ANN

or

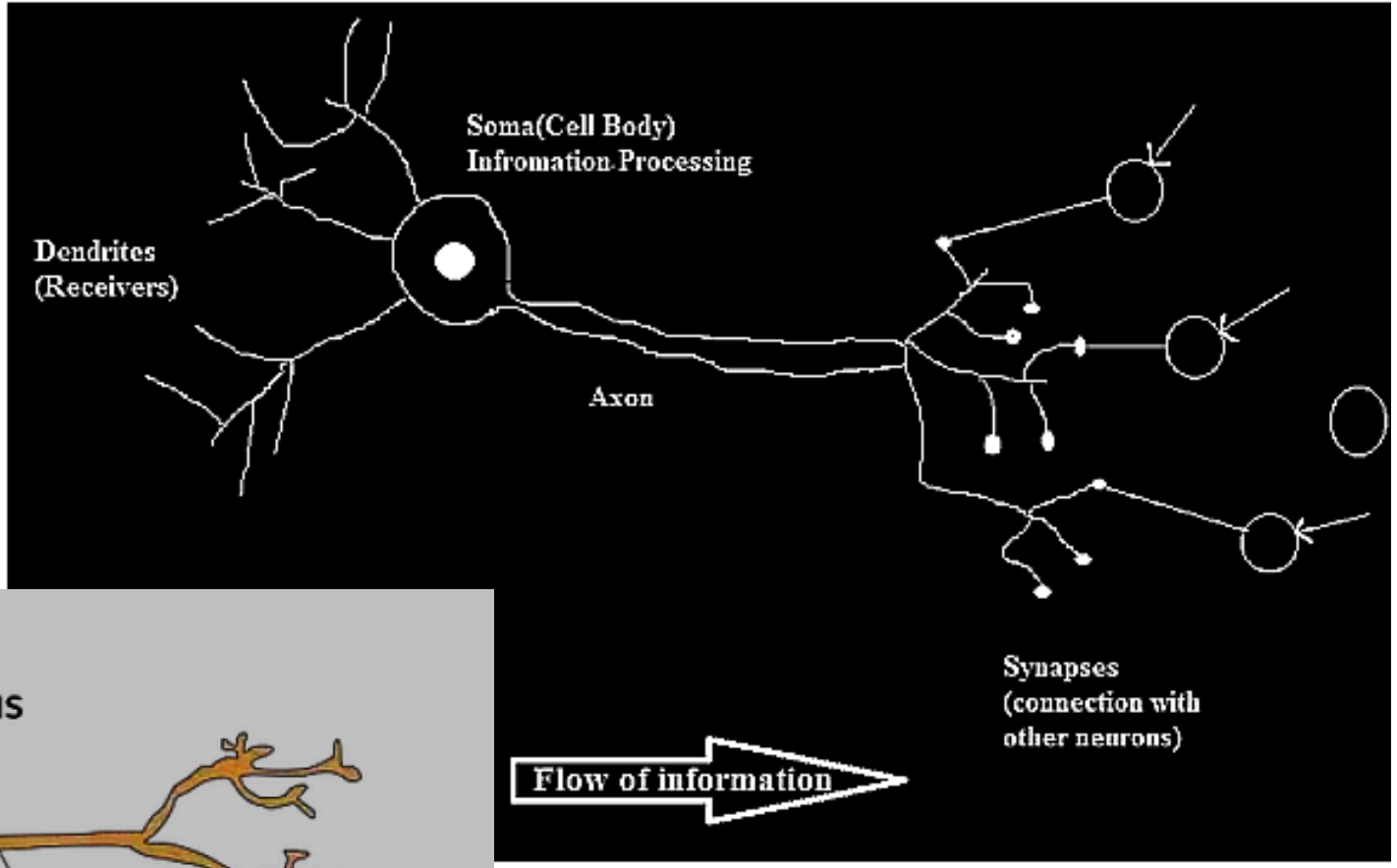
MLP

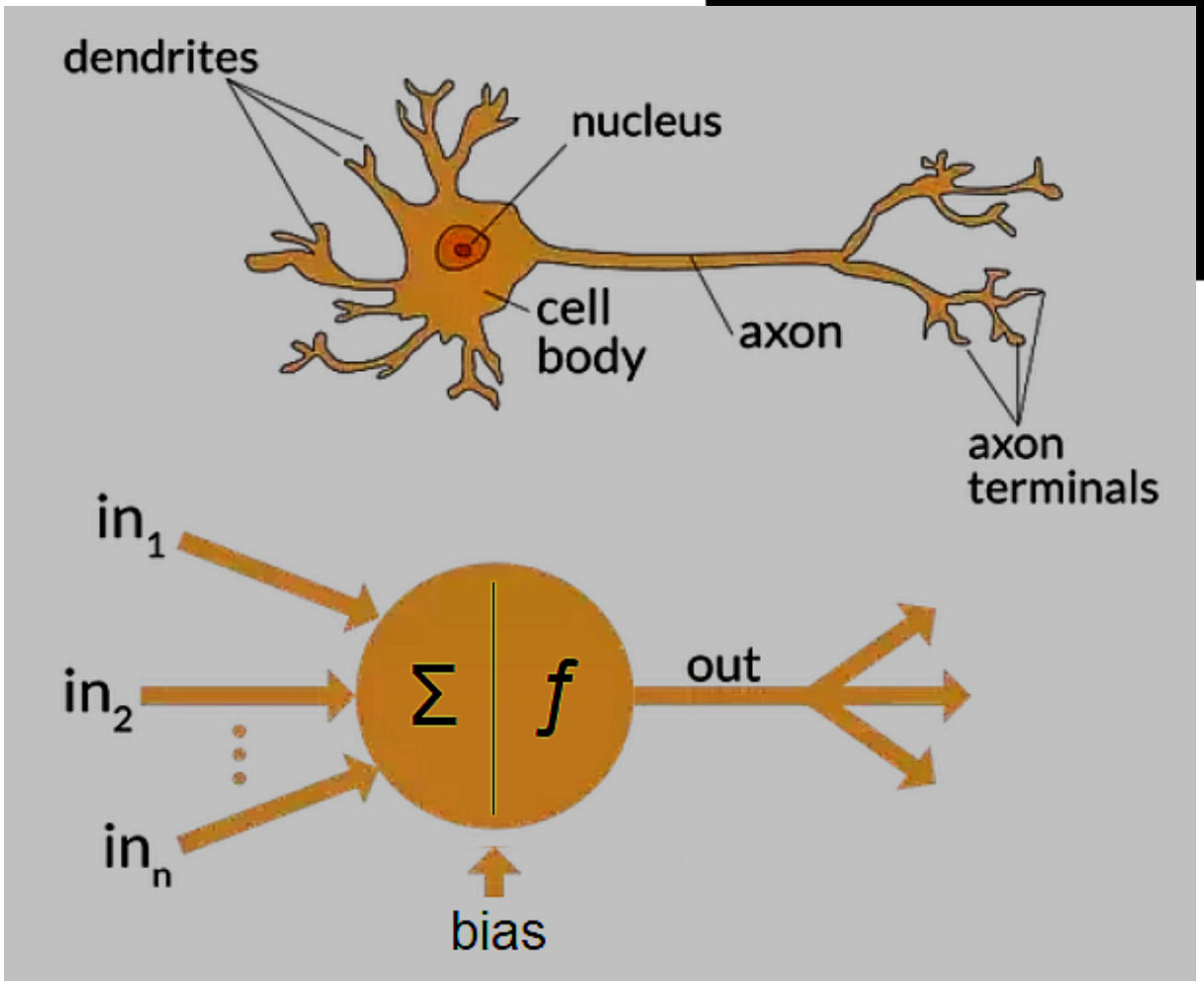
Feed-forward Network Functions

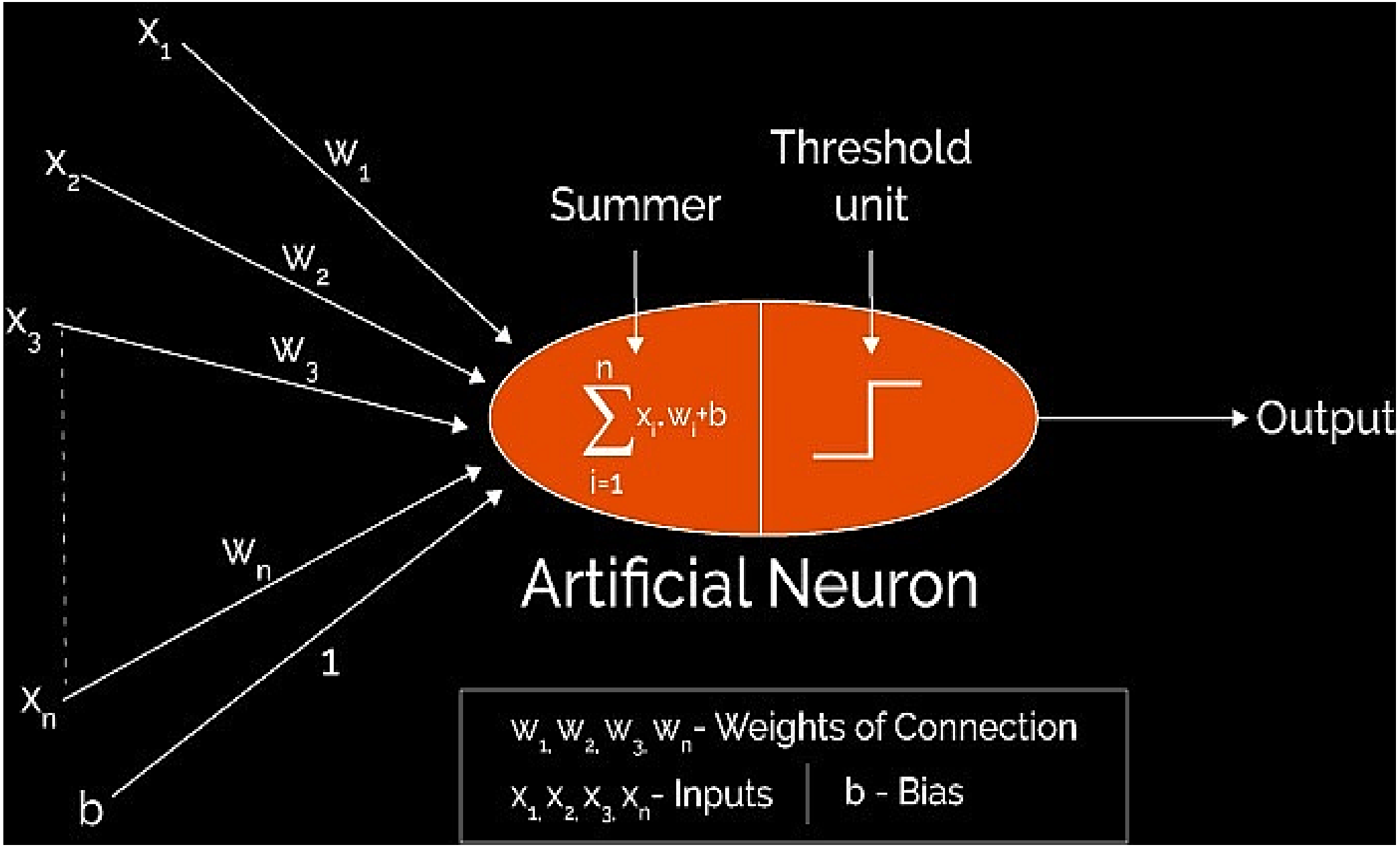
- The linear models for regression and classification are based on linear combinations of fixed nonlinear basis functions $\phi_j(\mathbf{x})$ and take the form

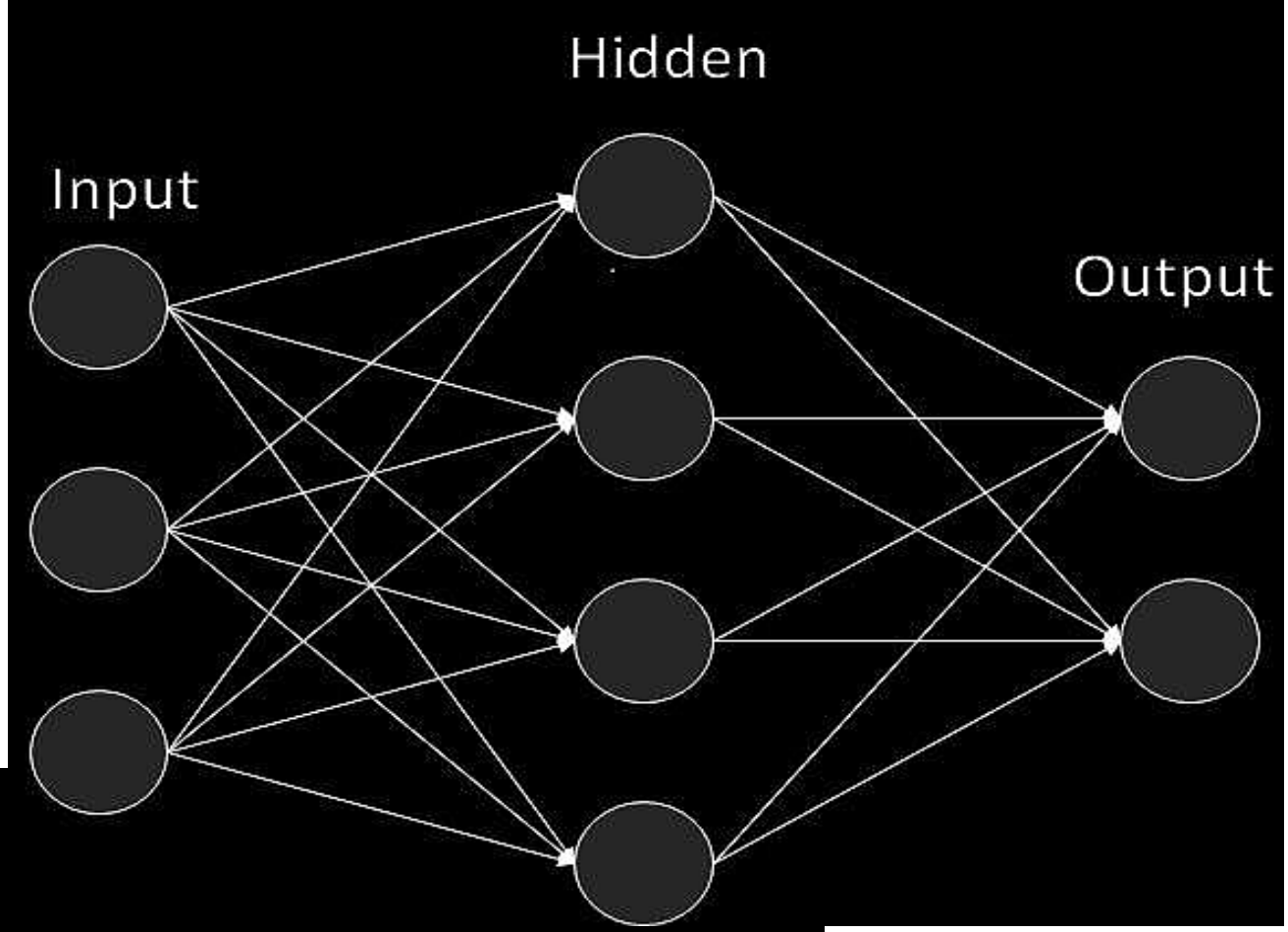
$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right)$$

