Tool CNN per analisi e predizione di crisi epilettiche

Manuel Cretone, Emilio Silvestri Tutor: prof Marco Piangerelli Group project



OUTLINE

Introduzione

Workflow progetto

Obiettivi

Tecnologie

Backend

Frontend

Il software

Tool analisi

Tool predizione

Tool training

Basi teoriche

Epilessia

Reti neurali

Reti neurali Convoluzionali

Training

Dataset



- ▶ studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend

- studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend

- studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend

- studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend

- studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend

- studio EEG e crisi epilettiche
- studio teorico CNN
- studio tool di sviluppo
- creazione di un dataset di allenamento bilanciato
- creazione e allenamento rete per le predizioni
- costruzione di architettura web backend e frontend



- Offrire una visione grafica di file .edf contenenti tracciati elettroencefalografici
- Ridurre la fase di individuazione di crisi epilettiche all'interno di un tracciato EEG
- Offrire un ambiente di allenamento di reti neurali personalizzate per l'individuazione di crisi epilettiche





- Offrire una visione grafica di file .edf contenenti tracciati elettroencefalografici
- Ridurre la fase di individuazione di crisi epilettiche all'interno di un tracciato EEG
- Offrire un ambiente di allenamento di reti neurali personalizzate per l'individuazione di crisi epilettiche





- Offrire una visione grafica di file .edf contenenti tracciati elettroencefalografici
- Ridurre la fase di individuazione di crisi epilettiche all'interno di un tracciato EEG
- Offrire un ambiente di allenamento di reti neurali personalizzate per l'individuazione di crisi epilettiche





- Offrire una visione grafica di file .edf contenenti tracciati elettroencefalografici
- Ridurre la fase di individuazione di crisi epilettiche all'interno di un tracciato EEG
- Offrire un ambiente di allenamento di reti neurali personalizzate per l'individuazione di crisi epilettiche



SVILUPPI FUTURI

- ► Deployment su server remoto
- ► Maggiore personalizzazione delle reti
- Ottimizzazione e diversificazione delle reti pre-allenate offerte

SVILUPPI FUTURI

- ► Deployment su server remoto
- ► Maggiore personalizzazione delle reti
- Ottimizzazione e diversificazione delle reti pre-allenate offerte

SVILUPPI FUTURI

- ► Deployment su server remoto
- ► Maggiore personalizzazione delle reti
- Ottimizzazione e diversificazione delle reti pre-allenate offerte

BACKEND

django, pytorch, pyedflib



FRONTEND

angular, ionic, highchart

TOOL ANALISI

blablabla



TOOL PREDIZIONE

blablabla



TOOL TRAINING

blablabla

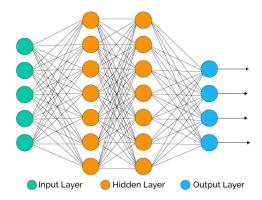


EPILESSIA

bla bla



RETI NEURALI



- Modelli computazionali basati su neuroni
- ► Ispirate al funzionamento biologico del cervello
- Applicate nella risoluzione di problemi ingegneristici, informatici, di simulazione...



RETI NEURALI

Funzionamento neurone:

- ▶ Una o più connessioni in ingresso e uscita
- Somma dei segnali in ingresso moltiplicati per i pesi delle connessioni
- Aggiunta di un eventuale bias
- Funzione di attivazione e segnale in uscita

Tipologie di rete:

- Feedforward: layer di neuroni con connessioni in una sola direzione
- Ricorsive: connessioni con neuroni dello stesso livello o all'indietro



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- ► Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ► Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- ► Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- ► Apprendimento per rinforzo

Passi allenamento supervisionato

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- ► Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- ► Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- ► Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- ► Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ► Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- ► Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation→ Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- ► Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Apprendimento per rinforzo

Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function → Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update



Tipologie di allenamento:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Apprendimento per rinforzo

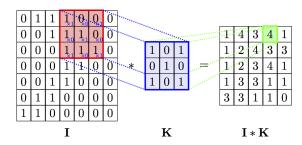
Passi allenamento supervisionato:

- Inizializzazione
- Feed-forward
- ▶ Loss function→ Calcolo errore
- ▶ Differentiation → Diminuzione loss
- Backpropagation e weights update

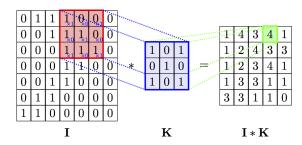


RETI NEURALI CONVOLUZIONALI

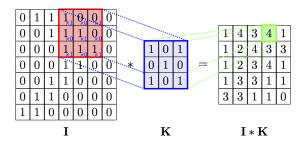
- ► Adatte ad input di grandi dimensioni
- ▶ Riconoscimento di pattern nell'architettura
- Applicazione di kernel per individuazione di features nei dati



- ► Adatte ad input di grandi dimensioni
- ▶ Riconoscimento di *pattern* nell'architettura
- Applicazione di kernel per individuazione di features nei dati



- ► Adatte ad input di grandi dimensioni
- ▶ Riconoscimento di pattern nell'architettura
- Applicazione di kernel per individuazione di features nei dati



