

Tuning Hyperparameters with Bayesian Optimization (B.O.)

作者 : 黃濬程

指導教授 : 謝孫源 博士

Author : Chun-Cheng Huang

Advisor : Sun-Yuan Hsieh Ph.D.

Institution : Department of Computer Science & Information Engineering, National Cheng Kung University

2021/3/27

Introduction

In the process of training a model , there are several parameters required to be determined beforehand . Those parameters include "the Number of dense layers" , "learning rate" , "the type of activation functions" , "dropout rate of the particular layer" , and "loss function" , to name just a few and are referred to as "Hyperparameters" . A well-chosen set of hyperparameters can extract the essence from the training set and contribute to a decent deep learning model . Nevertheless , it is often hard for one to come up with a satisfying set of hyperparameters in the early stage of constructing a deep learning model since human beings are not capable of handling high dimensional problems . Thus , instead of picking up a random set of hyperparameters , taking advantage of probabilistic models to choose a set of hyperparameters would be a wise decision when there is no additional information at hand . Fortunately , Bayesian Optimization is an algorithm which provides us with a probabilistic strategy to search for a proper set of hyperparameters ; therefore , Bayesian Optimization would be explored in more detail below this article .

Motivation & Purpose

As we are experiencing explosive growth of machine/deep learning and its application , accelerating the life cycle of building an well-trained model has become a topic deserves further study. There are many methods such as GPU-accelerated libraries or Adam optimizer to shorten the life cycle of building an well-trained model nowadays . However , there's still room for improvement when facing complicated neural networks . I believe that tuning hyperparameters automatically could serve as a way to reach the goal .

In this article , we aim to give a basic intuition on how Bayesian Optimization works and how it differs from other algorithms on tuning hyperparameters automatically . What's more , we would implement Bayesian Optimization on some simple examples to make further extrapolations .

To sum up , this article would mainly discusses on whether Bayesian Optimization plays an significant role in the future deep learning industry and its pros and cons when utilizing it .

Moreover , the main idea behind Bayesian Optimization , acquisition functions and Gaussian Process , would be abstractly covered in this article . In the end of this article , we hope that the readers could be much more familiar with the term "Bayesian Optimization" and realize the merits and demerits of this algorithm .

Literature Review

Before implementing Bayesian Optimization , the core concept of the algorithm should be illuminated while several terminologies also have to be introduced in case of the ambiguity afterwards . To briefly cover the concept of the algorithm , I'll quote from some well-written articles working on the basis of Bayesian Optimization and the reference would be appended at the end of this article . The main goal of this section would be giving intuitions on how B.O. works .

Gaussian Distribution [2] [3]

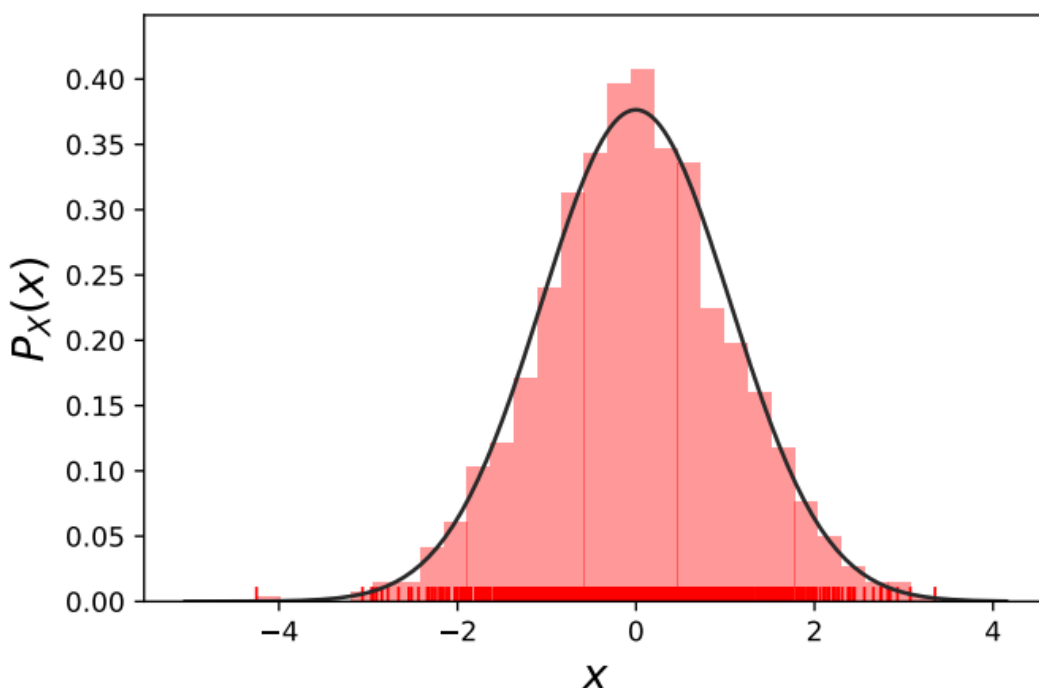
A random variable X is said to be normally distributed if it follows the probability density function below.

mean : μ

variance : σ^2

$$P_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

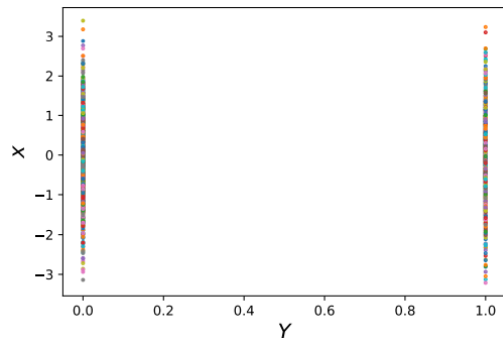
From the article [2] , the author generated 1000 examples which follow the uni-variate normal distribution and plotted those examples as little red vertical bars on X axis as the graph shows . The author also demonstrated the PDF on the vertical axis and fitted it with a bell curve . However , we would focus only on those little red vertical bars here and we should avoid connecting those values of X axis with specific meanings . All we have to know is that the 1000 little red vertical bars follow normal distribution .



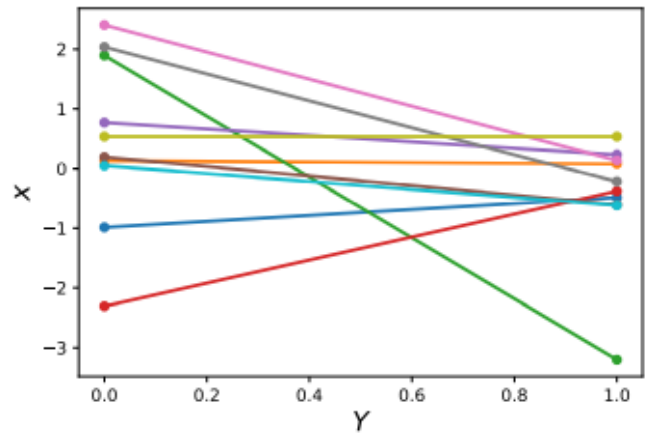
GRAPH FROM [2]

Moreover , normal distribution can be used to make predictions for regression tasks . To illustrate the point of view , the author of the article express the 1000 examples above as

vector \vec{x}_1 and generated another independent set of examples \vec{x}_2 (which also follows gaussian distribution). Next , \vec{x}_1 and \vec{x}_2 were plotted along the vertical axis respectively on $Y = 0$ and $Y = 1$ as the graph shown on the left .Then , the author chose 10 random points respectively from \vec{x}_1, \vec{x}_2 and connected them 1 to 1 randomly . This is just a reminder that \vec{x}_1, \vec{x}_2 was simply partial connected ; however , the two vectors are still independent .

