Tuning

configuration trackers, hyperband, schedulers

deep learning 4

Raoul Grouls, 7 oktober 2024

Manual search

Manual tuning is necessary to develop some intuition about complexity of models.

When encountering a problem, you will be able to make a reasonable guess:

- Have you already solved problems that are sort-of-similar?
- What are ranges of parameters you have seen working in similar complexity?
- How much time does it take / do you have to hypertune a problem?
- How do you spot learning (or a lack of it) in tensorboard?

Manual search

When encountering a problem, it helps to have reference points:

- This problem seems more complex than MNIST, but not as complex as the flower photos;

So let’s start with settings somewhere between what worked for those two.

However, *after* you have an intuition, we can get the help of algorithms.

A typical mistake of a beginner is to make the search space way to big.

Let’s say you consider amounts of:

- Linear units (between 16 and 256)
- Depth of linear layers (between 1 and 10)
- Convolution filters (between 16 and 64)
- kernel size (between 3 and 5)
- dropout (4 options)
- depth of convolutions (between 1 and 5)
- learning rate (4 options)
- optimizers (3 options)
- activation functions (3 options)
- batchsize (between 16 and 128)

Can you calculate how big the search space is?

The curse of dimensionality

This search space has almost 2.7×10^{10} options. This is roughly the amount of sand when you fill your $40m^2$ living room with 50 cm of sand.

If you test at random 50 options, you are just scanning a very small fraction of all options.

You shouldn’t be surprised if the result of random hypertuning is worse than the result of your manual hypertuning!

The curse of dimensionality

Things behave different in high-dimensional space.

A. Pick a random point in a d-dimensional unit square/hypercube. Let the distance from the border be δ.
    1. For $d = 2$ (square), $P(\delta < 0.001) = 0.004$
    2. For $d = 10{,}000$, $P(\delta < 0.001) = 0.99999999$

B. Pick two points randomly in a d-dimensional unit-square/cube/hypercube.
    1. For $d = 2$ (square), expected distance is 0.52
    2. For $d = 3$ (cube), expected distance is 0.66
    3. For $d = 1{,}000{,}000$, expected distance is 408.25


searchspace too big


Tool overkill


Your tool works by accident

Keeping track

There are a lot of ways to keep track of your experiments.

- **Gin-Config**: A simple configuration library for Python that uses dependency injection to simplify the process of configuring methods, see https://github.com/google/gin-config
- **TensorBoard**: A visualization toolkit for tracking and visualizing metrics such as loss and accuracy, view the architecture of your model, and visualize embeddings. An unscalable but common hack is to add parameters to the name of the folder., see https://www.tensorflow.org/tensorboard
- **MLflow**: An open-source platform for managing the end-to-end machine learning lifecycle, including experimentation, reproducibility, and deployment, see https://mlflow.org/
- **Ray**: A framework-agnostic, high-performance distributed computing library that provides a simple, universal API for building distributed applications. The ‘tune’ submodule has extensive options for distributed hyperparametertuning experiments, see https://docs.ray.io/en/latest/tune/index.html

Hypertuning

While hypertuning manually is a great way to

1. obtain intuitions and gutfeelings

of what works, we need to move to algorithmic hypertuning when

2. the searchspace is getting too complex for gutfeelings

You should learn to cooperate with the hypertuning algorithms, and understand which part they can take over and which part you should (learn) to do.

