Tuning Hyperparameters with

Bayesian Optimization

Author :
黃濬程
Chun-Cheng Huang

Advisor :
謝孫源
Sun-Yuan Hsieh

NCKU

Department of Computer Science and Information Engineering， National Cheng Kung University，Taiwan

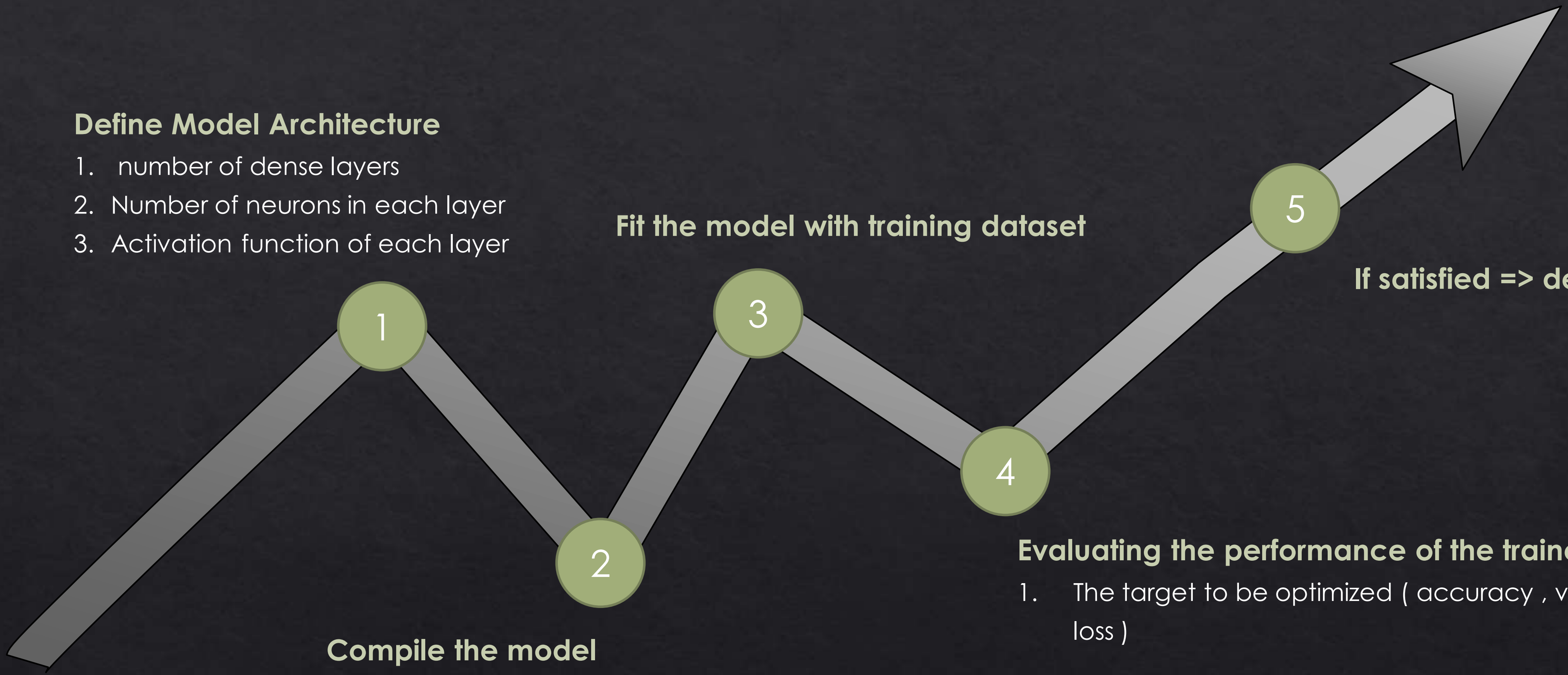
Contents

Actual Situations

part 1

Tuning Hyperparameters with Bayesian Optimization

Actual Situations



Define Model Architecture

1. number of dense layers
2. Number of neurons in each layer
3. Activation function of each layer

Compile the model

1. Determine the optimizer to be used
2. Determine the learning rate of the optimizer
3. Batch Size

Fit the model with training dataset

Evaluating the performance of the trained model

1. The target to be optimized (accuracy , validation loss)

If satisfied => deploy the model

Actual Situations

- The training process requires a lot of computing resource.

