# Binary classification problems

## Logistic regression from a neural networks perspective

Filippo Biscarini
*Senior Scientist*
*CNR, Milan (Italy)*

Nelson Nazzicari
*Research fellow*
*CREA, Lodi (Italy)*

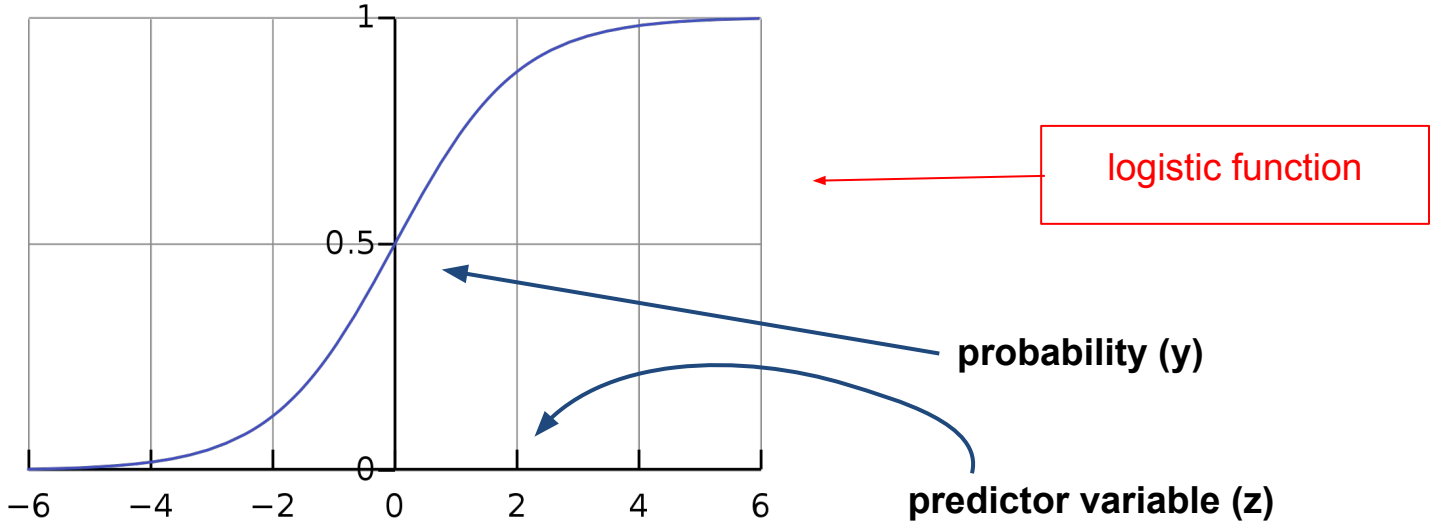
# Binary classification problems

- the response variable **y** is **qualitative** and takes up one of two values
- binary traits (e.g. cases/controls, resistant/susceptible, true/false, etc.)
- **y** = **label** (a.k.a. dependent variable)
- **X** = matrix of **features** (continuous, categorical)

# Binary classification problems

- the response variable **y** is **qualitative** and takes up one of two values
- binary traits (e.g. cases/controls, resistant/susceptible, true/false, etc.)
- **y** = **label** (a.k.a. dependent variable)
- **X** = matrix of **features** (continuous, categorical)
- we don't model the response directly, rather its **probability**: **P(y=1|x)**
- probabilities lie in [0,1] (not +/- infinity)

# Binary classification problems




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

# Logistic regression

- the logistic function is the basis for **logistic regression**
- **P(y=1|x)**
- **$Z = \beta_0 + \beta_1 x$**

$$P(y=1|x) = \sigma(z) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

we see here the familiar **model coefficients** (from linear regression) to be estimated and then used for predictions

# Logistic regression

- a little bit of algebra:

odds

$$\sigma(z) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \longrightarrow \frac{\sigma(z)}{1-\sigma(z)} = e^{\beta_0+\beta_1 x}$$

# Logistic regression

- a little bit of algebra:

odds

$$\sigma(z) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \longrightarrow \frac{\sigma(z)}{1-\sigma(z)} = e^{\beta_0+\beta_1 x}$$

log(odds): logit

$$log\left(\frac{\sigma(z)}{1-\sigma(z)}\right) = logit(\sigma(z)) = \beta_0 + \beta_1 x$$

# Logistic regression

- the **logit function** (log(odds)) is the **link function** between a linear expression of X and the probabilities of Y
- linear X expression ($\beta_0$+$\beta_1$x) → logit scale (continuous)
- logistic function: converts values on the logit scale back to probabilities

$$\begin{cases} logit(\sigma(z)) = \beta_0 + \beta_1 x \\ \sigma(\beta_0 + \beta_1 x) = P(y = 1|x) \end{cases}$$

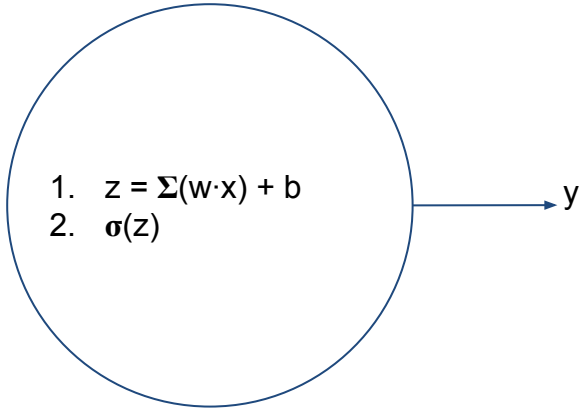
our objective!

