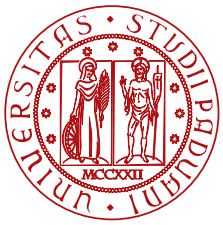


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Un approccio di deep learning per l'analisi del "segnale calcio" nella caratterizzazione della risposta biologica delle piante agli stimoli esterni

Relatore
Prof. Menegatti Emanuele

Laureando
Brillo Ivan

Correlatore
Prof. Rigoni Garola Andrea

ANNO ACCADEMICO 2023-2024

Data di laurea 19/07/2024

A papà, questo traguardo è anche il tuo.

Indice

Elenco delle figure

Capitolo 1

Introduzione e obiettivo

Le piante, proprio come tutti gli esseri viventi, subiscono costantemente un'ampia gamma di stress, ossia situazioni o condizioni che ne influenzano negativamente la crescita, lo sviluppo o la sopravvivenza stessa. Questi stress possono essere associati a stimoli di varia natura; si parla ad esempio stress idrico, termico, luminoso, nutritivo, meccanico, chimico o patogeno. Per semplicità si distingueranno in questo studio due categorie principali: stimoli abiotici se causati da condizioni ambientali esterne, e stimoli biotici quando provocati da un altro essere vivente o da un virus. Una delle maggiori strategie di contrasto a queste condizioni avverse avviene tramite lo ione calcio. Il cosiddetto "segnale calcio", ovvero la variazione nel tempo della concentrazione di ioni calcio presente all'interno della cellula, codifica un'informazione specifica relativa ai diversi fattori di stress. Questa particolare *signature* (forma) del segnale calcio permette alla pianta di attivare appropriati meccanismi di risposta mirata, ad esempio promuovendo la cicatrizzazione di una foglia dopo l'aggressione di un animale fitofago, oppure l'inizio della risposta immunitaria dovuta alla presenza di un virus. Alcuni aspetti importanti dei segnali calcio all'interno delle piante verranno approfonditi nella sezione 2.1.

Particolari trasduzioni del segnale calcio avvengono dopo aver innaffiato la pianta con acqua distillata entrata in contatto con del plasma (gas ionizzato). Quest'acqua è particolarmente reattiva chimicamente, poiché al suo interno sono presenti molteplici specie chimiche con diversa vita media, che interagiscono tra di loro. La composizione chimica di questa soluzione viene percepita dalla piante come uno stimolo esterno, promuovendo la trasduzione di un segnale calcio. La procedura di attivazione dell'acqua può essere modulata attraverso vari parametri, ad esempio il tempo di esposizione al plasma. Modificando questi parametri si possono ottenere differenti *signatures* del segnale calcio prodotto. L'aspetto chimico dell'acqua attivata e le sue ricadute biologiche varranno discussi nella sezione 2.2.

L'obiettivo della tesi è quello di ricavare i più adatti fattori di generazione dell'acqua attivata per riprodurre al meglio il segnale calcio proveniente da uno stress reale, come la disidratazione.

In questo modo si potrebbe indurre la pianta ad attivare specifiche risposte adattive prima che sperimenti particolari tipi di stress. Ad esempio si potrebbe codificare l'informazione per lo sviluppo di radici più profonde in vista di un'estate arida, semplicemente annaffiandola con un'acqua opportunamente attivata.

Studiare i segnali calcio con metodologie di analisi tradizionali è estremamente difficile. *In primis* l'equilibrio dinamico della soluzione è complicato per via dell'altissimo numero di fattori che lo influenzano e per la brevissima durata di alcune specie presenti. Bisogna poi tener conto che piante diverse, dunque con DNA differenti, possono rispondere in maniera diversa alla medesima acqua attivata, aggiungendo una nuova variabile. Per condurre queste analisi si è allora optato per un approccio di identificazione basata su tecniche di *deep learning*, che permette di mascherare questa complessità e di ricavare *pattern* utili per l'analisi dei segnali. Le tecniche utilizzate per raggiungere l'obiettivo saranno discusse nei capitoli 3, 6 e 7, mentre un'introduzione teorica a questi strumenti di *machine learning* nei capitoli 4 e 5.

I dati sono stati forniti dal Gruppo di ricerca coordinato dalla prof. Lorella Navazio presso il Dipartimento di Biologia dell'università di Padova in collaborazione con Dott. Vanni Antoni dell'Istituto CNR di Scienza e Tecnologia dei Plasmi.

Capitolo 2

Background biologico

2.1 Segnali calcio

Nelle piante, la concentrazione dello ione calcio all'interno del citosol cellulare ($[Ca^{2+}]_{cyt}$) aumenta in risposta a molti stimoli esterni [17]. Questi stimoli possono variare da semplice radiazione luminosa a fattori meccanici come il tocco. Risulta fondamentale l'azione di questi segnali in risposta sia a stress abiotici, che rappresentano fattori come siccità, temperature estreme e caratteristiche del terreno, sia a quelli biotici come azione a un possibile patogeno [9]. Lo ione calcio funziona come anello di collegamento tra stimoli ambientali e la loro trasduzione in segnali intracellulari che favoriranno una risposta biologica efficace per rispondere allo stress subito.

2.1.1 Schema della trasduzione dei segnali calcio

Lo schema di una generica traduzione del segnale in risposta a un stimolo esterno è fornita di seguito seguendo quanto riportato dall'articolo [17]:

1. Lo stress viene percepito dai recettori presenti nella membrana cellulare.

2. Lo stimolo viene trasdotto attraverso l'attivazione di particolari pompe calcio presenti nella membrana cellulare, che trasportano attivamente (utilizzando energia) lo ione all'esterno, oppure attraverso l'apertura di canali che tramite diffusione lasciano entrare Ca^{2+}. Per questo motivo lo ione calcio si dice appartenga alla categoria di messaggeri secondari nella pianta, dal momento che porta l'informazione dello stimolo passando prima per un processo di trasduzione. Stress diversi, utilizzano diversi meccanismi per la trasduzione, in particolare si è visto che stress abiotici differenti utilizzano diversi canali nella membrana cellulare per la segnalazione [20].

3. L'aumento della concertazione di Ca^{2+} viene captata da alcune proteine sensibili a questo ione che vengono chiamate chinasi.

4. L'enzima chinasi tramite fosforilazione e de-fosforilazione dei fattori di trascrizione (proteine che trascrivono il DNA in mRNA nel nucleo), riesce a modificare l'espressione di determinati geni all'interno del DNA, dal momento che ne controlla la loro trascrizione.

5. La traduzione dell'mRNA così trascritto, produrrà proteine importanti nella risposta allo stress subito, andando a modificare il metabolismo della pianta, che riuscirà a sviluppare un certo livello di tolleranza allo stimolo.

2.1.2 Ca^{2+} *signatures* e memoria cellulare

Stress diversi producono perturbazioni diverse nella concentrazione di $[Ca^{2+}]_{cyt}$ che andranno a generare risposte specifiche per lo stimolo coinvolto[17]. Questa peculiarità nei messaggi trasdotti dallo ione calcio prende il nome di Ca^{2+} *signatures*. Le proteine che funzionano come sensori dello ione calcio sono abbastanza precise nel riconosce specifiche Ca^{2+} *signatures* e attivarsi di conseguenza, andando a generare risposte complessive diverse.

Tuttavia va evidenziato che alte e prolungate concentrazioni di Ca^{2+} portano all'apoptosi della cellula, indicando che l'equilibrio dello ione calcio all'interno della cellula è estremamente delicato. Le cellule presentano anche una memoria dei precedenti stress, che permette una trasduzione più lieve a stress continuativi, non andando quindi a intaccare in maniera irreversibile l'ambiente intracellulare [17].

2.1.3 Misurazione dei segnali calcio nelle piante

Tutti gli esperimenti presi in esame in questa tesi sono stati fatti sulla pianta modello *Arabidopsis thaliana* (mostrata in figura 2.1), geneticamente modificata per esprimere una proteina bioluminescente, chiamata equorina, in grado di emettere radiazioni luminose quando legata a uno ione Ca^{2+}. Questo tipo di indicatore è utilizzato per misurare la concentrazione di calcio all'interno del citosol cellulare, misurando l'intensità luminosa dell'equorina attraverso un luminometro, cioè un dispositivo in grado di conteggiare i fotoni emessi. Un esempio illustrativo del funzionamento dell'equorina e del luminometro sono riportati nell'immagine 2.2 e 2.3.

Figura 2.1: Immagine di *Arabidopsis thaliana* presa dall'articolo [3].

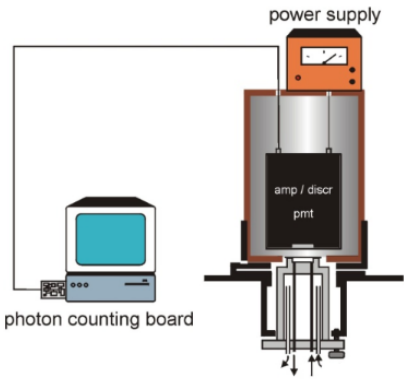


Figura 2.2: Schema di funzionamento di un luminometro. I fotoni passano per un tubo fotomoltiplicatore (PMT) e colpiscono un fotocatodo che emetterà elettroni per via dell'effetto fotoelettrico. La corrente finale, che sarà proporzionale al numero di fotoni incidenti, viene misurata producendo un segnale in uscita.

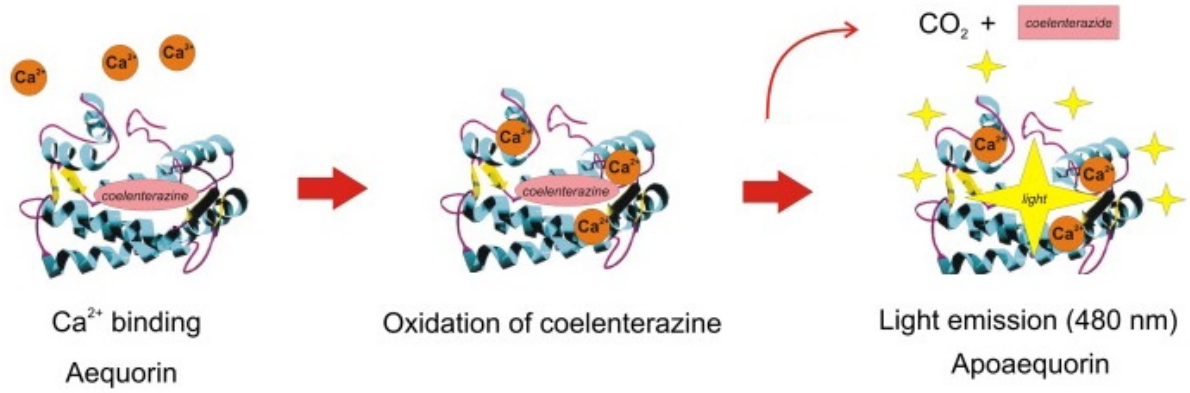


Figura 2.3: Immagine esemplificativa dell'utilizzo di equorina come indicatore della concentrazione di calcio all'interno del citosol cellulare. La reazione è irreversibile e porta all'emissione di un fotone. Immagine presa dall'articolo [4].

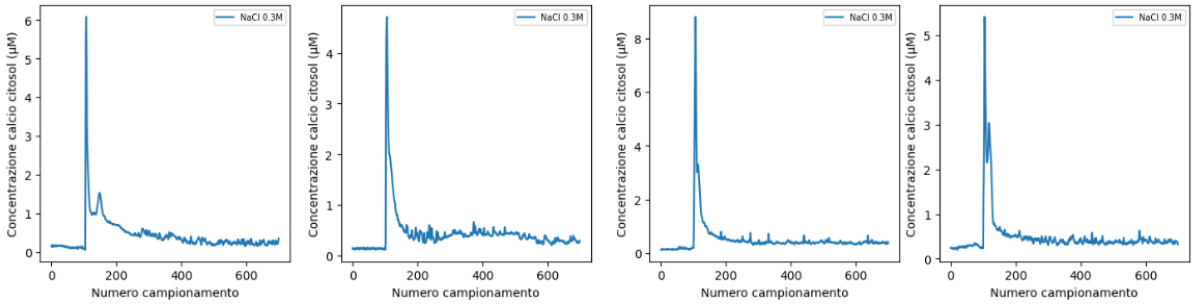
2.1.4 Caratteristiche dei segnali calcio nelle piante

Analizziamo la forma di alcuni segnali calcio in presenza di stress abiotici, indotti da due soluzioni, la prima contenente cloruro di sodio e la seconda contenente acqua ossigenata (figura 2.4). In generale i segnali calcio presentano diversi andamenti a seconda delle condizioni alle quali è sottoposta la pianta, ma possiamo rilevare alcune caratteristiche comuni:

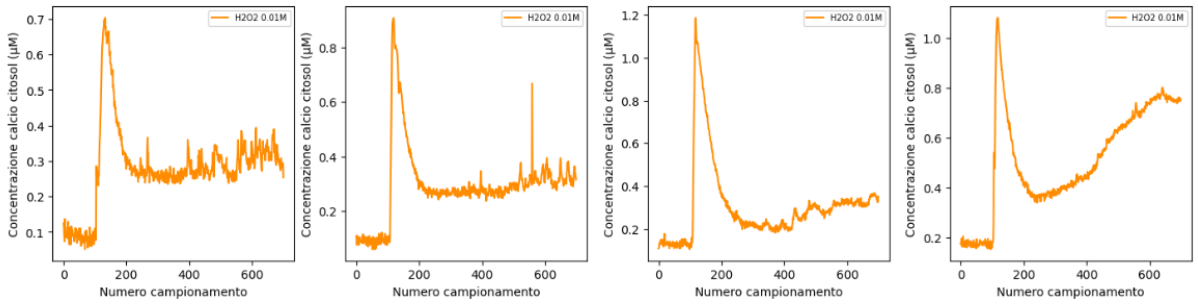
- Notiamo un *offset* iniziale che rappresenta la concertazione di calcio a riposo nel citosol della pianta, cioè prima dello stimolo.
- Quando la pianta viene stimolata, ad esempio tramite una soluzione salina o acqua ossigenata, viene rilevato un picco di concentrazione di calcio che corrisponde alla trasduzione

dello stress subito.

- Quello che succede dopo il picco può essere un ritorno alle condizioni di riposo oppure una deriva della concentrazione di calcio che può portare alla morte cellulare (apoptosi).



(a) Segnale calcio in risposta ad una soluzione di NaCl 0.3M.



(b) Segnale calcio in risposta ad una soluzione di H_2O_2 0.01M

Figura 2.4: Segnali calcio prodotti da *Arabidopsis thaliana* in seguito a stress abiotici.

2.2 Plasma-activated water (PAW)

Il plasma detto anche quarto stato della materia è un gas ionizzato che esibisce proprietà collettive. La famiglia dei plasmi si estende su un vasto intervallo di temperature e densità e di questa famiglia un gruppo particolare è costituito dai cosiddetti ”plasmi freddi” ovvero plasmi prodotti a pressione atmosferica e che si trovano in non equilibrio termico ovvero con temperatura degli elettroni molto maggiore di quella degli ioni.

Quando portiamo in contatto dell’acqua distillata con un plasma freddo, questa prende il nome di acqua attivata (PAW). Questo processo la rende ricca di sostanze altamente reattive, tutte presenti nella soluzione e governate da un complicato equilibrio dinamico.

2.2.1 Proprietà fisiche e biologiche dell'acqua attivata

L'acqua attivata dal plasma contiene alte concentrazioni di *Reactive Nitrogen Species* (RNS) come NO_2^-, NO_3^-, NO e $ONOO^-$ e di *Reactive Oxygen Species* (ROS) come O_3, H_2O_2, O_2^- e OH^- [19]. Ognuna di queste sostanze presenta diversi tempi di vita all'interno del PAW.

Recenti studi suggeriscono l'effetto benefico dell'irrigazione tramite PAW nell'agricoltura. Infatti alcuni dei composti presenti hanno note capacita antibatteriche, che si manifestano anche per via dell'acidità dell'acqua attivata [13]. Inoltre altri studi sottolineano miglioramenti nella germinazione dei semi, nella crescita e nella riproduzione delle piante [1]. Queste proprietà potrebbero portare al loro utilizzo in agricoltura come sostituzione ecologica ai pesticidi [9].

2.2.2 Metodi per la generazione di PAW

L'acqua attivata può essere generata attraverso diverse tecniche e sorgenti, ai nostri fini considereremo solo quella ottenuta tramite due dispositivi: una sorgente di tipo *atmospheric dielectric barrier discharge* (DBD) e una *plasma torch* (PT), in quanto sono state le uniche due utilizzate durante la raccolta dati presa in esame nella tesi.

Atmospheric dielectric barrier discharge (DBD)

Il DBD è un generatore di plasma costituito da diversi elettrodi in rame, ognuno racchiuso da due tubi in vetro concentrici. Alimentando l'apparecchio con picchi di tensione a 10 kV con frequenza utilizzata a 12 kHz oppure a 20 kHz, nella cavità venutasi a formare tra i due tubi in vetro si genera un anello di plasma, dovuto alla ionizzazione del gas presente. Attraverso un compressore, l'aria viene fatta fluire attraverso il plasma freddo fino a entrare in contatto con l'acqua distillata presente alla base del dispositivo, all'interno di una piastra Petri. Lo schema del generatore è mostrato in figura 2.5.

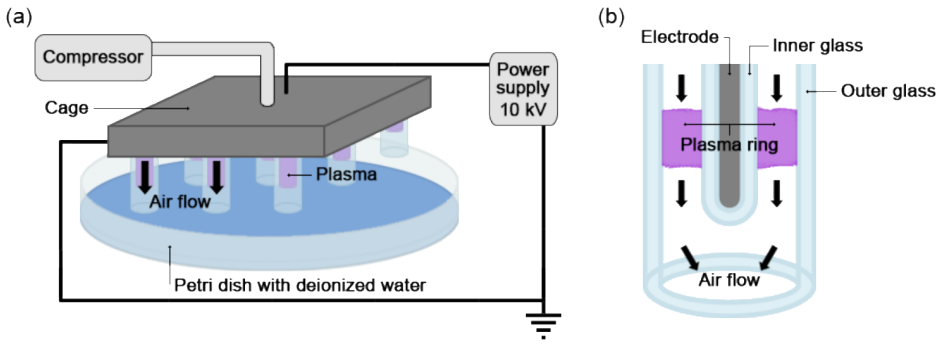


Figura 2.5: (a) Schema del DBD utilizzato. (b) Dettaglio di un elettrodo. Immagine presa dall'articolo [8].

Plasma torch (PT)

La torcia al plasma è il secondo metodo di generazione utilizzato. È un dispositivo industriale che permette il raggiungimento di potenze ben più elevate rispetto al DBD di circa 900W. Uno schema illustrativo del suo funzionamento è mostrato in figura 2.6.

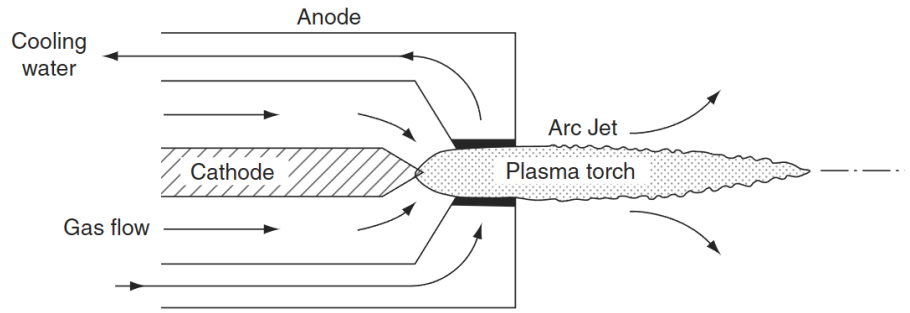


Figura 2.6: Schema generale di un plasma torch. Immagine presa dal libro [14].