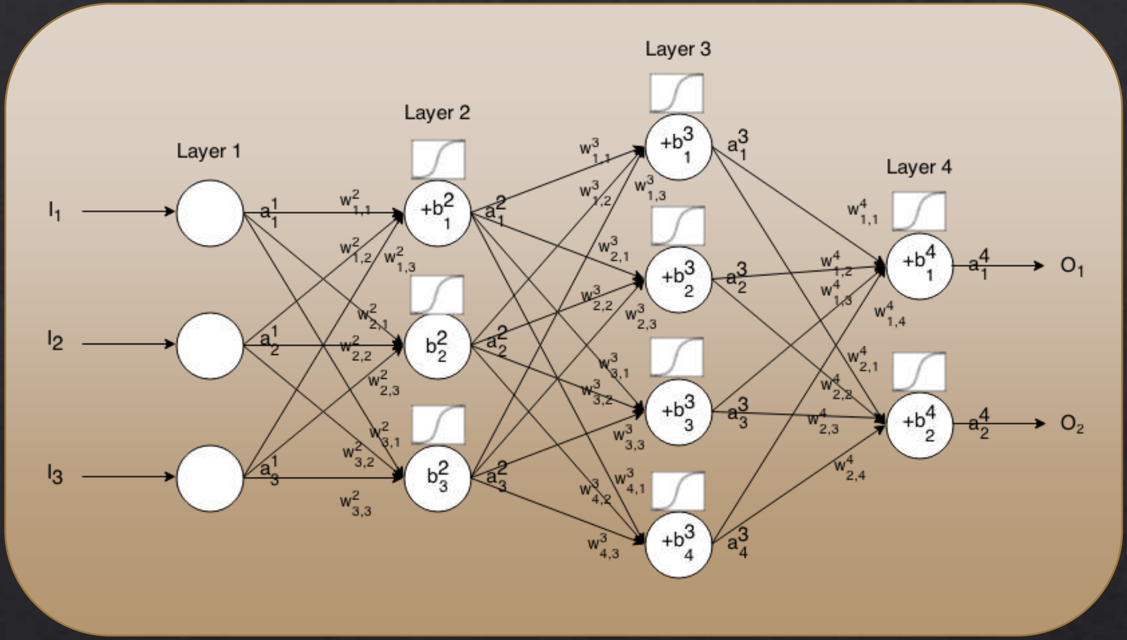
- It’s costly to try different sets of hyperparameters .
  >> takes iterations of training process shown in the previous figure .

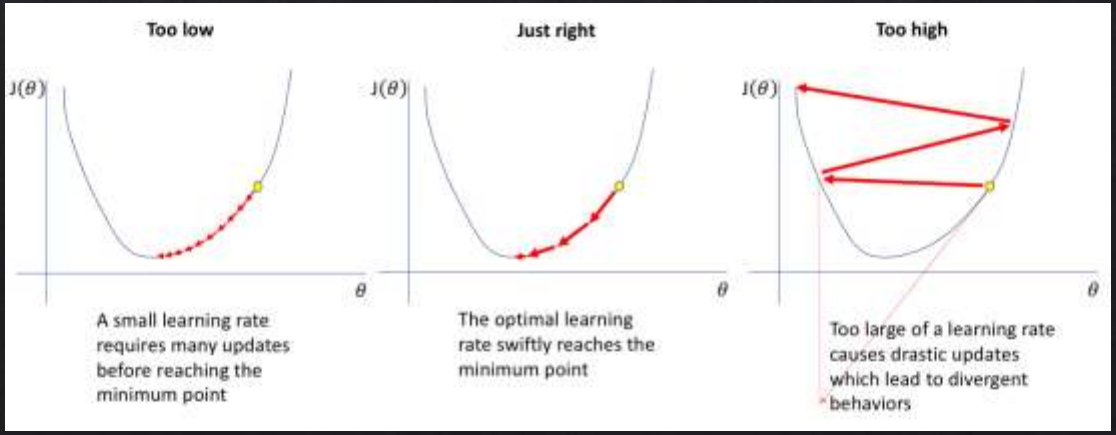
- Bayesian Optimization
  >> minimize the iterations of training process needed

Actual Situations

➢ **Hyperparameters (non - trainable):**
- Learning Rate
- Activation function of each layer
- Dropout rate of target layer
- Number of neurons in Dense layer

➢ **Algorithms :**
- Grid Search
- Random Search
- **Bayesian Optimization**





Motivation & Goal

part 2

Tuning Hyperparameters with Bayesian Optimization

Motivation

- Develop complex deep neural networks.
- A Probabilistic way of tuning hyperparameters .
- Accelerate the life cycle of producing a model.

Goal

- Get a deeper insight into Bayesian Optimization (understanding the mathematics behind)
- Implementing it through python
- Notice its pros and cons and give proper solutions

Structure of B.O.

part 3

Tuning Hyperparameters with Bayesian Optimization

Structure of Bayesian Optimization

Big Picture

- **Bayesian Optimization**

Gaussian Process:

Probabilistic model capable of predicting target function with prior belief .

Acquisition Function:

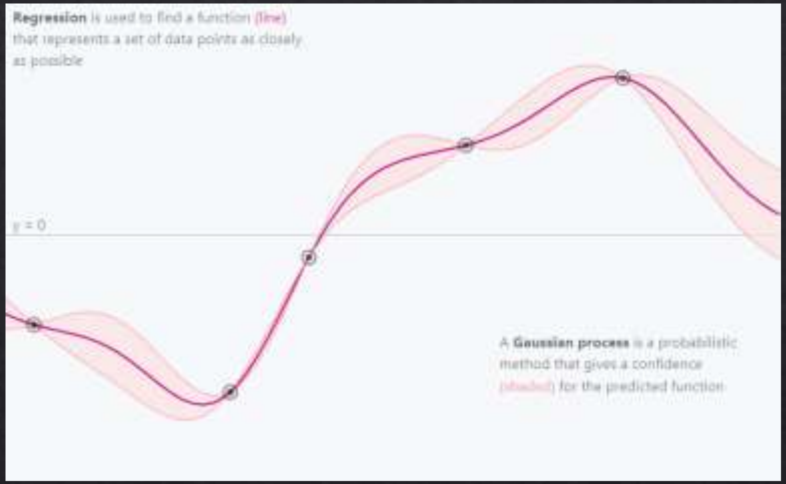
Strategy to find the next set of hyperparameters worth trying .

