Hopfield Network

Nicola Sebastianutto

August 2025

1 Introduzione

In questa relazione verrà discusso delle diverse scelte di design e di implementazione del codice, analizzando le principali funzionalità e scopi di ogni classe. Sarà motivata la scelta delle librerie usate e verranno fornita altri appunti necessari per comprendere il codice.

2 L'applicazione

Il programma genera un'applicazione interattiva dove l'utente può caricare delle immagini o generare pattern casuali. Tramite gli algoritmi della rete di Hopfield è possibile memorizzare dei pattern. Infatti, lo scopo principale di queste reti è quello di funzionare come una "memoria associativa". Questo significa che le reti neurali possono memorizzare diversi "pattern" sotto forma di bit. Come primo step va effettuato il training, ossia la rete neurale deve leggere ogni singolo bit di ogni pattern che si vuole memorizzare, poi tramite un algoritmo viene generata una matrice dei pesi. Tramite questa matrice è possibile riportare un pattern di un'immagine corrotta al suo stato originale.
Per la costruzione di questa matrice sono stati implementati due algoritmi: il primo è l'algoritmo di Hebb, che può memorizzare al massimo $0.138 \times N$ pattern, dove N è la dimensione del pattern. Se i pattern hanno in comune molti pixel, il training genera una sovrapposizione delle immagini di input, producendo un'immagine che è una combinazione delle originali. In questo caso l'algoritmo di risoluzione del pattern convergerà verso questo stato spurio.
L'altro algoritmo che usa la Pseudoinversa è più preciso e risolve questo tipo di problema, ma ha lo svantaggio di avere un addestramento computazionalmente più costoso.
Nel progetto sono stati inseriti due eseguibili: uno per i numeri reali e uno per i numeri complessi. Le reti di Hopfield, oltre a memorizzare numeri binari (rappresentati da -1 e $+1$), possono essere addestrate con numeri complessi in modo da memorizzare immagini in scala di grigi (bianco, grigio chiaro, grigio scuro e nero).
Dato che l'addestramento e la risoluzione del pattern richiedono un grande calcolo computazionale, l'utente può decidere di interrompere il processo. Un'altra

funzionalità del programma è la possibilità di salvare gli addestramenti memorizzando i dati in un file per un eventuale uso successivo.

Appena viene inserita un'immagine, questa viene convertita in pixel e l'utente può decidere la sua risoluzione, con un intervallo da 2x2 a 64x64 pixel. Viene mostrata sullo schermo la rappresentazione dello stesso pattern in tre diversi stati da sinistra a destra: il pattern originale, il pattern corrotto (che può essere modificato sia aggiungendo rumore sia interattivamente) e infine il pattern dinamico che mostra l'applicazione dell'algoritmo.
sulla destra è riportato il grafico dell'energia associata ad ogni passaggio dinamico.
Importante: l'esecuzione dell'algoritmo aggiorna i pixel una volta sola, quindi potrebbe essere necessario eseguire l'algoritmo più volte per raggiungere un minimo locale di energia.

3 Classi di `HopfieldSimulator`

3.1 `ITrainingPattern`

Il pattern con cui verrà addestrata la rete può essere un'immagine presa da un file oppure un pattern generato casualmente. Per questo è stato necessario costruire un'interfaccia `ITrainingPattern` con due implementazioni. Entrambe le classi derivate sono accomunate dal fatto che rappresentano il pattern con un vettore e forniscono un "getter" e la funzione `regrid()`, ossia il cambio dei numeri dei pixel. Tuttavia, operano in modo diverso:

- `ImageTrainingPattern` ha bisogno del percorso dell'immagine per essere valido; in caso contrario, viene interrotto il programma e mostrando il messaggio di errore corrispondente. L'immagine viene convertita in un'immagine integrale in modo tale che, quando verrà eseguito il `regrid()` (cioè quando verrà cambiato il numero di pixel), questo riscalamento sarà ottimizzato grazie alla pre-elaborazione.

- `RandomTrainingPattern` ha bisogno del livello di rumore come input, che una volta impostato non può essere modificato. Per semplificare il processo, un'operazione di `regrid()` su un'immagine casuale genererà semplicemente una nuova immagine casuale da zero.

