Karlsruher Institut für Technologie

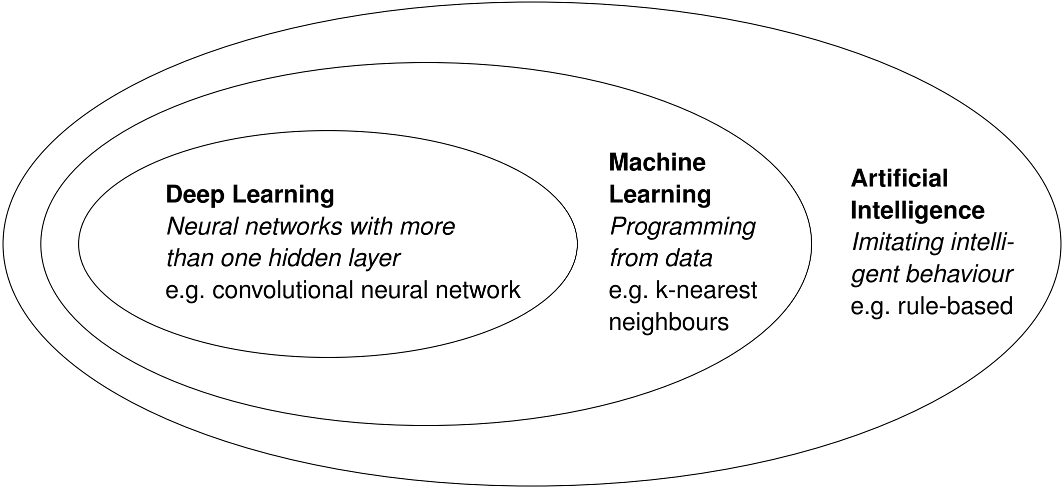
Steinbuch Centre for Computing

Scalable Artificial Intelligence

KSETA Course

Dr. Markus Götz | October 5th, 2023

Artificial Intelligence



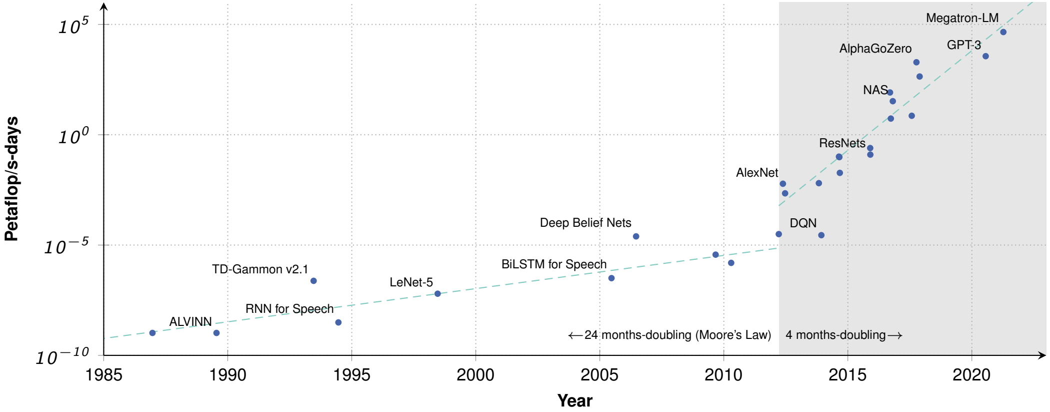
Artificial Intelligence

Karlsruher Institut für Technologie

1. **Technology revolution** – parallel processors (e.g. GPUs), auto-gradient software
2. **Data availability** – large-scale, publicly available and labeled data
3. **Methodological advances** – new neuron types, training approaches, embeddings

AI and Scaling Trends

“We’re very uncertain about the future of compute usage in AI systems, but it’s difficult to be confident that the recent trend of rapid increase in compute usage will stop, and we see many reasons that the trend could continue.” - Ilya Sutskever, OpenAI



Scalable AI: Large Models

Karlsruher Institut für Technologie

Generative Pre-Trained Transformer 3 (GPT-3) [1]

- Auto-regressive **Large language Model** (LLM)
- Imitation of human-made texts
- **175 million** trainable parameters
- Trained with **45 TB texts**

thoughts

(artificial) random musings of a machine

"On Friday, we abandon all we have learned this week."