Structure of Bayesian Optimization

Gaussian Process

- **Core Concept**

$$\begin{bmatrix} f(x') \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} k(x',x') & k(x',X)^T \\ k(x',X) & K_{xx} \end{bmatrix} \right) \qquad k(x',X) = \begin{bmatrix} k(x',x_1) \\ \vdots \\ k(x',x_n) \end{bmatrix}$$

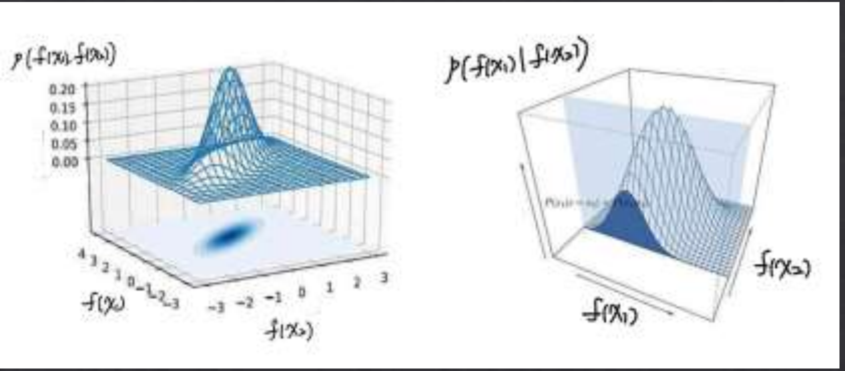


- **Determine the kernel function used**

- **Estimating Posterior with Prior Knowledge**
  - $f(x')|f(X) \sim N(k(x',)^T K_{xx}^{-1} f(X), k(x',x') + k(x',X)^T K_{xx}^{-1} k(x',X))$
  - $\mathbb{E}(f(x')|f(X)) = \Sigma_{i=1}^{n} K_{xx}^{-1} f(x_i) k(x',x_i)$

Structure of Bayesian Optimization

Gaussian Process



f(x) :

performance of the model trained using the specific set of hyperparameters

➢ **Multivariate Normal Distribution**

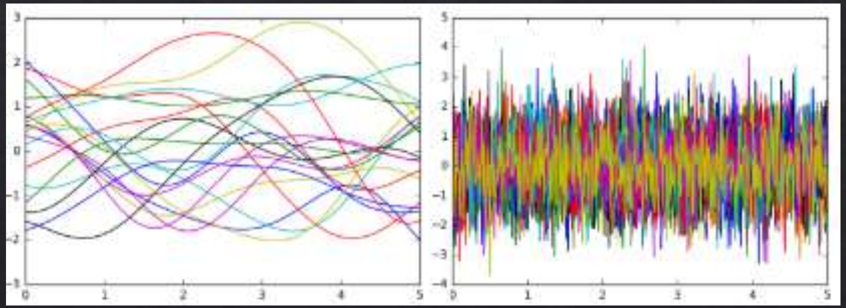
$$\mathcal{N}(x|\ \mu\ ,\Sigma\ ) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} exp[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)]$$

➢ **Kernel Function**

ex: RBF kernel

$$k(x_i, x_j) = exp\left(\frac{-(x_i - x_j)^2}{2r^2}\right)$$

r: length scale of the kernel



Structure of Bayesian Optimization

Acquisition Function

- **Goal : determine the next set of hyperparameters used on training model**

- **Probability of improvement (POI):**

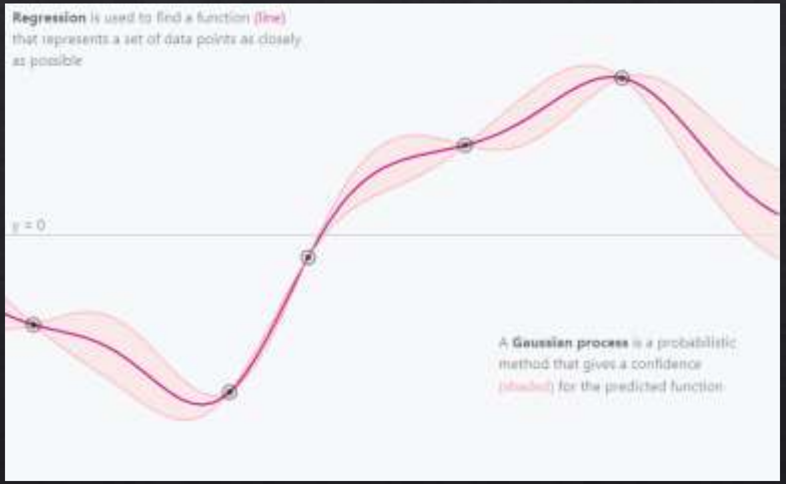
$$POI(x) = P\big(f(x) \geq f(x^+)\big) = \phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right)$$

