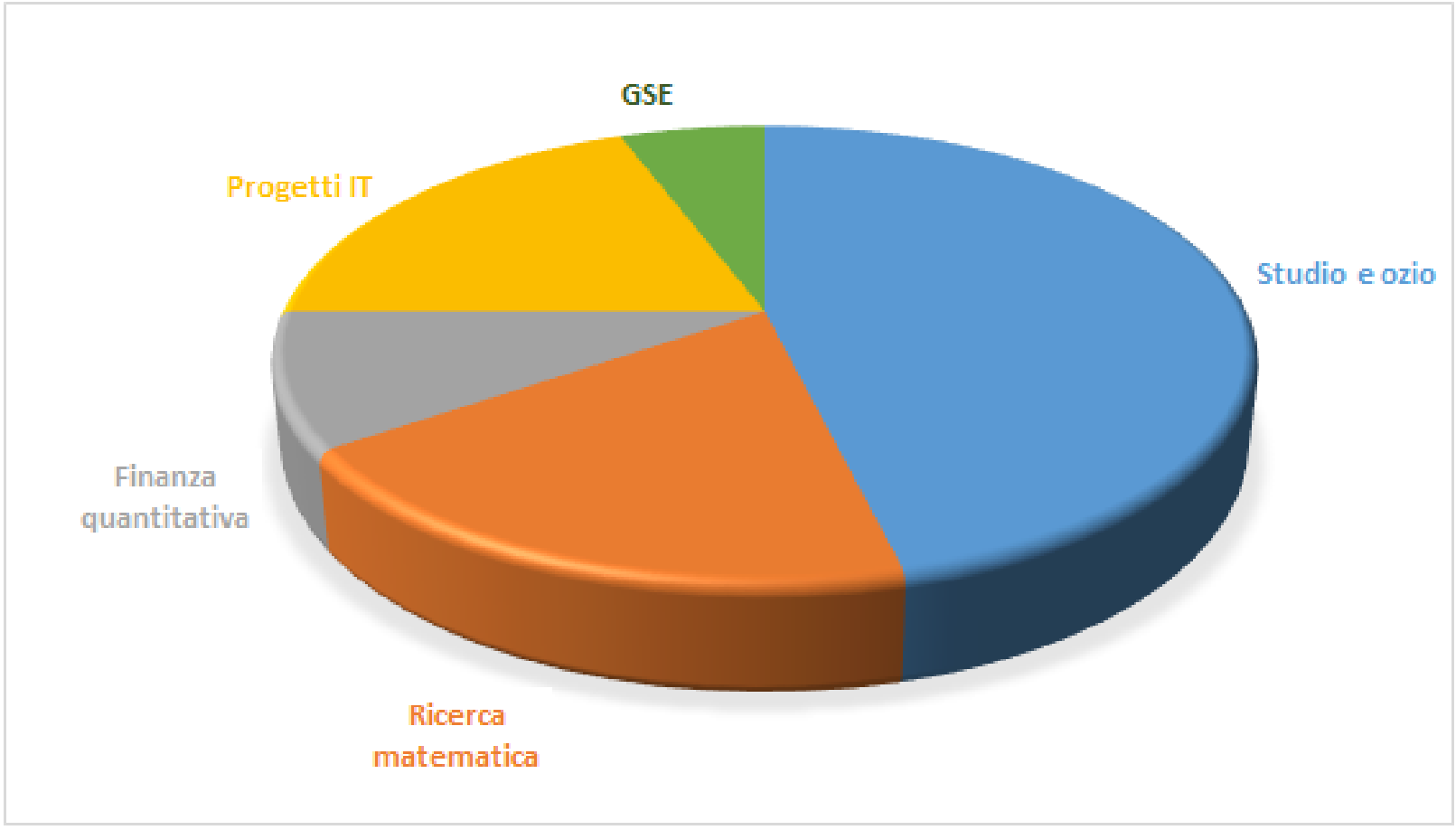
DEVCAST

Imparare a generare: introduzione alle GAN

Paolo Caressa, PhD

Chi vi parla: una vita con... torta



A che servono le reti neurali?

Le reti neurali sono algoritmi numerici di ottimizzazione non lineare che consentono di **predire** e **classificare**.

Queste attività si svolgono nel ML anche senza le reti (es: regressione, SVM, ...) ma le reti servono a entrambi gli scopi in modo egregio.

Le reti possiedono al loro interno dei parametri (pesi) che con algoritmi iterativi di ottimizzazione (discesa gradiente &Co) vengono fatti convergere (apprendimento) verso la minimizzazione degli errori commessi nell'interpolare dati di input su dati di output.

A che servono le reti neurali?

Le reti neurali sono algoritmi numerici di ottimizzazione non lineare che consentono di **predire** e **classificare**.

Queste attività si svolgono nel ML anche senza le reti (es: regressione, SVM, ...) ma le reti servono a entrambi gli scopi in modo egregio.

Le reti possiedono al loro interno dei parametri (pesi) che con algoritmi iterativi di ottimizzazione (discesa gradiente &Co) vengono fatti convergere (apprendimento) verso la minimizzazione degli errori commessi nell'interpolare dati di input su dati di output.

A che servono le reti neurali?

Le reti neurali sono algoritmi numerici di ottimizzazione non lineare che consentono di **predire** e **classificare**.

Queste attività si svolgono nel ML anche senza le reti (es: regressione, SVM, ...) ma le reti servono a entrambi gli scopi in modo egregio.

Le reti possiedono al loro interno dei parametri (pesi) che con algoritmi iterativi di ottimizzazione (discesa gradiente &Co) vengono fatti convergere (apprendimento) verso la minimizzazione degli errori commessi nell'interpolare dati di input su dati di output.

