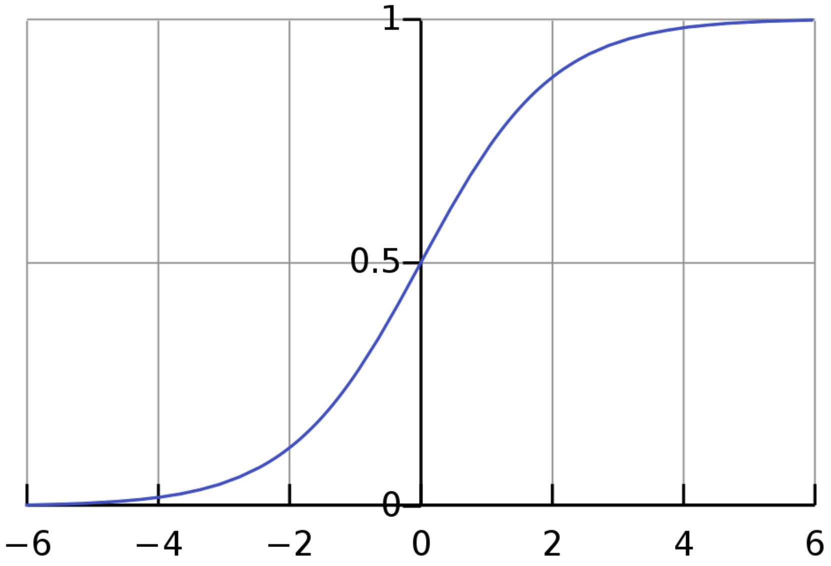
# Lecture 8: Logistic Regression

Andreas Wichert
Department of Computer Science and Engineering
Técnico Lisboa

- In statistics the artificial neuron with sigmoid activation function is called logistic regression
- The sigmoid function called the logistic function.



- Suppose we want to predict whether someone is female $C_1$ or male $C_2$ using height in centimeters
- If linear regression is used $net = y(\mathbf{w}, x) = \mathbf{w}^T \cdot \mathbf{x}$
- The predicted values will become greater than one and less than zero.
- Such values are inadmissible!

- However with

$$p(C_1|\mathbf{x}) = \frac{e^{(net)}}{1+e^{(net)}} = \frac{1}{1+e^{(-net)}} \in [0,1]$$

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x}) = \frac{1}{1+e^{(net)}} \in [0,1]$$

with odds being

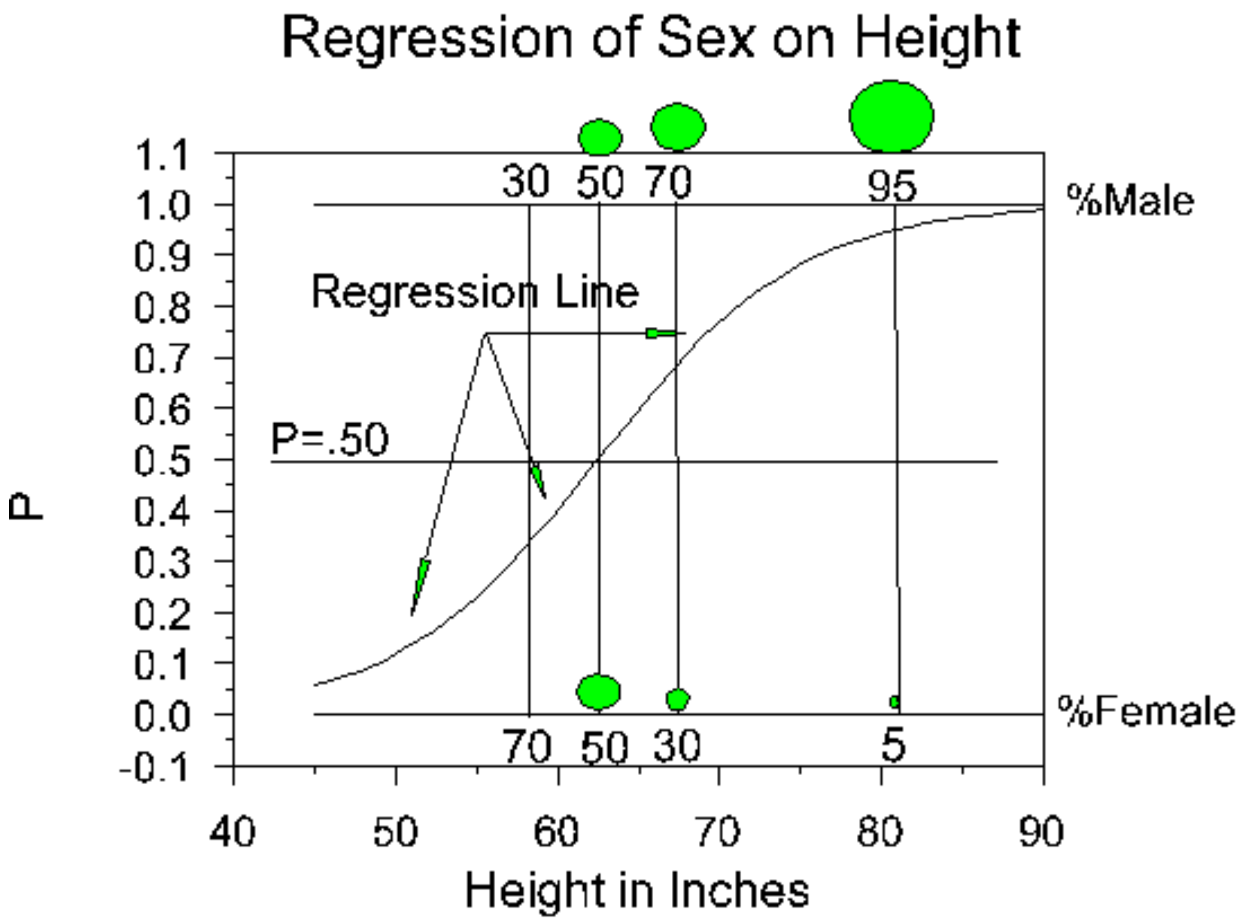
$$odds = \frac{p(C_1|\mathbf{x})}{p(C_2|\mathbf{x})} = \frac{p(C_1|\mathbf{x})}{1-p(C_1|\mathbf{x})} = e^{(net)}.$$

