

# **Implementazione di una Rete Convoluzionale in CUDA**

Michele Valsesia

Nicholas Aspes

Anno accademico 2018/2019

# Introduzione

## Obiettivi

- Descrivere l'architettura ed il funzionamento di una *Rete Neurale Semplice* e di una *Convoluzionale*

# Introduzione

## Obiettivi

- ▶ Descrivere l'architettura ed il funzionamento di una *Rete Neurale Semplice* e di una *Convoluzionale*
- ▶ Motivare le differenti scelte implementative adottate durante lo svolgimento del progetto

# Introduzione

## Obiettivi

- ▶ Descrivere l'architettura ed il funzionamento di una *Rete Neurale Semplice* e di una *Convoluzionale*
- ▶ Motivare le differenti scelte implementative adottate durante lo svolgimento del progetto
- ▶ Valutare l'accuratezza e lo speedup della rete rispetto ad una implementazione di tipo sequenziale

# Reti Neurali

# Reti Neurali

## Scopo

- Le *Reti Neurali* vengono principalmente usate per la classificazione di immagini

# Reti Neurali

## Scopo

- ▶ Le *Reti Neurali* vengono principalmente usate per la classificazione di immagini
- ▶ Il processo di classificazione consiste nell'assegnare ad un immagine un'etichetta che identifichi nel miglior modo possibile il suo contenuto semantico

# Reti Neurali

## Scopo

- ▶ Le *Reti Neurali* vengono principalmente usate per la classificazione di immagini
- ▶ Il processo di classificazione consiste nell'assegnare ad un immagine un'etichetta che identifichi nel miglior modo possibile il suo contenuto semantico
- ▶ Un'etichetta è meglio conosciuta con il nome di *classe*



# Reti Neurali

## Scopo

- ▶ Le *Reti Neurali* vengono principalmente usate per la classificazione di immagini
- ▶ Il processo di classificazione consiste nell'assegnare ad un immagine un'etichetta che identifichi nel miglior modo possibile il suo contenuto semantico
- ▶ Un'etichetta è meglio conosciuta con il nome di *classe*
- ▶ Le reti neurali ricevono in input un'immagine e restituiscono in output la relativa classe

# Reti Neurali

## Funzionamento

- Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi

# Reti Neurali

## Funzionamento

- ▶ Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi
- ▶ Un *esempio* è una coppia (immagine, etichetta)

# Reti Neurali

## Funzionamento

- ▶ Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi
- ▶ Un *esempio* è una coppia (immagine, etichetta)
- ▶ Un esempio viene creato da un team di persone che valuta il contenuto semantico di un immagine e le associa l'etichetta più adatta

# Reti Neurali

## Funzionamento

- ▶ Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi
- ▶ Un *esempio* è una coppia (immagine, etichetta)
- ▶ Un esempio viene creato da un team di persone che valuta il contenuto semantico di un immagine e le associa l'etichetta più adatta
- ▶ Il *training set* ed il *test set* sono insiemi di esempi

# Reti Neurali

## Funzionamento

- ▶ Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi
- ▶ Un *esempio* è una coppia (immagine, etichetta)
- ▶ Un esempio viene creato da un team di persone che valuta il contenuto semantico di un immagine e le associa l'etichetta più adatta
- ▶ Il *training set* ed il *test set* sono insiemi di esempi
- ▶ Il training set viene usato per l'addestramento (training) della rete

# Reti Neurali

## Funzionamento

- ▶ Una rete neurale deve *apprendere* come assegnare correttamente alle immagini le varie classi
- ▶ Un *esempio* è una coppia (immagine, etichetta)
- ▶ Un esempio viene creato da un team di persone che valuta il contenuto semantico di un immagine e le associa l'etichetta più adatta
- ▶ Il *training set* ed il *test set* sono insiemi di esempi
- ▶ Il training set viene usato per l'addestramento (training) della rete
- ▶ Il test set serve a controllare che la rete abbia imparato a discriminare correttamente le immagini

# Reti Neurali

## Training

- ▶ Per ognuno degli esempi del training set



# Reti Neurali

## Training

- ▶ Per ognuno degli esempi del training set
  - La rete assegna all'immagine corrente la classe che meglio rappresenta il suo contenuto semantico

# Reti Neurali

## Training

- ▶ Per ognuno degli esempi del training set
  - La rete assegna all'immagine corrente la classe che meglio rappresenta il suo contenuto semantico
  - Se la classe di output è diversa dall'etichetta dell'esempio, la rete corregge i suoi parametri interni e passa all'immagine successiva

# Reti Neurali

## Testing

- L'*accuratezza* della rete è data dal rapporto tra il numero di esempi classificati correttamente e la cardinalità del test set

# Reti Neurali

## Testing

- ▶ L'*accuratezza* della rete è data dal rapporto tra il numero di esempi classificati correttamente e la cardinalità del test set
- ▶ Per ognuno degli esempi del test set

# Reti Neurali

## Testing

- ▶ L'*accuratezza* della rete è data dal rapporto tra il numero di esempi classificati correttamente e la cardinalità del test set
- ▶ Per ognuno degli esempi del test set
  - La rete assegna all'immagine corrente la classe che meglio rappresenta il suo contenuto semantico

# Reti Neurali

## Testing

- ▶ L'*accuratezza* della rete è data dal rapporto tra il numero di esempi classificati correttamente e la cardinalità del test set
- ▶ Per ognuno degli esempi del test set
  - La rete assegna all'immagine corrente la classe che meglio rappresenta il suo contenuto semantico
  - Per sapere il numero di immagini classificate correttamente dalla rete è necessario definire un contatore

# Reti Neurali

## Testing

- ▶ L'*accuratezza* della rete è data dal rapporto tra il numero di esempi classificati correttamente e la cardinalità del test set
- ▶ Per ognuno degli esempi del test set
  - La rete assegna all'immagine corrente la classe che meglio rappresenta il suo contenuto semantico
  - Per sapere il numero di immagini classificate correttamente dalla rete è necessario definire un contatore
  - Il contatore viene incrementato quando l'output prodotto è uguale all'etichetta dell'esempio considerato

# Reti Neurali

## Significato Biologico

- Le *Reti Neurali* nascono con lo scopo di modellare una rete neurale biologica



# Reti Neurali

## Significato Biologico

- ▶ Le *Reti Neurali* nascono con lo scopo di modellare una rete neurale biologica
- ▶ Una rete neurale biologica si compone di unità cellulari di base: i *neuroni*

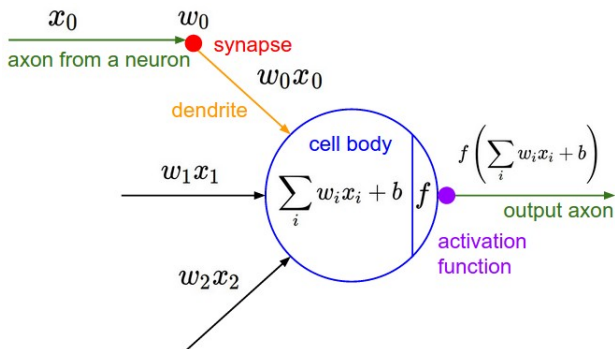
# Reti Neurali

## Significato Biologico

- ▶ Le *Reti Neurali* nascono con lo scopo di modellare una rete neurale biologica
- ▶ Una rete neurale biologica si compone di unità cellulari di base: i *neuroni*
- ▶ I neuroni sono collegati tra loro per mezzo di specifiche giunture chiamate *sinapsi*

# Reti Neurali

## Neurone



*Modello matematico di un neurone*

# Reti Neurali

## Funzionamento Neurone

- Attraverso un meccanismo di eccitazione ed inibizione i pesi sinaptici controllano quanto un neurone sia influenzato dagli altri

# Reti Neurali

## Funzionamento Neurone

- ▶ Attraverso un meccanismo di eccitazione ed inibizione i pesi sinaptici controllano quanto un neurone sia influenzato dagli altri
- ▶ I segnali in ingresso al neurone vengono pesati dalle differenti sinapsi, trasportati dai dendriti all'interno del corpo cellulare e sommati tra loro

# Reti Neurali

## Funzionamento Neurone

- ▶ Attraverso un meccanismo di eccitazione ed inibizione i pesi sinaptici controllano quanto un neurone sia influenzato dagli altri
- ▶ I segnali in ingresso al neurone vengono pesati dalle differenti sinapsi, trasportati dai dendriti all'interno del corpo cellulare e sommati tra loro
- ▶ Quando la somma supera una certa soglia, il neurone *spara* un segnale lungo l'assone

# Reti Neurali

## Funzionamento Neurone

- ▶ Attraverso un meccanismo di eccitazione ed inibizione i pesi sinaptici controllano quanto un neurone sia influenzato dagli altri
- ▶ I segnali in ingresso al neurone vengono pesati dalle differenti sinapsi, trasportati dai dendriti all'interno del corpo cellulare e sommati tra loro
- ▶ Quando la somma supera una certa soglia, il neurone *spara* un segnale lungo l'assone
- ▶ La *frequenza di sparo* del neurone viene modellata con una funzione di attivazione  $f$

# Reti Neurali

## Funzioni di Attivazione

### Definizione

Una *funzione di attivazione* è una funzione matematica non lineare usata per modellare l'output di un neurone. L'input è dato dalla somma pesata dei segnali in ingresso al neurone



# Reti Neurali

## Funzioni di Attivazione

### Definizione

Una *funzione di attivazione* è una funzione matematica non lineare usata per modellare l'output di un neurone. L'input è dato dalla somma pesata dei segnali in ingresso al neurone

- *Rectifier Linear Unit*

# Reti Neurali

## Funzioni di Attivazione

### Definizione

Una *funzione di attivazione* è una funzione matematica non lineare usata per modellare l'output di un neurone. L'input è dato dalla somma pesata dei segnali in ingresso al neurone

- ▶ *Rectifier Linear Unit*
- ▶ *Sigmoide*

# Reti Neurali

## Funzioni di Attivazione

### Definizione

Una *funzione di attivazione* è una funzione matematica non lineare usata per modellare l'output di un neurone. L'input è dato dalla somma pesata dei segnali in ingresso al neurone

- ▶ *Rectifier Linear Unit*
- ▶ *Sigmoide*
- ▶ *Tangente Iperbolica*

# Reti Neurali

## Funzioni di Attivazione

### Definizione

Una *funzione di attivazione* è una funzione matematica non lineare usata per modellare l'output di un neurone. L'input è dato dalla somma pesata dei segnali in ingresso al neurone

- ▶ *Rectifier Linear Unit*
- ▶ *Sigmoide*
- ▶ *Tangente Iperbolica*
- ▶ *Softplus*

# Reti Neurali

## Rectifier Linear Unit

### Definizione

La *Rectifier Linear Unit (ReLU)*  $r : \mathbb{R} \rightarrow [0, +\infty)$  è definita come  $r(x) = \max(0, x)$

# Reti Neurali

## Rectifier Linear Unit

### Definizione

La *Rectifier Linear Unit (ReLU)*  $r : \mathbb{R} \rightarrow [0, +\infty)$  è definita come  $r(x) = \max(0, x)$

- Si differenzia da una funzione di tipo lineare per metà del suo dominio in quanto  $\forall x < 0, \max(0, x) = 0$

# Reti Neurali

## Rectifier Linear Unit

### Definizione

La *Rectifier Linear Unit (ReLU)*  $r : \mathbb{R} \rightarrow [0, +\infty)$  è definita come  $r(x) = \max(0, x)$

- ▶ Si differenzia da una funzione di tipo lineare per metà del suo dominio in quanto  $\forall x < 0, \max(0, x) = 0$
- ▶ Presenta un punto di discontinuità in  $x = 0$

