UiO : University of Oslo

institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Lecture 8

Multi-layer neural networks and backpropagation

Jan Tore Lønning

8.1 Feed-forward Neural networks

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

Today

1. Feed-forward neural networks (Multi-layer Perceptron)
2. Matrix representations of neural networks
3. The Backpropagation Algorithm
4. Finer details
5. More on Evaluation

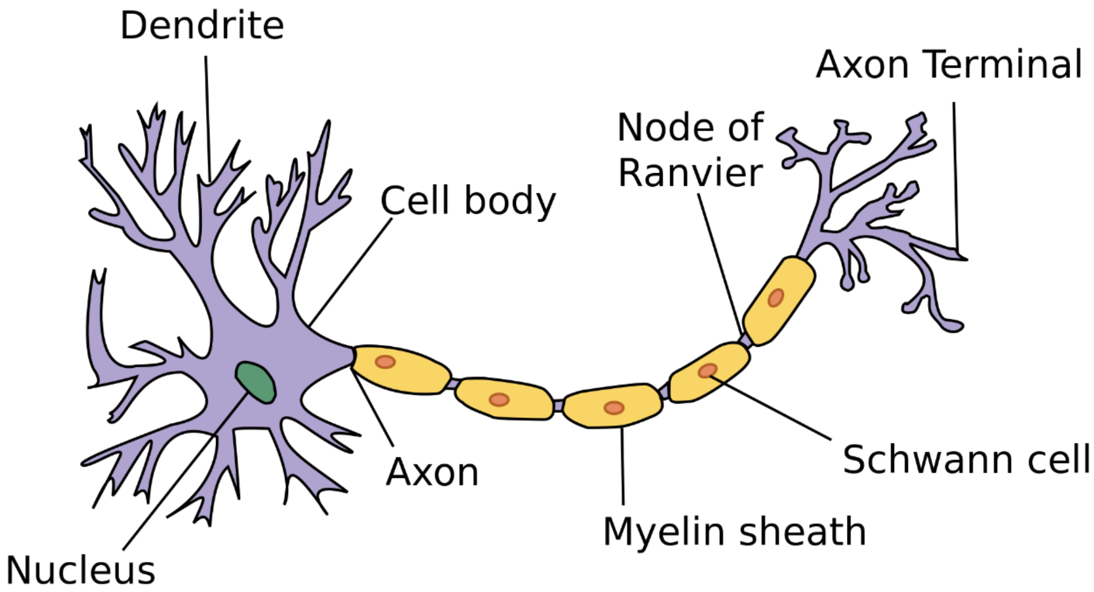
The neural inspiration

- So far inspired by one neuron
- That does not make intelligence

The human brain, roughly

  - 10^{11} Neurons
  - 10^{14} Synapses
  - The strength is the interactions

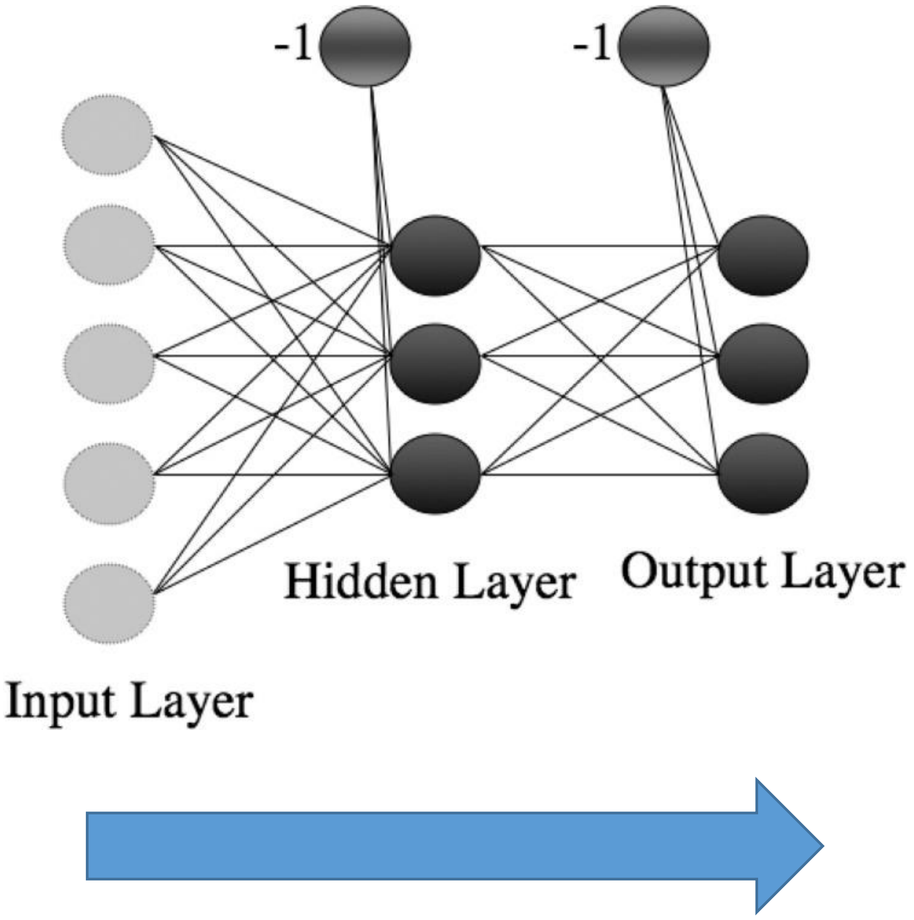
- Neural Networks



https://simple.wikipedia.org/wiki/Neuron#/media/File:Neuron.svg

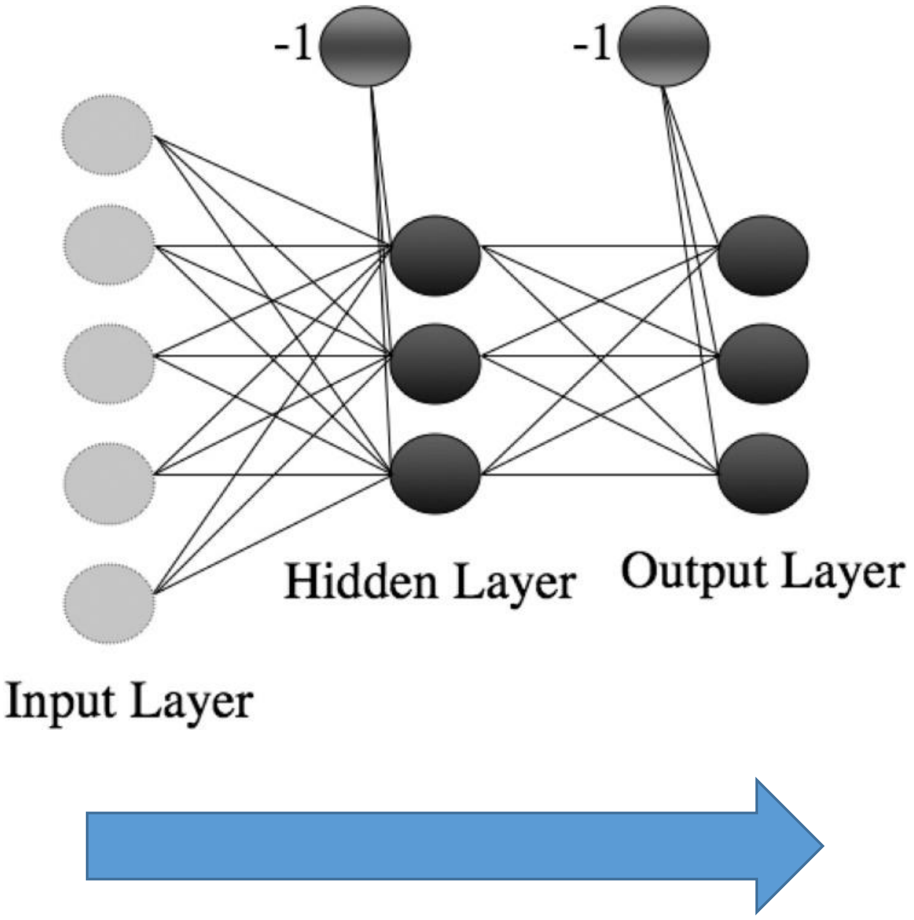
Artificial Neural Networks

- Inspired by the brain
- Does not pretend to be a model of the brain
- The simplest model is the
  - Feed forward network, also called
  - Multi-layer Perceptron



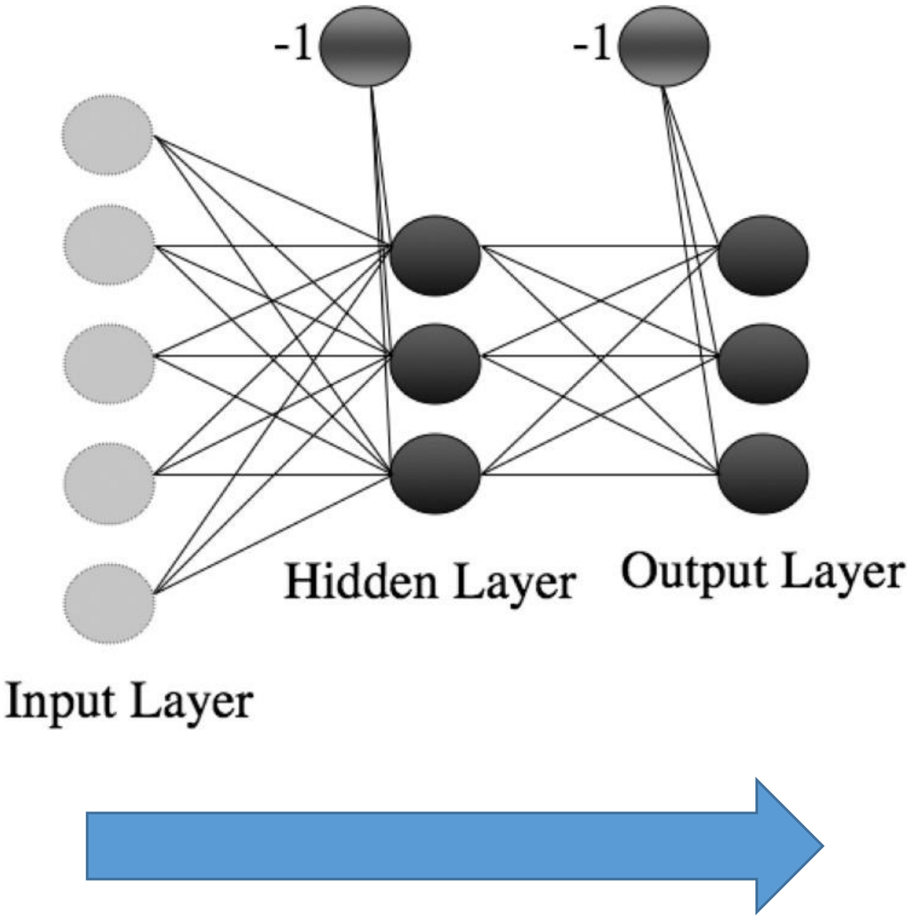
Feed forward network

- An input layer
- An output layer: the predictions
- One or more hidden layers
- Connections from nodes in one layer to nodes in the next layer (from left to right)
- The connections are marked with weights



Going forwards (predictions)

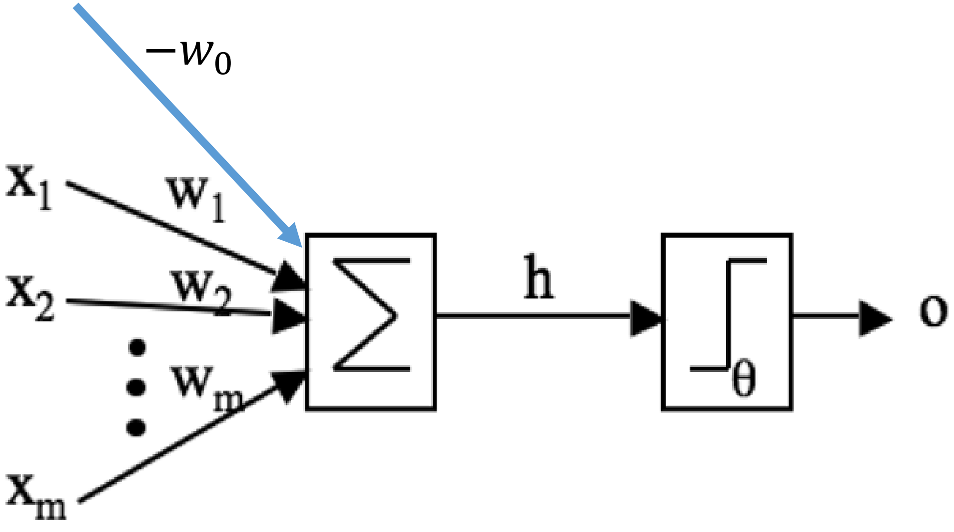
- There is one input node for each feature/dimension in an input vector: $(x_1, x_2, \dots, x_m)$
- In addition, an input bias node $x_0 = -1$
- The input values are multiplied with the weights and summed into each hidden node.
- There is some processing in the hidden node.
- The output values of the hidden nodes are fed to the next layer.
- (etc.)



