# Valutazione

# Valutazione

## Premessa

Procediamo alla valutazione delle performance con i seguenti assunti:

1. Conosciamo l'obbiettivo da raggiungere.
2. Possediamo i dati: sintetici o meno.
3. I dati sono stati convertiti in tensori.
4. I dati sono stati opportunamente pre-processati e normalizzati.
5. È stata definita la rete, i layer che la costituiscono e come eseguirli.
6. Il loop di addestramento è stato completato.

Il prossimo passo è la valutazione.

# Valutazione

## Premessa

Per valutare un modello si utilizzano diversi parametri; il termine utilizzato per descriverli è:

***Metriche***

*Ma…quali metriche utilizzare?*

La scelta, generalmente:

- È correlata al problema affrontato.
- È legata a ciò che, di quest'ultimo, vogliamo valutare.

# Valutazione

## Premessa

Esiste una prima distinzione che possiamo fare sulle metriche:

Metriche raccolte e monitorate **durante la fase di addestramento.**

*…Mano a mano che gli step completano le epoche e le epoche portano alla fine dell'apprendimento…*

---

Metriche raccolte e monitorate **ad addestramento completato.**

# Valutazione

## Fase di training

**Durante l'addestramento** è possibile tenere traccia di misurazioni che vengono ricalcolate o aggiornate *step* dopo *step*, *epoca* dopo *epoca*.

*Perché fare questo monitoraggio continuo?*

Il monitoraggio continuo di queste metriche è uno dei metodi migliori per:

- Prevedere come una rete sta apprendendo.
- Anticipare un comportamento anomalo.
- Modificare dinamicamente le modalità di addestramento.

# Valutazione

## Fase di training

Fra le metriche monitorate in fase di addestramento, la principale è la ***loss*** o *funzione di costo*.

Rappresenta un riassunto dell'errore commesso dalla rete rispetto a ciò che ci si attendeva e dall'analisi del trend di questa metrica possiamo scoprire:

- Un trend ad andamento decrescente.
- Un trend ad andamento similare ad un'oscillazione smorzata.
- Un trend ad andamento divergente.

E, ad ognuno di questi, darvi un significato per capire cosa sta realmente accadendo all'addestramento.

# Valutazione

## Fase di validazione

La **validazione** è una fase dell'addestramento che di norma:

- Si può eseguire ad ogni epoca o ogni *N* epoche.
- Si può esegue ogni *K* step.
- Utilizza una raccolta di campioni **mai visti in addestramento**.
- Permette di dare uno sguardo al 'mondo reale', uscendo dall'ambito dei soli dati di training.

Anche nella validazione, la principale metrica monitorata è la *loss*.

# Valutazione

## Fase di validazione

Sapendo quindi che le metriche possono essere calcolate sia su dati di addestramento che su dati di validazione, è importante cogliere il significato del perché questo si faccia.

*Calcolare e mettere a confronto metriche, come la **loss**, di training e di validazione permette di comprendere se l'addestramento sta dando buon esito **sia sui dati visti** che **su quelli mai visti**.*

Una rete capace di fare questo è una rete che ha acquisito dei concetti e, con questi, ha acquisito la **capacità di generalizzarli nel mondo reale.**

# Valutazione

## Fase di test

La **fase di test** avviene al termine della fase di addestramento.

Come per la fase di validazione:

- Valuta la conoscenza appresa su campioni mai visti in addestramento.
- Permette di dare uno sguardo al ‘mondo reale’.

***Nota**: Tanto per la validazione, quanto per la fase di test, i campioni utilizzati devono essere rappresentativi della realtà, e variabili, tanto quanto lo sono i campioni di addestramento.*

# Valutazione

## Fase di test

La metriche calcolabili in **fase di test** sono, in genere, le stesse utilizzate in fase di addestramento e validazione.

Prima fra tutte, la *loss*.

Idealmente, al termine di tutte queste fasi:

- Il trend visto in validazione dovrà somigliare al trend visto durante il test.
- La capacità di generalizzare i concetti sui campioni di validazione dovrà rispecchiarsi anche sui campioni di test.

