Introduzione alle Reti Neurali

Stefano Gualandi
Università di Pavia, Dipartimento di Matematica

email: stefano.gualandi@unipv.it
twitter: @famo2spaghi
blog: http://stegua.github.com

Reti Neurali

Terminator 2

John Connor: Can you learn stuff you haven't been programmed with so you could be... you know, more human? And not such a dork all the time?
The Terminator: **My CPU is a neural-net processor; a learning computer**. But Skynet presets the switch to read-only when we're sent out alone.
Sarah Connor: Doesn't want you doing too much thinking, huh?
The Terminator: No.

Introduzione

Le **Reti Neurali**, intese come *tecnologia software*, sono nate indipendentemente da due diversi gruppi di ricerca, di cui uno era motivato dallo studio dei processi di apprendimento nei sistemi biologici, mentre il secondo studiava direttamente degli algoritmi efficienti per l'apprendimento automatico.

Alcuni dati di riferimento

1. Si pensa che il cervello umano abbia circa 10^{11} neuroni

Alcuni dati di riferimento

1. Si pensa che il cervello umano abbia circa 10^{11} neuroni
2. Ogni neurone è collegato in media con altri 10^4 neuroni, e l'attività di un neurone è eccitata o disinibita da queste connessioni

Alcuni dati di riferimento

1. Si pensa che il cervello umano abbia circa 10^{11} neuroni
2. Ogni neurone è collegato in media con altri 10^4 neuroni, e l'attività di un neurone è eccitata o disinibita da queste connessioni
3. Il tempo minimo di attivazione è di 10^{-3} secondi, lento rispetto ai 10^{-10} secondi di un calcolatore

Alcuni dati di riferimento

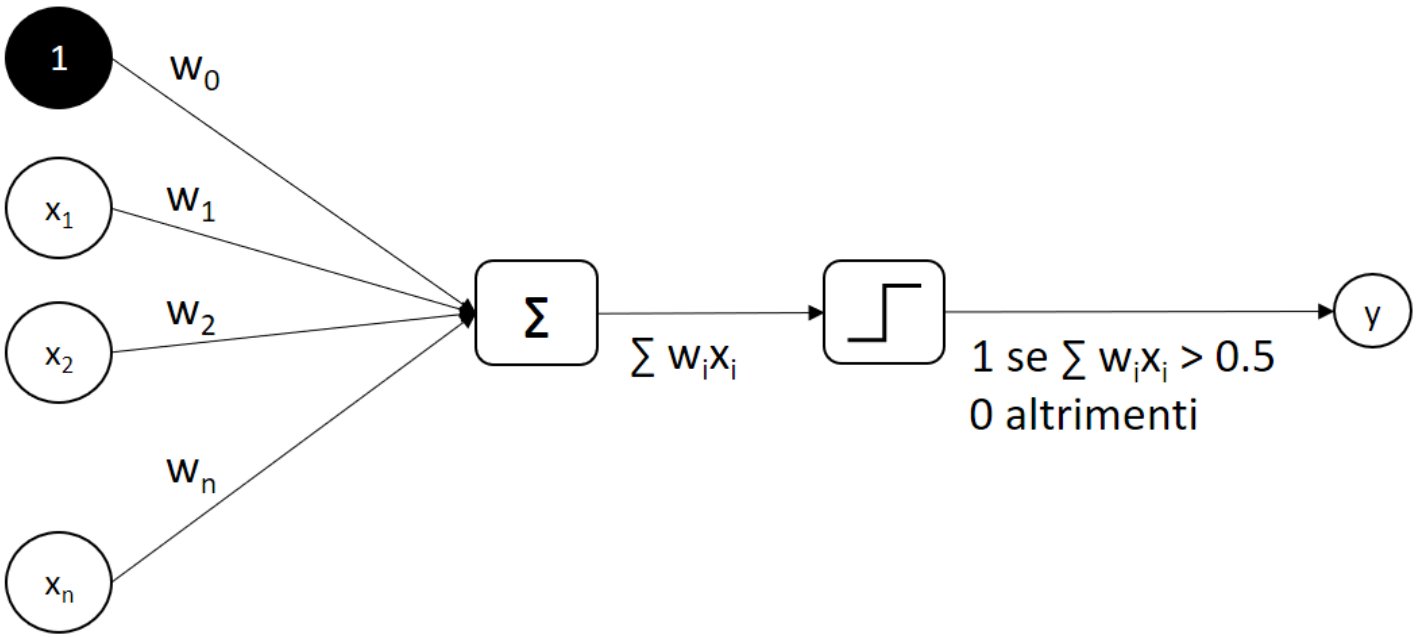
1. Si pensa che il cervello umano abbia circa 10^{11} neuroni
2. Ogni neurone è collegato in media con altri 10^4 neuroni, e l'attività di un neurone è eccitata o disinibita da queste connessioni
3. Il tempo minimo di attivazione è di 10^{-3} secondi, lento rispetto ai 10^{-10} secondi di un calcolatore
4. Il tempo per riconoscere un volto familiare è di 10^{-1} secondi: rapidissimo in confronto al tempi di attivazione di un neurone

