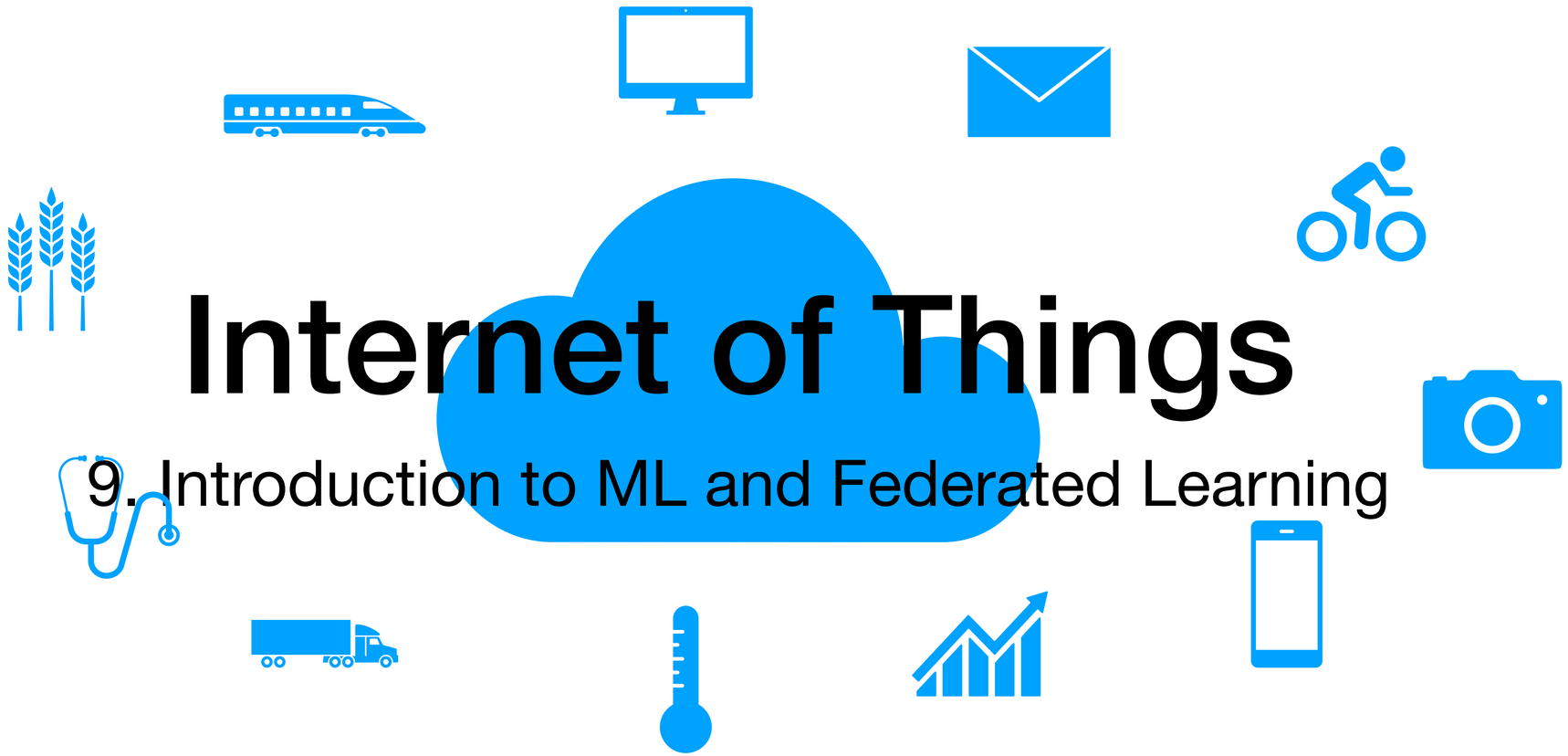


Internet of Things

9. Introduction to ML and Federated Learning

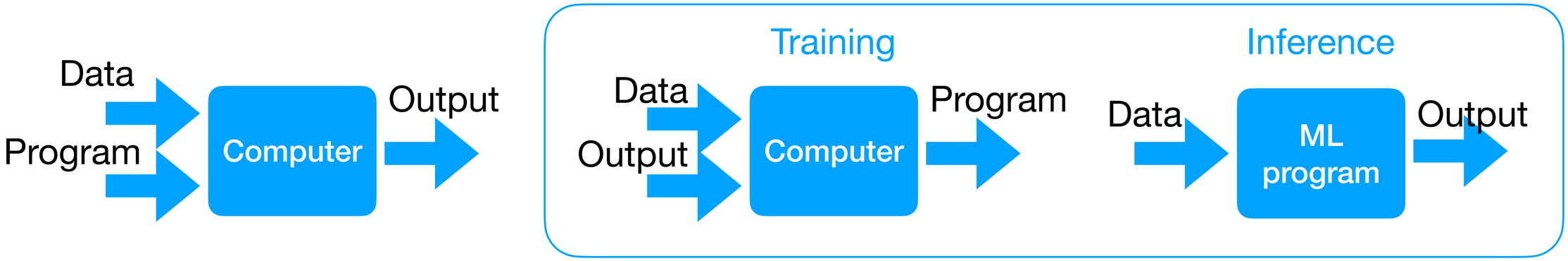
M.Sc. Computer Science 2024-2025

Viviana Arrigoni

ML/AI

- Artificial Intelligence is the field of developing computers and robots that are capable of behaving in ways that mimic human capabilities without human interference.

- Machine Learning is a sub-field of AI that uses algorithms to automatically learn insights and recognize patterns from **data**.

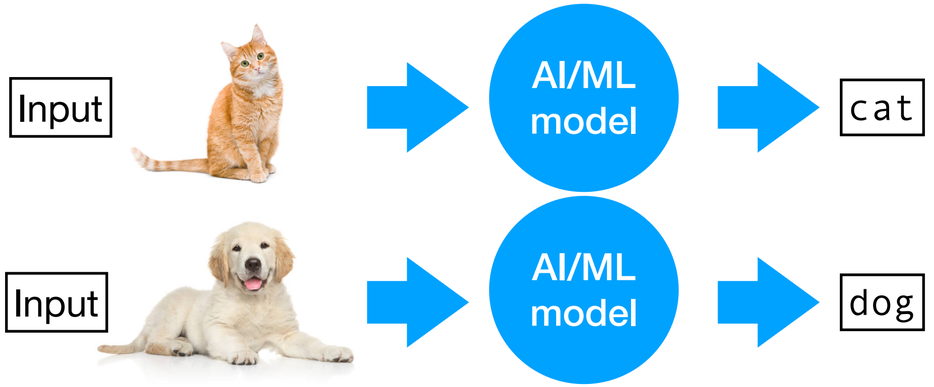


Traditional programming

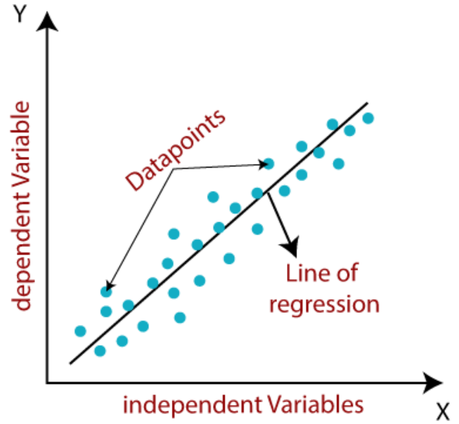
ML programming

Classification and regression

- In machine learning, there are two main types of task:
  - **Classification**- involves predicting a category or **class label**. The output is discrete, meaning the model tries to classify data into predefined labels or groups.

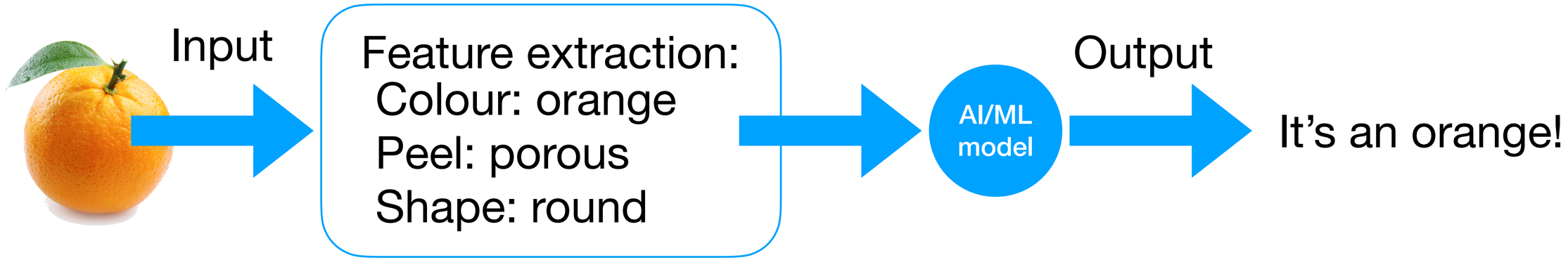


  - **Regression**- involves predicting a (possibly continuous) value or quantity (dependent variable) on a numerical scale by observing variables (features - independent variables) that influence prediction.



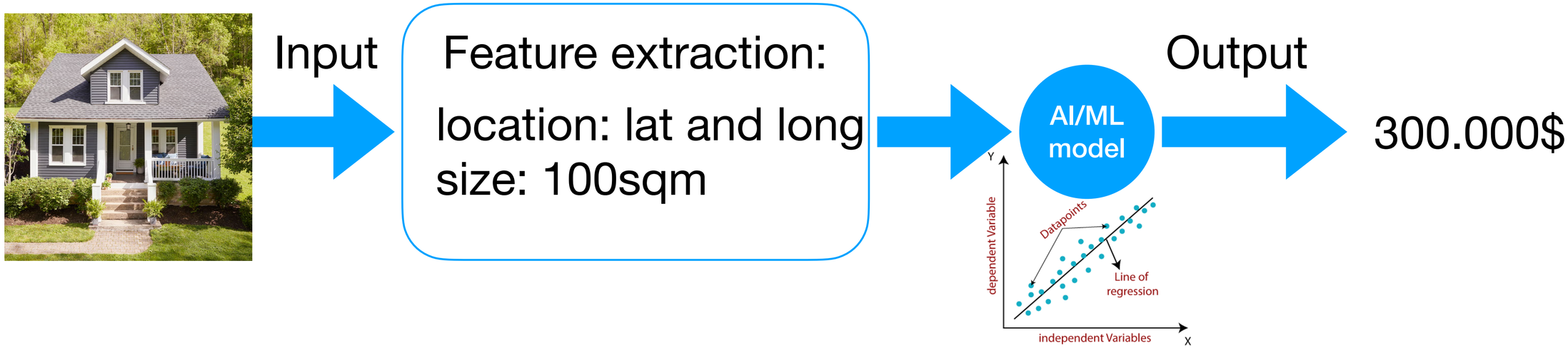
Features

- Data is preprocessed to identify **features**, i.e., characteristics of the input which help in describing it.
- High level classification example:
  task: identify the fruit
  three classes: apples, oranges, pears.
  - Features = colour, peel, shape.



Features

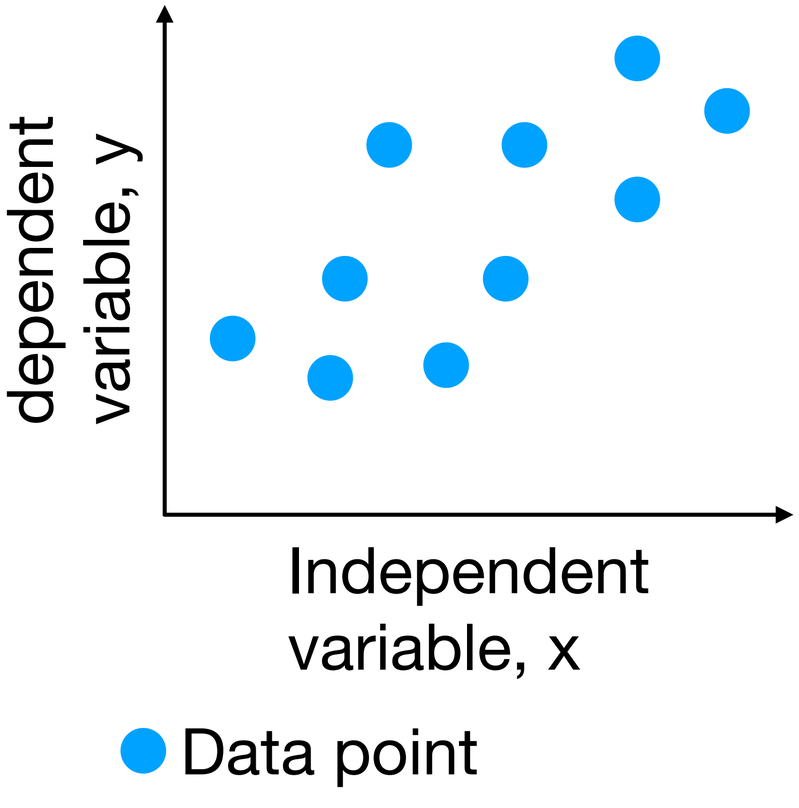
- Data is preprocessed to identify **features**, i.e., characteristics of the input which help in describing it.
- High level regression example:
  task: predict house price
  output: a value in $\mathbb{R}$
  Features = location, size.



9.1 Basics of Regression

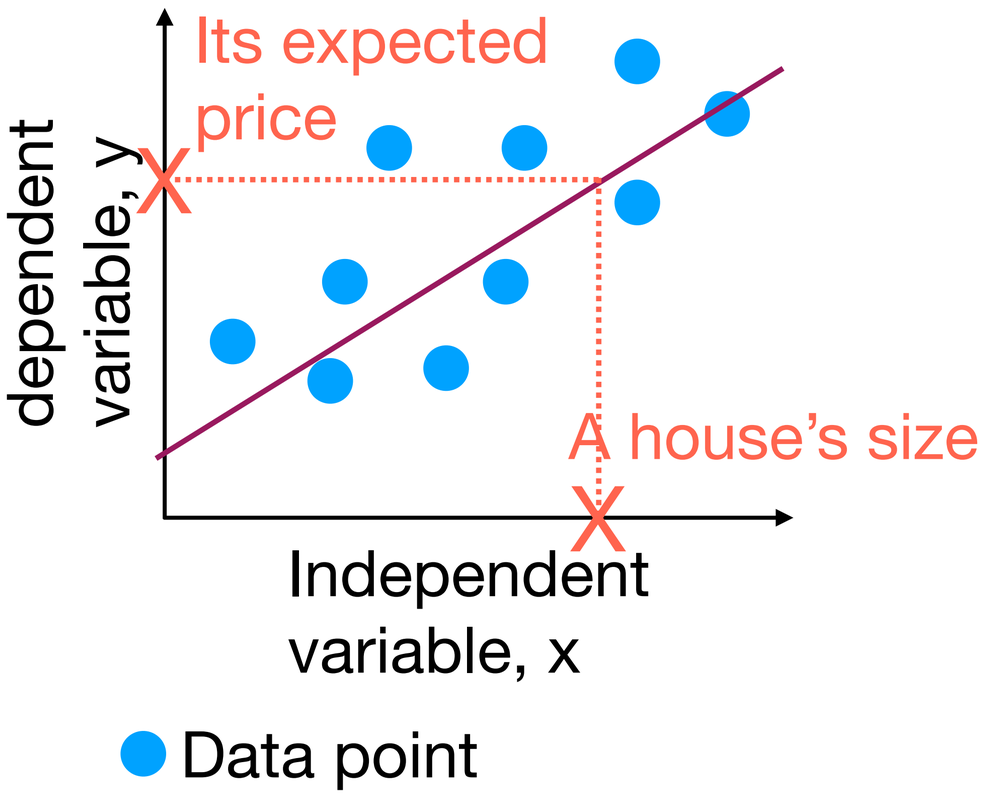
Linear Regression

- Consider a dataset as a set of data point, characterised by an independent variable (feature) and a dependent variable (to predict), for instance:
  - Independent variable = size
  - Dependent variable = cost
- Each data point is a house.
- By observing the distribution of the data points, we can notice a linear relationship between the two variables, e.g., the cost of the house grows linearly with its size.
- Objective: find the line that better “fits” the data point.

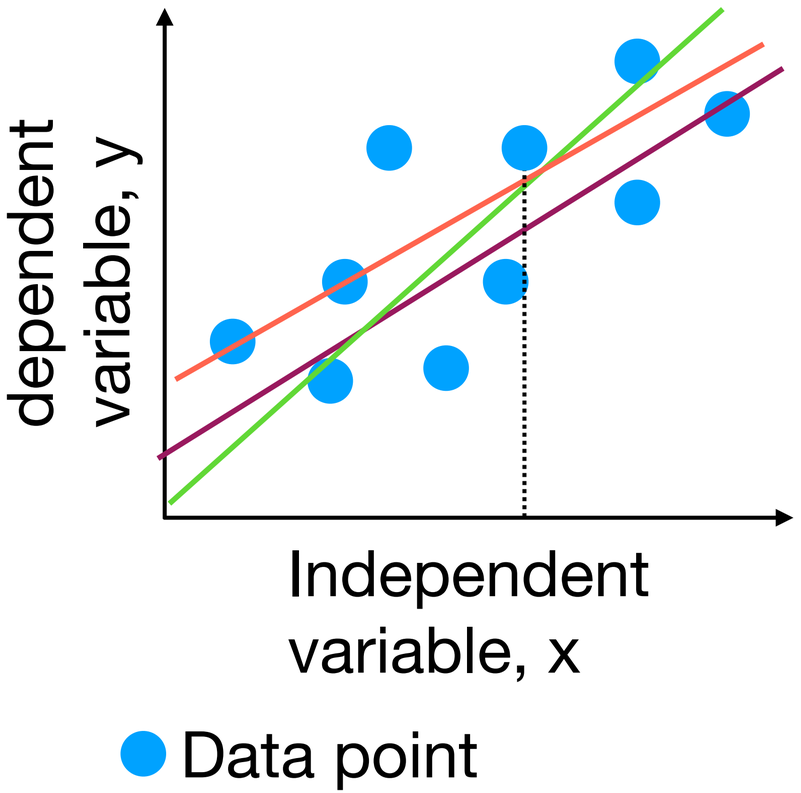


Linear Regression

- The line that better “fits” the data point is the line that better represents the relationship between the two variables, given a set of data points.
  - By finding such a line, given the size of a house, we can predict how much it costs with some confidence.

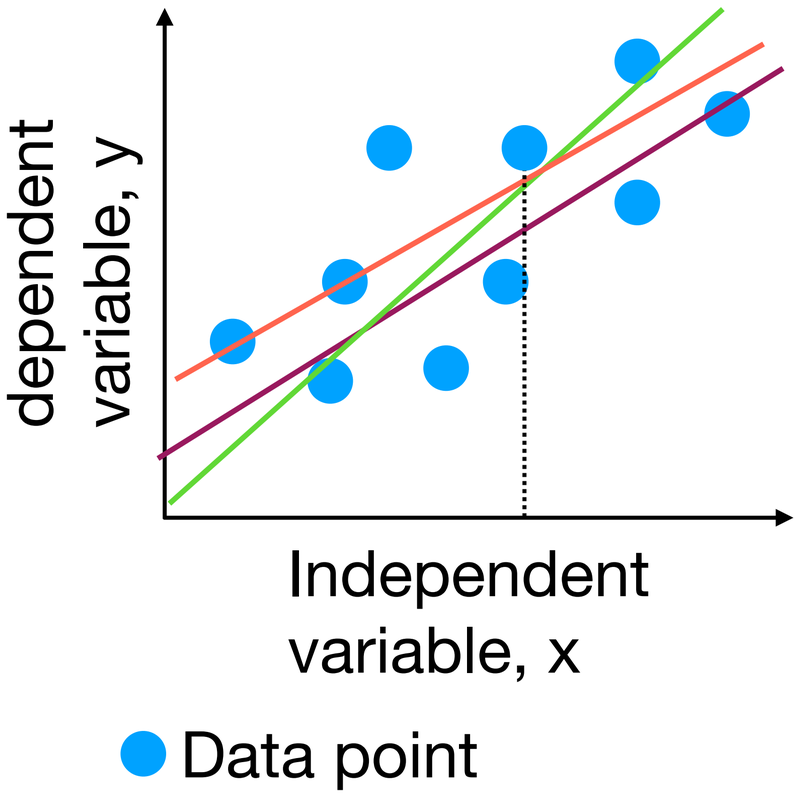


Linear Regression



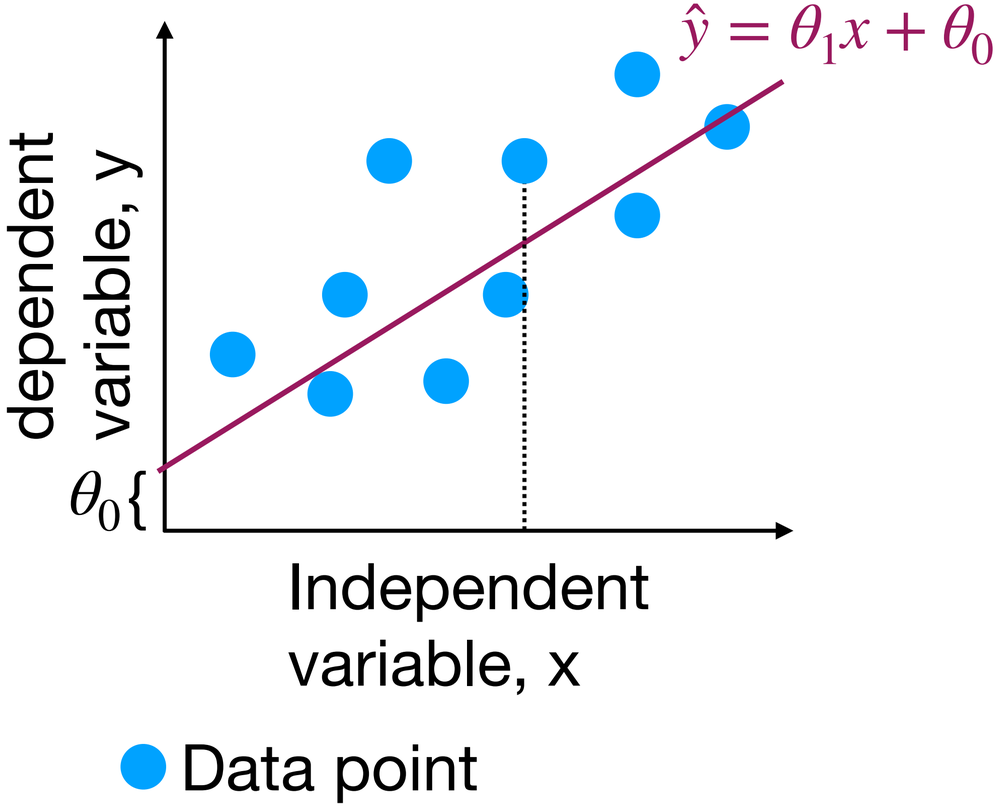
Linear Regression

- We can fit many lines, none of them is “perfect”, since data points are not aligned.
- How to we find the “best” one? And how do we decide if a line is “better” than the other?



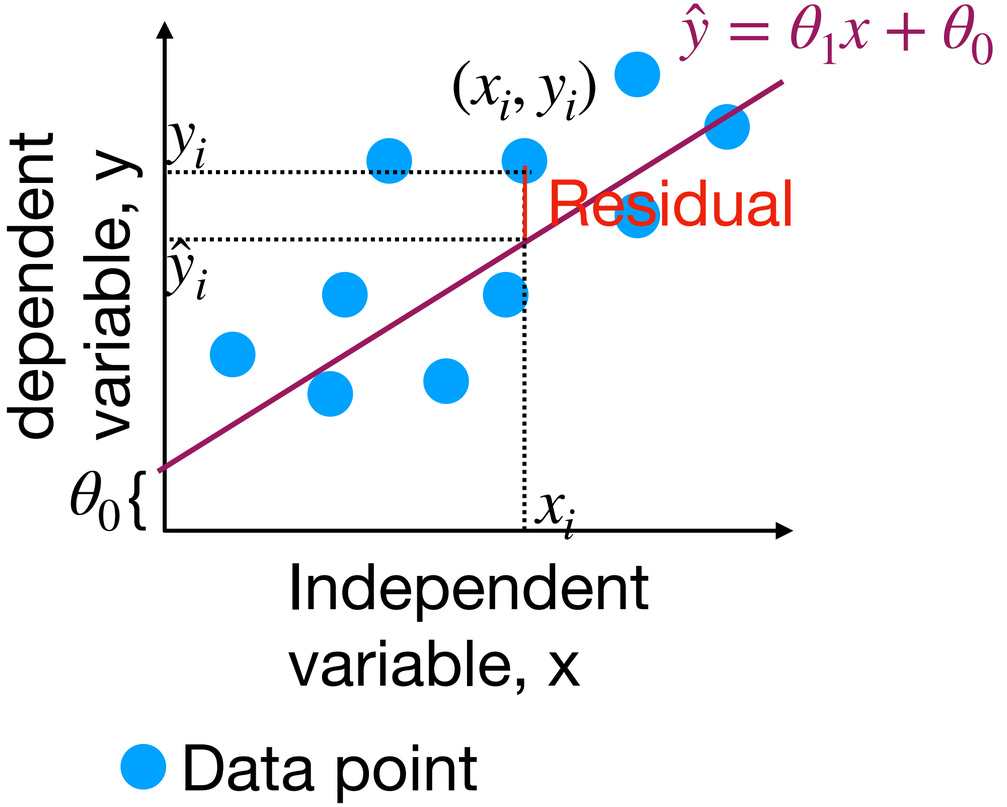
Linear Regression

- We can fit many lines, none of them is “perfect”, since data points are not aligned.
- How to we find the “best” one? And how do we decide if a line is “better” than the other?
- Recall that the function of a line is:
  $\hat{y} = \theta_1 x + \theta_0$,
  where θ_1 is the slope and θ_0 is the intercept.



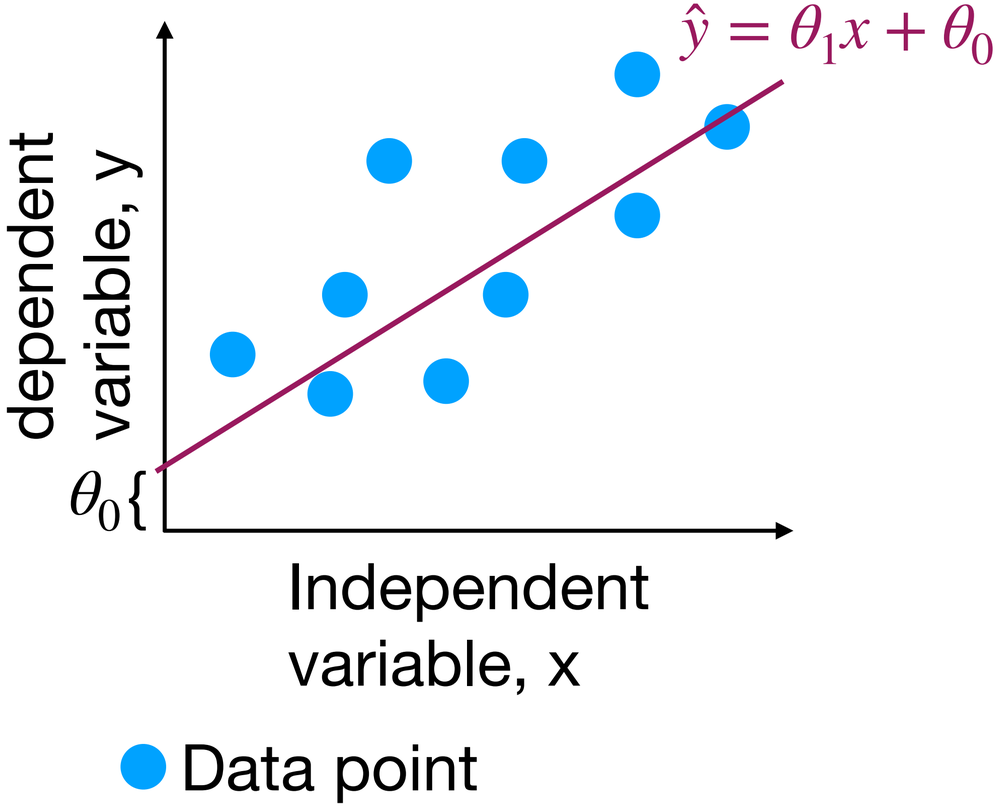
Linear Regression

- We can fit many lines, none of them is “perfect”, since data points are not aligned.
- How to we find the “best” one? And how do we decide if a line is “better” than the other?
- Recall that the function of a line is:
  $\hat{y} = \theta_1 x + \theta_0$,
  where θ_1 is the slope and θ_0 is the intercept.
- Given the function of a line, for each point (x_i, y_i), we can compute the **residual**, i.e., the difference between y_i (real value) and $\hat{y}_i = \theta_1 x_i + \theta_0$ (predicted value).



Linear Regression

- To define how good a line fits the data, we can define different *loss functions,* that are functions of the residuals of the data points in the dataset.
  - L1 loss = $\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - L2 loss = $\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
  - Mean Square Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
- Where n is the number of data points.



Which loss function to use? Usually…

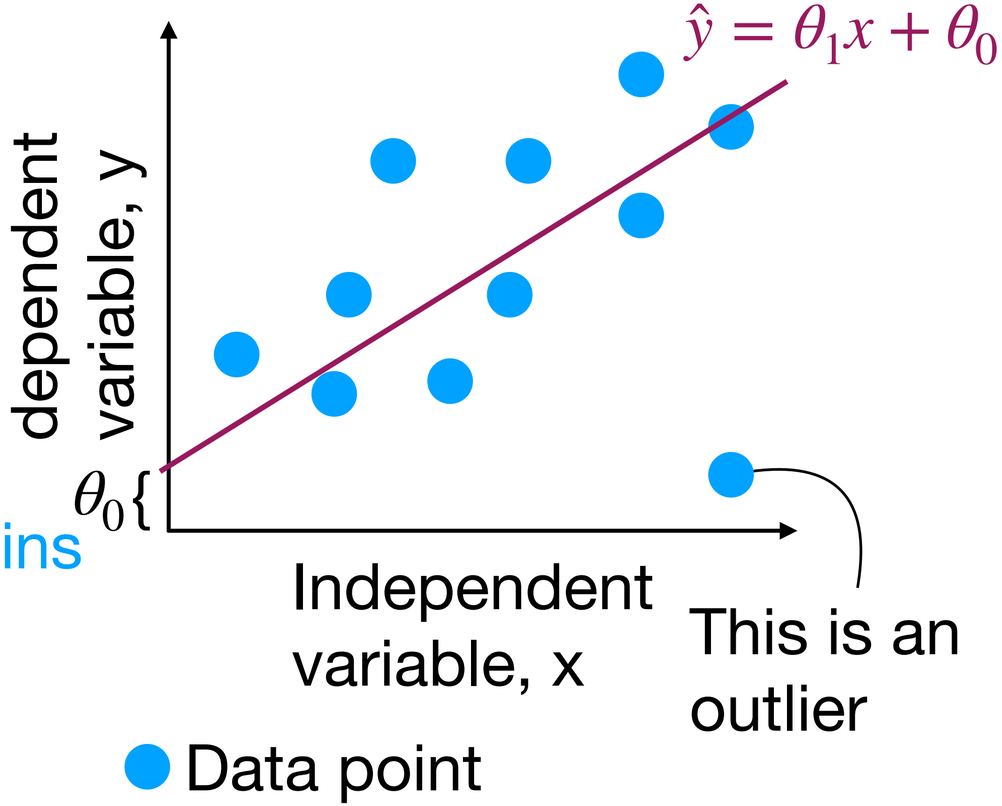
Linear Regression

- To define how good a line fits the data, we can define different *loss functions,* that are functions of the residuals of the data points in the dataset.
  - L1 loss = $\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - L2 loss = $\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
  - Mean Square Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
- Where n is the number of data points.

