

Hyperparameter Optimization for Machine Learning

Shahid beheshti University

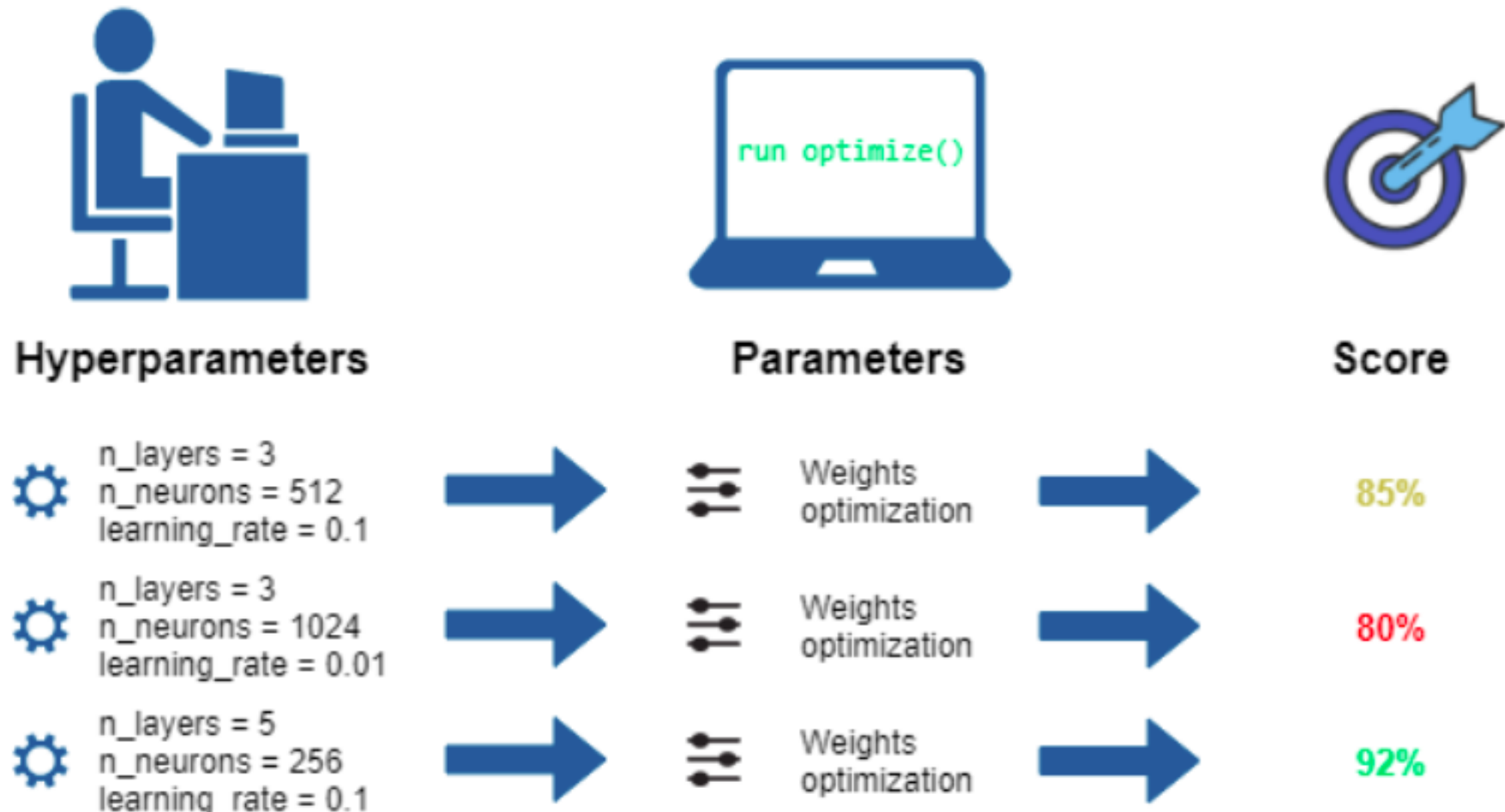


Zahra Taheri

za_taheri@sbu.ac.ir

May 17, 2022

Parameters VS hyperparameters



<https://www.kdnuggets.com/2020/02/practical-hyperparameter-optimization.html>

Examples

Model parameters:

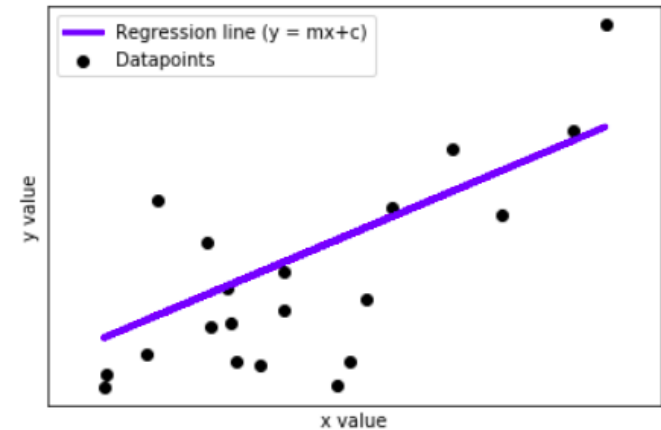
Learned by fitting model on training set

- m (slope) and c (intercept) in Linear Regression
- weights and biases in Neural Networks

Model hyperparameters:

- 1) Set by the user before training,
- 2) Not changed when fitting the model on data

- Learning rate in gradient descent
- Number of epochs
- Number of layers in a neural network
- Number of neurons per layer in a neural network
- Batch size
- Momentum parameter
- Number of clusters k in k-means clustering



Differences between parameter and hyperparameter?

Parameters	Hyperparameters
Required for making predictions	Required for estimating the model parameters
Not set manually	Set manually
The final parameters found after training will decide how the model will perform on unseen data	The choice of hyperparameters decide how efficient the training is. E. g., In gradient descent the learning rate decide how efficient and accurate the optimization process is in estimating the parameters

<https://www.geeksforgeeks.org/difference-between-model-parameters-vs-hyperparameters/>

Hyperparameter optimization (hyperparameter tuning)

- The **process of determining the right combination of hyperparameters** that maximizes the model performance(i. e., finding the set of hyperparameters leading to the lowest error on the validation set).
- It is an **important step** in any machine learning project since it **leads to optimal results for a model**.

In hyperparameter tuning technique,

- 1) We **build a model for every combination of various hyperparameters** and **evaluate each model**.
- 2) The model which gives the **highest performance wins**.

Different strategies for hyperparameter tuning


Manual tuning: Involves experimenting with different sets of hyperparameters manually i. e., each trial with a set of hyperparameters will be performed by you.



- You have **more control** over the process
- **Results are generally pretty good**



- There are **lots of efforts** and it can be **costly and time-consuming**.
- This **isn't very practical** when there are a lot of hyperparameters to consider.



Automated tuning: Utilizes already existing algorithms to automate the process of hyperparameter tuning.

Automated tuning steps:

- Specify the **hyperparameters space** (usually **as a dictionary**)
- **Run the algorithm on the specified space** to find the best set of hyperparameters that will give optimal results.

