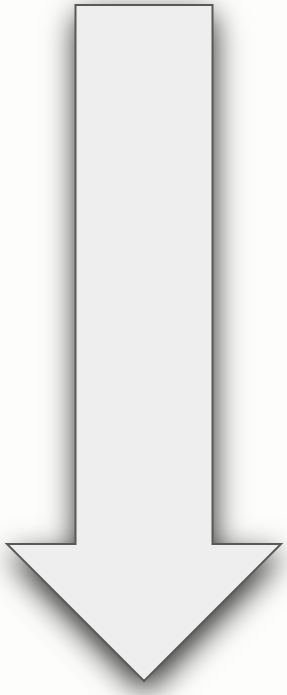
UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Next-Activity Prediction tramite Reti Convoluzionali Deformabili

Relatore:
Prof. Nicola Di Mauro

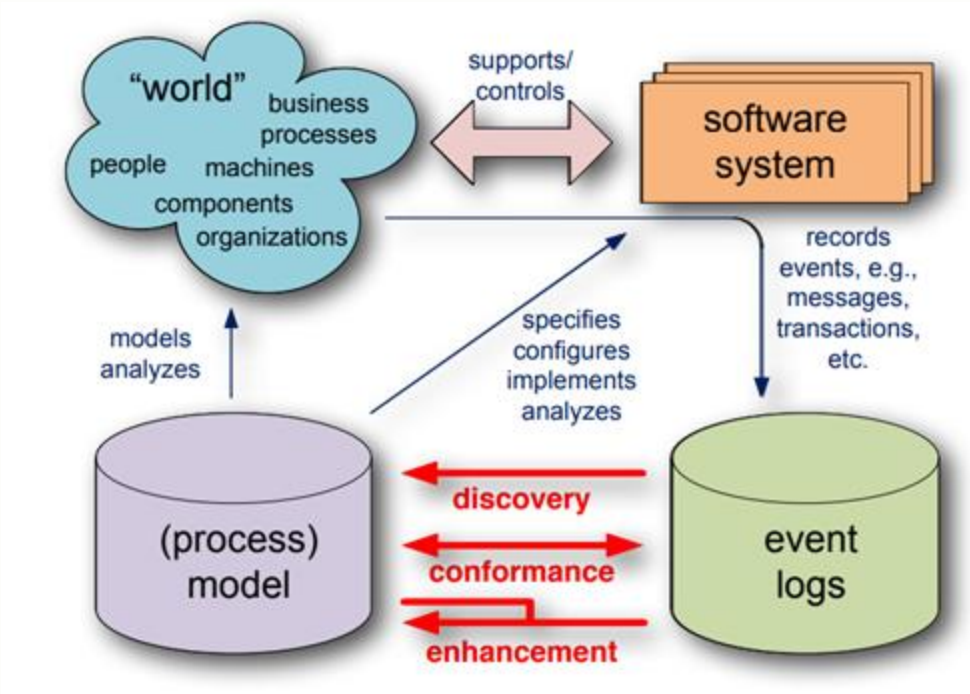
LACAM
Laboratorio per l'Acquisizione della Conoscenza e l'Apprendimento nelle Macchine

Candidato:
Antonio Matteo Carulli
Mat. 655891



> Introduzione
> Next-Activity Prediction
> Stato dell'arte
> Architetture progettate
> Implementazione
> Valutazione Sperimentale
> Conclusioni

Introduzione - Process Mining



Tipi di Process Mining da "Process Mining Manifesto"

It's a **data-driven age**!

I *sistemi informativi aziendali* registrano **event log** sulle attività interne dell'azienda

