Lez8_GuidaIperparametri

November 8, 2023

1 Lez 8 Guida iperparametri, Keras e TensorFlow.

2 NNs per Regressione - Linee guida

Non esistono regole che valgono sempre. Linee guida per problema di regressione tramite rete neurale. Abbiamo le seguenti scelte da prendere:

- *Quanti neuroni*? 1 per ogni feature che ho in input (se input fatto da coordinate di un punto in spazio tridimensionale avrò 3 features, una per ogni coordinata, dipende dal caso in analisi e dal risultato della feature engineering). Le reti neurali spesso permettono di risparmiare il feature engineering, questo non toglie che se vedo operazioni di feature engineering banali applicabili queste potrebbero migliorare le prestazioni della rete.
- *Quanti livelli nascosti*? Non si ha risposta univoca, in generale tra 1 e 5 sono sufficienti per la maggior parte dei problemi.
- *Quante unità per livello nascosto*? Range ragionevole tra 10 e 100, correlato comunque alle features in input (se ho 1000 features in input un primo livello di 10 potrebbe essere limitante).
- *Quante unità di output*? Dipende da quanti valori devo predire.
- *Quale funzione di attivazione utilizzare nel livello nascosto?* ReLU (ad oggi scelta standard per iniziare, si possono poi provare le altre anche a seconda del problema).
- *Quale livello di uscita*? Nessuna funzione di attivatione, oppure ReLU per output positici, oppure tanh per output che appartengono ad un certo intervallo.
- *Quale funzione di Loss*? MSE (errore quadratico medio) o MAE (errore medio assoluto), dipende dal problema il primo pesa di più i casi in cui sbaglio di tanto.

Graficamente:

Iperparametro	Possibili/Valori tipici
# neuroni di input	1 per caratteristica di input
# hidden layers	?? (1-5 per molte attività)
# unità nascoste per layer	?? (10-100 per la maggior parte delle attività)
# unità di output	1 per dimensione prevista
Attivazione nascosta	ReLU
Attivazione di output	Nessuna; ReLU per output positivi; tanh per output limitati
Loss	MSE o MA

Per definire gli *hidden layers* e *hidden units per layer* si va a tentativi, questo viene chiamato **hyperparameter optimization**.

2.1 Problema Classificazione - Linee guida

- *Quanti neuroni*?

  1 per ogni feature che ho in input (se input fatto da coordinate di un punto in spazio tridimensionale avrò 3 features una per ogni coordinata, dipende dal caso in analisi e dal risultato della feature engineering). Le reti neurali spesso permettono di risparmiare il feature engineering, questo non toglie che se vedo operazioni di feature engineering banali applicabili queste potrebbero migliorare le prestazioni della rete.

- *Quanti livelli nascosti*?

  Non si ha risposta univoca, in generale tra 1 e 5 succienti per la maggior parte dei problemi.

- *Quante unità per livello nascosto*?

  Range ragionevole tra 10 e 100, correlato comunque alle features in input (se ho 1000 features in input un primo livello di 10 potrebbe essere limitante).

- *Quante unità di output*?

  1 per classe, 1 se binario.

- *Quale funzione di attivazione utilizzare nel livello nascosto?*

  ReLU (ad oggi scelta standard per iniziare, si possono poi provare le altre anche a seconda del problema).

- *Livello di uscita*?

  Sigmoid per classificazione binaria poiché l'output è compreso tra 0 e 1 corretto perché ci dà una probabilità, Softmax per multi-classe (output vettore di valori che sommano a 1).

- *Loss*? Cross-Entropy loss.

Graficamente, riassumiamo con:

Iperparametro	Binary	Multiclass
Input e hidden layers	Come regressione	Come regressione
# unità di output	1	1 per classe
Output activation	Sigmoid	Softmax
Loss	Cross entropy	Cross entropy

2.1.1 Cross-Entropy loss

- Con 2 classi, abbiamo **BinaryCrossEntropy**:

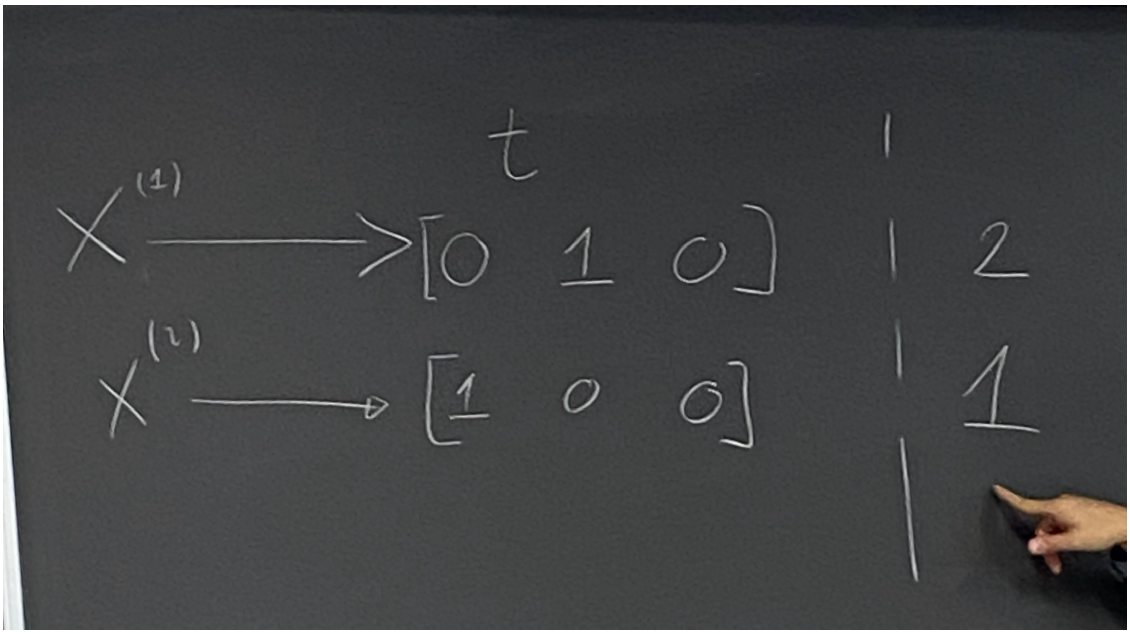
  $L = -(tlogy + (1-t)log(1-y))$; dove y è la mia probabilità, t è 0 o 1, solo uno dei due termini sarà diverso da zero. Più mi allontano, più il logaritmo diventa piccolo, ma mettendo il meno davanti, allora stiamo dicendo che tale valore cresce.

- con $C > 2$ classi l'espressione diventa: $L = \sum_{i=0}^{C} t_C log y_C$; solo un termine verrà diverso da 0.

In Keras tutto ciò è già implementato:

- **Classificazione binaria**: *BinaryCrossentropy*.
- **Caso del multi-classe**: Si hanno due possibilità a seconda di come sono fatti i vettori target:
  - *Codifica di tipo one-hot*: Vettore con tante componenti quante sono le classi, si avranno tanti 0 tranne un 1 che sta nella posizione corrispondente alla classe a cui appartiene. Usiamo *CategoricalCrossentropy*.
  - *Scalari compresi tra 1 e il numero di classi*: più intuitiva ma meno adatta per una algoritmo. Parliamo di *SparseCategoricalCrossentropy*.

In foto vediamo questi due sotto casi, nell'ordine elencato.



Keras deve sapere se ha a che fare con la prima o la seconda opzione.

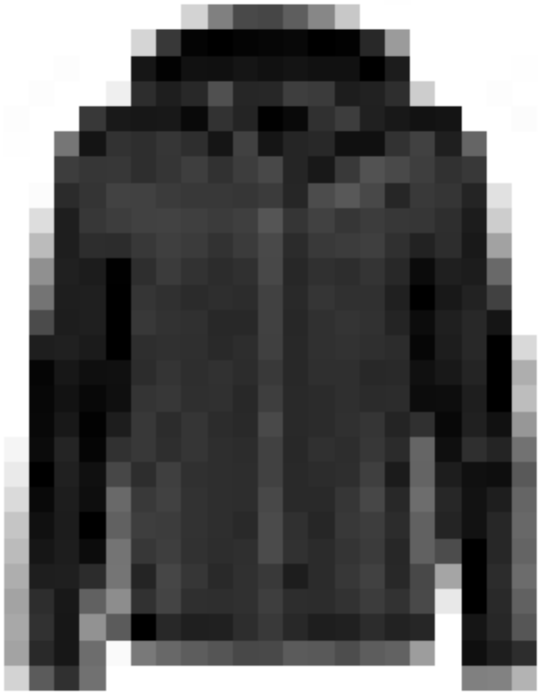
Esempio MNIST Dataset Fashion MNIST (oggetti di abbigliamento), versione più complicata da predire rispetto alle lettere (60000 immagine 28x28 con 10 classi). *Obiettivo*: dalle immagini in input si vuole capire di quale oggetto si sta parlando.

