Chapter ML:IV

IV. Neural Networks

Multilayer Perceptron Basics

Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if $p+1$ real numbers, $w_0, w_1, \ldots, w_p$, exist such that the following conditions holds:

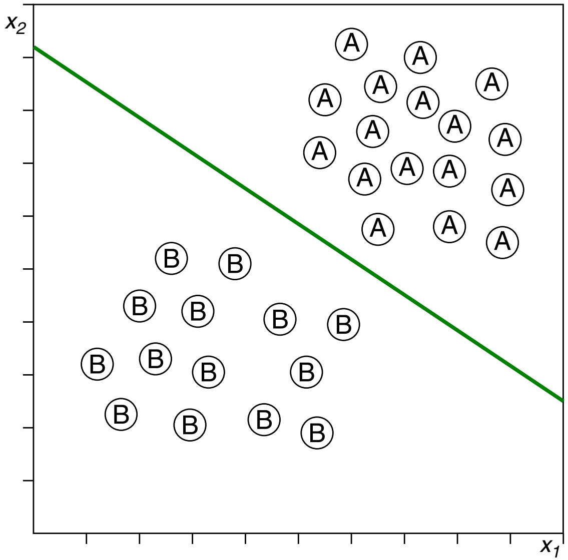
1. $\forall \mathbf{x} \in X_0$: $\sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1$: $\sum_{j=0}^{p} w_j x_j \geq 0$

Multilayer Perceptron Basics

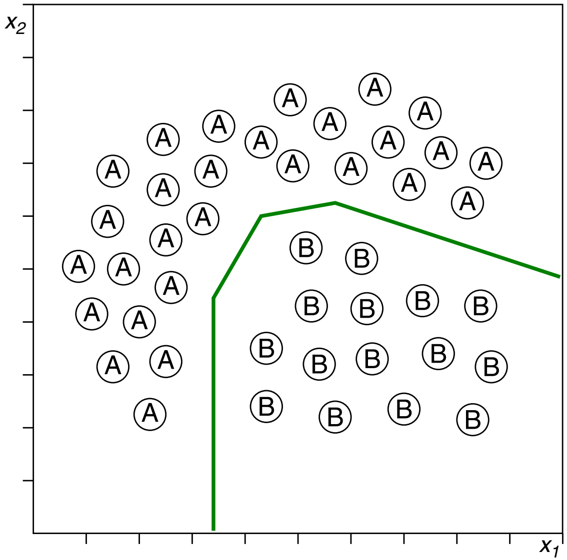
Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if $p+1$ real numbers, $w_0, w_1, \ldots, w_p$, exist such that the following conditions holds:

1. $\forall \mathbf{x} \in X_0$: $\sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1$: $\sum_{j=0}^{p} w_j x_j \geq 0$



linearly separable



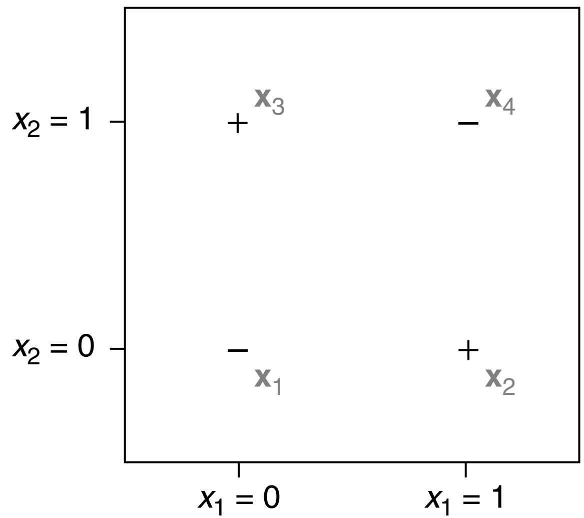
not linearly separable

Multilayer Perceptron Basics

Linear Separability (continued)

The *XOR* function defines two sets in the $\mathbf{R}^2$ that are not linearly separable:

	x_1	x_2	XOR	c
$\mathbf{x}_1$	0	0	0	$-$
$\mathbf{x}_2$	1	0	1	$+$
$\mathbf{x}_3$	0	1	1	$+$
$\mathbf{x}_4$	1	1	0	$-$

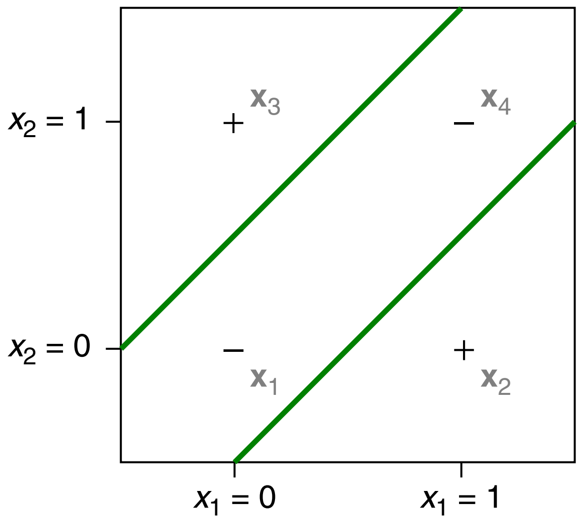


Multilayer Perceptron Basics

Linear Separability (continued)

The *XOR* function defines two sets in the $\mathbf{R}^2$ that are not linearly separable:

	x_1	x_2	XOR	c
$\mathbf{x}_1$	0	0	0	$-$
$\mathbf{x}_2$	1	0	1	$+$
$\mathbf{x}_3$	0	1	1	$+$
$\mathbf{x}_4$	1	1	0	$-$



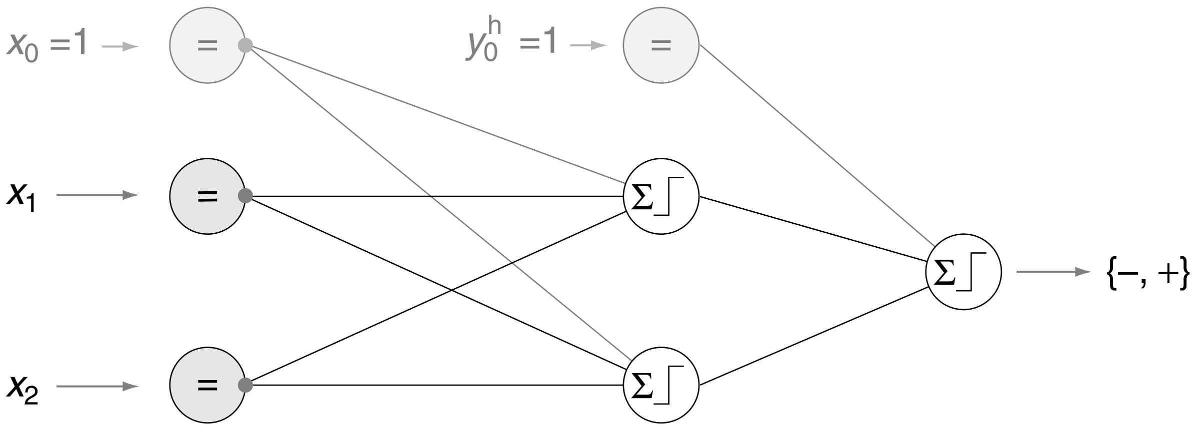
➜ Specification of several hyperplanes.

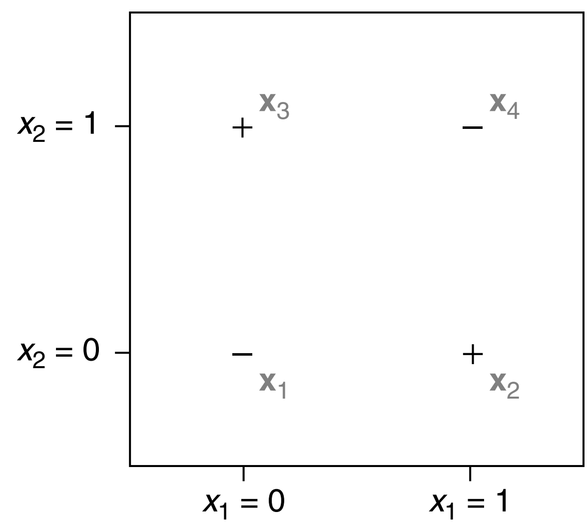
➜ Layered combination of several perceptrons: the multilayer perceptron.

Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:

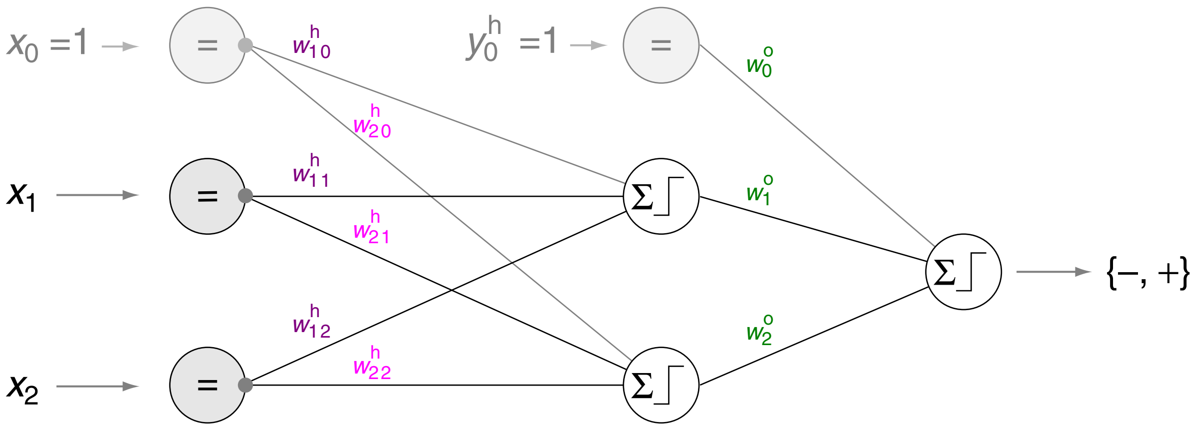


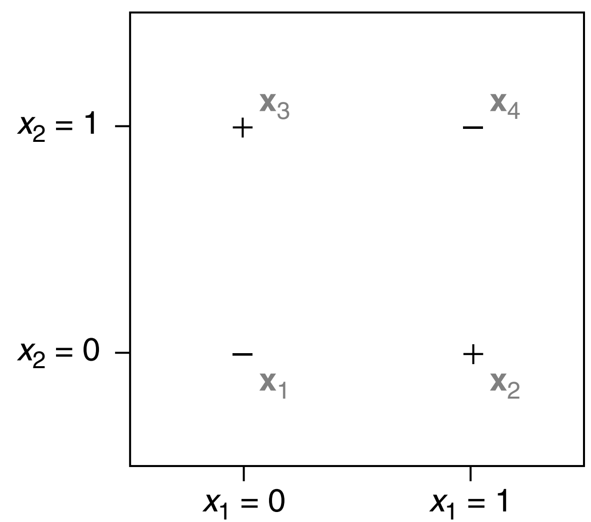


Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:

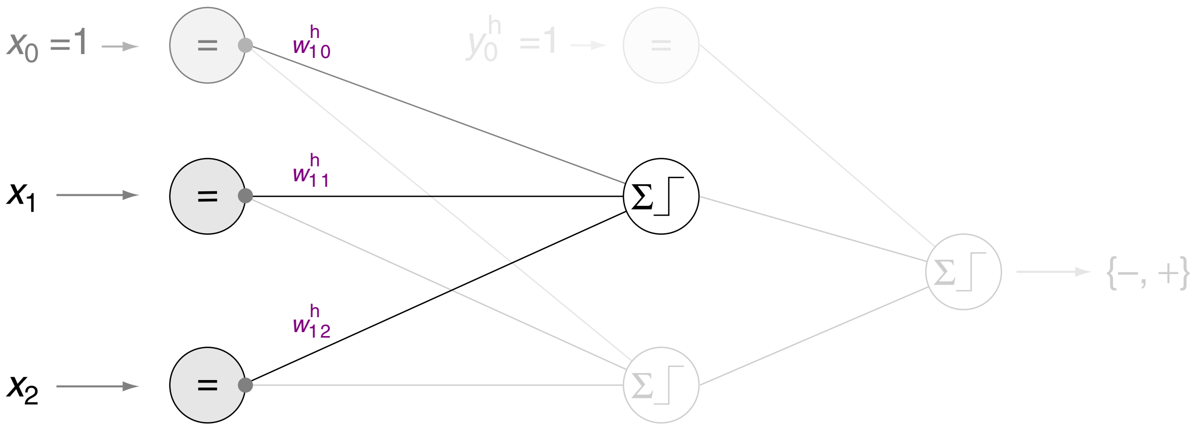


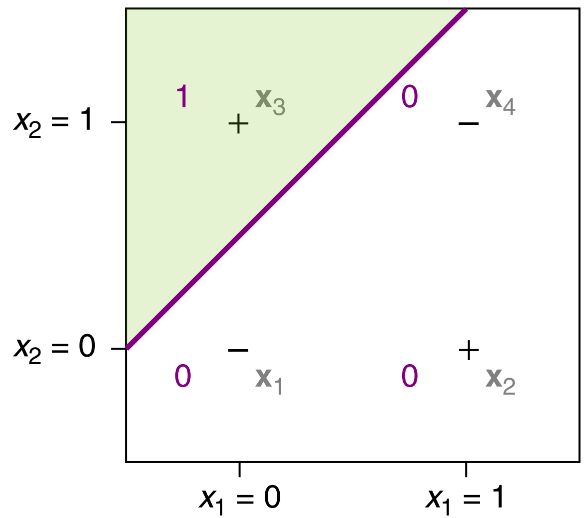


Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



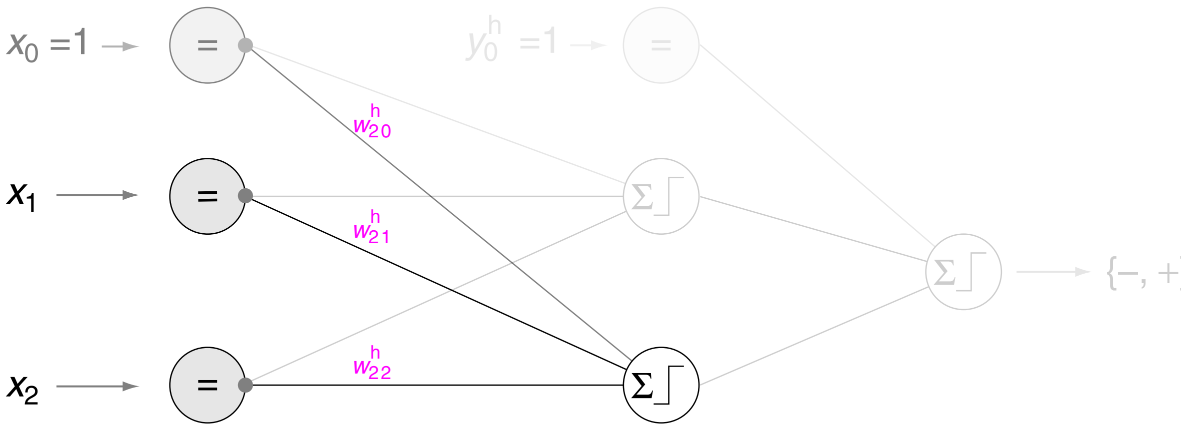


$$W^{\mathrm{h}} = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

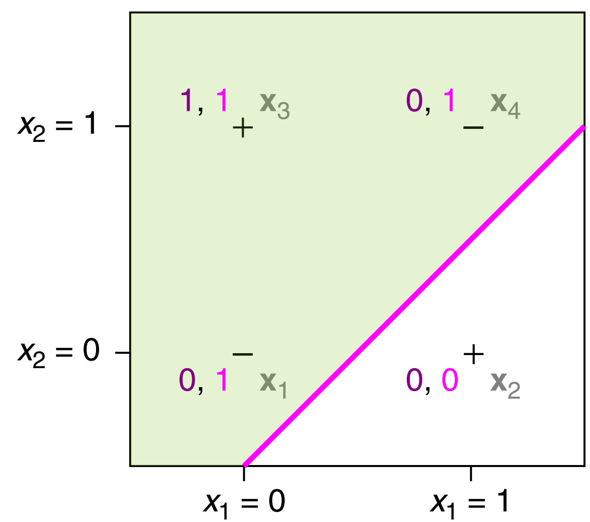
Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



