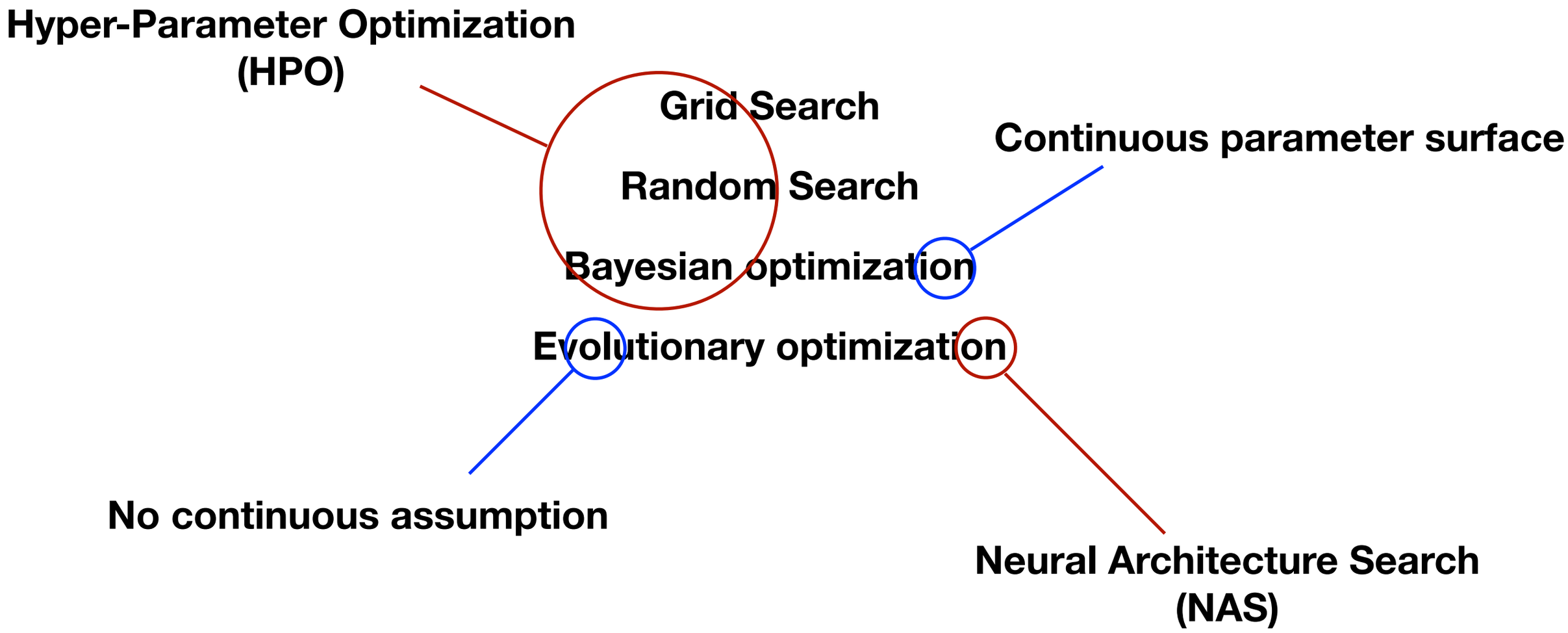
HPO

HPO and NAS

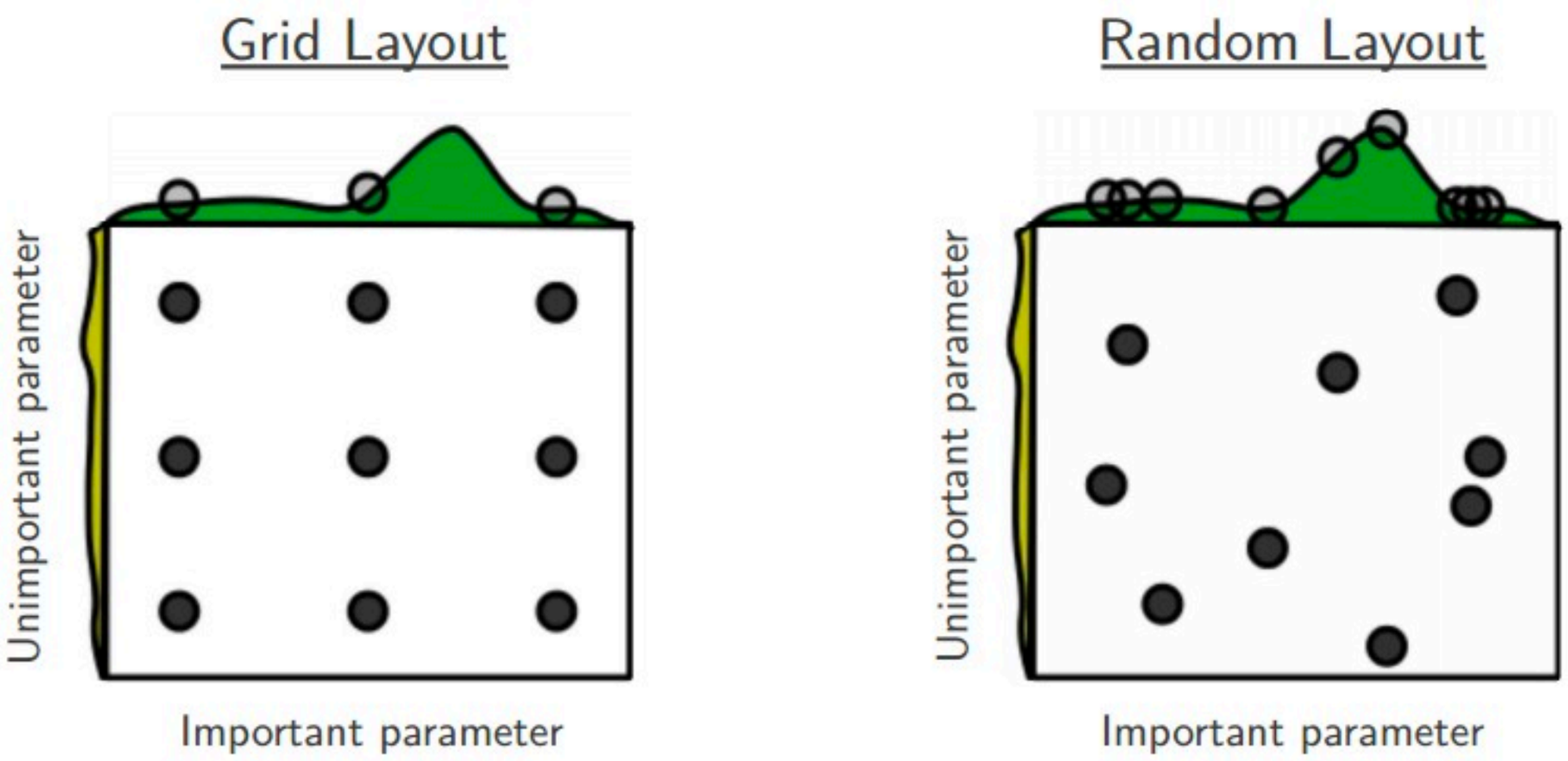

Hyper-Parameter Optimization
(HPO)
Grid Search
Random Search
Bayesian optimization
Evolutionary optimization
No continuous assumption
Continuous parameter surface
Neural Architecture Search
(NAS)

Grid Search

