Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

T-MachLe

6. Classification systems - logistic regression

Jean Hennebert
Andres Perez Uribe

Plan - Supervised Learning for Classification tasks

6.1 Recaps

6.2 Using Logistic Regression

6.3 Training Logistic Regression with Gradients

6.4 Link between Logistic Regression and Artificial Neural Networks

Practical Work 6

6.1 Recap

Classification task

k-nn

Bayes

Analyse the following training log

Activity

- What can we say looking at this evolution of the accuracy along the epochs ?
- What can we do if we have as "improvement budget" (engineering time):
  - 0h more
  - 15h more
  - 150h more

Supervised learning - classification tasks

A **classification task** maps inputs **x** to a finite set of discrete outputs **y**. The outputs are the class labels corresponding to the different categories we want to predict.

- Usually classes are **mutually exclusive**, i.e. only one label is output of the system.
  - However, some systems are said **multi-label** when a given input x belongs to more than one class.
- 2-class systems are sometimes called **detection** or **verification** systems, where the objective is to answer yes/no questions
  - Biometric example: *Is this the face of Sheldon? Is the identity verified?*
  - Warship example: *Is a torpedo going to hit us? Is a torpedo detected?*

Previously seen classification approaches

- We have seen a first intuitive classification system: the k-NN
  - Simple computation of the k nearest neighbours and majority voting on the k labels of the neighbours
  - Disadvantage: which distance metric, tuning of k, cpu load for large training sets, sensitivity to priors (unbalanced classes)
- We have seen the Bayesian approach:
  - Compute for each class k

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})}$$

  - Select class k with

$$\underset{k}{\operatorname{argmax}}\ P(C_k|\mathbf{x})$$

Previously seen classification approaches

- Bayesian systems are mathematically correct, inclusion of priors as a separate term
- Bayesian systems are said “generative” :
  - We need to model likelihoods and priors for each class, i.e. we have a computation for each class
  - We may actually generate data from the likelihoods ans priors

likelihood
“probability of observing x given class j”

a priori probability
“probability of class j”

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})}$$

a posteriori probability
“probability of class j given observation x”

evidence = probability of x
“…unconditional to any class…”

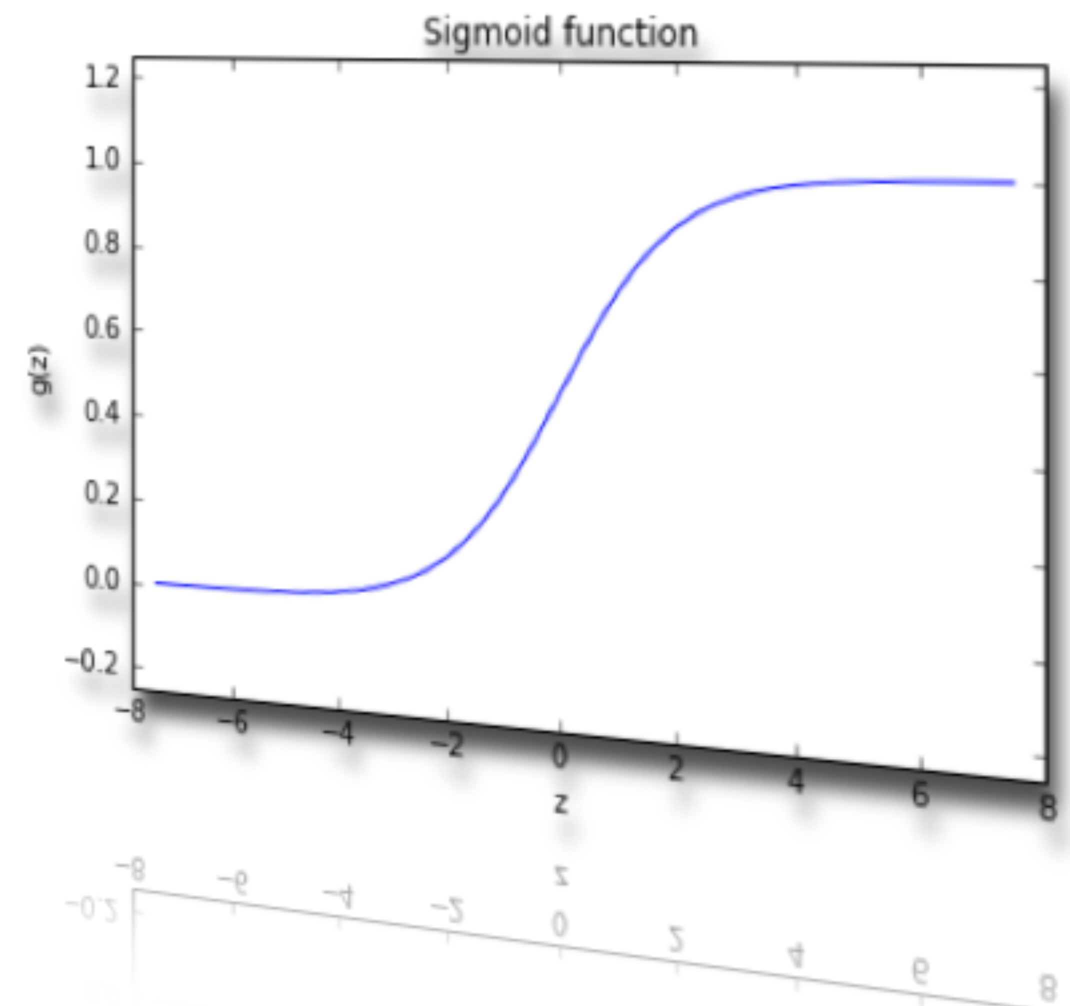
- Difficulty of Bayesian systems:
  - How to model the likelihoods? Histogram, Gaussian models, Gaussian mixture models?
  - It suffers quite hard from the curse of dimensionality

6.2 Using Logistic Regression

Intuition

Sigmoid

Decision boundary



Logistic regression - intuition

Disclaimer: there is a full probabilistic framework behind logistic regression - here we focus on ML and intuition.

- We assume a two class problem $\widehat{y} \in \{0, 1\}$
- We want our hypothesis function to output
  - 1 for the positive class, 0 for the negative class
  - the decision border will be when $\hat{y} = h_\theta(\mathbf{x}) = 0.5$
  - For this reason we will use a family of function that shows the property $0 \leqslant h_\theta(\mathbf{x}) \leqslant 1$

- We have to look at logistic regression as an extension of linear regression
  - We had before $h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots = \mathbf{x}\theta^T$
  - Now we move to $h_\theta(\mathbf{x}) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots) = g(\mathbf{x}\theta^T)$
  - The function $g()$ is a sigmoid that shows the property $0 \leq g() \leq 1$

With X = [1.0, x1, x2, ...] (1xD)

