Relazione finale - Ricostruttore di stati

Luca Lavazza, Francesco Rossi

Agosto 2022

1 Introduzione

In questo elaborato verranno presentati i risultati relativi al progetto di costruzione di un autoencoder e di un ricostruttore/predittore di stati.

Un autoencoder è una rete neurale capace di rappresentare in modo efficiente dati in input, creando i così detti "coding" o "embedding" senza supervisione umana.
Gli embedding costituiscono un'efficiente rappresentazione dei dati, ridotti a una dimensionalità inferiore. Questo fa degli autoencoder dei potenti strumenti di dimensionality reduction (come in questo caso) ma più in generale hanno appicazioni anche come feature detectors o generative models.

Lo scopo del ricostruttore/predittore è quello di associare a ciascuno stato un contesto di stati legati ad esso, in modo tale che l'embedding contenga informazioni sui *surroundings* dell'elemento centrale.
Questa funzionalità può essere meglio spiegata con un esempio legato al mondo del Natural Language Processing. Un autoencoder si limiterebbe a fornire una rappresentazione compatta di un dizionario, ad esempio, senza legare le parole tra loro in alcun modo. Quello che il predittore è invece in grado di fare, è di creare delle connessioni logiche tra parole, per esempio predicendo, come siamo abituati ormai a vedere nei nostri smartphone, possibili parole successive nella frase che stiamo componendo, insieme a sinonimi, o finamai emoji.

Nei successivi paragrafi verranno presentati inizialmente le informazioni preliminari inerenti al lavoro, e successivamente i 3 task su cui è stata basata la sperimentazione.

2 Dati ed indicazioni generali

Tramite funzioni d'utilità, e che non riportiamo in questo documento per brevità, abbiamo definito e creato dei dataset statici da importare prima dell'esecuzione dei task (sulla base dei dati non elaborati forniti all'inizio del lavoro), cosicchè questi potessero lavorare sugli stessi set ogni volta.
Si fa notare che è stato fatto un tentativo di controllo della presenza di duplicati tra training e test set, tuttavia, data la mole enorme di dati, non è stato possibile portarlo a termine: dopo 14 ore erano comunque stati trovati solamente 5 vettori uguali, il che fa ben pensare che il numero di doppioni sia insignificante e ininfluente.

È stato utilizzato un dataset molto ampio di vettori sparsi, che rappresentavano gli stati di un planner. Il dataset, composto da $1.382 \cdot 10^6$ vettori da 340 valori ciascuno (che possono assumere il valore 0 o 1), è stato suddiviso nei modi seguenti:

- **Task 1:**
  - Training set: $9.2 \cdot 10^5$ vettori;
  - Validation set: $2.31 \cdot 10^5$ vettori;
  - Test set: $2.31 \cdot 10^5$ vettori.
- **Task 2:**
  - Training set:
    - train_x: ($1.54 \cdot 10^5$, 4, 340)
    - train_y: ($1.54 \cdot 10^5$, 1, 340)
  - Validation set:
    - val_x: ($5.1 \cdot 10^4$, 4, 340)
    - val_y: ($5.1 \cdot 10^4$, 1, 340)
  - Test set:
    - test_x: ($5.1 \cdot 10^4$, 4, 340)
    - test_y: ($5.1 \cdot 10^4$, 1, 340)
  - Per far tornare i conti della suddivisione rispetto ai piani, sono stati scartati circa 10^5 vettori.
- **Task 3:**
  - Training set:
    - train_x: (10^5, 6, 340)
    - train_y: (10^5, 1, 340)
  - Validation set:
    - val_x: ($3.35 \cdot 10^4$, 4, 340)

    - val_y: ($3.35 \cdot 10^4$, 1, 340)
  - Test set:
    - test_x: ($3.35 \cdot 10^4$, 4, 340)
    - test_y: ($3.35 \cdot 10^4$, 1, 340)
  - Per far tornare i conti della suddivisione sulla base dei piani, sono stati scartati circa $2 \cdot 10^5$ vettori.

Per quanto riguarda le metriche di valutazione delle performance in fase di addestramento, è stato inizialmente difficile giudicare quali fossero le più adeguate da osservare. Abbiamo tuttavia concluso che precision e recall fossero le metriche principali per attestare la bontà delle reti addestrate, ed in generale per ottenere buoni risultati in fase di test (e quindi come codifica e decodifica) entrambe dovevano tendere ad 1. L'accuracy s'è rivelata meno influente per questi task, e infatti, metodi di ottimizzazione basati sull'accuracy come HPArams, dopo alcuni test si sono rivelati poco utili e poco propensi al miglioramento delle reti. La funzione di loss utilizzata è stata la "binary cross-entropy" vista la natura del dataset costituito da vettori binari. In generale ci si è accorti che per ottenere risultati soddisfacenti la loss doveva andare necessariamente sotto al 0.01. Tale indicazione è stata usata in fase di addestramento per capire se un modello potesse portare a buone performance o no. Infine, come funzione di output si è utilizzata una sigmoide: caratteristica di una sigmoide è restituire un risultato compreso tra 0 e 1, necessaria visto la natura binaria dei vettori del dataset.

Hardware utilizzato

È stata messa a disposizione la GPU per l'allenamento delle reti: in particolare una Nvidia Geforce RTX 2080 Ti e una Nvidia Geforce RTX 2060. In questo modo è stato possibile velocizzare gli allenamenti già di per sé discretamente lunghi (anche nell'ordine delle 6/8 ore in alcuni casi).
In particolare, le configurazioni hardware complete su cui sono stati effettuati gli allenamenti sono le seguenti:

- Rig 1:
  - CPU AMD Ryzen 7 2700X 4.1GHz 8c/16t;
  - 64GB Ram DDR4;
  - GPU Nvidia Geforce RTX 2080 Ti;
  - Windows 10 64 bit.
- Rig 2:
  - CPU AMD Ryzen 7 3800XT 3.9GHz 8c/16t;
  - 16GB Ram DDR4;
  - GPU Nvidia Geforce RTX 2060;
  - Windows 10 64 bit.

3 Task 1- Autoencoder classico

L'obiettivo del primo task è stato quello di verificare che l'embedding sui dati funzionasse e fosse efficace: si è data in input alla rete una serie di sequenze di 340 valori ciascuna, le quali sono state progressivamente ridotte considerevolmente di dimensione.
Nel caso l'embedding fosse efficiente, come si è mostrato, era poi possibile ricostruire le sequenze originali a partire da quelle rimpicciolite, con un basso numero di errori.