2.3 Relazione tra segnali calcio nelle piante e PAW

Si è trovato che l'acqua attivata dal plasma freddo viene riconosciuta dalla pianta tramite la trasduzione di specifici segnali calcio[9], che presentano caratteristiche diverse a seconda dei parametri con cui viene generato il PAW. Questi fattori saranno discussi nelle sezioni successive.

Per verificare la veridicità dell'impianto sperimentale, sono state condotte ulteriori prove mantenendo costanti altri fattori, come i nutrienti, e utilizzando solo acqua distillata per l'irrigazione. La mancanza di sali derivante dalla distillazione dell'acqua permette infatti di escludere con buona confidenza la presenza di specie ioniche che potrebbero stimolare le cellule. È stato dimostrato quindi che, non scatenando alcuna risposta in termini di generazione di segnali calcio nelle piantine, questo tipo di irrigazione può essere utilizzata come metodologia di controllo per la validazione della stimolazione cellulare mediante PAW [9].

È interessante notare come i composti ROS e RNS non producono una risposta significativa della pianta tramite segnali calcio se presi singolarmente e con le stesse concentrazioni con cui si trovano nell'acqua attivata, ma risulta essere la loro combinazione a far scatenare la specifica trasduzione del segnale [9].

2.3.1 Parametri di generazione del PAW e la loro influenza nel segnale calcio trasdotto

Diversi parametri nella generazione di PAW possono essere cambiati. Questi fattori possono influenzare le concentrazioni delle diverse specie chimiche all'interno dell'acqua attivata e con-

seguentemente la trasduzione dello stimolo nella pianta. Alcuni di questi parametri possono essere il tipo e la geometria dello strumento, la sua alimentazione e vicinanza al piatto di Petri, la quantità di acqua presente e molti altri. Alcuni di particolare importanza per questa tesi vengono presi in considerazione singolarmente.

Effetto del tempo di esposizione dell'acqua distillata al DBD e della sua frequenza di scarica nella risposta della pianta

La durata dell'esposizione dell'acqua distillata al trattamento con plasma freddo e la frequenza di scarica utilizzata nel DBD si correla fortemente con la magnitudo della concentrazione di Ca^{2+} nella risposta della pianta *Arabidopsis thaliana*, come mostrato in figura 2.7. Effetti simili vengono riportati per tempi di esposizione diversi anche nel PT-PAW [9].

Queste differenze possono essere interpretate grazie alle concentrazioni diverse dei singoli componenti (ROS e RNS) che sono presenti nell'acqua attivata a seconda della frequenza di scarica e alla durata di esposizione [8].

Effetto del tempo di storage e temperatura dell'acqua attivata tramite torcia nella risposta della pianta

Un altro fattore che influenza la risposta tramite segnale calcio di una pianta è la durata e la temperatura a cui è avvenuto lo storage del PAW. Gli effetti sono mostrati in figura 2.8. Una spiegazione di questo effetto è dovuto alla diminuzione di concentrazione dei composti presenti nel PAW dovuti a differenti equilibri chimici all'interno della soluzione. Queste reazioni sono favorite dalla temperatura.

Confronto tra DBD e PT

Le differenze tra le due metodologie di generazione di PAW sono molto accentuate, in particolare a bassi periodi di esposizione. Questo può essere parzialmente spiegato calcolando l'energia trasferita dal macchinario all'acqua distillata, considerandola in prima approssimazione un indicatore della sua attivazione. Nel nostro caso la PT a parità di tempo trasferisce una maggior quantità di energia rispetto al DBD. Questo *trend* tende a scomparire a periodi di esposizione più elevati, mostrando una saturazione per alte energie, come mostrato in figura 2.9.

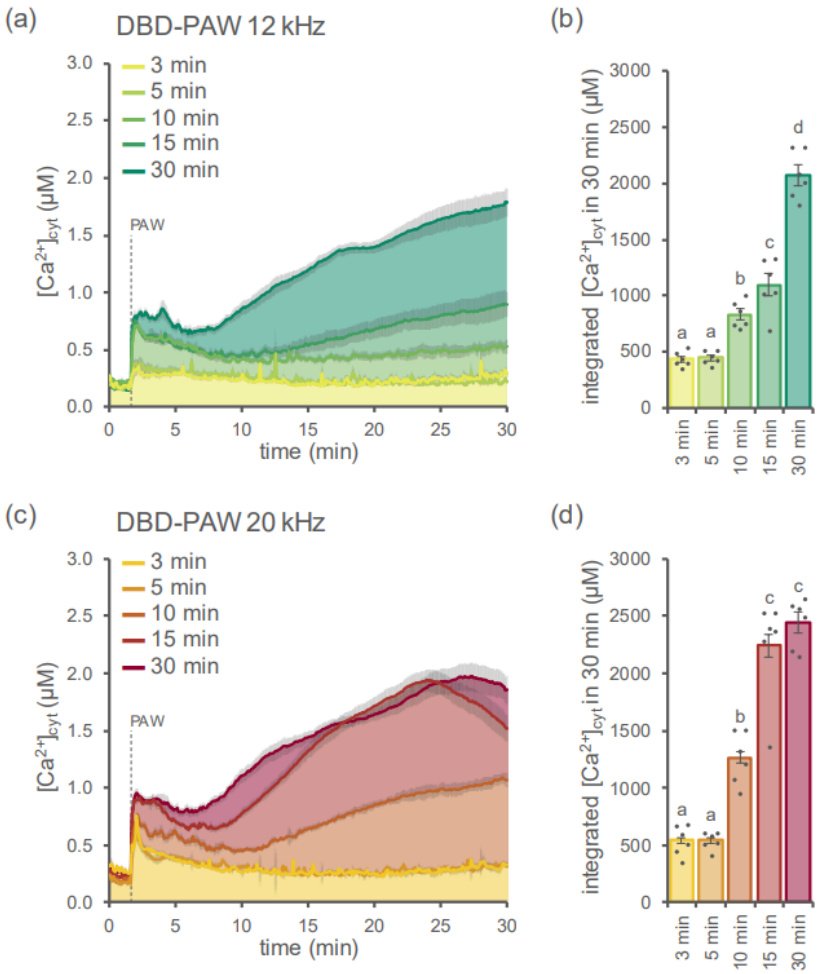


Figura 2.7: Risposta di *Arabidopsis thaliana* a DBD-PAW generati con diverse frequenze di scarica e tempi di esposizione. Immagine presa dall'articolo [8].

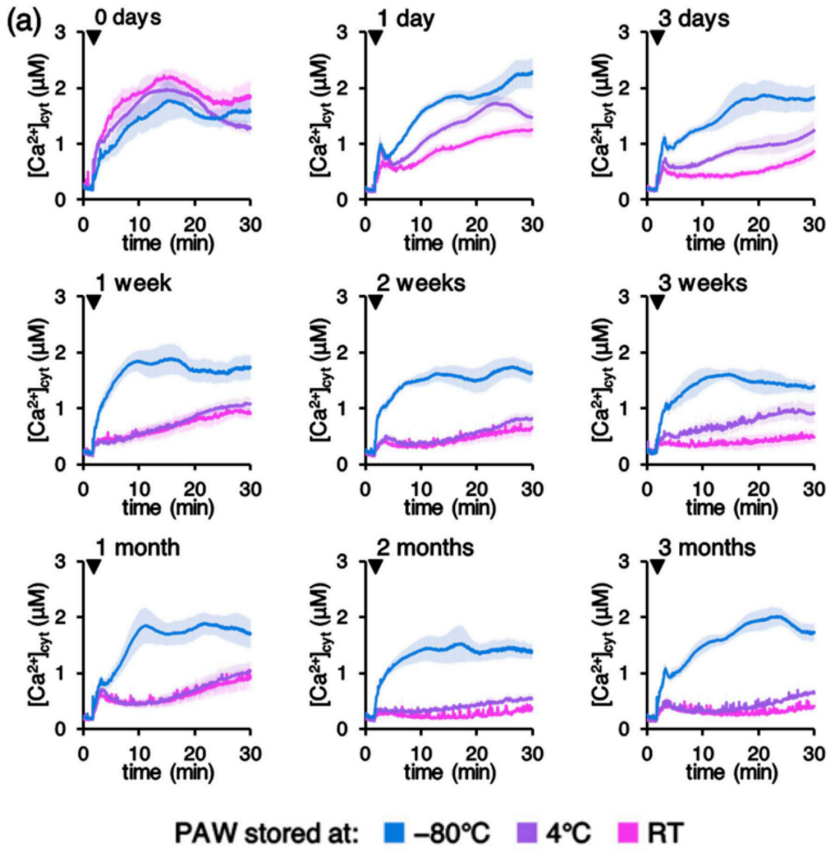


Figura 2.8: Risposta di *Arabidopsis thaliana* a PT-PAW con diversi tempi di storage e temperature. Immagine presa dall'articolo [9].

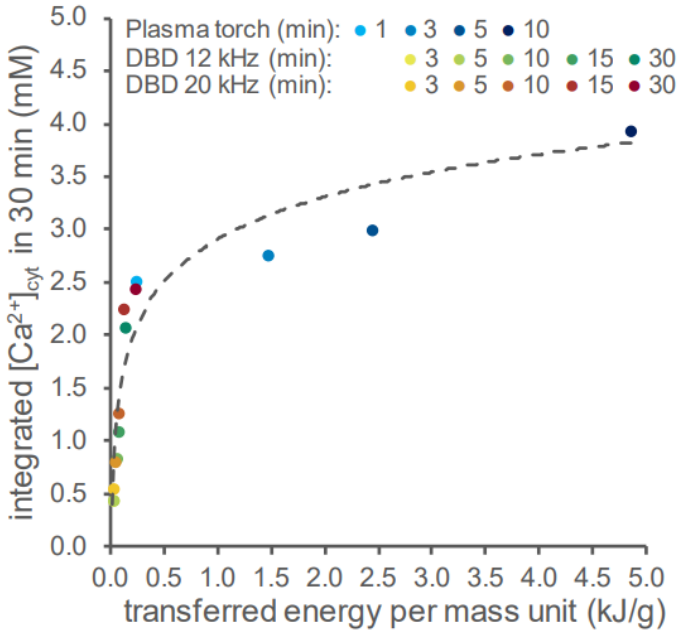


Figura 2.9: Risposta di *Arabidopsis thaliana* a PT-PAW (900W) e DBD-PAW con diversi tempi di esposizione. Immagine presa dall'articolo [8].

Capitolo 3

Data Preprocessing

3.1 Modello matematico della base di dati

Le serie temporali che rappresentano i segnali calcio nell'arco dei 30 minuti successivi alla somministrazione del PAW, sono stati campionati tramite 1800 punti. Possiamo descrivere questi segnali tramite funzioni, dove ogni punto del dominio, che rappresenta il numero del campionamento, viene mappato alla concentrazione di calcio nel citosol espresso in µM.

$$s_i : \{0, 1, 2, \ldots, 1799\} \to \mathbb{R}_0^+ \tag{3.1}$$

Una formulazione equivalente, ma più utile per i capitoli successivi, consiste nell'identificare ogni serie con un vettore a 1800 dimensioni, in particolare possiamo esprimere un segnale generico come: $\boldsymbol{x}_i \in \mathbb{R}^{1800}$.

Ogni sequenza temporale è corredata da alcuni metadati importati:

1. Data di acquisizione sperimentale dei dati.
   M_1 = {22/03/22, 22/03/29, 22/03/29, 22/03/30, 22/04/05, 22/05/03}

2. Strumento e alimentazione che ha creato il PAW.
   M_2 = {DBD 12 kHz, DBD 20 kHz, PT 900 W}

3. Tempo di esposizione dell'acqua distillata al processo.
   M_3 = {180 s, 300 s, 600 s, 900 s, 1800 s}

4. Tempo di storage dell'acqua attivata a temperatura ambiente.
   M_4 = {0 week, 1 week, 2 weeks}

I campioni distinti disponibili per questo progetto sono 239. Indicando con $\boldsymbol{x}_i$ la *i-esima* serie temporale, $i \in \{1, 2, \ldots, 239\}$, sappiamo esistere una funzione g che lega il segnale con i

suoi metadati:

$$g : I \rightarrow M_1 \times M_2 \times M_3 \times M_4 \tag{3.2}$$

Il database sarà dunque definito come:

$$D = \{(\boldsymbol{x}_i, g(i)), \forall i \in I\} \tag{3.3}$$

3.1.1 Prime considerazioni sui dati utilizzati

Da una considerazione preliminare circa la costruzione del dataset di addestramento è emerso che la dimensione limitata del database sperimentale avrebbe potuto rappresentare una complicazione restringendo il campo di applicabilità per molti approcci di *machine learning* mirati alla classificazione e identificazione di parametri ottimali. Questo aspetto rappresenta, per quanto discusso nel presente studio, uno dei principali attori che hanno vincolato la scelta del modello e dello scaling dei parametri che lo caratterizzano.

Oltre allo scarno numero di serie temporali, i parametri registrati sono parzialmente insoddisfacenti. Ad esempio tutti i campioni che sono stati conservati per settimane in storage, venivano mantenuti a temperatura ambiente invece di provare diverse combinazioni.

Inoltre è importante evidenziare che, data la comunque elevata difficoltà tecnica della procedura sperimentale, molti parametri potenzialmente correlati non sono stati considerati. Ad esempio la temperatura e l'umidità della stanza potrebbero alterare in maniera non trascurabile le reazioni chimiche all'interno dell'acqua attivata. Per le sperimentazioni future sarà importante raccogliere il maggior numero possibile di fattori che possano influenzare la chimica dell'acqua attivata.

3.2 Implementazione del database in python

Le serie temporali sperimentalmente trovate con il luminometro sono state inserite all'interno di fogli di calcolo, aventi come colonne le serie temporali e come metadati i parametri precedentemente discussi.

Attraverso uno script Python, utilizzando il modulo *openpyxl*, sono stati raccolti i segnali e i metadati dai diversi fogli di calcolo, applicando successivamente un filtro anti-duplicati. Si è creato così un dataframe pandas che implementa la struttura della base di dati, dove le colonne sono identificate con i seguenti nomi: *serie*, *date*, *type*, *exposure_time*, *supply_delay*. La serie è rappresentata come un *numpy array* da 1800 valori.

Per mitigare la mancanza dell'informazione di temperatura, si è deciso di integrare l'informazione temporale della data di acquisizione con la temperatura ambientale. Partendo dalla

data di registrazione dei dati, attraverso il modulo *meteostat*, è possibile risalire alle temperature medie di Padova in quel giorno. Sfortunatamente questa misura non è altamente rappresentativa della temperatura a cui sono avvenuti gli esperimenti, dunque verrà tralasciata in seguito.

3.3 Filtri temporali

Le serie temporali misurate attraverso il luminometro risultano particolarmente rumorose per via della sensibilità dello strumento. Prima di fornirle alla rete neurale è opportuno filtrarle cercando di mantenere un andamento più attinente possibile per non perdere dettagli importanti ma allo stesso tempo filtrare i picchi di rumore. Fare questo puó semplificare il lavoro che dovrà svolgere la rete neurale nell'analizzare le forme dei segnali. Sono stati provati diversi filtri per i segnali calcio, nel prossimo paragrafo verrà riportato solo quello più promettente.

In generale un filtro anti-rumore prende come parametro una finestra (*windows size*), che inizialmente è posizionata all'inizio della sequenza. Ad ogni iterazione la finestra viene spostata all'interno della serie temporale di una posizione, fino a scorrerla tutta. Ad ogni iterazione viene calcolato un valore tramite uno specifico algoritmo che andrà aggiunto alla lista contenente i punti filtrati.

Procedendo in questo modo la nuova sequenza di valori avrebbe una lunghezza pari a $n = sequence\_length - windows\_size + 1$, poiché arrivato alla posizione $i = sequence\_length - windows\_size + 1$, la nuova finestra uscirebbe dall'array iniziale.

Per ovviare a questo problema di dimensionalità diverse esistono diverse soluzioni, la più semplice è aggiungere un *padding* a destra e sinistra della nuova serie, che contengano entrambi $k = windows\_size/2$ valori, inizializzati a quello calcolato dall'algoritmo più vicino ad essi.

Indicando con $A(seq, params...)$ l'algoritmo specifico per il calcolo del nuovo valore filtrato, che prende in input una parte della sequenza iniziale di lunghezza *windows size* e altri parametri dipendenti dalla sua implementazione, possiamo definire l'algoritmo generale come segue:

3.3.1 Asymmetrical alpha-trimmed median filter

Il filtro scelto per il progetto inizialmente ordina i valori all'interno della finestra selezionata ed elimina le due code della distribuzione secondo un parametro alpha. In particolare elimina $\lfloor \alpha/4 \rfloor$ nella coda sinistra e $\lceil 3\alpha/4 \rceil$ nella coda destra. Come ultimo passo ritorna la media dei valori rimanenti. Il motivo di questa asimmetria nello scartare i dati si ritrova nel fatto empirico che la maggior parte del rumore risulta di segno positivo rispetto all'andamento sottostante della curva. L'algoritmo è sviluppato sotto (2), mentre uno schema del suo funzionamento è riportato in figura 3.2

Algorithm 1 Algoritmo generale filtro anti-rumore

Require: $windows\_size > 0$ e $sequence.length \geq windows\_size$
  procedure ANTINOISEFILTER($sequence, windows\_size, A, params...$)
    $i \leftarrow 0$
    $new\_list \leftarrow []$
    while $i \leq sequence.length - windows\_size$ **do**
      $new\_list[i] \leftarrow A(sequence[i : i + windows\_size], params...)$
    end while
    $new\_list \leftarrow new\_list[0] * \lceil \frac{windows\_size}{2} - 1 \rceil + new\_list + new\_list[-1] * \lfloor \frac{windows\_size}{2} \rfloor$
    return new_list
  end procedure

L'*Asymmetrical alpha-trimmed median filter* permette di mantenere una maggiore attinenza alla serie temporale rispetto ad altri algoritmi, senza avere un *overfit* del segnale, cioè riprodurre il segnale senza eliminare il rumore. Il risultato del suo funzionamento è mostrato in figura 3.1. La sequenza filtrata è ottenibile da: $AntiNoiseFilter(sequence, windows\_size, AATMF, \alpha)$

Algorithm 2 Asymmetrical alpha-trimmed median filter

Require: $\alpha > 0$ e $window.length > \alpha$
  procedure AATMF($window, \alpha$)
    $sort(window)$
    $window \leftarrow window[\lfloor \frac{\alpha}{4} \rfloor : window.length - \lceil \frac{3\alpha}{4} \rceil]$
    $filtered\_value \leftarrow sum(window)/(window.length)$
    return $filtered\_value$
  end procedure

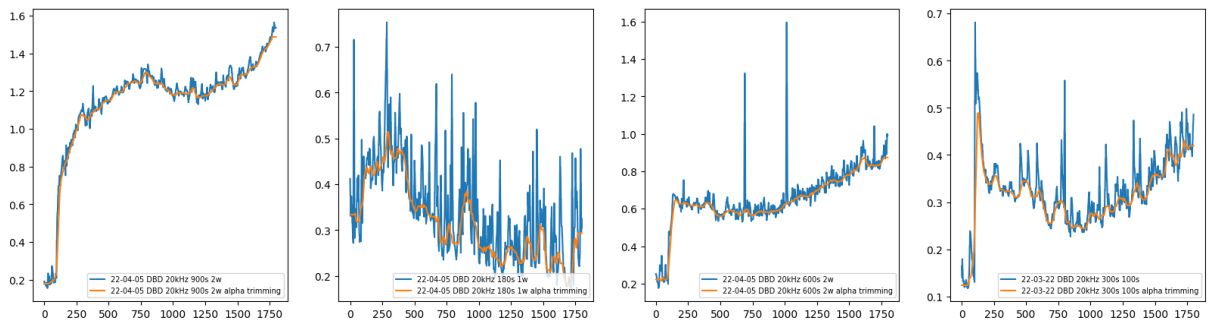


Figura 3.1: Asymmetrical alpha-trimmed median filter applicato ad un campione casuale di 4 elementi del database, con parametri $windows\_size = 60$ e $\alpha = 32$.

