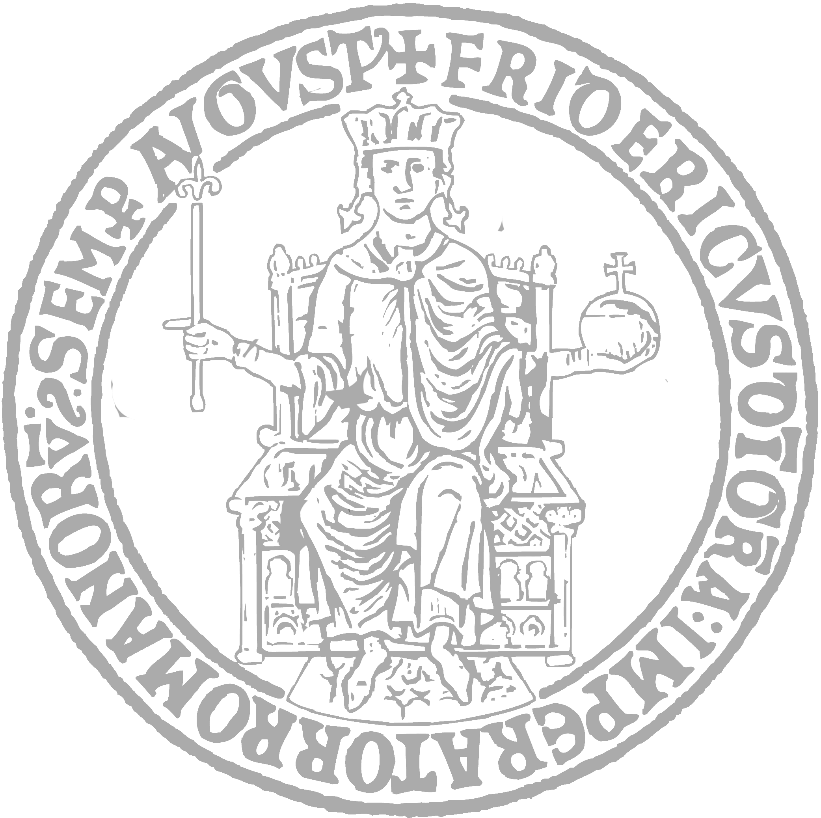
MLeA – Mod. B 2017/2018  
Progetto 5

Prof. Roberto Prevete

Djihad Boukara – N97000275

Emanuele Cioffi – N97000277



# Indice Generale

Sommario

[1. Indice Generale 2](#_Toc528862666)

[2. Indice delle figure 3](#_Toc528862667)

[3. Traccia 4](#_Toc528862668)

[4. Descrizione del problema 4](#_Toc528862669)

[5. Dataset 5](#_Toc528862670)

[6. Implementazione 5](#_Toc528862671)

[A. La rete neurale 5](#_Toc528862672)

[B. L’addestramento 6](#_Toc528862673)

[7. Test Plan 11](#_Toc528862674)

[8. K-Folding 13](#_Toc528862675)

[A. Dati Risultato 13](#_Toc528862676)

[B. Osservazioni sul K-Folding 16](#_Toc528862677)

[9. Risultati finali 17](#_Toc528862678)

[A. Combinazione: Errore Minore 18](#_Toc528862679)

[B. Combinazione: Accuratezza Migliore 19](#_Toc528862680)

[C. Combinazione: Migliori valori in media 22](#_Toc528862681)

[10. Conclusioni 23](#_Toc528862682)

[11. Appendice 24](#_Toc528862683)

[A. Kfold.m 24](#_Toc528862684)

[B. Singolo Training (Mnist.m) 25](#_Toc528862685)

[C. calculateError.m 26](#_Toc528862686)

[D. PlotBar.m 27](#_Toc528862687)

# Indice delle figure

[Figura 1 - Esempio di creazione rete neurale 5](#_Toc528862642)

[Figura 2 – Train 6](#_Toc528862643)

[Figura 3 – BackPropagation 7](#_Toc528862644)

[Figura 4 - Forward Propagation 7](#_Toc528862645)

[Figura 5 – SoftMax 8](#_Toc528862646)

[Figura 6 - Cross Entropy 9](#_Toc528862647)

[Figura 7 - Calcolo Derivate dei pesi 9](#_Toc528862648)

[Figura 8 - Discesa del gradiente 10](#_Toc528862649)

[Figura 9 – K-fold 11](#_Toc528862650)

[Figura 10 - KFold: tanH-Identity 13](file:///D:\Documents(D)\GitHub\ml_project\Documentazione.docx#_Toc528862651)

[Figura 11- KFold: tanH-ReLU 14](file:///D:\Documents(D)\GitHub\ml_project\Documentazione.docx#_Toc528862652)

[Figura 12 - KFold: sigmoid-Identity 14](file:///D:\Documents(D)\GitHub\ml_project\Documentazione.docx#_Toc528862653)

[Figura 13 - KFold: sigmoid-ReLU 15](file:///D:\Documents(D)\GitHub\ml_project\Documentazione.docx#_Toc528862654)

[Figura 14 - Kfold: top 9 Accuracy,Loss 16](#_Toc528862655)

[Figura 15 - Kfold: top 9 Loss,Accuracy 16](#_Toc528862656)

[Figura 16 - Migliori valori k-folding 17](#_Toc528862657)

[Figura 17 - Combinazione: Errore Minore (sigmoid- ReLU,0.1,250), Misurazione Accuratezza 18](#_Toc528862658)

[Figura 18 - Combinazione: Errore Minore (sigmoid- ReLU,0.1,250), Misurazione Errore Totale 18](#_Toc528862659)

[Figura 19 - Combinazione: Accuratezza Migliore (tanH,ReLU,0.01,500), Misurazione Accuratezza 19](#_Toc528862660)

[Figura 20 - Combinazione: Accuratezza Migliore (tanH,ReLU,0.01,500), Misurazione Errore Totale 19](#_Toc528862661)

[Figura 21- Combinazione: Accuratezza Migliore con eta aggiornato (tanH,ReLU,0.001,500), Misurazione Accuratezza 20](#_Toc528862662)

[Figura 22 - Combinazione: Accuratezza Migliore con eta aggiornato (tanH,ReLU,0.001,500), Misurazione Errore Totale 21](#_Toc528862663)

[Figura 23 - Combinazione: Migliori valori in media (sigmoid, identity,0.05,250), Misurazione Accuratezza 22](#_Toc528862664)

[Figura 24 - Combinazione: Migliori valori in media (sigmoid, identity,0.05,250), Misurazione Errore Totale 22](#_Toc528862665)

# Traccia

**PARTE A.**

“• Progettazione ed implementazione di funzioni per simulare la propagazione in avanti di una rete neurale multi-strato con almeno:

* due strati di pesi, con la sigmoide come funzione di output dei nodi interni e l'identità come funzione di output dei nodi di output.

(FACOLTATIVO: permettere all'utente di implementare reti con più di uno strato di nodi interni e con qualsiasi funzione di output per ciascun strato).

• Progettazione ed implementazione di funzioni per la realizzazione della back-propagation per reti neurali multi-strato con almeno:

* due strati di pesi, con la sigmoide come funzione dioutput dei nodi interni e l'identità come funzione di output dei nodi di output, con la somma dei quadrati come funzione di errore.

(FACOLTATIVO: permettere all'utente di realizzare la back-propagation con più di uno strato di nodi interni, con qualsiasi funzione di output per ciascun strato e con qualsiasi funzione di errore derivabile rispetto all'output).”

**PARTE B.**

“Si consideri come input le immagini raw del dataset mnist. Si ha, allora,un problema di classificazione a C classi, con C=10. Si estragga opportunamente un dataset globale di N coppie (ad esempio, N=200). Si fissi la discesa del gradiente come algoritmo di aggiornamento dei pesi, ed una rete neurale con un unico strato di nodi interni. Si scelgano gli iper-parametri del modello, cioè eta della regola di aggiornamento ed il numerodi nodi interni, sulla base di un approccio di cross-validation k-fold (ad esempio k=5).

Scegliere e mantenere invariati tutti gli altri "parametri" come, ad esempio, le funzioni di output e la funzione di errore. Se è necessario, per questioni di tempi computazionali espazio in memoria, si possono ridurre (ad esempio dimezzarle) le dimensioni delle immagini raw del dataset mnist (ad esempio utilizzando in matlab la funzione imresize).”

# Descrizione del problema

Per risolvere tale problema di classificazione con apprendimento supervisionato, si vuole implementare un modello di rete neurale artificiale feed forward, multistrato e full connected, che permetta all’utente di specificare gli iperparametri e le funzioni di attivazione ed errore desiderate. La rete sarà sviluppata in maniera quanto più generale possibile, nell’ottica di poter accogliere input di diverse tipologie .L’ambiente di sviluppo scelto per implementare tale modello è Matlab. Si vuole inoltre studiare tramite la tecnica di cross validation K-Fold, il miglioramento o il peggioramento delle performance di riconoscimento della rete neurale al variare degli iperparametri, in particolare learning rate, funzioni di attivazione e numero di nodi interni dello strato hidden.

Utilizzando il dataset per caratteri scritti a mano MNIST, dove le classi target discrete consistono dei caratteri da ‘0’ a ‘9’, verranno svolti test in ottica di classificazione (e non regressione), utilizzando come funzione d’errore la Cross Entropy, combinata con la funzione Soft Max per il post processing.

# Dataset

Il dataset impiegato per l’addestramento della rete è il database di caratteri scritti a mano Mnist. Tale dataset è composto da 60000 esempi di caratteri numerici (da 0 a 9) corredati di label compilate a mano e di 10000 esempi ulteriori per il testing. Il dataset è pubblicamente disponibile, e per leggerne i dati ed acquisire in campioni in formato leggibile per Matlab, ci si è serviti delle funzioni sviluppate all’università di Stanford (allegate al progetto, si veda il file *loadMNISTImages.m*). Tali funzioni, una volta letto l’intero dataset, restituiscono una matrice di dimensioni 784\*60000 (per il training set, 784\*10000 per il test set), in cui le colonne rappresentano i singoli campioni, e le righe rappresentano le feature dei campioni. Le 784 feature rappresentano il totale dei pixel per singola immagine grande 28x28, e descrivono tramite un valore numerico compreso tra 0 (il bianco) e 255 (il nero) il contenuto del pixel (in scala di grigi). Un singolo campione del training/test test può quindi essere visto come un vettore colonna della matrice generata. Tale vettore viene poi diviso valore per valore, per 255, per normalizzare i valori di input da consegnare alla rete.

# Implementazione

La rete neurale è stata quindi implementata in un’ottica di flessibilità e facilità di riutilizzo, specie considerando le necessità dettate dalla Cross Validation di poter eseguire l’addestramento su reti sempre diverse (nel senso di scelta di iperparametri). Per ciò, la rete sviluppata permette all’utilizzatore di scegliere un numero arbitrario di strati hidden tramite l’utilizzo di array contenitori, di specificare le funzioni di attivazione per tali strati e per lo strato di output, ed infine di specificare la funzione d’errore da utilizzare. Sebbene gli script corredati per il testing siano pensati per l’utilizzo della rete con il dataset MNIST, nulla vieta di utilizzare la rete con altri tipi di dataset a patto di fornire alla rete i dati nel formato giusto.

