Machine Learning: Implementazione rete neurale per la predizione della struttura secondaria delle proteine

First Author
Bernardo Tiezzi
E-mail address
bernardo.tiezzi@stud.unifi.it

Abstract

La seguente relazione descrive l'elaborato realizzato per l'esame finale del corso di Machine Learning, appartenente al corso di Laurea Magistrale in Ingegneria Informatica presso l'Università degli Studi di Firenze. Il task realizzato riguarda la predizione della struttura secondaria delle proteine su 8 classi. A tal fine sono state implementate varie architetture software e svolti numerosi esperimenti, eseguendo un confronto con lo stato dell'arte.

1. Introduzione

Una proteina è una catena di amminoacidi, costituita da 20 tipologie di amminoacidi differenti, noti anche come residui. La dimensione delle proteine esistenti risulta molto variabile: si passa da un limite inferiore di 40-50 residui, necessari a svolgere un'attività funzionale, a diverse migliaia per le proteine plurifunzionali e strutturate. Comunemente la stima di media lunghezza per una struttura proteica è di circa 300 amminoacidi.
Le proteine presentano 4 differenti tipologie di struttura: la **struttura primaria** è determinata dalla catena amminoacidica connessa da legami peptidici; la **struttura secondaria**, oggetto di nostro interesse, si riferisce a sottostrutture derivate dalla disposizione nello spazio della sequenza proteica, dalla forma regolare e ripetitiva, ottenuta grazie ai legami idrogeno; la **struttura terziaria** rappresenta la forma globale di una singola molecola proteica, generata dalla relazione spaziale di varie strutture secondarie; infine, la **struttura quaternaria**, risulta dall'unione di più molecole, le cui funzioni prendono parte al compito globale svolto dal complesso. La predizione della struttura secondaria delle proteine risulta quindi essere un problema imprescindibile, al fine di conoscere la struttura tridimensionale della catena peptidica e le rispettive funzioni svolte. Solitamente la struttura secondaria è caratterizzata da 3 tipologie di classi: **alfa-elica**, **struttura a foglio-beta** e **struttura a bobina**. I problemi di classificazione su tre classi sono stati a lungo trattati nel corso degli anni, ottenendo risultati molto validi in termini di accuratezza. Recentemente l'attenzione, riguardo la predizione della struttura secondaria, si è spostata dalla classificazione su 3 classi, alla classificazione su 8 classi, poichè consente di ottenere maggiori informazioni sulle strutture utilizzabili in una vasta varietà di applicazioni. Il problema che tratteremo si concentrerà sulla definizione di architetture per ottenere un'adeguata classificazione su 8 classi delle sequenze proteiche.

2. Lo Stato dell'Arte

Da quanto spiegato in precedenza si evince che la predizione della **struttura secondaria** delle proteine risulta necessaria per l'individuazione delle strutture realizzate da segmenti locali dei residui peptidici. Diversi studi condotti sul tema hanno tuttavia evidenziato che, in aggiunta alle