Sigmoid function

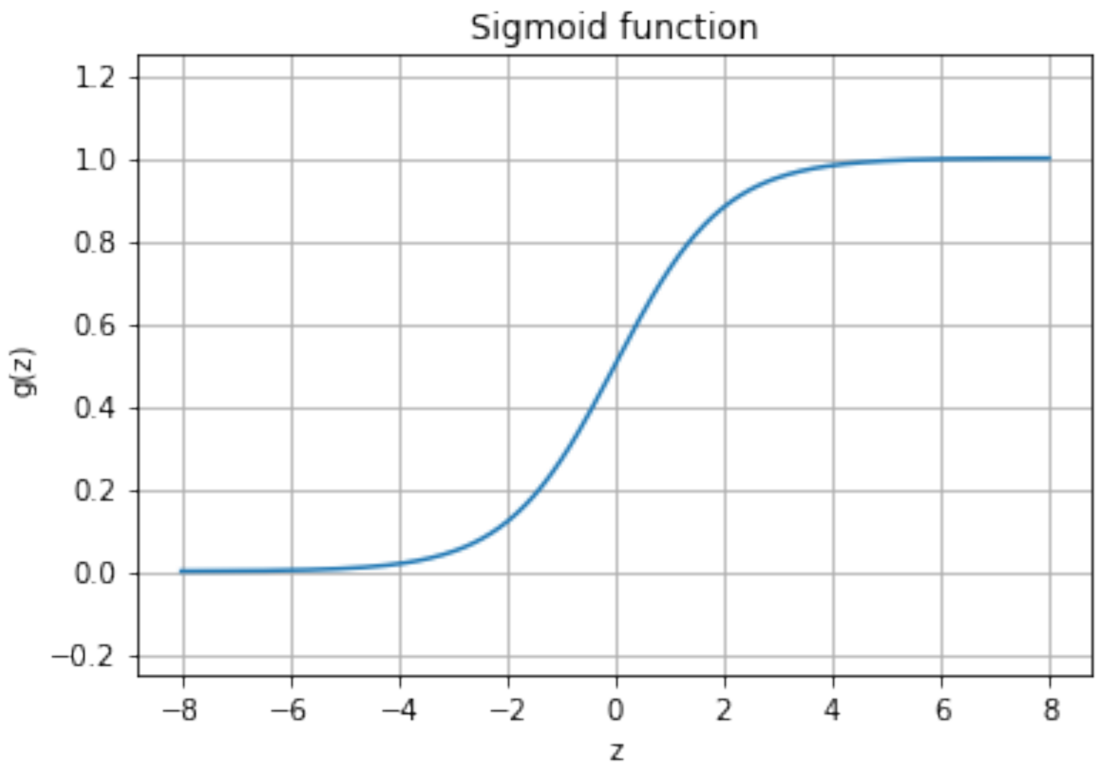
- The sigmoid is defined with

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

- If we define $z = \mathbf{x}\theta^T$
- We have for the hypothesis function

$$h_\theta(\mathbf{x}) = g(z) = g(\mathbf{x}\theta^T)$$
$$= \frac{1}{1+e^{-\mathbf{x}\theta^T}}$$

- This function saturates
  - to 1 when $\mathbf{x}\theta^T \gg 0$
  - to 0 when $\mathbf{x}\theta^T \ll 0$
- Decision boundary for $z = \mathbf{x}\theta^T = 0$

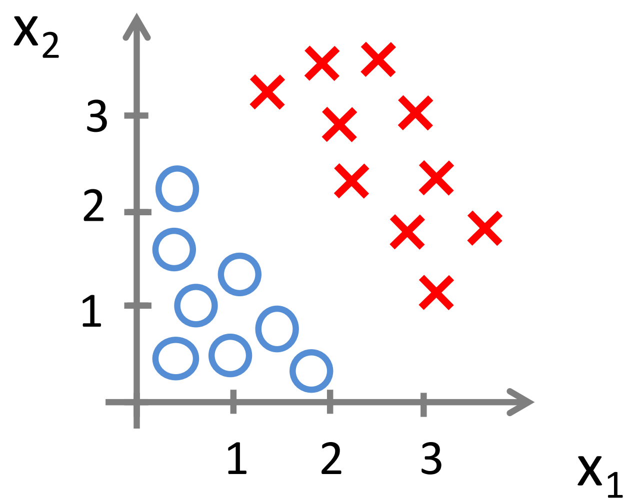


- A nice property of the sigmoid function is in the form of the derivative:

$$\boxed{g'(z) = g(z)(1-g(z))}$$

See development in notation file on Moodle

Logistic regression - decision boundary



To take the classification decision, we will say

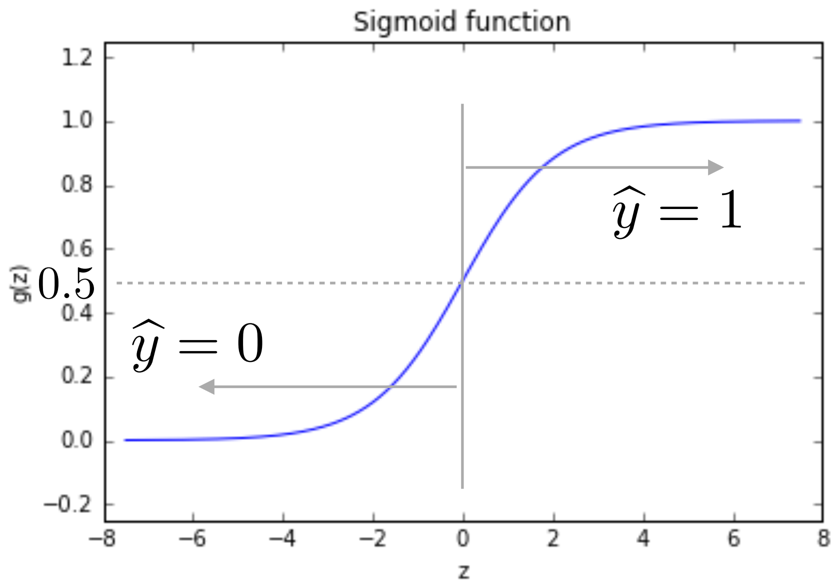
- class 1 when $h_\theta(\mathbf{x}) \geq 0.5$
- class 0 when $h_\theta(\mathbf{x}) < 0.5$

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

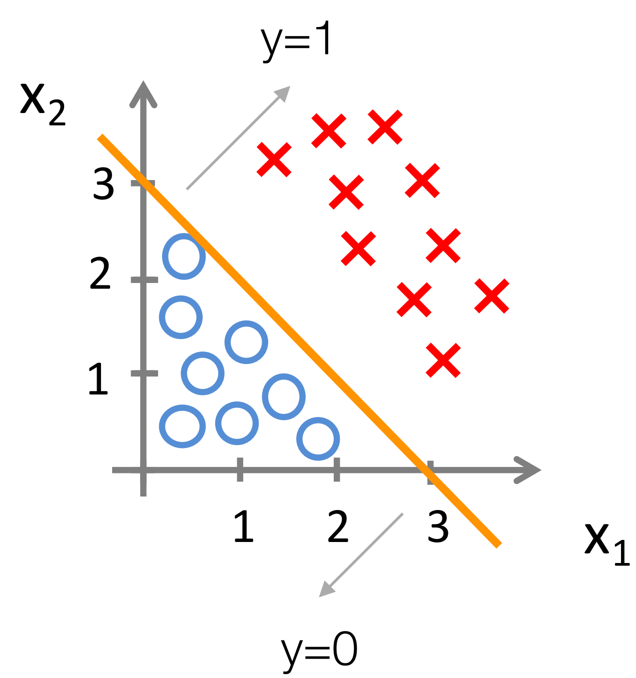
Let's assume the training gives us: $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

- Predict $\widehat{y} = 1$
  when $h_\theta(\mathbf{x}) \geq 0.5$
  or when $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$

$$-3 + x_1 + x_2 \geq 0$$



Logistic regression - decision boundary



$$h_\theta(x) = g(\underbrace{\theta_0 + \theta_1 x_1 + \theta_2 x_2}_{z})$$

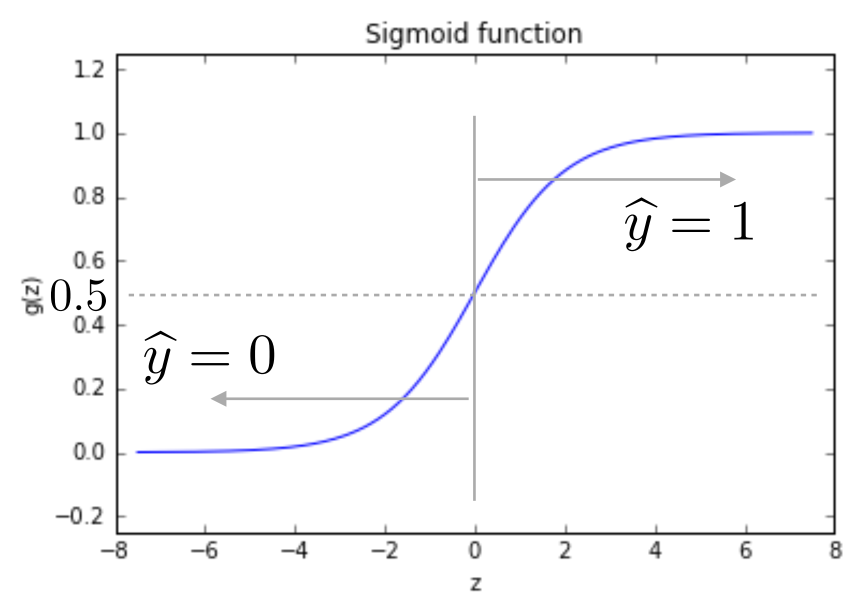
Let's assume the training gives us: $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$