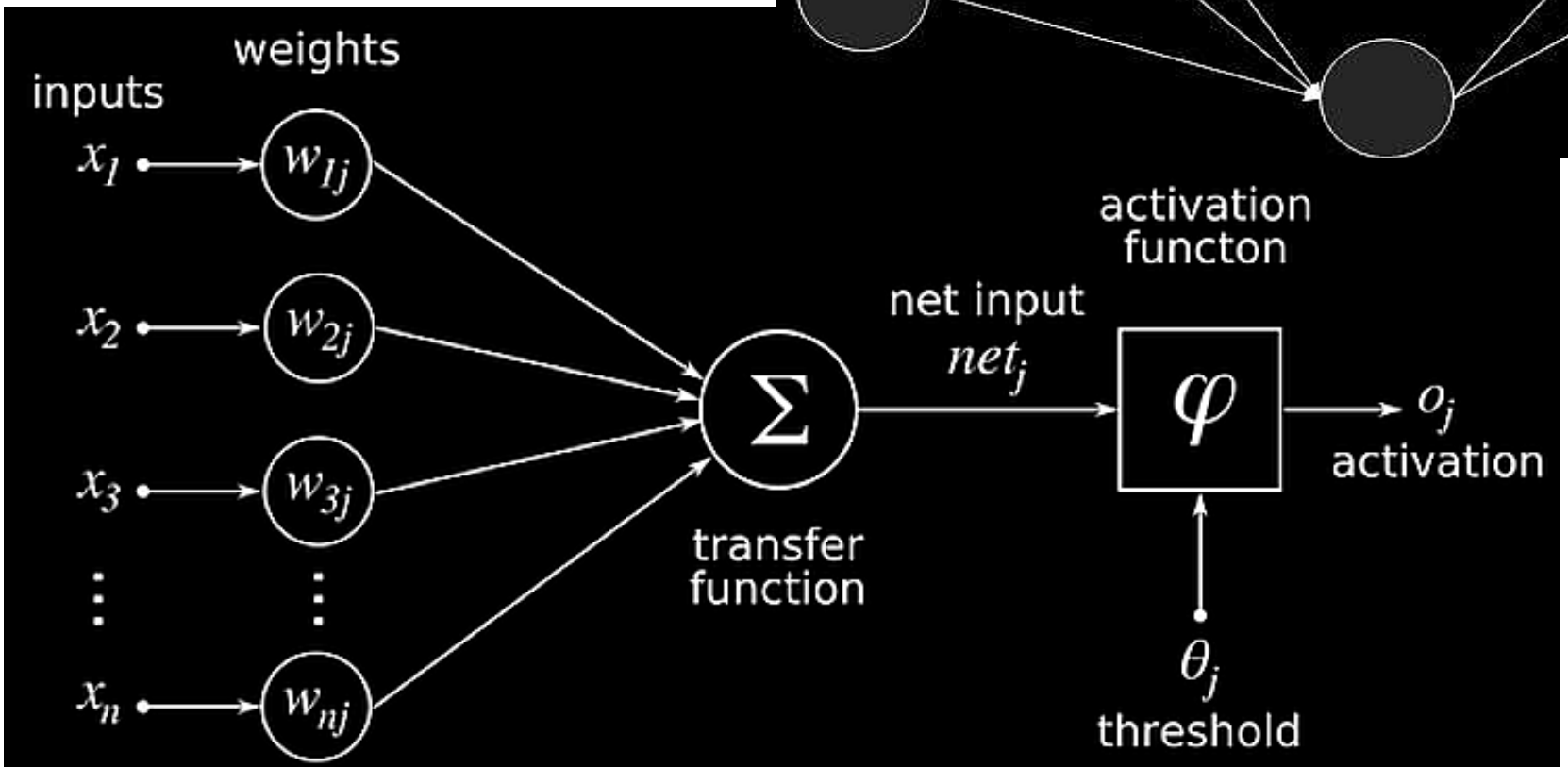
- where $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression.
- Our goal is to extend this model by making the basis functions $\phi_j(\mathbf{x})$ depend on parameters and then to allow these parameters to be adjusted, along with the coefficients $\{w_j\}$, during training.

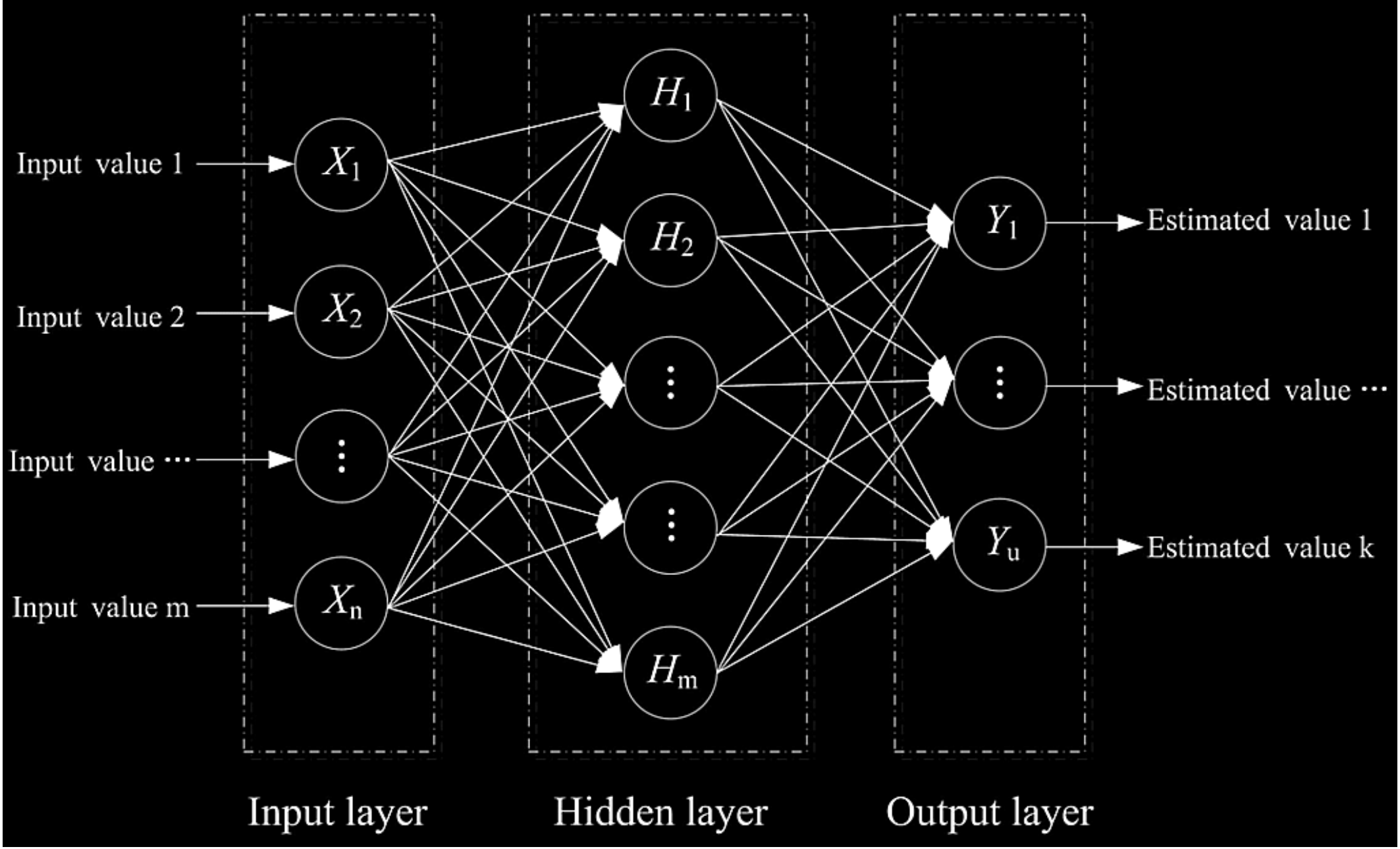

Soma(Cell Body)
Infromation Processing
Dendrites
(Receivers)
Axon
Synapses
(connection with
other neurons)
Flow of information

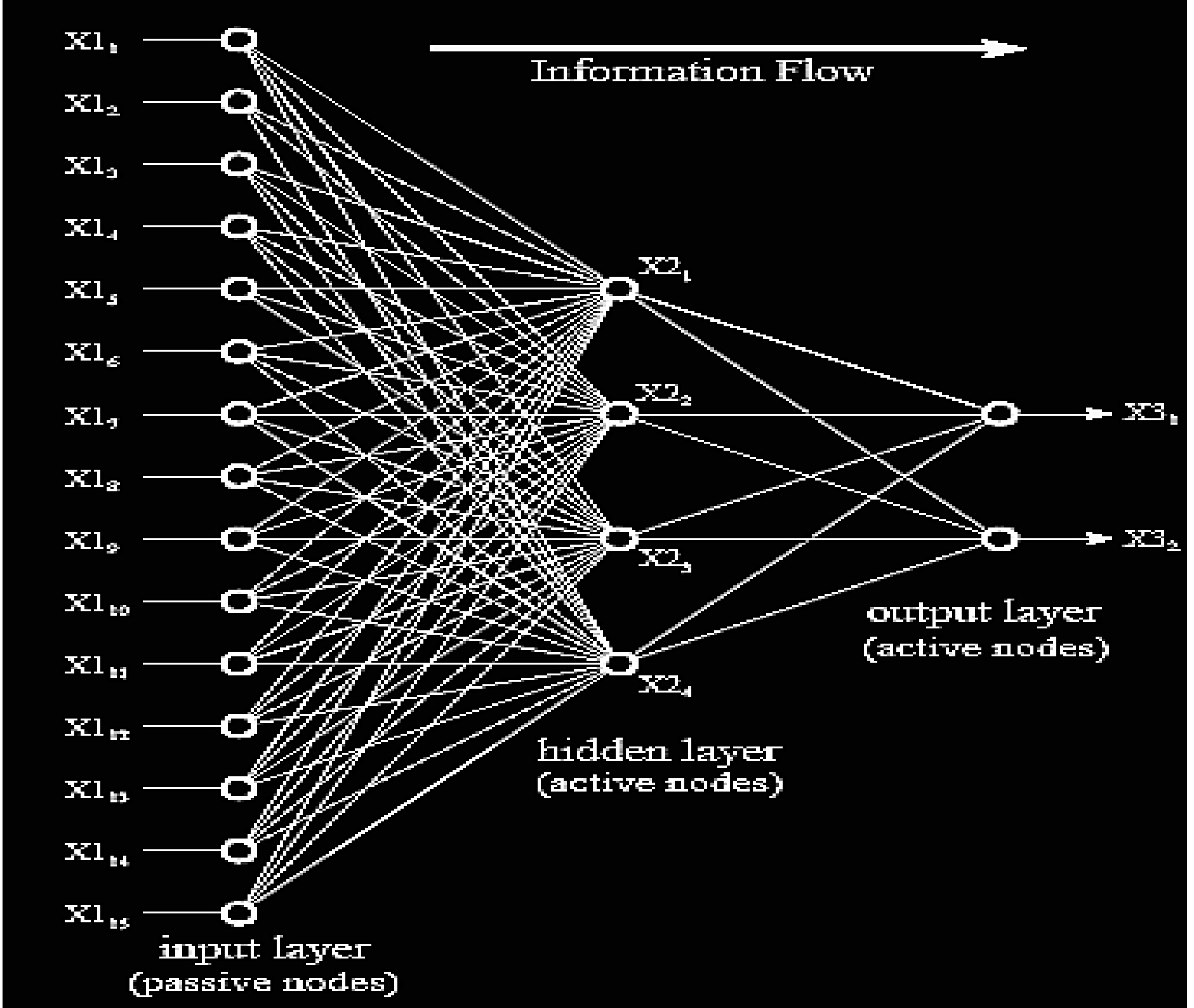

dendrites
nucleus
cell
body
axon
axon
terminals
in1
in2
inn
Σ
f
out
bias

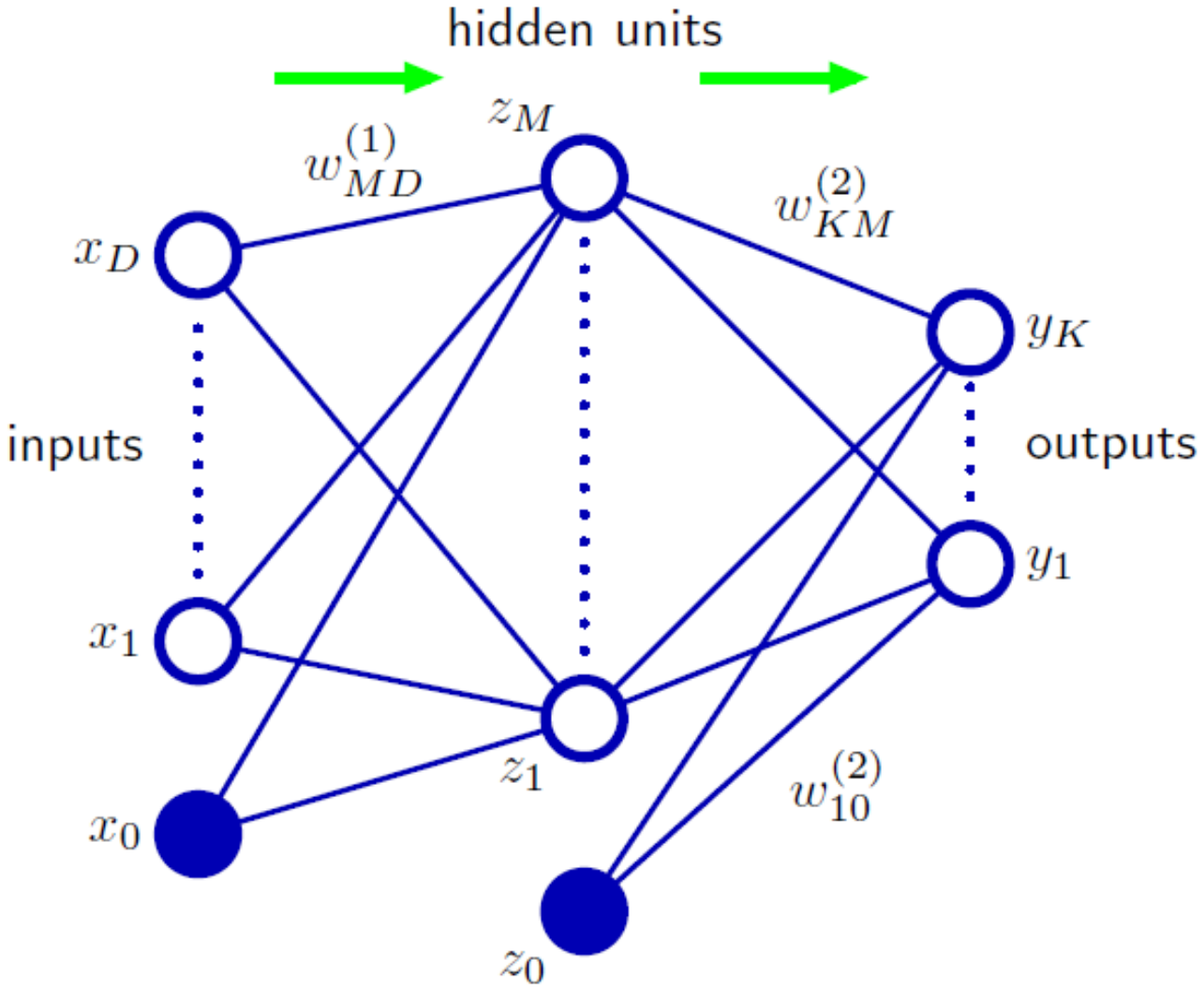

X1
X2
X3
Xn
W1
W2
W3
Wn
1
b
Summer
Threshold unit
Output
Artificial Neuron
W1, W2, W3, Wn - Weights of Connection
X1, X2, X3, Xn - Inputs
b - Bias


Hidden
Input
Output


weights
inputs
x_1
x_2
x_3
x_n
w_{1j}
w_{2j}
w_{3j}
w_{nj}
Σ
transfer
function
net input
net_j
activation
functon
φ
o_j
activation
θ_j
threshold


Input value 1
Input value 2
Input value ···
Input value m
X_1
X_2
X_n
H_1
H_2
H_m
Y_1
Y_u
Estimated value 1
Estimated value ···
Estimated value k
Input layer
Hidden layer
Output layer


Information Flow
input layer
(passive nodes)
hidden layer
(active nodes)
output layer
(active nodes)



Network diagram for the two layer neural network. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0.

Arrows denote the direction of information flow through the network during forward propagation.

