Recitation 7

Hyperparameter optimization

KONSTANTIN KRISMER

MIT - 6.802 / 6.874 / 20.390 / 20.490 / HST.506 - Spring 2019

2019-03-21

What is hyperparameter optimization?

Goal: Find hyperparameter configuration $\boldsymbol{\theta}$ that minimizes validation set loss of trained model, i.e.,

$$\operatorname*{argmin}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \mathcal{L}(\mathcal{M}(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}),$$

where

- $\boldsymbol{\Theta}$ is the hyperparameter space,
- $\mathcal{L}(\cdot, \cdot)$ is the loss function,
- $\mathcal{M}(\boldsymbol{x}; \boldsymbol{\theta})$ returns the predictions for $\boldsymbol{x}$ of the model trained with hyperparameter configuration $\boldsymbol{\theta}$.

General procedure:

1. *(outer loop)* select hyperparameter configuration $\boldsymbol{\theta} \in \boldsymbol{\Theta}$
2. *(inner loop)* train model $\mathcal{M}$ with hyperparameters $\boldsymbol{\theta}$ using training set
3. *(inner loop)* calculate loss on validation set of model $\mathcal{M}$
4. *(outer loop)* repeat or return $\boldsymbol{\theta}$ of model with lowest validation loss

Types of hyperparameters

Architectural hyperparameters:

- number of hidden layers
- type of a given hidden layer (dense, convolutional, recurrent)
- units per layer
- type of activation function
- strength and type of weight regularization and dropout
- weight initialization
- skip connections
- batch normalization

Hyperparameters of optimizer:

- type of optimizer (vanilla SGD, SGD with momentum, Adam)
- learning rate
- momentum

Hyperparameters of training process:

- batch size
- number of epochs

Types of hyperparameter search methods

Derivative-free / black box optimization methods:

- with independent draws of hyperparameter configurations:
  - grid search
  - random search
  - resource allocation, early stopping
- ealier draws inform later draws:
  - Bayesian optimization
  - evolutionary optimization
  - reinforcement learning based optimization

Gradient-based methods

Grid search vs random search?

Only difference in step 1 (how hyperparameters are chosen):

- grid search selects value of $\boldsymbol{\theta}$ based on rigid grid,
- random search samples $\boldsymbol{\theta}$ randomly* from $\boldsymbol{\Theta}$

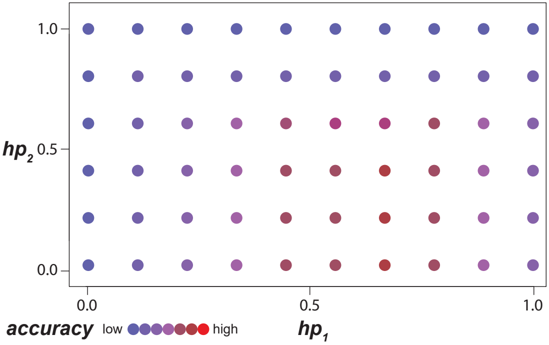


Figure: Grid search

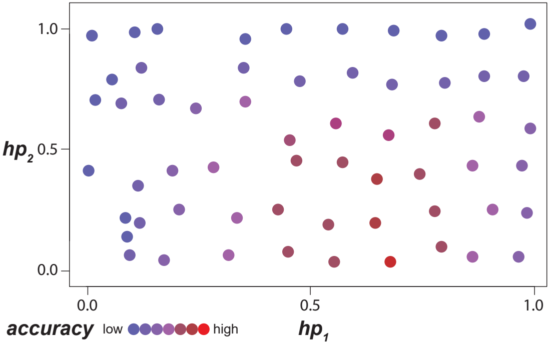


Figure: Random search

Grid search vs random search?