dipendenze stabilite **localmente** tra i vari amminoacidi della sequenza, un ruolo indispensabile è svolto da quelle dipendenze stabilite **globalmente**, attraverso l'intera catena amminoacidica. Per esempio, una struttura a foglio beta (beta-strand) è in grado di stabilire legami idrogeno con altre beta-strands, le quali possono essere molto distanti l'una dall'altra, all'interno di una sequenza proteica. Nel corso degli anni sono stati sviluppati numerosi metodi per catturare alcune tipologie di dipendenze spaziali, ottenendo buoni risultati. Tuttavia per migliorare ulteriormente l'accuratezza, un'architettura deve essere in grado di individuare anche le dipendenze più complesse. Tra i lavori più famosi degli ultimi anni è noto quello di J. Zhou ed Olga G. Troyanskaya [2014][3] che, tra le altre cose, fornisce una delle migliori fasi di prepocessing dei dati appartenenti al dataset generato da Cull PDB server, tra i database più comuni appartenenti a questo ambito di ricerca. Il lavoro descitto nell'articolo impiega reti neurali convoluzionali e modelli generativi per combinare tra di loro rappresentazioni globali e locali. L'accuratezza prodotta dal loro modello su 8 classi (Q8), testasto sul dataset *CB513*, è pari al 66.4%. Uno dei lavori su cui si è ispirato maggiormente il seguente progetto è di M. R. Uddin, S. Mahbub et al. [2020][1], che propone di realizzare un metodo per la predizione Q8, incorporando il meccanismo di self-attention (un concetto derivato dal NLP), con una rete Deep3I (Deep Inception-Inside-Inception); quello che si ottiene è una *deep network* in grado di catturare interazioni a corto e lungo raggio tra i residui amminoacidici. L'accuratezza che si ottiene è del 71.78% sul dataset *CASP12*, mentre non è nota l'accuratezza sul dataset *CB513*. Ai fini di compiere un lavoro più completo si è preso spunto da due ulteriori articoli, Zheng Li e Yizhou Yu[4], Buzhong Zhang e Jinyan Li[2], per realizzare un'architettura basata su reti convoluzionali in cascata con reti neurali ricorrenti in modo tale da catturare le dipendenze locali tramite le convoluzioni e le dipendenze globali tramite le RNNs. L'accuratezza Q8 ottenuta è rispettivamente del 71.4% e del 69.7%, testando il modello su CB513. Si è dunque deciso di realizzare due differenti tipologie di rete: la prima con self-attention+Deep3I e la seconda con CNNs+RNNs, andando ad analizzare i risultati.

3. Dataset Impiegati

Per eseguire l'addestramento delle reti neurali e le rispettive fasi di testing si è andati ad utilizzare i dataset derivati da *CullPDB*, che, come precedentemente specificato, risulta tra i dataset più completi open-source disponibili per il tema affrontato. Questo dataset contiene oltre 5000 sequenze differenti prodotte utilizzando il server PISCES, che consente di estrarre le sequenze proteiche dalla *Protein Data Bank* (PDB). CullPDB è stato successivamente processato e filtrato come viene spiegato da Zhou et al. nell'articolo precedentemente citato. Per eseguire la fase di testing è stato utilizzato il dataset *CB513*, che risulta essere completamente disgiunto dalla versione filtrata di *cullpdb+profile5916filtered*, da cui sono stati rimossi anche le sequenze duplicate. I dati sono raccolti all'interno di file *numpy* nella struttura N proteine x K features; per i dataset può essere rieseguito un reshape nella forma: N proteine x 700 residui x 57 features. Le features sono disposte nel seguente ordine:

- **[0,22)**: sono i residui amminoacidici
- **[22,31)**: rappresentano le etichette della struttura secondaria delle proteine
- **[31,33)**: sono i terminali N- e C-
- **[33,35)**: indica l'accessibilità dei solventi, sia assoluta che relativa (normalizzata dal massimo valore dell'accessibilità di una proteina)
- **[35,57)**: indica il profilo della sequenza

Si noti che, data la variabilità della lunghezza delle catene peptidiche, è stato posto il valore 700 come limite superiore della lunghezza delle varie sequenze. Se quest'ultime presentano una dimensione inferiore, subiscono uno zero padding. L'input X è costituito dalle features [0,22) e [35,57), mentre l'output è presentato come un vettore di probabilità di lunghezza 8, che rappresenta

la classificazione Q8. Per fornire alla rete l'input Y della sequenza, vengono utilizzate le features [22,31) del training set.

4. Architutture Implementate

4.1. SAINT: Deep3I and Self-Attention

La prima architettura reimplementata è quella presentata nall'articolo, già precedentemente indicato, di M. R. Uddin et al. (2020), che descrive una rete deep feed-forward convoluzionale che sfrutta il meccanismo di self-attention.

