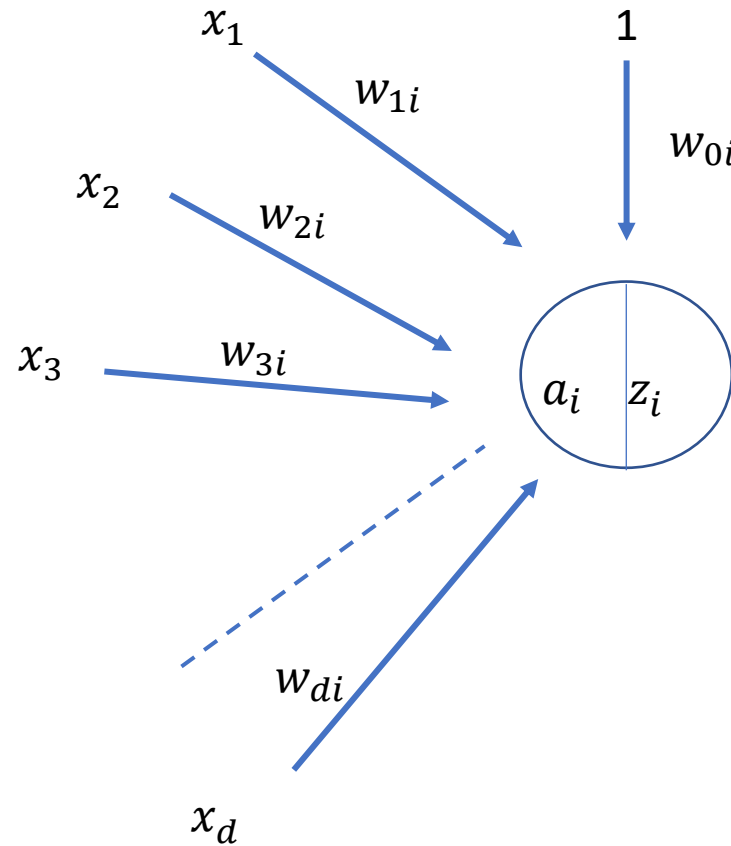


Neurone Artificiale

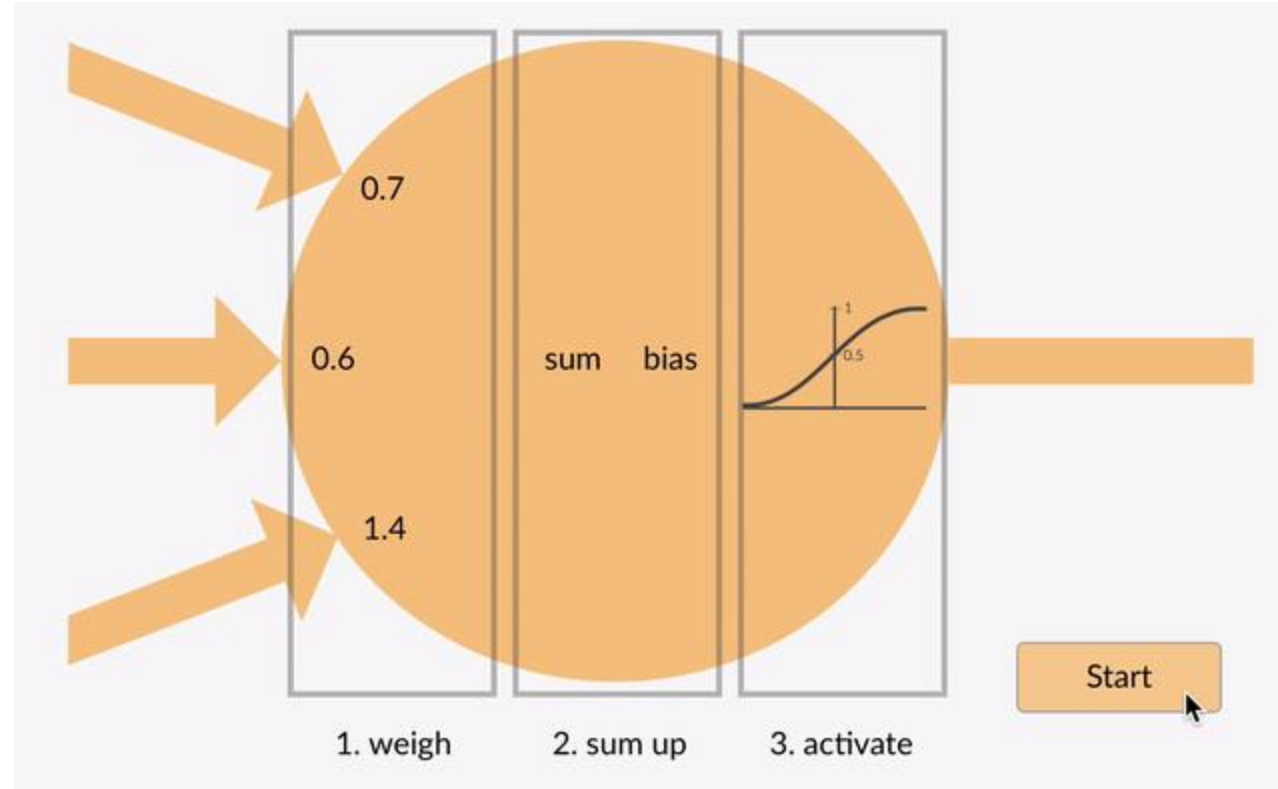


$$a_i = \sum_{j=1}^d w_{ji} x_j + w_{0i}$$

$$z_i = f(a_i) \quad \text{Output del neurone i-esimo}$$

x_1, x_2, \dots, x_d d ingressi che il neurone i riceve da assoni di neuroni afferenti

- $w_{1i}, w_{2i}, \dots, w_{di}$ sono i pesi (weight) che determinano l'efficacia delle connessioni sinaptiche dei dendriti (agiremo su questi valori durante l'apprendimento), l'importanza dell'input i -esimo sull'output.
- w_{0i} (detto bias) è un ulteriore peso che si considera collegato a un input fittizio con valore sempre 1 questo peso è utile per «tarare» il punto di lavoro ottimale del neurone.
- a_i è il livello di eccitazione globale del neurone (potenziale interno).
- $f(\cdot)$ è la funzione di attivazione che determina il comportamento del neurone (ovvero il suo output z_i in funzione del suo livello di eccitazione a_i). La funzione di attivazione simula il comportamento del neurone biologico di attivarsi solo se i segnali in ingresso superano una certa soglia.

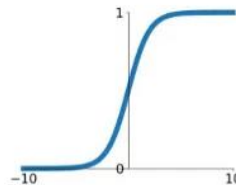


Funzioni di attivazione

Una funzione di attivazione determina se un neurone deve essere **attivato o meno**. Si tratta di alcune semplici operazioni matematiche per determinare se l'input del neurone alla rete è rilevante o meno nel processo di previsione. Le funzioni di attivazione possono essere di diversi tipi, ma in generale devono essere **non lineari** per consentire alla rete di apprendere relazioni complesse tra le sue variabili di input, e **derivabili**.

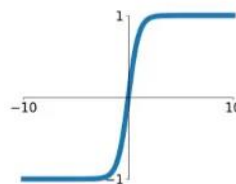
Sigmoid

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



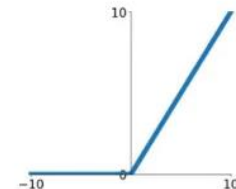
tanh

$$\tanh(x)$$



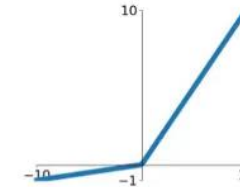
ReLU

$$\max(0, x)$$



Leaky ReLU

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

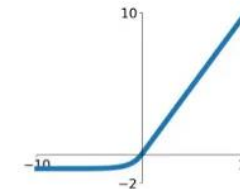


Maxout

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

ELU

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



Reti neurali artificiali (ANN)

- Similmente al cervello, una rete neurale artificiale ANN è costituita da neuroni artificiali collegati tra loro.
- Ogni **connessione** (chiamata **edge**), come le sinapsi in un cervello biologico, può trasmettere un segnale ad altri neuroni.
- Il **peso associato** a ciascuna connessione **aumenta o diminuisce la forza del segnale**.
- Tipicamente, i neuroni sono aggregati in livelli. Diversi livelli possono eseguire diverse trasformazioni sui loro input.
- I segnali viaggiano dal primo livello (il livello di input), all'ultimo livello (il livello di output), possibilmente dopo aver attraversato i livelli più volte.

Tipologie di reti neurali

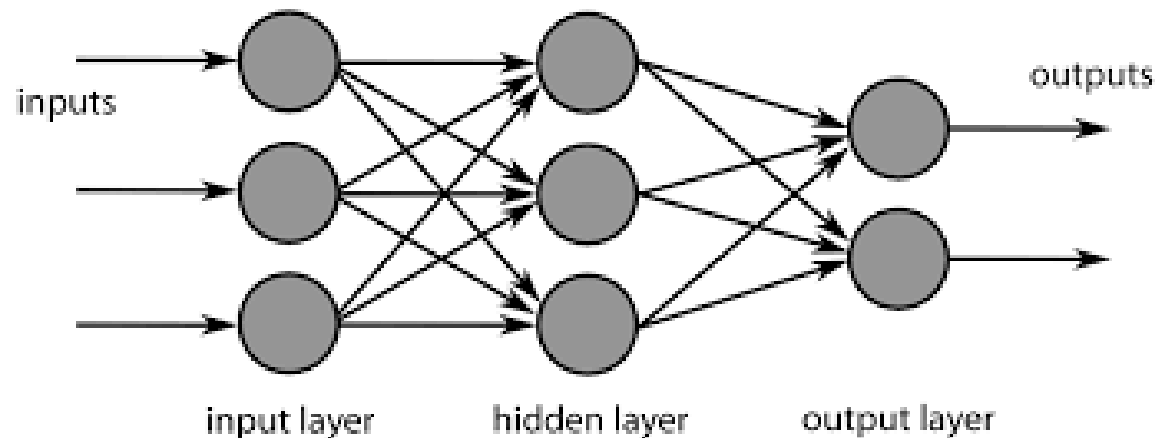
Le reti neurali sono composte da gruppi di neuroni artificiali organizzati in livelli.

Tipicamente sono presenti un livello di input, un livello di output, e uno o più livelli intermedi.

Ogni livello contiene uno o più neuroni. I **layer intermedi** sono chiamati **hidden layer** in quanto restano “invisibili” dall'esterno della rete, la quale si interfaccia all'ambiente solo tramite il layer di ingresso e quello di uscita.

Feedforward (FFNN)

nelle reti feedforward («alimentazione in avanti») le connessioni collegano i neuroni di un livello con i neuroni di un livello successivo. **Non** sono consentite connessioni all'indietro o connessioni verso lo stesso livello. È di gran lunga il tipo di rete più utilizzata



Multi Layer Perceptron (MLP)

Multi Layer Perceptron (MLP) è l'FFNN più comune costituito da tre o più layer (strati):

- un layer di input
- uno o più layer nascosti
- un layer di output

Le MLP sono fully-connected: ogni neurone in uno strato è connesso con ogni neurone nello strato successivo

Un mini-batch è un sottoinsieme del training set completo.

