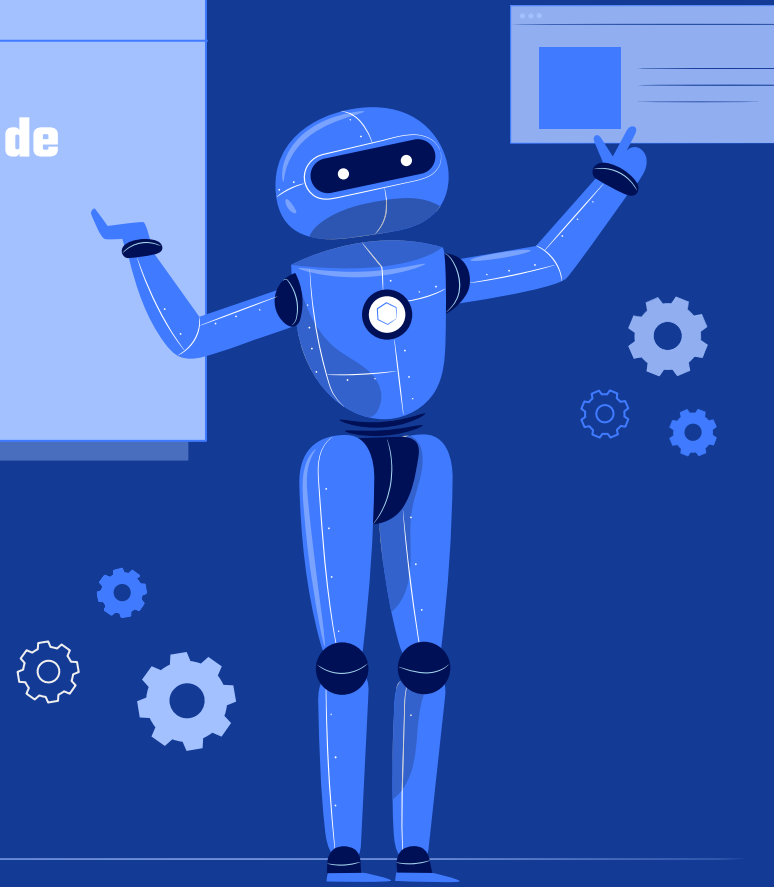
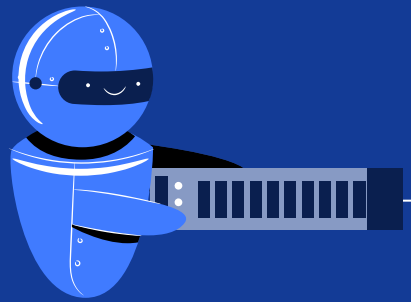
Deep reinforcement learning for de novo drug design

A ReLeaSe method execution on a Docker Environment

Relatori:
Ivan Buccella
Matteo Maiorano



01 Introduction



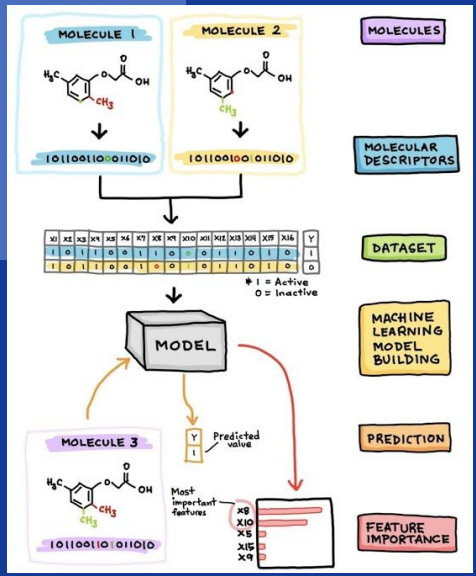
L'Intelligenza Artificiale

Rivoluzione della medicina

Deep Reinforcement Learning (DRL)

Nella scoperta di farmaci, l'obiettivo è di formulare un'ipotesi su come sintetizzare nuovi composti o selezionarli dalle librerie chimiche disponibili.

Librerie basate su SAR data



RETURN TO ISSUE | < PREV **RESEARCH ARTICLE** NEXT >

Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks

Marwin H. S. Segler*†, Thierry Kogej‡, Christian Tyrchan§, and Mark P. Waller*∥

View Author Information

Cite this: *ACS Cent. Sci.* 2018, 4, 1, 120−131
Publication Date: December 28, 2017
https://doi.org/10.1021/acscentsci.7b00512

Article Views 33214 | Altmetric 39 | Citations 600
LEARN ABOUT THESE METRICS

Share | Add to | Export (RIS)

PDF (2 MB) | Supporting Info (2) »

SUBJECTS: Drug discovery, Molecular modeling, Molecular structure, Molecules, Neural networks

ACS Central Science

- Le RNN possono essere addestrate come modelli generativi per le strutture molecolari
- Le proprietà delle molecole generate si correlano molto bene
- Genera nuove molecole per il de novo drug design

Research article | Open Access | Published: 04 September 2017

Molecular de-novo design through deep reinforcement learning

Marcus Olivecrona ✉, Thomas Blaschke, Ola Engkvist & Hongming Chen

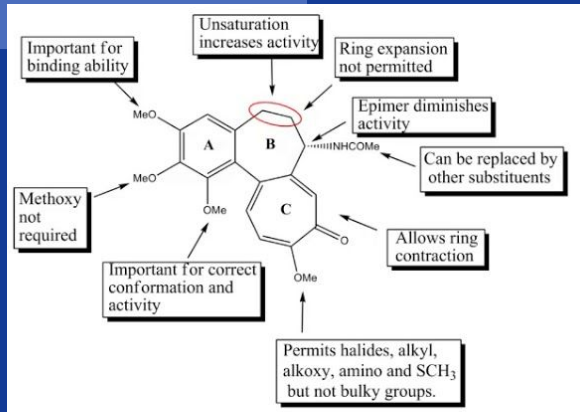
Journal of Cheminformatics **9**, Article number: 48 (2017) | Cite this article

- Generazione di composti che si prevede siano attivi contro un bersaglio biologico
- Contro il recettore della **dopamina di tipo 2** genera strutture di cui si prevede che oltre il 95% sia attivo
  - inclusi attivi confermati sperimentalmente
  - non inclusi né nel modello generativo né nel modello predittivo

Le structure-activity relationship (SAR)
Tecnica utilizzata in chimica farmaceutica e biologica per prevedere l'attività di un composto chimico basandosi sulla sua struttura molecolare.

