

Clausola di Horn: proposizione in cui valgono le restrizioni:

I. Non sono ammessi quantificatori esistenziali (\exists)

II. I quantificatori universali, se presenti, sono presenti solo nella parte iniziale della proposizione;

III. La parte di proposizione che segue i quantificatori universali è formata unicamente da disgiunzione di letterali;

IV. Non più di un letterale può essere positivo

$$\forall x_1, \forall x_2, \dots, \forall x_n (L_1 \vee L_2 \vee \dots \vee L_n)$$

Una clausola si dice definita se ammette un letterale positivo.

Una clausola si dice goal se non ammette letterali positivi.

Una teoria T formata unicamente da clausole di Horn, se esistono due clausole

$$\forall x_1, \forall x_2, \dots, \forall x_n (L_1 \vee L_2 \vee \dots \vee L_n)$$

$$\forall y_1, \forall y_2, \dots, \forall y_n (M_1 \vee M_2 \vee \dots \vee M_n)$$

che non condividono variabili e tali che esistano $1 \leq i \leq n$ e $1 \leq j \leq s$ per cui $L_i \theta = \neg M_j \theta$, con $\theta = \text{mgu}(L_i, M_j)$, allora è possibile costruire la clausola risolvente R di $L_i \wedge M_j$

Teorema di approssimazione universale

Dato $n \in \mathbb{N}_+$, sia $D \subset \mathbb{R}^n$ compatto e sia $f: S \rightarrow \mathbb{R}$ una f.ne reale a più variabili continua nell'aperto $S \subseteq \mathbb{R}^n$ con $D \subset S$.

Sia $g: \mathbb{R} \rightarrow \mathbb{R}$ una f.ne continua, monotona, strettamente crescente e limitata.

Per ogni $\epsilon \in \mathbb{R}$, esistono:

- $m \in \mathbb{N}_+$, numero neuroni del livello nascosto;
- v_j, θ_j , con $1 \leq j \leq m$, pesi e bias dei neuroni
- $w_{j,i} \in \mathbb{R}$, con $1 \leq i \leq n$ e $1 \leq j \leq m$, i pesi associati alle connessioni tra input e i neuroni del livello nascosto;

tali che la f.ne

$$h(x) = \sum_{j=1}^m v_j g\left(\sum_{i=1}^n w_{j,i} x_i - \theta_j\right)$$

soddisfi

$$\max_{x \in D} |f(x) - h(x)| < \epsilon$$

Spiegazione semplificata

Il teorema dice che possiamo approssimare qualsiasi f.ne continua f su un insieme compatto D usando una rete neurale di un solo strato nascosto con f.ne di attivazione g .

La f.ne di attivazione g è continua, monotona e limitata (sigmoide o ReLU).

Gli input x vengono combinati linearmente usando i pesi $w_{j,i}$ e θ_j .

Il risultato viene passato attraverso la f.ne di attivazione.

I risultati vengono combinati linearmente con i pesi di v_j .

Algoritmi dei vincoli

G & T

```
void solveGT(int index) {
    if (index == n)
        if (areConstraintsViolated())
            solutionFound();
        else
            if (variable[index] != FREE)
                solveGT(index + 1);
            else {
                for (int i = 0; i < d; i++)
                    variable[index] = dom[i];
                solveGT(index + 1);
            }
    variable[index] = FREE;
}
```

SPIEGAZIONE

La f.ne prende in input un indice $index$ che rappresenta la posizione corrente nella lista delle variabili. L'algoritmo procede in modo ricorsivo analizzando tutte le variabili nella lista.

Se l'indice è uguale al numero di variabili (n), significa che tutte le variabili sono state analizzate e, in questo caso, controlla se le constraint sono soddisfatte.

Se lo sono: l'alg. ha trovato una soluzione e lo "ritorna";

Se non lo sono: non esiste una soluzione per il problema con le assegnazioni fatte fin'ora.

Allora, se il valore della variabile corrente è diverso da FREE, significa che ha già un valore assegnato, in questo caso l'alg. passa alla variabile successiva, chiamando ricorsivamente solveGT.

