Bayesian Search for Hyperparameters

Antonio Aguirre
University of California, Santa Cruz

Introduction to Hyperparameter Search

Machine learning models often have internal settings, known as **hyperparameters**, which significantly affect their performance. These hyperparameters, such as the **learning rate** (how quickly the model updates itself during training) or the **latent space size** (how compressed the model's internal representation of data is), are not learned from the data but must be set by the researcher.

One type of model that relies heavily on hyperparameter tuning is the **autoencoder**. An autoencoder is a neural network designed to learn compressed representations of data by encoding input into a smaller, latent space and then reconstructing it. This is especially useful in fields like medicine, where autoencoders can identify patterns in complex data, such as brain activity or genetic expressions, while preserving key information.

Tuning hyperparameters for autoencoders is crucial for balancing compression and reconstruction accuracy. Traditional methods like **Grid Search** and **Random Search** can help, but they are either exhaustive and slow or inefficient and unguided.

Why Bayesian Search?

Bayesian Search provides a smarter, probabilistic approach to hyperparameter tuning. It leverages the **Bayes Rule** to incorporate knowledge from previous trials, prioritizing promising hyperparameter combinations while avoiding less likely candidates. This not only speeds up the search but also helps find better solutions in complex, high-dimensional spaces.

We can think of the ayes Rule in terms of *Prior* and *Posterior* belief updates:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)},$$

where:

- $P(A)$: *Prior* probability (initial belief about A).
- $P(B \mid A)$: Likelihood of observing B given A is true.
- $P(B)$: Total probability of observing B.
- $P(A \mid B)$: *Posterior* probability (updated belief about A after observing B).

Steps in Bayesian Search

The following steps describe how Bayesian Search applies the Bayes Rule to hyperparameter tuning:

1. Define the Search Space

First, we define the **search space**, which consists of all possible hyperparameter combinations to explore (e.g., learning rates, layer sizes). Each combination represents a potential candidate for achieving the best model performance. This search space forms the foundation for the Bayesian search process.

2. Assign Prior Probabilities

Next, we assign initial probabilities, $P(\text{hyperparameters})$, to each combination. These probabilities represent our initial belief about how likely each combination is to perform well. If no prior knowledge is available, we can assign equal probabilities to all combinations (a uniform or Laplacian prior). This step ensures that every candidate starts with a fair chance.

3. Test Initial Combinations

We randomly sample a few hyperparameter combinations from the search space and train the autoencoder using these settings. The performance of the model under these combinations provides the **observed evidence** (performance observed so far), which serves as the data we condition on to refine our search.

4. Update Probabilities

Using **Bayes' Rule**, we update our belief about how likely a new set of hyperparameters will perform well. This update incorporates the information from the observed historical performance of previously tested combinations:

$$P(\text{hyperparameters} \mid \text{observed performance}) = \frac{P(\text{observed performance} \mid \text{hyperparameters}) \cdot P(\text{hyperpara}}{P(\text{observed performance})}$$

Here's a breakdown of the components:

- **Prior:** $P(\text{hyperparameters})$ reflects our initial belief about the likelihood of each combination.
- **Likelihood:** $P(\text{observed performance} \mid \text{hyperparameters})$ quantifies how well a specific combination explains the observed performance.
- **Posterior:** $P(\text{hyperparameters} \mid \text{observed performance})$ updates our belief about the likelihood of each combination, conditioned on the performance data gathered so far.

This updated posterior distribution helps us identify promising new hyperparameter candidates. The process ensures that each new evaluation leverages the information learned from prior tests, focusing the search on areas of the search space that are more likely to yield high-performing results.

5. Select Next Combination

Choose the next combination to test using an **acquisition function**, balancing:

- **Exploration:** Try under-tested combinations.
- **Exploitation:** Focus on high-posterior-probability combinations.

6. Iterate Until Convergence

Repeat testing and updating until the optimal combination is found or resources are exhausted.