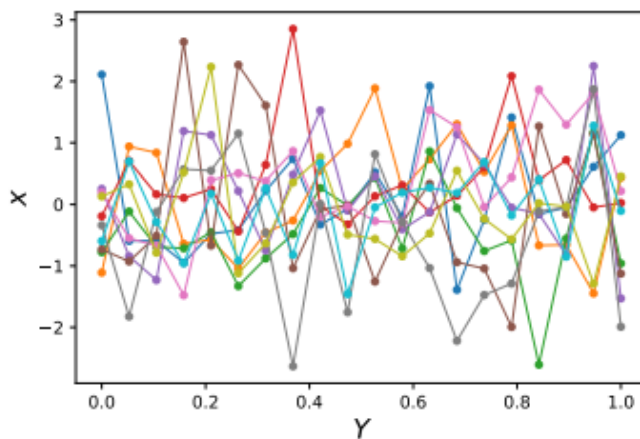


\vec{x}_1 was plotted on $Y = 0$ while \vec{x}_2 was plotted on $Y = 1$ [2]



\vec{x}_1, \vec{x}_2 were connected randomly [2]

There are ten lines in the graph on the right above, and each line can be viewed as a way of making prediction of a unspecified function between interval $[0, 1]$. And since each line is connected by two points which follow two independent normal distribution respectively , we can assign each line with a joint probability $P_{X_1}(x_i) \times P_{X_2}(x_j)$. What's more , rather than only using two vectors , if we plot more vectors between $[0, 1]$, we can make predictions based on probabilistic methods not only on 0 and 1 but also on the points with vectors on the vertical axis . Below is the example of the above idea with 20 vectors along vertical axis scattered on horizontal axis .



twenty vectors partially connected with 10 lines randomly [2]

So far , we've shown what normal distribution is capable of ; Nevertheless , the prediction of unspecified function based on the method above would end up being chaotic since there's no connection between point and point . That is , even if we know some data points of the function beforehand , our belief on other points wouldn't be updated . For the purpose of solving this problem , Multivariate Normal Distribution would be introduced down below .

Last but not least , it's also important to keep in mind that the twenty vectors in the graph above can be viewed as twenty independent Random Variables $f(Y = 0), f(Y =$

$a_i), \dots, f(Y = 1)$. So what multivariate normal distribution do would be building up the connections between those random variables .

Multivariate Normal Distribution [2]

To begin with , we will take a look at how probability density function of an Multivariate Normal Distribution is defined :

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

x : multidimensional variable

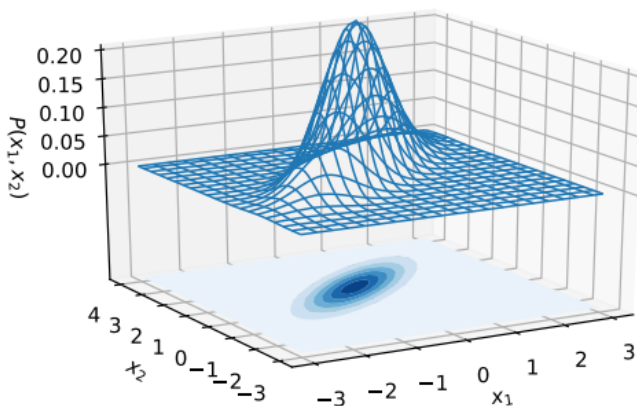
D : number of Dimension of the input

μ : expected value of multidimensional variable

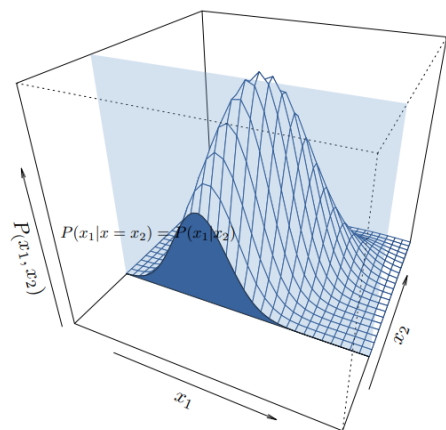
Σ : covariance matrix which stores the relationships between variable in different dimension

With a view to making the concept of multivariate normal distribution much more concrete , we would take bi-variate normal distribution as our example . This should help us on getting familiar with the complex symbols of the equation and getting a grasp of the properties of multivariate normal distribution .

First , we assume that there are two random variables (x_1, x_2) follow multivariate normal distribution . And the visualization of how x_1, x_2 are distributed can be shown in 3-D plot with the z-axis being the probability of the distribution at that point .

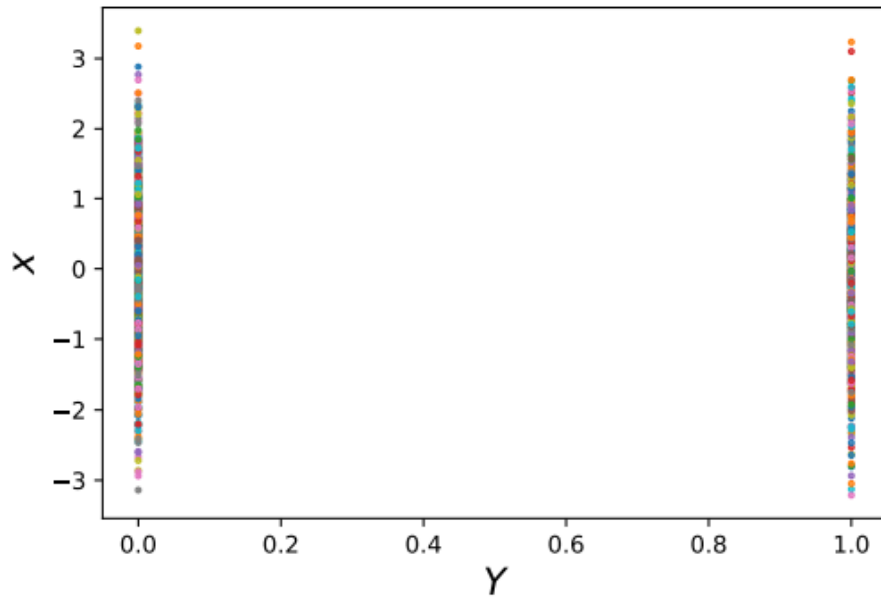


reference [2]



reference [2]

Practically , if we know the value of x_2 in advance , we can take a slice from the 3D curve along the value of x_2 . After that , we can make a wild guess on the possible value of x_1 based on that slice . In other words , we take advantage of the Bayesian Theorem based on our prior knowledge x_2 . In addition , multivariate normal distribution has a nice property that the conditional probability of the specific random variable would also follow normal distribution .



\vec{x}_1 was plotted on $Y = 0$ while \vec{x}_2 was plotted on $Y = 1$ reference [2]

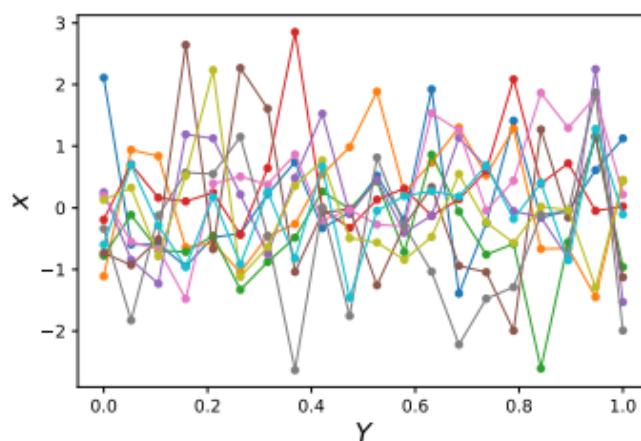
What's more , recalling the example we used in uni-variate gaussian process , $f(Y = 0)$ and $f(Y = 1)$ are two independent random variable as the graph shows . However , what if we assume that they are bi-variate normal distributed ? It would be exciting to find out that if we know the value of $f(Y = 0)$ in advance , we can make a guess on the value of $f(Y = 1)$ based on the slice we get from the 3D bell curve constructed by $f(Y = 0)$ and $f(Y = 1)$. To sum up , building relationships between different random variables would be the most important intuition of gaussian process and it would be utilized very often afterwards .