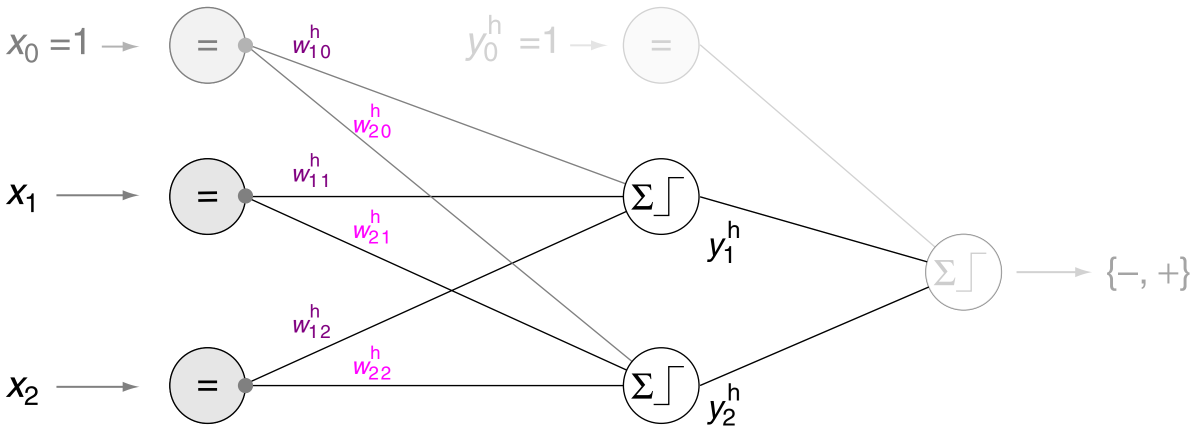
$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$



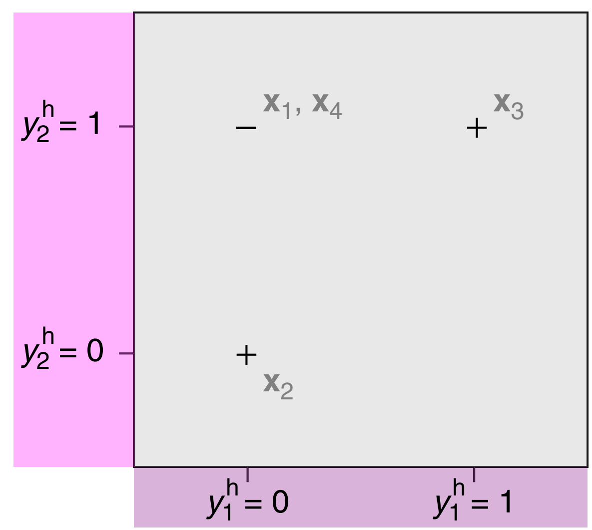
Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



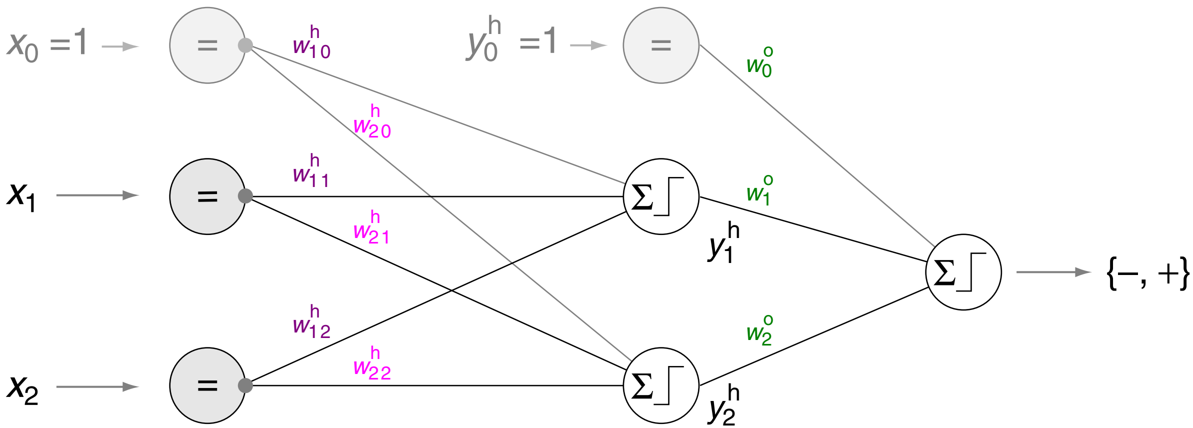
$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$



Multilayer Perceptron Basics

(1) Overcoming the Linear Separability Restriction

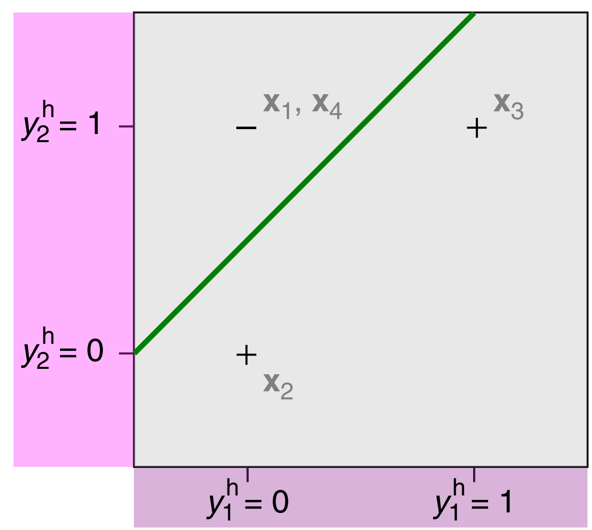
A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



$$y(\mathbf{x}) = \textit{heaviside}\left(W^o \begin{pmatrix} 1 \\ \textit{Heaviside}(W^h\,\mathbf{x}) \end{pmatrix}\right)$$

$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$W^o = \begin{bmatrix} 0.5 & 1 & -1 \end{bmatrix}$$



Remarks:

- The first, second, and third layer of the shown multilayer perceptron are called input, hidden, and output layer respectively. Here, in the example, the input layer is comprised of $p{+}1{=}3$ units, the hidden layer contains $l{+}1{=}3$ units, and the output layer consists of $k{=}1$ unit.

- Each input unit is connected via a weighted edge to all hidden units (except to the topmost hidden unit, which has a constant input $y_0^{\mathsf{h}} = 1$), resulting in six weights, organized as 2×3-matrix W^{h}. Each hidden unit is connected via a weighted edge to the output unit, resulting in three weights, organized as 1×3-matrix W^{o}.

- The input units perform no computation but only distribute the values x_0, x_1, x_2 to the next layer. The hidden units (again except the topmost unit) and the output unit apply the *heaviside* function to the sum of their weighted inputs and propagate the result.

  The nine weights $\mathbf{w} = (w_{10}^{\mathsf{h}}, \ldots, w_{22}^{\mathsf{h}}, w_1^{\mathsf{o}}, w_2^{\mathsf{o}}, w_3^{\mathsf{o}})$, organized as W^{h} and W^{o}, specify the multilayer perceptron (model function) $y(\mathbf{x})$ completely: $y(\mathbf{x}) = \textit{heaviside}(W^{\mathsf{o}} \binom{1}{\textit{Heaviside}(W^{\mathsf{h}}\,\mathbf{x})})$

- The function *Heaviside* (with capital H) denotes the extension of the scalar *heaviside* function to vectors.

  For $\mathbf{z} \in \mathbf{R}^d$ the function *Heaviside*$(\mathbf{z})$ is defined as $(\textit{heaviside}(z_1), \ldots, \textit{heaviside}(z_d))^T$.

Remarks (history) :

- The multilayer perceptron was presented by Rumelhart and McClelland in 1986. Earlier, but unnoticed, was a similar research work of Werbos and Parker [1974, 1982].
- Compared to a single perceptron, the multilayer perceptron poses a significantly more challenging training (= learning) problem, which requires continuous (and non-linear) threshold functions along with sophisticated learning strategies.
- Marvin Minsky and Seymour Papert in 1969 used the *XOR* problem to show the limitations of single perceptrons. Moreover, they assumed that extensions of the perceptron architecture (such as the multilayer perceptron) would be similarly limited as a single perceptron. A fatal mistake. In fact, they brought the research in this field to a halt that lasted 17 years. [Berkeley]

[Marvin Minsky: MIT Media Lab, Wikipedia]

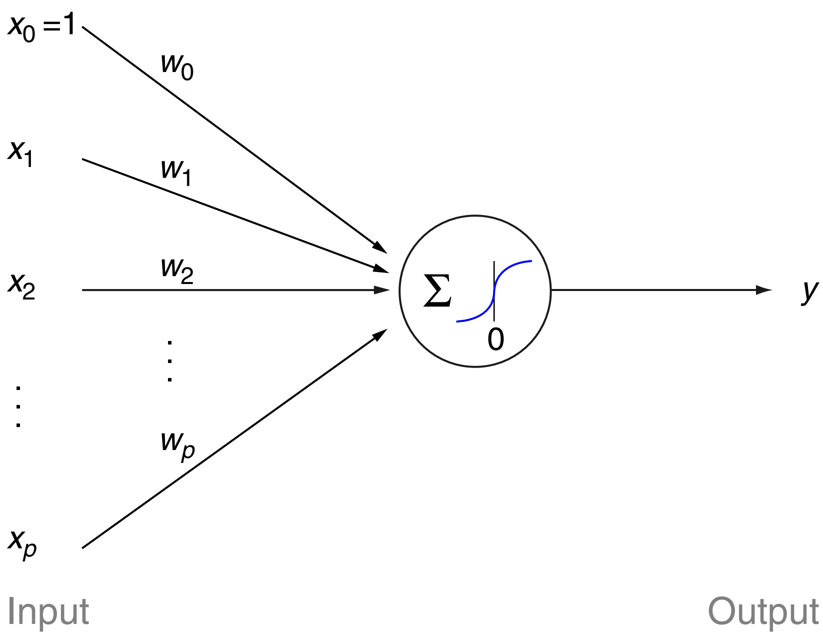
Multilayer Perceptron Basics

(2) Overcoming the Non-Differentiability Restriction

The sigmoid function $\sigma()$ as threshold function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \frac{d\,\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

➜ A perceptron with a non-linear *and* differentiable threshold function:



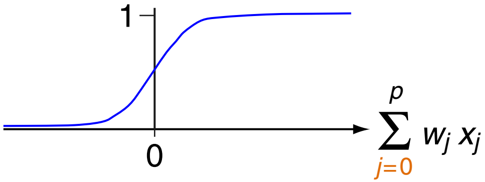
[heaviside]

Multilayer Perceptron Basics

(2) Overcoming the Non-Differentiability Restriction (continued)

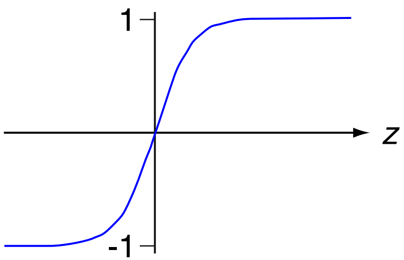
Computation of the perceptron output $y(\mathbf{x})$ with the sigmoid function $\sigma()$:

$$y(\mathbf{x}) \;=\; \sigma(\mathbf{w}^T\mathbf{x}) \;=\; \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$



An alternative to the sigmoid function is the $\tanh()$ function:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1}$$



Remarks:

- Employing a nonlinear function as threshold function in the perceptron, such as sigmoid or *heaviside*, is a prerequisite to synthesize complex nonlinear functions via layered composition.

- Note that a single perceptron with sigmoid activation is identical with the logistic regression model function.

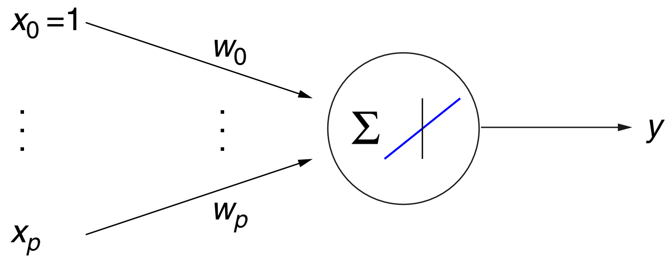
- The derivative of $\sigma()$ has a canonical form. It plays a central role for the computation of the gradient of the loss function in multilayer perceptrons. Derivation:

$$
\begin{aligned}
\frac{d\,\sigma(z)}{dz} &= \frac{d}{dz}\,\frac{1}{1+e^{-z}} = \frac{d}{dz}\,(1+e^{-z})^{-1} \\
&= -1 \cdot (1+e^{-z})^{-2} \cdot e^{-z} \cdot (-1) \\
&= \sigma(z) \cdot \sigma(z) \cdot e^{-z} \\
&= \sigma(z) \cdot \sigma(z) \cdot (1+e^{-z}-1) \\
&= \sigma(z) \cdot \sigma(z) \cdot (\sigma(z)^{-1}-1) \\
&= \sigma(z) \cdot (1-\sigma(z))
\end{aligned}
$$

Multilayer Perceptron Basics

(2) Overcoming the Non-Differentiability Restriction (continued)

Linear activation

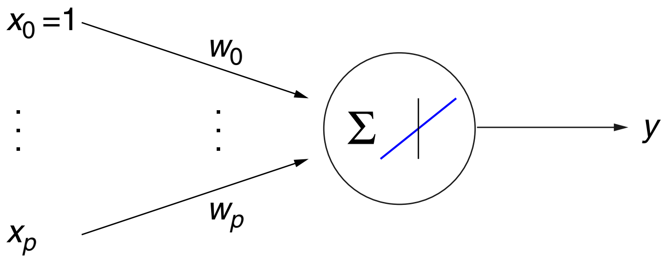


Linear regression

Multilayer Perceptron Basics

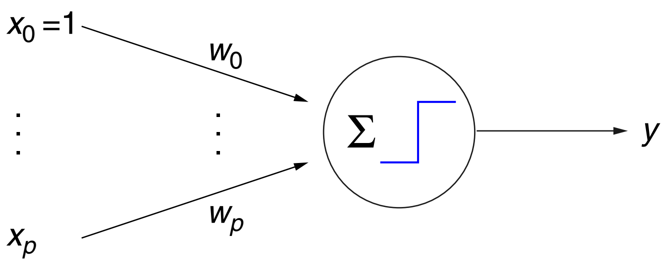
(2) Overcoming the Non-Differentiability Restriction (continued)

Linear activation



Linear regression

Heaviside activation

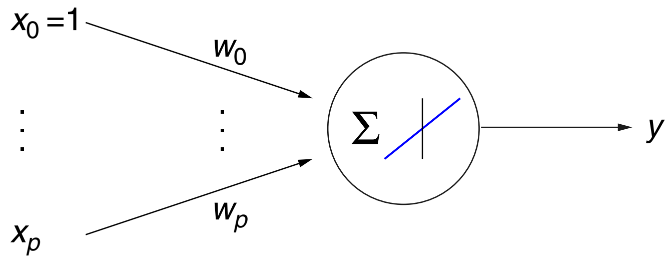


Perceptron algorithm

Multilayer Perceptron Basics

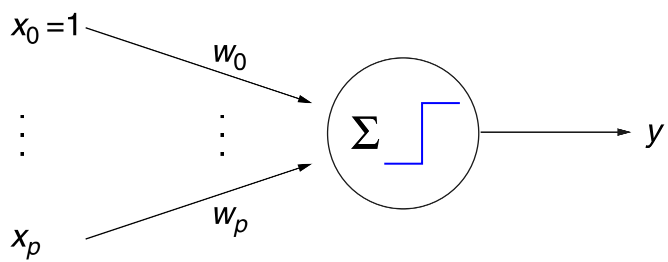
(2) Overcoming the Non-Differentiability Restriction (continued)