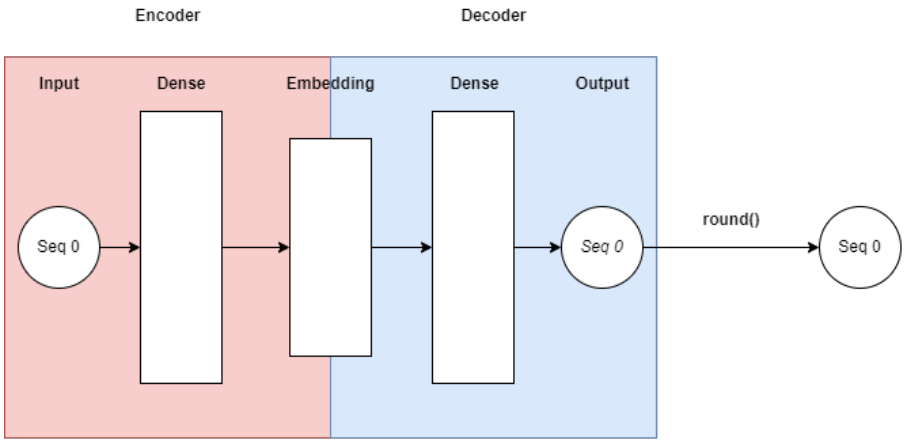

Figure 1: Struttura della rete per il primo task

3.1 Studio e creazione del modello

- Numero di layer: vista la dimensione dell'input (340) si è optato di utilizzare 3 livelli interni totali (compreso quello di embedding),. Più livelli avrebbero aumentato (inutilmente) la complessità del modello;
- Funzioni di attivazione: i migliori risultati sono stati ottenuti con la ReLU. Oltre a questa sono state testate swish e varianti della relu, quali pRelu, selu, leaky relu, ma senza particolari incrementi nelle prestazioni;
- Numero di unità: si è sempre cercato di mantenere una configurazione simmetrica della rete;
- Ottimizzatori: sono stati utilizzati adam ed Nadam;
- Metodi di regolarizzazione:
  - Early stopping: è stato utilizzato con patience pari a 10-15.
    S'è rivelato essenziale ed estremamente utile, concludendo spesso gli allenamenti ben prima del limite di epoche imposto;

- Kernel inizialization: avendo scelto una ReLU come funzione di attivazione principale, per l'inizializzazione dei pesi è stata utilizzata "he uniform", o al più "he normal";
- Kernel regolarization: L2 s'è rivelato il parametro di regolarizzazione più efficace nella maggioranza dei casi. Sono stati testati anche L1 e L1L2;
- Dropout: si è tentato di inserirlo, ma nel corso dei test i risultati migliori sono stati ottenuti in assenza di dropout;
- Batch normalization: è stata utilizzata, e si è rivelata estremamente utile e fondamentale per raggiungere prestazioni soddisfacenti.

3.2 Risultati del Task 1

Di seguito vengono riportati due modelli particolarmente interessanti, e i tre che hanno ottenuto i migliori risultati.

Modello 1

Il seguente modello è contenuto nella cartella "TestLogs/aeClassico/20220724-120127" allegata a questo report.

