



Sapere utile

IFOA
Istituto Formazione Operatori Aziendali

BIG DATA e Analisi dei Dati

Mauro Bellone,
Robotics and AI researcher

bellonemauro@gmail.com
www.maurobellone.com

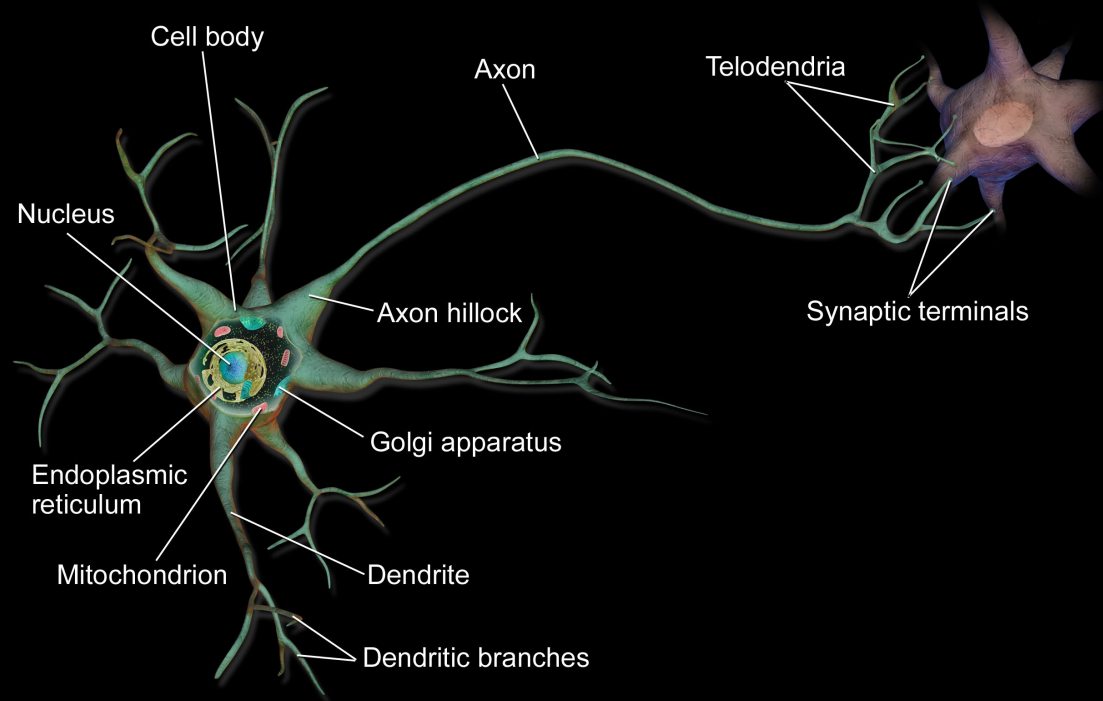
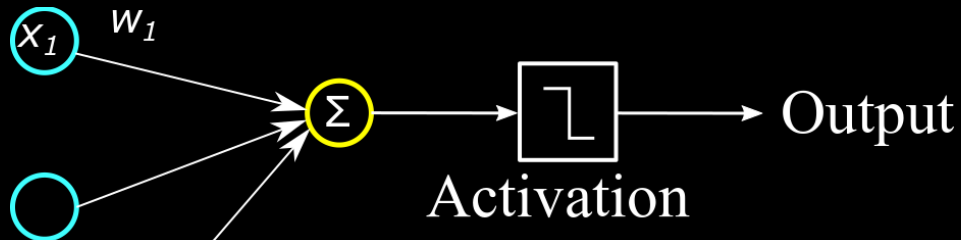
Obiettivo

- ✓ Funzioni di costo
- ✓ Tecniche di regolarizzazione delle reti
- ✓ Gradient decent
- ✓ Tutorial learning con percettrone

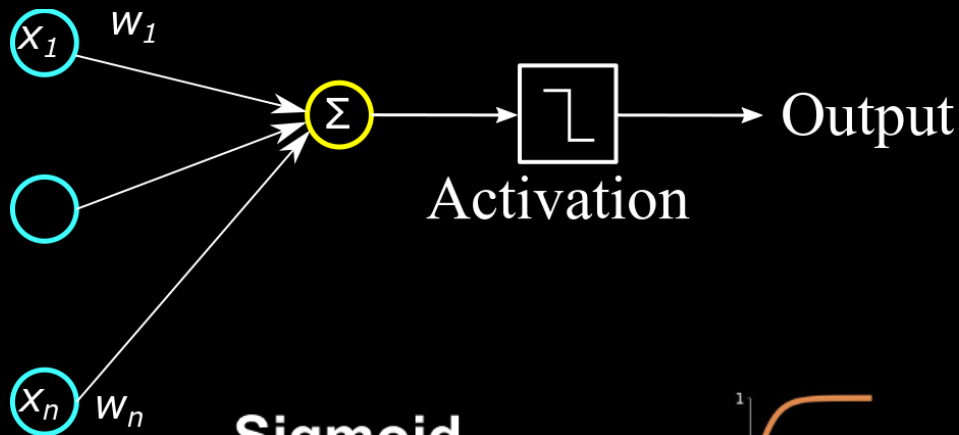
Link utili

- LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), pp.436-444. <https://www.deeplearningbook.org/>
- Eli Stevens, Luca Antiga, Thomas Viehmann, “Deep Learning With Pytorch: Build, Train, and Tune Neural Networks Using Python Tools” 2020
<https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf>
- Michael Nielsen - Dec 2019 – “neural network and deep learning”
<http://neuralnetworksanddeeplearning.com/index.html>
- ✓ <https://stanford.edu/~shervine/teaching/cs-229/>
- ✓ <https://cs231n.github.io/convolutional-networks/>

Il percettore



Funzioni di attivazione

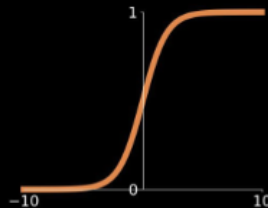


$$out = \sum x_i w_i * A$$

A è detta funzione di attivazione

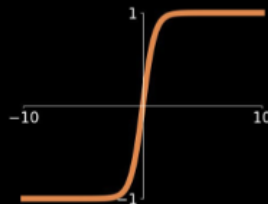
Sigmoid

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



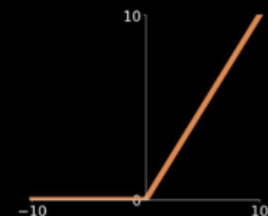
tanh

$$\tanh(x)$$



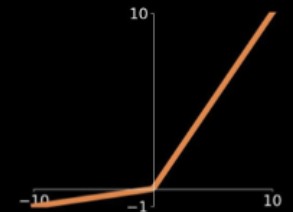
ReLU

$$\max(0, x)$$



Leaky ReLU

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

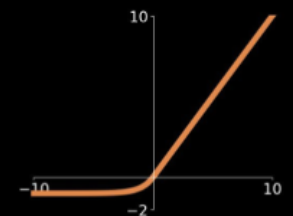


Maxout

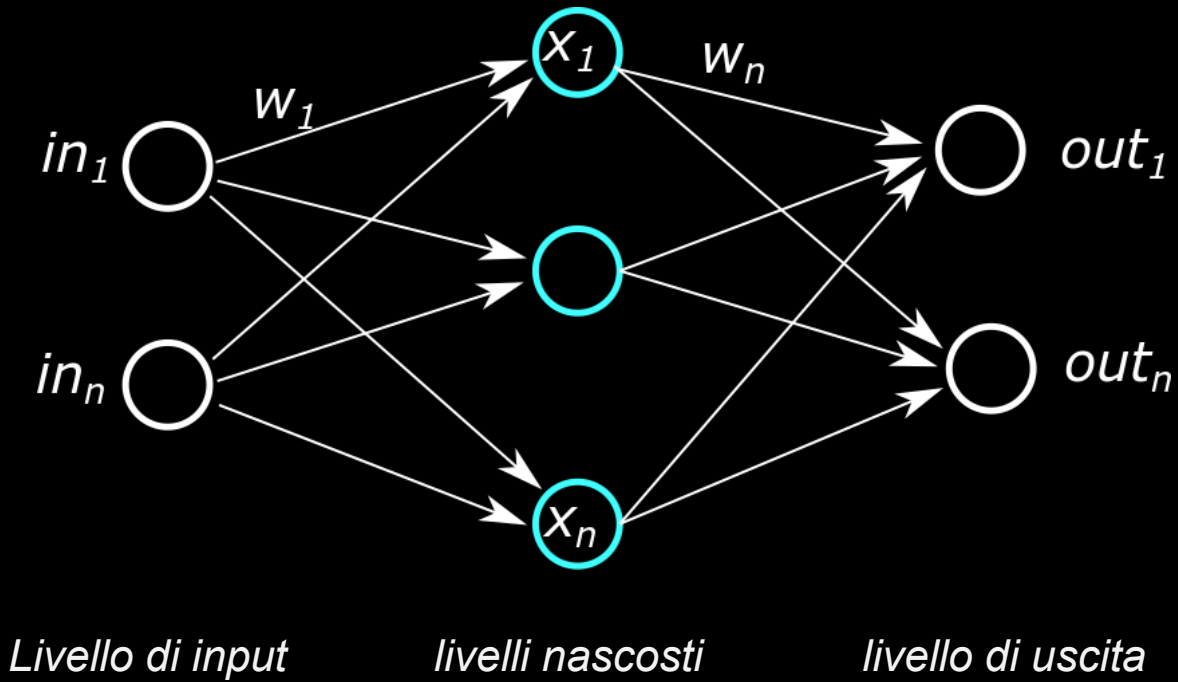
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

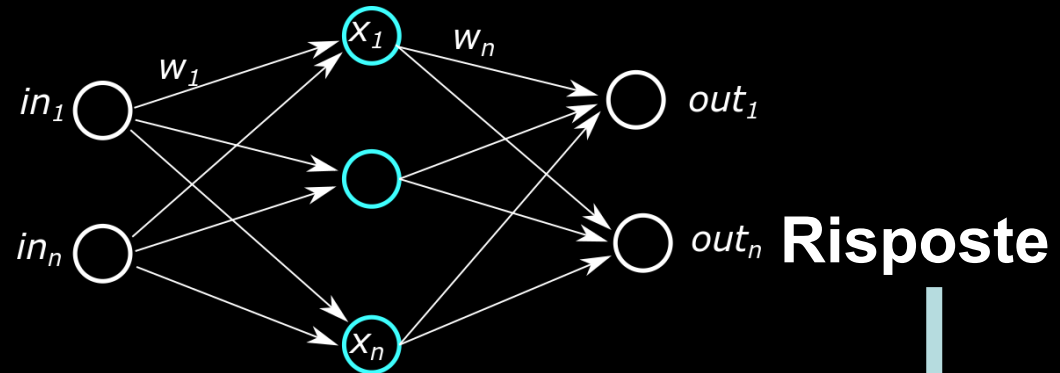


Reti neurali lineari



Apprendimento supervisionato

Training data



Supervisor

Validation data



Multilayer Feedforward Networks are Universal Approximators

KUR' HORNİK

Technische Universität Wien

MAXWELL STINCHCOMBE AND HALBER WHITE

University of California, San Diego

(Received 16 September 1988; revised and accepted 9 March 1989)

Abstract—*This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.*

Reti feedforward multilivello con almeno un livello nascosto aventi una arbitraria funzione di attivazione “squashing” sono in grado di approssimare qualunque funzione misurabile

Problemi di ottimizzazione

$$\begin{array}{ll} \min_x & c(x) \\ \text{soggetto a} & g_i(x) \leq 0 \text{ con } i = 1, 2, \dots, n \\ & h_j(x) = 0 \text{ con } j = 1, 2, \dots, m \end{array}$$

Dove

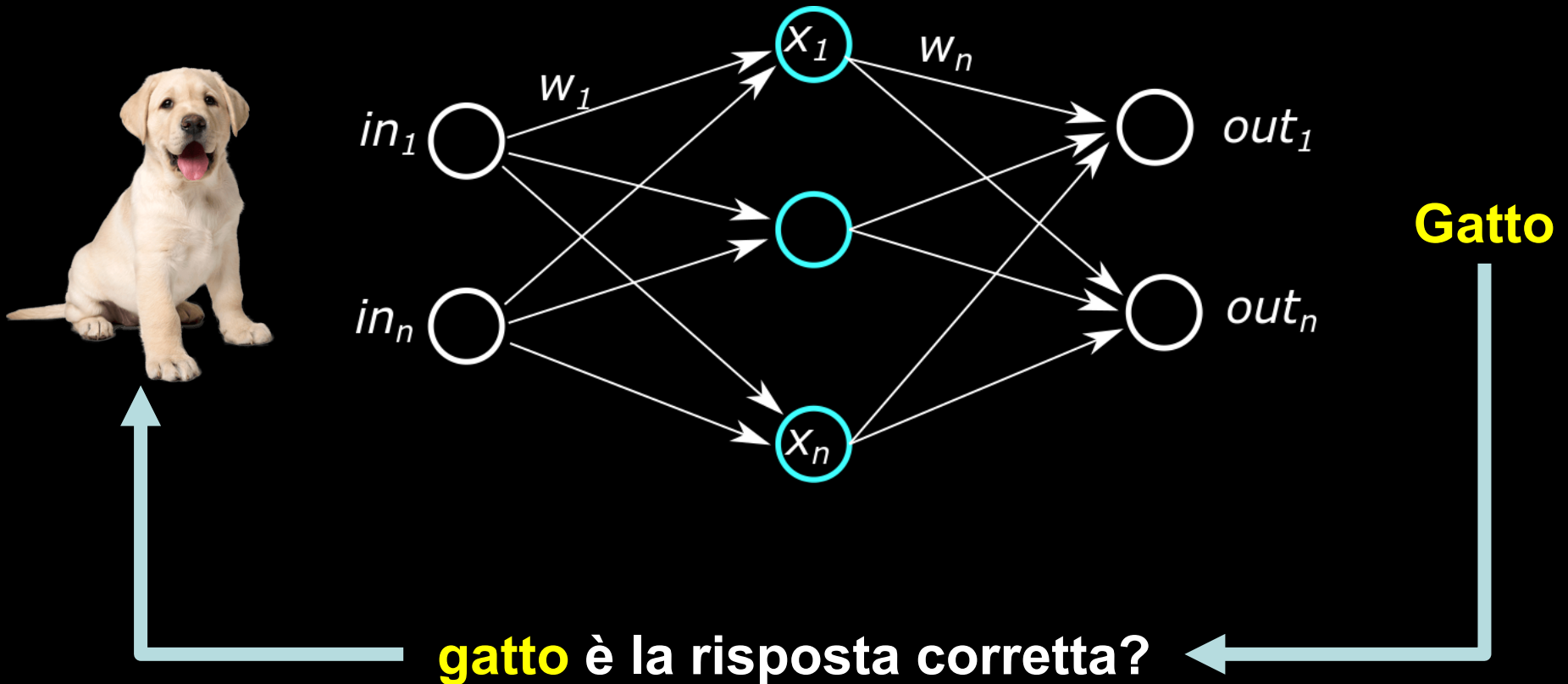
$$c: \mathbb{R}^n \rightarrow \mathbb{R}$$

si dice funzione obiettivo (cost function)

