Advanced Machine Learning

Giuseppe Magazzù

2021 - 2022

Contents

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 $f(x)$.

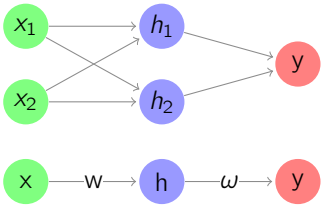
XOR Example

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



$h = f^{(1)}(x;w,c)$

$y = f^{(2)}(h;\omega,c)$

$f(x;w,c,\omega,b) = f^{(2)}(f^{(1)}(x))$

$f(x;w,c,\omega,b) = max\{0, xw + c\}\ \omega + b$

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 attraversi molteplici filtri

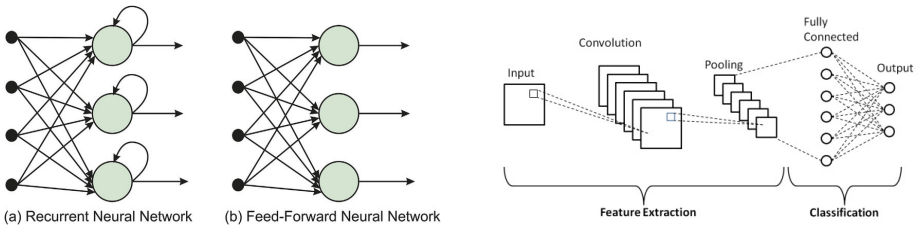


Figure 1.1: Differenza tra RNN, FFNN [1] e CNN [3]

1.2 Feed Forward Neural Network

Una FFNN è una rete fully connected, ovvero che ogni neurone in un layer e 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

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

Sia $\hat{y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_k)$ 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.

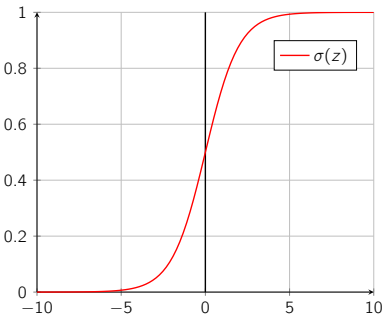
$\hat{y} = w^T h + b$

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 $z = \omega^T h + b$ e poi applichiamo la sigmoide per trovare $\hat{y} = \sigma(\omega^T h + b)$.

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

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

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

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

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

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

4.2.3 Softmax - Distribuzione Multinoulli

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

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

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

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

Calcolando il logaritmo della softmax possiamo riscriverla nel seguente modo:

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

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 invertono 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

Chapter 5

Regularization

Le tecniche di regolarizzazione puntano a ridurre l'errore della loss function sul validation set e sul test set.

La regolarizzazione può avvenire:

- Direttamente: cambiando i vincoli o la funzione obiettivo
- Indirettamente: aggiungendo dati

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

Controllare la complessità del modello non è semplice, non basta trovare la dimensione giusta il numero giusto di parametri. Nel deep learning si basa su trovare il miglior modello che è un modello grande che è stato propriamente regolarizzato.

5.1 Norm Penalities

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

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

$\alpha \in [0, \text{inf})$ 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.

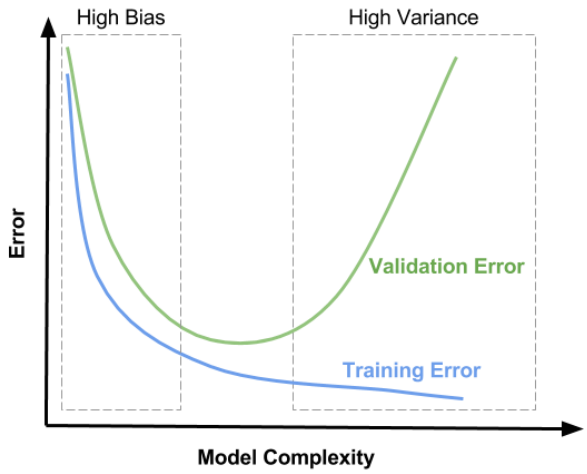


Figure 5.1: Bias-variance tradeoff example [2]. High bias → underfitting, High variance → overfitting

- 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

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

Bibliography

[1] Ashkan Eliasy and Justyna Przychodzen. The role of ai in capital structure to enhance corporate funding strategies. *Array*, 6:100017, 07 2020.

[2] Satya Mallick. Bias-variance tradeoff in machine learning — learnopencv, 02 2017. [visited on 21/10/2021].

[3] 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.