Nessuno è perfetto, nemmeno una rete neurale

Una rete neurale usata per discriminare o predire offre risultati eccellenti ma ovviamente non ci sono pasti gratis:

- Per l'addestramento servono generalmente training set supervisionati
- Peggio, servono enormi training set.
- Peggio ancora, occorre empiricamente calibrare degli iperparametri senza una reale teoria che dica come fare.

Alcune varianti o applicazioni particolari delle reti neurali (predizione di serie storiche) sono non supervisionate

Nessuno è perfetto, nemmeno una rete neurale

Una rete neurale usata per discriminare o predire offre risultati eccellenti ma ovviamente non ci sono pasti gratis:

- Per l'addestramento servono generalmente training set supervisionati
- Peggio, servono enormi training set.
- Peggio ancora, occorre empiricamente calibrare degli iperparametri senza una reale teoria che dica come fare.

Alcune varianti o applicazioni particolari delle reti neurali (predizione di serie storiche) sono non supervisionate

Nessuno è perfetto, nemmeno una rete neurale

Una rete neurale usata per discriminare o predire offre risultati eccellenti ma ovviamente non ci sono pasti gratis:

- Per l'addestramento servono generalmente training set supervisionati
- Peggio, servono enormi training set.
- Peggio ancora, occorre empiricamente calibrare degli iperparametri senza una reale teoria che dica come fare.

Alcune varianti o applicazioni particolari delle reti neurali (predizione di serie storiche) sono non supervisionate

Nessuno è perfetto, nemmeno una rete neurale

Una rete neurale usata per discriminare o predire offre risultati eccellenti ma ovviamente non ci sono pasti gratis:

- Per l'addestramento servono generalmente training set supervisionati
- Peggio, servono enormi training set.
- Peggio ancora, occorre empiricamente calibrare degli iperparametri senza una reale teoria che dica come fare.

Alcune varianti o applicazioni particolari delle reti neurali (predizione di serie storiche) sono non supervisionate

GAN: generare e non creare...

GAN = **generative, adversarial, networks**.

Network = Sono reti neurali (tipicamente profonde).

Generative = producono anziché consumare dati, in particolare generano dati imitando uno specifico **pattern** che apprendono dai dati di input. Non c'è "generazione spontanea" in Natura, anche i dati si producono da altri dati!

Adversarial = la generazione avviene per contrasto, in modo dialettico, tramite la competizione di due reti avversarie. Un concetto eracliteo, hegeliano e marxista.

GAN: generare e non creare...

GAN = **generative, adversarial, networks**.

Network = Sono reti neurali (tipicamente profonde).

Generative = producono anziché consumare dati, in particolare generano dati imitando uno specifico **pattern** che apprendono dai dati di input. Non c'è "generazione spontanea" in Natura, anche i dati si producono da altri dati!

Adversarial = la generazione avviene per contrasto, in modo dialettico, tramite la competizione di due reti avversarie. Un concetto eracliteo, hegeliano e marxista.

GAN: generare e non creare...

GAN = **generative, adversarial, networks**.

Network = Sono reti neurali (tipicamente profonde).

Generative = producono anziché consumare dati, in particolare generano dati imitando uno specifico **pattern** che apprendono dai dati di input. Non c'è "generazione spontanea" in Natura, anche i dati si producono da altri dati!

Adversarial = la generazione avviene per contrasto, in modo dialettico, tramite la competizione di due reti avversarie. Un concetto eracliteo, hegeliano e marxista.

GAN: generare e non creare...

GAN = **generative, adversarial, networks**.

Network = Sono reti neurali (tipicamente profonde).

Generative = producono anziché consumare dati, in particolare generano dati imitando uno specifico **pattern** che apprendono dai dati di input. Non c'è "generazione spontanea" in Natura, anche i dati si producono da altri dati!

Adversarial = la generazione avviene per contrasto, in modo dialettico, tramite la competizione di due reti avversarie. Un concetto eracliteo, hegeliano e marxista.

Perché sono famose e utilizzate?

- Offrono **risultati spettacolari**, creando nuovi dati (immagini, testi, suoni, etc.) a immagine e somiglianza di esempi che sono loro stati mostrati.
- Offrono un **nuovo paradigma** del ML, quello generativo, che combina quelli esistenti in qualcosa di nuovo.
- Sono state create recentemente (2014) da **eminenti esponenti** del DL: Ian Goodfellow (ex Google, ora Apple), Joshua Bengio (uno dei godfather), Aaron Courville (ricordate il loro libro?) e altri.

Perché sono famose e utilizzate?

- Offrono **risultati spettacolari**, creando nuovi dati (immagini, testi, suoni, etc.) a immagine e somiglianza di esempi che sono loro stati mostrati.
- Offrono un **nuovo paradigma** del ML, quello generativo, che combina quelli esistenti in qualcosa di nuovo.
- Sono state create recentemente (2014) da **eminenti esponenti** del DL: Ian Goodfellow (ex Google, ora Apple), Joshua Bengio (uno dei godfather), Aaron Courville (ricordate il loro libro?) e altri.

Perché sono famose e utilizzate?

- Offrono **risultati spettacolari**, creando nuovi dati (immagini, testi, suoni, etc.) a immagine e somiglianza di esempi che sono loro stati mostrati.
- Offrono un **nuovo paradigma** del ML, quello generativo, che combina quelli esistenti in qualcosa di nuovo.
- Sono state create recentemente (2014) da **eminenti esponenti** del DL: Ian Goodfellow (ex Google, ora Apple), Joshua Bengio (uno dei godfather), Aaron Courville (ricordate il loro libro?) e altri.

Perché sono famose e utilizzate?

