Introduction to Neural Networks

Ing. Zese Riccardo

riccardo.zese@unife.it

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Regione Emilia-Romagna

Università degli Studi di Ferrara

Outline

- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Sources

- These slides are taken from:
  - Intel Nervana AI Academy Intructional Content

  - Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

    https://www.deeplearningbook.org/lecture_slides.html
  - Chapter "Neural Networks" of "A Course in Machine Learning" by Hal Daume III.

    http://ciml.info/
  - Some parts from "CS231n: Convolutional Neural Networks for Visual Recognition", Stanford University

    http://cs231n.stanford.edu/

Outline

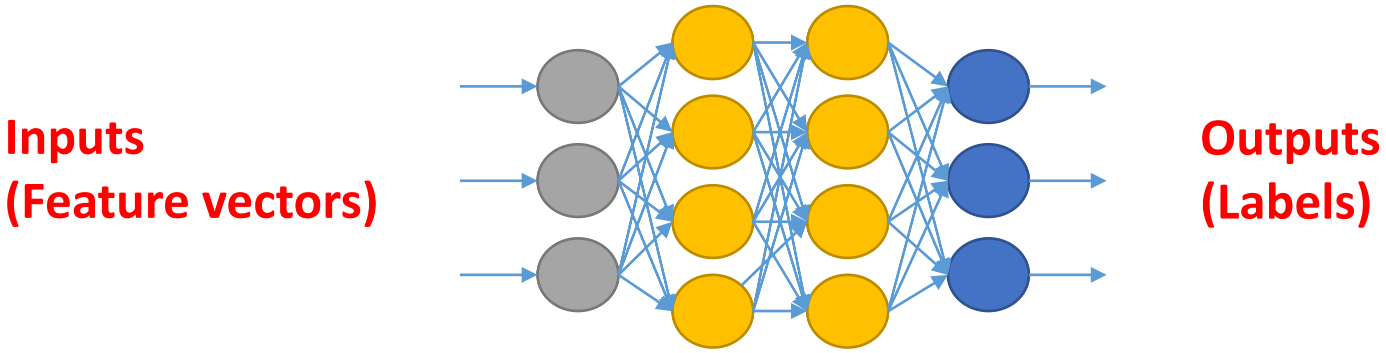
- Introduction to Python
- Introduction to Neural Networks
- Convolutional NN
- Recurrent NN
- Autoencoders and self supervised learning

Regione Emilia-Romagna

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Università degli Studi di Ferrara

How to train a Neural Net?



1. Put in Training inputs, get the output: **Forward Propagation**
2. Compare output to correct answers: Look at loss function J
3. Adjust and repeat!
4. **Backpropagation** (also called **backprop**) tells us how to make a single adjustment using calculus.

Backpropagation

- With the term back-propagation we are not indicating the whole learning algorithm.
- It refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent, is used to perform learning using this gradient.

How have we trained before?

- **Gradient Descent**.
  For most classical ML algorithms, the training happens here.

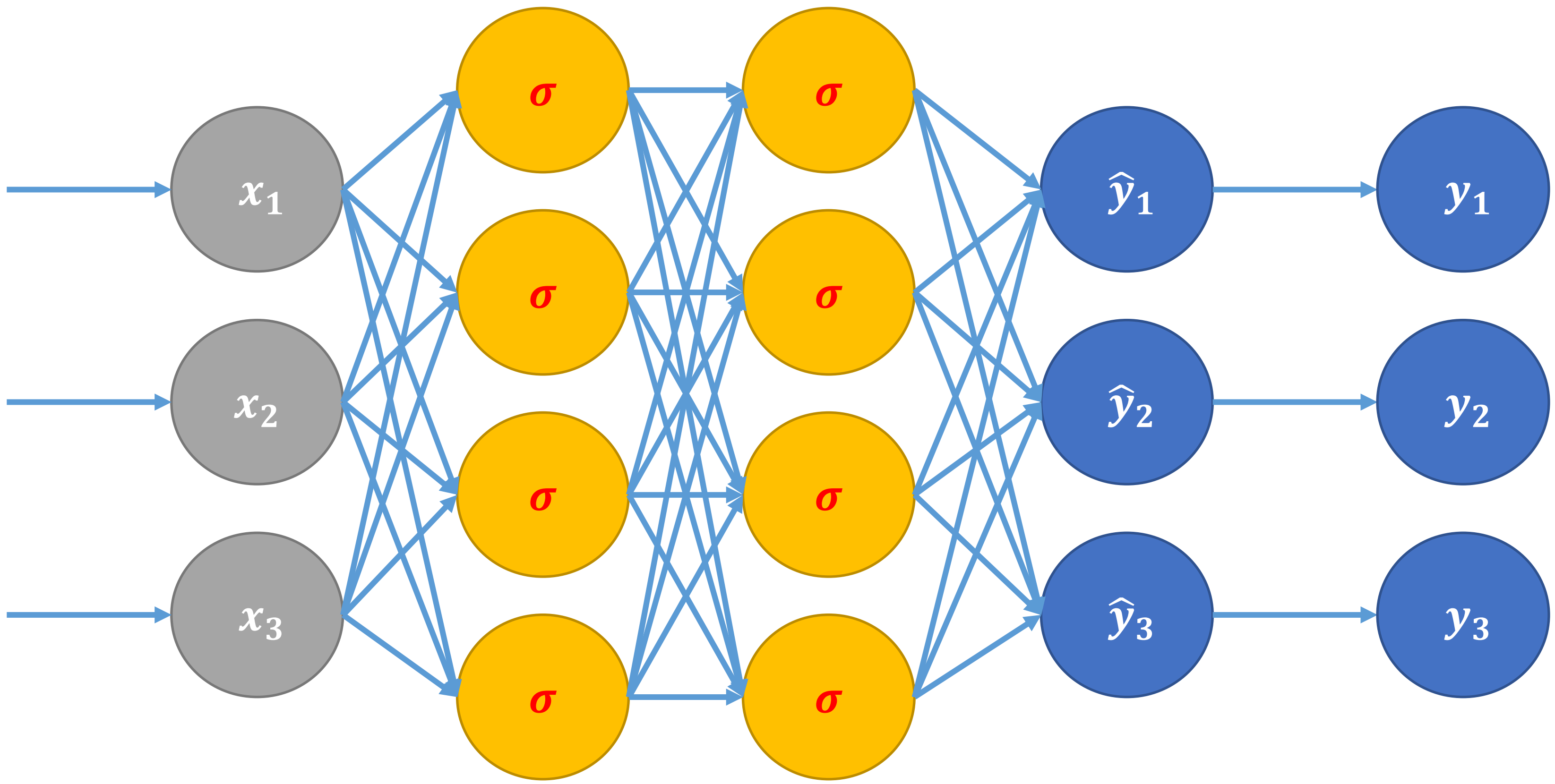
1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

