# Approfondimento di Intelligenza Artificiale

Tristano Munini

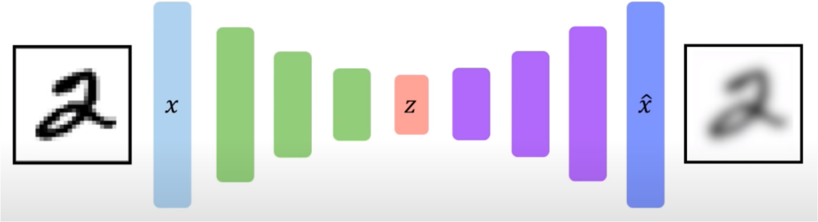
11 ottobre 2020

# Indice

- AutoEncoders (AE) e Variational AutoEncoders (VAE)
- Generative Adversarial Network (GAN)
- Reinforcement Learning (RL)
  - QNN
  - Policy Gradient
- GAN + RL per generazione di testi
  - SeqGAN
  - LeakGAN

# AutoEncoders

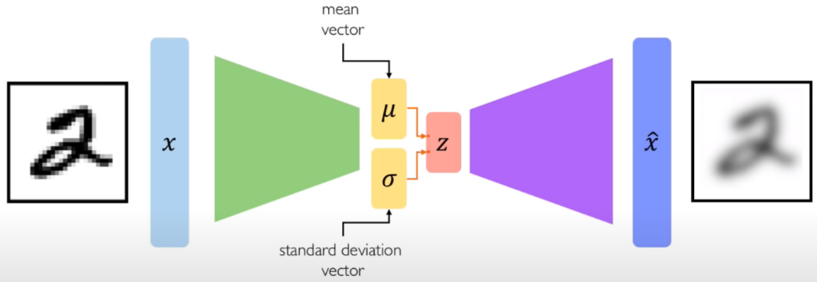
$x \in$ real data space
$z \in$ latent space
$\hat{x} \in$ real data space




The point of maximum compression, where $z$ is produced, is called bottleneck of the network

# Variational AutoEncoders

$z$ is sampled from the distribution with mean vector $\mu$ and standard deviation vector $\sigma$.
We want an $\hat{x}$ that is similar to $x$ but that has some differences.
We want a variation of $x$.



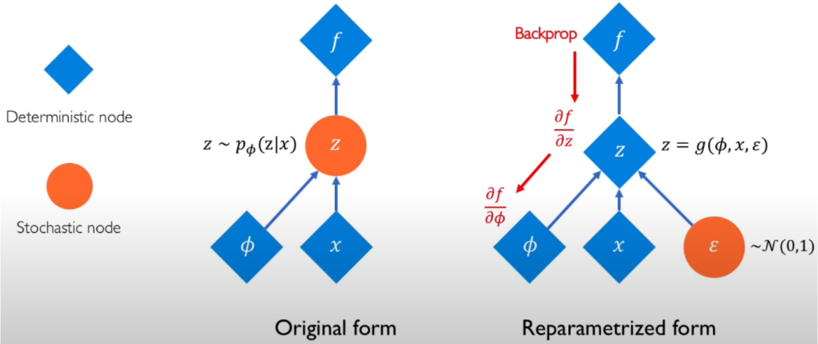
# Reparametrization Trick

We can not use $z \sim \mathcal{N}(\mu, \sigma^2)$
So we use

$$z = \mu + \sigma \odot \varepsilon \quad \text{where } \varepsilon \sim \mathcal{N}(0, 1)$$

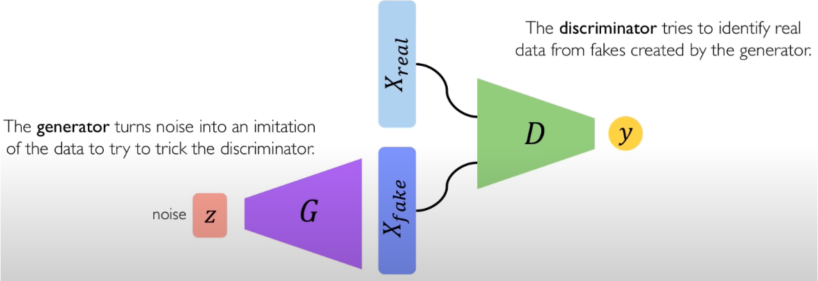
Where $\odot$ notes the element-wise multiplication
Now $z$ is the sum of two fixed vectors, $\mu$ and $\sigma$, and a random constant $\varepsilon$ used as a weight