Prestazioni in fase di addestramento

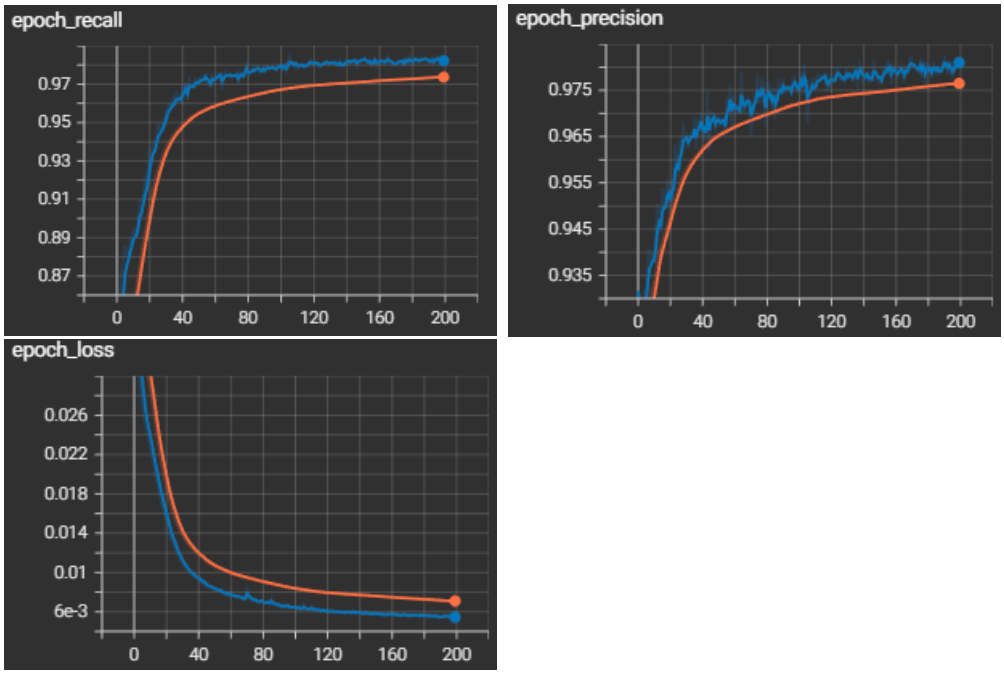

Figure 2: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

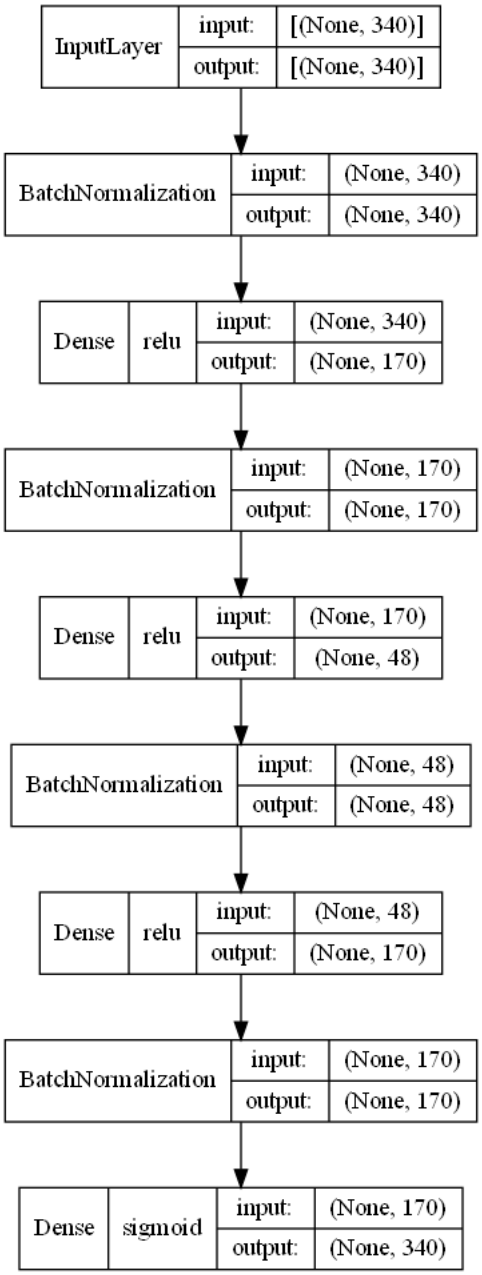

Figure 3: Architettura modello 1, ae classico con code=48

Parametri	Tipo	Valori
Kernel inizializer	he_uniform	-
Ottimizatore	Adam	lr=0.0001
Kernel regulizer	L2	10^{-5}
Batch size	-	150

Table 1: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 161064 su 230728 totali, corrispondente ad un 69.807% dell'insieme.

Modello 2

Il seguente modello è contenuto nella cartella "TestLogs/aeClassico/20220802-123318" allegata a questo report.

Prestazioni in fase di addestramento

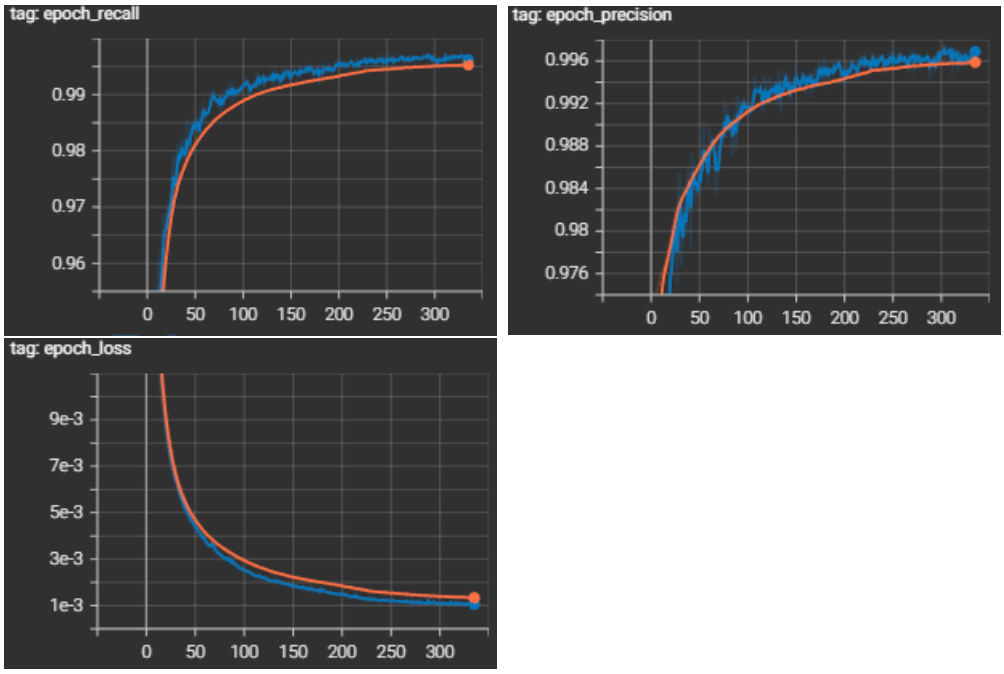

Figure 4: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

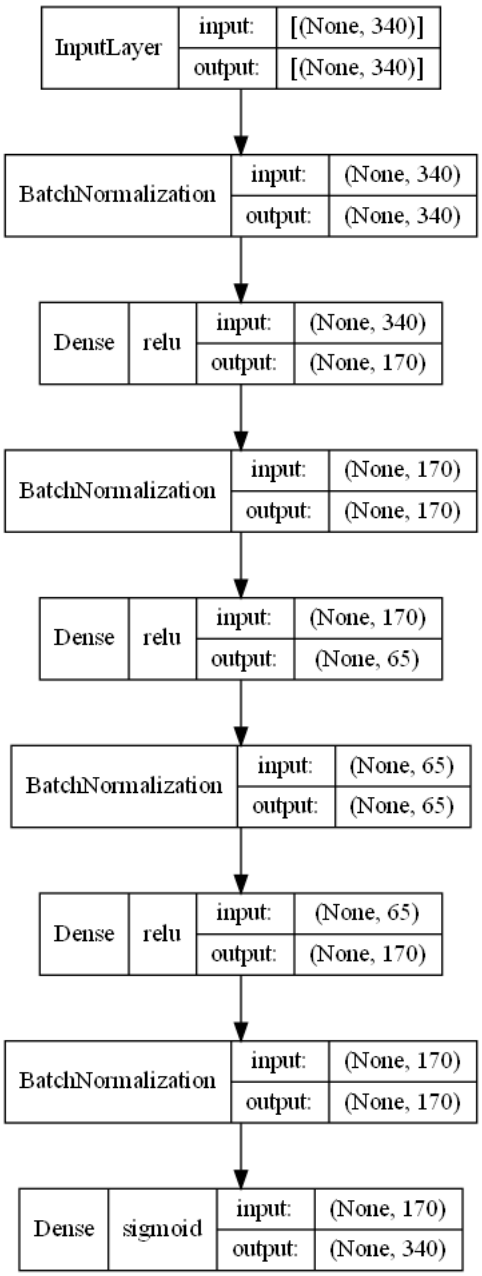

Figure 5: Architettura modello 2, ae classico con code=65

Parametri	Tipo	Valori
Kernel inizializer	heuniform	-
Ottimizatore	Nadam	lr=0.0007, $beta\_1$=0.95, $beta\_2$=0.999
Kernel regulizer	-	-
Batch size	-	5000

Table 2: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 213005 su 230728 totali, corrispondente ad un 92.319% dell'insieme.

