Assignment 3

Marana Nicolò

November 22, 2018

Abstract

L'assignment 3 consiste nell'utilizzare le Convolutional Neural Network (CNN) sul dataset Mnist utilizzando un modello con meno di 10.000 parametri.

1 Lavoro Svolto

Di seguito vengono brevemente illustrate le scelte implementative adottate.

1.1 Library

Di seguito vengono illustrate le principali librerie utilizzate:

1. **Keras**: Permette di creare modelli di NN in modo semplice, funzionando da wrapper e basandosi su Tensorflow o Theano.
2. **Matplotlib**: Permette di creare dei plot dei dati in modo da analizzarne i risultati.

1.2 Cartella Codice

Vengono descritti ora i due file presenti all'interno della cartella codice:

1.2.1 reproduce.py

Questo file python serve ad impostare i seed dell'ambiente (os, random, numpy, tensorflow e keras) in modo da ottenere la riproducibilità degli esperimenti, evitando così di ottenere risultati differenti sugli stessi parametri.

1.2.2 main.py

Questo file python contiene il codice dell'intero assignment 3, di seguito viene brevemente descritto.

Fix Seed Viene da subito chiamata la funzione per fissare il seed con lo scopo di ottenere la riproducibilità dell'esperimento.

```
from reproduce import *
setup(seed_value=42)
```

Import data Sono stati scaricati i dati tramite la funzione mnist.load_data()

```
(x_train, y_train),(x_test, y_test) = mnist.load_data()

'''
Data Dimension Analysis

    x_train.shape = (60.000, 28, 28)
    y_train.shape = (60.000,)
    x_test.shape = (10.000, 28, 28)
    y_test.shape = (10.000,)

'''
```

Processing on X Viene effettuato un reshape sui dati x_train e y_train.

```
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1],
    x_train.shape[2], 1)

x_test = x_test.reshape(x_test.shape[0], x_test.shape[1],
    x_test.shape[2], 1)

'''
Data Dimension Analysis

    x_train.shape = (60.000, 28, 28, 1)
    x_test.shape = (10.000, 28, 28, 1)

'''
```

Processing on Y Viene applicata la funzione categorical sui dati y_train e y_test in modo da trasformali nella codifica one-hot encoding.

```
# in mnist dataset are present 10 different classes
num_classes = 10

# Apply conversion of y transforming in One-hot encoding
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

'''
Data Dimension Analysis

    y_train.shape = (60.000, 10)
    y_test.shape = (10.000, 10)

'''
```

Model Di seguito è illustrato il modello scelto:

```
# Structure
model = Sequential()

model.add(ZeroPadding2D((1,1),input_shape=(28,28,1)))
model.add(Conv2D(28, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(14, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(num_classes, activation='softmax'))

# Compile model
model.compile(loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adadelta(),
                  metrics=['accuracy'])
```

Number of parameters Viene utilizzata la funzione summary() per valutare il numero di parametri.

```
# Print model summary for see number of parameter in this case
    the constrain is < 10.000.

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_18 (ZeroPaddi (None, 30, 30, 1)         0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 28, 28, 28)        280
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 14, 14, 28)        0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 12, 12, 14)        3542
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 6, 6, 14)          0
_________________________________________________________________
flatten_15 (Flatten)         (None, 504)               0
_________________________________________________________________
dropout_21 (Dropout)         (None, 504)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                5050
=================================================================
Total params: 8,872
Trainable params: 8,872
Non-trainable params: 0
_________________________________________________________________
```

Fit Model Di seguito viene effettuato il training del modello.

```
# Define hyper-paramenter
batch = 256
epoch = 10

# Fit Model
history = model.fit(x_train, y_train, batch_size=batch, epochs=
    epoch, verbose=1, validation_data=(x_test, y_test))
```

Plot result Vengono plottati i risultati di accuracy e loss per ogni epoca.

```
# Plot accuracy on train e validation
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# Plot loss on train e validation
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

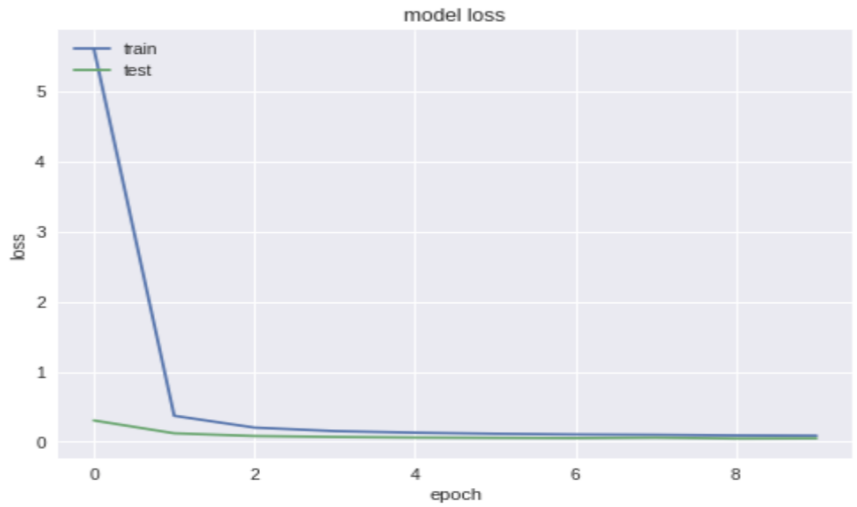
2 Risultati

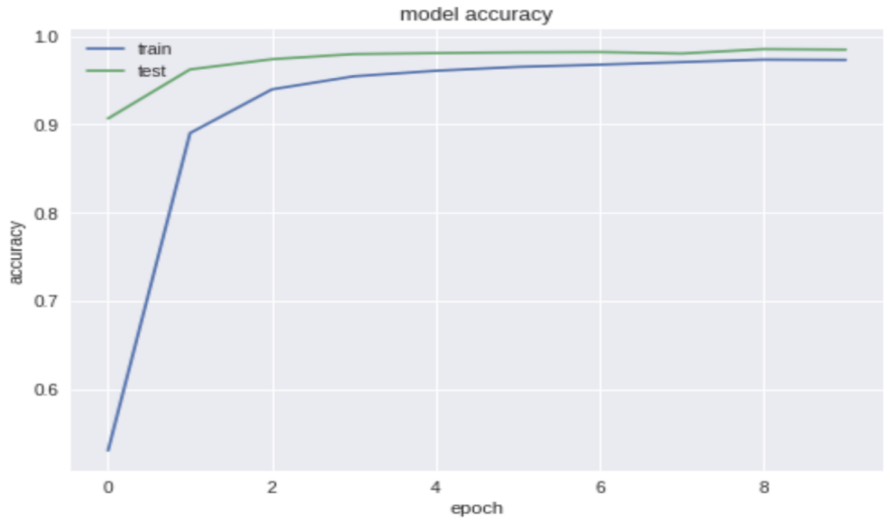
Sono stati effettuati alcuni esperimenti con diversi iperparametri:

- Batch: 32, 64, 128, 256
- Epochs: 5, 10
- Modificando la struttura del modello e provando a utilizzare diversi layer e parametri.

Il modello scelto ha utilizzato un batch pari a 256 ed è stato trainato per 10 epoche.

RESULT		
	Training	Testing
Loss	0.0862	0.0477
Accuracy	0.9730	0.9846


model loss
train
test
loss
epoch
0
1
2
3
4
5
0
2
4
6
8


model accuracy
train
test
accuracy
epoch
0.6
0.7
0.8
0.9
1.0
0
2
4
6
8