

**FACOLTA’ DI INGEGNERIA INFORMATICA**

*Laurea Magistrale - AA 2015/2016*

PROGETTO DIDATTICO

**TEORIA DELL’INFORMAZIONE**

Modello di riservazione passiva di banda durante il fenomeno dell’hand-over basato su predizione del path effettuato con Reti Neurali

Matteo Calabrese 174967

Antonio Caliò 175056

Valerio Russo 175057

**ABSTRACT**

Il progetto didattico propone un modello di riservazione passiva di banda sulle celle wireless presenti su un territorio, basato su un modello di predizione del path dell’utente attraverso l’impiego di reti neurali. Nella fattispecie l’algoritmo provvederà a fornire una lista di celle che probabilmente saranno attraversate. L’obiettivo è dunque fornire una stima delle celle quanto più accurata, ed evitare dunque la riservazione di celle che non saranno pertanto attraversate. Per ottenere una buona approssimazione si è fatto uso di 2 strutture di reti neurali (NN1 ed NN2) che si differenziano sulla natura della mobilità. Modellano infatti la nascita e il transito di un utente. I dati statistici della mobilità sono stati ottenuti attraverso il software C4R.

**ANALISI DEI REQUISITI e STRUTTURA BASE**

Per costruire l’applicazione è stato necessario approfondire la conoscenza di un framework di lavoro con cui modellare le reti neurali. Si è scelto di utilizzare *Encog* in abbinamento con il linguaggio di programmazione *Java*. L’applicazione deve essere dunque in grado di predire, sulla base dei dati statistici forniti. Si suppone che sulla mappa siano presenti delle celle, che rispecchiano la copertura wireless delle torri di comunicazione, che sono per comodità rappresentate con forma esagonale. Il sistema sarà dunque il grado di prenotare passivamente le risorse sulle torri adiacenti per far si che il fenomeno dell’hand-over non comporti cadute di connessione.

**DATI STATISTICI E DI MOBILITA’**

Per ottenere i dati su cui basare l’applicazione, è stato usato il software: *C4R*. Citymob per tabelle di marcia è un generatore di mobilità per le reti veicolari . Ciò significa che è un programma che permette di simulare il traffico in differenti posizioni utilizzando mappe reali ottenute da open street map.

****

Le porzioni di mappe scelte, della dimensione di circa 1,3km quadrati comprendono un’area urbana ove sono ovviamente presenti le strade. Per le simulazioni sono generate sempre due tracce, generalmente da 2000 veicoli e senza tenere conto del parametro che modella gli agglomerati urbani. Il modello seguito è quello elaborato da Krauß con parametri di accelerazione , sigma e tau standard. Il tempo di simulazione scelto è di 600s. La visualizzazione della simulazione avviene attraverso il plug-in esterno *SUMO NetConverter*, che è stato poi ripreso ed integrato nell’ interfaccia grafica del software sviluppato per visualizzare la mappa direttamente dal file della traccia.

Gli output della simulazione di nostro interesse sono dunque:

1. *Training.tcl* che contiene le tracce per l’addestramento della rete neurale
2. *Testing.tcl* che contiene le tracce per il test dell’applicazione
3. *Map.net.xml* che contiene i dati per ricostruire la mappa

****

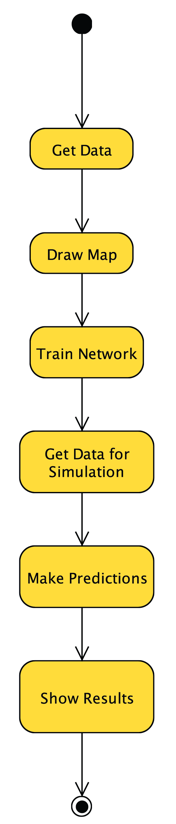
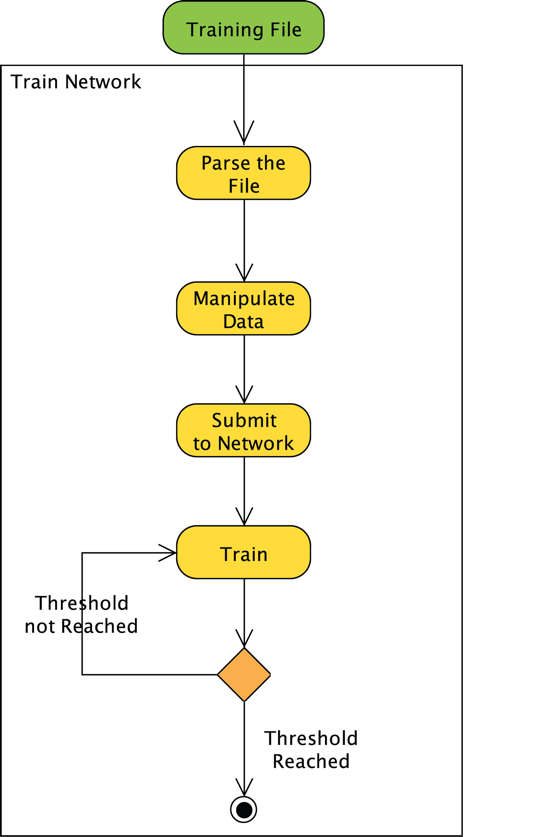
La precedente immagine è un esempio di traccia. Il path del veicolo ‘0’, nella fattispecie le prime 3 righe rappresentano la posizione iniziale nello spazio, mentre le successive righe sono le posizioni del veicolo nello spazio ai vari step temporali.

