

# **Machine Learning**

Piero Viscone

Appunti del corso tenuto dal Prof.  
Alessio Micheli

# Indice

<b>Indice</b>	<b>2</b>
<b>1 Introduzione al Machine Learning</b>	<b>1</b>
1.1 Dati . . . . .	1
Terminologia . . . . .	2
1.2 Task . . . . .	2
Supervised Learning . . . . .	3
Unsupervised Learning . . . . .	3
Altri tipi di task . . . . .	3
1.3 Modello . . . . .	4
1.4 Algoritmi di apprendimento . . . . .	4
Bias . . . . .	5
Loss . . . . .	6
Overfitting . . . . .	7
Statistical Learning Theory . . . . .	8
Validazione . . . . .	9
1.5 Recap . . . . .	10
Design Cycle . . . . .	10
Pitfalls . . . . .	11
<b>2 Modello lineare</b>	<b>12</b>
2.1 Regressione . . . . .	12
2.2 Classificazione (binaria) . . . . .	13
Loss function . . . . .	13
2.3 Algoritmo di apprendimento . . . . .	14
Approccio diretto . . . . .	14
Gradient Descent: approccio iterativo . . . . .	17
2.4 Considerazioni . . . . .	19
2.5 Generalizzazione . . . . .	19
2.6 Regolarizzazione: controllo della complessità . . . . .	20
Approccio diretto . . . . .	20
Gradient Descent . . . . .	21
<b>3 K Nearest Neighbor</b>	<b>23</b>
3.1 1-nn . . . . .	23
3.2 k-nn . . . . .	23
Multiclass . . . . .	25
K-nn pesato . . . . .	25
Generalization capabilities . . . . .	25
Inductive bias . . . . .	26
3.3 K-nn vs Linear model . . . . .	27
3.4 Limiti del knn . . . . .	27
<b>4 Neural networks</b>	<b>29</b>
4.1 Perceptron . . . . .	29
Linear threshold unit (McCulloch & Pitt) . . . . .	29

Algoritmo di apprendimento . . . . .	31
Differenze tra LMS e Perceptron learning algorithm . . . . .	32
Perceptron Convergence Theorem . . . . .	33
4.2 Neural Network Unit . . . . .	34
Funzione di attivazione . . . . .	34
Apprendimento . . . . .	35
4.3 MultiLayer Perceptron . . . . .	35
Interpretazione del modello . . . . .	36
NN come approssimante . . . . .	36
Problemi generali . . . . .	37
4.4 Backpropagation . . . . .	37
4.5 Problemi nel training delle NN . . . . .	40
Minimi multipli . . . . .	40
Tipo di apprendimento . . . . .	41
Rate di apprendimento . . . . .	41
4.6 Miglioramenti nell'apprendimento . . . . .	42
Momentum . . . . .	42
Learning rate variabile . . . . .	43
Criteri di stopping . . . . .	45
4.7 Overfitting e regolarizzazione . . . . .	45
4.8 Numero di unità . . . . .	46
Cascade correlation . . . . .	46
4.9 Input/Output . . . . .	48
<b>5 Validation</b>	<b>50</b>
Dati e sampling . . . . .	50
Grid Search . . . . .	51
5.1 Tecniche di validazione . . . . .	51
Hold out validation . . . . .	51
K fold cross validation . . . . .	52
Double kfold cross validation . . . . .	53
Quale scegliere? . . . . .	54
5.2 Pitfalls e riflessioni . . . . .	55
Early stopping . . . . .	55
Weight inizialization . . . . .	55
Selezione sequenziale . . . . .	55
<b>6 Bias-Variance decomposition</b>	<b>56</b>
6.1 Regolarizzazione . . . . .	57
6.2 Ensemble Learning . . . . .	57
Bagging . . . . .	57
Boosting . . . . .	58
<b>7 Statistical learning theory</b>	<b>59</b>
7.1 Shattering and VC dimension . . . . .	59
7.2 Bound analitici sul rischio . . . . .	60
Structural risk minimization . . . . .	60
<b>8 Support Vector Machine</b>	<b>62</b>
8.1 Hard margin SVM . . . . .	62
8.2 Soft margin SVM . . . . .	66

8.3	Non linear classification . . . . .	68
	Kernel Trick . . . . .	69
8.4	Non Linear Regression . . . . .	69
	Recap . . . . .	71
8.5	Osservazioni . . . . .	71
<b>9</b>	<b>Convolutional Neural Networks</b>	<b>73</b>
9.1	Padding . . . . .	74
9.2	Pooling . . . . .	74
9.3	Architettura . . . . .	75
9.4	Recap e osservazioni . . . . .	75
<b>10</b>	<b>Deep Learning</b>	<b>77</b>
10.1	Numero di Layer . . . . .	78
	Inductive bias . . . . .	79
	Curse of the dimensionality . . . . .	79
	Manifold learning . . . . .	80
	Recap . . . . .	80
10.2	Representation learning . . . . .	81
	PreTraining . . . . .	81
	Transfer Learning . . . . .	82
10.3	Distributed Representation . . . . .	83
10.4	Further topics . . . . .	84
10.5	Tecniche . . . . .	85
	Gradient Issues . . . . .	85
	ReLU . . . . .	85
	Batch Normalization . . . . .	86
	Dropout . . . . .	86
	L1 regularization . . . . .	87
<b>11</b>	<b>Randomized Machine Learning</b>	<b>88</b>
11.1	Randomized Neural Network . . . . .	88
11.2	Conclusioni . . . . .	89
<b>12</b>	<b>Unsupervised Learning</b>	<b>90</b>
12.1	Clustering . . . . .	90
	Vector quantization and K mean . . . . .	90
	Self organizing map: SOM . . . . .	92
<b>13</b>	<b>Recurrent Neural Networks</b>	<b>94</b>
13.1	Input delay neural network: IDNN . . . . .	94
13.2	RNN unit . . . . .	95
13.3	Apprendimento . . . . .	96
13.4	Echo State Network . . . . .	96

# Introduzione al Machine Learning 1

L'obiettivo del Machine Learning è di ricavare un modello (che sia empiricamente predittivo) da un set di dati che ci permetta di generalizzare un dato fenomeno altrimenti difficilmente descrivibile

Alcuni casi di applicazioni possono essere:

- ▶ Mancanza di conoscenze delle leggi che governano un dato fenomeno
- ▶ Eccessiva complessità analitica delle leggi esistenti
- ▶ Presenza di dati a cui è sovrapposto rumore
- ▶ Applicazione a comportamenti "personalizzati" es. (Riconoscimento dello spam)

"Si dice che un programma apprende dall'esperienza E con riferimento a alcune classi di compiti T e con misurazione della performance P, se le sue performance nel compito T, come misurato da P, migliorano con l'esperienza E." [6]

Quindi si può dire che un programma apprende se c'è un miglioramento delle prestazioni dopo un compito svolto.

Il concetto cruciale del machine learning è la **capacità di generalizzare** ovvero la capacità di fare predizioni su nuovi dati.

Le due fasi principali che compongono il processo di generalizzazione sono:

- ▶ **Fase di apprendimento (training):** Fase in cui il modello viene costruito a partire dai dati
- ▶ **Fase predittiva (deployment):** Applicazione del modello a nuovi dati
- ▶ **Test:** Valutazione delle performance del modello, ovvero dell'accuratezza delle nuove predizioni

Gli ingredienti principali per costruire un algoritmo di apprendimento automatico sono:

1. **Dati:** Misure da cui il modello deve apprendere
2. **Modello:** Definizione delle ipotesi
  - ▶ **Task:** Definisce gli scopi e gli obiettivi dell'algoritmo
  - ▶ **Apprendimento:** Meccanismo con cui il sistema sceglie o migliora il modello apprendendo dai dati
  - ▶ **Validazione:** Valutazione e miglioramento delle capacità di generalizzare
3. **Predizione**

## 1.1 Dati

I dati da cui l'algoritmo deve apprendere possono essere rappresentati in vari modi:

- Rappresentazione strutturata: liste, alberi, immagini, grafi, etc.
- Rappresentazione vettoriale: ogni dato è un vettore e ogni suo elemento costituisce una **feature** del dato

### Flat data

Una feature può assumere un valore sia continuo che discreto.

Nel caso di valore discreto la feature può essere rappresentata in diversi modi (**Data Encoding**)

- 1,2,3,...,k nel caso in cui la feature sia "ordinabile" (es. piccolo, medio, grande)
- **1-of-k**: Utile nel caso la feature sia un simbolo. Ad ogni possibile valore della feature si associa un vettore della base ortonormale di  $\mathbb{R}^k$

<b>A</b>	<b>1</b>	0	0
<b>B</b>	0	<b>1</b>	0
<b>C</b>	0	0	<b>1</b>

**Figura 1.1:** Rappresentazione 1-of-k (o 1-hot)

### Terminologia

- Def. 1.1.**
- **Rumore**: Sovrapposizione ai dati di variabili aleatorie dovute alla stocasticità associata al processo di misura o di errori sistematici.
  - **Outliers**: Singoli dati che si discostano significativamente dall'andamento della quasi totalità dei dati a causa di errori nel processo di misura. (La loro rimozione deve SEMPRE essere motivata statisticamente in modo quantitativo)
  - **Feature selection**: Scelta di un numero limitato di feature in modo tale da fornire la migliore rappresentazione possibile dei dati per il problema scelto
  - **Data preprocessing**: Insieme di tecniche atte alla manipolazione dei dati al fine di ridurre il rumore, aumentare il numero di dati creandone nuovi da quelli esistenti,etc.

## 1.2 Task

L'obiettivo di un algoritmo di apprendimento automatico può essere suddiviso in due macrocategorie:

- Supervised Learning
- Unsupervised Learning

## Supervised Learning

Dati dei dati in input **Etichettati** (ovvero associati a una classe) e una funzione di target  $f$  sconosciuta che fitti i dati e generalizzi il fenomeno, l'obiettivo dell'apprendimento supervisionato è quello di trovare una buona approssimazione a  $f$  (magari anche fissando delle ipotesi sul modello).

Il target della funzione (codominio) completa la definizione del task:

- **Classificazione (supervisionata):**  $f(\mathbf{x}) \in \{1, 2, \dots, k\}$   
Ovvero l'algoritmo predice l'appartenenza di un dato a una classe in modo discreto  
La classificazione può essere vista come la separazione dello spazio degli input in regioni appartenenti a classi diverse

**Esempio 1.2.**

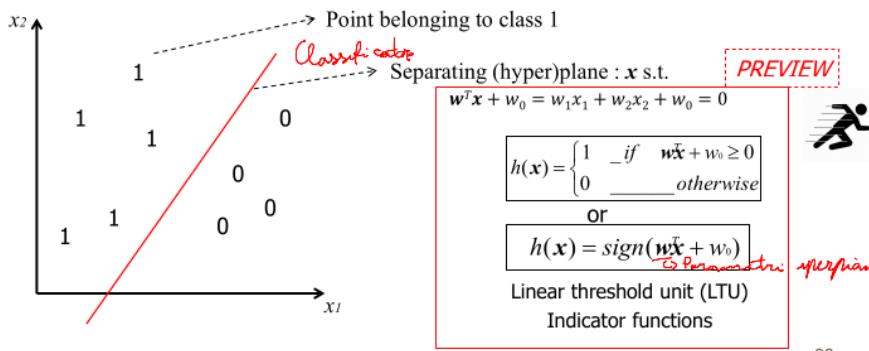


Figura 1.2: Classificazione binaria lineare in 2D

- **Regressione**  $f(\mathbf{x}) \in \mathbb{R}^k$   
Ovvero l'algoritmo approssima una funzione a valori reali continui.  
Una applicazione può essere, ad esempio, il curve-fitting

## Unsupervised Learning

Nell'apprendimento non supervisionato non è associato alcuna etichetta ai dati e l'obiettivo è trovare dei raggruppamenti di dati principalmente tramite tecniche di clustering

## Altri tipi di task

Altre possibilità di task sono:

- **Semip-supervised Learning:** combina l'approccio supervised a quello unsupervised con dati solo parzialmente labellati
- **Reinforcement Learning:** Usato soprattutto in sistemi autonomi per effettuare delle scelte. L'algoritmo impara tramite un sistema di "ricompense" e "punizioni" date in conseguenza ad una data azione

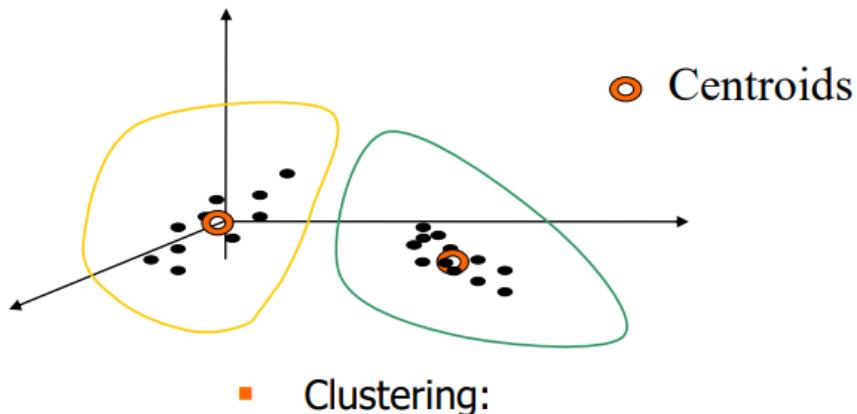


Figura 1.3: Clustering

### 1.3 Modello

Il modello ha l'obiettivo di descrivere le relazioni tra i dati e definisce un insieme di funzioni  $h(\mathbf{x}, \mathbf{w})$  dove  $\mathbf{w}$  sono i parametri associati.

- Def. 1.3.**
- *Target function f*: La funzione  $f$  (sconosciuta) che si vuole trovare
  - *Esempio di training (superv.)*: È la coppia (dato,label)  $(\mathbf{x}, f(\mathbf{x}) + \text{noise})$
  - *Ipotesi*: Una funzione  $h$  che si ipotizza approssimi  $f$
  - *Spazio delle ipotesi*: Spazio di tutte le ipotesi. (Spazio dei parametri)

**Achtung!! 1.4.** *NON ESISTE UN METODO DI APPRENDIMENTO MIGLIORE IN ASSOLUTO:*

**Teorema 1.5.** (*No free lunch theorem*): Se un algoritmo mostra risultati migliori in una situazione allora ne mostrerà inferiori in un'altra

Non tutti i modelli sono equivalenti: bisogna sempre bilanciare, in base alla situazione, la **flessibilità** e la **complessità** del modello.

### 1.4 Algoritmi di apprendimento

L'algoritmo di apprendimento ricerca la migliore ipotesi, ovvero la migliore approssimazione della funzione  $f$ , all'interno dello spazio delle ipotesi  $\mathcal{H}$

**Achtung!! 1.6.**  $\mathcal{H}$  potrebbe non coincidere con l'insieme di tutte le funzioni e la ricerca potrebbe non essere esaustiva

## Bias

Per creare un algoritmo di apprendimento possiamo fare delle assunzioni:

- **Language Bias:** Restringere lo spazio delle ipotesi  $\mathcal{H}$
- **Search Bias:** Definire delle preferenze nella strategia di ricerca della migliore ipotesi
- Entrambe

**Def. 1.7. Ipotesi consistente:** Un ipotesi  $h$  si dice consistente con il training set  $T$  se e solo se  $h(x) = d(x) \forall \langle x, d(x) \rangle \in T$

**Def. 1.8. Version space:** Definiamo il Version Space  $VS_{H,T}$  rispetto allo spazio delle ipotesi  $H$  e al training set  $T$  il sottoinsieme di  $H$  contenente tutte le ipotesi consistenti con  $T$

### Esempio 1.9. Funzione booleana:

Vogliamo creare un algoritmo di apprendimento che, dati 4 valori binari in input e un valore binario di label, riesca a ricavare la legge booleana che mappa le 4 feature nel label.

**Osservazione:** Questo problema è malposto, potremmo violare esistenza, unicità e stabilità delle soluzioni.

Lo spazio delle ipotesi discreto ha  $2^{24}$  possibili funzioni booleane. Se nel set di training abbiamo 7 esempi le funzioni che rispettano quelle 7 regole sono comunque  $2^9$ .

Nel caso generale il numero di possibili funzioni sono  $2^{2^n-k}$  dove  $n$  è il numero delle features (dimensione dell'input) e  $k$  sono il numero di esempi (dimensione del training set)

Example	$x_1$	$x_2$	$x_3$	$x_4$	$y$
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Questo significa che senza fare nessuna assunzione il modello non può fare alcuna predizione su nuovi input che non appartengano al training set. Quella che si ottiene è una **Lookup table**, ovvero una tabella in cui tutte le possibilità al di fuori del training set hanno output ignoto.

**Nessun inductive bias  $\implies$  Nessuna generalizzazione**

**Esempio 1.10. Conjutive Rule:**

Assumiamo ora che la funzione booleana sia composta da una serie di proposizioni legate da AND. Stiamo ponendo un **Inductive Bias**

Lo spazio delle ipotesi è composto da:

$$\mathcal{H} = \{l_2, l_1 \& l_2, true, false, \neg l_1 \& l_2, \dots\}$$

La dimensione dello spazio delle ipotesi è  $3^n + 1$ , dove n è il numero di features. (per ognuna delle n posizioni possiamo avere  $l_i, \neg l_i$ , don't care e +1 perché  $l_i \& \neg l_i = \text{false}$ )

**Osservazione 1.11.** Per 4 feature ora la dimensione dello spazio delle ipotesi è passato da 65536 a 82. Forse il bias è troppo restrittivo, la funzione di target potrebbe non essere nello spazio delle ipotesi. (es. Se la target function è  $x_1 \text{ OR } x_2$ )

**Teorema 1.12.** Un algoritmo di apprendimento unbiased è incapace di generalizzare su input esterni al training set

*Dimostrazione.* Dato T il training set e H spazio delle ipotesi

$$\forall h \in VS_{H,T} \exists h' \in VS_{H,T} \text{ t.c. } \exists x \notin T, h(x) \neq h'(x)$$

□

Un learner unbiased è capace di classificare in modo non ambiguo solo gli elementi presenti nel training set (si ottiene quindi la lookup table)

**Il bias non è solo assunto per migliorare l'efficienza ma è strettamente necessario affinché l'algoritmo sia in grado di generalizzare**

Ovviamente il bias va adattato ad ogni diversa situazione/approccio.

**Achtung!! 1.13.** Di solito Il search bias è preferito all'inductive bias. Questo perchè permette approcci più flessibili non escludendo nessuna soluzione a priori.

## Loss

Per trovare la migliore ipotesi dobbiamo definire una quantità che misura l'approssimazione della soluzione rispetto la target function.

Definiamo quindi una **loss function** (interna)  $\mathcal{L}(h(x), d)$  che misura la "distanza/errore" di una data ipotesi. (valore elevato  $\implies$  scarsa approssimazione)

**Def. 1.14. Errore:**

$$E(\mathbf{w}) = \frac{1}{l} \sum_{p=1}^l \mathcal{L}(h(x_p), d_p)$$

dove  $p$  è l'indice del sample,  $l$  il numero di sample,  $h$  l'ipotesi e  $d_p$  il valore atteso

La loss function va adattata alla natura del problema da risolvere e allo spazio delle ipotesi scelto.

Alcuni esempi possono essere:

- Caso di **Regressione**: Pearson  $\chi^2$  test
- Caso di **Classificazione binaria**: una funzione che torna 1 se la classificazione è errata, o altrimenti
- Caso di **Clustering**: Distanza di ogni sample dal proprio centroide

Tipicamente la migliore ipotesi verrà cercata minimizzando l'errore.

## Overfitting

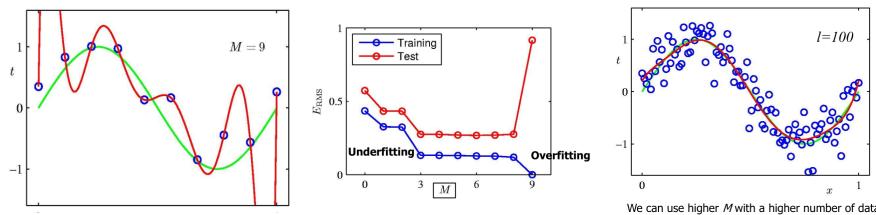
**Def. 1.15. Overfitting:** Data  $h$  la migliore ipotesi trovata con associato un errore sul training set  $E$  e un errore su tutti i dati (compreso test set)  $R$  (rischio), vi è overfitting se  $\exists h' \in H$  con loss  $E'$ ,  $R'$  t.c  $E' > E$  ed  $R' < R$ . (Cioè  $h'$  è "migliore" nonostante fitti peggio)

Consideriamo un caso particolare di regressione: curve fitting polinomiale di grado  $M$  tramite la minimizzazione dei minimi quadrati (usata come error function). La **complessità** dell'ipotesi scala con il grado del polinomio.

I dati sono affetti da rumore quindi usare un polinomio di grado troppo elevato (comparabile al numero di samples) ci darà un modello molto flessibile ma che rischia di assecondare troppo le fluttuazioni statistiche.

In questo caso conviene tenere d'occhio i gradi di libertà del sistema che in questo caso sono  $N_{dof} = N - (M + 1)$  dove  $N$  è il numero dei sample.

Avere un  $N_{dof}$  troppo basso porterà quasi sicuramente ad overfitting.



**Osservazione:** Nel caso di overfitting il valore dei parametri è decisamente elevato. Questo porta a oscillazioni violente nella curva.

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43

Il lato opposto della medaglia è il caso di underfitting ovvero la scelta di un modello troppo semplice e poco flessibile che porta a un grande errore anche sui dati di training.

## Statistical Learning Theory

La SLT è la teoria formale che ci permette di capire come le capacità di generalizzazione di un modello cambino al variare della complessità e del numero dei dati.

Finora quello che abbiamo fatto per massimizzare le capacità di generalizzazione è:

- ▶ Cercare di approssimare una funzione  $f$  ignota con un'ipotesi  $h$  minimizzando il rischio  $R = \int L(d, h(\mathbf{x})dP(\mathbf{x}, d))$  dove  $d$  è il label,  $L$  la Loss function e  $P(\mathbf{x}, d)$  la distribuzione di probabilità del dato  $\mathbf{x}$  (sconosciuta altrimenti problema banale)
- ▶ Problema: il nostro set di dati è finito quindi possiamo calcolare solo l'errore sul training set anche detto **Rischio Empirico**
- ▶ Per trovare  $h$  minimizziamo il rischio empirico  $R_{emp} = \frac{1}{n} \sum_{p=1}^n (d_p - h(\mathbf{x}_p))^2$

La domanda che ci poniamo è: E' davvero possibile usare il rischio empirico come approssimazione del rischio reale?

La risposta è sì, c'è un teorema che lo dimostra.

### Teorema 1.16. Vapnik-Chervonenkis inequality (VC inequality)

Sia  $VC$  una misura della complessità dell'ipotesi  $H$  e  $l$  il numero dei dati su cui è calcolato il rischio empirico allora, con una probabilità di  $1 - \delta$  vale la diseguaglianza

$$R \leq R_{emp} + \epsilon(l, VC, \delta)$$

dove  $\epsilon$  è una funzione (detta **VC confidence**) crescente con la **VC dimension** e decrescente con il numero di dati  $l$  e la costante  $\delta$ .

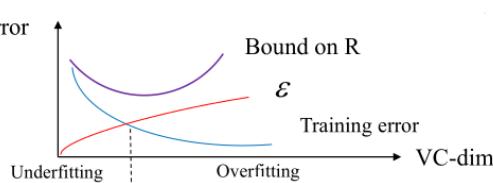
Ci sono diverse formulazioni (per  $\epsilon$ ) a seconda del tipo di funzione  $f$  che si cerca.

**Osservazione 1.17.** Una possibile **VC dimension** può essere come il numero di parametri del modello

Questo bound dimostra come sia necessario un compromesso tra la riduzione del rischio empirico (che porta ad underfitting) e la complessità del modello (che porta ad overfitting), infatti:

**Alto l:** VC confidence

minore  $\rightarrow R_{eff}$  più vicino a  $R$



**Bassa VC:** Un modello

troppo semplice porta a un alto  $R_{emp}$  quindi a un alto rischio reale (**underfitting**)

**Alta VC:** Un modello troppo complesso porta a un basso  $R_{emp}$  ma porta la VC confidence a crescere e con esso cresce il rischio reale (**Overfitting**)

Il compromesso migliore è rappresentato dal minimo tra la somma del rischio empirico e VC confidence (ovvero il rischio reale).

Questa teoria mostra in modo formale che il Machine Learning ha senso ed è ben definito in quanto il rischio reale (a noi inaccessibile) può essere approssimato da quello empirico.

## Validazione

Quello della SLT è un assetto formale ma avere accesso alla VC confidence potrebbe essere in molti casi altamente non banale quindi come possiamo stabilire il miglior compromesso tra complessità e fitting?

La risposta è tramite la **Validazione**.

La validazione ha due obiettivi:

- ▶ **Selezionare il modello:** stimare l'errore di diverse ipotesi e scegliere la migliore
- ▶ **Valutazione del modello:** dopo aver scelto l'ipotesi migliore valutare l'errore su nuovi dati

**Achtung!! 1.18.** *NON usare mai gli stessi dati per i due scopi. Usa un dataset per la scelta del modello e un altro per la sua valutazione*

Ci sono diversi metodi di validazione

- ▶ Hold out cross validation: E' l'approccio standard, un dataset viene diviso in 3 parti: una per il training, una per la validazione e una per il test (es. 60%/20%/20%)
- ▶ K-fold cross validation: I dati di training e validazione vengono divisi in k sottogruppi di cui, a turno, uno è per la validazione, gli altri per il training. Alla fine si fa la media dei risultati di validazione dei vari modelli su tutti i diversi dataset di validazione e si sceglie il migliore. ATTENZIONE: ogni volta che si cambia dataset bisogna riniziare il training da zero (quindi alto effort computazionale)

**Achtung!! 1.19.** *Una volta arrivati alla fase di test non si torna più indietro. Se tornassimo indietro per massimizzare l'accuratezza sul dataset di test sarebbe come se stessimo usando il dataset di test per la validazione. Questo porta inesorabilmente a peggiorare le capacità di generalizzazione.*

## Accuratezza nella classificazione

Nel caso di classificazione si utilizzano le **Matrici di confusione** ovvero tabelle che riportano la frazione di quanti dati sono stati classificati correttamente e quanti no (e in quali classi sono stati erroneamente classificati). Inoltre si definiscono **Specificità** (o True negative rate) e **Sensitività** (o True positive rate)

Predicted \ Actual	Positive	Negative
Positive	TP	FN
Negative	FP	TN

false positive (FP) :eqv. with  
false alarm

$$\text{Specificity} = \text{TN} / (\text{FP} + \text{TN})$$

(True Negative rate =  $1 - FPR$ )

$$\text{Sensitivity} = \text{TP} / (\text{TP} + \text{FN})$$

(True Positive rate or Recall)

$$(Precision = \text{TP}/(\text{TP}+\text{FP}))$$

Figura 1.4: Confusion matrix

**Osservazione 1.20.** Il peggior valore possibile di accuratezza è il 50% in quanto significa che la classificazione avviene in modo del tutto casuale. Una accuratezza minore del 50% vuol dire semplicemente che l'algoritmo sta facendo la scelta opposta.