The key to build up relationships between random variables would be Σ - Covariance Matrix .

$$\Sigma = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) & \dots & k(x_2, x_n) \\ \dots & \dots & \dots & \dots & \dots \\ k(x_n, x_1) & k(x_n, x_2) & k(x_n, x_3) & \dots & k(x_n, x_n) \end{bmatrix}$$

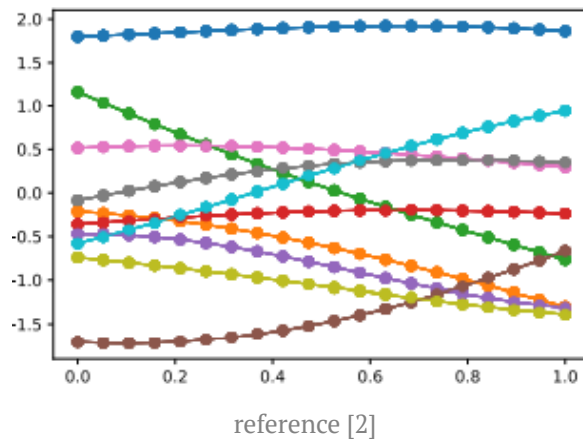
$k(x_i, x_j)$ is the kernel function determines the covariance between x_i and x_j

The graph below is the example we used above . Besides from viewing it as connecting 20 independent normal distributions with 10 lines , we can recognize it as 10 examples generated by twenty-variate normal distributions with covariance matrix (kernel) being identity matrix , i.e. 20 random variables ($f(Y_1), f(Y_2) \dots f(Y_{20})$) are independent from others .



twenty vectors partially connected with 10 lines randomly reference [2]

However, if we use "RBF kernel" as our covariance matrix rather than identity matrix and generate ten examples, we could get a graph with ten smoother lines shown below. These ten lines were assigned a probability based on multivariate normal distribution respectively. Thus, we can imagine that with multivariate normal distribution, we assigned a specific probability to every possible function on domain $Y = [0, 1]$. Moreover, we can use more than twenty dimension of multivariate normal distribution to approximate our target function and it might yield a better result for us. ($f(Y_1), f(Y_2) \dots f(Y_n)$)

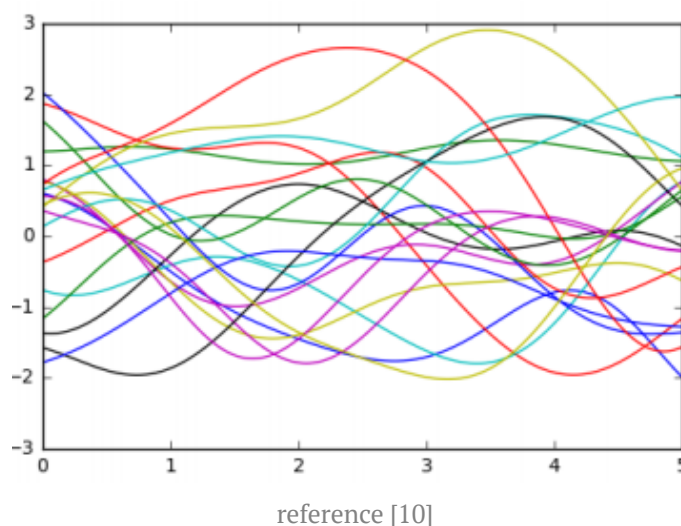


"RBF kernel" is one of the most frequently used kernel nowadays, RBF function is used as covariance function and is defined as: $k(x_i, x_j) = \exp\left(\frac{-(x_i - x_j)^2}{2\gamma^2}\right)$ where γ is the length scale of the covariance matrix.

So far, we've got acquainted with the capacity of multivariate normal distribution. It provides us with a probabilistic way of making predictions on a specific function. However, it should be noted that the covariance matrix (Σ) and the expectation value of multidimensional variable (μ) needed to be set beforehand. That is to say, we would determine how random variables are related to others. Here, we would show what would happen to our predictions on functions with different kernels applied.

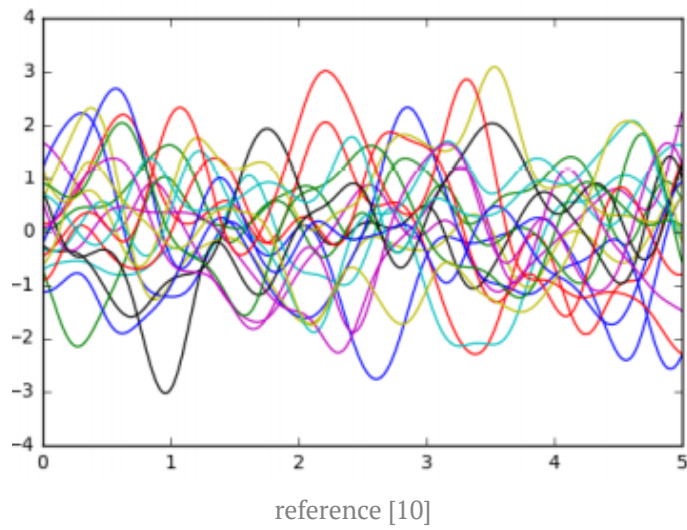
- RBF kernel

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



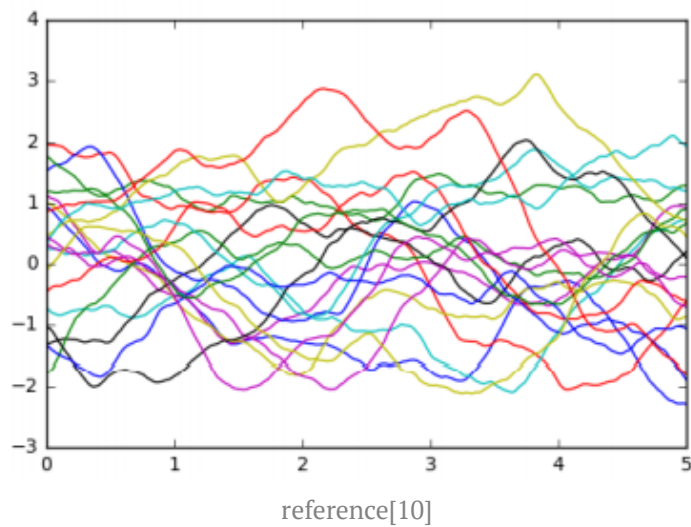
- Squared-exponential kernel

$$k(x_i, x_j) = \exp\left(-\frac{1}{2} \frac{(x_i - x_j)^2}{0.25^2}\right)$$



- Matern 3 kernel

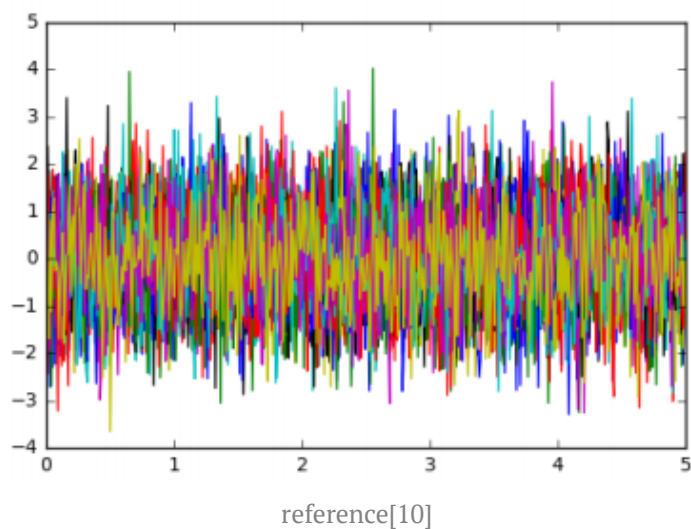
$$k(x_i, x_j) \sim (1 + |x_i - x_j|) \exp(-|x_i - x_j|)$$



- White Noise kernel

$$\text{if } x_i == x_j \rightarrow k(x_i, x_j) = 1 \text{ else } k(x_i, x_j) = 0$$

(i.e. the kernel function would construct an identity matrix)



How multivariate normal distribution works with predictions have been stated above , and it is the most important mathematics basis behind Bayesian Optimization . Next , Gaussian Process would be the protagonist in further discussion . Gaussian Process is a kind of

practical application of multivariate normal distribution and it also serves as surrogate model in Bayesian Optimization .

