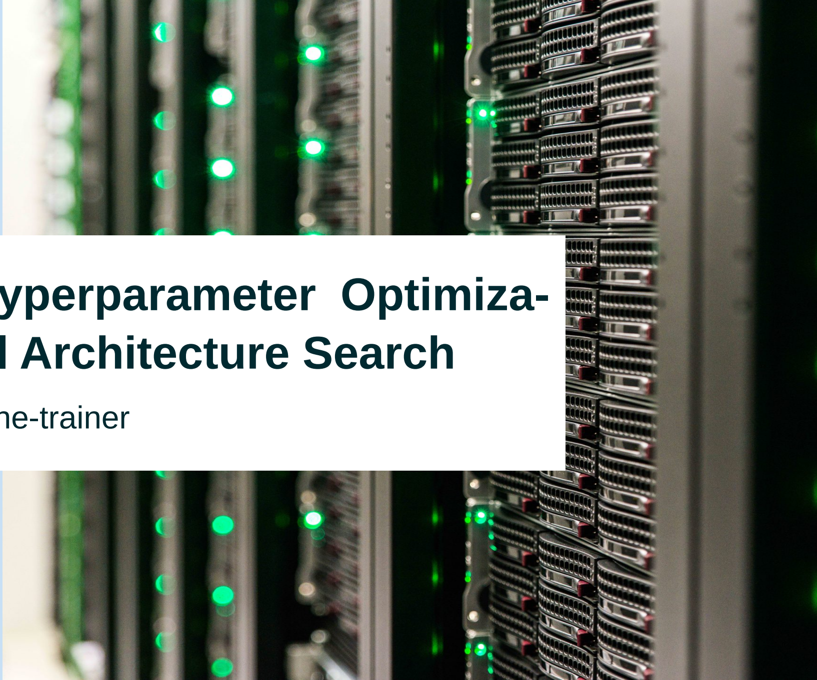
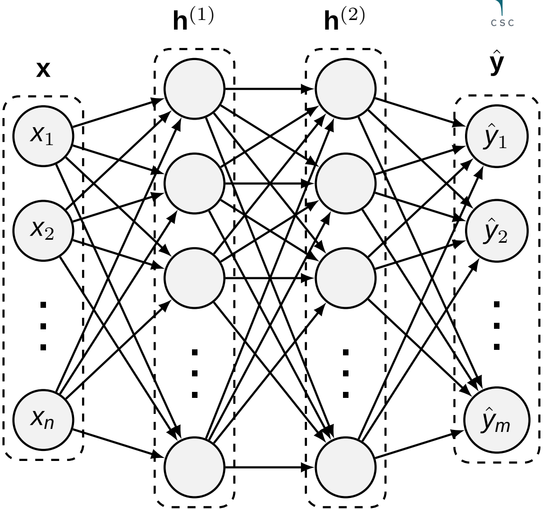


AutoML, Hyperparameter Optimization, Neural Architecture Search

Multi-GPU train-the-trainer

Motivation: Choices

Number of layers, types of layers, layer parameters, optimization algorithm, normalization, regularization parameters, dataset parameters, computation and scaling parameters...

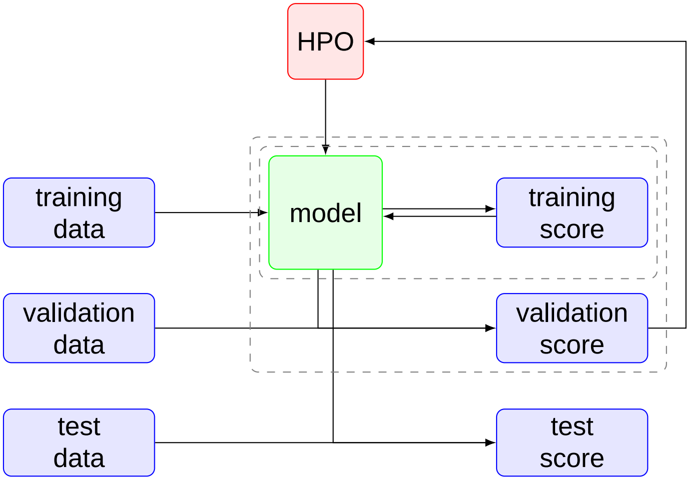


github.com/fraserlove/nntikz

Outline

- Blackbox search algorithms
- Tools and implementations
- Scaling and efficiency
- More resources

