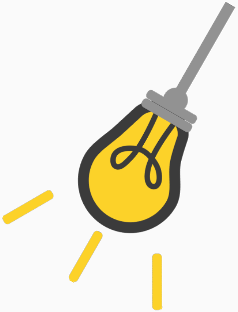
浙江大学爱丁堡大学联合学院
ZJU-UoE Institute

Lecture 17 - Hyperparameter tuning

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

- Give examples of hyperparameters and explain what are the issues with hyperparameters in deep learning.
- Discuss and compare different methods of tuning hyperparameters.
- Implement these methods in Python.



Introduction

What are hyperparameters?

Hyperparameters are values that are used by a ML model to control the learning process.

Examples include:

- Structural parameters
  - Number of layers
  - Number of units in each layer
  - Number of filters, their kernel size, stride and padding
  - Type of activation function
  - ...
- Learning parameters
  - Type of optimizer
  - Learning rate α
  - Schedule of learning rate
  - Loss function
  - Dropout rate
  - Weight regularization
  - ...
- ...

What are hyperparameters?

Hyperparameters are values that are used by a ML model to control the learning process.

Examples include:

- Structural parameters
    - Number of layers
    - Number of units in each layer
    - Number of filters, their kernel size, stride and padding
    - Type of activation function
    - ...
- Learning parameters
    - Type of optimizer
    - Learning rate α
    - Schedule of learning rate
    - Loss function
    - Dropout rate
    - Weight regularization
    - ...
- ...

Other **parameters** such as weights and biases are learned by the network during training; these will be affected by the choice of hyperparameters.

Problems with choosing hyperparameters

- Deep network have an extremely large number of hyperparameters.
- The search space is extremely large but often the "good solution" only covers a small region of this space.
- We cannot try all possible combinations.
- No definitive strategy exists.

So... how do we choose hyperparameters?

Choosing hyperparameters

Choosing hyperparameters

Manual methods

What did others do?

We can look at the hyperparameters of published models that performed similar tasks and start from there.

Pros

- If others have used this successfully, there are good chances that might work.
- Computationally fast! :)

What did others do?

We can look at the hyperparameters of published models that performed similar tasks and start from there.

Pros

- If others have used this successfully, there are good chances that might work.
- Computationally fast! :)

Cons

- It might be difficult to find a model doing exactly the same task.

Manual tweaking

- Start with a reasonable choice of hyperparameters
- Modify one parameter at a time to improve model performance

Manual tweaking

- Start with a reasonable choice of hyperparameters
- Modify one parameter at a time to improve model performance

Pros

- It generates high-quality results, when done by an expert.
- Humans can integrate knowledge from several sources and previous experience to make the best changes to hyperparameters.

Manual tweaking

- Start with a reasonable choice of hyperparameters
- Modify one parameter at a time to improve model performance

Pros

- It generates high-quality results, when done by an expert.
- Humans can integrate knowledge from several sources and previous experience to make the best changes to hyperparameters.

Cons

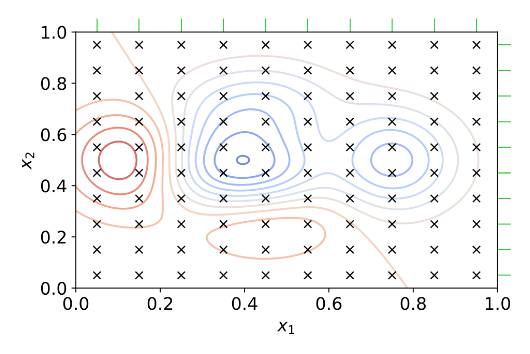
- It is not obvious to find a good starting point.
- It is extremely time-consuming.
- There is no guarantee to find the optimal model.
- Difficult process to replicate.

Choosing hyperparameters

Automated methods

Grid search

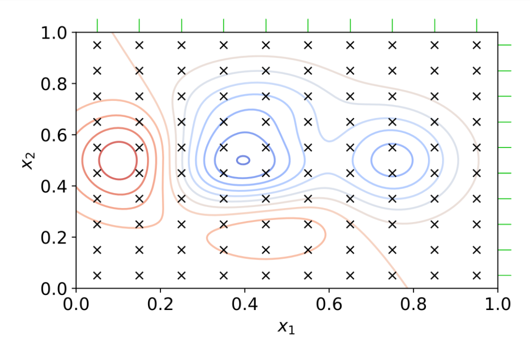
Grid search is the simplest method to use and is an exhaustive search over a defined parameter search space.



Grid search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1] - source Wikipedia

Grid search

Grid search is the simplest method to use and is an exhaustive search over a defined parameter search space.



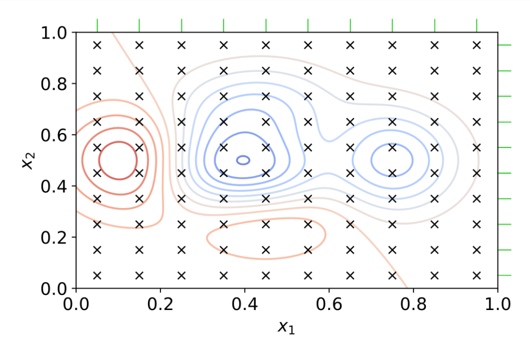
Grid search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1] - source Wikipedia

Pros

- It is easy to implement.
- Exhaustive search - guaranteed to find the optimal hyperparameters.
- Reproducible.