Training di una rete neurale

Il processo di apprendimento è una caratteristica chiave delle ANN ed è strettamente correlato al modo in cui il cervello umano apprende:

- eseguiamo un'azione
- siamo approvati o corretti da un trainer o coach per migliorare in un determinato compito.

Iterativamente

- i dati di addestramento vengono presentati alla rete (**forward**),
- quindi i pesi vengono aggiustati (**backward**) sulla base di quanto sono simili i valori restituiti dalla rete rispetto a quelli desiderati (**loss**)
- Dopo che tutti i casi sono stati presentati, il processo ricomincia da capo (un'altra epoca)

Durante la fase di apprendimento, i pesi vengono regolati per migliorare la performance sui dati di allenamento

→ **Epoca:** l'intero set di addestramento viene passato attraverso il modello, vengono eseguite la propagazione in avanti e la propagazione all'indietro e i parametri vengono aggiornati.

3 Tecniche di Aggiornamento dei Pesi:

- Stochastic Gradient Descent: i pesi della rete neurale vengono aggiornati dopo aver processato ogni singolo campione del training set.
- Batch Gradient Descent: i pesi vengono aggiornati solo dopo che tutti i campioni del training set sono stati processati.
- Mini-Batch Gradient Descent: i pesi vengono aggiornati dopo che un sottoinsieme del training set (un mini-batch) è stato processato.

Forward propagation

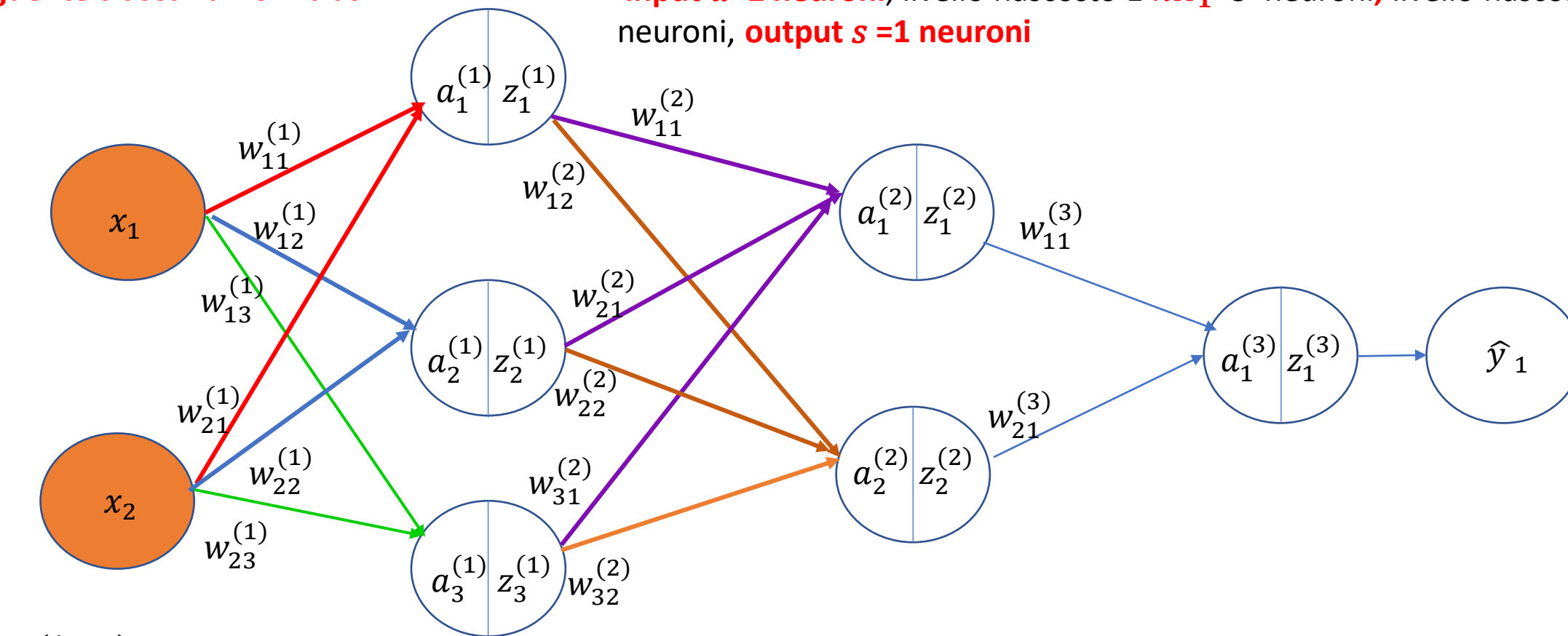
- Con forward propagation (o inference) si intende la propagazione delle informazioni in avanti dal livello di input a quello di output.
- Una volta addestrata, una rete neurale può semplicemente processare pattern attraverso forward propagation

Forward Propagation

Per semplicità, nell'esempio seguente trascuriamo il bias

nell'esempio una rete a 4 livelli $d:nH_1 : nH_2 : s$

input $d=2$ neuroni, livello nascosto 1 $nH_1=3$ neuroni, livello nascosto 2, $nH_2=2$ neuroni, output $s=1$ neuroni



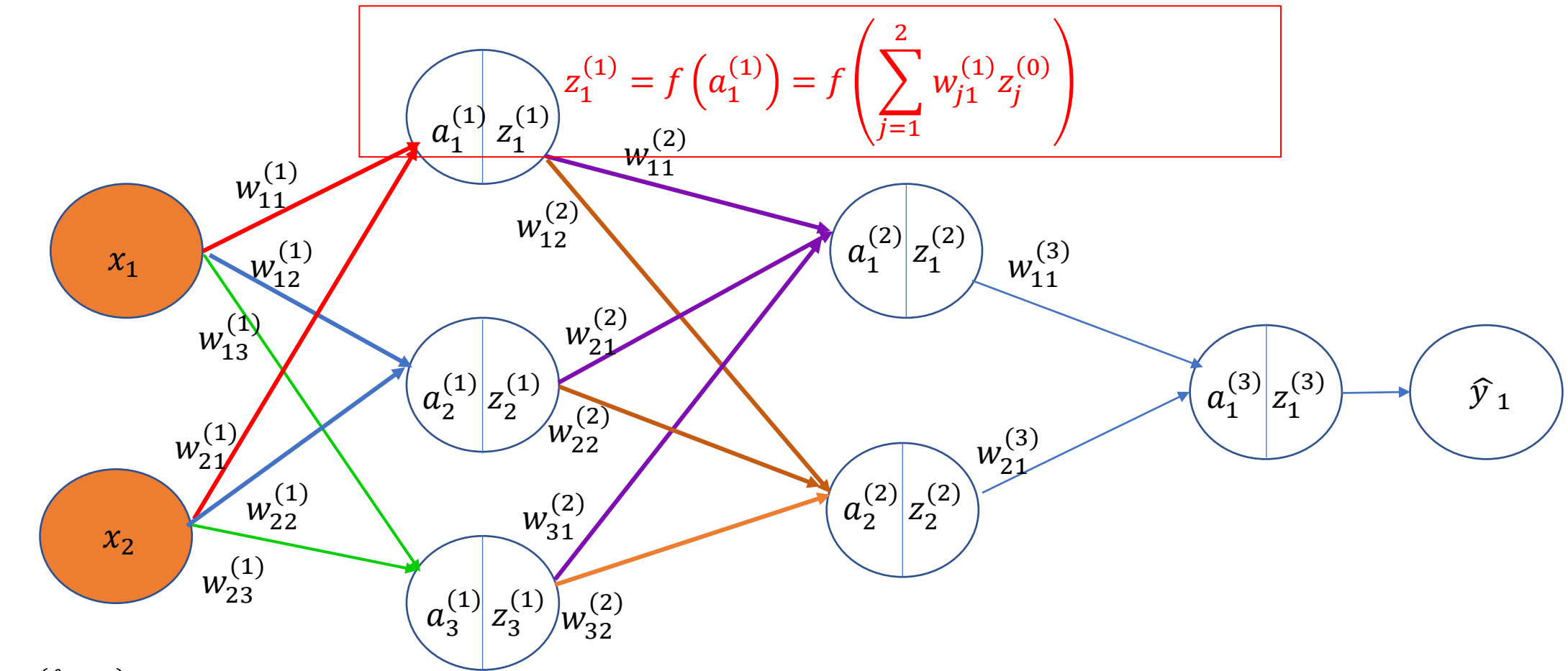
$w_{input,output}^{(layer)}$

$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.



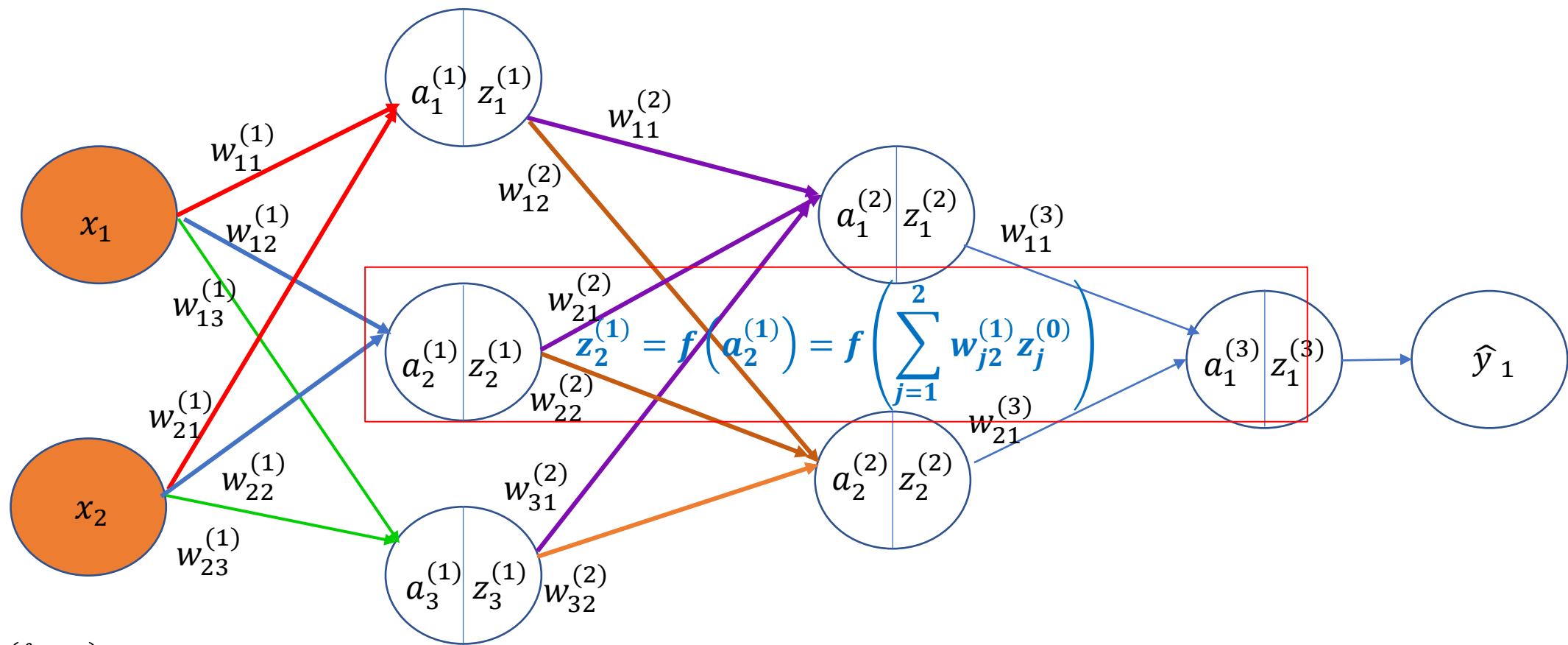
$w_{input,output}^{(layer)}$

$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.