Feed-forward Network Functions

- The basic neural network model can be described a series of functional transformations. First we construct *M* linear combinations of the input variables $x_1, \ldots, x_D$ in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

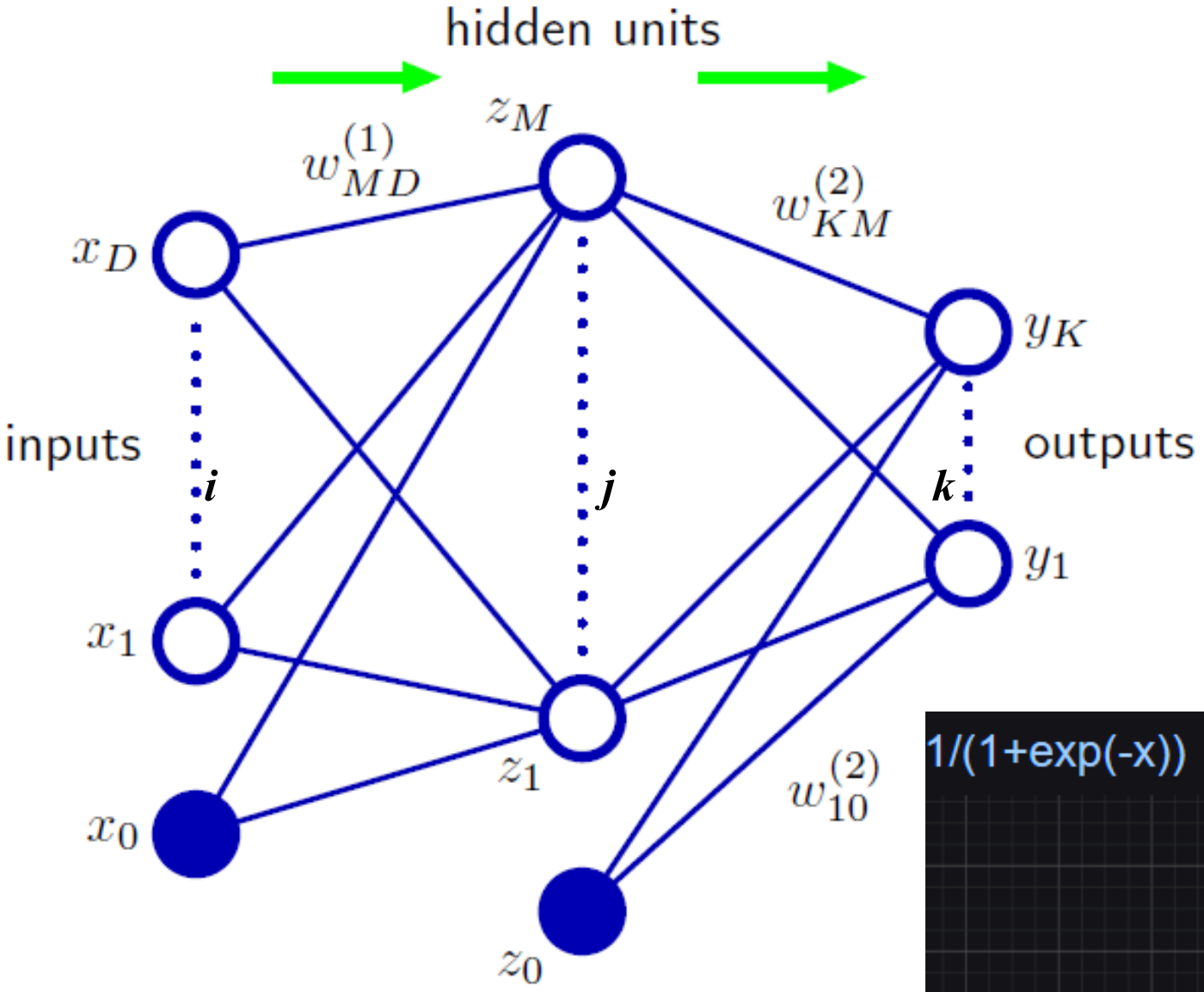
- where $j = 1, \ldots, M$, and the superscript (1) indicates that the corresponding parameters are in the first 'layer' of the network.
- We shall refer to the parameters $w_{ji}^{(1)}$ as *weights* and the parameters $w_{j0}^{(1)}$ as *biases.*
- The quantities a_j are known as *activations*.

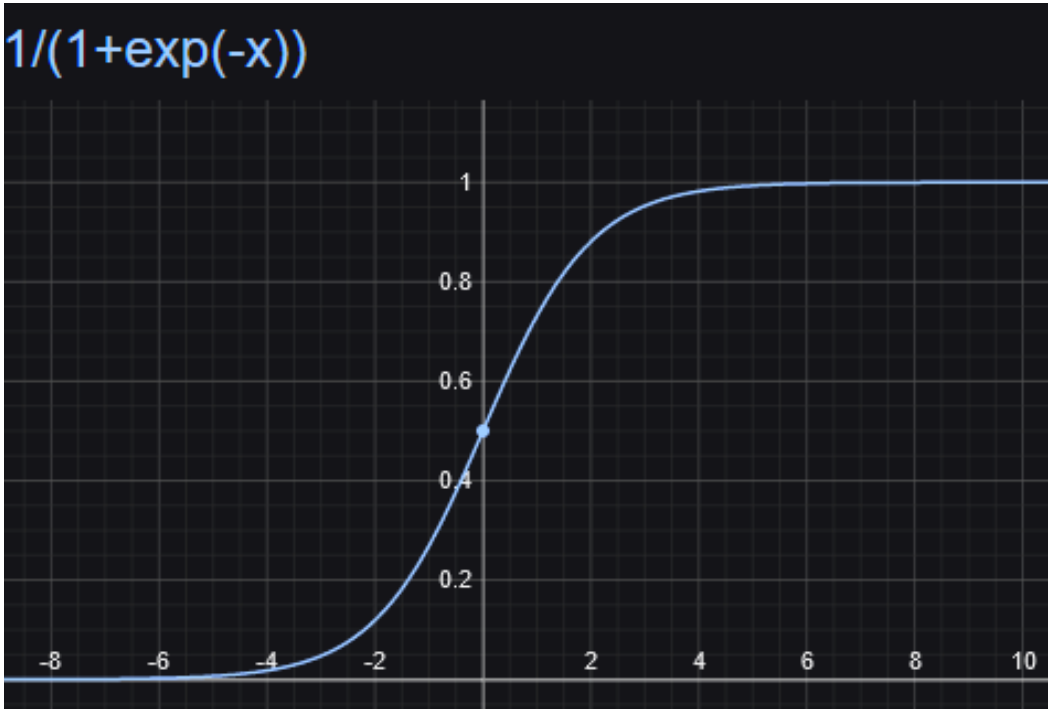
Let's go thru *only the sequence of equations*
(just the maths)

For the derivation of weight update rule, without much of text-based explanations.

Trailing slides will have the explanations – "*Directly from the Book*" ☺

Pl. help yourself, if needed, as per input from class earlier.


hidden units
z_M
$w^{(1)}_{MD}$
$w^{(2)}_{KM}$
x_D
y_K
inputs
outputs
i
j
k
y_1
x_1
z_1
$w^{(2)}_{10}$
x_0
z_0


1/(1+exp(-x))
1
0.8
0.6
0.4
0.2
-8
-6
-4
-2
2
4
6
8
10

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \qquad z_j = h(a_j). \qquad a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

$$y_k = \sigma(a_k) \qquad \sigma(a) = \frac{1}{1+\exp(-a)}.$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i.$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right)$$

Local Minima by MLE

$$y = \sigma(a) \equiv \frac{1}{1+\exp(-a)} \qquad 0 \le y(\mathbf{x}, \mathbf{w}) \le 1$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

$0 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1$. We can interpret $y(\mathbf{x}, \mathbf{w})$ as the conditional probability $p(\mathcal{C}_1|\mathbf{x})$, with $p(\mathcal{C}_2|\mathbf{x})$ given by $1 - y(\mathbf{x}, \mathbf{w})$. The conditional distribution of targets given inputs is then a Bernoulli distribution of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} [1 - y_k(\mathbf{x}, \mathbf{w})]^{1-t_k}$$

$t_k \in \{0, 1\}$, where $k = 1, \ldots, K$

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\}$$

y_{nk} denotes $y_k(\mathbf{x}_n, \mathbf{w})$.

For K binary Classifiers

$$\boldsymbol{y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})},$$

Using one-hot vector Representation (1-of-K coding) ;

NLL & Cross-entropy function;

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}).$$

General cross-entropy expression

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}).$$ $t_{kn} \in \{0,1\}$, 1-of-K

Softmax: $$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Error – BP: Generic case

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$$y_k = \sum_i w_{ki} x_i$$

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$$

y_{nk} denotes $y_k(\mathbf{x}_n, \mathbf{w})$.

Error BP, Sigmoid:

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j).$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

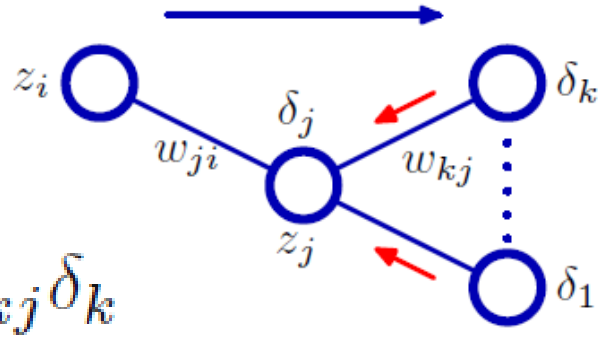
Notations: $$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

$$\frac{\partial a_j}{\partial w_{ji}} = z_i.$$

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

For Output nodes only:

$$\boxed{\delta_k = y_k - t_k}$$



$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \Rightarrow \delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

K reverse connections to j; for Hidden nodes only

The BP-formula
Derivation in next slide

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \qquad (5.55)$$

- From equation 5.51 which is $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ (5.51)

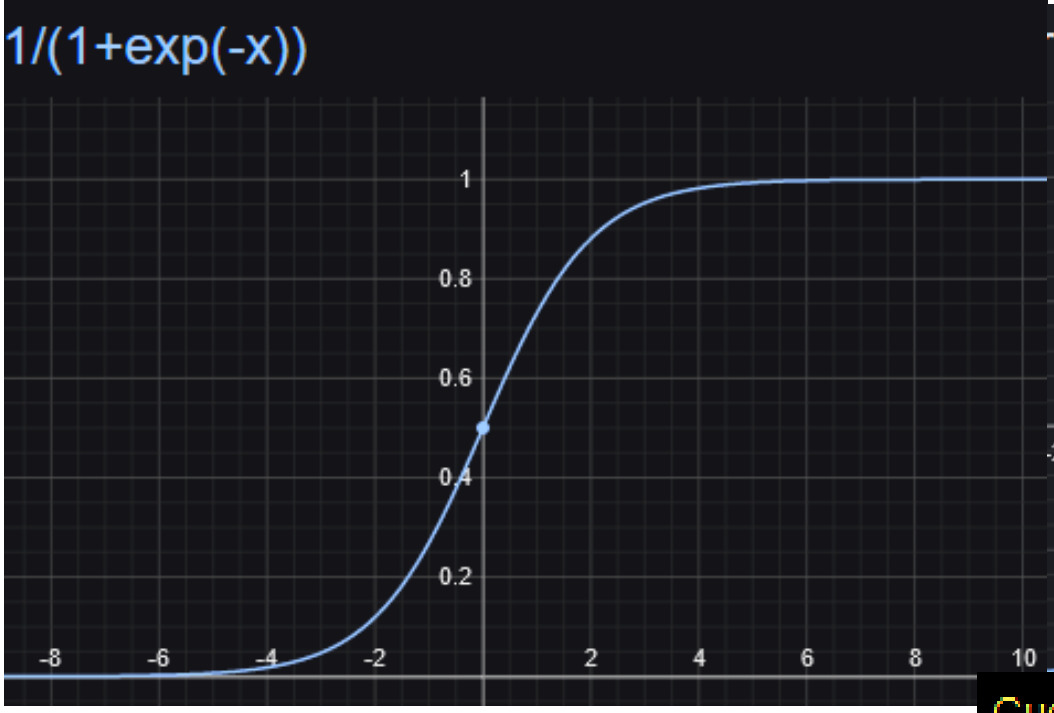
- $\frac{\partial E_n}{\partial a_k}$ can be written as δ_k. i.e., $\frac{\partial E_n}{\partial a_k} = \delta_k$ ----- (1)
  - According to chain rule, $\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j}$
- From equations 5.48 and 5.49, we have:

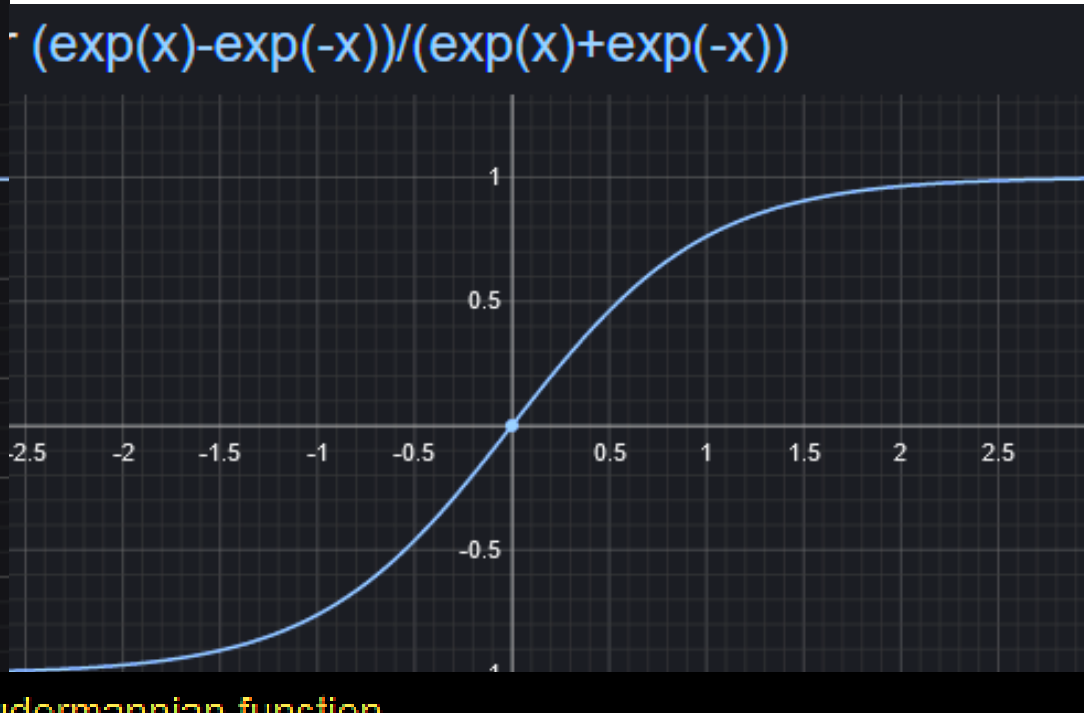
$$z_j = h(a_j). \qquad (5.49)$$

$$a_j = \sum_i w_{ji} z_i \qquad (5.48)$$

- $\frac{\partial a_k}{\partial z_j} = w_{kj}$ and $\frac{\partial z_j}{\partial a_j} = h'(a_j)$ ----- (2)
- Substituting (1) and (2) in eqn 5.55 we get

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \qquad (5.56)$$



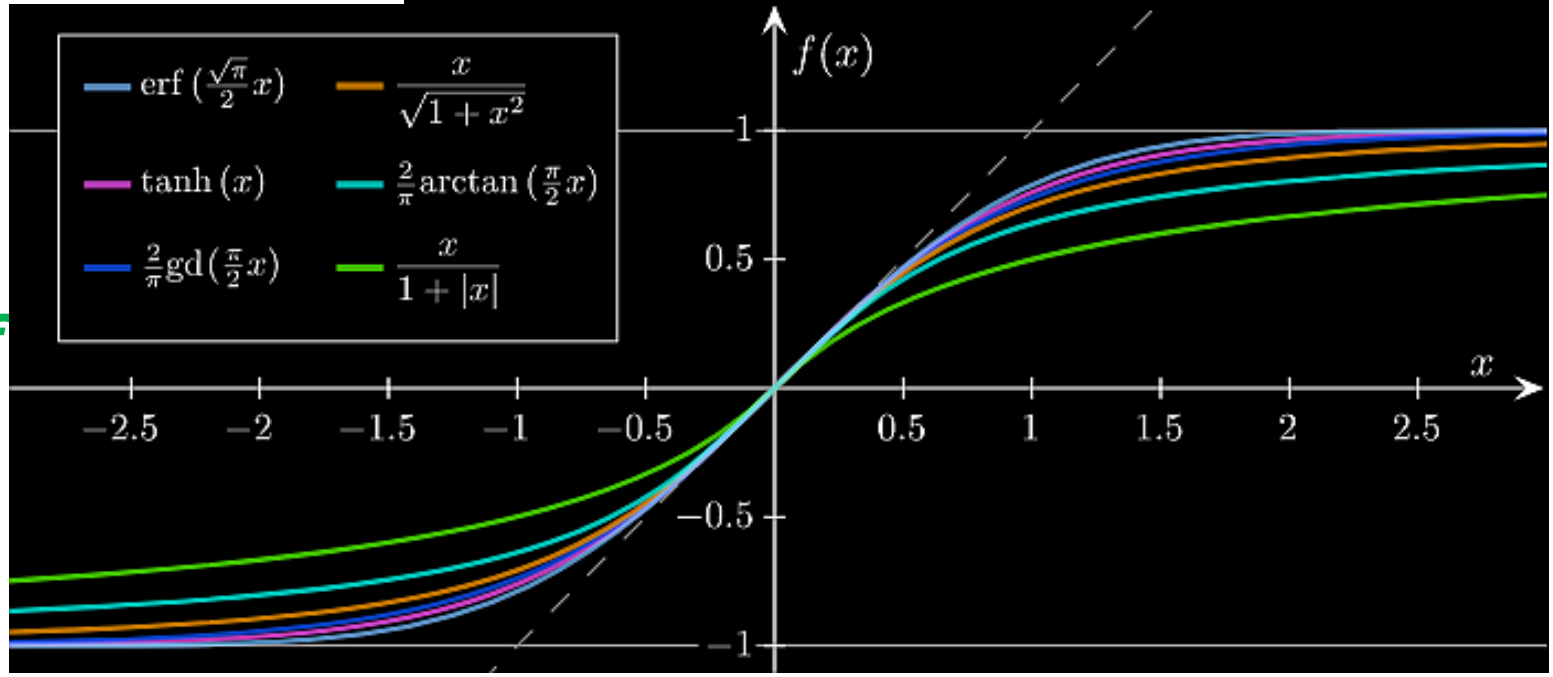


Skip the next few slides:

These are directly from book, piece-wise explanations of the Bank of Eqns. given in last few slides.