## La rete neurale

Il cuore della rete neurale è osservabile nella funzione *neuralNet.m*, che permette di creare la rete neurale specificando dei parametri desiderati:

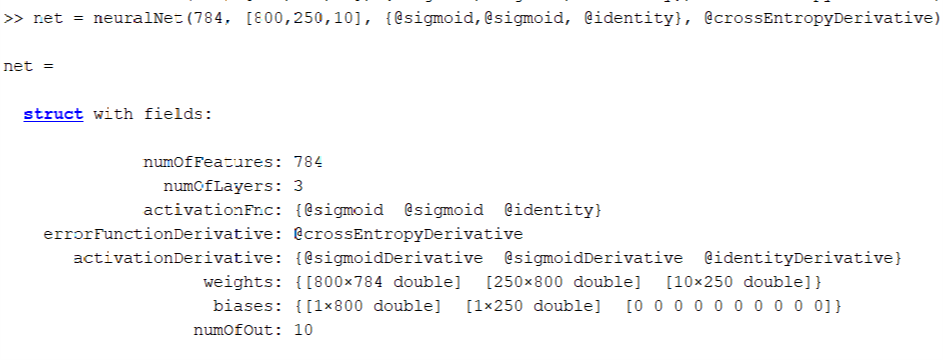


Figura 1 - Esempio di creazione rete neurale

In Figura 1 un esempio di creazione di rete neurale con due strati hidden rispettivamente formati da 800 e 250 nodi interni utilizzanti la funzione d’attivazione Sigmoide, e uno strato d’output da 10 nodi con funzione di attivazione Identità (il numero di input e output sono in questo caso quelli necessari al riconoscimento dei dati Mnist). La Cross Entropy è la funzione d’errore utilizzata in questo caso, ma è stata fornita l’implementazione anche per la somma dei quadrati.

Data la necessità di dover parametrizzare il calcolo della derivata (diverso per ogni funzione di attivazione) necessario nella backpropagation, si era pensato in un primo momento di adoperare uno dei toolbox di Matlab per la valutazione delle espressioni e il calcolo delle derivate su queste; tale soluzione si è rivelata però troppo dispendiosa in termini di tempo, in un contesto dove i tempi di esecuzione sono già lunghi di per sé. Pertanto, si è scelto di adottare una soluzione diversa, ossia di fornire in allegato al codice alcune funzioni di attivazione (Sigmoide, Identità, Tangente Iperbolica, ReLU), e le relative implementazioni delle derivate, che l’algoritmo potrà poi chiamare a seconda dei casi. I codici relativi alle funzioni d’attivazione sono stati allegati al progetto (es: *sigmoid.m)*, insieme alle rispettive funzioni per la derivata (es: *sigmoidDerivative.m)*. L’utente dovrà quindi solo specificare le funzioni d’attivazione scelte per ogni strato tramite l’handle matlab (@nomeFunzione), e il sistema provvederà ad accoppiargli le relative funzioni per il calcolo della derivata all’atto della creazione della rete.

Per gestire i pesi della rete, è stato creato un array cell di matrici *weights*, che comprende una matrice di pesi per ogni layer della rete. Ognuna di queste matrici conta una riga per ogni nodo del layer relativo alla matrice, ed una colonna per ogni nodo del layer precedente. Osservando quindi tale matrice riga per riga, si otterrà l’insieme dei pesi/archi entranti nel nodo relativo alla riga, se invece la si osserva per colonna, si otterranno tutti gli archi uscenti del nodo dello strato precedente relativo alla colonna della matrice (ed entranti nei rispettivi nodi dello strato corrente). I bias sono rappresentati tramite pesi fittizi, come matrici con una riga e numero di colonne pari al numero di nodi del layer relativo per facilitare i calcoli.

## L’addestramento

La rete è stata sviluppata in modo da poter essere addestrata tramite l’algoritmo della discesa del gradiente e gli script corredati per l’utilizzo permettono di specificare un learning di tipo Online, MiniBatch e Batch.

Cuore dell’addestramento è visibile nella funzione ’train’ presente nel file *train.m*.



Figura 2 – Train

Come visibile in Figura 2, è possibile partizionare il Training Set in porzioni grandi *batchSize* ed effettuare l’addestramento della rete su tali porzioni. Un singolo training consisterà quindi di una chiamata alla funzione *backPropagation.m,* che dopo aver effettuato forward propagation (*forwardPropagation.m)* e calcolo dei delta, ritornerà le derivate rispetto ai pesi e i bias che verranno poi impiegate dalla funzione *gradientDescent.m* per effettuare l’aggiornamento dei pesi.



Figura 3 – BackPropagation



Figura 4 - Forward Propagation

In Figura 4 è leggibile l’implementare della **Forward Propagation**. Il codice pre-alloca due cell array per un numero di elementi pari a quello degli strati della rete e calcolerà strato per strato i valori di A e Z, rispettivamente l’input e l’output di ogni nodo. Nel dettaglio, le formule implementate risultano essere:

E

Con *l* indice di layer e *g* funzione d’attivazione per lo strato (con z che assumerà valore x per lo strato 1).  
Il calcolo da effettuare in forma matriciale è descritto da:

L’implementazione è quindi stata effettuata tramite moltiplicazioni tra matrici (caso batch/minibatch) o vettori per matrice (caso online) così ottenuti. Per chiarire, ilcell array *a* conterrà in prima posizione, la matrice risultante dal prodotto tra la matrice di BatchSize righe e 784 (l’input) colonne PER la matrice di righe pari al numero di nodi del layer 1 e 784 colonne da trasporre, i cui elementi saranno sommati al bias.

Vi è poi la possibilità di applicare una funzione di post process ai nodi z così ottenuti, in particolare nel nostro caso questa applicherà il softmax per una più chiara classificazione degli esempi rispetto alle label formate da un vettore di dieci componenti, di cui nove impostate al valore 0 ed una al valore 1, in corrispondenza dell’indice corrispondente al valore assegnato.



Figura 5 – SoftMax

Il Softmax viene impiegato quindi come post processing su funzioni di attivazione lineari per normalizzare la somma dei valori (nel nostro caso dei 10 output) portandola a 1, dando così maggiore *peso* in termini di influenza ai valori di grandezza superiore all’interno dei 10 possibili target.

In Figura 3 è possibile osservare l’implementazione della **Back Propagation**. Questa si compone di una chiamata alla forward propagation, del calcolo dei delta dello strato di output e di quelli precedenti a ritroso ed infine del calcolo delle derivate rispetto ai pesi. Il calcolo dei delta dello strato di output è effettuato secondo la formula:

Dove F’ rappresenta la derivata della funzione d’attivazione dello strato i, ai il valore di uscita di un nodo dello strato i-esimo prima della funzione d’attivazione. La derivata dell’errore sull’esempio rispetto all’output Z è stata quindi resa parametrica per permettere di specificare diverse funzioni d’errore. Nel caso d’utilizzo in questione (con Cross Entropy come funzione d’errore su un output a cui è stato applicato softmax), la funzione passata come parametro (*net*.*errorFunctionDerivative*) sarà quindi quella visibile in Figura 5.



Figura 6 - Cross Entropy

Per i restanti delta da calcolare negli strati interni, la formula implementata risulta essere:

Dove g’ rappresenta la derivata della funzione d’attivazione per lo strato i-esimo, applicata sul valore pre-attivazione del nodo i allo strato l, e i termini della sommatoria sono i prodotti dei pesi uscenti dal nodo in questione per i delta associati al nodo dello strato successivo in entrata al peso. I singoli delta vengono calcolati a livello vettore/strato da matlab (*delta{layer*} nel codice) moltiplicando per ognuno la matrice dei pesi dello strato successivo per quella dei delta già calcolati al layer +1. Prima di terminare, la funzione back propagation calcolerà le derivate parziali rispetto ai pesi.



Figura 7 - Calcolo Derivate dei pesi

In Figura 7 è possibile vedere il codice per il calcolo delle derivate rispetto ai pesi. Il calcolo effettuato per ottenere tali derivate è:

Dove Znk assumerà valore Xi (ossia input) nel caso di n=1. BIAS



Figura 8 - Discesa del gradiente

In Figura 8 l’implementazione della discesa del gradiente, che aggiorna pesi e bias secondo:

# Test Plan

Per valutare quali iperparametri e quali combinazioni di funzioni di attivazione risultino più performanti in fase di classificazione, si è realizzata una procedura di cross validation su un sottoinsieme ristretto del training set, per testare la capacità di generalizzare le valutazioni da parte della rete. In particolare, si è utilizzata una strategia di K-Folding, visibile nello script *kfold.m*. Tale strategia consiste nel dividere un training set campione in k parti, fissare una delle parti come test set e utilizzare le restanti k-1 come porzione per l’addestramento, ripetendo tale operazione ruotando k volte la k-esima porzione di test. I risultati così ottenuti vengono poi mediati tra le varie rotazioni per fornire un risultato meno sensibile all’overfitting. Il k-fold è stato quindi impiegato per scegliere i migliori iperparametri tra un campione di combinazioni da valutare.

Il training set è stato limitato a 3000 esempi, e per i test effettuati si è scelto *dieci* come valore per il parametro K, formando così 10 raggruppamenti contenenti 300 esempi ciascuno, di cui 9 utilizzati per l’addestramento (in blu in Figura 9), ed uno per il test (in giallo), ruotando a turno su tutte e 10 le posizioni.

Figura 9 – K-fold

Il test plan programmato prevede di far spaziare i seguenti parametri, tra i valori di fianco riportati:

* Numero di **nodi** per lo strato interno:
  + 250
  + 500
  + 800
* Coppia di **funzioni** **d’attivazione** per strato interno ed output:
  + Tangente Iperbolica (tanH) – Unità Lineare Rettificata (ReLU)
  + Sigmoide – ReLU
  + Tangente Iperbolica – Identità
  + Sigmoide - ReLU
* **Eta** (Learning Rate)
  + 0.1
  + 0.05
  + 0.01
  + 0.001

Tali parametri sono stati scelti a seguito di una breve scrematura preventiva, effettuata osservando il funzionamento delle funzioni su determinati parametri nelle fasi iniziali di sviluppo della rete; sono quindi stati scelti i parametri che forniscono un quadro d’insieme più descrittivo e che abbracci una variegata serie di casistiche, escludendo casi limite o poco interessanti (ad esempio learning rate troppo piccoli che risultano sicuramente meno efficaci di vari ordini di grandezza rispetto a tutti gli altri)

Le 9 fasi di addestramento e la singola di test che contraddistinguono un k-folding sono quindi eseguite per un totale di 3x4x4 volte, incrociando le possibili combinazioni, utilizzando la modalità di apprendimento online.

Una volta ottenuta la tripla di parametri migliore, osservando come misura d’efficacia la deviazione standard dell’errore ottenuto dalla rotazione, si è proceduto ad effettuare dei test di valutazione finali in modalità online e minibatch per confronto.