# Reti Neurali

## Rectifier Linear Unit

### Definizione

La *Rectifier Linear Unit (ReLU)*  $r : \mathbb{R} \rightarrow [0, +\infty)$  è definita come  $r(x) = \max(0, x)$

- ▶ Si differenzia da una funzione di tipo lineare per metà del suo dominio in quanto  $\forall x < 0, \max(0, x) = 0$
- ▶ Presenta un punto di discontinuità in  $x = 0$
- ▶ La sua derivata è pari a  $r'(x) = \mathbb{1}(x \geq 0)$



# Reti Neurali

## Rectifier Linear Unit



*Rappresentazione grafica ReLU*

# Reti Neurali

## Sigmoide

### Definizione

La *Sigmoide*  $\sigma : \mathbb{R} \rightarrow [0, 1]$  è definita come  $\sigma(x) = \frac{1}{(1+e^{-x})}$

# Reti Neurali

## Sigmoide

### Definizione

La *Sigmoide*  $\sigma : \mathbb{R} \rightarrow [0, 1]$  è definita come  $\sigma(x) = \frac{1}{(1+e^{-x})}$

- Per elevati valori negativi di input la sigmoide restituisce 0: il neurone non spara affatto

# Reti Neurali

## Sigmoide

### Definizione

La *Sigmoide*  $\sigma : \mathbb{R} \rightarrow [0, 1]$  è definita come  $\sigma(x) = \frac{1}{(1+e^{-x})}$

- Per elevati valori negativi di input la sigmoide restituisce 0: il neurone non spara affatto
- Per elevati valori positivi la sigmoide restituisce 1: il neurone satura e spara con frequenza di sparo pari a 1

# Reti Neurali

## Sigmoide

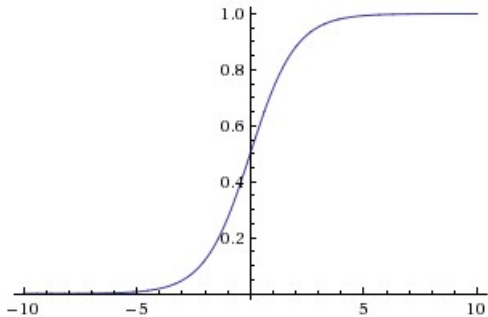
### Definizione

La *Sigmoide*  $\sigma : \mathbb{R} \rightarrow [0, 1]$  è definita come  $\sigma(x) = \frac{1}{(1+e^{-x})}$

- ▶ Per elevati valori negativi di input la sigmoide restituisce 0: il neurone non spara affatto
- ▶ Per elevati valori positivi la sigmoide restituisce 1: il neurone satura e spara con frequenza di sparo pari a 1
- ▶ La sua derivata è uguale a  $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

# Reti Neurali

## Sigmoide



*Rappresentazione grafica Sigmoide*

# Reti Neurali

## Tangente Iperbolica

### Definizione

La *Tangente Iperbolica*  $\tanh : \mathbb{R} \rightarrow [-1, 1]$  è definita come  $\tanh(x) = 2\sigma(2x) - 1$

# Reti Neurali

## Tangente Iperbolica

### Definizione

La *Tangente Iperbolica*  $\tanh : \mathbb{R} \rightarrow [-1, 1]$  è definita come  $\tanh(x) = 2\sigma(2x) - 1$

- La tangente iperbolica è una sigmoide scalata



# Reti Neurali

## Tangente Iperbolica

### Definizione

La *Tangente Iperbolica*  $\tanh : \mathbb{R} \rightarrow [-1, 1]$  è definita come  $\tanh(x) = 2\sigma(2x) - 1$

- ▶ La tangente iperbolica è una sigmoide scalata
- ▶ A differenza della sigmoide passa dall'origine per  $x = 0$

# Reti Neurali

## Tangente Iperbolica

### Definizione

La *Tangente Iperbolica*  $\tanh : \mathbb{R} \rightarrow [-1, 1]$  è definita come  $\tanh(x) = 2\sigma(2x) - 1$

- ▶ La tangente iperbolica è una sigmoide scalata
- ▶ A differenza della sigmoide passa dall'origine per  $x = 0$
- ▶ La sua derivata è uguale a  $\tanh'(x) = 1 - \tanh^2(x)$

# Reti Neurali

## Tangente Iperbolica



*Rappresentazione grafica Tangente Iperbolica*

# Reti Neurali

## Softplus

### Definizione

La *Softplus*  $s : \mathbb{R} \rightarrow (0, +\infty)$  è definita come  $s(x) = \log(1 + e^x)$

# Reti Neurali

## Softplus

### Definizione

La *Softplus*  $s : \mathbb{R} \rightarrow (0, +\infty)$  è definita come  $s(x) = \log(1 + e^x)$

- La softplus è una buona approssimazione della ReLU

# Reti Neurali

## Softplus

### Definizione

La *Softplus*  $s : \mathbb{R} \rightarrow (0, +\infty)$  è definita come  $s(x) = \log(1 + e^x)$

- ▶ La softplus è una buona approssimazione della ReLU
- ▶ Viene solitamente usata per sostituire la ReLU perché non presenta punti di discontinuità

# Reti Neurali

## Softplus

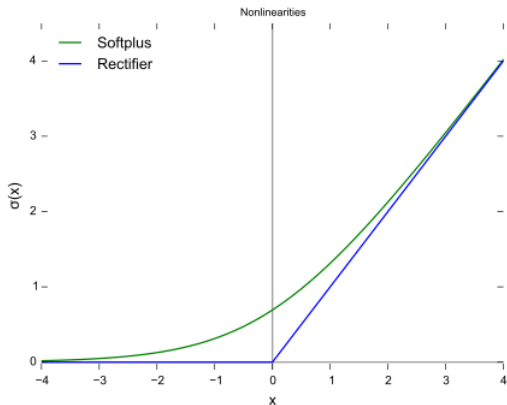
### Definizione

La *Softplus*  $s : \mathbb{R} \rightarrow (0, +\infty)$  è definita come  $s(x) = \log(1 + e^x)$

- ▶ La softplus è una buona approssimazione della ReLU
- ▶ Viene solitamente usata per sostituire la ReLU perché non presenta punti di discontinuità
- ▶ La sua derivata è uguale a  $s'(x) = \frac{1}{(1+e^{-x})} = \sigma(x)$

# Reti Neurali

## Softplus



*Confronto grafico tra ReLU e Softplus*



# Reti Neurali

## Rete Neurale

### Definizione

Una *Rete Neurale* è composta da un certo numero di neuroni organizzati in insiemi distinti chiamati *livelli* o *layer*

# Reti Neurali

## Rete Neurale

### Definizione

Una *Rete Neurale* è composta da un certo numero di neuroni organizzati in insiemi distinti chiamati *livelli* o *layer*

- I livelli sono connessi tra loro e sono posizionati uno di seguito all'altro in modo da formare una sequenza

# Reti Neurali

## Rete Neurale

### Definizione

Una *Rete Neurale* è composta da un certo numero di neuroni organizzati in insiemi distinti chiamati *livelli* o *layer*

- ▶ I livelli sono connessi tra loro e sono posizionati uno di seguito all'altro in modo da formare una sequenza
- ▶ I livelli intermedi prendono il nome di *hidden*

# Reti Neurali

## Rete Neurale

### Definizione

Una *Rete Neurale* è composta da un certo numero di neuroni organizzati in insiemi distinti chiamati *livelli* o *layer*

- ▶ I livelli sono connessi tra loro e sono posizionati uno di seguito all'altro in modo da formare una sequenza
- ▶ I livelli intermedi prendono il nome di *hidden*
- ▶ L'output dei neuroni di un livello diventano l'input dei neuroni del livello successivo

# Reti Neurali

## Rete Neurale

- ▶ Quando si effettua il conteggio dei livelli di una rete non si considera il livello di input

# Reti Neurali

## Rete Neurale

- ▶ Quando si effettua il conteggio dei livelli di una rete non si considera il livello di input
- ▶ Una rete a *singolo livello* non presenta livelli hidden

# Reti Neurali

## Rete Neurale

- ▶ Quando si effettua il conteggio dei livelli di una rete non si considera il livello di input
- ▶ Una rete a *singolo livello* non presenta livelli hidden
- ▶ Per determinare la grandezza di una rete ci si concentra sul numero di neuroni e sui relativi pesi ad essi associati

# Reti Neurali

## Livello Fully-Connected

### Definizione

Un livello è di tipo *Fully-Connected* quando i neuroni che lo compongono sono completamente connessi ai neuroni del livello successivo e non sono collegati tra loro internamente



# Reti Neurali

## Livello Fully-Connected

### Definizione

Un livello è di tipo *Fully-Connected* quando i neuroni che lo compongono sono completamente connessi ai neuroni del livello successivo e non sono collegati tra loro internamente

- I pesi dei neuroni di ciascun livello sono salvati all'interno di matrici

# Reti Neurali

## Livello Fully-Connected

### Definizione

Un livello è di tipo *Fully-Connected* quando i neuroni che lo compongono sono completamente connessi ai neuroni del livello successivo e non sono collegati tra loro internamente

- ▶ I pesi dei neuroni di ciascun livello sono salvati all'interno di matrici
- ▶ Le righe di una matrice identificano i neuroni del livello mentre le colonne contengono i pesi di ciascun neurone

# Reti Neurali

## Livello Fully-Connected

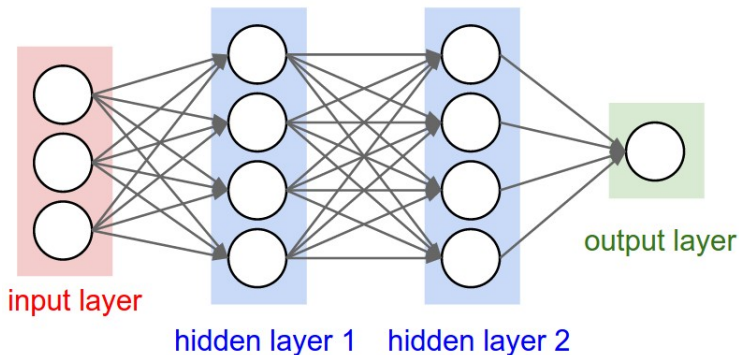
### Definizione

Un livello è di tipo *Fully-Connected* quando i neuroni che lo compongono sono completamente connessi ai neuroni del livello successivo e non sono collegati tra loro internamente

- ▶ I pesi dei neuroni di ciascun livello sono salvati all'interno di matrici
- ▶ Le righe di una matrice identificano i neuroni del livello mentre le colonne contengono i pesi di ciascun neurone
- ▶ La struttura a livelli di una rete neurale permette di sfruttare le potenzialità del calcolo matriciale

# Reti Neurali

## Livello Fully-Connected



*Una rete neurale a 3 livelli*

# Reti Neurali

## Funzionamento

Il processo di apprendimento di una rete neurale è suddiviso in quattro fasi distinte

# Reti Neurali

## Funzionamento

Il processo di apprendimento di una rete neurale è suddiviso in quattro fasi distinte

- *Inizializzazione dei pesi*

# Reti Neurali

## Funzionamento

Il processo di apprendimento di una rete neurale è suddiviso in quattro fasi distinte

- ▶ *Inizializzazione dei pesi*
- ▶ *Forward Propagation*

# Reti Neurali

## Funzionamento

Il processo di apprendimento di una rete neurale è suddiviso in quattro fasi distinte

- ▶ *Inizializzazione dei pesi*
- ▶ *Forward Propagation*
- ▶ *Calcolo della Funzione di Perdita*



# Reti Neurali

## Funzionamento

Il processo di apprendimento di una rete neurale è suddiviso in quattro fasi distinte

- ▶ *Inizializzazione dei pesi*
- ▶ *Forward Propagation*
- ▶ *Calcolo della Funzione di Perdita*
- ▶ *Back Propagation*