Gudermannian function

$$f(x) = \operatorname{gd}(x) = \int_0^x \frac{dt}{\cosh t} = 2\arctan\left(\tanh\left(\frac{x}{2}\right)\right)$$



Feed-forward Network Functions

- Each of them is then transformed using a differentiable, nonlinear *activation function* $h(\cdot)$ to give

$$z_j = h(a_j).$$

- These quantities, in the context of neural networks, are called *hidden units*.
- The nonlinear functions $h(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid or the 'tanh'.

- These values are again linearly combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

where $k = 1, \ldots, K,$ and K is the total number of outputs.

- This transformation corresponds to the second layer of the network, and again the $w_{k0}^{(2)}$ are bias parameters.

Feed-forward Network Functions

- Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k.
- The choice of activation function is determined by the nature of the data and the assumed distribution of target variables and follows the same considerations as for linear models.
- Thus for standard regression problems, the activation function is the identity so that $y_k = a_k$.
- Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$y_k = \sigma(a_k) \text{ where } \sigma(a) = \frac{1}{1 + \exp(-a)}.$$

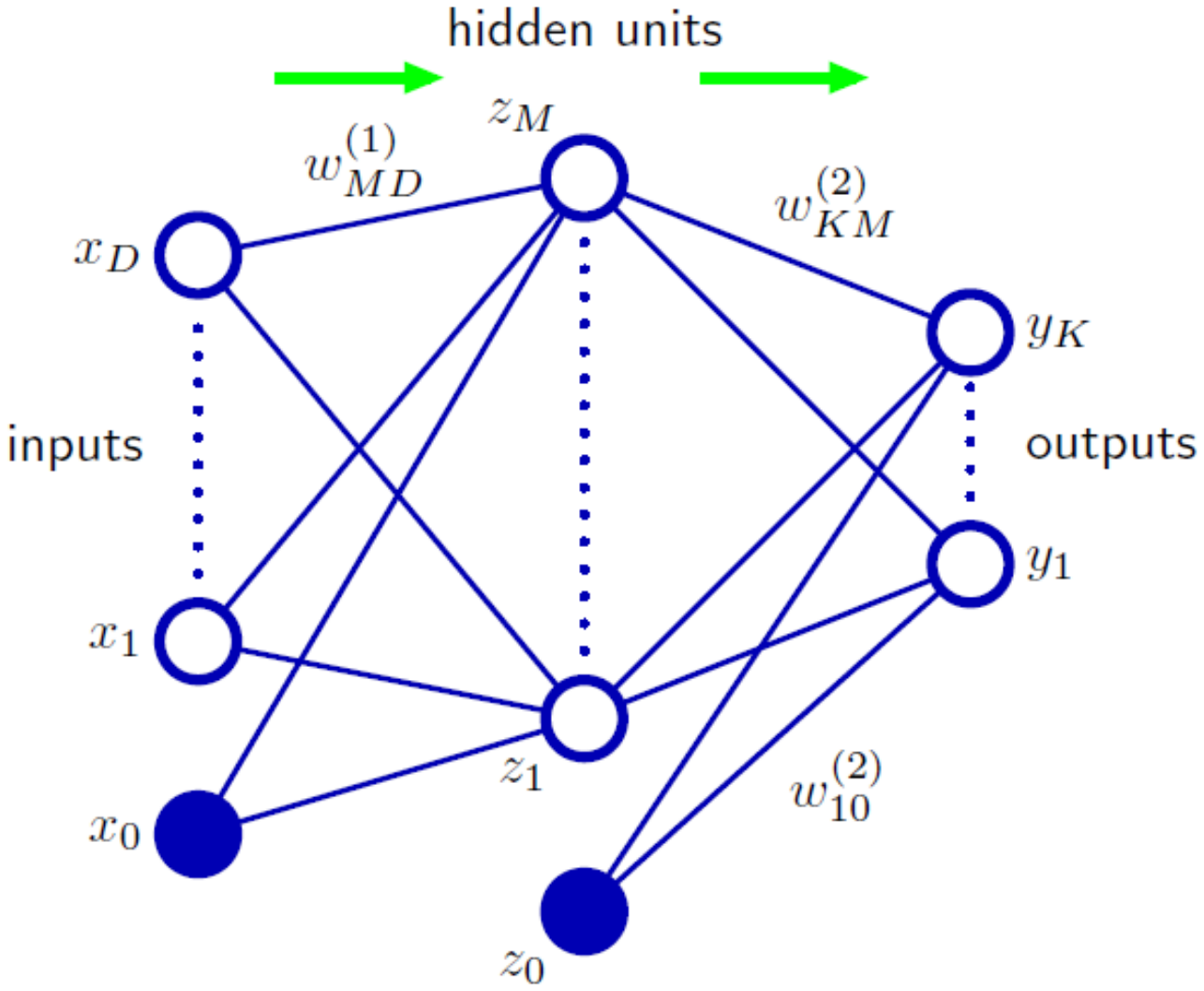
Feed-forward Network Functions

- Finally, for multiclass problems, a softmax activation function is used.
- We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

where the set of all weight and bias parameters have been grouped together into a vector **w**.

- Thus the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector **w** of adjustable parameters.



Network diagram for the two layer neural network. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0.
Arrows denote the direction of information flow through the network during forward propagation.

Feed-forward Network Functions

- The process of evaluating

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

can be interpreted as a *forward propagation* of information through the network.

- The bias parameters can be absorbed into the set of weight parameters by defining an additional input variable x_0 whose value is clamped at $x_0 = 1$, so that :

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i.$$

Feed-forward Network Functions

- We can similarly absorb the second-layer biases into the second-layer weights, so that the overall network function becomes

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right).$$

- If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units.
- Neural networks are said to be *universal approximators*. For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units.

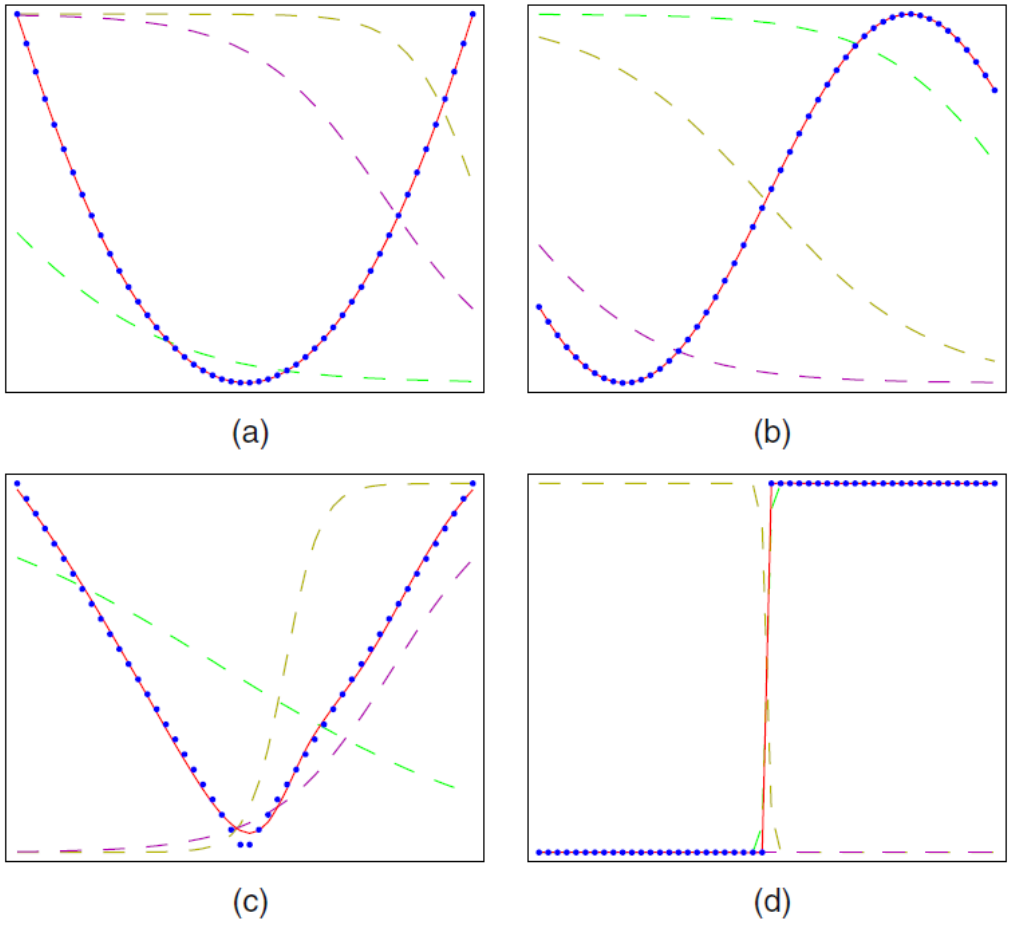


Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

Network training

- Given a training set comprising a set of input vectors $\{\mathbf{x}_n\}$, where $n = 1, \ldots, N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$ *for regression*, we minimize the error function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

- Now consider the case of binary classification in which we have a single target variable t such that $t = 1$ denotes class C_1 and $t = 0$ denotes class C_2.
- Consider a network having a single output whose activation function is a logistic sigmoid

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

so that $0 \leq y(\mathbf{x}, \mathbf{w}) \leq 1$.

Network training

- We can interpret $y(\mathbf{x}, \mathbf{w})$ as the conditional probability $p(C_1|\mathbf{x})$, with $p(C_2|\mathbf{x})$ given by $1 - y(\mathbf{x}, \mathbf{w})$.
- The conditional distribution of targets given inputs is then a Bernoulli distribution of the form

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{1-t}.$$

- If we consider a training set of independent observations, then the error function, which is given by the negative log likelihood, is then a *cross-entropy* error function of the form

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

where y_n denotes $y(\mathbf{x}_n, \mathbf{w})$.

Network training

- Using the cross-entropy error function instead of the sum-of-squares for a classification problem leads to faster training as well as improved generalization.
- If we have K separate binary classifications to perform, then we can use a network having K outputs each of which has a logistic sigmoid activation function.
- Associated with each output is a binary class label $t_k \in \{0, 1\}$, where $k = 1, \ldots, K$.
- If we assume that the class labels are independent, given the input vector, then the conditional distribution of the targets is

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k} \left[1 - y_k(\mathbf{x}, \mathbf{w})\right]^{1-t_k}.$$

Network training

- Taking the negative logarithm of the corresponding likelihood function then gives the following error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_{nk} + (1-t_{nk})\ln(1-y_{nk})\}$$

  where y_{nk} denotes $y_k(\mathbf{x}_n, \mathbf{w})$.

- Finally, we consider the standard multiclass classification problem in which each input is assigned to one of K mutually exclusive classes.
- The binary target variables $t_k \in \{0, 1\}$ have a 1-of-K coding scheme indicating the class, and the network outputs are interpreted as $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$, leading to the following error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn}\ln y_k(\mathbf{x}_n, \mathbf{w}).$$

$$H(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{C} y_i \log p_i$$

The **one-hot vector** $\mathbf{y}$ is:

$$\mathbf{y} = [0, 0, \ldots, 1, \ldots, 0]^\top \quad \text{with } y_c = 1 \text{ and } y_i = 0 \text{ for all } i \neq c$$

Because $\mathbf{y}$ is one-hot ($y_c = 1$ and all other $y_i = 0$), every term except the correct class vanishes:

$$H(\mathbf{y}, \mathbf{p}) = -y_c \log p_c = -\log p_c$$

Under the assumption that the true label follows a **multinomial distribution** (one sample, one correct class), the likelihood of observing the true class c is simply the predicted probability of that class:

$$\mathcal{L}(\mathbf{p}; c) = p_c$$

$$\log \mathcal{L} = \log p_c$$

log-likelihood (the loss we minimise) is:

$$-\log p_c$$

When k is 2 and n is 1, the **multinomial distribution** is the **Bernoulli distribution**. When k is 2 and n is bigger than 1, it is the **binomial distribution**. When k is bigger than 2 and n is 1, it is the **categorical distribution**.

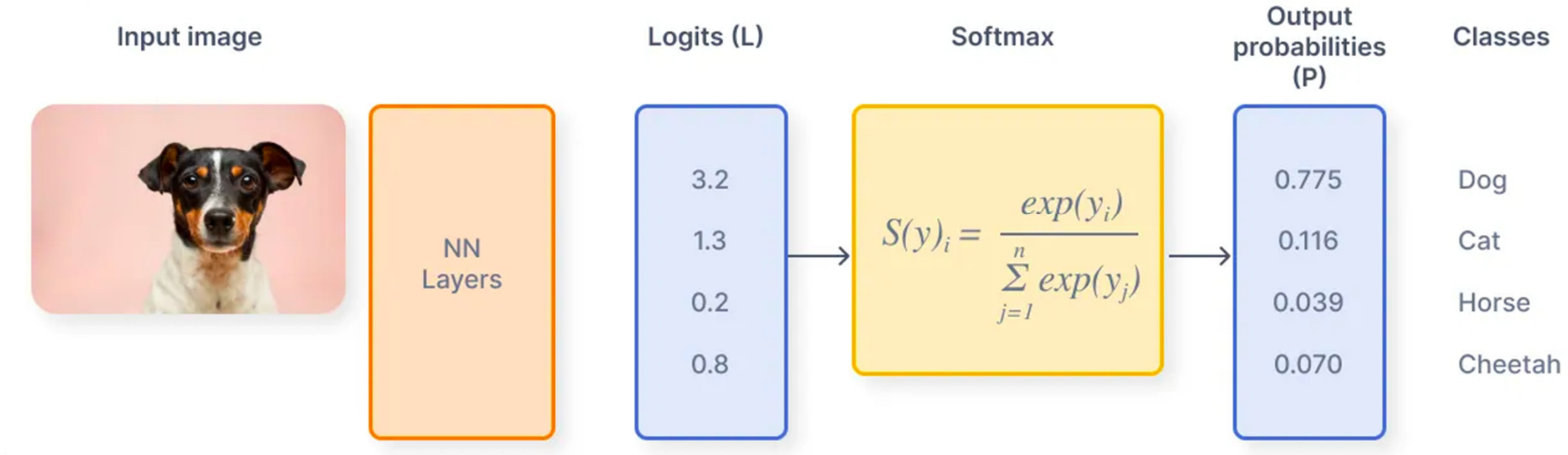
For predicted probability and the actual label:

$$\text{Binary Cross-Entropy}(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y}))$$

For true one-hot encoded class labels, vs the predicted probability for the corresponding class.