Utilizzo

- sintetizzare nuovi composti con attività biologica desiderata
- capire come una modifica strutturale può influire sull'attività di un composto esistente



Enorme quantitativo di sostanze chimiche sinteticamente fattibili.
Impossibile:

- l'esaminazione sistematica
- la verifica attraverso la costruzione della stessa
- valutazione di ogni singolo composto

Molto utili le Deep Neural Networks (DNN).

Apprendimento dati tramite algoritmi
Elaborare grandi quantità di dati

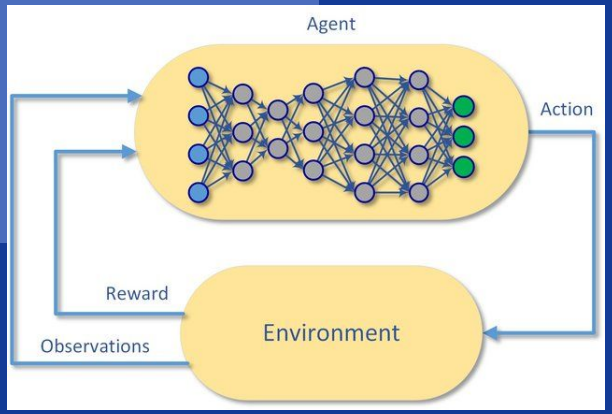
Deep Learning

Agente interagisce con il suo ambiente
Massimizzare ricompensa
Trial and Error

Reinforcement Learning (RL)

Deep Reinforcement Learning (DRL)

Deep Neural Networks to learn RL Policy



Deep Neural Network (DNN)

- Composta da neuroni
- Imita il modo in cui il cervello elabora i dati
- Apprendere automaticamente i modelli presenti nei dati di input

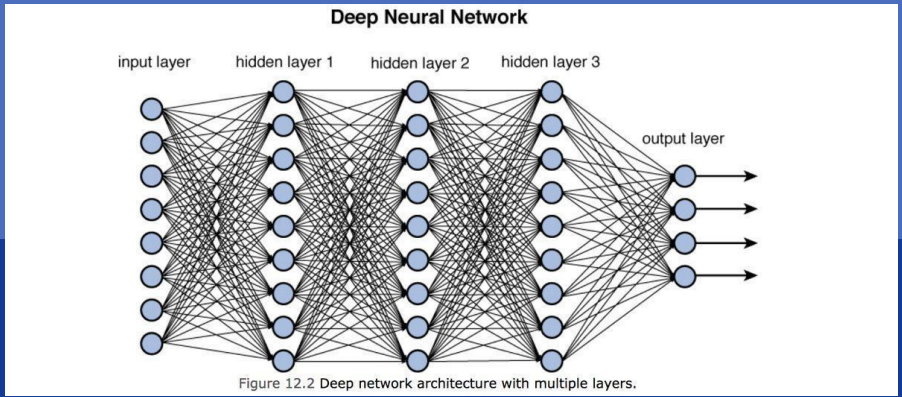
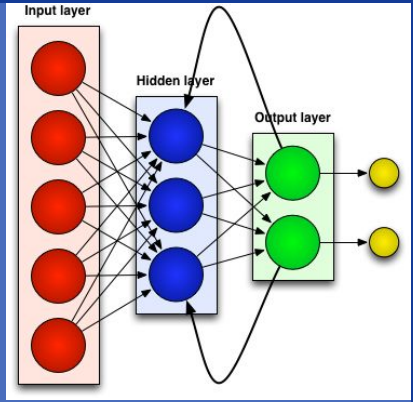


Figure 12.2 Deep network architecture with multiple layers.

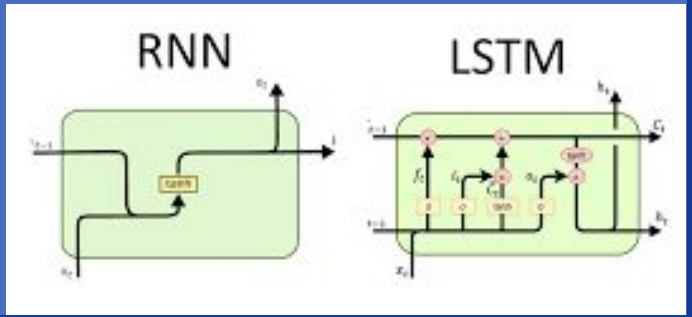
Recurrent Neural Network (RNN)



- Composta da neuroni
- Struttura a loop
- Elaborazione sequenziale, ma
- Informazioni precedenti influiscono sull'elaborazione attuale
- Utilizza uno strato Long Short-Term Memory

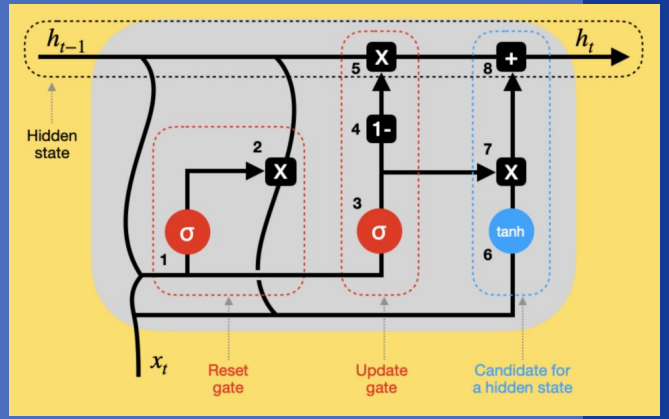
Long Short-Term Memory (LSTM)

- Memorizza le informazioni a lungo termine.
- Controlla quali informazioni vengono mantenute e dimenticate,
- Elaborazione di sequenze di input di lunga durata.



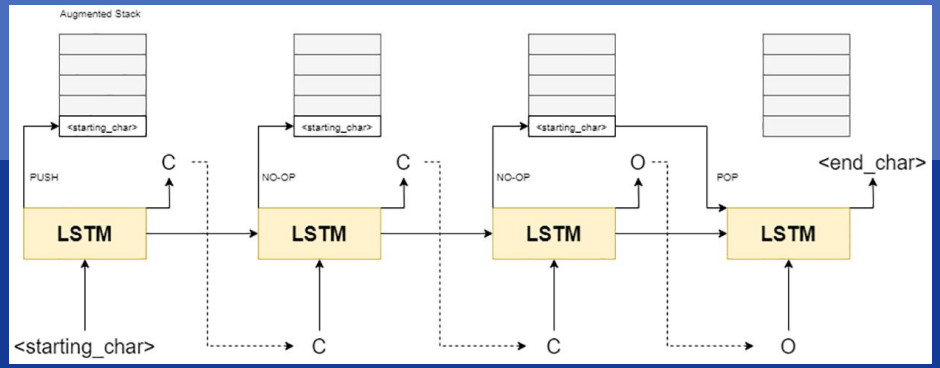
Gated Recurring Unit

- Un meccanismo di gating nelle RNN
- Porte per controllare il flusso di informazioni all'interno e all'esterno dell ' hidden state
- Impara quali parti della sequenza sono più efficaci