Alcuni dati di riferimento

1. Si pensa che il cervello umano abbia circa 10^{11} neuroni
2. Ogni neurone è collegato in media con altri 10^4 neuroni, e l'attività di un neurone è eccitata o disinibita da queste connessioni
3. Il tempo minimo di attivazione è di 10^{-3} secondi, lento rispetto ai 10^{-10} secondi di un calcolatore
4. Il tempo per riconoscere un volto familiare è di 10^{-1} secondi: rapidissimo in confronto al tempi di attivazione di un neurone
5. Deve esserci un forte "parallelismo" nell'elaborazione delle informazioni nel cervello umano

Perceptrons

La rete neurale più semplice è il *perceptron*



L'apprendimento consiste nel trovare i pesi $w \in \mathbb{R}^{p+1}$

Funzioni rappresentabili con un Perceptron

Con un perceptron si possono rappresentare tutte le funzioni che sono linearmente separabili.

Esempi:

- Funzione logica AND: due input, $w_0 = 0.8, w_1 = w_2 = 0.5$
- Funzione logica OR: CHE PESI USARE??
- Funzione logica NAND: CHE PESI USARE??
- Funzione logica XOR: CHE PESI USARE??

Regola di apprendimento per il Perceptron

Il modo più semplice per cercare un vettore dei pesi w è di iniziare con un vettore casuale e poi in maniera iterativa applicare la regola a ciascuno esempio del training set e di modificare i pesi ogni volta che si ha un errore di classificazione. Si potrebbe iterare questo procedimento o un numero prestabilito di volte, oppure sino a quando tutti gli esempi del training set sono classificati in modo corretto.

La regola di aggiornamento è:

$$w_i := w_i + \mu\,(y^{test} - y^{pred})\,x_i \quad (1)$$

$$= w_i + \mu\,(y - f(x, w))\,x_i \quad (2)$$

in cui μ viene chiamato il learning rate.

Apprendimento con il metodo del gradiente

Se prendiamo come funzione di Loss:

$$L(y, f(x, w)) = \frac{1}{2}(y - f(x, w))^2 \tag{3}$$

allora possiamo riscrivere la regola di apprendimento per il perceptron come:

$$w_i := w_i + \mu \frac{\partial L(w)}{\partial w_i} \tag{4}$$

$$= w_i + \mu \sum_{t \in T} (y_t - f(x_t, w)) x_{it} \tag{5}$$

Algoritmo con il metodo del gradiente

Input: Training set T e learning rate μ

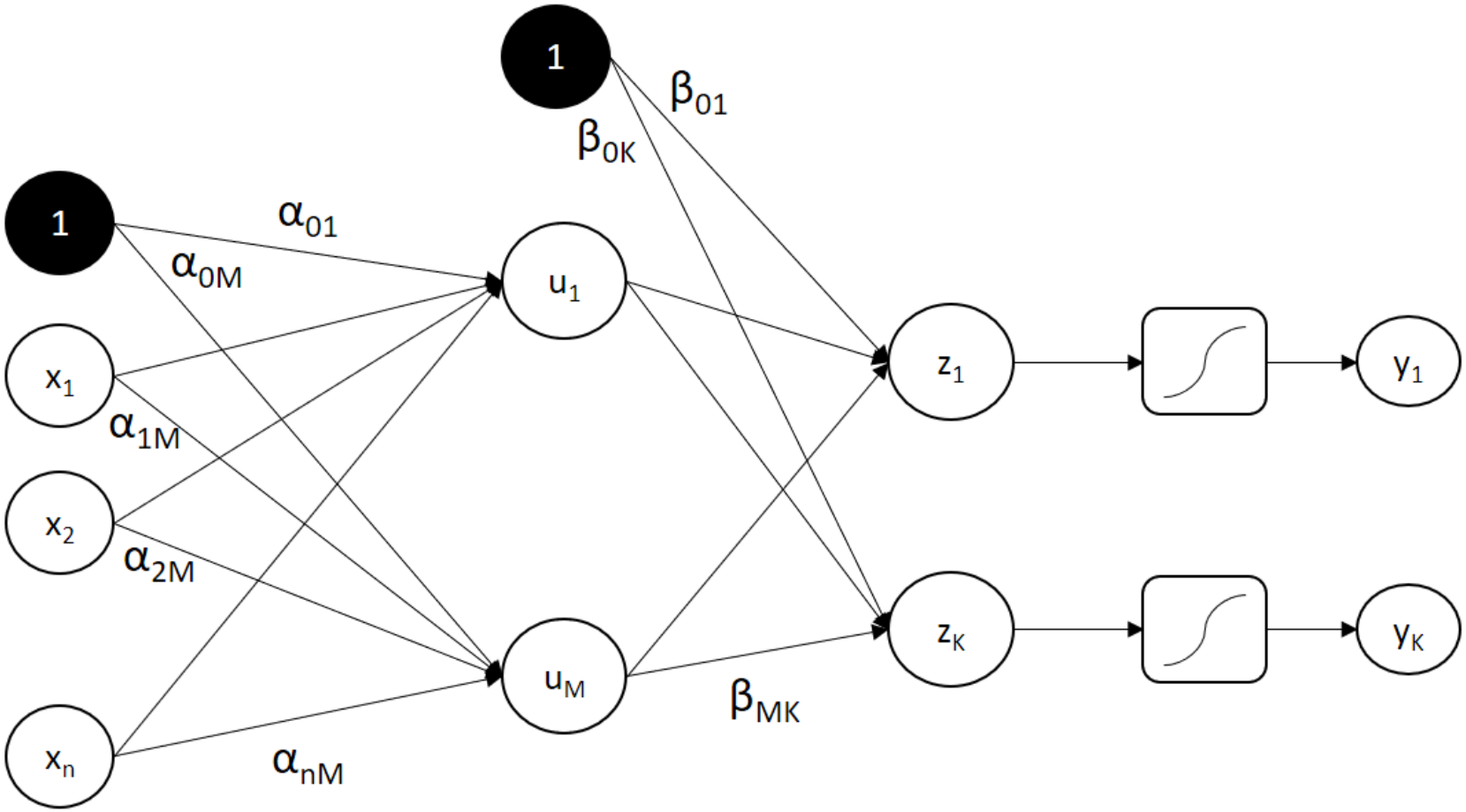
1. Inizializza il vettore w con dei piccoli valori casuali
2. Sino a quando non viene verificato un criterio di arresto:
   1. Inizializza $\Delta w_i = 0$
   2. Per ogni elemento nel training set $t \in T$
      1. Calcola $y_t = f(x_t, w)$
      2. Per ogni elemento del vettore dei pesi:

$$\Delta w_i := \Delta w_i + \mu(y_t - f(x_t, w))x_i \tag{6}$$

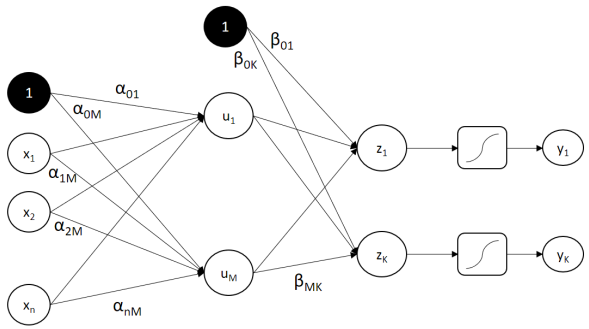
   3. Per ogni elemento di w:

$$w_i := w_i + \Delta w_i \tag{7}$$

Multi Layer Networks



Multi Layer Networks



Nelle reti neurali a due livelli, per un problema di classificazione a K classi, con M neuroni nel livello nascosto, abbiamo che

