

Advanced Machine Learning

Giuseppe Magazzù

2021 - 2022

Contents

1	Introduction	1
1.1	Non Linearity	1
1.2	Feed Forward Neural Network	3
2	Backpropagation	4
3	Gradient Based Optimization	5
3.1	Gradient Descent	5
3.2	Training as Optimization	6
3.3	Stochastic, Batch, Mini-Batch	7
3.4	Empirical Risk	8
3.5	Ill Conditioning	8
3.6	Local Minima	8
3.7	Altri Algoritmi	8
4	Cost Functions	9
4.1	Loss Functions	9
4.2	Output Units	10
4.2.1	Linear - Distribuzione Gaussiana	10
4.2.2	Sigmoid - Distribuzione Bernoulli	10
4.2.3	Softmax - Distribuzione Multinoulli	11
4.2.4	Gaussian Mixtures	11
5	Activation Functions	12
5.1	Sigmoid and Hyperbolic Tangent	12
5.2	ReLU	13
6	Variance-Bias Trade-off	15
7	Regularization	17
7.1	Norm Penalties	17
7.2	Data Augmentation	18
7.3	Label Smoothing	19

7.4	Early Stopping	19
7.5	Parameter Tying and Sharing	19
7.6	Multitask Learning	19
7.7	Bagging	20
7.8	Dropout	20

Chapter 1

Introduction

1.1 Non Linearity

Per estendere i modelli lineari a funzioni non lineari possiamo applicare una trasformazione non lineare $\phi(x)$ all'input.

La funzione $\phi(x)$ definisce una nuova rappresentazione di x .

La funzione $\phi(x)$ può essere generica come nelle *kernel machines* oppure può essere imparata aggiornando i parametri θ .

$$f(x; \theta; \omega) = \phi(x; \theta)^T \omega ,$$

dove θ sono i parametri e ω i pesi del modello.

Una rete con 1 hidden layer può imparare una qualsiasi funzione $f(x)$ non lineare. La difficoltà consiste nel trovare i pesi per determinare $f(x)$.

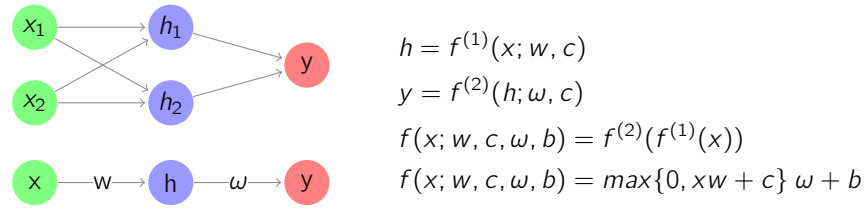
XOR Example

XOR function: $y = f^*(x)$

$$X = \{[0, 0]^T, [0, 1]^T, [1, 0]^T, [1, 1]^T\}$$

$$y = f(x; \theta) \Rightarrow f^*(x)$$

Supponiamo di scegliere un mapping lineare $f(x; \theta) = f(x; \omega, b) = x^T \omega + b$



Supponiamo di inizializzare i parametri nel seguente modo:

$$w = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad \omega = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\max(0, xw + c) \omega = [0, 1, 1, 0]^T$$

Multi-Layer Neural Networks:

- Feed Forward Neural Network (FFNN)
non hanno connessioni che formano loop
- Recurrent Neural Network (RNN)
hanno loop, utili per informazioni sequenziali
- Convolutional Neural Network (CNN)
catturano informazioni spaziali attraverso molteplici filtri

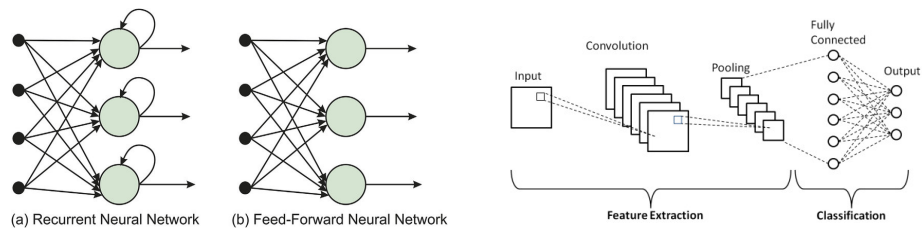


Figure 1.1: Differenza tra RNN, FFNN [2] e CNN [7]

1.2 Feed Forward Neural Network

Una FFNN è una rete fully connected, ovvero che ogni neurone in un layer è collegato a tutti gli altri del layer successivo. Una FFNN può essere pensata come una concatenazione di funzioni applicate all'input x , $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$.