Trend sulla stessa metrica, in disaccordo fra loro, sono i principali spunti di riflessione che mettono in discussione la rete e il modo in cui sta venendo addestrata.

# Valutazione

## Overfitting e underfitting

Ponendo lo sguardo sulla *loss* e andandone ad analizzare il trend, si potranno anticipare in particolare due situazioni critiche ai fini dell'apprendimento della rete:

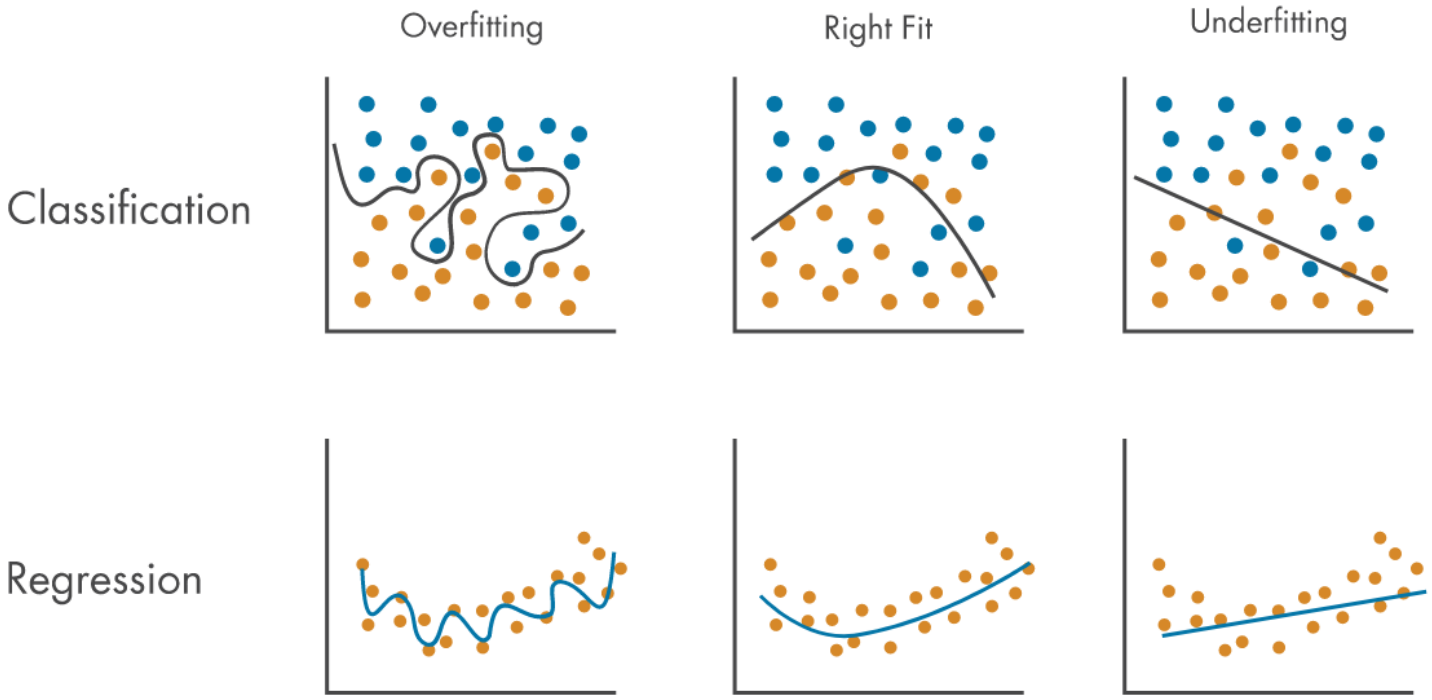
- ***Overfitting***: le performance sui campioni di validazione e/o test non sono «buone» come quelle sui campioni di addestramento. Il modello si è adattato ai campioni di addestramento tanto da raggiungere una grande confidenza nel dare risposte ottime su questi ultimi ma una scarsa capacità di generalizzare i concetti su dati mai visti.