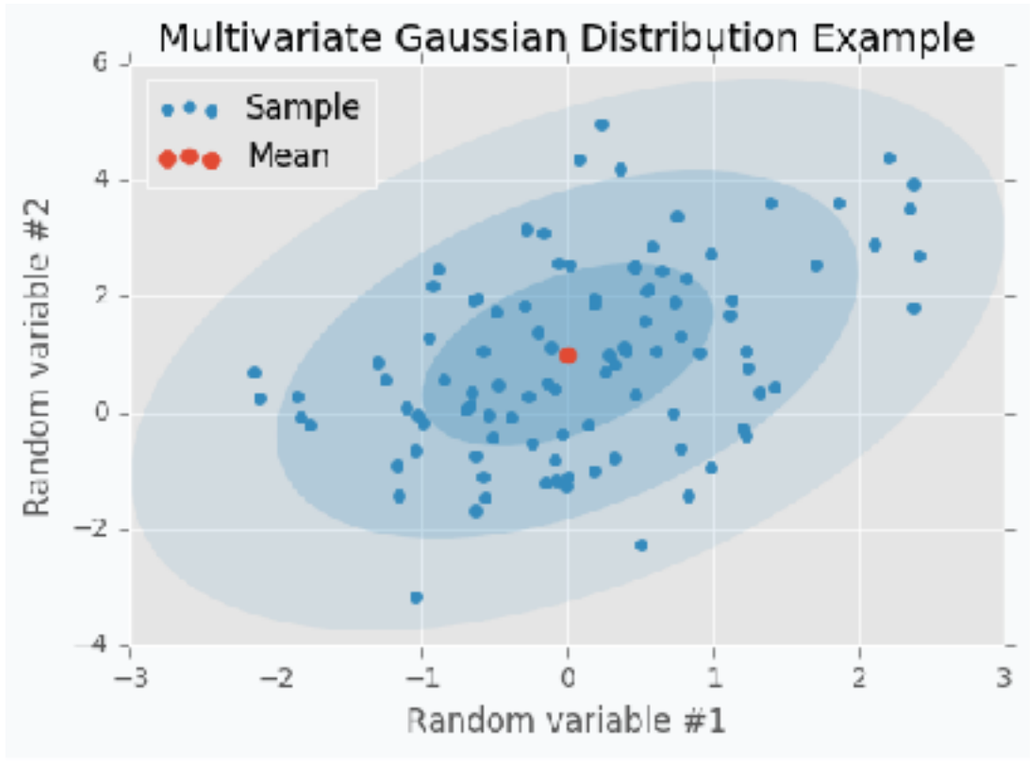
The simplest algorithms that you can use for hyperparameter optimization is a Grid Search. The idea is simple and straightforward. You just need to define a set of parameter values, train model for all possible parameter combinations and select the best one.