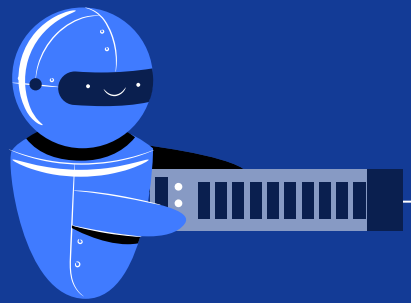


Stack-Augmented Recurrent Neural Network (Stack-RNN)

- Struttura di dati a stack per memorizzare e elaborare sequenze di dati
- Mantenere una gerarchia di informazioni
- Ideale per dati con una struttura ad albero o gerarchica
- Stato dello stack viene aggiornato ad ogni passo temporale

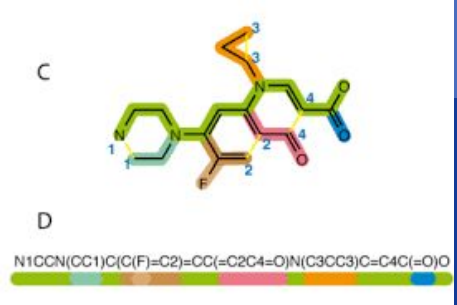


02 Background



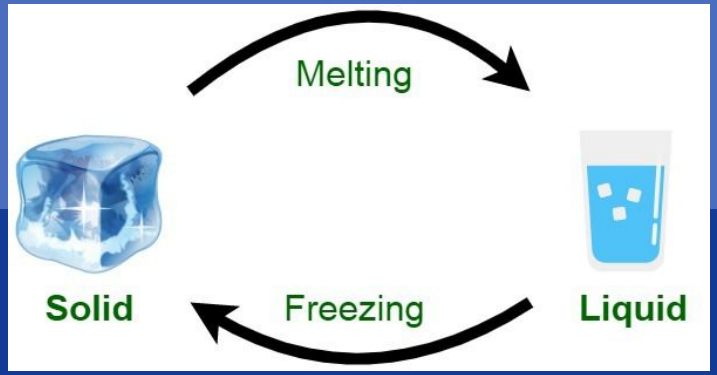
Simplified Molecular Input Line Entry System (SMILES)

- Descrivere la struttura di una molecola con una stringa ASCII
- Atomi: simbolo chimico chiuso tra parentesi quadre *[Au]* ORO
- Legame: uno tra i simboli . - = # $: / \
- Anelli: etichetta numerica, rotti in punto arbitrario
- Rami: descritti con parentesi *CCC(=O)O* ACIDO PROPIONICO
- Stringhe contenute in file con estensione *.smi*



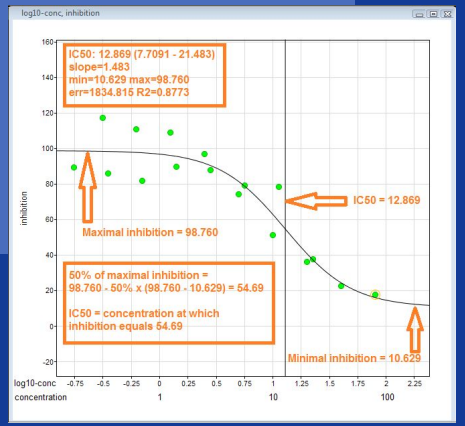
Melting Point (Tm)

- temperatura alla quale un solido passa allo stato liquido
- può influire sulla stabilità e la conservazione di un composto
- può influire sulla sua sicurezza e biodisponibilità



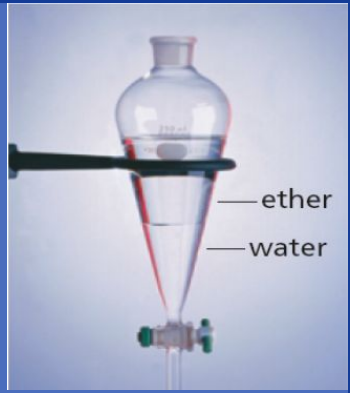
Inhibitory concentration (IC) & Half maximal IC (IC50)

- IC misura dell'efficacia di un composto chimico nell'inibire l'attività di un particolare bersaglio biologico
- IC50 = quantità di sostanza inibitoria (ad es. farmaco) necessaria per inibire del 50% un bersaglio biologico
- pIC50 = logaritmo negativo di IC50
- pIC50 usato per valutare l'efficacia di un farmaco
- Più alta è la pIC50 di un composto, maggiore è la sua attività biologica



Coefficiente di ripartizione (logP)

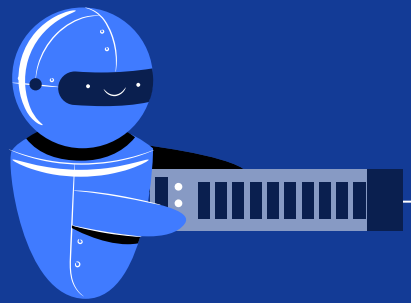
- Misura quanto di un soluto si dissolve nella porzione di acqua rispetto a una porzione organica
- Aiuta a determinare se un composto sarà più solubile in grasso o in acqua
- Proprietà di un composto chimico per prevedere la sua tossicità o l'attività biologica



Regola di Lipinski

- Insieme di criteri che vengono utilizzati per valutare la "farmacocinetica orale" di una sostanza chimica
  - la capacità di un composto chimico di essere assorbito e distribuito nel corpo attraverso l'ingestione orale
- Sviluppata per identificare composti chimici idonei per il trattamento farmacologico degli esseri umani attraverso l'ingestione orale

03 ReLeaSe



RESEARCH ARTICLE | COMPUTATIONAL BIOLOGY

Deep reinforcement learning for de novo drug design

MARIYA POPOVA, OLEXANDR ISAYEV, AND ALEXANDER TROPSHA Authors Info & Affiliations

