Hyperparameter optimization

Fraida Fund

Contents

Math prerequisites for this lecture: None.

Recall: Supervised learning recipe

1. Get **data**: $(\mathbf{x_i}, y_i), i = 1, 2, \cdots, n$
2. Choose a **model**: $\hat{y_i} = f(\mathbf{w}, \mathbf{x_i})$
3. Choose a **loss function**
4. Find model **parameters** that minimize loss

- **Parameters** are learned by the training algorithm in step 4.
- **Hyperparameters** are *not* learned by the training algorithm
  - some affect the shape of the model $f()$ in step 2, e.g. SVM kernel hyperparameters
  - some affect the training process in step 4, e.g. learning rate and batch size in gradient descent

We know how to find parameters - how do we find hyperparameters?

Hyperparameter optimization

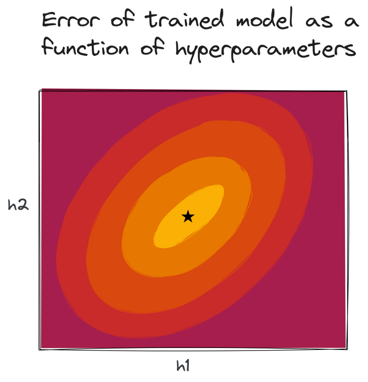

Figure 1: Hyperparameter search space

The validation MSE of the trained model depends on the hyperparameters. Goal of hyperparameter optimization: find the set of hyperparameters for which the validation MSE is minimized.

Grid search

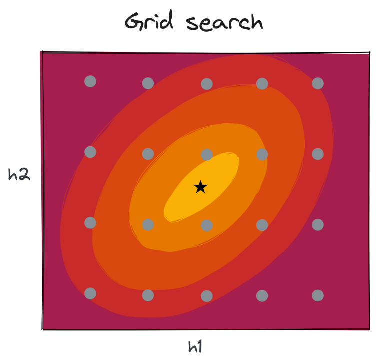

Figure 2: Grid search

Grid search is the extension of cross validation to higher dimensions. Note: you need to know which part of the hyperparameter space to search in the first place!

Depending on the initial results, you may consider extending the grid in another iteration:

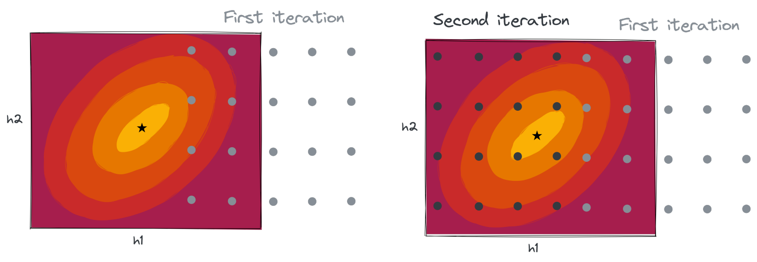

Figure 3: Extending the grid.

Or increasing the resolution of the grid in another iteration:

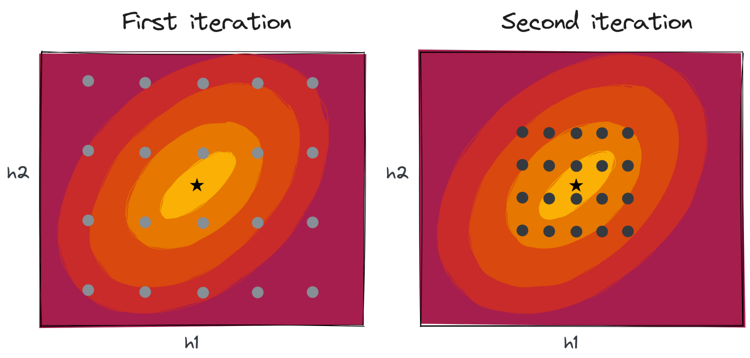

Figure 4: Searching a finer grid

One nice thing about grid search: if you have multiple cores available, you can train these models in parallel.