# K-Folding

## Dati Risultato

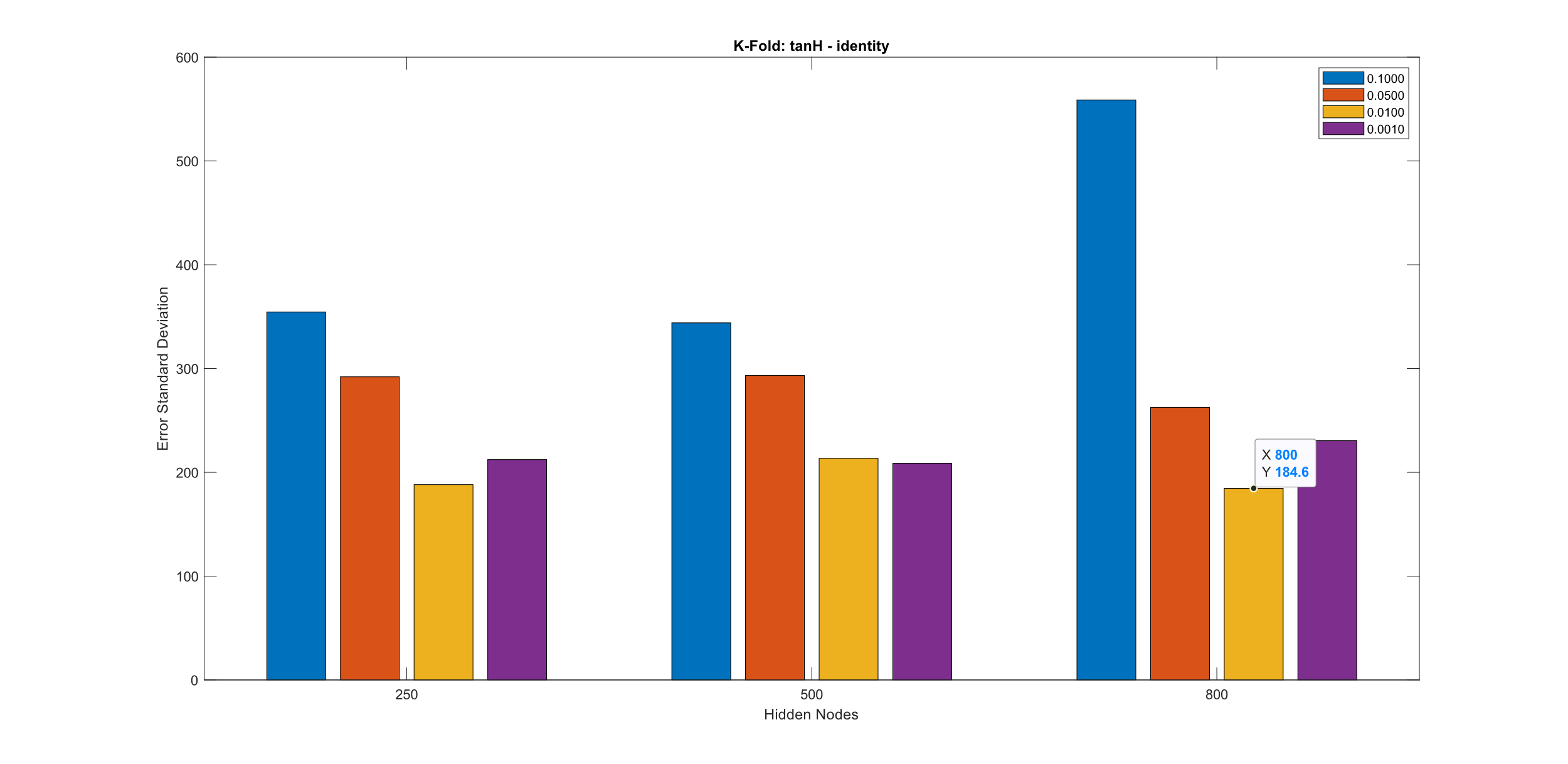
Di seguito vengono mostrati i grafici che descrivono i dati estrapolati dalla cross validation per la ricerca degli iperparametri migliori. I colori delle barre rappresentano il learning rate utilizzato mentre i raggruppamenti sull’asse delle ascisse indicano il numero di hidden nodes utilizzato. Il valore numerico ottenuto rappresenta la deviazione standard dell’errore dato cross entropy. I grafici corrispondono alle 4 coppie di funzioni di output utilizzate come campione d’esame. Per ogni grafico è stata segnato con un’etichetta il valore d’errore minimo rispetto a tutte le combinazioni.

Figura 10 - KFold: tanH-Identity

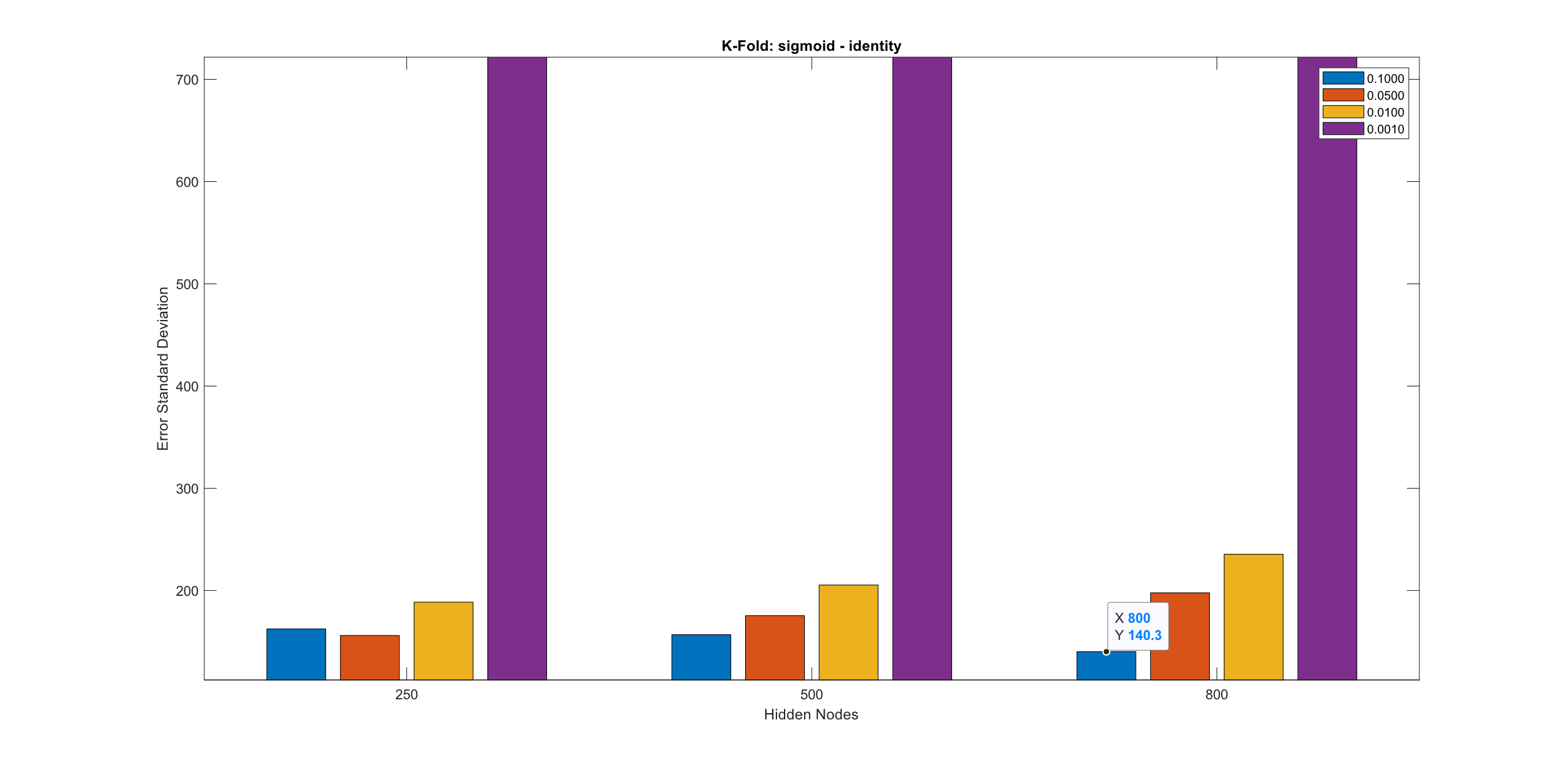
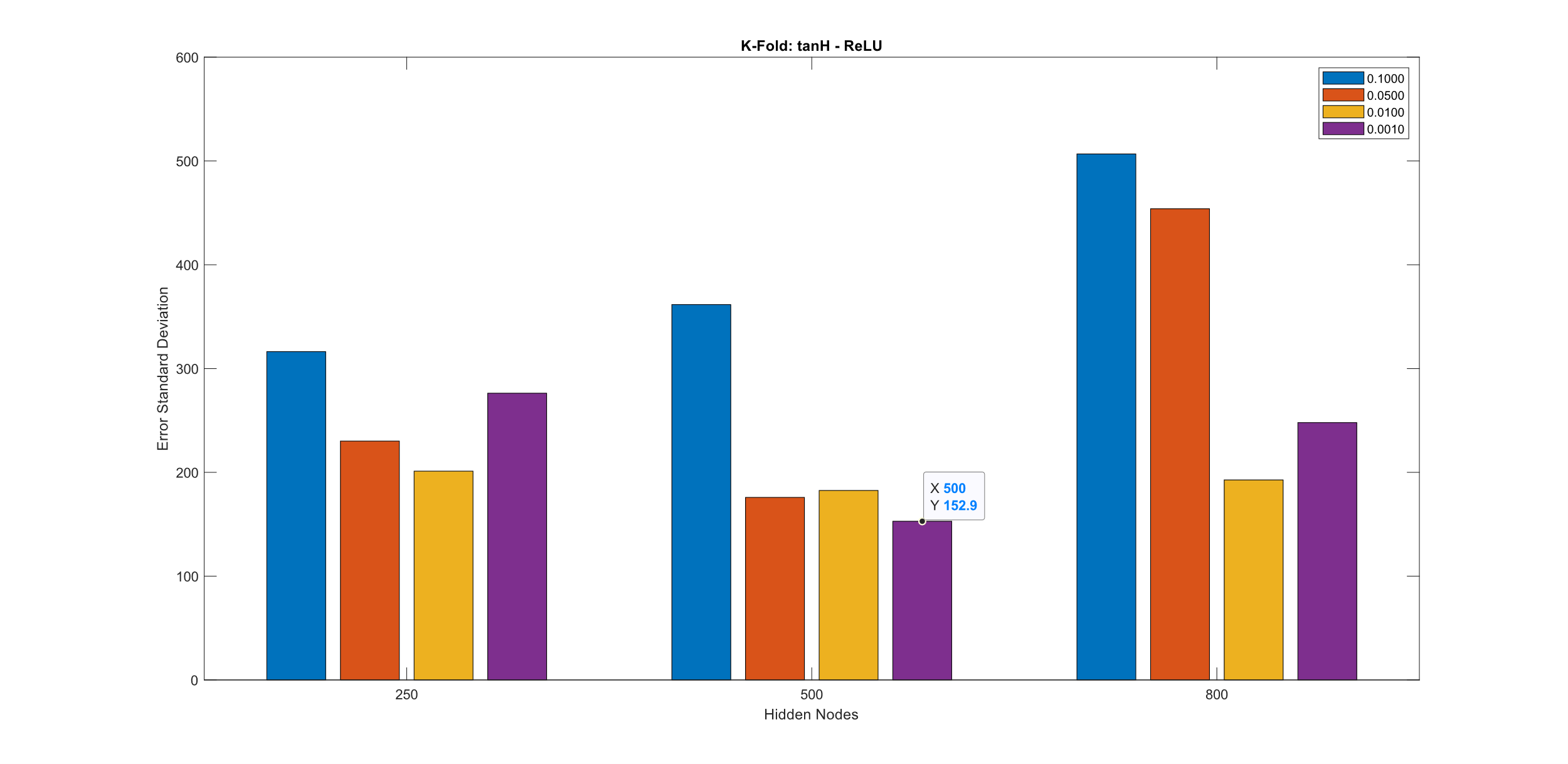


