Documentazione Tecnica

Modulo AI

Progetto Smart Office - IoT and 3D Intelligent Systems

Puzone Felicia

Arfaoui Bilel

Macellaro Vincenzo

Panoramica

Il modulo AI del progetto Smart Office per l'esame di IoT and 3D Intelligent Systems è stato sviluppato con l'obiettivo di suggerire alcuni parametri relativi alla personalizzazione del proprio ufficio sulla base di specifici criteri: **temperatura** della stanza, **intensità** e **colore** dell'illuminazione, che vengono proposti all'utente in fase di prima registrazione al sistema e possono essere liberamente confermati o modificati. Tale preferenza viene quindi memorizzata in un DB apposito che verrà periodicamente utilizzato per l'aggiornamento automatizzato del sistema.

Lo sviluppo è stato incentrato sull'utilizzo delle ANN (Artificial Neural Networks, Reti Neurali Artificiali) al fine di ottenere un sistema efficace ed efficiente, ma soprattutto flessibile, in grado di elaborare moli di dati importanti ed aggiornarsi senza dover richiedere interventi sul codice sorgente da parte del programmatore.

L'inferenza, così come l'aggiornamento del sistema, avvengono attraverso il server Flask `OfficeFlask.py`, hostato su un'istanza `EC2` AWS.

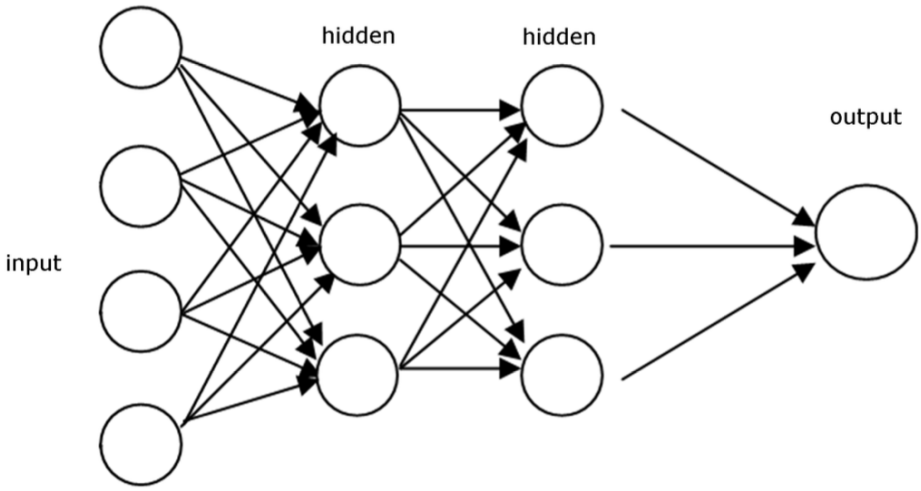
Tecnologie

Reti Neurali

Una **rete neurale** (dall'inglese "neural network") è un modello matematico composto da neuroni artificiali di ispirazione alle reti neurali biologiche, come quella umana, utilizzata per risolvere problemi ingegneristici di Intelligenza Artificiale legati a molteplici ambiti tecnologici come l'informatica o l'elettronica. Tra i numerosi modelli esistenti, si è fatto uso di quello della tipologia **feed-forward**, adatta alla soluzione di problemi di classificazione - come in questo caso - ovvero l'identificazione della categoria di appartenenza di un dato elemento di input.

FFNN

Le reti neurali feed-forward, **FFNN** in breve, sono composte da livelli di nodi (o neuroni artificiali), contengono un livello di input, uno o più livelli nascosti, ed un livello di output; una rappresentazione della loro struttura è visibile nella figura seguente.



Ciascun nodo si connette ad un altro, e ad ognuno di essi sono associati un peso ed una soglia; se l'output del singolo nodo è al di sopra di questo valore di soglia, lo stesso nodo viene attivato ed invia dati al successivo livello della rete, in caso contrario tale passaggio di informazioni non avviene. In questa tipologia le informazioni si muovono in un'unica direzione, ossia in avanti, non si prevedono cicli e non c'è memoria di input avvenuti in tempi precedenti; l'output è determinato esclusivamente dall'attuale input.