$$u_m = \sigma(\alpha_{0m} + \alpha_m^T x), \quad m = 1, \ldots, M \tag{8}$$

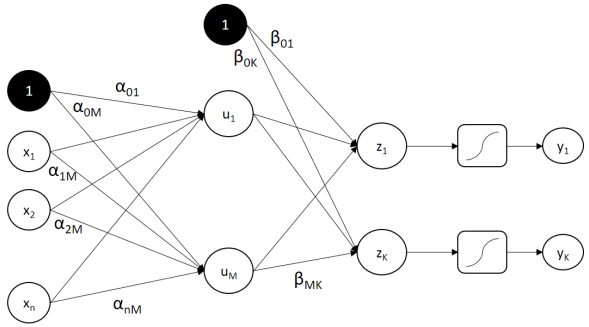
$$z_k = \beta_{0k} + \beta_k^T u, \quad k = 1, \ldots, K \tag{9}$$

$$f_k(x) = g_k(z), \quad k = 1, \ldots, K \tag{10}$$

in cui $u = (u_1, \ldots, u_M)$, $z = (z_1, \ldots, z_K)$, e $\sigma(v)$ è una funzione di attivazione, di solito posta pari a

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

Funzione di output



La funzione di ouput $g_k(z)$ permette una trasformazione finale del vettore degli output z.

Per i problemi di regressione si usa di solito la funzione identità $g_k(z) = z_k$.

Per i problemi di classificazione si usa la stessa funzione *softmax* usata per la regressione logistica

$$g_k(z) = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (11)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (12)$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (11)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (12)$$

Per i problemi di regressione si usa come *Loss function*

$$L(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Fitting di Reti a due strati: Parametri

Denotiamo l'insieme dei pesi di una rete neurale con θ, che consiste di

$$\alpha_{0m}, \alpha_m; m = 1, 2, \ldots, M : \text{sono } M(p+1) \text{ pesi} \quad (11)$$

$$\beta_{0k}, \beta_k; k = 1, 2, \ldots, K : \text{sono } K(M+1) \text{ pesi} \quad (12)$$

Per i problemi di regressione si usa come *Loss function*

$$L(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2.$$

Per i problemi di classificazione, oltre alla precedente, si usa la Loss function

$$L(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i)$$

e il classificatore restituisce $G(x) = \arg\max f_k(x_i)$.

Fitting di Reti a due strati: Back-propagation

L'approccio più generale per minimizzare la Loss function è attraverso metodi del gradiente, che in questo contesto vengono chiamate di Back-propagation, che nel caso di errore quadratico medio viene calcolato come segue.

$$L(\theta) = \sum_{i=1}^{N} L_i = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \tag{13}$$

con derivate

$$\frac{\partial L_i}{\partial \beta_{km}} = -2\left(y_{ik} - f_k(x_i)\right) g'(\beta_k^T u_i) u_{mi}, \tag{14}$$

$$\frac{\partial L_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g'(\beta_k^T u_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}. \tag{15}$$

Fitting di Reti a due strati: Back-propagation

Date le derivate prima precedenti, le regole di aggiornamento per un metodo di gradiente sono

$$\beta_{km}^{r+1} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \beta_{km}^{(r)}} \tag{16}$$

$$\alpha_{ml}^{r+1} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial L_i}{\partial \alpha_{ml}^{(r)}} \tag{17}$$

in cui γ_r è il *learning rate*.

Fitting di Reti a due strati: Back-propagation

Se ora scriviamo le derivate parziali come

$$\frac{\partial L_i}{\partial \beta_{km}} = \delta_{ki} u_{mi}, \tag{18}$$

$$\frac{\partial L_i}{\partial \alpha_{ml}} = s_{mi} x_{il}. \tag{19}$$

le quantità δ_{ki} e s_{mi} sono gli *errori* del modello corrente allo strato nascosto e a quello di output. Possiamo scrivere

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki},$$

che viene chiamata l'equazione di *back-propagation*.

Fitting di Reti a due strati: Back-propagation

Il calcolo dei pesi θ viene eseguito con un algoritmo che ha due fasi.