# Logistic regression

- the **logit function** (log(odds)) is the **link function** between a linear expression of X and the probabilities of Y
- linear X expression ($\beta_0+\beta_1 x$) → logit scale (continuous)
- logistic function: converts values on the logit scale back to probabilities

$$\begin{cases} logit(\sigma(z)) = \beta_0 + \beta_1 x \\ \sigma(\beta_0 + \beta_1 x) = P(y = 1|x) \end{cases}$$

Looks familiar!




# Estimating the coefficients

**how do we obtain the model coefficients β?**

# Estimating the coefficients

**how do we obtain the model coefficients β?**

- we need to define a **loss function** and then minimise it

| observations | predictions |
| --- | --- |
| **y** | $\hat{y} = \sigma(\beta_0 + \beta_1 x)$ |

difference between observed and predicted values

# Estimating the coefficients

**how do we obtain the model coefficients β?**

- we need to define a **loss function** and then minimise it
- $\hat{y} = \sigma(z)$

$$J(\beta) = \text{loss}(\hat{y}, y) =$$
$$-\left(y \cdot log(\hat{y}) + (1-y) \cdot log(1-\hat{y})\right)$$

# Loss function for logistic regression

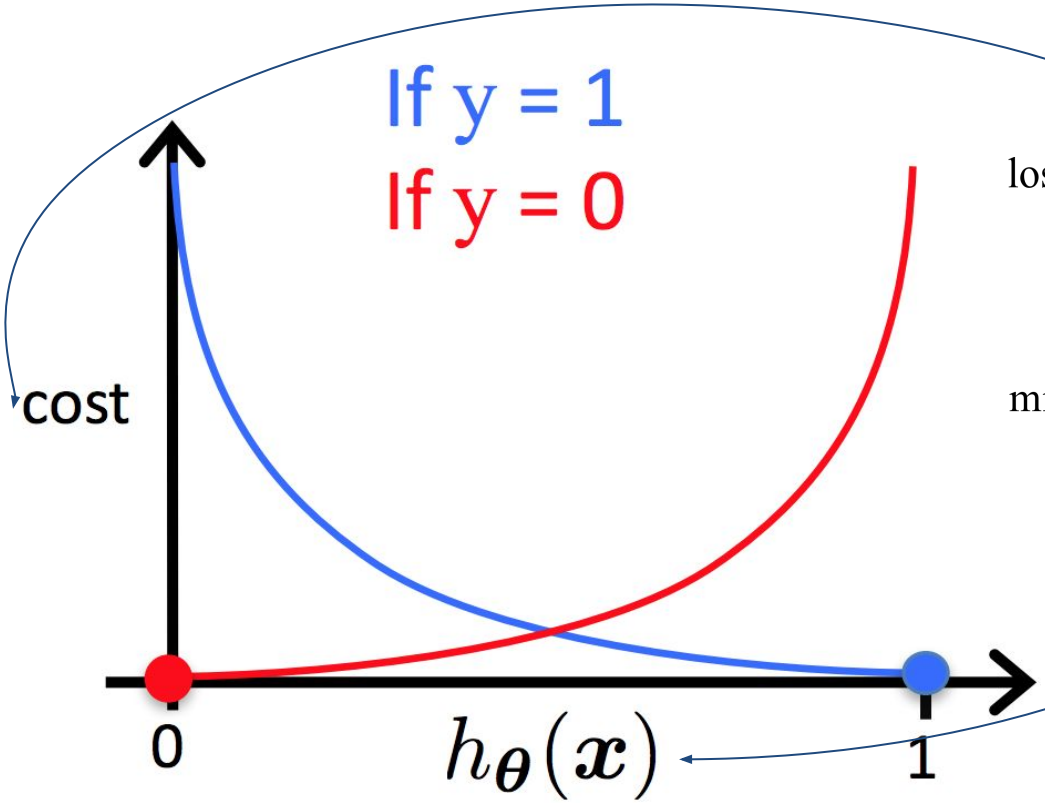
$$J(\beta) = \text{loss}(\hat{y}, y) =$$
$$-(y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y}))$$

<u>if y = 1</u>

- y_hat → 1, loss → 0
- y_hat → 0 (but y = 1), loss → infinity

the opposite holds if y = 0

# Cost/loss function for logistic regression




$$\text{loss}\,(\hat{y}, y) = J(\beta)$$

$$J(\beta) = -\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

$$\underset{\beta}{\text{minimize}}\, J(\beta)$$

From: https://datascience.stackexchange.com/questions/40982/logistic-regression-cost-function