4.1.1 Self-Attention

Il meccanismo di self-attention tende a prestare attenzione a parti specifiche dei dati forniti in input o delle features, durante la generazione della sequenza di output. L'implementazione definita nell'articolo prende spunto dalla versione presentata da Vaswani et al. (2017), in cui ogni sequenza di input è trasformata in 3 differenti vettori *query*, *key* and *values*, da 3 funzioni differenti. Ogni output generato è una somma pesata del vettore di *values*, dove i pesi sono calcolati in base alla compatibilità dei vettori *query* e *key*, nota anche come funzione di *compatibilità*. Il modulo self-attention prende in input le features del layer precedente x assieme alla posizione assoluta o relativa dei residui nella sequenza, poichè durante il procedimento tale informazione viene persa. La funzione di compatibilità utilizzata è la *scale-dot product attention*, così definita:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

dove S è il prodotto Q e K sono le matrici di *query* e *key* e d_k è la dimensione dello spazio delle feature K che funge da normalizzatore. In seguito i pesi sono ottenuti utilizzando la funzione di attivazione *softmax()* a cui viene aggiunta una *BatchNorm()*, ottenendo l'output della funzione di compatibilità. Infine una somma pesata element-wise del vettore di input x con il vettore ottenuto dal meccanismo di attention fornisce l'output finale del modulo.

4.1.2 Inception e Deep Inception Module

L'*Inception* module è una tecnica spesso utilizzata all'interno delle reti convoluzionali per consentire di ottenere un costo computazionale più efficiente e reti più profonde andando a ridurre la dimensionalità grazie all'impiego di convoluzioni 1x1. Tra le altre cose tale modulo consente di ridurre l'overfitting sui dati. L'idea alla base è quella dunque di eseguire più convoluzioni utilizzando filtri kernel di dimensione differente, posizionati per operare allo stesso livello e non sequenzialmente. La Fig.1 illustra come è stato progettato l'inception module.

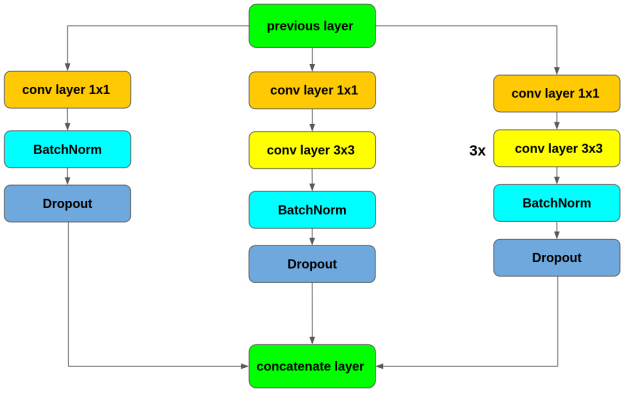

Figure 1: Inception Module

In seguito viene realizzata una rete *Deep3I* che va ad eseguire, allo stesso livello, tre sequenze dell'Inception, allo scopo di recuperare sia le dipendenze locali che le dipendenze globali ed ottenere in tal modo la miglior accuratezza possibile. Tuttavia i layer convoluzionali non sono in grado di catturare le dipendenze globali tra le features di una sequenza. Spesso quindi risulta necessario andare a realizzare reti molto profonde, portanto un significativo impatto sui costi computazionali. Per ovviare a tale problema si va ad aggiungere la Self-Attention dopo aver eseguito ogni branca della Deep3I ed eseguendo una concatenazione dei dati ottenuti da ciascun ramo, si ottiene l'output del modulo noto come *2A3I*. La tecnica descritta è illustrata in Fig.2(c).