# Reti Neurali

## Inizializzazione dei pesi

- Al momento della nascita gli esseri umani non sono in grado di discriminare nessun tipo di oggetto a causa del mancato addestramento della loro rete neurale biologica

# Reti Neurali

## Inizializzazione dei pesi

- ▶ Al momento della nascita gli esseri umani non sono in grado di discriminare nessun tipo di oggetto a causa del mancato addestramento della loro rete neurale biologica
- ▶ Per riprodurre questo comportamento, all'inizio della fase di training, i pesi sinaptici  $w_i$  di ciascun livello vengono inizializzati in maniera casuale

# Reti Neurali

## Forward Propagation

### Definizione

La *Forward Propagation* è il meccanismo utilizzato da una rete neurale per associare ad un'immagine una determinata classe

# Reti Neurali

## Forward Propagation

### Definizione

La *Forward Propagation* è il meccanismo utilizzato da una rete neurale per associare ad un'immagine una determinata classe

- L'output dei neuroni del livello  $i$  viene moltiplicato per la matrice dei pesi del livello  $i + 1$  ottenendo il vettore  $v$

# Reti Neurali

## Forward Propagation

### Definizione

La *Forward Propagation* è il meccanismo utilizzato da una rete neurale per associare ad un'immagine una determinata classe

- ▶ L'output dei neuroni del livello  $i$  viene moltiplicato per la matrice dei pesi del livello  $i + 1$  ottenendo il vettore  $v$
- ▶ Al vettore  $v$  viene aggiunto il vettore dei bias del livello  $i + 1$

# Reti Neurali

## Forward Propagation

### Definizione

La *Forward Propagation* è il meccanismo utilizzato da una rete neurale per associare ad un'immagine una determinata classe

- ▶ L'output dei neuroni del livello  $i$  viene moltiplicato per la matrice dei pesi del livello  $i + 1$  ottenendo il vettore  $v$
- ▶ Al vettore  $v$  viene aggiunto il vettore dei bias del livello  $i + 1$
- ▶ L'output del livello  $i + 1$  si ottiene applicando la funzione di attivazione  $f$  ad ogni entry del vettore  $v$

# Reti Neurali

## Forward Propagation

### Definizione

La *Forward Propagation* è il meccanismo utilizzato da una rete neurale per associare ad un'immagine una determinata classe

- ▶ L'output dei neuroni del livello  $i$  viene moltiplicato per la matrice dei pesi del livello  $i + 1$  ottenendo il vettore  $v$
- ▶ Al vettore  $v$  viene aggiunto il vettore dei bias del livello  $i + 1$
- ▶ L'output del livello  $i + 1$  si ottiene applicando la funzione di attivazione  $f$  ad ogni entry del vettore  $v$
- ▶ Le operazioni precedenti sono svolte per tutti i livelli ad eccezione dell'ultimo



# Reti Neurali

## Calcolo della funzione di perdita

### Definizione

Una *funzione di perdita*  $L$  viene utilizzata per determinare l'errore di classificazione di una rete neurale

# Reti Neurali

## Calcolo della funzione di perdita

### Definizione

Una *funzione di perdita*  $L$  viene utilizzata per determinare l'errore di classificazione di una rete neurale

- La funzione di perdita più usata è la *Mean Squared Error (MSE)*  
$$L = \frac{1}{2} \sum (y - o)^2$$

# Reti Neurali

## Calcolo della funzione di perdita

### Definizione

Una *funzione di perdita*  $L$  viene utilizzata per determinare l'errore di classificazione di una rete neurale

- ▶ La funzione di perdita più usata è la *Mean Squared Error (MSE)*  
$$L = \frac{1}{2} \sum (y - o)^2$$
- ▶  $y$  identifica l'etichetta dell'esempio considerato mentre  $o$  l'output della rete

# Reti Neurali

## Calcolo della funzione di perdita

### Definizione

Una *funzione di perdita*  $L$  viene utilizzata per determinare l'errore di classificazione di una rete neurale

- ▶ La funzione di perdita più usata è la *Mean Squared Error (MSE)*  
$$L = \frac{1}{2} \sum (y - o)^2$$
- ▶  $y$  identifica l'etichetta dell'esempio considerato mentre  $o$  l'output della rete
- ▶ Minimizzando la funzione di perdita  $L$  si riduce l'errore di una rete neurale

# Reti Neurali

## Calcolo della funzione di perdita

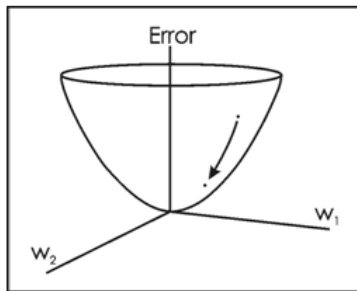
### Definizione

Una *funzione di perdita*  $L$  viene utilizzata per determinare l'errore di classificazione di una rete neurale

- ▶ La funzione di perdita più usata è la *Mean Squared Error (MSE)*  
$$L = \frac{1}{2} \sum (y - o)^2$$
- ▶  $y$  identifica l'etichetta dell'esempio considerato mentre  $o$  l'output della rete
- ▶ Minimizzando la funzione di perdita  $L$  si riduce l'errore di una rete neurale
- ▶ Calcolando la derivata di  $L$  in funzione dei pesi  $w_i$  si cerca di individuare il minimo globale della funzione di perdita

# Reti Neurali

## Funzione di perdita



*Mean Squared Error (MSE). I pesi  $w_1$  e  $w_2$  sono le variabili indipendenti. La funzione di perdita  $L$  è la variabile dipendente*

# Reti Neurali

## Back Propagation

### Definizione

La *Back Propagation* è il meccanismo utilizzato da una rete neurale per correggere gli errori di classificazione. Vengono individuati i pesi  $w_i$  che hanno influenzato maggiormente l'errore commesso e viene aggiornato il loro valore in modo da ridurre la funzione di perdita

# Reti Neurali

## Back Propagation

### Definizione

La *Back Propagation* è il meccanismo utilizzato da una rete neurale per correggere gli errori di classificazione. Vengono individuati i pesi  $w_i$  che hanno influenzato maggiormente l'errore commesso e viene aggiornato il loro valore in modo da ridurre la funzione di perdita

- Per calcolare la derivata della funzione  $L$  in funzione dei pesi  $w_i$  viene usata la *regola della catena* (*chain rule*)



# Reti Neurali

## Back Propagation

### Definizione

La *Back Propagation* è il meccanismo utilizzato da una rete neurale per correggere gli errori di classificazione. Vengono individuati i pesi  $w_i$  che hanno influenzato maggiormente l'errore commesso e viene aggiornato il loro valore in modo da ridurre la funzione di perdita

- ▶ Per calcolare la derivata della funzione  $L$  in funzione dei pesi  $w_i$  viene usata la *regola della catena* (*chain rule*)
- ▶ Questa regola è usata per trovare la derivata di una funzione composta

# Reti Neurali

## Aggiornamento dei Pesi e Learning Rate

- Il nuovo valore del peso  $w_i$  è dato dalla regola di aggiornamento  $w_i = w_i - \eta \frac{\partial L}{\partial w_i} = w_i + \Delta w_i$  con  $\eta > 0$

# Reti Neurali

## Aggiornamento dei Pesi e Learning Rate

- ▶ Il nuovo valore del peso  $w_i$  è dato dalla regola di aggiornamento  $w_i = w_i - \eta \frac{\partial L}{\partial w_i} = w_i + \Delta w_i$  con  $\eta > 0$
- ▶ Il *learning rate*  $\eta$  è un parametro usato per controllare la velocità di aggiornamento dei pesi

# Reti Neurali

## Aggiornamento dei Pesi e Learning Rate

- ▶ Il nuovo valore del peso  $w_i$  è dato dalla regola di aggiornamento  $w_i = w_i - \eta \frac{\partial L}{\partial w_i} = w_i + \Delta w_i$  con  $\eta > 0$
- ▶ Il *learning rate*  $\eta$  è un parametro usato per controllare la velocità di aggiornamento dei pesi
- ▶ Un learning rate alto comporta aggiornamenti rapidi, un tempo di esecuzione più basso, ma una maggiore probabilità di finire in un minimo locale

# Reti Neurali

## Aggiornamento dei Pesi e Learning Rate

- ▶ Il nuovo valore del peso  $w_i$  è dato dalla regola di aggiornamento  $w_i = w_i - \eta \frac{\partial L}{\partial w_i} = w_i + \Delta w_i$  con  $\eta > 0$
- ▶ Il *learning rate*  $\eta$  è un parametro usato per controllare la velocità di aggiornamento dei pesi
- ▶ Un learning rate alto comporta aggiornamenti rapidi, un tempo di esecuzione più basso, ma una maggiore probabilità di finire in un minimo locale
- ▶ Un learning rate basso diminuisce la probabilità di finire in un minimo locale, ma allunga notevolmente i tempi di esecuzione

# Reti Neurali

## Esempio Back Propagation



$$x \in \mathbb{R}^{n,1} \quad w^h \in \mathbb{R}^{n,m}$$

$$h \in \mathbb{R}^{m,1} \quad w^o \in \mathbb{R}^{1,m}$$

$$z_j^h = \sum_{i=0}^n w_{ij}^h x_i$$

$$h_j = f(z_j^h)$$

$$z^o = \sum_{j=0}^m w_j^o h_j$$

$$o = f(z^o)$$

# Reti Neurali

## Esempio Back Propagation

- Derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

# Reti Neurali

## Esempio Back Propagation

- Derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

- $\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$



# Reti Neurali

## Esempio Back Propagation

- Derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

- $\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$

- $\frac{\partial o}{\partial z^o} = f'(z^o)$

# Reti Neurali

## Esempio Back Propagation

- Derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial w_j}$$

- $\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$

- $\frac{\partial o}{\partial z^o} = f'(z^o)$

- $\frac{\partial z^o}{\partial w_j} = h_j$

# Reti Neurali

## Esempio Back Propagation

- Risultato della derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = -(y - o) \cdot f'(z^o) \cdot h_j = -\delta_j^o h_j$$

# Reti Neurali

## Esempio Back Propagation

- Risultato della derivata di  $L$  in funzione del peso  $w_j^o$

$$\frac{\partial L}{\partial w_j^o} = -(y - o) \cdot f'(z^o) \cdot h_j = -\delta_j^o h_j$$

- Aggiornamento del peso  $w_j^o$

$$\Delta w_j^o = \eta \delta_j^o h_j$$

# Reti Neurali

## Esempio Back Propagation



$$\frac{\partial L}{\partial w_{ij}^h} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial z^o} \cdot \frac{\partial z^o}{\partial h_j} \cdot \frac{\partial h_j}{\partial z_j^h} \cdot \frac{\partial z_j^h}{\partial w_{ij}^h}$$

# Reti Neurali

## Esempio Back Propagation

$$\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$$

# Reti Neurali

## Esempio Back Propagation

$$\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$$

$$\frac{\partial o}{\partial z^o} = f'(z^o)$$

# Reti Neurali

## Esempio Back Propagation

$$\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$$

$$\frac{\partial o}{\partial z^o} = f'(z^o)$$

$$\frac{\partial z^o}{\partial h_j} = w_j^o$$



# Reti Neurali

## Esempio Back Propagation

$$\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$$

$$\frac{\partial o}{\partial z^o} = f'(z^o)$$

$$\frac{\partial z^o}{\partial h_j} = w_j^o$$

$$\frac{\partial h_j}{\partial z_j^h} = f'(z_j^h)$$

# Reti Neurali

## Esempio Back Propagation

$$\frac{\partial L}{\partial o} = \frac{\partial}{\partial o} \left[ \frac{1}{2} (y - o)^2 \right] = -(y - o)$$

$$\frac{\partial o}{\partial z^o} = f'(z^o)$$

$$\frac{\partial z^o}{\partial h_j} = w_j^o$$

$$\frac{\partial h_j}{\partial z_j^h} = f'(z_j^h)$$

$$\frac{\partial z_j^h}{\partial w_{ij}^h} = x_i$$

# Reti Neurali

## Esempio Back Propagation

- Risultato della derivata di  $L$  in funzione del peso  $w_{ij}^h$

$$\frac{\partial L}{\partial w_{ij}^h} = -(y - o) \cdot f'(z^o) \cdot w_j^o \cdot f'(z_j^h) \cdot x_i = -\delta_j^h x_i$$

