

# **Bsc Data Science for Responsible Business**

## **Deep Learning Course**

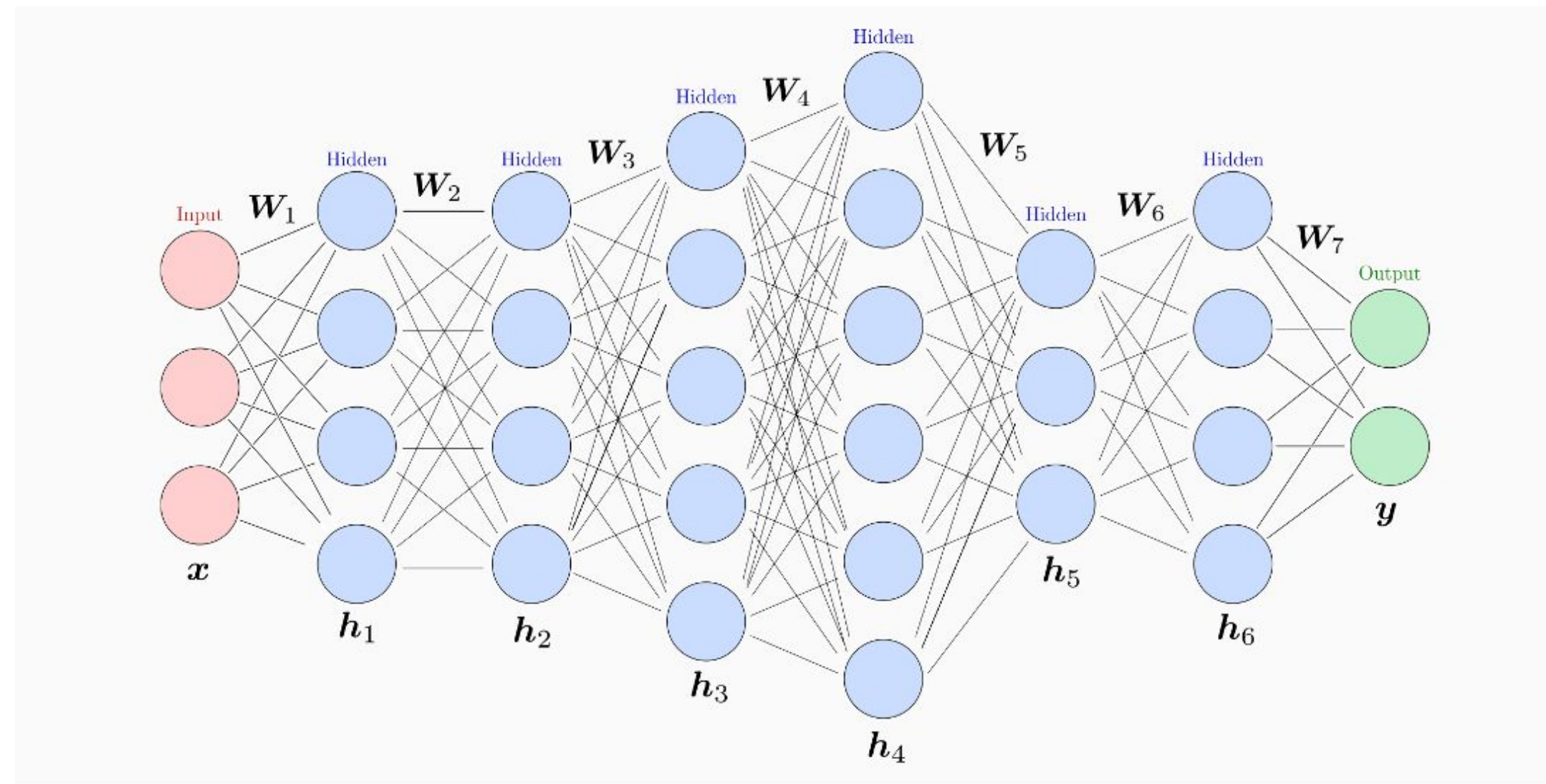
---

### **Neural Networks and Backpropagation**

**Emmanuel Dellandrea** - [emmanuel.dellandrea@ec-lyon.fr](mailto:emmanuel.dellandrea@ec-lyon.fr)

# Introduction to neural networks

**What is it ?** Neural networks are computing systems inspired by the human brain, designed to recognize patterns and learnt from data



## Why use Neural Networks ?

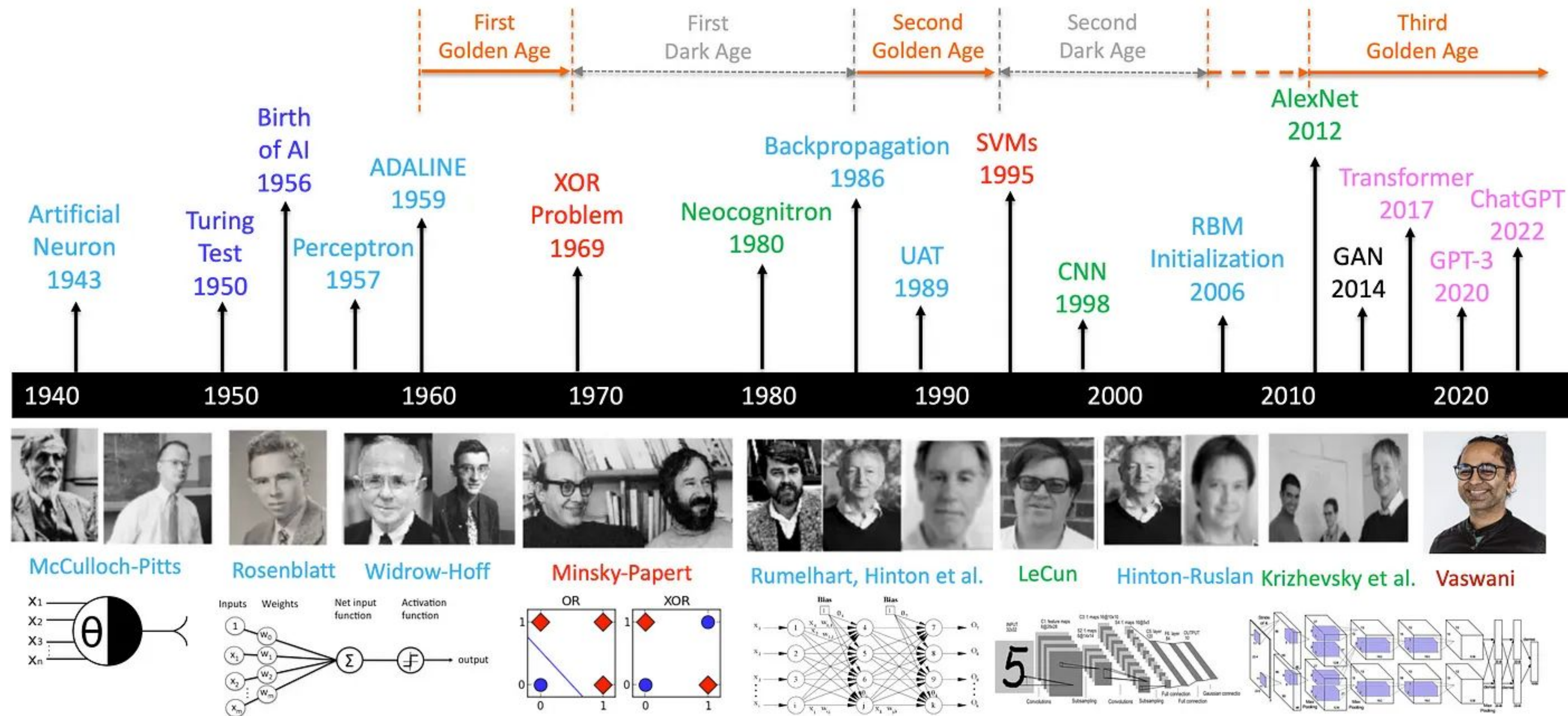
To Solve complex problems like image analysis, speech processing, and natural language understanding

To Handle large amounts of data and find patterns humans might miss

Neural Networks are key components of modern deep learning architectures (Convolutional Neural networks, Transformers, ...)



# History of Neural Networks



From: <https://medium.com/@Impo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b>

# Some Applications of Neural Networks



## Computer Vision

- **Image Recognition:** Object detection, facial recognition, and scene understanding (e.g., identifying people, animals, or objects in photos).
- **Medical Imaging:** Diagnosis using X-rays, MRIs, and CT scans (e.g., detecting tumors).
- **Autonomous Vehicles:** Real-time perception for lane detection, pedestrian recognition, and traffic sign identification.
- **Video Analysis:** Surveillance, action recognition, and video summarization.

## Speech and Audio Processing

- **Speech Recognition:** Converting spoken language to text (e.g., voice assistants, dictation apps).
- **Speech Synthesis:** Generating natural-sounding speech from text.
- **Music Composition:** Generating melodies and harmonies.
- **Audio Analysis:** Noise reduction, sound classification, and audio event detection

## Autonomous Systems

- **Self-Driving Cars:** Perception, planning, and decision-making in autonomous vehicles.
- **Robotics:** Enabling robots to perceive environments and perform tasks intelligently.

## Cybersecurity

- **Threat Detection:** Identifying phishing attacks or malware.
- **Network Security:** Monitoring anomalies in traffic to prevent cyberattacks.

## Energy and Utilities

- **Smart Grid Management:** Optimizing energy distribution.
- **Predictive Maintenance:** Monitoring equipment for faults or wear.

## Natural Language Processing (NLP)

- **Text Translation:** Language translation tools like Google Translate.
- **Chatbots and Virtual Assistants:** Siri, Alexa, chatGPT and customer service bots.
- **Sentiment Analysis:** Understanding opinions in reviews or social media posts.
- **Text Generation:** Models like GPT that create coherent and contextually relevant text

## Healthcare and Biomedicine

- **Drug Discovery:** Predicting molecular properties and identifying new drug candidates.
- **Disease Diagnosis:** Detecting diseases like cancer or Alzheimer's from medical data.
- **Predictive Analytics:** Forecasting patient outcomes or hospital resource needs.
- **Genomics:** Analyzing genetic sequences and predicting protein folding (e.g., AlphaFold).

## Gaming and Entertainment

- **Game AI:** Creating intelligent NPCs and enhancing gameplay dynamics.
- **Content Creation:** Generating characters, textures, or levels in games.

## Scientific Research

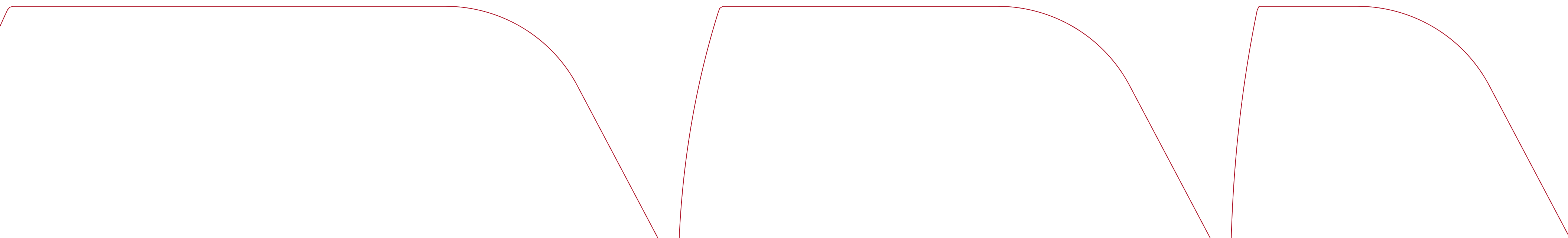
- **Astronomy:** Analyzing celestial images for galaxy classification or object detection.
- **Climate Science:** Predicting weather patterns or analyzing climate change impacts.

## Finance and Economics

- **Fraud Detection:** Identifying fraudulent transactions or activities.
- **Algorithmic Trading:** Making real-time trading decisions based on deep learning models.
- **Credit Scoring:** Predicting loan default risks.



**Some recaps first**



# Recap : Linear Regression

Prediction function:

$$f_{\theta}(x) = \sum_{i=0}^d \theta_i x_i = \theta^T x$$

Loss Function:

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^N (f_{\theta}(x^{(i)}) - y^{(i)})^2$$



# Recap : Logistic Regression

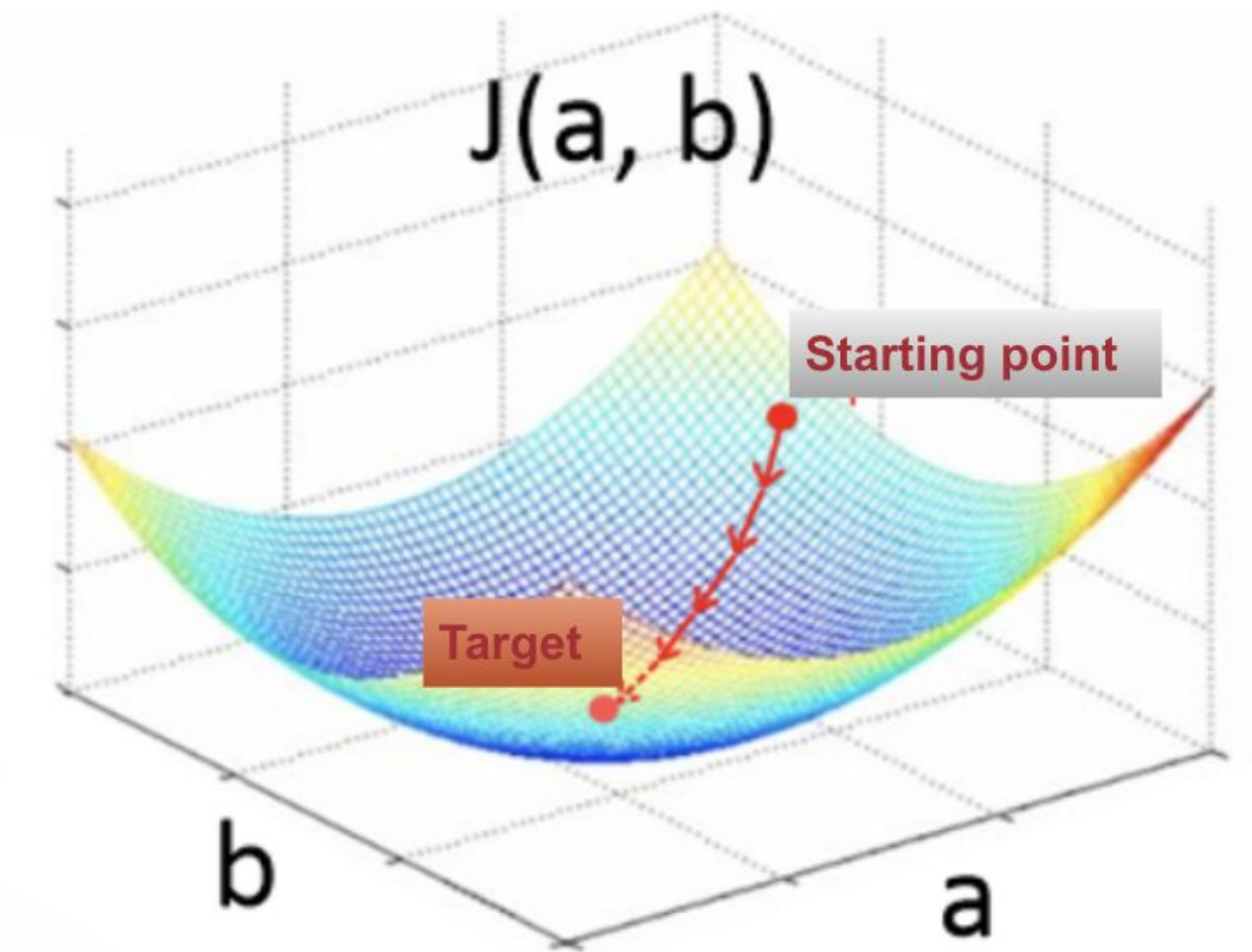
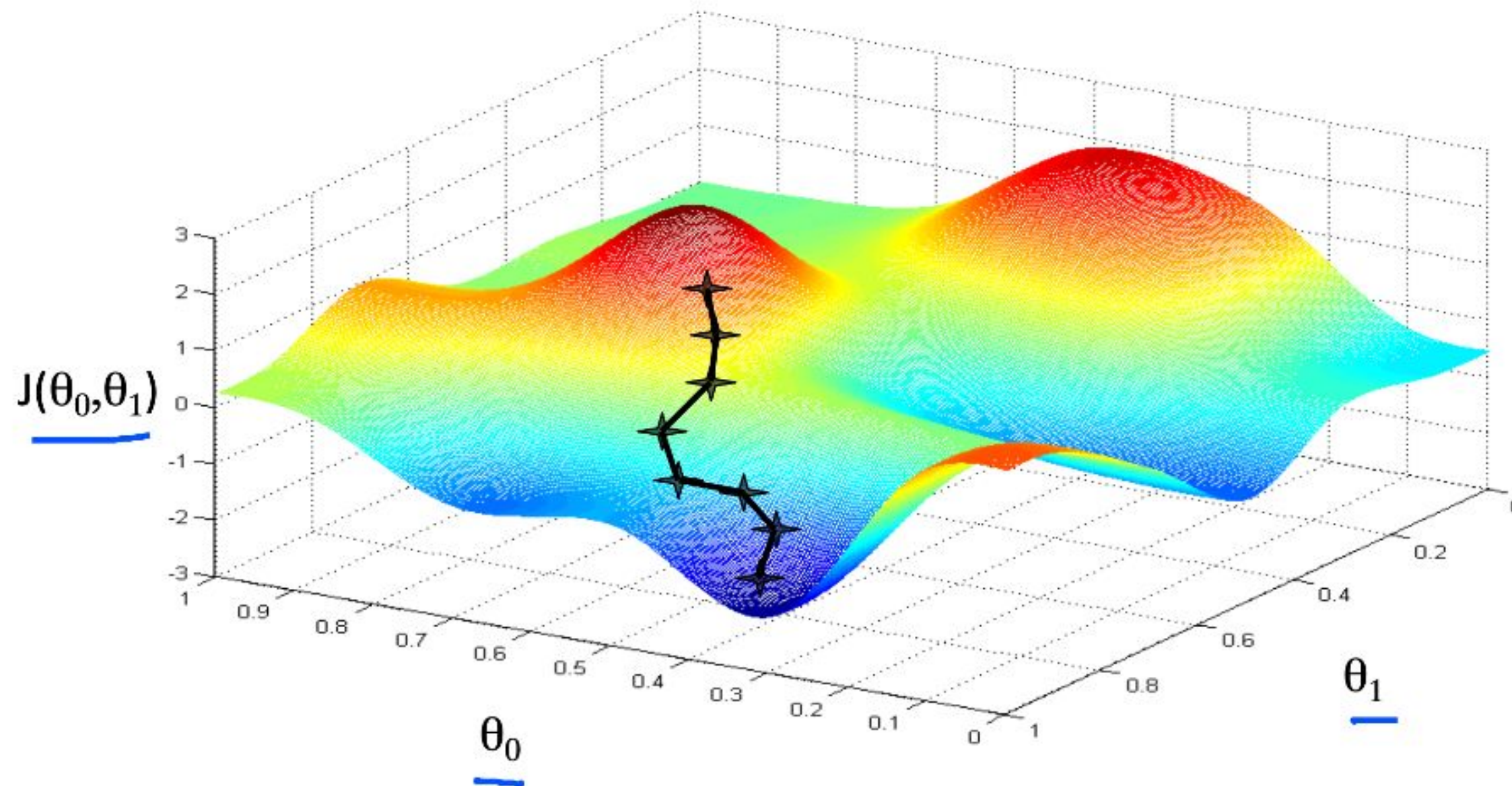
**Prediction function:**

$$f_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

**Loss Function:**

$$J(\theta) = -l(\theta) = -\sum_{i=1}^N y^{(i)} \log(f_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\theta}(x^{(i)}))$$

