Machine Learning: predizione della struttura secondaria delle proteine in 8 classi

Marioemanuele Ghianni
E-mail address
marioemanuele.ghianni@stud.unifi.it

Abstract

La relazione che verrà presentata fa riferimento al progetto finale del corso magistrale in ingegneria informatica di Machine Learning. Il task preso in esame è la "predizione della struttura secondaria della proteina in 8 classi" basandosi dunque unicamente sulla conoscenza della sequenza degli aminoacidi.L'implementazione è stata fatta in linguaggio di programmazione Python tramite le note librerie Tensorflow e Keras.
A tal proposito verranno presentati una serie di esperimenti e architetture usate per tentare di risolvere questo task e infine verranno messe a confronto con lo stato dell'arte, verranno tratte alcune conclusioni e saranno indicati possibili sviluppi futuri.

1. Introduzione alla SSP

l'SSP (Secondary Structure Prediction) è un problema fondamentale e un passo necessario per determinare la struttura tridimensionale della proteina e la sua funzione.
L'impiego di tecniche di predizione automatica permette dunque un notevole vantaggio in termini di tempistiche, e quindi di costi, per la ricerca in questo ambito.
Ogni proteina in natura presenta diversi livelli di organizzazione che si integrano originando la sua conformazione tridimensionale specifica.
In particolare, le proteine hanno 4 livelli di struttura: un *livello primario*, determinato dalla sola sequenza amminoacidica nella catena polipeptidica; un *livello secondario* che è quello di nostro interesse e che è dato dalla catena polipeptidica che assume nello spazio una dispozione regolare e ripetitiva stabilizzata da legami idrogeno; un *livello terziario* che è dato appunto dal legame di varie strutture secondarie e infine, un *livello quaternario* che è determinato dall'insieme di strutture terziarie ognuna determinata da più catene polipeptidiche.
Una panoramica di ciò è visibile in figura 1.

Ci concentreremo dunque sul livello secondario che, solitamente, comprende 3 classi principali di struttura: l'**alfa-elica**, la **struttura a foglietto beta** e la **struttura a bobina**.
In questo caso si parla dunque di predizione della struttura secondaria in 3 classi ed è un problema che è stato ampiamente trattato negli anni e si hanno risultati molto soddisfacenti in termini di accuratezza.
Il problema che invece andremo a trattare è invece la predizione della struttura secondaria in 8 classi, ovvero un problema più generale rispetto alla predizione in 3 classi e quindi più complesso, oggetto di ricerca anche in questi ultimi anni.

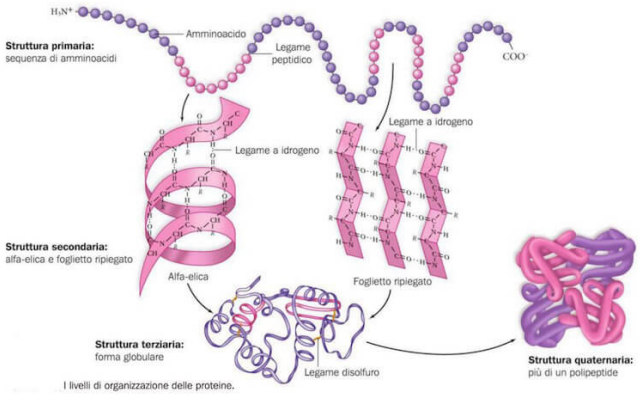


Figure 1: overview delle strutture delle proteine.

.

