DataScienceSeed

Performance di un approccio neurale al problema del caffè di FlairBit

Andrea Boero (andy.boero@gmail.com)

Precondizioni

Lo scopo del presente lavoro è quello di analizzare come variano le performance di una rete neurale *fully-connected* al variare di alcuni tra i principali *hyperparameters* che caratterizzano l'architettura e l'operatività di una rete neurale.

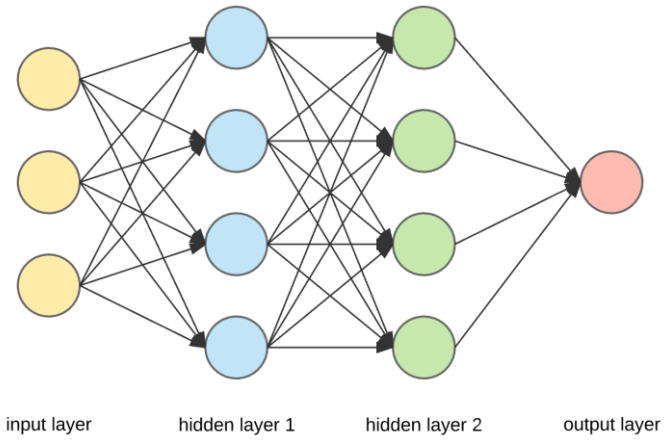

Fig.1: generica rete neurale fully-connected

lambda (fattore di regolarizzazione)

più è alto, più i pesi sinaptici vengono frenati nel loro valore assoluto causando uno smussamento della funzione di trasferimento della rete neurale, l'eliminazione dei picchi e in definiva il conferimento di un comportamento più deterministico ed equilibrato. Questo serve a contrastare il fenomeno dell'*overfitting*, ossia la tendenza naturale di una rete neurale a fornire ottime performance sul training set e scarse performance su validation set e test set e quindi a mostrare ottime capacità di memorizzazione e scarse capacità di generalizzazione, che invece rappresentano l'obiettivo di chi utilizza una rete neurale. L'overfitting può essere dovuto anche ad un'architettura esageratamente complessa rispetto al problema.

Nelle figure seguenti vengono illustrati i tre casi tipici in cui può incorrere il training di una rete neurale, rispettivamente in un problema di regressione ed in uno di classificazione: *underfitting* (scarse performance di apprendimento dovute a funzione di trasferimento troppo "semplice"), *overfitting* (funzione di trasferimento troppo complessa e probabili scarse performance su validation e test set e a maggior ragione sui "live data") e *rightfitting* (massima capacità di generalizzazione, ossia massime performance su validation e test set e sui "live data")