Nota: nel notebook originale, bisogna installare le seguenti dipendenze, per vedere alcuni grafici: `pip install pydot` `pip install graphviz`

Per risolverlo si ha il notebook:

[8]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.␣
 ↪#prendo 5000 per il validation, le rimanenti per il training set. In più␣
 ↪scalo tutti i valori per 255
#stessa normalizzazione anche per x_test
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.

#X_train_full.shape #ci dice come è fatto il dataset (#oggetti, dim, dim)
#X_train_full.dtype #vedo che sono tutti interi tra 0 e 255 mi danno intensità
plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```



Per vedere se tutto sta funzionando la posso graficare. In questo caso per ciascun elemento del training set mi viene detta la classe di appartenenza tra 0 e 9, chi mi dà il dataset mi deve dare la corrispondenza tra numero di classe e classe effettiva. Inserisco questa informazione in un array (class_names).

[10]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# Posso vedere anche quante istanze ho nel testset e nel validarion set.
```

```
# Poi c'è del codice che ci fa vedere come sono fatte altri tipi di istanze nel␣
 ↪training set.
```

[11]:
```
n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
plt.show()
```

Problema di Classificazione in merito a questo dataset E' facile porsi queste domande:

- *Voglio usare rete neurale*? Si/No. Nel nostro caso sì.
- *Quante unità metto nell'input layer*? Potrebbe essere un'idea 28x28 unità, potrei anche appiattire l'input creando un vettore. Tensorflow lavora con tensori quindi non c'è problema lavorare con input 28x28.
- *Quanti livelli intermedi*? Forse 3?
- *Quante unità per livello*? Scegliamo, senza criterio specifico: 50 primo, 30 secondo, 10 nell'ultimo.
- *Quanti neuroni nel livello di uscita e che funzione*? 10 con softmax (posso vedere l'ultimo livello nascosto e quello di uscita come un unico livello in cui prima applico la funzione di attivazione e poi la sigmax).

- *Funzione di loss*? Usiamo la cross-entropy loss.

Era proposta una rete con (300-100 e un livello di uscita con 10 unità), è più semplice inserire un livello Flatter per modificare l'input in modo da linearizzarlo (pre-processamento).

[12]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

[13]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

[14]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

[15]:
```
def build_model (n_hidden=3, n_units=100, learning_rate=0.01):
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=[28, 28]))
    for i in range(n_hidden):
        model.add(keras.layers.Dense(n_units, activation="relu"))
    model.add(keras.layers.Dense(10, activation="softmax"))

    model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(lr=learning_rate),
              metrics=["accuracy"])
    return model
```

[ ]:
```
from scipy.stats import reciprocal
import sklearn
from sklearn.model_selection import RandomizedSearchCV
```

```
param_distribs = {
    "n_hidden": [0, 1, 2, 3],
    "n_units": np.arange(1, 100).tolist(),
    "learning_rate": reciprocal(3e-4, 3e-2).rvs(1000).tolist(),
}

# wrapper
keras_clf = keras.wrappers.scikit_learn.KerasClassifier(build_model)

rnd_search_cv = RandomizedSearchCV(keras_clf, param_distribs, n_iter=7, cv=2,␣
 ↪verbose=2) # toy params
rnd_search_cv.fit(X_train, y_train, epochs=45,
                  validation_data=(X_valid, y_valid),
                  callbacks=[keras.callbacks.EarlyStopping(patience=5)])

rnd_search_cv.best_params_
```

[ ]:
```
model = rnd_search_cv.best_estimator_.model
model.evaluate(X_test, y_test)
```

Definiti i modelli con `model.summary()` posso vedere il numero di parametri. Il nostro modello ha molti meno parametri.

Esistono funzioni di Keras per graficare il modello, può essere utile per verificare di aver capito quello che si sta facendo (fa vedere per ogni livello le dimensioni, la prima sarà None perché non abbiamo specificato la dimensione dei mini-batch).

[ ]:
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

model.summary()
```