**FASE PROGETTUALE**

La strategia adottata ha previsto come detto nei precedenti paragrafi l’utilizzo del framework delle reti neurali in abbinamento al linguaggio di programmazione Java estrapolando i dati di mobilità dal software C4R. Si è proceduto dunque alla progettazione dell’applicazione, definendo dettagli e funzionalità attraverso i seguenti diagrammi:

**WORKFLOW:**

I diagrammi di workflow indicano le fasi che l’applicazione svolge per far si che funzioni. Il primo digramma più a sinistra, fornisce un’ottica globale di, mentre quello più a destra si concentra sulla fase di addestramento delle reti neurali e la predizione del path.

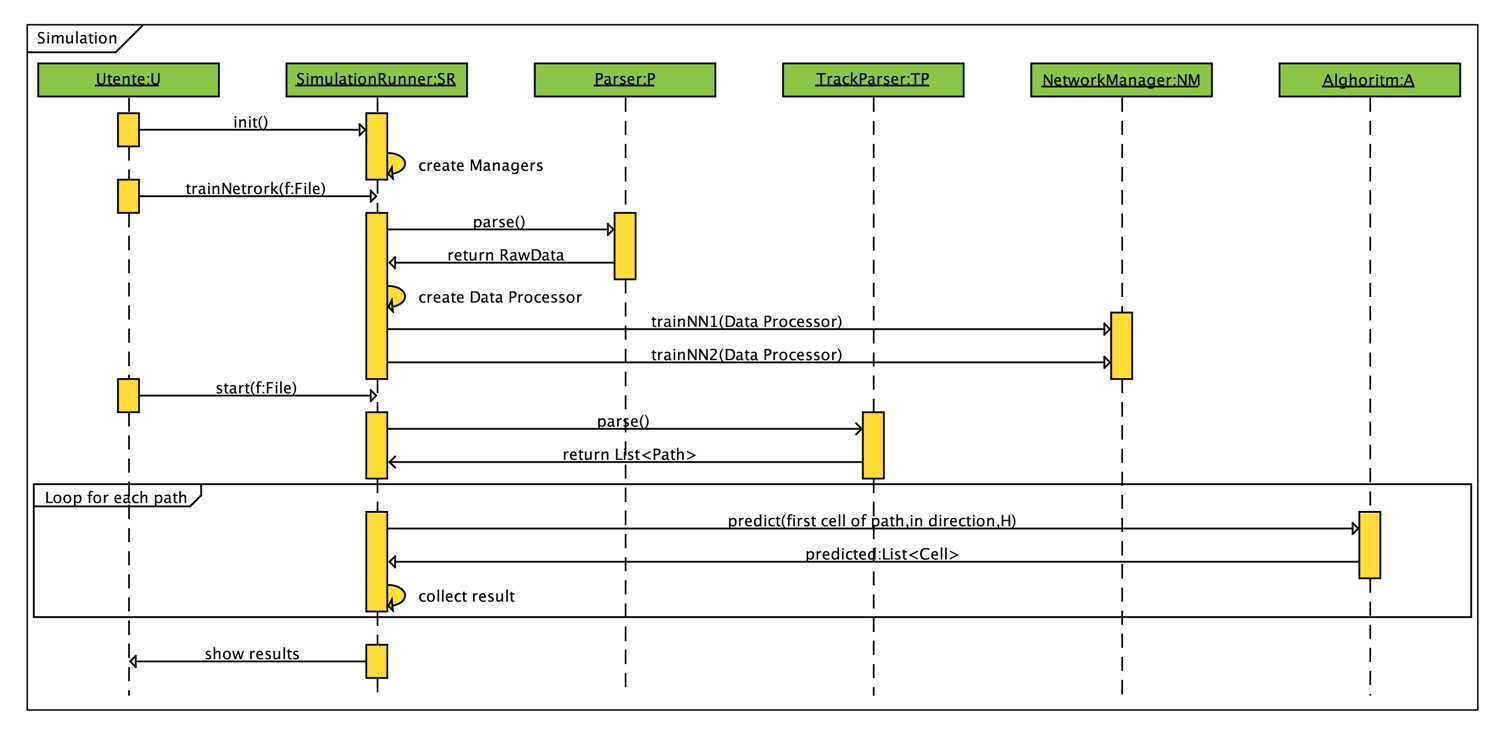
** **

**DIAGRAMMA DELLE CLASSI:**

****

Il diagramma delle classi descrive la struttura utilizzata nella progettazione dell’applicazione. Le classi colorate di giallo sono interfacce e classi astratte. Quelle colorate di verde sono la concretizzazione delle precedenti, mentre quella colorata di viola è una classe di *enum*. Discorso differente invece per la classe *NetworkManager* che nonostante sia una classe astratta è colorata di arancione, semplicemente per darle più risalto in quanto è la classe principale della logica che crea e rende disponibili i dati a quasi tutte le altre classi. La classe *PathPredictor* ne fa uso per lanciare l’algoritmo di predizione.

**DIAGRAMMA DI SEQUENZA:**

****

Tale diagramma mostra le interazioni che sussistono tra le classi precedentemente elencate ed indica come il flusso dati è orchestrato.