Overview of hyperparameter optimization methods

- **Grid Search**
- **Random Search**
- **Bayesian Optimization**
- **TPE Optimization**
- **HyperBand / ASHA (Bandits)**
- **Population-based Training**
-
-

Hyperparameter optimization methods

Grid search

- Define a grid of hyperparameters you want to try (**all the values are placed in the form of a matrix**)
- Each set of hyperparameters is taken into consideration and the obtained model performance is noted.
- Finally, it returns a set of hyperparameters with the best model performance.
- It is essentially a **brute force method**

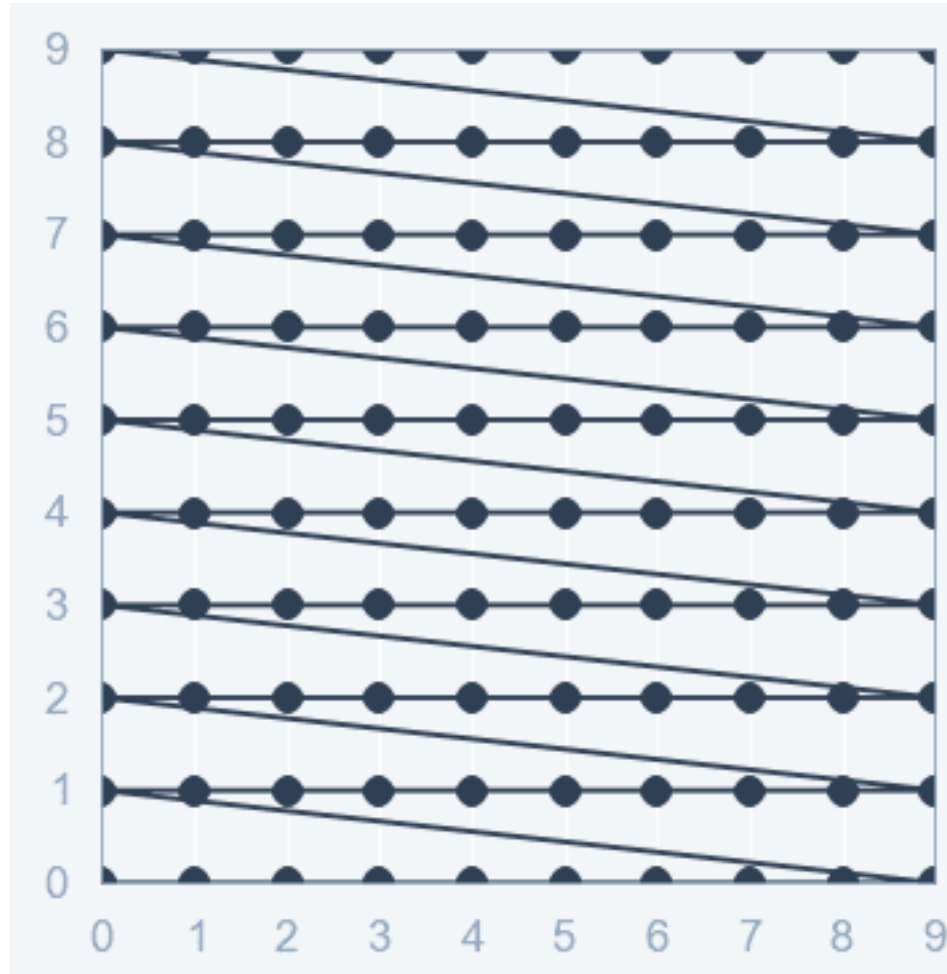


- It is **easy to implement**
- Can be **easily parallelized**



- It is insanely **computationally expensive** (as all brute force methods are)

A visual description of uniform search pattern of the grid search



Source

Random search

- Each iteration tries **a random combination of hyperparameters** values by running the system on this combination
- Finally, it returns a combination of hyperparameters with the best model performance.



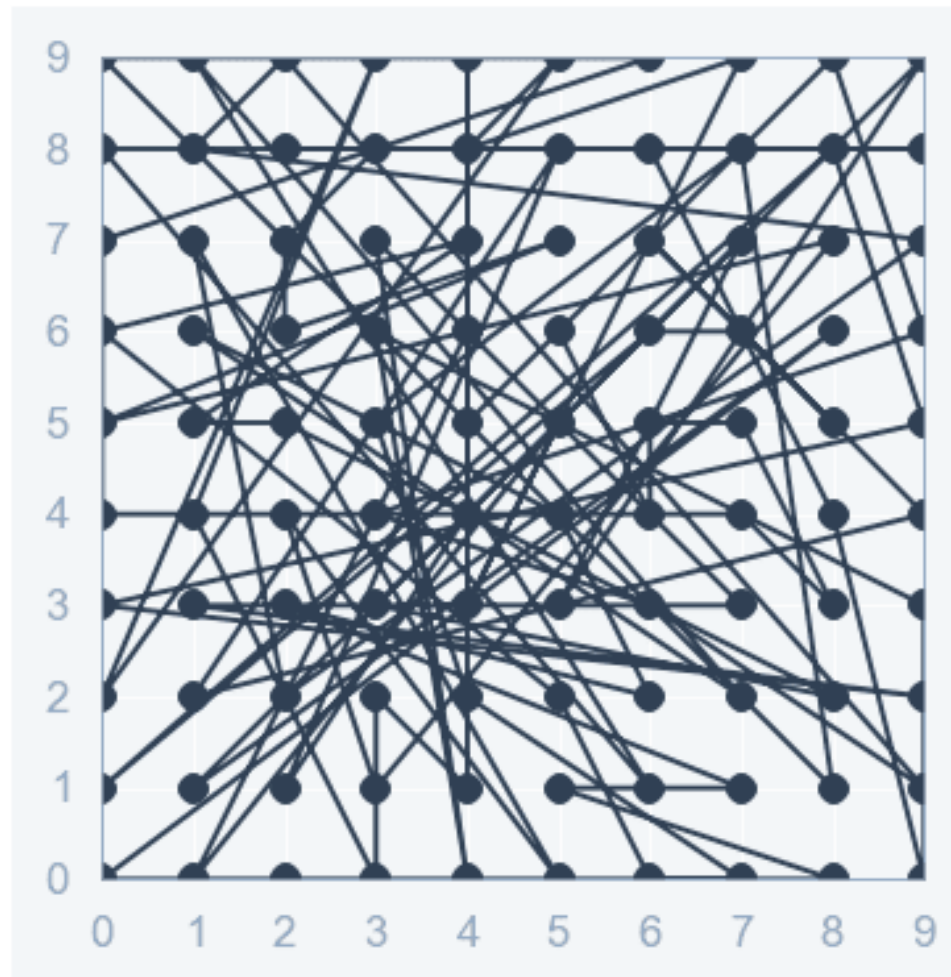
- Can be **easily parallelized**
- Probably works **best for lower dimensional data** since the time taken to find the right set is less with less number of iterations
- Works **good when the number of hyperparameters is low**
- It is similar to grid search, and yet it has proven to yield better results



- **Luck is involved** in this method
- There is **no guarantee of finding** a local minimum to some precision except
- It is **computationally expensive**

