

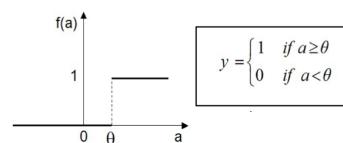
Artificial Neural Networks

le Artificial Neural Networks sono solitamente composte da:

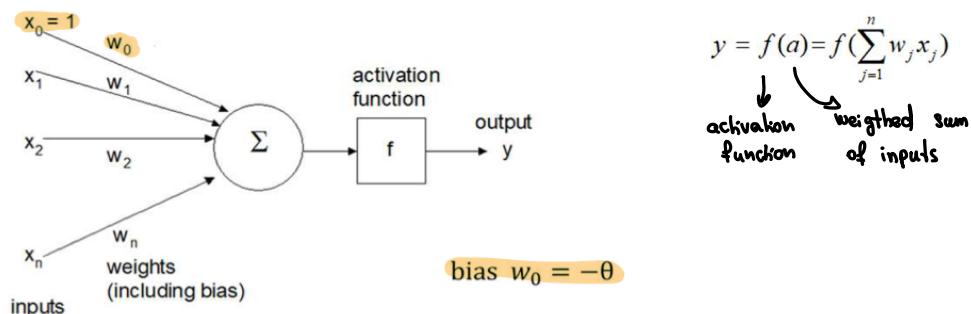
- neuroni artificiali, i quali solitamente hanno molti input ma un solo output.
- le connessioni tra i neuroni, le quali ognuna di esse ha un adjustable weight che venga modificato durante l'operazione di training.
- le activation functions, una per ogni neurone, la quale viene applicata sulla somma pesata degli inputs e produce un output "y".

DSS: Quando abbiamo delle binary activation functions

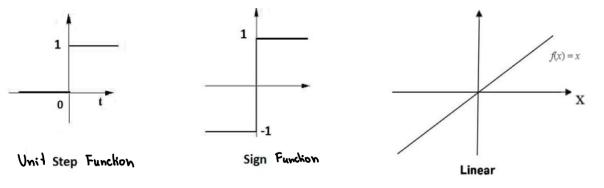
soltanente viene usata una soglia θ per indicare se l'output "y" vale 0 o 1.



Tipicamente, la soglia viene sottratta dalla somma pesata dell'input andando a considerare un input x_0 sempre uguale a 1 e il peso associato w_0 (bias) corrispondente a $-\theta$.

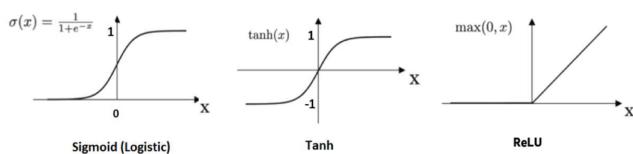


Activation Function utilizzate



DSS:

Sigmoid generalizza la unit step function usando lo slope parameter (x)



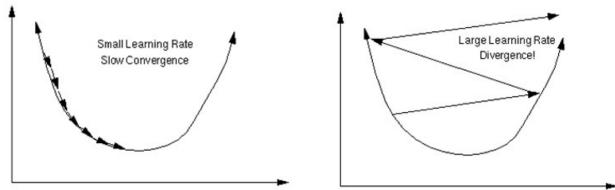
Neural Network Training (supervised)

Il training di una rete neurale consiste nel, dato ogni sample del training set, modificare i pesi per minimizzare l'errore, cioè minimizzare la differenza tra la risposta desiderata e la risposta della rete.

Questo procedimento viene ripetuto più volte (cambiando l'ordine in cui vengono presentati i dati di training) fin quando non ci sono importanti cambiamenti nei pesi.

Oss:

- I pesi vengono inizializzati randomicamente e aggiustati durante il training verso la direzione che minimizza l'errore (gradient descent algorithm).
- Durante il training viene usato il learning rate per gestire la velocità di convergenza:
 - a) learning rate troppo basso \rightarrow Troppo tempo per allenare la rete
 - b) learning rate troppo alto \rightarrow Oscillazioni intorno al minimo (può divergere)



Soltanamente, il learning rate è scelto sperimentalmente e può essere anche dinamico.

Learning Algorithms

- a) Online learning: i pesi vengono aggiornati immediatamente dopo ogni training sample.

Vantaggi:

- Può essere usato quando non c'è un fixed training set
- Richiede meno storage space
- Il gradiente di un singolo sample può essere considerato una noisy approximation dell'intero gradiente e può aiutare a scappare dai minimi locali.

2) Batch learning : Vengono accumulate le gradient contributions per tutti i sample del training set prima di aggiornare i pesi.

- Vantaggi:
 - Fornisce una stima accurata del gradient vector
 - Può essere parallelizzato

3) Mini-Batch learning : I pesi vengono aggiornati ogni n sample, con n > 1 ma minore della dimensione del training set.

Più utilizzato

Delta Rule (least mean square)

Per allenare una neural network con neuroni con f. lineare possiamo usare la delta rule (least mean square algo)

L'esempio per un dato esempio è $\delta = t - y$ dove:

- y = output della rete
- t = target output

l'adjustment che deve essere fatto ai pesi è dato da $\Delta w_i = \eta \delta x_i$ dove η = learning rate

Quindi i pesi aggiornati saranno: $w_i(n+1) = w_i(n) + \Delta w_i(n)$

OSS: Per funzioni di costo complesse (con più minimi) l'allenamento della rete può fermarsi in un local minima.

SOLUZIONE: Usare un termine chiamato "momentum" m moltiplicate per il "previous change" di w.

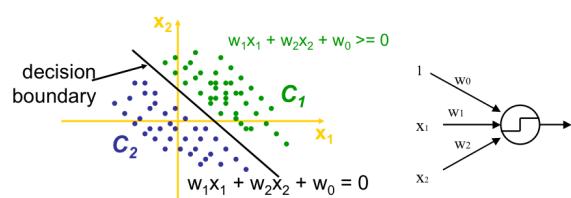
$$\Delta w_{ij}(n+1) = \eta \delta_j x_i + m \Delta w_{ij}(n) \quad \text{dove } 0 < m < 1 \text{ e } m \text{ va determinato tramite trial & error.}$$

Perceptron (classificazione lineare)

Un perceptron è un neurone con una funzione di attivazione non lineare usato per la binary classification.

Ogni perceptron definisce un decision boundary che divide lo spazio in due regioni.

Le activation function utilizzate sono solitamente la sign function o la unit step function.



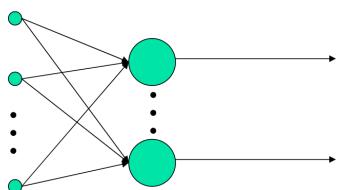
Learning Algorithm

- 1) Inizializza i pesi (a zero o valori piccoli random)
- 2) Sceglie un learning rate η (numero tra 0 e 1)
- 3) Per ogni training sample continua, fin quando i pesi non cambiano, a calcolare l'output activation $y = f(w \cdot x)$ e:
 - a) Se $y = t \Rightarrow$ non cambia i pesi
 - b) Se $y \neq t \Rightarrow$ aggiorna i pesi $\rightarrow w^{\text{new}} = w^{\text{old}} + \eta(t - y)x$

Essendo che il decision boundary si muove di una piccola dimensione verso ogni punto missclassified, abbiamo che la perceptron learning rule converge sicuramente a una soluzione in un numero finito di epoch (steps) se il problema è linearly separable

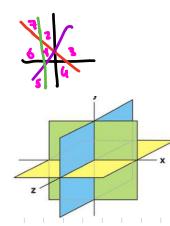
Multiple-neuron Perceptron

Usando più perceptron è possibile classificare più classi (multi-class classification).



OSS: Il numero di classi che possono essere rapp. dipende dalla dimensione dello spazio.

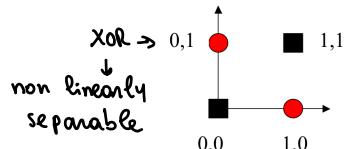
ES: $R^2 \rightarrow$ 3 perceptron, 7 classi
 $R^3 \rightarrow$ 3 perceptron, 8 classi



Non linear Problems

Il perceptron può solo risolvere problemi lineari, ad esempio non può risolvere lo XOR (simplest non linear problem).

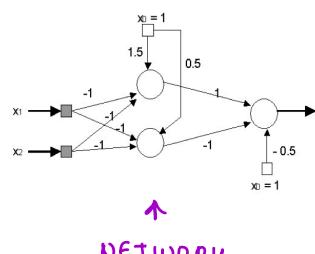
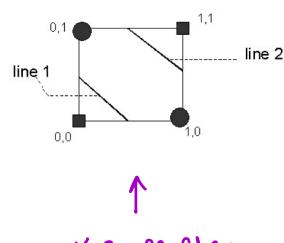
Possibili soluzioni:



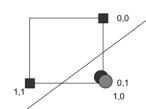
- 1) Definire una activation function specifica per il problema. Non vogliamo adottare questa soluzione perché vogliamo una general technique per rappresentare una qualsiasi funzione indipendentemente dalla complessità
- 2) Usare gli hidden layers

Esempio XOR

Usiamo una rete composta da un **hidden layer** composto da due neuroni, il quale **trasforma** i punti in modo che siano **linealmente separabili** (matematicamente) e rappresenta due linee (graficamente). L'output layer combina le info che riceve dall' hidden layer.



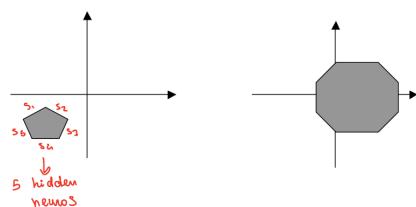
		Net input to hidden layer		Output from hidden layer	
x_1	x_2	unit 1	unit 2	unit 1	unit 2
1	1	-0.5	-1.5	0	0
1	0	0.5	-0.5	1	0
0	1	0.5	-0.5	1	0
0	0	1.5	0.5	1	1



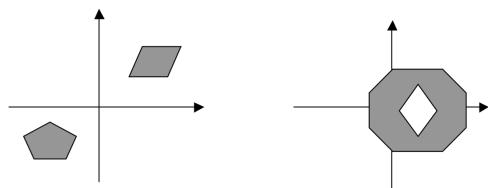
TRANSFORMATION OF HIDDEN LAYER

HIDDEN LAYERS

- +1) Una rete con un **hidden layer** può modellare regioni con un numero di lati al più uguale al numero di **hidden neurons**.



- 2) Una rete con due **hidden layers** può modellare decision regions di **complessità arbitraria**, perché permette di unire regioni insieme.



OSS: Nella pratica, la maggior parte dei problemi possono essere risolti con un solo **hidden layer**, a volte due **hidden layers**.

Attenzione: Per rappresentare funzioni più complesse abbiamo bisogno di aumentare il numero di hidden neurons MA troppi neuroni in un singolo hidden layer o troppi hidden layer possono avere effetti negativi sulle performance.

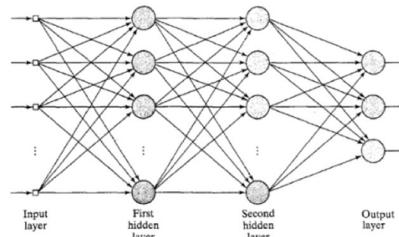
→ In generale, la migliore scelta è di iniziare con una rete che ha un numero minimo ragionevole di hidden neurons, allenarla e, SOLO SE NON OTTIENIAMO IL RISULTATO DESIDERATO, iniziamo ad aumentare il numero di hidden neurons / layers.

Universal Approximation theorem: Una rete con un hidden layer è abbastanza
hidden neurons può approssimare una qualunque
funzione continua con un qualsiasi grado di accuracy.

Multilayer Perceptron (MLP)

È una rete composta da più layer e neuroni.

I neuroni degli hidden layers non sono perceptron, hanno activation function differentiabili (per permettere la backpropagation)



Error Backpropagation (Feedforward Network)

Per allenare una rete multi-layer abbiamo bisogno del back propagation algorithm.

Consiste nei seguenti passi:

- 1) Forward Activation
- 2) Calculate the output error
- 3) Error backpropagation (usa una generalizzazione della delta rule)

Backpropagation Algorithm

L'idea è di calcolare l'errore dell'ultimo layer per poi usarlo durante la backpropagation per calcolare gli errori degli altri layer, fino all'input layer.

1) Inizializza i pesi con valori randomici piccoli

2) Dato il risultato del primo sample, calcola l'errore per ogni output neuron

$$\delta_j = (t_j - o_j) \cdot f'(net_j)$$

label ↓
model result ↓ somma pesata

3) Per tutti gli hidden layers calcola l'errore per ogni neurone (dall'output all'input)

$$\delta_j = f'(net_j) \sum_s \delta_s w_{js}$$

dove:

- $\delta_s = \delta$ dei nodi del layer precedente (^{ad esempio} e' output layer)
- $w_{js} =$ Peso dell'arco che collega j a s .

4) Per tutti i layers, aggiorna i pesi per ogni neurone utilizzando la δ corrispondente a quel layer.

$$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n)$$

↓ ↓
input layer momentum - per efficienza
 j (output layer i)

5) Ripeti per ogni training sample

6) Calcola l'errore sul training set: se l'errore è "tolerato" allora l'algoritmo è detto "converged", altrimenti continua con il training

Limitazioni di una network di linear units

Un multilayer Network con linear activation functions possono risolvere solo problemi che una single-layer network (perceptron) può risolvere.

