Ottimizzazione degli iperparametri per una rete neurale convoluzionale

Scaramuzziono Giovanna[1] *

[1]email: giovanna.scaramuzzino1@stud.unifi.it

L'obiettivo di tale elaborato è quello di ottimizzare gli iperparametri di una rete neurale convoluzionale. Per trovare tali parametri sono stati confrontati un algoritmo basato sulla ricerca randomica e due algoritmi basati sulla ricerca globale per funzioni costose.

Introduzione. Le reti neurali sono reti multistrato, ovvero costituite da un livello di input che riceve in ingresso i dati, uno o più livelli nascosti e un livello di output che genera il risultato finale. I nodi di uno strato si connettono solo ai nodi degli strati immediatamente precedenti e immediatamente seguenti secondo vari schemi di connessione. Ogni livello di nodi si allena su un insieme distinto di funzionalità basate sull'output del livello precedente. Avanzando nella rete perciò, le caratteristiche che i nodi riescono a riconoscere sono più complesse poiché aggregano e ricombinano le funzionalità del livello precedente a cui sono connessi. Tali reti possono essere usate ad esempio per la classificazione e la predizione. L'apprendimento della rete avviene tramite la regolarizzazione dei pesi, assegnati ad ogni connessione, andando a minimizzare una funzione di costo (loss), la quale rappresenta l'errore sui dati in uscita rispetto a quelli desiderati. Gli aggiornamenti di tali pesi vengono effettuati tramite la backpropagation e la discesa del gradiente. Affinchè l'algoritmo di discesa del gradiente funzioni al meglio, è necessario settare alcuni parametri manualmente (iperparametri), prima che inizi l'addestramento della rete. Alcuni degli iperparametri possono essere:

- **learning rate**: indica quanto velocemente permettiamo al parametro da aggiornare di andare in direzione opposta a quella del gradiente. Se viene impostato troppo piccolo, l'allenamento sarà affidabile, ma l'aggiornamento dei parametri sarà molto lento e ci vorrà molto tempo per ottenere una decrescita accettabile. Altrimenti, se viene impostato ad un valore troppo grande, potrebbe non convergere o addirittura divergere e le variazioni di peso potrebbero essere così grandi da superare il minimo della funzione.

- **decay**: parametro che permette di alleviare gli effetti dell'overfitting, andando a ridurre il learning rate all'aumentare delle epoche. Tramite tale aggiustamento del learning rate epoca per epoca è possibile ridurre la loss, aumentare l'accuracy ed in alcuni casi ridurre il tempo di addestramento della rete. Una delle possibile regole di aggiornamento del learning rate (LR) è la seguente:

$$LR = init\_LR * \frac{1.0}{1.0 + decay * iterazione} \quad (1)$$

Fig. 1. A sinistra cellula senza tumore, a destra cellula con tumore

Gli iperparametri hanno un impatto significativo sulla performance del modello addestrato. La scelta ottimale di essi è però costosa poichè richiede un addestramento della rete per ciascuno di essi. È possibile evitare di dover effettuare molte valutazioni costose della funzione tramite l'uso di vari algoritmi di ottimizzazione. In particolare sono stati usati l'algoritmo **Random Search** e gli algoritmi di ottimizzazione globale: **Radial Basis Function** (RBF) e **Bayesian optimizer**. Questi ultimi due, a differenza del primo, tengono conto delle precedenti valutazioni della funzione obiettivo e si basano sulla generazione di una sua approssimazione (detta *modello surrogato*), molto più facile da ottimizzare rispetto alla funzione reale. Tali algoritmi tentano perciò di trovare l'ottimo globale in un numero minimo di passaggi.
L'*ottimizzazione Bayesiana* come modello surrogato utilizza i Processi Gaussiani e, per determinare i nuovi punti da osservare, utilizza una *funzione di acquisizione* che genera tali punti in aree in cui si ha una maggiore probabilità di avere un miglioramento rispetto all'attuale migliore osservazione.
L'*RBF* invece, per la ricerca degli iperparametri ottimi, genera il modello surrogato tramite la combinazione di funzioni di base radiale. Per funzione di base radiale si intende una qualsiasi funzione $f(x)$ il cui valore dipende dalla distanza tra x ed un punto prefissato c del dominio. Tra le più comuni famiglie di funzioni a base radiale vi sono le gaussiane, le spline, le cubiche e le multiquadratiche, la cui scelta ha un grande impatto sulla qualità del modello. Tramite la minimizzazione di tale modello surrogato è possibile determinare un nuovo punto tramite il quale costruire il nuovo interpolante.
È stato inoltre usato il *random search*, come ulteriore algoritmo di confronto, il quale si basa sul determinare gli iperparametri da ottimizzare in modo randomico scengliendoli uniformemente su un dominio dato.