$$g_i(x) \leq 0 \text{ e } h_j(x) = 0$$

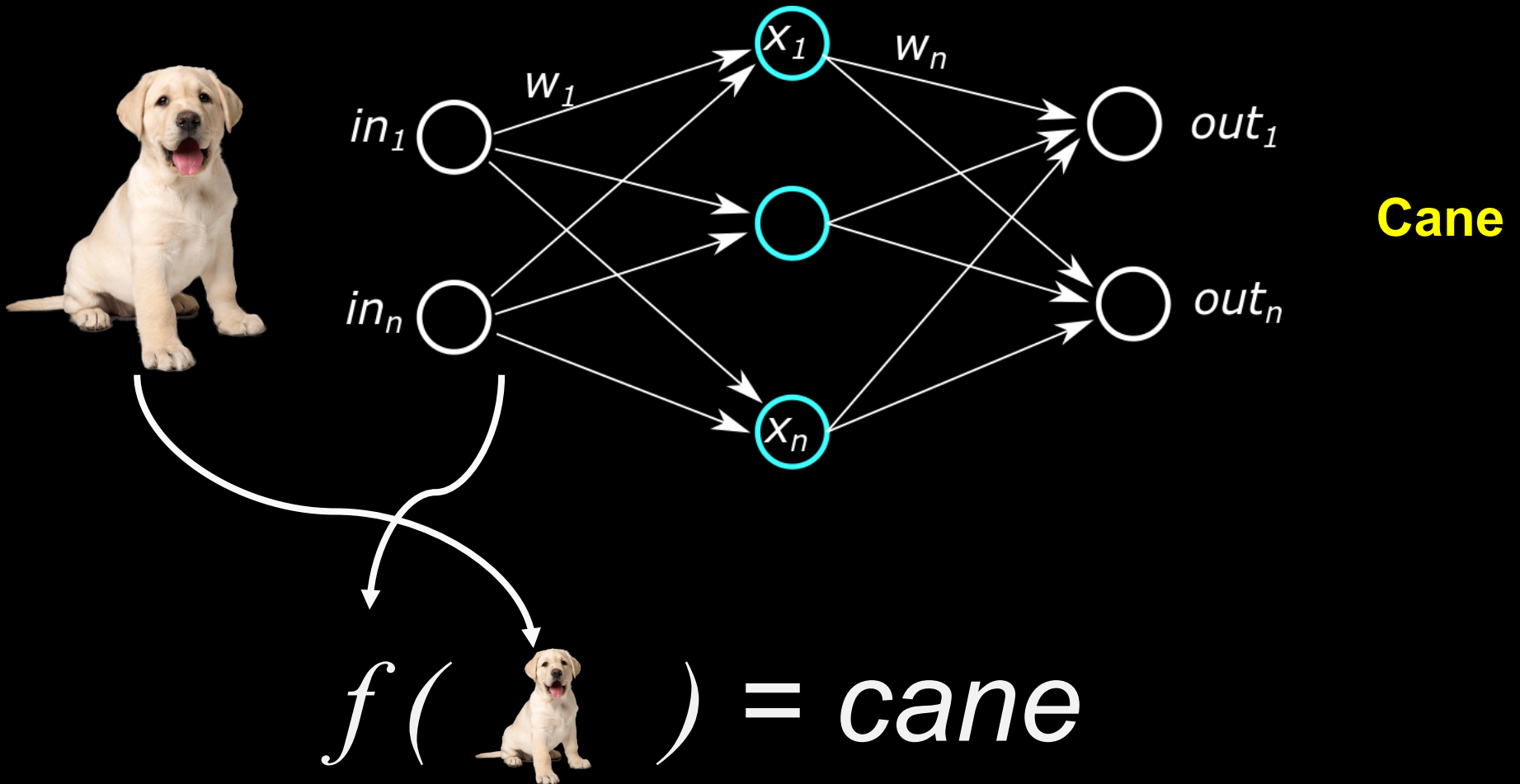
sono vincoli

Procedure di apprendimento

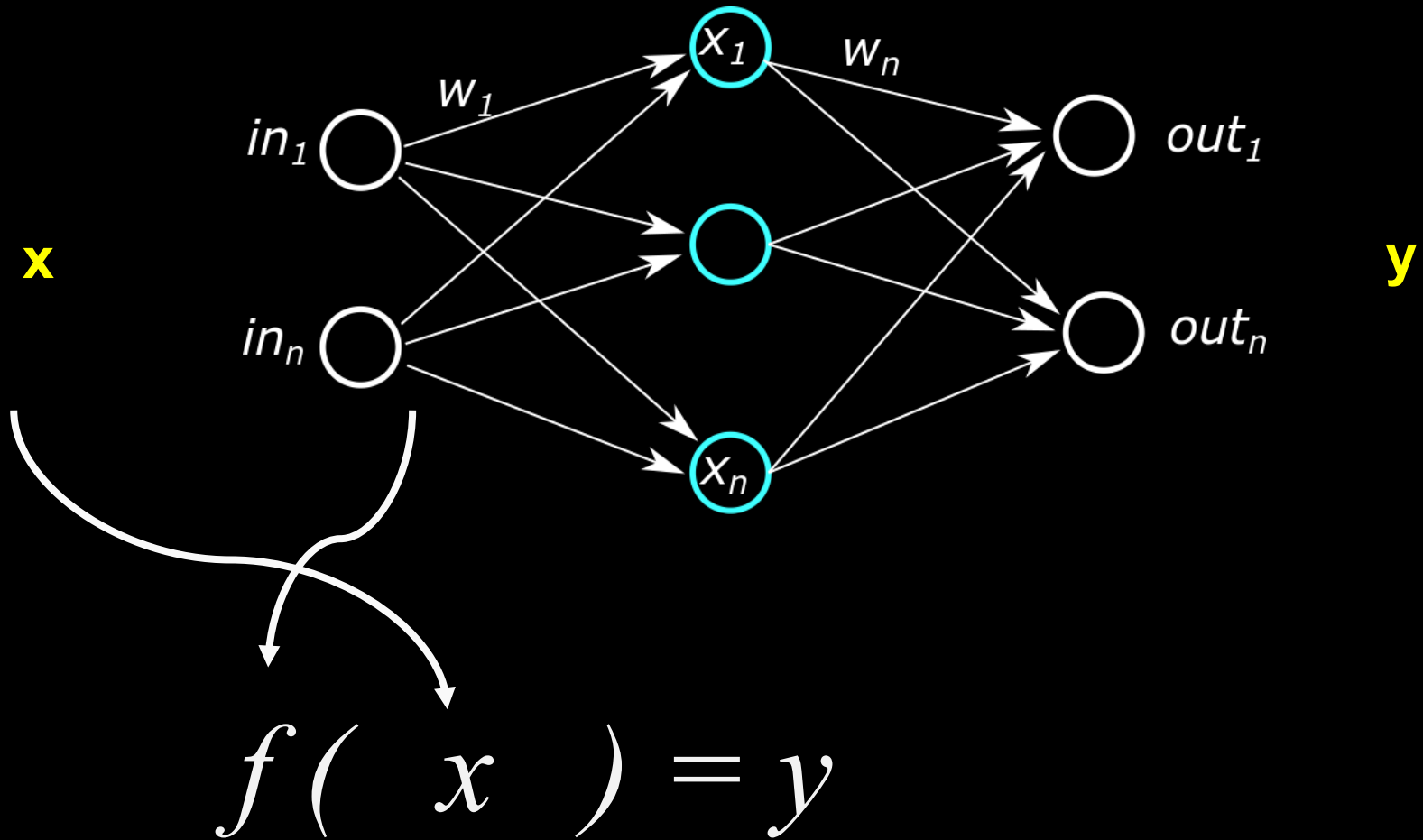


Se SI \rightarrow ok, la rete sta funzionando bene
se NO \rightarrow male, I pesi hanno bisogno di essere aggiustati

Procedure di apprendimento



Procedure di apprendimento



Problemi di ottimizzazione

$$\begin{array}{ll} \min_x & c(x) \\ \text{soggetto a} & g_i(x) \leq 0 \text{ con } i = 1, 2, \dots, n \\ & h_j(x) = 0 \text{ con } j = 1, 2, \dots, m \end{array}$$

Dove

$$c: \mathbb{R}^n \rightarrow \mathbb{R}$$

si dice funzione obiettivo (cost function)

$$g_i(x) \leq 0 \text{ e } h_j(x) = 0$$

sono vincoli

$$c(x) = \widehat{f(x)} - y$$

stima **vero**

Problemi di ottimizzazione

$$\begin{array}{ll} \min_x & c(x) \\ \text{soggetto a} & g_i(x) \leq 0 \text{ con } i = 1, 2, \dots, n \\ & h_j(x) = 0 \text{ con } j = 1, 2, \dots, m \end{array}$$

Dove

$$c: \mathbb{R}^n \rightarrow \mathbb{R}$$

si dice funzione obiettivo (cost function)

$$g_i(x) \leq 0 \text{ e } h_j(x) = 0$$

sono vincoli

$$c(x) = \widehat{f(x)} - y$$

Errore tra previsione e dato vero

stima **vero**

Funzioni di costo

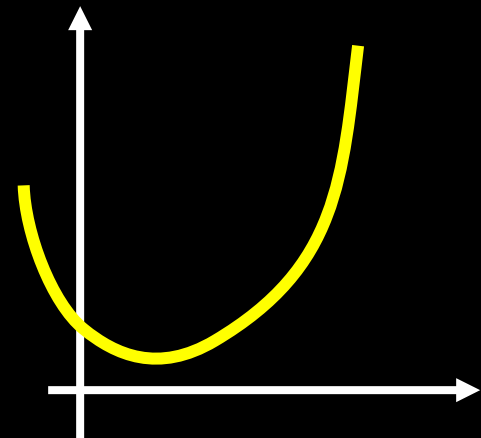
- ✓ La funzione di costo è una misura di quanto il modello è errato in termini di abilità di stima della relazione ingresso-uscita

Funzioni di costo

- ✓ La funzione di costo è una misura di quanto il modello è errato in termini di abilità di stima della relazione ingresso-uscita
- ✓ Tipicamente si esprime come una misura della differenza tra il valore predetto e il valore vero
 - errore
 - verosimiglianza
 - altre ...

Funzioni di costo

- ✓ La funzione di costo è una misura di quanto il modello è errato in termini di abilità di stima della relazione ingresso-uscita
- ✓ Tipicamente si esprime come una misura della differenza tra il valore predetto e il valore vero
 - errore
 - verosimiglianza
 - altre ...

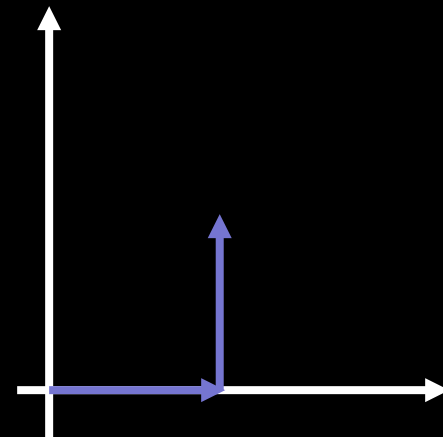
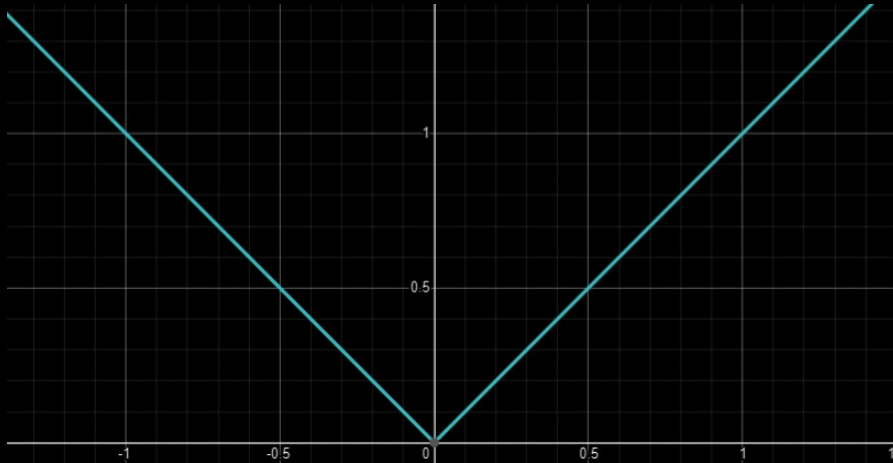


Una buona funzione di costo è positiva, differenziabile e ha un minimo assoluto

Funzioni di costo - MAE

- ✓ MAE – Mean absolute error o errore assoluto medio

$$MAE_{loss} = E[|\hat{y} - y|]$$

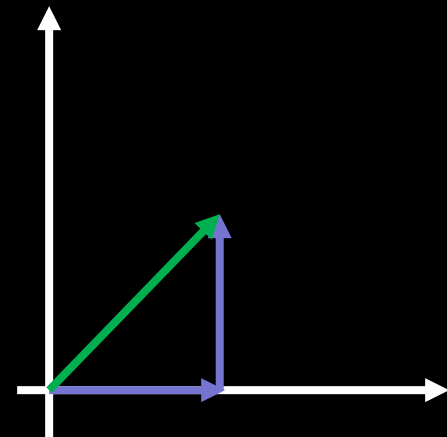
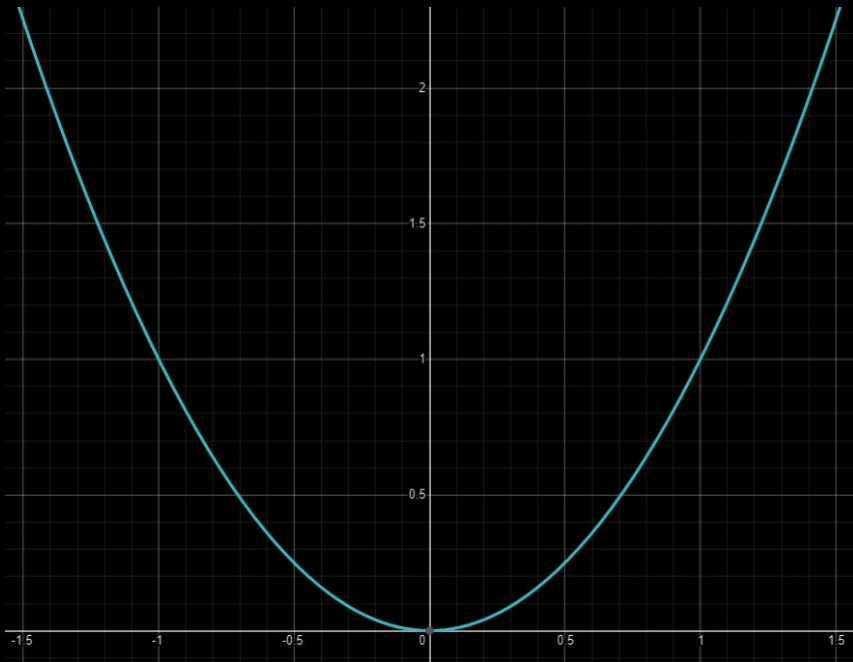


Anche chiamata L1-loss (è la norma L1)

Funzioni di costo - MSE

- ✓ MSE - Mean squared error o errore quadratico medio

$$MSE_{loss} = E[(\hat{y} - y)^2]$$

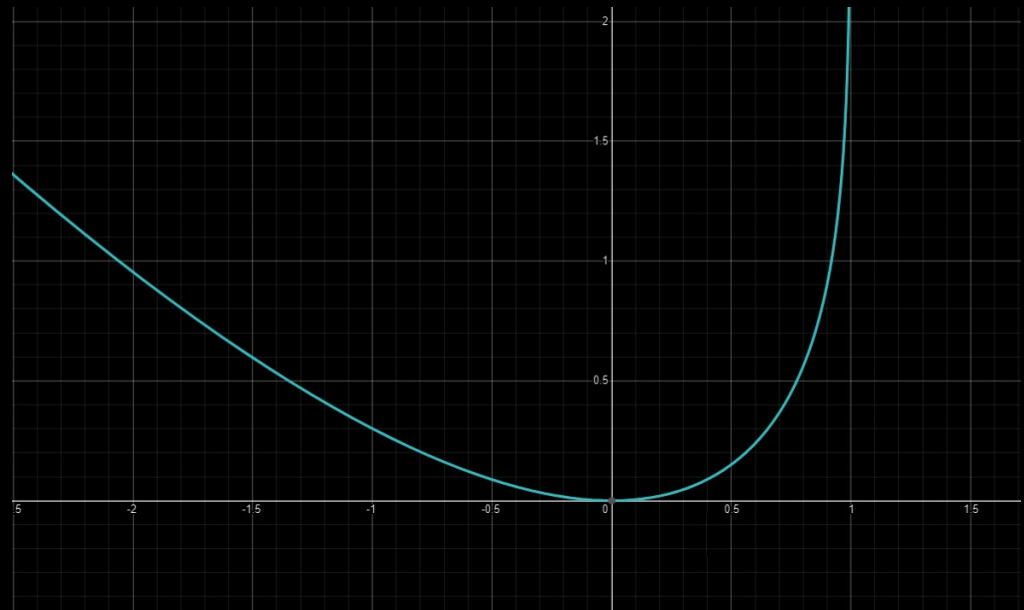


Anche chiamata L2-loss (è la norma L2) o norma Euclidea

Funzioni di costo – Cross entropy loss

- ✓ L'entropia incrociata (cross entropy) misura la verosimiglianza, quindi la probabilità che la stima effettuata sia corretta