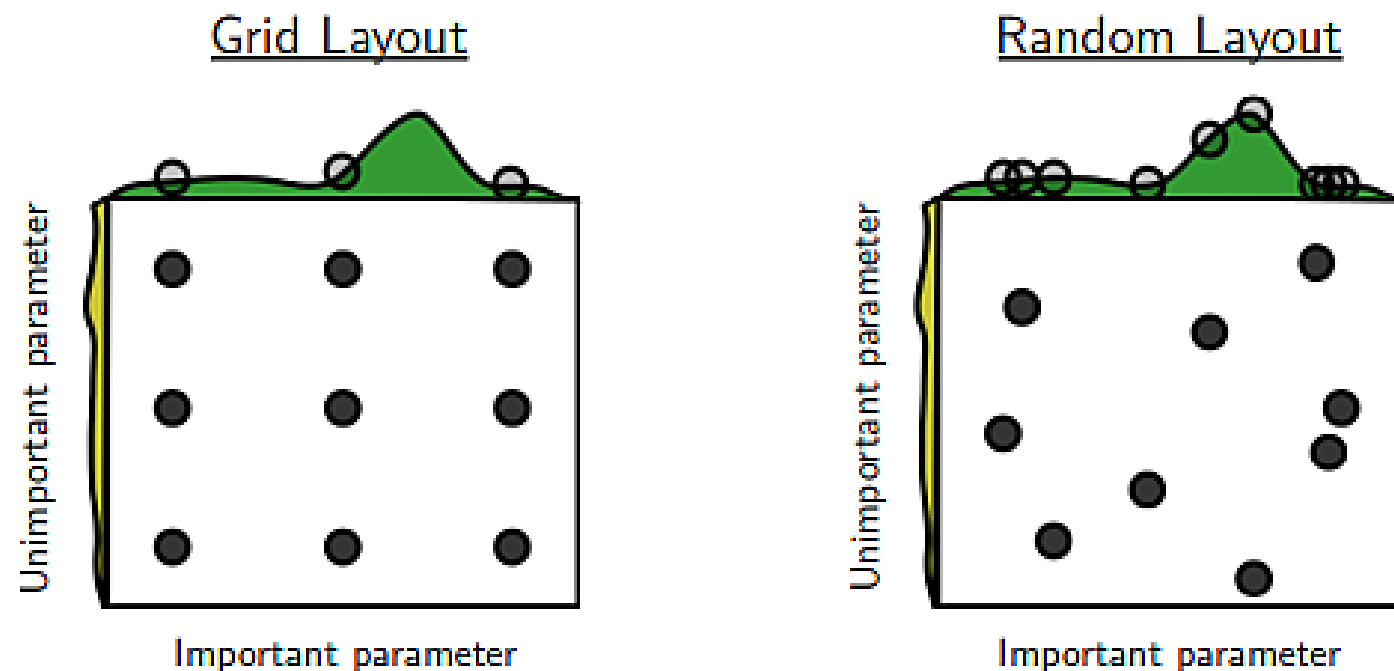
As random values are selected at each instance, it is highly likely that the whole action space has been reached because of the randomness, which takes a considerable amount of time to cover every aspect of the combination during grid search.

A visual description of search pattern of the random search



Source

Random search VS Grid search



Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g . This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bengio's and Bergstra's paper: Random Search for Hyper-Parameter Optimization

13 / 45

Random search

I love movies where the underdog wins, and I love machine learning papers where simple solutions are shown to be surprisingly effective. This is the storyline of "Random search for hyperparameter optimization" by Bergstra and Bengio. Random search is a slight variation on grid search. Instead of searching over the entire grid, random search only evaluates a random sample of points on the grid. This makes random search a lot cheaper than grid search.

Random search wasn't taken very seriously before. This is because it doesn't search over all the grid points, so it cannot possibly beat the optimum found by grid search. But then came along Bergstra and Bengio. They showed that, in surprisingly many instances, random search performs about as well as grid search. All in all, trying 60 random points sampled from the grid seems to be good enough.

In hindsight, there is a simple probabilistic explanation for the result: for any distribution over a sample space with a finite maximum, the maximum of 60 random observations lies within the top 5% of the true maximum, with 95% probability. That may sound complicated, but it's not. Imagine the 5% interval around the true maximum. Now imagine that we sample points from this space and see if any of it lands within that maximum. Each random draw has a 5% chance of landing in that interval, if we draw n points independently, then the probability that all of them miss the desired interval is $(1 - 0.05)^n$. So the probability that at least one of them succeeds in hitting the interval is 1 minus that quantity. We want at least a .95 probability of success. To figure out the number of draws we need, just solve for n in the equation:

$$1 - (1 - 0.05)^n > 0.95.$$

We get $n \geq 60$. Ta-da!

The moral of the story is: if the close-to-optimal region of hyperparameters occupies at least 5% of the grid surface, then random search with 60 trials will find that region with high probability.

<https://web.archive.org/web/20160701182750/http://blog.dato.com/how-to-evaluate-machine-learning-models-part-4-hyperparameter-tuning>



In grid search and random search, each hyperparameter **guess is independent**.

Random search and grid search drawback:

There is **no guarantee of finding a local minimum to some precision** except if the search space is thoroughly sampled.

When Random search and grid search are useful:

- 1) The **model is very fast to run** and
- 2) **the number of hyperparameters is low**.

Sequential search techniques

When we train in more complex models like neural networks, the training of the model becomes **very costly, both in time and money**.

Solution??

In sequential search techniques:

- 1) We **pick a few hyperparameter** settings,
- 2) **Evaluate the quality** and then
- 3) **Decide where to sample next**.

- Sequential search is an **iterative and sequential process**
- It is **not parallelizable** (at least not fully)
- **Goal:** Make **fewer evaluations**, only of those that have the **more promising candidate hyperparameters**

Trade-off:

Training time on machine learning models **is less**,
but then
we need **more time to estimate where to sample**
the hyperparameter space next.

Sequential Model-Based Optimization



SMBO

Example:

- Bayesian optimization

Bayesian optimization

Bayes' Rule

find a distribution for A
with the evidence of B

$$\begin{array}{c} \text{posterior} \\ \underbrace{\hspace{1.5cm}} \\ P(A | B) = \frac{\overbrace{P(B | A)}^{\text{likelihood}} \times \overbrace{P(A)}^{\text{prior}}}{\underbrace{P(B)}_{\text{Evidence}}} \end{array}$$

راست نمایی (Likelihood)
احتمال مشاهده و وقوع شواهد موجود
به شرط درستی فرضیه

تابع احتمال پیشین (Prior)
احتمال درستی فرضیه بدون در
نظر گرفتن شواهد خاص

$$P(x|y) = \frac{P(y|x) P(x)}{P(y)}$$

تابع احتمال پسین (Posterior)
احتمال وقوع فرضیه با در نظر
گرفتن شواهد موجود

تابع احتمال مرزی یا حاشیه‌ای (Marginal)
احتمال مشاهده و وقوع شواهد موجود
با در نظر گرفتن همه فرضیات ممکن

kalam.ir

With **Bayes' Rule** we infer a posterior $P(A | B)$ from a Prior (A)

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

Bayesian optimization

Bayes' Rule

The diagram shows the equation $P(A | B) \propto P(B | A) \times P(A)$. Above the equation, three labels are placed: 'posterior' above $P(A | B)$, 'likelihood' above $P(B | A)$, and 'prior' above $P(A)$. Each label is connected to its corresponding term by a bracket. A red arrow points from the word 'proportional' (located below the equation) to the symbol \propto .

$$\begin{array}{ccc} \text{posterior} & & \text{likelihood} & & \text{prior} \\ \hline P(A | B) & \propto & P(B | A) & \times & P(A) \end{array}$$

↑
proportional

With **Bayes' Rule** we infer a posterior $P(A | B)$ from a Prior $P(A)$

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

Bayesian optimization

Hyperparameter Optimization

Mathematically, we want to find the global maximizer (or minimizer) of an unknown (black-box) objective function f :

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

f can be considered as a function to be minimized (e.g., validation loss)

where \mathbf{x} are the hyperparameters.