Gaussian Process [3]

Briefly saying , with our prior belief on μ , Σ and some observation points of an unknown function , gaussian process aims to apply the technique mentioned above while fitting those observation points . In other words , gaussian process gives a probability to every function passing through those observation points . Moreover , the mean function yielded by gaussian process would be our best bet to approximate the target function (how the mean function was derived would be introduced below) . Furthermore , gaussian process has the nice property of being closed under marginalization and conditioning ; thus , when we get a new observation point , gaussian process can update its approximation of the function efficiently .

Assume that we have owned a few observation points $f(x_1), f(x_2), f(x_3), \dots, f(x_n)$. Now , we are interested in an unknown point $f(x')$ and here we're going to apply gaussian process to predict the value of $f(x')$. As we mentioned in the previous section , the kernel function and μ have to be determined in advance ; here , we would simply denote our kernel function as $k(x_1, x_2)$. Following , since gaussian process is closed under conditioning , the joint distribution of $[f(x'), f(x_1), f(x_2), \dots, f(x_n)]^T$ also follows multivariate normal distribution . (We will assume that $\mu = 0$.)

$$\begin{bmatrix} f(x') \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} \sim N \left(\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} k(x', x') & k(x', x)^T \\ k(x', x) & K_{xx} \end{bmatrix} \right) \quad k(x', x) = \begin{bmatrix} k(x', x_1) \\ k(x', x_2) \\ \vdots \\ k(x', x_n) \end{bmatrix}$$

Next , with conditioning rule , we can obtain the posterior of $f(x')$ is gaussian .

$$f(x')|f(x) \sim N(k(x', x)^T K_{xx}^{-1} f(x) , k(x', x') + k(x', x)^T K_{xx}^{-1} k(x', x))$$

Furthermore , with the above equation , we can estimate the posterior mean of $f(x')$

$$\mathbb{E}(f(x')|f(x)) = \sum_{i=1}^n K_{xx}^{-1} f(x_i) k(x', x_i)$$

By the steps above , we can get the approximation on $f(x')$, which is the value of the conditional expectation . Moreover , we could recursively implement those steps to map out our target function with high dimensional gaussian distribution . Last but not least , when we get a new data point of the target function , we could update our belief with the steps shown above . Below would be the visualization of *Gaussian Process* .

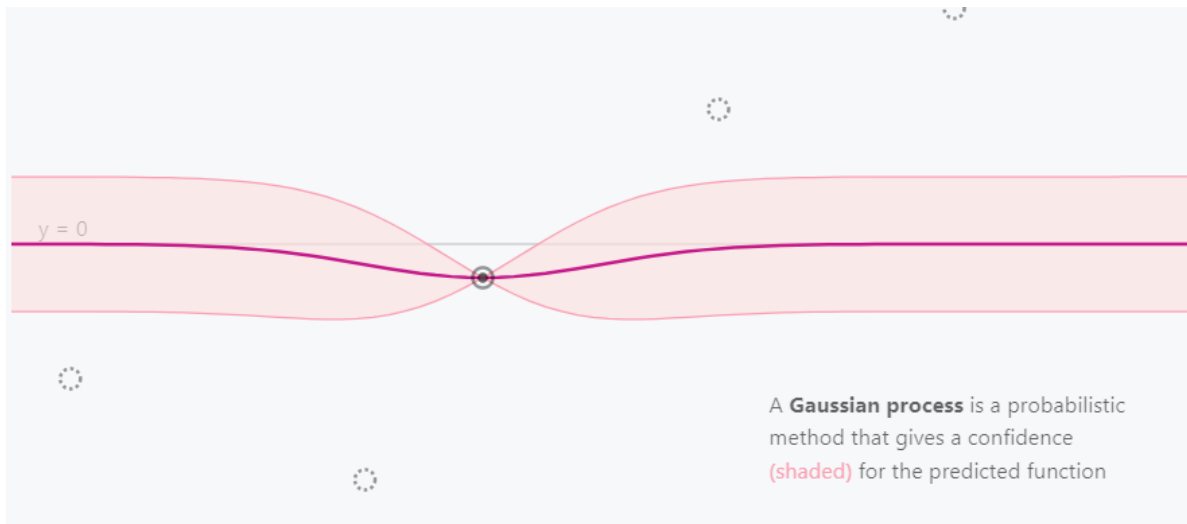


figure 1 -reference[3]

As figure 1 shows , Gaussian Process would try to fit the observation point and make predictions on each point in the domain continuously . The solid line is the mean function derived from the above mathematic equation while the shaded part are other points following multivariate normal distribution .

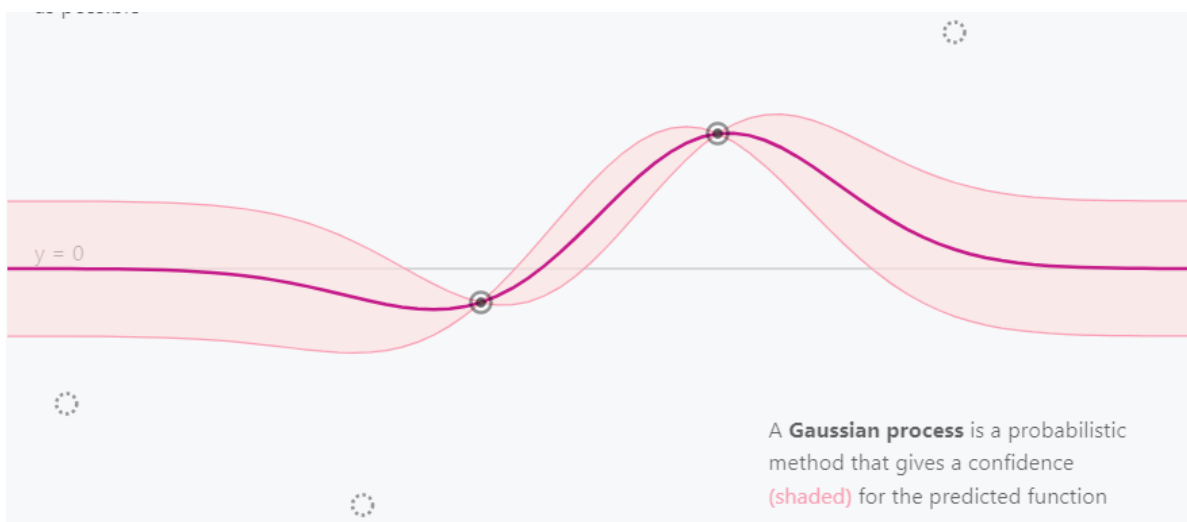


figure 2-reference[3]

After we get a new observation point as shown in figure 2 , we can discover that Gaussian Process updates its confidence on the target function . The mean function is updated meanwhile , and it could be used as our prediction of target function .