# Minimising the cost function

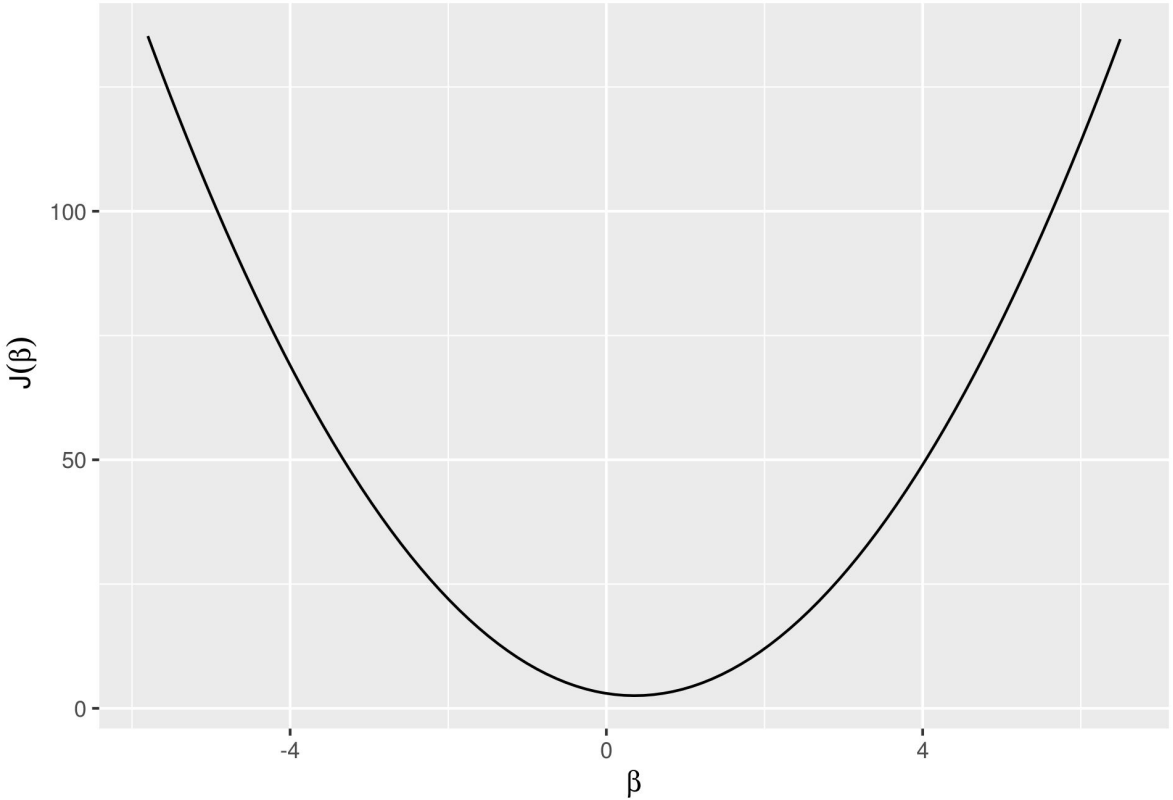
$$J(\beta) = -\left(y \cdot log(\hat{y}) + (1-y) \cdot log(1-\hat{y})\right)$$

$$\underset{\beta}{\text{minimize}}\, J(\beta)$$

# Minimising the cost function

- the defined cost function is convex
- can be minimised by **gradient descent**
- machine learning perspective: gradient descent is a general algorithm to solve models
- alternatively:
    - maximum likelihood
    - non-linear least squares