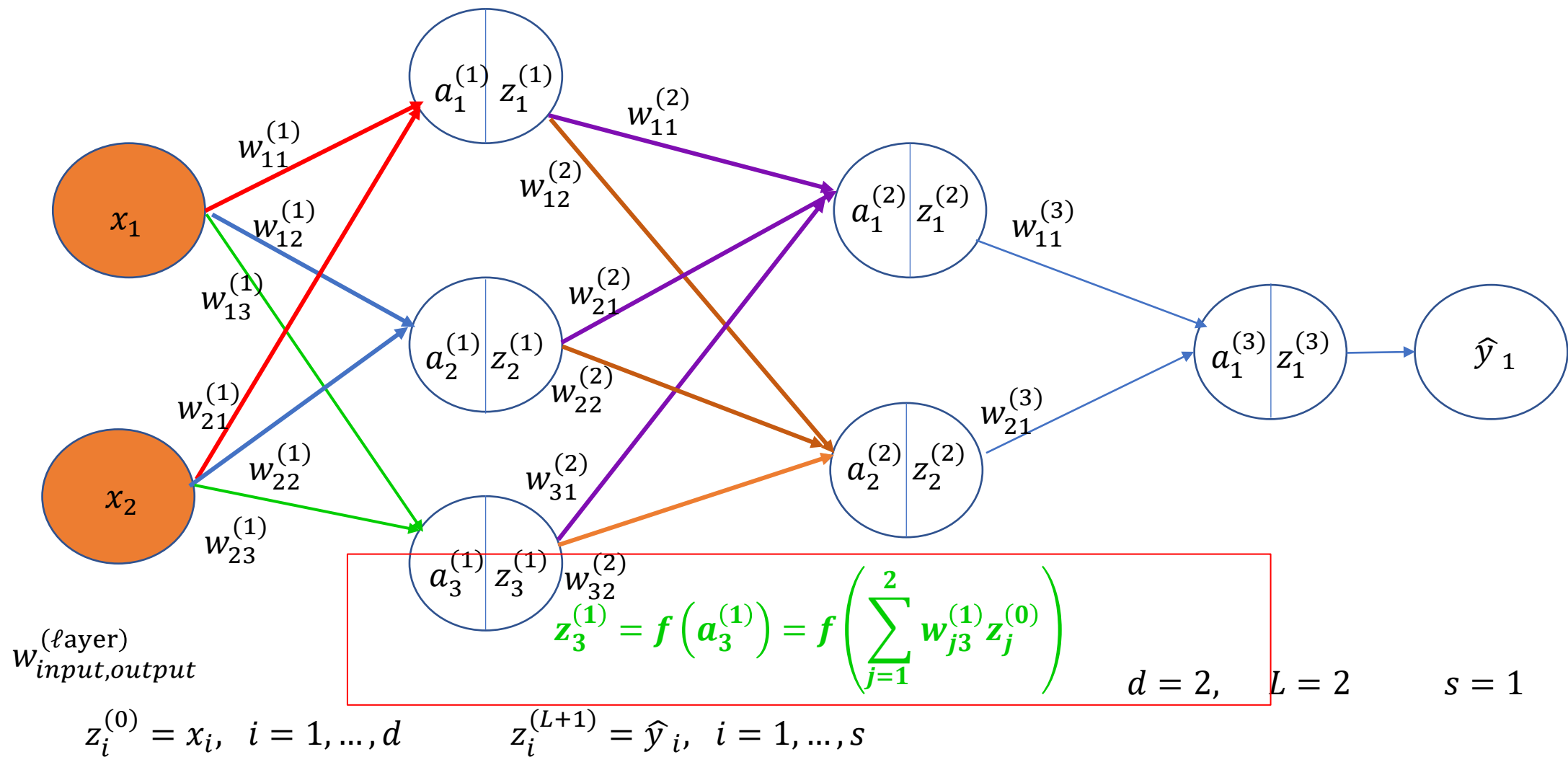
$w_{input,output}^{(layer)}$

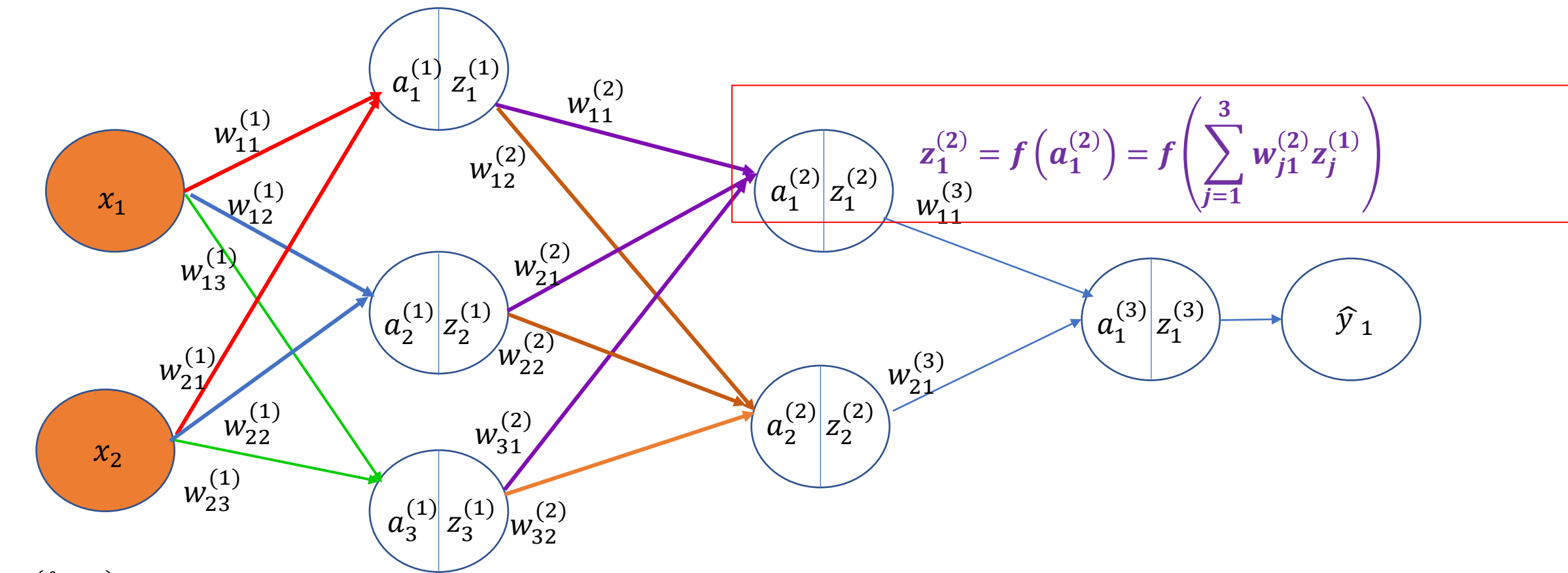
$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.





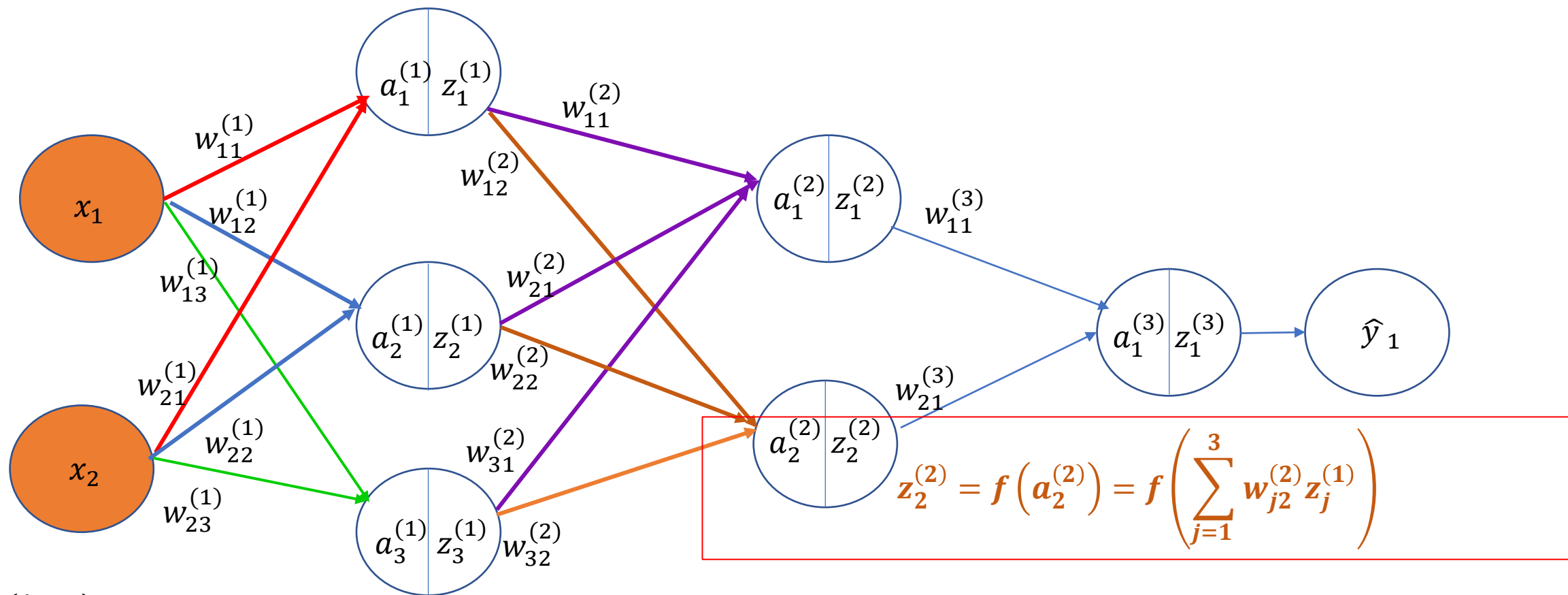
$w_{input,output}^{(layer)}$

$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.