- ***Underfitting***: le performance sui campioni di validazione e/o test risultano migliori di quelle sui campioni di addestramento. La rete non ha appreso abbastanza dai dati di addestramento.

# Valutazione

## Overfitting e underfitting

Di seguito una rappresentazione visiva dei concetti di underfitting (un apprendimento sommario), overfitting (un apprendimento aggressivo, a «mente» chiusa) e fitting corretto:



# Valutazione

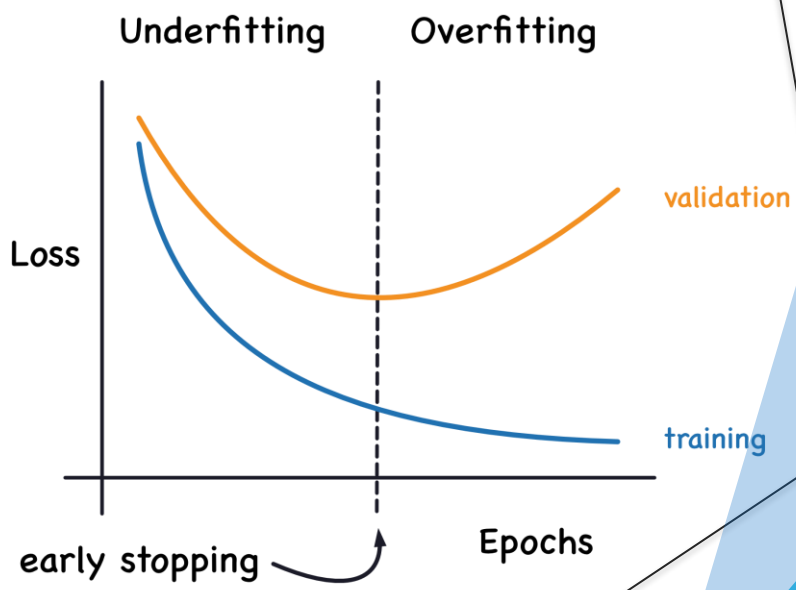
## Fermarsi al momento giusto

Assumiamo di avere un dataset **perfetto**, **bilanciato**, **numeroso**…e in addestramento di monitorare, insieme, la *loss* di training e validazione.

Dall'analisi è possibile trovare un «momento» in cui è opportuno terminare in anticipo l'addestramento chiamato, generalmente, **punto di equilibrio**.

*Qui* si noterà come:

- È possibile migliorare l'apprendimento sui campioni di training.
- Ma quelli di validazione mostrano un trend opposto.




Identificare il punto e decidere di fermarsi, è l'applicazione della tecnica di **Early Stop**.

# Valutazione

## Risolvere il problema

Diverse sono le tecniche che si possono mettere in pratica per tentare di risolvere il problema dell'**overfitting**.

Le principali:

- **Data augmentation**:
  Permette di aggiungere variabilità al dataset di training per evitare che il modello si specializzi su questi dati.

- **Dropout**:
  Obbliga la rete a non concentrarsi sull'utilizzare parametri o strade «privilegiate» per giungere all'obbiettivo imponendo un adattamento a randomiche deviazioni dell'apprendimento.

# Valutazione

## Risolvere il problema

Diverse sono le tecniche che si possono mettere in pratica per tentare di risolvere il problema dell'**underfitting**.

Le principali:

- Aumentare la durata dell'addestramento.
- Modificare gli iper-parametri della rete:

  *…numero di epoche, learning rate, momentum…*
- Aumentare il numero di campioni del dataset.
- Aumentare la complessità della rete.
- …

# Valutazione

## Metriche comuni

Per valutare il modello e, in particolare, come questo si comporta sui dati di validazione e/o di test, si sfruttano le *metriche*.

Ve ne sono molteplici; le più comuni:

- *Confusion Matrix*.
- *Accuracy*.
- *Balanced accuracy*.
- *Precision*.
- *Average precision*.
- *Recall*.
- *Curva ROC*.
- *Curva Precision-Recall*.