La lunghezza della catena corrisponde alla **depth** del modello. La dimensionalità degli hidden layer determina la **width** del modello.

Il training del modello consiste nel trovare una funzione $f(x)$ che si avvicini il più possibile a una funzione target $f^*(x)$. $f(x) \rightarrow f^*(x)$.

Chapter 2

Backpropagation

Chapter 3

Gradient Based Optimization

L'ottimizzazione è il processo di minimizzare o massimizzare una funzione $f(x)$ alterando il valore di x . La funzione che vogliamo ottimizzare si chiama funzione obiettivo. La soluzione ottima viene denotata $x^* = \arg \text{opt } f(x)$, $\text{opt} = \{\min, \max\}$.

Possiamo trovare i punti di massimo e minimo analiticamente ponendo il gradiente della funzione pari a zero $\nabla_x f(x^*) = 0$.

Nel caso di modelli lineari possiamo usare l'ottimizzazione convessa, mentre per quelli non lineari è necessario usare una procedura iterativa di ottimizzazione numerica che trovano solo una approssimazione.

3.1 Gradient Descent

Il gradient descent (discesa del gradiente) è un algoritmo di ottimizzazione iterativo per trovare il minimo di una funzione differenziabile $f(x)$.

Ad ogni iterazione ci si muove da un punto iniziale x_i nella direzione opposta a quella di massima crescita della funzione, ovvero $-\nabla_x f(x_i)$.

$$x_{i+1} = x_i - \eta \nabla_x f(x_i)$$

Il parametro η controlla l'intensità dello spostamento (Figura 3.1). Per valori molto piccoli la convergenza sarà lenta, mentre per valori troppo grandi si rischia di arrivare in un ottimo locale sub ottimale.

L'algoritmo si ferma in base a un criterio specificato ad esempio quando lo spostamento diventa molto piccolo o dopo un numero predefinito di iterazioni.

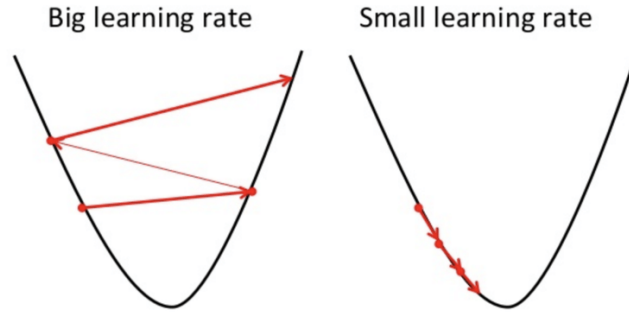


Figure 3.1: Comportamento del gradient descent con valori alti e bassi di η [6].

3.2 Training as Optimization

L'obiettivo di un algoritmo di training è quello di ritornare una funzione $f(x)$ che mappa con una certa accuratezza gli input x alle etichette corrispondenti y .

Per valutare l'accuratezza della funzione $f(x)$ introduciamo il concetto di loss function.

La funzione di loss $L(y, \hat{y})$ assegna uno score numerico (scalare) all'output predetto \hat{y} dato il valore di verità atteso y .

I parametri della funzione $f(x; \theta)$ sono scelti in modo da minimizzare la loss L sugli esempi di training.

Dato un training set etichettato $(x_{1:n}, y_{1:n})$, una funzione di loss L e una funzione parametrica $f(x; \theta)$, denotiamo la **cost function** come segue:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, y_i; \theta)$$

L'obiettivo dell'algoritmo di training è quindi impostare i parametri θ in modo che il valore di $J(\theta)$ sia minimizzato.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} J(\theta)$$

3.3 Stochastic, Batch, Mini-Batch

La cost function viene spesso decomposta come somma delle loss calcolate sui singoli esempi.

$$J(\theta) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(x, y; \theta) = \frac{1}{n} \sum_{i=1}^n L(x_i, y_i; \theta)$$

Il gradiente della loss $J(\theta)$ può essere calcolato come media dei gradienti delle singole loss.

$$\nabla_{\theta} J(\theta) = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L(x_i, y_i; \theta)$$

Tuttavia calcolare il gradiente per ogni esempio può essere costoso per n grandi, in quanto il costo è lineare nel numero di esempi.