Se il valore della corrente variabile è FREE, l'alg. prova ad assegnare uno per uno tutti i valori possibili; per ogni valore assegnato l'alg. passa alla variabile successiva tramite la chiamata ricorsiva.

Se tutte le assegnazioni possibili non portano ad una soluzione, il valore della variabile viene resettato a FREE.

Standard backtracking

```

void solveSBT (int index) {
    if (index == n)
        solutionFound();
    else
        if (variable [index] != FREE)
            solveSBT(index + 1);
        else {
            for (int i = 0; i < d; i++)
                variable [index] = dom [i];

            if (areConstraintsViolated())
                solveSBT(index + 1);
        }
    variable [index] = FREE;
}

```

Standard Backtracking

```

function BACKTRACKING-SEARCH (csp) return solution/fail
    return RECURSIVE-BACKTRACKING ({}, csp)
function RECURSIVE-BACKTRACKING (assignment, csp) return solution/fail
    if assignment == complete, return assignment
    var ← SELECT-UNASSIGNED-VARIABLE (variables(csp), assignments, csp)
    for each value in ORDER-DOMAIN-VALUES (var, assignment, csp)
        do
            if value is consistent with assignment according to Constraint [csp]
            then
                add {var = value} to assignment
                result ← RECURSIVE-BACKTRACKING (assignment, csp)
                if result != fail, return result
                remove {var = value} from assignment
    return fail
}

```

SPIEGAZIONE

L'obiettivo dell'algoritmo è trovare un'assegnazione di valori alle variabili che soddisfi tutte le constraint del problema.

Funzione BACKTRACKING-SEARCH: il parametro csp è il problema da risolvere, la f.n. ritornerà l'eventuale soluzione oppure fail.

La f.n. chiama ricorsivamente RECURSIVE-BACKTRACKING per trovare la soluzione.

RECURSIVE-BACKTRACKING prende in input un'assegnazione parziale e csp, ritorna una soluzione, se esiste o fail.

Se l'assegnazione parziale è completa, significa che tutte le variabili hanno un valore assegnato, in questo caso la f.n. ritorna l'assegnamento parziale.

Altrimenti la f.n. seleziona una var tra quelle non ancora assegnate tra gli assegnamenti parziali.

La scelta si basa sulle euristiche.

Per ogni valore possibile da assegnare a var, la f.n. esegue:

- aggiunge value in var nell'assegnazione parziale;
- chiama ricorsivamente RECURSIVE-BACKTRACKING con la nuova assegnazione parziale. Se la f.n. restituisce una soluzione, significa che è stata trovata una sol. completa; se la f.n. ritorna fail, significa che l'assegnazione parziale con value non porta a soluzione. La f.n. rimuove value da var nell'assegnazione parziale.

Se il ciclo termina senza trovare una soluzione per nessun valore possibile di var, significa che non esiste una soluzione al problema con le assegnazioni fatte fino ad ora. La f.n. ritorna fail.

CSP

Constraint Satisfaction Problem

$\langle V, D, C \rangle$

- V è un insieme non vuoto e finito di simboli detti variabili con $n = |V|$
- D è un insieme non vuoto di insiemi detti domini delle variabili con $|D| \leq n$
- C è un insieme finito di vincoli

Il dominio del CSP \mathcal{P} $\text{dom}(\mathcal{P}) = \prod_{x \in V} \text{dom}(x)$

Ogni vincolo $c \in C$ è una coppia $\langle V_c, \Delta_c \rangle$

· $V_c \subseteq V$ insieme delle variabili del vincolo

· Δ_c è l'insieme degli assegnamenti parziali del vincolo c .

$$\Delta_c = \prod_{x \in V_c} \text{dom}(x)$$

Dato un CSP $\mathcal{P} = \langle V, D, C \rangle$, l'insieme delle soluzioni di \mathcal{P} è:

$$\text{sol}(\mathcal{P}) = \bigcap_{c \in C} \text{img}(c)$$

Sostituzione e Sostituzione composta.