Un altro modo per valutare un classificatore è tramite la **ROC curve** che è composta da diverse curve al variare del TP rate della sensitività in funzione della specificità. La diagonale corrisponde al classificatore casuale

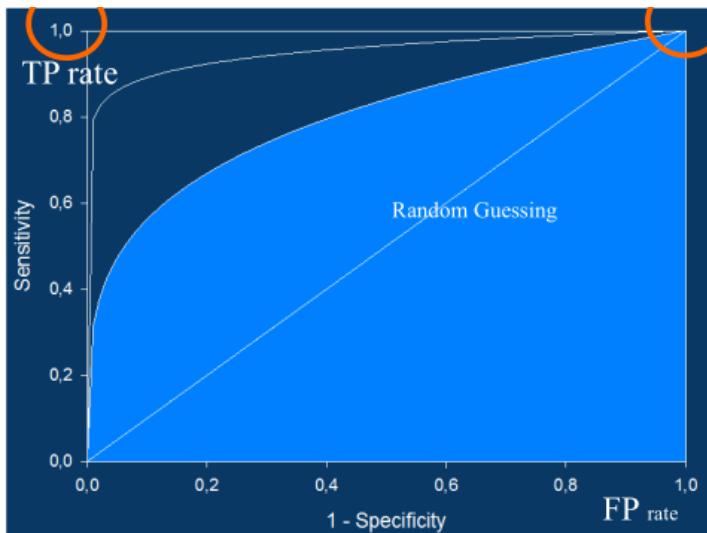


Figura 1.5: ROC curve

## 1.5 Recap

### Design Cycle

Riassumiamo quindi tutto il ciclo di apprendimento:

1. Data collection
2. Rappresentazione, selezione e preprocessing dei dati: selezione delle features, outliers, data augmentation, etc.
3. Scelta del modello: formulazione del problema e delle ipotesi e controllo della complessità (\*)
4. Creazione del modello: fase di apprendimento (\*)
5. Valutazione: Validazione e test del modello (\*)

(\*) Il corso si focalizza su questi aspetti

## Pitfalls

Per qualsiasi modello statistico bisogna fare attenzione ad alcuni aspetti:

- ▶ La causalità tra set di dati è da assumere, sarebbe impossibile creare dei modelli tra variabili totalmente indipendenti o su fenomeni casuali (es. lotteria)
- ▶ La causalità non può essere dedotta solamente dall'analisi dati. Correlazione non implica causalità. Inoltre potrebbe esserci anche causalità al di fuori dei dati a nostra disposizione.
- ▶ Alcuni modelli sono molto flessibili ma bisogna sempre validare bene i risultati in quanto il rischio di fissare variabili totalmente scorrelate potrebbe generare predizioni e interpretazioni del fenomeno totalmente errate

# Modello lineare 2

Vogliamo discutere come primo esempio un modello lineare nelle variabili di input sia con finalità di regressione che di classificazione

**Notazione:** indichiamo con  $x_{p,i}$  la feature i-esima del p-esimo input ( $\mathbf{X}$  matrice  $l \times n$ ). Inoltre indichiamo con  $\mathbf{x}_p$  il vettore delle features del p-esimo input. Il target lo indicheremo con  $y_p$  o d

**Achtung!! 2.1.** *Data A matrice e b vettore nella notazione vettoriale vale  $(Ab)^T = b^T A^T$  ma nella notazione indiciale non cambia nulla, gli oggetti con indici sono scalari, a loro non interessa se sono righe o colonne quindi il tresposto di un vettore non cambia nulla  $(A_{ij}b_j)^T = A_{ij}b_j$ .*

*NON fare l'errore di scambiare gli indici della matrice!!! Scrivere  $b_j A_{ji}$  sarebbe come scrivere  $A^T b$*

*Se vuoi fare derivate direttamente su operatori e vettori trasponi a dovere alcuni termini della somma in modo tale da sommare o tutti vettori riga o tutti vettori colonna. Nota che  $a^T b$  è solo una notazione malata per indicare un **prodotto scalare**. Convieni sempre vederlo in questi termini*

**Achtung!! 2.2.** *Quando  $w_0$  non scritto esplicitamente si intende incluso nel prodotto  $\mathbf{w}^T \mathbf{x}$  ponendo  $x_0 = 1$*

## 2.1 Regressione

L'insieme delle ipotesi ha la forma

$$h(\mathbf{x}_p) = w_0 + \mathbf{w}^T \mathbf{x}_p$$

Unicamente in questa sezione verranno riportati alcuni risultati sia in notazione indiciale che vettoriale per prendere confidenza con la notazione

e sceglieremo come Loss function la somma dei residui quadrati:

$$E = \sum_{p=1}^l (y_p - (w_0 + \mathbf{w}^T \mathbf{x}_p))^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Per trovare la retta di best fit minimizziamo l'errore:  $\frac{\partial E(\mathbf{w})}{\partial w_j} = 0$

$$\partial_{w_j} \left[ \sum_{p=1}^l (y_p - (w_0 + \mathbf{w}^T \mathbf{x}_p))^2 \right] = -2(y_p - (w_0 + \mathbf{w}^T \mathbf{x}_p))(\delta_{j0} + \theta(j)x_{pj}) = 0$$

oppure in notazione vettoriale, derivando rispetto un vettore riga:

$$\partial_{\mathbf{w}^T} [(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w})] = -[\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + ((\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X})^T] = -2\mathbf{X}^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \quad (2.1)$$

il secondo termine è trasposto in quanto altrimenti sarebbe un vettore riga. Ricordiamo inoltre che  $\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T\mathbf{X}$ : uno è un vettore colonna, l'altro riga.

Da qui in poi il  $w_0$  è incluso nel prodotto  $\mathbf{w}^T \mathbf{x}_p$  ponendo  $x_{p0} = 1$

**Osservazione 2.3.** *Nel caso di output multi dimensionali il problema può essere esteso rendendo  $w$  una matrice*

Prima di formulare un algoritmo di apprendimento definiamo il problema di classificazione con un modello lineare. Successivamente vedremo che l'algoritmo di apprendimento sarà lo stesso indipendentemente dal task.

## 2.2 Classificazione (binaria)

Banalmente quello che vogliamo fare è, dati dei punti in un piano appartenenti a due classi diverse, tracciare una retta in modo tale da dividere le due classi massimizzando l'accuratezza.

In questo caso come ipotesi usiamo la funzione segno

$$h(\mathbf{x}_p) = \theta(\mathbf{x}_p^T \mathbf{w})$$

In questo modo l'iperpiano che separa le classi è caratterizzato dalla equazione  $\mathbf{x}^T \mathbf{w} = 0$

**Osservazioni:**

- ▶ In questo caso  $w_0$ , detto anche offset, ha il ruolo di threshold in quanto la funzione segno cambia segno quando  $\mathbf{w}^T \mathbf{x} \geq -w_0$
- ▶ Scaling freedom: moltiplicando i pesi per una costante non cambia niente
- ▶ Il vettore  $\mathbf{w}$  dei pesi è ortogonale alla retta di separazione delle classi, infatti dati due punti  $\mathbf{x}_a$  e  $\mathbf{x}_b$  appartenenti all'iperpiano facendo la differenza dell'equazione dell'iperpiano valutata nei due punti diversi otteniamo  $\mathbf{w}^T(\mathbf{x}_a - \mathbf{x}_b) = 0$
- ▶ In generale la soluzione non è unica, è possibile avere infiniti iperpiani di separazione

### Loss function

Potremmo usare come loss function la **Loss 0/1**

$$\mathcal{L} = \sum_p (y_p - \theta(\mathbf{w}^T \mathbf{x}))^2$$

**PROBLEMA!!:** Questa funzione non è differenziabile, minimizzarla diventa problematico.

Quello che si fa è considerare nuovamente l'**errore quadratico medio**

$$\mathcal{L} = \sum_p (y_p - \mathbf{x}_p^T \mathbf{w})^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

Questa è una funzione quadratica quindi il minimo esiste sempre ma potrebbe non essere unico (se la matrice che definisce la funzione quadratica è definita positiva allora la funzione è convessa e ha un minimo globale. Vedi parte del corso di CM su ottimizzazione convessa (credo))

Questa Loss ha le stesse proprietà della loss 0/1 ma in più è differenziabile

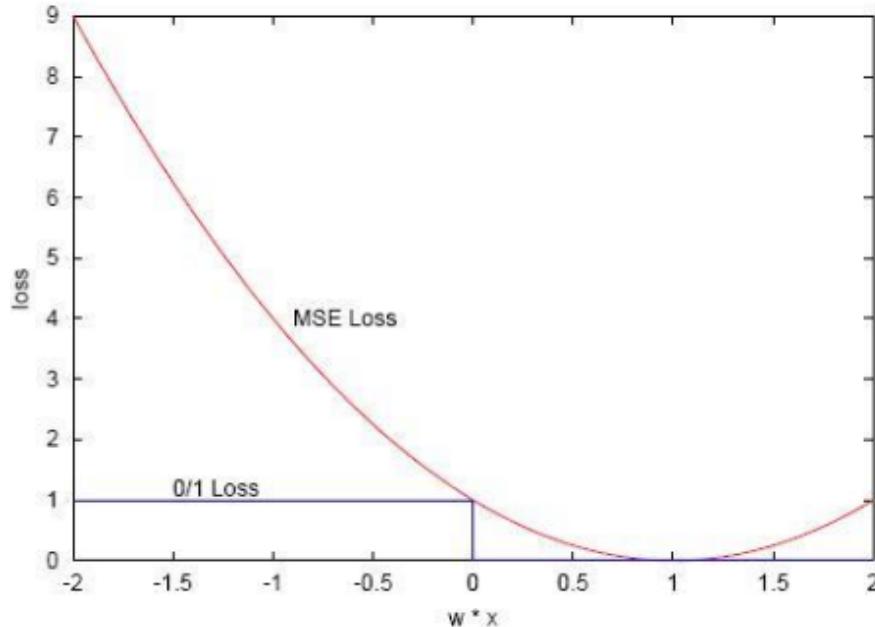


Figura 2.1: Loss 0/1 e MSE

## 2.3 Algoritmo di apprendimento

Data una loss basata sull'errore quadratrico medio proponiamo due tipi di algoritmi di apprendimento basati su due approcci diversi

### Approccio diretto

Possiamo trovare una soluzione analitica esatta all'equazione 2.1 invertendola.

Si ottiene

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{X}^+ \mathbf{y}$$

dove  $\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$  è detto **Pseudoinverso di Moore-Penrose**

### Pseudo inverso di Moore Penrose [ ]

Dato un'operatore lineare  $A \in \mathbb{R}^{m \times n}$  allora è definito come pseudoinverso di Moore Penrose l'operatore  $A^+ \in \mathbb{R}^{n \times m}$  che soddisfa le seguenti proprietà:

- $AA^+A = A$ : ( $AA^+$  non è necessariamente l'identità)
- $A^+AA^+ = A^+$
- $AA^+$  e  $A^+A$  sono hermitiani

Nel caso l'operatore  $A$  sia invertibile allora lo è anche  $A^T A$  e l'operatore pseudoinverso è scrivibile come  $A^+ = (A^T A)^{-1} A^T$  (nel caso complesso sostituire il trasposto con l'aggiunto).

In questo caso il pseudoinverso coincide con l'inverso  $A^+ = A^{-1}$  (per convincersene basta fare  $A^+A$  con la forma di  $A^+$  scritta sopra).

Nel caso  $A$  NON sia invertibile allora non esiste alcun vettore  $x$  t.c  $Ax = b$ .

Il sistema può non avere alcuna soluzione o averne infinite

**Teorema 2.4.** *Per un sistema lineare non determinato  $Ax = b$  la soluzione  $x = A^+b$  è quella che minimizza la quantità  $\|Ax - b\|$ , ovvero è la migliore approssimazione alla soluzione del sistema*

*Dimostrazione.* .

- Scriviamo il sistema come  $Ax - b = Ax - b + AA^+b - AA^+b = A(x - A^+b) - (I - AA^+)b$
- L'operatore ( $P = AA^+$ ) è un proiettore ortogonale in quanto, a partire dalla definizione di pseudo inverso, è banale verificare che  $P^2 = P$  e  $P^* = P$
- $P$  proietta ortogonalmente sull'immagine di  $A$  in quanto, sempre per le definizioni di pseudoinverso,  $PA = A$ . Quindi  $1 - P$  proietta sul complementare ortogonale di  $Im(A)$  [ovvero  $Ker(A^*) = Ker(A^+)$  (questo è da chiarire, non banale ma non utile ai fini della dimostrazione)]
- Quindi essendo il primo e il secondo termine dell'equazione sono vettori ortogonali.  

$$\|Ax - b\|^2 = |A(x - A^+b)|^2 + |(I - AA^+)(-b)|^2$$
- Posto  $x_0 = A^+b$  si ha  $|A(x - x_0)|^2 + |Ax_0 - b|^2 \geq |Ax_0 - b|^2$
- Quindi  $x_0$  è il vettore che approssima meglio il sistema lineare

□

Resta da capire come poter calcolare l'operatore pseudoinverso se l'operatore non è invertibile. Quello che si fa è usare la SVD (singular value decomposition). (Non è l'unico metodo ma uno dei più usati)

### Singular Value Decomposition [2]

Partiamo dal caso di matrice diagonalizzabile. Una matrice  $A$  può essere decomposta come

$$A_{ij} = \lambda_k v_{ki} w_{kj}$$

dove  $v_k$  sono gli autovettori di  $A$  e  $w_k$  vettori tali che  $w_{kj}v_{mj} = \delta_{mk}$ . In forma operatoriale si può scrivere anche come  $A = \mathbf{v}\Lambda\mathbf{w}^T$  dove  $\Lambda$  è la matrice  $A$  diagonalizzata.

Ci possiamo convincere della validità di questa uguaglianza applicando l'operatore a un autovettore di  $A$ , infatti

$$\lambda_k v_{ki} w_{kj} v_{mj} = \lambda_k v_{ki} \delta_{km}$$

**Teorema 2.5.** *Data  $A$  matrice quadrata, la matrice  $A^T A$  è diagonalizzabile con autovettori ortonormali*

*Dimostrazione.* La matrice  $A^T A$  è simmetrica:  $(A^T A)^T = A^T (A^T)^T = A^T A$  e quindi per il teorema spettrale è diagonalizzabile con autovettori ortonormali.  $\square$

**Teorema 2.6.** *Dati  $v_i$  l'insieme di autovettori ortonormali di  $A^T A$  allora gli autovalori  $\lambda_i$  di  $A^T A$  sono tutti positivi e vale  $\|Av_i\|^2 = \lambda_i$*

*Dimostrazione.*  $|Av_i|^2 = (Av_i)^T Av_i = v_i^T A^T Av_i = \lambda_i v_i^T v_i = \lambda_i \geq 0$   $\square$

**Def. 2.7. Valori singolari  $\sigma_i$ :** Assumiamo un ordinamento degli autovalori tale che  $\lambda_n \geq \lambda_{n+1}$  e definiamo i **valori singolari** come  $\sigma_i = \sqrt{\lambda_i} = |Av_i|$

Sia  $A \in \mathbb{R}^{m \times n}$  t.c  $\text{Rank}(A)=r$  (numero vettori riga o colonna nella matrice linearmente indipendenti) allora vale

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$$

$$\sigma_{r+1} = \sigma_{r+2} = \dots = \sigma_n = 0$$

Sia inoltre  $u_i = \frac{Av_i}{\|Av_i\|} = \frac{Av_i}{\sigma_i}$  con  $1 < i < r$  una base ortonormale di  $\text{Imm}(A)$  (dimensione  $r$ ) e  $\{u_{r+1}, \dots, u_m\}$  l'insieme di vettori (ottenuti tramite ortogonalizzazione di Gram Schmidt) che insieme a  $u_i$   $i \in [1, r]$  forma una base di  $\mathbb{R}^m$

**Teorema 2.8.** Vale  $A = \mathbf{U}\Sigma\mathbf{V}^T = \sigma_l v_{li} u_{lk}$  dove

$\Sigma = \text{diag}\{\sigma_1, \dots, \sigma_r, 0, 0, \dots, 0\}$  con  $n-r$  zeri più eventuali colonne o righe aggiuntive nulle per far tornare le dimensioni della matrice

*Dimostrazione.* ▶ Poichè  $u_i$  è una base di  $\text{Imm}(A)$  possiamo scrivere

$$Ax = \sum_{i=1}^r a_i u_i$$

▶ Poichè gli  $u_i$  sono ortonormali vale  $a_i = (Ax)^T u_i = x^T A^T \frac{Av_i}{\sigma_i}$

▶ Ma  $v_i$  sono gli autovettori di  $A^T A$  con autovalore  $\sigma_i^2$  quindi  $a_i = \sigma_i x^T v_i = \sigma_i v_i^T x$

▶ Quindi si ha  $Ax = \sum_{i=1}^r \sigma_i u_i v_i^T x$  che è esattamente quanto afferma la tesi

$\square$

Quindi, poichè vale  $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$  (dove le dimensioni sono  $\mathbf{X} = m \times n$ ,  $\mathbf{U} = m \times m$ ,  $\Sigma = n \times m$ ,  $\mathbf{V}^T = n \times n$ ) si ha:

$$\mathbf{X}^+ = (\mathbf{U}\Sigma\mathbf{V}^T)^+ = (\mathbf{V}^T)^+\Sigma^+\mathbf{U}^+$$

Poichè le matrici  $\mathbf{V}$  e  $\mathbf{U}$  sono unitarie, e quindi invertibili, vale  $\mathbf{U}^T = \mathbf{U}^+ = \mathbf{U}^{-1}$ , quindi

$$\mathbf{X}^+ = \mathbf{V}\Sigma^+\mathbf{U}^T$$

dove  $\Sigma^+ = \text{diag}\{1/\sigma_1, 1/\sigma_2, \dots, 1/\sigma_r, 0, 0, \dots, 0\}$  con  $n-r$  zeri. E' banale verificare che  $\Sigma^+$  soddisfa tutte le proprietà di operatore pseudoinverso.

**Achtung!! 2.9.** *Qui ho usato senza troppi problemi il trasposto, in realtà per tenere conto di matrici complesse andrebbe usato l'aggiunto. Quindi intendere tutti i trasposti delle matrici come aggiunti*

**PROBLEMA:** Risolvere il problema dei minimi quadrati con questo algoritmo ha un costo computazionale  $O(n^3)$  dove  $n$  è la dimensione della matrice. Catastrofico

## Gradient Descent: approccio iterativo

L'errore  $E = \sum_{p=1}^l (y_p - (\mathbf{w}^T \mathbf{x}_p))^2$  rappresenta una ipersuperficie.

Dato un punto  $\mathbf{w}_0$  appartenente alla ipersuperficie il vettore che punta nella direzione di massima discesa è  $-\nabla(\mathbf{w}_0)$ . Se l'ipersuperficie è convessa, come questo caso, tutti i minimi locali sono anche globali.

Possiamo proporre quindi un algoritmo per trovare il minimo di questa ipersuperficie:

1. Si prenda un vettore  $\mathbf{w}_0$  casuale (con valori bassi)
2. Si calcoli  $\Delta\mathbf{w} = -\nabla E(\mathbf{w})$
3. Si calcoli  $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta\Delta\mathbf{w}$  dove  $\eta \in [0, 1]$  è detto **Rate di apprendimento**
4. Ripeti 2 e 3 finchè  $\mathbf{w}_{n+1} - \mathbf{w}_n$  non è sufficientemente piccolo

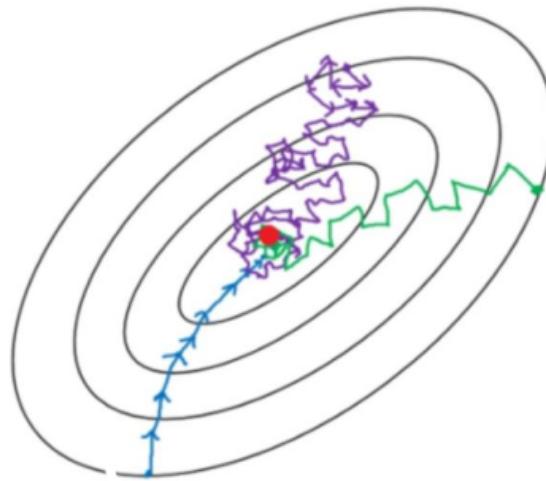
**Achtung!! 2.10.** *Stabilire un valore del learning rate significa scegliere il giusto compromesso tra la velocità dell'algoritmo e la sua stabilità. Un valore troppo alto rischia di far oscillare i valori furiosamente intorno al minimo, un valore troppo aumenta inutilmente il numero di iterazioni necessarie per ottenere la convergenza*

L'idea intuitiva dietro il gradient descent è: dato  $\Delta w_j = 2 \sum_{p=0}^l x_{pj}(y_p - x_p^T w)$  si ha che:

- se  $y_p - x_p^T w \leq 0$  allora significa che l'output predetto  $x^T w$  è troppo alto. Questo porta a  $\Delta w < 0$  che porta a diminuire l'output
- se  $y_p - x_p^T w \geq 0$  allora significa che l'output predetto  $x^T w$  è troppo basso. Questo porta a  $\Delta w > 0$  che porta ad aumentare l'output

Le 2 principali versioni di gradient descent sono:

- ▶ **Batch version:**  $\partial_{w_j} E(\mathbf{w}) = -2 \sum_{p=0}^l (y_p - (\mathbf{w}^T \mathbf{x}_p)) x_{pj}$   
Ovvero il gradiente è calcolato su tutti i dati sommati e i pesi vengono aggiornati dopo la somma. Questo approccio media sul rumore dei dati.
- ▶ **Online/stochastic versio:** I pesi vengono aggiornati calcolando il gradiente per un input alla volta. Può essere più veloce ma per evitare che dipenda troppo dai singoli dati necessita un  $\eta$  molto basso
- ▶ Ci sono approcci intermedi come il mini-batch



Paths over the error surface by Batch or On-line version

Figura 2.2: Stochastic vs Batch Gradient Descent

Come prima cosa quando si usa un algoritmo di gradient descent bisogna controllare la **Stabilità** e poi la velocità della curva di apprendimento, ovvero della curva Errore vs #iterazioni.

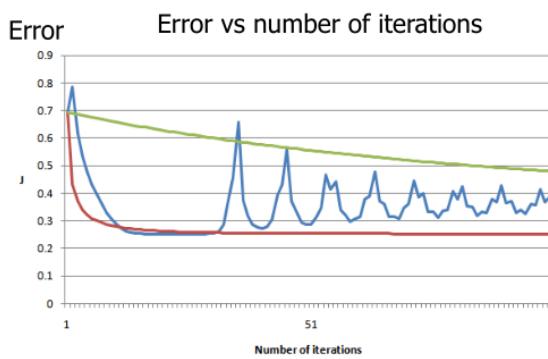


Figura 2.3: Curva di apprendimento  
In questo grafico possiamo notare in blu una curva instabile, in verde una curva molto lenta e in rosso il tipo di curva di apprendimento che desideriamo trovare

### Considerazioni finali

Il gradient descent è un esempio lampante di come un algoritmo possa imparare dai propri errori.

Questo approccio ci permette di cercare una soluzione in uno spazio delle ipotesi infinito e può essere applicato a qualsiasi spazio delle ipotesi continuo e con loss differenziabile.

## 2.4 Considerazioni

- ▶ Nel caso in cui i dati di ogni classe siano generati da 2 gaussiane con componenti scorrelati il modello lineare è quasi ottimale ma la regione di sovrapposizione è inevitabile.
- ▶ Questo modello ha dei fortissimi bias. Il language bias è che l'ipotesi sia lineare negli input, il search bias è che la ricerca sia guidata dalla ricerca del minimo dei minimi quadrati

Nel caso di errori gaussiani fare un fit ai minimi quadrati equivale totalmente a farlo massimizzando la likelihood

Il modello presenta alcune limitazioni, ad esempio non sempre un numero di 3/4 punti può essere separato da una retta. Inoltre può essere complicato estenderlo a task di classificazione a più di 2 classi. I 2 approcci principali sono:

- ▶ one vs all. Vengono creati K classificatori che discriminano in modo binario tutte le classi con un'unica scelta
- ▶ all vs all. Si creano tutte le possibili  $K(K-1)$  coppie di classificatori binari e il vincitore p quello con la somma di output maggiore rispetto a tutte le altre classi.

Però esistono alcuni modelli che sono multi-output per loro natura (es. linear discriminant analysis).

## 2.5 Generalizzazione

Quello che ci interessa è che il problema sia **lineare nei pesi  $w$** , non negli input, quindi come ipotesi possiamo considerare qualsiasi funzione scritta in **espansione in base lineare**

$$h_w(\mathbf{x}) = \sum_{k=0}^K w_k \varphi_k(\mathbf{x})$$

dove  $\varphi_k : \mathbb{R}^N \rightarrow \mathbb{R}$

**Osservazione 2.11.** Per  $K \rightarrow \infty$  e data  $\varphi_k$  base completa questa non è altro che una espansione di Fourier. Per evitare problemi di overfitting, ovviamente, la serie è troncata

L'insieme delle  $\varphi$  va scelto accuratamente (Questo è chiamato "dictionary approach")

- ▶ Pro: Le  $\varphi_k$  possono essere complicate a piacere, questo da molta flessibilità

- Contro: Con una base troppo ampia si rischia overfitting, la complessità va tenuta sotto controllo.  
Inoltre, a differenza dell'approccio adattivo delle NN, le funzioni  $\varphi_k$  vanno fissate a priori

## 2.6 Regolarizzazione: controllo della complessità

La complessità può essere tenuta sotto controllo modificando la loss function

$$\text{Loss}(\mathbf{w}) = \sum_p (y_p - \mathbf{x}_p \cdot \mathbf{w})^2 + \lambda \|\mathbf{w}\|^2$$

Questa tecnica si chiama **regolarizzazione di Tikhonov** oppure **ridge regression** e  $\lambda > 0$  è detto parametro di regolarizzazione (di solito molto piccolo).

**Achtung!! 2.12.** *Qui abbiamo ridefinito la Loss poiché usata come cost function durante la fase di training ma l'errore E che si usa per valutare l'accuratezza del modello è un'altra cosa (è la solita che abbiamo scritto finora)*

Minimizzare questa Loss corrisponde a trovare il giusto tradeoff tra il minimizzare il MSE e minimizzare la norma dei pesi (è tipo un moltiplicatore di lagrange).

**Osservazione 2.13.** *Per il no free lunch theorem non esistono metodi di regolarizzazione (ovvero di modifiche all'algoritmo di apprendimento col fine di migliorare le capacità di generalizzazioni) migliori di altri*

Questa procedura di regolarizzazione è molto utile in entrambi gli approcci di apprendimento.

### Approccio diretto

Partendo sempre da 2.1 è possibile invertire la relazione ma ora il termine di regolarizzazione aggiuntivo modifica la soluzione

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbb{1})^{-1} \mathbf{X}^T \mathbf{y}$$

La matrice  $(\mathbf{X}^T \mathbf{X} + \lambda \mathbb{1})$  ora però è sempre invertibile.

Non mi impelaggerò in questa dimostrazione (non fatta a lezione) ma proverò a dare una spiegazione intuitiva:

Abbiamo detto che  $\mathbf{X}^T \mathbf{X}$  è simmetrica quindi diagonalizzabile quindi restrin-giamoci al caso di matrice diagonale.

I valori presenti sulle diagonali non sono altro che le norme quadre dei vettori che compongono le colonne della matrice  $\mathbf{X}$  a valori reali quindi gli autovalori saranno  $\geq 0$ .

La matrice è non invertibile solo nel caso un autovalore sia 0 ma sommarci sopra

$\lambda \mathbb{1}$  forza la matrice ad avere tutti elementi diagonali  $> 0$  e quindi  $(X^T X + \lambda \mathbb{1})$  è sempre invertibile.

Ovviamente se la matrice  $X^T X$  non è diagonale la matrice ortogonale di cambio base mischierà la diagonale aggiuntiva dei  $\lambda$  e non è banale mostrare che il discorso valga lo stesso però questa non è una dimostrazione rigorosa ma solo una considerazione sul perché mai la regolarizzazione possa implicare una matrice sempre invertibile.

La dimostrazione seria è una roba impestata fatta sugli spazi di hilbert tramite una matrice di funzione di green. [5] (pag. 322)

## Gradient Descent

Ricordiamo che il gradient descent consiste nel sommare ad ogni passo al vettore dei per una quantità  $\eta \Delta \mathbf{w} = -\eta \nabla Loss(\mathbf{w})$ .

Quindi in questo caso, definito il  $\Delta \mathbf{w}_n = \sum_{p=0}^l \mathbf{x}_p (y_p - \mathbf{x}_p^T \mathbf{w}_n)$  come prima, la nuova regola del gradient descent, facendo il gradiente della Loss cambiato di segno, è

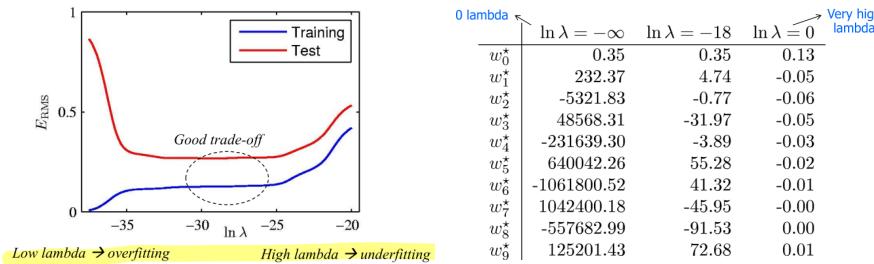
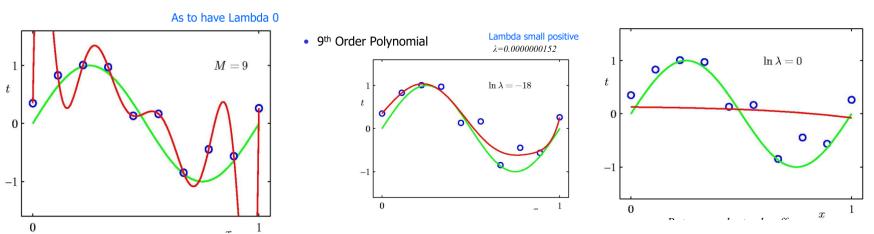
$$\mathbf{w}_{n+1} = \mathbf{w}_n + \eta \Delta \mathbf{w}_n - 2\lambda \mathbf{w}$$

(in realtà qui il parametro di apprendimento agisce solo sul primo termine della derivata della loss ma la cosa non cambia, è solo più comodo)

**Osservazione 2.14.**  $w_{n+1} - w_n = -2\lambda w + \dots$  implica che ogni step di aggiornamento dei pesi è soppresso da un fattore esponenziale  $e^{-2\lambda}$

Il termine di regolarizzazione tende a penalizzare tutti i valori alti di  $w$  e porta a preferire i valori bassi (nel limite  $\lim_{\lambda \rightarrow \infty} \|w\| \rightarrow 0$ ).

