# POLITECNICO DI TORINO

Collegio di Ingegneria Aerospaziale

**Corso di Laurea Magistrale**
**in Ingegneria Aerospaziale**

Tesi di Laurea Magistrale

## Risoluzione delle equazioni di Navier-Stokes tramite reti neurali


**Relatori**

Prof. Domenic D'Ambrosio
Dott. Manuel Carreno Ruiz

**Candidato**

Paolo Visentini

Aprile 2023

**Sommario**

La risoluzione delle equazioni di Navier-Stokes risulta essere un punto chiave nella realizzazione di qualsiasi progetto ingegneristico in ambito aerospaziale. Normalmente le tecniche classiche della CFD (Computational Fluid Dynamics) fanno riferimento principalmente a metodi basati su metodi a pannelli, differenze finite, elementi finiti oppure su volumi finiti, questo comporta la necessità di una discretizzazione dello spazio tramite la creazione di una mesh sulla quale verrà effettuata, tramite computazione in ogni suo nodo o cella, la simulazione. Storicamente la CFD affonda le sue origini all'inizio degli anni 30 del secolo scorso, anche se possiamo identificare l'inizio della CFD nella forma utilizzata modernamente negli anni 80, decennio nel quale si iniziava ad avere sufficiente forza computazionale per trattare problemi di interesse applicativo. Negli ultimi 20 anni il ML (Machine Learning) ha visto la sua crescita in diversi ambiti come: la ricostruzione di immagini, processazione del linguaggio, scienze cognitive e genomica. Grazie a questa spinta recentemente sono nati nuovi strumenti per la velocizzazione e il miglioramento delle simulazioni fluidodinamiche, in questa introduzione verrà quindi fornita una panoramica di queste tecniche. Le DNS (Direct Numerical Simulations) sono un approccio in cui le equazioni di governo della fluidodinamica vengono discretizzate e integrate nel tempo con un sufficiente numero di gradi di libertà per catturare tutte le strutture del campo di moto. I flussi turbolenti sono caratterizzati dalla presenza di strutture su diverse scale e si hanno strutture vorticose in un ampio range dimensionale ed energetico. Questa complessità richiede l'utilizzo di mesh molto fini e di schemi numerici molto accurati per evitare di compromettere i risultati a causa degli artefatti numerici. Sebbene con le DNS sia possibile ottenere una soluzione accurata, teoricamente a qualsiasi livello, bisogna pagare un alto costo computazionale, il quale incrementa all'aumentare del numero di Reynolds del problema. Diversi approcci di ML sono stati presentati per migliorare l'efficienza delle DNS. Ad esempio Bar-Sinai et al. [2] ha proposto una tecnica basata sul deep learning per stimare le derivate spaziali su griglie a bassa risoluzione, dimostrando di poter migliorare la qualità dei risultati rispetto a metodi alle differenze finite. Un approccio simile è stato anche sviluppato da Stevens e Colonius [4] per migliorare i risultati di uno schema alle differenze finite del quinto ordine nella cattura di un'onda d'urto. Questi sono solo alcuni degli esempi

dell'utilizzo del deep learning per migliorare gli schemi numerici "classici". Un altro esempio di come il ML possa velocizzare le simulazioni è stato proposto da Kochkov et al.[6] sulla falsa riga delle tecniche usate per la ricostruzione di immagini in alta definizione. Nel suo lavoro Kochkov considera il flusso di Kolmogorov bidimensionale con una forzante per mantenere le fluttuazioni, viene quindi eseguita una simulazione tradizionale su una griglia grezza, i risultati vengono quindi processati tramite deep learning per ottenere la soluzione su una griglia da 8 a 10 volte più fine. Nella Figura 1 viene riportata la distribuzione di vorticità ottenuta. Le tecniche fin ora presentate sebbene di grande interesse si basano su un approccio ibrido che richiede comunque la soluzione delle N-S tramite un risolutore "classico", non è quindi possibile ritenere queste tecniche come alternative ma semmai come strumenti per velocizzare e migliorare gli schemi già esistenti. L'obbiettivo di questa tesi è indagare un nuovo paradigma che si sta aprendo nella risoluzione delle PDE (Equazioni alle derivate parziali) tramite ML e di valutare l'applicazione di queste nuove tecniche alla CFD. Nel 2017 Raissi et al. hanno mostrato [14] la possibilità di applicare delle reti neurali per la soluzione di diverse equazioni differenziali, denominandole PINNs (Physics Informed Neural Networks). Le reti PINN hanno il grande vantaggio di non essere data-driven ovvero di non necessitare di dati ottenuti in precedenza e di poter quindi risolvere, almeno in teoria, le N-S al pari di un risolutore CFD classico. Per la compilazione delle reti neurali e l'implementazione del training sono state usate le librerie Modulus e Spline-PINN che a loro volta si basano su Pytorch, un framework open source per il ML. Verrà innanzitutto data una breve descrizione qualitativa del funzionamento di una rete neurale applicata ad un caso semplice, successivamente verrà introdotta la rete PINN e trattata la struttura matematica di quest'ultima. Affrontata la teoria verranno evidenziati i commentati alcuni risultati per il caso 1D dell'equazione di Burgers, evidenziando come la rete PINN riesca a catturare la discontinuità presente nella soluzione. Successivamente verrà introdotto il framework Nvidia Modulus, con il quale verranno affrontate le simulazioni stazionarie su domini 2D e mostrate alcune caratteristiche peculiari delle reti neurali come il TL (transfer learning), inoltre si mostrerà come l'architettura di base fully connected sulla quale poggia PINN sia insoddisfacente per simulazioni CFD che non siano effettuate a numeri di Reynolds estremamente ridotti. Infine verrà introdotta Spline-PINN una rete neurale convoluzionale con una architettura complessa (U-Net), in grado di for-

nire risultati più accurati a numeri di Reynolds maggiori, tenendo inoltre conto della dipendenza temporale. I risultati verranno discussi in una logica sia quantitativa che qualitativa, infatti per quanto sia di largo interesse ottenere simulazioni accurate nella stima dei valori delle grandezze del campo fluidodinamico, è certamente utile avere a disposizione, in fase di avanprogetto, uno strumento che sia in grado velocemente di valutare l'andamento di grandezze come ad esempio la resistenza aerodinamica al variare di diversi parametri geometrici.

# Indice

# Elenco delle figure

# Capitolo 1

# Cenni sul funzionamento di una rete neurale

Una rete neurale (ANN artificial neural network) è un modello di calcolo basato su una serie di algoritmi che cercano di ricalcare il funzionamento del sistema nervoso. È costituita da un numero di unità di elaborazione, chiamate "neuroni", che sono interconnesse tra loro e che lavorano insieme per risolvere problemi complessi.

In una rete neurale, ogni neurone riceve input da altri neuroni o da sensori esterni e calcola un output in base a questi input utilizzando una funzione di attivazione. L'output di un neurone viene quindi trasmesso agli altri neuroni della rete, che a loro volta calcolano i loro output utilizzando le stesse regole. Questo processo viene ripetuto fino a quando non viene raggiunto un risultato soddisfacente.

Una delle caratteristiche principali delle reti neurali è che possono "imparare" a partire da dati di esempio, senza dover essere programmate esplicitamente per risolvere un determinato problema. In altre parole, possono adattarsi a nuove situazioni in modo autonomo, migliorando le loro prestazioni man mano che acquisiscono nuove informazioni.

## 1.1 Neuroni e funzioni di attivazione

In questa sezione verranno presentate due tipologie di neuroni differenti: il percettrone e il sigmoide. Il primo servirà per introdurre la logica di base che sta dietro al funzionamento di un'unità neuronale, il secondo invece sarà un esempio operativo. Verrà

inoltre introdotto il concetto di funzione di attivazione, essa è l'effettiva parte analitica che computa il valore del neurone stesso e ne determina quindi la tipologia.

### 1.1.1 Percettrone

Per spiegare il funzionamento di una rete neurale bisogna partire dal cuore di essa ovvero il neurone artificiale. Una delle tipologie più semplice di neurone è il percettrone, esso è stato introdotto negli anni 50 da Frank Rosenblatt. Un percettrone prende diversi input binari e produce un singolo output binario come mostrato nella figura sottostante:

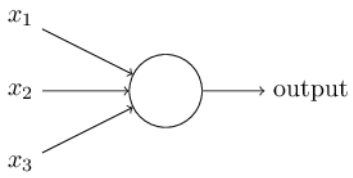


Figura 1.1: Percettrone a 3 ingressi.[24]

In questo esempio si hanno in ingresso 3 input $x_1, x_2, x_3$. Vengono introdotti dei pesi (weights $w_1, w_2, \cdots$) numeri reali che esprimono l'importanza degli input. L'output del neurone sarà 0 oppure 1 se il risultato della somma pesata $\sum_j w_j x_j$ è maggiore o minore di un dato valore di soglia, in forma algebrica:

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j <= soglia \\ 1, & \text{if } \sum_j w_j x_j > soglia \end{cases} \tag{1.1}$$

questa semplice modello matematico è la base del funzionamento di una rete neurale, variando i pesi e la soglia si possono ottenere diversi modelli decisionali. Per ottenere una rete neurale basta creare colonne di percettroni (layer) da collegare una con l'altra, in modo che un percettrone abbia come input i valori pesati dei percettroni nel layer precedente, come mostrato nella figura sottostante.

Il percettrone è definito con un singolo output, nel disegno le frecce che escono da ogni neurone hanno il semplice scopo di mostrare che lo stesso output prodotto viene dato in input a tutti i neuroni del layer successivo (architettura fully connected). Ora per una trattazione analitica più adeguata verrà riscritto il sistema precedente adottanto una notazione vettoriale e introducendo il concetto di bias: $b$ = - threshold, quindi in

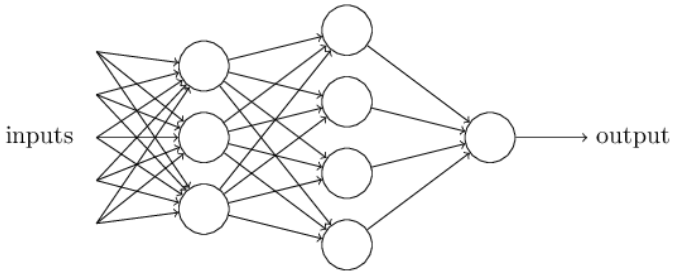


Figura 1.2: MPL (multy layer perceptron)[24].

formule:

$$output = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b <= 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases} \tag{1.2}$$

il bias può essere interpretato come una misura della facilità con cui il percettrone porta il suo output ad 1. Facendo un paragone con il neurone biologico, il bias misura la facilità con cui quel determinato neurone si attiva.

#### Difficoltà nell'utilizzo del percettrone

Supponiamo di avere una rete di percettroni e di volerla allenare alla risoluzione di un problema. Per ottenere il comportamento sperato dalla rete è necessario variare i pesi e i bias, quello che vorremmo è che a piccole variazioni dei pesi (o bias) corrisponda una piccola variazione dell'output.

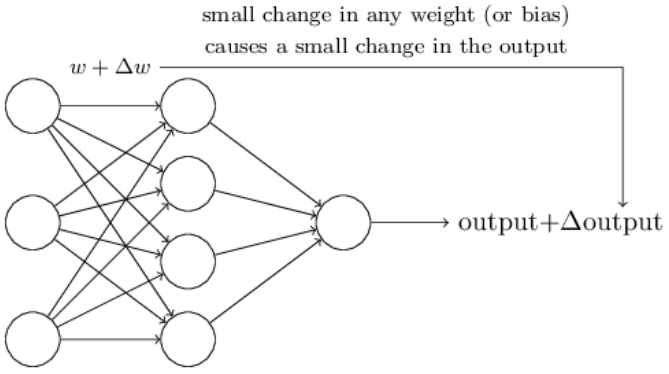


Figura 1.3: Cambiare il valore del peso di una connessione influenza l'output.[22]

Ad esempio una rete che deve identificare delle cifre rappresentate su delle immagini, può trovarsi nella situazione di essere in grado di riconoscere correttamente alcuni numeri e non altri, godere della proprietà per cui piccoli cambiamenti dei pesi portano

a piccoli cambiamenti sull'output permette di poter modificare i parametri della rete senza stravolgere il comportamento di quest'ultima, in questo modo facendo diversi aggiustamenti è possibile addestrare la rete al riconoscimento di nuovi numeri senza perdere "l'abilità" di riconoscere i precedenti.

Questo comportamento non è proprio di una rete di percettroni. Infatti, un piccolo cambiamento nei pesi di un singolo percettrone nella rete può portare l'output a variare drasticamente da 0 ad 1 (o viceversa). Questa variazione può cambiare completamente il comportamento della rete, rendendo il training piuttosto complesso.

### 1.1.2 Sigmoide

Il neurone sigmoide è simile al percettrone, ma con una struttura analitica tale che piccoli cambiamenti nei pesi e bias portino a piccoli cambiamenti nell'output, questa caratteristica permette ad una rete di sigmoidi di imparare. [1] Ecco una rappresentazione del sigmoide:

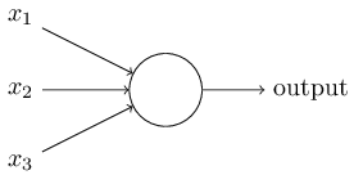


Figura 1.4: Sigmoide a 3 ingressi.[24]

come un percettrone, il sigmoide prende in input dei valori provenienti da altri neuroni, ma invece che ricevere solo il valore 0 o il valore 1, ogni valore di input può essere compreso tra 0 ed 1. Sempre come il percettrone i valori di ingresso vengono pesati e viene aggiunto il bias, l'output però non sarà più 0 o 1 ma sarà $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$, dove $\sigma$ è la funzione (di attivazione) sigmoide, la sua definizione analitica è la seguente:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.3}$$

l'argomento $z$ dipenderà dagli input pesati e dal bias, quindi esplicitando la dipendenza otteniamo:

$$\sigma = \frac{1}{1 + exp(-\sum w_j x_j - b)} = \frac{1}{1 + exp(-\mathbf{w} \cdot \mathbf{x} - b)} \tag{1.4}$$

[1] Il teorema dell'approssimazione universale (G. Cybenko) garantisce che sia possibile approssimare qualsiasi funzione continua con una rete neurale avente un sufficiente numero di sigmoidi anche con un solo hidden layer.

si può notare la somiglianza con il modello del percettrone, infatti se $z$ è un grosso numero positivo allora avremo che $\sigma$ tenderà ad 1, viceversa nel caso in cui $z$ sia fortemente negativo $\sigma$ tenderà a 0. La vera differenza risiede nei valori intermedi che la funzione sigmoide può assumere rispetto alla funzione a gradino che è peculiare del percettrone.

Matematicamente questo si traduce nel notare che la funzione sigmoide risulta essere una funzione liscia rispetto invece alla funzione a gradino.

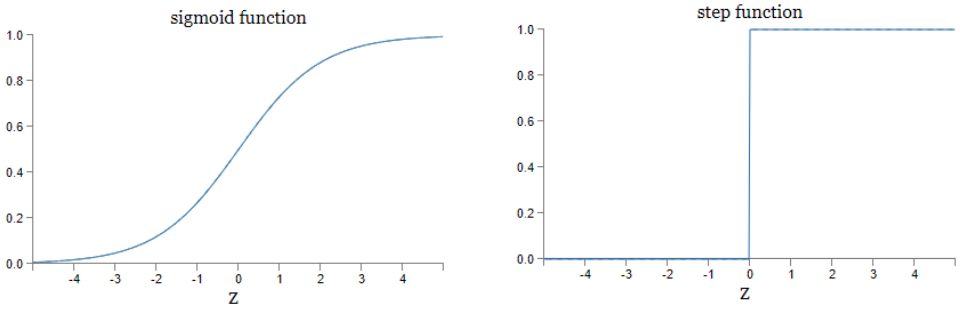


Figura 1.5: Confronto grafico funzione a gradino e funzione sigmoide.[22]

La liscezza di $\sigma$ permette di poter scrivere il differenziale dell'output e quindi si ha la garanzia che a piccole variazioni dei pesi o bias corrisponda una piccola variazione dell'output, infatti utilizzando una notazione discreta si può scrivere:

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{1.5}$$

La linearità di questa espressione è ciò che garantisce che le variazioni dell'output siano piccole a patto di piccole variazioni nei pesi e bias. Quindi nonostante il sigmoide abbia lo stesso comportamento qualitativo del percettrone, essi risultano più semplici da addestrare potendo adottare valori intermedi.

### 1.1.3 Funzioni di attivazione

Come è stato presentato precedentemente la differenza tra le due tipologie di neuroni è insita nella funzione analitica che determina il valore dell'output, questa funzione prende il nome di funzione di attivazione. Ciò che è importante notare è che non è importante la funzione in se, ad esempio il sigmoide, nel funzionamento di una rete ma piuttosto la forma della funzione utilizzata. In generale quindi si possono definire diverse funzioni di attivazione come la funzione ReLu oppure la funzione Wish, ad ogni

modo una qualsiasi funzione di attivazione può essere descritta nelle seguente forma $f = f(\mathbf{w} \cdot \mathbf{x} + b)$. Alcune funzioni si sono dimostrate storicamente migliori di altre in termini di velocità di training e stabilità.

## 1.2 Architettura e ricerca a gradiente

In questa sezione verrà introdotta la terminologia che descrive l'architettura di una rete neurale, verrà presentato il concetto di funzione di costo e di ricerca a gradiente.

### 1.2.1 Architettura di una rete neurale

Introduciamo ora alcuni termini, con riferimento alla figura sottostante, è stata scelto come esempio l'architettura di una rete che prende in input un immagine da 784 pixels (28 x 28) raffigurante un numero da 0 a 9.

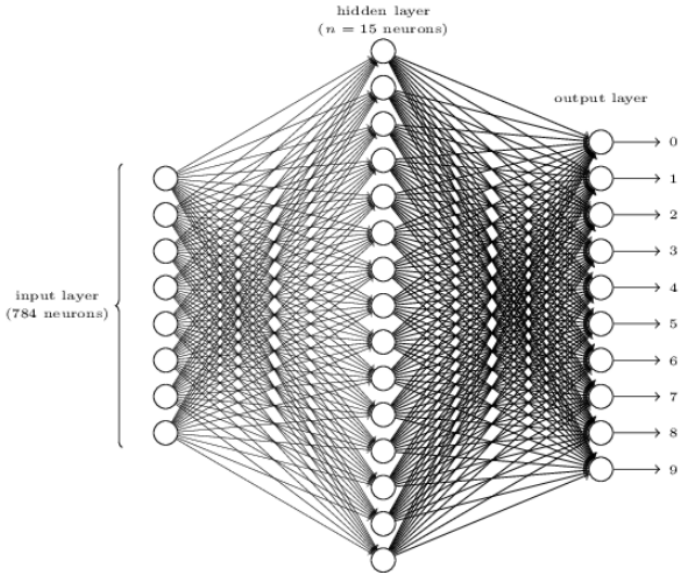


Figura 1.6: Rete neurale ad un hidden layer con architettura fully conncted foward.[12]

La colonna di neuroni più a sinistra è detta input layer e prende ogni singolo valore di colore assegnato a ciascun pixel dell'immagine, la colonna più a destra è detta output layer. Se ad esempio viene data un'immagine raffigurante un 3 in input, ci si aspetta che l'output sia un vettore avente tutti zero tranne nella posizione 4 dove ci aspettiamo un valore unitario. La colonna centrale è il vero cuore della rete ed è detta hidden layer, a seconda della complessità si possono avere più layer contenenti diversi numeri di neuroni.

La rete rappresentata rientra nella tipologia delle reti "forward" con layer "fully connected". Il primo termine si riferisce al fatto che i neuroni sono sempre connessi al layer successivo e non sono presenti loop, ovvero il valore di ogni neurone dipende sempre dai valori dei neuroni precedenti e mai successivi. Il secondo termine invece indica che tutti i neuroni di un layer siano collegati a tutti i neuroni del layer successivo, in contrasto, in architetture più avanzate, possono essere introdotti dei layer convoluzionali, i quali sfruttando delle connessioni "alternate" riescono ad effettuare delle operazioni di filtraggio dei dati inseriti. Le reti che sfruttano layer convoluzionali sono dette CNN (convolutional neural network).

Il numero di neuroni in ogni hidden layer e il numero di layer stesso rientrano negli hyperparameters della rete, ovvero quell'insieme di paramenti che caratterizzano una rete da un'altra e ne determinano le perfomance.

### 1.2.2 Funzione di costo

Nell'addestramento è necessario introdurre una funzione che misuri l'errore commesso dalla rete rispetto all'output voluto. Utilizzando sempre la rete precedente come esempio, viene chimato $\mathbf{x}$ il dato di input, in questo caso un vettore 784-dimensionale. Verrà invece definito $\mathbf{y}(\mathbf{x})$ il vettore di output desiderato, ad esempio si introduce un'immagine di input raffigurante un 6 il vettore di output desiderato sarà $\mathbf{y}(\mathbf{x}) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$.

Quello che ora verrà fatto è la creazione di un algoritmo che sia in grado di valutare se l'output ottenuto sia o meno soddisfacente, per raggiungere questo scopo basta introdurre la funzione di costo:

$$C(w, b) = \frac{1}{2n} \sum_x \|\mathbf{y}(\mathbf{x}) - \mathbf{a}\|^2 \tag{1.6}$$

In cui $w, b$ sono i pesi e bias della rete, $n$ è il numero di training inputs, $\mathbf{a}$ è il vettore di output prodotto dalla rete in risposta all'input $\mathbf{x}$ e la somma è eseguita su tutti i training inputs.

La funzione di costo non è altro che l'errore quadratico medio (MSE) della rete, si prende la versione quadratica dell'errore in modo tale da avere un termine sempre positivo, quando $C(w, b)$ risulta tendente a 0 allora si può concludere che l'errore medio è tendente a zero e che quindi la rete abbia imparato ad eseguire il suo compito. Il fatto di prendere la funzione di costo $C$ sempre positiva porta alla diretta conseguenza che

il suo minimo sia proprio il valore cercato, viene quindi naturale adottare un algoritmo di ricerca a gradiente per trovare i pesi e i bias che minimizzano la funzione di costo.

### 1.2.3 Ricerca a gradiente e ricerca a gradiente stocastica

La ricerca a gradiente viene effettuata per trovare il minimo (in genere locale) di una funzione, per vedere come funziona l'algoritmo si parte dalla scrittura discreta del differenziale in base ad uno spostamento $\Delta\mathbf{v}$, per semplificare la trattazione immaginiamo di essere in due dimensioni:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \tag{1.7}$$

Quello che si vuole fare è trovare un metodo per scegliere $\Delta v_1$ e $\Delta v_2$ in modo tale da ottenere un $\Delta C$ negativo e quindi diminuire la funzione di costo, definendo il gradiente di $C$ come $\nabla C$ si può scrivere la seguente relazione di proporzionalità:

$$\Delta C \approx \nabla C \cdot \Delta\mathbf{v} \tag{1.8}$$

Quindi scegliendo $\Delta\mathbf{v} = -\eta\nabla C$ ci si assicura che la variazione $\Delta C$ sia negativa, infatti:

$$\Delta C \approx -\eta\nabla C \cdot \nabla C \approx -\eta\|\nabla C\|^2 \tag{1.9}$$

siccome $\|\nabla C\|^2 \geq 0$ allora risulterà sempre $\Delta C \leq 0$, $\eta$ è chiamato learning rate ed è incluso negli hyperparameters della rete. Un $\eta$ più grande porta ad un più grande passo di addestramento e quindi a tempi di training minori, ma deve essere piccolo a sufficienza per garantire la validità dell'equazione (1.8). Si supponga di voler trovare $\Delta\mathbf{v}$ che faccia decrescere $C$ il più possibile, questo equivale a trovare il minimo di $\Delta C \approx \nabla C \cdot \Delta\mathbf{v}$, ovvero trovare il $\Delta C$ più negativo possibile. Dalla teoria del calcolo differenziale è possibile dimostrare che scelto un $\epsilon > 0$ piccolo e fissato, $\Delta\mathbf{v} = -\eta\nabla C$ minimizza $\Delta C \approx \nabla C \cdot \Delta\mathbf{v}$, dove $\eta = \frac{\epsilon}{\|\nabla C\|}$, risulta perciò che $\|\Delta\mathbf{v}\| = \epsilon$

### 1.2.4 Mini-batch e ricerca a gradiente stocastica

Ricapitolando il funzionamento della ricerca a gradiente, introducendo però questa volta al posto delle variabili $v_j$ i pesi e i bias. Il nostro obbiettivo è scrivere i nuovi pesi e bias ottenuti aggiornando i precedenti dopo un passo della discesa a gradiente questo si traduce in formule come:

$$w_k \to w_k^{'} = w_k - \eta \frac{\partial C}{\partial w_k} \tag{1.10}$$

$$b_l \to b_l^{'} = b_l - \eta \frac{\partial C}{\partial b_l} \tag{1.11}$$

Notare che il numero di pesi k non è lo stesso del numero di bias l. Per ottenere le derivate della funzione di costo rispetto ai pesi e bias è necessario calcolare il gradiente e di conseguenza la funzione di costo $C$. La funzione di costo ha la seguente forma $C = \frac{1}{n}\sum_x C_x$ ovvero una media delle funzioni di costo dei ottenute dai singoli training inputs $C_x = \frac{\|\mathbf{y}(\mathbf{x})-\mathbf{a}\|^2}{2}$. In pratica per calcolare il gradiente abbiamo bisogno di calcolare tutti i gradiente $\nabla C_x$ separatamente per ogni training input e calcolarne il valore medio $\nabla C = \frac{1}{n}\sum_x \nabla C_x$. Un ciclo completo sul dataset viene chiamato epoch. Sfortunatamente, quando il numero di training inputs risulta essere molto grande questa tecnica può richiedere molto tempo rallentando il training.

Un idea per risolvere questo problema è chiamata discesa a gradiente stocastica e può essere usata per velocizzare l'apprendimento della rete. In pratica si stima il valore del gradiente $\nabla C$ calcolando $\nabla C_x$ su un piccolo sample di training inputs scesi arbitrariamente. Eseguendo le operazioni di media su questo piccolo insieme di inputs si può calcolare il gradiente della funzione di costo con un'ottima precisione, velocizzando così il training. La discesa a gradiente stocastica lavora selezionando un piccolo numero $m$ di training input: $X_1, X_2, \cdots, X_m$, questo insieme di "punti" viene chiamato mini-batch. Se $m$ è sufficientemente grande ci aspettiamo che il valore medio di $\nabla C_{X_j}$ sia all'incirca uguale al valore medio su tutti i $\nabla C_x$, in formule:

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \tag{1.12}$$

dove la seconda somma è su tutto il database.

## 1.3 Backpropagation, come una rete neurale impara

Le reti neurali per modificare i propri pesi e bias in modo da azzerare la funzione di costo seguono una ricerca a gradiente, grazie a quest'ultima si identifica il vettore delle variazioni dei pesi e dei bias $-\eta\nabla C$ e grazie alle equazioni (1.10) e (1.11) si possono calcolare i nuovi pesi e bias che dovrebbero portare ad un valore minore della funzione di costo. Ripetendo questo processo si continua a diminuire il valore della funzione di costo fino, almeno in teoria, ad azzerarla.

Per eseguire la ricerca a gradiente è però necessario calcolare quest'ultimo, ovvero bisogna calcolare le derivate della funzione di costo $\frac{\partial C}{\partial w_k}$ e $\frac{\partial C}{\partial b_l}$. Per il calcolo di quest'ultime derivate è necessario applicare la regola della catena in quanto la funzione di costo C è "fortemente" una funzione composta, bisogna quindi scrivere un algoritmo che grazie alla regola della catena sia in grado di ottenere le derivate dei pesi e bias relativi ad ogni layer della rete. Si vedrà quindi come impostare un algoritmo di backproagation, il nome deriva dal fatto che si inizierà il computo delle derivate partendo dall'ultimo layer (output layer) e si tornerà indietro layer per layer lungo la rete ottenendo di volta in volta le derivate dell'ultimo layer raggiunto.

### 1.3.1 Procedura operativa

Si inizierà introducendo l'equazioni relative alla backpropagation e definendo la notazione:

$$\frac{\partial C}{\partial \mathbf{z}^{[L]}} = \frac{\partial C}{\partial \mathbf{a}^{[L]}} \odot g^{[L]'}(\mathbf{z}^{[L]}) \tag{1.13}$$

$$\frac{\partial C}{\partial \mathbf{z}^{[l-1]}} = [\mathbf{W}^{[l]}]^{\mathbf{T}} \cdot \frac{\partial C}{\partial \mathbf{z}^{[l]}} \odot g^{[l-1]'}(\mathbf{z}^{[l-1]}) \tag{1.14}$$

$$\frac{\partial C}{\partial \mathbf{W}^{[l]}} = \frac{\partial C}{\partial \mathbf{z}^{[l]}} \cdot \left[\mathbf{a}^{[l-1]}\right]^{\mathbf{T}} \tag{1.15}$$

$$\frac{\partial C}{\partial \mathbf{b}^{[l-1]}} = \frac{\partial C}{\partial \mathbf{z}^{[l-1]}} \tag{1.16}$$

L'apice tra parentesi quadre indica il layer a cui si riferisce la grandezza, il numero totale di layer è $L$, quindi come si può notare la prima equazione è riferita interamente all'ultimo layer ovvero l'output layer. Il vettore $\mathbf{a}$ contiene i valori neuronali in un determinato layer. La generica funzione di attivazione scelta è indicata con $g$, $g^{'}$ è la sua derivata e $z$ il suo argomento. $\mathbf{W}^{[l]}$ è la matrice dei pesi in ingresso in un dato layer, quindi $w_{i,j}^{[l]}$ sarà il peso per la connessione dall'i-esimo neurone del layer $[l-1]$ al j-esimo neurone nel layer scelto $[l]$, ecco un immagine per rendere più chiaro il concetto.

Per quanto riguarda i bias si ha il vettore $\mathbf{b}$, in quanto ogni neurone ha un solo bias. Il simbolo $\odot$ indica il prodotto di Hadamard, ovvero il prodotto elemento per elemento.