When data set contains many outliers

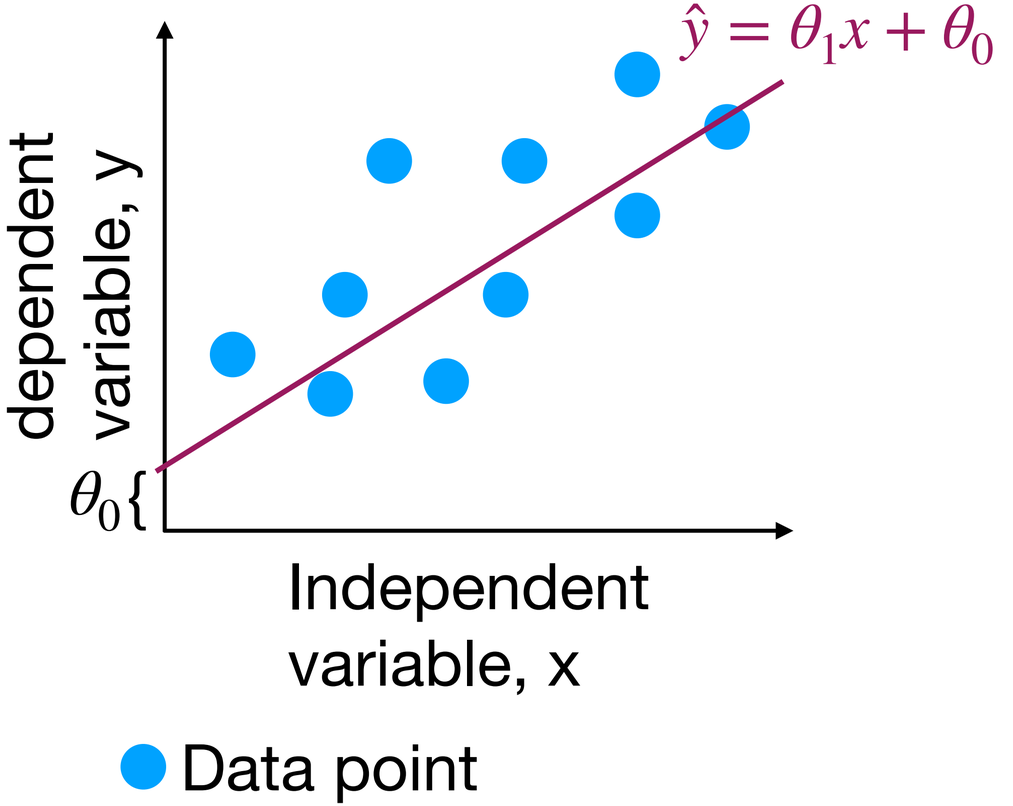
When data set does not contain many outliers

Which loss function to use? Usually…



Linear Regression

- To define how good a line fits the data, we can define different *loss functions,* that are functions of the residuals of the data points in the dataset.
  - L1 loss = $\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$
  - L2 loss = $\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
  - Mean Square Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$
- Where n is the number of data points.



Objective: choose one of these loss functions, and find the line that minimises it

Linear Regression

- For example, if we want to find the line that minimises the MSE, we need to solve the following problem:
  - $\min_{\theta_0,\theta_1} f_{MSE}(\theta_0, \theta_1) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\theta_i x_i + \theta_0 - y_i)^2$
- Analytically:
  - Find the zeros of the partial derivatives: $\frac{\partial f_{MSE}}{\partial \theta_0} = 0$, $\frac{\partial f_{MSE}}{\partial \theta_1} = 0$ ((θ_0^c, θ_1^c) "critical point", each is either a point of minimum, of maximum or a saddle point).
  - Compute the determinant of the Hessian matrix, $D = \frac{\partial^2 f_{MSE}}{\partial \theta_0^2} \cdot \frac{\partial^2 f_{MSE}}{\partial \theta_1^2} - \left(\frac{\partial^2 f_{MSE}}{\partial \theta_0 \partial \theta_1}\right)^2$, evaluate it in the critical points. If $D(\theta_0^c, \theta_1^c) > 0$ and $\frac{\partial^2 f_{MSE}(\theta_0^c, \theta_1^c)}{\partial \theta_0^2} > 0$, then (θ_0^c, θ_1^c) is a point of minimum, so the line that minimises the MSE has equation $\hat{y} = \theta_1^c x + \theta_0^c$

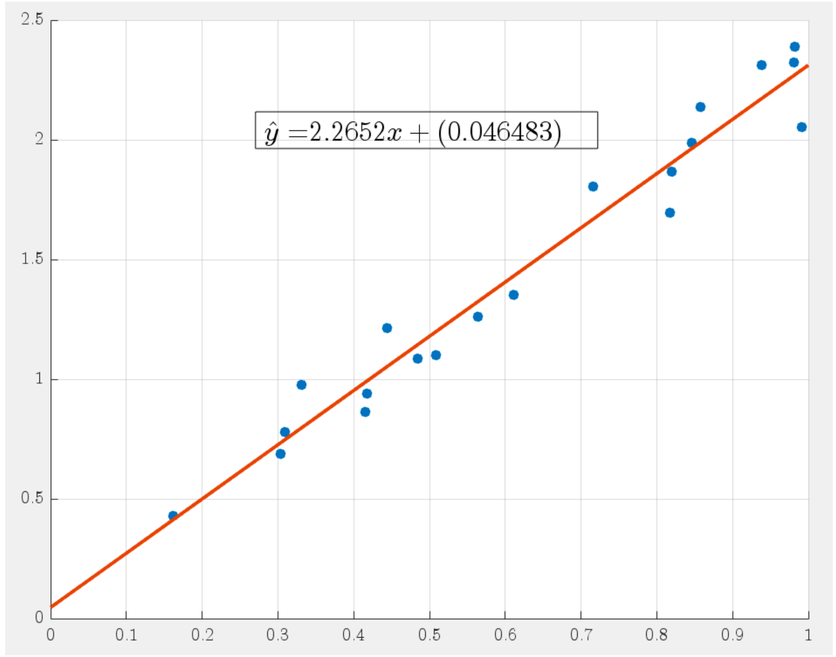
Linear Regression

- For example, if we want to find the line that minimises the MSE, we need to solve the following problem:
  - $\min_{\theta_0,\theta_1} f_{MSE}(\theta_0, \theta_1) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\theta_i x_i + \theta_0 - y_i)^2$
- Geometrically:
  - Let $X = [x_1, \ldots, x_n]^T$, $Y = [y_1, \ldots, y_n]^T$, and $X_+ = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$. Let $\theta = [\theta_0, \theta_1]^T$. Then:
    - $\theta = (X_+^T \cdot X_+)^{-1} \cdot (X_+^T \cdot Y)$
  - $X_+^T \cdot X_+\theta = X_+^T Y$ is called **normal equation.**

Linear Regression

- For example, if we want to find the line that minimises the MSE, we need to solve the following problem:
  - $$\min_{\theta_0,\theta_1} f_{MSE}(\theta_0,\theta_1) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\theta_i x_i + \theta_0 - y_i)^2$$
- Geometrically:

```
X = rand(20,1);
X = sort(X);
Y = 2 * X + 0.5*rand(20,1);
Xp = [ones(20,1),X];
theta = inv(Xp'*Xp)*(Xp'*Y);
```

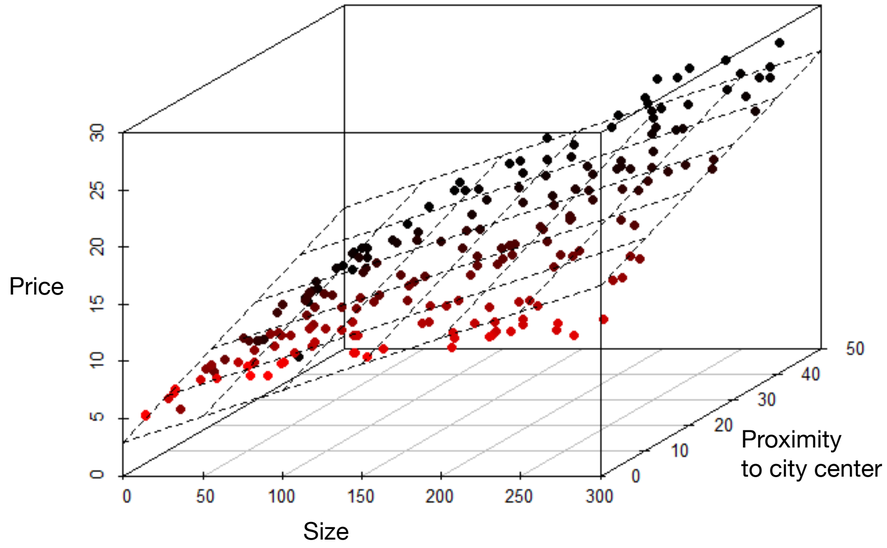


Linear Regression

- If the matrix $X_+^T \cdot X_+$ is not invertible, linear regression can still be used by computing its pseudo-inverse.
- The geometrical approach is very powerful because it gives a closed-form formula to perform linear regression for MSE minimisation!
- Very often, we do not have just one independent variable, but multiple ones.
  - For instance, the price of a house does not only depend on its size, but also on other factors, e.g., on the proximity to the city center.
  - Assume that also the relationship between the proximity to the city center and the price is also linear.

Linear Regression

- Each data point has 3 coordinates, (x, y, z), that are (size, proximity, price).
- We can still perform linear regression! In this case, we are searching for the **plane** that minimises a loss function. The equation of the plane is:
  - $\hat{z} = \theta_2 y + \theta_1 x + \theta_0$



- In general, if data points have coordinates: $(x_1, \ldots, x_d, y)$, then we want to find the **hyperplane**, which has general equation

$$\hat{y} = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$

that minimises a loss function.

Linear Regression

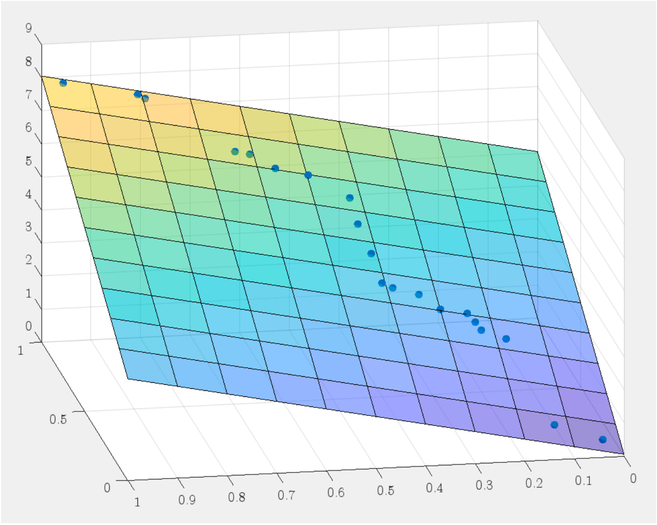
Linear Regression

- Can we still use the normal equations approach to solve this?

Linear Regression

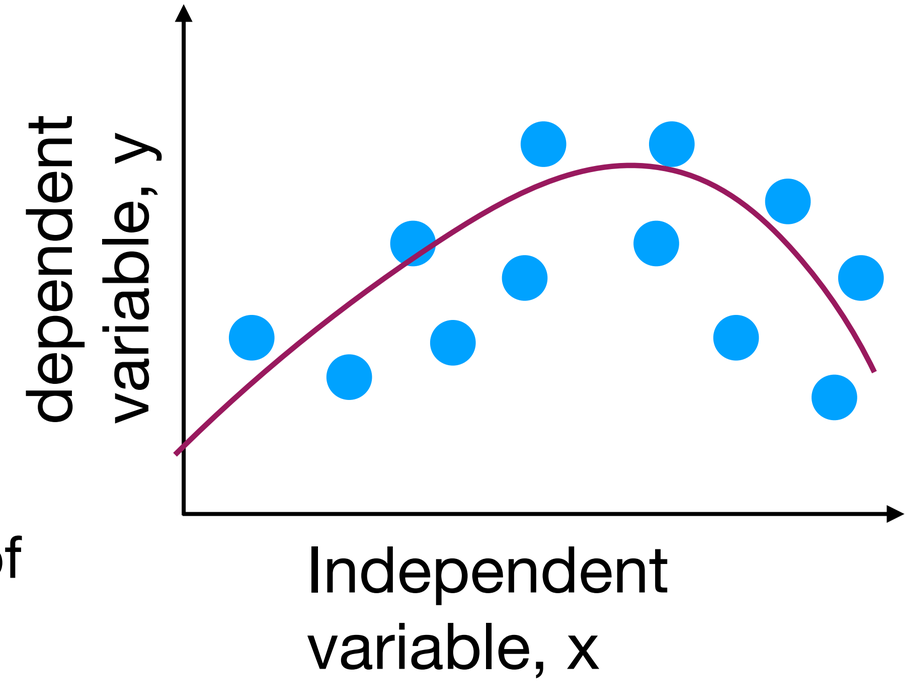
- Can we still use the normal equations approach to solve this?
  - Yes! It is enough to define X_+ as the matrix of size $n \times (d+1)$, where n is the number of data points, and d is the number of features (that are the independent variables - in the previous example, the house size and proximity to the center). The first column of X_+ is all ones. Y is still the vector $Y = [y_1, \ldots, y_n]^T$, and $\theta = [\theta_0, \theta_1, \ldots, \theta_d]^T$

```
Z = 5*X + 3*Y + rand(20,1)*0.1;
Xp = [ones(20,1), X, Y];
theta = inv(Xp'*Xp)*(Xp'*Z);
```



Polynomial Regression

- What if the data points do not exhibit a linear behaviour, but rather a polynomial one?
- In this case, we can search for the best polynomial that fits the data.
- $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_m x^m$
- As before, we can do that by searching for the coefficients $(\theta_0, \theta_1, \ldots, \theta_m)$ that minimise a loss of function.

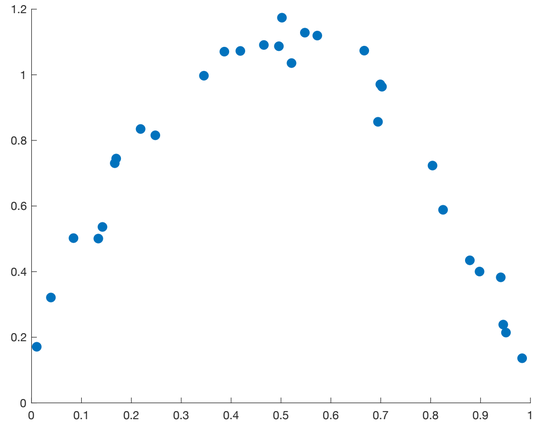


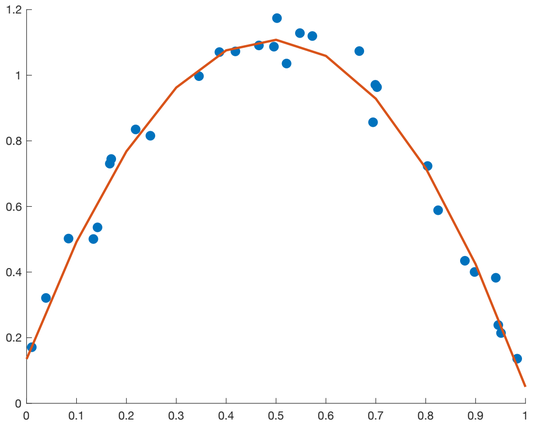
- Notice that, in the scenario that we are considering now, the dependent variable only depends on one independent variable, but it depends also on its powers.

Polynomial Regression

- If we want to find the polynomial that minimises the Mean Square Error, can we still use the normal equation?
- Yes! Given n data points (x_i, y_i), we define
- $$X_+ = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \text{ (known)}$$

  and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$ (unknown)
- Then we can find $\theta = (X_+^T \cdot X_+)^{-1} \cdot (X_+^T \cdot Y)$





Polynomial Regression

- What if the polynomial behaviour occurs over multiple variables?
- We can still use normal equations. Given n data points $(x_{1,i}, \ldots, x_{d,i}, y_i)$, we define
- $$X_+ = \begin{bmatrix} 1 & x_{1,1} & x_{1,1}^2 & \cdots & x_{1,1}^m & \cdots & x_{1,d} & x_{1,d}^2 & \cdots & x_{1,d}^m \\ 1 & x_{2,1} & x_{2,1}^2 & \ldots & x_{2,1}^m & \cdots & x_{2,d} & x_{2,d}^2 & \ldots & x_{2,d}^m \\ \vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_{n,1} & x_{n,1}^2 & \ldots & x_{n,1}^m & \cdots & x_{n,d} & x_{n,d}^2 & \ldots & x_{n,d}^m \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$ (known)

  and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{d \cdot m} \end{bmatrix}$ (unknown)

- Then we can find $\theta = (X_+^T \cdot X_+)^{-1} \cdot (X_+^T \cdot Y)$

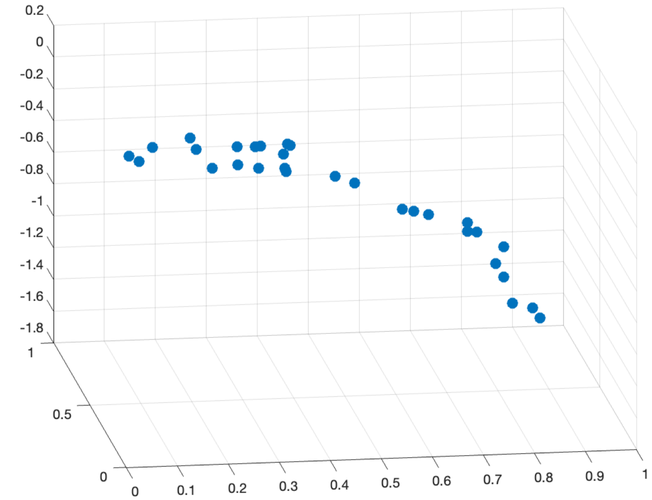
n = number of data points

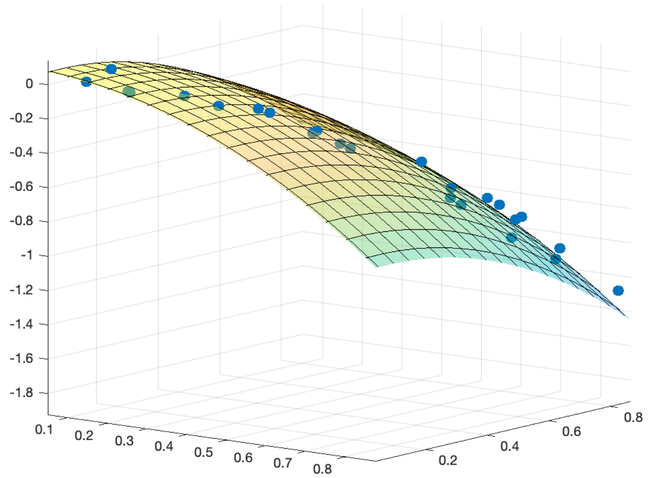
d = number of features

m = degree of the polynomials

Polynomial Regression

- Example with 30 datapoints, 2 features, degree = 2
- $$X_+ = \begin{bmatrix} 1 & x_{1,1} & x_{1,1}^2 & x_{1,2} & x_{1,2}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{30,1} & x_{30,1}^2 & x_{30,2} & x_{30,2}^2 \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{30} \end{bmatrix}$$
- $f = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_2 + \theta_4 x_2^2$





- The problems with this approach are:
  - Computing the (pseudo) inverse of $X_+^T \cdot X_+$ is computationally expensive if the matrix is big
  - Works only for MSE, and not for other loss functions as Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$
- Alternative: use a **numerical** approach.
- Recall the **chain rule** for derivatives of composition of function:

$$\frac{d[f(g(x))]}{dx} = g'(x) \cdot f'(g(x))$$

Chain Rule

$$\frac{d[f(g(x))]}{dx} = g'(x) \cdot f'(g(x))$$

- **Example 1**
  - $g(x) = 2x + 1, f(x) = \frac{1}{3}x - 2.$
  - The composition of f and g is $f(g(x)) = \frac{1}{3}(2x + 1) - 2.$
  - Its derivative is $\frac{d[f(g(x))]}{dx} = \frac{2}{3}$, which is equal to:

    $$\frac{d[f(g(x))]}{dx} = g'(x) \cdot f'(g(x)) = 2 \cdot \frac{1}{3}$$

Chain Rule

$$\frac{d[f(g(x))]}{dx} = g'(x) \cdot f'(g(x))$$

- **Example 2**
  - $g(x) = 2x^2 + 1, f(x) = 3\log(x).$
  - The composition of f and g is $f(g(x)) = 3\log(2x^2 + 1).$
  - $g'(x) = 4x, f'(x) = \frac{3}{|x|}$, then

$$\frac{d[f(g(x))]}{dx} = g'(x) \cdot f'(g(x)) = 4x \cdot \frac{3}{|g(x)|} = 4x \cdot \frac{3}{2x^2 + 1}$$

9.2 Gradient Descent

Gradient

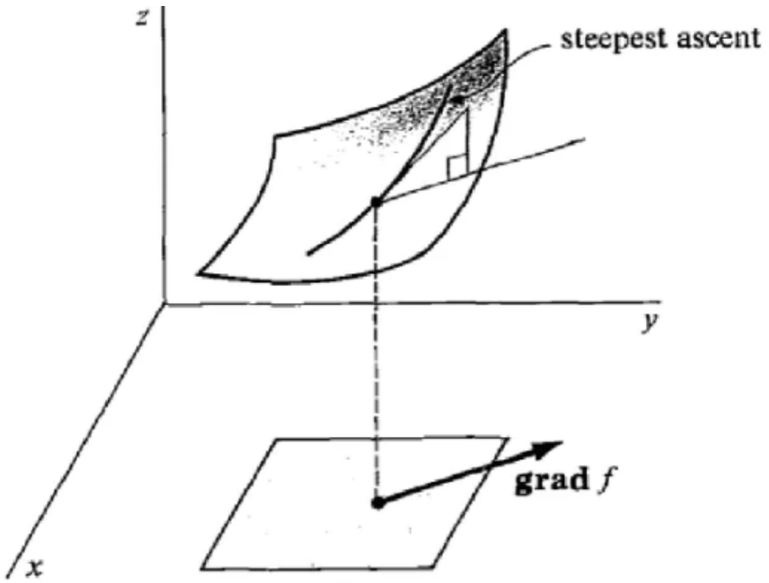
- The **gradient** of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the vector of its partial derivatives: $\nabla f(x_1, \ldots, x_n) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]$.

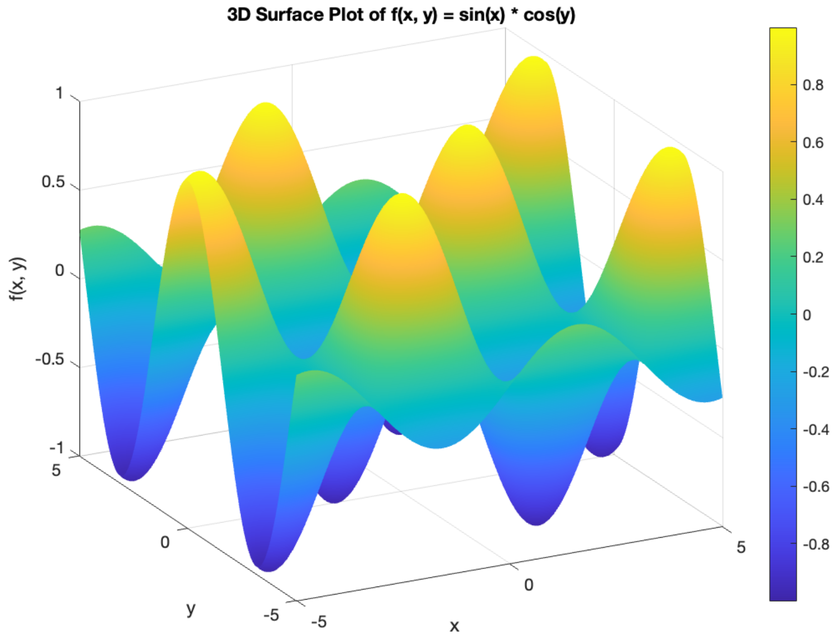
- If $n = 1$, the gradient is simply the derivative of f.

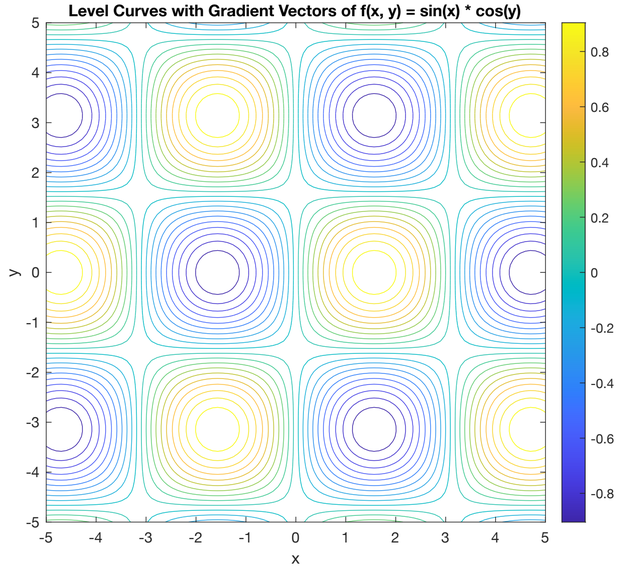
- The gradient generalises the concept of derivative in a multi-dimensional space.

- In each point of the domain, $(x_1, \ldots, x_n)$, it presents the **slope** of the surface $f(x_1, \ldots, x_n)$ in the **direction** of the **steepest ascent**.

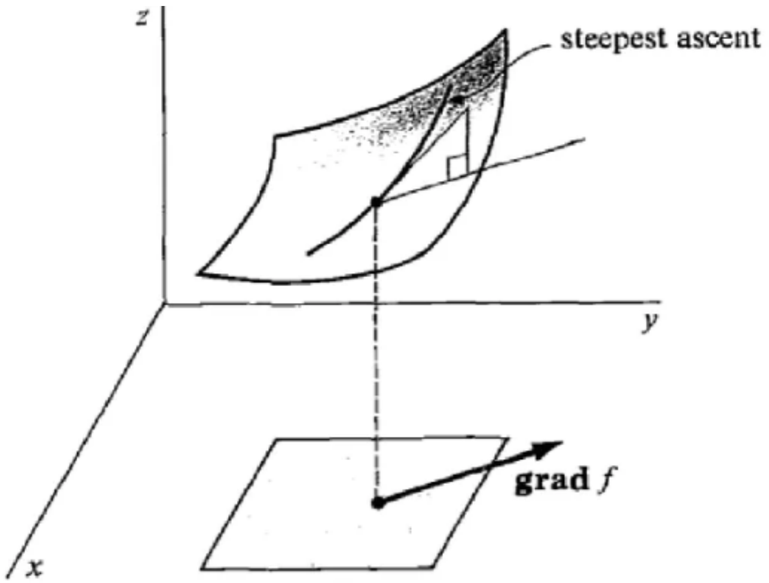
Gradient

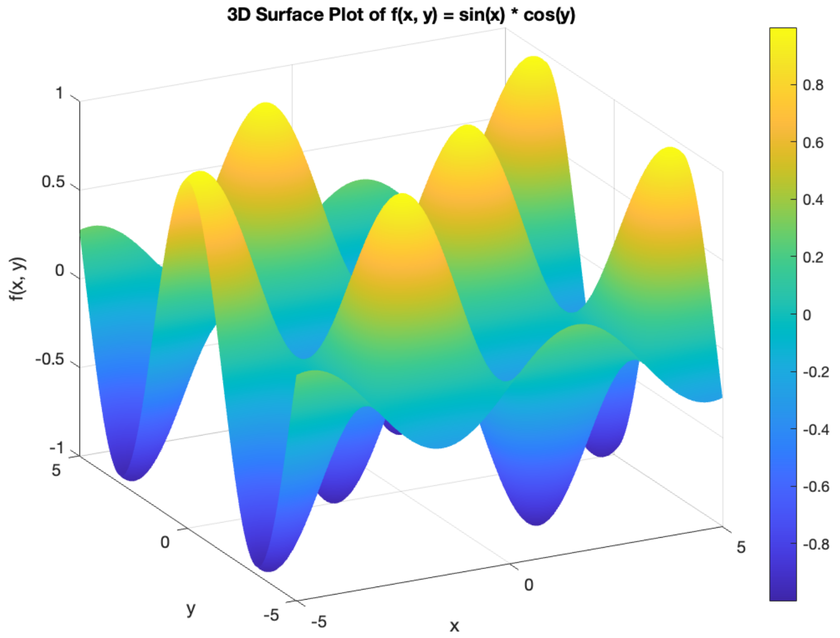

z
steepest ascent
y
grad f
x

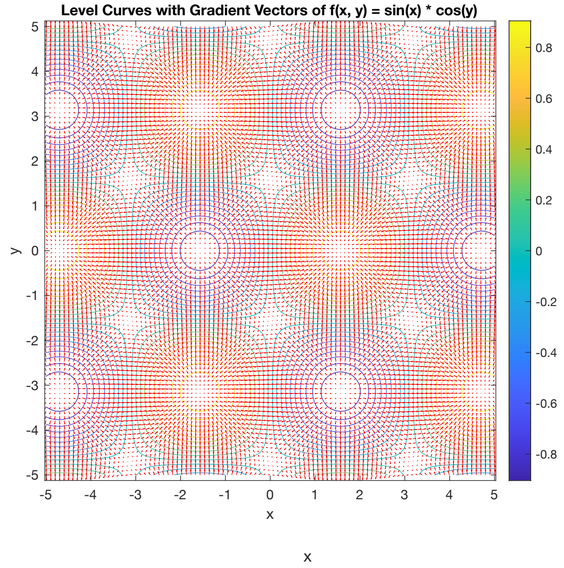

3D Surface Plot of f(x, y) = sin(x) * cos(y)
f(x, y)
1
0.5
0
-0.5
-1
5
0
y
-5
-5
0
5
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8


Level Curves with Gradient Vectors of f(x, y) = sin(x) * cos(y)
y
x
5
4
3
2
1
0
-1
-2
-3
-4
-5
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8

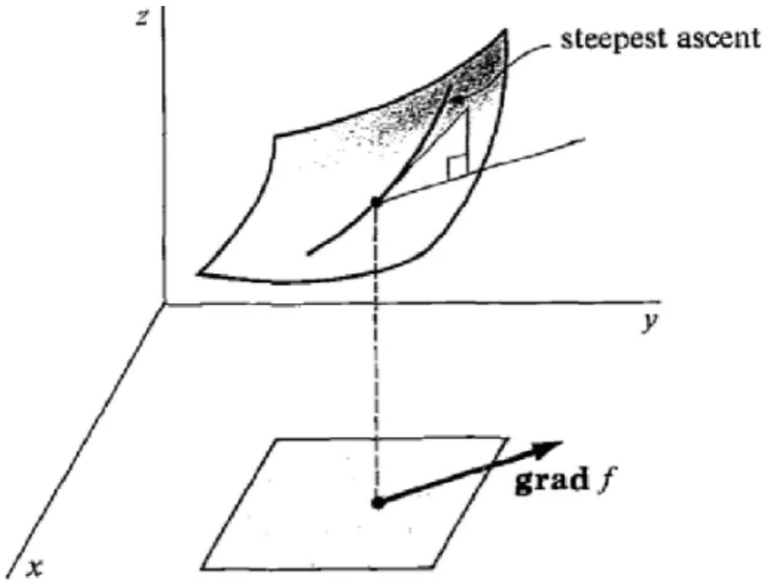
Gradient

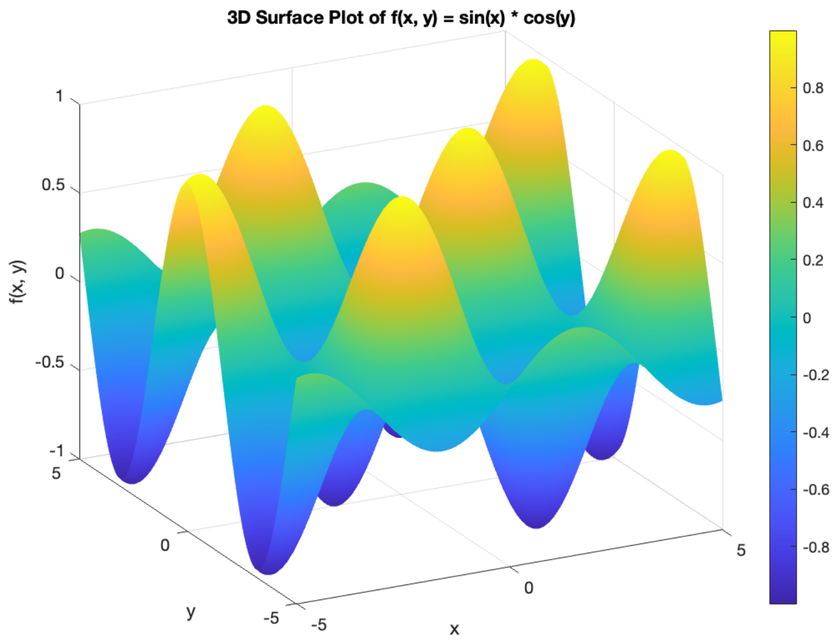

z
steepest ascent
y
grad f
x


3D Surface Plot of f(x, y) = sin(x) * cos(y)
f(x, y)
1
0.5
0
-0.5
-1
5
0
-5
y
-5
0
5
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8