Modello 3

Il seguente modello è contenuto nella cartella "TestLogs/aeClassico/20220802-115650" allegata a questo report.

Prestazioni in fase di addestramento

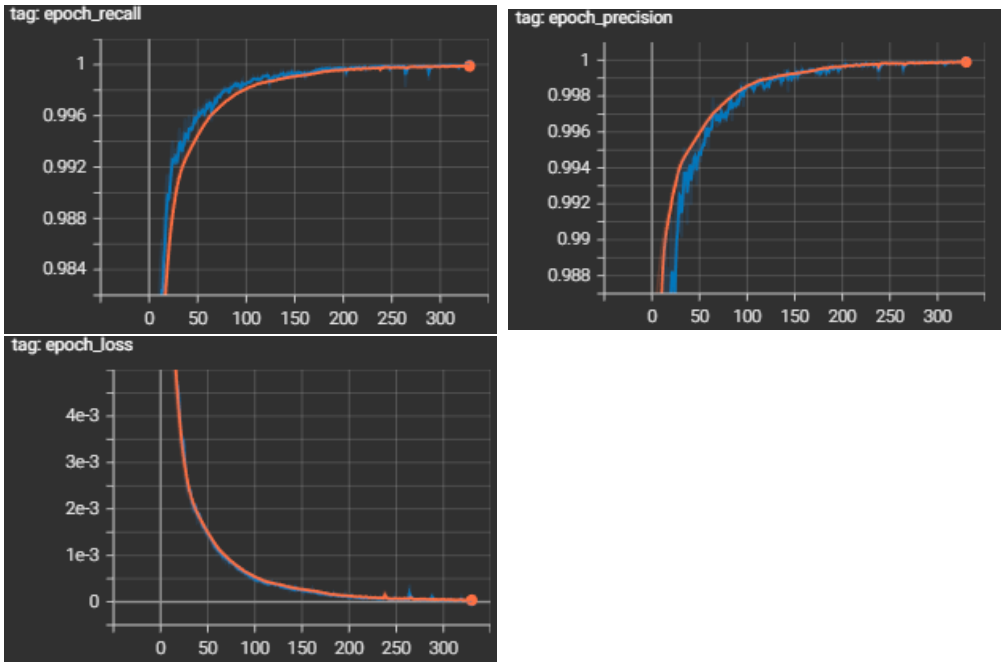

Figure 6: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

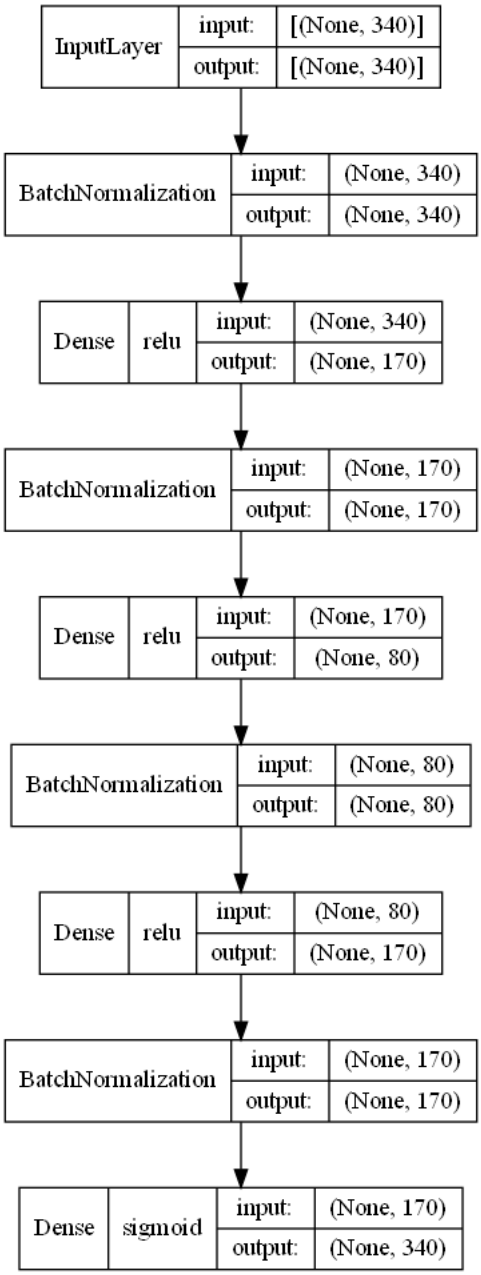

Figure 7: Architettura modello 3, ae classico con code=80

Parametri	Tipo	Valori
Kernel inizializer	he_uniform	-
Ottimizatore	Nadam	lr=0.001, $beta\_1$=0.9, $beta\_2$=0.999
Kernel regulizer	-	-
Batch size	-	5000

Table 3: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 230162 su 230728 totali, corrispondente ad un 99.755% dell'insieme.

Conclusione

Gli ottimi risultati ottenuti in questo task, sia in termini di performance che di encoding, hanno dimostrato la fattibilità del progetto.
Si è comunque osservato, , come prevedibile, che in concomitanza al rimpicciolimento dello strato di embedding, l'accuracy diminuiva: 48 è la dimensione alla quale ci siamo arrestati, cosicché si ottenessero risultati almeno discreti. Sotto quella soglia, i risultati erano pressoché inutilizzabili e il task perdeva di utilità.

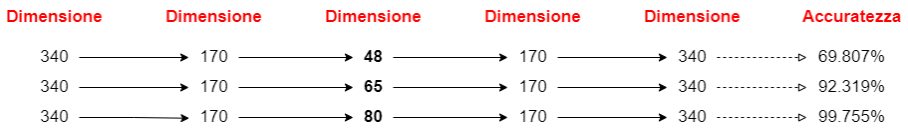

Figure 8: Recap delle reti realizzate per il primo task, e relativi risultati

4 Task 2- Ricostruttore 4 a 1

Data una sequenza di 5 vettori, L'obiettivo è stato quello di creare un modello in grado di ricostruire il vettore centrale della sequenza, fornendo in input i restanti 4.