Possiamo quindi usare un insieme di m esempi $B = \{x^1, x^2, \dots, x^m\}$ che viene detto **minibatch**.

$$g = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(x_i, y_i; \theta)$$

Gli algoritmi di gradient descent prendono un nome diverso in base a quanti esempi vengono utilizzati per calcolare il gradiente della funzione di costo (Figura 3.2).

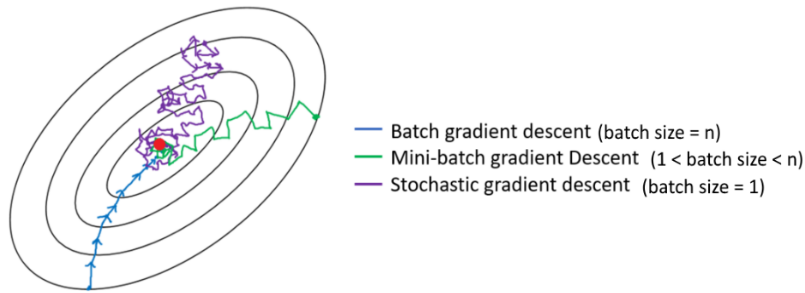


Figure 3.2: Rappresentazione del gradient descent con diverse batch size [4].

Con un numero di batch grande si ottiene una stima migliore, mentre con batch piccole una convergenza veloce. Solitamente le dimensioni scelte per la minibatch sono potenze di 2 (8, 16, 32, 64, ...) per sfruttare il calcolo parallelo su GPU.

Il numero di aggiornamenti per arrivare alla convergenza aumenta con in numero di esempi. L'aggiornamento del modello non dipende dal numero di esempi.

3.4 Empirical Risk

3.5 Ill Conditioning

3.6 Local Minima

3.7 Altri Algoritmi

Chapter 4

Cost Functions

Per valutare l'efficacia e le performance di un modello di deep learning usiamo una **cost function**.

Si usa il termine **loss function** o **error function** quando ci si riferisce un singolo esempio del training set, mentre **cost function** sull'intero training set (o mini-batch).

La **cost function** misura l'errore tra il valore predetto dal modello e il valore di verità. L'obiettivo è quello di minimizzare o massimizzare questa funzione in modo da ridurre l'errore.

La **cost function** può contenere anche un termine di regolarizzazione.

4.1 Loss Functions

Sia $y = (y_1, y_2, \dots, y_k)$ un vettore che rappresenta la distribuzione multinomiale di verità definito sulle etichette $1 \dots k$.

Sia $\hat{y} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_k)$ il vettore delle predizioni, dove $\hat{y}_i = P(y = i|x, \theta)$.

Negative Log Likelihood

$$J(\theta) = -\mathbb{E}_{x, y \sim \hat{P}_{data}} \log(P_{model}(y|x))$$

$$J(\theta) = L_{neg_likelihood}(y, \hat{y}) = -\sum_{i=1}^k y_i \log(\hat{y}_i)$$

Mean Squared Error (MSE)

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^N (y - \hat{y}_i)^2$$

Mean Absolute Error (MAE)

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^N |y - \hat{y}_i|$$

4.2 Output Units

La scelta della funzione di loss è legata all'unità di output.

4.2.1 Linear - Distribuzione Gaussiana

Spesso usato per ottenere la media di una distribuzione gaussiana condizionale.

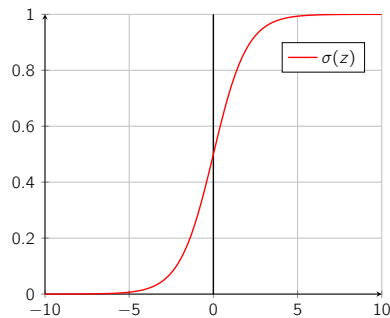
$$\hat{y} = w^T h + b$$

4.2.2 Sigmoid - Distribuzione Bernoulli

Usata per predire il valore di una variabile binaria $\hat{y} \in [0, 1]$.