Linear activation



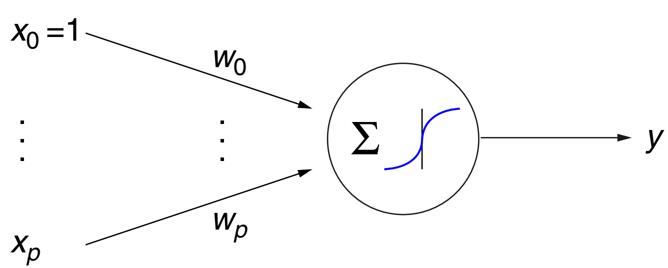
Linear regression

Heaviside activation



Perceptron algorithm

Sigmoid activation

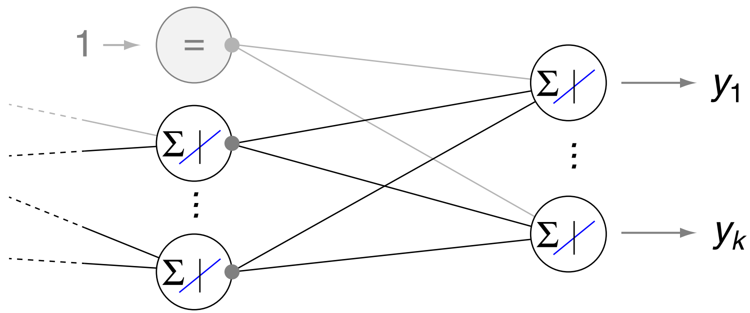


Logistic regression

Multilayer Perceptron Basics

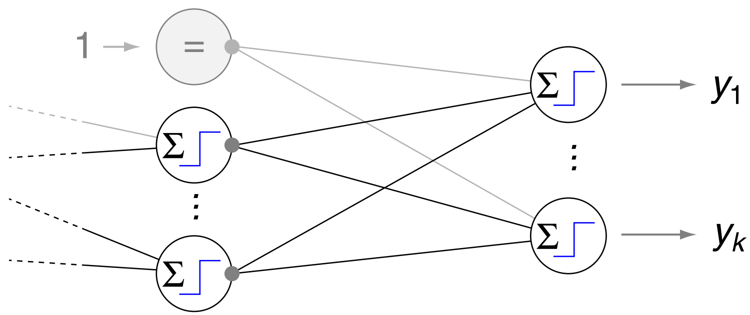
(2) Overcoming the Non-Differentiability Restriction (continued)

Network with linear units



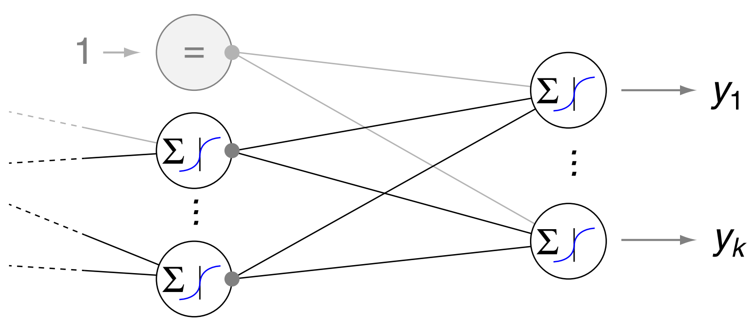
No decision power beyond a single hyperplane

Network with heaviside units



Nonlinear decision boundaries but no gradient information

Network with sigmoid units



Nonlinear decision boundaries and gradient information

Remarks (limitation of linear thresholds) :

- A multilayer perceptron with linear threshold functions can be expressed as a single linear function and hence is equivalent to the power of a single perceptron only.
- Consider the following exemplary composition of three linear functions as a multilayer perceptron with p input units, two hidden units, and one output unit: $y(\mathbf{x}) = W^{\mathsf{o}}\,[W^{\mathsf{h}}\,\mathbf{x}]$

  The weight matrices are as follows:

$$W^{\mathsf{h}} = \begin{bmatrix} w^{\mathsf{h}}_{11} & \ldots & w^{\mathsf{h}}_{1p} \\ w^{\mathsf{h}}_{21} & \ldots & w^{\mathsf{h}}_{1p} \end{bmatrix}, \qquad W^{\mathsf{o}} = \begin{bmatrix} w^{\mathsf{o}}_1 & w^{\mathsf{o}}_2 \end{bmatrix}$$

  A straightforward derivation then yields:

$$\begin{aligned} y(\mathbf{x}) = W^{\mathsf{o}}\,[W^{\mathsf{h}}\,\mathbf{x}] \;&=\; \begin{bmatrix} w^{\mathsf{o}}_1 & w^{\mathsf{o}}_2 \end{bmatrix} \begin{bmatrix} w^{\mathsf{h}}_{11}x_1 & + \ldots + & w^{\mathsf{h}}_{1p}x_p \\ w^{\mathsf{h}}_{21}x_1 & + \ldots + & w^{\mathsf{h}}_{1p}x_p \end{bmatrix} \\ &=\; w^{\mathsf{o}}_1 w^{\mathsf{h}}_{11}x_1 + \;\ldots\; + w^{\mathsf{o}}_1 w^{\mathsf{h}}_{1p}x_p \;+\; w^{\mathsf{o}}_2 w^{\mathsf{h}}_{21}x_1 + \;\ldots\; + w^{\mathsf{o}}_2 w^{\mathsf{h}}_{1p}x_p \\ &=\; (w^{\mathsf{o}}_1 w^{\mathsf{h}}_{11} + w^{\mathsf{o}}_2 w^{\mathsf{h}}_{21})x_1 + \;\ldots\; + (w^{\mathsf{o}}_1 w^{\mathsf{h}}_{1p} + w^{\mathsf{o}}_2 w^{\mathsf{h}}_{1p})x_p \\ &=\; w_1 x_1 + \;\ldots\; + w_p x_p \;=\; \mathbf{w}^T\mathbf{x} \end{aligned}$$

Multilayer Perceptron Basics

Unrestricted Classification Problems

Setting:

- X is a multiset of feature vectors from an inner product space $\mathbf{X}$, $\mathbf{X} \subseteq \mathbf{R}^p$.
- $C = \{0, 1\}^k$ is the set of all multiclass labelings for k classes.
- $D = \{(\mathbf{x}_1, \mathbf{c}_1), \ldots, (\mathbf{x}_n, \mathbf{c}_n)\} \subseteq X \times C$ is a multiset of examples.

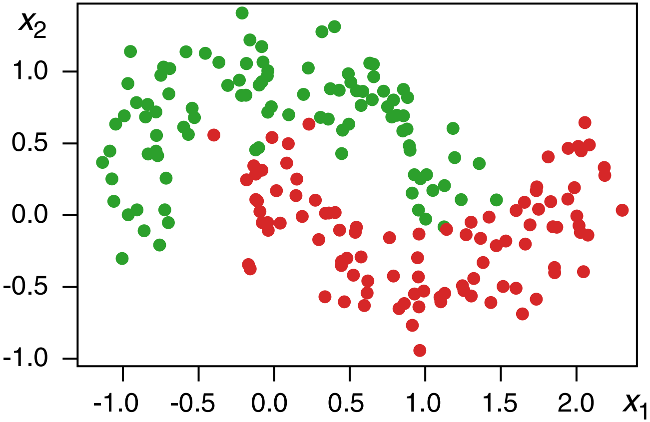
Learning task:

- Fit D using a multilayer perceptron $\mathbf{y}()$ with a sigmoid activation function.

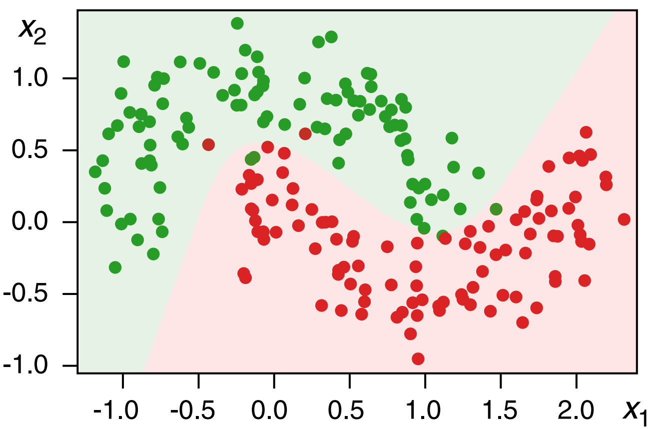
Multilayer Perceptron Basics

Unrestricted Classification Problems: Illustration

Two-class classification problem:



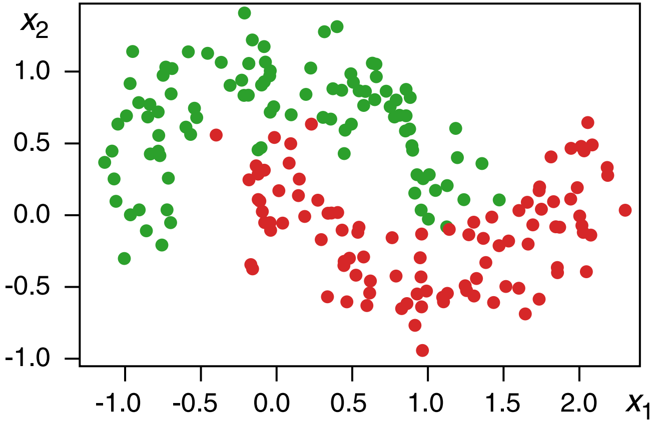
Separated classes:



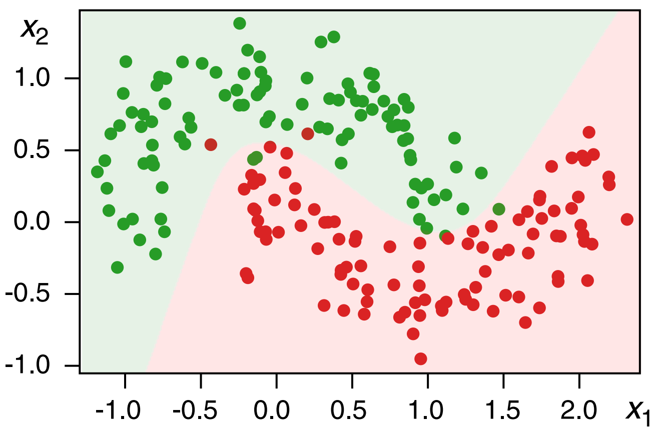
Multilayer Perceptron Basics

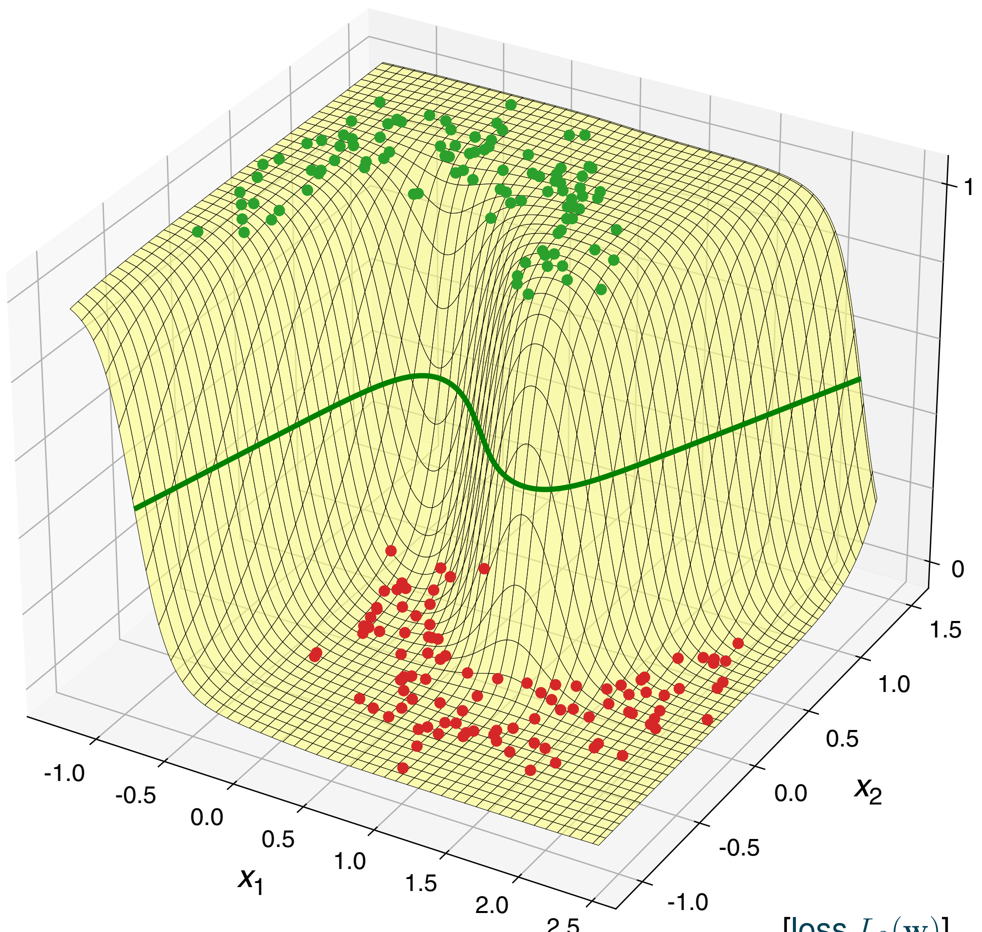
Unrestricted Classification Problems: Illustration

Two-class classification problem:



Separated classes:





[loss $L_2(\mathbf{w})$]

Chapter ML:IV

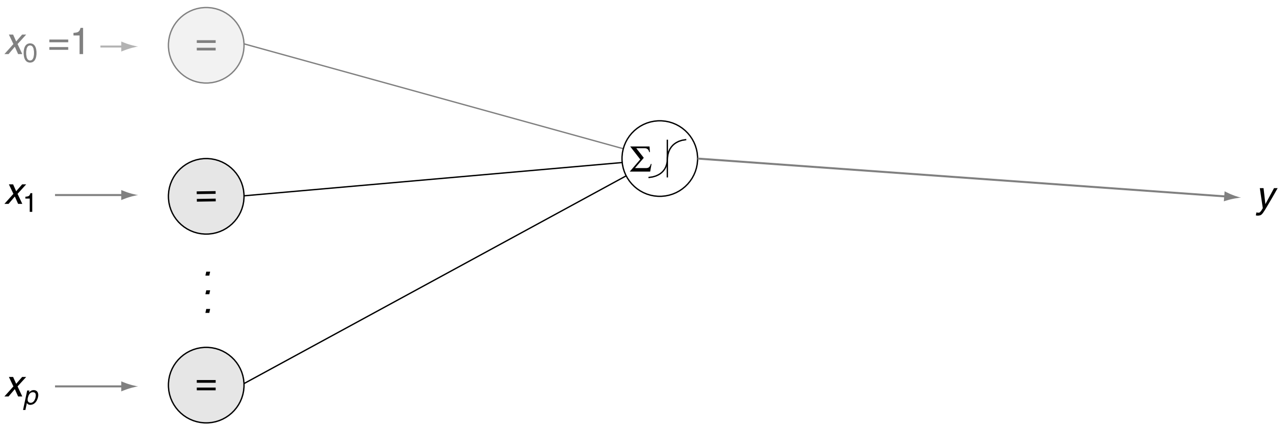
IV. Neural Networks

- Perceptron Learning
- Multilayer Perceptron Basics
- Multilayer Perceptron with Two Layers
- Multilayer Perceptron at Arbitrary Depth
- Advanced MLPs
- Automatic Gradient Computation