- ai generated tweet by openai's gpt3

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

https://lacker.io/ai/2020/07/06/giving-gpt-3-a-turing-test.html

Scalable AI: Long Training

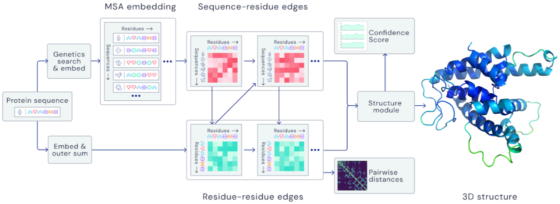
OpenAI Five [2]

- Team of five bots for **DOTA2** game
- Training: **10 month** auf verteiltem System
- 2018: **180 year** play experience
- 2019: Victory against **Team OG** (champions)
- Trained on **256 GPUs** and **128.000 CPUs**

https://openai.com/blog/openai-five/

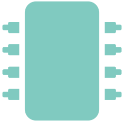
AlphaFold 2 [3]

- Prediction of **protein** tertiary structure from primary amino acid sequences
- **10 million iterations** for convergence
- Training: 3 weeks with **128 TPUs v3**



Jumper, John, et al. "Highly accurate protein structure prediction with AlphaFold."

Scalable AI: Bottlenecks



Compute Time

- Sequential computation on a single device takes too long
- Distributed algorithm on multiple nodes to reduce training time

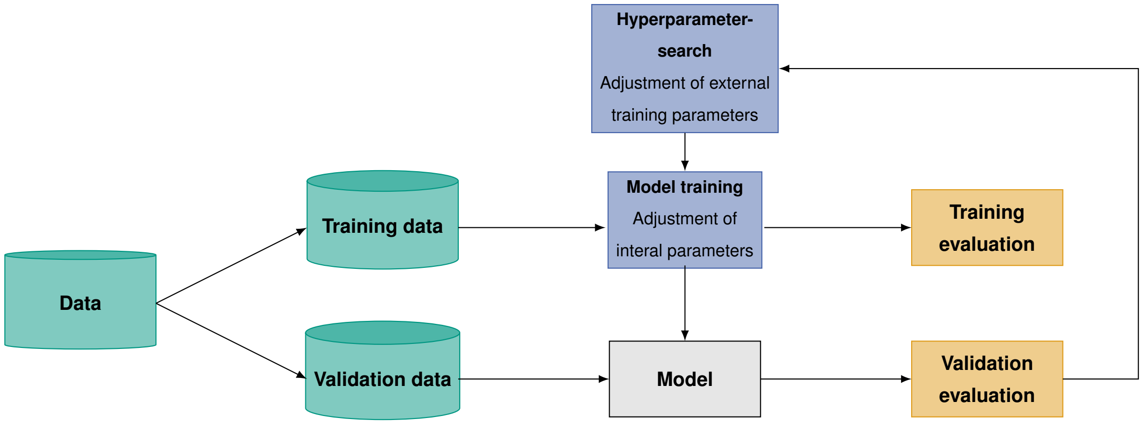
→ **Acceleration**

Memory

- Data is too large for a single computational node
- Computational speed is not focus
- Distributed algorithm to ingest data in chunks

→ **Enabling**

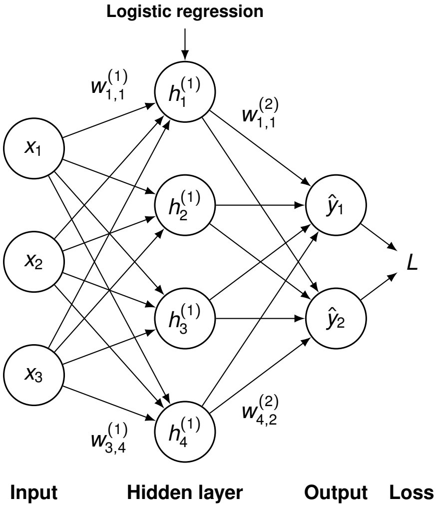
Modeling Process



Artificial Neural Networks

- Neural networks imitate biological behaviors [4].
  - **Neurons**: **smallest processing unit**
  - Graph of arithmetic operations
- **Weights** W: neuron connections, **free parameters** of the network
- Mathematical notation
  - $w_{i,j}^{(l)}$ – weight of input i wrt. neuron j, layer l
  - $a^{(l)}$ – activation function in layer l
  - $n_i^{(l)}$ – neural activation in layer l and neuron i
  - $h_j^{(l)}$ – hidden layer l, neuron j

