ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

Backpropagation

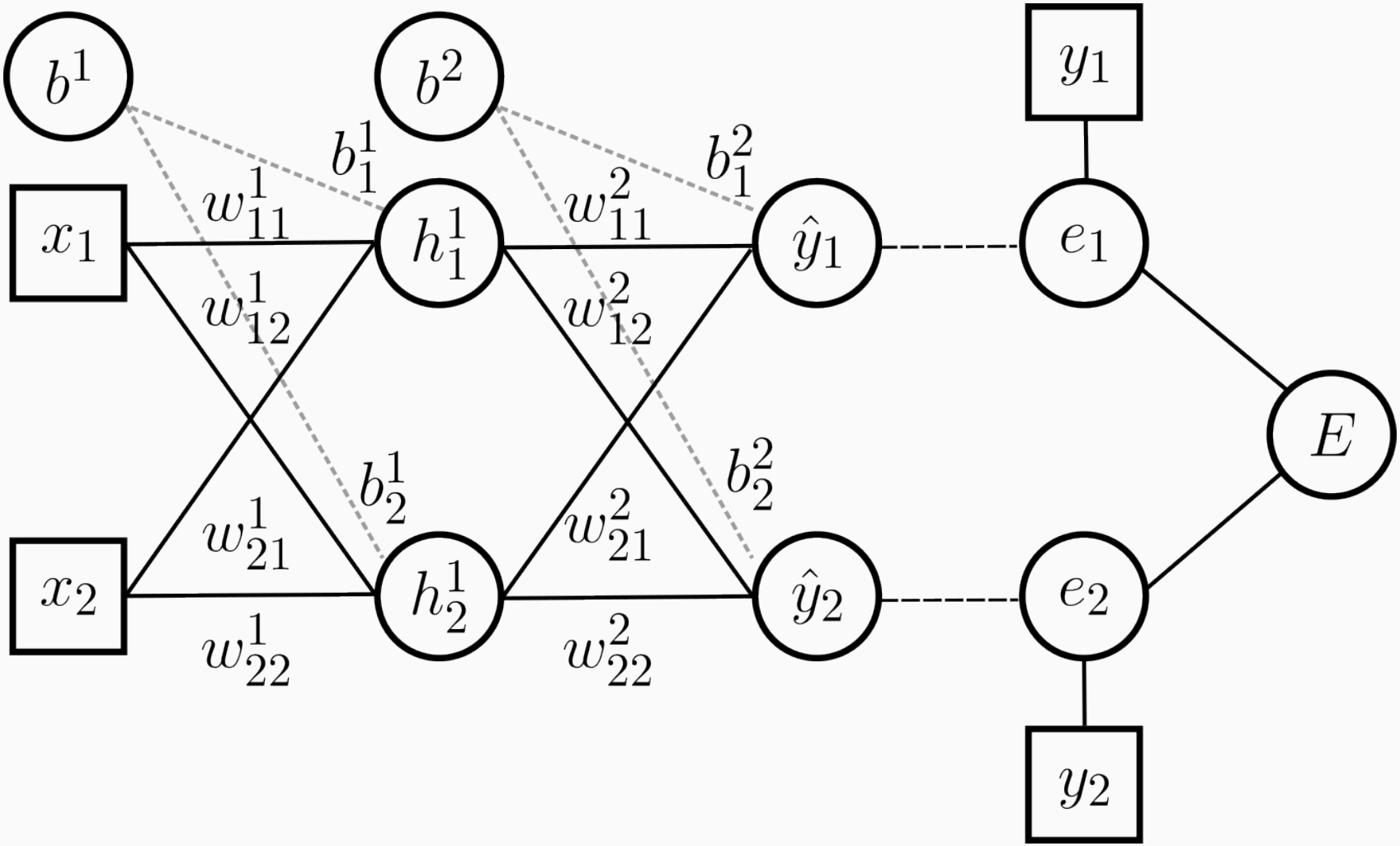
Joshua Llano

Universitat Politècnica de Catalunya

Table of contents

Introduction

Some notation



Some notation

Let's consider the bias as an additional weight for an input always equal to 1, we have:

Input layer: $X = [1, x_1, x_2]^T$; Hidden layer: $H^i = [1, h_1^i, h_2^i]^T$;

Output layer: $Y = [y_1, y_2]^T$; Matrix of weights: $W^i = \begin{bmatrix} b_1^i & w_{11}^i & w_{21}^i \\ b_2^i & w_{12}^i & w_{22}^i \end{bmatrix}$

Activation function: $h_i^k = f^k(z_i^k) = f^k(\sum_{j=0}^{N_{k-1}} w_{ji}^k h_j^{k-1})$

Output: $\hat{Y} = g(X, W) = f^2(W^2 H^1) = f^2(W^2 f^1(W^1 x))$

Loss Function: $E = E(Y, \hat{Y}) = E(y, g(X, W^i))$

Backpropagation

Backpropagation

Backpropagation

It's a training algorithm that aims to adjust the weights of a neural network using gradient descent. It has four steps

1. **Initialization**: Initialize the weights w and biases b with small random values.
2. **Forward Propagation**: Propagate the data trough the model, calculating the output of every neuron till getting the final output and the corresponding error (loss function).
3. **Backward Propagation**: Propagate the error back, calculating for each weight the corresponding gradient used to adjust the weight.
4. **Update Weights**: Adjust the weights and biases based on the computed gradients using a learning rate η.

Steps 2, 3 and 4 are repeated until some stability criteria is reached.

Why the gradients

For each weight w_{ij}^k, we want to know how much a change in w_{ij}^k affect the total error:

$$\frac{\delta E}{\delta w_{ij}^k}$$

This is the "direction" in which we want to move the weights, while the learning rate α is how much we are moving it (Gradient descent):

$$w_{ij}^k = w_{ij}^k - \alpha \frac{\delta E}{\delta w_{ij}^k}$$

The problem is how to get the derivative term

Get the gradients I

We start propagating the error through the output layer. Let's start from the chain rule, we know that if $h(x) = f(g(x))$, then $(h'(x) = f'(g(x))g'(x)$, hence:

$$\frac{\delta E}{\delta w_{ij}^k} = \frac{\delta E}{\delta h_j^k} \frac{\delta h_j^k}{\delta w_{ij}^k} = \frac{\delta E}{\delta h_j^k} \frac{\delta h_j^k}{\delta z_j^k} \frac{\delta z_j^k}{\delta w_{ij}^k}$$

The last term is easy to get:

$$\frac{\delta z_j^k}{\delta w_{ij}^k} = \frac{\delta}{\delta w_{ij}^k} \sum_{n=0}^{N_{k-1}} w_{nj} h_n^{k-1} = h_i^{k-1}$$

The second term is the derivative of the activation function:

$$\frac{\delta h_j^k}{\delta z_j^k} = \frac{\delta f(z_j^k)}{\delta z_j^k} = f'(z_j^k)$$

Get the gradients II

While the first term depends on the error function. Consider for example a MSE, for the last layer we have:

$$\frac{\delta E}{\delta h_j^k} = \frac{\delta E}{\delta \hat{y}} = \frac{\delta}{\delta \hat{y}} \frac{1}{2}(y - \hat{y})^2 = \hat{y} - y$$

So putting everything together we have:

$$\frac{\delta E}{\delta w_{ij}^k} = (\hat{y} - y) f'(z_j^k) h_i^{k-1}$$

But it is usually simplified as:

$$\frac{\delta E}{\delta z_{ij}^k} = \frac{\delta E}{\delta h_j^k} \frac{\delta h_j^k}{\delta z_j^k} = \delta_j^k$$