Esperimento. Per tale elaborato è stato usato il dataset Breast Cancer [2] contenente immagini del carcinoma duttale invasivo (IDC), ovvero il sottotipo più comune di tutti i tumori al seno. I dati origininali sono caratterizzati da 162

immagini scansionate a 40x, dalle quali sono state estratte 277,524 patch di dimensioni 50 x 50 (198.738 IDC negativi e 78.786 IDC positivi). Il dataset è diviso in cartelle contenenti i vari ID dei pazienti e, ciascuna di essa, è divisa a sua volta in due sottocartelle: 0 e 1. All'interno della cartella 0 vi sono tutte le immagini che non contengono tumore, viceversa nella cartella 1.
Per tale esperimento sono state utilizzate solo 60.000 immagini dell'intero dataset, di cui 48.000 per la fase di addestramento e 12.000 per il test, in modo tale da ridurre i tempi di allenamento della rete. È stato fatto ciò a discapito del raggiungimento dell'accuratezza ottimale, poiché non rientrava nell'obiettivo del progetto.
Le immagini di addestramento sono state inoltre divise in train (80%) e validation (20%).
Il nuovo dataset è stato creato in modo tale da avere lo stesso numero di immagini per entrambe le due classi: tumore, non tumore (vedi figura 1) poichè l'avere le classi bilanciate aiuta la rete a saperle riconoscere entrambe, potendo vedere lo stesso numero di esempi per ognuna di esse.
Per la classificazione di tali immagini è stata costruita una rete *from scratch* (figura 2), caratterizzata da una inizializzazione dei pesi casuale secondo la distribuzione di Glorot, utilizzando come libreria Keras. Per avere sempre la stessa inizializzazione dei pesi iniziali per ogni addestramento con i diversi iperparametri è stato settato un *seed* pari a 10.
Le immagini date in ingresso alla rete sono state decodificate, convertite in tensori floating point e ridimensionate ad una grandezza di 48x48. Sono stati inoltre ridimensionati i valori dei pixel nell'intervallo [0, 1], poiché le reti neurali preferiscono gestire piccoli valori in input.
Come funzione di attivazione è stata utilizzata la *RELU*, tranne nell'ultimo layer nel quale, essendo un problema di classificazione binaria, è stata impiegata la *sigmoid*.
È stata effettuata anche la tecnica del Dropout, la quale consiste nello spegnimento di una certa percentuale di collegamenti tra neuroni durante la fase di addestramento. Ciò permette di ridurre l'overfitting, ovvero l'adattamento ai dati di addestramento.

Per l'ottimizzazione degli iperparametri *learning rate* e *decay* è stata utilizzata, come funzione obiettivo, la loss sul validation, ed in particolare la *Binary cross entropy*. Per la discesa del gradiente, è stato scelto come metodo *Adam*, al quale vengono passati gli iperparametri.
Per l'algoritmo RBF è stata usata l'implementazione della libreria rbfopt [3] della quale sono state richiamate in particolare le classi: *RbfoptUserBlackBox*, alla quale è stata passata la funzione da ottimizzare e i valori minimi e massimi che è possibile assegnare ai parametri learning rate e decay; *RbfoptSettings* tramite la quale è stato possibile settare il massimo numero di iterazioni dell'algoritmo di ottimizzazione, il valore obiettivo che la funzione deve raggiungere ed il numero di punti iniziali che l'algoritmo deve individuare casualmente; infine è stata utilizzata *RbfoptAlgorithm* per eseguire l'algoritmo di ottimizzazione vero e proprio.
Per l'algortimo Bayesian optimizer, invece, è stata usata la libreria bayesian_optimization [1]: il costruttore è stato inizializzato passandogli la funzione obiettivo e i bounds degli iperparametri da individuare. Inoltre è stato utilizzato il metodo *maximize* per trovare il massimo della funzione obiettivo,per il quale sono stati settati: il numero di passi per l'esplorazione casuale e il massimo numero di iterazioni. Poiché tale algoritmo, massimizza la funzione obiettivo, è stato necessario passare all'ottimizzatore, la validation loss ottenuta in seguito all'addestramento, variata di segno.
Per entrambi gli algoritmi è stato settato a 3 il numero di passi per l'esplorazione casual, e a 25 il numero massimo di iterazioni.
Infine è stato implementato il metodo Random search, il quale si basa sul determinare randomicamente i due iperparametri, scegliendoli su una distribuzione uniforme su un intervallo dato. Ciò è stato ripetuto per 25 volte in modo da poter confrontare tale algoritmo con gli altri due.
Per tutti e tre gli algoritmi sono stati utilizzati come dominio per il learing rate e decay $(10^{-5}, 10^{-1})$. Ad ogni scelta di tale coppia di iperparametri è stato effettuato un addestramento della rete su 15 epoche.
Per ognuno dei tre algoritmi sono stati memorizzati, all'interno di un file csv, i risultati ottenuti ad ogni iterazione: la minor loss sul validation set, l'epoca alla quale è stata ottenuta e i corrispondenti valori dell'accuracy sul validation e