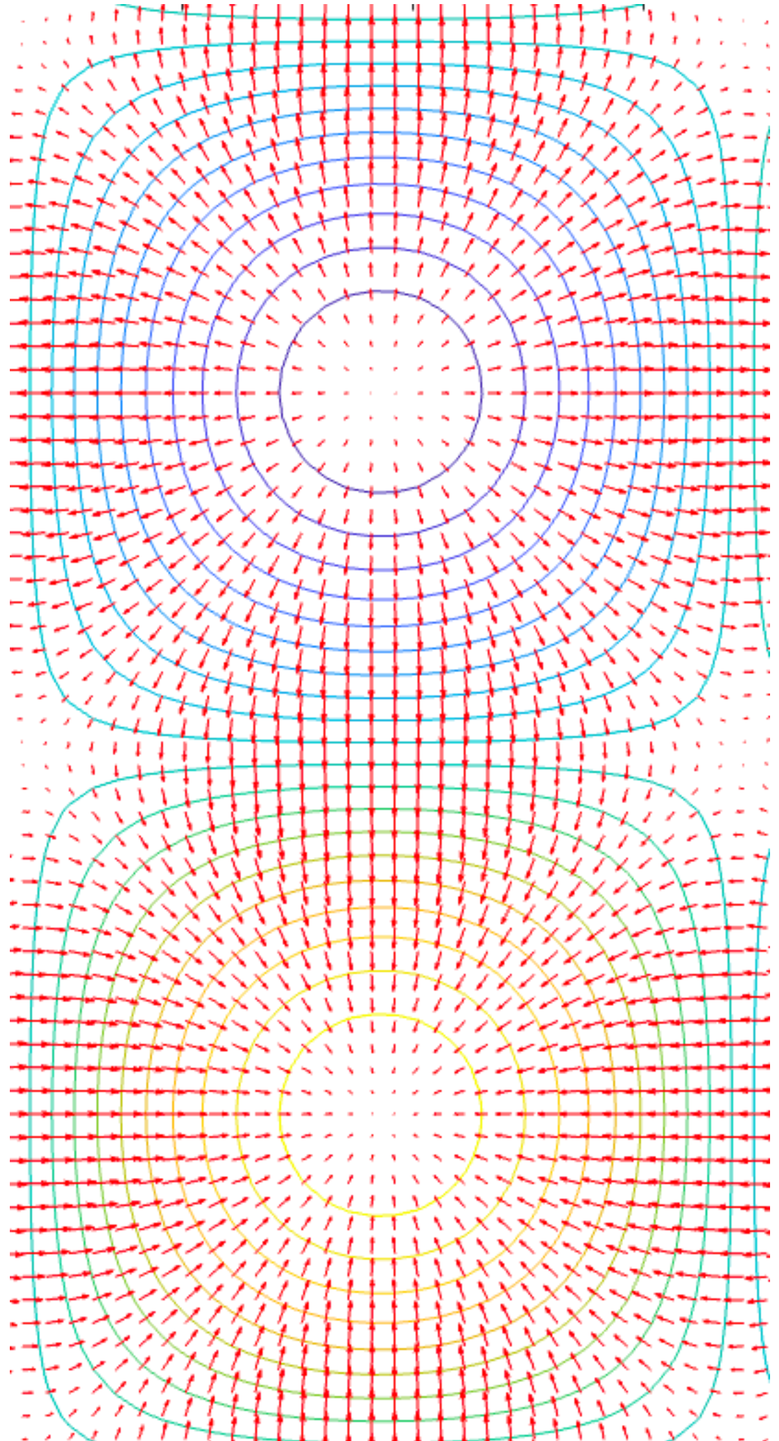

Level Curves with Gradient Vectors of f(x, y) = sin(x) * cos(y)
y
5
4
3
2
1
0
-1
-2
-3
-4
-5
-5 -4 -3 -2 -1 0 1 2 3 4 5
x
x
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8

Gradient


z
steepest ascent
y
grad f
x


3D Surface Plot of f(x, y) = sin(x) * cos(y)
f(x, y)
1
0.5
0
-0.5
-1
5
0
-5
y
-5
0
5
x

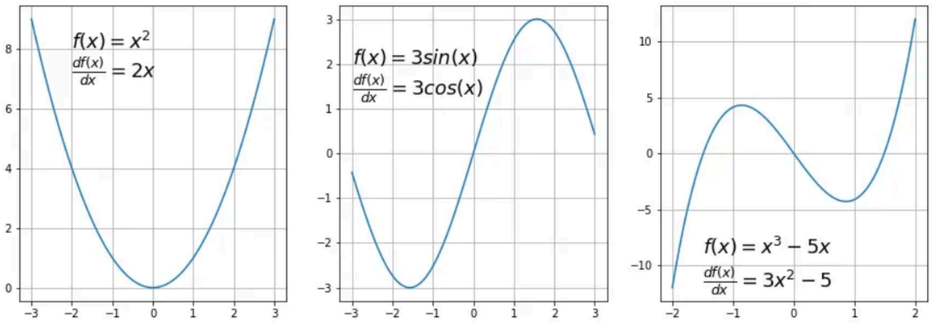
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8



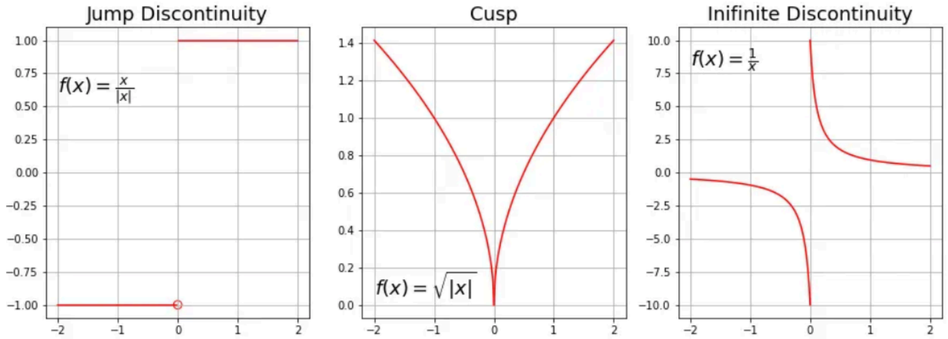
Gradient Descent

- The gradient descent is an iterative algorithm to find the minimum of a function f. It can be applied to f if:
  - f is differentiable.
    - If $f : D \subseteq \mathbb{R} \rightarrow \mathbb{R}$, then it is differentiable if its derivative is defined in each point of D.

Differentiable



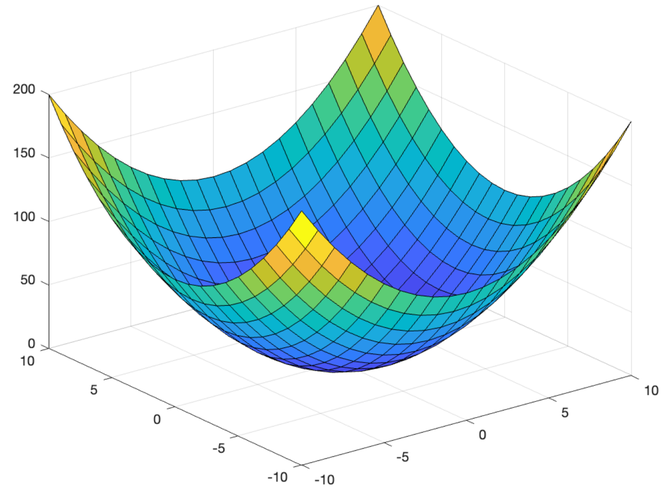
Non-differentiable



Gradient Descent

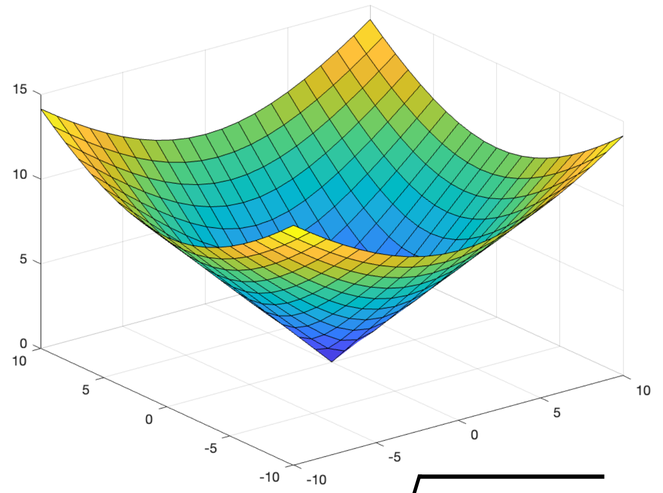
- If $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, saying whether it is differentiable or not is more complicated.
  - Sufficient condition: if for each point $a \in D$, all partial derivatives exist and are continuous in a neighbourhood of a, then the function is differentiable.

Differentiable



$$f(x_1, x_2) = x_1^2 + x_2^2$$

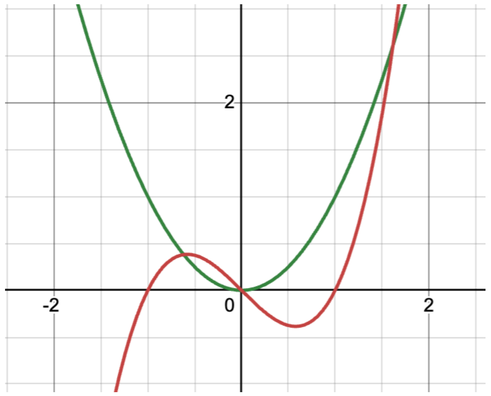
Non-differentiable



$$f(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$$

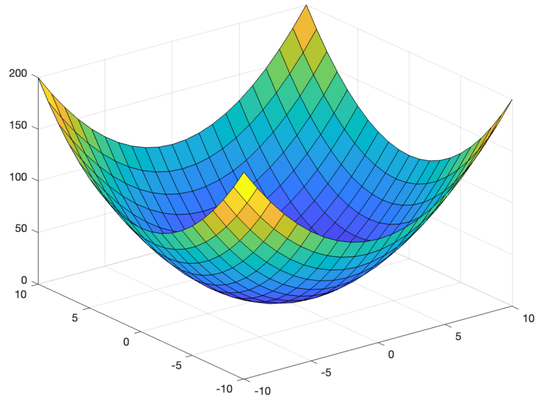
Gradient Descent

- The gradient descent is an iterative algorithm to find the minimum of a function f. It always finds the point of minimum if:
  - f is convex.



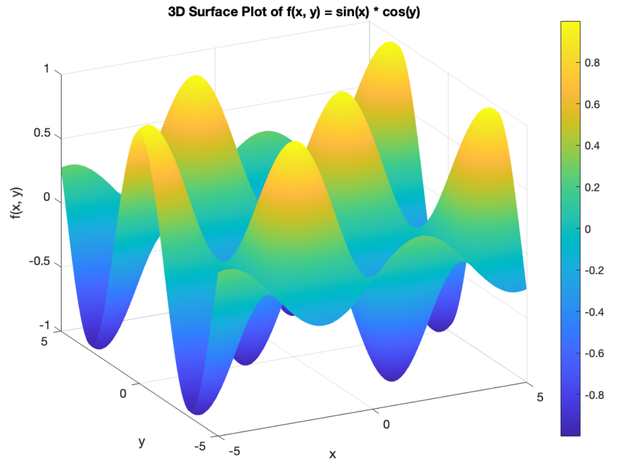
$f(x) = x^2$ Convex

$f(x) = x^3 - x$ Non convex



$f(x_1, x_2) = x_1^2 + x_2^2$

Convex



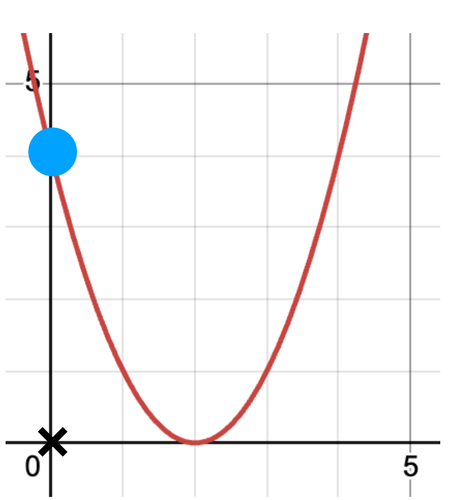
$f(x_1, x_2) = \sin(x_1) \cdot \cos(x_2)$

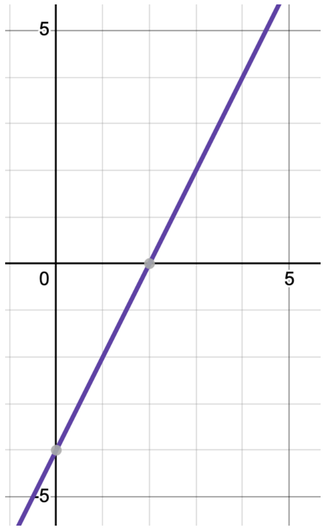
Non convex

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2 \quad f'(x) = 2(x-2)$



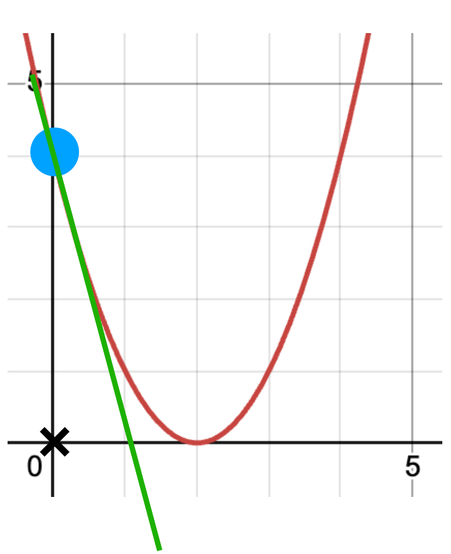


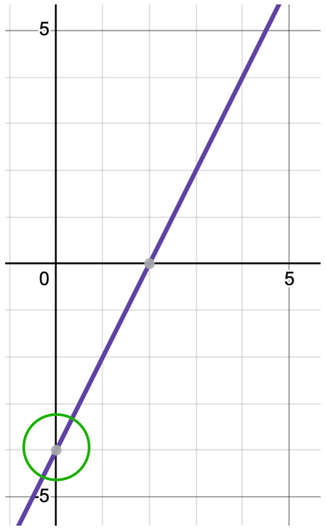
$x = 0$

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2$ $f'(x) = 2(x-2)$





$x = 0$ $f'(0) = -4$

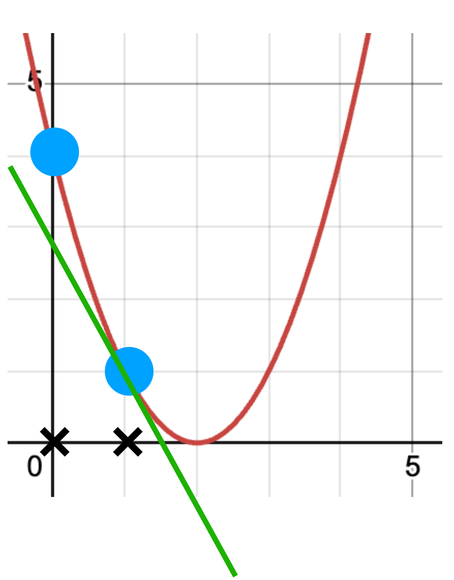
Line tangent to f in 0 has slope -4.

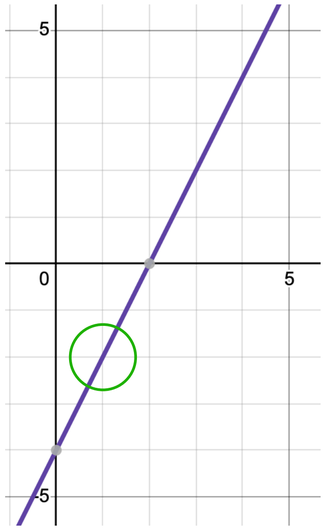
Means that f grows when "moving left". Since we want to move towards where the function decreases, we move right

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2 \quad f'(x) = 2(x-2)$





$x = 1$ $\qquad f'(1) = -2$

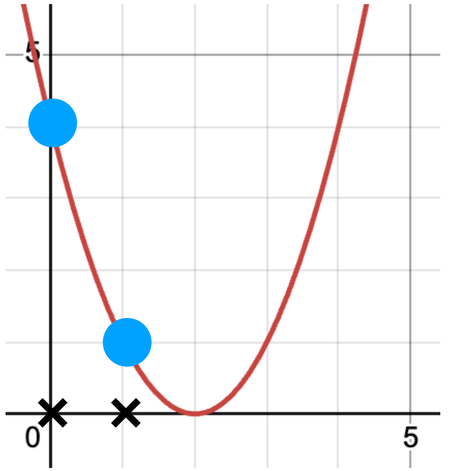
Line tangent to f in 1 has slope -2.

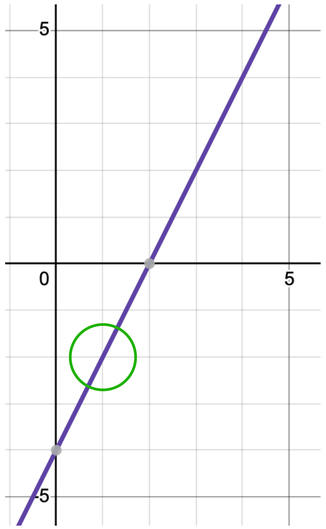
Means that f grows when "moving left". Since we want to move towards where the function decreases, we move right

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2 \quad f'(x) = 2(x-2)$



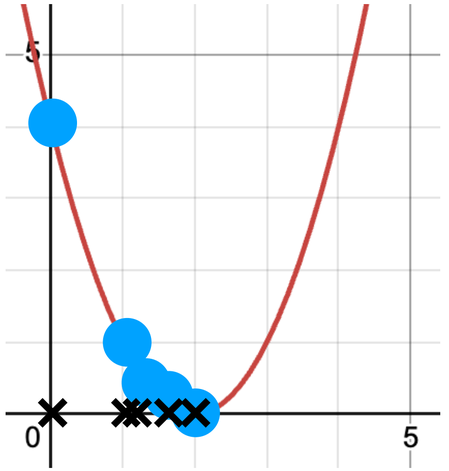


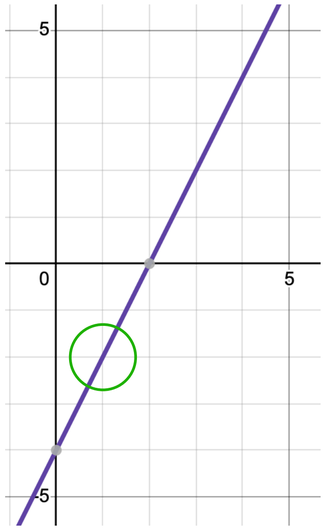
Continue until convergence

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2 \quad f'(x) = 2(x-2)$



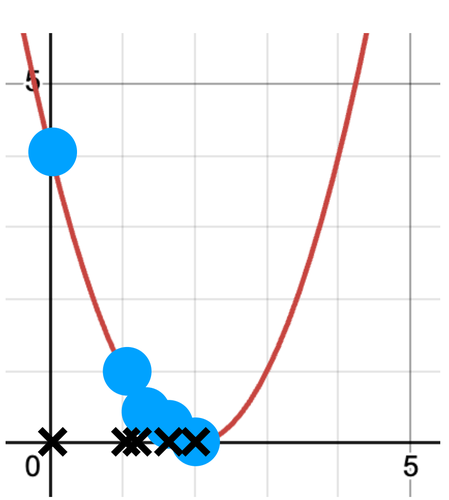


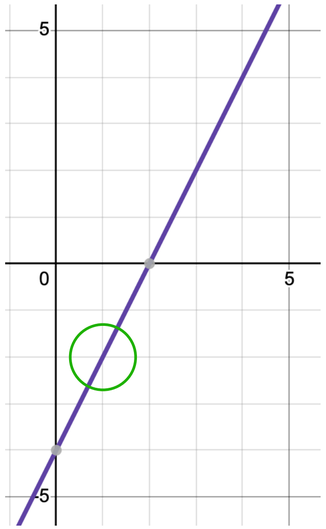
Continue until convergence

Gradient Descent

- Idea: given a function, choose an initial point.
- Compute the gradient of the function, and see what is its value at the initial point.
- Move of one step towards the opposite direction. You will land in another point.
- Repeat for a maximum number iterations, or until the improvement is smaller than the tolerance.

$f(x) = (x-2)^2$ $f'(x) = 2(x-2)$





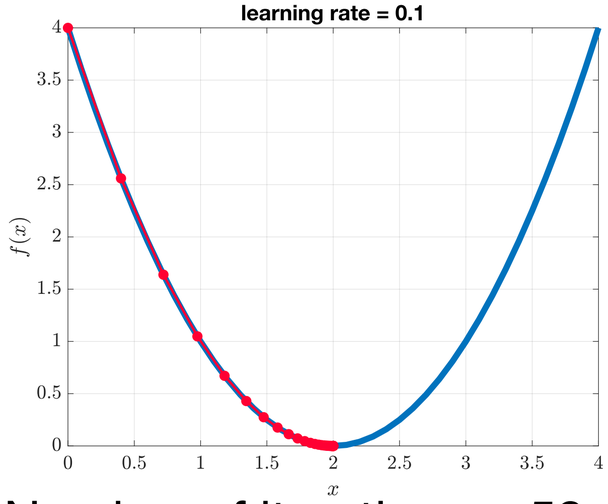
The step is known as “learning rate”

Large learning rate: move fast towards the opposite direction of the gradient
Small learning rate: move slowly towards the opposite direction of the gradient

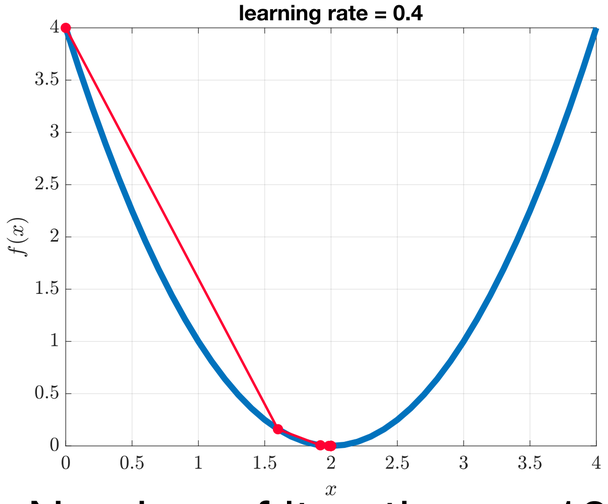
Gradient Descent

GradientDescent$(f(x), x_0, \eta, I_{max}, \epsilon)$

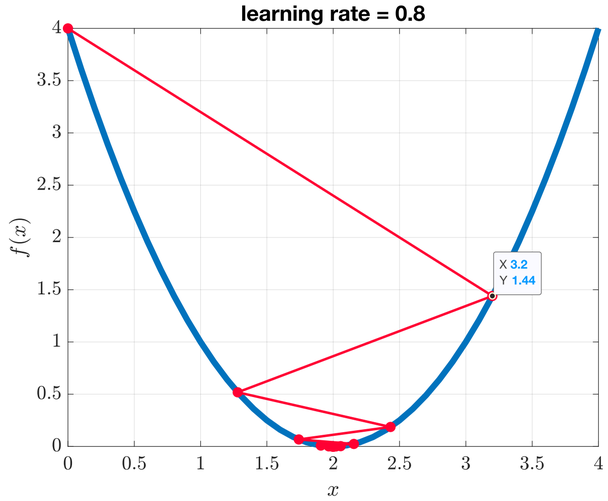
- Compute $\nabla f(x)$
- $x \leftarrow x_0$ (random)
- Initialise d
- For $i = 1, \ldots, I_{max}$
  - $d' \leftarrow \nabla f(x) \cdot \eta$
  - If $|d - d'| < \epsilon$
    - return x
  - $d \leftarrow d'$
  - $x \leftarrow x - d$



Number of iterations = 59



Number of iterations = 10



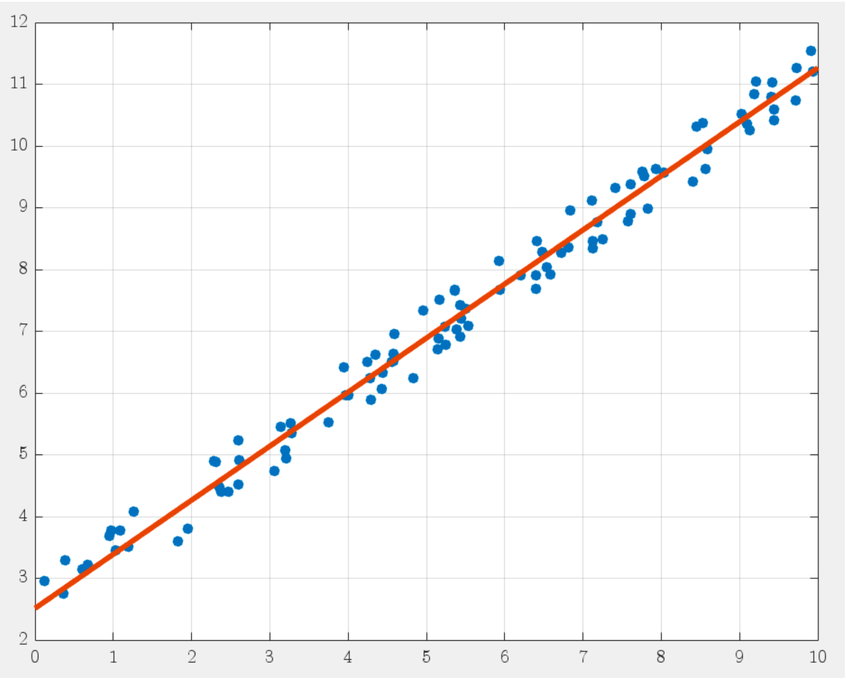
Number of iterations = 31

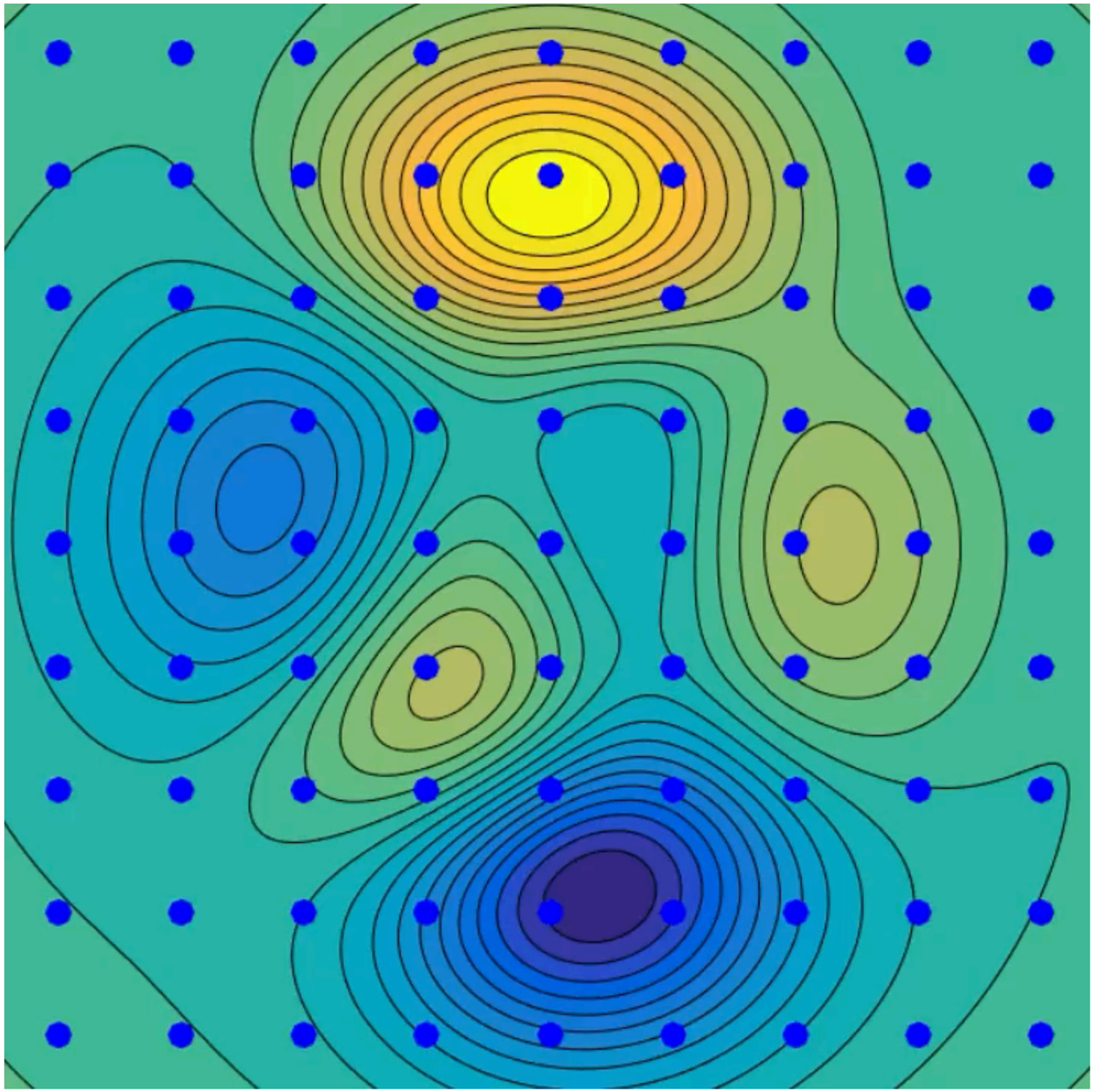
Number of iterations = I_{max}

Gradient Descent - example with MSE

GradientDescent$(f(\theta_0, \theta_1) = \frac{1}{n}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y_i)^2, \theta_{in}, \eta, I_{max}, \epsilon)$

- Compute $\nabla f(x) = \left[\frac{1}{n}\sum_{i=1}^{n} 2(\theta_0 + \theta_1 x_i - y_i), \frac{1}{n}\sum_{i=1}^{n} 2x_i(\theta_0 + \theta_1 x_i - y_i)\right]$
- $\theta \leftarrow \theta_{in}$
- Initialise d
- For $i = 1, \ldots, I_{max}$
  - $d' \leftarrow \nabla f(x) \cdot \eta$
  - If $||d - d'|| < \epsilon$
    - return θ
  - $d \leftarrow d'$
  - $\theta \leftarrow \theta - d$



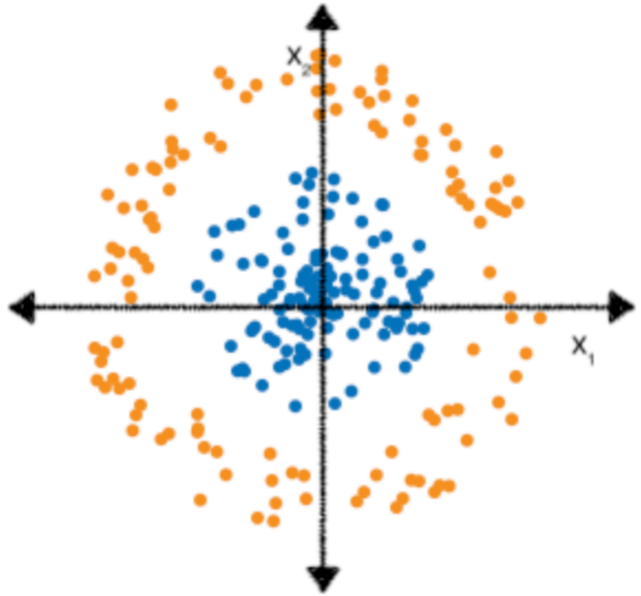


9.3 Introduction to Neural Networks (MLP and back propagation)

- Very often, data is represented with many features and exhibit complex non-linear patterns which are hard to detect with regression.