Loop Nesting

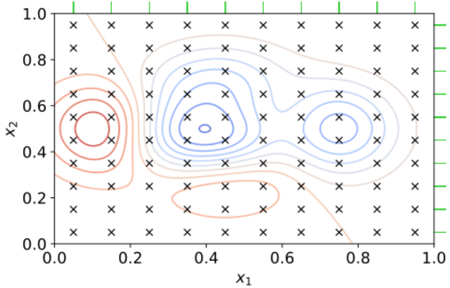


Parameter Space

- Continuous
    - Learning rate
    - Dropout rate
- Integer
    - Batch size
    - Number of layers
- Ordinal
- Categorical
    - Optimizer
    - Activation function

Naive Approaches

- Manual exploration
- Grid search
- Random search



CC Alexander Elvers

Exercise 1: Random Search

- Script for a grid search is given
- Adapt it to perform a random search

Exercise 1: Random Search

```bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --gpus-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --time=0-00:10:00
#SBATCH --partition=boost_usr_prod
#SBATCH --account=tra26_castiel2
#SBATCH --array=0-7

module purge
module load profile/deeplrn cineca-ai/4.3.0
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK


srun python train_mnist.py -r ${SLURM_ARRAY_TASK_ID}
```

Exploitation

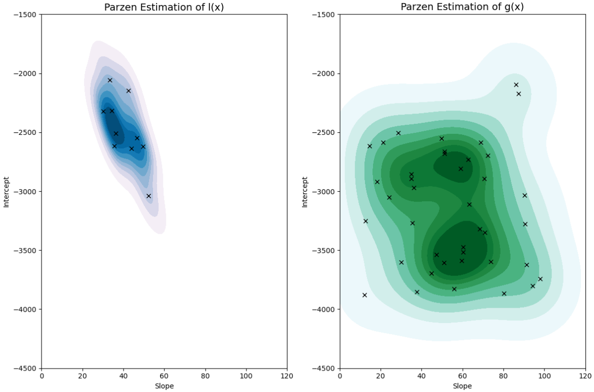
- Evolutionary strategies
  - Genetic optimization
  - Covariance matrix adaptaion - ES
- Bayesian optimization
  - Gaussian processes
  - Tree of Parzen estimators
- Particle swarms
- Nelder-Mead method
- …

Blackbox Optimization Tools

- Optuna
- Ray Tune
- Propulate
- DeepHyper
- SMAC3
- Nevergrad
- …

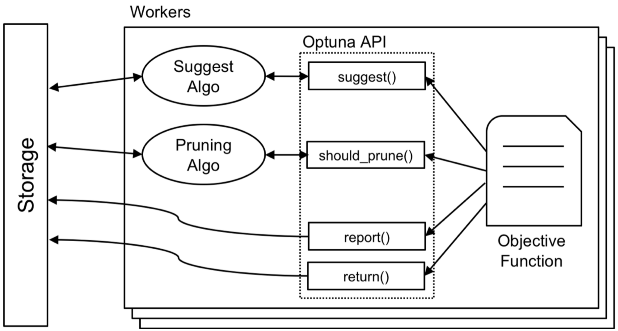
Bayesian Optimization with TPE

- Sample with prior
- Approximate posterior
- Derive acquistion function
- Optimize acquistion function



https://towardsdatascience.com/building-a-tree-structured-parzen-estimator-from-scratch-kind-of-20ed31770478/