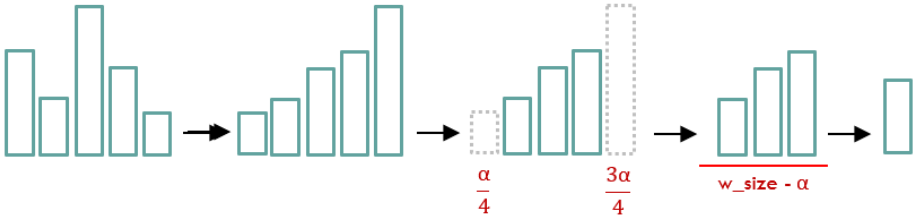


Figura 3.2: Schema del funzionamento dell'algoritmo Asymmetrical alpha-trimmed median filter.

Capitolo 4

Neural networks

4.1 Evoluzione degli algoritmi di *machine learning*

Gli algoritmi di *machine learning* sono strumenti per la risoluzione di problemi complessi che non richiedono una soluzione *hard-coded*. Questi algoritmi generano risultati basandosi su conoscenze acquisite durante la ricerca di pattern nei dati forniti precedentemente al programma [11]. La scelta di come rappresentare questi dati è di fondamentale importanza per l'efficienza e l'accuratezza di questi algoritmi. Inizialmente questi strumenti richiedevano un grande contributo umano nella creazione di un dataset opportuno. Sono stati così introdotti algoritmi che imparano non solo a risolvere il problema ma anche la migliore rappresentazione dei dati per conseguire questo obiettivo. I dati grezzi (non manipolati) possono essere presentati al programma che imparerà prima una loro rappresentazione efficiente, per poi risolvere il problema iniziale. Questo tipo di approccio viene chiamato *representation learning*. Gli algoritmi di *deep learning* appartengono a questa categoria. Essi producono diverse rappresentazioni dei dati che contengono ad ogni iterazione informazioni più astratte, ad esempio per quanto riguarda le immagini ricevendo in input semplici concatenazioni di pixel per arrivare a contorni e in ultima analisi a nomi di oggetti.

Le reti neurali sono modelli di *deep learning* ispirati inizialmente dal funzionamento del cervello umano, poichè ne imitano alcuni processi di elaborazione delle informazioni. Esse possono modificare il loro comportamento in relazione ad una funzione di apprendimento, chiamata *loss function*. Quest'ultima agisce come guida per migliorare la qualità dell'output della rete stessa.

4.2 *Feed-forward neural network*

Le reti neurali *feed-forward* vengono chiamate in questo modo per via del flusso unidirezionale delle informazioni, da *input* verso *output*. Non ci sono quindi meccanismi di *feedback* come nel caso delle reti ricorrenti. I modelli utilizzati durante questa ricerca sono di questa tipologia. Una rete neurale artificiale è definita mediante un algoritmo e dunque una rappresentazione procedurale di una funzione matematica, che puó essere specificata come segue:

$$f_\theta : \mathbb{R}^n \to \mathbb{R}^q \tag{4.1}$$

con n e q, rispettivamente dimensionalità dell'input e dell'output del modello.

Una rete neurale *feed-forward* viene comunemente utilizzata per l'approssimazione di una qualche funzione f^*. Le reti neurali con questo tipo di conformazione topologica risultano quindi particolarmente adatte alla soluzione di problemi di identificazione di modelli predittivi applicati ad esempio per la classificazione oppure la regressione di parametri. Come esempio applicativo si può immaginare una rete configurata per rappresentare un classificatore di immagini rappresentante caratteri numerici. Essa potrebbe quindi essere addestrata per associare un'immagine al numero contenuto al suo interno. Per poter fare questo la rete costruirà, attraverso i suoi parametri interni, una mappa tra gli input (immagini) e output (numeri), scegliendo i migliori parametri che è possibile variare all'interno del modello ($\boldsymbol{\theta}$) che permettono di approssimarla ($f_\theta \approx f^*$). Analogamente questo tipo di modelli può anche essere utilizzato per compiti di regressione, dove quindi l'obiettivo sarà di predire un valore numerico continuo anziché una classe discreta. Un esempio potrebbe essere la stima del prezzo di una casa in base a molteplici caratteristiche quali le sue dimensioni, il numero di stanze, la classe energetica e la posizione geografica.

Questi sono solo due dei possibili utilizzi delle reti neurali. Un terzo aspetto è invece la semplificazione, ovvero la riduzione di dimensionalità, di un problema complesso, partendo da un addestramento non supervisionato. In questo studio sarà trattato nello specifico una metodologia di questo tipo che viene chiamata *autoencoders*; in particolare gli aspetti tecnici relativi saranno trattati approfonditamente nel capitolo 5.

4.2.1 Il processo di apprendimento nelle reti neurali

Partizioniamo l'intero dataset in due insiemi, uno di *training* e uno di *validation*, che useremo come metrica per capire la qualità del nostro modello.

$$\begin{cases} D = D_T \cup D_V \\ D_T \cap D_V = \emptyset \end{cases} \tag{4.2}$$

Ogni elemento dell'insieme sarà una tupla contenente due vettori: $(\boldsymbol{x_i}, \boldsymbol{y_i}) \in D$. Dove $\boldsymbol{x_i}$ rappresenta l'input della rete neurale e $\boldsymbol{y_i}$ l'output desiderato. Vogliamo che il nostro modello approssimi sufficientemente bene questa relazione. Allenare la rete neurale significa risolvere un problema di ottimizzazione dove bisogna trovare la funzione f_θ che minimizzi una quantità, indice dell'errore di approssimazione. Indichiamo con Δ la metrica, cioè una funzione che prende come argomenti l'output del modello $\tilde{\boldsymbol{y}}_{\boldsymbol{i}}$ e quello desiderato $\boldsymbol{y_i}$, restituendo un valore che indichi la loro diversità $l_i = \Delta(\boldsymbol{y_i}, \tilde{\boldsymbol{y}}_{\boldsymbol{i}})$. Allora la quantità da minimizzare sarà data dalla media dei contributi di l_i su tutto il training dataset D_T.

Possiamo descrivere la funzione di costo (*loss function*), che sarà la funzione che la rete neurale minimizzerà per trovare la migliore f_θ, come:

$$\mathcal{L} = \mathbb{E}[\Delta(\boldsymbol{y_i}, f_\theta(\boldsymbol{x_i}))] \tag{4.3}$$

Quindi la funzione f_θ cercata sarà individuabile come:

$$f_\theta = \underset{f_\theta}{\text{argmin}} \quad \mathcal{L}(D_T, f_\theta) = \underset{f_\theta}{\text{argmin}} \quad \mathbb{E}[\Delta(\boldsymbol{y_i}, f_\theta(\boldsymbol{x_i}))] \tag{4.4}$$

Le metriche utilizzabili possono essere svariate e si adattano ai diversi tipi di problemi da ottimizzare. Nel nostro caso, abbiamo fatto uso di quella che viene chiamata errore quadratico medio (*mse*):

$$\Delta = MSE = |\boldsymbol{y_i} - \tilde{\boldsymbol{y}}_{\boldsymbol{i}}|^2 \tag{4.5}$$

Infine definiamo la nostra funzione di costo come:

$$\mathcal{L}_{MSE} = \mathbb{E}[|\boldsymbol{y_i} - \tilde{\boldsymbol{y}}_{\boldsymbol{i}}|^2] = \frac{1}{|D_T|} \sum_{i=1}^{|D_T|} |\boldsymbol{y_i} - \tilde{\boldsymbol{y}}_{\boldsymbol{i}}|^2 \tag{4.6}$$

Quest'ultima è una funzione convessa nella variabile $\tilde{\boldsymbol{y}}_{\boldsymbol{i}}$, ammette cioè un solo minimo che è globale e coincidente con $\tilde{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{y_i}$. La rete neurale, per minimizzare questa quantità, deve esprimerla in funzione delle variabili del modello e quindi di f_θ. Sfortunatamente, in questa formulazione, la *loss function* perde la sua proprietà di convessità, poiché composizione di funzioni che non sono tutte convesse e strettamente crescenti:

$$\mathcal{L}_{MSE} = \frac{1}{|D_T|} \sum_{i=1}^{|D_T|} |\boldsymbol{y_i} - f_\theta(\boldsymbol{x_i})|^2 \tag{4.7}$$

Ciò implicherà alcuni problemi. Infatti la quantità da minimizzare può presentare svariati minimi locali non coincidenti con quello assoluto. Il training potrebbe fermarsi in un punto

molto lontano dall'ottimo. Esistono alcune soluzioni avanzate che sfruttano un fattore chiamato *momentum* che permette, in alcuni casi, di uscire da un minimo locale e continuare l'ottimizzazione di $\mathcal{L}$. Le librerie di *machine learning* gestiscono questo processo autonomamente grazie ad algoritmi di ottimizzazione del training come Adam.

4.2.2 *Perceptron*

Single-neuron perceptron

In *machine learning* l'unità di base che permette lo svolgimento di compiti decisamente complessi viene chiamato neurone o perceptron. Esso puó essere immaginato come un blocco che utilizza gli input per calcolare un opportuno output, infatti esso è semplicemente una funzione che può essere specificata da tre parametri:

1. Un vettore di pesi $\boldsymbol{w} \in \mathbb{R}^d$, dove w_i sarà indice dell'importanza o meno dell'input *i-esimo* nel risultato del neurone.

2. Un valore di bias $b \in \mathbb{R}$.

3. Una funzione di attivazione $\sigma : \mathbb{R} \to \mathbb{R}$. Nel progetto le uniche due utilizzate saranno quella lineare $\sigma(x) = x$ oppure quella chiamata *ReLU* definita come $\sigma(x) = \max(0, x)$. L'originale funzione di attivazione del *perceptron*, così come presentata da Frank Rosenblatt nel paper *perceptron*, utilizzava lo step unitario, poi soppiantato per motivi di efficienza computazionale.

Dato un vettore di input $\boldsymbol{v}$, l'output $o \in \mathbb{R}$ del neurone sarà dato dalla seguente formula:

$$o = \sigma(\boldsymbol{w} \cdot \boldsymbol{v} + b) \tag{4.8}$$

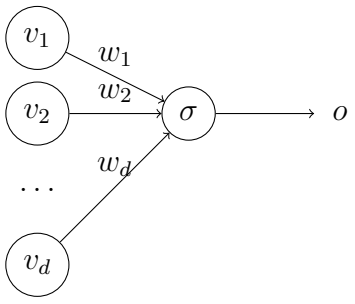


Figura 4.1: Schema di funzionamento di un neurone attraverso un grafo diretto.

Un singolo neurone può svolgere alcuni compiti basilari come regressioni lineari o classificazioni binarie linearmente separabili, ma fallisce nell'imparare la semplice relazione dello XOR

[11], per via della sua non separabilità lineare. Per risolvere questi problemi bisogna aumentare la capacità della rete, ciò aumentare il numero di *perceptron* che la compongono.

Multi-neuron perceptron

Quando utilizziamo m neuroni in parallelo otteniamo un *layer*, cioè un *multi-neuron perceptron*. Considerandolo come un oggetto unico, la sua forma matematica risulta estremamente semplice in notazione matriciale. Per descriverlo necessitiamo dei seguenti parametri:

1. Una matrice di pesi $W \in \mathbb{R}^{m \times d}$, dove la riga *k-esima* sarà il vettore dei pesi del corrispondente neurone.

2. Un vettore di bias $\boldsymbol{b} \in \mathbb{R}^m$, dove l'elemento b_i sarà il bias dell'*i-esimo* neurone.

3. Una funzione di attivazione $\sigma : \mathbb{R}^m \to \mathbb{R}^m$, che applicherà la stessa funzione definita per il singolo neurone ad ogni elemento del dominio.

Dato quindi un vettore di input $\boldsymbol{v}$, allora l'output $\boldsymbol{o} \in \mathbb{R}^m$ del layer sarà dato dalla seguente formula:

$$\boldsymbol{o} = \sigma(W\boldsymbol{v} + \boldsymbol{b}) \tag{4.9}$$

L'utilizzo di più neuroni per ogni singolo layer aumenta la complessità del modello e la sua capacità espressiva. Infatti ogni neurone può focalizzarsi su ristrette parti dell'input invece che sulla sua interezza. Avere più neuroni permette ad esempio di svolgere classificazioni non più binarie, ma su 2^m categorie.

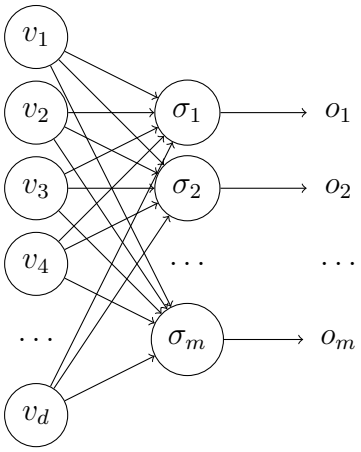


Figura 4.2: Schema di funzionamento di un layer attraverso un grafo diretto.

4.2.3 Deep neural networks (DNN)

Affiancando più layer in serie, cioè utilizzando l'output di uno come input del successivo otteniamo una rete neurale profonda, che è in grado di esprimere funzioni altamente non lineari. L'output del blocco sarà la composizione delle funzioni dei d layer che compongono la rete:

$$\boldsymbol{o} = \sigma^d(W^d\sigma^{d-1}(\dots\sigma^2(W^2\sigma^1(W^1\boldsymbol{v} + \boldsymbol{b^1}) + \boldsymbol{b^2})) + \boldsymbol{b^d}) \tag{4.10}$$

I pesi all'interno dei vari neuroni rappresenteranno le proprietà e *pattern* imparati all'interno di D_T. I parametri costituiscono dunque la "memoria" della rete neurale. La speranza finale è che essa si generalizzi sufficientemente bene ad elementi sconosciuti alla rete neurale come ad esempio D_V.

Se le funzioni di attivazione dei diversi layer fossero tutte lineari la 4.10 risulterebbe equivalente ad una trasformazione affine, rendendo dunque tutta la rete neurale equivalente ad un singolo layer lineare, che può imparare solo funzioni affini. Per questo in *machine learning* è importante aggiungere non-linearità attraverso le funzioni di attivazione.

Esistono prove matematiche che reti neurali con un solo layer nascosto (layer il cui output viene processato da un altro layer), richiedono un numero esponenziale di neuroni in più rispetto alla controparte più profonda [15]. Questo avviene ad un costo, cioè la difficoltà nel training della rete neurale per problemi di *vanishing* o *exploding gradient*, cioè il fatto che i layer a profondità diverse imparano a velocità differenti smettendo di migliorarsi oppure diventando instabili.

Come accennato nell'introduzione, la complessità del problema della chimica del PAW e della risposta biologica nelle piante, ci ha indotto ad utilizzare un approccio di *machine learning* al problema. In particolare sono state utilizzare DNN, poiché molto efficienti nel trovare nessi causali in sistemi altamente non lineari di grandi dimensioni.

Universal approximation theorem

Si può dimostrare che data una funzione $f(\boldsymbol{x})$ continua, esiste sempre una rete neurale con almeno un layer nascosto che approssima la funzione con precisione a piacere $|g(\boldsymbol{x}) - f(\boldsymbol{x})| < \epsilon$, $\forall x \in D_T$ [15]. Intuitivamente la funzione deve essere continua perché composizione di funzioni continue è continua (4.10). La rete neurale descritta dal teorema esiste a patto che il numero di neuroni per layer e la profondita siano appropriate alla funzione da approssimare.

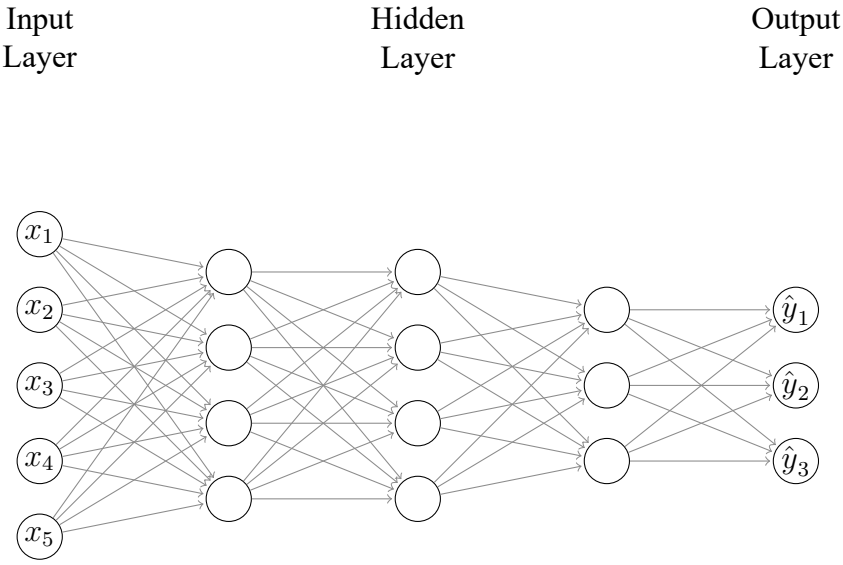


Figura 4.3: Rappresentazione di una *feed-forward deep neural network* tramite grafo diretto.

4.3 *Gradient descent* e *Back-propagation*

La rete neurale nel processo di training, per minimizzare $\mathcal{L}_{MSE}$ deve capire come variare di conseguenza i suoi pesi e bias. Per fare questo dovrà calcolare le derivate parziali della funzione rispetto ai parametri. Questi valori indicheranno la direzione per massimizzare $\mathcal{L}_{MSE}$, la rete aggiornerà quindi i parametri nella direzione opposta. Questo algoritmo, iterato più volte su tutto il D_T prende il nome di *gradient descent*. Possiamo darne la seguente formulazione generale, indicando con $\boldsymbol{\theta}^t$ tutti i parametri della rete neurale (pesi e bias) all'interazione t dell'algoritmo, l'interazione $t+1$ aggiornerà i parametri nel seguente modo:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \alpha \frac{\partial \mathcal{L}(D_T, \boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}} \tag{4.11}$$

Indichiamo con α il *learning rate* del *gradient descent*, un parametro che permette di specificare quanto spostarci nello spazio dove vive $\boldsymbol{\theta}$ in ogni iterazione dell'algoritmo. Questo valore se troppo alto può portare ad uno *overshoot* del minimo locale oppure se troppo basso ad un tempo di convergenza estremamente lungo.

