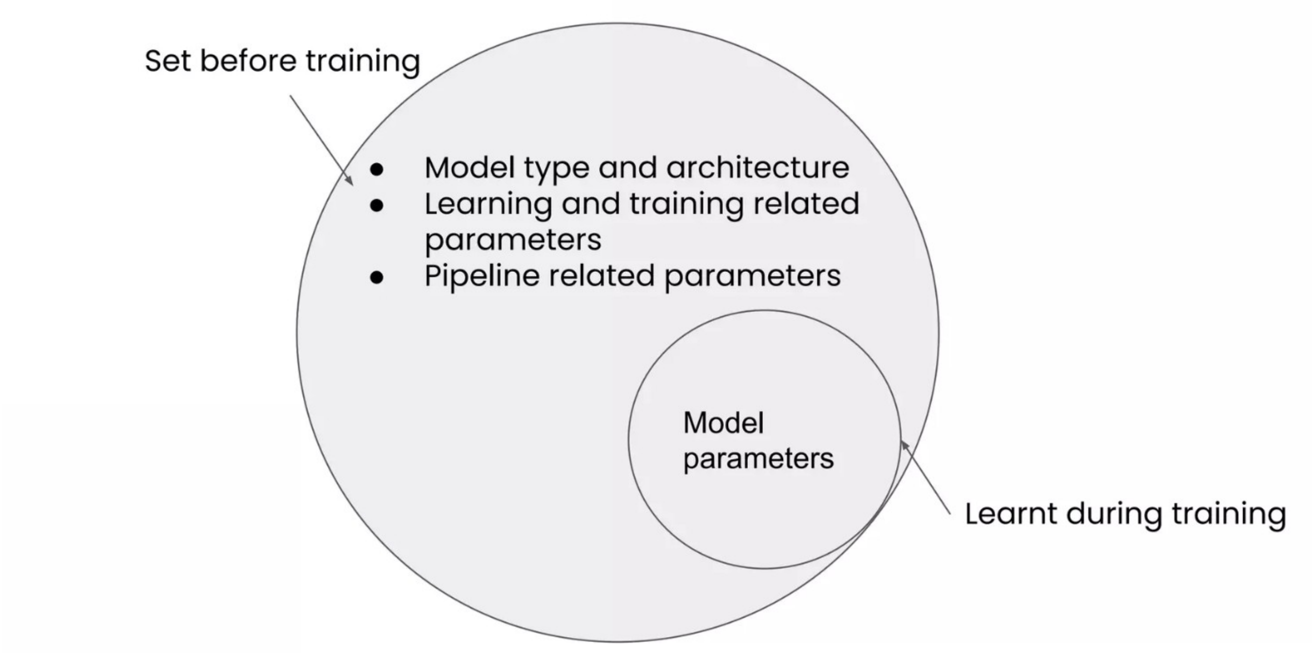
Hyper-parameter tuning using Ray on LUMI

Gregor Decristoforo – LUMI User Support Team
Norwegian research infrastructure services (NRIS) – UiT, Norway

What are hyper-parameters?



Source: www.slideshare.net/slideshow/cutting-edge-hyperparameter-tuning-made-simple-with-ray-tune/250862262