we can overcome this problem.

The *logit* function that is the inverse of logistic (sigmoid $\sigma$) function with

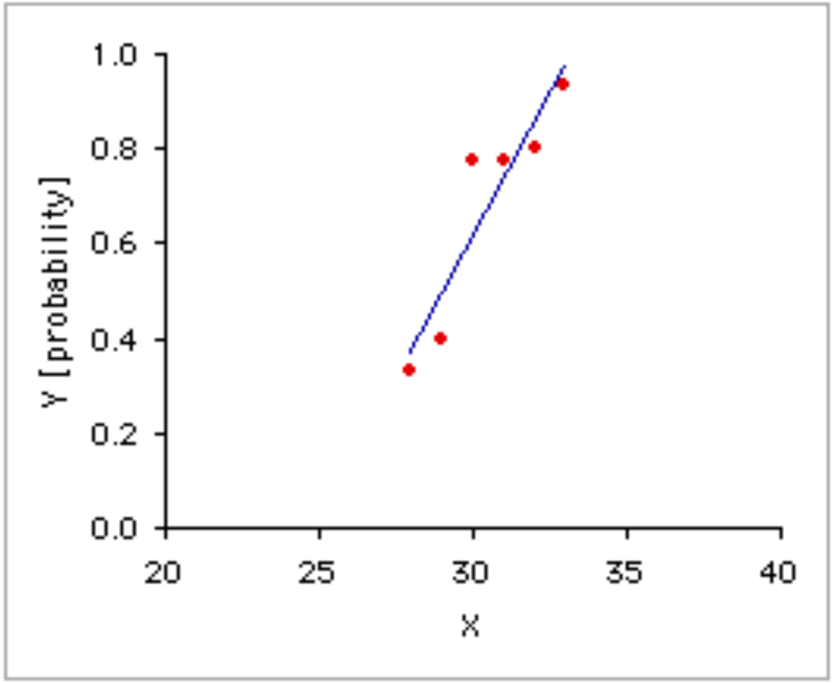
$$\log(odds) = logit(\mathbf{w}^T \cdot \mathbf{x}) = \log\left(\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})}\right) = \mathbf{w}^T \cdot \mathbf{x} = net$$

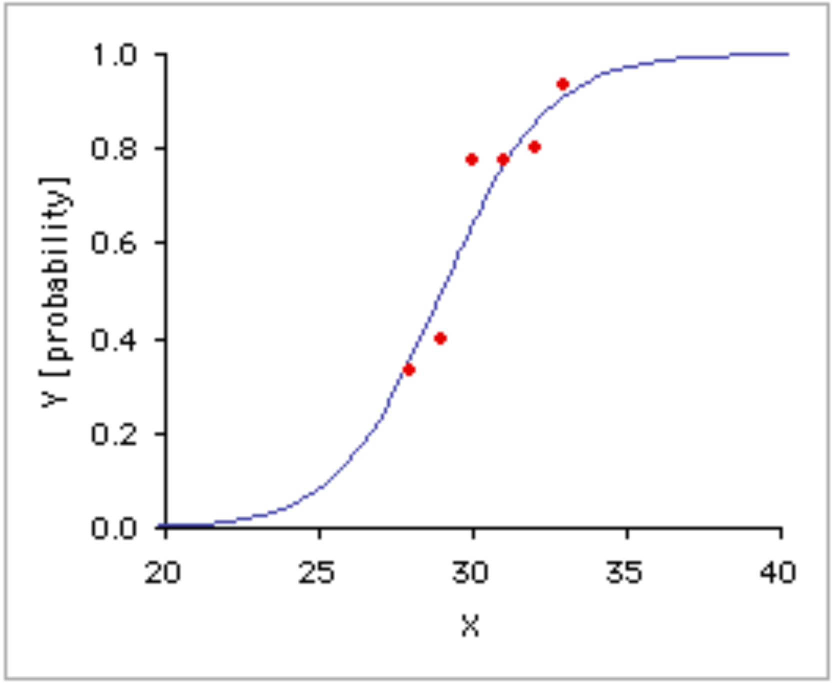
corresponds to the linear regression. It is common in machine learning community to call a sigmoid unit with cross entropy loss function a logistic regression.




