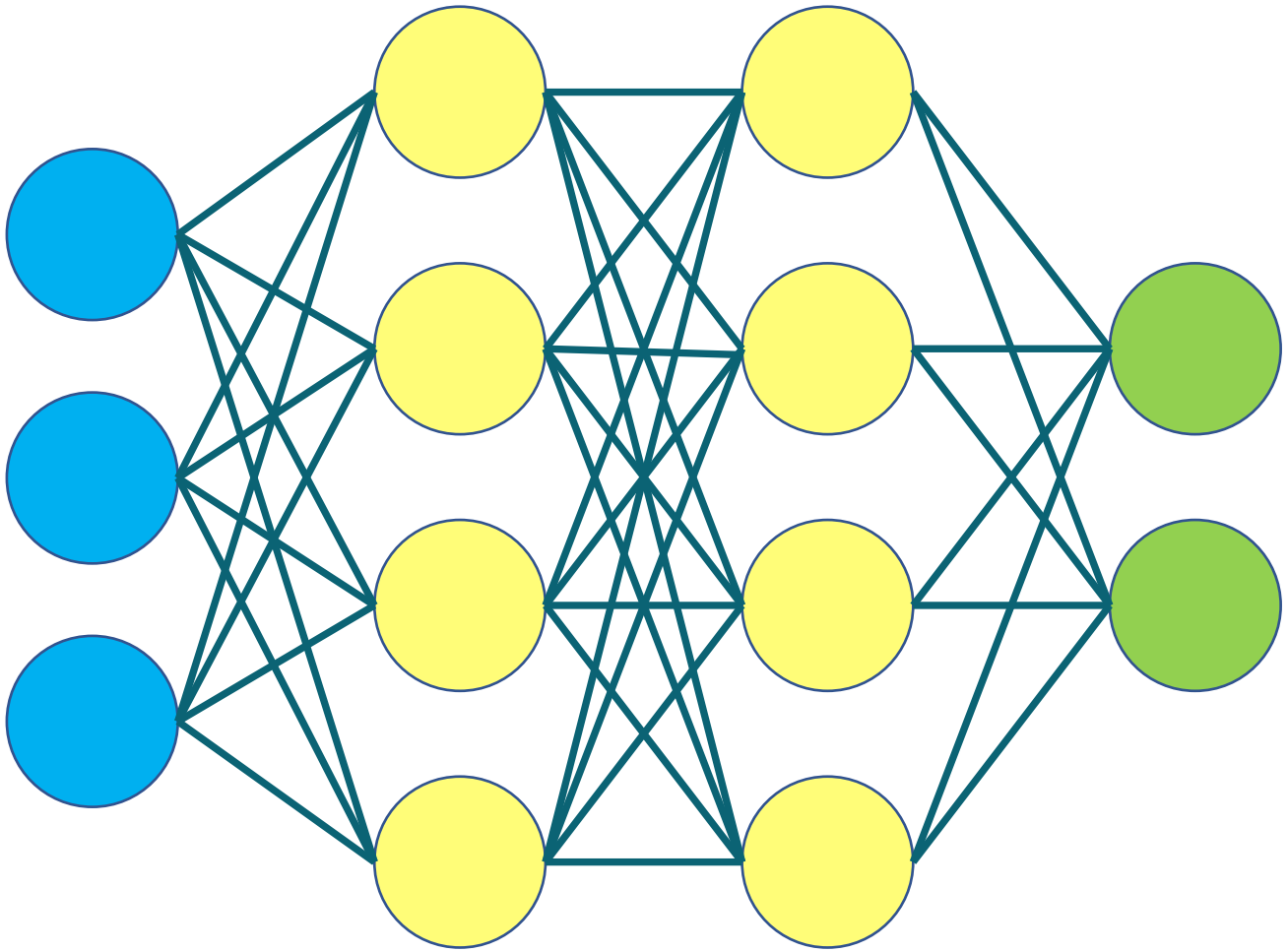




x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



Artificial Neural Network

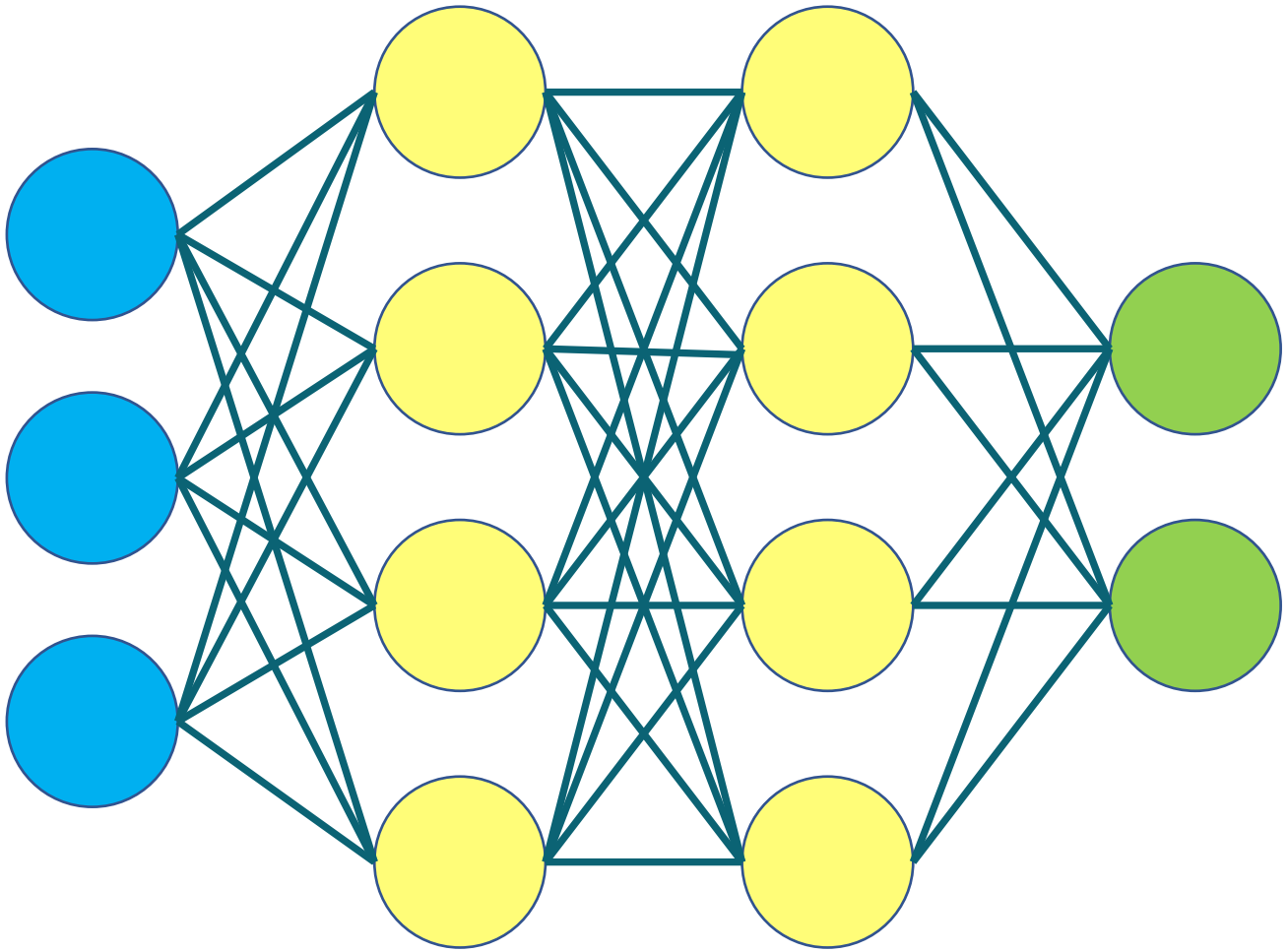


Artificial Neural Network

Humidity: 60%

Temperature: 17°C

Pressure: 102HPa



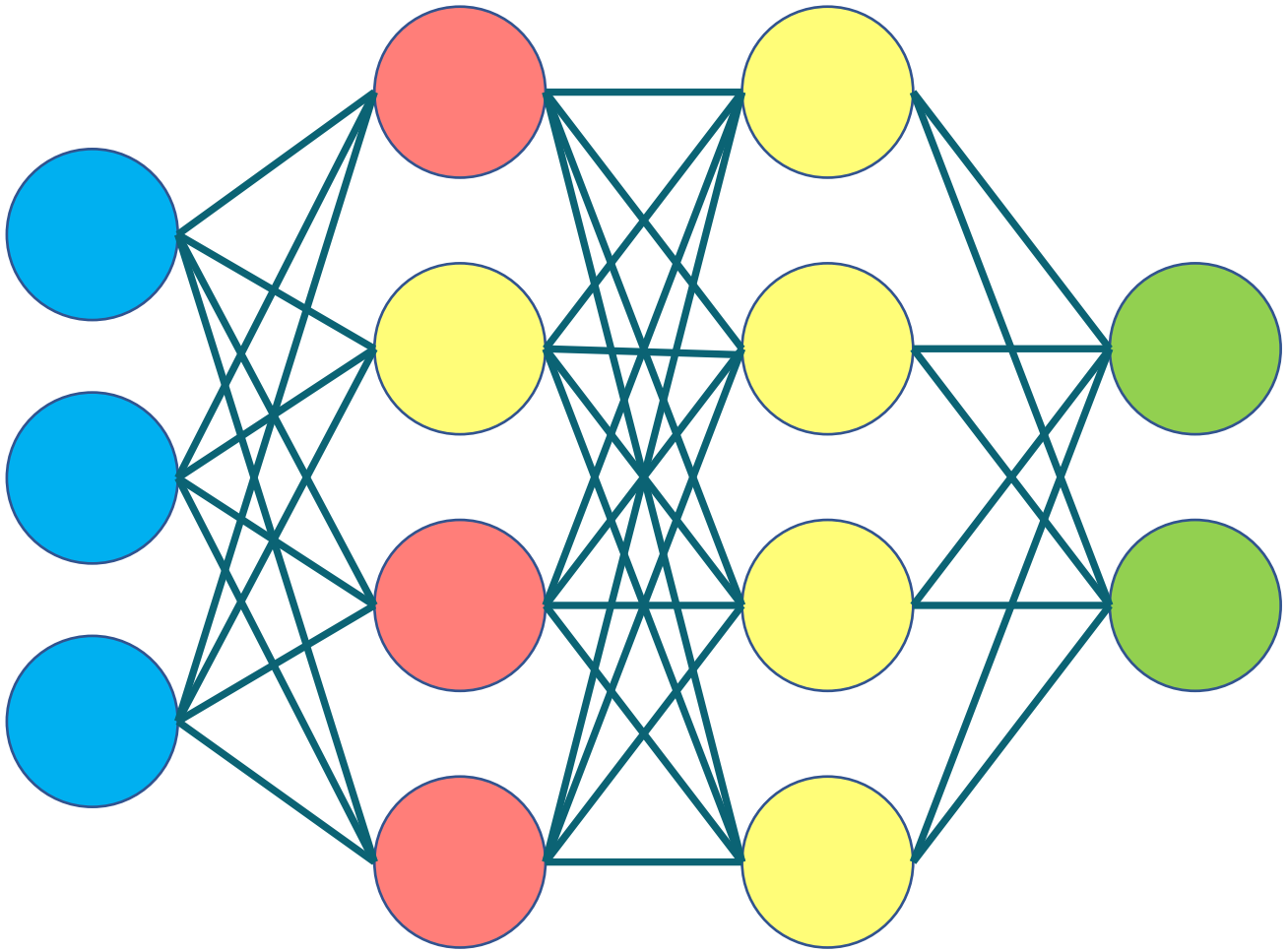
input hidden layers output

Artificial Neural Network

Humidity: 60%

Temperature: 17°C

Pressure: 102HPa



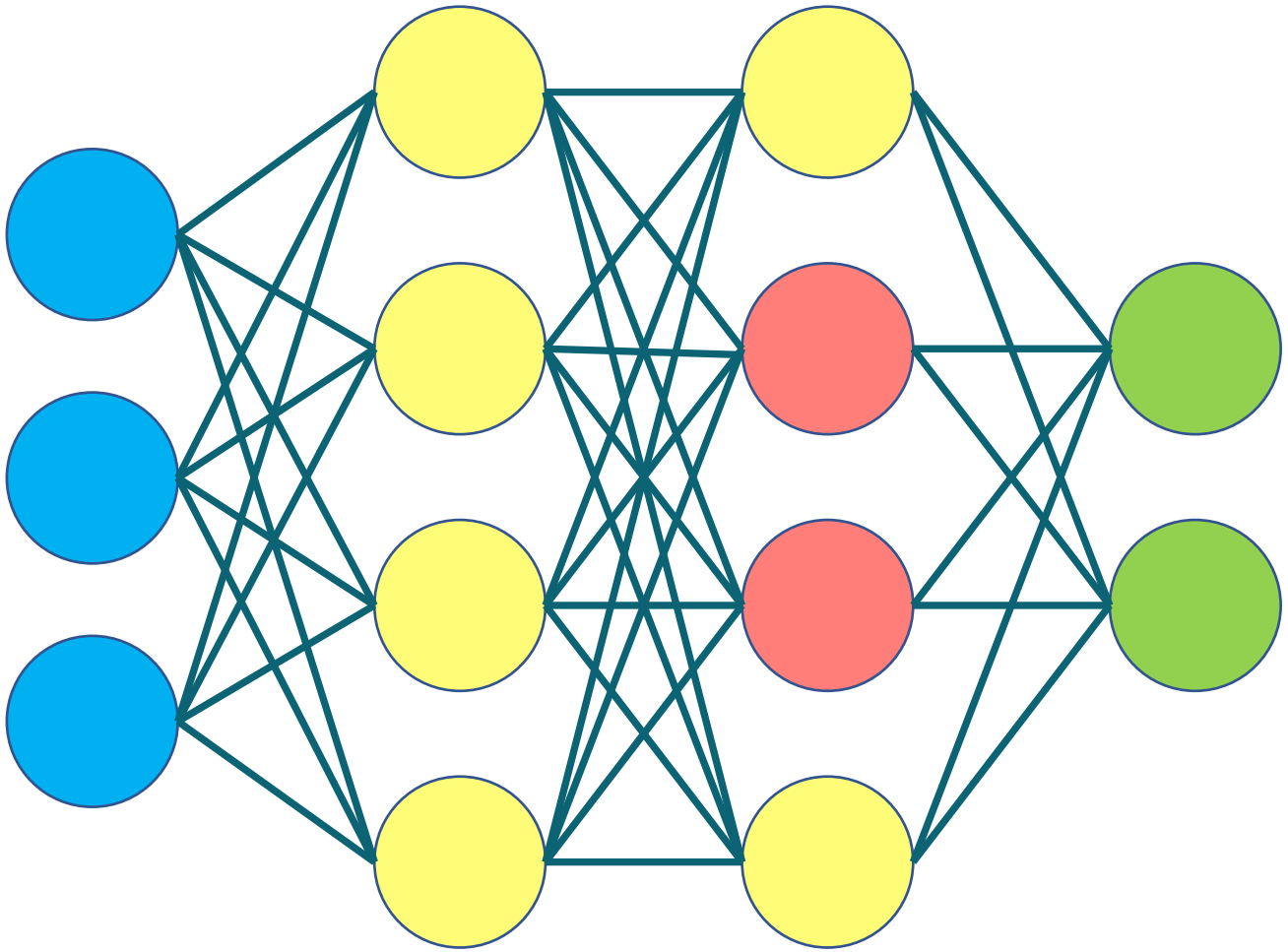
input hidden layers output

Artificial Neural Network

Humidity: 60%

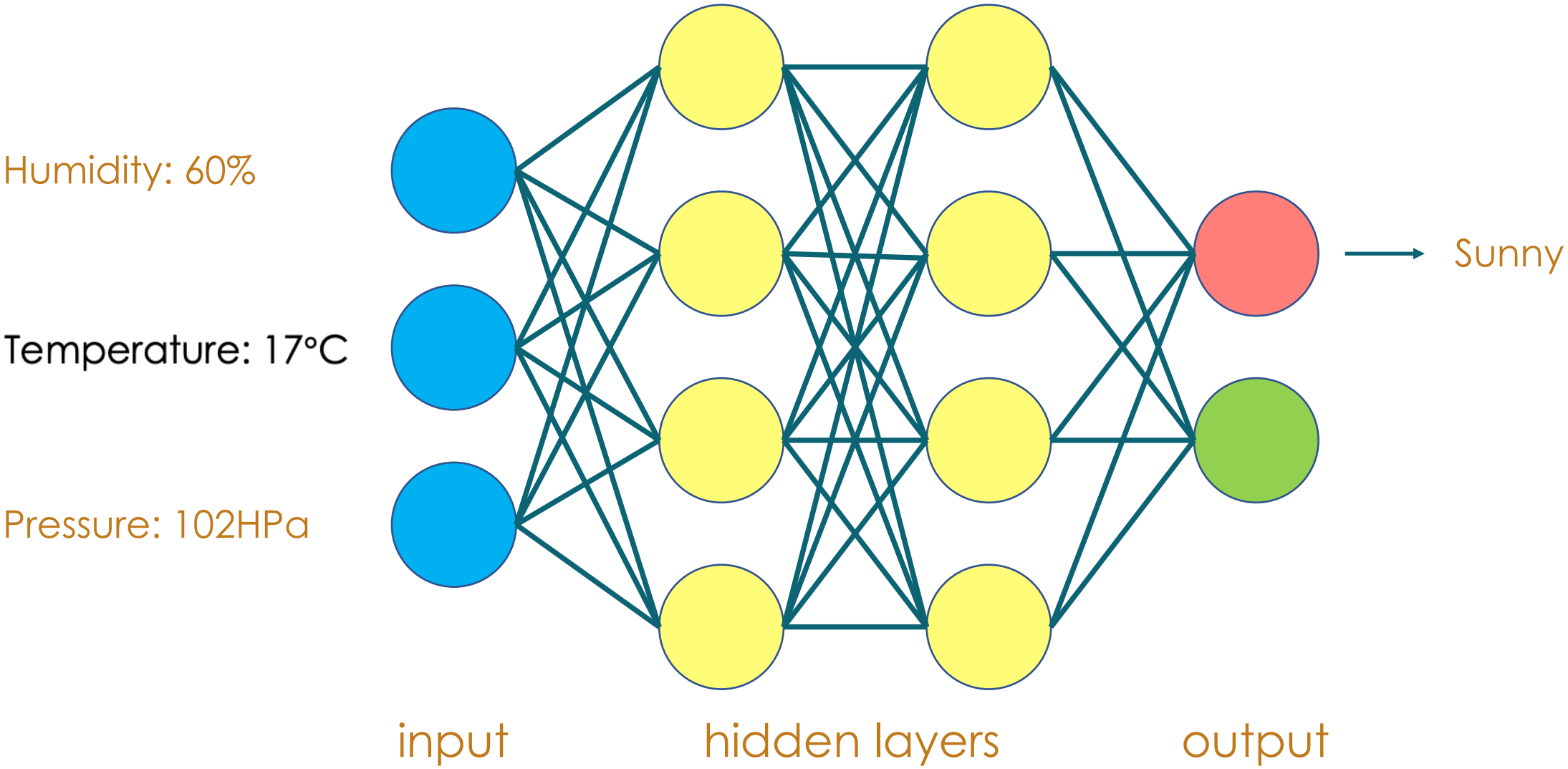
Temperature: 17°C

Pressure: 102HPa



input hidden layers output