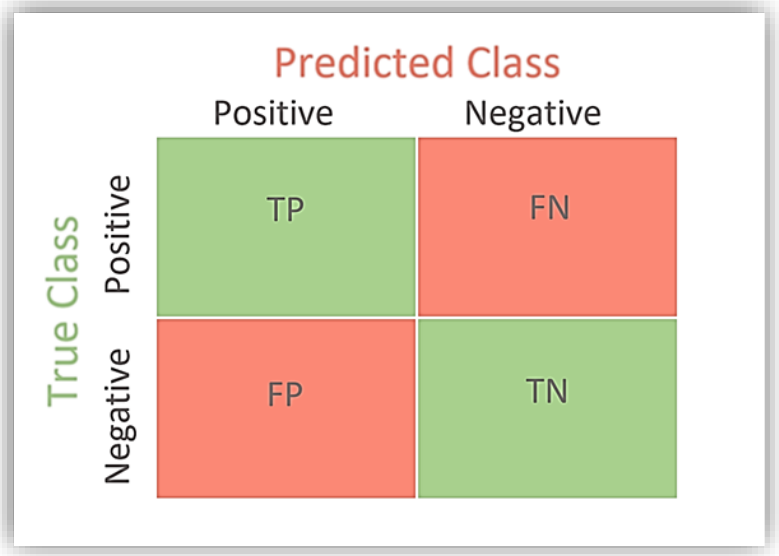
# Valutazione

## Confusion Matrix

Utilizzata soprattutto per i task di classificazione.

Permette di valutare quanti campioni sono stati classificati correttamente e di confrontare la classi l'una contro le altre.

- *TP (True positive)*:
  Campioni positivi classificati positivi.
- *TN (True Negative)*:
  Campioni negativi classificati negativi.
- *FN (False Negative)*:
  Campioni positivi classificati negativi.
- *FP (False Positive)*:
  Campioni negativi classificati positivi.



# Valutazione

## Confusion Matrix

In caso di $n$ classi la confusion matrix sarà una tabella $n \times n$, che mostra quanti campioni sono stati classificati correttamente e quanti no.

Ogni classe verrà messa a confronto con tutte le altre.

| | | Predicted | | | |
|---|---|---|---|---|---|
| | | $C_1$ | $C_2$ | ⋯ | $C_n$ |
| Actual | $C_1$ | $N_{11}$ | $N_{12}$ | ⋯ | $N_{1n}$ |
| | $C_2$ | $N_{21}$ | $N_{22}$ | ⋯ | $N_{2n}$ |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | $C_n$ | $N_{n1}$ | $N_{n2}$ | ⋯ | $N_{nn}$ |

# Valutazione

## Accuracy, Precision e Recall

Dalla confusion matrix si ottengono i valori precedentemente indicati come:

*TP*, *TN*, *FP*, *FN*.

Partendo da questi si possono calcolare metriche come:

***Accuracy:***

Misura quanti campioni sono stati classificati correttamente.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

***Precision:***

Misura quanti campioni positivi sono stati classificati correttamente rispetto a tutti i campioni che sono stati classificati positivi.

$$Precision = \frac{TP}{TP + FP}$$

***Recall:***

Misura quanti campioni positivi sono stati classificati correttamente rispetto a tutti i campioni realmente positivi.

$$Recall = \frac{TP}{TP + FN}$$

# Valutazione

## F1-Score, Sensitivity e Specificity

Dalla confusion matrix si ottengono i valori precedentemente indicati come:

*TP*, *TN*, *FP*, *FN*.

Partendo da questi si possono calcolare metriche come:

***F1-Score:***

Misura il bilanciamento tra precision e recall.

$$F1Score = 2\frac{precision * recall}{precision + recall}$$

***Sensitivity:***

Detto anche TPR (True Positive Rate). Equivale alla recall.

$$Sensitivity = TPR = \frac{TP}{TP + FN}$$

***Specificity:***

Detto anche TNR (True Negative Rate). Misura quanti negativi sono stati classificati correttamente rispetto a tutti i campioni realmente negativi.

$$Specificity = TNR = \frac{TN}{TN + FP}$$

# Valutazione

## Balanced accuracy

Nel caso in cui i dati non siano bilanciati, è opportuno calcolare delle metriche che tengano conto dello sbilanciamento.

