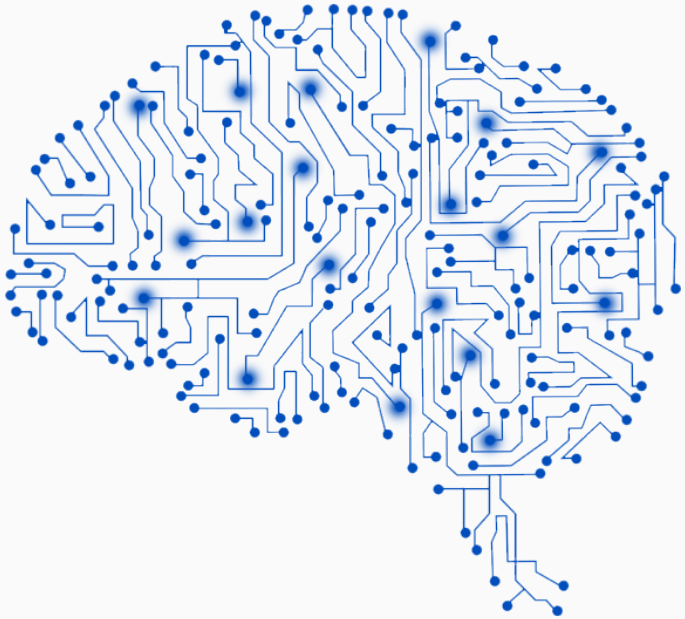
Master 1 Statistique & Data Science, Ingénierie Mathématique

Apprentissage pour l'image
Machine learning for image processing

Course II – Introduction to Artificial Neural Networks: Backpropagation

Emile Pierret

Lundi 31 mars 2025



Objectifs

À la fin du cours :

- Comprendre ce qu'est un CNN (Réseau de Neurones Convolutifs)
- Implémenter l'entraînement d'un CNN pour la classification avec Pytorch

Pour cette session :

- Backpropagation pour calculer les gradients dans les Réseaux de Neurones

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ?

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\frac{\partial z}{\partial x}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial w}\frac{\partial w}{\partial x}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\begin{aligned} \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\ &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \end{aligned}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\begin{aligned} \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\ &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \\ &= 1 \times \end{aligned}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\begin{aligned} \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\ &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \\ &= 1 \times \frac{1}{v} \times \end{aligned}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\begin{aligned} \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\ &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \\ &= 1 \times \frac{1}{v} \times y \end{aligned}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

$$\begin{aligned}
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\
&= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \\
&= 1 \times \frac{1}{v} \times y \\
&= \frac{y}{v}
\end{aligned}$$

Un exemple simple

Considérons la fonction $f : (x, y) \in \mathbb{R}^2 \mapsto \log(xy)$. Comment différencier f étape par étape par rapport à x ? Nous pouvons décomposer f ainsi :

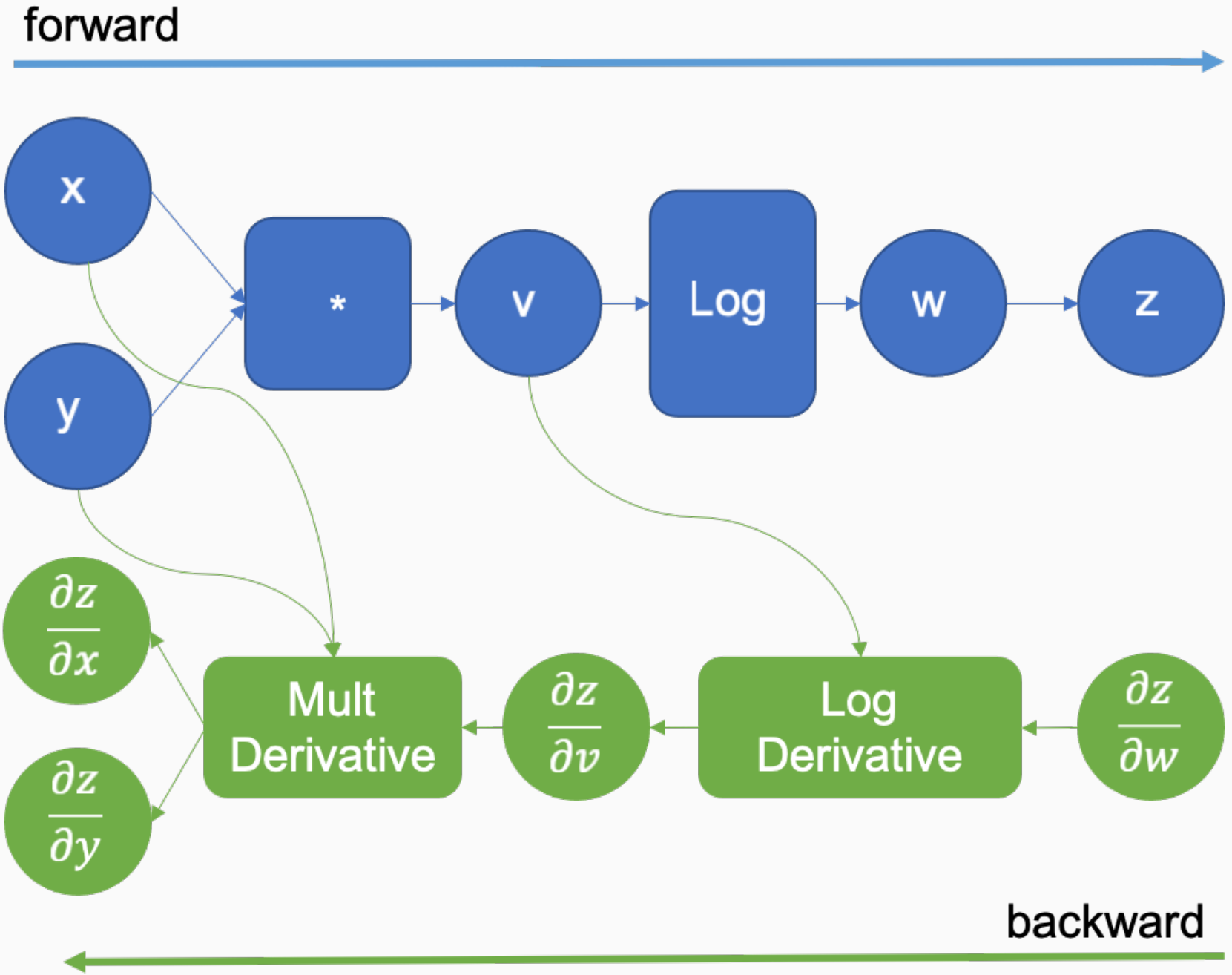
1. $v = xy$
2. $w = \log(v) = \log(xy)$
3. $z = w$

Rappelons que pour $g, h : \mathbb{R} \to \mathbb{R}$,

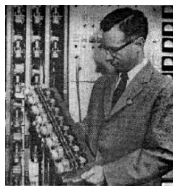
$$(g \circ f)'(x) = g'(f(x))f'(x)$$

Par conséquent,

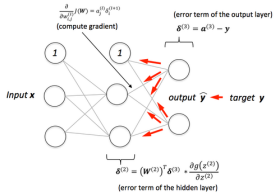
$$\begin{aligned}
\frac{\partial z}{\partial x} &= \frac{\partial z}{\partial w}\frac{\partial w}{\partial x} \\
&= \frac{\partial z}{\partial w}\frac{\partial w}{\partial v}\frac{\partial v}{\partial x} \\
&= 1 \times \frac{1}{v} \times y \\
&= \frac{y}{v} \\
&= \frac{1}{x}
\end{aligned}$$


forward
x
y
*
v
Log
w
z
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
Mult Derivative
$\frac{\partial z}{\partial v}$
Log Derivative
$\frac{\partial z}{\partial w}$
backward

Timeline of (deep) learning



1958 Perceptron



1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

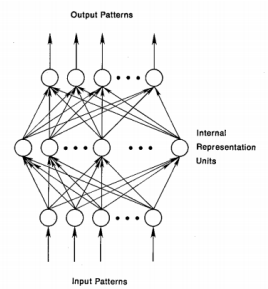
Google Brain Project on 16k Cores

2012

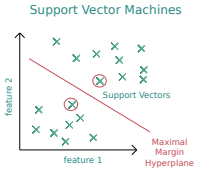
awkward silence (AI winter)

1969 Perceptrons book

~1980 Multilayer network



1995 SVM reigns

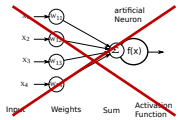


2006 Restricted Boltzmann Machine

2012 AlexNet wins ImageNet

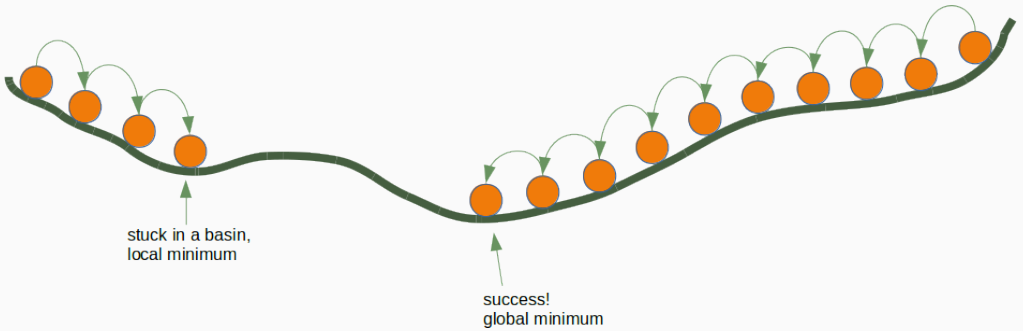
IMAGENET

Perceptron criticized

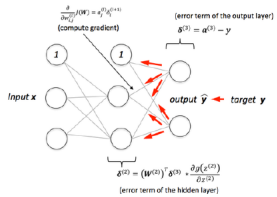


(Source: Lucas Masuch & Vincent Lepetit)

Backpropagation



Learning with backpropagation



Backpropagation

1958 Perceptron

1974

1969 Perceptrons book

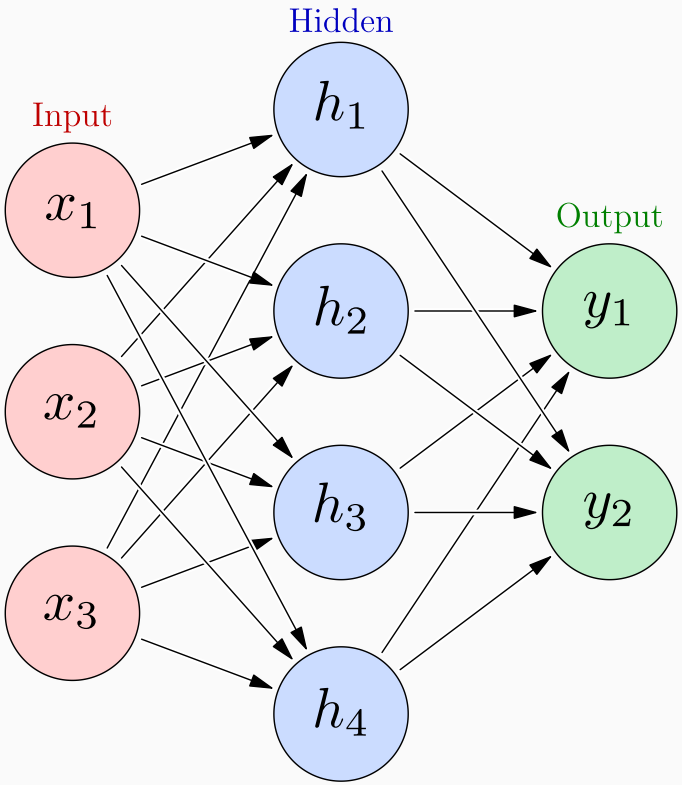
~1980 Multilayer network

1989 Universal Approximation Theorem

Perceptron criticized

(Source: Lucas Masuch & Vincent Lepetit)

Artificial neural network / Multilayer perceptron / NeuralNet



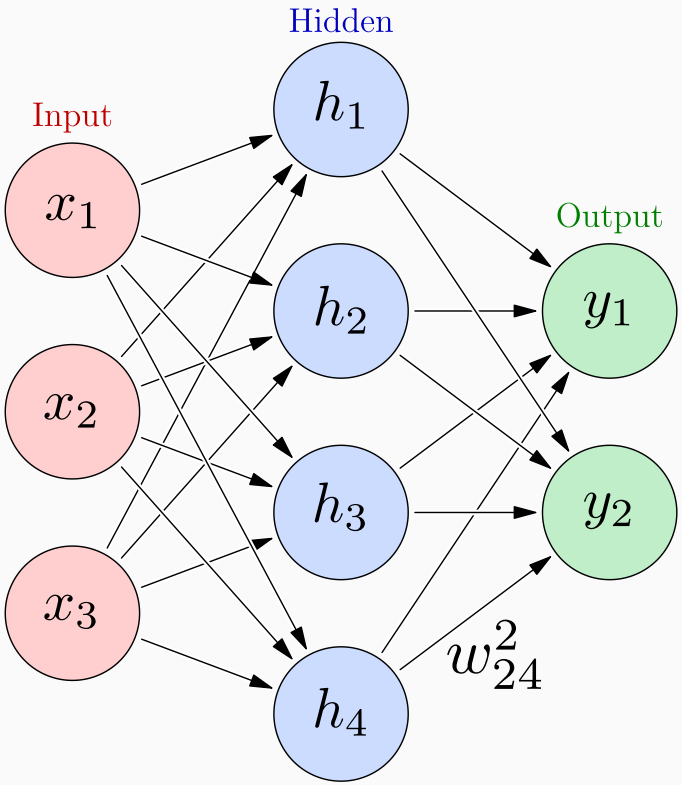
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

$$y_1 = g_2\left(w^2_{11}h_1 + w^2_{12}h_2 + w^2_{13}h_3 + w^2_{14}h_4 + b^2_1\right)$$
$$y_2 = g_2\left(w^2_{21}h_1 + w^2_{22}h_2 + w^2_{23}h_3 + w^2_{24}h_4 + b^2_2\right)$$

w^k_{ij} synaptic weight between previous node j and next node i at layer k.

g_k are any activation function applied to each coefficient of its input vector.

Artificial neural network / Multilayer perceptron / NeuralNet



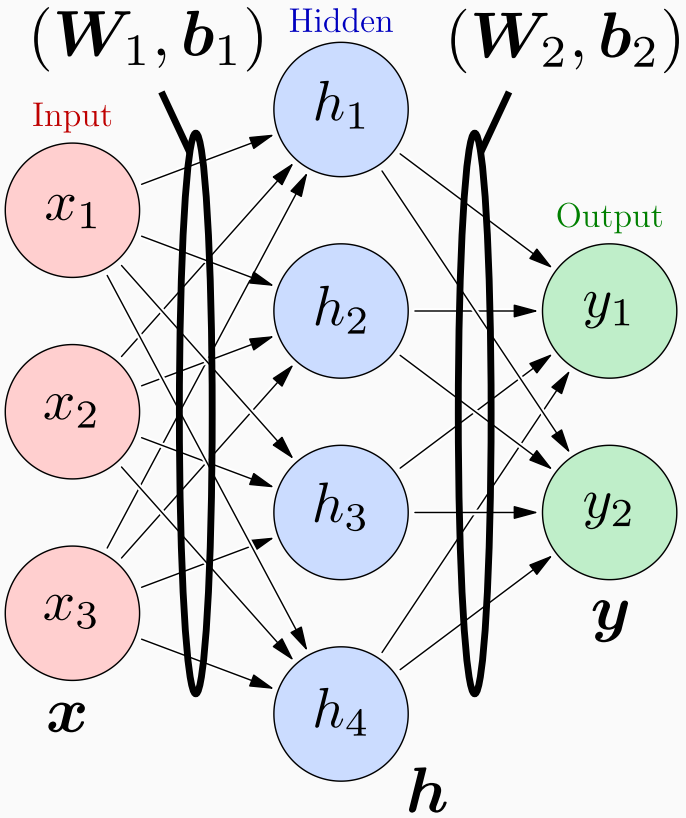
$$h_1 = g_1\left(w^1_{11}x_1 + w^1_{12}x_2 + w^1_{13}x_3 + b^1_1\right)$$
$$h_2 = g_1\left(w^1_{21}x_1 + w^1_{22}x_2 + w^1_{23}x_3 + b^1_2\right)$$
$$h_3 = g_1\left(w^1_{31}x_1 + w^1_{32}x_2 + w^1_{33}x_3 + b^1_3\right)$$
$$h_4 = g_1\left(w^1_{41}x_1 + w^1_{42}x_2 + w^1_{43}x_3 + b^1_4\right)$$

$$y_1 = g_2\left(w^2_{11}h_1 + w^2_{12}h_2 + w^2_{13}h_3 + w^2_{14}h_4 + b^2_1\right)$$
$$y_2 = g_2\left(w^2_{21}h_1 + w^2_{22}h_2 + w^2_{23}h_3 + w^2_{24}h_4 + b^2_2\right)$$

w^k_{ij} synaptic weight between previous node j and next node i at layer k.

g_k are any activation function applied to each coefficient of its input vector.