TensorFlow & Keras

Sviluppata dal team Google Brain, **TensorFlow** è una libreria software open source per calcolo numerico ed apprendimento automatico su larga scala: riunisce una serie di modelli ed algoritmi di machine learning e deep learning, e può addestrare ed eseguire reti neurali profonde per un enorme numero di obiettivi, tra cui la classificazione di cifre e lettere scritte a mano, il riconoscimento di immagini o l'elaborazione di linguaggio naturale. L'attuale versione 2.11.0 funziona su quasi tutte le principali piattaforme, dai dispositivi mobile e desktop, alle workstation specializzate, dai dispositivi embedded ai cluster distribuiti di server sul cloud. **Keras** è una libreria di alto livello scritta in Python per l'apprendimento automatico e le reti neurali; supporta TensorFlow nel back-end. Keras offre moduli utili per l'organizzazione di differenti modelli tra cui il tipo principale è quello sequenziale, ovvero una pila lineare di livelli, e consente una prototipazione facile e veloce; supporta sia reti convoluzionali (CNN) che reti ricorrenti (RNN) o combinazioni di entrambe, schemi di connettività multi-input e multi-output e funziona sia su CPU che GPU. Keras sfrutta una serie di dipendenze (fra cui *scipy*, *pyyaml*, *HDF5* e *cuDNN*) ed è spesso citata nella letteratura scientifica perché utilissima per le sperimentazioni veloci e poco costose: si passa velocemente dall'idea al risultato senza preoccuparsi delle caratteristiche matematiche dell'algebra tensoriale, dei metodi di ottimizzazione e delle tecniche

numeriche. Differenze chiave tra le due librerie stanno nel fatto che Keras, molto facile da apprendere ed implementare, consente di utilizzare TensorFlow nel backend, tuttavia non sarà adatto nel caso in cui si debbano rendere necessarie al modello modifiche di basso livello. Inoltre, le prestazioni di Keras sono inferiori rispetto a TensorFlow, quindi normalmente la prima viene utilizzata per piccoli set di dati e la seconda per set di dati di grandi dimensioni e che richiedono grandi prestazioni. Keras risulta quindi essere più user-friendly e molto semplice da usare rispetto a TensorFlow, ma quest'ultimo è più adatto per lavori specifici e che richiedono interventi di più basso livello.

Flask

Flask è un micro-framework di sviluppo di applicazioni web scritto in Python, basato sullo strumento Werkzeug **WSGI** (**Web Server Gateway Interface**, protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra server ed applicazioni web scritte nel linguaggio Python; interfaccia standard del web service per la programmazione in Python) e con il motore di template **Jinja2**. È distribuito con licenza libera BSD (**Berkeley Software Distribution**).

È chiamato "micro-framework" perché viene fornito con funzionalità e requisiti minimi incorporati, rendendo semplice l'avvio e flessibile l'utilizzo. Non prevede uno strato di astrazione per la base di dati, validazione dei formulari, o qualsiasi altra componente per fornire funzionalità comuni per le quali esistono librerie di terze parti, ma supporta estensioni che possono aggiungere funzionalità ad un'applicazione come se fossero implementate dallo stesso Flask, tra cui integrazione di DB, account ed autenticazione.

Amazon EC2

Amazon Elastic Compute Cloud, abbreviato in Amazon **EC2**, è una parte centrale della piattaforma di cloud computing Amazon Web Services (AWS) di Amazon, che permette agli utenti di affittare **macchine virtuali** sulle quali eseguire le proprie applicazioni.