L'equazione (1.13) serve per effettuare il primo passo, infatti l'obbiettivo è quello di calcolare il gradiente rispetto ai pesi e bias, ovvero di calcolare per ogni layer l'equazione (1.15) e (1.16). L'operazione si esegue ottenendo il gradiente rispetto all'argomento delle funzioni di attivazione tramite la (1.14). Al primo passo però la (1.14) non è utilizzabile

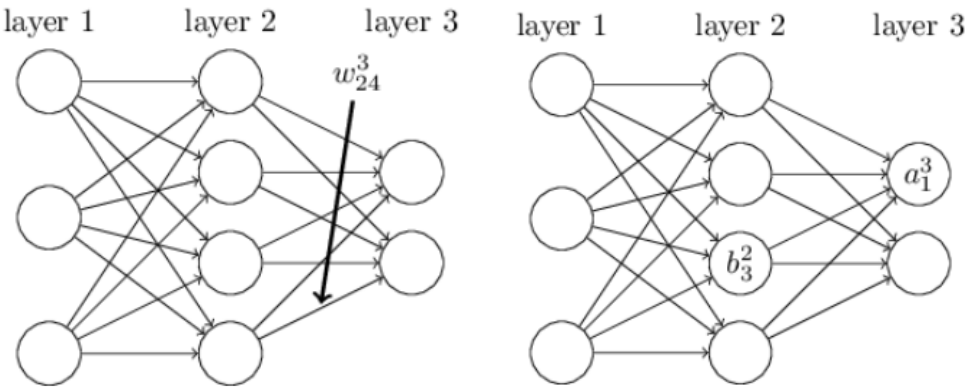


Figura 1.7: Ogni connessione neuronale presenta un peso, ed ogni neurone possiede un bias.[24]

in quanto non esiste il layer $[L+1]$, ma ottenere le derivate $\frac{\partial C}{\partial \mathbf{z}^{[L]}}$ risulta immediato in quanto la funzione di costo $C$ dipende esplicitamente dai neuroni dell'output layer e quindi risulta immediato calcolare $\frac{\partial C}{\partial \mathbf{a}^{[L]}}$. Infatti se si utilizza come funzione di costo il MSE: $C_x = \frac{\|\mathbf{y}(\mathbf{x})-\mathbf{a}^{[\mathbf{L}]}\|^2}{2} = \frac{1}{2}\sum_{n^{[L]}}(y_i - a_i^{[L]})^2$ la sua derivata rispetto ad $\mathbf{a}^{[L]}$ sarà semplicemente $\frac{\partial C}{\partial \mathbf{a}^{[L]}} = \{(a_1 - y_1), (a_2 - y_2), \cdots, (a_{n^{[L]}} - y_{n^{[L]}})\}$, $n^{[L]}$ è il numero di neuroni nell'output layer.

Quindi riassumendo la procedura operativa per effettuare la backpropagation consiste nelle seguenti mosse:

1. Usare eq. (1.13) e calcolare $\frac{\partial C}{\partial \mathbf{z}^{[L]}}$
2. Ottenere $\frac{\partial C}{\partial \mathbf{W}^{[L]}}$ e $\frac{\partial C}{\partial \mathbf{b}^{[L]}}$ dalla eq. (1.15) e (1.16)
3. Calcolare $\frac{\partial C}{\partial \mathbf{z}^{[L-1]}}$ dalla eq. (1.14)
4. Ripetere la procedura per tutti layer fino all'input layer

Una volta giunti all'input layer si avrà concluso la procedura e si avranno calcolate tutte le derivate rispetto ai pesi e bias. Da notare che questa procedura va eseguita per ogni training input nel mini-batch, una volta finito si prenderà il valor medio per ogni derivata e solo dopo si andranno a correggere i pesi e i bias per poi iniziare una nuova epoch.

### 1.3.2 Dimostrazione dell'equazioni di backpropagation (1.13) e (1.14)[19]

Per effettuare la backpropagation si inizia nell'ultimo layer L e si procede all'indietro un layer alla volta. Per ogni layer viene calcolato il vettore degli errori:

$$\frac{\partial C}{\partial \mathbf{z}^{[L]}}, \quad \frac{\partial C}{\partial \mathbf{z}^{[L-1]}}, \quad \frac{\partial C}{\partial \mathbf{z}^{[L-2]}}, \quad etc. \tag{1.17}$$

La derivata della funzione di costo rispetto all'argomento della funzione di attivazione di un neurone è detto errore, infatti se la derivata ha un valore elevato vuol dire che per piccole variazioni dell'input pesato si hanno grandi variazioni sulla funzione di costo, invece se il valore è piccolo, al limite zero, vuol dire che indipendentemente dall'input, quando questo viene pesato, crea variazioni trascurabili nella funzione di costo. Il neurone quindi se l'errore è piccolo si trova in una situazione di stabilità e vedrà variare di poco i valori dei propri pesi e del proprio bias.

Si assuma ora di essere arrivati al generico layer $l$ e di voler scrivere l'errore per il layer $l-1$. La funzione di costo dipende dell'input pesato $\mathbf{z}$ del layer stesso:

$$C(z_1{}^{[l]}, z_2{}^{[l]}, ..., z^{[l]}_{n^{[l]}}) \tag{1.18}$$

Questa è l'equazione di cui vorremmo calcolare le derivate rispetto a layer $l-1$, gli input pesati $z_k^{[l]}$ sono delle funzioni dipendenti dai valori neuronali del layer precedente $(l-1)$, in formule:

$$z_k^{[l]} = z_k^{[l]}(a_1^{[l-1]}, a_2^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}}) \tag{1.19}$$

questo risulta evidente se andiamo a scrivere l'input pesato $z_k$ per il layer $l$:

$$z_k^{[l]} = w_{k,1}^{[l]} \cdot a_1^{[l-1]} + w_{k,2}^{[l]} \cdot a_2^{[l-1]} + ... + w^{[l]}_{k,n^{[l-1]}} \cdot a^{[l-1]}_{n^{[l-1]}} + b_k^{[l]} \quad \forall k \in \{1, ..., n^{[l]}\} \tag{1.20}$$

quindi sostituendo nella (1.18) possiamo esplicitare la dipendenza dai valori neuronali:

$$C(a_1^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}}) = C\left(z_1^{[l]}(a_1^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}}), ..., z^{[l]}_{n^{[l]}}(a_1^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}})\right) \tag{1.21}$$

Applicando ora la regola della catena andiamo a scrivere le derivate della funzione di costo rispetto ai valori neuronali del layer precedente:

$$\frac{\partial C(a_1^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}})}{\partial a_i^{[l-1]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C(z_1^{[l]}, ..., z^{[l]}_{n^{[l]}})}{\partial z_k^{[l]}} \cdot \frac{\partial z_k^{[l]}(a_1^{[l-1]}, ..., a^{[l-1]}_{n^{[l-1]}})}{\partial a_i^{[l-1]}} \tag{1.22}$$

il primo termine della sommatoria è noto in quanto è l'errore del layer $l$ e si suppone di averlo già calcolato. Il secondo termine può essere valutato analizzando la sua espressione:

$$\frac{\partial z_k^{[l]}}{\partial a_i^{[l-1]}} = w_{k,1}^{[l]} \cdot \frac{\partial a_1^{[l-1]}}{\partial a_i^{[l-1]}} + ... + w_{k,i}^{[l]} \cdot \frac{\partial a_i^{[l-1]}}{\partial a_i^{[l-1]}} + ... + w^{[l]}_{k,n^{[l-1]}} \cdot \frac{\partial a^{[l-1]}_{n^{[l-1]}}}{\partial a_i^{[l-1]}} + \frac{\partial b_j^{[k]}}{\partial a_i^{[l-1]}} = w_{k,i}^{[l]} \tag{1.23}$$

Siccome il valore di un neurone non dipende dal valore dei neuroni appartenenti al proprio layer, l'unica derivata diversa da zero è $\frac{\partial a_i^{[l-1]}}{\partial a_i^{[l-1]}} = 1$, quindi possiamo scrivere la (1.22) come:

$$\frac{\partial C}{\partial a_i^{[l-1]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C}{\partial z^{[l]}} \cdot \frac{\partial z_k^{[l]}}{\partial a_i^{[l-1]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C}{\partial z_k^{[l]}} \cdot w_{k,i}^{[l]}, \quad \forall i \in \{1, \dots, n^{[l-1]}\} \tag{1.24}$$

per chiarezza di notazione sono stati eliminati gli argomenti delle funzioni. Passiamo ora alla notazione matriciale:

$$\begin{pmatrix} \frac{\partial C}{\partial a_1^{[l-1]}} \\ \frac{\partial C}{\partial a_2^{[l-1]}} \\ \vdots \\ \frac{\partial C}{\partial a_{n^{[l-1]}}^{[l-1]}} \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n^{[l]}} w_{k,1}^{[l]} \cdot \frac{\partial C}{\partial z_k^{[l]}} \\ \sum_{k=1}^{n^{[l]}} w_{k,2}^{[l]} \cdot \frac{\partial C}{\partial z_k^{[l]}} \\ \vdots \\ \sum_{k=1}^{n^{[l]}} w_{k,n^{[l-1]}}^{[l]} \cdot \frac{\partial C}{\partial z_k^{[l]}} \end{pmatrix} = \begin{pmatrix} w_{1,1}^{[l]} & w_{2,1}^{[l]} & \cdots & w_{n^{[l]},1}^{[l]} \\ w_{1,2}^{[l]} & w_{2,2}^{[l]} & \cdots & w_{n^{[l]},2}^{[l]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n^{[l-1]}}^{[l]} & w_{2,n^{[l-1]}}^{[l]} & \cdots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial C}{\partial z_1^{[l]}} \\ \frac{\partial C}{\partial z_2^{[l]}} \\ \vdots \\ \frac{\partial C}{\partial z_{n^{[l]}}^{[l-1]}} \end{pmatrix} \tag{1.25}$$

come si può notare le sommatorie sono state sostituite dal prodotto matriciale. In forma compatta risulta:

$$\frac{\partial C}{\partial \mathbf{a}^{[l-1]}} = [\mathbf{W}^{[l]}]^{\mathbf{T}} \cdot \frac{\partial C}{\partial \mathbf{z}^{[l]}} \tag{1.26}$$

$$\frac{\partial C}{\partial \mathbf{a}^{[l-1]}} \in \mathbb{R}^{n^{[l-1]}}, \quad \mathbf{W}^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}, \quad \frac{\partial C}{\partial \mathbf{z}^{[l]}} \in \mathbb{R}^{n^{[l]}}$$

quello che è stato fatto è propagare all'indietro l'errore del layer $l$ attraverso il vettore dei bias e la matrice dei pesi, arrivando cosi all'output del layer $l-1$. Adesso si dovrà continuare la propagazione per giungere all'errore del layer $l-1$, questo si traduce in sostanza nell'apllicazione ulteriore della regola della catena. Come è già stato presentato la funzione di costo dipende dai valori neuronali del layer $l-1$:

$$C\big(a_1{}^{[l-1]}, ..., a_{n^{[l-1]}}^{[l-1]}\big) \tag{1.27}$$

ma i valori neuronali del layer $l-1$ sono funzioni che dipendono dagli input pesati tramite la funzione di attivazione $g$:

$$a_i^{[l-1]} = g^{[l-1]}\big(z_i^{[l-1]}\big), \quad \forall i \in \{1, \cdots, n^{[l-1]}\} \tag{1.28}$$

quindi sostituendo la (1.28) nella (1.27) possiamo esplicitare la dipendenza della funzione di costo dalle funzioni di attivazione di ciascun neurone nel layer $l-1$

$$C(z_1{}^{[l-1]}, \cdots, z_{n^{[l-1]}}^{[l-1]}) = C\big(g^{[l-1]}\big(z_1^{[l-1]}\big), \cdots, g^{[l-1]}\big(z_{n^{[l-1]}}^{[l-1]}\big)\big) \tag{1.29}$$

ora è possibile applicare la regola della catena e valutare le derivate.

$$\frac{\partial C(z_1{}^{[l-1]}, \cdots, z_{n^{[l-1]}}^{[l-1]})}{z_i^{[l-1]}} = \sum_{k=1}^{n^{[l-1]}} \frac{\partial C\big(a_1{}^{[l-1]}, ..., a_{n^{[l-1]}}^{[l-1]}\big)}{\partial a_k^{[l-1]}} \cdot \frac{\partial g^{[l-1]}(z_k^{[l-1]})}{\partial z_i^{[l-1]}} \tag{1.30}$$

Il primo termine della sommatoria è già noto dalla equazione (1.26), il secondo termine è la derivata della funzione di attivazione rispetto al proprio argomento $g^{'}$. La funzione di attivazione del neurone $k-esimo$ in un dato layer non dipende dai valori degli input pesati degli altri neuroni.

$$\frac{\partial g^{[l-1]}(z_k^{[l-1]})}{\partial z_i^{[l-1]}} = \begin{cases} g^{[l-1]'}(z_k^{[l-1]}), & \text{if } k = i \\ 0 & \text{else} \end{cases} \tag{1.31}$$

Quindi la (1.30) può essere semplificata nelle seguente forma:

$$\frac{\partial C}{\partial z_i^{[l-1]}} = \sum_{k=1}^{n^{[l-1]}} \frac{\partial C}{\partial a_k^{[l-1]}} \cdot \frac{\partial g^{[l-1]}(z_k^{[l-1]})}{\partial z_i^{[l-1]}} = \frac{\partial C}{\partial a_i^{[l-1]}} \cdot g^{[l-1]'}(z_i^{[l-1]}) \tag{1.32}$$

Di nuovo non sono stati riportati gli argomenti per chiarezza, riscriviamo ora l'equazione in forma matriciale. A questo scopo verrà introdotto il prodotto di hadamard $\odot$, l'operazione prende due vettori e li moltiplica elemento per elemento $[(A_{ij} \odot B_{ij}) = (A_{ij})(B_{ij})]$.

$$\begin{pmatrix} \frac{\partial C}{\partial z_1^{[l-1]}} \\ \frac{\partial C}{\partial z_2^{[l-1]}} \\ \vdots \\ \frac{\partial C}{\partial z_{n^{[l-1]}}^{[l-1]}} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a_1^{[l-1]}} \cdot g^{[l-1]'}(z_1^{[l-1]}) \\ \frac{\partial C}{\partial a_2^{[l-1]}} \cdot g^{[l-1]'}(z_2^{[l-1]}) \\ \vdots \\ \frac{\partial C}{\partial a_{n^{[l-1]}\,[l-1]}} \cdot g^{[l-1]'}(z_{n^{[l-1]}}^{[l-1]}) \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a_1^{[l-1]}} \\ \frac{\partial C}{\partial a_2^{[l-1]}} \\ \vdots \\ \frac{\partial C}{\partial a_{n^{[l-1]}}^{[l-1]}} \end{pmatrix} \odot \begin{pmatrix} g^{[l-1]'}(z_1^{[l-1]}) \\ g^{[l-1]'}(z_2^{[l-1]}) \\ \vdots \\ g^{[l-1]'}(z_{n^{[l-1]}}^{[l-1]}) \end{pmatrix} \tag{1.33}$$

Infine riscriviamo la (1.23) in notazione vettoriale per compattezza:

$$\frac{\partial C}{\partial \mathbf{z}^{[l-1]}} = \frac{\partial C}{\partial \mathbf{a}^{[l-1]}} \odot g^{[l-1]'}(\mathbf{z}^{[l-1]}) \tag{1.34}$$

$$\frac{\partial C}{\partial \mathbf{z}^{[l-1]}} \in \mathbb{R}^{n^{[l-1]}}, \quad \frac{\partial C}{\partial \mathbf{a}^{[l-1]}} \in \mathbb{R}^{n^{[l-1]}}, \quad g^{'[l-1]}(\mathbf{z}^{[l-1]}) \in \mathbb{R}^{n^{[l-1]}}$$

l'equazione (1.34) è fondamentale per eseguire il primo passo partendo dall'output layer $L$, infatti come precedentemente scritto il termine $\frac{\partial C}{\partial \mathbf{a}^{[l-1]}}$ risulta esplicito per il layer $L$. Per i layer interni (hidden layer) invece bisognerà utilizzare l'equazione (1.26),

$$\frac{\partial C}{\partial \mathbf{z}^{[l-1]}} = [\mathbf{W}^{[l]}]^{\mathbf{T}} \cdot \frac{\partial C}{\partial \mathbf{z}^{[l]}} \odot g^{[l-1]'}(\mathbf{z}^{[l-1]}) \tag{1.35}$$

in questo modo si completa la dimostrazione per le due prime equazioni della backpropagation.

### 1.3.3 Dimostrazione delle equazioni di backpropagation (1.15) e (1.16)[19]

Iniziamo andando a calcolare il gradiente di $C$ rispetto ai pesi, il procedimento è analogo a quello già visto. Scriviamo la dipendenza della funzione di costo dagli input pesati:

$$C(z_1{}^{[l]}, z_2{}^{[l]}, ..., z^{[l]}_{n^{[l]}}) \tag{1.36}$$

a questo punto andiamo a esplicitare di nuovo l'espressione dell'input pesato, in modo tale da mettere in evidenza la dipendenza dai pesi e dal bias.

$$z^{[l]}_k = w^{[l]}_{k,1} \cdot a^{[l-1]}_1 + w^{[l]}_{k,2} \cdot a^{[l-1]}_2 + ... + w^{[l]}_{k,n^{[l-1]}} \cdot a^{[l-1]}_{n^{[l-1]}} + b^{[l]}_k \quad \forall k \in \{1, ..., n^{[l]}\} \tag{1.37}$$

Avendo messo in evidenza la dipendenza dai pesi possiamo scrivere l'input pesato come funzione della matrice dei pesi $\mathbf{W}$:

$$z^{[l]}_k = z^{[l]}_k(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}}) \tag{1.38}$$

Ora è possibile scrivere l'equazione (1.36) esplicitando la dipendenza dai pesi:

$$C(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}}) = \tag{1.39}$$

$$C\Big(z^{[l]}_1(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}}), \cdots, z^{[l]}_{n^{[l]}}(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}})\Big)$$

avendo esplicitato le dipendenze è possibile applicare la regola della catena alla (1.39).

$$\frac{\partial C(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}})}{\partial w^{[l]}_{i,j}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C(z^{[l]}_1, \cdots, z^{[l]}_{n^{[l]}})}{\partial z^{[l]}_k} \cdot \frac{\partial z^{[l]}_k(w^{[l]}_{1,1}, w^{[l]}_{1,2}, \cdots, w^{[l]}_{n^{[l]},n^{[l-1]}})}{\partial w^{[l]}_{i,j}} \tag{1.40}$$

Il primo termine della sommatoria risulta essere sempre l'errore nel layer $l$ che si suppone noto, il secondo termine può essere valutato in questo modo:

$$\frac{\partial z^{[l]}_k}{\partial w^{[l]}_{i,j}} = \begin{cases} a^{[l-1]}_j, & \text{if } k = i \\ 0, & \text{else} \end{cases} \tag{1.41}$$

per dimostrare questa relazione si scriverà in notazione matriciale il vettore degli input pesati:

$$\begin{pmatrix} z_1^{[l]} \\ z_2^{[l]} \\ \vdots \\ z_{n^{[l]}}^{[l]} \end{pmatrix} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} = \begin{pmatrix} w_{1,1}^{[l]} & w_{1,2}^{[l]} & \cdots & w_{1,n^{[l-1]}}^{[l]} \\ w_{2,1}^{[l]} & w_{2,2}^{[l]} & \cdots & w_{2,n^{[l-1]}}^{[l]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{[l]},1}^{[l]} & w_{n^{[l]},2}^{[l]} & \cdots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{pmatrix} \cdot \begin{pmatrix} a_1^{[l-1]} \\ a_2^{[l-1]} \\ \vdots \\ a_{n^{[l-1]}}^{[l-1]} \end{pmatrix} + \mathbf{b}^{[l]} \quad (1.42)$$

come si può notare ogni input pesato dipende solo da una riga della matrice dei pesi $\mathbf{W}$, quindi la derivata rispetto a pesi di altre righe risulta nulla, infatti se esaminiamo la derivata di $z_2$ rispetto ad un peso appartenente alla 3 riga, quindi riguardante la connessione entrante nel nodo 3 del layer $l$, otterremo 0. Invece rispetto ad un peso della stessa riga otterremo il valore del neurone del layer $l-1$ da cui parte la connessione a cui fa riferimento il peso stesso.

$$\frac{\partial z_2^{[l]}}{\partial w_{3,2}^{[l]}} = \frac{\partial w_{2,1}^{[l]}}{\partial w_{3,2}^{[l]}} \cdot a_1^{[l-1]} + \frac{\partial w_{2,2}^{[l]}}{\partial w_{3,2}^{[l]}} \cdot a_2^{[l-1]} + \cdots + \frac{\partial w_{2,n^{[l-1]}}^{[l]}}{\partial w_{3,2}^{[l]}} \cdot a_{n^{[l-1]}}^{[l-1]} + \frac{\partial b_2^{[l]}}{\partial w_{3,2}^{[l]}} = 0 \quad (1.43)$$

$$\frac{\partial z_2^{[l]}}{\partial w_{2,2}^{[l]}} = \frac{\partial w_{2,1}^{[l]}}{\partial w_{2,2}^{[l]}} \cdot a_1^{[l-1]} + \frac{\partial w_{2,2}^{[l]}}{\partial w_{2,2}^{[l]}} \cdot a_2^{[l-1]} + \cdots + \frac{\partial w_{2,n^{[l-1]}}^{[l]}}{\partial w_{2,2}^{[l]}} \cdot a_{n^{[l-1]}}^{[l-1]} + \frac{\partial b_2^{[l]}}{\partial w_{2,2}^{[l]}} = a_2^{[l-1]} \quad (1.44)$$

L'equazione (1.43) risulta nulla, infatti il valore di un peso nel layer $l$ e indipendente dal valore dei pesi nello stesso layer. Invece nella equazione (1.44) abbiamo anche la derivata del peso rispetto a se stesso (perché siamo nella stessa riga) che è di valore unitario. Quindi la derivata della funzione $C$ rispetto ai pesi in un dato layer assume la forma:

$$\frac{\partial C(w_{1,1}^{[l]}, w_{1,2}^{[l]}, \cdots, w_{n^{[l]},n^{[l-1]}}^{[l]})}{\partial w_{i,j}^{[l]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C}{\partial z_k^{[l]}} \cdot \frac{\partial z_k^{[l]}}{\partial w_{i,j}^{[l]}} = \frac{\partial C}{\partial z_i^{[l]}} \cdot a_j^{[l-1]} \quad (1.45)$$

passando alla forma matriciale:

$$\frac{\partial C}{\partial \mathbf{W}^{[l]}} = \begin{pmatrix} \frac{\partial C}{\partial z_1^{[l]}} \cdot a_1^{[l-1]}, & \frac{\partial C}{\partial z_1^{[l]}} \cdot a_2^{[l-1]}, & \cdots & \frac{\partial C}{\partial z_1^{[l]}} \cdot a_{n^{[l-1]}}^{[l-1]} \\ \frac{\partial C}{\partial z_2^{[l]}} \cdot a_1^{[l-1]}, & \frac{\partial C}{\partial z_2^{[l]}} \cdot a_2^{[l-1]}, & \cdots & \frac{\partial C}{\partial z_2^{[l]}} \cdot a_{n^{[l-1]}}^{[l-1]} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial z_{n^{[l]}}^{[l]}} \cdot a_1^{[l-1]}, & \frac{\partial C}{\partial z_{n^{[l]}}^{[l]}} \cdot a_2^{[l-1]}, & \cdots & \frac{\partial C}{\partial z_{n^{[l]}}^{[l]}} \cdot a_{n^{[l-1]}}^{[l-1]} \end{pmatrix} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}} \quad (1.46)$$

infine riscrivendo la (1.46) con la notazione matriciale otteniamo proprio la (1.15).

$$\frac{\partial C}{\partial \mathbf{W}^{[l]}} = \frac{\partial C}{\partial \mathbf{z}^{[l]}} \cdot \left[\mathbf{a}^{[l-1]}\right]^{\mathbf{T}} \quad (1.47)$$

$$\frac{\partial C}{\partial \mathbf{W}^{[l]}} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}, \quad \frac{\partial C}{\partial \mathbf{z}^{[l]}} \in \mathbb{R}^{n^{[l]}}, \quad \mathbf{a}^{[l-1]} \in \mathbb{R}^{n^{[l-1]}}$$

Passiamo ora al calcolo del gradiente rispetto ai bias, i passaggi sono identici a quelli appena eseguiti ma in questo caso le funzioni $z_k$ sono considerate in funzione del vettore dei bias $\mathbf{b}$:

$$z_i^{[l]} = z_i^{[l]}\big(b_1^{[l]}, b_2^{[l]}, \cdots, b_{n^{[l]}}^{[l]}\big) \tag{1.48}$$

viene applicata direttamente la regola della catena, andando a scrivere l'equazione per la derivata di $C$ rispetto ai bias.

$$\frac{\partial C\big(b_1^{[l]}, b_2^{[l]}, \cdots, b_{n^{[l]}}^{[l]}\big)}{\partial b_i^{[l]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C(z_1^{[l]}, \cdots, z_{n^{[l]}}^{[l]})}{\partial z_k^{[l]}} \cdot \frac{\partial z_k^{[l]}\big(b_1^{[l]}, b_2^{[l]}, \cdots, b_{n^{[l]}}^{[l]}\big)}{\partial b_i^{[l]}} \tag{1.49}$$

Il primo termine della sommatoria è sempre noto, il secondo termine si valuta facilmente considerando che $z_k$ dipenderà da un solo valore del vettore $\mathbf{b}$, infatti:

$$\begin{pmatrix} z_1^{[l]} \\ z_2^{[l]} \\ \vdots \\ z_{n^{[l]}}^{[l]} \end{pmatrix} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \begin{pmatrix} b_1^{[l]} \\ b_2[l] \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{pmatrix} \tag{1.50}$$

questo porta che la derivata di $z_k$ rispetto ad un bias del layer $l$ sarà:

$$\frac{\partial z_k^{[l]}}{\partial b_i^{[l]}} = \begin{cases} 1, & \text{if } k = 1 \\ 0, & \text{else} \end{cases} \tag{1.51}$$

quindi possiamo infine riscrivere la (1.49).

$$\frac{\partial C\big(b_1^{[l]}, b_2^{[l]}, \cdots, b_{n^{[l]}}^{[l]}\big)}{\partial b_i^{[l]}} = \sum_{k=1}^{n^{[l]}} \frac{\partial C}{\partial z_k^{[l]}} \cdot \frac{\partial z_k^{[l]}}{\partial b_i^{[l]}} = \frac{\partial C}{\partial z_i^{[l]}} \tag{1.52}$$

Riscrivendo la (1.52) in notazione vettoriale si ottiene l'equazione (1.16):

$$\frac{\partial C}{\partial \mathbf{b}^{[l-1]}} = \frac{\partial C}{\partial \mathbf{z}^{[l-1]}} \tag{1.53}$$

$$\frac{\partial C}{\partial \mathbf{b}^{[l-1]}} \in \mathbb{R}^{n^{[l-1]}}, \quad \frac{\partial C}{\partial \mathbf{z}^{[l-1]}} \in \mathbb{R}^{n^{[l-1]}}$$

questo conclude la dimostrazione delle equazioni di backpropagation.

# Capitolo 2

# PINN Physics Informed Neural Network

In questo capitolo verrà introdotto il concetto di physics informed neural network. Come verrà presentato, una scelta oculata della funzione di costo $C$ permette di trasformare una rete neurale in un risolutore di PDE. L'approccio risolutivo può essere ricondotto al metodo di Galerkin, modificato tramite un approccio basato sul machine-learning. Il metodo di Galerkin utilizza una combinazione lineare costituita da funzioni di base per approssimare la soluzione della PDE da risolvere, invece nell'approccio proposto la soluzione verrà approssimata tramite una rete neurale. La rete PINN viene addestrata per soddisfare l'operatore differenziale, le condizioni iniziali e al contorno usando la ricerca a gradiente stocastica effettuata su punti campionati casualmente nel dominio di calcolo.

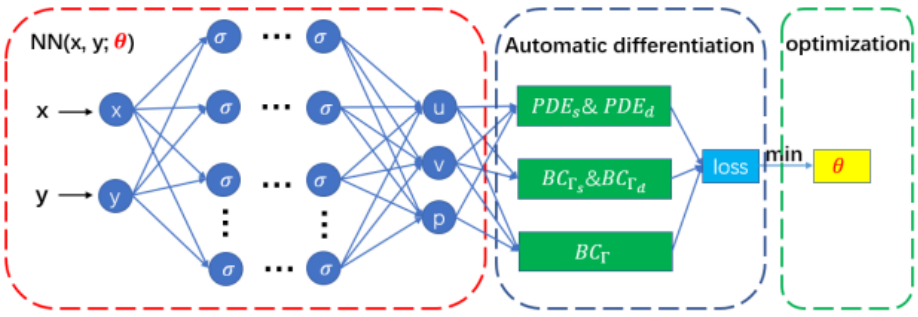


Figura 2.1: Rappresentazione schematica di una rete neurale PINN per la risoluzione delle equazioni di Navier-Stokes.[9]

La figura (2.1) mostra il diagramma di una rete neurale di tipo PINN; il layer di

input contiene i valori delle coordinate spaziali e di quella temporale, il layer di output contiene invece i valori delle grandezze incognite della PDE. I valori ottenuti dalla rete sono necessari per il calcolo della funzione di costo (loss), grazie alla quale tramite backpropagation si può procedere alla ottimizzazione dei parametri della rete (pesi e bias). Con il termine differenziazione automatica si intende una tecnica di implementazione della backprogation che viene utilizzata sia per determinare le derivate rispetto ai pesi e bias ma anche rispetto all'input. L'algoritmo di differenziazione automatica si può immaginare come un estensione dell'algoritmo basato sulla regola della catena presentato nel capitolo precedente, esso introduce alcuni vantaggi di utilizzo della memoria e versatilità nel calcolo del gradiente e di grandezze da esso dipendenti (come il gradiente del gradiente)[26]

## 2.1 Approssimazione tramite rete neurale per la risoluzione di PDE

Si consideri la seguente PDE, in cui $\mathcal{L}$ è l'operatore differenziale spaziale:

$$\begin{cases} \partial_t u(t,x) + \mathcal{L}u(t,x) = 0, \quad (t,x) \in [0,T] \times \Omega \\ u(0,x) = u_0(x), \quad x \in \Omega \\ u(t,x) = g(t,x), \quad x \in [0,T] \times \partial\Omega \end{cases} \tag{2.1}$$

dove $\partial\Omega$ è il bordo di $\Omega$. La soluzione u(t,x) è incognita, ma una sua approssimazione può essere calcolata minimizzando la norma $L^2$ dell'errore (sinonimo di funzione di costo).

$$J(f) = \|\partial_t f + \mathcal{L}f\|^2_{2,[0,T]\times\Omega} + \|f - g\|^2_{2,[0,T]\times\partial\Omega} + \|f(0,\cdot) - u_0\|^2_{2,\Omega} \tag{2.2}$$

La funzione dell'errore $J(f)$ misura l'accuratezza della funzione approssimante $f$ nel soddisfare l'equazione alle derivate parziali, la condizioni al contorno e la condizione iniziale. Risulta importate notare come non sia necessaria nessuna conoscenza a priori della soluzione $u$, ponendo di fatto questo metodo nella categoria dei solutori di PDE; $J(f)$ può essere direttamente calcolata dalla (2.1) per qualsiasi approssimazione f. L'obbiettivo è costruire delle funzioni $f$ per le quali la funzione $J(f)$ sia il più possibile vicino a zero. Viene definito $\mathfrak{C}^n$ come lo spazio delle reti neurali con un solo hidden layer e $n$ unità neuronali. Una rete neurale con questa architettura è una funzione del

tipo $\mathfrak{C}^n = \{h(t,x) : \mathbb{R}^{1+d} \mapsto \mathbb{R} : h(t,x) = \sum_{i=1}^{n} \beta_1 \Psi(\alpha_{1,i} t + \sum_{j=1}^{d} \alpha_{j,i} x_j + c_j)\}$ dove $\Psi : \mathbb{R} \mapsto \mathbb{R}$ è una funzione non lineare, come il sigmoide ad esempio.