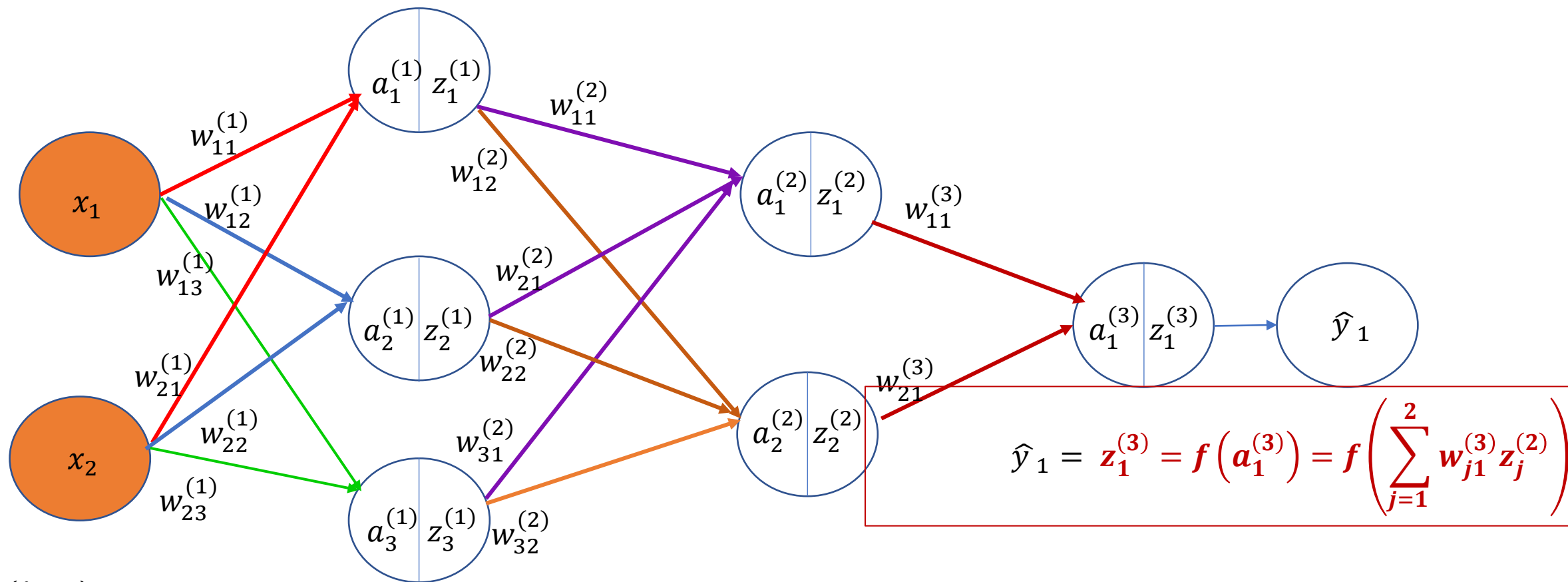
$w_{input,output}^{(layer)}$

$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.



$w_{input,output}^{(layer)}$

$$d = 2, \quad L = 2 \quad s = 1$$

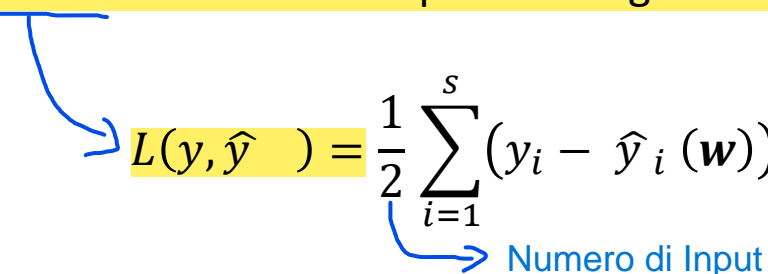
$$z_i^{(0)} = x_i, \quad i = 1, \dots, d \quad z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=1}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)})$$

f : activation function: funzione **derivabile** non lineare.

Sia $\hat{y} = [\hat{y}_1, \dots, \hat{y}_s]$ l'output prodotto dalla rete in corrispondenza all'esempio di training $x = [x_1, x_2, \dots, x_d]$, e sia $y = [y_1, y_2, \dots, y_s]$ il vettore delle etichette corrispondente dell'esempio di training.

Scegliamo come loss function la somma dei quadrati degli errori tra l'output della rete e la corrispondente etichetta:


$$L(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^s (y_i - \hat{y}_i(\mathbf{w}))^2 = \frac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}(\mathbf{w})\|_2^2$$

Numero di Input

dove $\mathbf{w} = \{w^{(1)}, w^{(2)}, \dots, w^{(l)}\}$, dove $w^{(j)}$ rappresenta l'insieme dei pesi al layer j -esimo.

Il problema dell'addestramento

La costruzione di una rete multistrato con d ingressi e s uscite consiste nello **scegliere l'architettura** (numero di strati (layer) e numero di neuroni di ogni strato), e **nell'addestrare la rete**, ossia nel determinare il vettore $\mathbf{w} = \{w^{(1)}, w^{(2)}, \dots, w^{(l)}\}$ le cui componenti corrispondono ai parametri incogniti (pesi e soglie dei neuroni nei vari strati).

La scelta dei parametri, in addestramento di tipo **supervisionato**, per una architettura fissata, viene in genere effettuata definendo un opportuno sottoinsieme dei dati disponibili etichettati

$$T = \{(x^{(j)}, y^{(j)}), x^{(j)} \in \mathbb{R}^d, y^{(j)} \in \mathbb{R}^s, j = 1, \dots, n_T\},$$

che costituisce il **training set** e risolvendo successivamente un problema di ottimizzazione del tipo:

$$\mathbf{w}^* \in \arg \min_{\mathbf{w}} \left\{ \mathcal{C}(\mathbf{w}) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)}(\mathbf{w})) \right\}$$

dove $\hat{y}^{(j)}(\mathbf{w}) = [\hat{y}_1^{(j)}, \dots, \hat{y}_s^{(j)}]$ è l'output prodotto dalla rete in corrispondenza al j -esimo esempio di training $x^{(j)} = [x_1^{(j)}, x_2^{(j)}, \dots, x_d^{(j)}]$, e sia $y = [y_1^{(j)}, y_2^{(j)}, \dots, y_s^{(j)}]$ il vettore delle etichette corrispondente dell'esempio di training.

$L(y^{(j)}, \hat{y}^{(j)}(\mathbf{w}))$ è il termine di errore relativo al j -esimo campione di training e misura la distanza tra l'uscita desiderata $y^{(j)}$ e l'uscita $\hat{y}^{(j)}$ fornita dalla rete. La misura dell'errore più usata è l'errore quadratico ma è possibile considerare anche funzioni di errore di struttura diversa. Nel seguito ci limiteremo a supporre che $\mathcal{C}(\mathbf{w})$ sia una funzione continuamente differenziabile.

Scopo dell'addestramento non è quello di interpolare i dati di training, quanto piuttosto quello di modellare il processo che ha generato i dati.

Ciò implica che:

- la scelta dell'architettura,
- la selezione dei dati da includere in T ,
- la definizione di L
- e la strategia di addestramento

devono tener conto dell'esigenza di assicurare buone capacità di generalizzazione.

Dal punto di vista teorico, uno dei problemi più importanti è quello di definire opportunamente la complessità del modello, e quindi il numero di parametri liberi, in relazione ai dati disponibili. Per le reti multistrato, a partire dai risultati sulla teoria statistica dell'apprendimento sono state stabilite delle stime del numero minimo dei campioni di training occorrenti per addestrare una rete in modo tale che si abbia una "buona" capacità di generalizzazione.

In pratica, tuttavia le stime teoriche possono essere inadeguate ed **occorre basarsi su opportune euristiche per la scelta della struttura e la definizione del training set**. In linea di massima, nel caso delle reti multistrato, vengono seguite due strategie fondamentali.

La **prima**, chiamata **stabilizzazione strutturale** consiste nello scegliere il numero di unità (neuroni), attraverso **l'addestramento di una sequenza di reti in cui viene fatto crescere (o diminuire) il numero di neuroni**.

- Per ciascuna di tali reti i parametri (**pesi e soglie dei neuroni nei vari strati**) vengono determinati minimizzando l'errore sul training set e le prestazioni delle varie reti sono confrontate attraverso una tecnica di *cross-validation*, valutando l'errore che ogni rete commette su un altro insieme di dati (*validation set*) non inclusi in T . **La rete selezionata è quella che fornisce l'errore minimo sul validation set.**

Le prestazioni di una rete addestrata vengono valutate utilizzando un terzo insieme di dati denominato *test set*, che non deve essere stato utilizzato né per la scelta dell'architettura, né per la determinazione dei parametri.

Controllare e ridurre l'ampiezza (o il valore assoluto) dei pesi durante l'addestramento. Questo processo è fondamentale per prevenire il sovra-addestramento (overfitting), migliorare la generalizzazione e stabilizzare l'addestramento della rete.

La **seconda strategia** si basa su una **tecnica di regolarizzazione** e consiste nell'aggiungere alla funzione di errore un termine di penalità sulla norma dei pesi w che ha l'effetto di restringere l'insieme entro cui vengono scelti i parametri.

Ciò equivale, essenzialmente, ad imporre condizioni di regolarità sulla classe di funzioni realizzata dalla rete.

L'addestramento della rete viene effettuato definendo la nuova funzione obiettivo

$$w^* \in \arg \min_w \left\{ C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)}(w)) + \gamma \|w\|_2^2 \right\}$$

con $\gamma > 0$.

iper-parametro

Il valore "ottimale" di γ può essere determinato utilizzando, anche in questo caso, una tecnica di cross-validation. In particolare, in corrispondenza a differenti valori di γ , si addestrano varie reti minimizzando la funzione d'errore C e viene successivamente prescelto il valore di γ a cui corrisponde il minimo errore sul validation set.

In alternativa alla tecnica di regolarizzazione, una strategia euristica talvolta utilizzata è quella **di interrompere prematuramente la minimizzazione (early stopping) della funzione d'errore.**

Questa tecnica si basa sull'idea di **valutare periodicamente**, durante il processo di minimizzazione, **l'errore che la rete commette su un validation set ausiliario.**

In generale, nelle prime iterazioni l'errore sul validation set diminuisce con la funzione obiettivo, mentre può aumentare se l'errore sul training set diviene "sufficientemente piccolo".