In questo modo si implementa un nuovo metodo per controllare la complessità del modello tramite il paramentro  $\lambda$  che tende a ridurre la VC dimension



In questo modo è possibile gestire l'overfitting tramite il tuning di un parametro.

Eistono altri tipi di regolarizzazione come ad esempio il Lasso che è la regolarizzazione rispetto alla norma L1 che però non è differenziabile. Questa tende a portare alcuni pesi direttamente a zero (fa una sorta di feature selection automatica) ma non essendo differenziabile ha bisogno di approcci diversi dal gradient descent.

Un altro approccio è l'elastic net che sfrutta una mistura di norma L2 e L1.

**Ricapitolando:**

- ▶ Possiamo aumentare la flessibilità tramite la linear basis expansion
- ▶ Possiamo controllare la complessità tramite la regolarizzazione

In ogni caso il modello lineare soffre di un language bias molto forte quindi risulta essere un modello molto rigido. Ora studieremo , come modello in contrapposizione al modello lineare, un modello molto flessibile: il KNN

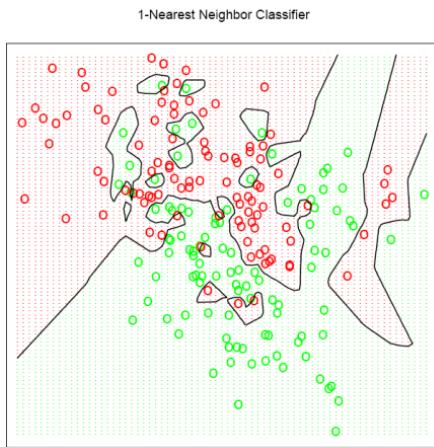
# K Nearest Neighbor 3

Il principio dietro questo modello è: se un gruppo di punti appartiene alla stessa classe allora i punti vicini a loro apparterranno alla stessa classe.

Questo algoritmo immagazina i dati di training e formula una ipotesi ad hoc per un punto di test solo quando viene presentato e non a priori. Questo approccio di apprendimento viene chiamato **Lazy**. (riguarda il timing dell'algoritmo di apprendimento)

## 3.1 1-nn

Consideriamo il caso più semplice ovvero **1-nn**:



**Figura 3.1:** 1-nn: Il dato di test viene classificato come il punto a lui più vicino

In questo caso ogni dato di test viene posto nella stessa classe del punto di training a lui più vicino.

- ▶ Modello molto flessibile
- ▶ Sui dati di training l'errore è 0
- ▶ I bordi sono fortemente irregolari (e non lineari)
- ▶ E' presente rumore non necessario (in altre parole c'è un overfitting pauroso)

## 3.2 k-nn

Generalizziamo questo modello al **k-nn**:

Dato un punto di test consideriamo i  $k$  punti di training più vicini e attribuiamo

al punto di test la stessa classe a cui appartiene la maggioranza dei  $k$  punti di training selezionati. Questo ci permette di fare una sorta di smoothing sul rumore

**Osservazione 3.1.** Ovviamente conviene che sia  $k$  dispari in quanto con  $k$  pari ci si potrebbe trovare nella situazione di non avere una maggioranza e quindi di non poter classificare il dato

Per formalizzare questa idea definiamo la media come:

$$\text{avg}_k(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$$

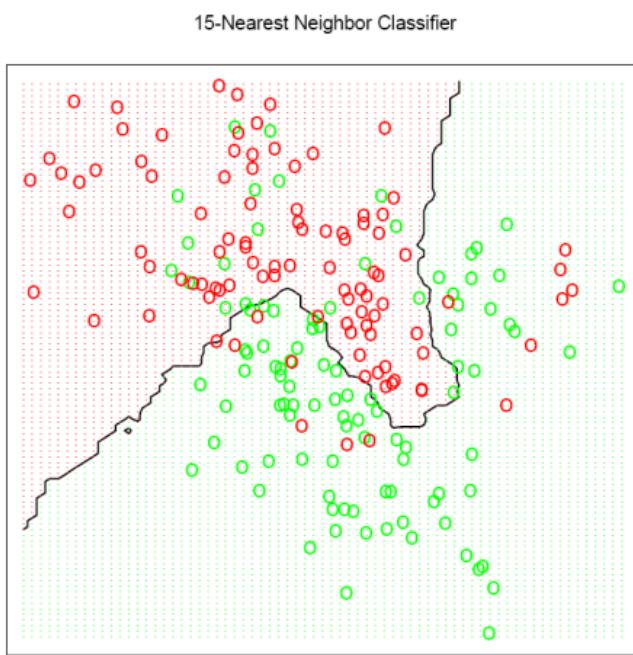
dove  $N_k(\mathbf{x})$  è l'insieme dei  $k$  punti di training più vicini al punto  $\mathbf{x}$

E formalizziamo le ipotesi come:

- **Regressione:**  $h(\mathbf{x}) = \text{avg}_k(\mathbf{x})$
- **Classificazione:**  $h(\mathbf{x}) = \theta(\text{avg}_k(\mathbf{x}) - 0.5)$  con  $y_i \in \{0, 1\}$

**Achtung!! 3.2.** Qui si sta sempre intendendo il concetto di "più vicino" secondo la norma euclidea  $\sqrt{\sum_t (x_t - x_{pt})^2} = \|\mathbf{x} - \mathbf{x}_p\|$  MA posso usare la norma che preferisco (Es. norma  $L_1, L_\infty, \dots$ )

Aumentando il  $k$  a 15 si puo osservare come già ad occhio sia ridotto l'overfitting



**Figura 3.2:** k-nn con  $k=15$ . Confrontando questo plot con il caso  $k=1$  possiamo subito notare come il parametro  $k$  ci permetta di controllare la complessità del modello

Un altro effetto che si ottiene aumentando il  $k$  è che crescono il numero di misclassificazioni nel training set. Questo è abbastanza naturale sia perchè è

esattamente quello che vogliamo per evitare che il modello asseconti troppo le fluttuazioni dei dati, sia perché il modello non è costruito tramite un algoritmo di apprendimento che minimizzi l'errore sul training set

## Multiclass

Il modello è generalizzabile a **più classi** definendo l'ipotesi

$$h(\mathbf{x}) = \operatorname{argmax}_v \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} \delta_{vy_i}$$

ovvero contando le classi nel vicinato per ogni tipo di classe diversa (qui indicizzata con  $v$ ) e prendendo quella di maggioranza (argmax seleziona la classe con più presenze nel k-vicinato).

## K-nn pesato

E' possibile definire anche una variante del modello in cui non si considerano più in modo equivalente tutti i punti del k-vicinato ma ogni punto in questione viene pesato, ad esempio, con il quadrato inverso della distanza con il punto  $\mathbf{x}$ , ovvero:

$$h(\mathbf{x}) = \operatorname{argmax}_v \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} \frac{\delta_{vy_i}}{d(\mathbf{x}, \mathbf{x}_i)^2}$$

Ovviamente il peso si può ridefinire abbastanza liberamente

## Generalization capabilities

Il numero di gradi di libertà per il k-nn è  $ndof=1/k$  dove  $l$  è il numero dei dati. Per  $ndof$  troppo alto si ha, come nel modello lineare, overfitting

La linea viola corrisponde al Bayesian error rate (Bayes classifier).

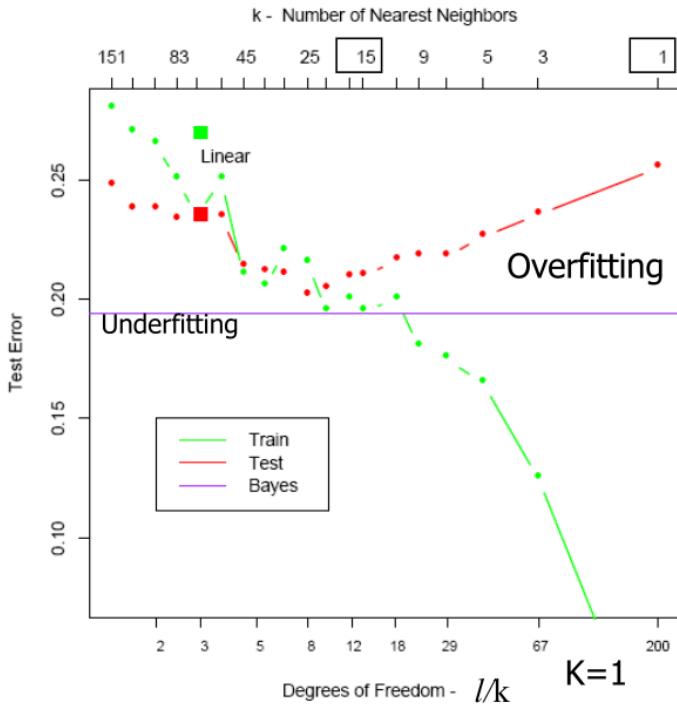
I dati in figura 3.2 sono stati generati tramite 2 distribuzioni gaussiane quindi sappiamo la densità di probabilità  $P(\mathbf{x}, y)$ .

Possiamo quindi classificare ogni punto tramite il teorema di bayes: la classe più probabile del dato  $\mathbf{x}$  sarà  $\max P(v|\mathbf{x})$  con  $v$  indice della classe.

Il rate di errore del classificatore ottimale di bayes è chiamato Bayes rate ed è il minimo error rate che è possibile ottenere sapendo a priori la distribuzione di probabilità dei dati

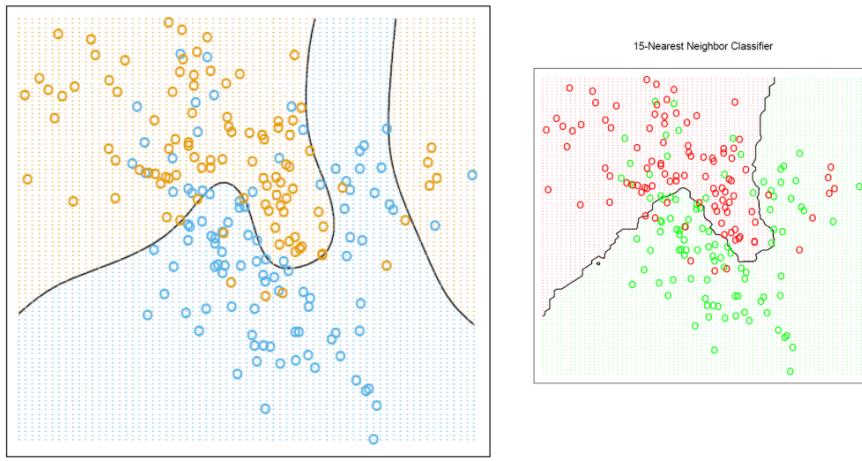
K-nn approssima la soluzione dell'optimal bayes classifier ad eccezione del fatto che:

- ▶ La probabilità condizionata in un punto viene sostituita con la probabilità condizionata entro l'intorno del punto stesso
- ▶ Le probabilità sono stimate prendendo campioni del training set



**Figura 3.3:** Questo plot mostra come si possa trovare il risultato migliore controllando la complessità tramite il parametro  $k$

Ad esempio possiamo vedere come, per i dati generati, 15-nn approssimi abbastanza bene l'optimal bayes classifier



**Figura 3.4:** Confronto tra l'optimal bayes classifier e k-nn con  $k=15$  su dati generati a partire da 2 distribuzioni gaussiane

## Inductive bias

Il bias nel knn è dato dall'assunzione che la somiglianza tra dati appartenenti alla stessa classe sia caratterizzata dalla distanza tra i punti.

Inoltre è un forte bias anche la scelta della metrica. In questo caso abbiamo considerato la metrica euclidea ma i risultati possono cambiare sensibilmente

usando metriche alternative come  $L_1$ ,  $L_\infty$ , etc.

Questo implica che il modello è invarianto per trasformazioni di scala globali mentre è altamente fragili per trasformazioni di scala di singole variabili. Quello che di solito si fa è effettuare trasformazioni di scala su ogni variabile in modo tale da portarle ad essere a media nulla e varianza unitaria.

Fare una trasformazione di scala di questo tipo equivale a cambiare la metrica.

### 3.3 K-nn vs Linear model

Il k-nn e il modello lineare rappresentano due modelli dai principi opposti.

**Il modello lineare:**

- ▶ modello rigido (cambiare di poco la posizione di un punto nel training set non cambia di molto l'ipotesi trovata)
- ▶ Timing dell'algoritmo di apprendimento **eager**: Analizza i dati di training e formula una ipotesi
- ▶ Approssimazione globale della target function

Invece **k-nn**:

- ▶ Modello estremamente flessibile (cambiare un punto nel training può cambiare molto)
- ▶ Timing dell'algoritmo di apprendimento **lazy**: immagazina dati di training e formula un'ipotesi solo al presentarsi di un dato di test
- ▶ Approssimazione locale: l'ipotesi formulata nell'intorno di un punto dipende solamente dai punti appartenenti a quell'intorno (distance-based)

Vedremo come, però, un modello così flessibile sia tremendamente pericoloso

### 3.4 Limiti del knn

knn è un modello molto flessibile ma presenta delle problematiche rilevanti:

- ▶ **Metrica e trasformazioni di scala**: Il modello dipende fortemente dalla metrica e dalle trasformazioni di scala di singole variabili (come già discusso)
- ▶ **Costo computazionale**: Tutto il costo computazionale è delegato alla fase di predizione ed è proporzionale al numero di dati nel training set e a  $k$ . Questo può essere un problema per alcune applicazioni dove è necessaria una risposta del modello abbastanza rapida
- ▶ **Inefficace per dimensioni alte**

In caso si abbia un alto numero di feature si presentano due problematiche.

1. Le feature non rilevanti ai fini della classificazione portano solo rumore e può seriamente compromettere i risultati. Più la dimensionalità è alta più il problema è grave.

E' necessario fare una feature selection molto restrittiva per eliminare tutte le feature non rilevanti

2. In alta dimensione è difficile trovare punti vicini e i k-intorni diventano spazialmente molto grandi e questo rende impossibile applicare il principio di similitudine dei dati in funzione della distanza.

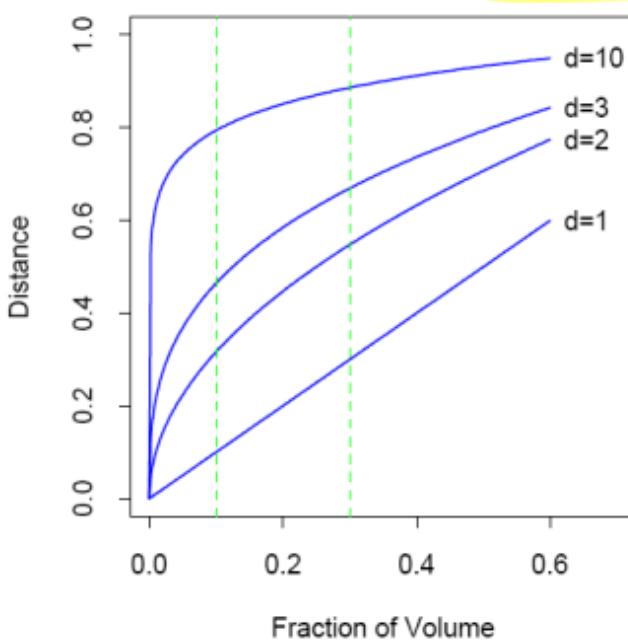
Un modo per diminuire il volume dei k-intorni è diminuire k ma questo porta inesorabilmente all'overfitting (ricorda che  $\text{ndof} = l/k$ ).

Cerchiamo di capire perché la distanza tra punti vicini tende ad aumentare in alta dimensione:

Consideriamo i dati distribuiti uniformemente in un cubo unitario in d dimensioni. Un sottovolume che contiene k punti ha lato di lunghezza  $s = (k/l)^{1/d}$  dove  $k/l$  è la dimensione del sottovolume.

Questo significa che se abbiamo 1000 dati e  $k=20$  in dimensione 2 la larghezza del k intorno è il 14% dell'area del cubo mentre in dimensione 10 è il 68% della lunghezza del lato del cubo.

In 10D abbiamo bisogno di coprire l'80% del range di ogni coordinata per contenere il 10% dei dati in quanto  $0.1^{1/10} = 0.8$ .



**Figura 3.5:** Lunghezza del lato del sottovolume in funzione della frazione di volume al variare della dimensionalità

Inoltre se per approssimare una funzione in  $\mathbb{R}$  sono necessari m punti, in  $\mathbb{R}^d$  ne servono  $m^d$

L'unico modo di arginare questi problemi (senza poterli risolvere del tutto) è fare una feature selection molto severa per ridurre la dimensionalità e il rumore sui dati e cercare la migliore metrica possibile tentando anche di inserire vari pesi.

Spesso è necessario selezionare sottoinsiemi di dati o di feature tramite tecniche di clustering

# Neural networks 4

## 4.1 Perceptron

Il perceptron è la forma più semplice di rete neurale e corrisponde al singolo neurone (detto anche unità).

**Def.** 4.1. Dati  $\{x_1, \dots, x_n\}$  input definiamo

$$net_i(\mathbf{x}) = \sum_{j=0} w_{ij} x_j$$

$$o_i = f(net_i(\mathbf{x}))$$

dove  $w_{ij}$  con  $j = \{1, \dots, n\}$  è il vettore dei pesi per il neurone  $i$ ,  $o_i$  è l'output del neurone  $i$ -esimo e  $f$  è la funzione di attivazione caratteristica del neurone

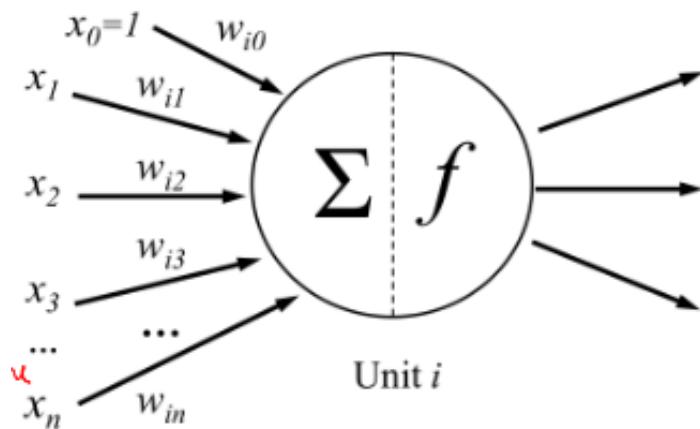


Figura 4.1: Schematizzazione del perceptron

**Osservazione 4.2.** La somma nel net parte da 0 quindi è presente anche il bias

### Linear threshold unit (McCulloch & Pitt)

Questo particolare modello si basa sui seguenti principi:

- I neuroni possono essere solo in due stati: 0 (spento) e 1 (acceso)
- Tutte le connessioni sono equivalenti e caratterizzate da un peso  $w$  che è positivo per le connessioni "eccitatorie", negativo per quelle "inebitorie"
- Il neurone  $i$ -esimo si attiva quando  $net_i(\mathbf{x}) > 0$
- Sia l'input che l'output è lineare

Quindi per questo modello si ha che la funzione di attivazione è

$$f(net_i(\mathbf{x})) = \theta(net_i(\mathbf{x})) = \theta(w_{ij} x_j)$$

### ESEMPIO: Funzioni booleane

Consideriamo un input con 2 feature binarie e label dato da una funzione booleana applicata alle feature.

Vogliamo usare l' LTU per creare un classificatore Caso AND e OR Il modello non è nient'altro che un classificatore lineare

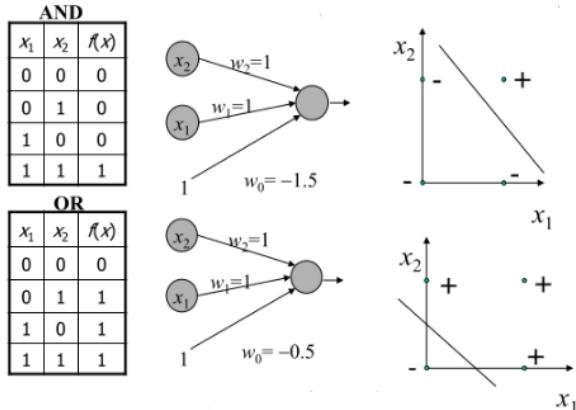


Figura 4.2: LTU applicato ai casi AND e OR

Si osservi come l'unica differenza nei due casi è l'offset Caso NOT Caso banale con una singola feature con  $w_0 = -0.5$  e  $w_1 = 1$  Caso XOR Nel caso la funzione booleana sia uno XOR si presenta un problema. C'è però un modo di risolvere

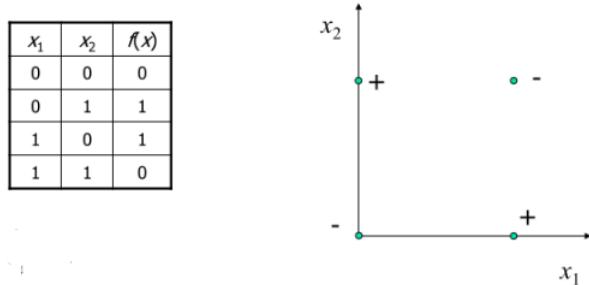


Figura 4.3: Si nota subito ad occhio che lo XOR non è linearmente separabile

il problema.

Consideriamo l'isomorfismo dell'algebra di Boole (per usare le usuali operazioni ed evitare le maledettissime regole di de morgan) con  $\mathbb{Z}_2$  dove :

- True=1, False=0
- A and B = AB
- A or B = A + B + AB
- A xor B = A + B
- not A = A+1

In questo modo cerchiamo di riscrivere lo xor tramite l'and, il not e l'or che siamo già riusciti a modellizzare tramite l'LTU perceptron

Poichè in  $\mathbb{Z}_2$  si ha che  $A^2 = A$  e che i numeri pari sono nulli allora

$$A + B = A + B + 4AB = (A + B + AB)(AB + 1) = (A \text{ or } B) \text{ and } (\text{not } (A \text{ and } B))$$

E' possibile quindi effettuare un cambio di variabili in modo tale che il problema sia linearmente separabile

Definiamo  $h_1 = x_1 \text{ and } x_2$  e  $h_2 = x_1 \text{ or } x_2$

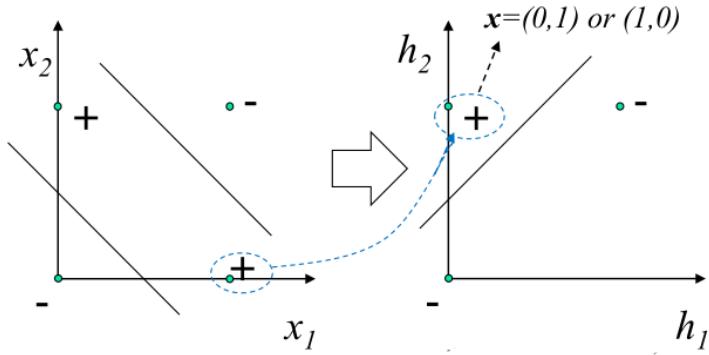


Figura 4.4: XOR nelle nuove variabili  $h_1$  e  $h_2$

Nelle nuove variabili il problema è linearmente separabile quindi possiamo usare il perceptron.

Fare questo cambio di variabili equivale ad aggiungere un layer alla rete

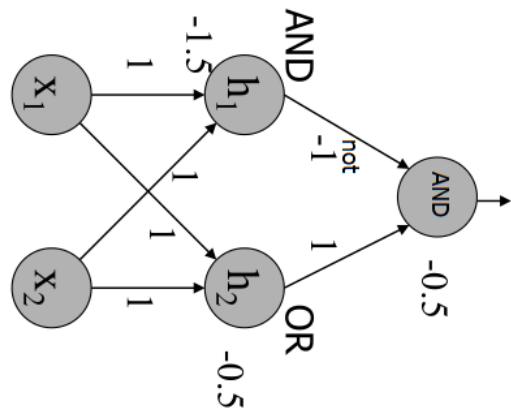


Figura 4.5: Multi layer perceptron per modellizzare la funzione booleana xor

Questo ci mostra come aggiungere un layer corrisponde a fare un cambio di variabili tramite la composizione di funzioni e questo è un fattore chiave nello sviluppo delle reti neurali.

Si può dimostrare che qualsiasi funzione booleana può essere rappresentata da una rete neurale a 2 layer (anche se più layer possono aumentare l'efficienza)

## Algoritmo di apprendimento

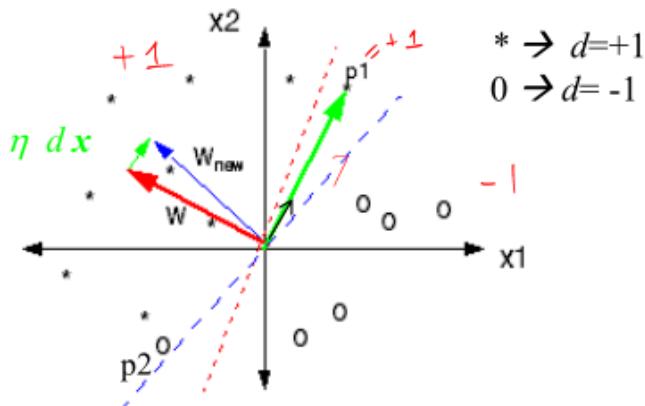
Ci sono due metodi di apprendimento per il perceptron:

- **Adaline** = soluzione del problema dei minimi quadrati. (E' quello che già conosciamo e che generalizzeremo ai multi layer perceptron)
- **Rosenblatt perceptron** = Solo per la Classificazione. Si usa con funzioni a gradino o simili

Avendo già visto l'approccio dell'Adaline qui ci concentreremo sul **Perceptron di Rosenblatt**:

1. Inizializzare i pesi a un valore casuale
2. Scegliere un learning rate (tra 0 e 1)
3.  $\mathbf{w}_{n+1} = \mathbf{w} + \eta(d - out)\mathbf{x}/2$  dove  $d$  è il label e  $out = sign(\mathbf{w}^T \mathbf{x})$  (oss: i pesi cambiano solo in caso di misclassificazione)
4. Ripeti finché non viene soddisfatta una condizione

Dal punto di vista geometrico, se  $p_1$  con  $d_{p1} = 1$  viene misclassificato il vettore dei pesi  $w$  si muove nella stessa direzione di  $p_1$ , se  $d_{p1} = -1$  si muove in direzione opposta. Considerando il fatto che il vettore dei pesi è ortogonale alla retta di separazione delle classi questo ci dà una spiegazione geometrica di quello che sta succedendo



**Figura 4.6:** Interpretazione geometrica dell'algoritmo di apprendimento

I pesi cambiano solo in caso di misclassificazione quindi questa è una forma di apprendimento di error-correction, ovvero il nostro modello apprende dai suoi errori

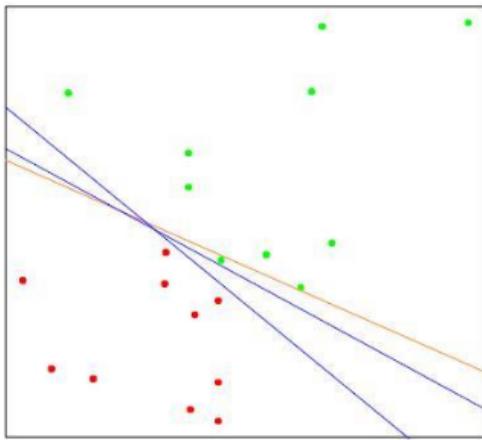
### Differenze tra LMS e Perceptron learning algorithm

L'algoritmo di apprendimento del perceptron e quello dei minimi quadrati sono apparentemente simili ma hanno delle differenze. **LMS**:

- $\delta = (d - \mathbf{w}^T \mathbf{x})$
- Minimizza l'errore con  $out = \mathbf{w}^T \mathbf{x}$
- Converge asintoticamente anche per problemi non linearmente separabili
- Ammette la presenza di errori di classificazione anche in caso di problema linearmente separabile
- Basato sul gradient descent quindi applicabile anche alle reti neurali

#### Perceptron:

- $\delta = (d - sign(\mathbf{w}^T \mathbf{x}))$  solo per i pattern misclassificati
- minimizza le misclassificazioni con  $out = sign(\mathbf{w}^T \mathbf{x})$
- Converge sempre per problemi linearmente separabili al classificatore ottimale
- Per problemi non linearmente separabili non converge
- È difficile da estendere a una rete neurale



**FIGURE 4.14.** A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the perceptron learning algorithm with different random starts.

**Figura 4.7:** In questo grafico è possibile vedere come anche per problemi linearmente separabili il classificatore trovato tramite LMS non è il classificatore ottimale e misclassifica un parametro. In questo caso è possibile notare che la causa di questo si il punto verde più lontano dalla retta che "tira" verso di se la retta di classificazione. Inoltre è possibile notare come il classificatore ottenuto dal perceptron non sia unico, dipende dai valori iniziali ma in ogni caso non misclassifica nessun pattern

## Perceptron Convergence Theorem

**Teorema 4.3.** Se il problema è linearmente separabile, il perceptron converge alla soluzione ottimale in un numero finito di passaggi

**Osservazione 4.4.** Linearmente separabile significa che  $\exists \mathbf{w}^* \text{ t.c. } d_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq \alpha = \min_i d_i(\mathbf{w}^* \cdot \mathbf{x}_i) > 0$  poiché  $\text{out}=\text{sign}(\mathbf{w}^* \cdot \mathbf{x}) \implies \mathbf{w}^* \cdot \mathbf{x}_i \text{ ha lo stesso segno di } d_i$

**Dimostrazione.** ► Definendo  $\mathbf{x}'_i = d_i \mathbf{x}_i$  si ha che  $d_i(\mathbf{w}^* \cdot \mathbf{x}_i) = \mathbf{w}^*(d_i \mathbf{x}_i) = \mathbf{w}^* \mathbf{x}'_i \geq \alpha$ .