# Recap : Gradient descent





# Recap : Gradient descent

Iteratively repeat until convergence (simultaneously for each  $j$ )

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

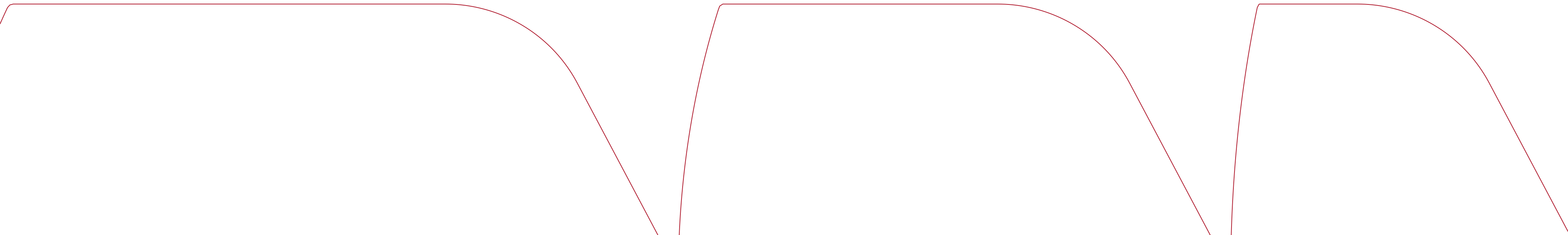
i.e.

$$\theta_j = \theta_j - \alpha \frac{1}{N} \sum_{i=1}^N (f_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

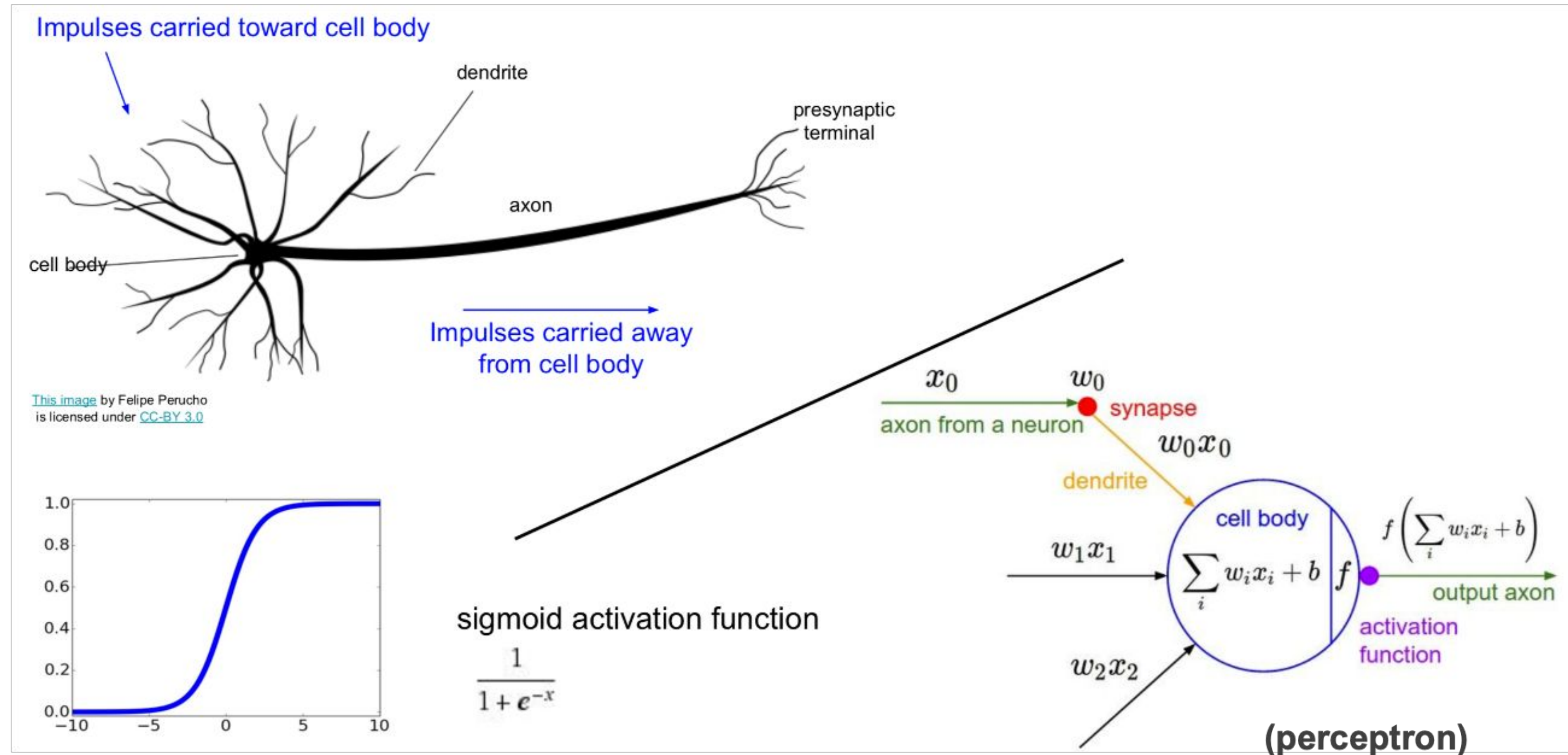
→ same expression for both linear and logistic regression, only  $f_{\theta}(x)$  is different



# From biological neurons to artificial neurons

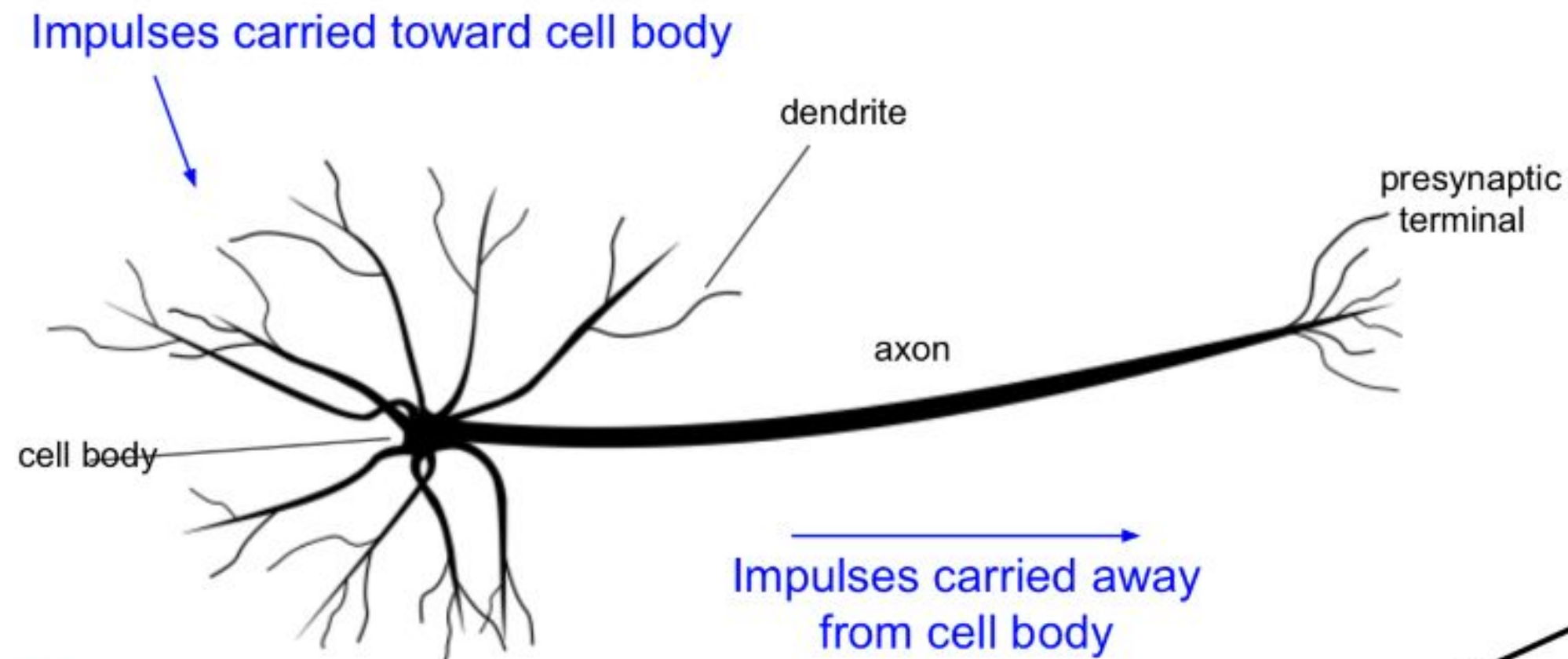


# Biological neuron vs artificial neuron



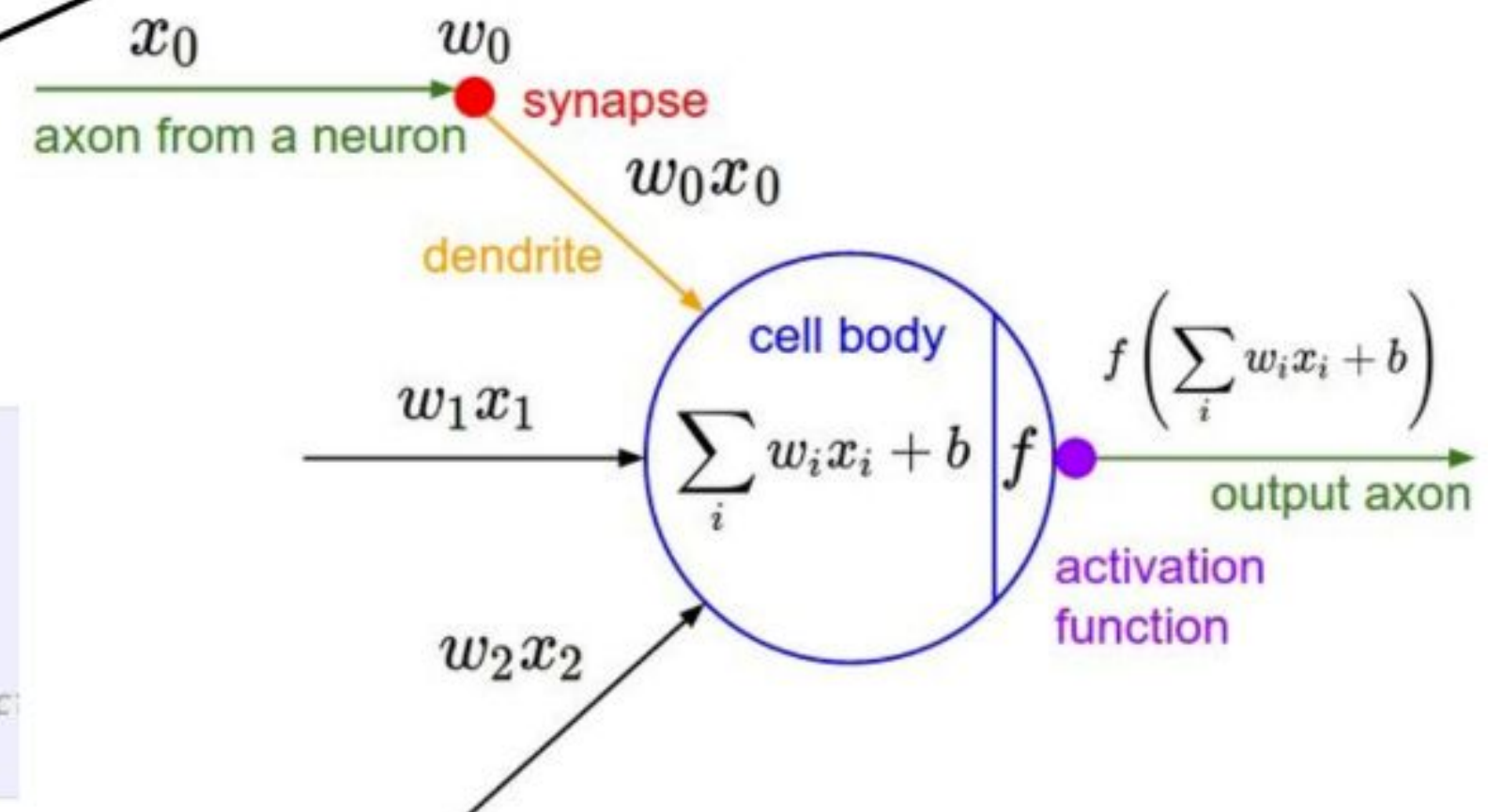


# Biological neuron vs artificial neuron



This image by Felipe Perucho  
is licensed under [CC-BY 3.0](https://creativecommons.org/licenses/by/3.0/)

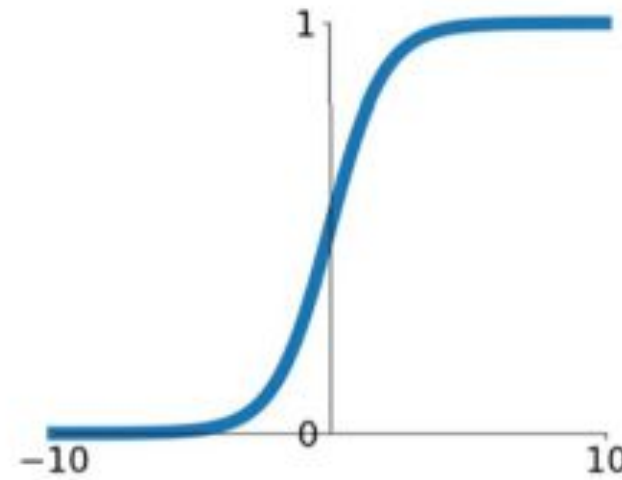
```
class Neuron:
    # ...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation func
        return firing_rate
```



# Activation functions

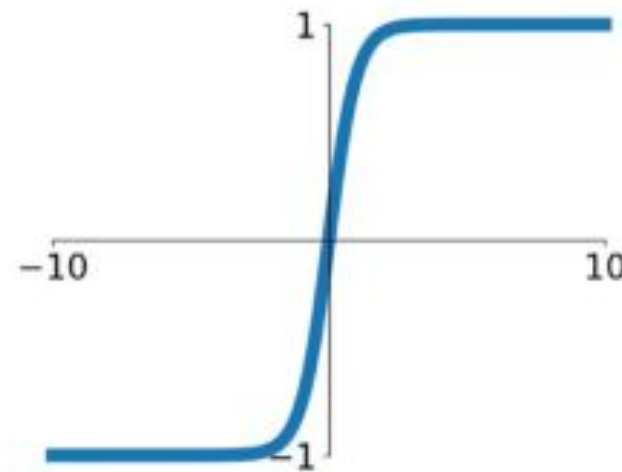
## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



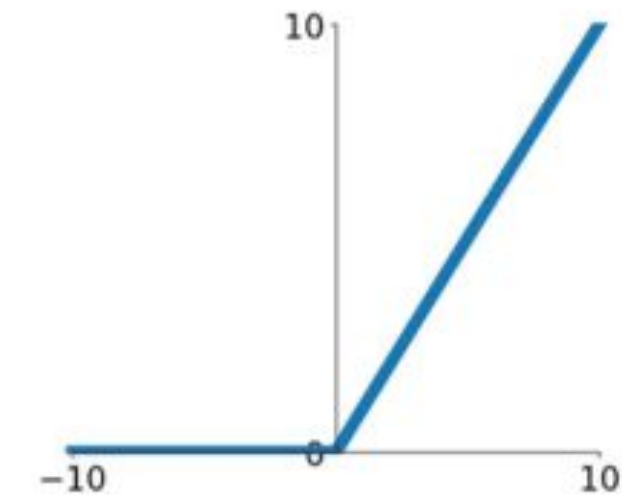
## tanh

$$\tanh(x)$$



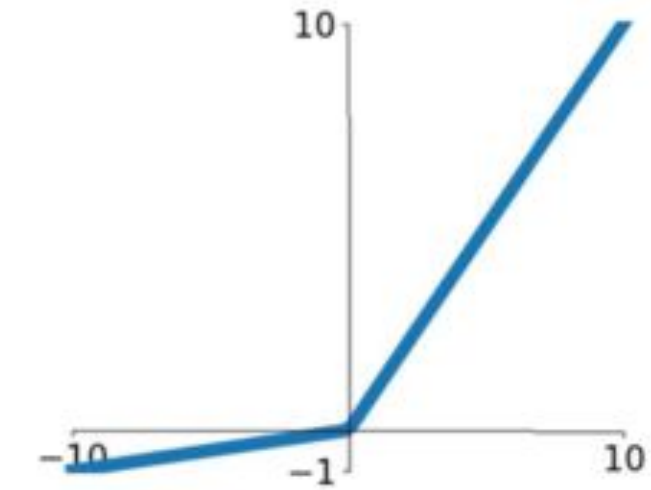
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$