- Offrono **risultati spettacolari**, creando nuovi dati (immagini, testi, suoni, etc.) a immagine e somiglianza di esempi che sono loro stati mostrati.
- Offrono un **nuovo paradigma** del ML, quello generativo, che combina quelli esistenti in qualcosa di nuovo.
- Sono state create recentemente (2014) da **eminenti esponenti** del DL: Ian Goodfellow (ex Google, ora Apple), Joshua Bengio (uno dei godfather), Aaron Courville (ricordate il loro libro?) e altri.

Componenti di una GAN

Per giungere alla sintesi dialettica della creazione di dati secondo un modello, una GAN contempla due reti neurali interne: una tesi **generatore** e una antitesi **discriminatore**.

Il generatore viene addestrato a produrre un output, da un input casuale, ma apprendendo a produrlo in modo da ingannare il discriminatore. È il falsario...

Il discriminatore viene addestrato a riconoscere un input dal rumore di fondo, e viene usato per smascherare i dati che non sono originari del trainining set ma che sono prodotti dal generatore. È il perito del tribunale...

Componenti di una GAN

Per giungere alla sintesi dialettica della creazione di dati secondo un modello, una GAN contempla due reti neurali interne: una tesi **generatore** e una antitesi **discriminatore**.

Il generatore viene addestrato a produrre un output, da un input casuale, ma apprendendo a produrlo in modo da ingannare il discriminatore. È il falsario...

Il discriminatore viene addestrato a riconoscere un input dal rumore di fondo, e viene usato per smascherare i dati che non sono originari del trainining set ma che sono prodotti dal generatore. È il perito del tribunale...

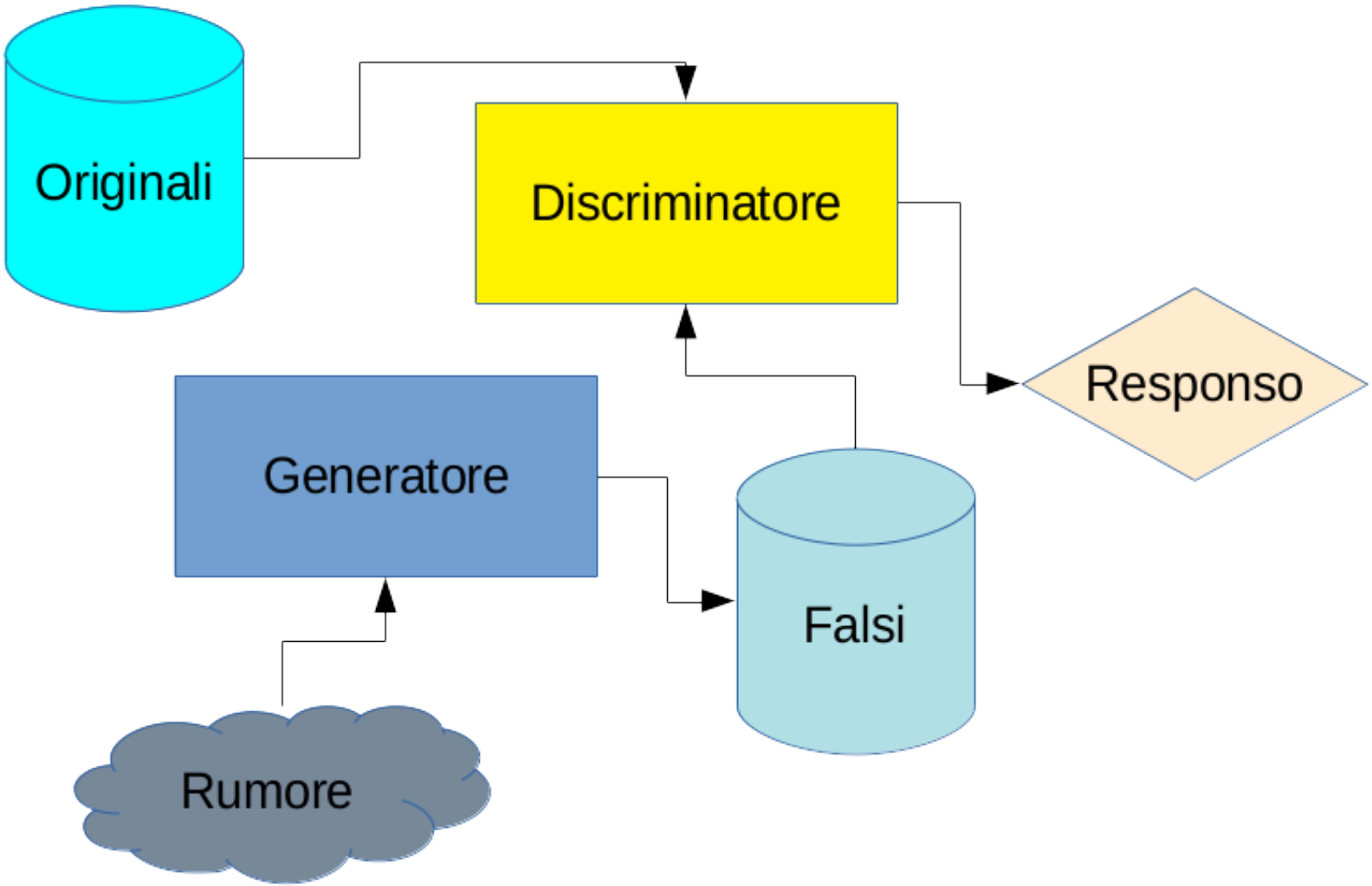
Componenti di una GAN

Per giungere alla sintesi dialettica della creazione di dati secondo un modello, una GAN contempla due reti neurali interne: una tesi **generatore** e una antitesi **discriminatore**.

Il generatore viene addestrato a produrre un output, da un input casuale, ma apprendendo a produrlo in modo da ingannare il discriminatore. È il falsario...