2. Stato dell'arte

Predire la struttura secondaria di una proteina significa dunque, in sostanza, determinare gli stati strutturali di segmenti locali di amminoacidi residui ma, da diversi studi,

emerge come, oltre a **dipendenze locali**, la struttura secondaria presenta anche **dipendenze globali**; ad esempio, una struttura a foglietto beta è stabilizzata da legami idrogeno formati con altri filamenti beta che possono essere molti distanti l'uno dall'altro nella sequenza proteica.
E' proprio quest'ultima infatti la "sfida" più grande nell'affrontare questo tipo di problema.
Negli anni ci sono stati vari approcci di machine learning per modellare dipendenze locali e quindi sfruttando reti neurali con finestre temporali che hanno avuto dei buoni risultati ma, per migliorare ulteriormente l'accuratezza, si è reso necessario apprendere una rappresentazione globale della sequenza dei residui.
Uno dei lavori più importanti degli ultimi anni è di Zhou and Troyanskaya, del 2014 [4] che ha contribuito anche a un filtraggio e una migliore fase di preprocessing dei dati dal database più comune in questo ambito di ricerca.
Nel loro lavoro vengono impiegate reti neurali convoluzionali e modelli generativi supervisionati per combinare, appunto, una rappresentazione ad alto livello con una rappresentazione locale.
In termini di accuratezza Q8, questo lavoro riporta un'accuratezza del **66.4%** sul dataset *CB513*.
Un'altro importante lavoro su cui si è anche maggiormente ispirato questo progetto è il lavoro di Zhen Li e Yizhou Yu [2] che propone un'architettura basata su reti convoluzionali in cascata con reti neurali ricorrenti in modo tale che le prime si occupino di apprendere features locali e le RNNs vadano a cogliere dipendenze a lungo termine.
Entrambe le features vengono poi concatenate e con questo lavoro si raggiungono livelli di accuratezza maggiori e che rappresentano lo stato dell'arte con un'accuratezza Q8 su CB513 del **69.7%**.
Relativamente recente è invece il lavoro di Vaswani et al. 2017 sull'architettura **Transformer** che parte come architettura per il Natural Language Processing ma che può essere estesa ad altri campi.
Oltre a descrivere un'architettura basata su CNNs+RNNs, si è dunque tentato di adattare anche il Transformer a questo tipo di task.

3. Dataset

Seguendo i vari lavori di predizione in questo campo, è stato selezionato come dataset, *CullPDB* che è uno dei piu' grandi e completi dataset open-source per questo tipo di problema.
Il dataset CullPDB contiene oltre 5000 sequenze non omologhe prodotte usando PISCES; un server pubblico per estrarre insiemi di sequenze di proteine dal PDB(Protein Data Bank).
Questo dataset è stato poi processato e filtrato come descritto nel lavoro di Zhou et al. precedentemente citato e dunque si è partiti da questo come training set di base.
Per quanto riguarda il testset, è stato usato *CB513* che è il piu' utilizzato nei vari lavori in quanto compatto e totalmente disgiunto dalla versione filtrata del dataset "cullpdb+profile6133filtered".
Inquadrando meglio i dati, nel dataset essi sono disposti nel formato numpy come (N proteine x k features) ed è possibile fare un reshape in (N proteine 700 aminoacidi e 57 features) dove 700 è imposto come limite superiore della sequenza degli amminoacidi e le sequenze piu' corte subiscono uno zero padding.
Per quanto riguarda le features, abbiamo :

- [0 , 22) i residui degli amminoacidi
- [22, 31) etichette della struttura secondaria
- [31, 33) i terminali N e C della proteina
- [33, 35) accessibilita' dei solventi, relativa e assoluta
- [35, 57) profilo della sequenza

Dove abbiamo quindi i dati da 0 a 22 e da 33 a 57 che costituiscono l'input x della rete mentre l'output deve essere un vettore di probabilita' di lunghezza 8 che rappresenta la classificazione negli 8 stati.
Di conseguenza, la sequenza da 22 a 31 è l'input y della rete su cui viene fatto il training.

4. Preprocessing

5. Architetture analizzate

Lo sviluppo delle architetture per la risoluzione di questo task è stato incrementale.
Inizialmente, la prima architettura analizzata è stata una semplice rete RNN feedforward composta, come in figura 2, da:

- un layer *Embedding* con 50 unità che mappa il features vector one-hot in input, in un vettore più denso a 50 elementi.
- un layer *Concatenate* che concatena il vettore embedded sopra ottenuto con l'altro input della rete.