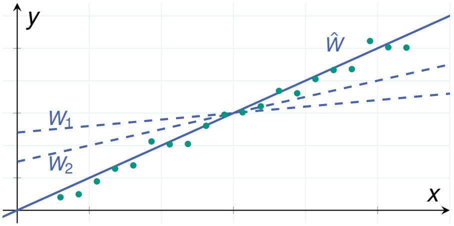
$$h_j^{(l)} = a^{(l)}\left(n_i^{(l)}\right) = a^{(l)}\left(\Sigma_{i=1}^{n} w_{i,j}^{(l)} \cdot x_i\right)$$



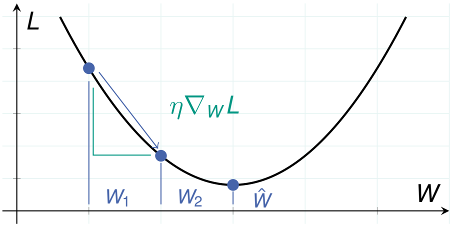
Determining W – Gradient Descent

- **Iterative approach** to determine W

$$W_{i+1} = W_i - \eta \nabla_W L$$



- **Random initial state**, e.g. $W_0 \neq 0$
- η is step size, called **learning rate**.



- Extensions and variants
  - **Standard** – gradient descent after every sample, batch size $B = 1$
  - **Stochastic** – randomized sample, $B = 1$
  - **Batch** – all samples, $B = |X|$
  - **Mini-Batch** – sample subset, $1 \leq B \leq |X|$

Backpropagation

- **Algorithms** for calculating **gradients**

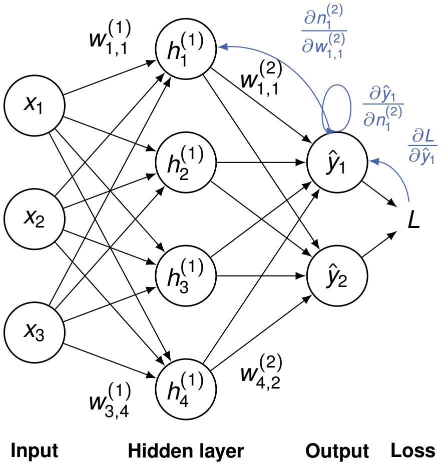
- Idea 1: **Divide** into **subproblems** – every weight with partial gradient.

$$\nabla_w L = \left(\frac{\partial L}{\partial w_{1,1}^{(1)}}, \frac{\partial L}{\partial w_{1,2}^{(1)}}, \ldots, \frac{\partial L}{\partial w_{i,j}^{(l)}} \right)$$

- Idea 2: “denesting” of neurons via **chain rule**

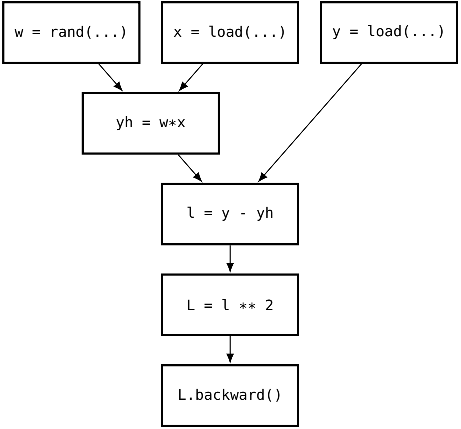
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

- Solution from output to input



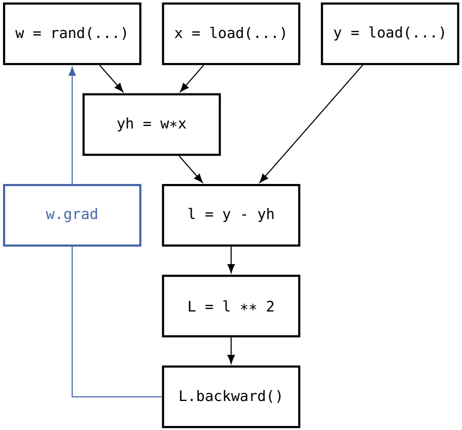
Automatic Differentiation (AD)

- Practical application of backpropagation
  - Efficient, since it only propagates partial derivatives
  - Multiple weights in a layer as matrices
- Backpropagation by hand is prone to errors.
- **Automatic Differentiation (AD)**: technique to generate derivatives in a program.
  - Atomic operations ($+$, $-$, $*$, $/$) and certain functions (sin, exp, max) with explicit gradients.
  - Combination via chain rule.
  - Common implementations: `TensorFlow` [5], `PyTorch` [6],...