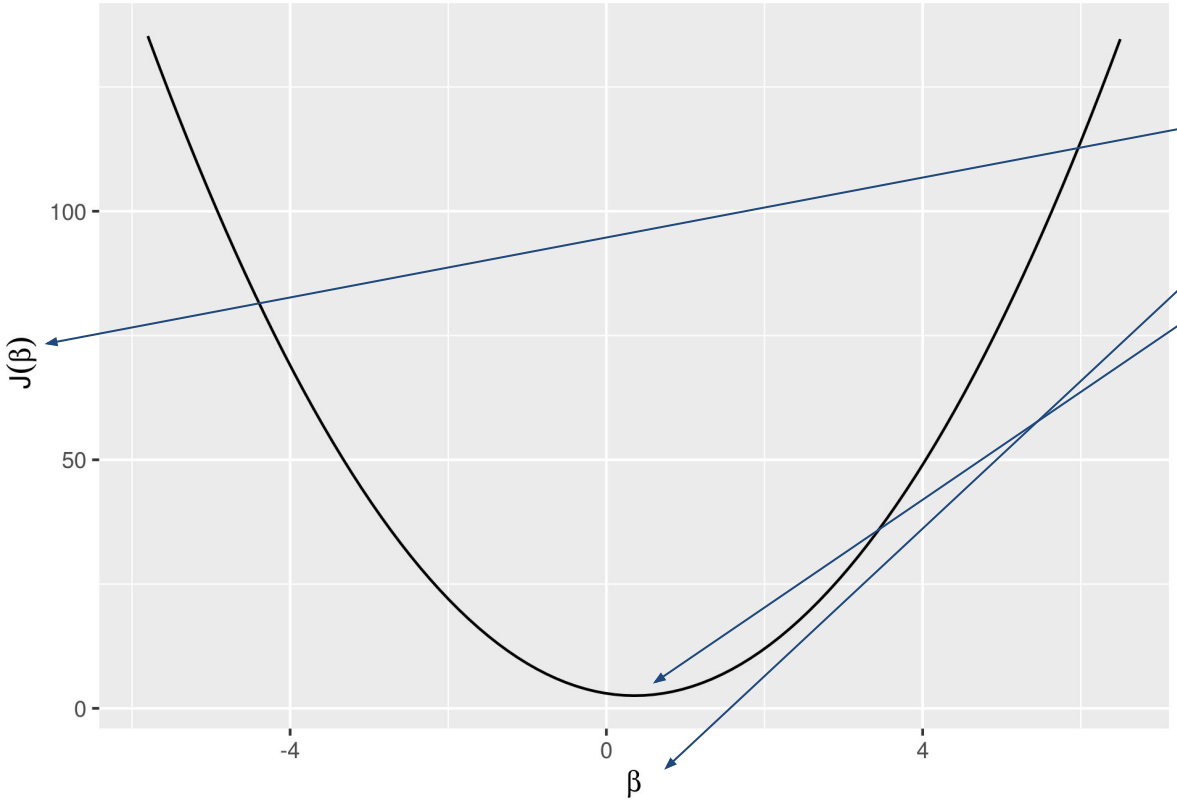
# Minimise the cost function




Simple logistic regression (1 parameter):

$$z = \beta \cdot x$$

# Minimise the cost function

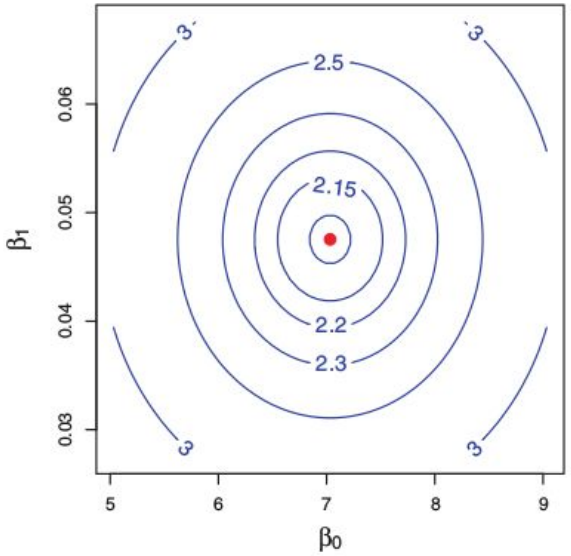


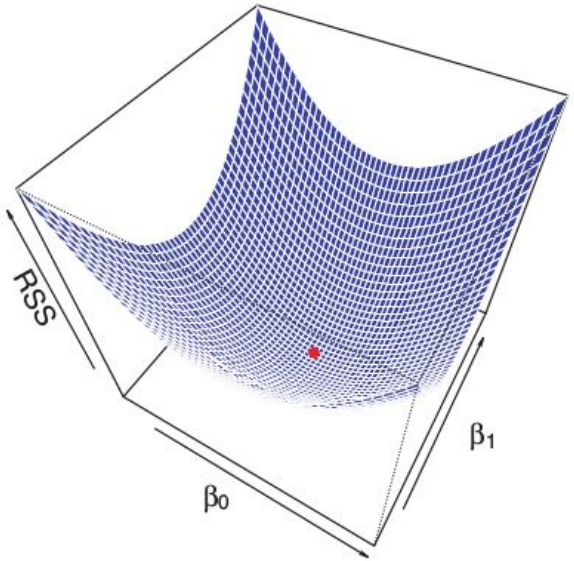

1. loss function
2. model parameters
3. minimum

Simple logistic regression (1 parameter):

$$z = \beta \cdot x$$

# Minimise the cost function




Multiple logistic regression (e.g. 2 parameters):

$$y = \beta_0 + \beta_1 \cdot x$$

Multiple logistic regression (> 2 parameters):