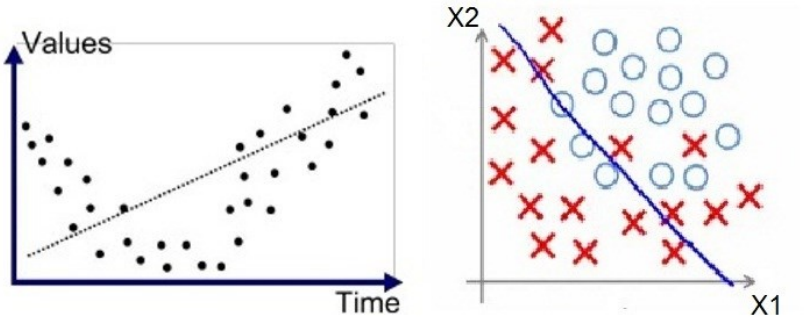

Fig.1: underfitting

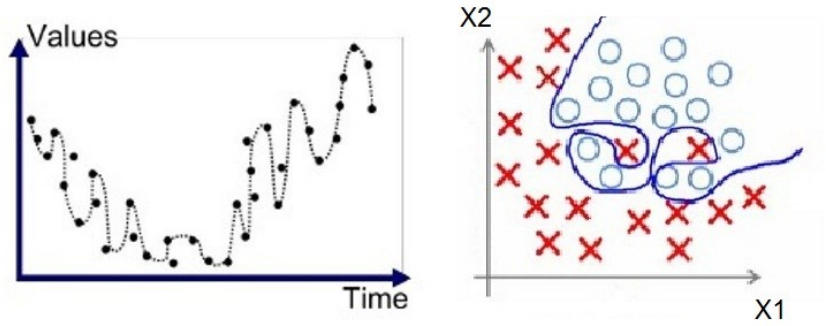

Fig.2: Overfitting

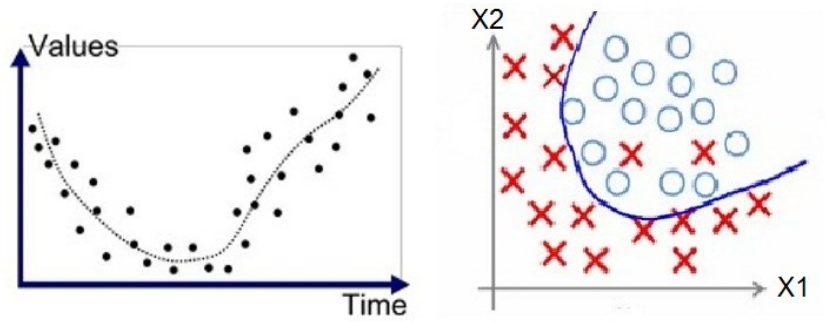

Fig.3: Rightfitting

numero di layer e numero di neuroni per layer

Uno dei principali dilemmi nella progettazione di una rete neurale consiste nella scelta dell'architettura. Pochi neuroni inducono una funzione di trasferimento con variazioni graduali, al contrario con molti neuroni (e molti layer) la funzione di trasferimento assume una dinamicità molto maggiore e talvolta inaccettabile (overfitting). In ogni caso si preferisce regolarizzare (ossia controbattere l'overfitting) con un lambda alto, piuttosto che con un numero di neuroni basso, tagliando le alte frequenze della funzione di trasferimento senza pregiudicarne la flessibilità

numero di epochs

una *epoch* è il ciclo all'interno del quale tutti i campioni del training set vengono sottoposti una volta all'algoritmo di discesa del gradiente (*backpropagation*). Tipicamente si esegue un numero di epoch dell'ordine delle migliaia.

Un'altra tecnica di regolarizzazione, meno popolare, è quella di mantenere sufficientemente basso il numero di epochs per non raggiungere il minimo (locale o globale) della funzione di costo evitando un "appiattimento" del comportamento della rete sui dati del training set a scapito della generalizzazione. Qui, comunque, si vuole solo analizzare come variano le performance all'aumentare del numero di epochs, dal momento che ogni epoch comporta un certo costo computazionale e potrebbe non valere la pena andare oltre una certa soglia.

batch size

nell'ambito di ogni epoch i campioni del training set possono essere sottoposti alla rete neurale singolarmente o in batch, nel qual caso il gradiente della funzione di costo viene calcolato sulla base dell'insieme dei campioni componenti il batch. Si vuole verificare se variazioni di questo parametro influenzano le performance.

activation

l'output di ogni neurone non arriva direttamente ai neuroni del layer successivo ma viene sottoposto alla cosiddetta "funzione di attivazione" che mappa tale valore su un altro dominio, tipicamente schiacciandolo nel range (0,1) (*sigmoid activation*) o (-1,1) (*hyperbolic tangent activation*). Molto utilizzata, soprattutto in ambito deep learning, è la "*rectified linear unit*" (ReLU). La sigmoid activation viene generalmente utilizzata come output dell'ultimo layer nei casi di classificazione binaria in quanto il dominio (0,1) ben si adatta a rappresentare la probabilità di appartenenza di un campione ad una classe o all'altra. In caso di classificazione multipla si usa una variante denominata "*softmax*".