- Predict $\widehat{y} = 1$

  when $h_\theta(\mathbf{x}) \geq 0.5$

  or when $\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$

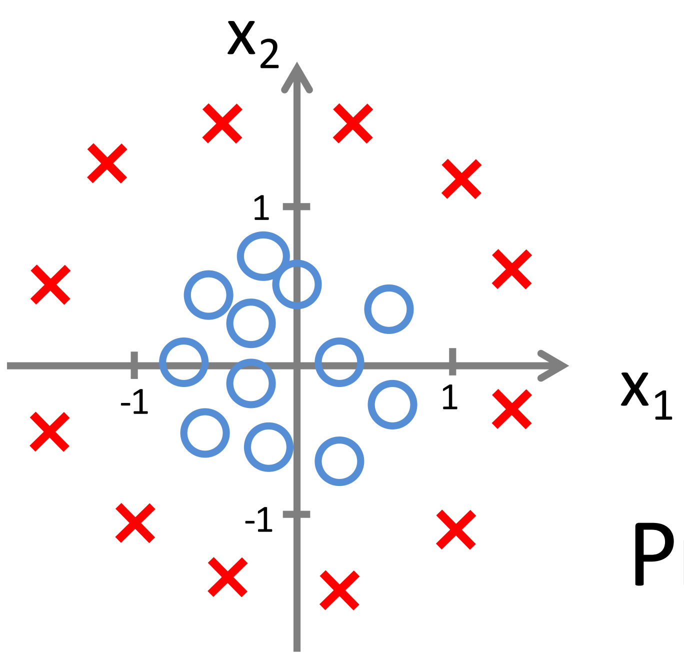
  $-3 + x_1 + x_2 \geq 0$



The boundary can be plot when z = 0

Logistic regression - non linear decision boundary

Let's assume the training gives us:

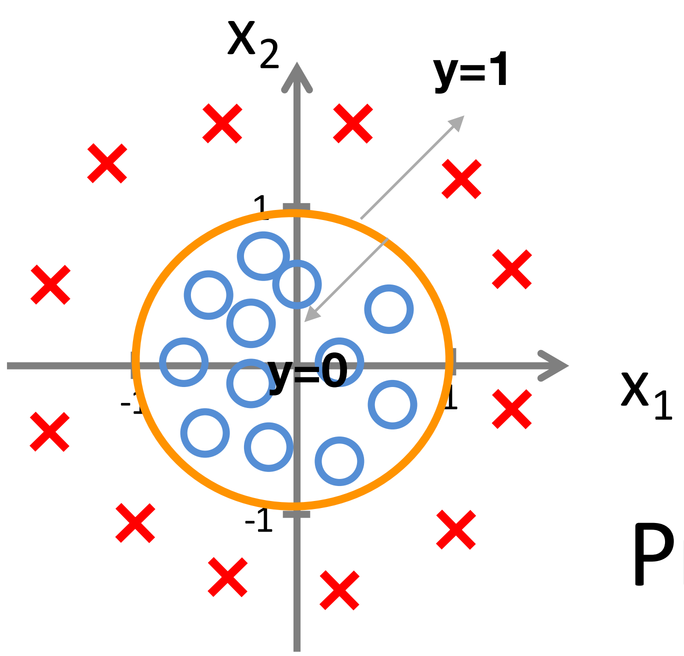


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

Logistic regression - non linear decision boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Let's assume the training gives us:

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

x1	*x1^2*	…	…	*y*
26	26*26	…	…	B
37	1369	…	…	A
57	3249	…	…	A
48	2304	…	…	B
…	…	…	…	…

Remember: as for linear regression, compute a new column $x_2 = x_1^2$ and treat it as for a linear decision!

How to discover the thetas?

- Model fitting for logistic regression has no known closed-form solution
- Iterative procedures need to be used such as
  - Maximum likelihood with Newton procedure
  - Iteratively reweighed least squares (IRLS)
  - **Gradient approaches** (see next section)

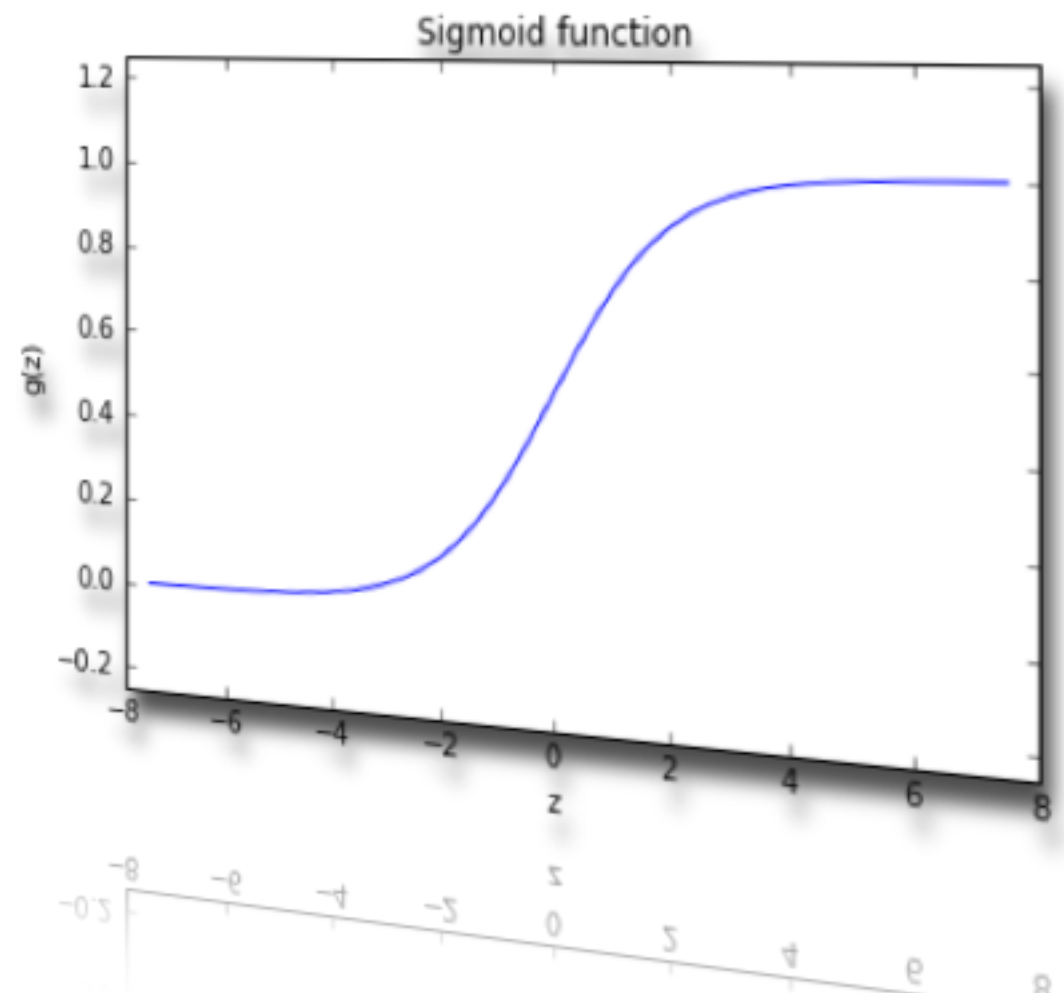
This is different to linear regression that has a closed form solution (see chapter on linear regression).

6.3 Training Logistic Regression with Gradients

Objective function

Gradient Ascent

Example and implementation considerations



Logistic regression - objective function

- Now, given the logistic regression model, how do we fit the parameters θ for it?
  - The cost function $J(\theta)$ of chapter 4 could be used here but we have two difficulties with it:
    - first, it is more designed for prediction problems than classification,
    - second it will induce us more difficulties in the computation of the gradient.
- For classification tasks, we usually prefer to use an objective function that will maximise the number of correct classifications.
- Looking back to Bayes rule, we want to have $h_\theta(\mathbf{x})$ to be an estimator of the posterior probability $P(C_k|\mathbf{x})$

$$P(C_1|\mathbf{x_n};\theta) \hat{=} P(\hat{y}_n = 1|\mathbf{x_n};\theta) = h_\theta(\mathbf{x}_n)$$
$$P(C_2|\mathbf{x_n};\theta) \hat{=} P(\hat{y}_n = 0|\mathbf{x_n};\theta) = 1 - h_\theta(\mathbf{x}_n)$$

Logistic regression - math development

- For a given training samples

$$P(y_n = 1|\mathbf{x_n};\theta) = h_\theta(\mathbf{x_n})$$
$$P(y_n = 0|\mathbf{x_n};\theta) = 1 - h_\theta(\mathbf{x_n})$$
$$P(y_n|\mathbf{x_n};\theta) = h_\theta(\mathbf{x_n})^{y_n}(1 - h_\theta(\mathbf{x_n}))^{1-y_n}$$

This is a mathematical trick to express the computation for all positive and negative classes. When in C_1 then $y_n = 1$, when in C_2 then $y_n = 0$

- For the whole set $\overrightarrow{y}$ of *N* training samples, assuming their independence

$$P(\vec{y}|X;\theta) = \prod_{n=1}^{N} P(y_n|\mathbf{x_n};\theta)$$
$$= \prod_{n=1}^{N} h_\theta(\mathbf{x_n})^{y_n}(1 - h_\theta(\mathbf{x_n}))^{1-y_n}$$

According to Bayes rule, we want to maximise the a posteriori probability for all samples - so this will become our perf function.