- un layer bidirezionale *LSTM* con 64 unità e un *dropout* fissato a 0.2

- due layer *Dense* con rispettivamente 256 e 128 unità intervallati da un dropout a 0.5.

- infine, un layer *Dense* con 8 unità e attivazione *softmax* per costruire l'output della rete.

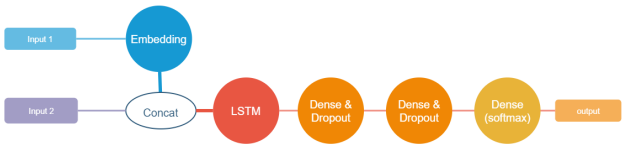


Figure 2: schema dell'architettura di partenza

L'accuracy Q8 ottenuta con questa architettura di base, sul testset *CB513*, è di circa **65,7 %** e risulta molto inferiore allo stato dell'arte ma comunque un buon punto di partenza.
Effettuando un passo indietro e sostituendo il layer LSTM bidirezionale con un layer LSTM base, anche raddoppiando il numero di unità, si arriva ad un'accuratezza solo del 58%.
Successivamente è stata ampliata l'architettura inserendo in cascata al layer LSTM altri 2 layer bidirezionali LSTM da 64 unità raggiungendo un'accuratezza media del **66.7 %**, paragonabile e leggermente migliore dell'architettura di Zhou e Troyanskaya.
Si è poi successivamente provato a sostituire i layer bidirezionali LSTM con GRU e si è potuto dunque fare un paragone; in particolare si è passati da 335 000 parametri a 270 000 con conseguente miglioramento della velocità di addestramento della rete di circa il 20% e un'accuratezza che è addirittura leggermente aumentata intorno al 66.9% di media mostrando come, per questo tipo di problema, le GRU siano più efficaci.
Particolare attenzione è stata rivolta poi al **padding**; seppur alcuni lavori [1] mostrino come, per reti RNNs, sia migliore un pre-padding rispetto ad un post-padding, in questo caso stiamo utilizzando delle BiGRU e quindi entrambi gli approcci hanno prodotto gli stessi risultati.
Usare il padding è una scelta che è sostanzialmente un tradeoff tra performance e accuratezza del modello e, essendo non ammissibile usare un batch size di 1 per i tempi di addestramento, una soluzione che è stata trovata inizialmente è quella del **Masking**.
Keras infatti, mette a disposizione un *Masking layer* in cui è possibile specificare i valori in input da ignorare (in questo caso le sequenze di 0 date dal padding); il layer creerà un tensore maschera booleano per istruire i layers successivi su quali valori "skippare" dalla computazione.
Entrambi gli input quindi passano attraverso un Masking Layer e la maschera viene propagata ai layer successivi in modo da ignorare i possibili effetti di "rumore" del padding.
Attraverso questa tecnica si hanno dei miglioramenti ma non così sostanziali, arrivando a un'accuracy Q8 media del 67.9% nell'architettura con 3 layer GRU.
In seguito, è stata estesa l'architettura seguendo in parte l'idea alla base del lavoro sopra citato di Zhen Li e Yizhou Yu; sono stati dunque aggiunti, dopo la concatenazione degli input, una serie di layer convoluzionali in cascata con filtri di dimensione diversa per ottenere le features locali.
In particolare è stato strutturato un "blocco convoluzionale" come in figura 3; l'input passa in parallelo su 4 diversi layer convoluzionali con kernel diversi da 1 a 11, ognuno di questi output passa attraverso una batch norm per poi venire concatenato con gli altri e con l'input iniziale del blocco.
Infine, si ha un dropout a 0.5 e si ottiene l'output del blocco.