$$CE_{loss} = - \sum_x p(x) \log q(x)$$



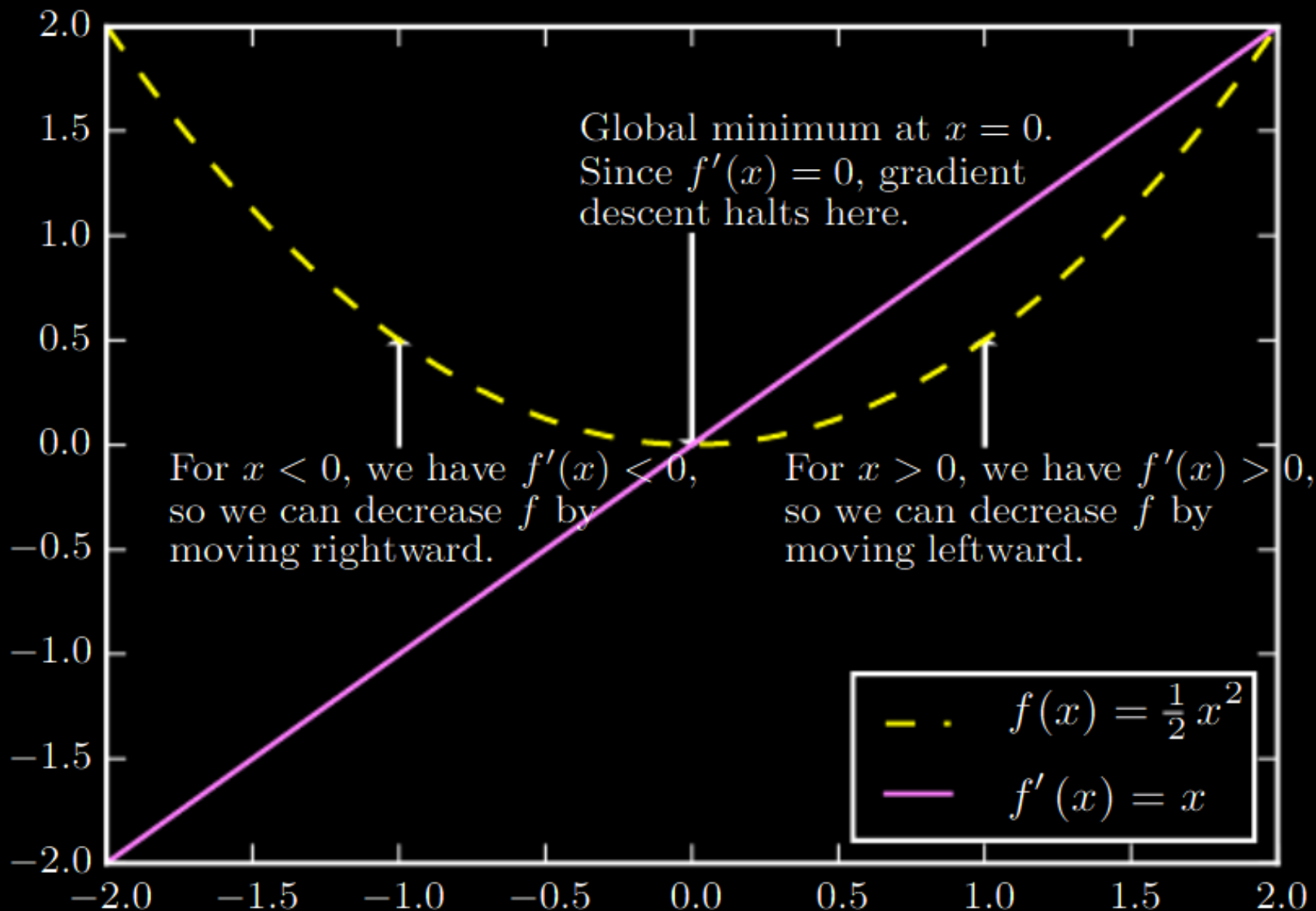
Metodo di minimizzazione del gradiente

Il metodo del gradiente è un algoritmo di risoluzione di problemi di minimizzazione (ottimizzazione) nei quali la direzione di ricerca è definita dal gradiente della funzione obiettivo in un determinato punto.

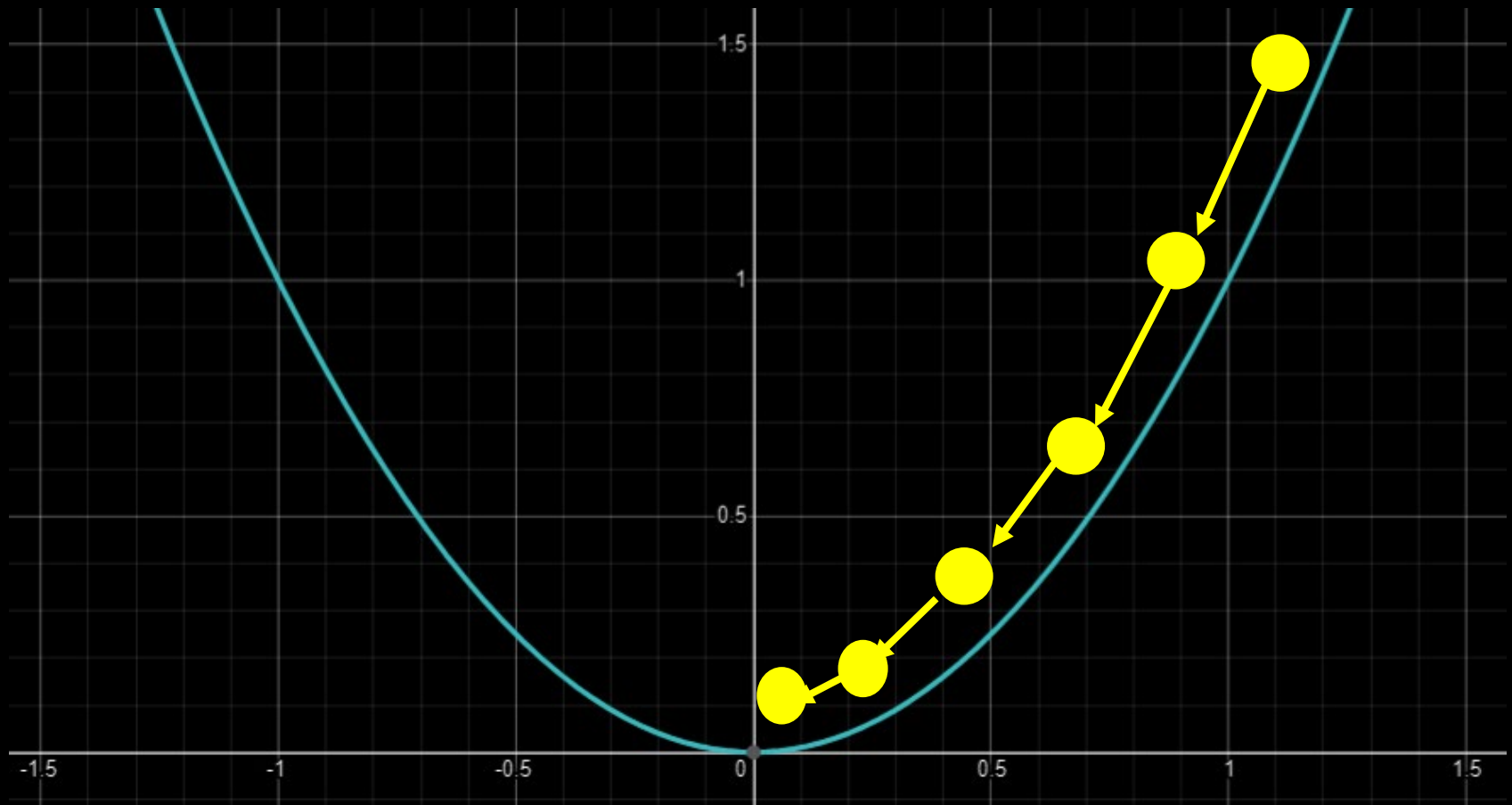
Esempi:

- ✓ Gradient descent
- ✓ Stochastic gradient descent
- ✓ Conjugate gradient

Metodo di minimizzazione del gradiente



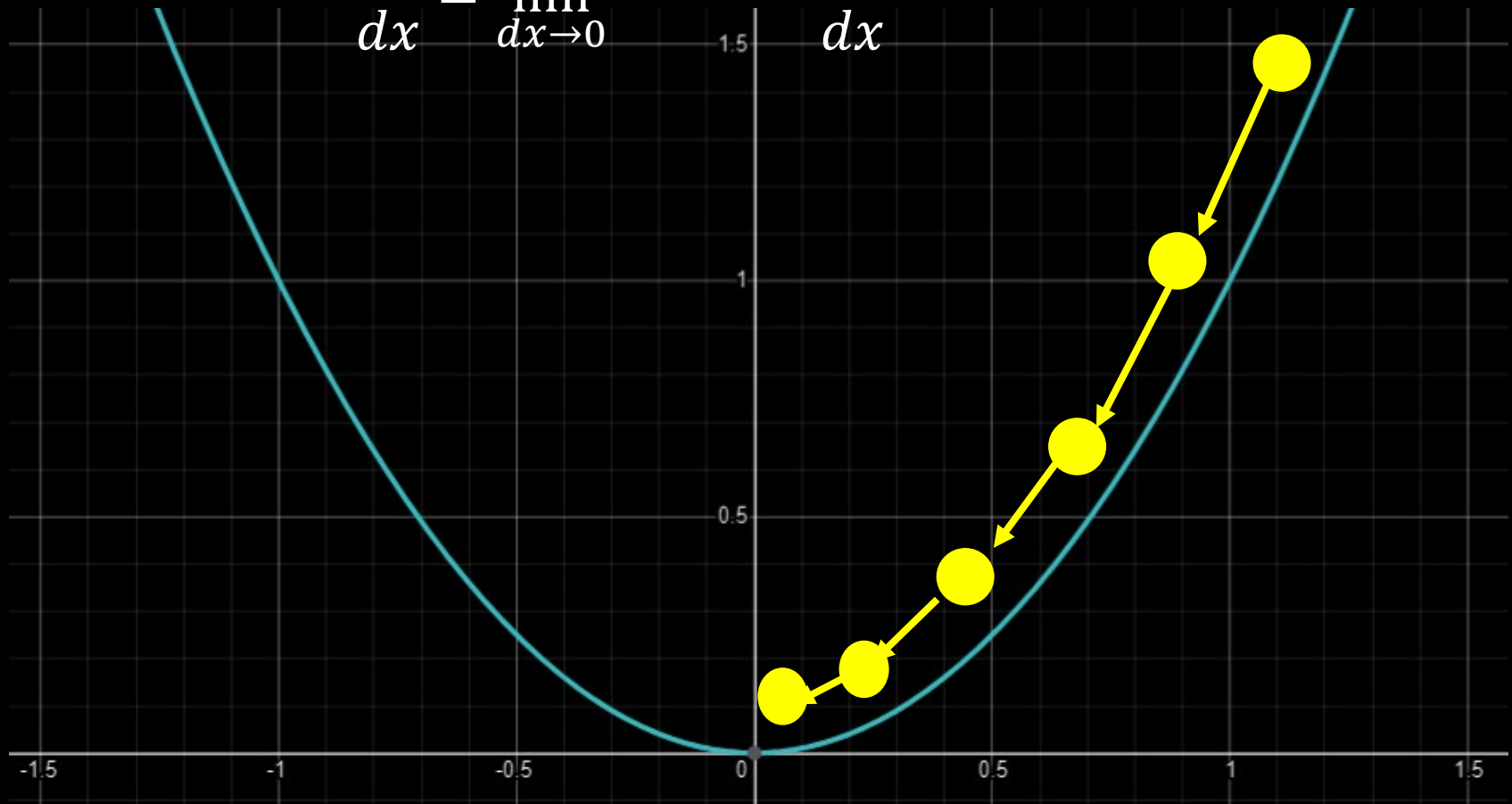
Metodo di minimizzazione del gradiente



Metodo di minimizzazione del gradiente

Formula della derivata (rapporto incrementale)

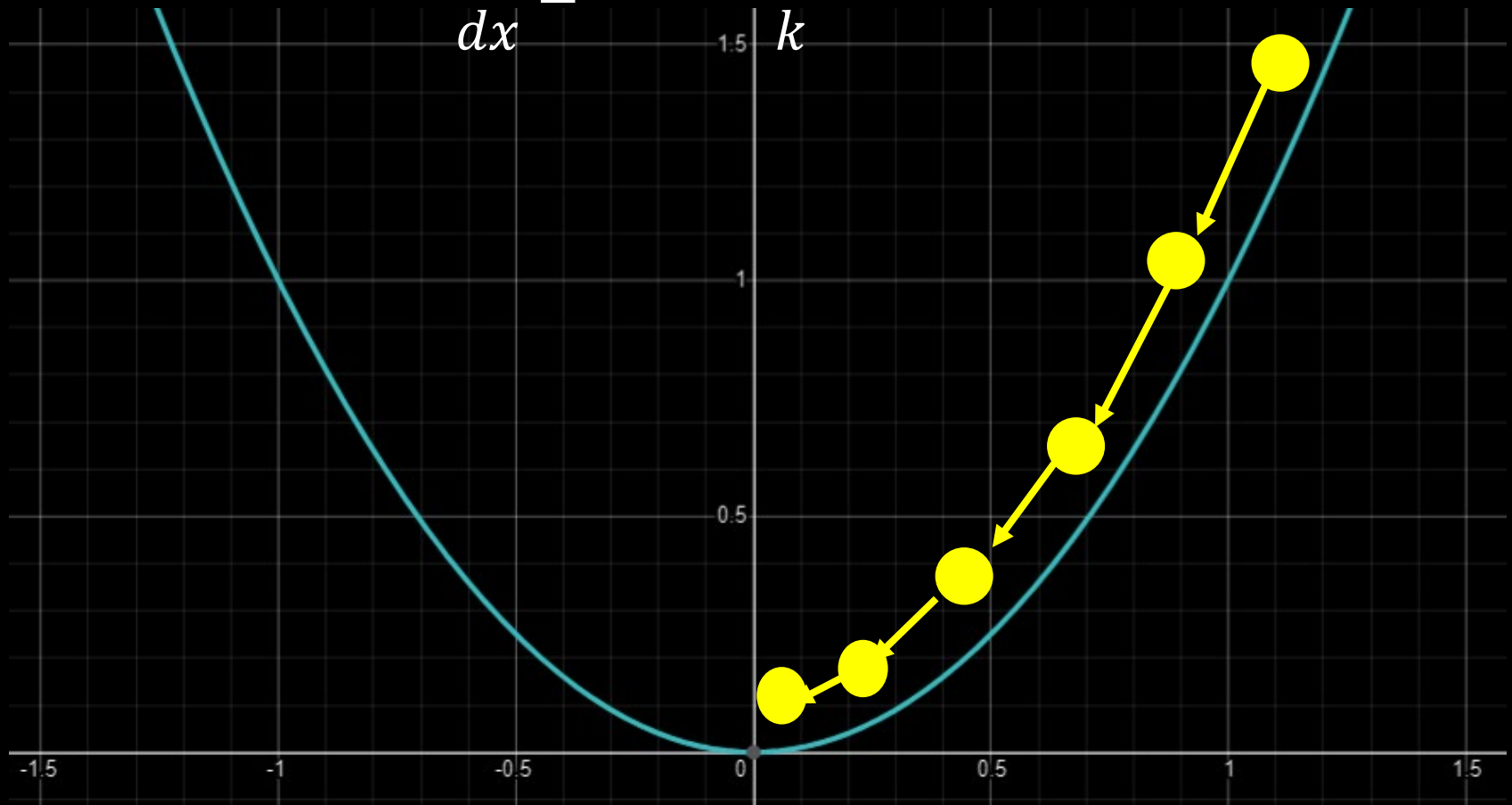
$$\frac{dy}{dx} = \lim_{dx \rightarrow 0} \frac{(y(x + dx) - y(x))}{dx}$$