Dato un insieme T di termini costruito su un insieme di atomi A e un insieme di variabili V , una sostituzione è un insieme finito ed eventualmente vuoto della forma:

$$\left\{ \frac{t_1}{x_1}, \frac{t_2}{x_2}, \frac{t_3}{x_3}, \dots, \frac{t_n}{x_n} \right\}$$

dove:

- x_1, \dots, x_n sono variabili
- t_1, \dots, t_n sono termini
- Per ogni $1 \leq i \leq n$, $t_i \neq x_i$
- Per ogni $1 \leq i \leq n$ e $1 \leq j \leq n$, se $i \neq j$ allora $x_i \neq x_j$

Date due sostituzioni $\theta = \left\{ \frac{t_1}{x_1}, \frac{t_2}{x_2}, \dots, \frac{t_n}{x_n} \right\}$ e $\sigma = \left\{ \frac{u_1}{y_1}, \frac{u_2}{y_2}, \dots, \frac{u_n}{y_n} \right\}$, la sostituzione composta $\theta \circ \sigma$ si ottiene dal seguente insieme:

$$\left\{ \frac{t_1 \sigma}{x_1}, \frac{t_2 \sigma}{x_2}, \dots, \frac{t_n \sigma}{x_n} \right\}$$

eliminando tutti gli elementi

- $t_j \sigma / x_j$ se vale $t_j \sigma = x_j$;
- u_j / y_j se $y_j \in \{x_1, x_2, \dots, x_n\}$

MGU

Most General Unifier è la sostituzione più generale che unifica un insieme di espressioni.

Una sostituzione θ è il MGU di un insieme di termini T_1, T_2, \dots, T_n se:

- $T_1 \theta = T_2 \theta = \dots = T_n \theta$, cioè θ unificatore
- Per ogni altro unificatore σ di T_1, T_2, \dots, T_n esiste una sostituzione τ tale che $\sigma = \theta \circ \tau$

Prolog

I. Dato un programma π scritto in Prolog, e un goal r , la f.ne non deterministica σ^π può essere esplicitata così:

$$\sigma^\pi(v) = \begin{cases} \sigma_G^\pi(g, \pi) & \text{se } g \text{ head}(v) \wedge \perp = \text{rest}_G(v) \\ \theta \circ \sigma^\pi(\neg r \theta) & \text{se } g \text{ head}(v) \wedge r = \text{rest}_G(v) \wedge r \neq \perp \wedge \theta = \sigma_G^\pi(g, \pi) \end{cases}$$

II. Se il goal è formato da un unico congiunto, allora si usa la f.ne non deterministica σ_6^π :

$$\sigma_6^\pi(g, \pi) = \begin{cases} \sigma_F^\pi(g, c) & \text{se } c = \text{head}_p(\pi) \wedge c = h \quad \text{comportamento della f.ne se la prima clausola è definita un fatto.} \\ \sigma_R^\pi(g, c) & \text{se } c = \text{head}_p(\pi) \wedge c = h:-b \quad \text{comportamento della f.ne se la prima clausola è definita una regola.} \\ \sigma_G^\pi(g, r) & \text{se } r = \text{rest}_p(\pi) \wedge r \neq \perp \quad \text{comportamento della f.ne se non sono state elaborate tutte le clausole definite dal programma.} \end{cases}$$

La f.ne σ_F^π prende come primo argomento un congiunto di un goal e come secondo argomento un fatto:

$$\sigma_F^\pi(g, f) = \theta \quad \text{se } f^1 = h^1 \wedge \theta = \text{mgu}\{\{g, h^1\}\} \wedge \theta \neq \perp$$

La f.ne σ_R^π riceve come primo argomento un congiunto di un goal e come secondo argomento una regola:

$$\sigma_R^\pi(g, r) = \theta \circ \sigma^\pi(\neg b \theta) \quad \text{se } r^1 = h^1:-b^1 \wedge \theta = \text{mgu}\{\{g, h^1\}\} \wedge \theta \neq \perp$$

DFS

node (a).
node (b).
node (c).
node (d).
node (f).
node (g).
node (h).
node (i).
node (j).