[ ]:
```
keras.utils.plot_model(model, "my_fashion_mnist_model.png", show_shapes=True)
```

Si utilizza poi compile indicando `loss`, metriche da osservare e algoritmo di ottimizzazione da usare invocando il nome della classe keras che fa questo lavoro (scorciatoia attraverso stringhe). Il compile serve per addestrare il modello.

Siamo pronti per il training, chiamiamo il modello passando dati di training, epoche, validation set (utile perché tra epoche calcola il valore di nostro interesse).

È possibile stampare anche lo storico, si nota che mentre la loss continua sempre a scendere andando avanti con le epoche, sul validation set sembra aumentare; si vede anche sull'accuratezza, da un certo punto in poi sul validation set diminuisce (questo fa pensare ad overfitting, il modello non sta più migliorando le prestazioni sul validation set).

`evaluate` permette di valutare le prestazioni sul test set. Non c'è overfitting perché il modello si comporta in modo simile su tutti i set.

Se voglio posso prendere istante dal test set e attraverso `model.predict` vedere la probabilità che l'oggetto appartenga ad una determinata classe.

Se faccio argmax scopro l'indice della classe predetta e utilizzandolo per accedere all'array dei nomi della classe scopro cosa sono.

Altre metriche possono essere ottenute tramite classifcation report di scikit learn a cui passiamo vettore predizioni fatte dal modello e il vettore delle etichette. Viene fuori una tabellina con i valori per ciascuna classe.

2.2 Salvataggio dei modelli

`model.save` e lo salva su un file, lo si fa quando si è soddisfatti del modello che si ha. Dopo addestrata.

Per ricaricare il modello `model.load`.

2.3 TensorBoard

Caricare i processi di addestramento sul browser, va data una cartella dove mettere i log, istanziare un nuovo modello

```
tensorboard --logdir=/path/alla/cartella --port=porta_in_ascolto
```

Altrimenti se si sta utilizzando jupyter è possile avere la tensorboard in una cella.

Per farla funzionare bisogna istanziare una callback indicando la cartella dei log e tra i callback indicare questo oggetto.

Una volta avviata si vede l'accuratezza nelle varie epoche, aggiornando la pagina si aggiorna anche il grafico man mano che il training va avanti.

3 Tuning Hyperparameters

Reti neurali sono molto espressive, ma hanno il problema di avere molti iperparametri da dover impostare. Questo è un lavoro che fatto a tentativi non porta da nessuna parte. È necessario automatizzare questo processo. Una volta scelto il tipo di rete bisogna ottimizzare gli iperparametri. Non è possibile fare una prova esaustiva con tutti i parametri, quindi di solito si utilizza la ricerca randomizzata. È bene dire che ci sono librerie specializzate per risolvere questo problema perché si possono fare cose un po' furbe per non scegliere le combinazioni completamente a caso (tenendo conto delle prove fatte in precedenza).

3.1 Ottimizzazione degl iperparametri applicata al dataset precedente.

Collegato al precedente, sempre lo stesso problema, ma ottimizziamo gli iperparametri (prima parte uguale, si ha una nuova funzione *build model*). Questa funzione definita per comodità costruisce un modello Keras parametrico. Gli passo numero di livelli nascosti, numero di unità nel livello nascosto e learning rate. Questa funzione quindi costruisce semplicemente il modello ogni volta che ne voglio uno. Utilizziamo keras insieme a scikit learn.

Definiamo il range dei parametri da osservare, per poter usare keras insieme a scikit-learn dobbiamo usare un *wrapper* costruito sulla base di un modello Keras che offre tutte le funzionalità che scikit learn si aspetta ma con un modello Keras.

Alla fine di tutto il processo viene fuori la migliore combinazione.