Automatic Differentiation (AD)

- Practical application of backpropagation
  - Efficient, since it only propagates partial derivatives
  - Multiple weights in a layer as matrices
- Backpropagation by hand is prone to errors.
- **Automatic Differentiation (AD)**: technique to generate derivatives in a program.
  - Atomic operations $(+, -, *, /)$ and certain functions $(\sin, \exp, \max)$ with explicit gradients.
  - Combination via chain rule.
  - Common implementations: `TensorFlow` [5], `PyTorch` [6],...



How to Train a Neural Network in PyTorch

1. **Data**
   - Managed via classes in `torch.utils.data`
   - PyTorch's `DataSet` handles data indexing
   - `DataLoader` responsible for reading strategy
   - Common datasets in `torchvision`

```python
import torch
import torchvision

trainset = torchvision.datasets.CIFAR10(
    root='./data',
    train=True,download=True
)
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=2
)
```

How to Train a Neural Network in `PyTorch`

2. Model

- Defines neural network architecture
- Different layer types in `torch.nn`
- `__init__` and `forward` user-provided

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(16 * 5 * 5, 120)
        self.out = nn.Linear(120, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc(x))
        x = self.out(x)

        return x
```

How to Train a Neural Network in `PyTorch`

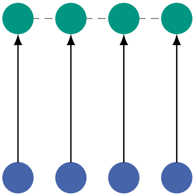
3. Training loop

- Define loss function and optimizer
- Loops over epochs (training iterations)
- For each mini-batch in `DataLoader` (`torch.utils.data`)
  - Initialize gradients (`optimizer.zero_grad()`)
  - Pass samples through model
  - Calculate loss between model output and targets
  - Backpropagation: (`loss.backwards()`)
  - Optimizer updates model weights based on gradients (`optimizer.step()`)

```python
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(
    net.parameters(),
    lr=0.001,
    momentum=0.9
)

for epoch in range(2):
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

HTC versus HTC

High-Throughput Computing (HTC)

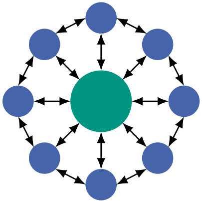


Technologies and methods to efficiently process several loosely coupled task to maximize throughput.

▶ mainly inference

● ≙ Data
● ≙ Compute Nodes

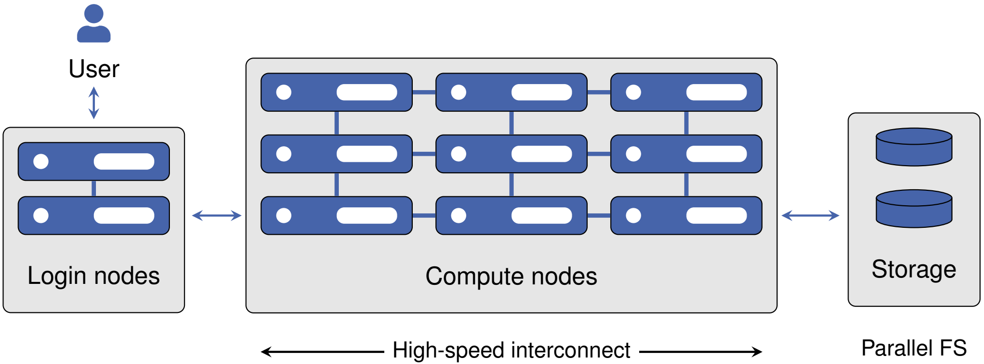
High-Performance Computing (HPC)



Technologies and methods to solve complex computational tasks that are strongly coupled and need to be parallelized on fine granular scale..

▶ mainly model training

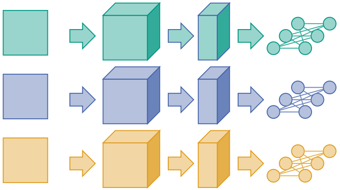
Anatomy of a High-performance Clusters



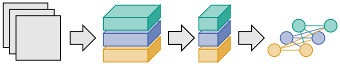
- Multi-user, multi-node distributed memory computing system
- Off-the-shelf components connected with high-speed network, e.g. Infiniband

Parallel Neural Networks

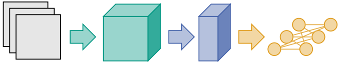
Data Parallelism



Model Parallelism



Pipelining



- Copy of the model on each processor
- Data is distributed across processors in disjoint sets
- Usually: **acceleration**

- Model is distributed, i.e. each processor holds subset W_p of model weights
- Usually: **enabling**