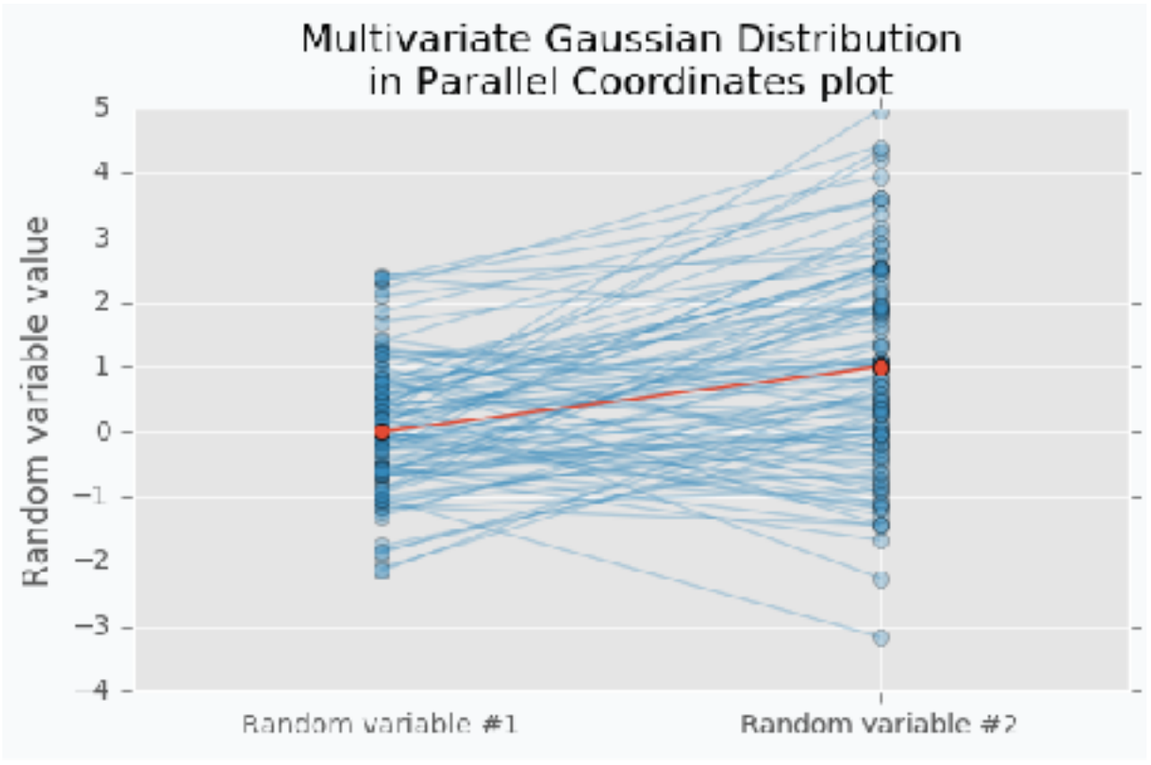
Random Search

The idea is similar to Grid Search, but instead of trying all possible combinations we will just use randomly selected subset of the parameters.