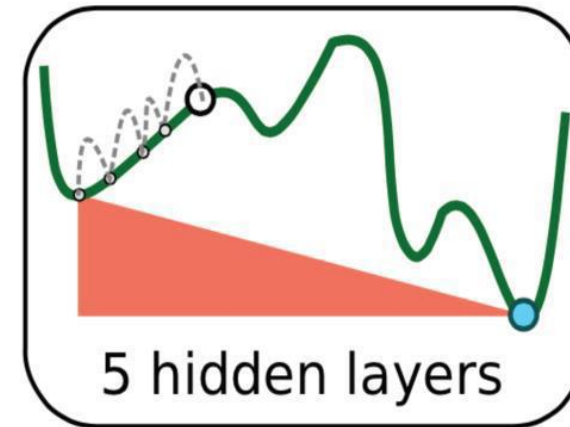
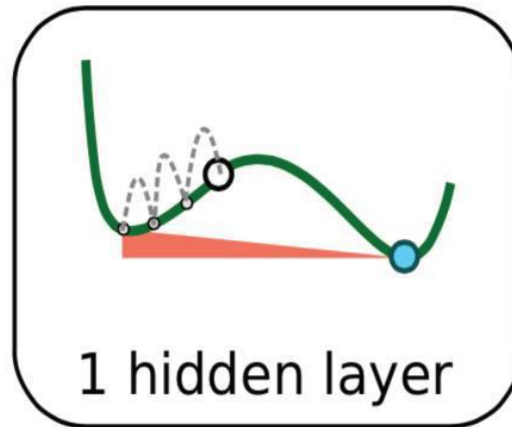
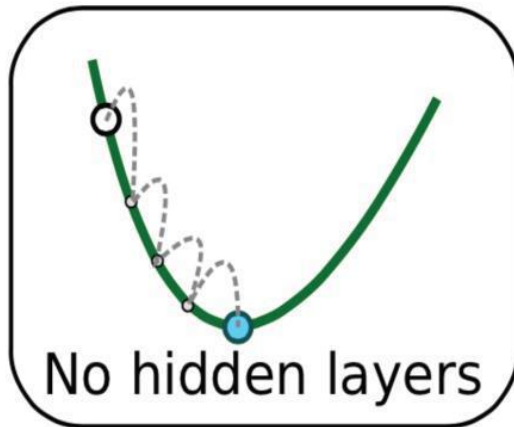
→ Il processo di addestramento termina quindi quando l'errore sul validation set inizia ad aumentare, perchè ciò potrebbe evidenziare l'inizio della fase di overfitting della rete, cioè della fase in cui la rete tende a interpolare i dati di training a scapito della capacità di generalizzazione.

Quale che sia la strategia di addestramento seguita, pur non essendo possibile ridurre banalmente i problemi di addestramento alla soluzione di un problema di ottimizzazione, **è necessario**, in pratica, **ripetere la minimizzazione in corrispondenza a varie architetture o a varie funzioni di errore**. La disponibilità di **algoritmi efficienti di ottimizzazione costituisce, quindi, uno strumento essenziale per la costruzione delle reti neurali**. La minimizzazione dell'errore di training C è, in generale, un difficile problema di ottimizzazione non lineare, in cui le difficoltà computazionali sono tipicamente dovute a

- **forti non linearità di $C(w)$, che creano “valli ripide” e zone “piatte” nella superficie della funzione di errore;**
- **elevata dimensionalità di w ed elevato numero n_T di campioni;**
- **presenza di minimi locali non globali.**

Quasi tutti i problemi di ottimizzazione che sorgono in Deep Learning sono non convessi

- introducendo la non linearità nella rete (aggiungendo strati nascosti), la funzione costo diventa non convessa e compaiono i minimi locali



Numero di pesi in una rete MLP

Calcoliamo il numero di pesi in una rete MLP (Multi-Layer Perceptron) con i seguenti parametri:
(in questo calcolo non viene considerato il termine di bias e si suppone che i layer nascosti siano formati dallo stesso numero di nodi:)

d : Numero di nodi nel layer di input

s : Numero di nodi nel layer di output

l : Numero di layer nascosti

p : Numero di nodi per ogni layer nascosto

Connessioni tra il layer di input ed il primo layer nascosto

- Ogni nodo nel primo layer nascosto (ed ogni layer nascosto è formato da p nodi) è connesso a tutti i nodi del layer di input. Numero di connessioni $p \cdot d$. I pesi associati a queste connessioni rappresentano i pesi tra i nodi del layer di input e i nodi del primo layer nascosto.

Connessioni tra layer nascosti:

Ogni nodo in un layer nascosto è connesso a tutti i nodi nel layer precedente.

Quindi, per ogni layer nascosto (tranne il primo e l'ultimo), il numero di connessioni è: $p \cdot p$

I pesi associati a queste connessioni rappresentano i pesi tra i nodi di un layer nascosto e i nodi del layer successivo.

Connessioni tra ultimo layer nascosto e layer di output:

Ogni nodo nel layer di output è connesso a tutti i nodi dell'ultimo layer nascosto.

Quindi, il numero di connessioni è: $s \cdot p$

I pesi associati a queste connessioni rappresentano i pesi tra i nodi dell'ultimo layer nascosto e i nodi del layer di output.

Numero totale di pesi:

Sommando il numero di connessioni per tutti i tipi di connessioni, otteniamo il numero totale di pesi nella rete MLP:

$$\text{Numero totale di pesi} = p \cdot d + l \cdot p^2 + s \cdot p$$

Esempio:

Se la rete ha 4 nodi di input, 3 nodi di output, 2 layer nascosti con 5 nodi ciascuno, il numero totale di pesi sarà:

$$\text{Numero totale di pesi} = 5 * 4 + 2 * 5^2 + 3 * 5 = 20 + 50 + 15 = 85$$

Uno dei primi algoritmi proposti per il calcolo dei pesi in una rete neurale è il metodo iterativo noto come **metodo di backpropagation**. La riscoperta di questo metodo verso la metà degli anni '80 da parte di **Rumelhart, Hinton e Williams** ha reso possibile definire algoritmi di addestramento per reti multistrato e quindi è stata alla base del successivo considerevole sviluppo degli studi sulle reti neurali negli ultimi due decenni.

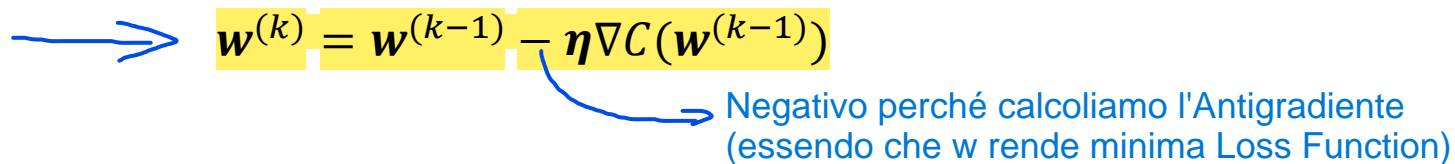
Il metodo di *backpropagation* (BP) è tuttora uno dei metodi di addestramento più diffusi. Il termine “**backpropagation**” (retropropagazione) è legato essenzialmente **alla tecnica utilizzata per il calcolo delle derivate della funzione di errore, basata sulle regole di derivazione delle funzioni composte**

Per calcolare il vettore dei pesi

$$\mathbf{w}^* = \{w^{(1)}, w^{(2)}, \dots, w^{(l)}\} \in \arg \min_{\mathbf{w}} C(\mathbf{w}) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(\mathbf{y}^{(j)}, \hat{\mathbf{y}}^{(j)})$$

ricorriamo al metodo del gradiente

Il metodo di backpropagation si basa effettivamente su una combinazione del Gradient Descent e Chain Rule per calcolare i gradienti necessari per aggiornare i pesi della rete neurale.



The diagram shows the weight update equation $\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} - \eta \nabla C(\mathbf{w}^{(k-1)})$. A blue arrow points from the left towards the equation. Another blue arrow points from the minus sign in the equation to the explanatory text below.

$$\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} - \eta \nabla C(\mathbf{w}^{(k-1)})$$

Negativo perché calcoliamo l'Antigradiente
(essendo che w rende minima Loss Function)

dove η rappresenta lo step-size (che nel contesto del machine learning prende il nome **di learning rate**)

Risulta quindi necessario calcolare $\nabla C(\mathbf{w})$

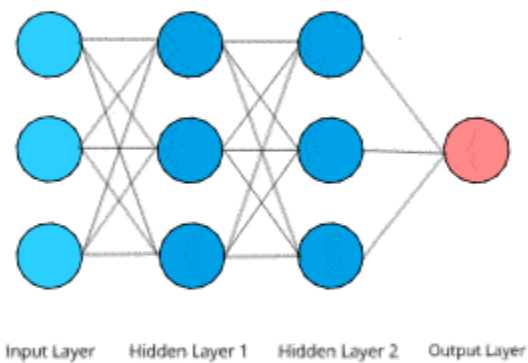
Adesso introduciamo la **procedura di calcolo del gradiente mediante backpropagation**, che si può ricondurre a una particolare tecnica di *differenziazione automatica* .

Richiamiamo prima i seguenti strumenti di calcolo differenziale utili per la comprensione della **backpropagation**

Feed new data



X1
X2
X3



Y_pred



Error

Y

Derivata di una funzione composta $R \rightarrow R$

Chain rule

- Se $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $g \circ f: R \rightarrow R$ è derivabile,
e se poniamo $h(x) = g(f(x))$
- $\frac{dh}{dx} = g'(f(x)) \cdot f'(x)$
- Se $q: R \rightarrow R$ e $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $q \circ g \circ f: R \rightarrow R$ è derivabile,

$$h(x) = q(g(f(x)))$$

- $\frac{dh}{dx} = q'(g(f(x))) \cdot g'(f(x)) \cdot f'(x)$

Derivata composta di funzioni di più variabili reali:

- Se $x(t) = (x_1(t), x_2(t), \dots, x_n(t))^T$ è un vettore di R^n le cui componenti sono funzioni derivabili e se f è una funzione differenziabile in $x(t)$, allora la funzione composta $F(t) = f(x(t))$ è differenziabile nella variabile t e si ha:

$$F'(t) = \sum_{i=1}^n \frac{\partial f(x(t))}{\partial x_i} \cdot x'_i(t) = \langle \nabla f(x(t)), x'(t) \rangle$$

- Esempio: data la funzione $f(h(t), g(t))$, la derivata di f rispetto a t si calcola come:

$$\frac{df}{dt} = \frac{\partial f}{\partial h} \frac{dh}{dt} + \frac{\partial f}{\partial g} \frac{dg}{dt}$$

Esempio:

Sia $x(t) = (x_1(t), x_2(t))^T = (t, e^t)^T$ e sia $f(x) = f(x_1, x_2) = x_1 + x_2^2$

Il teorema afferma che la derivata di $F(t) = f(x(t))$ rispetto a t si ottiene facendo il prodotto scalare del gradiente di f valutato in $x(t)$ per il vettore $x'(t) = (x_1'(t), x_2'(t))^T$

$$\frac{dF(t)}{dt} = \nabla f(x(t))^T \cdot x'(t) =$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 \\ 2x_2 \end{bmatrix}$$

$$\text{Valutazione di } \nabla f(x(t)) = \nabla f(t, e^t) = \begin{bmatrix} 1 \\ 2e^t \end{bmatrix}$$

$$x'(t) = (x_1'(t), x_2'(t))^T = [1, e^t]^T$$

$$\frac{dF(t)}{dt} = \nabla f(x(t))^T \cdot x'(t) = [1 \ 2e^t] \begin{bmatrix} 1 \\ e^t \end{bmatrix} = 1 + 2e^{2t}$$

Verifica

Sostituiamo x_1 con t e x_2 con e^t

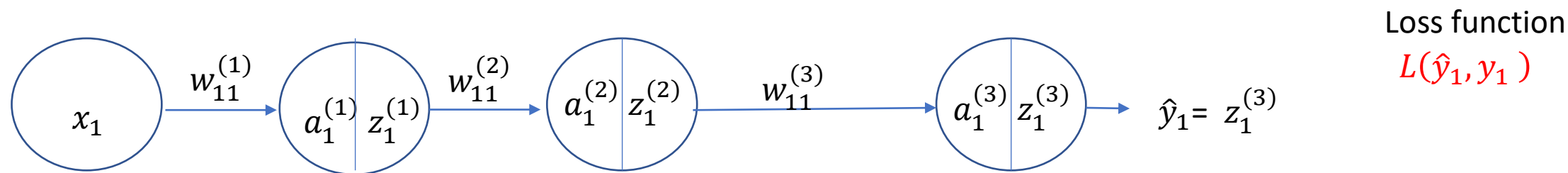
Avremo

$$f(t) = t + e^{2t}$$

Calcoliamo

$$\frac{df(t)}{dt} = 1 + 2e^{2t}$$

Consideriamo il seguente esempio di rete MLP molto semplice, formata da un nodo di input, 2 layer nascosti ciascuno dei quali costituito da un solo neurone ed un nodo di output:



Sia y_1 l'etichetta di x_1 . La loss function che misura l'errore tra l'output prodotto dalla rete e il valore atteso è $L(y_1, \hat{y}_1)$.