In particolare, invece dell'*accuracy* se ne andrà ad utilizzare una versione «bilanciata»:

*Balanced accuracy:*

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2}$$

$$Sensitivity = \frac{20}{20 + 70} = 0.22$$

$$Specificity = TNR = \frac{5000}{5000 + 30} = 0.99$$

$$BalancedAccuracy = \frac{0.22 + 0.99}{2} = 0.60$$

| | Positive | Negative |
|---|---|---|
| **Positive** | 20 (*TP*) | 30 (*FP*) |
| **Negative** | 70 (*FN*) | 5000 (*TN*) |

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{5020}{5120} = 0.98$$

# Valutazione

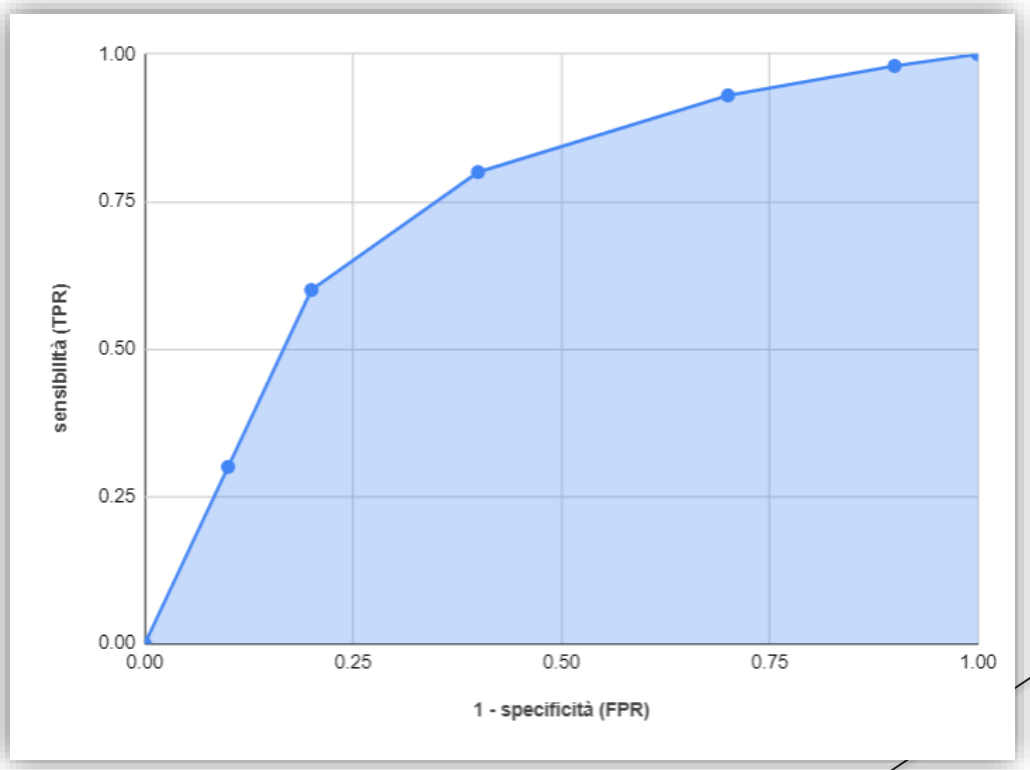
## Curva ROC

Acronimo di *Receiver Operating Characteristic*, traccia i cambiamenti di TPR ed FPR per diverse soglie di classificazione.

Sulla curva:

- Ogni punto rappresenta una soglia.
- Il punto in alto a sinistra rappresenta la migliore performance del modello.

Quella in cui ***TPR è massimo*** e ***FPR è minimo***.