Artificial Neural Network

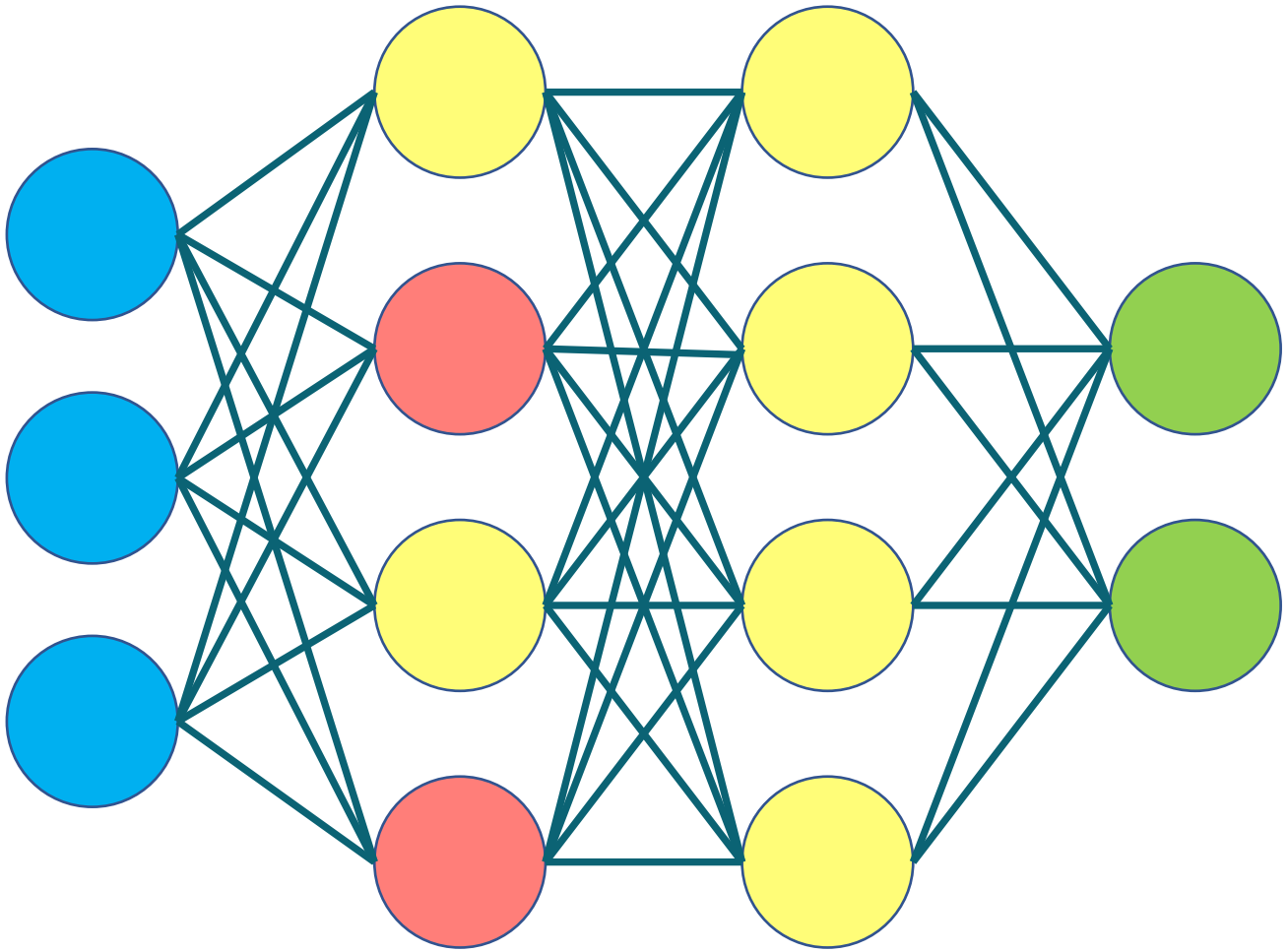


Artificial Neural Network

Humidity: 90%

Temperature: 16°C

Pressure: 100HPa



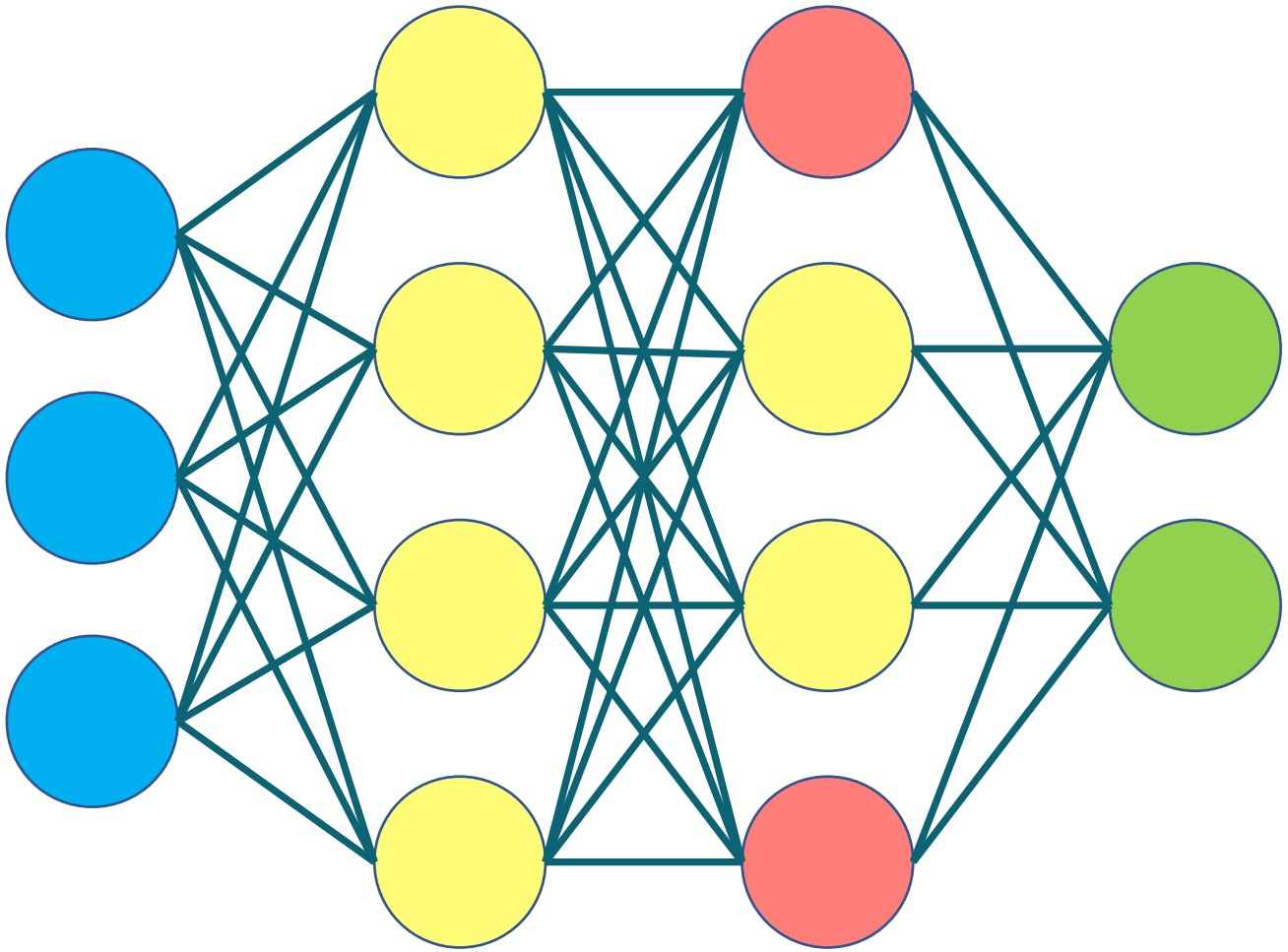
input hidden layers output

Artificial Neural Network

Humidity: 90%

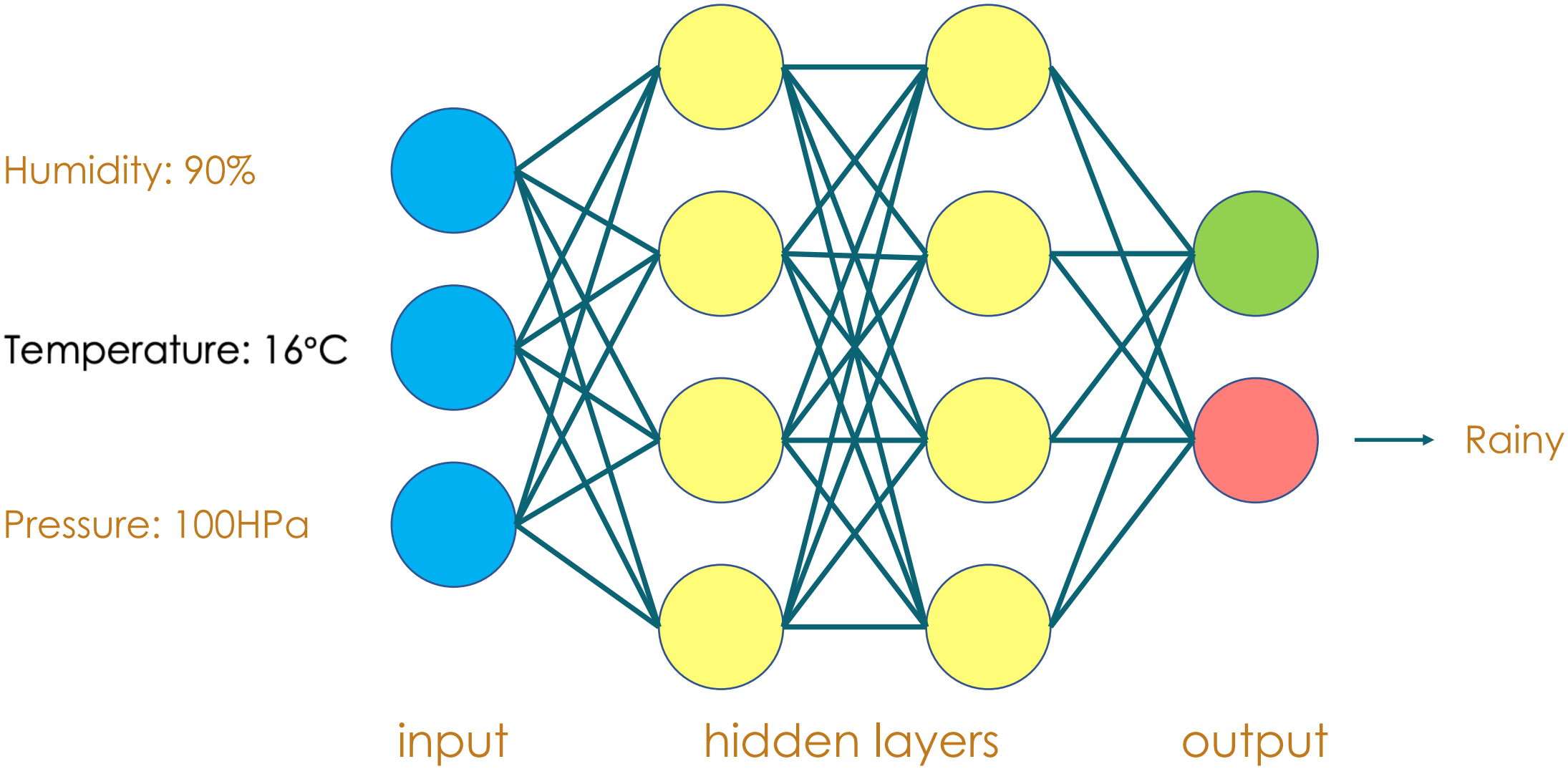
Temperature: 16°C

Pressure: 100HPa



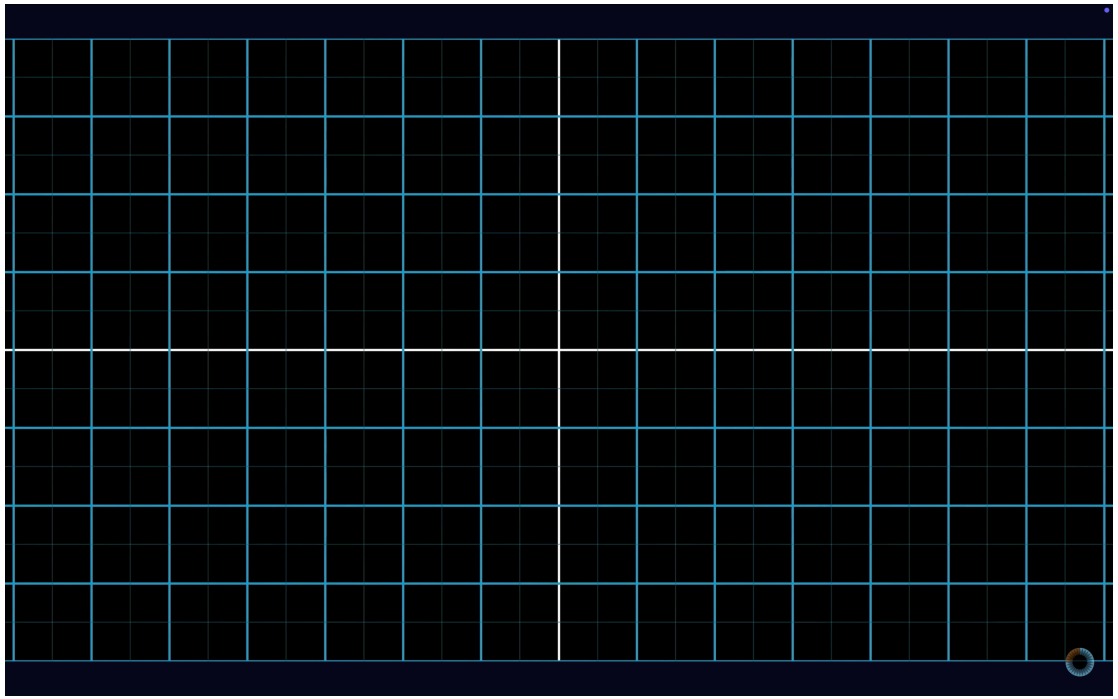
input hidden layers output

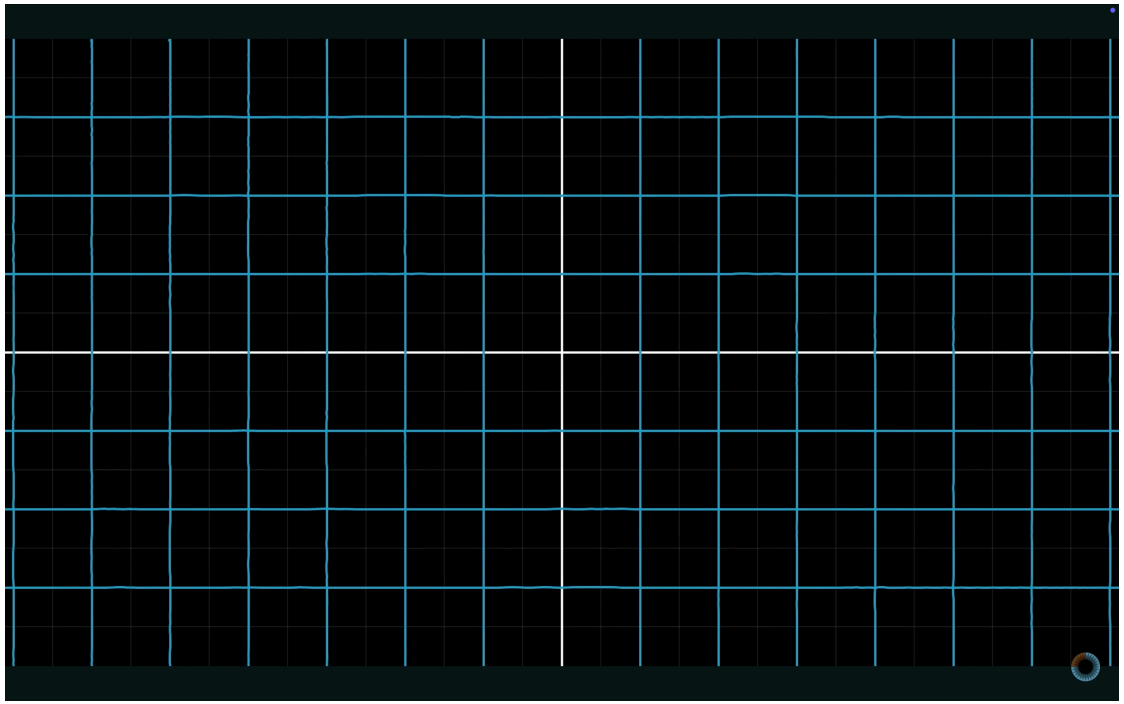
Artificial Neural Network



Matrices as linear transformations

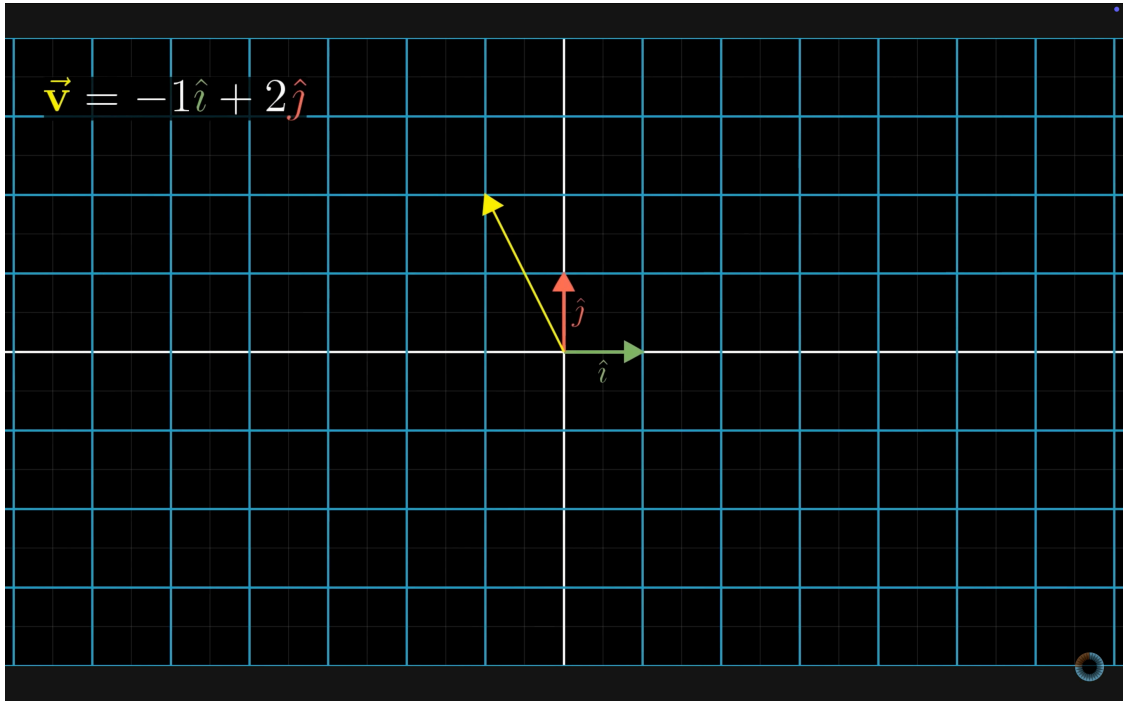
- Center remains at origin
- Lines remain parallel