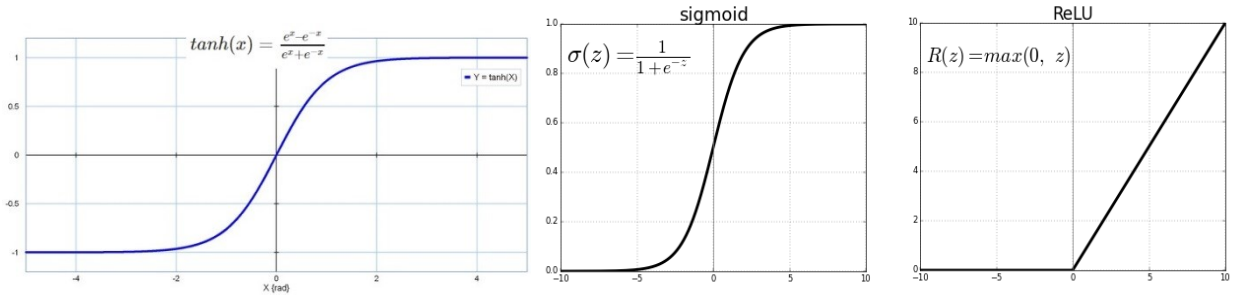

Fig.4: Activation functions: tanh, sigmoid, ReLU

Il learning rate non è stato oggetto di analisi e mantenuto costante a 0.01

Al problema di determinare l'architettura migliore dato un problema non esiste una risposta univoca: il suggerimento di Andrew Ng, fondatore di Coursera e titolare del corso online "Machine Learning" e della "Deep Learning Specialization", è quello di eseguire il training con tante combinazioni, verificare qual è quella con la maggiore accuratezza sul validation set ed eleggerla come architettura di riferimento sulla quale operare il training definitivo di cui valutare l'accuratezza su un test set separato.

Le suddette analisi vengono ripetute su tre varianti del dataset di partenza, FlairBit_5d, che mappa le informazioni sul modello della macchina del caffè e l'andamento di ogni contatore nei 15 giorni precedenti sull'occorrenza di almeno un fault nei 5 giorni successivi:

1) FlairBit_5d_onehot_scaled: in questo dataset (shape 62299x504) l'informazione sul modello viene convertita in un vettore *one-hot* (la singola feature di tipo intero riportante il numero di modello viene sostituita da un insieme di feature booleane, una per modello; in tal modo per ogni campione si avrà una sola feature settata a 1, quella corrispondente al modello), inoltre si applica un'operazione di *scaling* a tutti i valori di ogni feature per cui questi vengono compressi nel range [0,1]. Ciò rende "dolci" le concavità della funzione di costo; al contrario, differenze di ordini di grandezza tra features genera "gole" strette all'interno delle quali l'algoritmo di discesa del gradiente potrebbe non entrare o dalle quali potrebbe uscire.

2) FlairBit_5d_onehot_rm15_scaled: questo dataset (shape 62299x54), realizzato a partire dal precedente, è il risultato dell'accorpamento, per ogni contatore, delle 15 colonne relative al suo valore dal giorno -1 al -15 in un'unica colonna col valore medio

3) FlairBit_5d_nomodel_scaled: questo dataset (shape 62299x481) è analogo al primo ma senza l'informazione sul modello.

Risultati

Partiamo subito con l'analisi di **batch size** ed **activation** in quanto differenti scelte di tali parametri non hanno comportato variazioni di performance.

Per quanto riguarda **lambda** è emerso che, una volta impostato a zero, la differenza di accuratezza tra training set e validation set si è mantenuta in tutti i casi bassissima, il che indica che i modelli costruiti a partire dai dataset in nostro possesso non soffrono di overfitting, probabilmente per via del numero elevato di campioni (oltre 60000). Di conseguenza è stato mantenuto a 0 per tutto il resto dell'analisi.

Gli hyperparameters più influenti sull'accuratezza sono risultati essere invece il numero di epochs (sono stati provati i valori 0, 1, 10, 100), il numero di layer (2, 3, 5, 10) ed il numero di neuroni per layer (8, 16, 32, 64, 128, 256).