How have we trained before?

- **Gradient Descent.**
  For most classical ML algorithms, the training happens here.

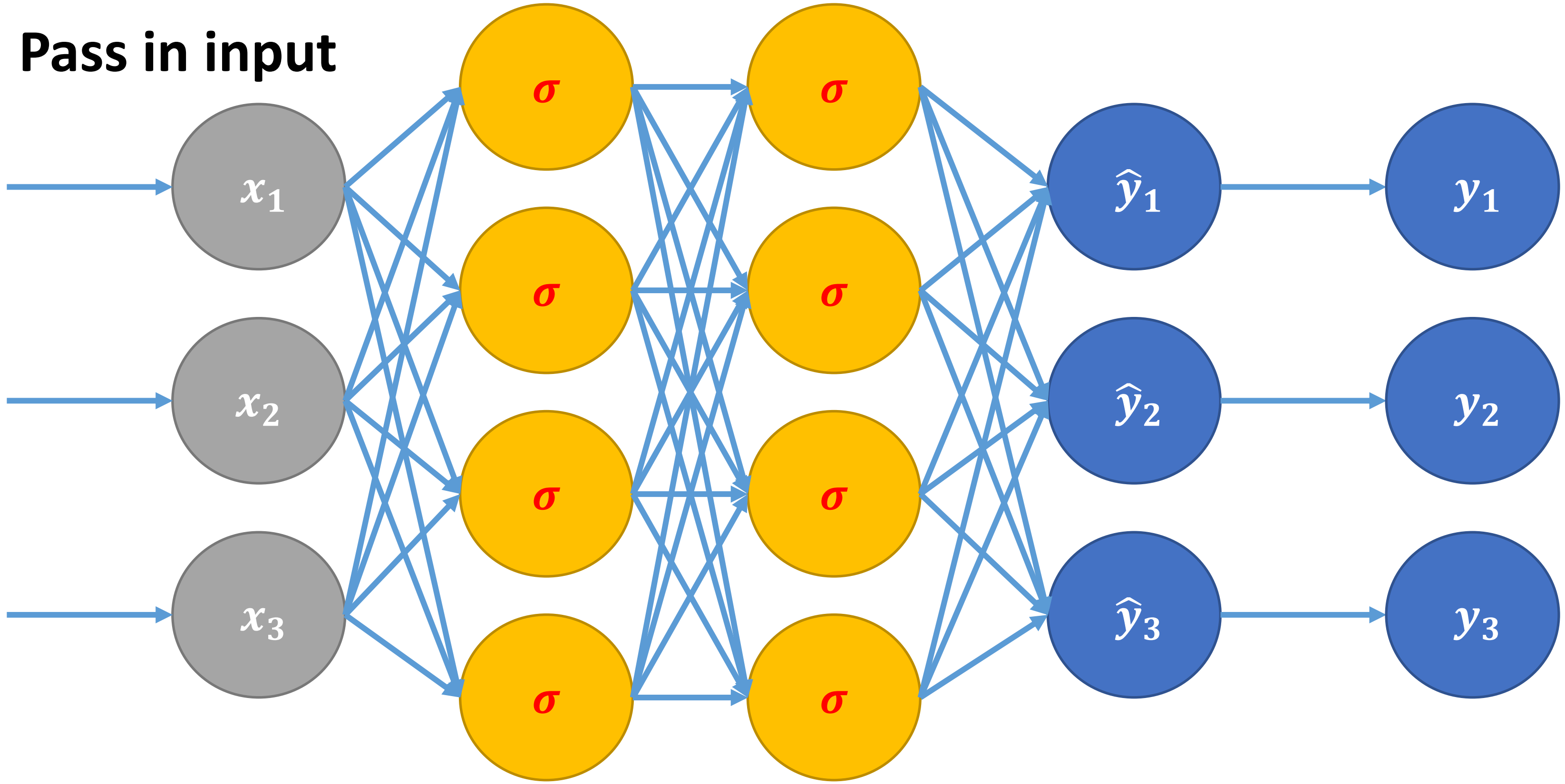
1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