Artificial neural network / Multilayer perceptron / NeuralNet



$$h_1 = g_1\left(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1\right)$$
$$h_2 = g_1\left(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1\right)$$
$$h_3 = g_1\left(w_{31}^1 x_1 + w_{32}^1 x_2 + w_{33}^1 x_3 + b_3^1\right)$$
$$h_4 = g_1\left(w_{41}^1 x_1 + w_{42}^1 x_2 + w_{43}^1 x_3 + b_4^1\right)$$

$$\boldsymbol{h} = g_1\left(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1\right)$$

$$y_1 = g_2\left(w_{11}^2 h_1 + w_{12}^2 h_2 + w_{13}^2 h_3 + w_{14}^2 h_4 + b_1^2\right)$$
$$y_2 = g_2\left(w_{21}^2 h_1 + w_{22}^2 h_2 + w_{23}^2 h_3 + w_{24}^2 h_4 + b_2^2\right)$$

$$\boldsymbol{y} = g_2\left(\boldsymbol{W}_2 \boldsymbol{h} + \boldsymbol{b}_2\right)$$

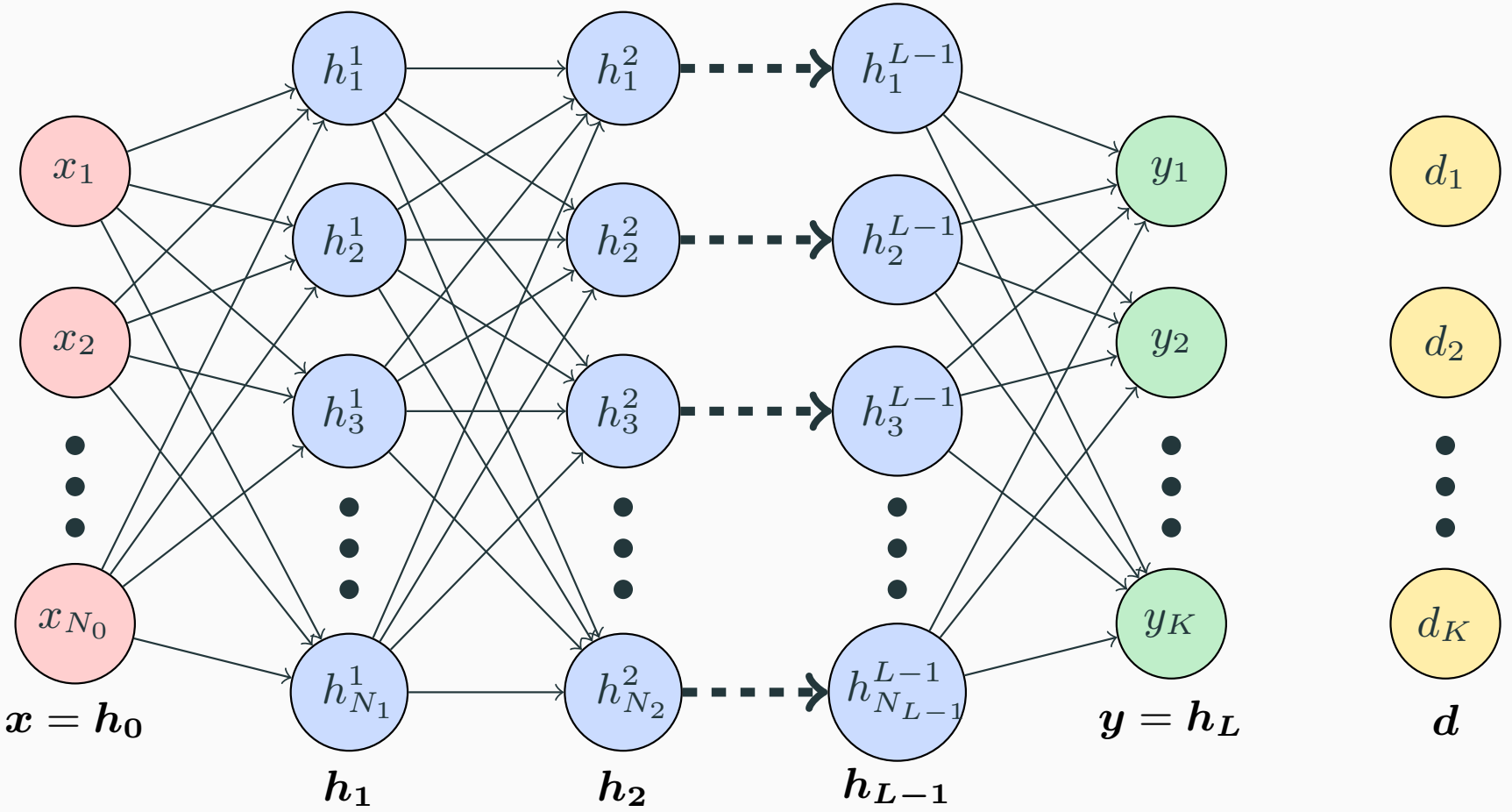
w_{ij}^k synaptic weight between previous node j and next node i at layer k.

g_k are any activation function applied to each coefficient of its input vector.

The matrices $\boldsymbol{W_k}$ and biases $\boldsymbol{b}_k$ are learned from labeled training data.

Feedforward Artificial Neural Network

Recall the feedforward structure

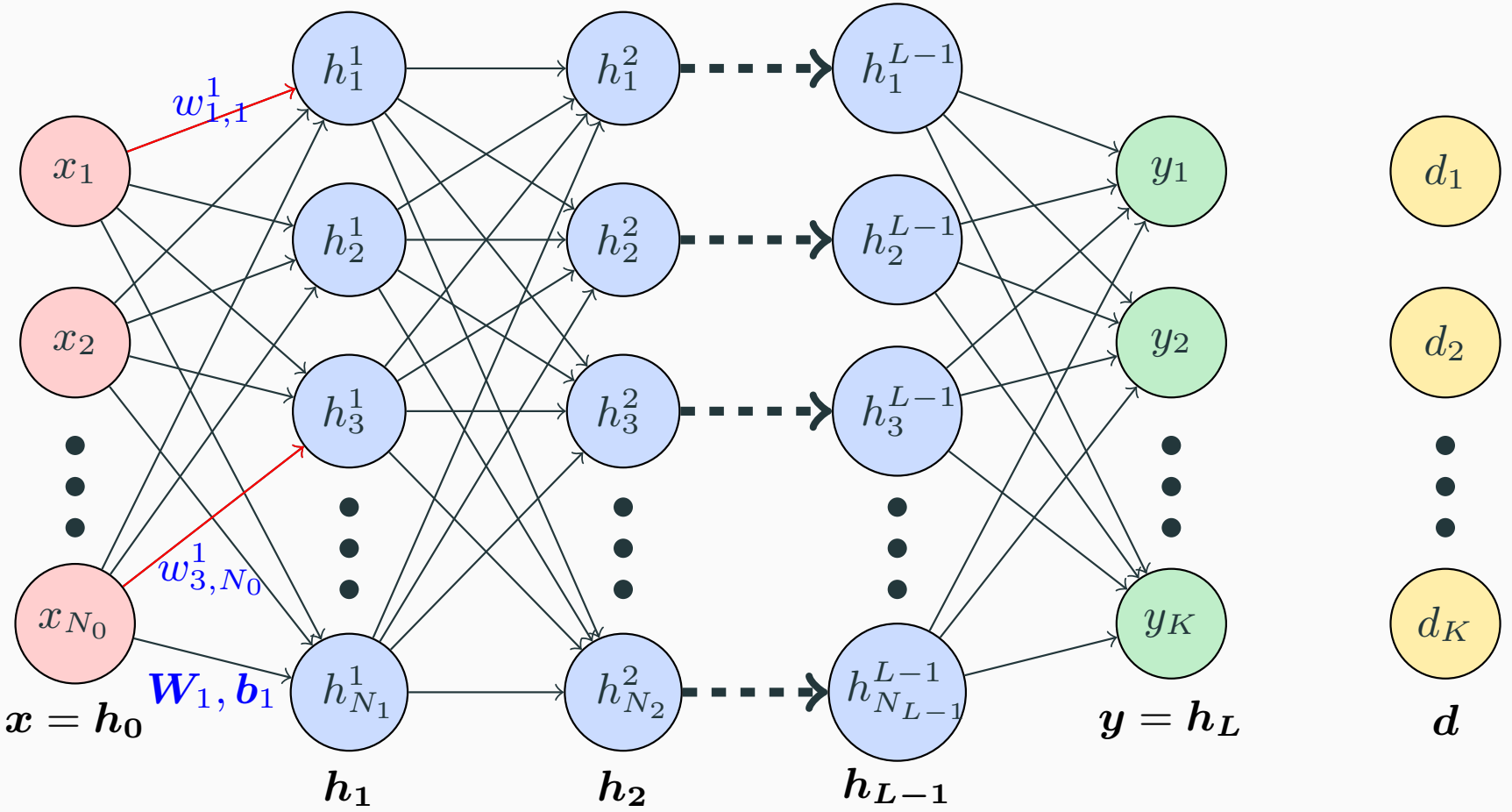


Loss: $L(\boldsymbol{y}; \boldsymbol{d})$

Input Layer | Hidden Layers | Output Layer | Label

Feedforward Artificial Neural Network

Recall the feedforward structure

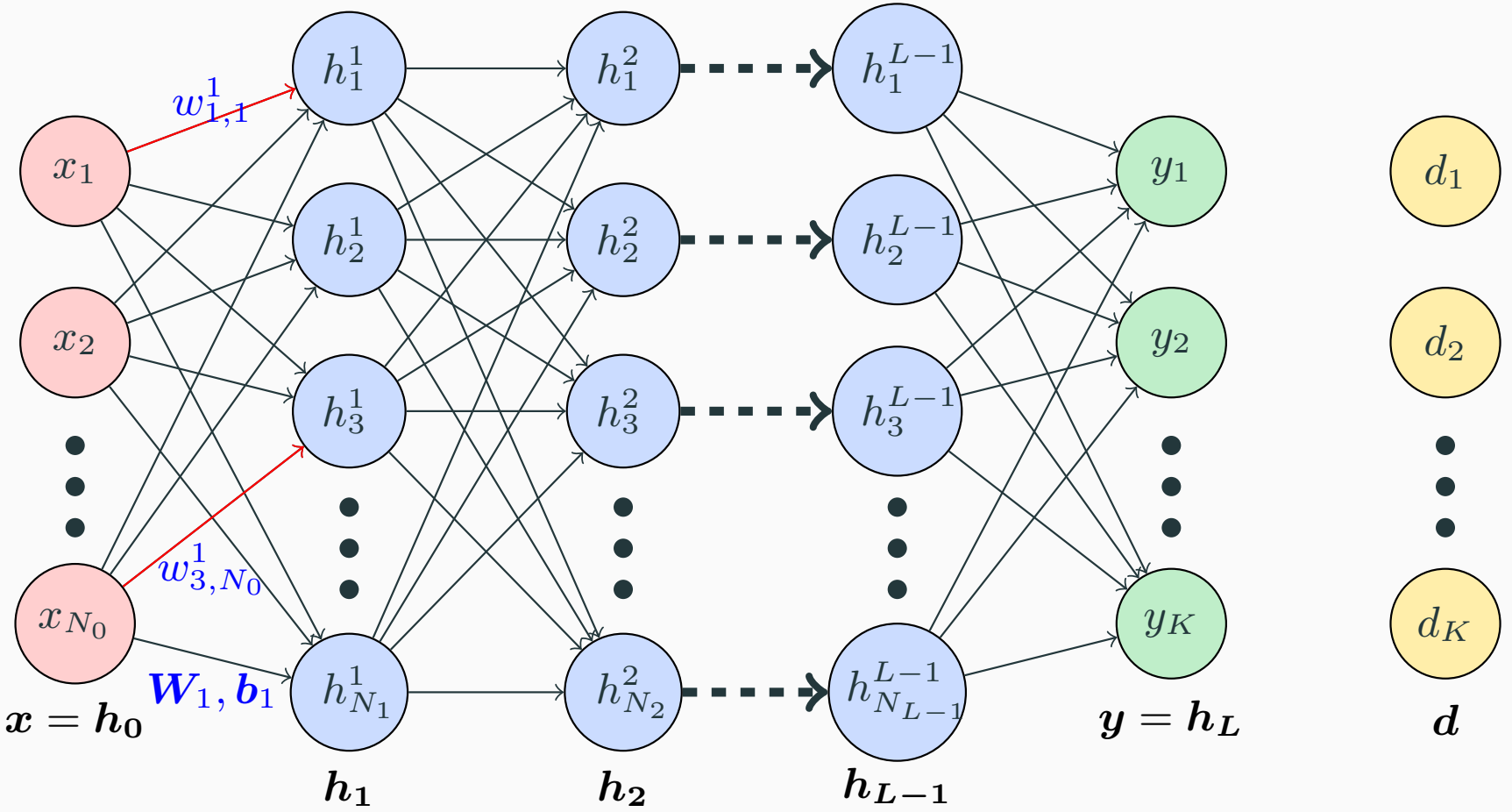


Loss: $L(\boldsymbol{y}; \boldsymbol{d})$

Input Layer | Hidden Layers | Output Layer | Label

Feedforward Artificial Neural Network

Recall the feedforward structure



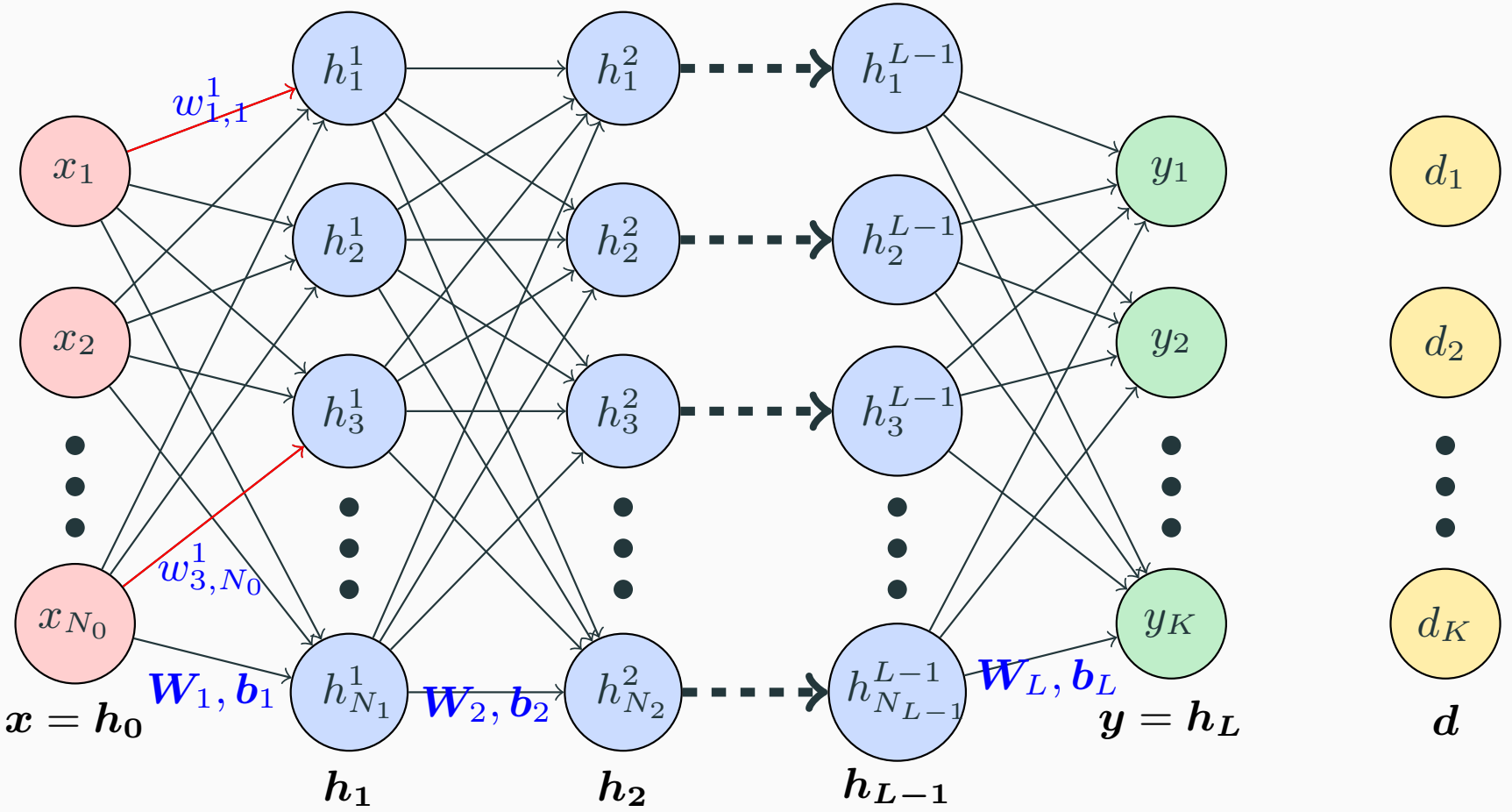
$= g_1(\boldsymbol{W_1 h_0} + \boldsymbol{b_1})$

Loss: $L(\boldsymbol{y}; \boldsymbol{d})$

Input Layer | Hidden Layers | Output Layer | Label

Feedforward Artificial Neural Network

Recall the feedforward structure



$$= g_1\left(\boldsymbol{W}_1 \boldsymbol{h_0} + \boldsymbol{b}_1\right)$$

Loss: $L(\boldsymbol{y}; \boldsymbol{d})$

Input Layer | Hidden Layers | Output Layer | Label

ANN – Optimization

- Les hyperparamètres du réseau de neurones sont

  le nombre de couches, la taille des couches, les fonctions d'activation

ANN – Optimization

- Les hyperparamètres du réseau de neurones sont

  le nombre de couches, la taille des couches, les fonctions d'activation

  N'évoluent pas au cours de l'entraînement !

ANN – Optimization

- Les hyperparamètres du réseau de neurones sont

  le nombre de couches, la taille des couches, les fonctions d'activation

  N'évoluent pas au cours de l'entraînement !