- We want to maximise this “performance” or “objective” function: before we had a cost function (loss function) to minimise…

$$\boxed{J(\theta) = P(\vec{y}|X;\theta) = \prod_{n=1}^{N} P(y_n|\mathbf{x_n};\theta)}$$

But finding the derivative of this function is difficult. Also, a product of values between 0 and 1 will give a veeeeery small value, probably below the limit of float representations. For these reasons, we use again a mathematical trick: use the log() of this formula.

Logistic regression - mathematical development

- We can <u>also</u> *maximise* any monotonic increasing function.

  The *log* is a monotonic increasing function and it will simplify the derivation we will have to do later (the product becomes a sum, the exponent becomes a product)

$J(\theta)$ = "**Objective**" function

$$J(\theta) = \frac{1}{N} \log P(\vec{y}|X;\theta) \quad = \frac{1}{N} \log(\prod_{n=1}^{N} P(y_n|\mathbf{x}_n;\theta))$$
$$= \frac{1}{N} \sum_{n=1}^{N} y_n \log h_\theta(\mathbf{x_n}) + (1-y_n) \log(1-h_\theta(\mathbf{x_n}))$$

- After few mathematical development (see notation file)

$$\frac{\partial J(\theta)}{\partial \theta_i} = \frac{1}{N} \sum_{n=1}^{N} (y_n - h_\theta(\mathbf{x}_n)) x_{n,i}$$

Target value

Gotten output

Gradient ascent principle

1. Start with some initial θ 's (for example random or null)
2. Visit the full training set to compute new values of the θ's augmenting $J(\theta)$
3. Loop in 2 until convergence

A visit of the training set is called an **epoch**

- The new values of θ 's are chosen according to the "gradient" of $J(\theta)$, i.e. in the direction of the slope.

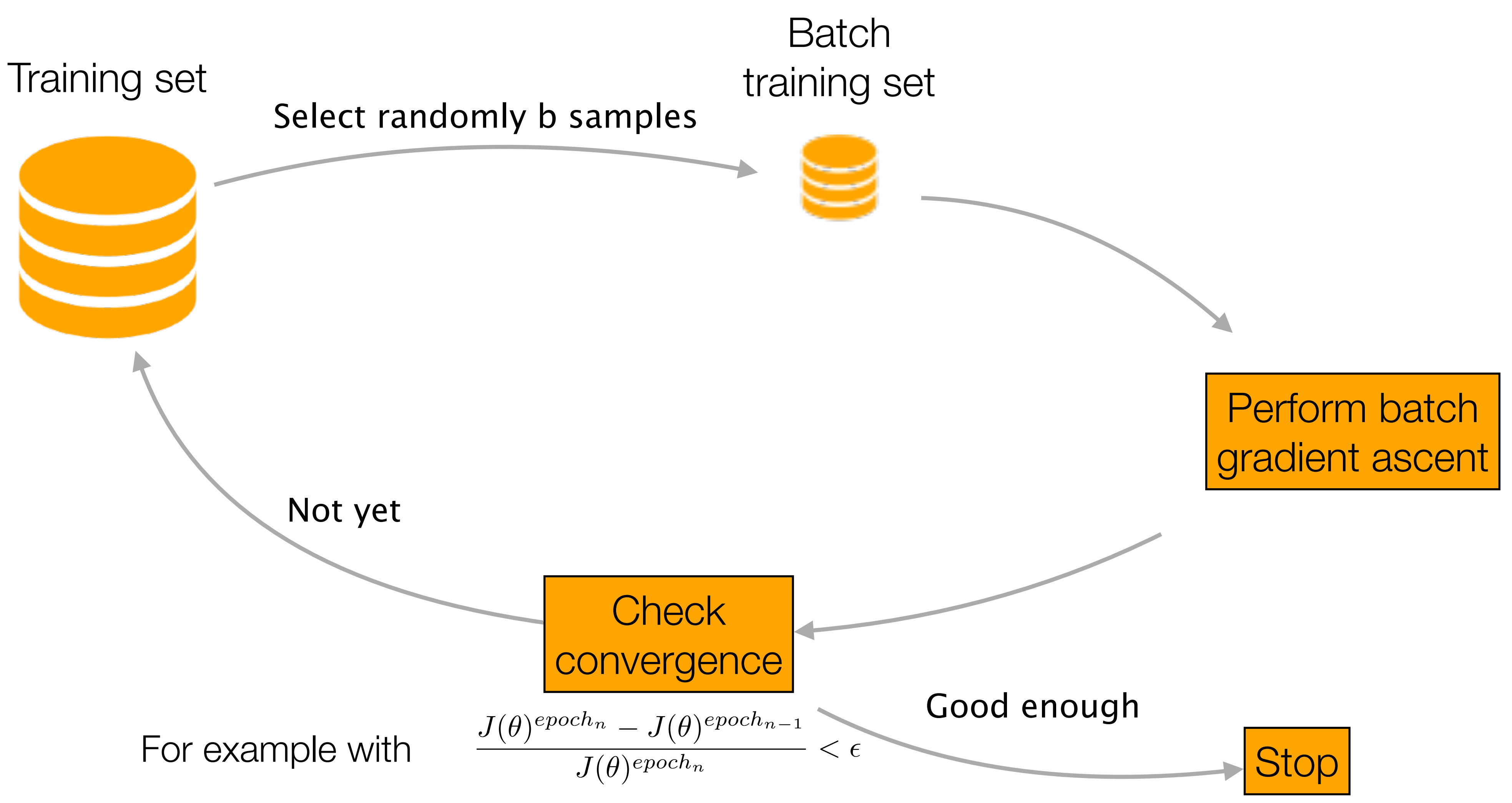
Logistic regression - gradient ascent

- The update rule then states that the parameter value θ_i is updated to the previous value plus the step α times the sum over all training data of the product of the difference between the target y_n and the gotten output $h_\theta(\mathbf{x}_n)$ by the training coefficient $x_{n,i}$.

$$\theta_i \leftarrow \theta_i + \alpha \frac{1}{N} \sum_{n=1}^{N} \underbrace{(y_n - h_\theta(\mathbf{x}_n))}_{\text{“error signal”}} x_{n,i}$$