1. **Rappresentazione delle celle nella Mappa**

La fase progettuale ha visto dipanare le idee seguendo un approccio bottom-up, costruendo modularmente dal basso verso l’alto le varie classi necessarie. Si è cominciato con l’implementare il package *geographic* che fornisce le operazioni per gestire:

* Mappa
* Coordinate
* Celle

Nello specifico le classi di questo package espongono i metodi per creare il retino tassellabile delle celle sulla mappa, restituire le coordinate e l’appartenenza di una determinata coordinata ad una cella, nonché metodi ausiliari per la creazione della struttura.

1. **Parsing delle tracce**

Nel package *geographic* sono inoltre presenti le classi per effettuare il parsing e la memorizzazione delle tracce fornite dal software. Per eseguire l’addestramento della rete neurale, di cui parleremo più avanti è necessario fornire i dati secondo alcuni standard ben definiti. Semplificando per ora la questione, diremo solo che è nata l’esigenza di estrapolare i dati dalle rispettive tracce e mapparle all’interno di strutture dati idonee: liste ed hashmap.

1. **La struttura delle reti neurali**

Nell’ambito di questo progetto sono state utilizzare due tipi di rete neurale, Sono dunque implementate due soluzioni che chiameremo:

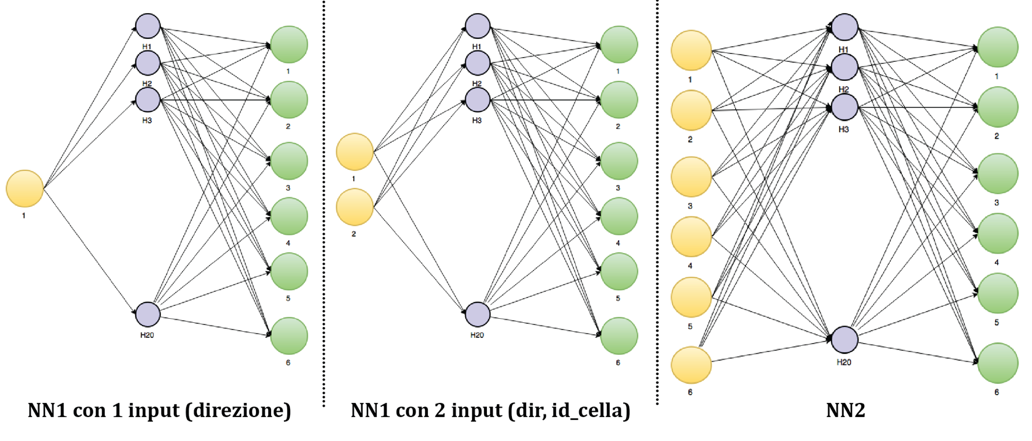
* NN1 (neural network 1)
* NN2 (neural network 2)

La differenza tra le due, oltre che per lo scopo, sta nel tipo di input. Infatti la rete neurale 1, ha in input 1 o 2 parametri a seconda dell’approccio utilizzato che può essere di tipo:

* **DISTRIBUITO** ( 1 neurone in input)
* **CENTRALIZZATO** (2 neuroni in input)

L’approccio centralizzato prevede la presenza di un’unica rete neurale che gestisce l’intero sistema, vale a dire che esiste una sola rete per tutte le celle presenti. Prende in input 2 parametri: l’id della cella e la direzione. L’approccio distribuito invece prevede la presenza di una rete neurale per ogni cella del sistema. Prende in input solo la direzione.

La differenza che intercorre tra NN1 ed NN2 sta semplicemente nel fatto che la rete neurale 1 caratterizza le celle dove nascono gli utenti, mentre la rete neurale 2, che ha 6 neuroni in input, descrive le celle in cui gli utenti transitano. Entrambe le reti hanno 6 neuroni di output che nient’altro sono che il numero di lati della cella esagonale.



1. **Peculiarità delle reti neurali**

Le reti neurali artificiali sono modelli matematici che rappresentano l'interconnessione tra elementi, chiamati: neuroni artificiali, ossia costrutti matematici che in qualche misura imitano le proprietà dei neuroni viventi. Questi modelli matematici possono essere utilizzati sia per ottenere una comprensione delle reti neurali biologiche, ma ancor di più per risolvere problemi ingegneristici. In termini pratici strutture non-lineari di dati statistici organizzate come strumenti di modellazione. Tali reti, possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riescono a rappresentare.

Le principali caratteristiche sono:

* forte parallelismo
* forte interconnessione
* rapida comunicazione
* semplici unità elaborative
* grande quantità di memoria nelle connessioni
* naturalmente associativa
* non programmata, ma addestrata mediante apprendimento automatico

Una rete neurale riceve segnali esterni su uno strato di nodi (unità di elaborazione) d'ingresso. Ciascuno dei quali è collegato con i nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.