- Les paramètres du réseau de neurones sont

$$\boldsymbol{W} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

ANN – Optimization

- Les hyperparamètres du réseau de neurones sont

  le nombre de couches, la taille des couches, les fonctions d'activation

  N'évoluent pas au cours de l'entraînement !

- Les paramètres du réseau de neurones sont

$$\boldsymbol{W} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

- Entraîner le réseau = minimiser la fonction de perte d'entraînement $E(\boldsymbol{W})$

$$\textbf{Objectif:} \quad \min_{\boldsymbol{W}} E(\boldsymbol{W}) \quad \text{où} \quad E(W) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- **Solution:** pas de solution sous forme analytique

ANN – Optimization

- Les hyperparamètres du réseau de neurones sont

  le nombre de couches, la taille des couches, les fonctions d'activation

  N'évoluent pas au cours de l'entraînement !

- Les paramètres du réseau de neurones sont

$$\boldsymbol{W} = (\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \ldots, \boldsymbol{W}_L, \boldsymbol{b}_L)$$

- Entraîner le réseau = minimiser la fonction de perte d'entraînement $E(\boldsymbol{W})$

$$\textbf{Objectif:} \quad \min_{\boldsymbol{W}} E(\boldsymbol{W}) \quad \text{où} \quad E(W) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- **Solution:** pas de solution sous forme analytique $\Rightarrow$ utilisation de la descente de gradient stochastique (SGD).

$$\Rightarrow \nabla E(\boldsymbol{W}) = \begin{pmatrix} \nabla_{\boldsymbol{W}_1} E(\boldsymbol{W}) & \nabla_{\boldsymbol{b}_1} E(\boldsymbol{W}) & \ldots & \nabla_{\boldsymbol{W}_L} E(\boldsymbol{W}) & \nabla_{\boldsymbol{b}_L} E(\boldsymbol{W}) \end{pmatrix}$$

Minimisation de la perte d'entraînement

Pour les réseaux de neurones multicouches, $\boldsymbol{W} \mapsto E(\boldsymbol{W})$ est non convexe
$\Rightarrow$ Aucune garantie de convergence.
Même si la convergence a lieu, la solution dépend de l'initialisation et du pas d'apprentissage γ.

Néanmoins, de très bons minima ou points selles sont atteints en pratique grâce à

$$\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^{t} - \gamma \nabla E(\boldsymbol{W}^{t}), \quad \gamma > 0$$

La descente de gradient peut être exprimée coordonnée par coordonnée comme suit :

$$w_{i,j}^{k,t+1} \leftarrow w_{i,j}^{k,t} - \gamma \frac{\partial E(\boldsymbol{W}^{t})}{\partial w_{i,j}^{k}}$$

pour tous les poids $w_{i,j}^{k}$ reliant un nœud j à un nœud i dans la couche suivante k.

Optimisation

Fonctions de perte : Les fonctions de perte classiques sont :

Optimisation

Fonctions de perte : Les fonctions de perte classiques sont :

Pour divers applications : $\boldsymbol{d}^i \in \mathbb{R}^K$

- Erreur quadratique

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \frac{1}{2} \|\boldsymbol{y}^i - \boldsymbol{d}^i\|_2^2$$

Optimisation

Fonctions de perte : Les fonctions de perte classiques sont :

Pour divers applications : $\boldsymbol{d}^i \in \mathbb{R}^K$

- Erreur quadratique

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \frac{1}{2} \|\boldsymbol{y}^i - \boldsymbol{d}^i\|_2^2$$

Pour la classification multi-classes : En notant $d^i \in \{1, \ldots, K\}$ les labels,

- Cross-entropy avec softmax comme dernière couche

$$E(\boldsymbol{W}) = - \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \sum_{k=1}^{K} d_k^i \log y_k^i$$

$$\text{with} \quad \boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{W}) = \text{softmax}(\boldsymbol{a}^i) \in (0,1)^K.$$

Optimisation

- Les fonctions de perte sont de la forme

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Par linéarité,

$$\nabla E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \nabla L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Ici, la sortie du réseau de neurones $\boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{W})$ est une fonction des données d'entrée $\boldsymbol{x}^i$ et des poids du réseau $\boldsymbol{W}$.
- Nous connaissons le gradient de $L(\boldsymbol{y}^i; \boldsymbol{d}^i)$ par rapport à la variable $\boldsymbol{y}$
  - Régression / Erreur quadratique :

$$L(\boldsymbol{y}; \boldsymbol{d}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{d}\|_2^2 \quad \Rightarrow \quad \nabla_{\boldsymbol{y}} L(\boldsymbol{y}; \boldsymbol{d}) = \boldsymbol{y} - \boldsymbol{d}$$

  - Classification multi-classes / Entropie croisée :

$$L(\boldsymbol{y}; \boldsymbol{d}) = -y_d + \log\left(\sum_{k=1}^{K} \exp(y_k)\right) \Rightarrow (\nabla_{\boldsymbol{y}} L(\boldsymbol{y}; \boldsymbol{d}))_\ell = \mathrm{softmax}(\boldsymbol{y})_\ell - \delta_{\ell,d}.$$

Optimisation

- Les fonctions de perte sont de la forme

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

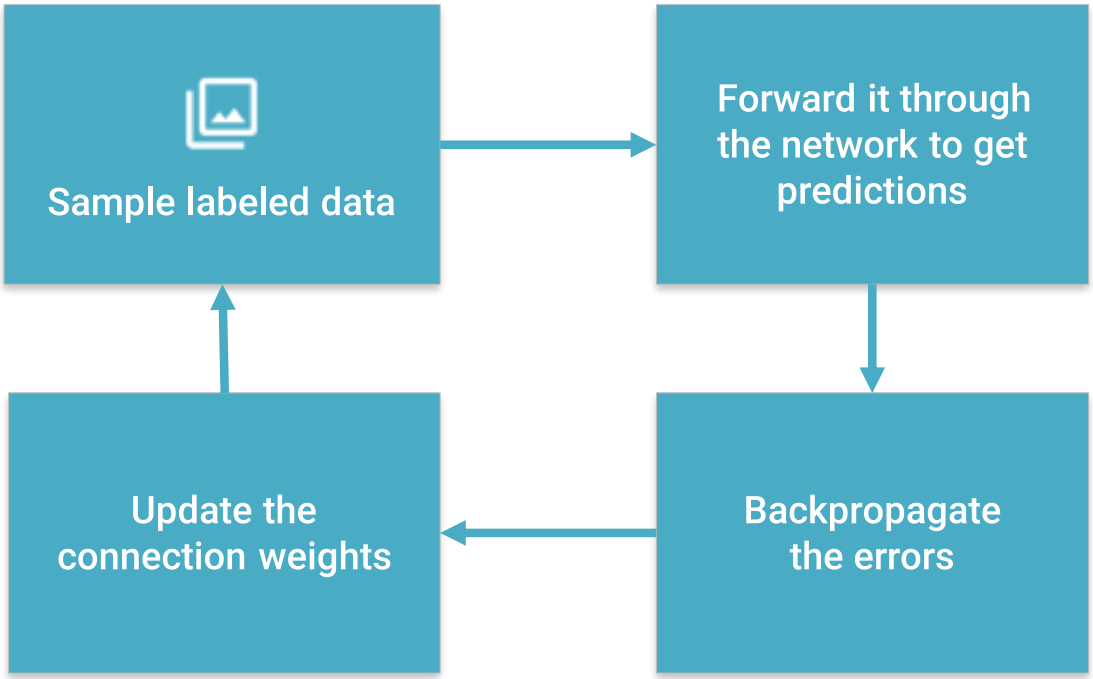
- Par linéarité,

$$\nabla E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i)} \nabla L(\boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Ici, la sortie du réseau de neurones $\boldsymbol{y}^i = f(\boldsymbol{x}^i; \boldsymbol{W})$ est une fonction des données d'entrée $\boldsymbol{x}^i$ et des poids du réseau $\boldsymbol{W}$.
- Nous connaissons le gradient de $L(\boldsymbol{y}^i; \boldsymbol{d}^i)$ par rapport à la variable $\boldsymbol{y}$
- Il reste à calculer

$$\nabla_{\boldsymbol{W}_k} L(\boldsymbol{y}; \boldsymbol{d}) \quad \text{and} \quad \nabla_{\boldsymbol{b}_k} L(\boldsymbol{y}; \boldsymbol{d}) \quad \text{pour } k = 0, \ldots, L.$$

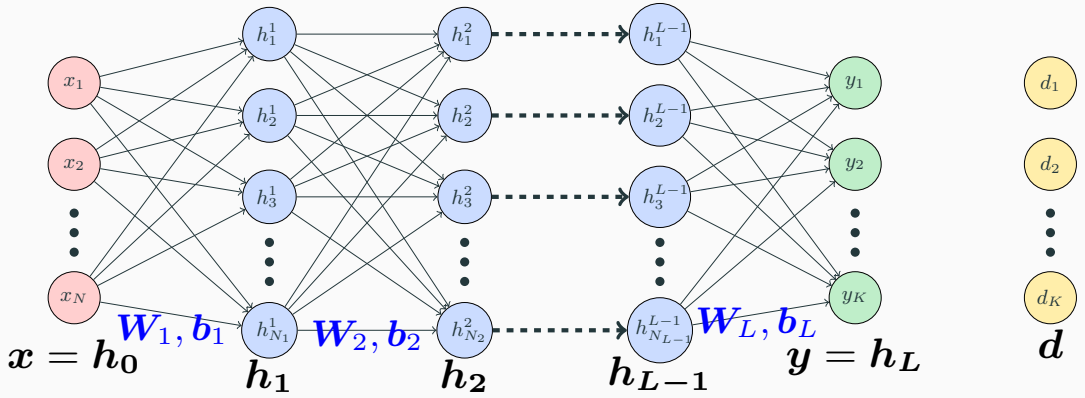
Training process



Learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then using this error signal to change the weights (or parameters) so that predictions get more accurate.

(Source: Lucas Masuch)

RNA – Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'forward'

Initialisation :

$\boldsymbol{h}_0 = \boldsymbol{x}$

for couche $k = 1$ **to** L **do**

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$

Activation :

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$

end

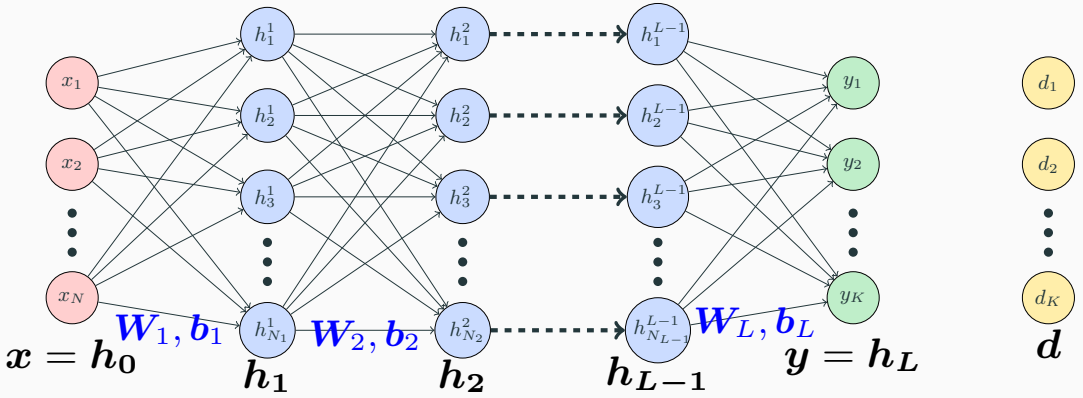
Output :

$\boldsymbol{y} = \boldsymbol{h}_L$

Calcul de la perte (loss) :

$E = L(\boldsymbol{y}; \boldsymbol{d})$

RNA – Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'forward'

Initialisation :

$\boldsymbol{h}_0 = \boldsymbol{x}$

for couche $k = 1$ **to** L **do**

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$

Activation :

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$

end

Output :

$\boldsymbol{y} = \boldsymbol{h}_L$

Calcul de la perte (loss) :

$E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'backward'

Objectif : Calculer le gradient par rapport à tous les paramètres

$$\frac{\partial E}{\partial w_{i,j}^k} = ? \qquad \frac{\partial E}{\partial b_i^k} = ?$$

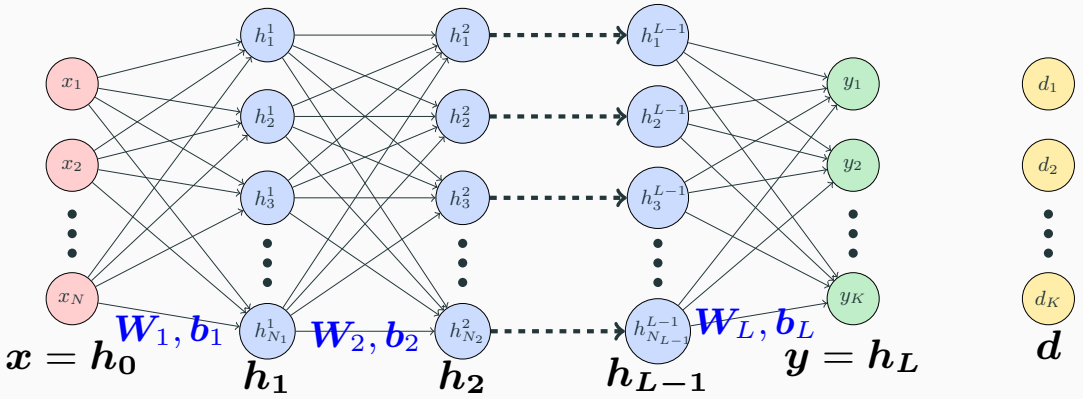
pour all

$k \in \{1, \ldots, L\}$,

$i \in \{1, \ldots, N_k\}$,

$j \in \{1, \ldots, N_{k-1}\}$.

Backpropagation



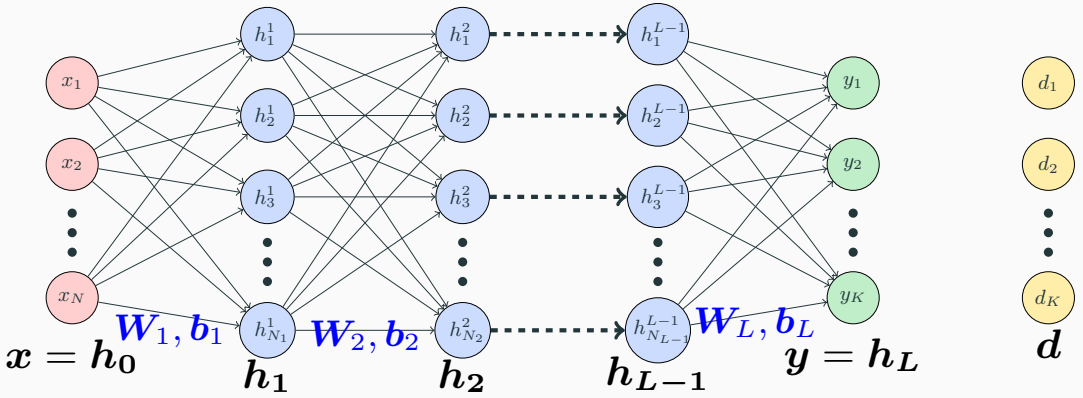
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Backward

- Nous savons comment calculer la fonction de perte et son gradient :

$$\nabla_{\boldsymbol{h}_L} E = \nabla L(\boldsymbol{y}; \boldsymbol{d})$$

Backpropagation

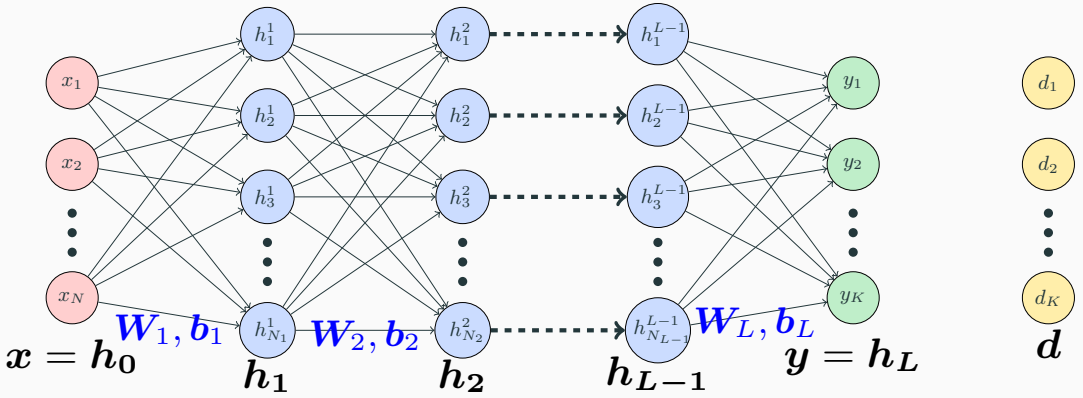


Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport à la sortie de la dernière unité linéaire a_L

$$\boldsymbol{h}_L = g_L(\boldsymbol{a}_L)$$

Backpropagation



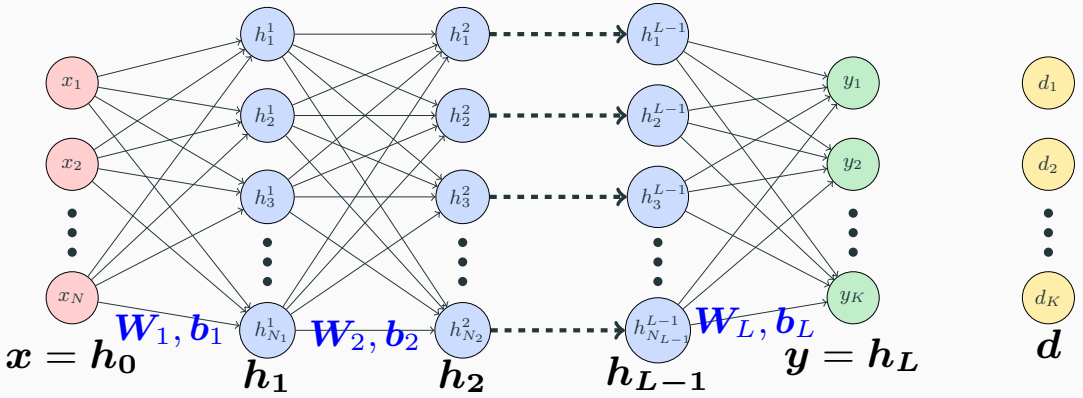
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport à la sortie de la dernière unité linéaire a_L

$$\boldsymbol{h}_L = g_L(\boldsymbol{a}_L)$$

C'est-à-dire que pour tout $i \in \{1, \ldots, N_L\}$, $h_i^L = g_L(a_i^L)$.