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

$$P(w | D) = \frac{P(D | w) \times P(w)}{P(D)}$$

$w=f(x)$ where x is a hyperparameter
 D is the set of hyperparameters x
 that we know the value of $f(x)$

- Let w be $f(x)$ and D be the available data
- w is unknown, so we treat it as a random function and place a prior over it $\rightarrow P(w)$
- $P(w)$ captures our beliefs about the possible values of w
- sequentially Given D and the likelihood model $P(D | w)$, we can infer the posterior $P(w | D)$ using Bayes' rule

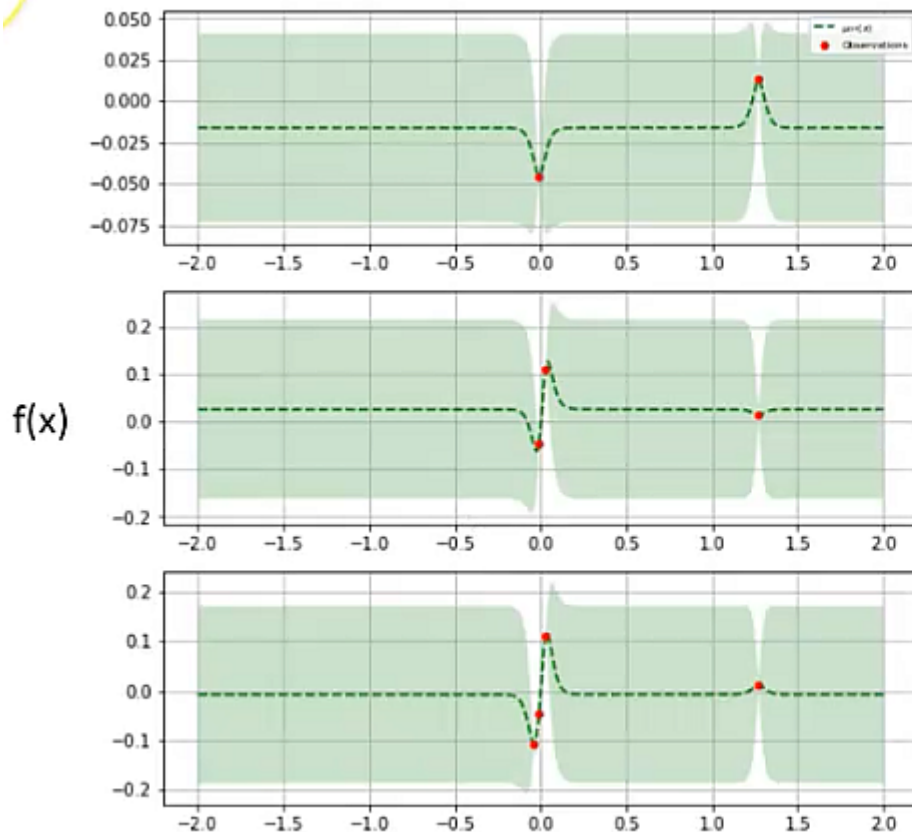
- The posterior $P(w | D)$ represents our updated belief of w after contemplating D .
- Given the posterior distribution $P(w | D)$, we construct an acquisition function to determine the next query point to sample w .

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

21 / 45

Bayesian optimization with an example

$f(x)$ can take every values in green area

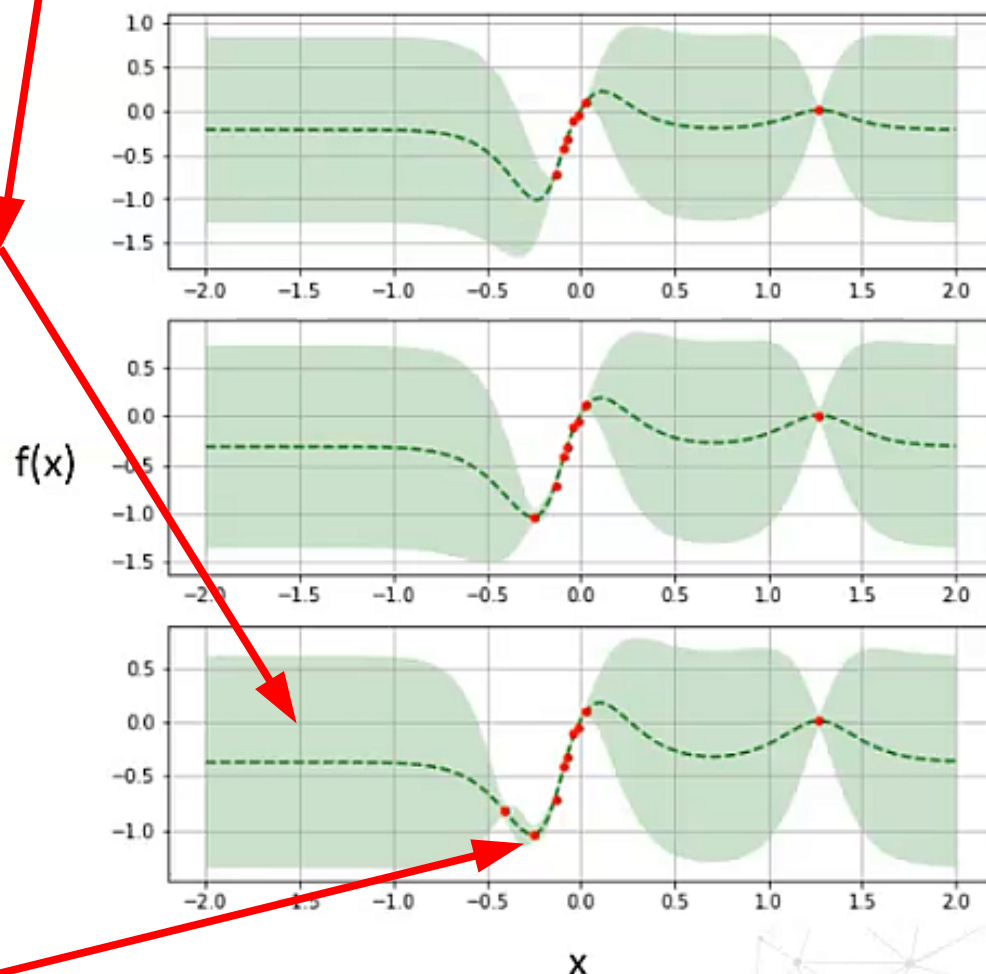
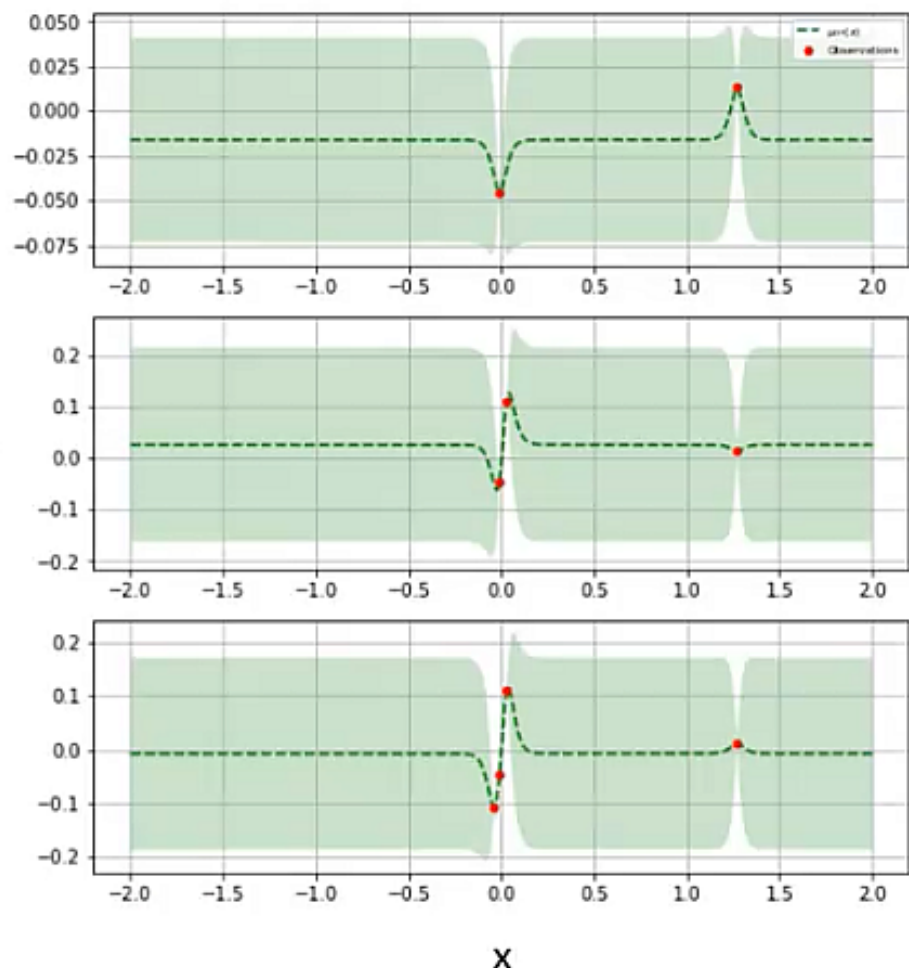