- None of the observations (data points) fall on the regression line
- They are all zero or one

# Linear vs. Logistic regression



It followos

$$o = \sigma(net) = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_j\right) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}\right)$$

$$\sigma'(net) = \sigma(net) \cdot (1 - \sigma(net))$$

and

$$\sigma(-net) = 1 - \sigma(net)$$

We can think of the sigmoid output unit as using the linear input $net$ and using the sigmoid activation function to convert $net$ into a probability. It can be interpreted as logistic regression with two classes $C_1$ and $C_2$ with

$$p(C_1|\mathbf{x}) = \sigma\left(\sum_{j=0}^{N} w_j \cdot x_j\right) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}\right)$$

and

$$p(C_2|\mathbf{x}) = 1 - p(C_1|\mathbf{x})$$

# Training

How to train

$$Data = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \cdots, , (\mathbf{x}_N, t_N)\}, \quad t_k \in \{0, 1\}$$

The likelihood function can now be written

$$p(\mathbf{t}|\mathbf{w}) = \prod_{k=1}^{N} p(C_1|\mathbf{x}_k)^{t_k} \cdot (1 - p(C_1|\mathbf{x}_k))^{1-t_k}$$

with

$$o_k = p(C_1|\mathbf{x}_k) = \sigma\left(\mathbf{w}^T \cdot \mathbf{x}_k\right) = \sigma(net_k)$$

$$p(\mathbf{t}|\mathbf{w}) = \prod_{k=1}^{N} o_k^{t_k} \cdot (1 - o_k)^{1-t_k}$$

# Error function

- Error function is defined by negative logarithm of the likelihood

$$E(\mathbf{w}) = -\log(p(\mathbf{t}|\mathbf{w})) = -\sum_{k=1}^{N} (t_k \log o_k + (1 - t_k) \log(1 - o_k))$$

which gives the cross entropy error with

$$\neg t_k = (1 - t_k)$$

$$H(t, p) = -\sum_{k=1}^{N} (t_k \cdot \log(p(C_1|\mathbf{x}_k)) + \neg t_k \cdot \log(p(C_2|\mathbf{x}_k)))$$

$$\neg t_k = (1 - t_k)$$

$$H(t, p) = -\sum_{k=1}^{N} (t_k \cdot \log(p(C_1|\mathbf{x}_k)) + \neg t_k \cdot \log(p(C_2|\mathbf{x}_k)))$$

The gradient that minimizes the error (loss) function is given by

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \left( -\sum_{k=1}^{N} (t_k \log o_k + (1 - t_k) \log(1 - o_k)) \right)$$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j}\left(-\sum_{k=1}^{N}(t_k \log o_k + (1-t_k)\log(1-o_k))\right)$$

with

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N}\left(\frac{t_k}{o_k} + -\frac{(1-t_k)}{(1-o_k)}\right)\cdot\frac{\partial}{\partial w_j}(o_k)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}\left(\frac{(1-t_k)}{(1-o_k)} - \frac{t_k}{o_k}\right)\cdot\frac{\partial}{\partial w_j}(o_k)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}\left(\frac{(1-t_k)}{(1-o_k)} - \frac{t_k}{o_k}\right)\cdot(o_k\cdot(1-o_k))\cdot\frac{\partial}{\partial w_j}\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N}((1-t_k)\cdot(o_k)) - (t_k)\cdot(1-o_k))\cdot\frac{\partial}{\partial w_j}\left(\sum_{j=0}^{D} w_j \cdot x_{k,j}\right)$$