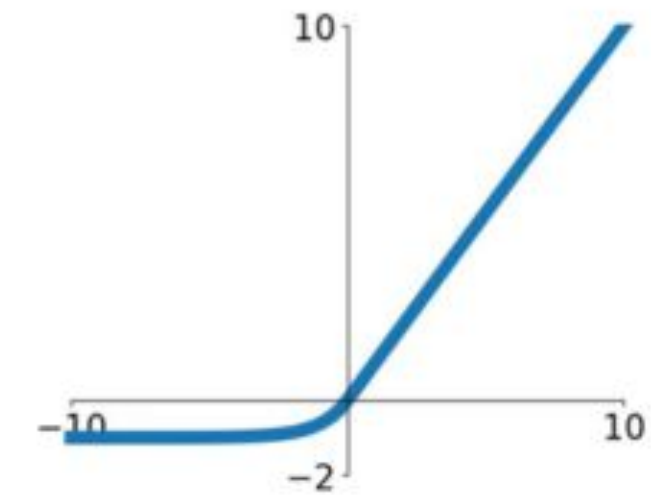


## Maxout

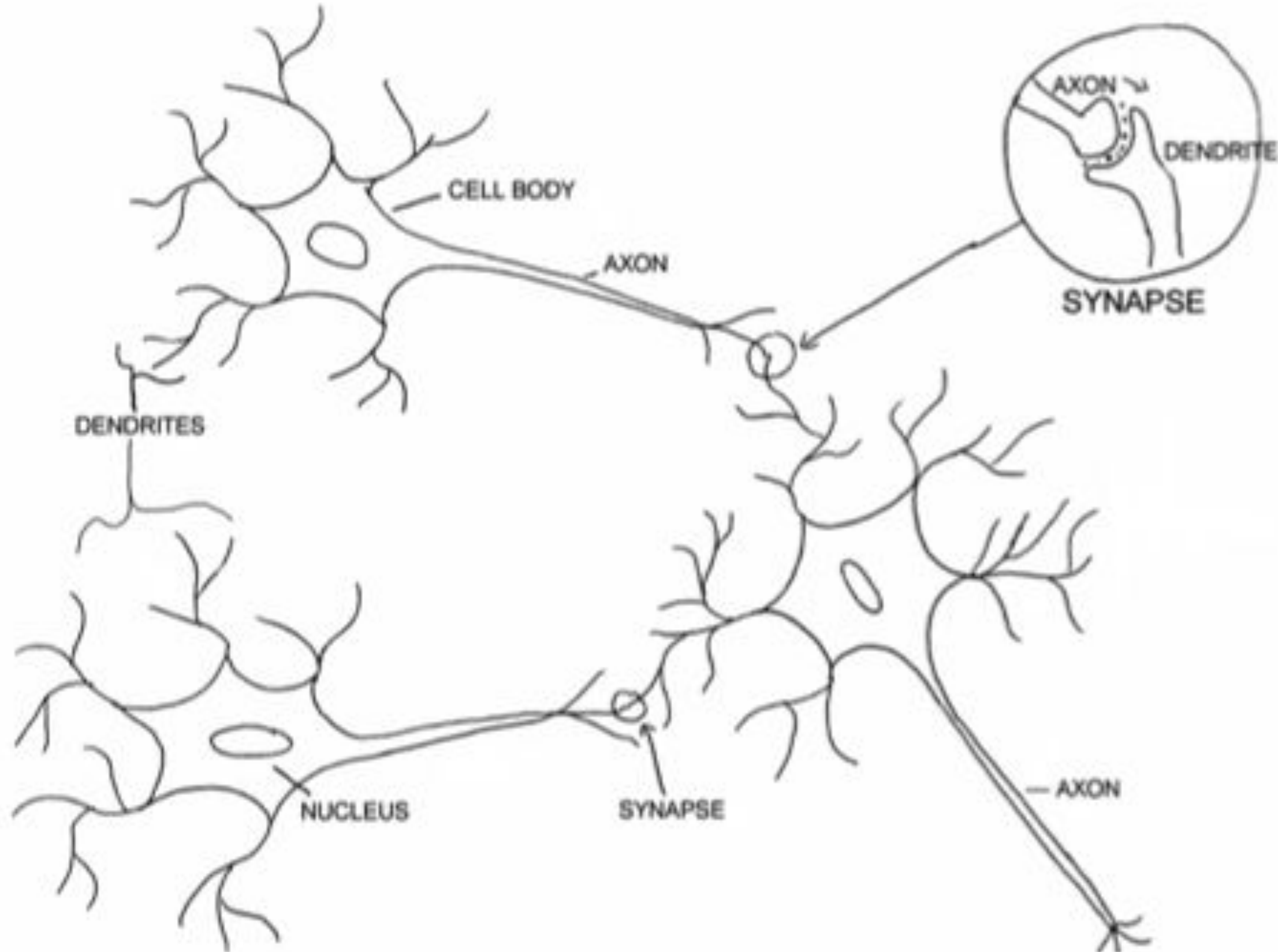
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



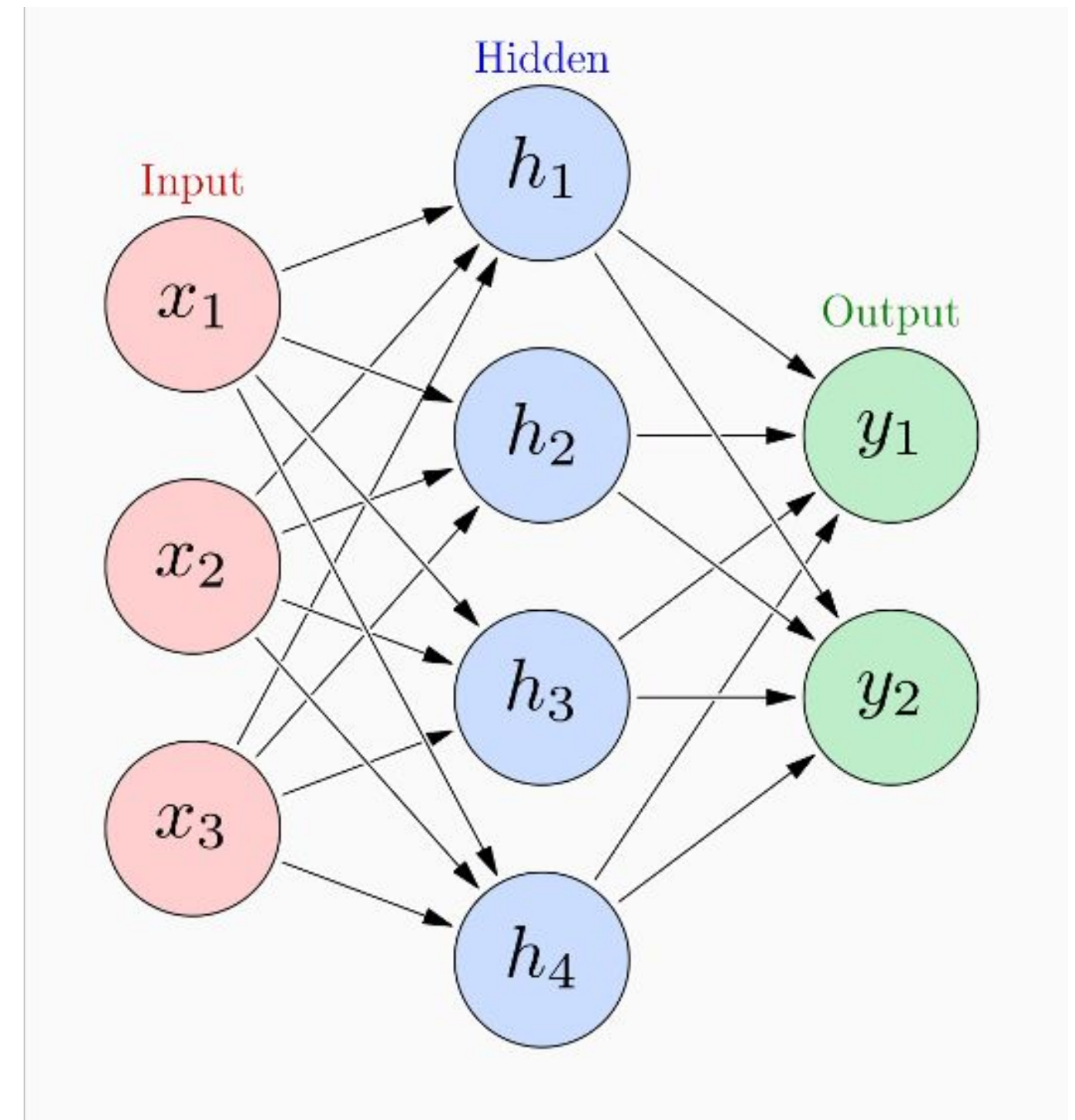
# Biological neural networks





# Artificial neural networks

- Interconnection of neurons
- Organization of neurons into layers
  - Input Layer
  - Hidden Layer(s)
  - Output Layer
- Each neuron is connected to all the neurons in the previous layer and all those in the next layer

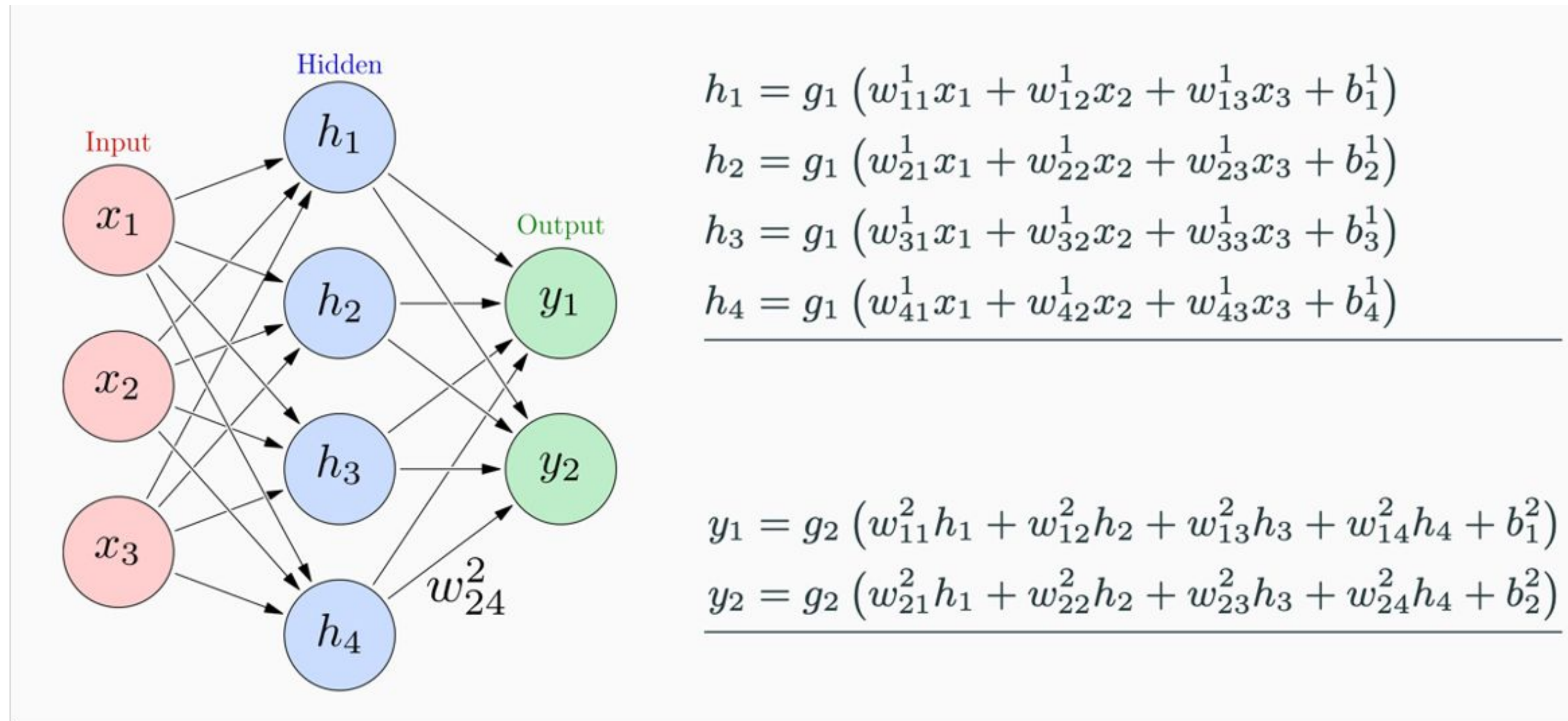




# Neural networks and backpropagation

Some materials are borrowed from Bruno Galerne's course on Artificial Neural Networks

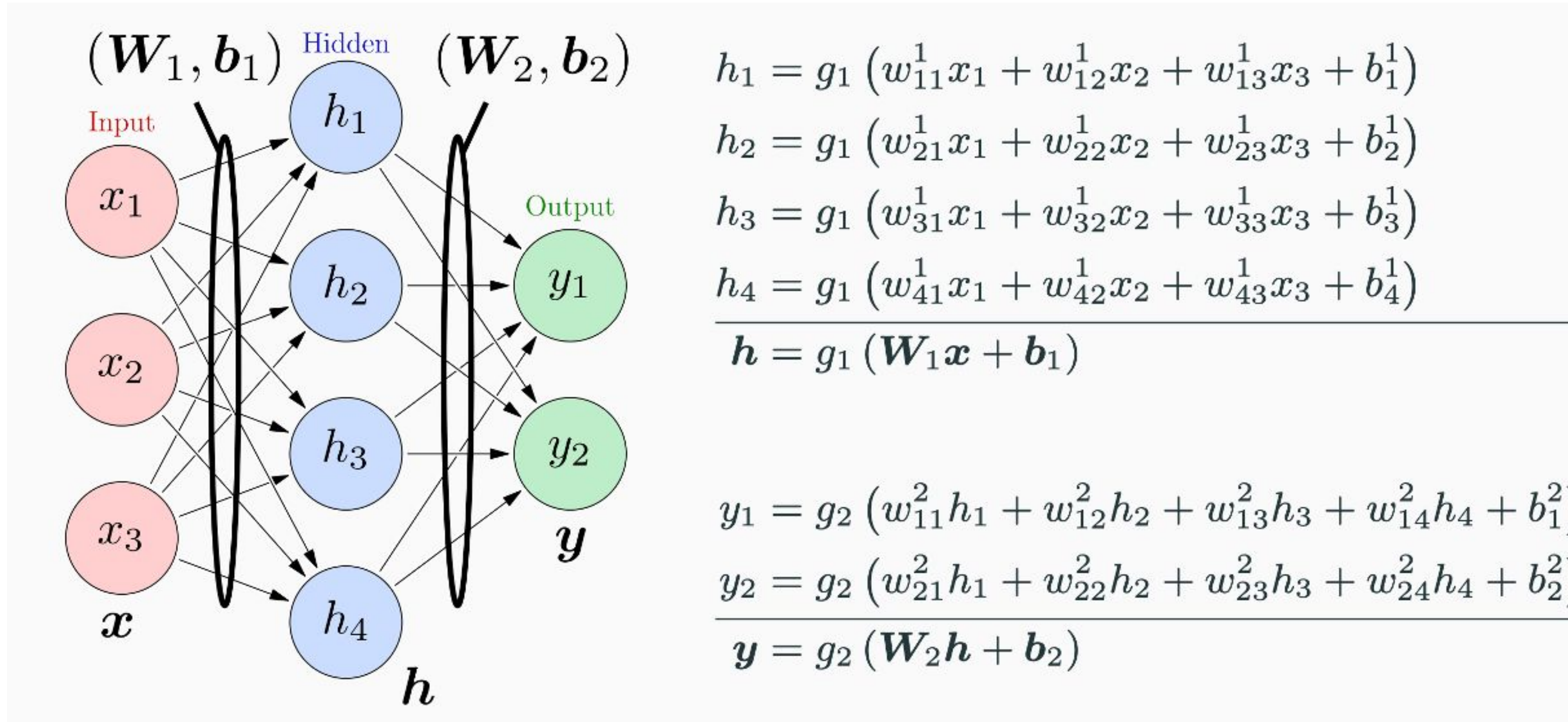
# Neural network



- $\omega_{ij}^k$ : synaptic weights between the previous neuron  $j$  and the next neuron  $i$  in layer  $k$
- $g_k$ : activation functions applied to each input potential (linear unit)

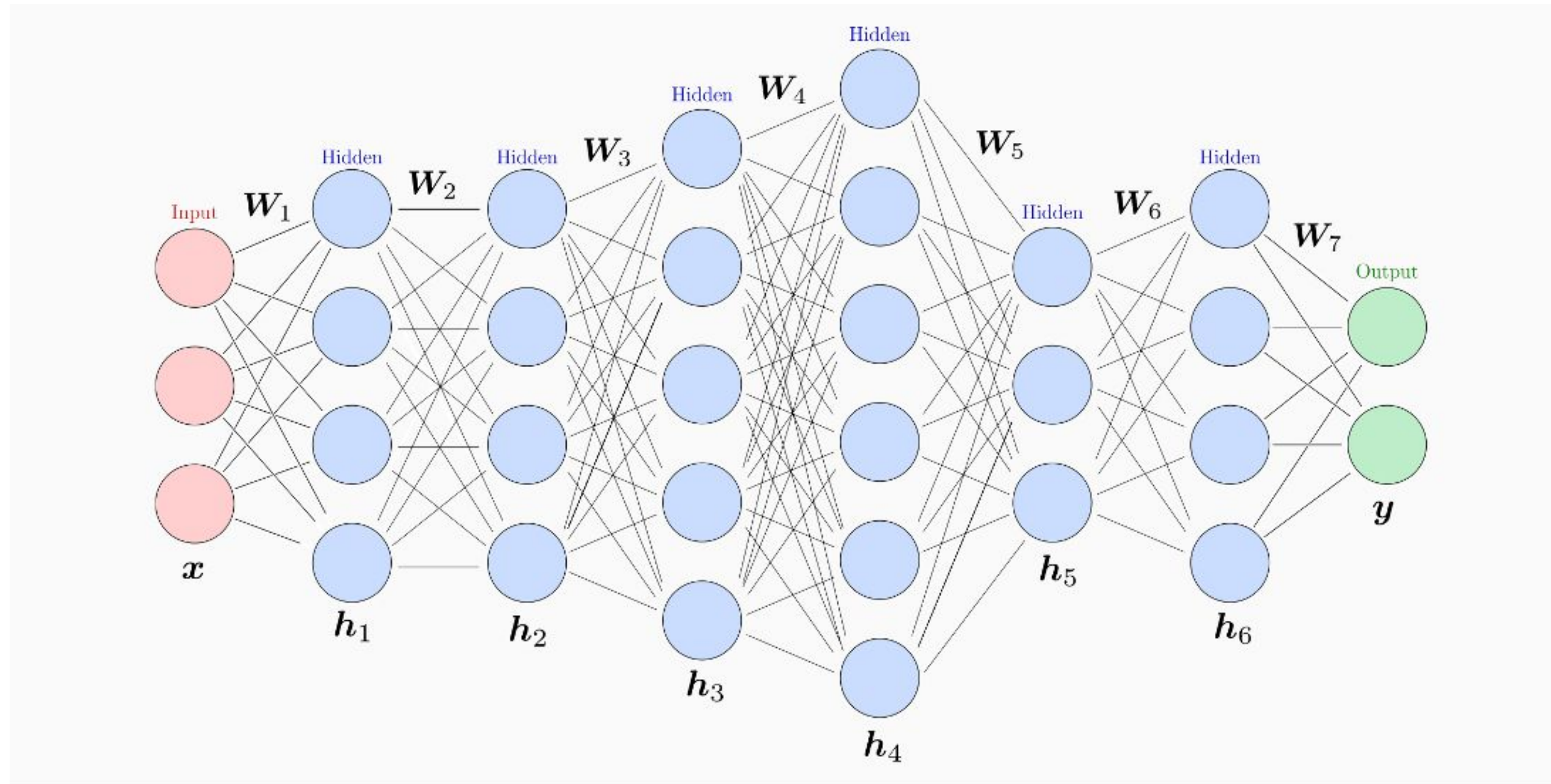


# Neural network



The weight matrices  $W_k$  and the bias vectors  $b_k$  are learnt from the training data