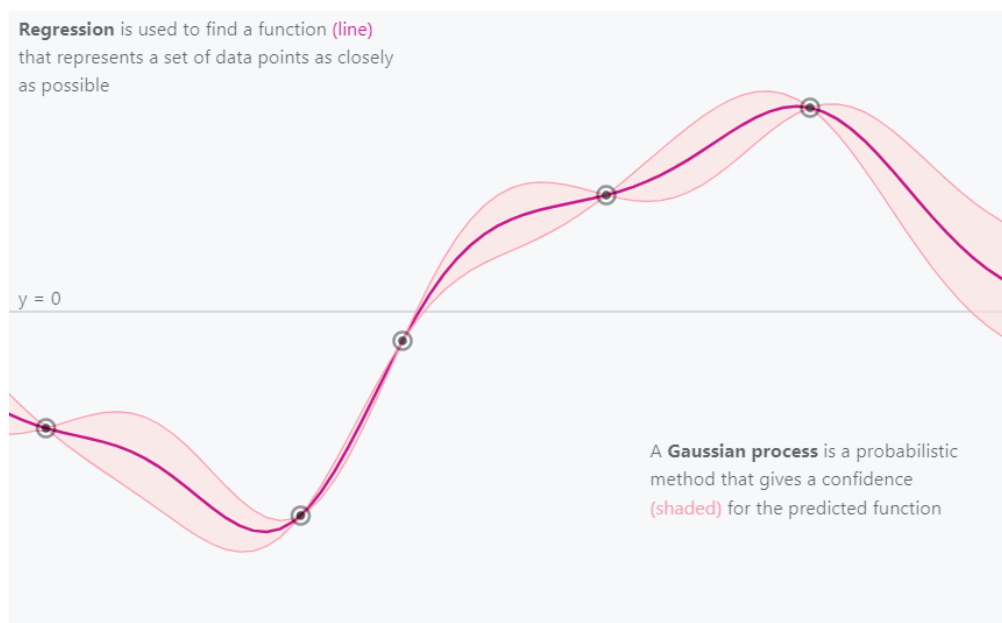


figure 3 -reference[3]

As we possess more and more observation points , the uncertainty of Gaussian Process narrows and the prediction on rest of the points should be more accurate .

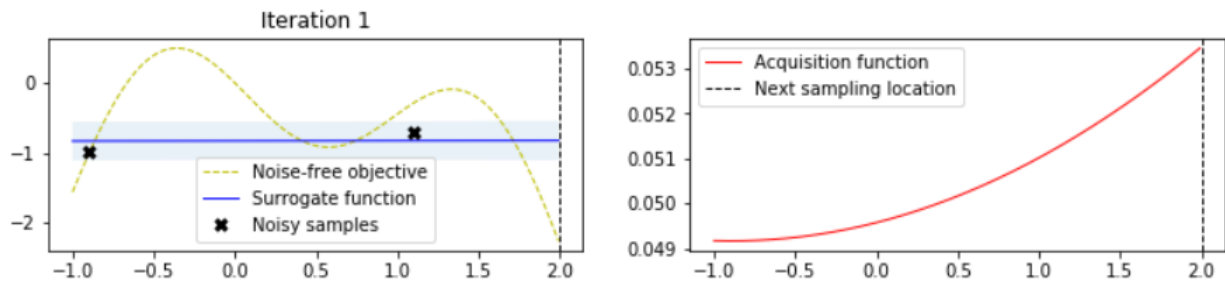
Bayesian Optimization

So far , we are able to utilize Gaussian Process to approximate a specific function . In Bayesian Optimization , what Gaussian Process would try to predict is the function of hyperparameters which outputs the variables representing the quality of our trained model (such as accuracy of test dataset) . Here , Bayesian Optimization takes over the predicted function generated by Gaussian Process and determine the next observation point according to the result of acquisition function. Bayesian Optimization is suitable for complex deep neural networks , since it allows programmers using less iterations to find a decent set of hyperparameters while it's usually costly to train a deep neural network repeatedly . Before showing the visualization of Bayesian Optimization , I'm going to give a brief explanation on what acquisition function is .

There are three commonly used acquisition functions in Bayesian Optimization .

- Probability of Improvement (PI)
- Expected Improvement (EI)
- Lower Confidence Bound (LCB)

Basically , those algorithms are aimed at finding the next probing point with the prediction of Gaussian Process . And these algorithms have its trade-off between exploitative and explorative respectively . The mathematic equation is relatively simple so I'm not going list them out . Next , with all the theorems above , we're able to get a grasp of the big picture of Bayesian Optimization easily .



graph 1

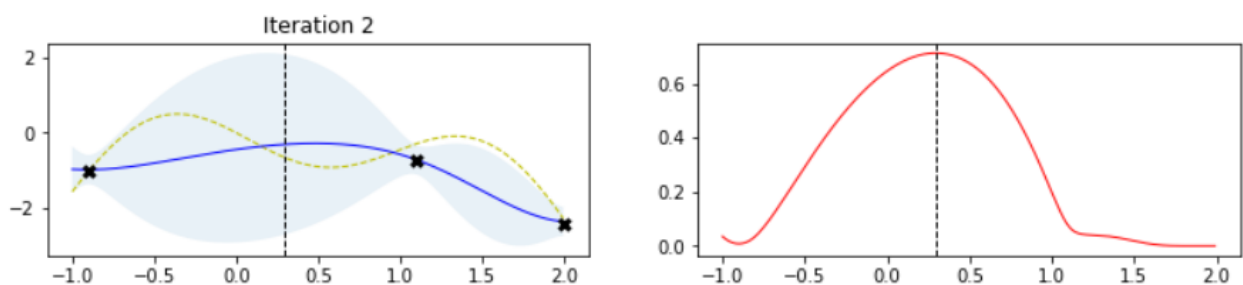
First things first , the x axis of the figure on the left side can be viewed as " a set of hyperparameters " while the the y axis stands for " an estimation of how satisfying the set of hyperparameters is (which is often accuracy of the trained model) ". Second , we have to keep in mind that the dashed line is the true condition of the function which we do not know before using different sets of hyperparameters to train the model .

Thus ,as the figure on the left shows , we've trained two different models with two sets of hyperparameters. Nevertheless , if we take a peek at the true answer (dashed line) , both sets of hyperparameters are not suitable to be the optimal hyperparameters since the optimal hyperparameters should be around -0.5 according to the dashed line .

However , what's the next point we should try ? That is to say , what is the next set of hyperparameters should we use to train our model ?

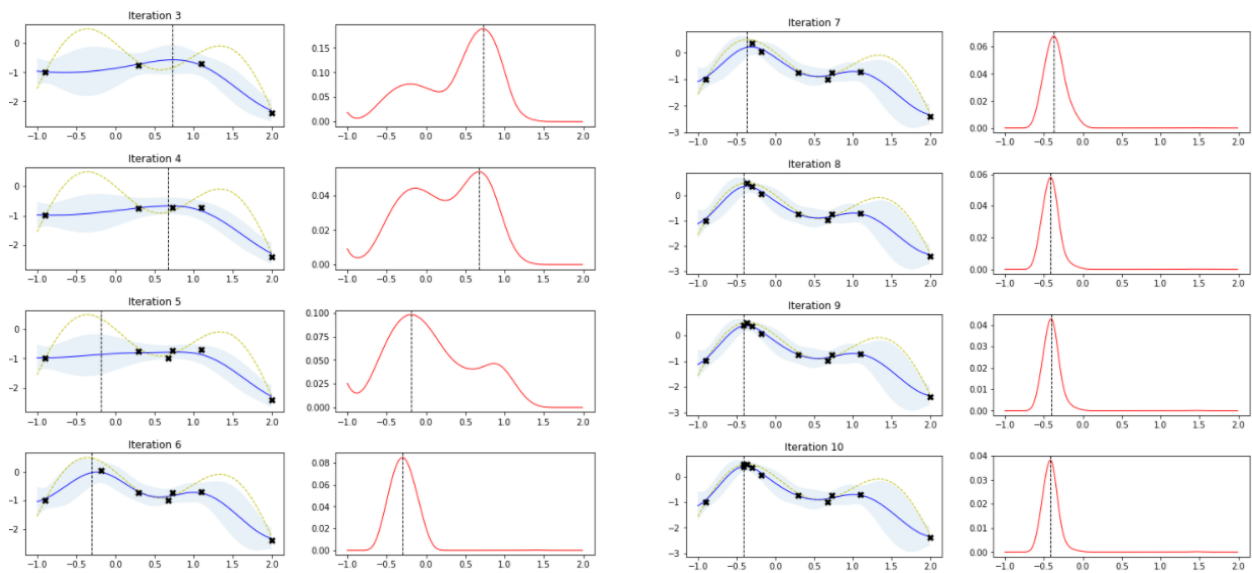
Based on the solid line constructed by gaussian process , Bayesian Optimization would check uniformly on x axis and estimate which point owns the largest potential improvement with acquisition function . In this example, Bayesian Optimization would choose the rightmost point as the new set of hyperparameters. And it would be the set of hyperparameters of the model we're going to train in the next iteration.