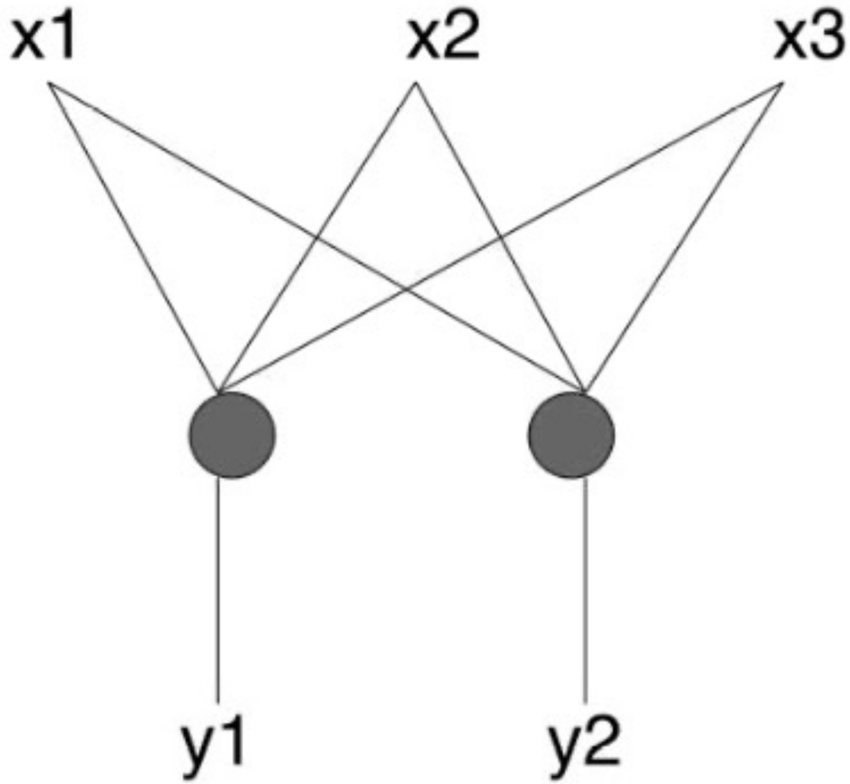
$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N} (o_k - t_k) \cdot \frac{\partial}{\partial w_j} \left( \sum_{j=0}^{D} w_j \cdot x_{k,j} \right)$$

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{N} (o_k - t_k) \cdot x_{k,j}$$

$$\frac{\partial E}{\partial w_j} = -\sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

The update rule for gradient decent is given by

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$




2 artificial neurons correspond to two perceptrons with the same input x and two target values represented by the vector $y = (y_1, y_2)$, the actual output computed by the network is $o = (o_1, o_2)$.

For $K$ artificial neurons with a continuous activation function $\phi$ an index $t$ is used to identify the corresponding artificial linear neuron with

$$t \in \{1, 2, \cdots, K\}$$

to identify the weight vector $\mathbf{w}_t$ and the $t$ the output $\mathrm{o}_t$ from the neuron,

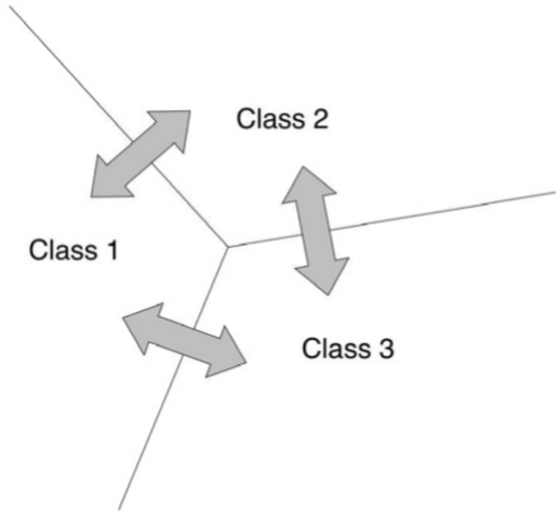
$$o_t = \phi\left(\sum_{j=1}^{D} w_{t,j} \cdot x_j\right)$$

Assuming that the training set consists of $N$ observations

$$X = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k, \cdots, \mathbf{x}_N)^T$$

and respective target values represented as vectors of dimension $K$ (since $t$ is used as an index, we will use $y_{kt}$ to indicate the specific target)

$$Y = (\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_k, \cdots, \mathbf{y}_N)^T.$$