Sia $f^n$ una rete neurale con $n$ unità neuronali che minimizza la funzione $J(f)$. Allora è possibile dimostrare [27], sotto certe condizioni, che:

$$\begin{cases} \text{esiste } f^n \text{ tale per cui } J(f^n) \to 0, \text{ se } n \to \infty, \text{ inoltre} \\ f^n \to u \text{ se } n \to \infty \end{cases} \tag{2.3}$$

$f^n$ converge fortemente in $L^\rho([0,T] \times \Omega)$, con $\rho < 2$, per una classe di PDE paraboliche quasi-lineari (le reti neurali PINN sperimentalmente riescono a risolvere categorie di PDE più generali, ad ogni modo una dimostrazione analitica non è ancora disponibile). La dimostrazione richiede l'analisi combinata della capacità di approssimazione delle reti neurali insieme alle proprietà di continuità delle PDE. Attenzione, dire che $J(f) \to 0$ per $n \to \infty$ non implica necessariamente che $f^n \to u$. Si andrà quindi ad imporre che ogni rete neurale $\{f^n\}_{n=1}^{\infty}$ soddisfi la PDE con un termine sorgente $h^n(t,x)$, solo a questo punto sarà possibile, sotto determinate condizioni, provare che $f^n \to u$ se $n \to \infty$ in $L^\rho([0,T] \times \Omega)$, per $\rho < 2$, usando la liscezza delle funzioni approssimanti e la compattezza degli argomenti[27].

### 2.1.1 Procedura operativa

Si consideri la seguente PDE con $d$ dimensioni spaziali:

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} + \mathcal{L}u(t,x) = 0, \quad (t,x) \in [0,T] \times \Omega \\ u(0,x) = u_0(x), \quad x \in \Omega \\ u(t,x) = g(t,x), \quad x \in [0,T] \times \partial\Omega \end{cases} \tag{2.4}$$

dove $x \in \Omega \subset \mathbb{R}^d$. La rete PINN approssima u(t,x) con il proprio output $f(t,x;\theta)$, dove $\theta \in \mathbb{R}^k$ sono i parametri della rete neurale. Si noti che gli operatori differenziali $\frac{\partial f}{\partial t}(t,x;\theta)$ e $\mathcal{L}f(t,x;\theta)$ possono essere calcolati analiticamente. Costruiamo ora la funzione di costo $J(f)$ nella seguente maniera.

$$J(f) = \|\frac{\partial f}{\partial t}(t,x;\theta) + \mathcal{L}f(t,x;\theta)\|^2_{2,[0,T]\times\Omega,\nu_1} + \|f(t,x;\theta) - g(t,x)\|^2_{2,[0,T]\times\partial\Omega,\nu_2} + \|f(0,x;\theta) - u_0(x)\|^2_{2,\Omega,\nu_3} \tag{2.5}$$

Con il simbolismo $\|f(y)\|_{\Upsilon,\nu}$ si intende $\int_\nu |f(y)|^2 \nu(y) dy$, dove $\nu(y)$ è la densità di probabilità di $y \in \Upsilon$. Se $J(f) = 0$ allora $f(t,x;\theta)$ è soluzione della PDE (2.4).

L'obbiettivo è quello di trovare un set di parametri $\theta$ tali per cui la funzione $f(t, x\theta)$ minimizzi l'errore $J(f)$. Se l'errore $J(f)$ è piccolo, allora $f(t, x; \theta)$ soddisferà con un piccolo scarto: gli operatori differenziali della PDE, le condizioni al contorno e la condizione iniziale. Inoltre un set $\theta$ che minimizza la funzione $J(f(.; \theta))$ produce un output $f(t, x; \theta)$ che approssima la soluzione della PDE $u(t, x)$. Stimare il valore dei parametri $\theta$ non è praticabile per via diretta quando la dimensione d è grande, dal momento che il costo computazionale di integrare lungo $\Omega$ sarebbe insostenibile. Quindi verrà adotta la SGD (discesa a gradiente stocastica) applicata ad una sequenza di punti spazio-temporali campionati casualmente sul dominio $\Omega$ e sul suo bordo $\partial\Omega$. L'algoritmo risulta quindi:

1. Generare casualmente un set di punti $(t_n, x_n)$ da $[0, T] \times \Omega$ e $(\tau_n, z_n)$ da $[0, T] \times \partial\Omega$ seguendo le densità di probabilità $\nu_1$ e $\nu_2$. Generare, inoltre, un set $w_n$ da $\Omega$ secondo la densità di probabilità $\nu_3$

2. Calcolare l'errore quadratico $G(\theta_n, s_n)$ processo i punti campionati $s_n = \{(t_n, x_n), (\tau_n, z_n), w_n\}$ dove:
   $G(\theta_n, s_n) = J(f) = \Big(\frac{\partial f}{\partial t}(t_n, x_n; \theta_) + \mathcal{L}f(t_n, x_n; \theta_n)\Big)^2 + \Big(f(\tau_n, z_n; \theta_n) - g(\tau_n, z_n)\Big)^2 + \Big(f(0, w_n; \theta_n) - u_0(w_n)\Big)^2$

3. Effettuare un passo della discesa al generico punto $s_n$:
   $\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta G(\theta_n, s_n)$

4. Ripetere la procedura fino a che il criterio di convergenza è soddisfatto.

Il learning rate $\alpha_n$ diminuisce con $n$. Il legame tra $\nabla_\theta G(\theta_n, s_n)$ e $\nabla_\theta J(\cdot; \theta_n)$ è il seguente mostrato. Ovvero risulta che $\nabla_\theta G(\theta_n, s_n)$ sia la stima imparziale di $\nabla_\theta J(\cdot; \theta_n)$.

$$\mathbb{E}[\nabla_\theta G(\theta_n, s_n)|\theta_n] = \nabla_\theta J(f(\cdot; \theta_n))$$

**Stimatore imparziale**

Uno stimatore di un dato parametro è definito imparziale (unbiased) se il valore atteso è equivalente al vero valore del parametro. In altre parole, una stimatore risulta imparziale se produce valori attesi che risultano corretti in media. Uno stimatore $\hat{\gamma(\xi)}$ è definito imparziale se e solo se $\mathbb{E}[\hat{\gamma(\xi)}] = \gamma_0$, dove il valore atteso è calcolato rispetto alla distribuzione di probabilità del campione $\xi$ (sample).

La SGD prevede che venga effettuata una operazione di media su tutti i passi ottenuti da un mini-batch. Quindi per essere più precisi bisogna dividere $s_n$ in mini batch, effettuare la procedura sopra elencata per tutti i punti in un batch (mini batch) e poi prende un effettivo passo di discesa. In questo modo si avrà che $J(\cdot;\theta_{n+1}) < J(f(\cdot;\theta_n))$.

Rispettando alcune (relativamente deboli) condizioni [28] questo algoritmo convergerà verso un punto critico della funzione dell'errore $J(\cdot;\theta_n)$ se $n \to \infty$:

$$\lim_{n\to\infty} \|\nabla_\theta J(f(\cdot;\theta_n))\| = 0$$

è importante notare che il set $\theta_n$ potrebbe far convergere soltanto ad un minimo locale quando $f(t,x;\theta)$ è una funzione non convessa. Questo fatto è generale per qualsiasi problema di ottimizzazione con funzioni non convesse, non risulta un problema di questo algoritmo specifico. Inoltre è ben noto che la SGD porta solo ad un minimo locale nell'addestramento delle reti neurali. Nonostante questi fatti, la SGD si è rivelata molto efficace nella pratica e risulta essere un elemento fondamentale nell'approccio all'addestramento di qualsiasi modello di deep learning.

## 2.2 Esempio applicativo: Equazione di Burgers 1D

In questa sezione verrà presentato un esempio applicativo 1D per la risoluzione dell'equazione di Burgers, verranno poi presentanti e discussi i risultati ottenuti da Sirignano e Spilioposulus.[27]

La soluzione di diverse PDE su un range di diversi problemi (e.g., differenti condizioni fisiche e differenti condizioni al contorno) è sicuramente un problema di interesse ingegneristico. In generale quindi il problema da risolvere potrebbe avere un'alta dimensionalità, questo comporta un alto costo computazionale.

Sia $p$ la variabile che rappresenta il setup del problema (e.g. condizioni fisiche, condizioni al contorno e iniziali). La variabile $p$ prende valori dallo spazio $\mathcal{P}$ e si vuole trovare la soluzione della PDE $u(t,x;p)$. In particolare, si supponga che $u(t,x;p)$ soddisfi la PDE seguente.

$$\begin{cases} \dfrac{\partial u(t,x;p)}{\partial t} + \nu \dfrac{\partial^2 u(t,x;p)}{\partial x^2} - \alpha\, u(t,x;p)\, \dfrac{\partial u(t,x;p)}{\partial x} = 0, \quad (t,x) \in [0,T] \times \Omega \\ u(t=0,x;p) = h_p(x), \quad x \in \Omega \\ u(t,x;p) = g_p(t,x), \quad x \in [0,T] \times \partial\Omega \end{cases} \tag{2.6}$$

Un approccio "classico" passerebbe attraverso la discretizzazione dello spazio $\mathcal{P}$ e la soluzione iterativa della PDE in diversi punti $p$. Ad ogni modo, il numero totale di punti (o celle) della griglia (e anche il numero di PDEs da risolvere) cresce esponenzialmente con il numero di dimensioni, in applicazioni ingegneristiche la spazio $\mathcal{P}$ può avere facilmente un grande numero di dimensioni.

### 2.2.1 Procedura operativa

La rete neurale verrà addestrata seguendo la procedura operativa precedentemente mostrata (riportata in un formato più condensato), in più avremo anche il campionamento dei punti $p$ rappresentanti il setup del problema scelto.

1. Inizializzare i parametri della rete $\theta$
2. Generare casualmente un set di punti $(t, x, p)$ da $[0, T] \times \Omega \times \mathcal{P}$ e un set $(\hat{t}, \hat{x})$ da $[0, T]$ e un set di coordinate $\tilde{x}$ da $\partial\Omega$
3. Costruire la funzione dell'errore, con $\mathcal{L}_p = \nu \frac{\partial^2 f}{\partial x^2} - \alpha\, f\, \frac{\partial f}{\partial x}$
   $J(\theta) = \Big(\frac{\partial f}{\partial t}(t, x, p; \theta) - \mathcal{L}_p f(t, x, p; \theta)\Big)^2 + \Big(g_p(\hat{x}) - f(\hat{t}, \hat{x}, p; \theta)\Big)^2 + \Big(h_p(\tilde{x}) - f(0, \tilde{x}, p; \theta)\Big)^2$
4. Aggiornare i parametri $\theta$ della rete tramite un passo di SGD

   $$\theta \to \theta - \alpha \nabla_\theta J(\theta)$$

   dove $\alpha$ è il learning rate.

### 2.2.2 Equazione di Burgers 1D risultati

L'equazione risolta, con le condizioni al contorno e iniziali scelte è riportata:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nu \dfrac{\partial^2 u}{\partial x^2} - \alpha u \dfrac{\partial u}{\partial x}, \quad (t, x) \in [0, 1] \times [0, 1] \\ u(t, x = 0) = a, \\ u(t, x = 1) = b, \\ u(t = 0, x) = g(x), \quad x \in [0, 1] \end{cases} \tag{2.7}$$

lo spazio dei parametri è $\mathcal{P} = (\nu, \alpha, a, b) \in \mathbb{R}^4$. La condizione iniziale $g(x)$ è la retta che passa per le condizioni al contorno $u(t, x = 0) = a$ e $u(t, x = 1) = b$. La rete è stata addestrata per approssimare la soluzione $u(t, x; p)$ sull'intero spazio $(t, x, \nu, \alpha, a, b) \in$

$[0,1] \times [0,1] \times [10^{-2}, 10^{-1}] \times [10^{-2}, 1] \times [-1,1] \times [-1,1]$. La rete utilizzata possiede 6 layers con 200 unità neuronali ciascuno. I risultati riportati in figura (2.2) mostrano la soluzione predetta per diversi valori delle variabili di setup $p$ confrontate con la soluzione esatta, il confronto rende quasi indistinguibili le due curve. In figura (2.3) viene riportato invece un confronto per diversi valori di $t$ e $\nu$.

La soluzione esatta è riportata in rosso, la soluzione predetta dalla rete in blu. Le soluzioni sono riportate per $t = 1$. Le curve sono quasi completamente sovrapposte, in alcuni casi indistinguibili.I parametri scelti dallo spazio $\mathcal{P}$, in senso antiorario, sono $(\nu, \alpha, a, b) = (0.01, 0.95, 0.9, -0.9), (0.02, 0.95, 0.9, -0.9), (0.01, 0.95, -0.95, 0.95),$ $(0.02, 0.9, 0.9, 0.8), (0.01, 0.75, 0.9, 0.1)$ e $(0.09, 0.95, 0.5, -0.5)$.

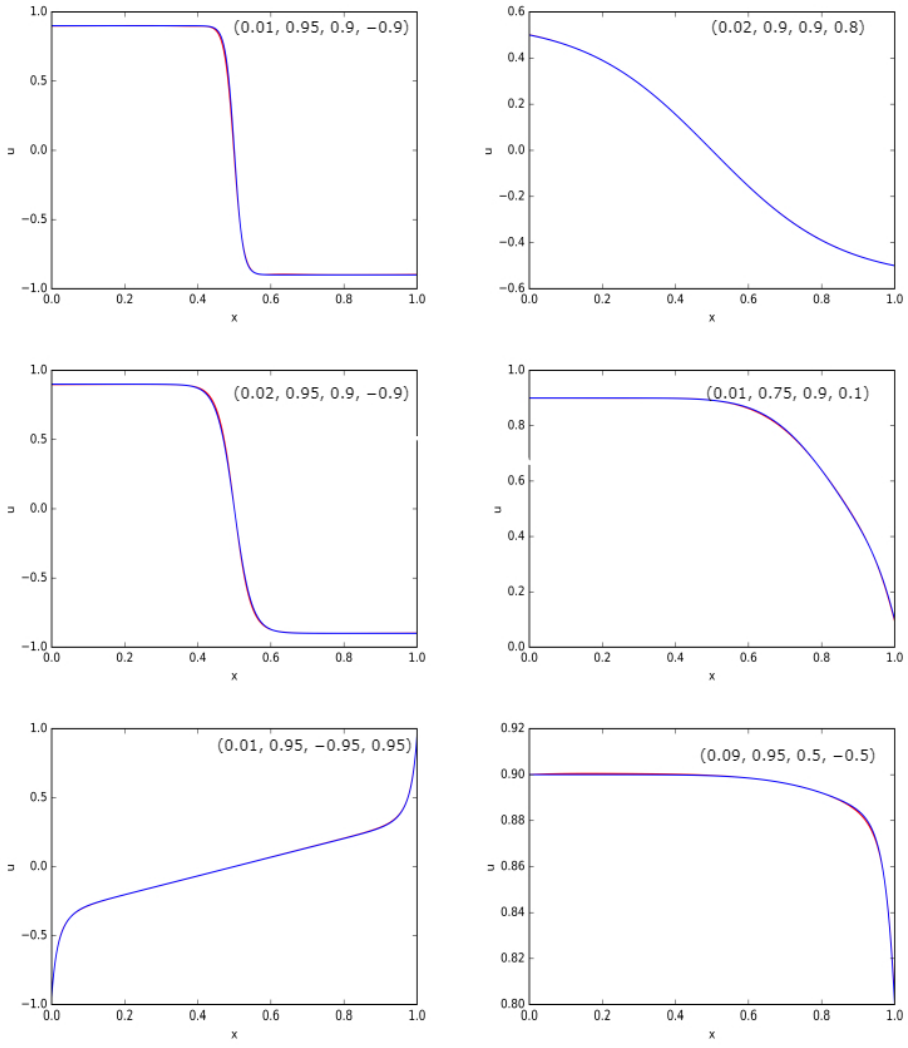


Figura 2.2: Confronto risultati equazione di Burgers per diversi valori di $p$[27]

**Grafico a sinistra**: Confronto della soluzione $t = 0.1, 0.25, 0.5, 1$ per $(\nu, \alpha, a, b) = (0.03, 0.9, 0.95, -0.95)$. **Grafico a destra**: Confronto della soluzione per $\nu = 0.01, 0.02, 0.05, 0.09$ al tempo $t = 1$ e con $(\alpha, a, b) = (0.8, 0.75, -0.75)$.

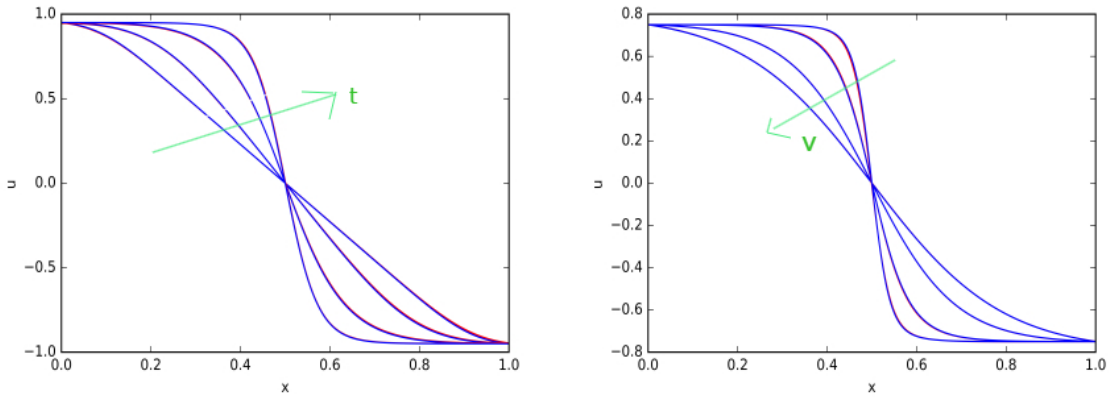


Figura 2.3: Soluzione per l'equazione di Burgers a diversi $t$ e $\nu$[27]

# Capitolo 3

# Applicazione di una rete PINN su geometrie 2D

In questo capitolo verrà testata una rete PINN per la risoluzione delle equazioni di Navier-Stokes (N-S), nel caso stazionario e incomprimibile. Nel capitolo precedente è stata presentata la procedura per risolvere una PDE, in questo capitolo verrà presentata l'estensione ad il sistema di PDE delle N-S.

## 3.1 Risoluzione delle equazioni di Navier-Stokes tramite PINN

L'estensione di una rete PINN per la risoluzione di sistemi di equazioni differenziali è immediata, infatti è semplicemente necessario scegliere adeguatamente i termini della loss. Supponiamo di voler risolvere il seguente sistema:

$$\begin{cases} \nabla \cdot \mathbf{V} = 0 \quad (x,y) \in \Omega \\ \mathbf{V} \cdot \nabla \mathbf{V} = -\nabla p + \nu(\nabla^2 \mathbf{V}) \quad (x,y) \in \Omega \\ \mathbf{V} = \mathbf{V}_\Gamma \quad (x,y) \in \Gamma_u \in \partial\Omega \\ p = p_\Gamma \quad (x,y) \in \Gamma_p \in \partial\Omega \end{cases} \tag{3.1}$$

con $\Gamma_u$ si indica il bordo dove imporre la condizione al contorno sulla velocità, $\Gamma_p$ è invece il bordo dove viene imposta la condizione al contorno sulla pressione.

I termini della funzione di costo (loss) saranno quindi le due equazioni più l'imposizione delle condizioni al contorno:

$$J(\theta) = (\nabla \cdot \mathbf{V} - 0)^2 + (\mathbf{V} \cdot \nabla \mathbf{V} + \nabla p - \nu(\nabla^2 \mathbf{V}) - 0)^2 + (\mathbf{V} - \mathbf{V}_\Gamma)^2 + (p - p_\Gamma)^2 \tag{3.2}$$

In verità le equazioni sono trattate nella loro forma scalare, in quanto la rete prevederà le componenti di velocità e la pressione come riportato nel diagramma sottostante.

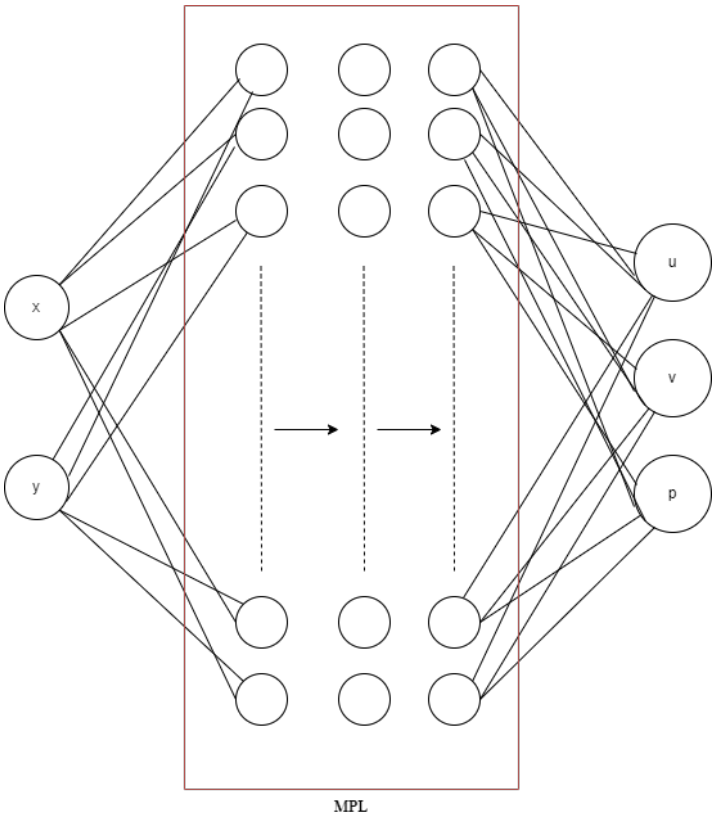


Figura 3.1: Rappresentazione schematica di una rete neurale per la risoluzione delle equazioni di Navier-Stokes 2D nel caso instazionario e incomprimibile

Per effettuare l'addestramento della rete è necessario creare il set di punti sul dominio $\Omega$ e sul contorno $\partial\Omega$, a questo scopo si può effettuare un campionamento probabilistico del dominio e delle condizioni al contorno, nel caso 1D, per esempio, sono state utilizzate delle distribuzioni random di numeri tra due estremi. Trattando geometrie bidimensionali complesse è invece preferibile utilizzare metodi di campionamento più sofisticati come il Latin Hypercube Sampling (LHS) o le sequenze di Sobol. Come si può notare l'utilizzo di una rete PINN non comporta la necessità di costruire una mesh ma è sufficiente fornire una distribuzione omogenea di punti sul dominio di calcolo. L'estensione a geometrie tridimensionali complesse risulta quindi più immediata rispetto all'utilizzo di risolutore classico, infatti già la geometria costruita tramite software CAD è sufficiente per ottenere la nuvola di punti su cui effettuare l'addestramento. In questa

tesi il campionamento è stato implementato tramite le librerie presenti nel framework Modulus che verrà presentato successivamente. Una volta ottenuti i punti costituenti il dataset della rete, si procede alla divisione in batch (mini-batch) dei punti campionati nel dominio e sul suo bordo. In genere il dataset risulta essere molto grande, almeno qualche milioni di punti, perciò l'utilizzo di batch di dimensione opportuna è necessario.

La funzione di loss (3.2) viene quindi divisa in 4 termini diversi, un termine per la continuità un termine per la quantità di moto lungo x, un termine per la quantità di moto lungo y e un termine per le condizioni al contorno.

$$J(\theta) = (J_{continuità} + J_{q.d.m.x} + J_{q.d.m.y})_{\Omega} + (J_{bc})_{\partial\Omega} \tag{3.3}$$

L'obbiettivo durante il training è riuscire ad azzerare la funzione di loss totale, ad ogni modo i termini che compongono $J(\theta)$ hanno un diverso comportamento, non è detto che avere una loss globale piccola comporti ad aver minimizzato tutti i termini che la compongono. A questo proposito si possono introdurre degli ulteriori parametri all'interno della rete che vanno a pesare ogni singolo contributo all'interno della funzione di loss.

$$J(\theta) = (\alpha J_{continuità} + \beta J_{q.d.m.x} + \gamma J_{q.d.m.y})_{\Omega} + (\sigma J_{bc})_{\partial\Omega} \tag{3.4}$$

A seconda del valore adottato si darà importanza ad un termine della loss globale rispetto ad un altro, scegliere per esempio un valore grande di $\alpha$ porterà la rete a trovare un set di pesi e bia $\theta$ che maggiormente minimizza il termine riguardante la continuità. La scelta di questi parametri avviene su base empirica, osservando quali termini durante il training tendono ad essere poco ottimizzati. I pesi della loss globale fanno sempre parte degli hyperparameters della rete.

## 3.2 Nvidia Modulus

Per la sperimentazione e la creazione della rete neurale è stato usato il framework Nvidia Modulus[21]. Modulus è un framework creato per il physics machine learning, esso include diverse api (application programming interface) per la creazione di rete neurali di diversa tipologia e architettura. Inoltre Modulus si interfaccia nativamente con le più utilizzate librerie per il machine learning come Pytorch, facilitando la programmazione.

### 3.2.1 Importare la geometria

All'interno del framework è possibile gestire sia la creazione interna di geometrie tramite constructive solid geometry (CSG) che l'import esterno tramite file STL.

Modulus fornisce diverse primitive (1D, 2D e 3D) che vengono utilizzate per la creazione delle nuvole di punti necessarie per il training della rete. Le geometrie primitive supportano le operazioni booleane come l'unione, la sottrazione e l'intersezione. L'implementazione è molto intuitiva, una volta chiamata la primitiva basterà sommarla o sottrarla ad un'altra primitiva per ottenere una geometria più complessa. Ecco un esempio per la creazione di uno dei test case utilizzati per la sperimentazione, nella figura sopra viene riportata la geometria che si vuole creare e sotto il codice per generarla.

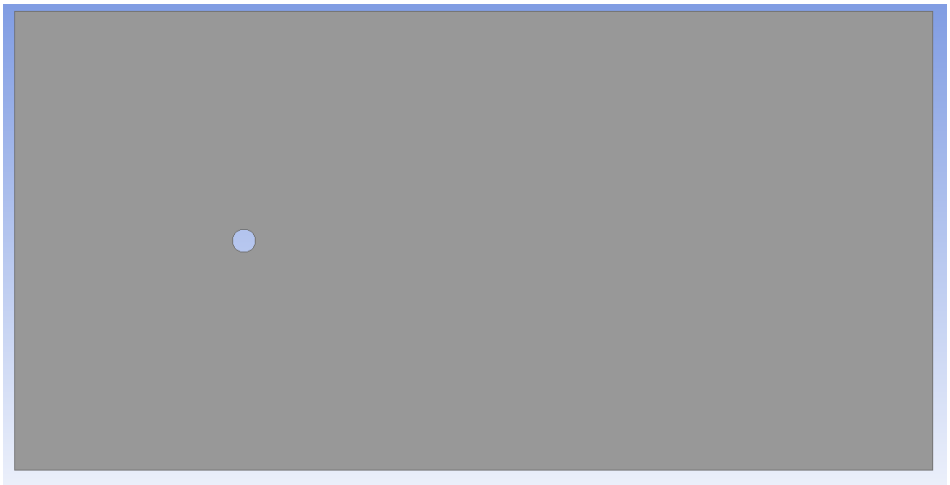

Figura 3.2: Geometria generata, lunghezza = 40m, altezza = 20m, diametro = 1m

```
# geometry
    channel_length = (quantity(-10, "m"), quantity(30, "m"))
    channel_width = (quantity(-10, "m"), quantity(10, "m"))
    cylinder_center = (quantity(0, "m"), quantity(0, "m"))
    cylinder_radius = quantity(0.5, "m")
    channel_length_nd = tuple(map(lambda x: nd.ndim(x), channel_length))
    channel_width_nd = tuple(map(lambda x: nd.ndim(x), channel_width))
    cylinder_center_nd = tuple(map(lambda x: nd.ndim(x), cylinder_center))
    cylinder_radius_nd = nd.ndim(cylinder_radius)

    channel = Channel2D(
        (channel_length_nd[0], channel_width_nd[0]),
        (channel_length_nd[1], channel_width_nd[1]),
    )
```

```
    inlet = Line(
        (channel_length_nd[0], channel_width_nd[0]),
        (channel_length_nd[0], channel_width_nd[1]),
        normal=1,
    )
    outlet = Line(
        (channel_length_nd[1], channel_width_nd[0]),
        (channel_length_nd[1], channel_width_nd[1]),
        normal=1,
    )
    cylinder = Circle(cylinder_center_nd, cylinder_radius_nd)
    volume_geo = channel - cylinder
```

Listing 3.1: CSG per costruire la geometria cilindro 2D in un condotto

Si è voluto simulare un cilindro immerso in una corrente uniforme, quindi si è utilizzata la primitiva "Channel2D" per creare il rettangolo rappresentate la scena. Inoltre sono state utilizzate le primitive "Line" per estrarre i punti per l'inlet e outlet, su queste linee si andrà ad imporre la condizione al contorno di ingresso e uscita. Successivamente con la primitiva "Circle" si è creato un cerchio, tramite l'operazione booleana di sottrazione si è, infine, creata la scena per la simulazione (bucando il dominio rettangolare con il cerchio). La geometria viene infine salvata nella variabile "volume_geo".

