Modello di un neurone

Giulia Fioravanti, Filippo Mauri, Anna Rebecca Sala

1 Introduzione

Il modello di neurone che abbiamo sviluppato ha il seguente funzionamento: prende in input dei dati, sottoforma di punti (x,y), li processa e restituisce come output la classe di appartenenza, che può essere 0 o 1.
Il modello di ciascun neurone si basa sulla classe "neuron" che ha come attributi:

- i pesi (`double Wx, Wy`) e il bias (`double b`), che vengono modificati dal codice per adattarsi al modello e processare i dati.

- una funzione di attivazione (`double (*f) (double)`), in questo progetto utilizziamo la funzione sigmoide $y = \frac{1}{1+e^{-x}}$.

- un type (`int type`) che indica il neurone che si deve attivare per i dati di tipo 0 oppure 1 (nel codice che abbiamo sviluppato sono presenti due neuroni, uno di tipo 0 e uno di tipo 1).

Il neurone è in grado di processare i dati grazie alla funzione (`double output`), che utilizza la funzione di attivazione del neurone, e restituisce il suo stato, ovvero se in risposta all'input si è attivato (1) o meno (0).

```
class neuron
{
  double Wx, Wy;
  double b;
  double (*f) (double);
  int type;

public:
  neuron () { Wx = 0; Wy = 0; b = 0; f = 0; type = -1;}

  neuron (double Wx_, double Wy_, double b_,
          double (*f_) (double), int type_)
    { Wx = Wx_; Wy = Wy_, b = b_; f = f_; type = type_;}

  double get_Wx () {return Wx;}
  double get_Wy () {return Wy;}
  double get_b  () {return b;}
  int get_type  () {return type;}

  double output (double x, double y) {return f (b + Wx * x + Wy * y);}
};
```

2 Training

Per modellare i due neuroni è necessario un processo di training che permetta di modificare i valori dei pesi e dei bias di ognuno dei neuroni, così da allenare i neuroni a riconoscere la categoria.
Nel training si analizzano una serie di punti dei quali si sanno le coordinate spaziali (`x, y`) e la categoria, che nel nostro caso può essere 0 o 1.
Si inizia assegnando valori casuali a pesi e bias, dopo di che il codice scorre tutti i punti più volte cambiando ogni volta il valore dei parametri in modo da minimizzare l'errore di assegnazione della categoria, calcolato con la funziona qui sotto espressa.

```
double error (vector<neuron> NN, Entry data)
{
  double out = 0;
  for (int i = 0; i < NN.size(); i++)
  {
    out += pow (NN[i].output (data.get_x(), data.get_y()) -
           static_cast<double>(data.get_Y()[NN[i].get_type()]), 2);
  }
  return out;
}
```

La minimizzazione dell'errore è possibile grazie alla funzione ChangeParameters, che cambia i parametri con il metodo dello *steepest descent*. Viene trovato un vettore gradiente nello spazio dei parametri che è direzionato verso l'errore minimo. Questo vettore viene moltipolicato per un parametro di learning rate `double t`, che indica di quanto spostarsi nello spazio dei parametri.

```
void ChangeParameters (vector<neuron> &N, Entry data, double t)
{
  for (int i = 0; i < N.size(); i++)
  {
    double dEdb = t * 2 * (N[i].output(data.get_x(), data.get_y()) -
            static_cast<double>(data.get_Y()[N[i].get_type()]))
            * dsigmoid (N[i].get_b() +
            N[i].get_Wx() * data.get_x() + N[i].get_Wy() * data.get_y());

    double bnew = N[i].get_b() - dEdb;
    double Wxnew = N[i].get_Wx() - data.get_x() * dEdb;
    double Wynew = N[i].get_Wy() - data.get_y() * dEdb;

    neuron newN (Wxnew, Wynew, bnew, &sigmoid, N[i].get_type());
    N[i] = newN;
  }
}
```

Più l'errore nell'assegnare la cateoria si fa piccolo (calcolato come l'errore medio tra gli errori dei vari punti), più viene minimizzato anche il learning rate per raggiungere una maggiore precisione dei parametri individuati.

3 Testing

Si passa poi alla fase di testing, in cui i due neuroni, usando i parametri trovati nel training, analizzano dei dati e ne trovano la categoria, in particolare se si attiva il neurone di tipo 0 vuol dire che il dato ha categoria 0, mentre se si attiva il neurone di tipo 1 vuol dire che il dato ha categoria 1.
Tramite un semplice contatore all'interno di una condizione logica `if` verrà calcolara l'efficienza del neurone. In particolare si analizzano gli output dei due neuroni: se l'output del neurone 0 è maggiore dell'output del neurone 1 allora vuol dire che si è attivato il neurone 0 mentre il neurone 1 è rimasto disattivato, e viceversa. Gli output vengono confrontati con le categorie effettive dei punti e se si corrispondono avviene un incremento del contatore.

4 Risultati e Considerazioni

Il processo di allenamento è fatto utilizzando il 70% di un set di dati, mentre il restante 30% viene utilizzato per il processo di testing, in questo modo si avranno dei risultati più attendibili possibili in quanto i neuroni si trovano ad assegnare la categoria a punti nuovi, non incontrati nella fase di allenamento.

I parametri che troviamo, i pesi e il bias, corrispondono a una retta nello spazio

$$Wy * y + Wx * x + b = 0$$

In particolare troviamo che i parametri del neurone di tipo 0 e quelli del neurone di tipo 1 sono opposti. Questo significa che le due rette che si formano sono esattamente la stessa retta. Per questo è possibile realizzare il codice con anche un solo neurone, i codici di training e testing funzionano allo stesso modo e permettono di ottenere la stessa identica retta ottenuta nel codice con due neuroni.

Sia con due neuroni che con un neurone abbiamo ottenuto un'efficienza di 98.3607%. È un ottimo risultato e il fatto che non riesca ad arrivare al 100% può essere dovuto al fatto che il parametro di learning rate non diventi piccolo abbastanza da trovare esattamente i parametri. Avendo un "pool" di dati più grande probabilmente si arriverebbe a una efficienza ancora maggiore.