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

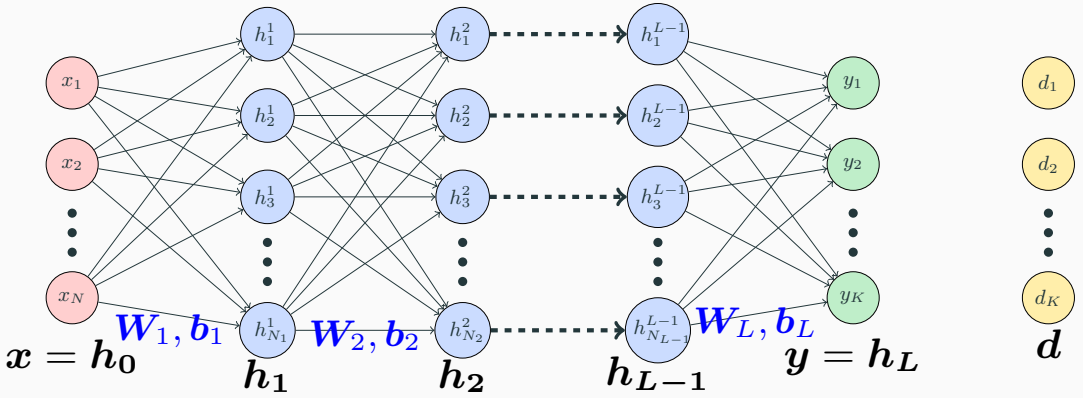
Gradient par rapport à la sortie de la dernière unité linéaire a_L

$$\boldsymbol{h}_L = g_L(\boldsymbol{a}_L)$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $h_i^L = g_L(a_i^L)$. Par la règle de la chaîne,

$$\frac{\partial E}{\partial a_i^L} = \frac{\partial E}{\partial h_i^L} \frac{\partial h_i^L}{\partial a_i^L} = [\nabla_{\boldsymbol{h}_L} E]_i \, g_L'(a_i^L)$$

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport à la sortie de la dernière unité linéaire a_L

$$\boldsymbol{h}_L = g_L(\boldsymbol{a}_L)$$

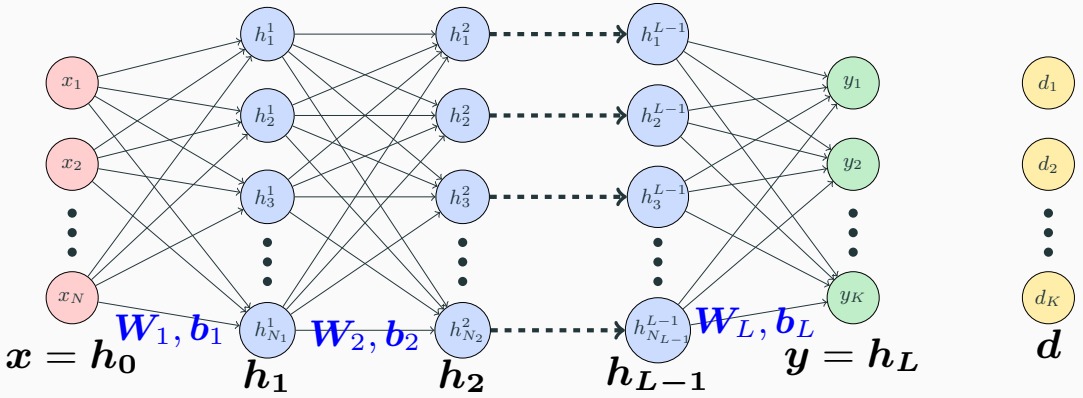
C'est-à-dire que pour tout $i \in \{1, \ldots, N_L\}$, $h_i^L = g_L(a_i^L)$. Par la règle de la chaîne,

$$\frac{\partial E}{\partial a_i^L} = \frac{\partial E}{\partial h_i^L} \frac{\partial h_i^L}{\partial a_i^L} = [\nabla_{\boldsymbol{h}_L} E]_i \, g_L'(a_i^L)$$

Formule vectorielle : $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g_L'(\boldsymbol{a}_L)$

où $\odot$ est le produit composant par composantes entre les vecteurs, c'est-à-dire le produit de Hadamard.

Backpropagation

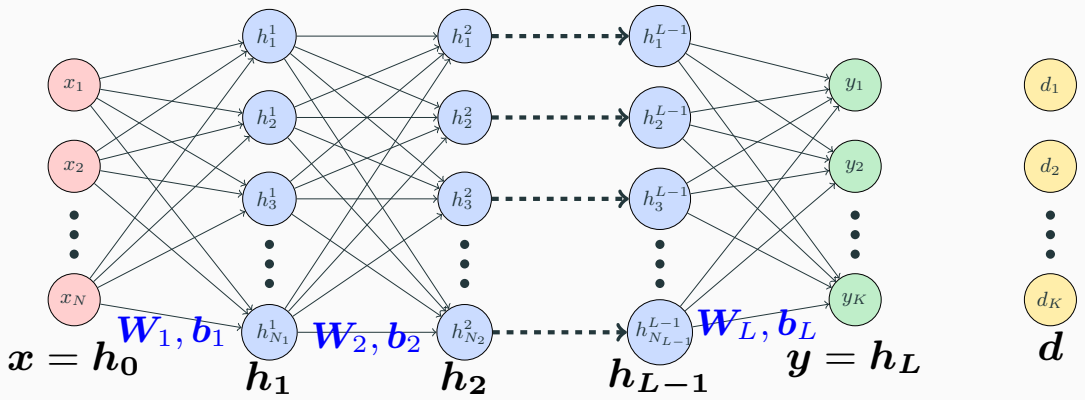


Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport au dernier biais et au dernier poids synaptique

$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

Backpropagation



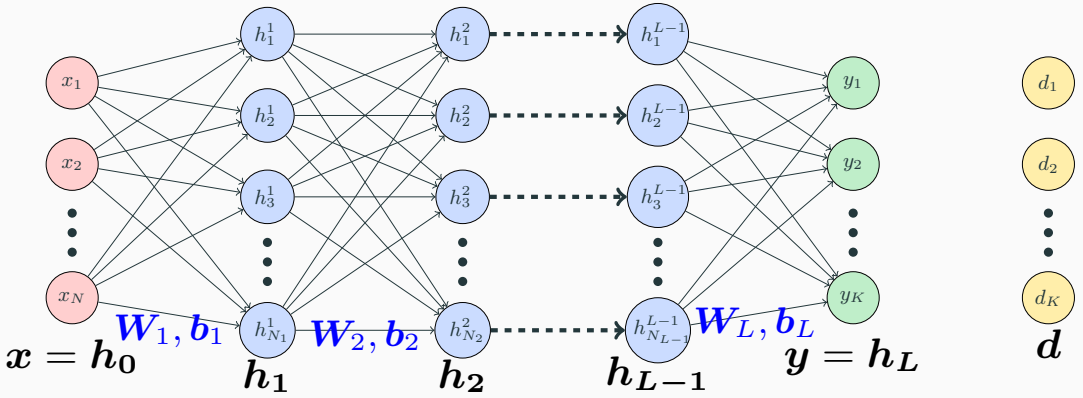
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport au dernier biais et au dernier poids synaptique

$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

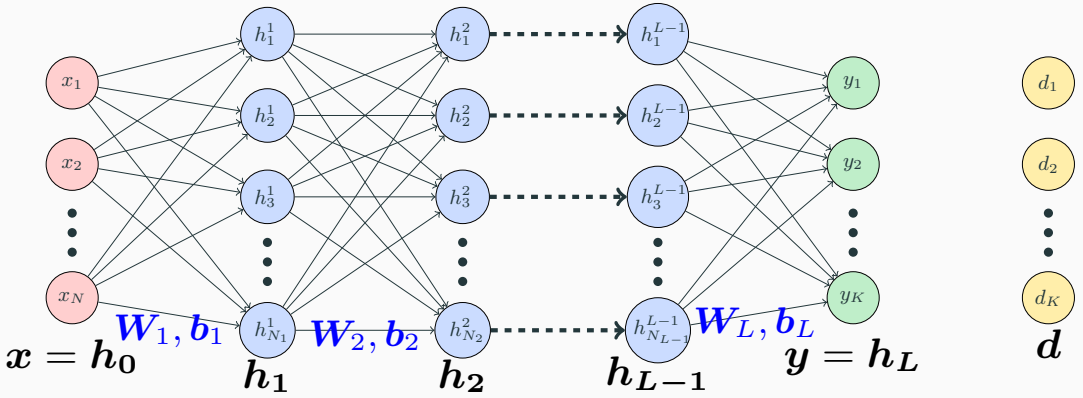
Gradient par rapport au dernier biais et au dernier poids synaptique

$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$. Par la règle de la chaîne, pour tout $i \in \{1, \dots, N_L\}$,

$$\frac{\partial E}{\partial b_i^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial b_i^L}}_{=1} = \frac{\partial E}{\partial a_i^L} = [\nabla_{\boldsymbol{a}_L} E]_i$$

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport au dernier biais et au dernier poids synaptique

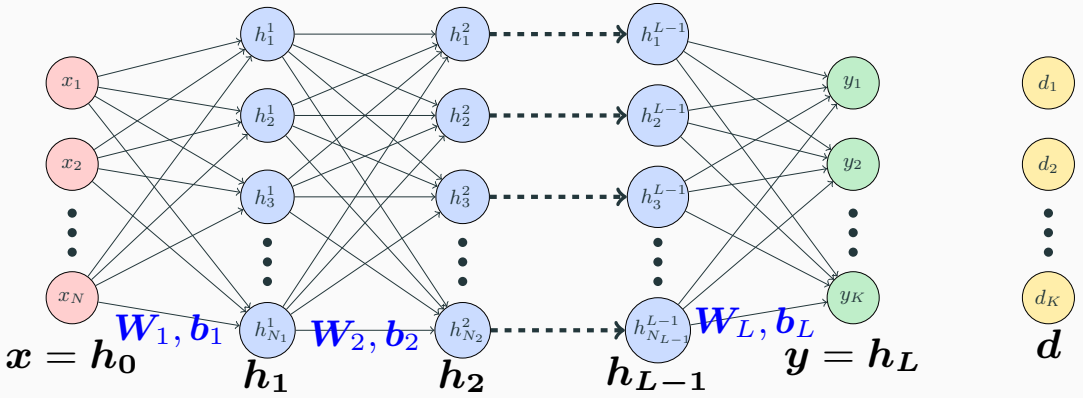
$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$. Par la règle de la chaîne, pour tout $i \in \{1, \dots, N_L\}$,

$$\frac{\partial E}{\partial b_i^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial b_i^L}}_{=1} = \frac{\partial E}{\partial a_i^L} = [\nabla_{\boldsymbol{a}_L} E]_i$$

Formule vectorielle : $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$

Backpropagation



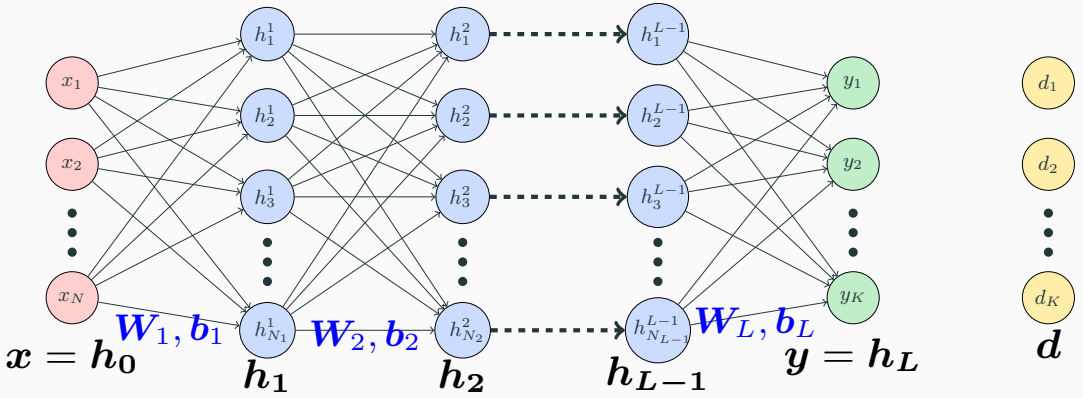
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport aux poids de la dernière unité linéaire W_L

$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

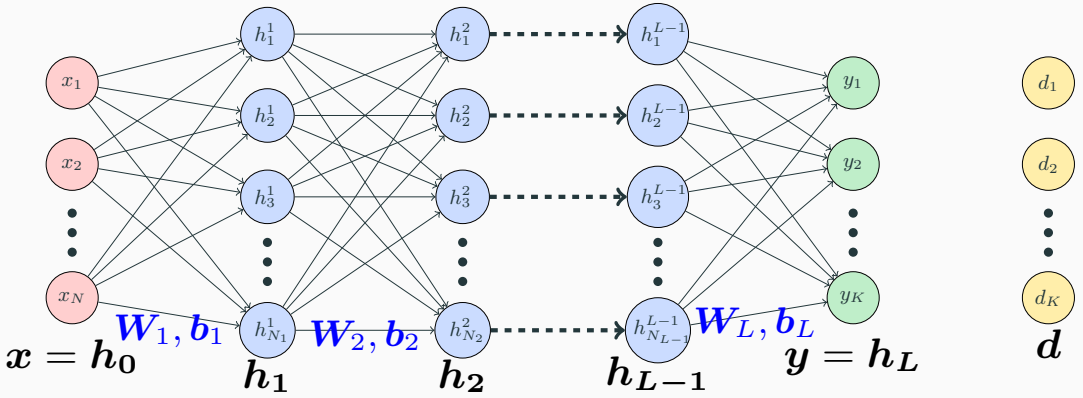
Gradient par rapport aux poids de la dernière unité linéaire $\boldsymbol{W}_L$

$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$. Par la règle de la chaîne, pour tout $i \in \{1, \dots, N_L\}$ et $j \in \{1, \dots, N_{L-1}\}$,

$$\frac{\partial E}{\partial w_{i,j}^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial w_{i,j}^L}}_{= h_j^{L-1}} = \frac{\partial E}{\partial a_i^L} h_j^{L-1} = \left[\nabla_{\boldsymbol{a}_L} E\right]_i \left[\boldsymbol{h}_{L-1}\right]_j$$

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport aux poids de la dernière unité linéaire $\boldsymbol{W}_L$

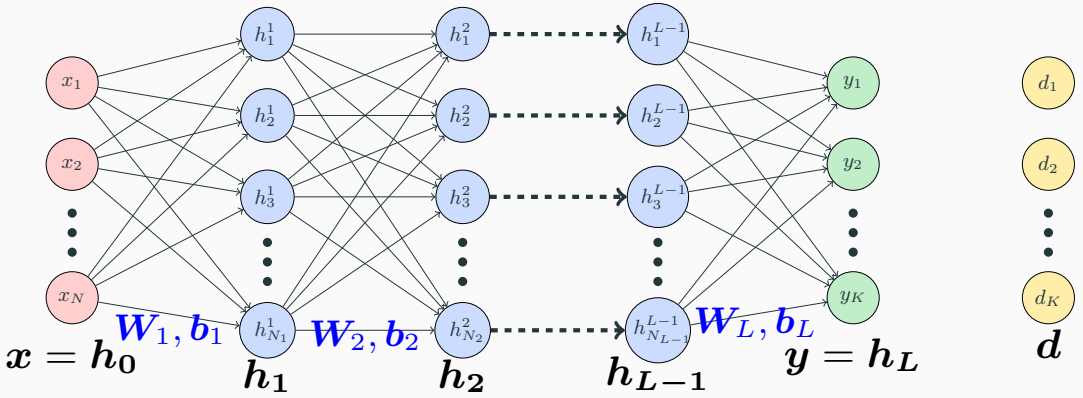
$$\boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$$

C'est-à-dire que pour tout $i \in \{1, \dots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$. Par la règle de la chaîne, pour tout $i \in \{1, \dots, N_L\}$ et $j \in \{1, \dots, N_{L-1}\}$,

$$\frac{\partial E}{\partial w_{i,j}^L} = \frac{\partial E}{\partial a_i^L} \underbrace{\frac{\partial a_i^L}{\partial w_{i,j}^L}}_{= h_j^{L-1}} = \frac{\partial E}{\partial a_i^L} h_j^{L-1} = [\nabla_{\boldsymbol{a}_L} E]_i \, [\boldsymbol{h}_{L-1}]_j$$