Random search

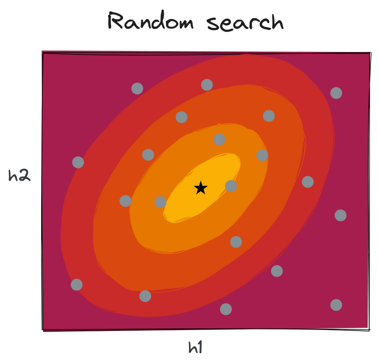

Figure 5: Random search

Points in the hyperparameter space are sampled from some pre-specified distribution -

- distribution does not need to be uniform!
- you can specify the number of points to sample, to control the search
- can often find good hyperparameters more quickly than grid search (sample fewer points)

Successive halving

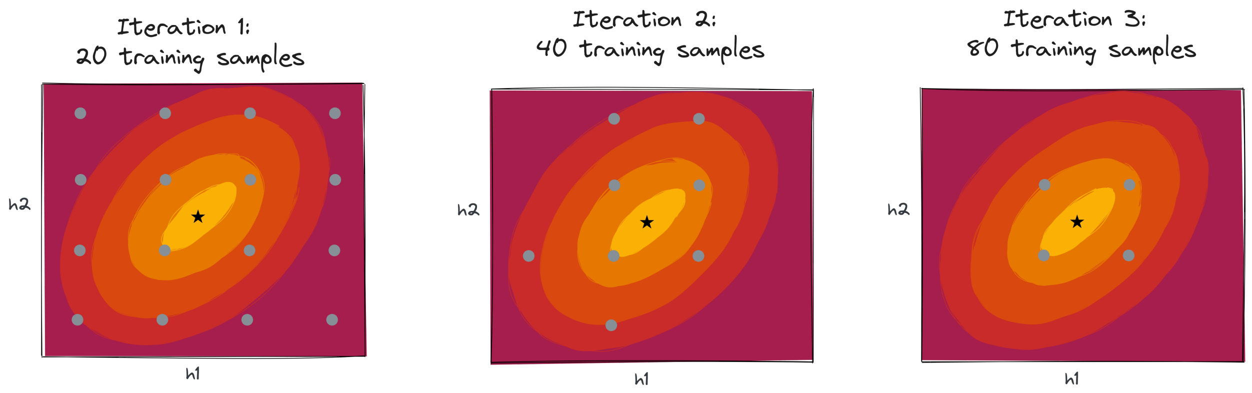

Figure 6: Successive halving with grid search

Works with either grid or random search -

- **Iteration 1**: train n models on a small subset of training data - say, m samples
- **Iteration 2**: train the $\frac{n}{2}$ *best* models from the previous iteration on $2m$ samples
- **Iteration 3**: train the $\frac{n}{4}$ *best* models from the previous iteration on $4m$ samples
- ... and so on

Idea: spend a little bit of compute resources to explore a lot of the hyperparameter space, then spend more compute resources to focus on the most promising parts of the hyperparameter space.

Adaptive search (Bayes search)

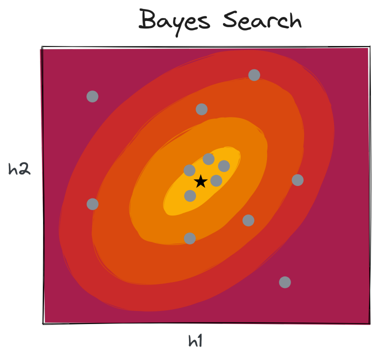

Figure 7: Bayes search

Finally, we can consider a more advanced search technique, in which we use the information we have already gathered about the hyperparameter surface in order to decide which points to visit next.

We will choose the next sample in order to balance exploration of the hyperparameter space, versus focus on the areas where we believe (based on what we have seen so far) that the best model is likely to be.