Questo algoritmo è un processo iterativo per la ricerca di un minimo. Visto che la funzione $\mathcal{L}$, come già detto, non è convessa, questa ricerca ci porterà probabilmente ad un minimo locale $\nabla_\theta \mathcal{L}(D_T, \boldsymbol{\theta}^t) = 0$. Esistono algoritmi più complessi che permettono l'uscita da un buco di potenziale della funzione di loss, come Adam. Questo tipo di algoritmi non verranno trattati, ma saranno utilizzati durante la fase di *training*.

Una via alternativa al *gradient descent* è quella di calcolare esplicitamente i parametri che

annullano il gradiente e scegliere quelli che minimizzano $\mathcal{L}$. Avremmo così trovato il minimo assoluto della funzione di *loss*. Questo metodo non è computazionalmente efficiente e bisogna tenere presente che il minimo assoluto potrebbe non essere il punto dove vorremmo fermarci. Infatti quest'ultimo potrebbe essere soggetto a grande *over-fitting*, cioè estrema precisione nell'output di valori $\in D_T$, ma al contempo un'alta MSE per campioni $\in D_V$.

4.3.1 Definizione delle notazione utilizzata

Chiamiamo w^l_{jk} il peso della connessione tra il *k-esimo* neurone del layer $l-1$ verso il *j-esimo* neurone del layer l. Definiamo b^l_j il bias del neurone *j-esimo* nel layer l, mentre con o^l_j l'attivazione (output) del medesimo neurone.

L'output di un neurone nel layer l è intrinsecamente legato a quelli dei layer precedenti dalla seguente equazione:

$$o^l_j = \sigma\left(\sum_k w^l_{jk} o^{l-1}_k + b^l_j\right) \tag{4.12}$$

Possiamo introdurre una notazione matriciale più compatta ed efficiente da calcolare chiamando W^l la matrice dei pesi tra il layer $l-1$ ed il successivo. Come spiegato precedentemente, possiamo inserire in vettori gli output e bias dei due layer, dove l'*i-esimo* elemento corrisponderà al valore corrispondente del neurone i. Utilizzando la convezione che la funzione di attivazione σ operi *element-wise* su ogni elemento del vettore di input, allora possiamo scrivere che:

$$\boldsymbol{o}^l = \sigma(W^l \boldsymbol{o}^{l-1} + \boldsymbol{b}^l) \tag{4.13}$$

Un'altra quantità importante da considerare è il valore intermedio $\boldsymbol{k}^l = W^l \boldsymbol{o}^{l-1} + \boldsymbol{b}^l$, che è la quantità calcolata internamente dal neurone e data in input alla funzione di attivazione σ.

Con questa notazione possiamo riscrivere la funzione da ottimizzare come:

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x_i} - \boldsymbol{o}^L(\boldsymbol{x_i})|^2 \tag{4.14}$$

Dove abbiamo indicato $\boldsymbol{o}^L(\boldsymbol{x_i})$ il vettore di output della rete (con L layer) quando il suo input è $\boldsymbol{x_i}$.

L'ultima variabile da introdurre è δ^l_j che descrive l'errore del *j-esimo* neurone, nel layer l. Essa viene definita come:

$$\delta^l_j = \frac{\partial \mathcal{L}_{MSE}}{\partial k^l_j} \tag{4.15}$$

Il termine δ_j^l rappresenta come la *loss function* è sensibile ad cambiamento di k_j^l. Se il suo valore è vicino a zero significa che il neurone è arrivato ad un punto di relativa stabilità.

Ora sviluppiamo le equazioni che permetteranno di aggiornare i parametri del modello tramite *gradient descent*.

4.3.2 Errore δ^L nell'output layer

Partendo dalla definizione di δ_j^L possiamo utilizzare la *chain rule* per la derivata di una funzione composta, andando a riscrivere il valore come:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial k_j^l} = \frac{\partial \mathcal{L}}{\partial o_j^L} \cdot \frac{\partial o_j^L}{\partial k_j^L} \tag{4.16}$$

Riconosciamo nel secondo termine la derivata della funzione di attivazione di quel neurone perché definita come $o_j^L = \sigma(k_j^L)$. Possiamo riscrivere l'errore come:

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial o_j^L} \cdot \frac{\partial o_j^L}{\partial k_j^L} = \frac{\partial \mathcal{L}}{\partial o_j^L} \cdot \sigma'(k_j^L) \tag{4.17}$$

Utilizzando MSE come loss function possiamo calcolare esplicitamente il primo termine, infatti: $\frac{\partial \mathcal{L}_{MSE}}{\partial o_j^L} = 2(o_j^L - x_j)$

È possibile portarsi in notazione matriciale definendo $\nabla_{\boldsymbol{o}^L} \mathcal{L}$ il vettore che ha per componenti le derivate parziali $\frac{\partial \mathcal{L}}{\partial o_j^L}$. Allora l'errore per il layer L prende la forma:

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{o}^L} \mathcal{L} \odot \sigma'(\boldsymbol{k}^L) \tag{4.18}$$

Dove l'operatore $\odot$ esegue una moltiplicazione tra i due vettori *element-wise*. Considerando la funzione di *loss* una MSE possiamo semplificare l'espressione fino a farla diventare:

$$\boldsymbol{\delta}^L = 2(\boldsymbol{o}^L - \boldsymbol{x}) \odot \sigma'(\boldsymbol{k}^L) \tag{4.19}$$

4.3.3 Errore δ^l in funzione di δ^{l+1}

L'intento è scrivere δ_j^l in funzione di δ_k^{l+1}. Per fare ciò utilizziamo nuovamente la *chain rule*:

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial k_j^l} = \sum_i \frac{\partial \mathcal{L}}{\partial k_i^{l+1}} \cdot \frac{\partial k_i^{l+1}}{\partial k_j^l} \tag{4.20}$$

Notiamo che il primo termine non è altro che l'errore *i-esimo* del layer $l+1$, allora possiamo riscrivere l'espressione come:

$$\delta_j^l = \sum_i \frac{\partial k_i^{l+1}}{\partial k_j^l} \delta_i^{l+1} \tag{4.21}$$

Per semplificare il primo termine ricordiamoci la definizione di k_i^{l+1}:

$$k_i^{l+1} = \sum_j w_{ij}^{l+1} o_j^l + b_i^{l+1} = \sum_j w_{ij}^{l+1} \sigma(k_j^l) + b_i^{l+1} \tag{4.22}$$

Differenziando l'equazione otteniamo:

$$\frac{\partial k_i^{l+1}}{\partial k_j^l} = w_{ij}^{l+1} \sigma'(k_j^l) \tag{4.23}$$

Sostituendo dentro 4.21 risulta:

$$\delta_j^l = \sum_i w_{ij}^{l+1} \delta_i^{l+1} \sigma'(k_j^l) \tag{4.24}$$

Possiamo riscriverla in forma vettoriale molto semplicemente, come segue:

$$\boldsymbol{\delta}^l = ((W^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{k}^l) \tag{4.25}$$

Utilizzando le due equazioni ricavate (4.19 e 4.25) possiamo ottenere il vettore di errore per qualsiasi layer della rete neurale semplicemente applicando le equazioni in maniera ricorsiva.

4.3.4 Derivata parziale di $\mathcal{L}$ rispetto ad ogni parametro

Rispetto ad un generico bias

Per ottenere la velocità di cambiamento di $\mathcal{L}$ rispetto ad un bias qualsiasi nella rete neurale b_j^l possiamo procedere come segue:

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \frac{\partial \mathcal{L}}{\partial k_j^l} \cdot \frac{\partial k_j^l}{\partial b_j^l} = \frac{\partial k_j^l}{\partial b_j^l} \delta_j^l \tag{4.26}$$

Ricordando la definizione di $k_j^l = \sum_i w_{ji}^l o_i^{l-1} + b_j^l$, la sua derivata rispetto a b_j^l è pari a 1. Dunque possiamo semplificare l'equazione 4.26 in:

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \tag{4.27}$$

Rispetto ad un generico peso

Per calcolare la derivata parziale di $\mathcal{L}$ rispetto ad ogni peso, procediamo in modo similare a quanto fatto con i bias:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^l} = \frac{\partial \mathcal{L}}{\partial k_j^l} \cdot \frac{\partial k_j^l}{\partial w_{ji}^l} = \frac{\partial k_j^l}{\partial w_{ji}^l} \delta_j^l \tag{4.28}$$

Ricordando la definizione di $k_j^l = \sum_h w_{jh}^l o_h^{l-1} + b_j^l$, la sua derivata rispetto a w_{ji}^l è pari a o_i^{l-1}. Dunque possiamo semplificare l'equazione 4.28 in:

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^l} = o_i^{l-1} \delta_j^l \tag{4.29}$$

Se $o_i^{l-1} \approx 0$ il neurone imparerà molto lentamente siccome i suoi pesi non cambieranno particolarmente durante il training.

4.3.5 *Stochastic gradient descent*

Abbiamo descritto come la tecnica della *back-propagation* funziona nel calcolare il gradiente di $\mathcal{L}$ rispetto ad un singolo campione $\in D_T$. In realtà, per velocizzare il processo di training si partiziona D_T in insiemi costituiti circa da m elementi (*batch-size*) chiamati *mini-batch* (D_m). L'algoritmo di *stochastic gradient descent* calcolerà il gradiente per ognuno di questi insiemi, aggiornando di volta in volta i parametri. L'algoritmo verrà ripetuto T volte, dove T prende il nome di *epochs*. L'algoritmo è mostrato nello *snippet* 3.

4.4 *Convolutional neural network*

Una rete neurale convoluzionale è un tipo di rete neurale che presenta almeno un layer che svolge un'operazione di convoluzione.

4.4.1 Convoluzione 1D

Considerando due vettori $\boldsymbol{v} \in \mathbb{R}^d$ e $\boldsymbol{g} \in \mathbb{R}^q$ (chiamato abitualmente *kernel* o filtro), con $d \geq q$, definiamo l'operatore lineare di convoluzione come:

$$\boldsymbol{o} : \mathbb{R}^d \times \mathbb{R}^q \to \mathbb{R}^{d-q+1} \tag{4.30}$$

$$\boldsymbol{o}[i] = (\boldsymbol{v} * \boldsymbol{g})[i] = \sum_{j=1}^{q} \boldsymbol{v}[i+q-j] \cdot \boldsymbol{g}[j] \tag{4.31}$$

Algorithm 3 Stochastic Gradient Descent

for t in $\{1, 2, \ldots, T\}$ **do**
  for D_m in D_T **do**
    for $\boldsymbol{x}$ in D_m **do**
      for l in $\{2, 3, \ldots, L\}$ **do** ▷ Forward pass
        $\boldsymbol{k}^{\boldsymbol{x},l} = W^l \boldsymbol{o}^{\boldsymbol{x},l-1} + \boldsymbol{b}^l$
        $\boldsymbol{o}^{\boldsymbol{x},l} = \sigma(\boldsymbol{k}^{\boldsymbol{x},l})$
      end for
      $\boldsymbol{\delta}^{\boldsymbol{x},L} = \nabla_{\boldsymbol{o}^L} \mathcal{L} \odot \sigma'(\boldsymbol{k}^{\boldsymbol{x},L})$ ▷ Output error
      for l in $\{L-1, L-2, \ldots, 2\}$ **do** ▷ Backpropagation
        $\boldsymbol{\delta}^{\boldsymbol{x},l} = ((W^{l+1})^T \boldsymbol{\delta}^{\boldsymbol{x},l+1}) \odot \sigma'(\boldsymbol{k}^{\boldsymbol{x},l})$
      end for
    end for
    for l in $\{L-1, L-2, \ldots, 2\}$ **do** ▷ Gradient descent
      $W^l \leftarrow W^l - \frac{\alpha}{m} \sum_x \boldsymbol{\delta}^{\boldsymbol{x},l} (\boldsymbol{o}^{\boldsymbol{x},l-1})^T$
      $\boldsymbol{b}^l \leftarrow \boldsymbol{b}^l - \frac{\alpha}{m} \sum_x \boldsymbol{\delta}^{\boldsymbol{x},l}$
    end for
  end for
end for

L'operazione semplicemente prende la versione speculare del vettore $\boldsymbol{g}$, lo moltiplica con il vettore $\boldsymbol{v}$ e ne somma le componenti. Il filtro sarà poi fatto scorrere lungo tutto il vettore $\boldsymbol{v}$, fino a raggiungerne il limite. Proprio per la sua linearità, si potrebbe definire una opportuna matrice $W \in \mathbb{R}^{(d-q+1)\times d}$ per calcolare questa operazione in maniera efficiente. Nonostante l'apparente elevato numero di parametri, quelli veramente indipendenti risultano essere q. Vista in forma matriciale, l'operazione di convoluzione può essere descritta in questo modo: $\boldsymbol{o} = W\boldsymbol{v}$.

4.4.2 Layer convoluzionale

In un layer convoluzionale il vettore $\boldsymbol{v}$ rappresenta il suo input. Inoltre per aggiungere un ulteriore fattore di non-linearità il prodotto dalla convoluzione sarà applicata *element-wise* una funzione di attivazione σ (es. ReLu), proprio come in un layer denso. Verrà aggiunto un parametro di bias b che verrà ripetuto per raggiungere la dimensione di un vettore $\in \mathbb{R}^{d-q+1}$

Se il layer ha un singolo filtro possiamo rappresentare il suo output come:

$$\boldsymbol{o} = \sigma(W\boldsymbol{v} + \boldsymbol{b}) \tag{4.32}$$

Nelle reti neurali complesse è comune trovare più filtri per ogni layer. Ciò significa semplicemente che l'operazione di convoluzione definita prima, viene ripetuta più volte con filtri diversi. Introducendo k filtri nel layer, l'output sarà dunque una matrice $O \in \mathbb{R}^{(d-q+1)\times k}$. Ciò permette ad ogni filtro di specializzarsi nel riconoscimento di particolari *features* negli input.

4.4.3 Parametri aggiuntivi ai layer convoluzionali

Padding

Il padding viene utilizzato per variare la dimensione dell'output del layer convoluzionale. Vengono aggiunti tanti zeri quanti quelli specificati dal parametro in modo tale da aumentare la dimensione dell'output. Nelle librerie di *machine learning* spesso si utilizza la parola *"padding same"* in modo da aggiungere un padding sufficiente per mantenere la dimensione dell'input anche nell'output.

Strides

Abbiamo detto che il filtro si muove di posizione in posizione all'interno del vettore di input. Il parametro di *stride* permette di specificare la dimensione dello *step* con cui muovermi all'interno del vettore di volta in volta. Il parametro è molto utile per creare un *down-sample* della sequenza iniziale, catturando contemporaneamente le caratteristiche spaziali del segnale più importanti.

Dimensione finale dell'output

Si può dimostrare che applicando questi parametri ad un input, con p il valore di padding e s il valore di stride, la dimensione dell'output per ogni singolo filtro sarà:

$$dim_o = \lfloor \frac{d + 2p - q}{s} \rfloor + 1 \tag{4.33}$$

4.4.4 Motivazione dell'introduzione di layer convoluzionali

I layer convoluzionali riescono a riconoscere dei patter spaziali per via del loro funzionamento. Il punto di forza dei layer convoluzionali, rappresenta anche la sua differenza con un layer denso. Infatti la condivisione del kernel all'interno di tutta la sequenza produce alcune importanti caratteristiche [21]:

1. I pattern trovati dai filtri risultano invarianti alle traslazioni, quindi individuabili lungo tutta la serie temporale.

2. Inserendo più layer convoluzionali in serie, attraverso uno strides riusciamo a ricavare pattern sempre più complessi e a lungo termine. Infatti i neuroni finali della sequenza saranno influenzati dalla maggior parte del vettore di input.

3. Il numero di parametri risulta ridotto rispetto a layer completamente densi. Ciò permette lo sviluppo di algoritmi più efficienti con sostanziale riduzione di dimensione e tempo di training.

Facendo seguire a dei layer convoluzionali alcuni densi, riusciamo ad utilizzare le *features* spaziali estratte dai layer convoluzionali e mapparle nell'output. È prassi prendere gli output delle convoluzione nella forma matriciale e appiattirli, affiancando ogni riga per ottenere un vettore da dare come input a quelli densi. In questo modo si elimina la relazione spaziale originale. Nonostante ciò i layer densi possono lavorare in maniera indipendente da essa poichè ogni input ha il suo peso individuale all'interno di un neurone.

Layer convoluzionali interpretati come layer densi con *prior* infinitamente forte

Come abbiamo visto dall'equazione 4.32, un layer convoluzionale può essere interpretato come un layer denso con un numero di neuroni dato dall'equazione 4.33. Una *prior* (distribuzione di probabilità che indica le nostre ipotesi iniziali nei confronti di un evento) infinitamente forte, forza i valori di alcuni parametri a seguire alcune regole determinate [11]. In questo caso, i pesi di un neurone devono essere uguali a quello del suo vicino ma traslati nello spazio. Stiamo cioè assumendo delle relazioni spaziali tra i valori di input. Ovviamente questa imposizione può portare a *under-fitting* nel caso in cui la nostra ipotesi non sia sufficientemente accurata, cioè quando non esistono evidenti relazioni spaziali nei dati. Nonostante ciò immagini e serie temporali presentano ragionevolmente questa caratteristica.

Capitolo 5

Autoencoders

5.1 Introduzione agli autoencoders

Gli autoencoders sono reti neurali usate principalmente per *learning* non supervisionato. Questo significa che il dataset D non conterrà label indicanti l'output desiderato $\boldsymbol{y_i}$. Il loro scopo è quello di imparare una rappresentazione significativa dei dati di partenza, tramite la minimizzazione della *loss function* tra l'output del modello e la sequenza iniziale [2]. Il modello verrà allenato per rappresentare più fedelmente possibile i dati di *training*.

Introduciamo un generico dataset composto da M valori senza label:

$$D = \{\boldsymbol{x_i}, \forall i \in \{0, 1, \ldots, M\}\} \tag{5.1}$$

Il vettore di input è definito come $\boldsymbol{x_i} \in \mathbb{R}^n$, $n \in \mathbb{N}$.

Come le altre reti neurali, anche l'autoencoder è un algoritmo e quindi una funzione. Questa prospettiva ci permette di darne una formulazione matematica rigorosa. Gli autoencoder sono formati da tre parti principali: un *encoder*, uno spazio latente e un *decoder*.

Un encoder è una funzione g così definita:

$$g : \mathbb{R}^n \to \mathbb{R}^q \tag{5.2}$$

Lo spazio latente è uno spazio vettoriale definito su $\mathbb{R}^q$. Mentre la rappresentazione del segnale *i-esimo* nello spazio latente $\boldsymbol{z_i} \in \mathbb{R}^q$ è un vettore così calcolato:

$$\boldsymbol{z_i} = g(\boldsymbol{x_i}) \tag{5.3}$$

Il decoder svolge il compito opposto rispetto all'encoder, infatti ha come dominio lo spazio

latente, che viene mappato in un vettore con dimensionalità pari a quella del segnale originale:

$$f : \mathbb{R}^q \to \mathbb{R}^n \tag{5.4}$$

In particolare definiamo il segnale ricostruito come:

$$\tilde{\boldsymbol{x_i}} = f(\boldsymbol{z_i}) = f(g(\boldsymbol{x_i})) \tag{5.5}$$

Obiettivo dell'autoencoder sarà dunque trovare le due funzioni che minimizzano la metrica Δ scelta, tipicamente MSE:

$$\underset{f,g}{\text{argmin}} \quad \mathbb{E}[\Delta(\boldsymbol{x_i}, f(g(\boldsymbol{x_i})))] \tag{5.6}$$

5.1.1 *Undercomplete autoencoder*

Ricordando la definizione, l'autoencoder deve imparare una rappresentazione informativa delle osservazioni. Se $q \geq n$, ciò non sarà possibile poiché il modello potrà ricostruire perfettamente l'osservazione iniziale: $\boldsymbol{x_i} = f(g(\boldsymbol{x_i}))$, infatti la rete neurale dovrà imparare semplicemente la funzione identità, mappando nello spazio latente il segnale originale per poi ricavarlo senza perdita. Questa complicazione deriva direttamente dall'*universal approximator theorem* delle reti neurali [11].