# Neural network

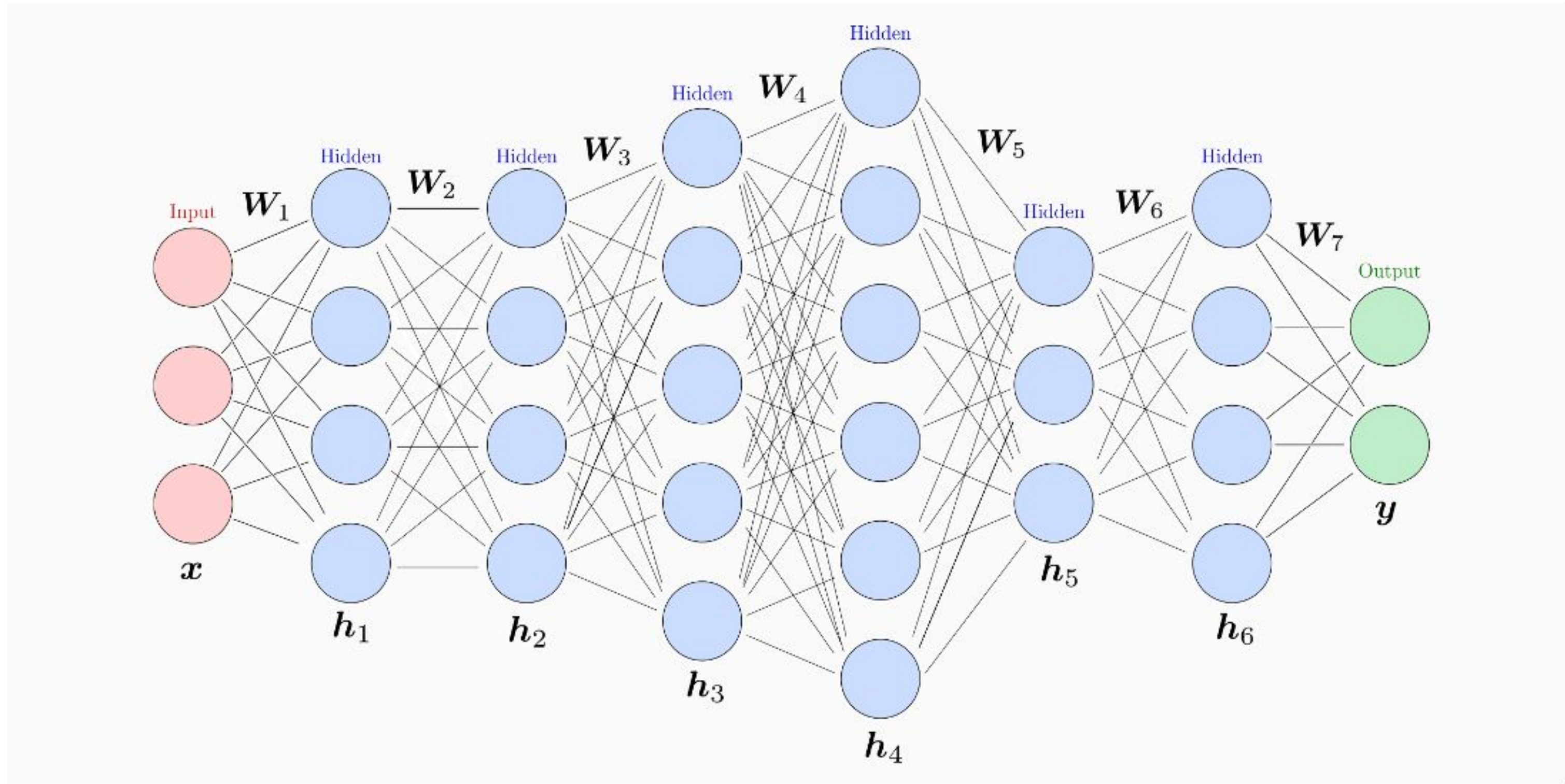


- If more than one hidden layer: "deep" network
- Each layer can have a different number of neurons
- Activation functions can vary between layers



# Typical architectures

## Regression



- Hidden layers :  $\text{ReLU}(a) = \max(a, 0)$
- Linear output layer :  $g(a) = a$



# Typical architectures

## Regression

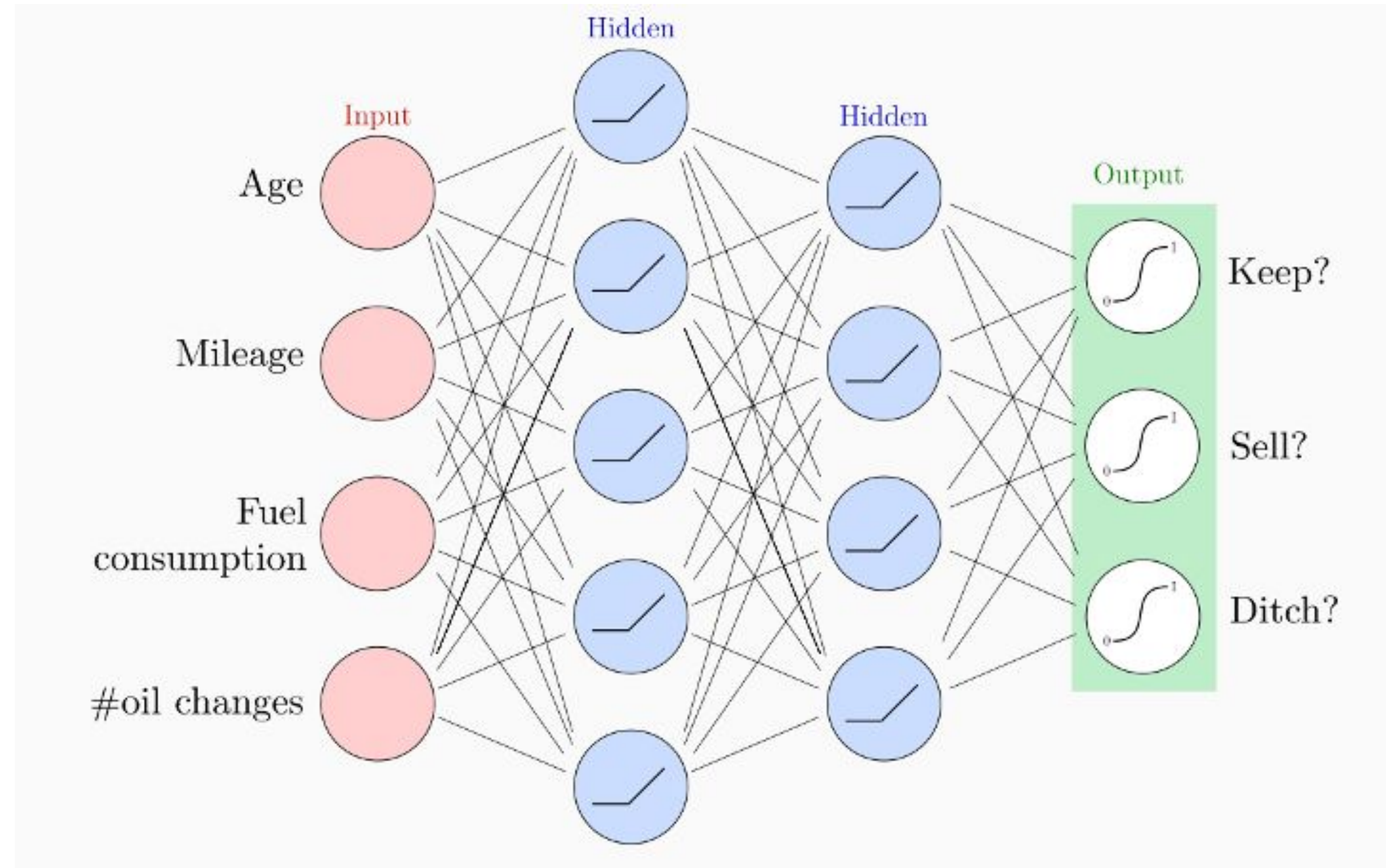
- Loss function ( Mean Square Error)

$$E(\mathbf{W}) = \sum_{i=1}^N \|\mathbf{y}^i - \mathbf{d}^i\|_2^2 = \sum_{i=1}^N \|f(\mathbf{x}^i; \mathbf{W}) - \mathbf{d}^i\|_2^2$$

- Minimize  $E(W)$  by gradient descent

# Typical architectures

## Multiclass classification



- Hidden layers :  $\text{ReLU}(a) = \max(a, 0)$
- Output layer : softmax (like a probability)
- Final decision : class of the output neuron having the highest probability

$$\text{softmax}(\mathbf{a})_k = \frac{\exp(a_k)}{\sum_{\ell=1}^K \exp(a_\ell)}$$

# Typical architectures

## Multiclass classification

- Loss function : Cross-entropy

$$E(\mathbf{W}) = - \sum_{i=1}^N \sum_{k=1}^K d_k^i \log y_k^i \quad \text{with} \quad \mathbf{y}^i = f(\mathbf{x}^i; \mathbf{W}) = \text{softmax}(\mathbf{a}) \in (0, 1)^K$$

- Minimize  $E(W)$  by gradient descent

Parameters of the network :

$$\mathbf{W} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \dots, \mathbf{W}_L, \mathbf{b}_L)$$

Training : minimize the loss function  $E(\mathbf{W})$  by gradient descent

**Objective:**  $\min_{\mathbf{W}} E(\mathbf{W})$  where  $E(\mathbf{W}) = \sum_{(\mathbf{x}^i, \mathbf{d}^i) \in \mathcal{T}} L(\mathbf{y}^i; \mathbf{d}^i)$

$$\Rightarrow \nabla E(\mathbf{W}) = \left( \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_1} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_1} \quad \dots \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}_L} \quad \frac{\partial E(\mathbf{W})}{\partial \mathbf{b}_L} \right)^T = 0$$

Gradients :

$$\nabla_{\mathbf{W}_k} E(\mathbf{W}) \quad \text{and} \quad \nabla_{\mathbf{b}_k} E(\mathbf{W})$$



Gradient descent :

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

How to compute

$$\frac{\partial E(\mathbf{W}^t)}{\partial w_{i,j}^k}$$

??

→ By retropropagation

## Standard Loss functions

**Regression** : Mean Square Error

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

**Multiclass classification** : Cross-Entropy (with a softmax output layer)

→ Cross-entropy measures how "surprised" the model is by the true labels.

- If the model assigns a high probability to the true class, the surprise (and loss) is low.
- If the model assigns a low probability, the surprise (and loss) is high.
- The goal is to train the model to make predictions that align closely with the true labels, thereby reducing its "surprise" and minimizing the cross-entropy loss.

$$\mathbf{d}^i \in \{1, \dots, K\}, \text{ coded by } \mathbf{d}^i \in \{0, 1\}^K$$

$$E(\mathbf{W}) = - \sum_{(\mathbf{x}^i, \mathbf{d}^i)} \sum_{k=1}^K d_k^i \log y_k^i \quad \text{with} \quad \mathbf{y}^i = f(\mathbf{x}^i; \mathbf{W}) = \text{softmax}(\mathbf{a}^i) \in (0, 1)^K$$

Loss function expression :

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

Gradient expression :

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

## Gradient of $L(\mathbf{y}; \mathbf{d})$

with respect to  $\mathbf{y}$  in case of regression

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

with respect to  $\mathbf{a}$  in case of classification

$$L(\mathbf{y}; \mathbf{d}) = - \sum_{k=1}^K d_k^i \log y_k^i \quad \Rightarrow \quad \nabla_{\mathbf{a}} L(\mathbf{y}; \mathbf{d}) = \mathbf{y} - \mathbf{d}$$

with  $\mathbf{y} = \text{softmax}(\mathbf{a})$



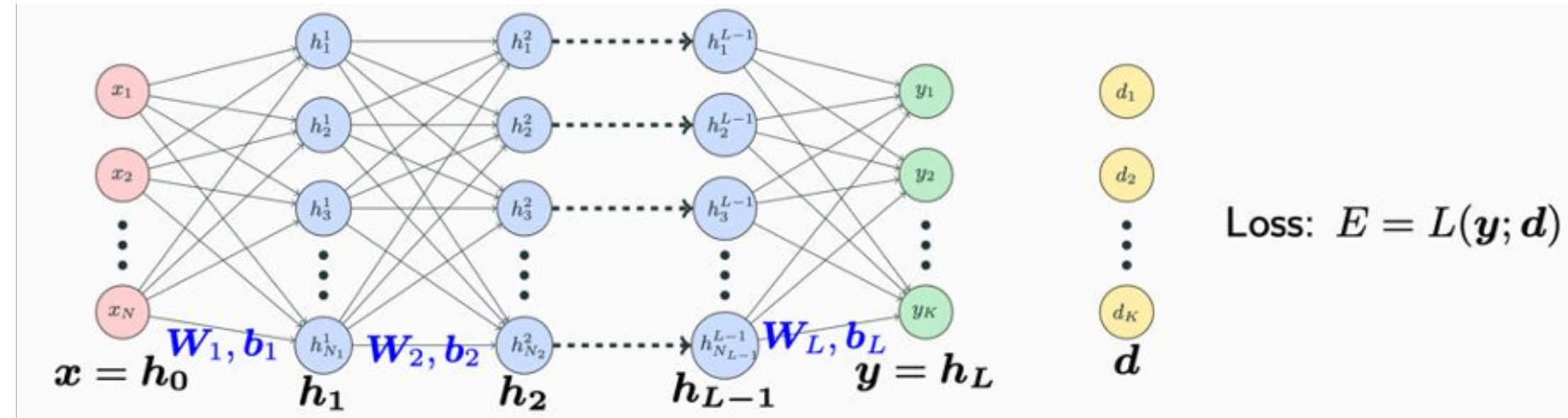
Thus, we know how to compute the gradient  $L(y;d)$  according to  $y$  (or  $a$ ) (output layer)

How to compute the gradients for the other layers ??

$$\nabla_{w_k} L(y; d) \quad \text{and} \quad \nabla_{b_k} L(y; d) \quad \text{for } k = 0, \dots, L$$

→ By backward propagation of the loss error : Backpropagation

# Backpropagation



## Forward pass

Initialization:

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

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

Linear unit:

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

Componentwise non-linear activation:

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

**end**

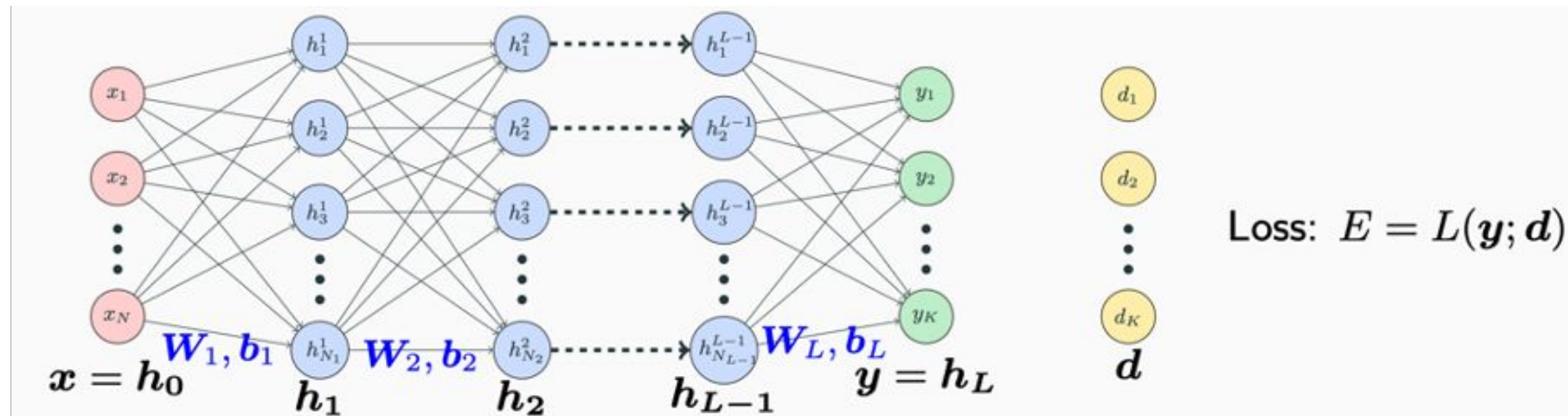
Output layer:

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

Compute loss:

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

# Backpropagation



## Forward pass

Initialization:

$$h_0 = x$$

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

Linear unit:

$$a_k = W_k h_{k-1} + b_k$$

Componentwise non-linear activation:

$$h_k = g_k(a_k)$$

**end**

Output layer:

$$y = h_L$$

Compute loss:

$$E = L(y; d)$$

## Backward pass

**Goal:** Compute the gradient with respect to all parameters

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

for all

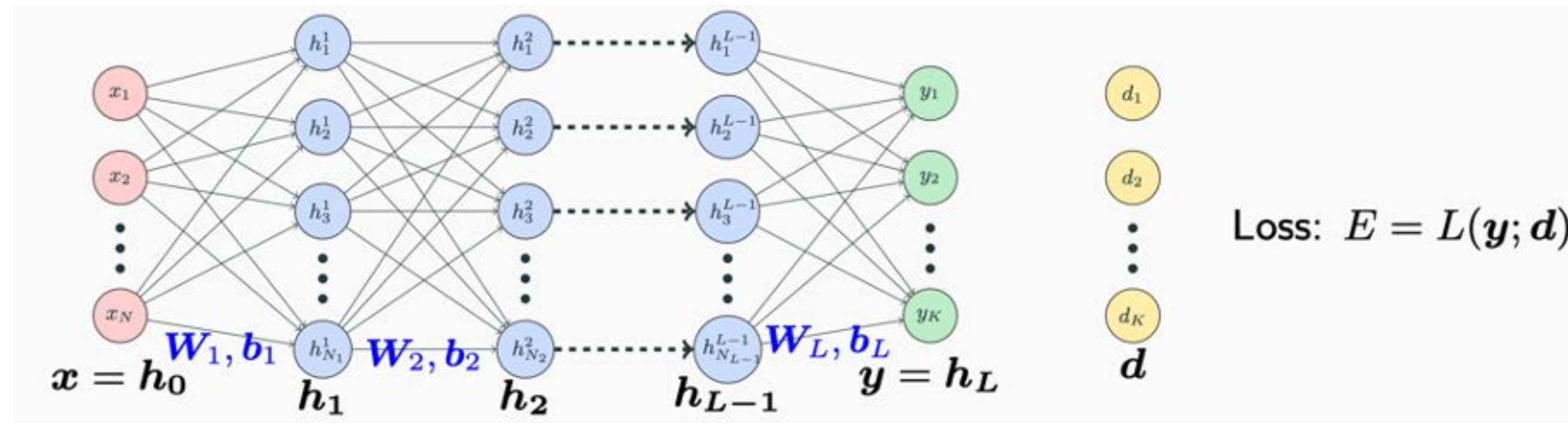
$$k \in \{1, \dots, L\},$$

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

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



# Backpropagation



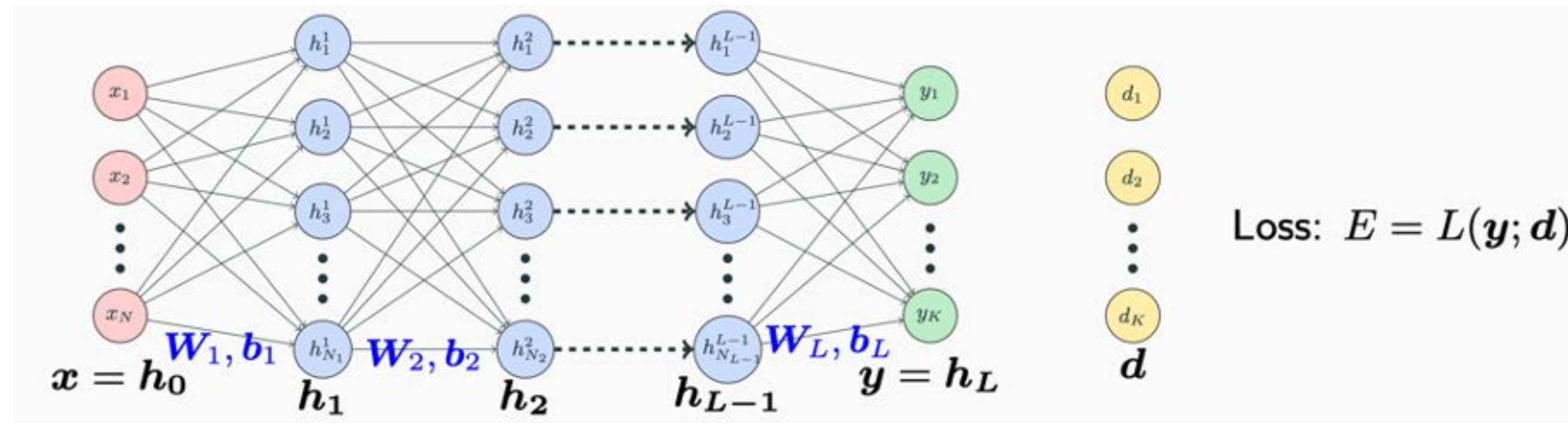
## Backward pass

We know how to compute the loss function and its gradient according to the output layer