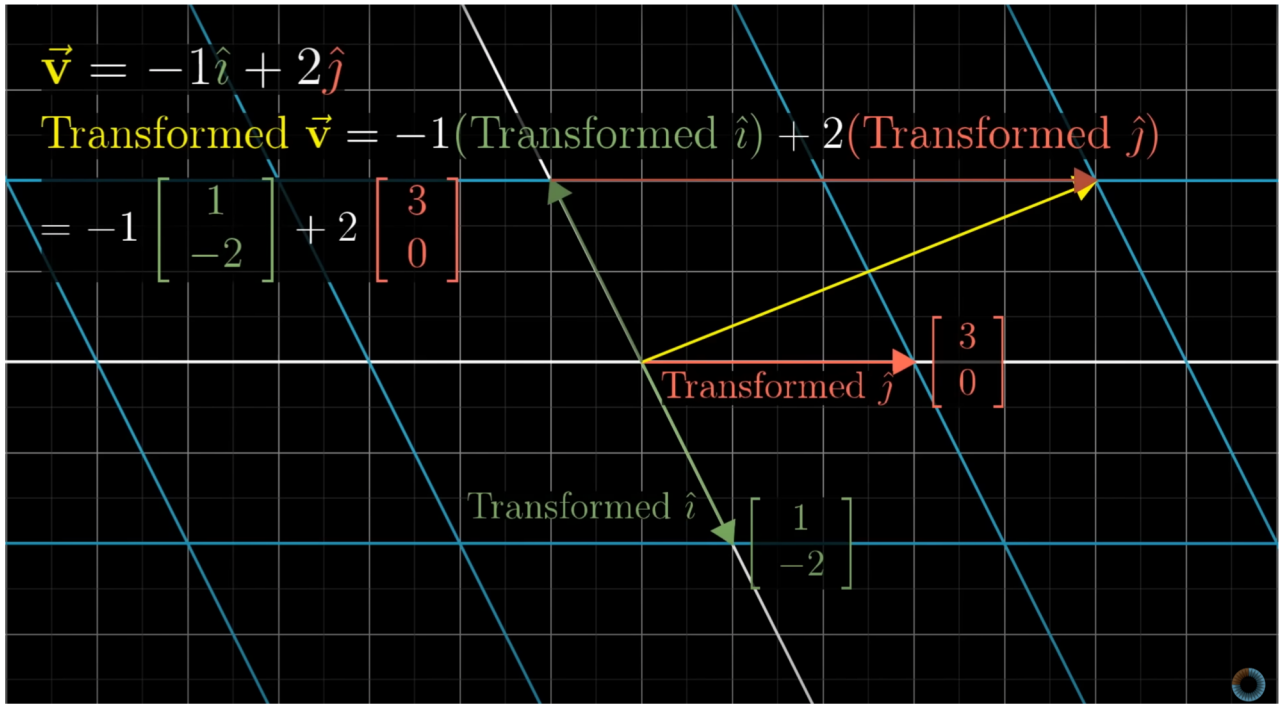




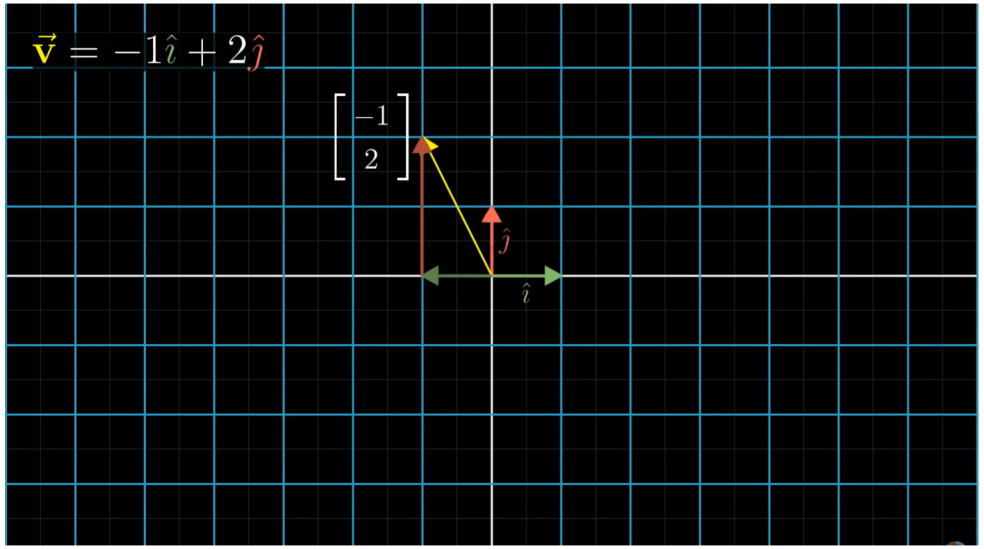
https://www.youtube.com/@3blue1brown

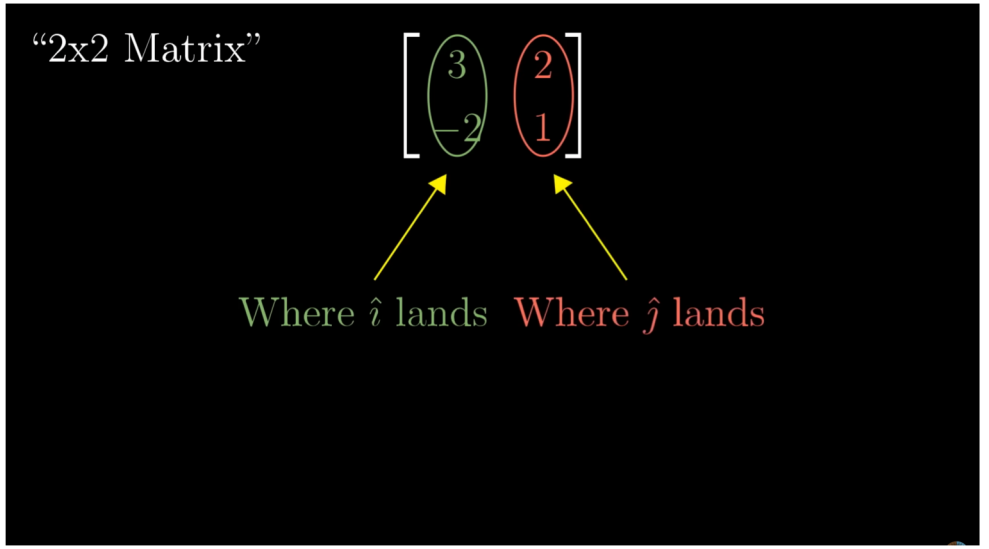
Matrices as linear transformations

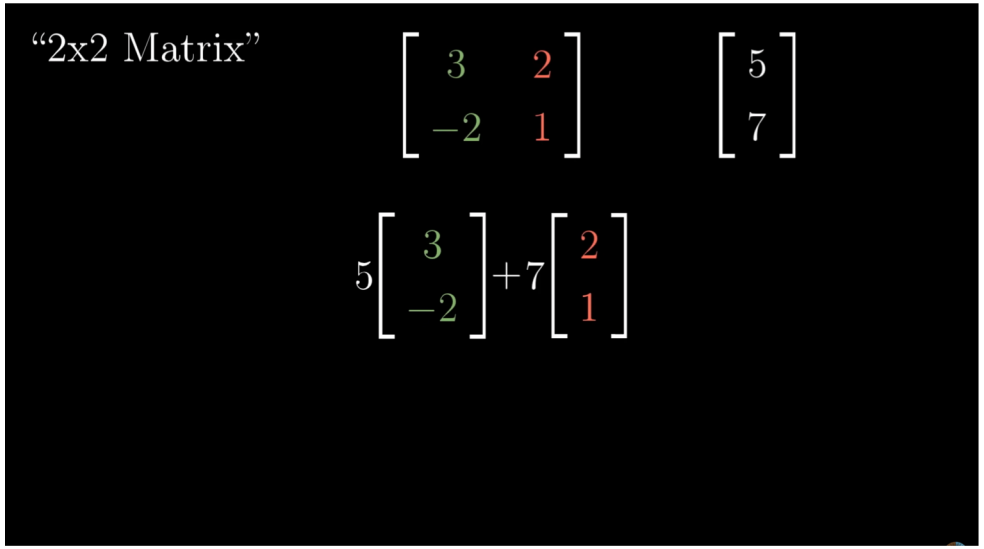


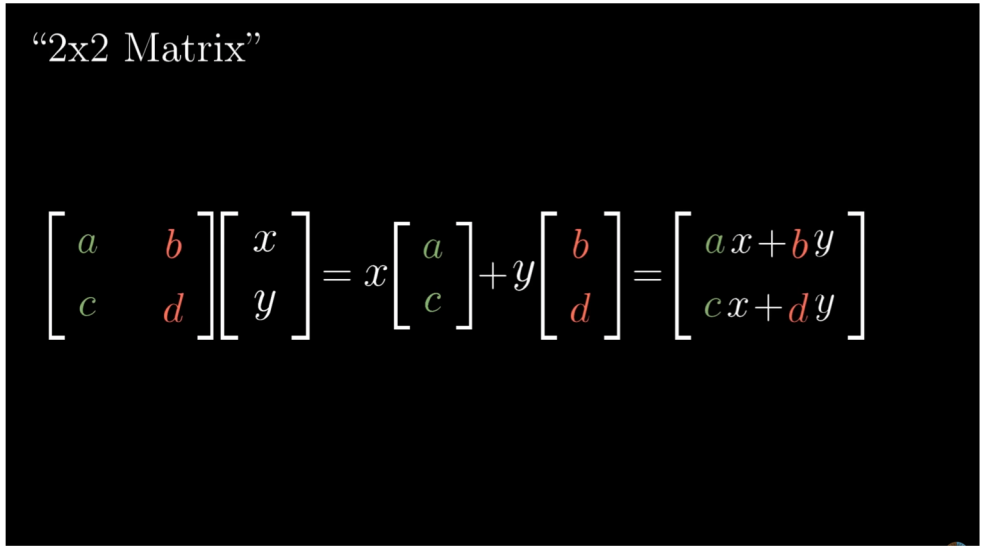


Matrices as linear transformations



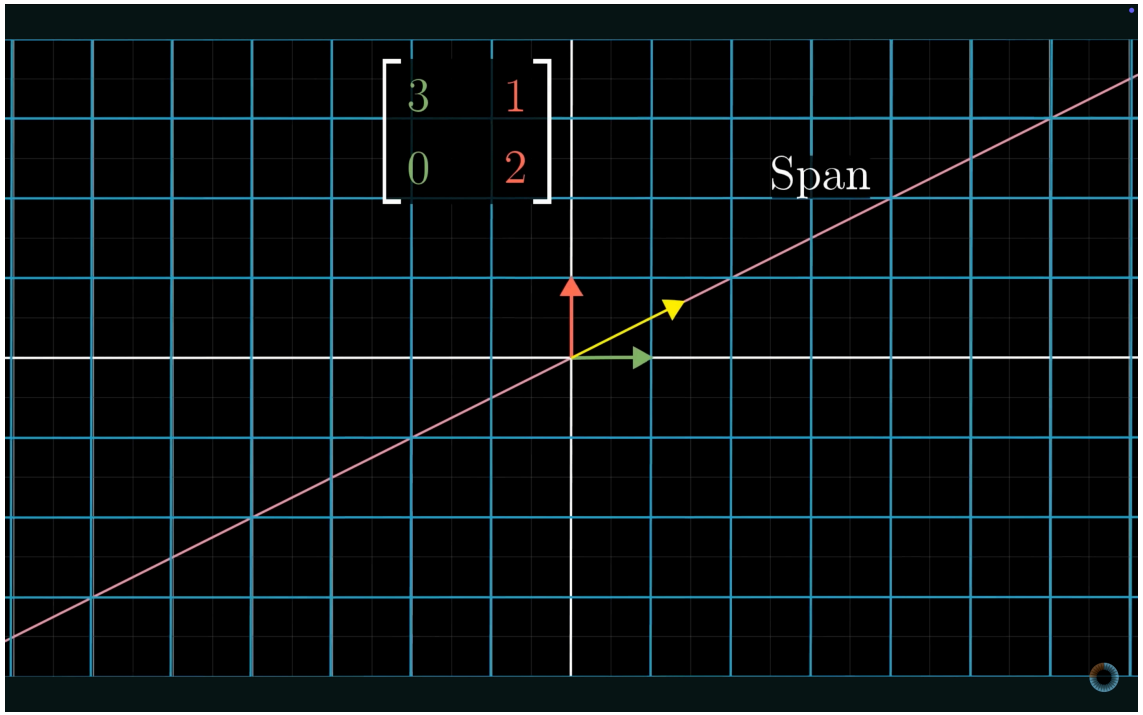




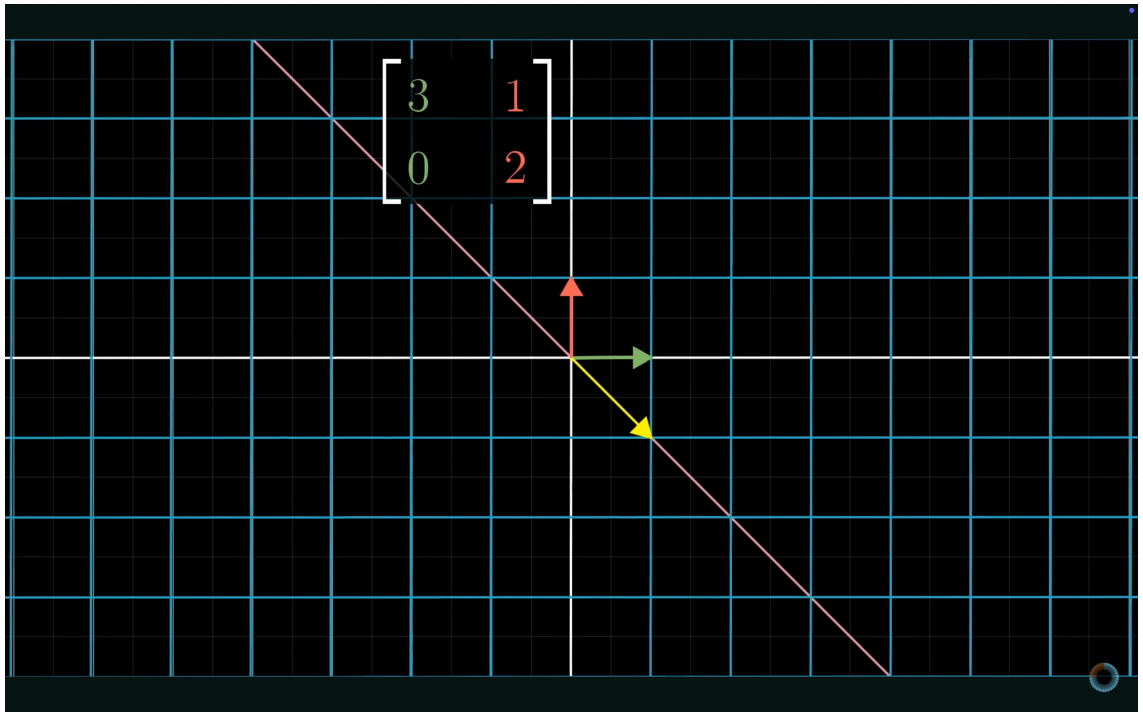


Matrices as linear transformations

- Eigenvectors are only scaled by the eigenvalues

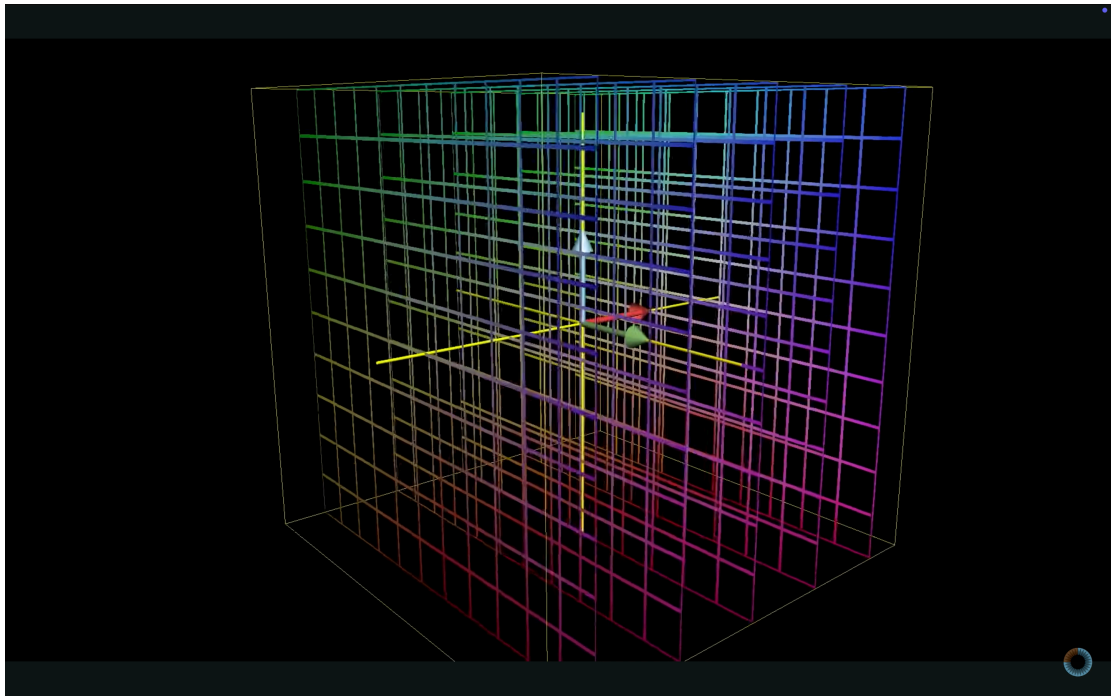


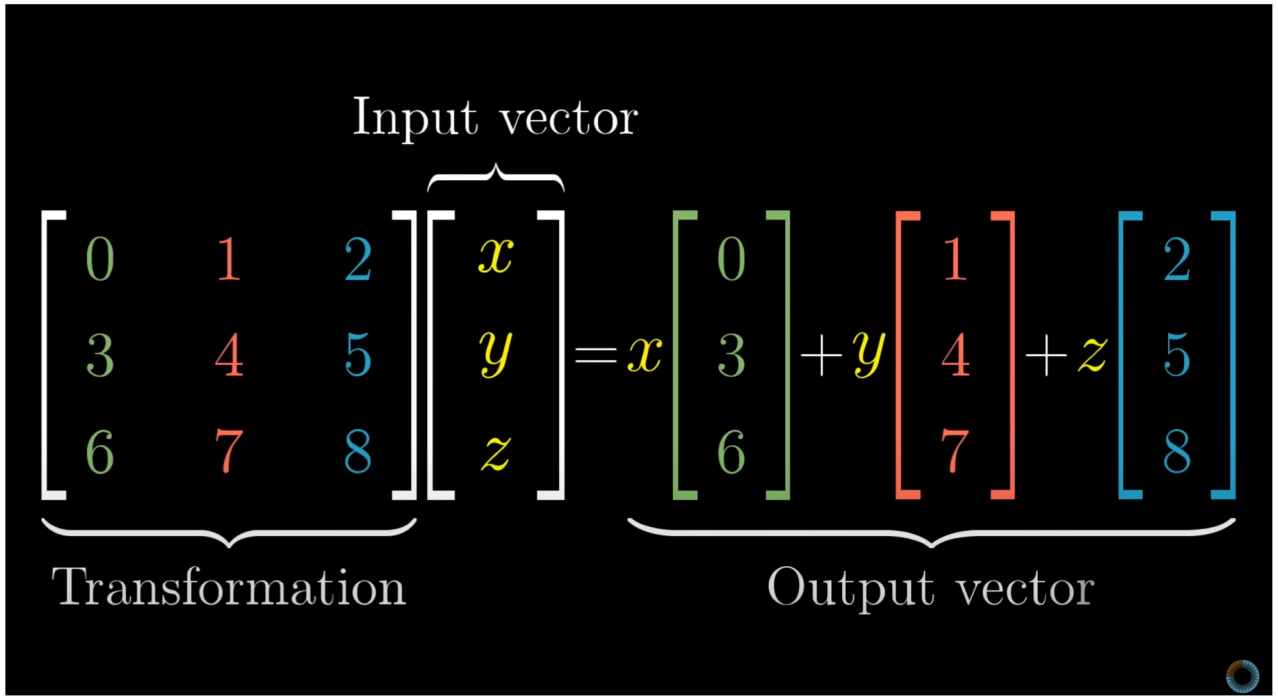
No eigenvector



Eigenvector

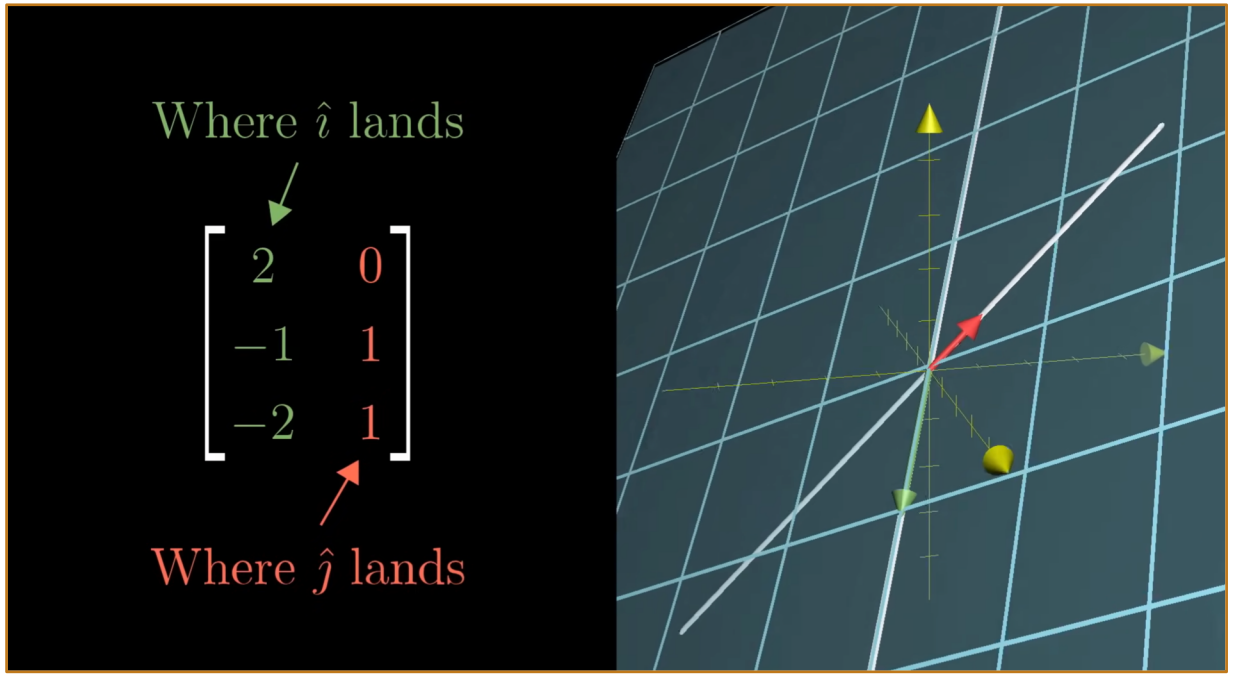
Linear transformations in 3D

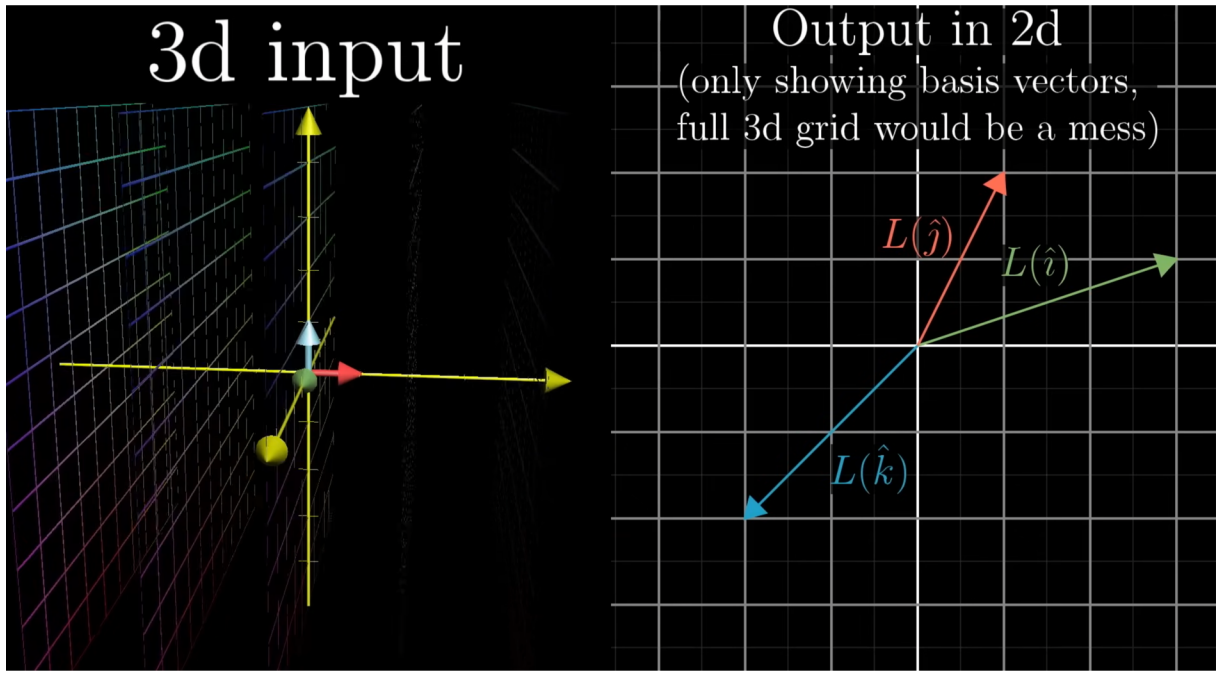




Non-square matrices as transformations between dimensions

- What transformation does a 3x2 matrix represent? And a 3x2?

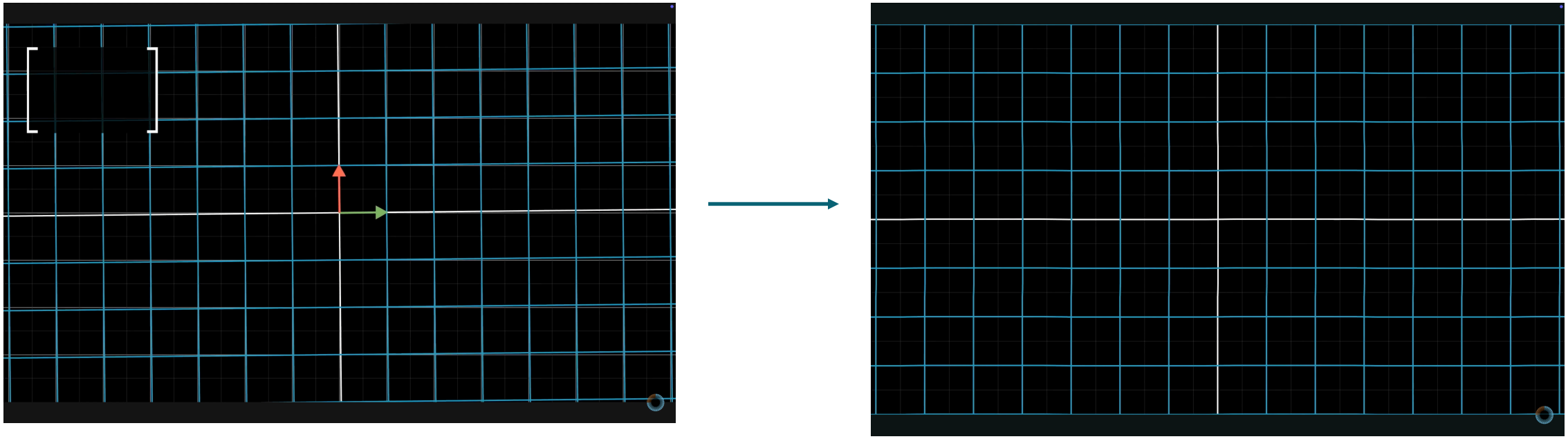




Matrix multiplication is linear → we can repeatedly apply transformations to a vector between any number of dimensions

Neural Networks