Multilayer Perceptron with Two Layers

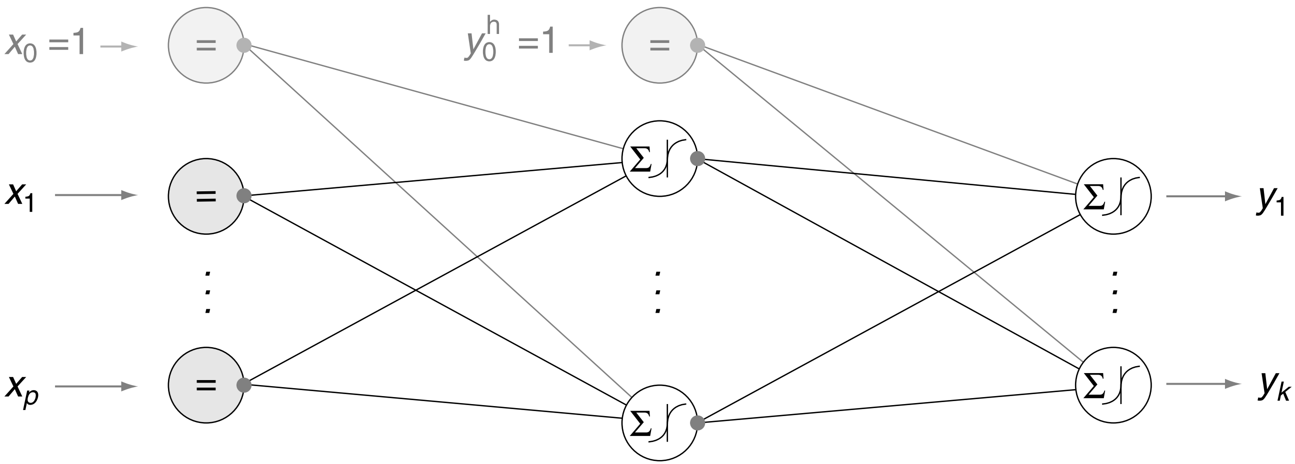
Network Architecture

A single perceptron $y(\mathbf{x})$:



Multilayer Perceptron with Two Layers
Network Architecture

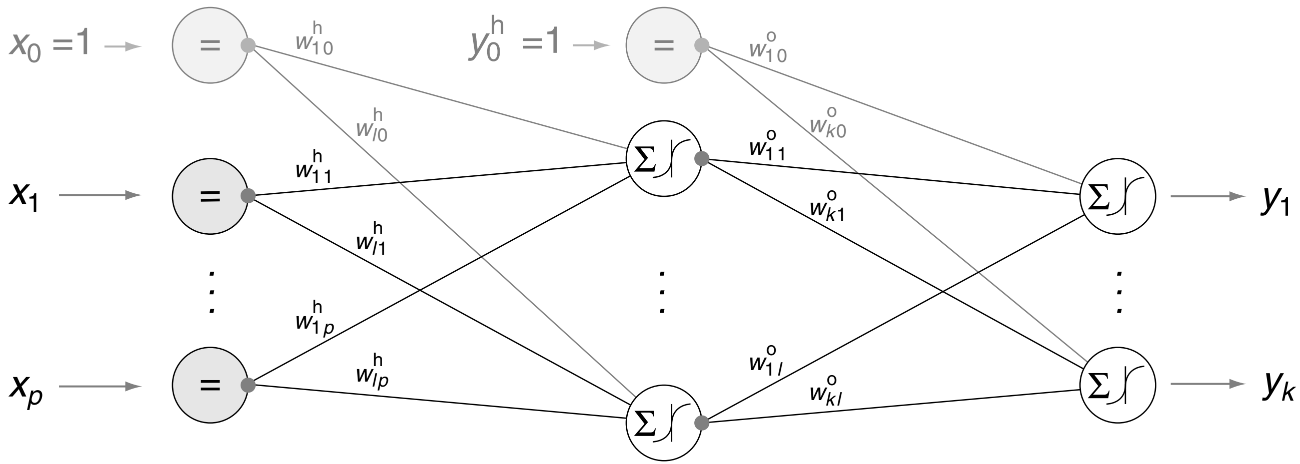
Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with a hidden layer and k-dimensional output layer:



Multilayer Perceptron with Two Layers

Network Architecture

Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with a hidden layer and k-dimensional output layer:

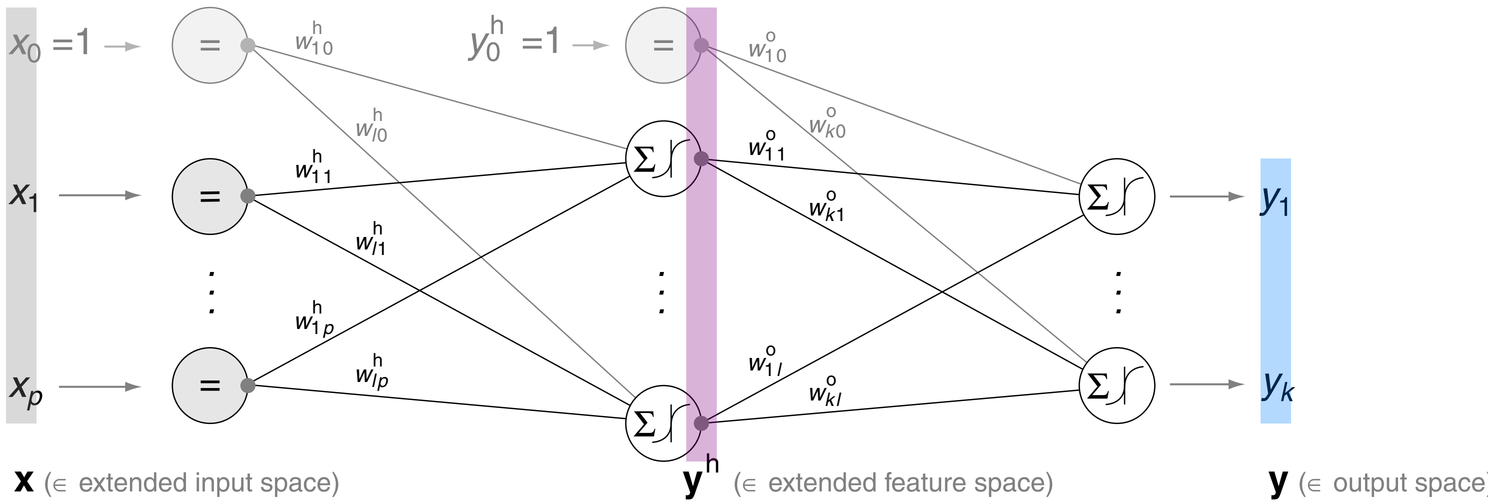


Parameters $\mathbf{w}$: $W^h \in \mathbf{R}^{l\times(p+1)}$ $W^o \in \mathbf{R}^{k\times(l+1)}$

Multilayer Perceptron with Two Layers

Network Architecture

Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with a hidden layer and k-dimensional output layer:

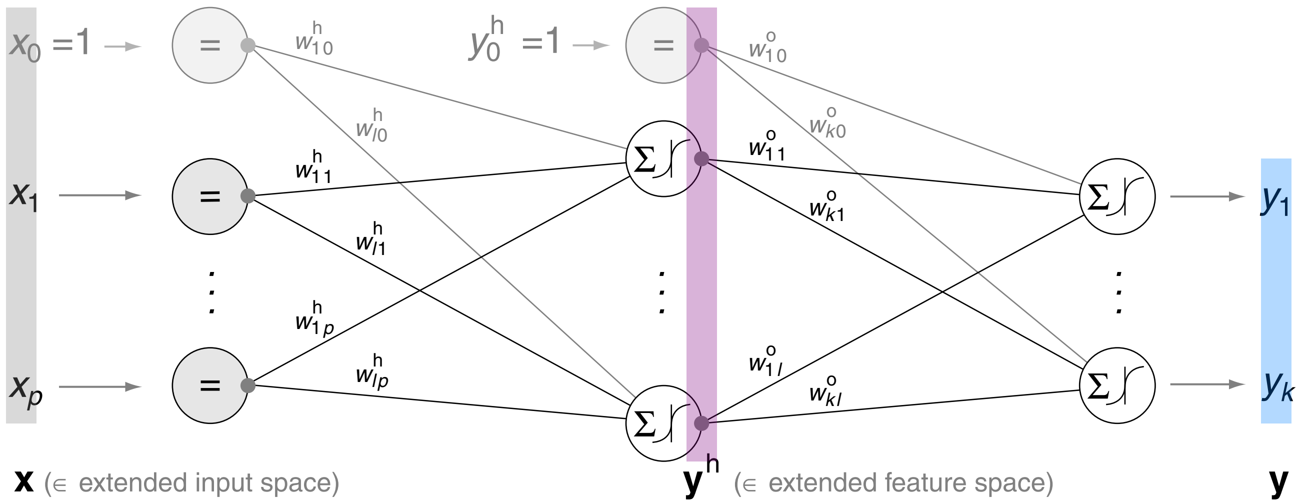


Parameters $\mathbf{w}$: $W^h \in \mathbf{R}^{l \times (p+1)}$ $W^o \in \mathbf{R}^{k \times (l+1)}$

Multilayer Perceptron with Two Layers

(1) Forward Propagation [mlp arbitrary depth]

Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with a hidden layer and k-dimensional output layer:



Parameters $\mathbf{w}$: $\underbrace{\quad}_{W^h \in \mathbf{R}^{l\times(p+1)}}$ $\underbrace{\quad}_{W^o \in \mathbf{R}^{k\times(l+1)}}$

Model function evaluation (= forward propagation) :

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^o\, \mathbf{y}^h(\mathbf{x})\right) = \boldsymbol{\sigma}\left(W^o \begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^h\, \mathbf{x}\right)\end{pmatrix}\right)$$

Remarks:

- Each input unit is connected to the hidden units $1, \ldots, l$, resulting in $l \cdot (p+1)$ weights, organized as matrix $W^{\mathsf{h}} \in \mathbf{R}^{l \times (p+1)}$. Each hidden unit is connected to the output units $1, \ldots, k$, resulting in $k \cdot (l+1)$ weights, organized as matrix $W^{\mathsf{o}} \in \mathbf{R}^{k \times (l+1)}$.

- The hidden units and the output unit(s) apply the (vectorial) sigmoid function, $\boldsymbol{\sigma}$, to the sum of their weighted inputs and propagate the result as $\mathbf{y}^{\mathsf{h}}$ and $\mathbf{y}$ respectively. For $\mathbf{z} \in \mathbf{R}^d$ the vectorial sigmoid function $\boldsymbol{\sigma}(\mathbf{z})$ is defined as $(\sigma(z_1), \ldots, \sigma(z_d))^T$.

- The parameter vector $\mathbf{w} = (w^{\mathsf{h}}_{10}, \ldots, w^{\mathsf{h}}_{lp}, w^{\mathsf{o}}_{10}, \ldots, w^{\mathsf{o}}_{kl})$, organized as matrices W^{h} and W^{o}, specifies the multilayer perceptron (model function) $\mathbf{y}(\mathbf{x})$ completely: $\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{o}} \binom{1}{\boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x})})$.

- The shown architecture with k output units allows for the distinction of k classes, either within an exclusive class assignment setting or within a multi-label setting. In the former setting a so-called “softmax layer” can be added subsequent to the output layer to directly return the class label $1, \ldots, k$.

- The non-linear characteristic of the sigmoid function allows for networks that approximate every (computable) function. For this capability only three “active” layers are required, i.e., two layers with hidden units and one layer with output units. Keyword: universal approximator [Kolmogorov theorem, 1957]

- Multilayer perceptrons are also called multilayer networks or (artificial) neural networks, ANN for short.

Multilayer Perceptron with Two Layers

(1) Forward Propagation (continued) [network architecture]

(a) Propagate $\mathbf{x}$ from input to hidden layer: [$\mathrm{IGD}_{\mathrm{MLP}_2}$ algorithm, Line 5]

$$W^{\mathsf{h}} \in \mathbf{R}^{l\times(p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{h}} & \ldots & w_{1p}^{\mathsf{h}} \\ & \vdots & \\ w_{l0}^{\mathsf{h}} & \ldots & w_{lp}^{\mathsf{h}} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y_1^{\mathsf{h}} \\ \vdots \\ y_l^{\mathsf{h}} \end{bmatrix}$$

Multilayer Perceptron with Two Layers

(1) Forward Propagation (continued) [network architecture]

(a) Propagate $\mathbf{x}$ from input to hidden layer: [$\mathrm{IGD}_{\mathrm{MLP}_2}$ algorithm, Line 5]

$$W^{\mathsf{h}} \in \mathbf{R}^{l\times(p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{h}}_{10} & \dots & w^{\mathsf{h}}_{1p} \\ & \vdots & \\ w^{\mathsf{h}}_{l0} & \dots & w^{\mathsf{h}}_{lp} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y^{\mathsf{h}}_1 \\ \vdots \\ y^{\mathsf{h}}_l \end{bmatrix}$$

(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: [$\mathrm{IGD}_{\mathrm{MLP}_2}$ algorithm, Line 5]

$$W^{\mathsf{o}} \in \mathbf{R}^{k\times(l+1)} \qquad \mathbf{y}^{\mathsf{h}} \in \mathbf{R}^{l+1} \qquad \mathbf{y} \in \mathbf{R}^{k}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{o}}_{10} & \dots & w^{\mathsf{o}}_{1l} \\ & \vdots & \\ w^{\mathsf{o}}_{k0} & \dots & w^{\mathsf{o}}_{kl} \end{bmatrix} \begin{bmatrix} 1 \\ y^{\mathsf{h}}_1 \\ \vdots \\ y^{\mathsf{h}}_l \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Multilayer Perceptron with Two Layers

(1) Forward Propagation: Batch Mode [network architecture]

(a) Propagate $\mathbf{x}$ from input to hidden layer: [$\text{IGD}_{\text{MLP}_2}$ algorithm, Line 5]

$$W^{\mathsf{h}} \in \mathbf{R}^{l\times(p+1)} \qquad X \subset \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{h}}_{10} & \ldots & w^{\mathsf{h}}_{1p} \\ & \vdots & \\ w^{\mathsf{h}}_{l0} & \ldots & w^{\mathsf{h}}_{lp} \end{bmatrix} \begin{bmatrix} 1 & \ldots & 1 \\ x_{11} & \ldots & x_{1n} \\ & \vdots & \\ x_{p1} & \ldots & x_{pn} \end{bmatrix}\right) = \begin{bmatrix} y^{\mathsf{h}}_{11} & \ldots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \ldots & y^{\mathsf{h}}_{ln} \end{bmatrix}$$

(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: [$\text{IGD}_{\text{MLP}_2}$ algorithm, Line 5]

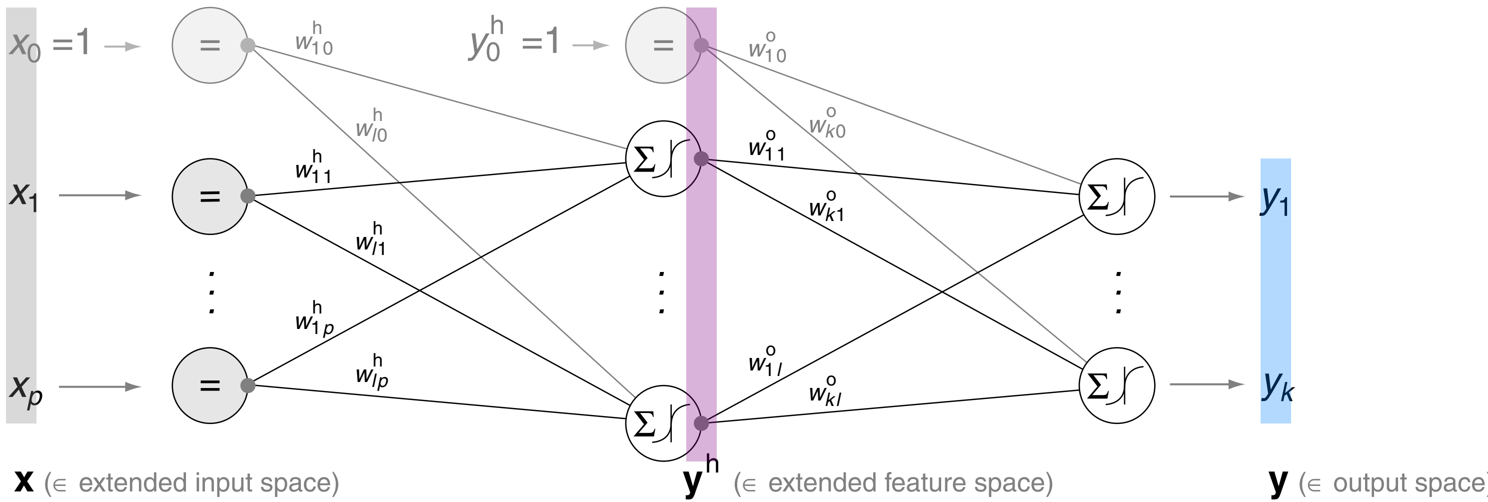
$$W^{\mathsf{o}} \in \mathbf{R}^{k\times(l+1)}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{o}}_{10} & \ldots & w^{\mathsf{o}}_{1l} \\ & \vdots & \\ w^{\mathsf{o}}_{k0} & \ldots & w^{\mathsf{o}}_{kl} \end{bmatrix} \begin{bmatrix} 1 & \ldots & 1 \\ y^{\mathsf{h}}_{11} & \ldots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \ldots & y^{\mathsf{h}}_{ln} \end{bmatrix}\right) = \begin{bmatrix} y_{11} & \ldots & y_{1n} \\ & \vdots & \\ y_{k1} & \ldots & y_{kn} \end{bmatrix}$$