$$\nabla_{h_L} E = \nabla L(y; d)$$



# Backpropagation



Gradient with respect to the last linear unit  $a_L$

$$h_L = g_L(a_L)$$

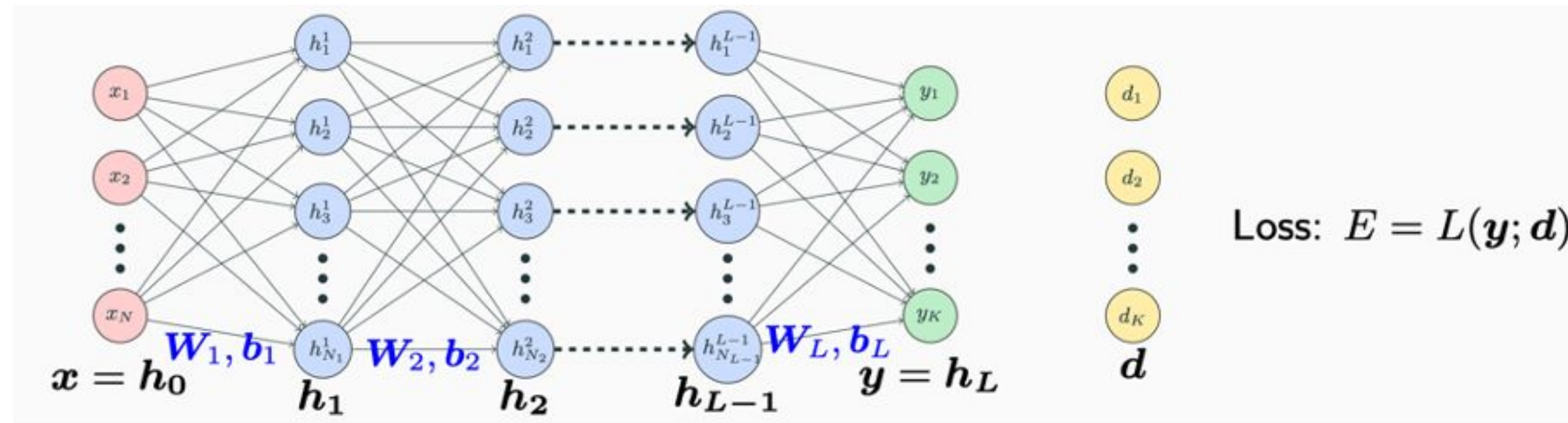
For all  $i \in \{1, \dots, N_L\}, h_i^L = g_L(a_i^L)$

Chaining of the partial derivatives :

$$\frac{\partial E}{\partial a_i^L} = \frac{\partial E}{\partial h_i^L} \frac{\partial h_i^L}{\partial a_i^L} = [\nabla_{h_L} E]_i g'_L(a_i^L)$$

Matrix formulation :  $\nabla_{a_L} E = \nabla_{h_L} E \odot g'_L(a_L)$  where  $\odot$  stands for the element-wise product between two matrices

# Backpropagation



Gradient with respect to the bias  $b_L$  of the last last linear unit

$$a_L = W_L h_{L-1} + b_L$$

For all  $i \in \{1, \dots, N_L\}$ ,  $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$

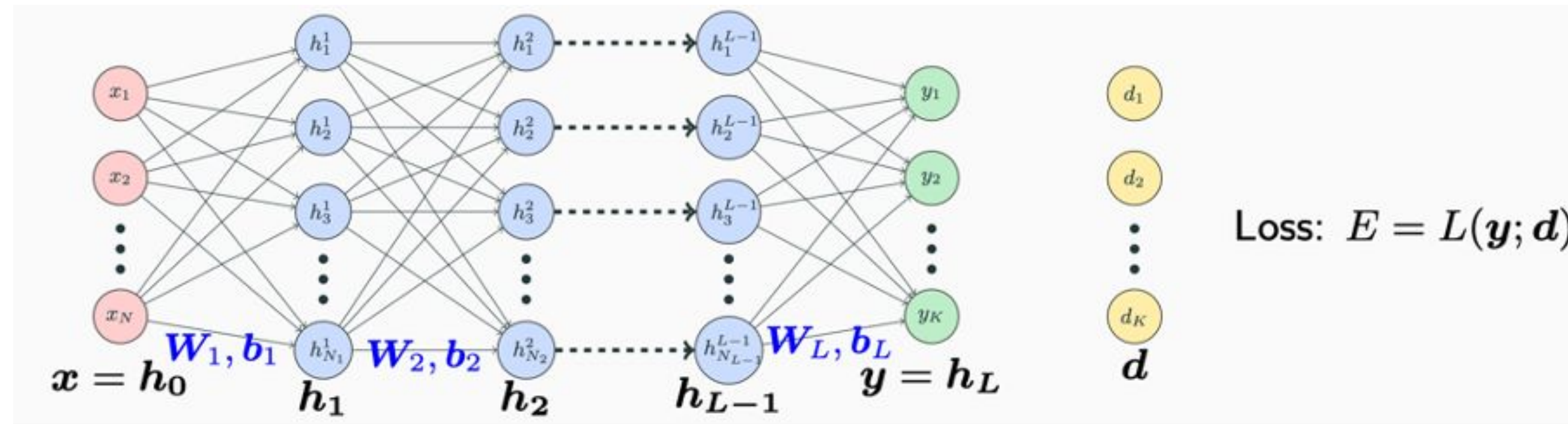
Chaining of the partial derivatives :

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

Matrix formulation :  $\nabla_{b_L} E = \nabla_{a_L} E$



# Backpropagation



Gradient with respect to the weights  $W_L$  of the last linear unit

$$a_L = W_L h_{L-1} + b_L$$

For all  $i \in \{1, \dots, N_L\}$ ,  $a_i^L = \sum_{j=1}^{N_{L-1}} w_{i,j}^L h_j^{L-1} + b_i^L$

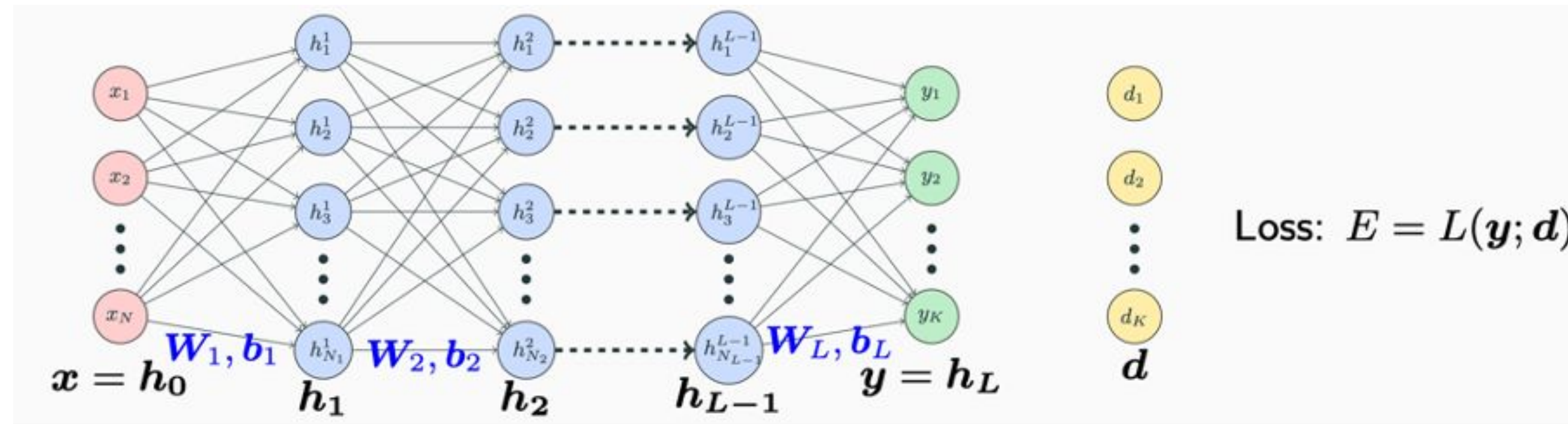
Chaining of the partial derivatives :

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

Matrix formulation :  $\nabla_{W_L} E = \nabla_{a_L} E h_{L-1}^T$



# Backpropagation



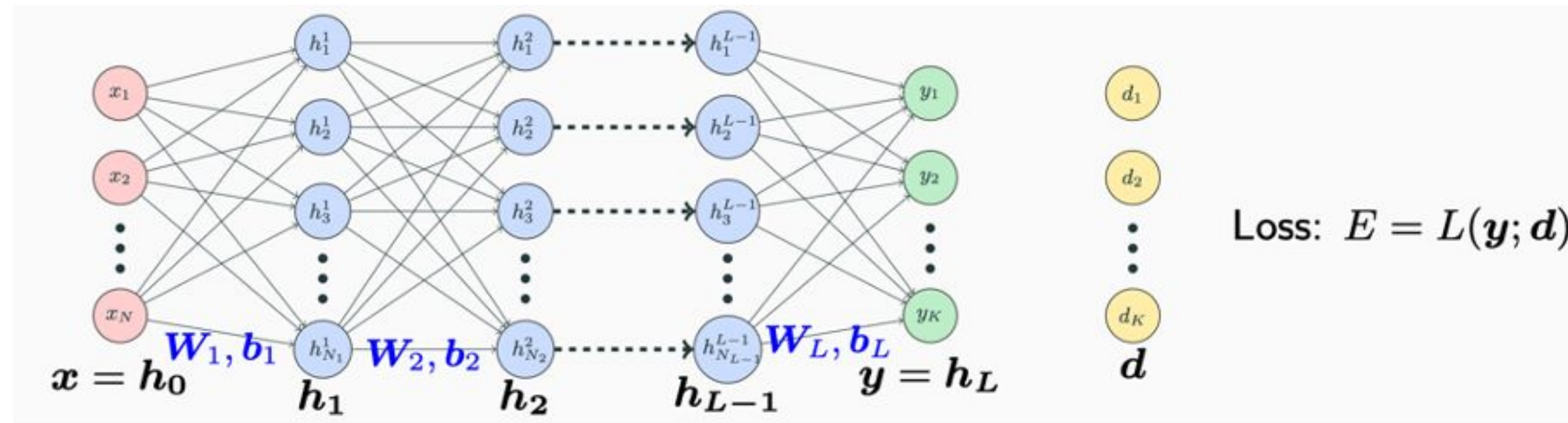
## Gradients for the parameters of the output layer

Knowing the gradient with respect to the output layer  $\nabla_{h_L} E$

We can compute

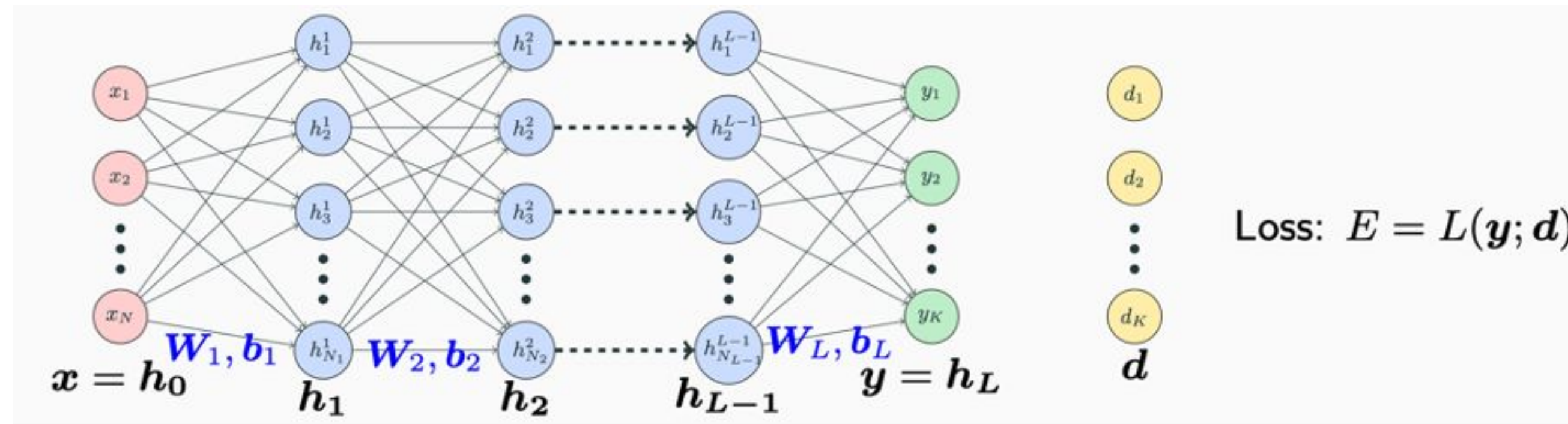
- $\nabla_{a_L} E = \nabla_{h_L} E \odot g'_L(a_L)$
- $\nabla_{b_L} E = \nabla_{a_L} E$
- $\nabla_{W_L} E = \nabla_{a_L} E h_{L-1}^T$

# Backpropagation



Now, how to compute the gradients for the parameters of the layer  $L-1$  just before the output layer (penultimate layer) ???

# Backpropagation



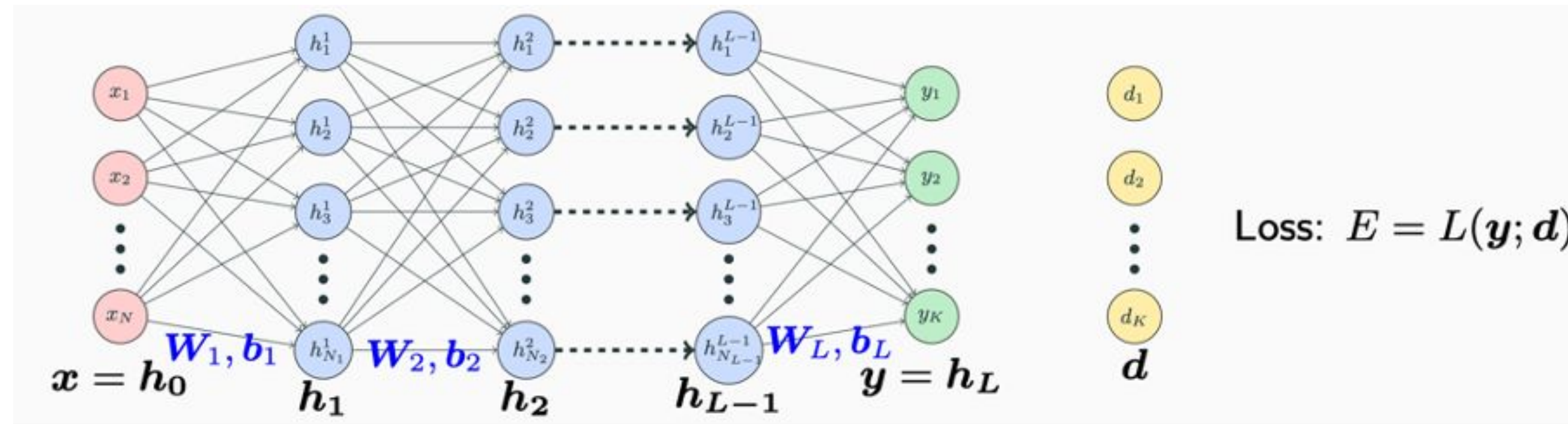
Now, how to compute the gradients for the parameters of the layer  $L-1$  just before the output layer (penultimate layer) ???