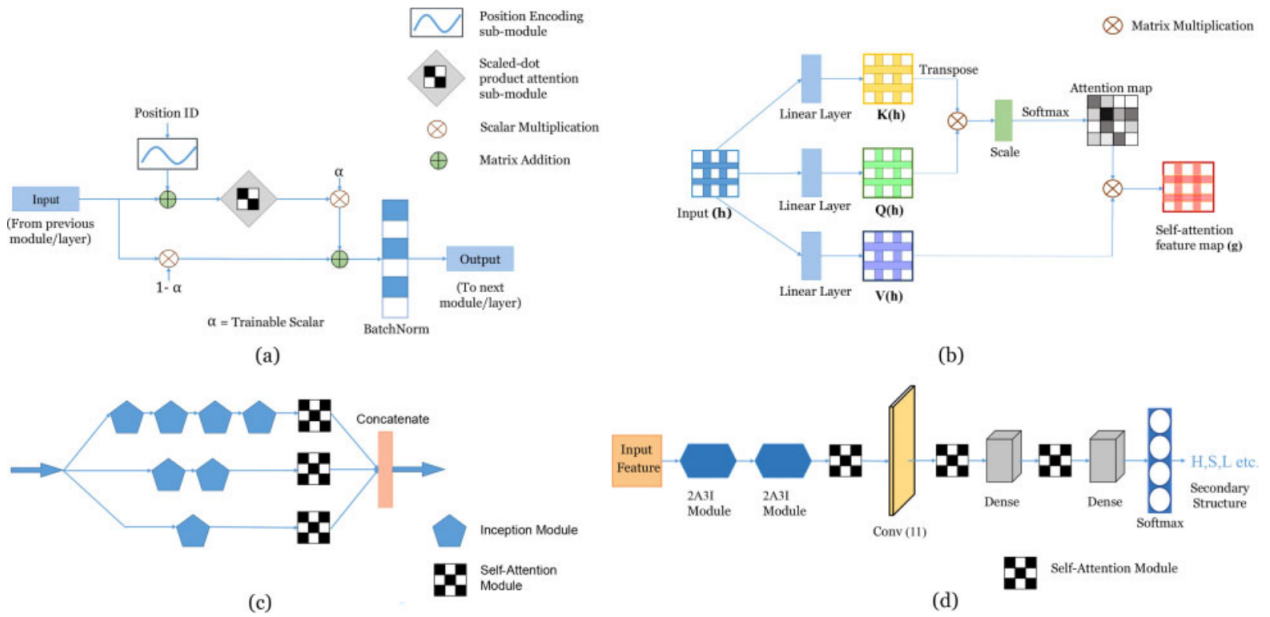

Figure 2: Architettura Saint

4.1.3 Overview di Saint

In Fig.2 è illustrata una panoramica dell'architettura di Saint. Come si può vedere, Saint inizia con due moduli *2A3I* consecutivi, seguiti da un modulo Self-Attention, al fine di catturare quelle interazioni sfuggite in precedenza. L'aggiunta di un layer convoluzionale con kernel di 11x11 è un ulteriore ausilio nella ricerca delle interazioni a lunga distanza. Un modulo self-attention è posto sia dopo aver eseguito la convoluzione sia tra i due dense layer che precedono la funzione di attivazione *softmax()* che definisce il vettore di output dell'architettura completa.

4.2. CNNs + RNNs

Per realizzare la rete si è preso spunto dalle architetture presentate negli articoli di Zheng Li e Yizhou Yu, Buzhong Zhang e Jinyan Li, andando quindi a riprodurre una RNNs per la classificazione multiclasse. L'idea è quella di suddividere l'estrazione delle interazioni a corto o lungo raggio tra le features della sequenza. La rete è dunque strutturata impiegando layer convoluzionali, con dimensioni del kernel differenti, che operano allo stesso livello e, i cui output vengono concatenati in uscita. L'implementazione è simile a quella realizzata per l'architettura SAINT. In Fig.3 ne viene mostrata la struttura.

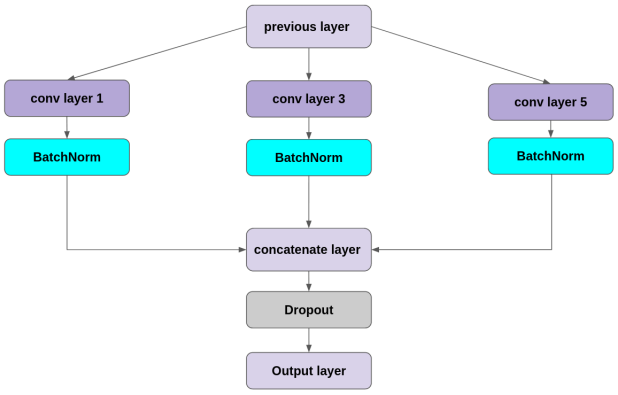

Figure 3: Convolution module

L'utilizzo di filtri differenti per eseguire le convoluzioni è utile per identificare dipendenze locali con finestre di varia grandezza. Dopo ogni convoluzione si esegue una *BatchNorm()* ed una concatenazione delle features. Infine, attraverso un dropout, si ottiene l'output del modulo.