La distribuzione di output è una distribuzione di Bernoulli definita da $P(y = 1|x)$.

La funzione sigmoide [4.1] ci permette di avere un gradiente forte quando abbiamo una predizione errata.



$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (4.1)$$

Per prima cosa calcoliamo l'argomento $z = w^T h + b$ e poi applichiamo la sigmoide per trovare $\hat{y} = \sigma(w^T h + b)$.

$$J(\theta) = -\mathbb{E}[\log P(y|x)]$$

$$\log(\tilde{P}(y)) = yz \quad \tilde{P}(y) = \exp(yz)$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^1 \exp(y'z)} = \frac{\exp(yz)}{1 + \exp(z)} = \sigma((2y - 1)z)$$

$$P(y = 0) = \sigma((2 * 0 - 1)z) = \sigma(-z)$$

$$P(y = 1) = \sigma((2 * 1 - 1)z) = \sigma(z)$$

$$J(\theta) = -\mathbb{E}[\log P(y|x)] = -\log \sigma((2y - 1)z)$$

4.2.3 Softmax - Distribuzione Multinoulli

Vogliamo rappresentare una distribuzione di probabilità \hat{y} definita su una variabile discreta con n valori possibili.

$$\hat{y} = P(y|x), \quad \hat{y}_i = P(y = i|x), \quad i = 1..n$$

$$z = w^T h + b \quad z_i = \log(\tilde{P}(y = i|x))$$

$$\text{Softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}$$

Calcolando il logaritmo della softmax possiamo riscriverla nel seguente modo:

$$\begin{aligned} \log \text{Softmax}(z)_i &= \log(\exp(z_i)) - \log \sum_{j=1}^n \exp(z_j) \\ &= z_i - \log \sum_{j=1}^n \exp(z_j) \end{aligned}$$

Quando massimizziamo, a valori alti del primo termine corrispondono valori bassi del secondo. Quindi possiamo tenere in considerazione solo il $\max_j z_j$.

Le altre funzioni di loss che non invertano l'esponenziale possono dare problemi di saturazione. Quindi è stata definita una versione più stabile

$$\text{Softmax}(z) = \text{Softmax}(z - \max_i z_i)$$

4.2.4 Gaussian Mixtures

<https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95>

<https://scikit-learn.org/stable/modules/mixture.html>

Chapter 5

Activation Functions

La maggior parte delle hidden unit consiste di una trasformazione affine $wx + b$ dell'input x a cui viene applicata una funzione $g(wx + b)$ chiamata funzione di attivazione.

Le funzioni di attivazione non lineari permettono al modello di apprendere una struttura più complessa nei dati.

Solitamente si utilizza la stessa activation function in tutti gli hidden layer.

5.1 Sigmoid and Hyperbolic Tangent

Entrambe le funzioni sono differenziabili e monotoniche, mentre la loro derivata non è monotonica.

La sigmoide è specialmente usata nell'output unit se l'output voluto è una probabilità poiché la sigmoide ha valori tra 0 e 1.

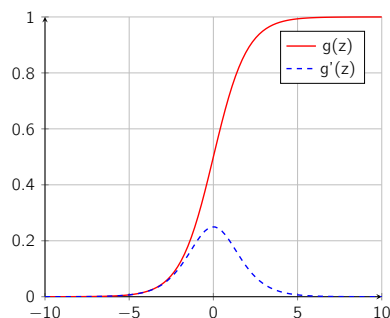


Figure 5.1: Sigmoide

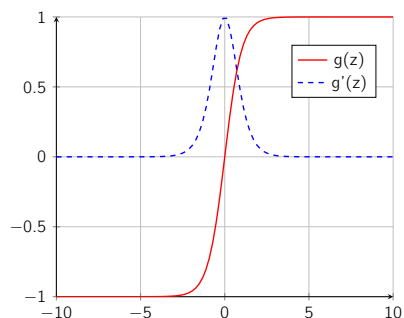


Figure 5.2: Tangente iperbolica

Il vantaggio della tangente iperbolica (\tanh) è che i valori vengono mappati tra -1 e 1. Quindi i valori negativi in valori fortemente negativi, mentre i valori vicino a zero vengono mappati vicino a zero.