→ m-dimensional hyperspace

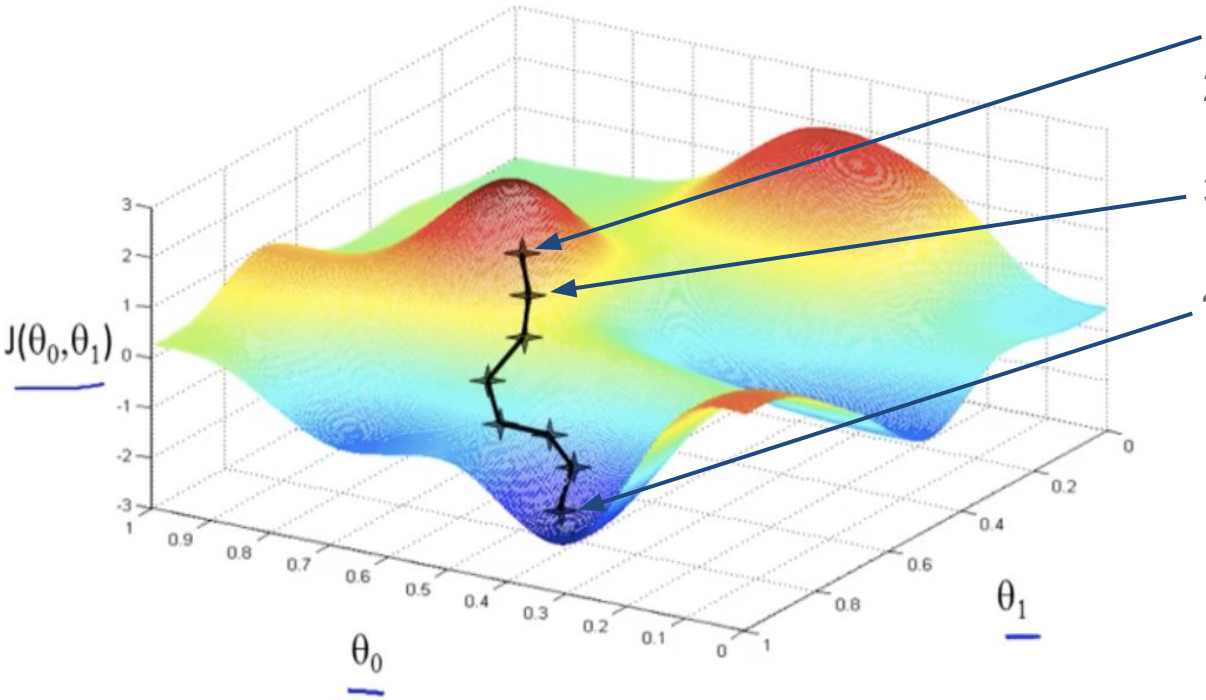
# Cost function: finding the minimum?

Gradient Descent:

$$\underset{\beta}{\text{minimize}}\, J(\beta)$$

1. Start with initial values for $\beta$ : (initialisation)
2. Change $\beta$ in the direction of reducing J($\beta$) : (descent)
3. Stop when the minimum is reached : (minimisation)

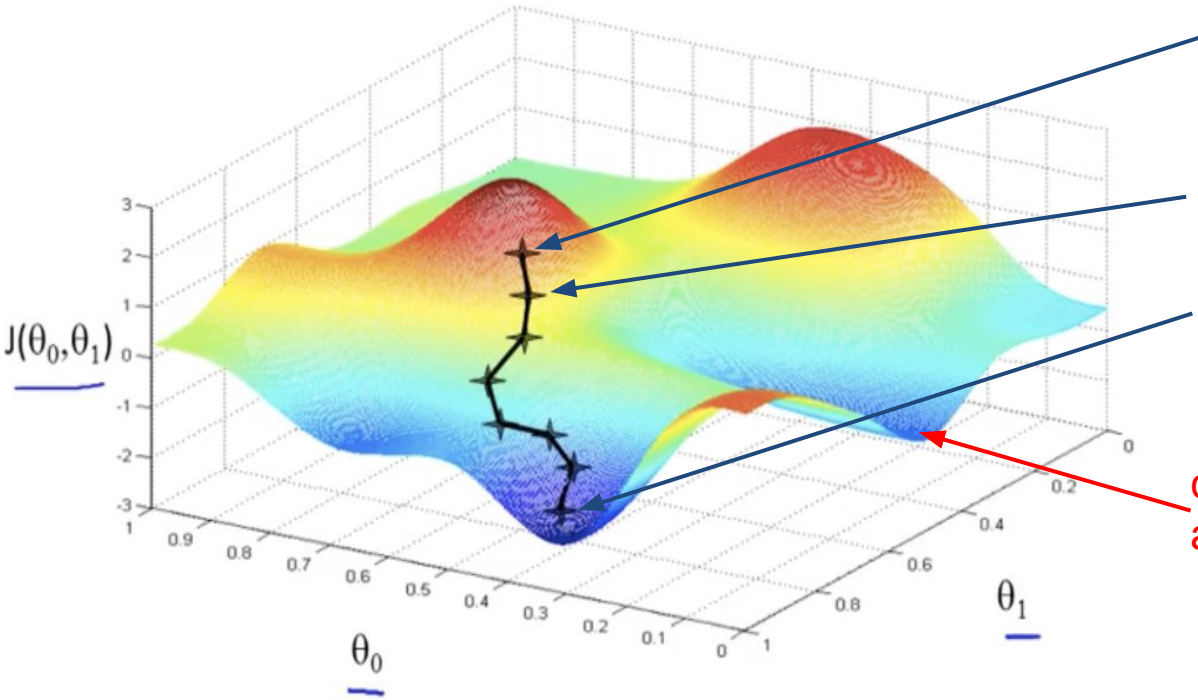
# Gradient descent




1. starting point (initialisation)
2. find the steepest direction around the starting point
3. take one step in this direction
4. repeat until convergence (local minimum)