BFS

node (a).
node (b).
node (c).
node (d).
node (f).
node (g).
node (h).
node (i).
node (j).

```

node (k).
arc (a,b).
arc (a,c).
arc (b,e).
arc (b,f).
arc (b,g).
arc (c,h).
arc (c,i).
arc (c,j).
arc (i,k).

```

```

concat ([] , L , L).
concat ([H|R] , L , [H|NEWRESULT]) :-  

    concat (R , L , NEWRESULT).

```

```

reverse_([],[]).
reverse_([H|R],RESULT) :-  

    reverse_(R,RREVERSE),  

    concat(RREVERSE,[H],RESULT).

```

```

nonmember ( _ , [ ] ) :- ! .  

nonmember ( x , [ x | _ ] ) :-  

    ! ,  

    fail.  

nonmember ( x , [ _ | R ] ) :-  

    nonmember ( x , R ) .

```

```

getchildrenrec ( X , Children , Result ) :-  

    arc ( X , Y ) ,  

    nonmember ( Y , Children ) ,  

    getchildrenrec ( X , [ Y | Children ] , Result ) ,  

    ! .  

getchildrenrec ( X , Children , Children ) :-  

    arc ( X , Y ) ,  

    member ( Y , Children ).  

getchildrenrec ( _ , Children , Children ) .

```

```

getchildren ( X , Result ) :-  

    getchildrenrec ( X , [ ] , Result ) .

```

```

dfsrec ([] , VisitedNodes , VisitedNodes).
dfs rec ([ currentNode | Node ] , VisitedNodes , Result ) :-  

    getchildren ( currentNode , Children ) ,  

    concat ( Children , Nodes , VisitQueue ) ,  

    dfsrec ( VisitQueue , [ currentNode | VisitedNodes ] , Result ) .

```

```

dfs ( Source , NodeList ) :-  

    node ( Source ) ,  

    dfsrec ( Source , [] , NodeList ) .

```

Teorema di Hopfield + Dimostrazione

Teorema di convergenza delle Reti Neurali di Hopfield: Si consideri una rete neurale di Hopfield formata da $n \in \mathbb{N}_+$ neuroni e descritta da una matrice dei pesi w e da un vettore di bias b . Se la matrice dei pesi è simmetrica, allora indipendentemente dallo stato iniziale, tutti i neuroni della rete tenderanno ad uno stato stazionario che corrisponde ad un minimo locale dell'energia della rete.

$$E(s) = -\frac{1}{2} s^T w s + b s$$

```

node (k).
arc (a,b).
arc (a,c).
arc (b,e).
arc (b,f).
arc (b,g).
arc (c,h).
arc (c,i).
arc (c,j).
arc (i,k).

```

```

concat ([] , L , L).
concat ([H|R] , L , [H|NEWRESULT]) :-  

    concat (R , L , NEWRESULT).

```

```

reverse_([],[]).
reverse_([H|R],RESULT) :-  

    reverse_(R,RREVERSE),  

    concat(RREVERSE,[H],RESULT).

```

```

nonmember ( _ , [ ] ) :- ! .  

nonmember ( x , [ x | _ ] ) :-  

    ! ,  

    fail.  

nonmember ( x , [ _ | R ] ) :-  

    nonmember ( x , R ) .

```

```

getchildrenrec ( X , Children , Result ) :-  

    arc ( X , Y ) ,  

    nonmember ( Y , Children ) ,  

    getchildrenrec ( X , [ Y | Children ] , Result ) ,  

    ! .  

getchildrenrec ( X , Children , Children ) :-  

    arc ( X , Y ) ,  

    member ( Y , Children ).  

getchildrenrec ( _ , Children , Children ) .

```

```

getchildren ( X , Result ) :-  

    getchildrenrec ( X , [ ] , Result ) .