One hidden unit

1. First sum of weighted inputs :
   - $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
2. Then the result is run through an activation function, g to produce $g(z) = g(\boldsymbol{w} \cdot \boldsymbol{x})$

- The activation function could be the step function,
  - c.f. the XOR-example:
    - Marsland sec 3.4..2 & start ch. 4

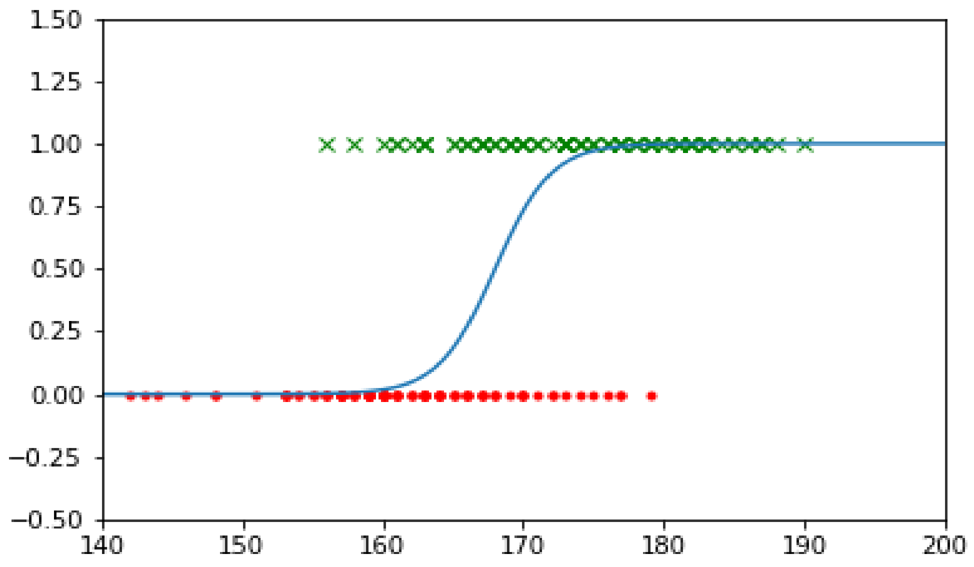


It is the non-linearity of the activation function which makes it possible for MLP to predict non-linear decision boundaries

A differentiable activation function

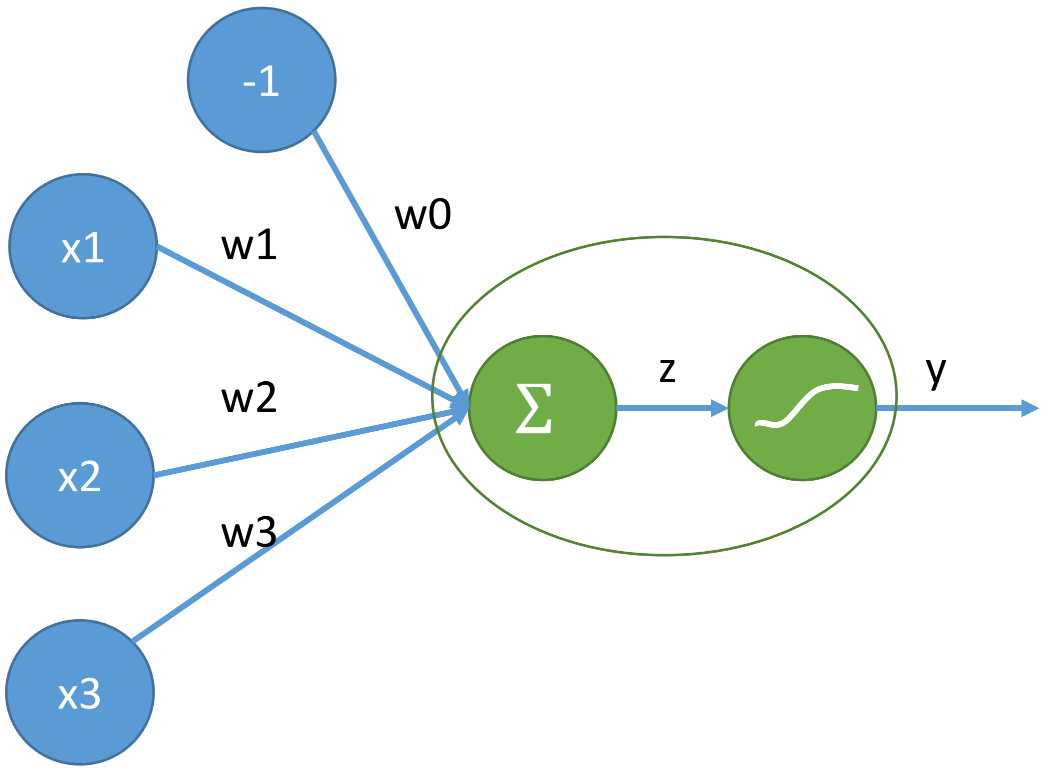
- It is unclear how to update the weights if g isn't differentiable

- One option is to use the logistic (sigmoid) function
  - $y = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$
  - Differentiable
  - $y' = y(1-y)$
- (There are alternative activation functions.)



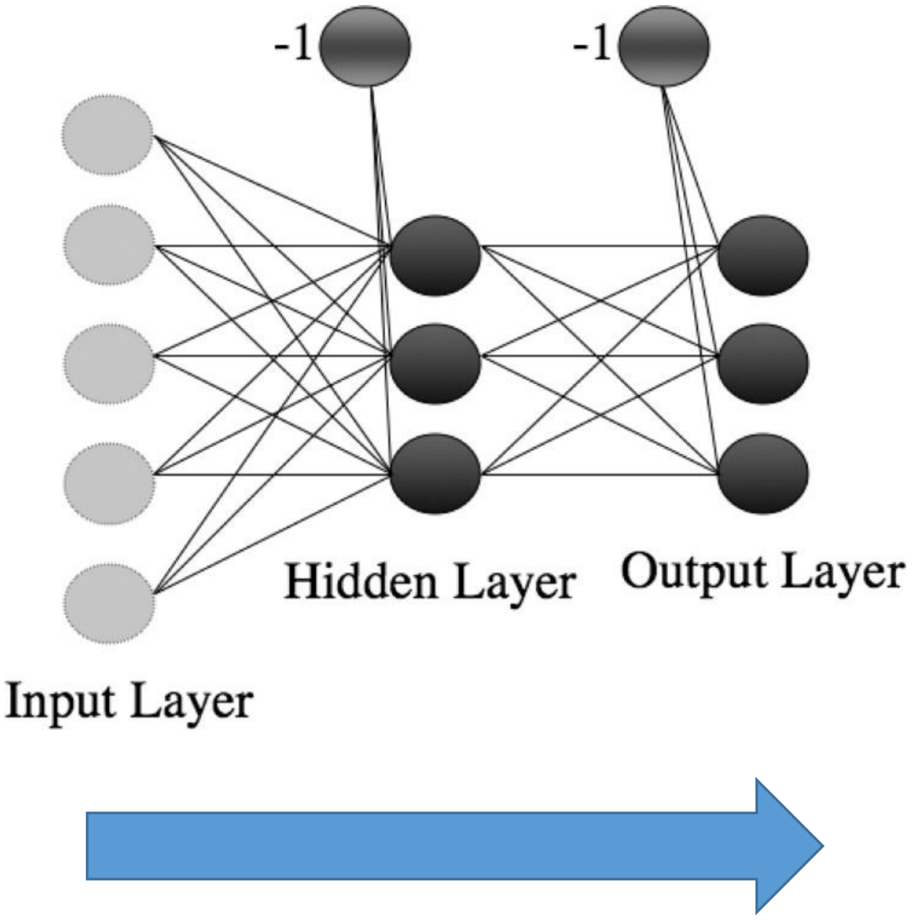
One hidden node

1. First sum of weighted inputs:
   - $z = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
2. Then
   - $y = g(z) = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$



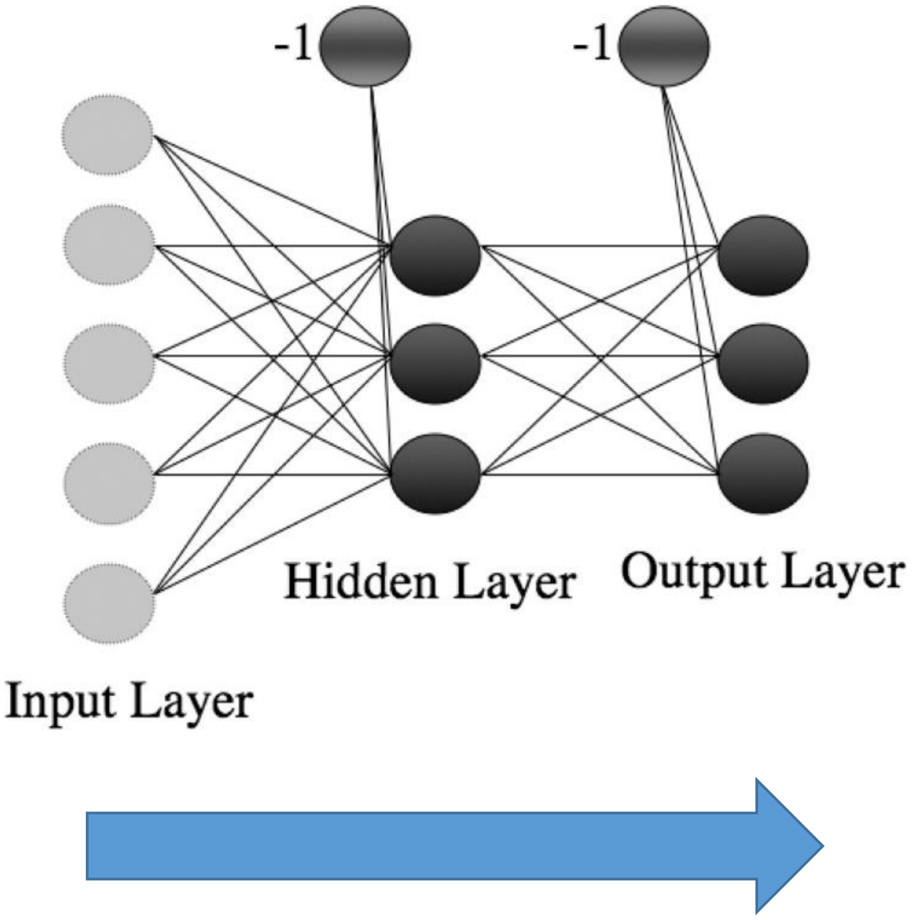
Going forwards (predictions)

- After the processing in the hidden layer, the output is taken as input to the next layer
- One must also add a bias term at this layer.
  - Observe that this has to be done:
    - During processing
    - E.g., over again each time we process the same training item



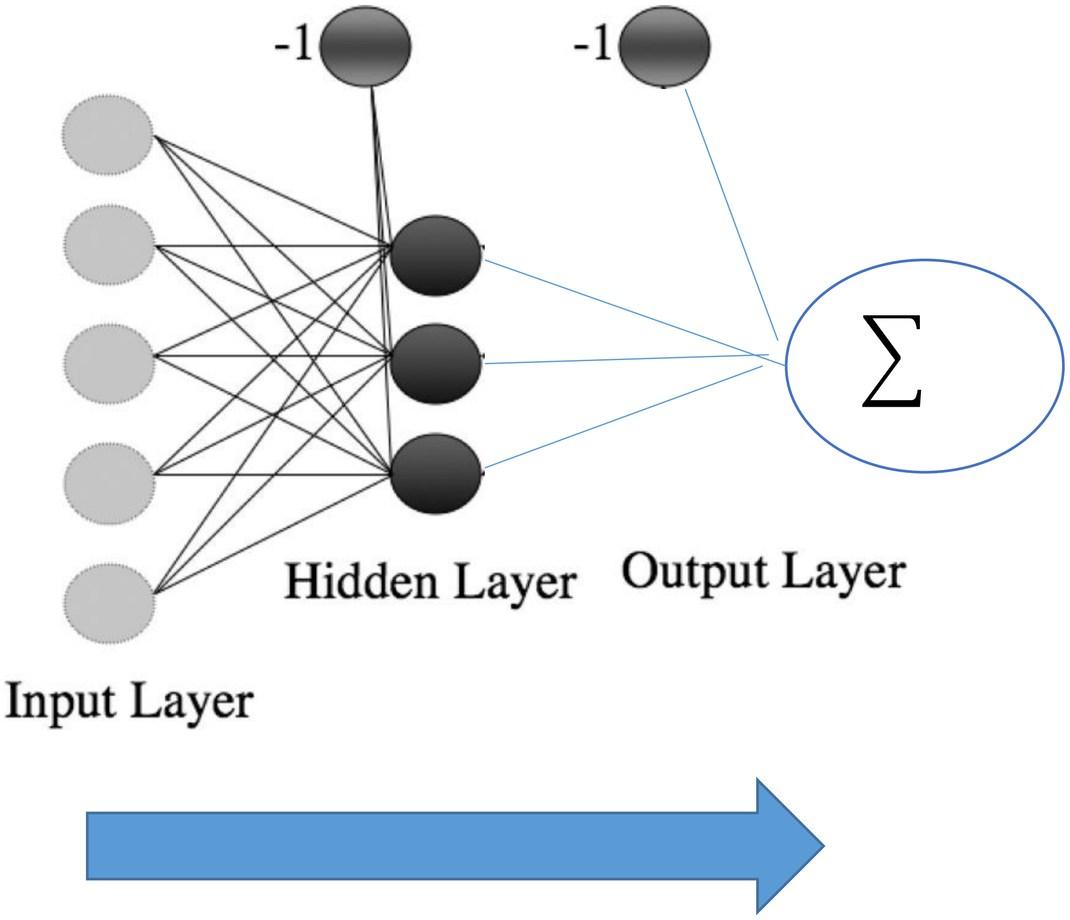
Output layer

- Several possibilities, depending on the task, including:
  - Regression
  - Binary classification
  - Multi-label classification
  - Multi-class classification
- From the last layer to the output layer is like the same tasks without multiple layers!
- c.f. Marsland, sec. 4.2.3



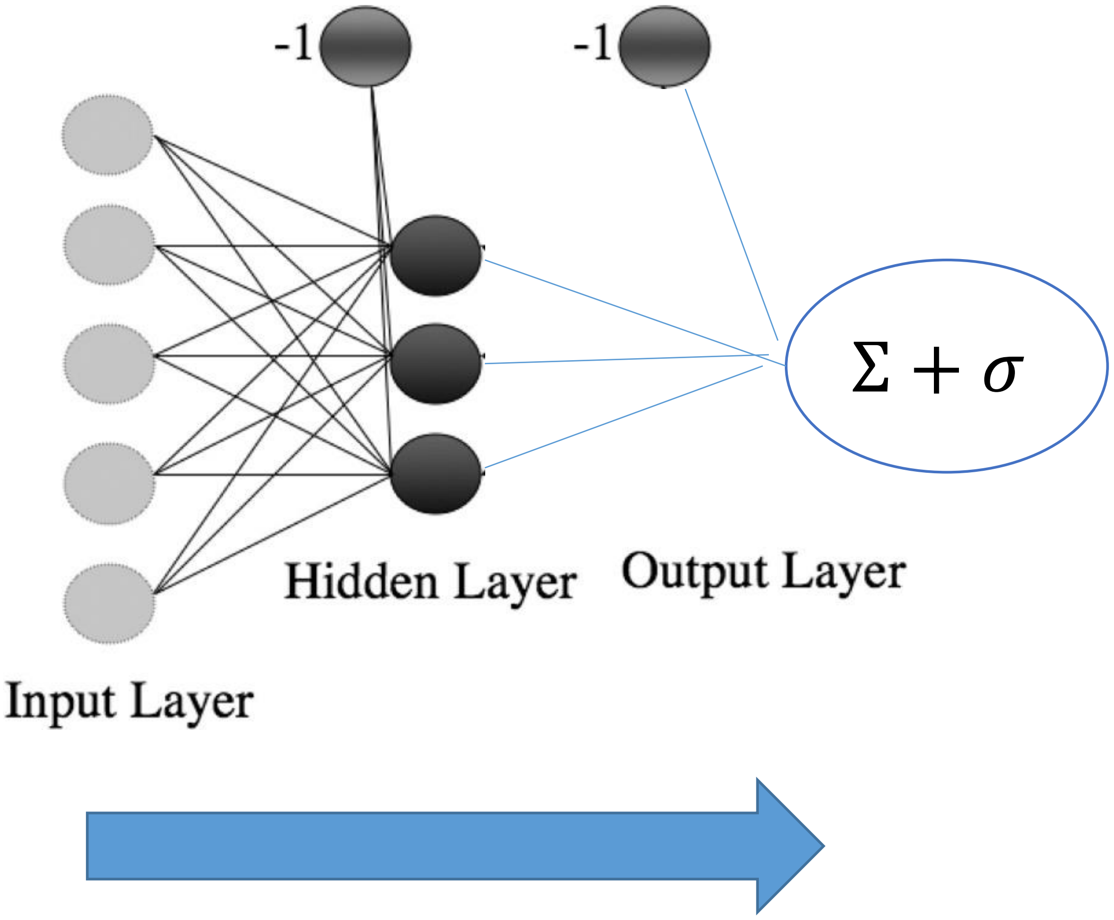
1. Regression

- One output node
- No activation function
  - = activation function is the identity function
- Observe that this can predict non-linear functions!