Di conseguenza, per le multilayer Networks, sono richieste non-linear activation functions differenziabili (per il backpropagation algorithm).

Creazione di un Training Set

Si parte dai raw data e dobbiamo svolgere:

- 1) Data Analysis
- 2) Variable selection
- 3) Data Preprocessing
- 4) Remove outliers
- 5) Gestione missing values
- 6) Gestione dati non numerici
- 7) Gestione unbalanced data

Quanti training examples?

Il numero di training examples deve essere un rappresentative sample dei dati su cui la rete lavorerà.

Solitamente una rete neurale grossa richiede più training data che una piccola.

Possibili heuristic guidelines:

1) 5-10 training sample per ogni peso nella rete

2) $K = O\left(\frac{w}{\epsilon}\right)$ dove:

$K = n^o$ training sample

$w = n^o$ weights

$\epsilon =$ accuracy che vogliamo ottenere

Training stopping condition

Training stops quando:

- 1) Un certo numero di epoch passano (dove epoch è la presentazione di tutti i training data)
- 2) L'errore raggiunge un livello accettabile
- 3) L'errore non migliora ulteriormente

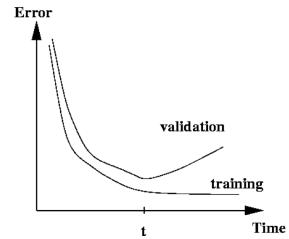
Early stopping

Per prevenire l'overfitting abbiamo bisogno di un terzo dataset chiamato validation set che viene usato durante il training per verificare che la rete non sia in overfitting.

Questo perché sia l'errore di training che di validation

diminuiscono ma mano con il training della rete, ma a un certo

punto l'errore di validation inizia a aumentare leggermente (overfitting).



Dobbiamo fermare il training al punto minimo della curva del validation errore. (early stopping).

Network size

1) Large network = Buono per funzioni complesse quando abbiamo dati a sufficienza.

D'altra parte, troppi pesi possono essere uno svantaggio perché la neural network può usare per memorizzare i training data (overfit)

2) Small network = La rete non può imporre il problema completamente.

Strategy: Iniziare con pochi hidden neurons e aumentare il numero monitorando la generalizzazione, tramite il validation, ad ogni epoca.

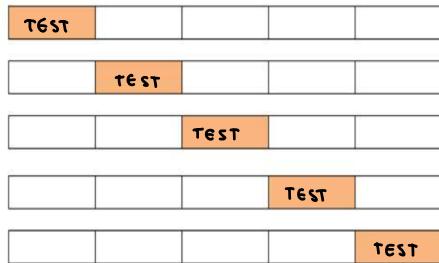
K-fold cross validation

La k-fold cross validation viene solitamente usata quando ci sono pochi dati disponibili per effettuare "con successo" la divisione in train, validation e test set.

Questo metodo serve per valutare due architetture differenti mettendo a confronto i risultati ottenuti.

Algoritmo:

- 1) Shuffle del dataset (ordinamento random)
- 2) Il dataset viene suddiviso in k-folds
- 3) Vengono fatte k iterazioni di train e test usando per ognuna di esse una differente fold come test set e le rimanenti fold come train set.



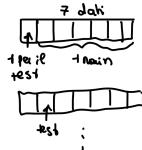
- 4) Ogni iterazione produce due score (train-errore e test-errore).

Venne calcolata successivamente la media tra questi valori.

Possibili varianti dell'algoritmo:

1) **leave-one-out**: Viene usata quando i dati sono pochissimi.

Ogni fold è composta da solo 1 elemento.



2) **stratified**: le fold vengono create in modo da mantenere la stessa proporzione delle classi.

↳ es: Dopo aver bilanciato le classi (ad esempio 50/50), ogni fold contenrà le classi con la stessa proporzione (50/50).

3) **blended**: Vengono svolte più iterazioni della k-fold cross validation (più shuffle del dataset)

e viene calcolato l'avg sulle varie k-fold cross validation.

Preventing overfitting

Quali sono le possibili tecniche per prevenire l'overfitting?

1) **early stopping**

2) **ottenere più training data**: Perché overfitting può succedere quando abbiamo troppi pesi rispetto al numero di training data.

↳ problema: A volte impossibile ottenere nuovi dati o fare labelling di molti dati (time consuming)

3) **Data augmentation**: Aumentare "artificialmente" la grandezza del dataset introducendo differenti tipologie di trasformazioni o distorsioni dei dati disponibili.

4) **Ridurre la dimensione della rete**

5) **aggiungere la weight regularization**: Fontane i pesi ad ottenere piccoli valori aggiungendo una penalità che penalizza i pesi grandi.

• **L_2 regularization**: $J_{\text{reg}} = \frac{1}{2} \lambda \sum_i w_i^2$

• **L_1 regularization**: $J_{\text{reg}} = \lambda \sum_i |w_i|$

$$J_{\text{Tot}} = J + J_{\text{Reg}}$$

$$\frac{\partial J_{\text{Reg}}}{\partial w_k} = \frac{\partial}{\partial w_k} \left(\frac{1}{2} \lambda \sum_i w_i^2 \right) = \lambda \cdot w_k$$

DSS: λ è la regularization constant, più λ è grande maggiore viene data importanza alla penalità

→ Tipicamente 0.1

6) Dropout: il dropout consiste nel definire un dropout rate il quale specifica la percentuale di neuroni che vengono temporaneamente rimossi da ogni layer (tranne l'output layer) scelti random.

Osservazioni:

- 1) Il dropout rate può essere specificato per ogni layer o in modo "globale"
 - 2) Il dropout avviene solo durante il training
 - 3) Ad ogni minibatch vengono scelti diversi neuroni per il dropout
- Perché usarlo?
- 1) Ogni neurone è costretto a capire il real mapping tra input e output perché non può "fare affidamento" sugli altri neuroni non sapendo quali tra essi saranno affetti dal dropout
 - 2) La rete è composta da tante composizioni di diverse reti, perché ad ogni mini-batch diversi neuroni sono attivi (viene svolta una specie di ensembling technique)

Multi-class classification

Per poter fare la multi-class classification usiamo per l'output layer la "softmax function" la quale converte un vettore di numeri reali in un probability distribution vector.

Ogni output neuron produce in output un valore $[0,1]$ t.c. la loro somma sia uguale a 1.

Questo valore indica la probabilità che il sample appartenga ad una determinata classe.

$$\text{Softmax function } y_j = \frac{e^{net_j}}{\sum_k e^{net_k}}$$

Per fare il training per questo modello vogliamo minimizzare la funzione di costo, chiamata cross-entropy.

$$L = - \sum_{i=1}^c t_i \log(y_i) = -t \cdot \log(y)$$

→ OSS: Questa formula è ottenuta dalla formula usata per calcolare l'entrope tra due distribuzioni.

$$H(p, q) = -\sum_x p(x) \log(q(x))$$

$p(x)$ = true distribution,
 $q(x)$ = estimated distribution

dove:

per il sample

- t_i = è un binary indication \rightarrow + se il class label è corretto; 0 altrimenti $\rightarrow t = [0, \dots, 1, \dots, 0]$
- y_i = Probabilità che il sample appartenga alla classe c_i (output della rete neurale)

↳ ps: posso avere più di un +

OSS: Se t è un one-hot-vector $[0, \dots, 1, \dots, 0]$ la loss function diventa: $L = -\log(y_k)$ (un solo +)

Radial Basis Functions (RBF)

Sono funzioni che possono essere usate al posto delle activation functions le quali risposte diminuiscono (o aumentano) monotonically con la distanza da un punto centrale.

I parametri di queste funzioni sono il centro e lo spread (raggio).

- Small spread: la funzione è molto reflettiva, cioè solo i punti vicini al centro producono valori massimi.
- Large spread: Viceversa.

La RBF più usata è la Gaussian function:
$$h(x) = \exp\left(-\frac{(x-t)^2}{2\sigma^2}\right)$$

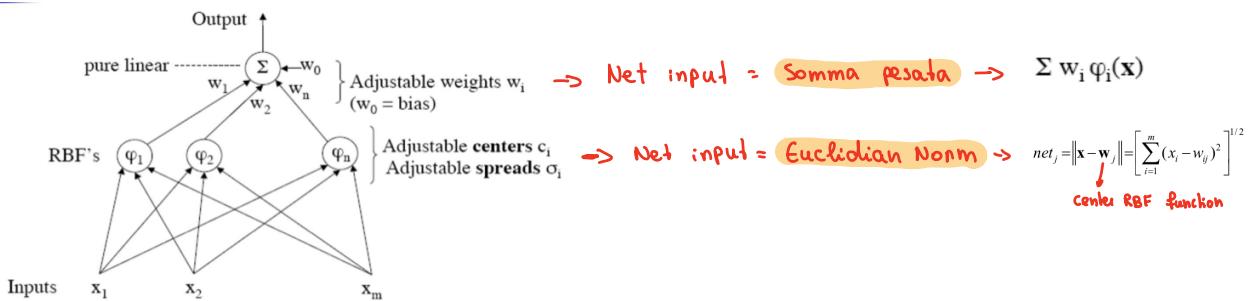
↑ centro
↓ spread

RBF network

le RBF networks hanno come particolarità la seguente caratteristica:

- 1) Gli hidden layers usano RBF functions \rightarrow Mapping input-hidden layers non lineare
 - 2) Gli output layers usano funzioni lineari \rightarrow Mapping hidden layers-output è lineare
- Di solito (non sempre) il numero di neuroni dell'hidden layer è maggiore di quello dell'input layer, perché se mappo un problema in uno spazio dimensionale maggiore è più probabile che il problema diventi linearmente separabile.

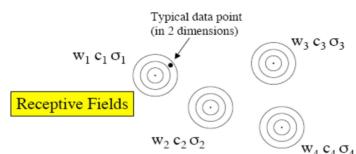
Negli hidden layer viene applicata la RBF function alla EUCLIDIAN NORM degli input rispetto al centro delle RBF functions



OSS:

1) Una funzione viene approssimata come una linear combination delle RBFs.

2) Una RBF risponde solo a una piccola regione dell'input space dove la funzione è centrata.



3) Una RBF network può essere usata non solo per function fitting ma anche per classification.
→ Usando le RBF function per mappare i punti in uno spazio linearmente separabile

RBF network Training

Durante il Training di una RBF network dobbiamo sistemare tre parametri:

- y centri delle RBFs
- gli spreads delle RBFs
- y pesi dall'hidden layer all'output layer

Abbiamo visto 3 tipologie di learning algorithm:

1) Fixed centers selected at random

CENTERS → a) y centri vengono scelti a random dai training data

SPREADS → b) gli spread sono scelti tramite "normalization", cioè la standard deviation delle Gaussian Functions è fissata tramite normalizzazione in accordo con lo spread dei centri:

$$\sigma = \frac{d}{\sqrt{2M}}$$

dove : • d = Massima distanza tra i centri (Max dist. tra le coppie di centri)
• H = n° di centri

Questa scelta per lo spread assicura che le Gaussian functions non sono too peaked o troppo piatte, vogliamo un tradeoff.
(\L) (\sim)

- c) Dati i due settings per i centri e lo spread, l'activation function per gli hidden neurons è la seguente:
- $$\varphi_i(\|x - t_i\|) = \exp\left(-\frac{M}{d^2} \|x - t_i\|^2\right), \quad i=1, \dots, M$$

- $\text{output weights} \rightarrow d$) i pesi dell'output layer vengono invece calcolati tramite lo pseudo-inverse method :
- Per ogni esempio (x_i, d_i) , l'output della rete vogliamo sia uguale a d_i ,
- cioè :
- $$y(x_i) = w_1 \varphi_1(\|x_i - t_1\|) + \dots + w_M \varphi_M(\|x_i - t_M\|) = d_i$$
- output network

- Possiamo risolverlo in matrix form per N esempi:

$$\begin{bmatrix} \varphi_1(\|x_1 - t_1\|) & \dots & \varphi_M(\|x_1 - t_M\|) \\ \dots \\ \varphi_1(\|x_N - t_1\|) & \dots & \varphi_M(\|x_N - t_M\|) \end{bmatrix} [w_1 \dots w_M]^T = [d_1 \dots d_N]^T$$

- Sia la matrice uguale a Φ , possiamo scrivere:

$$\Phi \begin{bmatrix} w_1 \\ \dots \\ w_M \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

- Sia Φ^+ la pseudo inversa di Φ , otteniamo i pesi usando la seguente formula:

$$[w_1 \dots w_M]^T = \Phi^+ [d_1 \dots d_N]^T$$

2) Self-organized selection of centers

- 4 centri vengono trovati tramite un clustering algorithm, permettendo di piattarli in regioni dell' input space dove dati significativi sono presenti.
- Spreads sono scelti tramite normalization (come visto sopra)
- Viene usata una supervised learning rule (ad esempio la delta rule) per calcolare i linear weights dell' output layer.

3) Supervised selection of centers

In questo appuccio le update formula vengono usate per allenare contemporaneamente weights, centers e spread usando il gradient descent.