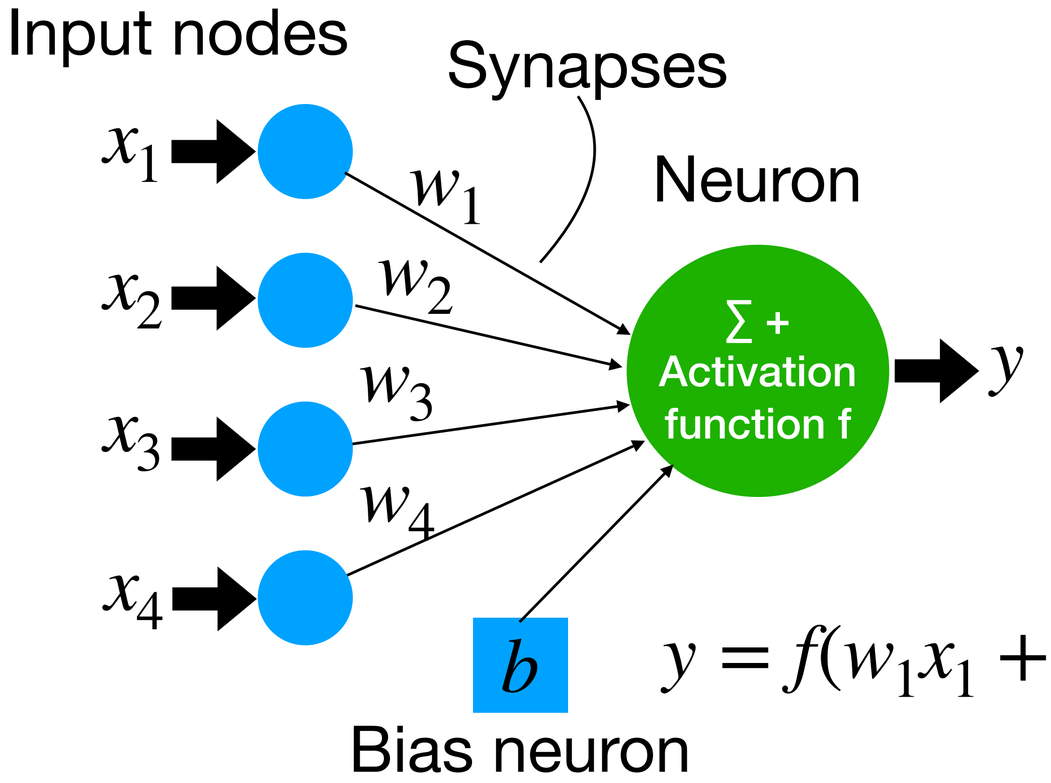
- Neural networks are a family of model architectures designed to find nonlinear patterns in data.

- Their structure and components are inspired by the biological structure of the humans' neuron.



Perceptron

- The perception is the building block of a neural network. The number of input nodes is given by the number of features of the data points.

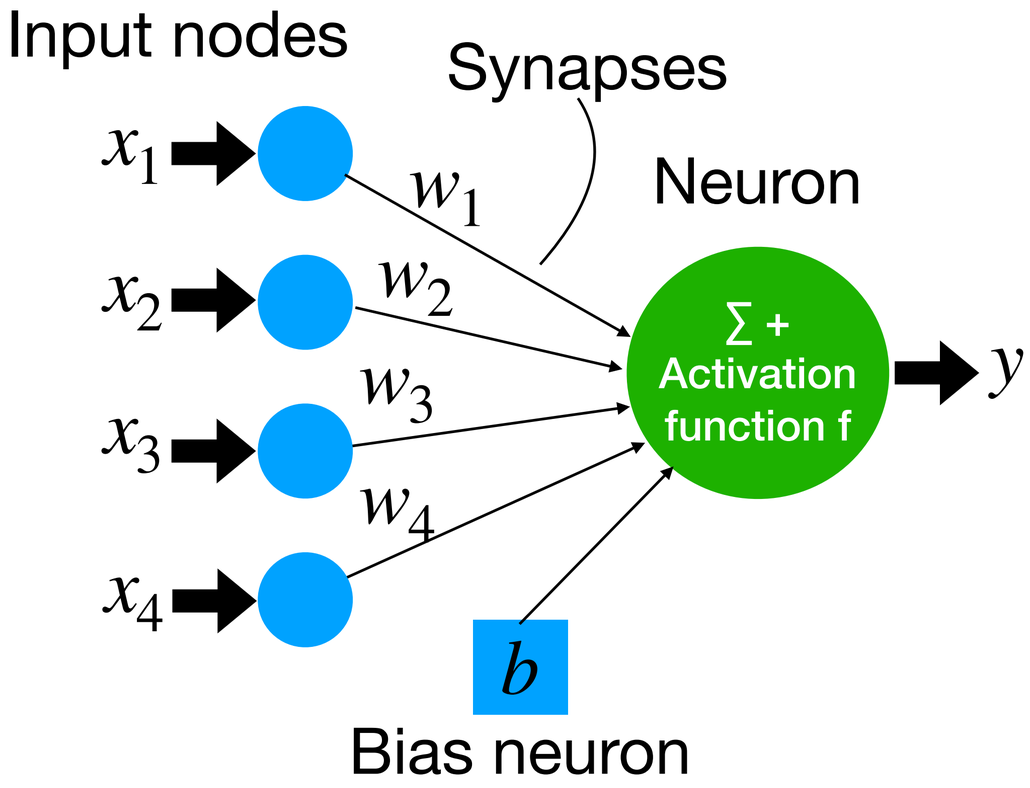


- If the activation function f is the identity function, this is a simple linear regression model, where b is the intercept.

$$y = f(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b)$$

Perceptron

- The perception is the building block of a neural network.



- Given an input data point with d features $x_1, \ldots, x_d$, the neuron:
- Computes a linear combination of the input and the synapses weights, possibly adding the bias:

  $b + \sum_{i=1}^{d} w_i x_i$, or, in matrix form: $b + \mathbf{w}^T \cdot \mathbf{x}$

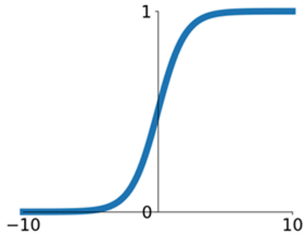
  where $\mathbf{w} = [w_1, \ldots, w_d]^T$ and $\mathbf{x} = [x_1, \ldots, x_d]^T$
- Applies the **activation function** f to the combination.

Activation functions

- Activation functions enables a perceptron to learn nonlinear and complex relationships between features and label.

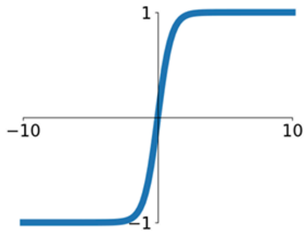
Sigmoid

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



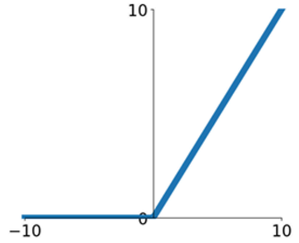
tanh

$$\tanh(x)$$



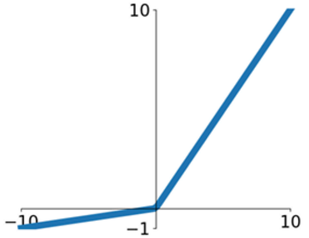
ReLU

$$\max(0, x)$$



Leaky ReLU

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



Maxout

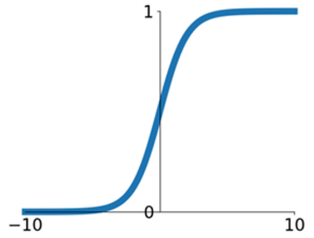
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Activation functions

- Activation functions enables a perceptron to learn nonlinear and complex relationships between features and label.

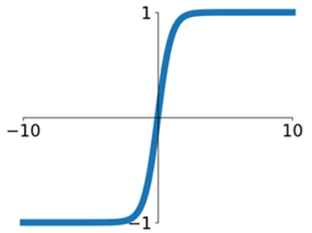
Sigmoid

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



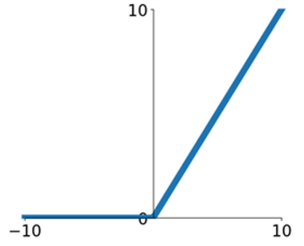
tanh

$$\tanh(x)$$



ReLU

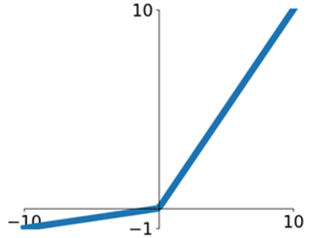
$$\max(0, x)$$



Notice that this is not differentiable

Leaky ReLU

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

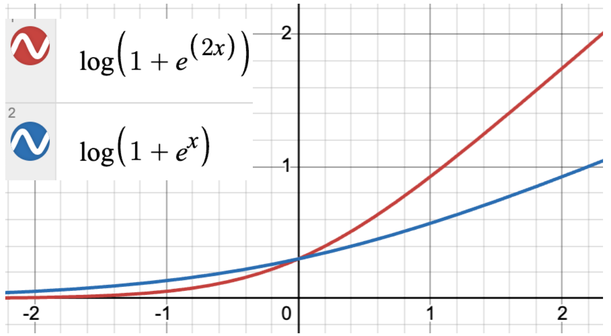


Maxout

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

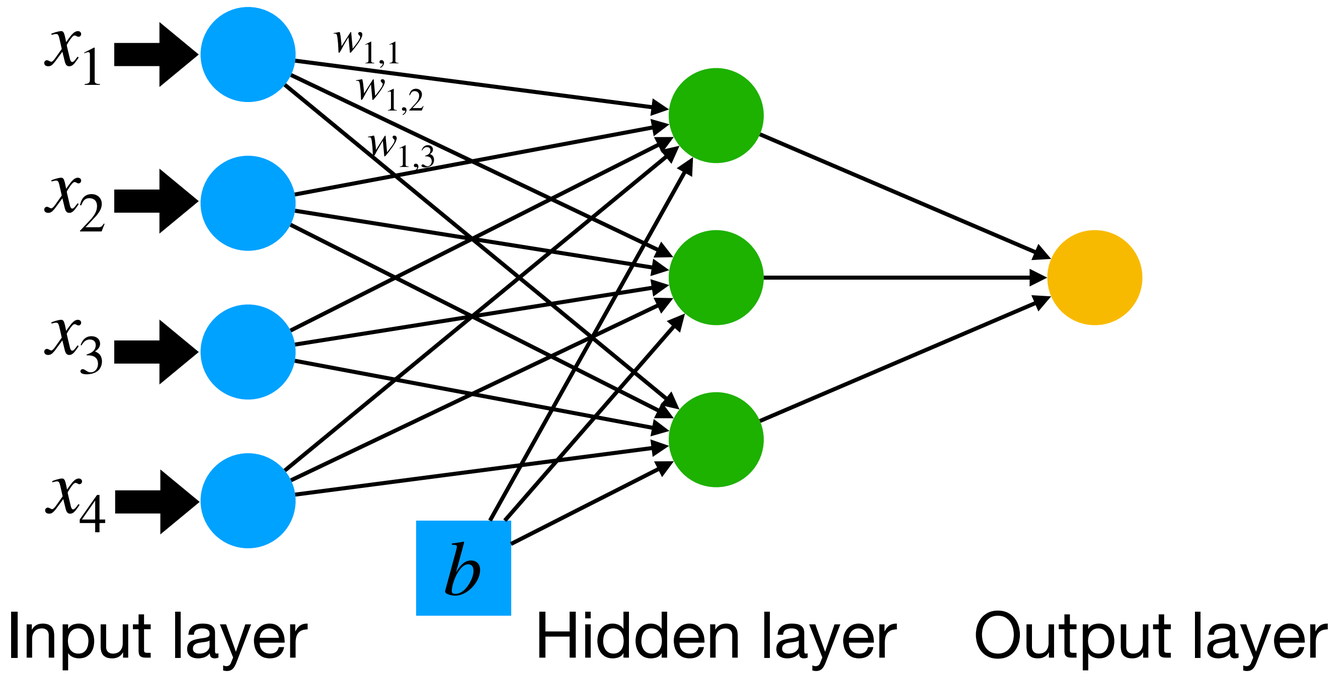
Softplus

$$\log(1 + e^x)$$



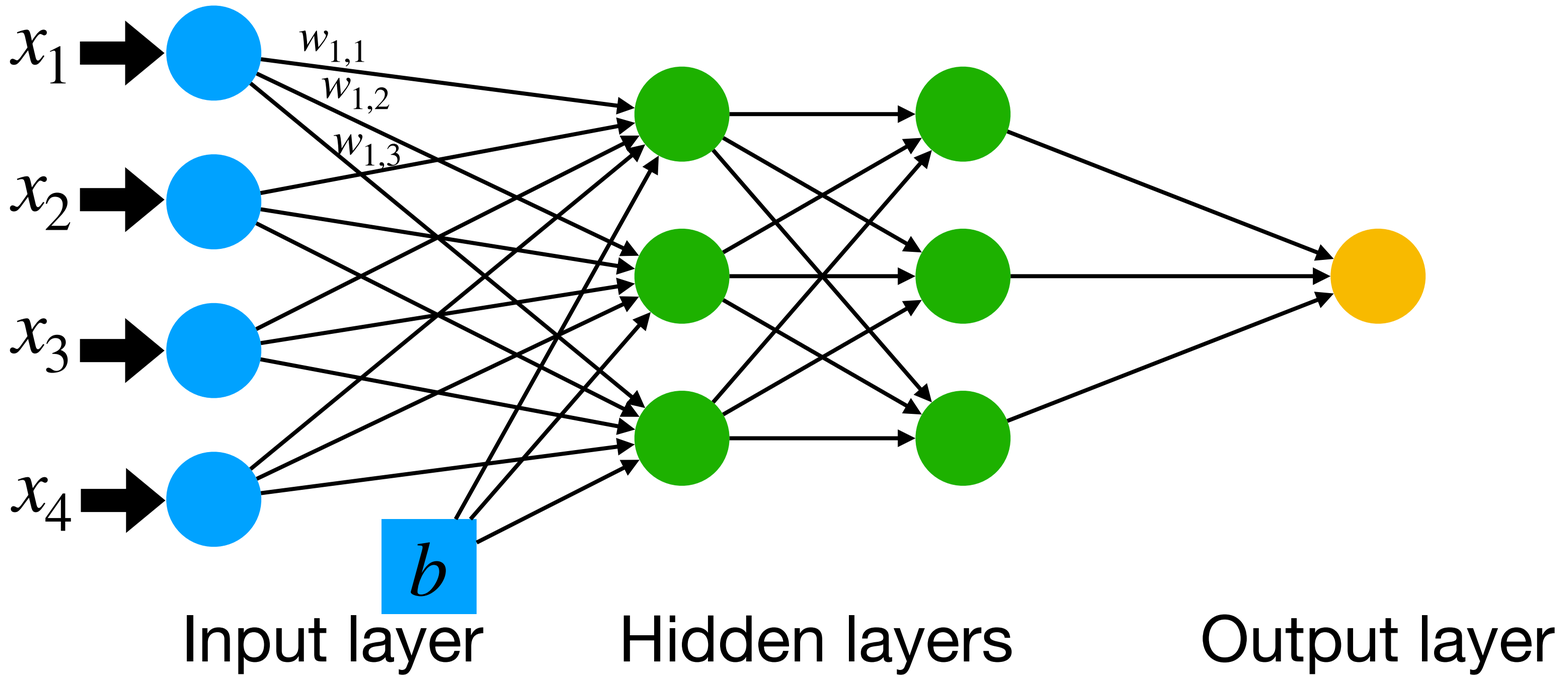
Single layer Perceptron

- Neurons and Synopsis are organised in layers: the input layer, one or more **hidden layers**, and the output layer.

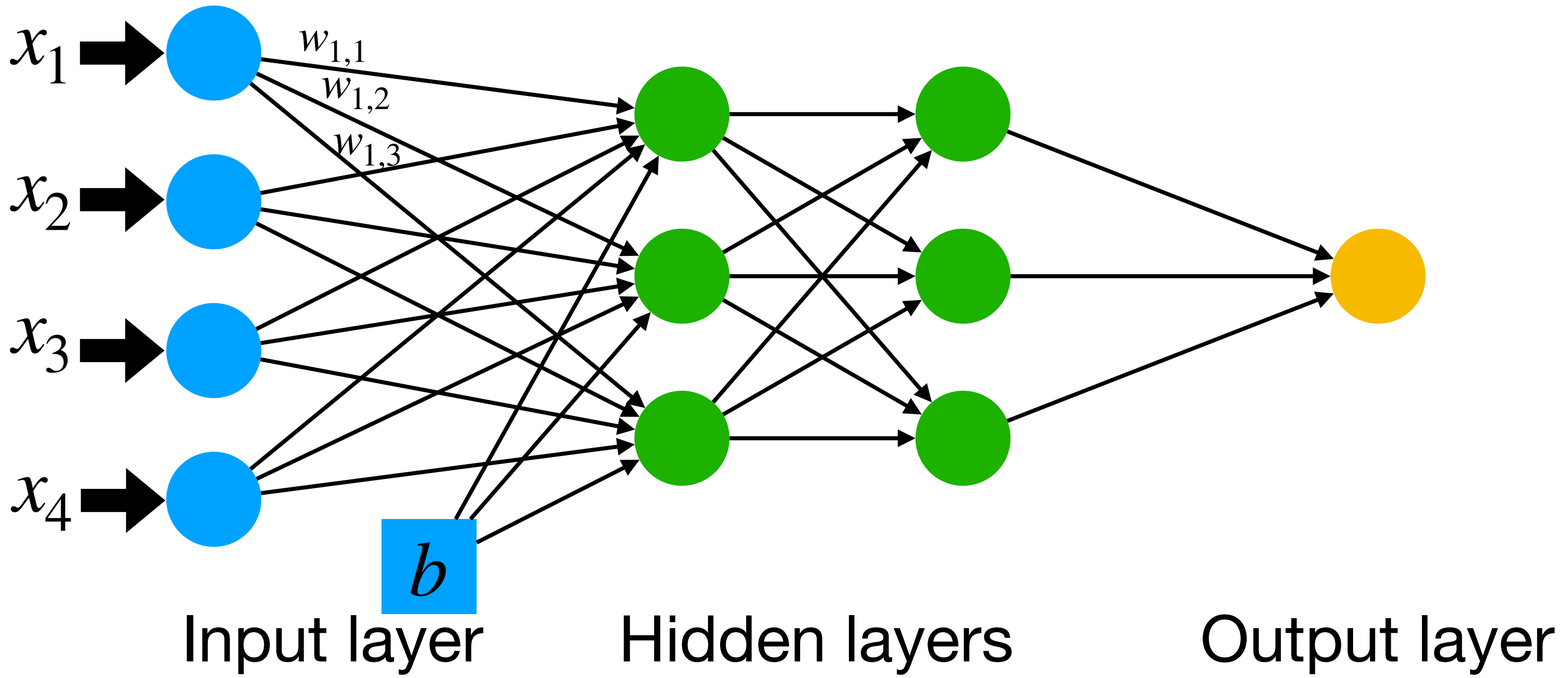


Multi layer Perceptron (MLP)

- Neurons and Synopsis are organised in layers: the input layer, one or more hidden layers, and the output layer.



- **Deep learning**: is a subfield of machine learning based on multilayered neural networks (at least one hidden layer) to perform regression and classification
- Multilayer perceptron is an example of **feed forward neural network**, i.e. the directed links in the network do not form cycles.
- This is in contrast to **recurrent neural network**, where the output of a node at hidden layer j can be fed to a node in a hidden layer < j.



- Weights and biases are the **model parameters,** and learnt during the **training** process.

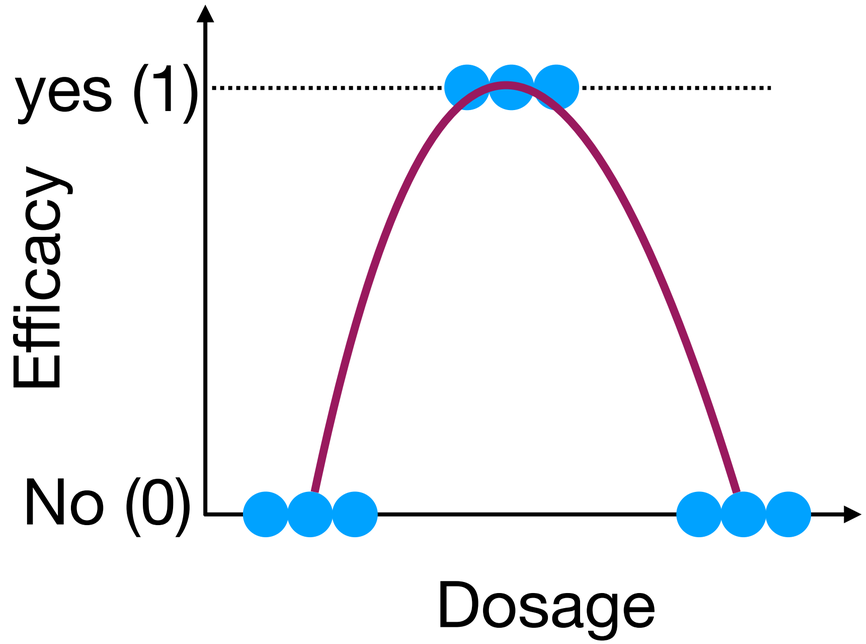
- The structure of the neural network, the number of layers, the activation functions, the number of nodes per hidden layers, are **hyperparameters**, which are defined before training and influence the performance of a neural network.

  - They are not learnt during the training process

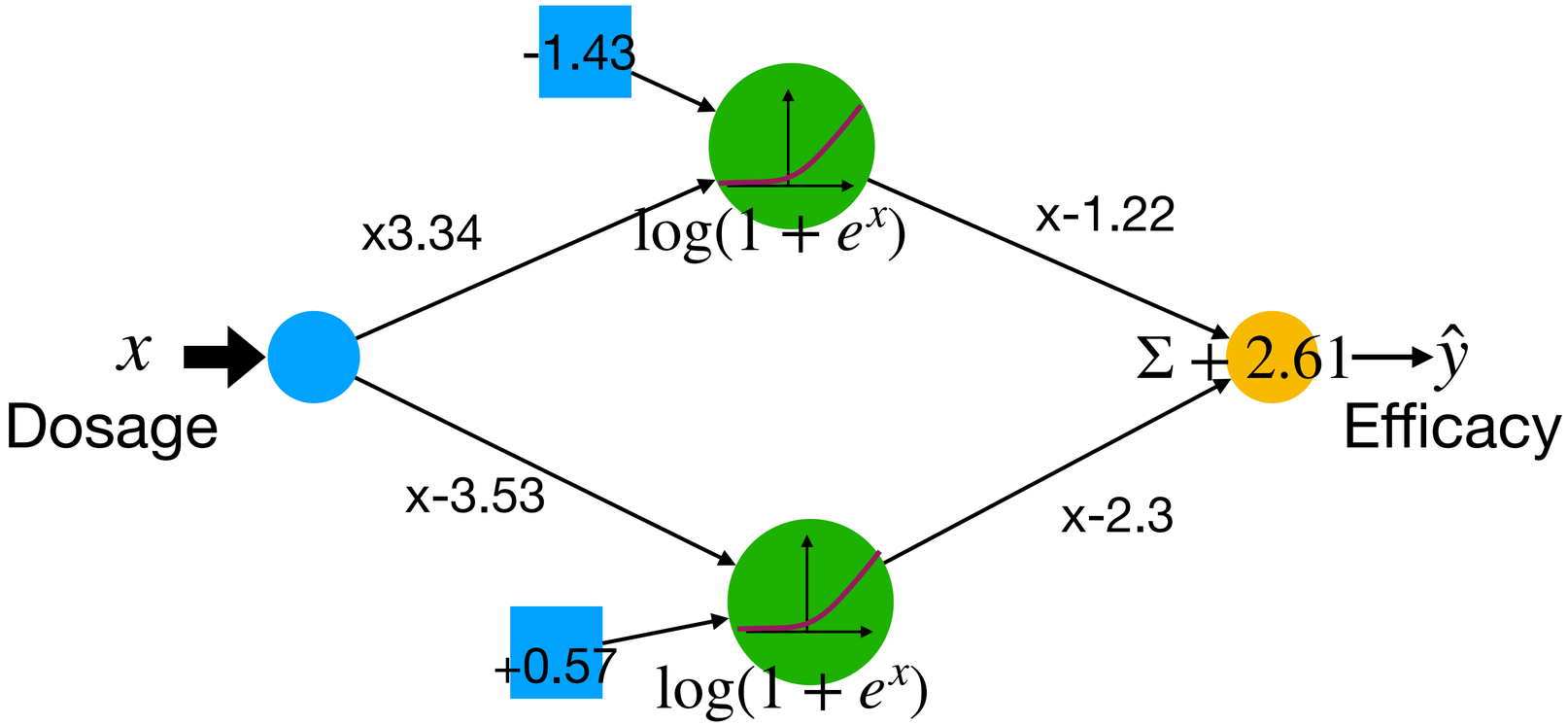
Example

- Assume we are testing a drug to treat an illness. We gave the drug to three different groups of people with three different dosages: low, medium and high.

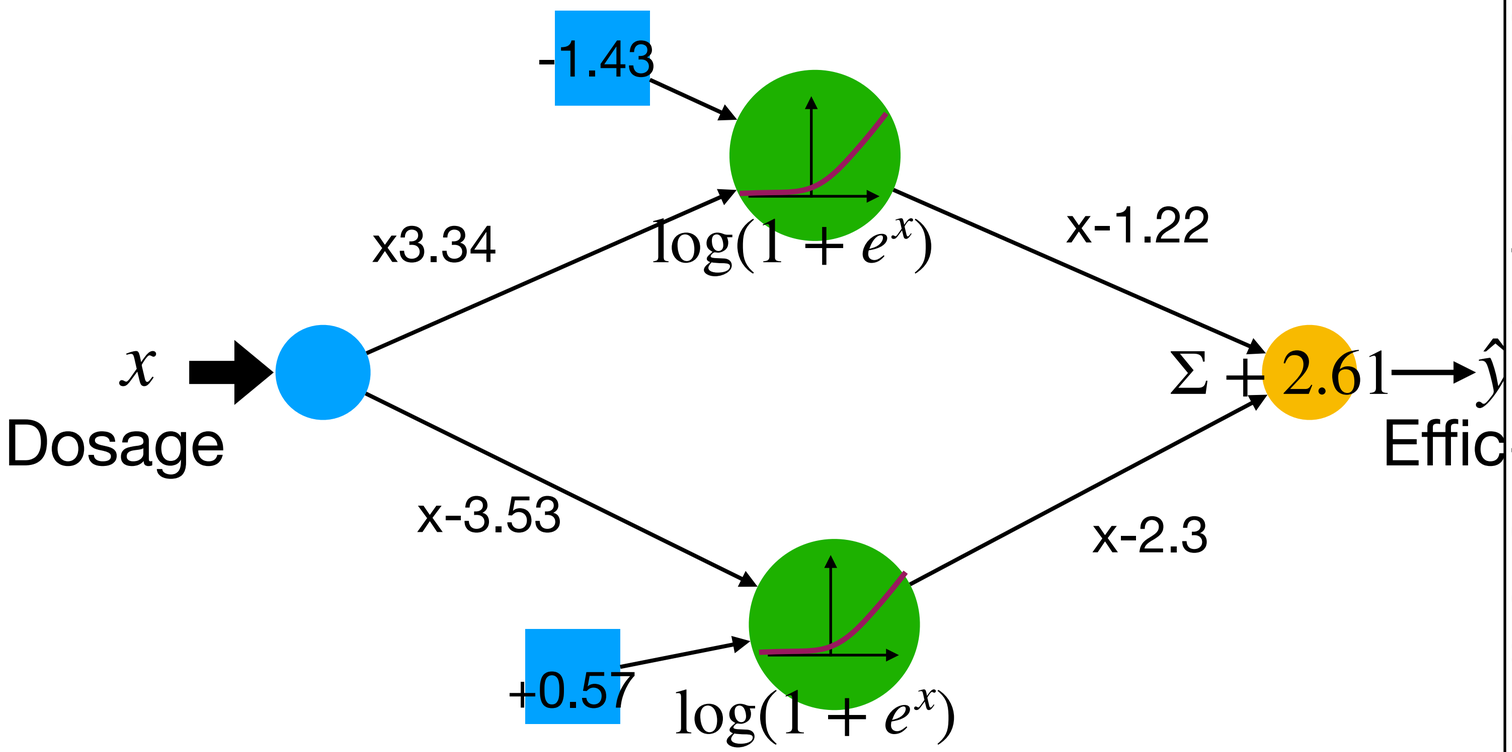
- Low and high dosages were not effective, while medium dosages were effective.



Example


-1.43
x3.34
$\log(1+e^x)$
x-1.22
x
Dosage
$\Sigma + 2.61 \rightarrow \hat{y}$
Efficacy
x-3.53
x-2.3
+0.57
$\log(1+e^x)$

Example



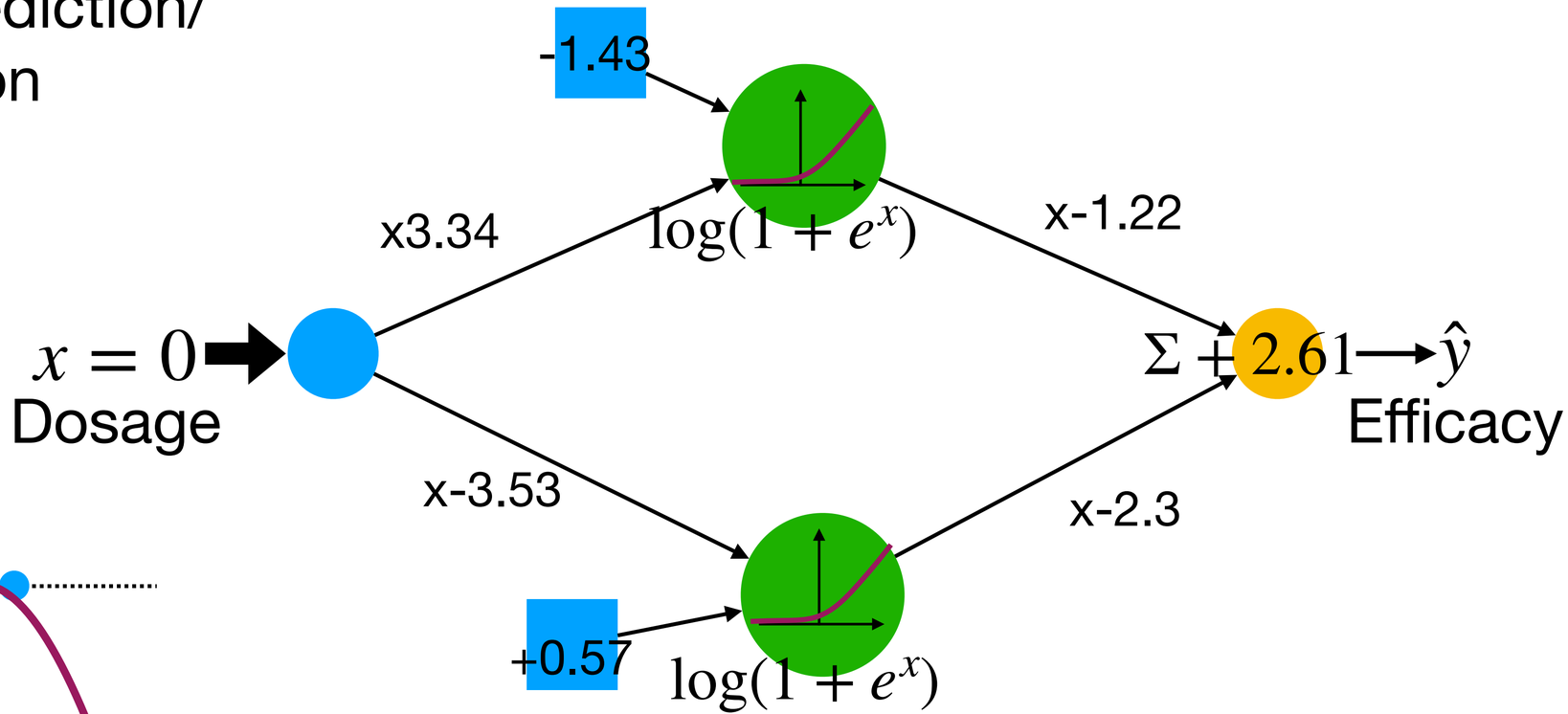
Notice that the parameters (weights and bias) of this neural network are already defined.