In questo modo  $\mathbf{w}^*$  è soluzione per gli esempi  $(\mathbf{x}_i, +1)$  quindi possiamo ridurci al caso  $d_i = +1$  without loss of generality

► Definiamo  $\beta = \max_i \|\mathbf{x}_i\|^2$ , consideriamo  $\eta = 1$  e assumiamo  $\mathbf{w}(0) = 0$

Inoltre identifichiamo con il pedice  $i_j$  unicamente i pattern misclassificati

► Poiché  $\mathbf{w}_n = \mathbf{w}_{n-1} + \mathbf{x}_{i_j}$  (ricorda che  $d = \eta = 1$ ) dopo  $q$  errori si ha  $\mathbf{w}_q = \sum_{j=1}^q \mathbf{x}_{i_j}$

► **Lower bound:**  $\mathbf{w}^* \cdot \mathbf{w}_q = \mathbf{w}^* \sum_j^q \mathbf{x}_{i_j} \geq q\alpha$

Per Cauchy-Schwartz si ha che  $\|\mathbf{w}^*\|^2 \|\mathbf{w}_q\|^2 \geq (\mathbf{w}^* \cdot \mathbf{w}_q)^2 \geq (q\alpha)^2$

$$\implies \|\mathbf{w}_q\|^2 \geq (q\alpha)^2 / \|\mathbf{w}^*\|^2$$

► **Upper bound:**  $\|\mathbf{w}_q\|^2 = \|\mathbf{w}_{q-1} + \mathbf{x}_{i_q}\|^2 = \|\mathbf{w}_{q-1}\|^2 + \|\mathbf{x}_{i_q}\|^2 + 2\mathbf{w}_{q-1} \cdot \mathbf{x}_{i_q}$  poiché  $d = +1$  e  $w_{q-1} \cdot x_{i_q} < 0$  (poiché  $x_{i_q}$  misclassificati) si ha

$$\|\mathbf{w}_q\|^2 \leq \|\mathbf{w}_{q-1}\|^2 + \|\mathbf{x}_{i_q}\|^2$$

Per iterazione, poiché si ha  $w_0 = 0$ , allora

$$\|\mathbf{w}_q\|^2 \leq \sum_{j=1}^q \|\mathbf{x}_{i_j}\|^2 \leq q\beta$$

- Mettendo insieme i bound si ha  $q\beta \geq \|\mathbf{w}_q\|^2 \geq (q\alpha)^2/\|\mathbf{w}^*\|^2$ .  
Definendo  $\alpha' = \alpha^2/\|\mathbf{w}^*\|^2$  si ottiene

$$q \leq \beta/\alpha'$$

□

## 4.2 Neural Network Unit

Ora vogliamo generalizzare il perceptron per poter costruire i singoli neuroni della neural network

### Funzione di attivazione

Possiamo ridefinire l'output di una singola unità. Finora abbiamo visto il perceptron che restituisce come output la funzione a scalino del net e l'adaline che restituisce il net stesso.

La funzione che l'unità restituisce in output è chiamata **funzione di attivazione**. Poichè vogliamo usare il metodo dei minimi quadrati per minimizzare l'errore di una neural network abbiamo bisogno di una funzione differenziabile anche se vogliamo che si comporti in modo simile al perceptron.

Due funzioni che rispettano questa proprietà sono:

- **Sigmoide:**  $f_\sigma(x) = \frac{1}{1+e^{-ax}} \in (0, 1)$
- **Tangente iperbolica:**  $\tanh(ax/2) \in (-1, 1)$

**Osservazione 4.5.** *a è detto slope parameter e influenza la pendenza e la larghezza della zona di linearità della funzione. Anche questo parametro è importante per il controllo della complessità in quanto uno slope parameter troppo piccolo aumenterebbe a dismisura la zona di linearità riducendosi al modello lineare, riducendo la VC dimension e quindi perdendo complessità*

Per effettuare task di classificazione, ad esempio per la sigmoide, è possibile considerare  $d = +1$  quando  $f_\sigma(\text{net}) > 0.5$  e  $d = 0$  quando minore.

E' possibile fare anche cose un po' più sofisticate come modificare la threshold oppure aggiungere un' intervallo di incertezza (**rejection zone**) entro il quale l'algoritmo lascia il dato non classificato evitando decisioni fragili. Queste sono decisioni che ovviamente vanno prese studiando le efficienze.

Le possibilità di scelta per una funzione di attivazione sono innumerevoli:

- RBF  $f(x) = e^{-ax^2}$  con  $x = \|w - x_{input}\|$
- Stochastic: +1 con probabilità  $P(\text{net})$  o altrimenti (Boltzmann machine)
- ReLU  $\max(0, x)$
- Softplus  $\ln(1 + e^x)$  (versione smooth della ReLU)
- SoftMax  $\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$  dove  $\sigma : \mathbb{R}^K \rightarrow \left\{ z \in \mathbb{R}^K \mid z_i > 0, \sum_{i=1}^K z_i = 1 \right\}$

## Apprendimento

Vogliamo trovare la delta rule per una generica funzione di attivazione (con buone proprietà, come minimo differenziabile con derivata continua) tramite i minimi quadrati e il gradient descent.

$$E(\mathbf{w}) = \sum_p (d_p - f(\mathbf{w} \cdot \mathbf{x}_p))^2$$

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \eta \nabla_{\mathbf{w}} E(\mathbf{w}_n) = \mathbf{w}_n + \eta \sum_p (d_p - f(\mathbf{w}_n \cdot \mathbf{x}_p)) f'(\mathbf{x}_p \cdot \mathbf{w}_n) \mathbf{x}_p$$

**Def. 4.6.**  $\delta_p = (d_p - f(\mathbf{w}_n \cdot \mathbf{x}_p)) f'(\mathbf{x}_p \cdot \mathbf{w}_n)$

In questo modo  $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta \sum_p \delta_p \mathbf{x}_p$

**Osservazione 4.7.** ► Dal gradiente dell'errore sarebbe uscito un 2 che però è stato riassorbito nel learning rate

- E' sempre una regola di correzione MA cambia pesi anche in caso di corretta classificazione (c'è il termine label-out che è nullo in caso di corretta classificazione)
- Nel caso di saturazione (verso gli asintoti) la derivata tende a 0. Questo ci suggerisce che partire con pesi elevati è una pessima idea poiché essendo la derivata quasi nulla i pesi variano molto poco ad ogni passo
- Il parametro di slope, a causa della derivata della funzione, influenza direttamente sul rate di apprendimento

## 4.3 MultiLayer Perceptron

L'idea è quella di costruire una rete di unità interconnesse per costruire una funzione di ipotesi tramite composizione di funzioni non lineari.

Le unità sono connesse tramite connessioni pesate e le unità sono raggruppate sotto forma di layer.

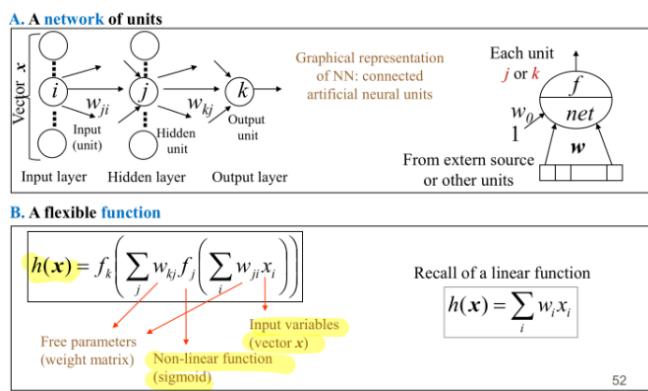


Figura 4.8: Multi Layer Perceptron

La rete è caratterizzata dalle funzioni di attivazioni delle unità, dal numero di layer e di unità, dalla topologia e dal suo algoritmo di apprendimento.

Il processo di feedforwarding consiste, per ogni dato di input:

1. Il dato di input è caricato nel layer di input (non viene applicato il net, semplicemente copiato)
2. Si calcola l'out di ogni hidden layer fornendo gli out come input al layer successivo
3. si calcola l'errore nel layer di output. Nel caso di classificazione multclasse il layer di output

In questo modo si ha una rete feedforward che "propaga" i dati in avanti nella rete. E' possibile usare topologie più complicate come le reti ricorrenti in cui alcune connessioni tornano indietro nella rete o addirittura in se stessi.

## Interpretazione del modello

La funzione trovata alla fine del training è una funzione non lineare nei pesi. Nel caso di rete a 2 layer si ha

$$h(x) = f_k \left( \sum_j w_{kj} f_j \left( \sum_i w_{ji} x_i \right) \right)$$

Gli hidden layer trasformano gli input, fungono da cambio di variabile tramite la composizione di funzioni non lineari

**Achtung!! 4.8.** La non linearità della funzione di attivazione è necessaria in quanto se si usano funzioni di attivazioni lineari è come avere una rete con un unico layer (lineare). Questo perchè la composizione di funzioni lineari è lineare

## NN come approssimante

Le neural network sono degli approssimatori universali ovvero una MLP può approssimare arbitrariamente bene qualsiasi funzione dagli input agli output. Una rete a singolo hidden layer con funzioni di attivazioni logistiche può approssimare qualsiasi funzione continua dato un numero sufficiente di unità (con maggiori layer si potrebbe migliorare il risultato)

L'espressività delle NN è influenzato dal numero di unità che è legato alla **VC dimension** (poichè il numero di pesi è proporzionale al numero di unità) e dalla sua architettura.

Inoltre la complessità è influenzata anche dal valore dei pesi in quanto per pesi piccoli si è nella zona lineare della funzione di attivazione mentre per pesi grandi si è nella zona di saturazione e la complessità è maggiore.

### In genere la complessità di una NN dipende da tutti gli iper-parametri

Inductive bias L'inductive bias delle NN è che si considera lo spazio delle ipotesi come lo spazio delle funzioni continue. Questo ha senso in quanto ci aspettiamo che per piccole variazioni nell'input si abbiano piccole variazioni nell'output. Una funzione non continua come un generatore di numeri casuali non può essere generalizzata

## Problemi generali

- ▶ Il modello è spesso over parametrizzato
- ▶ Il problema di ottimizzazione è non convesso e instabile
- ▶ Vanno controllati numerosi iper-parametri per il controllo della complessità

Inoltre data una rete neurale il problema di capire se esiste un set di pesi tali che la rete sia consistente con gli esempi è un problema NP completo. In pratica una rete neurale può essere trainata in un tempo ragionevole ma la soluzione ottimale non è garantita

## 4.4 Backpropagation

La backpropagation è l'algoritmo di apprendimento per una rete neurale feed-forward che descrive come aggiornare i pesi dei layer intermedi.

Quello qui sotto rappresenta la backpropagation per singolo pattern.

Il percorso che segue la BP è determinato dai dati, dal modello, dal rate di apprendimento, dal criterio di stopping e dai valori iniziali dei pesi

## BACK PROPAGATION

Consideriamo l' errore sul singolo pattern

$$E_p = \frac{1}{2} \sum_n (d_n - o_n^L)^2$$

dove  $\sum_n$  è la somma sui neuroni di output

Sia  $W^l$  matrice dei pesi tra il layer  $l-1$  e layer  $l$   
 $w_{st}^l$  peso tra unità  $s$  layer  $l-1$  e unità  $t$  layer  $l$

$$L = \# \text{ layer}$$

$$o_n^l = f_n^l (\sum_i w_{in}^l o_i^{l-1}) \text{ output neurone } n \text{ del layer } l$$

Def OUTER PRODUCT  $\vec{A} \otimes \vec{B})_{ij} = A_i B_j$

HADAMARD PRODUCT  $\vec{A} \circ \vec{B} = A_n B_n$  Senza sommare su  $n$

È un vettore. Prodotto componente per componente

- Layer di output (Sulle 8 si assume notaz Einstein)

$$\begin{aligned} \frac{\partial E}{\partial w_{st}^L} &= -\sum_n (d_n - o_n^L) \frac{\partial o_n^L}{\partial w_{st}^L} \\ &= -\sum_n (d_n - o_n^L) f'_n (\sum_i w_{it}^L o_i^{L-1}) \sum_i S_i S_t o_i^{L-1} = \\ &= -(d_t - o_t^L) f'_t (\sum_i w_{it}^L o_i^{L-1}) o_t^{L-1} \end{aligned}$$

$$\Rightarrow \frac{\partial E}{\partial w^L} = - [(\vec{d} - \vec{o}^L) \circ \vec{f}^L (W^L \vec{o}^{L-1})] \otimes \vec{o}^{L-1}$$

vettore riga                  vettore colonna

Def  $\vec{s}_L \vec{s}_L = (\vec{d} - \vec{o}^L) \circ \vec{f}^L (W^L \vec{o}^{L-1})$

- Layer  $L-1$

$$\begin{aligned} \frac{\partial E}{\partial w_{st}^{L-1}} &= -\sum_n (d_n - o_n^L) \sum_j \frac{\partial o_n^L}{\partial o_j^{L-1}} \frac{\partial o_j^{L-1}}{\partial w_{st}^{L-1}} = \\ &= -\sum_{n,j} (d_n - o_n^L) \frac{\partial}{\partial o_j^{L-1}} f_n^L (\sum_i w_{in}^L o_i^{L-1}) \frac{\partial}{\partial w_{st}^{L-1}} f_j^{L-1} (\sum_n w_{nj}^{L-1} o_n^{L-2}) \\ &= -\sum_{n,j} (d_n - o_n^L) f'_n (\sum_i w_{in}^L o_i^{L-1}) (\sum_i S_{ij} w_{ik}^L f_k^{L-1} (\sum_n w_{nj}^{L-1} o_n^{L-2})) \sum_n S_{nh} \delta_{tj} o_h^{L-2} \\ &= -\sum_n (d_n - o_n^L) f'_n (\sum_i w_{in}^L o_i^{L-1}) w_{tn}^L f_t^{L-1} (\sum_n w_{nt}^{L-1} o_n^{L-2}) o_s^{L-2} \end{aligned}$$

MA abbiamo definito  $s_n^L = (d_n - o_n^L) f_n^L(z_i w_{in}^L o_i^{L-1})$

$$\Rightarrow \frac{\partial E}{\partial w_{in}^{L-1}} = - \left[ \underbrace{(w^L s_L) o^L}_{s^{L-1}} f^{L-1}(w^{L-1}^\top o^{L-2}) \right] \otimes o^{L-2}$$

- Regole generali per iterazione (all'indietro)

- $s^L = (\vec{o}^L - \vec{out}) o^L f^L(w^{L-1}^\top o^{L-1})$
- $s^j = (w^{j+1} s_{j+1}) o^j f^j(w^{j-1}^\top o^{j-1})$
- $\frac{\partial E}{\partial w^j} = -s_j \otimes o^{j-1}$
- $w^j(n+1) = w^j(n) + \eta s_j \otimes o^{j-1}$

## 4.5 Problemi nel training delle NN

### Valori di partenza

I pesi sono usualmente inizializzati casualmente con valori piccoli intorno allo zero.

Bisogna assolutamente evitare:

- ▶ Valori nulli: La rete non apprende in quanto l'algoritmo di backpropagation include un prodotto tra i pesi e la delta
- ▶ Valori molto alti: Portano le funzioni di attivazione in saturazione e la loro derivata è nulla. Anche in questo caso non c'è apprendimento
- ▶ Valori tutti uguali: In questo caso i pesi cambiano tutti nello stesso modo, c'è forte instabilità

Alcuni metodi utili sono, definito il **fan-in** come il numero di input di un layer

- ▶ Distribuzione uniforme tra  $-m$  ed  $m$  dove  $m = \text{range}_{\text{dati}} * 2/\text{fan-in}$
- ▶ **basic**: o per i bias e uniforme in  $[-1/a, 1/a]$  con  $a = \sqrt{\text{fan-in}}$
- ▶ **Xavier initialization**: risultato recente, leggi se interessato

### Minimi multipli

La loss è fortemente non convessa nei parametri e possiede numerosi minimi locali e punti di sella (crescono col crescere delle dimensioni del problema)

**NB: PROVA SEMPRE diverse configurazioni di valori iniziali (5-10) In quanto il punto di arrivo dipende fortemente da questi**

**Suggerimento:** Fai la media degli errori ottenuti con diversi starting values e la varianza per valutare il modello. Successivamente puoi:

- ▶ Prendere il modello con l'errore minore
- ▶ Considerare i diversi modelli ottenuti con diversi starting point e fare la media degli output (**Committee approach**)

In realtà il problema dei minimi locali è un falso problema. Spesso un buon minimo locale è sufficiente. Ricordiamo che stiamo cercando il minimo del rischio reale (a noi inaccessibile), non del rischio empirico.

Spesso fermiamo il training in un punto a gradiente non nullo, quindi neanche in un minimo locale.

Inoltre le NN creano uno spazio delle ipotesi di dimensione variabile: la VC dimension aumenta durante il training e l'errore diminuisce. Il problema diventa fermare l'apprendimento prima che si raggiunga una condizione di overtraining (ovvero di overfitting) a prescindere dal problema dei minimi locali

## Tipo di apprendimento

Possiamo scegliere diversi modi di aggiornare i pesi, il più generico è chiamato **mini-batch**.

Nel mini batch creiamo  $k$  sottoinsiemi dei dati contenenti  $m$  dati ciascuno e sommiamo sugli  $m$  dati di ogni sottoinsieme il gradiente della loss prima di aggiornare i pesi. Questo va fatto iterativamente per ogni sottoinsieme per coprire la totalità dei dati.

- ▶  **$m=1$  stochastic/online:** In questo caso viene fatto l'aggiornamento dei pesi per ogni singolo dato. Questo può avere un effetto di regolarizzazione aggiungendo un rumore al processo di apprendimento.  
E' anche utile per evitare punti di sella e minimi locali in quanto il percorso seguito è casuale MA può essere instabile e non è possibile parallelizzarlo.  
**E' buona regola fare il calcolo sui dati in ordine casuale ad ogni step per evitare pattern o bias nella discesa del gradiente**
- ▶  **$m=N$  (su tutti i dati) Batch:** In questo modo si media sul rumore dei dati ottenendo una stima del gradiente più accurata. Il processo di apprendimento in questo modo è più lento ma può essere parallelizzato

E' possibile fare un ulteriore improverment del mini batch creando il **Minibatch stochastic method**. Si ottiene semplicemente facendo uno shuffle dei dati a ogni step e ricreando i sottoinsiemi del minibatch

## Rate di apprendimento

Un rate di apprendimento alto rende l'apprendimento più veloce ma potrebbe rendere il processo instabile.

Genericamente con apprendimento tipo **batch** è possibile usare valori più alti in quanto la stima del gradiente è più accurata

La stabilità della curva di apprendimento è importante (**CONTROLLALA SEMPRE**), bisogna sempre evitare una curva di apprendimento irregolare o noisy **Suggerimento:** è utile usare la media del gradiente sull'epoca di appren-

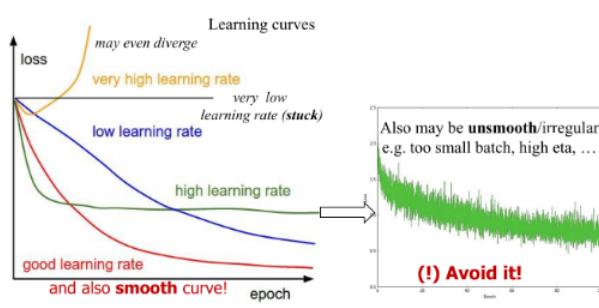


Figura 4.9: Learning curves

dimento (Somma dei gradienti /  $N$ )

Questo corrisponde a usare il **Least mean square** o a fissare  $\eta = \eta/l$

Ricorda che il minibatch su m dati in ogni sottoinsieme richiede un eta più piccolo quanto più m è basso

## 4.6 Miglioramenti nell'apprendimento

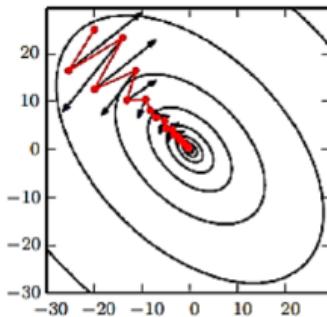
### Momentum

Definiamo una nuova regola di aggiornamento dei pesi

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \eta \Delta \mathbf{w}_n + \alpha \Delta \mathbf{w}_{n-1}$$

In questo modo le fluttuazioni nel percorso di discesa del gradiente vengono smorzate e si ha un effetto di inerzia che ci permette di usare learning rate maggiori. Il momentum è particolarmente utile nell'apprendimento di tipo

- Effect on a canyon of the error surface (# poor conditioning of the Hessian matrix).
- In red the path with the momentum, lengthwise traversing the valley toward the minimum ...
- instead of zig-zag along the canyon walls (following the black directions given by the gradient)



It shows the gradient for each red point descending with the moment

Figura 4.10: Momentum

batch ma è possibile usarlo anche nell'apprendimento di tipo stocastico.

Alla fine è come se fosse una media mobile su tutti i gradienti precedenti.

Nel caso del nminibacth equivale alla media mobile dei gradienti precedenti su esempi diversi per ogni sottoinsieme

### Nesterov Momentum

Questa variante prevede prima di sommare il vecchio gradiente e poi di calcolare il nuovo

1.  $k_n = w_n + \alpha \Delta w_{n-1}$
2.  $w_{n+1} = w_n - \eta \partial_w E(k_n) + \alpha \Delta w_{n-1}$

Quindi

$$\Delta w_n = \alpha w_{n-1} - \eta \partial_w E(w_n + \alpha \Delta w_{n-1})$$

Questo approccio mostra miglioramenti nel rate di convergenza nell'apprendimento di tipo batch, NON nel caso stocastico/minibatch

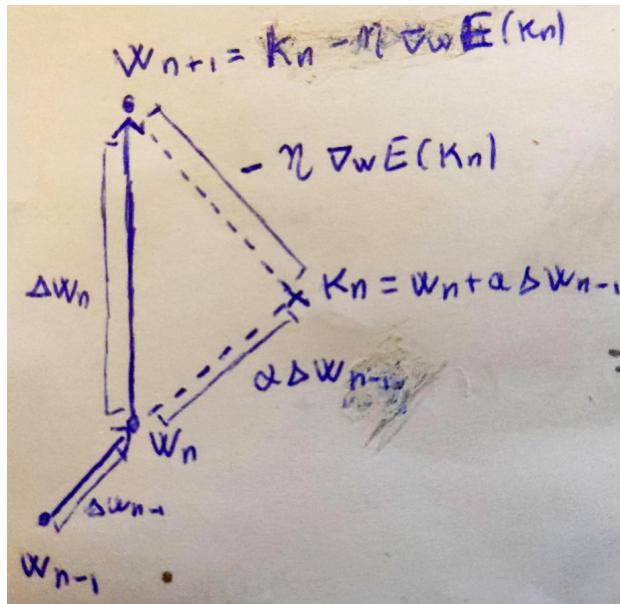


Figura 4.11: Nesterov accelerated momentum

## Learning rate variabile

Usando il minibatch il gradiente non scenderà mai a zero nelle vicinanze del minimo, per questo **bisogna evitare di fissare il learning rate**.

Una possibile scelta potrebbe essere il **decadimento lineare** del learning rate  $\eta_n = (1 - \frac{n}{\tau})\eta_0 + \frac{n}{\tau}\eta_t au$  dove  $\tau$  è il numero massimo di iterazioni dopo il quale il learning rate non cambia più e  $\eta_0$  è il learning rate iniziale.

L'idea è quella di partire con un learning rate alto e diminuirlo linearmente fino a un valore limine arbitrariamente piccolo.

*soltanente*  $\eta_\tau \approx 0.01\eta_0$ ,  $\tau \approx 100$

NB. La scelta di  $\eta_0$  rimane arbitraria e potrebbe essere necessaria una grid search.

Esistono anche altri metodi di learning rate adattivo che, possibilmente, evitano l'introduzione di altri iperparametri come  $\tau$  o  $\eta$ . Alcuni esempi sono [4]:

- **Adagrad:** Usualmente usato con il mini-batch.

Viene introdotto un parametro  $\delta$  piccolo ( $\approx 10^{-7}$  per stabilità).

Sia  $r_n = \sum_{i=0}^n g_i \odot g_i = r_{n-1} + g_n \odot g_n$  con  $g_i$  gradiente i-esimo, allora il  $\Delta w_n$  diventa

$$\Delta w_n = -\frac{\eta_0}{\delta + \sqrt{r_n}} \odot g_n$$

**Problema:** Performa bene in caso di problema convesso ma nel caso delle NN l'accumulazione della radice del gradiente **dall'inizio del training** può causare un **decadimento prematuro** del learning rate.

(potrebbe essere utile iniziare il decadimento dopo un certo numero di step)

Inoltre potrebbe succedere la situazione in cui il learning rate inizia a rallentare poiché in presenza di una sella ma una volta scavallata, dipendendo dall'intera storia del gradiente, il learning rate muore e i pesi non scendono nel minimo

► **RMSProp**: adattamento dell'Adagrad a problemi non convessi.

L'idea è quella di cambiare l'accumulazione del gradiente in una media mobile esponenzialmente pesata per liberarsi della storia più remota del gradiente in modo tale da non morire dopo aver scavallato una sella.

Il funzionamento è analogo all'Adagrad solo che, definito un **rate di decadimento**  $\rho$  e  $\delta \simeq 10^{-6}$  l'accumulo del gradiente è definito come

$$\mathbf{r}_n = \rho \mathbf{r}_{n-1} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$$

► **Adam**: Nell'ADAM il momento è direttamente incorporato come una stima del momento di primo ordine con peso esponenziale del gradiente. Inoltre Adam include la correzione del bias della stima del momento (di distribuzione) di primo ordine e del momento di secondo ordine non centrato.

(Anceh RMSProp include la stima del momento di secondo ordine ma manca di un fattore correttivo che può portare a un bias rilevante nel training).

Genericamente Adam è ritenuto abbastanza robusto nella scelta degli iperparametri anche se il learning rate iniziale spesso va riadattato.

Definiti due learning rate e due variabili si momento  $\mathbf{s}$  e  $\mathbf{r}$

- Update della stima del momento primo biased  $\mathbf{s}_n = \rho_1 \mathbf{s}_{n-1} + (1 - \rho_1) \mathbf{g}_n$
- Update della stima del momento secondo biased  $\mathbf{r}_n = \rho_2 \mathbf{r}_{n-1} + (1 - \rho_2) \mathbf{g}_n \odot \mathbf{g}_n$
- Correzione del bias del momento primo  $\hat{\mathbf{s}}_n = \frac{\mathbf{s}_n}{1 - \rho_1^n}$
- Correzione del bias del momento primo  $\hat{\mathbf{r}}_n = \frac{\mathbf{r}_n}{1 - \rho_2^n}$
- $\Delta W_n = -\eta_0 \frac{\hat{\mathbf{s}}_n}{\delta + \sqrt{\hat{\mathbf{r}}_n}}$

► Adamax, Adadelta, etc..

► Metodi di approssimazione al secondo ordine: sono algoritmi come il metodo di newton ma mediamente falliscono miseramente in problemi non convessi come le NN.

Inoltre il numero di punti di sella cresce esponenzialmente con la dimensione e i metodi al secondo ordine che sfruttano l'hessiana sono molto sensibili a i punti in cui il gradiente si annulla, per questo non vengono molto usati. Il gradient descent lineare, invece, mostra (empiricamente) che esce molto facilmente dalle selle

► Metodi Hessian Free: metodi al secondo ordini che non sfruttano il calcolo della matrice hessiana.

Es. Gradiente coniugato

► Altre soluzioni per velocizzare la convergenza è usare algoritmi di apprendimento differenti dalla backpropagation come ad esempio Rprop o Quickprop

Come tutto quello che abbiamo visto finora non esiste una scelta migliore in senso assoluto, tutto dipende dal tipo di task che vogliamo risolvere.

Ad ogni modo la discesa del gradiente stocastica con momento rappresenta un buon **punto di partenza**

## Criteri di stopping

Alcuni criteri usati sono:

- ▶ **Mean error:** Quando l'errore medio scende sotto un certo valore fissato (richiede un'ottima conoscenza dei dati e della loro tolleranza)
- ▶ **Max error**
- ▶ **Classification error:** numero di dati misclassificati
- ▶ **Gradient norm:** Quando la norma del gradiente scende sotto un certo valore (NB: può risultare in stopping prematuro soprattutto per valori bassi di  $\eta$  o in presenza di selle.  
Può essere utile se applicato dopo un certo numero di epoche)

In ogni caso tieni a mente che conviene fermarsi dopo un numero eccessivo di epoche (convergenza troppo lenta) ma EVITA DI FISSARE UN NUMERO ARBITRARIO DI EPOCHE e di fermarti con un errore di training troppo basso

## 4.7 Overfitting e regolarizzazione

Ricorda che non vogliamo minimizzare il rischio empirico (overfitting) ma quello reale.