Segue una descrizione testuale e grafica dei risultati. Le performance sui tre dataset sono rappresentate nei grafici rispettivamente col colore verde, rosso e blu.

La variazione del numero di **epochs** da 0 (nessun training) ad 1 (campioni sottoposti una volta all'algoritmo di ottimizzazione) ha visto le performance passare mediamente dal 50% all'80%. Con 10 epochs si è passati all'85%, con 100 intorno all'87%. Le performance sul dataset 3 sono sempre risultate mediamente inferiori di un punto percentuale.

```
#varying n_of_epochs
lambda_array = [0]
neurons_per_layer_array = [16]
activation_array = ["tanh"]
epoch_array = [0,1,10,100]
n_of_layers_array = [3]
batch_size_array = [100]
training_set_percentage_array = [0.8]
```

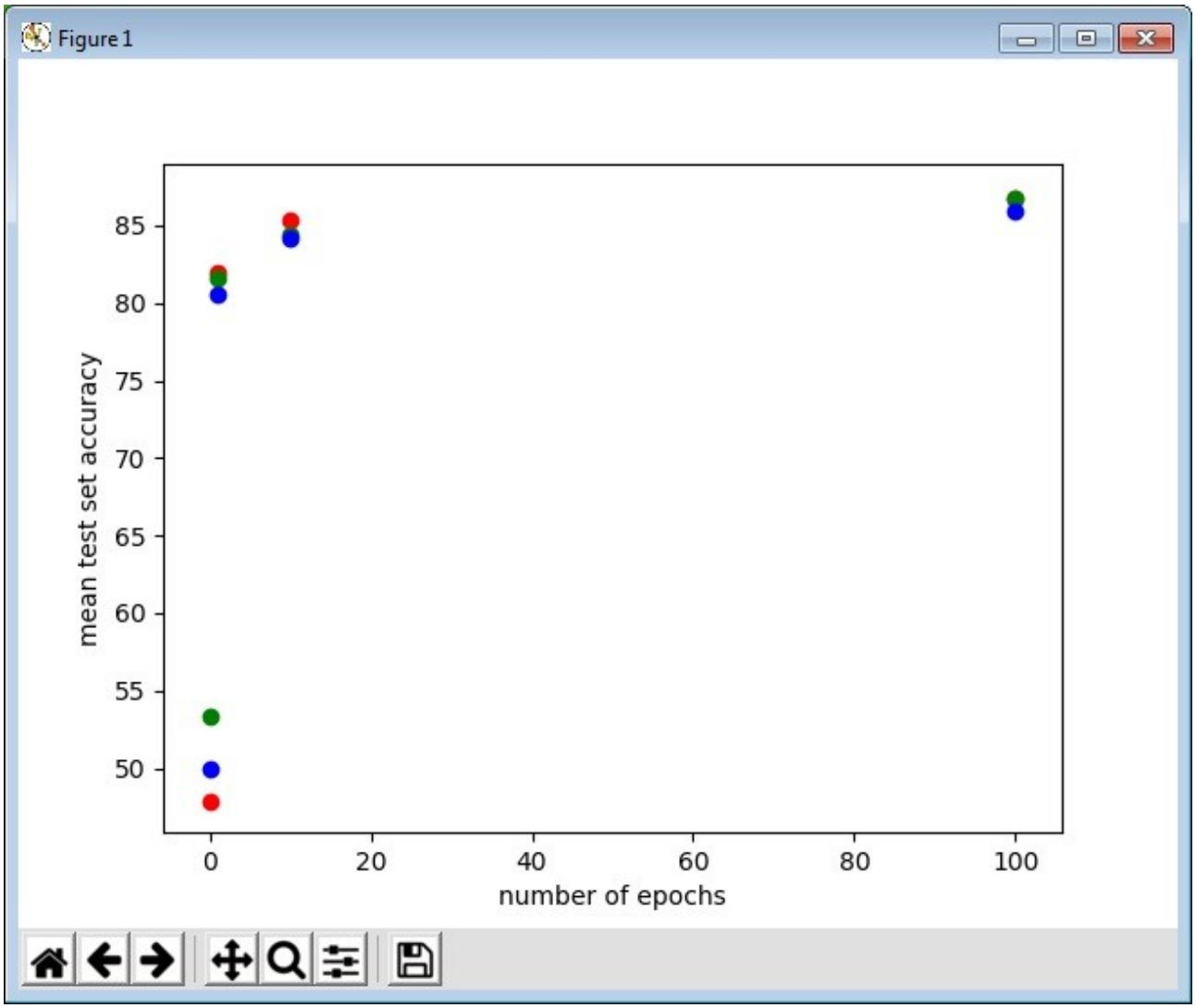

Fig.5: Accuracy media a fronte di 0, 1, 10 e 100 epoch

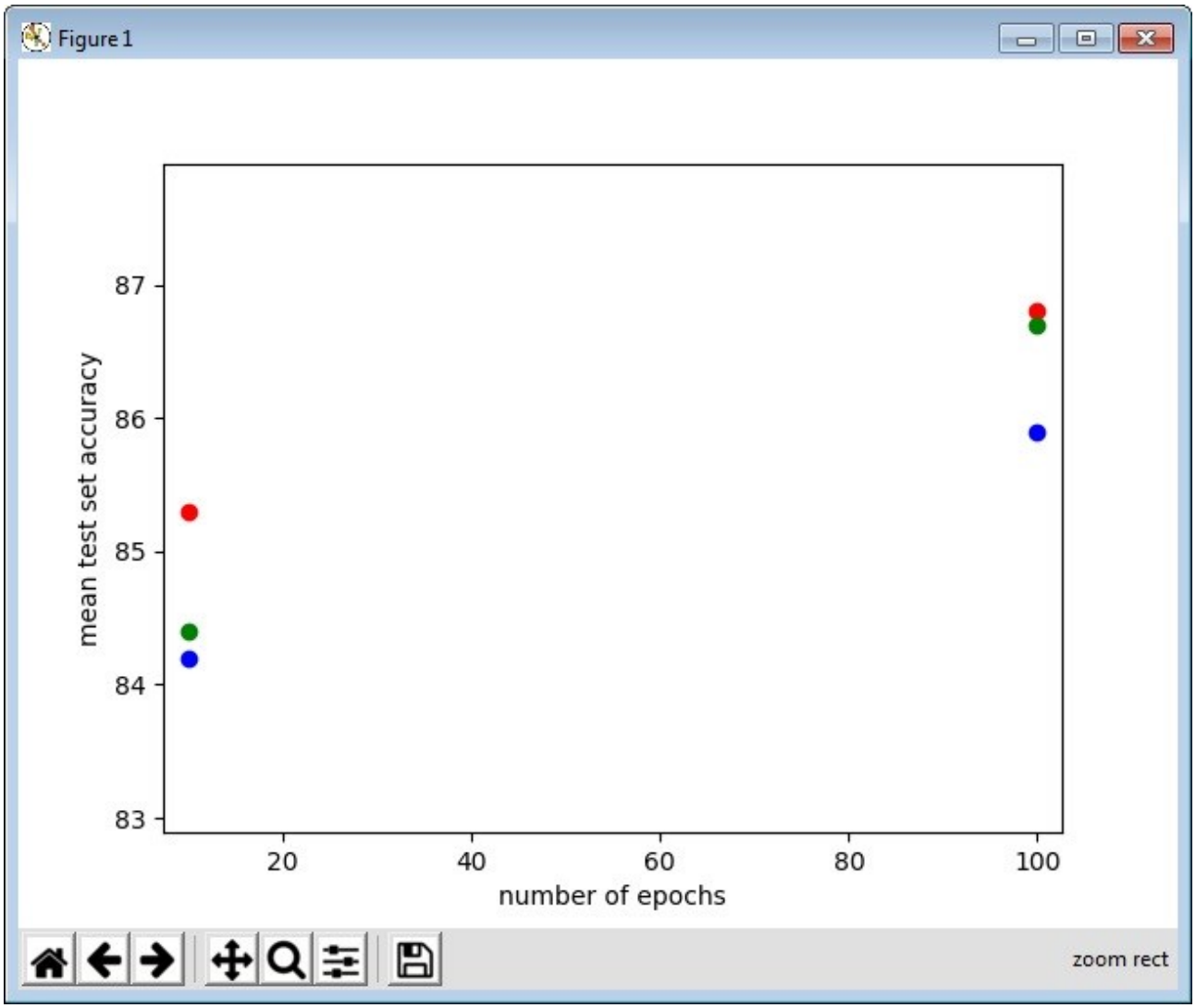

Fig.6: Accuracy media (zoom su 10 e 100 epoch)

In un caso, col dataset 2, ci si è spinti fino a 10000 epochs su un'architettura con 128 neuroni al primo layer, più altri 3 layer da 64, raggiungendo un'accuratezza del 90% sul validation set (l'accuratezza ottenuta andrebbe poi confermata su un test set dopo una nuova fase di training).

Per quanto riguarda il numero di **layer** (mantenendo fisso il numero di neuroni a 16 ed il numero di epoch a 100, come da code snippet), le performance con 2, 3 e 5 sono risultate essere molto simili tra loro (intorno all'86.5% sul validation set) sui singoli dataset ma anche in questo caso sul dataset 3 le performance sono state inferiori (in questo caso di circa 2 punti percentuali). Con 10 layers si è verificato un calo generale delle prestazioni, forse legato al fenomeno del vanishing\exploding gradient.

```
#varying n_of_layers
lambda_array = [0]
neurons_per_layer_array = [16]
activation_array = ["tanh"]
epoch_array = [100]
n_of_layers_array = [2,3,5,10]
```

```
batch_size_array = [100]
training_set_percentage_array = [0.8]
```