La sigmoide e la tangente iperbolica soffrono del problema della scomparsa del gradiente (vanishing gradient). Grandi cambiamenti nell'input corrispondono a piccoli cambiamenti nell'output, rendendo di conseguenza le derivate piccole. Durante la backpropagation le derivate vengono moltiplicate insieme, quindi al crescere del numero di layer il gradiente scomparirà sempre di più.

Entrambe le funzioni sono molto sensibili a cambiamenti vicino al punto centrale, 0.5 per \tanh e 0 per la sigmoide.

5.2 ReLU

La Rectified Linear Unit (ReLU) è la funzione di attivazione più comune negli hidden layers. E' semplice da implementare e calcolare, inoltre è meno soggetta al problema del vanishing gradient. Tuttavia può soffrire di altri problemi come la saturazione o neuroni morti.

Non è una funzione lineare e non è differenziabile nel punto 0. Nella pratica è trascurabile poiché è poco probabile ottenere 0. Per valori negativi il risultato sarà zero e quindi i neuroni non verranno attivati rendendo sparsa la rete e quindi più semplice da computare.

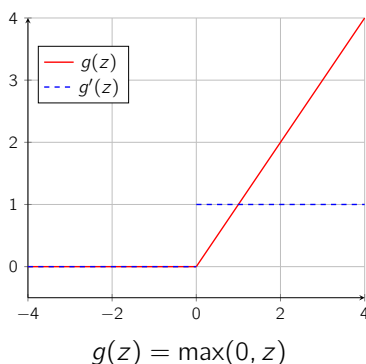


Figure 5.3: funzione ReLU e la sua derivata

Generalization of ReLU

Softplus: versione smussata che approssima la ReLU e la rende derivabile in 0. nella pratica non funziona come dovrebbe.

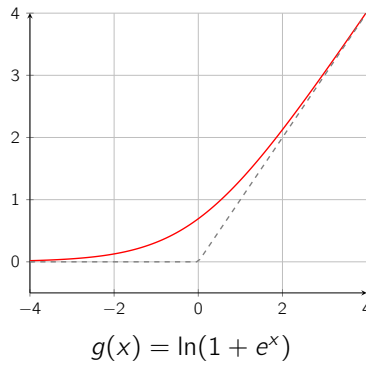


Figure 5.4: Softplus

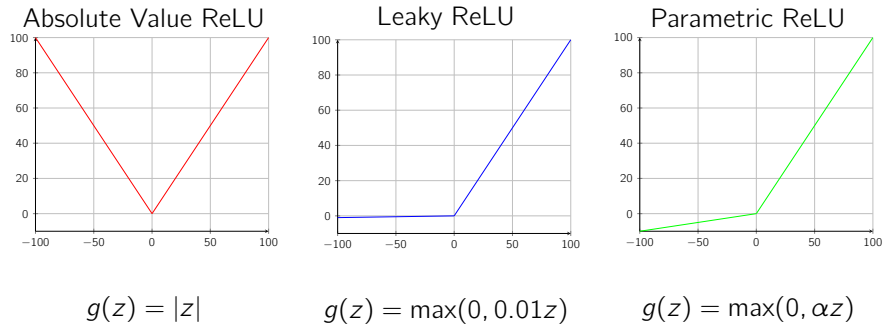
Possono essere definite altre forme di ReLU tramite la seguente formula:

$$h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Absolute value rectification: utile per trovare features che siano invarianti per qualche caratteristica. $\alpha_i = -1$, $g(z) = |z|$

Leaky ReLU: per valori negativi viene definita una componente lineare molto piccola con $\alpha_i = 0.01$. Può essere utile nelle reti con tanti neuroni disattivati.

Parameter RLU: il parametro α_i viene imparato dalla rete



Maxout units: impara la funzione da se. Viene divisa il parametro z in gruppi da k parti e viene ritornato il massimo di ogni gruppo.

$$g(z)_i = \max_j z_j \quad j \in G^{(i)} : \text{indice } j\text{-esimo del gruppo } G^{(i)}$$

Dato che richiede più parametri da essere imparati sono necessario più dati o più regolarizzazione.