Tuning frameworks

Algorithms for tuning algorithms

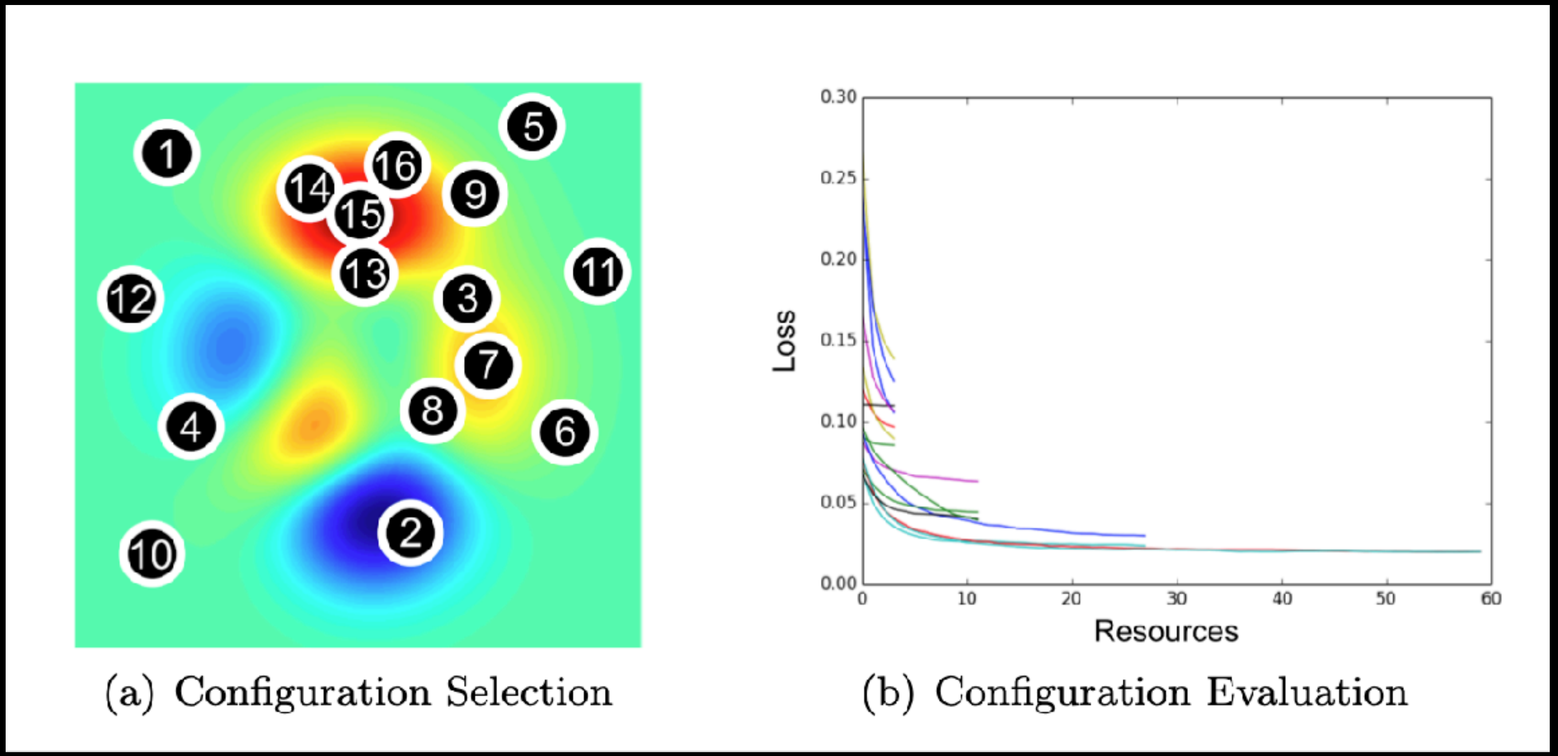
- we will use Ray tune
- HyperOpt is another alternative
- You want a framework that is agnostic to ML frameworks; this way you dont have to switch if your framework changes.
- Ray offers parallel computing, also for training, data preprocessing, etc.