Until now, it seems quite vague for one to realize what Bayesian Optimization is trying to do , but I believe that with more examples below , the concept of Bayesian Optimization would be clear.



graph 2

As we said in the iteration 1 , the rightmost point is the set of hyperparameters of the new model. Nonetheless , the figure on the left shows the new model we've trained performs even worse than the other two models . Consequently , Bayesian Optimization finds the new set of hyperparameters for the next try. (which is about 0.25 on the right figure)



graph 3

graph 4

Through iterations of training new models with different set of hyperparameters, we can observe that Bayesian Optimization find a decent approximate of the optimal hyperparameters.

With the optimal hyperparameters we get from the Bayesian Optimization, I believe that the model could be quite satisfying. Above all, the optimal hyperparameters is automatically found rather than manually found, which could save our time and effort with this probabilistic method.

To sum up, with Bayesian Optimization , it's more probable for us to find a decent set of hyperparameters with minimum iterations of training different models.

Research Methods

- Implement Bayesian Optimization through python code on colab .
- Compare the result and verify whether Bayesian Optimization functions well .
- Finding pros and cons of Bayesian Optimization .
- Visualization of the result.

Results & Discussion

In this section , we would implement Bayesian Optimization on a few classic machine learning models and make further discussions on those specific examples .

Mnist Model

- Part of the code

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

def fit_with(input_shape, verbose, dropout_rate, lr):
    model=get_model(input_shape, dropout_rate) //get the model above
    optimizer=rmsprop.RMSprop(learning_rate=lr)
```

```

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer=optimizer,
              metrics=['accuracy'])
model.fit(x=x_train,y=y_train,epochs=5,verbose=0)
score=model.evaluate(x=x_test,y=y_test,verbose=0)

return score[1]

//the domain BayesianOptimization explore
pbounds={'dropout_rate':(0.1,0.5),'lr':(1e-4,1e-2)}

optimizer=BayesianOptimization(
    f=fit_with_partial,
    pbounds=pbounds,
    verbose=2,
    random_state=1,
)

optimizer.maximize(init_points=5,n_iter=10,acq='poi',)

```

As the code shown above , for this model , our hyperparameters are dropout_rate and learning_rate . What Bayesian Optimization would do here is to find the best set of dropout_rate and learning_rate to maximize the accuracy on test dataset from keras mnist. And the result is down below .

(In this example , the kernel of the gaussian process is Matern 3/2 and the acquisition function adopted is POI .)

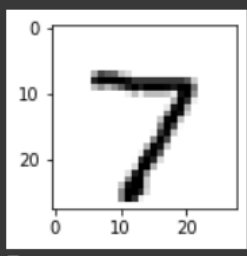
iter	target	dropou...	lr
1	0.9702	0.2668	0.007231
2	0.9717	0.1	0.003093
3	0.9734	0.1587	0.001014
4	0.9745	0.1745	0.003521
5	0.9712	0.2587	0.005434
6	0.9493	0.1	0.000100
7	0.9647	0.1012	0.00982
8	0.9748	0.1745	0.003521
9	0.9691	0.1003	0.005993
10	0.9616	0.1087	0.01
11	0.973	0.1624	0.002776
12	0.9762	0.1764	0.002729
13	0.9766	0.1775	0.00219
14	0.9751	0.1783	0.001697
15	0.9708	0.178	0.003953

Each iteration of Bayesian Optimization (target stands for the value of model.evaluate(x=x_test,y=y_test,verbose=0))