- **Expected improvement (EI) :**

$$EI(x) = \begin{cases} \big(\mu(x) - f(x^+)\big)\phi(Z) + \sigma(x)\phi(Z) & if\ \sigma(x) > 0 \\ 0 & if\ \sigma(x) = 0 \end{cases}$$

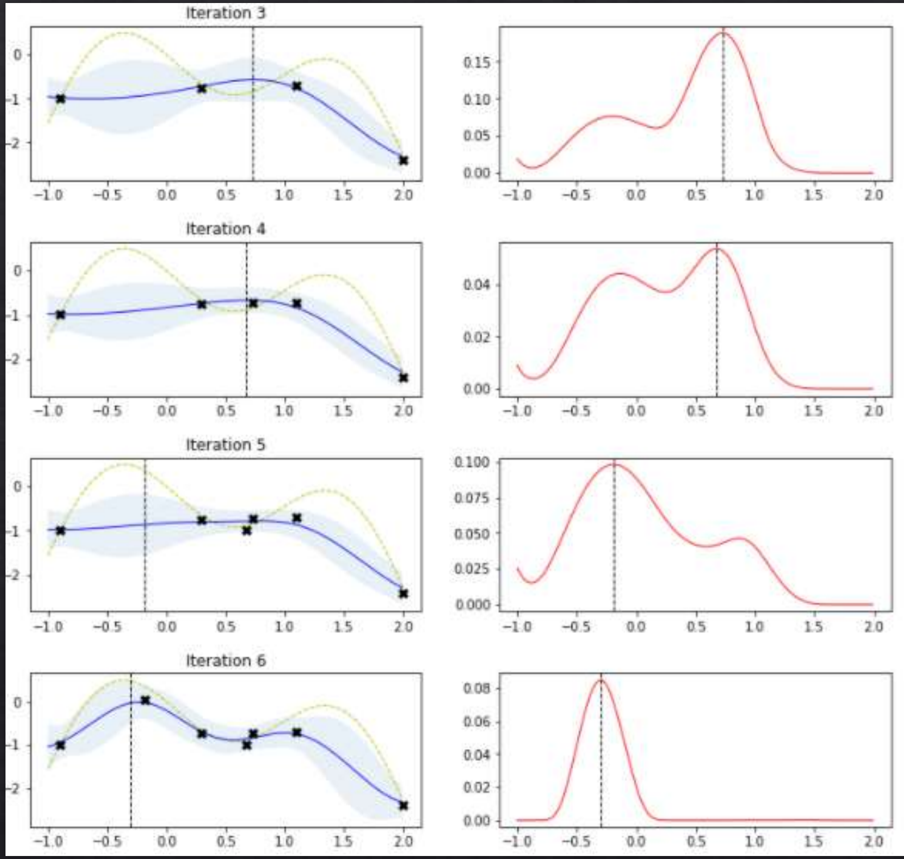
- **Upper confidence bound (UCB):**

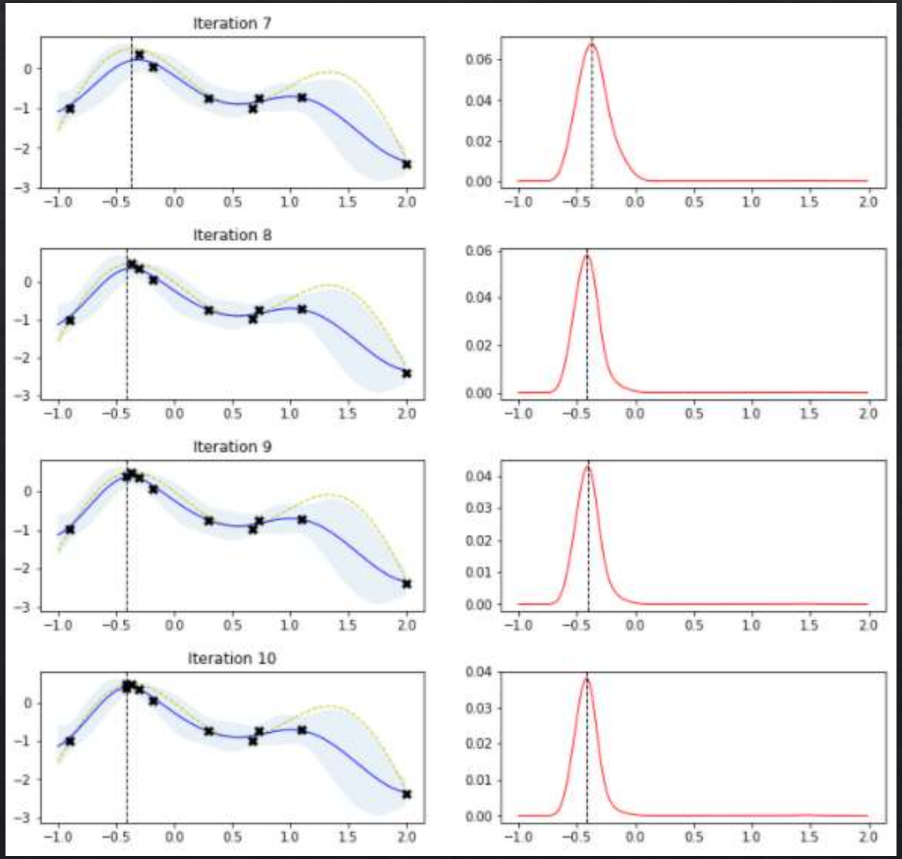
$$UCB(x) = \mu(x) + \kappa\sigma(x)$$



Structure of Bayesian Optimization

Full View





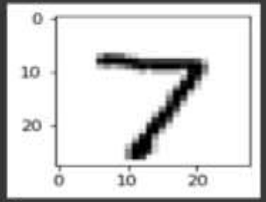
Application on Mnist

part 4

Tuning Hyperparameters with Bayesian Optimization

Application on Mnist model

```
import  numpy  as  np
probability_model(x_test[:5])
plot_image(x_test[0].reshape(28, 28))
result=probability_model.predict(x_test[0:1])
print(np.argmax(result))
```




```
7
```

- After implementation of Bayesian Optimization through python , I applied it to a Mnist model as my first try .

```
def get_model(input_shape,dropout_rate=None):
  model=tf.keras.models.Sequential()
  model.add(l.Flatten(input_shape=input_shape))
  model.add(l.Dropout(dropout_rate))
  model.add(l.Dense(128,activation='relu'))
  model.add(l.Dense(10,activation='softmax'))
  return model
```

- Mission for Bayesian Optimization :