SCIENCE ADVANCES · 25 Jul 2018 · Vol 4, Issue 7 · DOI: 10.1126/sciadv.aap7885

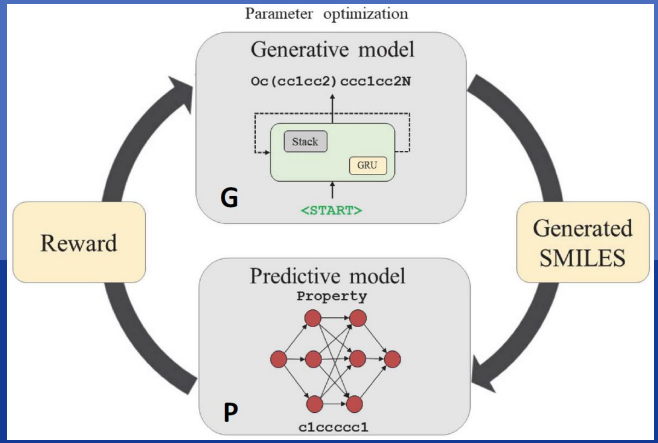
15,649

Reinforcement Learning for Structural Evolution

- Metodo per generare composti chimici con proprietà fisiche, chimiche e/o bioattive desiderate basate sul DRL
- Prevedere diverse proprietà descritte in precedenza come **logP, Tm e pIC50**
- **Può imparare a selezionare le molecole più promettenti per lo sviluppo di nuovi farmaci!!!**

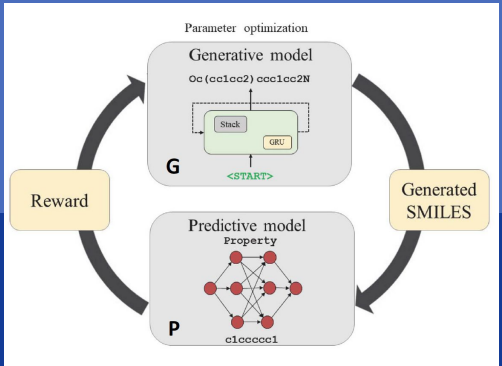
Training Process

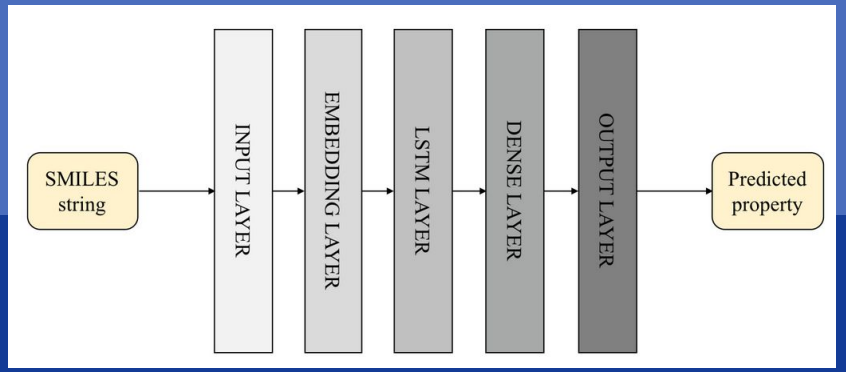
- Viene costruito il generative model, poi
- entrambi i modelli, generative e predictive, sono combinati in un unico sistema di Reinforcement Learning



Predictive Model

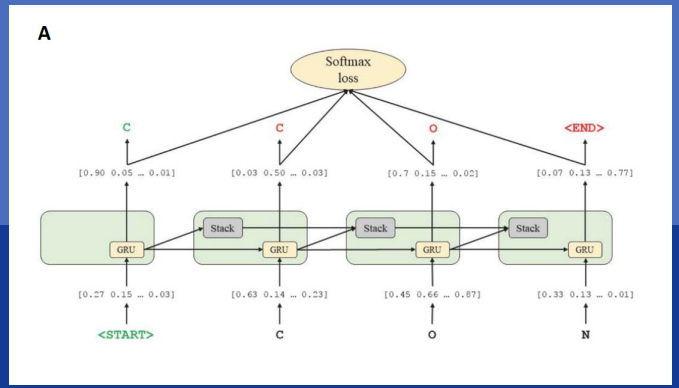
- Stima delle proprietà fisiche, chimiche o biologiche delle molecole
- Riceve in input una SMILES string
- Restituisce in output la predicted property (logP)

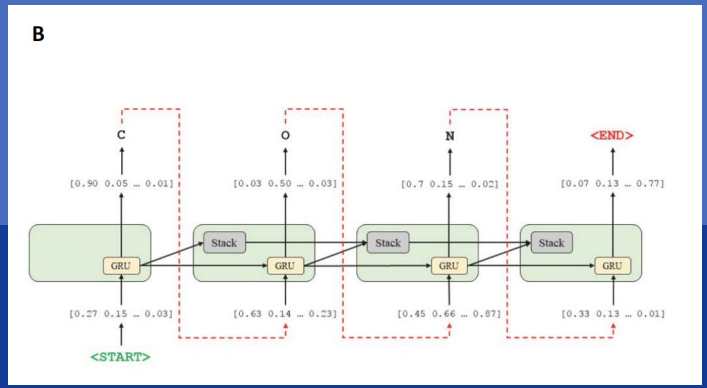




Generative Model

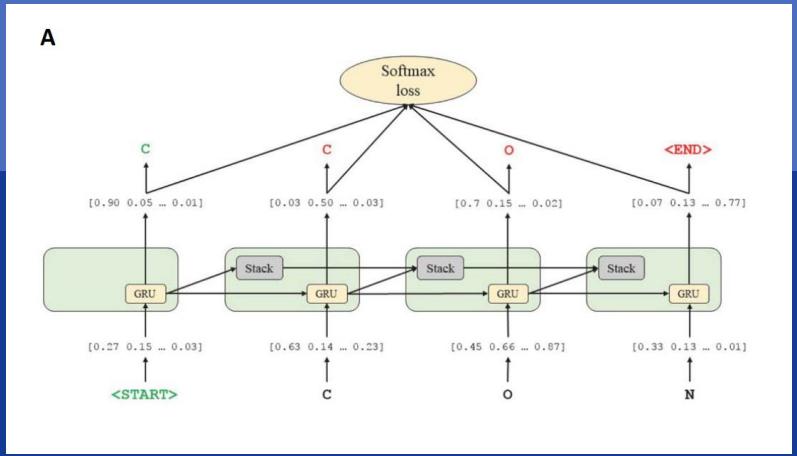
- Utilizzo di Stack-RNN
- Apprendere regole nascoste per formare sequenze di lettere che corrispondono a stringhe SMILES legittime.
- Due moodalità: training (A) e generating (B)