Multilayer Perceptron with Two Layers

(2) Backpropagation [linear regression] [mlp arbitrary depth]

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:

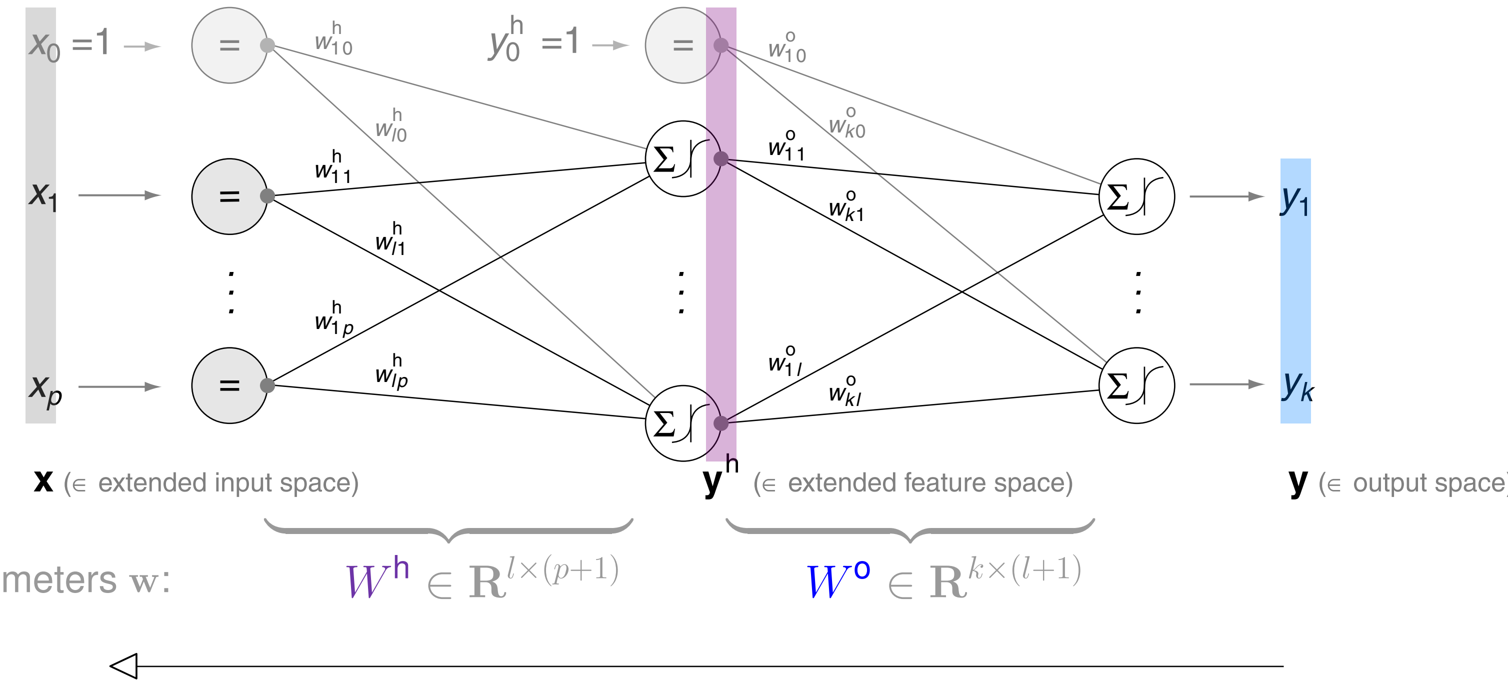


Parameters $\mathbf{w}$: $W^h \in \mathbf{R}^{l\times(p+1)}$ $W^o \in \mathbf{R}^{k\times(l+1)}$

Multilayer Perceptron with Two Layers

(2) Backpropagation [linear regression] [mlp arbitrary depth]

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:



Calculation of derivatives (= backpropagation) wrt. the global squared loss:

$$L_2(\mathbf{w}) = \frac{1}{2}\cdot \text{RSS}(\mathbf{w}) = \frac{1}{2}\cdot \sum_{(\mathbf{x},\mathbf{c})\in D} \sum_{u=1}^{k} (c_u - y_u(\mathbf{x}))^2$$

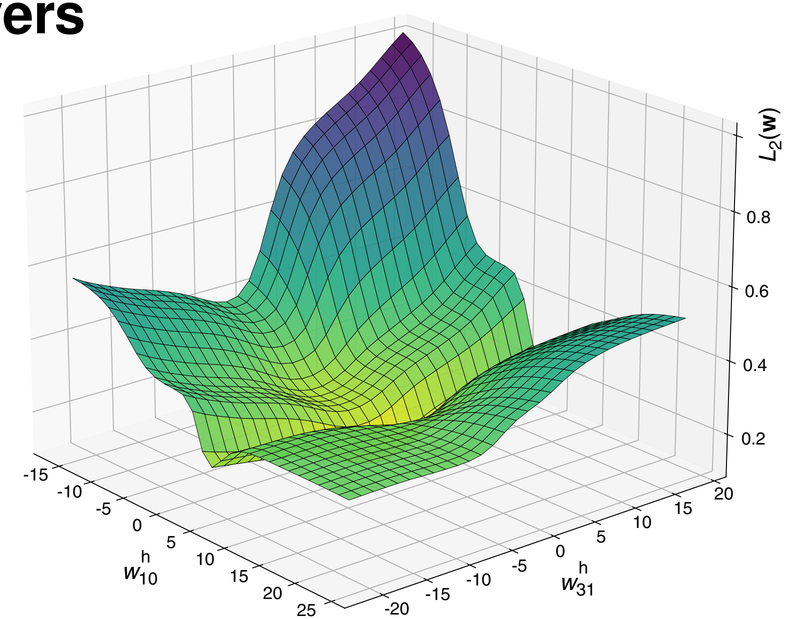
Multilayer Perceptron with Two Layers

(2) Backpropagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\begin{pmatrix}1\\ \boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x})\end{pmatrix}\right)$$

$$L_2(\mathbf{w}) = \frac{1}{2}\cdot\sum_{(\mathbf{x},\mathbf{c})\in D}\sum_{u=1}^{k}(c_u - y_u(\mathbf{x}))^2$$

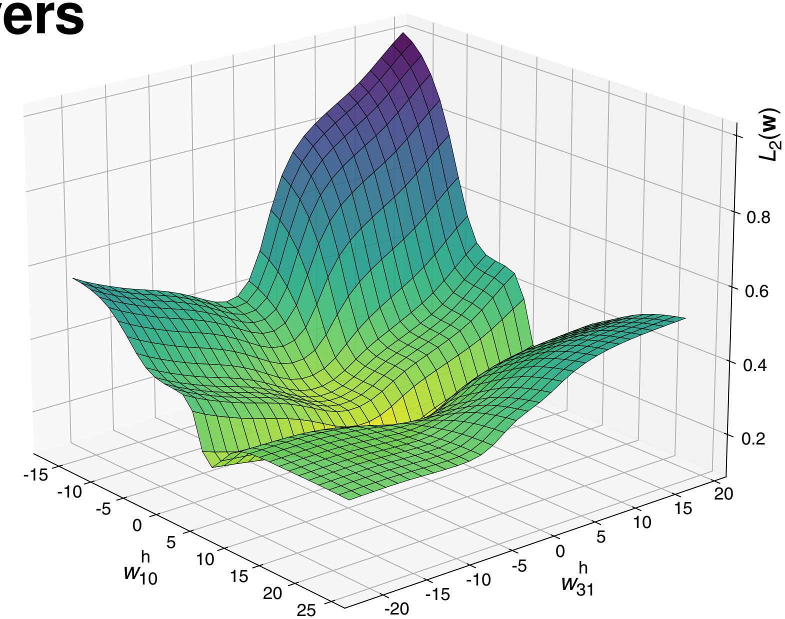


[model function $\mathbf{y}(\mathbf{x})$]

Multilayer Perceptron with Two Layers

(2) Backpropagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:



[model function $\mathbf{y}(\mathbf{x})$]

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\left(\frac{1}{\boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x})}\right)\right)$$

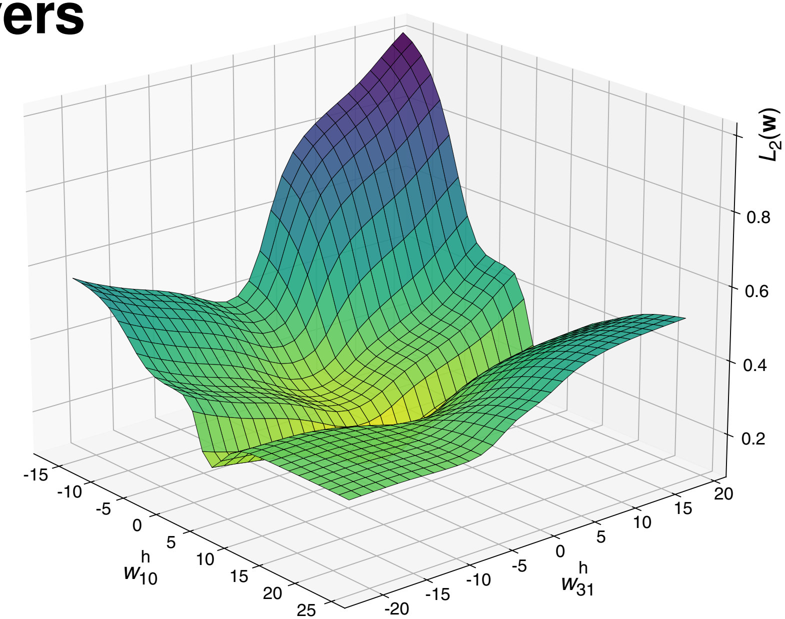
$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c})\in D} \sum_{u=1}^{k} (c_u - y_u(\mathbf{x}))^2$$

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{o}}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{kl}^{\mathsf{o}}}, \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{lp}^{\mathsf{h}}}\right)^T$$

Multilayer Perceptron with Two Layers

(2) Backpropagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:



[model function $\mathbf{y}(\mathbf{x})$]

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\left(\begin{smallmatrix}1\\ \boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x})\end{smallmatrix}\right)\right)$$

$$L_2(\mathbf{w}) = \frac{1}{2}\cdot\sum_{(\mathbf{x},\mathbf{c})\in D}\sum_{u=1}^{k}(c_u - y_u(\mathbf{x}))^2$$

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}},\ldots,\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}},\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}},\ldots,\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}}\right)^T$$

(a) Gradient in direction of W^{o}, written as matrix:

$$\begin{bmatrix}\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{1l}}\\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{k0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}}\end{bmatrix} \equiv \nabla^{\mathsf{o}} L_2(\mathbf{w})$$

(b) Gradient in direction of W^{h}:

$$\begin{bmatrix}\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{1p}}\\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{l0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}}\end{bmatrix} \equiv \nabla^{\mathsf{h}} L_2(\mathbf{w})$$

Remarks:

- "Backpropagation" is short for "backward propagation of errors". Backpropagation is a method of calculating the derivatives (the gradient).

- Basically, the computation of the gradient $\nabla L_2(\mathbf{w})$ is independent of the organization of the weights in matrices W^{h} and W^{o} of a network (model function) $\mathbf{y}(\mathbf{x})$. Adopt the following view instead:

  To calculate $\nabla L_2(\mathbf{w})$ one has to calculate each of its components $\partial L_2(\mathbf{w})/\partial w$, $w \in \mathbf{w}$, since each weight (parameter) has a certain impact on the global loss $L_2(\mathbf{w})$ of the network. This impact—as well as the computation of this impact—is different for different weights, but it is canonical for all weights of the same layer though: observe that each weight w influences "only" its direct and indirect successor nodes, and that the structure of the influenced successor graph is identical for all weights of the same layer.

  Hence it is convenient, but not necessary, to process the components of the gradient layer-wise (matrix-wise), as $\nabla^{\text{o}} L_2(\mathbf{w})$ and $\nabla^{\text{h}} L_2(\mathbf{w})$ respectively. Even more, due to the network structure of the model function $\mathbf{y}(\mathbf{x})$ only two cases need to be distinguished when deriving the partial derivative $\partial L_2(\mathbf{w})/\partial w$ of an arbitrary weight $w \in \mathbf{w}$: (a) w belongs to the output layer, or (b) w belongs to some hidden layer.

- The derivation of the gradient for the two-layer MLP (and hence the weight update processed in the IGD algorithm) is given in the following, as special case of the derivation of the gradient for MLPs at arbitrary depth.

Multilayer Perceptron with Two Layers

(2) Backpropagation (continued) [linear regression] [mlp arbitrary depth]

(a) Update of weight matrix W^{o} : [$\mathrm{IGD}_{\mathrm{MLP}_2}$ algorithm, Lines 7+8]

$$W^{\mathrm{o}} = W^{\mathrm{o}} + \Delta W^{\mathrm{o}},$$

using the ∇^{o}-gradient of the loss function $L_2(\mathbf{w})$ to take the steepest descent:

$$\Delta W^{\mathrm{o}} \; = \; -\eta \cdot \nabla^{\mathrm{o}} L_2(\mathbf{w})$$

Multilayer Perceptron with Two Layers

(2) Backpropagation (continued) [linear regression] [mlp arbitrary depth]

(a) Update of weight matrix W^{o} : [IGD$_{\mathrm{MLP}_2}$ algorithm, Lines 7+8]

$$W^{\mathrm{o}} = W^{\mathrm{o}} + \Delta W^{\mathrm{o}},$$

using the ∇^{o}-gradient of the loss function $L_2(\mathbf{w})$ to take the steepest descent:

$$\Delta W^{\mathrm{o}} = -\eta \cdot \nabla^{\mathrm{o}} L_2(\mathbf{w})$$

$$= -\eta \cdot \begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathrm{o}}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{1l}^{\mathrm{o}}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w_{k0}^{\mathrm{o}}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{kl}^{\mathrm{o}}} \end{bmatrix}$$

$\vdots$ [derivation]

$$= \eta \cdot \sum_{D} \underbrace{\left[(\mathbf{c} - \mathbf{y}(\mathbf{x})) \odot \mathbf{y}(\mathbf{x}) \odot (1 - \mathbf{y}(\mathbf{x}))\right]}_{\delta^{\mathrm{o}}} \otimes \mathbf{y}^{\mathrm{h}}$$

Multilayer Perceptron with Two Layers

(2) Backpropagation (continued) [mlp arbitrary depth]

(b) Update of weight matrix W^{h} : [$\text{IGD}_{\text{MLP}_2}$ algorithm, Lines 7+8]

$$W^{\mathsf{h}} = W^{\mathsf{h}} + \Delta W^{\mathsf{h}},$$

using the ∇^{h}-gradient of the loss function $L_2(\mathbf{w})$ to take the steepest descent:

$$\Delta W^{\mathsf{h}} = -\eta \cdot \nabla^{\mathsf{h}} L_2(\mathbf{w})$$

$$= -\eta \cdot \begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{1p}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{l0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}} \end{bmatrix}$$

$\vdots$ [derivation]

$$= \eta \cdot \sum_D \underbrace{\left[\left(\left(W^{\mathsf{o}}\right)^T \boldsymbol{\delta}^{\mathsf{o}}\right) \odot \mathbf{y}^{\mathsf{h}}(\mathbf{x}) \odot \left(1 - \mathbf{y}^{\mathsf{h}}(\mathbf{x})\right)\right]_{1,\ldots,l}}_{\boldsymbol{\delta}^{\mathsf{h}}} \otimes \mathbf{x}$$

Multilayer Perceptron with Two Layers

The IGD Algorithm

Algorithm:	$\text{IGD}_{\text{MLP}_2}$	Incremental Gradient Descent for the two-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units. (= hypothesis)

initialize_random_weights$(W^{\text{h}}, W^{\text{o}})$, $t = 0$

REPEAT

  $t = t + 1$

  FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO

  ENDDO

UNTIL(*convergence*$(D, \mathbf{y}(), t)$)

return$(W^{\text{h}}, W^{\text{o}})$

[Python code]

Multilayer Perceptron with Two Layers

The IGD Algorithm (continued) [mlp arbitrary depth]

Algorithm:	$\text{IGD}_{\text{MLP}_2}$	Incremental Gradient Descent for the two-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l\cdot(p+1)$ hidden and $k\cdot(l+1)$ output layer units. (= hypothesis)

1. *initialize_random_weights*$(W^{\text{h}}, W^{\text{o}})$, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
5. $\quad\quad \mathbf{y}^{\text{h}}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\text{h}}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$
$\quad\quad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\text{o}}\,\mathbf{y}^{\text{h}}(\mathbf{x}))$
6.
7a.
7b.
8.
9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\text{h}}, W^{\text{o}})$

[Python code]

Multilayer Perceptron with Two Layers

The IGD Algorithm (continued) [mlp arbitrary depth]

Algorithm: $\text{IGD}_{\text{MLP}_2}$ Incremental Gradient Descent for the two-layer MLP.