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 32)	896
activation_1 (Activation)	(None, 48, 48, 32)	0
batch_normalization_1 (Batch	(None, 48, 48, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
activation_2 (Activation)	(None, 16, 16, 64)	0
batch_normalization_2 (Batch	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
activation_3 (Activation)	(None, 16, 16, 64)	0
batch_normalization_3 (Batch	(None, 16, 16, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 64)	262208
activation_4 (Activation)	(None, 64)	0
batch_normalization_4 (Batch	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65
activation_5 (Activation)	(None, 1)	0

Total params: 319,489
Trainable params: 319,041
Non-trainable params: 448

Fig. 2. Architettura della rete

Optimizer	Learning Rate	Decay	Train Accuracy	Train Loss	Val Accuracy	Val Loss	Test Accuracy	Test Loss
RBF	0.09607	0.00171	0.9422	0.1581	0.8432	**0.36**	0.7714	0.5114
Bayesian	0.08186	0.06998	0.8819	0.2951	0.7883	0.4546	0.6011	0.7356
Random	0.09082	0.00073	0.9543	0.1262	0.8327	0.4124	0.7267	0.6739

Table 1. Learning rate e decay ottimi individuati distintamente dai tre algoritmi di ottimizzazione, ottimizzando sul validation loss. Per ciascuna coppia di iperparametri ottimi individuati da ogni ottimizzatore sono riportati: la minima loss ottenuta sul validation set in una delle 25 iterazioni ed in una delle 15 epoche (val loss), l'accuratezza sul validation e l'accuracy e loss per train e test set ottenute in tale epoca.

la loss e l'accuracy sul train set.
Sugli iperparametri migliori, ottenuti da ciascun algoritmo, sono stati infine calcolati la loss e l'accuracy sul test set, in modo tale da poter valutare il modello ottenuto su dati mai visti dalla rete.

Risultati. Effettuando 25 iterazioni di addestramento, ciascuna su 15 epoche, per tutti e tre gli algoritmi di ottimizzazione, il *Radial basis function* è l'ottimizzatore che riesce ad ottenere il minimo valore sul validation loss. Tale risultato è possibile osservarlo in tabella 1 all'interno della quale sono riportati i risultati migliori ottenuti sul validation loss, per i tre algoritmi (in Appendice è possibile trovare tre tabelle nelle quali sono mostrati i risultati ottenuti ad ogni iterazione per ciascun algoritmo). Inoltre, su tali risultati migliori sono stati valuti l'accuracy e la loss sul test set, risultando essere migliori, come previsto dal validation set, per RBF. Tale algoritmo permette di raggiungere un'accuratezza sul test set del 77%, valore superiore rispetto agli altri due algoritmi, anche se non troppo alto, a causa dell'aver addestrato la rete su poche immagini. Inoltre, per ridurre l'overfitting in modo tale non vi sia adattamento ai dati di addestramento e migliorare perciò le capacità di generalizzazione della rete sul test set, è possibile applicare la tecnica dell'early stopping e del cross validation. In tal caso, tali tecniche non sono state applicate per ridurre i tempi di esecuzione. È stata invece memorizzata la loss minima ottenuta, addestrando per 15 epoche sul validation set.
Il Bayesian optimizer è comunque un buon algoritmo, ma necessita di un numero maggiore di iterazioni iniziali, potendo così esplorare più zone in modo casuale, per ottenere risultati migliori
Il Random Search, invece, a differenza dei due algoritmi precedenti, esegue una scelta degli iperparametri in modo casuale senza tener conto dei risultati ottenuti alle iterazioni precedenti. Per tale motivo con RBF e Bayesian optimizer, si ha maggiori possibilità di raggiungere il risultato ottimo in un minor tempo, e perciò di effettuare minori valutazioni della funzione obiettivo.