Ricordiamo che il nostro compito è, in questo semplice esempio, aggiornare i pesi w in maniera tale da rendere minimo $L(y_1, \hat{y}_1)$, cioè individuare i pesi w tali che

$$w \in \arg \min_w L(y_1, \hat{y}_1(w))$$

Per ottenere l'insieme di pesi $w = [w_{11}^{(1)}, w_{11}^{(2)}, w_{11}^{(3)}]$ ricorreremo al metodo di discesa del gradiente.

$$w^{k+1} = w^k - \eta \nabla L(w^k),$$

E' necessario calcolare:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} \\ \frac{\partial L}{\partial w_{11}^{(2)}} \\ \frac{\partial L}{\partial w_{11}^{(3)}} \end{bmatrix}$$

Calcoliamo la **derivata parziale di L rispetto a $w_{11}^{(3)}$**

$$L(\hat{y}_1) = L(z_1^{(3)}) = L(z_1^{(3)}(a_1^{(3)})) = L(z_1^{(3)}(a_1^{(3)}(w_{11}^{(3)})))$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}}$$

$$\frac{\partial L}{\partial z_1^{(3)}} = \frac{\partial L}{\partial \hat{y}_1}$$

Si calcola facilmente
derivando rispetto all'output della rete
l'espressione analitica della loss-function

$$z_1^{(3)} = f(a_1^{(3)}) \quad \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} = f'(a_1^{(3)})$$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)} \quad \frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}} = z_1^{(2)}$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) z_1^{(2)}$$

Calcoliamo **la derivata parziale di L rispetto a $w_{11}^{(2)}$**

$$L(\hat{y}_1) = L(z_1^{(3)}) = L(z_1^{(3)}(a_1^{(3)})) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(w_{11}^{(2)}))))$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)}$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} = w_{11}^{(3)}$$

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)} \quad \frac{\partial a_1^{(2)}}{\partial w_{11}^{(2)}} = z_1^{(1)}$$

$$z_1^{(2)} = f(a_1^{(2)}) \quad \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} = f'(a_1^{(2)})$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) z_1^{(1)}$$

Calcoliamo la **derivata parziale di L rispetto a $w_{11}^{(1)}$**

$$L(\hat{y}_1) = L(z_1^{(3)}) = L(z_1^{(3)}(a_1^{(3)})) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)})))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(z_1^{(1)})))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(z_1^{(1)}(a_1^{(1)}(w_{11}^{(1)})))))) =$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}}$$

$$\frac{\partial L}{\partial z_1^{(3)}} = \frac{\partial L}{\partial \hat{y}_1}$$

Si calcola facilmente

derivando rispetto all'output della rete
l'espressione analitica della loss-function

$$z_1^{(3)} = f(a_1^{(3)}) \quad \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} = f'(a_1^{(3)})$$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)} \quad \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} = w_{11}^{(3)}$$

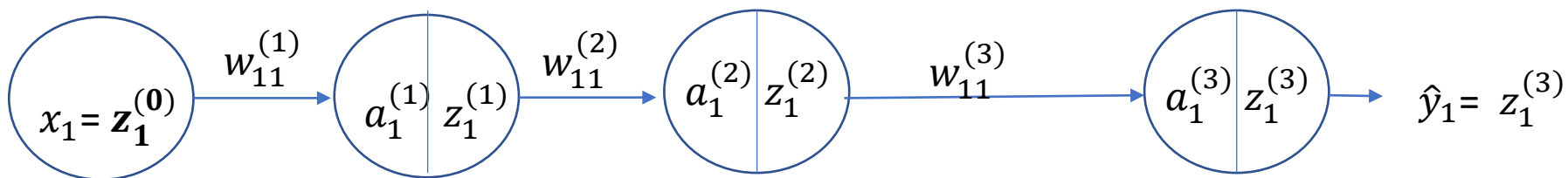
$$z_1^{(2)} = f(a_1^{(2)}) \quad \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} = f'(a_1^{(2)})$$

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)} \quad \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} = w_{11}^{(2)}$$

$$z_1^{(1)} = f(a_1^{(1)}) \quad \frac{\partial z_1^{(1)}}{\partial a_1^{(1)}} = f'(a_1^{(1)})$$

$$z_1^{(0)} = x_1, \quad a_1^{(1)} = w_{11}^{(1)} z_1^{(0)} \quad \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}} = z_1^{(0)}$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) w_{11}^{(2)} f'(a_1^{(1)}) z_1^{(0)}$$



$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) z_1^{(2)}$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) z_1^{(1)}$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) w_{11}^{(2)} f'(a_1^{(1)}) z_1^{(0)} =$$

Poniamo:

$$\delta_1^{(3)} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)})$$

$$\delta_1^{(2)} = \delta_1^{(3)} w_{11}^{(3)} f'(a_1^{(2)})$$

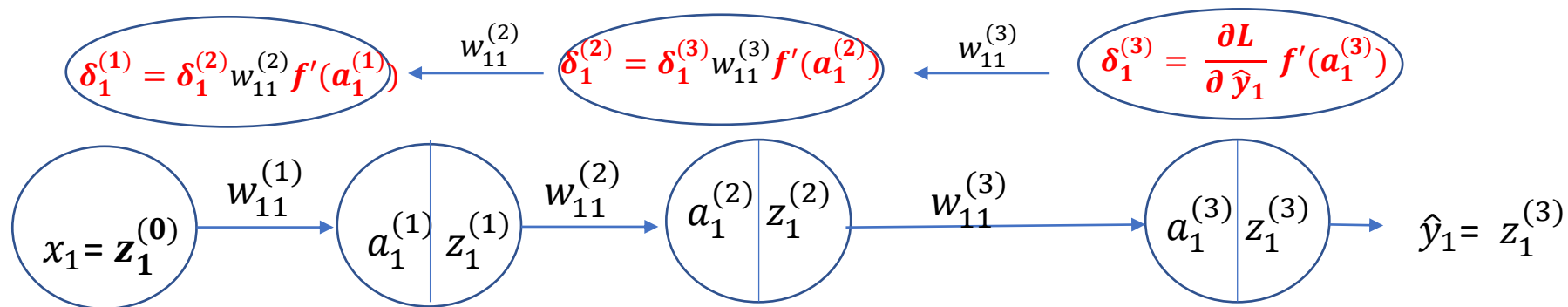
$$\delta_1^{(1)} = \delta_1^{(2)} w_{11}^{(2)} f'(a_1^{(1)})$$

Le formule del gradiente della loss function L rispetto a tutti i pesi $\{w_{11}^{(1)}, w_{11}^{(2)}, w_{11}^{(3)}\}$ si potranno quindi così esprimere:

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \delta_1^{(3)} z_1^{(2)}$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \delta_1^{(2)} z_1^{(1)}$$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \delta_1^{(1)} z_1^{(0)}$$



Aggiornamento dei pesi per l'epoca successiva:

$$w^{k+1} = w^k - \eta \nabla L(w^k),$$

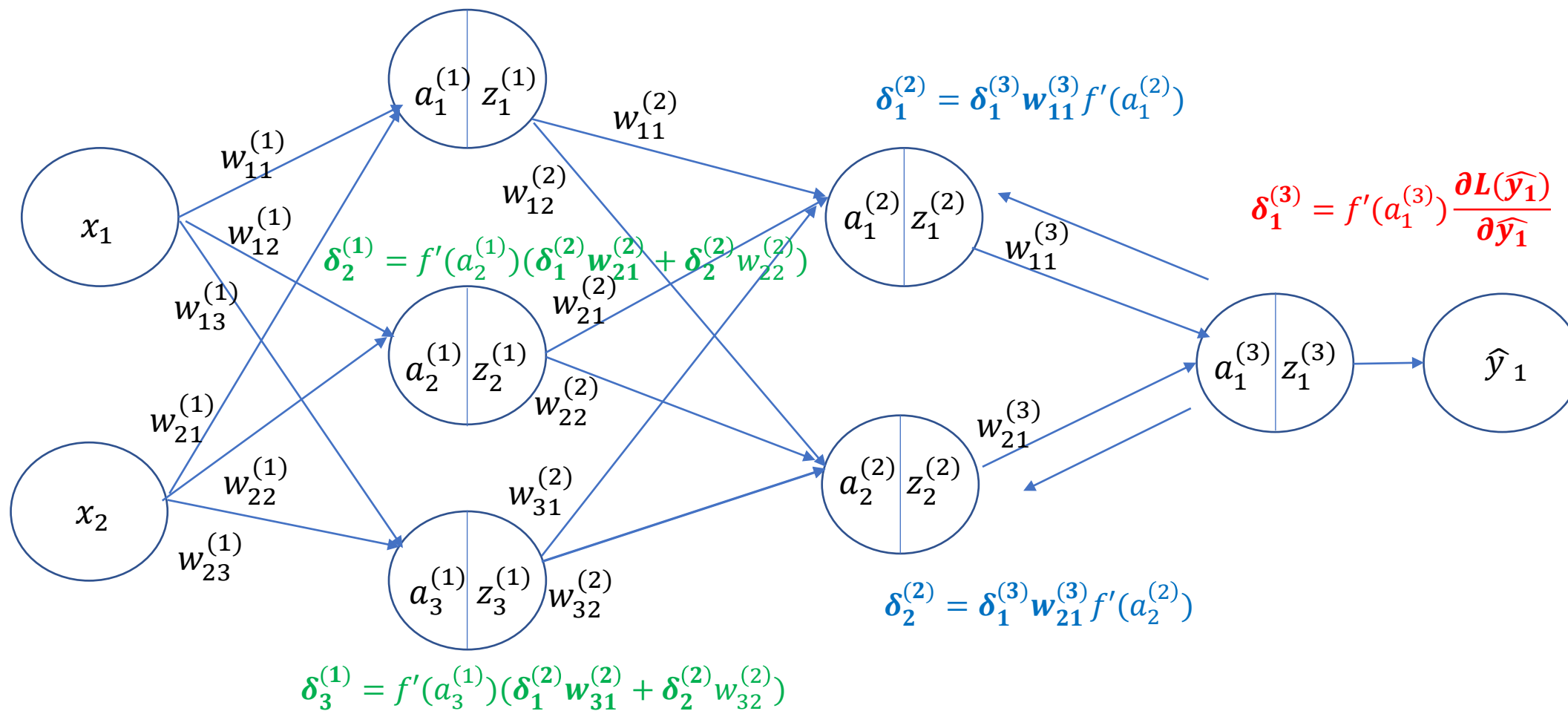
Omettendo per semplicità di scrittura l'indice k dell'epoca, l'ultima relazione è equivalente a:

$$w_{11}^{(3)} = w_{11}^{(3)} - \eta \delta_1^{(3)} z_1^{(2)}$$

$$w_{11}^{(2)} = w_{11}^{(2)} - \eta \delta_1^{(2)} z_1^{(1)}$$

$$w_{11}^{(1)} = w_{11}^{(1)} - \eta \delta_1^{(1)} x_1$$

$$\delta_1^{(1)} = f'(a_1^{(1)}) (\delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{12}^{(2)})$$



$$\frac{\partial L}{\partial w_{ji}^{(\ell)}} = \delta_i^{(\ell)} z_j^{(\ell-1)}$$

dove, se il neurone $i, i = 1, \dots, k$, appartiene al layer di uscita L ,

$$\delta_i^{(L)} = \frac{\partial L(\hat{y}_i)}{\partial a_i^{(L)}} = f'(a_i^{(L)}) \frac{\partial L(\hat{y}_i)}{\partial \hat{y}_i}$$

se il neurone $i, i = 1, \dots, k$, non appartiene al layer di uscita, ma al layer nascosto ℓ ,

$$\delta_i^{(\ell)} = f'(a_i^{(\ell)}) \sum_{k=0}^{N^{(\ell)}} \delta_k^{(\ell+1)} w_{ik}^{(\ell+1)}$$

dove $N^{(\ell)}$ è il numero di neuroni nascosti del layer ℓ -esimo

Calcolo di $\frac{\partial L}{\partial \hat{y}_i}$

Sia $\hat{y} = [\hat{y}_1, \dots, \hat{y}_s]$ l'output prodotto dalla rete in corrispondenza all'esempio di training $x = [x_1, x_2, \dots, x_d]$, e sia $y = [y_1, y_2, \dots, y_s]$ il vettore delle etichette corrispondente dell'esempio di training.

Se scegliamo come **loss function** la somma dei quadrati degli errori tra l'output della rete e la corrispondente etichetta:

$$L(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^s (y_i - \hat{y}_i)^2 = \frac{1}{2} \|y - \hat{y}\|_2^2$$

Ricordando che:

$$L(y, \hat{y}) = \frac{1}{2} ((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \dots + (y_i - \hat{y}_i)^2 + \dots + (y_s - \hat{y}_s)^2)$$

Segue facilmente che la derivata parziale di L rispetto all'output della rete $\hat{y}_i, i = 1, \dots, s$ sarà dato da :

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{1}{2} 2(y_i - \hat{y}_i)(-1) = -(y_i - \hat{y}_i) = \hat{y}_i - y_i$$

Tecniche di ottimizzazione: Gradiente e sue varianti

Batch Gradient Descent

Per il calcolo della funzione costo $C(w)$ vengono usati tutti gli n_T campioni del training set .

$$T = \{(x^{(j)}, y^{(j)}), x^{(j)} \in \mathbb{R}^d, y^{(j)} \in \mathbb{R}^s, j = 1, \dots, n_T\},$$

$$w^* \in \arg \min_w \left\{ C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)}) \right\}$$

Quindi:

- si considera l'intero set di addestramento,
- si esegue la Forward Propagation e si calcola la funzione di costo.
- si aggiorniamo i parametri usando il tasso di variazione di questa funzione di costo rispetto ai parametri.

Epoca: l'intero set di addestramento viene passato attraverso il modello, vengono eseguite la propagazione in avanti e la propagazione all'indietro e i parametri vengono aggiornati.

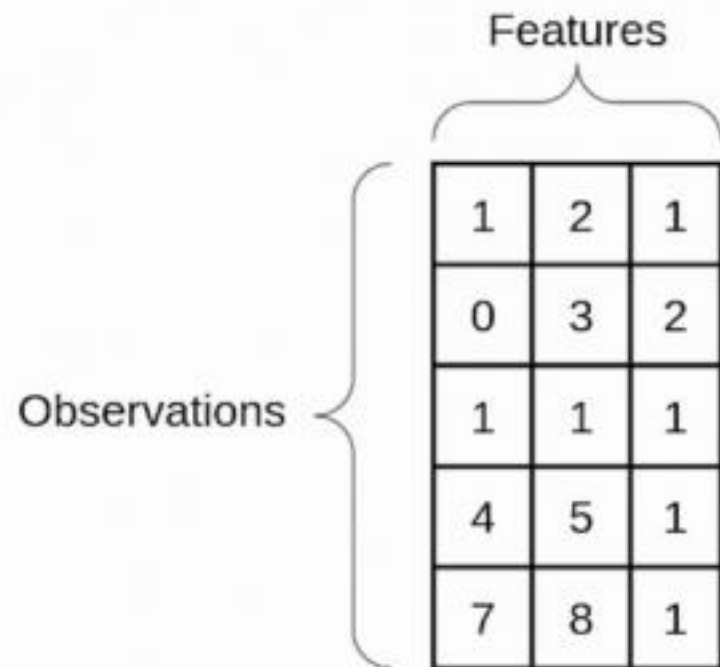
Nel batch Gradient Descent poiché stiamo utilizzando l'intero set di addestramento, i parametri verranno aggiornati solo una volta per epoca.

Discesa del gradiente stocastico (SGD)

Se si utilizza una singola osservazione per calcolare la funzione di costo, si parla di **Stochastic Gradient Descent**, comunemente abbreviato **SGD**. Passiamo una sola osservazione alla volta, calcoliamo il costo e aggiorniamo i parametri.

Esempio:

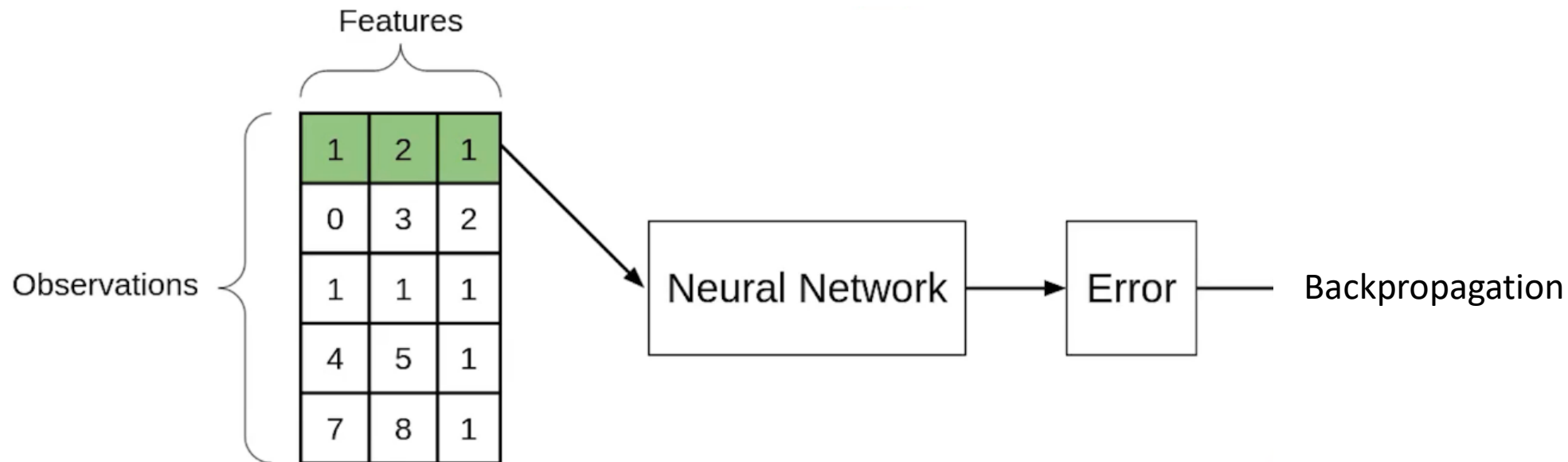
Supponiamo di avere un dataset di training formato da 5 campioni, ognuno dei quali costituito da 3 input



The diagram shows a dataset with 5 observations and 3 features. A bracket on the left labeled 'Observations' spans the five rows. A bracket on top labeled 'Features' spans the three columns. The data is as follows:

Features		
1	2	1
0	3	2
1	1	1
4	5	1
7	8	1

Ora, se usiamo l'SGD, prenderemo la prima osservazione, poi la passeremo attraverso la rete neurale, calcoleremo l'errore e quindi aggiorneremo i parametri.



Quindi entrerà in input la seconda osservazione ed saranno eseguiti passaggi simili su di essa. Questo passaggio verrà ripetuto fino a quando tutte le osservazioni non saranno passate attraverso la rete e i parametri non saranno stati aggiornati.

Ogni aggiornamento dei parametri, viene chiamato **Iterazione**.

In questo esempio, poiché abbiamo 5 osservazioni, i parametri verranno aggiornati 5 volte (sono state effettuate 5 iterazioni).

Nel caso del Batch Gradient Descent tutte le osservazioni insieme sarebbero state elaborate dalla rete ed i parametri sarebbero stati aggiornati solo una volta.

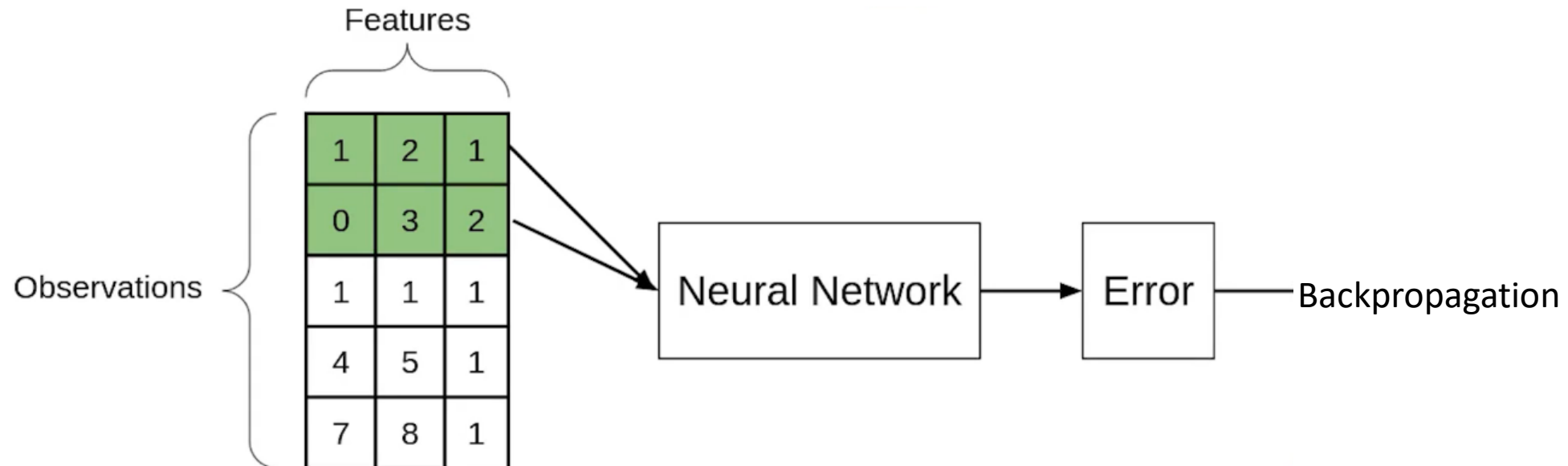
Nel caso di SGD, ci saranno n_T iterazioni per epoca, dove n_T è il numero di osservazioni nel dataset di training.