Source: Andrew Ng
https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635

# Gradient descent

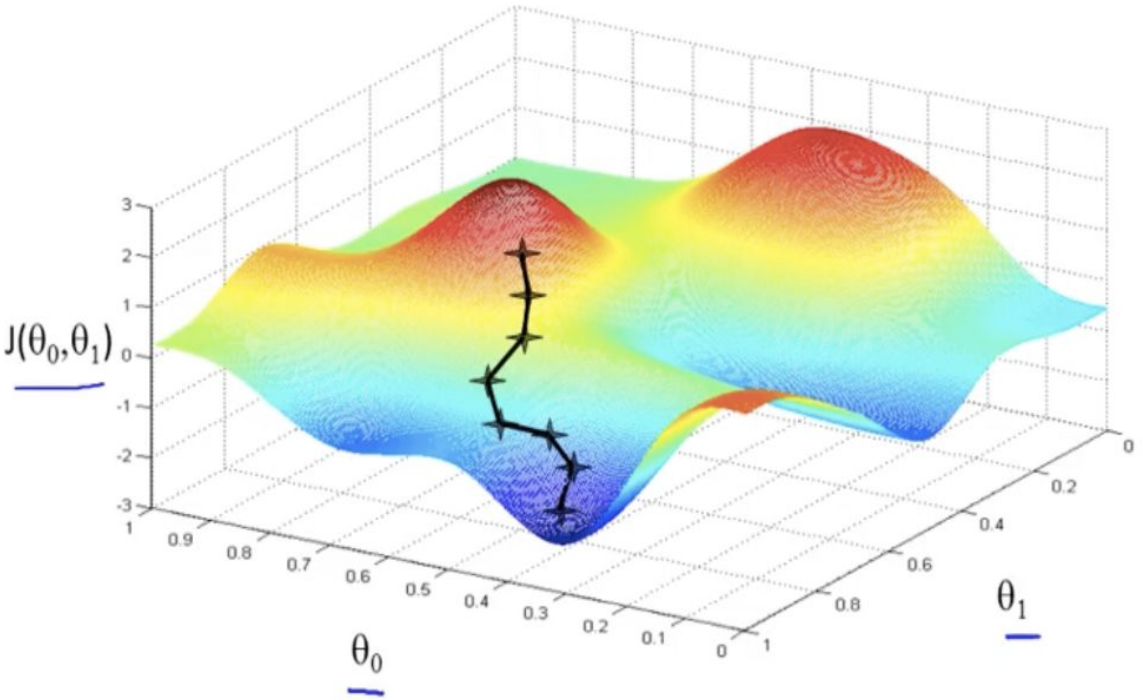



1. starting point (initialisation)
2. find the steepest direction around the starting point
3. take one step in this direction
4. repeat until convergence (local minimum)

changing initialisation may lead to a different local minimum!

Source: Andrew Ng
https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635

# Gradient descent




**How do we find the steepest direction?**

Source: Andrew Ng
https://medium.com/@DBCerigo/on-why-gradient-descent-is-even-needed-25160197a635

# Gradient descent

new value for $\beta j$

old value for $\beta j$

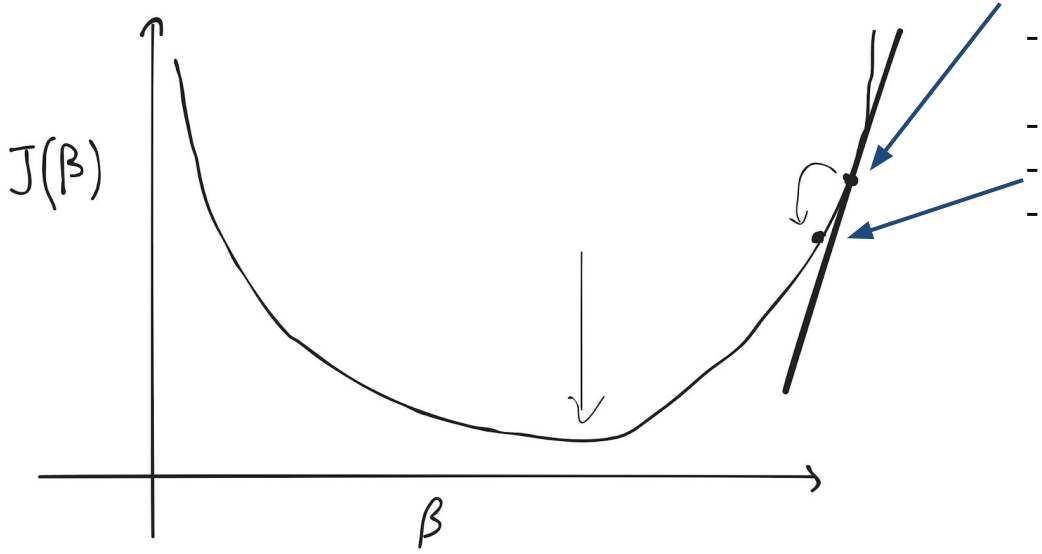
$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

assignment operator

learning rate (size of the steps)