Formule matricielle : $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

Backpropagation



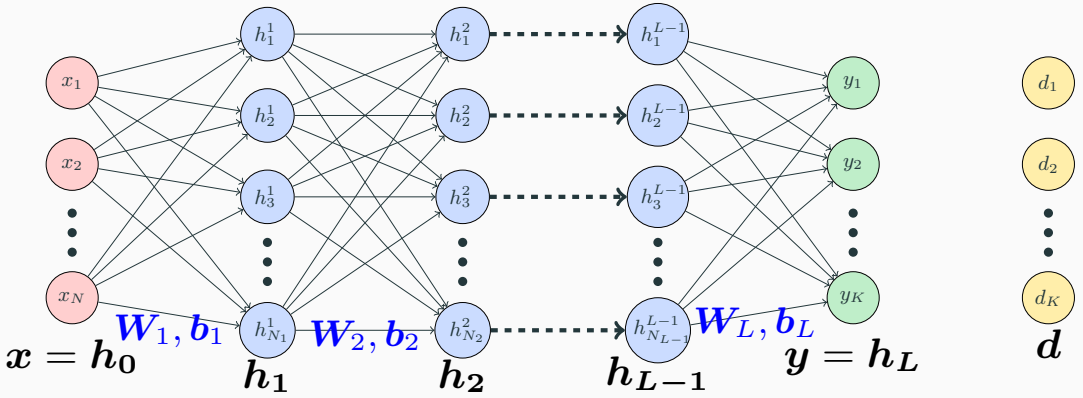
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradients pour les paramètres de la dernière couche

Étant donné le gradient par rapport à la couche de sortie $\nabla_{\boldsymbol{h}_L} E$, nous pouvons maintenant calculer :

- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

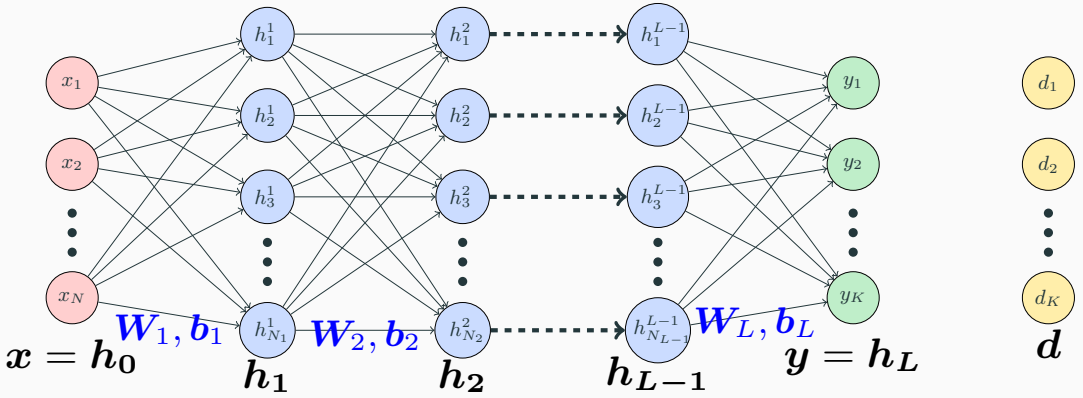
Gradients pour les paramètres de la dernière couche

Étant donné le gradient par rapport à la couche de sortie $\nabla_{\boldsymbol{h}_L} E$, nous pouvons maintenant calculer :

- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

Comment pouvons-nous calculer les gradients pour les paramètres de la couche $L-1$?

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradients pour les paramètres de la dernière couche

Étant donné le gradient par rapport à la couche de sortie $\nabla_{\boldsymbol{h}_L} E$, nous pouvons maintenant calculer :

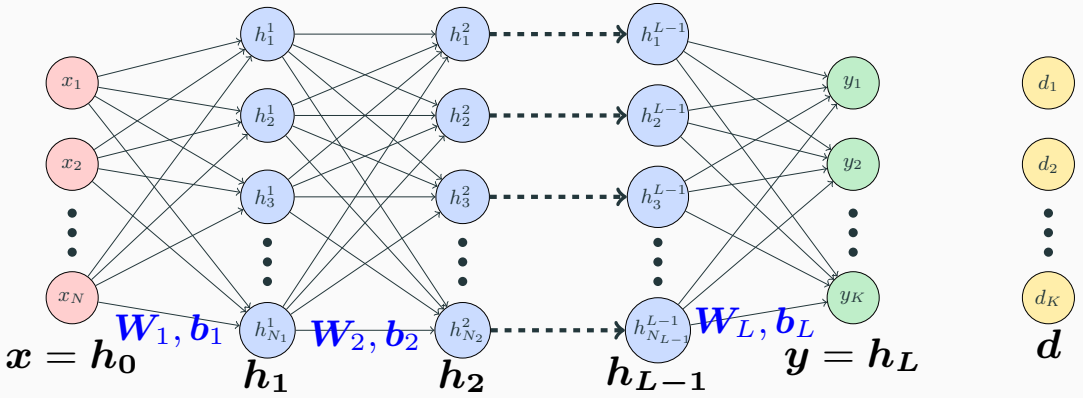
- $\nabla_{\boldsymbol{a}_L} E = \nabla_{\boldsymbol{h}_L} E \odot g'_L(\boldsymbol{a}_L)$
- $\nabla_{\boldsymbol{b}_L} E = \nabla_{\boldsymbol{a}_L} E$
- $\nabla_{\boldsymbol{W}_L} E = \nabla_{\boldsymbol{a}_L} E \, \boldsymbol{h}_{L-1}^T$

Comment pouvons-nous calculer les gradients pour les paramètres de la couche $L-1$?

Nous avons besoin de l'expression du gradient par rapport à la couche cachée juste avant la dernière, $\boldsymbol{h}_{L-1}$... et ensuite, les mêmes formules s'appliquent !

$$\nabla_{\boldsymbol{h}_{L-1}} E = ?$$

Backpropagation



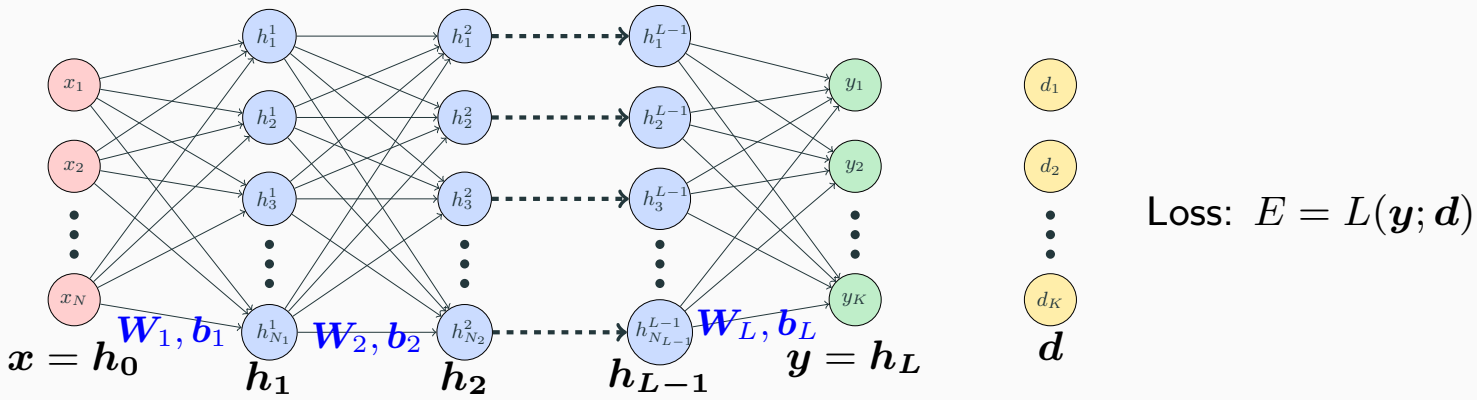
Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport à la couche cachée juste avant la dernière h_{L-1}

Ici, même pour calculer la dérivée partielle $\dfrac{\partial E}{\partial h_j^{L-1}}$, nous devons utiliser le calcul différentiel pour les fonctions multidimensionnelle, car h_j^{L-1} apparaît dans chaque composant de $\boldsymbol{a}_L$:

Pour tout $i \in \{1, \ldots, N_L\}$, $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$.

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Gradient par rapport à la couche cachée juste avant la dernière h_{L-1}

Rappelons la règle de dérivation pour la composition avec des applications affines :

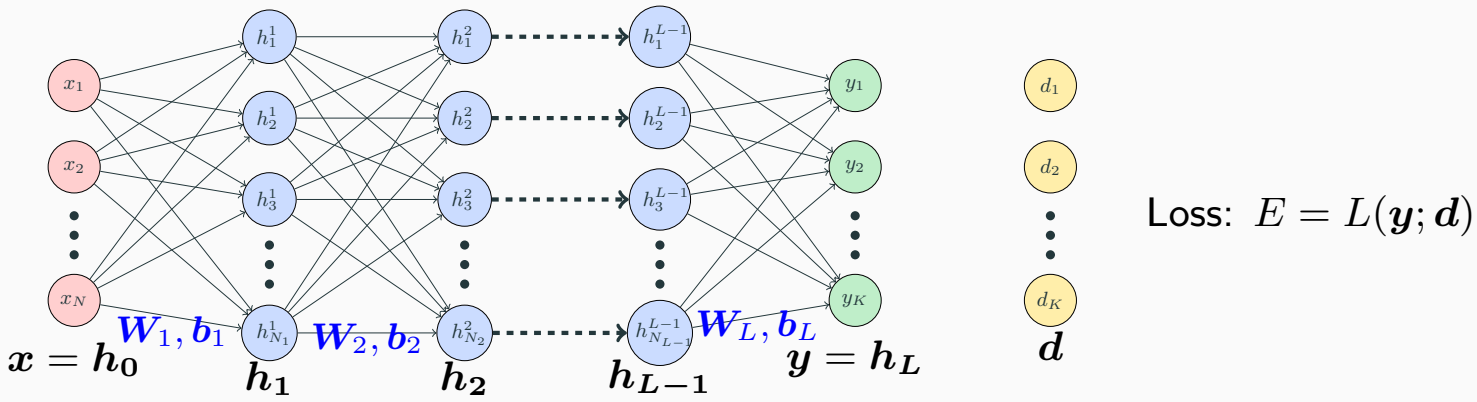
$$\text{Pour} \quad \varphi(\boldsymbol{x}) = f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}) \quad \text{on a} \quad \nabla\varphi(\boldsymbol{x}) = \boldsymbol{A}^T \nabla f(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}).$$

En utilisant la décomposition

$$\begin{array}{ccccc} \mathbb{R}^{N_{L-1}} & \to & \mathbb{R}^{N_L} & \to & \mathbb{R} \\ \boldsymbol{h}_{L-1} & \mapsto & \boldsymbol{a}_L = \boldsymbol{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L & \mapsto & E \end{array}$$

Formule vectorielle : $\nabla_{\boldsymbol{h}_{L-1}} E = \boldsymbol{W}_L{}^T \nabla_{\boldsymbol{a}_L} E$

Backpropagation

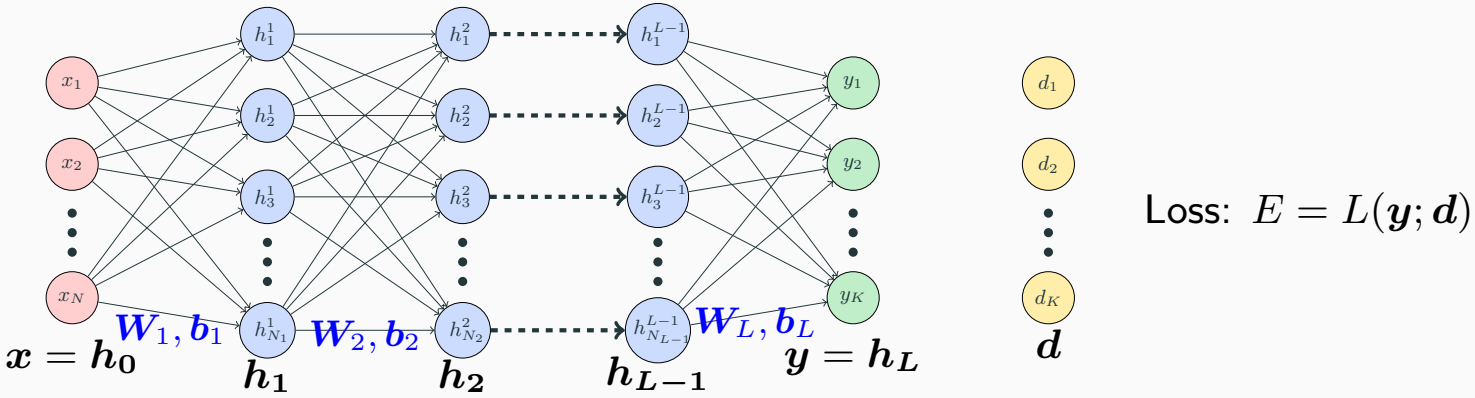


Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

On connaît $\nabla_{\boldsymbol{h}_{L-1}} E$ et :

- $\boldsymbol{h}_{L-1} = g_{L-1}(\boldsymbol{a}_{L-1}) \Rightarrow \nabla_{\boldsymbol{a}_{L-1}} E = \nabla_{\boldsymbol{h}_{L-1}} E \odot g'_{L-1}(\boldsymbol{a}_{L-1})$
- $\boldsymbol{a}_{L-1} = W_{L-1}\boldsymbol{h}_{L-2} + \boldsymbol{b}_{L-1}$
- Comme précédemment,
- $\nabla_{\boldsymbol{b}_{L-1}} E = \nabla_{\boldsymbol{a}_{L-1}} E$
- $\nabla \boldsymbol{W}_{L-1} E = \nabla_{\boldsymbol{a}_{L-1}} E \boldsymbol{h}_{L-2}^T$

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'Forward'

Initialization:

$\boldsymbol{h}_0 = \boldsymbol{x}$

for layer $k = 1$ **to** L **do**

Linear unit:

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$

Componentwise non-linear activation:

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$

end

Output layer:

$\boldsymbol{y} = \boldsymbol{h}_L$

Compute loss:

$E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'Backward'

Initialization: Gradient of output layer:

$\nabla_{\boldsymbol{h}_L} E = \nabla L(\boldsymbol{y}; \boldsymbol{d})$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g_k'(\boldsymbol{a}_k)$

Gradient of layer bias:

$\nabla_{\boldsymbol{b}_k} E = \boldsymbol{\delta}_k$

Gradient of weights:

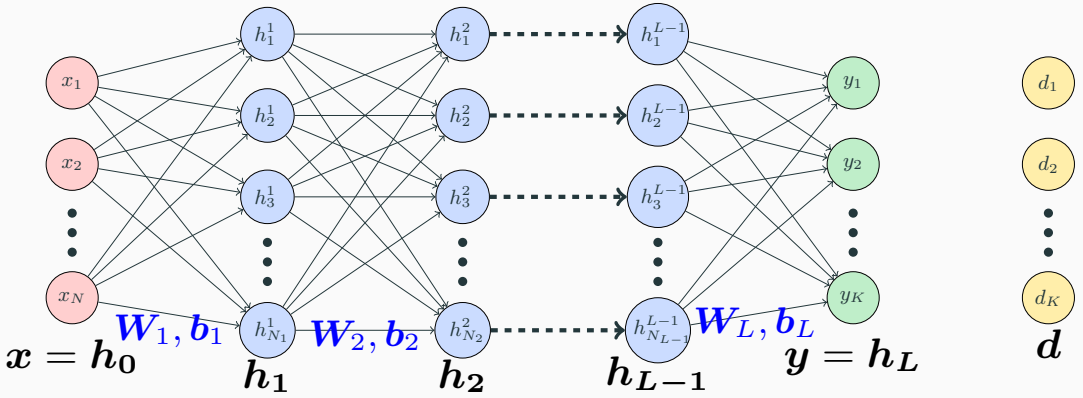
$\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

Gradient of previous hidden layer:

$\nabla_{\boldsymbol{h}_{k-1}} E = \boldsymbol{W}_k^T \boldsymbol{\delta}_k$

end

Backpropagation



Loss: $E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'Forward'

Initialization:

$\boldsymbol{h}_0 = \boldsymbol{x}$

for layer $k = 1$ **to** L **do**

Linear unit:

$\boldsymbol{a}_k = \boldsymbol{W}_k \boldsymbol{h}_{k-1} + \boldsymbol{b}_k$ **(stored)**

Componentwise non-linear activation:

$\boldsymbol{h}_k = g_k(\boldsymbol{a}_k)$ **(stored)**

end

Output layer:

$\boldsymbol{y} = \boldsymbol{h}_L$

Compute loss:

$E = L(\boldsymbol{y}; \boldsymbol{d})$

Passage 'Backward'

Initialization: Gradient of output layer:

$\nabla_{\boldsymbol{h}_L} E = \nabla L(\boldsymbol{y}; \boldsymbol{d})$

for layer $k = L$ **to** 1 **do**

Componentwise gain of error:

$\boldsymbol{\delta}_k = \nabla_{\boldsymbol{a}_k} E = \nabla_{\boldsymbol{h}_k} E \odot g_k'(\boldsymbol{a}_k)$

Gradient of layer bias:

$\nabla_{\boldsymbol{b}_k} E = \boldsymbol{\delta}_k$

Gradient of weights:

$\nabla_{\boldsymbol{W}_k} E = \boldsymbol{\delta}_k \boldsymbol{h}_{k-1}^T$

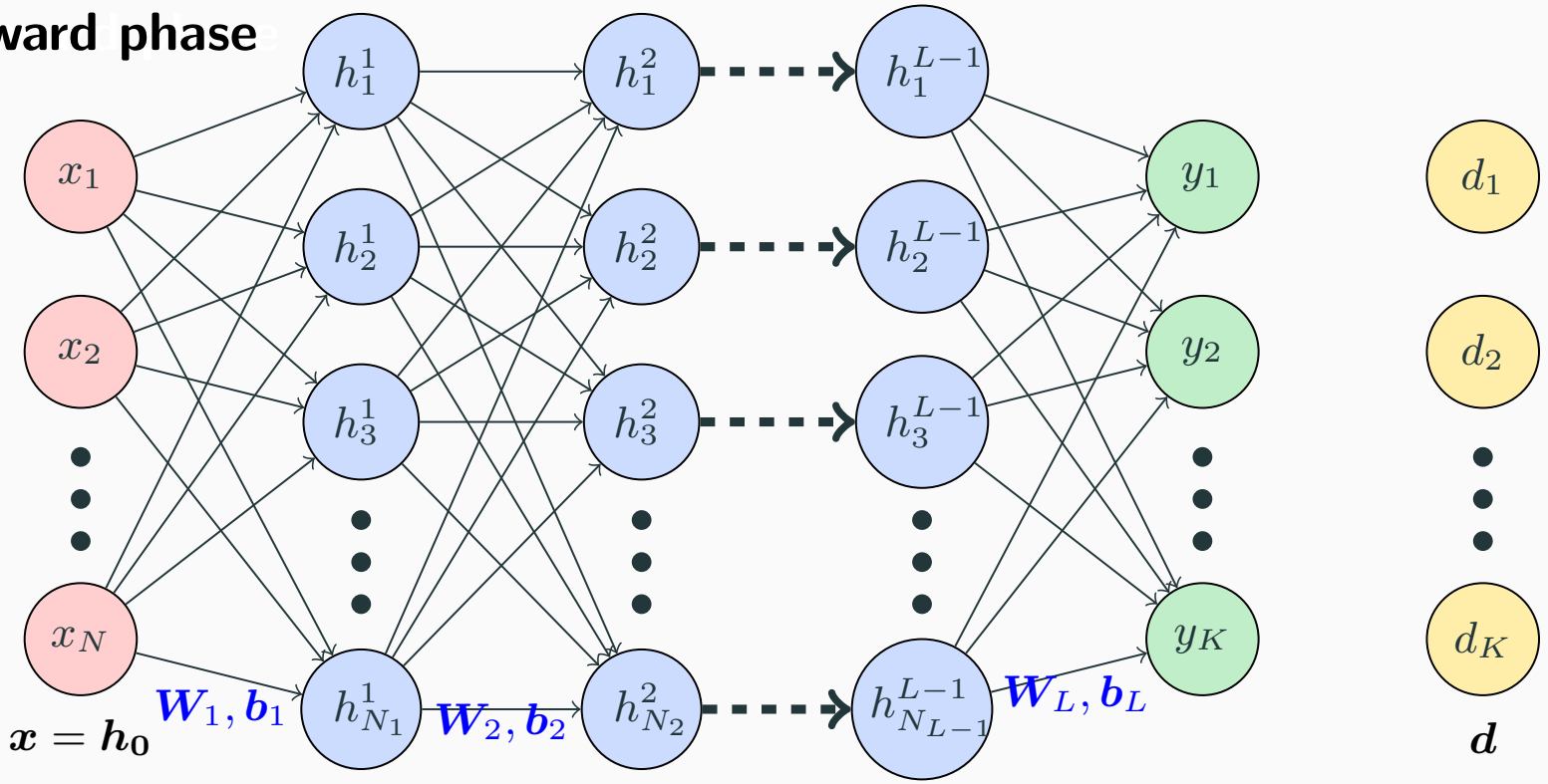
Gradient of previous hidden layer:

$\nabla_{\boldsymbol{h}_{k-1}} E = \boldsymbol{W}_k^T \boldsymbol{\delta}_k$

end

Error backpropagation

Forward phase

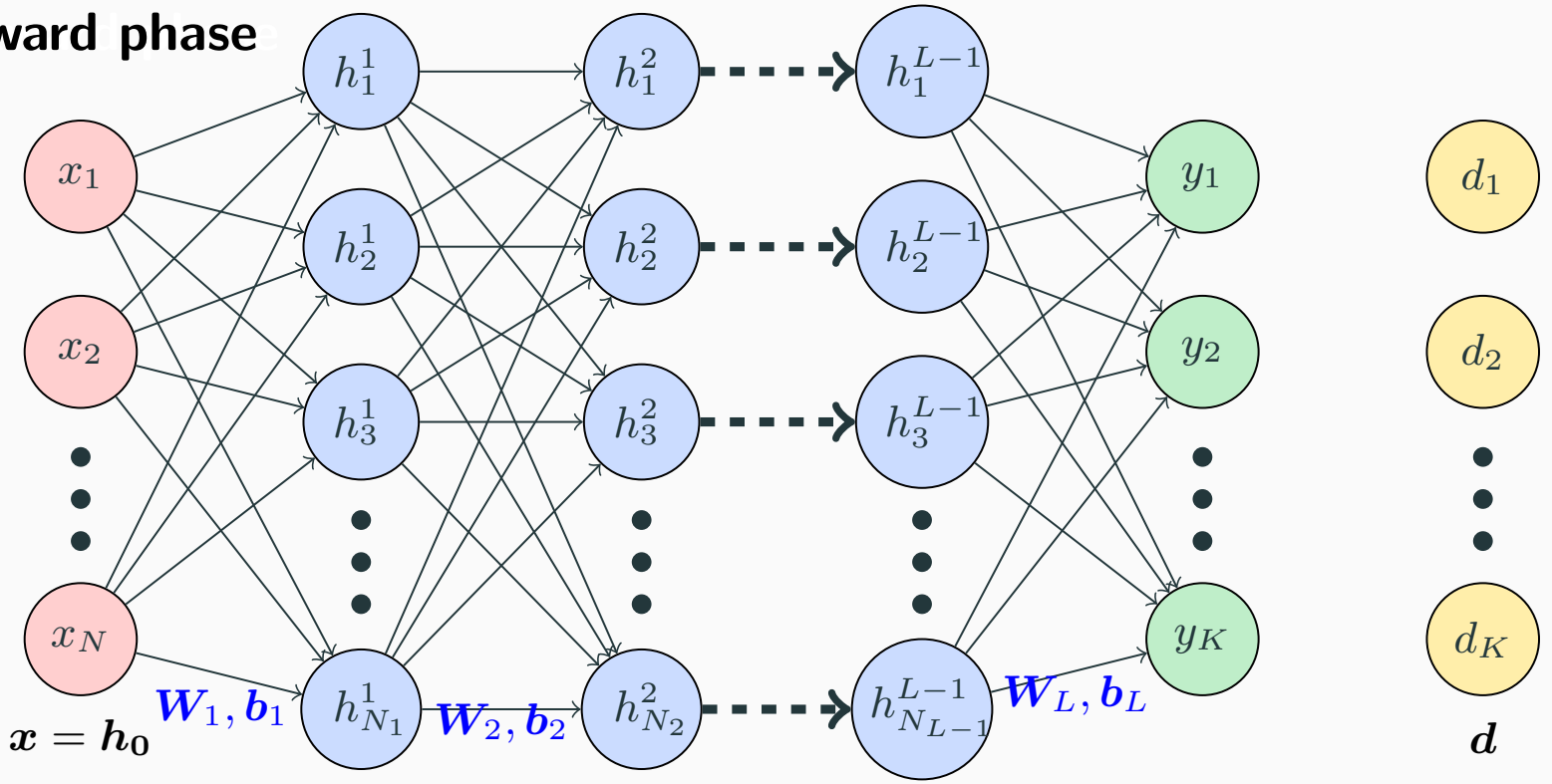


$E = L(y, d)$

Input Layer | Hidden Layers | Output Layer | Label

Error backpropagation

Forward phase



$$\boldsymbol{a_1} = \boldsymbol{W}_1 \boldsymbol{h_0} + \boldsymbol{b}_1$$

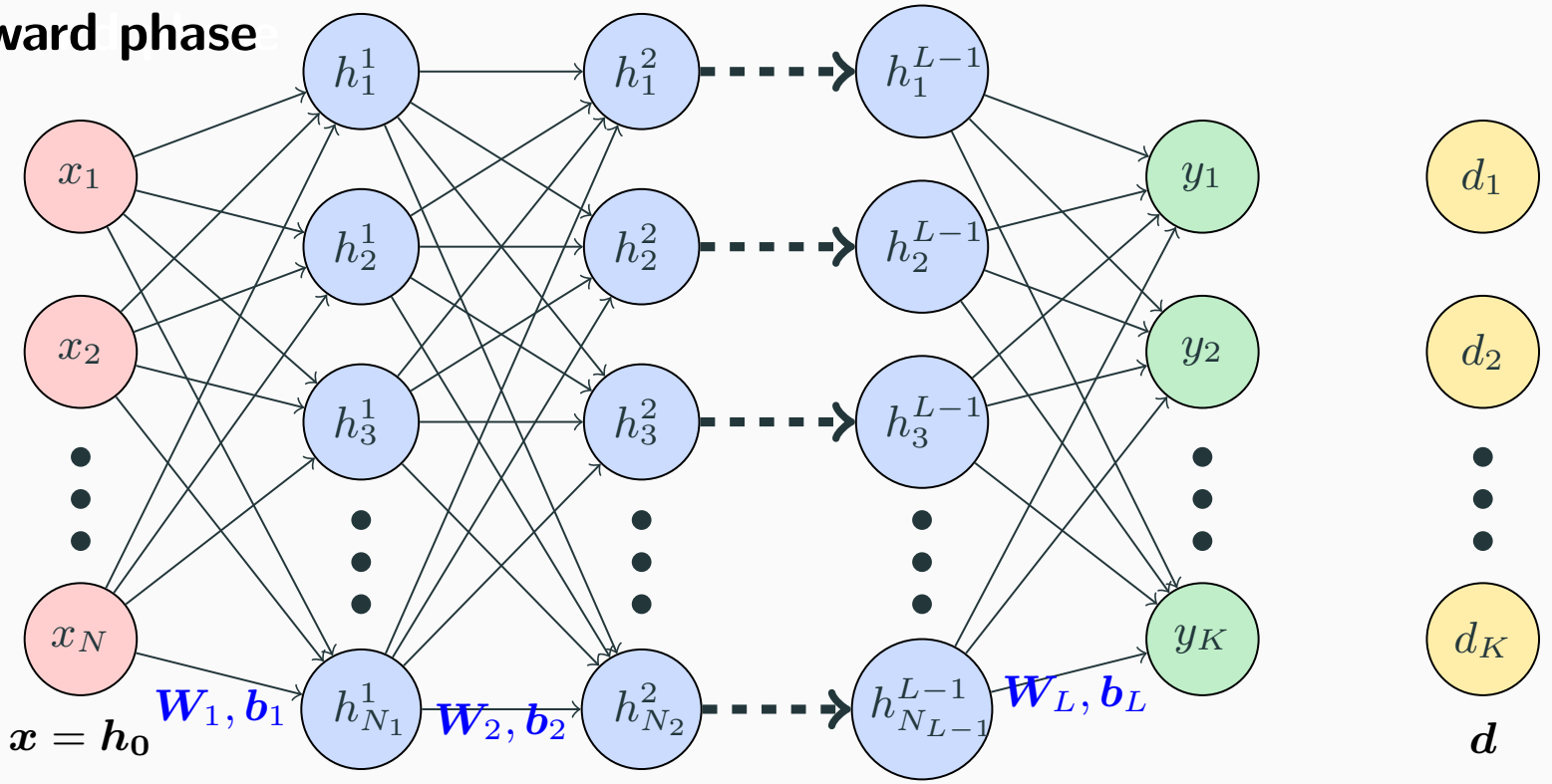
$$\boldsymbol{h_1} = g_1(\boldsymbol{a_1})$$

$E = L(y, d)$

Input Layer | Hidden Layers | Output Layer | Label

Error backpropagation

Forward phase



$\boldsymbol{a_1} = \boldsymbol{W_1 h_0} + \boldsymbol{b_1}$ $\quad \boldsymbol{a_2}$

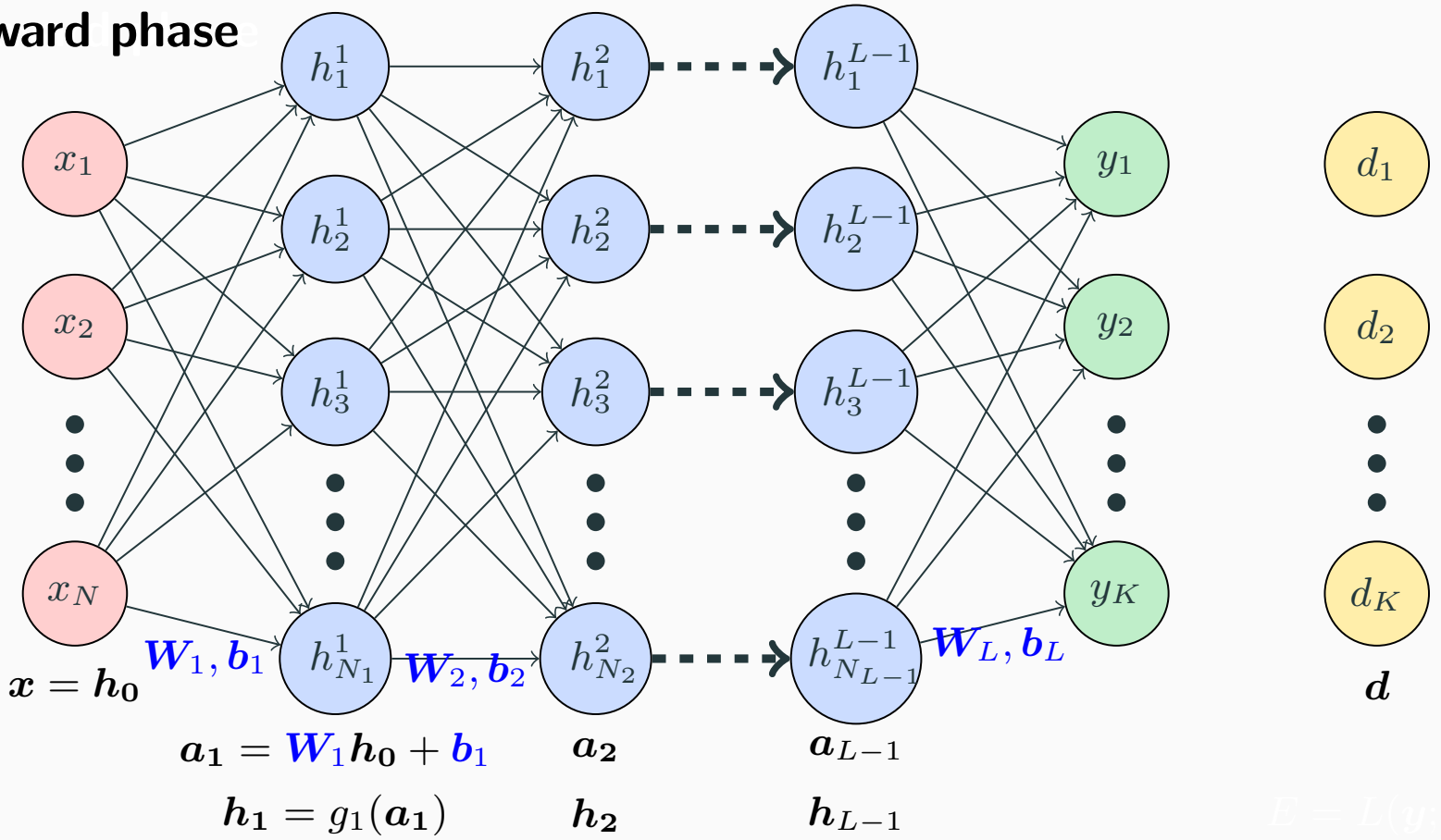
$\boldsymbol{h_1} = g_1(\boldsymbol{a_1})$ $\quad \boldsymbol{h_2}$