Optuna



[0] Akiba et al. 2019

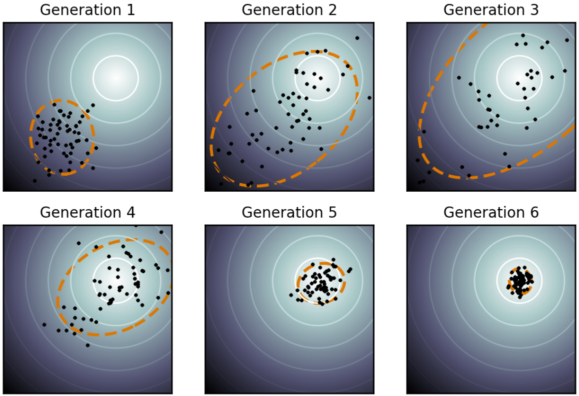
Exercise 2: Optuna

```
import optuna

def objective(trial):
    i = trial.suggest_int(name, min, max)
    f = trial.suggest_float(name, min, max)
    c = trial.suggest_categorical(name, [choice1, choice2, ...])

    # test suggested parameter values

study = optuna.create_study(name,
                            storage(JournalFileBackend(file_path=file_path)),
                                load_if_exists=True,
                            )

study.optimize(objective, n_trals=n_trials)
```

More Optuna Features

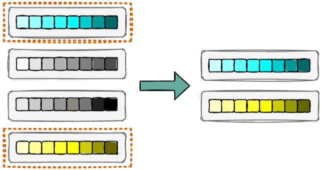
- Pruning
  - Successive halving
  - Hyperband
- Visualization
  - Dashboard
- Algorithms
  - CMA-ES
  - …