→ We need to have the gradient with respect to the penultimate hidden layer  $h_{L-1}$  and then apply the same formulas

$$\nabla_{h_{L-1}} E = ?$$



# Backpropagation



Gradient with respect to the penultimate hidden layer  $h_{L-1}$

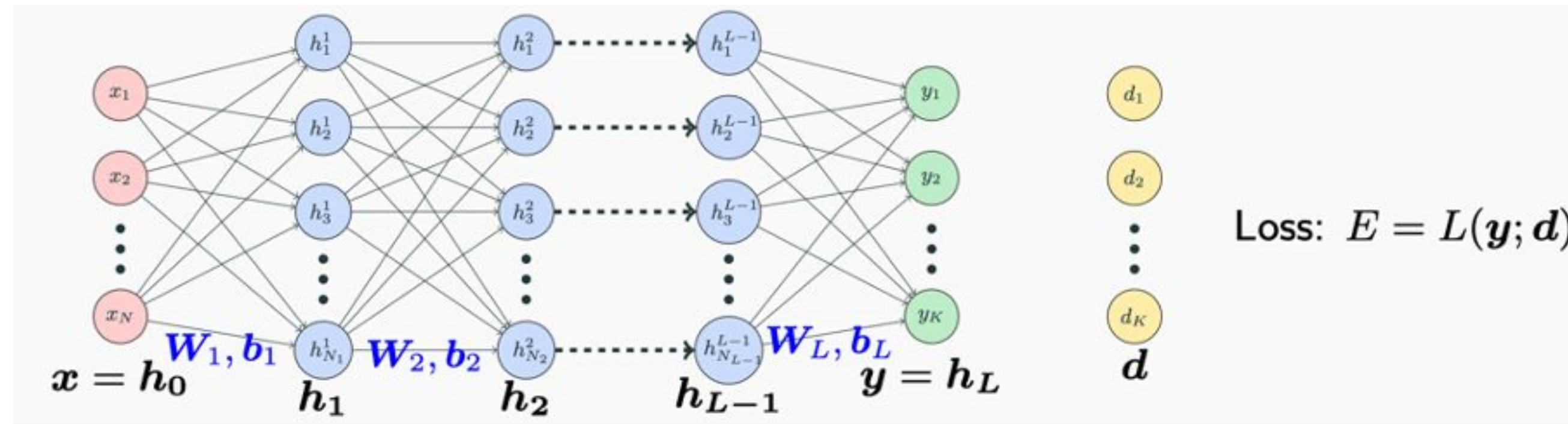
We have for all

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

And we want to compute

$$\frac{\partial E}{\partial h_j^{L-1}}$$

# Backpropagation



Gradient with respect to the penultimate hidden layer  $h_{L-1}$

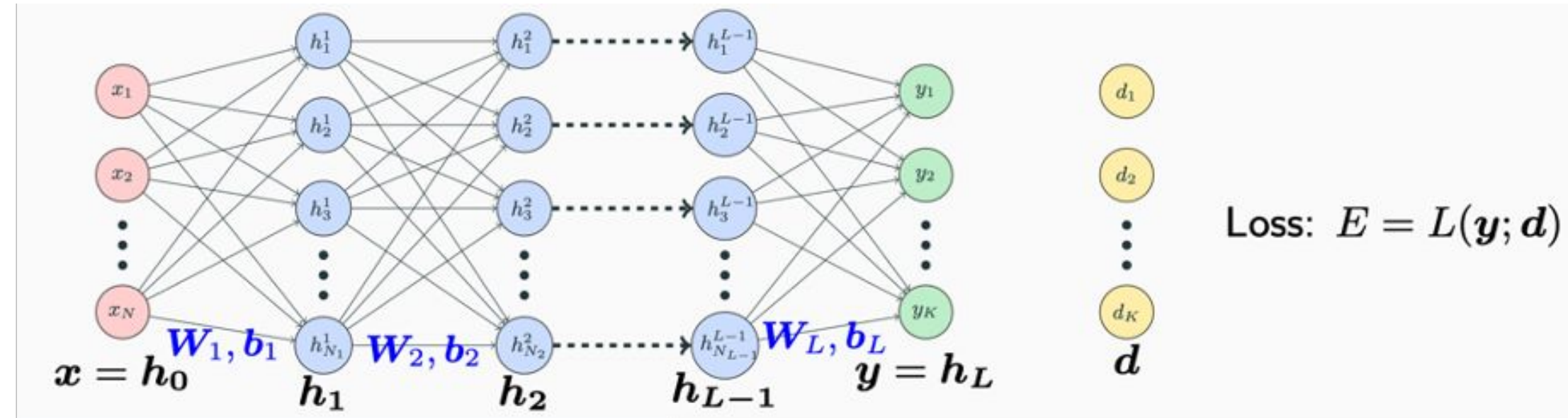
Derivative calculation rule for compositions of affine functions : for  $\varphi(x) = f(Ax + b)$  we have  $\nabla \varphi(x) = A^T \nabla f(Ax + b)$

Which gives :

$$\begin{array}{ccccc} \mathbb{R}^{N_{L-1}} & \rightarrow & \mathbb{R}^{N_L} & \rightarrow & \mathbb{R} \\ h_{L-1} & \mapsto & a_L = W_L h_{L-1} + b_L & \mapsto & E \end{array}$$

Matrix formulation :  $\nabla_{h_{L-1}} E = W_L^T \nabla_{a_L} E$

# Backpropagation



## Forward pass

Initialization:

$$h_0 = x$$

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

Linear unit:

$$a_k = \mathbf{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k$$

Componentwise non-linear activation:

$$\mathbf{h}_k = g_k(a_k)$$

**end**

Output layer:

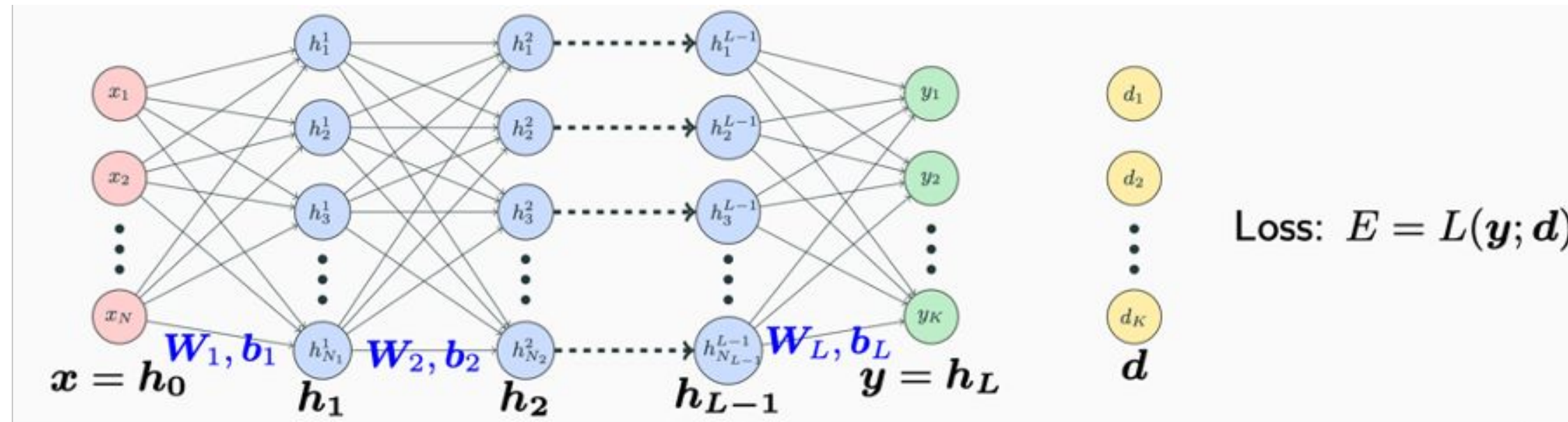
$$\mathbf{y} = \mathbf{h}_L$$

Compute loss:

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



# Backpropagation



## Forward pass

Initialization:

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

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

Linear unit:

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

Componentwise non-linear activation:

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

**end**

Output layer:

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

Compute loss:

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

## Backward pass

Initialization: Gradient of output layer:

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

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

Componentwise gain of error:

$$\delta_k = \nabla_{\mathbf{a}_k} E = \nabla_{\mathbf{h}_k} E \odot g'_k(\mathbf{a}_k)$$

Gradient of layer bias:

$$\nabla_{\mathbf{b}_k} E = \delta_k$$

Gradient of weights:

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

Gradient of previous hidden layer:

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

**end**

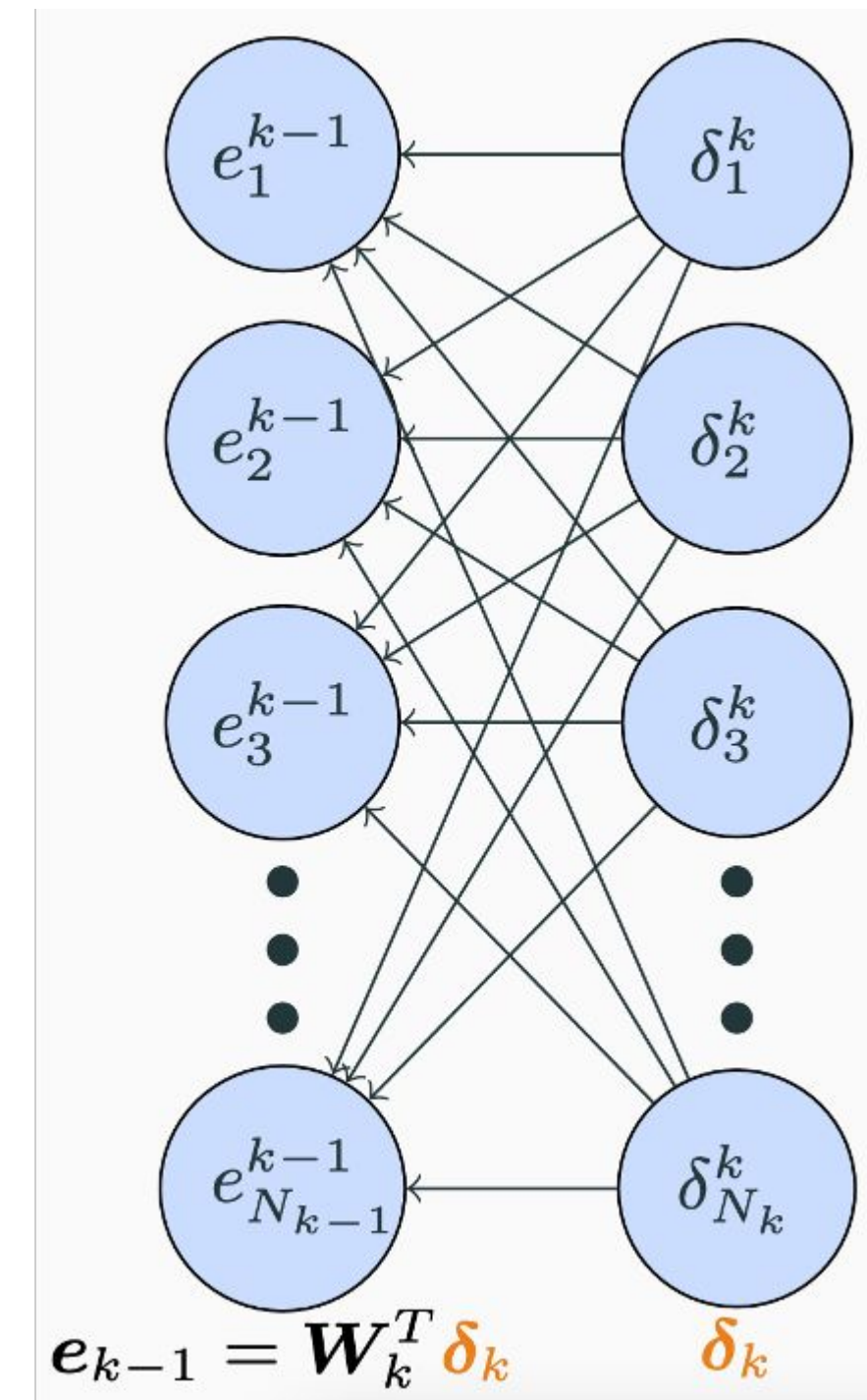
# Backpropagation

Gradient of the previous hidden layer

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

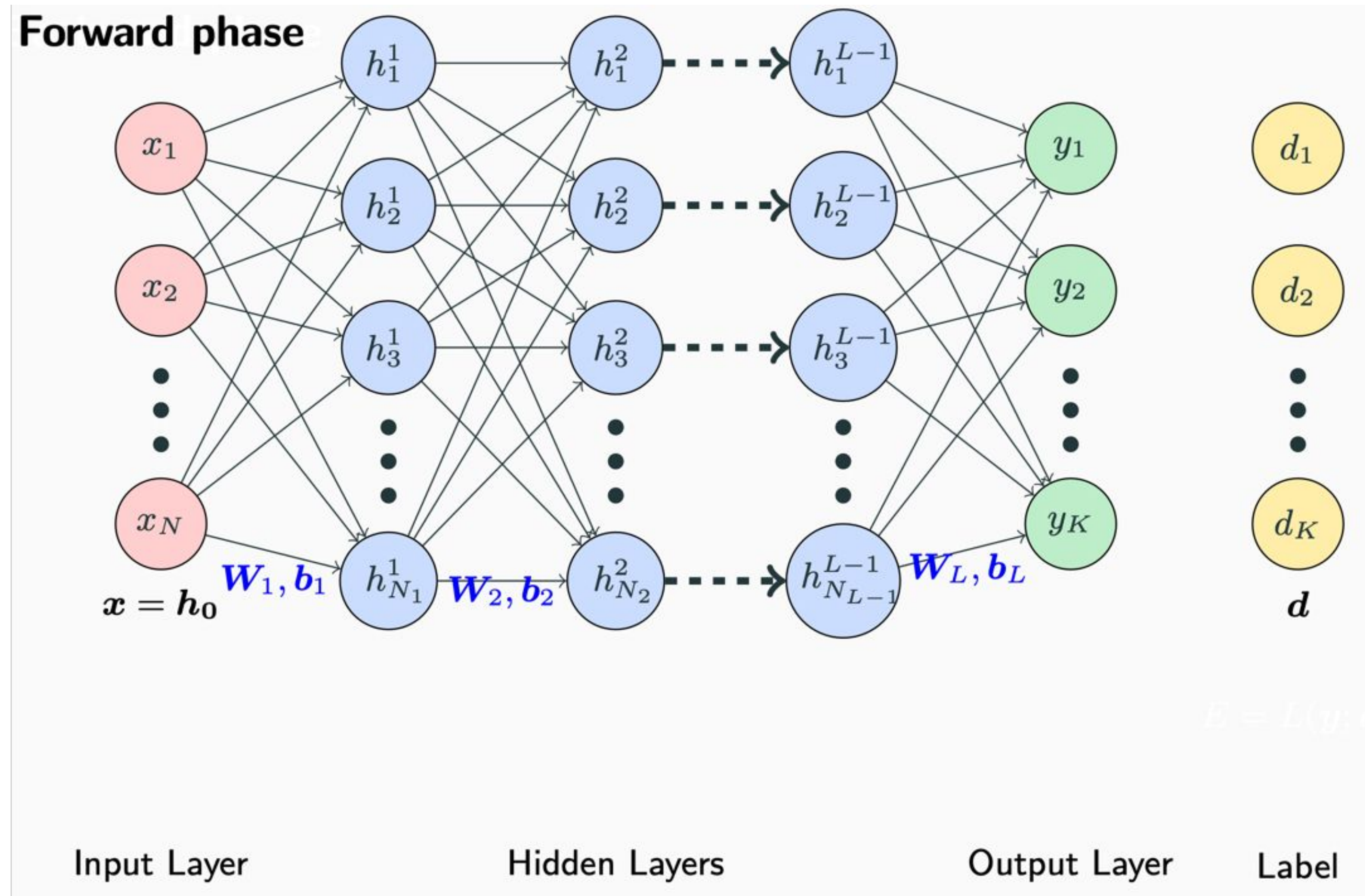
Multiplying by  $\mathbf{W}_k^T$  amounts to propagating the error to the previous layers

The error is backpropagated layer by layer in order to calculate the gradient of the cost function with respect to the parameters of each layer