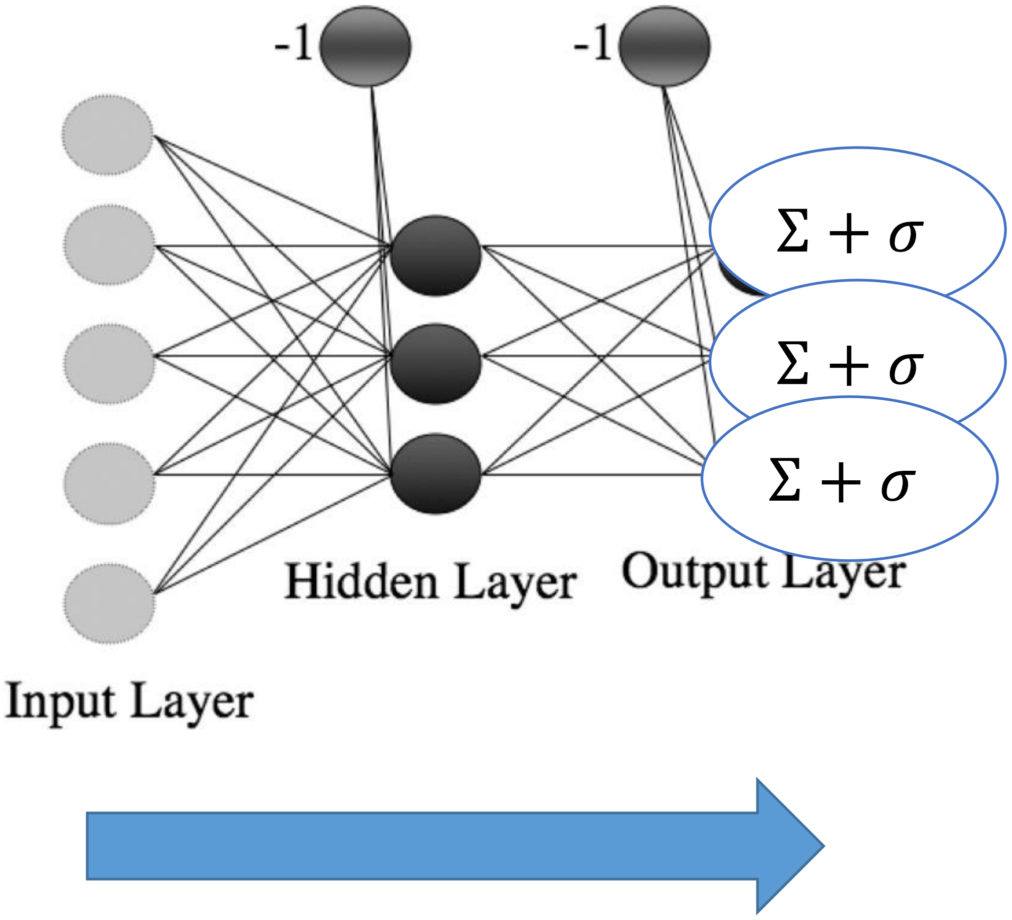
2. Binary classification

- One output node
- Logistic activation function
- Similar to logistic regression
- Can produce non-linear decision boundaries



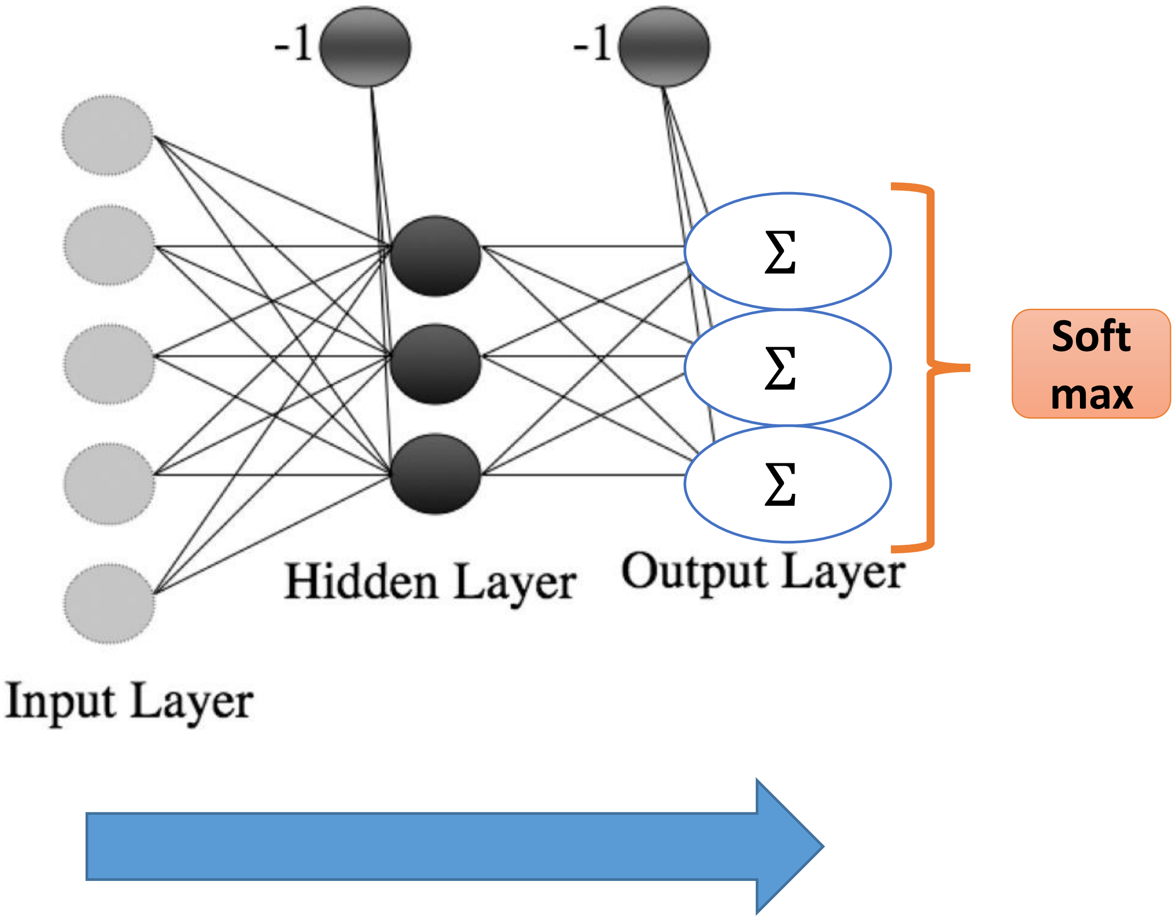
3. Multi-label classification

- Several output nodes
- Logistic activation function
- Can be made multi-class classification by one vs. rest.
- The model Marsland considers



4. Multi-class classification

- Several output nodes
- Sum the weighted inputs at each nodes
- The sums are brought together in the soft-max



8.2 Matrix representations

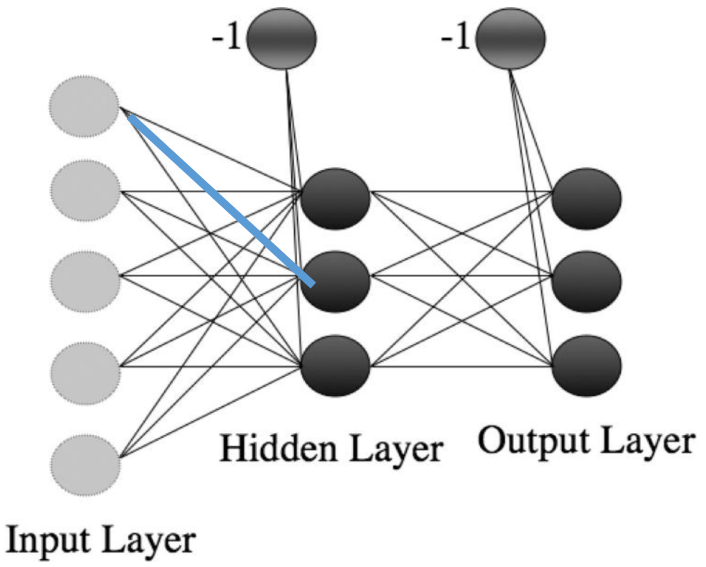
IN3050/IN4050 Introduction to Artificial Intelligence

and Machine Learning

Representing the connections

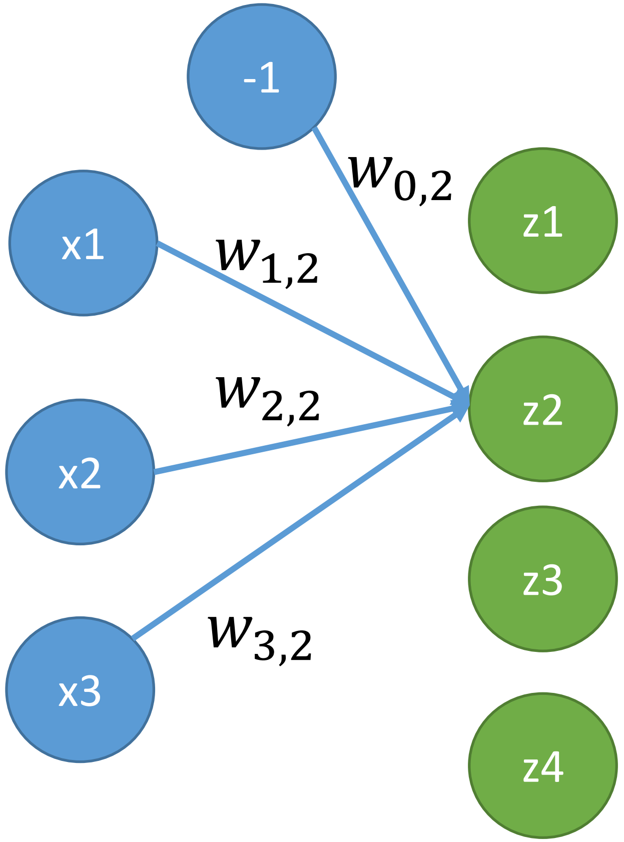
$$
\begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_m \end{bmatrix}
\begin{bmatrix}
w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\
w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\
w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
w_{m,1} & w_{m,2} & \cdots & w_{m,n}
\end{bmatrix}
= \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}
$$

- We use a matrix to represent the connections
- Element $w_{i,j}$ is the connection:
  - from node i
  - to node j
- (Beware, some texts do it differently)



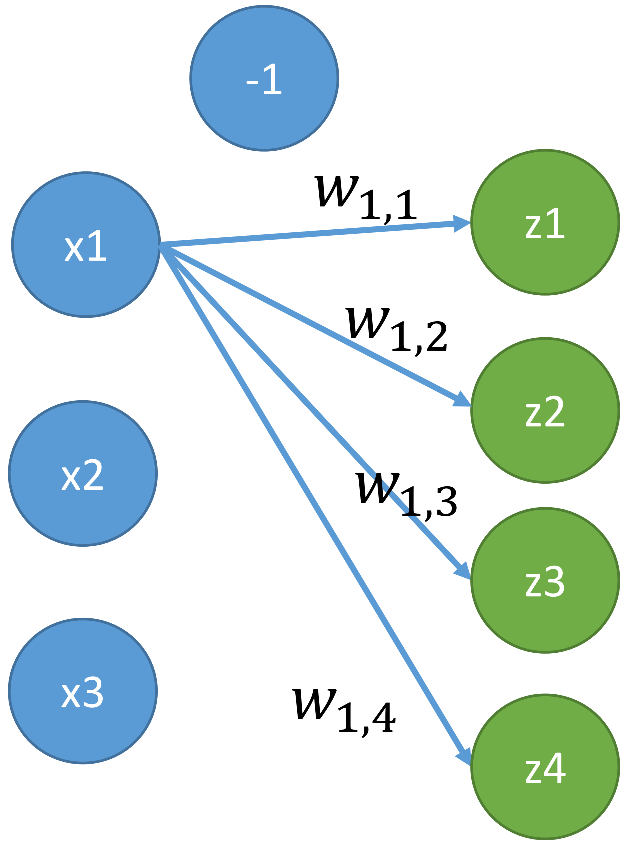
Connections going into a node

$$\begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_m \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}$$



Connections going out of a node

$$\begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_m \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}$$



Batch-processing

$$\begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N,1} & z_{N,2} & \cdots & z_{N,n} \end{bmatrix}$$

- In batch-processing we can multiply by weights and (i) sum the results for (iii) each input item, and (ii) each hidden node in one operation
- Three nested loops by just: XW

Activation function

$$\begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,n} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N,1} & z_{N,2} & \cdots & z_{N,n} \end{bmatrix}$$

- Each $z_{i,j}$ is passed through the activation function: $y_{i,j} = g(z_{i,j})$
- In NumPy this can be done by one operation: $g(XW)$
- Reminder: g may be the logistic function, but doesn't have to
  - i.e., $g(z_{i,j}) = \sigma(z_{i,j}) = \frac{1}{1+e^{-z_{i,j}}}$

Footnote: Notation

- Half of all texts follow us and Marsland with respect to notation
- The other half does differently

	We	Them
Connection from node i to node j	$w_{i,j}$	$w_{j,i}$
Data and weights	XW	WX
Applying activation function	$g(XW)$	$g(WX)$

- It amounts to the same.
- But don’t mix them up!