Only difference in step 1 (how hyperparameters are chosen):

- grid search selects value of $\boldsymbol{\theta}$ based on rigid grid,
- random search samples $\boldsymbol{\theta}$ randomly* from $\boldsymbol{\Theta}$

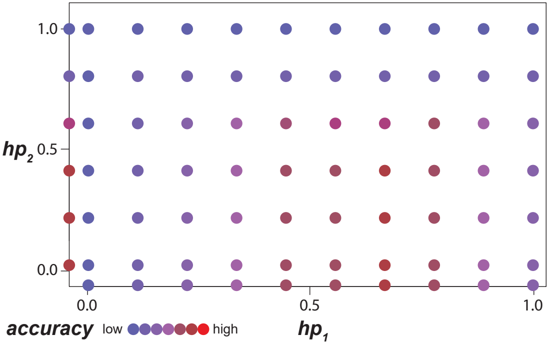


Figure: Grid search

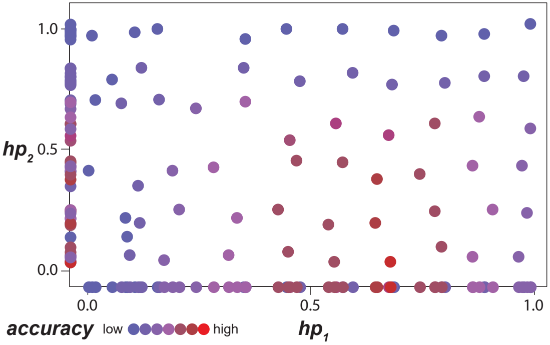


Figure: Random search

Grid search vs random search?

Only difference in step 1 (how hyperparameters are chosen):

- grid search selects value of $\boldsymbol{\theta}$ based on rigid grid,
- random search samples $\boldsymbol{\theta}$ randomly* from $\boldsymbol{\Theta}$

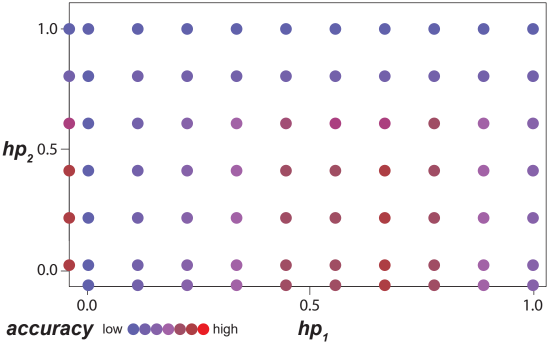


Figure: Grid search

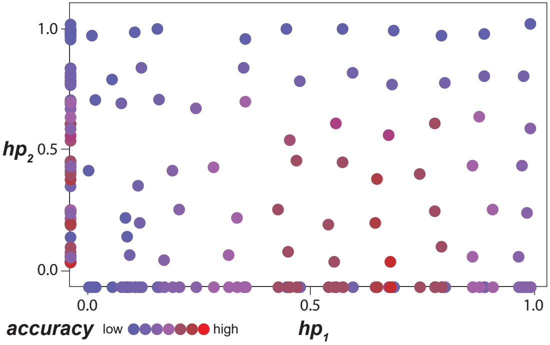


Figure: Random search

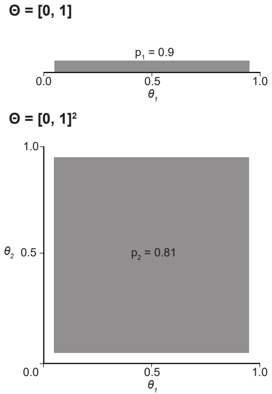
random search advantages:

- explores more values for each hyperparameter, given the same amount of trials
- (and often many hyperparameters only have limited influence on the loss function)

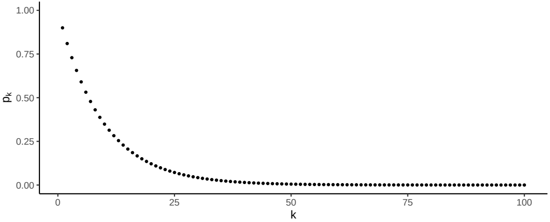
Curse of dimensionality

Problem: If $\mathbf{\Theta}$ is high-dimensional, sampling hyperparameters independently uniformly to obtain $\boldsymbol{\theta}$ fails to give uniformly looking draws

In hyperparameter space $\mathbf{\Theta} = [0,1]^k$, what is the probability p_k that a uniform draw of a hyperparameter configuration falls inside the hypercube[1] with all sides going from 0.05 to 0.95?



