Elementary Hyperparameter Tuning

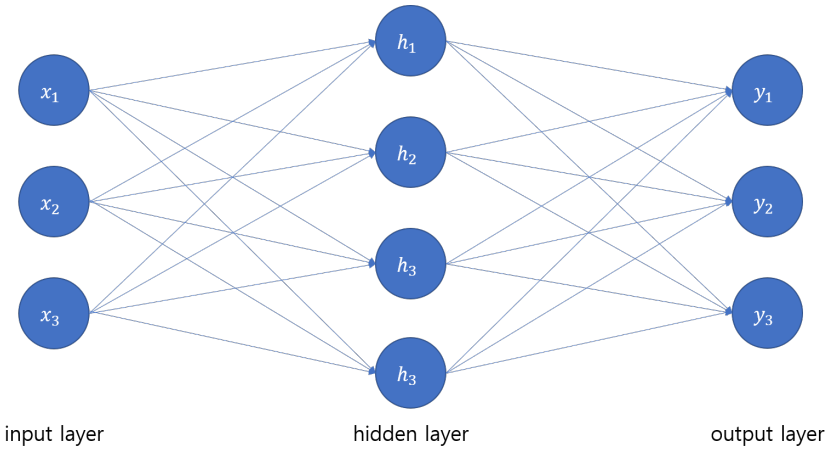
김선중

August 24, 2021

Table of Contents

Parameters

Consider a simple neural network, representing a function $f : \mathbb{R}^3 \to \mathbb{R}^3$.



With two matrices $W \in \mathbb{R}^{3\times4}$ and $W' \in \mathbb{R}^{4\times3}$, two vectors $b \in \mathbb{R}^4$ and $b' \in \mathbb{R}^3$ and the sigmoid function $\sigma(t) = (1 + e^{-t})^{-1}$, we have

$$f(x) = \sigma(W'\sigma(Wx + b) + b'), \tag{1}$$

where σ is evaluated elementwisely.

Parameters

For a dataset $D = \{(x^n, y^n) : x^n, y^n \in \mathbb{R}^3, n = 1, \cdots, N\}$, the cost can be evaluated as

$$C = \sum_{n=1}^{N} ||f(x^n) - y^n||_2{}^2. \tag{2}$$

Denote

$$\begin{aligned}\Theta &= (W, b, W', b') \\ &= (w_{11}, \cdots, w_{34}, b_1, b_2, b_3, b_4, w'_{11}, \cdots, w'_{43}, b'_1, b'_2, b'_3).\end{aligned}$$

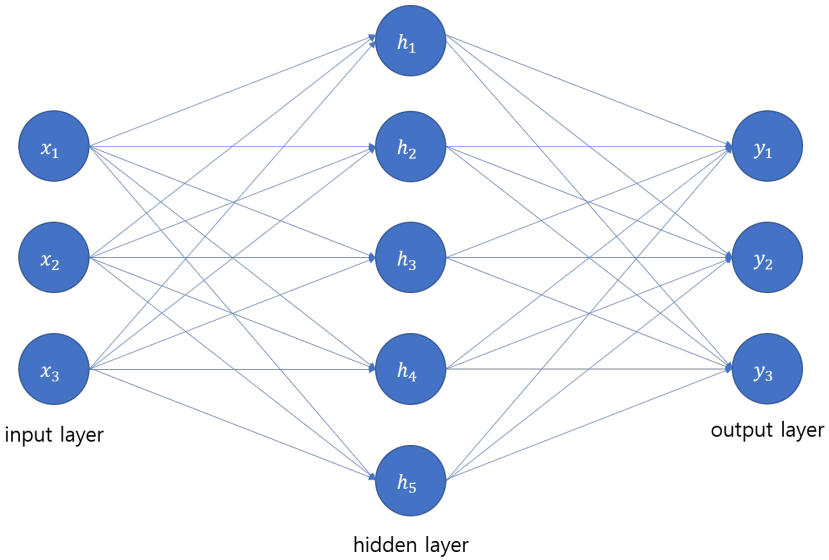
Since f depends on Θ, we may think of $C : \mathbb{R}^{31} \to \mathbb{R}$ as a function of Θ.