Feedforward Neural Network



Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Feedforward Neural Network

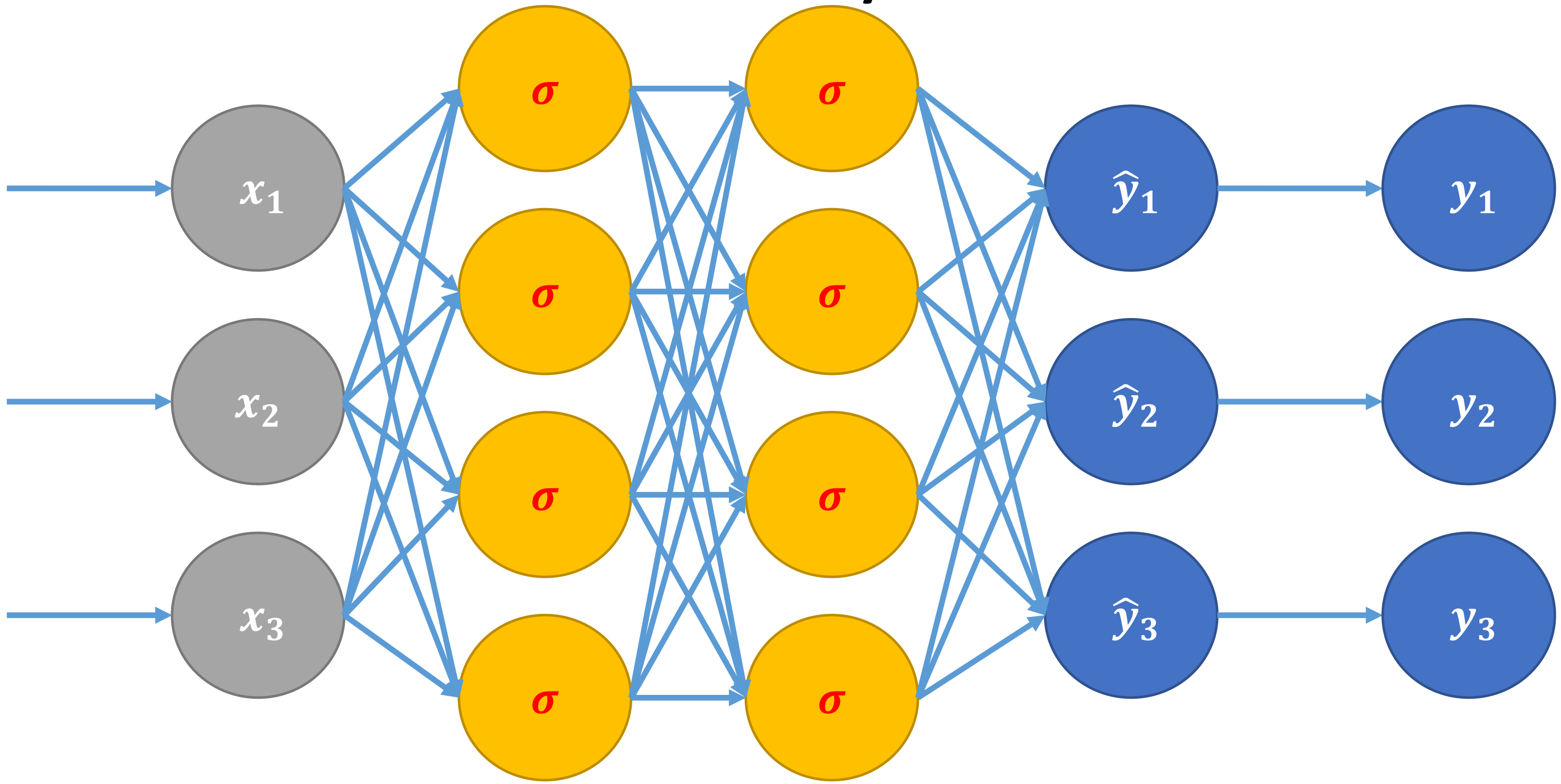


Inputs are rows of our training data

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Feedforward Neural Network

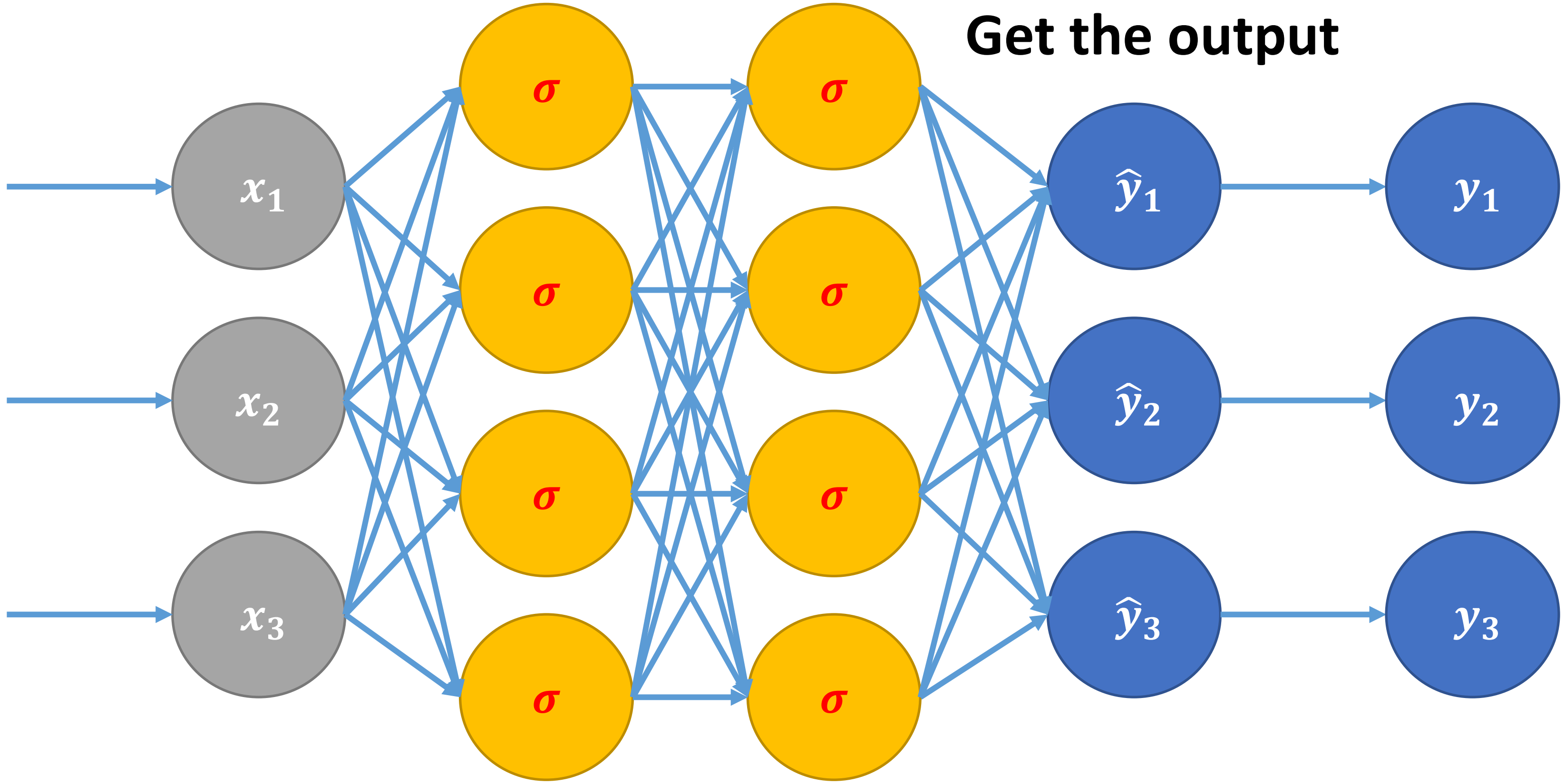
Calculate each layer



Perform the matrix multiplications and activation functions in order to calculate each layer.

Feedforward Neural Network

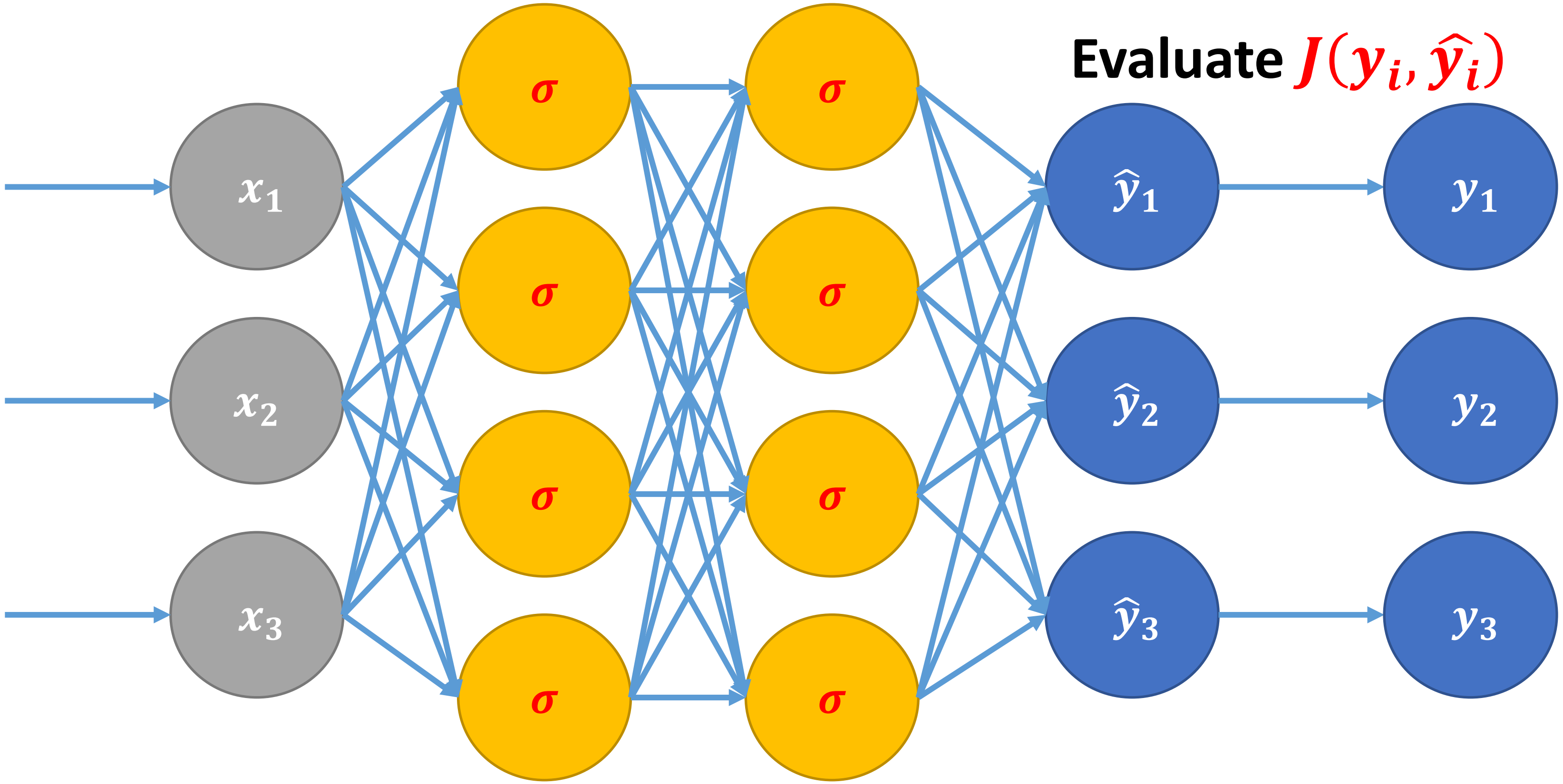
Get the output



Output is the prediction made by the net.

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Feedforward Neural Network



Compare the predictions to the known ground truths.
Specifically, calculate the loss function $J(y_i, \hat{y}_i)$.

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

How have we trained before?

- **Gradient Descent**.
  For most classical ML algorithms, the training happens here.

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

How to calculate gradient

- **Chain rule:** used to compute the derivatives of functions formed by composing other functions whose derivatives are known.
- Back-propagation is an algorithm that computes the chain rule, with a specific order of operations that is highly efficient.