# Reparametrization Trick



# Generative Adversarial Networks

GANs are a way to make a generative model by having two neural networks compete with each other




$z$ can be sampled form $\mathcal{N}(0, 1)$ as in the VAEs

# GAN Training

It is a min-max game

$$min_G \ max_D \ V(D, G) = \mathbb{E}_{x \sim P_{real(x)}}[logD(x)] + \mathbb{E}_{z \sim P_z(x)}[log(1 - D(G(z))]$$

Where $P_{real(x)}$ is the probability distribution of the real data and $P_z(x) = \mathcal{N}(0, 1)$ in our case

# GAN Problems

- Hard to converge to a good solution
- Vanishing gradient problem
- Model collapse
- Hard to find equilibrium

# template


Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

Discounted Total Reward (Return)

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \ldots + \gamma^{t+n} r_{t+n} + \cdots$$

$\gamma$: discount factor; $0 < \gamma < 1$

# template

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

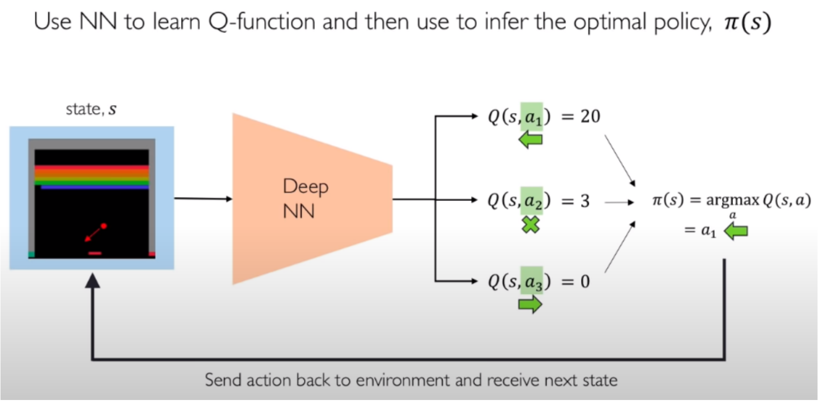
Total reward, $R_t$, is the discounted sum of all rewards obtained from time $t$

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

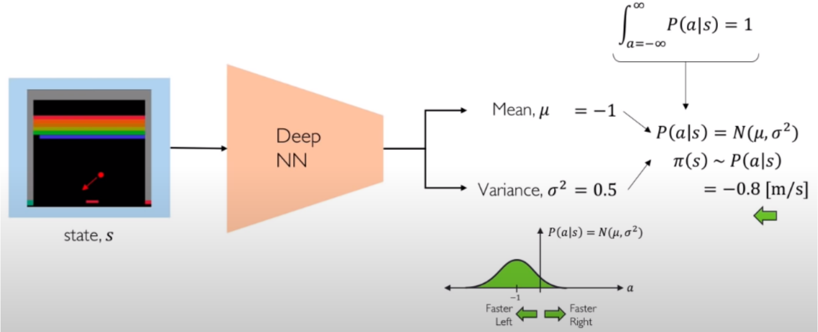
The Q-function captures the **expected total future reward** an agent in state, $s$, can receive by executing a certain action, $a$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

# template

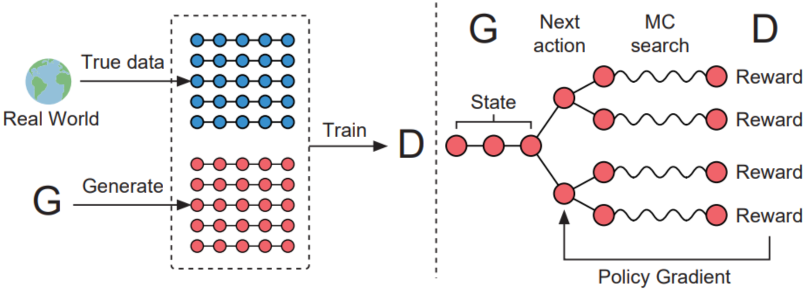

Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$
state, $s$
Deep NN
$Q(s, a_1) = 20$
$Q(s, a_2) = 3$
$Q(s, a_3) = 0$
$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$
$= a_1$
Send action back to environment and receive next state