Since we want to minimize $f(x)$, the algorithm choose the 3rd point as above on the left (not right) and then update $p(w|D)$ based on this new value and choose the 4th point

the dotted green line is the mean of $f(x)$

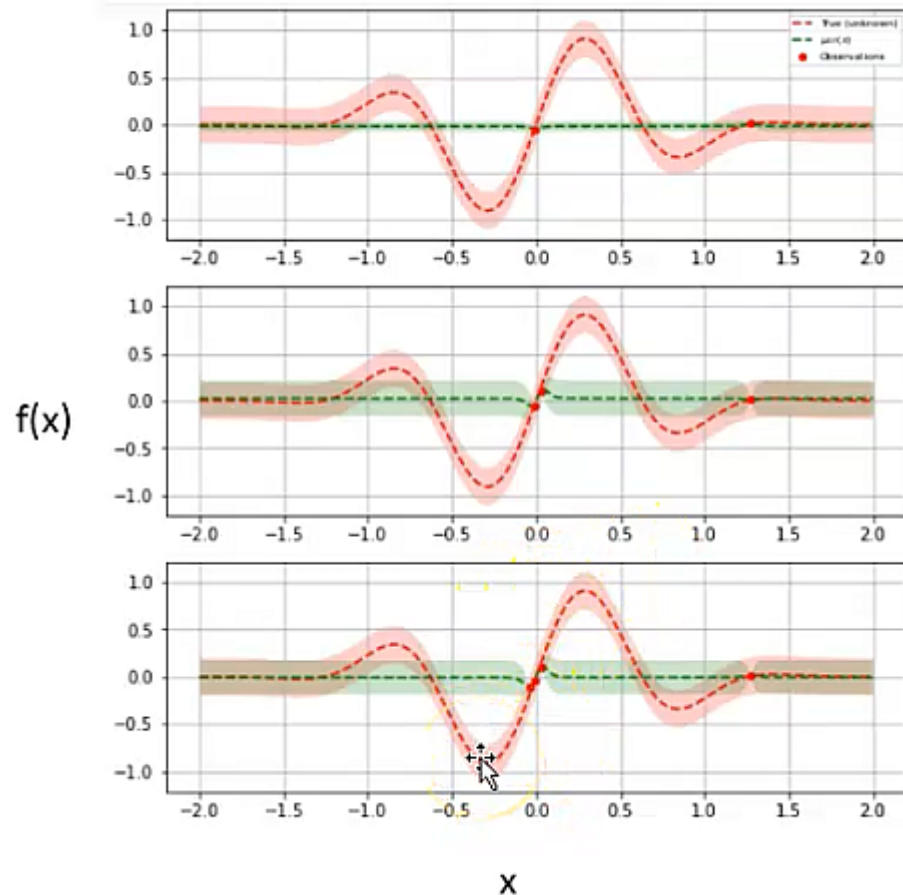
- The green area is our Prior, $P(w)$
- D is the red dots, where we evaluate $f(x)$
evidence
- We adjust the posterior $P(w | D)$ based on D .
- The uncertainty decreases around the evidence. As D , take the hyperparameters 0 and 2, and do cross-validation to evaluate the loss of validation (values of $f(x)$) for each of these values
choose to explore on the left or exploit near points 1 and 3

we know nothing about the shape of $f(x)$ here but there is no matter because our goal is to find the minimum of $f(x)$ not the real shape of it



The algorithm continue this procedure to reach a local minima. Then it can decide to continue to explore or stop at this point.

Bayesian optimization with an example



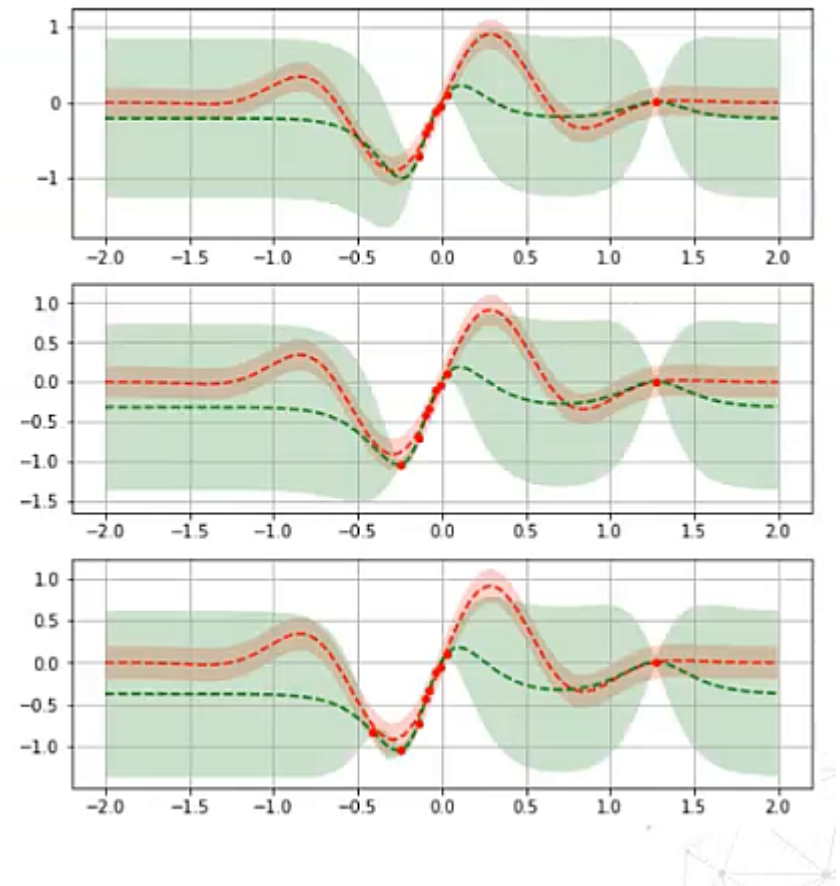
- Red is $f(x)$
- The green area is our Prior, $P(w)$
- D is the red dots, where we evaluate $f(x)$
- We adjust the posterior $P(w | D)$ based on D .
- Note how with Bayes Optimization we find the minimum of $f(x)$.

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

24 / 45

Bayesian optimization with an example

- Red is $f(x)$
- As we gather more data, the posterior $P(w \mid D)$ increases its uncertainty in unexplored areas and decreases its uncertainty in explored areas