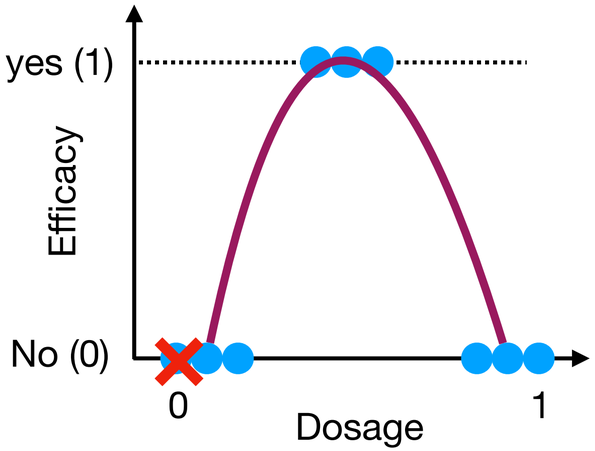
The neural network was already trained.

The parameter estimates are analogous to the slope and intercept estimates that we saw in linear regression.

Example

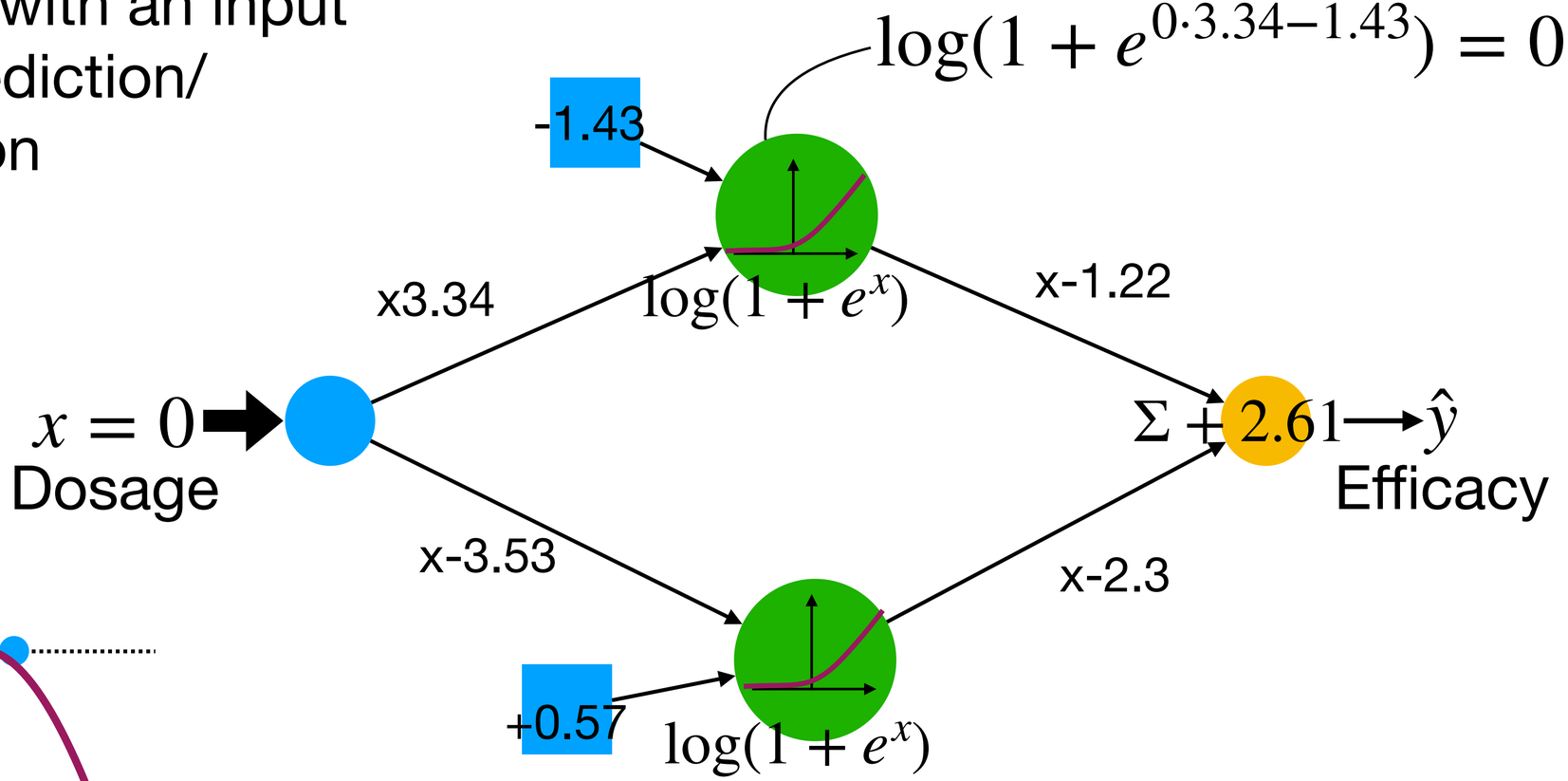
Inference: given a trained NN, feed it with an input to get a prediction/ classification

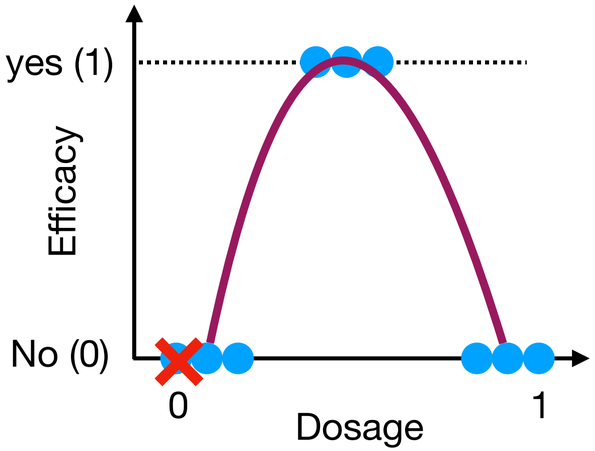




Example

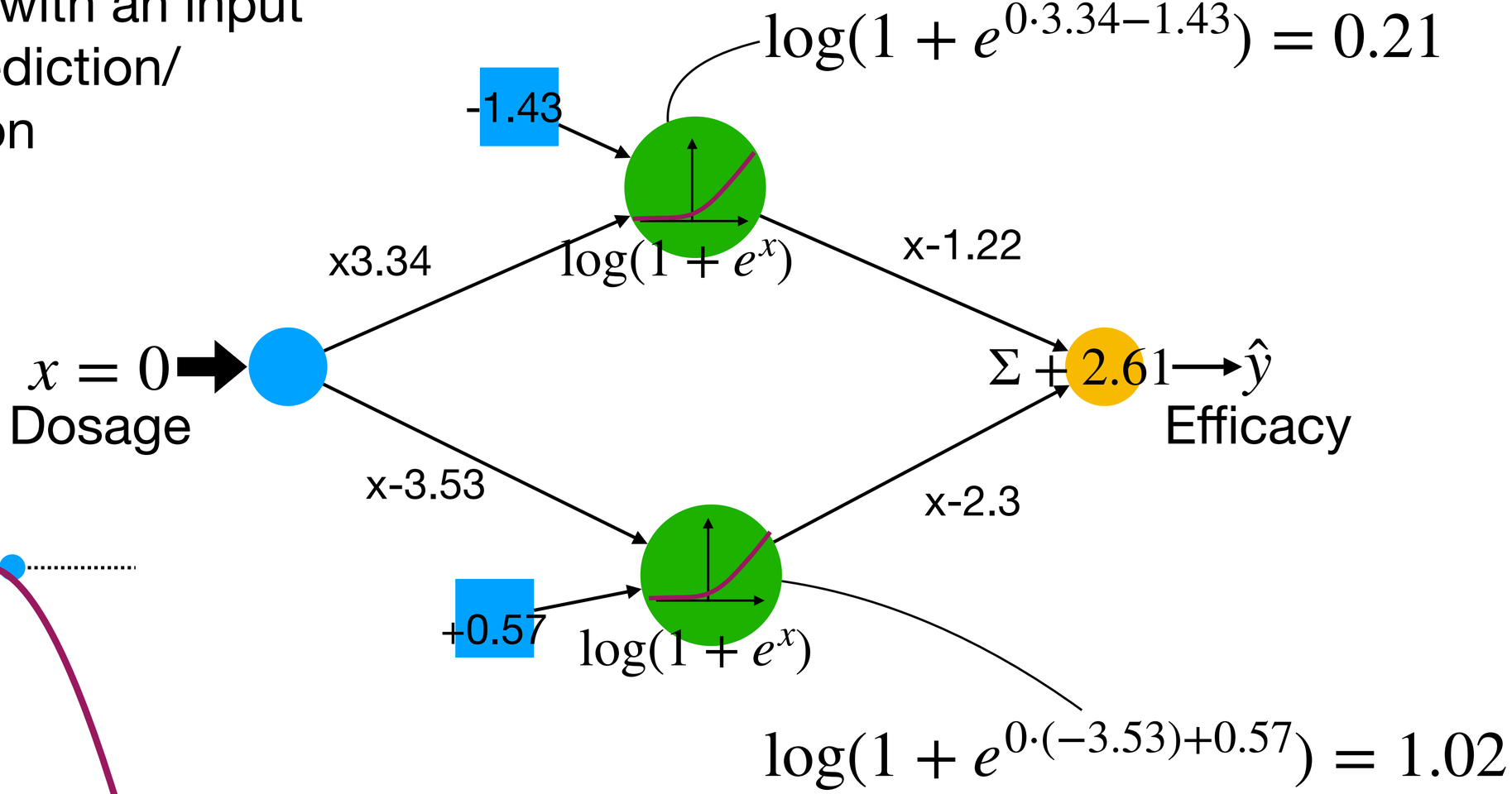
Inference: given a trained NN, feed it with an input to get a prediction/ classification

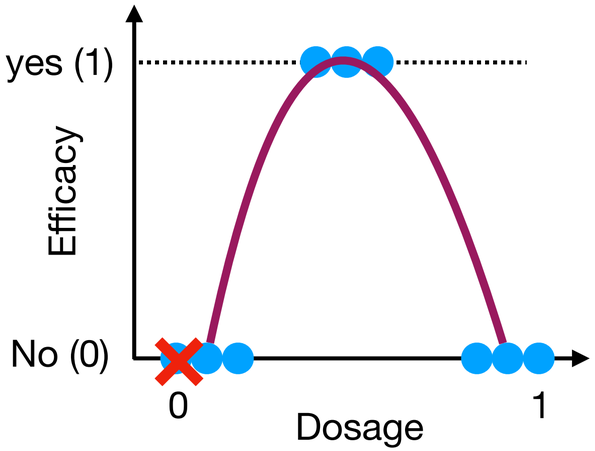




Example

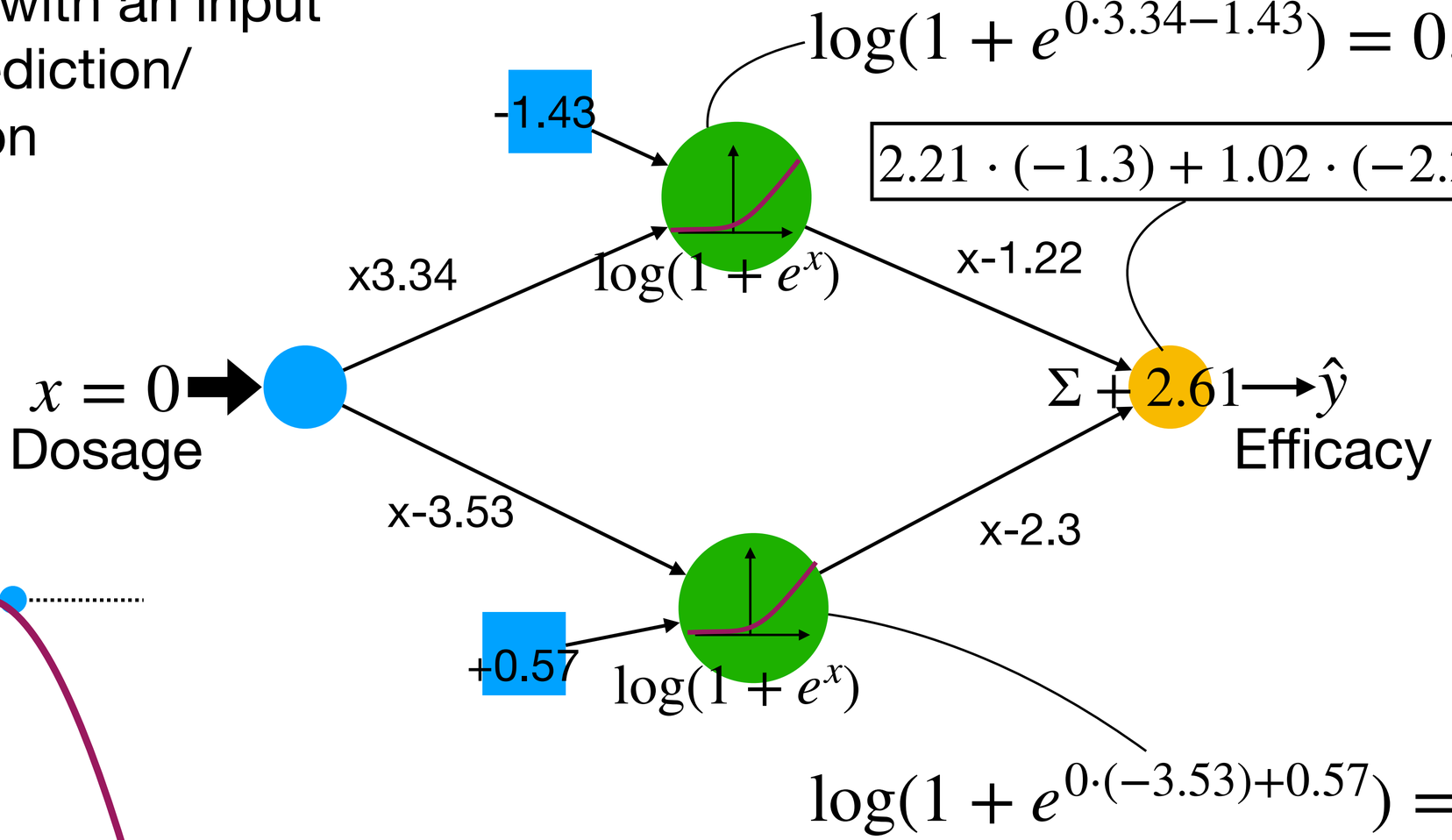
Inference: given a trained NN, feed it with an input to get a prediction/ classification



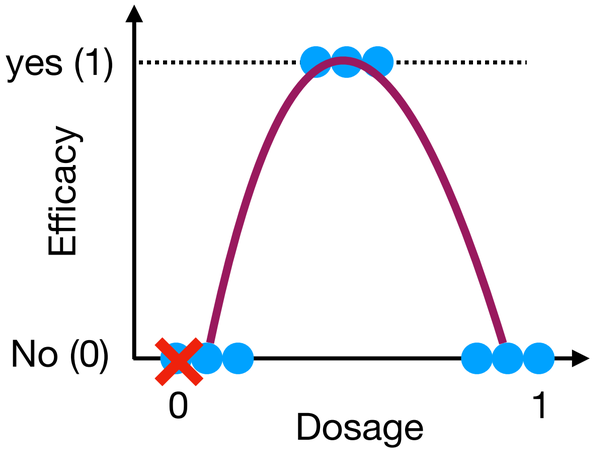


Example

Inference: given a trained NN, feed it with an input to get a prediction/ classification

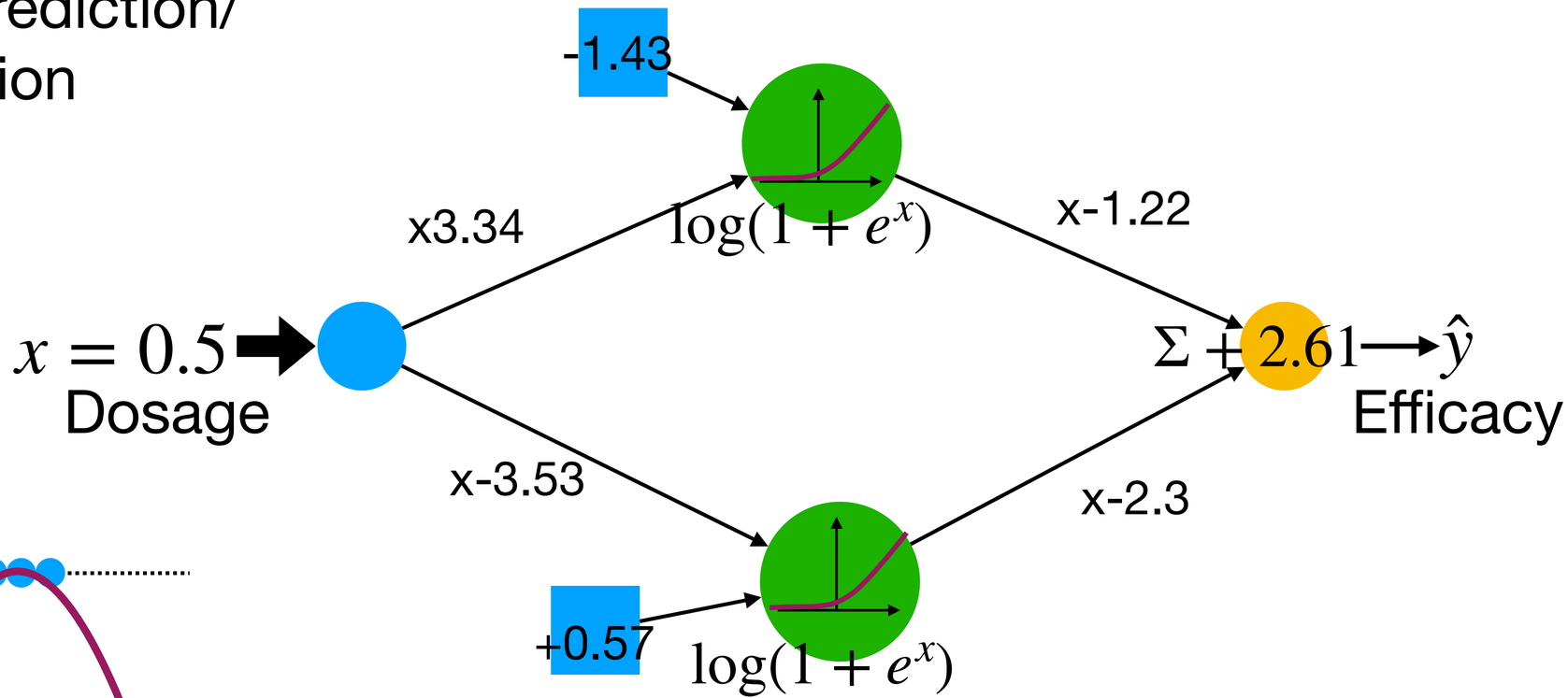


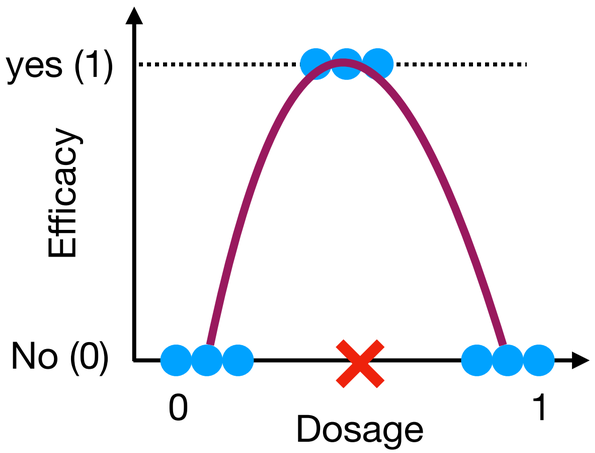
Since the output is close to 0, we infer that the dosage is not effective.



Example

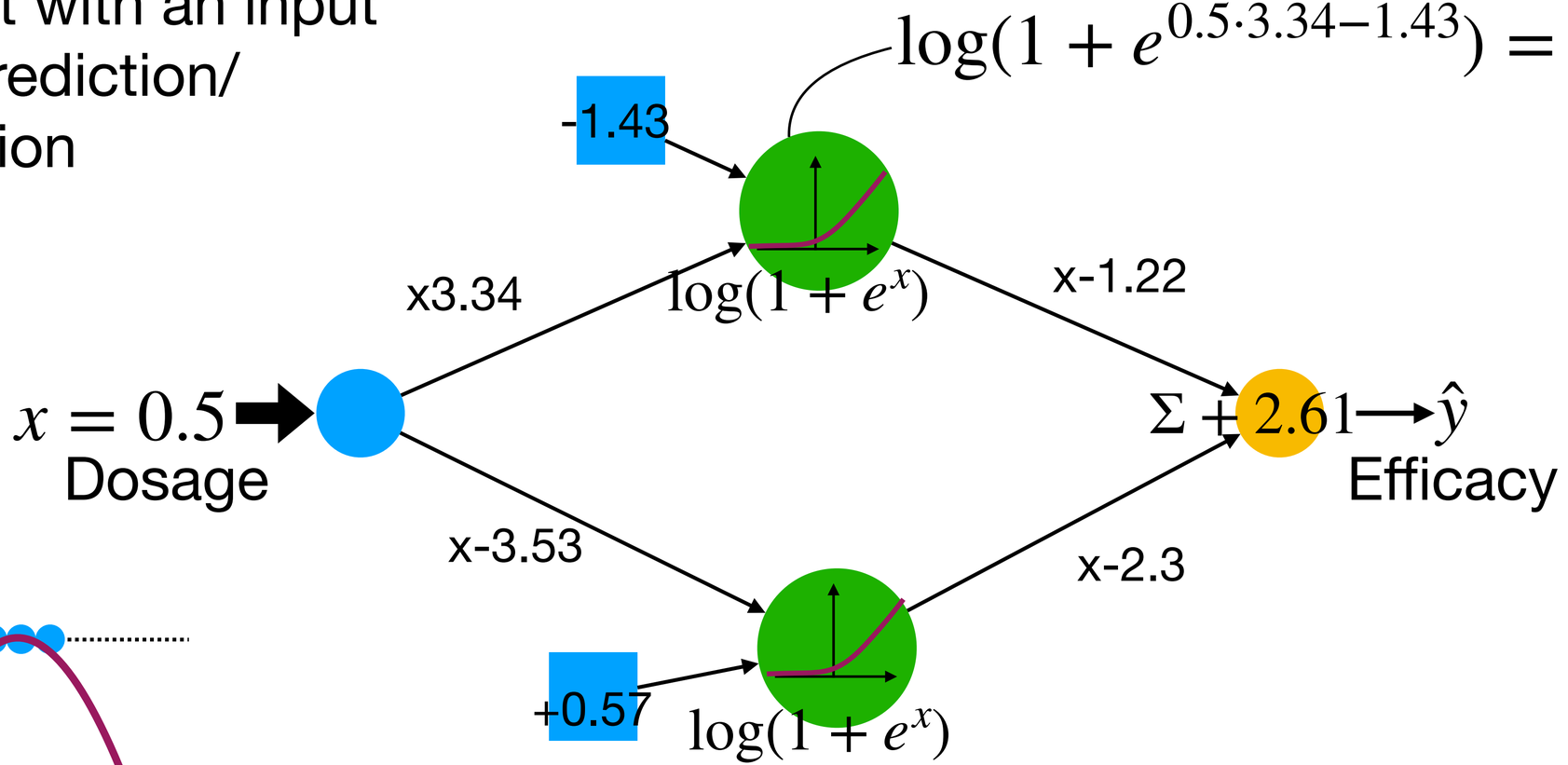
Inference: given a trained NN, feed it with an input to get a prediction/ classification

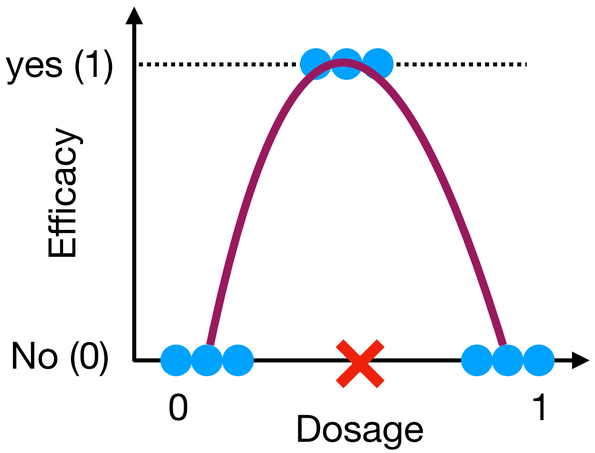




Example

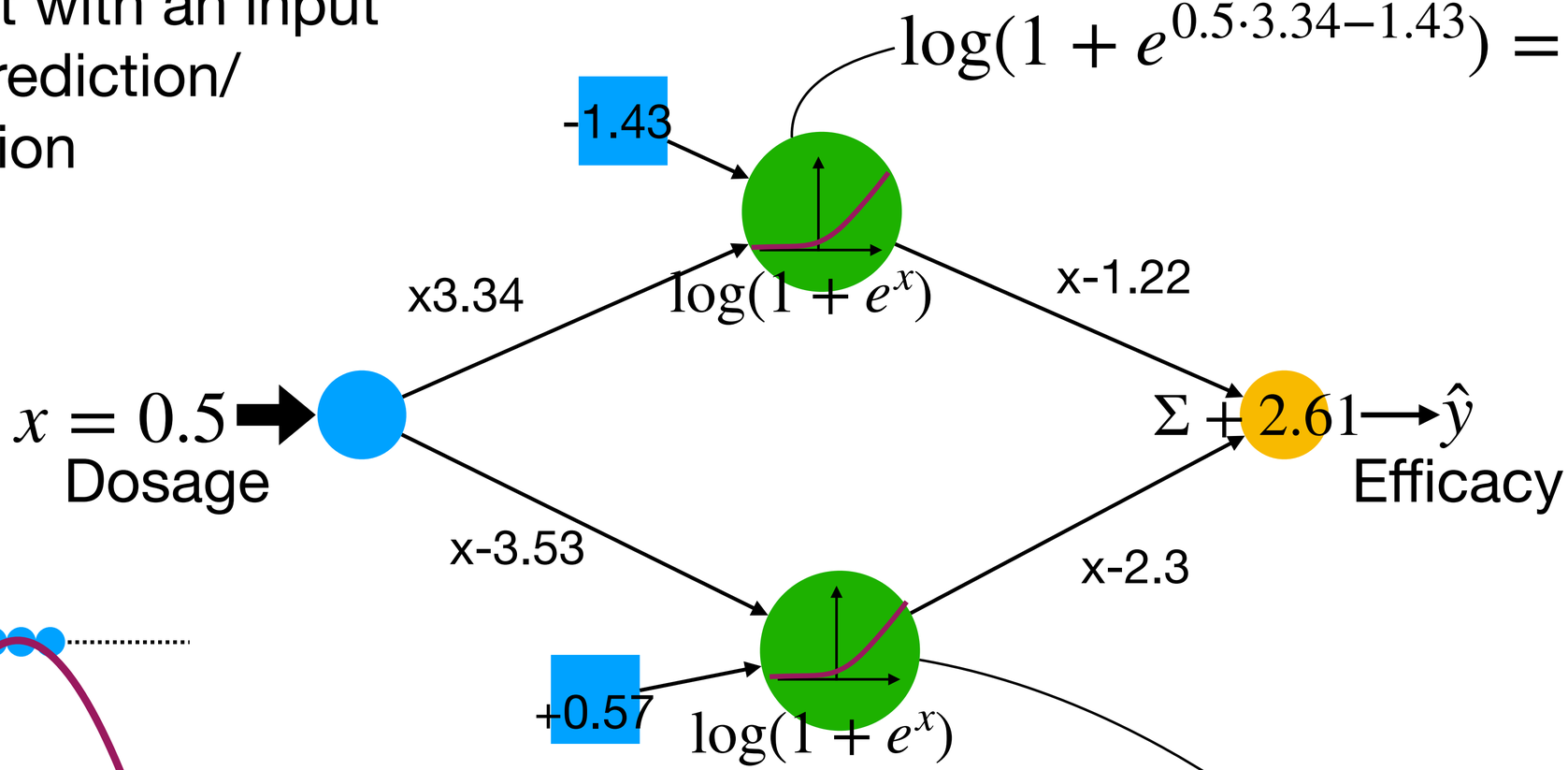
Inference: given a trained NN, feed it with an input to get a prediction/ classification

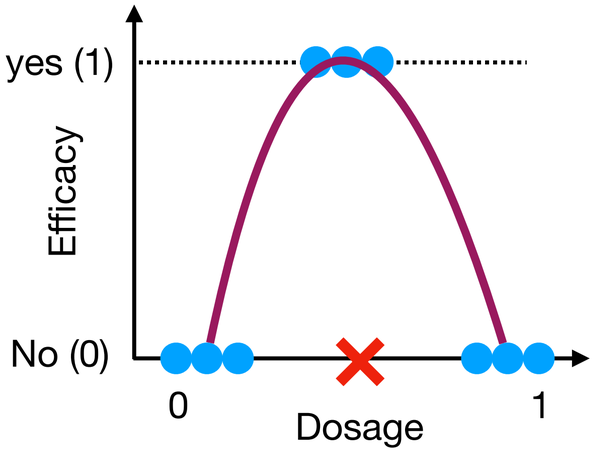




Example

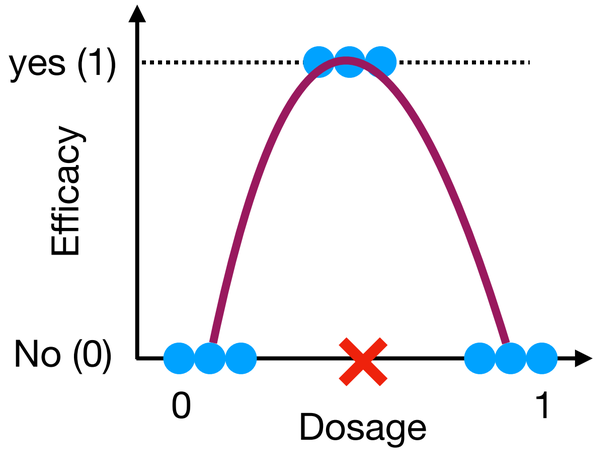
Inference: given a trained NN, feed it with an input to get a prediction/ classification





Example

Inference: given a trained NN, feed it with an input to get a prediction/ classification



$\log(1 + e^{0.5 \cdot 3.34 - 1.43}) = 0.82$

$0.82 \cdot (-1.3) + 0.26 \cdot (-2.23) + 2.61 = 0.94$

-1.43

x3.34

$\log(1 + e^x)$

x-1.22

$x = 0.5$
Dosage

$\Sigma + 2.61 \rightarrow \hat{y}$
Efficacy

x-3.53

x-2.3

+0.57

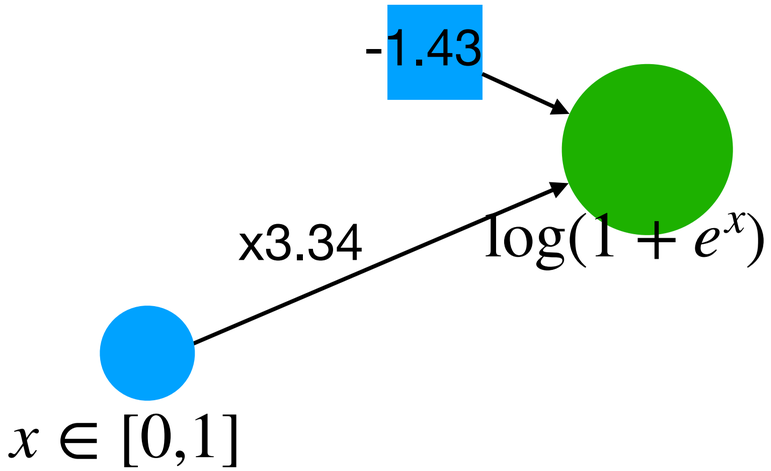
$\log(1 + e^x)$

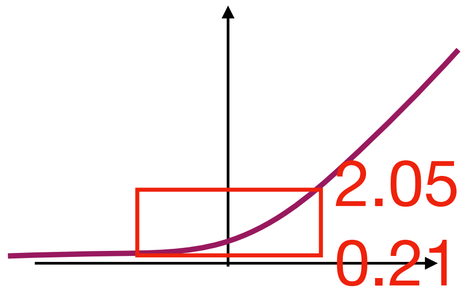
$\log(1 + e^{0.5 \cdot (-3.53) + 0.57}) = 0.26$

Since the output is close to 1, we infer that the dosage is effective.