Figura 11- KFold: tanH-ReLU

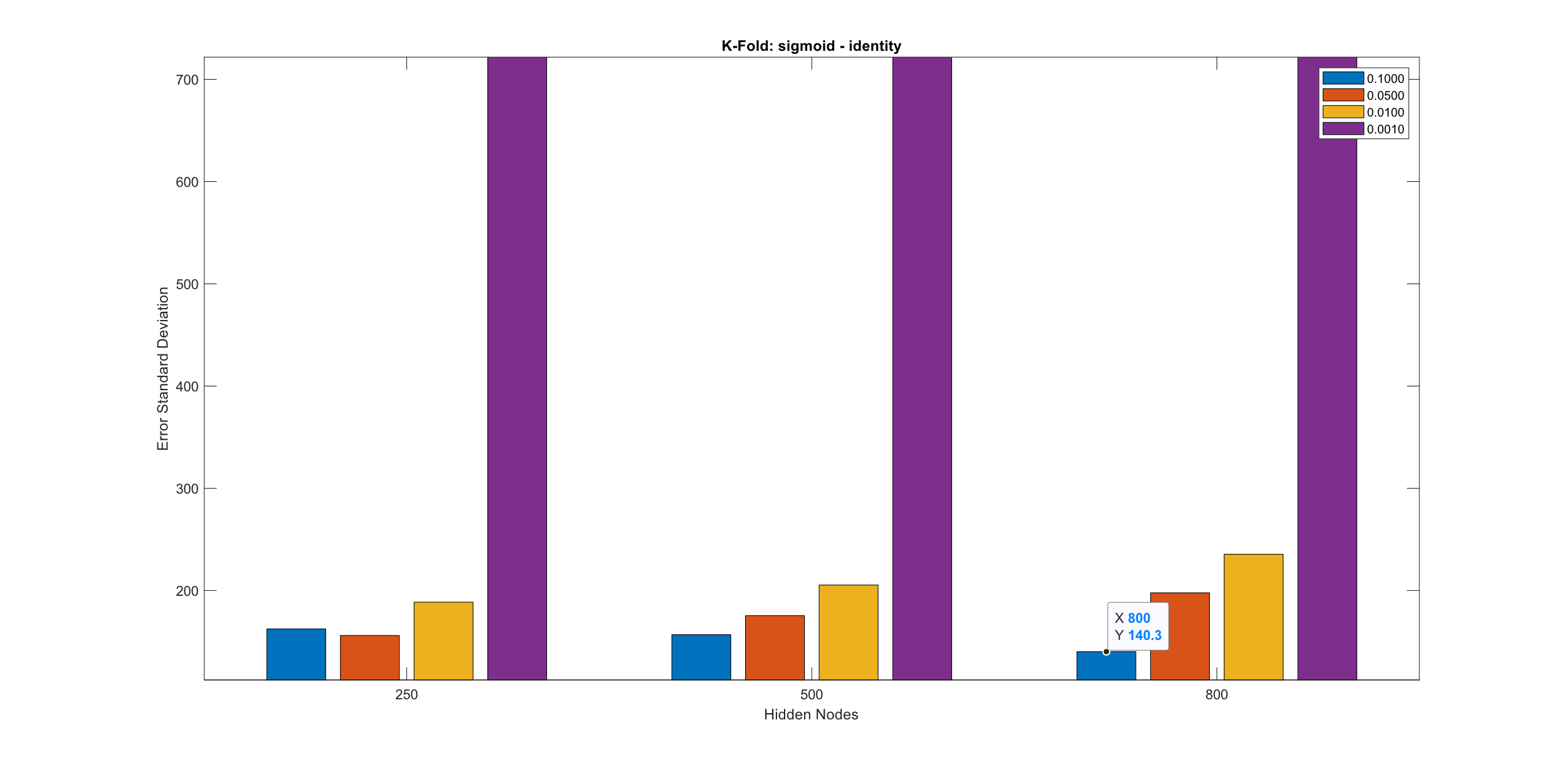


Figura 12 - KFold: sigmoid-Identity

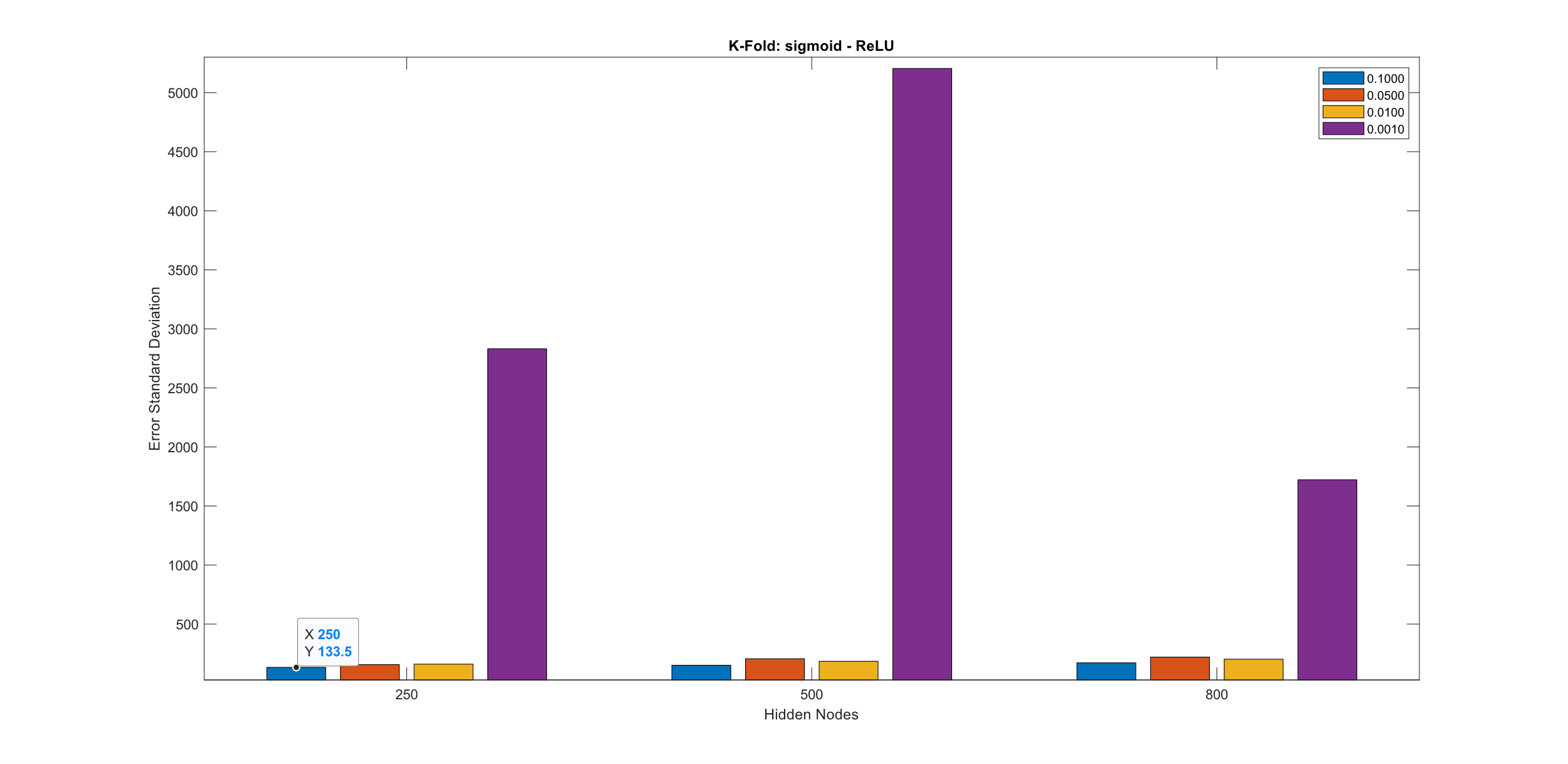


Figura 13 - KFold: sigmoid-ReLU

Il tempo d’esecuzione per l’intero k-folding è ammontato a circa 3 ore e 10 minuti.

## Osservazioni sul K-Folding

Tra le considerazioni che è possibile effettuare osservando i grafici figura la differenza nei risultati d’apprendimento quando si usa la sigmoide rispetto alle altre funzioni. Infatti leggendo i risultati, è possibile vedere come l’utilizzo di tale funzione richieda un learning rate alto per funzionare al meglio, e come visibile nei due casi d’uso (Figura 12,Figura 13), l’uso combinato di sigmoide e l’ultimo learning rate provato (0.001) si traduce in un valore d’errore molto più elevato e fuori scala (e quindi una peggiore classificazione) rispetto a tutte le altre combinazioni. Per quanto riguarda il numero di nodi interni, si evince come l’aumento del numero di nodi dello strato hidden non si traduca in un automatico miglioramento delle performance. Preventivi test effettuati sulla rete hanno inoltre evidenziato nel migliore dei casi, una performance di classificazione scarsa, e nel peggiore il problema dell’esplosione dei gradienti, quando vengono utilizzati learning rate alti e numero di nodi interni elevati. Ciò è visibile nei grafici relativi ai test in cui è stata utilizzata la tangente iperbolica: i valori peggiori sono infatti osservabili in prossimità dell’eta più grande (0.1) e il numero di nodi interni più elevato (800).

Come si evince dall’ultimo grafico (Figura 13), il valore più basso in termini di deviazione standard dell’errore della cross entropy è riscontrato nel test composto da:

* Funzioni: Sigmoide - ReLU:
* Eta: 0.1
* Numero di nodi interni: 250.

Tuttavia, al migliore risultato in termini di errore minimo, non corrisponde un miglior risultato in termini di accuracy. Ciò ci ha quindi spinto a valutare i risultati, mettendo a confronto le migliori combinazioni ottenute in termini di errore minimo e di accuracy massima.



Figura 14 - Kfold: top 9 Accuracy,Loss



Figura 15 - Kfold: top 9 Loss,Accuracy

In Figura 14 sono mostrati i migliori 9 risultati ottenuti dal k-folding, ordinati per migliore accuratezza in ordine decrescente. In Figura 15 invece sono mostrati i migliori 9 questa volta ordinati per deviazione standard del costo della cross entropy in ordine cresente. Analizzando i valori è stato quindi possibile estrapolare la migliore tripletta di iperparametri in termini di errore minimo, di accuratezza massima, ed infine una tripletta di miglior valore medio tra accuratezza ed errore rispetto ai migliori 10 risultati di ciascun parametro.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Hidden layer A.F | Output Layer A.F. | Eta | Hidden nodes | Mean Accuracy | C.E. Standard deviation |
| Best Average Score | sigmoid | identity | 0,05 | 250 | 87,77 | 156,04 |
| Highest Accuracy | tanH | ReLU | 0,01 | 500 | 88,37 | 182,55 |
| Lowest Error | sigmoid | ReLU | 0,1 | 250 | 87,6 | 133,53 |

Figura 16 - Migliori valori k-folding

# Risultati finali

Ottenute le 3 combinazioni di parametri migliori rispetto ai criteri elencati nel paragrafo precedente, si è proceduto ad effettuare dei test finali, **utilizzando** training set più grandi, con molteplici epoche e confrontando le modalità di apprendimento Online e **Minibatch** per ognuna di queste. Nella modalità di apprendimento Online, l’aggiornamento dei pesi segue dopo ogni addestramento su singolo esempio, realizzando una rete in grado di convergere ad un minimo in minor numero di epoche impiegando però un tempo d’esecuzione maggiore. La modalità minibatch invece effettua l’addestramento contemporaneamente su porzioni di esempi di grandezza fissata (ad esempio 32 esempi per addestramento). La modalità batch (qui non trattata) può essere vista come un caso particolare della minibatch in cui l’aggiornamento dei pesi viene effettuato una singola volta per epoca sull’intero training set. Per i nostri test è stato scelto un batchSize (grandezza delle partizioni per il minibatch learning) di 32 ed un training set grande complessivamente 3200 esempi. Dopo ogni epoca di apprendimento la rete è stata valutata sull’intero test set di 10000 esempi. Le misurazioni dell’errore consistono della deviazione standard dell’errore sui test effettuati.