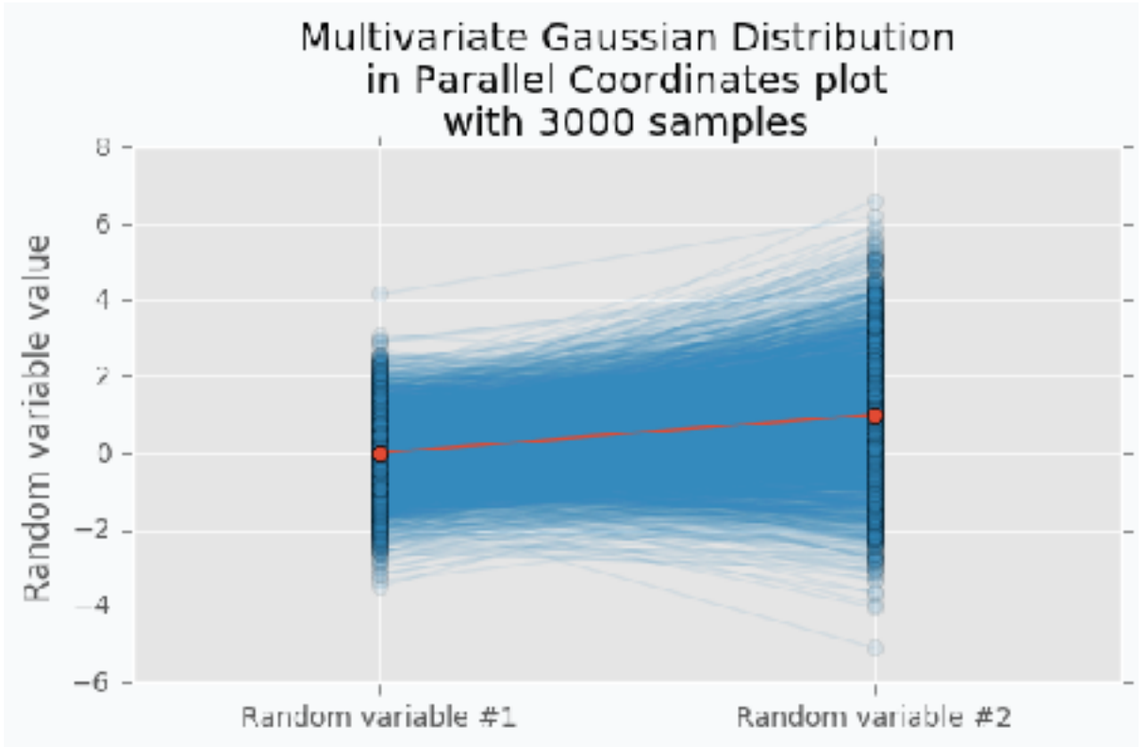


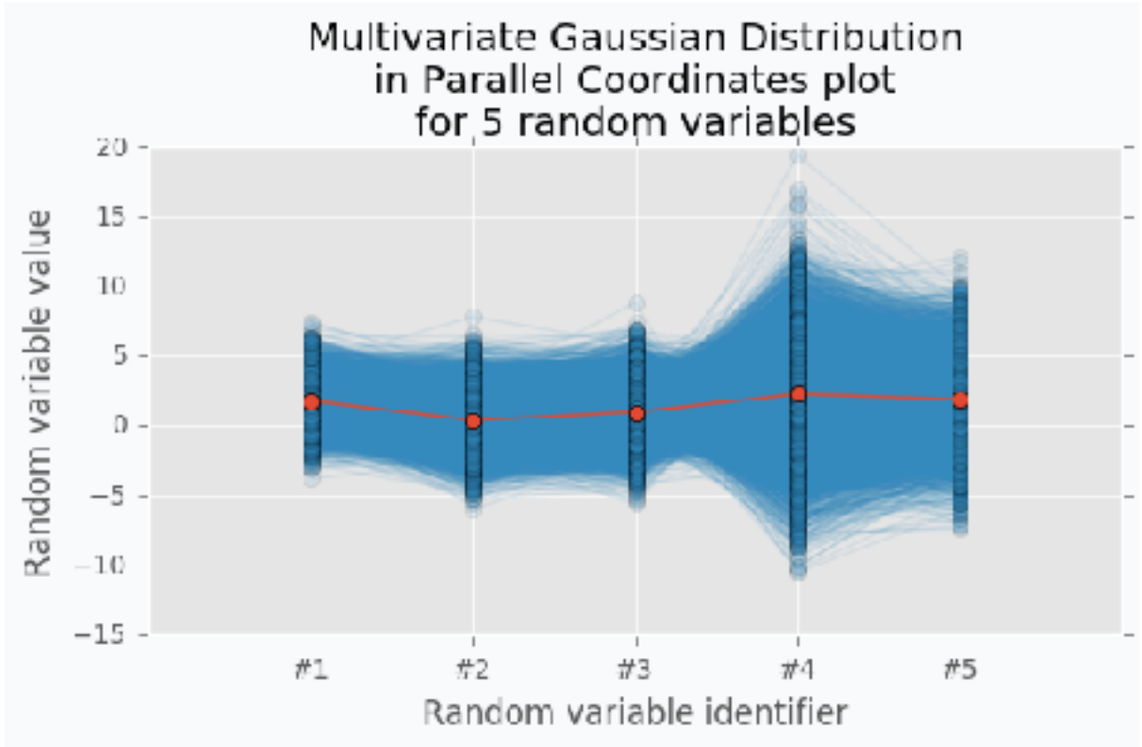
Gaussian Distribution





Gaussian Distribution





Gaussian Process

Theorem 4.3.1 (Marginals and conditionals of an MVN). *Suppose* $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ *is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} \tag{4.67}$$