3.2 `NoisyPattern`

`NoisyPattern` è la classe che rappresenta il pattern corrotto. Prende come input il vettore di un pattern originale e genera, a partire da un valore di rumore (`noise`), il pattern modificato. Il rumore può essere applicato quante volte si vuole. Se il rumore è 0, il pattern corrotto sarà identico all'originale. Se il rumore è 1, ogni pixel viene generato in modo completamente casuale.

3.3 `EvolvingPattern`

A differenza delle classi di pattern precedenti, che tramite `getPattern()` forniscono un riferimento costante e quindi non modificabile, `EvolvingPattern` deve essere aggiornato per mostrare il cambiamento dinamico. Per questo motivo il suo stato interno è modificabile e la classe fornisce un riferimento non costante ad esso.

3.4 `HopfieldNetwork`

Questa classe è fondamentale in quanto contiene la matrice dei pesi dell'addestramento. In questo paragrafo vengono elencate le formule usate per determinare la matrice dei pesi con i due metodi, quello di Hebb e quello della Pseudoinversa (usando la notazione di Einstein), il calcolo dell'energia e come aggiornare un pattern.

- **Calcolo tramite Hebb:**
  $W_{ij} = \frac{1}{N} \sum_k p_k \otimes p_k^\dagger$, poi si impone $W_{ii} = 0$, dove N è la dimensione del pattern e k è l'indice che scorre sui pattern.

- **Calcolo tramite Pseudoinversa:**
  Dati i pattern, costruisco la matrice X inserendoli come vettori colonna. Per esempio, se ho due pattern $\vec{q} = (q_1, q_2, q_3)$ e $\vec{s} = (s_1, s_2, s_3)$, avrò $X = \begin{pmatrix} q_1 & s_1 \\ q_2 & s_2 \\ q_3 & s_3 \end{pmatrix}$. La formula per calcolare la matrice dei pesi è $W = X(X^\dagger X)^{-1} X^\dagger$, imponendo sempre $W_{ii} = 0$.

- **Calcolo dell'energia:**
  $E = -\frac{1}{2} \Re(x_i^\dagger W_{ij} x_j)$, dove x è la rappresentazione vettoriale dello stato attuale del pattern.

- **Calcolo della differenza di energia:**
  $\Delta E_i = -\Re(\overline{(x_i' - x_i)} \cdot \ell_i)$, dove l'apice indica il nuovo stato e ℓ_i è il valore del campo locale prima della modifica. Il campo locale si calcola come: $\ell_i = \sum_j W_{ij} x_j$.

- **Evoluzione temporale:**
  $x_i(t+1) = \text{sign}\left(\sum_j W_{ij} x_j(t)\right)$. Nel caso dei numeri complessi, il segno va interpretato come il valore (che può essere assunto da un bit) che ha la distanza euclidea minore dal valore aggiornato.

3.5 `CoherenceSetPattern`

Questa classe ha il compito di gestire un set di tre pattern: `ITrainingPattern`, `NoisyPattern` ed `EvolvingPattern`, e di mantenerli consistenti. Ogni volta che l'utente cambia la griglia, la classe si assicura che tutti i suoi elementi abbiano la stessa dimensione. Una modifica del pattern originale si ripercuote sul

`NoisyPattern` (che aggiunge rumore) e di conseguenza sull' `EvolvingPattern` (che parte dal pattern corrotto). Al suo interno è presente la funzione `resolveEvolvingPattern` che permette a `HopfieldNetwork` di evolvere il pattern dinamico.

3.6 `HopfieldSimulator`

Questa è la classe che unisce il tutto, legando un array di `CoherenceSetPattern` e la `HopfieldNetwork`. Infatti, i `CoherenceSetPattern` non possono risolvere i pattern corrotti senza la matrice dei pesi, e la `HopfieldNetwork` non può addestrarsi senza un set di pattern. Questa classe non funge solo da ponte, ma è fondamentale per garantire la consistenza dei processi e ridurre errori. Un altro compito di `HopfieldSimulator` è quello di limitare le chiamate a funzioni computazionalmente onerose da parte di un utente esterno grazie all'uso di `std::mutex`.

4 Classi di `GraphicsManager`

All'interno del file `GraphicsManager.hpp` abbiamo una classe e due struct.

4.1 `GraphicsManager`

La classe `GraphicsManager` ha il compito di occuparsi delle librerie grafiche: inizializzare, gestire e chiudere tutti i processi grafici.