Un modo per far imparare all'autoencoder delle rappresentazioni significative è quello di limitare la dimensione dello spazio latente a $q < n$. Questo tipo di rete neurale prende il nome di *undercomplete autoencoder*. La dimensione dello spazio latente è importante per apprendere rappresentazioni significative di D ma risulta fondamentale limitare anche la capacità dell'autoencoder (il numero di parametri che descrivono f e g), altrimenti l'algoritmo potrebbe imparare a ricostruire il segnale semplicemente dal suo indice i [11].

Generalmente il campione ricostruito dall'autoencoder non è interessante come la sua rappresentazione nello spazio latente. Gli utilizzi principali di un autoencoder sono algoritmi di compressione *lossy* o di *data augmentation*. Noi lo utilizzeremo per assegnare delle *label* ai campioni di partenza.

5.1.2 Gli autoencoders imparano una varietà (*manifold*) nello spazio $\mathbb{R}^n$

Definiamo informalmente una varietà come un insieme di punti $\in \mathbb{R}^n$ connesso, che è descrivibile utilizzando un numero inferiore di gradi di libertà [11]. In *machine learning* si è avanzata l'ipotesi che la dimensionalità di molti dataset è alta in modo superflue e che essi possono essere rappresentati, con buona approssimazione, in termini di una varietà che vive nello spazio

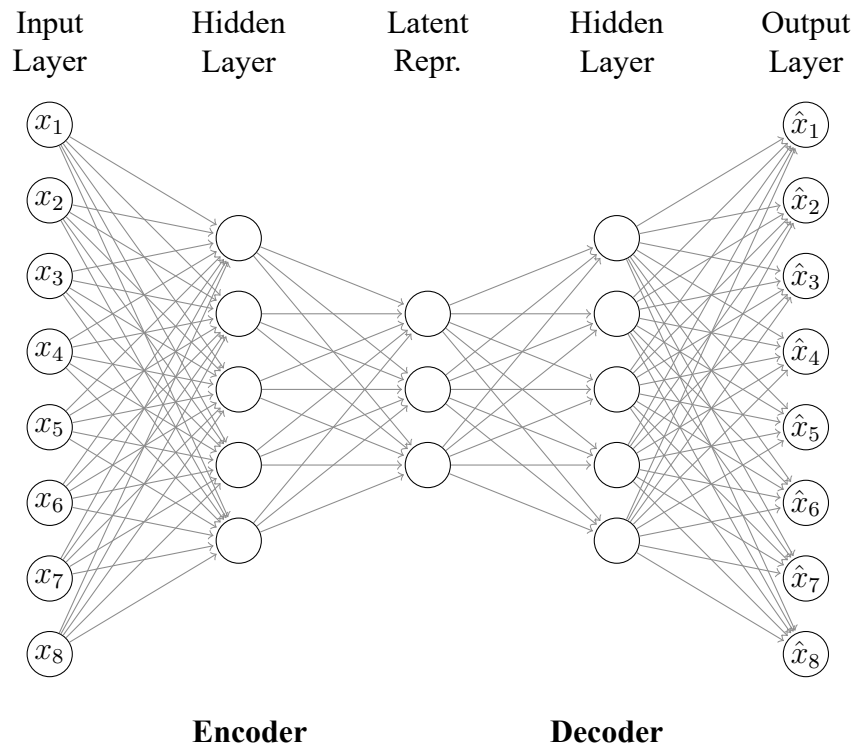


Figura 5.1: Rappresentazione di un *undercomplete* autoencoder tramite un grafo diretto.

del dataset [6]. Moltissimi algoritmi di *machine learning* sfruttano questa ipotesi, considerando inutili la maggior parte degli input di $\mathbb{R}^n$ e concentratosi solo su un loro sottoinsieme. Nel nostro caso non ci interessa considerare qualsiasi vettore $\in \mathbb{R}^{1800}$, ma solo quelli campionati da una distribuzione di probabilità che descrive correttamente il fenomeno fisico sottostante. Un autoencoder cerca di imparare implicitamente un sistema di coordinate locali per la varietà che meglio si adatta ai campioni di D_T [11]. Il vettore $\boldsymbol{z_i}$ rappresenta infatti le coordinate della proiezione di $\boldsymbol{x_i}$ sulla varietà imparata.

Siccome $\boldsymbol{z_i}$ è la proiezione del segnale sulla varietà, essa sarà sensibile solamente alle variazioni sull'iperpiano tangente al *manifold* (π_i) nell'intorno di $\boldsymbol{x_i}$, mentre non presterà attenzione alle variazioni sullo spazio ortogonale, perché ritenute poco informative. Matematicamente $\boldsymbol{z_i} = g(\boldsymbol{x_i}) = g(\boldsymbol{x_i^{\perp}} + \boldsymbol{x_i^{//}}) = g(\boldsymbol{x_i^{//}})$. Bisogna tener presente che se la varietà non risulta particolarmente regolare, l'autoencoder avrà bisogno di una cardinalità molto elevata di D_T per impararne la forma, altrimenti produrrà risultati poveri nella generalizzazione a variazioni mai viste.

Autoencoder lineari (che hanno funzioni di attivazioni lineari) convergono ad un risultato simile a quello di una PCA (*principal component analysis*) se presentano una *loss function* MSE. Infatti nel *training* dell'autoencoder vogliamo trovare le basi di uno spazio vettoriale, che definiscono le coordinate dello spazio latente, dove sono minimizzati i quadrati delle distanze tra le

proiezioni e i dati stessi, esattamente quello che PCA svolge. A differenza di PCA, non abbiamo il *constraint* di basi ortonormali e ordinate in ordine di varianza decrescente. Nonostante ciò, lo spazio generato dalle basi risulta uguale a quello di una PCA [16].

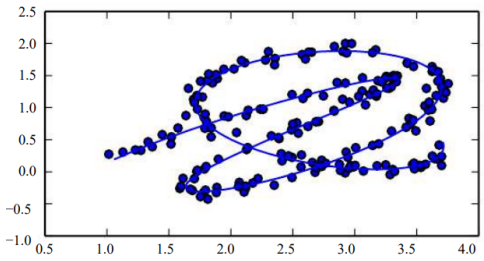


Figura 5.2: Dati in $\mathbb{R}^2$ campionati da una distribuzione concentrata vicino a una varietà unodimensionale. Immagine presa da [11].

5.1.3 Convoluzione trasposta negli autoencoder convoluzionali

Spesso negli autoencoder vengono utilizzati layer convoluzionali, il che presenta a prima vista un problema. Visto la simmetria tra encoder e decoder bisogna definire un layer che svolge una operazione opposta rispetto alla convoluzione. Viene introdotta quindi l'operazione di convoluzione trasposta da utilizzare all'interno del decoder, per recuperare la dimensione dell'input. Questo operatore non è l'inverso matematico della funzione di convoluzione, infatti componendo le due funzioni non otterremmo l'identità. Nonostante ciò, questo non è un grande problema siccome le reti neurali riescono ad aggiustare i parametri per ottenere l'output più utile, l'importante è recuperare la dimensionalità dell'input. Per calcolare la convoluzione trasposta possiamo semplicemente prendere la matrice trasposta della convoluzione e moltiplicarla per l'input del layer.

5.2 *Variational* autoencoder

A differenza degli autoencoders visti finora, i *variational* autoencoders sono modelli probabilistici invece che deterministici. Infatti dato un campione $\boldsymbol{x_i}$, la sua rappresentazione latente sarà un vettore $\boldsymbol{z_i}$ campionato da una particolare distribuzione di probabilità. Questo avrà una serie di benefici che provengono dalla nuova struttura che assumerà il latent space.

Assumiamo che i nostri dati vengano campionati da un processo sconosciuto, con distribuzione di probabilità $p^*(\boldsymbol{x})$. Noi cercheremo di trovare una distribuzione che la approssimi al meglio $p_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx p^*(\boldsymbol{x})$.

Una variabile latente è una variabile che è parte del modello ma non viene osservata direttamente e dunque non appartiene al dataset D. Ad esempio una immagine potrebbe avere come

variabile latente l'oggetto in essa presente, ma non viene in alcun modo associata al processo di misura del campione.

Definiamo la distribuzione marginale dei campioni come:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z})\, d\boldsymbol{z} \tag{5.7}$$

L'obiettivo finale sarebbe quello di massimizzare:

$$p_{\boldsymbol{\theta}}(D_T) = \prod_{i=1}^{|D_T|} p_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \tag{5.8}$$

in modo da rendere più probabile la generazione di campioni provenienti dal dataset da parte del modello. Sfortunatamente questa quantità viene definita *intractable*, non riusciamo cioè ad ottenerne una forma chiusa, utile per calcolarne il gradiente per la *back-propagation*.

La condizione di non trattabilità di $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ si ripercuote anche su $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.

Per ovviare a questo problema, nel *variational* autoencoder chiamiamo encoder la distribuzione di probabilità $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ che cercheremo di ottimizzare in modo tale da avere: $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. Questa approssimazione, che permette la definizione di una forma chiusa dell'encoder, ci consentirà di ottimizzare la distribuzione marginale 5.8.

5.2.1 *Evidence lower bound*: ELBO

In teoria della probabilità viene spesso usata una funzione chiamata divergenza di Kullback-Leibler (D_{KL}), come metrica per la distanza di due distribuzioni di probabilità. Essa è definita come segue:

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}\left[\log \frac{P(x)}{Q(x)}\right] \tag{5.9}$$

Possiamo iniziare a ricavare la funzione di *loss*:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \tag{5.10}$$

$$= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{5.11}$$

$$= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] + \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{5.12}$$

Chiamiamo il primo termine ELBO e lo indichiamo con $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$, mentre il secondo notiamo che rappresenta la divergenza KL tra $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ e la sua approssimazione $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) + D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \tag{5.13}$$

Riscriviamo allora l'ELBO e cerchiamo di darne un significato:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \tag{5.14}$$

Questa funzione rappresenta un *lower-bound* al primo termine (chiamato *log-likelihood*) poichè per definizione il termine $D_{KL} \geq 0$, visto che rappresenta una distanza. Massimizzando 5.14 otteniamo due importanti risultati:

1. Massimizziamo la distribuzione marginale $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, facendo generare al modello campioni più attinenti a D_T.

2. Minimizziamo D_{KL} cioè la distanza tra le la mia approssimazione e $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.

Ricordandoci la definizione della funzione di ELBO, possiamo continuare la manipolazione per arrivare ad una forma più utile al training dell'autoencoder:

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && (5.15)\\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\right] && (5.16)\\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] && (5.17)\\
&= \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z})) && (5.18)
\end{aligned}$$

In questa formulazione del tutto equivalente non troviamo più i termini *intractables* che non permettevano una formulazione chiusa della funzione. Il primo termine prende il nome di *reconstruction likelihood*. Il secondo termine, nel contesto delle *loss functions*, viene chiamato termine di regolarizzazione che impone alla approssimazione $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ una forma simile a $p_{\boldsymbol{\theta}}(\boldsymbol{z})$.

5.2.2 *Variational family* usate da VAE

Come già detto la distribuzione $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ permette una formulazione chiusa del modello. Per ottenere ciò dobbiamo imporre una classe di appartenenza per q. Spesso viene presa la famiglia delle gaussiane così definita:

$$\mathcal{Q} = \{\mathcal{N}(h^1_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(h^2_{\boldsymbol{\phi}}(\boldsymbol{x})))\} \tag{5.19}$$

Dove le funzioni deterministiche $h^1_{\boldsymbol{\phi}}(\boldsymbol{x})$ e $h^2_{\boldsymbol{\phi}}(\boldsymbol{x})$ mappano l'osservazione al vettore media e alla matrice di covarianza (diagonale in questo caso). Imponendo l'appartenenza di q all'insieme, stiamo vincolando la sua forma alla funzione del tipo:

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(h^1_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(h^2_{\boldsymbol{\phi}}(\boldsymbol{x}))) \tag{5.20}$$

Dunque la rappresentazione latente del vettore $\boldsymbol{x}$ risulta:

$$\boldsymbol{z} \sim \mathcal{N}(h^1_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(h^2_{\boldsymbol{\phi}}(\boldsymbol{x}))) \tag{5.21}$$

5.2.3 *Prior*

Definiamo *prior* la distribuzione $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. La forma di questa distribuzione non è molto importante perché il decoder può utilizzare i primi layer come mappa tra lo spazio campionato e qualsiasi rappresentazione latente utile al modello [10].

La funzione più utilizzata e anche più semplice, che non necessità di parametri da imparare, è una distribuzione normale $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(0, I)$, con I la matrice identità di dimensione pari a quella dello spazio latente.

Quando sia *prior* che *conditional posterior* ($q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$) sono gaussiane, il termine $D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}))$ assume un valore estremamente semplice [10]:

$$\begin{aligned} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z})) &= D_{KL}(\mathcal{N}(h^1_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(h^2_{\boldsymbol{\phi}}(\boldsymbol{x})))\|\mathcal{N}(0, I)) && (5.22)\\ &= D_{KL}(\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma^2})\|\mathcal{N}(0, I))) && (5.23)\\ &= \frac{1}{2}\sum_{i=1}^{k}(\mu_i^2 + \sigma_i^2 - 1 - \log\sigma_i^2) && (5.24) \end{aligned}$$

5.2.4 Struttura del latent space

Il termine di regolarizzazione del latent space è fondamentale per avere le proprietà di completezza e continuità. Con completezza intendiamo la proprietà che garantisce di ottenere una osservazione plausibile quando campioniamo la *prior* $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. La proprietà di continuità garantisce che presi due punti relativamente vicini nel *latent space* la loro ricostruzione sarà simile. Questo rende il *variational* autoencoder adatto a generare nuovi elementi simili al dataset. Se non fosse presente il termine di regolarizzazione, il modello collasserebbe ad un autoencoder normale. Infatti potrebbe ridurre fortemente la varianza generata fino ad essere assimilabile ad un singolo punto, oppure potrebbe generare distribuzioni con medie molto distanti, non permettendo il campionamento dello spazio latente.

Nonostante la sua importanza se il termine di ricostruzione venisse a mancare non vedremo *cluster* nel latent space ma una perfetta gaussiana, con un modello avente una pessima qualità di ricostruzione. Bilanciare correttamente questi due termini è fondamentale per avere un autoencoder efficiente e sarà proprio l'obiettivo del *β-VAE*, che verrà discusso in uno dei prossimi paragrafi.

Figura 5.3: (a) Latent space 2D senza regolarizzazione (b) Latent space 2D con regolarizzazione. Immagine presa da [18].

5.2.5 Decoder probabilistico

Una volta campionata la distribuzione $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ e ottenuto così la rappresentazione latente di $\boldsymbol{x}$ posso generare un nuovo campione $\boldsymbol{x'} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. In linea di principio anche il decoder è una distribuzione di probabilità. Viene spesso imposta $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(f_{\boldsymbol{\theta}}(\boldsymbol{z}), \sigma_{decoder} I)$, dove $\sigma_{decoder}$ è il rumore gaussiano attorno alla media $f_{\boldsymbol{\theta}}(\boldsymbol{z})$, output effettivo del decoder. Per ottenere $\boldsymbol{x'}$ posso campionare la distribuzione oppure, più comunemente, utilizzare la media come valore ricostruito.

Ricordandosi l'espressione della distribuzione gaussiana:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \frac{1}{\sqrt{(2\pi\sigma^2_{decoder})}} \prod_{i=1}^{n} \exp\left(-\frac{1}{2\sigma^2_{decoder}}(x_i - \mu_i)^2\right) \tag{5.25}$$

è immediato vedere che la *log likelihood* può essere espressa come:

$$\mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{\sqrt{2\pi\sigma^2_{decoder}}} - \frac{|\boldsymbol{x} - f_\theta(\boldsymbol{z})|^2}{2\sigma^2_{decoder}}\right] \tag{5.26}$$

La formulazione è molto simile a quella di MSE a meno di due costanti. Utilizzeremo quindi la seguente *loss function*:

$$\mathcal{L}_{\boldsymbol{\theta},\phi}(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[|\boldsymbol{x} - f_\theta(\boldsymbol{z})|^2] - D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z})) \tag{5.27}$$

dove è subito evidente un termine di ricostruzione e uno di regolarizzazione.

5.2.6 *Reparameterization trick*

Con questa formulazione il processo di *back propagation* è impossibile per via della natura stocastica dell'operazione di campionamento del vettore $\boldsymbol{z} \sim \mathcal{N}(h^1_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(h^2_{\boldsymbol{\phi}}(\boldsymbol{x})))$. Possiamo applicare un *trick* matematico per separare la parte stocastica da quella deterministica e poter così calcolare il gradiente della *loss function*. Facciamo il campionamento di una normale standard $\epsilon \sim \mathcal{N}(0, I)$ ed esprimiamo il vettore $\boldsymbol{z}$ come:

$$\boldsymbol{z} = h^1_{\boldsymbol{\phi}}(\boldsymbol{x}) + \sqrt{h^2_{\boldsymbol{\phi}}(\boldsymbol{x})} \odot \epsilon \tag{5.28}$$

L'operatore $\odot$ esegue una moltiplicazione *element-wise* tra i due vettori. L'equazione così definita risulta una operazione deterministica e quindi differenziabile.

5.2.7 *β-VAE*

Un β-VAE è una modifica di un *variational autoencoder* che introduce un *hyperparameter* β, che può aumentare o diminuire l'importanza del termine di D_{KL}:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \beta D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})) \tag{5.29}$$

La spinta che $\beta > 1$ introduce nella *posterior* $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, porta alla creazione di uno spazio latente più strutturato a discapito della capacità di ricostruzione dell'autoencoder. Infatti per aumentare l'informazione che Z può ricostruire bisognerebbe disperdere le *posterior* dei vari campioni aumentando la distanza delle medie oppure diminuendone la varianza. Entrambe queste soluzioni porterebbero ad un aumento del termine D_{KL}. La ricostruzione con questo termine di regolarizzazione spinge elementi vicini nello spazio dei dati ad essere vicini anche in quello latente. Infatti avendo distribuzioni vicine nello spazio latente, al momento del sampling, qual'ora tramite il criterio di *maximum likelihood* scambiassi una distribuzione per l'altra, la *reconstruction loss* sarà minimizzata perché i due campioni iniziali erano simili. Chiamiamo questa capacità, proprietà di località [5].