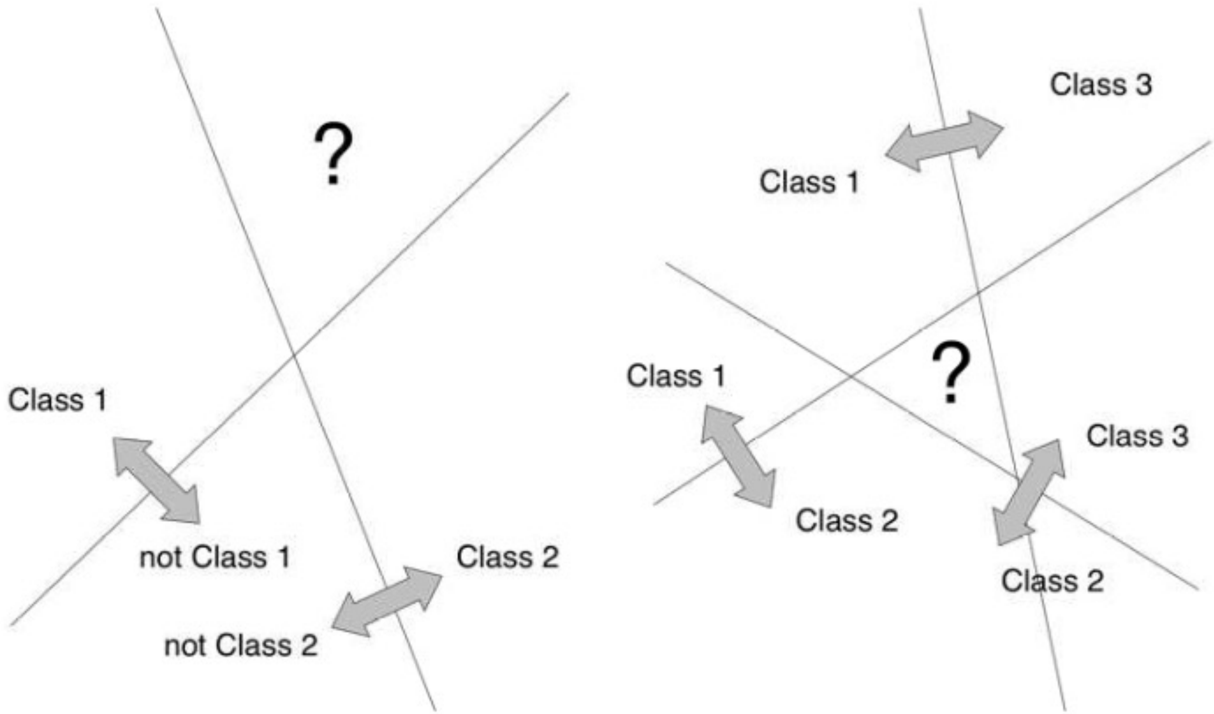



During training, each neuron is trained individually with its target value $y_{kt}$

$$y_{kt} \in \{0, 1\}, \quad \sum_{t=1}^{K} y_{kt} = 1$$

After training, the prediction for an input pattern $\mathbf{x}$ is done using

$$\arg\max_{t} \left(\phi\left(\mathbf{w}_t^T \cdot \mathbf{x}\right)\right). \tag{5.51}$$




For the perceptron we cannot apply the $\arg\max_t$ operation since the output of each neuron is either 1 or 0 and building a $K-1$ classifiers each of which solves a two class problem creates ambiguous regions

For $K$ artificial neurons with the sigmoid activation the results cannot be interpreted as probabilities any more

$$o_t = \sigma\left(\sum_{j=1}^{D} w_{t,j} \cdot x_j\right) = \sigma(net_t)$$

since

$$\sum_{t=1}^{K} o_t \neq 1.$$

To be able to interpret them again as such, we can simply normalize the values with

$$o_s = \frac{\sigma(net_s)}{\sum_{t=1}^{K} \sigma(net_t)}.$$

Since each neuron can be viewed as a single classifier as well with odds

$$odds_s = \frac{p(C_{s,1}|\mathbf{x})}{p(C_{s,2}|\mathbf{x})} = \frac{p(C_{s,1}|\mathbf{x})}{1 - p(C_{1,s}|\mathbf{x})} = e^{(net_s)}.$$

$$\log(odds) = logit(\sigma(\mathbf{w}^T \cdot \mathbf{x})) = \log\left(\frac{p(C_1|\mathbf{x})}{1 - p(C_1|\mathbf{x})}\right) = \mathbf{w}^T \cdot \mathbf{x}$$

$$o_s = \frac{odd_s}{\sum_{t=1}^{K} odd_t} = \frac{\exp(net_s)}{\sum_{t=1}^{K} \exp(net_t)}$$

$$p(C_s|\mathbf{x}) = \sigma(\mathbf{net})_s = \frac{\exp(net_s)}{\sum_{t=1}^{K} \exp(net_t)}$$

$$p(C_\kappa|\mathbf{x}) = \sigma(\mathbf{net})_\kappa = \frac{p(\mathbf{x}|C_\kappa) \cdot p(C_\kappa)}{\sum_{j=1}^{K} p(\mathbf{x}|C_j) \cdot p(C_j)} = \frac{\exp(net_k)}{\sum_{j=1}^{K} \exp(net_j)}$$

- The **softmax function** is used in various multi class classification methods, such as multinomial logistic regression (also known as softmax regression) with the prediction

$$\arg\max_\kappa \left(\sigma(\mathbf{net})_\kappa\right) = \arg\max_\kappa \left(\mathbf{w}_\kappa^T \cdot \mathbf{x}\right)$$

