Tuning

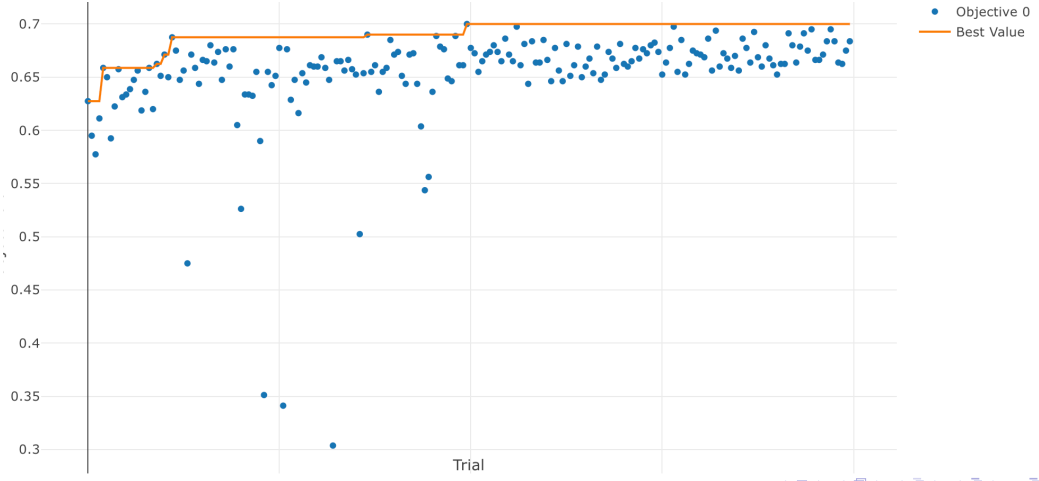
Nasy

Mar 24, 2023

Outline

Introduction

Target: Maximizing the performance of the model.
Question? How to get good results with deep learning?
Why? My model does not work!!!



Get Start

- Model Architecture
- Optimizer
- Batch Size
- Hyperparameters
  - Initial Learning rate, momentum, weight decay, etc.
- ref[1]

[1] *Deep Learning Tuning Playbook*. Google Research, Mar. 24, 2023. URL: https://github.com/google-research/tuning_playbook (visited on 03/24/2023).

Model Architecture

- Try to reuse the existing models that already work.
- Not only the model layers, but also the hyperparameters settings.

Optimizer

No optimizer is the "best" across all types of machine learning problems and model architectures.[2]

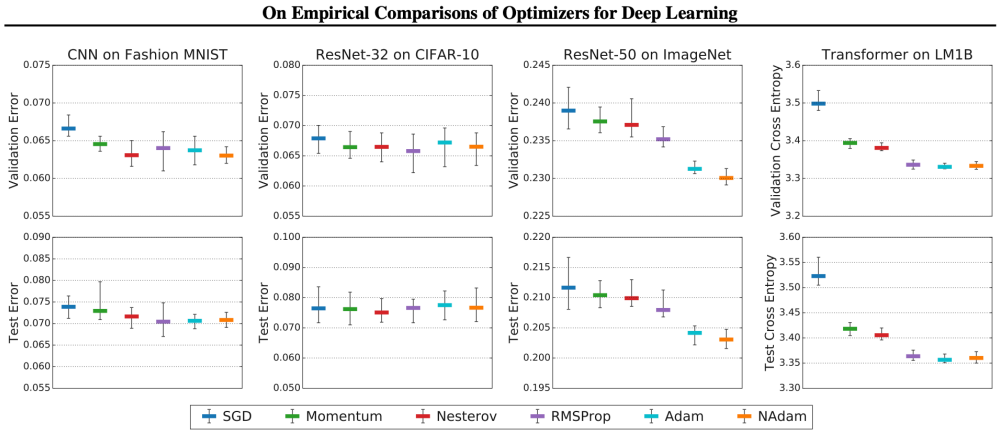


Figure 1: The relative performance of optimizers is consistent with the inclusion relationships, regardless of whether we compare final validation error (top) or test error (bottom). For all workloads, we tuned the hyperparameters of each optimizer separately, and selected the trial that achieved the lowest final validation error. Optimizers appear in the same order as the legend in all plots in this paper.