Hence:

$$\frac{\delta E}{\delta w_{ij}^k} = \delta_j^k h_i^{k-1}$$

Get the gradients III

The problem is for the other layers, again we have to use the chain rule, considering the error as a function of all the neurons that receive some input from the neuron j:

$$\frac{\delta E}{\delta h_j^k} = \sum_{l=0}^{N_{k+1}} \frac{\delta E}{\delta z_l^{k+1}} \frac{\delta z_l^{k+1}}{\delta h_j^k} = \sum_{l=0}^{N_{k+1}} \frac{\delta E}{\delta h_l^{k+1}} \frac{\delta h_l^{k+1}}{\delta z_l^{k+1}} \frac{\delta z_l^{k+1}}{\delta h_j^k} = \sum_{l=0}^{N_{k+1}} \delta_l^{k+1} w_{jl}^{k+1}$$

Hence...

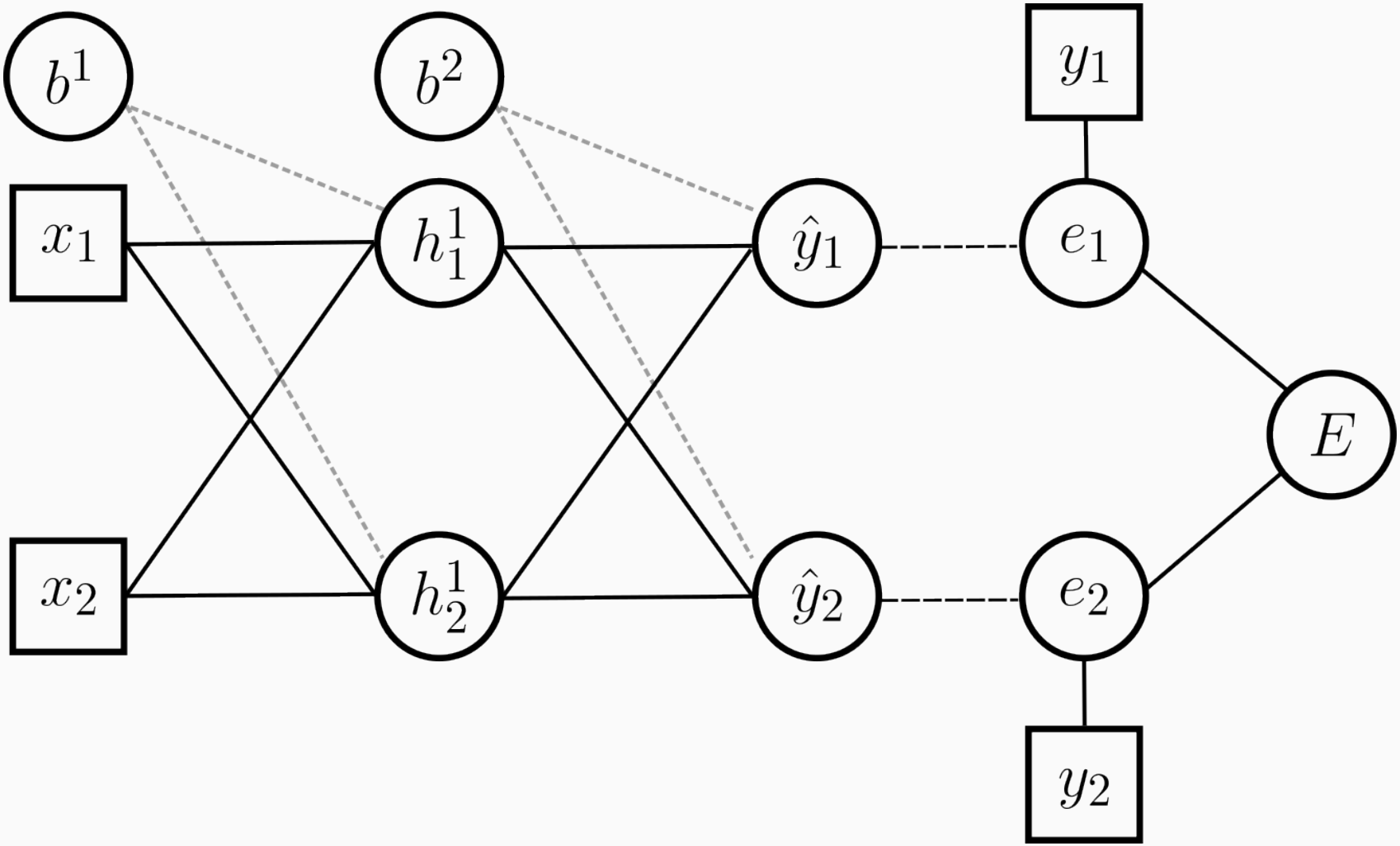
So putting everything together we have:

$$\frac{\delta E}{\delta w_{ij}^k} = \delta_j^k h_i^{k-1}$$

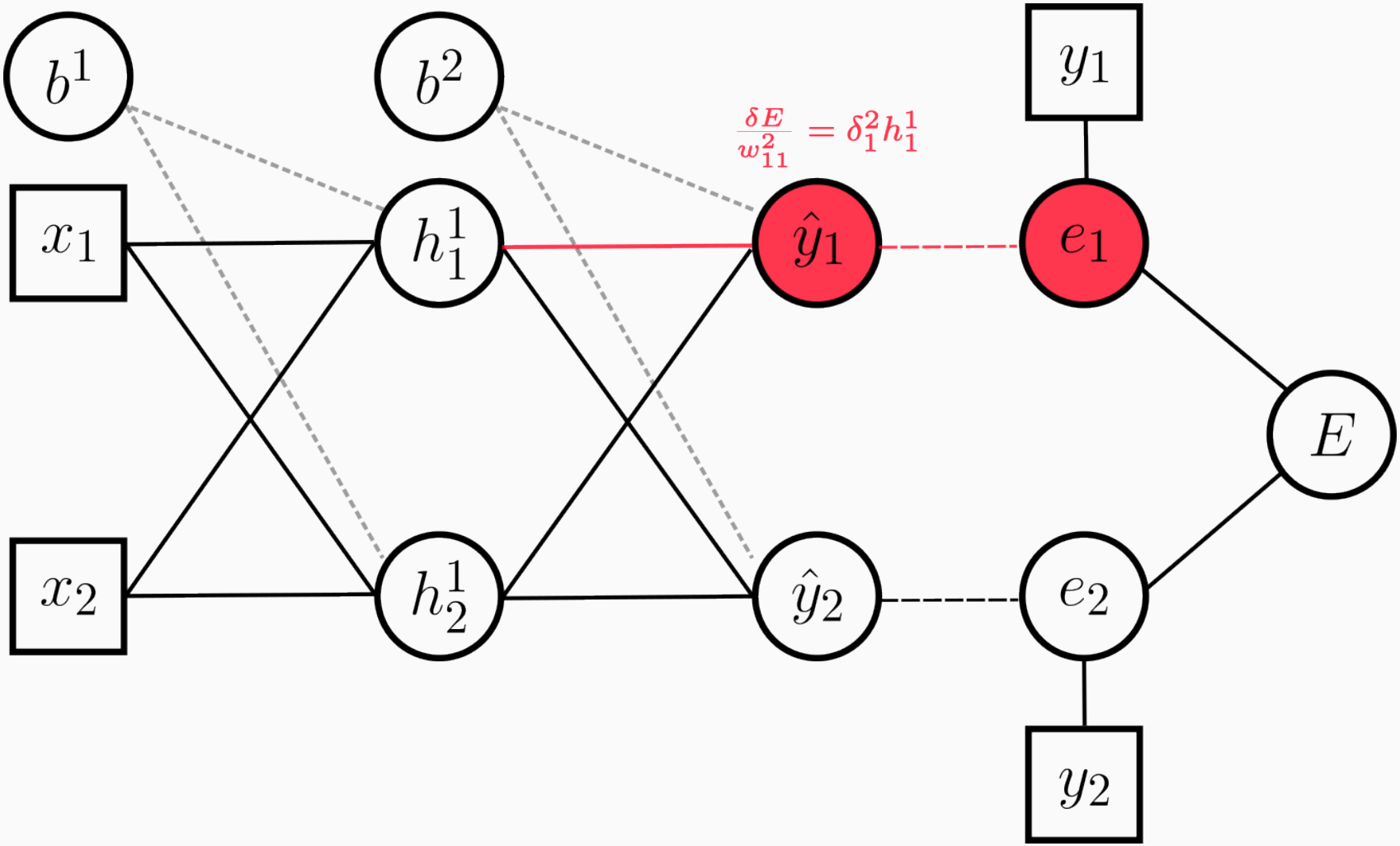
Where:

$$\delta_j^k = \begin{cases} \dfrac{\delta loss(y, \hat{y})}{\delta \hat{y}} f'(a_j^k) & \text{if j is output neuron} \\ (\sum_{l=0}^{N_{k+1}} \delta_l^{k+1} w_{jl}^{k+1}) f'(a_j^k) & \text{if j is inner neuron} \end{cases}$$

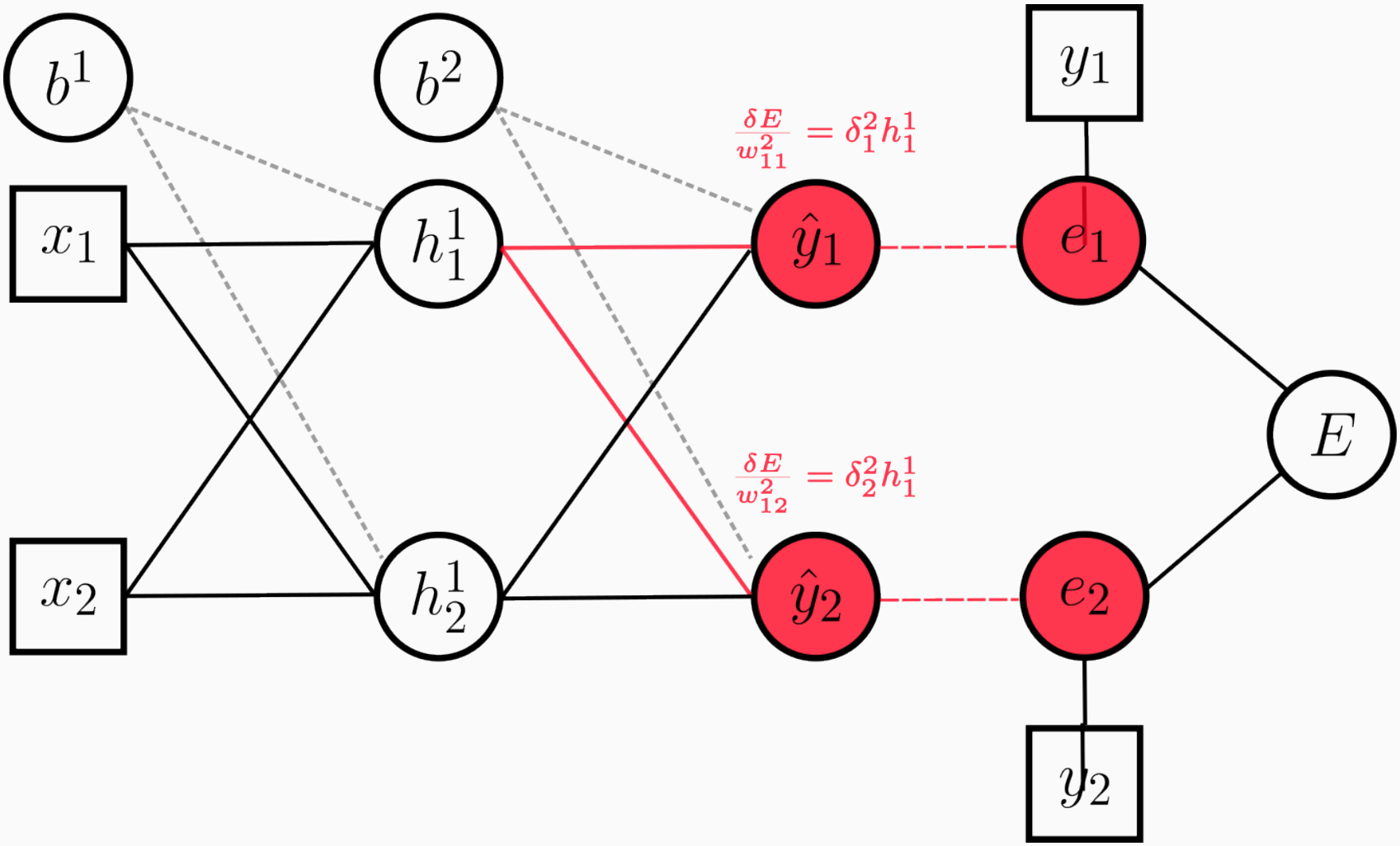
For example...