## Combinazione: Errore Minore

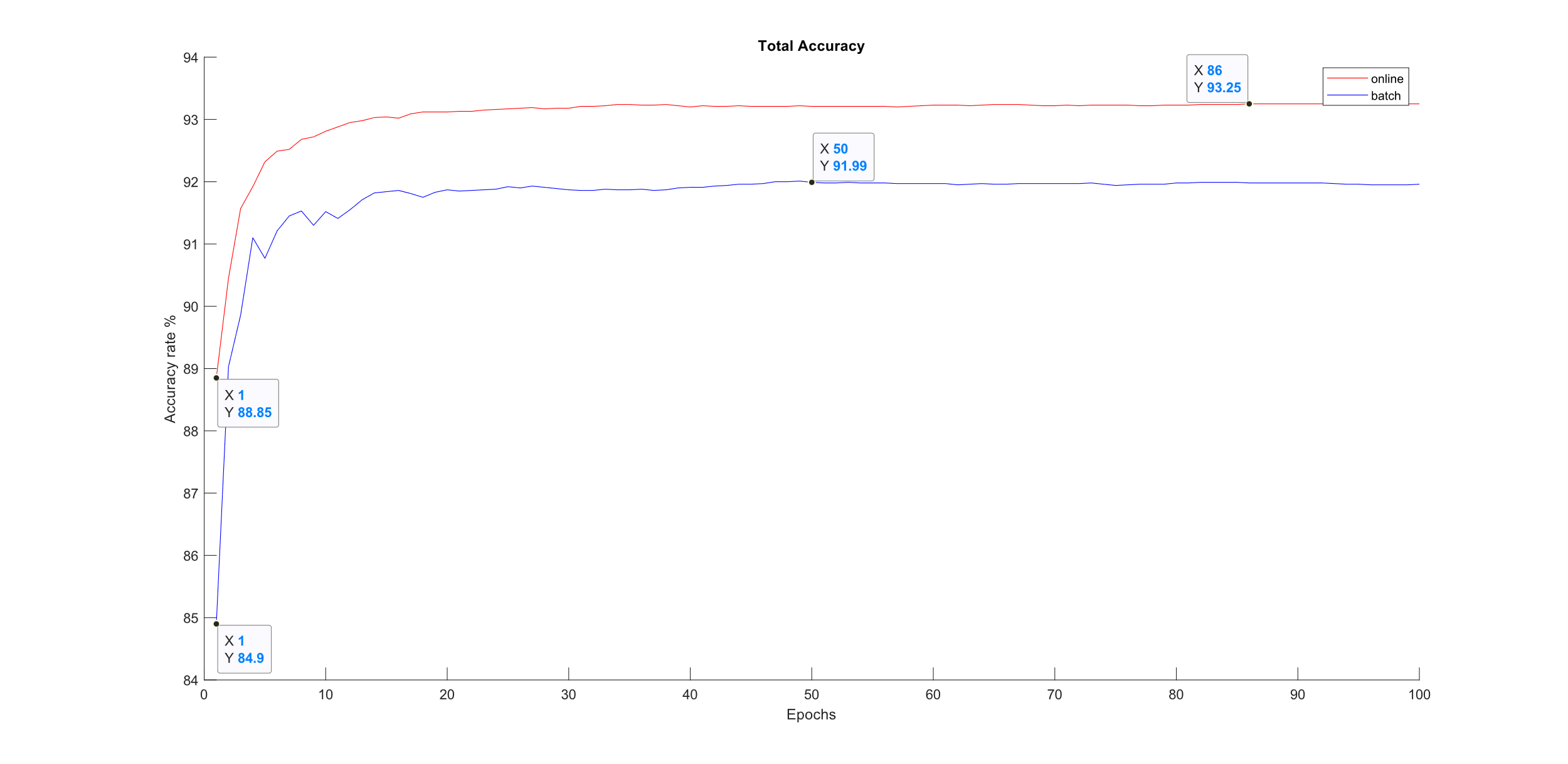


Figura 17 - Combinazione: Errore Minore (sigmoid- ReLU,0.1,250), Misurazione Accuratezza

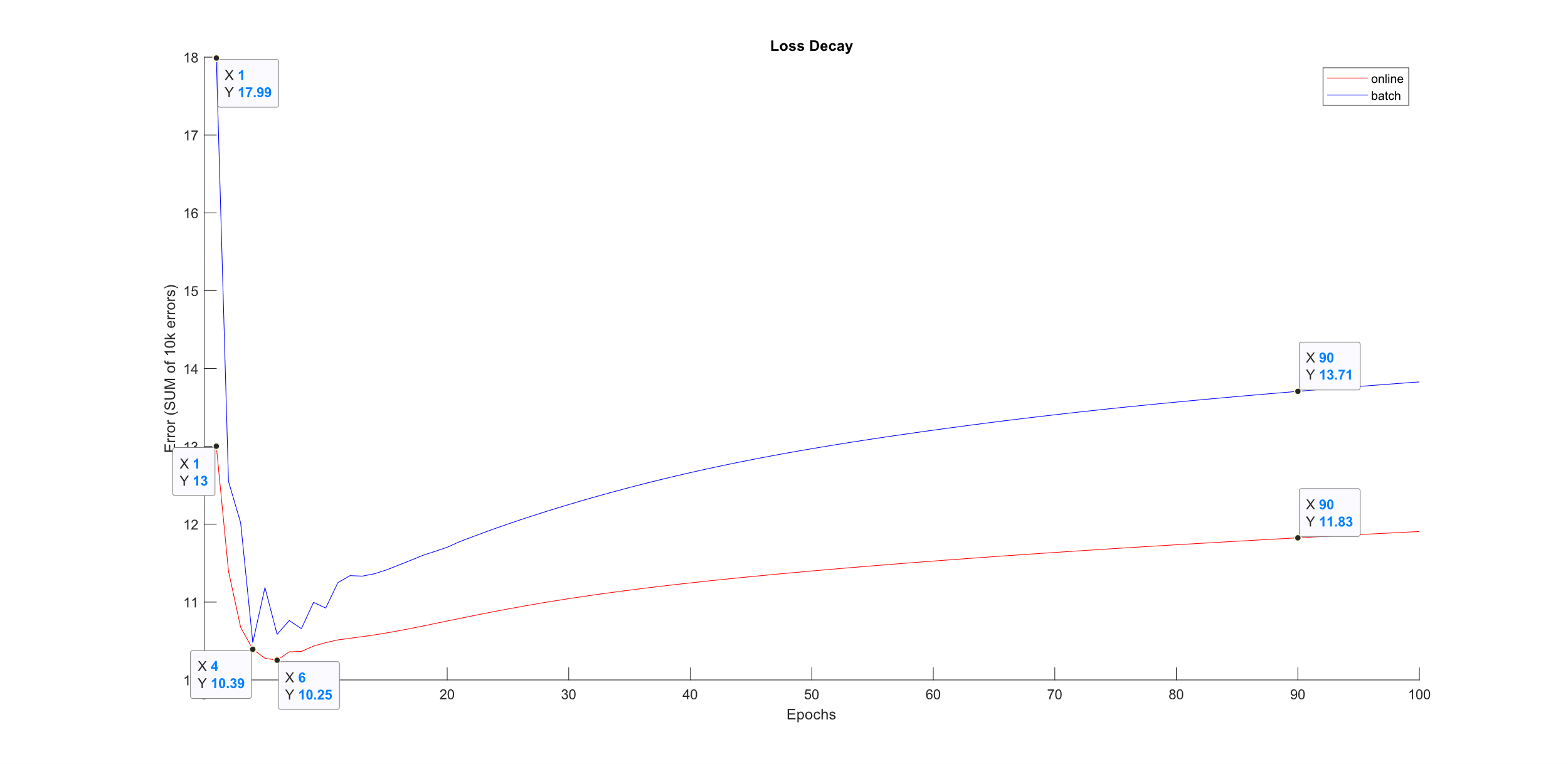


Figura 18 - Combinazione: Errore Minore (sigmoid- ReLU,0.1,250), Misurazione Errore Totale

La combinazione mostrata appena sopra presenta un comportamento prevedibile e in linea con quanto ci si aspetterebbe. Il dato in Figura 17 mostra infatti un valore di accuratezza dopo la prima epoca di addestramento, più alto per l’addestramento di tipo online, ed analogamente un valore di errore minore alla prima epoca dell’online learning, come visibile in Figura 18. L’errore totale propagato risulta immediatamente più basso nel caso online rispetto all’addestramento minibatch alla prima epoca, e raggiunge inoltre un punto di minimo alla sesta epoca nel caso del learning online, per poi risalire all’aumentare delle epoche in entrambe le modalità, traducendosi in un possibile overfitting dell’addestramento. Le differenze in termini di tempo di computazione tra le due modalità risultano evidenti: il tempo totale di addestramento per cento epoche nel caso batch è ammontato a 33 secondi, 453 (8 minuti circa) invece per il caso online.