In pratica viene minimizzato lo squared error

$$E = \frac{1}{2} \sum_{j=1}^N e_j^2 \quad \text{dove:}$$

- $N = \text{n° training samples}$
- $e_j = \text{error} \rightarrow e_j = d_j - y(x_j) = d_j - \sum_{i=1}^M w_i \varphi_i(\|x_j - t_i\|)$

Le risultato della minimizzazione dell' errore sono le update formula:

a) Centers $\rightarrow \Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$

b) Spread $\rightarrow \Delta \sigma_j = -\eta_{\sigma_j} \frac{\partial E}{\partial \sigma_j}$

c) Weights $\rightarrow \Delta w_{ij} = -\eta_{w_{ij}} \frac{\partial E}{\partial w_{ij}}$

OSS: Come si può notare dalle formule, ognuna di esse usa learning rate differenti.

RBF network spread analysis

Lo spread è un fattore importante per l'under/over fitting delle RBF networks.

1) Spread piccolo : Non ci sono due RBF neurons che hanno un forte output, per nessun input \rightarrow la rete non generalizza \rightarrow overfitting

→ spread più piccolo della distanza tra due input adiacenti

2) Spread grande : Tutti gli RBF neurons danno in output valori alti (≈ 1) per tutti gli input \rightarrow Non possono essere usati per generare risposte differenti \rightarrow Underfitting

Bisogna scegliere uno spread più grande della distanza tra input vectors adiacenti (per generalizzare bene), ma più piccoli della distanza across the entire input space.

MLP vs RBF networks

Entrambe le due tipologie di reti sono universal approximators e, inoltre, c'è sempre una MLP che può "mimare" una RBF e viceversa.

Le differenze sono:

- MLP costruisce global approximators per il mapping non-lineare input-output. Quindi, sono in grado di generalizzare in regioni dell'input space con pochi o nessun training data disponibile.
- RBF networks costruiscono local approximators per il mapping non-lineare input-output. Quindi, sono in grado di imparare velocemente e avere una sensibilità ridotta sull'ordine in cui vengono presentati i dati.

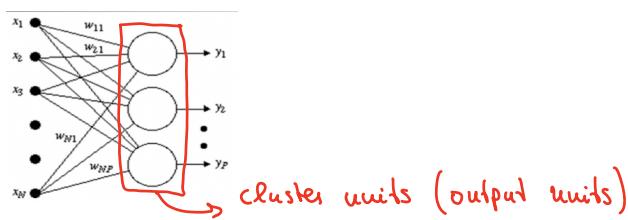
Unsupervised learning

Per quanto riguarda l'unsupervised learning (clustering), abbiamo visto due differenti unsupervised neural networks:

- 1) Competitive Networks
- 2) Self-organizing map (SOM)

Entrambe queste reti hanno un neurone di input per ogni features, e uno di output (cluster unit) per ogni cluster (svolgono il ruolo di cluster prototypes - centroidi). Ogni neurone di input è fully connected ad ogni neurone di output, quindi il numero totale di pesi connessi ad un neurone di output equivale alla dimensione dell'input vectors.

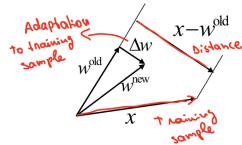
Questi pesi sono considerati come le coordinate che descrivono la posizione del cluster nell' input space. E S :



Competitive Networks

La cluster unit più vicina, secondo una similarity measure (Euclidian dist o cosine dist) al training vector (input) viene considerata la "winning unit".

- 1) I pesi vengono scelti inizialmente in modo randomico.
- 2) Dato un sample, i pesi del winning neuron vengono aggiornati per muovere il neurone in modo da avvicinarsi al training vector.



$$w^{\text{new}} = w^{\text{old}} + \Delta w = w^{\text{old}} + \alpha(x - w^{\text{old}}), \quad \alpha \in (0,1)$$

Problema: La competitive network non permette di imparare la topologia dell' input vectors, cioè non è detto che neuroni vicini rappresentino cluster vicini nell' input space.

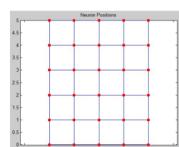
Self-organizing map (SOM)

Per risolvere questo problema, SOM aggiorna i pesi del winner neuron e anche i pesi dei neuroni nel suo vicinato (all'interno di un raggio con il winner neuron come centro).

In questo modo, SOM impara sia la distribuzione (identifica i cluster, come il competitive network), sia la topologia dell' input vectors usati come training.

OSS:

- Le cluster units sono posizionate solitamente come array/griglie
- Il raggio del vicinato diminuisce durante il training
- La learning rate determina di quanto il neurone si muove verso il training vector e diminuisce man mano durante il training (come il raggio).



Roc (Receiver Operating Characteristics) Analysis

Prima di arrivare alla Roc analysis, definiamo alcuni concetti:

1) **classifier**: È una funzione che associa un feature space ad un insieme di classi.

Viene costruito da un labeled training set e inferisce le classi ai nuovi oggetti:

2) **classification accuracy**: Indica quanto un classificatore è accurato assumendo che le probabilità di ogni classe siano costanti e bilanciate.

→ Problema: le classi sono quasi sempre sbilanciate → Non funziona.

3) **Precision e Recall**

- Siano:
- P = positive class
 - N = negative class
 - Y = predicted positive class
 - N̄ = predicted negative class
 - TP = positive obj classified positive
 - FN = positive obj classified negative
 - TN = negative obj classified negative
 - FP = negative obj classified positive

la confusion matrix è così definita:

		TRUE CLASS		Column totals
		P	N	
PREDICTED CLASS	Y	True Positives False Positives	False Negatives	P
	N	False Negatives	True Negatives	

$$\Rightarrow \begin{aligned} \text{Precision} &= \frac{TP}{TP+FP} && (\text{specificity}) \\ \text{Recall} &= \frac{TP}{P} && (\text{sensitivity}) \end{aligned}$$

Problema: i costi di missclassification di un positivo e negativo, sono equivalenti.

- $c(Y, n)$ = cost of a false positive error
- $c(N, p)$ = cost of a false negative error

$$\rightarrow c(Y, n) = c(N, p)$$

Nella realtà questo è raro, in questi casi impiattiamo la accuracy maximization con la cost minimization. → Roc

Roc Graphs → Particolarmente utili quando classi sbilanciate e costi missclassification diversi.

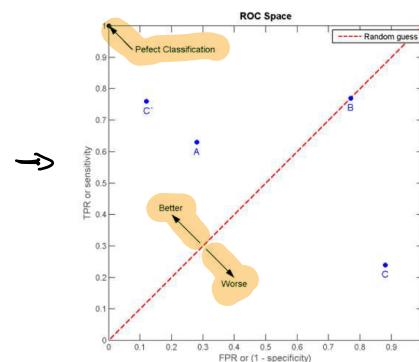
I Roc graphs sono utili per selezionare i classificatori in base alle loro performances.

Sono grafici due-dimensionali, FPR sulle x e TPR sulla y.

Roc graphs mostrano il trade-off tra i benefici (TP) e i costi (FP).

Un classificatore produce una coppia (FP rate, TP rate), la quale corrisponde ad un punto nel grafico (Roc space).

	A	B	C	C'
TP=63	FP=28	TP=77	TP=24	TP=76
FN=37	TN=72	FP=77	FP=88	FP=12
100	100	100	100	100
TPR = 0.63	TPR = 0.77	TPR = 0.24	TPR = 0.76	TPR = 0.76
FPR = 0.28	FPR = 0.77	FPR = 0.88	FPR = 0.12	FPR = 0.12
ACC = 0.68	ACC = 0.50	ACC = 0.18	ACC = 0.82	ACC = 0.82



I punti importanti in Roc sono:

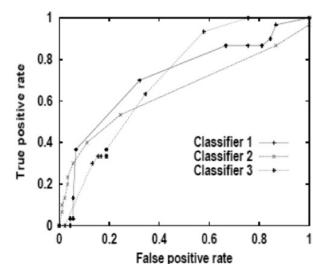
- Il punto $(0,0)$ il quale rappresenta il "never alarming", cioè non c'è mai una classificazione positiva.
- Il punto $(1,1)$ rappresenta l'"always alarming", cioè non c'è mai una classificazione negativa.
- Il punto $(0,1)$ rappresenta la perfect classification.
- I punti sulla diagonale rappresentano la strategia del "random guessing".

Roc Curves

Produce un real value

Nel caso avessimo un continuo classificatore, dobbiamo definire una soglia che discrimina tra classe positiva e negativa.

La scelta di soglie diverse porta a diverse coppie (FPR, TPR) le quali rappresentano i punti della curva nel Roc space.



Problema: è difficile scegliere quale classificatore sia il migliore se le curve si sovrappongono.

Soluzione: Possiamo usare la maximum area under the curve (AUC),

la quale aggomba le performance del classifier tra i vari tradeoff (FPR - TPR pairs).

Più è grande l'AUC, meglio è, con 0.5 che indica il "random guessing" e 1 che rappresenta la "perfect performance".

Convolutional Neural Network (CNN)

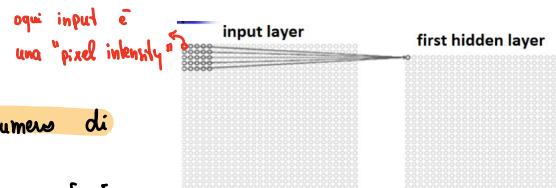
le reti CNN combinano tre idee architettoniche:

1) local receptive fields

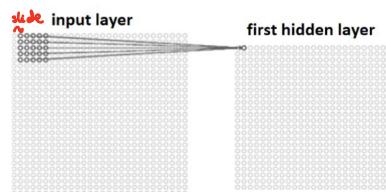
I neuroni delle CNN non sono fully connected, ma sono connessi solo a una local region.

Quindi, ogni neurone del primo hidden layer è connesso a una piccola regione quadrata, chiamata local receptive field dell' input space.

Questo ci permette di ridurre notevolmente il numero di connessioni, ad esempio per una griglia 5×5 abbiamo 5×5 pesi + 1 bias = 26 parametri.



OSS: lo slide e la grandezza della griglia sono hyperparametri.



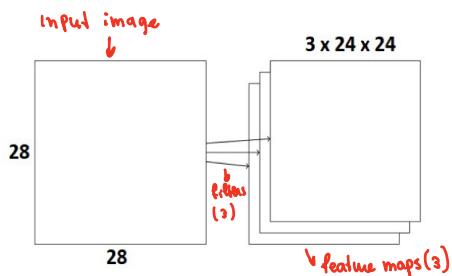
2) Shared weights and biases

Nelle CNN usiamo gli stessi pesi e lo stesso bias per tutti gli hidden neurons.

Questo vuol dire che tutti i neuroni nel primo hidden layer rilevano la stessa feature, ma in differenti posizioni dell' input image (perché applica l' activation function con gli stessi pesi ma in punti diversi dell' immagine). → Riduce il numero di weights

Quindi ogni hidden layer rappresenta un filter (definito da quale weight e bias viene scelto) e, data una input image in input ottiene in output una feature map, la quale mappa la presenza, della feature corrispondente al filtro, nell' immagine.

In genere, applico più filtri (diversi pesi e bias) creando più features maps, una feature map per ogni filtro.



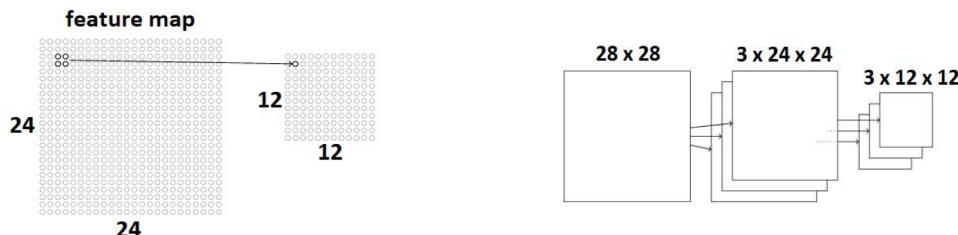
In conclusione, po ho nello stesso numero diverse features nell'immagine, dove n è il numero di features maps.

3) Pooling

Un pooling layer viene usato subito dopo un convolutional layer, il quale per ogni feature map (del conv. layer) prepara una feature map condensata (un "assunto").

Ci possono essere diversi tipi di pooling, più usati: Max e avg pooling.

Il pooling viene applicato ad ogni feature map separatamente.



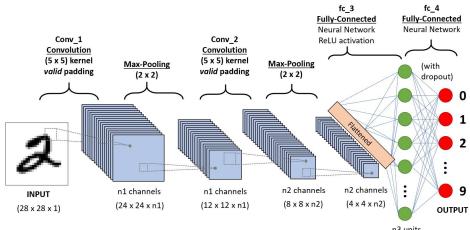
Il max-pooling, data una input region (es: 2x2, è un hyperparametro), fornisce solo il valore più alto (maximum activation)

CNN architecture

Una CNN consiste in:

- 1) Input layer: layer connesso ai pixel di un'immagine
- 2) Intermediate layers: una serie di conv e pooling layers
- 3) Un flattened / Global Avg / ... layer: fornisce i valori delle matrici ottenute dal max pooling in un unico lungo array.
- 4) Un Fully connected layer (o più)

4) **Output layer**, dipende dal task che vogliamo svolgere, ad esempio per la classificazione avrò un neurone per ogni classe.



Activation Function

Venne usata la **RELU** ($f(x) = \max(0, x)$) per cercare di evitare il gradient vanishing problem,

essendo le reti molto deep. → Perché i gradienti durante la backprop. valgono 0 o 1.

(con RELU)

Shuffle

Consiste nel far muovere un figlio non di uno step unitario (un neurone per volta) ma di uno step più lungo. Riduce la dimensione delle feature maps, ma può essere un'accorta inferiore.