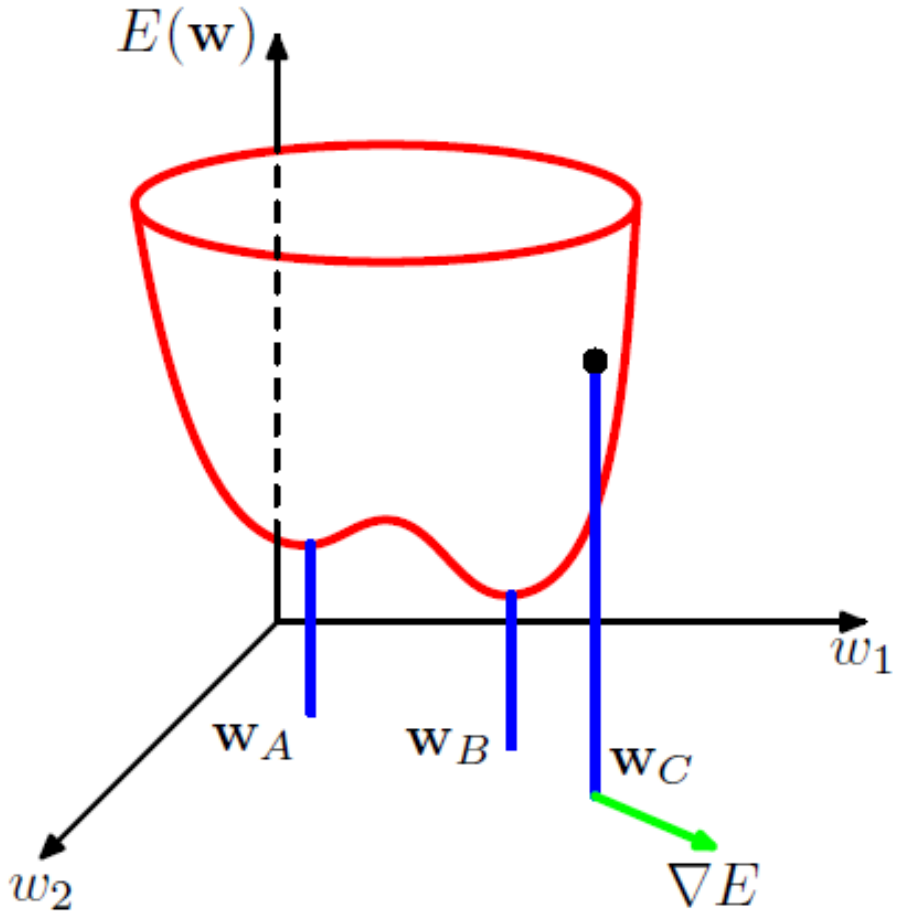
$$\text{Categorical Cross-Entropy}(y, \hat{y}) = -\sum_{i} y_i \cdot \log(\hat{y}_i)$$

Example	Ground Truth (One-Hot Encoded)	Predicted Probabilities (Model Output)
1	[1, 0, 0] (Car)	[0.8, 0.1, 0.1]
2	[0, 1, 0] (Bike)	[0.2, 0.7, 0.1]
3	[0, 0, 1] (Person)	[0.1, 0.2, 0.7]
4	[1, 0, 0] (Car)	[0.6, 0.2, 0.2]



Network training

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}).$$



Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.

Network training

- The output unit activation function is given by the softmax function

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

- which satisfies $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.

	Outputs	
	Real Values	Probabilities
Output Activation	Linear	Softmax
Loss Function	Squared Error	Cross Entropy

Gradient descent optimization

- The simplest approach to using gradient information is to choose the weight update to comprise a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

where the parameter $\eta > 0$ is known as the *learning rate*.

- After each such update, the gradient is re-evaluated for the new weight vector and the process repeated.
- Note that the error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate ∇E.
- At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as *gradient descent* or *steepest descent*.

Gradient descent optimization

- On-line gradient descent, also known as *sequential gradient descent* or *stochastic gradient descent*, makes an update to the weight vector based on **one data point at a time**, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}).$$

Error Backpropagation

- Our goal in this section is to find an efficient technique for evaluating the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network.
- We shall see that this can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as ***error backpropagation***, or sometimes simply as *backprop*.
- We now derive the backpropagation algorithm for a general network having arbitrary feed-forward topology, arbitrary differentiable nonlinear activation functions, and a broad class of error function.
- The resulting formulae will then be illustrated using a simple layered network structure having a single layer of sigmoidal hidden units together with a sum-of-squares error.

Error Backpropagation

- Many error functions of practical interest, for instance those defined by maximum likelihood for a set of i.i.d. data, comprise a sum of terms, one for each data point in the training set, so that

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

- Here we shall consider the problem of evaluating $\nabla E_n(\mathbf{w})$ for one such term in the error function.
- This may be used directly for sequential optimization, or the results can be accumulated over the training set in the case of batch methods.

Error Backpropagation

- Consider first *a simple linear model* in which the outputs y_k are linear combinations of the input variables x_i so that

$$y_k = \sum_i w_{ki} x_i$$

together with an error function that, for a particular input pattern n, takes the form

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

where, $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$.

The gradient of this error function with respect to a weight w_{ji} is given by

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

Error Backpropagation

- In a general feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ji} z_i$$

- where z_i is the activation of a unit, or input, that sends a connection to unit j, and w_{ji} is the weight associated with that connection.
- This sum is transformed by a nonlinear activation function $h(\cdot)$ to give the activation z_j of unit j in the form

$$z_j = h(a_j).$$

- Now consider the evaluation of the derivative of E_n with respect to a weight w_{ji}.

Error Backpropagation

- First we note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j. We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}.$$

- We now introduce a useful notation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

where the δ's are often referred to as *errors.*

- Using $a_j = \sum_i w_{ji} z_i$ we can write $\frac{\partial a_j}{\partial w_{ji}} = z_i.$

Error Backpropagation

- We thus obtain

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

- For the output units, we have

$$\delta_k = y_k - t_k$$

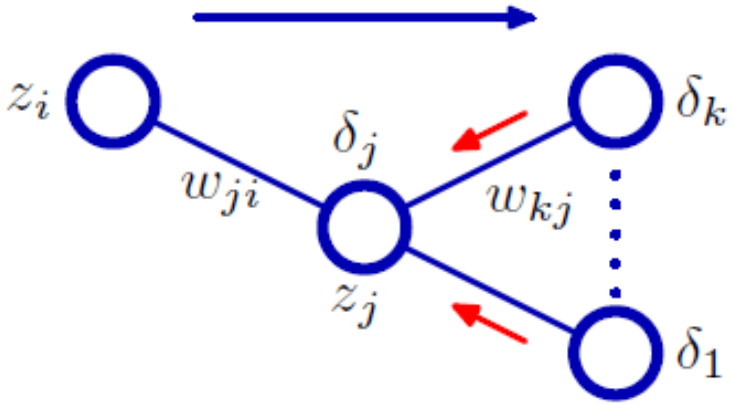


Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

Error Backpropagation

- To evaluate the δ's for hidden units, we again make use of the chain rule for partial derivatives,

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

where the sum runs over all units k to which unit j sends connections.

- If we now substitute the definition of δ we obtain the following *backpropagation* formula

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

To revise the last part of BP-derivation

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{5.55}$$

- From equation 5.51 which is

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \tag{5.51}$$

- $\frac{\partial E_n}{\partial a_k}$ can be written as δ_k. i.e., $\frac{\partial E_n}{\partial a_k} = \delta_k$ ----- (1)
  - According to chain rule, $\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j}$
- From equations 5.48 and 5.49 we have:

$$a_j = \sum_i w_{ji} z_i \tag{5.48}$$

$$z_j = h(a_j). \tag{5.49}$$

- $\frac{\partial a_k}{\partial z_j} = w_{kj}$ and $\frac{\partial z_j}{\partial a_j} = h'(a_j)$ ----- (2)
- Substituting (1) and (2) in eqn 5.55 we get

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{5.56}$$

Marker slide –

for continuity only,

after skipping
Descriptions of the analytics

Error Backpropagation: Summary

The backpropagation procedure can therefore be summarized as follows:

- Apply an input vector $\mathbf{x}_n$ to the network and forward propagate through the network to find the activations of all the hidden and output units.
- Evaluate the δ_k for all the output units.
- Backpropagate the δ's to obtain δ_j for each hidden unit in the network.
- Evaluate the required derivatives.

For batch methods, the derivative of the total error E can then be obtained by repeating the above steps for each pattern in the training set and then summing over all patterns:

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}.$$

Backpropagation Algorithm: Definitions

- Each training example is a pair of the form $(\vec{x}, \vec{t})$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.
- η is the learning rate (e.g., 0.05). , D is the number of network inputs, M the number of units in the hidden layer, and K the number of output units. The input from unit p into unit q is denoted x_{qp}, and the weight from unit p to unit q is denoted w_{qp}.

Backpropagation Algorithm

- Create a feed-forward network with D inputs, M hidden units, and K output units.
- Initialize all network weights to small random numbers.
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in training examples, Do
    - Propagate the input forward through the network:
      1. Input the instance $\vec{x}$ to the network and compute the output y_k, of every unit k in the network.
    - Propagate the errors backward through the network:

Backpropagation Algorithm

1. Propagate the errors backward through the network:
2. For each network output unit k, calculate its error term δ_k

$$\boldsymbol{\delta_k \leftarrow y_k(1-y_k)(t_k-y_k)}$$

3. For each hidden unit z_j, calculate its error term δ_z

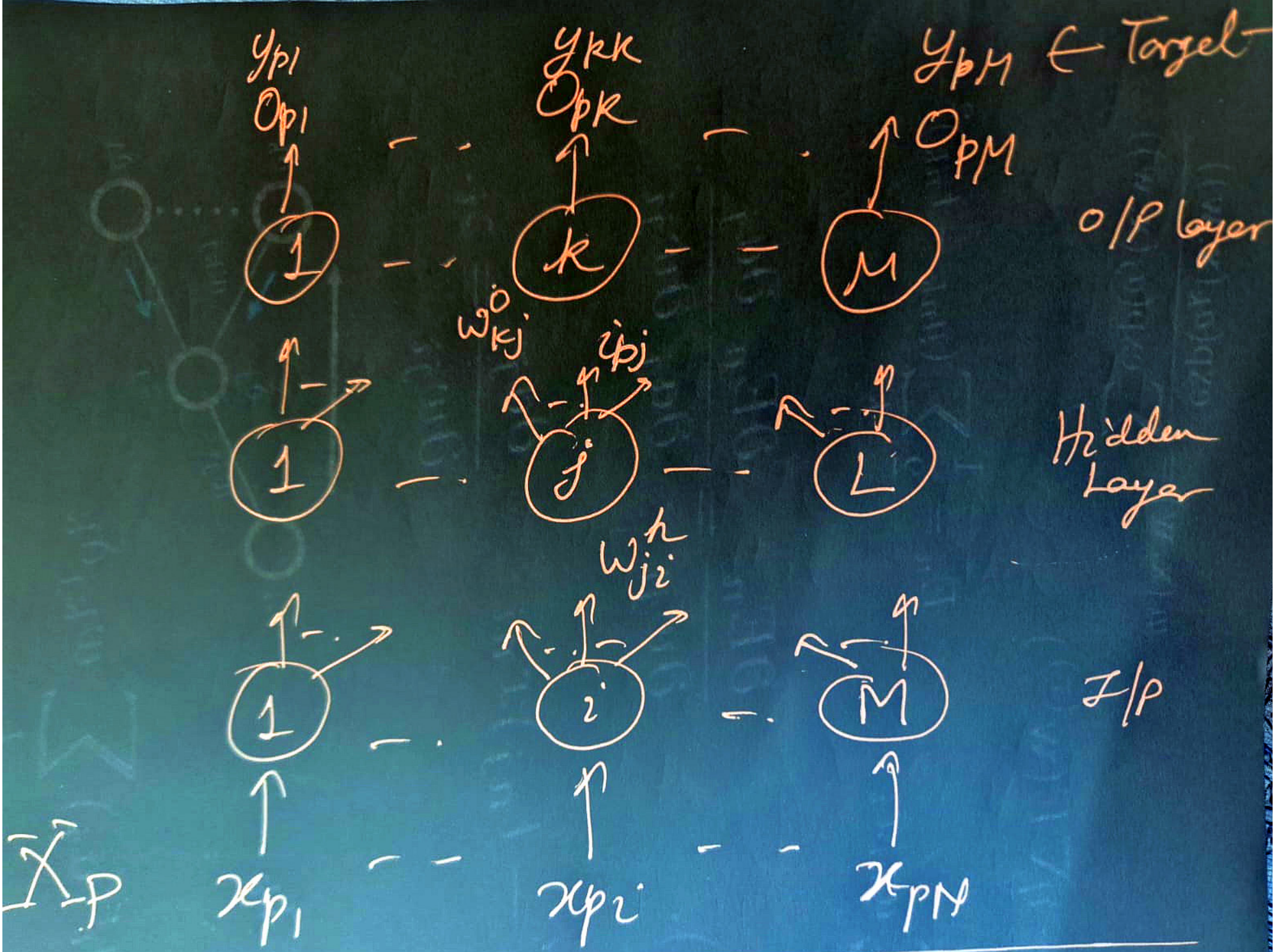
$$\boldsymbol{\delta_z \leftarrow z_j(1-z_j)\sum_{k\in outputs} w_{kj}\delta_k}$$

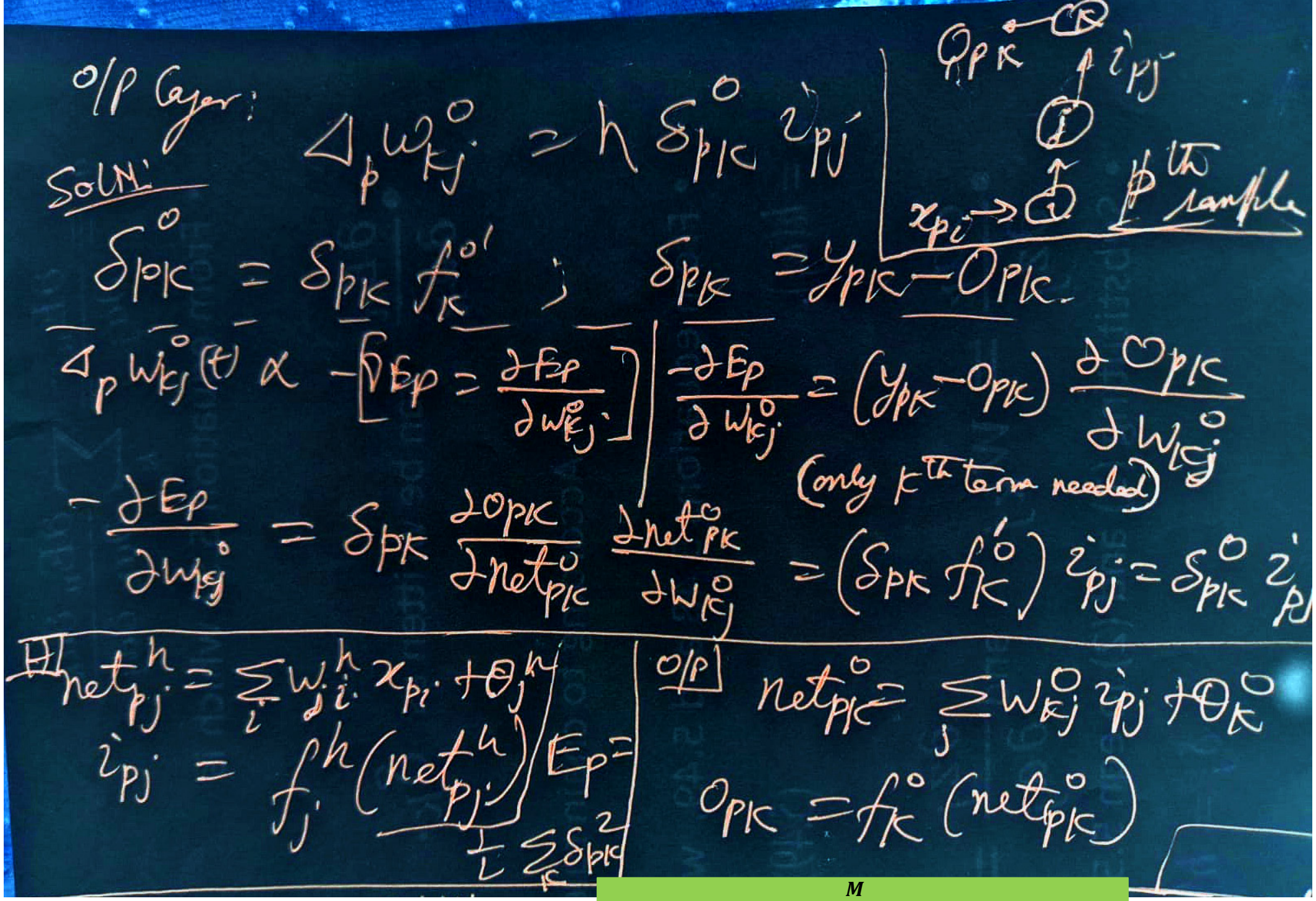
4. Update each network weight w_{qp}

$$\boldsymbol{w_{qp} \leftarrow w_{qp} + \nabla w_{qp}}$$

where, $\boldsymbol{\nabla w_{qp} = \eta \delta_q x_{qp}}$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \Rightarrow \delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$


y_{p1}
O_{p1}
y_{pk}
O_{pk}
y_{pM} ∈ Target
O_{pM}
1
k
M
O/P layer
w^o_{kj}
i_{pj}
1
j
L
Hidden Layer
w^h_{ji}
1
i
M
I/P
$\bar{X}_p$
x_{p1}
x_{pi}
x_{pN}



$$E_p = (1/2) \sum_{k=1}^{M} (y_{pk} - o_{pk})^2$$

Inter. layer: Soln:
Hidden

$$\Delta_p w_{ji}^h = \eta\, \delta_{pj}^h\, x_{pi}$$

$$\delta_{pj}^h = \left(f_j^{h\prime}\right) \sum_{k=1}^{M} w_{kj}\, \delta_{pk}^o$$