Risulta comunque possibile definire una geometria per punti come è stato fatto per il caso del profilo NACA 2412

```
import math
import matplotlib.pyplot as plt
import numpy as np
from sympy import Number, Symbol, Heaviside, atan, sin, cos, sqrt
import os

from modulus.geometry.primitives_2d import Polygon
from modulus.geometry.parameterization import Parameterization, Parameter
from modulus.utils.io.vtk import var_to_polyvtk

#Definizione della linea di camber del profilo
def camber_line(x, m, p, c):
    cl = []
    for xi in x:
        cond_1 = Heaviside(xi, 0) * Heaviside((c * p) - xi, 0)
        cond_2 = Heaviside(-xi, 0) + Heaviside(xi - (c * p), 0)
        v_1 = m * (xi / p ** 2) * (2.0 * p - (xi / c))
        v_2 = m * ((c - xi) / (1 - p) ** 2) * (1.0 + (xi / c) - 2.0 * p)
        cl.append(cond_1 * v_1 + cond_2 * v_2)
    return cl
```

```
21
22 #Computazione della derivata della camber line rispetto ad x e dell'angolo theta per
       il calcol dell spessore
23 def dyc_over_dx(x, m, p, c):
24     dd = []
25     for xi in x:
26         cond_1 = Heaviside(xi) * Heaviside((c * p) - xi)
27         cond_2 = Heaviside(-xi) + Heaviside(xi - (c * p))
28         v_1 = ((2.0 * m) / p ** 2) * (p - xi / c)
29         v_2 = (2.0 * m) / (1 - p ** 2) * (p - xi / c)
30         dd.append(atan(cond_1 * v_1 + cond_2 * v_2))
31     return dd
32
33 #Definizione dello spessore in funzione di x per un profilo simmetrico
34 def thickness(x, t, c):
35     th = []
36     for xi in x:
37         term1 = 0.2969 * (sqrt(xi / c))
38         term2 = -0.1260 * (xi / c)
39         term3 = -0.3516 * (xi / c) ** 2
40         term4 = 0.2843 * (xi / c) ** 3
41         term5 = -0.1015 * (xi / c) ** 4
42         th.append(5 * t * c * (term1 + term2 + term3 + term4 + term5))
43     return th
44
45 #Generazione dei punti del profilo tramite la primitiva "linea" (una linea upper e una
       lower)
46 def naca4(x, m, p, t, c=1):
47     th = dyc_over_dx(x, m, p, c)
48     yt = thickness(x, t, c)
49     yc = camber_line(x, m, p, c)
50     line = []
51     for (xi, thi, yti, yci) in zip(x, th, yt, yc):
52         line.append((xi - yti * sin(thi), yci + yti * cos(thi)))
53     x.reverse()
54     th.reverse()
55     yt.reverse()
56     yc.reverse()
57     for (xi, thi, yti, yci) in zip(x, th, yt, yc):
58         line.append((xi + yti * sin(thi), yci - yti * cos(thi)))
59     return line
60
61
62
63 #Da eseguire solo se nel main (in genere viene chiamata come funzione)
64 if __name__ == "__main__":
65     # make parameters for naca airfoil
66     m = 0.02
```

```
    p = 0.4
    t = 0.12
    c = 1.0

    # make naca geometry
    x = [x for x in np.linspace(0, 0.2, 10)] + [x for x in np.linspace(0.2, 1.0, 10)][
        1:
    ]  # higher res in front
    line = naca4(x, m, p, t, c)[:-1]
    geo = Polygon(line)

    # sample different parameters
    s = geo.sample_boundary(nr_points=100000)
    var_to_polyvtk(s, "naca_boundary")
    s = geo.sample_interior(nr_points=100000)
    var_to_polyvtk(s, "naca_interior")
```

Listing 3.2: Definizione per punti di un profilo alare NACA 2412

Per la creazione del profilo si è seguita la seguente procedura:

1. Si è definita la linea dell spessore per un profilo
   $y_t = 5t[0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4]$, con $t$ lo spessore massimo come frazione della corda.

2. Si è definita la linea di camber come

$$y_c = \begin{cases} \frac{m}{p^2}(2px - x^2), & 0 \leq x \leq p \\ \frac{m}{(1-p)^2}((1-2p) + 2px - x^2), & p \leq x \leq 1 \end{cases}$$

   in cui $m$ è il camber massimo e $p$ è la posizione dove si ha il massimo camber.

3. Si calcola $\theta = \frac{dy_c}{dx}$

4. Si calcolano i punti della linea che definisce il contorno superiore e inferiore del profilo:

$$\begin{cases} x_U = x - y_t \sin\theta \quad y_U = y_c + y_t \cos\theta, \\ x_L = x + y_t \sin\theta \quad y_L = y_c - y_t \cos\theta \end{cases}$$

una volta creati i punti che descrivono il dorso e il ventre del profilo essi sono stati inseriti nella geometria primitiva "Line". Tuttavia, la sola definizione della geometria

primitiva non è sufficiente per la generazioni della nuvola di punti, ma è necessario creare una figura chiusa passando la funzione linea della funzione "polygon". Una volta creata la geometria, essa viene salvata nella variabile "geo". Successivamente si procede al campionamento della geometria, è importante notare che il campionamento in questo caso è eseguito solo sulla superficie del profilo. Infatti il secondo termine (riportato per completezza) serve al campionamento di punti all'interno della geometria e non ha applicazione nel caso analizzato.

### 3.2.2 Imporre le condizioni al contorno

Una volte creata la geometria e dopo averla salvata in una variabile (e.g. volume_geo), si passa alla definizione dei batch size, inoltre è necessario specificare il valore delle condizioni al contorno. Il codice proposto è stato utilizzato per il caso del quadrato immerso in una corrente bidimensionale.

```
1
2     # inlet
3     inlet = PointwiseBoundaryConstraint(
4         nodes=nodes,
5         geometry=inlet,
6         outvar={"u": nd.ndim(inlet_u), "v": nd.ndim(inlet_v)},
7         batch_size=cfg.batch_size.inlet,
8     )
9     domain.add_constraint(inlet, "inlet")
10
11    # outlet
12    outlet = PointwiseBoundaryConstraint(
13        nodes=nodes,
14        geometry=outlet,
15        outvar={"p": nd.ndim(outlet_p)},
16        batch_size=cfg.batch_size.outlet,
17    )
18    domain.add_constraint(outlet, "outlet")
19
20    # full slip (channel walls)
21    walls = PointwiseBoundaryConstraint(
22        nodes=nodes,
23        geometry=channel,
24        outvar={"u": nd.ndim(inlet_u), "v": nd.ndim(inlet_v)},
25        batch_size=cfg.batch_size.walls,
26    )
27    domain.add_constraint(walls, "walls")
28
```

```
    # no slip
    no_slip = PointwiseBoundaryConstraint(
        nodes=nodes,
        geometry=square,
        outvar={"u": nd.ndim(noslip_u), "v": nd.ndim(noslip_v)},
        batch_size=cfg.batch_size.no_slip,
    )
    domain.add_constraint(no_slip, "no_slip")

    # interior contraints
    interior = PointwiseInteriorConstraint(
        nodes=nodes,
        geometry=volume_geo,
        outvar={"continuity": 0, "momentum_x": 0, "momentum_y": 0},
        batch_size=cfg.batch_size.interior,
        bounds=Bounds({x: channel_length_nd, y: channel_width_nd}),
    )
    domain.add_constraint(interior, "interior")
```

Listing 3.3: Definizione delle condizioni al controno per il caso del quadrato

Il primo pezzo di codice di riferisce all'inlet, creato tramite la primitiva "Line", per il campionamento dei punti su questa geometria bisogna specificare il batch size (definito in questo caso tramite file di configurazione) tramite il comando "batch_size". Inoltre sempre nella stessa istanza viene anche imposto il valore da utilizzare per imporre la condizione al contorno. In questo caso si impone che la $u$ e la $v$ siano pari a dei valori prestabiliti "inlet_u" e "inlet_v". Allo stesso modo viene trattata la condizione di pressione nulla all'outlet, la condizione di full slip sul bordo superiore e inferiore del "canale" e la condizione di aderenza sul cilindro. L'ultimo pezzo di codice invece riguarda il campionamento all'interno del dominio, infatti viene data in input la geometria "volume_geo", come si può notare anche in questo caso si specificano dei valori da imporre, viene passato le zero in quanto vogliamo annullare le equazioni. Un valore diverso poteva essere specificato in presenza di forzanti, inoltre il parametro "bounds" viene utilizzato per definire gli estremi dove applicare il sampling e le condizioni espresse, ad esempio in caso si volesse applicare diverse forzanti/sorgenti su diverse zone del dominio.

```
    # physical quantities
    nu = quantity(0.02, "kg/(m*s)")
    rho = quantity(1.0, "kg/m^3")
    inlet_u = quantity(1.0, "m/s")
    inlet_v = quantity(0.0, "m/s")
```

```
    noslip_u = quantity(0.0, "m/s")
    noslip_v = quantity(0.0, "m/s")
    outlet_p = quantity(0.0, "pa")
    velocity_scale = inlet_u
    density_scale = rho
    length_scale = quantity(20, "m")
    nd = NonDimensionalizer(
        length_scale=length_scale,
        time_scale=length_scale / velocity_scale,
        mass_scale=density_scale * (length_scale ** 3),
    )
```

Listing 3.4: Definizione delle variabili del test case scelto

La parte di codice soprastante riporta la definizione delle condizioni al contorno e delle grandezze fisiche utilizzate per la simulazione. Inoltre viene anche definito un adimensionalizzatore, non è necessario adimensionalizzare l'input che riceve la rete ma risulta essere una buona pratica. Adimensionalizzare l'input rispetto alle scali più grandi del problema in questione previene possibili errori di overflow nella rete, evitando che l'input pesato in ingresso alla funzione di costo risulti essere un numero eccedente il massimo numero di macchina esprimibile.

### 3.2.3 Configurazione della rete

La configurazione della rete avviene tramite l'utilizzo del "Hydra configuration framework", il quale offre una gestione immediata dei parametri della rete. Viene utilizzato un file YAML il quale contiene gli hyperparameters della rete. Viene proposto il file di configurazione utilizzato per generare i risultati che verranno riportati successivamente.

```
defaults :
  - modulus_default
  - arch:
      - fully_connected
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_

scheduler:
  decay_rate: 0.95
  decay_steps: 2000

training:
  rec_results_freq : 1000
```

```
  rec_constraint_freq: 10000
  max_steps : 200000

batch_size:
  inlet: 320
  outlet: 320
  walls: 320
  no_slip: 128
  interior: 3200
    )
```

Listing 3.5: File di configurazione YAML

Il file di configurazione Hydra supporta più parametri di quelli riportati in questo file, in modo da poter implementare una qualsiasi rete neurale. Ai fini di questa tesi si è scelto di utilizzare lo standard offerto da modulus, la rete scelta è composta da 6 hidden layer con 512 unità ciascuno, la funzione di attivazione scelta è la tangente iperbolica $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. L'architettura utilizzata è del tipo fully connected (tutti i neuroni vengono collegati), mentre per l'implementazione della back propagation si è utilizzato l'ottimizzatore Adam. Il parametro "loss" invece regola l'aggregazione dei vari termini della loss derivanti da diverse imposizioni, in questo caso è stata scelta la semplice somma.

Nella sezione riguardante lo scheduler invece si imposta il decadimento del learning rate, infatti nel progredire del training ci si aspetta di avvicinarsi sempre di più al minimo (locale) della funzione di loss, perciò saranno necessari aggiustamenti (step) sempre più piccoli. Nel caso mostrato si diminuisce il learning rate del 5% ogni 2000 step completati.

Nella sezione training si imposta la frequenza con cui salvare i risultati e il numero massimo di step.

L'ultima sezione invece riguarda la dimensione dei batch effettuati su ogni geometria definita nella precedente sezione. Da notare la dimensione relativamente piccola adottata per i batch, questo è dovuto ai limiti del hardware utilizzato (Nvidia RTX 3070 8GB). Infatti per ogni epoch vengono campionati 1000 batch per ogni geometria, questo significa che, per esempio, all'interno del dominio vengono campionati 3.200.000 di punti, quindi si necessita di schede video con elevata VRAM. Nvidia consiglia l'utilizzo della scheda A100 da 80GB, ad ogni modo questa GPU ha un costo medio intorno ai 10.000 euro, in questa tesi si è voluto utilizzare un hardware di medio livello in quanto

era di interesse confrontare le performance con un classico calcolatore CFD senza la necessità di dover ricorrere a cluster di GPU. Quindi sia le simulazioni tramite risolutore CFD e tramite rete neurale sono state ottenute con un hardware accessibile a qualsiasi individuo o azienda.

## 3.3 Risultati ed analisi

Verranno ora presentati i risultati ottenuti per il caso del cilindro, del quadrato e del profilo NACA2412. Il problema fluidodinamico è lo stesso in tutti e tre i casi, e il Reynolds è fissato a $Re = 50$. Per il cilindro il diametro è fissato ad 1 metro, lo stesso valore è stato usato per il lato del quadrato e per la corda del profilo. Il dominio è $\Omega - B_r$ con $\Omega = [-10, 30] \times [-10, 10]$ e $B_r$ l'insieme dei punti costituenti il corpo. La densità è posta unitaria $\rho = 1\frac{Kg}{m^3}$ invece la viscosità dinamica è pari ad $\mu = 0,02\frac{Kg}{ms}$. Sul corpo $B_r$ è imposta la condizione di aderenza (velocità nulla). Sull'inlet $\Gamma_3 = -10 \times [-10, 10]$, sul bordo superiore $\Gamma_1 = [-10, 30]$ e sul bordo inferiore $\Gamma_2 = [-10, 30]$ è imposta la condizione $u = 1\frac{m}{s}$. All'outlet $\Gamma_4 = 30 \times [-10, 10]$ è imposta la condizione di pressione nulla $p = 0$.

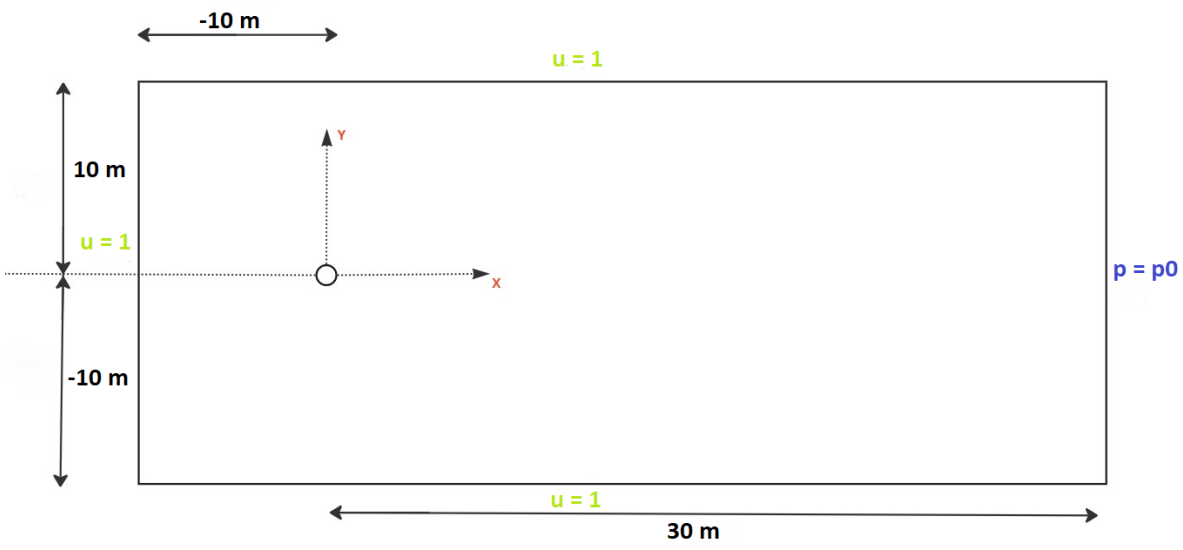


Figura 3.3: Geometria utilizzata nel caso del cilindro

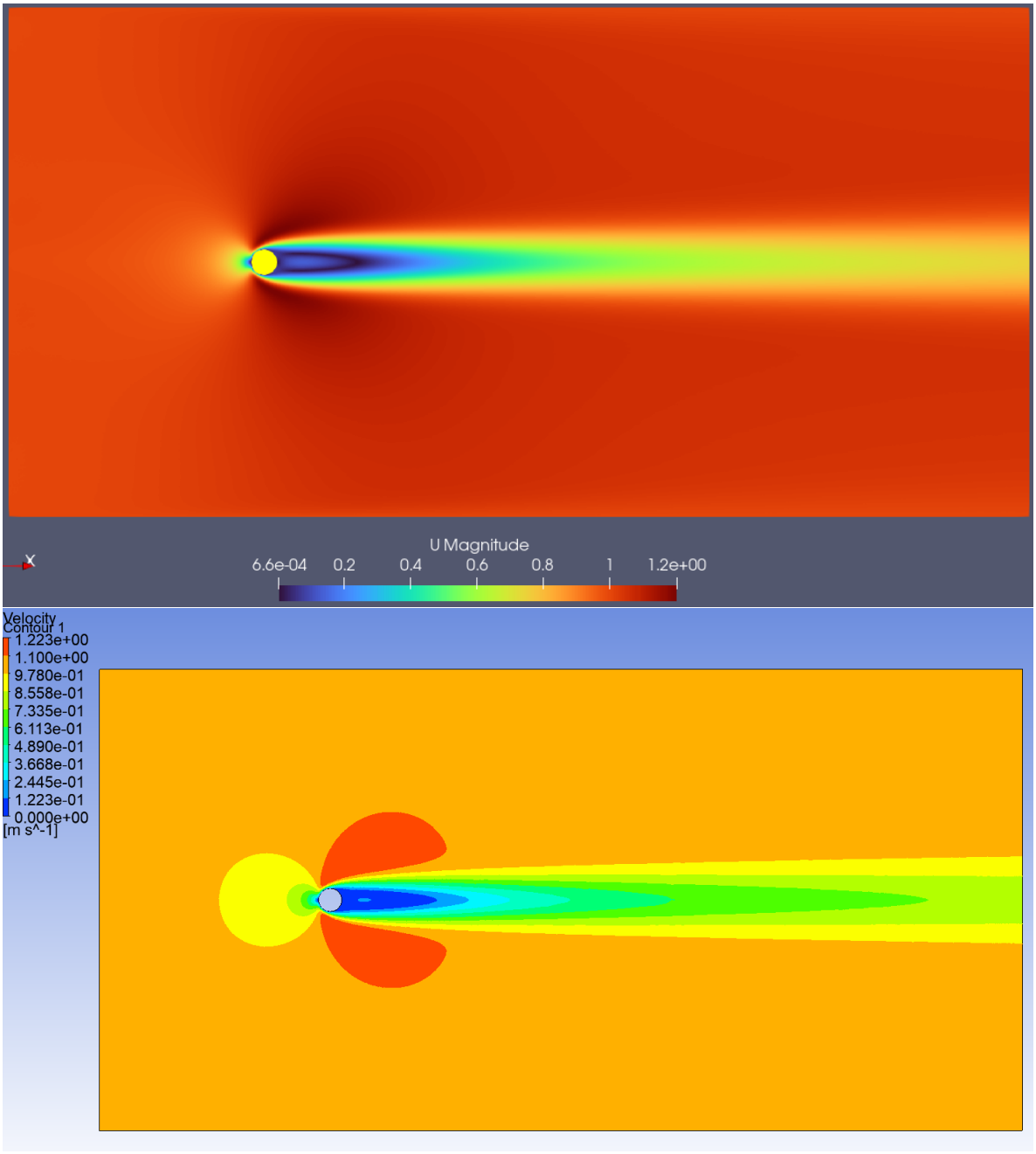


Figura 3.4: Modulo della velocità. Rete Neurale (sopra) Ansys (sotto).

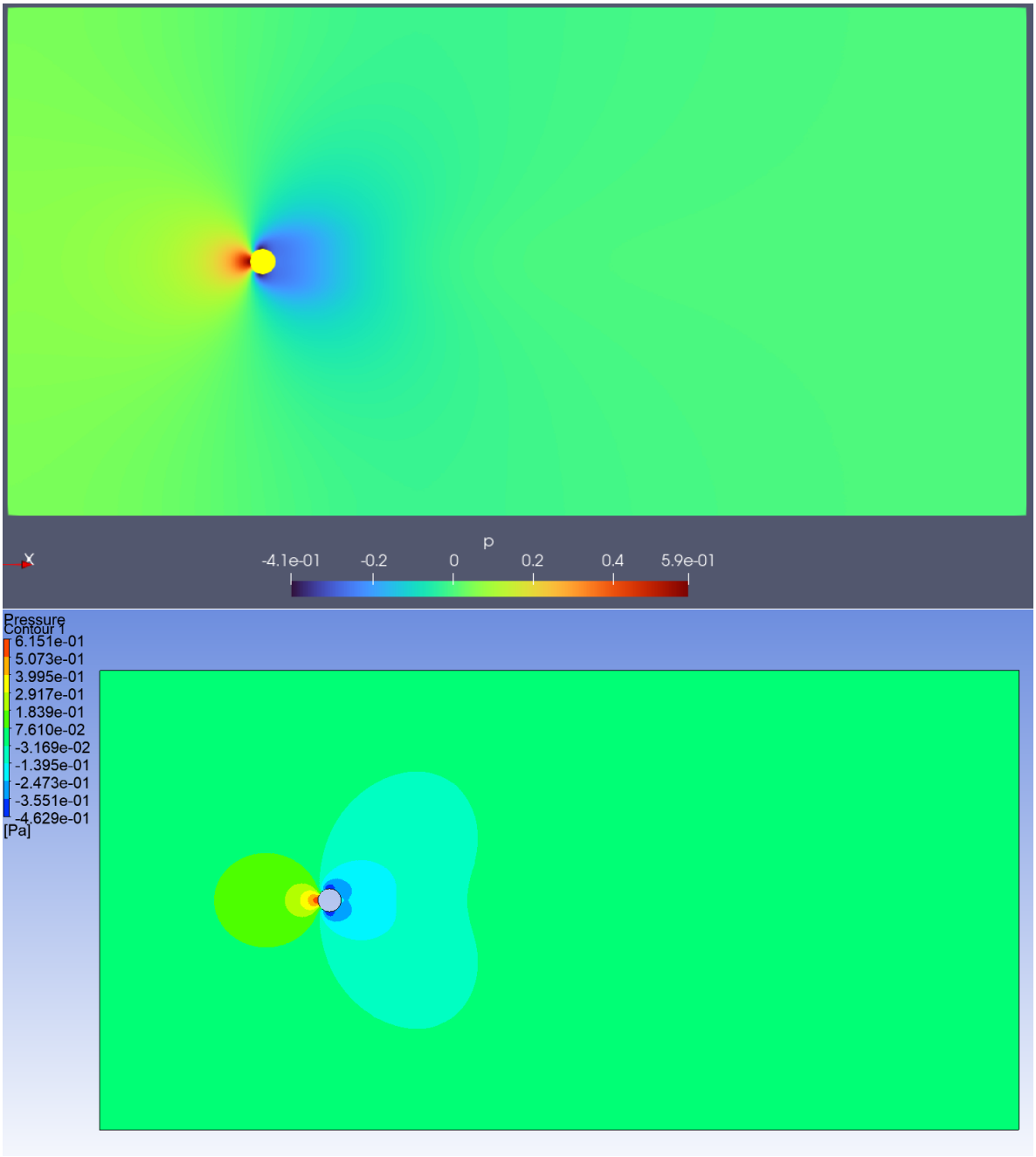


Figura 3.5: Campo di pressione. Rete Neurale (sopra) Ansys (sotto).

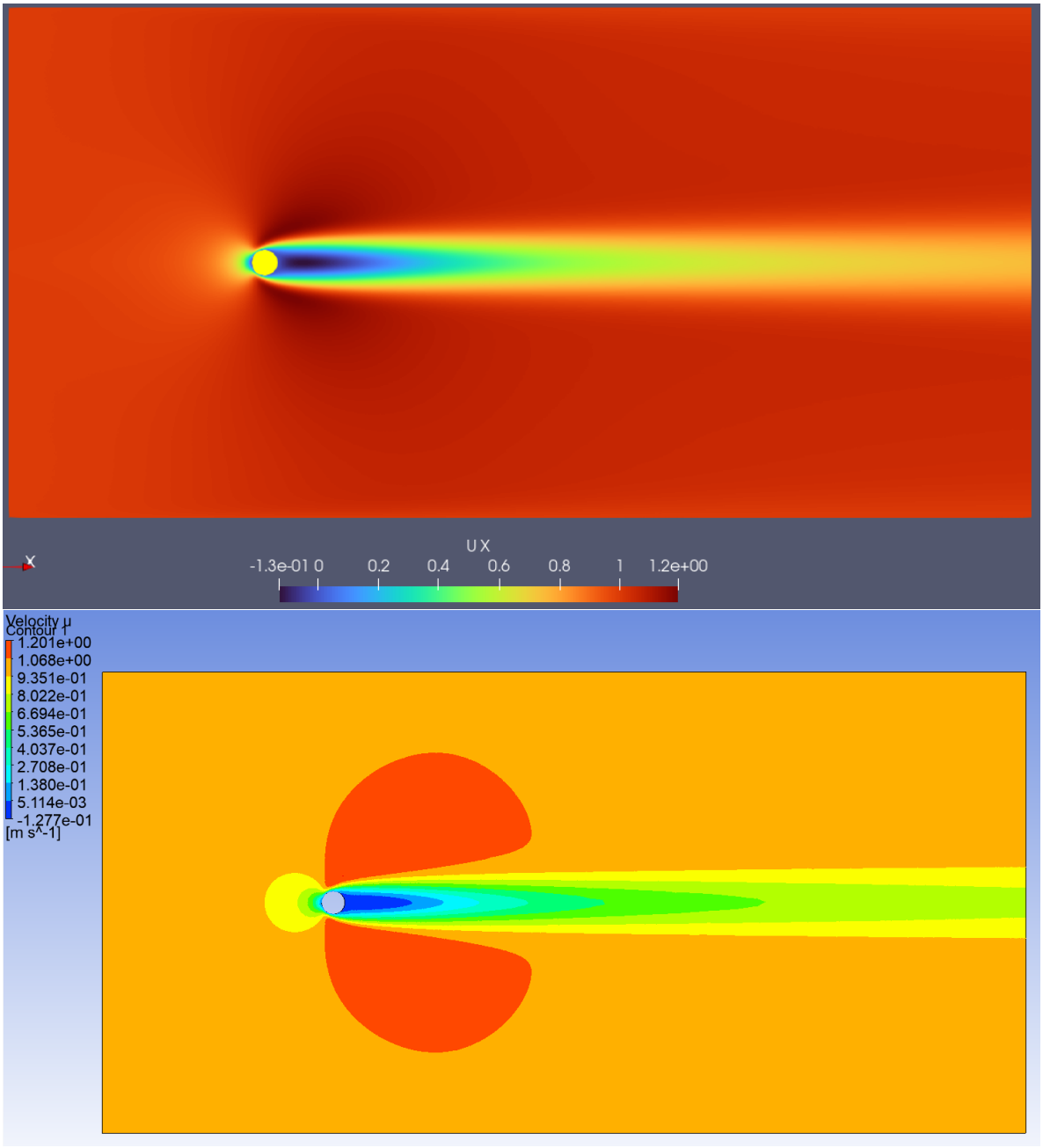


Figura 3.6: Componente orizzontale. Rete Neurale (sopra) Ansys (sotto).

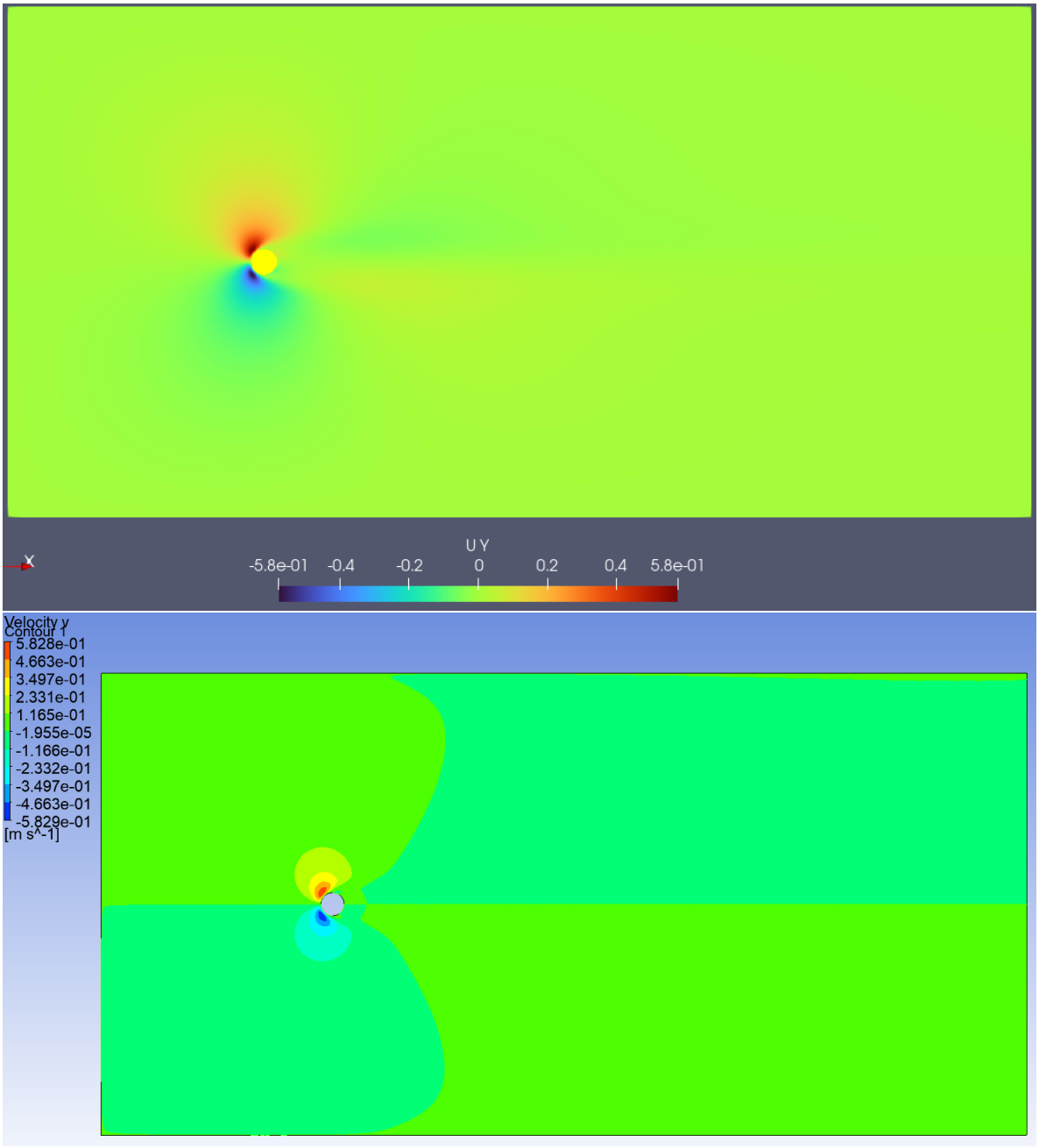


Figura 3.7: Componente verticale. Rete Neurale (sopra) Ansys (sotto).