Partial derivative of J(**β**) with respect to $\boldsymbol{\beta}_j$

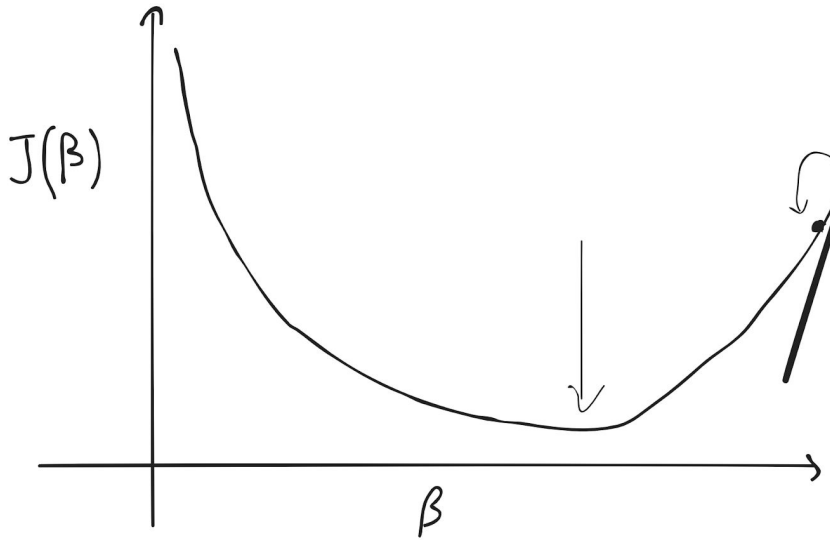
# Gradient descent




- starting point (initial value for $\beta$)
- calculate the (partial) derivative in that point
- → positive slope
- update the value for $\beta$
- repeat



# Gradient descent



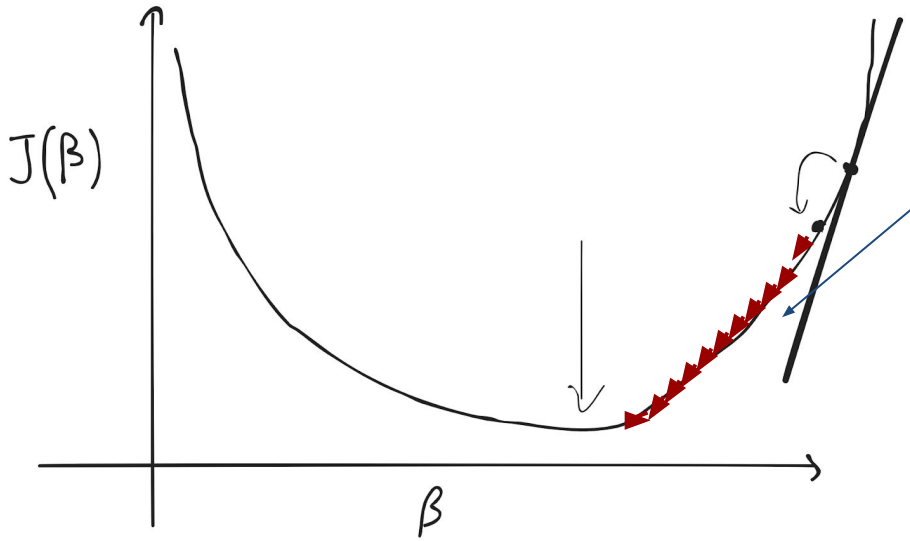

- starting point (initial value for $\beta$)
- calculate the (partial) derivative in that point
- → positive slope
- **update** the value for $\beta$
- repeat

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$

- positive slope → reducing the value of $\beta$ (and the other way around)