# Reti Neurali

## Esempio Back Propagation

- Risultato della derivata di  $L$  in funzione del peso  $w_{ij}^h$

$$\frac{\partial L}{\partial w_{ij}^h} = -(y - o) \cdot f'(z^o) \cdot w_j^o \cdot f'(z_j^h) \cdot x_i = -\delta_j^h x_i$$

- Aggiornamento del peso  $w_{ij}^h$

$$\Delta w_{ij}^h = \eta \delta_j^h x_i$$

# Reti Neurali

## Rete Neurale Convoluzionale

### Definizione

Una *Rete Neurale Convoluzionale* è una variante di una rete neurale classica. Permette la condivisione dei pesi sinaptici tra i neuroni di un livello e consente di discriminare le varie feature che compongono un'immagine

# Reti Neurali

## Rete Neurale Convoluzionale

### Definizione

Una *Rete Neurale Convoluzionale* è una variante di una rete neurale classica. Permette la condivisione dei pesi sinaptici tra i neuroni di un livello e consente di discriminare le varie feature che compongono un'immagine

- ▶ Viene definito un nuovo tipo di livello: il *Livello Convoluzionale*

# Reti Neurali

## Rete Neurale Convoluzionale

### Definizione

Una *Rete Neurale Convoluzionale* è una variante di una rete neurale classica. Permette la condivisione dei pesi sinaptici tra i neuroni di un livello e consente di discriminare le varie feature che compongono un'immagine

- ▶ Viene definito un nuovo tipo di livello: il *Livello Convoluzionale*
- ▶ Un livello convoluzionale è formato da diversi *filtri*

# Reti Neurali

## Rete Neurale Convoluzionale

### Definizione

Una *Rete Neurale Convoluzionale* è una variante di una rete neurale classica. Permette la condivisione dei pesi sinaptici tra i neuroni di un livello e consente di discriminare le varie feature che compongono un'immagine

- ▶ Viene definito un nuovo tipo di livello: il *Livello Convoluzionale*
- ▶ Un livello convoluzionale è formato da diversi *filtri*
- ▶ La *profondità (depth)* di un livello convoluzionale è data dal numero di filtri che lo compongono



# Reti Neurali

## Filtri e Livelli Convoluzionali

- I filtri sono le matrici contenenti i pesi sinaptici del livello convoluzionale

# Reti Neurali

## Filtri e Livelli Convoluzionali

- ▶ I filtri sono le matrici contenenti i pesi sinaptici del livello convoluzionale
- ▶ Ogni filtro ricerca all'interno delle immagini della rete una o più *feature*: linee, curve, pattern

# Reti Neurali

## Filtri e Livelli Convoluzionali

- ▶ I filtri sono le matrici contenenti i pesi sinaptici del livello convoluzionale
- ▶ Ogni filtro ricerca all'interno delle immagini della rete una o più *feature*: linee, curve, pattern
- ▶ Per apprendere nel miglior modo possibile il contenuto semantico di un'immagine, la rete deve saper ricercare feature sempre più complesse

# Reti Neurali

## Filtri e Livelli Convoluzionali

- ▶ I filtri sono le matrici contenenti i pesi sinaptici del livello convoluzionale
- ▶ Ogni filtro ricerca all'interno delle immagini della rete una o più *feature*: linee, curve, pattern
- ▶ Per apprendere nel miglior modo possibile il contenuto semantico di un'immagine, la rete deve saper ricercare feature sempre più complesse
- ▶ Mettendo in sequenza più livelli convoluzionali si possono ottenere feature complesse

# Reti Neurali

## Filtri e Livelli Convoluzionali

- ▶ I filtri sono le matrici contenenti i pesi sinaptici del livello convoluzionale
- ▶ Ogni filtro ricerca all'interno delle immagini della rete una o più *feature*: linee, curve, pattern
- ▶ Per apprendere nel miglior modo possibile il contenuto semantico di un'immagine, la rete deve saper ricercare feature sempre più complesse
- ▶ Mettendo in sequenza più livelli convoluzionali si possono ottenere feature complesse
- ▶ L'output di un generico livello convoluzionale  $i$  diventa l'input del successivo livello  $i + 1$ . Le feature prodotte da  $i$  sono meno complesse di quelle ottenute da  $i + 1$

# Reti Neurali

## Funzionamento

- ▶ I pesi dei filtri di un livello convoluzionale sono inizializzati in maniera casuale

# Reti Neurali

## Funzionamento

- ▶ I pesi dei filtri di un livello convoluzionale sono inizializzati in maniera casuale
- ▶ Vengono utilizzate le stesse funzioni di attivazione e le stesse funzioni di perdita dei livelli fully-connected

# Reti Neurali

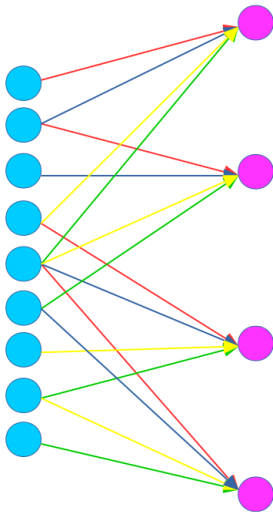
## Funzionamento

- ▶ I pesi dei filtri di un livello convoluzionale sono inizializzati in maniera casuale
- ▶ Vengono utilizzate le stesse funzioni di attivazione e le stesse funzioni di perdita dei livelli fully-connected
- ▶ La forward e la back propagation sono le uniche fasi definite diversamente



# Reti Neurali

## Forward Propagation



# Reti Neurali

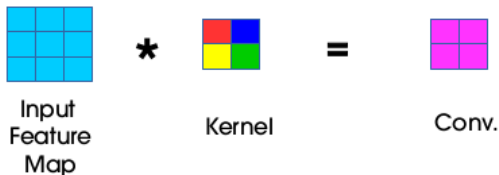
## Forward Propagation



- Le matrici di input e di output di un livello convoluzionale prendono il nome di *feature map*

# Reti Neurali

## Forward Propagation



- ▶ Le matrici di input e di output di un livello convoluzionale prendono il nome di *feature map*
- ▶ I filtri sono meglio conosciuti con il nome di *kernel*

# Reti Neurali

## Forward Propagation

- All'inizio della forward propagation, il kernel viene sovrapposto alla parte superiore sinistra della feature map di input

# Reti Neurali

## Forward Propagation

- ▶ All'inizio della forward propagation, il kernel viene sovrapposto alla parte superiore sinistra della feature map di input
- ▶ Viene eseguita la *convoluzione* tra le due sottomatrici ed il risultato ottenuto viene salvato nella feature map di output

# Reti Neurali

## Forward Propagation

- ▶ All'inizio della forward propagation, il kernel viene sovrapposto alla parte superiore sinistra della feature map di input
- ▶ Viene eseguita la *convoluzione* tra le due sottomatrici ed il risultato ottenuto viene salvato nella feature map di output
- ▶ Il kernel viene spostato di una posizione verso destra e viene rieseguita nuovamente la convoluzione

# Reti Neurali

## Forward Propagation

- ▶ All'inizio della forward propagation, il kernel viene sovrapposto alla parte superiore sinistra della feature map di input
- ▶ Viene eseguita la *convoluzione* tra le due sottomatrici ed il risultato ottenuto viene salvato nella feature map di output
- ▶ Il kernel viene spostato di una posizione verso destra e viene rieseguita nuovamente la convoluzione
- ▶ Terminata la riga, il kernel viene posizionato nuovamente nella parte sinistra della feature map di input, ma una riga più in basso

# Reti Neurali

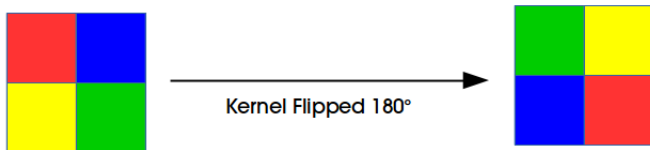
## Forward Propagation

- ▶ All'inizio della forward propagation, il kernel viene sovrapposto alla parte superiore sinistra della feature map di input
- ▶ Viene eseguita la *convoluzione* tra le due sottomatrici ed il risultato ottenuto viene salvato nella feature map di output
- ▶ Il kernel viene spostato di una posizione verso destra e viene rieseguita nuovamente la convoluzione
- ▶ Terminata la riga, il kernel viene posizionato nuovamente nella parte sinistra della feature map di input, ma una riga più in basso
- ▶ Gli ultimi due passaggi vengono ripetuti fino a quando non è stata riempita completamente tutta la feature map di output



# Reti Neurali

## Forward Propagation



*Il kernel viene ruotato di 180° per poter eseguire la convoluzione*

# Reti Neurali

## Forward Propagation



*Forward Propagation di un livello convoluzionale*

# Reti Neurali

## Considerazioni Forward Propagation

- Al termine della forward propagation, la funzione di attivazione  $f$  viene applicata ad ogni elemento della feature map di output

# Reti Neurali

## Considerazioni Forward Propagation

- ▶ Al termine della forward propagation, la funzione di attivazione  $f$  viene applicata ad ogni elemento della feature map di output
- ▶ Un kernel è una matrice quadrata di dimensione  $K \times K$

# Reti Neurali

## Considerazioni Forward Propagation

- ▶ Al termine della forward propagation, la funzione di attivazione  $f$  viene applicata ad ogni elemento della feature map di output
- ▶ Un kernel è una matrice quadrata di dimensione  $K \times K$
- ▶ La feature map di input ha dimensione  $W \times H$  con  $W = H$

# Reti Neurali

## Considerazioni Forward Propagation

- ▶ Al termine della forward propagation, la funzione di attivazione  $f$  viene applicata ad ogni elemento della feature map di output
- ▶ Un kernel è una matrice quadrata di dimensione  $K \times K$
- ▶ La feature map di input ha dimensione  $W \times H$  con  $W = H$
- ▶ La feature map di output è una matrice quadrata di dimensione  $O \times O$  con  $O = (W - K) + 1$

# Reti Neurali

## Back Propagation

### Definizione

La Back Propagation di una rete neurale convoluzionale ha come obiettivo l'aggiornamento dei pesi contenuti nei kernel di un livello. Per ciascuno dei pesi di un kernel viene calcolata la derivata parziale  $\frac{\partial L}{\partial w_{m',n'}^l}$  che rappresenta l'influenza del peso  $w_{m',n'}^l$  sulla funzione di perdita  $L$

# Reti Neurali

## Back Propagation

### Definizione

La Back Propagation di una rete neurale convoluzionale ha come obiettivo l'aggiornamento dei pesi contenuti nei kernel di un livello. Per ciascuno dei pesi di un kernel viene calcolata la derivata parziale  $\frac{\partial L}{\partial w_{m',n'}^l}$  che rappresenta l'influenza del peso  $w_{m',n'}^l$  sulla funzione di perdita  $L$