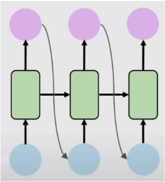
# template



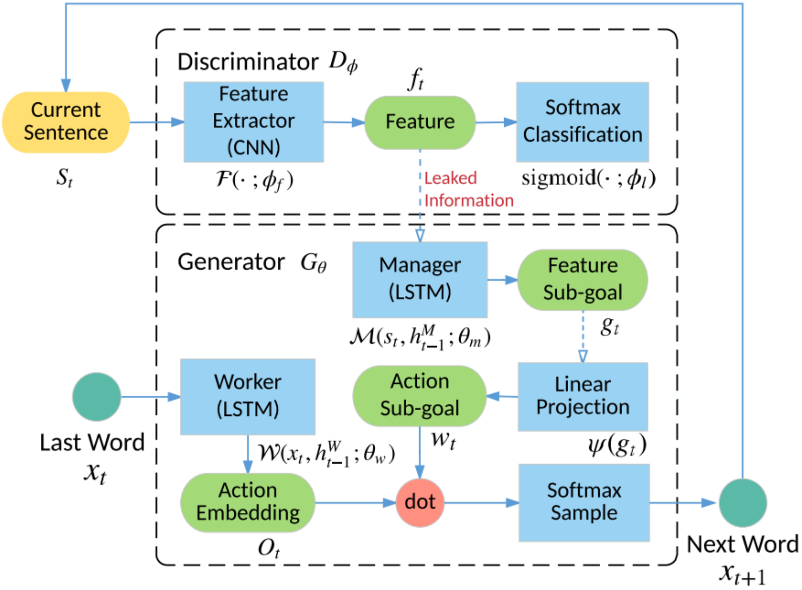
$\int_{a=-\infty}^{\infty} P(a|s) = 1$
Deep
NN
state, $s$
Mean, $\mu$ $= -1$
Variance, $\sigma^2$ $= 0.5$
$P(a|s) = N(\mu, \sigma^2)$
$\pi(s) \sim P(a|s)$
$= -0.8$ [m/s]
$P(a|s) = N(\mu, \sigma^2)$
$a$
$-1$
Faster
Left
Faster
Right

# template

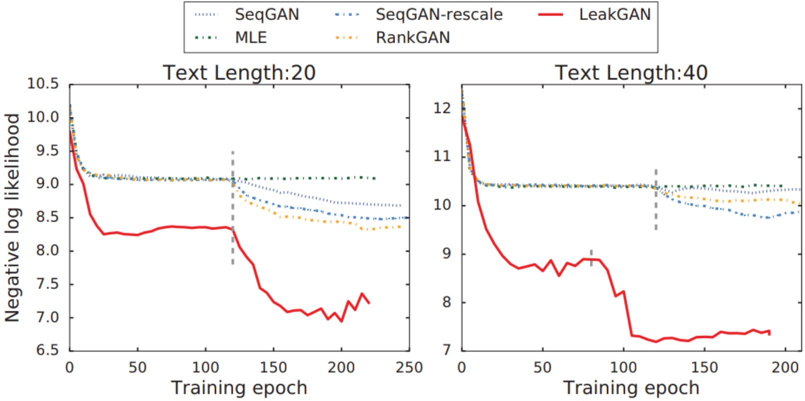

True data
Real World
Generate
G
Train
D
G
Next action
MC search
D
State
Reward
Reward
Reward
Reward
Policy Gradient



# template


Discriminator $D_\phi$
Current Sentence
$S_t$
Feature Extractor (CNN)
$\mathcal{F}(\cdot\,;\phi_f)$
$f_t$
Feature
Softmax Classification
sigmoid$(\cdot\,;\phi_l)$
Leaked Information
Generator $G_\theta$
Manager (LSTM)
$\mathcal{M}(s_t, h^M_{t-1};\theta_m)$
Feature Sub-goal
$g_t$
Linear Projection
$\psi(g_t)$
Action Sub-goal
$w_t$
Worker (LSTM)
$\mathcal{W}(x_t, h^W_{t-1};\theta_w)$
Last Word
$x_t$
Action Embedding
$O_t$
dot
Softmax Sample
Next Word
$x_{t+1}$

# template


SeqGAN
SeqGAN-rescale
LeakGAN
MLE
RankGAN
Text Length:20
Text Length:40
Negative log likelihood
10.5
10.0
9.5
9.0
8.5
8.0
7.5
7.0
6.5
0
50
100
150
200
250
Training epoch
12
11
10
9
8
7
0
50
100
150
200
Training epoch