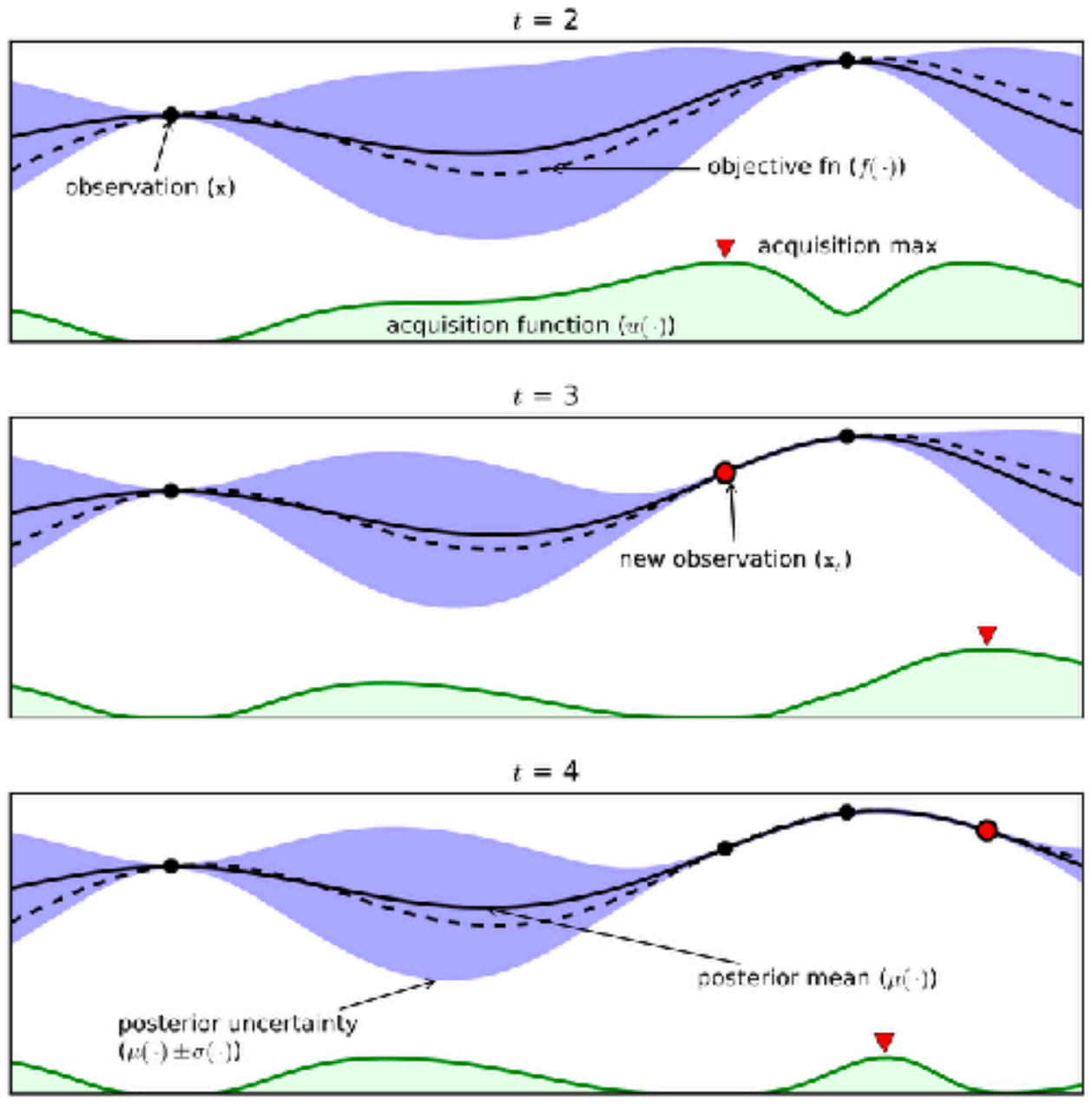
Then the marginals are given by

$$\begin{aligned} p(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ p(\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \end{aligned} \tag{4.68}$$

and the posterior conditional is given by

$$\begin{aligned} p(\mathbf{x}_1|\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1}\boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2}\left(\boldsymbol{\Lambda}_{11}\boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12}(\mathbf{x}_2 - \boldsymbol{\mu}_2)\right) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned} \tag{4.69}$$