# Cross Entropy Loss Function for softmax

- Taking the error of the gradient function with respect to the vector $\boldsymbol{w}_j$ we obtain

$$\nabla_{\mathbf{w}_s} E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = \sum_{k=1}^{N} (o_{ks} - y_{ks}) \cdot \mathbf{x}_k = -\sum_{k=1}^{N} (y_{ks} - o_{ks}) \cdot \mathbf{x}_k$$

$$\sigma(net_{ks}) = \frac{\exp(net_{ks})}{\sum_{t=1}^{K} \exp(net_{kt})}$$

which is called the normalized exponential or softmax function. For simplification of the derivative function we drop the index $k$ that indicates the $k$-th training pattern and we we get

$$\frac{\partial \sigma(net_s)}{\partial net_t} = \sigma(net_s) \cdot (I_{st} - \sigma(net_t)) = o_s \cdot (I_{st} - o_t)$$

means if $t = s$

$$\frac{\partial \sigma(net_s)}{\partial net_t} = \sigma(net_s) \cdot (1 - \sigma(net_s)) = o_s \cdot (1 - o_s)$$

if $t \neq s$

$$\frac{\partial \sigma(net_s)}{\partial net_t} = -\sigma(net_t) \cdot \sigma(net_s)) = -o_t \cdot o_s$$

# Kronecker Function

with the Kronecker Function

$$I_{st} = \delta_{st} = \begin{cases} 1, & \text{if } s = t \\ 0, & \text{otherwise.} \end{cases}$$

- Taking the error of the gradient function with respect to $net_j$ over all training patterns we get

$$E(\mathbf{w}) = -\sum_{k=1}^{N}\sum_{t=1}^{K} y_{kt} \cdot \log o_{kt}$$

$$\frac{\partial E}{\partial o_{kt}} = -\sum_{k=1}^{N}\sum_{t=1}^{K} \frac{y_{kt}}{o_{kt}}, \qquad \frac{\partial o_{kt}}{\partial net_s} = o_{kt} \cdot (I_{ts} - o_{ks}),$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}\sum_{t=1}^{K} \frac{y_{kt}}{o_{kt}} \cdot o_{kt} \cdot (I_{ts} - o_{ks})$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}\sum_{t=1}^{K}(y_{kt} \cdot (I_{ts} - o_{ks}),$$

since

$$\sum_{t=1}^{K} y_{kt} = 1,$$

$$\frac{\partial E}{\partial net_s} = -\sum_{k=1}^{N}(y_{ks} - o_{ks})$$

Since

$$net_{ks} = \sum_{j=1}^{D} w_{js} x_{kj}$$

we get for all $N$ training patterns

$$\frac{\partial E}{\partial w_{js}} = -\sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

$$\frac{\partial E}{\partial w_{js}} = -\sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

with the learning rule

$$\Delta w_{js} = \eta \cdot \sum_{k=1}^{N}(y_{ks} - o_{ks}) \cdot x_{kj}$$

$$w_{js}^{new} = w_{js}^{old} + \Delta w_{js}.$$

This algorithm is called the logistic regression.

# Logistic Regression Algorithm

Given a training set (sample)

$$Data = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_k, \mathbf{y}_k), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$$

with $\mathbf{y}_k$ represented as vectors of dimension $K$. During the training each neuron is trained individually withs its target value $y_{kt}$

$$y_{kt} \in \{0, 1\}, \quad \sum_{t=1}^{K} y_{kt} = 1$$

the goal of the algorithm is to correctly classify the test set (population) into $K$ classes $C_1 = 100\cdots$, $C_2 = 010\cdots$, $C_3 = 001\cdots$, $\cdots$

(1) iterations=0;

(2) $\eta \in (0, 1]$;

(3) FOR t=1 TO K

{

Initialise all the weights $w_{0t}, w_{1t}, \cdots, w_{Dt}$ to some random values;

}

(4) Choose a pattern $\mathbf{x}_k$ out of the training set;

(5) FOR t=1 TO K

{

Compute $net_{kt} = \sum_{i=1}^{D} w_{jt} \cdot x_{kj} + w_{0t} = \langle \mathbf{x}_k | \mathbf{w}_t \rangle + w_{0t} \cdot x_0$;

Compute $odds_{kt} = \exp(net_{kt})$;

}

(6) FOR t=1 TO K

{

Compute $o_{kt} = \frac{odds_{kt}}{\sum_t odds_{kt}}$;

Compute $\Delta w_{jt} = \eta \cdot (y_{kt} - o_{kt}) \cdot x_{k,j}$;

Update the weights $w_{jt} = w_{jt} + \Delta w_{jt}$;

}

(7) iterations++;

(8) If no change in weights for all training set or maximum number of iteration THEN STOP ELSE GOTO 4;

- An epoch corresponds to adapting all trainings vectors (patterns) $\mathbf{x}_k$ with $k = 1,2,\cdots,N$, with epoch = iterations/N.
- Depending on the training set, the initialization and the size of the learning rate, the epochs have to be repeated several times untill a solution is reached.