Example

The output of this node are always in this range:





The input to the soft plus function ranges in

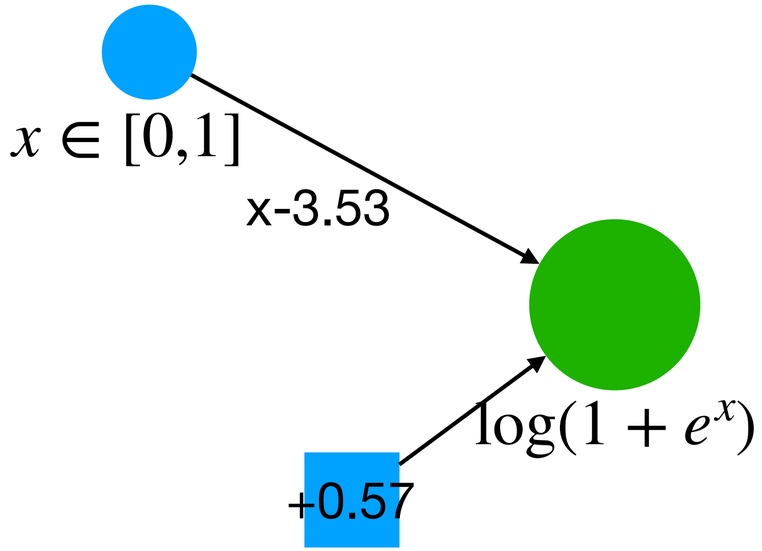
$$[0 \cdot 3.34 - 1.43, 1 \cdot 3.34 - 1.43] =$$

$$[-1.43, 1.91]$$

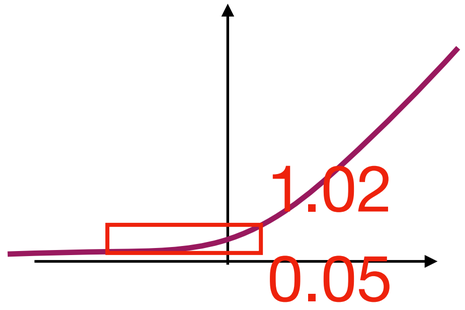
Example

The input to the soft plus function ranges in

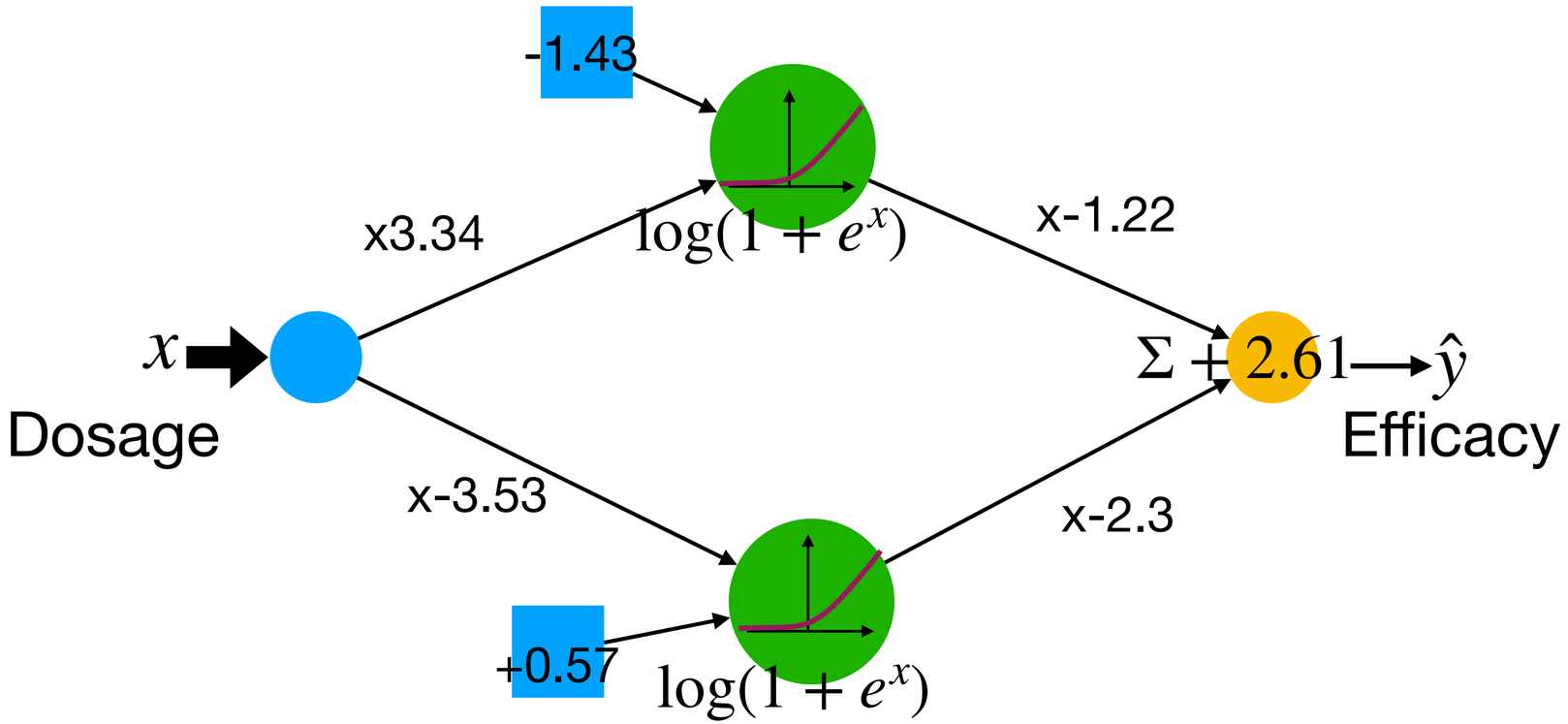
$$[1 \cdot (-3.53) + 0.57, 0 \cdot (-3.53) + 0.57] = [-2.96, 0.57]$$



The output of this node are always in this range:



Example



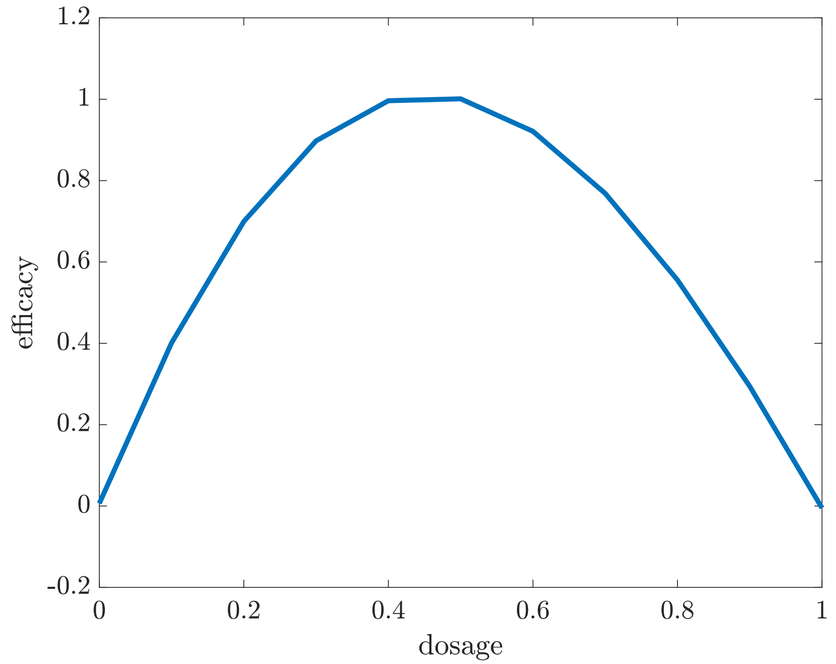
- At the end of the day, this neural network is just this big function of x:

$$-1.22\log(1+\exp(3.34x-1.43))-2.3\log(1+\exp(-3.53x+0.57))+2.61$$

1

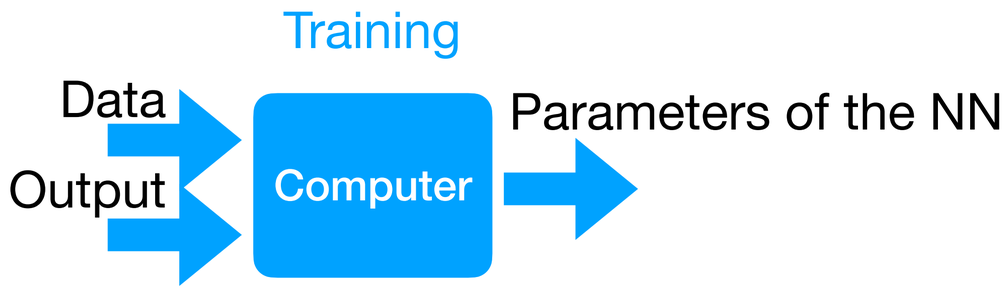
Example

- By plotting the function $-1.22\log(1+\exp(3.34x-1.43))-2.3\log(1+\exp(-3.53x+0.57))+2.61$ for $x \in [0,1]$, we get the following graph.



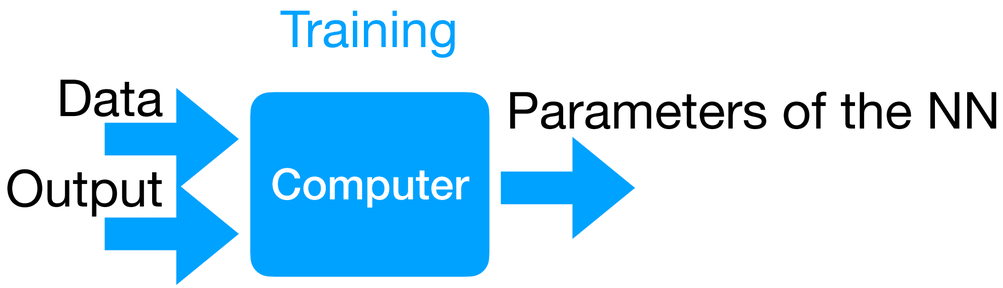
These are unknown

Backpropagation



Backpropagation

- Backpropagation is an algorithm to **train** feedforward neural networks.
- Training a neural network means to find the optimal **parameters** (weights and bias) that model the behaviour of a phenomenon (e.g., how the dosage of a drug influence its efficacy).
- The phenomenon is not described by a mathematical model (e.g., differential equations modelling the body's response to the chemicals in the drug), but only by **observed data** (**data points**, e.g., couples (dosage, efficacy)).
  - Hence, to find the optimal parameters means to find the parameters that minimise the error with respect to the input data.
- To train a neural network, we need to feed it with many data points or samples, that are the **training set**- supervised learning.

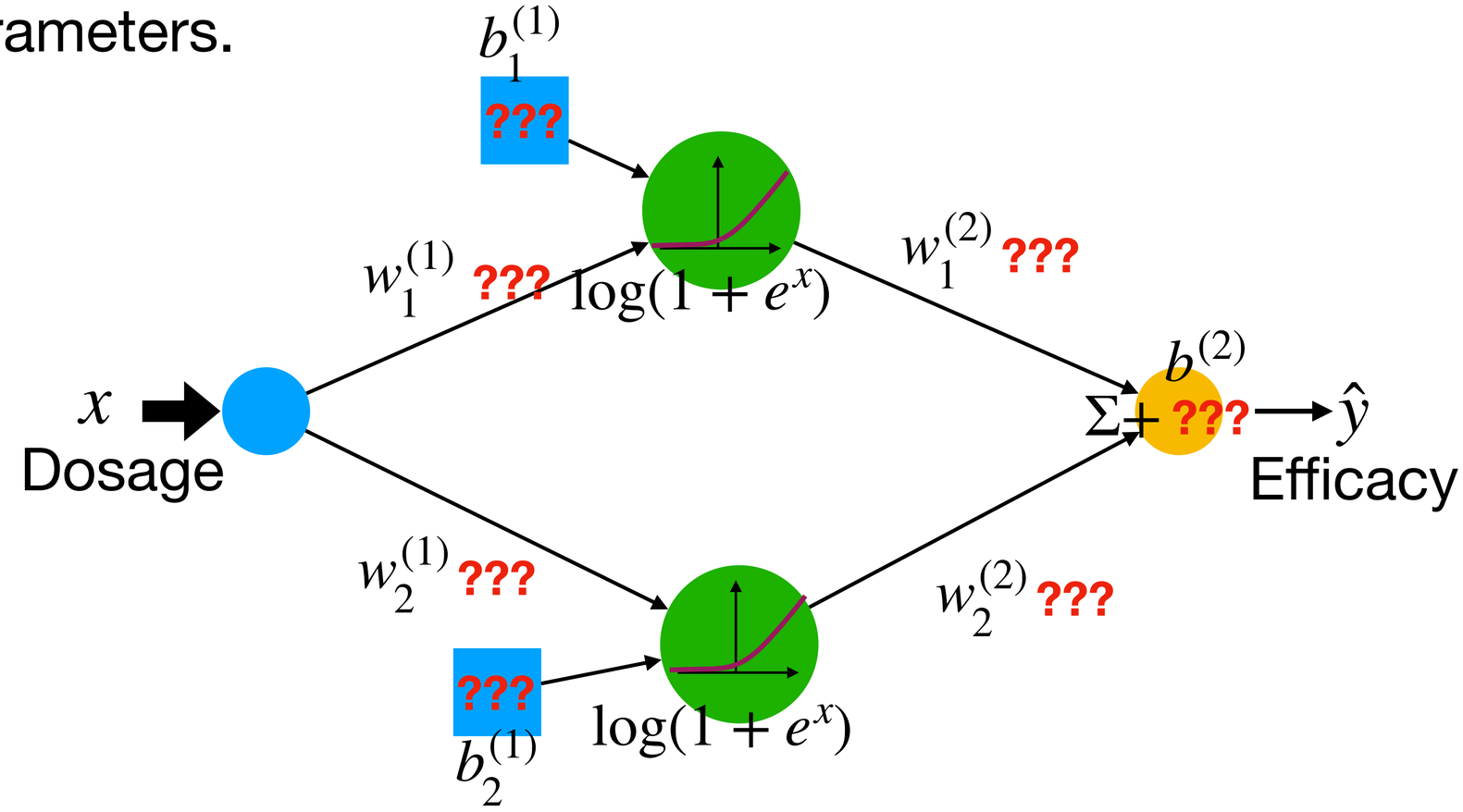


Backpropagation - high level idea

- **Initialisation:** initialise weights and biases at random.
- **Forward step:** feed the network with a data point $(x_1, \ldots, x_d, y)$ and get the expressions of the parameters layer by layer, by writing each parameter as a function of the parameters in the previous layers.
- **Backward step:** Compute the gradients of a loss function $\mathscr{L}$ with respect to **each parameter**, using the **chain rule** from the rightmost layer.
  - Update the parameters with the **gradient descent** step:
    $w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \partial_{w_{i,j}^{(l)}} \mathscr{L}$
    $b_{j}^{(l)} \leftarrow b_{j}^{(l)} - \eta \partial_{b_{j}^{(l)}} \mathscr{L}$
  - Repeat the updates as in the gradient descent until stopping criteria.

Backpropagation - example

- Consider the same scenario as before, but we have no clue about any optimal parameters.

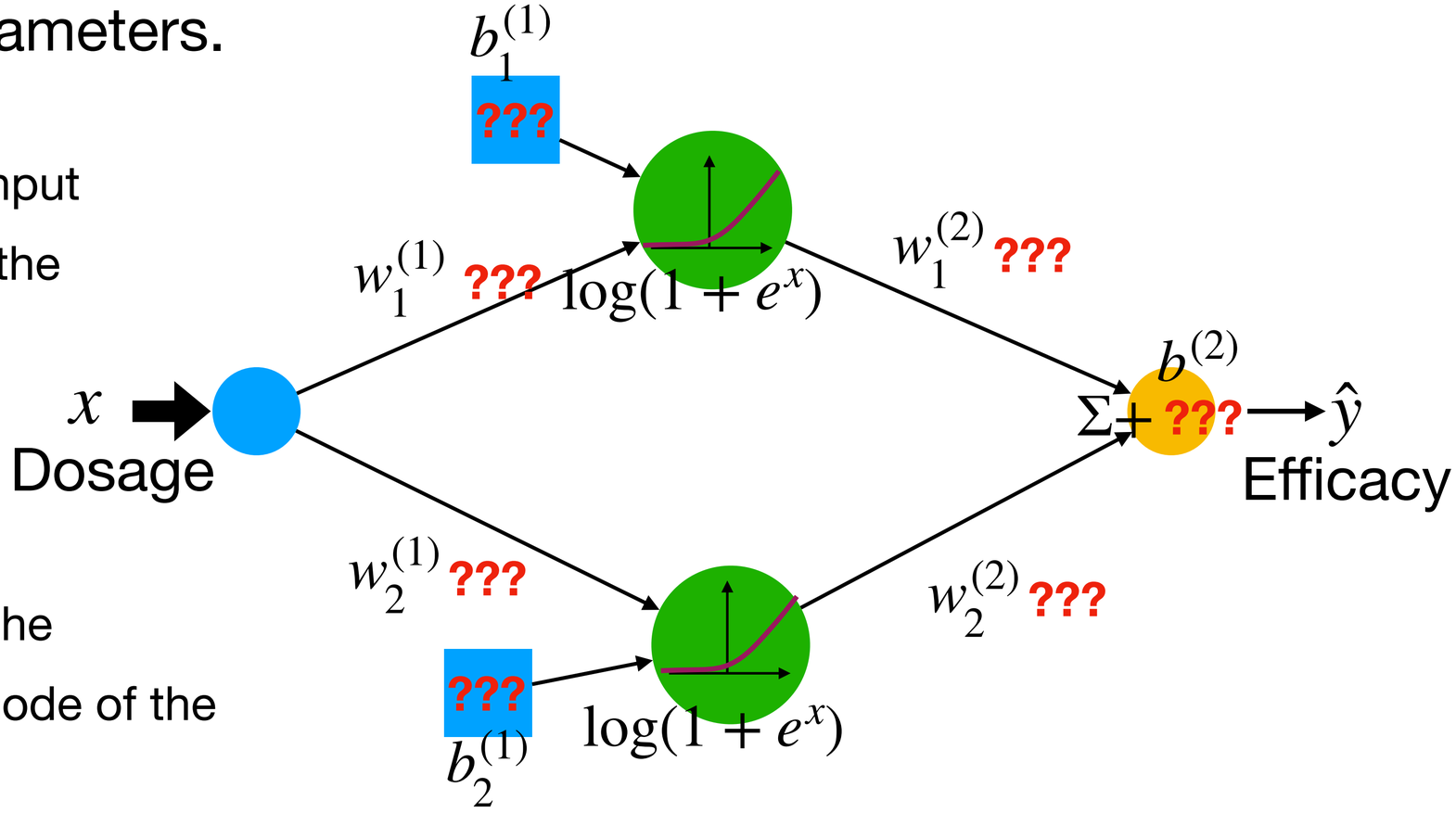


Backpropagation - example

- Consider the same scenario as before, but we have no clue about any optimal parameters.

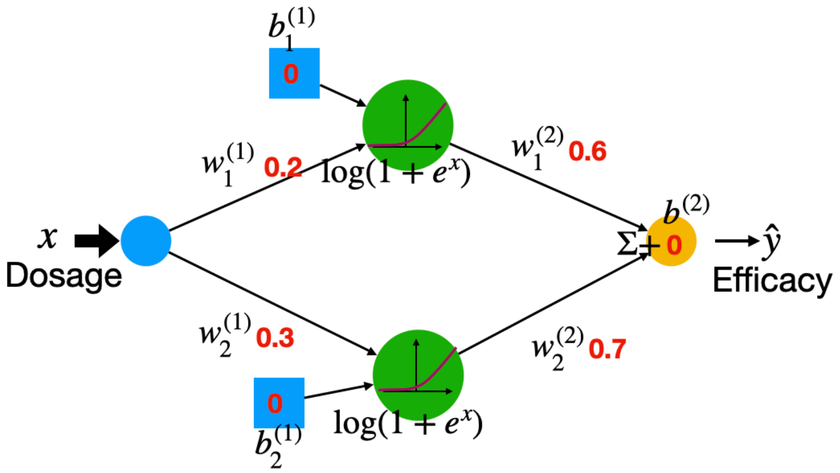
Let's call $net_i^{(l)}$ the input to the $i-$th node of the $l-$th layer.

And let's call $out_i^{(l)}$ the output of the $i-$th node of the $l-$th layer.



Backpropagation - example

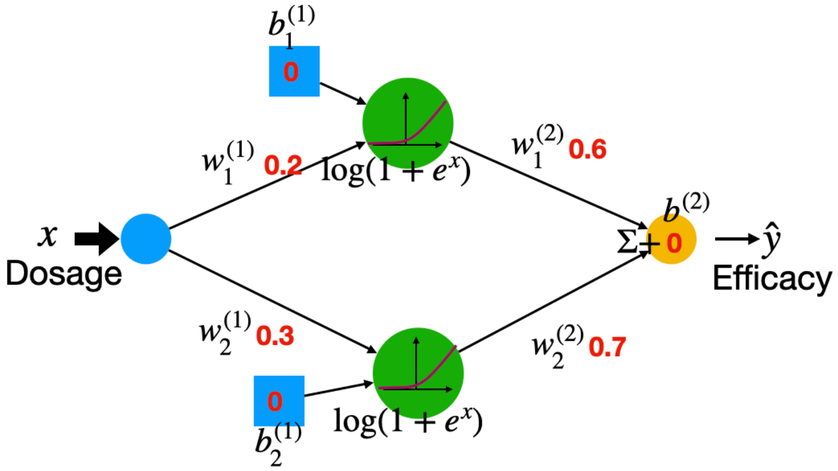
- Random initialisation
- Forward step on training point (0,0)

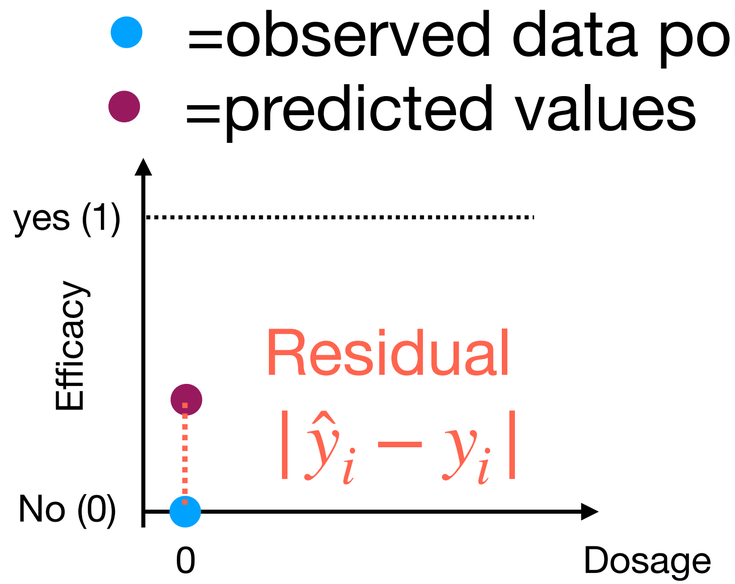


- $net_1^{(1)} = w_1^{(1)}x + b_1^{(1)} = 0$
- $net_2^{(1)} = w_2^{(1)}x + b_2^{(1)} = 0$
- $out_1^{(1)} = \log(1 + \exp(net_1^{(1)}) = 0.69$
- $out_2^{(1)} = \log(1 + \exp(net_2^{(1)}) = 0.69$
- $net^{(2)} = w_1^{(2)}out_1^{(1)} + w_2^{(2)}out_2^{(1)} = 0.9$
- $\hat{y} = net^{(2)} + b^{(2)} = 0.9$

Backpropagation - example

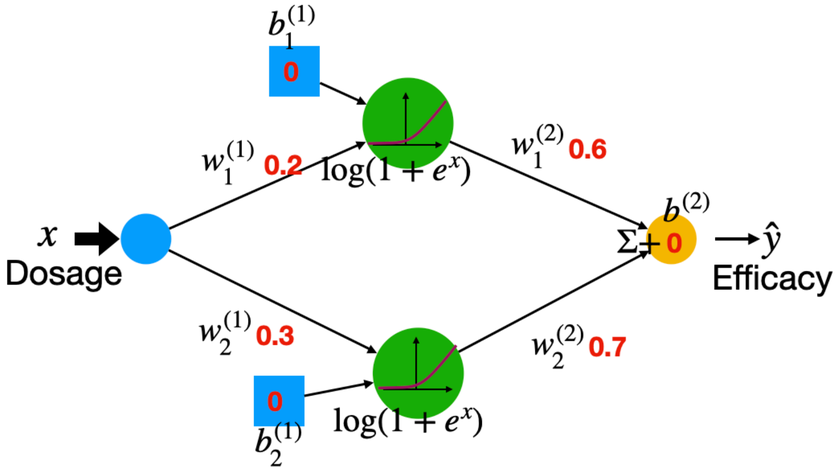
- Calculate the value of the loss function $\mathscr{L} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.9 - 0)^2 = 0.41$





Backpropagation - example

- Backward step:
  - Let's begin by computing the gradients of the loss function wrt all parameters, starting from right-most layer and using the chain rule.



$$net_1^{(1)} = w_1^{(1)}x + b_1^{(1)} = 0 \qquad \mathscr{L} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.9 - 0)^2 = 0.41$$

$$net_2^{(1)} = w_2^{(1)}x + b_2^{(1)} = 0$$

$$out_1^{(1)} = \log(1 + \exp(net_1^{(1)}) = 0.69$$

$$out_2^{(1)} = \log(1 + \exp(net_2^{(1)}) = 0.69$$

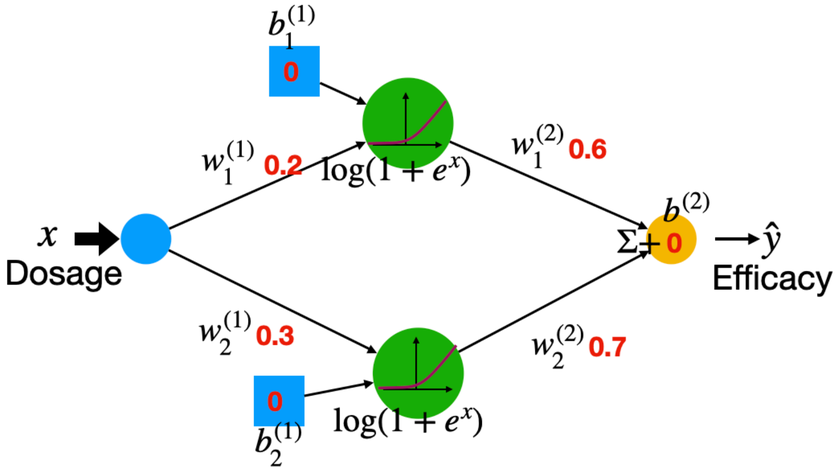
$$net^{(2)} = w_1^{(2)}out_1^{(1)} + w_2^{(2)}out_2^{(1)} = 0.9$$

$$\hat{y} = net^{(2)} + b^{(2)} = 0.9$$

- $\dfrac{\partial \mathscr{L}}{\partial \hat{y}} = \hat{y} - y = 0.9$
- $\dfrac{\partial \mathscr{L}}{\partial b^{(2)}} = \dfrac{\partial \mathscr{L}}{\partial \hat{y}}\dfrac{\partial \hat{y}}{\partial b^{(2)}} = (\hat{y} - y) \cdot 1 = 0.9$
- $\dfrac{\partial \mathscr{L}}{\partial w_1^{(2)}} = \dfrac{\partial \mathscr{L}}{\partial \hat{y}}\dfrac{\partial \hat{y}}{\partial net^{(2)}}\dfrac{\partial net^{(2)}}{\partial w_1^{(2)}} = (\hat{y} - y) \cdot 1 \cdot out_1^{(1)} = 0.9 \cdot 0.69 = 0.62$
- $\dfrac{\partial \mathscr{L}}{\partial w_2^{(2)}} = \dfrac{\partial \mathscr{L}}{\partial \hat{y}}\dfrac{\partial \hat{y}}{\partial net^{(2)}}\dfrac{\partial net^{(2)}}{\partial w_2^{(2)}} = (\hat{y} - y) \cdot 1 \cdot out_2^{(1)} = 0.9 \cdot 0.69 = 0.62$

Backpropagation - example

- Backward step:
  - Let's begin by computing the gradients of the loss function wrt all parameters, starting from right-most layer and using the chain rule.



$$net_1^{(1)} = w_1^{(1)}x + b_1^{(1)} = 0 \qquad \mathscr{L} = \frac{1}{2}(\hat{y}-y)^2 = \frac{1}{2}(0.9-0)^2 = 0.41$$

$$net_2^{(1)} = w_2^{(1)}x + b_2^{(1)} = 0$$

$$out_1^{(1)} = \log(1+\exp(net_1^{(1)}) = 0.69$$

$$out_2^{(1)} = \log(1+\exp(net_2^{(1)}) = 0.69$$

$$net^{(2)} = w_1^{(2)}out_1^{(1)} + w_2^{(2)}out_2^{(1)} = 0.9$$

$$\hat{y} = net^{(2)} + b^{(2)} = 0.9$$

- $$\frac{\partial \mathscr{L}}{\partial b_1^{(1)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial net^{(2)}}\frac{\partial net^{(2)}}{\partial out_1^{(1)}}\frac{\partial out_1^{(1)}}{\partial net_1^{(1)}}\frac{\partial net_1^{(1)}}{\partial b_1^{(1)}} = (\hat{y}-y)\cdot 1\cdot w_1^{(2)}\frac{\exp(net_1^{(1)})}{(1+\exp(net_1^{(1)}))}\cdot 1 = 0.315$$
- $$\frac{\partial \mathscr{L}}{\partial b_2^{(1)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial net^{(2)}}\frac{\partial net^{(2)}}{\partial out_2^{(1)}}\frac{\partial out_2^{(1)}}{\partial net_2^{(1)}}\frac{\partial net_2^{(1)}}{\partial b_2^{(1)}} = (\hat{y}-y)\cdot 1\cdot w_2^{(2)}\frac{\exp(net_2^{(1)})}{(1+\exp(net_2^{(1)}))}\cdot 1 = 0.315$$
- $$\frac{\partial \mathscr{L}}{\partial w_1^{(1)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial net^{(2)}}\frac{\partial net^{(2)}}{\partial out_1^{(1)}}\frac{\partial out_1^{(1)}}{\partial net_1^{(1)}}\frac{\partial net_1^{(1)}}{\partial w_1^{(1)}} = (\hat{y}-y)\cdot 1\cdot w_1^{(2)}\frac{\exp(net_1^{(1)})}{(1+\exp(net_1^{(1)}))}\cdot x = 0$$
- $$\frac{\partial \mathscr{L}}{\partial w_2^{(1)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial net^{(2)}}\frac{\partial net^{(2)}}{\partial out_2^{(1)}}\frac{\partial out_2^{(1)}}{\partial net_2^{(1)}}\frac{\partial net_2^{(1)}}{\partial w_2^{(1)}} = (\hat{y}-y)\cdot 1\cdot w_2^{(2)}\frac{\exp(net_2^{(1)})}{(1+\exp(net_2^{(1)}))}\cdot x = 0$$

Backpropagation - example

$\delta^{(2)}$
- $\frac{\partial \mathcal{L}}{\partial \hat{y}} = \hat{y} - y$
- $\frac{\partial \mathcal{L}}{\partial b^{(2)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b^{(2)}} = (\hat{y} - y) \cdot 1 = \delta^{(2)} \frac{\partial \hat{y}}{\partial b^{(2)}}$
- $\frac{\partial \mathcal{L}}{\partial w_1^{(2)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial w_1^{(2)}} = (\hat{y} - y) \cdot 1 \cdot out_1^{(1)} = \delta^{(2)} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial w_1^{(2)}}$
- $\frac{\partial \mathcal{L}}{\partial w_2^{(2)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial w_2^{(2)}} = (\hat{y} - y) \cdot 1 \cdot out_2^{(1)} = \delta^{(2)} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial w_2^{(2)}}$

$\delta_1^{(1)}$
- $\frac{\partial \mathcal{L}}{\partial b_1^{(1)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_1^{(1)}} \frac{\partial out_1^{(1)}}{\partial net_1^{(1)}} \frac{\partial net_1^{(1)}}{\partial b_1^{(1)}} = (\hat{y} - y) \cdot 1 \cdot w_1^{(2)} \frac{\exp(net_1^{(1)})}{(1 + \exp(net_1^{(1)}))} \cdot 1 = \delta_1^{(1)} \frac{\partial net_1^{(1)}}{\partial b_1^{(1)}}$