- La Back Propagation viene suddivisa in due fasi distinte



# Reti Neurali

## Back Propagation

### Definizione

La Back Propagation di una rete neurale convoluzionale ha come obiettivo l'aggiornamento dei pesi contenuti nei kernel di un livello. Per ciascuno dei pesi di un kernel viene calcolata la derivata parziale  $\frac{\partial L}{\partial w_{m',n'}^l}$  che rappresenta l'influenza del peso  $w_{m',n'}^l$  sulla funzione di perdita  $L$

- La Back Propagation viene suddivisa in due fasi distinte
  - Il calcolo della matrice degli errori  $\delta$

# Reti Neurali

## Back Propagation

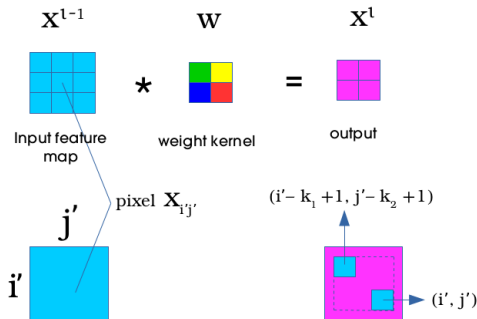
### Definizione

La Back Propagation di una rete neurale convoluzionale ha come obiettivo l'aggiornamento dei pesi contenuti nei kernel di un livello. Per ciascuno dei pesi di un kernel viene calcolata la derivata parziale  $\frac{\partial L}{\partial w_{m',n'}^l}$  che rappresenta l'influenza del peso  $w_{m',n'}^l$  sulla funzione di perdita  $L$

- La Back Propagation viene suddivisa in due fasi distinte
  - Il calcolo della matrice degli errori  $\delta$
  - L'aggiornamento dei pesi del kernel

# Reti Neurali

## Calcolo matrice dei $\delta$



*Le linee tratteggiate presenti nella feature map di output individuano la regione dei pixel influenzati dal pixel  $x_{i',j'}$ .  $k_1$  e  $k_2$  definiscono la grandezza della regione considerata*

# Reti Neurali

## Calcolo matrice dei $\delta$

- L'influenza del pixel  $x_{i',j'}$  sulla funzione di perdita  $L$  è data da

$$\delta_{i',j'}^l = \frac{\partial L}{\partial x_{i',j'}^l}$$

# Reti Neurali

## Calcolo matrice dei $\delta$

- L'influenza del pixel  $x_{i',j'}$  sulla funzione di perdita  $L$  è data da

$$\delta_{i',j'}^l = \frac{\partial L}{\partial x_{i',j'}^l}$$

- Applicando la regola della catena si ottiene

$$\begin{aligned}\frac{\partial L}{\partial x_{i',j'}^l} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial L}{\partial x_{i'-m,j'-n}^{l+1}} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^l} \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} \frac{\partial x_{i'-m,j'-n}^{l+1}}{\partial x_{i',j'}^l}\end{aligned}$$

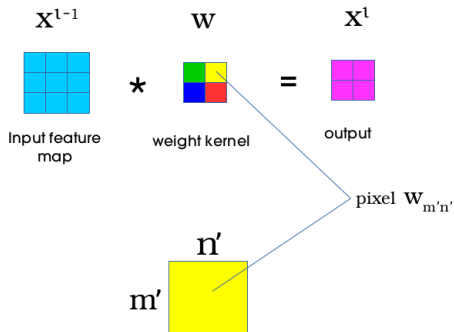
# Reti Neurali

## Calcolo matrice dei $\delta$

$$\begin{aligned}\frac{\partial L}{\partial x_{i',j'}^l} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'-m,j'-n}^{l+1} w_{m,n}^{l+1} f' \left( x_{i',j'}^l \right) \\ &= \text{rot}_{180^\circ} \left\{ \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta_{i'+m,j'+n}^{l+1} w_{m,n}^{l+1} \right\} f' \left( x_{i',j'}^l \right) \\ &= \delta_{i',j'}^{l+1} * \text{rot}_{180^\circ} \left\{ w_{m,n}^{l+1} \right\} f' \left( x_{i',j'}^l \right)\end{aligned}$$

# Reti Neurali

## Aggiornamento dei pesi



*Durante la fase di forward propagation, il peso  $w_{m',n'}$  ha contribuito a calcolare tutti i valori che costituiscono la feature map di output*

# Reti Neurali

## Aggiornamento dei pesi

- Il calcolo di  $\frac{\partial L}{\partial w_{m',n'}^l}$  usando la regola della catena è dato da

$$\begin{aligned}\frac{\partial L}{\partial w_{m',n'}^l} &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \frac{\partial L}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial w_{m',n'}^l} \\ &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \delta_{i,j}^l \cdot o_{i+m',j+n'}^{l-1} \\ &= \text{rot}_{180^\circ} \{ \delta_{i,j}^l \} * o_{m',n'}^{l-1}\end{aligned}$$



# Reti Neurali

## Aggiornamento dei pesi

- Il calcolo di  $\frac{\partial L}{\partial w_{m',n'}^l}$  usando la regola della catena è dato da

$$\begin{aligned}\frac{\partial L}{\partial w_{m',n'}^l} &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \frac{\partial L}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial w_{m',n'}^l} \\ &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \delta_{i,j}^l \cdot o_{i+m',j+n'}^{l-1} \\ &= \text{rot}_{180^\circ} \{ \delta_{i,j}^l \} * o_{m',n'}^{l-1}\end{aligned}$$

- $x_{i,j}^l = \sum_m \sum_n w_{m,n}^l o_{i+m,j+n}^{l-1} + b^l$

# Reti Neurali

## Aggiornamento dei pesi

- Il calcolo di  $\frac{\partial L}{\partial w_{m',n'}^l}$  usando la regola della catena è dato da

$$\begin{aligned}\frac{\partial L}{\partial w_{m',n'}^l} &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \frac{\partial L}{\partial x_{i,j}^l} \frac{\partial x_{i,j}^l}{\partial w_{m',n'}^l} \\ &= \sum_{i=0}^{W-K} \sum_{j=0}^{W-K} \delta_{i,j}^l \cdot o_{i+m',j+n'}^{l-1} \\ &= \text{rot}_{180^\circ} \{ \delta_{i,j}^l \} * o_{m',n'}^{l-1}\end{aligned}$$

- $x_{i,j}^l = \sum_m \sum_n w_{m,n}^l o_{i+m,j+n}^{l-1} + b^l$
- $o_{m',n'}^{l-1} = f(x_{i',j'}^{l-1})$

# Reti Neurali

## Aggiornamento dei pesi

- Il risultato della convoluzione tra  $\delta_{i,j}^l$  e  $o_{m',n'}^{l-1}$  individua il nuovo valore del peso  $w_{m',n'}^l$

# Reti Neurali

## Aggiornamento dei pesi

- ▶ Il risultato della convoluzione tra  $\delta_{i,j}^l$  e  $o_{m',n'}^{l-1}$  individua il nuovo valore del peso  $w_{m',n'}^l$
- ▶ La convoluzione è svolta per ciascuno dei pesi che costituiscono un kernel

# Reti Neurali

## Back Propagation



*La matrice degli errori  $\delta$  deve essere ruotata di 180° per poter eseguire la convoluzione*

# Reti Neurali

## Esemplio Back Propagation



# Reti Neurali

## Esempio Back Propagation



*Il kernel aggiornato viene ricavato dalla convoluzione tra la matrice degli errori  $\delta$  e la feature map di input*

# Implementazione della Rete



# Implementazione della Rete

## Obiettivo

- Sfruttando l'architettura per il calcolo parallelo *CUDA*, si vuole costruire una rete neurale convoluzionale che permetta di riconoscere cifre numeriche scritte a mano

# Implementazione della Rete

## Obiettivo

- ▶ Sfruttando l'architettura per il calcolo parallelo *CUDA*, si vuole costruire una rete neurale convoluzionale che permetta di riconoscere cifre numeriche scritte a mano
- ▶ Le cifre da riconoscere sono salvate come immagini in scala di grigio a 8 bit. Un pixel può assumere solo valori compresi nell'intervallo  $[0, 255]$

# Implementazione della Rete

## Obiettivo

- ▶ Sfruttando l'architettura per il calcolo parallelo *CUDA*, si vuole costruire una rete neurale convoluzionale che permetta di riconoscere cifre numeriche scritte a mano
- ▶ Le cifre da riconoscere sono salvate come immagini in scala di grigio a 8 bit. Un pixel può assumere solo valori compresi nell'intervallo  $[0, 255]$
- ▶ L'output della rete è dato dalle 10 cifre numeriche che si vogliono riconoscere

# Implementazione della Rete

## Obiettivo

- ▶ Sfruttando l'architettura per il calcolo parallelo *CUDA*, si vuole costruire una rete neurale convoluzionale che permetta di riconoscere cifre numeriche scritte a mano
- ▶ Le cifre da riconoscere sono salvate come immagini in scala di grigio a 8 bit. Un pixel può assumere solo valori compresi nell'intervallo  $[0, 255]$
- ▶ L'output della rete è dato dalle 10 cifre numeriche che si vogliono riconoscere
- ▶ La rete riceve in input un'immagine e le associa la cifra numerica corrispondente

# Implementazione della Rete

## Dati

- La dimensione delle immagini che costituiscono gli esempi del training e del test set hanno una dimensione di  $28 \times 28$

# Implementazione della Rete

## Dati

- ▶ La dimensione delle immagini che costituiscono gli esempi del training e del test set hanno una dimensione di  $28 \times 28$
- ▶ Le etichette sono rappresentate da numeri interi positivi

# Implementazione della Rete

## Dati

- ▶ La dimensione delle immagini che costituiscono gli esempi del training e del test set hanno una dimensione di  $28 \times 28$
- ▶ Le etichette sono rappresentate da numeri interi positivi
- ▶ Il training ed il test set provengono dal database *MNIST* e contengono rispettivamente 60000 esempi di train e 10000 di test

# Implementazione della Rete

## Rete CUDA

- Per potere sfruttare le potenzialità del paradigma ad oggetti, il codice host della rete CUDA è scritto in C++



# Implementazione della Rete

## Rete CUDA

- ▶ Per potere sfruttare le potenzialità del paradigma ad oggetti, il codice host della rete CUDA è scritto in C++
- ▶ I livelli implementati devono rispettare un'interfaccia comune e vengono modellati come classi

# Implementazione della Rete

## Rete CUDA

- ▶ Per potere sfruttare le potenzialità del paradigma ad oggetti, il codice host della rete CUDA è scritto in C++
- ▶ I livelli implementati devono rispettare un'interfaccia comune e vengono modellati come classi
- ▶ Le funzioni abibite alla fase di train, di test e al calcolo dell'accuratezza sono contenute in unica classe chiamata *Network*

# Implementazione della Rete

## Rete CUDA

- Quando viene definito un livello, l'utente non deve inserire anche la dimensione di output del livello precedente, ma solo quella del livello che sta definendo

# Implementazione della Rete

## Rete CUDA

- ▶ Quando viene definito un livello, l'utente non deve inserire anche la dimensione di output del livello precedente, ma solo quella del livello che sta definendo
- ▶ I pesi iniziali della rete possono essere generati in maniera casuale

# Implementazione della Rete

## Rete CUDA

- ▶ Quando viene definito un livello, l'utente non deve inserire anche la dimensione di output del livello precedente, ma solo quella del livello che sta definendo
- ▶ I pesi iniziali della rete possono essere generati in maniera casuale
- ▶ Per migliorare l'accuratezza della rete, viene utilizzato il meccanismo delle *epoche* che consiste nel ripetere più volte la forward e la back propagation sulla stessa immagine