In sintesi, la struttura di una rete neurale è composta da:

* + un insieme di *unità elaborative*
  + uno *stato di attivazione*
  + una *funzione di output* per ogni unità
  + le *connessioni* fra le unità
  + una *funzione di attivazione* per combinare gli input con il valore di attivazione corrente e produrre un nuovo livello di attivazione
  + una *regola di apprendimento* per modificare le connessioni

La struttura appunto, non viene programmata per eseguire una certa attività, bensì addestrata (utilizzando un algoritmo di apprendimento) mediante una serie di esempi della realtà da modellare (dataset di training). Sono stati previsti 4 differenti algoritmi di addestramento/apprendimento supervisionati:

* + - 1. **RESILIENT PROPAGATION**
      2. **BACKPROPAGATION**
      3. **MANHATTAN PROPAGATION**
      4. **QUICK PROPAGATION**

Ognuno con differenti peculiarità e punti di forza. Discorso diverso invece per le connessioni nascoste tra le unità e sui meccanismi di propagazione dei dati tra i neuroni. Nel nostro caso lo strato più interno, detto *hiddenlayer* contiene 20 neuroni. Questo numero scaturisce dalle opinioni trovate nei paper che trattano questa materia, in cui si sottolinea come tale numero è frutto di un trade-off tra efficienza e complessità. La funzione di attivazione tra i neuroni è la *Sigmoide* che è continua e derivabile, avendo una derivata prima non negativa o non positiva. E’ dotata di un minimo locale ed un massimo locale.

1. **Dati di addestramento delle reti neurali**

Per lanciare gli algoritmi di propagazione, è necessario fornire dei dati ben strutturati ed organizzati sulla base del layout della rete. Le classi di network.data e nella fattispecie:

* *Parser*
* *TrackParser*
* *DataProcessor*
* *BasicData Processor*
* *NN2MLDataPair*

mettono a disposizione tutte le funzionalità necessarie per leggere il file con le tracce, estrapolare i dati per addestrare le due reti neurali ed infine memorizzarli nella struttura dati richiesta (matrice di double).

Distinguiamo tre tipologie di strutture:

* **Struttura per NN1 DISTRIBUITA:** per ogni traccia, si considera la direzione di avanzamento dell’utente: [direzione]
* **Struttura per NN1 CENTRALIZZATA:** per ogni dato della traccia, si risale alla cella in cui ricade e si individua la direzione di avanzamento dell’utente: [id\_cella,direzione]
* **Struttura per NN2:** essendo celle di transito, si risale al lato di ingresso dell’utente nella cella, e si definisce l’array. Si hanno dunque 6 valori nel caso di celle esagonali: [lato1,lato2,lato3,lato4,lato5,lato6]

I dati provenienti dalla traccia sono poi oggetto di un lavoro di normalizzazione, in un range compreso tra [-1,1], nonchè di un lavoro statistico. E’ necessario raccogliere tutte le tracce e per ogni cella calcolare il numero di volte che un utente entra ed esce da un determinato lato. Infine bisogna fornire dei valori che rispettano questo stato. Ad esempio la coppia: [0,1,0,0,0,0] [0.4,0,0,0,0,0.6] indica che l’utente entra nella cella dal lato2 ed esce con probabilità *0.4* dal lato1 e con probabilità *0.6* dal lato 6. Le probabilità sono detti pesi, e si ottengono solo dopo aver effettuato il processo di training. In tale processo è necessario stabilire una soglia di precisione che in abbinamento al numero di epoche fornirà addestramenti e dunque valori più o meno precisi.

1. **L’Algorimo di Predizione**

****Le descrizioni sopra elencate non sono altro che le fondamenta su cui si basa il cuore del progetto, ossia l’algoritmo di predizione che è definito dal seguente pseudocodice:

Si determina inizialmente un upper bound (H) che rappresenta il numero di celle probabilmente visitate da un utente durante la traccia. Si determina un input per la rete NN1 a seconda che si utilizzi un approccio centralizzato o distribuito e la si avvia. Fintanto che una variabile h è minore di H, si procede a valutare ogni valore di probabilità delle celle adiacenti e si incrementa h. Se questo valore di probabilità è più alto della soglia di predizione, si procede a prelevare la direzione indicata dalla rete neurale NN1 per poi chiedere alla rete di ottenere gli adiacenti ed effettuare la riservazione passiva. Dalla cella predetta si estrapola la direzione di uscita e la si assegna ad un nuovo input per la rete NN2 che fornirà a sua volta un nuovo output che sarà poi assegnato in maniera ricorsiva alla successiva rete fino a quando non sarà più vera la condizione h<H. Al termine di questo procedimento, si avrà a disposizione la lista di celle predette.

**GUI & FASE DI TEST**

Dopo aver concluso la fase progettuale e verificato in maniera analitica la correttezza di quanto realizzato, l’algoritmo è stato testato con alcuni dataset creati per l’occorrenza. Sono stati eseguiti i test su una porzione di mappa della città di Londra caratterizzata da 2000 tracce e generata secondo i parametri indicati precedentemente. Il file di testing è stato però ridotto a 250 tracce per comodità computazionale.

* 1. **Interfaccia Grafica**

Per rendere più fruibile l’applicazione e dunque permettere agli utenti di inserire una propria mappa e dei propri dati, si è pensato di implementare l’interfaccia grafica del progetto.

Dopo averla lanciata, appare la finestra in cui è possibile caricare i dati e settare tutti i parametri richiesti dal metodo *runner* che fisicamente provvede all’esecuzione della parte logica.



In preset c’è un insieme di parametri ottenuti sulla base della rilevanza dei risultati , scartando dunque quelli che poco significativi.