1. Nella prima fase (*forward pass*) si calcolano i valori predetti di $\hat{f}(x_i)$ con l'equazioni che definiscono la rete.
2. Nella seconda fase (*backward pass*) si calcolano gli errori δ_{ki} che vengono poi propagati all'indietro per trovare gli errori s_{mi}.
3. Alla fine gli errori δ_{ki} e s_{mi} sono usati per calcolare i gradienti per le regole di aggiornamento.

I vantaggi di questo metodo sono la semplicità e il fatto di essere "locale": ogni neurone passa e riceve informazioni solo dai neuroni con cui condivide una connessione, e quindi si presta a metodi di calcolo parallelo.

Fitting di Reti a due strati: Learning rate

Il learning rate γ_r di solito viene preso costante (determinato in maniera empirica).

In teoria per garantire convergenza, si dovrebbe avere

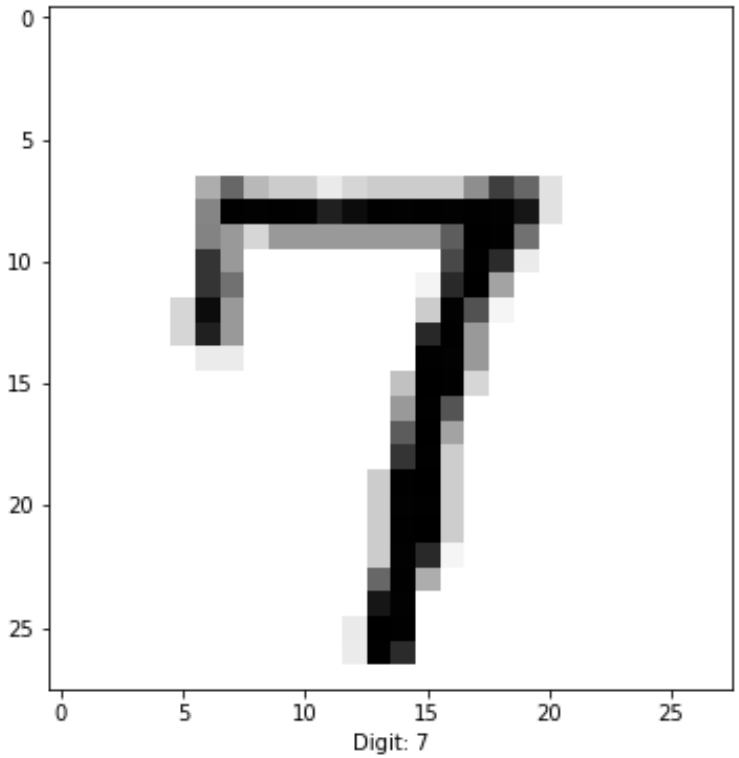
$$\gamma_r \rightarrow 0, \tag{20}$$

$$\sum_r \gamma_r = \infty, \tag{21}$$

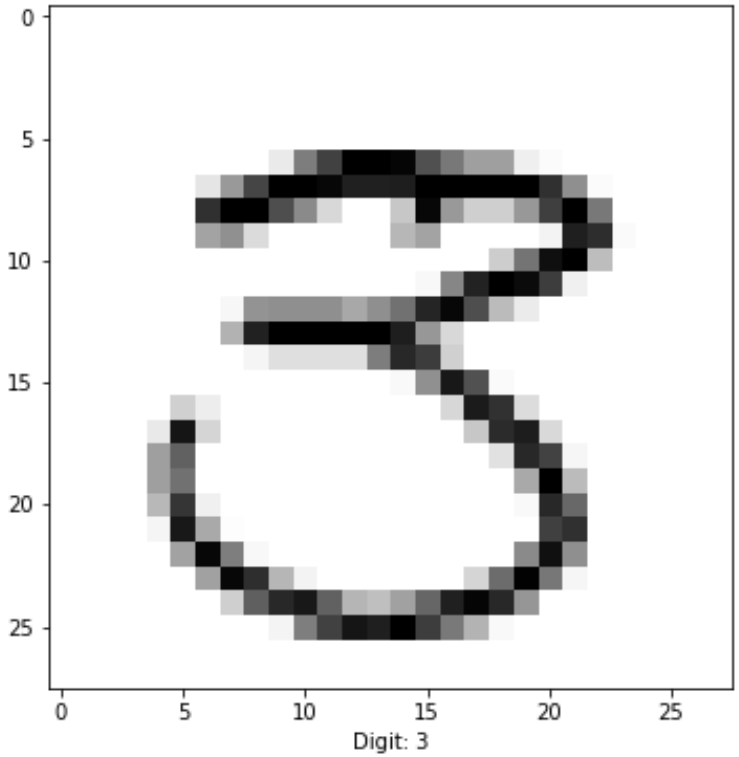
$$\sum_r \gamma_r^2 < \infty. \tag{22}$$