Metodo di minimizzazione del gradiente

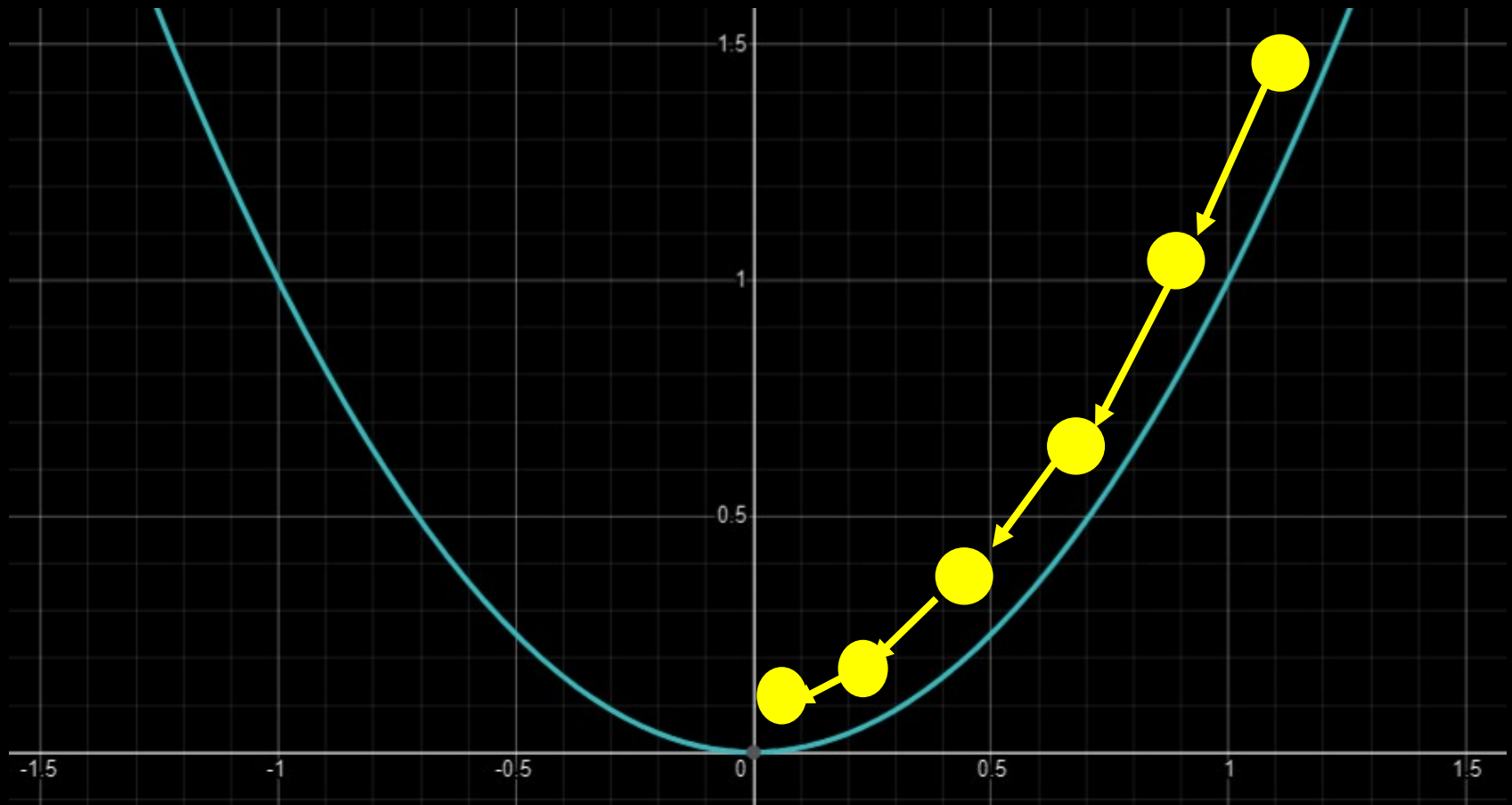
Formula della derivata (rapporto incrementale)

$$\frac{dy}{dx} = \frac{y(x + k) - y(x)}{k}$$



Metodo di minimizzazione del gradiente

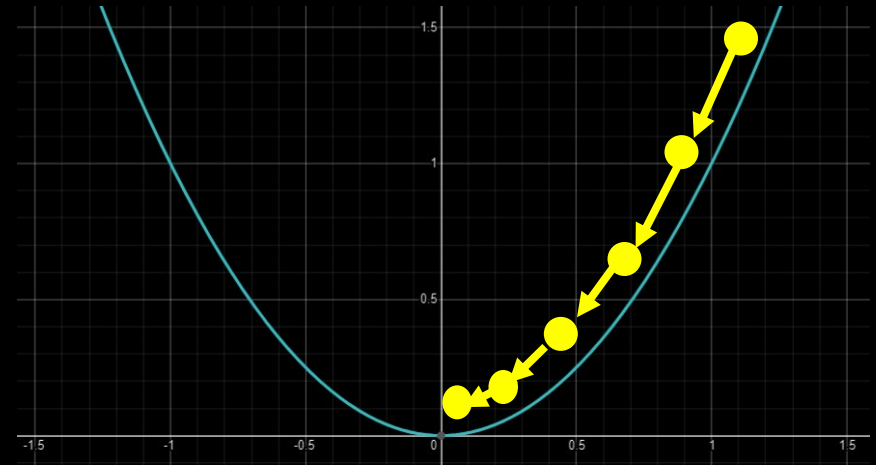
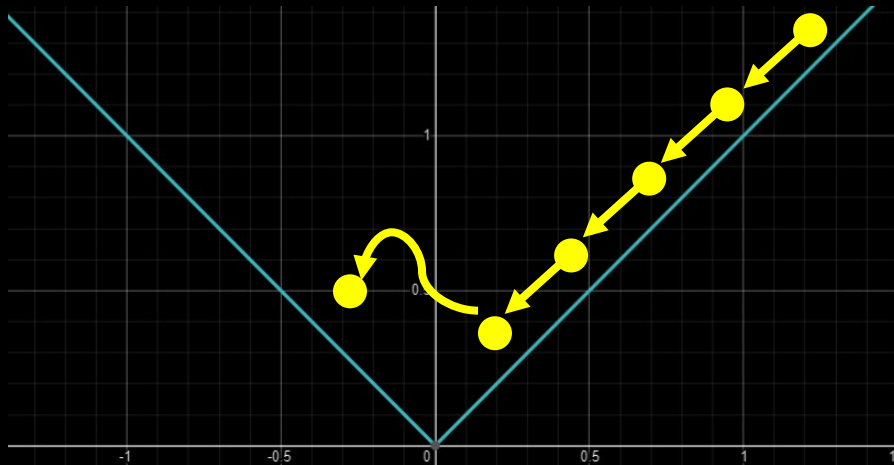
Il gradiente è semplicemente una differenza
il segno (+ o -) indica la direzione



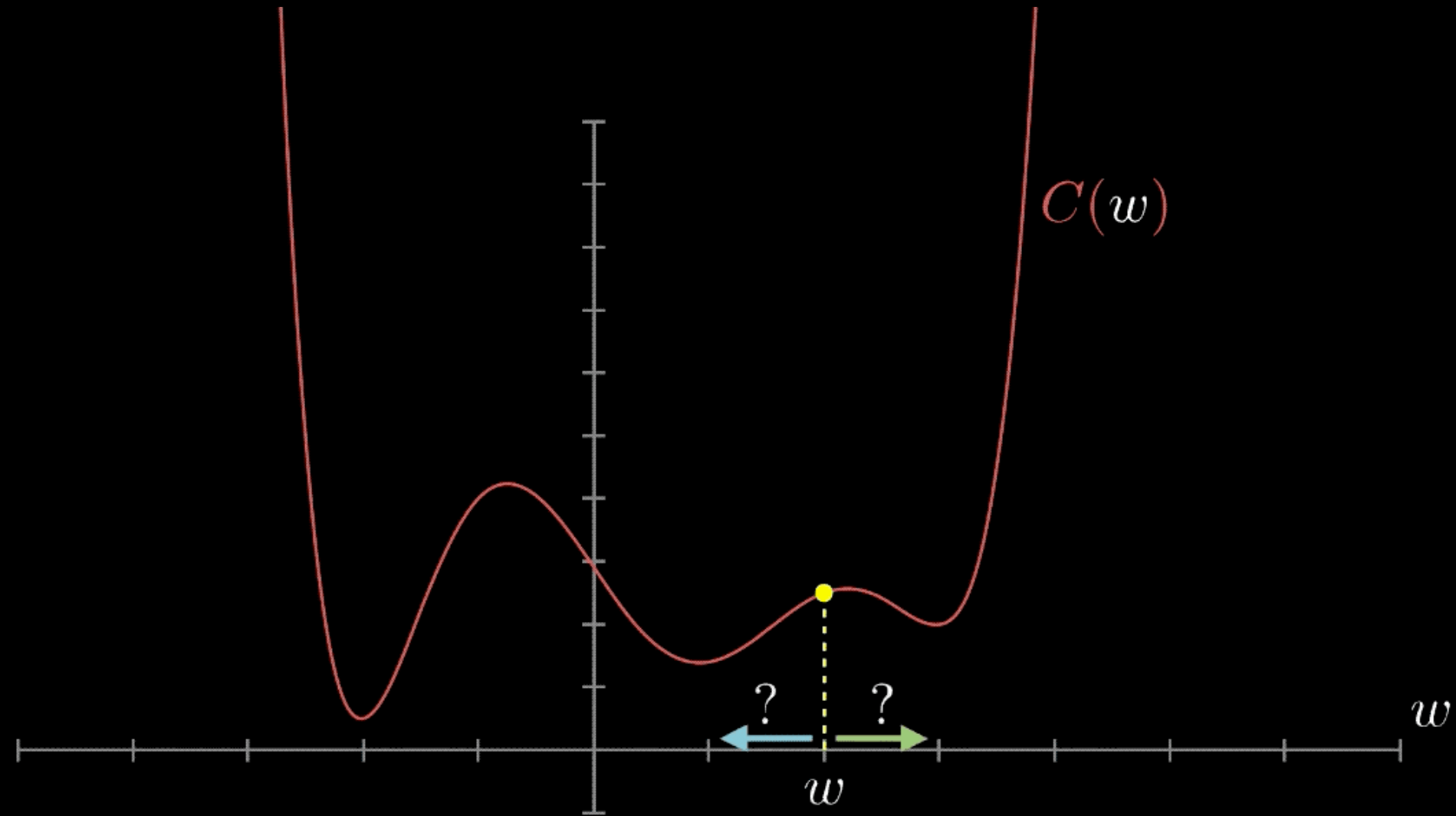
Metodo di minimizzazione del gradiente

- ✓ La scelta della funzione di costo è fondamentale per la convergenza dell'algoritmo.

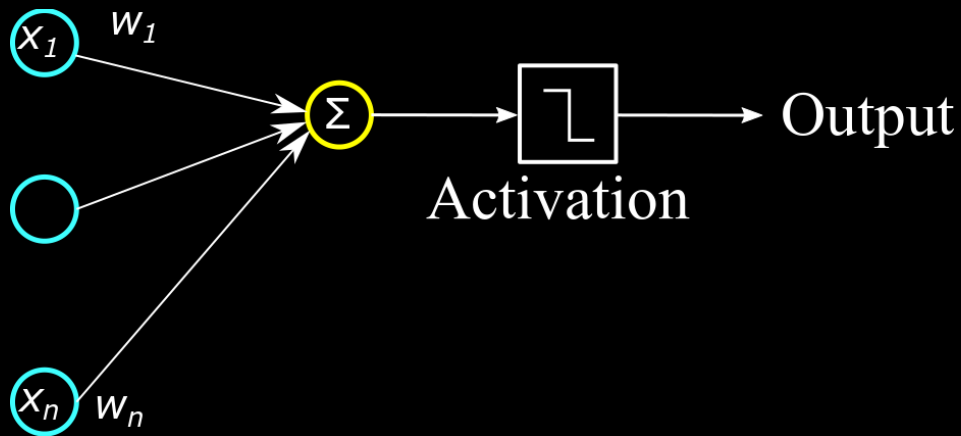
Es. MAE vs MSE



Problema dei minimi locali



Procedure di apprendimento



$$y = \varphi \left(\sum \omega_i x + b_i \right)$$

$$\omega_{new} = \omega_{old} - \boxed{l_r} \left(\frac{\delta \text{error}}{\delta \omega} \right)$$

Learning rate
 $\text{error} = y - y_{des}$

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $\mathcal{C} > \epsilon$ )  
    for ( $x \in S$ )  
         $\hat{y} = f(x)$                                 > calcolo la stima del modello  
         $c_i = \mathcal{C}(\hat{y}, y)$                         > calcolo la loss sul modello  
        backward( $f(x), c_i$ )                        > calcolo le derivate sul modello  
        optimizer()                                > aggiorno i pesi del modello  
    end  
end
```

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $\mathcal{C} > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = \mathcal{C}(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Prima di iniziare la procedura è necessario definire i dati di training e il metodo di caricamento dei dati (data loader)

> calcolo la stima del modello

> calcolo la loss sul modello

> calcolo le derivate sul modello

> aggorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

init (w_i)

while ($\mathcal{C} > \epsilon$)

il learning rate è un parametro molto importante:

alto learning rate → apprendimento veloce

basso learning rate → apprendimento lento

backward($f(x), c_i$)

optimizer()

end

end

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $\mathcal{C} > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = \mathcal{C}(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

é necessario definire a priori la struttura del modello, in particolare:

- numero di neuroni
- funzioni di attivazione
- numero di livelli
- tipologia di livelli

> calcolo la stima del modello

> calcolo la loss sul modello

> calcolo le derivate sul modello

> aggorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

la funzione di costo dipende dal task
es.

MSE per regressione,
CE per classificazione

```
init( $w_i$ )  
while ( $\mathcal{C} > \epsilon$ )  
     $y = f(x)$   
     $c_i = \mathcal{C}(\hat{y}, y)$   
    backward( $f(x), c_i$ )  
    optimizer()  
end  
end
```

- > calcolo la stima del modello
- > calcolo la loss sul modello
- > calcolo le derivate sul modello
- > aggiorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $\mathcal{C} > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = \mathcal{C}(\hat{y}, y)$ 
```

```
  backward( $f(x), c_i$ )
```

```
  optimizer()
```

```
  end
```

```
end
```

mi aspetto in output da questo algoritmo
un insieme di valori w (pesi) che
andranno a caratterizzare il modello

> calcolo la loss sul modello

> calcolo le derivate sul modello

> agguirno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

init (w_i)

Nel primo passo devo inizializzare i pesi della rete, ci sono diverse strategie, ma per ora supponiamo di inizializzarli in maniera casuale

```
while (t > ε)
  for (x ∈ S)
    ŷ = f(x)
    c_i = C(ŷ, y)
    backward(f(x), c_i)
    optimizer()
  end
end
```

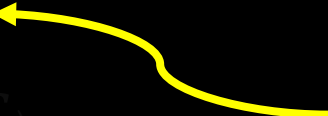
> calcolo la stima del modello
> calcolo la loss sul modello
> calcolo le derivate sul modello
> agguorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )
```



```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = C(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Di base l'algoritmo deve iterare fino a quando un determinato criterio di ottimizzazione non è raggiunto, in questo caso ho indicato il costo minore di un certo ϵ prestabilito.

Criteri comuni:

- $C > \epsilon$
 - massimo numero di epoche
 - loss non migliora in k passi $C_i > C_{i-k}$
- > calcolo la stima del modello
> calcolo la derivata sul modello
> aggiorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = C(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Per tutti i dati nel training set si itera la procedura.

I dati non sono iterati ma sono selezionati in maniera random in relazione al data loader quindi non itero x_i , con $i = 1, \dots, n$

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = C(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Per tutti i dati nel training set si itera la procedura.

I dati non sono iterati ma sono selezionati in maniera random in relazione al data loader quindi non itero x_i , con $i = 1, \dots, n$

randomizzazione del data loader:

$x_i \leftarrow \text{data_loader}(\text{rand}(i))$

Data loading

For loop:

```
for (int i=0, i<n, i++)  
    input = data[i]
```


Data loading

For loop:

```
for (int i=0, i<n, i++)  
    input = data[i]
```

Iteratore:

```
Iterator<Type> it = var_type.iterator();  
input = it.next();
```

Data loading

For loop:

```
for (int i=0, i<n, i++)  
    input = data[i]
```

Iteratore:

```
Iterator<Type> it = var_type.iterator();  
input = it.next();
```

Data loader class

```
dataLoader<type> data_loader = new myDataLoader<type>()  
input = data_loader.get_item()
```

Data loading

For loop:

```
for (int i=0, i<n, i++)  
    input = data[i]
```

Iteratore:

```
Iterator<Type> it = var_type.iterator();  
input = it.next();
```

Data loader class

```
dataLoader<type> data_loader = new myDataLoader<type>()  
input = data_loader.get_item()
```

**Il task di programmazione dove un AI-scientist
spende la maggior parte del suo tempo**

Classi di data loading

Tipicamente in quasi tutte le piattaforme pytorch, tensorflow, keras, etc. forniscono una classe astratta da ereditare per creare i data loaders

Un data loader

- Influisce fortemente sul sistema di learning
- è tipicamente il maggiore collo di bottiglia
- è dedicato


Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )  
  for ( $x \in S$ )  
     $\hat{y} = f(x)$  > calcolo la stima del modello  
     $c_i = C(\hat{y}, y)$  > calcolo la loss sul modello  
    backward( $f(x), c_i$ ) > calcolo le derivate sul modello  
    optimizer() > aggorno i pesi del modello  
  end  
end
```

forward pass – calcolo la stima \hat{y}



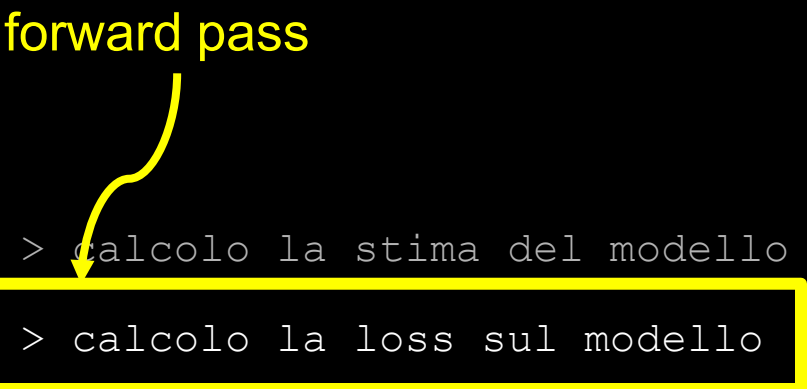
Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )  
    for ( $x \in S$ )  
         $\hat{y} = f(x)$  > calcolo la stima del modello  
         $c_i = C(\hat{y}, y)$  > calcolo la loss sul modello  
        backward( $f(x), c_i$ ) > calcolo le derivate sul modello  
        optimizer() > aggiorno i pesi del modello  
    end  
end
```

forward pass



Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )  
    for ( $x \in S$ )  
         $\hat{y} = f(x)$  > calcolo la stima del modello  
         $c_i = C(\hat{y}, y)$  > calcolo la loss sul modello  
        backward( $f(x), c_i$ ) > calcolo le derivate sul modello  
        optimizer() > aggrorno i pesi del modello  
    end  
end  
end
```

backward step

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )  
  for ( $x \in S$ )  
     $\hat{y} = f(x)$   
     $c_i = C(\hat{y}, y)$   
    backward( $f(x), c_i$ )  
    optimizer()  
  end  
end
```

ottimizzatore

- > calcolo la stima del modello
- > calcolo la loss sul modello
- > calcolo le derivate sul modello
- > aggiorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )
```