Il processo di apprendimento inizia con dei pesi casuali piccoli. Mano a mano che si procede con l'apprendimento le unità degli hidden layer tendono a saturare aumentando il numero effettivo di parametri liberi (introducendo non linearità) e quindi aumentando la VC dimension. (Ricordiamo che le NN forniscono uno spazio delle ipotesi di dimensioni variabili, che cambia durante il training)

Per evitare l'overfitting ci sono due strade (oltre alla scelta del modello):

- ▶ **Early stopping:** usando il validation set per determinare quando fermarsi (ovvero quando l'errore sul validation set inizia a crescere).  
Non è banale capire esattamente quando fermarsi anche perché l'errore del validation set può fluttuare nelle epoche.  
Inoltre nota che poiché il numero effettivo di parametri liberi cambia durante il training, alterare il training stesso implica limitare la complessità effettiva del modello
- ▶ **Regolarizzazione:** Abbiamo già visto la regolarizzazione di Tikhonov. Tipicamente si usa con valori di  $\lambda \approx 0.01$  e il suo valore viene scelto nella fase di scelta del modello tramite il validation set

Alcune questioni che è utile tenere a mente sono:

- E' importante sottolineare che **La regolarizzazione non è una tecnica per controllare la stabilità della convergenza del training ma per controllare la complessità del modello (la VC dimension)**

- il **bias**  $w_0$  è omesso dalla regolarizzazione poiché la sua inclusione fa sì che i risultati non siano indipendenti da shift/scaling del target. Volendo può essere incluso ma con un coefficiente a parte
- Tipicamente applicato alla versione Batch.  
Per la versione stochastic/mini-batch se lo si vuole comparare con la versione batch conviene usare un  $\lambda_{batch} = \lambda_{batch} \cdot \frac{mb}{\#patterns}$

Esistono altri tipi di regolarizzazione come la weight elimination.

Per chiarezza definiamo:

- **Loss:** Errore quadratico (medio o non) + termine di penalizzazione.  
E' la funzione che andiamo a minimizzare tramite il gradient descent
- **Error/risk:** Errore quadratico (medio o non).  
E' la funzione con il quale valutiamo l'errore del modello
- **Pruning methods:** metodi con cui vengono eliminati pesi e connessioni nella rete

Tutte le tecniche viste finora sono combinabili e si possono usare insieme.

Si può anche scegliere se nel termine  $\Delta w_{n-1}$  (usato nel momento) includere o non il termine di regolarizzazione (tipicamente è incluso)

## 4.8 Numero di unità

Il numero di unità influisce pesantemente sulla complessità del modello (poche unità = underfitting, troppe unità = overfitting) ed è un problema riguardante la model selection.

Soltanamente il numero di unità, come altri iperparametri, viene scelto tramite cross-validation confrontando risultati con diversi numeri di unità.

**NB:** il numero di unità "ottimale" è influenzato anche dagli altri iperparametri. Ad esempio in caso di regolarizzazione potrebbe essere necessario aumentare il numero di unità all'aumentare di  $\lambda$ .

Ci sono due strade percorribili per gestire automaticamente il numero di unità:

- **Pruning methods** Partendo da una rete molto grande vengono progressivamente eliminati pesi o unità
- **Approcci costruttivi:** L'algoritmo di apprendimento decide autonomamente quando aggiungere unità a un layer partendo da una configurazione iniziale. (es. Classificazione=Tower,Tiling,Upstart, Regressione=Cascade correlation)

### Cascade correlation

La cascade correlation fa parte degli approcci costruttivi. E' una tecnica con cui la rete determina automaticamente la propria dimensione e topologia durante la fase di apprendimento

1. Inizia con una configurazione iniziale  $N_0$  senza hidden units.
2. Fai il training per  $N_0$  e calcola errore
3. Se  $N_0$  non può risolvere il problema passa a rete  $N_1$  in cui è aggiunta una hidden unit in modo tale che la **correlazione tra l'output dell'unità e l'errore residuo della rete  $N_0$  sia massimizzata tramite il training dei pesi della nuova hidden unit**

La correlazione è [3]

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right| = \sum_o S_o$$

dove:

$o$  è la somma sulle unità di output

$p$  è la somma sui pattern

$V$  è l'output dell'hidden unit che stiamo aggiungendo

$E_{p,o}$  è l'errore residuo del pattern  $p$  sull'output  $o$

$\bar{V}$  e  $\bar{E}_o$  sono le medie su tutti i pattern di  $V$  ed  $E$

Per massimizzare  $S$  dobbiamo calcolare  $\partial_{w_i} S$  ovvero la derivata di  $S$  rispetto i pesi dell'unità che vogliamo aggiungere.

Supponendo che  $\bar{V}$  non dipenda da  $w_i$

$$\partial_{w_i} S = \sum_{p,o} \text{sign}(S_o)(E_{p,o} - \bar{E}_o) \partial f_p(\text{net}) I_{i,p}$$

dove  $I_{i,p}$  è l'input dell'unità che stiamo aggiungendo che viene dall'unità  $i$  relativo al pattern  $p$

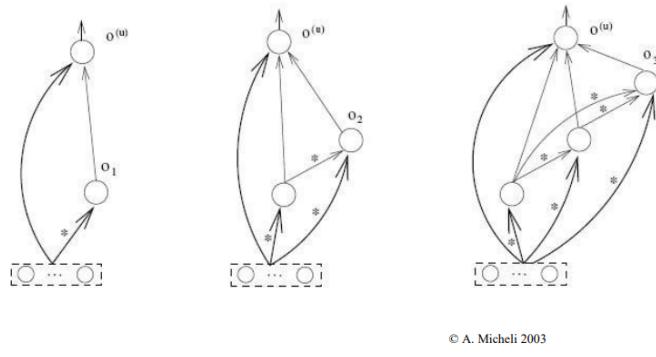
Per finire dobbiamo fare il gradient ascent ovvero

$$w_i^{(n+1)} = w_i^{(n)} + \partial_{w_i} S$$

4. Dopo il training i pesi della nuova unità vengono congelati (non vengono più trainati negli steps successivi) e i pesi rimanenti (dell'output layer) vengono ritrainingati
5. Se la rete  $N_1$  non risolve il problema viene aggiunta un'altra unità che ha come input l'input layer e le unità aggiunte precedentemente tramite cascade correlation
6. L'algoritmo procede in questo modo iterativamente fino alla risoluzione del problema (raggiungimento di una condizione che soddisfa un criterio di stopping come ad esempio il raggiungimento di una threshold dell'errore) aggiungendo progressivamente unità

Il ruolo delle hidden unit è quello di risolvere uno specifico sotto problema agendo da "feature detector"

**Pool:** La massimizzazione della correlazione è ottenuta tramite gradient ascent su una superficie con molti massimi. Per evitare di cadere in massimi locali si può fare il training di un pool di unità simultaneamente (ogni unità inizializzata con pesi diversi) e poi scegliere l'unità che genera la correlazione massima



**Figura 4.12:** Cascade correlation: Ad ogni step viene aggiunta una unità che ha come input l'input layer e tutte le unità aggiunte tramite cascade correlation. I pesi delle unità aggiunte negli step precedenti sono congelati

**NB:** Genericamente la cascade correlation ha bisogno di regolarizzazione in quanto porta facilmente a overfitting

## 4.9 Input/Output

Il preprocessing dei dati può avere un effetto notevole.

Soltanente le tecniche più usate sono:

- **Standardization:** Per ogni feature riscalare i dati in modo tale che abbiamo medio zero e deviazione standard 1
- **Rescaling:** Riscalare i dati i modo tale che siano tutti compresi nel range [0,1]

Inoltre bisogna scegliere la funzione di attivazione del layer di output in base al task da eseguire

- **Regessione:** Output lineare
- **Classificazione:** Output in 1 of k encoding (questo perchè altrimenti se nn indecisa tra classe 1 e classe 5 potrebbe dare classe 3)

- Sigmoids: Bisogna scegliere una threshold per assegnare la classe (e caso mai una zona di reiezione)
- Tanh (logistica simmetrica): solitamente apprende più velocemente [5]
- Può essere usato 0.9 al posto di 1 come valore di target per evitare convergenza asisntotica
- E' possibile usare, per target 0/1, la **Softmax** che può essere interpretata come la probabilità della classe  $p(class = 1|x)$ .

$$o_k(x) = \frac{e^{-net_k}}{\sum_{j=1}^K e^{-net_j}}$$

- E' possibile usare come loss per task di classificazione binaria invece che lo scarto quadratiko la **Binary crossentropy** ovvero - la

loglikelihood di una binomiale

$$\text{Loss} = - \sum d_i \log(\text{out}(x_i)) + (1 - d_i) \log(1 - \text{out}(x_i))$$

# Validation 5

Il dataset di training non può fornire una stima dell'errore di test in quanto fornisce il rischio empirico (in caso di underfitting si ha un alto bias, in caso di overfitting si ha una grande varianza).

Il nostro obiettivo è quello di trovare dei metodi per fornire una stima del rischio reale di un modello.

Esistono tecniche **analitiche** come il criterio di informazione bayesiana (BIC) o lo structural risk minimization (SRK) ma si usano più spesso tecniche di stima empirica tramite il validation set

La validazione ha due obiettivi:

- ▶ **Model Selection:** stima delle performance di diversi modelli al fine di scegliere il migliore. **Ritorna un modello**
- ▶ **Model Assessment:** Dopo aver scelto il modello, alla fine, stima del rischio su dei dati di test **TOTALMENTE nuovi**. **Misura della qualità del modello scelto**

La regola cruciale (Golden Rule) è: **I DATI USATI PER QUESTI DUE OBIETTIVI DEVONO ESSERE TOTALMENTE SEPARATI.**

**Non usare i dati di test per fare model selection, il test va fatto come ultimissimo passaggio dopo aver scelto il modello, dopo aver fatto il test non si può più tornare indietro**

Se il test viene usato più volte quello che si ottiene è un efficienza maggiore di quella reale poiché abbiamo usato il test set per scegliere il modello che performa meglio su quello specifico set di dati, quindi abbiamo usato il test set come validation set.

Il test set andrebbe separato dal resto dei dati all'inizio, prima di fare qualsiasi altra cosa, e non essere toccato fino alla fine.

## Dati e sampling

Il processo di apprendimento e di testing è molto sensibile ai dati stessi.

Spesso viene effettuata una procedura di **Stratificazione**: è il processo con il quale si raggruppano membri di una popolazione in sottogruppi omogenei (es. Nel caso di classificazione binaria è bene che ci siano approssimativamente tanti o quanti 1).

Spesso il sampling con il quale si separano dati di training e validation è fatto in maniera randomica.

Nel caso si abbiano pochi dati è difficile capire come ripartizionare i dati in maniera ottimale

Oltre la stratificazione conviene evitare di avere feature o classi mancanti nel training e cercare di identificare e rimuovere gli outliers (soprattutto se si ha una conoscenza a priori dei dati)

Inoltre conviene fare attenzione al fatto che il blind test set non sia formato da una distribuzione differente, misurato in un'altra scala, uncleaned, non preprocessato come i dati di training o fuori dal range dei dati

## Grid Search

La generalization capabilities di un modello è fortemente influenzata dagli iperparametri scelti.

Quello che si può fare è creare una griglia di iperparametri e per ogni possibile combinazione fare il trainind della rete e alla fine scegliere gli iperparametri che generano il modello più performante.

La grid search ha un alto effort computazionale anche se è facilmente parallelizzabile. Quello chesi può fare per velocizzare il processo è fare una prima grid search con certi valori abbastanza grossolani dei parametri e poi fare una grid search successiva (o più di una) con valori sempre più densi intorno i valori trovati nelle grid search precedenti

## 5.1 Tecniche di validazione

Ricordiamo che

- ▶ Per la classificazione sul singolo patter L=1 se misclassificato o altrimenti
- ▶ Per la regressione  $L = (h(x) - y)^2$
- ▶ Siano  $Z_1 \dots Z_l$  un l-tupla di sample di una pdf  $p(z)$  indipendenti e sia  $D_l = \{z_1 \dots z_l\}$  una particolare istanza
- ▶ L'obiettivo è minimizzare il valore d'aspettazione del rischio reale

$$R(h)E_Z[L(z, h)] = \int_Z L(z, h)p(z)dz$$

selezionando  $h^* = \operatorname{argmin}_{h \in H} R(h)$

PROBLEMA: la pdf  $p(z)$  è sconosciuta e non abbiamo accesso a tutti gli  $L(z, h)$  ma solo al rischio empirico

Consideriamo tutte le tecniche più comuni considerando si a la validation che il test

### Hold out validation

Consiste semplicemente nello splittare il dataset in dataset di training, validation <https://it.overleaf.com/project/61426ddc1eb1fce6d957eco4> e test

**Merge: Training – Validation – Test (Hold Out)**

- Select a hypothesis space with **hyper-parameter**  $\theta$
- **Divide** your dataset (training set)  $D_l$  into three parts  $D^{TR}, D^{VL}, D^{TS}$ ,  $tr + vl + ts = l$
- For each value  $\theta_m$  of the hyper-parameter  $\theta$  [grid search]
  - **select**  $h_{\theta_m}^*(D^{TR}) = \arg \min_{h \in H_{\theta_m}} R_{emp}(h, D^{TR})$  [training]
  - **let**  $R_{emp}(h_{\theta_m}^*(D^{TR}), D^{VL}) = \frac{1}{vl} \sum_{z_i \in D^{VL}} L(z_i, h_{\theta_m}^*(D^{TR}))$
- **select**  $\theta_m^* = \arg \min_{\theta_m} R_{emp}(h_{\theta_m}^*(D^{TR}), D^{VL})$  [best on VL set]
- **select**  $h^*(D^{TR} \cup D^{VL}) = \arg \min_{h \in H_{\theta_m^*}} R_{emp}(h, D^{TR} \cup D^{VL})$  [retraining]
- **estimate**  $R(h^*(D^{TR} \cup D^{VL}))$  with  $\frac{1}{ts} \sum_{z_i \in D^{TS}} L(z_i, h^*(D^{TR} \cup D^{VL}))$



(retraining is not shown)

**Figura 5.1:** Hold out: Il dataset viene semplicemente spartito in 3 parti**K fold cross validation**

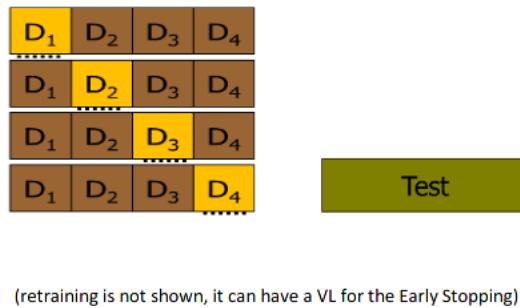
Per prima cosa separiamo il dataset di test da quello di training e validazione.  
Dopo di che il dataset di training viene diviso in k partizioni.

A rotazione useremo una partizione come dataset di validazione e il resto come dataset di training (ogni volta bisogna rifare il training)

Per scegliere il modello è possibile fare la media dei risultati sul validation test di ogni modello e prendere il modello che minimizza la media delle loss sui differenti validation set

### Merge: K-fold CV for model selection + Hold Out Test

- Select a hypothesis space with **hyper-parameter**  $\theta$
- Divide your dataset  $D_l$  in two parts  $D^{TR}$  and  $D^{TS}$ ,  $tr + ts = l$
- For each value  $\theta_m$  of the hyper-parameter  $\theta$  [grid search]
  - estimate  $R(h^*(D^{TR}))$  with  $D^{TR}$  using a K-fold CV
- select  $\theta_m^* = \arg \min_{\theta_m} R(h_{\theta_m}^*(D^{TR}))$  [best from the K-fold CVs]
- select  $h^*(D^{TR}) = \arg \min_{h \in H_{\theta_m^*}} R_{emp}(h, D^{TR})$  [retraining]
- estimate  $R(h^*(D^{TR}))$  with  $\frac{1}{ts} \sum_{z_i \in D^{TS}} L(z_i, h^*(D^{TR}))$



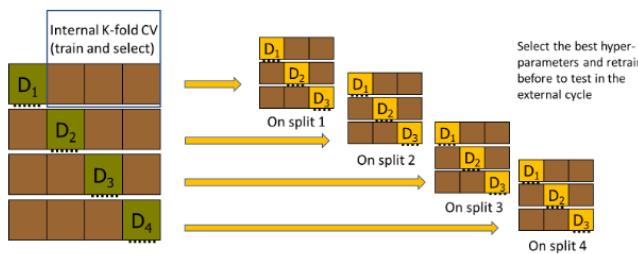
**Figura 5.2:** k-fold cross validation con  $k=4$ . Per stabilire quale modello scegliere si può fare la media sui vari kfolds oppure usare altre tecniche statistiche

### Double kfold cross validation

Consiste nel fare un kfold tra training+validation e test e per ogni partizione fare un k-fold tra training e validation sul dataset di training+validation **NB:**

#### Merge: Double K-fold CV (also known as Nested K-fold CV)

- Select a hypothesis space with **hyper-parameter**  $\theta$
- Divide your dataset (training set)  $D_l$  into  $K$  distinct and equal parts  $D^1, \dots, D^K, \dots, D^K$
- For each part  $D^k$  (and its counterpart  $\bar{D}^k$ ) [**external cycle**]
  - select  $h^*(\bar{D}^k)$  by an internal K-fold CV on  $\bar{D}^k$  (with  $K'$  that can be  $\neq K$ ) [**inner cycle**]
  - estimate  $R(h^*(\bar{D}^k))$  with [test error for each external split]
$$R_{emp}(h^*(\bar{D}^k), D^k) = \frac{1}{|D^k|} \sum_{z_i \in D^k} L(z_i, h^*(\bar{D}^k))$$
- estimate  $R(h^*(D_l))$  with  $\frac{1}{K} \sum_k R(h^*(\bar{D}^k))$  [the mean over the test folds]



**Figura 5.3:** Double k-fold oppure Nested k-fold cross validation

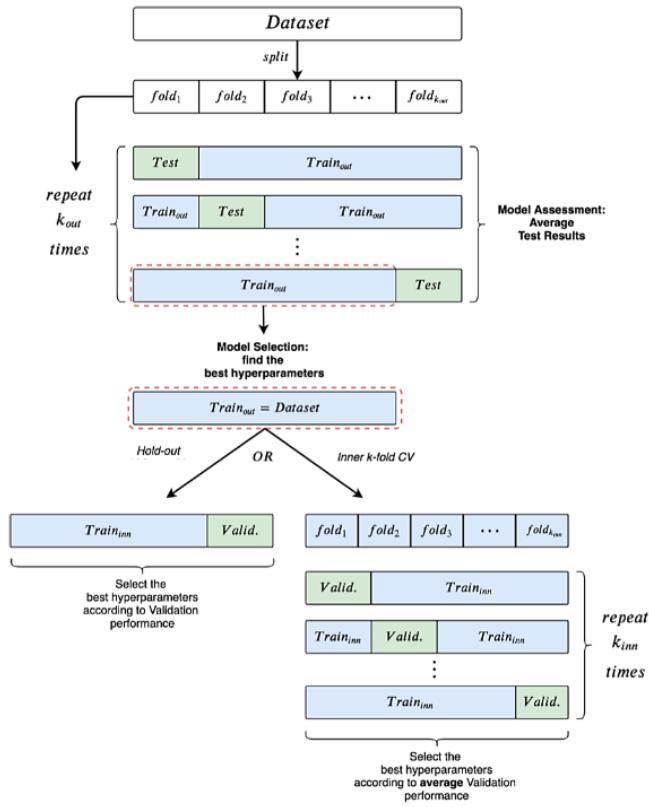
Questa tecnica non fornisce un modello finale unico poiché potremmo differenti modelli su kfolds di test diversi.

Se è necessario un modello finale è possibile fare una processo di selezione separato tra i vari modelli che hanno performato meglio sui singoli kfolds di

validazione

Il nested kfold massimizza l'uso dei dati per il training e l'evaluation del modello ma ha un alto effort computazionale

Nested/Double CV (a further different view, also with the inner Hold Out case)



**Figura 5.4:** E' possibile combinare il k fold sia con un altro k fold che con un hold out

## Quale scegliere?

- L'hold out è semplice e a basso effort computazionale però il sampling condiziona fortemente la qualità dei dati, può essere fortunato o meno. I risultati di validazione sono soggetti ad alta varianza
- k-fold cross validation: con  $k=\#$ dati usiamo tutti i dati per fare validazione e test ma ha un effort computazionale molto grande può avere un comportamento strano.  
Più  $k$  è piccolo più dati vengono "persi" (10 fold perde il 10% dei dati) ma l'effort computazionale si riduce

Quando si fa un kfold potrebbe essere un idea utile tentare il kfold con vari  $k$ , magari a potenze di 2. In ogni caso 5-fold o 10-fold sono un buon compromesso basato sulla stima di bias e varianza

Nel caso di knn e tutti gli altri modelli non parametrici usare un  $k$  alto (anche uguale al numero di pattern non aumenta l'effort computazionale)

## 5.2 Pitfalls e riflessioni

### Early stopping

Fissare un numero arbitrario di epoche per il training non è una buona idea, si rischia di cadere in underfitting o overfitting.

Anche selezionare il numero di epoche dai risultati della validazione non è una buona idea anche perchè il modello finale verrà ritrainingato su tutti i dati, compresi quelli di validazione e il numero di epoche ottimale potrebbe variare.

Alcune tecniche di early stopping sono:

- ▶ fermare il training quando la variazione della loss è sotto una threshold tenendo comunque conto di una patience (una tolleranza, in modo tale che il training non venga fermato appena raggiunta la threshold).  
Questa tecnica richiede l'uso di un validation set quindi alla fine non è possibile ritrainingare il modello su tutti i dati
- ▶ Rendere il numero di epoche un iperparametro e tramite la cross validation fare la media del numero di epoche ottimale sopra i k folds.  
**Rimane il problema che quando si va a fare il training su tutti i dati il numero ottimale di epoche potrebbe cambiare**
- ▶ Si Considera il training error nel punto in cui è minore il validation error. Quando si fa il retraining su tutti i dati ci si ferma nel punto in cui il training error raggiunge il valore precedente osservato

### Weight inizialization

Diversi starting point per i pesi portano a diversi modelli.

Solitamente si tentano diversi starting point calcolando media e varianza dell'errore/accuratezza per i diversi tentativi. L'obiettivo è ridurre il valore di aspettazione dell'errore (la media) ma avere una varianza elevata significa avere un modello particolarmente instabile.

Per scegliere il migliore starting point usare sempre il validation set.

### Selezione sequenziale

All'inizio del training vanno scelti quali parametri tenere fissati e quali testare nella grid search.

Fare una prima grid search con dei parametri fissati e lasciarli liberi in una seconda grid search equivale a introdurre un bias

# 6

## Bias-Variance decomposition

Il training set è solo un "microstato" di tutti i possibili sottoinsiemi dei dati.  
Differenti training set possono dare risultati diversi

L'errore su vari training set si può scomporre in 3 termini: rumore, varianza e bias.

Supponiamo che la funzione vera sia  $y = f(x) + \epsilon$  dove  $\epsilon$  è un termine di rumore gaussiano a media nulla e deviazione standard  $\sigma$ .

Dato un set di 1 esempi consideriamo il caso di regressione lineare con loss l'errore quadratico.

Nota che essendoci limitati alla classe di funzioni lineari avremmo sempre un **errore sistematico**

Assumendo che tutti i dati siano indipendentemente e identicamente distribuiti con una pdf P, dato un nuovo esempio  $x$  l'errore su questo sarà

$$E_P[(y - h(x))^2]$$

Nota che per ogni differente training set si avrà una  $y$  e una  $h$  (funzione trovata) diversa

Questo valore di aspettazione si decompone in 3 parti:

Usando il noto risultato

$$Var(x) = E[x^2] - \bar{x}^2 \implies E[x^2] = Var(x) + \bar{x}^2$$

possiamo scrivere

$$E[(y-h)^2] = E[y^2 + h^2 - 2yh] = Var(f(x)+\epsilon) + E[y]^2 + Var(h) + E[h]^2 - 2E[(f(x)+\epsilon)h(x)]$$

Ricordando che  $E[y] = E[f + \epsilon] = E[f]$  e  $Var(y) = Var(\epsilon) = \sigma^2$  e che, poiché  $\epsilon$  e  $h$  sono indipendenti,  $E[\epsilon h] = E[\epsilon]E[h]$ , si ha

$$\begin{aligned} &\implies (E[f(x)]^2 - 2E[f(x)h(x)] + E[h(x)]^2) + \sigma^2 + Var(h(x)) = \\ &= \sigma^2 + Var(h(x)) + E^2[f(x) - h(x)] \end{aligned}$$

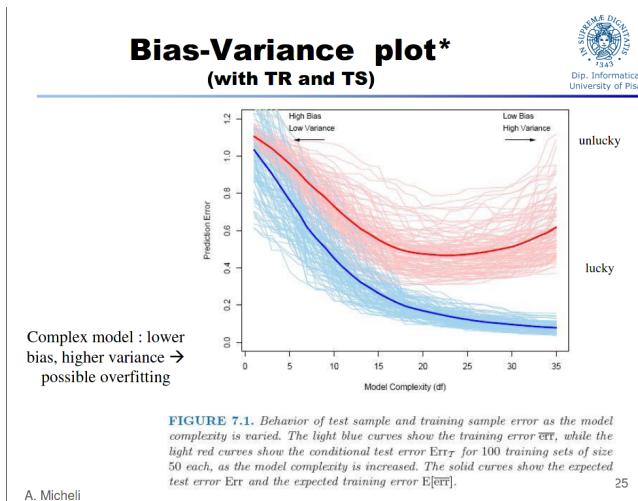
Quindi l'errore si decompone in:+

- ▶  $\sigma^2$ : Rumore dovuto al termine gaussiano nei label. Non dipende dal modello
- ▶  $Var(h)$ : Varianza del modello. Quantifica la variabilità della risposta del modello  $h$  per diversi training set. Cresce al crescere della flessibilità del modello
- ▶ Bias quadrato  $E^2[f - h]$ : Rappresenta la media della discrepanza tra la funzione vera e il nostro modello. Cresce al diminuire della flessibilità (modelli rigidi sono biased)

Ovviamente, poichè siamo interessati a minimizzare l'errore totale è necessario un tradeoff tra bias e varianza in modo tale che sia minimizzata la loro somma.

## 6.1 Regolarizzazione

Ricordiamo che nella loss il parametro aggiuntivo  $\lambda \|\mathbf{w}\|^2$  riduce la complessità del modello al crescere di lambda quindi al crescere di lambda diminuiamo la varianza ma aumentiamo il bias.



**Figura 6.1:** Andamento dell'errore al variare della complessità del modello

## 6.2 Ensemble Learning

L'ensemble learning è costituito a una serie di tecniche di insieme che usano modelli multipli per ottenere una migliore prestazione predittiva rispetto ai modelli da cui è costruito.

Per esempio è possibile, in caso di classificazione, far classificare lo stesso pattern a più classificatori e prendere come scelta definitiva la classe più votata oppure, in caso di regressione, fare una media dei risultati dei modelli

### Bagging

Questa tecnica mira a creare un insieme di classificatore aventi la stessa importanza ovvero il voto di ogni classificatore ha lo stesso peso.

Nel bagging più modelli dello stesso tipo vengono addestrati su training set diversi ottenuti tramite bootstrap ovvero una tecnica di campionamento con reimmissione.

Il bagging migliora le performance di modelli con alta varianza in quanto questa sui diversi modelli alla fine tende a mediare a zero

## Boosting

Nel boosting ciascun classificatore influisce sulla votazione finale con un certo peso. Tale peso sarà calcolato in base all'errore di accuratezza che ciascun modello commetterà nella fase del learning.

I modelli vengono generati consecutivamente dando sempre più peso agli errori effettuati nei modelli precedenti.

In algoritmi come l'adaboost l'output finale è dato dalla somma pesata delle predizioni dei singoli modelli. Ogni qualvolta un modello viene addestrato ci sarà una fase di ripesaggio delle istanze. L'algoritmo darà un peso maggiore alle istanze misclassificate nella speranza che il modello successivo sia più esperto su queste ultime.

Il boosting, a differenza del bagging mira a ridurre il bias e, incredibilmente, non aumenta la varianza (resiste all'overfitting). Soffre molto sul rumore (a cui è assegnato un peso maggiore)

# Statistical learning theory

7

Ci chiediamo se il training sample, consistente in N esempi indipendenti identicamente distribuiti, contengano sufficiente informazioni per raggiungere un grado di generalizzazione sufficiente.

## 7.1 Shattering and VC dimension

La VC dimension misura la complessità dello spazio delle ipotesi  $H$  NON dal numero di ipotesi distinte  $|H|$  MA dal numero di istanze diverse da  $X$  che possono essere completamente discriminate usando  $H$ .