- Each layer of (trainable) weights applies a linear transformation to the input vector, followed by a non-linear activation. The output of a given layer becomes the input to the following layer

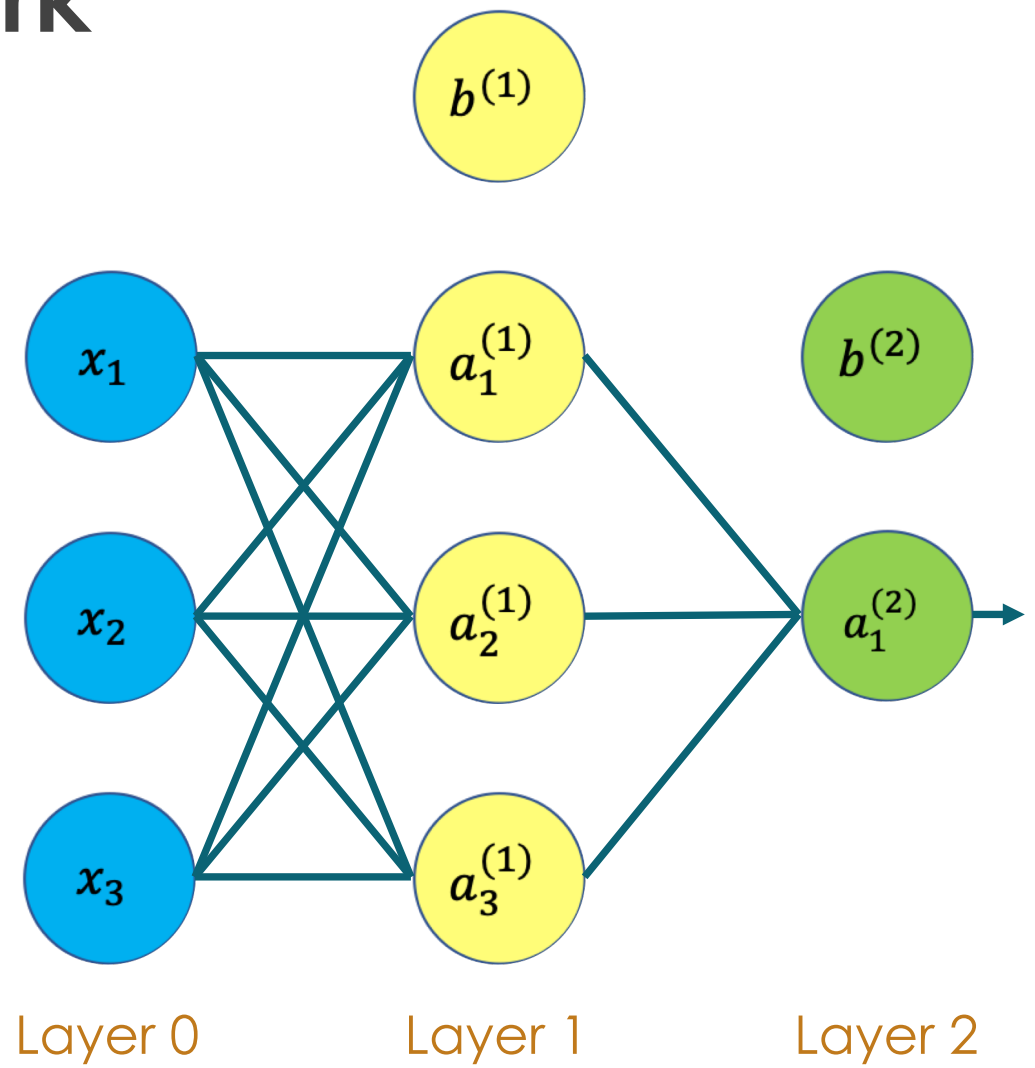


Building a neural network

- **Input layer (layer 0):** this layer consists of the input features x_1, x_2 and x_3
- **Hidden layer (layer 1):** contains values that we don't observe in the training set
- **Output layer (layer 2):** it has a neuron/units (can have more units) that outputs the final value computed by the hypothesis

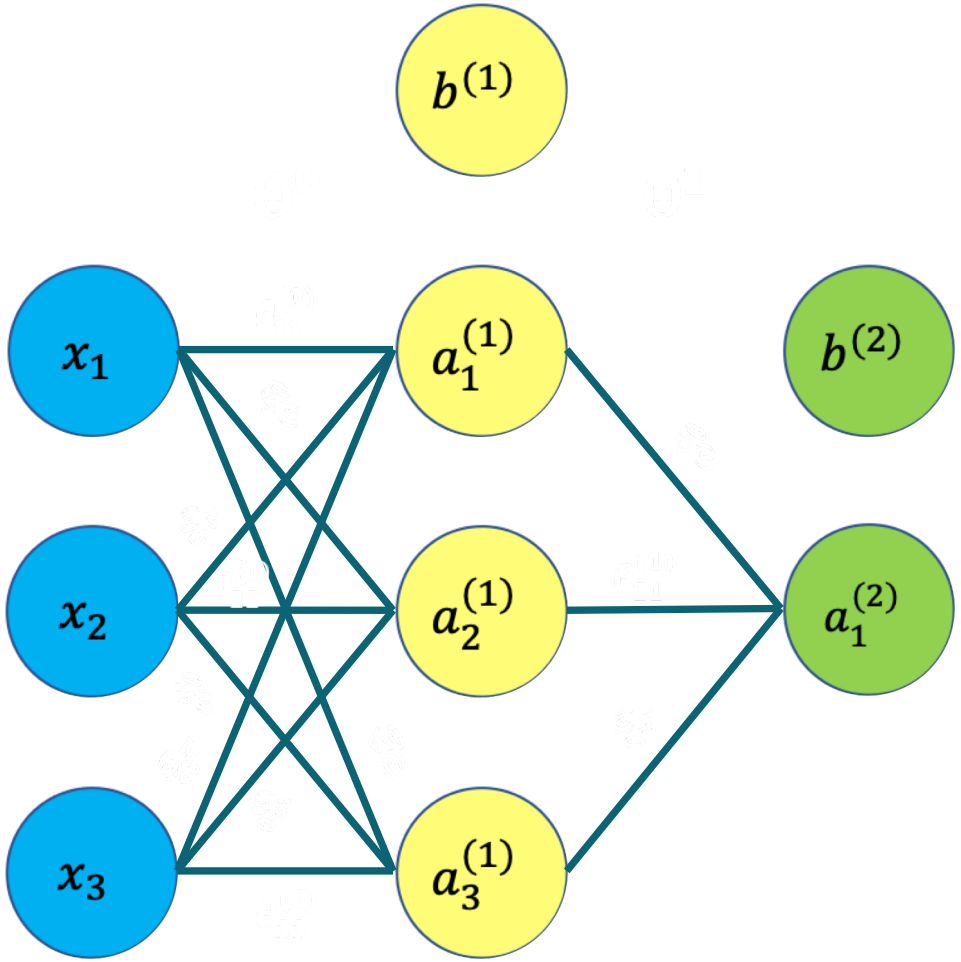
$f: \mathbb{R}^M \rightarrow \mathbb{R}^N$

$f: \mathbb{R}^3 \rightarrow \mathbb{R}^1$



Building a neural network

$$f: \mathbb{R}^3 \rightarrow \mathbb{R}^1$$



$$a_1^{(1)} = g(\theta_{11}^{(0)} x_1 + \theta_{21}^{(0)} x_2 + \theta_{31}^{(0)} x_3 + b_1^{(1)})$$

$$a_2^{(1)} = g(\theta_{12}^{(0)} x_1 + \theta_{22}^{(0)} x_2 + \theta_{32}^{(0)} x_3 + b_2^{(1)})$$

$$a_3^{(1)} = g(\theta_{13}^{(0)} x_1 + \theta_{23}^{(0)} x_2 + \theta_{33}^{(0)} x_3 + b_3^{(1)})$$

$$h_{\Theta,b}(x) = a_1^{(2)} = \theta_{11}^{(1)} a_1^{(1)} + \theta_{21}^{(1)} a_2^{(1)} + \theta_{31}^{(1)} a_3^{(1)} + b_1^{(2)}$$