Di seguito il riepilogo del significato dei parametri:

* **Soglia Training**: il valore massimo di errore che l’addestramento della rete neurale può tollerare
* **Struttura della rete**: può essere Distribuita o Centralizzata. La differenza risiede solo nella struttura della rete neurale NN1.
* **Tipo di propagazione**: E’ la tipologia di algoritmo utilizzato nella fase di addestramento della rete neurale.
* **Soglia algoritmo**: è il valore che indica la percentuale da soddisfare per la propria predizione. Es: un valore di soglia 0.30 significa che bisogna restituire un output che sia verificato almeno nel 30% dei casi. All’atto pratico, il lato di uscita di una cella è determinato dalle tracce che attraversano quella cella in determinate circostanze ed escono almeno nel 30% delle volte da quel lato.
* **Prediction Step**: rappresenta il valore H dell’algoritmo. E’ l’upperbound delle chiamate ricorsive.
* **Lato della cella**: banalmente indica la dimensione del lato della cella

Una volta premuto il pulsante *AVVIA*, dopo aver costruito la mappa attraverso il plug-in SUMO *NetConverter*, l’applicazione provvede ad effettuare il *train* e dunque a fornire la *predizione* del path.



Al termine di questa operazione è possibile visualizzare il set delle tracce predette con le relative proprietà :



Cliccando sulla singola traccia, è possibile visualizzare graficamente le celle correttamente predette, quelle sprecate e quelle non predette.



* 1. **Ottimalità dell’applicazione**

Per ottenere una stima sulla bontà della soluzione implementata, si è pensato di introdurre una classe *SimulationResult* che in abbinamento alle classi *PathMatcher* e *SimulationParameter* fornisce una stima dell’accuratezza e dello spreco.

Effettuando un foreach sulle tracce ottenute dal PathMatcher si calcolano i parametri di:

* **Accuratezza** effettuando un rapporto tra il numero di celle correttamente predette ed il numero di celle realmente attraversate.
* **Spreco** definito come il rapporto tra il numero di celle sprecate ed il numero di celle predette.

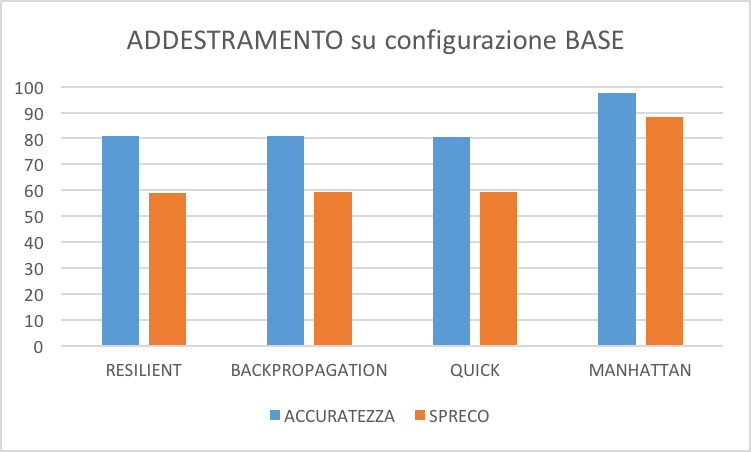
Entrambi i valori sono normalizzati con il numero di tracce derivanti dalla classe *PathMatcher.*

* 1. **Grafici**

Per fornire una visualizzazione analitica del lavoro effettuato, si è proceduto a rapportare i risultati prodotti attraverso dei grafici. Nello specifico si è proceduto dapprima a capire quale fosse la configurazione di riferimento, ed in seguito si è provveduto a verificare quale fosse il miglior algoritmo di addestramento. La configurazione di base scelta è:

* **Struttura della rete**: DISTRIBUITA
* **Soglia Training**: 0,04
* **Soglia dell’algoritmo**: 0,30
* **Prediction Step**: 7
* **Lato della cella**: 120

Il miglior algoritmo di addestramento è risultato essere **“RESILIENT PROPAGATION”** che garantisce una buona accuratezza ed uno spreco leggermente minore rispetto agli altri.

****

Balza subito agli occhi, che l’algoritmo Manhattan arriva ad un picco di accuratezza pari al 97,58%. Ha però di contro l’alta percentuale di spreco pari al 44%. Di seguito la tabella che riporta i dati ottenuti:

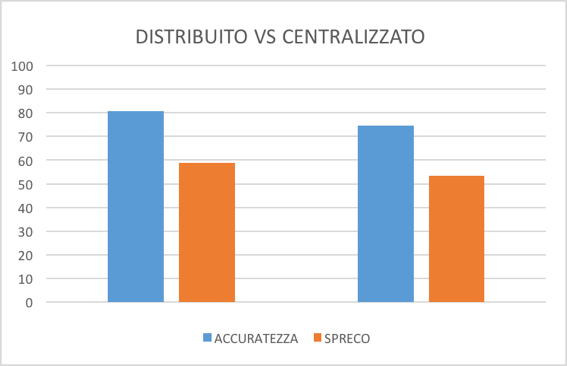
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Prop.Type** | **Approccio** | **H** | **AlgThr** | **TrainThr** | **Dim.Cella** | **#Celle** | **ACC** | **SPR** | **DIFF** |
| RESILIENT | DISTRIBUITO | 7 | 0,04 | 0,3 | 120 | 63 | 80,81 | 58,9 | 21,91 |
| BACKPROPAGATION | DISTRIBUITO | 7 | 0,04 | 0,3 | 120 | 63 | 81,06 | 59,26 | 21,8 |
| QUICK | DISTRIBUITO | 7 | 0,04 | 0,3 | 120 | 63 | 80,61 | 59,51 | 21,1 |
| MANHATTAN | DISTRIBUITO | 7 | 0,04 | 0,3 | 120 | 63 | 97,58 | 88,44 | 9,14 |

La colonna *DIFF*, indica la differenza tra accuratezza e spreco. Per i successivi test si fissa il riferimento sulla configurazione di base con addestramento RESILIENT.