Padding

Aggiunge un bordo (zero values) all'input volume per permettere alla rete di imparare informazioni sui bordi e angoli dell'immagine, che potrebbero essere perse altrimenti.

Transfer learning

Allenare CNN su un nuovo problema richiede un training set con labels di una dimensione considerabile e può richiedere molto tempo.

Può essere usata in alternativa la tecnica del **transfer learning**:

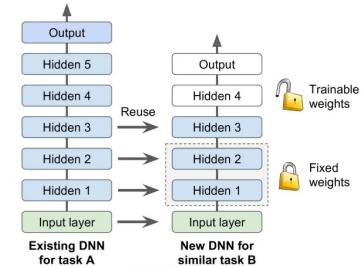
1) **Feature reuse**: Una rete già esistente e allenata viene usata come "feature extraction", cioè vengono estratti i layer intermedi e vengono usati per allenare un classificatore esterno (es: SVM...).

2) **Fine-tuning**: Utilizzare da una rete già allenata su un problema simile e
a) Aggiungere la custom network in testa alla rete già allenata

(usa la base)

b) Freziona la base e allenare solo la nuova parte aggiunta

c) Definire alcuni top layer della base
ed allenarli insieme a quelli nuovi.



CNN sentence classification

le CNN possono essere usate per fare sentence classification

- 1) Tratta la sentence come una matrice di word embeddings (ottenuti tramite word2vec)
- 2) Analizza la matrice tramite convolutional filters, per estrarre le info locali, e pooling layer per catturare le info più importanti
- 3) Viene usato un dense layer (fully connected) seguito da una softmax function per produrre l'output

CNN for object detection

Per localizzare e classificare gli oggetti in un'immagine possono essere usati due algoritmi:

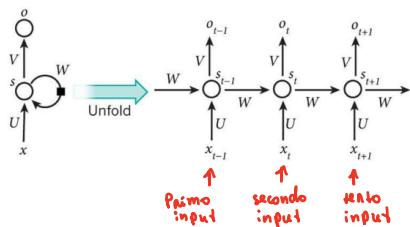
- 1)
 - a) Seleziona regioni di interesse (bounding box)
 - b) Classificalo

} R-CNN, FASTER C-NN, FASTER R-CNN
- 2) Predisci le classi e bounding box allo stesso tempo } YOLO e SSD

Recurrent Neural Network (RNN)

le reti RNN processano una input sequence un elemento per volta e mantengono uno state vector nelle loro hidden units.

Rappresentazione



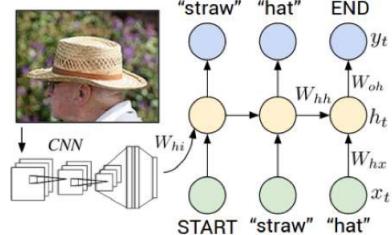
- x_t è l'input al tempo t
- dove: s_t è lo stato al tempo t $s_t = f(Ux_t + WS_{t-1})$
- o_t è l'output al tempo t $o_t = f(Vs_t)$
- (U, V, W) parametri shared ad ogni step cioè performano lo stesso task ad ogni step

Applicazioni

le applicazioni delle RNN sono principalmente:

1) Image Captioning (image \rightarrow seq. di parole)

Una CNN estrae le features da un'immagine e una RNN traduce le features in frasi predicendo la prossima parola in una frase.



Vengono usate anche i segnali di initio ("start") e fine ("end").

2) Sentiment classification (seq. di parole \rightarrow classe)

Viene presentata alla RNN una sequenza di word embeddings e la RNN produce in output un "riassunto" delle contextual info del testo (context vector).

Viene poi dato in input a un fully connected layer seguito da una softmax per ottenere la probability distribution della sentiment class.

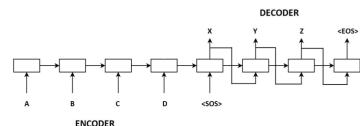
3) Machine translation (seq. di parole \rightarrow seq. di parole)

Vengono usate due RNN, una come encoder e una come decoder.

a) Encoder: Prende in input la sequenza di word embedding e produce un context vector, di lunghezza fissata, che riassume il significato del testo in input.

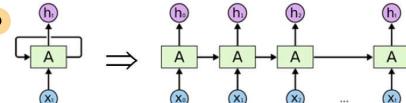
b) Decoder: Prende in input il context vector e produce una parola per volta tenendo traccia del contesto usando l'hidden state.

L'obiettivo è tradurre un testo da un linguaggio a un altro.



Allenare le RNN

1) Unfold the network: Passare dalla forma con il loop a quella sequenziale



2) Backpropagation over time:

Essendo che le unità condividono i parametri nel tempo, viene applicata una variante chiamata Backpropagation over time.

La differenza sta nel fatto che l'aggiornamento dei pesi è basato sulla somma dei gradienti accumulati nei vari timestamps.

Backpropagation standard

1. Propagate the inputs forward through the network to compute the outputs.
2. Compute the error between the predicted outputs and the ground-truth outputs.
3. Propagate the error backwards through the network to compute the gradients of the parameters with respect to the error.
4. Update the parameters using the computed gradients, for example using stochastic gradient descent (SGD) or Adam.

Backpropagation over time

1. Propagate the inputs forward through the unrolled RNN to compute the outputs.
2. Compute the error between the predicted outputs and the ground-truth outputs.
3. Propagate the error backwards through the unrolled RNN to compute the gradients of the parameters with respect to the error.
4. Accumulate the gradients over all time steps.
5. Update the parameters using the accumulated gradients, for example using stochastic gradient descent (SGD) or Adam.

Serve accumulare per permettere alla rete di imparare dall'intera sequenza.

Long - Short term Memory (LSTM)

Le LSTM sono nei RNN speciali usate per risolvere il problema delle long-term dependencies nelle RNN.

Il long-term dependency problem consiste nel fatto che più tempo passa tra l'arrivo di una info e dove essa è necessaria, più una RNN non è capace di connettere queste info (Non ha memoria a lungo termine, l'informazione "perde di efficacia" nel tempo).

Per fare ciò:

- Le LSTM usano al posto di un semplice tanh layer (come nelle RNN) quattro fully connected layer che interagiscono fra di loro.
- Usano tre gates per imparare a capire cosa salvare, rimuovere e leggere nel long-term state (sono layer che usano la sigmoid e una elementwise multiplication operation)
- le LSTM dividono lo stato di una cella in due:
 - $h_{(t)}$: short-term state
 - $c_{(t)}$: long-term state

Come funziona la LSTM component?

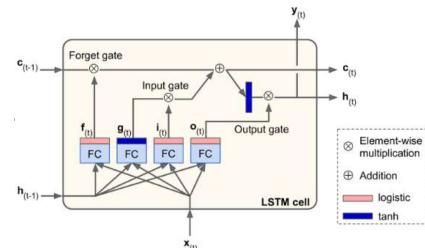
1) Nella parte di $c_{(t-1)}$

a) Passa dal forget gate il quale doppia della memoria

b) Viene aggiunta della memoria tramite l'addition op.
selezionata dall' input gate

c) Il risultato $c_{(t)}$ viene mandato subito fuori

• Il risultato $c_{(t)}$ viene anche copiato, passa da una tanh function e viene filtrato dall' output gate, diventando il nuovo short-term memory $h_{(t)}$ (che è anche l'output della cella, $y_{(t)}$)



2) Nella parte di $h_{(t-1)}$

a) $h_{(t-1)}$ viene inviato, insieme all' input vector, ai quattro fully connected layers

• FC che produce $g_{(t)}$ è il layer principale ed applica la tanh function agli input (come in una normale cella RNN).

L' output di questo layer non va direttamente fuori (come nelle RNN), ma la parte più importante viene salvata nel long-term state (e il resto buttata).

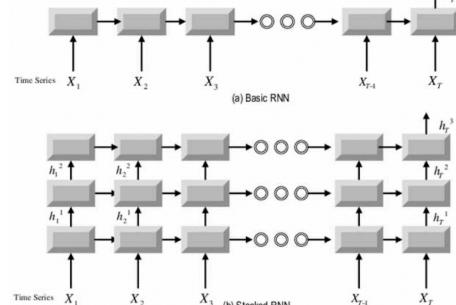
- Gli altri tre FC layers (che producono $f_{(t)}$, $i_{(t)}$ e $o_{(t)}$) sono gate controllers.
Usano la logistic function (il loro output è $[0,1]$) e vengono usati per aprire (output = 1) o chiudere (output = 0) il gate corrispondente (essendo element-wise multiplications).
 - $f_{(t)}$ controlla il forget gate, cioè quale parte di $c_{(t-1)}$ deve essere rimossa
 - $i_{(t)}$ controlla l'input gate, cioè quale parte di $g_{(t)}$ dovrebbe essere aggiunta a $c_{(t-1)}$
 - $o_{(t)}$ controlla l'output gate, cioè quale parte di $c_{(t)}$ dovrebbe essere letta e data in output allo step t , sia per $h_{(t)}$ che $y_{(t)}$.

OSS: le gated Recurrent unit (GRU) sono una versione semplificata di LSTM e sono composte da un memory state e due gates (update e reset gate).

Stacked RNN

È una rete composta da più RNN layers e permette di aumentare la complessità di modeling.

Questo tipo di rete però soffre maggiormente i problemi delle RNN, come il gradient vanishing problem.

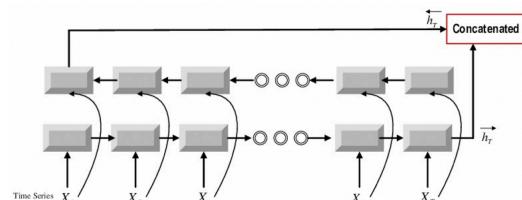


Bidirectional RNN

È un tipo di RNN architecture che permette di processare l'input sequence sia forward (come le RNN classiche) che backward.

Questo permette alla rete di catturare le informazioni sia dal passato che dal futuro della input sequence,

permettendo di avere miglioramenti in alcuni task come il sentiment analysis.



1D CNN

è un tipo di CNN disegnata per processare 1-Dimensional signals (come sequenze di parole o time-series).

- 1) Applica i filtri a regioni locali della input sequence (cercando di catturare delle features)
- 2) Queste features vengono passate al pooling per produrre una rappresentazione compatta della input sequence

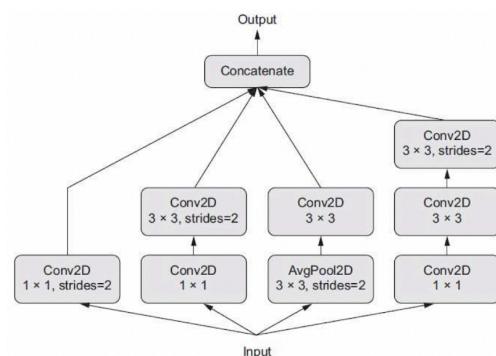
Le 1D CNN sono conosciute per la loro capacità di catturare le features e generalizzare bene, inoltre sono efficienti (posso gestire grandi datasets)

Inception Module

L'idea dell'inception module è di usare più filtri, di dimensioni diverse, e pooling layers in parallelo.

Quindi catturano le informazioni a scale diverse.

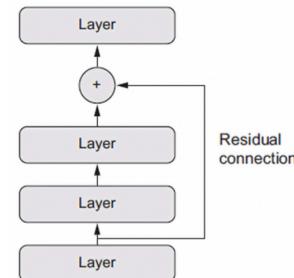
Permette di imparare rapp. più complesse e di essere più preciso.



Residual connection

La residual connection consiste nell'aggiungere l'output di un layer precedente come input di un layer successivo.

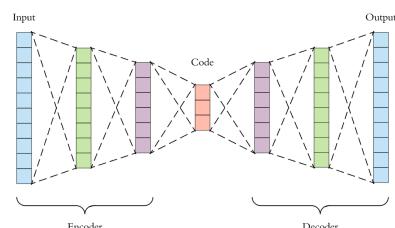
Viene usato per evitare il fenomeno del vanishing gradient.



Classical Autoencoder

È possibile usare una rete composta da un encoder e un decoder per ottenere una rappresentazione compressa degli input (usando l'encoder) e data la versione compressa riprodurre l'input con il minore errore possibile (decoder)

Reconstruction error



Generative Adversarial Network (GAN)

È un'architettura composta da due reti, un generator e un discriminator.

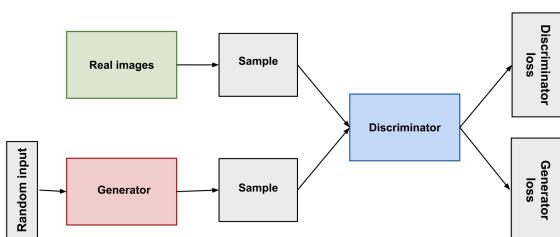
1) Generator

- Riceve un vettore di valori randomici e genera dati con la stessa struttura dei training data
- È allenato per "ingannare" il discriminator, cioè convincerlo che i dati generati da lui sono dati reali

2) Discriminator

È un classificatore che impara a classificare se i dati sono "reali" o "generati".

Oss: più il generatore migliora, più il discriminatore dovrebbe peggiorare perché non riesce a costruire a distinguere



Fuzzy Logic

La Fuzzy logic è una logica in cui si può attribuire a ciascuna proposizione un grado di verità compreso tra 0 e 1.