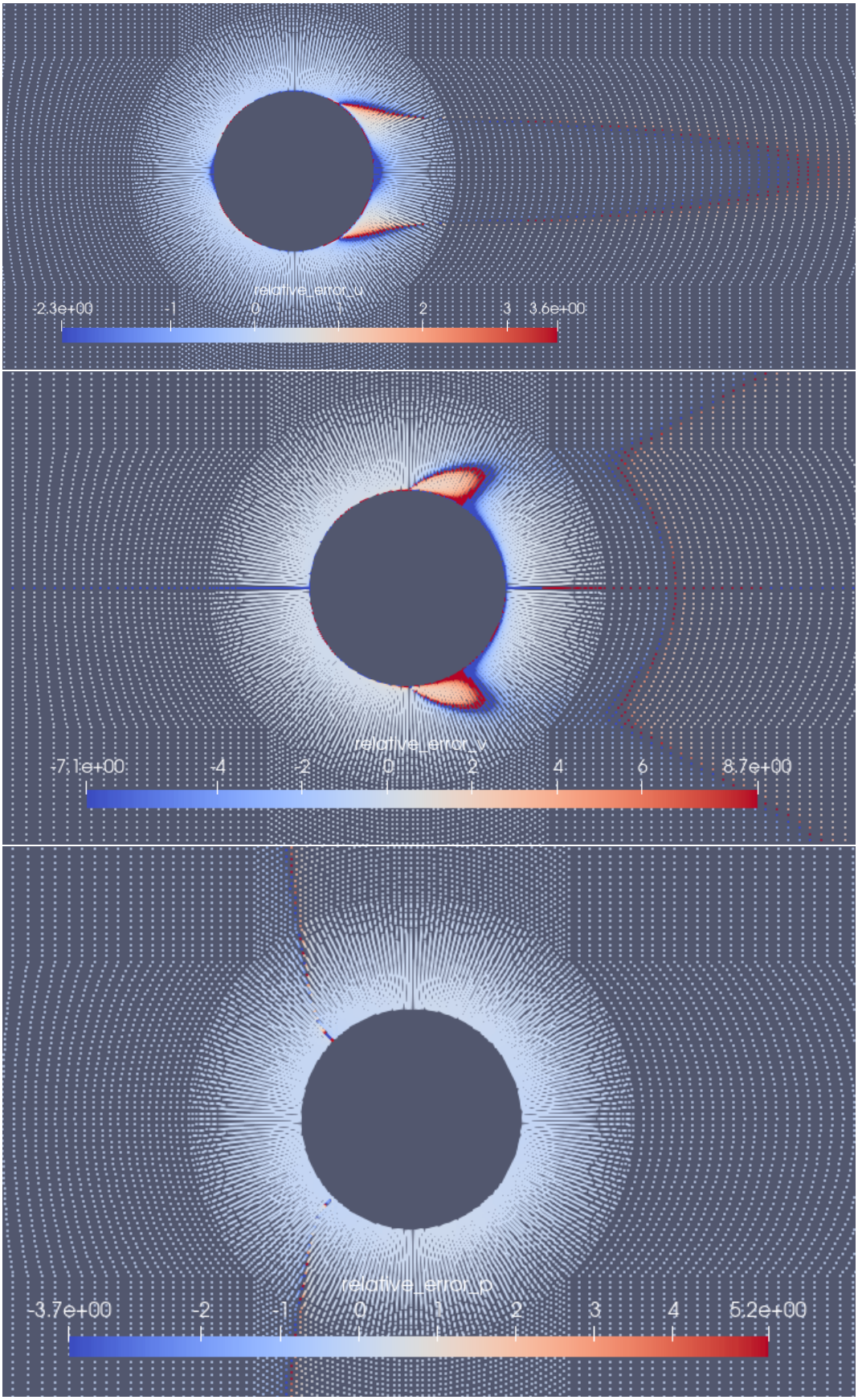


Figura 3.8: Errore percentuale rispetto ai risultati generati con Ansys (oriz,vert,p)

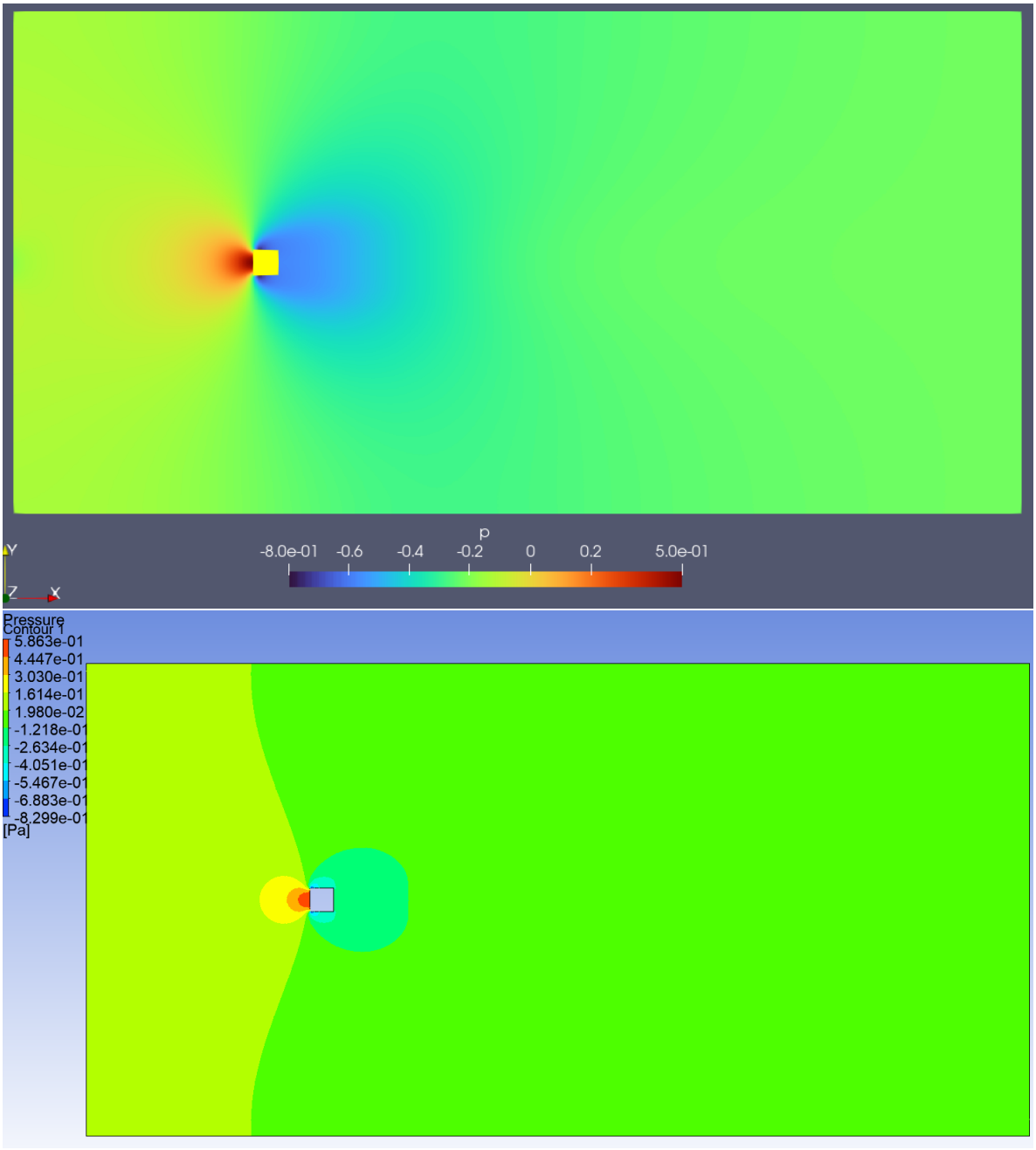


Figura 3.9: Campo di pressione. Rete Neurale (sopra) Ansys (sotto).

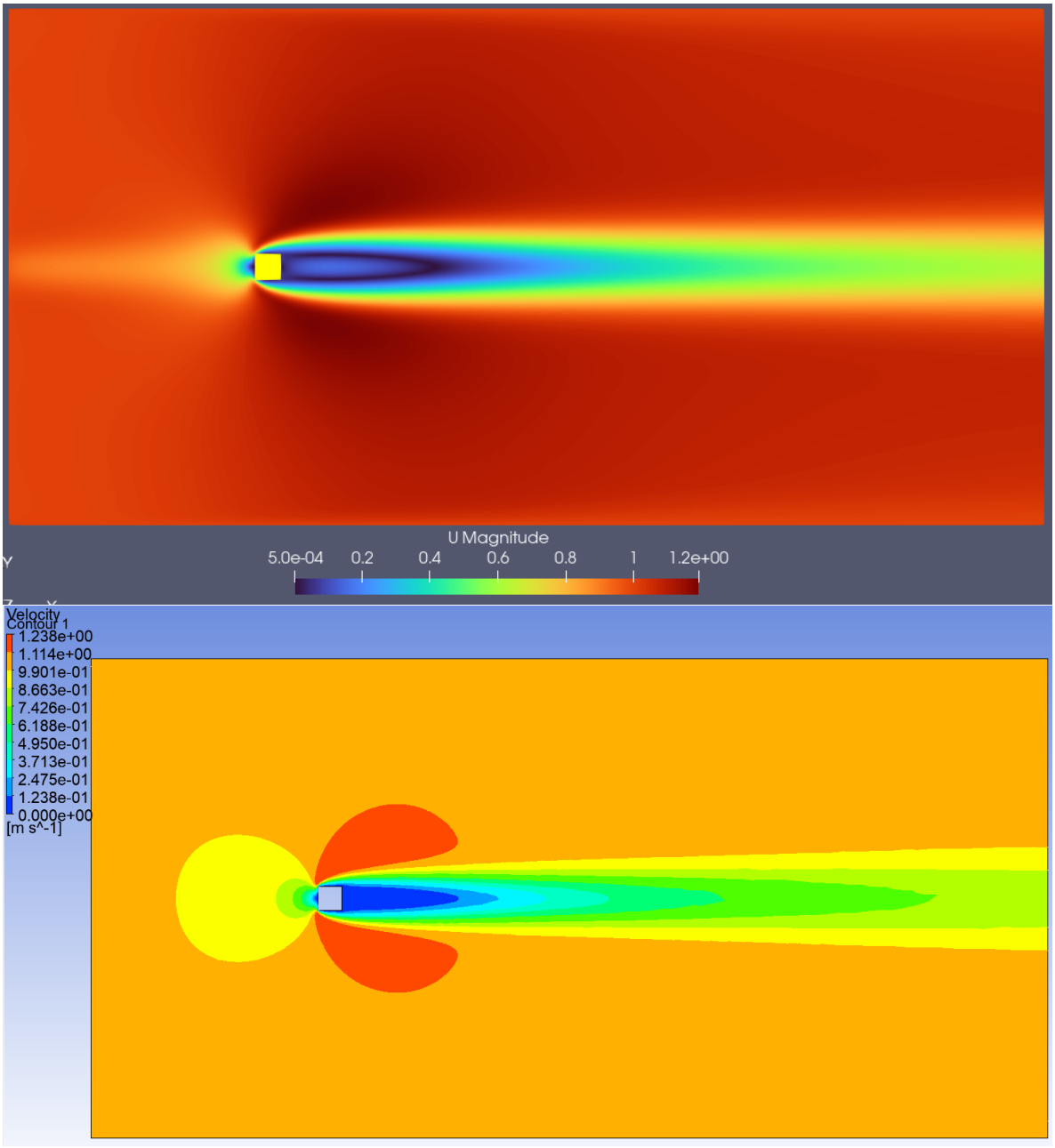


Figura 3.10: Modulo della velocità. Rete Neurale (sopra) Ansys (sotto).

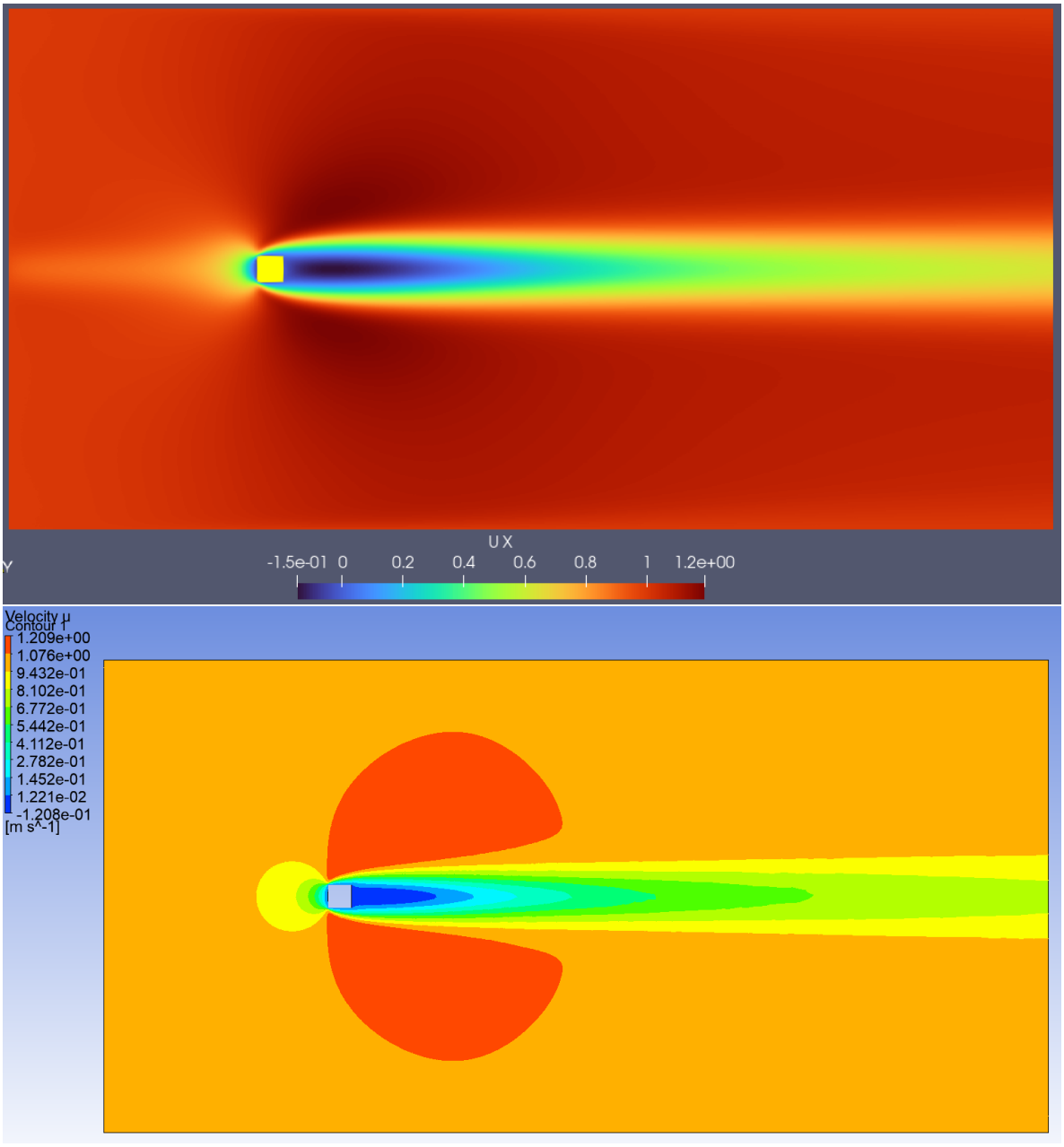


Figura 3.11: Componente orizzontale. Rete Neurale (sopra) Ansys (sotto).

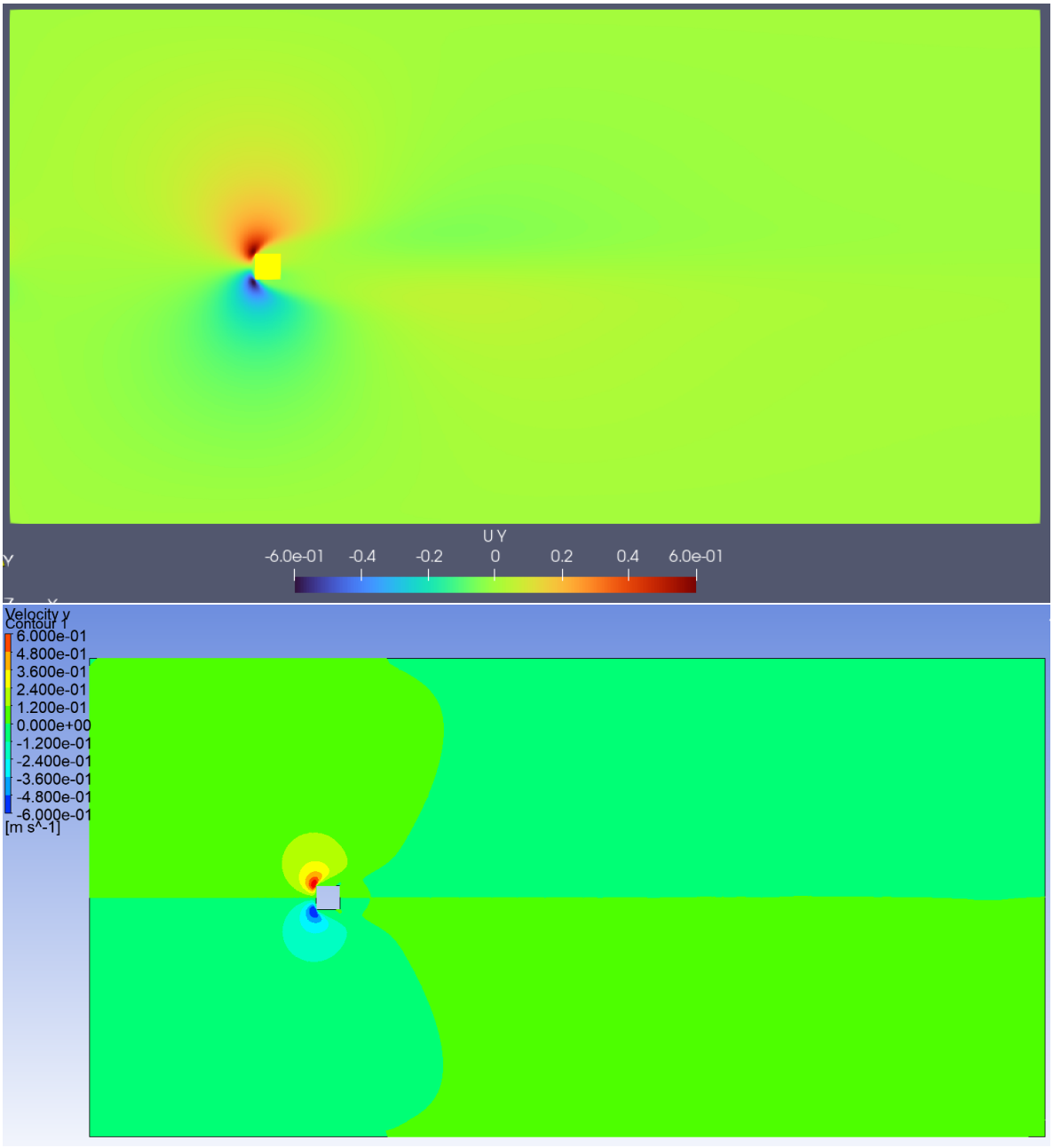


Figura 3.12: Componente verticale. Rete Neurale (sopra) Ansys (sotto).

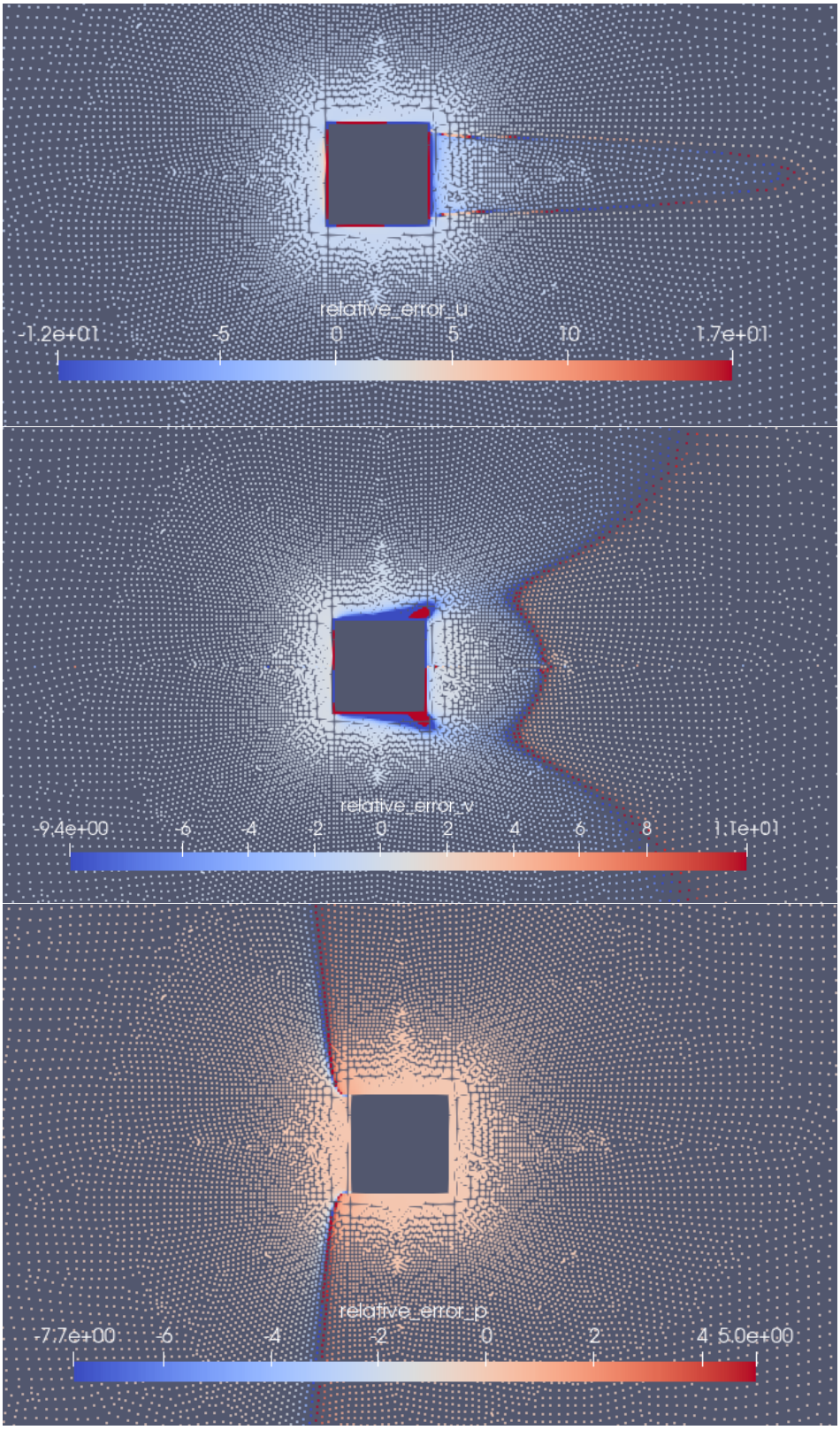


Figura 3.13: Errore percentuale rispetto ai risultati generati con Ansys (oriz,vert,p)

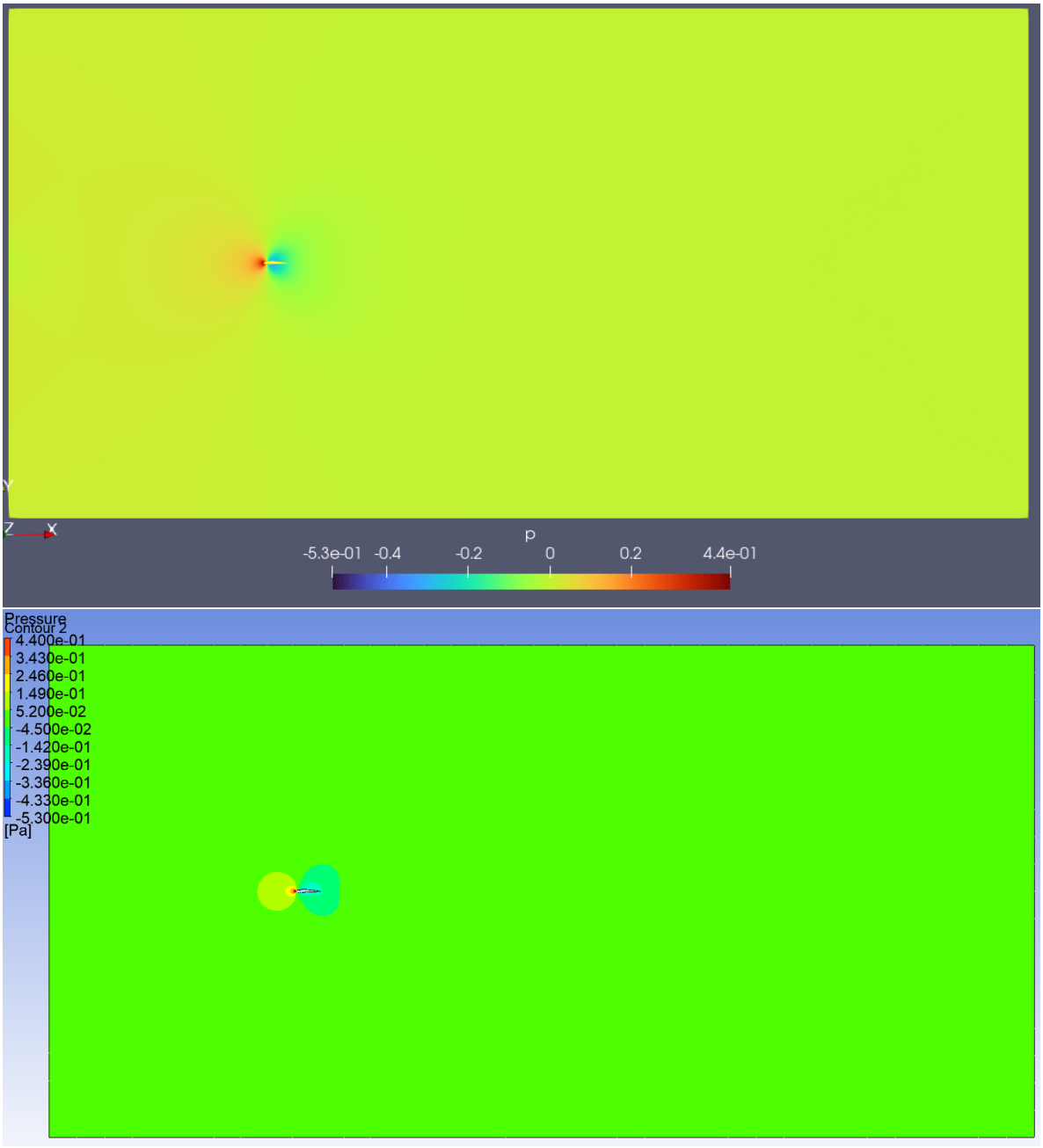


Figura 3.14: Campo di pressione. Rete Neurale (sopra) Ansys (sotto).

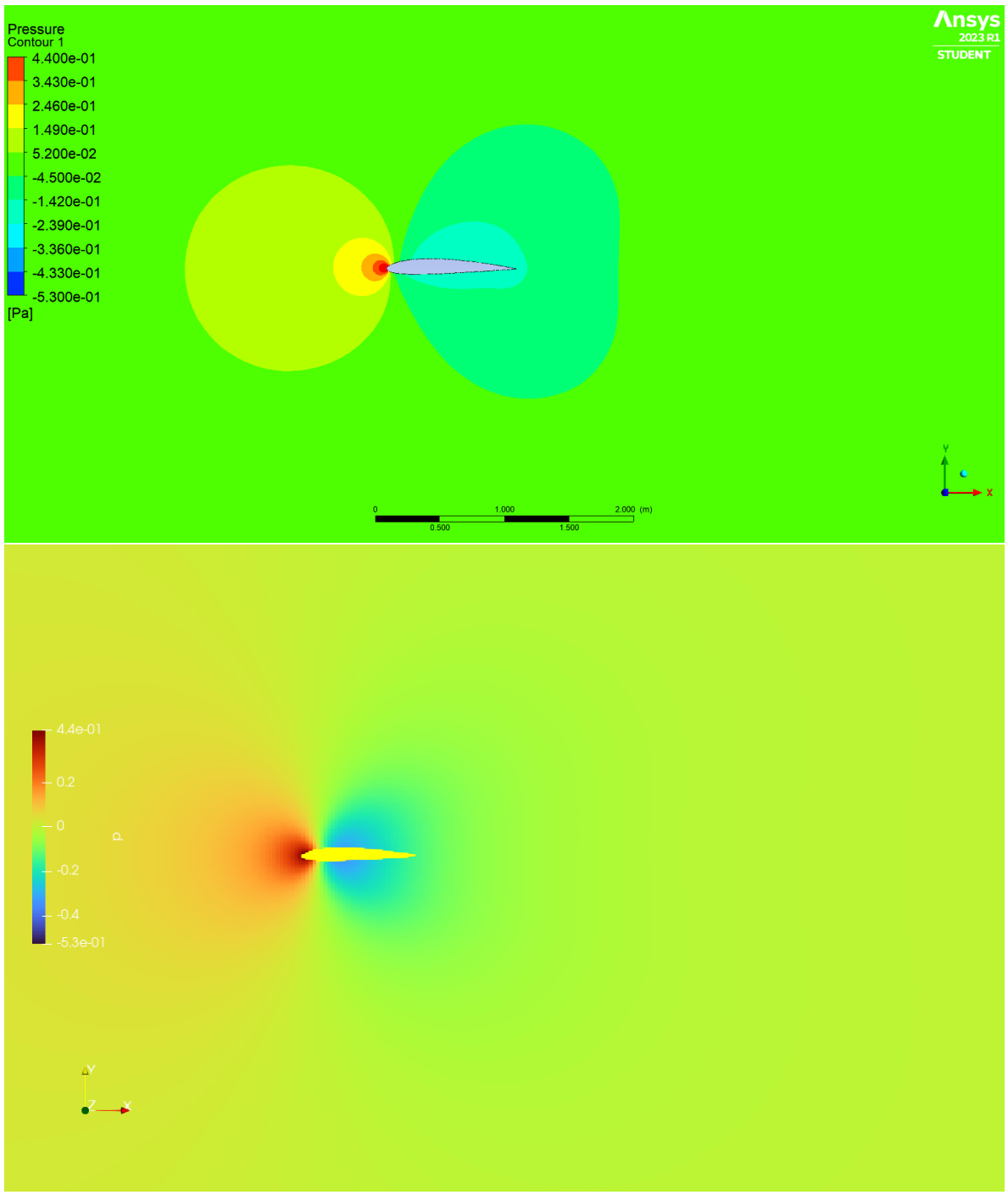


Figura 3.15: Campo di pressione (zoom). Rete Neurale (sopra) Ansys (sotto).