# Sigmoid Unit versus Logistic Regression

Sigmoid Unit is with target, should be positive (between zero and one):

$$\Delta w_j = \eta \cdot \alpha \cdot \sum_{k=1}^{N} (t_k - o_k) \boxed{\sigma\left(net_{k,j}\right) \cdot \left(1 - \sigma\left(net_{k,j}\right)\right)} \cdot x_{k,j}$$

Logistic Regression is with target $t_k \in \{0, 1\}$

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

If we assume $\alpha = 1$ then the difference between sigmoid unit and the logistic regression that was derived by maximising the negative logarithm of the likelihood is

$$\sigma\left(net_{k,j}\right) \cdot \left(1 - \sigma\left(net_{k,j}\right)\right) \geq 0$$

the step size in the direction of gradient. Does it mean that Sigmoid Unit converge faster?

# Linear Unit versus Logistic Regression

Target can be any value and can be solved by closed-form solution, by pseudo inverse

$$o_k = \sum_{j=0}^{D} w_j \cdot x_{k,j}$$

Target $t_k \in \{0,1\}$ cannot be solved by closed-form solution

$$o_k = \frac{1}{1 + e^{(-\alpha \cdot (\sum_{j=0}^{N} w_j \cdot x_{k,j}))}}$$

$$\Delta w_j = \eta \cdot \sum_{k=1}^{N} (t_k - o_k) \cdot x_{k,j}.$$

Logistic Regression as well as the sigmoid unit gives a better decision boundary.

For Sigmoid (Logistic) distant points from the decision boundary have the same impact

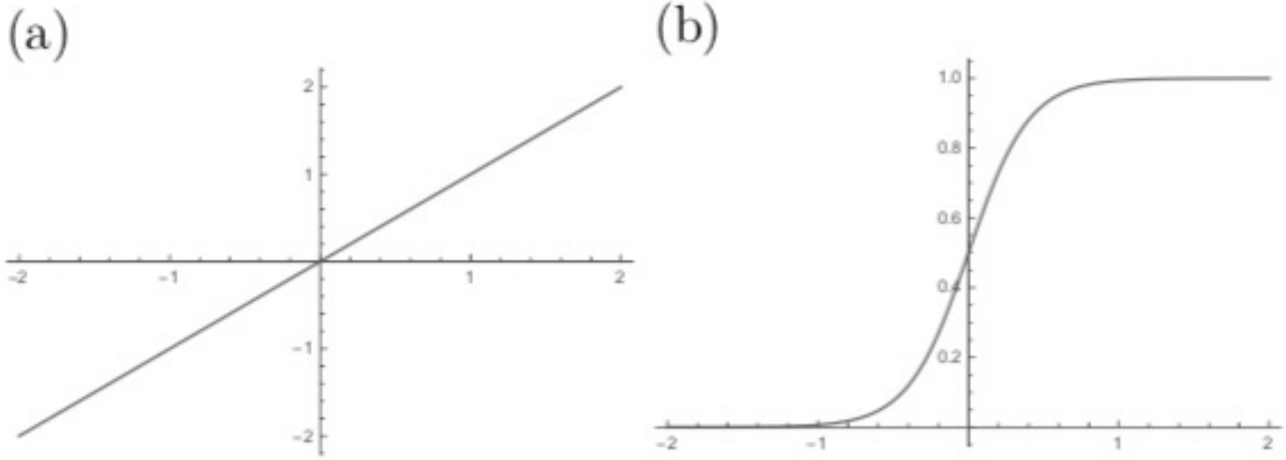
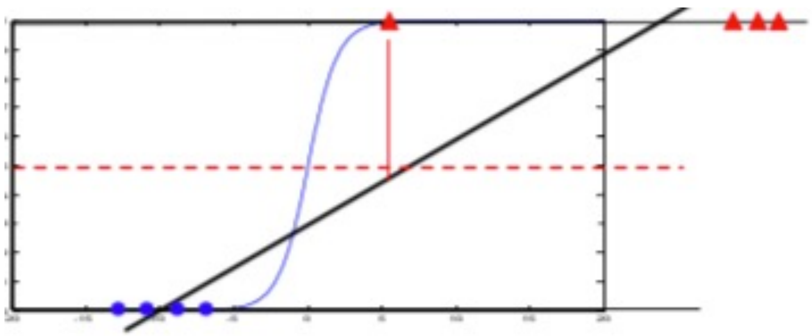
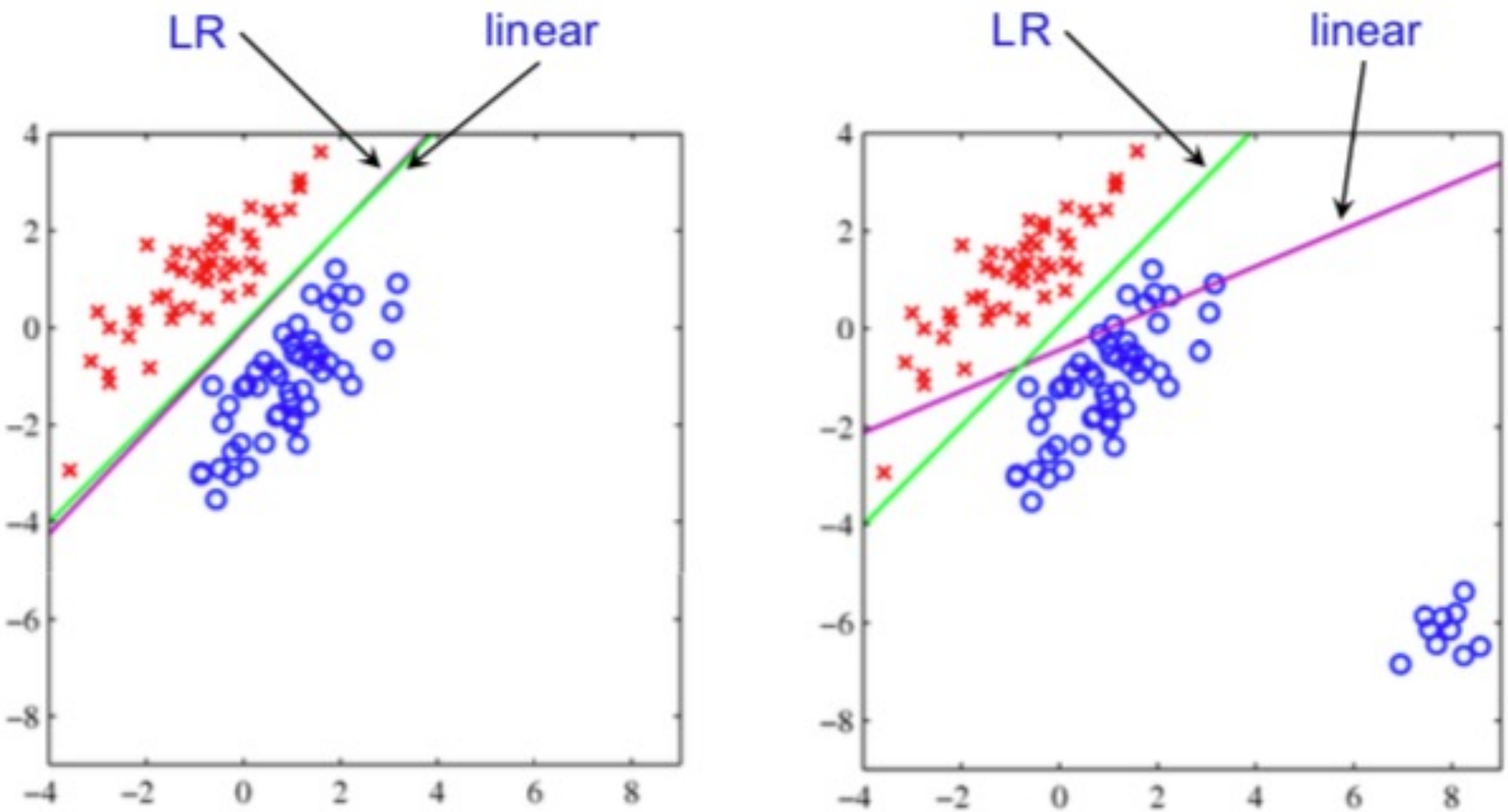



