

Expressions Recognizer

Membri del Team		
Nome e Cognome	Numero Di Matricola	Indirizzo email
Alessio Erasmo	273323	Alessio.erasmo@student.univaq.it

Repository Github: https://github.com/alessioerasmo/Expression_Recognizer

Cos'è Expressions Recognizer

Questo progetto contiene un agente intelligente in grado di riconoscere con un certo grado di accuratezza delle espressioni matematiche scritte a mano di lunghezza variabile.

Ecco alcuni esempi:



3 ÷ 4



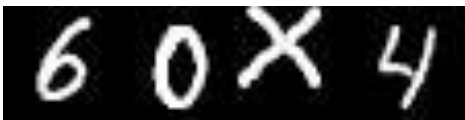
7 - 7 * 5 - 3



8 * 8 2



4 1 + 5



6 0 * 4



7 5 * 9 2

Per ognuna di esse l'agente Intelligente restituisce una stringa contenente l'espressione letta.
(Nei due esempi visti sopra, l'output sarà "3/4", "7-7*5-3" e così via)

Uso Dell'Agente

Dopo aver effettuato il download della repository è sufficiente seguire questi passi.

Per rendere disponibile l'agente è necessario fare i seguenti import:

```
from expressionsAgent import *           # codice agente
from expressionsEnvironment import *      # codice environment
```

Dopodichè bisogna creare l'environment per l'agente, passandogli in input l'immagine che vogliamo che il nostro agente legga.

Per farlo, ad esempio, possiamo leggere un'immagine dal filesystem utilizzando la classe "Image" della libreria "PIL":

```
from PIL import Image
img = Image.open("imagepath\image_to_read.jpg")
```

una volta che abbiamo letto l'immagine possiamo creare l'environment, passarlo in input all'agente e fargli leggere il contenuto:

```
exp_env = expressionsEnvironment(img)      # creazione environment
expr_ag = expressionAgent(exp_env)         # creazione agente

pred = expr_ag.go()                       # prediction stringa

pred = expr_ag.go(evaluate=True)          # prediction intero
```

Caratteristiche dell'immagine in input

Poichè l'elaborazione (preprocessamento) dell'immagine richiede tempo ulteriore e maggiori competenze, si è deciso di tralasciare questa parte, quindi l'agente assume che l'immagine in input abbia le seguenti caratteristiche:

L'espressione è una sequenza di immagini .jpg in scala di grigi di taglia 28x28, ognuna di queste contiene un numero oppure un'operazione

Esempio:

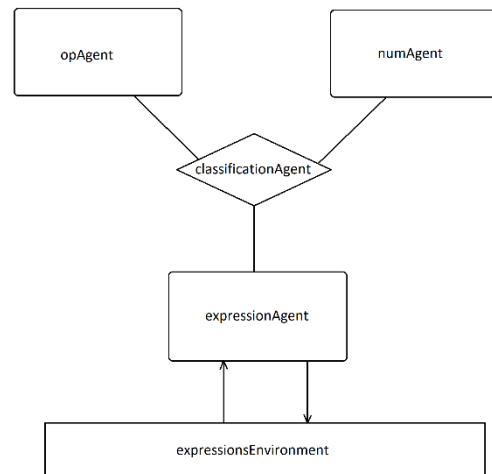


(NB: si potrebbero potenziare le capacità dell'agente se tramite tecniche di computer vision diventasse in grado di ricondurre a questa forma immagini di forma diversa)

Struttura dell'agente

L'agente "expressionAgent" fa uso di altri 3 agenti, ognuno allenato tramite tecniche di machine learning (guarda capitolo successivo) con uno scopo specifico

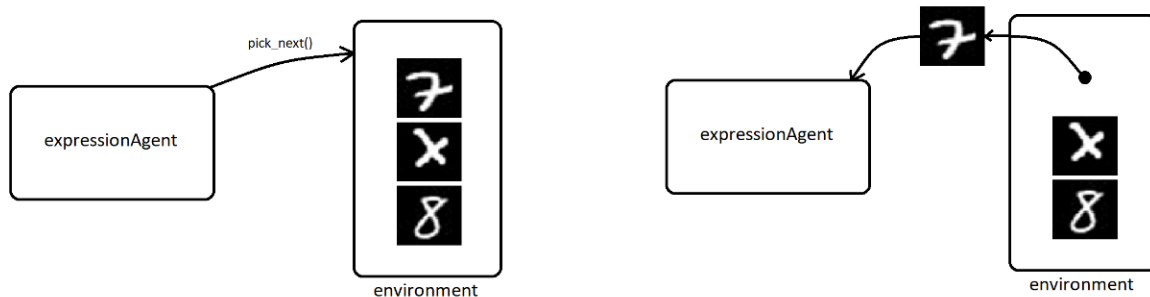
- opAgent: l'agente prende in input una o più immagini contenenti un carattere numerico e prova ad indovinare il o i numeri corretti
- numAgent: l'agente prende in input una o più immagini contenenti un operatore binario tra quelli classici: somma, sottrazione, divisione e moltiplicazione e prova ad indovinare i corretti operatori
- classificationAgent: l'agente prende in input una o più immagini e per ognuna vede se rappresentano un'operazione oppure un numero