Queste richieste vengon soddisfatte per esempio da $\gamma_r = \frac{1}{r}$

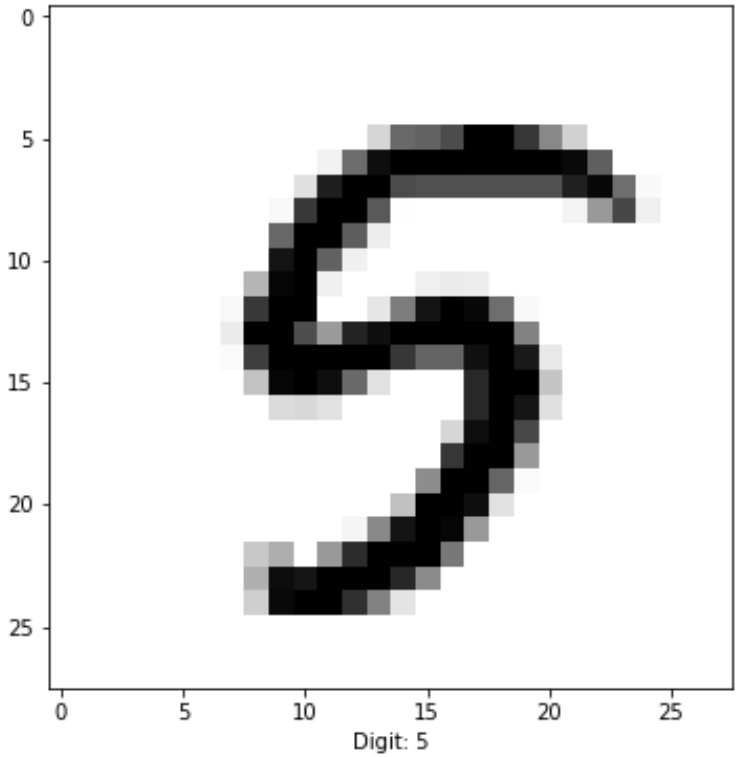
Riconoscimento automatico di codici postali



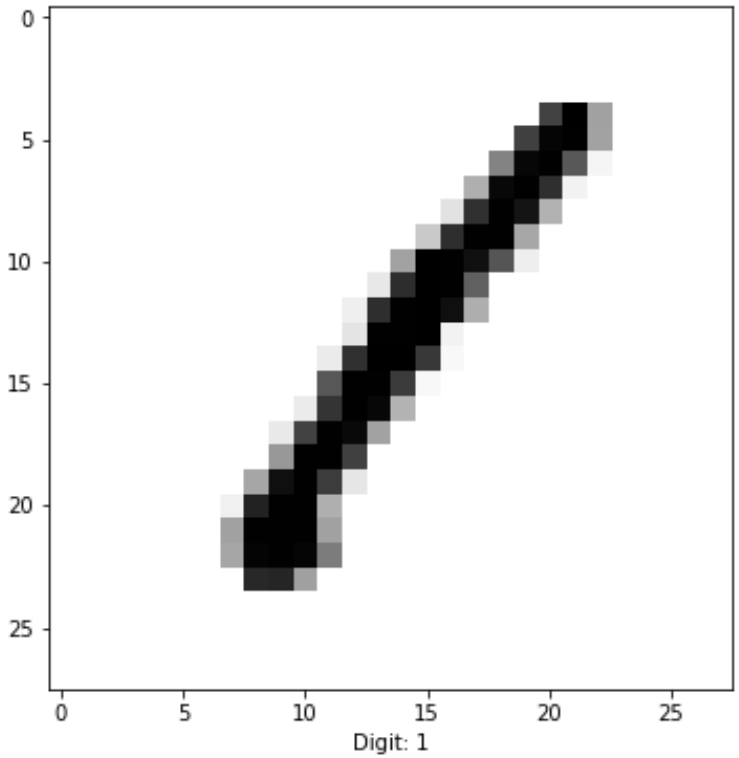
Riconoscimento automatico di codici postali