	HyperOpt	Optuna	Keras-Tuner	HpBandSter	Tune
Distributed Execution	✅	✅	☐	✅	✅
Fault Tolerance	☐	☐	☐	☐	✅
Search algorithms	2	4	3	3	6
Framework Support (TF/Keras, Sklearn, PyTorch)	✅	✅	☐	☐	✅

Baysian Hyperband

Some nice solutions are:

- bayesian search (a)
- hyperband (b)
- Bayesian Optimization and Hyperband (BOHB) combines bayesian & hyperband techniques.
- Tree Parzen Estimators (bayesian optimization over awkward search spaces with real, discrete and conditional parameters)



(a) Configuration Selection

(b) Configuration Evaluation

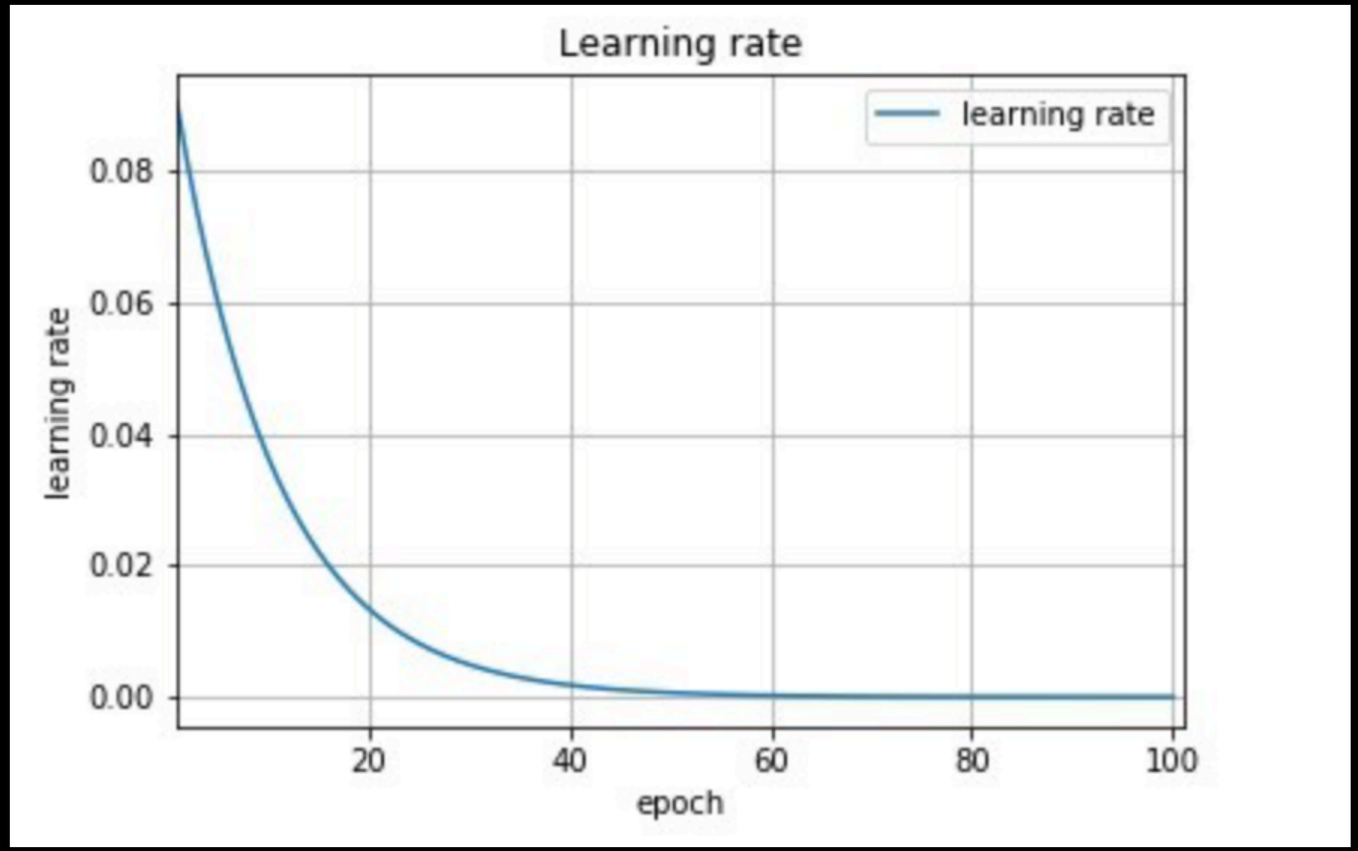
Learning rate schedulers

The learning rate η tells how fast weights are adjusted

$$W_{t+1} \leftarrow W_t - \eta \times gradient$$

The learning rate can be compared to different sizes of tools to carve wood. Big tools go fast, but miss the details.