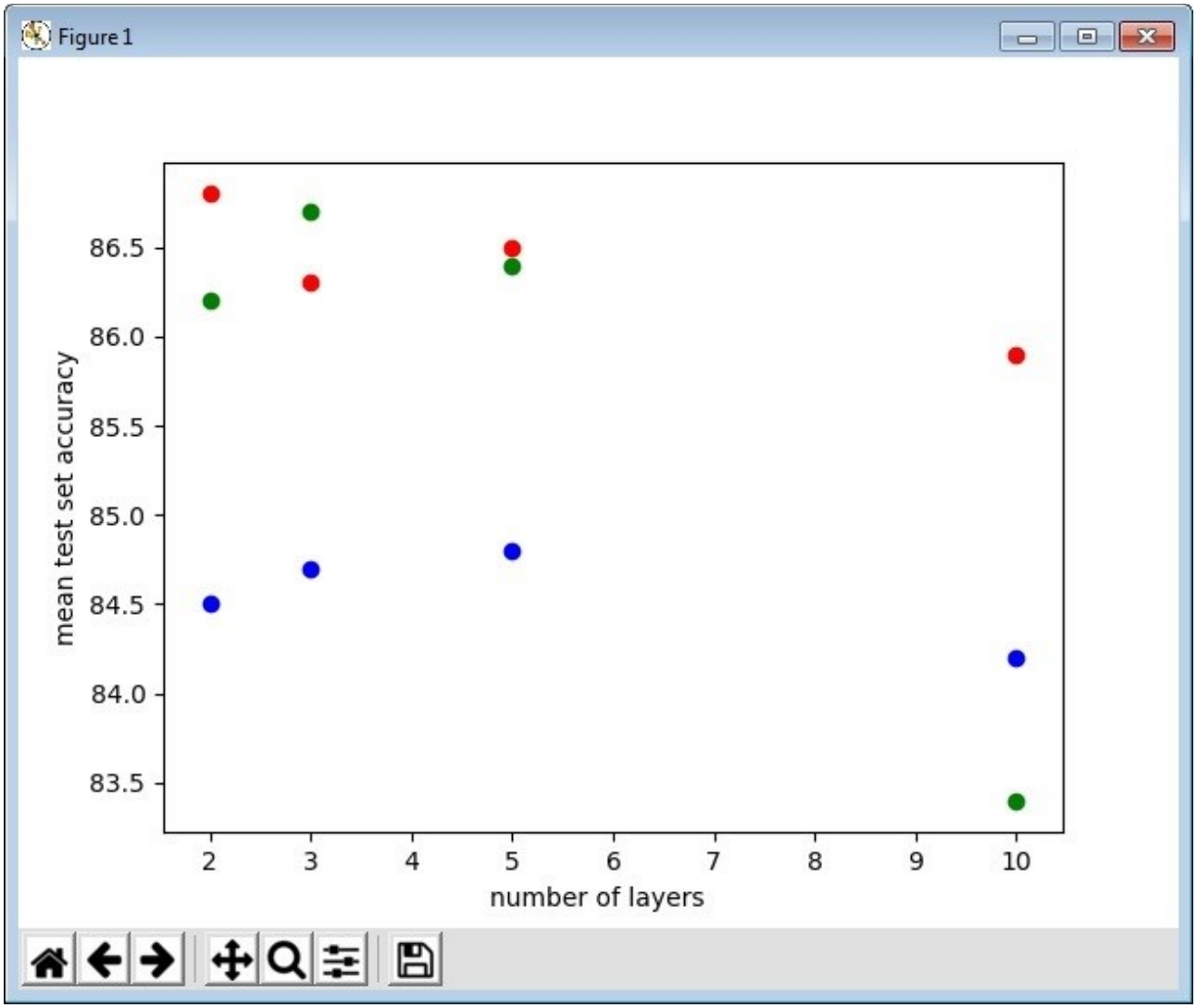

Fig.7: Accuracy media a fronte di 2, 3, 5 e 10 layers

L'aumento di **neuroni** per layer ha comportato sempre miglioramenti, più apprezzabili per bassi numeri di neuroni, meno per gli alti, evidenziando una sostanziale invarianza dai 64 in su. La sparsità dei dataset, soprattutto del dataset 1, ha fatto sì che i neuroni del primo layer, ancorché pochi, siano stati in grado di comprimere l'informazione senza perdite sostanziali.

Ancora una volta le performance sul dataset 3 sono state inferiori di quasi 2 punti percentuali rispetto agli altri due dataset.

```
#varying neurons_per_layer
lambda_array = [0]
neurons_per_layer_array = [8,16,32,64,128,256]
activation_array = ["tanh"]
epoch_array = [100]
n_of_layers_array = [3]
```

```
batch_size_array = [100]
training_set_percentage_array = [0.8]
```