$E = L(y, d)$

Input Layer | Hidden Layers | Output Layer | Label

Error backpropagation

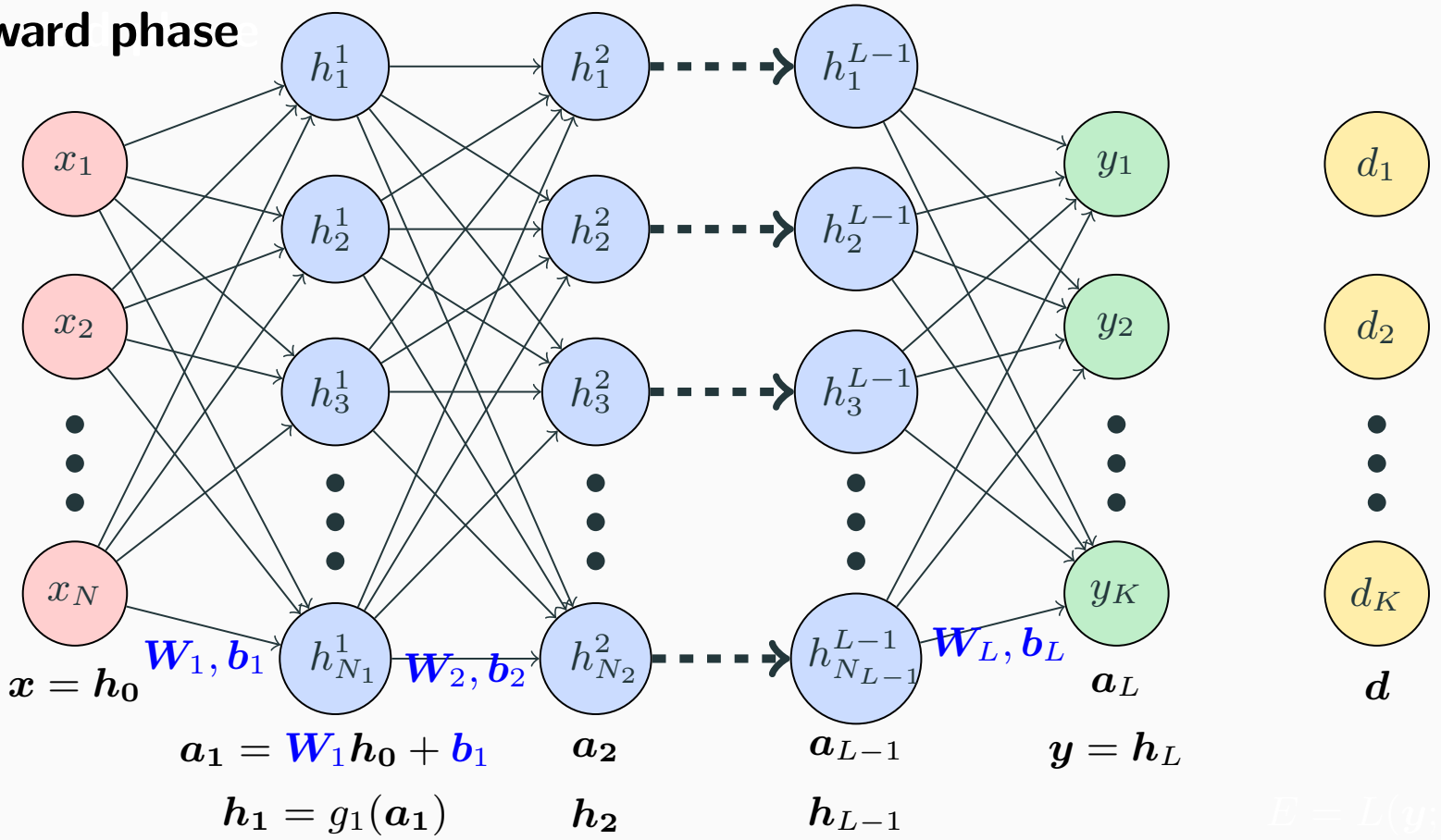
Forward phase



Input Layer | Hidden Layers | Output Layer | Label

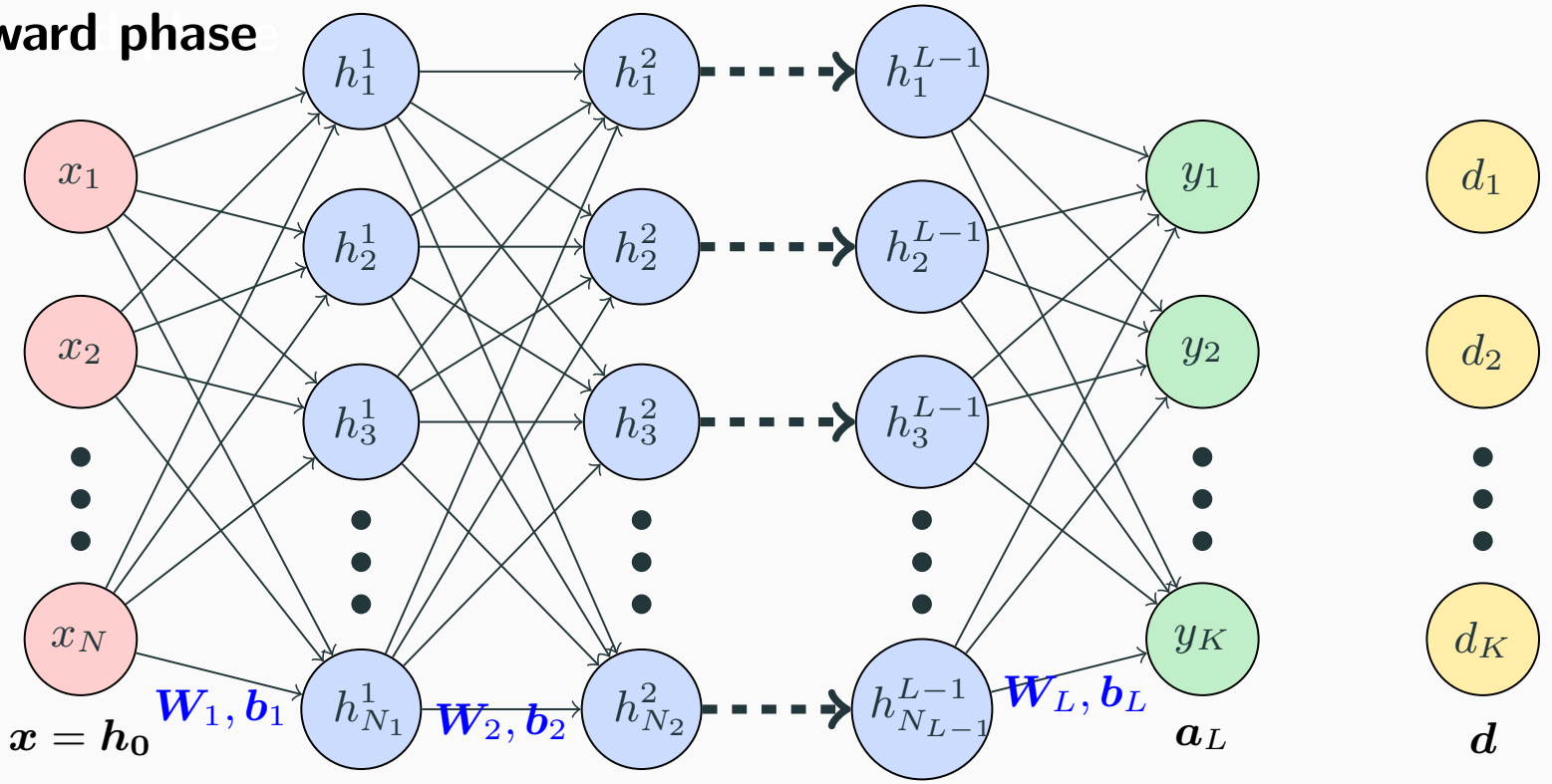
Error backpropagation

Forward phase



Error backpropagation

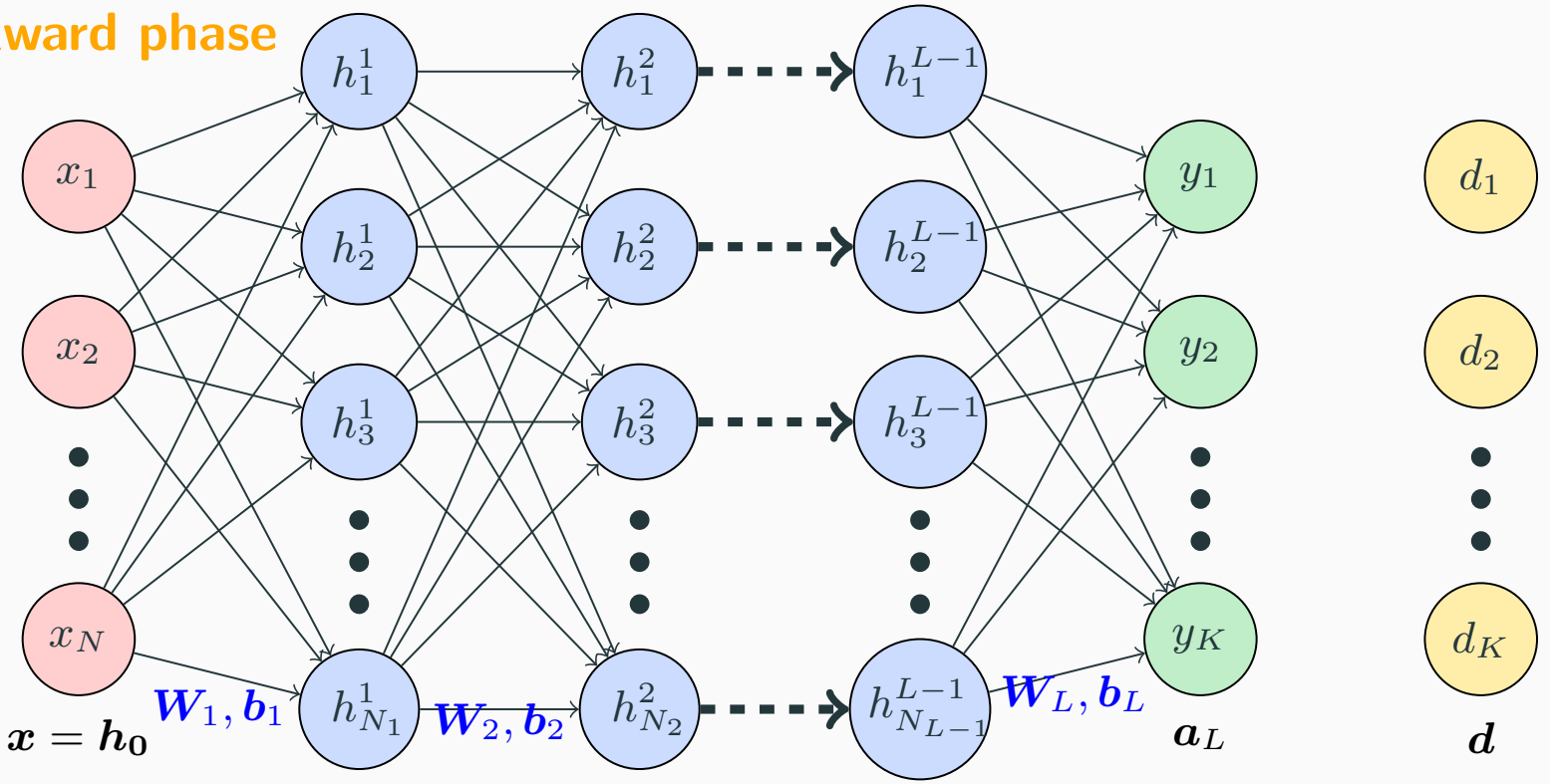
Forward phase



Input Layer | Hidden Layers | Output Layer | Label

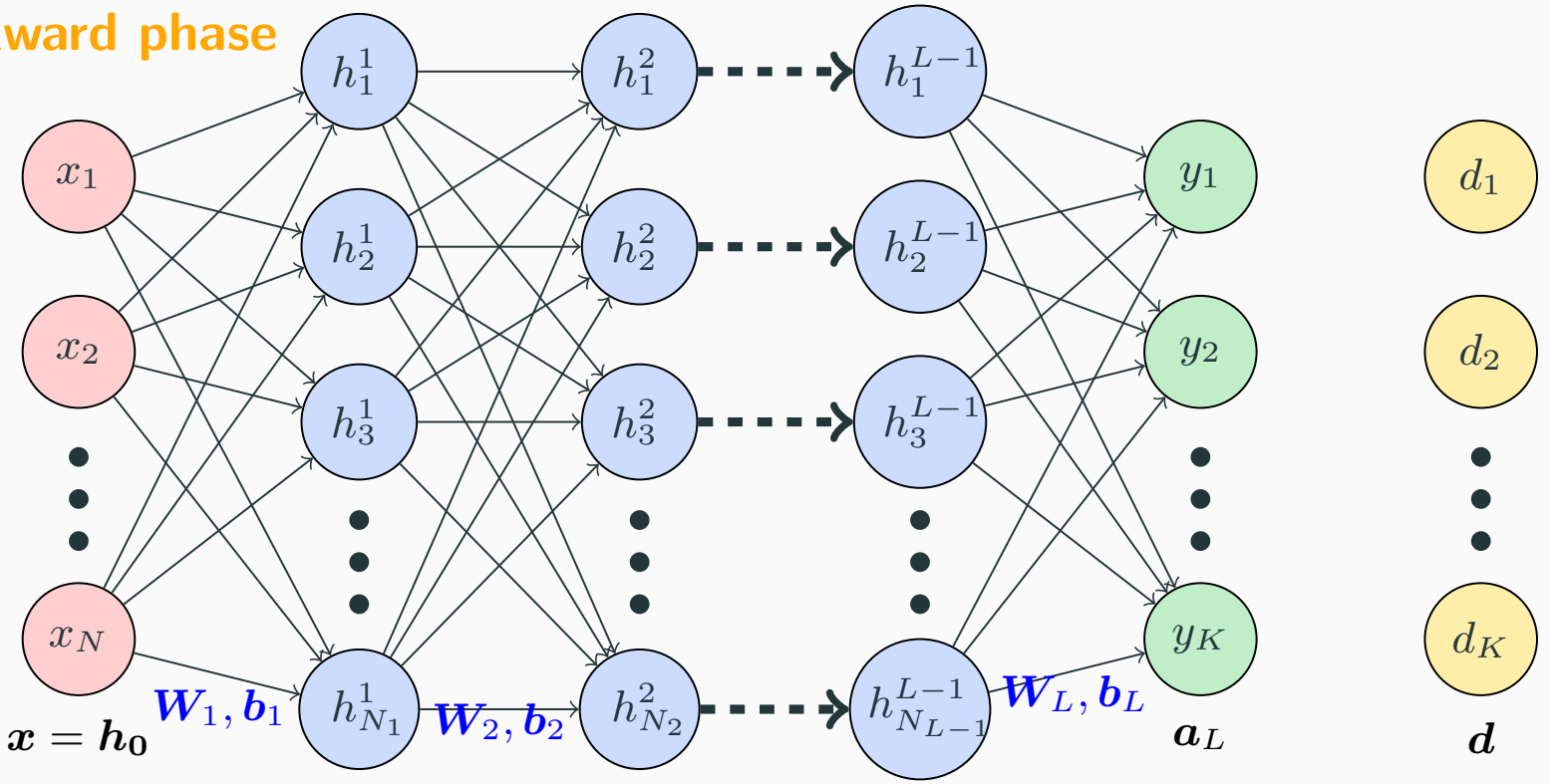
Error backpropagation

Backward phase



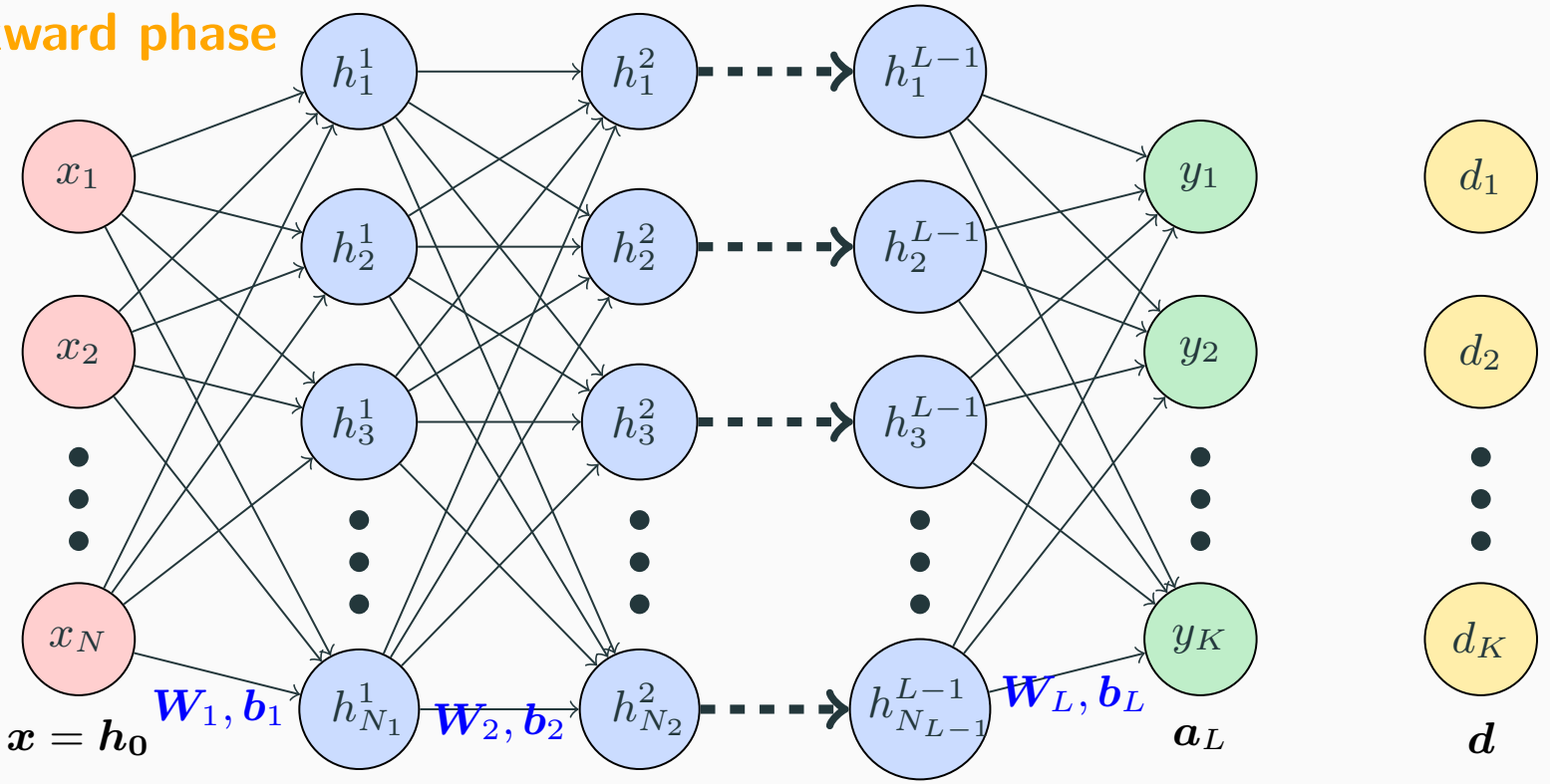
Error backpropagation

Backward phase



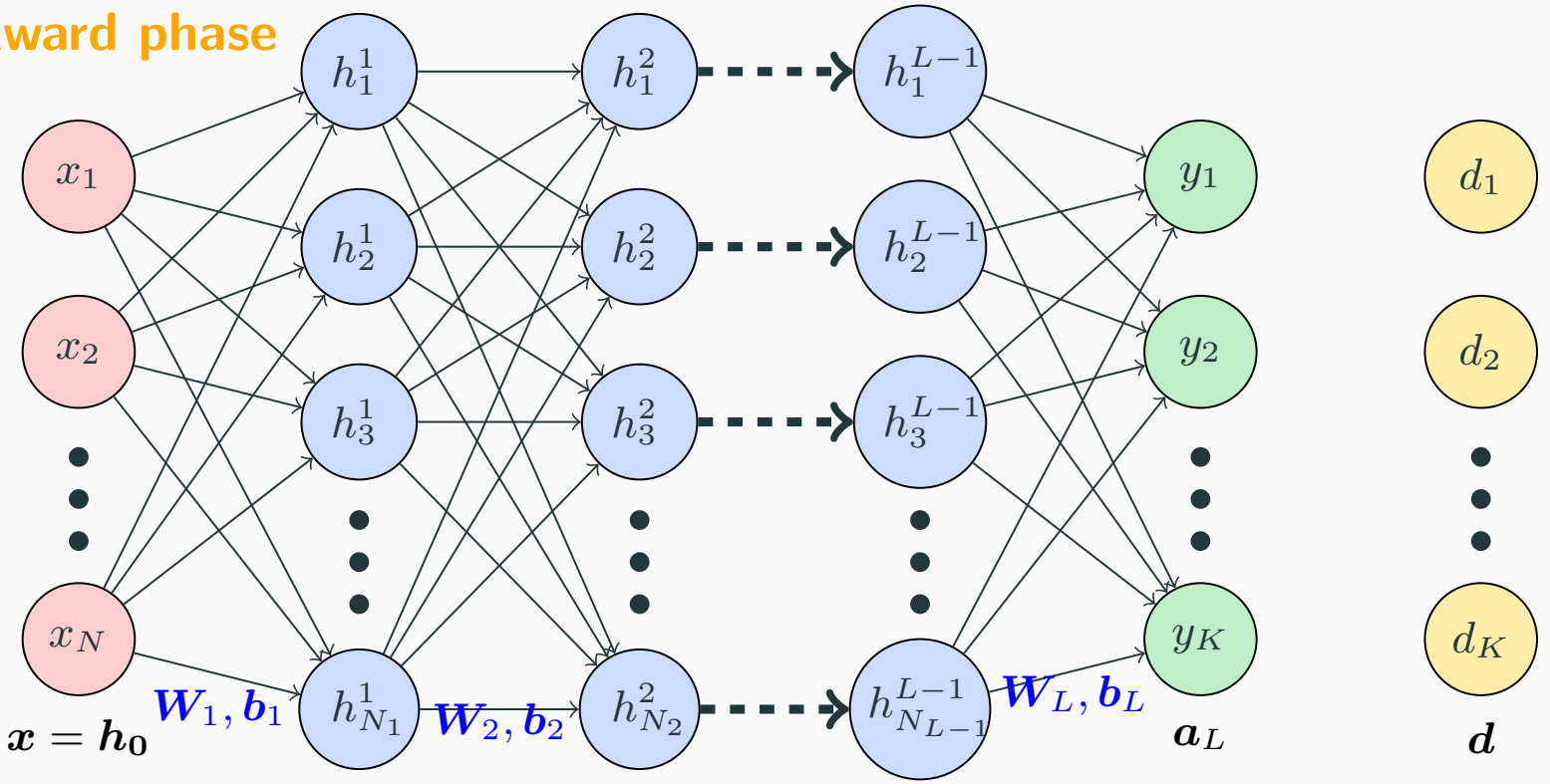
Error backpropagation

Backward phase



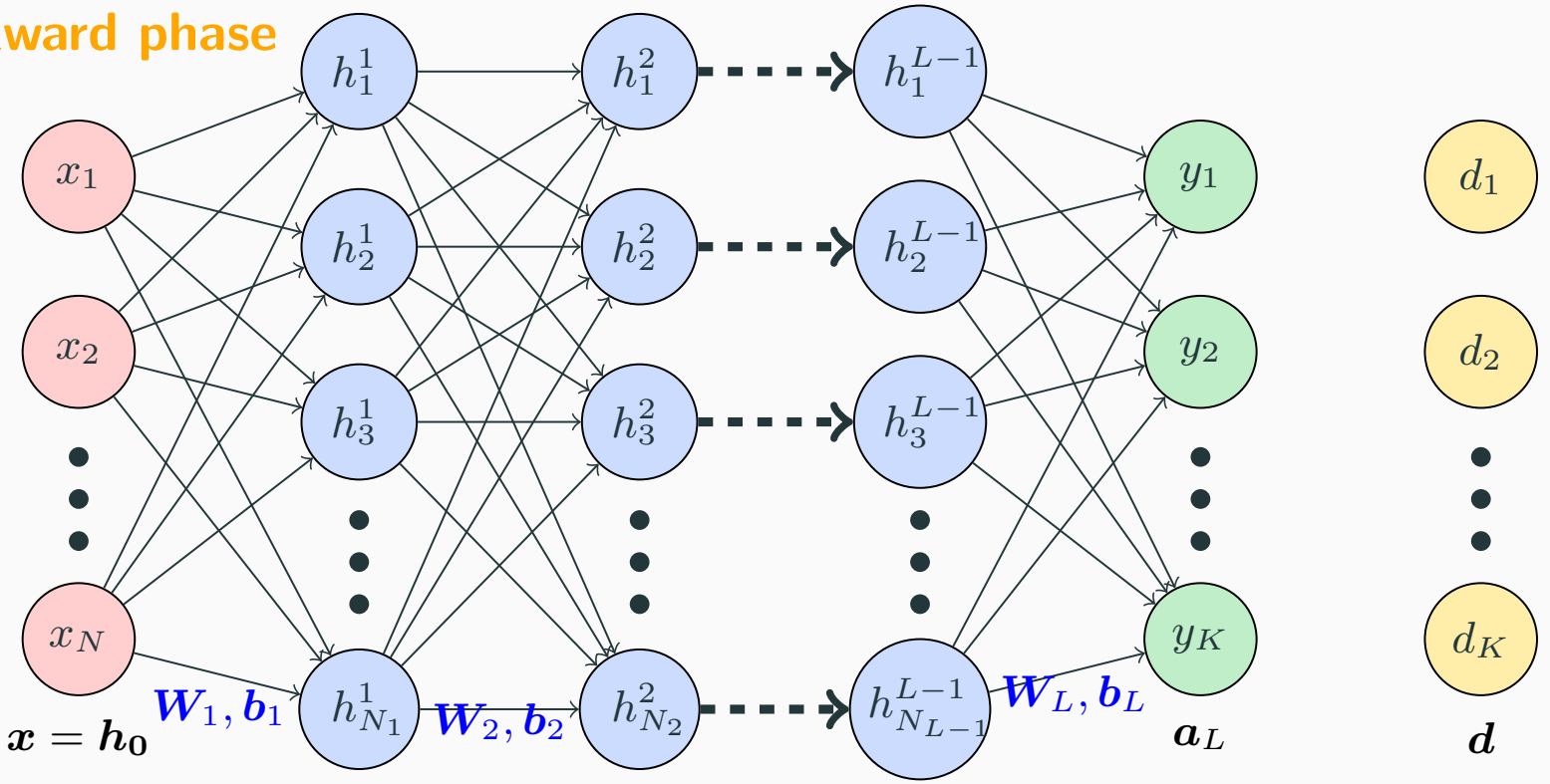
Error backpropagation

Backward phase



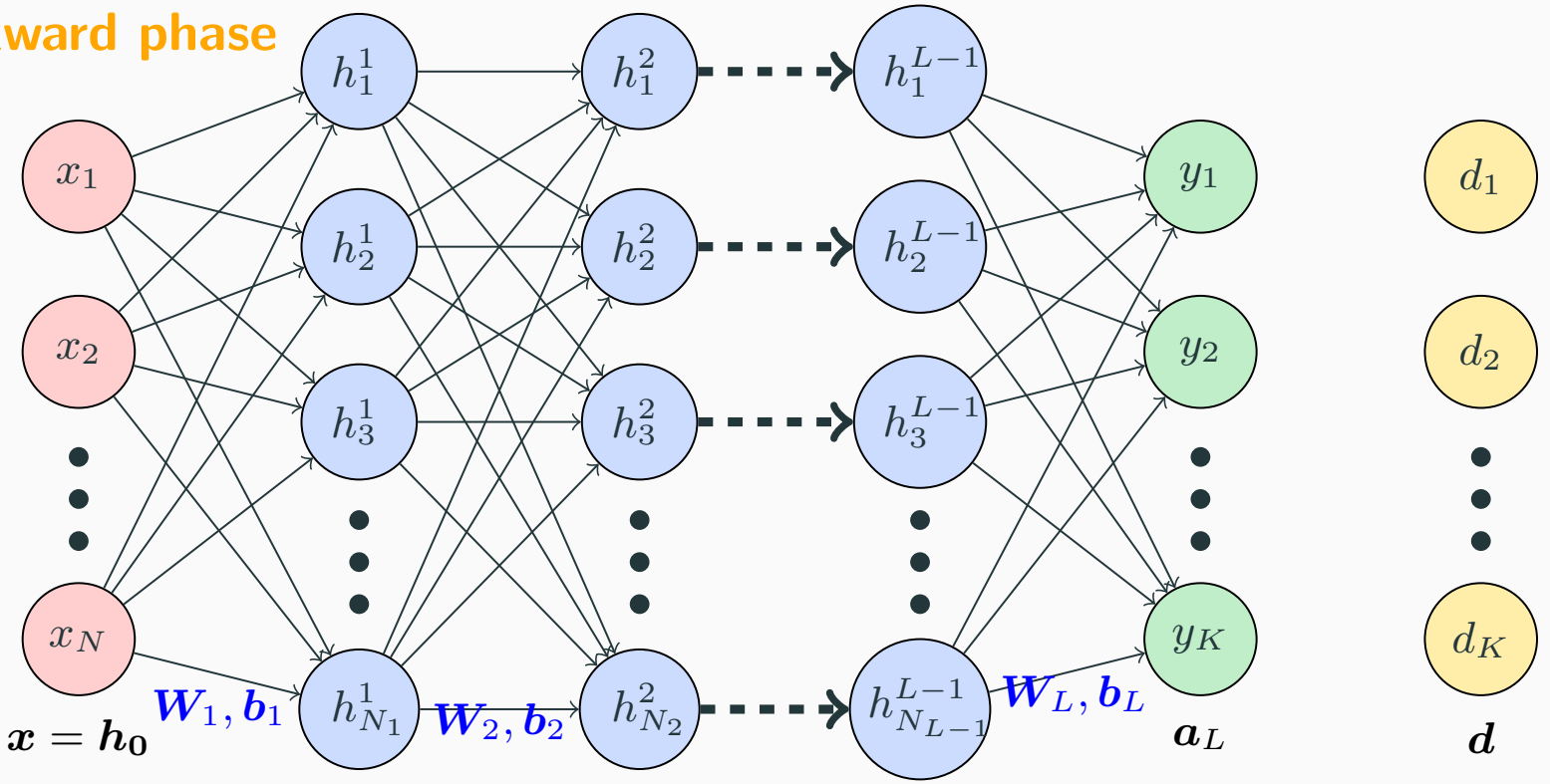
Error backpropagation

Backward phase



Error backpropagation

Backward phase



Pourquoi la backpropagation est-elle si efficace ?

- Évite de refaire de nombreux calculs
- Aucun paramètre à ajuster
- Facile à implémenter
- Cette méthode peut être vue comme une multiplication Jacobienne "dans la bonne direction"

Quelques limitations :

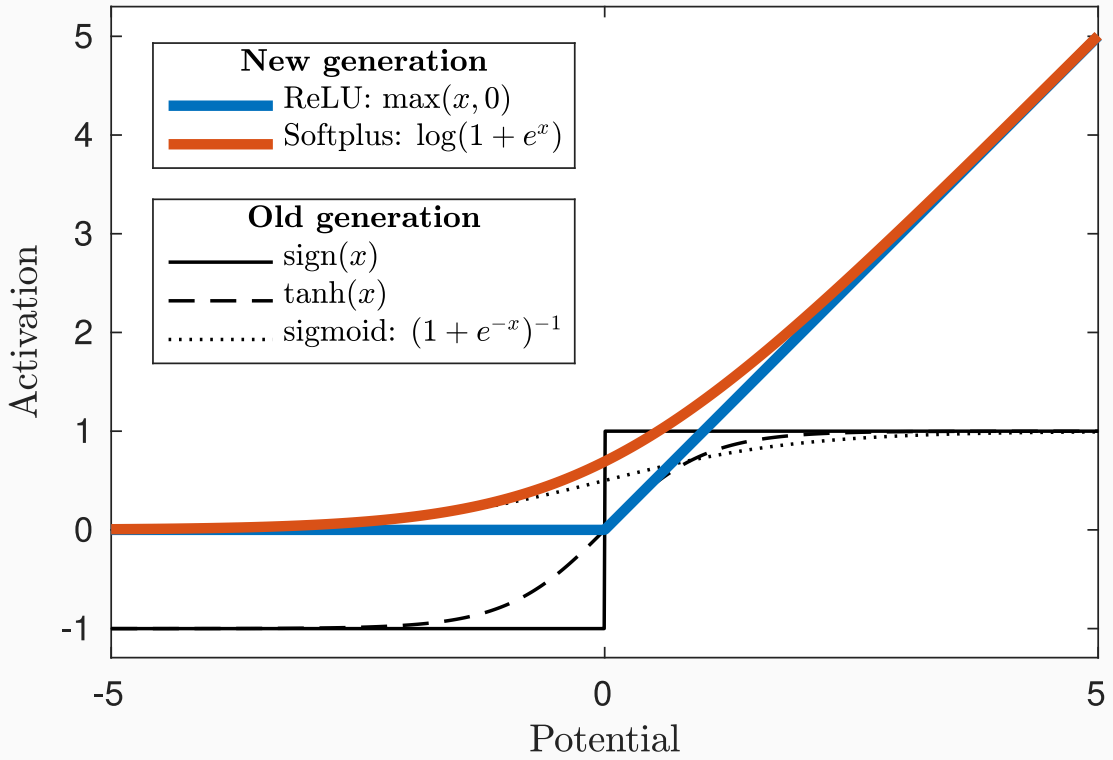
- Gourmande en mémoire
- L'apprentissage par backpropagation ne nécessite pas la normalisation des vecteurs d'entrée ; cependant, la normalisation peut améliorer les performances.

Retour sur les fonctions d'activation

En considérant

$$\boldsymbol{h}_{L-1} = g_{L-1}(\boldsymbol{a}_{L-1}) \Rightarrow \nabla_{\boldsymbol{a}_{L-1}} E = \nabla_{\boldsymbol{h}_{L-1}} E \odot g'_{L-1}(\boldsymbol{a}_{L-1})$$

Pourquoi ReLU est-il préférable à la fonction signe ?



Quelques mots sur la descente de gradient stochastique

Dans le TP1:

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

Algorithme utilisé:

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} E(\boldsymbol{W})$
    (où $\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$)

Quelques mots sur la descente de gradient stochastique

Dans le TP1:

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

Algorithme utilisé:

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} E(\boldsymbol{W})$
    (où $\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$)

Rien de stochastique ! Il s'agit d'une descente de gradient à pas constant.

Une autre écriture

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} E(\boldsymbol{W})$

Une autre écriture

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} E(\boldsymbol{W})$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Pour tout $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$

Une autre écriture

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Pour tout $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$