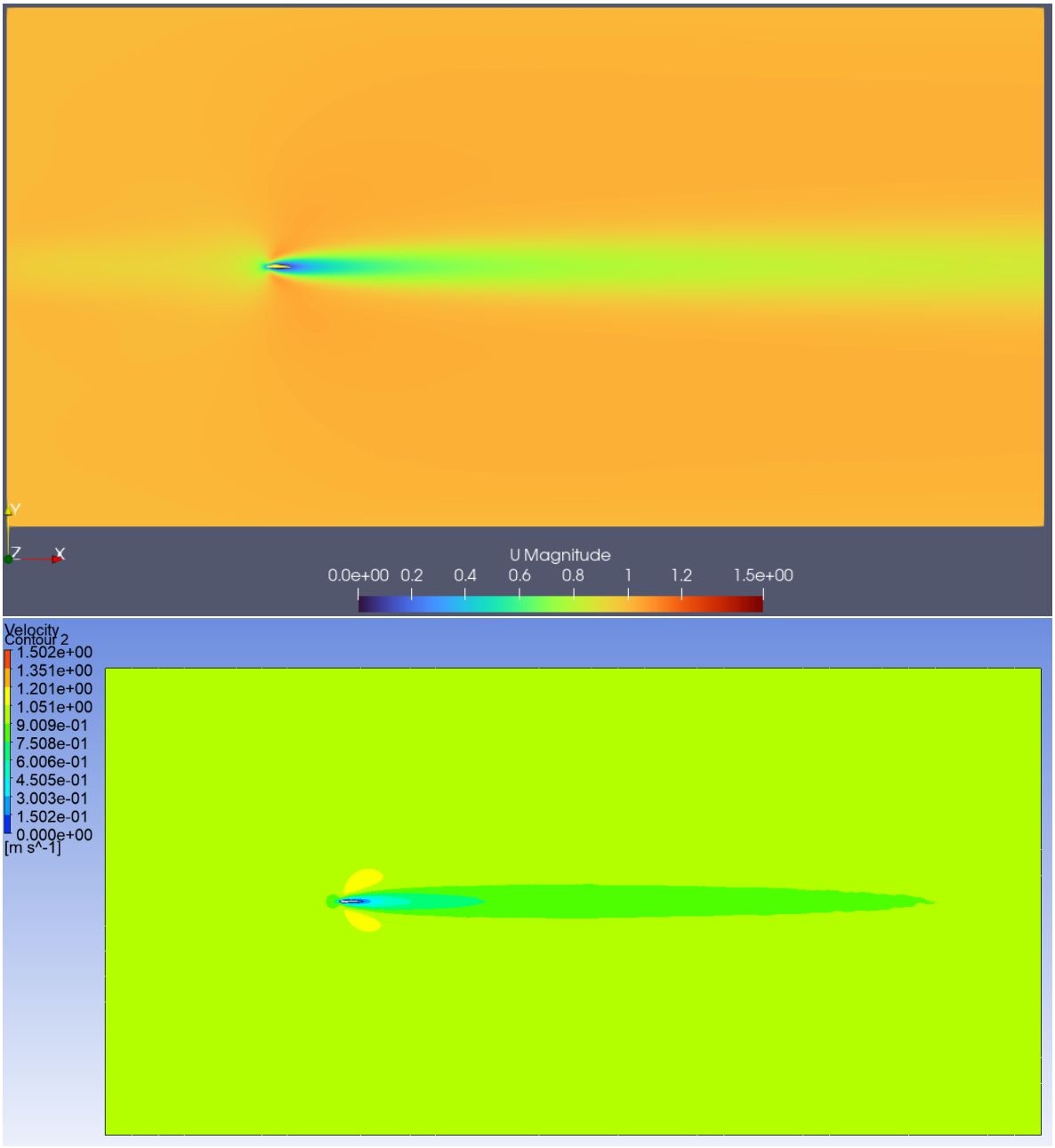


Figura 3.16: Modulo della velocità. Rete Neurale (sopra) Ansys (sotto).

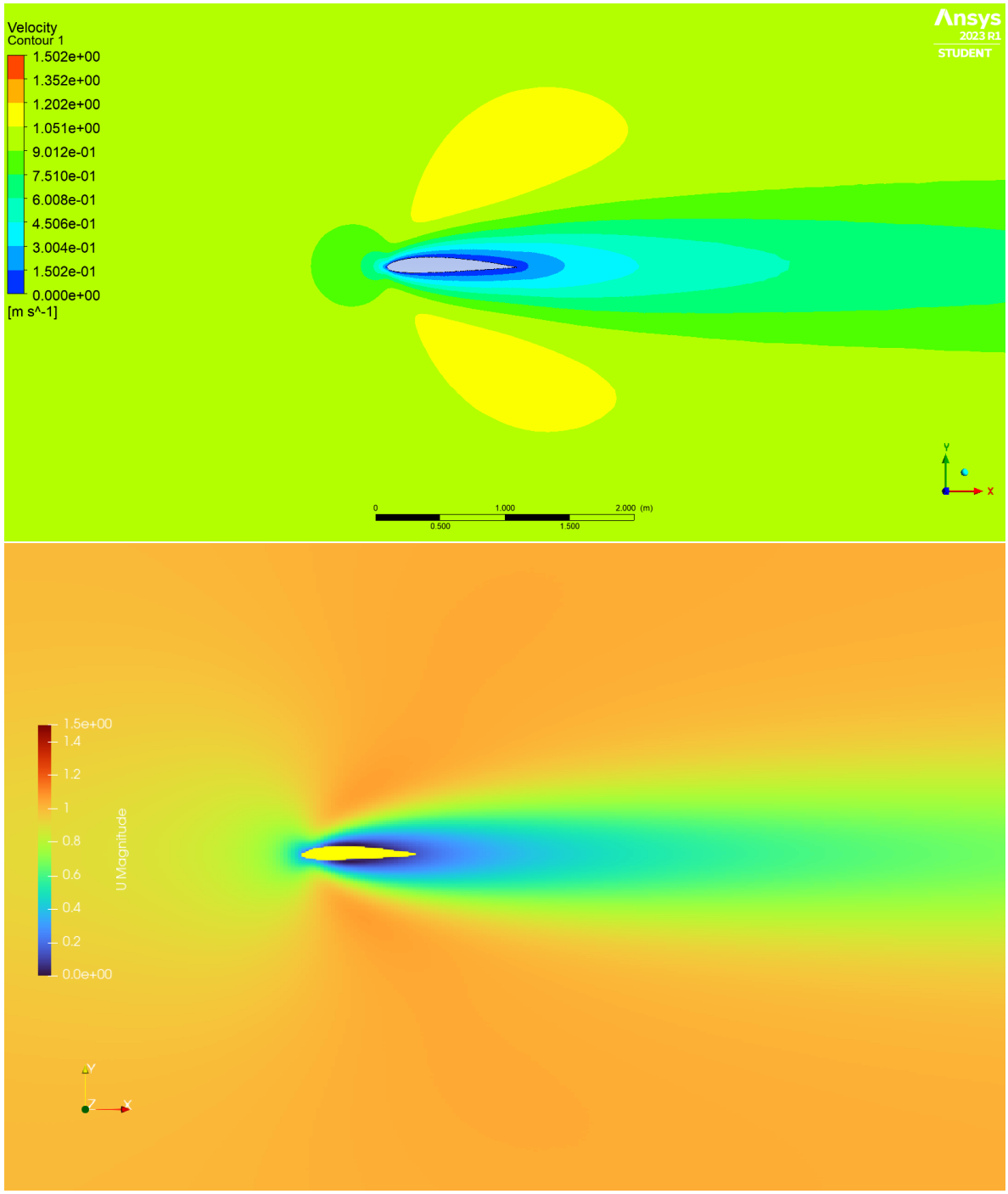


Figura 3.17: Modulo della velocità (zoom). Rete Neurale (sopra) Ansys (sotto).

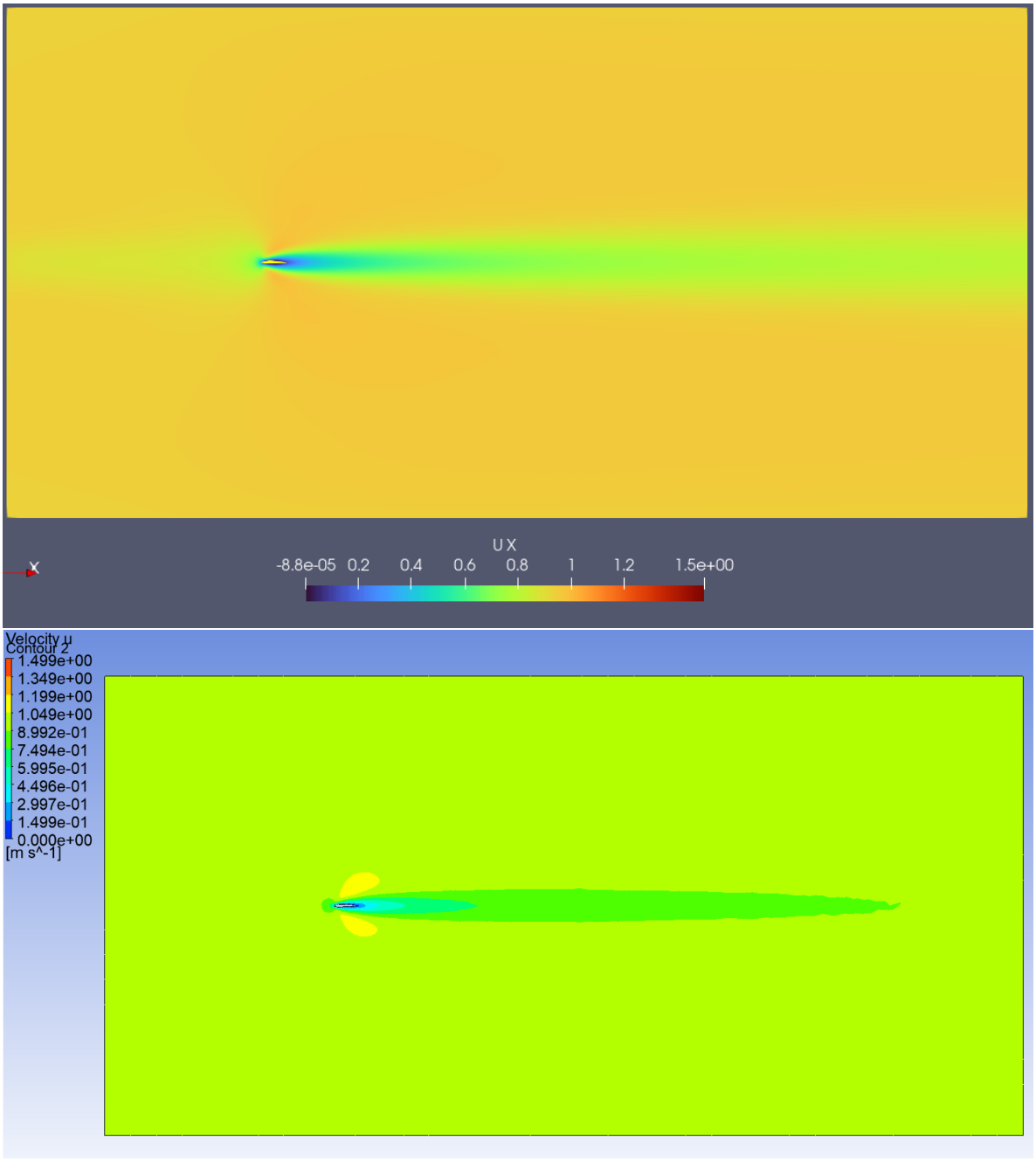


Figura 3.18: Componente orizzontale. Rete Neurale (sopra) Ansys (sotto).

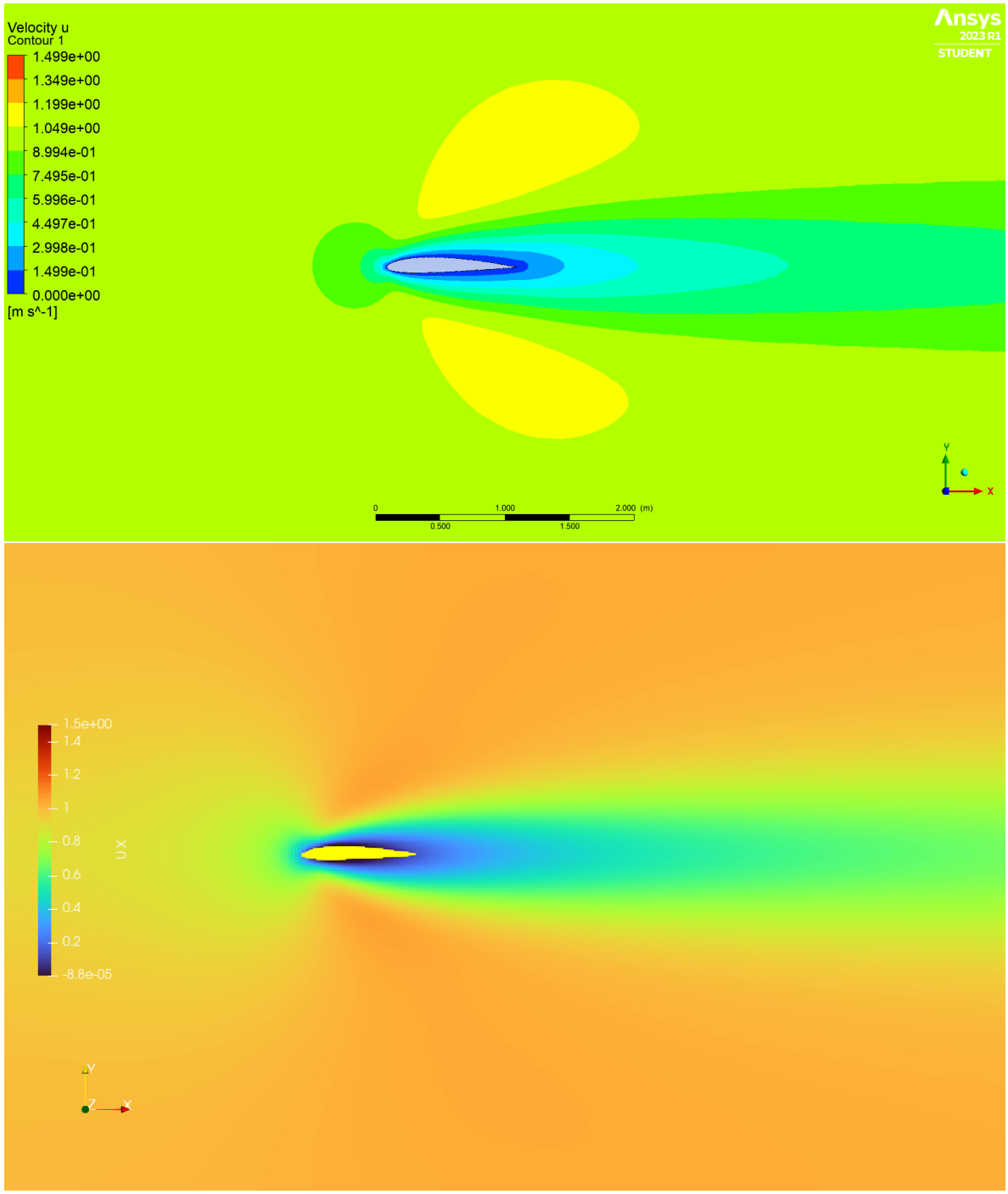


Figura 3.19: Componente orizzontale (zoom). Rete Neurale (sopra) Ansys (sotto).

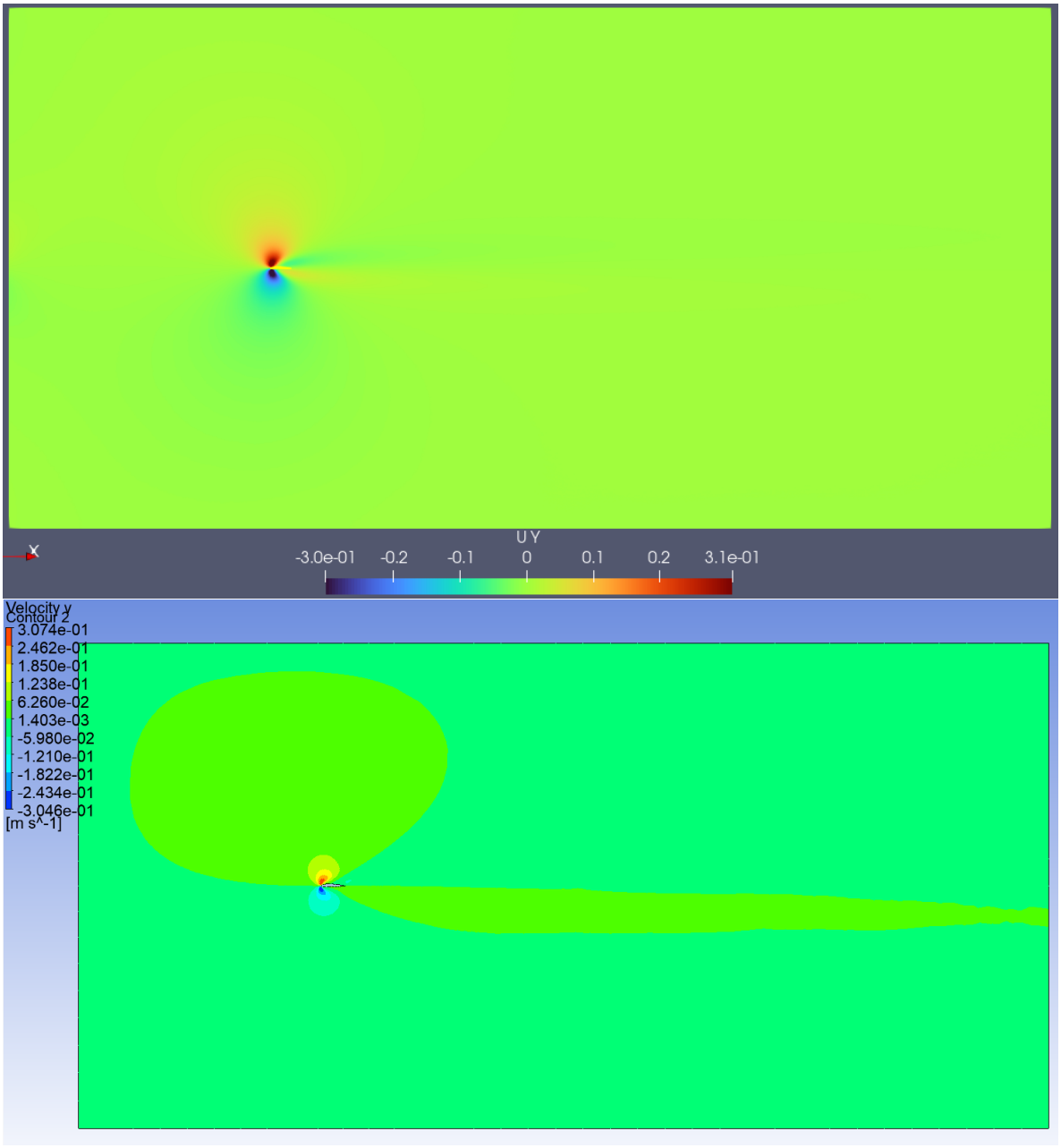


Figura 3.20: Componente verticale. Rete Neurale (sopra) Ansys (sotto).

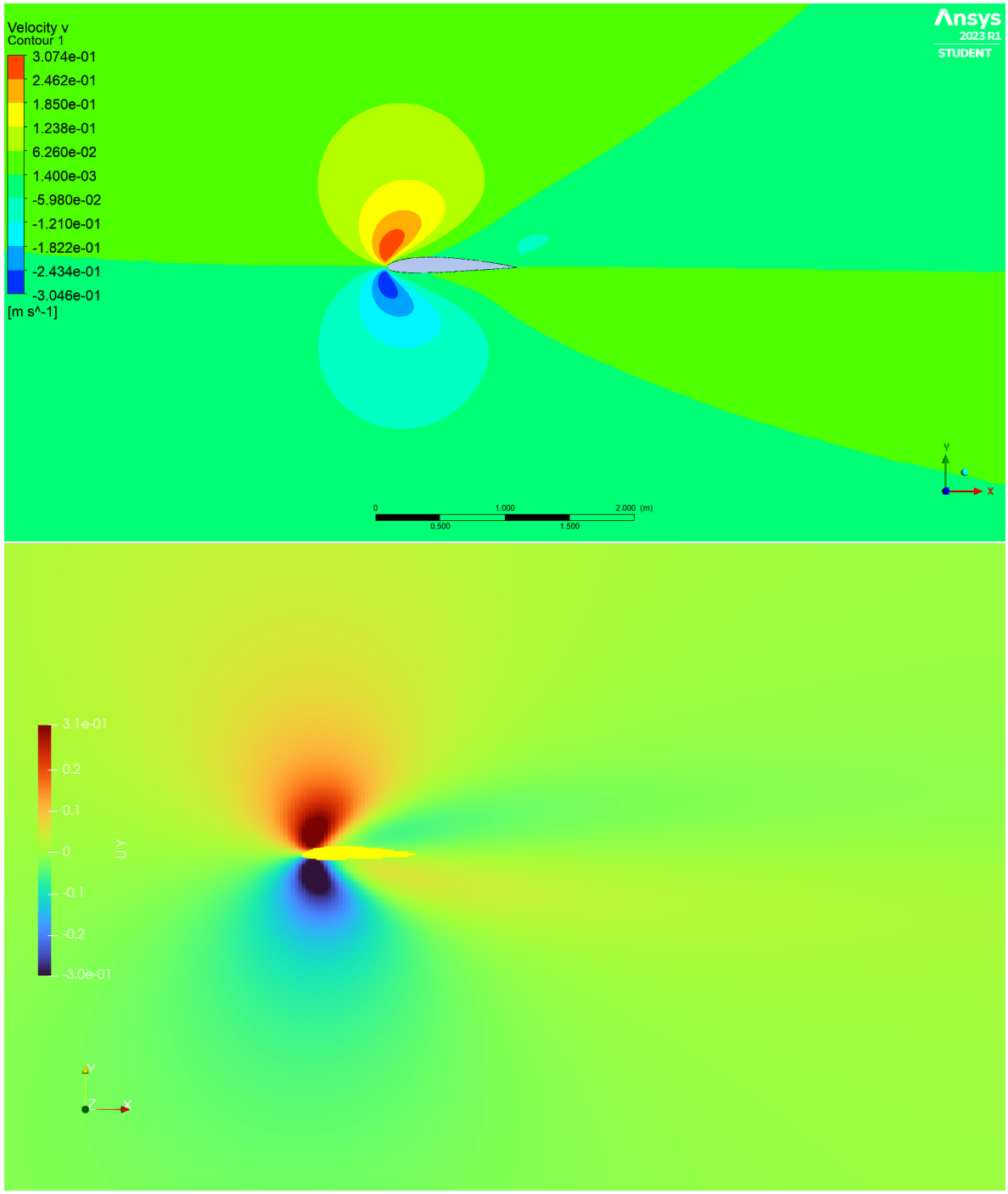


Figura 3.21: Componente verticale (zoom). Rete Neurale (sopra) Ansys (sotto).

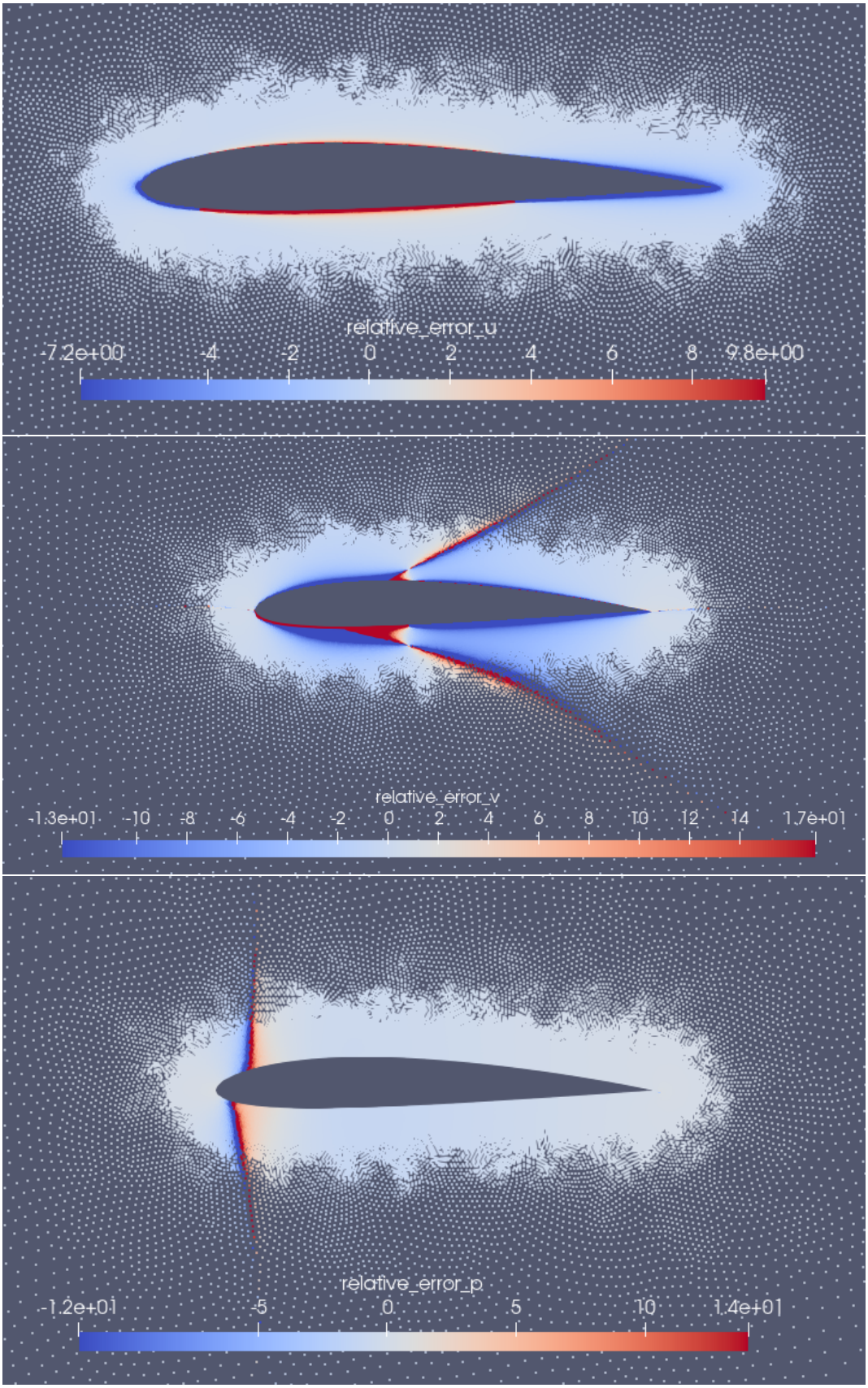


Figura 3.22: Errore percentuale rispetto ai risultati generati con Ansys (oriz,vert,p)

La rete neurale è riuscita a catturare la soluzione globalmente, ad ogni modo in prossimità del corpo si osserva un fenomeno di instabilità che porta ad un drastico aumento dell'errore. Specialmente nella predizione del campo di velocità, con l'errore massimo registrato per la componente verticale $v$. La generazione di questo errore è da imputare ad una imposizione "soft" della condizione di aderenza della rete neurale. Una precisione maggiore si sarebbe potuta ottenere aumentando il batch size, questo però non è stato possibile in quanto si sarebbero superati gli 8GB di memoria utilizzabili sulla scheda video disponibile. Inoltre il tempo di training per ogni caso è stato mediamente di 8 ore, in forte contrasto con i pochi minuti richiesti dal software Ansys per la generazione delle simulazioni. L'accuratezza delle simulazioni è fortemente influenzata dal numero di Reynolds, i risultati sono stati presentati per il numero di Reynolds più grande per cui è stato possibile ottenere simulazioni soddisfacenti.

La rete neurale è riuscita ad ottenere i risultati partendo solamente da una nuvola di punti fornita come dataset, questo svincola questa tecnica dalla generazione di una griglia. Quindi l'introduzione di geometrie più complesse è immediata, sarà semplicemente necessario aumentare la quantità di punti campionati. Inoltre la soluzione ottenuta è del tutto analitica, infatti la rete neurale risulta essere una funzione differenziabile. Questa caratteristica può essere utile per analisi di stabilità, ove risulti utile ad esempio una stima dei gradienti del campo di moto.

Nonostante l'errore maggiore è stato riscontrato nell'intorno del corpo la rete è riuscita comunque a predire il valore del coefficiente di resistenza e portanza. Inoltre è importante notare come per quanto affetti da errore i coefficienti rispettano qualitativamente il comportamento atteso riuscendo ad esempio a predire che il quadrato porta la resistenza più grande. I risultati sono riportati nella seguente tabella:

$$C_D = \frac{2F_D}{\rho V^2 L} \qquad C_L = \frac{2F_L}{\rho V^2 L}$$

| $C_D$ | Cilindro | Quadrato | NACA 2412 |
|---|---|---|---|
| Ansys | 1.4782 | 1.6316 | 0.626 |
| R.N. | 0.8782 | 1.0372 | 0.2342 |

| $C_L$ | Cilindro | Quadrato | NACA 2412 |
|---|---|---|---|
| Ansys | 0 | 0 | 0.029 |
| R.N. | -0.0003 | -0.0001 | 0.002 |

sebbene i risultati siano affetti da un errore non trascurabile viene catturato dalla rete l'andamento generale delle soluzioni, riuscendo ad esempio a prevedere quale geometria è quella a generare più resistenza o portanza.

Le performance della rete non sono paragonabili ai metodi di risoluzione classici, sia per i bassi numeri di Reynolds per cui la rete è in grado di portare risultati plausibili sia per i tempi di addestramento nettamente superiori a quelli di risoluzione tramite codici classici di CFD. L'utilizzo di una rete neurale fully-connected è risultato inadeguato per la risoluzione delle equazioni di Navier-Stokes, per quanto sia possibile ottenere i campi di moto non risulta vantaggioso l'applicazione della rete stessa. Ad ogni modo questo genere di architettura risulta essere uno dei più semplici ed è ragionevole aspettarsi un confronto negativo se confrontata con tecniche che godono di oltre 60 anni di sviluppo, come ad esempio codici ai volumi finiti.

# Capitolo 4

# Applicazione di una rete neurale convoluzionale per il miglioramento delle performance

Nel capitolo precedente sono stati evidenziati i limiti nell'utilizzo di una rete neurale fully-connected, infatti la rete neurale ha prodotto simulazioni solo per numeri di Reynolds bassi e non ha mostrato nessuna capacità di generalizzazione, necessitando di una nuova procedura di training per ogni caso risolto. Inoltre il dataset che deve essere fornito alla rete risulta avere una dimensione ingente, richiedendo hardware ad alte prestazioni.