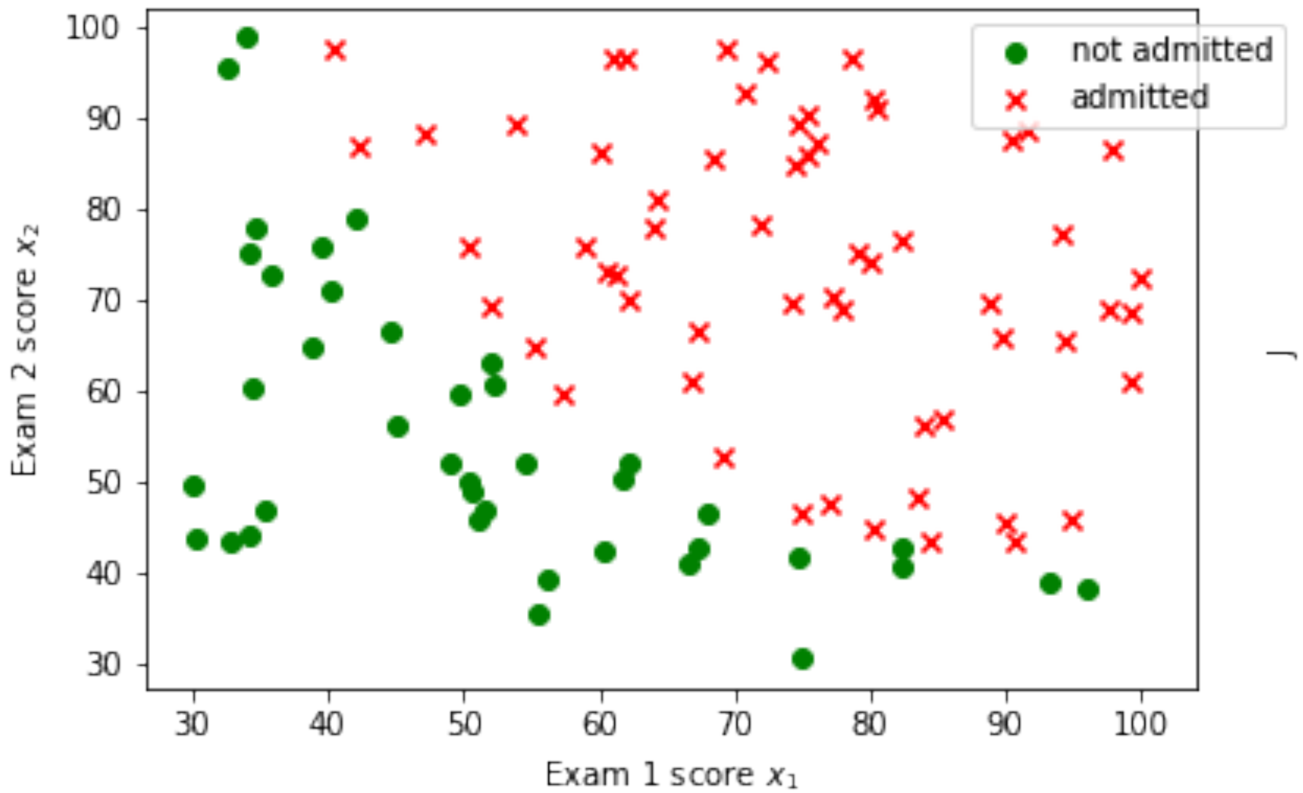
Logistic regression - gradient ascent

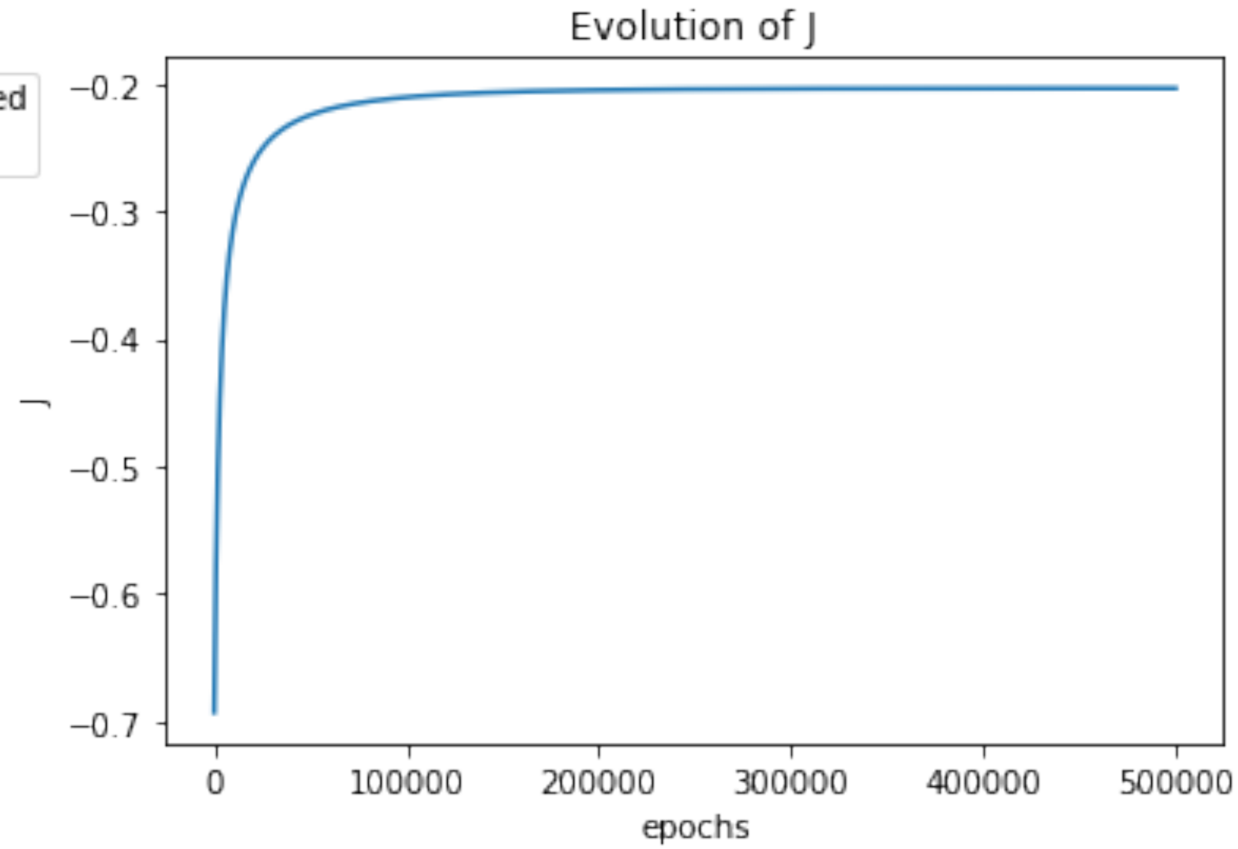
- The same considerations can be done as in the case of gradient descent
  - The alpha needs to be tuned
    - too large = oscillation around the maximum
    - too small = too slow to converge
  - We can build the gradient ascent with: full batch, stochastic, mini batch



For example with $\frac{J(\theta)^{epoch_n} - J(\theta)^{epoch_{n-1}}}{J(\theta)^{epoch_n}} < \epsilon$

Example - student dataset see PW






```
def gradientAscent(X, y, learning_rate, num_epoch):  # X has shape (N,D), y has shape (N,)
    N = X.shape[0]       # number of samples
    D = X.shape[1]       # dimensions
    theta = np.zeros(D) # init thetas to some values - in theory it can be anything
                        # but values at zeros or close to zeros may help convergence

    ... TO COMPLETE

    return theta, J
```

```
theta, J = gradientAscent(X, y, 1e-3, 500000)
```

Implementation considerations

- Always try to have implementations able to process whole data sets X.

$$X = \begin{pmatrix} 1 & x_{1,1} & & \dots & & x_{1,D} \\ 1 & & \ddots & & & \\ 1 & \vdots & & x_{n,d} & & \vdots \\ 1 & & & & \ddots & \\ 1 & x_{N,1} & & \dots & & x_{N,D} \end{pmatrix}$$

- Rely on broadcasting to do that.
- For $h_\theta(\mathbf{X})$:

```
def hypothesis(X,theta):
    # X has shape (N,D) and theta has shape (D,).
    # The dot product is then broadcasted to all samples in X.
    return ... TO COMPLETE  # array of shape (N,)
```

