Predizione di serie storiche

Relazione di progetto IA e SSD

Michele Proverbio

Goal

Costruzione di un modello di machine learning per la predizione di serie storiche facendo uso di reti neurali ricorrenti (LSTM e GRU).

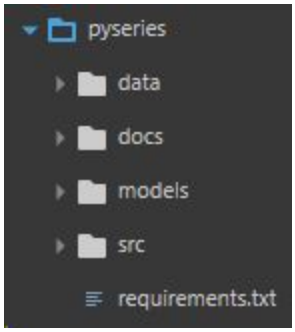
In particolare il modello deve riuscire a produrre proiezioni nel futuro e pulire dataset con valori mancanti.

Tecnologie

Il software è stato sviluppato in python (3.6). I modelli e gli algoritmi di machine learning si basano sulla libreria Tensorflow.

In particolare, per la costruzione delle reti neurali è stato usato il modulo Keras, che fornisce astrazioni di alto livello per layer, ottimizzatori e training.

Struttura del progetto



Il file *requirements.txt* raccoglie le dipendenze del progetto (vedi sezione Setup).

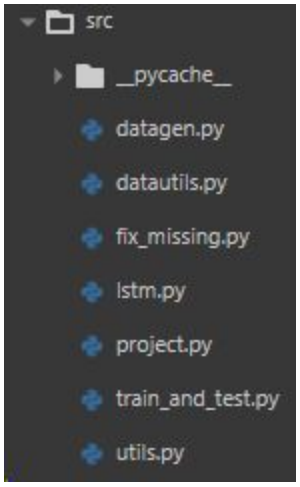
Sotto “data” ci sono vari dataset già convertiti in csv. Alcuni dataset sono stati modificati con valori mancanti per testare la funzionalità di data cleaning.

In “docs” vengono salvati documenti e output delle fasi di training e test.

“models” raccoglie i modelli pre-addestrati.

In “src” ci sono i file sorgenti python.

File sorgenti



- datagen.py - generatore di dataset.
- datautils.py - collezione di funzioni per la manipolazione di dataset.
- fix_missing.py - funzione di data cleaning. Carica un modello pre-addestrato e lo utilizza per intuire i valori mancanti del dataset.
- lstm.py - classe principale che astrae il modello di machine learning. Mette a disposizione metodi per fare training e proiezioni.
- project.py - funzione per fare predizioni (proiezioni) su un dataset dato un modello pre-addestrato.
- train_and_test.py - funzione per fare training di un nuovo modello.
- utils.py - collezione di funzioni per il plotting di dataset e altro.

Setup

1. Download del progetto: $ git clone https://gitlab.com/proch92/AIprojects.git
2. Installazione dipendenze: $ pip install -r requirements.txt
3. Test training: $ python3 src/train_and_test.py test_sin sin.csv
4. Test proiezione: $ python3 src/project.py test_sin sin.csv 30

Accenni al funzionamento delle RNN

Le reti neurali ricorsive “consumano” gli input un elemento alla volta costruendo iterativamente una memoria chiamata stato della cella. Ad ogni iterazione il prossimo elemento del dataset viene concatenato allo stato dell’iterazione precedente. Il tensore risultante diventa l’input della funzione di attivazione della cellula che restituirà in output il nuovo stato.

Le LSTM (Long Short-Term Memory) introducono un ulteriore stato che viene propagato durante l’esecuzione sdoppiando in questo modo i concetti di memoria e stato della cella.

Le varianti RNN offerte da keras (RNN, LSTM, GRU) possono essere intercambiate facilmente sostituendoli nella definizione del modello senza altri cambiamenti nel codice.

Modello della rete neurale

```
def model_definition(self, batch_size):
    return tf.keras.Sequential([
        tf.keras.layers.LSTM(
                120,
                dropout=0.2,
                recurrent_dropout=0.2,
                return_sequences = True,
                stateful = True,
                batch_input_shape = (batch_size, None, 1)),
        tf.keras.layers.LSTM(
                60,
                dropout=0.2,
                recurrent_dropout=0.2,
                return_sequences = True,
                stateful = True),
        tf.keras.layers.Dense(1)
    ])
```

Il modello usato nei test è una rete neurale multi livello con 2 layer LSTM e un layer DNN di output.