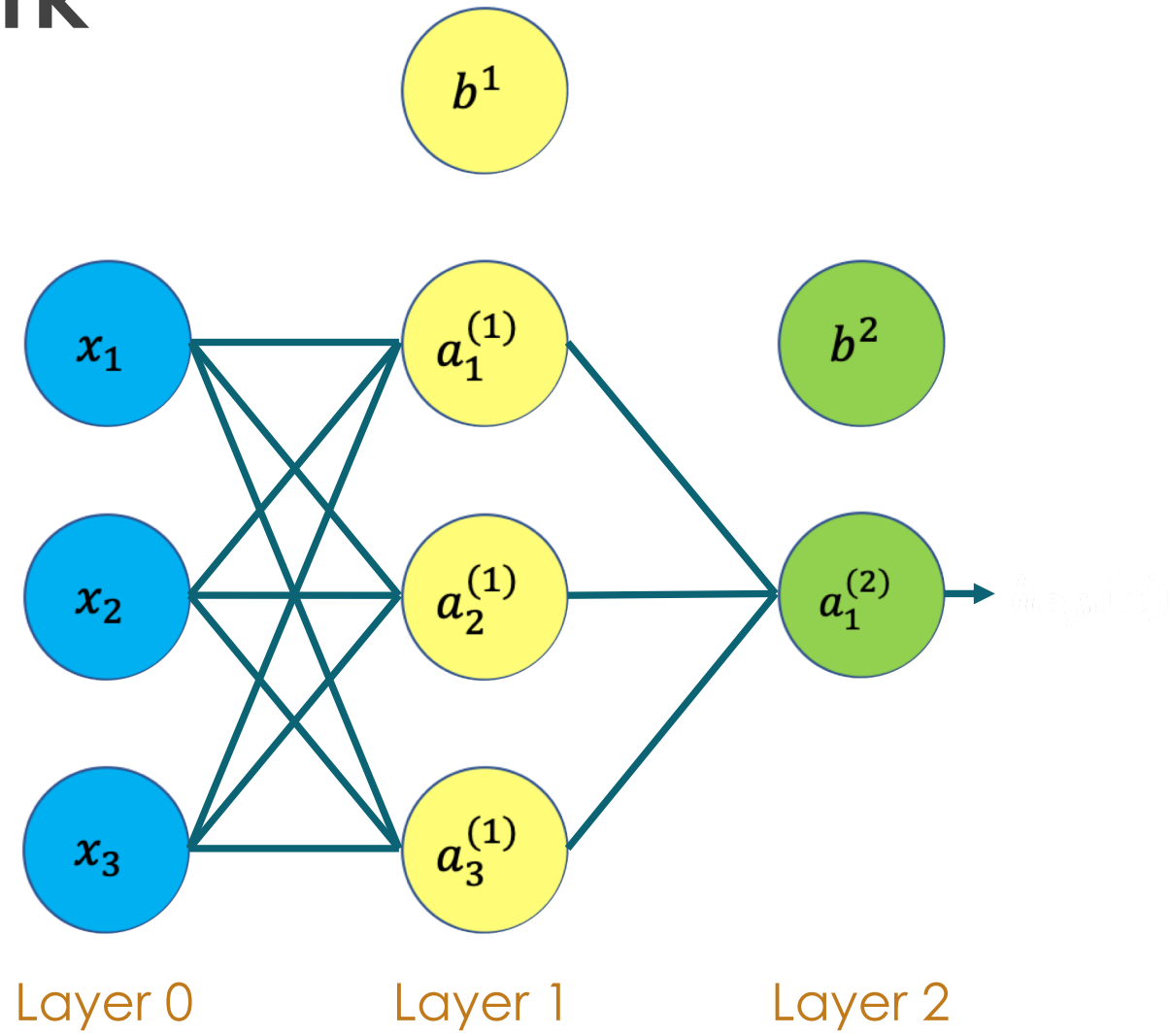
$$\Theta^{(0)} = \begin{pmatrix} \theta_{11}^{(0)} & \theta_{12}^{(0)} & \theta_{13}^{(0)} \\ \theta_{21}^{(0)} & \theta_{22}^{(0)} & \theta_{23}^{(0)} \\ \theta_{31}^{(0)} & \theta_{32}^{(0)} & \theta_{33}^{(0)} \end{pmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

$$a^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix} = (\Theta^{(0)})^T \cdot x + b^{(1)}$$

$$a^{(j)} = (\Theta^{(j-1)})^T \cdot a^{(j-1)} + b^{(j)}$$

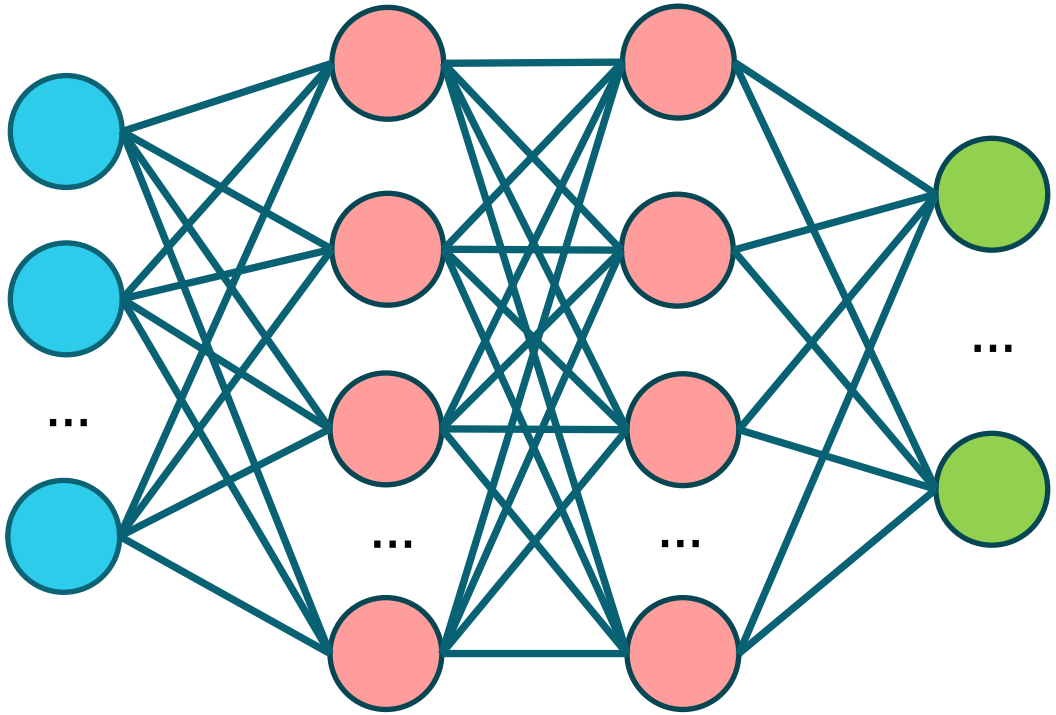
Building a neural network

- In this example the computation of $h_{\Theta,b}(x)$ is known as **forward propagation**
- It starts with the multiplication of the input units with the weights of the first layer.
- It propagates to the hidden layer computing the activation function of the second layer.
- The propagation follows with the activation of the output layer



Number of parameters

- How many parameters does the following neural network have?



- 3 layer neural network with 2 hidden layers
- Sigmoid activation funcions
- Softmax on the last layer

Universal approximation theorem

- A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, given appropriate activation functions
- Arbitrary width
- Arbitrary depth

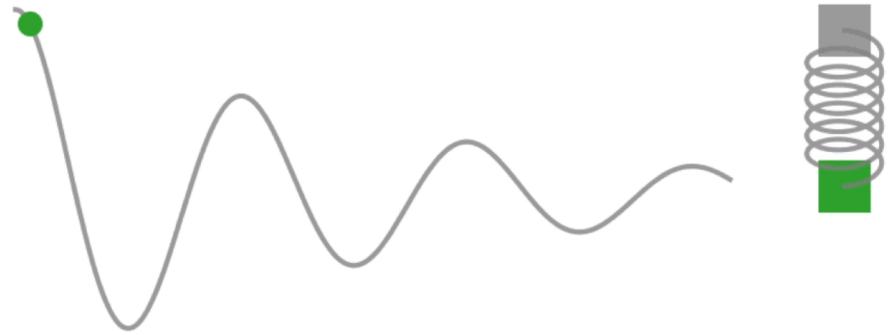
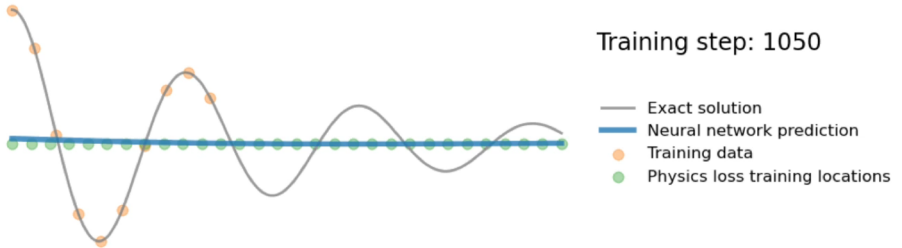


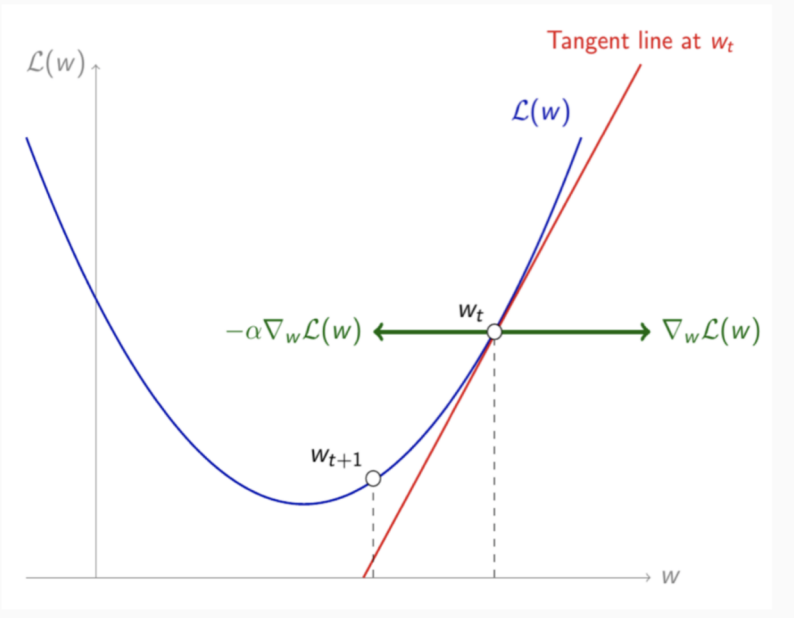
Fig 3: a 1D damped harmonic oscillator

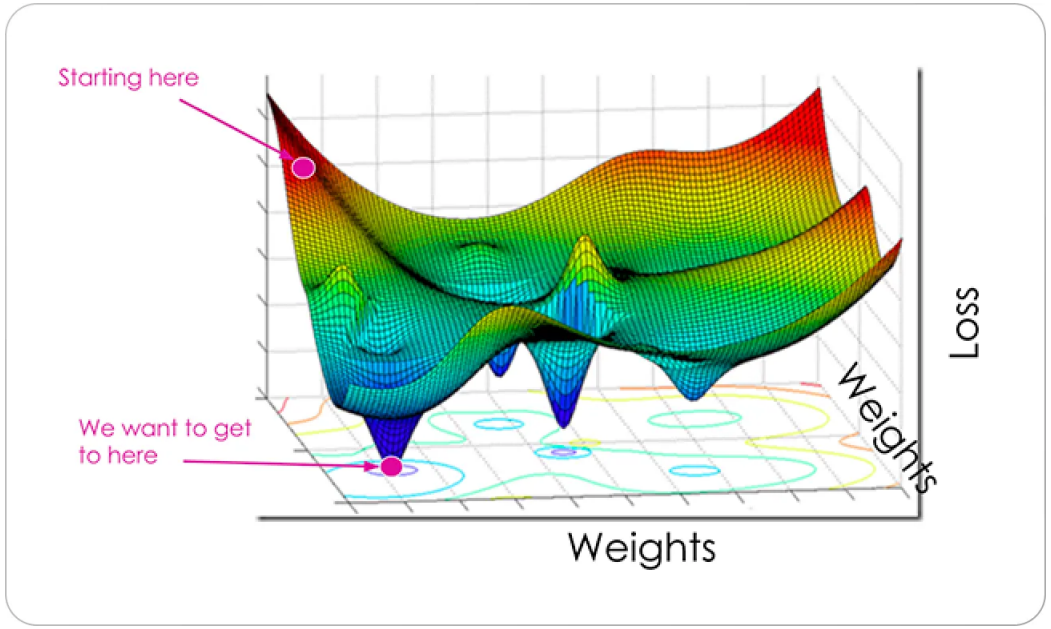


https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network/

Learning in neural networks

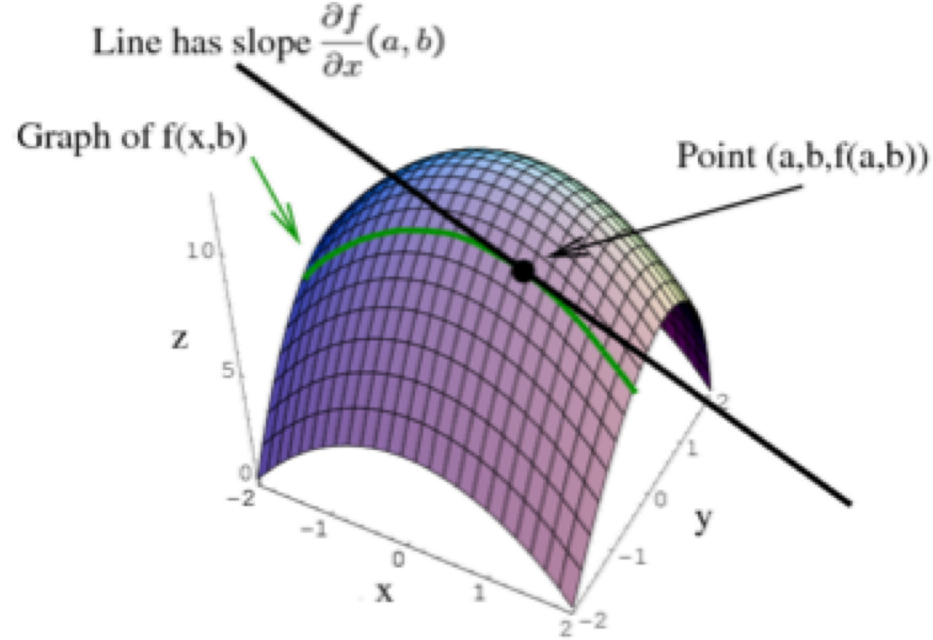
- The process of finding the combination of weights and biases that minimize the error on the training set – optimization problema
- Generalization: a well-trained model should perform roughtly equal on seen vs unseed data





Partial derivatives

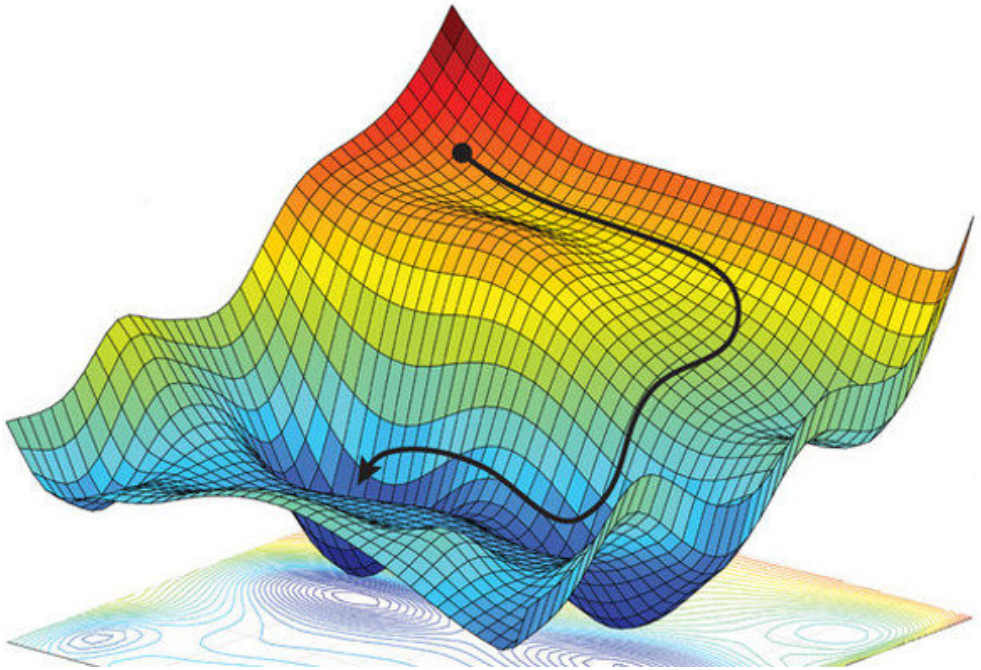
- In a multivariate function, the partial derivatives tell us how the output of our function changes with respect to each one of the variables in the input set
- When we derivate with respect to one of the variables, we treat the rest of the variables as constants



Gradient

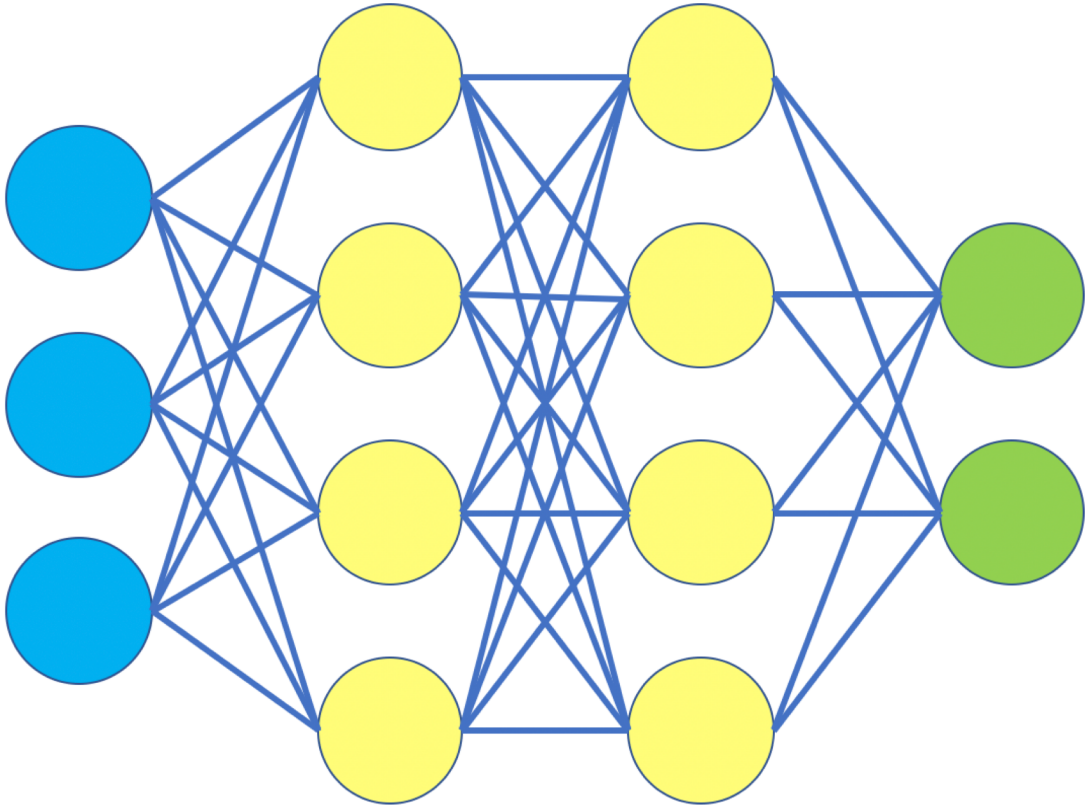
From Wikipedia:

The gradient of a scalar-valued differentiable function f of several variables is a vector field (or vector-valued function) ∇f whose value at a point p gives the direction and the rate of fastest increase. If the gradient of a function is non-zero at a point p, and the magnitude of the gradient is the rate of increase in that direction



Gradient

- During the training phase, we input a batch of samples from our tranining set and compute the gradient of the loss for that particular batch
- This will give us a vector with dimensionality equal to the number of parameters in our model, which is the direction of steepest ascent w/r to the loss function

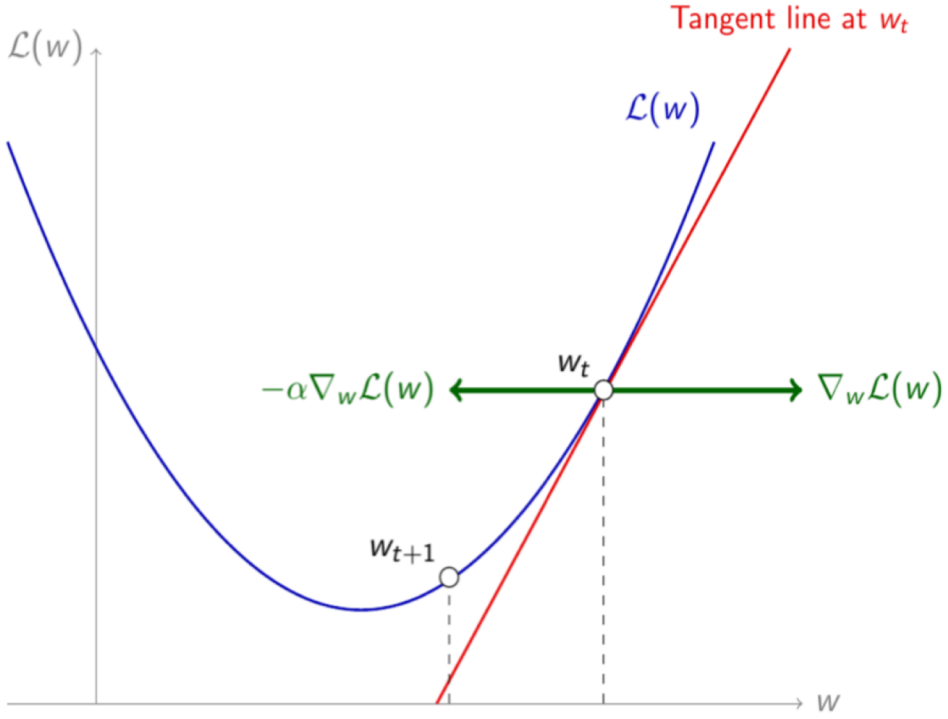


$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \qquad \begin{bmatrix} 0.2 \\ \dots \\ -0.1 \end{bmatrix}$$

Gradient Descent

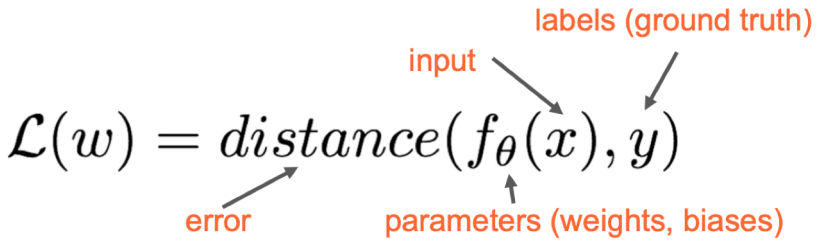
- Goal: to find the set of weights and biases that minimizes the loss function with respect to the training set
- **Gradient descent algorithm**
- The gradient tells us in which direction the loss function changes locally with respect to a change in our model's parameters

$$\theta_i^{j+1} := \theta_i^j - \alpha \cdot \frac{\partial L}{\partial \theta_i^j}$$



Loss function

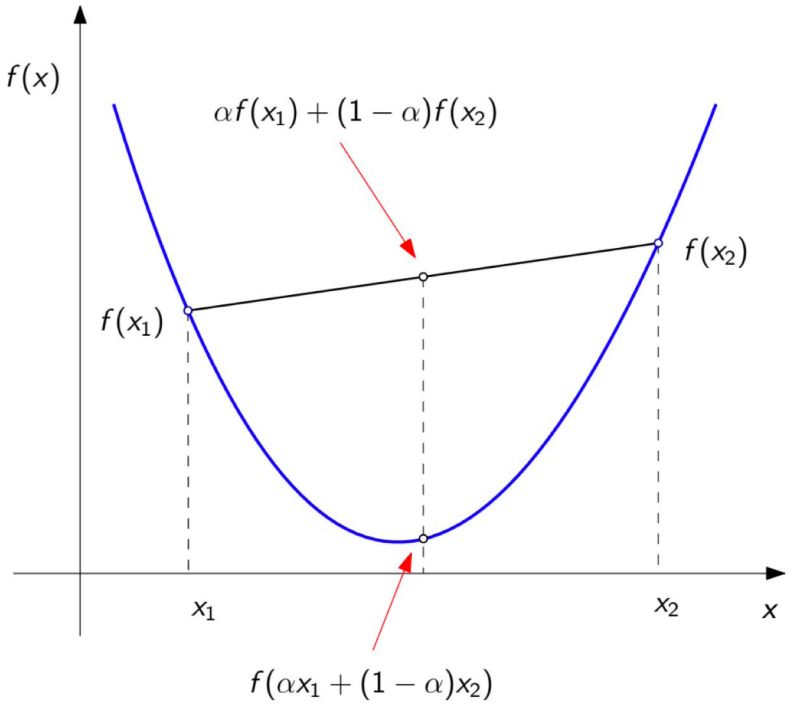
- The loss function gives us a distance (error) between our predictions and the target values



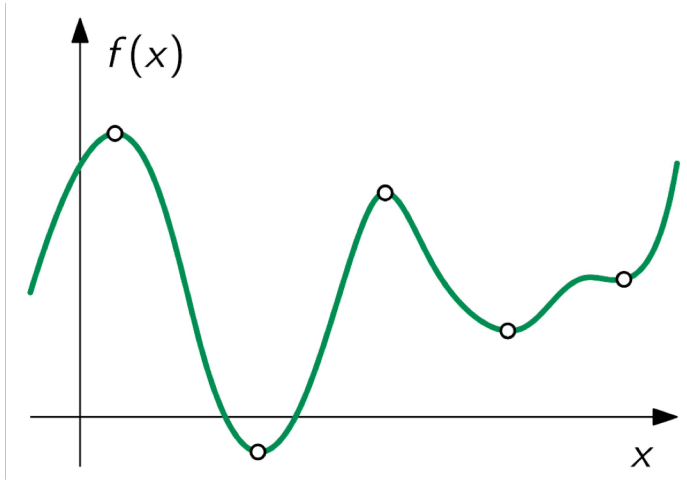
Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] =$	Quantify the difference between two probability

Loss function

- Convex functions have a single minimum
- Most loss functions of the problems that we tackle with deep learning are high-dimensional non-convex functions with many local minima



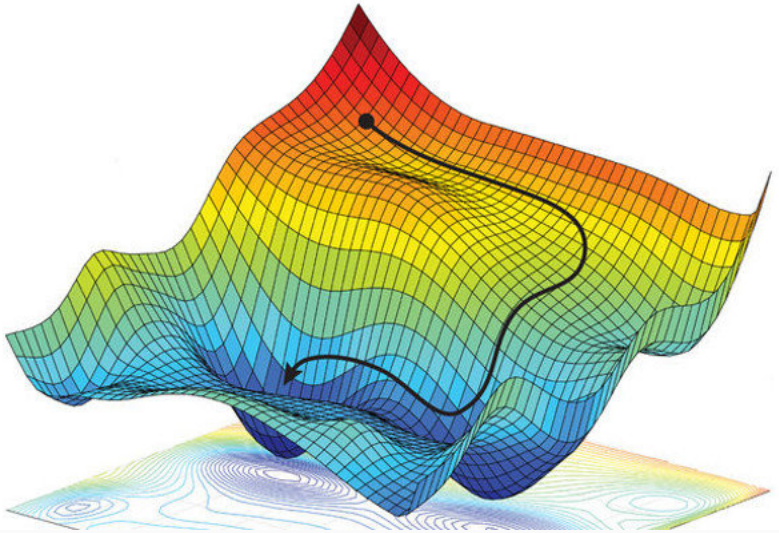
convex

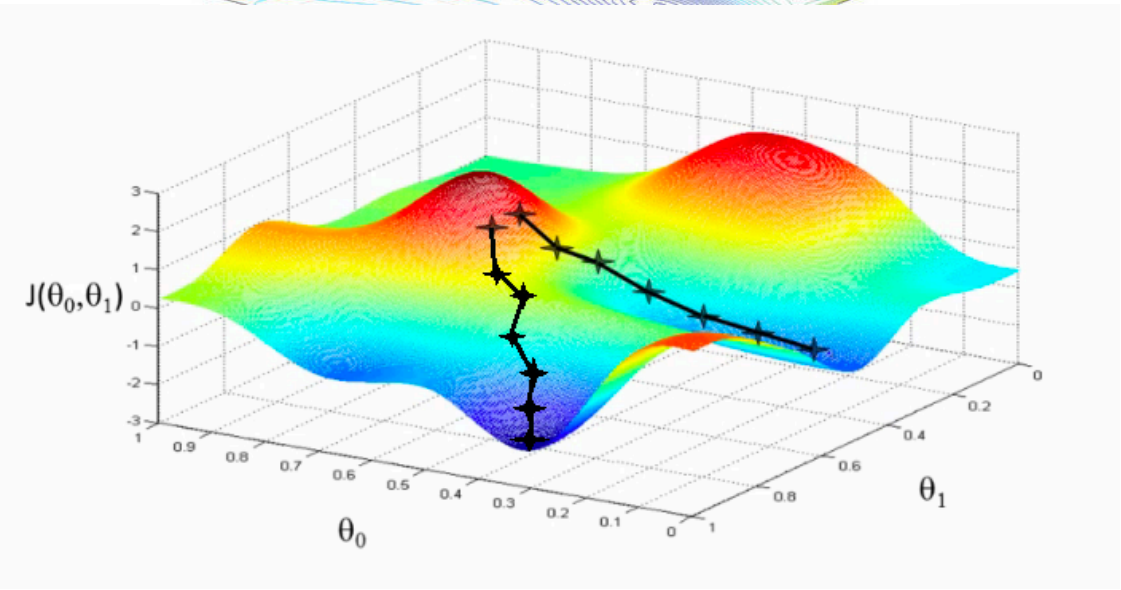


non-convex

Stochastic Gradient Descent (SGD)

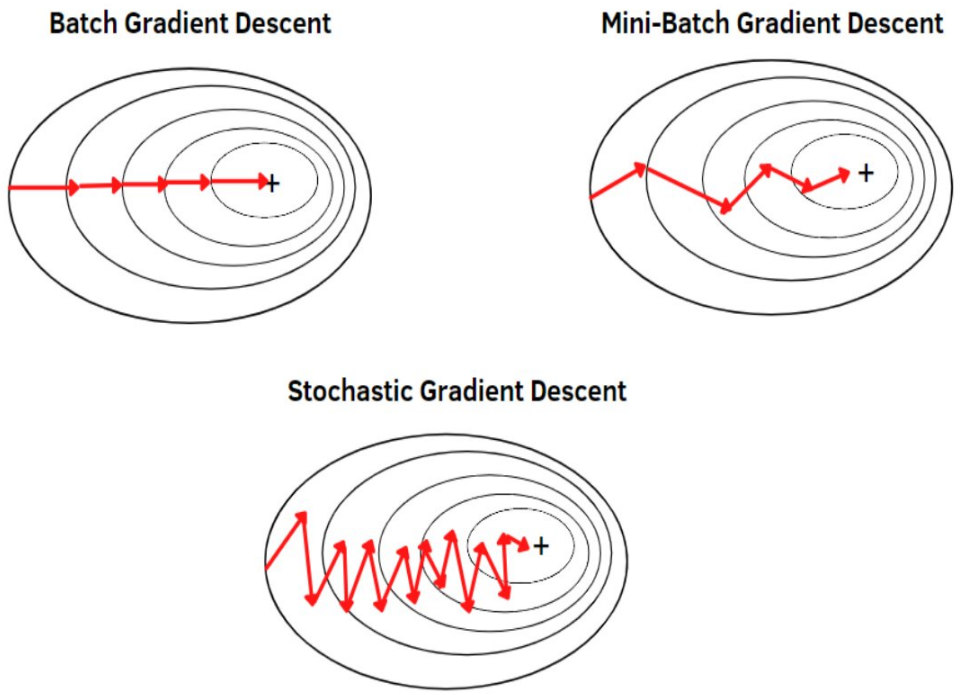
- Pick a random training exemple
- Estimate the loss on that example
- Compute the gradient with respect to this loss
- Take a step in the opposite direction of the gradient
- Repeat with a new example





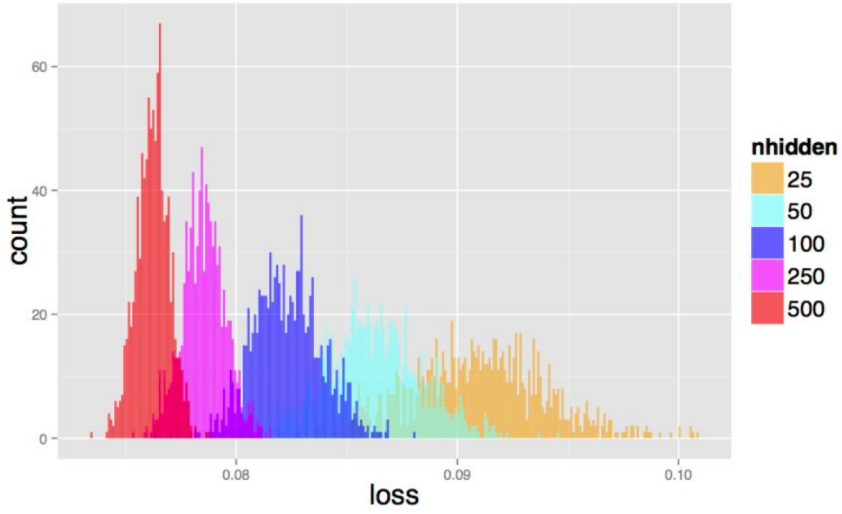
Stochastic Gradient Descent

- Stochastic gradient descent: We compute the gradient with one sample at a time
- Batch gradient descent: we compute the gradient for the entire training set and then take a step towards the minumum
- **Mini-batch gradient descent**: we select a subset of samples (batch) and compute the gradient for that given subset

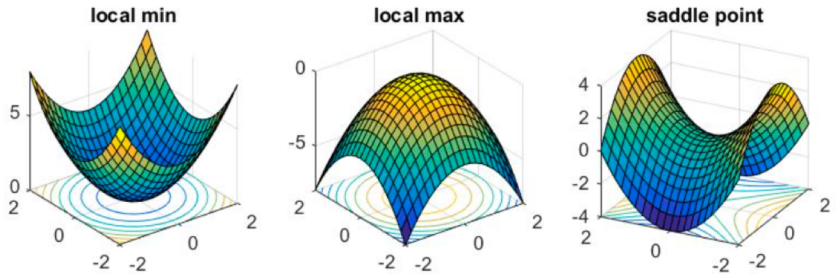


Stochastic Gradient Descent

- Does it get stuck at local minima? It does, but...
- Theory and experiments suggest that for high dimensional deep models, value of loss function at most local minima is close to value of loss function at global minimum
- Weight initialization is important: small non-zero values

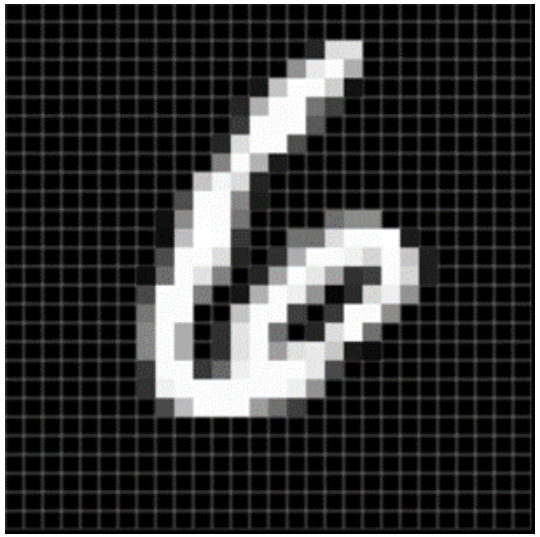


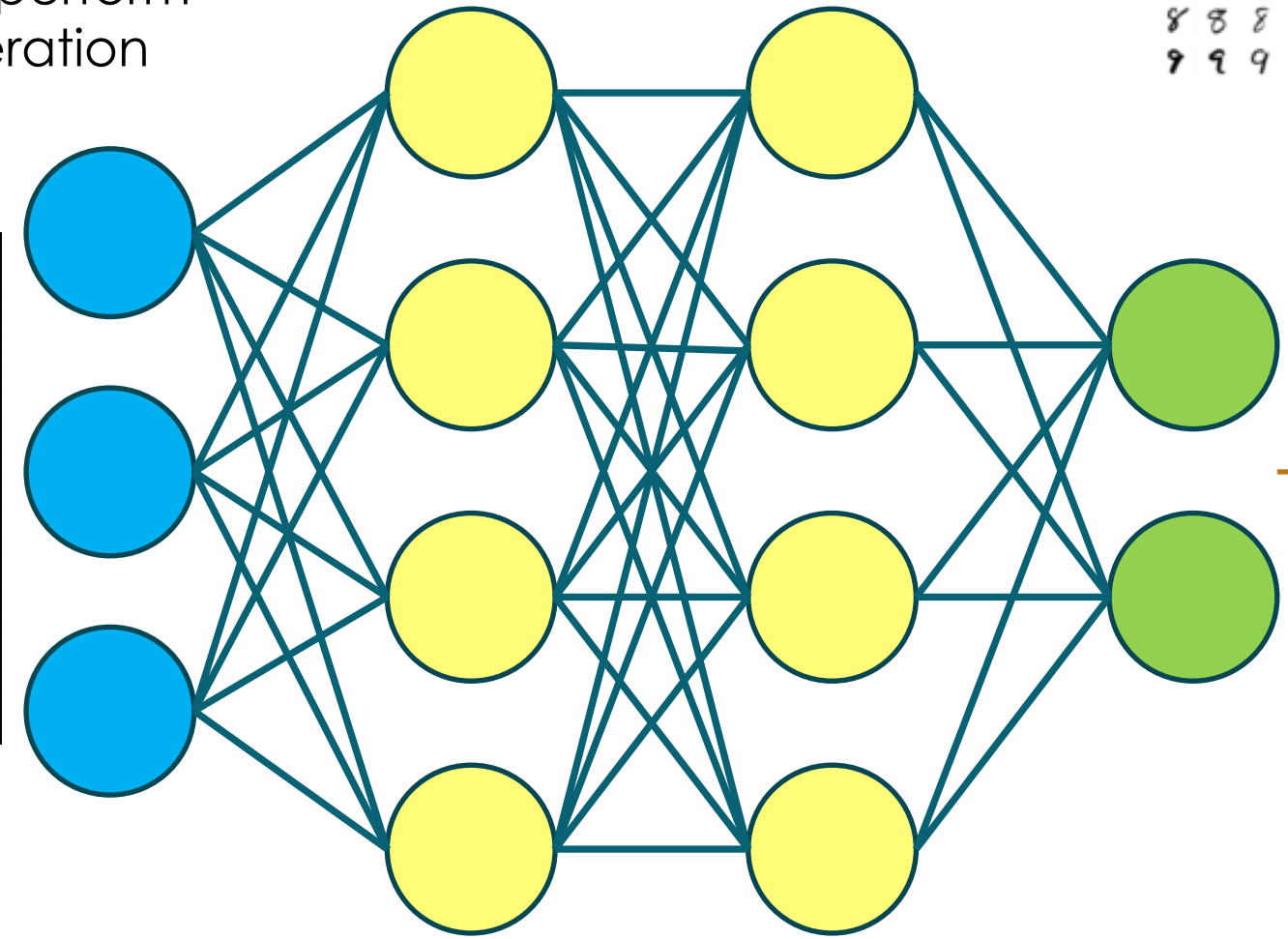
Value of local minima found by running SGD for 200 iterations on a simplified version of MNIST from different initial starting points. As number of parameters increases, local minima tend to cluster more tightly.