$\delta_2^{(1)}$
- $\frac{\partial \mathcal{L}}{\partial b_2^{(1)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_2^{(1)}} \frac{\partial out_2^{(1)}}{\partial net_2^{(1)}} \frac{\partial net_2^{(1)}}{\partial b_2^{(1)}} = (\hat{y} - y) \cdot 1 \cdot w_2^{(2)} \frac{\exp(net_2^{(1)})}{(1 + \exp(net_2^{(1)}))} \cdot 1 = \delta_2^{(1)} \frac{\partial net_2^{(1)}}{\partial b_2^{(1)}}$
- $\frac{\partial \mathcal{L}}{\partial w_1^{(1)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_1^{(1)}} \frac{\partial out_1^{(1)}}{\partial net_1^{(1)}} \frac{\partial net_1^{(1)}}{\partial w_1^{(1)}} = (\hat{y} - y) \cdot 1 \cdot w_1^{(2)} \frac{\exp(net_1^{(1)})}{(1 + \exp(net_1^{(1)}))} \cdot x = \delta_1^{(1)} \frac{\partial net_1^{(1)}}{\partial w_1^{(1)}}$
- $\frac{\partial \mathcal{L}}{\partial w_2^{(1)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_2^{(1)}} \frac{\partial out_2^{(1)}}{\partial net_2^{(1)}} \frac{\partial net_2^{(1)}}{\partial w_2^{(1)}} = (\hat{y} - y) \cdot 1 \cdot w_2^{(2)} \frac{\exp(net_2^{(1)})}{(1 + \exp(net_2^{(1)}))} \cdot x = \delta_2^{(1)} \frac{\partial net_2^{(1)}}{\partial w_2^{(1)}}$

$$\delta_1^{(1)} = \delta^{(2)} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_1^{(1)}} \frac{\partial out_1^{(1)}}{\partial net_1^{(1)}}$$

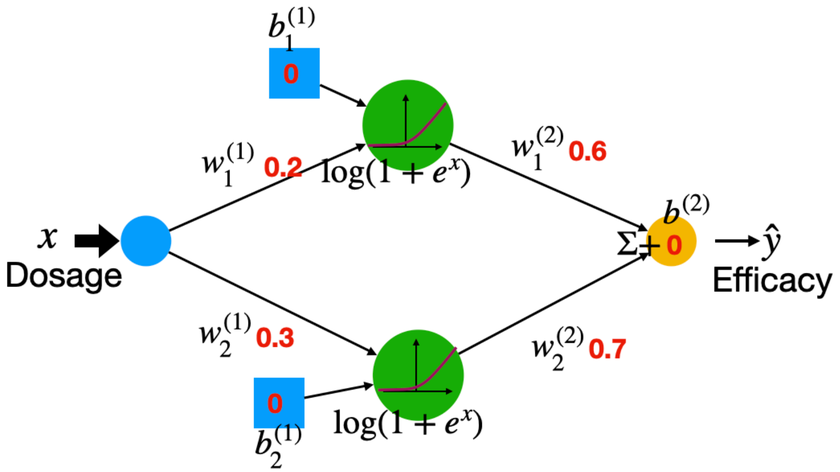
$$\delta_2^{(1)} = \delta^{(2)} \frac{\partial \hat{y}}{\partial net^{(2)}} \frac{\partial net^{(2)}}{\partial out_2^{(1)}} \frac{\partial out_2^{(1)}}{\partial net_2^{(1)}}$$

$$\delta_v^{(l)} = f'\left(net_v^{(l)}\right) \cdot \sum_{i=1}^{n} w_{v,i}^{(l+1)} \cdot \delta_i^{(l+1)}$$

General formulation for a node v at a hidden layer l, v having n outgoing edges

Backpropagation - example

- Backward step:
  - Updates all weights and biases and iterate until convergence.



$$w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \frac{\partial \mathscr{L}}{\partial w_{i,j}^{(l)}}$$

$$b_j^{(l)} \leftarrow b_j^{(l)} - \eta \frac{\partial \mathscr{L}}{\partial b_j^{(l)}}$$

$$net_1^{(1)} = w_1^{(1)} x + b_1^{(1)} = 0 \qquad \mathscr{L} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.9 - 0)^2 = 0.41$$

$$net_2^{(1)} = w_2^{(1)} x + b_2^{(1)} = 0$$

$$out_1^{(1)} = \log(1 + \exp(net_1^{(1)}) = 0.69$$

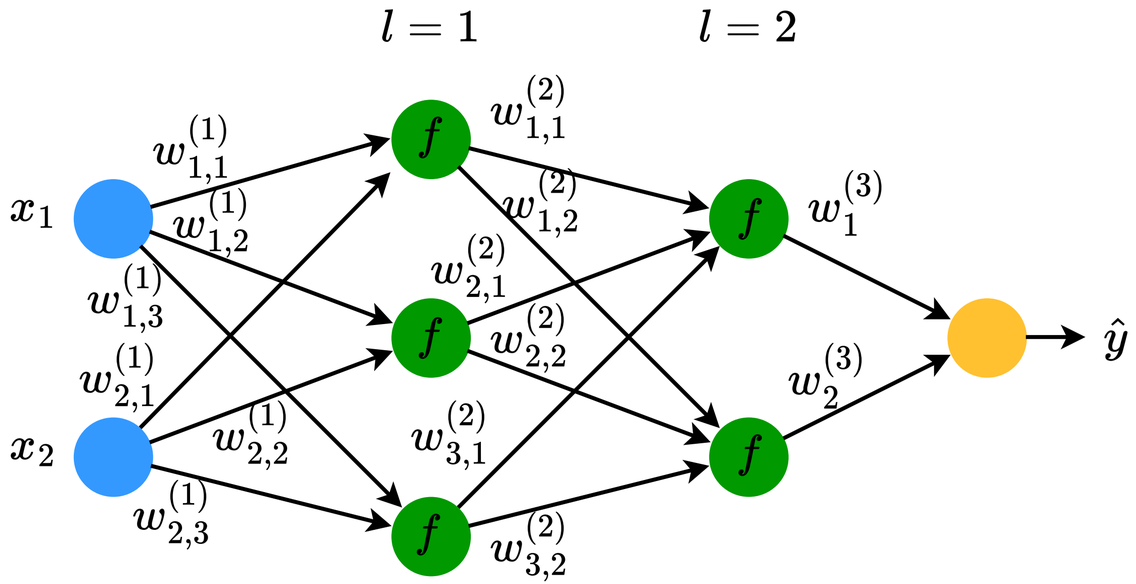
$$out_2^{(1)} = \log(1 + \exp(net_2^{(1)}) = 0.69$$

$$net^{(2)} = w_1^{(2)} out_1^{(1)} + w_2^{(2)} out_2^{(1)} = 0.9$$

$$\hat{y} = net^{(2)} + b^{(2)} = 0.9$$

Backpropagation - example 2

- Forward step

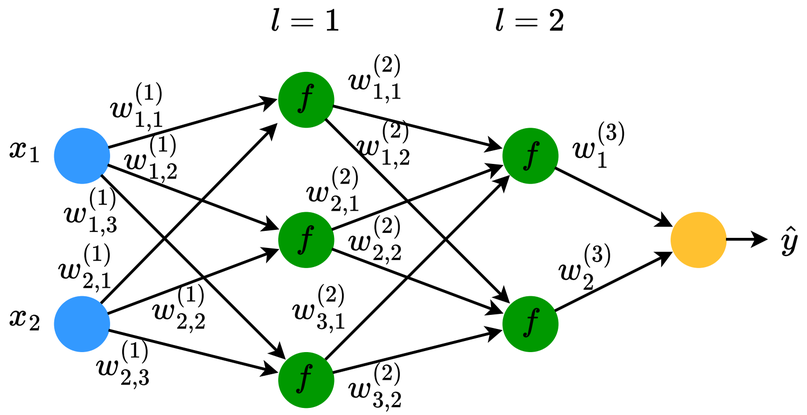


$$f(x) = \log(1 + e^x)$$

- $net_1^{(1)} = w_{1,1}^{(1)} x_1 + w_{2,1}^{(1)} x_2$
- $net_2^{(1)} = w_{1,2}^{(1)} x_1 + w_{2,2}^{(1)} x_2$
- $net_3^{(1)} = w_{1,3}^{(1)} x_1 + w_{2,3}^{(1)} x_2$
- $out_1^{(1)} = \log(1 + \exp(net_1^{(1)}))$
- $out_2^{(1)} = \log(1 + \exp(net_2^{(1)}))$
- $out_3^{(1)} = \log(1 + \exp(net_3^{(1)}))$
- $net_1^{(2)} = w_{1,1}^{(2)} out_1^{(1)} + w_{2,1}^{(2)} out_2^{(1)} + w_{3,1}^{(2)} out_3^{(1)}$
- $net_2^{(2)} = w_{1,2}^{(2)} out_1^{(1)} + w_{2,2}^{(2)} out_2^{(1)} + w_{3,2}^{(2)} out_3^{(1)}$
- $out_1^{(2)} = \log(1 + \exp(net_1^{(2)}))$
- $out_2^{(2)} = \log(1 + \exp(net_2^{(2)}))$
- $\hat{y} = out_1^{(2)} w_1^{(3)} + out_2^{(2)} w_2^{(3)}$

Backpropagation - example 2

- Define loss function as $\mathscr{L} = \frac{1}{2}(\hat{y} - y)^2$ and compute the gradients backwards



$$f(x) = \log(1 + e^x)$$

$$net_1^{(1)} = w_{1,1}^{(1)}x_1 + w_{2,1}^{(1)}x_2$$
$$net_2^{(1)} = w_{1,2}^{(1)}x_1 + w_{2,2}^{(1)}x_2$$
$$net_3^{(1)} = w_{1,3}^{(1)}x_1 + w_{2,3}^{(1)}x_2$$
$$out_1^{(1)} = \log(1 + \exp(net_1^{(1)}))$$
$$out_2^{(1)} = \log(1 + \exp(net_2^{(1)}))$$
$$out_3^{(1)} = \log(1 + \exp(net_3^{(1)}))$$

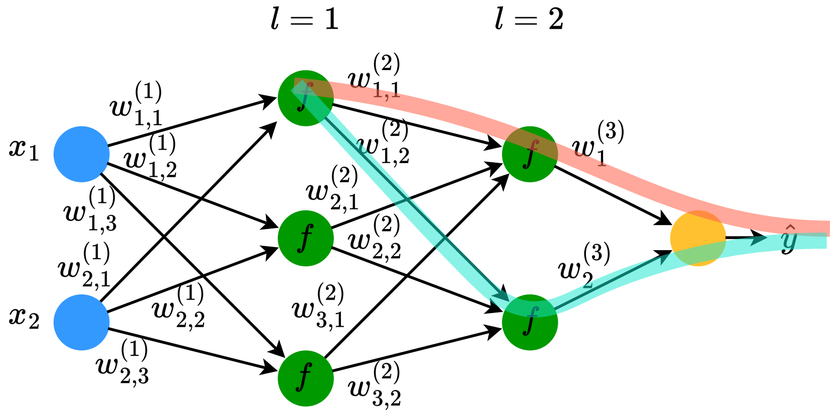
$$net_1^{(2)} = w_{1,1}^{(2)}out_1^{(1)} + w_{2,1}^{(2)}out_2^{(1)} + w_{3,1}^{(2)}out_3^{(1)}$$
$$net_2^{(2)} = w_{1,2}^{(2)}out_1^{(1)} + w_{2,2}^{(2)}out_2^{(1)} + w_{3,2}^{(2)}out_3^{(1)}$$
$$out_1^{(2)} = \log(1 + \exp(net_1^{(2)}))$$
$$out_2^{(2)} = \log(1 + \exp(net_2^{(2)}))$$
$$\hat{y} = out_1^{(2)}w_1^{(3)} + out_2^{(2)}w_2^{(3)}$$

- $\frac{\partial \mathscr{L}}{\partial \hat{y}} = (\hat{y} - y)$
- $\frac{\partial \mathscr{L}}{\partial w_1^{(3)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_1^{(3)}} = (\hat{y} - y) \cdot out_1^{(2)}$
- $\frac{\partial \mathscr{L}}{\partial w_{1,1}^{(2)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial out_1^{(2)}}\frac{\partial out_1^{(2)}}{\partial net_1^{(2)}}\frac{\partial net_1^{(2)}}{\partial w_{1,1}^{(2)}} =$

$$= (\hat{y} - y) \cdot w_1^{(3)}\frac{\exp(net_1(2))}{(1 + \exp(net_1^{(2)}))}out_1^{(1)}$$

Backpropagation - example 2

- Define loss function as $\mathscr{L} = \frac{1}{2}(\hat{y} - y)^2$ and compute the gradients backwards



$$f(x) = \log(1 + e^x)$$

$$net_1^{(1)} = w_{1,1}^{(1)} x_1 + w_{2,1}^{(1)} x_2$$

$$net_2^{(1)} = w_{1,2}^{(1)} x_1 + w_{2,2}^{(1)} x_2$$

$$net_3^{(1)} = w_{1,3}^{(1)} x_1 + w_{2,3}^{(1)} x_2$$

$$out_1^{(1)} = \log(1 + \exp(net_1^{(1)}))$$

$$out_2^{(1)} = \log(1 + \exp(net_2^{(1)}))$$

$$out_3^{(1)} = \log(1 + \exp(net_3^{(1)}))$$

$$net_1^{(2)} = w_{1,1}^{(2)} out_1^{(1)} + w_{2,1}^{(2)} out_2^{(1)} + w_{3,1}^{(2)} out_3^{(1)}$$

$$net_2^{(2)} = w_{1,2}^{(2)} out_1^{(1)} + w_{2,2}^{(2)} out_2^{(1)} + w_{3,2}^{(2)} out_3^{(1)}$$

$$out_1^{(2)} = \log(1 + \exp(net_1^{(2)}))$$

$$out_2^{(2)} = \log(1 + \exp(net_2^{(2)}))$$

$$\hat{y} = out_1^{(2)} w_1^{(3)} + out_2^{(2)} w_2^{(3)}$$

$$\frac{\partial \mathscr{L}}{\partial w_{1,1}^{(1)}} = \frac{\partial \mathscr{L}}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial out_1^{(2)}} \frac{\partial out_1^{(2)}}{\partial net_1^{(2)}} \frac{\partial net_1^{(2)}}{\partial out_1^{(1)}} + \frac{\partial \hat{y}}{\partial out_2^{(2)}} \frac{\partial out_2^{(2)}}{\partial net_2^{(2)}} \frac{\partial net_2^{(2)}}{\partial out_1^{(1)}} \right) \frac{\partial out_1^{(1)}}{\partial net_1^{(1)}} \frac{net_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

The partial derivatives with respect to the other weights in the first layer are computed similarly.

Backpropagation

- We can train the neural network over a dataset of N samples $(x_{i,1}, \ldots, x_{i,d}, y_i)$.
- The process works as follows:
  - Initialise weights and biases at random.
  - For each batch of n elements:
    - Feed the network with each sample (data point), obtain the prediction $\hat{y}_i$ and compute the corresponding values of the loss function $\mathscr{L}(x_{i,1}, \ldots, x_{i,d}, y_i) = \mathscr{L}_i = \frac{1}{2}(\hat{y}_i - y)^2$
    - Consider the average of the values of the loss functions: $\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}_i$
    - Compute gradients of $\mathscr{L} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}_i$ wrt each parameter, and proceed until stopping

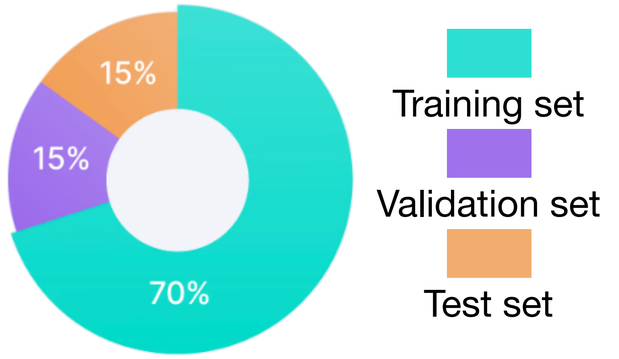
This is one epoch

Backpropagation

- This procedure can be repeated on multiple **epochs**.
  - Divide the training set into a number of batches.
  - **First epoch:** Initialise weights and biases at random. Train the network on the **all the batches**, one at a time (forward and backward).
  - **k-th epoch**: start the training using the latest parameters computed in the previous epoch. Train the network on all the batches.
  - Repeat until the maximum number of epochs has been reached.

- Notice:
  - When we do not apply the gradient descent to the entire dataset, but to just one sample or a batch of samples, we refer to it as to "**stochastic gradient descent**" (**SDG**). Batches are usually chosen uniformly at random.

- The entire dataset is divided into a **training set**, a **validation set**, and a **test set.**
- At each epoch, evaluate the performance of the obtained NN on an independent **validation set**.

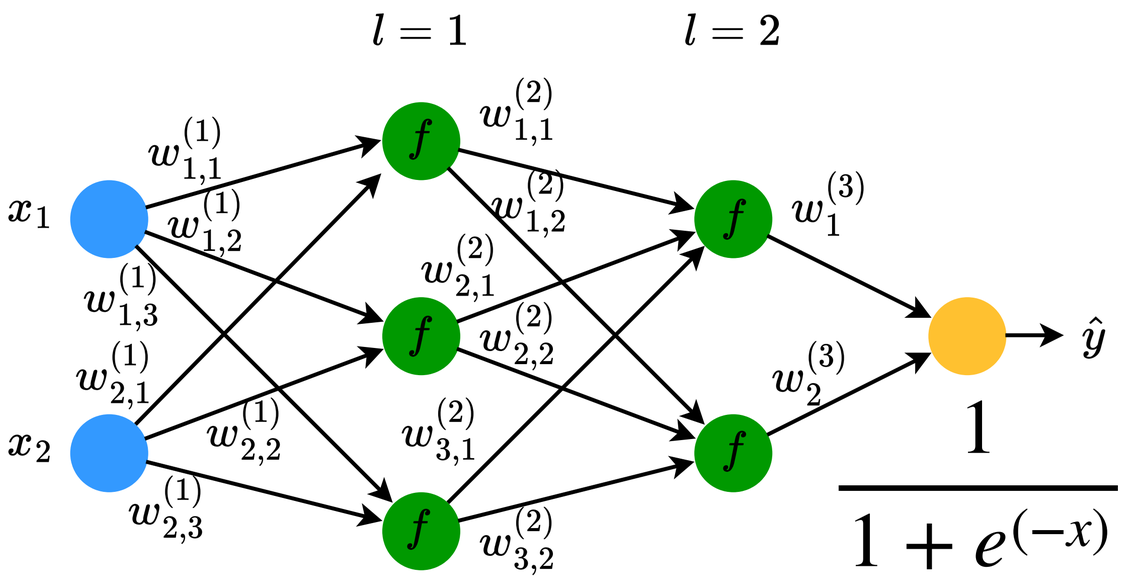


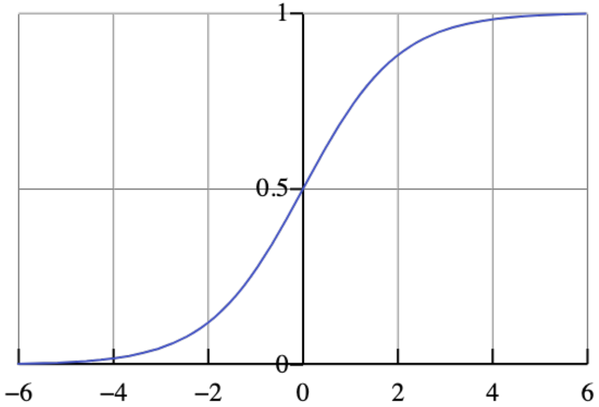
- After the last epoch, select the parameters that achieved the best performance on the validation set.
- Evaluate the generalisability of the NN on an independent **test set.**
- How to evaluate it?
  - If the NN's task is **regression**, then use MAE, MSE.
  - If the NN's task is **classification**, then use accuracy/precision/recall/F1-score.
- Learning rate, batch size, number of epochs are additional hyper parameters to be set before the training phase.

NN for classification

- Neural network for **binary classification**, where the samples belong to one of two possible classes (e.g., spam detection, disease detection).

- One output node with sigmoid activation function.



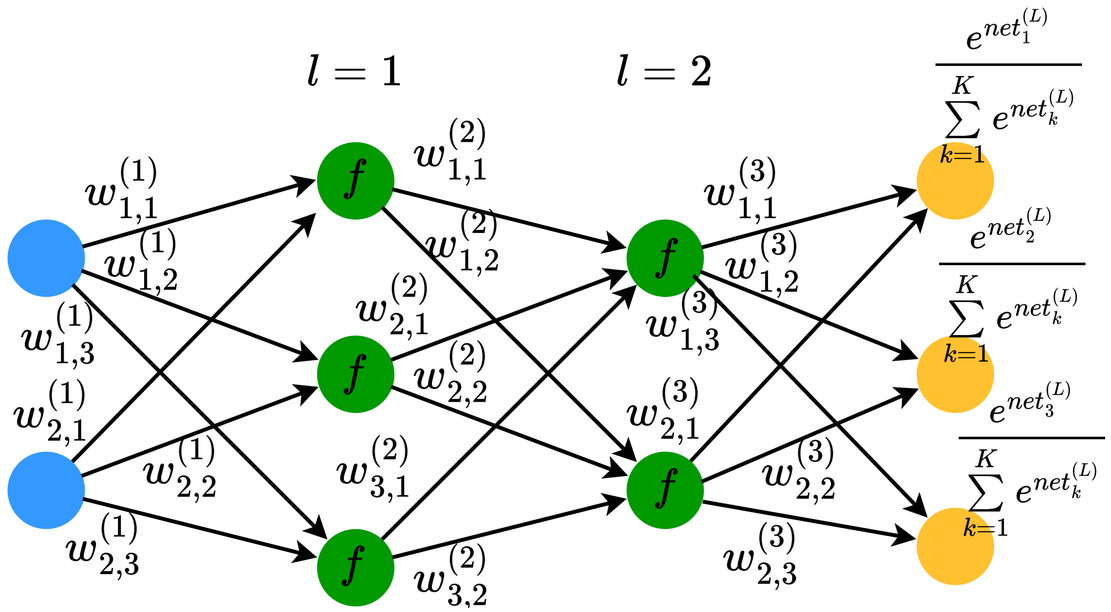


The output is a number between 0 and 1, that represents the "probability" that the sample belongs to one of the two classes

NN for classification

- Neural network for classification, where the samples belong to one of K >2 possible classes (e.g., image classification, sentiment analysis, etc).

- Number of output nodes = number of classes, each with **softmax activation** function.



Each output node outputs a number between 0 and 1, representing the probability of belonging to the class associated with the node.

NN for classification

- If classes are not numerical (e.g., animal recognition, presence or absence of a disease), we perform **label encoding**, meaning that we assign a numerical value to each class (e.g., cats=0, dogs = 1, mice=2, etc).
  - Some similar techniques are used also for representing non numerical features
- What loss function to use for classification tasks?

Binary classification:
binary cross entropy

One sample: $y \log_2(p) + (1-y)\log(1-p)$

n samples: $-\frac{1}{n}\sum_{i=1}^{n}[y_i \log_2(p_i) + (1-y_1)\log(1-p_i)]$

Multi-class classification:
Categorical Cross-Entropy

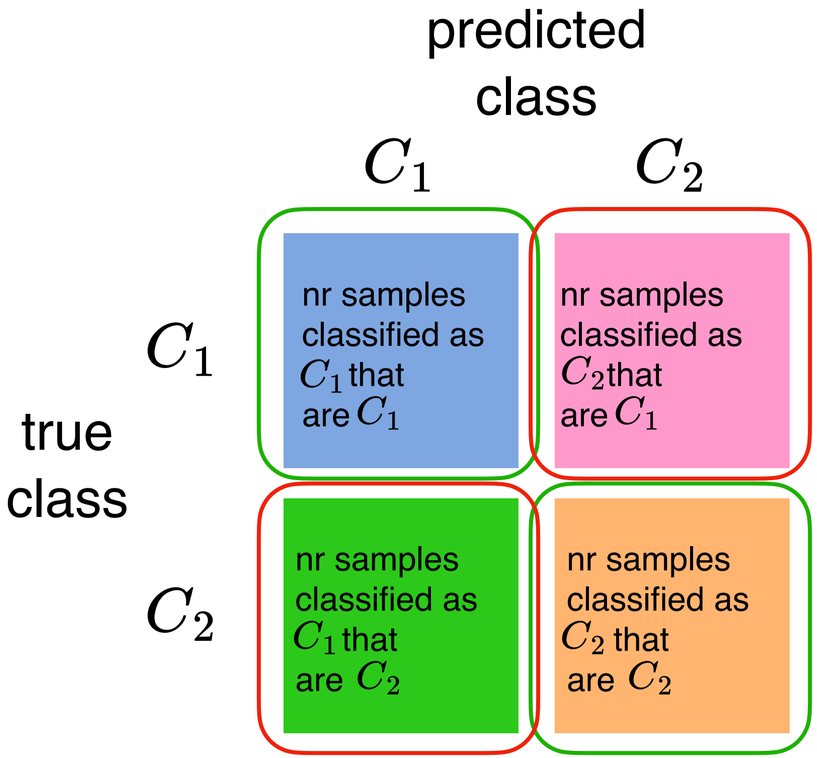
One sample: $-\sum_{k=1}^{K} y_k \log_2(p_k)$

n samples: $-\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_i \log_2(p_i)$

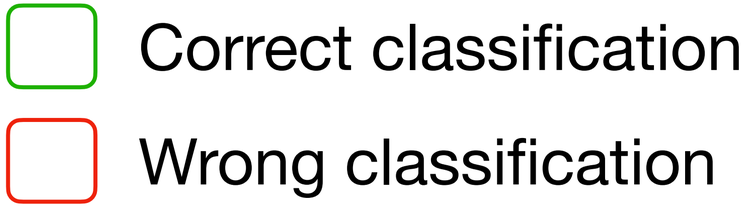
p is the inferred probability and equivalent to $\hat{y}$

Metrics for classification

- Binary classification, two classes, C_1 and C_2.

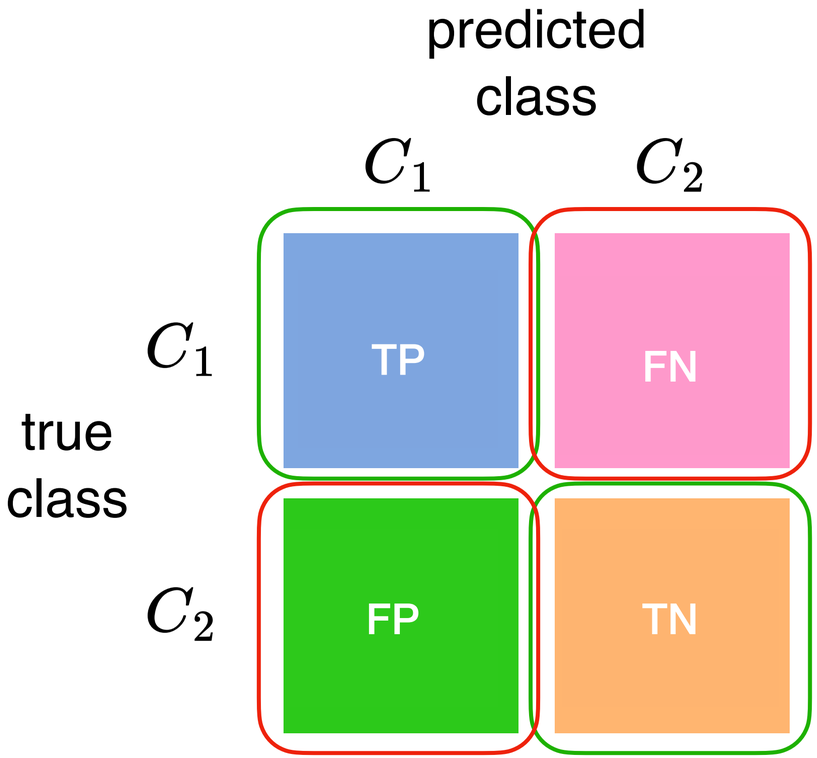


Confusion matrix



Metrics for classification

- Binary classification, two classes, C_1 and C_2.



Confusion matrix

Correct classification

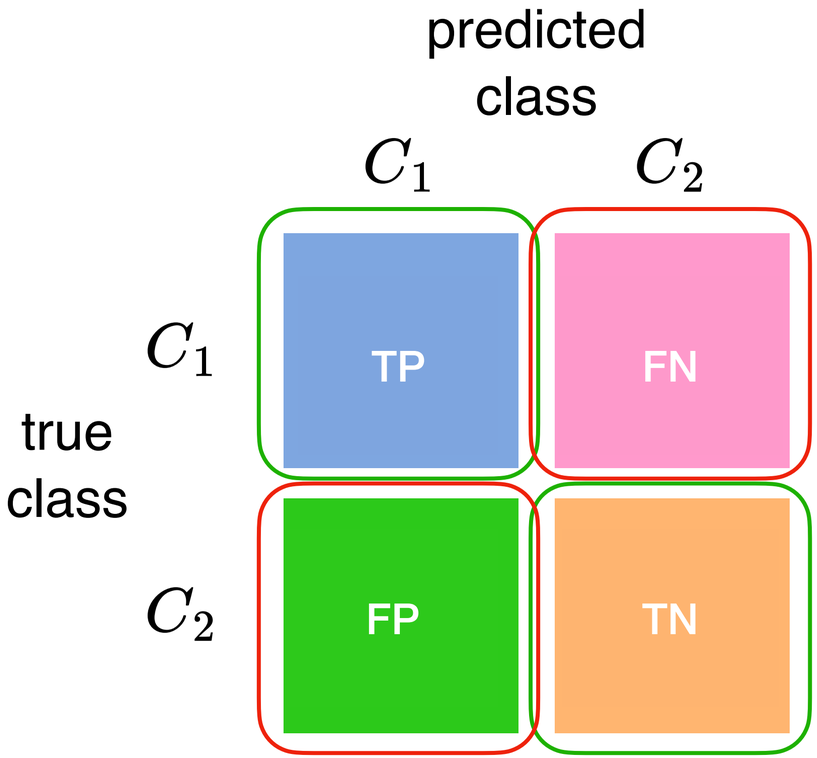
Wrong classification

- Assume C_1 is the "positive" class and C_2 is the negative class.
- TP = true positive
- TN = true negative
- FP = false positive
- FN = false negative

By dividing them by the total number of samples, we get their rates.

Metrics for classification

- Binary classification, two classes, C_1 and C_2.



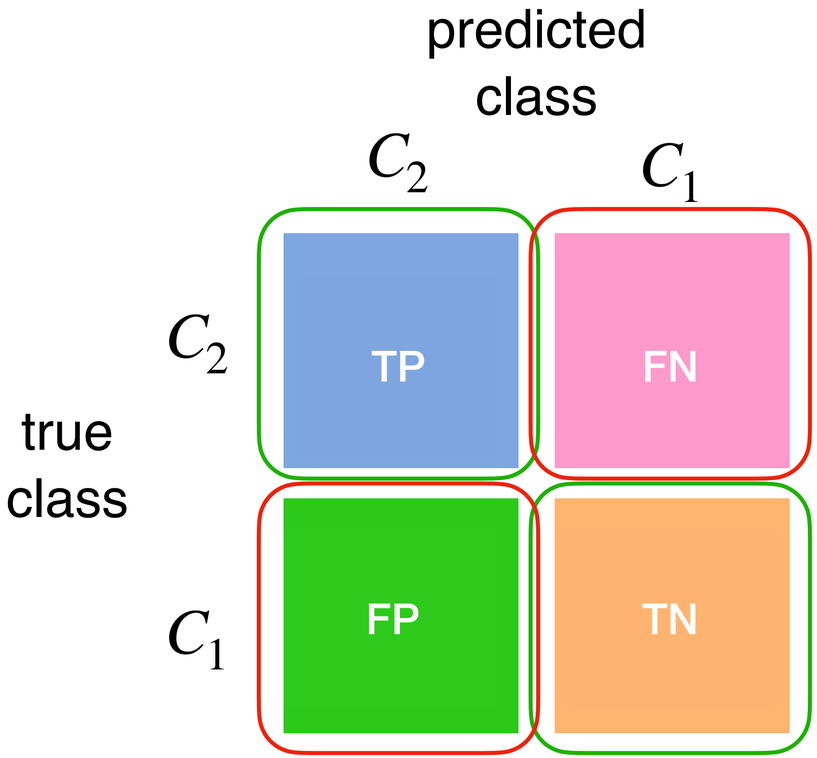
Confusion matrix

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$ Symmetric
- Recall $_{C_1}$: $\frac{TP}{P} = \frac{TP}{TP + FN}$ Non Symmetric
- Precision $_{C_1}$: $\frac{TP}{TP + FP}$ Non Symmetric

$$F1_{C_1} = \frac{1}{2} \frac{\text{Precision}_{C_1} \cdot \text{Recall}_{C_1}}{\text{Precision}_{C_1} + \text{Recall}_{C_1}}$$

Metrics for classification

- Binary classification, two classes, C_1 and C_2.