Backpropagation

- You can summarize backpropagation as:

Backpropagation = gradient descent + chain rule

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Backpropagation

- Example: consider a two layers network whose output is

$$\hat{y} = \sum_i w_{2,i} f\left(w_{1,i} x\right)$$

where W_1 is a matrix containing weights of the first layer and W_2 a vector containing the weights of the second layer. f is the activation function of the first layer, which is usually non-linear (ReLU, tanh,…).

Backpropagation

- The overall objective is to minimize the loss function by choosing values for the weights:

$$\min_{W_1,W_2} \sum_n \frac{1}{2}\left(y_n - \sum_i w_{2,i} f\left(w_{1,i} x_n\right)\right)^2$$

- The easy case is to differentiate it with respect to the weights of the output unit W_2

$$-y_n + \sum_i w_{2,i} f\left(w_{1,i} x_n\right)$$

- From this perspective, it is just a linear model, attempting to minimize squared error → the input here is $f(W_1 x)$, not x.

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Backpropagation

- Therefore, the gradient is

$$\nabla_{W_2} = -\sum_n \left(-y_n + \sum_i w_{2,i} f\left(w_{1,i} x_n\right) \right) f\left(W_1 x_n\right)$$

- This is exactly like the linear case.
- It shows how the output weights have to change to make the prediction better.
    - It is easy to measure how their changes affect the output.

Backpropagation

- When we move on the weights of the first layer the problem becomes more complicated.
- These weights usually are not trying to produce certain values (e.g., 0 or 1) but they are trying to produce values able to activate output units to return certain values.
  - So the change they want to make depends crucially on how the output layer interprets them.

How to train a NN?

- How could we change the weights to make our Loss Function smaller?
- Think of the neural net as a function $f: x \rightarrow y$
- f is a complex computation involving many weights W_k
- Given the structure, the weights "define" the function f (and therefore define our model)
- Loss Function is $J\big(y, f(x)\big)$

How to train a NN?

- How could we change the weights to make our Loss Function smaller?
- Think of the neural net as a function $f: x \rightarrow y$
- f is a complex computation involving many weights W_k
- Given the structure, the weights "define" th ne our model)
- Loss Function is $J\big(y, f(x)\big)$

The goal is to change the weights to make the loss function smaller.

How to train a NN?

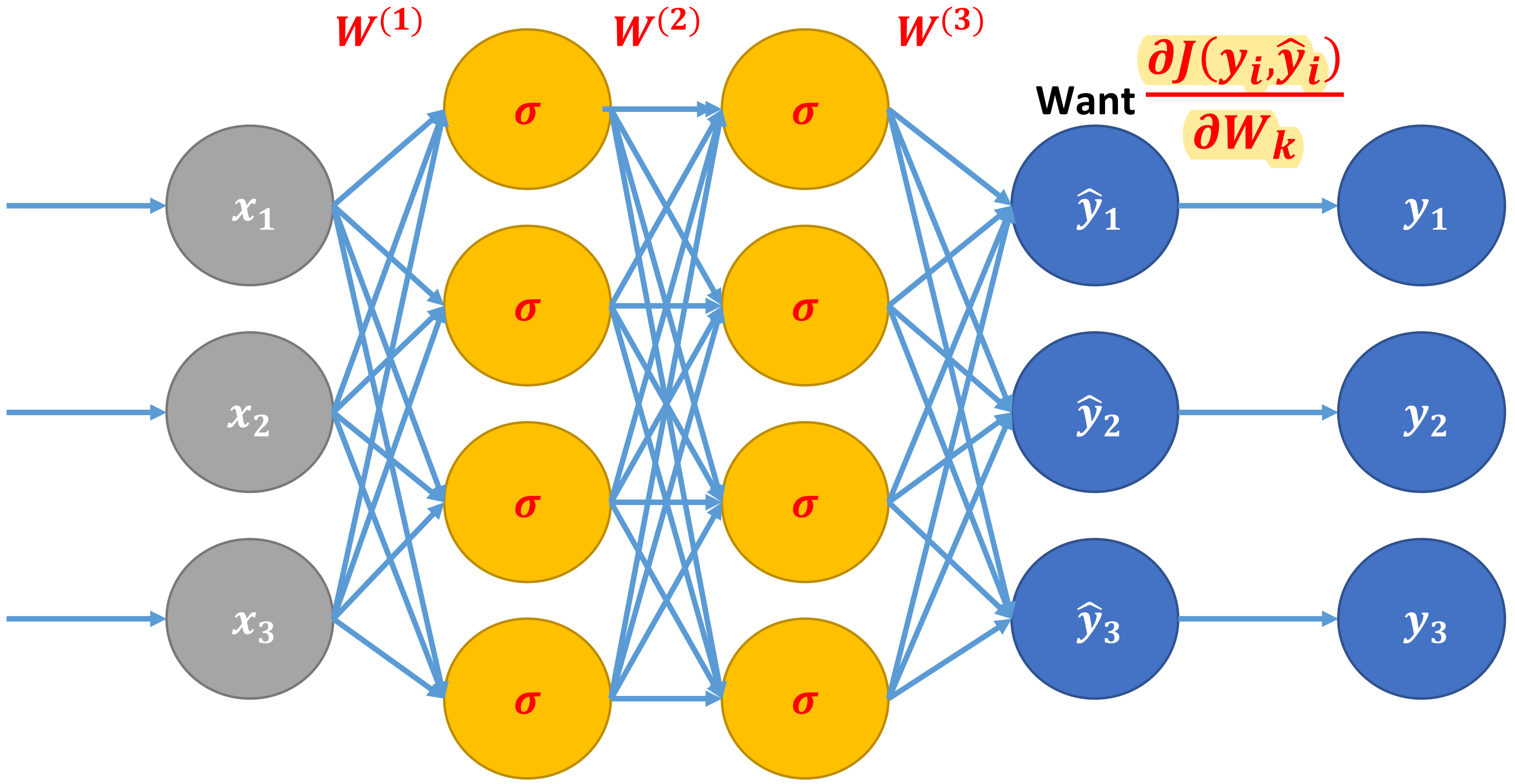
- Get $\frac{\partial J}{\partial W_k}$ for every weight in the network.
- This tells us what direction to adjust each W_k if we want to lower our loss function.
- Make an adjustment and repeat!