Chapter 6

Variance-Bias Trade-off

L'errore di generalizzazione è determinato da diversi fattori:

- **Irreducible Error**: errore che dipende dalla qualità e dalla quantità dei dati che non può essere ridotto.
- **Variance**: errore dovuto alla sensibilità del modello alle piccole variazioni nei dati. Un'alta varianza determina un elevato adattamento ai dati.
- **Bias**: errore che misura quanto le predizioni del modello si distaccano dai valori attesi. Un alto bias determina una bassa accuratezza.

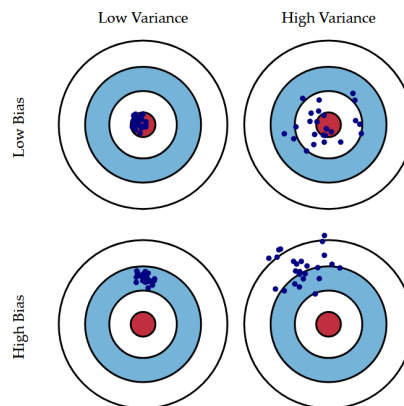


Figure 6.1: Illustrazione grafica di bias e variance [3].

Un modello con bias alto fa delle assunzioni troppo semplici sui dati (underfitting). Mentre un modello con alta varianza si adatta troppo ai dati (overfitting). Un buon modello deve avere bassa varianza e basso bias.

La complessità del modello influenza bias e variance. All'aumentare della complessità la variance crescerà e il bias decrescerà.

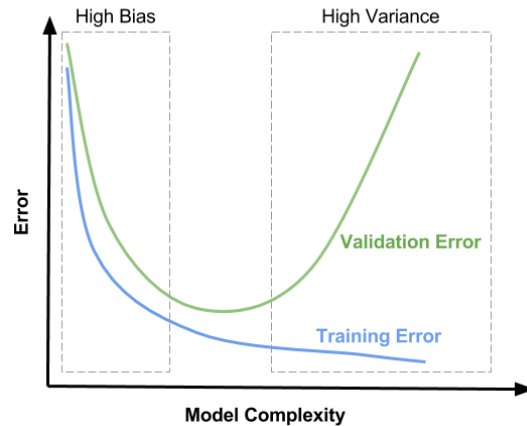


Figure 6.2: Bias-variance tradeoff [5].

Confrontando l'errore sul train set e sul validation set è possibile osservare il comportamento di bias e variance.

Quando le due curve di errore sono vicine significa che la varianza è bassa e si adatta ai dati di entrambi gli insiemi. Viceversa quando le curve sono distanti la varianza è alta e si adatta molto sul train set (overfitting).

Se l'errore è alto vuol dire che il bias è alto e che il modello è troppo semplice e non riesce a rappresentare i dati (underfitting).

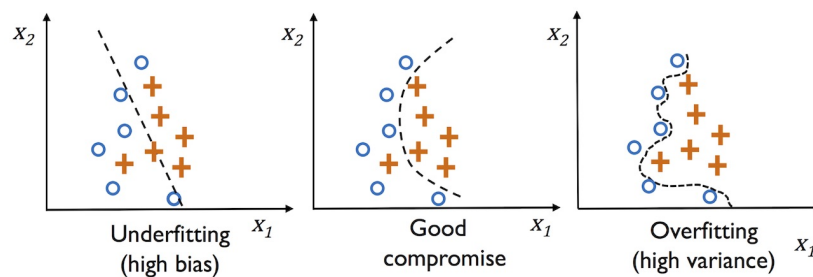


Figure 6.3: Underfitting e overfitting [9].

Per ridurre l'errore di generalizzazione (variance e bias) è possibile usare tecniche di regolarizzazione, boosting o bagging.

Chapter 7

Regularization

Le tecniche di regolarizzazione puntano a ridurre l'errore della loss function sul validation set e sul test set al costo di un incremento ragionevole sul train error.

La regolarizzazione comprende qualsiasi tecnica che effettua una modifica nel processo di apprendimento al fine di ridurre l'errore di generalizzazione, ma non il train error.