Training a Neural Network

Input image and perform feed-forward operation





$$\begin{bmatrix} 0.1 \\ 0.02 \\ 0.05 \\ 0.03 \\ 0.03 \\ 0.01 \\ 0.7 \\ 0.01 \\ 0.03 \\ 0.02 \end{bmatrix}$$

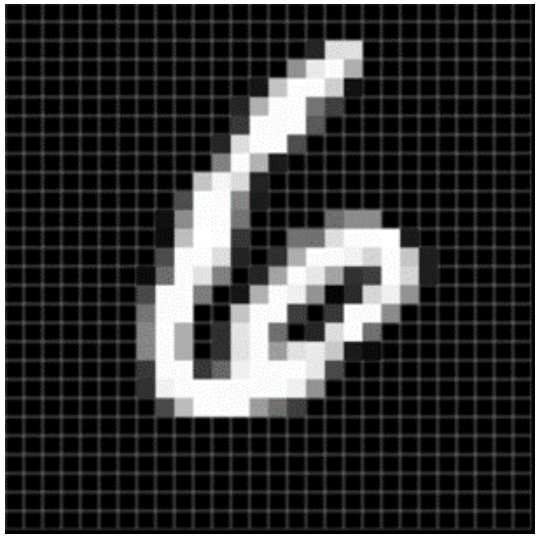
Output

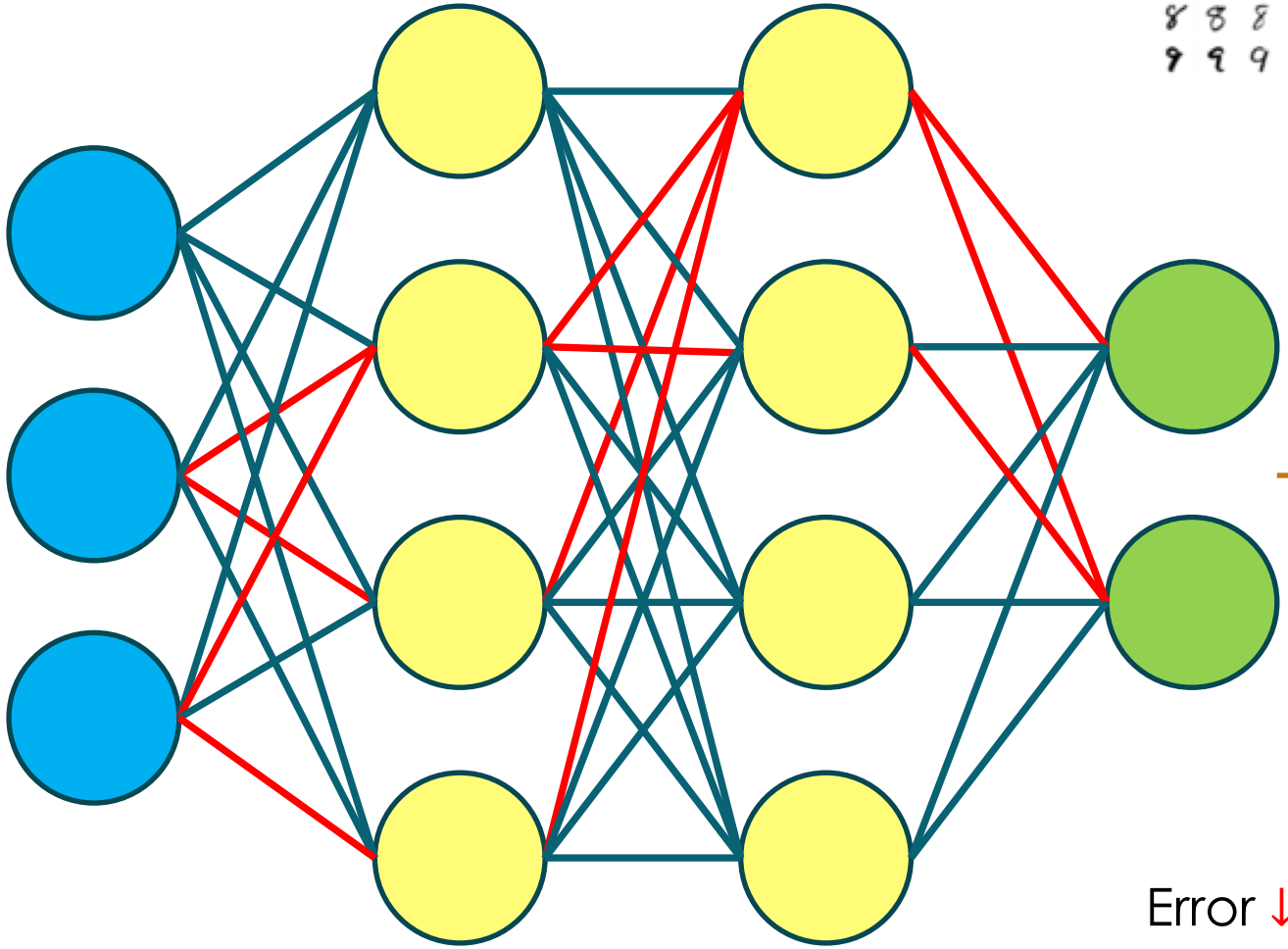
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Label

Training a Neural Network

Update weights





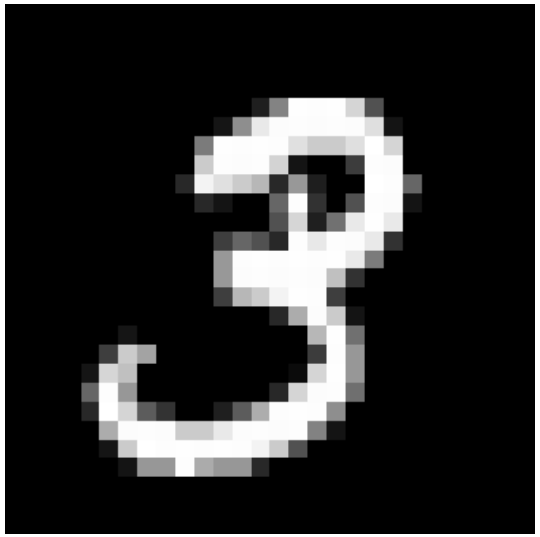
$$\begin{bmatrix} 0.1 \\ 0.02 \\ 0.04 \\ 0.02 \\ 0.02 \\ 0.01 \\ 0.74 \\ 0.01 \\ 0.02 \\ 0.02 \end{bmatrix} \quad \text{error} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

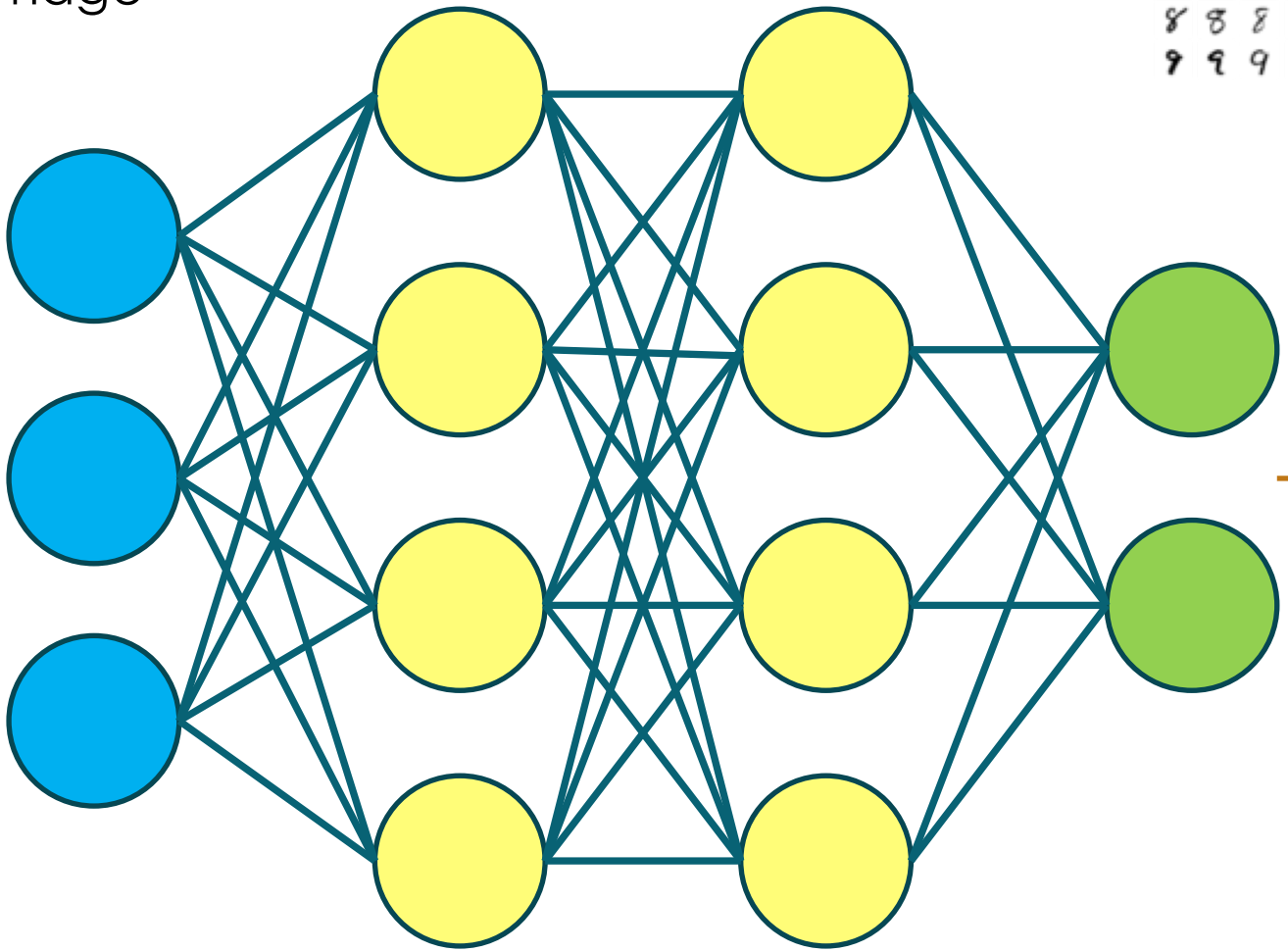
Output Label

Error ↓

Training a Neural Network

Repeat with new image





$$\begin{bmatrix} 0.1 \\ 0.03 \\ 0.03 \\ 0.5 \\ 0.02 \\ 0.07 \\ 0.18 \\ 0.02 \\ 0.01 \\ 0.02 \end{bmatrix}$$

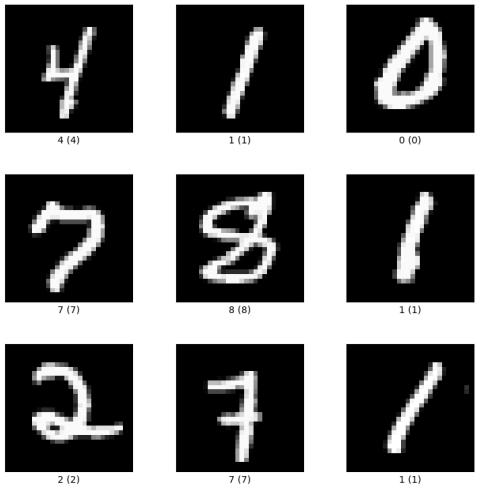
Output

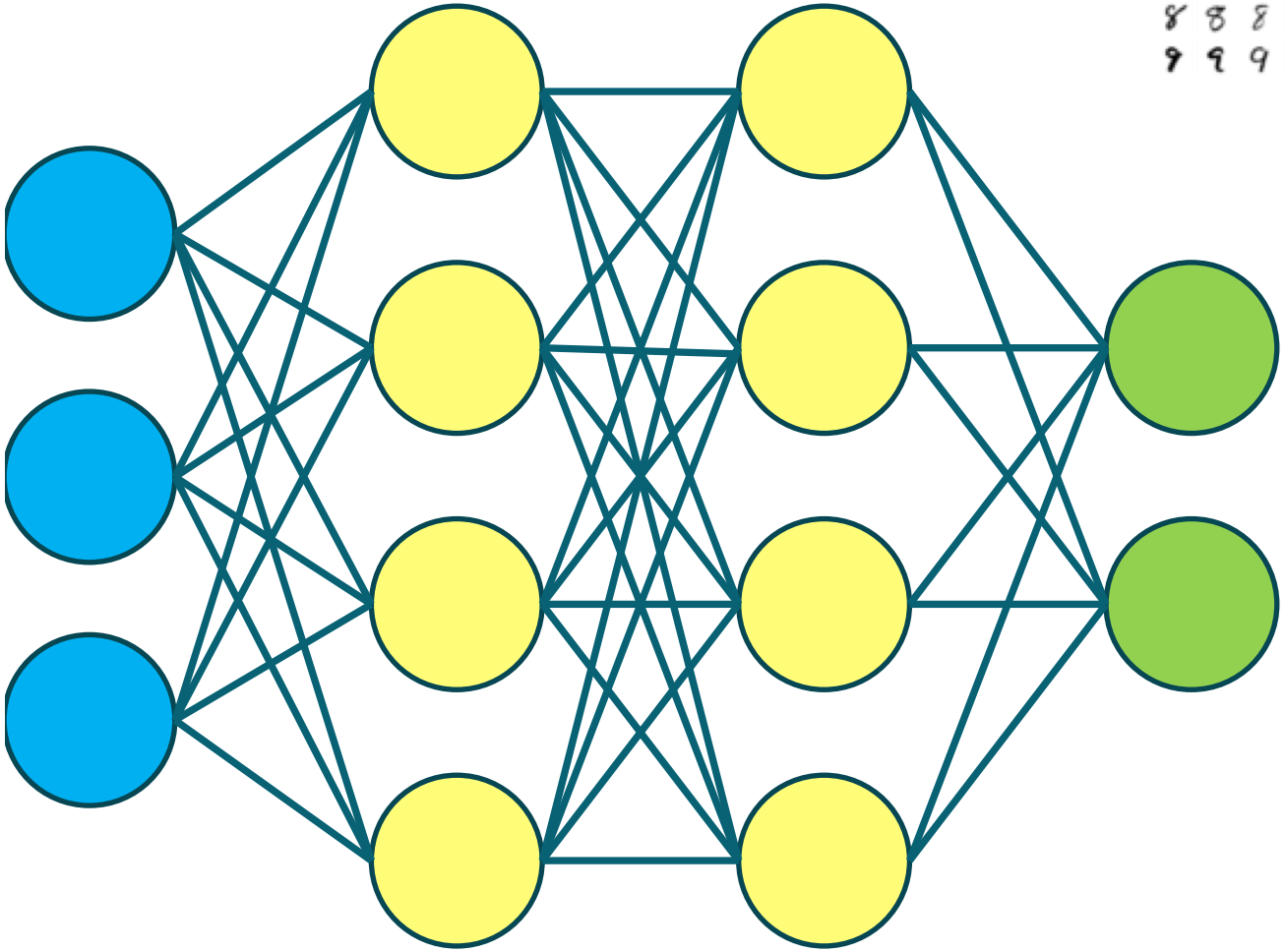
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Label

Training a Neural Network

Batch





$$\begin{pmatrix} 0.1 & 0.2 & \dots \\ 0.3 & 0.05 & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Outputs

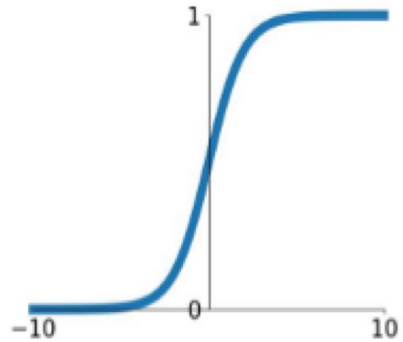
$$\begin{pmatrix} 0 & 0 & \dots \\ 0 & 1 & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Labels

Activation Functions

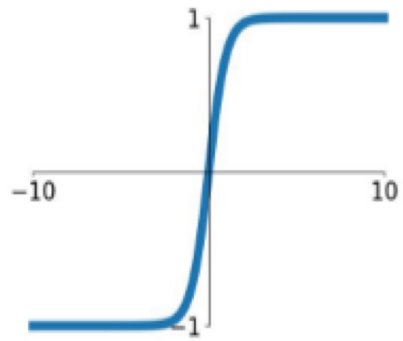
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



