Ottimizzazione di iperparametri in una rete neurale

Pietro Zarri 7042627

9 aprile 2020

1 Introduzione

L'obiettivo di questo elaborato è mostrare come gli algoritmi di ottimizzazione per funzioni costose possano essere utilizzati per scegliere gli iperparametri all'interno dell'addestramento di una rete neurale. In questo caso particolare è stato sfruttato un modello surrogato che si basa su l'interpolazione di punti tramite Radial Base Function (RBF).

2 Reti neurali e iperparametri

Le reti neurali sono una delle tecniche di intelligenza artificiale che sta guadagnando più popolarità negli ultimi anni e sono particolarmente efficaci per risolvere problemi di natura predittiva o di classificazione.
L'addestramento di una rete neurale ha come scopo la calibrazione della stessa al fine di migliorare i risultati su un insieme di test. Per migliorare tali prestazioni è possibile regolare un certo numero di parametri e iperarametri.
I parametri (o pesi) della rete neurale sono determinati, fissato il numero di livelli e neuroni, durante l'addestramento della stessa tramite varie tecniche al fine di ottimizzare una funzione obiettivo detta *loss*, che misura quanto le predizioni della rete si discostano dagli output desiderati.
Le reti neurali riescono però a sfruttare al massimo le loro potenzialità con la configurazione dei loro iperparametri. Le tecniche di addestramento per i pesi però non riescono ad attribuire un appropriato valore agli iperparametri, che devono essere assegnati a priori.

3 Definizione del problema

Poiché l'addestramento di una rete neurale può comportare un importante utilizzo di tempo e di risorse possiamo ricondurre la scelta degli iperparamentri ad un problema di ottimizzazione in cui il calcolo della funzione obiettivo risulta molto dispendioso. Esistono molte tecniche che possono essere utilizzate a questo scopo, come ad esempio *Random-search* o *Bayesian*. In questo progetto utilizziamo una strategia differente per esplorare lo spazio degli ipeparamentri ovvero le *Radial Base Function*.

3.1 Radial Base Function e Bumpiness

Dati un insieme di punti già testati $h_1, i = 1 \dots k$, è definita una funziona ausiliaria s nel seguente modo:

$$s(h) = \sum_{j=1}^{k} \lambda_j \phi(\|h - h_j\|) + p(x)$$

Ci sono alcune possibili scelte per la funzione ϕ come *cubic, thin splate spline* o *gaussian*.
Una volta costruita s, dato l'elevato costo di chiamata della funzione obiettivo, è necessario stabilire un criterio che, basandosi proprio su s, possa suggerire un nuovo punto potenzialmente migliore di quelli già esplorati.
Il metodo basato sulle RBF elaborato da Gutmann prevede l'introduzione del concetto di *Bumpiness* di una funzione, che può essere associato con l'irregolarità dell'andamento della stessa. Assumiamo che la funzione obiettivo sconosciuta non oscilli troppo e quindi che l'approssimante interpoli i dati minimizzando la *bumpiness*.

Più formalmente andiamo a scegliere come nuovo punto $h_{k+1} = \hat{h}$ su cui fare una valutazione quello che minimizza la funzione di bumpiness:

$$\lambda^T \Phi \lambda$$

dove Φ e λ sono relativi al modello surrogato costruito sul precedente insieme di punti, più $(\hat{h}, \hat{f})$.

4 Implementazione

Per un'analisi pratica sono sono state testate due diverse reti neurali.
La prima si basa sul dataset Fashion MNIST, che contiene 60000 immagini 28 x 28 pixel di un catalogo indumenti, classificate in base all'oggetto rappresentato.

Figura 1: Esempi di elementi all'interno del MNIST dataset.

La rete si compone tre livelli:

1. Livello di tipo Flatten che prende in ingresso ogni punto della foto.
2. Livello di tipo Dense a 128 nodi.
3. Livello di output che classifica l'immagine con una delle 10 possibilità.

Il secondo dataset è il Pima Indians diabetes database dove ogni elemento presenta otto diverse rilevazioni mediche su un paziente e la positività o meno dello stesso al diabete. In totale sono presenti 768 osservazioni. Questo è quindi un problema di classificazione binaria.