CMA-ES


Generation 1
Generation 2
Generation 3
Generation 4
Generation 5
Generation 6

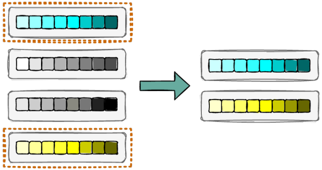
Genetic Optimization

Selection

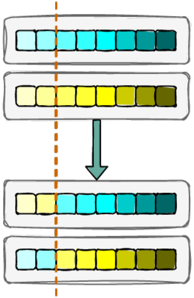


Genetic Optimization

Selection

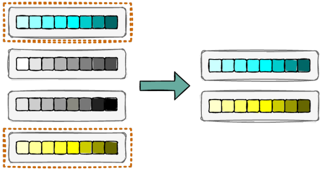


Crossover

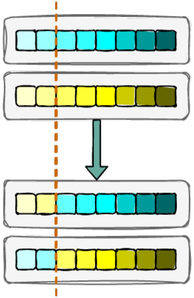


Genetic Optimization

Selection



Crossover



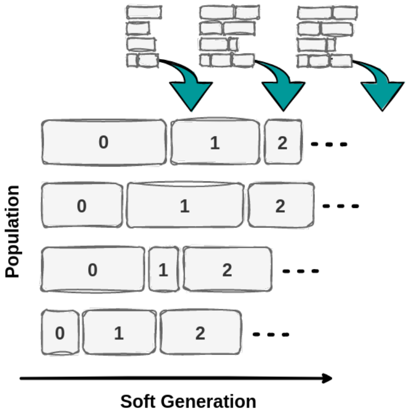
Mutation

Propulate Features

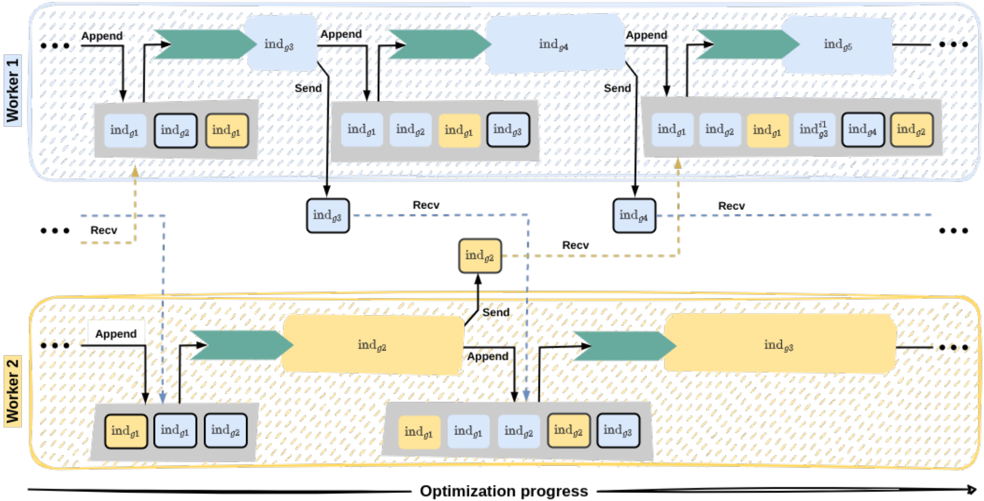
- HPC focused
- Decentralized
- Lazily synchronized

Propulate

- Breed and evaluate **asynchronously** using **continuous** population of all individuals evaluated so far.
- Maximize use efficiency by independent workers not waiting for each other!



Propulate



Worker 1
Worker 2
Append
Send
Recv
ind$_{g1}$
ind$_{g2}$
ind$_{g3}$
ind$_{g4}$
ind$_{g5}$
ind$_{g3}^{i1}$
Optimization progress

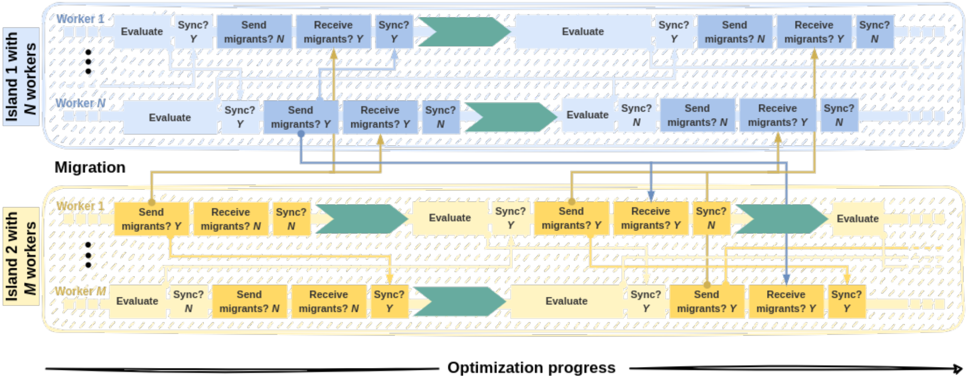
Exercise 3: Propulate

```
from propulate import Propulator
from propulate.utils import get_default_propagator

def ind_loss(params):
    name = params[name]
    # train and compute loss
    return -accuracy

limits = {name: (min, max), # int/float is inferred from type
          name: [val1, val2, val2]
          }
propagator = get_default_propagator(pop_size, limits, rng)
propulator = Propulator(ind_loss, propagator, rng, generations=10,
                        checkpoint_path)
propulator.propulate()
```

More Propulate Features

- Islands models
- Multi-rank workers for mutli-GPU training
- Pruning
- More algorithms

Propulate: Islands



Propulate: Multi-Rank Workers

```
islands = Islands(
    ...
    ranks_per_worker=2
    )

# Loss function has to accept an MPI communicator
def ind_loss(params, comm):
    # init process group for DDP

    # wrap model and dataloaders in torch DDP
    ...
    return val_loss
```

Next Steps

- Reinforcement learning
- Meta learning
- NASWOT