Esempio: Si può ad esempio dire che:

- un neonato è "giovane" di valore 1
- un diciottenne è "giovane" di valore 0,8
- un sessantacinquenne è "giovane" di valore 0,15

Fuzzy sets

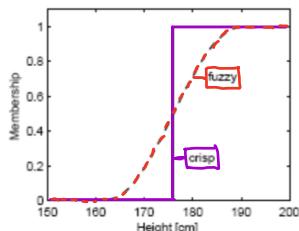
(chiamati anche "crisp")

A differenza dei set classici dove un elemento può appartenere o meno al set, nel fuzzy set ogni elemento può appartenere ad esso con un qualsiasi grado tra 0 e 1.

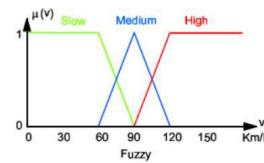
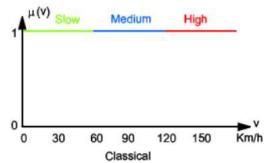
Universe of discourse:insieme di elementi sui quali è definito un fuzzy set

Membership function: grado di appartenenza al fuzzy set.

Example: set of tall men



Another example: speed



Formalmente:

Un fuzzy set A è caratterizzato da una membership function $\mu_A: X \rightarrow [0,1]$ dove X è l'universe of discourse (che può essere discreto o continuo).

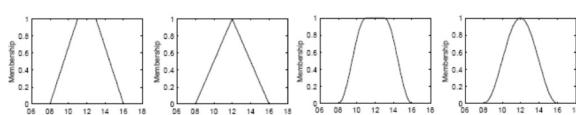
Il fuzzy set A è composto dalle coppie $(x, \mu_A(x))$, cioè:

$$A = \{ (x, \mu_A(x)) \mid x \in X \}$$

Domini

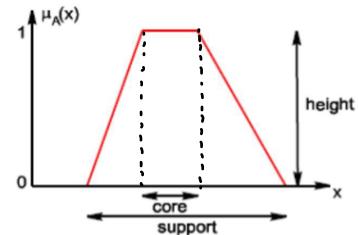
- X discreto $\rightarrow A = \sum_X \mu_A(x)/x$
 - X continuo $\rightarrow A = \int_X \mu_A(x)/x$
- $\rightarrow A(x) = \mu_A(x)$

OSS: le membership functions sono tipicamente bell-shaped o triangular o trapezoidal



Proprietà dei Fuzzy sets

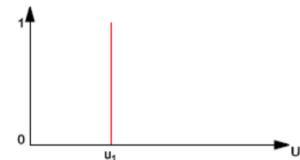
1) Height (A) = $\sup_{x \in X} \mu_A(x)$ → Limite superiore valore $\mu_A(x)$



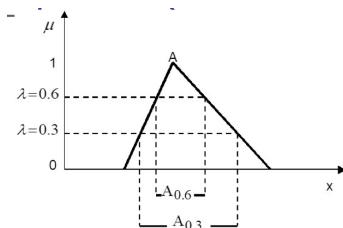
2) cone (A) = $\{x \in X \mid \mu_A(x) = 1\}$ → insieme x t.c $\mu_A(x) = 1$

3) support (A) = $\{x \in X \mid \mu_A(x) > 0\}$ → insieme x t.c $\mu_A(x) > 0$

4) Fuzzy singleton: Fuzzy set il quale supporto è un singolo punto



5) α -cut: è il crisp set $A_\alpha = \{x \mid \mu_A(x) \geq \alpha\}$



Oss: Tutti gli α -cut t.c $\alpha \in [0,1]$ sono convex sets

Fuzzy sets operations

Le operazioni di intersezione, unione e complemento possono essere estese ai fuzzy sets e

Sono:

1) Triangular norm (t-norms) → Intersezione ($A \cap B(x) = T[A(x), B(x)]$)

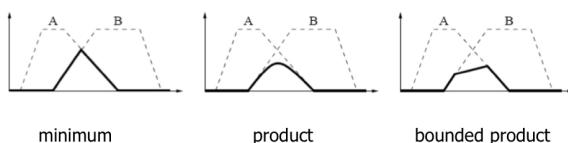
- Commutativity: $T(a, b) = T(b, a)$
- Monotonicity: $T(a, b) \leq T(c, d)$ if $a \leq c$ and $b \leq d$
- Associativity: $T(a, T(b, c)) = T(T(a, b), c)$
- Identity element: $T(a, 1) = a$

è una funzione $T: [0,1] \times [0,1] \rightarrow [0,1]$ con le seguenti proprietà:

a) minimum t-norm: $T(A, B) = \min(\mu_A(x), \mu_B(x)) \quad \forall x \in X \rightarrow$ standard operation

b) product t-norm: $T(A, B) = \mu_A(x) \cdot \mu_B(x) \quad \forall x \in X$

c) Bounded product t-norm: $T(a, b) = \max(0, \mu_A(x) + \mu_B(x) - 1) \quad \forall x \in X$



2) Triangular conorms (T-conorms o S-norms) \rightarrow Unione

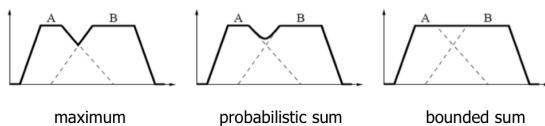
Sono complementari alle t-norms e hanno le stesse proprietà:

- Commutativity: $S(a, b) = S(b, a)$
- Monotonicity: $S(a, b) \leq S(c, d)$ if $a \leq c$ and $b \leq d$
- Associativity: $S(a, S(b, c)) = S(S(a, b), c)$
- identity element: $S(a, 0) = a$

a) maximum conorm: $T(A, B) = \max(\mu_A(x), \mu_B(x)) \quad \forall x \in X \rightarrow$ standard operation

b) probabilistic sum conorm: $T(A, B) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x) \quad \forall x \in X$

c) Bounded sum conorm: $T(A, B) = \min(1, \mu_A(x) + \mu_B(x)) \quad \forall x \in X$



3) Fuzzy complement \rightarrow Complements

è definito come $\bar{A}(x) = C(A(x))$ dove $C: [0,1] \rightarrow [0,1]$ e soddisfa:

Sequenti requisiti:

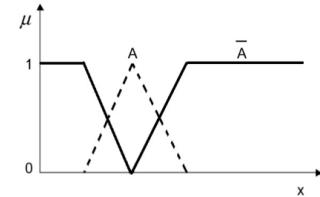
- $C(0) = 1$ and $C(1) = 0$ (boundary conditions)
- C is monotonic decreasing
- C is a continuous function
- C is involutive, i.e., $C(C(x)) = x$

a) standard complement $\bar{A}(x) = 1 - A(x) \rightarrow$ standard operation

b) round complement $\bar{A}(x) = \sqrt{1 - [A(x)]^2}$

c) Yager $\bar{A}(x) = (1 - [A(x)]^p)^{1/p}, p \in (0, \infty)$

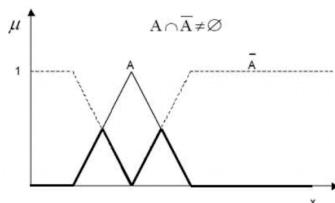
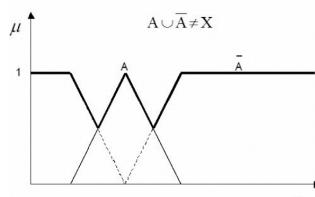
d) Sugeno $\bar{A}(x) = \frac{1-x}{1+sx}, s \in (-1, \infty)$



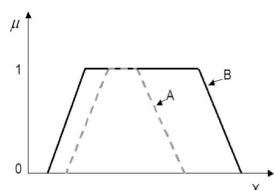
Attenzione:

• $A \cup \bar{A} \neq X$

• $A \cap \bar{A} \neq \emptyset$



4) Fuzzy set inclusion: $A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x) \quad \forall x \in X$

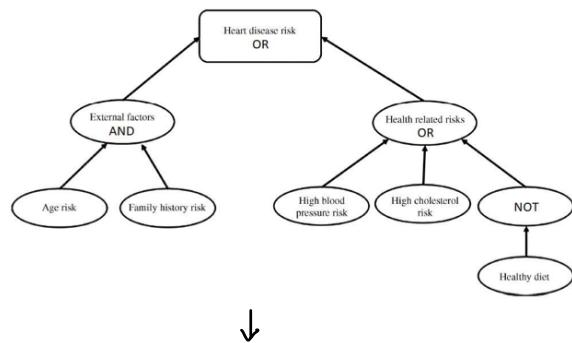


Example 1

A four-person family wants to buy a house that is comfortable and large:

- given the universe $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ (number of bedrooms) we define the fuzzy sets
 $\begin{aligned} \text{Comfortable} &= [0.2 \ 0.5 \ 0.8 \ 1 \ 0.7 \ 0.3 \ 0 \ 0 \ 0 \ 0] \\ \text{Large} &= [0 \ 0 \ 0.2 \ 0.4 \ 0.6 \ 0.8 \ 1 \ 1 \ 1 \ 1] \end{aligned}$
- The intersection of Comfortable and Large is
 $\min(\text{Comfortable}, \text{Large}) = [0 \ 0 \ 0.2 \ 0.4 \ 0.6 \ 0.3 \ 0 \ 0 \ 0 \ 0]$
- If the goal were to satisfy at least one criterion, we would perform the union of Comfortable and Large
 $\max(\text{Comfortable}, \text{Large}) = [0.2 \ 0.5 \ 0.8 \ 1 \ 0.7 \ 0.8 \ 1 \ 1 \ 1 \ 1]$
- If the children move away from the family within a year or two, parents may choose to buy a house that is Comfortable and Not Large, or
 $\min(\text{Comfortable}, 1 - \text{Large}) = [0.2 \ 0.5 \ 0.8 \ 0.6 \ 0.4 \ 0.2 \ 0 \ 0 \ 0 \ 0]$

Example 2



In molti casi reali vengono usati : fuzzy decision tree
di diversi criteri per fare una confident choice

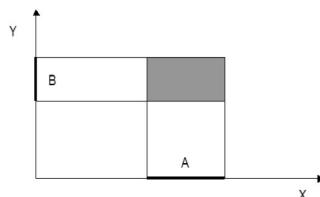
Crisp and Fuzzy relations

Descriviamo la differenza delle relations nei crisp e fuzzy sets.

OSS: Il prodotto cartesiano di due insiemi A e B è un singolo insieme che contiene le coppie ordinate (a, b) ed è denotato dal simbolo $A \times B$.

Crisp Relations

Dato un prodotto cartesiano tra due crisp sets A e B



$$A \times B = \{(x, y) | x \in A, y \in B\}$$

Una crisp relation è un sottoinsieme del prodotto cartesiano che soddisfa specifiche condizioni.

La funzione caratteristica di una relazione è definita come: $R(x, y) = \begin{cases} 1 & \text{if } (x, y) \in R \\ 0 & \text{if } (x, y) \notin R \end{cases}$

Date due crisp relations $R: X \rightarrow Y$ e $S: Y \rightarrow Z$, le posso componere (se hanno un set in comune).

$$W = R \circ S: X \rightarrow Z$$

- Max-min composition:** $\chi_W(x, z) = \bigvee_{y \in Y} (\chi_R(x, y) \wedge \chi_S(y, z))$

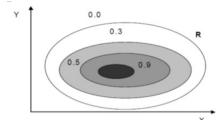
↓
 max over Y
 (common set)
 ↓
 minimum
 characteristic function of R

- max-product composition:** $\chi_W(x, z) = \bigvee_{y \in Y} (\chi_R(x, y) \cdot \chi_S(y, z))$

↓
 max over Y
 (common set)
 ↓
 product
 characteristic function of R

Fuzzy relations

Una fuzzy relation è un fuzzy set definito su un prodotto cartesiano



con lo scopo di portare i valori del prodotto cartesiano in un intervallo chiuso $[0,1]$.