Di seguito la spiegazione dei parametri dei layer LSTM:

- Il primo argomento rappresenta il numero di celle LSTM nel layer. Un numero maggiore di celle aumenterà il potere espressivo del modello permettendogli di cogliere pattern più complessi, ma aumentando anche il rischio di overfitting e i tempi di esecuzione e convergenza.
- I parametri di dropout settano appunto il dropout dei pesi durante la fase di training. Il dropout è una tecnica che migliora l'errore di generalizzazione.
- "*return_sequences*" dice al layer ricorsivo di restituire i valori di output di tutte le iterazioni intermedie e non solo l'output dell'ultimo step.
- Il parametro "stateful = True" istruisce l'esecutore di tensorflow a salvare lo stato delle celle LSTM anche dopo la fine dell'esecuzione. Lo stato salvato verrà usato

nella prossima esecuzione del modello. Per resettare manualmente gli stati si può usare il metodo *reset_states()*.

Fase di training

La serie storica di training viene spezzettata in sottosequenze di lunghezza predefinita. Durante il training, per ogni sottosequenza viene svolta una completa fase di fitting del modello (loss, gradient descent, weights adjustment)

Lo stato delle celle viene mantenuto tra le fasi per permettere alla rete neurale di apprendere dipendenze funzionali che sono più distanti nel tempo rispetto alla finestra scelta. Finestre più grandi velocizzano la fase di training ma riducono la stabilità dell'apprendimento.

Dopo la preparazione dei dati il modello viene compilato in un grafo consistente per Tensorflow specificando il numero di batch, la funzione di loss e l'ottimizzatore di learning.

Di seguito la funzione di training del modello:

```
def train(self, modelname, trainset, epochs, batch_size, time_steps):
    (x, y) = self.prepare_data(trainset, batch_size, time_steps)

    model = self.model_definition(batch_size)
    model.compile(loss='mse',
            optimizer=tf.train.RMSPropOptimizer(0.001),
            metrics=['mse'])
    model.summary()

    model.reset_states()
    history = model.fit(
            x,
            y,
            epochs=epochs,
            batch_size=batch_size,
            shuffle = False,
            verbose=1)

    model.save(os.path.join('models', modelname + '.h5'))
```

Da notare il parametro "shuffle = False" che disabilita la funzione di shuffling dei mini batch pre-training. Vogliamo che i mini batch vengano eseguite in modo sequenziale data la

natura del nostro problema e dato che lo stato di memoria LSTM viene mantenuto durante tutto il processo.

Preparazione dei dati

La serie di training viene normalizzata a media-0 e stddev-1 e differenziata, ad ogni elemento viene sottratto il precedente.

Es. differenziazione

[1, 4, 2, 3, 0, -1, 3] -> [1, 3, -2, 1, -3, -1, 4]

Differenziare la serie aiuta la rete a cogliere macro pattern, come la crescita costante delle temperature medie globali in una serie con granularità giornaliera.