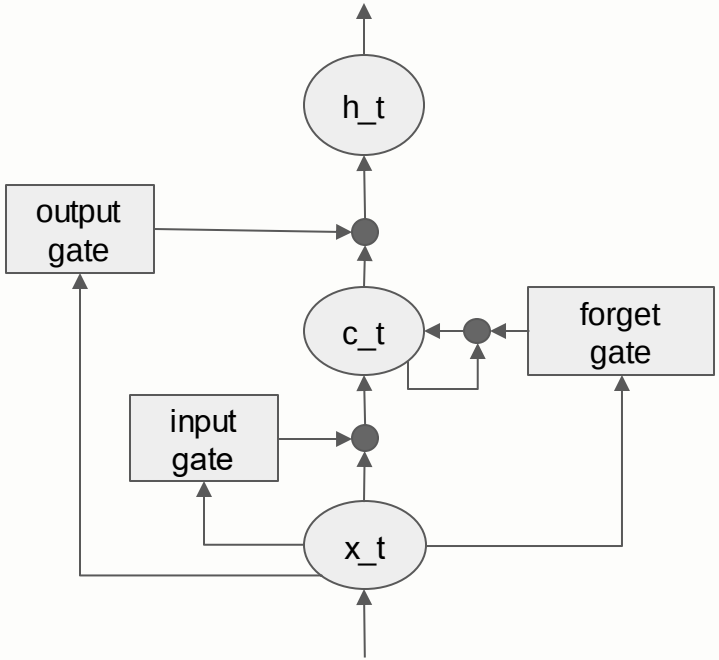
Con tecniche di **Process Mining** da questi registri si formulano dei **modelli** sulla base dei quali *migliorare* e *confrontare* un *processo aziendale*

Event Log e Next-Activity Prediction

Case ID	Event	Timestamp	Activity	Actor	Cost	
3	35654521	30-12-2010:14.32	register request	Pete	50	…
	35654522	30-12-2010:15.06	examine casually	Mike	400	…
	35654524	30-12-2010:16.34	check ticket	Ellen	100	…
	35654525	06-01-2011:09.18	decide	Sara	200	…
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	…
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	…
	35654530	08-01-2011:11.43	check ticket	Pete	100	…
	35654531	09-01-2011:09.55	decide	Sara	200	…
	35654533	15-01-2011:10.45	pay compensation	Ellen	200	…
4	35654641	06-01-2011:15.02	register request	Pete	50	…
	35654643	07-01-2011:12.06	check ticket	Mike	100	…
	35654644	08-01-2011:14.43	examine thoroughly	Sean	400	…
	35654645	09-01-2011:12.02	decide	Sara	200	…
	35654647	12-01-2011:15.44	reject request	Ellen	200	…

Esempio di event log tratto da “Process Mining Manifesto”

Stato dell'arte - LSTM



LSTM Cell

L'approccio più diffuso per elaborare dati sequenziali si basa su reti neurali ricorrenti LSTM

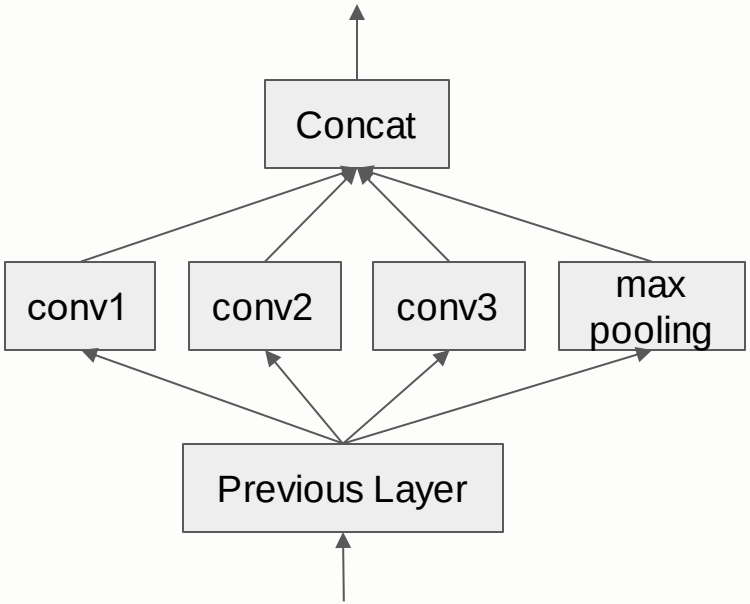
PRO:

- Osservazione di lunghe dipendenze temporali tra i dati

CONTRO:

- Alto costo computazionale
- Grafi di elaborazioni molto profondi

Stato dell'arte - Inception Networks



Naïve Inception Module
[Szegedy et al. 2014] [Di Mauro et al., 2019]

Anche le reti convoluzionali possono gestire dati sequenziali

Il modulo a inception unisce più convoluzioni dal kernel diverso per "allargare" il campo recettivo della rete

I risultati raggiunti in recenti articoli hanno dimostrato la superiorità di questo metodo rispetto alle reti LSTM

Architetture Progettate

In questo lavoro di tesi si è voluta portare avanti la ricerca sulle reti convoluzionali mettendo a confronto tre approcci diversi:

- una rete convoluzionale tradizionale
- un metodo mutuato dalla computer vision, *Deformable ConvNets*
- una convoluzione il cui kernel rimane fisso, ma deformato da una maschera binaria casuale, che abbiamo chiamato *MaskedConv*

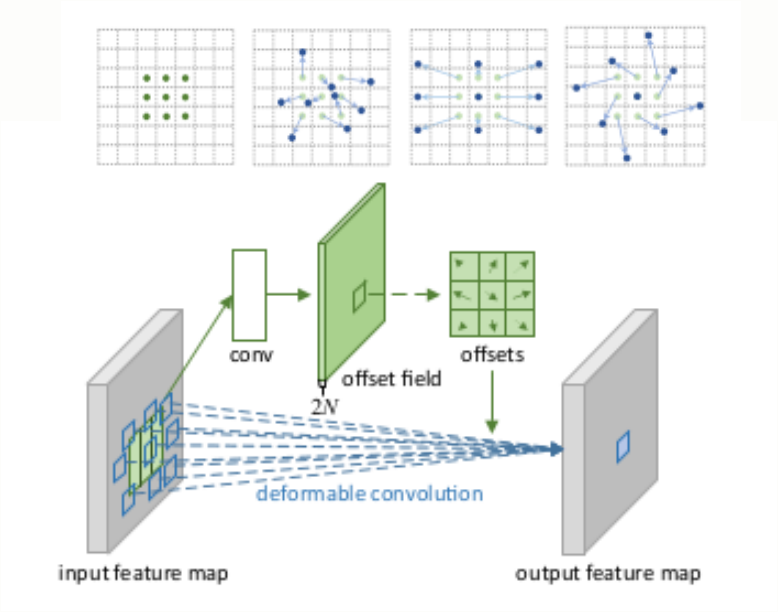
Implementazione

Tutte le reti sono state implementate in Python3, sfruttando la libreria open source di deep learning Keras e il backend TensorFlow 2.0

Per la scelta degli iperparametri, si è invece utilizzata la libreria HyperOPT di hyperparameter optimization

Infine l'effettiva elaborazione dei dataset è avvenuta sulla piattaforma online Google Colab

Deformable Convolutional Network



[Dai et Al., Deformable Convolutional Networks, 2017]

Il campo recettivo della rete viene dilatato tramite l'applicazione di offset appresi dal layer precedente e interpolati con le posizioni di input

L'assunzione è che effettuando un'osservazione più completa, la rete possa notare dipendenze tra i dati anche a lunga distanza temporale

Implementazione - DeformableConv1D

```python
# Deformable 1D Convolution
class DeformableConv1D(tf.keras.layers.Layer):
    def __init__(self, filters, kernel_size):
        super(DeformableConv1D, self).__init__()
        self.filters = filters
        self.kernel_size = kernel_size
        self.R = tf.constant(self.regularGrid(self.kernel_size),
                             tf.float32)

    def build(self, input_shape):
        W_shape = (self.kernel_size, 1)
        self.W = self.add_weight(
            name='W',
            shape=W_shape,
            trainable=True,
            dtype=self.dtype)
        super(DeformableConv1D, self).build(input_shape)

    def call(self, x):
        offconv = Conv1D(x.shape[-1]*2,
                         self.kernel_size,
                         padding='same',
                         activation='relu',
                         trainable=True)
        offset = offconv(x)
        y = self.linearInterpolation(x, offset)
        y = tf.reduce_sum(self.W * y, [0])
        y = tf.reshape(y, [-1, x.shape[1], x.shape[2]])
        return y
```

Ogni layer in Keras si basa su tre metodi:

- **__init__()** per l'inizializzazione degli attributi della classe
- **build()** per la definizione dei pesi
- **call()** per la definizione del forward pass

La nostra classe DeformableConv1D aggiunge:

- **regularGrid()** per la definizione della tassellatura regolare di campionamento
- **linearInterpolation()** per la preparazione degli attributi all'operazione di interpolazione
- **g()** per la logica del kernel dell'operazione di convoluzione

Implementazione - DeformableConv1D

```python
def linearInterpolation(self, x, offset):
    # input locations
    Q = tf.where(tf.equal(K.flatten(x), K.flatten(x)))
    Q = tf.cast(Q, tf.float32)

    offset = offset - x
    offset = K.flatten(offset)

    # offset locations
    P = Q + offset

    # regulard grid sampling
    ylist = []
    for pn in tf.unstack(self.R):
      G = self.g(Q, P+pn)
      ylist.append(G * K.flatten(x))

    return tf.stack(ylist)
```

```python
    def g(self, q, p):
        g = tf.subtract(tf.squeeze(q),
                        tf.squeeze(p))
        g = tf.abs(g)
        g = tf.subtract(1.0, g)
        return tf.maximum(0.0, g)
```