# Implementazione della Rete

## Rete CUDA

- ▶ Quando viene definito un livello, l'utente non deve inserire anche la dimensione di output del livello precedente, ma solo quella del livello che sta definendo
- ▶ I pesi iniziali della rete possono essere generati in maniera casuale
- ▶ Per migliorare l'accuratezza della rete, viene utilizzato il meccanismo delle *epoche* che consiste nel ripetere più volte la forward e la back propagation sulla stessa immagine
- ▶ Le funzioni di attivazione implementate dalla rete sono la *Sigmoide*, la *Tangente Iperbolica* e la *SoftPlus*

# Implementazione della Rete

## Caratteristiche tecniche

- Per implementare i livelli della rete sono stati usati gli *unique pointer* perché permettono all'utente di non dover specificare il numero di livelli che vuole usare e non dover preoccuparsi di deallocarli al termine dell'esecuzione

# Implementazione della Rete

## Caratteristiche tecniche

- ▶ Per implementare i livelli della rete sono stati usati gli *unique pointer* perché permettono all'utente di non dover specificare il numero di livelli che vuole usare e non dover preoccuparsi di deallocarli al termine dell'esecuzione
- ▶ All'inizio della fase di train, l'intero training set viene caricato nella GRAM utilizzando il meccanismo della *pinned memory*



# Implementazione della Rete

## Caratteristiche tecniche

- ▶ Per implementare i livelli della rete sono stati usati gli *unique pointer* perché permettono all'utente di non dover specificare il numero di livelli che vuole usare e non dover preoccuparsi di deallocarli al termine dell'esecuzione
- ▶ All'inizio della fase di train, l'intero training set viene caricato nella GRAM utilizzando il meccanismo della *pinned memory*
- ▶ Prima della fase di test, il training set viene deallocato e al suo posto viene caricato il test set sempre con il meccanismo *pinned memory*

# Implementazione della Rete

## Caratteristiche tecniche

- ▶ Per implementare i livelli della rete sono stati usati gli *unique pointer* perché permettono all'utente di non dover specificare il numero di livelli che vuole usare e non dover preoccuparsi di deallocarli al termine dell'esecuzione
- ▶ All'inizio della fase di train, l'intero training set viene caricato nella GRAM utilizzando il meccanismo della *pinned memory*
- ▶ Prima della fase di test, il training set viene deallocato e al suo posto viene caricato il test set sempre con il meccanismo *pinned memory*
- ▶ Le strutture dati usate dai singoli livelli vengono deallocate automaticamente al termine dell'esecuzione del processo

# Implementazione della Rete

## Librerie

- Per eseguire le varie operazioni algebriche è stata utilizzata la libreria CUDA chiamata *cuBLAS*

# Implementazione della Rete

## Librerie

- ▶ Per eseguire le varie operazioni algebriche è stata utilizzata la libreria CUDA chiamata *cuBLAS*
- ▶ Durante la costruzione della rete, più di preciso quando si eseguivano i test intermedi, si è osservato che la funzione adibita al calcolo del prodotto tra matrici *cublasGemm* risulta più lenta di *cublasGemm* che effettua il prodotto tra matrici e vettori

# Implementazione della Rete

## Librerie

- ▶ Per eseguire le varie operazioni algebriche è stata utilizzata la libreria CUDA chiamata *cuBLAS*
- ▶ Durante la costruzione della rete, più di preciso quando si eseguivano i test intermedi, si è osservato che la funzione adibita al calcolo del prodotto tra matrici *cublasGemm* risulta più lenta di *cublasGemv* che effettua il prodotto tra matrici e vettori
- ▶ A fronte di questi risultati, si è scelto di usare la funzione *cublasGemv* per eseguire i prodotti matrice-vettore

# Implementazione della Rete

## Librerie

- ▶ Per eseguire le varie operazioni algebriche è stata utilizzata la libreria CUDA chiamata *cuBLAS*
- ▶ Durante la costruzione della rete, più di preciso quando si eseguivano i test intermedi, si è osservato che la funzione adibita al calcolo del prodotto tra matrici *cublasGemm* risulta più lenta di *cublasGemv* che effettua il prodotto tra matrici e vettori
- ▶ A fronte di questi risultati, si è scelto di usare la funzione *cublasGemv* per eseguire i prodotti matrice-vettore
- ▶ I pesi iniziali della rete vengono generati in maniera casuale utilizzando l'algoritmo *xorWow* contenuto nella libreria di CUDA *cuRand*

# Implementazione della Rete

## Stream

- Se la funzione *cublasGemm* viene utilizzata insieme agli *streams*, il tempo di calcolo per eseguire i prodotti tra matrici dovrebbe ridursi notevolmente

# Implementazione della Rete

## Stream

- ▶ Se la funzione *cublasGemm* viene utilizzata insieme agli *streams*, il tempo di calcolo per eseguire i prodotti tra matrici dovrebbe ridursi notevolmente
- ▶ Nel caso della rete CUDA, eseguire una *cublasGemm* per stream su matrici relativamente piccole, sia per livelli convoluzionali che fully-connected, non ha portato a nessun miglioramento del tempo di calcolo



# Implementazione della Rete

## Stream

- ▶ Se la funzione *cublasGemm* viene utilizzata insieme agli *streams*, il tempo di calcolo per eseguire i prodotti tra matrici dovrebbe ridursi notevolmente
- ▶ Nel caso della rete CUDA, eseguire una *cublasGemm* per stream su matrici relativamente piccole, sia per livelli convoluzionali che fully-connected, non ha portato a nessun miglioramento del tempo di calcolo
- ▶ Il tempo impiegato con e senza stream è lo stesso, questo è dovuto al fatto che l'overhead della *cublasGemm* su diversi stream è maggiore dell'effettivo tempo di calcolo del prodotto matriciale

# Implementazione della Rete

## Rete Sequenziale

- Le accuratezze e i tempi di esecuzione della rete CUDA sono stati confrontati con quelli di una rete sequenziale chiamata *EduCNN*

# Implementazione della Rete

## Rete Sequenziale

- ▶ Le accuratezze e i tempi di esecuzione della rete CUDA sono stati confrontati con quelli di una rete sequenziale chiamata *EduCNN*
- ▶ La scelta della rete sequenziale è ricaduta sulla EduCNN perché consente di trovare buone accuratezze in un tempo adeguato

# Implementazione della Rete

## Rete Sequenziale

- ▶ Le accuratezze e i tempi di esecuzione della rete CUDA sono stati confrontati con quelli di una rete sequenziale chiamata *EduCNN*
- ▶ La scelta della rete sequenziale è ricaduta sulla EduCNN perché consente di trovare buone accuratezze in un tempo adeguato
- ▶ Ammette sia livelli di tipo fully-connected che convoluzionali

# Implementazione della Rete

## Rete Sequenziale

- ▶ Le accuratezze e i tempi di esecuzione della rete CUDA sono stati confrontati con quelli di una rete sequenziale chiamata *EduCNN*
- ▶ La scelta della rete sequenziale è ricaduta sulla EduCNN perché consente di trovare buone accuratezze in un tempo adeguato
- ▶ Ammette sia livelli di tipo fully-connected che convoluzionali
- ▶ L'unica funzione di attivazione implementata è la sigmoide

# Implementazione della Rete

## Rete Sequenziale

- ▶ Le accuratezze e i tempi di esecuzione della rete CUDA sono stati confrontati con quelli di una rete sequenziale chiamata *EduCNN*
- ▶ La scelta della rete sequenziale è ricaduta sulla EduCNN perché consente di trovare buone accuratezze in un tempo adeguato
- ▶ Ammette sia livelli di tipo fully-connected che convoluzionali
- ▶ L'unica funzione di attivazione implementata è la sigmoide
- ▶ La EduCNN è scritta con il linguaggio di programmazione *C++*

# Implementazione della Rete

## Differenze tra le due Reti

- Per ricavare i valori dei pesi iniziali, la EduCNN utilizza come generatore di numeri casuali l'algoritmo *xorshift128*

# Implementazione della Rete

## Differenze tra le due Reti

- ▶ Per ricavare i valori dei pesi iniziali, la EduCNN utilizza come generatore di numeri casuali l'algoritmo *xorshift128*
- ▶ Quando si definisce un livello, la EduCNN richiede all'utente di specificare il numero di livelli che vuole usare e la dimensione di output del livello precedente



# Implementazione della Rete

## Differenze tra le due Reti

- ▶ Per ricavare i valori dei pesi iniziali, la EduCNN utilizza come generatore di numeri casuali l'algoritmo *xorshift128*
- ▶ Quando si definisce un livello, la EduCNN richiede all'utente di specificare il numero di livelli che vuole usare e la dimensione di output del livello precedente
- ▶ La memoria allocata per ciascun livello deve essere eliminata dall'utente al termine dell'esecuzione del processo

# Implementazione della Rete

## Considerazioni

- I calcoli interni alla rete vengono svolti usando il formato di dato *double* per non perdere precisione numerica tra un passaggio e un altro

# Implementazione della Rete

## Considerazioni

- ▶ I calcoli interni alla rete vengono svolti usando il formato di dato *double* per non perdere precisione numerica tra un passaggio e un altro
- ▶ All'inizio della fase di training i pixel delle immagini vengono riscaldati nell'intervallo  $[0, 1]$  per poter essere compatibili con il formato di dato usato dalla rete

# Implementazione della Rete

## Configurazioni

- Il testing della rete viene effettuato combinando tra loro differenti tipi di livelli

# Implementazione della Rete

## Configurazioni

- ▶ Il testing della rete viene effettuato combinando tra loro differenti tipi di livelli
- ▶ Le diverse combinazioni vengono chiamate *Configurazioni*

# Implementazione della Rete

## Configurazioni

- ▶ Il testing della rete viene effettuato combinando tra loro differenti tipi di livelli
- ▶ Le diverse combinazioni vengono chiamate *Configurazioni*
- ▶ Vengono definite quattro configurazioni in modo da poter analizzare il comportamento della rete in determinate situazioni

# Implementazione della Rete

## Configurazioni

- Il learning rate  $\eta$  ed i pesi iniziali di ciascuna configurazione sono gli stessi per entrambe le reti

# Implementazione della Rete

## Configurazioni

- ▶ Il learning rate  $\eta$  ed i pesi iniziali di ciascuna configurazione sono gli stessi per entrambe le reti
- ▶ All'inizio della computazione, i pesi vengono posti ad un valore fisso di 0.001



# Implementazione della Rete

## Configurazioni

- ▶ Il learning rate  $\eta$  ed i pesi iniziali di ciascuna configurazione sono gli stessi per entrambe le reti
- ▶ All'inizio della computazione, i pesi vengono posti ad un valore fisso di 0.001
- ▶ Le configurazioni vengono inoltre testate generando i pesi iniziali in maniera casuale in modo da poter valutare le differenti accuratezze ottenute dalle due reti. Il seed scelto per effettuare i test è pari a 0

# Implementazione della Rete

## Configurazioni

- ▶ Il learning rate  $\eta$  ed i pesi iniziali di ciascuna configurazione sono gli stessi per entrambe le reti
- ▶ All'inizio della computazione, i pesi vengono posti ad un valore fisso di 0.001
- ▶ Le configurazioni vengono inoltre testate generando i pesi iniziali in maniera casuale in modo da poter valutare le differenti accuratezze ottenute dalle due reti. Il seed scelto per effettuare i test è pari a 0
- ▶ Il numero di nodi e di livelli per ciascuna configurazione viene scelto tenendo conto della componentistica hardware che si ha a disposizione per eseguire i test

# Implementazione della Rete

## Configurazioni

<i>Livello</i>	<i>Output</i>
Fully Connected	$300 \times 1$
Fully Connected	$10 \times 1$

**Table:** Configurazione 1

<i>Livello</i>	<i>Output</i>	<i>Dimensione Filtro</i>
Convoluzionale	$24 \times 24$	$5 \times 5 \times 3$
Convoluzionale	$20 \times 20$	$5 \times 5 \times 3$
Convoluzionale	$16 \times 16$	$5 \times 5 \times 3$
Fully Connected	$10 \times 1$	<b>X</b>