Conclusioni. L'ottimizzazione degli iperparametri analizzati in tale elaborato, nel caso di funzioni costose, da quanto si evince dai risultati ottenuti, è preferibile effettuarla tramite l'algoritmo RBF. Tale algoritmo, a differenza del Bayesian optimizer ha bisogno di meno iterazioni iniziali per fornire dei buoni risultati, ma risultano essere entrambi migliori rispetto al Random Search. Ciò poichè i primi due scelgono i valori da valutare tenendo conto dei risultati ottenuti alle iterazioni precedenti, aumentando anche la possibilità di avere migliori prestazioni sul test set. I vantaggi rispetto al random search sono ancor più evidenti per set di dati grandi o uso di molti parametri, anche se la ricerca casuale può restituire iperparamentri migliori nelle prime iterazioni solo per pura fortuna. Per tale motivo il random search è meglio evitarlo nel caso di funzioni costose.
I risultati ottenuti però dipendono molto dal dominio di ricerca scelto e dal set di dati utilizzati. Pertanto, in diverse situazioni, diverse tecniche di ottimizzazione potrebbero avere prestazioni migliori di altre.

Riferimenti

[1] *Bayesian optimization.* URL: https://pypi.org/project/bayesian-optimization/.
[2] *Breast histopatology (dataset).* URL: https://www.kaggle.com/paultimothymooney/breast-histopathology-images.
[3] *Radial Basis Function.* URL: https://pypi.org/project/rbfopt/.

Appendice:

Tabelle contenenti i risultati

Radial basis function					
Learning Rate	Decay	Train Accuracy	Train Loss	Val Accuracy	Val Loss
0.08933	0.09928	0.9234	0.2035	0.7164	0.5722
0.02046	0.00144	0.9028	0.2422	0.7752	0.5089
0.05	0.05	0.9488	0.1434	0.7844	0.4973
0.00043	0.09988	0.757	0.5446	0.7293	0.5628
0.09981	0.00029	0.9373	0.1751	0.8459	0.3861
0.09982	1e-05	0.9389	0.1801	0.8032	0.4488
0.09994	0.01677	0.9491	0.1429	0.7494	0.5397
0.05993	0.01077	0.9512	0.1351	0.7367	0.6385
0.00043	0.04657	0.7204	0.6179	0.8097	0.4249
0.09932	0.00504	0.9419	0.159	0.8016	0.479
0.09996	0.00158	0.885	0.2937	0.8268	0.3983
0.08085	0.00481	0.9406	0.16	0.8221	0.4308
0.08181	0.04347	0.9297	0.1894	0.7471	0.5195
0.0996	0.00075	0.9499	0.1384	0.8075	0.4644
0.1	0.00045	0.937	0.171	0.8164	0.4565
0.09607	0.00171	0.9422	0.1581	0.8432	**0.36**
0.1	0.00083	0.9403	0.1628	0.8621	0.3856
0.09891	0.00073	0.9573	0.1188	0.8421	0.4634
0.0994	0.06861	0.9327	0.1779	0.7534	0.508
0.08742	2e-05	0.9345	0.1852	0.8475	0.3713
0.09237	0.00213	0.9643	0.0986	0.7859	0.4993
0.08394	2e-05	0.941	0.1665	0.7986	0.7427
0.00045	0.01931	0.8855	0.3033	0.78	0.5091
0.02596	0.02745	0.9243	0.2019	0.7336	0.5536
0.08087	0.01326	0.9592	0.1166	0.7975	0.4692

Table 2. Risultati ottenuti con l'algoritmo Radial Basis Function. In ogni riga è possibile visualizzare al variare del Learning rate e del Decay (si ricorda che ogni coppia di iperparametri individuata corrisponde ad un'iterazione dell'algoritmo): Accuracy e loss sul train e validation set. In grassetto ed evidenziato di arancione è possibile osservare il valore ottimo trovato da tale algoritmo, ottimizzando sul validation loss. Evidenziati di giallo vi sono i corrispondenti Learning rate e decay ottimi.