```
while ( $C > \epsilon$ )
```

```
  for ( $x \in S$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = C(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Train

> calcolo la stima del modello

> calcolo la loss sul modello

> calcolo le derivate sul modello

> aggiorno i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , learning rate l_r ,
modello non allenato f_0 , funzione di loss C

Output: parametri del modello $W = \{w_i\}$

```
init ( $w_i$ )  
while ( $C > \epsilon$ )
```

```
  for ( $x \in V$ )
```

```
     $\hat{y} = f(x)$ 
```

```
     $c_i = C(\hat{y}, y)$ 
```

```
    backward( $f(x), c_i$ )
```

```
    optimizer()
```

```
  end
```

```
end
```

Test

> calcolo la stima del modello

> calcolo la loss sul modello

> calcolo le derivate sul modello

> aggiornamento i pesi del modello

Algoritmo di apprendimento semplificato

Input: Dati di training S , Dati di validazione V , learning rate l_r
modello non allenato f_0 , funzione di loss \mathcal{C}

Output: parametri del modello $W = \{w_i\}$

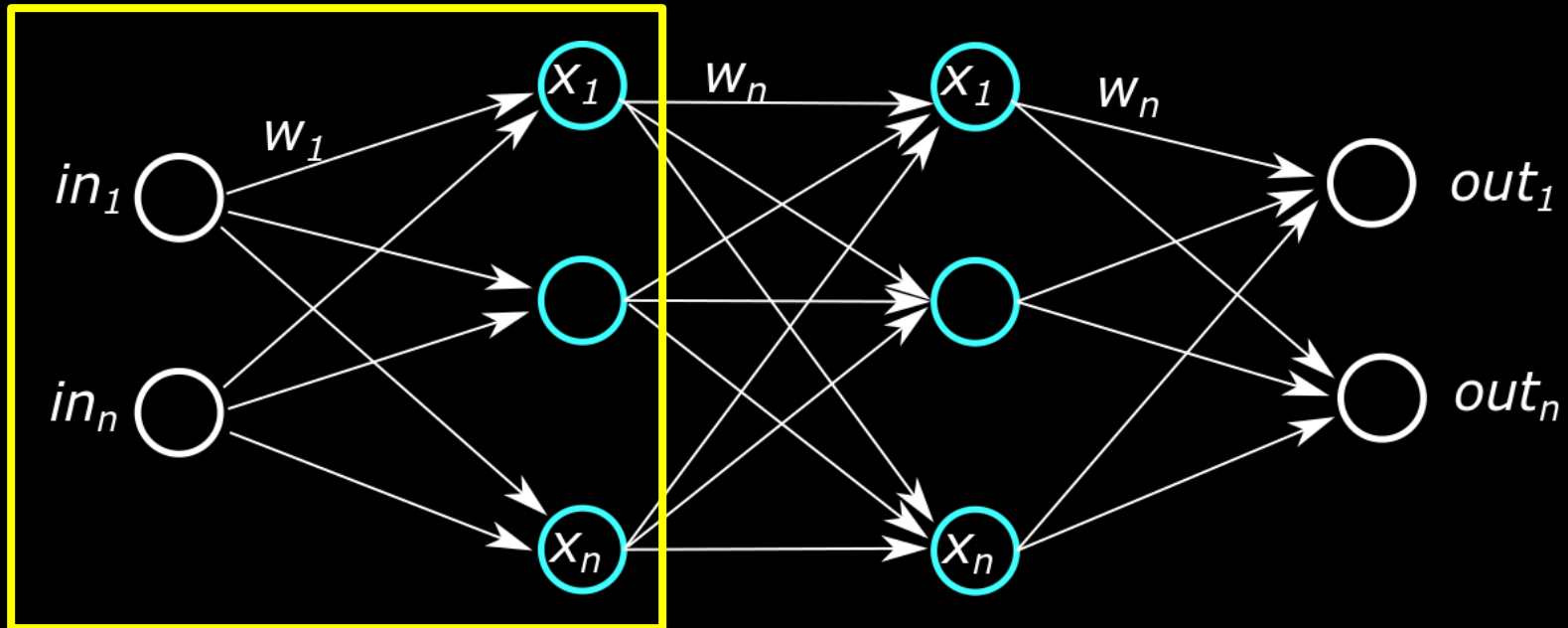
```
init ( $w_i$ )  
for each epoch  
    train ( $S$ )  
    test ( $V$ )  
end
```

Back propagation

La back propagation è la tecnica di propagazione dell'errore all'indietro nelle reti neurali che permette l'apprendimento (correzione dei pesi)

Back propagation

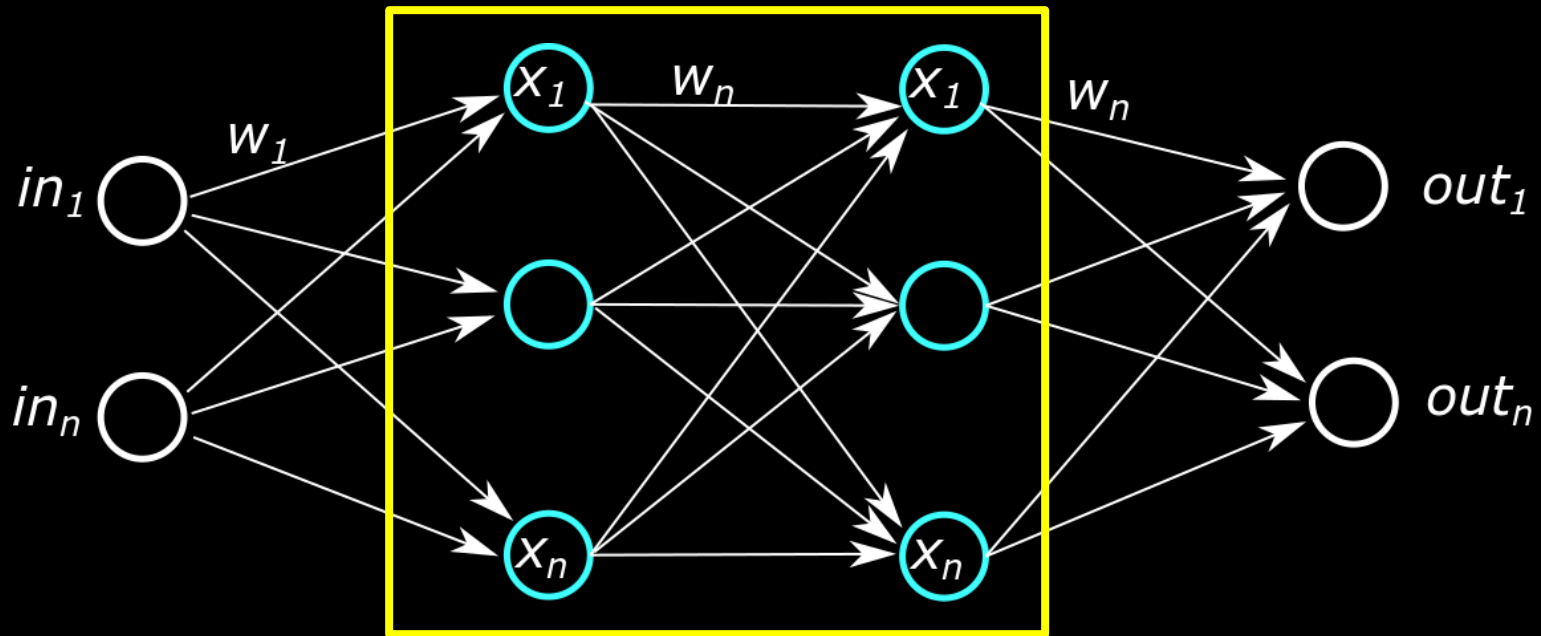
L'uscita del primo livello è una equazione matriciale lineare funzione del livello precedente



$$\varphi \left(\begin{bmatrix} in_1 \\ in_m \end{bmatrix} * \begin{bmatrix} w_{1,n} \\ w_{n,m} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_n \end{bmatrix} \right) = \begin{bmatrix} y_1 \\ y_n \end{bmatrix}$$

Back propagation

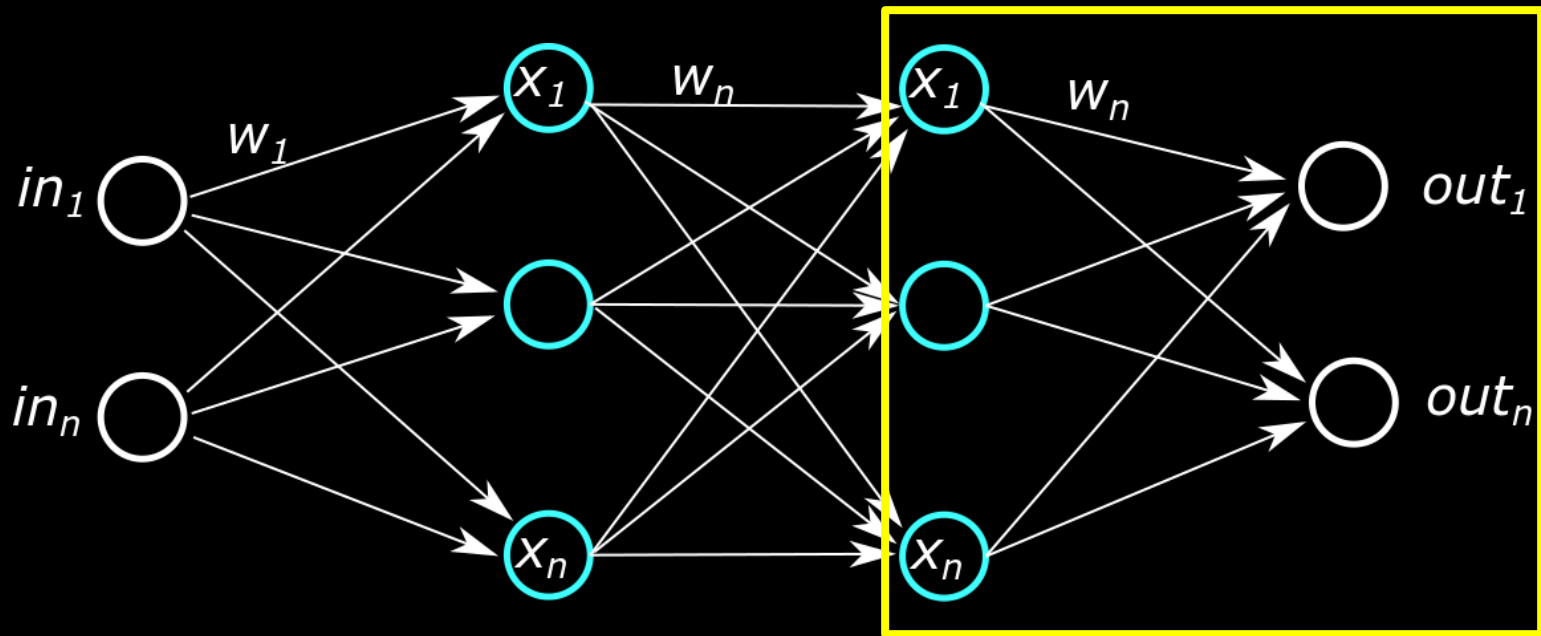
A tutti i livelli, l'uscita del livello successivo è funzione del livello precedente con una relazione matriciale lineare, i pesi w che compongono la matrice sono i pesi da imparare



$$\varphi \left(\begin{bmatrix} y_{1,k-1} \\ y_{m,k-1} \end{bmatrix} * \begin{bmatrix} w_{1,n} \\ w_{n,m} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_n \end{bmatrix} \right) = \begin{bmatrix} y_{1,k} \\ y_{n,k} \end{bmatrix}$$

Back propagation

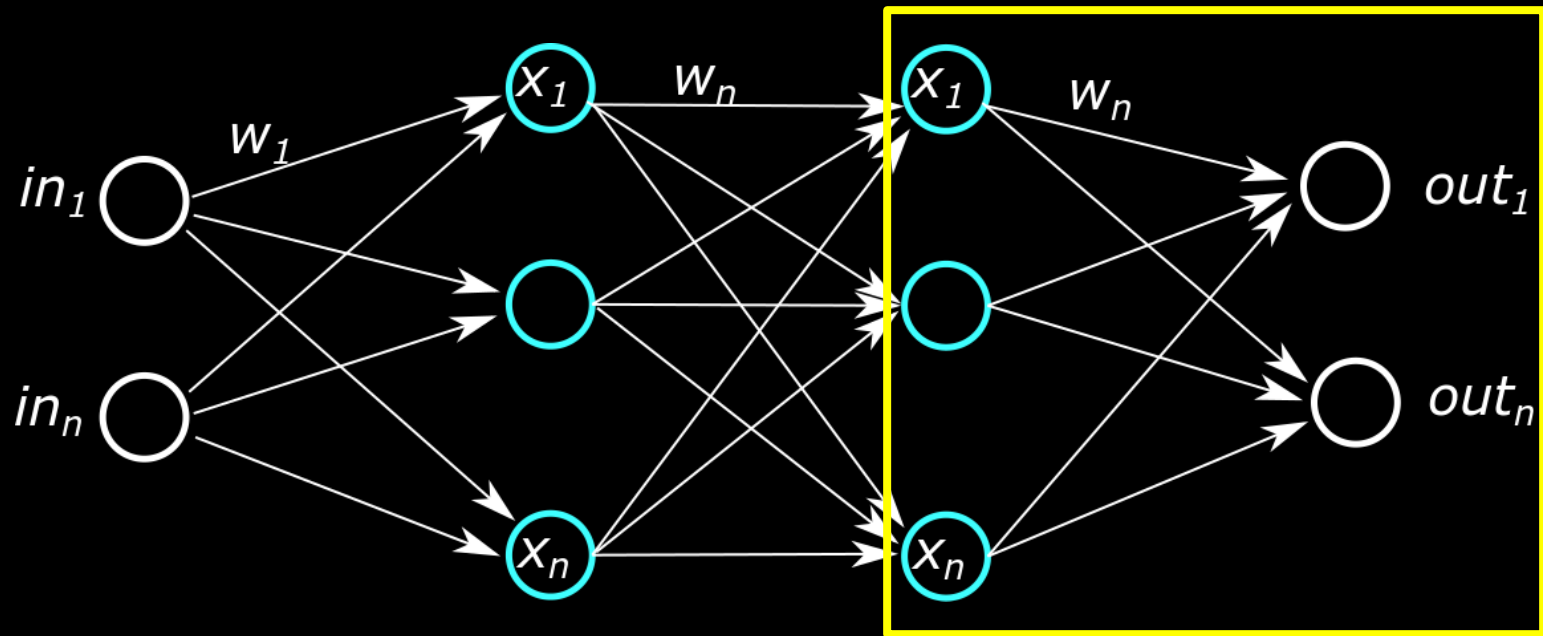
L'ultimo livello è calcolato esattamente alla stessa maniera dai precedenti, questa procedura è nota come forward pass



$$\varphi \left(\begin{bmatrix} y_{1,k-1} \\ y_{m,k-1} \end{bmatrix} * \begin{bmatrix} w_{1,n} \\ w_{n,m} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_n \end{bmatrix} \right) = \begin{bmatrix} out_1 \\ out_n \end{bmatrix}$$