8.3 Learning by Back-propagation

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

institutt for informatikk

Background

Marsland (p.74), ‘’…just three things that you need to know…’’:

1. If $f(x) = \frac{1}{2}x^2$ then $f'(x) = x$
2. If $f(x) = c$ then $f'(x) = 0$
3. If $f(x) = h(g(x))$ then $f'(x) = h'\left(g(x)\right)g'(x)$ (the chain rule)

He forgot

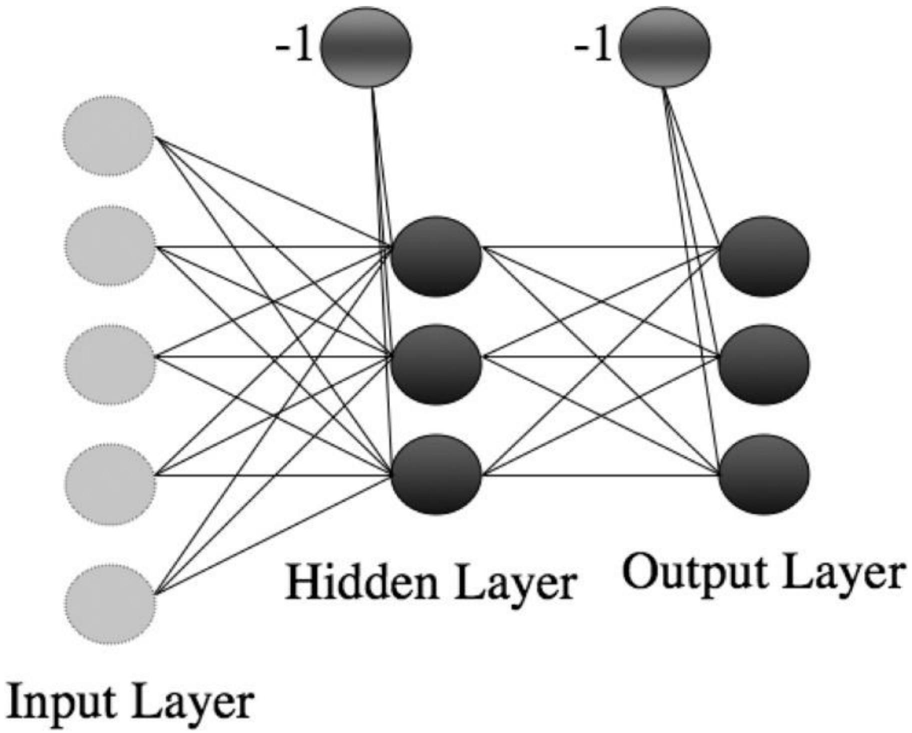
4. If $y = \sigma(z) = \frac{1}{1+e^{-\vec{w}\cdot\vec{x}}}$, then $y' = y(1-y)$

In addition

We will make use of the following which we have already seen:

- The logistic regression model
- Gradient descent
- GD applied to
  - Linear regression
  - Logistic regression
- Loss-functions:
  - MSE, Cross-Entropy

Training



- Given a set of training instances
  - $\{(\boldsymbol{x}_1, t_1), (\boldsymbol{x}_2, t_2),\ldots, (\boldsymbol{x}_N, t_N)\}$:
- Forwards:
  - Run them forwards and get predictions
    - $\{y_1, y_N, \ldots, y_N\}$
- Backwards
  - Use a suitable loss function and compare these to the target values
    - $\{t_1, t_2, \ldots, t_N\}$
  - Apply gradient descent to update the weights (partial derivatives)

How do we update the weights

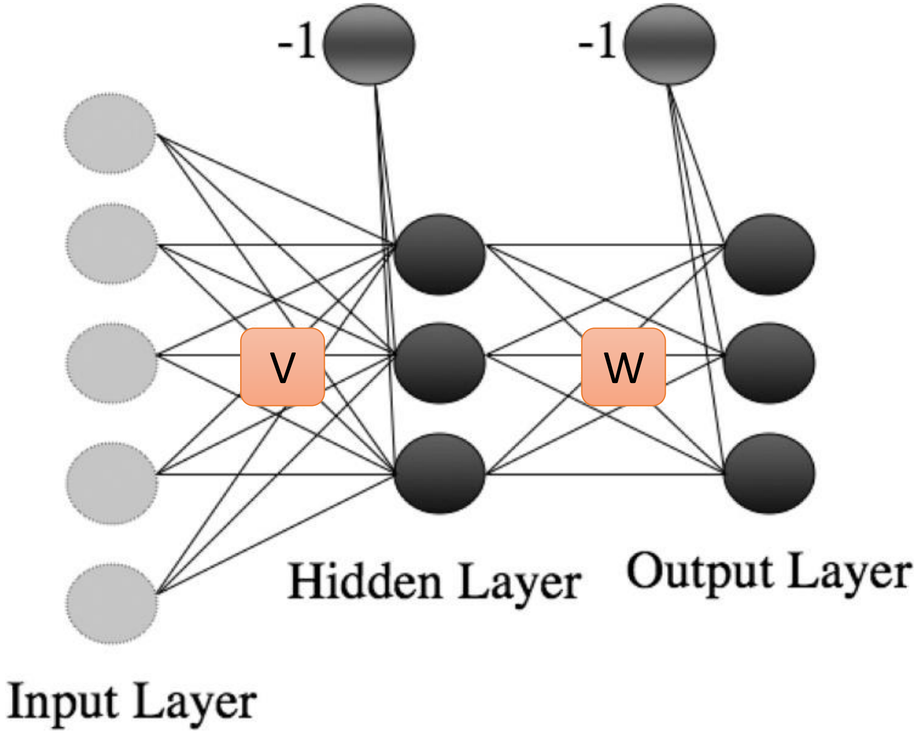
Last layer

- (easy)
- Like the same problems for linear regression or logistic regression without a hidden layer

The first layer

- The big question:
- How do we update the first layer?
- We don't have a loss (error) here

Solution: Backpropagation



- Let's be a little more formal
- Let the matrix **V** be the connections from *input* to *hidden* and **W** from *hidden* to *output*
  - $\dim(V) = ((m+1) \times k)$
  - $\dim(W) = ((k+1) \times n)$
- Activation functions:
  - Hidden layers: g
  - Hidden output layer: f

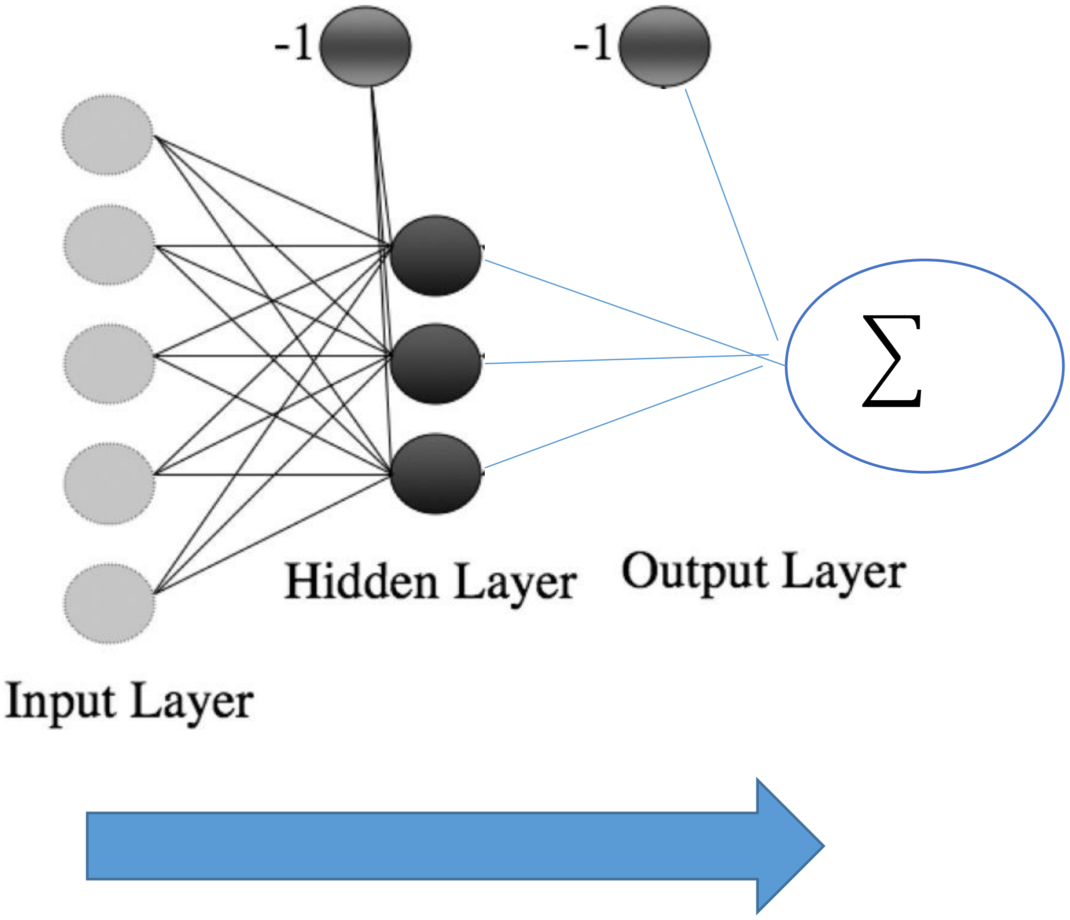
- Let us in the following consider SGD where we update for one input $\boldsymbol{x} = (x_1, x_2, \dots x_m)$

Forwards (notation)

- Add bias and send
  - $\boldsymbol{x}^{+} = (x_0, x_1, \dots x_m)$
- through the first layer to get
  - $\boldsymbol{h} = \boldsymbol{x}^{+}\boldsymbol{V} = (h_1, h_2, \dots, h_k)$, where
  - $h_j = \sum_{i=0}^{m} x_i v_{i,j}$
  - k is the number of hidden nodes
- Apply activation function to get
  - $\boldsymbol{a} = g(\boldsymbol{h}) = (a_1, a_2, \dots, a_k)$,
  - where $a_j = g(h_j)$
- Add bias and send
  - $\boldsymbol{a}^{+} = (a_0, a_1, a_2, \dots, a_k)$
- through the second layer to get
  - $\boldsymbol{z} = \boldsymbol{a}^{+}\boldsymbol{W} = (z_1, z_2, \dots, z_n)$, where
  - $z_j = \sum_{i=0}^{k} a_i w_{i,j}$
  - n is the number of output nodes
- Apply activation function to get
  - $\boldsymbol{y} = f(\boldsymbol{z}) = (y_1, y_2, \dots, y_n)$,
  - where $y_j = f(z_j)$

Backwards: 1.Regression

- We will consider various output tasks, starting with the simple regression
- There is only one output node
- The output activation function, f, is identity



Backwards: Update last layer

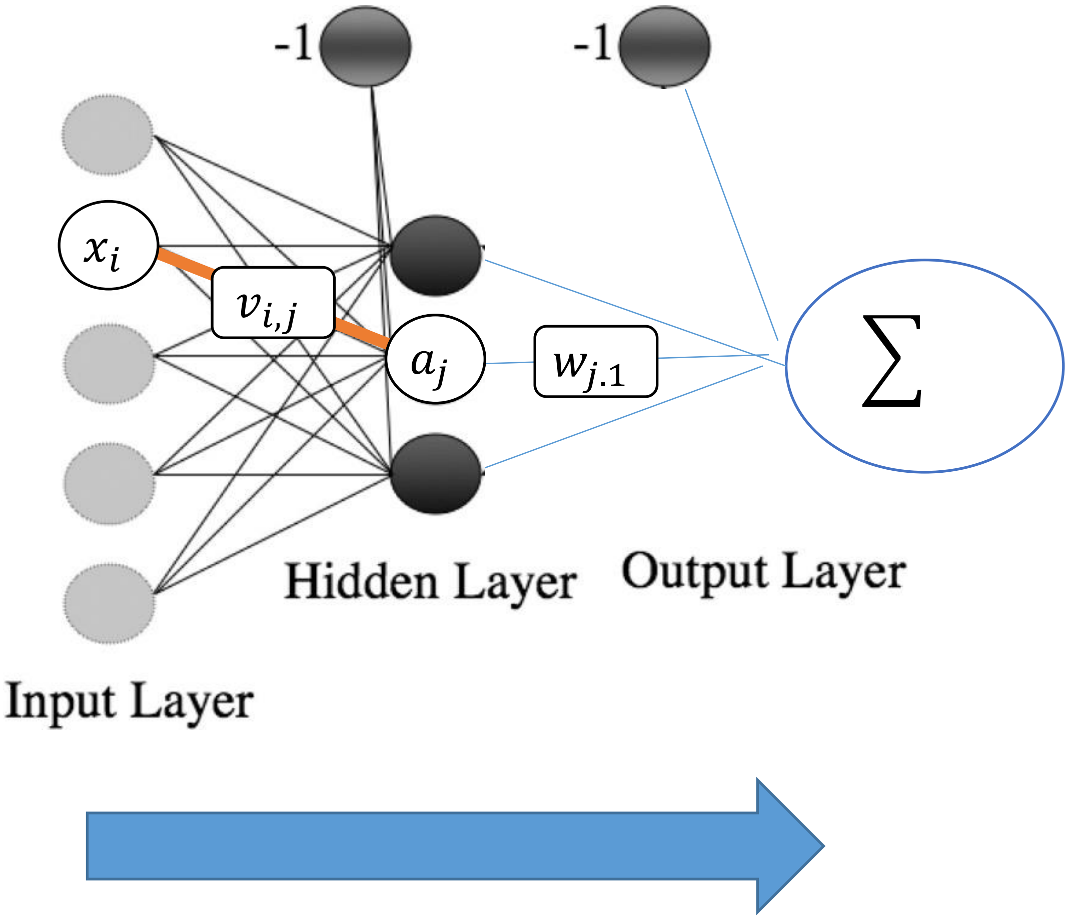
- For loss, we use MSE, or , as Marsland, the simpler
  Sum of Squares Error (SE): $L_{SE}(\mathbf{t}, \boldsymbol{y}) = \frac{1}{2}\sum_{j=1}^{N}\left(t_j - y_j\right)^2$
  - (The index j here, runs over the input items. There is only one output node)
- We have seen that
- $\frac{\partial}{\partial w_{i,1}} L_{SE}(\mathbf{t}, \boldsymbol{y}) = \frac{\partial}{\partial \boldsymbol{y}} L_{SE}(\mathbf{t}, \boldsymbol{y}) \left(\frac{\partial}{\partial w_{i,1}} \boldsymbol{y}\right) = \sum_{j=1}^{N}\left((t_j - y_j)(-a_{j,i})\right)$
- For SGD where we update for one input $\boldsymbol{x} = (x_1, x_2, \dots x_m)$
- $\frac{\partial}{\partial w_{i,1}} L_{SE}(\mathrm{t}, y) = \frac{\partial}{\partial \boldsymbol{y}} L_{SE}(\mathrm{t}, y) \left(\frac{\partial}{\partial w_{i,1}} y\right) = (t - y)(-a_i) = (y - t)(a_i)$

Backwards: Update last layer, ctd.