Figure 1.2: (a) Linear activation function. (b) The function $\sigma(net)$ with $\alpha = 5$

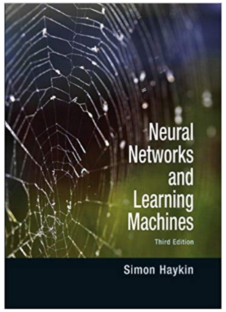
Distance



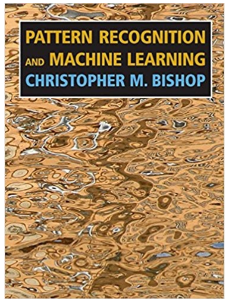
# Better Decision Boundary of Logistic Regression LR (sigmoid) to Linear Unit



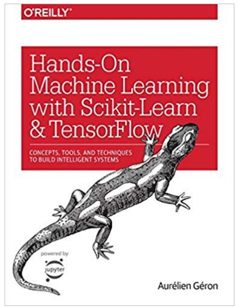
# Literature




- Simon O. Haykin, Neural Networks and Learning Machine, (3rd Edition), Pearson 2008
  - Chapter 1

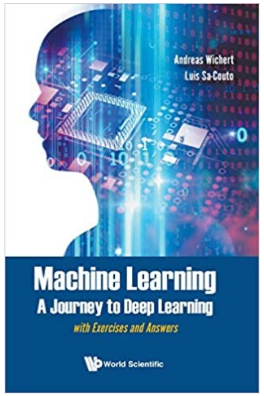



- Christopher M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer 2006
  - Chapter 4




- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurélien Géron , O'Reilly Media; 1 edition, 2017
  - Chapter 4

# Literature




- Machine Learning - A Journey to Deep Learning, A. Wichert, Luis Sa-Couto, World Scientific, 2021
  - Chapter 5