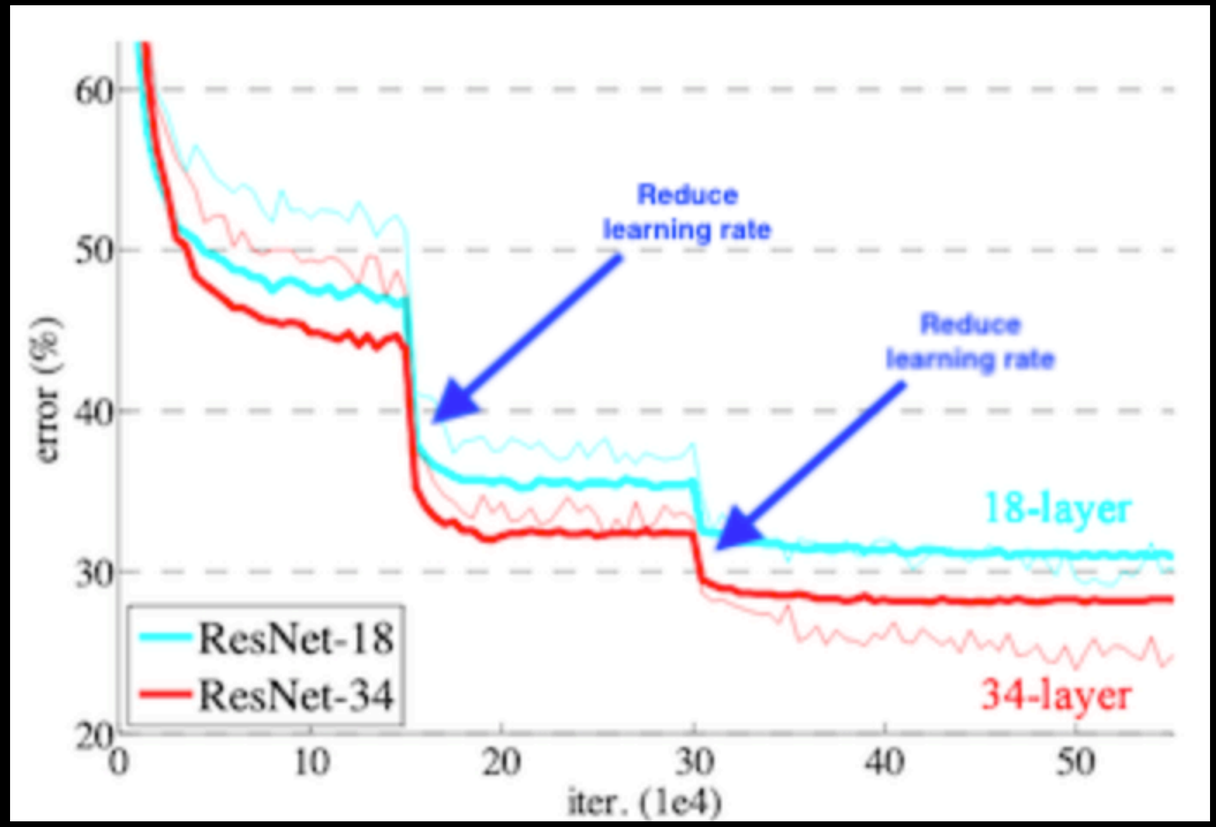
So, often a decay of the learning rate is a good idea.



Learning rate schedulers

ResNet can be trained for very long. From the paper:

> *"The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to* 60×10^4 *iterations"*



Learning rate schedulers

The first iterations can have very high gradients, and could “lock” the model in the wrong direction.

A solution is to gradually increase the learning rate from 0 on to the originally specified learning rate in the first few iterations.

This is typically used with transformer architectures.