Il discriminatore viene addestrato a riconoscere un input dal rumore di fondo, e viene usato per smascherare i dati che non sono originari del trainining set ma che sono prodotti dal generatore. È il perito del tribunale...

Componenti di una GAN



Esempi di generazione (a destra gli originali)

da Goodfellow; Pouget-Abadie; Mirza; Xu; Warde-Farley; Ozair; Courville; Bengio (2014). *Generative Adversarial Nets*. Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

Codice: https://github.com/goodfeli/adversarial

Esempi di generazione (a destra gli originali)

da Goodfellow; Pouget-Abadie; Mirza; Xu; Warde-Farley; Ozair; Courville; Bengio (2014). *Generative Adversarial Nets*. Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

Codice: https://github.com/goodfeli/adversarial

https://www.thispersondoesnotexist.com/

Codice: https://github.com/NVlabs/stylegan2

Facciamolo anche noi!

Per capire i dettagli di questo schema e come una GAN effettivamente funzioni la cosa migliore è provare a farne una noi!

Poiché le applicazioni più immediate sono alla generazione di immagini ci cimenteremo con questa: per rendere le cose semplici non faremo generare cifre o altro (trovate molti *tutorial* in rete) per non entrare nei dettagli delle CNN necessarie, etc.

Più pigri e inetti, qui costruiremo una GAN che viene addestrata a produrre curve sinusoidali, come quelle della teoria dei segnali, il che può farsi senza usare reti profonde (quindi tempi accettabili anche su un vecchio laptop).

Facciamolo anche noi!

Per capire i dettagli di questo schema e come una GAN effettivamente funzioni la cosa migliore è provare a farne una noi!

Poiché le applicazioni più immediate sono alla generazione di immagini ci cimenteremo con questa: per rendere le cose semplici non faremo generare cifre o altro (trovate molti *tutorial* in rete) per non entrare nei dettagli delle CNN necessarie, etc.

Più pigri e inetti, qui costruiremo una GAN che viene addestrata a produrre curve sinusoidali, come quelle della teoria dei segnali, il che può farsi senza usare reti profonde (quindi tempi accettabili anche su un vecchio laptop).

Facciamolo anche noi!

Per capire i dettagli di questo schema e come una GAN effettivamente funzioni la cosa migliore è provare a farne una noi!

Poiché le applicazioni più immediate sono alla generazione di immagini ci cimenteremo con questa: per rendere le cose semplici non faremo generare cifre o altro (trovate molti *tutorial* in rete) per non entrare nei dettagli delle CNN necessarie, etc.

Più pigri e inetti, qui costruiremo una GAN che viene addestrata a produrre curve sinusoidali, come quelle della teoria dei segnali, il che può farsi senza usare reti profonde (quindi tempi accettabili anche su un vecchio laptop).

Passi necessari

1. Ottenere un dataset di immagini “genuine” di sinusoidi
2. Impostare il piedi generatore e discriminatore
3. Addestrare il discriminatore a fare il suo lavoro
4. Costruire la GAN usando generatore e discriminatore
5. Addestrare la componente generativa della GAN a generare sinusoidi che la componente discriminativa giudichi per tali
6. Contemplare il risultato: una sinusoide prodotta dal rumore di fondo...

Un dataset di sinusoidi

Poiché si tratta di curve facili da descrivere matematicamente, il dataset delle sinusoidi "genuine" possiamo generarlo noi, considerando i grafici delle funzioni

$$y = a \sin(bx + c)$$

al variare dei parametri $a, b, c \in \mathbb{R}$ che determinano rispettivamente l'ampiezza, la frequenza e la fase della sinusoide

Nella slide seguente alcune sinusoidi di questo tipo (tutte della stessa ampiezza):

Un dataset di sinusoidi

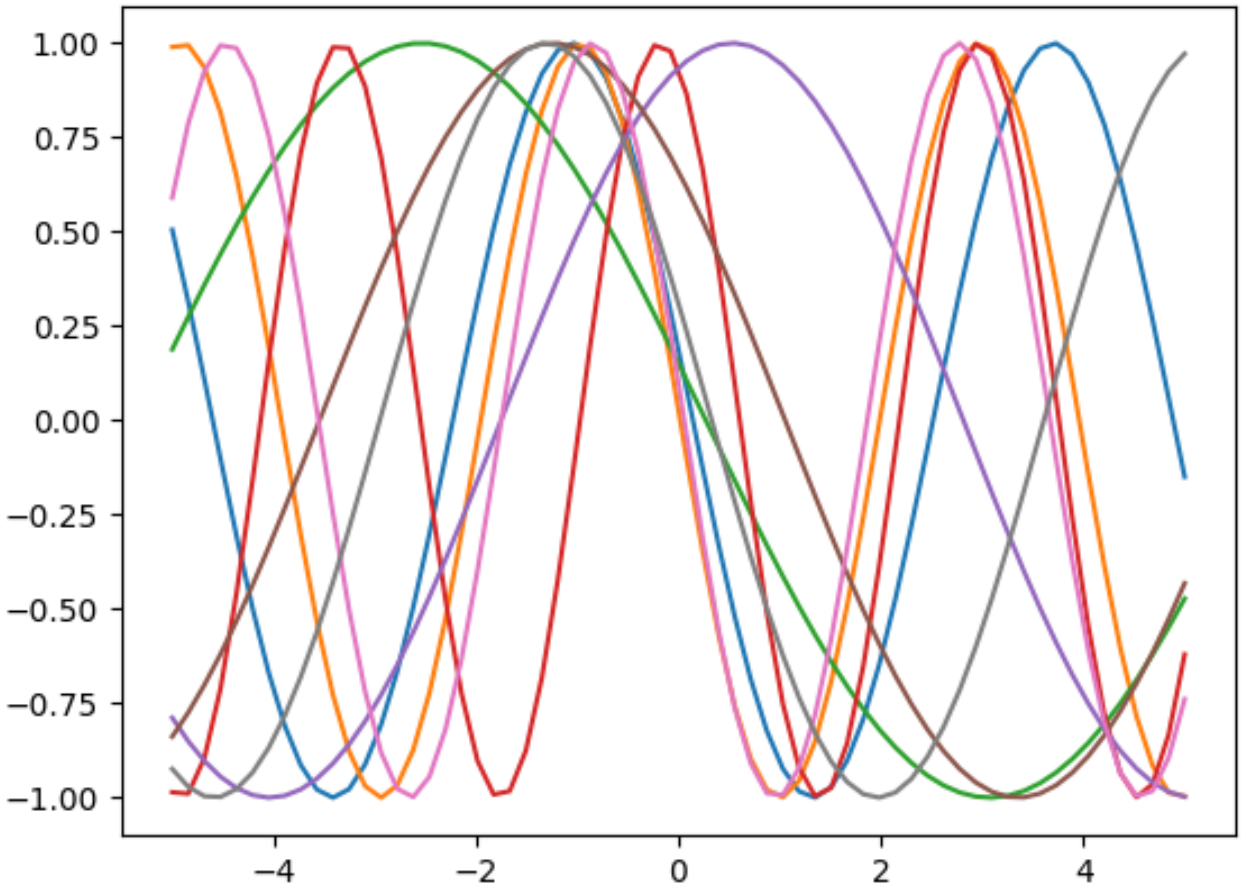
Poiché si tratta di curve facili da descrivere matematicamente, il dataset delle sinusoidi "genuine" possiamo generarlo noi, considerando i grafici delle funzioni