Il metodo linearInterpolation():

- calcola gli spostamenti
- applica l'operazione di interpolazione

Il metodo g() definisce il kernel dell'operazione di convoluzione tramite interpolazione lineare tra gli spostamenti calcolati e l'input

Formalmente, l'operazione finale è

$$\boldsymbol{y}(\boldsymbol{p}_0) = \sum_{\boldsymbol{p}_n \in R} \boldsymbol{w}(\boldsymbol{p}_n) \cdot \boldsymbol{x}(\boldsymbol{p}_0 + \boldsymbol{p}_n + \Delta \boldsymbol{p}_n)$$ con

$$\boldsymbol{x}(\boldsymbol{p}) = \sum_{\boldsymbol{q}} G(\boldsymbol{q}, \boldsymbol{p}) \cdot \boldsymbol{x}(\boldsymbol{q})$$ dove $\boldsymbol{p} = \boldsymbol{p}_0 + \boldsymbol{p}_n + \Delta \boldsymbol{p}_n$

e $g(a, b) = max(0, 1 - | a - b |)$

Implementazione - MaskedConv1D

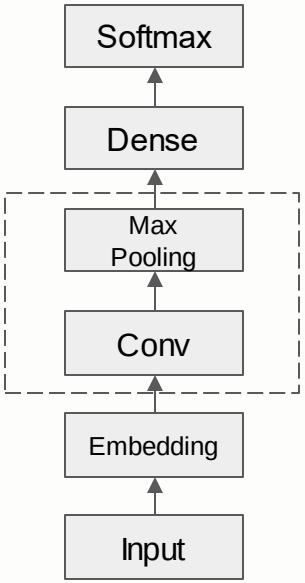
```
class MaskedConv1D(tf.keras.layers.Conv1D):
    def __init__(self, filters, kernel_size, **kwargs):
        super(MaskedConv1D, self).__init__(filters,
                                           kernel_size,
                                           trainable=False,
                                           **kwargs)

    def call(self, x):
        # random boolean mask
        mask = np.random.randint(2,
                                 size=(self.kernel_size[0], 1, 1))
        self.kernel = self.kernel * mask
        return super(MaskedConv1D, self).call(x)
```

La classe MaskedConv1D eredita direttamente dalla classe Conv1D

Il metodo build() è chiamato automaticamente subito prima dell'esecuzione del corpo del metodo call(), definendo il parametro kernel

Il metodo call() definisce e applica la maschera binaria casuale al kernel ed esegue la convoluzione sull'input

Valutazione Sperimentale



Esempio di rete a un modulo

I log adottati sono i seguenti:

- **Receipt phase**: contiene i record dell'esecuzione della fase di ricevuta del processo di applicazione del permesso di costruzione in un anonimo comune olandese.
- **helpdesk** : contiene gli eventi di un processo di biglietteria del reparto helpdesk di una software agency italiana.
- **sepsis**: contiene eventi di casi di sepsi registrati dal sistema ERP di un ospedale.
- **bpi12**: descrive il processo di un'applicazione di prestiti.
- **nasa**: contiene eventi al livello di chiamate dei metodi della classe NASA Crew Exploration Vehicle descritti da una esecuzione di una esaustiva suite di test d'unità.

Valutazione Sperimentale

Metodo	Media Brier Score	Media Accuracy
Standard	0.020	0.779
Deformable	0.020	0.780
Masked	0.021	0.752

- L'approccio deformabile si è dimostrato in media marginalmente migliore di quello tradizionale, confermando la fondatezza delle assunzioni.
- L'approccio a kernel fisso invece si è dimostrato peggiore in tutte le metriche, anche qui secondo le aspettative.
- I due metodi implementati hanno tuttavia quasi sempre mostrato un numero minore di parametri nei modelli finali, occupando di conseguenza uno spazio inferiore

Conclusioni

Sono da indagare i *contributi* di ogni punto nel campo recettivo deformato e da testare diverse forme di *inizializzazione dei pesi* che possono influenzare differentemente la struttura del campionamento, ampliando ulteriormente l'area di ricerca nelle reti convoluzionali in contesti diversi dalla computer vision

Grazie per l'attenzione