Bayesian Optimization

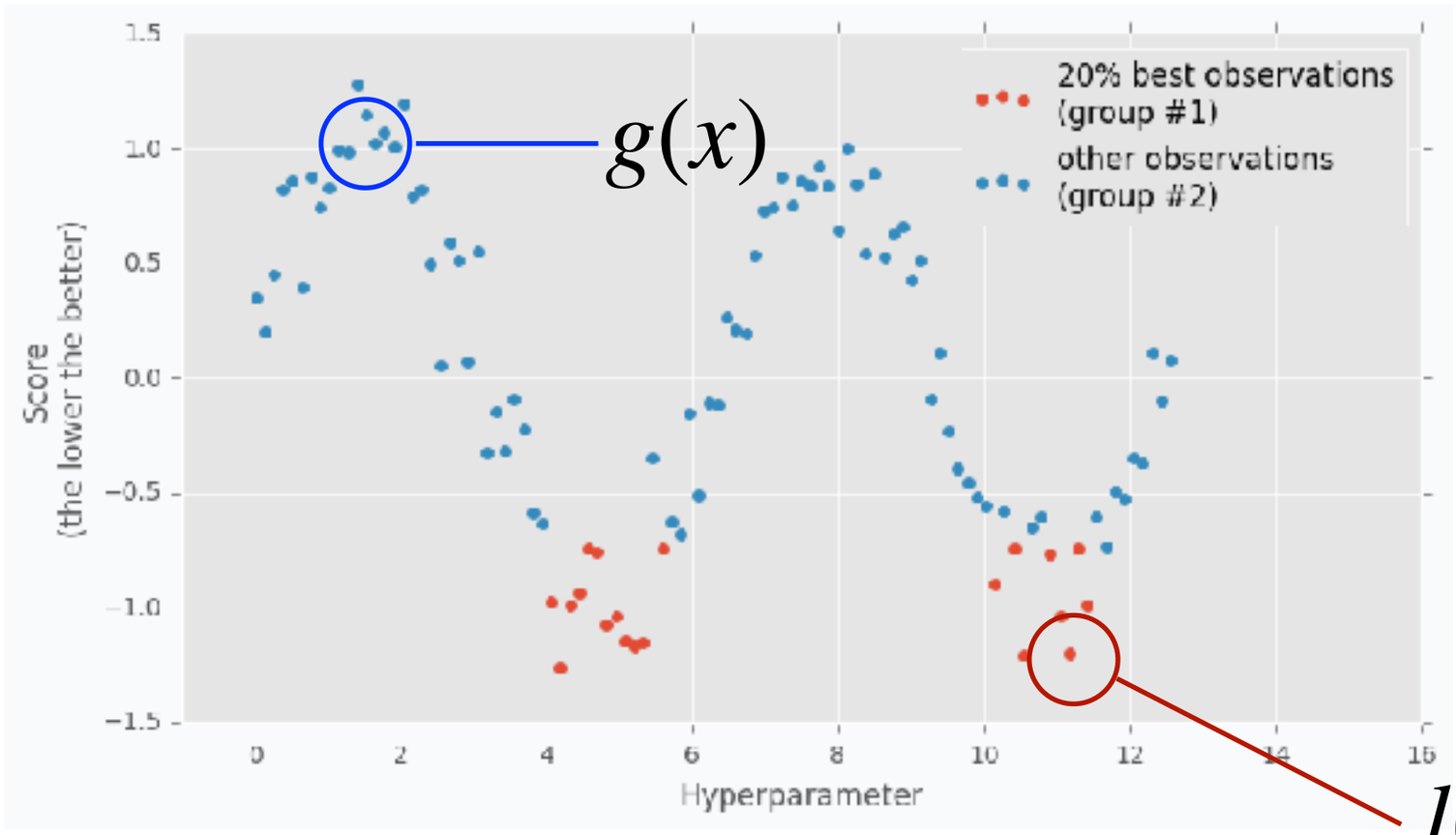


Bayesian Optimization - Disadvantage

1. It doesn't work well for **categorical variables**. In case if neural networks it can be a type of activation function.
2. GP with EI selects new set of parameters based on the best observation. Neural Network usually involves randomization (like weight initialization and dropout) during the training process which influences a final score. **Running neural network with the same parameters can lead to different scores**. Which means that our best score can be just lucky output for the specific set of parameters.
3. It can be **difficult to select right hyperparameters for Gaussian Process.** Gaussian Process has lots of different kernel types. In addition you can construct more complicated kernels using simple kernels as a building block.
4. It **works slower when number of hyperparameters increases.** That's an issue when you deal with a huge number of parameters.