Input: D Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.

η Learning rate, a small positive constant.

Output: $W^{\mathsf{h}}, W^{\mathsf{o}}$ Weights of $l\cdot(p+1)$ hidden and $k\cdot(l+1)$ output layer units. (= hypothesis)

```
1.  initialize_random_weights(W^h, W^o),  t = 0
2.  REPEAT
3.    t = t + 1
4.    FOREACH (x, c) ∈ D DO
5.      y^h(x) = (1, σ(W^h x))  // forward propagation; x is extended by x_0 = 1
        y(x) = σ(W^o y^h(x))
6.      δ = c − y(x)
7a.

7b.

8.
9.    ENDDO
10. UNTIL(convergence(D, y(), t))
11. return(W^h, W^o)
```

[Python code]

Multilayer Perceptron with Two Layers

The IGD Algorithm (continued) [mlp arbitrary depth]

Algorithm:	$\text{IGD}_{\text{MLP}_2}$	Incremental Gradient Descent for the two-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}}, W^{\mathsf{o}}$	Weights of $l\cdot(p+1)$ hidden and $k\cdot(l+1)$ output layer units. (= hypothesis)

1. *initialize_random_weights*$(W^{\mathsf{h}}, W^{\mathsf{o}})$, $t = 0$

2. **REPEAT**

3. $\quad t = t + 1$

4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO

5. $\quad\quad \mathbf{y}^{\mathsf{h}}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}}\,\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$

   $\quad\quad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{o}}\,\mathbf{y}^{\mathsf{h}}(\mathbf{x}))$

6. $\quad\quad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$

7a. $\quad\quad \boldsymbol{\delta}^{\mathsf{o}} = \boldsymbol{\delta} \odot \mathbf{y}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}(\mathbf{x}))$ // backpropagation (Steps 7a+7b)

   $\quad\quad \boldsymbol{\delta}^{\mathsf{h}} = [((W^{\mathsf{o}})^T\,\boldsymbol{\delta}^{\mathsf{o}}) \odot \mathbf{y}^{\mathsf{h}} \odot (\mathbf{1} - \mathbf{y}^{\mathsf{h}})]_{1,\ldots,l}$

7b. $\quad\quad \Delta W^{\mathsf{h}} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}} \otimes \mathbf{x})$

   $\quad\quad \Delta W^{\mathsf{o}} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{o}} \otimes \mathbf{y}^{\mathsf{h}}(\mathbf{x}))$

8.

9. $\quad$ ENDDO

10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)

11. *return*$(W^{\mathsf{h}}, W^{\mathsf{o}})$

[Python code]

Multilayer Perceptron with Two Layers

The IGD Algorithm (continued) [mlp arbitrary depth]

Algorithm:	$\text{IGD}_{\text{MLP}_2}$	Incremental Gradient Descent for the two-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units. (= hypothesis)

1. *initialize_random_weights*$(W^{\text{h}}, W^{\text{o}})$, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
5. $\quad\quad \mathbf{y}^{\text{h}}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\text{h}}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$
   $\quad\quad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\text{o}}\,\mathbf{y}^{\text{h}}(\mathbf{x}))$
6. $\quad\quad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$
7a. $\quad\quad \boldsymbol{\delta}^{\text{o}} = \boldsymbol{\delta} \odot \mathbf{y}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}(\mathbf{x}))$ // backpropagation (Steps 7a+7b)
   $\quad\quad \boldsymbol{\delta}^{\text{h}} = [((W^{\text{o}})^T \boldsymbol{\delta}^{\text{o}}) \odot \mathbf{y}^{\text{h}} \odot (\mathbf{1} - \mathbf{y}^{\text{h}})]_{1,\ldots,l}$
7b. $\quad\quad \Delta W^{\text{h}} = \eta \cdot (\boldsymbol{\delta}^{\text{h}} \otimes \mathbf{x})$
   $\quad\quad \Delta W^{\text{o}} = \eta \cdot (\boldsymbol{\delta}^{\text{o}} \otimes \mathbf{y}^{\text{h}}(\mathbf{x}))$
8. $\quad\quad W^{\text{h}} = W^{\text{h}} + \Delta W^{\text{h}}$, $\quad W^{\text{o}} = W^{\text{o}} + \Delta W^{\text{o}}$
9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\text{h}}, W^{\text{o}})$

[Python code]

Multilayer Perceptron with Two Layers

The IGD Algorithm (continued) [mlp arbitrary depth]

Algorithm:	$\text{IGD}_{\text{MLP}_2}$	Incremental Gradient Descent for the two-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units. (= hypothesis)

1. *initialize_random_weights*$(W^{\text{h}}, W^{\text{o}})$, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
5. Model function evaluation.
6. Calculation of residual vector.

7a.

7b. Calculation of derivative of the loss.

8. Parameter vector update $\widehat{=}$ one gradient step down.
9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\text{h}}, W^{\text{o}})$

[Python code]

Remarks:

- The symbol »$\odot$« denotes the Hadamard product, also known as the element-wise or the Schur product. It is a binary operation that takes two matrices of the same dimensions and produces another matrix of the same dimension as the operands, where each element is the product of the respective elements of the two original matrices. [Wikipedia]

- The symbol »$\otimes$« denotes the dyadic product, also called outer product or tensor product. The dyadic product takes two vectors and returns a second order tensor, called a dyadic in this context: $\mathbf{v} \otimes \mathbf{w} \equiv \mathbf{v}\mathbf{w}^T$. [Wikipedia]

- $[W]_{1,\ldots,l}$ denotes the projection operator, which returns the rows 1 through l of matrix W as a new matrix.

- ΔW and $\scriptstyle\Delta W$ indicate an update of the weight matrix per batch, D, or per instance, $(\mathbf{x}, \mathbf{c}) \in D$, respectively.

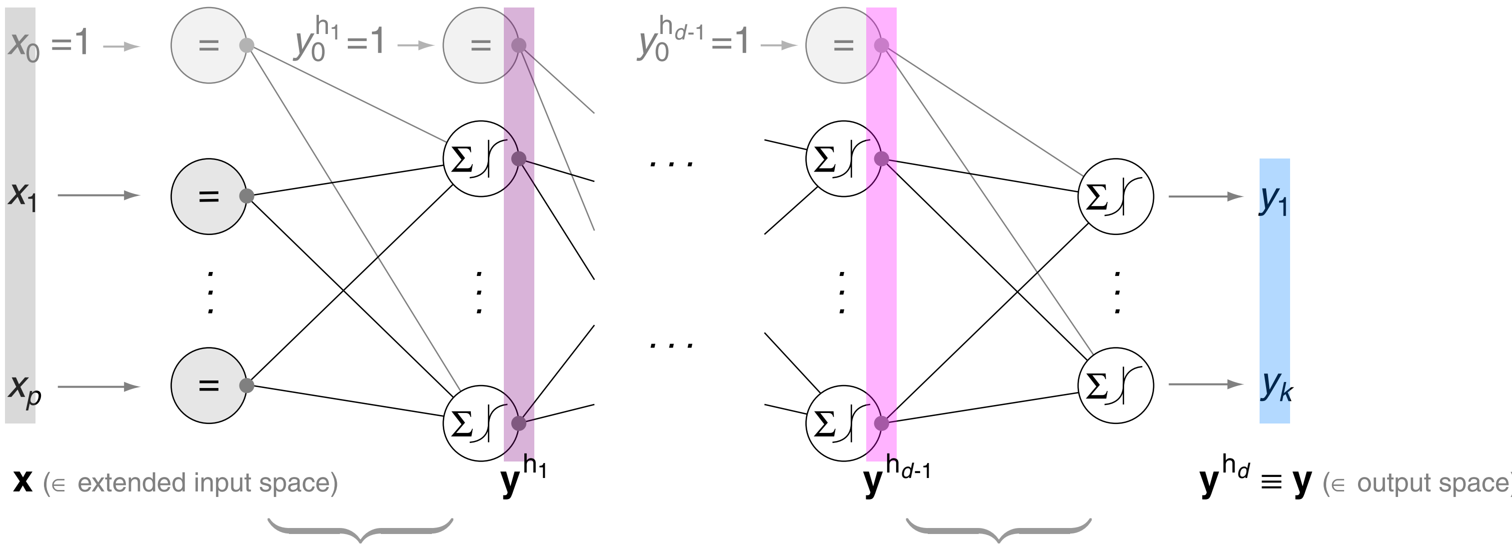
Chapter ML:IV

IV. Neural Networks

Multilayer Perceptron at Arbitrary Depth

Network Architecture [mlp two layers]

Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with d layers and k-dimensional output:

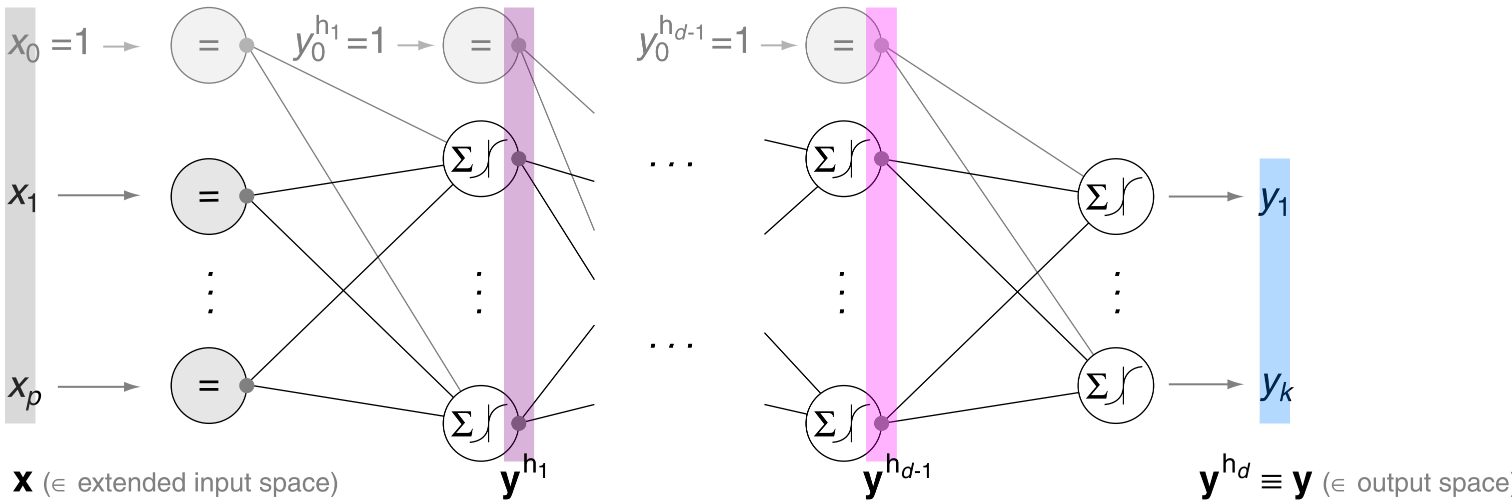


Parameters $\mathbf{w}$: $W^{h_1} \in \mathbf{R}^{l_1 \times (p+1)}$ $W^{h_d} \equiv W^{o} \in \mathbf{R}^{k \times (l_{d-1}+1)}$

Multilayer Perceptron at Arbitrary Depth

(1) Forward Propagation [mlp two layers]

Multilayer perceptron $\mathbf{y}(\mathbf{x})$ with d layers and k-dimensional output:



Parameters $\mathbf{w}$: $W^{h_1} \in \mathbf{R}^{l_1 \times (p+1)}$ $W^{h_d} \equiv W^{o} \in \mathbf{R}^{k \times (l_{d-1}+1)}$

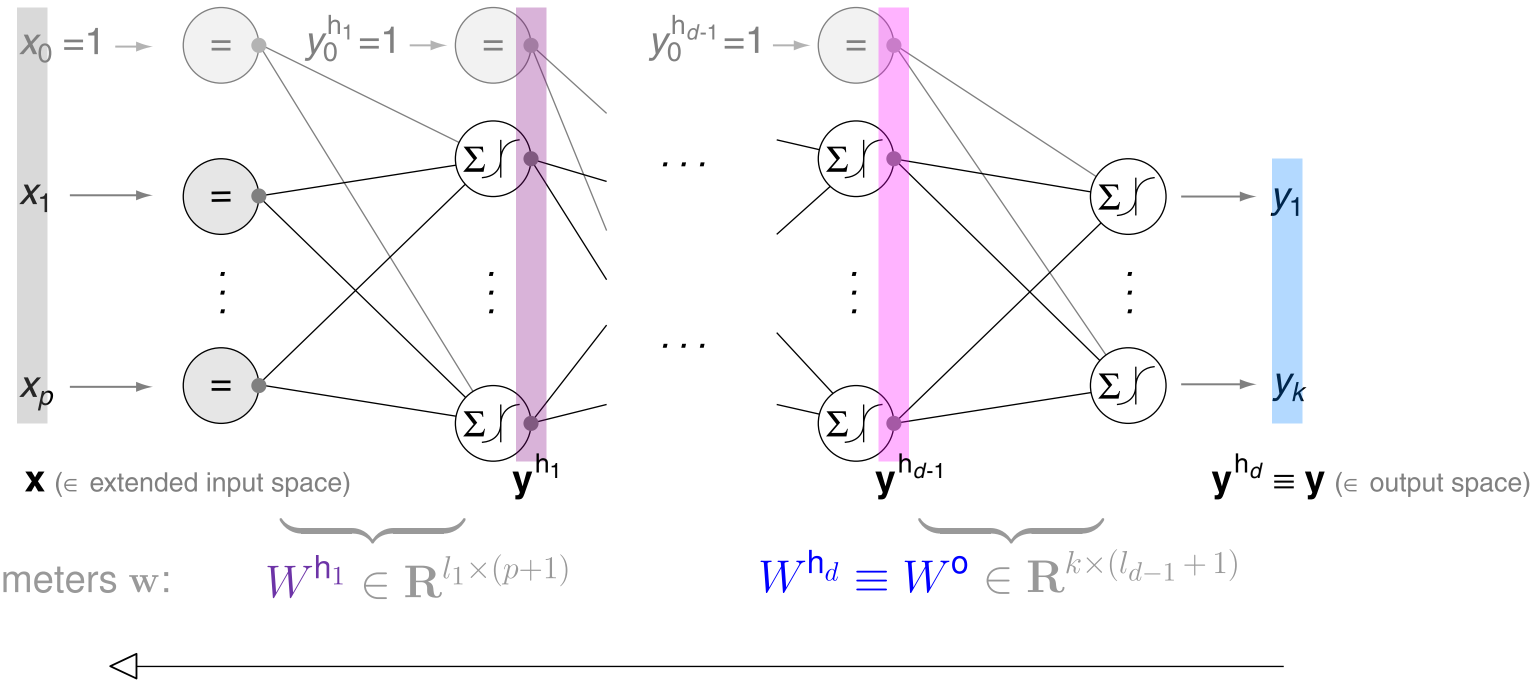
Model function evaluation (= forward propagation) :

$$\mathbf{y}^{h_d}(\mathbf{x}) \equiv \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{h_d}\, \mathbf{y}^{h_{d-1}}(\mathbf{x})\right) = \ldots = \boldsymbol{\sigma}\left(W^{h_d}\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(\ldots\begin{pmatrix}1\\ \boldsymbol{\sigma}(W^{h_1}\,\mathbf{x})\end{pmatrix}\ldots\right)\end{pmatrix}\right)$$