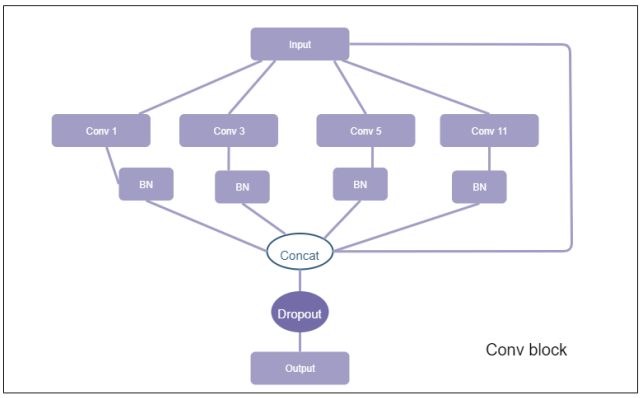


Figure 3: schema del blocco convoluzionale

Nell'architettura finale sono stati dunque aggiunti 3 di questi blocchi prima dei layer GRU e l'output di questi ultimi viene infine concatenato con l'output dei blocchi convoluzionali in modo da avere la combinazione tra features globali e locali come nel lavoro di Zhen Li et al.
L'architettura complessiva è visibile nello schema semplificato in figura **??**.
In particolare, sfruttando unicamente la concatenazione e non combinando gli output si raggiunge un accuracy Q8 media del **69.3% (+- 0.2)**, molto vicina all'accuracy di 69.4% del modello di riferimento senza le successive operazioni di *ensemble model* che han portato l'accuratezza

al 69.7%.
Combinando poi infine le features locali con quelle globali e inserendo un input fisso aggiuntivo, come nello schema finale, si arriva a un'accuratezza del **70% (69.998, per la precisione)** senza sfruttare l'ensemble model.
Questo risultato sembra indicare come questa architettura abbia una maggiore espressività e una migliore capacità di estrarre buone features locali attraverso i blocchi convoluzionali, se messa a paragone con il lavoro di Zhen Li.
Successivamente verrà presentata un'analisi più approfondita e accurata con dei test e dei risultati sperimentali su questa architettura.

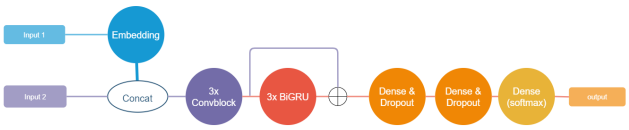


Figure 4: schema dell'architettura finale

Un'ultima architettura analizzata è quella del *Transformer* dal celebre lavoro di Vaswani et al. del 2017 [3].
Il Transformer è un modello di deep learning che sfrutta il meccanismo dell'*attenzione* pesando in maniera diversa la rilevanza di ogni parte dei dati in input.
E' usato principalmente nel campo del NLP (Natural Language Processing) ma è un' architettura flessibile che sta venendo applicata in vari campi tra cui anche la visione artificiale.
Per adattare i dati a questo task è stata necessaria una fase aggiuntiva di preprocessing.
L'intuizione è di partire dalla sequenza degli aminoacidi e convertire i dati numerici in lettere che rappresentano, appunto, i vari aminoacidi.
Stesso procedimento viene fatto per il target, ovvero la struttura secondaria, portando il vettore one-hot ad una lettera che corrisponde alla struttura.
Vengono creati così due vocabolari, uno per l'input e uno per il target, che associano ad ogni lettera un numero.
La sequenza source e la sequenza target vengono poi in tal modo date in pasto al Transformer strutturato come nel lavoro di Vaswani et al.
In seguito un overview dei risultati ottenuti con le varie architetture e un'analisi comparativa con i lavori simili per poi trarre delle conclusioni.