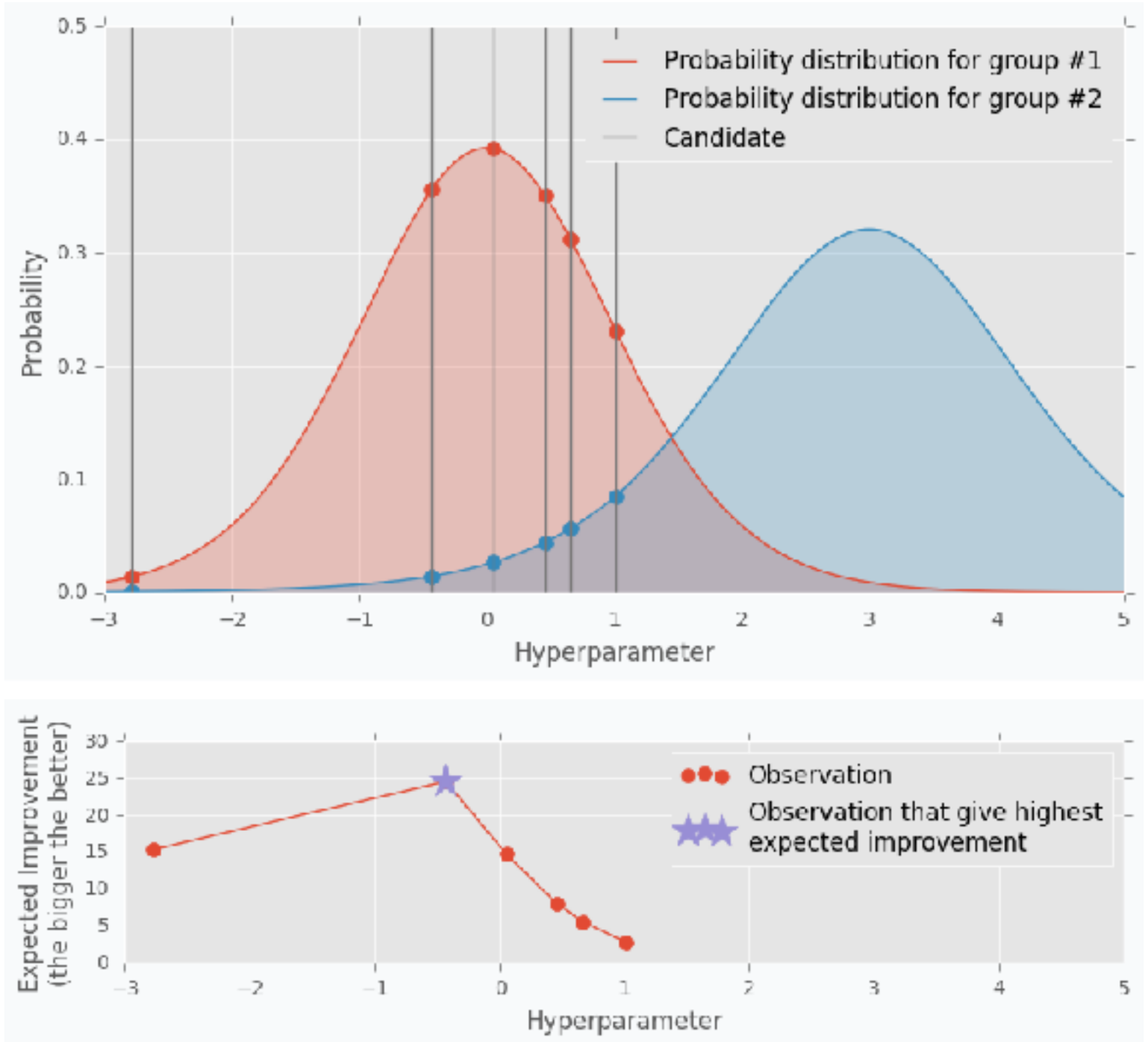
Tree-structured Parzen Estimator

Tree-structured Parzen Estimators (TPE) fixes disadvantages of the Gaussian Process. Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next. The main idea is similar, but an algorithm is completely different