<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

Estimating the prior $p(w) = p(f(x))$

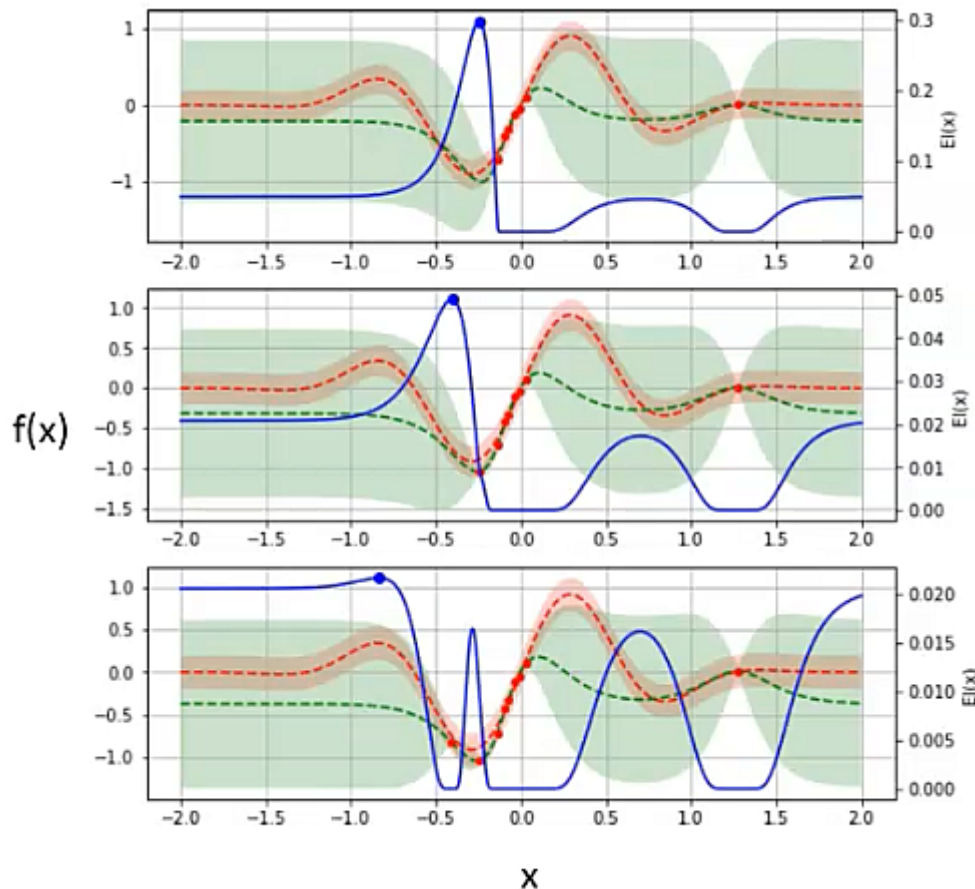
- Gaussian processes
- Tree-parzen estimator TPE
- Random Forests

We need to find a surrogate model for $f(x)$

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

26 / 45

Acquisition function



- We also need a way to estimate where to sample next → **acquisition function**
- Acquisition function values are high where the mean of $f(x)$ is low (or high) → **exploitation**.
- Acquisition function values are high when the variance of $f(x)$ is high → **exploration**.

The algorithm choose the next hyperparameter (evidence) where the **acquisition function** value is high

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

27 / 45

Acquisition Function

- Expected Improvement (EI)
- Gaussian process upper confidence bound (UCB)

<https://www.courses.trainindata.com/p/hyperparameter-optimization-for-machine-learning>

28 / 45

Tree-based Parzen Estimators (TPE) Optimization

- The idea of Tree-based Parzen optimization is **similar to Bayesian optimization**.
- Instead of finding the values of $p(w|D)$ where w is the function to be minimized (e.g., validation loss) and D is the value of hyperparameter the **TPE models $P(D|w)$ and $P(w)$** .
- **TPE works extremely well in practice** and was battle-tested across most domains.

Search spaces (What range to choose?)

A workflow to specify appropriate search spaces:

- Choose what hyperparameters are reasonable to optimize
- Define broad ranges for each of the hyperparameters (including the default where applicable)
- At first, run a small number of trials
- Select the runs with the best score
- Move the range towards those higher/lower values when the best runs' hyperparameter values are pushed against one end of a range
- Determine whether certain hyperparameter values cause fitting to take a long time (and avoid those values)
- Re-run with more trials
- Repeat until the best runs are comfortably within the given search bounds and none are taking excessive time

Search spaces

Use **.quniform** for scalars, **.choice** for categoricals

Consider choosing the maximum depth of a tree building process. This must be an integer like 3 or 10.

Some hyperparameter optimization tools offers **.choice** and **.randint** to choose an integer from a range

Users commonly choose **.choice** as a sensible-looking range type

These are exactly the wrong choices for such a hyperparameter

Search spaces

Use `.quniform` for scalars, `.choice` for categoricals

While these will generate integers in the right range, in these cases, optimizer would not consider that a value of “10” is larger than “5” and much larger than “1”, as if scalar values. Yet, that is how a maximum depth parameter behaves. If 1 and 10 are bad choices, and 3 is good, then it should probably prefer to try 2 and 4, but **it will not learn that with `.choice` or `.randint`.**

Instead, **the right choice is `.quniform` (“quantized uniform”) or `.qloguniform`** to generate integers.

`.choice` is the right choice when, for example, choosing among categorical choices (which might in some situations even be integers, but not usually).

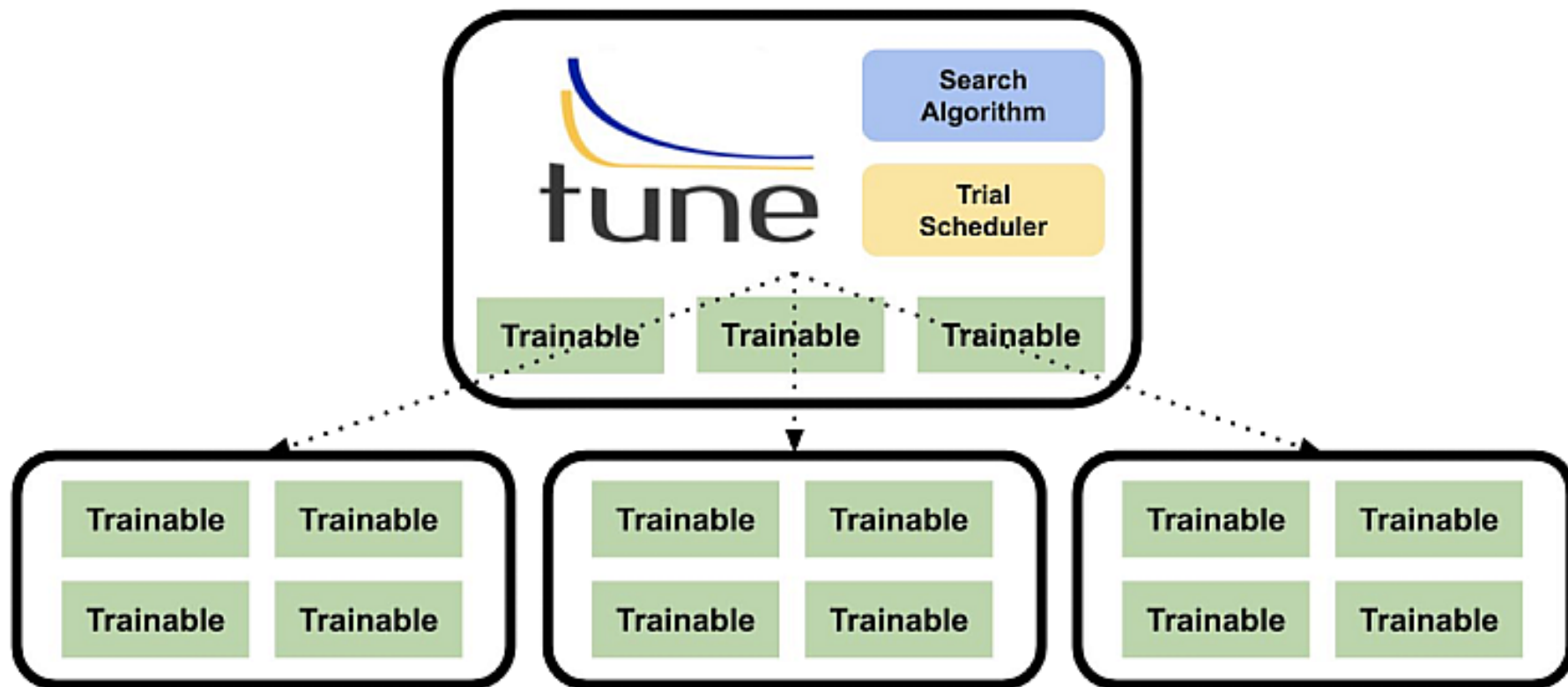
Best tools for hyperparameter optimization