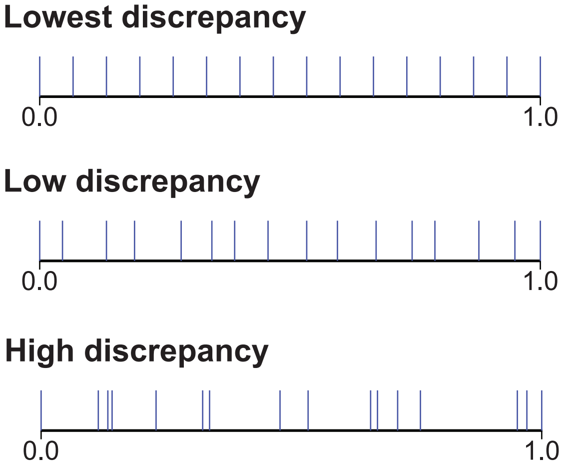
- if $k = 1$: $p_1 = P(\theta_1 \geq 0.05, \theta \leq 0.95) = 0.9$
- if $k = 2$: $p_2 = p_1^2 = 0.81$
- if $k = 3$: $p_3 = p_1^3 = 0.729$
- $p_k = p_1^k$



[1] interval (if $k = 1$), square (if $k = 2$), cube (if $k = 3$) or hypercube (if $k > 3$)

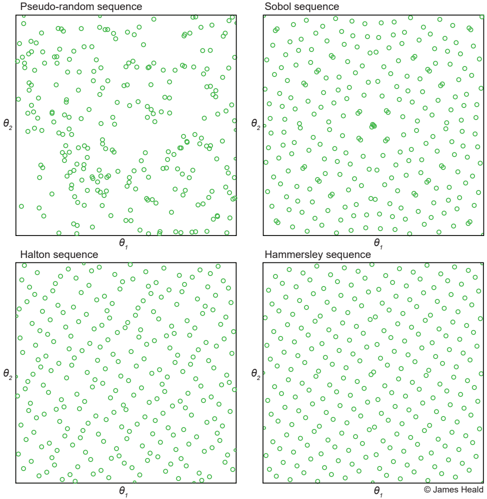
Random search sampling techniques

Solution: More representative of random uniform draws than actual random uniform draws are deterministic low-discrepancy sequences, where discrepancy is a measurement of highest or lowest density of points in a sequence.



Random search sampling techniques

Solution: More representative of random uniform draws than actual random uniform draws are deterministic low-discrepancy sequences, where discrepancy is a measurement of highest or lowest density of points in a sequence



- Halton sequence [Halton, 1964]
- Sobol sequence [Sobol, 1967]
- Hammersley point set [Hammersley, 1960]
- Poisson disk sampling [Gamito and Maddock, 2009]
- latin hypercube sampling [McKay et al., 1979]

Can we do better than random search?

Hyperparameter optimization:

- Hyperband: resource allocation method based on infinite-armed bandit problem [Li et al., 2017]
- DNGO: Bayesian, neural network based [Snoek et al., 2015]
- Bayesian hyperparameter optimization based on Gaussian processes [Snoek et al., 2012]

Neural network architecture search (AutoML):
Idea: avoid hand crafted architectural solutions, set architectural hyperparameters based on optimization framework, not based on convention / previous experience

- ENAS: Efficient Neural Architecture Search [Pham et al., 2018]
- Neural architecture search and reinforcement learning [Zoph and Le, 2016, Baker et al., 2016]
- Neural architecture search and evolutionary algorithms [Chen et al., 2018]