Parametri layer convoluzionale:

- ► Input (I)
- ► Profondità (K)
- ► Stride (S)
- ► Zero-Padding (P)

Dimensione del volume di output

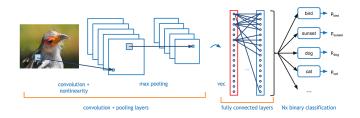
$$O = \frac{(I - K - 2P)}{S} + 1 \tag{1}$$

Parametri layer convoluzionale:

- ► Input (I)
- ► Profondità (K)
- ► Stride (S)
- ► Zero-Padding (P)

Dimensione del volume di output:

$$O = \frac{(I - K - 2P)}{S} + 1 \tag{1}$$



- ► Convolutional layers
- ▶ Pooling Layers (max, average...)
- ► Fully Connected Layers



CONVOLUZIONI 1D



In this example for natural language processing, a sentence is made up of 9 words. Each word is a vector that represents a word as a low dimensional representation. The feature detector will always cover the whole word. The height determines how many words are considered when training the feature detector. In our example, the height is two. In this example the feature detector will iterate through the data 8 times.

In this example for computer vision, each pixel within the image is represented by its x- and y position as well as three values (RGB). The feature detector has a dimension of 2×2 in our example. The feature detector will now slide both horizontally and vertically across the image.

Conv 1D (segnali, NLP...)

- ► Input bidimensionale (*C*, *W*)
- ► Il kernel scorre lungo la dimensione W

Conv 2D (immagini)

► Input tridimensionale (*C*, *H*, *W*)

Y position

► Il kernel scorre lungo le dimensioni H e W



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

- Sequential:
 - Convolution
 - ▶ ReLU activation
 - MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

- Convolution
- ► ReLU activation
- MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

- Convolution
- ReLU activation
- MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): ConvId(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
   )
   (layer2): Sequential(
    (0): ConvId(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
   )
   (drop_out): Dropout(p=0.5)
   (fc1): Linear(in_features=35, out_features=15, bias=True)
   (fc3): Linear(in_features=15, out_features=2, bias=True)
   (soft): Softmax()
}</pre>
```

- Convolution
- ReLU activation
- MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

- Convolution
- ReLU activation
- MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): ConvId(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
   )
   (layer2): Sequential(
    (0): ConvId(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
   )
   (drop_out): Dropout(p=0.5)
   (fc1): Linear(in_features=35, out_features=15, bias=True)
   (fc3): Linear(in_features=15, out_features=2, bias=True)
   (soft): Softmax()
}</pre>
```

- Sequential:
 - Convolution
 - ReLU activation
 - MaxPool

- Dropout
- Fully-connected
- ► Softma



```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): ConvId(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
   )
   (layer2): Sequential(
    (0): ConvId(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolld(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
   )
   (drop_out): Dropout(p=0.5)
   (fc1): Linear(in_features=35, out_features=15, bias=True)
   (fc3): Linear(in_features=15, out_features=2, bias=True)
   (soft): Softmax()
}</pre>
```

- Sequential:
 - Convolution
 - ReLU activation
 - MaxPool

- Dropout
- ► Fully-connected
- ► Softma



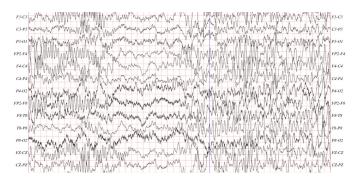
```
<bound method Module.state_dict of ConvNet(
  (layer1): Sequential(
    (0): Conv1d(23, 10, kernel_size=(10,), stride=(5,), padding=(9,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=15, stride=10, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv1d(10, 5, kernel_size=(5,), stride=(4,), padding=(4,))
    (1): ReLU()
    (2): MaxPoolId(kernel_size=10, stride=5, padding=0, dilation=1, ceil_mode=False)
  )
  (drop_out): Dropout(p=0.5)
  (fc1): Linear(in_features=35, out_features=15, bias=True)
  (fc3): Linear(in_features=15, out_features=2, bias=True)
  (soft): Softmax()
)>
```

- Sequential:
 - Convolution
 - ReLU activation
 - MaxPool

- Dropout
- Fully-connected
- Softmax



DATASET



- Children's Hospital Boston database (22 pazienti totali)
- Pazienti 1,2,3,5,7 e 8
- ▶ File con almeno un evento di crisi
- ► Frequenza di campionamento 256Hz
- 23 canali di campionamento



TRAINING DELLA RETE

Preparazione del dataset:

- Isolamento dei segnali di crisi
- Suddivisione in finestre da 30 secondi, con stride di 1
- Eliminazione segnali pre e post ictali (5 minuti)
- Selezione finestre non di crisi

► Training:

- ▶ 30 epoche
- ► Cross Entropy Loss $loss(x, class) = -x[class] + log(\sum_{j} exp(x[j]))$
- Adam Optimizer (Adaptive moment estimation)
- ▶ Validazione: 80% training set, 20% validation set
- ► **Test** su file esterni al training set



