Advanced Machine Learning - Assignment 1

Luca Gandolfi, Matricola 807485

Ottobre 2019

1 Introduzione

Il dataset contiene informazioni riguardanti i clienti di una carta di credito in Taiwan nel 2015. Lo scopo del progetto consiste nel costruire un modello di deep learning in grado di predire l'attributo binario riguardante il *default payment*. In questo report verranno discusse e motivate le scelte di costruzione del modello di deep learning utilizzato.

2 Data Processing

Dal dataset sono state separate inizialmente le feature dal target, che da ora in avanti verranno chiamati rispettivamente *train* e *labels*. Sul train set è stata, quindi, applicata una standardizzazione su tutti gli attributi, per permettere un migliore apprendimento. In particolare, è stata utilizzata la funzione *Standard-Scaler* della libreria *sklearn*, che ha permesso di standardizzare ogni attributo individualmente. Questo vuol dire che ogni attributo avrà distribuzione con media = zero e deviazione standard = 1. Lo scaler adattato sul train set è stato poi applicato anche sul test set, per mantenere omogeneità di scala.

Non è stata applicata subito una divisione tra train set e validation set, la quale verrà svolta in modo automatico durante la *Fit* del modello, come discusso nel relativo paragrafo, con una divisione 70-30: 70% per il train e 30% per il validation set.

3 Costruzione della FeedForward Network

E' stato scelto di costruire una rete *fully-connected* e sequenziale.

3.1 Numero di layer e neuroni per layer

Durante la costruzione della rete è stato deciso di utilizzare 2 hidden layer + 1 output layer. Questa scelta è basata esclusivamente sulle performance registrate. Infatti, mantenendo lo stesso numero di neuroni ed effettuando un confronto tra la scelta di 1, 2 e 3 hidden layer, pur in valori molto bassi ma 2 layer forniscono

una accuracy maggiore sul validation set. I risultati di questo confronto sono mostrati nella seguente tabella.

N°$HiddenLayer$	Val Loss	Val Accuracy
1	0.4223	0.8279
2	0.4146	0.8341
3	0.4123	0.8321

Table 1: Confronto risultati del modello con numero di layer diversi.

Data la stocasticità del modello, è bene precisare che i valori ottenuti sono solo indicativi. Infatti, ad ogni esecuzione, si potrebbero ottenere risultati leggermente diversi. In ogni caso, dopo alcune esecuzioni, la media dei risultati rispecchia i valori mostrati in tabella.

Per quanto riguarda il numero di neuroni, è stato scelto di mantenere nei due livelli interni lo stesso numero di neuroni pari al numero di feature del train set. In particolare, non sono state notate significative differenze aumentando il numero di neuroni, sia nel primo, nel secondo o in contemporanea su entrambi gli hidden layer, anzi, mantenendo 23 e 23 sono stati ottenuti risultati, in media, leggermente migliori per la accuracy del validation set. Si è scelto, quindi, di mantenere relativamente basso il numero di neuroni.

3.2 Funzioni di attivazione

Nei due hidden layer è stata utilizzata la Rectified linear Unit - ReLU activation function, mentre nel output layer la Sigmoid function. Il motivo dietro la scelta della ReLU riguarda principalmente il suo ampio uso nel deeplearning e per i buoni risultati ottenuti rispetto ad altre funzioni di attivazione. In particolare, la ReLU permette una buona velocità di apprendimento andando a non considerare neuroni con pesi troppo bassi.

Durante la fase di scelta, è stata testata anche la funzione tangente *Tanh* in coppia con la *Hinge* Loss Function (dopo un opportuno reshape del target in -1, 1), ottenendo risultati non molto soddisfacenti. Per quanto riguarda l'output layer, la scelta della Sigmoide è stata quasi obbligata, in quanto è ottima per classificazione binaria.

In fase di scelta dei parametri, è stato valutato anche l'utilizzo di un one-hot-vector per la classe target, utilizzando la funzione *to_categorical* di *keras.utils*. Questa scelta ha portato, ovviamente, a modifiche nel output layer, utilizzando una Softmax function (adatta alla classificazione multi-classe) e 2 neuroni di output, uno per classe. Inoltre, è stato necessario utilizzare una *categorical_crossentropy* invece della corrispondente versione binaria, come verrà spiegato nel seguente paragrafo. In questo caso, mantenendo gli altri parametri invariati, i risultati emersi si sono rivelati molto simili a quelli ottenuti senza binarizzare il target. Come conseguenza, è stato deciso di non trattare questo problema di classificazione come un generico problema multi-classe ma di considerarlo come un problema binario con una Sigmoide e un solo neurone di

output.

3.3 Optimizer e funzione di Loss

Come optimizer è stato utilizzato Adam - Adaptive Moment Estimation. La scelta è ricaduta su questo optimizer dopo numerosi tentativi svolti con il fine di individuare il migliore per questo dataset. Adam ha il vantaggio di richiedere in genere poca memoria rispetto ad altri optimizer e sfrutta il Momentum per velocizzare la discesa lungo il gradiente. In particolare, Adam è stato implementato con i parametri di default, quindi con un learning rate pari a 0,001.

Per quanto riguarda la funzione di loss, come già anticipato precedentemente, è stata utilizzata la *binary-crossentropy*, una versione particolare della crossentropy per problemi di classificazione binaria, con valori nell'insieme 0,1. Questa funzione è la scelta di default per problemi binari e va utilizzata insieme alla Sigmoid function e ad un solo neurone di output.