Parameters

- Using the gradient descent or the likes, we can find an approximated minimum of $C(\Theta)$.
- The optimal f^* is then obtained from the the minimum argument Θ^*.
- Components w_{ij}, b_k, w'_{ij}, b'_k of Θ are called the learnable parameters, or simply the parameters.

Hyperparameters

- We may change the structure of the network slightly to produce a different function f.
- Specifically, we can use different numbers of nodes in the hidden layer, say, 5, to enhance the performance.
- The number of hidden nodes is an example of hyperparameter.



Hyperparameters

- Beside the number of hidden nodes, the learning rate and the type of optimizers is also a hyperparameter.
- That is, we may think of an operator F with its arguements ϕ_1(the learning rate), ϕ_2(the number of hidden nodes) and ϕ_3 (the type of optimizer) where

$$F[\phi_1, \phi_2, \phi_3] = f,$$

Hyperparameters

- Suppose that each hyperparameters ϕ_i has its value in a set ;

$$\phi_1 \in [0.001, 0.01]$$
$$\phi_2 \in \{1, 2, 3, 4, 5, 6\}$$
$$\phi_3 \in \{\text{SGD}, \text{Adam}, \text{RMSprop}\}$$

- We may think of ϕ_i as random variables.
- The sample spaces of ϕ_i's are

$$S_1 = [0.001, 0.01]$$
$$S_2 = \{1, 2, 3, 4, 5, 6\}$$
$$S_3 = \{\text{SGD}, \text{Adam}, \text{RMSprop}\}$$

Hyperparameters

- For each $\Phi = (\phi_1, \phi_2, \phi_3)$, there corresponds a real value $C(\Theta^*)$.
- We are to find the optimal hyperparameters $\Phi = (\phi_1, \phi_2, \phi_3)$ that minimizes $C(\Theta^*)$.

The task of determining the set of hyperparameter to minimize the cost, given a dataset and the architecture of the model, is called the hyperparameter tuning, hyperparameter optimization, or hyperparameter selection.

Hyperparameters

Beside

- the learning rate
- the number of nodes in the hidden layers
- the type of optimizer,

the followings are also possible hyperparameters ;

- the size of minibatch
- the number of epochs
- the type of activation function
- the type of weight initialization
- the dropout parameter
- the regularization parameter

Manual Tuning

- As the most primitive and natural method, we may tune the hyperparameters by hands.
- We move from one point in the hyperparameter space to the other, each time evaluating the cost.
- There is no general rule for manual tuning.

$$(\phi_1, \phi_2, \phi_3) \longrightarrow (\phi_1 + \epsilon_1, \phi_2, \phi_3) \longrightarrow ({\phi_1}^*, \phi_2, \phi_3)$$
$$\longrightarrow ({\phi_1}^*, \phi_2 + \epsilon_2, \phi_3) \longrightarrow ({\phi_1}^*, {\phi_2}^*, \phi_3) \longrightarrow \cdots$$

Manual Tuning

Advanatges

- We can learn the behavior of hyperparameters by heart.