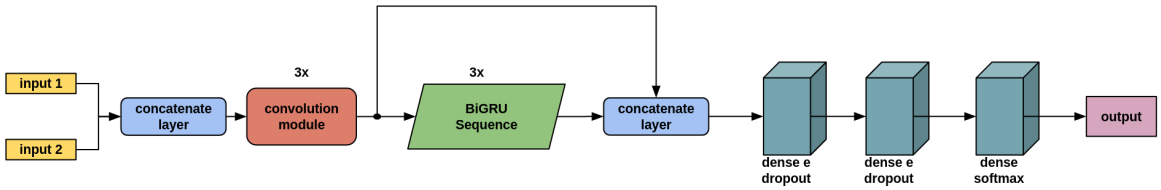

Figure 4: CNNs+RNNs

4.2.1 RNNs

Per individuare le dipendenze globali si fa affidamento su architetture proprie delle reti neurali ricorrenti quali i Gater Recurrent Unit (GRU). Il GRU è una variante delle RNNs tradizionali che risulta utile per individuare le interazioni tra i dati a lunga distanza l'uno dall'altro; tale struttura è costituita da due gate interni: reset gate e update gate. Il reset gate individua nella sequenza le features di cui la rete non deve tenere memoria perchè scorrelate. L'update gate al contrario individua quelle features che risultano importanti per il futuro, copiando in dettaglio il passato e risolvendo così il problema del *vanishing gradient*, che colpisce le reti tradizionali. L'utilizzo della funzione *bidirectional* del GRU consente di monitorare le fueatures da entrambe le direzioni della sequenza. Al termine della fase di individuazioni delle relazioni locali e globali tra le features viene eseguita una concatenazione tra l'output del *convolution module* e l'output del *BiGRU layer*; il tensore generato viene processato da due dense layer, tra i quali sono posizionati opportunamente i dropout layer. Infine, viene eseguito il *softmax()*, che restituisce l'output finale dell'RNNs.

5. Risultati sperimentali

Di seguito sono presentati le tabelle dei risultati ottenuti, andando a testare le architetture implementate sul dataset di Testing **CB513**. Per ogni rete implementata sono stati riportati i valori di F1-Score, precision e recall, su ognuna delle classi individuate. Il campo **RNNs+SA** della Tabella 1 fa riferimento ad un tentativo svolto al fine di aumentare l'espressività della RNNs implementata. Per individuare con maggior efficacia le features globali, si è andati ad aggiungere il modulo self-attention, utilizzato dall'architettura Saint, all'interno della rete neurale ricorrente presentata in figura 4. In tabella 2 si è eseguito un confronto sull'accuratezza delle reti implementate e le reti originali. Non è stato possibile inserire i dati riguardanti Saint poichè, all'interno del paper di riferimento, la rete non era stata testata sul dataset CB513. Per ogni rete, sono mostrati i grafici definiti durante la fase di addestramento sul dataset *cullpdb+profile5916filtered*.