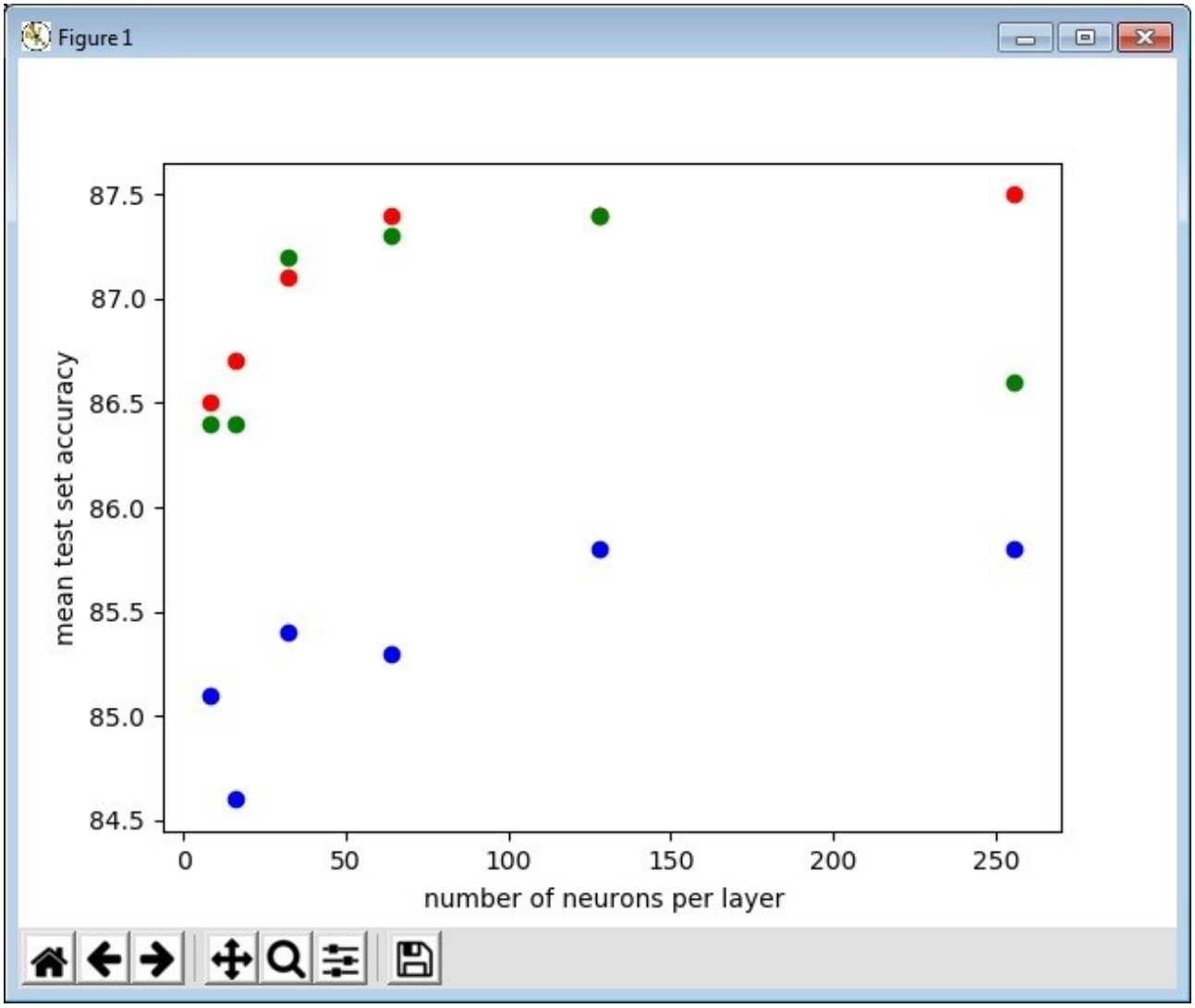

Fig.8: Accuracy media a fronte di 8, 16, 32, 64, 128 e 256 neuroni per layer

Conclusione

Concludendo, le performance sui differenti dataset, ravvisate come peggiori sul dataset 3 e migliori sul dataset 2, da una parte confermano l'importanza dell'informazione associata al modello di macchina, dall'altra suggeriscono che sembra essere preferibile una rappresentazione meno sparsa dei dati, ossia che mediare il comportamento delle macchine sui 15 giorni anziché distinguere giorno per giorno non danneggia in alcun modo l'accuracy: è pur vero che l'operazione di media aritmetica per definizione "brucia" informazione ma questo fenomeno viene ampiamente compensato dalla riduzione del numero di feature da 504 a 54, il che semplifica considerevolmente la funzione di costo.

Viene evidenziato inoltre che non è necessaria un'architettura particolarmente complessa per ottenere performance vicine a quelle massime, mentre aumentare il numero di epochs aiuta ad aumentare le performance in considerazione del fatto che i modelli non soffrono di overfitting.