How to train a NN?

- Get $\frac{\partial J}{\partial W_k}$ for every weight in the network.
- This tells us what direction to adjust each W_k if we want to lower our loss function.
- Make an adjustment and repeat!

With this setting, from an abstract mathematical standpoint, there is no difference between this and classical ML gradient descent.
It's just that the function involved is much more complicated, and computation of gradients is mathematically and computationally more challenging.

Feedforward Neural Network



So we want to be able to compute the partial derivative of the loss function w.r.t. the weights W_k

An Example...

Consider Cross-Entropy (Log-loss)

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot x$$

- Recall that: $\sigma'(z) \; = \; \sigma(z)(1 - \sigma(z))$
- Though they appear complex, above are easy to compute!

An Example...

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot x$$

Early layers reuse computation from later layers → **BACK** propagation.
E.g., the gradient of $W^{(1)}$ uses the gradient of $W^{(2)}$

- Recall that: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Though they appear complex, above are easy to compute!

An Example…

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

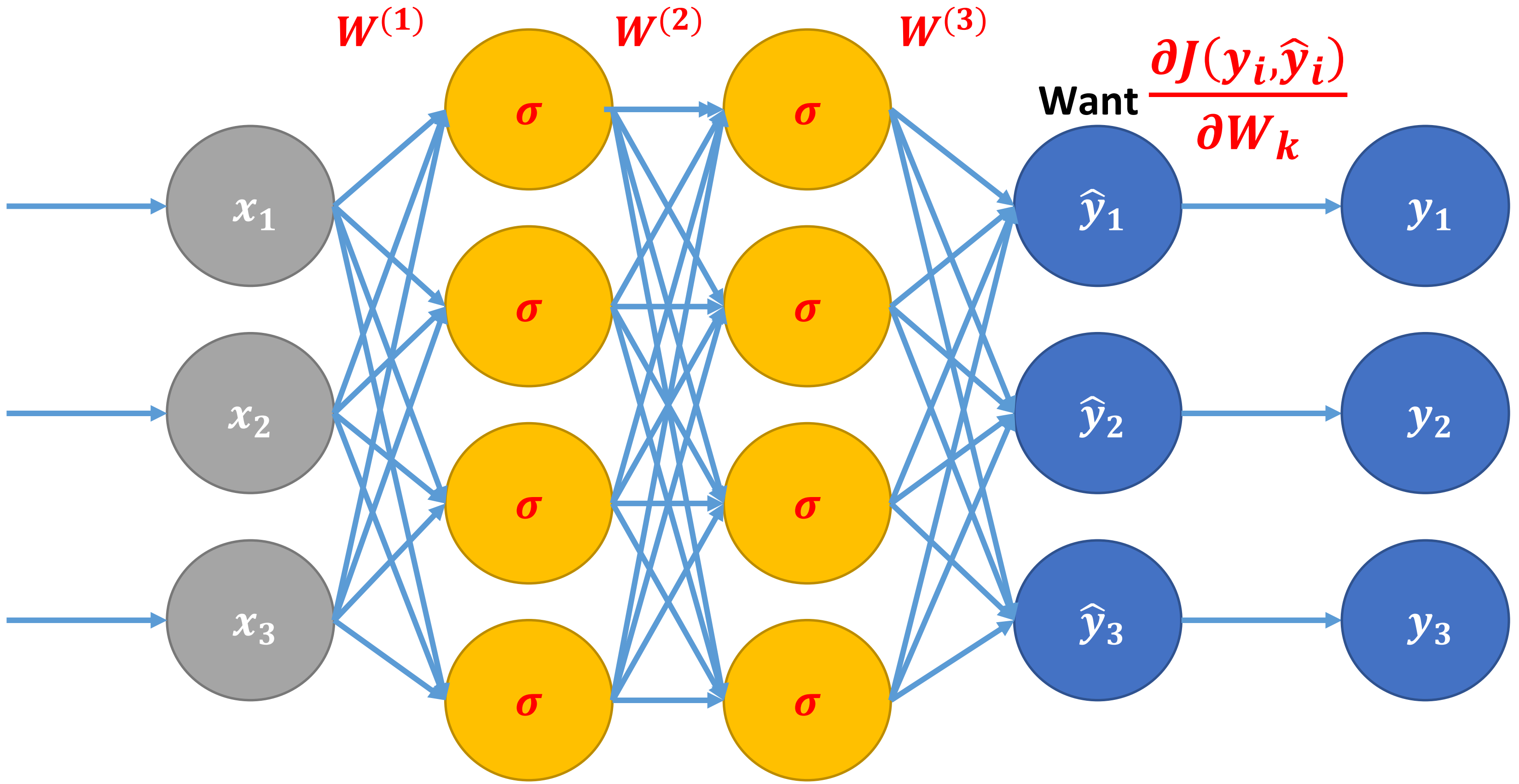
$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot x$$