```

```

dfsrec ([] , VisitedNodes , VisitedNodes).
dfs rec ([ currentNode | Node ] , VisitedNodes , Result ) :-  

    getchildren ( currentNode , Children ) ,  

    concat ( Node , Children , VisitQueue ) ,  

    dfsrec ( VisitQueue , [ currentNode | VisitedNodes ] , Result ) .

```

```

dfs ( Source , NodeList ) :-  

    node ( Source ) ,  

    dfsrec ( Source , [] , NodeList ) .

```

Dim: se $s \in \mathbb{R}^n$ è lo stato della rete, $s' \in \mathbb{R}^n$ sarà lo stato successivo, allora la variazione di energia sarà: $\Delta E = E(s') - E(s)$.

$$\begin{aligned}\Delta E &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{j,i} \cdot s_i \cdot s_j + \sum_{i=1}^n b_i \cdot s_i - \left(-\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{j,i} \cdot s_i \cdot s_j + \sum_{i=1}^n b_i \cdot s_i \right) \\ &= -\frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n w_{j,i} \cdot s_i \cdot s_j - \sum_{i=1}^n \sum_{j=1}^n w_{j,i} \cdot s_i \cdot s_j \right) + \sum_{i=1}^n b_i (s_i - s_i) \\ &= -\frac{1}{2} \left[\sum_{i=1}^n \sum_{j=1}^n w_{j,i} (s_i \cdot s_j - s_i \cdot s_j) \right] + \sum_{i=1}^n b_i (s_i - s_i) \\ &\quad \xrightarrow{\text{Quando la differenza } \neq 0 \text{?}} \text{Le reti di Hopfield vengono simulate a tempo discreto, aggiornando l'uscita di un neurone casualmente scelto per ogni istante di simulazione, e quindi quella differenza sarà } \neq 0 \text{ quando: } (i=a \wedge j \neq a) \vee (i \neq a \wedge j=a) \\ &\quad \text{Quindi fissato un istante } i \leq a \leq n, \text{ si può riscrivere } \Delta E \text{ così:} \\ &= -\frac{1}{2} \left[\sum_{i=1}^n w_{a,i} \cdot s_i \cdot s_a + \sum_{j=1}^n w_{j,a} \cdot s_j \cdot s_a - \sum_{i=1}^n w_{a,i} \cdot s_i \cdot s_a + \sum_{j=1}^n w_{j,a} \cdot s_j \cdot s_a \right] + b_a (s_a - s_a) \quad \text{Per ipotesi, la matrice dei pesi è simmetrica quindi per } i \neq a \\ &= -\frac{1}{2} \left[2 \sum_{i=1}^n w_{a,i} \cdot s_i \cdot s_a - 2 \sum_{i=1}^n w_{a,i} \cdot s_i \cdot s_a \right] + b_a (s_a - s_a) \\ &= \left[s_a \sum_{i=1}^n w_{a,i} \cdot s_i - s_a \sum_{i=1}^n w_{a,i} \cdot s_i \right] + b_a (s_a - s_a) \\ &= -(s_a - s_a) \left(\sum_{i=1}^n w_{a,i} \cdot s_i \right) + b_a (s_a - s_a) \\ &= -(s_a - s_a) \left(\sum_{i=1}^n w_{a,i} - b_a \right)\end{aligned}$$

casi:

- 1) Se $s'_a = s_a \vee a=0$, punto stazionario
- 2) Se $s'_a = 1 \Rightarrow a=1 \Rightarrow \Delta E < 0$ e $E(s') < E(s)$
- 3) Se $s'_a = -1 \Rightarrow a=-1 \Rightarrow \Delta E < 0$ e $E(s') < E(s)$

Teorema di Addestramento con la regola di Hebb + Dimostrazione

Data una rete neurale di Hopfield con $n \in \mathbb{N}$, neuroni, si consideri un training set $T = \{x_k\}_{k=1}^m$ formato da $m \in \mathbb{N}_+$ vettori di \mathbb{R}^n tra loro mutuamente ortogonali e privi di componenti nulle, se la rete ha vettore di bias nullo e matrice dei pesi:

$$W = \frac{1}{m} \sum_{K=1}^m x_k x_k^T - I_n, \quad \text{dove } I_n \text{ è la matrice identità } n \times n, \text{ allora l'energia della rete ammette minimi locali in corrispondenza dei vettori del training set.}$$