- $\frac{\partial}{\partial w_{i,1}} L_{SE}(\mathrm{t}, y) = (y - t)(a_i)$
- We know from lect. 6 how to update this ($\boldsymbol{a}$ corresponds to $\boldsymbol{x}$ then)
- But wait!
- We first have to find how to update the first layer.

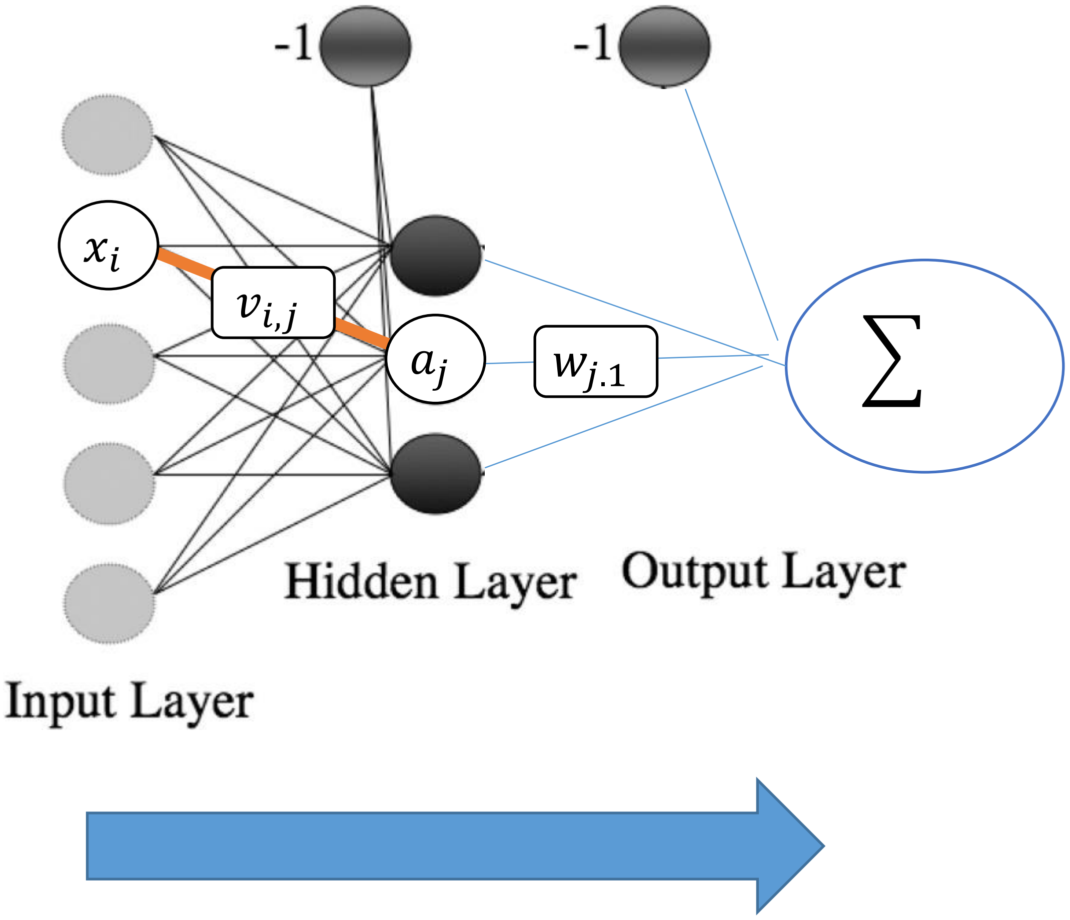
Backwards: Update first layer: V, 1

- $\boldsymbol{y} = f(\mathbf{z}) = \mathbf{z}$, where $\boldsymbol{z} = \boldsymbol{a}^{+}\boldsymbol{W}$
- $\boldsymbol{a} = g(\boldsymbol{h})$, where $\boldsymbol{h} = \boldsymbol{x}^{+}\boldsymbol{V}$
- $\frac{\partial}{\partial v_{i,j}} L_{SE}(\mathrm{t}, y)$=
- $\frac{\partial}{\partial \boldsymbol{a}} L_{SE}(\mathrm{t}, y)\left(\frac{\partial}{\partial v_{i,j}} \boldsymbol{a}\right) =$
- $\frac{\partial}{\partial a_j} L_{SE}(\mathrm{t}, y)\left(\frac{\partial}{\partial v_{i,j}} a_j\right)$
- because $\left(\frac{\partial}{\partial v_{i,j}} a_k\right) = \mathbf{0}$ for $k \neq j$



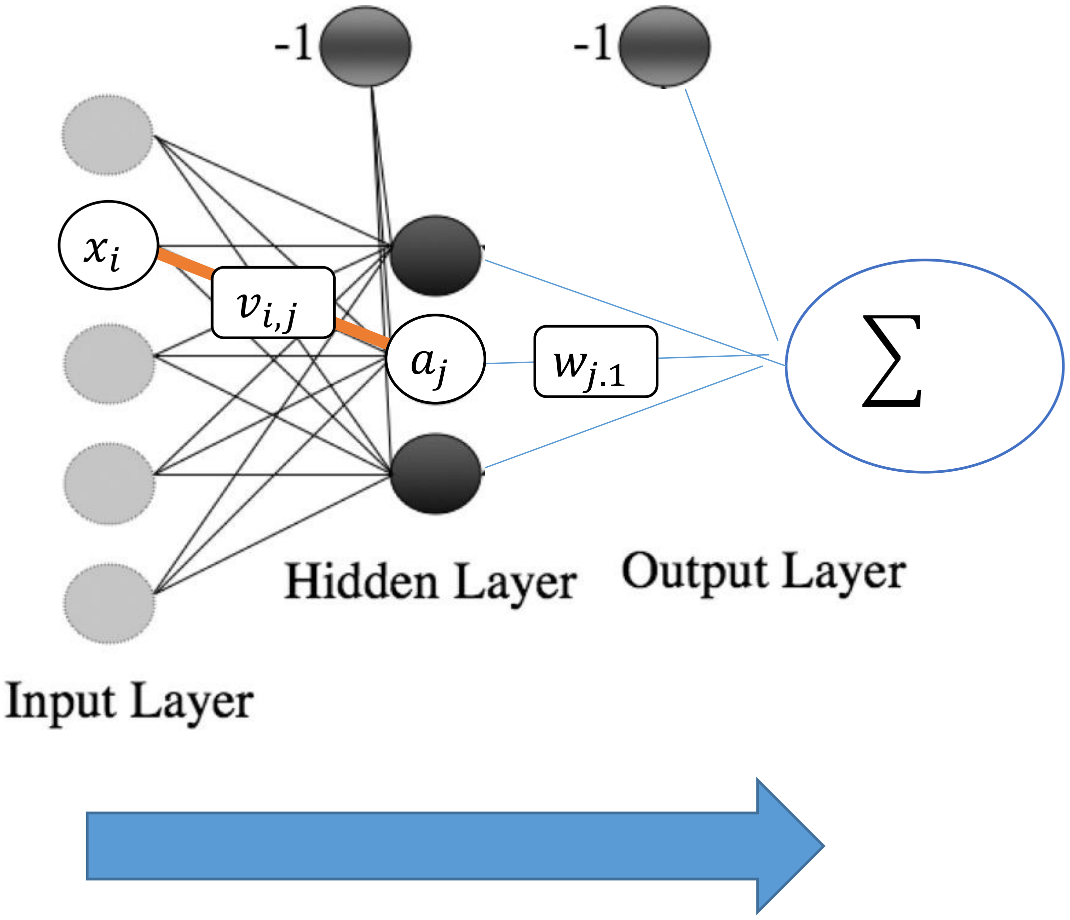
Backwards: Update first layer: V, 2

- $\boldsymbol{y} = f(\mathbf{z}) = \mathbf{z}$, where $\boldsymbol{z} = \boldsymbol{a}^{+}\boldsymbol{W}$
- $\frac{\partial}{\partial a_j} L_{SE}(\mathrm{t}, y) = \frac{\partial}{\partial \boldsymbol{y}} L_{SE}(\mathrm{t}, y) \left(\frac{\partial}{\partial a_j} y\right) = (t - y)(-w_{j,1}) = (y - t)(w_{j,1})$
- Observe similarities and <u>differences</u> to
- $\frac{\partial}{\partial w_{i,1}} L_{SE}(\mathrm{t}, y) = (y - t)(a_i)$
- We call the common part: $(y - t)$ for the delta term, $\delta_o(\kappa)$ of the end node κ.



Backwards: Update first layer: V, 3

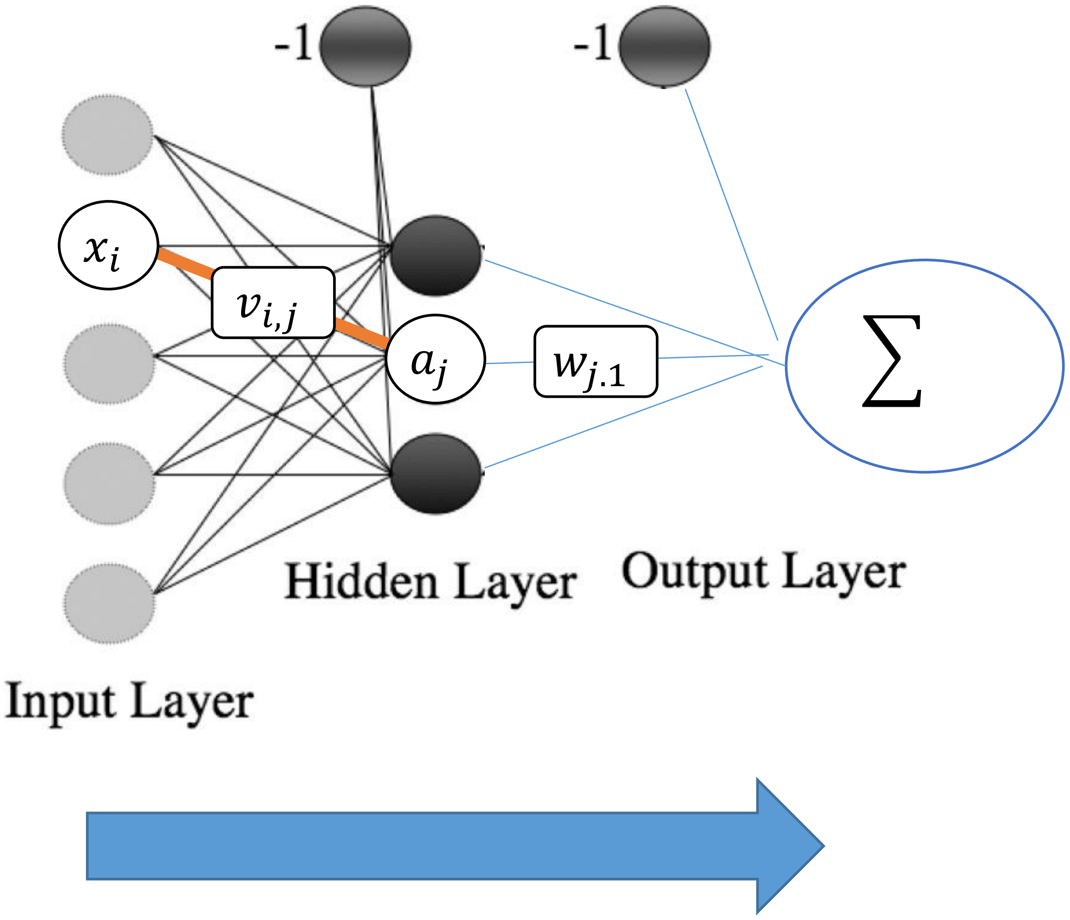
- $\boldsymbol{a} = g(\boldsymbol{h})$, where $\boldsymbol{h} = \boldsymbol{x}^{+}\boldsymbol{V}$
- $\left(\frac{\partial}{\partial v_{i,j}} a_j\right) = \left(\frac{\partial}{\partial \boldsymbol{h}} g\right)\left(\frac{\partial}{\partial v_{i,j}} \boldsymbol{h}\right)=$
  $= \left(\frac{\partial}{\partial h_j} g\right)\left(\frac{\partial}{\partial v_{i,j}} h_j\right)$
- $\frac{\partial}{\partial v_{i,j}} h_j = x_i$
- If $a_j = g(h_j) = \sigma(h_j)$, then
  - $\left(\frac{\partial}{\partial h_j} g\right)= a_j(1 - a_j)$
  - $\left(\frac{\partial}{\partial v_{i,j}} a_j\right) = a_j(1 - a_j)x_i$



Backwards: Update first layer: V, 4

- $\boldsymbol{y} = f(\boldsymbol{z}) = \mathbf{z}$, where $\boldsymbol{z} = \boldsymbol{a}^{+}\boldsymbol{W}$
- $\boldsymbol{a} = g(\boldsymbol{h})$, where $\boldsymbol{h} = \boldsymbol{x}^{+}\boldsymbol{V}$
- $\frac{\partial}{\partial v_{i,j}} L_{SE}(\mathrm{t}, y) = \frac{\partial}{\partial a_j} L_{SE}(\mathrm{t}, y) \left(\frac{\partial}{\partial v_{i,j}} a_j\right) =$
- $\underbrace{\overbrace{(y-t)}^{\delta_o(\kappa)}(w_{j,1})a_j(1-a_j)x_i}_{\delta\text{-term at the node marked with } a_j}$

