Sviluppo Futuro Rete Neurale

Il seguente documento è una guida su come progettare la rete neurale per la classificazione di stream di oggetti (immagini, video, audio e file) di una connessione tra client e server di IM.

La rete neurale è una deep feed forward DFF per un problema di multiclass classification e si forniranno degli snipet di codice di implementazione utilizzando la libreria tensorflow keras.

Classi:

- Immagini – Tag “I”
- Video – Tag “V”
- Audio – Tag “A”
- File – Tag “F”

Data Preparation

1. **Import and split data**

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Utilizzando la libreria pandas è possibile leggere un file CSV che costituisce il dataset, il quale contiene le features di vari oggetti estrapolate tramite il tool IMSD Parser.

Una volta caricati il dataset princiaple, possiamo dividere le colonne in input (X) e output (y) per la modellazione.

- X – contiene le features di input ('DURATION_TIME', 'NUMB_OF_PACKETS', 'TOTAL_BYTE' , 'MIN_BYTE', 'MAX_BYTE' , 'AVG_BYTE' , 'CLI_NUMB_OF_PACKETS' , 'CLI_TOTAL_BYTE' , 'CLI_MIN_BYTE' , 'CLI_MAX_BYTE' , 'CLI_AVG_BYTE' , 'SVR_NUMB_OF_PACKETS' , 'SVR_TOTAL_BYTE' , 'SVR_MIN_BYTE' , 'SVR_MAX_BYTE' , 'SVR_AVG_BYTE')

- Y – contiene l’output (target), ovvero rappresentato dalla features ‘TAG’ (1° colonna)

```
dataset = pd.read_csv('DATASET_SENDER_MUL_FREQ.csv',
skipinitialspace=True,skiprows=1, names = COLUMNS, low_memory=False)
dataset.describe()
Y = dataset.iloc[:, :1]
X = dataset.iloc[:, 1:]
```

2. **Preprocessing**

Viene effettuata una trasformazione dei valori delle features in maniera tale da essere di tipo float.

```
X = X.astype('float64')
```

Poiché le features di input sono su scale diverse, è necessario standardizzare l'input.

```
sc = StandardScaler()
X = sc.fit_transform(X)
```

La rete neurale dovrà dare in output una classe (I, V, A, F). Per poter essere processati i dati devono essere dei numeri, per cui occorre effettuare una codifica dei valori della features TAG Per ciascuna classe viene assegnato un valore intero univoco compreso tra 0 e (numero classi - 1).

```
encoder = LabelEncoder()
Y = encoder.fit_transform(Y)
```

3. Create Train/Test set

Ora che il set di dati è pronto, possiamo dividerlo 80/20: 80% per il set di allenamento e il 20% percento per il set di test.

Si utilizza la funzione train_test_split: Il primo argomento è il dataframe di input mentre il secondo è il dataframe di output (target). È possibile specificare la dimensione del test set con test_size.

random_state è il seme utilizzato dal generatore di numeri casuali.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
random_state=0)
```

Building Neural Netowork

1. Define the model

La rete neurale è una deep feed forward composta da:

- Input Layer – 16 nodi (fetures)
- 2 Hidden Layer – 10 nodi
- Output Layer – 4 nodi (target, categoria di oggetti tag “I”, “V”, “A”, “F”)

Il modello utilizzato di Keras è Sequential per costruire la rete e Dense per costruire i layer.

Le funzioni di attivazione utilizzate sono:

- ReLu → per i livelli nascosti;
- Softmax → per il livello di output - prevedere la probabilità per ogni classe

kernel_initializer definisce il modo di impostare i pesi casuali iniziali dei livelli di Keras. Per una distribuzione uniforme, possiamo usare inizializzatori uniformi casuali.

```
model = keras.Sequential()
model.add(keras.layers.Dense(10, activation='relu',
kernel_initializer='random_normal', input_dim=16))
model.add(keras.layers.Dense(10, activation='relu',
kernel_initializer='random_normal'))
model.add(keras.layers.Dense(4, activation='softmax',
kernel_initializer='random_normal'))
```

2. **Compile the model**

Prima che il modello sia pronto per l'apprendimento, è necessaria qualche impostazione in più. Queste sono aggiunte durante i passi di *compilazione* del modello:

Funzione perdita —Misura quanto è accurato il modello durante l'apprendimento. La volontà è di minimizzare questa funzione per "dirigere" il modello nella giusta direzione.

Ottimizzatore —Indica com'è aggiornato il modello sulla base dei dati che tratta e della sua funzione perdita.

Metriche —Usate per monitorare i passi di addestramento e verifica.

Il modello è adatto usando stochastic gradient descent con un tasso di apprendimento predefinito di 0,01 e un momentum di 0,9.

Cross-Entropy è la funzione di perdita predefinita da utilizzare per problemi di classificazione multi-class.

La Cross-Entropy calcolerà uno score che sintetizza la media della differenza tra le distribuzioni di probabilità effettive e previste per tutte le classi del problema. Lo score è ridotto al minimo e un perfetto è 0.

La Cross-Entropy può essere specificata come funzione di perdita in Keras specificando "categorical_crossentropy" durante la compilazione del modello.

Usiamo l'accuratezza come metrica per misurare le prestazioni del modello.

```
opt = keras.optimizers.SGD(lr=0.01, momentum=0.9)
model.compile(loss='sparse_categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])
```

3. **Fit the model**

Esaminiamo oltre 100 epoche per addestrare il modello. Un'epoca è un'iterazione sull'intero set di dati.

```
history = model.fit(X_train, Y_train,
validation_data=(X_test, Y_test),
epochs=100, verbose=2)
```

4. Evaluate the model

Le metriche di valutazione del modello sono necessarie per quantificare le prestazioni del modello.

La precisione è una metrica di valutazione comune per i problemi di classificazione. È il numero di previsioni corrette fatte in rapporto a tutte le previsioni fatte.

Di seguito calcoliamo l'accuratezza in addestramento e l'accuratezza in test.

```
_, train_acc = model.evaluate(X_train, Y_train, verbose=2)
_, test_acc = model.evaluate(X_test, Y_test, verbose=2)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

5. Prediction

Una volta addestrato il modello, può essere usato per fare previsioni. Dopo la previsione, è necessario eseguire l'operazione inversa per recuperare il tag della classe associata.

Occorre definire un prediction set e un prediction target.

```
predictions = model.predict_classes(prediction_set)
prediction_ = np.argmax(keras.utils.to_categorical(predictions), axis = 1)
prediction_ = encoder.inverse_transform(prediction_)

for i, j in zip(prediction_ , predict_target):
    print( " the nn predict {}, and the species to find is {}".format(i,j))
```