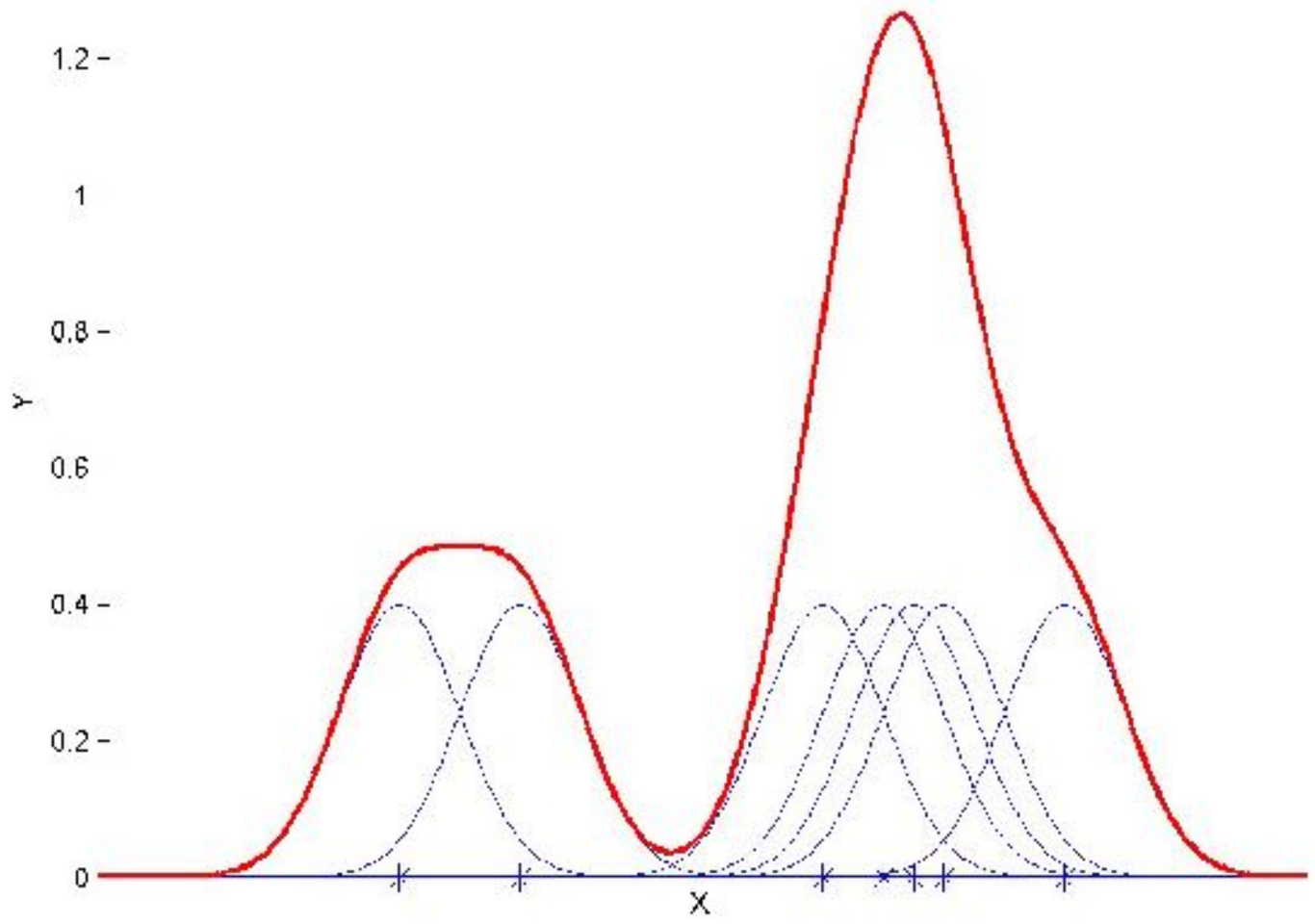
$$EI(x) = \frac{l(x)}{g(x)}$$

Tree-structured Parzen Estimator



Parzen Window Estimator

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$



Framework

```
%reset -f

from hyperopt import fmin, tpe, hp
import matplotlib.pyplot as plt

def f(x):
    return x**2 - x + 1

plt.plot(range(-5, 5), [f(x) for x in range(-5, 5)])
plt.title("Function to optimize: f(x) = x^2 - x + 1")
plt.show()

space = hp.uniform('x', -5, 5)

best = fmin(
    fn=f,  # "Loss" function to minimize
    space=space,  # Hyperparameter space
    algo=tpe.suggest,  # Tree-structured Parzen Estimator (TPE)
    max_evals=1000  # Perform 1000 trials
)

print("Found minimum after 1000 trials:")
print(best)
```

Framework

```
%reset -f

from hyperopt import fmin, tpe, hp
import matplotlib.pyplot as plt

def f(x):
    return x**2 - x + 1

plt.plot(range(-5, 5), [f(x) for x in range(-5, 5)])
plt.title("Function to optimize: f(x) = x^2 - x + 1")
plt.show()

space = hp.uniform('x', -5, 5)

best = fmin(
    fn=f,  # "Loss" function to minimize
    space=space,  # Hyperparameter space
    algo=tpe.suggest,  # Tree-structured Parzen Estimator (TPE)
    max_evals=1000  # Perform 1000 trials
)

print("Found minimum after 1000 trials:")
print(best)
```



```
Found minimum after 1000 trials:
{'x': 0.500084824485627}
```

Hand tuning

it is clear that **human can select parameters better than Grid search or Random search algorithms.** The main reason why is that **we are able to learn from our previous mistakes.** After each iteration, we memorize and analyze our previous results. This information gives us a much better way for selection of the next set of parameters. **And even more than that. The more you work with neural networks the better intuition you develop** for what and when to use.

- **작은 모형부터 시작**
- **Baseline 모형을 benchmark 하는 것이 좋음**
- **경험적으로는 Batch normalization 혹은 dropout을 넣으면 훨씬 안정적으로 학습**