$$y = a \sin(bx + c)$$

al variare dei parametri $a, b, c \in \mathbb{R}$ che determinano rispettivamente l'ampiezza, la frequenza e la fase della sinusoide

Nella slide seguente alcune sinusoidi di questo tipo (tutte della stessa ampiezza):

Esempi di sinusoidi



Generazione del dataset

```
import numpy as np
from numpy.random import randint, uniform
import matplotlib.pyplot as plt

TRAIN_LEN = 64
TRAIN_SIZE = 8192
BATCH = 128
X_MIN, X_MAX = -5.0, 5.0
Y_MIN, Y_MAX = -1.0, 1.0

X_COORDS = np.linspace(X_MIN , X_MAX, TRAIN_LEN)
X_TRAIN = np.zeros((TRAIN_SIZE, TRAIN_LEN))

for i in range(0, TRAIN_SIZE):
    scale = uniform(0.5, 2.0)
    phase = uniform(np.math.pi)
    X_TRAIN[i] = np.array([np.sin(scale*x +
   phase) for x in X_COORDS])
```

Visualizziamo il dataset

Notiamo che `X_TRAIN` è una matrice le cui righe sono i singoli training case e le cui colonne sono i singoli valori di ciascun training case. Poiché ciascun training case è una curva, la rappresentiamo "campionandola" nei punti le cui coordinate sono i valori del vettore `X_COORDS`.

Con il codice seguente possiamo visualizzare alcune curve di questo dataset, come nella figura della slide precedente (prodotta in questo modo).

```
# Disegniamo le prime otto curve
fig, axis = plt.subplots(1, 1)
for i in range(8):
    axis.plot(X_COORDS, X_TRAIN[i])
```

Visualizziamo il dataset

Notiamo che `X_TRAIN` è una matrice le cui righe sono i singoli training case e le cui colonne sono i singoli valori di ciascun training case. Poiché ciascun training case è una curva, la rappresentiamo "campionandola" nei punti le cui coordinate sono i valori del vettore `X_COORDS`.

Con il codice seguente possiamo visualizzare alcune curve di questo dataset, come nella figura della slide precedente (prodotta in questo modo).

```
# Disegniamo le prime otto curve
fig, axis = plt.subplots(1, 1)
for i in range(8):
    axis.plot(X_COORDS, X_TRAIN[i])
```

Creaiamo il discriminatore

Come discriminatore prenderemo una rete neurale con due strati interni: lo strato di input ha tanti neuroni quanti sono gli elementi di ciascun training case, quindi i punti campionati di una curva, mentre lo strato di output ha un solo neurone, dunque emette un numero.

I due stati interni hanno tanti neuroni quanta è la lunghezza del vettore che rappresenta una curva; gli strati hanno come funzione di attivazione la ReLU e uno strato di regolarizzazione con la *dropout* che va a cancellare alcuni neuroni a caso, nella frazione indicata come parametro. Si tratta di una tecnica standard per evitare l'overfitting.

Creaiamo il discriminatore

Come discriminatore prenderemo una rete neurale con due strati interni: lo strato di input ha tanti neuroni quanti sono gli elementi di ciascun training case, quindi i punti campionati di una curva, mentre lo strato di output ha un solo neurone, dunque emette un numero.

I due stati interni hanno tanti neuroni quanta è la lunghezza del vettore che rappresenta una curva; gli strati hanno come funzione di attivazione la ReLU e uno strato di regolarizzazione con la *dropout* che va a cancellare alcuni neuroni a caso, nella frazione indicata come parametro. Si tratta di una tecnica standard per evitare l'overfitting.

Definizione del discriminatore

```
from keras.models import Sequential
from keras.layers import Dense, Dropout

DIS_DROPOUT = 0.4
discriminator = Sequential()
discriminator.add(Dense(TRAIN_LEN, activation =
    "relu"))
discriminator.add(Dropout(DIS_DROPOUT))
discriminator.add(Dense(SAMPLE_LEN, activation =
    "relu"))
discriminator.add(Dropout(DIS_DROPOUT))
discriminator.add(Dense(1, activation = "sigmoid
    "))
discriminator.compile(optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = ["accuracy"])
```

Addestrare il discriminatore

Una volta creata la rete che discrimina, addestriamola a discriminare: per farlo le sottoponiamo un dataset che è per metà formato dal nostro dataset di sinusoidi, per metà da curve a caso, cioè sequenze di numeri a caso sui punti di campionamento. Le sinusoidi hanno label 1, le curve a caso label 0.