**Def. 7.1.** *Shattering (frammentazione) :Dato  $X$  spazio delle istanze e  $H$  spazio delle ipotesi si dice che  $H$  frammenta  $X$  se  $H$  può rappresentare tutte le possibili dicotomie di  $X$ . (ovvero esiste un ipotesi  $h$  in  $H$  tale che il task sia risolvibile con 0 errori)*

**NB:** In questo linguaggio malato "tutte le possibili dicotomie" significa "spazio delle parti di  $X$ " che ricordiamo avere cardinalità  $2^N$  con  $N$  cardinalità di  $X$

La capacità di frammentare l'insieme  $X$  è strettamente legata all'inductive bias dello spazio delle ipotesi, uno spazio unbiased frammenta sempre lo spazio  $X$ .

Se  $H$  non frammenta  $X$  potrebbe comunque frammentare un suo sottoinsieme. Più è largo questo sottoinsieme più le capacità di generalizzazione di  $H$  sono alte. Basandosi su questo concetto si definisce la Vc dimension

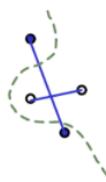
**Def. 7.2.** *VC dimension La VC dimension di una classe di funzioni  $H$  è la massima cardinalità di un sottoinsieme di punti in  $X$  che può essere frammentata da  $H$*

Esempio:  $VC(H)=p \implies H$  frammenta almeno un sottoinsieme di  $X$  costituito da  $p$  punti e nessun sottoinsieme costituito da almeno  $p+1$  punti

Se un insieme finito arbitrariamente grande di  $X$  può essere frammentato da  $H$  allora la VC dim è infinita

**Osservazione 7.3.** ▶ Potrebbe succedere che non tutte le possibili configurazioni di  $N$  punti possano essere frammentate (ad esempio 3 punti allineati non sono linearmente separabili) ma è sufficiente trovare una sola configurazione di  $N$  punti tale che lo siano

- $\text{VC}(\mathcal{H}) \geq 3$  (shown on the 3 points)
- $\text{VC}(\mathcal{H}) < 4$



Can always draw six lines between pairs of four points.  
Two of those lines will cross.  
If we put points linked by the crossing lines in the same class they can't be linearly separated (do you remember the XOR?)  
So a line can shatter 3 points but not 4

- Conclusion:  $\text{VC}(\mathcal{H}) = 3$ ,  $\mathcal{H}$  hyperplanes in  $\mathbb{R}^2$  (LTU)

**Figura 7.1:** In  $\mathbb{R}^2$  la VC dimension per  $\mathcal{H}$  classe di funzioni lineari è 3

- Un generale la VC dimension per una classe di funzioni lineari (iperpiani) in uno spazio  $n$  dimensionale  $n+1$

**NB** La VC dim è legata ai numeri di parametri del modello ma non è la stessa cosa, può anche esistere un modello con un singolo parametro e VC dim infinita (altro esempio, la vc dim del 1-nn è infinita poiché si hanno sempre errori sul training set. Il numero di parametri, corrispondente al numero di istanze, è infinito e anche il numero di parametri, poiché dato da  $n/k$ . Infatti con  $k=1$  abbiamo overfitting, l'errore è molto alto così come la vc dimension)

## 7.2 Bound analitici sul rischio

Esistono diverse formulazioni di bound in accordo con diverse classi di task e di funzioni.

Ad esempio in caso di classificazione binaria per la loss 0/1 il bound  $R < R_{emp} + \epsilon$  è dato da

$$\epsilon(VC, N, \delta) = \sqrt{\frac{VC(\ln(\frac{2N}{VC}) + 1) - \ln(\frac{\delta}{4})}{N}}$$

con probabilità di almeno  $1 - \delta$   $\forall VC < N$

Questo ci fornisce un modo per stimare l'errore sui dati futuri basandoci unicamente sul training error e sulla vc dimension di  $\mathcal{H}$

**Osservazione 7.4.** ► Per molte classi di funzioni "ragionevoli" (es. funzioni lineari) la VC dim è lineare nel numero dei parametri.

- Questo mostra che per arrivare a una buona capacità di generalizzazione abbiamo bisogno di un numero sdi esempi che sia lineare con la vc dimension

## Structural risk minimization

E' possibile usare la vc dimension come unparametro di controllo per minimizzare il bound su  $R$ .

Assumendo un VC finita possiamo definire una struttura annidata di spazi di ipotesi in modo tale che valga

$$\mathcal{H}_1 \subseteq \dots \subseteq \mathcal{H}_n$$

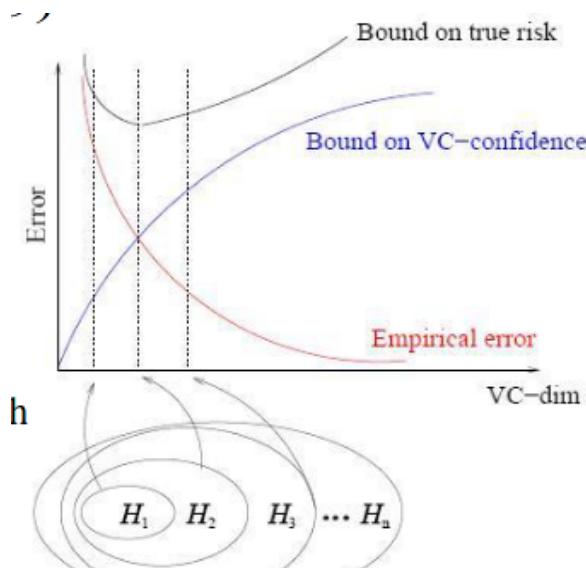
$$VC(\mathcal{H}_1) \leq \dots \leq VC(\mathcal{H}_n)$$

**esempi:**

- ▶ NN con un numero crescente di unità o epoche (NB iperparametri influiscono su vc dim)
- ▶ Polinomi con un grado crescente

Al crescere della VC dimension l'errore empirico decresce e la VC confidence cresce.

Usare l'SRM per scegliere un modello equivale a scegliere il modello  $h$  con il miglior bound sul rischio reale



Questo nella pratica non funziona molto bene in quanto la vc confidence è molto conservativa e può risultare centinaia di volte più grande dell' effetto empirico dell'overfitting oltre al fatto che può essere molto difficile calcolare la VC confidence per una classe di modelli arbitraria

E' possibile però adottare due approcci pratici:

- ▶ Scegliere una struttura appropriata del modello, minimizzare il training error usando tecniche di regolarizzazione (es. early stopping,...,che equivalgono a tecniche implice di SRM)
- ▶ Fissare il training error e minimizzare la VC confidence (è quello che fa il support vector machine)

In ogni caso SRM fornisce una base teorica su cui si basano tutti i principi del machine learning

# 8

## Support Vector Machine

Consideriamo un problema di classificazione binaria linearmente separabile in cui non si hanno errori nei dati (toy model).

Dato un training set  $T = \{(x_i, d_i)\}_{i=1}^N$  la soluzione consiste nel trovare l'i-iperpiano  $w^T x + b = 0$  che separa gli esempi.

Dovrà essere  $w^T x_i + b \geq 0$  per  $d_i = +1$  e  $w^T x_i + b \leq 0$  per  $d_i = -1$

Abbiamo assunto  $g(x) = w^T x + b$  come funzione sdiscriminante e  $h(x) = \text{sign}(g(x))$  come ipotesi

La soluzione non è unica, esistono infinite rette che separano linearmente i punti del training set.

### 8.1 Hard margin SVM

Volendo possiamo definire un margine  $\rho$  definito come il doppio della distanza tra l'iperpiano e il/i punto/i più vicini (nel margine non ci sono punti). Anche in questo caso però esistono soluzioni diverse che differiscono per il margine

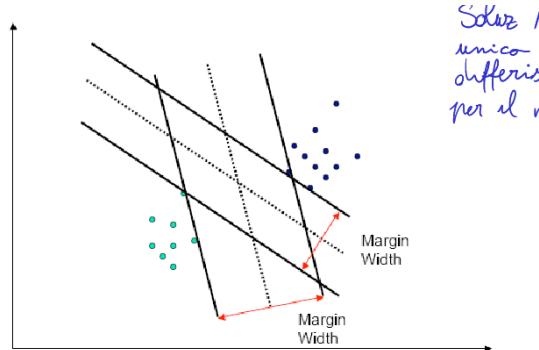


Figure: Varying the separating hyperplane the margin varies as well.

**Figura 8.1:** Variando la larghezza del margine otteniamo iperpiani idversi

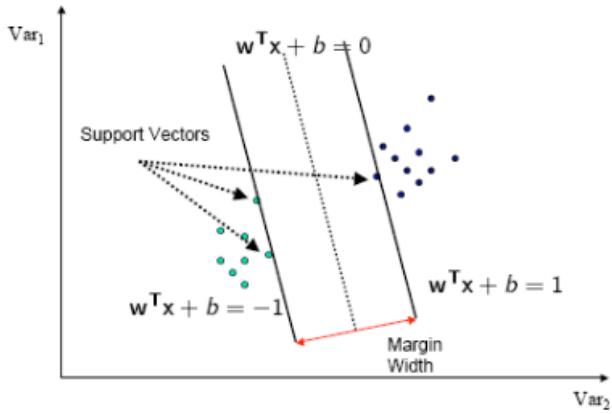
**Def. 8.1.** Viene chiamato iperpiano ottimale l'iperpiano  $w_0^T + b_0 = 0$  che massimizza il margine  $\rho$

**Teorema 8.2.** Massimizzare il margine  $\rho$  equivale a minimizzare la norma dei pesi  $\|w\|$  poiché vale  $\rho = \frac{2}{\|w\|}$ . Quindi il margine è un iperparametro di regolarizzazione

**Def. 8.3. Support vector:** Un vettore di supporto  $x^{(s)}$  è un vettore che soddisfa l'equazione

$$d^{(s)}(w^T x^{(s)} + b) = 1$$

Ovvero i punti più vicini all'iperpiano



**Figura 8.2:** I support vector sono i punti più vicini all'iperpiano

*Dimostrazione.* ► Ricordiamo che per risolvere il problema deve essere

$$w^T x_i + b \geq 0 \text{ se } d_i = +1$$

$$w^T x_i + b \leq 0 \text{ se } d_i = -1$$

- Possiamo riscalare  $w$  e  $b$  in modo tale che i support vector soddisfino  $|g(x_i)| = |w^T x_i + b| = 1$
- I bound scritti sopra, essendo i support vector i punti più vicini all'iperpiano, diventano più forti

$$w^T x_i + b \geq 1 \text{ se } d_i = +1$$

$$w^T x_i + b \leq -1 \text{ se } d_i = -1$$

oppure in forma più compatta

$$d_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \dots, N$$

- Dato  $g(x) = w_o^T x + b_o$  con  $\mathbf{w}_o$  il vettore ortogonale all'iperpiano definiamo  $r$  come la distanza tra un generico punto  $x$  e l'iperpiano

$$\mathbf{x} = \mathbf{x}_p + \mathbf{r} \frac{\mathbf{w}_o}{\|\mathbf{w}_o\|}$$

dove  $x_p$  è la proiezione del punto  $x$  sull'iperpiano

- 

$$g(x) = g(x_p + r \frac{w_o}{\|w_o\|}) = g(x_p) + r \mathbf{w}_o^T \frac{\mathbf{w}_o}{\|\mathbf{w}_o\|}$$

Poiche  $x_p$  appartiene all'iperpiano, per definizione  $g(x_p) = 0$  quindi

$$g(x) = r \|w_o\| \implies r = \frac{g(x)}{\|w_o\|}$$

- Consideriamo la distanza tra l'iperpiano e un support vector positivo. Per definizione la distanza tra iperpiano e support vector è  $\rho/2$  e, sempre

per definizione vale anche  $g(x^{(s)}) = 1$  quindi vale

$$\rho = \frac{2}{\|\mathbf{w}_0\|}$$

- Quindi l'iperpiano ottimale massimizza  $\rho$  che implica minimizzare  $\|\mathbf{w}\|$

□

SVM è un problema di ottimizzazione quadratica.

La forma primale del problema è:

### Hard Margin SVM: Problema primale

**Problema primale:** Dato un training sample T trovare i valori ottimali di  $w$  e  $b$  in modo tale da minimizzare

$$\psi(w) = \frac{w^T w}{2} \quad (\min.(\|\mathbf{w}\|))$$

in modo tali che sia soddisfatto il vincolo (con zero errori di classificazione)

$$d_i(w^T x_i + b) \geq 1 \quad \forall i = 1, \dots, N$$

Questo problema è risolvibile tramite i moltiplicatori di lagrange  $\alpha_i$ .

Definiamo una funzione lagrangiana

$$J = \frac{w^T w}{2} - \sum_{i=1}^N \alpha_i (d_i(w^T x_i + b) - 1)$$

Ogni termine della somma corrisponde a un vincolo e  $J$  deve essere minimizzata rispetto a  $w$  e  $b$  e massimizzata rispetto a  $\alpha$  (Quindi la soluzione corrisponde a un punto di sella di  $J$ )

**Teorema 8.4.** Il risultato è

$$w_0 = \sum_{i=1}^N \alpha_{0,i} d_i \mathbf{x}_i^{(s)}$$

$$b_0 = 1 - \sum_i \alpha_{0,i} d_i \mathbf{x}_i^{(s)T} \mathbf{x}_i^{(s)}$$

(dove  $x^{(s)}$  è un qualsiasi support vector positivo ( $d=+1$ ) e  $\sum_i x_i^{(s)}$  è la somma su tutti i support vector (positivi e negativi) )

*Dimostrazione.* ► Minimizziamo  $J$  per  $w$   $\partial_w J = 0 \implies \mathbf{w} = \sum_i \alpha_i d_i \mathbf{x}_i$

► Minimizziamo  $J$  per  $b$   $\partial_b J = 0 \implies \sum_i \alpha_i d_i = 0$

- ▶ Condizioni di Kuhn - Tucker: J va minimizzato nel dominio definito dal constraint e sul suo bordo. Definiamo g l'espressione che definisce il bordo allora, tenendo conto di entrambe le situazioni posso scrivere

$$\nabla J(x) + \alpha \nabla g(x)$$

Per minimizzare nell'interno dell'insieme dovrò porre  $\alpha_i = 0$ , per minimizzare sul bordo, considerando che  $g(x) = 0$  avrò  $\alpha_i \geq 0$

- ▶ Quindi le condizioni di Kuhn Tucker per il nostro problema saranno:

- Sul bordo, ovvero sui support vector,  $\alpha_i \geq 0$  con  $d_i(\mathbf{w}^T \mathbf{x}_i^{(s)} + b) = 1$
- All'interno del dominio definito dal constraint (oltre i support vector) con  $x_i$  NON support vector e  $\alpha_i = 0$

- ▶ Quindi il parametro dei pesi ottimale sarà

$$w_o = \sum_i \alpha_{o,i} d_i \mathbf{x}_i^{(s)}$$

*L'iperpiano dipende solo dai support vector*

Resta solo da trovare i valori ottimali di  $\alpha$

- ▶ Sostituendo w (ottenuto dalla derivata di J) nel problema primale e tenendo conto del nuovo constraint dato dalla derivata di b otteniamo il problema duale da massimizzare

$$Q(\alpha) = \sum_{i,j} \alpha_i - \frac{1}{2} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

con il constraint  $\alpha_i \geq 0$  AND  $\sum_i \alpha_i d_i = 0$

- ▶ Gli  $\alpha_{o,i}$  si ottengono tramite approcci di programmazione quadratica. Il costo computazionale scala linearmente con la cardinalità del training set
- ▶ Da  $b = 1 - w^T x^{(s)}$  ottengo il b ottimale corrispondente a un support vector positivo (con d=+1)

$$b_o = 1 - \sum_i \alpha_{o,i} d_i \mathbf{x}_i^{(s)T} \mathbf{x}^{(s)}$$

□

Quindi dopo aver trovato i moltiplicatori di lagrange si calcola  $f(x) = \mathbf{w}_o^T \mathbf{x} + b_o$  e si classifica x tramite  $\text{sgn}(f(x))$

**Osservazione 8.5.** ▶ Il vincolo è lineare in w e la  $\psi$  è quadratica e convessa. La

soluzione di questo problema scala linearmente con le dimensioni dello spazio degli input

- ▶ Abbiamo fissato a mano il training error (imponendolo a zero).
- ▶ Massimizzare il margine equivale a minimizzare la norma dei pesi e quindi a minimizzare la VC dimension e la VC confidence
- ▶ Hard margin SVM fornisce un metodo automatizzato per fare SRM e minimizzare la VC confidence nel processo di training

- E' una soluzione che si concentra solo su una parte specifica dei dati di training, i support vector
- E' ottimo nel caso ideale in quanto è un modello che non dipende dai punti più lontani che potrebbero essere anche degli outliers.
- Il problema è che è molto sensibile ai casi in cui i support vector sono soggetti a rumore

**Teorema 8.6. Vapnik:** Sia  $D$  il diametro della palla più piccola intorno i punti  $\mathbf{x}_1, \dots, \mathbf{x}_N$  e  $\#X$  la dimensione dell'input space.

Ogni classe di iperpiani separatori descritti dall'equazione  $w^T x + b = 0$  l'upper bound sulla VC dimension è

$$\text{VC} \leq \min\left(\frac{D^2}{\rho^2}, \#X\right) + 1$$

(E' un constraint, significa che la VC dimension può essere più piccola di  $\#X + 1$  cercando la soluzione regolarizzata con il margine massimo)

## 8.2 Soft margin SVM

Consideriamo il caso, più reale, in cui i punti nel dataset di training sono soggetti a rumore

Allora almeno un punto violerà la condizione

$$d_i(w^T x_i + b) \geq 1 \text{ (Violata)}$$

Ammettere punti all'interno della zona di margine ci permette di avere un margine più grande

**Def. 8.7. Slack Variables** Sono scalari non negativi

$$\xi_i \geq 0 \quad \forall i = 1, \dots,$$

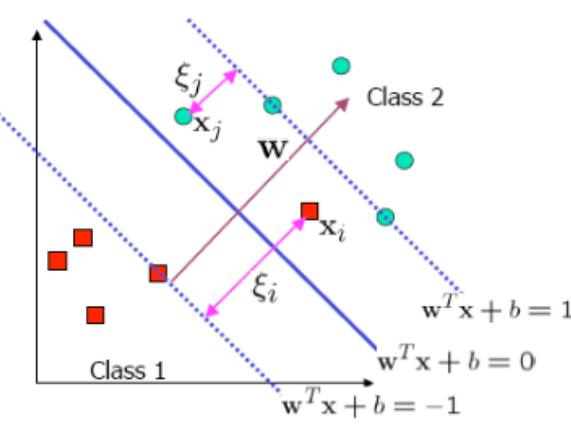
tali che

$$d_i(w^T x_i + b) \geq 1 - \xi_i$$

**Def. 8.8. (Soft margin) support vectors** Un support vector per il soft margin SVM soddisfa la condizione

$$d_i(w^T x_i + b) = 1 - \xi_i$$

**NB:** Per il soft margin NON vale il teorema di vapnik poichè all'interno del margine sono presenti altri punti



**Figura 8.3:** Soft margin SVM: la slack variables  $\xi_i$  è la distanza tra il punto i-esimo e il 'suo' margine ovvero il margine dal lato della classe a cui il punto i-esimo stesso appartiene

### Soft Margin SVM: Problema primale

Dato il training set  $T$  bisogna trovare i valori di  $w$  e  $b$  che minimizzano la funzione

$$\psi(w, \xi) = \frac{1}{2}w^T w + C \sum_i \xi_i$$

sotto il vincolo

$$d_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1, \dots, N$$

con  $\xi_i \geq 0 \quad \forall i$

Il risultato è

$$w_o = \sum_i \alpha_{o,i} d_i x_i$$

$$b_o = d_j - \sum_i \alpha_{o,i} d_i x_i^T x_j$$

Per un pattern j tale che  $0 < \alpha_j < C$

*Dimostrazione.* ▶ Come prima definiamo la funzione  $J$  con i moltiplicatori di lagrange  $\alpha$  a moltiplicare per la funzione che definisce il vincolo e minimizziamo per  $w$  e  $b$  più un termine  $\sum_i \mu_i \xi_i$  che forza le slack variables ad essere non negative ( $\mu_i$  sono altri moltiplicatori di lagrange=

- Nel caso del soft margin le condizioni di kuhn tucker sono

$$\alpha_i(d_i(w^T x_i + b) + \xi_i - 1) = 0 \quad \forall i = 1, \dots, N$$

$$\mu_i \xi_i = 0 \quad \forall i$$

- In questo modo otteniamo che sul margine (ovvero per i support vector) vale  $0 < \alpha_i < C \implies \xi = 0$  mentre all'interno del margine  $\alpha_i = C \implies \xi_i \geq 0$
- Gli alpha si ottengono sempre risolvendo il problema duale ottenuto dalla sostituzione delle condizioni trovate per la minimizzazione di  $J$  in  $J$

□

**Osservazione 8.9.**  $C$  è un iperparametro di regolarizzazione che va regolato in modo tale da raggiungere un tradeoff tra la minimizzazione del rischio empirico e della VC confidence

Questo significa che abbiamo perso la feature della SRM automatizzata. Ora siamo noi a regolare la minimizzazione del rischio tramite un parametro

Basso  $C \implies$  Rischio empirico alto  $\implies$  Underfitting

Alto  $C \implies$  Rischio empirico basso  $\implies$  Overfitting

Inoltre si noti che per  $C \rightarrow \infty$  si riottiene l'Hard Margin SVM poiché forza le  $\xi$  ad annullarsi

### 8.3 Non linear classification

Per problemi non linearmente separabili una soluzione potrebbe essere mappare i dati (anche in spazi di dimensionalità superiore) in modo tale che diventino linearmente separabili MA la scelta della mappa  $\varphi$  non è automatica ed è critica per la corretta soluzione del problema

In ogni caso la risolvibilità di un problema non lineare tramite SVM è garantita dal teorema di ricoprimento.

In ogni caso avere uno spazio di dimensionalità troppo alta può portare facilmente a overfitting se non si tiene sotto controllo la complessità

#### Non linear function map

Data una mappa  $\varphi : \mathbb{R}^{m_0} \rightarrow \mathbb{R}^{m_1}$  definiamo il training set come  $T = \{(\varphi(x_i), d_i)\}_{i=1}^N\}$

L'iperpiano che separa i dati nel nuovo spazio è  $\mathbf{w}^T \varphi(\mathbf{x}) + b = 0$

Definiamo per semplicità  $w_0 = b$  e  $\varphi_0(\mathbf{x}) = 1$ , in questo modo l'iperpiano è dato semplicemente da  $\mathbf{w}^T \varphi(\mathbf{x}) = 0$

**Questo non è altro che la linear basis expansion**

La soluzione del problema è

$$\mathbf{w} = \sum_i \alpha_i d_i \varphi(x_i)$$

con gli alpha soluzione del problema duale (identico al caso soft margin ma al posto delle  $x_i$  avremo  $\varphi(x_i)$ )

**NB** Le  $w$  sono da intendere nello spazio degli input trasformati in quanto deve rispettare la dimensionalità del codominio

**NB** La soluzione ottimale del problema dà come risultato una soluzione sparsa negli  $\alpha_i$  in quanto tutti i moltiplicatori di lagrange non corrispondenti a support vectors sono nulli

## Kernel Trick

Poichè l'iperpiano separatore è definito da

$$\sum_i \alpha_i d_i \varphi^T(x_i) \varphi(x) = 0$$

Spesso invece di definire la  $\varphi$  conviene definire un **kernel** come il prodotto scalare tra le  $\varphi$  ovvero

$$k : \mathbb{R}^{m_0} \times \mathbb{R}^{m_1} \rightarrow \mathbb{R} \quad k(\mathbf{x}_i, \mathbf{x}) = \varphi^T(\mathbf{x}_i) \varphi(\mathbf{x})$$

In questo modo è possibile definire la **kernel matrix** come

$$K_{ij} = k(x_i, x_j) = k(x_j, x_i)$$

**Teorema di Mercer:** Un kernel equivale a un prodotto scalare nel codominio solo se la kernel matrix è semidefinita positiva.

Alcune proprietà dei kernel sono:

- I kernel sono lineari ovvero la combinazione lineare (con coeff. positivi per preservare semidef. pos. della matrix) di kernel è un kernel ben definito
- $k = k_1 \cdot k_2$  è un kernel ben definito

Alcuni esempi di kernel sono:

- Polinomiale  $k = (x^T x_i + 1)^p$
- Radiale  $k = e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|}{2\sigma^2}}$
- **NB** Questo kernel corrisponde a una  $\varphi$  infinito dimensionale
- Two layer perceptron:  $k = \tanh(\beta_0 x^T x_i + \beta_1)$  con  $\beta_0 > 0$  e  $\beta_1 < 1$

Per classificare un pattern alla fine dovremo calcolare  $h(x) = \text{sgn}(\sum_i \alpha_i d_i k(x, x_i))$

**OSS:** Per la fase di test abbiamo bisogno di memorizzare gli  $x_i$  esattamente come facevamo nel knn

**Osservazione 8.10.** Volendo fare un parallelismo tra SVM e NN il numero di support vector è l'equivalente al numero di hidden unit ma è solo un parallelismo da non prendere troppo seriamente

## 8.4 Non Linear Regression

Supponiamo che la funzione reale che caratterizza i nostri dati sia  $d = f(x) + v$  dove  $v$  è un termine di rumore stocastico indipendente da  $x$

Possiamo stimare  $d$  usando una espansione lineare di funzioni non lineari  $\varphi$  ovvero

$$y = h(x) = w^T \varphi(x)$$

dove il bias è incluso in  $w$  (e  $\varphi_0 = 1$ )

Abbiamo bisogno di un modello che non sia troppo sensibile a piccoli cambiamenti nei parametri del modello e che quindi sia abbastanza robusto.

Uno stimatore particolarmente robusto è il minimax.

Quando la pdf del rumore è simmetrica il minimax consiste nella minimizzazione di  $L = |d - y|$  [5]

Per costruire una SVM possiamo estendere questa loss function e usare la **epsilon insensitive loss**

$$L = \theta(|d - y| - \epsilon)(|d - y| - \epsilon)$$

dove  $\epsilon$  è un parametro definito dall'utente

Anche in questo caso gli unici punti che portano alla soluzione sono i support

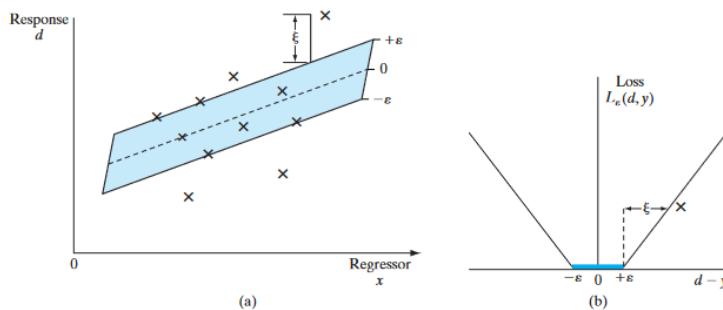


FIGURE 6.9 Linear regression (a) illustrating an  $\epsilon$ -insensitive tube of radius  $\epsilon$ , fitted to the data points shown as 'x's. (b) The corresponding plot of the  $\epsilon$ -insensitive loss function.

Ricordiamo che la minimizzazione dei minimi quadrati nel limite di alta statistica equivale allo stimatore di max likelihood ma vale solo nel caso valga l'ipotesi di regolarità della likelihood. Usare il valore assoluto al posto del quadrato porta a prestazioni peggiori nel caso ottimale di regolarità ma decisamente migliori nei casi più generici in cui la likelihood può essere molto irregolare

vector

Introducendo le slack variables non negative  $\xi'$  e  $\xi$  il problema di ottimizzazione è formulabile come

$$-\xi'_i - \epsilon \leq d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \quad \forall i = 1, \dots, N$$

This leads to the following constraints:

$$\left. \begin{array}{l} d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \\ \mathbf{w}^T \Phi(\mathbf{x}_i) - d_i \leq \epsilon + \xi'_i \\ \xi_i \geq 0 \\ \xi'_i \geq 0 \end{array} \right\} \forall i = 1, \dots, N$$

Il problema duale si ricava come già visto per il task di classificazione solo che qui avremo 2 moltiplicatori  $\mu$  per la positività delle  $\xi$  e 2  $\alpha$  per rispettare entrambi i constraint.