δ-term at the node marked with a_j

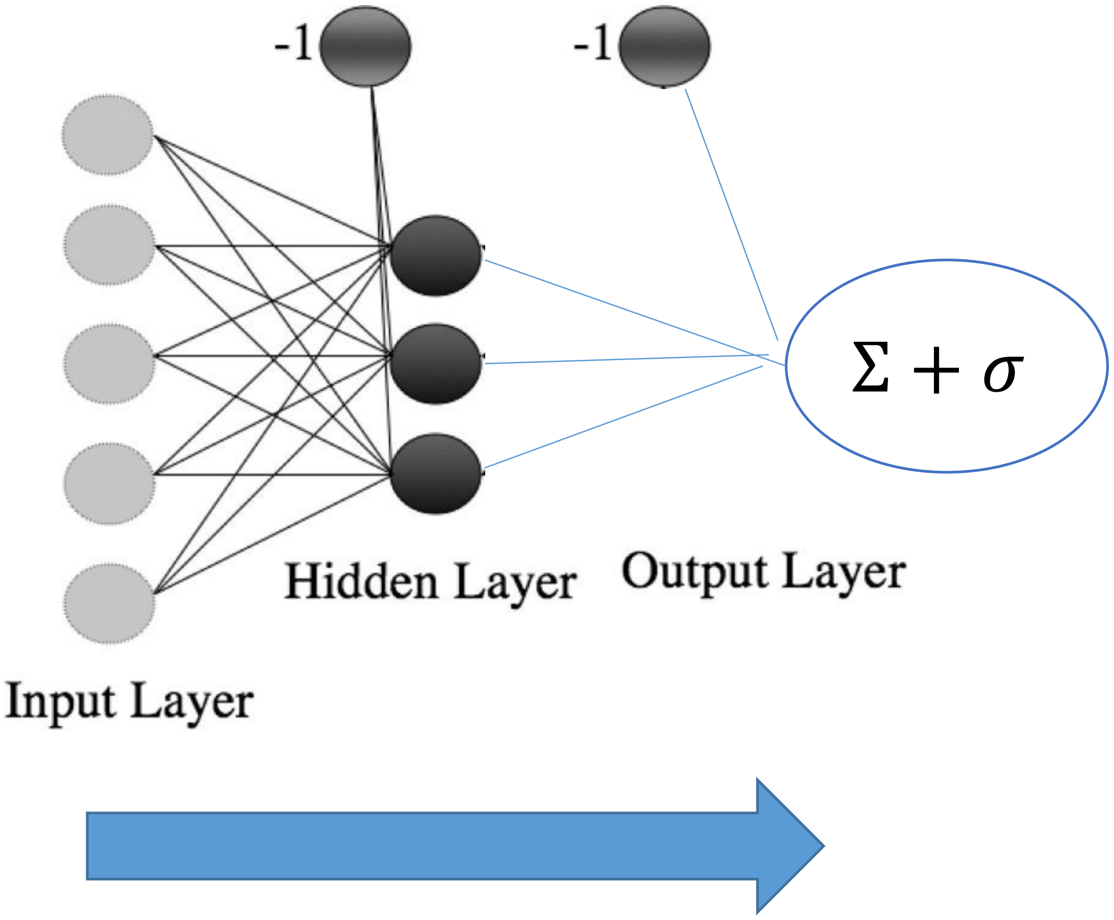


Putting it together: the Algorithm

- Use the loss function and the derivative of the activation function to compute the delta term at the final node(s), here: $\delta_o(\kappa_1) = \ (y - t)$
- Compute the delta terms for each node in the hidden layer, from the delta term(s) and the hidden layer and the weights at the connections
  - here: $\delta\left(hidden_j\right) = \delta_o(\kappa_1)\left(w_{j,1}\right)a_j(1 - a_j)$
- Update the weights by the deltas:
  - $w_{i,1} = w_{i,1} - \eta\delta_o(\kappa_1)a_i$
  - $v_{i,j} = v_{i,j} - \eta\delta\left(hidden_j\right)x_i$

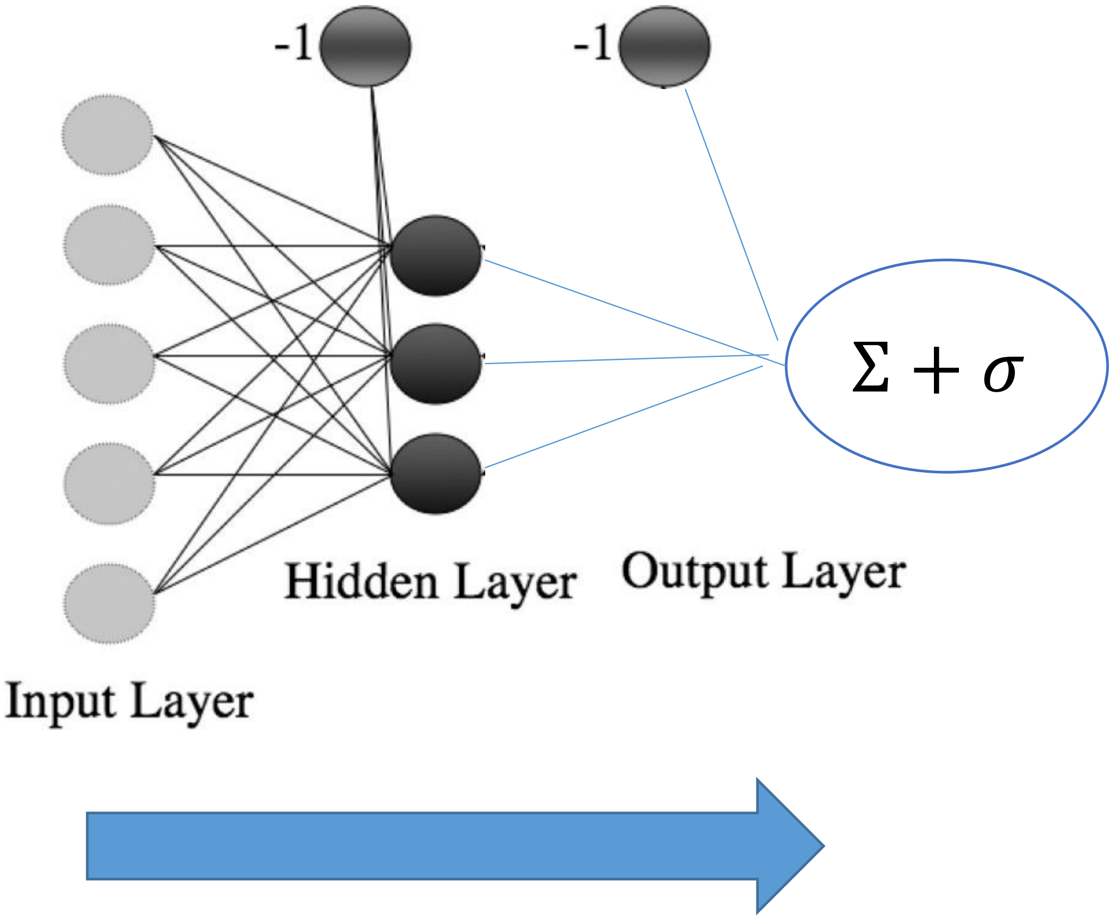
2. Binary classification, take one

- Like Marsland, and regression, for loss use (SE): $L_{SE}(\mathbf{t}, \boldsymbol{y}) = \frac{1}{2}\sum_{j=1}^{N}\left(t_j - y_j\right)^2$
- The only difference to regression is the logistic activation function: $y = \sigma(x) = \frac{1}{1+e^{-x}}$
- Since the derivative of this is $y(1-y)$, we get
- $\delta_o(\kappa_1) = (y-t)y(1-y)$
- The rest is as for regression



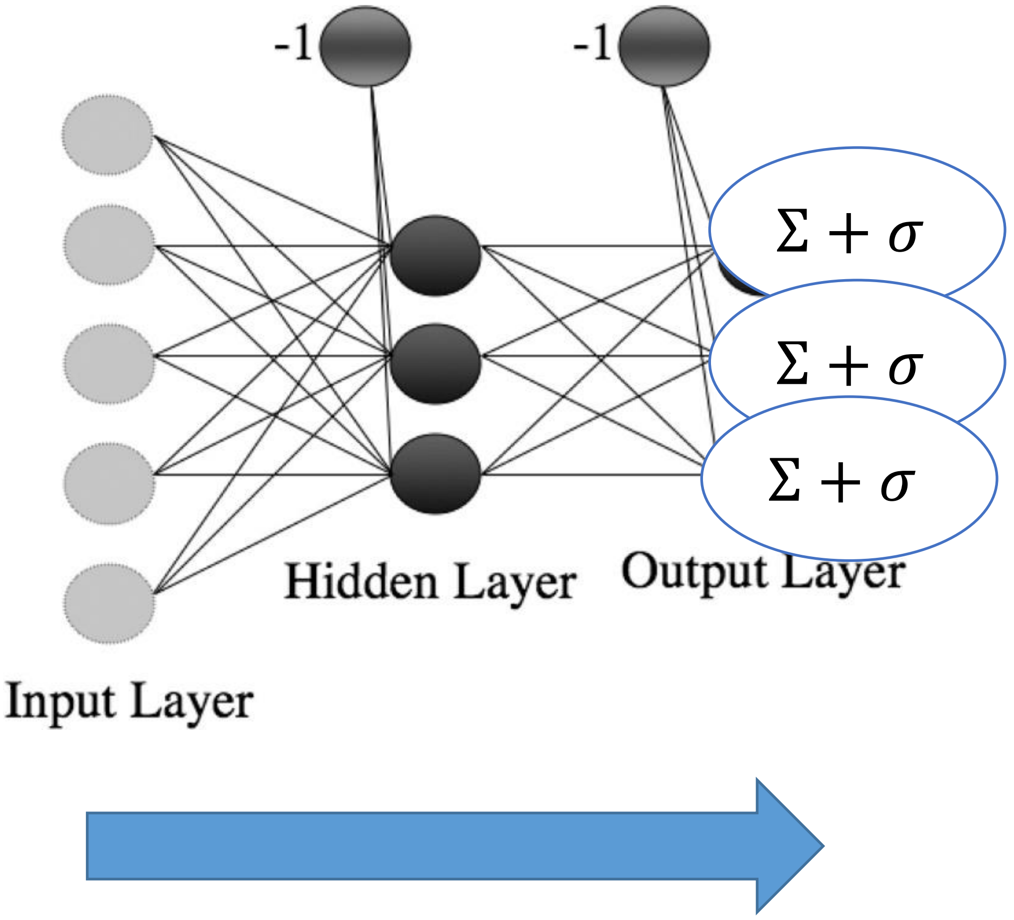
2. Binary classification, take two

- Use instead cross-entropy loss (cf. Lecture 7, Marsland 4.6.6)
- $\frac{\partial}{\partial y} L_{CE}(\mathrm{t}, y) = -\frac{(t-y)}{y(1-y)}$
- Logistic activation
- $\delta_o(\kappa_1) = -\frac{(t-y)}{y(1-y)} y(1-y) = (y-t)$
- The rest is as for regression and take one



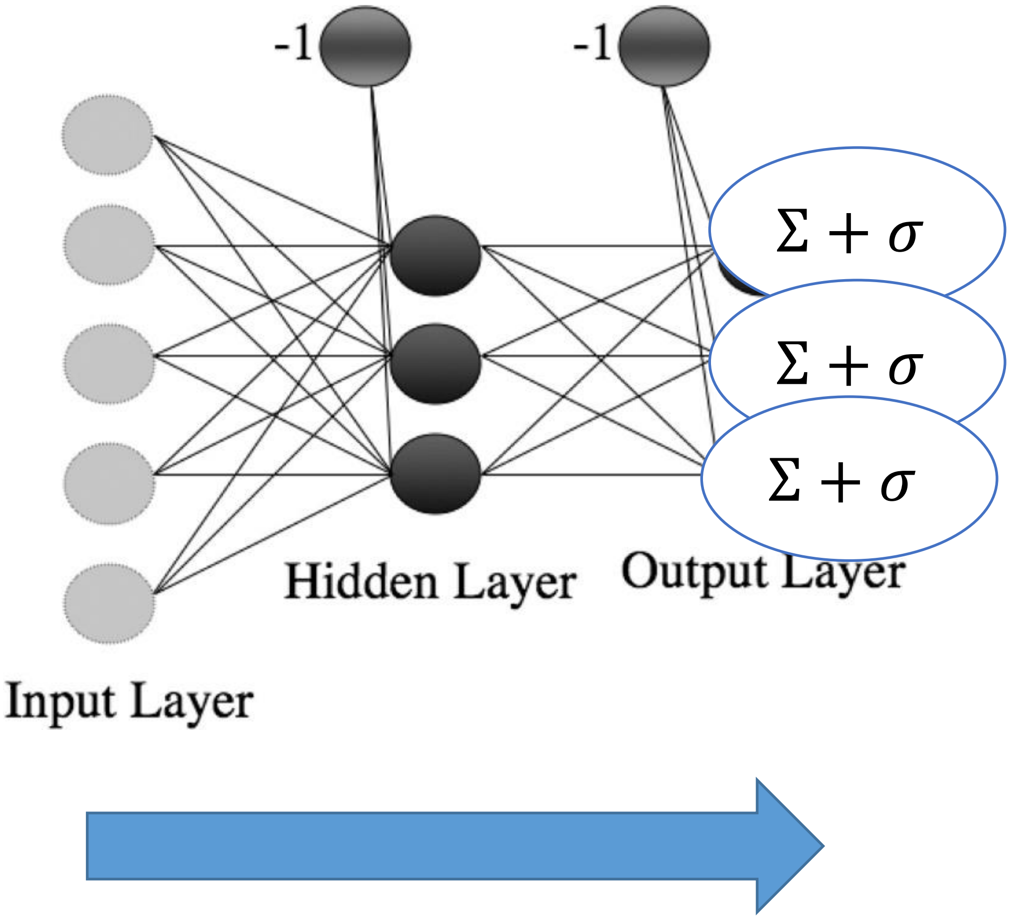
3. Multi-label classification

- Several output nodes
- Logistic activation function
- The model Marsland considers
- $L_{SE}(\mathbf{t}, \boldsymbol{y}) = \frac{1}{2} \sum_{j=1}^{N} (t_j - y_j)^2$
  - (The index j here, runs over the output nodes.)
  - We still look at one input only