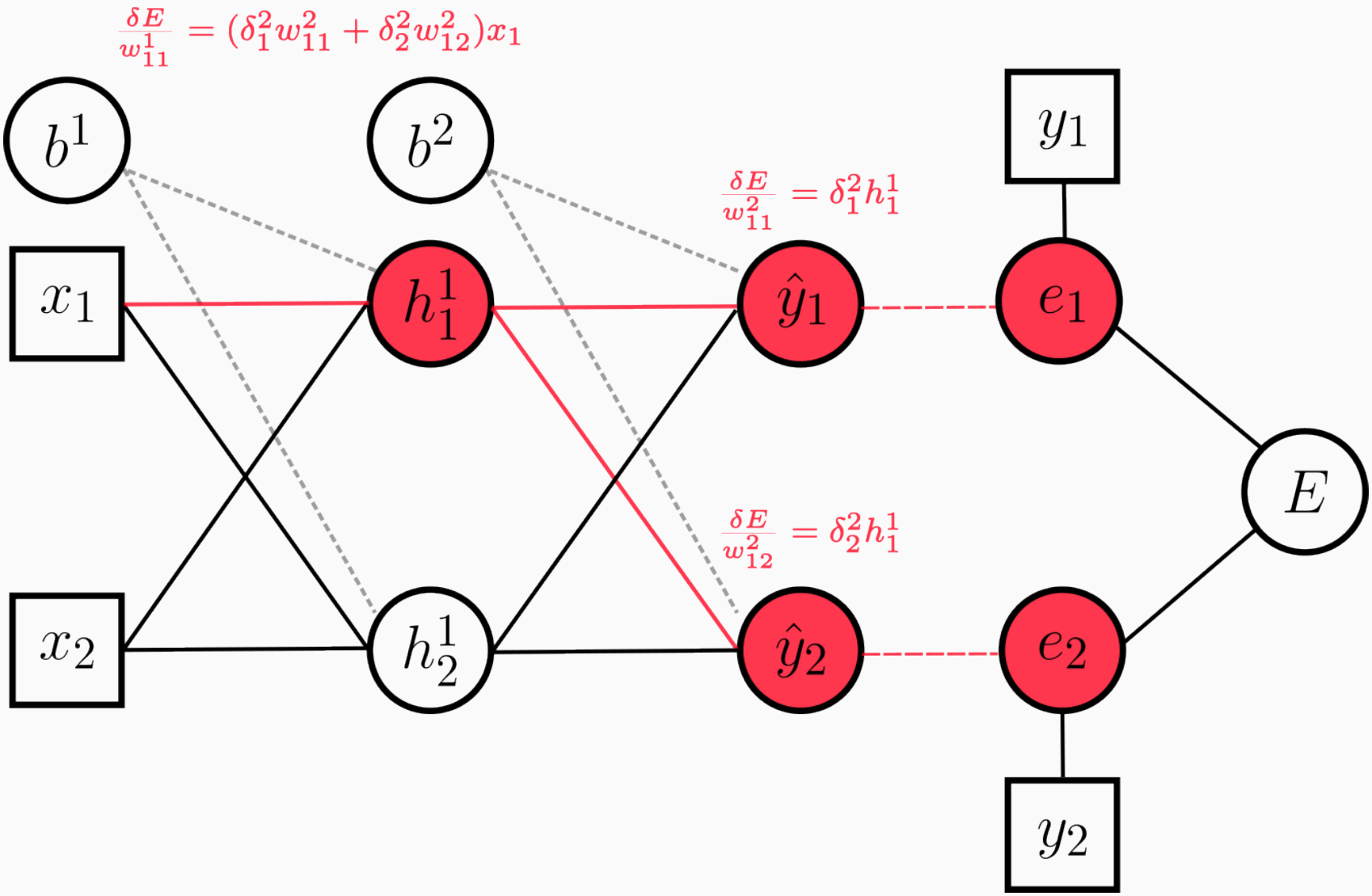
For example...