La regolarizzazione può avvenire:

- Direttamente: aggiungendo vincoli ai parametri o alla funzione obiettivo
- Indirettamente: aggiungendo dati e introducendo rumore nei dati

Un regolarizzatore efficace riduce significativamente la varianza mentre non aumenta molto il bias.

7.1 Norm Penalties

Si limita la capacità del modello aggiungendo una penalità $\Omega(\theta)$ alla funzione obiettivo J .

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

$\alpha \in [0, \infty)$ è un iperparametro che pesa il contributo della **norm penalty** nella funzione obiettivo.

Solitamente la penalty Ω penalizza solo i pesi della trasformazione affine di ogni layer. I bias nella trasformazione affine richiedono meno dati per fittare, quindi non vengono regolarizzati.

Più parametri ci sono nel modello più questi sono sensibili a varianza, e quindi creano più instabilità nel modello.

- somma assoluta dei pesi

$$\Omega(w, b) = \sum_{w_j} |w_j|$$

- somma quadratica dei pesi

$$\Omega(w, b) = \sqrt{\sum_{w_j} |w_j|^2}$$

Data la funzione obiettivo seguente

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

Il gradiente é:

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

La regola di aggiornamento dei pesi usando la norma L2 diventa la seguente

$$\begin{aligned} w &= w - \varepsilon (\alpha w + \nabla_w J(w; X, y)) \\ &= (1 - \varepsilon \alpha) w + \varepsilon \nabla_w J(w; X, y) \end{aligned}$$

7.2 Data Augmentation

Il miglior modo per ottenere una migliore generalizzazione è usare più dati per l'addestramento. Spesso i dati sono limitati e necessitano di essere etichettati.

Una soluzione è quella di generare nuovi dati artificialmente riutilizzando quelli già presenti applicando delle trasformazioni o aggiungendo rumore.

- Flip (Horizontal, Vertical)
- Random Noise Injection (all'input o ai pesi)
- Rotations
- Cropping
- Color Modification

E' utile effettuare l'addestramento sia sul dataset esteso che sul dataset di partenza per poterne valutare i vantaggi.

7.3 Label Smoothing

Può capitare che alcuni dataset presentino errori nelle etichette. Questi possono essere dovuti a errori umani o al risultato di algoritmi di labelling automatico. Per gestire questi possibili errori si modella esplicitamente il rumore nelle label.

Il label smoothing è una tecnica che si basa su una softmax con k valori di output in cui 0 e 1 vengono sostituiti nel seguente modo:

$$0 \Rightarrow \frac{\varepsilon}{k-1} \quad 1 \Rightarrow 1 - \varepsilon$$

in cui ε è la probabilità che la label sia corretta.

7.4 Early Stopping

L'early stopping è una tecnica di regolarizzazione che evita di addestrare eccessivamente il modello sul train set in modo da evitare l'overfitting.

Ad ogni epoca viene osservato il validation error e vengono salvati i parametri del modello migliore. Dopo un numero di iterazioni (patience) che il validation error continua a peggiorare l'algoritmo si ferma.

Esistono più strategie per sfruttare questa tecnica:

1.
 - Si esegue l'early stopping
 - Viene riaddestrato il modello fino al punto in cui si è fermato con i parametri ottenuti
2.
 - Si esegue l'early stopping
 - Si continua l'addestramento con i parametri ottenuti utilizzando solo il validation set

7.5 Parameter Tying and Sharing

7.6 Multitask Learning

Il multitask learning è un approccio in cui si cerca di imparare diversi tasks contemporaneamente. Permette di migliorare la generalizzazione mettendo in comune degli esempi per più task.

Il task aggiuntivo aggiunge dei vincoli ai parametri dei layer condivisi e permette di imparare una rappresentazione condivisa e quindi di ottenere una maggiore generalizzazione.

L'architettura di una rete multitask è comunemente composta da due parti: i **task-specific layers** che traggono beneficio solo dagli esempi definiti per i task specifici e gli **shared layers** che traggono beneficio dai dati messi in comune.