Alla fine oltre ad avere le stesse condizioni ottenute nel caso di soft margin (tutti

**Primal Problem**

Given the training set  $T = \{(\mathbf{x}_i, d_i)\}_{i=1}^N$  find the optimal values of  $\mathbf{w}$  such that the following objective function is minimized

$$\Psi(\mathbf{w}, \xi') = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N (\xi_i + \xi'_i)$$

under the constraints

$$\left. \begin{array}{l} d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \\ \mathbf{w}^T \Phi(\mathbf{x}_i) - d_i \leq \epsilon + \xi'_i \\ \xi_i \geq 0 \\ \xi'_i \geq 0 \end{array} \right\} \forall i = 1, \dots, N$$

gli  $\alpha < C$ ) avremo anche  $\sum_i \alpha_i - \alpha'_i = \sum_i \gamma_i 0$

Alla fine i valori ottimali saranno gli  $w = \sum_i \gamma_i \varphi(x_i)$  e i support vector saranno tutti gli  $x$  per cui  $\gamma_i(x)$  è non nulla

L'ipotesi ottimale finale sarà

$$h(x) = \sum_i \gamma_i \varphi^T(x_i) \varphi(x) = \sum_i \gamma_i k(x_i, x)$$

**Recap**

- ▶ Scegli il parametro  $C$  e il parametro  $\epsilon$
- ▶ Scegli il kernel  $k$
- ▶ Risolvi il problema di ottimizzazione per ottenere i valori dei moltiplicatori di lagrange
- ▶ Calcola il valore ottimale di  $w$  e successivamente di  $b$
- ▶ calcola  $h(x)$

## 8.5 Osservazioni

- ▶ I problemi principali di SVM sono dati dal fatto che bisogna comunque fare una scelta per i parametri, l'algoritmo è di tipo batch e problemi molto grossi possono avere un effort computazionale molto alto
- ▶ La scelta del kernel è fortemente impattante sulla VC dimension in quanto equivale a una linear basis expansion e avere una base troppo grande porta a overfitting
- ▶ In alcuni casi (non-lineari) si può arrivare ad avere un numero molto alto di support vector (anche la metà del totale)
- ▶ Le proprietà dell'hard margin SVM non si estendono al soft margin. L'hard margin rimane un toy model utile per mostrare i concetti del SRM e automatizzarlo
- ▶ Un kernel gaussiano con una varianza abbastanza piccola (mollificatore tendente alla delta) può classificare correttamente qualunque punto del training set e trasformando il modello praticamente in un 1-NN e quindi porta ad avere una VC dimension infinita.

Al contrario una varianza molto alta porta a fare una media globale sui

dati e quindi abbassa la VC dimension.

**La varianza della gaussiana risulta un ulteriore parametro utile al controllo della vc dimension**

- ▶ Come per le NN conviene selezionare i migliori parametri tramite cross validation e grid search
- ▶ La tecnica dei kernel può essere estesa ben oltre SVM e si basa sulla linear basis expansion e sul comparare coppie di punti.  
Un kernel può essere interpretato come una misura di somiglianza tra i due dati (tipicamente basato sulla metrica)

# Convolutional Neural Networks

9

Può essere necessario, soprattutto per task che hanno a che fare con immagini, che la NN sia in grado di estrarre features locali dai dati cambiando l'architettura della rete.

Inoltre tramite la condivisione di pesi tra le varie unità possiamo ridurre il numero di parametri liberi creando delle connessioni più complesse. In questo modo un layer diventa in grado di eseguire la stessa operazione su parti diverse dell'input

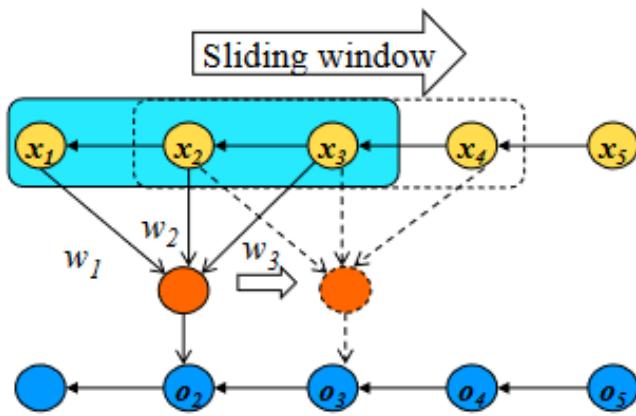
Esempio: 1-unit CNN con un array in input

Consideriamo un array di lunghezza N come input e una unità di CNN con kernel (inteso come funzione di convoluzione) una finestra di lunghezza M che ha come elementi i pesi dell'unità

Allora l'output sarà un array lungo N-M+1

$$out_t = f\left(\sum_{i=1}^M w_i x_{t+i-M+1}\right)$$

dove f è la funzione di attivazione (tipicamente una relu)



**Figura 9.1:** CNN a layer singolo con un array come input corrisponde semplicemente a fare una media mobile sull'array di input pesata con i pesi dell'unità

La convoluzione (discreta, su cui poi fa applicata elemento per elemento la funzione di attivazione) è generalizzabile al caso 2D (in generale in nD).

$$S(i, j) = \sum_{m,n} I(m, n)K(i - m, j - n)$$

dove I è l'immagine e K il kernel.

I pesi di una unità agiscono come un filtro che ha il ruolo di riconoscere

delle feature/pattern nelle immagini

- ▶ Più il kernel è piccolo più la rete tenderà a concentrarsi su feature più spazialmente concentrate (più locali) mentre kernel grandi tendono a estrarre feature più globali
- ▶ Lo stesso filtro è applicato all'intera immagine. Questo permette alla rete di individuare una feature in un'immagine a prescindere dalla sua posizione spaziale all'interno di essa.
- ▶ Questa proprietà è chiamata **Invarianza traslazionale**
- ▶ I filtri non sono impostati dall'utente ma la rete impara i filtri ottimali in autonomia
- ▶ Vari layer vengono sovrapposti in modo tale che i filtri possano estrarre feature via via sempre più grandi nell'immagine

## 9.1 Padding

Prima abbiamo visto che per un array lungo N e un kernel lungo M l'out ha dimensione  $N-M+1$ .

Una tecnica usata per gestire la dimensione dell'input è il padding

- ▶ **Valid Padding:** E' la convoluzione usuale, l'output ha dimensione  $N-M+1$
- ▶ **Same padding:** All'input vengono aggiunti  $M-1$  zeri (tipicamente in modo simmetrico ma può essere utile metterli da un solo lato nel caso di serie temporali) in modo tale che l'output abbia la stessa dimensione dell'input.
- ▶ Questo però introduce effetti di bordo.

## 9.2 Pooling

Man mano che si va avanti con i layer è necessario fare una riduzione della dimensionalità dei dati estraendo solo le feature rilevanti.

Per ottenere questo risultato introduciamo i layer di pooling

Un modo per sottocampionare i dati e ridurre la loro dimensionalità è introdurre lo stride nei layer convolutivi

Lo stride non è altro che il passo della convoluzione (più è alto più la dimensionalità è ridotta (stride più alto tende a dare meno importanza alla località delle feature))

$$S(i, j) = \sum_{m,n} I(m, n)K(i - s_1 m, j - s_2 n)$$

Nel caso di un'immagine definiamo un layer di pooling dotato di un kernel di dimensione  $M \times M$  e stride s (per semplicità considero il caso di kernel e stride simmetrico che è quello più usato a meno che non ci siano motivazioni sensate per evitarlo).

L'output avrà dimensione (in caso di assenza di padding) di  $N-M+2-S$ . Tipicamente il layer di pooling possono essere:

- **Max Pooling:** L'out è una matrice che a come elementi il massimo degli elementi delle sottomatrici ottenuti tramite la sliding windows sull'immagine
- **Average pooling** Come max pooling ma prendendo una media (aritmetica o pesata) al posto del massimo

In questo modo oltre a ridurre la dimensionalità e estrarre le feature rilevanti rendiamo il modello insensibile a piccole variazioni dei singoli pixel e quindi al rumore (La media del rumore su un numero sufficiente di pixel potrebbe mediare a zero, il massimo ignora la quasi totalità degli input e quindi è insensibile al rumore globale)

### 9.3 Architettura

**NB:** Genericamente viene chiamato 'Convolutional layer' l'insieme del convolutional layer + la funzione di attivazione

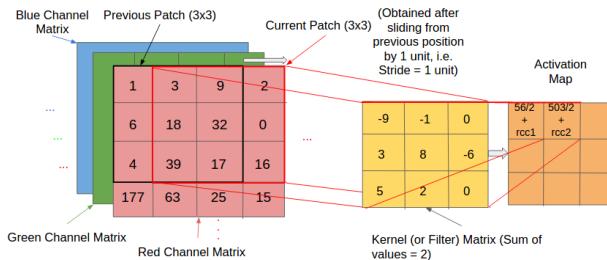


Figura 9.2: Funzionamento di un convolutional layer

Tipicamente una CNN viene costruita mettendo in serie, in maniera alternata, dei convolutional layer e dei layer di pooling al fine di estrarre le feature rilevanti.

Per eseguire il task di classificazione (o regressione (es. individuare una posizione all'interno dell'immagine)) viene usata una rete fully connected che ha come input l'output dell'ultimo layer di pooling appiattito (matrice trasformata in array)

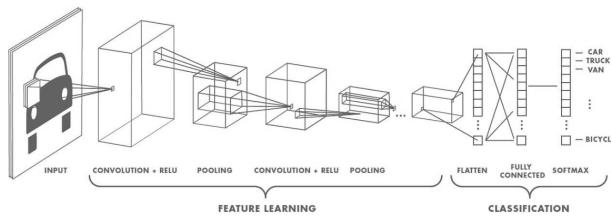


Figura 9.3: Struttura di una CNN (AlexNet)

### 9.4 Recap e osservazioni

- E' un modello che permette di estrarre feature locali e invarianti per traslazione
- Il concetto di convoluzione permette di ridurre il numero di parametri liberi

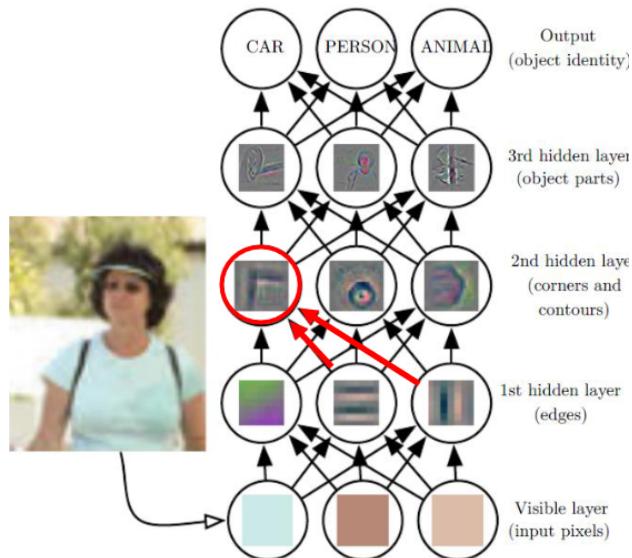
- ▶ Il pooling permette di fare subsampling e ridurre la dimensionalità dei dati rendendo il modello meno sensibile al rumore dei singoli pixel
- ▶ Mano a mano che si va avanti con i layer il grado di astrazione delle feature estratte cresce
- ▶ L'apprendimento avviene tramite l'usuale backpropagation, non cambia nulla
- ▶ Le CNN tendono a diventare facilmente molto grosse (si arriva facilmente a 60 milioni di parametri) e usare tecniche come cross validation e grid search diventa molto problematico. Questo significa che è necessaria molta più esperienza per fare il tuning di tutti gli iperparametri a mano (Tutti gli iperparametri visti finora si applicano ugualmente anche alle CNN)
- ▶ Un altro metodo di regolarizzazione è il dropout ovvero un valore che indica la probabilità che una connessione venga interrotta
- ▶ Le CNN, numericamente, consistono per lo più in moltiplicazioni di matrici e tensori. Questo significa che il calcolo è facilmente parallelizzabile e può essere affidato alle GPU

# Deep Learning 10

*Il deep learning permette di creare modelli composti da numerosi layer in grado di imparare a rappresentare i dati con un alto livello di astrazione*

Si possono seguire diversi approcci per implementare metodi di deep learning. Tipicamente viene fatto costruendo architetture composte da molti layer di unità non lineare oppure creando strati di modelli di apprendimento supervisionato e non supervisionato per rappresentare ed estrarre meglio le feature dai dati.

Nel deep learning invece di concentrarsi manualmente sulla feature selection i modelli sono costruiti in modo tale da performare una estrazione gerarchica delle features (come nelle CNN). In questo modo i **modelli di deep learning sono in grado di aumentare il livello di astrazione andando avanti con i layer introducendo il concetto di apprendimento della rappresentazione dei dati.**



**Figura 10.1:** Prendendo, ad esempio, il caso di una cnn è possibile vedere come i primi layer estraggano feature che non sembrano interpretabili mentre andando avanti nei layer iniziano a distinguere contorni o addirittura oggetti

Le feature astratte possono essere usate nei layer più alti per combinare delle istanze e generarne altre durante il training (data augmentation in training) ad esempio aggiungendo o sottraendo gli occhiali alla foto di un volto.

In generale Nelle reti con molti layer sono necessari meno unità per layer (meno parametri e meno dati rischiesti per il training) MA avere molti layer rende la rete difficile da essere ottimizzata.

Altri problemi sono legati dal fatto che il gradiente si attenua andando indietro nei layer e la regolarizzazione diventa di difficile gestione a causa della grandezza della rete

## 10.1 Numero di Layer

Esempio: Odd Parity Boolean Function

Un circuito logico a due layer può rappresentare qualsiasi funzione booleana e qualsiasi funzione booleana può essere scritta come combinazioni di not, AND e OR.

Il problema è che con una rete a 2 layer la maggior parte delle funzioni booleane richiedono un numero di unità (gates AND/OR) che è esponenziale nella size dell'input.

Dati degli input di dimensione N consideriamo la funzione logica che torna 1 se un numero dispari di feature sono 1 e torna 0 altrimenti (**Odd parity boolean function**)

Nel caso N=2 si ha semplicemente lo XOR che come già visto è

$$A + B = (A \cdot \bar{B}) \text{ OR } (\bar{A} \cdot B)$$

Quindi per N=2 sono sufficienti 3 gates (2 and nell'hidden e un or nell'out).

Per N=3 la odd parity si scrive come

$$ABC \text{ OR } A\bar{B}\bar{C} \text{ OR } \bar{A}BC \text{ OR } \bar{A}\bar{B}\bar{C}$$

Per N=n con un unico layer servono  $2^{n-1} + 1$  gates

In alternativa è possibile creare un albero binario di XOR. Sono necessari in totale N-1 XOR che significa  $3(N-1)$  gates.

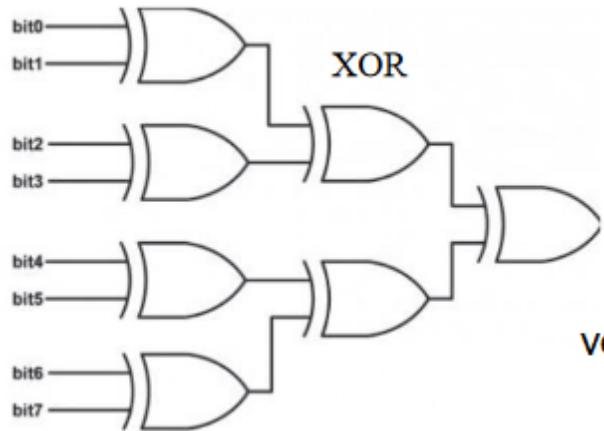
Aggiungendo layer il numero di unità da esponenziale è diventato polinomiale. (Per input di dimensione 8 siamo passati da 129 a 21 gates)

Bisogna comunque tenere conto del fatto che questa trattazione non vale per tutte le classi di funzioni e cercare di classificare le funzioni per cui vale questa proprietà (ovvero che l'aggiunta di layer fa passare il numero di unità richieste da esponenziale a polinomiale) è un campo di ricerca aperto

Il teorema di approssimazione universale è il fondamento teorico su cui si basano le NN, mostra che un hidden layer è sufficiente ma non dice nulla riguardo il numero di unità richieste. Questo significa che usare un unico layer potrebbe non essere la soluzione più efficiente.

Non esiste una teoria che ci dice quante unità siano necessarie in generale.

Un modello deep sfrutta la composizione per cambiare la rappresentazione dei dati mentre un modello a singolo layer è costretto a sfruttare forzatamente e unicamente la rappresentazione combinatoria esplicita delle feature.



**Figura 10.2:** Circuito logico multilayer che rappresenta la odd parity function per un input 8 dimensionale

Esistono tutta una serie di teoremi (no flattening theorem) che affermano che le funzioni ottenute tramite composizione in una DNN non sono ottenibili appiattendo la rete a un singolo layer mantenendo immutata l'efficienza (un esempio lo abbiamo visto con l'odd parity boolean function).

Una lista completa di questi teoremi (a oggi non completa) permetterebbe di sapere esattamente quando una rete deep sarebbe più efficiente di una con pochi layer

## Inductive bias

Non bisogna guardare alle reti deep come l'analogo di modelli shallow (pochi layer) molto grandi ma possono essere visti come modelli molto compatti (anche esponenzialmente compatti) di modelli shallow potenzialmente molto largi.

Questo significa che hanno molti meno parametri e sono più efficienti sia dal punto di vista computazionale sia dal punto di vista dell'apprendimento (migliore capacità di generalizzazione con meno dati di training)

Scegliere un modello deep significa assumere che la funzione di target sia facilmente scrivibile come somma di funzioni ottenute tramite la composizione di funzioni più semplici.

Se il nostro task matcha questo inductive bias sicuramente una rete deep risulterà molto più efficiente.

Alcuni esempi di task che rispettano questo bias sono tutti quelli legati alle immagini, al linguaggio e alla musica (per la loro stessa struttura intrinseca) MA non tutti i task rispettano necessariamente questo bias (anche se è un'assunzione abbastanza generale e genericamente rispettata)

## Curse of the dimensionality

Molti modelli si basano sul concetto di approssimazione locale (ovvero si fa l'assunzione che localmente i dati si somiglino tra loro).

Modelli che fanno questa assunzione sono ad esempio k-nn, decision tree, kernel

particolarmente strettii, etc.

Spesso però questa assunzione non basta e può risultare necessario un numero molto grande di esempi su cui eseguire il training per coprire sufficientemente lo spazio (la cui dimensione può essere enorme)

Quindi per diminuire il numero di dati necessari per effettuare il training è necessario fare delle assunzioni di base sulla distribuzione di probabilità che genera i dati (inductive bias) in modo da ottenere delle capacità di generalizzazione non locali

Si potrebbe risolvere il problema facendo delle assunzioni che riguardino in modo molto specifico il task che si vuole risolvere.

L'approccio del deep learning invece è scegliere un inductive bias molto generale che riguarda la composizione di funzioni:

**L'idea fondante del deep learning è l'assunzione che i dati siano generati da una composizione di fattori o di feature in livelli multipli secondo una specifica gerarchia**

Questo permette di ottenere un guadagno in termini di efficacia esponenziale rispetto al numero di esempi e di regioni che possono essere distinte ottenendo una capacità di generalizzazione non locale

## Manifold learning

Il manifold learning è un altro approccio per ridurre la dimensionalità di problemi non lineari.

I modelli basati su questo approccio sono basati sull'idea che la dimensionalità di molti dati è alte in modo "artificiale".

La tecnica più semplice per eseguire una riduzione di dimensionalità è fare una proiezione dei dati. La PCA è un altro esempio di modello appartenente alla classe del manifold learning.

Tipicamente appartengono a questa classe tutti gli algoritmi che richiedono di eseguire proiezioni o embedding in spazi di dimensionalità minore

## Recap

Non c'è una risposta generale alla domanda di quanti layer o unità vadano usate.

In generale una rete deep permette di usare meno unità in ogni layer diminuendo il numero di parametri e di dati richiesti per arrivare a una buona capacità di generalizzazione.

Il rovescio della medaglia è dato dal fatto che una rete deep può diventare molto difficile da ottimizzare e potrebbe non essere più possibile usare

tecniche di cross validation e grid search per la scelta degli iperparametri (a causa del loro alto numero) e vanno fissati "a mano"

## 10.2 Representation learning

Il representation learning è un insieme di metodi che permettono a una macchina di scoprire automaticamente da dei dati raw le loro rappresentazioni necessarie al fine della soluzione del task.

Il deep learning è un metodo di representation learning costruito su più livelli (l'abbiamo già visto con le CNN) ma il concetto è più astratto e applicabile in generale alle NN

Nel machine learning **una buona rappresentazione è una rappresentazione che facilita l'apprendimento.**

Il supervised learning è un esempio di come la scelta della rappresentazione avvenga in modo automatico invece che manuale (come, ad esempio, per la linear basis expansion dove siamo costretti a fissare manualmente le funzioni della base)

## PreTraining

Le reti deep soffrono del **Problema del gradiente evanescente** ovvero più un layer è lontano dall'output meno i pesi cambiano a ogni learning step. [4]

Un modo per risolvere questo problema è il **pretraining**: esso consiste nell'aggiungere progressivamente nuovi layer facendo il training solo per il nuovo layer mantenendo il resto dei pesi dei layer precedenti fissati.

Questo approccio, chiamato anche **Greedy layer-wise** scomponete il problema in più componenti e risolve i sottotask in modo ottimale e in isolamento. **Purtroppo l'unione delle soluzioni ottimali delle componenti potrebbe non corrispondere alla soluzione ottimale del problema completo.**

Il pretraining non supervisionato (tipicamente un autoencoder) può essere utile quando abbiamo un grande numero di esempi non labellati che possiamo usare per inizializzare i pesi della rete al fine di eseguire successivamente un task supervisionato su un numero molto minore di dati.

In conclusione il pretraining è molto utile per semplificare il processo di apprendimento, il processo di sviluppo e da usare come strategia di inizializzazione. In ogni caso potrebbe compromettere visibilmente le capacità di generalizzazione del modello finale

## Autoencoders

Un autoencoder è una rete neurale in cui i label sono gli input stessi.

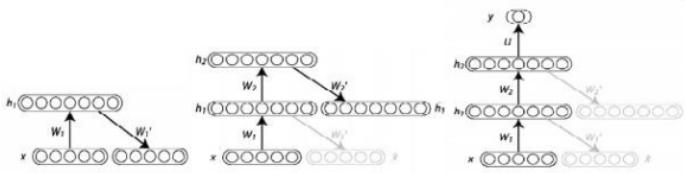
E' diviso in due parti: un encoder che trasforma gli input in un'altra rappresentazione (codifica) e un decoder che ritrasforma la codifica nei dati originali.

Esistono due forme di autoencoder:

- **Undercomplete:** L'hidden layer usato come output per l'encoder ha dimensione minore all'input. Questo forza la rete a estrarre le feature più rilevanti dell'input riducendo la dimensionalità
- **Overcomplete:** L'hidden layer usato come output per l'encoder ha dimensione maggiore dell'output. Questo rende la codifica più robusta al rumore. Tramite regolarizzazione è possibile ottenere delle codifiche sparse che facilitano il task di classificazione

Un modo per sfruttare gli autoencoders per fare il pretraining di una DNN è l'**algoritmo di Bengio (Layer-wise pretraining)**:

1. Traina il primo layer come un autoencoder per minimizzare l'errore di ricostruzione degli input grezzi (task unlabeled)
2. L'output layer dell'encoder è usato come input del layer successivo che viene trainato sempre come un autoencoder (non ho capito se ritraininga tutta la rete o lascia i pesi dei layer precedenti fissati)
3. Iterare a piacere
4. L'ultimo encoder sarà usato come input per una layer supervisionata
5. Fai il training della NN supervisionata (lasciando i pesi dei layer precedenti fissati)
6. Fai il tuning degli iperparametri della NN supervisionata e rifai il training di tutti i layer



**Figura 10.3:** Rappresentazione dell'algoritmo di Bengio

Questo algoritmo si basa sul fatto che gli autoencoder riescano a portare i parametri dei primi layer in una zona non molto lontana dal minimo della Loss.

In ogni caso il pretraining è stato un approccio molto usato (soprattutto in campi com NLP) ma al giorno d'oggi è stato abbastanza abbandonato (anche il fatto che l'apprendimento avvenga in fasi rende la gestione della rete e degli iperparametri abbastanza complicata)

## Transfer Learning

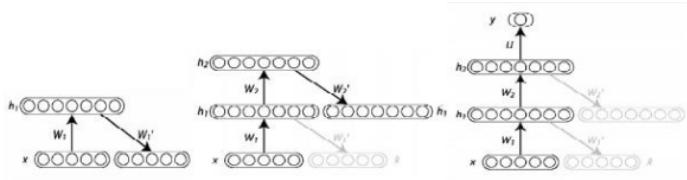
Il trasfer learning è un metodo per usare un modello (o parte di esso) come punto di partenza per un modello che deve risolvere un task simile al modello precedente.

Questo approccio si basa sull'assunzione che esistano feature che sono utili alla soluzioni di diversi task e che la loro estrazione dai dati avvenga in maniera indipendente dal task che si vuole risolvere

**Esempio:** Consiamo di aver costruito una CNN per un task A di classificazione di immagini. Se vogliamo risolvere un altro task di classificazione B possiamo prendere la parte convolutiva della CNN costruita per il task A, attaccarla a un'altra MLP e risolvere il task B

### 10.3 Distributed Representation

Nella rappresentazione distribuita le feature non sono mutualmente esclusive e le loro possibili configurazioni corrispondono a variazioni nei dati. [4]



**Figura 10.4:** Differenze tra rappresentazione simbolica (es. one hot encoding) e distribuita

La rappresentazione distribuita permette una rappresentazione più ricca tramite la condivisione di attributi che permettono la generalizzazione di concetti diversi.

In generale la rappresentazione è usata:

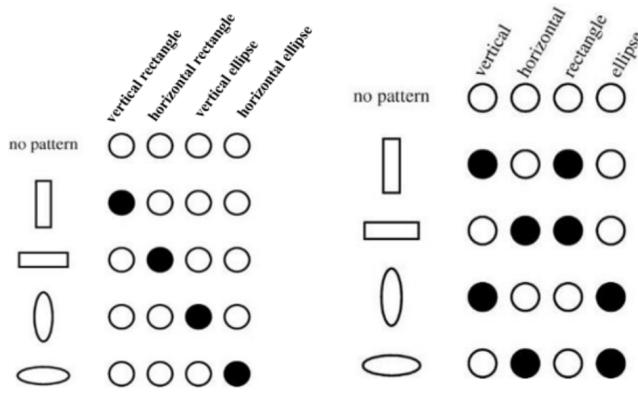
- **In input:** Se si hanno conoscenze a priori sui dati per fissarle
- **Automaticamente** nell'apprendimento, internamente al modello. Tipicamente il one hot encoding è usato per l'input e il modello apprende automaticamente la rappresentazione distribuita necessaria al task

Il one hot encoding necessita di una dimensione per ogni feature necessaria ed ha 2 tipi di problemi:

1. Non è possibile stabilire un grado di similitudine tra 2 feature. Non c'è nessuna informazione su come si relazionano tra loro le feature. Mentre in rappresentazione distribuita spesso elementi simili sono spazialmente vicini. (In NLP questo succede con parole semanticamente simili)
2. Poiché ogni dimensione rappresenta una feature unica questa rappresentazione è incapace di gestire features non viste o sconosciute

La rappresentazione distribuita, date n feature ciascuna con k valori, può RAPPRESENTARE  $k^n$  concetti, mentre il one hot solamente n.

Inoltre condividere attributi permette al modello di trattarli in modo simile e apprendere automaticamente la loro somiglianza.



**Figura 10.5:** Ad esempio se in questa situazione si presentasse un cerchio nel one hot encoding il modello non saprebbe come interpretarlo (poichè riconosce solo rettangoli o ellissi) mentre in rappresentazione distribuita potrebbe essere rappresentato con un ellisse AND verticale AND orizzontale

Il problema principale della rappresentazione distribuita è che rende l'interpretazione dei dati molto più difficile. Il DL sfrutta la rappresentazione distribuita attraverso molti layer ottenendo diversi livelli di astrazione tramite composizione o creando delle gerarchie.

## 10.4 Further topics

- ▶ Per un dato numero di parametri una rete deep è più smooth di una shallow. Questo perchè ogni layer lavora sulla superficie dell'output del layer precedente
- ▶ Le reti deep sembra che imparino meglio a parità del numero di unità. Questo è dato da vincoli di smoothness impliciti (invece di vincoli esplicativi come in altri modelli di classificazione)
- ▶ Le reti deep hanno una sorta di regolarizzazione implicita forzata dal gradient descent (massimizzazione del margine nel caso di classificazione). Il motivo di questa proprietà sta nel fatto che il gradient descent per reti deep intorno al minimo si comporta nello stesso modo che per un modello lineare. [7]
- Questo significa che metodi di regolarizzazione esplicita (es. Tikhonov) possono risultare anche non necessari per reti deep
- ▶ Nelle reti deep anche il gradient descent lineare standard tende a massimizzare il margine L<sub>2</sub> senza usare una regolarizzazione esplicita (**è un teorema**)
- ▶ Le reti deep sono **ridondanti** ovvero, anche se sono sovraparametrizzate, la maggior parte dei pesi (anche il 90%) sono ridondanti e possono essere ricavati dal resto dei pesi.  
E' possibile quindi comprimere i parametri senza perdere accuratezza.

Questo potrebbe far pensare che potrebbero bastare reti molto più piccole. Il problema è che in reti shallow le performance dipendono fortemente da una fortunata inizializzazione dei pesi della rete o di parti di essa. Una rete deep contiene un numero di sottoreti esponenzialmente più grande rispetto una rete shallow e quindi la probabilità di trovare un

inizializzazione dei pesi fortunata per qualche sottorete per le reti deep è molto più alta (**lottery ticket hypothesis**).