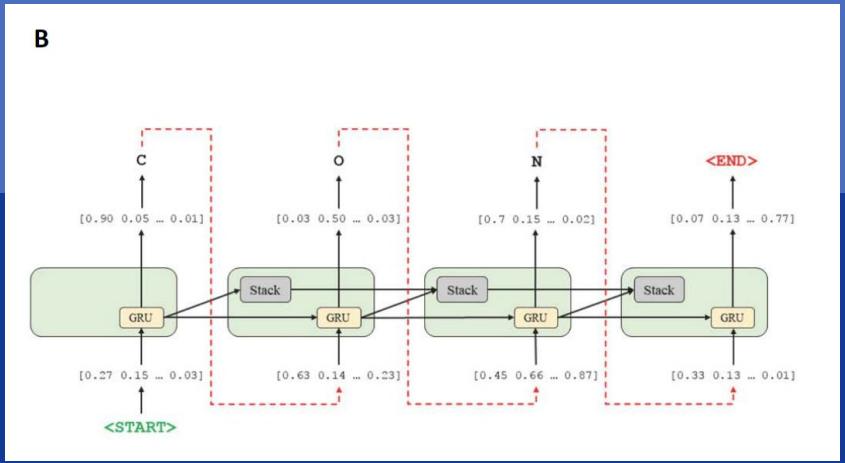
Fase 1 - Generative Model - Training

- Prende un prefisso corrente del training object e prevede la distribuzione di probabilità del carattere successivo
- Il carattere successivo viene campionato da questa distribuzione di probabilità prevista
- Funzione poi confronta la previsione con la correttezza della stringa SMILES



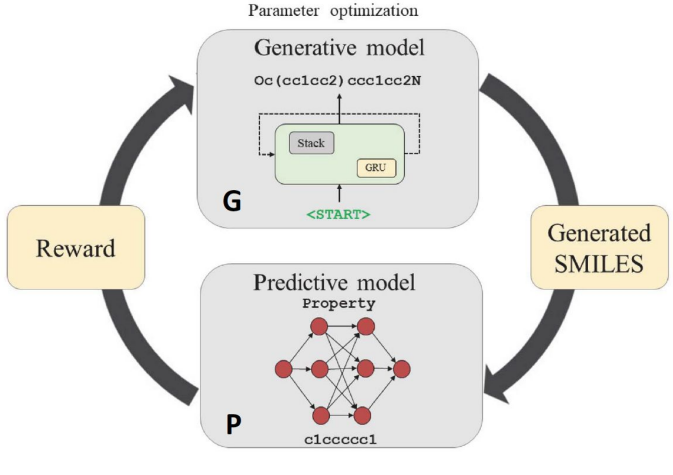
Fase 1 - Generative Model - Generating

- Prende un prefisso di sequenze già generate
- Prevede la distribuzione di probabilità del carattere successivo
- La campiona da questa distribuzione prevista



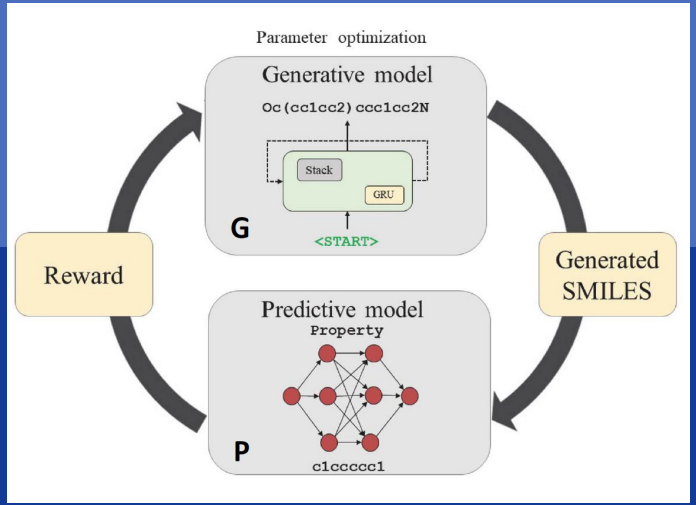
Fase 2 - Reinforcement Learning

- Reward: funzione della proprietà numerica calcolata dal predictive model
- Generative Model: viene addestrato per massimizzare la ricompensa attesa



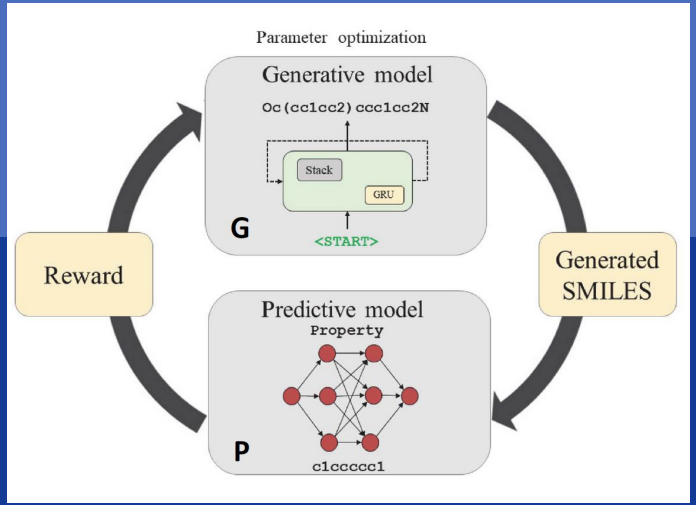
Fase 2 - Generative Model

- Ruolo di agente
- spazio delle azioni è rappresentato dall'alfabeto della notazione SMILES
- spazio degli stati è rappresentato da tutte le possibili stringhe di questo alfabeto

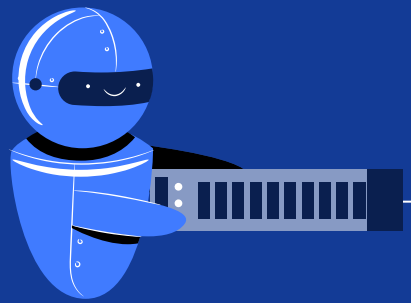


Fase 2 - Predictive Model

- Stima il comportamento dell'agente
- Assegna una ricompensa (reward) numerica ad ogni molecola generata



04 The Execution



ReLeaSe

Produzione di molecole con valori di logP all'interno di drug-like regions secondo la regola di Lipinsky.

Policy Gradient Algorithm con una funzione di reward personalizzata

Il generator produce una nuova stringa SMILES valutata dal predictor.

Sulla base della previsione ottenuta e dell'obiettivo, viene assegnato un valore di ricompensa numerico e vengono aggiornati i parametri del generator utilizzando un policy gradient algorithm.

Il policy gradient loss è definito come segue:

$$L(S|\theta) = -\frac{1}{n}\sum_{i=1}^{|S|}\sum_{j=1}^{length(s_i)} R_i \cdot \gamma^i \cdot \log p(s_i|s_0 \dots s_{i-1}\theta),$$

Prerequisites

Hardware

- Linux OS oppure WSL2 su Windows 10 (o superiore) macchina.
- NVIDIA GPU, compatibile con CUDA 11.3.