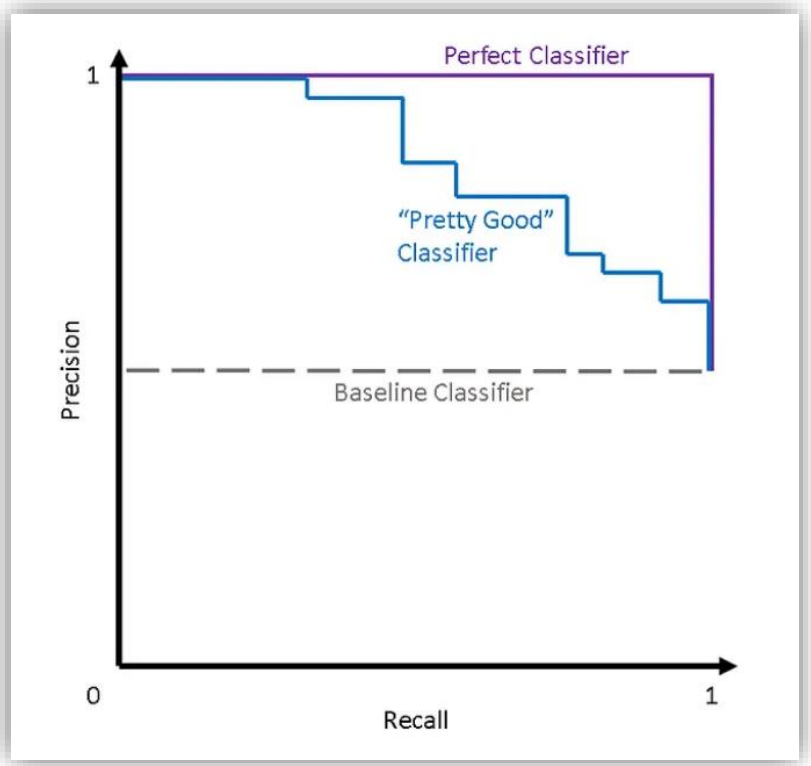
# Valutazione
## Curva ROC

Rappresenta la relazione tra la Precision e la Recall per diverse soglie di classificazione.

Sulla curva:

- Ogni punto rappresenta una soglia.
- Il punto in alto a destra rappresenta la migliore performance del modello.

Quella in cui la ***precision è massima*** e la ***recall è massima***.



# Valutazione

## Curve ROC e PR: casi d'uso

Usate, generalmente, per:

- Confrontare le prestazioni di diversi modelli fra loro.
- Trovare la soglia ottima di classificazione.

In genere:

- La curva **ROC** è più indicata quando il dataset ha un numero elevato di negativi rispetto ai positivi.
- La curva **PR** è più indicata quando il dataset ha un numero elevato di positivi rispetto ai negativi.
- Un buon modello di classificazione dovrebbe avere l'area sotto la ROC curve (AUC-Area Under the Curve) e l'area sotto la PR curve (AUPRC-Area Under the Precision-Recall Curve) vicine a 1.

In questi casi il modello è in grado di classificare correttamente la maggior parte delle istanze.

# PyTorch testing loop

```python
# Setup empty lists to keep track of model progress
epoch_count = []
train_loss_values = []
test_loss_values = []

# Pass the data through the model for a number of epochs (e.g. 100) pochs):
for epoch in range(epochs):

    ### Training loop code here ###

    ### Testing starts ###

    # Put the model in evaluation mode
    model.eval()

    # Turn on inference mode context manager
    with torch.inference_mode():
        # 1. Forward pass on test data
        test_pred = model(X_test)

        # 2. Caculate loss on test data
        test_loss = loss_fn(test_pred, y_test)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        epoch_count.append(epoch)
        train_loss_values.append(loss)
        test_loss_values.append(test_loss)
        print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")
```

*Note: all of this can be turned into a function*

Create empty lists for storing useful values (helpful for tracking model progress)

Tell the model we want to evaluate rather than train (this turns off functionality used for training but not evaluation)

Turn on `torch.inference_mode()` context manager to disable functionality such as gradient tracking for inference (gradient tracking not needed for inference)

Pass the test data through the model (this will call the model's implemented `forward()` method)

Calculate the test loss value (how wrong the model's predictions are on the test dataset, lower is better)

Display information outputs for how the model is doing during training/testing every ~10 epochs (note: what gets printed out here can be adjusted for specific problems)

# Proviamo?