Hyperband and Successive Halving

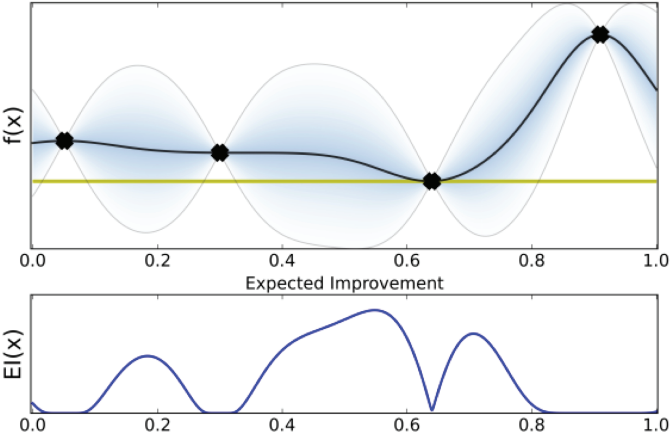
Hyperband: resource allocation method based on infinite-armed bandit problem [Li et al., 2017] (method for tuning iterative algorithms, trains many models with random hyperparameter configurations, aborts most of them early)

Hyperband uses *Successive Halving* at its core:

1. uniformly allocate a budget to a set of hyperparameter configurations
2. evaluate the performance of all configurations
3. throw out the bottom half of the configurations
4. repeat until one configuration remains

Bayesian hyperparameter optimization

Remember Bayesian optimization with expected improvement (Lecture 11):



where $x \in \Theta$ and $f(x)$ is the validation loss of model trained using hyperparameter configuration x (i.e., θ).

Hyperparameter search with evolutionary algorithms

Evolutionary algorithms are population-based metaheuristic optimization algorithms that use mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection.

- population-based: unlike gradient-based methods which follow one solution (one trajectory) through search space, population-based methods follow many solutions through search space (e.g., ant colony optimization, particle swarm optimization, evolutionary algorithms)
- heuristics: unlike exact algorithms, heuristics do not guarantee to find the optimal solution in a finite amount of time
- metaheuristics: high-level, problem-independent strategies to develop heuristic optimization algorithms

Evolution as optimization method

Analogies:

validation loss (objective function)	fitness (survivability, fertility)
hyperparameter configuration (solution candidate)	individual
solution representation θ	genome
solution manipulation	recombination and mutation
choice of solution	natural selection (reproductive success)

Operators:

- selection operator: select hyperparameter configurations with probability inversely proportional to their validation loss
- recombination or crossover operator: combines two parental hyperparameter configurations to create one offspring
- mutation operator: randomly change one or more hyperparameters of a hyperparameter configuration

Generational replacement schemes:

- full generational replacement
- n-elitism
- tournament replacement

Evolution as optimization method

Procedure:

1. evaluate fitness of all individuals in population (calculate validation loss of all models)
2. select parents for recombination with selection operator
3. introduce new genetic diversity with recombination and mutation operators
4. select individuals for new generation
5. repeat

Libraries for hyperparameter search

Don't reinvent the wheel!

- **Ray Tune**: `https://ray.readthedocs.io/en/latest/tune.html`
- **Optunity**: `https://github.com/claesenm/optunity`
- DEAP (evolutionary computation framework) [Fortin et al., 2012]: `https://github.com/deap/deap`
- HyperEngine (Bayesian hyper-parameters optimization): `https://github.com/maxim5/hyper-engine`
- Hyperopt (tree-structured Parzen estimators): `http://hyperopt.github.io/hyperopt/`
- skopt (sequential model-based optimization): `https://scikit-optimize.github.io/`

Hyperparameter search framework: Ray Tune

Supports these search algorithms:

- grid search
- random search
- Hyperband [Li et al., 2017]
- BayesOpt search: sequential model-based hyperparameter optimization [Snoek et al., 2012]
- tree-structured Parzen estimators [Bergstra et al., 2011]

Using **Ray Tune** on top of **Ray** to run hyperparameter optimization on several machines / GPUs in parallel, derivative-free

- Ray Tune: a hyperparameter tuning framework for long-running tasks such as training Deep Residual Networks
- Ray: high-performance distributed execution framework with built-in support for CPUs and GPUs

Hyperparameter search framework: Optunity

Supports these search algorithms:

- grid search
- random search
- Nelder-Mead method (downhill simplex method) [Nelder and Mead, 1965]
- particle swarm optimization [Kennedy and Eberhart, 1995]
- CMA-ES: covariance matrix adaptation evolution strategy [Hansen and Kern, 2004]
- tree-structured Parzen estimators [Bergstra et al., 2011]
- BayesOpt search: sequential model-based hyperparameter optimization [Snoek et al., 2012]