Prerequisites

Software

- Docker and Docker Compose (Application containers engine) https://www.docker.com
- Il supporto GPU è abilitato sulla macchina https://docs.docker.com/compose/gpu-support/

Prerequisites
Nvidia Container Toolkit

https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#install-guide

N.B necessario installare il CUDA Toolkit sul sistema host.
N.B necessario installare i driver NVIDIA sul sistema

The Dockerfile

Image Nvidia/cuda

Installazione di Miniconda

Creazione della release environment

Installazione dei pacchetti conda e pip richiesti

Clone Repository ReLeaSe

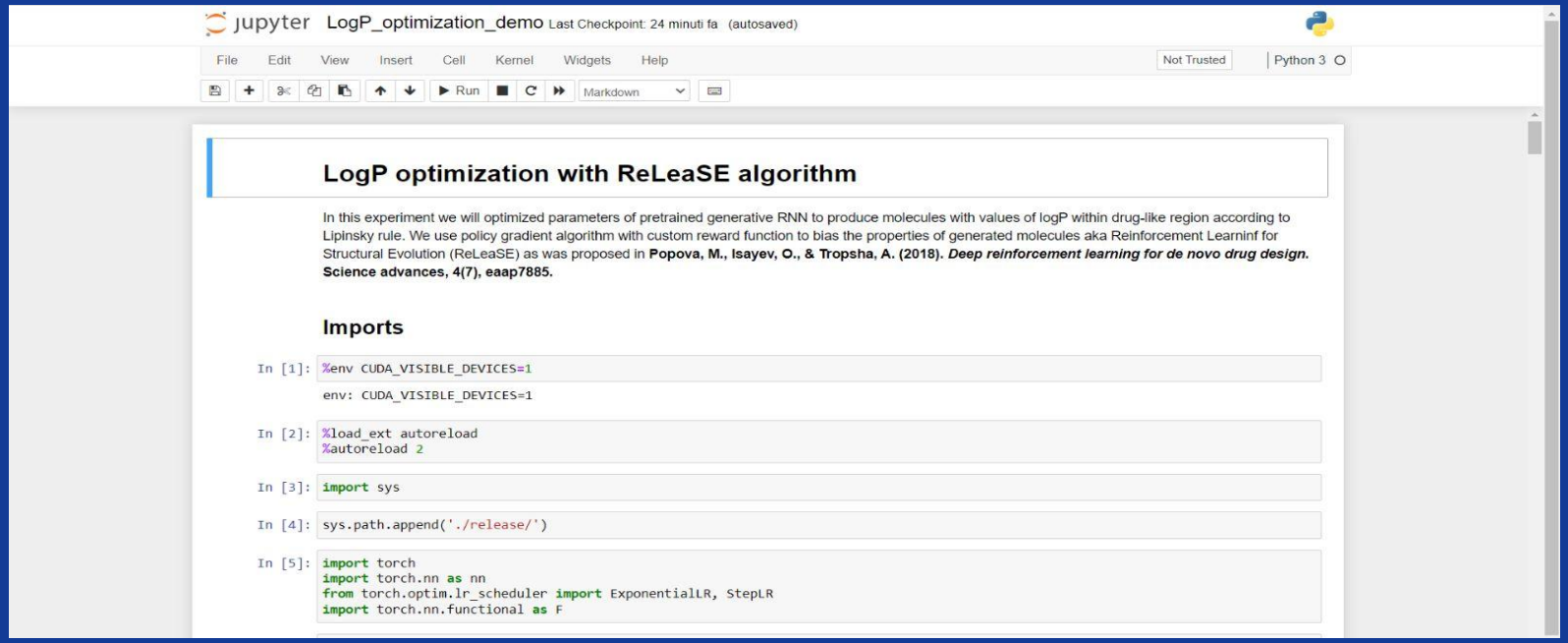
```
FROM nvidia/cuda:11.3.0-runtime-ubuntu20.04

RUN apt-get update
RUN apt-get install -y git
RUN apt-get install -y wget

RUN wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
RUN bash Miniconda3-latest-Linux-x86_64.sh -b -p /miniconda
ENV PATH=$PATH:/miniconda/pcondabin:/miniconda/bin

RUN conda update -n base -c defaults conda
RUN conda create -n release python=3.6

SHELL ["conda","run","-n","release","/bin/bash", "-c"]

COPY environment environment
RUN conda install --yes --file environment/conda.txt
RUN conda install -c rdkit rdkit nox cairo
RUN conda install pytorch=1.10.2 torchvision=0.11.3 -c pytorch -c conda-forge
RUN pip install -r environment/pip.txt

WORKDIR /home

RUN git clone https://github.com/isayev/ReLeaSE.git .

ENTRYPOINT ["conda", "run", "-n", "release", "jupyter", "notebook", "--ip=0.0.0.0", "--port=8888",
"--allow-root", "--NotebookApp.token=''", "--NotebookApp.password=''"]
```

The Docker Compose

Servizio con linux con risorse riservate Nvidia GPU

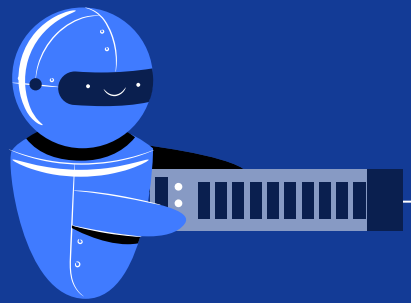
```yaml
version: "3.8"

services:
  app:
    platform: linux/amd64
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - ${HTTP_PORT}:8888
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              capabilities: [gpu]
```

The Jupyter Notebook



04.1

First stage
Setting up the generator



Caricamento dei dati per il generator

```
gen_data_path = './data/chembl_22_clean_1576904_sorted_std_final.smi'
```

```
tokens = ['<', '>', '#', '%', ')', '(', '+', '-', '/', '.', '1', '0', '3', '2', '5', '4', '7',
          '6', '9', '8', '=', 'A', '@', 'C', 'B', 'F', 'I', 'H', 'O', 'N', 'P', 'S', '[', ']',
          '\\', 'c', 'e', 'i', 'l', 'o', 'n', 'p', 's', 'r', '\n']
```

```
gen_data = GeneratorData(training_data_path=gen_data_path, delimiter='\t',
                         cols_to_read=[0], keep_header=True, tokens=tokens)
```