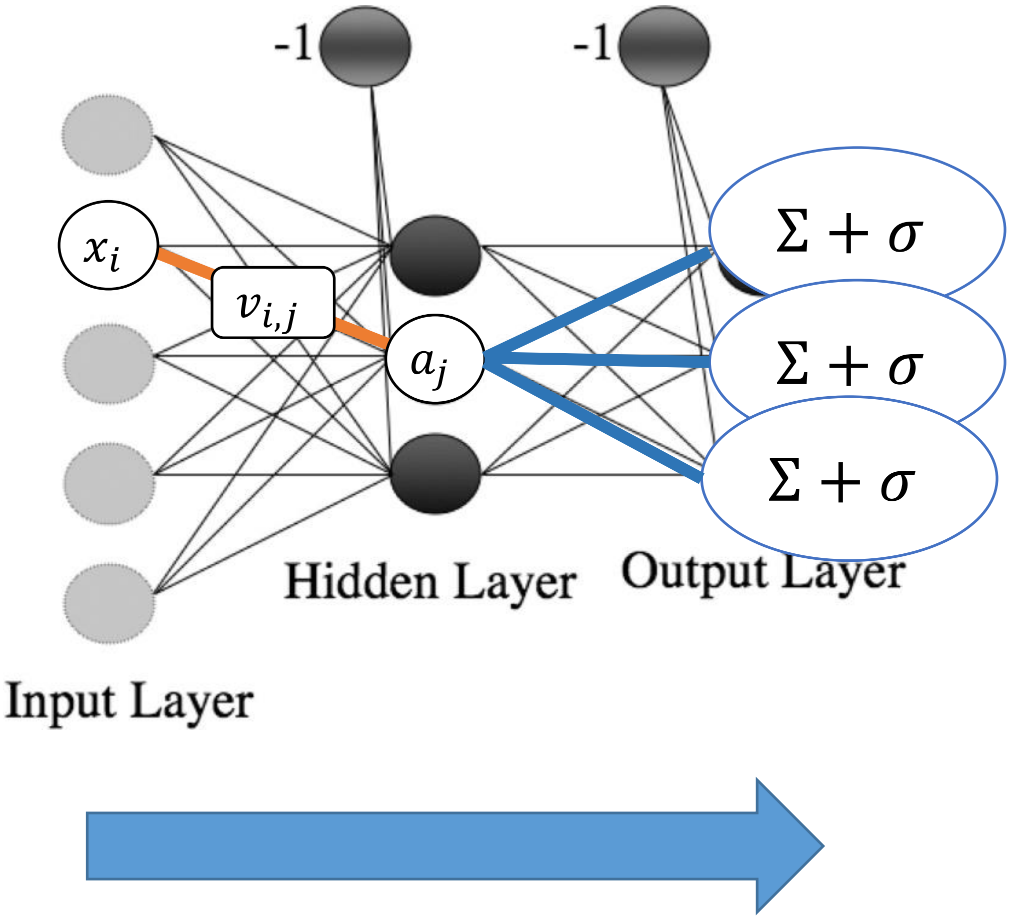
3. Multi-label classification

- (SE loss, logistic output activation)
- We compute a delta term at each output node, κ_j:
- $\delta_o(\kappa_j) = (y_j - t_j)y_j(1 - y_j)$



3. First layer

- (SE loss,
  logistic output activation)
- $\delta(hidden_j) =$
- $a_j(1 - a_j)\sum_{i=1}^{n} \delta_o(\kappa_i) w_{j,i}$
- i.e., sum of delta at output weighted by the connections between them
- The rest as for the others

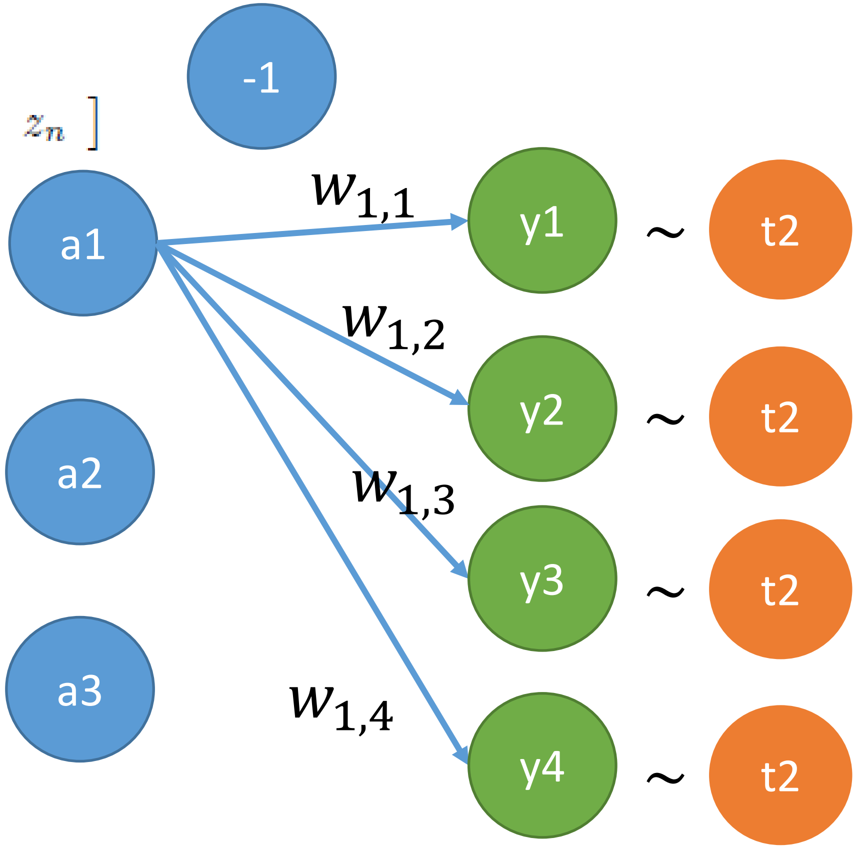


Putting it together: the Algorithm

- Use the loss function and the derivative of the activation function to compute the delta term at the final node(s),
  - here: $\delta_o(\kappa_j) = (y_j - t_j)y_j(1 - y_j)$ for each node κ_j for $j = 1, \dots, n$
- Compute the delta terms for each node in the hidden layer,
  - here: $\delta(hidden_j) = a_j(1 - a_j)\sum_{i=1}^{n} \delta_o(\kappa_i)w_{j,i}$ for $j = 1, \dots, k$
- Update the weights by the deltas in both layers
  - $w_{i,j} = w_{i,j} - \eta\delta_o(\kappa_j)a_i$
  - $v_{i,j} = v_{i,j} - \eta\delta(hidden_j)x_i$

By the way:

$$\begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_m \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} z_1 & z_2 & \cdots & z_n \end{bmatrix}$$



- To calculate $\sum_{j=1}^{m} w_{l,j}\delta_j$ by matrices, use
- $[\delta(\kappa_1), \delta(\kappa_2), \ldots \delta(\kappa_n)]\boldsymbol{W}^T$

Congratulation!

- You just survived backpropagation!
- You now deserve a break and cake!

8.4 Finer details

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

Practical advices

- Scaling
- Initializing the weights
- Local minima
- Early stopping
- Batch, stochastic, mini-batch
- Number of hidden nodes and hidden layers?
- Activation functions

Scaling

- The $\mathrm{z} = \boldsymbol{w} \cdot \boldsymbol{x}$ shouldn't be too large for this to work, roughly $|z|$ shouldn't be much more than 1
- For example, normalization (scikit: standardscaler) of each feature

Normalization

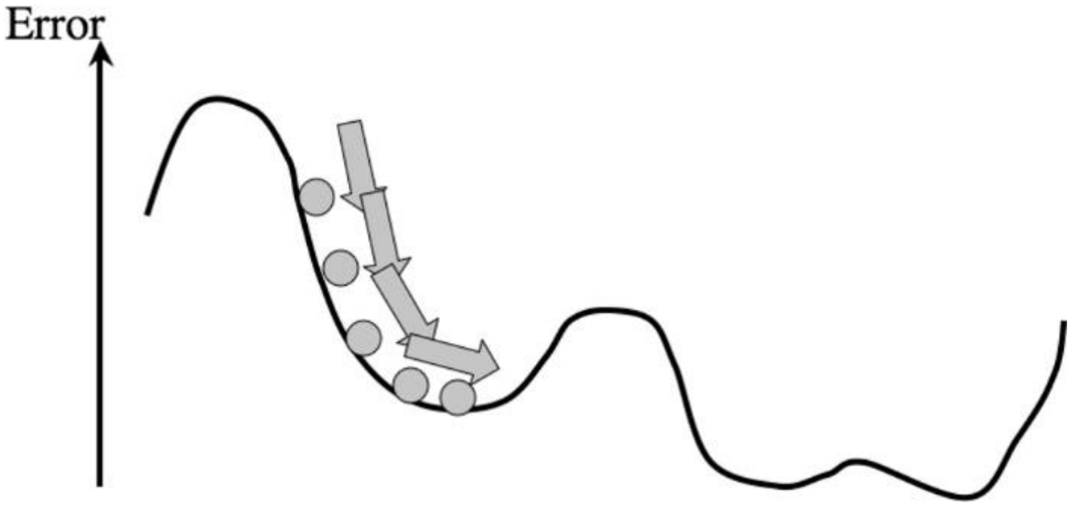
- Training data, dimension i: $X_i = \{x_{1i}, x_{2i}, \dots x_{Ni}\}$.
- Let m be the mean value:
  - $m = \frac{1}{N}\sum_{j=1}^{N} x_{j,i}$
- Let s be the standard deviation
- Define $scale_i(x_{ji}) = \frac{x_{ji} - m}{s}$
- Use the same scaler on all test data!

Initializing the weights

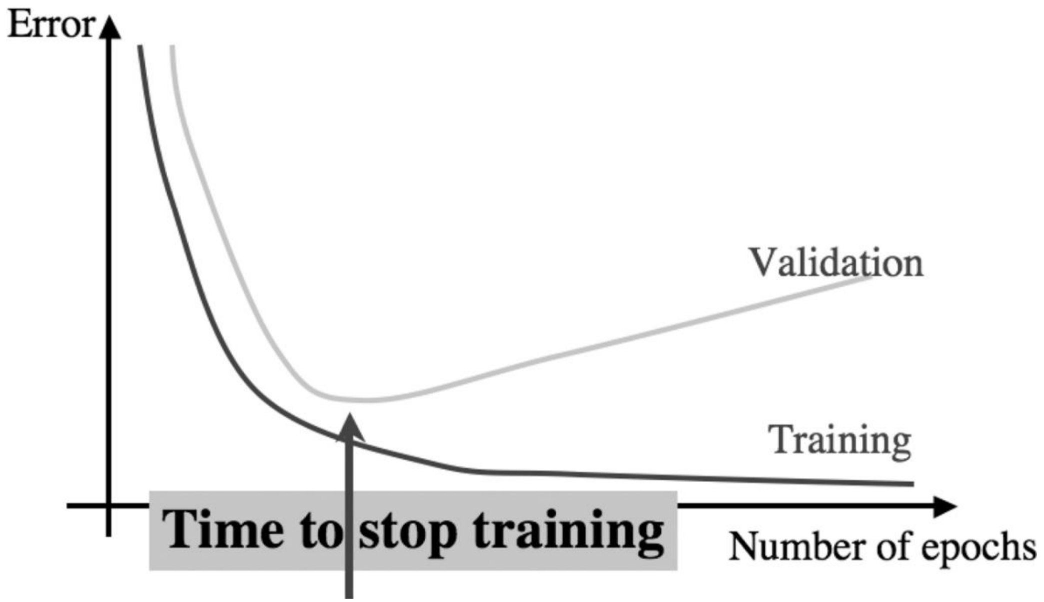
- The weights:
  - should not be initialized to 0
  - should be initialized to random numbers
  - should be initialized to numbers between -1 and 1
- In addition, Marsland recommends to multiply with $\frac{1}{\sqrt{m}}$
  - where m is the number of input nodes

Local minima

- The loss function for MLP is not convex
- It can be caught in local minima
- Hence:
  - Make several runs with different initializations and compare the results (mean and std.dev.)
  - Consider methods for escaping local minima, cf. lecture 2 and adding momentum



Early stopping

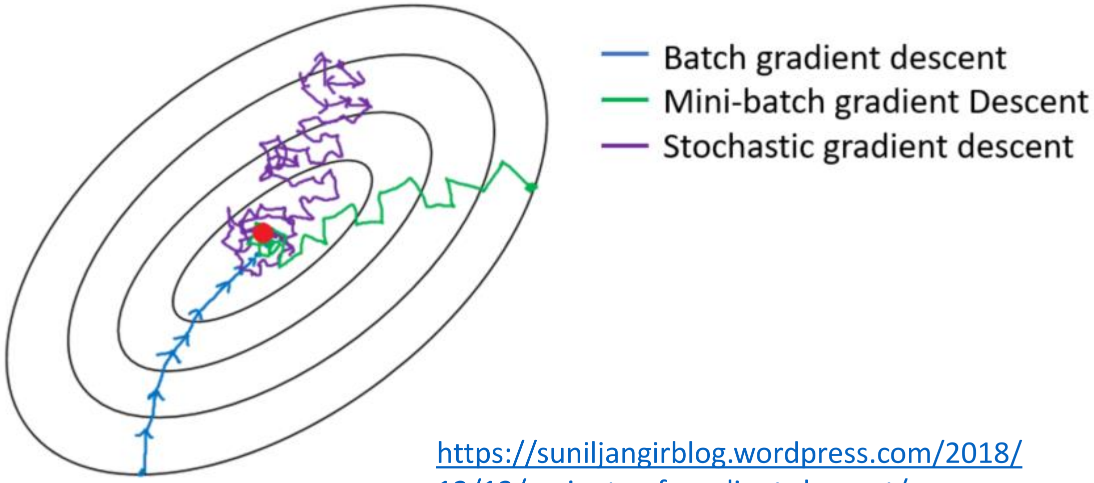


- The loss on the training data will decrease during training
- There is a danger of overfitting by training for too long:
  - The network knows the training set very well
  - but does not generalize

- Use a validation set V different from the training set.
- After *k* rounds for some fixed *k* (e.g., 100):
  - check the loss on V
  - if the loss starts to increase, stop training!