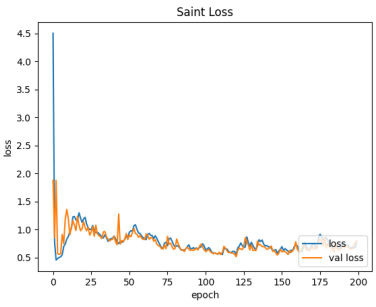

Figure 5: Saint Loss

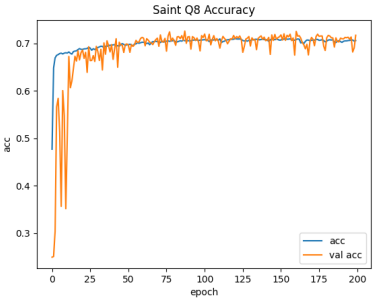

Figure 6: Saint Accuracy

Table 1: Risultati F1, Precision e Recall Calcolati sul dataset **CB513**

	Precision				Recall				Frequency
	SAINT	**RNNs**	**RNNs**	**RNNs + SA**	**SAINT**	**RNNs**	**RNNs**	**RNNs + SA**	
Metodi			**Riprodotto**	**Riprodotto**			**Riprodotto**	**Riprodotto**	
L	0.529	0.573	0.552	0.558	0.653	0.662	0.668	0.654	0.58
B	0.75	0.558	0.726	0.486	0.004	0.049	0.005	0.087	0.01
E	0.719	0.739	0.764	0.756	0.814	0.862	0.818	0.829	0.76
G	0.438	0.432	0.404	0.428	0.208	0.331	0.347	0.316	0.26
I	0	0	0	0	0	0	0	0	0
H	0.823	0.836	0.861	0.847	0.917	0.927	0.919	0.924	0.87
S	0.498	0.521	0.553	0.535	0.156	0.275	0.194	0.187	0.24
T	0.534	0.549	0.549	0.538	0.475	0.572	0.540	0.543	0.5

Table 2: Accuratezza calcolata su dataset **CB513**

	SAINT	**RNNs**	**RNNs**	**RNNs + SA**
Accuratezza	**Riprodotto**		**Riprodotto**	**Riprodotto**
Q8	0.679 ± 0.002	0.697 ± 0.002	0.701 ± 0.002	0.701 ± 0.002

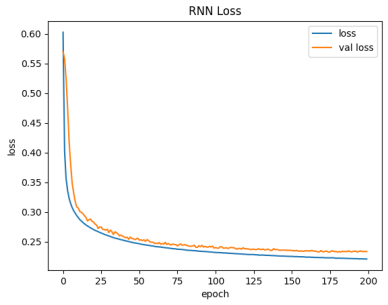

Figure 7: RNNs Loss

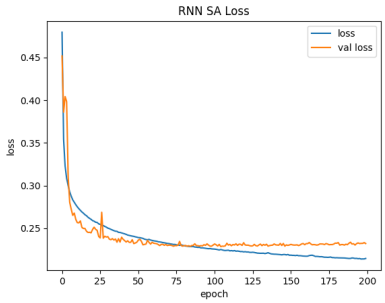

Figure 9: RNNs+SA Loss

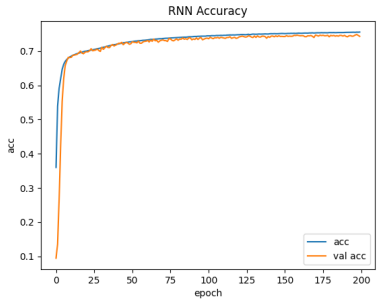

Figure 8: RNNs Accuracy

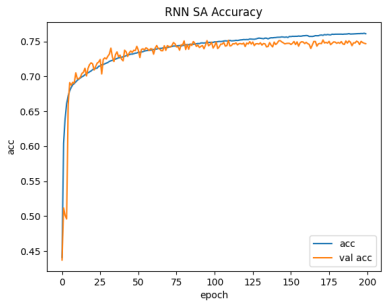

Figure 10: RNNs+SA Accuracy

References

[1] M. R. U. S. M. M. S. R. M. S. Bayzid. Saint: self-attention augmented inception-inside-inception network improves protein secondary structure prediction. *Bioinformatics*, 36(17):4599–4608, 2020.

[2] J. L. Buzhong Zhang and Q. Lu. Prediction of 8-state protein secondary structures by a novel deep learning architecture. *Bioinformatics*, 2018.

[3] O. G. T. Jian Zhou. Deep supervised and convolutional gen-

erative stochastic network for protein secondary structure prediction. *JZTHREE@PRINCETON.EDU*, 2014.

[4] Y. Y. Zhen Li. Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. 2016.