  Tuning dropout rate and learning rate of the model to optimize the accuracy of the test dataset .

Aa Iteration	# Accuracy of test dataset	# dropout rate	# learning rate
12	0.9762	0.1764	0.002729
13	0.9766	0.1775	0.00219

KNN on classifying dogs and cats

part 5

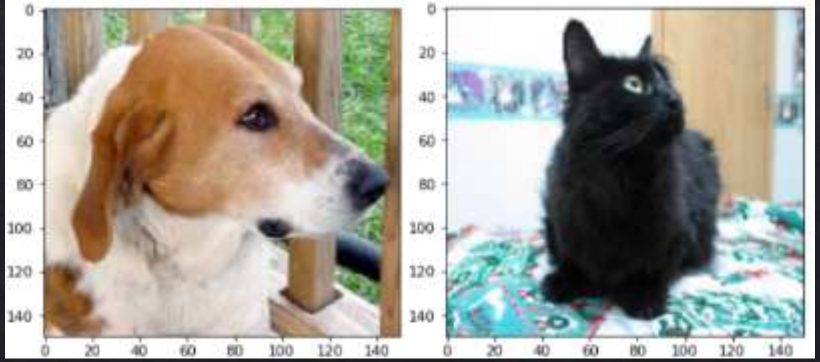
Tuning Hyperparameters with Bayesian Optimization

KNN on classifying dogs & cats

- Mission for Bayesian Optimization :

  Determine the number of dense layers used and the dropout rate of each dense layers while learning rate would be chosen as well .

  Goal : minimize the validation loss .



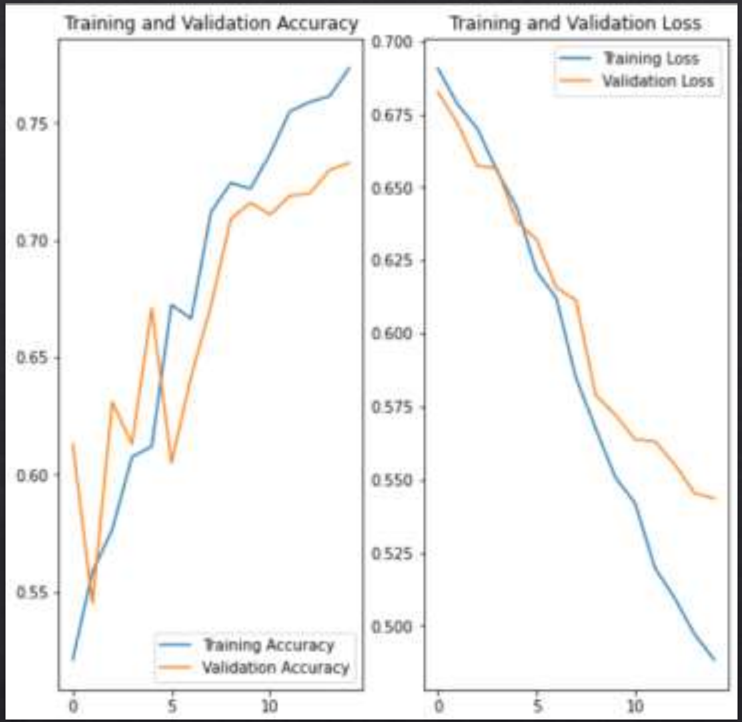
```
def get_model(NUM_DENSE=1,dropout_rate=0):
  model=tf.keras.models.Sequential()
  model.add(l.Conv2D(32,(3,3),activation='relu',input_shape=(150,150,3)))
  model.add(l.MaxPooling2D(2,2))

  model.add(l.Conv2D(64,(3,3),activation='relu'))
  model.add(l.MaxPooling2D(2,2))

  model.add(l.Conv2D(128,(3,3),activation='relu'))
  model.add(l.MaxPooling2D(2,2))

  model.add(l.Flatten())
  for i in range(NUM_DENSE):
    model.add(l.Dropout(dropout_rate))
    model.add(l.Dense(512,activation='relu'))
  model.add(l.Dense(2))

  return model
```