## Combinazione: Accuratezza Migliore

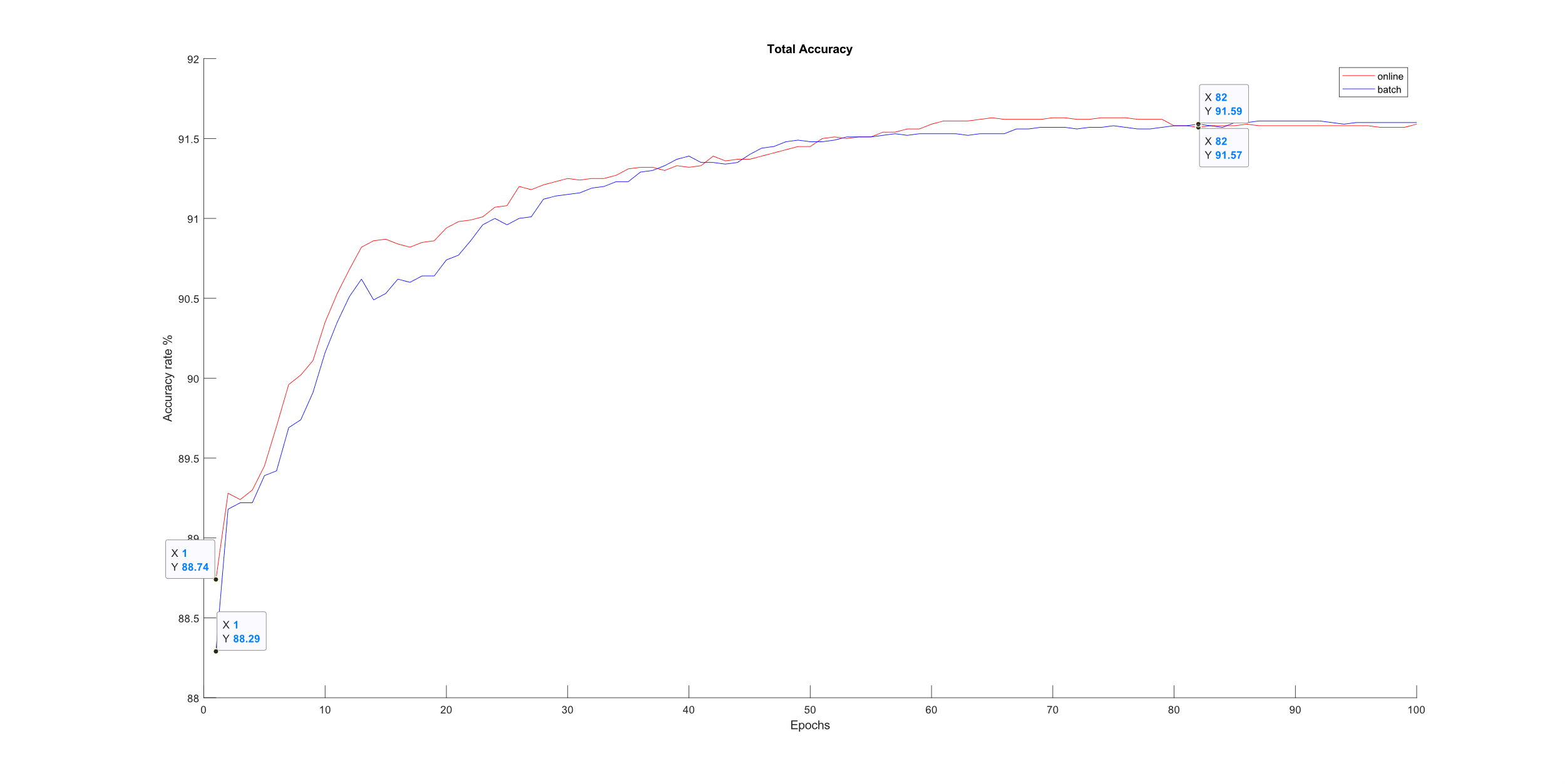


Figura 19 - Combinazione: Accuratezza Migliore (tanH,ReLU,0.01,500), Misurazione Accuratezza

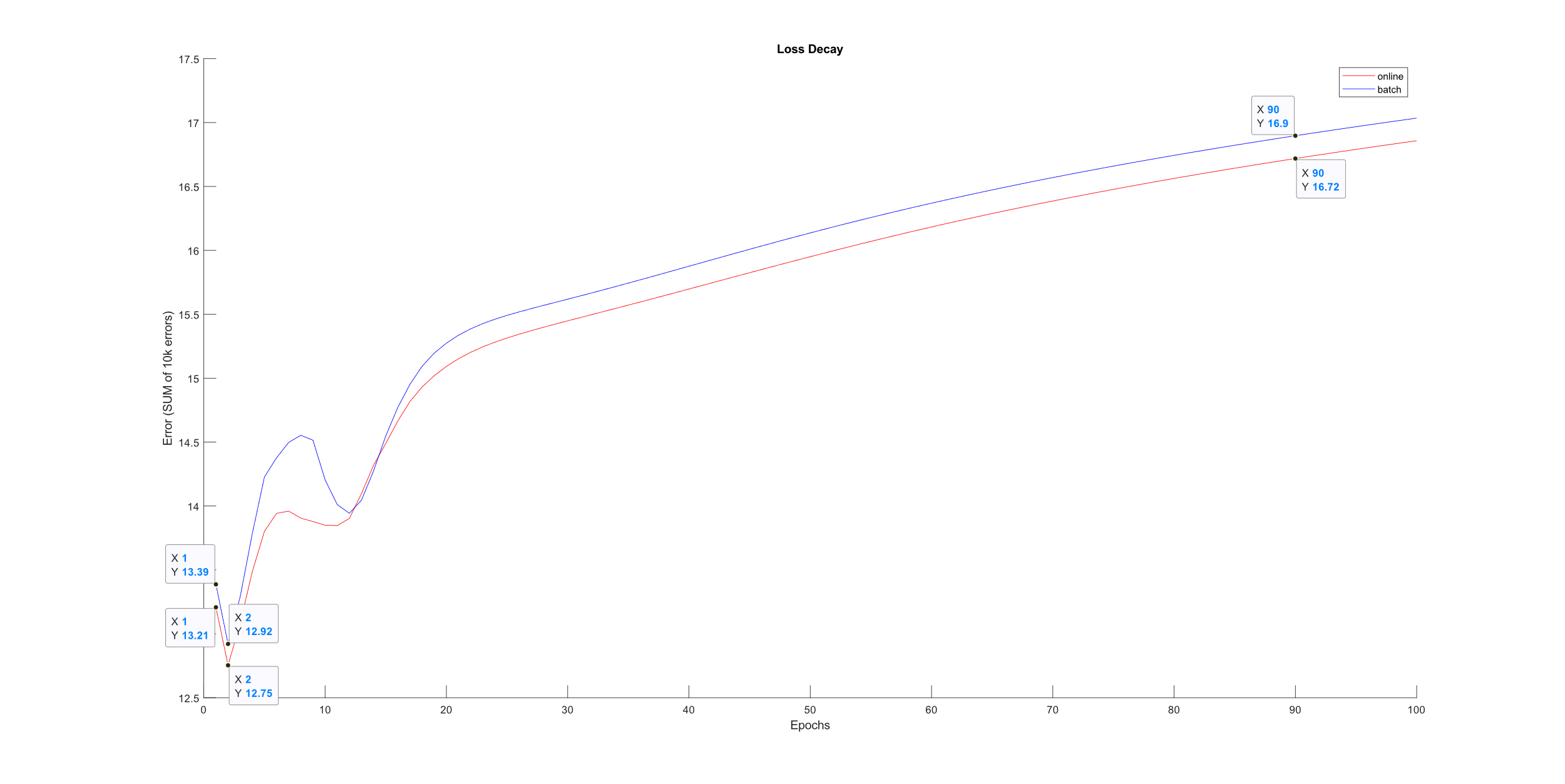


Figura 20 - Combinazione: Accuratezza Migliore (tanH,ReLU,0.01,500), Misurazione Errore Totale

La combinazione sopra mostrata esibisce invece un comportamento non aspettato. L’accuratezza infatti mostra un andamento piuttosto irregolare, registrando valori peggiori in caso di modalità online rispetto alla minibatch in più punti. Ma è soprattutto il grafico in Figura 20 a mostrare un comportamento non convincente, in cui l’errore accumulato, tende a crescere piuttosto che a diminuire, all’aumentare delle epoche, verosimilmente portando la rete a peggiore all’aumentare del numero di addestramenti. Il sospetto poi rivelatosi fondato che l’aumentare del numero di esempi del training set utilizzato, abbia portato ad un comportamento del genere in corrispondenza di un numero di hidden nodes superiore a quello usato nelle altre combinazioni rapportato al learning rate utilizzato, ci ha spinto a riprovare la stessa combinazione con gli stessi iperparametri e numero di campioni d’esempio, ma con un learning rate più basso (0.001). I risultati che seguono mostrano quindi il comportamento con un eta più adeguato, che riportano la rete a performare come ci aspetterebbe, confermando la necessità di tenere in considerazione il learning rate anche quando si variano il numero di esempi impiegati.

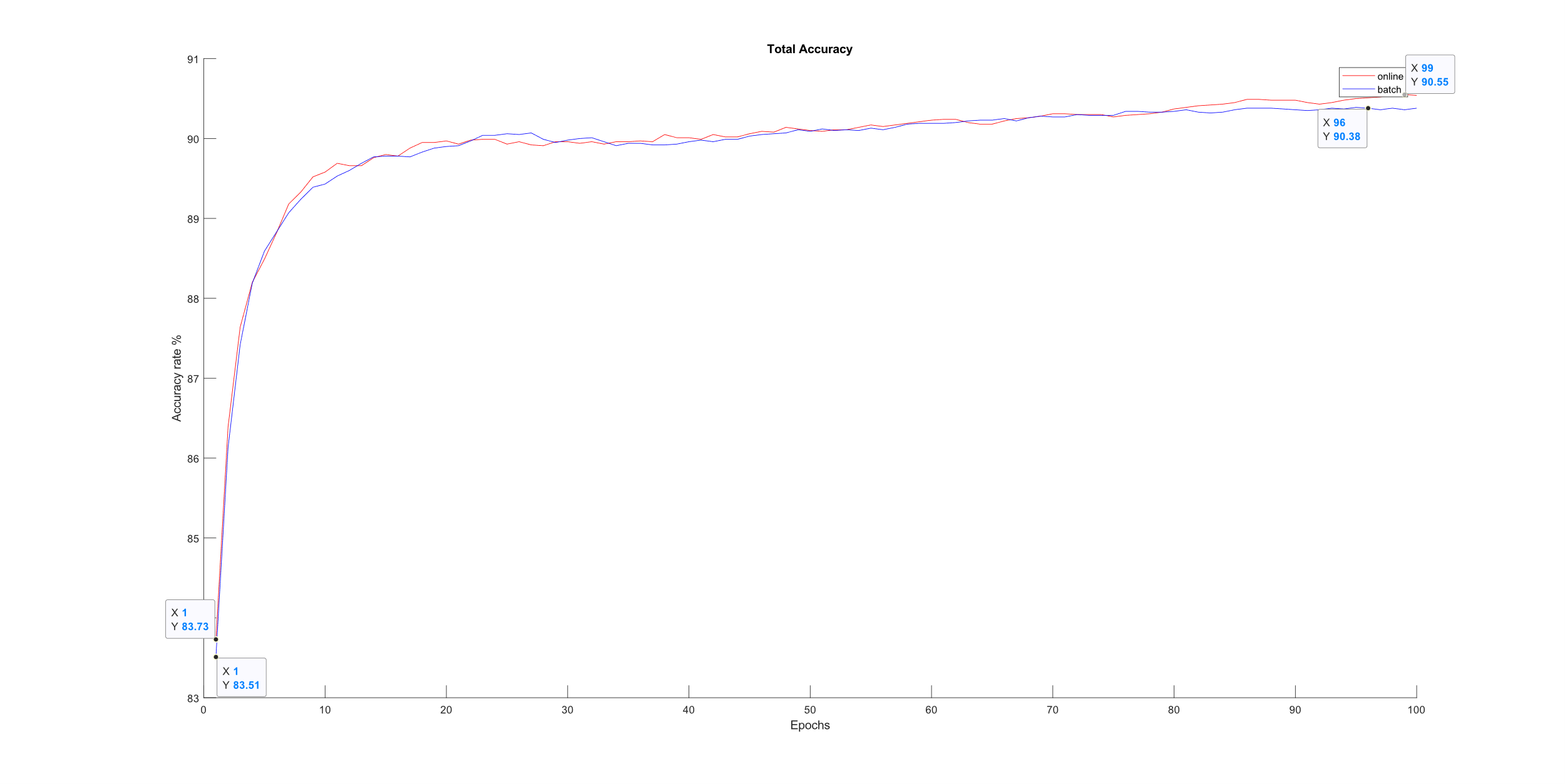


Figura 21- Combinazione: Accuratezza Migliore con eta aggiornato (tanH,ReLU,0.001,500), Misurazione Accuratezza