In questo capitolo si introdurranno le reti neurali a layer convoluzionali (CNN), verrà poi adottata la rete CNN U-Net[3] e il metodo Spline-PINN. L'applicazione di questa rete ha permesso di risolvere le equazioni di Navier-Stokes per il caso instazionario e incomprimibile, portando il numero di Reynolds fino a 200. L'aspetto però più interessante è la capacità di questa rete di riuscire a generalizzare su diverse geometrie con un solo ciclo di training, ottenendo di fatto simulazioni istantanee per un diverso numero di parametri.

## 4.1 Layer Convoluzionale

Le reti CNN sono nate per l'analisi e il processamento delle immagini, esse estendono il concetto di layer da semplice vettore a matrice, creando quindi dei layer matriciali sui

quali avvengono delle operazioni di filtraggio ovvero le convoluzioni. Questo genere di reti sono state introdotte per la loro capacità di estrarre caratteristiche e comportamenti (feature) dal dataset fornito tramite delle operazioni di filtraggio. La tecnica Spline-PINN sfrutta questa caratteristica effettuando diversi cicli di training inizializzando la rete tra un ciclo e un altro su un nuovo dominio, in tale modo si è riusciti ad avere una rete in grado di generalizzare su diversi problemi.

### 4.1.1 Funzionamento

Un layer convoluzionale agisce tramite un filtro $m \times m$, su un layer di neuroni $N \times N$ (non è per forza necessario lavorare con matrici quadrate). La convoluzione consiste nello far scorre lungo le colonne una riga alla volta, ricavando per ogni movimento un valore che corrisponde all'input pesato del layer convoluzionale. Il risultato della applicazione del

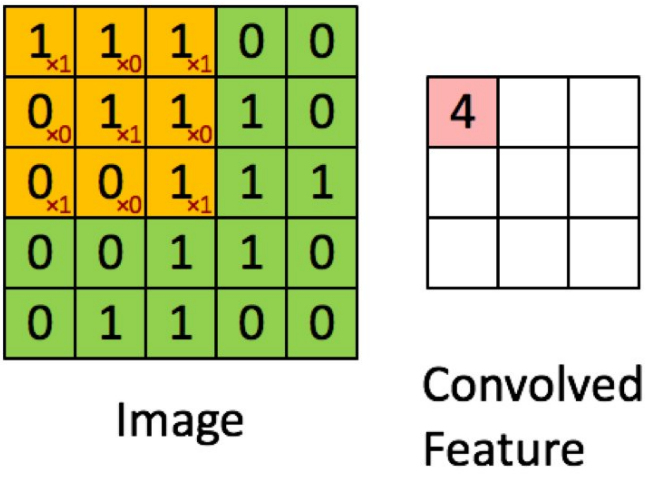


Figura 4.1: Applicazione di un filtro $3 \times 3$, il primo elemento dell'input pesato viene calcolato tramite i pesi $w_{i,j}$[25]

filtro è nuovamente una matrice ma con dimensione $(N-m+1) \times (N-m+1)$, questa matrice ha come elementi i singoli output pesati calcolati moltiplicando elemento per elemento i neuroni del layer in ingresso con i valori dei pesi del filtro e sommando i vari prodotti.

$$z_{i,j}^{[l]} = \sum_{s=0}^{m-1} \sum_{t=0}^{m-1} w_{s,t} a_{(i+s),(j+t)}^{[l-1]} \tag{4.1}$$

Una volta ottenuti gli output pesati $z_{i,j}^{[l]}$ è possibile calcolare il valori dei neuroni nel layer convoluzionale applicando la funzione di attivazione desiderata:

$$a_{i,j}^{[l]} = g(z_{i,j}^{[l]}) \tag{4.2}$$

il filtro contiene i pesi da addestrare durante la backpropagation. Grazie a questa architettura è possibile estrarre diverse informazioni applicando diverse operazioni di filtraggio, le quali possono lavorare in parallelo. Ad esempio nella rete U-net di hanno 5 diverse schiere di layer convoluzionali che lavorano in parallelo.

### 4.1.2 Max-Pooling layer

Tra due layer convoluzionali può essere eseguita un operazione di max-pooling, questa operazione serve principalmente a diminuire la dimensionalità dell'input e non ha rilevanza durante l'addestramento, ovvero questi layer non sono in grado di imparare tramite backporpagation. Questo tipo di layer riduce blocchi $k \times k$ dell'input ad un solo valore, la scelta del singolo valore in genere è il valore più grande presente nel blocco (da qui il termine max) ma può anche essere effettuata una operazione di media. Data una matrice in ingresso $N \times N$ l'output di questo layer avrà dimensione $\frac{N}{k} \times \frac{N}{k}$.

### 4.1.3 Modifica delle equazioni di backpropagation[10]

Le equazioni di backpropagation si modificano tenendo conto della applicazione del filtro. Il gradiente della funzione di costo rispetto ai pesi del filtro può essere espresso con la regola della catena.

$$\frac{\partial C}{\partial w_{s,t}^{[l]}} = \sum_{s=0}^{m-1}\sum_{t=0}^{m-1} \frac{\partial C}{\partial z_{i,j}^{[l]}} \frac{\partial z_{i,j}^{[l]}}{\partial w_{s,t}^{[l]}} = \sum_{s=0}^{m-1}\sum_{t=0}^{m-1} \frac{\partial C}{\partial z_{i,j}^{[l]}} a_{(i+s),(j+t)}^{[l-1]} \tag{4.3}$$

Gli errori relativi agli output pesati invece vengono calcolati in questo modo:

$$\frac{\partial C}{\partial z_{i,j}^{[l]}} = \frac{\partial C}{\partial a_{i,j}^{[l]}} \frac{\partial a_{i,j}^{[l]}}{\partial z_{i,j}^{[l]}} = \frac{\partial C}{\partial a_{i,j}^{[l]}} \sigma'(x_{i,j}^{[l]}) \tag{4.4}$$

L'ultima equazione da modificare è quella relativa al calcolo dell'errore rispetto al valore generico valore neuronale nel layer precedente:

$$\frac{\partial C}{\partial a_{i,j}^{[l-1]}} = \sum_{s=0}^{m-1}\sum_{t=0}^{m-1} \frac{\partial C}{\partial z_{(i-s),(j-t)}^{[l]}} \frac{\partial x_{(i-s),(j-t)}^{[l]}}{\partial a_{i,j}^{[l-1]}} = \sum_{s=0}^{m-1}\sum_{t=0}^{m-1} \frac{\partial C}{\partial x_{(i-s),(j-t)}^{[l]}} w_{s,t} \tag{4.5}$$

Queste tre equazioni permettono di implementare la backpropagation attraverso layer convoluzionali. Per quanto riguarda i layer di max-pooling non è necessario introdurre nessuna nuova equazione aggiuntiva in quanto non introducono nessun parametro addestrabile all'interno della rete.

## 4.2 Spline-PINN

In questa sezione viene introdotto il metodo sviluppato da N. Wandel et al [16] Spline-PINN. Con questa tecnica viene introdotta una nuova procedura di training, inoltre la soluzione viene costruita come funzione a tratti tramite Hermite-spline, la rete avrà il compito di prevedere i coeffcienti delle funzioni spline. La rete ottenuta, oltre a permettere un'analisi instazionaria ed alzare il numero di Reynolds della simulazione, ha permesso di risolvere, quasi in tempo reale, diverse configurazioni di un determinato caso di studio. In particolare si è potuto ricavare l'andamento del coefficiente di portanza e resistenza, nel caso di un cilindro in un condotto, per diversi valori del diametro e posizione del cilindro.

### 4.2.1 Impostazione del problema

Si riportano ora le equazioni di Navier-Stokes per il caso incomprimibile e instazionario:

$$\begin{cases} \nabla \cdot \mathbf{V} = 0 \quad (x, y) \in \Omega \\ \partial_t \mathbf{V} + \mathbf{V} \cdot \nabla \mathbf{V} = -\nabla p + \nu(\nabla^2 \mathbf{V}) \quad (x, y) \in \Omega \\ \mathbf{V} = \mathbf{V}_d \quad (x, y) \in \partial\Omega \end{cases} \tag{4.6}$$

la rete al momento gestisce soltanto la condizione di Dirichlet per la velocità. La condizione di divergenza nulla non verrà utilizzata direttamente nella funzione di loss ma invece verrà imposta indirettamente introducendo un potenziale vettore della velocità.

$$\mathbf{V} = \nabla \times \mathbf{a} \tag{4.7}$$

In questo modo si sfrutta la proprietà matematica per cui la divergenza di un rotore è sempre nulla, di conseguenza i campi ottenuti cercando una soluzione del tipo (4.7) sicuramente rispettano la condizione di continuità $\nabla \cdot \mathbf{V} = 0$. Volendo risolvere le equazioni per il caso 2D l'unica componente del potenziale vettore $\mathbf{a}$ richiesta è $a_z$

infatti il campo di velocità risulta della seguente forma.

$$(u, v) = \left(\frac{\partial a_z}{\partial y}, -\frac{\partial a_z}{\partial x}\right) \tag{4.8}$$

La funzione scalare $a_z$ e il campo di pressione $p$ verranno interpolati con delle funzioni spline, i coefficienti delle spline saranno predetti dalla rete. Per rappresentare la soluzione tramite queste funzioni è necessario introdurre una griglia dove collocare i punti di supporto delle spline. La griglia però è necessaria solo alla rappresentazione della funzione definita a tratti ma non è utilizzata dalla rete per la previsione dei coefficienti. nella figura (4.3) vengono visualizzati dei campi di moto $(u, v)$ ottenuti dal rotore di potenziali vettori generati tramite Hermite-spline 2D del secondo ordine.

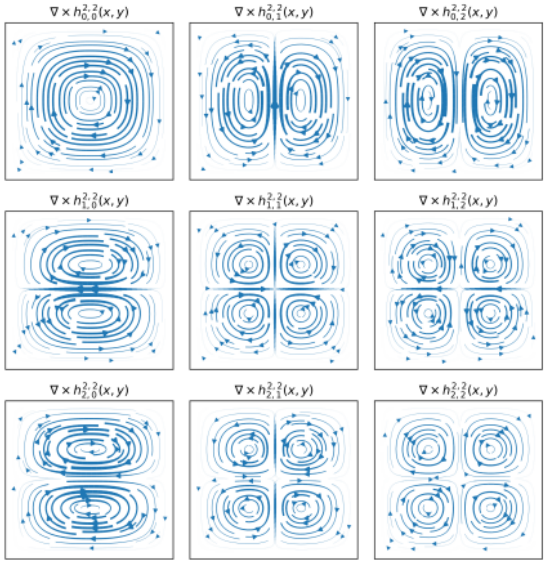

Figura 4.2: Componente z della divergenza del potenziale vettore **a**. I campi di moto generati rispettano, per ipotesi, la condizione a divergenza nulla.[16]

### 4.2.2 Hermite-Spline

Le Hermite spline sono delle funzioni polinomiali definite a tratti definite imponendo delle determinate condizioni sul valore della funzione e delle sue prime $n$ derivate su dei punti detti di supporto. Per facilitare l'apprendimento della rete i punti di supporto sono stati collocati su una griglia uniforme. La figura 4.3 mostra degli esempi di Hermite spline per il caso 1D per $n = 0, 1, 2$. Le funzioni di base $h_i^n(x)$ sono definite in modo tale che le prime $n$ derivate sui punti si supporto $(x = -1, 0, 1)$ sono poste a zero, tranne per la iesima derivata in $x = 0$, la quale dovrà essere tale da prendere valori tra

$[-1, 1]$, ovvero che $h_i^n(x) \in [-1, 1]$. A seconda del valore $n$ scelto si generano diversi set di polinomi di vario grado, il valore di $n$ è scelto in base alla regolarità della funzione richiesta. Nel nostro caso sarà necessario almeno $n = 2$ in quanto sarà necessario calcolare la derivata terza di $a_z$.

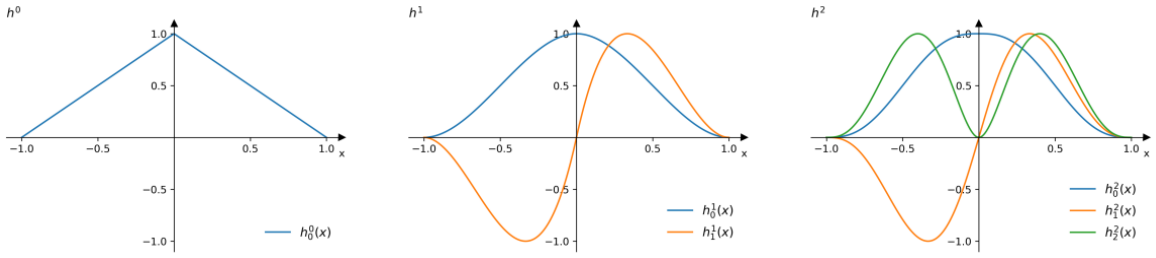


Figura 4.3: Hermite spline 1D per $n = 0, 1, 2$. Siccome le funzioni spline sono di classe $C^n$ allora si ha che la $n + 1$esima derivata è comunque una funzione a variazione limitata.[16]

Per ottenere le funzione di base anche per il caso bidimensionale sarà sufficiente moltiplicare la funzioni spline definite per le diverse variabili in questo modo:

$$h_{i,j,k}^{l,m,n}(x, y, t) = h_i^l(x) h_j^m(y) h_k^n(t) \tag{4.9}$$

la funzione ottenuta per linearità godrà delle stesse proprietà di regolarità e differenziabilità delle funzioni di base che la costituiscono. Andiamo quindi a scrivere la nostra soluzione sommando la funzioni spline ottenute su tutti i punti di supporto della griglia $(\hat{x}, \hat{y}, \hat{t}) \in \hat{X}, \hat{Y}, \hat{T}$ moltiplicandole per i coefficienti $c_{\hat{x},\hat{y},\hat{t}}^{i,j,k}$ ottenuti dalla rete CNN, si ottiene quindi:

$$g(x, y, t) = \sum_{\substack{i,j,k \in [0:l] \times [0:m] \times [0:n] \\ \hat{x},\hat{y},\hat{t} \in \hat{X},\hat{Y},\hat{T}}} c_{\hat{x},\hat{y},\hat{t}}^{i,j,k} h_{i,j,k}^{l,m,n}(x - \hat{x}, y - \hat{y}, t - \hat{t}) \tag{4.10}$$

l'obbiettivo è quello di addestrare la rete a prevedere i coefficienti $c_{\hat{x},\hat{y},\hat{t}}^{i,j,k}$ tali per cui la funzione $g(x, y, t)$ approssimi la soluzione della PDE. Inoltre siccome la funzione $g$ risulta comunque analitica è possibile computare le derivate rispetto alle variabili direttamente senza dover effettuare la backpropagation per risalire alle derivate rispetto agli input come nel caso precedente con l'architettura fully-connected.

La funzione di costo viene compiuta tramite integrazione sul dominio spazio temporale, la rete mira quindi a trovare i coefficienti che minimizzano i seguenti integrali:

$$L_p = \int_{\Omega} \int_{\hat{t}}^{\hat{t}+dt} \|\partial_t \mathbf{V} + \mathbf{V} \cdot \nabla \mathbf{V} = -\nabla p + \nu(\nabla^2 \mathbf{V})\|^2 \tag{4.11}$$

$$L_b = \int_{\partial\Omega} \int_{\hat{t}}^{\hat{t}+dt} \|\mathbf{V} - \mathbf{V}_d\|^2 \tag{4.12}$$

$$L_{tot}^{flow} = \alpha L_p + \beta L_b \tag{4.13}$$

nella equazione 4.13 sono statti aggiunti due pesi alle funzioni di loss rispettivamente per la quantità di moto e le condizioni al contorno. I pesi $\alpha$ e $\beta$ sono stati scelti in baso ad un indagine sul loro effetto riportata in figura (4.4), i valori scelti sono $\alpha = 10$ e $\beta = 20$. Tali valori permettono di imporre un peso maggiore della condizione al contorno, e quindi influenzano i parametri per una cattura migliore di tale condizione.

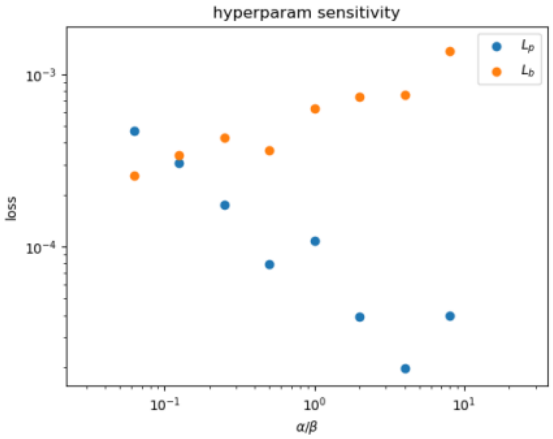


Figura 4.4: Sensibilità della loss globale alla variazione degli hyperparameters $\alpha$ e $\beta$.[16]

### 4.2.3 Procedura di addestramento[17]

La procedura di addestramento comincia con l'inizializzazione di una "training pool" $\{\Omega_k^0, (v_d)_k^0, (a_z)_k^0, p_k^0\}$ composta da geometrie primitive casuali e condizioni al contorno casuali, la componente $z$ del potenziale vettore e il campo di pressione vengono inizializzati a zero. Ad ogni step di addestramento viene campionato un mini-batch selezionando una determinata geometria e condizione al contorno le quali vengono fornite alla rete, per ogni passo temporale $dt$ la rete dovrà prevedere i coefficienti per i polinomi interpolanti il campo $p$ e la componente $z$ del potenziale vettore $a_z$. L'ottimizzazione dei pesi de bias della rete viene eseguita usando l'ottimizzatore Adam.

Durante la procedura di training avvengono delle reinizializzazioni su nuovi domini. La reinizializzazione avviene cambiando geometria e condizione al contorno e ponendo a zero il campo di pressione e la componente $z$ del potenziale vettore. Questa procedura

permette di introdurre nuove geometrie all'interno della rete permettendo lo sviluppo di capacità di generalizzazione, inoltre il riportare il campo di pressione e velocità a zero allena la rete alle "ripartenze a freddo". Allenare la rete a re-imparare le nuove soluzioni impostando a zero i campi di moto si è evidenziata una strategia vincente abbassando i tempi di training.

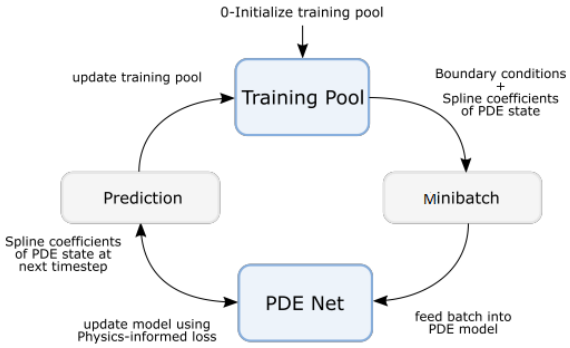


Figura 4.5: Ciclo di training utilizzato.[16]

### 4.2.4 Risultati

La rete è stata utilizzata per predirre i coefficienti di forza attorno ad un cilindro immerso in un condotto, per valutare la capacità di questa rete alla generalizzazione si è fatto variare il raggio del cilindro e si è valutato la capacità della rete di prevedere su nuove geometrie la soluzione senza necessitare di ulteriore training. La geometria di partenza è la seguente: il raggio diametro del cilindro è inizialmente posto posto ad

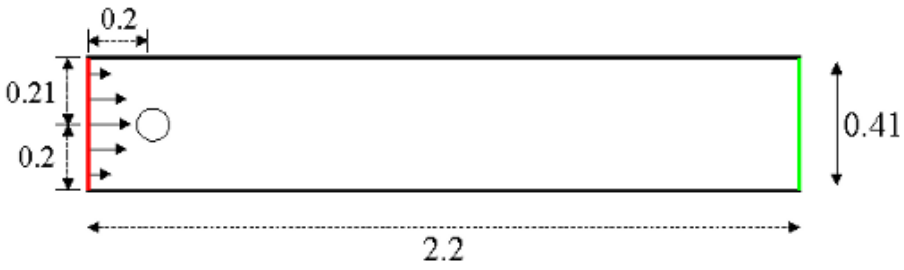


Figura 4.6: Geometria utilizzata.[7]

0,1 metri, questo procude un Reynolds pari a 100. Progressivamente il diametro del cilindro viene aumentato con un passo di 0,02 metri fino ad 0,2 metri. In questo modo non solo si modifica la geometria del problema ma si varia anche il numero di Reynolds, arrivando ad un numero di Reynolds finale pari ad 200.

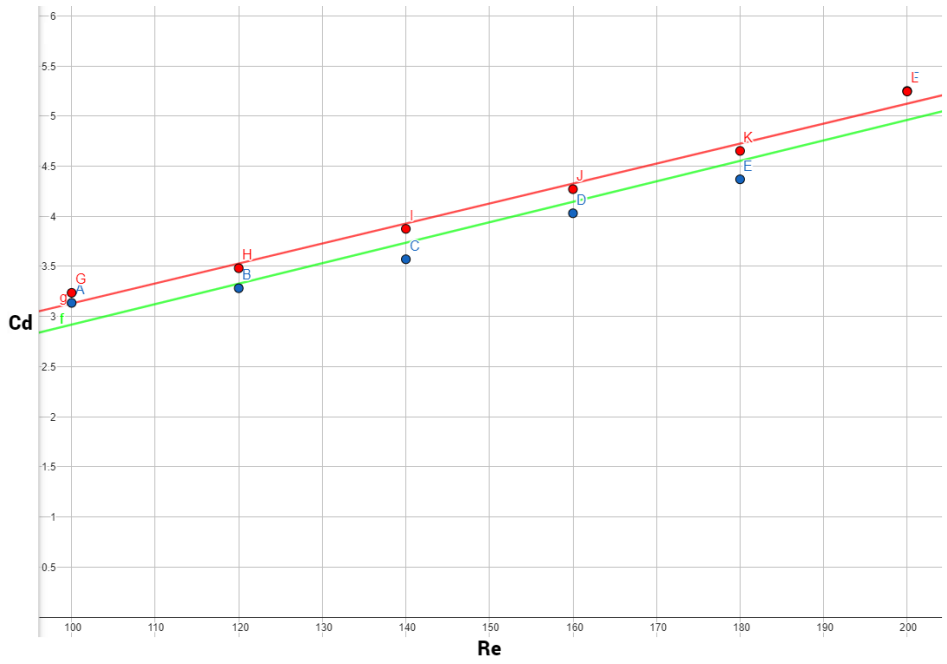


Figura 4.7: Coefficiente di resistenza massimo in funzione del Reynolds. Rete Neurale (verde) Ansys (rosso).

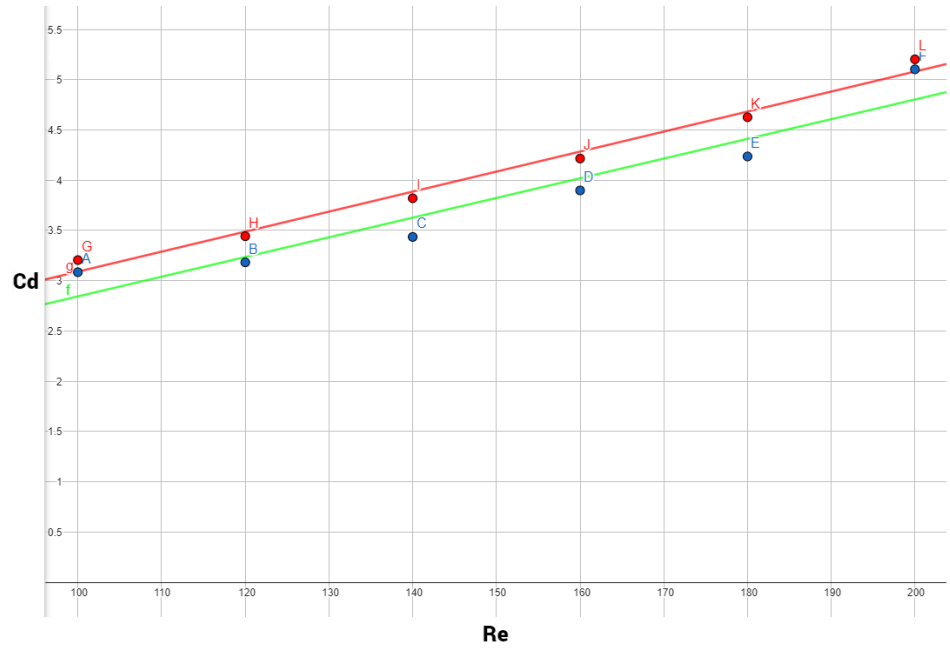


Figura 4.8: Coefficiente di resistenza medio in funzione del Reynolds. Rete Neurale (verde) Ansys (rosso).

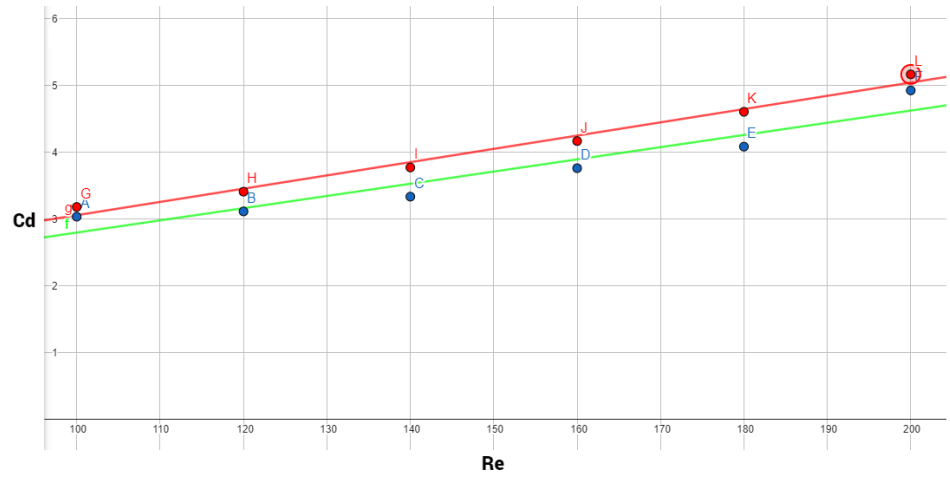


Figura 4.9: Coefficiente di resistenza minimo in funzione del Reynolds. Rete Neurale (verde) Ansys (rosso).

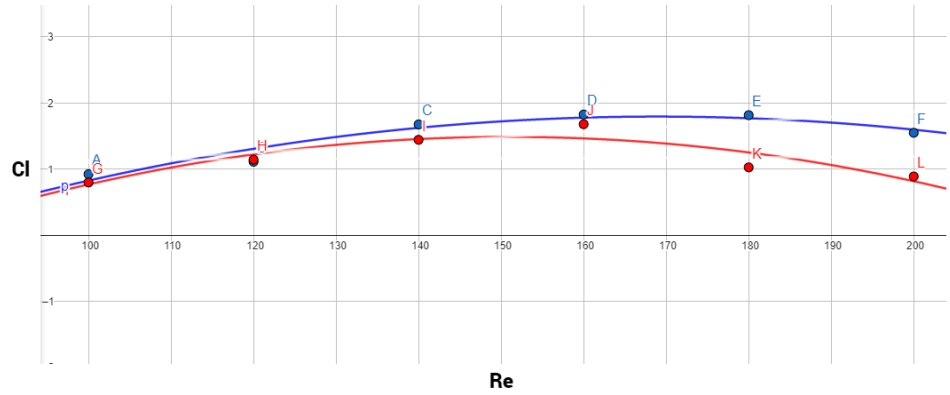


Figura 4.10: Coefficiente di portanza massimo in funzione del Reynolds. Rete Neurale (blu) Ansys (rosso).

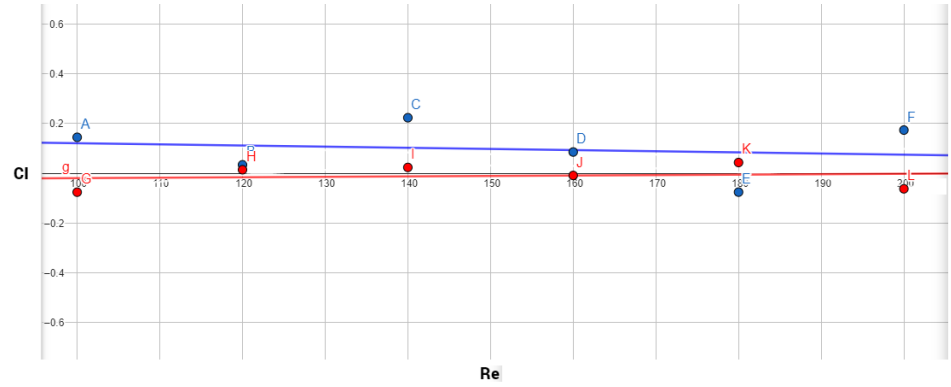


Figura 4.11: Coefficiente di portanza medio in funzione del Reynolds. Rete Neurale (blu) Ansys (rosso).

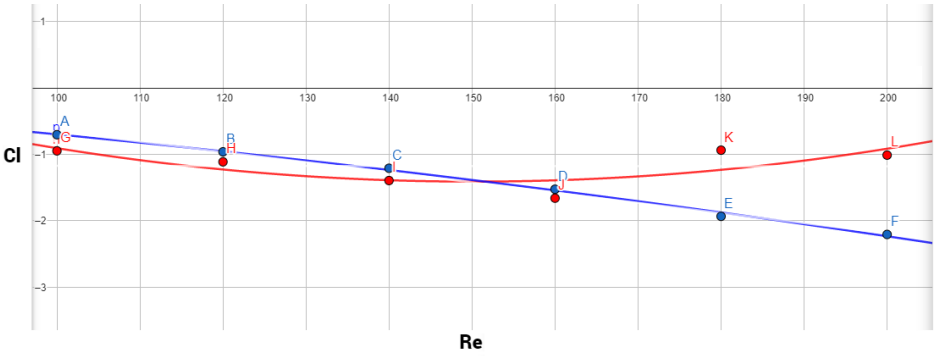


Figura 4.12: Coefficiente di portanza minimo in funzione del Reynolds. Rete Neurale (blu) Ansys (rosso).