$\delta_{pk}^o = \delta_{pk} f_k^{o\prime}$

$$\Delta w_{ji}^h(t) \propto \left[-\frac{\partial E_p}{\partial w_{ji}^h} = \sum_{k=1}^{M} (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial w_{ji}^h} \right.$$

$$\frac{\partial o_{pk}}{\partial w_{ji}^h} = \frac{\partial o_{pk}}{\partial net_{pk}^o} \frac{\partial net_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial net_{pj}^h} \frac{\partial net_{pj}^h}{\partial w_{ji}^h}$$

$$\left. -\frac{\partial E_p}{\partial w_{ji}^h} = f_k^{o\prime} w_{kj}^o f_j^{h\prime} x_{pi} \right]$$

$$= f_j^{h\prime} x_{pi} \sum_k \left[\delta_{pk} f_k^{o\prime} w_{kj}^o \right]$$

$$= f_j^{h\prime} x_{pi} \sum \left[\underbrace{\delta_{pk}^o}_{\delta_{pk} f_k^{o\prime}} w_{kj}^o \right]$$

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta\, \delta_{pj}^h\, x_{pi}$$

where,

$$\delta_{pj}^h = f_j^{h\prime} \sum \delta_{pk}^o\, w_{kj}^o$$

o/p: $\Delta w_{kj}^{o} = \eta \delta_{pk}^{o} i_{pj}$

$= \eta \delta_{pk} f_k^{o\prime} i_{pj}$

$= \eta (y_{pk} - o_{pk}) f(f-1) i_{pj}$

H/L: $\Delta w_{ji}^{h} = \eta \delta_j^{h} x_{pi} = \eta (f_j^{h\prime}) \sum_k [w_{kj}^{o} \cdot \delta_{pk}^{o}] x_{pi}$

$= \eta f_j^{h\prime} \sum_k [\delta_{pk} f_k^{o\prime} w_{kj}^{o}] x_{pi}$

$= \eta f_j^{h}(f_j^{h}-1) \sum_k [\delta_{pk} f_k^{o} (f_k^{o}-1) w_{kj}^{o}] x_{pi}$

Src: WIKI

- x: input (vector of features)
- y: target output

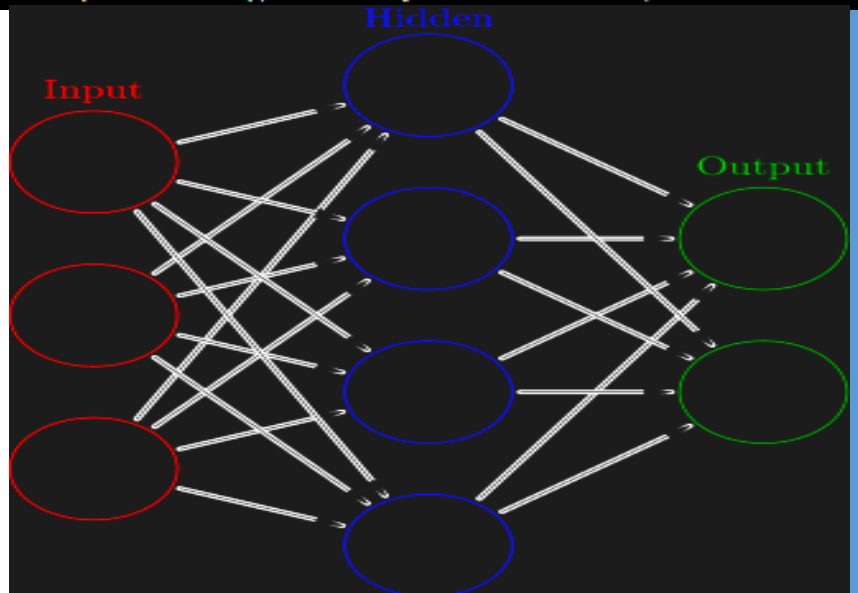
  For classification, output will be a vector of class probabilities (e.g., $(0.1, 0.7, 0.2)$, and target output is a specific class, encoded by the one-hot/dummy variable (e.g., $(0, 1, 0)$).

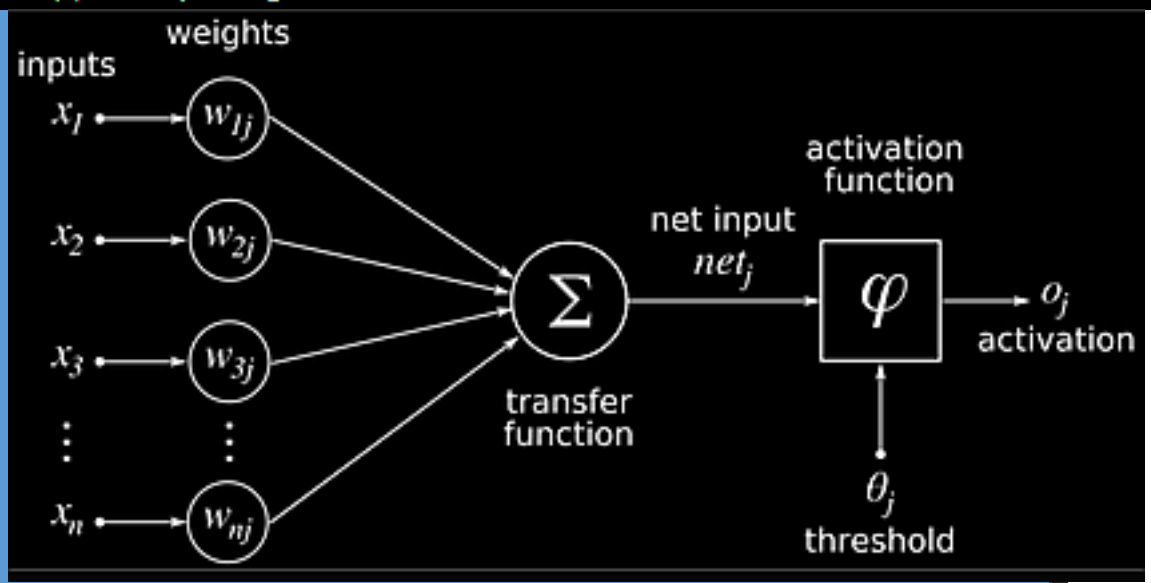
- C: loss function or "cost function"[a]

  For classification, this is usually cross entropy (XC, log loss), while for regression it is usually squared error loss (SEL).

- L: the number of layers
- $W^l = (w^l_{jk})$: the weights between layer $l-1$ and l, where w^l_{jk} is the weight between the k-th node in layer $l-1$ and the j-th node in layer l[b]
- f^l: activation functions at layer l

  For classification the last layer is usually the logistic function for binary classification, and softmax (softargmax) for multi-class classification, while for the hidden layers this was traditionally a sigmoid function (logistic function or others) on each node (coordinate), but today is more varied, with rectifier (ramp, ReLU) being common.





The gradient descent method involves calculating the derivative of the loss function with respect to the weights of the network. This is normally done using backpropagation. Assuming one output neuron,[h] the squared error function is

$$E = L(t, y)$$

where

L is the loss for the output y and target value t,

t is the target output for a training sample, and

y is the actual output of the output neuron.

For each neuron j, its output o_j is defined as

$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} x_k\right),$$

where the activation function φ is non-linear and differentiable over the activation region (the ReLU is not differentiable at one point). A historically used activation function is the logistic function:

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

which has a convenient derivative of:

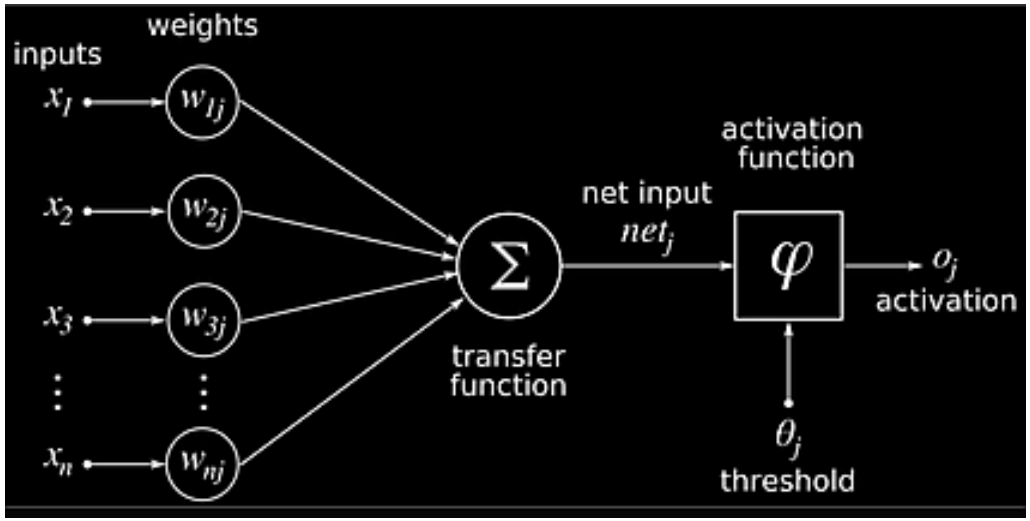
$$\frac{d\varphi(z)}{dz} = \varphi(z)(1 - \varphi(z))$$

The input net_j to a neuron is the weighted sum of outputs o_k of previous neurons. If the neuron is in the first layer after the input layer, the o_k of the input layer are simply the inputs x_k to the network. The number of input units to the neuron is n. The variable w_{kj} denotes the weight between neuron k of the previous layer and neuron j of the current layer.

Src: WIKI

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$

$$\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij} o_i = o_i$$



$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial \varphi(\mathrm{net}_j)}{\partial \mathrm{net}_j}$$

which for the logistic activation function

$$\frac{\partial o_j}{\partial \mathrm{net}_j} = \frac{\partial}{\partial \mathrm{net}_j}\varphi(\mathrm{net}_j) = \varphi(\mathrm{net}_j)(1-\varphi(\mathrm{net}_j)) = o_j(1-o_j)$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y}$$

If half of the square error is used as loss function

$$\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2 = y - t$$

Considering E as a function with the inputs being all neurons $L = \{u, v, \ldots, w\}$ receiving input from neuron j,

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(\mathrm{net}_u, \mathrm{net}_v, \ldots, \mathrm{net}_w)}{\partial o_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} o_i$$

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

$$\frac{\partial E}{\partial o_j} = \sum_{\ell \in L}\left(\frac{\partial E}{\partial \mathrm{net}_\ell}\frac{\partial \mathrm{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \mathrm{net}_\ell}\frac{\partial \mathrm{net}_\ell}{\partial o_j}\right) = \sum_{\ell \in L}\left(\frac{\partial E}{\partial o_\ell}\frac{\partial o_\ell}{\partial \mathrm{net}_\ell} w_{j\ell}\right)$$

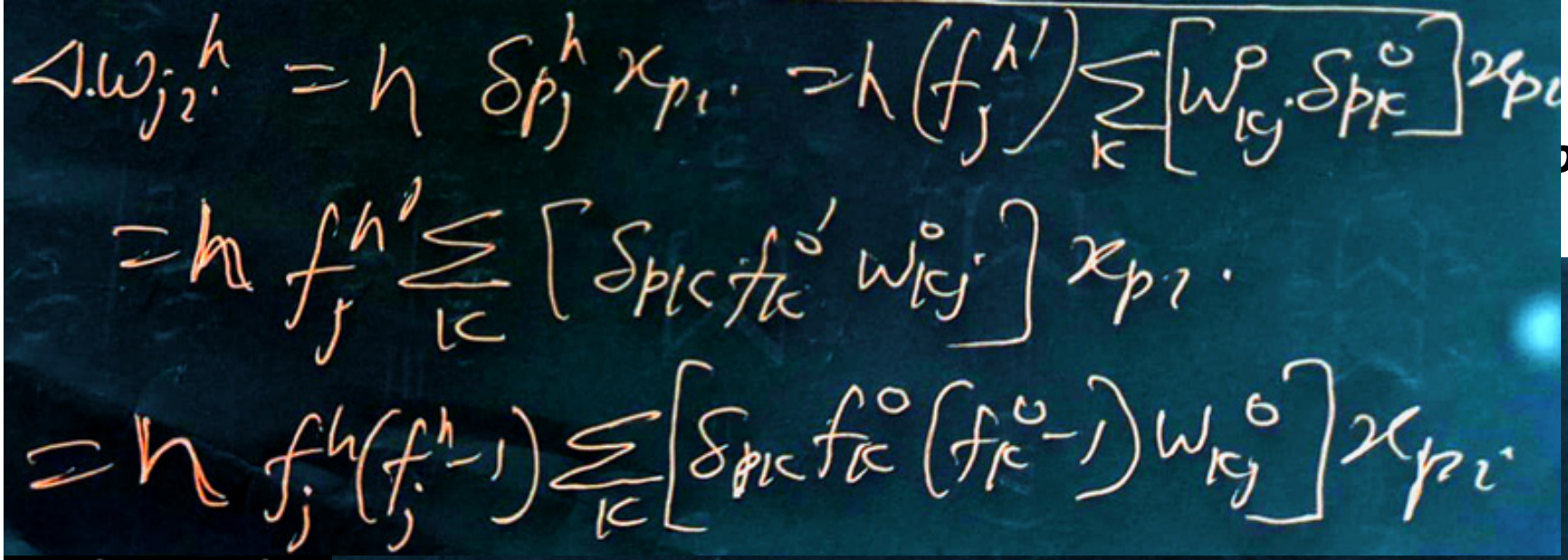
with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} \frac{\partial L(o_j, t)}{\partial o_j}\frac{d\varphi(\mathrm{net}_j)}{d\mathrm{net}_j} & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)\frac{d\varphi(\mathrm{net}_j)}{d\mathrm{net}_j} & \text{if } j \text{ is an inner neuron.} \end{cases}$$

if φ is the logistic function, and the error is the square error:

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j) & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)o_j(1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$



...o compare all three

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j$$

with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} \frac{\partial L(o_j,t)}{\partial o_j}\frac{d\varphi(\text{net}_j)}{d\text{net}_j} & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)\frac{d\varphi(\text{net}_j)}{d\text{net}_j} & \text{if } j \text{ is an inner neuron.} \end{cases}$$

if φ is the logistic function, and the error is the square error:

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j)o_j(1-o_j) & \text{if } j \text{ is an output neuron,} \\ \left(\sum_{\ell \in L} w_{j\ell}\delta_\ell\right)o_j(1-o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

What is fed back to input layers from output layers, through the hidden layers??