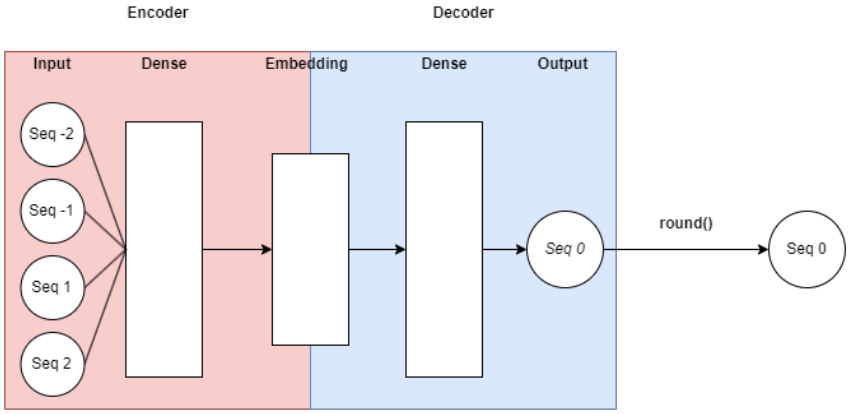

Figure 9: Struttura della rete per il secondo task

4.1 Studio e creazione del modello

- Numero di layer: Sono stati utilizzati 5 livelli interni totali, compreso quello di embedding;
- Funzioni di attivazione: anche in questo caso la relu è stata la predefinita;
- Numero di unità: si è sempre cercato di mantenere una configurazione simmetrica della rete;
- Ottimizzatori: sono stati testati Adam ed Nadam, ma i risultati migliori sono stati ottenuti col primo;
- Metodi di regolarizzazione:
  - Early stopping: metodo fondamentale che è entrato in funzione nella maggior parte dei casi.
  - Kernel inizialization: avendo scelto una ReLU come funzione di attivazione principale, per l'inizializzazione dei pesi è stata utilizzata "he uniform", o al più "he normal";
  - Kernel regolarization: utilizzo di funzioni quali L1,L2, L1L2, ma alla fine la più efficiente si è rivelata essere la L2.
  - Dropout: si è tentato di inserirlo, ma nel corso dei test i risultati migliori sono stati ottenuti in assenza di dropout;

– Batch normalization: è stata utilizzata, e si è rivelata estremamente utile e fondamentale per raggiungere prestazioni soddisfacenti.

4.2 Risultati del Task 2

Modello 1

Il seguente modello è contenuto nella cartella "TestLogs/Seq5/20220718-090055" allegata a questo report.

Prestazioni in fase di addestramento

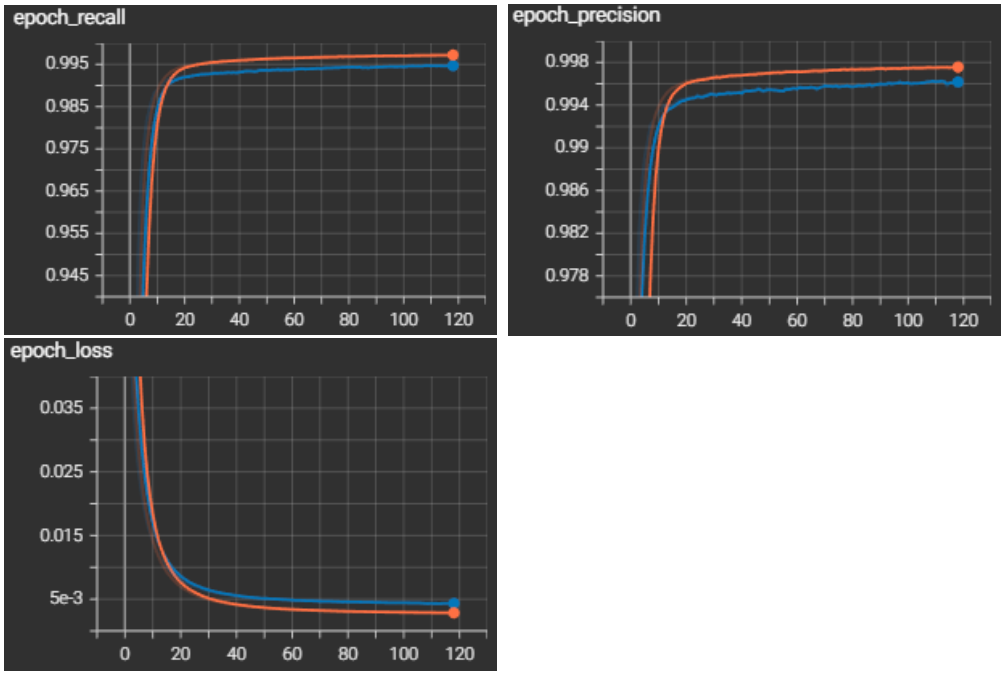

Figure 10: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

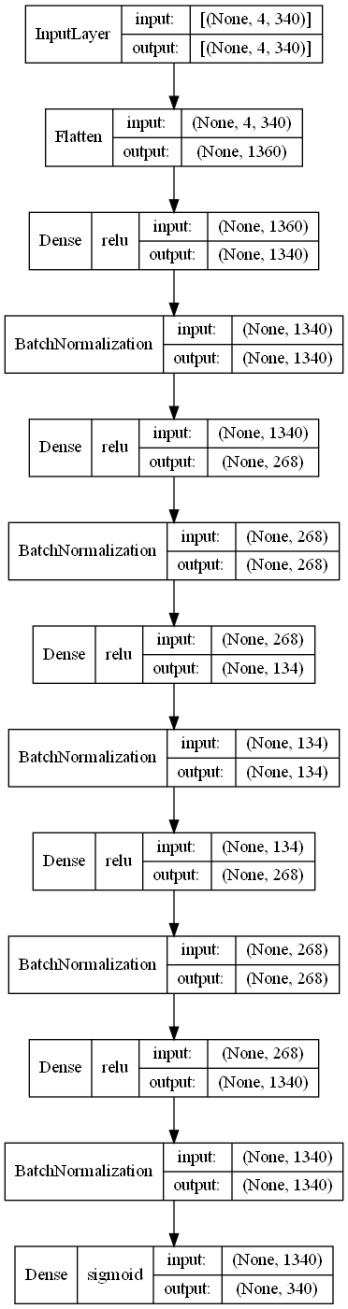

Figure 11: Architettura modello 1, ricostruttore 4 ad 1

Parametri	Tipo	Valori
Kernel inizializer	he_uniform	-
Ottimizatore	Adam	lr=$7 \cdot 10^{-5}$
Kernel regulizer	L2	$5 \cdot 10^{-6}$
Batch size	-	50

Table 4: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 47395 su 51231 totali, corrispondente ad un 92.512% dell'insieme.