Funzionamento dell'agente ed interazione con l'environment

una volta inizializzato correttamente gli agenti ed aver ottenuto in input un environment nella forma corretta, il processo che avviene è il seguente:

- 0) l'environment "trasforma" l'immagine in uno pseudo stack da cui l'agente può fare ogni volta un'operazione simile a quella di "pop"
- 1) L'agente preleva dall'environment una figura(pop):



- 2) L'agente interroga il "classificationAgent" per vedere se l'immagine rappresenta un numero oppure un'operazione.
NB: per minimizzare la possibilità di errore, il classification agent non viene usato sempre. Infatti, quando l'agente sta per leggere per la prima volta un'immagine oppure legge un'immagine subito dopo aver letto un'operazione, ha la certezza che la prossima immagine sarà un numero, quindi non ha bisogno di classificare quelle immagini.
- 3) In base al risultato del classificatore, l'immagine viene mandata all'agente opportuno: opAgent se viene rilevata un'operazione oppure numAgent se viene rilevato un numero.
- 4) Il risultato è poi concatenato alla stringa contenente la risposta. Dopodichè e si torna al punto 1, fino a quando lo stack non è vuoto. Infine, viene restituito il risultato

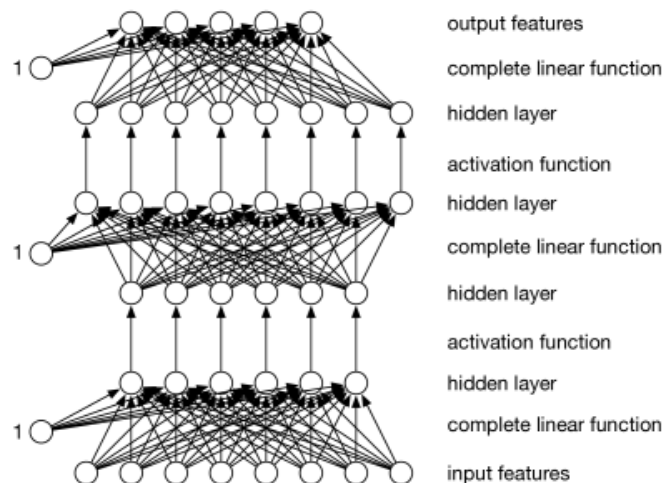
Modelli di machine learning utilizzati

Di seguito verranno illustrati i modelli di machine learning alla base di ogni agente utilizzato dall'agente principale.

Tutti e 3 questi agenti utilizzano una rete neurale di tipo "feed-forward", la cui struttura può essere riassunta dall'immagine a destra.

Ognuno dei modelli presi in esame ha come input features tutti i 784 pixel dell'immagine 28x28.

Tutti i modelli illustrati di seguito sono stati generati mediante l'utilizzo delle librerie "keras" e "tensorflow".
I valori delle input features corrispondono alle intensità in scala di grigi dei pixel.
Il valore, che di norma varierebbe tra 0 e 255 viene poi schiacciato nell'intervallo 0-1.



numAgent

Si può vedere la struttura della rete neurale utilizzata da questo agente nel file "Learning/dilatedMNISTlearning.py":

Ogni layer è separato da una funzione di attivazione, partendo dall'ultima:

- **Softmax**: fa in modo che il valore della somma delle output features sia uguale ad uno. Questo permette alla rete neurale di emettere una risposta sotto forma di densità di probabilità per ogni numero
- **Sigmoid**: funzione che schiaccia x nell'intervallo (0,1)
- **Relu**: definita come $\max(0, x)$

```
model = Sequential()  
model.add(Flatten(input_shape=(28,28)))  
model.add(Lambda(lambda x : x/255.0))  
model.add(Dense(1000))  
model.add(Activation('relu'))  
model.add(Dense(100))  
model.add(Activation('sigmoid'))  
model.add(Dense(10))  
model.add(Activation('softmax'))
```

Oltre alla sigmoide, si è scelto di usare una funzione di attivazione diversa, questo perché utilizzare la funzione Logistica(sigmoide) in troppi layer interni può portare a fenomeni di scomparsa del gradiente.

Infatti, testando diverse configurazioni di reti neurali e funzioni di attivazione, si è visto come utilizzare solo sigmoidi riducesse le performance del modello sull'insieme di testing.