Comparison with Other Methods

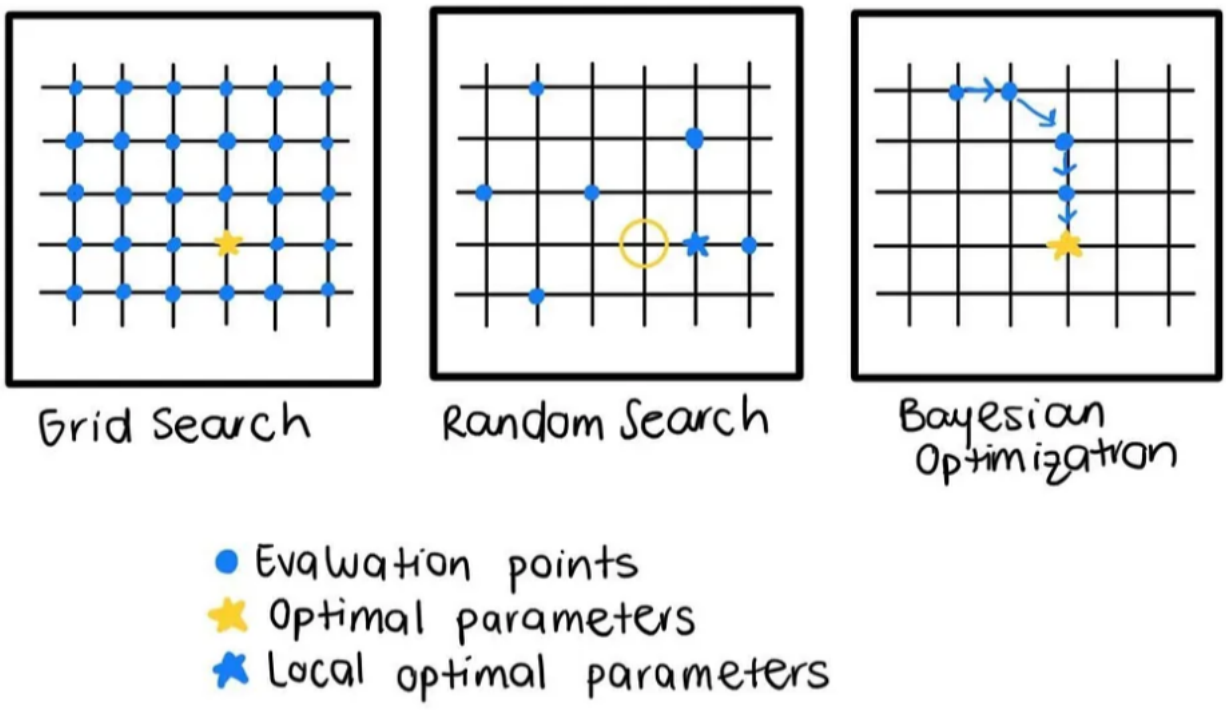

Figure 1: Comparison of the three main search strategies

.

Comparison of Search Methods

- **Grid Search:** Exhaustively tests all combinations. Computationally expensive and inefficient for large spaces.
- **Random Search:** Samples combinations randomly. More efficient but lacks direction.
- **Bayesian Search:** Uses probabilistic reasoning to focus on promising areas, balancing exploration and exploitation.

Takeaways

- Bayesian Search uses the **Bayes' Rule** to update probabilities of hyperparameter success based on performance.
- It avoids the inefficiency of Grid Search and the randomness of Random Search.
- By focusing on high-probability regions, it finds optimal hyperparameters efficiently.

Example: Bayesian Search for Autoencoders in Medical Imaging

Suppose we are training an autoencoder to compress and reconstruct MRI scans. The goal is to identify optimal hyperparameters for accurate reconstruction. We'll consider two hyperparameters:

- **Learning Rate** (α): Controls how much the model adjusts during training.
- **Latent Dimension Size** (d): The size of the compressed representation.

Step 1: Define the Search Space

The possible values for the hyperparameters are:

$$\alpha \in \{0.001, 0.01, 0.1\}, \quad d \in \{8, 16, 32\}.$$

This creates a search space of $3 \times 3 = 9$ combinations.

Step 2: Assign Prior Probabilities

We begin with no prior knowledge, assigning equal probabilities to each combination:

$$P(\alpha, d) = \frac{1}{9} \quad \text{for all combinations.}$$

Step 3: Test Initial Combinations

Randomly sample two combinations from the search space, train the autoencoder, and evaluate performance using a reconstruction error metric (e.g., Mean Squared Error, MSE). Suppose we test:

- $(\alpha = 0.01, d = 8)$ with an MSE of 0.15.
- $(\alpha = 0.1, d = 16)$ with an MSE of 0.20.

Step 4: Update Probabilities Using the Bayes Rule

To update the probabilities, we use the Bayes Rule :

$$P(\alpha, d \mid \text{performance}) = \frac{P(\text{performance} \mid \alpha, d) \cdot P(\alpha, d)}{P(\text{performance})}.$$

Likelihood Calculation

The likelihood $P(\text{performance} \mid \alpha, d)$ reflects the probability of observing the given MSE for each combination. Assigning a higher likelihood to combinations with lower errors:

$$P(\text{performance} \mid \alpha, d) \propto \frac{1}{\text{MSE}(\alpha, d)}.$$

This proportionality simplifies the computation, as the denominator $P(\text{performance})$ acts as a normalization factor, ensuring that all probabilities sum to 1.

Posterior Calculation

For $(\alpha = 0.01, d = 8)$:

$$P(\alpha = 0.01, d = 8 \mid \text{performance}) \propto \frac{1}{0.15} \cdot \frac{1}{9}.$$

For $(\alpha = 0.1, d = 16)$:

$$P(\alpha = 0.1, d = 16 \mid \text{performance}) \propto \frac{1}{0.20} \cdot \frac{1}{9}.$$

Normalize the results to compute the posterior probabilities. After normalization:

$$P(\alpha = 0.01, d = 8 \mid \text{performance}) = 0.6, \quad P(\alpha = 0.1, d = 16 \mid \text{performance}) = 0.4.$$

Step 5: Select Next Combination

Using the posterior probabilities, prioritize testing combinations that are most likely to perform well. For example, test $(\alpha = 0.01, d = 32)$, which has $P(\alpha, d) = \frac{1}{9}$ initially but is promising based on similar tested combinations.

Step 6: Iterate and Refine

Repeat the process: test new combinations, evaluate performance, and update probabilities. Over iterations, the search narrows to the most promising combinations, balancing exploration (trying diverse values) and exploitation (focusing on high-probability regions).

Advantages of Proportionality Notation

Proportionality notation ($\propto$) simplifies Bayesian Search update by avoiding the explicit calculation of $P(\text{performance})$. This term ensures probabilities sum to 1 but is not required when comparing relative probabilities. For exact probabilities, divide each posterior by the sum of all posteriors:

$$P(\alpha, d \mid \text{performance}) = \frac{P(\text{performance} \mid \alpha, d) \cdot P(\alpha, d)}{\sum_{\text{all combinations}} P(\text{performance} \mid \alpha, d) \cdot P(\alpha, d)}.$$