```

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

```



prediction of the best model we have got

As the template demonstrated on the left , we can see that in training phase , Bayesian Optimization would choose different set of hyperparameters and get our observation point in every iteration . According to the template , it is indeed that Bayesian Optimization keeps making progress on finding better set of hyperparameters and it converges roughly to the set {"dropout_rate"= 0.1775 , "learning_rate"=0.00219 } , which leads to 97.66% accuracy of prediction on test dataset .

From this example , we discovered that Bayesian Optimization do have the capability to find satisfying results and even make further progress with a few iterations . After realizing the subtle and fascinating power that Bayesian Optimization holds , we would move on to the next example .

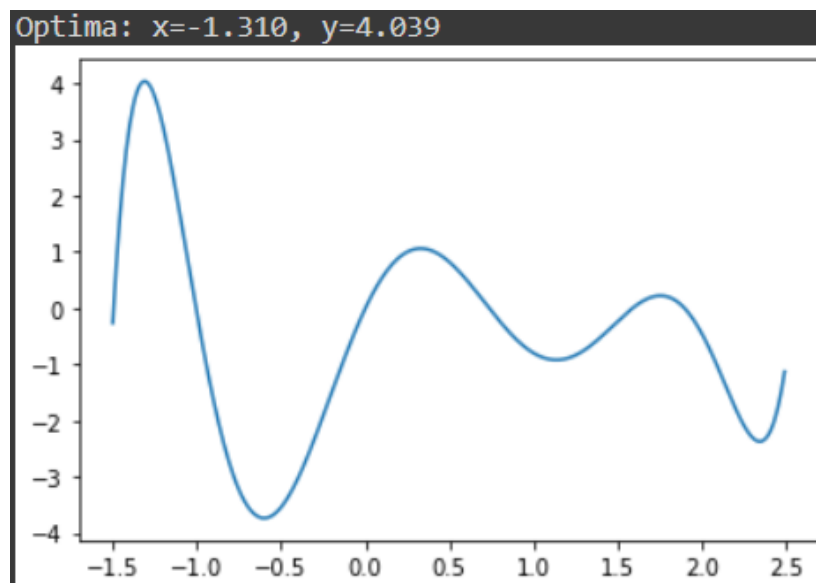
Problem of Global Maximum

Although Bayesian Optimization has done well on the previous example, we think there's more to ponder on. In most of the techniques finding optimum of specific function, it is inevitable that we might stuck in local optimum when our target function is non-convex function and Bayesian Optimization is no exception. In this example, we will exert Bayesian Optimization on an non-convex function.

(In this example, the kernel of the gaussian process is Matern 3/2 and the acquisition function adopted is EI.)

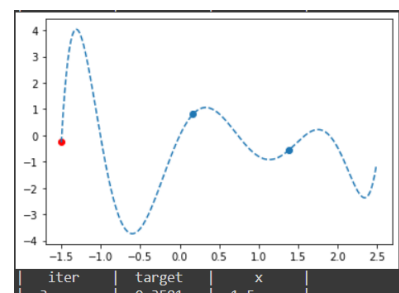
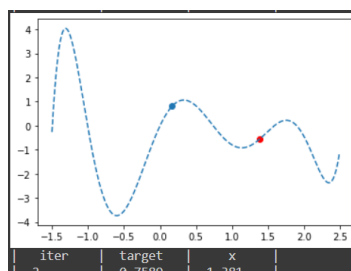
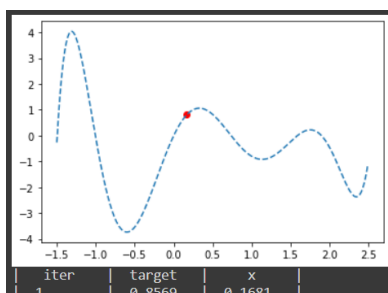
```
def objective(x,noise=0.1):  
    noise= normal(loc=0,scale=noise)  
    return 0.75*x**7-3.19*x**6+0.66*x**5+10.64*x**4-8.09*x**3-7.90*x**2+6.33*x+0+noise
```

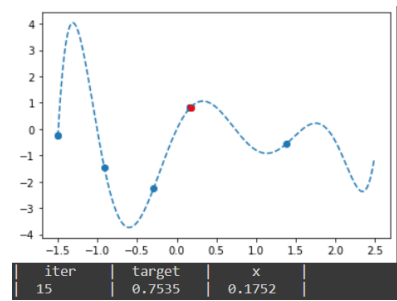
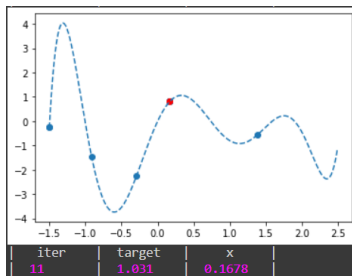
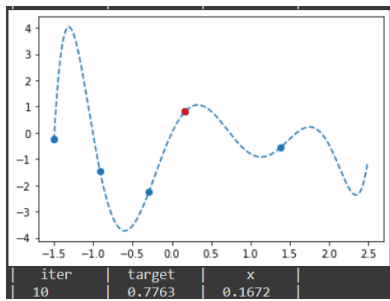
Above is the non-convex target function and we would utilize Bayesian Optimization to find its global maximum. We add some noises to simulate the real world problem since it's not always possible for us to get accurate observation points of the specific function.



Visualization of the target function and the optima of the function

The visualization of the objective function and the optima point are listed in the graph up there. We should notice that programmers doesn't know those information beforehand. Now, let's see what if we utilize Bayesian Optimization on the objective function. In each iteration, we would probe an observation point through Bayesian Optimization and we would recursively implement this behavior 15 iterations.

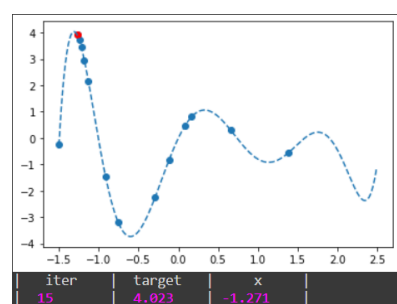
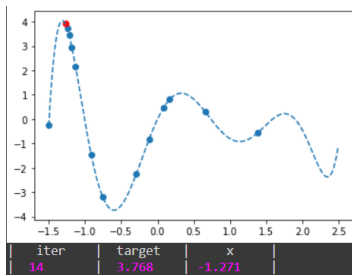
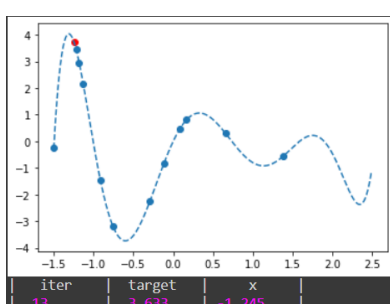
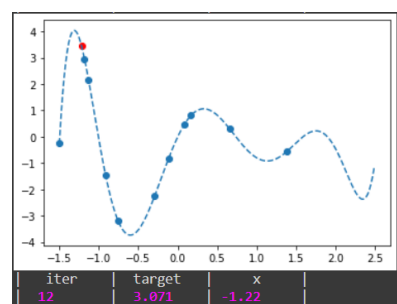
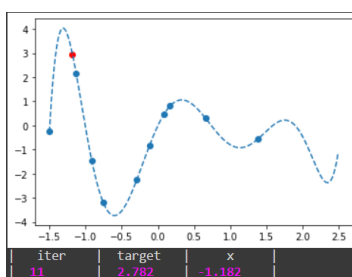
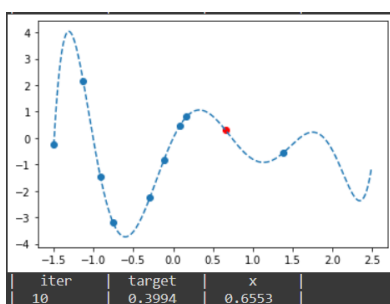




From the result shown above , we could find out that Bayesian Optimization indeed has the problem of sticking in the local maximum . And here is my solution based on many times of trials and errors .

To begin with , we know that as we own more and more prior knowledge of the target function , Gaussian Process would update its prediction on the target function . However , when Bayesian Optimization converges , the new observation point we probed is often adjacency to the previous observation points . Thus , the update belief of Gaussian Process would only has a slightly difference from the previous belief . To rephrase it , under such cases , Gaussian Process is not able to catch the big picture of our target function and make precise predictions ; therefore , it often ends up that acquisition function suggests the wrong point to probe in the next iteration . In consequence , our mission is to help Gaussian Process gain the big picture of our function .

To reach the goal discussed above , before reinforcing Bayesian Optimization , we would uniformly probed a several points of our target function in advance . The more continual our uniformly probed observation points are , the better our gaussian process would fit the function . Now , we would show what would happen if we own 10 uniformly distributed observation points of the objective function before implementing Bayesian Optimization .



As the graphs illustrated above , we can find out that though both implementations of Bayesian Optimization take 15 iterations to find the maximum value of the objective

function , the latter implementation with 10 uniformly distributed observation points has resolved the problem of getting stuck in local maximum . Moreover, it only takes 5 iterations of Bayesian Optimization to get a decent approximation of the global maximum . Still , it keeps getting closer and closer to the global maximum in every iteration .

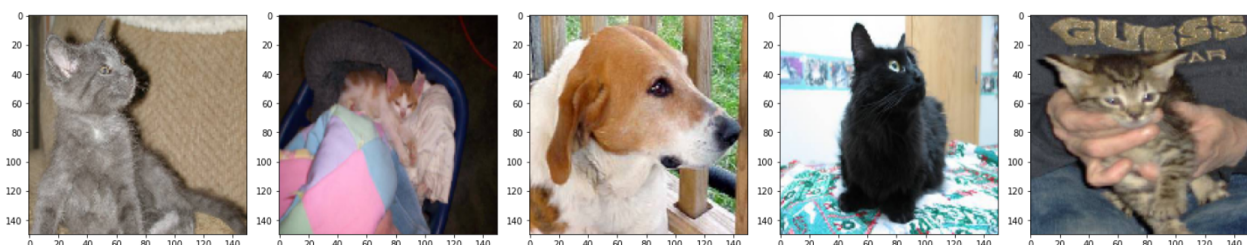
Our approximation of global maximum derived from Bayesian Optimization is 4.023 , while the true optima is 4.039 . I believe that our approximation is quite satisfying since we basically know nothing about the function in our initial state . Not to mention that the observation points we get has some noises added to it .

From this example , we could extrapolate that as long as we hold the big picture of our objective function , Bayesian Optimization could be a powerful tool finding a decent set of hyperparameters . Last but not least , though the scheme above can ameliorate the reliability of finding global maximum by Bayesian Optimization , I believe there is still room for improvement of solving this problem .

In conclusion , when we're facing high dimensional problems which is often costly to probe a observation point , I believe through the scheme similar to grid search combined with Bayesian Optimization , we should yield a relatively satisfying result whether it is global maximum or not.

KNN on classifying picture of dogs and cats with Bayesian Optimization

In this section , we will see how Bayesian Optimization works with KNN . There are some interesting behavior carried out by Bayesian Optimization and we would get into it below this paragraph .



In this implementation of Bayesian Optimization , I adopted the dataset from 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip' and the dataset would be split into two sections - training dataset and validation dataset . I utilize Keras's Sequential Model to accomplish this task . The structure of my model is shown in the python code below .

```
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

```

At this point , the " NUM_DENSE " and "dropout_rate" weren't determined yet . Thus , these two parameters would be tuned by Bayesian Optimization and the learning rate of the model would be automatically tuned as well .

Next , I implemented Bayesian Optimization on my model with a view to getting the least validation loss to prevent my model from getting overfitting . (I've tried to use Bayesian Optimization to find the set of hyperparameters which optimize the value of validation accuracy ; however , though it did quite well on the task , the problem of overfitting the training set remained unsolved in that case .)