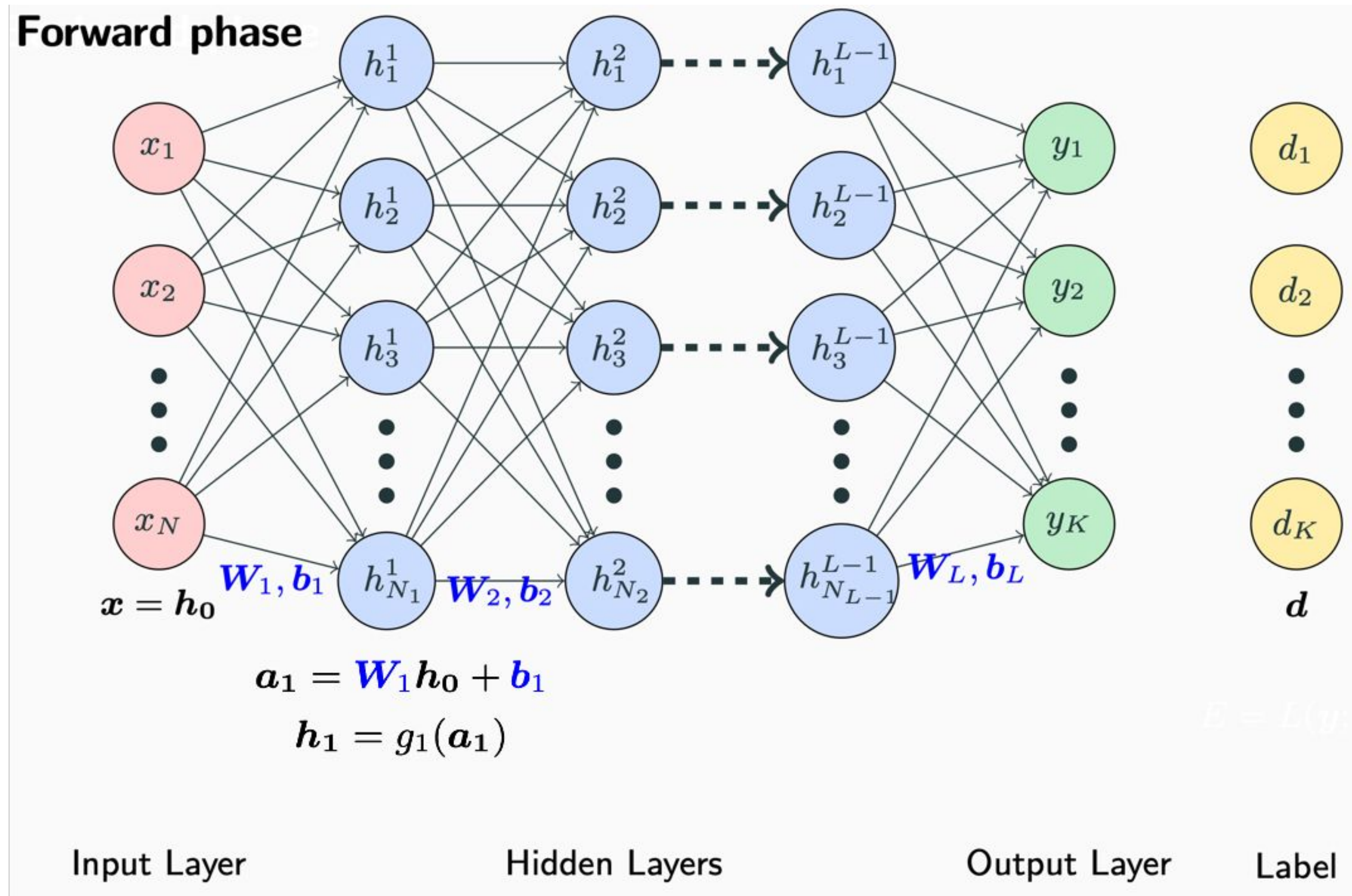


# Backpropagation



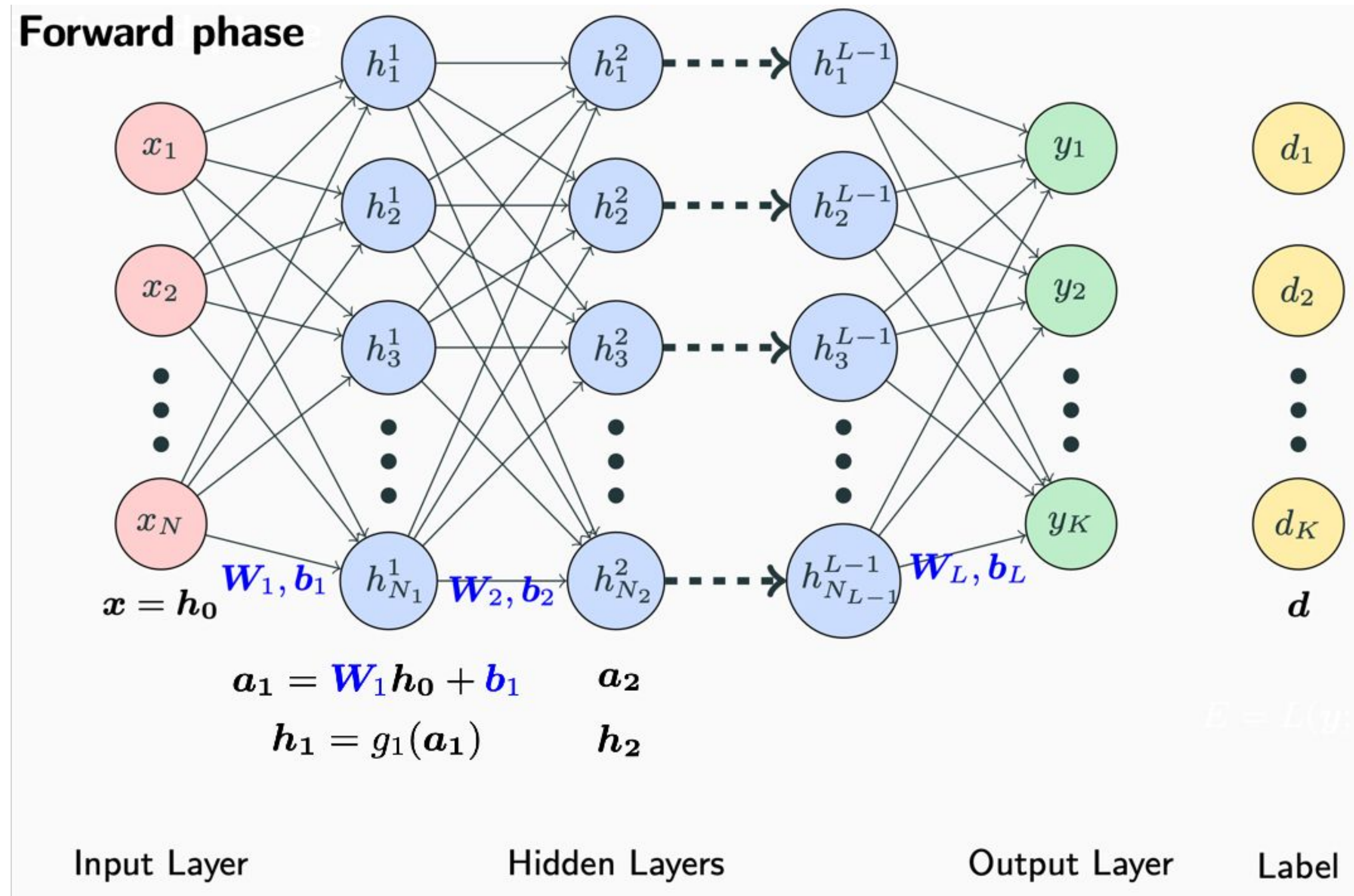


# Backpropagation



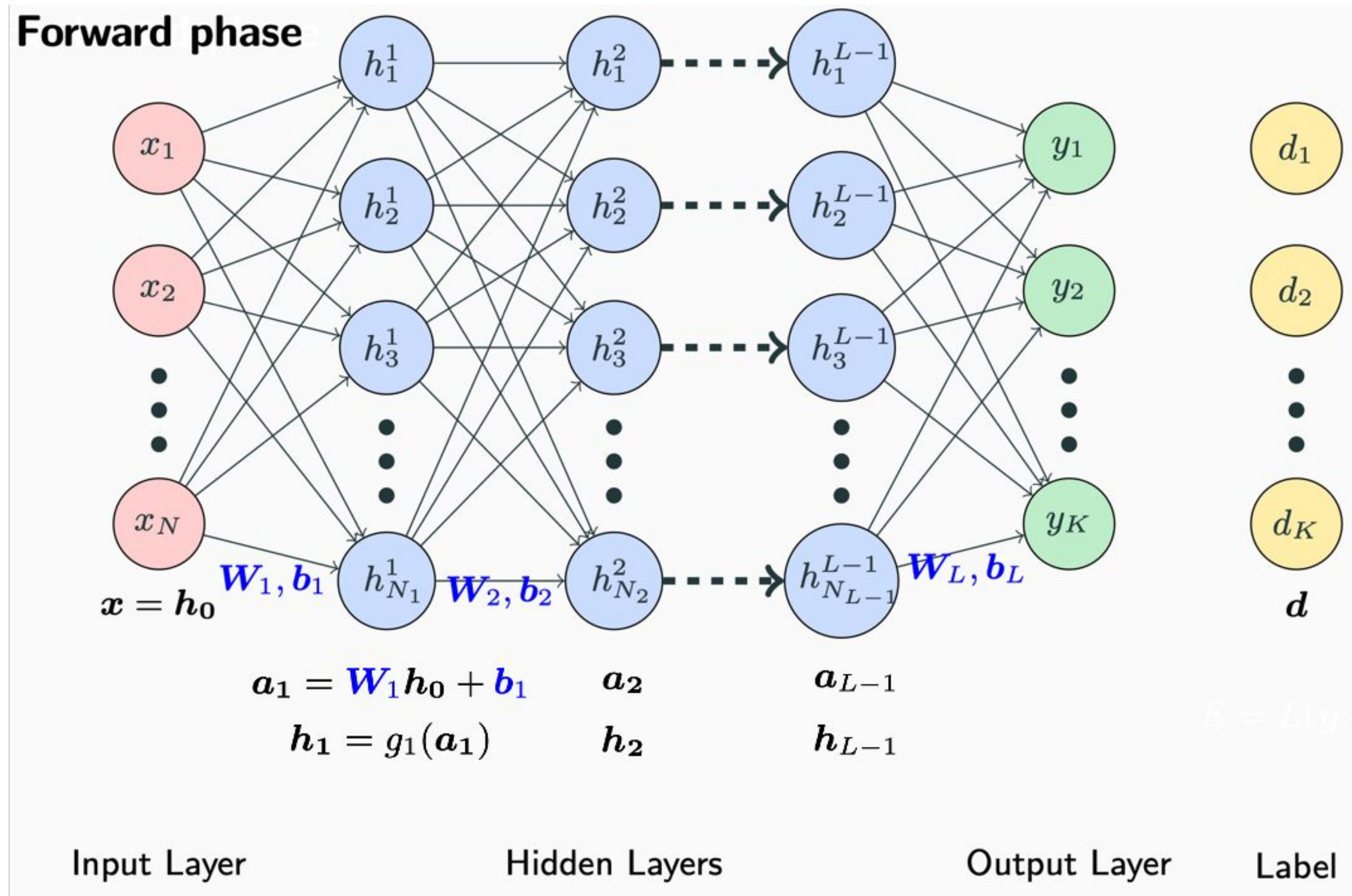


# Backpropagation



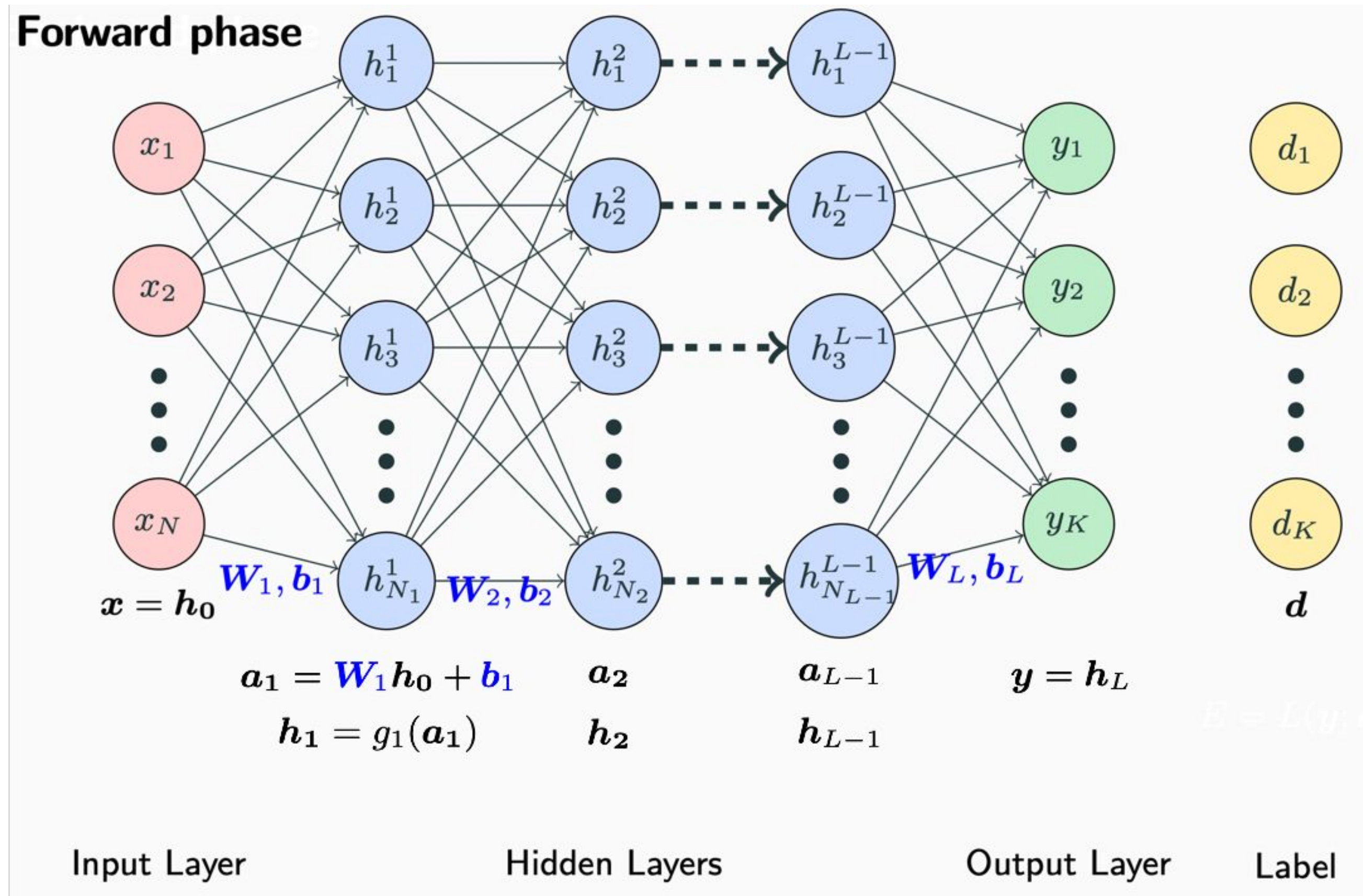


# Backpropagation



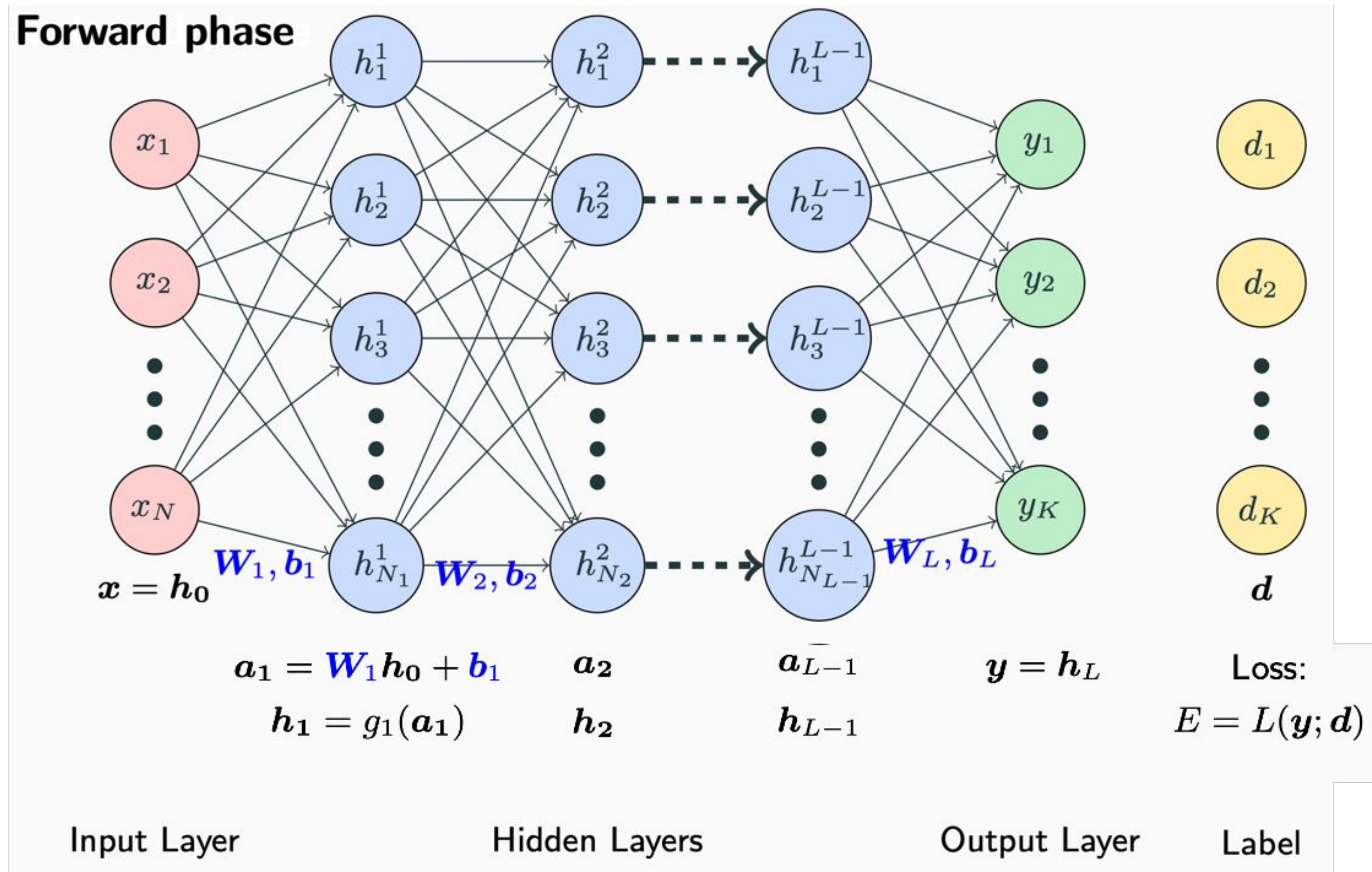


# Backpropagation



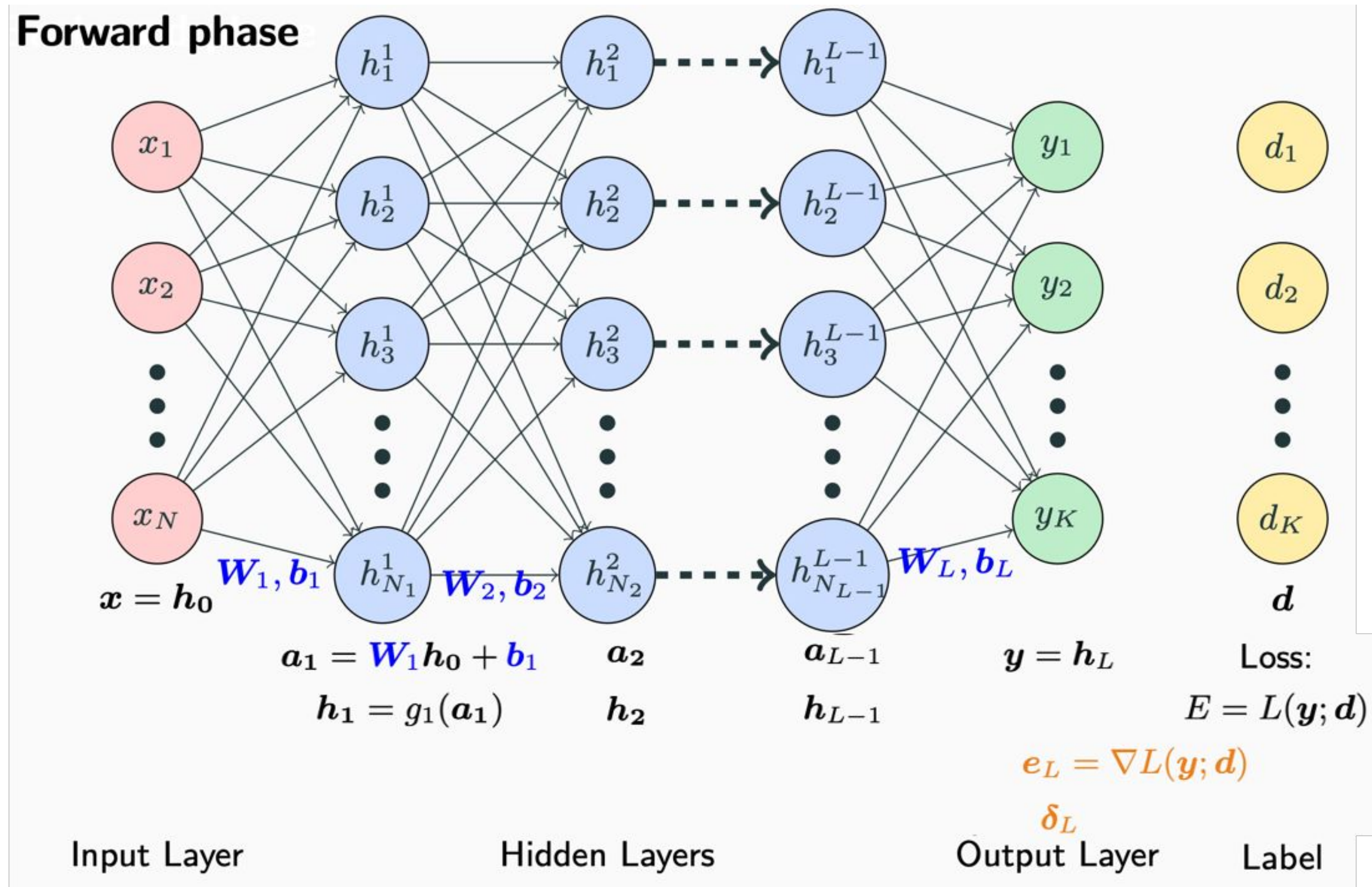


# Backpropagation



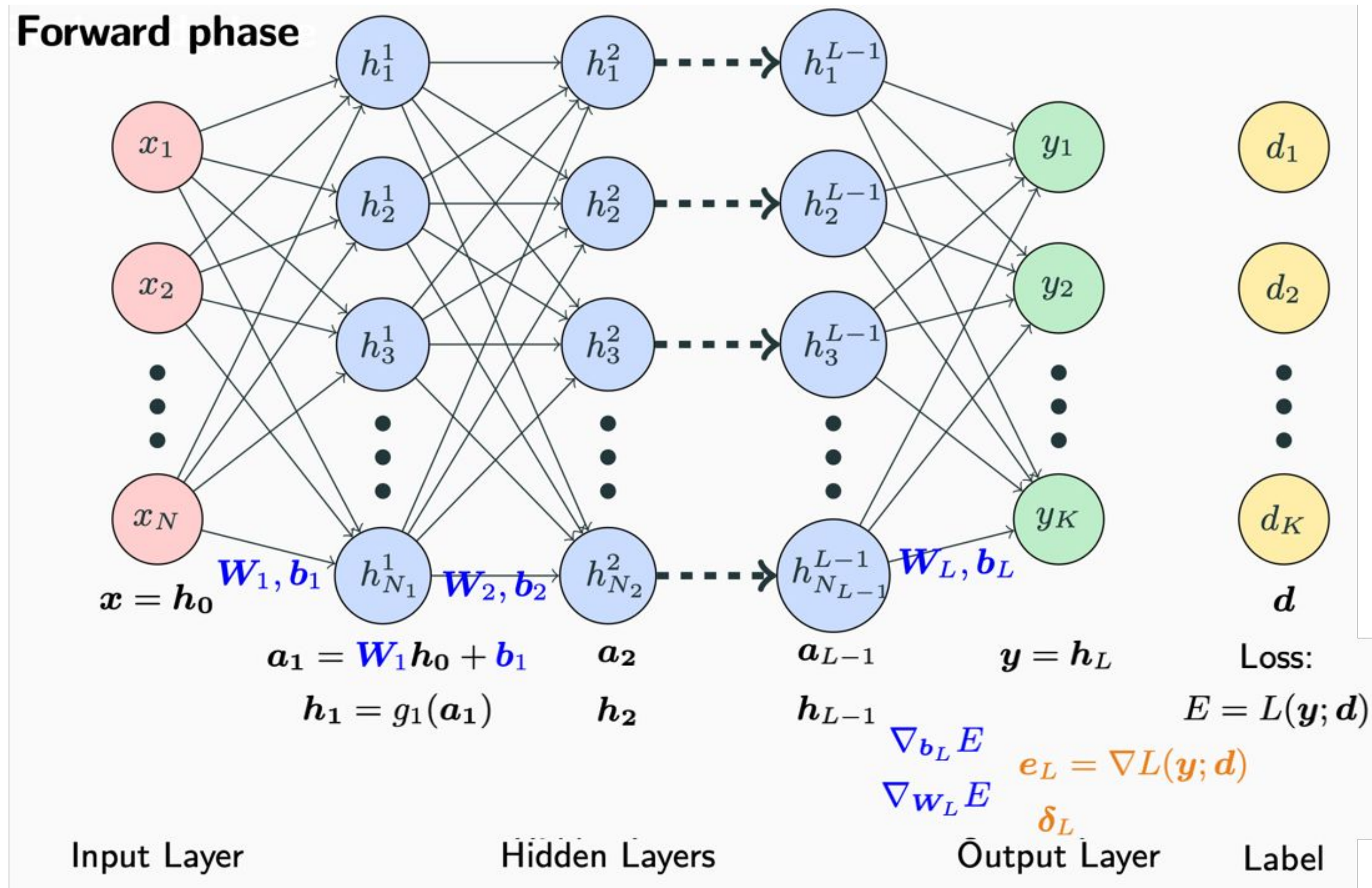


# Backpropagation



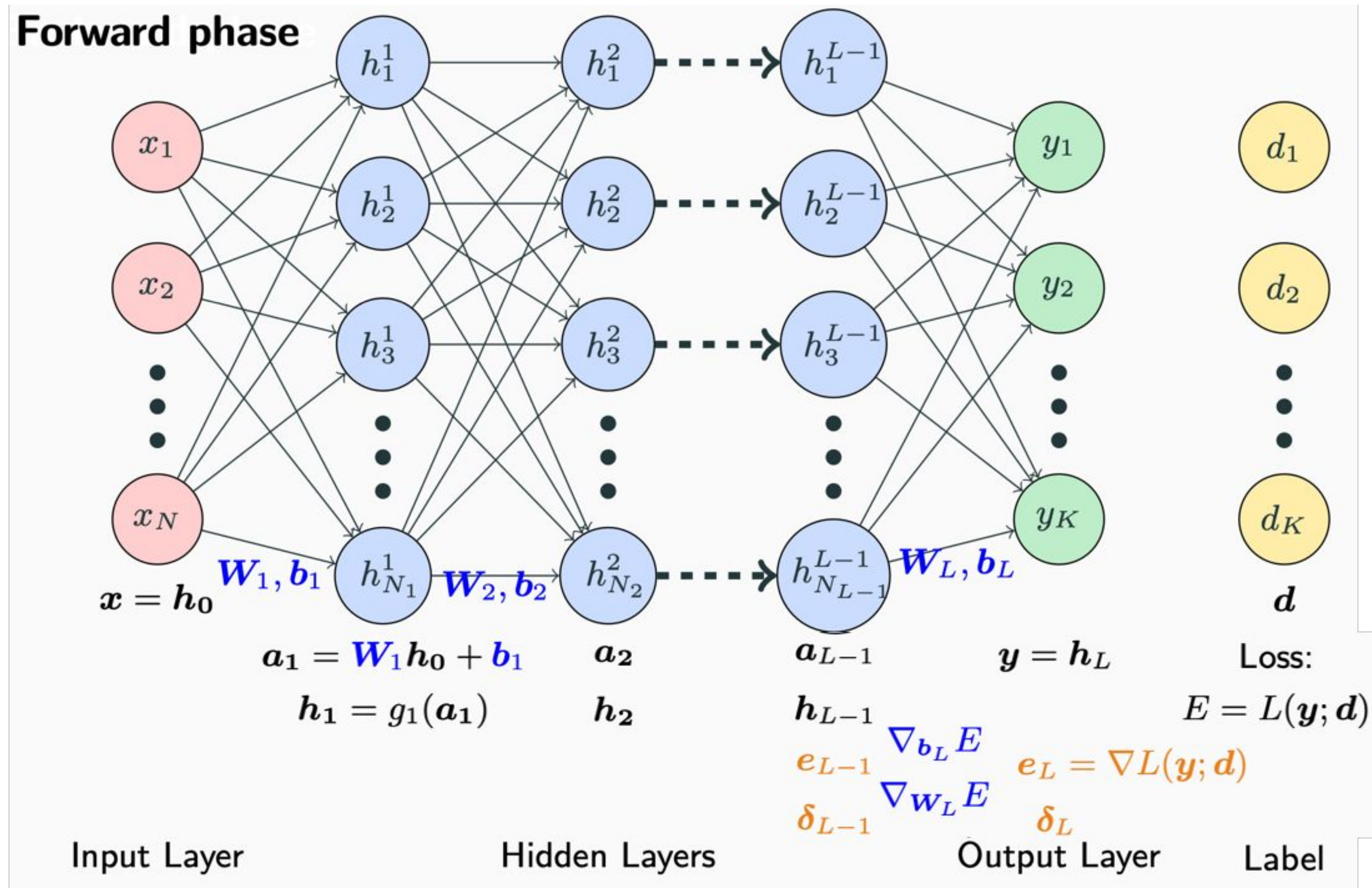


# Backpropagation



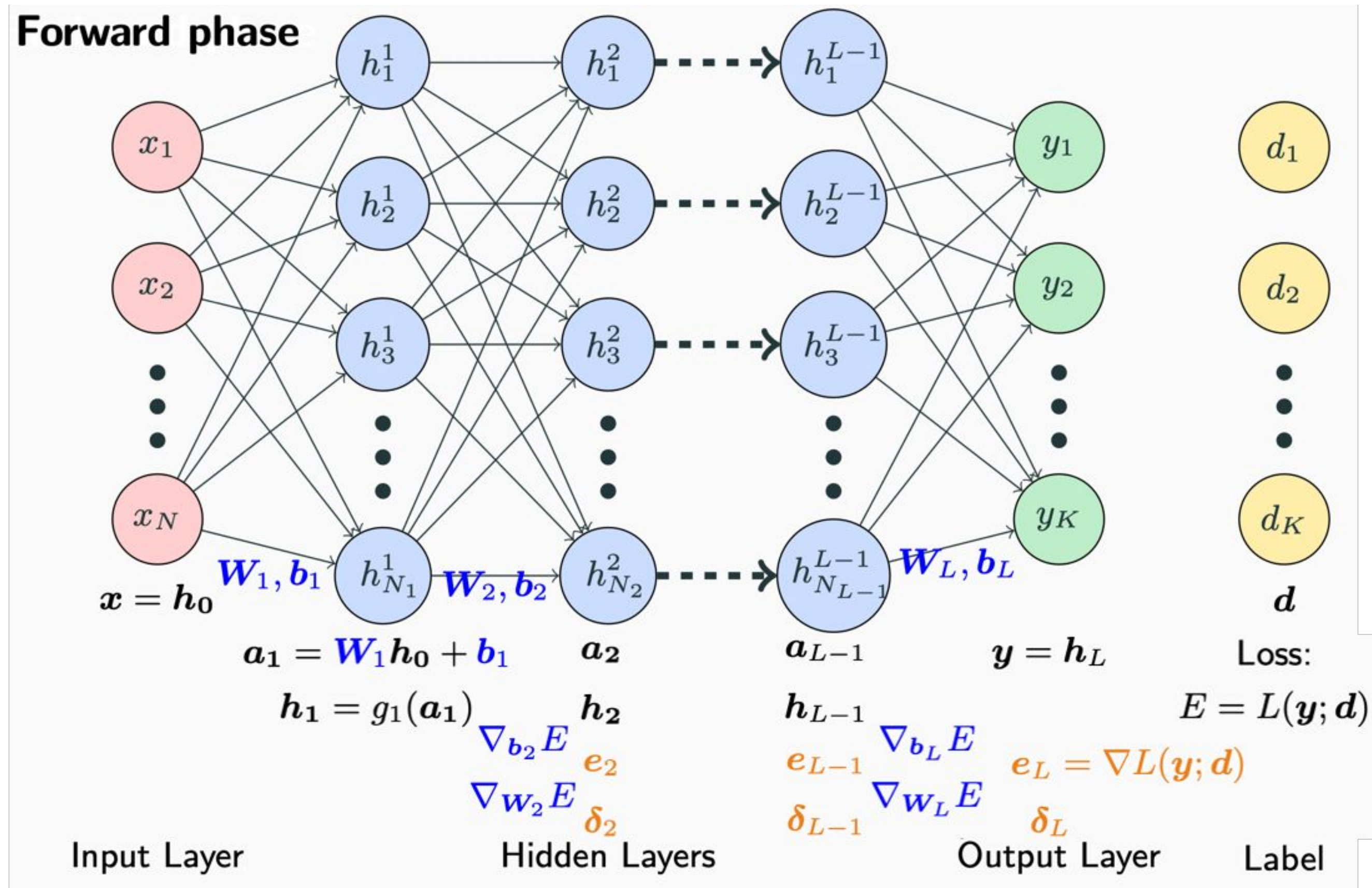


# Backpropagation



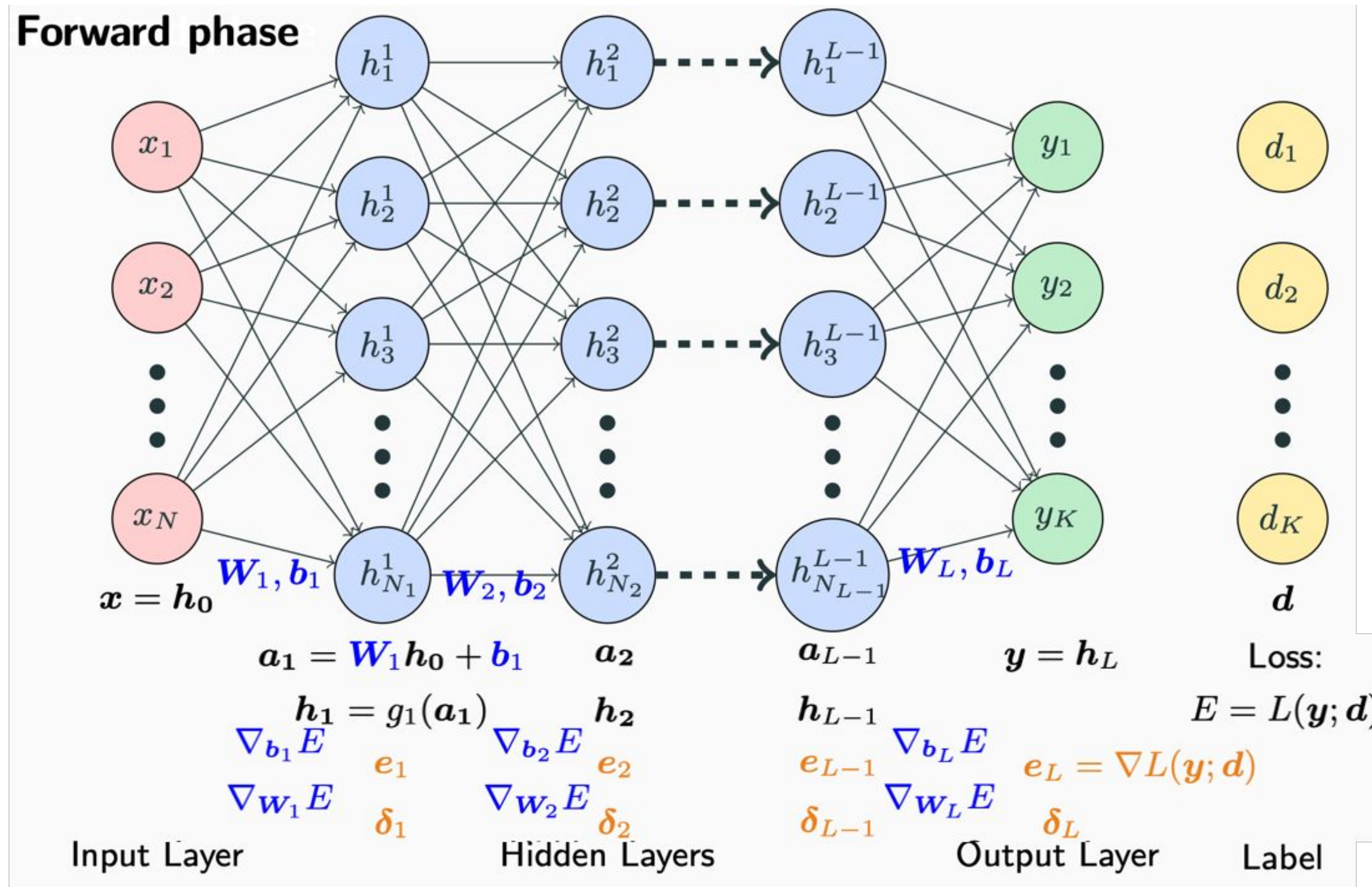


# Backpropagation



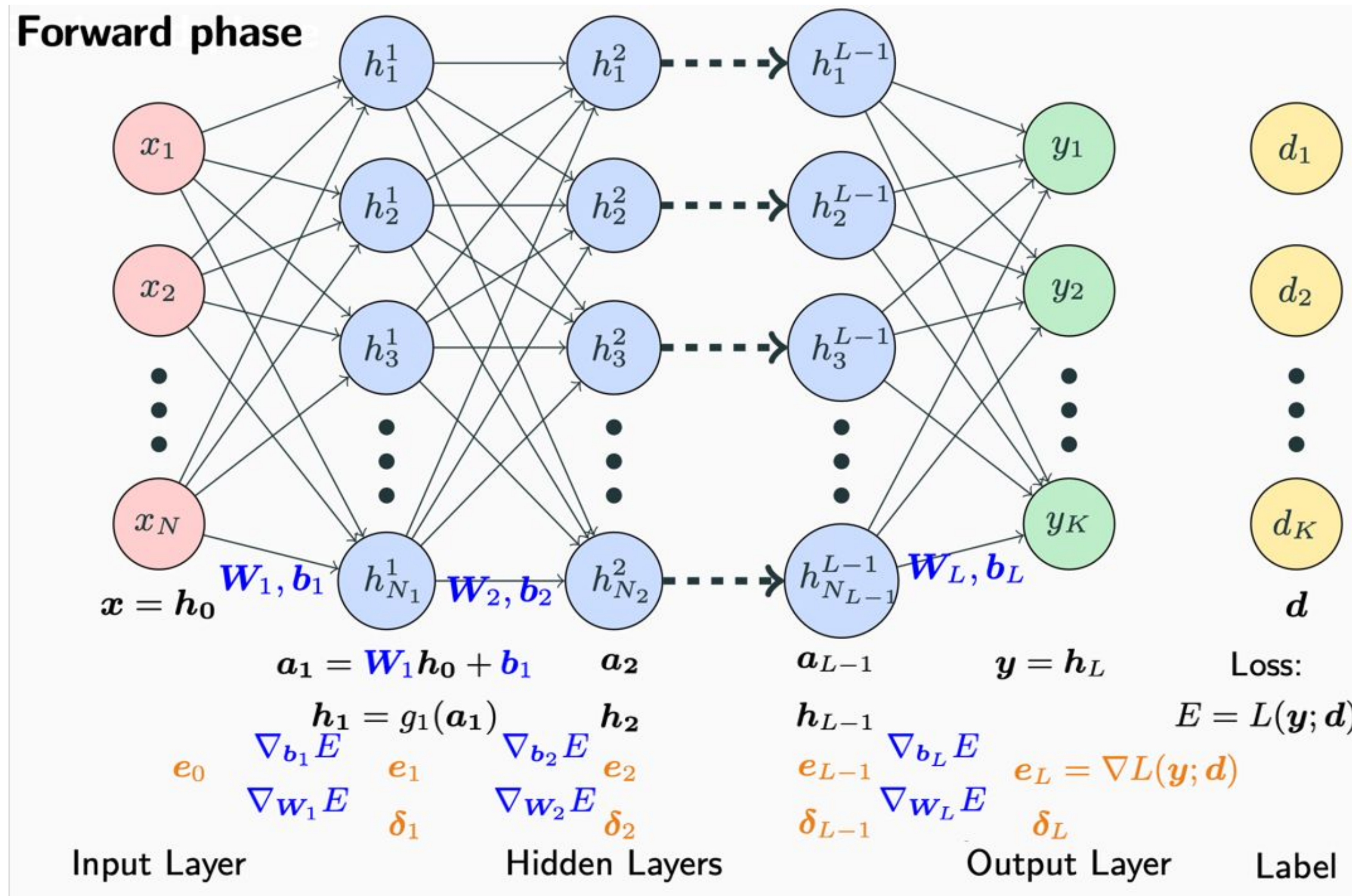