La relazione $R: X \times Y \rightarrow [0,1]$ è quindi così definita: $R(X, Y) = \{(x, y), \mu_R(x, y) | (x, y) \in X \times Y\}$

Osservazioni:

- $R \cup \bar{R} \neq E$ dove $E = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
complete relation
- $R \cap \bar{R} \neq D$ dove $D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Null relation

Esempio:

- relation 'very far'

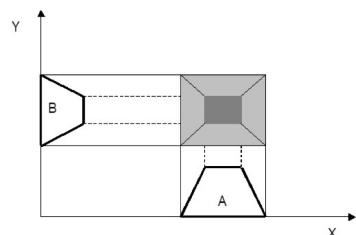
	New York	Paris
Beijing	1	.9
New York	0	.6
Rome	.7	.3

$$R(X, Y) = \{1/\text{NewYork}, \text{Beijing} + 0/\text{NewYork}, \text{NewYork} + \dots\}$$

Prodotto cartesiano tra fuzzy relations (fuzzy sets)

Data $A: X \rightarrow [0,1]$ e $B: Y \rightarrow [0,1]$, il loro prodotto $\underset{\vee}{\wedge}_{A \times B}: X \times Y \rightarrow [0,1]$

è una fuzzy relation dove $\mu_{A \times B}$ è un operatore T-norm, ad esempio il min ($A(x), B(y)$)



$$A = \frac{0.2}{x_1} + \frac{0.5}{x_2} + \frac{1}{x_3}$$

$$B = \frac{0.3}{y_1} + \frac{0.9}{y_2}$$

$$A \times B = R = \begin{bmatrix} y_1 & y_2 \\ x_1 & 0.2 & 0.2 \\ x_2 & 0.3 & 0.5 \\ x_3 & 0.3 & 0.9 \end{bmatrix} \leftarrow \text{Minimo!}$$

Standard composition tra fuzzy relations (standard max-min composition)

Data $R(x, y)$ e $S(y, z)$ con un set in comune (Y), la loro standard

Composition produce una relazione binaria $R \circ S$ su $X \times Z$, cioè:

$$(R \circ S)(x, z) = \max_{y \in Y} \min\{R(x, y), S(y, z)\}$$

Esempio

$$\begin{array}{c} \text{Da } y_1 \text{ e } y_2 \text{ sceglio il} \\ \text{massimo tra i minimi} \end{array}$$

$$\begin{array}{c} y_1 & y_2 \\ x_1 & [0.3 & 0.8] \\ x_2 & [0.6 & 0.9] \end{array} \circ \begin{array}{c} z_1 & z_2 \\ y_1 & [0.5 & 0.9] \\ y_2 & [0.4 & 1] \end{array} = \begin{array}{c} z_1 & z_2 \\ x_1 & [0.4 & 0.8] \\ x_2 & [0.5 & 0.9] \end{array}$$

$\min = 0.3$ $\min = 0.4$
 $\max = 0.4$

Sup-T composition tra fuzzy relations

Dato $R(X, Y)$ e $S(Y, Z)$ con un set in comune (Y), la sup-T composition, dove T fa riferimento alla t-norm, generalizza la standard max-min composition:

$$(R \circ S)(x, z) = \sup_{y \in Y} T\{R(x, y), S(y, z)\}$$

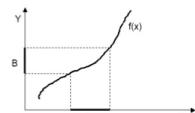
generalization of max

generalization of min

Cioè posso generalizzare la composizione ed usare il sup insieme a una qualsiasi t-norm.

Extension Principle

È un metodo usato per estendere una crisp function in una fuzzy function.

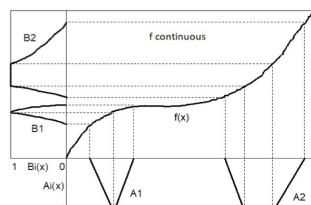


Dato una crisp function $f: X \rightarrow Y$ ed un fuzzy set A contenuto in X ,

tale che $A = \frac{\mu_A(x_1)}{x_1} + \dots + \frac{\mu_A(x_n)}{x_n}$, l'extension principle afferma che l'immagine

di A dato dal mapping della funzione f , può essere espressa come un fuzzy set B t.c.

$$B = f(A) = \frac{\mu_A(x_1)}{f(x_1)} + \dots + \frac{\mu_A(x_n)}{f(x_n)} \quad \text{dove } f(x_i) = y_i;$$



In generale:

Sia $f: P(X_1 \times X_2 \times \dots \times X_n) \rightarrow P(Y)$, quindi $B = f(A_1, A_2, \dots, A_n)$, otteniamo:

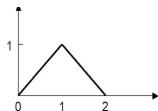
$$\mu_B(y) = \sup_{y=f(x_1, x_2, \dots, x_n)} \left\{ \min \left[\mu_{A_1}(x_1), \mu_{A_2}(x_2), \dots, \mu_{A_n}(x_n) \right] \right\}$$

Fuzzy numbers

Fuzzy numbers + usati zona Triangular e Trapezoidal

Un fuzzy number è un fuzzy set definito su \mathbb{R} con una membership function normale e convex.

Ese:



→ nappa. real nums
vicini a 1

$$\tilde{1} = \left\{ \frac{2}{0}, \frac{1}{1}, \frac{2}{2} \right\}$$

→ nappa. interi vicini
a 1

$$\tilde{1} + \tilde{1} = \left(\frac{2}{0}, \frac{1}{1}, \frac{2}{2} \right) + \left(\frac{2}{0}, \frac{1}{1}, \frac{2}{2} \right) =$$

$$\begin{aligned} & \min(0.2, 0.2) + \max[\min(0.2, 1), \min(1, 0.2)] \\ & + \max[\min(0.2, 0.2), \min(1, 1), \min(0.2, 0.2)] \end{aligned}$$

$$+ \max[\min(1, 0.2), \min(0.2, 1)], \min(0.2, 0.2)$$

$$= \frac{2}{0}, \frac{2}{1}, \frac{2}{2} + \frac{2}{3}, \frac{2}{4} = \tilde{2}$$

Siano I e J due fuzzy numbers e * una delle

4 op. aritmetiche $(+, -, \cdot, :)$, otteniamo:

$$(I^*J)(z) = \sup_{z=x*y} \min[I(x), J(y)]$$

Oss: I^*J è un fuzzy number

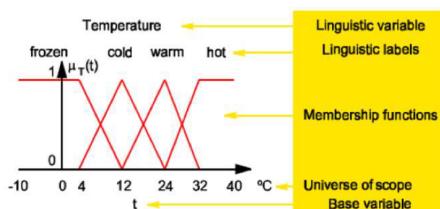
Esempio

Linguistic Variables

Rappresenta un termine del natural language

Una linguistic variable è una variabile i quali valori di essa sono fuzzy numbers.

Essa è definita in termini di base variable, cioè il suo universe of scope sono real numbers.



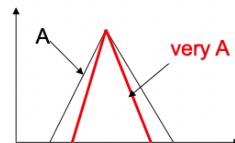
→ In questo esempio: "trapezoidal"

Posso avere anche dei linguistic hedges, i quali sono termini linguistici speciali che modificano altri linguistic terms (es: very, fairly, extremely ...).

Ogni linguistic hedge può essere considerato come una operazione unaria sull' intervallo $[0,1]$.

Ad esempio, l' hedge "very" può essere considerato come $h(a) = a^2$,

mentre "more or less" come $h(a) = \sqrt{a}$.



Fuzzy logic

Insieme di principi e regole di inferenza

La fuzzy logic è un'estensione della logica classica, la quale permette di esprimere un grado di verità compreso tra $[0,1]$.

Modus Ponens

La regola del modus ponens nella logica classica esprime che, dati P e Q ,

se P è vero ed implica Q , allora anche Q è vero.

$$(p \wedge (p \rightarrow q)) \rightarrow q$$

Il problema del modus ponens così definito è che non modelliamo i diversi gradi di verità, ad esempio:

x is very high (p)
if x is high, then y is low (p')

y is very low

Queste due proposizioni non coincidono, usano x diverse, non posso usare il modus ponens.

Possiamo quindi definire il Generalized Modus Ponens come segue:

$$(p' \wedge (p \rightarrow q)) \rightarrow q'$$

dove p' è leggermente differente da p e q' è leggermente differente da q .

Oss: p' , $p \rightarrow q$ e q' sono fuzzy propositions.

Fuzzy Propositions

Una fuzzy proposition può avere due forme:

- $X \text{ is } A \rightarrow X$ è una linguistic variable (temp, speed etc...) e A è un fuzzy set, il quale rappresenta il valore di X (high, medium, low...).

Per ogni valore x di X , il grado di verità della proposizione (p)

$$"X \text{ is } A" \text{ è dato da } T(p) = A(x) = M_A(x)$$

- "Se $X \in A$ allora $Y \in B$ " \rightarrow X e Y sono linguistic variables (temp, speed etc...) e A e B sono fuzzy sets.

$$X \text{ is } A \Rightarrow Y \text{ is } B$$

Questa seconda forma è chiamata fuzzy rule ed

è una fuzzy relation (anche chiamata "implication relation")

es: "se la lavatrice è piena a metà, allora il tempo di lavaggio è corto"

Sia la generalized modus ponens rule come segue:

$A' \rightarrow$ Proposition (fatto)

$A \rightarrow B \rightarrow$ Fuzzy rule

$$(A' \wedge (A \rightarrow B)) \Rightarrow B'$$

$B' \rightarrow$ Conclusion

abbiamo:

- A' è una relazione (unary relation sull'universo U dove A' è definita)
- $A \rightarrow B$ è una relazione (implication relation sugli universi U e V dove A e B sono definiti)
- B' è una relazione ottenuta dalla composizione delle due relazioni (definita su V)

In sintesi, abbiamo due fuzzy relations, A' e $A \rightarrow B$, che combinate producono B' .

Fuzzy implication

Una fuzzy implication I è una funzione $I : [0,1] \times [0,1] \rightarrow [0,1]$ la quale per ogni possibile valore di verità x e y delle rispettive fuzzy propositions (p, q) , produce un valore di verità, $I(x, y)$, per la conditional proposition "if p then q ".

Quando estendiamo le formule della logica classica per l'implicazione nella fuzzy logic, consideriamo la disjunction come t-norm, la conjunction come t-conorm e la negation come fuzzy complement. \rightarrow Quindi l'estensione della logica classica è diversa.

Ottieniamo quindi la implicazione nella fuzzy logic definita come segue:

■ Kleene-Dienes implication $I(a, b) = \max(1 - a, b)$

■ Lukasiewicz implication $I(a, b) = \min(1, 1 - a + b)$

■ Zadeh implication $I(a, b) = \max(\min(a, b), 1 - a)$

■ Mamdani implication $I(a, b) = \min(a, b)$ \rightarrow Questa non è un'estensione dell'impl. classica

■ Larsen implication $I(a, b) = a \cdot b$

Composizione di regole di inferenza

Sia la composizione così definita $B' = A' \circ (A \rightarrow B)$, abbiamo quindi

$$B'(y) = \sup_{x \in U} \min [A'(x), I(A(x), B(y))]$$

In generale posso usare una qualsiasi t-norm, quindi:

$$B'(y) = \sup_{x \in U} T[A'(x), I(A(x), B(y))] \quad \text{dove : } T \text{ è la t-norm}$$

Example

Suppose that the fact A' is a fuzzy singleton (with support x_0).

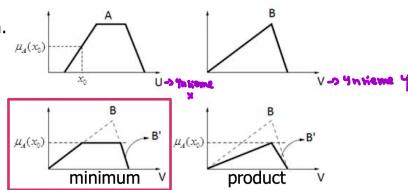
The approximate consequence B' is computed (with Mamdani implication) as

$$B'(y) = \sup_x \min \{A'(x), \min_x \{A(x), B(y)\}\} \quad \text{for all } y$$

Observing that $A'(x) = 0, \forall x \neq x_0$, the supremum turns into a simple minimum

$$B'(y) = \min \{A'(x_0), A(x_0), B(y)\} = \min \{1, A(x_0), B(y)\} = \min \{A(x_0), B(y)\} *$$

Similarly, in the case of Larsen implication.



Fuzzy Systems (Fuzzy inference Systems)

I fuzzy systems sono composti da un insieme di regole le quali sono dei conditional statement nei quali l'antecedent e il consequent sono fuzzy propositions contenenti delle linguistic variables e linguistic operators (es: and, or).

Single - input single output (SISO) fuzzy rules

Ogni regola produce un conseguente (fuzzy set) e le n

consequenze devono essere aggregate per produrre B'

$$\begin{aligned} 1) & \text{ Regole} \\ R_1 &: \text{if } X \text{ is } A_1 \text{ then } Y \text{ is } B_1 \\ R_2 &: \text{if } X \text{ is } A_2 \text{ then } Y \text{ is } B_2 \\ & \dots \\ R_n &: \text{if } X \text{ is } A_n \text{ then } Y \text{ is } B_n \end{aligned}$$

$$2) \text{ fatto } \} \text{ Fact: } X \text{ is } A'$$

$$3) \text{ Conclusion } \} \text{ Conclusion: } Y \text{ is } B'$$

Possiamo aggregare in due modi:

1) First infer then Aggregate (FitA)

Ogni regola è valutata individualmente e le conclusioni aggregate.

Dato un insieme di n regole $I(A_i(x), B_i(y))$ e un "fact" $A'(x)$, la conclusione è data da:

$$B^{(1)} = h(B'_1, \dots, B'_n) \quad \text{dove:}$$

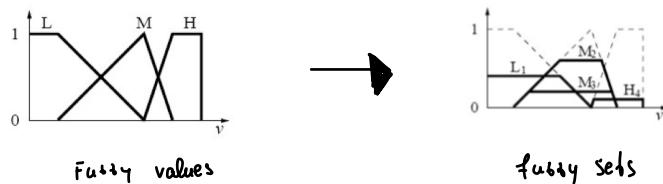
$$\bullet \quad B'_i(y) = A'(x) \circ I(A_i(x), B_i(y)) = \sup_{x \in U} T[A'(x), I(A_i(x), B_i(y))] \quad (1)$$

- h è un aggregation operator (solitamente min o max)

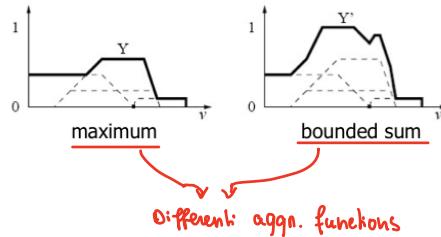
Esempio:

L'output variable di un fuzzy system ha tre fuzzy values: L, M, H.

In un dato istante, 4 regole vengono attivate con activation level 0.4, 0.6, 0.2, 0.1 producendo: fuzzy sets L_1, M_1, M_2, H_4 rispettivamente.



Il fuzzy set finale, aggregando le conclusioni delle regole, rappresenta un linguistico term.



Esso può essere usato così com'è o defuzzified per produrre un valore numerico.

Con il defuzzify vogliamo ottenere risultati precisi da info vaghe e imprecise.