Bayesian					
Learning Rate	Decay	Train Accuracy	Train Loss	Val Accuracy	Val Loss
0.01443	0.08489	0.916	0.2205	0.7799	0.5414
0.0208	0.06667	0.9144	0.2276	0.7814	0.5022
0.00378	0.05223	0.9041	0.2509	0.7748	0.511
0.1	1e-05	0.9333	0.1956	0.6759	0.7884
0.1	0.1	0.9288	0.1881	0.7615	0.4795
1e-05	1e-05	0.849	0.3708	0.7333	0.6186
0.08186	0.06998	0.8819	0.2951	0.7883	**0.4546**
0.06945	0.09586	0.9266	0.1996	0.7494	0.5061
0.1	0.07211	0.933	0.1772	0.7309	0.5724
0.05037	0.02025	0.9357	0.1738	0.7652	0.5361
0.05893	0.0531	0.9339	0.1758	0.7772	0.4884
0.04086	1e-05	0.9505	0.1373	0.7675	0.5051
0.06509	0.07465	0.8859	0.2808	0.741	0.5508
1e-05	0.1	0.5217	1.039	0.5036	0.7882
0.08431	0.03867	0.9268	0.1987	0.7258	0.6076
0.0207	0.02596	0.9425	0.1556	0.7434	0.5235
0.03612	0.1	0.9115	0.2299	0.7058	0.5774
0.06613	1e-05	0.9382	0.1762	0.7716	0.6068
0.086	0.1	0.9241	0.2069	0.7998	0.4566
0.0871	0.08637	0.883	0.2907	0.6855	0.6028
1e-05	0.02745	0.5376	1.0009	0.6035	0.6828
1e-05	0.07079	0.5474	0.9867	0.6032	0.6932
0.03123	0.04425	0.9251	0.1967	0.7854	0.4871
0.07858	0.05694	0.9236	0.199	0.7391	0.6536
0.1	0.02886	0.936	0.1764	0.7241	0.6311

Table 3. Risultati ottenuti con l'algoritmo Bayesian optimizer. In ogni riga è possibile visualizzare al variare del Learning rate e del Decay (si ricorda che ogni coppia di iperparametri individuata corrisponde ad un'iterazione dell'algoritmo): Accuracy e loss sul train e validation set. In grassetto ed evidenziato di arancione è possibile osservare il valore ottimo trovato da tale algoritmo, ottimizzando sul validation loss. Evidenziati di giallo vi sono i corrispondenti Learning rate e decay ottimi.

Random					
Learning Rate	Decay	Train Accuracy	Train Loss	Val Accuracy	Val Loss
0.07259	0.09506	0.8827	0.2918	0.6864	0.6979
0.04273	0.08839	0.916	0.2139	0.764	0.485
0.0238	0.08405	0.9323	0.1787	0.7019	0.7598
0.09082	0.00073	0.9543	0.1262	0.8327	**0.4124**
0.09106	0.0472	0.8948	0.2678	0.8039	0.4388
0.00415	0.03628	0.8978	0.2664	0.779	0.4632
0.05095	0.08623	0.9082	0.2389	0.7542	0.5119
0.08204	0.05019	0.8916	0.2779	0.7475	0.5127
0.04023	0.0709	0.9246	0.2003	0.7131	0.6287
0.02366	0.07617	0.898	0.2616	0.7061	0.5987
0.00747	0.09363	0.9068	0.2522	0.7293	0.5801
0.09857	0.05262	0.9311	0.1903	0.761	0.5366
0.0777	0.00444	0.9542	0.1252	0.7621	0.5298
0.06797	0.0718	0.9352	0.1759	0.7666	0.4804
0.07138	0.09699	0.9364	0.1769	0.7466	0.5414
0.0948	0.00335	0.9524	0.1337	0.7595	0.5508
0.07648	0.08996	0.9322	0.1833	0.7055	0.5944
0.06236	0.00363	0.9378	0.1647	0.7342	0.6105
0.08248	0.04496	0.9217	0.2086	0.7464	0.5854
0.07046	0.07845	0.9244	0.206	0.7518	0.5011
0.01516	0.09095	0.9083	0.2424	0.7194	0.5361
0.00267	0.06878	0.8459	0.3636	0.6355	1.5399
0.05	0.05202	0.892	0.2703	0.6949	0.6064
0.08037	0.0835	0.9107	0.2353	0.697	0.6096
0.00372	0.0532	0.8651	0.3338	0.7519	0.5191

Table 4. Risultati ottenuti con l'algoritmo Random search. In ogni riga è possibile visualizzare al variare del Learning rate e del Decay (si ricorda che ogni coppia di iperparametri individuata corrisponde ad un'iterazione dell'algoritmo): Accuracy e loss sul train e validation set. In grassetto ed evidenziato di arancione è possibile osservare il valore ottimo trovato da tale algoritmo, ottimizzando sul validation loss. Evidenziati di giallo vi sono i corrispondenti Learning rate e decay ottimi.