Implementation considerations

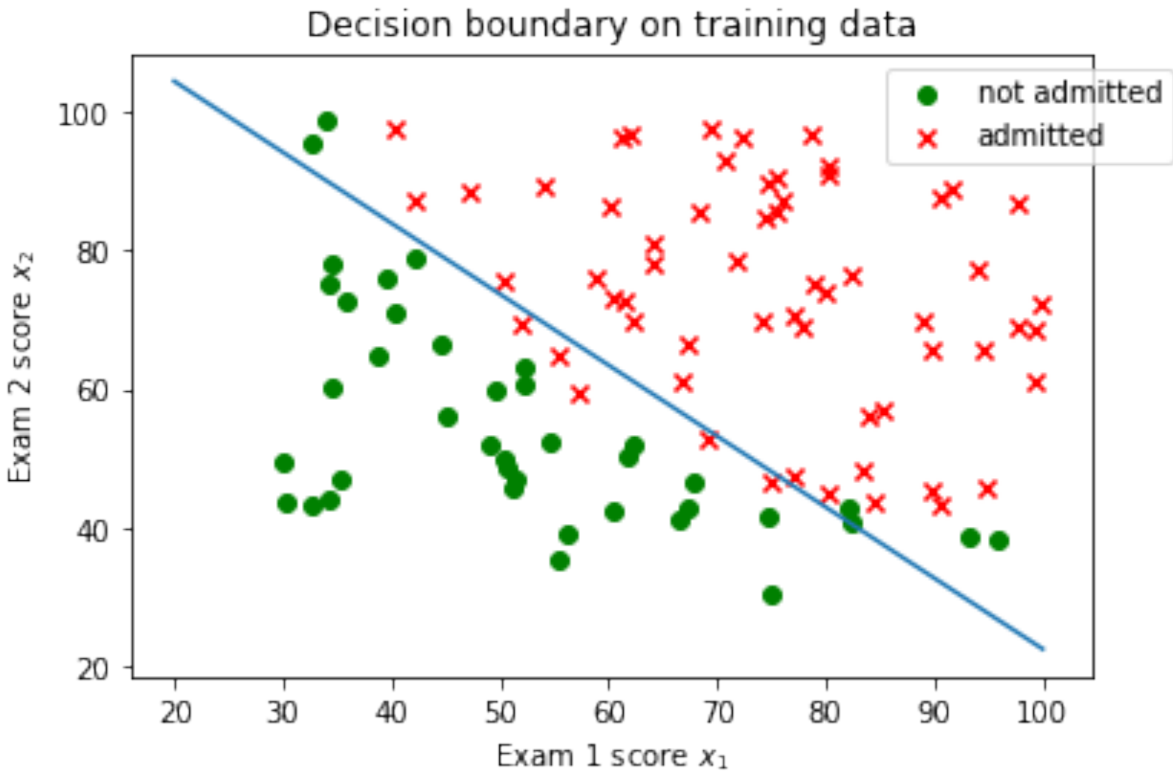
- For the computation of the objective function J, we should avoid to compute log(0). A solution is to add a small epsilon inside of the log:

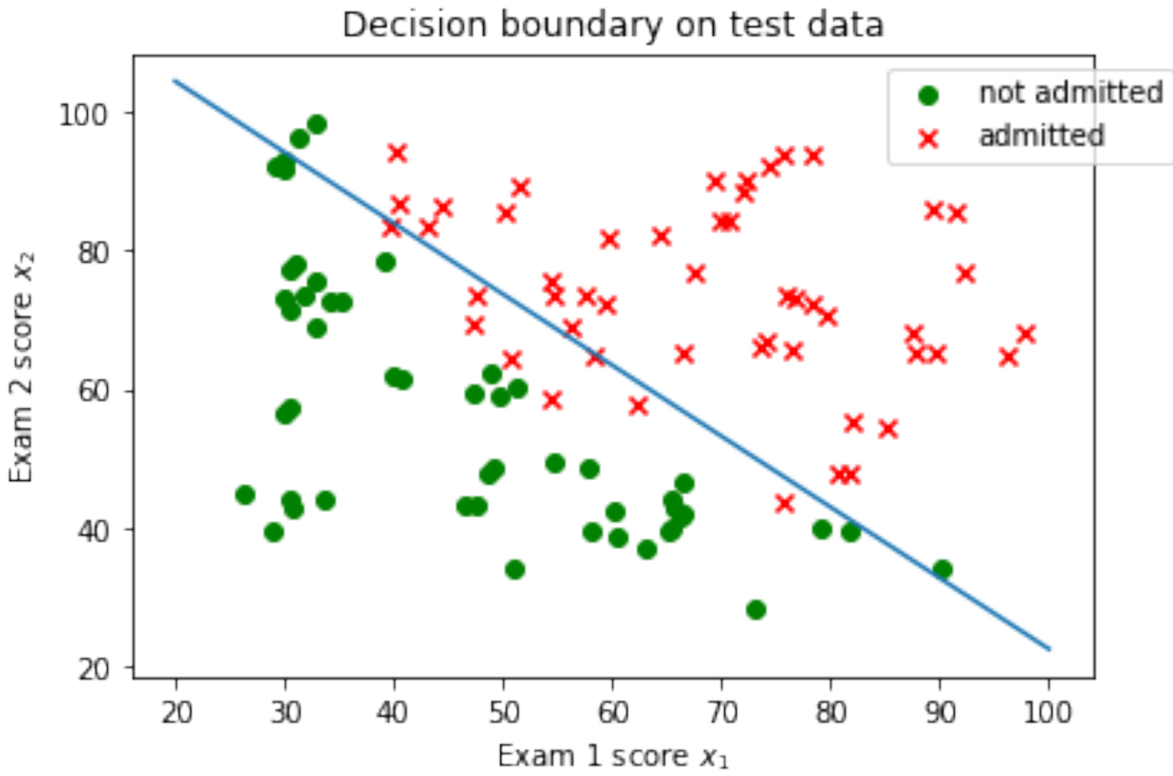
```
def objective(X,y,theta):    # X has shape (N,D), y has shape (N,), theta has shape (D,)
    epsilon = 1e-6
    h = hypothesis(X,theta) # h has shape (N,)
    N = X.shape[0]
    tmp = y * np.log(h + epsilon) + (1-y) * np.log(1 - h + epsilon)
    return np.sum(tmp)/N
```

In which situations do we risk to have a log(0)?

Example - student dataset see PW

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

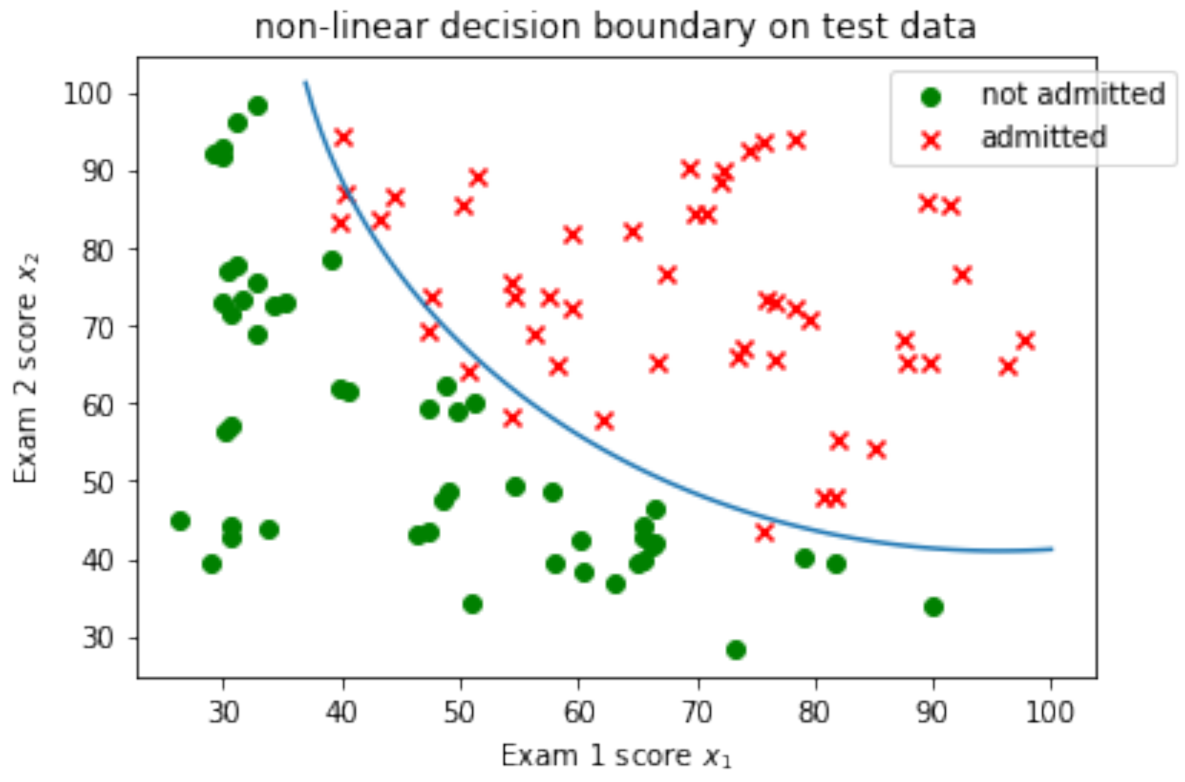





```
# correct    :  89
# missed     :  11
error rate :  11.00 %
```