Disadvantages

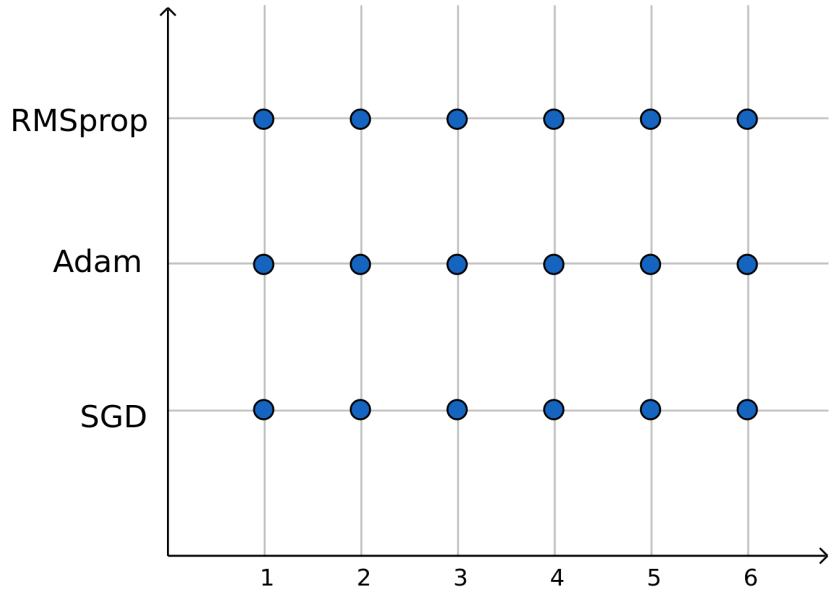
- Manual works are required.
- There is no general rule.

Grid Search

- Consider tuning ϕ_2 and ϕ_3 while ϕ_1 is fixed.

$$\phi_2 \in S_2 = \{1, 2, 3, 4, 5, 6\}$$
$$\phi_3 \in S_3 = \{\text{SGD}, \text{Adam}, \text{RMSprop}\}$$

- There are $|S_2| \times |S_3| = 6 \times 3 = 18$ possibilities that (ϕ_2, ϕ_3) can attain.



Grid Search

- Consider tuning ϕ_1, ϕ_2, ϕ_3 simultaneously.
- Since ϕ_1 lies in a (continuous) closed interval $S_1 = [0.001, 0.01]$ we need to discretize it ;

$$\phi_1 \in \{0.001, 0.005, 0.01\}.$$

- There are $3 \times 6 \times 3 = 54$ cases ;
  i.e. 54 points on the hyperparameter space $S_1 \times S_2 \times S_3$.
- For each cases, we train the model and test the performance independently.
- By comparing all the cost we get, we find the set of optimal hyperparameters and the corresponding cost value.

Grid Search

Advanatges

- We can cover all possible prospective sets of hyperparameters.

Disadvantages

- It may take too long ; the running time increase exponentially as the number of types of hyperparameter increases.
- There exist holes between grids.

Grid Search : Code

1. Grid Search for Classification

In [1]:

```
%%time
# grid search logistic regression model on the sonar dataset
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold, GridSearchCV
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/sonar.csv'
dataframe = read_csv(url, header=None)
# split into input and output elements
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# define model
model = LogisticRegression()
# define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define search space
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
# define search
search = GridSearchCV(model, space, scoring='accuracy', n_jobs=-1, cv=cv)
# execute search
result = search.fit(X, y)
# summarize result
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.7828571428571429
Best Hyperparameters: {'C': 1, 'penalty': 'l2', 'solver': 'newton-cg'}
Wall time: 29.1 s
```

Random Search

- Think of the hyperparameters ϕ_i as random variables with adequate distribution.
- Sample from the joint distribution of $\Phi = (\phi_1, \phi_2, \phi_3)$, build a model and test the performance.
- We can repeat this procedure multiple times and select the best one as the optimal set of hyperparameters.

Random Search

Advanatges

- Sampling can find unexpected points in the hyperparameter space with good performance.
- To get more accurate model, we simply increase the number of sampling.

Disadvantages

- If the hyperparameter space is big, then some hyperparameters might not be explored.
- We must specify the probability distribution of the sample space in advance.