4.2 `Components`

Questa struct ha due funzioni principali:

- `drawGrid`: prende in input un pattern sotto forma vettoriale e lo trasforma in una griglia. Questo serve per la rappresentazione dei tre pattern: `ITrainingPattern`, `NoisyPattern` ed `EvolvingPattern`. La griglia può essere impostata come cliccabile, attivando un'altra funzione: nel nostro caso, il click su una cella chiamerà la funzione `CyclePixelState` che cambia lo stato del pixel premuto.
- `drawPlot`: prende in input un vettore (in questo caso il vettore delle energie) e lo disegna in un grafico.

4.3 `FileDialogHelper`

Questa struct serve ad per aprire i file interattivamente: gestisce il tipo di file che si può aprire, una funzione da chiamare in caso di successo e una funzione che gestisce la chiusura della finestra di dialogo.

5 Librerie usate

5.1 `stb`

Questa libreria è essenziale per la lettura di immagini. In `ImageTrainingPattern`, `stb` permette di trasformare un file immagine in un'immagine integrale (rappresentata da un vettore di interi), rendendo possibile l'addestramento a partire da file grafici.

5.2 `eigen`

Per un calcolo efficiente su oggetti di grandi dimensioni, la libreria `eigen` permette di eseguire in modo ottimizzato calcoli di algebra lineare, accelerando gli algoritmi matematici necessari per l'addestramento, il calcolo dell'energia e la risoluzione dei pattern.

5.3 `imgui` e `ImGuiFileDialog`

`imgui` è stato scelto per la sua versatilità nel creare interfacce utente (slider, pulsanti) in modo rapido. È stata integrata anche la libreria `ImGuiFileDialog` per permettere un accesso semplice al file system.

6 Note aggiuntive

6.1 `main.cpp` e `run_application.hpp`

Il `main` è stato costruito in modo che CMake possa compilare sia la versione con i numeri reali che quella con i numeri complessi. Per questo, il codice principale dell'applicazione si trova in `run_application.hpp`. L'interfaccia è costruita con la libreria ImGui. Per una maggiore sicurezza e per dare un feedback visivo all'utente, ogni pulsante che avvia un'operazione lunga disabilita l'uso di altri pulsanti, sebbene questo flusso sia già gestito in modo sicuro dalla classe `HopfieldSimulator` con i mutex.

6.2 Classi aggiuntive

Nella cartella `HopfieldSimulator` sono presenti tre classi di utilità:

- `Cast.hpp`: dato che lo `static_cast` della libreria standard non supporta la conversione tra `std::complex<int>` e `std::complex<double>`, è stata implementata questa funzionalità, necessaria per i calcoli con i numeri complessi.
- `MathDimension.hpp`: definisce alcune costanti matematiche non presenti nella libreria standard, come i vertici dell'ipercubo usati nelle reti di Hopfield. Per i numeri reali, i vertici sono $\{-1, +1\}$. Per i numeri complessi, sono $\{-1 - i, -1 + i, 1 - i, 1 + i\}$.

- `RandomUtils.hpp`: questa classe fornisce un generatore di numeri casuali il cui seme può essere modificato tramite la funzione `seedGenerator()`. Viene usata sia da `RandomTrainingPattern` che da `NoisyPattern`.

6.3 Template

Nell'intero progetto si è fatto largo uso dei template C++ per rendere il codice valido sia per i numeri reali sia per i numeri complessi. L'uso dei template è necessario per implementare questa dualità. Alcune funzioni possono apparire verbose (per esempio `std::real` su un numero già reale), ma questo è necessario per mantenere un'unica base di codice per tipi diversi.

6.4 Memory leak

Se eseguito in un ambiente virtuale Linux come WSL, il programma potrebbe mostrare dei memory leak che non sono dovuti al codice dell'applicazione ma alla libreria grafica ImGui. Se si compila il programma in modalità DEBUG, questi leak appariranno alla chiusura. Per evitare che questi falsi positivi nascondano eventuali leak reali, consiglio l'uso del comando:

```
export LSAN_OPTIONS=suppressions=./lsan.supp
```

prima di ogni esecuzione. Per rendere questa impostazione permanente, si può usare il comando:

```
echo 'export LSAN_OPTIONS="suppressions=/path/to/your/
    project/lsan.supp"' >> ~/.bashrc
```

Assicurarsi che il file `lsan.supp` sia presente nel progetto. Per evitare del tutto questi memory leak, si consiglia l'uso di un sistema operativo Linux nativo.