Dim:

Le singole componenti della matrice W sono definite:

$$w_{i,j} = \frac{1}{m} \sum_{k=1}^m x_{k,i} \cdot x_{k,j} - \delta_{i,j} \quad \text{con } \delta_{i,j} = \begin{cases} 1 & \text{se } i=j \\ 0 & \text{altrimenti} \end{cases}$$

Notiamo che le componenti di W sulla diagonale sono nulle dato che $|x_{i,j}| = 1$

$$w_{s,s} = \frac{1}{m} \sum_{k=1}^m x_{k,i} \cdot x_{k,i} - \delta_{i,i} = \frac{1}{m} \sum_{k=1}^m x_{k,i} \cdot x_{k,i} - \delta_{i,i} = w_{j,i}$$

Consideriamo la derivata dell'energia di un istante di tempo $s \in \mathbb{R}^n$ rispetto alla sua componente s_a , $1 \leq a \leq n$

$$\begin{aligned}\frac{\partial E(s)}{\partial s_a} &= \frac{\partial}{\partial s_a} \left[-\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{i,j} \cdot s_i \cdot s_j \right] + \sum_{i=1}^n b_i s_i = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \frac{\partial [w_{i,j} s_i s_j]}{\partial s_a} + \sum_{i=1}^n b_i \frac{\partial s_i}{\partial s_a} = -\frac{1}{2} \left[\sum_{i=1}^n \sum_{j=1}^n w_{i,j} \cdot \frac{\partial (s_i s_j)}{\partial s_a} \right] + b_a s_a = \\ &= -\frac{1}{2} \left[\sum_{i=1}^n \sum_{j=1}^n w_{i,j} s_j \frac{\partial s_i}{\partial s_a} + \sum_{i=1}^n \sum_{j=1}^n w_{i,j} s_i \frac{\partial s_j}{\partial s_a} \right] + b_a s_a \\ &= -\frac{1}{2} \left[\sum_{j=1}^n w_{a,j} s_j + \sum_{i=1}^n w_{i,a} s_i \right] + b_a s_a \quad \text{Ma essendo } W \text{ simmetrica e, per ipotesi, } b \text{ è nullo} \\ &= -\frac{1}{2} \cdot 2 \sum_{i=1}^n w_{a,i} \cdot s_i = -\sum_{i=1}^n w_{a,i} \cdot s_i\end{aligned}$$

Verificare che la rete ammette minimi locali dell'energia in corrispondenza dei vettori del training set.

Considero $x_n \in \mathbb{R}^n$ un vettore del training set, allora: $\frac{\partial E(x_n)}{\partial s_a} = 0$

$$\begin{aligned}\frac{\partial E(x_n)}{\partial s_a} &= -\sum_{i=1}^n w_{a,i} \cdot s_i = -\sum_{i=1}^n \left[\frac{1}{m} \sum_{k=1}^m x_{k,a} x_{k,i} - \delta_{a,i} \right] \cdot x_{n,i} = \\ &= -\frac{1}{m} \sum_{i=1}^n \sum_{k=1}^m x_{k,a} \cdot x_{k,i} \cdot x_{n,i} + \sum_{i=1}^n \delta_{a,i} \cdot x_{n,i} \quad \begin{cases} x_{k,a} & \text{se } a=1 \\ 0 & \text{altrimenti} \end{cases}\end{aligned}$$

$$\begin{aligned}&\xrightarrow{\text{Sfruttando il fatto che i vettori del training set sono mutuamente ortogonali tra di loro, } \langle x_k, x_n \rangle = 0 \text{ se } k \neq n, \text{ allora:}} \\ &= -\frac{1}{m} \sum_{k=1}^m x_{n,a} + x_{n,a} = -\frac{1}{m} \cdot m \cdot x_{n,a} + x_{n,a} = 0\end{aligned}$$