2) First aggregate then infer (FATI)

L'insieme delle regole viene usato per generare una singola fuzzy relation e poi combinata con il "fact"

Dato un insieme di n regole $I(A_i(x), B_i(y))$ e un "fact" $A'(x)$, la
 conclusione è data da:

$$B^{(2)} = A'(x) \circ R(x, y) = \sup_{x \in U} T(A'(x), R(x, y)) \text{ dove:}$$

- $R(x, y) = h(I(A_1(x), B_1(y)), \dots, I(A_n(x), B_n(y)))$
- h è un aggregation operator (solitamente min o max)

Multi-Input single-output (MISO) fuzzy rules

- Nel caso multi-input, una fuzzy rule ha la forma:

$\text{if } X_1 \text{ is } A_1 \text{ and } X_2 \text{ is } A_2 \dots \text{ and } X_m \text{ is } A_m \text{ then } Y \text{ is } B_1 \rightarrow \text{Prima avevamo solo "X"}$

L'antecedent è una fuzzy relation dove T è una t-norm (solitamente "min").

$R(X_1, \dots, X_m) = T(A_1(x_1), \dots, A_m(x_m)) \rightarrow \text{Risolvendo gli and per ottenere una regola con un singolo fuzzy set nella "if-part"}$

- Il "fact" ha la forma:

$$X_1 \text{ is } A'_1 \text{ and } \dots \text{ and } X_m \text{ is } A'_m$$

Quindi abbiamo:

$$B'(y) = \sup_{x_1, \dots, x_m} t \left[T \left(A'_1(x_1), \dots, A'_m(x_m) \right), I \left(T \left(A'_1(x_1), \dots, A'_m(x_m) \right), B(y) \right) \right] \text{ dove } t \text{ e } T \text{ possono essere t-norm diff.}$$

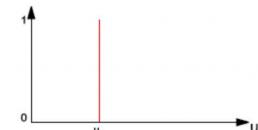
Aggrego: "fact" (A') implicazione ($A \Rightarrow B$)

Fuzzification e Defuzzification

Soltanente gli input di un fuzzy system sono crisp values e devono essere convertiti in fuzzy sets.

Fuzzification: Mapping da crisp value a fuzzy set

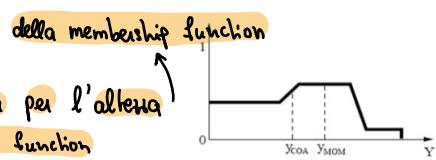
→ Il più usato è il singleton fuzzifier



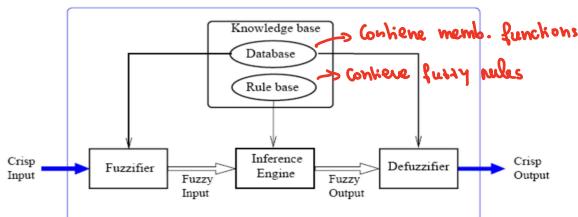
Defuzzification: Mapping da fuzzy set a crisp value

Tipici defuzzifiers:

- Center of Area (COA) → Divide l'area totale sotto la curva per l'altezza della membership function
- Mean of maxima (MOM) → Trova il max valore y della membership function e fa la media delle x corrispondenti.



Structure of a fuzzy system



Inference steps with FITA strategy:

1) Fuzzification dell' input

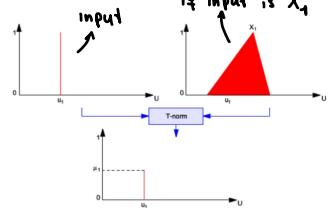
Valutiamo quanto l'input soddisfa il concetto logico rappresentato dall'antecedent fuzzy set

2) Valutazione dell' "activation strength" di ogni regola

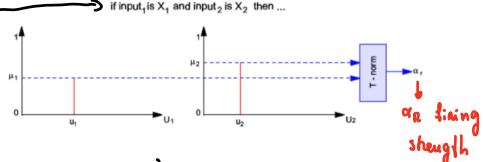
a) Calcola il "membership degree" di ogni input rispetto al fuzzy set contenuto nell' antecedente di ogni regola.

Viene fatto applicando una t-norm al fuzzified

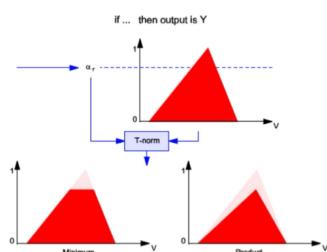
input e l'antecedent fuzzy sets.



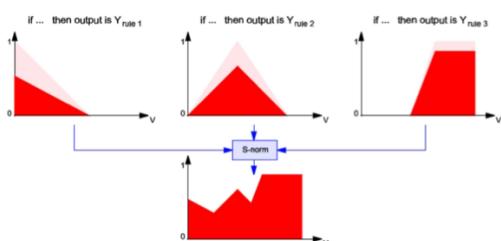
b) Viene applicata una t-norm ai membership degree ottenuti nello step (a) per ottenere la firing strength della regola



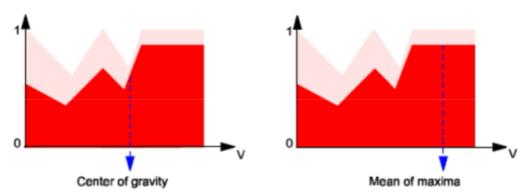
3) Implication e composition (assumiamo min or product implication)



4) Rule aggregation (assumiamo 3 regole)

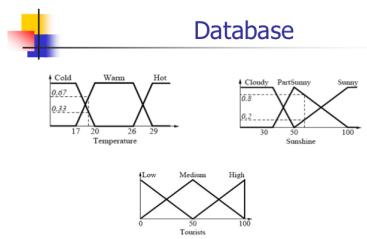


5) Defuzzification



Example: the tourist prediction system

- The fuzzy system predicts the number of tourists visiting a resort.
- 2 input variables: **Temperature** (in degrees) and **Sunshine** (a percentage of the maximum expected sunshine).
- 1 output variable: **Tourists**, which is the estimated amount of tourists (percentage of resort capacity).
- In the example, we refer to a slightly cold and partially sunny day: the temperature and sunshine are, respectively, 19 degrees and 60%.



Rule base

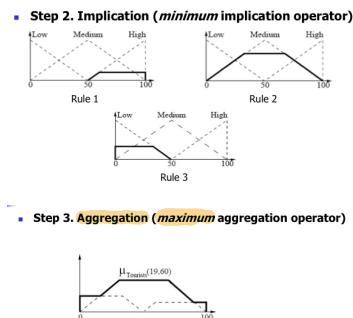
- Rule 1:** if (Temperature is Hot) or (Sunshine is Sunny) then (Tourists is High)
- Rule 2:** if (Temperature is Warm) and (Sunshine is Partially Sunny) then (Tourists is Medium)
- Rule 3:** if (Temperature is Cold) or (Sunshine is Cloudy) then (Tourists is Low)

Fuzzifier

Temperature	Sunshine
$\mu_{Cold}(19)=0.33$	$\mu_{Cloudy}(60)=0$
$\mu_{Warm}(19)=0.67$	$\mu_{PartSunny}(60)=0.8$
$\mu_{Hot}(19)=0$	$\mu_{Sunny}(60)=0.2$

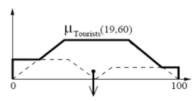
Inference engine

- Step 1. Antecedent activation**
- Rule 1:** if (Temperature is Hot) or (Sunshine is Sunny)
 $\mu_{Rule1} = \mu_{Hot}(19) \vee \mu_{Sunny}(60) = \max(0, 0.2) = 0.2$
- Rule 2:** if (Temperature is Warm) and (Sunshine is Partially Sunny)
 $\mu_{Rule2} = \mu_{Warm}(19) \wedge \mu_{PartSunny}(60) = \min(0.67, 0.8) = 0.67$
- Rule 3:** if (Temperature is Cold) or (Sunshine is Cloudy)
 $\mu_{Rule3} = \mu_{Cold}(19) \vee \mu_{Cloudy}(60) = \max(0.33, 0) = 0.33$



Defuzzifier

- COA defuzzification method



- The final output value is approximately 48%.

Fuzzy vs Probability

Fuzzy ≠ Probability

• Prob. deals uncertainty e likelihood

• Fuzzy logic deals with ambiguity e vagueness

Example 1

- John owns three houses.
- The probability that John owns four houses is zero.
- The fuzzy membership of John in the set of people with four houses, however, is non-zero.

Types of fuzzy rules

1) Hamdani fuzzy rule : if X_1 is A_{i1} and ... and X_m is A_{im} then Y is B_i

• Vantaggi: high interpretability

• Svantaggi: a) low accuracy, perché dipende dal defuzzification method scelto

b) high computational cost, perché usa funzioni non lineari (min e max)

2) Takagi - Sugeno - Kang (TSK) fuzzy rule

where $a_{i0}, a_{i1}, \dots, a_{im}$ are real numbers.

if X_1 is A_{i1} and ... and X_m is A_{im} then Y is $a_{i0} + a_{i1} X_1 + \dots + a_{im} X_m$

Il consequent è una funzione, solitamente lineare, delle input variables.

Ogni regola può essere quindi considerata come un local linear model.

L'output del sistema è una media pesata degli output di ogni regola.

- Vantaggi: High accuracy del Hamdani system perché l'output è una combinazione lineare degli output di ogni regola.
- Svantaggi: Low interpretability

Coming back to the tourist-prediction example

- Let us convert the system to a TSK system by replacing the three linguistic consequents – Low, Medium and High – with corresponding, accurately calculated linear functions of the input variables
- The three rule consequents become:

Rule 1: if ... then Tourists = $f_1(T, S) = 2T + 0.8S - 40$

Rule 2: if ... then Tourists = $f_2(T, S) = T + S - 23$

Rule 3: if ... then Tourists = $f_3(T, S) = 0.5T + 0.3S$

- With these consequents, the implication step produces the following predictions:

	Rule 1	Rule 2	Rule 3
Truth level μ_{Rule_i}	0.2	0.67	0.33
Estimated tourists	$f_1(19,60)=46\%$	$f_2(19,60)=56\%$	$f_3(19,60)=27.5\%$

- The aggregation step does not change these values. Hence, the defuzzifier produces:

$$\begin{aligned} output &= \frac{\sum_{i=1}^3 f_i(19,60) \cdot \mu_{Rule_i}}{\sum_{i=1}^3 \mu_{Rule_i}} \\ &= \frac{0.2 \times 46\% + 0.67 \times 56\% + 0.33 \times 27.5\%}{0.2 + 0.67 + 0.33} \\ &= 46.4\% \end{aligned}$$

3) Singleton fuzzy rule

Se il rule consequent è un valore costante.

Essa può essere considerata un caso speciale di una Mamdani / TSK fuzzy rule.

perché un valore costante è sia un singleton fuzzy set che una linear fun.

con i coeff. delle input variables uguali a 0.

Vantaggio: a) Tradeoff tra interpretability e accuracy.

b) Defuzzification richiede meno computazione

dovuto alla discrete representation dell'

output variable

Coming back to the tourist-prediction example

- The Mamdani-type system is easily converted to a singleton-type system by replacing the fuzzy values Low, Medium and High with their corresponding centers of areas: 0%, 50% and 100%.

- With these new consequents, the implication step produces:

	Rule 1	Rule 2	Rule 3
Estimated tourists	High=100%	Medium=50%	Low=0%

$$\begin{aligned} \text{The defuzzifier produces: } output &= \frac{\sum_{i=1}^3 f_i(19,60) \cdot \mu_{Rule_i}}{\sum_{i=1}^3 \mu_{Rule_i}} \\ &= \frac{0.2 \times 100\% + 0.67 \times 50\% + 0.33 \times 0\%}{0.2 + 0.67 + 0.33} \\ &= 44.4\% \end{aligned}$$

Fino ad ora abbiamo applicato fuzzy inference per modellare sistemi la cui rule structure è predeterminata dall'interpretazione dell'utente delle caratteristiche delle variabili del modello.

Ora, invece di avere necessariamente una predeterminata model structure e quindi scegliere

i parametri associati con una membership fun. arbitraria, questi parametri possono essere scelti

in modo da adattare le membership fun. agli input/output dati.

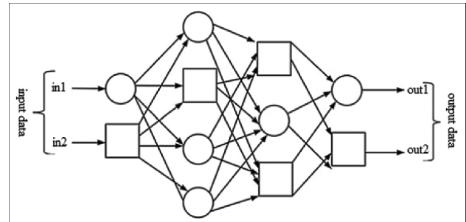
Per fare ciò useremo le Adaptive Networks.

Adaptive Networks

È una multi-layer feed forward network dove ogni nodo svolge una particolare funzione (node function) sugli incoming signals.

La rete è composta da:

- Adaptive node (square): Nodi con parametri.
- Fixed node (circle): Nodi senza parametri.
- Links: Flow direction del segnale tra i nodi.
↳ Non hanno peso



Poi ottengono un input-output mapping vengono updated i parametri usando dei training data e il gradient descent.

Adaptive-Network-based Fuzzy inference system (ANFIS)

ANFIS sono le Adaptive-Networks che sono functionally equivalent ai fuzzy inference systems.

Sia il nostro fuzzy system composto da due input (X e Y) e un output f .