Multilayer Perceptron at Arbitrary Depth

(2) Backpropagation [mlp two layers]

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:



Calculation of derivatives (= backpropagation) wrt. the global squared loss:

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \text{RSS}(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c}) \in D} \sum_{u=1}^{k} (c_u - y_u(\mathbf{x}))^2$$

Multilayer Perceptron at Arbitrary Depth

(2) Backpropagation (continued) [mlp two layers]

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{l_1 p}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_d}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{k l_{d-1}}^{\mathsf{h}_d}} \right)^T \quad \text{where } l_s = \textit{no.\_rows}(W^{\mathsf{h}_s})$$

Multilayer Perceptron at Arbitrary Depth

(2) Backpropagation (continued) [mlp two layers]

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{l_1 p}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_d}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{k l_{d-1}}^{\mathsf{h}_d}} \right)^T \quad \text{where } l_s = \textit{no.\_rows}(W^{\mathsf{h}_s})$$

Update of weight matrix W^{h_s}, $1 \leq s \leq d$: [$\text{IGD}_{\text{MLP}_d}$ algorithm, Lines 7+8]

$$W^{\mathsf{h}_s} = W^{\mathsf{h}_s} + \Delta W^{\mathsf{h}_s},$$

using the ∇^{h_s}-gradient of the loss function $L_2(\mathbf{w})$ to take the steepest descent:

$$\Delta W^{\mathsf{h}_s} \; = \; -\eta \cdot \nabla^{\mathsf{h}_s} L_2(\mathbf{w})$$

Multilayer Perceptron at Arbitrary Depth

(2) Backpropagation (continued) [mlp two layers]

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{l_1 p}^{\mathsf{h}_1}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_d}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_{k l_{d-1}}^{\mathsf{h}_d}} \right)^T \quad \text{where } l_s = \textit{no.\_rows}(W^{\mathsf{h}_s})$$

Update of weight matrix W^{h_s}, $1 \leq s \leq d$: [IGD$_{\text{MLP}_d}$ algorithm, Lines 7+8]

$$W^{\mathsf{h}_s} = W^{\mathsf{h}_s} + \Delta W^{\mathsf{h}_s},$$

using the ∇^{h_s}-gradient of the loss function $L_2(\mathbf{w})$ to take the steepest descent:

$$\begin{aligned} \Delta W^{\mathsf{h}_s} &= -\eta \cdot \nabla^{\mathsf{h}_s} L_2(\mathbf{w}) \\ &= -\eta \cdot \begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}_s}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{1 l_{s-1}}^{\mathsf{h}_s}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w_{l_s 0}^{\mathsf{h}_s}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{l_s l_{s-1}}^{\mathsf{h}_s}} \end{bmatrix}, \quad \text{where} \quad \begin{array}{l} l_s = \textit{no.\_rows}(W^{\mathsf{h}_s}), \\ \mathbf{y}^{\mathsf{h}_0} \equiv \mathbf{x}, \\ \mathbf{y}^{\mathsf{h}_d} \equiv \mathbf{y} \end{array} \end{aligned}$$

$\hookrightarrow$ p. 110

Multilayer Perceptron at Arbitrary Depth

(2) Backpropagation (continued) [mlp two layers]

⋮ [derivation]

$$\Delta W^{\mathsf{h}_s} = \begin{cases} \eta \cdot \sum\limits_{D} \underbrace{\left[(\mathbf{c} - \mathbf{y}(\mathbf{x})) \odot \mathbf{y}(\mathbf{x}) \odot (1 - \mathbf{y}(\mathbf{x}))\right]}_{\boldsymbol{\delta}^{\mathsf{h}_d} \equiv \boldsymbol{\delta}^{\mathsf{o}}} \otimes \mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}) & \text{if } s = d \\ \eta \cdot \sum\limits_{D} \underbrace{\left[\left(\left(W^{\mathsf{h}_{s+1}}\right)^T \boldsymbol{\delta}^{\mathsf{h}_{s+1}}\right) \odot \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) \odot (1 - \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}))\right]_{1,\ldots,l_s}}_{\boldsymbol{\delta}^{\mathsf{h}_s}} \otimes \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x}) & \text{if } 1 < s < d \\ \eta \cdot \sum\limits_{D} \underbrace{\left[\left(\left(W^{\mathsf{h}_2}\right)^T \boldsymbol{\delta}^{\mathsf{h}_2}\right) \odot \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) \odot (1 - \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}))\right]_{1,\ldots,l_1}}_{\boldsymbol{\delta}^{\mathsf{h}_1}} \otimes \mathbf{x} & \text{if } s = 1 \end{cases}$$

where $l_s = \textit{no.\_rows}(W^{\mathsf{h}_s})$

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm

Algorithm:	$\text{IGD}_{\text{MLP}_d}$	Incremental Gradient Descent for the d-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}_1}, \ldots, W^{\text{h}_d}$	Weight matrices of the d layers. (= hypothesis)

FOR $s = 1$ TO d DO *initialize_random_weights*(W^{h_s}) ENDDO, $t = 0$

REPEAT

  $t = t + 1$

  FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO

  ENDDO

UNTIL(*convergence*$(D, \mathbf{y}(), t)$)

return$(W^{\text{h}_1}, \ldots, W^{\text{h}_d})$

[Python code]

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm (continued) [mlp two layers]

Algorithm: $\text{IGD}_{\text{MLP}_d}$ Incremental Gradient Descent for the d-layer MLP.

Input: D Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.

η Learning rate, a small positive constant.

Output: $W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$ Weight matrices of the d layers. (= hypothesis)

1. FOR $s=1$ TO d DO *initialize_random_weights*(W^{h_s}) ENDDO, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
5. $\quad\quad \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_1}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$

   $\quad\quad$ FOR $s=2$ TO $d{-}1$ DO $\mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_s}\mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO

   $\quad\quad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{h}_d}\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}))$
6.

7a.

7b.

8.

9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$

[Python code]

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm (continued) [mlp two layers]

Algorithm:	$\text{IGD}_{\text{MLP}_d}$	Incremental Gradient Descent for the d-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

1. FOR $s = 1$ TO d DO *initialize_random_weights*(W^{h_s}) ENDDO, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO
5. $\quad\quad \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_1}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$
$\quad\quad$ FOR $s = 2$ TO $d{-}1$ DO $\mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_s}\mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO
$\quad\quad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{h}_d}\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}))$
6. $\quad\quad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$

7a.

7b.

8.

9. $\quad$ ENDDO
10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$

[Python code]

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm (continued) [mlp two layers]

Algorithm:	$\text{IGD}_{\text{MLP}_d}$	Incremental Gradient Descent for the d-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

1. FOR $s = 1$ TO d DO *initialize_random_weights*(W^{h_s}) ENDDO, $t = 0$

2. **REPEAT**

3. $\quad t = t + 1$

4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO

5. $\qquad \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_1}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$

$\qquad$ FOR $s = 2$ TO $d{-}1$ DO $\mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_s}\mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO

$\qquad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{h}_d}\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}))$

6. $\qquad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$

7a. $\qquad \boldsymbol{\delta}^{\mathsf{h}_d} = \boldsymbol{\delta} \odot \mathbf{y}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}(\mathbf{x}))$ // backpropagation (Steps 7a+7b)

$\qquad$ FOR $s = d{-}1$ DOWNTO 1 DO $\boldsymbol{\delta}^{\mathsf{h}_s} = [((W^{\mathsf{h}_{s+1}})^T \boldsymbol{\delta}^{\mathsf{h}_{s+1}}) \odot \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}))]_{1,\ldots,l_s}$ ENDDO

7b. $\qquad \Delta W^{\mathsf{h}_1} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_1} \otimes \mathbf{x})$

$\qquad$ FOR $s = 2$ TO d DO $\Delta W^{\mathsf{h}_s} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_s} \otimes \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x}))$ ENDDO

8.

9. $\quad$ ENDDO

10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)

11. *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$

[Python code]

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm (continued) [mlp two layers]

Algorithm:	$\text{IGD}_{\text{MLP}_d}$	Incremental Gradient Descent for the d-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

1. FOR $s = 1$ TO d DO *initialize_random_weights*(W^{h_s}) ENDDO, $t = 0$

2. **REPEAT**

3. $\quad t = t + 1$

4. $\quad$ FOREACH $(\mathbf{x}, \mathbf{c}) \in D$ DO

5. $\qquad \mathbf{y}^{\mathsf{h}_1}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_1}\mathbf{x}) \end{pmatrix}$ // forward propagation; $\mathbf{x}$ is extended by $x_0 = 1$

$\qquad$ FOR $s = 2$ TO $d{-}1$ DO $\mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}_s}\mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})) \end{pmatrix}$ ENDDO

$\qquad \mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{h}_d}\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}))$

6. $\qquad \boldsymbol{\delta} = \mathbf{c} - \mathbf{y}(\mathbf{x})$

7a. $\qquad \boldsymbol{\delta}^{\mathsf{h}_d} = \boldsymbol{\delta} \odot \mathbf{y}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}(\mathbf{x}))$ // backpropagation (Steps 7a+7b)

$\qquad$ FOR $s = d{-}1$ DOWNTO 1 DO $\boldsymbol{\delta}^{\mathsf{h}_s} = [((W^{\mathsf{h}_{s+1}})^T \boldsymbol{\delta}^{\mathsf{h}_{s+1}}) \odot \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}))]_{1,\ldots,l_s}$ ENDDO

7b. $\qquad \Delta W^{\mathsf{h}_1} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_1} \otimes \mathbf{x})$

$\qquad$ FOR $s = 2$ TO d DO $\Delta W^{\mathsf{h}_s} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}_s} \otimes \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x}))$ ENDDO

8. $\qquad$ FOR $s = 1$ TO d DO $W^{\mathsf{h}_s} = W^{\mathsf{h}_s} + \Delta W^{\mathsf{h}_s}$ ENDDO

9. $\quad$ ENDDO

10. **UNTIL**(*convergence*$(D, \mathbf{y}(), t)$)

11. *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$

[Python code]

Multilayer Perceptron at Arbitrary Depth

The IGD Algorithm (continued) [mlp two layers]

Algorithm:	$\text{IGD}_{\text{MLP}_d}$	Incremental Gradient Descent for the d-layer MLP.
Input:	D	Multiset of examples $(\mathbf{x}, \mathbf{c})$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c} \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d}$	Weight matrices of the d layers. (= hypothesis)

1. `FOR` $s = 1$ `TO` d `DO` *initialize_random_weights*(W^{h_s}) `ENDDO`, $t = 0$
2. **`REPEAT`**
3. $\quad t = t + 1$
4. $\quad$`FOREACH` $(\mathbf{x}, \mathbf{c}) \in D$ `DO`
5. $\quad\quad$ `Model function evaluation.`
6. $\quad\quad$ `Calculation of residual vector.`

7a.

7b. $\quad\quad$ `Calculation of derivative of the loss.`

8. $\quad\quad$ `Parameter vector update` $\widehat{=}$ `one gradient step down.`
9. $\quad$`ENDDO`
10. **`UNTIL`**(*convergence*$(D, \mathbf{y}(), t)$)
11. *return*$(W^{\mathsf{h}_1}, \ldots, W^{\mathsf{h}_d})$

[Python code]

Remarks (derivation of $\nabla^{\mathsf{h}_s} L_2(\mathbf{w})$) :

- Partial derivative for a weight in a weight matrix W^{h_s}, $1 \leq s \leq d$:

$$
\begin{aligned}
\frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} L_2(\mathbf{w}) &= \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c})\in D} \sum_{u=1}^{k} (c_u - y_u(\mathbf{x}))^2 \\
&= \frac{1}{2} \cdot \sum_{D} \sum_{u=1}^{k} \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} (c_u - y_u(\mathbf{x}))^2 \\
&= -\sum_{D} \sum_{u=1}^{k} (c_u - y_u(\mathbf{x})) \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} y_u(\mathbf{x}) \\
&\overset{(1,2)}{=} -\sum_{D} \sum_{u=1}^{k} \underbrace{(c_u - y_u(\mathbf{x})) \cdot y_u(\mathbf{x}) \cdot (1 - y_u(\mathbf{x}))}_{\delta_u^{\mathsf{h}_d} \equiv \delta_u^{\mathsf{o}}} \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} W_{u*}^{\mathsf{h}_d}\, \mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}) \\
&\overset{(3)}{=} -\sum_{D} \sum_{u=1}^{k} \delta_u^{\mathsf{h}_d} \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} \sum_{v=0}^{l_{d-1}} w_{uv}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x})
\end{aligned}
$$

- Partial derivative for a weight in W^{h_d} (output layer), i.e., $s = d$:

$$
\frac{\partial}{\partial w_{ij}^{\mathsf{h}_d}} L_2(\mathbf{w}) = -\sum_{D} \sum_{u=1}^{k} \delta_u^{\mathsf{h}_d} \cdot \sum_{v=0}^{l_{d-1}} \frac{\partial}{\partial w_{ij}^{\mathsf{h}_d}} w_{uv}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x})
$$

// Only for the term where $u = i$ and $v = j$ the partial derivative is nonzero. See the illustration.

$$
= -\sum_{D} \delta_i^{\mathsf{h}_d} \cdot y_j^{\mathsf{h}_{d-1}}(\mathbf{x})
$$

Remarks (derivation of $\nabla^{\mathsf{h}_s} L_2(\mathbf{w})$) : (continued)

- Partial derivative for a weight in a weight matrix W^{h_s}, $s \leq d{-}1$:

$$\frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} L_2(\mathbf{w}) = -\sum_D \sum_{u=1}^{k} \delta_u^{\mathsf{h}_d} \cdot \sum_{v=0}^{l_{d-1}} \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} w_{uv}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x})$$

// Every component of $\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})$ except $y_0^{\mathsf{h}_{d-1}}$ depends on $w_{ij}^{\mathsf{h}_s}$. See the illustration.

$$\overset{(1,2)}{=} -\sum_D \sum_{u=1}^{k} \delta_u^{\mathsf{h}_d} \cdot \sum_{v=1}^{l_{d-1}} w_{uv}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x}) \cdot (1 - y_v^{\mathsf{h}_{d-1}}(\mathbf{x})) \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} W_{v*}^{\mathsf{h}_{d-1}}\, \mathbf{y}^{\mathsf{h}_{d-2}}(\mathbf{x})$$

$$\overset{(4)}{=} -\sum_D \sum_{v=1}^{l_{d-1}} \sum_{u=1}^{k} \delta_u^{\mathsf{h}_d} \cdot w_{uv}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x}) \cdot (1 - y_v^{\mathsf{h}_{d-1}}(\mathbf{x})) \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} W_{v*}^{\mathsf{h}_{d-1}}\, \mathbf{y}^{\mathsf{h}_{d-2}}(\mathbf{x})$$

$$\overset{(5)}{=} -\sum_D \sum_{v=1}^{l_{d-1}} \underbrace{(W_{*v}^{\mathsf{h}_d})^T \boldsymbol{\delta}^{\mathsf{h}_d} \cdot y_v^{\mathsf{h}_{d-1}}(\mathbf{x}) \cdot (1 - y_v^{\mathsf{h}_{d-1}}(\mathbf{x}))}_{\delta_v^{\mathsf{h}_{d-1}}} \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} W_{v*}^{\mathsf{h}_{d-1}}\, \mathbf{y}^{\mathsf{h}_{d-2}}(\mathbf{x})$$

$$\overset{(3)}{=} -\sum_D \sum_{v=1}^{l_{d-1}} \delta_v^{\mathsf{h}_{d-1}} \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} \sum_{w=0}^{l_{d-2}} w_{vw}^{\mathsf{h}_{d-1}} \cdot y_w^{\mathsf{h}_{d-2}}(\mathbf{x})$$