Batch Size

- Only affects the speed of training.
- The batch size should not be treated as a tunable hyperparameter.[3]
- In most case, batch size should be the largest batch size supported by the available hardware.
- Do not change during tuning.
  - Some layers, like the Batchnorm, will be affected.

[3] Christopher J. Shallue et al. *Measuring the Effects of Data Parallelism on Neural Network Training*. July 18, 2019. DOI: 10.48550/arXiv.1811.03600. arXiv: arXiv:1811.03600. URL: http://arxiv.org/abs/1811.03600 (visited on 03/23/2023). preprint.

Initial Hyperparameters

Before we start tuning hyperparameters:

- Model ready (e.g. num of layers).
- Max traning steps
- Pipeline ready (e.g. preprocessing, evaluation.)

Select the initial hyperparameters:

- Learning rate (start from fix instead of schedule)
- Optimizer parameters
  - Adam: lr, beta1, beta2, weight_decay
  - SGD: lr, momentum, weight_decay
- Layers parameters
  - Batchnorm: momentum, epsilon
  - Dropout: rate
  - Conv2D: kernel_size, strides, padding, activation
  - Dense: activation, output features
  - leaky_relu: alpha
- Search spaces

Hyperparameters

Scientific hyperparameters For ablation study. (e.g. num of GCN layers)

Nuisance hyperparameters Optimized for fair comparison for scientific hyperparameters. (e.g. optimizer parameters)

Fixed hyperparameters No need to change when comparing scientific hyperparameters. (e.g. batch size)

Strategies

- Grid search
  - Small search spaces
- Random search
  - Explore the search space
- Quasi-random search
  - Explore the search space (like the boundary of search space)
- Bayesian optimization
  - Exploit the correlation between hyperparameters.
  - TPE
  - Gaussian process
  - ...

Tuning Tools

Optuna https://github.com/optuna/optuna

NNI https://github.com/microsoft/nni

Vizier https://github.com/google/vizier

hyperopt (outdated) https://github.com/hyperopt/hyperopt.git

advisor (outdated) https://github.com/tobegit3hub/advisor.git

Bayesian Optimization

We have a set of hyperparameters $X = x_1, x_2, ..., x_n$, and we want to find the best one for function $f : x \to R$, where $x \in X$:

$$x^* = \arg\max_{x \in X} f(x)$$

Algorithm

Input f, X, S, M

- f The blackbox function we want to optimize.
- X The hyperparameters we want to tune.
- S Acquisition function
- M Model of BO. (Gaussian process; TPE; Random forest; …)

```
def algorithm(f, X, S, M):
    D = init_samples()  # initial x and y = f(x)
    for i in range(100):
        p = M.fit(D)  # p(y|x, D)
        new_x = S(X, p)
        new_y = f(new_x)
        D.append((new_x, new_y))
```

Acquisition Function

- An inexpensive function that can be evaluated at a given point that is commensurate with how desirable evaluating f at x is expected to be for the problem
- Probability of improvement
- Expected improvement
- Entropy search
- Gaussian Process-Upper Confidence Bound

Refs I

[1] Dami Choi et al. *On Empirical Comparisons of Optimizers for Deep Learning*. June 15, 2020. DOI: 10.48550/arXiv.1910.05446. arXiv: arXiv:1910.05446. URL: http://arxiv.org/abs/1910.05446 (visited on 03/23/2023). preprint.

[2] *Deep Learning Tuning Playbook*. Google Research, Mar. 24, 2023. URL: https://github.com/google-research/tuning_playbook (visited on 03/24/2023).

[3] Christopher J. Shallue et al. *Measuring the Effects of Data Parallelism on Neural Network Training*. July 18, 2019. DOI: 10.48550/arXiv.1811.03600. arXiv: arXiv:1811.03600. URL: http://arxiv.org/abs/1811.03600 (visited on 03/23/2023). preprint.