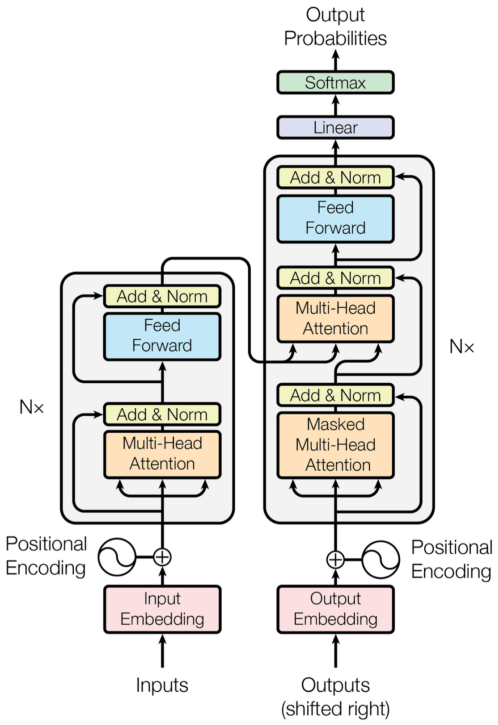


Figure 1: The Transformer - model architecture.

6. Risultati sperimentali

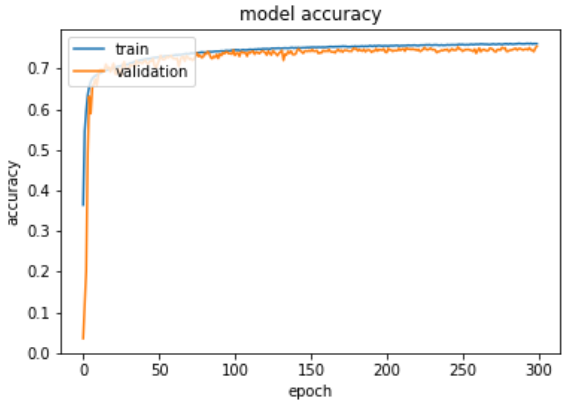
Di seguito sono presenti due tabelle per il confronto tra l'architettura sviluppata e i risultati dei precedenti lavori già citati.
In particolare è stata confrontata l'architettura riprodotta usando come test set parte del database usato anche per il training mentre in tabella 2 sono presenti i risultati utilizzando il test set di riferimento cb513.
Sempre su quest'ultimo è stato testato il Transformer che, nonostante non sia stato fatto particolare tuning, ha raggiunto un'accuratezza paragonabile alla DCRNN in 1/5 del tempo di addestramento.
Infine è presente uno scatter plot di riepilogo delle architetture e delle varianti utilizzate.

	Precision			Recall.			Frequency
	DCRNN reproduced	**DCRNN**	**GSN**	**DCRNN reproduced**	**DCRNN**	**GSN**	
Q8	**0.749**	**0.732**	**0.721**				
H	0.879	0.878	0.828	0.936	0.927	0.935	0.309
E	0.817	0.792	0.748	0.836	0.862	0.823	0.213
L	0.562	0.589	0.541	0.704	0.662	0.663	0.211
T	0.588	0.577	0.548	0.578	0.572	0.506	0.118
S	0.566	0.518	0.423	0.205	0.275	0.159	0.098
G	0.479	0.434	0.496	0.333	0.311	0.133	0.037
B	0.703	0.596	0.500	0.042	0.049	0.001	0.014
I	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 1: Tabella risultati usando come test set parte del database *cullPDB6133*.

	Precision			Frequency
	DCRNN reproduced	**DCRNN**	**GSN**	
Q8	**0.6998**	**0.694**	**0.664**	
H	0.850	0.836	0.831	0.309
E	0.764	0.739	0.717	0.213
L	0.567	0.573	0.518	0.211
T	0.552	0.549	0.496	0.118
S	0.548	0.521	0.444	0.098
G	0.405	0.432	0.450	0.037
B	0.633	0.558	0.000	0.014
I	0.000	0.000	0.000	0.000

Table 2: Tabella risultati sul test set di riferimento *cb513*.