Example - student dataset see PW

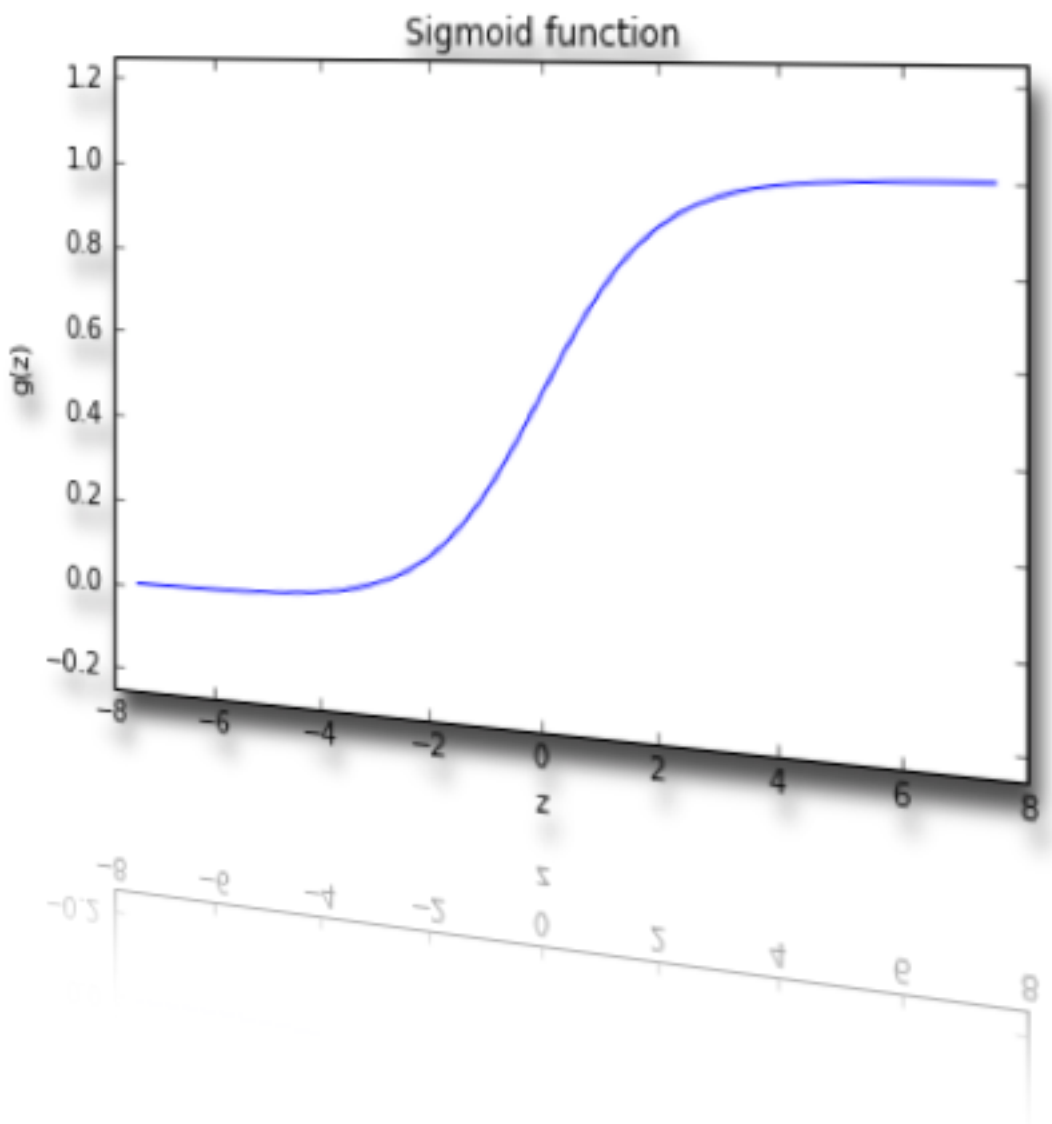
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$



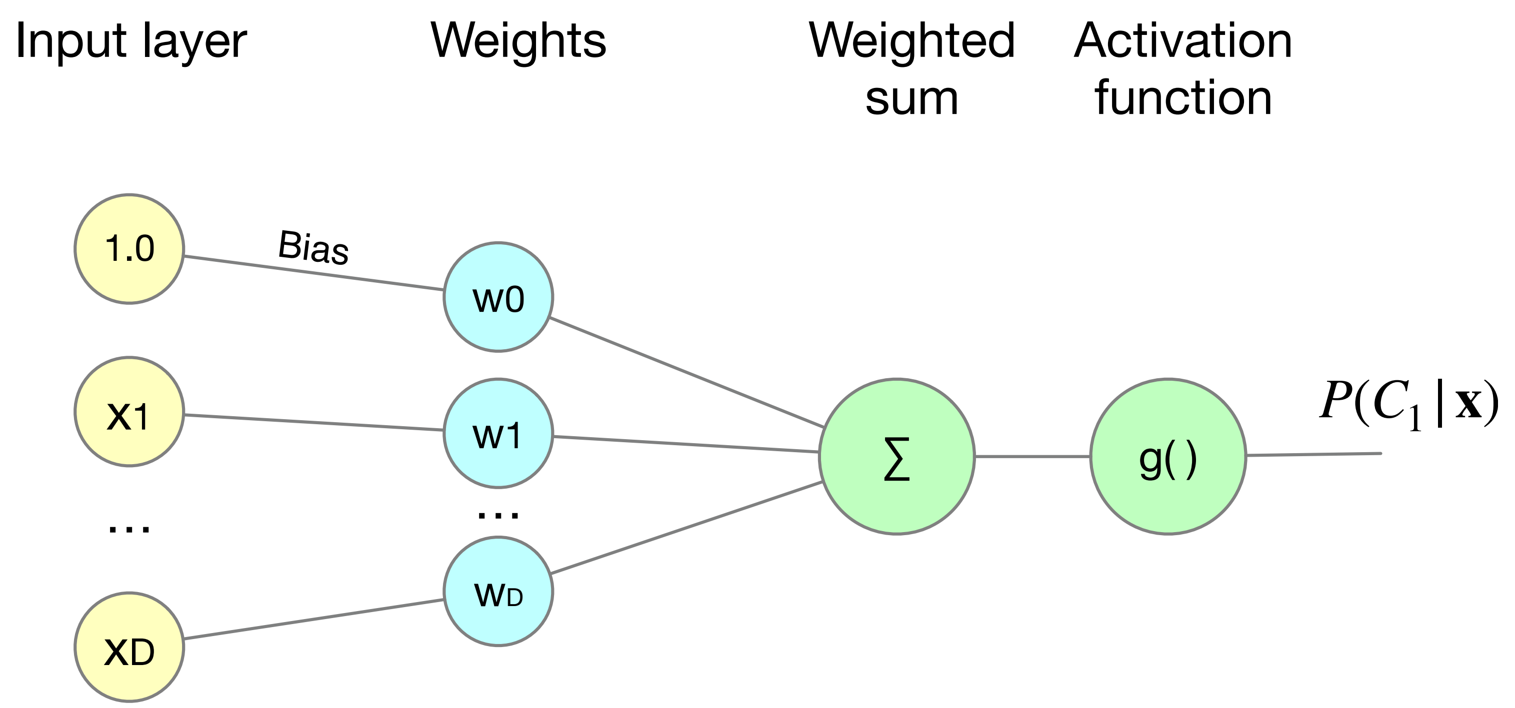

```
# correct   :  94
# missed    :  6
error rate  :  6.00 %
```