**APPROCCIO**

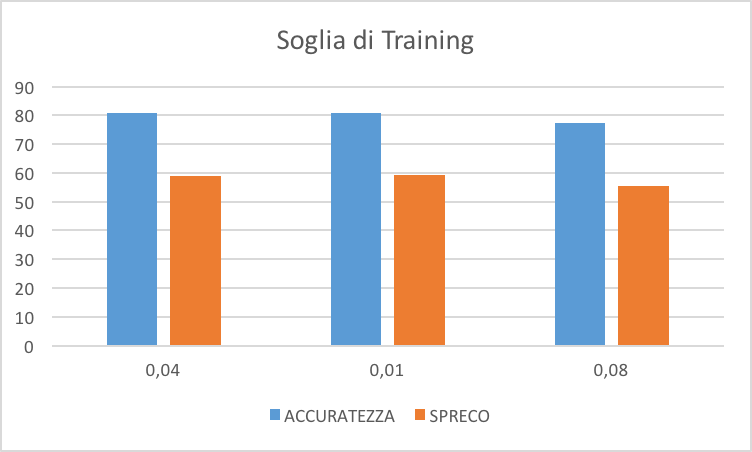
L’approccio utilizzato nella configurazione base è **distribuito**. Variandolo, e dunque utilizzando quello **centralizzato**, si ottiene che la percentuale di accuratezza si riduce del 6% mentre lo spreco di soli 4,5 punti percentuali.

In definitiva, quello distribuito è da prediligere.



**SOGLIA DI TRAINING**

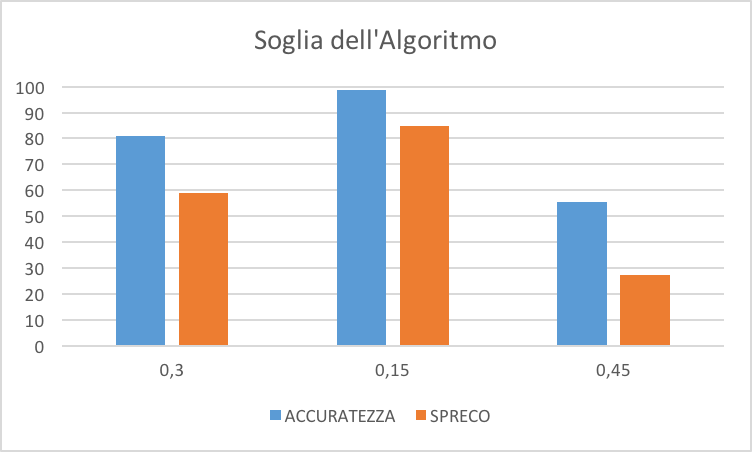
Se dalla configurazione di base si varia il parametro della **soglia di training** della rete neurale, portando il valore da *0.04* a *0.01* ed a *0.08* si ottiene: nel secondo caso una leggera flessione nei valori di accuratezza e spreco; mentre nel primo caso la situazione resta pressoché invariata. Conviene quindi mantenere una soglia *a 0.04* in quanto il processo di addestramento risulta un po’ più rapido.

**SOGLIA DELL’ALGORITMO**

Effettuando invece dei test sul parametro che determina la soglia dell’algoritmo, si evince una sostanziale differenza tra i dati di output.

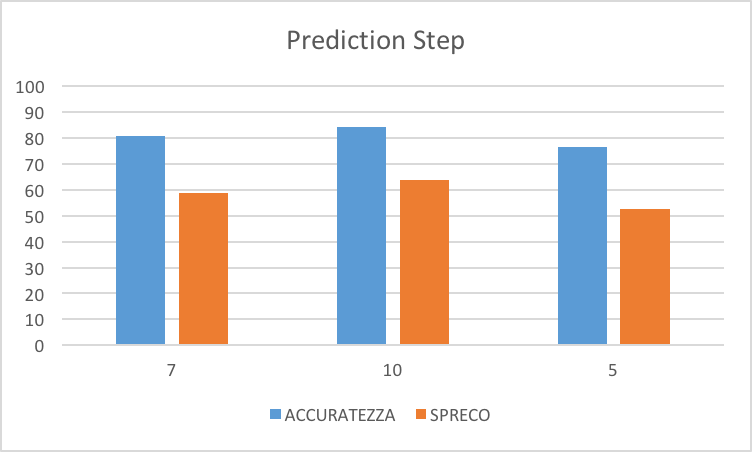
E’ subito evidente come una soglia di *0,15* procuri un aumento di precisione prossimo al 99%. Purtroppo però, l’aumento riguarda anche il valore di spreco che sfiora l’85%.

Utilizzando invece una soglia di *0,45*, rispetto al valore di riferimento, si ha un abbassamento notevole di precisione e di spreco. Questo perché non si hanno sufficienti dati per poter addestrare la rete e dunque avere una predizione più accurata con queste caratteristiche.