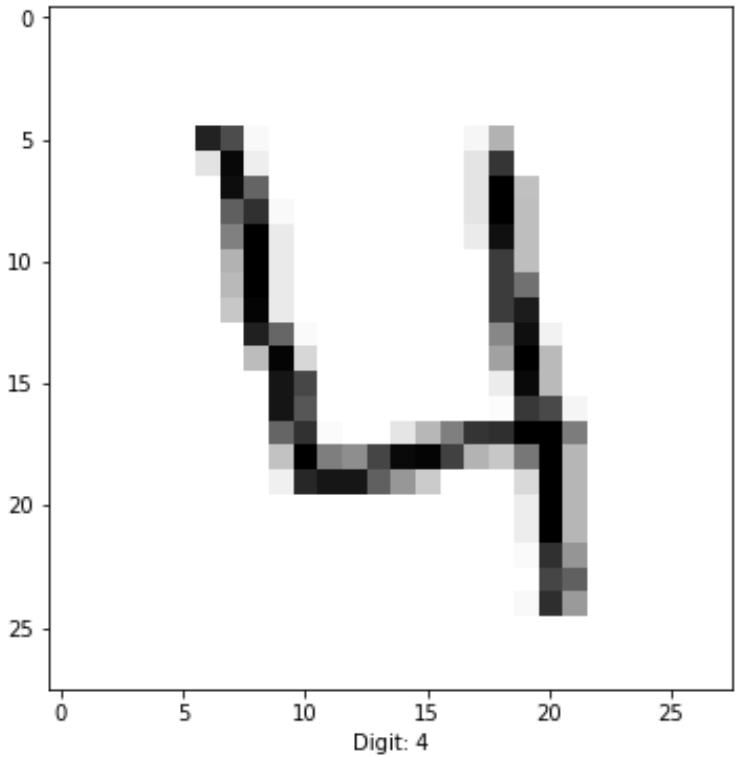
Riconoscimento automatico di codici postali



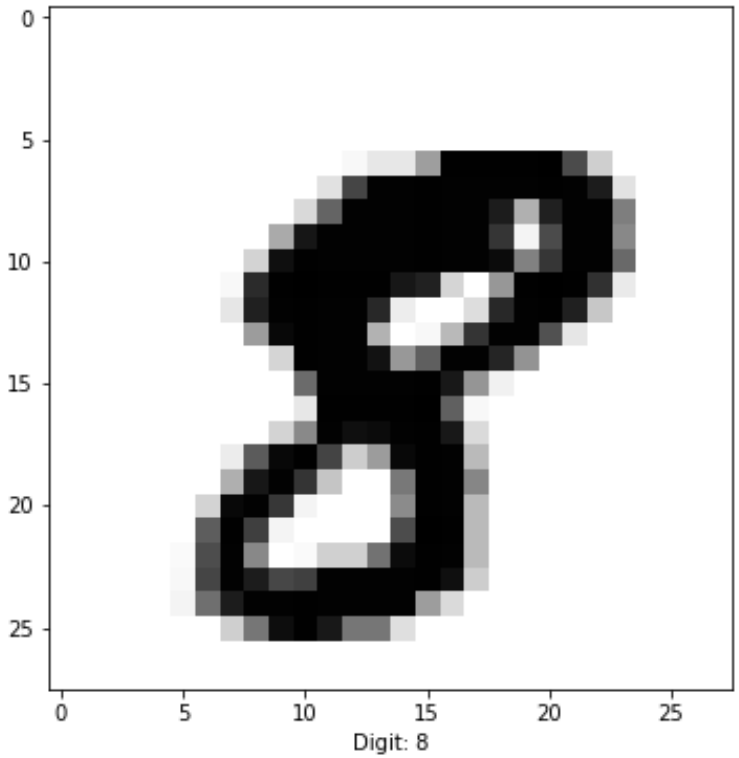
Riconoscimento automatico di codici postali



Riconoscimento automatico di codici postali



Riconoscimento automatico di codici postali



Riconoscimento automatico di codici postali