Modello 2

Il seguente modello è contenuto nella cartella "TestLogs/Seq5/20220718-101159" allegata a questo report.

Prestazioni in fase di addestramento

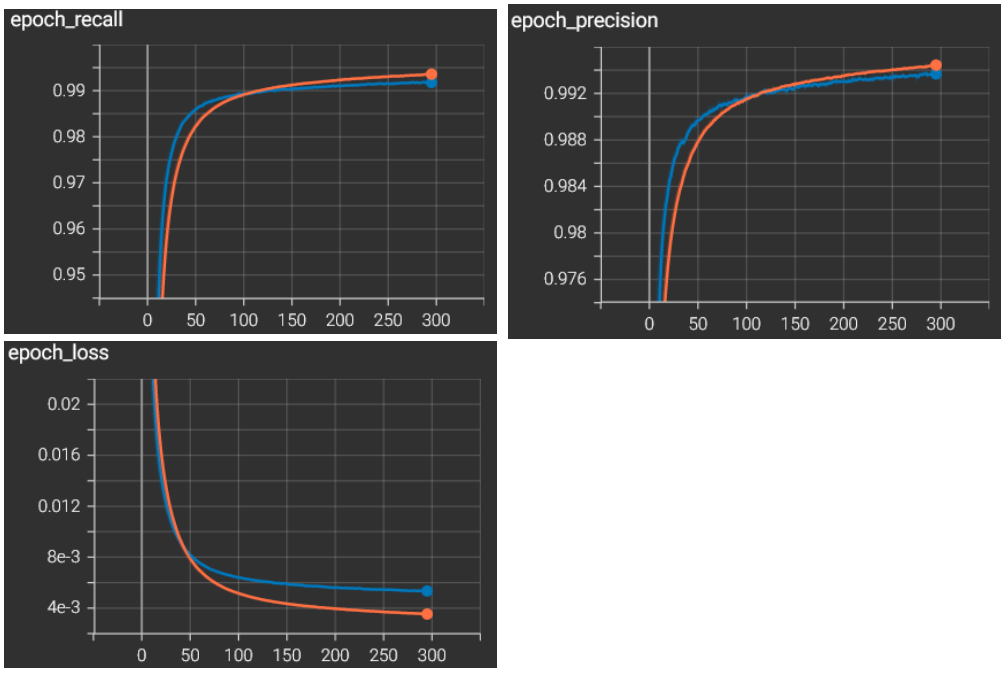

Figure 12: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

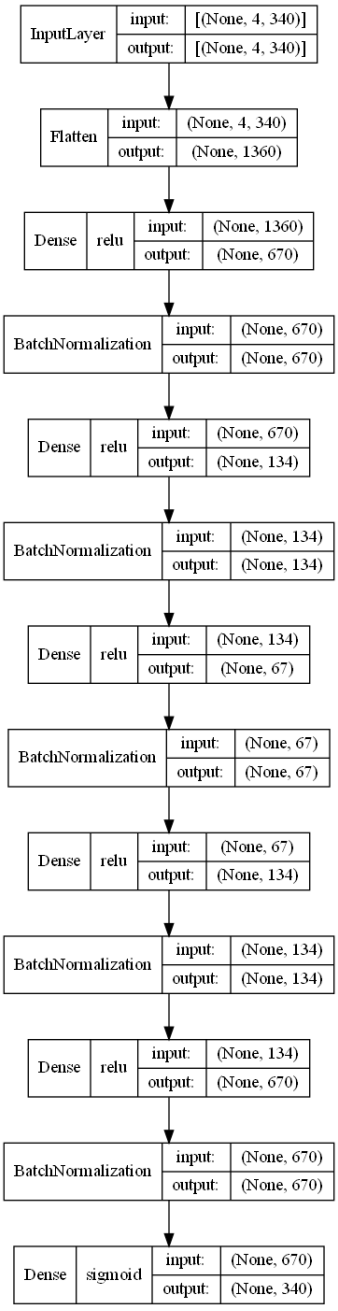

Figure 13: Architettura modello 2, ricostruttore 4 ad 1

Parametri	Tipo	Valori
Kernel inizializer	he_uniform	-
Ottimizatore	Adam	lr=$7 \cdot 10^{-5}$
Kernel regulizer	L2	$5 \cdot 10^{-6}$
Batch size	-	50

Table 5: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 46457 su 51231 totali, corrispondente ad un 90.681% dell'insieme.

Conclusione

I risultati sono soddisfacenti, così come il rimpicciolimento raggiunto, facendo concludere che l'esperimento possa considerarsi un successo e che l'obiettivo preposto sia stato raggiunto.

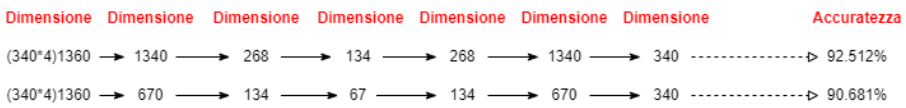

Figure 14: Recap delle reti realizzate per il secondo task, e relativi risultati

5 Task 3- Ricostruttore 6 a 1

Data una sequenza di 7 vettori, L'obiettivo è stato quello di creare un modello in grado di ricostruire il vettore centrale della sequenza, fornendo in input i restanti 6.