KNN on classifying dogs & cats

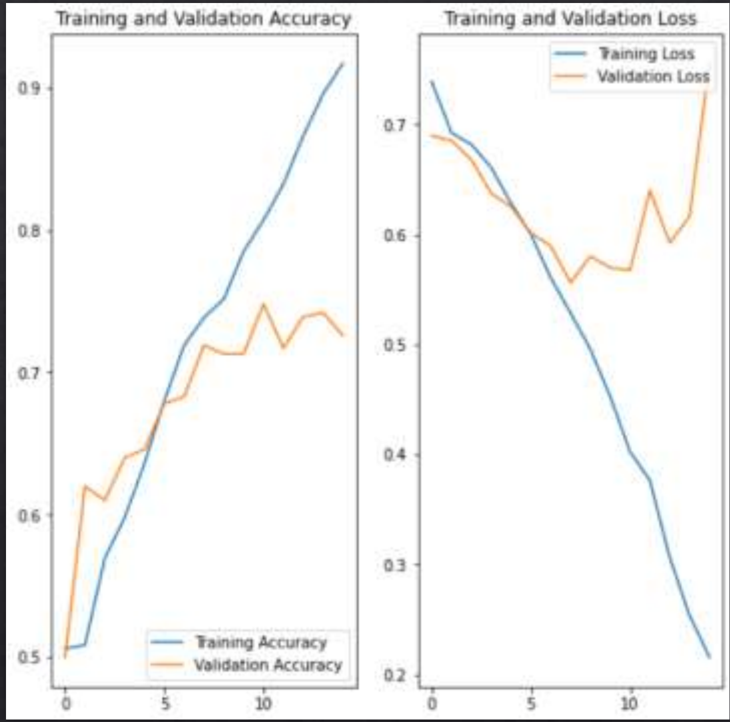
Iteration	Number of Dense Layers	Dropout Rate of Dense Layer	Learning Rate	Test Set Accuracy	Test Loss
Iteration 1	3	0.7042271454095106	0.009998867689308286	0.5	0.6931524276733398
Iteration 2	2	0.3027291235719791	0.009085847911788902	0.5	0.6931489109992981
Iteration 3	1	0.44189250893013343	0.006072002005116367	0.5950000286102295	0.6703671813011169
Iteration 4	0	0.8774479954751082	0.0001	0.7329999804496765	0.5435753464698792
Iteration 5	7	0.20441075994440278	0.0001	0.718999981880188	1.0654526948928833
Iteration 6	0	0.20512040660294945	0.0001	0.7179999947547913	0.561133861541748
Iteration 7	0	0.878349648606946	0.0001	0.7009999752044678	0.56093430519104
Iteration 8	0	0.814340423014835	0.0001	0.7070000171661377	0.5644177198410034
Iteration 9	5	0.88642096332855	0.0001	0.7160000205039978	1.0443283319473267
Iteration 10	0	0.5876140250785195	0.0001	0.6930000185966492	0.5836883187294006
Iteration 11	3	0.6649025612669275	0.0001	0.7039999961853027	0.7287527918815613
Iteration 12	6	0.895918584305263	0.0001	0.7149999737739563	0.9349175691604614
Iteration 13	2	0.8922765480338355	0.0001	0.7170000076293945	0.6769360303878784