EC2 permette l'implementazione di applicazione fornendo un servizio web attraverso il quale un utente può avviare una Amazon Machine Image (AMI), modelli preconfigurati per le istanze contenenti i pacchetti di bit necessari per il server (compresi il sistema operativo ed il software aggiuntivo) per creare una macchina virtuale che conterrà il software desiderato. Un utente può creare, lanciare, e chiudere istanze, pagando i server a ora (da cui l'aggettivo "elastica" nel nome del servizio).

EC2 fornisce agli utenti il controllo sulla collocazione geografica delle istanze che permettono un'ottimizzazione della latenza e alti livelli di ridondanza.

Alcune delle caratteristiche offerte da Amazon EC2 sono:

- Ambienti di elaborazione virtuale, noti come *istanze*;
- Varia configurazioni di CPU, memoria, archiviazione e capacità di rete per le istanze, note come *tipi di istanza*;
- Informazioni di login sicure per le istanze mediante le *coppie di chiavi* (AWS archivia la chiave pubblica, l'utente archivia la chiave privata in un luogo sicuro);
- Volumi di archiviazione per i dati temporanei che verranno eliminati quando l'istanza viene arrestata o interrotta, noti come *volumi di archivio istanza*;
- Firewall che consente di specificare i protocolli, le porte e gli intervalli di indirizzi IP di origine che possono raggiungere le istanze tramite i *gruppi di sicurezza*;
- Indirizzi IPv4 statici per il cloud computing dinamico, noti come *indirizzi IP elastici*;
- Reti virtuali, che possono essere isolate logicamente dal resto del cloud AWS e collegabili alla propria rete personale, note come *cloud privati virtuali* (VPC);

Sviluppo del progetto

Costruzione del dataset

Una rete neurale fa affidamento sui dati di addestramento per migliorare la sua capacità di interpretare e riconoscere dati. Si è reso quindi necessario modellare un **tipo di dato elementare** su cui poter basare tale addestramento: avendo a che fare con una comunicazione fra sistemi diversi ed in Cloud la scelta è ricaduta su un dato in formato **`JSON`,** il quale essendo costituito da coppie **`chiave:valore`** risulta essere facile sia da leggere che da scrivere per le persone, facile da generare e da analizzare per le macchine.

Il dato di base è la cosiddetta Sessione (**`Session`**), un oggetto avente 16 campi:

- `session_id`: stringa identificativa della sessione, tipo di dato **`String`**
- `date`: data della sessione, formato **`yyyy/MM/dd`**, tipo di dato **`String`**
- `open_time`: ora di inizio sessione, formato **`hh:mm:ss`**, tipo di dato **`String`**
- `close_time`: ora di fine sessione, formato **`hh:mm:ss`**, tipo di dato **`String`**
- `zone_id`: identificativo della zona, tipo di dato **`int`**
- `building_id`: identificativo del palazzo, tipo di dato **`int`**
- `user_id`: stringa identificativa dell'utente, tipo di dato **`String`**
- `user_age`: età dell'utente, tipo di dato **`int`**

- `user_sex`: sesso dell'utente, tipo di dato **`int`**
  - `user_sex = 0`: non specificato
  - `user_sex = 1`: maschio
  - `user_sex = 2`: femmina
- `user_task`: categoria di appartenenza del task dell'utente, tipo di dato **`int`**
  - `user_task = 0`: intrattenimento/svago
  - `user_task = 1`: studi/ricerche
  - `user_task = 2`: lavoro d'ufficio
  - `user_task = 3`: lavoro manuale
  - `user_task = 4`: risorse umane
  - `user_task = 5`: servizi pubblici
- `ext_temp`: temperatura ambientale, tipo di dato **`int`**
- `ext_humidity`: umidità esterna, tipo di dato **`int`**
- `ext_light`: intensità di luce ambientale, tipo di dato **`int`**
  - `ext_light = 0`: nulla, associata ad una situazione di buio
  - `ext_light = 1`: bassa
  - `ext_light = 2`: media
  - `ext_light = 3`: alta
- `user_color`: colore dell'illuminazione, tipo di dato **`int`**
  - `user_color = 0`: nessun colore
  - `user_color = 1`: rosso
  - `user_color = 2`: arancione
  - `user_color = 3`: giallo
  - `user_color = 4`: verde
  - `user_color = 5`: verde acqua
  - `user_color = 6`: blu
  - `user_color = 7`: viola
  - `user_color = 8`: fucsia
  - `user_color = 9`: RGB/colore variabile
- `user_light`: intensità di luce del proprio spazio personale, tipo di dato **`int`**
  - `user_light = 0`: luce spenta
  - `user_light = 1`: bassa
  - `user_light = 2`: media
  - `user_light = 3`: alta
- `user_temp`: temperatura utente, del proprio spazio personale, tipo di dato **`int`**