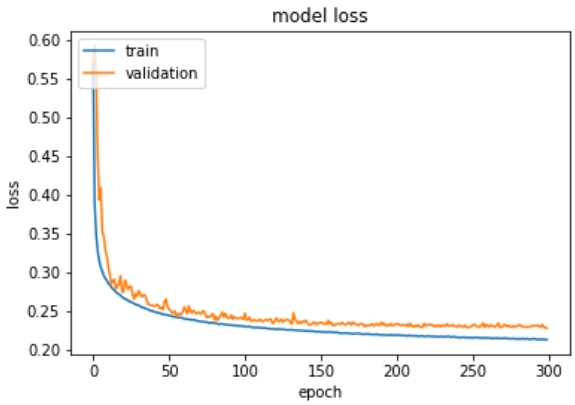


Figure 5: accuracy e loss dell'architettura riprodotta.

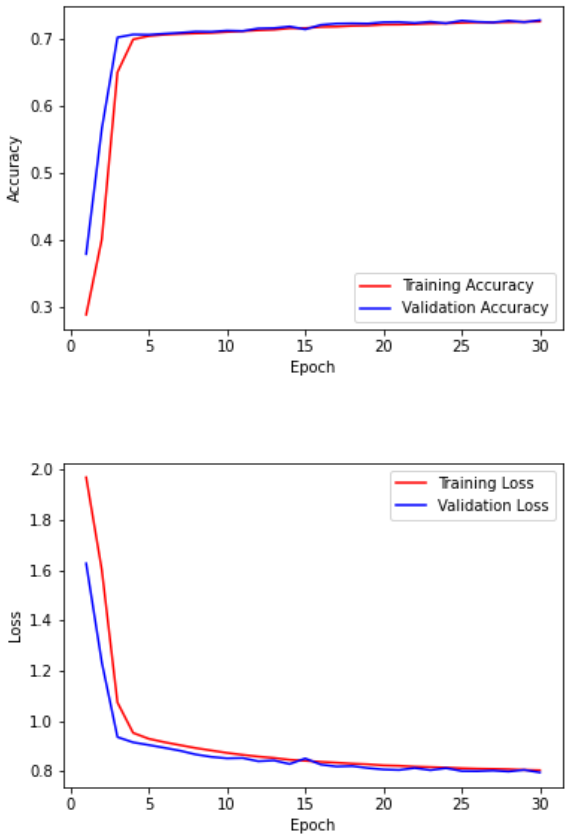


Figure 6: accuracy e loss dell'architettura Transformer.

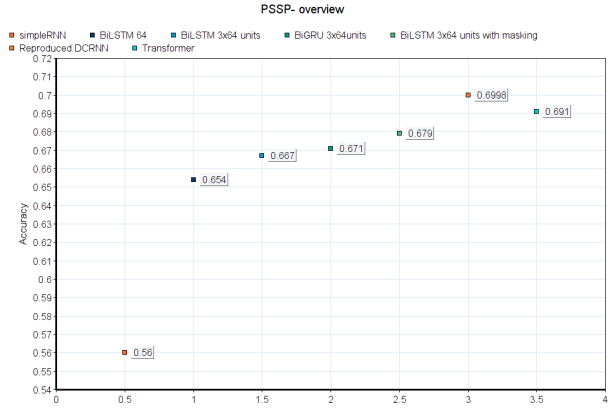


Figure 7: Scatter plot delle varie architetture utilizzate.

References

[1] Mahidhar Dwarampudi and N Reddy. *Effects of padding on LSTMs and CNNs*. Mar. 2019.

[2] Zhen Li and Yizhou Yu. *Protein Secondary Structure Prediction Using Cascaded Convolutional and Recurrent Neural Networks*. 2016. URL: https://www.ijcai.org/Abstract/16/364.

[3] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[4] Jian Zhou and Olga Troyanskaya. "Deep Supervised and Convolutional Generative Stochastic Network for Protein Secondary Structure Prediction". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 745–753. URL: http://proceedings.mlr.press/v32/zhou14.html.