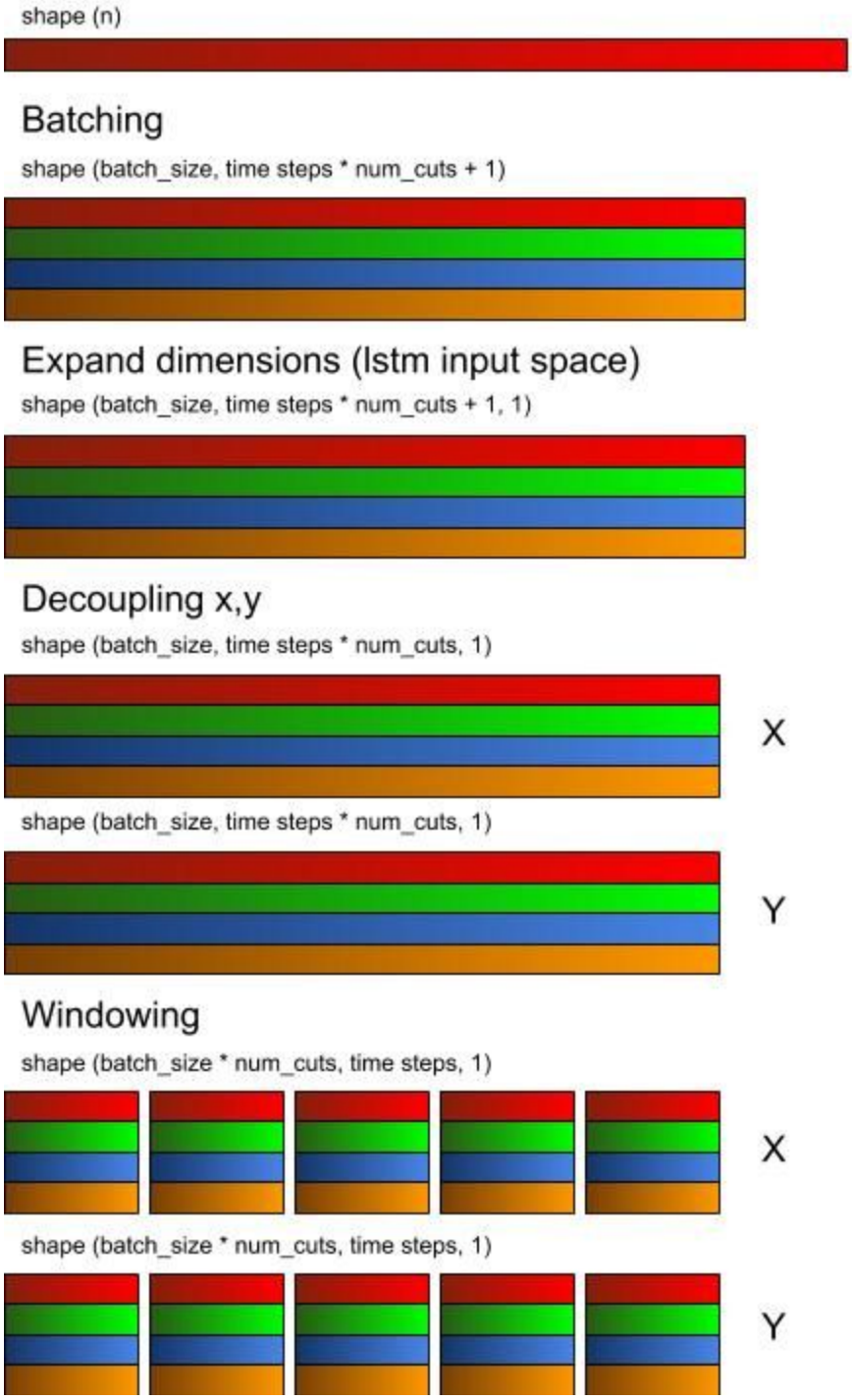
La serie normalizzata e differenziata viene poi duplicata e shiftata verso destra per creare il tensore di target.

[1,2,3,4,5,6] -> x = [1,2,3,4,5] y = [2,3,4,5,6]

Batch learning

Per velocizzare il training del modello la serie viene duplicata per poter creare dei batch. Ad ogni serie duplicata viene applicata una funzione di roll verso sinistra per renderle differenti l'una dall'altra senza però modificarne le dipendenze temporali; gli elementi reinseriti a destra dalla funzione di roll vengono quindi scartati.