Random Search : Code

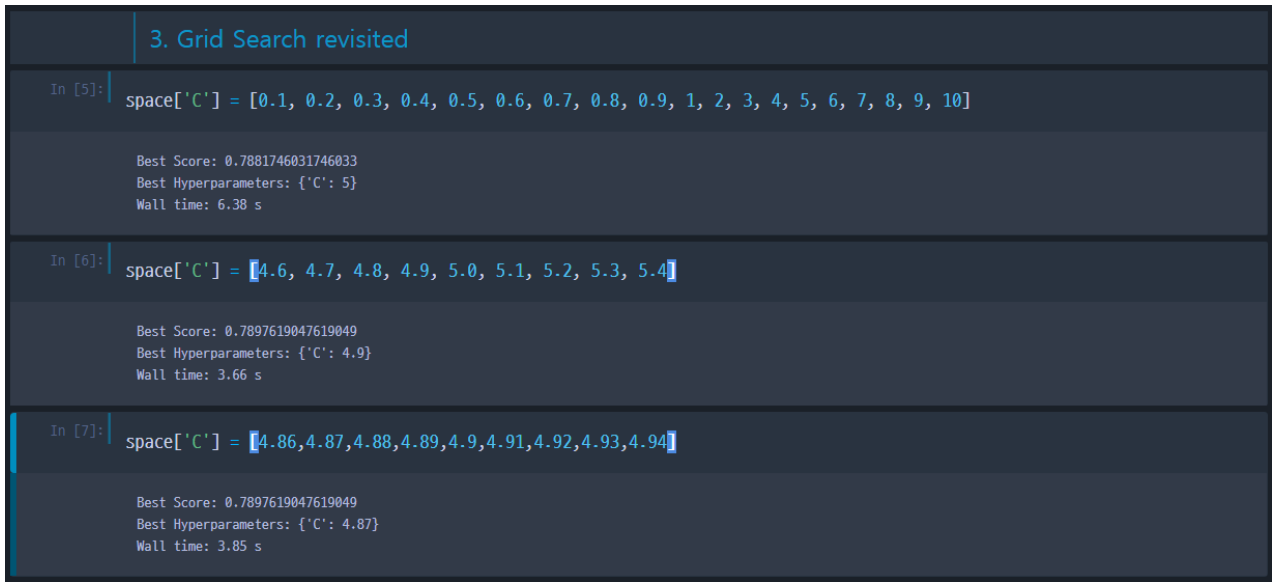
2. Random Search for Classification

In [2]:

```
%%time
# random search logistic regression model on the sonar dataset
from scipy.stats import loguniform
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold, RandomizedSearchCV
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/sonar.csv'
dataframe = read_csv(url, header=None)
# split into input and output elements
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# define model
model = LogisticRegression()
# define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define search space
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)
# define search
search = RandomizedSearchCV(model, space, n_iter=500, scoring='accuracy', n_jobs=-1, cv=cv, random_state=1)
# execute search
result = search.fit(X, y)
# summarize result
print('Best Score: %s' % result.best_score_)
print('Best Hyperparameters: %s' % result.best_params_)
```

```
Best Score: 0.7897619047619049
Best Hyperparameters: {'C': 4.878363034905756, 'penalty': 'l2', 'solver': 'newton-cg'}
```

Grid Search vs Random Search



- Grid method takes less time (29.1s, 6.38s, 3.66s, 3.85s) than random search(1min 21s).
- But random search requires just one step to implement and we don't have to set the grid carefully like we did in grid method.
- Morever, it produces quite an accurate result, compared to the random search.

Bayesian Optimization

- A drawback of random search : we must specify the probability distribution of the sample space in advance.
- In bayesian optimization, the probability distribution evolves from one to another.
- Specifically, given a prior distribution $P(f)$, we get the posterior distribution $(f|D)$ iteratively.
- Bayes Rule :

$$P(f|D) \propto P(D|f)P(f).$$

- More detailed explanation and the code will be covered in the final presentaion.

An Overview of Bayesian Optimizaiton

- Bayesian optimization is a method for maximizing (or minimizing) a function $f : A \to \mathbb{R}$ where $A \subset \mathbb{R}^d$. Sometimes we write

$$\max_{x \in A} f(x).$$

- The objective function f is unknown, we are to sample from the domain A using surrogate function and acquisition function to achieve our goal.