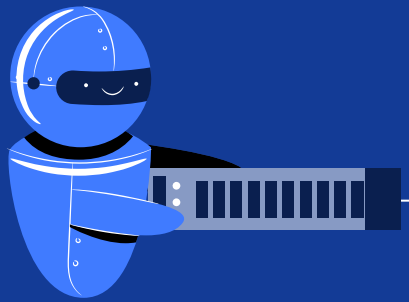
function estimate_and_update:

```
def estimate_and_update(generator, predictor, n_to_generate):
    generated = []
    pbar = tqdm(range(n_to_generate))
    for i in pbar:
        pbar.set_description("Generating molecules...")
        generated.append(generator.evaluate(gen_data, predict_len=120)[1:-1])

    sanitized = canonical_smiles(generated, sanitize=False, throw_warning=False)[:-1]
    unique_smiles = list(np.unique(sanitized))[1:]
    smiles, prediction, nan_smiles = predictor.predict(unique_smiles, use_tqdm=True)

    plot_hist(prediction, n_to_generate)

    return smiles, prediction
```

Inizializzazione stack-augmented generative RNN:

```
hidden_size = 1500
stack_width = 1500
stack_depth = 200
layer_type = 'GRU'
lr = 0.001
optimizer_instance = torch.optim.Adadelta

my_generator = StackAugmentedRNN(input_size=gen_data.n_characters, hidden_size=hidden_size,
                                 output_size=gen_data.n_characters, layer_type=layer_type,
                                 n_layers=1, is_bidirectional=False, has_stack=True,
                                 stack_width=stack_width, stack_depth=stack_depth,
                                 use_cuda=use_cuda,
                                 optimizer_instance=optimizer_instance, lr=lr)
```

04.2 First stage Setting up the predictor



Inizializzazione RNN predictor

```
from rnn_predictor import RNNPredictor
```

```
predictor_tokens = tokens + [' ']
```

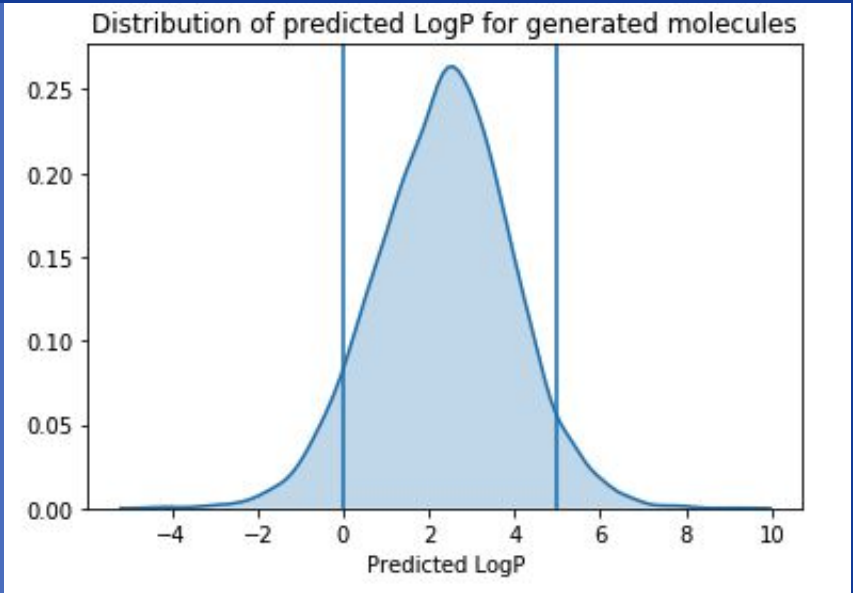
```
path_to_params = './checkpoints/logP/model_parameters.pkl'
path_to_checkpoint = './checkpoints/logP/fold_'
```

```
my_predictor = RNNPredictor(path_to_params, path_to_checkpoint, predictor_tokens)
```

Produzione del unbiased distribution della proprietà

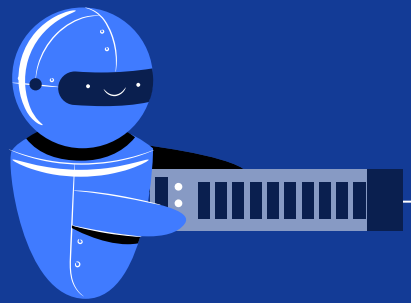
```
smiles_unbiased, prediction_unbiased = estimate_and_update(my_generator,
                                                           my_predictor,
                                                           n_to_generate=10000)
```

Output di unbiased distribution

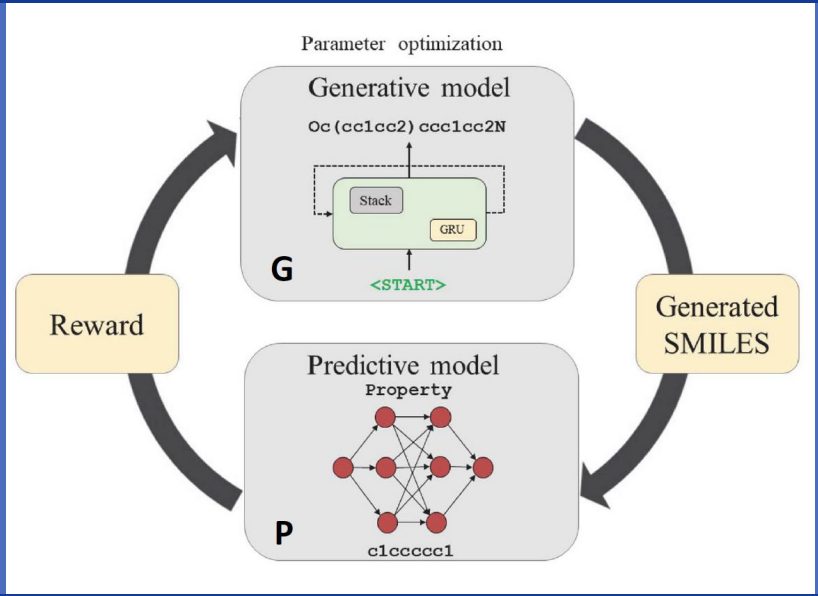


04.3

Second stage Biasing the distribution with RL (policy gradient)



Modelli generativi e predittivi in un unico sistema Reinforcement Learning con funzione di Reward



Creazione di una copia del generator che sarà ottimizzato

```
my_generator_max = StackAugmentedRNN(input_size=gen_data.n_characters,
                                     hidden_size=hidden_size,
                                     output_size=gen_data.n_characters,
                                     layer_type=layer_type,
                                     n_layers=1, is_bidirectional=False, has_stack=True,
                                     stack_width=stack_width, stack_depth=stack_depth,
                                     use_cuda=use_cuda,
                                     optimizer_instance=optimizer_instance, lr=lr)

my_generator_max.load_model(model_path)
```

La funzione di reward è la seguente:

```
def get_reward_logp(smiles, predictor, invalid_reward=0.0):
    mol, prop, nan_smiles = predictor.predict([smiles])
    if len(nan_smiles) == 1:
        return invalid_reward
    if (prop[0] >= 1.0) and (prop[0] <= 4.0):
        return 11.0
    else:
        return 1.0
```