Il metodo di Keras `train_on_batch` eseguirà l'addestramento per noi: il training set viene sottoposto alla rete con dei batch di taglia determinata empiricamente per un certo numero di epoche (aumentando le quali ovviamente aumenta l'accuratezza).

Addestrare il discriminatore

Una volta creata la rete che discrimina, addestriamola a discriminare: per farlo le sottoponiamo un dataset che è per metà formato dal nostro dataset di sinusoidi, per metà da curve a caso, cioè sequenze di numeri a caso sui punti di campionamento. Le sinusoidi hanno label 1, le curve a caso label 0.

Il metodo di Keras `train_on_batch` eseguirà l'addestramento per noi: il training set viene sottoposto alla rete con dei batch di taglia determinata empiricamente per un certo numero di epoche (aumentando le quali ovviamente aumenta l'accuratezza).

Addestramento del discriminatore

```
BATCH_SIZE = 128
EPOCHS = 32
ONES = np.ones((BATCH_SIZE//2))    # vettore di
    label 1
ZEROS = np.zeros((BATCH_SIZE//2)) # vettore di
    label 0
ONEZEROS = (ONES, ZEROS)
NOISE = uniform(Y_MIN, Y_MAX, size = (TRAIN_SIZE
    , TRAIN_LEN))
print(" epoca | accuratezza ")
for i in range(EPOCHS):
  # Sceglie BATCH_SIZE//2 indici e li torna in n
  n = randint(0, TRAIN_SIZE, size = BATCH_SIZE
   //2)
  x = np.concatenate((X_TRAIN[n], NOISE[n]))
  y = np.concatenate(ONEZEROS)
  dummy, acc = discriminator.train_on_batch(x, y
   )
  print(f"  {i:3}  |   {acc}")
```

Output dell'addestramento

epoca	accuratezza
0	0.5
1	0.4609375
2	0.5078125
3	0.5078125
4	0.53125
5	0.5625
6	0.625
...	...
25	0.84375
26	0.84375
27	0.828125
28	0.828125
29	0.8359375
30	0.8359375
31	0.890625

Creazione del generatore

Anche il generatore è una rete neurale con uno singolo strato di input che consuma una curva come nel caso del discriminatore, uno strato interno di 256 neuroni (un numero come un altro determinato empiricamente), e uno strato di output della stessa dimensione della curva. Quindi il generatore prende in input una curva (casuale) e genera una curva (sinusoide).

Il generatore non viene addestrato singolarmente in quanto, per quel che abbiamo visto, il suo addestramento è a cura della GAN: quindi il discriminatore "parte avvantaggiato" ed è questo uno dei motivi che fanno funzionare l'addestramento.

Creazione del generatore

Anche il generatore è una rete neurale con uno singolo strato di input che consuma una curva come nel caso del discriminatore, uno strato interno di 256 neuroni (un numero come un altro determinato empiricamente), e uno strato di output della stessa dimensione della curva. Quindi il generatore prende in input una curva (casuale) e genera una curva (sinusoide).

Il generatore non viene addestrato singolarmente in quanto, per quel che abbiamo visto, il suo addestramento è a cura della GAN: quindi il discriminatore "parte avvantaggiato" ed è questo uno dei motivi che fanno funzionare l'addestramento.

Definizione del generatore e della GAN

```
generator = Sequential()
generator.add(Dense(SAMPLE_LEN, activation = "
    relu"))
generator.add(Dense(256, activation = "relu"))
generator.add(Dense(SAMPLE_LEN, activation = "
    tanh"))
generator.compile(optimizer = "adam",
     loss = "mse",
     metrics = ["accuracy"])

gan = Sequential()
gan.add(generator)
discriminator.trainable = False
gan.add(discriminator)
gan.compile(optimizer = "adam",
     loss = "binary_crossentropy",
     metrics = ["accuracy"])
```

Addestrare la GAN

A questo punto abbiamo una rete neurale, la GAN, che è ottenuta mettendo in sequenza il generatore e il discriminatore, il che vuol dire che la GAN prende in input un insieme di numeri a caso (che passa al generatore) e offre in output un numero (che produce il discriminatore).

Ma a noi non interessa l'output della GAN bensì quello del generatore, e l'addestramento della GAN, che include sia discriminatore che generatore, ci interessa solo perché implica l'addestramento del generatore stesso.

Questo addestramento coinvolge, per ogni batch di dati di traininig, tanto il discriminatore quanto il generatore all'interno della GAN.

Addestrare la GAN

A questo punto abbiamo una rete neurale, la GAN, che è ottenuta mettendo in sequenza il generatore e il discriminatore, il che vuol dire che la GAN prende in input un insieme di numeri a caso (che passa al generatore) e offre in output un numero (che produce il discriminatore).

Ma a noi non interessa l'output della GAN bensì quello del generatore, e l'addestramento della GAN, che include sia discriminatore che generatore, ci interessa solo perché implica l'addestramento del generatore stesso.

Questo addestramento coinvolge, per ogni batch di dati di traininig, tanto il discriminatore quanto il generatore all'interno della GAN.

Addestrare la GAN

A questo punto abbiamo una rete neurale, la GAN, che è ottenuta mettendo in sequenza il generatore e il discriminatore, il che vuol dire che la GAN prende in input un insieme di numeri a caso (che passa al generatore) e offre in output un numero (che produce il discriminatore).