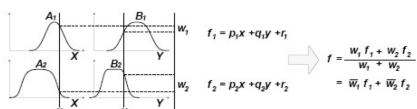
Inoltre, la rule base contenga due regole di tipo TSU:

Rule1: if X is A_1 and Y is B_1 then $f_1 = p_1x + q_1y + r_1$

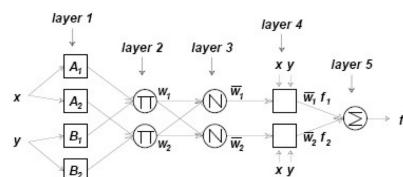
Rule2: if X is A_2 and Y is B_2 then $f_2 = p_2x + q_2y + r_2$

Ottieniamo che:

- Fuzzy reasoning



- ANFIS architecture



Functionally equivalent to

a TSU-type fuzzy inf. system

Layer 1: Square nodes con node function $O_i^1 = \mu_{A_i}(x) \rightarrow$ i parametri della fun. sono chiamati premise parameters

Layer 2 : Circle nodes i quali moltiplicano gli incoming signals e lo danno in output.

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \quad i=1,2$$

Layer 3 : Circle nodes i quali calcolano il rapporto tra la i -th rule's fine strength e la somma delle rules' fine strengths

$$\bar{w}_i = \frac{w_i}{w_1 + w_2}, \quad i=1,2$$

Layer 4 : Square nodes con node function $O_i^4 = \bar{w}_i f_i = \bar{w}_i (p_i x + q_i y + r_i)$ dove \bar{w}_i è l'output del layer 3 e p_i, q_i, r_i sono chiamati consequent parameters.

Layer 5 : Single circle node (output) il quale calcola la media pesata degli incoming signals del layer 4.

$$O_1^5 = \text{overall output} = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

Network Training

Possiamo usare un algoritmo di learning ibrido, il quale combina il gradient method e il least square estimate (LSE).

Ad ogni epoca l'algoritmo consiste in due passi:

1) Forward step : Dat: i valori dei premise parameters, vengono dati in input i dati e i functional signals vanno forward fino al layer 4 per calcolare ogni node output.

Gli consequent parameters vengono identificati tramite la LSE.

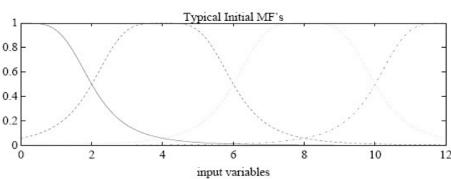
2) Backward step : I error signals vengono propagati all'indietro e i premise parameters vengono aggiornati con il gradient descent.

-	forward pass	backward pass
premise parameters	fixed	gradient descent
consequent parameters	least squares estimate	fixed
signals	node outputs	error rates

Initial Membership function setting

Le numeri di membership functions è scelto empiricamente e/o facendo "try and error".

Tipicamente, i valori initiali del MFs sono equally spaced ed inoltre il fuzzy system fornisce una smooth transition e un sufficiente overlapping da un linguistic label a un altro.



Modellare una two-input non-linear function

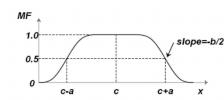
L'obiettivo è usare un ANFIS model per modellare una non linear sinc equation:

$$z = \text{sinc}(x, y) = \frac{\sin(x)}{x} \times \frac{\sin(y)}{y}$$

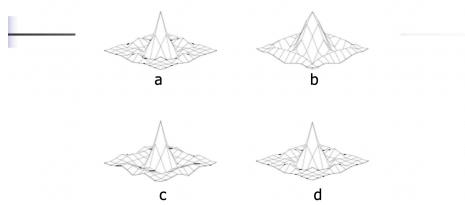
Come membership function usiamo una bell function con 3 parametri:

- c è il centro della funzione
- a è metà dell'altezza
- b, insieme ad a, controlla la slope nel crossover point (dove MF = 0.5)

$$\mu_a(x) = \frac{1}{1 + \left[\frac{(x - c)^2}{a^2} \right]^b}$$

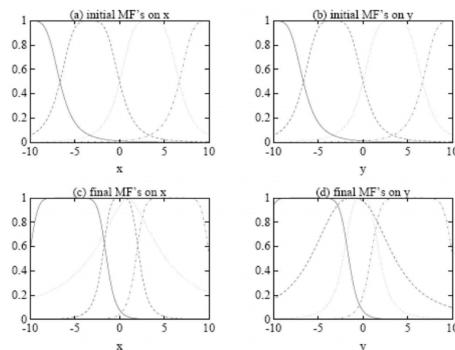


La ANFIS usata contiene 16 regole con 4 membership functions per ogni input variable, abbiamo quindi 24 premise params e 48 consequent params, per un totale di 72 parametri.



- Training data (a) and reconstructed surfaces at 0.5 (b), 99.5 (c) and 249.5 (d) epochs.

(Note that since the error is computed after the forward pass, the epoch numbers always end with "5". Further, the surface after 0.5 epochs is only due to the identification of consequent parameters).



Genetic Algorithms

Genetic algorithms sono metodi di ricerca e ottimizzazione che mimano la "natural evolution" la quale è basata sui processi fondamentali:

- La selezione, la quale sceglie gli individui che producono la prole.
 - La recombinação (crossover), il quale combina due individui per produrre la prole.
 - La mutazione, la quale è la variazione randomica del materiale genetico esistente.
- I Genetic Algorithms performano una random search con lo scopo di migliorare la probabilità di trovare global optimum solutions.

Definizioni:

- Un GA considera una popolazione di individui, dei quali ognuno di essi può essere una soluzione all'optimization problem.
- Gli individui sono composti da un insieme di genes (attributi) e corrispondono a un punto nel search space.
- Per valutare la qualità di un individuo rispetto all'optimization task è definito da una scalar objective function, chiamata fitness function.

Genetic Algorithms Search

La ricerca viene svolta andando a cambiare la popolazione di individui, generazione dopo generazione.

Selezione : Vengono selezionati i migliori individui (con le migliori caratteristiche) sull'intera popolazione

Crossover : Combinare due individui (parents) per produrre uno nuovo (prole)

Mutation : La mutazione della prole appena generata permette di introdurre variabilità nella popolazione e prevenire convergenza prematura verso soluzioni subottimali.

GA algorithms permettono un buon tradeoff tra exploitation (local search, se svolta troppo può portare a early convergence) e exploration (global search, se fatta troppo può ignorare info already disponibili).

L'obiettivo di un GA è l'ottimizzazione della fitness function, di conseguenza ci deve essere abbastanza "fitness difference" tra gli individui della popolazione per poter fare la ricerca.

GA search :

- Ad ogni iterazione t l'algoritmo crea una popolazione (generation). Solitamente le generazioni hanno tutte la stessa dimensione (num. di individui).
- La nuova generazione dovrebbe contenere individui migliori (miglior fitness value).
- Con la generazione iterativa di nuove generazioni, possiamo osservare un trend che va verso il global opt. della fitness function.

Genetic Algorithm elements

1) **Evaluation :** GA valuta la fitness di ogni individuo della popolazione

2) **Selection :** Vengono selezionati gli individui della popolazione $P(t)$ basandosi sulla fitness. Essi rappresentano la intermediate population P_i e entrano nella mating pool con

una data probabilità (crossover probability).

3) Recombination e modification

Gli individui nella mating pool vengono accoppiati usando il crossover e generano una nuova popolazione intermedia (P_2).

Gli individui in P_2 (prole) vengono modificati con la mutazione con una data probabilità producendo la popolazione P_3 .

La generazione per $P(t+1)$ contiene gli individui in P_3 e potrebbe includere altri individui selezionati.

Possible termination conditions:

- Massimo numero di generazioni
- Definire un "fitness" ideale e viene ottenuto dal best individual nella population
- Generazioni consecutive senza cambiamenti
- Average fitness di tutti gli individui maggiore di una certa percentuale (97/98%) del fitness del best individual.

Soluzione: Quando abbiamo la terminazione la soluzione al problema viene considerata il miglior individuo dell'ultima generazione.

È utile però mantenere anche il best individual ad ogni iterazione (t) perché il best individual potrebbe essere ottenuto prima dell'ultima generazione.

Questo può essere ottenuto usando l'"elitism", cioè una percentuale dei filisti individuali della generazione corrente vengono copiati nella prossima generazione.

Selection

La selezione può avvenire in due modi:

- Proportional selection: Ogni individuo (i) della popolazione corrente ha una probabilità $p(i)$

$$p(i) = \frac{f(i)}{\sum_{j=1}^n f(j)}$$

di essere selezionato in base alla sua fitness $f(i)$

```
S1. Set t = 0
S2. Initialize a chromosome population P(t)
    (Random initialization is the basic choice)
S3. Evaluate P(t) using a fitness measure
S4. while (termination condition not satisfied) do
begin
    S4.1. Select for recombination chromosomes from P(t)
        Let P1 be the set of selected chromosomes
        Choose individuals from P1 to enter the mating pool (MP)
    S4.2. Recombine the chromosomes in MP to form population P2
    S4.3. Mutate chromosomes in P2 to form population P3
    S4.4. Select for replacement from P3 and P(t) to form P(t+1)
    Set t = t + 1
end
```

- **Tournament selection:** **K** individui vengono selezionati randomicamente dalla popolazione e svolto n times il fittest di essi è considerato per la riproduzione.

La **selection pressure** è il grado con il quale i migliori individui vengono favoriti dal **selection operator** (**Migliore \Rightarrow favoured**).

La **selection pressure** influenza sulla convergence rate del GA e ha un importante effetto sul tradeoff tra **exploitation** e **exploration** cioè:

- **Selection pressure alto** \Rightarrow premature convergence to local optimum
- **Selection pressure basso** \Rightarrow GA takes longer per trovare la soluzione ottimale

Crossover

Possa avere due tipologie di crossover:

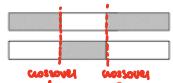
- 1) **Binary Encoding Crossover**: Genes di un individuo rapp. come una stringa di numeri binari.

a) One-point crossover

parents		11001
		10110
offspring	→ SWAP	11110
	Crossover point	10001

Seleziona un crossover point dopo il quale scambia i geni dei parents per produrre i due offsprings.

b) Two point crossover



Seleziona due crossover point dopo il quale scambia i geni dei parents per produrre i due offsprings.

c) Uniform crossover

- Per ogni gene del primo offspring, viene scelto, con una data probabilità, il parente che darà il valore.
- Per il secondo offspring, prendiamo il valore dall'altro parente non scelto per quel determinato gene.

OSS: I geni di ogni offspring possono essere calcolati anche indipendentemente, quindi, un parent può dare il valore di un gene a entrambi i discendenti.

VARIANTE

2) Real-valued encoding crossover

Ogni individuo (cromosoma) è un vettore di numeri reali (ognuno è un gene (attributo)).

a) Discrete crossover: Per ogni posizione i del primo offspring, scegliamo con probabilità fissata il parente che trasmetterà al discendente l' i -esimo gene.

Alla corrispondente posizione del secondo offspring verrà dato il valore del corrispondente gene dell'altro parente.

b) Average crossover: Alcuni geni vengono scelti ^{RANDOM} e i corrispondenti geni nei discendenti rappresentano la media aritmetica dei corrispondenti geni dei parenti.

Esempio: Gene 3 e 5 selezionati

$$\text{parents} \quad x = (x_1, x_2, x_3, x_4, x_5) \\ y = (y_1, y_2, y_3, y_4, y_5)$$

$$\text{offspring} \quad x' = (x_1, x_2, (x_3 + y_3)/2, x_4, (x_5 + y_5)/2) \\ y' = (y_1, y_2, (x_3 + y_3)/2, y_4, (x_5 + y_5)/2)$$

c) Convex crossover: I discendenti sono rappresentati come una combinazione convessa dei parenti.

- the i -th gene of the only descendant of chromosomes x and y is:

$$z_i = \alpha x_i + (1-\alpha) y_i, \quad \alpha \in [0, 1]$$

- the i -th gene of the descendants u and v are:

$$u_i = \alpha x_i + (1-\alpha) y_i \\ v_i = \alpha y_i + (1-\alpha) x_i$$

Oss: La crossover probability assume valori generalmente dell'ordine di 10^{-1}

Mutation

Binary encoding mutation

- The effect of mutation is to change a single gene (or, possibly, more genes) within a chromosome. Mutation ensures that the full range of gene values is available for the search.
- In **strong mutation**, the position selected for mutation automatically changes its value. In **weak mutation**, the selected position changes value with a certain probability.

Real-valued encoding mutation

- We can have **uniform** and **non-uniform** mutation. The action of a non-uniform operator depends on the generation.

Uniform mutation

- One-position mutation:** a single (randomly chosen) gene is replaced with a randomly generated real number within the domain of the corresponding parameter.
- All-positions mutation:** all genes are similarly perturbed. There are several methods, e.g., an additive normal mutation:

$$x'_i = x_i + \alpha_i N(0, \sigma_i)$$

where the real parameters α_i and the standard deviations σ_i can be the same or different for the genes considered.

Non-uniform mutation

- Genes undergo significant changes in the first generations. Then the changes gradually decrease. Therefore, in the initial stage of the search process, important progress is made and in the last phases there is a refinement, or fine control, of the search process.

Small values of mutation probability p_m are traditionally used ($p_m \in [0.001, 0.01]$).

Solving an optimization problem using a GA

- The following five issues need to be addressed:
 - a genetic representation of the candidate solutions,
 - creating an initial population of solutions,
 - definition of the fitness function,
 - choice of selection operator and genetic operators,
 - choice of the parameters of the GA, e.g., population size, number of generations, probabilities of genetic operators.