- Scikit-learn
- Scikit-Optimize
- Optuna
- Hyperopt
- Ray Tune
- BayesianOptimization
- GPyOpt
- SigOpt

	- Collapse all	Ray Tune	Optuna	Hyperopt	Scikit-Optimize
	Open sourced	✓	✓	✓	✓
	Complete package				
	Algorithms	Ax/Botorch, HyperOpt, and Bayesian Optimization	AxSearch, DragonflySearch, Hyperband	Random Search, Tree of Parzen Estimators, Adaptive TPE	Bayesian Hyperparameter Optimization
	Various supported frameworks	Pytorch, Tensorflow, XGBoost, LightGBM, Scikit-Learn, and Keras	Any ML or Deep Learning framework, PyTorch, TensorFlow, Keras, MXNet, Scikit-Learn, LightGBM	sklearn, xgboost, Tensorflow, pytorch, etc	Machine Learning algorithms offered by the scikit-learn library
	Few lines of codes	✓	✓	✓	✓
	Uses GPUs	✓		✓	
	Easy scalability with little or no changes to the code	✓	✓	✓	✓
	Parallelized training	✓	✓	✓	✓
	Distributed optimization	✓	✓	✓	
	Auto filled metrics				
	Handling large datasets	✓	✓	✓	
	Free or paid	Free	Free	Free	Free

Hyperparameters Optimization with Ray Tune

Tune handles hyperparameter search execution.



<https://docs.ray.io/en/latest/tune/index.html>

©2017 RISELab

Ray Tune

Tune is built with Deep Learning as a priority.



<https://docs.ray.io/en/latest/tune/index.html>

36 / 45

Ray Tune

Tune Algorithm Offerings

Trial Schedulers Provided

- **Population-based Training**
- **HyperBand**
- **ASHA**
- **Median-stopping Rule**
- **BOHB**

Search Algorithms Provided

- **HyperOpt (TPE)**
- **Bayesian Optimization**
- **SigOpt**
- **Nevergrad**
- **Scikit-Optimize**
- **Ax/Botorch (PyTorch BayesOpt)**

<https://docs.ray.io/en/latest/tune/index.html>

37 / 45

Some hyperparameter optimization algorithms

Ray Tune

Bayesian optimization

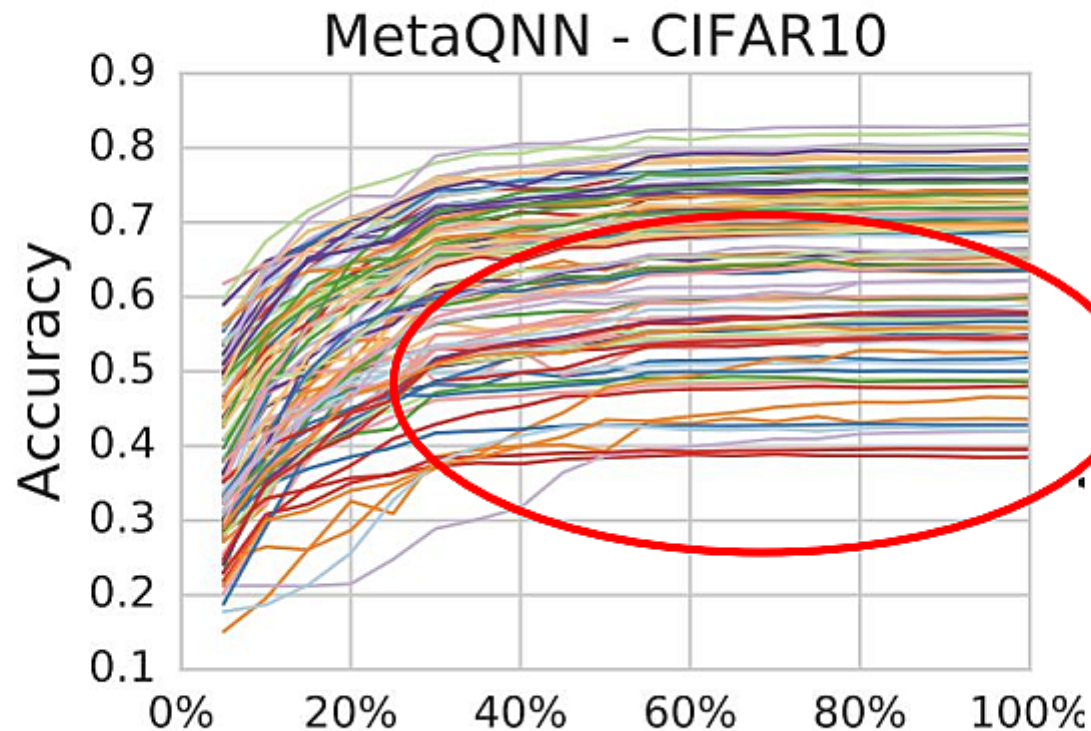


- Can utilize prior information
 - Semi-Parallelizable
- (Kandasamy 2018)

Still can do better!

Ray Tune

We can do better by exploiting structure!



*Why
waste
resources
on this?*

<https://docs.ray.io/en/latest/tune/index.html>

Ray Tune

HyperBand/ASHA (early stopping algorithms)

```
trial = sample_from(hyperparameter_space)
while trial.iter < max_epochs:
    trial.run_one_epoch()
    if trial.at_cutoff():
        if is_top_fraction(trial, trial.iter):
            trial.extend_cutoff()
        else:
            # allow new trials to start
            trial.pause(); break
```

Intuition:

1. Compare relative performance
2. Terminate bad performing trials
3. Continue better trials for longer period of time

Notes:

1. Can be combined with Bayesian Optimization
2. Can be easily parallelized

Ray Tune

Population-based training

Main idea:

Evaluate a population in parallel.

Terminate lowest performers.

Copy weights of the best performers and mutates hyperparameters

Benefits:

1. Easily parallelizable
2. Can search over "schedules"
3. Terminates bad performers

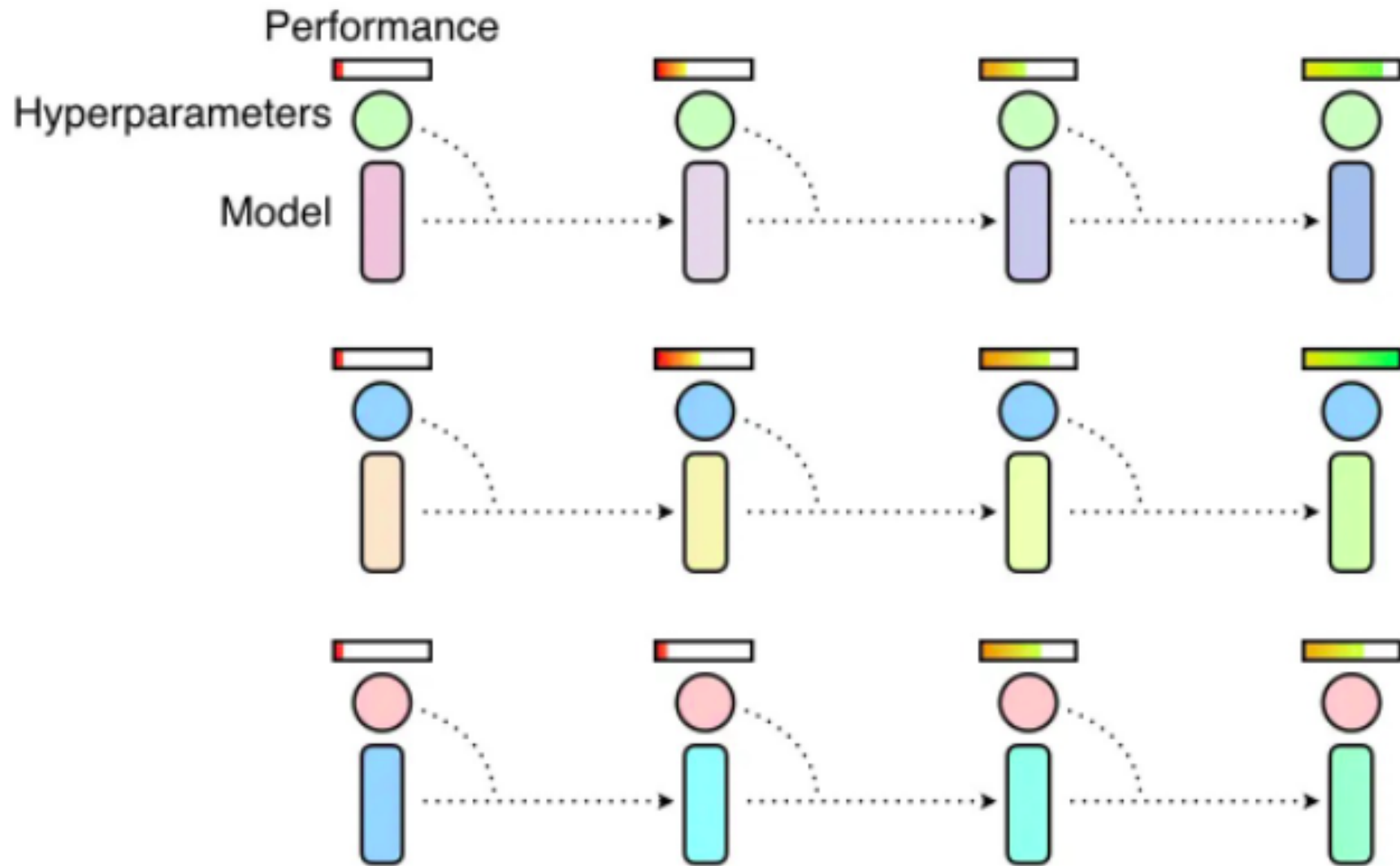


©2017 RISELab

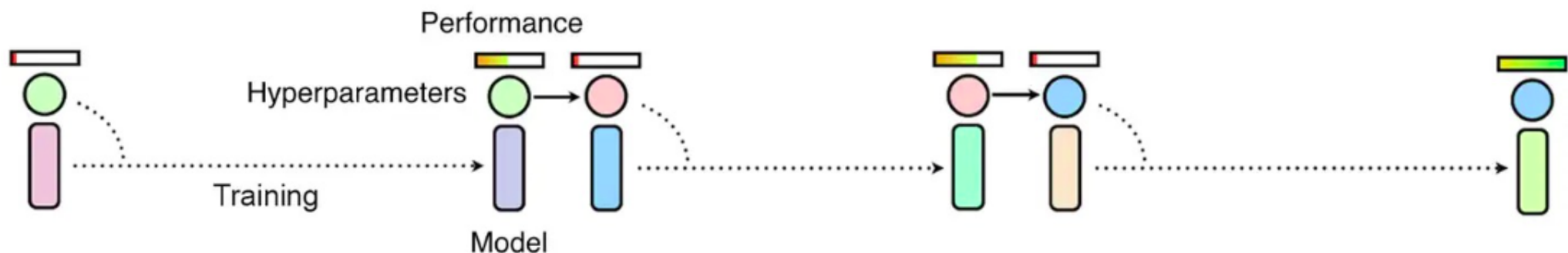


<https://docs.ray.io/en/latest/tune/index.html>

42 / 45

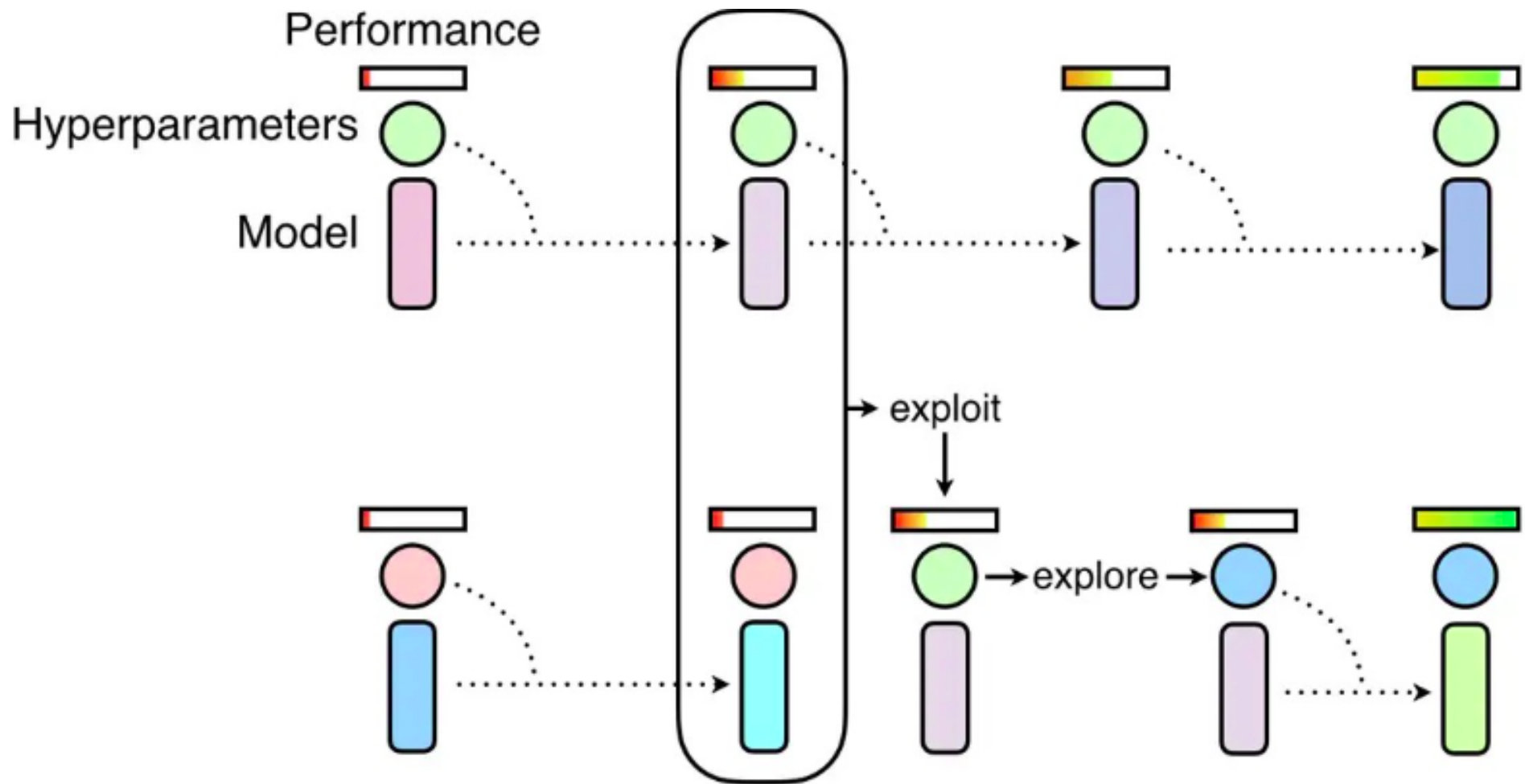


Random search of hyperparameters, where many hyperparameters are tried in parallel, but independently. Some hyperparameters will lead to models with good performance, but others will not



Methods like hand tuning and Bayesian Optimization make changes to the hyperparameters by observing many training runs sequentially, making these methods slow

<https://www.deepmind.com/blog/population-based-training-of-neural-networks>



Population Based Training of neural networks starts like random search, but allows workers to exploit the partial results of other workers and explore new hyperparameters as training progresses

<https://www.deepmind.com/blog/population-based-training-of-neural-networks> 45 / 45