Une autre écriture

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Pour tout $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Echantillonner $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

Une autre écriture

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Pour tout $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$

- Initialiser $\boldsymbol{W}$ aléatoirement
- Jusqu'a convergence
  - Echantillonner $(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}$

$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \nabla_{\boldsymbol{W}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$

En fait, on a:

$$E(\boldsymbol{W}) \propto \mathbb{E}_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \sim p_{\mathcal{T}}} \left[L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i) \right]$$

En pratique

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

On découpe l'ensemble de données $\mathcal{T}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{T} = \bigsqcup \mathcal{B}$

En pratique

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

On découpe l'ensemble de données $\mathcal{T}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{T} = \bigsqcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch $\mathcal{B}$ de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer $L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i$ pour $(\boldsymbol{y}^i; \boldsymbol{d}^i)$ dans le batch
    - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{B}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
    - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

En pratique

$$E(\boldsymbol{W}) = \sum_{(\boldsymbol{x}^i, \boldsymbol{d}^i) \in \mathcal{T}} L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i)$$

On découpe l'ensemble de données $\mathcal{T}$ en "batchs" $\mathcal{B}$ tels que $\mathcal{T} = \bigsqcup \mathcal{B}$

- Pour chaque époque :
  - Pour chaque batch $\mathcal{B}$ de l'ensemble de données d'entraînement, pris dans un ordre aléatoire
    - Calculer $L(\boldsymbol{W}; \boldsymbol{y}^i; \boldsymbol{d}^i$ pour $(\boldsymbol{y}^i; \boldsymbol{d}^i)$ dans le batch
    - Calculer $\boldsymbol{g} = \sum_{\boldsymbol{x} \in \mathcal{B}} \nabla_{\boldsymbol{W}} \ell(W, \boldsymbol{x})$
    - Faire un pas de SGD :

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \boldsymbol{g}$$

- époque = désigne un passage complet du jeu de données d'entraînement par l'algorithme.
- batch = morceau du jeu de données
- Pour cette raison, input d'un réseau de la forme $[b, c, M, N]$

Questions?

Sources, images courtesy and acknowledgment

Charles Deledalle **V. Lepetit** **L. Masuch**