Per generare il DB vero e proprio è stato sviluppato uno script apposito in linguaggio Python, `CreateDB.py` - path: `Script/Generators` - diviso in due parti.

Nella prima parte vengono generati `n_users` utenti, i cui parametri sono ottenuti attraverso l'operatore `UserDataOperator.py` - path: `Script/Operators`.

Nella seconda viene generato il mondo, quindi le `n_zones` zone con i rispettivi edifici, e quindi le `sessions` vere e proprie, su base giornaliera - path: `Script/Operators`. Questa parte utilizza una serie di operatori sviluppati ad-hoc:

- `TemporalDataOperator`: creazione e l'elaborazione di dati temporali, come costruzione ed appropriata formattazione della data relativa ad una sessione e dei timestamp di inizio e fine sessione;
- `WorldDataOperator`: creazione delle `n_zones` zone del mondo
  Ciascuna zona ha un numero (random) tra 3 e 10 edifici - modificabile;
  Ciascun edificio ha un numero (random) tra 3 e 50 uffici - modificabile;
- `GlobalDataOperator`: caricamento e salvataggio dati da e su disco, normalizzazione e conversione dei dati ricevuti in input e da restituire in output;
- `LocationDataOperator`: elaborazione dati relativi ad umidità e temperatura sulla base della posizione geografica della zona, intensità luminosa sulla base dell'orario della giornata;
- `UserDataOperator`: elaborazione dati utente relativi a preferenze di temperatura, colore ed intensità luminosa;
- `SessionDataOperator`: operatore che si occupa di mettere tutti i dati insieme e creare la vera e propria sessione;

Il dataset così sviluppato si compone di

- 131476 sessioni di Training;
- 9391 sessioni di Testing;

salvate come sequenza di file .JSON al path `Data/Sessions/TrainSessions` e `Data/Sessions/TestSessions`.