Recommendations for projects

- model: use `tf.keras.models.Sequential()` or own class (which inherits from `tf.keras.models.Model()`)
- layers: use `tf.keras.layers`, not `tf.layers`
- (use `tf.nn` for low-level control)
- (TF2.0) losses: use `tf.keras.losses`, not `tf.losses`
- (TF2.0) optimizers: use `tf.keras.optimizers`, not `tf.train`
- (TF2.0) metrics: use `tf.keras.metrics`, not `tf.metrics`
- training loop: use tf.keras `model.fit` or `tf.estimator.train_and_evaluate`
- use TensorBoard to visualize training process
- use Ray Tune or Optunity for hyperparameter optimization
- save models to disk

check out TensorFlow Probability:

- `https://www.youtube.com/watch?v=BrwKURU-wpk`
- `https://www.tensorflow.org/probability`

TensorBoard: visualize your training process

`tensorboard --logdir=path/to/log-directory`

`http://localhost:6006/`

TensorBoard: visualize your training process

Listing 1: Tensorboard hook for tf.keras training loop

```
import time
import tensorflow as tf

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(10, input_shape=(784,)))
model.add(tf.keras.layers.Activation('softmax'))

model.compile(optimizer='sgd', loss='categorical_crossentropy')

tensorboard = tf.keras.callbacks.TensorBoard(log_dir="logs/{}".format(time.time()))

model.fit(x_train, y_train, verbose=1, callbacks=[tensorboard])
```

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016).
Designing neural network architectures using reinforcement learning.
CoRR, abs/1611.02167.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011).
Algorithms for hyper-parameter optimization.
In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, USA. Curran Associates Inc.

Chen, Y., Zhang, Q., Huang, C., Mu, L., Meng, G., and Wang, X. (2018).
Reinforced evolutionary neural architecture search.
CoRR, abs/1808.00193.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012).
DEAP: Evolutionary algorithms made easy.
Journal of Machine Learning Research, 13:2171–2175.

Gamito, M. N. and Maddock, S. C. (2009).
Accurate multidimensional poisson-disk sampling.
ACM Trans. Graph., 29(1):8:1–8:19.

References II

Halton, J. H. (1964).
Algorithm 247: Radical-inverse quasi-random point sequence.
Commun. ACM, 7(12):701–702.

Hammersley, J. M. (1960).
Monte carlo methods for solving multivariable problems.
Annals of the New York Academy of Sciences, 86(3):844–874.

Hansen, N. and Kern, S. (2004).
Evaluating the CMA evolution strategy on multimodal test functions.
In Yao, X. et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer.

Kennedy, J. and Eberhart, R. C. (1995).
Particle swarm optimization.
In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017).
Hyperband: A novel bandit-based approach to hyperparameter optimization.
J. Mach. Learn. Res., 18(1):6765–6816.

References III

McKay, M. D., Beckman, R. J., and Conover, W. J. (1979).
A comparison of three methods for selecting values of input variables in the analysis of output from a computer code.
Technometrics, 21(2):239–245.

Nelder, J. A. and Mead, R. (1965).
A Simplex Method for Function Minimization.
The Computer Journal, 7(4):308–313.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018).
Efficient neural architecture search via parameters sharing.
In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104, Stockholmsmässan, Stockholm Sweden. PMLR.

Snoek, J., Larochelle, H., and Adams, R. P. (2012).
Practical bayesian optimization of machine learning algorithms.
In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2951–2959, USA. Curran Associates Inc.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015).
Scalable bayesian optimization using deep neural networks.
In *International Conference on Machine Learning*, pages 2171–2180.

References IV

Sobol, I. (1967).
On the distribution of points in a cube and the approximate evaluation of integrals.
USSR Computational Mathematics and Mathematical Physics, 7(4):86 – 112.

Zoph, B. and Le, Q. V. (2016).
Neural architecture search with reinforcement learning.
CoRR, abs/1611.01578.