KNN on classifying dogs & cats

Bayesian version



Model trained by others



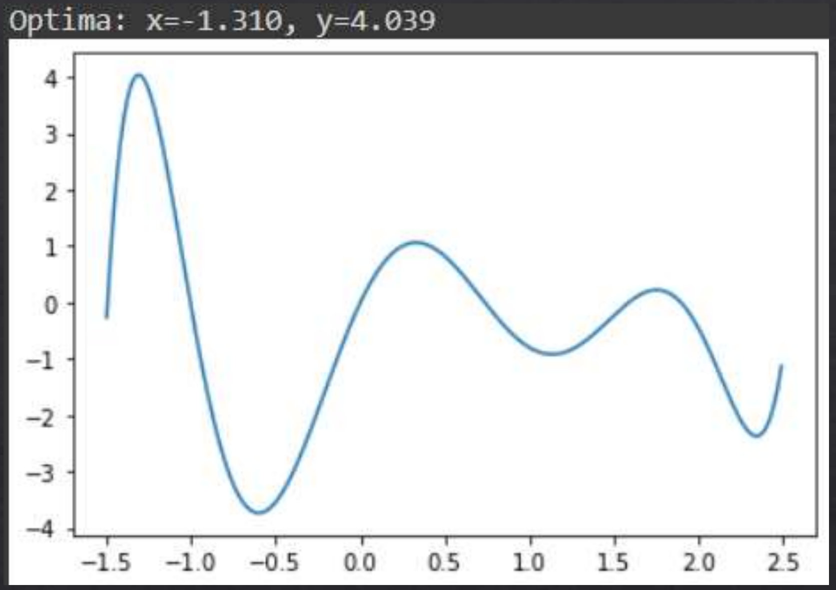
Problem of local maximum

part 6

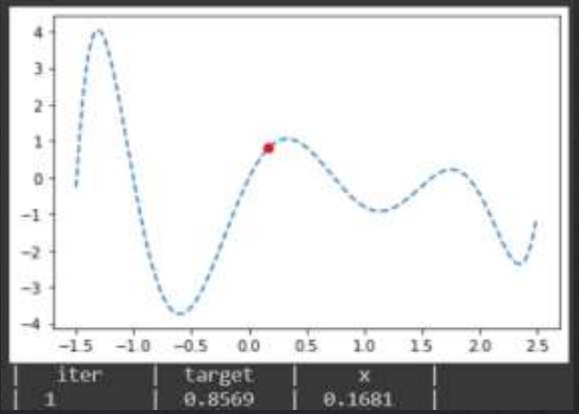
Problem of local maximum

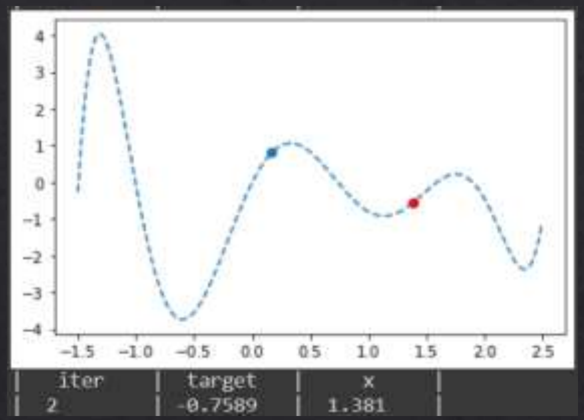
- The target function which Bayesian Optimization aims to find the maximum value of is often a non–convex function .

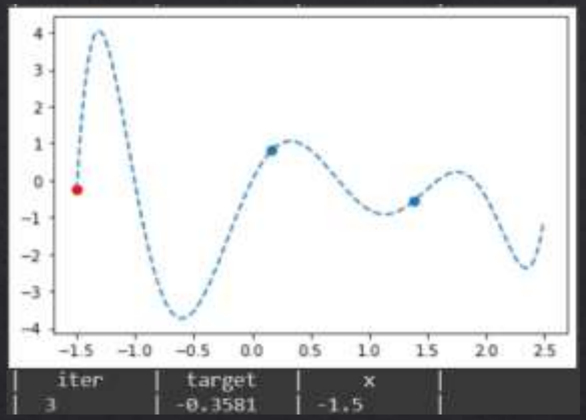
- For acquisition functions with high exploitation , Bayesian Optimization often ends up getting stuck in local maximum .