MLP, regola di aggiornamento dei pesi

Fissato $x \in \mathbb{R}^n$ vettori del training set, allora l'errore commesso del MLP, per approssimare un f.n.e f è $\epsilon(w, x) = F(x) - O(w, x)$

Introducendo l'errore quadratico $\epsilon(w, x) = \frac{1}{2} \epsilon^2(w, x)$, possiamo utilizzare l'alg. di discesa del gradiente per minimizzare l'errore quadratico

Consideriamo il j -esimo neurone che collega il neurone d'uscita con il j -esimo neurone dello strato intermedio

$$\frac{\partial \epsilon(w)}{\partial v_j} = \frac{1}{2} \cdot 2 \cdot \epsilon(w) \cdot \frac{\partial \epsilon(w)}{\partial v_j} = -g'(v \cdot y) \cdot y_j$$

$$\frac{\partial \epsilon(w)}{\partial v_j} = -\frac{\partial O(w)}{\partial v_j} = -\frac{\partial g(v \cdot y)}{\partial v_j} = -g'(v \cdot y) \cdot \frac{\partial (v \cdot y)}{\partial v_j} =$$

$$\frac{\partial (v \cdot y)}{\partial v_j} = \frac{\partial}{\partial v_j} \left[\sum_{i=1}^n v_i \cdot y_i \right] = \sum_{i=1}^n \frac{\partial (v_i \cdot y_i)}{\partial v_j} = \sum_{i=1}^n y_i \cdot \frac{\partial v_i}{\partial v_j} = y_i = y_j$$

{ 1 se $i=j$
0 altrimenti }

Quindi la regola di aggiornamento dell'ultimo stato sarà:

$$v_j \leftarrow v_j + \alpha \cdot (f(x) - g(v \cdot y)) \cdot g'(v \cdot y) \cdot y_j$$

Adesso vanno aggiornate le componenti $w_{j,i}$, con $1 \leq j \leq m-1$, interne che collegano lo strato di input con lo strato nascosto. Anche in questo caso, andiamo a minimizzare l'errore quadratico andando ad applicare l'algoritmo di discesa del gradiente.

$$\frac{\partial \epsilon(w, x)}{\partial w_{j,i}} = \frac{1}{2} \cdot 2 \cdot \epsilon(w) \cdot \frac{\partial \epsilon(w)}{\partial w_{j,i}} =$$

$$\frac{\partial \epsilon(w)}{\partial w_{j,i}} = -\frac{\partial O(w, x)}{\partial w_{j,i}} = -\frac{\partial g(v \cdot y)}{\partial w_{j,i}} = -g'(v \cdot y) \cdot \frac{\partial (v \cdot y)}{\partial w_{j,i}} =$$

$$\frac{\partial (v \cdot y)}{\partial w_{j,i}} = \sum_{k=1}^m \frac{\partial (v_k y_k)}{\partial w_{j,i}} = \sum_{k=1}^m v_k \cdot \frac{\partial y_k}{\partial w_{j,i}} = \sum_{k=1}^m v_k \cdot \frac{\partial g(w_k \cdot x)}{\partial w_{j,i}} = \sum_{k=1}^m v_k \cdot g'(w_k \cdot x) \cdot \frac{\partial (w_k \cdot x)}{\partial w_{j,i}} =$$

$$\frac{\partial (w_k \cdot x)}{\partial w_{j,i}} = \frac{\partial}{\partial w_{j,i}} \left[\sum_{t=1}^m w_{k,t} \cdot x_t \right] = \sum_{t=1}^m x_t \cdot \frac{\partial w_{k,t}}{\partial w_{j,i}} = x_t = x_i$$

{ 1 se $t=i \wedge k=j$
0 altrimenti }

Quindi la regola di aggiornamento è

$$w_{j,i} \leftarrow w_{j,i} + \alpha \cdot (f(x) - g(y \cdot v)) \cdot g'(y \cdot v) \cdot y_j \cdot g'(w_j \cdot x) \cdot x_i$$