Grid search

Grid search is the simplest method to use and is an exhaustive search over a defined parameter search space.



Grid search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1] - source Wikipedia

Pros

- It is easy to implement.
- Exhaustive search - guaranteed to find the optimal hyperparameters.
- Reproducible.

Cons

- It is computationally expensive -> early stop and parallelism help.
- Need to choose a range.
- It does not scale well; 5 hyperparameters with 10 values each, give 10^5 combinations.
- A lot of wasted computation time to sample low-accuracy combinations.

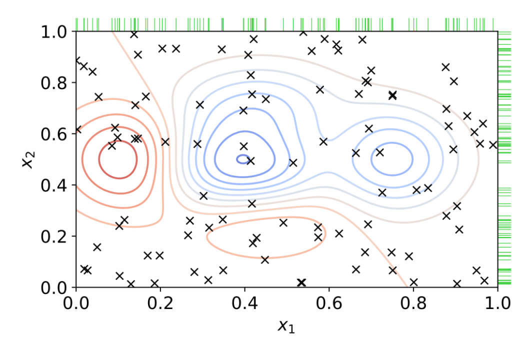
Grid search using Python

We will now see a simple implementation of grid search in Python.

See the attached `GridSearch.ipynb` notebook.

Random search

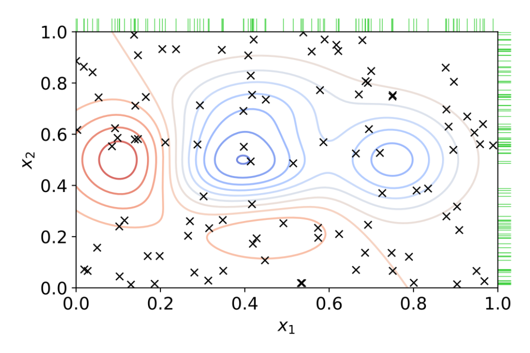
In **random search**, we randomly sample hyperparameters from the search space, rather than exploring a fixed grid.



Random search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Random points are chosen in the grid space - source Wikipedia

Random search

In **random search**, we randomly sample hyperparameters from the search space, rather than exploring a fixed grid.



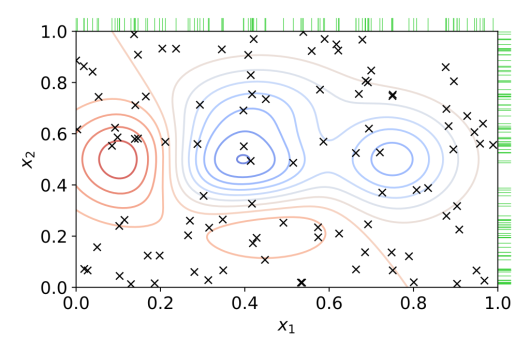
Random search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Random points are chosen in the grid space - source Wikipedia

Pros

- Still easy to implement.
- Helps when dealing with large search spaces.

Random search

In **random search**, we randomly sample hyperparameters from the search space, rather than exploring a fixed grid.



Random search - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Random points are chosen in the grid space - source Wikipedia

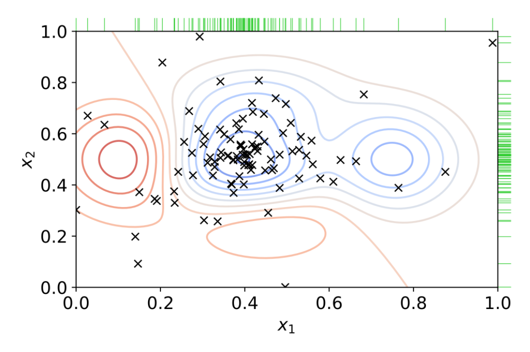
Pros

- Still easy to implement.
- Helps when dealing with large search spaces.

Cons

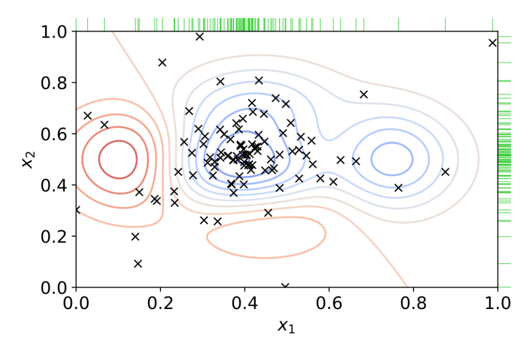
- Less reproducible.
- Not guaranteed to hit the optimum.
- Need to choose a range for the hyperparameters.

Bayesian optimization, builds a probability model of the objective function; it uses this model to select the most promising set of hyperparameters. At each iteration, it updates its underlying model, and repeats the process.