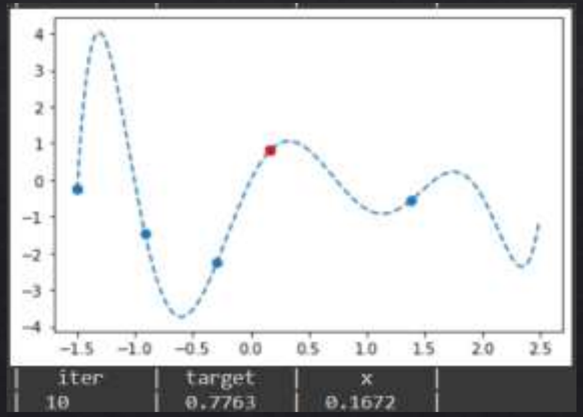


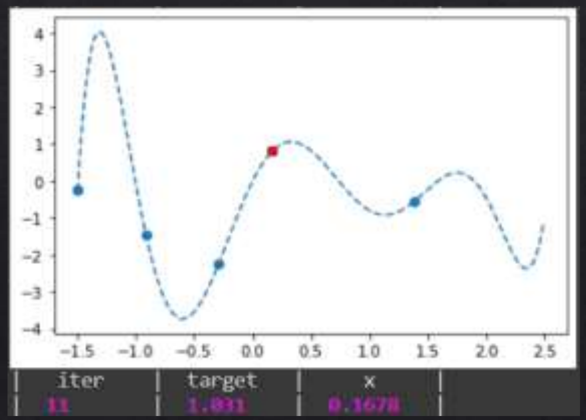
Problem of local maximum

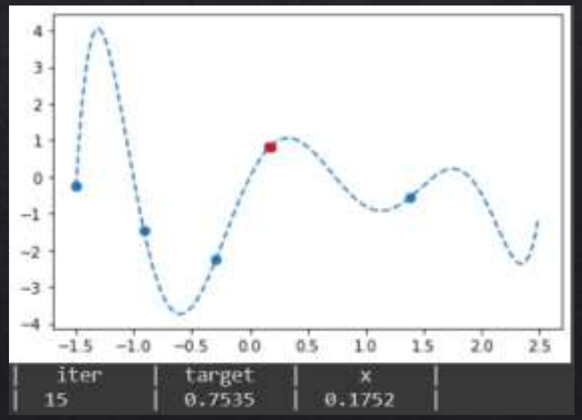








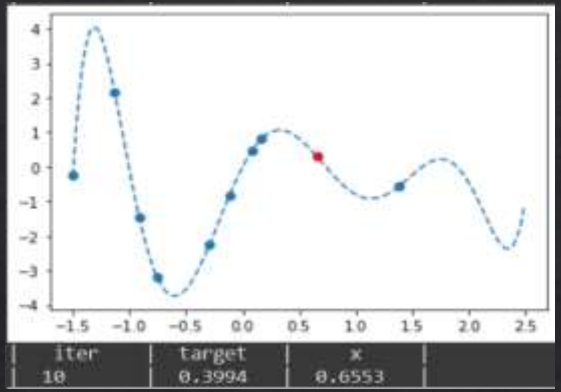


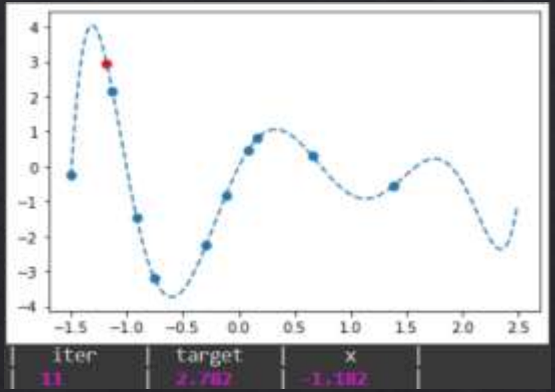


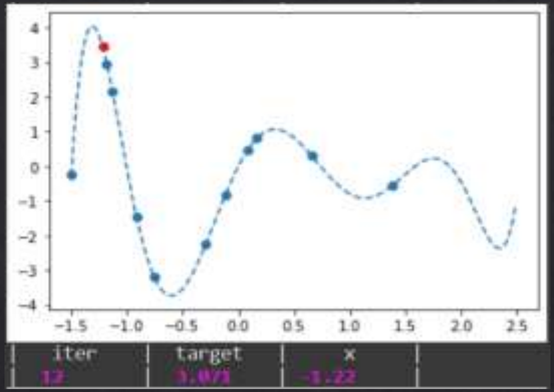
Problem of local maximum

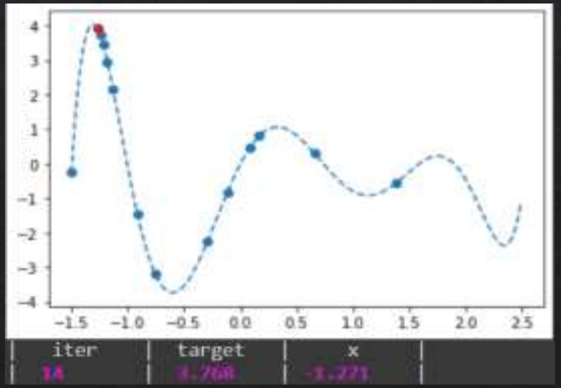
- **To strike a balance between exploitation and exploration:**
    - à Combine the strategy of grid search.
    - à Probe data points according to uniform distribution models .

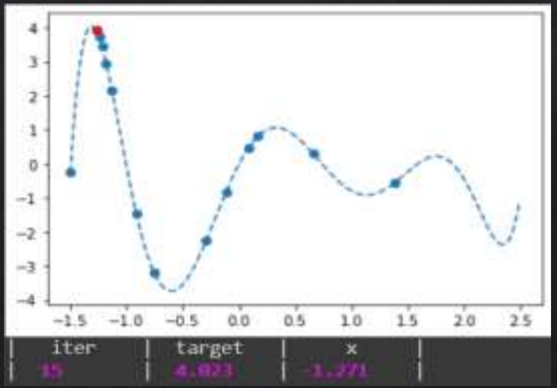
Problem of local maximum

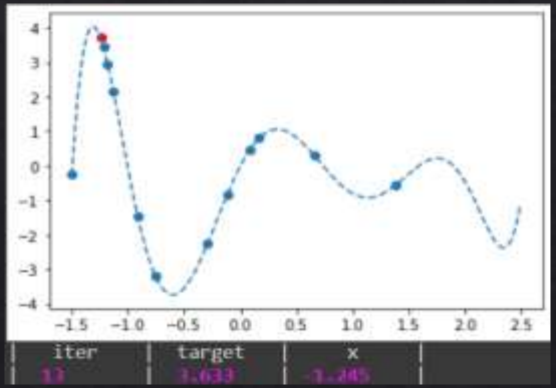












Summary

THANK YOU!