Si mostra anche che se si ricrea una rete shallow con dei pesi inizializzati come la sottorete ottimale della rete deep si ottengono le stesse performances

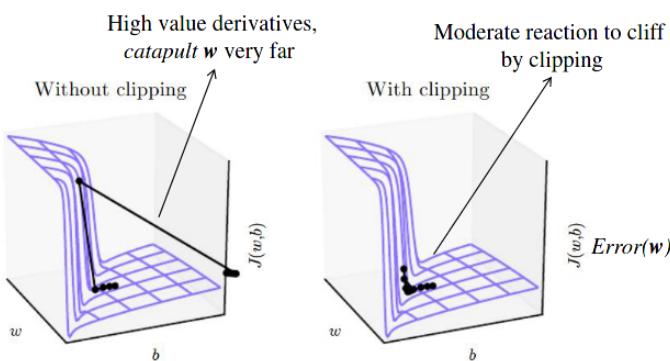
## 10.5 Tecniche

### Gradient Issues

Se i pesi sono piccoli il gradiente mentre si propaga all'indietro nei layer tende ad annullarsi mentre se i pesi sono grandi tende a crescere esponenzialmente.

Per ovviare al problema del gradiente che esplode si può usare la tecnica del **clipping gradient**.

$$\begin{cases} \mathbf{g} = \mathbf{g} & \text{if } |\mathbf{g}| < \nu \\ \mathbf{g} = \nu \frac{\mathbf{g}}{\|\mathbf{g}\|} & \text{if } |\mathbf{g}| > \nu \end{cases} \quad (10.1)$$



**Figura 10.6:** Il gradient clipping evita che, in caso di forti variazioni della loss, i pesi siano catapultati molto lontano dal loro punto di partenza

Il problema del **vanishing gradient** invece, è dato dal fatto che nel layer d (partendo a contare dall'output layer) il gradiente è la derivata di d funzione composta e il gradiente del layer d-esimo sarà dell'ordine  $w^d$ . Quindi se i pesi sono piccoli il gradiente scompare esponenzialmente all'aumentare dei layer.

Per risolvere questo problema esistono diversi approcci come RProp (resilient propagation), RNN (randomized NN) e l'uso della RELU come funzione di attivazione

### ReLU

$$f(x) = \theta(x)x$$

- L'attivazione è sparsa ovvero per un'inizializzazione dei pesi uniforme intorno allo zero il 50% delle unità sarà spento. (**NB** Per la ReLU Xavier non è l'inizializzazione ottimale, conviene usare la He inizialization: uniforme in  $\pm \frac{1}{\sqrt{n}}$  dove n è il numero di unità del layer)
- Per avere la maggior parte delle unità attive conviene inizializzare i bias a un valore piccolo positivo come 0.1. Questo permette inizialmente di far passare tutte le derivate
- Previene i problemi di exploding e vanishing gradient
- Non satura come la sigmoide (questo contribuisce a risolvere il problema del gradiente vanescente in quanto a saturazione il gradiente tende a 0)
- Efficiente dal punto di vista computazionale (lineare o nulla)
- Non differenziabile in zero ma non è un problema
- E' possibile usare versioni modificate della relu nel caso si volesse procedere nell'apprendimento anche per  $x < 0$  (es. Leaky ReLU, ELU,...)

## Batch Normalization

Il batch normalization è il metodo che normalizza ogni batch (portandolo a media nulla e varianza un) ad ogni layer. (La batch normalization può essere inclusa direttamente nella backpropagation)

Si osserva empiricamente che il batch normalization permette un apprendimento più rapido e aumenta l'accuratezza per modelli deep

## Dropout

Il dropout è il metodo per il quale si seleziona casualmente un sottoinsieme della rete durante la fase di training.

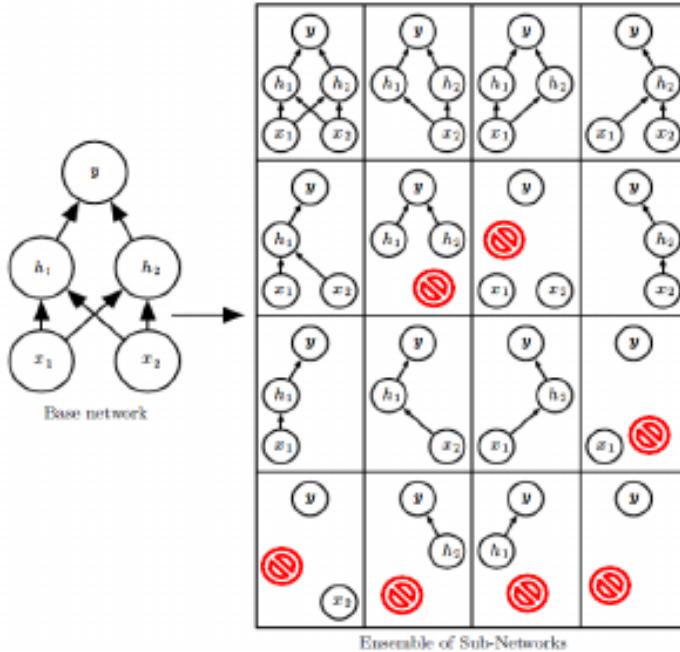
E' una sorta di ensembling: come il bagging corrisponde a combinare tra loro i risultati di più modelli ma nel caso del dropout non si ha un aumento del costo computazionale (mentre nel caso del bagging ogni modello va trainato a parte)

Il dropout è un metodo semplice per evitare l'overfitting quindi è un altro metodo per effettuare .

Il dropout ha l'obbiettivo di approssimare il processo di bagging ma con un numero di sottoreti esponenzialmente largo mentre, nella fase di predizione, abbiamo un'unica rete

## Algoritmo:

- Ogni volta che carichiamo un minibatch facciamo un sampling casuale di una mask binaria da applicare a tutte le unità della rete ad eccezione di quelle di output.



**Figura 10.7:** Ensembling di una rete con un solo hidden layer da 2 unità.  
La rete viene allenata ma a ogni step di training esiste una probabilità (altro iperparametro) che una unità in un hidden layer venga rimossa.  
La probabilità di ottenere reti non funzionanti per reti molto grandi è infinitesima

- ▶ Il valore booleano associato a ogni unità è indipendente e la probabilità che l'unità venga rimossa rappresenta un altro iperparametro
- ▶ Una volta rimosse le unità dalla rete si esegue il feed forwarding e la back propagation. Questo rappresenta il learning step della sola sottorete ottenuta da un campionamento casuale
- ▶ Le unità rimosse si reinseriscono nella rete con i pesi che avevano precedentemente e il processo viene reiterato

#### Insights:

- ▶ Il dropout, evitando che tutta la rete sia allenata e riducendo il numero di interazioni tra le unità, ha un effetto di regolarizzazione
- ▶ Come nel caso del bagging il dropout **riduce la varianza**
- ▶ Inserisce un rumore strutturale durante il processo di apprendimento (es. Nel caso delle immagini il dropout potrebbe far arrivare a un layer un'immagine lievemente deformata di una parte di un volto). Questo permette di estrarre in modo efficiente una feature in più contesti diversi
- ▶ Si può dimostrare che nel caso di regressione lineare il dropout equivale alla regolarizzazione di tikhonov con un *lambda* diverso per ogni input
- ▶ Può essere usato per ogni modello che sfrutta la rappresentazione distribuita e la discesa stocastica del gradiente

#### L<sub>1</sub> regularization

Il termine di regolarizzazione  $L^1$  nella Loss  $+\lambda\|\mathbf{w}\|$  funziona come un meccanismo di eliminazione delle feature producendo alla fine un modello più semplice

La soluzione può essere approssimata ma spesso richiede tecniche di ottimizzazione non differenziabili (es. smooth approximation of L<sub>1</sub> norm)

# Randomized Machine Learning 11

La casualità può essere vista come un approccio al Machine Learning in quanto può costituire un fattore chiave per la costruzione di modelli efficienti.

La casualità può essere sfruttata sia per migliorare le performance di predizione che per mitigare diverse problematiche legate al modello scelto, ad esempio:

- ▶ **Data splitting:** Come già visto esistono diverse tecniche di splitting del dataset (Hold-out, k-fold,...) che sfruttano la casualità.
- ▶ **Data generation** (es. Bootstrap, Boosting, ...)
- ▶ **Learning:** Tecniche come il minibatch o l'SGD
- ▶ **Selezione di iperparametri** come ad esempio la random search (si provano gli iperparametri generati da distribuzioni di probabilità prestabilite)
- ▶ **Regolarizzazione** (es. Dropout)
- ▶ La casualità può stare anche alla base deò funzionamento di un modello, si vedano le boltzmann machine o il random forest

## 11.1 Randomized Neural Network

L'idea di base è quella di inizializzare la rete con pesi fissati casualmente lasciando tutti i pesi fissati tranne quello dell'output layer con funzione di attivazione lineare (quindi semplice regressione lineare sull'output della rete casuale)  
Questo approccio riduce enormemente l'effort computazionale rispetto una usuale NN delle stesse dimensioni

La rete casuale mappa i dati in uno spazio di dimensione maggiore dove il problema potrebbe diventare linearmente separabile (questa probabilità cresce con la dimensionalità del codominio). (E' possibile creare questa mappa anche tramite linear basis expansion)

### Teorema 11.1. Teorema di ricoprimento:

*Un problema di classificazione qualsiasi embeddato in uno spazio di dimensionalità maggiore tramite una mappa non lineare ha una probabilità di essere linearmente separabile nel codominio crescente con la dimensionalità del codominio stesso. (In particolare la mappa che manda i punti nei vertici di un triangolo  $l-1$  dimensionale rende ogni partizione binaria dei punti linearmente separabile)*

Gli hidden layer di random NN su comportano come una **linear basis expansion randomizzata**.

**Osservazione 11.2.** Poichè lo spazio su cui si fa la regressione lineare ha dimensionalità molto alta è molto facile cadere in overfitting quindi la regolarizzazione per le random NN è molto importante

Una random forest è un insieme di alberi decisionali i cui nodi sono stati costruiti, per ogni albero indipendentemente, in modo casuale campionando casualmente una parte dei dati di training.  
Alla fine il risultato sarà quello che avrà ricevuto più "voti" all'interno dell'ensemble (bagging)

Esistono vari modelli di Random NN (es. extreme learning machine (ELM), radial basis function con centro casuale (RBF), random vector functional link (RVFL))

## 11.2 Conclusioni

- ▶ Le random NN con un alto numero di parametri **FISSATI** riescono a rendere problemi molto complessi linearmente separabili
- ▶ Sono estremamente efficienti dal punto di vista computazionale (al pari della regressione lineare)
- ▶ Un gran numero di unità casuali provvedono a fornire una linear basis expansion sufficiente a risolvere il problema (basato sul teorema di approssimazione universale)
- ▶ Esistono delle varianti che possono aiutare a trovare la rappresentazione migliore ad esempio aggiungendo unità (come in cascade correlation) o eliminandole
- ▶ sono molto utili in contesti di big data e sistemi embedded (per la loro efficienza) e in generale in tutti quei contesti in cui risparmiare costo computazionale è importante

# Unsupervised Learning 12

Consideriamo il caso in cui il nostro training set non abbia dei label (cosa molto comune). Vogliamo comunque trovare un modo per classificare i dati

- **Clustering:** Consiste nel trovare raggruppamenti naturali dei dati (clustering)
- **Riduzione dimensionale** Tecniche come la PCA sono utili per estrarre le informazioni rilevanti dal dataset. Sono molto utili anche per fare preprocessing in vista di un successivo apprendimento supervisionato

## 12.1 Clustering

Assumiamo, come per knn, che la similità dei dati sia legata alla loro distanza. Questo significa che la metrica scelta nello spazio dei dati assume un ruolo cruciale (spesso viene usata quella euclidea ma possono esistere scelte migliori).

### Vector quantization and K mean

#### Vector Quantization

**Def. 12.1.** *Vector quantization* Dato un set di dati  $x$  e un set finito di vettori di riferimento (codebook vectors)  $\mathbf{w} = \{\mathbf{w}_1, \dots, \mathbf{w}_k\} \in \mathbb{R}^n$  definiamo gli insiemi

$$V_i = \{x \in V \text{ t.c. } \|x - \mathbf{w}_i\| \leq \|x - \mathbf{w}_j\| \forall j\}$$

Ogni data  $x$  sarà associato al vettore di riferimento più vicino.

Con questa procedura si va a creare il *Poliedro di Voronoi* che divide lo spazio dei dati in  $k$  partizioni

L'obiettivo è trovare una partizione ottimale di una distribuzione di dati sconosciuta all'interno di cluster definiti da un prototipo (i vettori di riferimento).

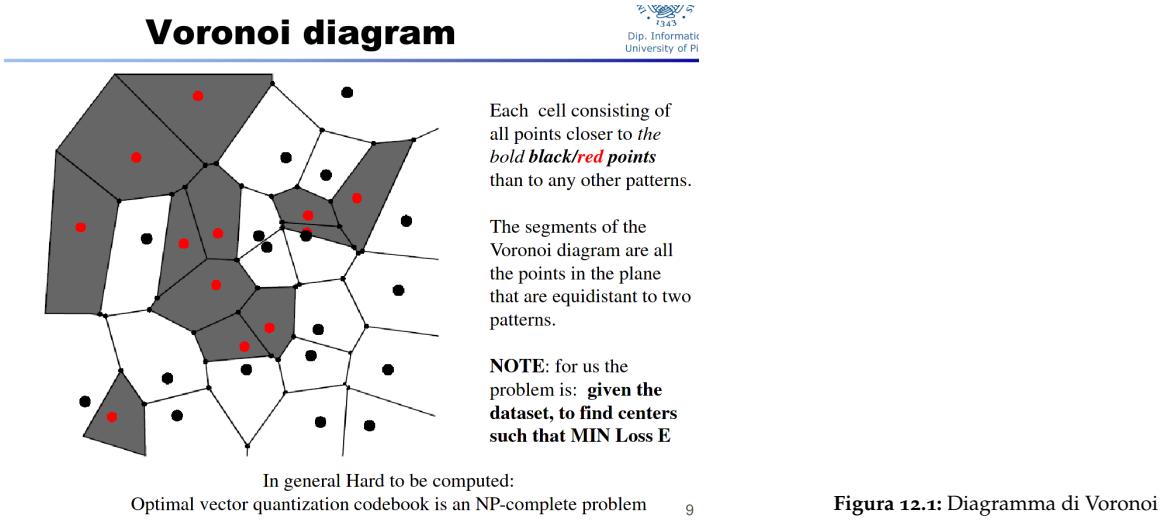
Per risolvere questo task definiamo la loss

$$E = \int \|x - \mathbf{w}_{i(x)}\|^2 p(x) dx$$

dove  $p(x)$  è la pdf dei dati e  $\mathbf{w}_{i(x)}$  è il vettore prototipo associato al dato  $x$ .

Poichè il numero di punti è discreto la loss discretizzata è

$$E = \sum_i^l \sum_j^K \|x_i - \mathbf{w}_j\| \delta_{ij}^{winner}$$



dove  $\delta_{ij}^{winner}$  è 1 solo se  $w_j$  è il vettore prototipo associato a  $x_i$ , o altrimenti.

Il set di vettori  $\mathbf{w}$  che minimizza  $E$  è la soluzione al problema della vector quantization

**Online K means** Calcolando  $\partial_{w_j} E$  otteniamo le regole di apprendimento per la vector quantization.

La versione **online** per ogni  $x$  è

$$\Delta w_j = \eta \delta_{ij} (x_i - w_j)$$

Si noti che solo i prototipi associati al punto  $x_i$  cambiano e si adattano a esso. Questo algoritmo dipende fortemente dall'inizializzazione dei vettori prototipi e il loro numero è un parametro che va fornito dall'utente

### K means batch algorithm

1. Fissa  $k$  vettori prototipo scegliendo casualmente  $k$  punti tra i dati a disposizione
2. Assegna ogni punto al suo vettore prototipo più vicino  $j(x) = \operatorname{argmin}_i \|x - w_i\|^2$  ( $j$  indice del cluster a cui appartiene  $x$ )
3. Ricalcola il centro del cluster facendo una media spaziale di tutti i membri in un cluster, quindi  $w_i = \frac{1}{\#K_i} \sum_{j \in K_i} x_j$  dove  $K_i$  è il cluster  $i$ -esimo
4. Ripeti 2) e 3) fino al raggiungimento di un criterio di convergenza (es. variazione nella loss o dati che non cambiano più cluster)

**Osservazione 12.2.** *K means è un algoritmo efficiente che però dipende fortemente dall'inizializzazione dei  $w$  e dalla scelta del numero di cluster e non permette di proiettare i dati in spazi di dimensione minore.*

*Inoltre, poiché basato sulla metrica euclidea, è in grado di distinguere solo cluster ipersferici (sarebbe più corretto dire, in generale, insiemi convessi???)*

Per evitare di cadere in minimi locali è possibile usare la **regola di soft max** ovvero: per ogni  $x_i$  non viene modificato solo il vettore prototipo a lui associato ma vengono modificati tutti i vettori pesando al modifica con la distanza tra il punto  $x$  e un dato vettore prototipo (ad esempio paesando la modifica con la gaussiana nella distanza, rendendo minori le modifiche per i  $w$  più lontani da  $x$ )

## Self organizing map: SOM

Consiste in  $N$  neuroni disposti su un reticolo 2D. Ogni unità riceve lo stesso input  $x$ , ha un peso  $w$  e può essere identificata dalla sua posizione nella griglia. L'unità con il vettore peso  $w$  più vicino al vettore di input è "la vincitrice", ovvero l'unità a cui è associato il punto  $x$ . Per questo si definisce una SOM come una **mappa topologicamente ordinata**

In questo modo punti spazialmente vicini vengono mappati in unità vicine nel reticolo della SOM.

Questo modello è un primo esempio di una classe di modelli che sfruttano il concetto di **competitive learning**: un processo adattivo in cui le unità diventano gradualmente più sensibili e specializzate a diverse categorie di input tramite la competizione (Il vincitore ha diritto a imparare di più).

**Algoritmo di apprendimento** Ricapitolando ogni unità può essere identificata con la sua posizione nella mappa, riceve lo stesso input  $x$  e ha un peso  $w$ . Inoltre  $\text{Dim}(w) = \text{dim}(x) = n$

1. I pesi della mappa sono inizializzati casualmente
2. **Competitive stage** Per un dato  $x$  l'unità vincitrice è quella con il peso  $w$  più simile all'input  $x$ :  $j(x) = \arg\min_i \|x - w_i\|$  dove  $i$  è l'indice dell'unità nella mappa (2D)
3. **Cooperative stage**: Aggiorna i pesi delle unità che hanno una relazione topologia con il vincitore (praticamente le unità vicine oppure tramite la regola di soft max)

Il peso dell'unità vincitrice e i suoi vicini sono modificati in modo tale che si avvicinino all'input (**Hebbian learning**)

All'iterazione  $t$  si ha

$$w_i^{(t+1)} = w_i^{(t)} + \eta(t) h(t)_{i,j(x)} (x - w_i^{(t)})$$

dove  $h(t)_{i,j(x)} = \exp\left(-\frac{\|r_i - r_j\|^2}{2\sigma_{nh}^2(t)}\right)$  è la neighborhood function .

Il learning rate e la largezza della neighborhood function  $\sigma$  vengono diminuite all'aumentare delle iterazioni.

All'inizio la larghezza è sufficiente a coprire un certo numero di unità vicine per conferire al modello un ordinamento globale e poi viene diminuita per raggiungere una convergenza

4. Itera fino a convergenza

E' importante notare che la regola di aggiornamento pesi della fase cooperativa è fondamentale per la formazione di una mappa topologicamente ordinata infatti i pesi sono modificati in sottoinsiemi topologicamente collegati.

Un vantaggio delle SOM è che sono facilmente interpretabili

# Recurrent Neural Networks 13

Una RNN è una rete che include connessioni tra unità appartenenti allo stesso layer o a se stessi (**feedback loops**: l'output di un unità è collegato all'input della stessa).

La presenza di self loops fornisce l'unità di una **memoria** di ciò che è stato calcolato in precedenza.

Questo tipo di architettura è necessaria quando i dati sono **strutturati** e non sono semplici vettori composti da feature indipendenti e quando l'output dipende dalla storia pregressa. Alcuni esempi sono: Sequenze, segnali, processi dinamici, testi (si pensi alla traduzione), alberi, grafi, strutture complesse (come molecole o proteine), etc.

Le RNN possono risolvere due tipi di **task**:

- ▶ **Classificazione/Sequence recognition:** Reti con un singolo output alla fine
- ▶ **Sequence transduction:** Si ha un output per ogni input step. La trasduzione nel dominio sequenziale equivale a mappare la sequenza di input in una sequenza di output

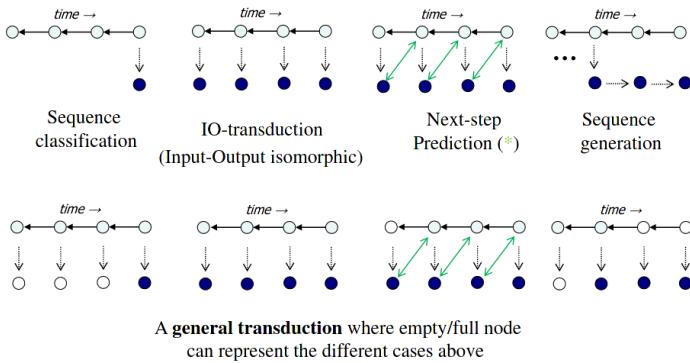


Figura 13.1: Diversi tipi di trasduzione

## 13.1 Input delay neural network: IDNN

L'obiettivo è creare una rete in cui l'output dipenda dall'input precedente. Un semplice modo per fare questo è far scorrere sulla sequenza temporale una sliding window. Il problema è che il numero di pesi dipende dalla size della finestra che, tra l'altro, va fissata a mano.

E' essenzialmente come una CNN

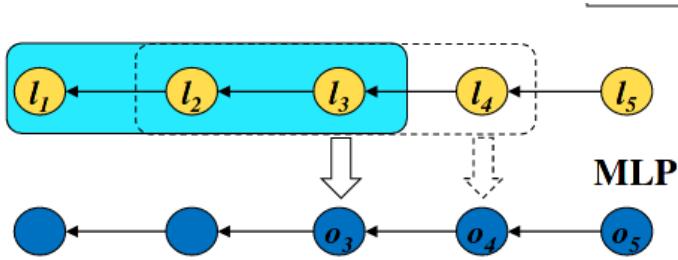


Figura 13.2: Input delay neural network

## 13.2 RNN unit

Una recurrent unit usa l'input corrente più una informazione di stato. In questo modo l'output dell'unità è

$$o(t) = f(wx(t) + b - \hat{w}o(t-1))$$

dove  $\hat{w}$  è il peso associato alla parte ricorrente dell'unità e  $o(t-1)$  è l'output dell'unità al passo  $t-1$ . In questo modo lo stato interno riassume l'informazione passata e il suo peso associato è un parametro libero del modello

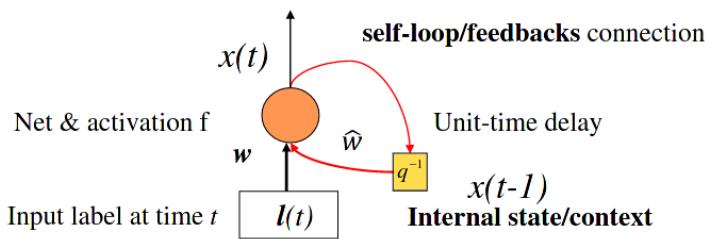


Figura 13.3: RNN unit

Nel formare una rete intera è possibile realizzare molte architetture diverse, la più comune è quella in figura.

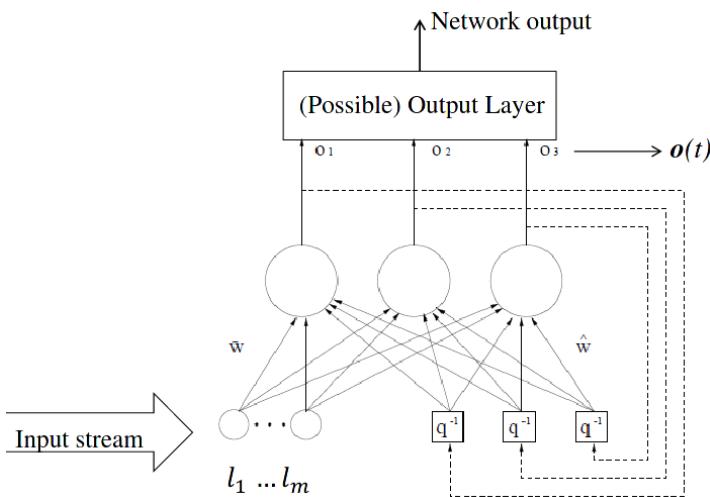


Figura 13.4: Full recurrent neural network

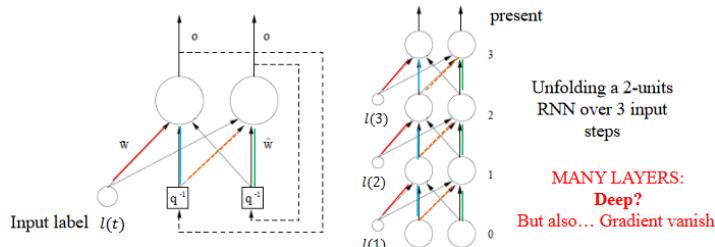
Le RNN sono **Approssimatori universali di sistemi dinamici non lineari** e possono simulare qualsiasi automa.

Esse sono basate sulle **assunzioni** di:

- **Causalità:** L'output al tempo  $t_0$  dipende dagli input a tempi  $t \leq t_0$  (implementata con lo stato interno)
- **Stazionarietà** ovvero invarianza temporale. Sia  $o(t) = \tau(x(t), o(t-1))$ , la stazionarietà implica che la funzione  $\tau$  sia la stessa a qualsiasi tempo  $t$  (ovvero per qualsiasi nodo della sequenza)
- **Adattività:** La funzione  $\tau$  viene appresa dalla rete in modo autonomo tramite la modifica dei pesi

### 13.3 Apprendimento

L'**Unfolding** di un modello è l'evoluzione del modello nel tempo. E' possibile costruire una MLP (encoding neural network) è equivalente a una RNN data una specifica sequenza in input.



**Figura 13.5:** Unfolding di una RNN. Si noti che, per il requisito di stazionarietà, alcuni pesi sono condivisi nella rete

Sulla encoding network è possibile applicare le regole di backpropagation che già conosciamo ma la encoding network può essere molto profonda e quindi può presentarsi il problema del gradiente vanescente.

Alcuni degli algoritmi di apprendimento usati sono

- Backpropagation through time
- Real time current learning

Questi calcolano, in modo diverso, il valore del gradiente dell'errore sulla rete unfolded equivalente. Il problema del gradiente vanescente rende difficile imparare relazioni a grande tra elementi nella sequenza a grande distanza temporale.

Alcune varianti della semplice unità che abbiamo visto sono le LSTM (non adatte se necessaria memoria molto corta), GRU (LSTM semplificata), BRNN (bidirezionali).

Per gestire il problema del gradiente vanescente vengono usate sia tecniche di pretraining che ottimizzatori hessian free

L'approccio visto nelle RNN può essere generalizzato a strutture più complicate come alberi ottenendo le **Recursive neural network**

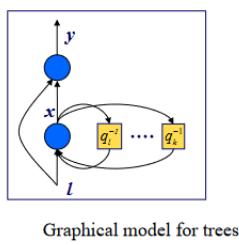
### 13.4 Echo State Network

Le ESN sono un paradigma per modellare le RNN in modo efficiente. Una ESN consiste in:

- Un ampio serbatorio di unità ricorrenti connesse in modo sparso e non allenate (solo inizializzate casualmente)
- Un semplice layer di output feedforward con funzione di attivazione lineare

Questo tipo di modello permette di ottenere una particolare stabilità del sistema dinamico.

Inoltre lo stato ricorrente dipende asintoticamente solo dalla storia degli input e questo permette di discriminare tra diversi input basandosi sul suffisso (input con lo stesso suffisso vengono mappati in punti vicini nello spazio degli stati) senza dover fare il training delle unità ricorrenti



Graphical model for trees

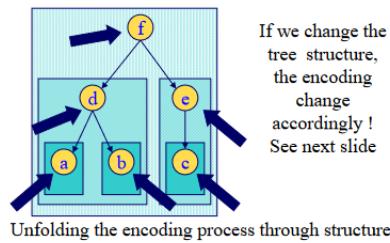


Figura 13.6: Recursive neural network

# Bibliografia

- [1] [https://en.wikipedia.org/wiki/Moore%20%80%93Penrose\\_inverse](https://en.wikipedia.org/wiki/Moore%20%80%93Penrose_inverse).
- [2] <https://towardsdatascience.com/understanding-singular-value-decomposition-and-its-application-in-data-science-388a54be95d>.
- [3] Scott E. Fahlman and Christian Lebiere. *The Cascade-Correlation Learning Architecture*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [5] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.
- [6] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [7] Tomaso Poggio, Kenji Kawaguchi, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Xavier Boix, Jack Hidary, and Hrushikesh Mhaskar. Theory of deep learning iii: explaining the non-overfitting puzzle, 2018.