Bayesian optimization - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Bayesian optimization uses a statistical model to choose the next set of hyperparameters *smartly* based on previous results - source Wikipedia

Bayesian optimization, builds a probability model of the objective function; it uses this model to select the most promising set of hyperparameters. At each iteration, it updates its underlying model, and repeats the process.



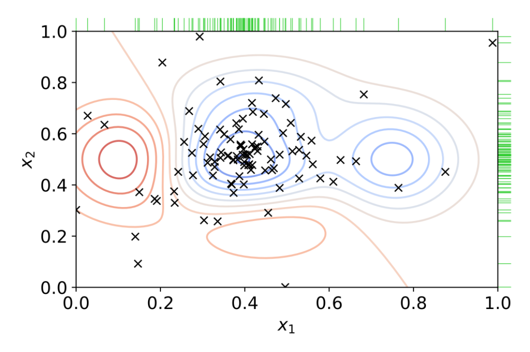
Bayesian optimization - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Bayesian optimization uses a statistical model to choose the next set of hyperparameters *smartly* based on previous results - source Wikipedia

Pros

- Efficient search of hyperparameter space.
- Performs better than grid and random search.

Bayesian optimization

Bayesian optimization, builds a probability model of the objective function; it uses this model to select the most promising set of hyperparameters. At each iteration, it updates its underlying model, and repeats the process.



Bayesian optimization - In this example we tune hyperparameters x_1 and x_2 in the range [0, 1]. Bayesian optimization uses a statistical model to choose the next set of hyperparameters *smartly* based on previous results - source Wikipedia

Pros

- Efficient search of hyperparameter space.
- Performs better than grid and random search.

Cons

- Can be very computationally expensive when using a lot of hyperparameters.
- Complex to implement.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum “budget” B (e.g. number of epochs, time,...) to evaluate these configurations.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum “budget” B (e.g. number of epochs, time,...) to evaluate these configurations.
- Each configuration takes up B/N of the budget.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum "budget" B (e.g. number of epochs, time,...) to evaluate these configurations.
- Each configuration takes up B/N of the budget.
- When the budget is exhausted, discard the worst-performing half of the configurations.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum “budget” B (e.g. number of epochs, time,...) to evaluate these configurations.
- Each configuration takes up B/N of the budget.
- When the budget is exhausted, discard the worst-performing half of the configurations.
- Continues until only one configuration remains.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum "budget" B (e.g. number of epochs, time,...) to evaluate these configurations.
- Each configuration takes up B/N of the budget.
- When the budget is exhausted, discard the worst-performing half of the configurations.
- Continues until only one configuration remains.

Early-stopping based methods - Successive Halving

Early-stopping methods work by early stopping of evaluations on combinations of parameters that are not promising.

Successive Halving algorithm

- Choose a random set of N hyperparameters configurations
- Set a maximum "budget" B (e.g. number of epochs, time,...) to evaluate these configurations.
- Each configuration takes up B/N of the budget.
- When the budget is exhausted, discard the worst-performing half of the configurations.
- Continues until only one configuration remains.

Problem: do we consider a small N with a large budget for each configuration, or a large N with a small budget?

Early-stopping based methods - Hyperband

Hyperband extends the Successive Halving algorithm by running multiple *brackets* of hyperparameters.

It performs a grid search on the number of configurations N that we are going to choose.

It can be 10-20x faster than Bayesian optimization.

KerasTuner

The **Keras Tuner** library (install via `pip install keras-tuner`) provides a simple way to tune hyperparameters in a Keras model.

It implements the random search, Bayesian optimization and Hyperband strategies.

In order to use Keras Tuner we need to have a function that generates and returns our Keras model. We will pass that function to the tuner class of our choice.

KerasTuner example

```
import keras
import keras_tuner as kt

# hp contains all the hyperparameters and is passed automatically by the tuner
def build_model(hp):
       model = keras.Sequential()
       # Create an integer hyperparameter, going from 10 to 100
       # We can also have hp.Float, hp.Choice, hp.Bool
       n_filters = hp.Int('filters', min_value=10, max_value=100, step=10)
       model.add(keras.layers.Conv2D(n_filters, (3, 3),
                 input_shape=(512, 512, 1)))
       ...
       model.compile(...)
       ...
       return model
```

KerasTuner example

```
rs = kt.tuners.RandomSearch(build_model,
       objective='val_accuracy',
       max_trials=10,
       directory="output_dir")

# Instead of calling model.fit we call tuner.search
rs.search(x_train, y_train,
       validation_data=(x_test, y_test),
       epochs=10, batch_size=32)

best_hyperparameter = rs.get_best_hyperparameters(1)[0]
best_model = rs.get_best_models(1)[0]
```

We can now use a model with the best configuration for our task!

Summary

- Hyperparameters are values that control the learning process of a model.
- Manual methods are time-consuming, require expertise, and are not guaranteed to find the optimal model.
- Automated methods such as grid search, random search, Bayesian optimization, and early-stopping based methods can help.
- KerasTuner is a library that can help us tune hyperparameters in a Keras model.