Vengono ora proposte delle tabelle riassuntive raffiguranti i dati riportati nei grafici precedenti, inoltre è aggiunto un confronto tra i risultati ottenuti con la rete e un benchmark di riferimento[7]

| $C_D$ | Minimo | Medio | Massimo | Reynolds |
|---|---|---|---|---|
| Ansys | 3.174 | 3.205 | 3.236 | 100 |
| R.N. | 3.0286 | 3.0843 | 3.1356 | 100 |
| Ansys | 3.4033 | 3.4425 | 3.4817 | 120 |
| R.N. | 3.1074 | 3.1827 | 3.2819 | 120 |
| Ansys | 3.7657 | 3.82 | 3.8743 | 140 |
| R.N. | 3.10329074 | 3.4354 | 3.5709 | 140 |
| Ansys | 4.1612 | 4.2156 | 4.27 | 160 |
| R.N. | 3.7548 | 3.8979 | 4.0298 | 160 |
| Ansys | 4.6 | 4.6261 | 4.6522 | 180 |
| R.N. | 4.0778 | 4.236 | 4.3686 | 180 |
| Ansys | 5.160 | 5.2025 | 5.245 | 200 |
| R.N. | 4.92 | 5.103 | 5.2501 | 200 |

| $C_L$ | Minimo | Medio | Massimo | Reynolds |
|---|---|---|---|---|
| Ansys | -0.948 | -0.076 | 0.796 | 100 |
| R.N. | -0.7049 | 0.1448 | 0.921 | 100 |
| Ansys | -1.1133 | 0.0142 | 1.1417 | 120 |
| R.N. | -0.9625 | 0.0341 | 1.1098 | 120 |
| Ansys | -1.3947 | 0.0235 | 1.4414 | 140 |
| R.N. | -1.2099 | 0.2235 | 1.6746 | 140 |
| Ansys | -1.66 | -0.00875 | 1.6775 | 160 |
| R.N. | -1.5254 | 0.0856 | 1.8243 | 160 |
| Ansys | -0.9378 | -0.0758 | 1.814 | 180 |
| R.N. | -1.9334 | -0.0758 | 1.814 | 180 |
| Ansys | -1.012 | -0.0625 | 0.887 | 200 |
| R.N. | -2.206 | 0.1738 | 1.5479 | 200 |

| $C_D$ | Minimo | Medio | Massimo | Reynolds |
|---|---|---|---|---|
| Bench.[7] | -1.021 | -0.017 | 0.986 | 100 |
| R.N. | -0.7049 | 0.1448 | 0.921 | 100 |

| $C_L$ | Minimo | Medio | Massimo | Reynolds |
|---|---|---|---|---|
| Bench.[7] | 3.157 | 3.188 | 3.220 | 100 |
| R.N. | 3.0286 | 3.0843 | 3.1356 | 100 |

### 4.2.5 Visualizzazione dei campi di moto ed estrapolazione qualitativa su nuove geometrie

In questa sotto-sezione verrà da prima presentato un confronto tra i risultati ottenuti tramite ansys e le predizioni della rete neurale per il caso del cilindro a numero di Reynolds pari a 100. Successivamente verrà presentato un confronto per il caso di un profilo alare. Questo caso è di particolare interesse, infatti durante il training alla rete sono state presentate solo delle geometrie primitive cilindriche e rettangolari (in differenti posizioni), nonostante ciò, la rete è riuscita a prevedere correttamente (da un punto di vista qualitativo) il comportamento del flusso intorno alla nuova geometria presentata. Questo risultato dimostra ulteriormente le potenzialità di generalizzazione che questa tipologia di rete neurale offre.

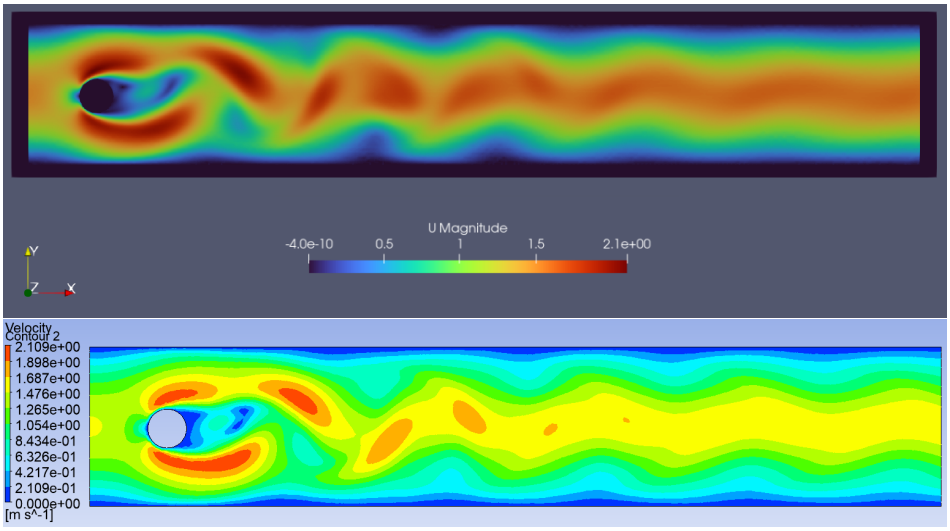


Figura 4.13: Modulo velocità caso Re=100 confronto tra rete neurale (sopra) e Ansys (sotto).

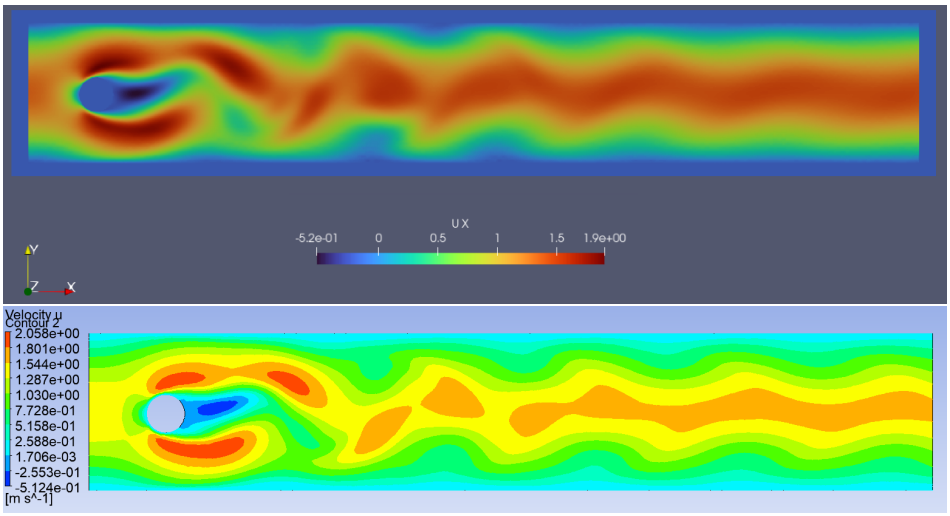


Figura 4.14: Componente x velocità caso Re=100 confronto tra rete neurale (sopra) e Ansys (sotto).

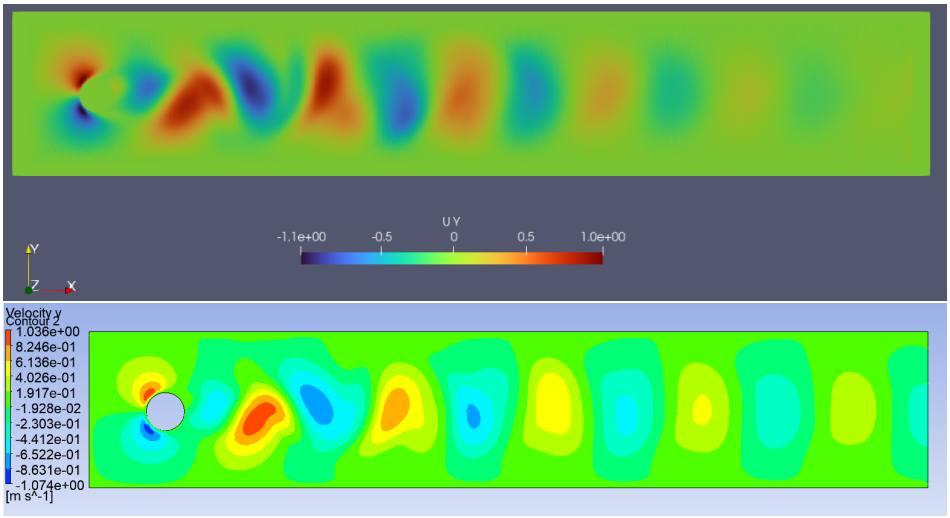


Figura 4.15: Componente y velocità caso Re=100 confronto tra rete neurale (sopra) e Ansys (sotto).

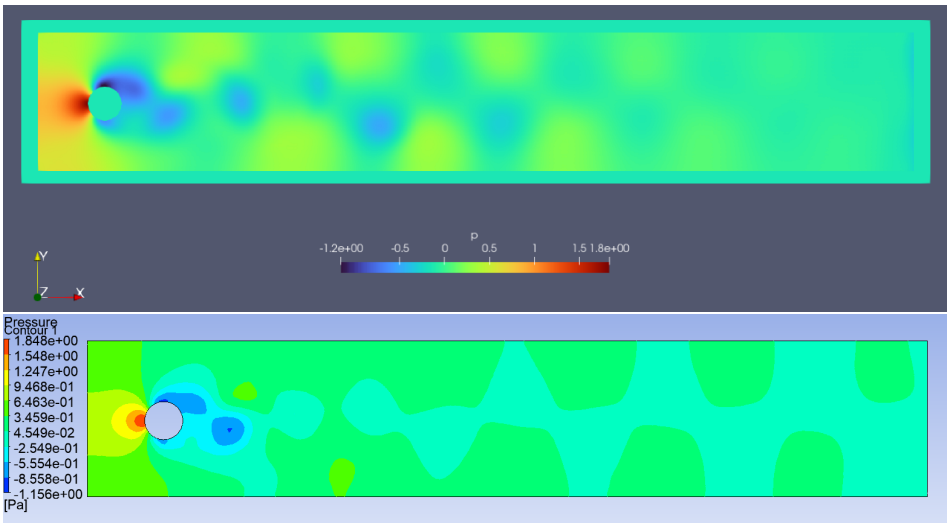


Figura 4.16: Pressione caso Re=100 confronto tra rete neurale (sopra) e Ansys (sotto).

Viene ora presentato il caso del profilo, è interessante notare come la rete sia in grado di riconoscere le zone di depressione in presenza di una superficie curva e il conseguente incremento di velocità. Inoltre viene catturata la scia dietro il corpo e il punto di massima pressione viene localizzato nel punto di arresto della corrente. L'estrapolazione effettuata dalla rete può essere interpretata come una composizione delle geometrie primitive utilizzate durante il training, in questo senso la rete "vede" una geometria come quella nella figura sottostante.

Figura 4.17: Composizione delle geometrie primitive per la creazione del profilo. Rete Neurale (sopra), Ansys (sotto).

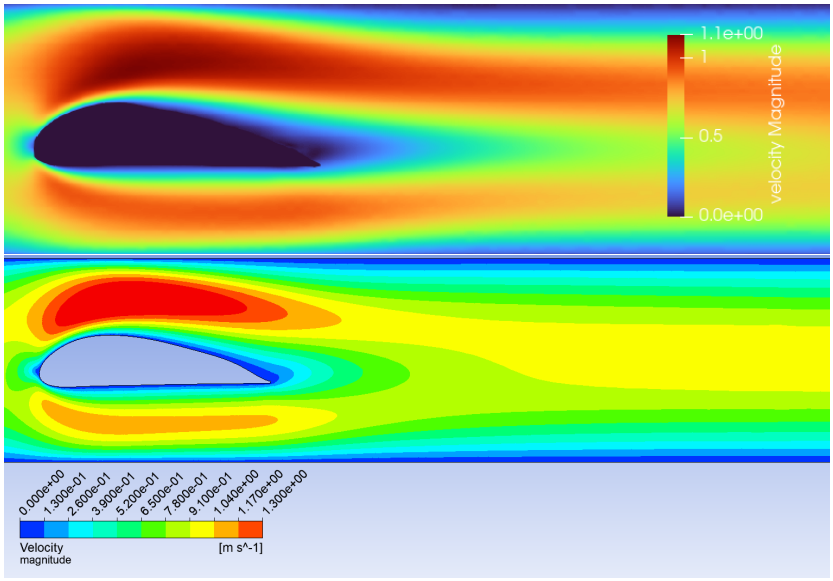


Figura 4.18: Profilo alare caso Re=100: modulo velocità. Rete Neurale (sopra), Ansys (sotto).

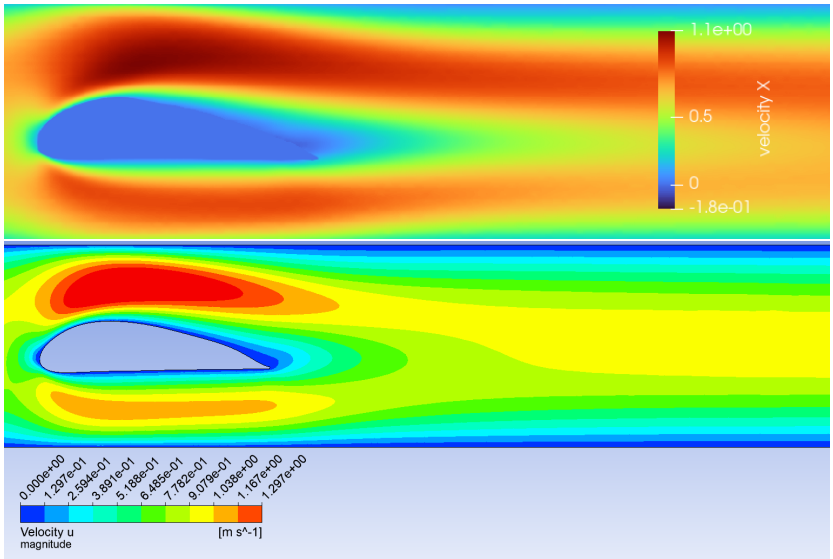


Figura 4.19: Profilo alare caso Re=100: componente orizzontale. Rete Neurale (sopra), Ansys (sotto).

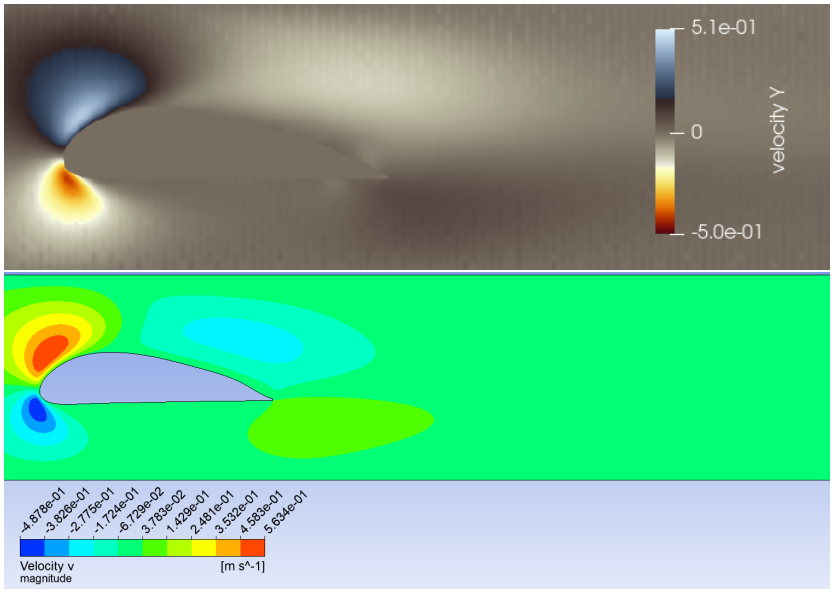


Figura 4.20: Profilo alare caso Re=100: componente verticale. Rete Neurale (sopra), Ansys (sotto).

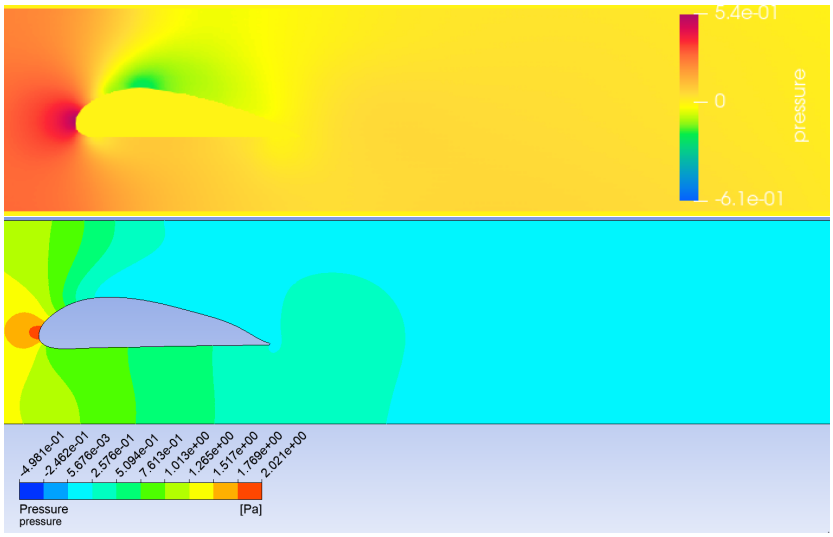


Figura 4.21: Profilo alare caso Re=100: pressione. Rete Neurale (sopra), Ansys (sotto).

### 4.2.6 Analisi dei risultati

I risultati prodotti utilizzando la rete U-Net abbinata all'approccio Spline-PINN risultano un grosso passo avanti rispetto ai risultati ottenuti tramite una rete fully-connected. Non solo è stato possibile estendere il calcolo ad un caso instazionario ma inoltre la rete ha dimostrato la capacità di generalizzazione su diverse configurazioni del test-case scelto, riuscendo a prevedere i coefficienti di forza per diversi valori del numero di Reynolds, maggiori rispetto al valore massimo raggiunto con l'architettura fully-connected.

L'aspetto più interessante da notare è il salto di qualità ottenuto cambiando l'approccio sia per quanto riguarda il training sia per quanto riguarda l'architettura. La procedura di training riesce a migliorare le qualità di generalizzazione della rete tramite l'addestramento su geometrie primitive, grazie alle quali la rete riesce ad imparare i comportamenti del flusso in determinate situazioni riuscendo successivamente ad e estrapolare dei comportamenti su geometrie più complesse, nella sotto-sezione seguente verrà dato un esempio di questa caratteristica. L'architettura invece gioca un ruolo fondamentale sulla capacità di astrazione della rete, ad esempio tramite l'utilizzo di filtri che catturino vari comportamenti del campo di moto. Inoltre la geometria viene gestita tramite una mappa binaria data in input alla rete stessa, questo permette di non dover effettuare un campionamento della geometria. Il vantaggio di questo approccio è duplice, infatti la rete riceve tutto il dominio nella sua interezza senza dipendere dalla quantità di punti utilizzati (batch size) per campionare la geometria, questo permette, inoltre, di utilizzare meno memoria video della scheda grafica permettendo l'utilizzo di reti più complesse anche senza l'adozione di hardware costoso e settoriale. L'introduzione della variabile temporale, tramite questo approccio, ha l'unico effetto di introdurre un nuovo termine nella costruzione della funzione spline approssimante ma non genera nessun aumento di memoria utilizzata.

L'ultimo vantaggio evidenziato da questa tecnica si riscontra nel confronto delle tempistiche di risoluzione con un risolutore "classico" CFD. Infatti nella generazione degli andamenti riportati nella sezione precedente, il risolutore Ansys ha impiegato circa 30 minuti per ogni caso (6 in totale) per un tempo totale di 3 ore circa. I risultati prodotti con la rete neurale sono stati ottenuti in tempo reale, ad ogni modo il tempo di training è stato di circa 10 ore. La differenza di tempo risulta essere ristretta rispetto al caso fully-connected e aumentando il numero di geometrie utilizzate, tramite un passo più piccolo di variazione per il diametro ad esempio, è sicuramente possibile colmare il gap temporale tra i due approcci. Inoltre la rete utilizzata è stata allenata per adattarsi a geometrie che esulano il caso del cilindro, un approccio più mirato per la geometria cilindrica potrebbe richiedere un tempo di training minore.

# Conclusione

In questa tesi sono state analizzate delle nuove tecniche di machine learning applicate per la risoluzione di equazioni differenziali alla derivate parziali, nello specifico per la risoluzione delle equazioni di Navier-Stokes. La risoluzione di tali equazioni è un problema cardine della CFD ed in genere richiede ingente potenza di calcolo e tempo macchina. L'analisi è stata concentrata su un nuovo paradigma che si sta aprendo nella risoluzione delle equazioni di N-S, le tecniche PINN. Si è quindi andanti a costruire una rete neurale di tipo fully-connected, utilizzando le funzioni sigmoidi come unità neuronali. La rete è stata addestrata utilizzando come funzione di loss le equazioni stesse, la rete ha previsto la grandezze fisiche presenti nell'equazione, quest'ultima è stata la funzione da minimizzare tramite ricerca a gradiente. Questo approccio si è dimostrato efficace nei casi 1D, dimostrando la capacità della rete di generalizzare su un dato set di parametri. Successivamente la stessa architettura è stata applicata ad un caso 2D riuscendo a prevedere le soluzione ad un livello, almeno, qualitativamente soddisfacente, però non è stato possibile notare nessuna capacità di generalizzazione. Inoltre la rete per limitazioni hardware è stata "rilegata" alla risoluzione delle sole equazioni stazionarie e per un numero di Reynolds relativamente basso. L'architettura fully-connected è risultata inefficace nel caso 2D, portando a lunghi tempi di training con risultati di scarsa qualità predittiva, in sostanza non si è evidenziato nessun possibile vantaggio nell'utilizzo di questo approccio rispetto alle tecniche di risoluzione classica, da notare però la natura totalmente mesh-less dell'approccio che quindi ben si pone verso problemi fortemente dimensionali. Nell'ultimo capitolo invece si è provato a migliorare le performance di questo approccio, nella fattispecie e si è introdotta una architettura più complessa utilizzando la rete U-Net e si è cambiata la procedura di training. Inoltre alla rete non è stata più affidata la previsione delle grandezze fisiche desiderate (velocità e pressione) ma si è introdotta una interpolazione della soluzione tramite delle Hermite-

Spline, alla rete è stato affidato il compito di computare i coefficienti del polinomio (a tratti) interpolante. Questo approccio ha evidenziato notevoli vantaggi, la gestione della geometria tramite una rete convoluzionale permette di fornire tutto il dominio come input alla rete e la risoluzione dello stesso non è più influenzata dalla quantità di punti ottenuti tramite campionamento, l'assenza del campionamento permette una gestione ottimizzata della memoria video permettendo la soluzione anche del caso instazionario, questo ha comportato un aumento del numero di Reynolds massimo per il quale è possibile effettuare la simulazione. Inoltre è stata definita una procedura di training alternativa basata sull'utilizzo di domini elementari. I risultati hanno mostrato un incremento di prestazioni non trascurabile, infatti la rete non solo ha esteso il range di previsione della rete ma grazie alle spiccate caratteristiche di generalizzazione ha permesso lo studio dei coefficienti di forza su un cilindro parametrizzato e in questo contesto è stato evidenziato come sia possibile colmare il gap temporale di risoluzione tra questo approccio e le tecniche risolutiva classiche.

L'analisi di questa tesi sicuramente evidenzia lo stato ancora embrionale di queste nuove tecniche CFD, ma vengono anche messe in luce le potenzialità di quest'ultime. Sicuramente la capacità di generalizzazione è un punto cruciale su cui si basa il possibile vantaggio rispetto alle tecniche di risoluzione classica. Le reti PINN potranno in futuro essere adottate per procedure di prototipazione veloce. Un software in grado di fornire simulazioni qualitativamente soddisfacenti (in termini di andamenti delle grandezze fisiche) in tempo reale su un variegato insieme di geometrie avrebbe sicuramente una utilità pratica, e sicuramente giustifica la continuazione della ricerca in questo campo. L'idea è quindi creare delle reti neurali in grado di catturare i comportamenti del campo di moto e quindi fornire indicazioni comparative in fase di avanprogetto per poi successivamente concentrare l'analisi tramite tecniche classiche ottenendo valori per il confronto qualitativo. Questo approccio potrebbe ottimizzare le tempistiche di simulazione andando a concentrare le risorse computazionali solo sulle geometrie che risultano essere di interesse. Questa tesi ha concentrato la propria analisi sull'utilizzo di reti neurali senza la necessità di possedere un database iniziale pre-ottenuto, ovvero risolvendo l'equazioni alla pari di un risolutore classico, però queste tecniche, modificando la funzione di loss, possono essere estese all'incorporamento di dati ottenuti da esperimenti o simulazioni precedentemente effettuate. In questo senso quindi è possibile, almeno

in linea teorica, specializzare le reti in determinati contesti, ad esempio un'azienda potrebbe incorporare i risultati ottenuti in galleria del vento, ottenendo delle simulazioni che tengano conto delle evidenze sperimentali.

Le potenzialità di queste tecniche non sono state analizzate nella loro interezza in quanto le reti neurali risultano essere versatili e adattabili a diversi ambiti come ad esempio nella determinazione dei coefficienti di una equazione differenziale a partire da dei dati pre-ottenuti [15]. Sicuramente nel prossimo futuro verrà determinata una direzione più definita sull'utilizzo di questo approccio per la risoluzione delle Navier-Stokes, e potrebbero evidenziarsi, per lo meno, degli ambiti in cui le reti neurali avranno pari dignità rispetto alle tecniche classiche. Nel breve futuro, invece, si prevede l'adozione di queste tecniche per la creazione di software dedicato alla velocizzazione delle fasi iniziali del progetto aerospaziale. Un altra strada da percorrere è invece quella relativa allo sviluppo di un approccio ibrido. Infatti la possibilità di generare veloci soluzioni approssimanti un campo di moto stazionario, risulta utile per l'inizializzazione degli algoritmi di risoluzione iterativi. Infatti una inizializzazione della soluzione più accurata comporta tempi di calcolo notevolmente ridotti. Questo è stato fatto da Illarramendi et al. [5] con ottimi risultati. Nel suo lavoro Illarramendi ha incrementato le prestazioni di un risolutore CFD per il caso incomprimibili di 5-7 volte. Questo è stato possibile grazie all'utilizzo di una rete CNN per prevedere il campo di pressione correttivo utilizzato nell'algoritmo SIMPLE. Sicuramente in futuro si vedranno ulteriori applicazioni, questo è dovuto alla natura estremamente versatile delle reti neurali. Con questa tesi si è voluto evidenziare uno degli aspetti più interessanti, al momento, per il physics informed machine learning.

# Ringraziamenti

Dedico questa tesi alla mia famiglia. A mia madre e mio padre per il loro amore inesauribile e per avermi permesso di realizzare questo grande traguardo. Ai miei fratelli, per me sempre fonte di ispirazione.

Un ringraziamento speciale al mio amico dott. Calogero Zarbo per l'incredibile passione e conoscenza profusa e per avermi spronato, motivato e consigliato durante l'intero sviluppo.

Un sentito grazie al dott. Manuel Carreño Ruiz e al prof. Domenic D'Ambrosio per aver creduto in questa tesi e per avermi supportato durante la stesura.

# Bibliografia


[1] T. Pfaff M. Fortunato A. Sanchez P. W. Battaglia. "Learning mesh based simulation with graph networks". In: *International Conference on Learning Representations (ICLR)* (2021).

[2] Y. Bar-Sinai S. Hoyer J. Hickey M. P. Brenner. "Learning data driven discretizations for partial differential equations". In: *Proceeding of the national Academy of Sciences* (2019).

[3] O. Ronneberger P. Fischer T. Brox. "U-Net Convolutional networks for biomedical image segmentation". In: *International Conference on medical image computing and computer-assisted intervention* (2015).

[4] B. Stevens T. Colonius. "Enhancement of shock capturing methods via machine learning". In: *Theoretical and Computational Fluid Dynamics* (2020).

[5] Illarramendi Bauerheim Cuenot. "Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network". In: *Cambridge University Press* (2021).

[6] M. P. Brenner S. Hoyer D. Kochkov J. A. Smith A. Alieva Q. Wang. "Machine learning accelerated computational fluid dynamics". In: *Proceedings of the national Academy of Science* (2021).

[7] *DFG Benchmark*. URL: https://www.mathematik.tu-dortmund.de/~featflow/en/benchmarks/cfdbenchmarking/flow/dfg_benchmark2_re100.html. (accessed: 01.01.2023).

[8] M. Ainsworth J Dong. "Galerkin neural networks: A framework for approximating variational equations with error control". In: (2021).

[9] R. Pu X. Feng. "Physics Informed Neural Networks for Solving Coupled Stokes-Darcy Equation". In: (2022).

[10] A. Gibiansky. *Convolutional Neural Networks*. URL: https://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks. (accessed: 01.01.2023).

[11] K. Hornik. "Approximation capabilities of multilayer feedfoward networks". In: (1991).

[12] P. V. marcal J. Fong R. Rainsberger. "Least Squares Finite Element Interpolations fo Analysis of Model Free Problems". In: (2016).

[13] M. Raissi P. Perdikaris G. E. Karniadakis. "Physics Informed Deep Learning Part 1 Data-driven Discovery of Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* (2017).

[14] M. Raissi P. Perdikaris G. E. Karniadakis. "Physics Informed Deep Learning Part 1 Data-driven Solutions of Nonlinear Partial Differential Equations". In: *Journal of Computational Physics* (2017).

[15] X. Jin S. Cai H. Li G. E. Karniadakis. "NFSnets (Navier Stokes Flows nets) Physics informed neural networks for the incompressible Navier Stokes equations". In: (2020).

[16] N. Wandel M. Weinmann M. Neidlin R. Klein. "Spline-PINN Approaching PDEs without Data using Fast Physics Informed Hermite Spline CNNs". In: (2022).

[17] N. Wandel M. Weinmann R. Klein. "Learning incompressible fluid dynamics from scratch toward fast, differentiable fluid models that generalize". In: *9th International Conference on Learning Represantations ICLR* (2021).

[18] N. Wandel M. Weinmann R. Klein. "Teaching the incompressible navier-stokes equations to fast neural surrogate models in 3d". In: *International Conference on Learning Represantations ICLR* (2021).

[19] T. Kurbiel. *Deriving the Backpropagation from Scratch Part 2*. URL: https://towardsdatascience.com/deriving-the-backpropagation-equations-from-scratch-part-2-693d4162e779. (accessed: 01.01.2023).

[20] Zhiping Mao, Ameya D. Jagtap e George Em Karniadakis. "Physics-informed neural networks for high-speed flows". In: *Computer Methods in Applied Mechanics and Engineering* (2020).

[21] *Modulus User Guide*. URL: https://docs.nvidia.com/deeplearning/modulus/index.html. (accessed: 01.01.2023).

[22] M. Nielsen. *Neural Network and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/. (accessed: 01.01.2023).

[23] Y. Zhu N. Zabaras P. Koutsourelakis P. Perdikaris. "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data". In: (2019).

[24] M. Raissi, P. Perdikaris e G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* (2019).

[25] S. Sahoo. *Deciding optimal kernel size for CNN*. URL: https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363/. (accessed: 01.01.2023).

[26] A. G. Baydin B. A. Pearlmutter A. A. Radul J. M. Siskind. "Automatic Differentiation in machine Learning a Survey". In: *Journal of Machine learning Research* (2018).

[27] J. Sirignano K. Spiliopoulos. “DGM A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* (2018).

[28] D. Bertsekas J. Tsitsiklis. “Gradient convergence in gradient methods via errors”. In: (2000).