b^1
b^2
y_1
$\frac{\delta E}{w_{11}^2} = \delta_1^2 h_1^1$
x_1
h_1^1
$\hat{y}_1$
e_1
E
x_2
h_2^1
$\hat{y}_2$
e_2
y_2

For example...


b^1
b^2
x_1
x_2
h_1^1
h_2^1
$\hat{y}_1$
$\hat{y}_2$
$\frac{\delta E}{w_{11}^2} = \delta_1^2 h_1^1$
$\frac{\delta E}{w_{12}^2} = \delta_2^2 h_1^1$
y_1
y_2
e_1
e_2
E

For example...


$\frac{\delta E}{w_{11}^1} = (\delta_1^2 w_{11}^2 + \delta_2^2 w_{12}^2) x_1$
b^1
b^2
x_1
x_2
h_1^1
h_2^1
$\hat{y}_1$
$\hat{y}_2$
$\frac{\delta E}{w_{11}^2} = \delta_1^2 h_1^1$
$\frac{\delta E}{w_{12}^2} = \delta_2^2 h_1^1$
y_1
y_2
e_1
e_2
E

In practice

In practice I

When you build a model, all these steps are already implemented, what we need to do is:

1. Data preparation

This include all the steps to get the correct X and Y matrices:

- **Data cleaning** detect and eventually remove outliers and other wrong values.
- **Missing imputation** note that the input for a MLP can not have missing values in it, so if you do not have a data point you have to came up with some idea about it.
- **Feature scaling** Consider to standardize or normalize the data according to the problem.

In practice II

When you build a model, all these steps are already implemented, what we need to do is:

2. Model design

For a MLP the hyper parameters we have to tune are:

- Number of hidden layers
- Number of neurons in each layer
- Activation functions of each layer, usually the hidden layers have all the same activation function (ReLu is often used), while the output layer activation function is crucial for the task:
  - **Linear** for regression task, where the output can be any real number
  - **Sigmoid** for binary/multilabel classification problems
  - **Softmax** for multiclass classification problems

In practice III

When you build a model, all these steps are already implemented, what we need to do is:

3. Experiment design

There are some other parameters that affect the training process:

- **Loss function** It depends on the problem, for example:
  - **MSE** for regression problems
  - **Binary Cross-Entropy** for binary classification problems
  - **Multi-class Cross-Entropy** for multi class classification problems
- **Learning rate** The standard approach is to use an optimizer, an algorithm that adapt the learning rate depending on the momentum of the gradient. The classical one is Adam optimizer.

In practice IV

When you build a model, all these steps are already implemented, what we need to do is:

3. Experiment design

There are some other parameters that affect the training process:

- **Number of epochs**: defines the number times that the learning algorithm will work through the entire training dataset. Usually we fix a max number and then we early stop the training if the model learned "enough".
- **Batch size** is the number of training samples to work through before the model's internal parameters are updated. Smaller batch sizes are faster and improve generalization, but can lead to less stable results and it s easier to get stuck in some local minima.
- **Split** of the data in train, test and validate (crossvalidation?).