	Number of times pregnant	Plasma glucose	Diastolic blood pressure	Triceps skinfold thickness	serum insulin	Body mass index	Diabetes pedigree function	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0

Figura 2: Esempi di elementi all'interno del dataset Pima.

Anche questa rete si compone tre livelli:

1. Livello di input che prende in ingresso gli otto parametri con 4 nodi.
2. Livello di tipo Dense a 4 nodi.

3. Livello di output con un solo nodo.

Gli iperparamentri individuati da ottimizzare sono *Learning rate* e *Decay* per entrambi i dataset.
L'addestramento avviene per il primo su 5 epoche per il secondo su 100 ed è stato stato fissato a 10 il numero massimo di valutazioni sulle funzioni stesse.
La reti sono state create sfruttando la libreria open source Keras. Essa fa da interfaccia alla libreria Tensorflow. Essendo una libreria di alto livello permette in modo intuitivo e rapido la creazione di una semplice rete neurale come quelle di questo progetto. La parte cruciale però di questo elaborato, ovvero la costruzione del modello surrogato per la scelta di nuovi punti da testare, come descritto in sezione 3.1, è eseguito dalla libreria RBFopt.

4.1 RBFopt

RBFopt è una libreria per l'ottimizzazione di funzioni black-box, scritta in Python.
Per questo semplice progetto sono state utilizzate le tre classi fondamentali di questa libreria:

- RbfoptUserBlackBox: La classe che implementa i metodi necessari per descrivere una funzione black-box che deve essere minimizzata, dove l'utente inserisce tutti i dati richiesti.
- RbfoptSettings: Classe che contiene tutte le impostazioni per l'esecuzione dell'algoritmo.
- RbfoptAlgorithm: Implementa l'algoritmo principale di ottimizzazione e richiede un oggetto che deriva dalla classe astratta RbfoptBlackBox che descrive il problema che deve essere ottimizzato.

In particolare la classe RbfoptBlackBox richiede: la dimensione del problema, un limite inferiore e superiore per ogni variabile e ovviamente la funzione che deve essere ottimizzata.

```
rbfopt.RbfoptUserBlackBox(2,np.array(lower_bounds),np.array(upper_bounds),
    np.array(['R', 'R']), evaluate_hyp)
```

In questo caso passiamo in un caso funzione *evaluate_ hyp* e nell'altro la funzione *evaluate_ hyp2*, che ricevono in ingresso i due valori per *learning rate* e *decay*, eseguono un addestramento della rete neurale sul dataset e restituiscono il valore del punteggio ottenuto sul test set.
Per inizializzare la funzione surrogata sono eseguiti automaticamente tre iterazioni preliminari della funzione con valori scelti casualmente per i due iperparamentri.
Il progetto è strutturato su tre file:

- **text.py**: Contiene la funzione *evaluate_ hyp* per la valutazione degli iperparamentri ed istanza tutte le componenti per l'ottimizzazione tramite RBFopt.
- **model.py**: Contiene il modello della rete neurale utilizzata per il dataset fashion MINIST.
- **model2.py**: Contiene il modello della rete neurale utilizzata per il dataset Pima Indians diabetes.

5 Risultati

Come detto è stato fissato a 10 il numero di valutazioni sulla funzione principale, più le tre iterazioni iniziali, ogni volta vengono salvati i valori di *learning rate*, *decay* e i punteggi di *loss* e *accuracy* ottenuti sull'insieme di test per entrambe le reti.
Di seguito sono riportate le due tabelle che riassumono questi valori per un esecuzione di prova del progetto:

Learning Rate	Decay	Test loss	Test accuracy
0.08934253253606392	0.09928417280036561	0.7146126368999481	0.7619
0.020531632600504484	0.0014256297623185645	0.502266062116623	0.826
0.050050000000000004	0.05	0.7217527620315551	0.7642
0.09989063449450059	0.00039028207839080236	0.3639147037029266	0.8717
0.06044466722827648	0.0004114827631144902	0.3885692417860031	0.8626
0.08078253569895169	0.0006662893435766537	0.38856506924629214	0.8631
0.0999297273911008	5.002104818567329e-05	0.35901788012981417	0.8707
0.09997469376707234	2.5647200355379978e-06	0.3635076806306839	0.8693
0.09437261873965078	1.596191589813495e-12	0.36503599228858946	0.8682
0.00027590476890442466	0.09990054286891763	2.410877621459961	0.0861
0.05465297819017532	0.08698808100571286	0.7923473996162415	0.7372
0.041105277570245735	5.963071084446048e-05	0.3900707960605621	0.8605
0.09747423620691924	6.241749693719862e-06	0.36422256443500517	0.8706

Tabella 1: Tabella esecuzione su dataset fashion MNIST.