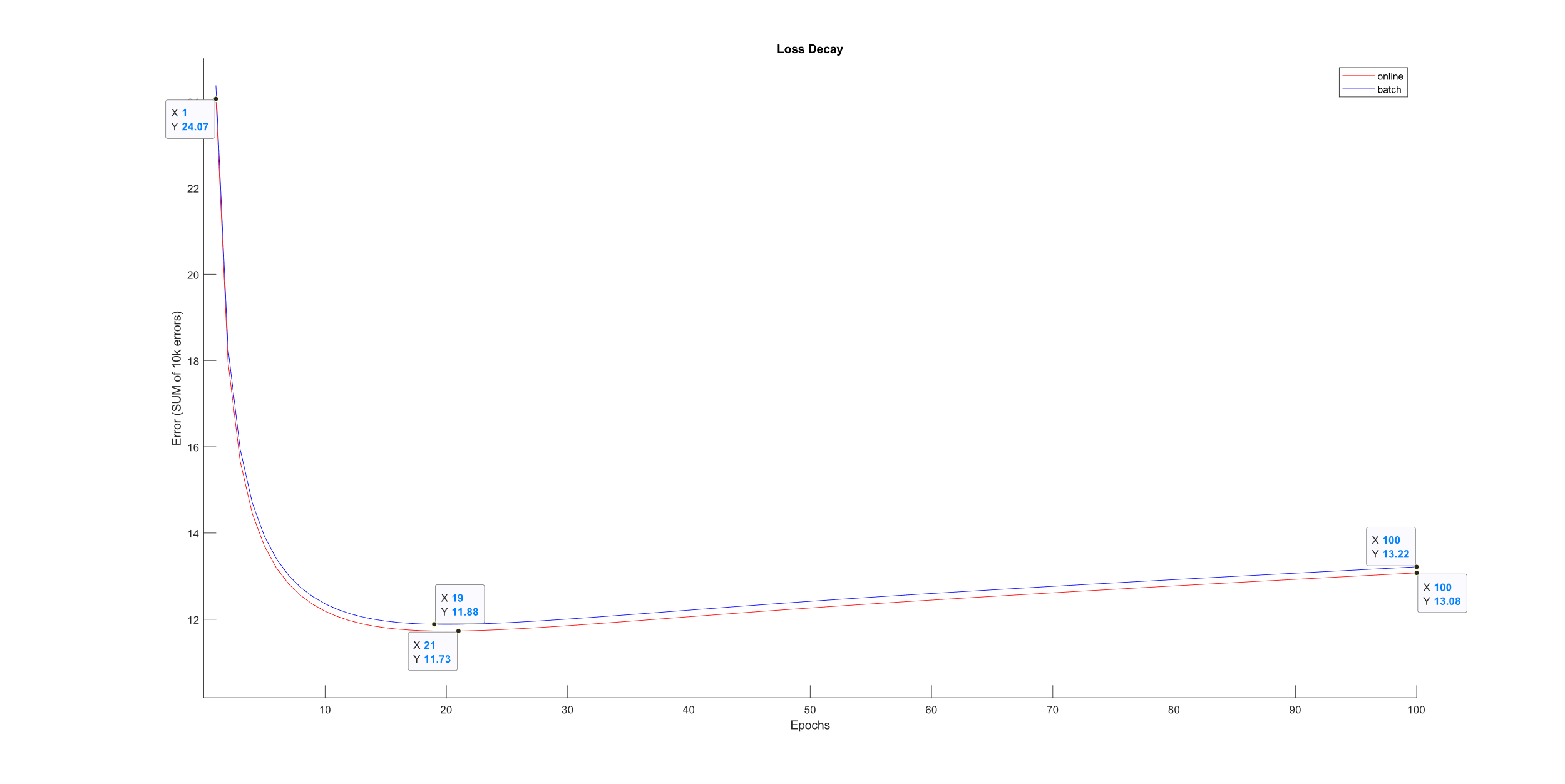


Figura 22 - Combinazione: Accuratezza Migliore con eta aggiornato (tanH,ReLU,0.001,500), Misurazione Errore Totale

Il tempo d’esecuzione è ammontato a 868 (14 minuti circa) secondi per l’apprendimento online, e 65 secondi per quello minibatch. Tale raddoppio rispetto ai tempi raccolti rispetto alla combinazione precedente sono ovviamente da imputare al corrispondente raddoppio del numero di nodi hidden impiegati.

## Combinazione: Migliori valori in media

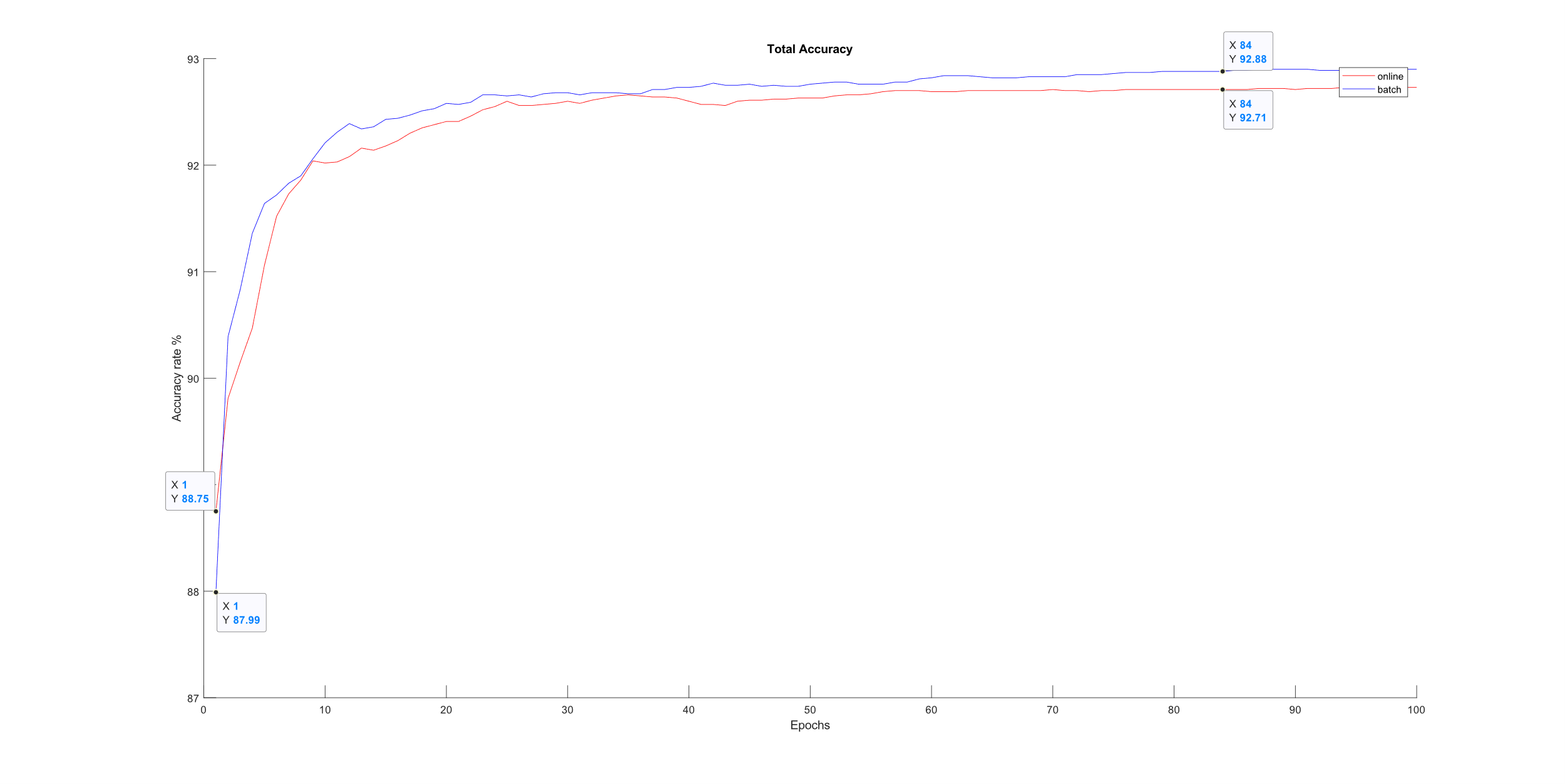


Figura 23 - Combinazione: Migliori valori in media (sigmoid, identity,0.05,250), Misurazione Accuratezza

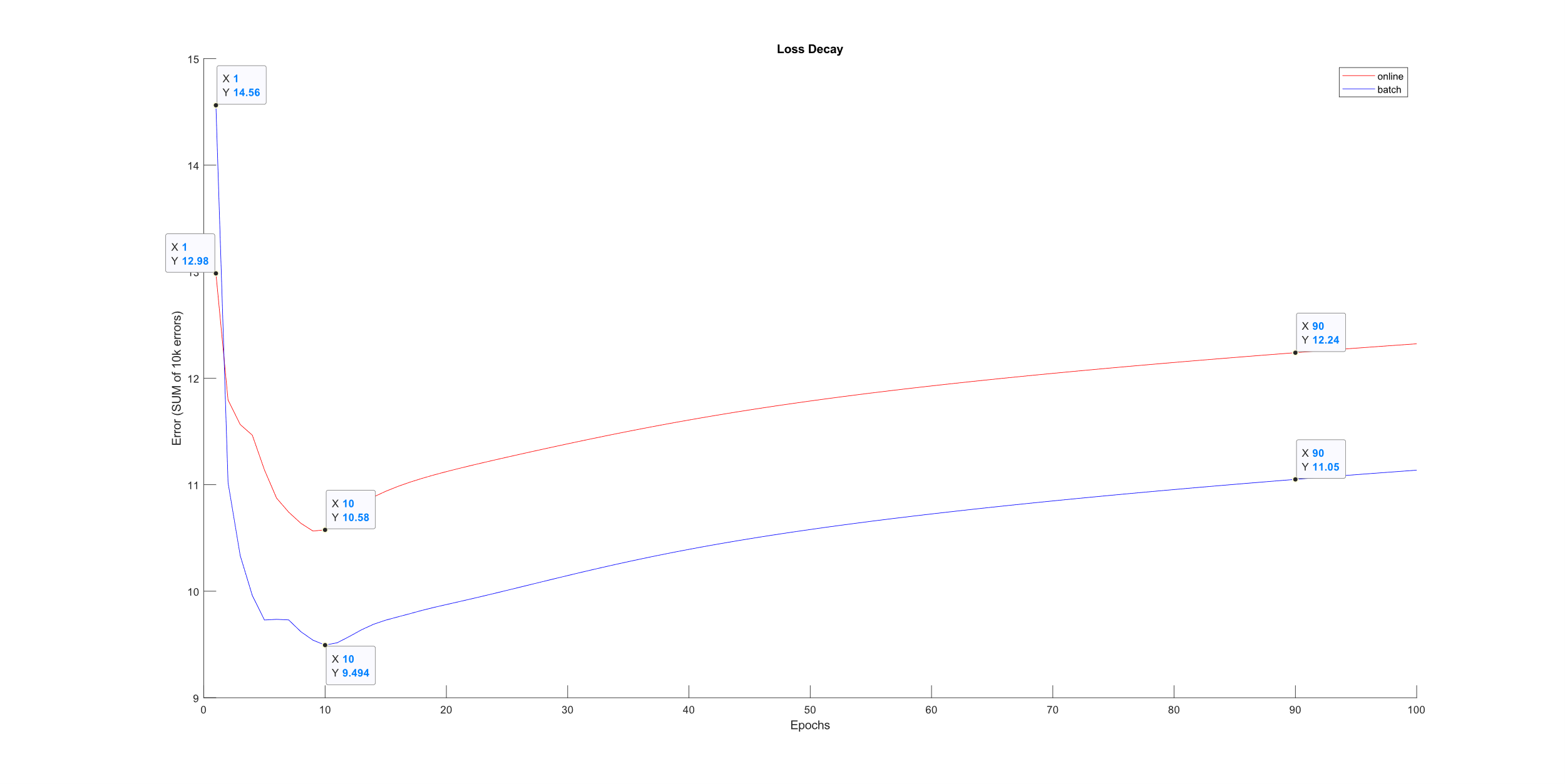


Figura 24 - Combinazione: Migliori valori in media (sigmoid, identity,0.05,250), Misurazione Errore Totale

La combinazione sopra mostrata è stata ottenuta scegliendo dalle due “*classifiche”* ordinate per migliore accuratezza e miglior errore, la combinazione che manteneva una posizione più alta in entrambe. Tale combinazione presenta valori di accuratezza previsti, con una percentuale di accuracy più alta alla prima epoca per l’online rispetto alla minibatch e trascurabilmente più bassa all’aumentare delle epoche. Comportamento simile è riscontrabile in Figura 24 per ciò che concerne la deviazione standard dell’errore, con però risultati che diventano presto favorevoli alla modalità minibatch. Il tempo di esecuzione è risultato essere di 490 secondi per la modalità online e di 35 per quella minibatch.