Early layers have more terms → smaller numbers → vanishing gradient

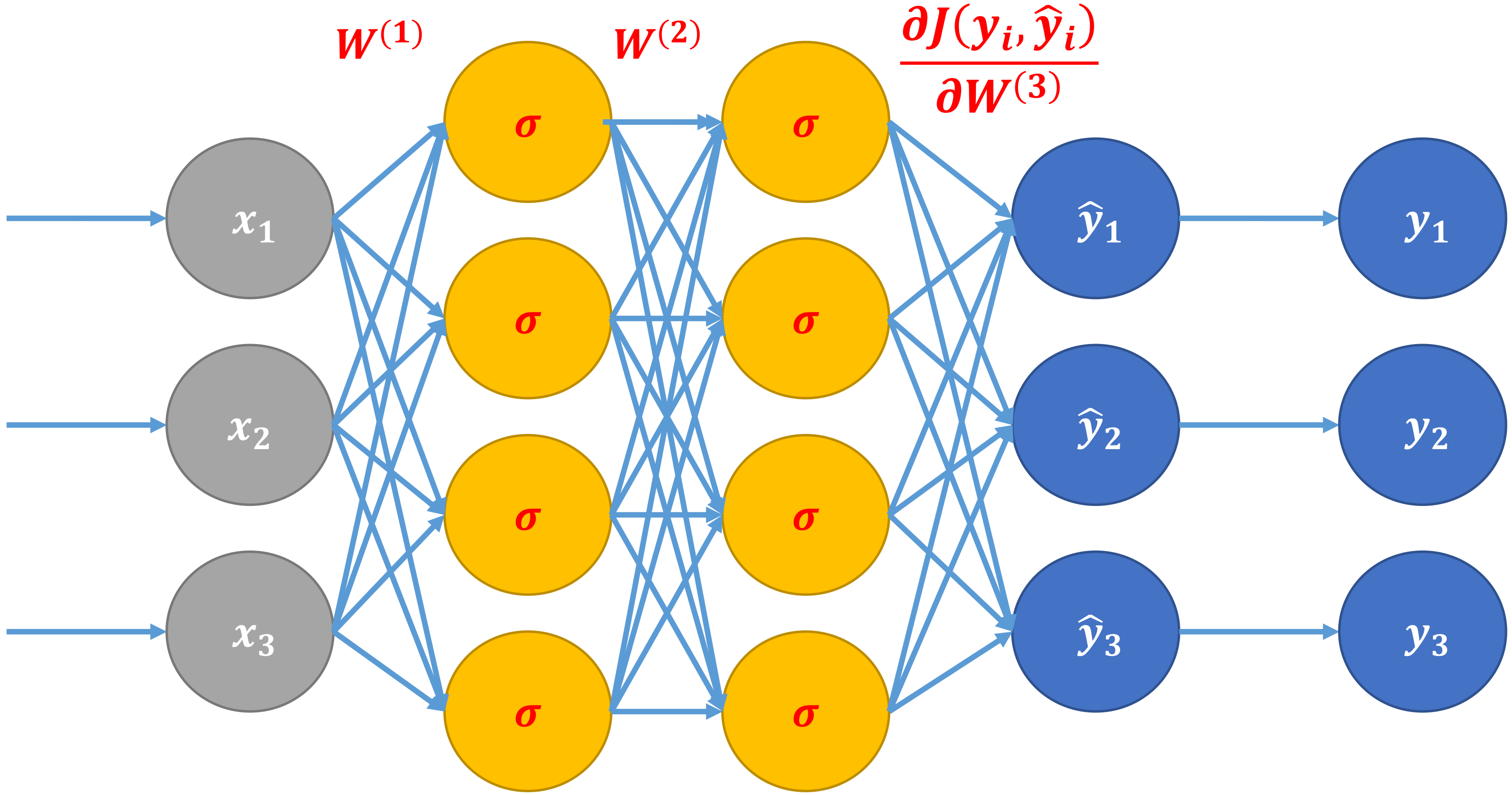
- Recall that: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Though they appear complex, above are easy to compute!

Backpropagation



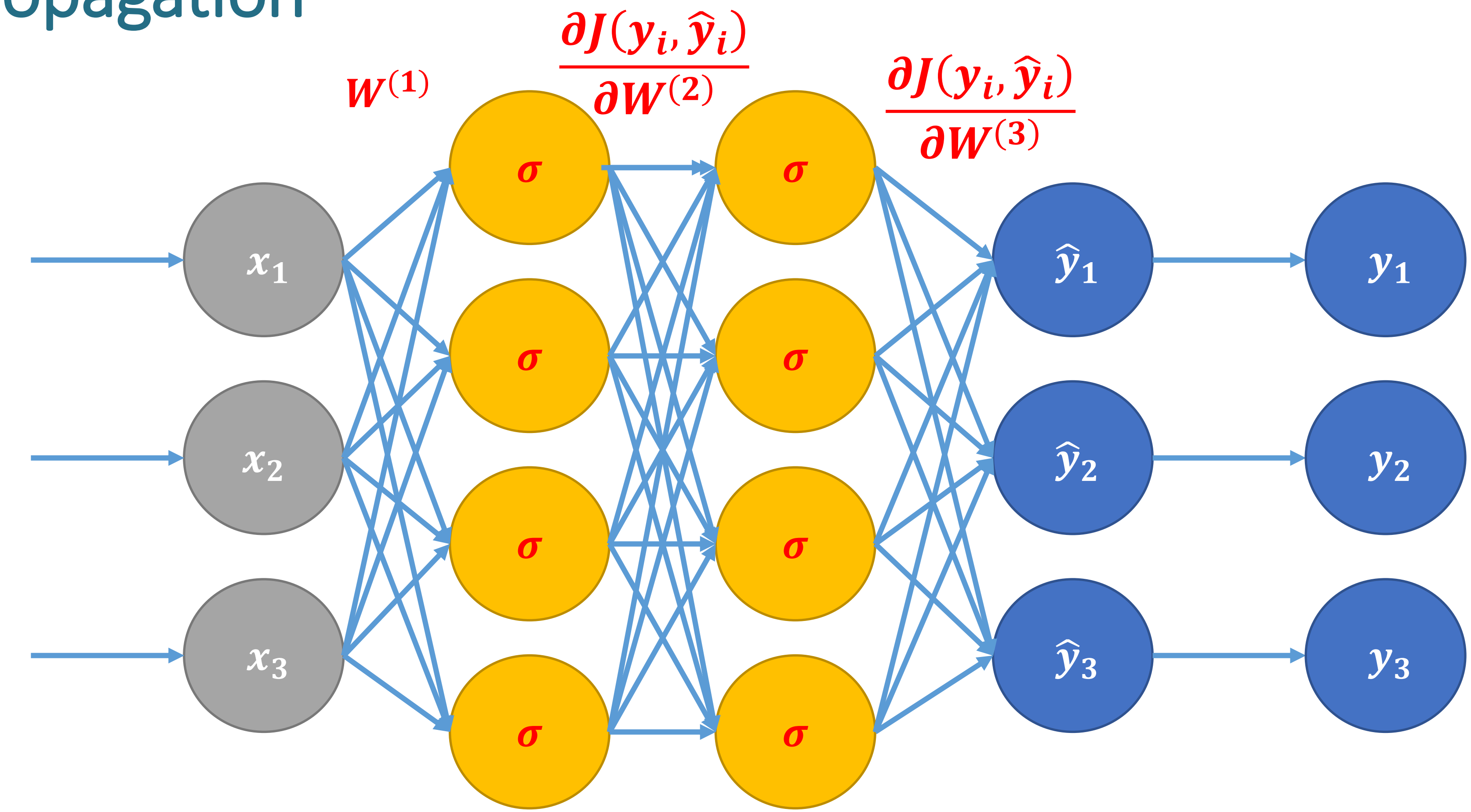
So we want to be able to compute the partial derivative of the loss function w.r.t. the weights W_k