Back propagation

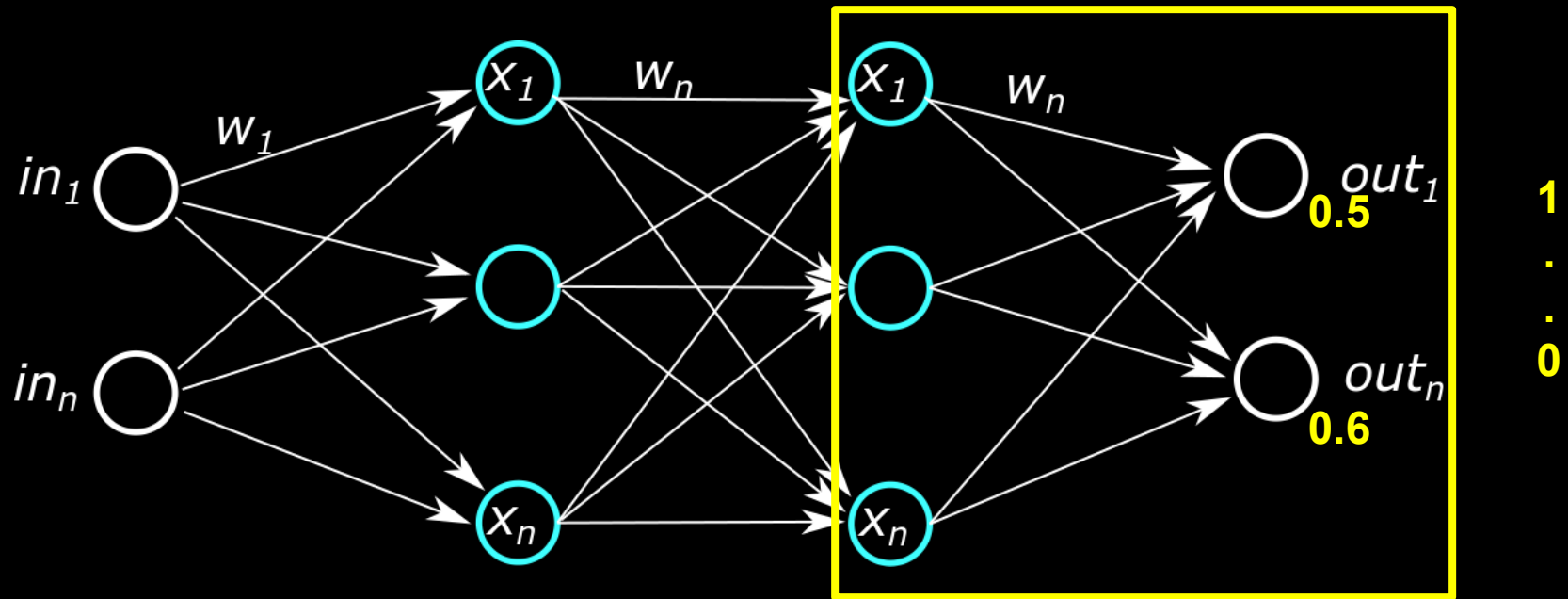
Nell'ultimo step si calcola l'errore usando la funzione di loss che abbiamo scelto, ora inizia la parte di correzione dei pesi



$$c = loss \left(\begin{bmatrix} \widehat{out_1} \\ \widehat{out_n} \end{bmatrix} - \begin{bmatrix} out_1 \\ out_n \end{bmatrix} \right)$$

Back propagation

inserisco la label $[1 \dots 0]$ e ne calcolo la loss rispetto al vettore $[out_1, \dots out_n]$

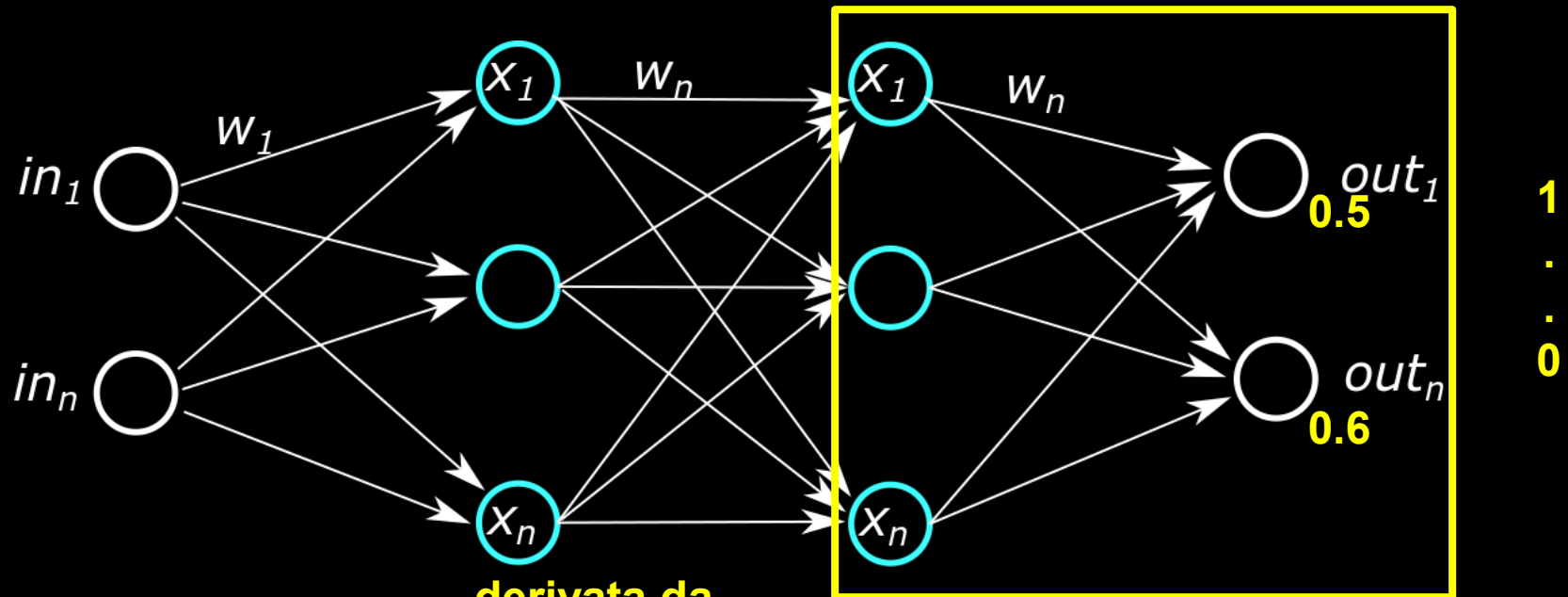


$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

$$c = loss \left(\begin{bmatrix} \widehat{0.5}_1 \\ \widehat{0.6}_n \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

Back propagation

da notare che nel caso di esempio il peso relativo al primo neurone deve salire quello relativo all'ultimo deve scendere, sono proprio queste componenti i gradienti di interesse

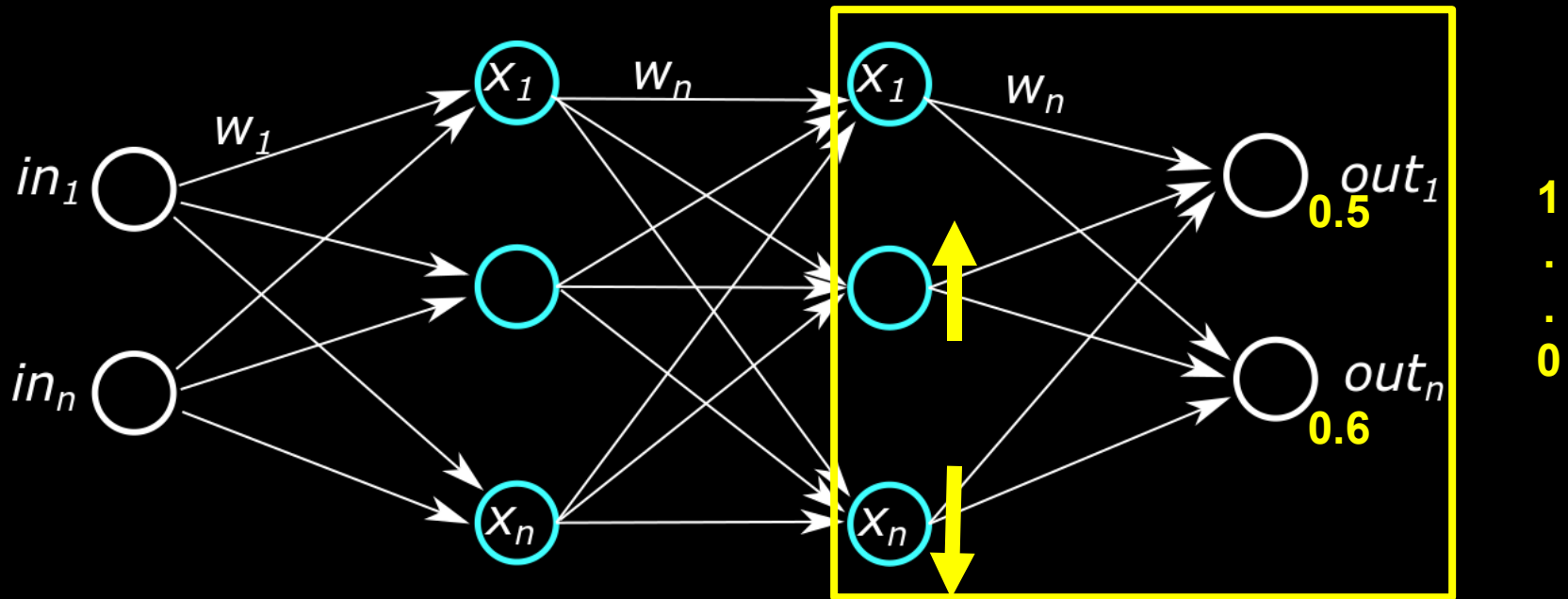


$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

$$c = loss \left(\begin{bmatrix} \widehat{0.5}_1 \\ \widehat{0.6}_n \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

Back propagation

da notare che nel caso di esempio il peso relativo al primo neurone deve salire quello relativo all'ultimo deve scendere, sono proprio queste componenti i gradienti di interesse

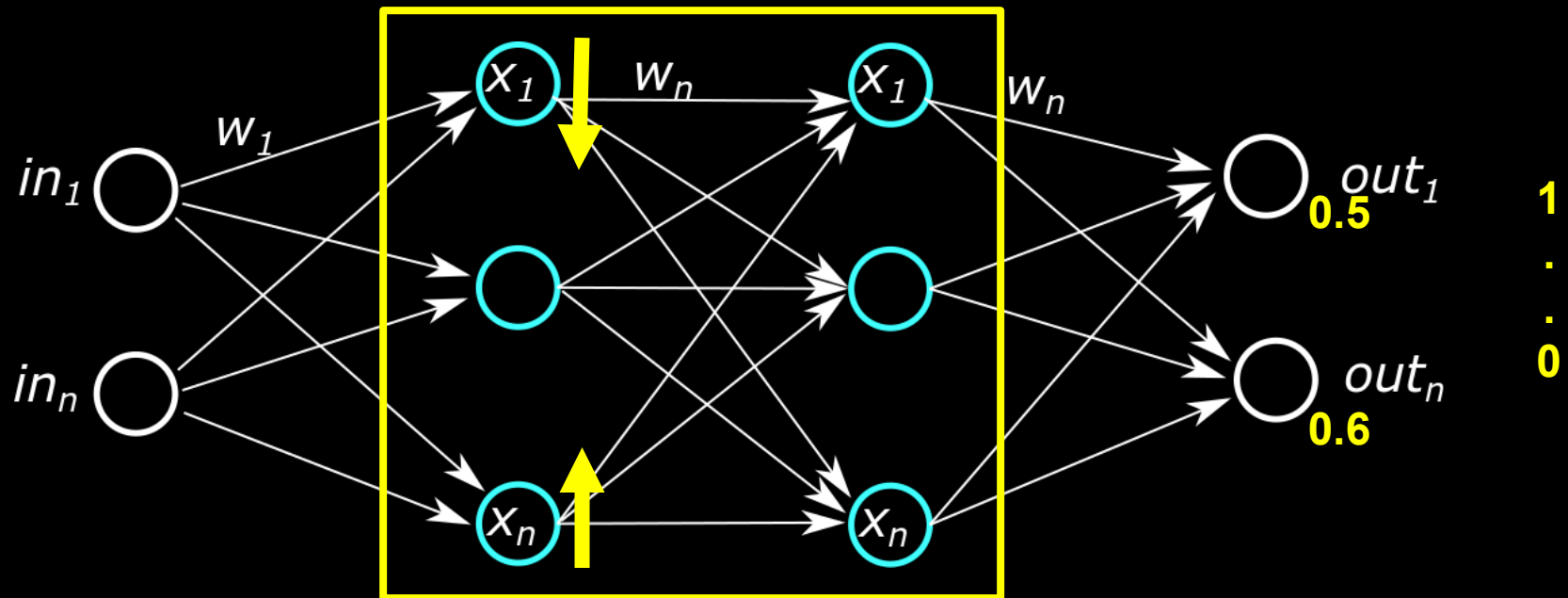


$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

$$c = loss \left(\begin{bmatrix} \widehat{0.5}_1 \\ \widehat{0.6}_n \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

Back propagation

la necessità è quella di calcolare quando un cambiamento dei pesi in un livello influenza l'uscita del livello successivo



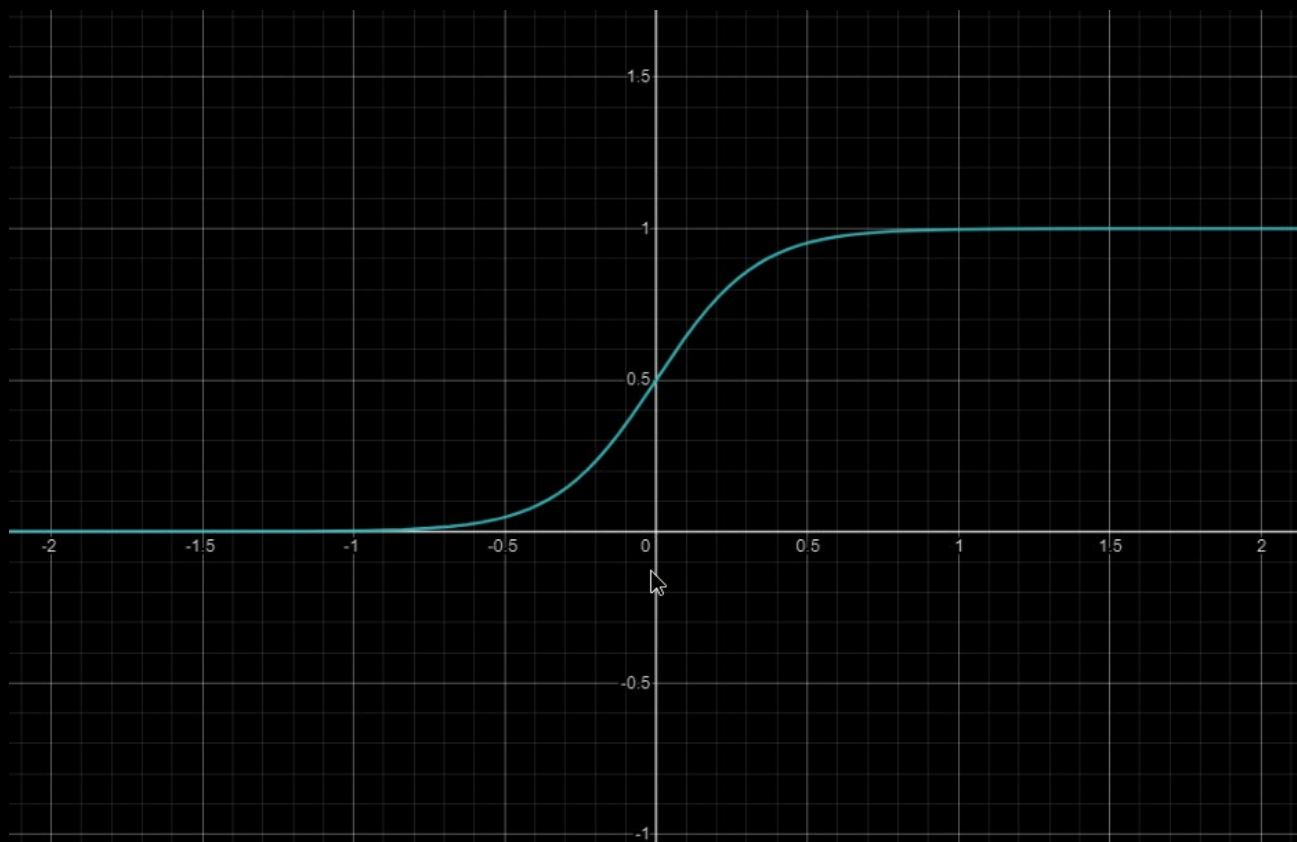
$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

$$c = loss \left(\begin{bmatrix} \widehat{0.5}_1 \\ \widehat{0.6}_n \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)$$