- the error signal (also known as the error derivative or gradient)

Gradients of the Loss Function: Using the chain rule of calculus, the algorithm calculates partial derivatives (gradients) of the total error with respect to every weight and bias in the network.

These fed-back gradients indicate the direction and magnitude of the adjustments needed to minimize error in the next training epoch. The weights and biases are then updated, typically using an optimization method like gradient descent.

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

- δ_j is the error term for each unit,
- η is the learning rate.

$$\delta_3 = y_3(1 - y_3)(w_{1,3} \times \delta_5)$$

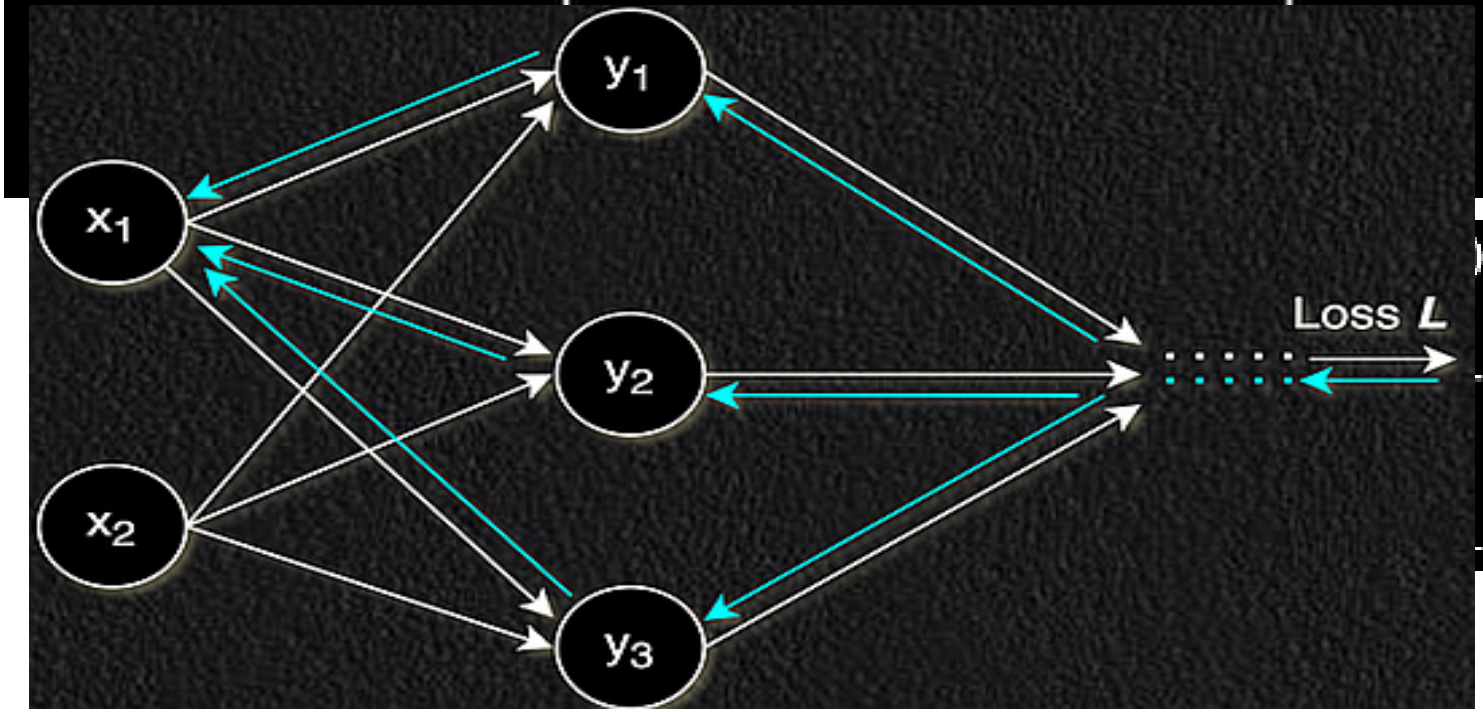
Definition: For a function mapping input vector $\mathbf{x} \in \mathbb{R}^n$ to output $\mathbf{y} \in \mathbb{R}^m$, the Jacobian $\mathbf{J} \in \mathbb{R}^{m \times n}$ is:

$$J_{i,j} = \frac{\partial y_i}{\partial x_j}$$

$$\mathbf{x} = [x_1, x_2, \ldots, x_m]^T,\ \mathbf{y} = [y_1, y_2, \ldots, y_n]^T,\ f: \mathbb{R}^m \to \mathbb{R}^n$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

known as the Jacobian matrix



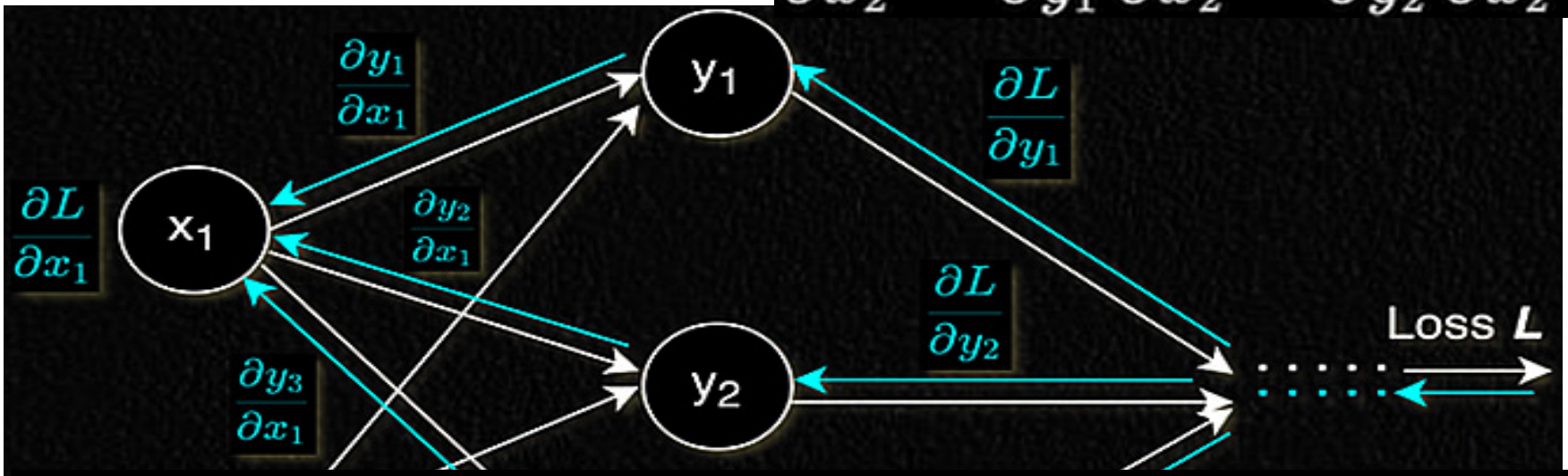
$)]^T$, the 2x2 Jacobian matrix is

$$\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}$$

scalar loss L at the end of the network:

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_1}$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_2} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_2} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_2}$$



$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \frac{\partial y_3}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_3}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \frac{\partial L}{\partial y_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{bmatrix}^T \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \frac{\partial L}{\partial y_3} \end{bmatrix}$$

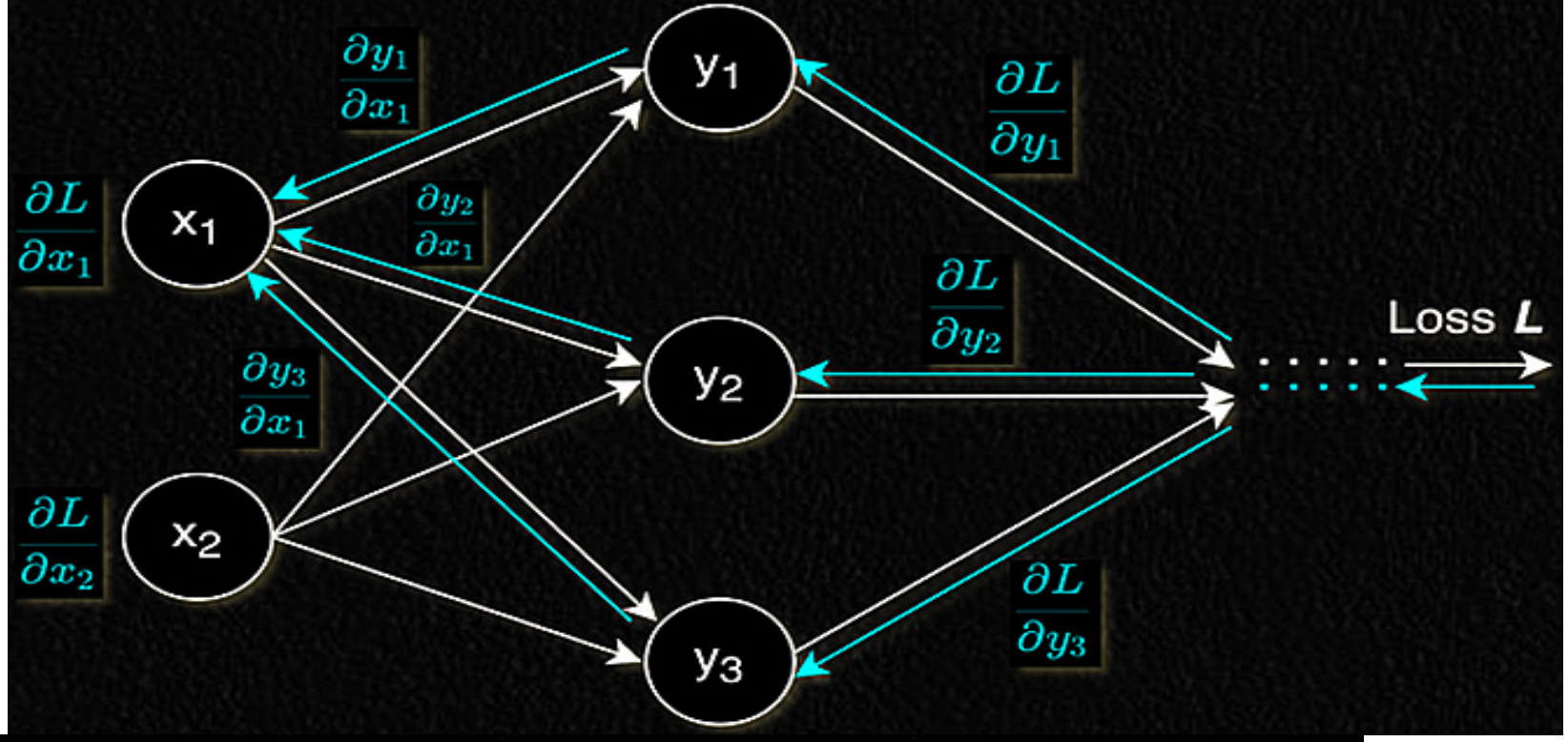
$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \frac{\partial y_3}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_3}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \frac{\partial L}{\partial y_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{bmatrix}^T \begin{bmatrix} \frac{\partial L}{\partial y_1} \\ \frac{\partial L}{\partial y_2} \\ \frac{\partial L}{\partial y_3} \end{bmatrix}$$

$\mathbf{x} = [x_1, x_2, \ldots, x_m],\ \mathbf{y} = [y_1, y_2, \ldots, y_n],\ f : \mathbb{R}^m \to \mathbb{R}^n$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{bmatrix}$$

Jacobian

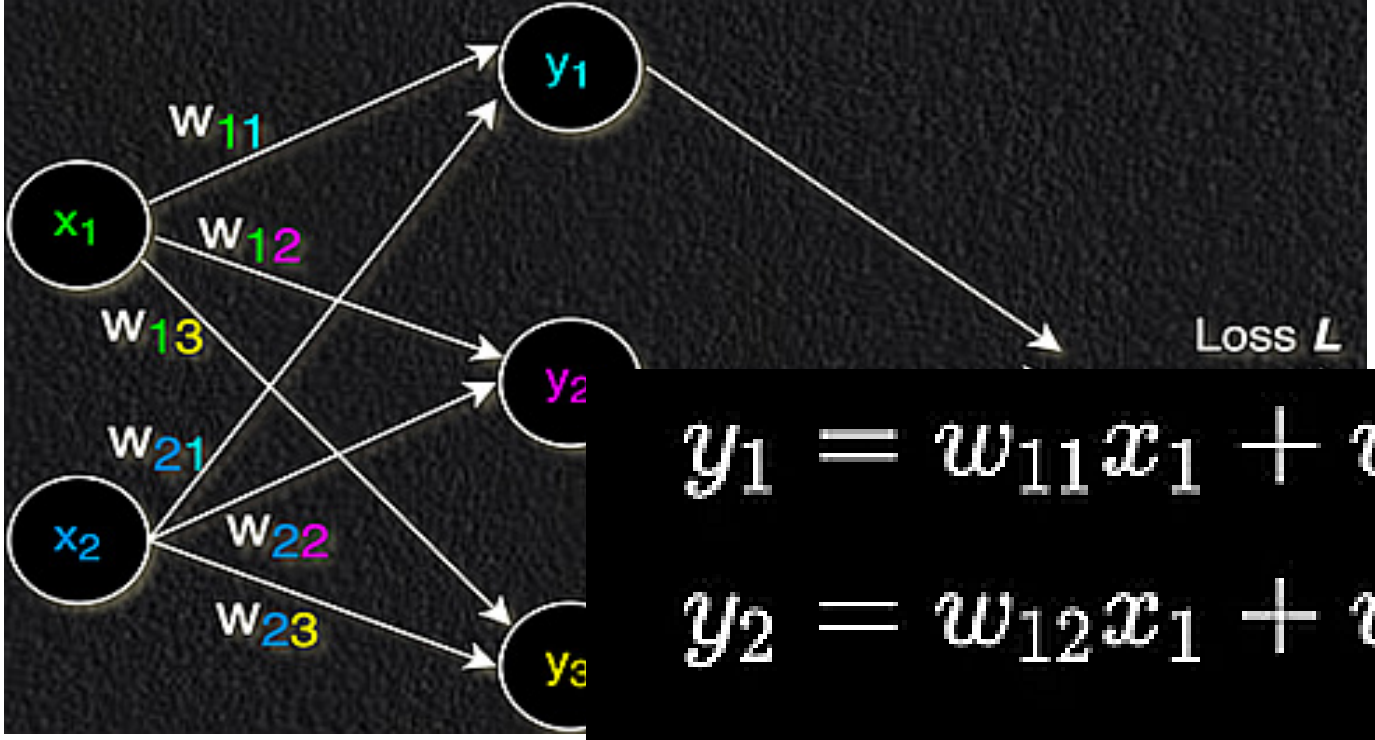


$$\mathbf{x} = [x_1, x_2, \ldots, x_m],\ \mathbf{y} = [y_1, y_2, \ldots, y_n],\ f : \mathbb{R}^m \to \mathbb{R}^n$$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