Ma a noi non interessa l'output della GAN bensì quello del generatore, e l'addestramento della GAN, che include sia discriminatore che generatore, ci interessa solo perché implica l'addestramento del generatore stesso.

Questo addestramento coinvolge, per ogni batch di dati di traininig, tanto il discriminatore quanto il generatore all'interno della GAN.

Addestramento della GAN

```
EPOCHS = 64
print(" epoca | accuratezza discrimin. |
    accuratezza generatore ")
for e in range(EPOCHS):
  for k in range(TRAIN_SIZE//BATCH):
      n = randint(0, TRAIN_SIZE, size = BATCH
    //2)
      x = np.concatenate((X_TRAIN[n], generator.
    predict(NOISE[n])))
      y = np.concatenate(ONEZEROS)
      discriminator.trainable = True
      d_loss, d_acc = discriminator.
    train_on_batch(x, y)
      discriminator.trainable = False
      g_loss, g_acc = gan.train_on_batch(NOISE[n
    ], ONES)
  print(f"  {e:03n}  |        {d_acc:.5f}
     |        {g_acc:.5f}")
```

Facciamole generare qualcosa...

Che ci crediate o no, ora il generatore produce ordine dal caos: dandogli in input numeri a caso offrirà in output una sinusoide. Naturalmente, vista la semplicità delle reti che abbiamo usato, disegnerà questa sinusoide un po' con mano tremula ma insomma la forma c'è.

Per interrogare il generatore usiamo il codice seguente:

```
x = uniform(Y_MIN, Y_MAX, size = (1, SAMPLE_LEN)
    )
y = generator.predict(x)[0]     # output lista!
plt.plot(X_COORDS, y)
plt.show()
```

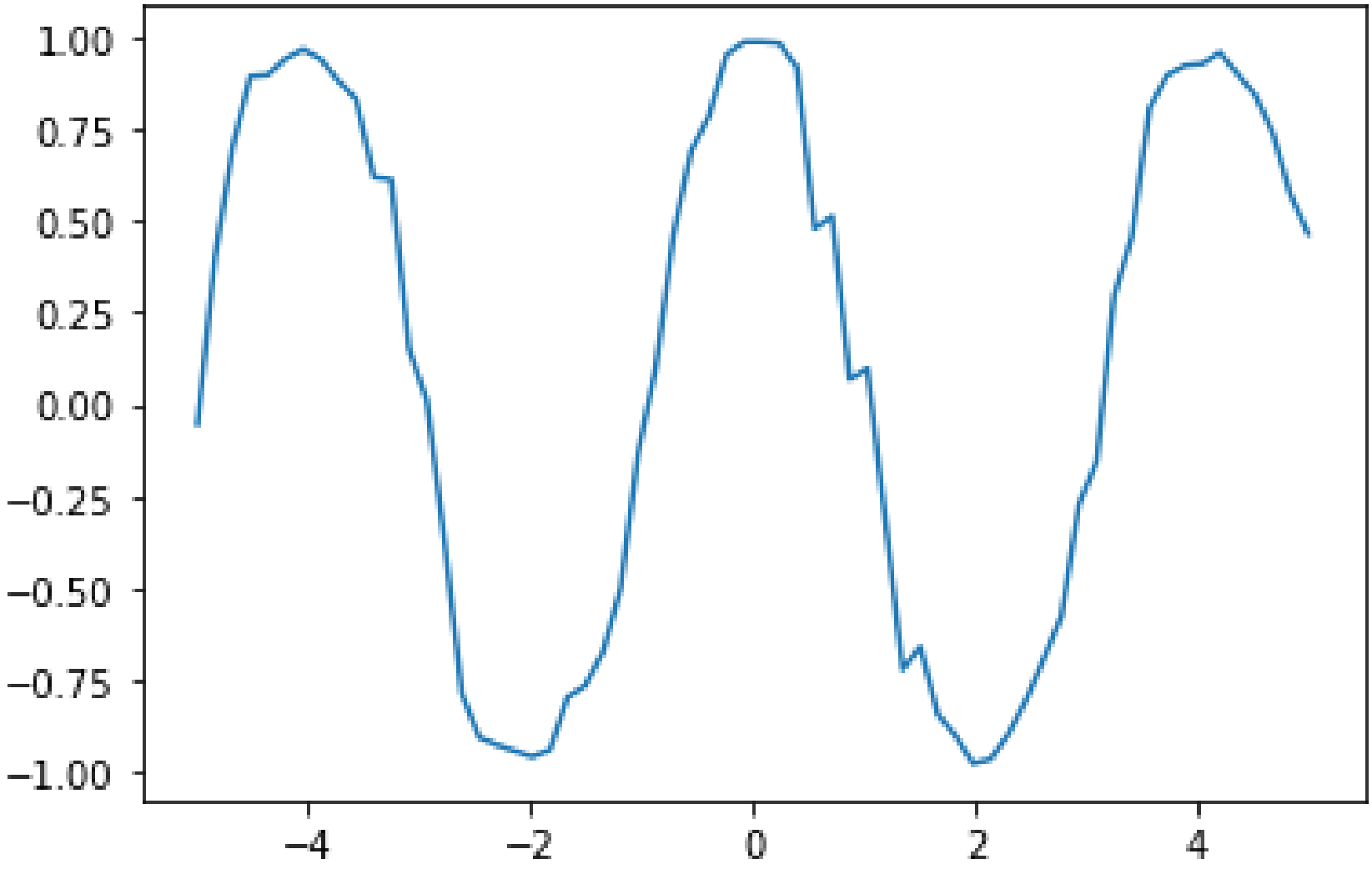
Facciamole generare qualcosa...