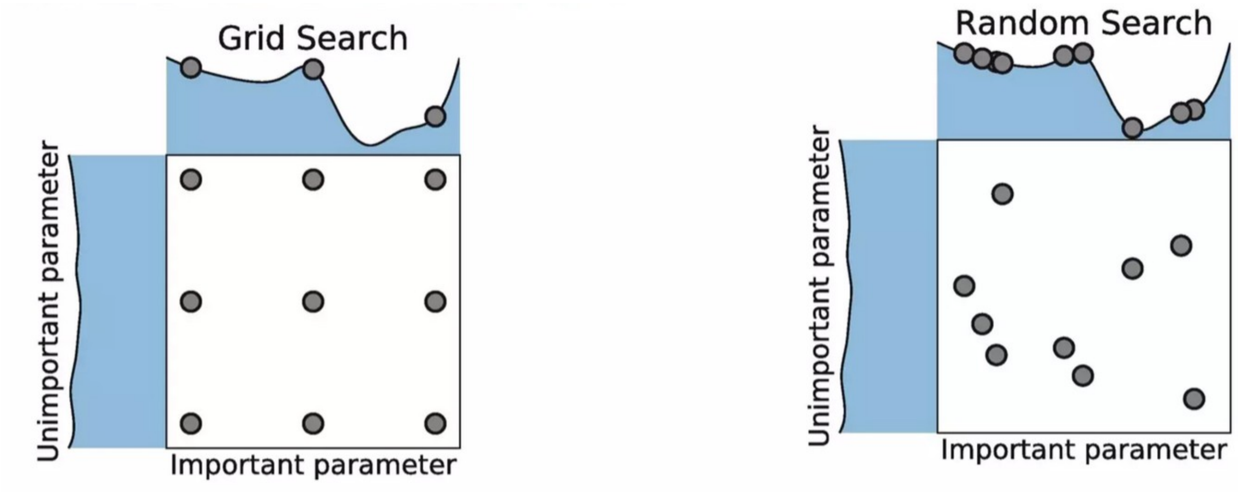
Hyper-parameter optimization (HPO) is expensive

- HPO is the trail and error process of finding the optimal set of hyper-parameters for a machine learning task
- Search space is typically non-liniar, convex and high-dimensional
- Every evaluation / trial involves model training

`Ray tune` makes HPO easier

Source: speakerdeck.com/richardliaw/a-modern-guide-to-hyperparameter-optimization?slide=23

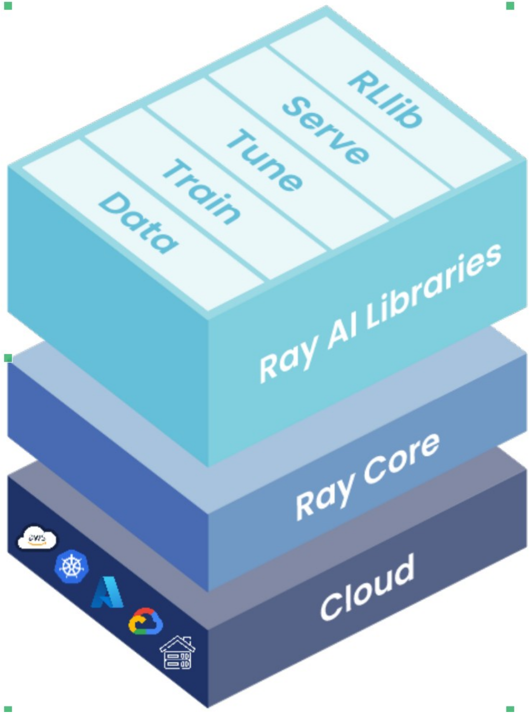
`Ray tune` provides wide range of HPO algorithms



Source: www.slideshare.net/slideshow/cutting-edge-hyperparameter-tuning-made-simple-with-ray-tune/250862262

More advanced algorithms included for Bayesian optimization, early stopping (HyperBand, ASHA), Population-based training, etc.

Ray framework



Source: docs.ray.io/en/latest/ray-overview/index.html

Ray consists of three layers:

1) **Ray AI Libraries:** high-level libraries that enable simple scaling of AI workloads

2) **Ray Core**: a low-level distributed computing framework with a concise core and Python-first API

3) **Ray Cluster**: A set of worker nodes connected to a common Ray head node

Installing / Using `Ray` on LUMI

Multiple options available:

- Use existing container that has `Ray` included:

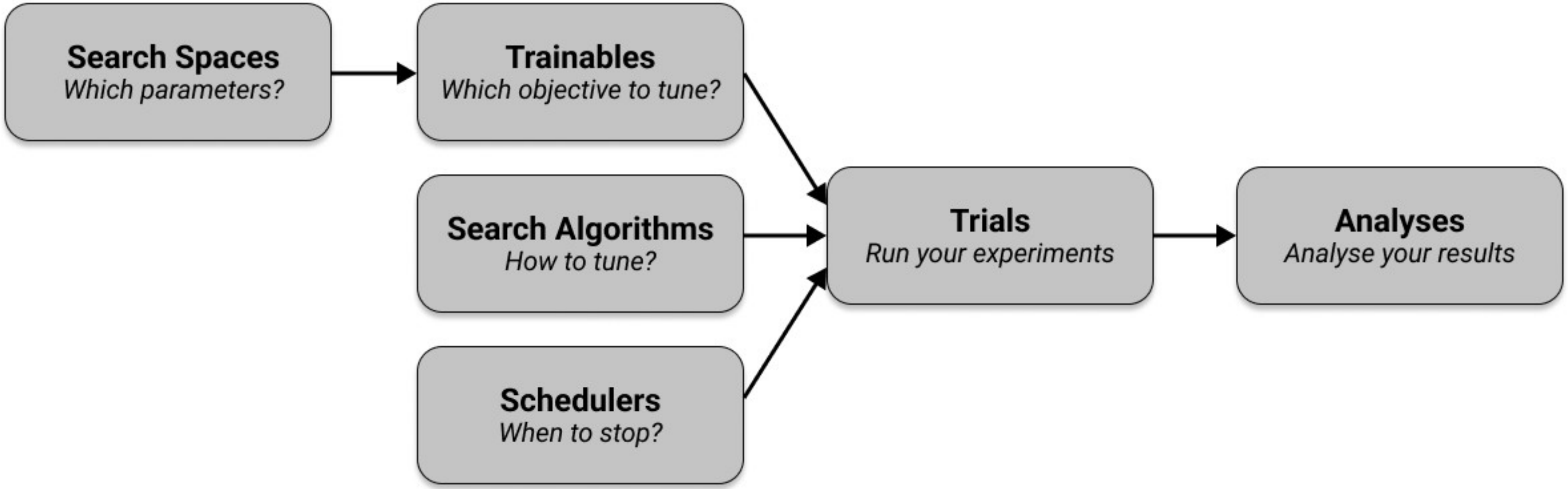
  ```
  CONTAINER=/scratch/project_465001063/containers/pytorch_transformers.sif
  ```

- Add `ray-tune` to conda environment file and create a container with `cotainr` (see lecture *Converting your conda/pip AI environment to a container using cotainr*)

- Extend existing container via a virtual environment and install `Ray` (see lecture *Extending containers with virtual environments for faster testing*):

  ```
  conda install -c conda-forge "ray-tune" / pip install ray[tune]
  ```

Key Concepts of `Ray Tune`



Tune Search Spaces

Tune offers various functions to define search spaces and sampling methods.

```
config = {
    "uniform": tune.uniform(-5, -1),  # Uniform float between -5 and -1
    "quniform": tune.quniform(3.2, 5.4, 0.2),  # Round to multiples of 0.2
    "loguniform": tune.loguniform(1e-4, 1e-1),  # Uniform float in log space
    "qloguniform": tune.qloguniform(1e-4, 1e-1, 5e-5),  # Round to multiples of 0.00005
    "randn": tune.randn(10, 2),  # Normal distribution with mean 10 and sd 2
    "qrandn": tune.qrandn(10, 2, 0.2),  # Round to multiples of 0.2
    "randint": tune.randint(-9, 15),  # Random integer between -9 and 15
    "qrandint": tune.qrandint(-21, 12, 3),  # Round to multiples of 3 (includes 12)
    "lograndint": tune.lograndint(1, 10),  # Random integer in log space
    "qlograndint": tune.qlograndint(1, 10, 2),  # Round to multiples of 2
    "choice": tune.choice(["a", "b", "c"]),  # Choose one of these options uniformly
    "func": tune.sample_from(
        lambda spec: spec.config.uniform * 0.01
    ),  # Depends on other value
    "grid": tune.grid_search([32, 64, 128]),  # Search over all these values
}
```