Use this Jacobian based formula to propagate the gradient backwards

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{bmatrix}$$



Altn Method to avoid Jacobian (in DL):

$$y_1 = w_{11}x_1 + w_{21}x_2$$

$$y_2 = w_{12}x_1 + w_{22}x_2$$

$$y_3 = w_{13}x_1 + w_{23}x_2$$

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$

$$y_j = \sum_i w_{ij} x_i$$

$$\frac{\partial y_j}{\partial x_i} = w_{ij}, \quad i = 1, 2, j = 1, 2, 3$$

J is Memory and compute Intensive

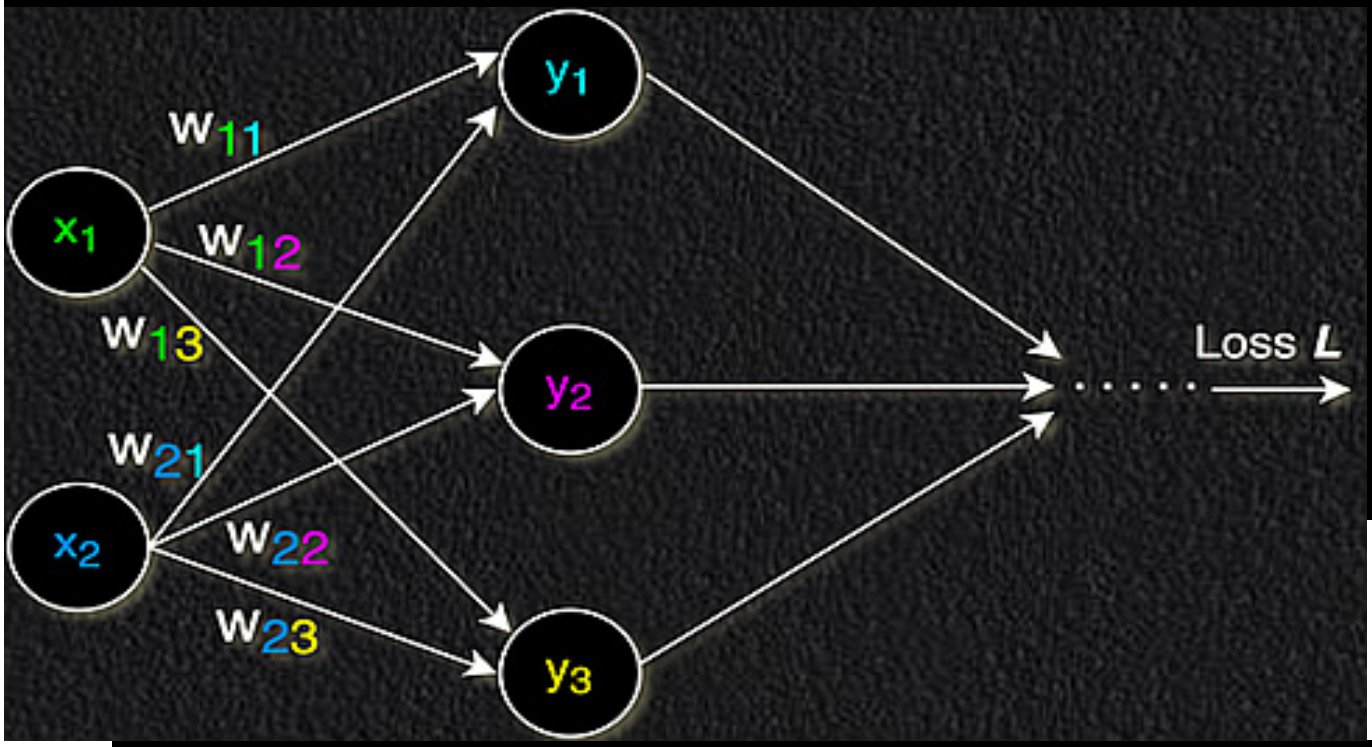
$$\begin{bmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} \end{bmatrix}$$

$$\frac{\partial y_j}{\partial x_i} = w_{ij}$$

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix}$$

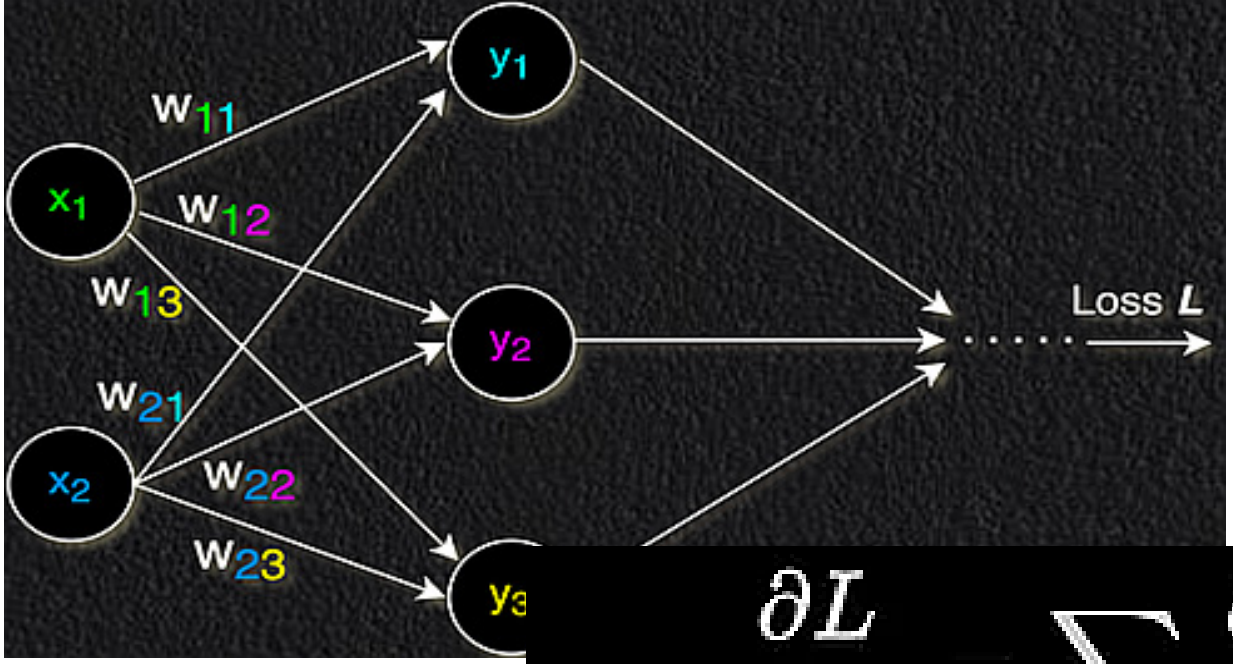
$$= \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}^T$$

$$X = [x_1, x_2, \ldots, x_m],\ Y = [y_1, y_2, \ldots, y_n],\ W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$



$$\frac{\partial L}{\partial w_{ij}} = \sum_k \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} x_i$$

only when k = j, y_k is connected to w_{ij}



$$\frac{\partial L}{\partial w_{ij}} = \sum_k \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} x_i$$

only when $\mathrm{k} = \mathrm{j}$, y_k is connected to w_{ij}

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \cdots & \frac{\partial L}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{m1}} & \cdots & \frac{\partial L}{\partial w_{mn}} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_1} & \cdots & \frac{\partial L}{\partial y_n} \end{bmatrix} = X^T \frac{\partial L}{\partial Y}$$

$$X = [x_1, x_2, \ldots, x_m],\ Y = [y_1, y_2, \ldots, y_n],\ W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

$$Y = XW + B$$

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}$$

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y}$$

$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \cdots & \frac{\partial L}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{m1}} & \cdots & \frac{\partial L}{\partial w_{mn}} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial y_1} & \cdots & \frac{\partial L}{\partial y_n} \end{bmatrix} = X^T \frac{\partial L}{\partial Y}$$

UNIVERSITY OF MICHIGAN

https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/linear-backprop.html#mjx-eqn-eq%3Afinal

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \quad W = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix}$$

$$Y = XW = \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} & x_{1,1}w_{1,2} + x_{1,2}w_{2,2} & x_{1,1}w_{1,3} + x_{1,2}w_{2,3} \\ x_{2,1}w_{1,1} + x_{2,2}w_{2,1} & x_{2,1}w_{1,2} + x_{2,2}w_{2,2} & x_{2,1}w_{1,3} + x_{2,2}w_{2,3} \end{pmatrix}$$

Jacobian

$$\frac{\partial L}{\partial W} = \frac{\partial Y}{\partial W} \frac{\partial L}{\partial Y}$$

$$\frac{\partial L}{\partial Y} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix}$$

$$W = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} \implies \frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \frac{\partial L}{\partial w_{1,3}} \\ \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \frac{\partial L}{\partial w_{1,3}} \end{pmatrix}$$

$$\frac{\partial L}{\partial w_{i,j}} = \sum_{i'=1}^{N} \sum_{j'=1}^{M} \frac{\partial L}{\partial y_{i',j'}} \frac{\partial y_{i',j'}}{\partial w_{i,j}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{i,j}}$$

$$\frac{\partial L}{\partial w_{1,1}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{1,1}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} x_{1,1} & 0 & 0 \\ x_{2,1} & 0 & 0 \end{pmatrix} = \frac{\partial L}{\partial y_{1,1}} x_{1,1} + \frac{\partial L}{\partial y_{2,1}} x_{2,1}$$

$$\frac{\partial L}{\partial w_{1,2}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{1,2}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} 0 & x_{1,1} & 0 \\ 0 & x_{2,1} & 0 \end{pmatrix} = \frac{\partial L}{\partial y_{1,2}} x_{1,1} + \frac{\partial L}{\partial y_{2,2}} x_{2,1}$$

$$\frac{\partial L}{\partial w_{1,3}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{1,3}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & x_{1,1} \\ 0 & 0 & x_{2,1} \end{pmatrix} = \frac{\partial L}{\partial y_{1,3}} x_{1,1} + \frac{\partial L}{\partial y_{2,3}} x_{2,1}$$

$$\frac{\partial L}{\partial w_{2,1}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{2,1}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} x_{1,2} & 0 & 0 \\ x_{2,2} & 0 & 0 \end{pmatrix} = \frac{\partial L}{\partial y_{1,1}} x_{1,2} + \frac{\partial L}{\partial y_{2,1}} x_{2,2}$$

$$\frac{\partial L}{\partial w_{2,2}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{2,2}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} 0 & x_{1,2} & 0 \\ 0 & x_{2,2} & 0 \end{pmatrix} = \frac{\partial L}{\partial y_{1,2}} x_{1,2} + \frac{\partial L}{\partial y_{2,2}} x_{2,2}$$

$$\frac{\partial L}{\partial w_{2,3}} = \frac{\partial L}{\partial Y} \cdot \frac{\partial Y}{\partial w_{2,3}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & x_{1,2} \\ 0 & 0 & x_{2,2} \end{pmatrix} = \frac{\partial L}{\partial y_{1,3}} x_{1,2} + \frac{\partial L}{\partial y_{2,3}} x_{2,2}$$

$$
\begin{aligned}
\frac{\partial L}{\partial W} &= \begin{pmatrix} \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \frac{\partial L}{\partial w_{1,3}} \\ \frac{\partial L}{\partial w_{2,1}} & \frac{\partial L}{\partial w_{2,2}} & \frac{\partial L}{\partial w_{2,3}} \end{pmatrix} \\
&= \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} x_{1,1} + \frac{\partial L}{\partial y_{2,1}} x_{2,1} & \frac{\partial L}{\partial y_{1,2}} x_{1,1} + \frac{\partial L}{\partial y_{2,2}} x_{2,1} & \frac{\partial L}{\partial y_{1,3}} x_{1,1} + \frac{\partial L}{\partial y_{2,3}} x_{2,1} \\ \frac{\partial L}{\partial y_{1,1}} x_{1,2} + \frac{\partial L}{\partial y_{2,1}} x_{2,2} & \frac{\partial L}{\partial y_{1,2}} x_{1,2} + \frac{\partial L}{\partial y_{2,2}} x_{2,2} & \frac{\partial L}{\partial y_{1,3}} x_{1,2} + \frac{\partial L}{\partial y_{2,3}} x_{2,2} \end{pmatrix} \\
&= \begin{pmatrix} x_{1,1} & x_{2,1} \\ x_{1,2} & x_{2,2} \end{pmatrix} \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \\
&= \boxed{X^T \frac{\partial L}{\partial Y}}
\end{aligned}
$$

$$
\frac{\partial L}{\partial W} = \frac{\partial Y}{\partial W} \frac{\partial L}{\partial Y}
$$

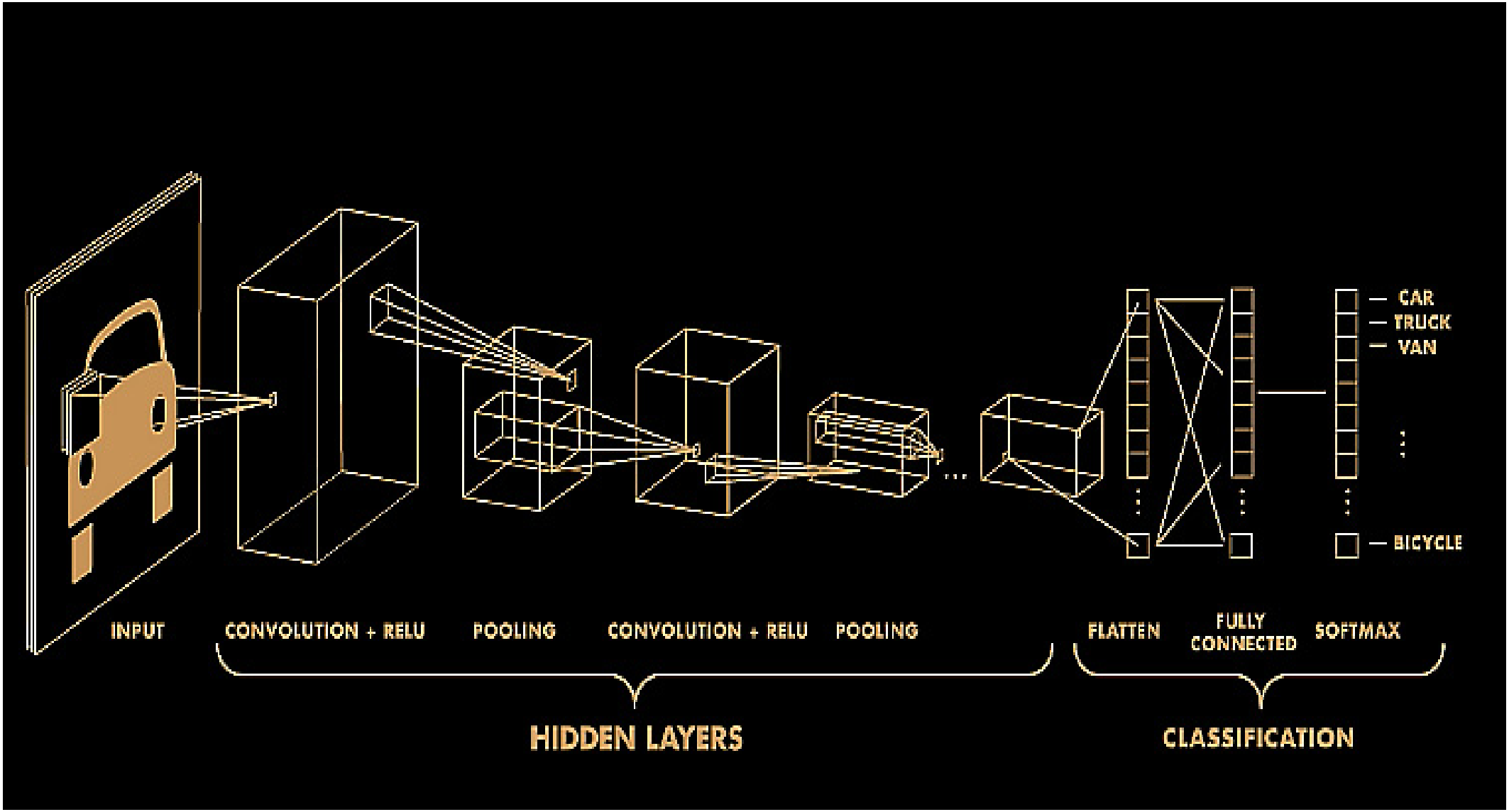
$Y = XW$:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} W^T$$

$$\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}$$

No need to use Jacobian (or even the Hessian)

$$\mathbf{J} = \frac{d\,\mathbf{f}(\mathbf{x})}{d\,\mathbf{x}} = \left[\frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} \cdots \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_u} \right] = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_u} \\ \vdots & & \vdots \\ \frac{\partial f_v(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_v(\mathbf{x})}{\partial x_u} \end{bmatrix}$$


CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
HIDDEN LAYERS
CLASSIFICATION

Various types of ANN Architectures:

- Boltzmann Machine,
- Hopfield Network
- CAM (Content Addressable memories);
- BAM (Bidirectional associative memory)
- SOM (self-organizing maps)
- Deep Belief Networks
- RBM, RBF

- CNN, Relu; RESNET, YOLO, SOLO, VGG, INCEPTION, Segnet, AlexNet,..
- GAN
- Auto-encoders (AE), VAE, ENC-DEC
- LSTM; Attn. modules