Se si utilizza l'intero set di dati per calcolare la funzione di costo si parla di **Batch Gradient Descent** e se si utilizza una singola osservazione per calcolare il costo si parla di **SGD**.

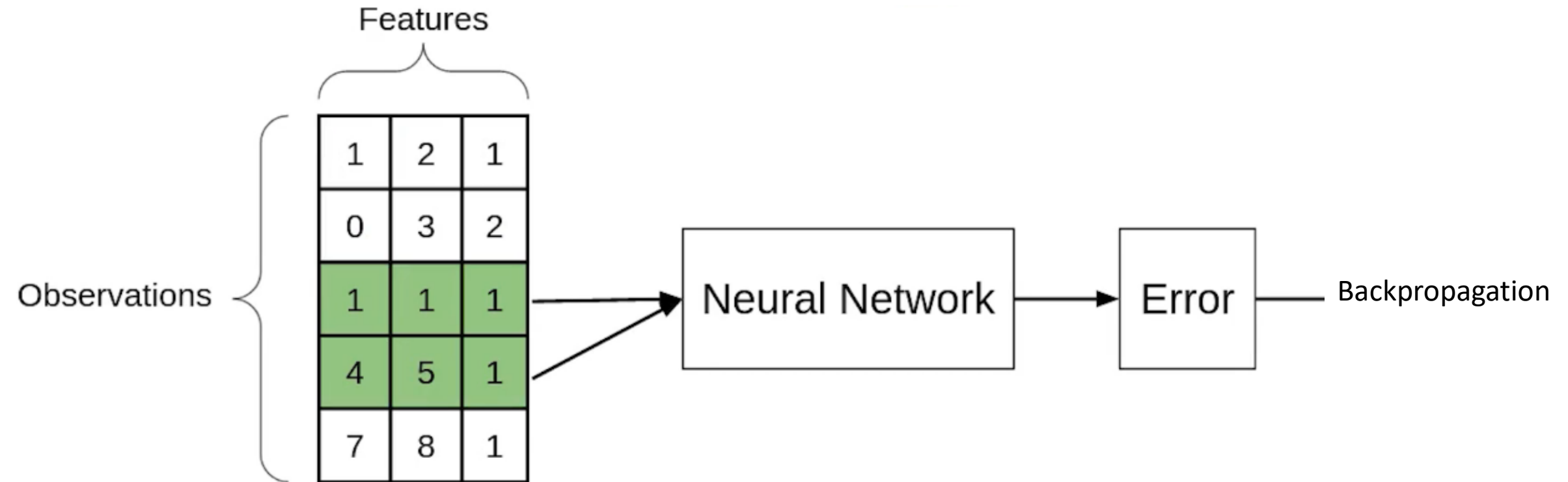
Mini-batch Stochastic Gradient Descent:

Per calcolare la funzione di costo si considera un sottoinsieme dell'intero set di dati. Quindi, se ci sono n_T osservazioni, il numero di osservazioni in ciascun sottoinsieme o mini-batch sarà maggiore di 1 e minore di n_T .

Ancora una volta prendiamo lo stesso esempio. Supponiamo che la dimensione del batch sia 2. Quindi prenderemo le prime due osservazioni, le passeremo attraverso la rete neurale, calcoleremo l'errore e quindi aggiorneremo i parametri.



Quindi prenderemo le due osservazioni successive ed eseguiremo passaggi simili, passeremo attraverso la rete, calcoleremo l'errore e aggiorneremo i parametri.



Poiché ci rimane la singola osservazione nell'iterazione finale, ci sarà solo una singola osservazione e i parametri saranno aggiornati utilizzando questa osservazione.

Confronto tra Batch GD, SGD e Mini-batch SGD:

Segue un confronto tra le diverse varianti di Gradient Descent

Confronto: numero di osservazioni utilizzate per l'aggiornamento.

In **batch Gradient Descent**

- prendiamo l'intero set di dati
- calcoliamo la funzione di costo
- aggiorniamo i parametri

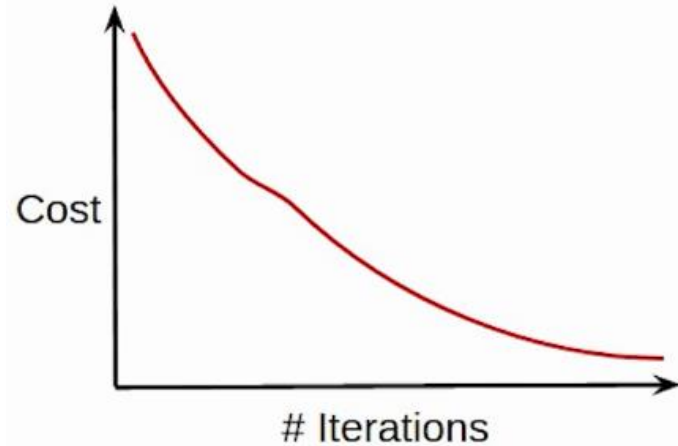
Nel caso di **Stochastic Gradient Descent**, aggiorniamo i parametri dopo ogni singola osservazione e sappiamo che ogni volta che i pesi vengono aggiornati si parla di iterazione.

Nel caso di **Mini-batch Stochastic Gradient Descent**, prendiamo un sottoinsieme di dati e aggiorniamo i parametri in base a ogni sottoinsieme.

Confronto Funzione costo

Batch

- Cost function reduces smoothly



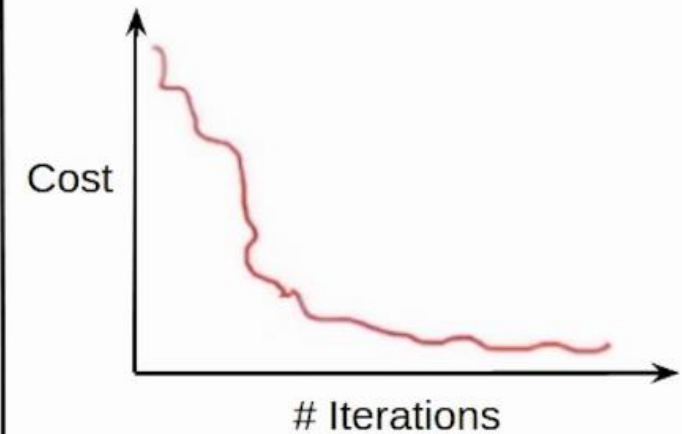
Stochastic

- Lot of variations in cost function



Mini-Batch

- Smoother cost function as compared to SGD



- Poiché nel caso del **Batch GD** aggiorniamo i parametri utilizzando l'intero set di dati la funzione costo, in questo caso, si riduce uniformemente.
- Questo aggiornamento nel caso di **SGD** non è così fluido. Poiché stiamo aggiornando i parametri sulla base di una singola osservazione, ci sono molte iterazioni. Potrebbe anche essere possibile che il modello inizi ad apprendere anche il rumore.
- L'aggiornamento della funzione di costo nel caso di **Mini-batch Gradient Descent** è più fluido rispetto a quello della funzione di costo in SGD. Dal momento che non aggiorniamo i parametri dopo ogni singola osservazione ma dopo ogni sottoinsieme dei dati

Veniamo ora al **costo di calcolo e al tempo impiegato** da queste varianti **di Gradient Descent**.

Poiché dobbiamo caricare l'intero set di dati alla volta, eseguire la propagazione in avanti su di esso e calcolare l'errore e quindi aggiornare i parametri, **nel caso Batch Gradient Descent** il costo di calcolo è molto elevato.

Il tempo di calcolo della funzione costo nel caso di **SGD** è inferiore rispetto a **Batch Gradient Descent** poiché dobbiamo caricare ogni singola osservazione alla volta, ma il tempo di calcolo qui aumenta poiché ci sarà un numero maggiore di aggiornamenti che si tradurrà in un numero maggiore di iterazioni .

Nel caso di **Mini-batch Stochastic Gradient Descent**, prendendo un sottoinsieme dei dati ci sono un numero minore di iterazioni o aggiornamenti e quindi **il tempo di calcolo** nel caso di mini-batch Gradient Descent è inferiore a SGD.

Inoltre, poiché non stiamo caricando l'intero set di dati alla volta mentre carichiamo un sottoinsieme dei dati, anche il tempo di calcolo della funzione costo è inferiore rispetto alla discesa del gradiente batch. Questo è il motivo per cui di solito **si preferisce utilizzare il Mini-batch Stochastic Gradient Descent**.

Iperparametri

Gli iperparametri sono parametri esterni al modello di machine learning che devono essere impostati prima dell'avvio del processo di addestramento. A differenza dei parametri del modello, che vengono appresi durante il processo di addestramento stesso, gli iperparametri influenzano il comportamento del processo di addestramento e la configurazione del modello.

Gli **iperparametri** determinano come avviene l'addestramento del modello e possono includere:

- 1. Learning rate** : Determina quanto velocemente o lentamente il modello si adatta ai dati.
- 2. Numero di epoche** : il numero di volte in cui l'intero set di dati di addestramento viene utilizzato per addestrare il modello. Un numero insufficiente di epoche può portare a un modello non addestrato adeguatamente, mentre un numero eccessivo di epoche può portare a un overfitting.
- 3. Dimensione del mini-batch** : il numero di esempi di addestramento utilizzati in ciascuna iterazione dell'algoritmo di ottimizzazione (ad esempio, SGD o mini-batch GD). La dimensione del mini-batch può influenzare la velocità di apprendimento e la stabilità dell'addestramento.

4. **Architettura del modello**: la struttura del modello, inclusi il numero di strati, il numero di unità in ciascun strato e le connessioni tra gli strati. La scelta dell'architettura dipende dal problema e dai dati specifici.
5. **Regolarizzazione**: i parametri che controllano la regolarizzazione del modello, come il peso della regolarizzazione L1 o L2, che influenzano la complessità del modello e la tendenza all'overfitting.
6. **Inizializzazione dei pesi**: il metodo utilizzato per inizializzare i pesi del modello. Una buona inizializzazione può favorire una convergenza più rapida e una migliore performance.
7. **Funzione di attivazione**: la funzione utilizzata per calcolare l'output di un'unità nel modello. Le funzioni di attivazione comuni includono ReLU, sigmoid e tanh.

La scelta corretta degli iperparametri può influenzare significativamente le prestazioni del modello. Spesso, gli iperparametri vengono regolati attraverso tentativi ed errori, eseguendo iterazioni multiple di addestramento e valutando le prestazioni su un set di dati di validazione. L'esperienza, la conoscenza del dominio e le best practice possono guidare la scelta degli iperparametri ottimali per un determinato problema di machine learning.