7 Installazione e istruzioni

Di seguito riporto la traduzione del file `README.md` del progetto, che potete trovare online nella repository su GitHub.

RETE DI HOPFIELD

In questo progetto, è implementata una Rete di Hopfield che può essere addestrata sia con bit reali che complessi. Il progetto genererà due eseguibili: uno per i numeri reali e uno per i numeri complessi. Con un'interfaccia interattiva è possibile caricare immagini, scegliere il numero di pixel della griglia quadrata, addestrare la rete con l'algoritmo di Hebb o con la Pseudoinversa e osservare come la rete neurale risolve un pattern corrotto.

Istruzioni per la Compilazione, Librerie ed Esecuzione

Per accedere al repository del progetto su GitHub, assicurarsi che Git sia installato sul sistema e poi clonare il repository con il seguente comando:

```
git clone git@github.com:NicolaSebastianuttoUnibo/
    HopfieldNetwork.git
```

Successivamente, all'interno della cartella, scaricare le librerie necessarie dai loro repository GitHub: **stb**

```
git clone git@github.com:nothings/stb.git
```

imgui

```
git clone git@github.com:ocornut/imgui.git
git clone git@github.com:aiekick/ImGuiFileDialog.git
sudo apt update && sudo apt install libsdl2-dev libgl1-mesa-
    dev
```

eigen

```
git clone git@github.com:ceptontech/eigen.git
```

Dopo aver installato le librerie richieste, procedere alla compilazione del programma. Per una build di debug, eseguire il seguente comando:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug
```

Per una build di release, usare questo comando:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
```

Una volta completata la fase di compilazione si può creare l'eseguibile usando:

- per i numeri reali:

  ```
  cmake --build build --target hopfield_real
  ```

- per i numeri complessi:

  ```
  cmake --build build --target hopfield_complex
  ```

- se si vuole compilare i test, eseguire:

  ```
  cmake --build build --target all.t
  ```

Ogni volta che si esegue il programma nella modalità DEBUG, si consiglia di lanciare prima questo comando:

```
export LSAN_OPTIONS=suppressions=./lsan.supp
```

Per rendere questa impostazione permanente, si salva nel sistema con il comando:

```
echo 'export LSAN_OPTIONS="suppressions=/percorso/alla/
    cartella/del/progetto/lsan.supp"' >> ~/.bashrc
```

Infine, per eseguire il codice:

- Per i numeri reali:

  `./build/hopfield_real`

- Per i numeri complessi:

  `./build/hopfield_complex`

- Per i test:

  `./build/all.t`

Comandi

- I parametri possono essere modificati usando gli slider.
- Si può caricare qualsiasi file immagine con `Open Images`. Per selezionare più immagini tenere premuto `SHIFT`.
- Si può anche generare immagini casuali con `Generate Image`. Se il rumore (Noise) è zero, l'immagine casuale sarà tutta nera; se il rumore è 1, sarà un'immagine completamente casuale.
- Se il numero di pattern è piccolo, il ridimensionamento della griglia (regrid) è automatico; altrimenti si ricorda di premere `Apply Grid`.
- Si può decidere quale tipo di addestramento utilizzare, Hebb (più veloce ma meno preciso) o Pseudoinversa (più lento ma più efficace), cliccando sul tipo di addestramento il testo mostrerà quale tipo è selezionato.
- `Train Network` addestrerà la rete.
- `Save Trained Network` aprirà una finestra dove si dovrà scegliere un nome per il file di addestramento.
- `Open Training` aprirà un addestramento già salvato.
- `Stop!` interromperà il processo di addestramento.
- `<` e `>` mostreranno il pattern precedente e successivo.
- `Check Dimensions` verificherà se si può evolvere un pattern controllando la compatibilità delle dimensioni. Se si cambia la griglia per errore, ma si ricorda la dimensione corretta, il programma ti impedisce di elaborare il pattern; questo pulsante permette di procedere se è stata ripristinata la dimensione corretta.

- `Delete` eliminerà il pattern corrente.
- `Corrupt` corromperà il pattern in base al valore di rumore: con noise=0, l'immagine corrotta sarà identica all'originale; con noise=1, l'immagine sarà completamente casuale.
- `Evolve` evolverà il pattern corrotto.
- `Stop` interromperà il processo di evoluzione.
- Il grafico è interattivo, quindi si può passare il mouse sui punti per vederne il valore.