**PREDICTION STEP**

Tale valore rappresenta l’upperbound delle chiamate ricorsive. Nella configurazione di base è fissato a *7*. Si è pensato di effettuare delle variazioni sia aumentando che diminuendo tale valore. Effettuando una variazione con parametro *10* si ottiene un leggero innalzamento di entrambi i valori di accuratezza e spreco, mentre effettuando una diminuzione fino a *5*, si ottiene un decremento di entrambi i parametri. Questo comportamento tuttavia è atteso. Si è pensato dunque di utilizzare come trade-off, il corrispondente valore *7*.



**LATO DELLA CELLA**

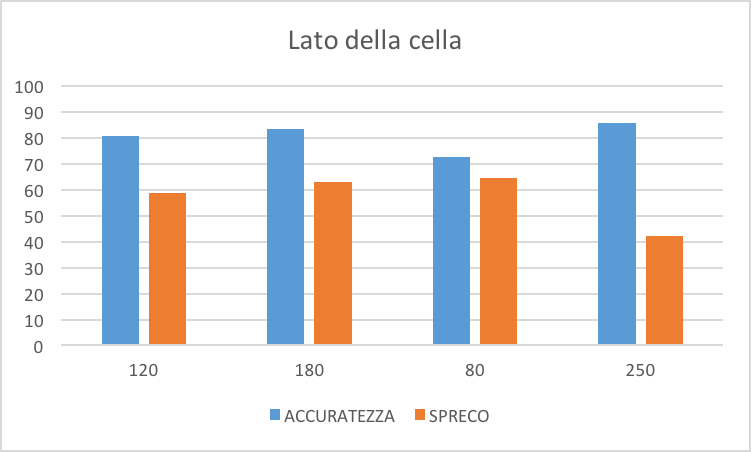
Il parametro più interessante da variare è sicuramente la dimensione della cella. Rispetto alla configurazione base si è provveduto ad effettuare più simulazioni con differenti dimensioni di lato.

Settando un lato di *180* e di *80* si osserva un comportamento tipico, ossia nel primo caso una lieve crescita dell’accuratezza e dello spreco. Nel secondo caso invece abbiamo un decremento della precisione ed una crescita del numero di celle non utili. Questo è dovuto alla mancanza di sufficienti dati di addestramento della rete che ovviamente ne soffre quando la cella si rimpicciolisce.

Un discorso differente va invece fatto nel caso in cui il lato della cella è fissato a *250*. Dal grafico si evince un piccolo incremento dell’accuratezza ed una notevole discesa dello spreco prossima al 40% contro una media del 60%. Il motivo risiede sempre nella numerosità di tracce adoperate per l’addestramento. E’ ovvio quindi che una cella che occupa una maggiore porzione di mappa dispone di più dati per poter effettuare la propria previsione, ragion per cui, sarebbe opportuno fornire un training corposo alla rete.

Il motivo per cui non è stato fatto, risiede nel limite fisico del software utilizzato per generare il dataset, in quanto superando un determinato numero di tracce (poco più di 2000), lancia l’errore *heap overflow*.

Si stima che utilizzando un numero 10 volte maggiore di tracce, si sarebbero potuti raggiungere livelli di accuratezza nettamente migliori con spreco decisamente minore dei valori graficati.