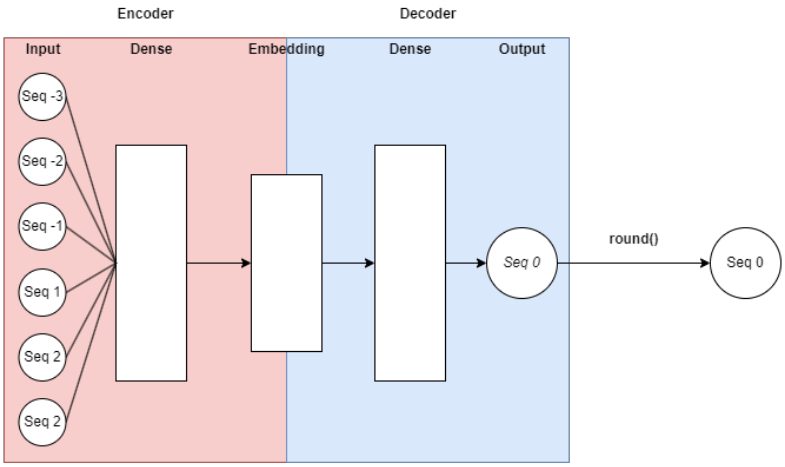

Figure 15: Struttura della rete per il terzo task

5.1 Studio e creazione del modello

- Numero di layer: In questo caso avendo un input di 2040 valori (340x6) si è optato a testare architetture su reti più profonde rispetto alle precedenti: alla fine i risultati migliori si sono avuti utilizzando un nu;ero di layer (di attivazione) interni compresi tra 3 e 5.
- Funzioni di attivazione: anche in questo caso la ReLU è stata la predefinita;
- Numero di unità: si è sempre cercato di mantenere una configurazione simmetrica della rete;
- Ottimizzatori: sono stati utilizzati adam ed Nadam;
- Metodi di regolarizzazione: in questo specifico task si è registrata un'elevata tendenza nell'andare in overfitting; Si è quindi proceduto ad incrementare il testing di strumenti di regolarizzazione con i seguenti risultati:
  - Early stopping: metodo fondamentale che è entrato in funzione nella maggior parte dei casi;
  - kernel inizialization: anche qui discorso che vale come nei casi precedenti: "he uniform" o "he normal" per i livelli con relu;

- Kernel regolarization: utilizzo di funzioni quali L1,L2, L1L2, alla fine la più efficiente si è rivelata essere la L2 con valori di iper-parametro comprese tra 1e−4 e 1e−10, ma non è bastato per avere dei risultati significativi;
- Dropout: sono stati effettuati tentativi di inserimento in tutte le posizioni della rete con percentuali comprese tra lo 0.05% e 0.4%, anche se si sono notati miglioramenti in generale sul rapporto train-validation, questi non si sono stati sufficienti per ottenere prestazioni finali soddisfacenti;
- Batch normalization: strumento che si è rivelato molto buono anche in questo task: la loss si è sempre comportata meglio con il suo utilizzo rispetto che senza, è stata applicata dopo ogni layer (ad esclusione di quello di output);

Ciononostante, l'applicazione delle seguenti tecniche anche congiunte, non è riuscita ad apportare significativi miglioramenti in fase di testing. Il fenomeno dell'overfitting ha contribuito in modo significativo alle prestazioni finali dei modelli.

5.2 Risultati del Task 3

Di seguito vengono riportati i 2 modelli che hanno prodotto i migliori risultati:

Modello 1

Il seguente modello è contenuto nella cartella "TestLogs/Seq7/20220803-235521" allegata a questo report.

Prestazioni in fase di addestramento

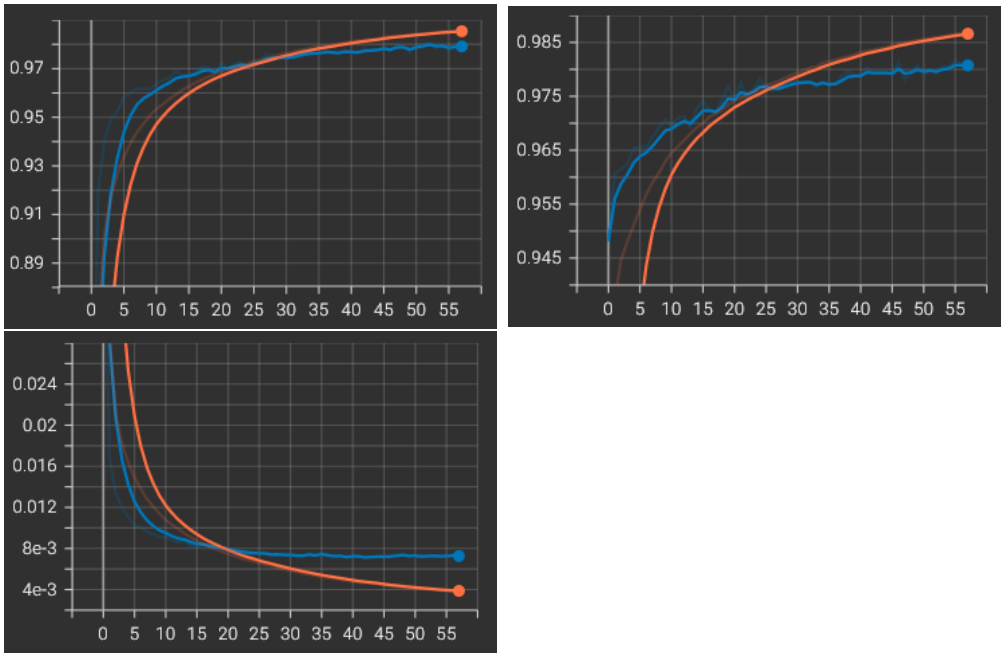

Figure 16: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

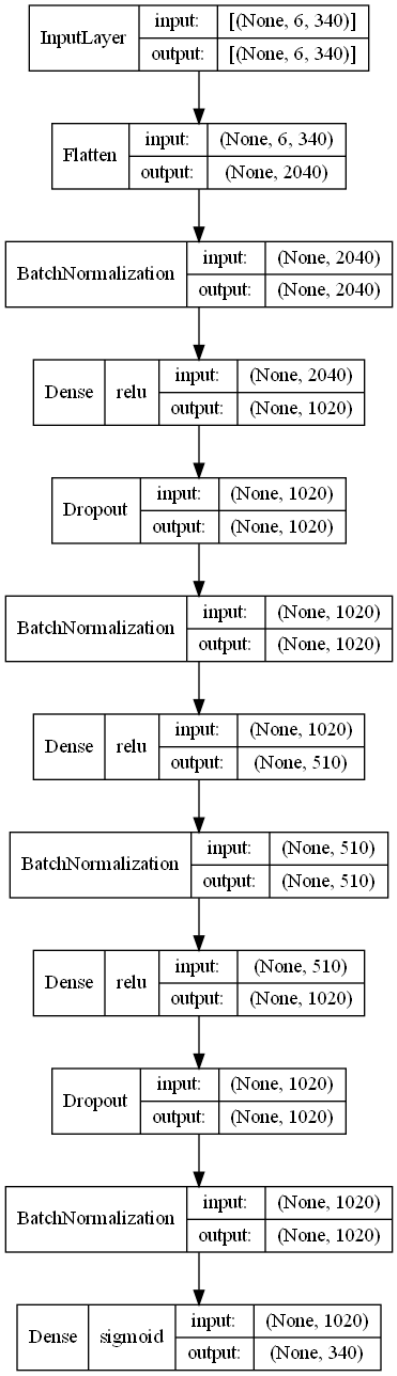

Figure 17: Architettura modello 1, ricostruttore 6 ad 1

Parametri	Tipo	Valori
Kernel inizializer	he_normal	-
Optimizers	Nadam	$lr = 0.009$, $beta\_1 = 0.95$
Kernel regulizer	L2	1e-10
Batch size	-	400

Table 6: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 23468 su 33459 totali, corrispondente ad un 70.14% dell'insieme.