Variations of gradient descent

- Mini-batch training:
  - Pick a subset of the training set of a certain size
  - Calculate the loss for this subset
  - Make one move in the direction of this gradient
  - Repeat (an epoch)
- Batch training
  - Use the whole training set in each epoch
- Stochastic gradient descent:
  - Pick one datapoint at random and use in each epoch

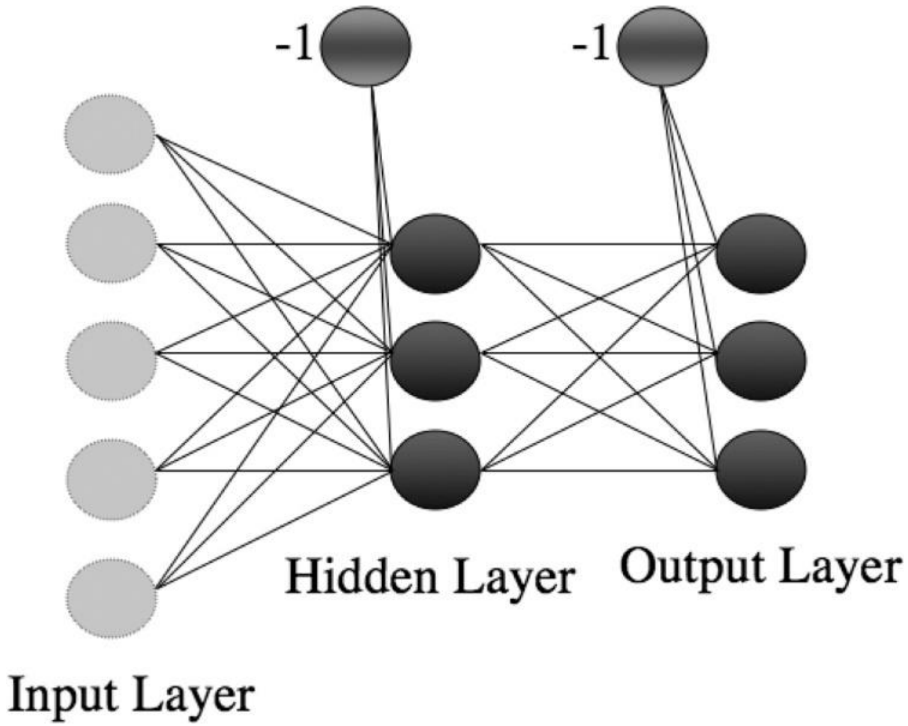


https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

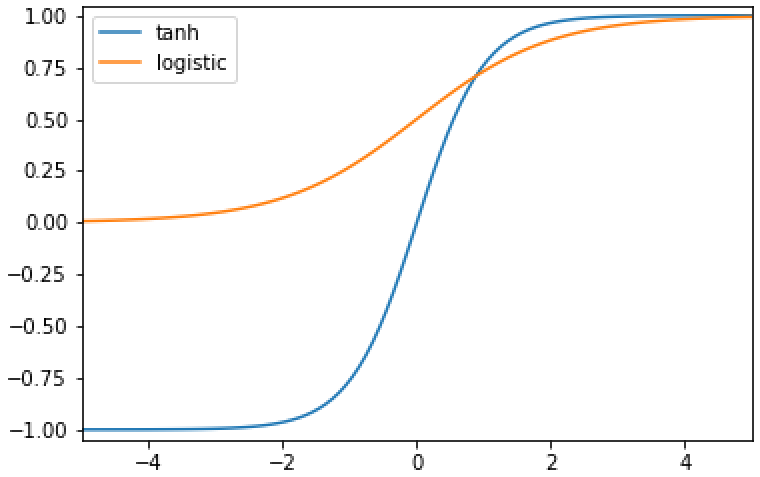
- SGD/Mini-batch can be a way to avoid local minima

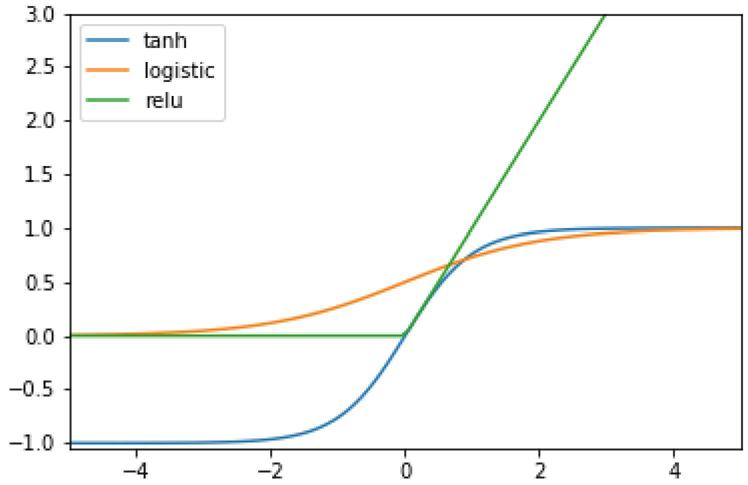
Number of hidden nodes and hidden layers?

- Very much an empirical question
- Use an independent validation set
- Run with different settings and evaluate on the validation set
- Choose the settings which give the best result
- Called hyper-parameter tuning
  - (The hyper-parameters are the parameters that you have to set.)

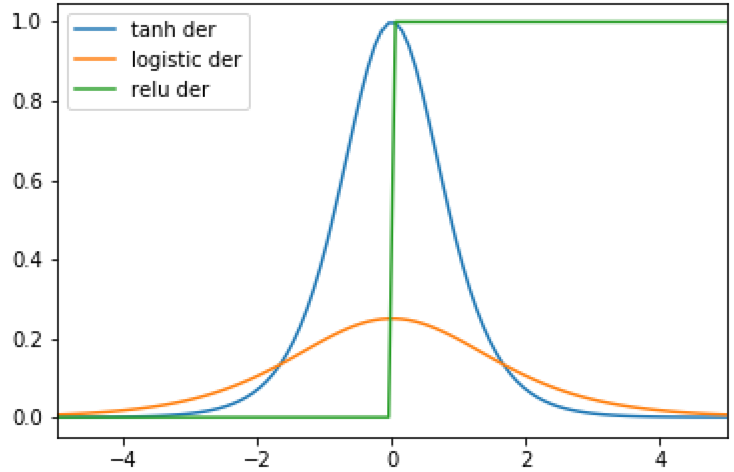


Alternative activation functions in the hidden layer

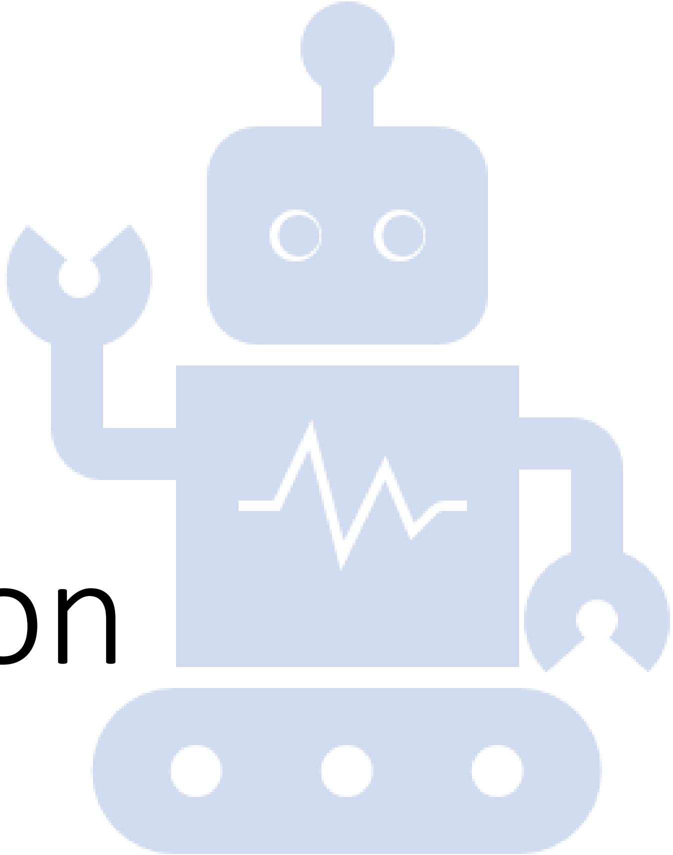




- There are alternative activation functions
- One may use different functions at different layers
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- $ReLU(x) = \max(x, 0)$
- ReLU is the preferred method in deep networks



8.5 More on evaluation



IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

Evaluation measures

		Is in C	
		Yes	NO
Classifier	Yes	tp	fp
	No	fn	tn

- Accuracy: (tp+tn)/N
- Precision:tp/(tp+fp)
- Recall: tp/(tp+fn)

- F-score combines P and R
- $F_1 = \frac{2PR}{P+R} \left(= \frac{1}{\frac{\frac{1}{R}+\frac{1}{P}}{2}} \right)$
- F_1 called "harmonic mean"
- General form
  - $F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$
  - for some $0 < \alpha < 1$

Confusion matrix

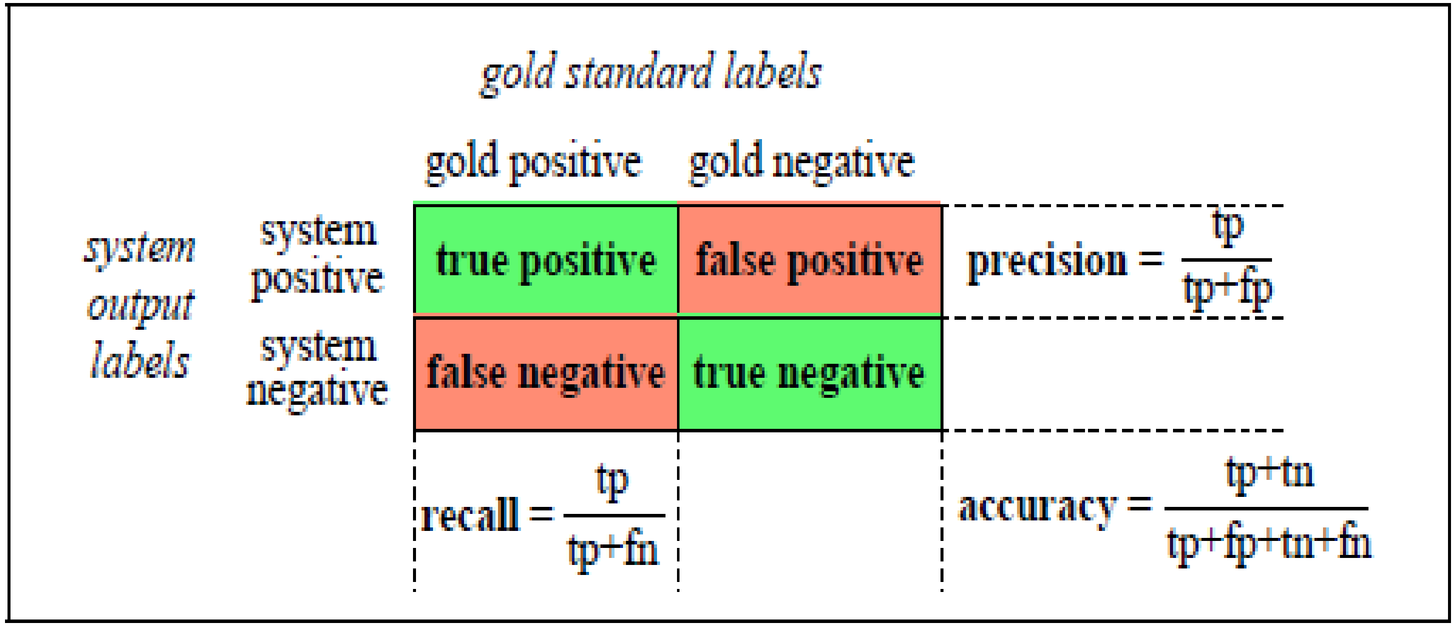


Figure 6.4 Contingency table

- Beware what the rows and columns are:
  - Marsland swaps them

Confusion matrix

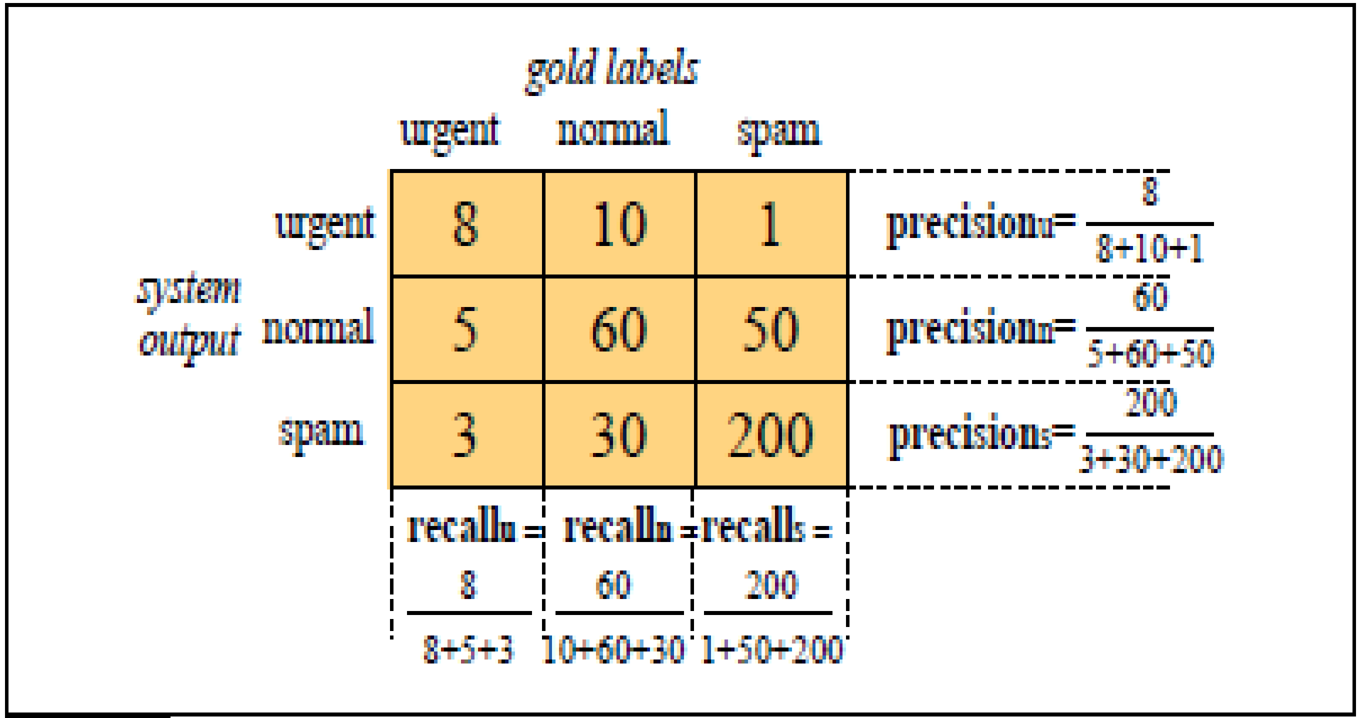


Figure 6.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

- Precision, recall and f-score can be calculated for each class against the rest