Learning Rate	Decay	Test loss	Test accuracy
0.08934253253606392	0.09928417280036561	0.6691106307558167	0.6103895902633667
0.020531632600504484	0.0014256297623185645	0.6373398291084157	0.6666666865348816
0.050050000000000004	0.05	0.6541292561597122	0.6406926512718201
0.0995765466325091	4.5847130038878527e-05	0.5200358461507987	0.7489177584648132
0.0998794514509853	0.05063274740938013	0.6529991139065136	0.6406926512718201
0.09254715138048217	0.000163720672107881	0.48076684366572986	0.761904776096344
0.07981841791594298	2.740108706037858e-05	0.49936046912556603	0.761904776096344
0.08991631891753259	1.4878383158641802e-05	0.5951658745348711	0.7229437232017517
0.09508492664179147	0.0009888093377641096	0.45049614268979987	0.7835497856140137
0.052495752758248734	3.977705222810713e-05	0.47504916309794304	0.761904776096344
0.07463639699288176	0.07071293032386686	0.6404567489892373	0.6709956526756287
0.06923707473632093	0.00010321186326669452	0.5655806978027542	0.7445887327194214
0.0787257737781176	1.9280477953687385e-05	0.4773204294376043	0.7402597665786743

Tabella 2: Tabella esecuzione su dataset Pima.

Comparando i dati anche su un grafico cartesiano possiamo vedere come entrambi abbiamo ottenuto un buon miglioramento grazie a questo metodologia, rispetto ai casi in cui l'addestramento è stato eseguito senza ottimizzare gli iperparamentri, ad esempio le tre iterazioni preliminari necessarie per creare la funzione surrogata. In particolare per il primo dataset riesce ad arrivare ad un valore di loss pari a 0.35 oltre l'87% di accuracy, mentre il secondo arriva a una loss nel caso migliore pari a 0.45 e al 78% di accuracy.

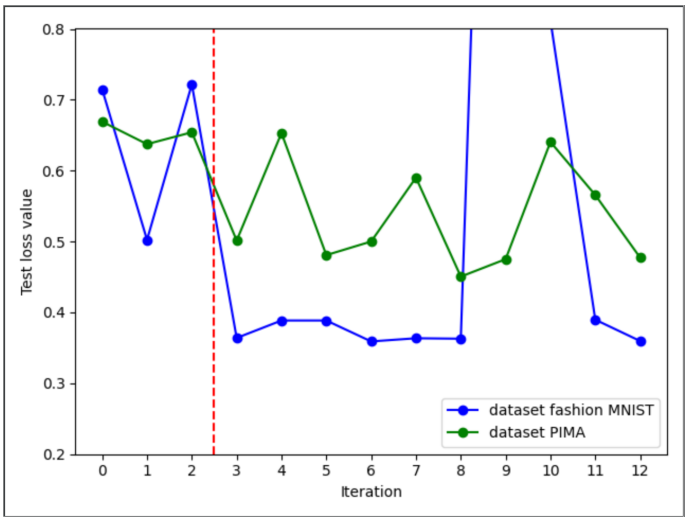

Figura 3: Grafico riassuntivo di un esecuzione dell'algoritmo su entrambi i dataset.

6 Conclusione

Ottimizzare gli iperparamentri durante l'addestramento di una rete neurale sfruttando le funzioni RBF come modello surrogato e la bumpiness come funzione di merito per la scelta di nuovi punti si è rivelato un buon metodo. Infatti già dopo poche valutazioni è stato possibile migliorare i valori di loss e accuracy durante il test di entrambi i dataset presi in considerazione.