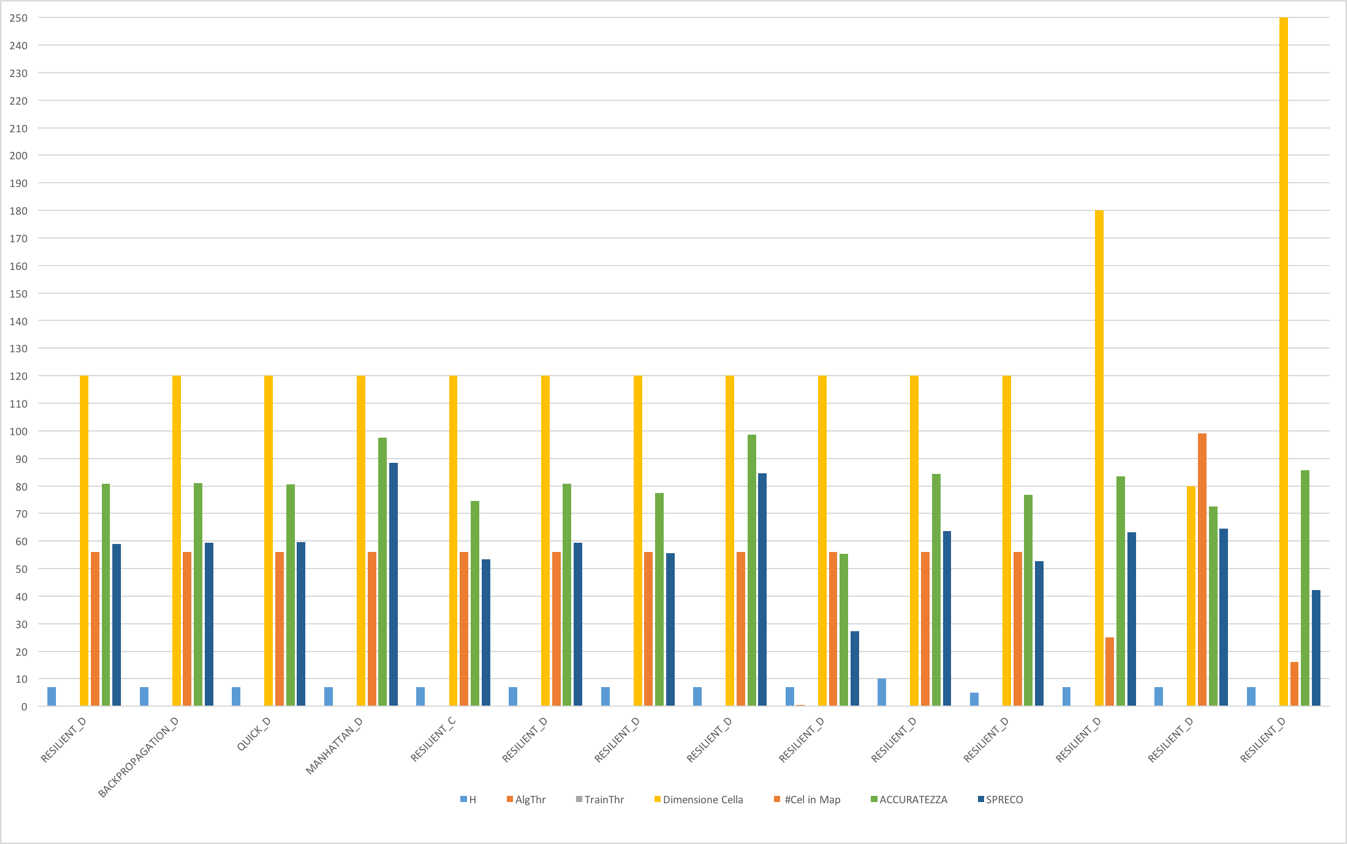
**RIEPILOGO**

L’epilogo di questo lavoro giunge generando una visione globale dei test effettuati.

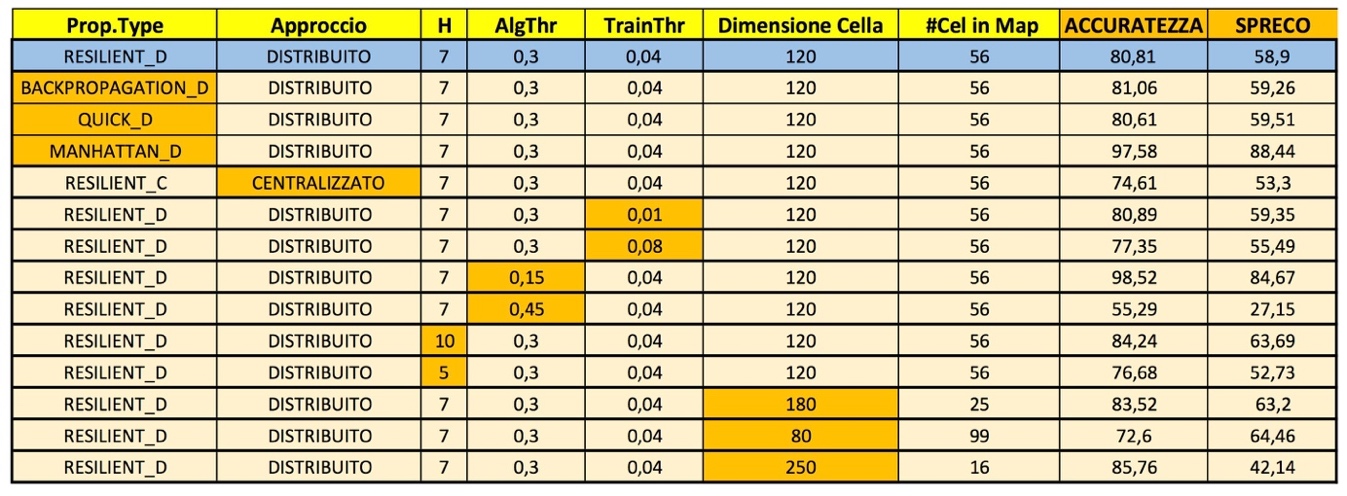
In maniera evidente, si afferma che a determinare una sostanziale variazione dei parametri di bontà dell’applicazione sono i valori di :

* Soglia dell’algoritmo
* Dimensione della cella

Infatti l’ultimo blocco di barre, rappresenta uno stato ottimale del sistema.



Di seguito invece i dati che indicano lo scostamento dei valori rispetto alla configurazione base evidenziata in turchese:



**CONCLUSIONI**

Realizzare questo sistema ha rappresentato una sfida nella comprensione e nelle dominazione della tipologia di struttura tecnologica che sta alla base della predizione. Pur non avendo alcuna nozione passata di metodi di apprendimento e reti neurali, è stato interessante essere riusciti ,con buona approssimazione, a determinare la predizione ed a vedere dunque un buon funzionamento.

Questa applicazione, seppure distante da uno scenario reale, ci ha consentito di conoscere ed approfondire i meccanismi alla base di tanti sistemi che ignoravamo.

Uno sviluppo futuro potrebbe essere migliorare l’algoritmo di predizione, eliminando il valore H che limita staticamente il percorso e sostituendolo con un parametro dinamico che potrebbe scaturire dall’analisi della mobilità degli utenti effettuata tramite un processo di analisi di BigData.