6.4 Logistic Regression and ANNs

Computational graph schematic



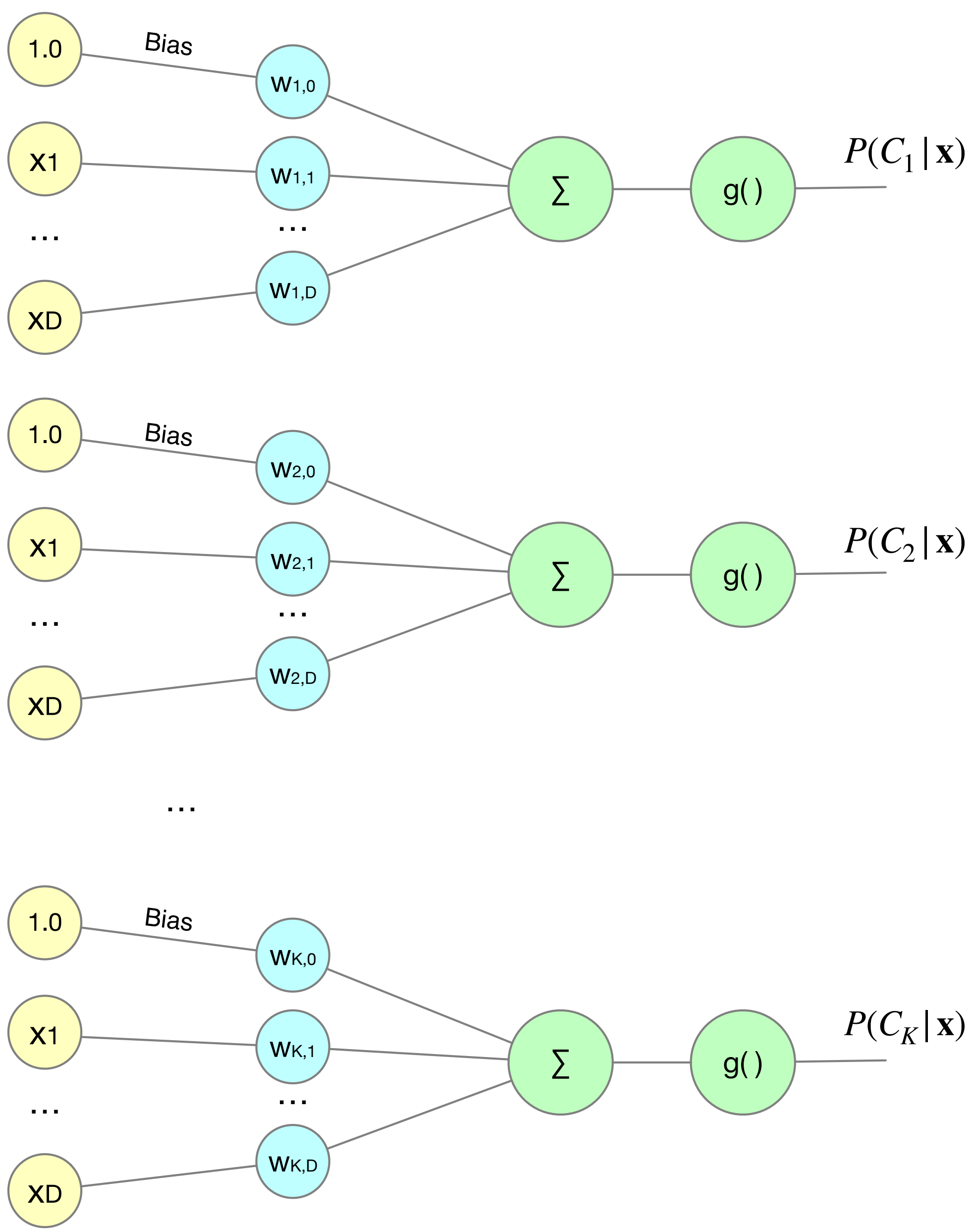
Schematic of a logistic regression classifier



- A logistic regression is actually a one layer neural network where
  - the thetas are the weights
  - the activation function is a sigmoid

Multi-class classification with logistic regression

- We can go to multi class by training K different logistic regression in a “**one vs rest**” approach.



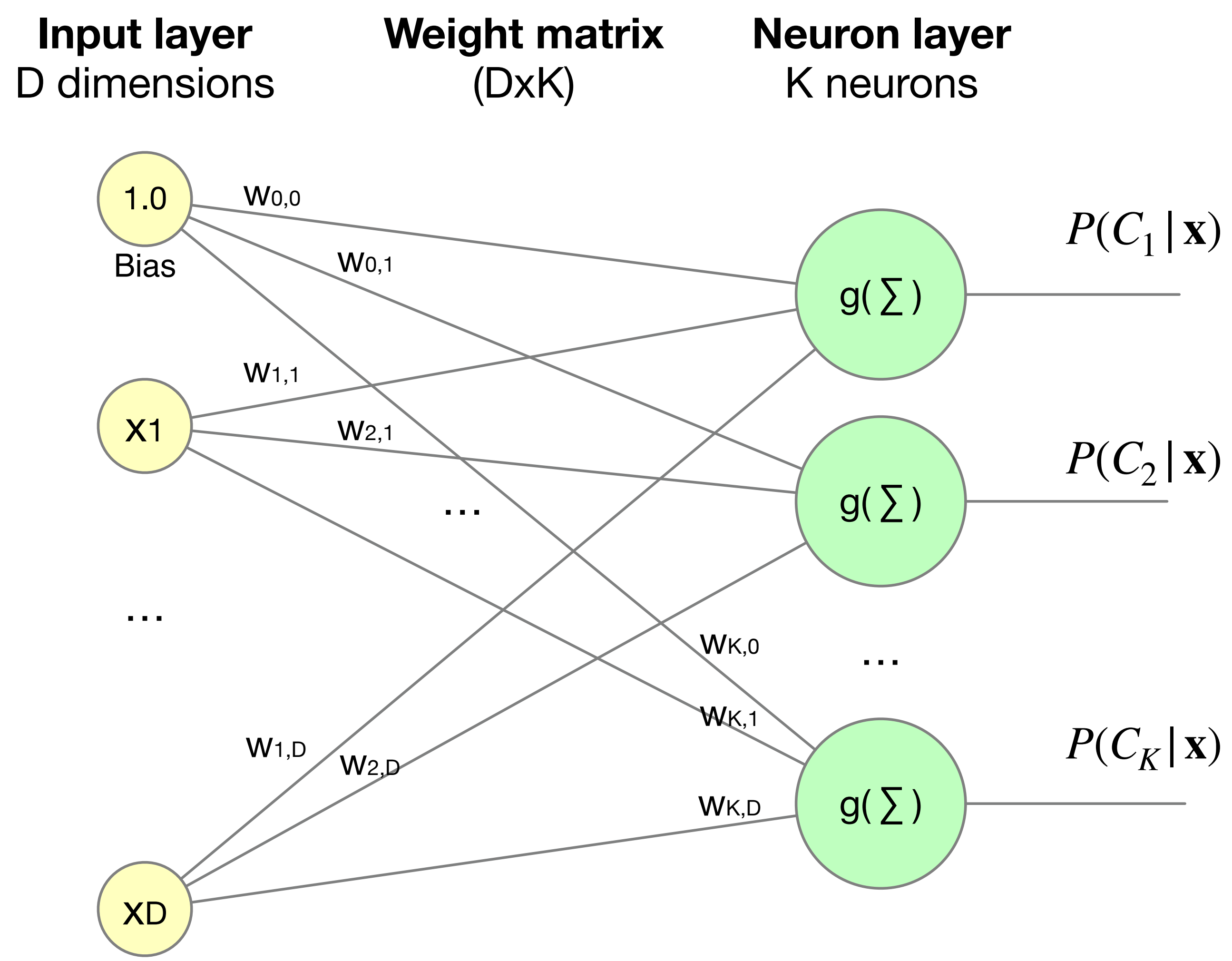
Class 1 vs the rest. Estimates $P(C_1 | \mathbf{x})$

Class 2 vs the rest. Estimates $P(C_2 | \mathbf{x})$

⋮

Class K vs the rest. Estimates $P(C_K | \mathbf{x})$

Link between logistic regression and ANNs



A logistic regression is equivalent to a 1 layer neural network

- Number of neurons = number of classes
- Trainable with (stochastic) gradient approaches using a **1-hot encoding** at the output
- The objective function of logistic regression is actually the **cross-entropy** (with a minus sign) used when training Artificial Neural Network

Conclusions

- Another way to build classification systems is to model a decision boundary that will maximise the number of correct classifications.
- One way to do that is by using a **logistic regression**
  - The logistic regression can be solved in a similar way as with linear regression, i.e. with a **gradient** approach
  - The objective function of logistic regression is to **maximise the a posteriori probability** of each class
- Basic logistic regression is done for 2-classes problems but it can be extended to multi-class with 1-vs-rest approaches
- There is a clear **link between logistic regression and ANN**
  - A multiclass logistic regression is actually a 1 layer ANN trained using 1-hot on a cross-entropy loss function

References

- Coursera, Machine learning, Andrew Ng, Stanford University, https://www.coursera.org/course/ml