Fase del Reinforcement Learning

```
RL_logp = Reinforcement(my_generator_max, my_predictor, get_reward_logp)
```

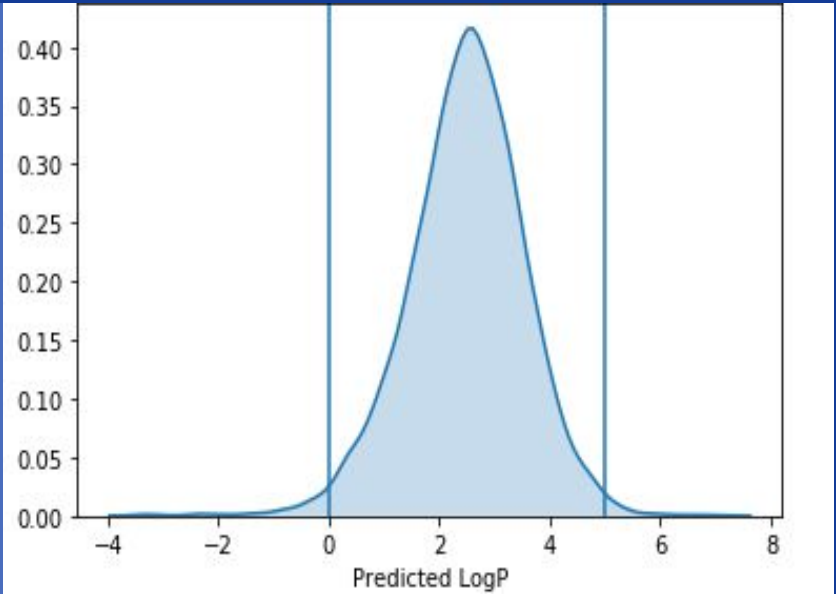
```
rewards = []
rl_losses = []
```

```
for i in range(n_iterations):
    for j in trange(n_policy, desc='Policy gradient...'):
        cur_reward, cur_loss = RL_logp.policy_gradient(gen_data)
        rewards.append(simple_moving_average(rewards, cur_reward))
        rl_losses.append(simple_moving_average(rl_losses, cur_loss))

    plt.plot(rewards)
    plt.xlabel('Training iteration')
    plt.ylabel('Average reward')
    plt.show()
    plt.plot(rl_losses)
    plt.xlabel('Training iteration')
    plt.ylabel('Loss')
    plt.show()

    smiles_cur, prediction_cur = estimate_and_update(RL_logp.generator,
                                                     my_predictor,
                                                     n_to_generate)
    print('Sample trajectories:')
    for sm in smiles_cur[:5]:
        print(sm)
```

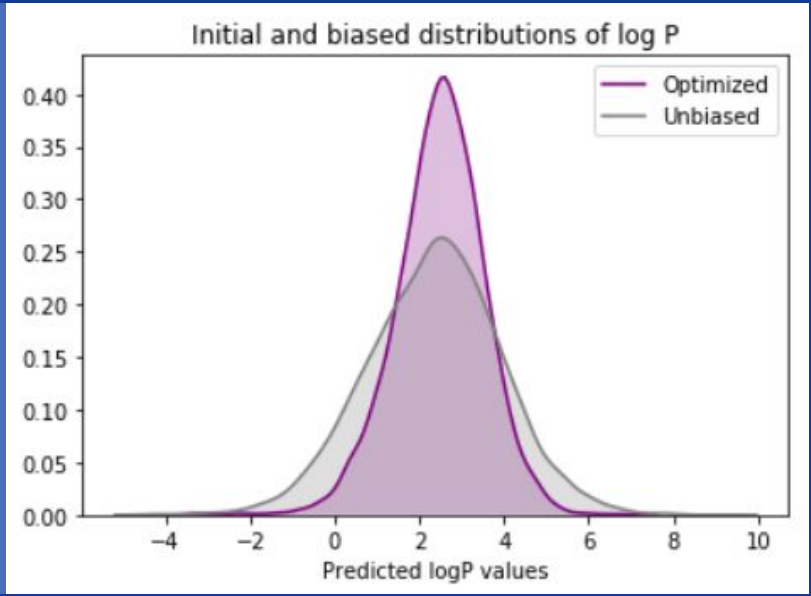
Produzione del biased distribution della proprietà

```
smiles_biased, prediction_biased = estimate_and_update(RL_logp.generator,
                                                       my_predictor,
                                                       n_to_generate=10000)
```

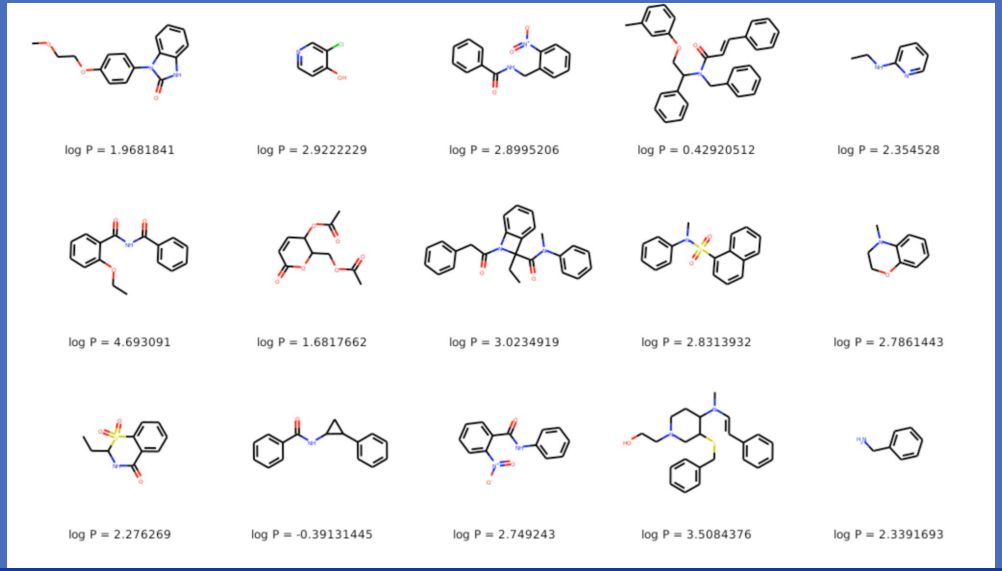
The output of the biased distribution



Unbiased and biased distribution



L ' output della biased library sarà qualcosa simile a questo



THAT'S ALL

GitHub Repository:

A special thanks to:

- Prof. Rocco Zaccagnino