5.2.8 *Disentangling*

Una rappresentazione *disentangled* si ottiene quando singole dimensioni dello spazio latente codificano differenze in un solo fattore generativo dei campioni [5]. Ad esempio un autoencoder che viene addestrato su immagini di macchine potrebbe imparare l'angolazione della foto, il colore, le dimensioni, il tipo di autovettura e così via. Questa proprietà risulta particolarmente interessante nella generazione di nuovi campioni. Ad esempio, se l'autoencoder non ha mai visto un'immagine di una utilitaria rossa, ma è stato trainato a sufficienza con immagini di

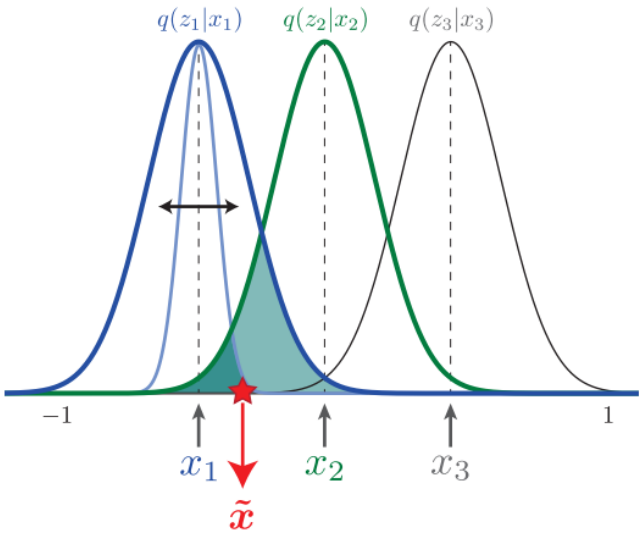


Figura 5.4: Un campione $\tilde{\boldsymbol{x}} \sim q_\phi(\boldsymbol{z_2}|\boldsymbol{x_2})$ è probabile sia confuso con uno generato da $q_\phi(\boldsymbol{z_1}|\boldsymbol{x_1})$. Nonostante ciò se $\boldsymbol{x_1}$ e $\boldsymbol{x_2}$ sono vicini nello spazio dei dati il termine di ricostruzione sarà minimo. Immagine presa da [5].

utilitarie nere e di berline rosse, spostandomi accuratamente nello spazio latente dovrei essere in grado di generare i nuovi campioni desiderati. Spesso i fattori di variazione che influenzano fortemente il termine di ricostruzione sono anche quelli macroscopicamente individuabili da un essere umano, per questo appare che i *variational autoencoder* organizzino lo spazio latente in modo estremamente logico.

I β-VAE raggiungono una capacità più elevata di decomporre i fattori di variazione nelle variabili dello spazio latente rispetto ad un semplice VAE. I motivi di questa abilità di VAE e β-VAE rimangono ancora in parte sconosciuti [5], sebbene si pensa che la proprietà di località e la definizione di *posterior* come una gaussiana con matrice di covarianza diagonale (dimensioni non correlate), giochino un ruolo fondamentale.

Capitolo 6

Autoencoders per l'estrazione di *features* nei segnali calcio

Al fine di ricavare i fattori generatori del PAW, in modo tale che il segnale calcio prodotto dalla pianta si confonda con quello prodotto da stress abiotici e biotici, si è proceduto a più stadi.

- Per prima cosa si è progettato un autoencoder convoluzionale per ricavare le 80 *features* più significative dai dati.
- È stato aggiunto un secondo autoencoder, innestato al primo, per ridurre ulteriormente la dimensione dello spazio latente fino ad una rappresentazione 2D. Per questo scopo è stato addestrato sia un *fully dense* autoencoder che un β-VAE.
- Si è sviluppato un applicativo grafico attraverso il quale è possibile studiare il *latent space* 2D, per ricavare importanti informazioni su come continuare la ricerca e avere un'idea dei fattori generatori più importanti.

Dato il limitato campione di segnali e l'insufficiente numero di fattori di controllo sul PAW, non ci aspettiamo di generare segnali particolarmente simili tra PAW e stress esterni. Tuttavia, miriamo a fornire un percorso per orientare la futura ricerca e la raccolta di nuovi dati. Per lo studio di queste serie temporali non è stato utilizzato una rete ricorrente, come ad esempio tramite LSTM, per via della limitata cardinalità del dataset e per la potenza di calcolo necessaria ad allenare una rete di questo tipo. Il fatto di avere serie di lunghezza determinata ci ha permesso di utilizzare facilmente le *feed forward neural network*, tramite gli autoencoder.

6.1 *Outer* autoencoder

Per la prima riduzione di dimensionalità si è optato per un autoencoder convoluzionale. La scelta è basata sulla volontà di catturare le relazioni spaziali dei segnali calcio. Sono stati così

introdotti nell'encoder due layer convoluzionali seguiti da una parte densa, mentre nel decoder la struttura speculare con un parte densa e due layer di convoluzione trasposta. La struttura dell'encoder è mostrata in figura 6.1, metre quella del decoder è perfettamente speculare.

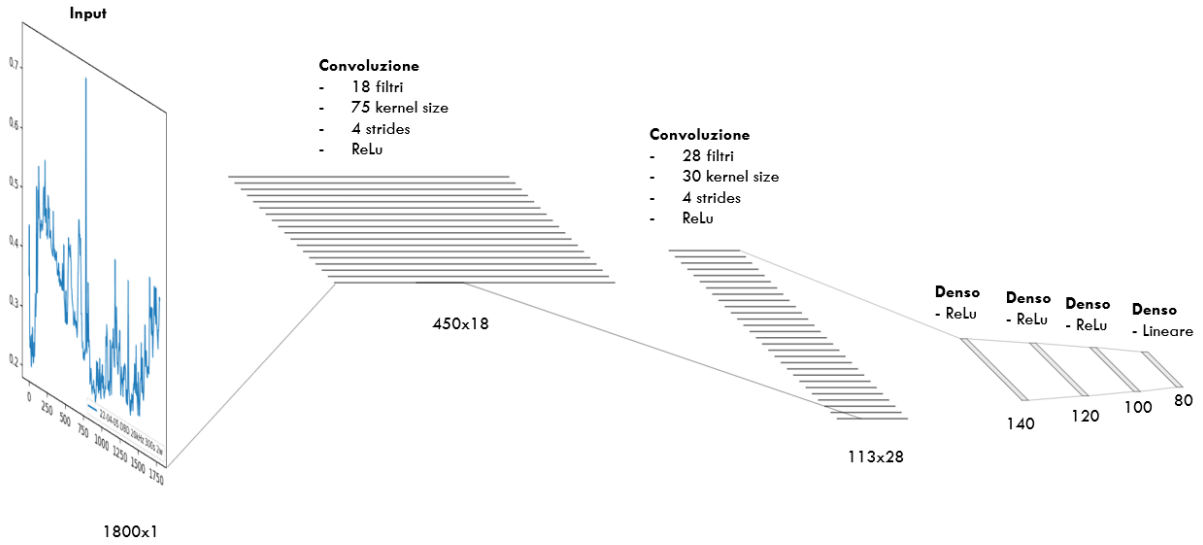


Figura 6.1: Outer encoder visualization, immagine ottenuta grazie a NN-SVG.

Lo strides è stato impostato a quattro per la opportuna riduzione di dimensionalità dell'ingresso. Alcuni parametri, come il numero di filtri e il *kernel size*, sono stati trovati grazie a una *utility* chiamata *KerasTuner* che permette di iniziare più volte il training con alcune combinazioni di parametri e trovare così la più promettente. L'ultimo layer denso nell'encoder ha una funzione di attivazione lineare in modo da permettere una distribuzione più uniforme dello spazio latente, che altrimenti sarebbe limitato ai valori positivi in ogni dimensione se si fosse utilizzata, anche in questo layer, ReLu.

6.1.1 Risultati del *training*

Il dataset iniziale è stato mescolato e partizionato in due insiemi, uno di *training*, contenente l'80% dei campioni e uno di *validation* con i rimanenti. È stato applicato l'*asymmetrical alpha-trimmed median filter* per eliminare il rumore nei dati. La ricostruzione qualitativa di un sottoinsieme del dataset di validation è mostrato in figura 6.2, mentre i risultati quantitativi sono mostrati in tabella 6.1.

5000 epoche	Training	Validation
MSE $[(\mu M)^2]$	30.4μ	54.8μ

Tabella 6.1: Risultati del training per l'outer autoencoder.

Per il *training* dell'autoencoder si sono scelte 5000 epoche. Si è utilizzato anche una particolare funzione *call-back*, ossia una parte di codice che viene chiamata al termine di ogni epoca, chiamata *EarlyStopping*. Questa permette di interrompere il training dopo che per un determinato numero di iterazioni il risultato di MSE validation non viene migliorato (nel nostro caso 300), ripristinando in automatico i migliori parametri del modello. Ciò permette di non dover controllare un eventuale stato di l'*over-fitting* durante il training, potendo mantenere comunque elevato il numero di epoche senza rischio di divergenza o consumo inutile di risorse di calcolo.

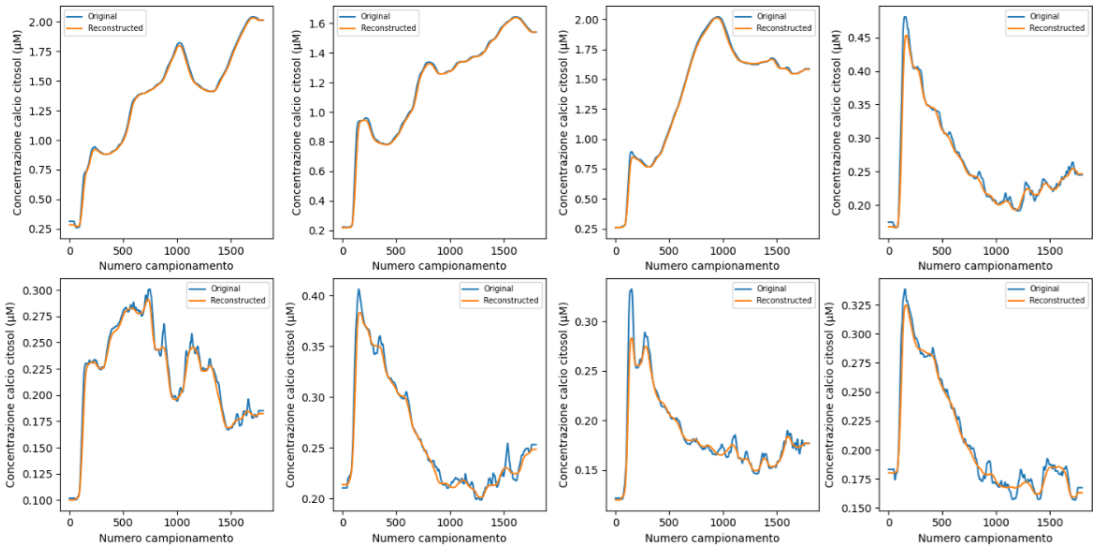


Figura 6.2: Ricostruzione segnali outer autoencoder da un sottoinsieme di D_V.

6.2 *Inner* autoencoder

La fase successiva è stata quella di innestare un altro autoencoder nella nostra rete. Abbiamo reso non *trainable* l'outer autoencoder, in modo tale che i suoi parametri rimanessero costanti durante l'allenamento. La struttura dell'autoencoder interno è mostrata in figura 6.3. Esso è costituito solamente da layer densi, perché è appropriato pensare che il suo input non abbia una struttura spaziale accentuata.

Non abbiamo creato direttamente un unico autoencoder, dalla dimensione dell'input fino alla rappresentazione 2D, per diverse ragioni. La prima perché è utile avere una rappresentazione latente più precisa, dove è possibile innestare altre reti, come un classificatore, e una meno accurata ma rappresentabile in 2D per poterlo studiare e trarre considerazioni importanti. La seconda è una ragione pratica, spazzare il *training* in due, permette di velocizzarlo in particolare quando avremo a che fare con il β-vae che ha tempi di allenamento più lunghi.

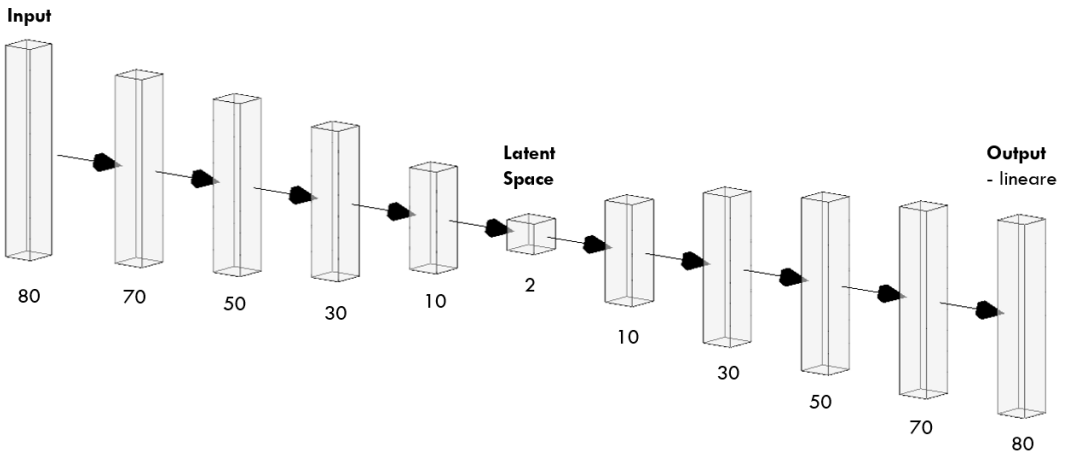


Figura 6.3: Inner autoencoder visualization, immagine ottenuta grazie a NN-SVG.

6.2.1 Risultati del *training*

L'autoencoder è stato trainato sulla medesima partizione effettuata per l'outer autoencoder. I risultati si possono vedere in tabella 6.2.

5000 epoche	Training	Validation
MSE $[(\mu M)^2]$	354μ	463μ

Tabella 6.2: Risultati del training per l'inner autoencoder.

I segnali hanno perso circa 10 volte la loro accuratezza, secondo la metrica MSE, a fronte però di una riduzione di un fattore 40 della dimensione dello spazio latente. Il risultato qualitativo dell'autoencoder si può osservare in figura 6.4.

Le rappresentazioni nel latent space di D sono mostrate in figura 6.5, ne studieremo meglio la distribuzione nel prossimo capitolo. La forma risulta appiattita sulla diagonale, mentre la dimensione ortogonale presenta una varianza minore, probabilmente catturando differenze meno accentuate nei segnali.

6.3 *Inner variational* autoencoder

L'ultimo autoencoder che abbiamo allenato è un β *variational* autoencoder, utile per la sua abilità di strutturare il latent space e per le sue capacità di *data augmentation*. Avendo considerato il limiti di rappresentatività dovuti alla parte variazionale dei modelli VAE, in particolare in presenza di una ridotta dimensionalità nello spazio latente, si è deciso di rilassare il contributo stocastico durante il training attraverso un *annealing* del parametro variazionale. Ovvero, du-

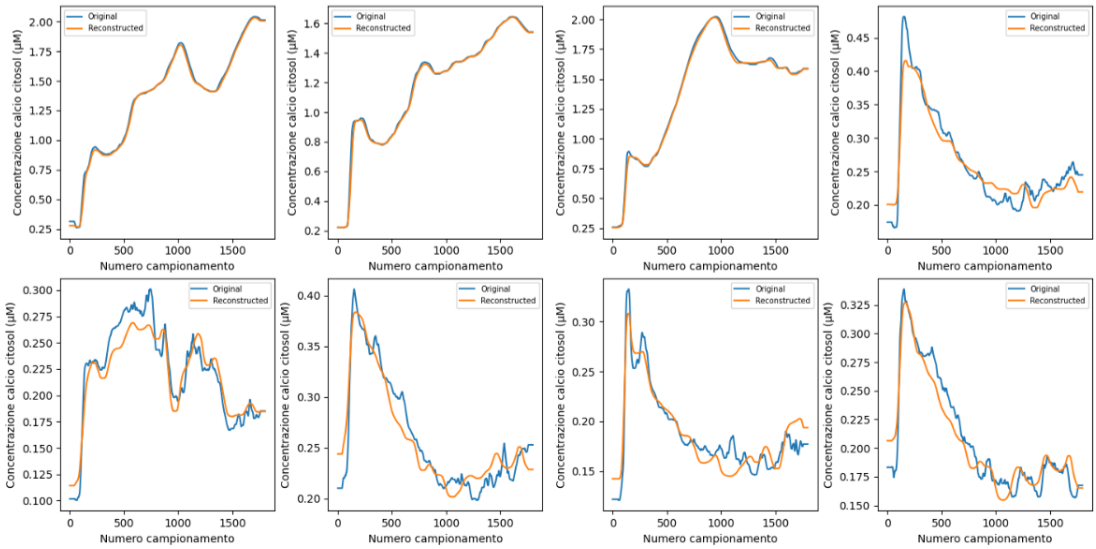


Figura 6.4: Ricostruzione segnali inner autoencoder da un sottoinsieme di D_V.

rante l'allenamento di questo modello, è stato fatto diminuire il valore di β ad ogni iterazione del *training*. L'idea è quella di aumentare progressivamente la capacità di *encoding* dello spazio latente, facendo imparare progressivamente nuovi fattori generativi al modello, mantenendo al contempo la struttura nello spazio latente e il *disentangle* delle variabili, comportamenti tipici di un β-vae con $\beta > 1$. L'idea è stata adattata da [5].

La struttura risulta molto simile a quella dell'autoencoder precedente, l'unica differenza è insita nello spazio latente dove abbiamo due layer densi che codificano per la media e la varianza della distribuzione normale per ogni campione. La struttura dell'autoencoder è mostrato in figura 6.6.

6.3.1 Risultati del *training*

L'autoencoder è stato addestrato sulla medesima partizione effettuata per gli altri due autoencoder. I risultati si possono vedere in tabella 6.3. La strategia di diminuzione di β durante il *training* ha permesso una degradazione contenuta delle capacità di ricostruzione dei segnali.

5000 epoche	Training	Validation
MSE $[(\mu M)^2]$	1214μ	1691μ

Tabella 6.3: Risultati del training per l'inner variational autoencoder.

Il risultato qualitativo dell'autoencoder si può osservare in figura 6.7. Alcuni segnali, con una struttura meno frequente nel dataset a disposizione, hanno una ricostruzione deteriorata mentre gli altri, più comuni, mantengono una buona accuratezza. Questo è un comportamento

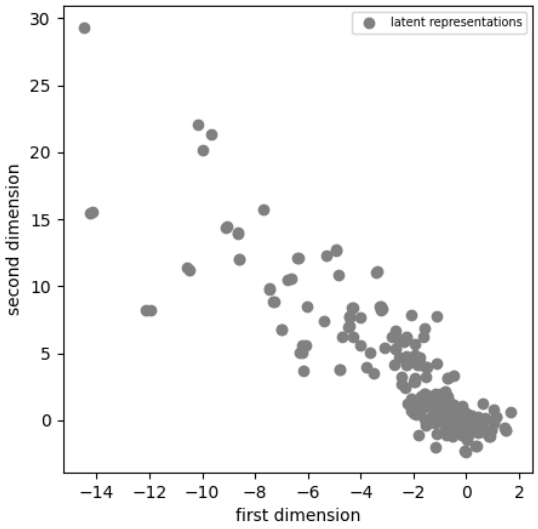


Figura 6.5: Latent space inner autoencoder.