Nel paragrafo precedente è stato descritto come sia stata testata anche la Hinge Loss function in coppia con la funzione di attivazione Tanh. Per svolgere questa prova è stato necessario applicare un reshape alle etichette e portarli da 0,1 a -1,1. I risultati ottenuti hanno mostrato come entrambi i valori di accuracy e di loss tendevano a stabilizzarsi molto presto e a rimanere costanti nel corso delle epochs. Tuttavia, non è mai stato superato lo 0,80 di accuracy e i valori di loss non hanno mai mostrato significative differenze con l'altro metodo in analisi.

3.4 Scelte in fase di Fit

Dopo aver costruito la struttura della rete è necessario definire i parametri della fase di apprendimento. Le scelte più delicate in questa fase riguardano il numero di epochs e la dimensione di batch. Per un dataset come quello utilizzato non è necessario svolgere una fase di apprendimento per un periodo prolungato, infatti, già dopo 10/15 epochs i valori di performance tendono a stabilizzarsi. I risultati sono stati controllati anche con un monitor *early stopping* definito sulla validation loss, il quale è stato poi rimosso nella release finale del codice per permettere l'esecuzione dello stesso numero di epochs ogni volta. Con queste considerazioni, quindi, è stato scelto di allenare il modello con 20 epochs, un numero abbastanza alto da vedere i risultati stabilizzarsi e, al contempo, abbastanza basso da prevenire un overfit sui dati di train.

Come dimensione del batch è stato scelto 256, in quanto è abbastanza ampio da considerare un buon numero di record e abbastanza piccolo da permettere diversi aggiustamenti dei pesi durante le varie epochs. In ogni caso, non sono state notate differenze di performance scegliendo una dimensione inferiore.

In questa fase è stata inoltre specificata la divisione tra train e validation set, in particolare tramite la funzione *validation_split* è stato assegnato il 30% del train come validation.

Una scelta importante ha riguardato anche il non utilizzo dei pesi per bilanciare il dataset durante l'apprendimento. Come è possibile osservare in Figure

1, il dataset di train presenta uno sbilanciamento importante verso la classe 0 e questo ha portato alla considerazione di utilizzare la funzione *class_weight* nella Fit per cercare di bilanciare l'apprendimento.

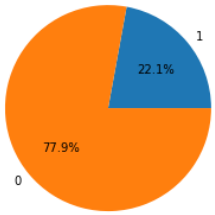


Figure 1: Grafico a torta relativo alla distribuzione delle classi del train set.

I risultati ottenuti utilizzando i pesi sono stati confrontati nel seguente paragrafo. Riassumendo, non sono stati utilizzati pesi nella release finale del codice per favorire un'accuratezza più alta.

4 Analisi dei risultati

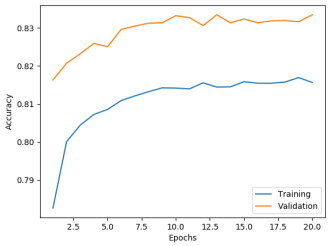
Sono state analizzate le misure di loss e di accuracy tra il modello allenato con i pesi e il modello allenato senza pesi. I pesi sono stati calcolati utilizzando la funzione *compute_class_weight* di *sklearn*. Come è possibile notare nei grafici, l'accuracy ottenuta utilizzando i pesi è più bassa rispetto a quella senza pesi, mentre, viceversa, la loss ottenuta utilizzando i pesi è più alta rispetto a quella senza pesi. E' necessario tenere presente dello sbilanciamento importante del quale non vi si tiene conto nel modello senza pesi, infatti, un alto livello di accuracy potrebbe indicare che il modello ha imparato a classificare bene la classe di maggioranza ma non è in grado di discriminare la classe di minoranza. Tuttavia, considerando anche la loss, è stato scelto di utilizzare il modello senza pesi per predire la classe dei record nel test set.

Tipo NN	Val Loss	Val Accuracy
Pesi	0.5478	0.7744
Senza pesi	0.4146	0.8341

Table 2: Performance dopo 20 epochs

E' bene considerare inoltre, che il modello con i pesi tendeva ad andare in overfit molto più velocemente rispetto all'altro modello. In alcune esecuzioni, questa problematica è emersa già dopo 20 epochs.

Figure 2: Accuracy



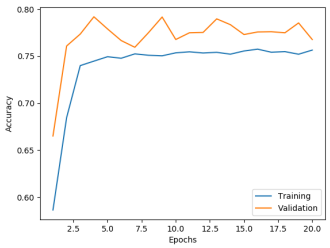
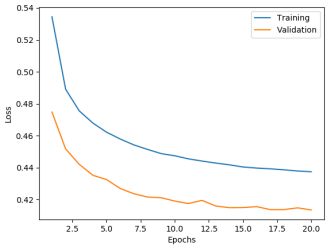


Figure 3: Accuracy nel modello pesato

Figure 4: Loss



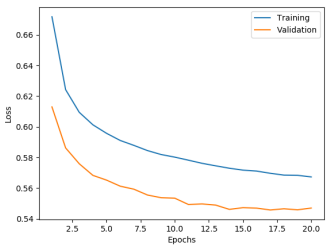


Figure 5: Loss nel modello pesato

5 Migliorie del modello

5.1 PCA e risultati

Per provare a migliorare le performance del modello è stato sperimentato un approccio diverso: un allenamento su un nuovo dataset di train ottenuto applicando Principal Component Analysis al train. Mantenendo il 95% della varianza del dataset originale, PCA ha estratto 15 nuovi attributi. La rete è stata quindi costruita in modo analogo a prima, utilizzando come numero di neuroni nei due hidden layer lo stesso numero di feature, 15. I risultati non hanno registrato migliorie rispetto alla soluzione proposta, per questo motivo è stato scelto di tenere commentata questa parte nel codice.

5.2 Differenti metriche di performance

Per ottenere un modello più solido, potrebbe essere utile considerare anche altre misure di performance, come ad esempio Precision, Recall e F-Measure. Queste misure non sono state prese in considerazione per scegliere il modello.