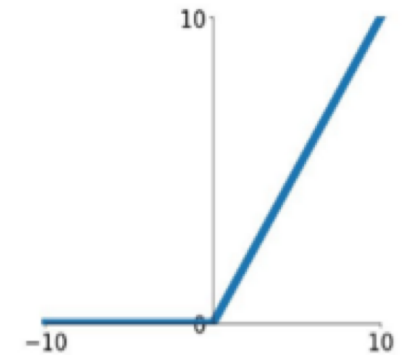
tanh

$$\tanh(x)$$



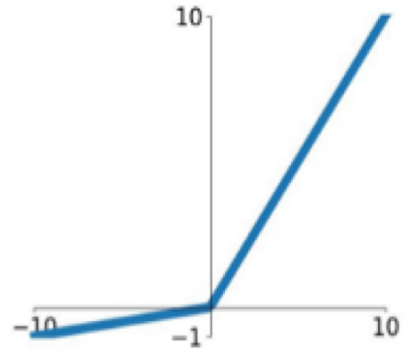
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

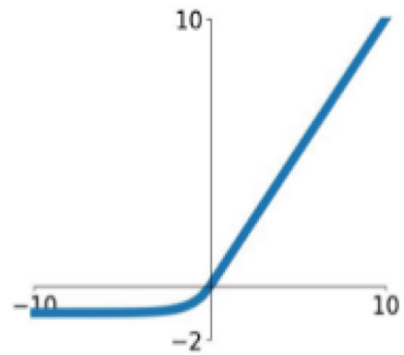


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

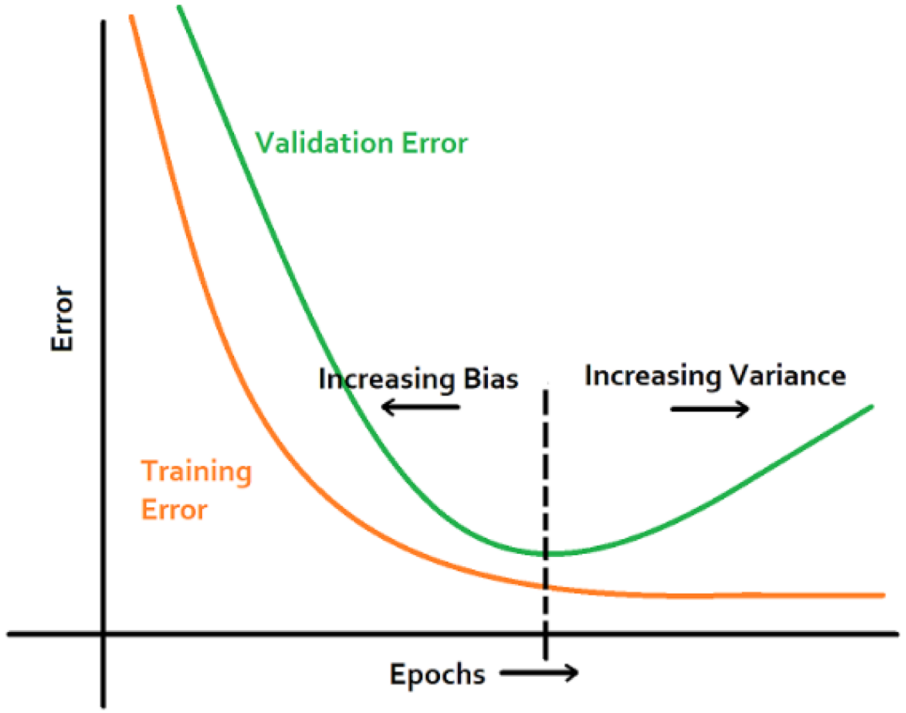


Regularization

- Neural networks are models with a large number of parameters that are sometimes prone to overfitting
- Regularization techniques are used to improve the ability of neural networks to generalize
- By restricting how much the model can be adjusted to the training set or creating artificial data we can improve the general performance of the model
- This is at the cost of performing worse in the training set, but improving the results on new data

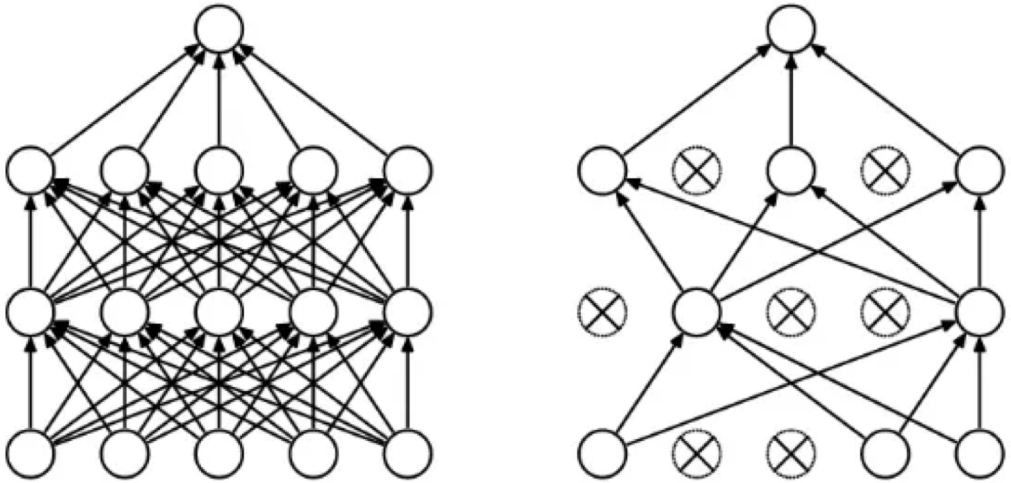
Early stopping

- We can monitor the training and validation errors during the training phase
- At some point we might see that the training loss keeps decreasing while the validation loss starts increasing or stays the same
- Since we are not using the data in the validation set to update the parameters of the model, it is likely that the one with lowest validation error generalizes better than the other models



Dropout

- It helps the neural network learn more general patterns
- During training, the inputs to some of the neurons are randomly set to 0 with a probability of our choice at each iteration
- This prevents the neural network from learning the noise in the training data
- This procedure is only applied during training

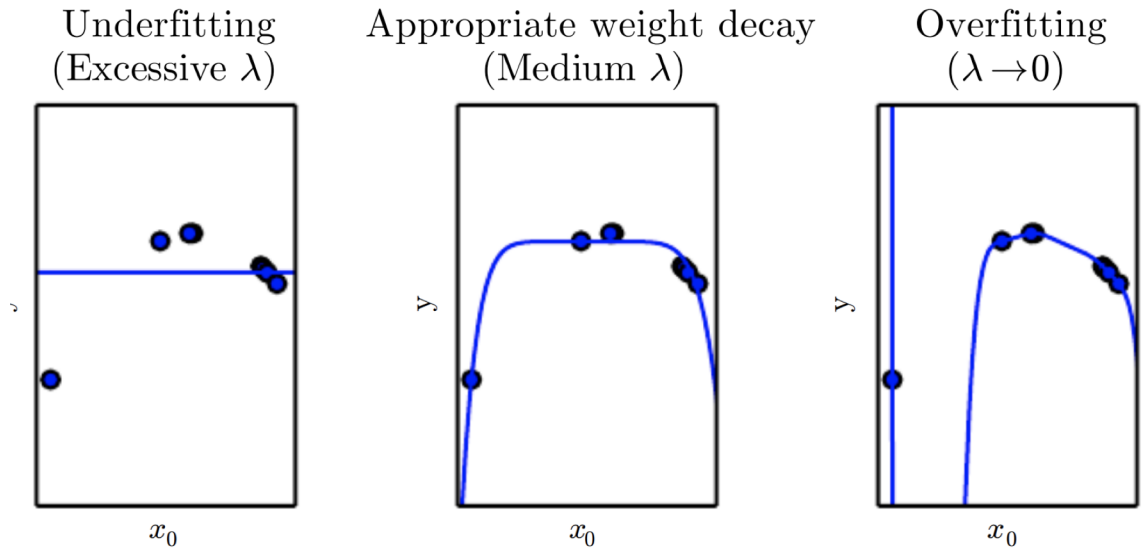


(a) Standard Neural Net (b) After applying dropout.

E.g. $dropout = 0.2$ means 20% of the inputs in the forward propagation will be set to 0

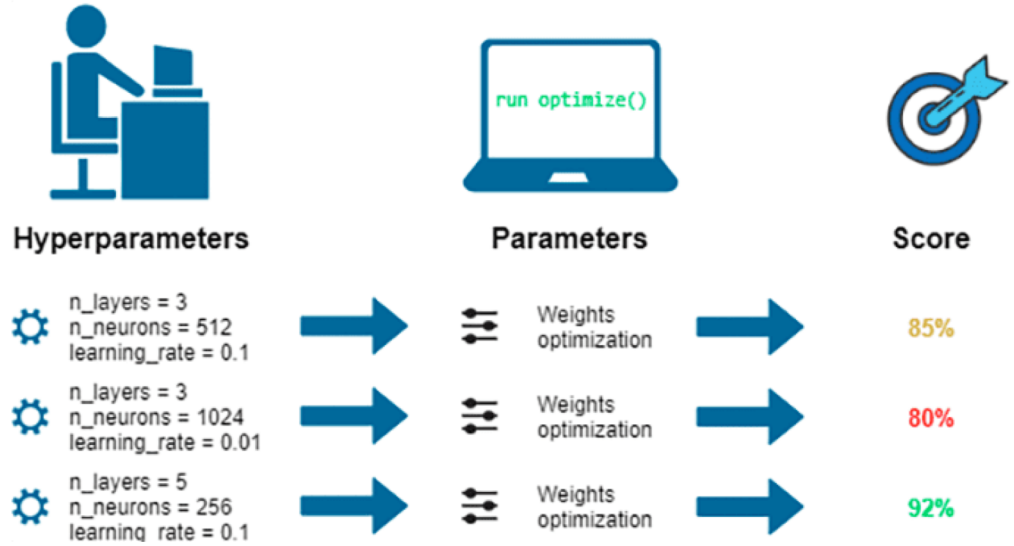
Weight Decay or L2 regularization

- We add a term in the loss function that penalizes large
- weights: $L' = L + \lambda w^2$
- λ is the weight decay (typically < 0.01)
- By restricting how big the weights can get, we avoid giving too much importance to a single connection or training example



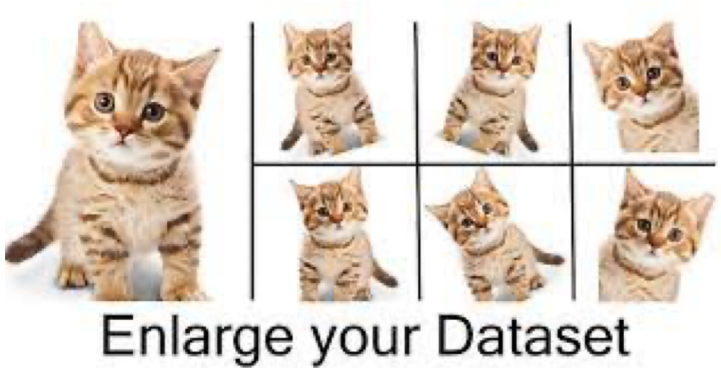
Hyperparameter tuning

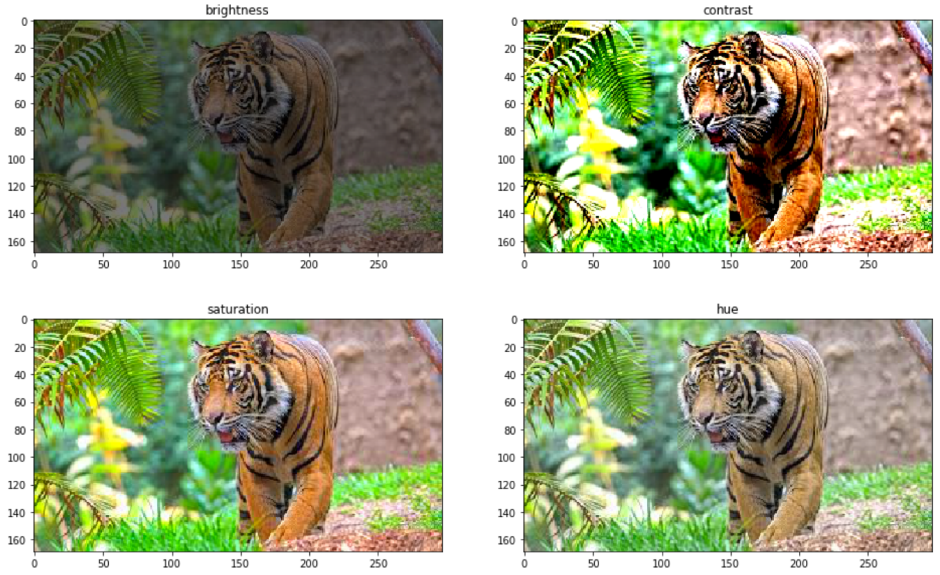
- Learning rate, weight decay, number of epochs, batch size, number of neurons, number of layers…
- They are not "proper" parameters of the model, but they have an effect on training
- **We use the training set to find the model's parameters, the validation set to tune the hyperparameters and the test set to test the model's final performance**

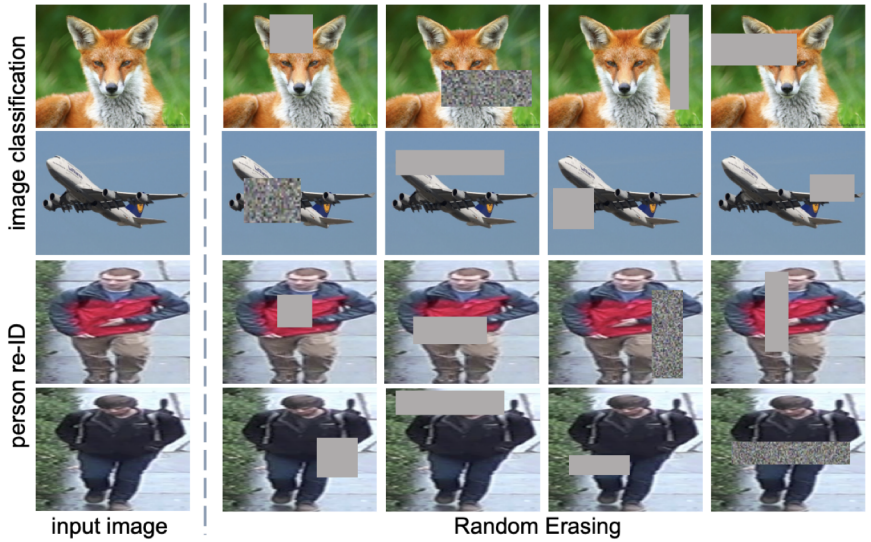


Data Augmentation

- Sometimes we don't have access to large amounts of data
- We can artificially enlarge our training data by adding distortions
- We should still check the performance with the raw test set







Loss functions

- We choose the loss function according to the problem we want to solve
- Ultimately, we want a measure of performance that we can optimize for to solve our problem

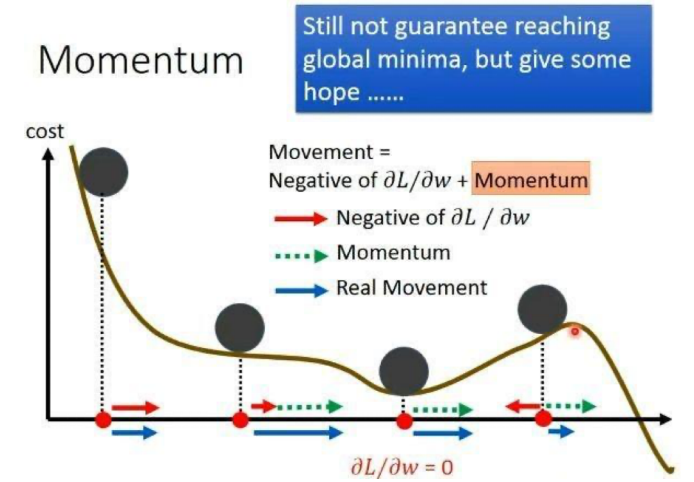
- This measure may not coincide with the loss function, but it will be correlated

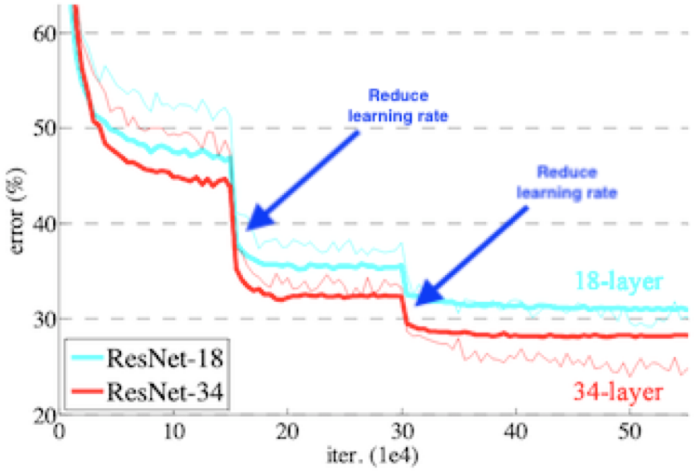
Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] =$	Quantify the difference between two probability

https://pytorch.org/docs/stable/nn.html

Momentum & adaptive learning rate

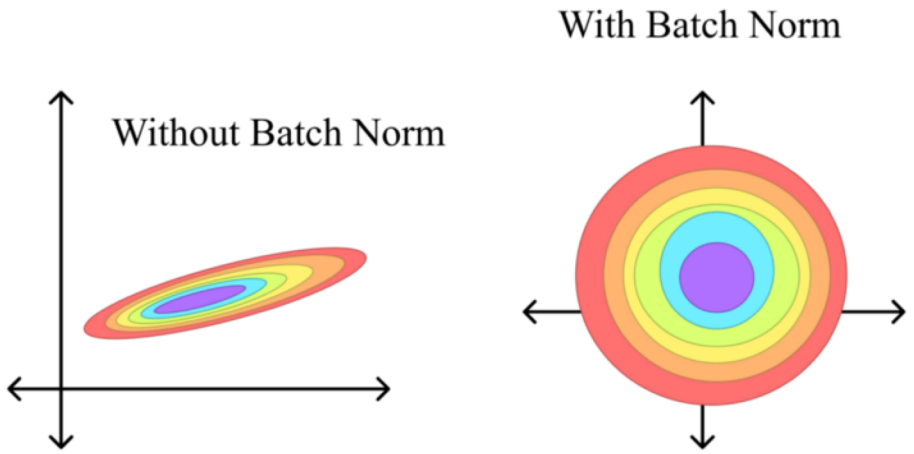
- It simulates momentum in physics
- We add the previous gradient computation to the current one
- This helps overcome small "hills" in the loss function
- Adaptive learning rate starts high and decreases as it gets closer to the minimum





Batch Normalization

- Proposed by Sergey Ioffe and Christian Szegedy in 2015
- It normalizes samples inside the network during forward propagation across the batch
- It makes learning faster and more stable
- It mitigates the problem of *internal covariance shift*



$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \text{ and } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2.$$

Batch of size m