**Table:** Configurazione 2

# Implementazione della Rete

## Configurazioni

<i>Livello</i>	<i>Output</i>	<i>Dimensione Filtro</i>
Convoluzionale	$24 \times 24$	$5 \times 5 \times 3$
Fully Connected	$400 \times 1$	<b>X</b>
Convoluzionale	$16 \times 16$	$5 \times 5 \times 3$
Fully Connected	$10 \times 1$	<b>X</b>

**Table:** Configurazione 3

# Implementazione della Rete

## Configurazioni

<i>Livello</i>	<i>Output</i>	<i>Dimensione Filtro</i>
Convoluzionale	$24 \times 24$	$5 \times 5 \times 3$
Fully Connected	$400 \times 1$	<b>X</b>
Convoluzionale	$16 \times 16$	$5 \times 5 \times 3$
Fully Connected	$100 \times 1$	<b>X</b>
Convoluzionale	$6 \times 6$	$5 \times 5 \times 3$
Fully Connected	$10 \times 1$	<b>X</b>

**Table:** Configurazione 4

# Implementazione della Rete

## Configurazioni

- La *Configurazione 1* ha lo scopo di testare il comportamento della rete CUDA quando vengono utilizzati solo livelli di tipo fully-connected

# Implementazione della Rete

## Configurazioni

- ▶ La *Configurazione 1* ha lo scopo di testare il comportamento della rete CUDA quando vengono utilizzati solo livelli di tipo fully-connected
- ▶ Allo stesso modo, la *Configurazione 2* verifica come si comporta la rete quando viene utilizzata una sequenza di livelli convoluzionali

# Implementazione della Rete

## Configurazioni

- ▶ La *Configurazione 1* ha lo scopo di testare il comportamento della rete CUDA quando vengono utilizzati solo livelli di tipo fully-connected
- ▶ Allo stesso modo, la *Configurazione 2* verifica come si comporta la rete quando viene utilizzata una sequenza di livelli convoluzionali
- ▶ Le restanti configurazioni sono costruite alternando tra loro livelli convoluzionali di profondità tre e livelli fully-connected. Questo tipo di approccio serve a valutare se i valori di accuratezza ed i tempi di esecuzione ottenuti dalla rete CUDA sono validi anche per reti che ammettono una certa *profondità*



# Implementazione della Rete

## Cifar

- Per poter analizzare più approfonditamente le potenzialità computazionali della rete CUDA, sono state definite delle nuove configurazioni che usano come esempi di train e di test immagini contenute nel database chiamato *Cifar*

# Implementazione della Rete

## Cifar

- ▶ Per poter analizzare più approfonditamente le potenzialità computazionali della rete CUDA, sono state definite delle nuove configurazioni che usano come esempi di train e di test immagini contenute nel database chiamato *Cifar*
- ▶ Le configurazioni sono state scelte in modo da ottenere buoni risultati di accuratezza, ma al tempo stesso consentire una buona valutazione dei tempi

# Implementazione della Rete

## Cifar

- ▶ Per poter analizzare più approfonditamente le potenzialità computazionali della rete CUDA, sono state definite delle nuove configurazioni che usano come esempi di train e di test immagini contenute nel database chiamato *Cifar*
- ▶ Le configurazioni sono state scelte in modo da ottenere buoni risultati di accuratezza, ma al tempo stesso consentire una buona valutazione dei tempi
- ▶ I pesi sono inizializzati in maniera casuale e ciascuna configurazione è stata testata utilizzando un numero crescente di epoche fino ad un massimo di 10

# Implementazione della Rete

## Cifar

- ▶ Per poter analizzare più approfonditamente le potenzialità computazionali della rete CUDA, sono state definite delle nuove configurazioni che usano come esempi di train e di test immagini contenute nel database chiamato *Cifar*
- ▶ Le configurazioni sono state scelte in modo da ottenere buoni risultati di accuratezza, ma al tempo stesso consentire una buona valutazione dei tempi
- ▶ I pesi sono inizializzati in maniera casuale e ciascuna configurazione è stata testata utilizzando un numero crescente di epoche fino ad un massimo di 10
- ▶ Non è stato possibile individuare una rete sequenziale per *Cifar*, in modo da poter confrontare i risultati ottenuti dalla rete CUDA

# Implementazione della Rete

## Cifar

- Le immagini contenute in Cifar sono rappresentate per mezzo del modello di colore *RGB* che si compone di tre differenti piani di colore: rosso, verde, blu

# Implementazione della Rete

## Cifar

- ▶ Le immagini contenute in Cifar sono rappresentate per mezzo del modello di colore *RGB* che si compone di tre differenti piani di colore: rosso, verde, blu
- ▶ La dimensione di ciascuna immagine è  $32 \times 32 \times 3$ . Il valore della terza dimensione corrisponde al numero dei piani colore del modello RGB

# Implementazione della Rete

## Cifar

- ▶ Le immagini contenute in Cifar sono rappresentate per mezzo del modello di colore *RGB* che si compone di tre differenti piani di colore: rosso, verde, blu
- ▶ La dimensione di ciascuna immagine è  $32 \times 32 \times 3$ . Il valore della terza dimensione corrisponde al numero dei piani colore del modello RGB
- ▶ Le etichette sono rappresentate da numeri interi positivi ed identificano animali e oggetti del mondo reale

# Implementazione della Rete

## Cifar

- ▶ Le immagini contenute in Cifar sono rappresentate per mezzo del modello di colore *RGB* che si compone di tre differenti piani di colore: rosso, verde, blu
- ▶ La dimensione di ciascuna immagine è  $32 \times 32 \times 3$ . Il valore della terza dimensione corrisponde al numero dei piani colore del modello RGB
- ▶ Le etichette sono rappresentate da numeri interi positivi ed identificano animali e oggetti del mondo reale
- ▶ Il training ed il test set contengono rispettivamente 50000 esempi di train e 10000 di test



# Implementazione della Rete

## Configurazioni Cifar

<i>Livello</i>	<i>Output</i>
Fully Connected	$300 \times 1$
Fully Connected	$10 \times 1$

**Table:** Configurazione 1

<i>Livello</i>	<i>Output</i>	<i>Dimensione Filtro</i>
Convoluzionale	$24 \times 24$	$5 \times 5 \times 1$
Convoluzionale	$20 \times 20$	$5 \times 5 \times 1$
Fully Connected	$100 \times 1$	<b>X</b>
Convoluzionale	$6 \times 6$	$5 \times 5 \times 1$
Fully Connected	$10 \times 1$	<b>X</b>

**Table:** Configurazione 2

# Implementazione della Rete

## Configurazioni Cifar

<i>Livello</i>	<i>Output</i>	<i>Dimensione Filtro</i>
Convoluzionale	$24 \times 24$	$5 \times 5 \times 1$
Fully Connected	$400 \times 1$	$\times$
Convoluzionale	$16 \times 16$	$5 \times 5 \times 1$
Fully Connected	$10 \times 1$	$\times$

**Table:** Configurazione 3

# Analisi dei Risultati

# Analisi dei Risultati

## Hardware

- Le varie configurazioni sono state eseguite su due differenti tipi di macchine

# Analisi dei Risultati

## Hardware

- ▶ Le varie configurazioni sono state eseguite su due differenti tipi di macchine
- ▶ La prima presenta le seguenti caratteristiche tecniche

# Analisi dei Risultati

## Hardware

- ▶ Le varie configurazioni sono state eseguite su due differenti tipi di macchine
- ▶ La prima presenta le seguenti caratteristiche tecniche
  - Processore Intel Core i7-4510 da 2.00GHz

# Analisi dei Risultati

## Hardware

- ▶ Le varie configurazioni sono state eseguite su due differenti tipi di macchine
- ▶ La prima presenta le seguenti caratteristiche tecniche
  - Processore Intel Core i7-4510 da 2.00GHz
  - RAM da 6GB

# Analisi dei Risultati

## Hardware

- ▶ Le varie configurazioni sono state eseguite su due differenti tipi di macchine
- ▶ La prima presenta le seguenti caratteristiche tecniche
  - Processore Intel Core i7-4510 da 2.00GHz
  - RAM da 6GB
  - Scheda grafica Nvidia GeForce 820M da 1GB con Architettura Fermi



# Analisi dei Risultati

## Hardware

- ▶ Le varie configurazioni sono state eseguite su due differenti tipi di macchine
- ▶ La prima presenta le seguenti caratteristiche tecniche
  - Processore Intel Core i7-4510 da 2.00GHz
  - RAM da 6GB
  - Scheda grafica Nvidia GeForce 820M da 1GB con Architettura Fermi
  - Sistema operativo Ubuntu 17.10

# Analisi dei Risultati

## Hardware

- Le caratteristiche tecniche della seconda macchina sono

# Analisi dei Risultati

## Hardware

- ▶ Le caratteristiche tecniche della seconda macchina sono
  - Processore Intel Core i7-6500u da 2.50GHz (3.10GHz in Turbo Boost)

# Analisi dei Risultati

## Hardware

- ▶ Le caratteristiche tecniche della seconda macchina sono
  - Processore Intel Core i7-6500u da 2.50GHz (3.10GHz in Turbo Boost)
  - RAM da 12GB

# Analisi dei Risultati

## Hardware

- ▶ Le caratteristiche tecniche della seconda macchina sono
  - Processore Intel Core i7-6500u da 2.50GHz (3.10GHz in Turbo Boost)
  - RAM da 12GB
  - Scheda grafica Nvidia GeForce 940M da 4GB con Architettura Maxwell

# Analisi dei Risultati

## Hardware

- ▶ Le caratteristiche tecniche della seconda macchina sono
  - Processore Intel Core i7-6500u da 2.50GHz (3.10GHz in Turbo Boost)
  - RAM da 12GB
  - Scheda grafica Nvidia GeForce 940M da 4GB con Architettura Maxwell
  - Sistema operativo Windows 10 Pro

# Analisi dei Risultati

## Parametri

- I diversi valori del learning rate  $\eta$  utilizzati per testare la rete sono stati ottenuti campionando l'intervallo  $[0.001, 1.50]$  a step variabili

# Analisi dei Risultati

## Parametri

- ▶ I diversi valori del learning rate  $\eta$  utilizzati per testare la rete sono stati ottenuti campionando l'intervallo  $[0.001, 1.50]$  a step variabili
- ▶ Per ciascuna configurazione vengono ricavati i tempi di computazione della rete sequenziale, della rete CUDA ed il relativo speedup



# Analisi dei Risultati

## Parametri

- ▶ I diversi valori del learning rate  $\eta$  utilizzati per testare la rete sono stati ottenuti campionando l'intervallo  $[0.001, 1.50]$  a step variabili
- ▶ Per ciascuna configurazione vengono ricavati i tempi di computazione della rete sequenziale, della rete CUDA ed il relativo speedup
- ▶ Le configurazioni mostrate nei risultati sono quelle che hanno ottenuto il massimo valore di accuratezza in entrambe le reti tra tutti i learning rate  $\eta$  testati

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	30.50%	00:03:17	3.12
		CUDA	30.50%	00:01:03	
2	0.24	EduCNN	88.35%	00:03:04	2.60
		CUDA	88.35%	00:01:11	
3	0.62	EduCNN	93.80%	00:09:59	3.10
		CUDA	93.78%	00:03:13	
4	1.11	EduCNN	88.63%	00:11:20	3.16
		CUDA	92.17%	00:03:35	

**Table:** Risultati ottenuti eseguendo le configurazioni in ambiente Linux

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	30.50%	00:04:58	4.51
		CUDA	30.49%	00:01:06	
2	0.24	EduCNN	88.35%	00:05:05	3.91
		CUDA	88.35%	00:01:18	
3	0.62	EduCNN	93.80%	00:15:31	5.03
		CUDA	93.79%	00:03:05	
4	1.11	EduCNN	88.63%	00:16:31	4.67
		CUDA	92.17%	00:03:32	

**Table:** Risultati ottenuti eseguendo le configurazioni in ambiente Windows

# Analisi dei Risultati

## Analisi configurazioni con i pesi fissati

- Le prime tre configurazioni ottengono la stessa accuratezza in entrambe le reti e per ciascuno dei sistemi operativi, mentre la *Configurazione 4* mostra una differenza del 6%.