```

pbounds={'NUM_DENSE':(0,7), 'lr':(1e-2,1e-4), 'dropout_rate':(0.2,0.9)}

```

Above is the domain which Bayesian Optimization would dive in . I recorded every hyperparameter set Bayesian Optimization probed and constructed the table below .

(In this example , the kernel of the gaussian process is Matern 3/2 and the acquisition function adopted is EI .)

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

Generally , machine learning or deep learning models often try to strike a balance between generalization and overfitting . In addition , it's a common way for a machine learning programmer to apply dense layers in their models with the aim of making their models capable of fitting high dimensional problems . Nevertheless , improper usage of dense layers often leads to overfitting . Moreover , when machine learning programmers confronting a brand new problem , it is technically difficult for one to realize what's the proper numbers of dense layer to the problem . In this example , we can bring to light that Bayesian Optimization may come in hand when we are facing such catastrophes .

From the table shown above , I think there are some mesmerizing behaviors of Bayesian Optimization we could notice .

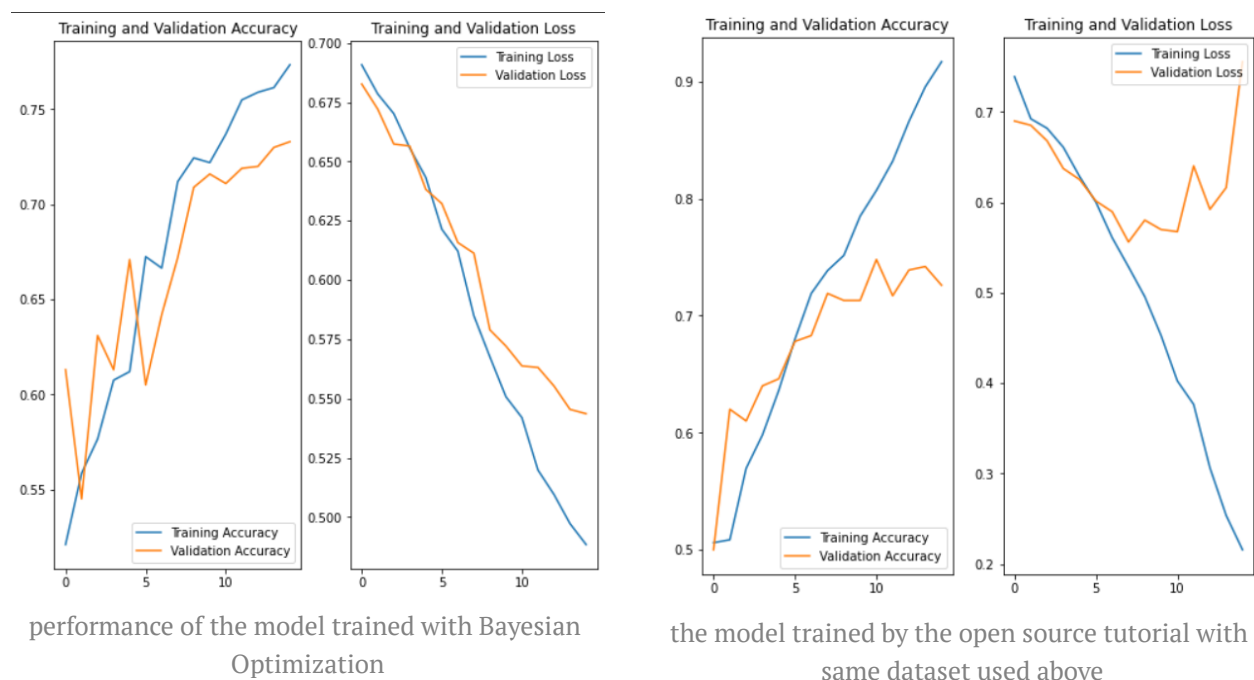
After a few iterations , we could discover that Bayesian Optimization probed several set of hyperparameters with the attribute "NUM_DENSE" being zero . It's not hard for us to

envision that without dense layer , the condition of overfitting the training dataset could be improved , so it's the right thing for the Bayesian Optimization to do .

Nonetheless , from iteration 9 to iteration 13 , there were some captivating things on foot . Rather getting stuck on applying zero dense layer , Bayesian Optimization tried numbers of dense layers with high dropout rate as its hyperparameter . I think it is a kind of strategy similar to those programmers might use . We want our models to own the potential of describing complex functions while we would exert regularizations on our models . Thus , it is pretty exciting to perceive that Bayesian Optimization do make wise decisions . In conclusion , when we don't know where to start on training a complex neural networks , maybe Bayesian Optimization could illuminate our way up or even accomplish our goal subtly .

There is still one thing worth mentioning . We could spot that the learning rate hasn't changed since iteration 4 . To know whether Bayesian Optimization is functioning normally , I changed the bound of learning rate to see whether Bayesian Optimization would probe on a different learning rate . Surprisingly , Bayesian Optimization take the smallest value in the bound of learning rate . After a few attempts , I notice that it's quite reasonable for Bayesian Optimization to pick the smallest number it could use as learning rate . Because from the long term , the smaller the value of learning rate is , the more possible our model could get close to global optimum due to the property of gradient descent . However , it might not converge well since our number of iteration is limited , so we need to be cautious when we are tuning learning rate through Bayesian Optimization .

In the end of this example , I want to compare the model taking advantage of Bayesian Optimization with the model trained with the same dataset on the open source .



As the line chart shows , we could see that Bayesian Optimization meet our expectations on resolving the problem of overfitting ; yet , it is a pity that the dataset aren't good enough to increase our accuracy . To sum up , I think the model collaborating Bayesian Optimization do surpass the performance of the general model .

Epilogue

After going through those details of Bayesian Optimization , I hope that the one who read this article could feel the essence of it and enjoy the fascinating probabilistic properties behind it . We've spot how powerful Bayesian Optimization is and also recognize some of the flaws of Bayesian Optimization . I hope that in the future , the open source community can construct more practical tools with Bayesian Optimization and I would spare no effort to modify or make use of Bayesian Optimization . Thanks to this opportunity , I have gained an insight into deep learning industry and at the same time sharpen my python programming skills . It would be the end of the article . We appreciate your time spending on reading through this article .

References

- [1] Drew Bagnell & Stephane Ross (2009) Statistical Techniques in Robotics (16-831, F09)Gaussian Process - Part 2.
Retrieved from https://www.cs.cmu.edu/~16831-f14/notes/F09/lec21/16831_lecture21.sross.pdf
- [2] Jie Wang (2021) An Intuitive Tutorial to Gaussian Processes Regression
Retrieved from <https://arxiv.org/pdf/2009.10862.pdf>
- [3] Jochen Görtler , Rebecca Kehlbeck & Oliver Deussen (2019) A Visual Exploration of Gaussian Processes
Retrieved from <https://distill.pub/2019/visual-exploration-gaussian-processes/>
- [4] Jason Brownlee (2019) How to Implement Bayesian Optimization from Scratch in Python.
Retrieved from <https://machinelearningmastery.com/what-is-bayesian-optimization/>
- [5] Chengwei (2019) How to do Hyper-parameters search with Bayesian optimization for Keras model
Retrieved from <https://www.dlology.com/blog/how-to-do-hyperparameter-search-with-baysian-optimization-for-keras-model/>
- [6] Jason Brownlee (2019) Your First Deep Learning Project in Python with Keras Step-By-Step
Retrieved from <https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>
- [7] Apoorv Agnihotri & Nipun Batra (2020) Exploring Bayesian Optimization
Retrieved from <https://distill.pub/2020/bayesian-optimization/>
- [8] Fernando Nogueira (2014) Pure Python implementation of bayesian global optimization with gaussian processes.
Retrieved from <https://github.com/fmfn/BayesianOptimization>
- [9] Peter I. Frazier (2018) A Tutorial on Bayesian Optimization
Retrieved from <https://arxiv.org/abs/1807.02811>
- [10] Richard Wilkinson (2019) Introduction to GPs
Retrieved from <https://rich-d-wilkinson.github.io/talks.html>