Problemi del gradiente

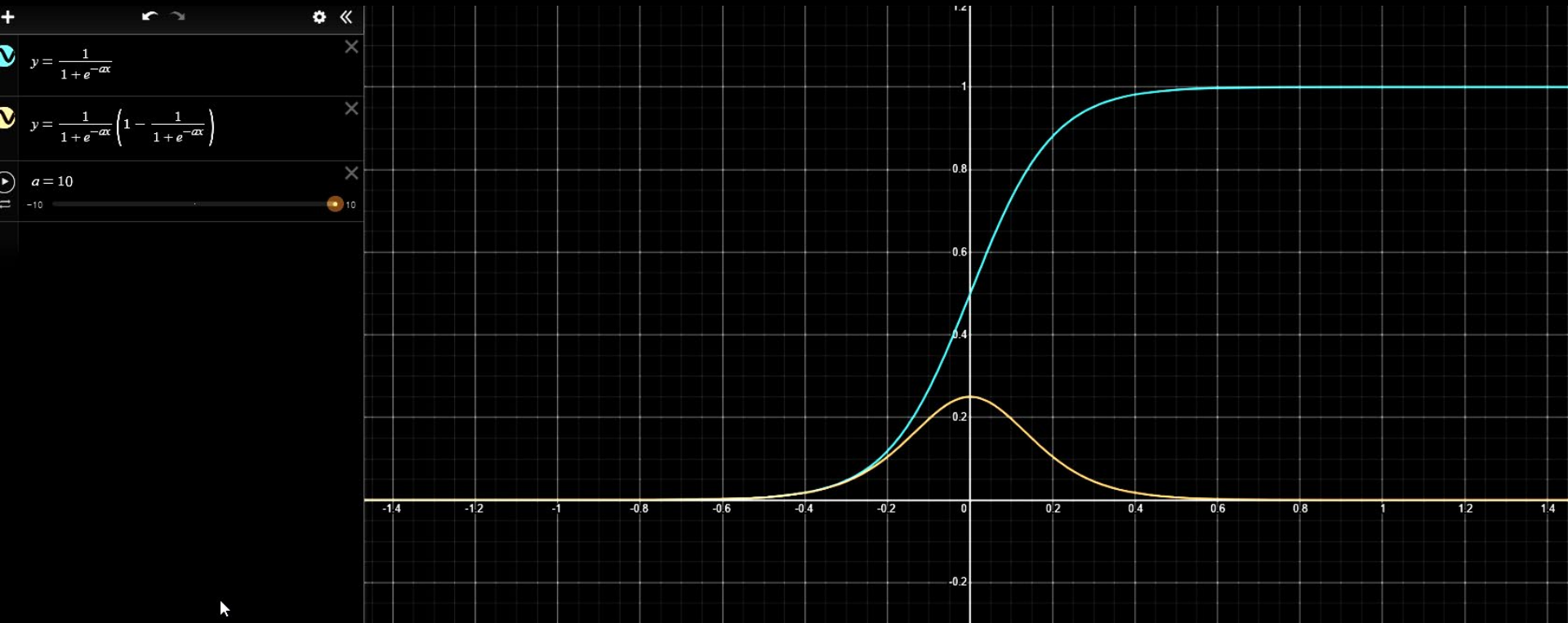
- ✓ Vanishing gradient → Valori di gradiente molto bassi che tendono ad annullare pesi e neuroni
- ✓ Exploding gradient → Valori di gradiente molto alti che tendono ad annullare l'effetto di correzione del learning rate quindi l'algoritmo di gradient descent non andrà mai a convergenza

Funzione sigmoide



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

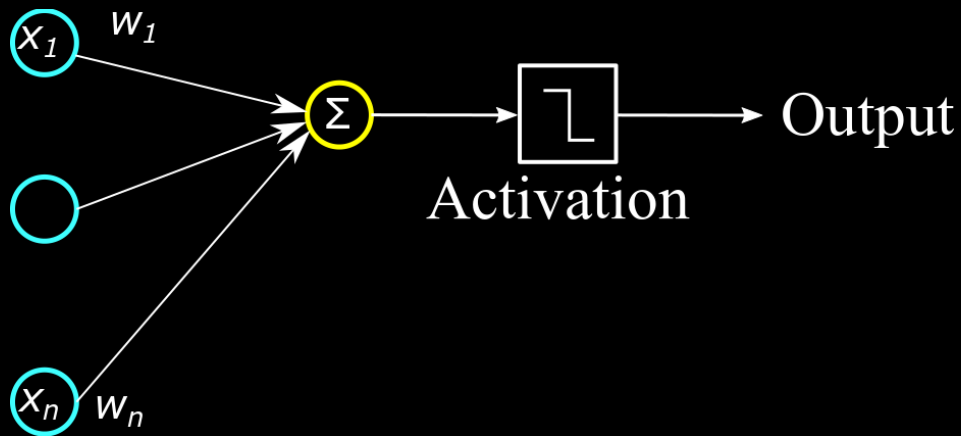
Derivata della funzione sigmoide



$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

**la derivata della sigmoide è
sempre compresa tra 0 e 0.25**

Vanishing gradient



$$y = \varphi \left(\sum \omega_i x + b_i \right)$$

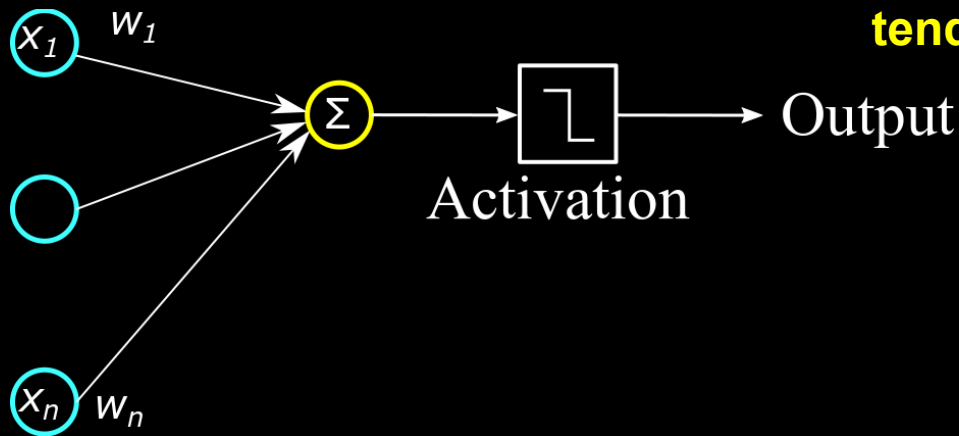
$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

**derivata da
calcolare**

$$error = y - y_{des}$$

Vanishing gradient

Se la derivate della sigmoide è sempre compresa tra 0 e 0.25 i pesi tenderanno sempre a decrementare



$$y = \varphi \left(\sum \omega_i x + b_i \right)$$

$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

derivata da
calcolare

$$error = y - y_{des}$$

Inizializzazione dei pesi

Caratteristiche dei pesi:

- essere piccoli
- essere diversi
- varianza

Inizializzazione dei pesi

Metodi comuni:

- Distribuzione uniforme

$$w_{i,j} \sim \text{uniform} \left[-\frac{1}{\sqrt{n_i}} \quad + \frac{1}{\sqrt{n_i}} \right]$$

- Xavier / Gorat

- Xavier normale

$$w_{i,j} \sim N(0, \sigma) \text{ con } \sigma = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

- Xavier uniforme

$$w_{i,j} \sim \text{uniform} \left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \quad + \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}} \right]$$

- He

- He normale

$$w_{i,j} \sim N(0, \sigma) \text{ con } \sigma = \sqrt{\frac{2}{n_{in}}}$$

- He uniforme

$$w_{i,j} \sim \text{uniform} \left[-\frac{\sqrt{6}}{\sqrt{n_{in}}}, \quad + \frac{\sqrt{6}}{\sqrt{n_{in}}} \right]$$

Normalizzazione degli input

Normalizzazione in $[0,1]$ $in_{norm} = \frac{in - min}{max - min}$

Standardizzazione $in_{stand} = \frac{in - E[in]}{\sigma[in]}$

Avere input normalizzati:

- ✓ Migliora la stabilità della rete
- ✓ Evita la disattivazione di neuroni in maniera permanente (dead neuron)
- ✓ Accelera la convergenza dell'algoritmo

Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

Come misuriamo la capacità di astrazione di un sistema di inferenza statistica?

Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

Supponiamo di avere il nostro dataset (grande a piacere)



intero dataset

Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

Per valutare un sistema di inferenza abbiamo bisogno di dati di training e test



Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

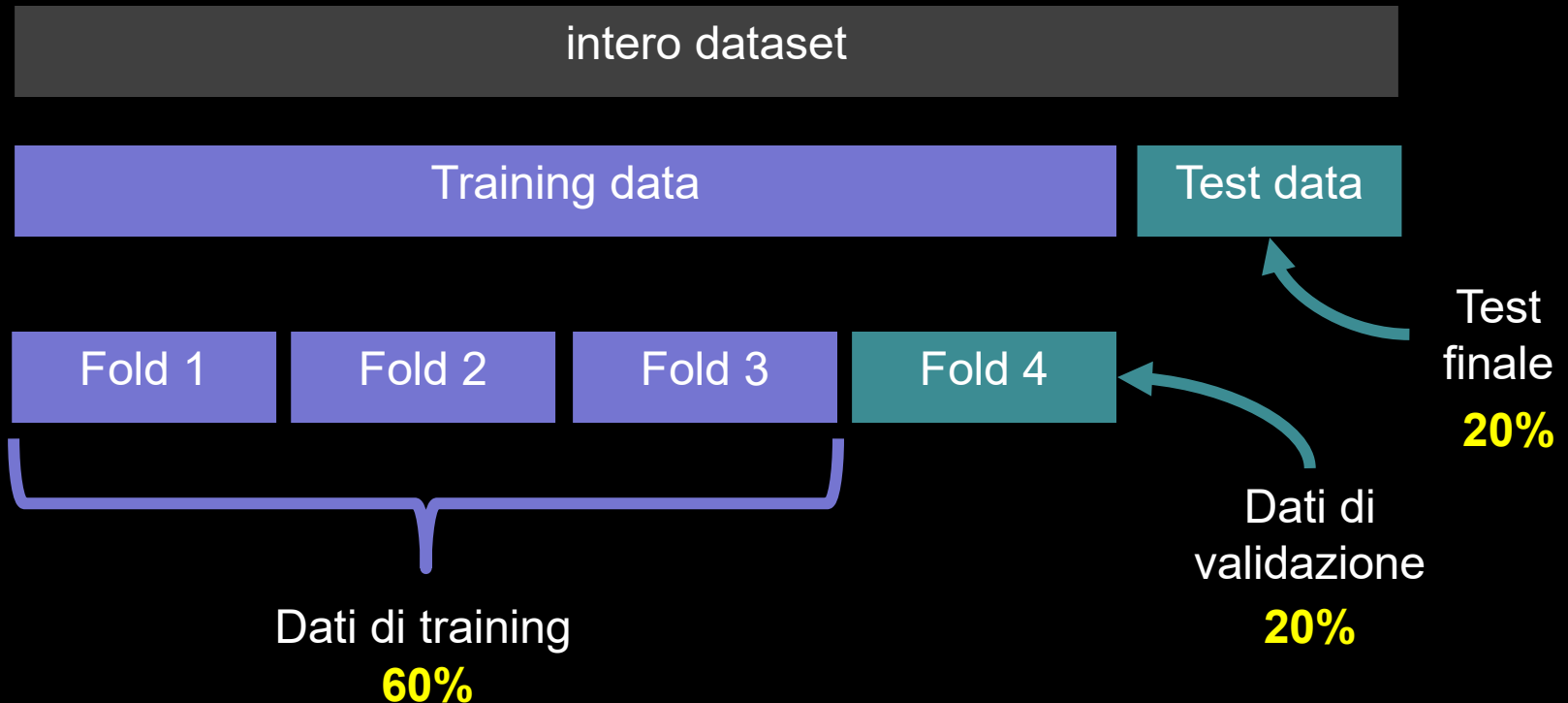
I dati di training si dividono ulteriormente in sottoinsiemi



Astrazione di conoscenza

Per astrazione si intende la capacità di un sistema di inferenza statistica di effettuare previsioni con alta accuratezza su dati mai visti

Un valore tipico è quello del 20% sulla divisione dei dati



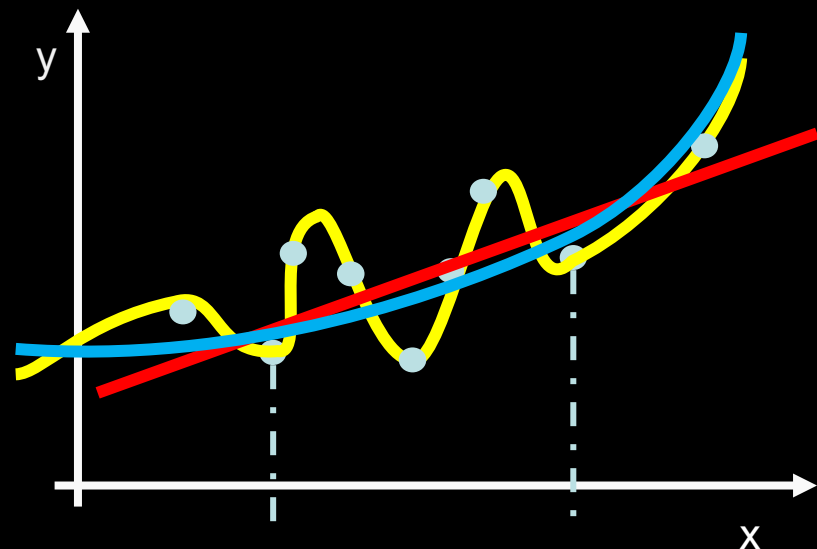
Overfitting e underfitting

In presenza di underfitting:

- Il modello funziona poco bene sui dati di training
- il modello funziona male sui dati di validazione e test

In presenza di overfitting:

- il modello funziona molto bene sui dati di training
- Il modello funziona molto male sui dati di validazione e test



k-fold cross validation

Nel training di un sistema di inferenza statistica tramite big data c'è sempre il problema di capire se le performance di classificazione o regressione sono dettate dal caso o dalla conoscenza astratta nella rete



k-fold cross validation

Nel training di un sistema di inferenza statistica tramite big data c'è sempre il problema di capire se le performance di classificazione o regressione sono dettate dal caso o dalla conoscenza astratta nella rete



k-fold cross validation

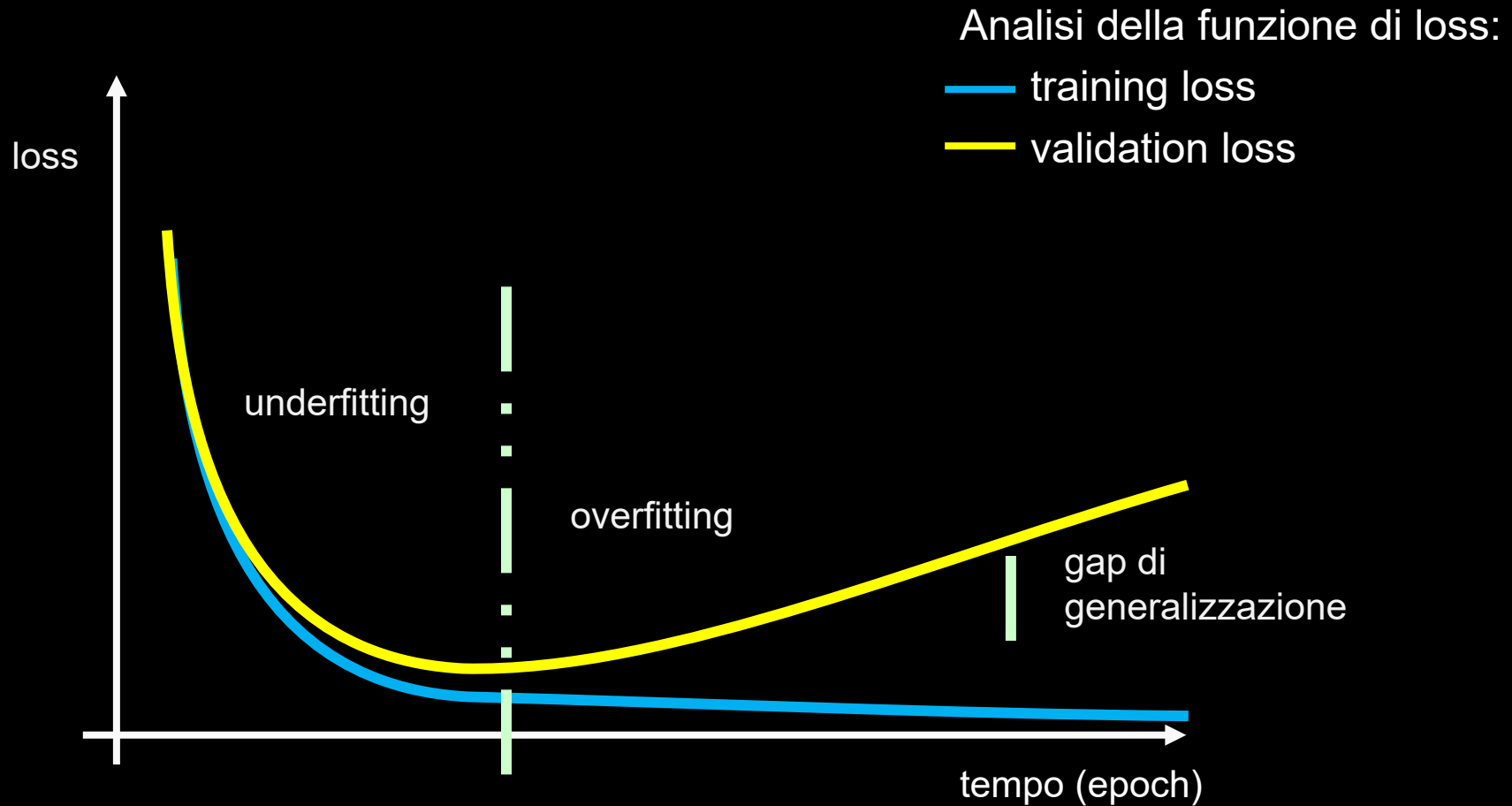
Nel training di un sistema di inferenza statistica tramite big data c'è sempre il problema di capire se le performance di classificazione o regressione sono dettate dal caso o dalla conoscenza astratta nella rete