# Analisi dei Risultati

## Analisi configurazioni con i pesi fissati

- ▶ Le prime tre configurazioni ottengono la stessa accuratezza in entrambe le reti e per ciascuno dei sistemi operativi, mentre la *Configurazione 4* mostra una differenza del 6%.
- ▶ La *Configurazione 4* è formata da tanti livelli con un elevato numero di nodi e questo comporta una maggiore perdita di precisione nel calcolo dei pesi rispetto alle altre configurazioni. L'accuratezza ottenuta dalla rete CUDA è migliore di quella della EduCNN grazie ad un parametro di compilazione chiamato *fmad* impostato a false che privilegia la precisione alla velocità

# Analisi dei Risultati

## Analisi configurazioni con i pesi fissati

- ▶ Le prime tre configurazioni ottengono la stessa accuratezza in entrambe le reti e per ciascuno dei sistemi operativi, mentre la *Configurazione 4* mostra una differenza del 6%.
- ▶ La *Configurazione 4* è formata da tanti livelli con un elevato numero di nodi e questo comporta una maggiore perdita di precisione nel calcolo dei pesi rispetto alle altre configurazioni. L'accuratezza ottenuta dalla rete CUDA è migliore di quella della EduCNN grazie ad un parametro di compilazione chiamato *fmad* impostato a false che privilegia la precisione alla velocità
- ▶ Gli speedup sono più alti su Windows perché l'esecuzione sequenziale necessita di più risorse da parte del sistema operativo rispetto a Linux

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	92.18%	00:03:17	3.12
		CUDA	92.51%	00:01:03	
2	0.24	EduCNN	86.07%	00:03:04	2.60
		CUDA	80.57%	00:01:11	
3	0.62	EduCNN	94.32%	00:09:59	3.10
		CUDA	93.11%	00:03:13	
4	1.11	EduCNN	9.58%	00:11:20	3.16
		CUDA	9.58%	00:03:35	

**Table:** Risultati ottenuti eseguendo le configurazioni a partire da pesi iniziali generati casualmente in ambiente Linux

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	92.36%	00:04:58	4.51
		CUDA	92.51%	00:01:06	
2	0.24	EduCNN	87.35%	00:05:05	3.91
		CUDA	80.57%	00:01:18	
3	0.62	EduCNN	93.78%	00:15:31	5.03
		CUDA	93.11%	00:03:05	
4	1.11	EduCNN	9.58%	00:16:31	4.67
		CUDA	9.58%	00:03:32	

**Table:** Risultati ottenuti eseguendo le configurazioni a partire da pesi iniziali generati casualmente in ambiente Windows



# Analisi dei Risultati

## Analisi configurazioni con i pesi iniziali casuali

- L'uso dei numeri casuali migliora le accuratezze delle configurazioni 1 e 3 sia su Linux che su Windows. La rete CUDA ottiene un'accuratezza inferiore a quella della EduCNN nella *Configurazione 2* ed entrambe le reti producono risultati pessimi nella *Configurazione 4* a causa dell'overfitting

# Analisi dei Risultati

## Analisi configurazioni con i pesi iniziali casuali

- ▶ L'uso dei numeri casuali migliora le accuratze delle configurazioni 1 e 3 sia su Linux che su Windows. La rete CUDA ottiene un'accuratezza inferiore a quella della EduCNN nella *Configurazione 2* ed entrambe le reti producono risultati pessimi nella *Configurazione 4* a causa dell'overfitting
- ▶ Lo speedup è lo stesso ottenuto dalle configurazioni a pesi fissi

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	31.96%	00:35:12	3.40
		CUDA	31.96%	00:10:22	
2	0.24	EduCNN	90.11%	00:33:20	2.85
		CUDA	90.11%	00:11:42	
3	0.62	EduCNN	97.49%	01:45:00	3.18
		CUDA	97.70%	00:33:00	
4	1.11	EduCNN	13.68%	01:55:00	3.28
		CUDA	18.87%	00:35:05	

**Table:** Risultati ottenuti eseguendo le configurazioni con un numero di epoche pari a 10 in ambiente Linux

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	31.96%	00:45:46	4.38
		CUDA	31.96%	00:10:27	
2	0.24	EduCNN	90.11%	00:47:13	3.67
		CUDA	90.11%	00:12:51	
3	0.62	EduCNN	97.49%	02:32:02	4.90
		CUDA	97.71%	00:31:00	
4	1.11	EduCNN	13.68%	02:43:01	4.69
		CUDA	18.88%	00:34:44	

**Table:** Risultati ottenuti eseguendo le configurazioni con un numero di epoche pari a 10 in ambiente Windows

# Analisi dei Risultati

## Analisi configurazioni con epoche e pesi fissati

- Le configurazioni 1, 2, 3 migliorano la loro accuratezza eseguendo l'intero processo di train più volte, mentre la *Configurazione 4*, nonostante il valore di accuratezza della rete CUDA sia più preciso di quello della eduCNN, è affetta da overfitting

# Analisi dei Risultati

## Analisi configurazioni con epoche e pesi fissati

- Le configurazioni 1, 2, 3 migliorano la loro accuratezza eseguendo l'intero processo di train più volte, mentre la *Configurazione 4*, nonostante il valore di accuratezza della rete CUDA sia più preciso di quello della eduCNN, è affetta da overfitting
- Lo speedup su Windows rimane più alto di quello su Linux. La rete CUDA riduce il tempo per la fase di train da un massimo di circa 3 ore ad un massimo di circa 40 minuti nelle configurazioni più computazionalmente pesanti 3 e 4

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	97.55%	00:35:12	3.40
		CUDA	97.72%	00:10:22	
2	0.24	EduCNN	89.19%	00:33:20	2.85
		CUDA	89.34%	00:11:42	
3	0.62	EduCNN	97.65%	01:45:00	3.18
		CUDA	97.78%	00:33:00	
4	1.11	EduCNN	9.58%	01:55:00	3.28
		CUDA	9.58%	00:35:05	

**Table:** Risultati ottenuti eseguendo le configurazioni a partire da pesi iniziali generati casualmente e numero di epoche pari a 10 in ambiente Linux

# Analisi dei Risultati

## Risultati

<i>Configurazione</i>	$\eta$	<i>Rete</i>	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>	<i>Speedup</i>
1	0.09	EduCNN	97.73%	00:45:46	4.38
		CUDA	97.72%	00:10:27	
2	0.24	EduCNN	90.47%	00:47:13	3.67
		CUDA	90.53%	00:12:51	
3	0.62	EduCNN	97.61%	02:32:02	4.90
		CUDA	97.78%	00:31:00	
4	1.11	EduCNN	9.58%	02:43:01	4.69
		CUDA	9.58%	00:34:44	

**Table:** Risultati ottenuti eseguendo le configurazioni a partire da pesi iniziali generati casualmente e numero di epoche pari a 10 in ambiente Windows



# Analisi dei Risultati

## Analisi configurazioni con epoche e pesi iniziali casuali

- L'utilizzo dei numeri casuali consente di ottenere un'accuratezza con un errore inferiore al 3% per le configurazioni 1 e 3. La *Configurazione 2* peggiora del 1% su Linux e migliora di 0.3% in Windows. La *Configurazione 4* rimane affetta da overfitting come per i pesi fissi.

# Analisi dei Risultati

## Analisi configurazioni con epoche e pesi iniziali casuali

- ▶ L'utilizzo dei numeri casuali consente di ottenere un'accuratezza con un errore inferiore al 3% per le configurazioni 1 e 3. La *Configurazione 2* peggiora del 1% su Linux e migliora di 0.3% in Windows. La *Configurazione 4* rimane affetta da overfitting come per i pesi fissi.
- ▶ Lo speedup è lo stesso ottenuto dalle configurazioni con le epoche e pesi fissi

# Analisi dei Risultati

## Risultati Cifar

<i>Configurazione</i>	$\eta$	Epoche	<i>Accuratezza</i>	<i>Tempo [h:m:s]</i>
1	0.2	1	33.09%	00:03:01
2	1.6	10	31.82%	00:06:49
3	1.2	10	33.06%	00:15:38

**Table:** Risultati ottenuti eseguendo le configurazioni scelte per Cifar

# Analisi dei Risultati

## Analisi configurazioni Cifar

- Le accuratezze ottenute dalle diverse configurazioni sono estremamente basse, ma potrebbero aumentare utilizzando particolari tecniche:

# Analisi dei Risultati

## Analisi configurazioni Cifar

- ▶ Le accuratezze ottenute dalle diverse configurazioni sono estremamente basse, ma potrebbero aumentare utilizzando particolari tecniche:
  - L'aggiunta di un livello *MaxPooling* per ridurre la dimensione spaziale dei singoli piani colore in modo da individuare con maggior facilità le caratteristiche discriminanti di ciascuna immagine

# Analisi dei Risultati

## Analisi configurazioni Cifar

- ▶ Le accuratze ottenute dalle diverse configurazioni sono estremamente basse, ma potrebbero aumentare utilizzando particolari tecniche:
  - L'aggiunta di un livello *MaxPooling* per ridurre la dimensione spaziale dei singoli piani colore in modo da individuare con maggior facilità le caratteristiche discriminanti di ciascuna immagine
  - Effettuare un pre-processing dei dati di input per eliminare il noise presente nelle immagini ed evidenziare in maniera chiara le forme geometriche e i colori che compongono le immagini stesse

# Analisi dei Risultati

## Analisi configurazioni Cifar

- ▶ Le accuratze ottenute dalle diverse configurazioni sono estremamente basse, ma potrebbero aumentare utilizzando particolari tecniche:
  - L'aggiunta di un livello *MaxPooling* per ridurre la dimensione spaziale dei singoli piani colore in modo da individuare con maggior facilità le caratteristiche discriminanti di ciascuna immagine
  - Effettuare un pre-processing dei dati di input per eliminare il noise presente nelle immagini ed evidenziare in maniera chiara le forme geometriche e i colori che compongono le immagini stesse
  - Combinare diverse reti addestrate tra loro seguendo regole specifiche: il cosiddetto *Network Surgeon*

# Conclusioni



# Conclusioni

## Conclusioni

- La rete CUDA consente di ottenere dei buoni risultati di accuratezza e dei bassi tempi di computazione rispetto alla corrispettiva rete sequenziale

# Conclusioni

## Conclusioni

- ▶ La rete CUDA consente di ottenere dei buoni risultati di accuratezza e dei bassi tempi di computazione rispetto alla corrispettiva rete sequenziale
- ▶ Utilizzando la rete CUDA si evita di sovraccaricare ulteriormente la CPU di lavoro e di riempire con grandi quantità di dati la memoria RAM

# Conclusioni

## Conclusioni

- ▶ La rete CUDA consente di ottenere dei buoni risultati di accuratezza e dei bassi tempi di computazione rispetto alla corrispettiva rete sequenziale
- ▶ Utilizzando la rete CUDA si evita di sovraccaricare ulteriormente la CPU di lavoro e di riempire con grandi quantità di dati la memoria RAM
- ▶ Non dipende da librerie di terze parti e può essere eseguita anche sul sistema operativo macOS