Modello 2

Il seguente modello è contenuto nella cartella "TestLogs/Seq7/20220811-123712" allegata a questo report.

Prestazioni in fase di addestramento

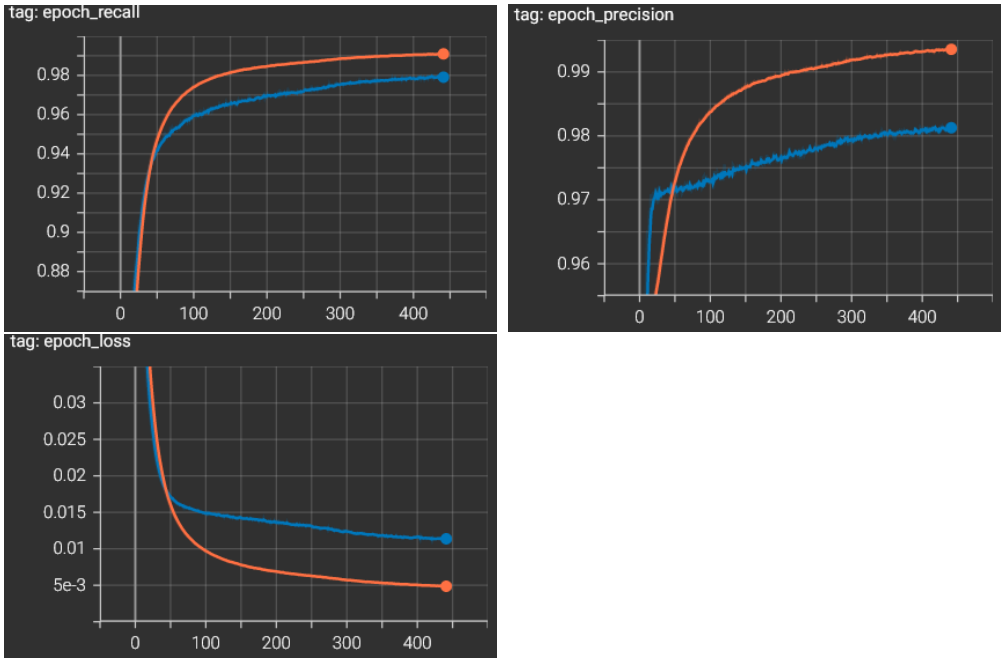

Figure 18: Andamento di recall, precision e loss su train e validation sulla base delle epoche.

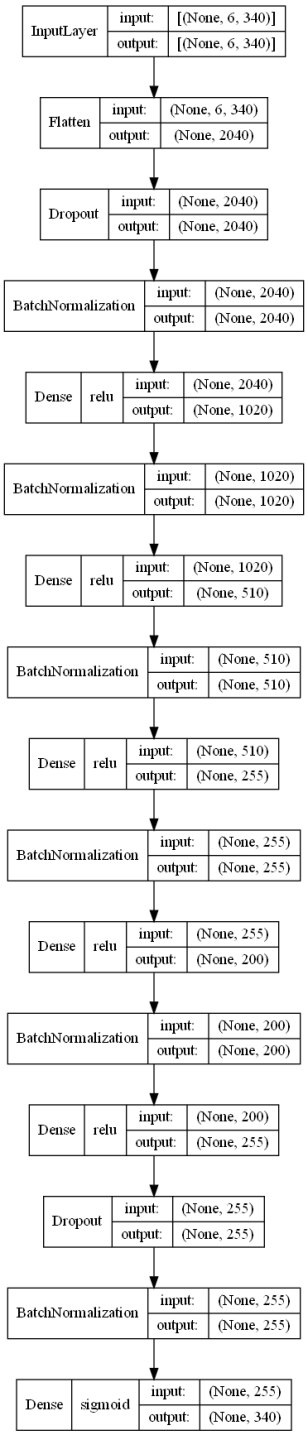

Figure 19: Architettura modello 2, ricostruttore 6 ad 1

Parametri	Tipo	Valori
Kernel inizializer	he_normal	-
Optimizer	adam	$lr = 0.0003700088651332496$
Kernel regulizer	L2	$1.967657265571901e-06$
Dropout	-	0.05 entrambi
Batch size	-	1000

Table 7: Ulteriori parametri del modello

Risultati finali sul testset:
Percentuale dei vettori ricostruiti correttamente: 24527 su 33459 totali, corrispondente ad un 73.305% dell'insieme. In media la maggioranza dei test condotti ha portato ad avere dei valori di ricostruzione non oltre l'intervallo di 50%-60%.

Conclusione

In questo task si è registrata notevole difficoltà ad ottenere un modello con prestazioni migliori rispetto alla media, ciò favorito da un notevole overfitting non registrato nei casi precedenti. C'è però da sottolineare come l'input questa volta avesse una dimensione di 2040, il 50% in più rispetto al task precedente, con un elevato livello di sparsità di valori (molti 0 e pochissimi 1). Un numero così alto di "0" ed una dimensione di input così elevata ha sicuramente contribuito a rendere particolarmente difficile la ricostruzione del vettore centrale della sequenza attraverso un architettura densa.

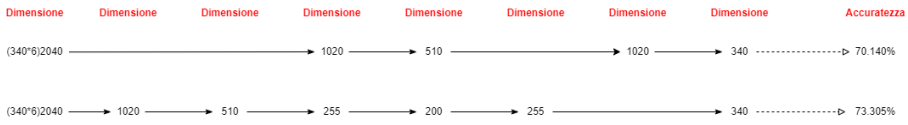

Figure 20: Recap delle reti realizzate per il terzo task, e relativi risultati