Il dataset utilizzato è quello MNIST, che contiene circa 60'000 caratteri numerici scritti a mano

opAgent

Si può vedere la struttura della rete neurale utilizzata da questo agente nel file "Learning/operatorslearning.py":

I layer e le funzioni di attivazione sono posizionate seguendo lo stesso ragionamento effettuato per l'agente "numAgent". Anche qui, come nel primo esempio, la configurazione finale è stata scelta testando diverse configurazioni di layer e funzioni di attivazione, selezionando poi quella che offriva le migliori prestazioni sull'insieme di testing.

```
model = Sequential()  
model.add(Flatten(input_shape=(28,28)))  
model.add(Lambda(lambda x : x/255.0))  
model.add(Dense(200))  
model.add(Activation('sigmoid'))  
model.add(Dense(20))  
model.add(Activation('relu'))  
model.add(Dense(4))  
model.add(Activation('softmax'))
```

In questo caso il dataset utilizzato è autoprodotta: è stato chiesto a diverse persone di disegnare delle operazioni matematiche, queste operazioni sono poi state salvate e classificate manualmente.

Per ovvie ragioni pratiche, questo dataset non è enorme: conta circa 600 esemplari nel set di training e circa 180 nel set di testing e lo si può visualizzare all'interno della cartella "Learning\Datasets\operators"

classificationAgent:

Si può vedere la struttura della rete neurale utilizzata da questo agente nel file "Learning/op_number_classifier.py":

L'output di questo agente è semplicemente un valore compreso tra zero ed uno, quindi l'attivazione dopo il layer finale è una sigmoide.

La rete restituirà un valore:

- vicino ad 1 se pensa che il risultato sia un'operatore
- vicino a 0 se invece crede che sia un numero

```
model = Sequential()  
model.add(Flatten(input_shape=(28,28)))  
model.add(Lambda(lambda x : x/255.0))  
model.add(Dense(200))  
model.add(Activation('sigmoid'))  
model.add(Dense(1))  
model.add(Activation('sigmoid'))
```

Il set di training utilizzato è un mix tra i due set precedenti.

Dato che vi è una grande sproporzione tra i due dataset, in realtà gli esempi numerici sono solo un piccolo sottoinsieme del MNIST dataset.

Nello specifico, si è visto come avere un rapporto di 3:1 tra le immagini numeriche e quelle contenenti operazioni portasse in media alle migliori performance del modello su vari set di test.

In tutti e tre i casi il numero di "neuroni" per layer è arbitrario:

dato che è molto difficile stabilire quale sia la configurazione ottima, si è cercato, attraverso l'allenamento di diversi modelli, di stabilire quale tra tutti fornisca un migliore compromesso tra dimensioni (complessità) e accuratezza del modello.

Considerazioni e test e prestazioni dell'agente

Un grande limite di questo agente è sicuramente il dataset degli operatori (usato da 2 agenti su 3) di qualità certamente inferiore rispetto al dataset MNIST, sia in termini di numero che di "varianza". Questo perché il campione di persone utilizzato per scrivere i caratteri è molto limitato quando viene messo a paragone.

Nonostante ciò, l'agente in certe istanze riesce a raggiungere un buon livello di accuratezza nelle sue previsioni.

Testing:

Per testare l'agente è stato predisposto lo script "Test\example_generator.py" che preleva valori random dai set di test e genera espressioni random di lunghezza variabile, ecco alcuni esempi:



Queste immagini vengono generate in numero variabile nella cartella "Test\exports" e con loro viene fornito il file di testo "labels.txt" che contiene le stringhe corrispondenti ad ogni espressione, in modo che ognuna di esse possa essere confrontata per valutare l'output dell'agente.

Le immagini generate dallo script "example_generator.py" vengono utilizzate dallo script "test.py", che per ogni immagine chiede una prediction all'agente e la confronta con la sua label, per poi alla fine vedere quante espressioni sono state indovinate sul totale.

Dai risultati si è visto che l'accuratezza dell'agente sui dati generati è di circa:

- tra l'80 ed il 90% per singoli numeri di lunghezza variabile da 3 fino a 7 caratteri



- intorno all'80% per espressioni con un numero di operandi variabile tra 2 e 3, ognuno con una o due cifre



- tra il 70 e l'80% per espressioni con un numero di operandi variabile tra 3 e 4, ognuno con una o due cifre



- tra il 60 ed il 70% per espressioni con un numero di operandi variabile tra 4 e 5, ognuno con una o due cifre



NB: I dati si riferiscono ai modelli esportati presenti nella cartella "Learning\Model_Exports", generati con i parametri di learning presenti nei rispettivi file che li hanno generati.

Ovviamente, l'accuratezza continua a scendere all'aumentare delle dimensioni dell'espressione, questo perché l'agente si affida a diverse predictions, che combinate una dopo l'altra tendono ad aumentare di volta in volta la probabilità di errore.

Quest'ultimo, è di fatto maggior limite dell'agente, che però potrebbe comunque migliorare le sue performance lavorando sui dataset utilizzati per la fase di learning, nello specifico migliorando quello delle operazioni matematiche.