Recap preparazione dei dati



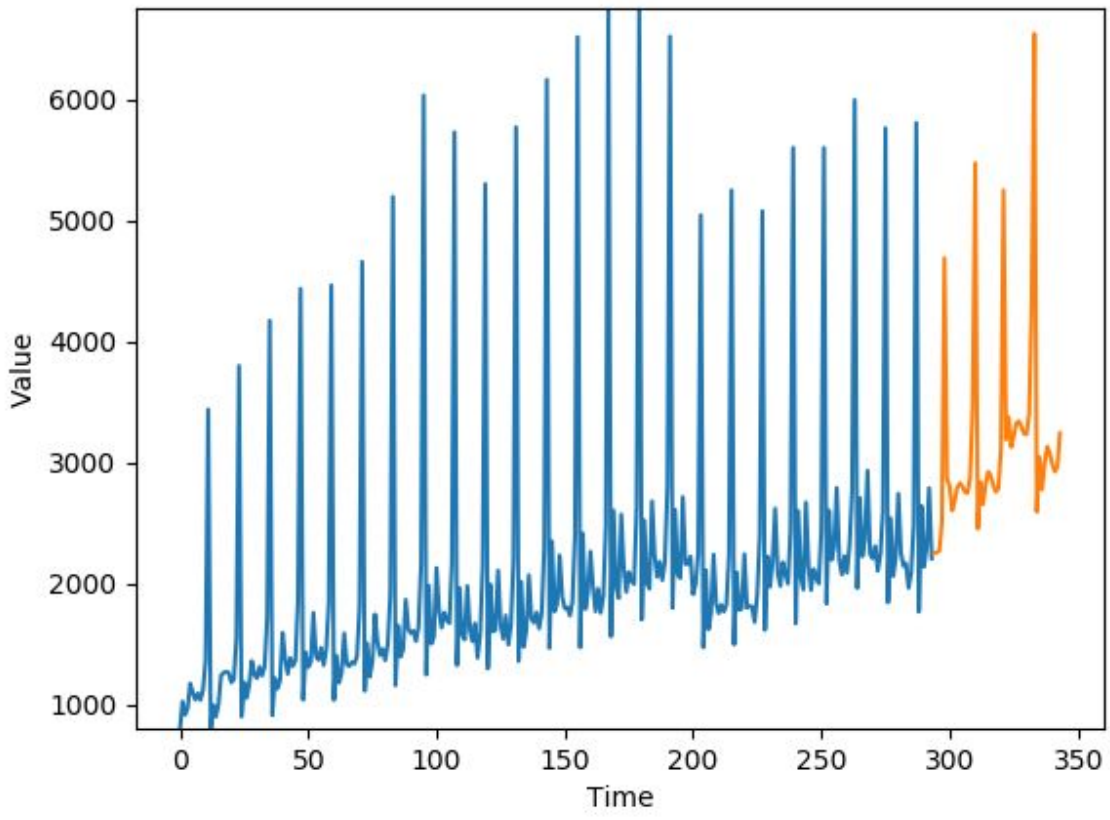
Evaluation su nuove sequenze. Proiezioni nel futuro di serie temporali

Dato un modello pre addestrato possiamo utilizzarlo per fare proiezioni future.

La classe LSTM mette a disposizione la funzione evaluate che presi in ingresso il modello pre addestrato, un test set e la lunghezza della proiezione restituisce la predizione.

Il funzionamento è abbastanza semplice: si fa consumare la serie di test al modello per produrre degli stati di memoria consistenti al contesto, successivamente si fanno predizioni, un elemento alla volta, in base all'elemento precedente. Si procede iterativamente in base alla lunghezza della proiezione desiderata.

Un esempio di proiezione sul dataset delle vendite delle gioiellerie:



Data cleaning. Riempimento di valori mancanti

La funzione di proiezione può essere usata anche per predire valori mancanti di un dataset.

Ad esempio, la seguente serie

[1,2,3,4,nan,nan,nan,nan,9,10,11,12,13,nan,nan,nan,nan,nan,nan,20,21,22,23]

può essere scomposta in oggetti di tipo (serie, lunghezza proiezione) da dare in pasto alla funzione di proiezione spiegata nella sezione precedente, ottenendo:

([1,2,3,4], 4) ([9,10,11,12,13], 6)

Inserendo le proiezioni all'interno della serie originale otteniamo una serie completa senza valori mancanti. Sotto, un esempio di esecuzione; in blu la serie reale, in arancione quella predetta dal modello.