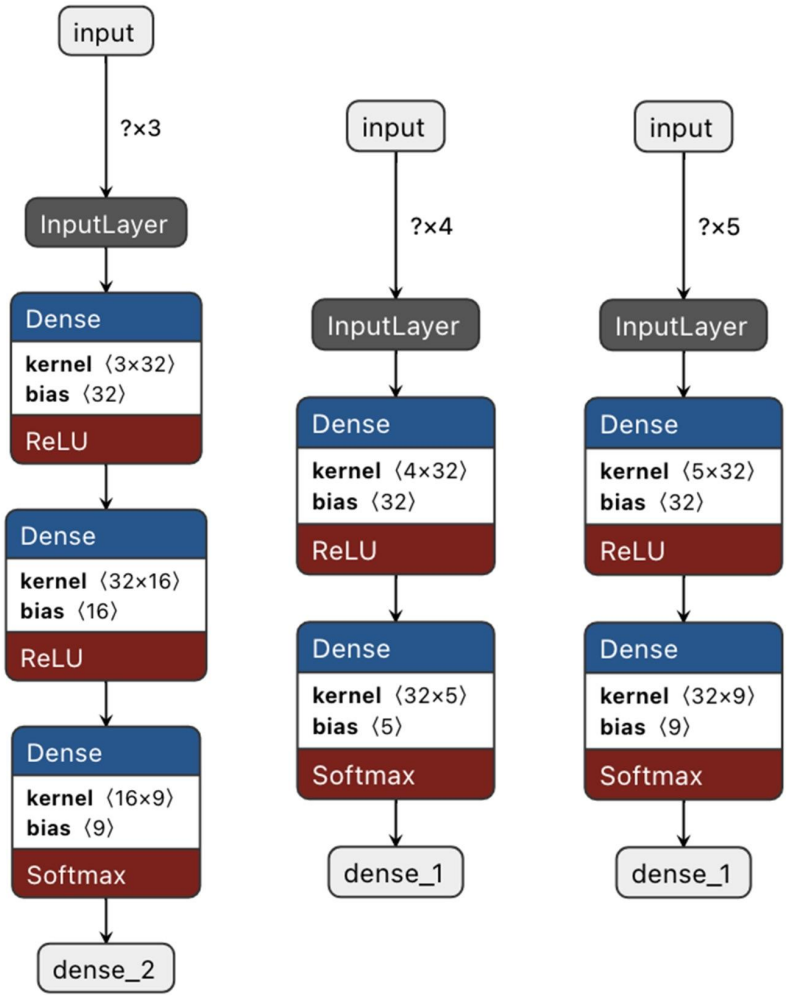
Modelli di rete

Sono stati sviluppati 3 modelli di reti, ognuno per uno specifico obiettivo, implementati attraverso il modello `Sequential` di Keras - path `Script/Generators/ModelSetup.py`.

- `color_model`: utilizzato per la predizione del colore, funziona sulla base dei soli parametri utente - [`user_age`, `user_sex`, `user_task`];
- `light_model`: per la predizione dell'intensità di luce, funziona sulla base di parametri utente ed ambientali - [`user_age`, `user_sex`, `user_task`, `ext_light`];

- `temp_model`: utilizzato per la predizione della temperatura dell'ufficio, funziona sulla base di parametri utente ed ambientali - [`user_age`, `user_sex`, `user_task`, `ext_temp`, `ext_humidity`];

Di seguito le rispettive strutture



Addestramento

Per ciascuna rete è stato previsto un training di reference composto come segue

- Epoche: `2000`
- Learning rate: 10^{-3}
- Optimizer: `Adam`
- Validation split: `0.2`

- Early Stopping: `monitor = 'loss', patience = 4`

`light_model` e `temp_model` hanno richiesto rispettivamente circa `300` e `200` `epoche` per raggiungere una accuracy del 100%, `color_model` ne ha richieste circa 1200 (con learning rate 10^{-4}) per ottenere una accuracy del 99%.

Fase di predizione & server Flask

La predizione dei parametri stabiliti avviene mediante il server Flask `OfficeFlask.py`.

A seconda della route scelta si può scegliere quale delle tre reti interrogare:

- `/getUserColor` fa riferimento il modello `color_model`, riceve il JSON dal server Principale con i dati della sessione attraverso una `POST`, li converte in un `np.array` di `shape (-1, 3)` attraverso l'operatore `GlobalDataOperator`, effettua l'inferenza mediante l'`AIOperator` e quindi invia in risposta il valore ottenuto dalla rete;
- `/getUserLight` fa riferimento il modello `light_model`, riceve il JSON dal server Principale con i dati della sessione attraverso una `POST`, li converte in un `np.array` di `shape (-1, 4)` attraverso l'operatore `GlobalDataOperator`, effettua l'inferenza mediante l'`AIOperator` e quindi invia in risposta il valore ottenuto dalla rete;
- `/getUserTemp` fa riferimento il modello `temp_model`, riceve il JSON dal server Principale con i dati della sessione attraverso una `POST`, li converte in un `np.array` di `shape (-1, 5)` attraverso l'operatore `GlobalDataOperator`, effettua l'inferenza mediante l'`AIOPerator` e quindi invia in risposta il valore ottenuto dalla rete;

Sul server Flask è presente una ulteriore route, `/systemUpdate`, utilizzata per l'aggiornamento del sistema: all'inizio di ogni nuovo mese solare il server triggera il re-training delle reti; scarica dal DB Firebase le sessioni relative al mese precedente, sotto forma di lista di JSON, ne normalizza i valori e quindi procede al re-training. Alla fine del processo sostituisce le reti precedentemente utilizzate con quelle appena sviluppate, che potranno fornire predizioni sulla base dei parametri aggiornati.