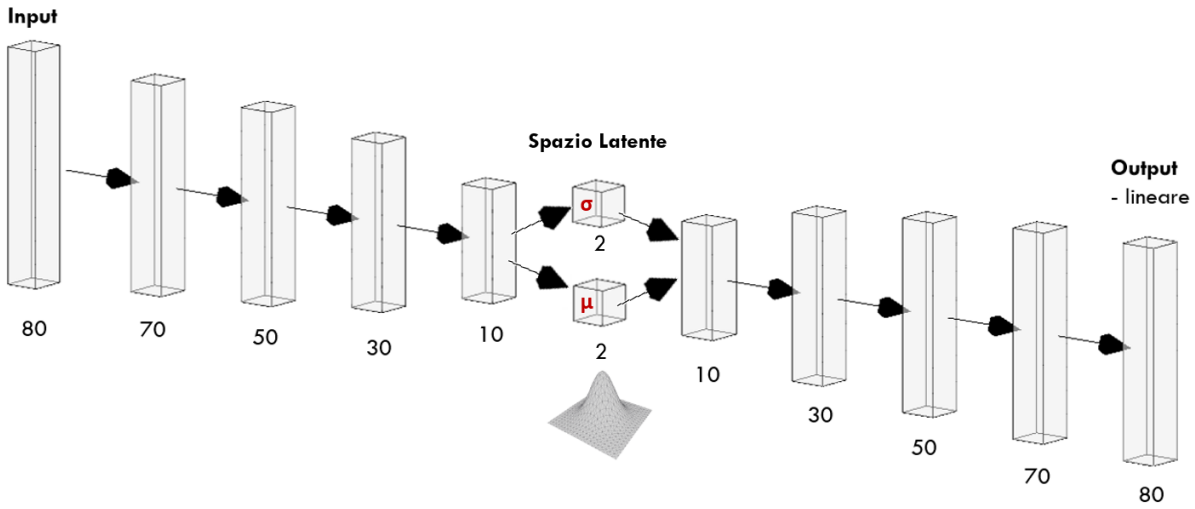


Figura 6.6: Inner variational autoencoder visualization, immagine ottenuta grazie a NN-SVG.

tipico dei variational autoencoder che necessitano un maggior numero di campioni per ottenere risultati comparabili con la loro controparte non probabilistica.

6.3.2 *Latent space traversing*

Per avere un'idea del *disentanglement* delle variabili e di cosa codificano, viene spesso fatto quello che prende il nome di *traversing* dello spazio latente. Viene cioè campionato a distanze costanti lungo le sue dimensioni, due nel nostro caso. È possibile vedere il risultato di questa procedura in figura 6.8. L'asse delle ascisse sembra codificare per la parte finale del segnale, se presenta decadimento o deriva. L'asse delle ordinate sembra codificare per il picco iniziale e la conca centrale del segnale.

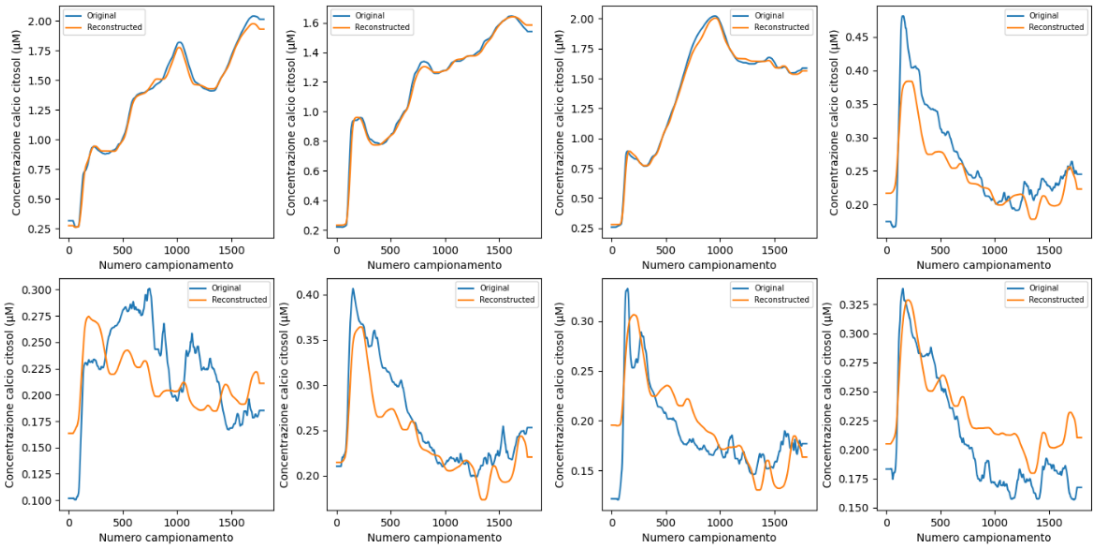


Figura 6.7: Ricostruzione segnali inner variational autoencoder da un sottoinsieme di D_V.

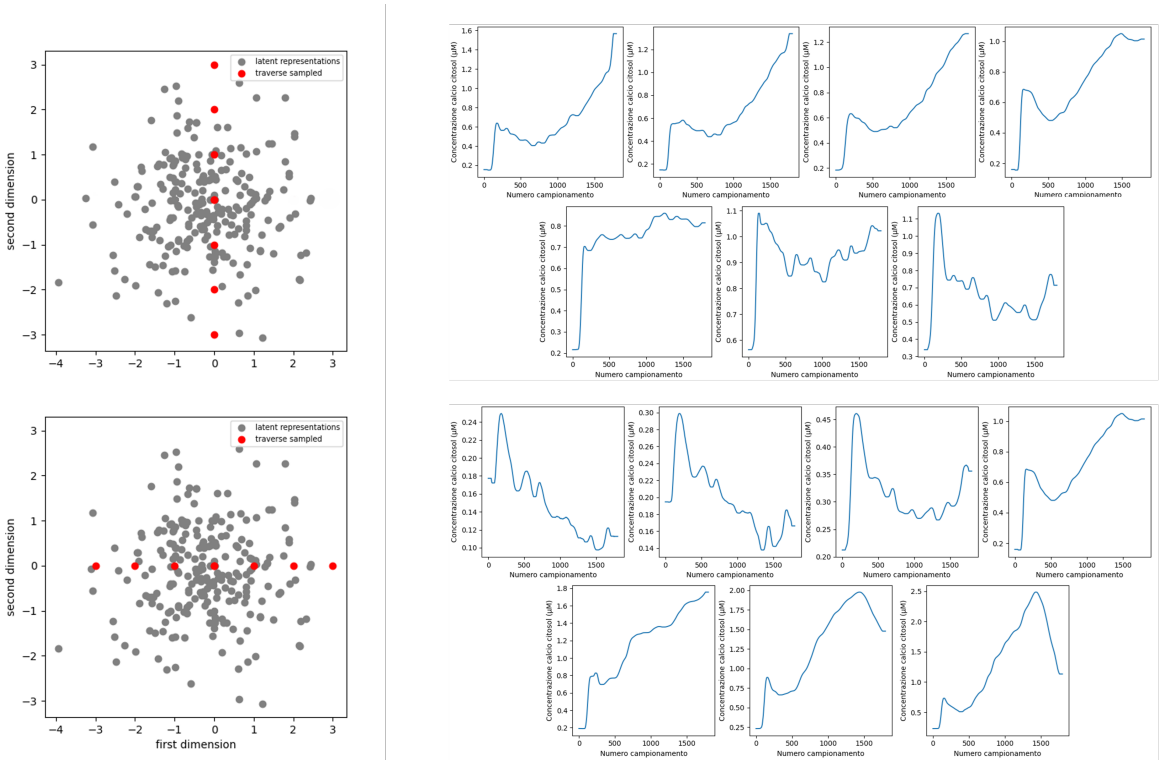


Figura 6.8: Latent space traverse, dimensione verticale e orizzontale.

6.3.3 *Data augmentation*

Un possibile utilizzo noto di questa particolare struttura di rete neurale è quello di generare nuovi segnali, campionandone lo spazio latente. Questi nuovi vettori potrebbero addirittura essere riutilizzati nel training di altre reti portando potenzialmente ad alcuni benefici, come l'incre-

mento dell'accuratezza in un possibile classificatore [7]. Questo studio non è stato affrontato nel corso di questo progetto anche se rimane una importante abilità dei β-vae che può essere valutata. Un campione di esempio è mostrato in figura 6.9. La ragione principale che, sebbene di fronte ad un dataset molto limitato, ha portato a non valutare questa possibilità è stato che l'intera analisi del presente studio è stata basata su una indagine diretta dello spazio latente dell'encoder. Il beneficio di un aumento di feature nel training dataset è quindi limitato e potrebbe essere indagato solo per valutare la possibilità di migliorare il training in regioni del latent space a bassa statistica. Tuttavia se durante una ulteriore campagna di studio dovesse risultare utile sviluppare modelli ulteriori per la predizione di parametri, ovvero la classificazione di regimi di reazione alla PAW, questa realizzazione assumerebbe grande utilità per ottenere un training efficace anche a fronte di uno scarso dataset sperimentale.

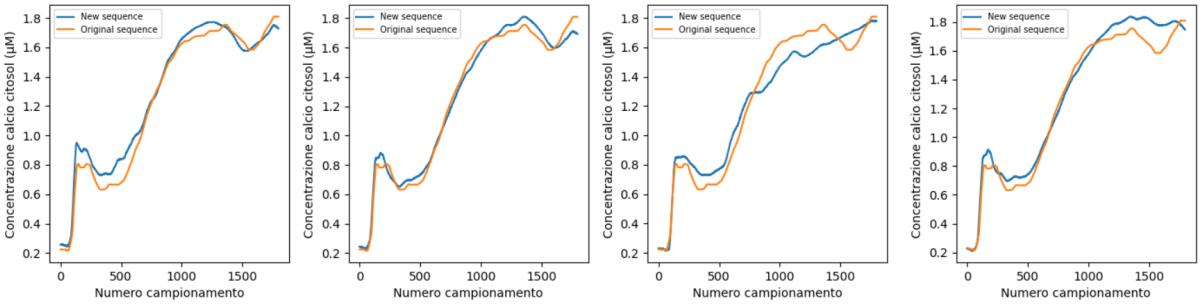


Figura 6.9: Data augmentation tramite beta variational autoencoder. $\boldsymbol{z} \sim \mathcal{N}(h^1_\phi(\boldsymbol{x}), 0.2)$.

Capitolo 7

Applicativo per lo studio dei latent space 2D prodotti

7.1 Presentazione del programma

Per studiare i diversi latent space prodotti dagli autoencoder abbiamo optato per una visualizzazione 2D. È stato sviluppato un applicativo python, tramite il modulo di *matplotlib* e *tkinter*, per la loro visualizzazione. La pagina principale è visibile in figura 7.1.

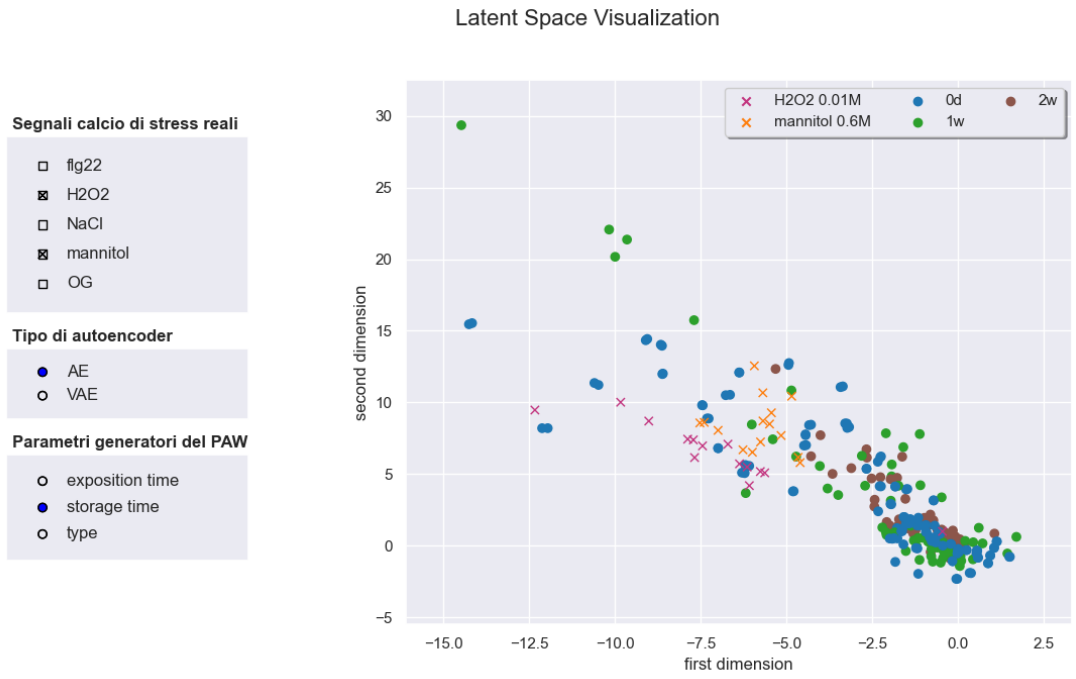
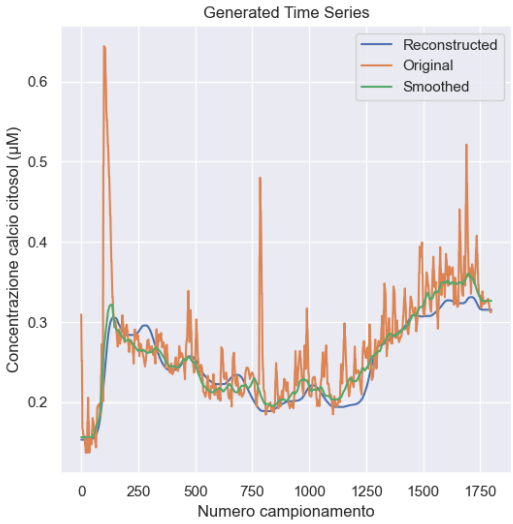


Figura 7.1: Schermata principale del tool di visualizzazione dei latent space generati da segnali calcio.

Se un punto del *latent space* viene premuto tramite il mouse, una finestra *pop-up* appare con il segnale ricostruito. Se la coordinata scelta corrisponde anche ad un punto reale del dataset allora vengono mostrate ulteriori informazioni, come il segnale originale e la sua versione filtrata. Questo è utile anche per renderci conto della qualità dell'autoencoder. Un esempio è mostrato in figura 7.2.

(a) Punto del latent space ricostruito a partire da coordinate 2D.



(b) Segnale nel dataset ricostruito a partire dalla sua rappresentazione latente.

Figura 7.2: Schermata pop-up a seguito di un click all'interno dello spazio latente.

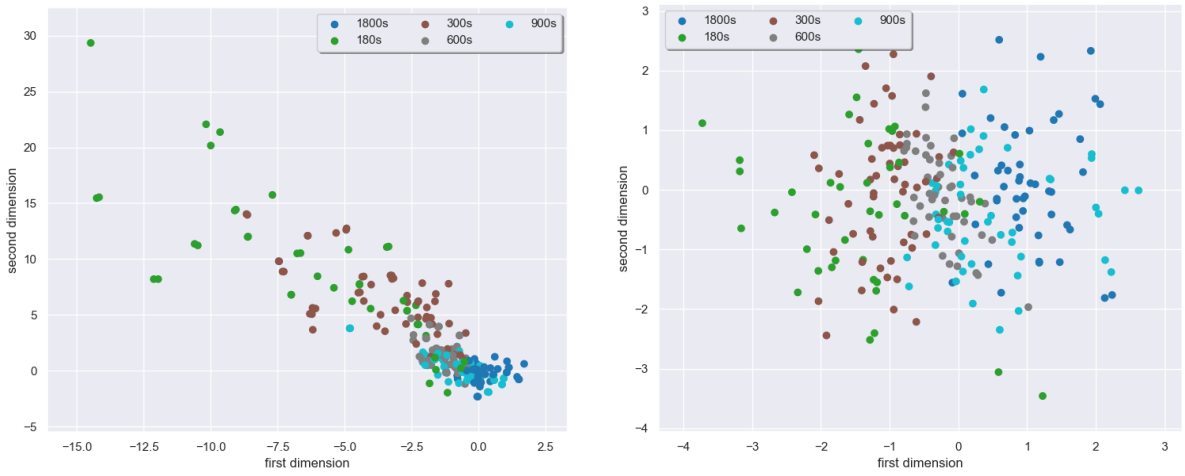
I comandi a sinistra della schermata permettono di modificare alcune caratteristiche del latent space. La prima se mostrare o meno alcuni segnali di stress reali, sconosciuti alla rete, cioè non *trainata* a riconoscerli. La seconda permette di cambiare il tipo di autoencoder da classico a variazionale, cambiando quindi la forma del *latent space*. L'ultima permette di evidenziare, attraverso dei colori diversi, una categoria specifica di fattori generatori del plasma, nel nostro caso tempo di esposizione, tempo di storage e tipo di generatore di PAW.

7.2 Analisi dei fattori generativi del plasma

7.2.1 Tempo di esposizione

Il tempo di esposizione è il fattore generatore più importante tra quelli disponibili. Infatti vediamo da figura 7.3 che in entrambi gli autoencoder questa categoria produce *cluster* riconoscibili. Le differenze principali tra i segnali con differenti tempi di esposizione al DBD o PT risiede nell'altezza del picco iniziale e nella caratteristica finale di deriva o decadimento della con-

centrazione. Queste caratteristiche sono le più evidenti nei segnali e dunque quelle su cui è strutturato il *latent space*.



(a) AE. (b) VAE.

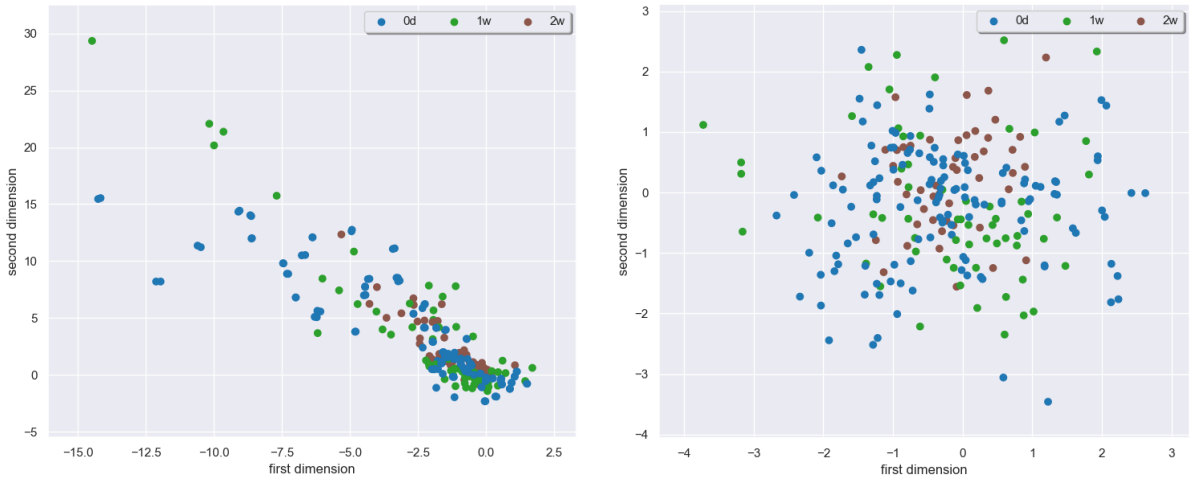
Figura 7.3: Latent space colorato in base agli elementi corrispondenti al tempo di esposizione.