Ciclo	TPR	TNR	FPR	FNR	F1-score
1					
2					
3					
4					
Media					
Varianza					

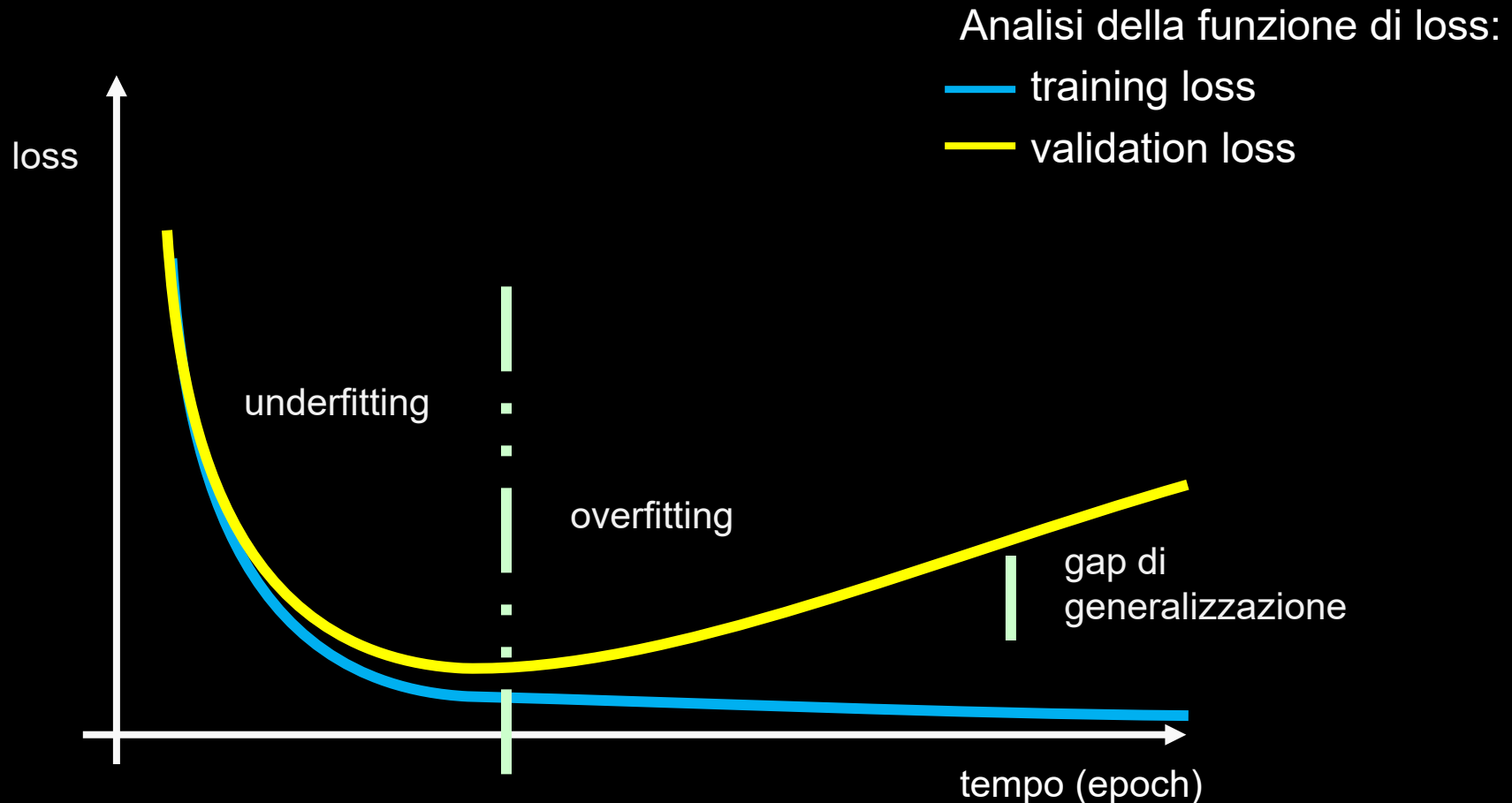
Nella tecnica di cross-validazione si compilano delle tabelle dove, per ogni ciclo di training e validazione, si inseriscono i dati di performance su test

Bassa varianza tipicamente è indice di bontà del training

Come individuare un underfitting/overfitting in una rete neurale



Come individuare un underfitting/overfitting in una rete neurale



validation loss ~ training loss → underfitting
validation loss >> training loss → overfitting

Tecniche di regolarizzazione

Tutte i sistemi basati su reti neurali sono inclini all'overfitting in quanto possiedono grande capacità di memorizzazione.

Regolarizzazione consiste in un insieme di tecniche volte ad evitare overfitting

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ **Weight decay (L2 regularization)**
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

$$error = y - y_{des}$$

$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right)$$

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ **Weight decay (L2 regularization)**
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

$$error = y - y_{des}$$

$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right) + \lambda \omega_{old}$$

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ **Weight decay (L2 regularization)**
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

$$error = y - y_{des}$$

$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right) + \lambda \omega_{old}$$

Per λ piccolo e negativo, quel contributo fa scendere i pesi della funzione ad ogni iterazione

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ **Weight decay (L2 regularization)**
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

L'effetto complessivo è quello di un minore overfitting a vantaggio di una maggiore capacità di generalizzazione

$$error = y - y_{des}$$

$$\omega_{new} = \omega_{old} - l_r \left(\frac{\delta error}{\delta \omega} \right) + \lambda \omega_{old}$$

Per λ piccolo e negativo, quel contributo fa scendere i pesi della funzione ad ogni iterazione

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ **Dataset augmentation**
- ✓ Early stopping
- ✓ Dropout



cambio in maniera random colori, trasformazioni, fuoco, specchiature, aggiungo rumore random, etc.

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ **Dataset augmentation**
- ✓ Early stopping
- ✓ Dropout

L'idea è usare gli stessi dati per aumentare la dimensione del dataset fornendo il maggior numero possibile di casi vicini alla realtà



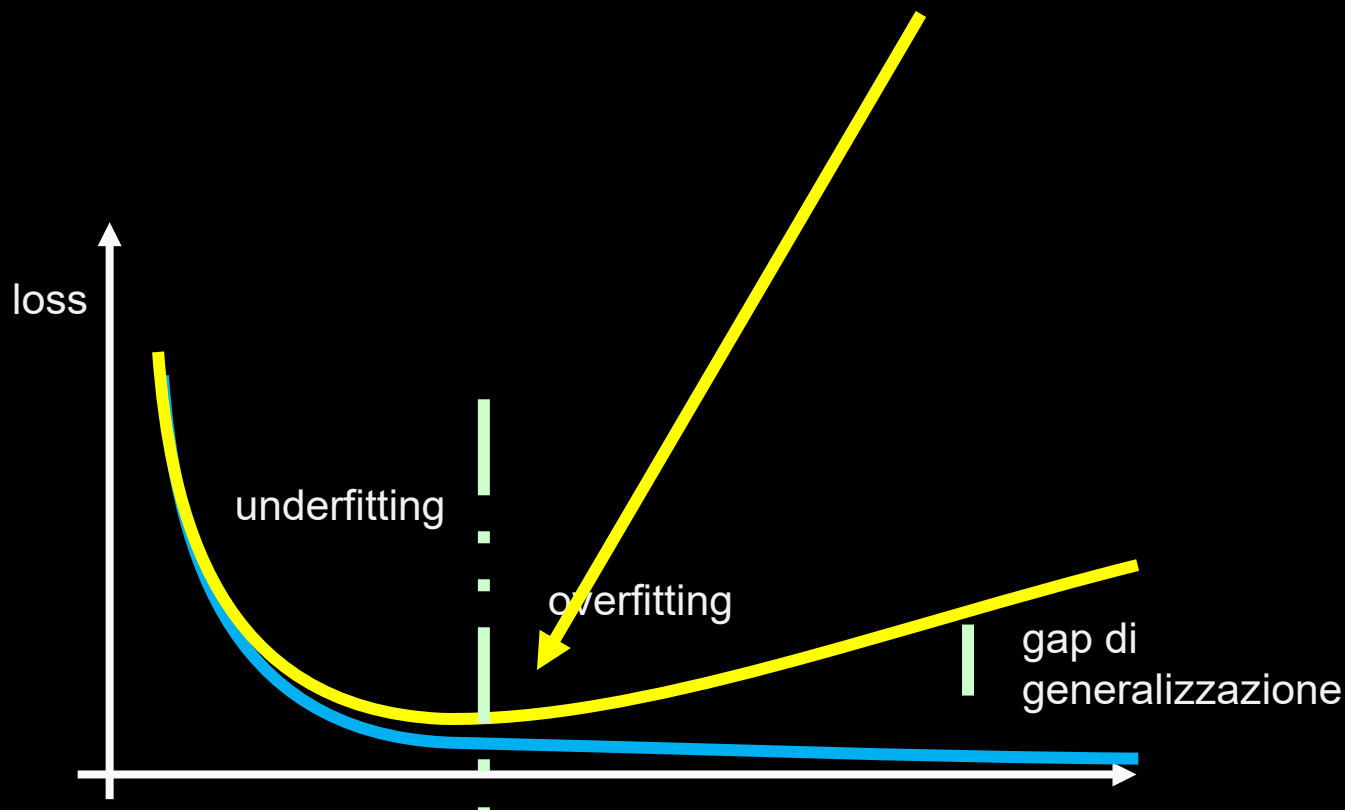
cambio in maniera random colori, trasformazioni, fuoco, specchiature, aggiungo rumore random, etc.

Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ Dataset augmentation
- ✓ **Early stopping**
- ✓ Dropout

Scelgo di fermare il training
prima che la loss di validazione
e training divergano

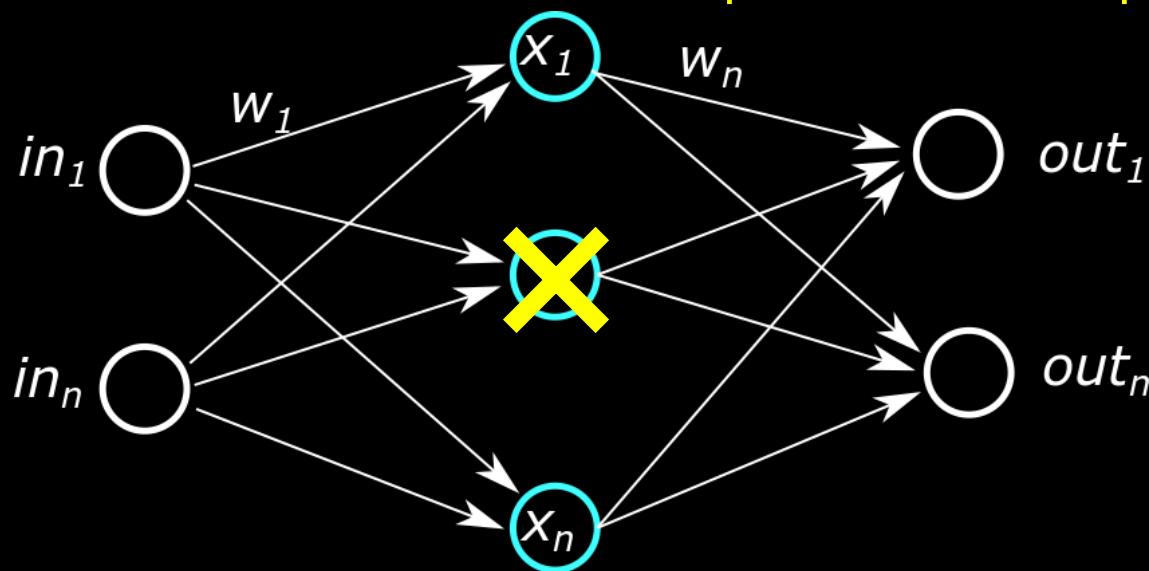


Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ **Dropout**

disattivo temporaneamente in
maniera random alcuni neuroni
inserendo un parametro di
probabilità di dropout



Tecniche di regolarizzazione

Tecniche più comuni:

- ✓ Weight decay (L2 regularization)
- ✓ Dataset augmentation
- ✓ Early stopping
- ✓ Dropout

Tutte le procedure di regolarizzazione sono volte ad evitare l'overfitting ma hanno un prezzo in termini computazionali e in termini di capacità di fitting dei dati di training

Hyperparameter tuning

Quale valore scegliere per:

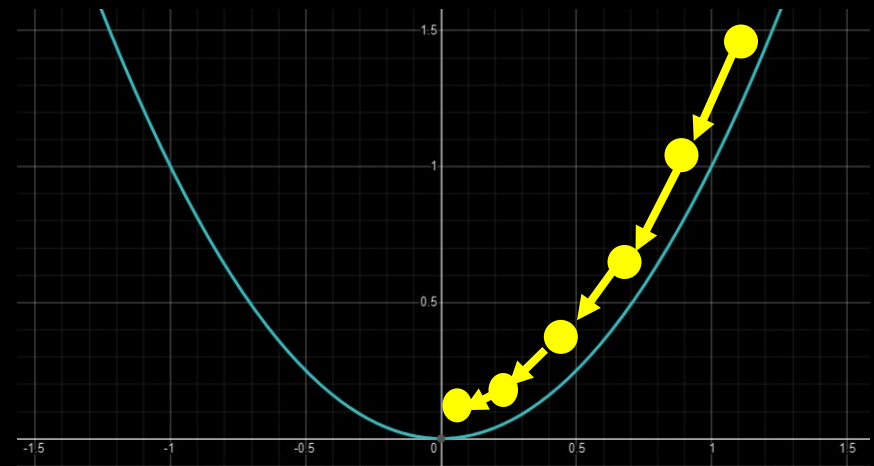
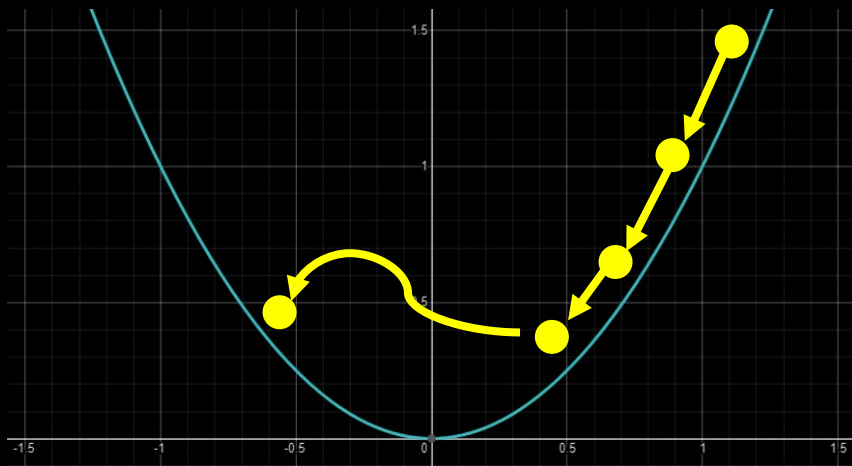
- Learning rate
- Weight decay
- Epoche
- Valori di data augmentation
- Probabilità di dropout
- .
- .
- .

La procedura iniziale di training del modello con diversi parametri scelti secondo qualche criterio va sotto il nome di **hyperparameter tuning**

è una procedura tipicamente lunga e dispendiosa in termini computazionali

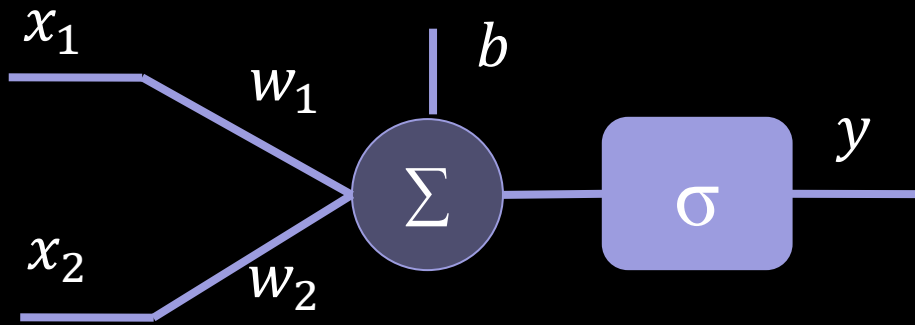
Learning rate scheduler

Al fine di evitare minimi locali è possibile usare delle policy di learning rate decay



Tutorial perceptrone

Il codice realizzato implementa un perceptrone in una classe dedicata:



PerceptroneSemplice

```
# Num_input  
# Learning_rate  
# Num_iterazioni  
# Costo_corrente  
# Param_dict  
# Grad_dict  
# Costi
```

```
+ init  
+ attivazione  
+ inizializzazione_pesi  
+ propaga  
+ ottimizza  
+ predict
```