Trainables

Create a function (trainable) that takes in a dictionary of hyper-parameters. This function computes a score and reports it back to Tune.

```
from ray import train


def objective(x, a, b):  # Define an objective function.
    return a * (x**0.5) + b


def trainable(config):  # Pass a "config" dictionary into your trainable.

    for x in range(20):  # "Train" for 20 iterations and compute intermediate scores.
        score = objective(x, config["a"], config["b"])

        train.report({"score": score})  # Send the score to Tune.
```

Tune Trials

To execute and manage hyper-parameter tuning, generate trials with tuner.fit().

```
space = {"a": tune.uniform(0, 1), "b": tune.uniform(0, 1)}

tuner = tune.Tuner(
    trainable, param_space=space, tune_config=tune.TuneConfig(num_samples=10)
)

tuner.fit()
```

Example: perform HPO for `pt-imdb-model`

We perform hyper-parameter tuning for the learning rate for the pt-imdb-model from lecture "*Your first AI training job on LUMI*"

The goal is to test different learning rates utilizing all GPUs on one LUMI-G node simultaneously

Find code and instructions at:
`github.com/Lumi-supercomputer/Getting_Started_with_AI_workshop/tree/main/09_Hyper-parameter_tuning_using_Ray_on_LUMI`

Define trainable for `pt-imdb-model`

```python
def model_training(config):

    args = config["args"]
    learning_rate = config["learning_rate"]

    ...
    # train pt-imdb-model
    ...

    trainer.train(resume_from_checkpoint=args.resume)

    # report results back to ray
    eval_results = trainer.evaluate()
    train.report(
        dict(
            loss=eval_results["eval_loss"],
            perplexity=math.exp(eval_results["eval_loss"]),
        )
    )
```

Initialize Ray with correct resources

Slurm parameters are not automatically passed on to Ray

```
# We need to manually set the number of CPUs and GPUs.
# Othewise, ray tries to use the whole node and crashes.

ray.init(num_cpus=, num_gpus=, log_to_driver=False)
```

Start tuning process

LUMI

```
# Create a Tuner object
tuner = Tuner(
    tune.with_resources(
        model_training, resources={"cpu": , "gpu": }  # Set resources for every trial run
    ),
    param_space=config,
    tune_config=tune.TuneConfig(
        num_samples=8,  # Number of samples
        metric="perplexity",  # Metric to optimize
        mode="min",  # Minimize the metric
    ),

)

# Run the tuning process
results = tuner.fit()
```

Desired output

Trial name	status	learning_rate	iter	total time (s)	loss	perplexity
model_training_d4346_00000	TERMINATED	0.00063136	1	365.562	6.53819	691.033
model_training_d4346_00001	TERMINATED	0.000553059	1	363.708	5.21202	183.465
model_training_d4346_00002	TERMINATED	0.000322119	1	365.409	3.47681	32.3564
model_training_d4346_00003	TERMINATED	0.000317334	1	298.858	3.47216	32.2063
model_training_d4346_00004	TERMINATED	0.000819981	1	365.591	6.17962	482.81
model_training_d4346_00005	TERMINATED	0.000502913	1	365.225	6.45786	637.696
model_training_d4346_00006	TERMINATED	0.000825948	1	298.899	6.07116	433.182
model_training_d4346_00007	TERMINATED	0.000158792	1	365.383	3.33857	28.1787

Outlook: running `Ray` on multiple nodes on LUMI

- SLURM support for `RAY` is community-maintained and still a work in progress
- Requires manual setup of `Ray` head node and worker nodes
- Guide on documentation:
  docs.ray.io/en/latest/cluster/vms/user-guides/community/slurm.html
- Please contact us if you would like more LUMI-specific guides on Ray-related topics