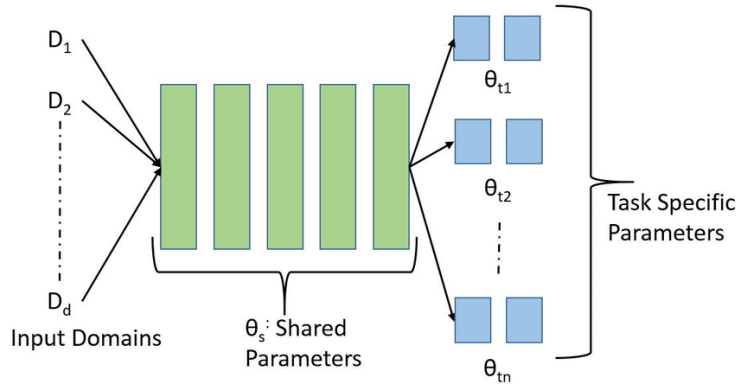


Figure 7.1: Schema di rete multitask [8]

E' necessario trovare il giusto insieme di parametri da condividere: una scarsa condivisione di parametri non risulta efficace, mentre un'eccessiva condivisione potrebbe portare a ottenere prestazioni peggiori [1].

Il miglioramento della generalizzazione avviene solo se i task hanno qualcosa in comune (correlazione).

7.7 Bagging

Il bagging (bootstrap aggregation) è una tecnica di **ensemble method** che riduce l'errore di generalizzazione combinando diversi modelli.

Viene effettuato l'addestramento di k modelli diversi su k insiemi del training set ottenuti tramite un campionamento casuale con sostituzione¹(bootstrap). I risultati vengono aggregati (aggregation) calcolando una media (soft voting) nel caso di regressione o tramite una votazione (hard voting) nel caso di classificazione.

Se un singolo membro dell'ensemble fa degli errori, l'ensemble funzionerà meglio dei membri.

Solitamente il numero di modelli è piccolo poiché risulta oneroso gestirne tanti in termini di computazione e di memoria. E' meglio avere un numero di modelli dispari per non avere casi di parità.

7.8 Dropout

¹Nel campionamento casuale con sostituzione i campioni possono essere scelti più volte

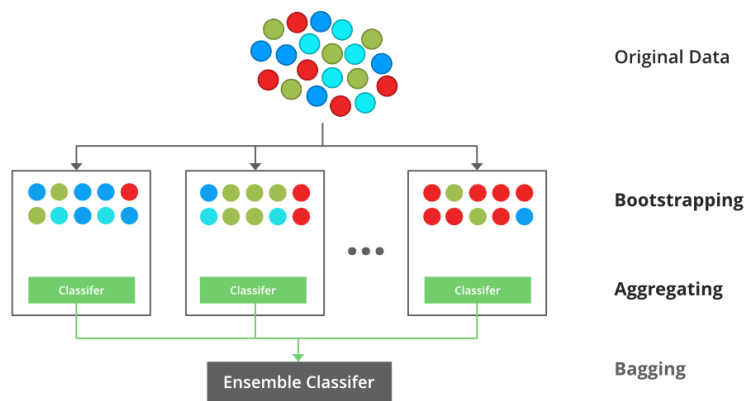


Figure 7.2: Schema di funzionamento del bagging [10]

Bibliography

- [1] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*, 2020.
- [2] Ashkan Eliasy and Justyna Przychodzen. The role of ai in capital structure to enhance corporate funding strategies. *Array*, 6:100017, 07 2020.
- [3] Scott Fortmann-Roe. Understanding the bias-variance tradeoff, 6 2012. [visited on 07/11/2021].
- [4] Simon Larsson. Possible for batch size of neural network to be too small?, 05 2019. [visited on 30/10/2021, modified].
- [5] Satya Mallick. Bias-variance tradeoff in machine learning — learnopencv, 02 2017. [visited on 21/10/2021].
- [6] Mohamed Nagy. A modified method for detecting ddos attacks based on artificial neural networks - scientific figure on researchgate, 04 2019. [visited on 30/10/2021].
- [7] Phung and Rhee. A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, 9:4500, 10 2019.
- [8] Rajeev Ranjan. An all-in-one convolutional neural network for face analysis - scientific figure on researchgate, 11 2016. [visited on 04/11/2021].
- [9] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning, 3rd Ed.* Packt Publishing, Birmingham, UK, 3 edition, 2019.
- [10] soumya7. Bagging vs boosting in machine learning, 05 2019. [visited on 04/11/2021].