Che ci crediate o no, ora il generatore produce ordine dal caos: dandogli in input numeri a caso offrirà in output una sinusoide. Naturalmente, vista la semplicità delle reti che abbiamo usato, disegnerà questa sinusoide un po' con mano tremula ma insomma la forma c'è.

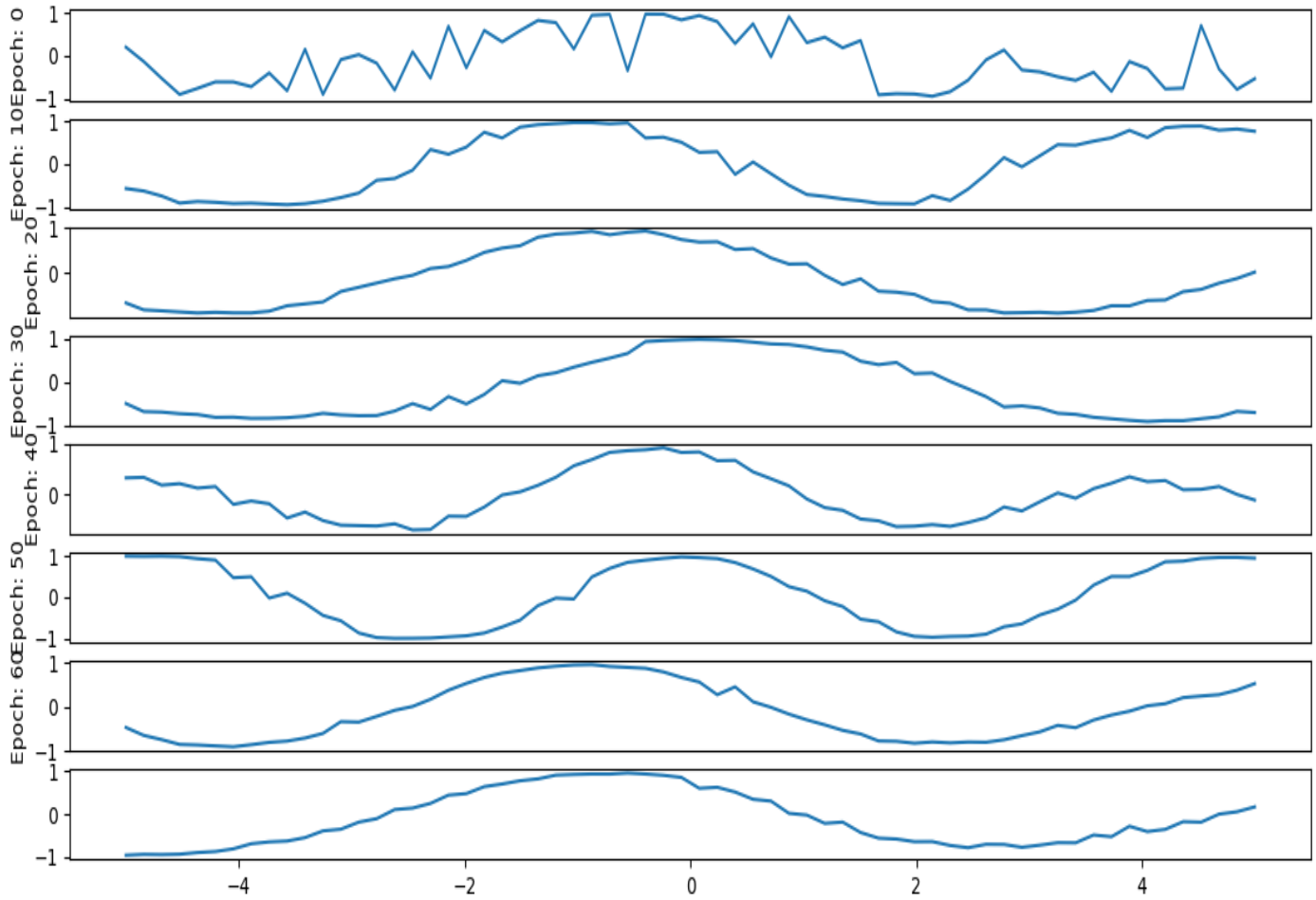
Per interrogare il generatore usiamo il codice seguente:

```
x = uniform(Y_MIN, Y_MAX, size = (1, SAMPLE_LEN)
    )
y = generator.predict(x)[0]     # output lista!
plt.plot(X_COORDS, y)
plt.show()
```

Sinusoide prodotta dalla GAN



Curve generate al migliorare dell'accuratezza



Q&A

Paolo Caressa
IT Expert @*Gestore dei Servizi Energetici*

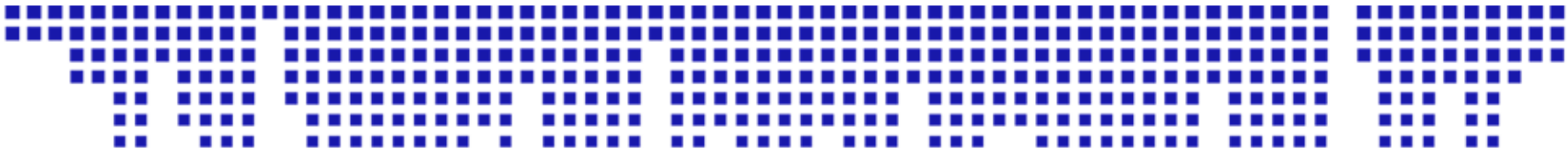
https://www.linkedin.com/in/paolocaressa/
https://github.com/pcaressa
@www_caressa_it

Il codice Python contenuto in queste slide e le slide stesse le trovate su

https://gist.github.com/pcaressa/be67ff30dfd9ca4959de9458944c1f29

Grazie a Luca Piccinelli per aver segnalato delle imprecisioni nella prima versione di queste slide e nel codice su GitHub.

DEVCAST



Thank you!