7.2.2 Tempo di storage

Il tempo di storage, come mostrato in figura 7.4, in prima analisi non viene mappato nel *latent space* nel modo atteso. Infatti un maggiore tempo di storage, come detto nell'introduzione, dovrebbe trasdurre un segnale calcio meno accentuato, sovrapponendosi, almeno in parte, alle aree dello spazio latente dove venivano mappati i segnali con tempo di esposizione minore. Notiamo comunque che a parità di tempo di esposizione, il tempo di storage tende a spostare la rappresentazione nello spazio latente verso aree meno accentuate, in particolare per tempi di esposizione più lunghi. Questa caratteristica è mostrata in figura 7.5.

7.2.3 Tipo di generatore di PAW

L'ultimo fattore è il tipo di generatore del PAW e la sua alimentazione. Come già detto nel secondo capitolo la frequenza del DBD tende ad aumentare leggermente l'area sotto la curva, cosa che viene ritrovata nei due *latent space* in figura 7.6. La plasma torch a $900W$ fondendo più energia all'acqua distillata dovrebbe produrre PAW con segnali trasdotti più accentuati. Sfortunatamente i pochi dati misurati non permettono di essere interpretati in maniera confidente.



(a) AE. (b) VAE.

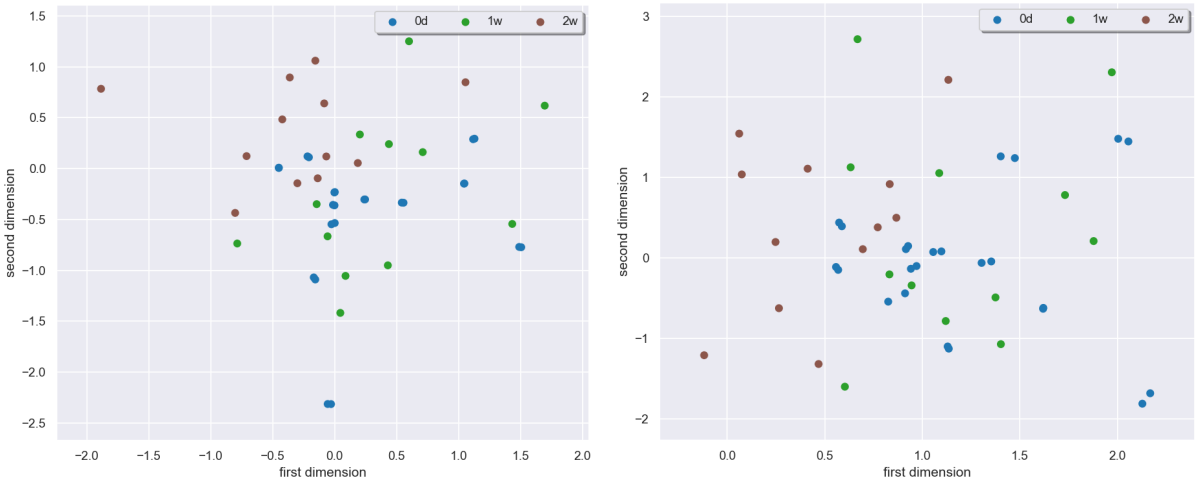
Figura 7.4: Latent space colorato in base agli elementi corrispondenti al tempo di storage.

7.3 Analisi della rappresentazione nel latent space di segnali reali

Non avendo allenato i due autoencoder con i dati provenienti da stress reali possiamo osservare dove la loro rappresentazione nel *latent space* si posiziona in relazione alle altre. La ricostruzione da parte degli autoencoder dei segnali reali non sarà ottima, ma osservando dove questi segnali vengono mappati, possiamo ricavare informazioni qualitative sui migliori parametri generatori del plasma. Di conseguenza il seguito del capitolo sarà dedicato a sviluppare questo tipo di analisi per il peptide *flg22*.

7.3.1 Segnali calcio trasdotti dal peptide *flg22*

Il peptide *flg22* (flagellina) è una proteina presente nei flagelli batterici. Essa viene riconosciuta dai recettori presenti nella membrana cellulare della pianta e ne stimola la risposta immunitaria [12]. Possiamo quindi utilizzarla per simulare un'infezione all'interno dell'*Arabidopsis thaliana*. Nell'immagine 7.7 viene mostrato il *latent space* colorato per tempo di esposizione, dove la rappresentazione latente dei segnali trasdotti dopo somministrazione di una soluzione 1µM di *flg22* è evidenziata tramite delle "x". Osservando specialmente la figura b, appare chiaro che il tempo di esposizione migliore per l'imitazione di questo tipo di stress è minore o uguale a $180s$. Siccome il tempo di esposizione è il fattore più caratterizzante, come osservato nelle sezioni precedenti, filtriamo il dataset per questo valore e vediamo come effettuare il *tuning* per gli altri due.

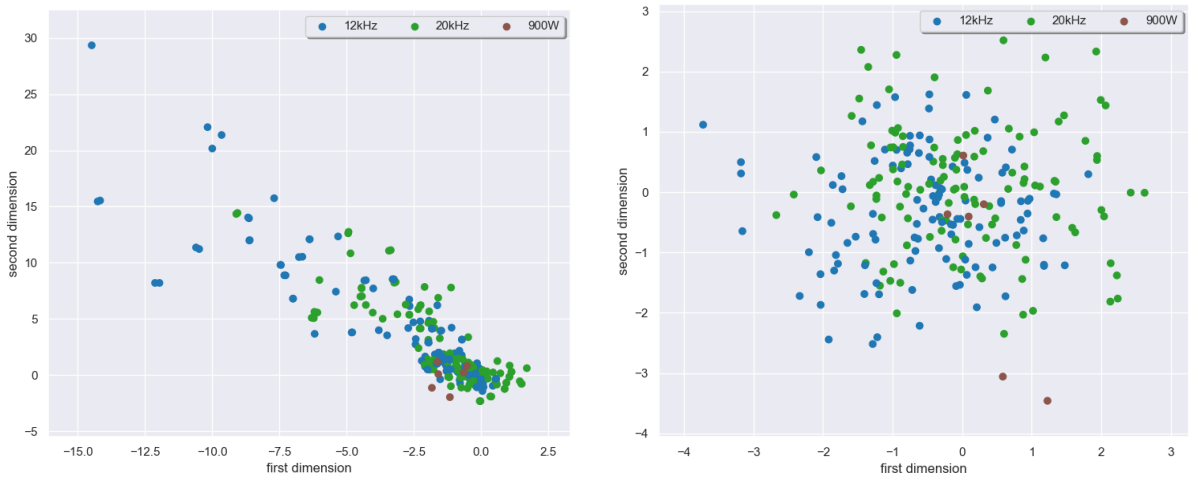


(a) AE. (b) VAE.

Figura 7.5: Rappresentazioni latenti dei segnali con tempo di esposizione di $1800s$, evidenziate per tempo di storage.

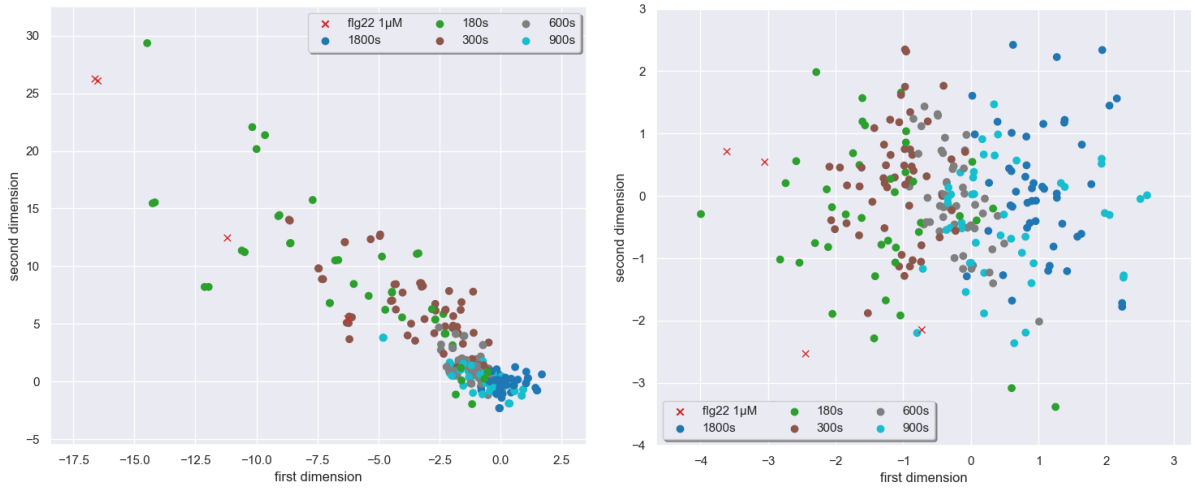
Per quanto riguarda il tempo di storage, dalla figura 7.8 non riusciamo a trarre informazioni particolari, mentre per il tipo di generatore di PAW e la sua alimentazione, la figura 7.9, indica la migliore come DBD a $12kHz$.

L'analisi dei latent space indica non solo i parametri più simili, ma anche quelli dove concentrare la ricerca per la raccolta di nuovi dati e una loro possibile variazione. Ad esempio, per riprodurre i segnali trasdotti da *flg22*, sarebbe utile raccogliere dati su segnali calcio prodotti con acqua esposta a tempi inferiori di $180s$ al DBD $12kHz$.



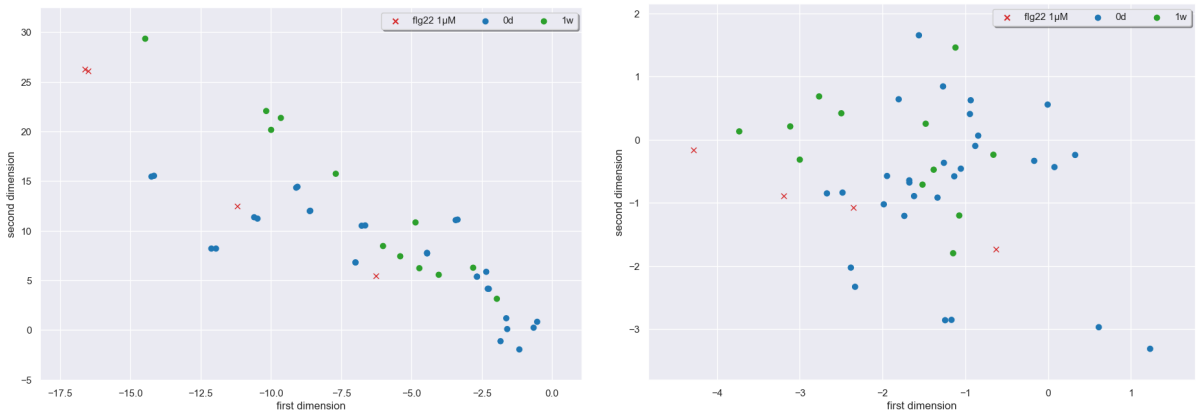
(a) AE. (b) VAE.

Figura 7.6: Latent space colorato in base agli elementi corrispondenti al tipo di generatore di PAW.



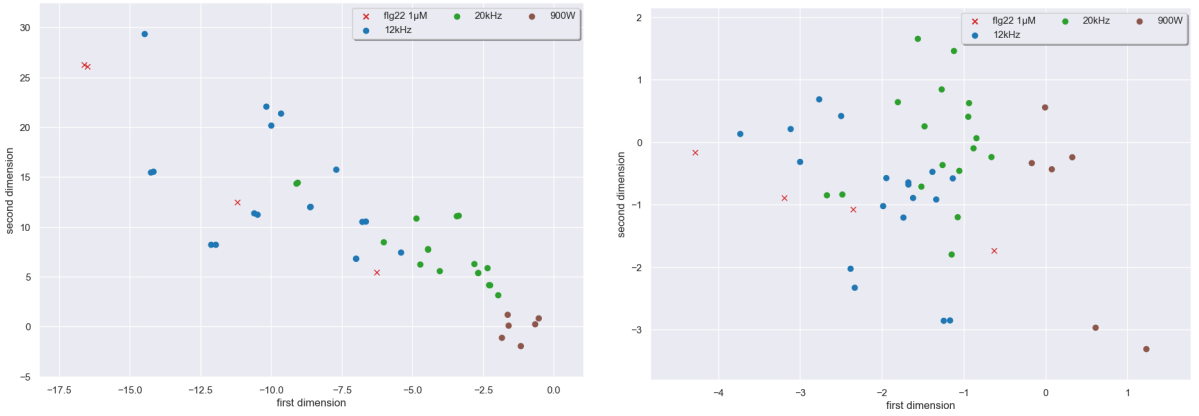
(a) AE. (b) VAE.

Figura 7.7: Latent space colorato in base agli elementi corrispondenti al tempo di esposizione del PAW e al peptide flg22.



(a) AE. (b) VAE.

Figura 7.8: Latent space colorato in base agli elementi corrispondenti al tempo di storage del PAW e al peptide flg22. Dataset filtrato per valori del tempo di esposizione pari a $180s$.



(a) AE. (b) VAE.

Figura 7.9: Latent space colorato in base agli elementi corrispondenti al tipo di generatore del PAW e al peptide flg22. Dataset filtrato per valori del tempo di esposizione pari a $180s$.

Capitolo 8

Conclusioni

In questa tesi si sono analizzati, attraverso modelli di *machine learning*, i risultati della somministrazione di acqua attivata (PAW) a piante modello di *Arabidopsis thaliana*. Sono stati sviluppati due autoencoder in grado di modellare la relazione altamente non lineare tra il segnale calcio trasdotto all'interno della pianta e i parametri generatori del PAW. È stato sviluppato un applicativo grafico per visualizzare in maniera interattiva il latent space 2D generato dai due autoencoder. Questo spazio latente permette di utilizzare le informazioni apprese durante il training sui segnali generati da PAW per intuire i migliori parametri con cui generare l'acqua attivata in modo da emulare al meglio un segnale calcio proveniente da uno stress fisico.

L'applicativo permette di indirizzare la raccolta di nuovi dati verso i parametri più promettenti del PAW. In futuro si raccoglieranno maggiori informazioni sui fattori variazionali del PAW che hanno la possibilità di influenzarne la chimica, come ad esempio temperatura e umidità. L'applicativo è facilmente scalabile poiché il training non utilizza queste informazioni.

Sarà possibile utilizzare un classificatore innestato nello spazio latente dell'*outer* autoencoder per ottenere una risposta quantitativa sui migliori parametri da scegliere. Sarà anche possibile sfruttare le capacità del *variational autoencoder* per incrementare ulteriormente il dataset su cui allenare il classificatore.

Con l'aumentare di dati e fattori di variazione ci aspettiamo che l'approccio delineato dalla tesi permetta una simulazione sempre migliore di stress fisici tramite l'acqua attivata.

Bibliografia

[1] B. Adhikari, M. Adhikari e G. Park, «The Effects of Plasma on Plant Growth, Development, and Sustainability,» *Applied Sciences*, vol. 10, n. 17, 2020, ISSN: 2076-3417. DOI: 10.3390/app10176045. indirizzo: https://www.mdpi.com/2076-3417/10/17/6045.

[2] D. Bank, N. Koenigstein e R. Giryes, *Autoencoders*, 2021. arXiv: 2003.05991 [cs.LG].

[3] B. Bölter, F. Seiler e J. Soll, «Analysis of *Arabidopsis thaliana* Growth Behavior in Different Light Qualities,» *J. Vis. Exp.*, n. 132, e57152, 2018. DOI: 10.3791/57152.

[4] M. Brini, «Calcium-sensitive photoproteins,» *Methods (San Diego, Calif.)*, vol. 46, pp. 160–6, nov. 2008. DOI: 10.1016/j.ymeth.2008.09.011.

[5] C. P. Burgess, I. Higgins, A. Pal et al., «Understanding disentangling in β-VAE,» *arXiv preprint arXiv:1804.03599*, 2018.

[6] L. Cayton, «Algorithms for manifold learning,» lug. 2005.

[7] C. Chadebec, E. Thibeau-Sutre, N. Burgos e S. Allassonnière, «Data Augmentation in High Dimensional Low Sample Size Setting Using a Geometry-Based Variational Autoencoder,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, n. 3, pp. 2879–2896, 2023. DOI: 10.1109/TPAMI.2022.3185773.

[8] E. Cortese, A. Galenda, A. Famengo et al., «Quantitative Analysis of Plant Cytosolic Calcium Signals in Response to Water Activated by Low-Power Non-Thermal Plasma,» *International Journal of Molecular Sciences*, 2022.

[9] E. Cortese, A. G. Settimi, S. Pettenuzzo et al., «Plasma-Activated Water Triggers Rapid and Sustained Cytosolic Ca2+ Elevations in Arabidopsis thaliana,» *Plants*, vol. 10, n. 11, 2021, ISSN: 2223-7747. DOI: 10.3390/plants10112516. indirizzo: https://www.mdpi.com/2223-7747/10/11/2516.

[10] C. Doersch, «Tutorial on Variational Autoencoders,» *arXiv preprint arXiv:1606.05908*, 2016.

[11] I. Goodfellow, Y. Bengio e A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016, ISBN: 978-0262035613.

[12] J. Jelenska, S. M. Davern, R. F. Standaert, S. Mirzadeh e J. T. Greenberg, «Flagellin peptide flg22 gains access to long-distance trafficking in Arabidopsis via its receptor, FLS2,» *Journal of Experimental Botany*, vol. 68, n. 7, pp. 1769–1783, 2017. DOI: 10.1093/jxb/erx060.

[13] A. Mai-Prochnow, R. Zhou, T. Zhang et al., «Interactions of plasma-activated water with biofilms: inactivation, dispersal effects and mechanisms of action,» *Nature*, 2021.

[14] «Major Plasma Disharges and their Applicability for Plasma Medicine,» in *Plasma Medicine*. 2013, cap. 4, pp. 165–267, ISBN: 9781118437704. DOI: https://doi.org/10.1002/9781118437704.ch4. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118437704.ch4. indirizzo: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118437704.ch4.

[15] M. A. Nielsen, *Neural Networks and Deep Learning*. d26a51a7, 2015.

[16] E. Plaut, *From Principal Subspaces to Principal Components with Linear Autoencoders*, apr. 2018.

[17] N. Tuteja e S. Mahajan, «Calcium signaling network in plants: an overview,» *Plant Signaling & Behavior*, vol. 2, n. 2, pp. 79–85, mar. 2007. DOI: 10.4161/psb.2.2.4176.

[18] «Understanding Variational Autoencoders (VAEs).» Articolo online. (), indirizzo: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (visitato il giorno 19/03/2024).

[19] K. S. Wong, N. S. L. Chew, M. Low e M. K. Tan, «Plasma-Activated Water: Physicochemical Properties, Generation Techniques, and Applications,» *Processes*, vol. 11, n. 7, 2023, ISSN: 2227-9717. DOI: 10.3390/pr11072213. indirizzo: https://www.mdpi.com/2227-9717/11/7/2213.

[20] T. Xu, J. Niu e Z. Jiang, «Sensing Mechanisms: Calcium Signaling Mediated Abiotic Stress in Plants,» *Frontiers in Plant Science*, vol. 13, p. 925 863, giu. 2022. DOI: 10.3389/fpls.2022.925863.

[21] R. Yamashita, M. Nishio, R. K. G. Do e K. Togashi, «Convolutional neural networks: an overview and application in radiology,» *Insights into Imaging*, vol. 9, n. 4, pp. 611–629, 2018, ISSN: 1869-4101. DOI: 10.1007/s13244-018-0639-9. indirizzo: https://doi.org/10.1007/s13244-018-0639-9.