# Conclusioni

Dopo aver eseguito il K-folding in modalità online su un data set ristretto ed aver analizzato i dati restituiti in output, non è stata trovata un’unica combinazione che restituisse un valore ottimo in termini di accuracy e minimizzazione dell’errore di classificazione. Per meglio comprendere il rapporto che lega accuratezza e grandezza dell’errore, si è deciso di estrarre tre combinazioni ottimali dai dati del k-folding, una rappresentante il miglior dato di accuratezza, una il minor quantitativo di errore, ed una rappresentante la più equilibrata in termini dei due fattori di valutazione. Sono quindi stati effettuati i test di valutazione su un training set più esteso (3200 esempi), utilizzando per ognuno sia la modalità di apprendimento online che quella minibatch (utilizzando un batch size di 32 elementi) e valutando l’andamento di accuratezza ed errore su un totale di 100 epoche. Se le combinazioni “errore minimo” e quella bilanciata hanno mostrato comportamenti prevedibili, senza risentire dell’aumento di elementi su cui addestrare la rete, la combinazione “migliore accuratezza” ha evidenziato come all’aumentare del numero di elementi, specie in corrispondenza di un numero di hidden nodes maggiori, possa rivelarsi necessario impiegare un learning rate più piccolo per evitare comportamenti errati. Test non mostrati in questo documento hanno poi evidenziato la grande importanza del learning rate come iperparametro, ed il suo ruolo chiave nel corretto addestramento della rete, essendo questo il parametro in grado di separare una rete funzionante da una non funzionante, rispetto alla mera percentuale di accuratezza ottenibile.

Il confronto fra modalità di apprendimento online e batch ha mostrato come l’utilizzo della modalità online produca un miglioramento in termine di accuratezza di meno del 2% una volta raggiunto un numero di epoche considerevole (es. 50 epoche), e di circa 4% alla prima epoca. Se tale scarto di precisione risulta essere trascurabile nell’ambito dell’utilizzo della rete neurale, il risultato ottenuto dal minibatch learning diventa di molto più appetibile, considerando come le 100 epoche di addestramento siano state calcolate in un tempo di esecuzione 14 volte più veloce rispetto alla modalità online.

In conclusione, la combinazione di iperparametri più affidabile in termini di accuracy è risultata essere quella originariamente indicata come “errore minimo”, avendo prodotto un valore di accuratezza massima di 93.25% alla 86esima epoca. Risultati migliori sono certamente auspicabili addestrando la rete sull’intero dataset di diecimila esempi.

# Appendice

Seguono estratti dal codice sorgente degni di menzione. Vengono omesse per brevità porzioni di codice non interessanti.

## Kfold.m

Script utilizzato per la valutazione del k-folding. Il cuore dello script è rappresentato dai 4 cicli for innestati che variano tra le combinazioni di iperparametri e ruotano tra le k “fette” del training/validation set.

%load the training set

% 60000 (examples) x 784 (features)

train\_im = loadMNISTImages('train-images.idx3-ubyte')';

%load the training set labels

% 60000 x 10

train\_lb = loadMNISTLabels('train-labels.idx1-ubyte');

train\_lb = train\_lb';

train\_lb(train\_lb==0) = 10;

train\_lb = dummyvar(train\_lb);

%load the test set

% 784 x 10000

test\_im = loadMNISTImages('t10k-images.idx3-ubyte')';

% 10000 x 10

test\_lb = loadMNISTLabels('t10k-labels.idx1-ubyte');

test\_lb = test\_lb';

test\_lb(test\_lb==0) = 10;

test\_lb = dummyvar(test\_lb);

% Dimension of total training+validation set for k-fold

ts\_size = 3000;

% Number of folds

k = 10;

%compute slice size for k-folding

slice\_size = int32(ts\_size / k);

resized\_im = train\_im(1:ts\_size, :);

resized\_lb = train\_lb(1:ts\_size, :);

% Fixed hyperparams

errorDerivative = @crossEntropyDerivative;

errorFnc = @crossEntropy;

% Hyperparameters to test

netFnc = {{@tanH, @identity}, {@sigmoid, @identity}, ...

{@tanH, @ReLU},{@sigmoid, @ReLU}};

netNodes = [250, 500, 800];

netEtas = [0.1, 0.05, 0.01, 0.001];

fprintf("Hidden function; Output function; Eta; Hidden nodes; Mean Accuracy; C.E. Standard deviation\n");

bestErr = Inf;

elapsedTime = 0;

tic

%for each hyper param:

for fnc = 1: length(netFnc)

%used to store current error values for plotting

deviationsNodes = cell(length(netNodes),1);

nodeCounter=0;

for node = netNodes

%used to store current error values for plotting

nodeCounter=nodeCounter+1;

deviationsEtas=cell(length(netEtas),1);

etaCounter = 0;

for eta = netEtas

etaCounter=etaCounter+1;

%store current error and accuracy for the k slice

k\_error = zeros(k, 1);

k\_accuracy = zeros(k, 1);

%rotate the leave one out k validation part

for i = 0: k-1

% Calculate the slice index for testing and

% validation part

start\_idx = slice\_size \* i + 1;

stop\_idx = start\_idx + slice\_size - 1;

%cut the submatrix for the training set

k\_train\_im = [resized\_im(1:start\_idx-1, :); resized\_im(stop\_idx+1:ts\_size, :)];

k\_train\_lb = [resized\_lb(1:start\_idx-1, :); resized\_lb(stop\_idx+1:ts\_size, :)];

%cut the submatrix for the validation set

k\_test\_im = resized\_im(start\_idx:stop\_idx, :);

k\_test\_lb = resized\_lb(start\_idx:stop\_idx, :);

%train the network with current params

net = neuralNet(784, [node, 10], netFnc{fnc}, errorDerivative);

net = train(net, k\_train\_im, k\_train\_lb, eta, size(k\_train\_im, 1), 1);

%correctly predicted examples

guessed = 0;

%value storing computed error

currError = 0;

%test on the validation part

[~, z] = forwardPropagation(net, k\_test\_im);

%compute guessed examples and errore

for n = 1: size(z{1,2}, 1)

[val, idx] = max(z{1,2}(n,:));

if( idx == find( k\_test\_lb(n, :) ) )

guessed = guessed + 1;

end

currError = currError + sum(errorFnc(z{1,2}(n,:),k\_test\_lb(n, :)));

end

%store current error for compute average between k rotations later

k\_error(i+1) = currError;

k\_accuracy(i+1) = guessed / size(k\_test\_im, 1) \* 100;

end

%compute mean between al the k rotations

mean\_accuracy = sum(k\_accuracy) / k;

%compute a standard deviation value for the k-fold with these

%hyperparams

mu = sum(k\_error) / k;

variance = sum((k\_error - mu).^2) / k;

deviation = sqrt(variance);

fnc1 = func2str(netFnc{fnc}{1});

fnc2 = func2str(netFnc{fnc}{2});

str = sprintf("%s; %s; %.4f; %d; %.2f; %.2f\n", fnc1, fnc2, eta, node, mean\_accuracy, deviation);

fprintf("%s", str);

deviationsEtas{etaCounter} = deviation;

if (deviation < bestErr)

bestResult = str;

bestErr = deviation;

end

end

deviationsNodes{nodeCounter}=deviationsEtas;

end

elapsedTime = elapsedTime + toc;

%plot the current functions

plotBar(fnc1, fnc2, netNodes, netEtas, deviationsNodes);

. . .

## Singolo Training (Mnist.m)

. . .

%activation function for hidden layer

hiddenFnc = @sigmoid;

%activation function for output layer

outputFcn = @identity;

errorDerivative = @crossEntropyDerivative;

errorFnc = @crossEntropy;

TsSize = 3200;

batchSize = 3200;

eta = 0.05;

epochNumber = 100;

hiddenNodes = 250;

% Create neural network

net = neuralNet(784, [hiddenNodes, 10], {hiddenFnc, outputFcn}, errorDerivative);

%store error and accuracy computed after each epoch on all test set

errorsOnline = zeros(epochNumber, 1);

errorsBatch = zeros(epochNumber, 1);

accuracyOnline = zeros(epochNumber, 1);

accuracyBatch = zeros(epochNumber, 1);

%start training for each epoch (online learning)

tic

fprintf('Online Training\n');

for epoch = 1: epochNumber

%train the network with a batchSize of 1 (online)

net = train(net, train\_im, train\_lb, eta, TsSize, 1);

% Test neural net

correct = 0;

[~, out] = forwardPropagation(net, test\_im, @softmax);

%compute correctly guessed exemples

for i = 1: size(out{1,2}, 1)

[~, idx] = max(out{1,2}(i,:));

if( idx == find( test\_lb(i, :) ) )

correct = correct + 1;

end

end

%store current error standard deviation(for plot purpose)evaluated on

%each test case

errorsOnline(epoch) = calculateError(out{1,2}, test\_lb, errorFnc);

%store accuracy for this epoch

accuracyOnline(epoch)=(correct/size(test\_im, 1))\*100;

fprintf('epoch: %3d; accuracy: %3.2f%%; error: %3f\n', epoch,accuracyOnline(epoch) ,errorsOnline(epoch));

end

toc

tic

%start training for each epoch (miniBatch)

fprintf('MiniBatch Training\n');

net = neuralNet(784, [hiddenNodes, 10], {hiddenFnc, outputFcn}, errorDerivative);

for epoch = 1: epochNumber

net = train(net, train\_im, train\_lb, eta, TsSize, batchSize);

% Test neural net

correct = 0;

[~, out] = forwardPropagation(net, test\_im, @softmax);

%compute correctly guessed exemples

for i = 1: size(out{1,2}, 1)

[~, idx] = max(out{1,2}(i,:));

if( idx == find( test\_lb(i, :) ) )

correct = correct + 1;

end

end

%store current error standard deviation(for plot purpose)evaluated on

%each test case

errorsBatch(epoch) = calculateError(out{1,2}, test\_lb, errorFnc);

%store accuracy for this epoch

accuracyBatch(epoch) = (correct/size(test\_im, 1))\*100;

fprintf('epoch: %3d; accuracy: %3.2f%%; error: %3f\n', epoch,accuracyBatch(epoch) ,errorsBatch(epoch));

end

## calculateError.m



Funzione utilizzata per il calcolo della deviazione standard dell’errore a partire dall’output della forward propagation (usata dopo i test sul test set).

## PlotBar.m



Funzione utilizzata per creare i grafici del k-folding.