# Backpropagation





# Backpropagation



**To come next : Convolutional Neural Networks, Transformers, ....**



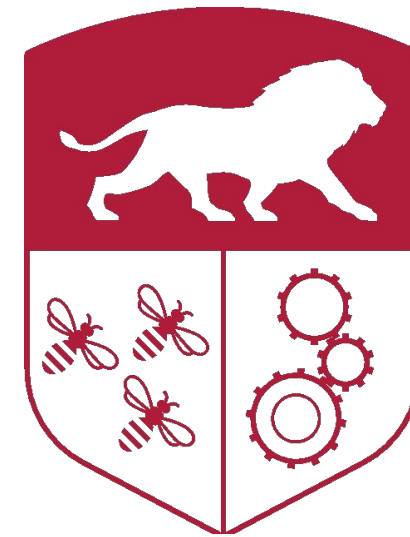
# **Some useful references**

**A. Ng, “Machine Learning” Course, Stanford University.**

**R.Duda, P. Hart, “Pattern Classification and Scene Analysis”, John Wiley & Sons, 2001.**

**C. M. Bishop, “Pattern recognition and machine learning”, Springer, 2006.**

**Numpy Library, <https://numpy.org/>**



**CENTRALE  
LYON**

---

36, avenue Guy de Collongue 69130 Écully  
[www.ec-lyon.fr](http://www.ec-lyon.fr) | [@centralelyon](https://twitter.com/centralelyon)