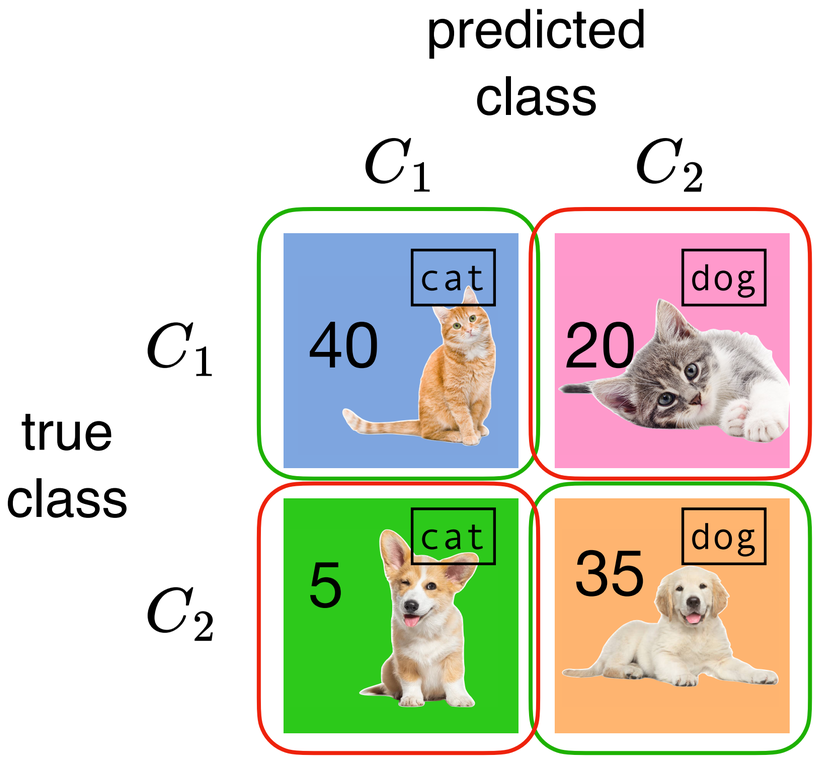
Confusion matrix

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$
- Recall $_{C_2}$: $\frac{TP}{P} = \frac{TP}{TP + FN}$
- Precision $_{C_2}$: $\frac{TP}{TP + FP}$

Invert classes in the confusion matrix or invert positives and negatives

Metrics for classification - example

- Binary classification, two classes, C_1, cats, and C_2, dogs.
- Training set: 100 samples, 60 cats and 40 dogs.

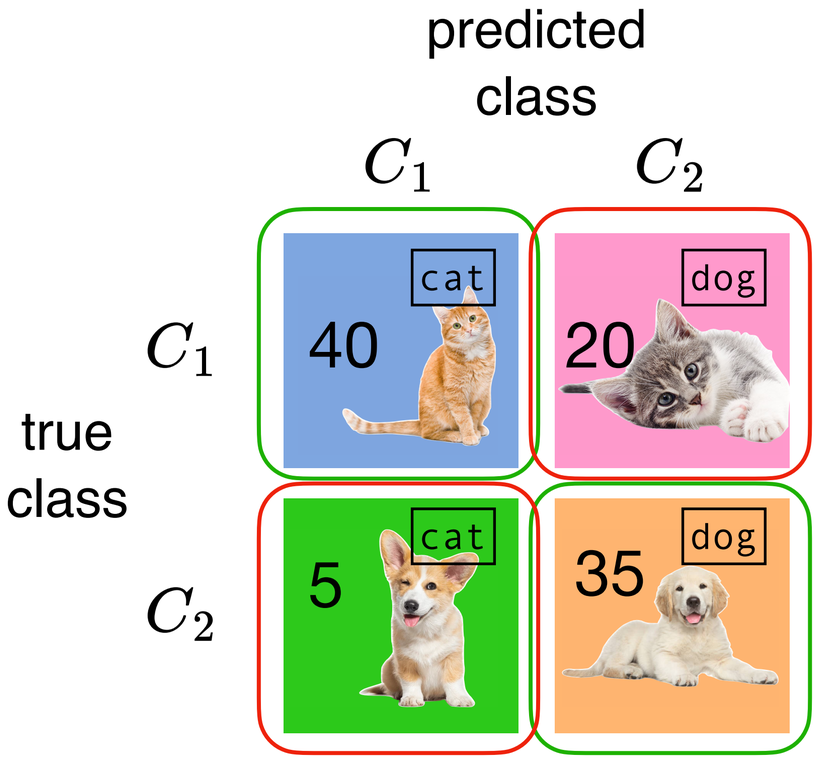


TP=40, TN=35, FP = 5, FN = 20

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN} = \frac{75}{100}$
- Recall$_{C_1}$: $\frac{TP}{P} = \frac{TP}{TP + FN} = \frac{40}{60}$
- Precision$_{C_1}$: $\frac{TP}{TP + FP} = \frac{40}{45}$

Metrics for classification - example

- Binary classification, two classes, C_1, cats, and C_2, dogs.
- Training set: 100 samples, 60 cats and 40 dogs.

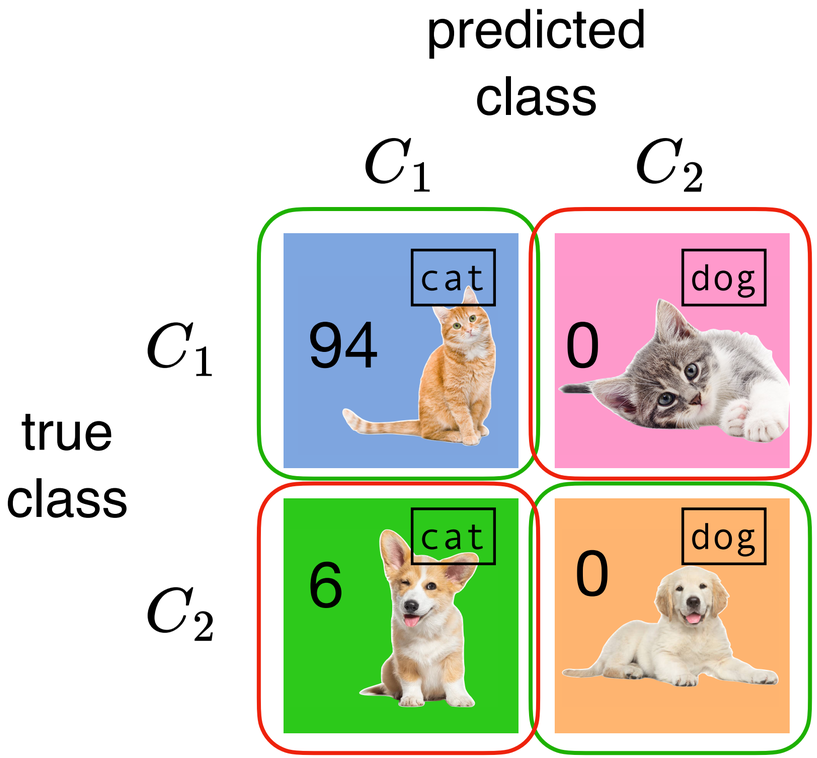


TP=40, TN=35, FP = 5, FN = 20

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN} = \frac{75}{100}$
- Recall$_{C_2}$: $\frac{TP}{P} = \frac{TP}{TP + FN} = \frac{35}{40}$
- Precision$_{C_2}$: $\frac{TP}{TP + FP} = \frac{35}{55}$

Metrics for classification - example

- Accuracy can be misleading for unbalanced datasets.
- Training set: 100 samples, 94 cats and 6 dogs. NN just outputs “cats”.

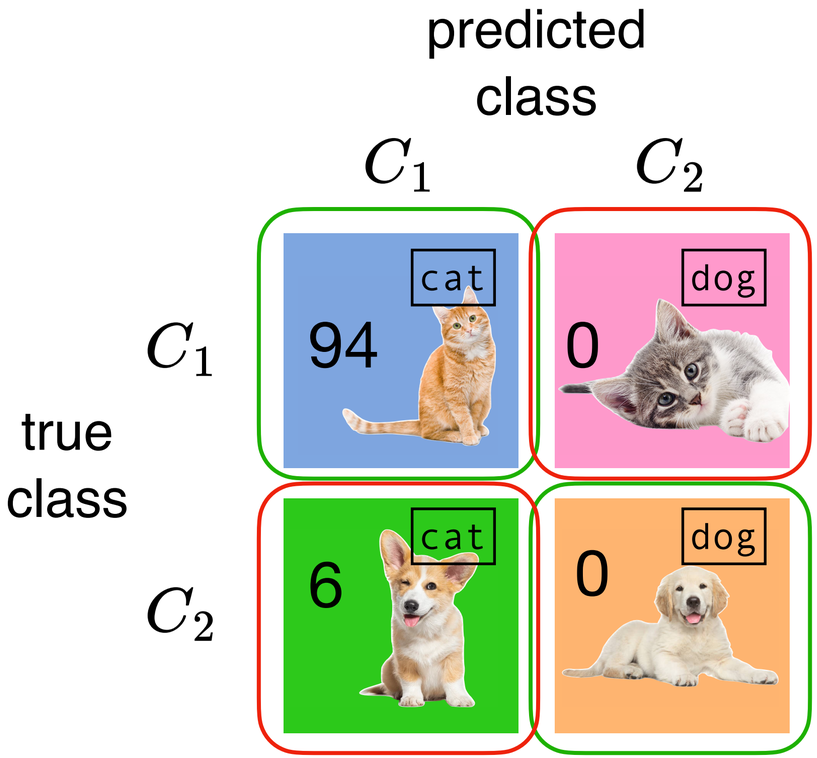


TP=94, TN=0, FP = 6, FN = 0

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN} = \frac{94}{100}$
- Recall$_{C_1}$: $\frac{TP}{P} = \frac{TP}{TP + FN} = \frac{94}{94}$
- Precision$_{C_1}$: $\frac{TP}{TP + FP} = \frac{94}{100}$

Metrics for classification - example

- Accuracy can be misleading for unbalanced datasets.
- Training set: 100 samples, 94 cats and 6 dogs. NN just outputs “cats”.

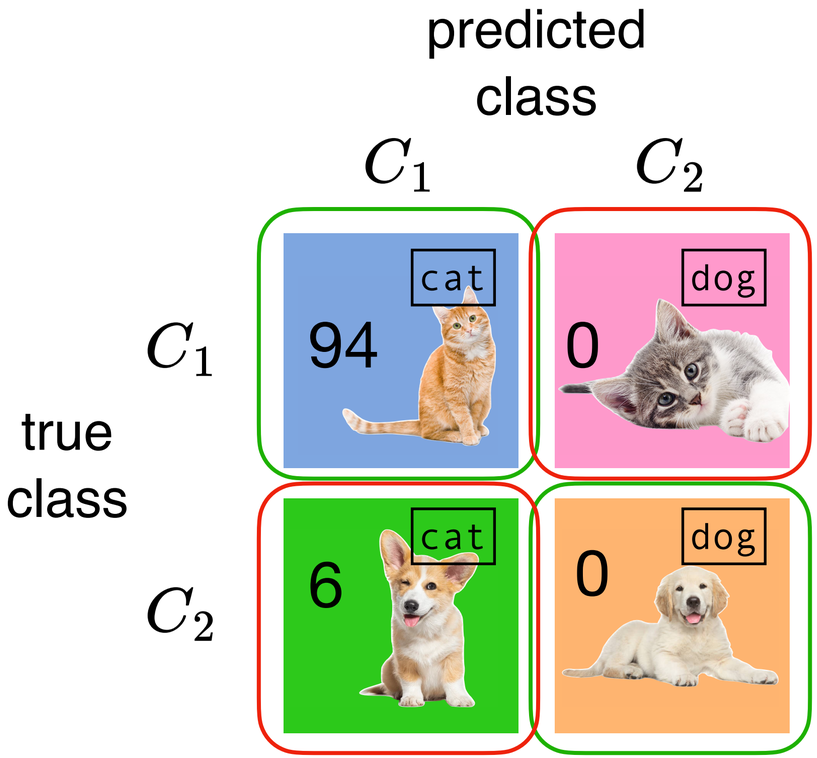


TP=94, TN=0, FP = 6, FN = 0

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN} = \frac{94}{100}$
- Recall$_{C_2}$: $\frac{TP}{P} = \frac{TP}{TP + FN} = \frac{0}{6}$
- Precision$_{C_2}$: $\frac{TP}{TP + FP} = \frac{0}{0}$

Metrics for classification - example

- Training set: 100 samples, 94 cats and 6 dogs. NN just outputs “cats”.



TP=94, TN=0, FP = 6, FN = 0

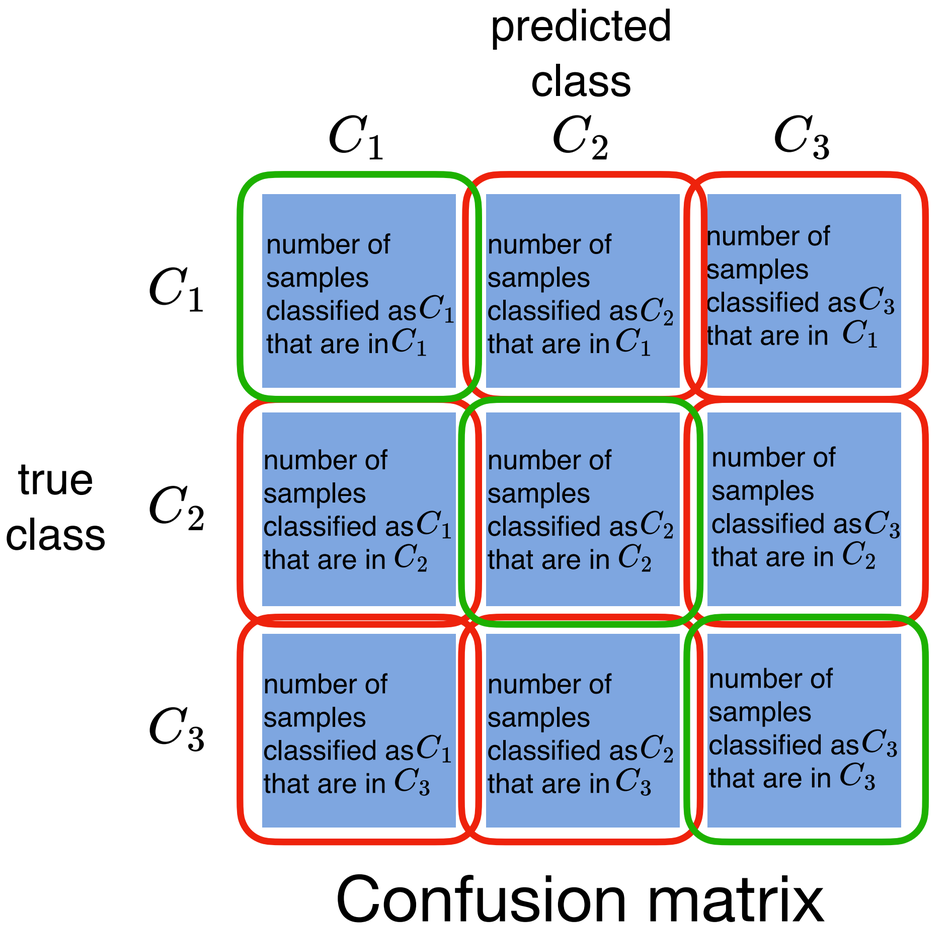
- Possible metric for unbalanced datasets:
- Balanced accuracy

$$\frac{1}{2}\left(\frac{TP}{TP+FN}+\frac{TN}{TN+FP}\right)=0.5$$

Balanced accuracy equal to 0.5 means either that the model is just guessing, or that it can predict well just one class.

Metrics for classification

- Multi-class classification, three classes, C_1, C_2 and C_3



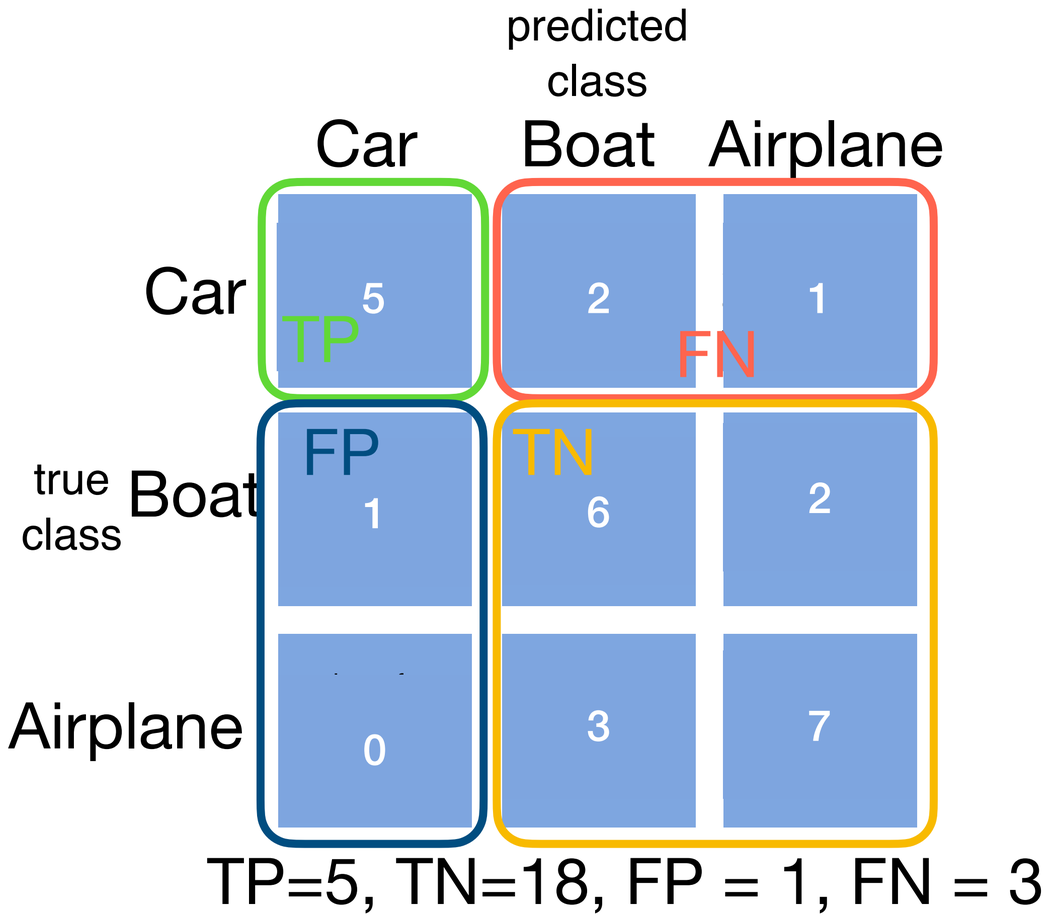
Confusion matrix

Correct classification

Wrong classification

Metrics for classification

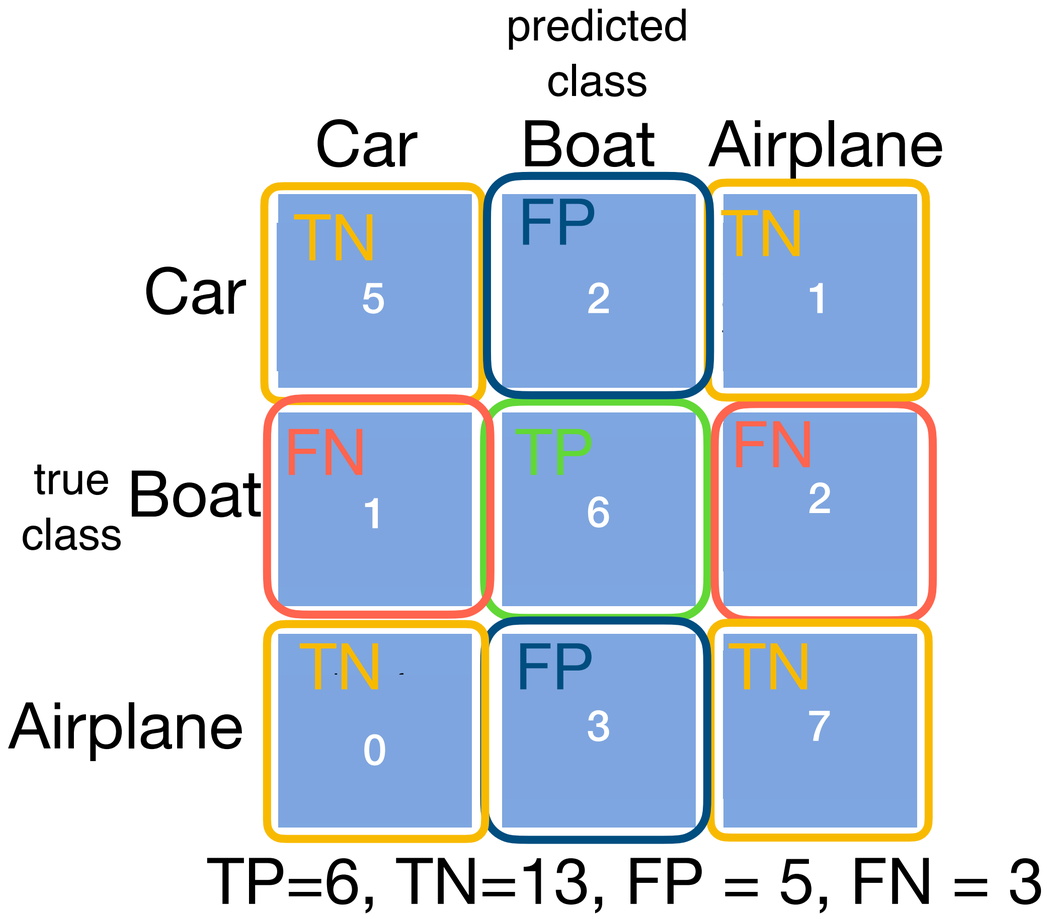
- Classes are: Car, boat, airplane. For each class, we can define the TP, FP, TN, FN, and accuracy, precision, recall and F1 score accordingly



TP, FP, TN, FN for class “car”.

Metrics for classification

- Classes are: Car, boat, airplane. For each class, we can define the TP, FP, TN, FN, and accuracy, precision, recall and F1 score accordingly

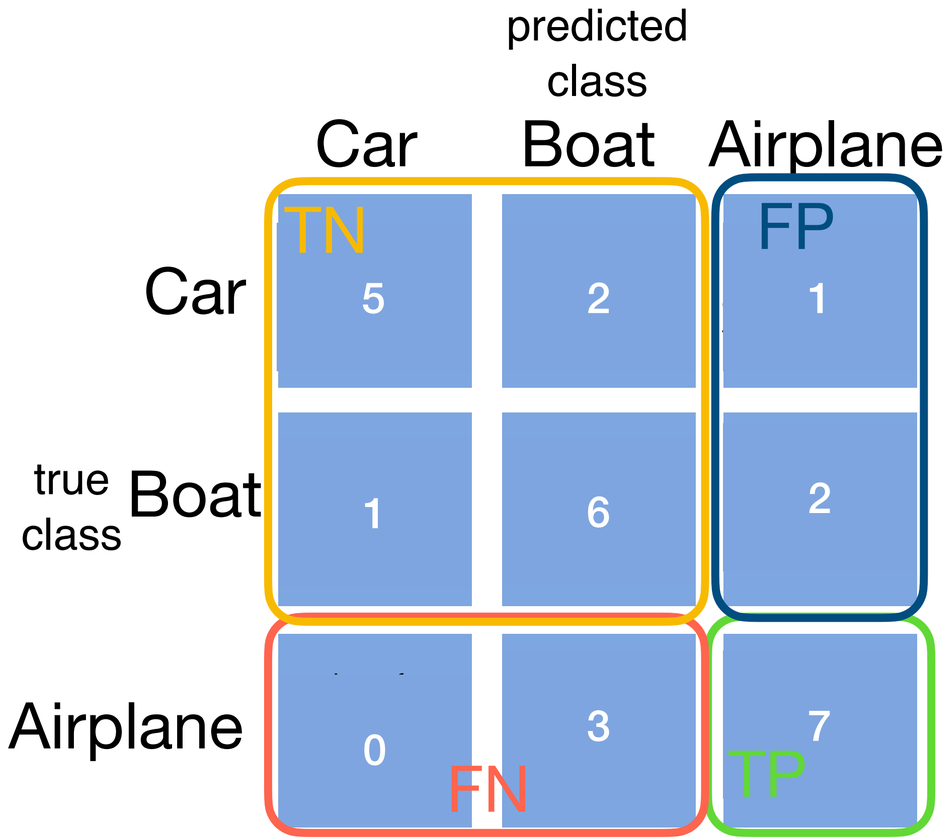


TP=6, TN=13, FP = 5, FN = 3

TP, FP, TN, FN for class “boat”.

Metrics for classification

- Classes are: Car, boat, airplane. For each class, we can define the TP, FP, TN, FN, and accuracy, precision, recall and F1 score accordingly

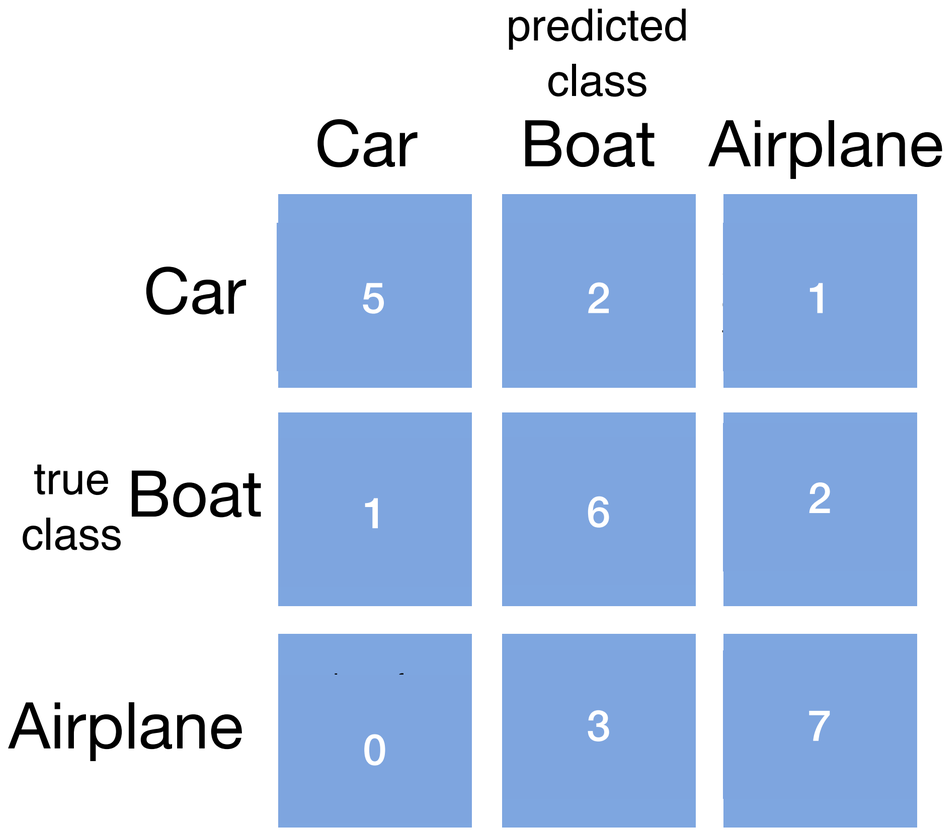


TP=7, TN=14, FP = 3, FN = 3

TP, FP, TN, FN for class “airplane”.

Metrics for classification

- Classes are: Car, boat, airplane. For each class, we can define the TP, FP, TN, FN, and accuracy, precision, recall and F1 score accordingly



$$\text{Accuracy} = \frac{\text{correct positives}}{\text{all samples}} = \frac{18}{27}$$

	Precision	Recall	F1-score
Car	0.83	0.62	0.18
Boat	0.55	0.67	0.15
Airplane	0.7	0.7	0.17

Metrics for classification

- Classes are: Car, boat, airplane. For each class, we can define the TP, FP, TN, FN, and accuracy, precision, recall and F1 score accordingly
- We then need to take the average of precision, recall, and F1-score for each class. There are two ways for doing that.

	Precision	Recall	F1-score
Car	0.83	0.62	0.18
Boat	0.55	0.67	0.15
Airplane	0.7	0.7	0.17

Micro average

$$P_\mu = \frac{TP_{\text{car}} + TP_{\text{boat}} + TP_{\text{airplaine}}}{TP_{\text{car}} + TP_{\text{boat}} + TP_{\text{airplaine}} + FP_{\text{car}} + FP_{\text{boat}} + FP_{\text{airplaine}}} = \frac{18}{27}$$

$$R_\mu = \frac{TP_{\text{car}} + TP_{\text{boat}} + TP_{\text{airplaine}}}{TP_{\text{car}} + TP_{\text{boat}} + TP_{\text{airplaine}} + FN_{\text{car}} + FN_{\text{boat}} + FN_{\text{airplaine}}} = \frac{18}{27}$$

$$P_M = \frac{P_{\text{car}} + P_{\text{boat}} + P_{\text{airplaine}}}{\text{number of classes}} = \frac{2.08}{3} = 0.69$$

$$R_M = \frac{R_{\text{car}} + R_{\text{boat}} + R_{\text{airplaine}}}{\text{number of classes}} = \frac{1.99}{3} = 0.66$$

Macro average

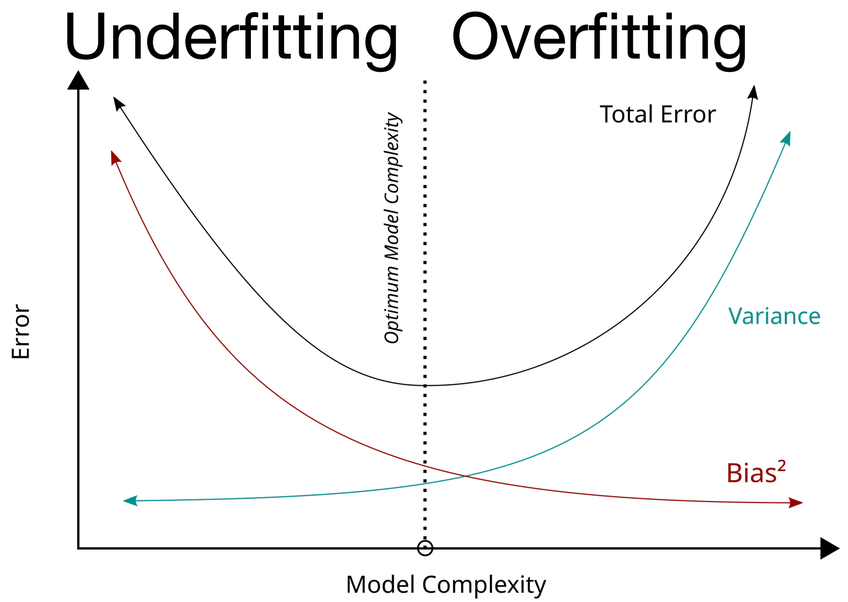
Bias

- Machine learning (ML) models are not inherently objective.
- Selection bias occur when the sample selection of the training set does not accurately reflect the target population.
- Historical bias occur when the data distribution changed during time, and the model is not adapted to cope with the new distribution.
    - Example: an IoT air quality sensor is installed outdoors and trains a local model to predict pollution spikes. The model is trained in the summer, when most of the town activities are shut.

Bias-Variance Tradeoff

- The bias–variance tradeoff describes the relationship between a model's complexity (i.e., number of hidden layers, number of nodes), the accuracy of its predictions, and how well it can make predictions on previously unseen data.

Underfitting: the model is not complex enough, it makes many prediction errors (biases) as it cannot model the complexity of relevant relations between features and target outputs



Overfitting: the model is too complex, it fitted the training data perfectly, missing the necessary generalisation capabilities. It makes many prediction mistakes as it sees different data than its training data (variance errors).