An Overview of Bayesian Optimizaiton

In our problem, we are assuming the followings;

- d, the dimension of the domain, is assumed to be less than 20.
- x, the independent variable, is assumed to be within a simple domain A like k-cube $\prod_{i=1}^{k}[a_i, b_i]$.
- f is assumed to be a good function in a sense that it is (Lipschitz) continuous. But it is not differentiable. So it is impossible to approximate f using its derivatives or its second derivatives.
- Still, f is really complicated function, so one can not evaluate the value in low cost. And it lacks special structure like concavity or linearity.

Overview of Bayesian Optimization

- It is called *Bayesian* because it uses the famous "Bayes theorem"
$$P(M|E) \propto P(E|M)P(M).$$
- The posterior probability of a model M given evidence E is proportional to the likelihood of E given M, multiplied by the prior probability of M.
- Let x_i be the ith sample. Consider the set
$$D_{1:t} = \{(x_i, f(x_i)) \,|1 \leq i \leq t\}$$
of ordered pair $(x_i, f(x_i)$, which plays a role of evidence. We have
$$P(f|D_{1:t}) \propto P(D_{1:t}|f)P(f).$$

Overview of Bayesian Optimization

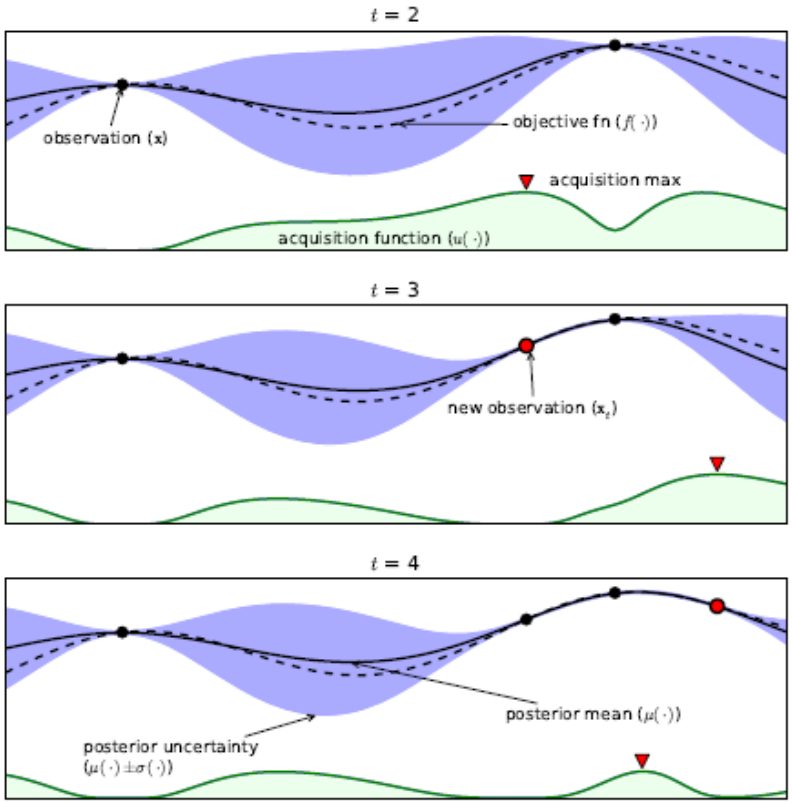


Figure 1: *An example of using Bayesian optimization on a toy 1D design problem. The figures show a Gaussian process (GP) approximation of the objective function over four iterations of sampled values of the objective function. The figure also shows the acquisition function in the lower shaded plots. The acquisition is high where the GP predicts a high objective (exploitation) and where the prediction uncertainty is high (exploration)—areas with both attributes are sampled first. Note that the area on the far left remains unsampled, as while it has high uncertainty, it is (correctly) predicted*

References

1. James Bergestra, et al., 2011, “Algorithms for Hyper-Parameter Optimzation”
2. Jasper Snoek, et al., 2012, “Practical Bayesian Optimization of Machine Learning Algorithms”

Thank you