Backpropagation



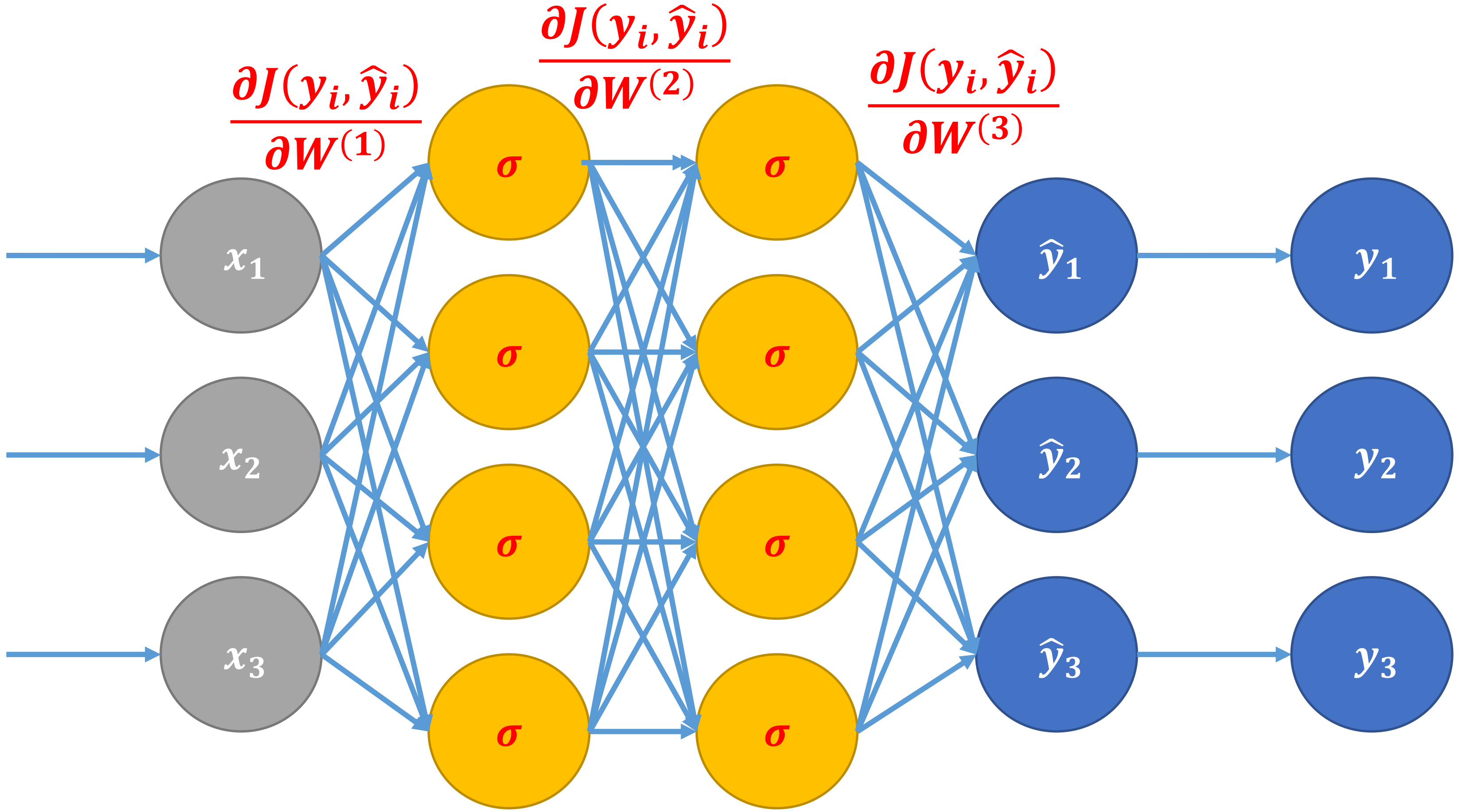
We first compute this, math turns out to be simpler for the last layer.

Backpropagation



Using that value, we compute the partial derivative for the previous layer → **Back propagation!**

Backpropagation



Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

How have we trained before?

- **Gradient Descent**.
  For most classical ML algorithms, the training happens here.

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021

Gradient descent

Update parameters by taking a step in the opposite direction

- Once we have the gradient, we just take a step in the opposite direction.
- Specifically; $w = w - learning\_rate * gradient$
- This part is the same as in ML.