# Gradient descent

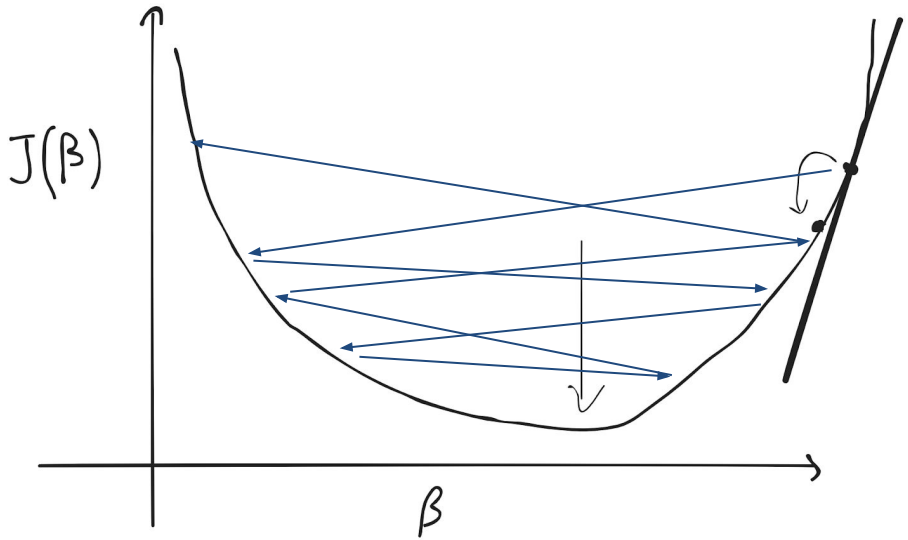



- $\alpha$ controls the size of the updating step
- small $\alpha \rightarrow$ slow descent

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$



# Gradient descent



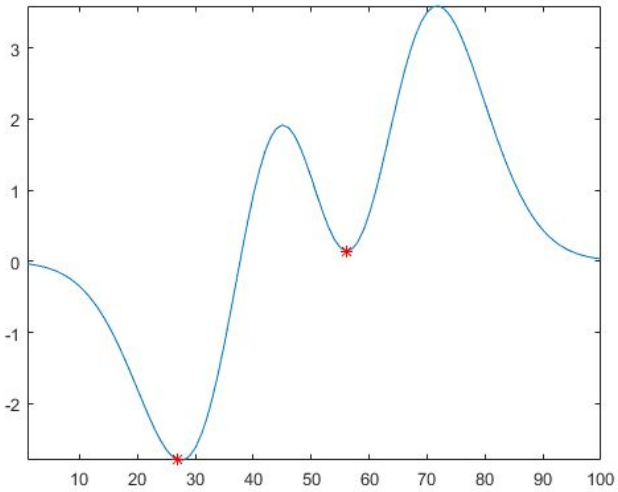

- $\alpha$ controls the size of the updating step
- large $\alpha \rightarrow$ overshooting: failure to converge

$$\beta_j := \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\beta)$$



# Gradient descent - recap

- general method to **solve machine learning models** (e.g. multiple linear regression)
- optimise (minimise) the cost function → **optimiser**
- importance of the **learning rate**
- local minimum → **momentum**




# Logistic regression - recap

1) $z = w \cdot x + b$

2) $\hat{y} = \sigma(z)$

3) $J(w) = -\left(y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})\right)$

introduce **w** (weight) as parameter (+ b): NN notation

# Logistic regression - recap

1) $z = w \cdot x + b$
2) $\hat{y} = \sigma(z)$
3) $J(w)$

forward propagation

# Logistic regression - recap

backward propagation

$\frac{\partial \hat{y}}{\partial z}$

$\frac{\partial J}{\partial \hat{y}}$

$$z = w \cdot x + b$$

$$\hat{y} = \sigma(z)$$

$$J(w)$$

forward propagation

$$-\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

# Logistic regression - recap

backward propagation

$\frac{\partial \hat{y}}{\partial z}$

$\frac{\partial J}{\partial \hat{y}}$

$$z = w \cdot x + b$$

$$\hat{y} = \sigma(z)$$

$$J(w)$$

forward propagation

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}$$

**chain rule**

$$-\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right)$$

# Logistic regression - recap

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}$$

our objective!

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z} \cdot x_1$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z} \cdot x_2$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z}$$

$$w_1 := w_1 - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial J}{\partial w_2}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

updating step

# Take away message

- **back propagation** is a way (algorithm) to calculate partial derivatives (of the cost function with respect to the parameters) easily and efficiently
- partial derivatives are then used to **update** the values of the **parameters**
- in this way **gradient descent** can work to **minimise** the **cost function** and **estimate** the best values for the **parameters**
- this is very important to efficiently learn the weights (parameters) of deep neural networks

# Binary classification metrics

# Binary classification: measuring performance

- the most common metric to measure the performance of a binary classifier is the **error rate**:

$$\frac{1}{n}\sum_{i=1}^{n} I(y \neq \hat{y})$$

# Confusion matrix

| | | True observation | |
|---|---|---|---|
| | | 1 | 0 |
| **Prediction** | 1 | TP | FP |
| | 0 | FN | TN |

- **FPR** = FP/(FP+TN)
- **FNR** = FN/(FN+TP)
- TER = (FN+FP)/(FN+FP+TN+TP)

# Confusion matrix

| | | True observation | |
|---|---|---|---|
| | | 1 | 0 |
| **Prediction** | 1 | TP | FP |
| | 0 | FN | TN |

Not only total error rate!

- **FPR** = FP/(FP+TN)
- **FNR** = FN/(FN+TP)
- TER = (FN+FP)/(FN+FP+TN+TP)

# Logistic regression

- demonstration 04a
- exercise 04a.1

→ code_04a_logistic_regression.ipynb