- Partial derivative for a weight in $W^{\mathsf{h}_{d-1}}$ (next to output layer), i.e., $s = d{-}1$:

$$\frac{\partial}{\partial w_{ij}^{\mathsf{h}_{d-1}}} L_2(\mathbf{w}) = -\sum_D \sum_{v=1}^{l_{d-1}} \delta_v^{\mathsf{h}_{d-1}} \sum_{w=0}^{l_{d-2}} \frac{\partial}{\partial w_{ij}^{\mathsf{h}_{d-1}}} w_{vw}^{\mathsf{h}_{d-1}} \cdot y_w^{\mathsf{h}_{d-2}}(\mathbf{x})$$

// Only for the term where $v = i$ and $w = j$ the partial derivative is nonzero.

$$= -\sum_D \delta_i^{\mathsf{h}_{d-1}} \cdot y_j^{\mathsf{h}_{d-2}}(\mathbf{x})$$

Remarks (derivation of $\nabla^{\mathsf{h}_s} L_2(\mathbf{w})$) : (continued)

- Instead of writing out the recursion further, i.e., considering a weight matrix W^{h_s}, $s \leq d-2$, we substitute s for $d-1$ (similarly: $s+1$ for d) to derive the general backpropagation rule:

$$\frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} L_2(\mathbf{w}) = -\sum_D \delta_i^{\mathsf{h}_s} \cdot y_j^{\mathsf{h}_{s-1}}(\mathbf{x}) \quad // \ \delta_i^{\mathsf{h}_s} \text{ is expanded based on the definition of } \delta_v^{\mathsf{h}_{d-1}}.$$

$$= -\sum_D \underbrace{(W_{*i}^{\mathsf{h}_{s+1}})^T \boldsymbol{\delta}^{\mathsf{h}_{s+1}} \cdot y_i^{\mathsf{h}_s}(\mathbf{x}) \cdot (1 - y_i^{\mathsf{h}_s}(\mathbf{x}))}_{\delta_i^{\mathsf{h}_s}} \cdot y_j^{\mathsf{h}_{s-1}}(\mathbf{x})$$

- Plugging the result for $\frac{\partial}{\partial w_{ij}^{\mathsf{h}_s}} L_2(\mathbf{w})$ into $-\eta \cdot [\cdots]$ yields the update formula for ΔW^{h_s}. In detail:

  – For updating the output matrix, $W^{\mathsf{h}_d} \equiv W^{\mathsf{o}}$, we compute

$$\boldsymbol{\delta}^{\mathsf{h}_d} = (\mathbf{c} - \mathbf{y}(\mathbf{x})) \odot \mathbf{y}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}(\mathbf{x}))$$

  – For updating a matrix W^{h_s}, $1 \leq s < d$, we compute

$$\boldsymbol{\delta}^{\mathsf{h}_s} = [((W^{\mathsf{h}_{s+1}})^T \boldsymbol{\delta}^{\mathsf{h}_{s+1}}) \odot \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}) \odot (\mathbf{1} - \mathbf{y}^{\mathsf{h}_s}(\mathbf{x}))]_{1,\ldots,l_s},$$

where
$W^{\mathsf{h}_{s+1}} \in \mathbf{R}^{l_{s+1} \times (l_s+1)}$,
$\boldsymbol{\delta}^{\mathsf{h}_{s+1}} \in \mathbf{R}^{l_{s+1}}$,
$\mathbf{y}^{\mathsf{h}_s} \in \mathbf{R}^{l_s+1}$,
and $\mathbf{y}^{\mathsf{h}_0}(\mathbf{x}) \equiv \mathbf{x}$.

Remarks (derivation of $\nabla^{\mathsf{h}_s} L_2(\mathbf{w})$) : (continued)

- Hints:

(1) $y_u(\mathbf{x}) = \left[\boldsymbol{\sigma}\left(W^{\mathsf{h}_d}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})\right)\right]_u = \sigma\left(W^{\mathsf{h}_d}_{u*}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})\right)$

(2) Chain rule with $\frac{d}{dz}\sigma(z) = \sigma(z)\cdot(1-\sigma(z))$, where $\sigma(z) := y_u(\mathbf{x})$ and $z = W^{\mathsf{h}_d}_{u*}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})$:

$$\frac{\partial}{\partial w^{\mathsf{h}_s}_{ij}} y_u(\mathbf{x}) \equiv \frac{\partial}{\partial w^{\mathsf{h}_s}_{ij}}\left(\sigma\left(W^{\mathsf{h}_d}_{u*}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})\right)\right) \equiv \frac{\partial}{\partial w^{\mathsf{h}_s}_{ij}}\left(\sigma\left(z\right)\right) = y_u(\mathbf{x})\cdot(1-y_u(\mathbf{x}))\cdot\frac{\partial}{\partial w^{\mathsf{h}_s}_{ij}}\left(W^{\mathsf{h}_d}_{u*}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})\right)$$

Note that in the partial derivative expression the symbol $\mathbf{x}$ is a constant, while $w^{\mathsf{h}_s}_{ij}$ is the variable whose effect on the change of the loss L_2 (at input $\mathbf{x}$) is computed.

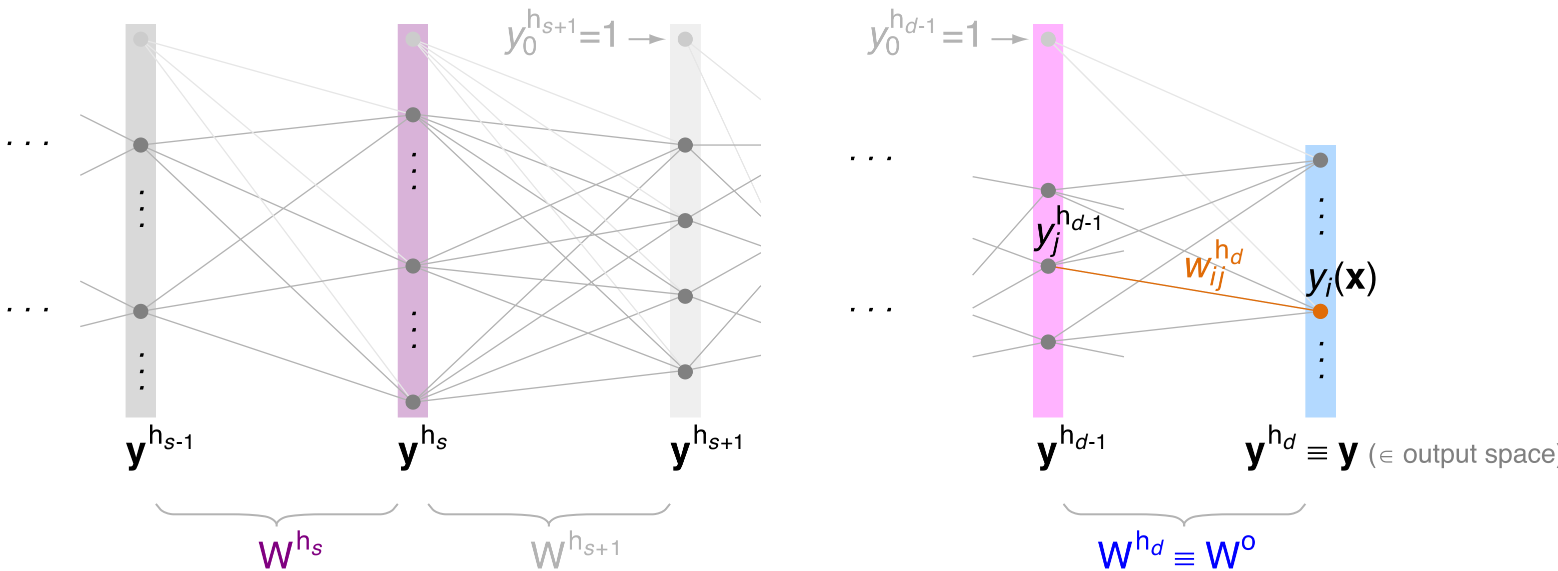
(3) $W^{\mathsf{h}_d}_{u*}\,\mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x}) = w^{\mathsf{h}_d}_{u0}\cdot y^{\mathsf{h}_{d-1}}_0(\mathbf{x}) + \ldots + w^{\mathsf{h}_d}_{uj}\cdot y^{\mathsf{h}_{d-1}}_j(\mathbf{x}) + \ldots + w^{\mathsf{h}_d}_{ul_{d-1}}\cdot y^{\mathsf{h}_{d-1}}_{l_{d-1}}(\mathbf{x})$,
where $l_{d-1} =$ *no._rows*$(W^{\mathsf{h}_{d-1}})$.

(4) Rearrange sums to reflect the nested dependencies that develop naturally from the backpropagation. We now can define $\delta^{\mathsf{h}_{d-1}}_v$ in layer $d-1$ as a function of $\boldsymbol{\delta}^{\mathsf{h}_d}$ (layer d).

(5) $\sum_{u=1}^{k} \delta^{\mathsf{h}_d}_u \cdot w^{\mathsf{h}_d}_{uv} = (W^{\mathsf{h}_d}_{*v})^T \boldsymbol{\delta}^{\mathsf{h}_d}$ (scalar product).

Remarks (derivation of $\nabla^{\mathsf{h}_s} L_2(\mathbf{w})$) : (continued)

- $\mathbf{y}(\mathbf{x})$ as a function of some $w_{ij}^{\mathsf{h}_s}$ in the output layer W^o and some middle layer W^{h_s}. To calculate the partial derivative of $y_u(\mathbf{x})$ with respect to $w_{ij}^{\mathsf{h}_s}$, determine those terms in $y_u(\mathbf{x})$ that depend on $w_{ij}^{\mathsf{h}_s}$ (shown orange). All other terms are in the role of constants.



$$y_u(\mathbf{x}) = \left[\boldsymbol{\sigma}\left(W^{\mathsf{h}_d}\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(\ldots\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(W^{\mathsf{h}_{s+1}}\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(W^{\mathsf{h}_s}\ \mathbf{y}^{\mathsf{h}_{s-1}}(\mathbf{x})\right)\end{pmatrix}\right)\end{pmatrix}\ldots\right)\end{pmatrix}\right)\right]_u$$

$\boldsymbol{\sigma}(\ldots) \sim \mathbf{y}^{\mathsf{h}_d}(\mathbf{x}) \equiv \mathbf{y}(\mathbf{x})$

$((\ldots)) \sim \mathbf{y}^{\mathsf{h}_{d-1}}(\mathbf{x})$

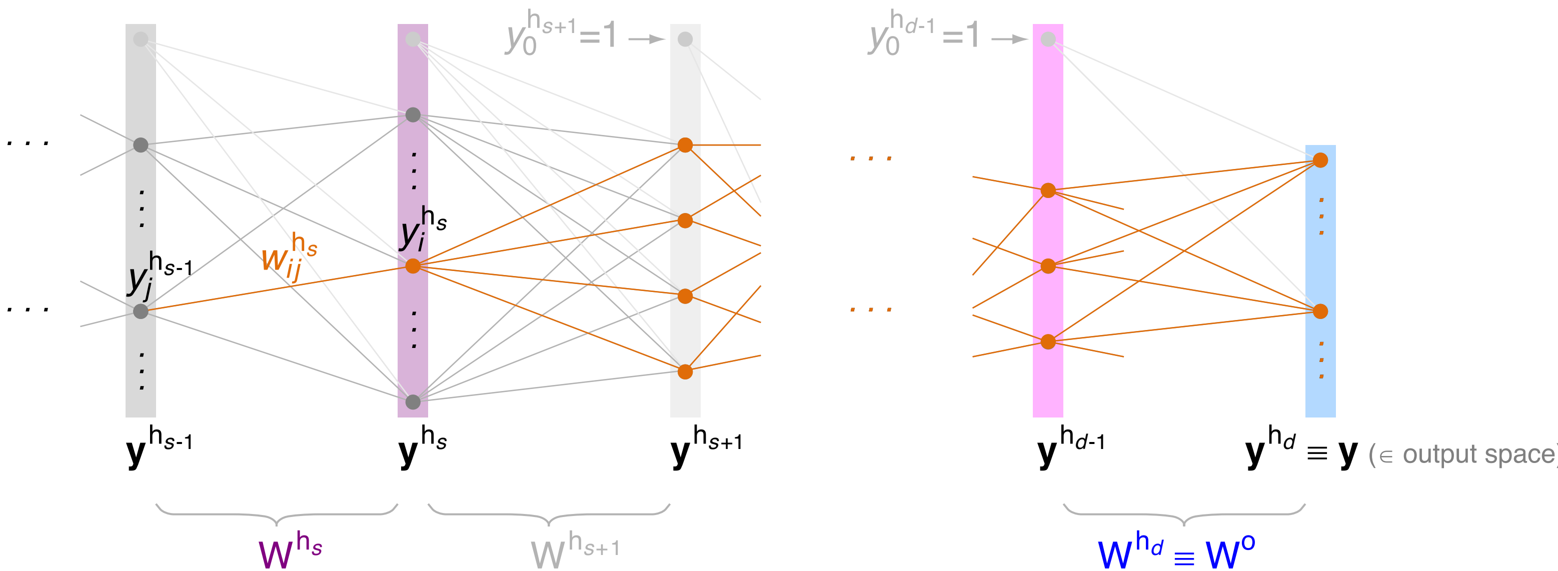
$((\ldots)) \sim \mathbf{y}^{\mathsf{h}_{s+1}}(\mathbf{x})$

$((\ldots)) \sim \mathbf{y}^{\mathsf{h}_s}(\mathbf{x})$

- Compare the above illustration to the multilayer perceptron network architecture.

Remarks (derivation of $\nabla^{\text{h}_s} L_2(\mathbf{w})$) : (continued)

- $\mathbf{y}(\mathbf{x})$ as a function of some $w_{ij}^{\text{h}_s}$ in the output layer W^o and some middle layer W^{h_s}. To calculate the partial derivative of $y_u(\mathbf{x})$ with respect to $w_{ij}^{\text{h}_s}$, determine those terms in $y_u(\mathbf{x})$ that depend on $w_{ij}^{\text{h}_s}$ (shown orange). All other terms are in the role of constants.



$$y_u(\mathbf{x}) = \left[\boldsymbol{\sigma}\left(W^{\text{h}_d}\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(\ldots\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(W^{\text{h}_{s+1}}\begin{pmatrix}1\\ \boldsymbol{\sigma}\left(W^{\text{h}_s}\ \mathbf{y}^{\text{h}_{s-1}}(\mathbf{x})\right)\end{pmatrix}\right)\end{pmatrix}\ldots\right)\end{pmatrix}\right)\right]_u$$

$\boldsymbol{\sigma}(\ldots) \sim \mathbf{y}^{\text{h}_d}(\mathbf{x}) \equiv \mathbf{y}(\mathbf{x})$

$((\ldots)) \sim \mathbf{y}^{\text{h}_{d-1}}(\mathbf{x})$

$((\ldots)) \sim \mathbf{y}^{\text{h}_{s+1}}(\mathbf{x})$

$((\ldots)) \sim \mathbf{y}^{\text{h}_s}(\mathbf{x})$

- Compare the above illustration to the multilayer perceptron network architecture.

Remarks (derivation of $\nabla^{\mathsf{o}} L_2(\mathbf{w})$ and $\nabla^{\mathsf{h}} L_2(\mathbf{w})$ for MLP at depth one) :

- $\nabla^{\mathsf{o}} L_2(\mathbf{w}) \equiv \nabla^{\mathsf{h}_d} L_2(\mathbf{w})$, and hence $\boldsymbol{\delta}^{\mathsf{o}} \equiv \boldsymbol{\delta}^{\mathsf{h}_d}$.
- $\nabla^{\mathsf{h}} L_2(\mathbf{w})$ is a special case of the s-layer case, and we obtain $\boldsymbol{\delta}^{\mathsf{h}}$ from $\boldsymbol{\delta}^{\mathsf{h}_s}$ by applying the following identities:

  $W^{\mathsf{h}_{s+1}} = W^{\mathsf{o}}$, $\boldsymbol{\delta}^{\mathsf{h}_{s+1}} = \boldsymbol{\delta}^{\mathsf{h}_d} = \boldsymbol{\delta}^{\mathsf{o}}$, $\mathbf{y}^{\mathsf{h}_s} = \mathbf{y}^{\mathsf{h}}$, and $l_s = l$.