Rest API

/getUserColor [POST]

Descrizione: route che restituisce il colore della luce suggerito a partire da un JSON ricevuto mediante richiesta POST. Il suo funzionamento è il seguente:

- Verifica dello stato del server [`server_status`]

- Se il server è disponibile effettua la convalida del JSON, quindi ne converte e normalizza i dati contenuti nel formato appropriato, mediante la function `from_flask_in(data, target, nv_path)` di `GlobalDataOperator`
- Effettua l'inferenza sulla rete `color_model.h5` attraverso l'operatore `AIOperator`
- Invia in risposta il valore ottenuto (`user_color`)
- Se il server non è disponibile invia in risposta la stringa `"000"`, la richiesta verrà così gestita dall'AI locale del server principale

Parametri: `[user_age, user_sex, user_task]`

Rete: `color_model.h5`

Output: `user_color`

/getUserLight [POST]

Descrizione: route che restituisce l'intensità della luce suggerita a partire da un JSON ricevuto mediante richiesta POST. Il suo funzionamento è il seguente:

- Verifica dello stato del server `[server_status]`
- Se il server è disponibile effettua la convalida del JSON, quindi ne converte e normalizza i dati contenuti nel formato appropriato, mediante la function `from_flask_in(data, target, nv_path)` di `GlobalDataOperator`
- Effettua l'inferenza sulla rete `light_model.h5` attraverso l'operatore `AIOperator`
- Invia in risposta il valore ottenuto (`user_light`)
- Se il server non è disponibile invia in risposta la stringa `"000"`, la richiesta verrà così gestita dall'AI locale del server principale

Parametri: `[user_age, user_sex, user_task, ext_light]`

Rete: `light_model.h5`

Output: `user_light`

/getUserTemp [POST]

Descrizione: route che restituisce la temperatura dell'ufficio suggerita a partire da un JSON ricevuto mediante richiesta POST. Il suo funzionamento è il seguente:

- Verifica dello stato del server `[server_status]`
- Se il server è disponibile effettua la convalida del JSON, quindi ne converte e normalizza i dati contenuti nel formato appropriato, mediante la function `from_flask_in(data, target, nv_path)` di `GlobalDataOperator`
- Effettua l'inferenza sulla rete `temp_model.h5` attraverso l'operatore `AIOperator`

- Converte il valore ottenuto dalla predizione nel valore di temperatura appropriato mediante la function `from_temp_idx(idx, uq_path)` di `GlobalDataOperator`
- Invia in risposta il valore ottenuto (`user_temp`)
- Se il server non è disponibile invia in risposta una stringa `"000"`, la richiesta verrà così gestita dall'AI locale del server principale

Parametri: `[user_age, user_sex, user_task, ext_temp, ext_humidity]`

Rete: `temp_model.h5`

Output: `user_temp`

/systemUpdate [POST]

Descrizione: route che avvia l'aggiornamento del sistema all'inizio di ogni nuovo mese solare. Imposta la variabile globale `server_status` a 0 fino al completamento della procedura, rendendo il server temporaneamente irraggiungibile. Il server AI contatta il DB Firebase mediante una GET e scarica tutte le sessioni relative al mese precedente sotto forma di lista di .JSON, ne normalizza i dati e quindi avvia il processo di update delle reti. Alla fine del processo aggiorna sia i modelli che i file utilizzati per la normalizzazione dei dati (`NormValues.json` e `UniqueValues.json`), quindi ritorna raggiungibile settando la variabile globale `server_status` al suo valore iniziale 1.