- Special type of model parallelism
- Connected parts of model are distributed across processors
- Usually: **avoid**

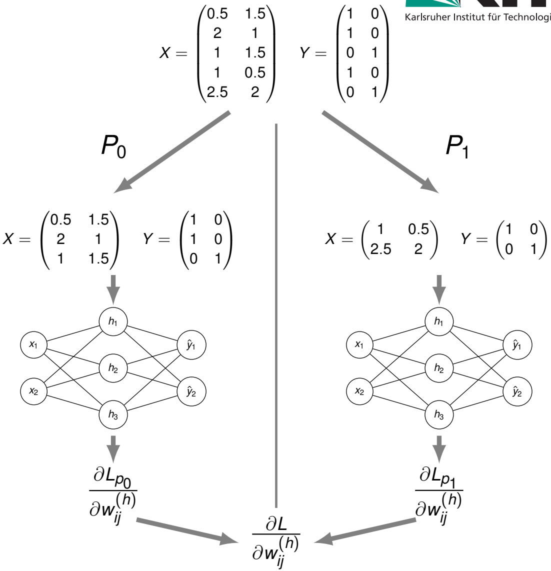
Data-Parallel Neural Networks

Karlsruher Institut für Technologie

- Data is distributed as **disjoint subsets** (chunks) across all processors.
- Each process conducts forward-backward pass on its data with **local model copy**
- Model **weights** are **synchronized** across all processes

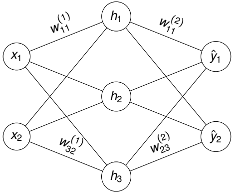
→ Communication, **averaging** of gradients

- After synchronization all copies are identical



Data-Parallel Neural Networks

Process P_0

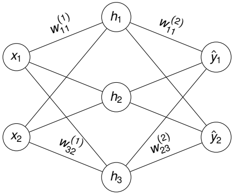


$$X = \begin{pmatrix} 0.5 & 1.5 \\ 2 & 1 \\ 1 & 1.5 \end{pmatrix}$$

$$Y = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow \quad \frac{\partial L_{p_0}}{\partial w_{11}^{(1)}} = -0.00288$$

Process P_1



$$X = \begin{pmatrix} 1 & 0.5 \\ 2.5 & 2 \end{pmatrix}$$

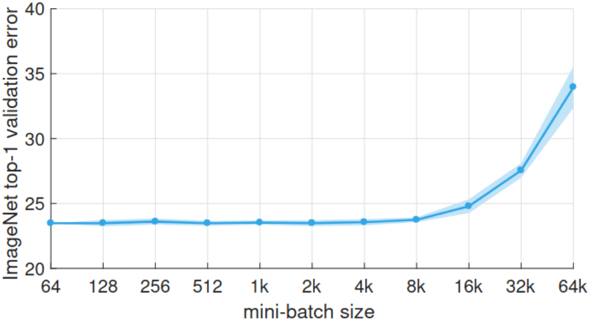
$$Y = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow \quad \frac{\partial L_{p_1}}{\partial w_{11}^{(1)}} = 0.08635$$

$\Longrightarrow$ Averaging (communication): $\frac{\partial L_p}{\partial w_{11}^{(1)}}$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{1}{2}\left(\frac{\partial L_{p_0}}{\partial w_{11}^{(1)}} + \frac{\partial L_{p_1}}{\partial w_{11}^{(1)}}\right) = \mathbf{0.04173}$$

Large-batch Effects



- **Increasing parallelism** increases global batch size, effectively **reducing predictive performance**

→ Data-parallelism is not infinitely scalable due to large-**batch effects**

HAICORE – System

- **H**elmholtz **AI CO**mputing **RE**sources
- System for applied AI **research**, free-of-charge
- Self-registration on FELS:

1. Navigate to https://fels.scc.kit.edu/
2. Sign in with KIT identity provider
3. Set up two-factor authentication
4. Register HAICORE service

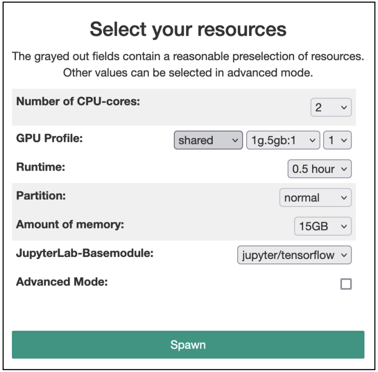
Source: Steinbuch Centre for Computing

HAICORE – Access

- Browser access via **Jupyter**
  - KIT: haicore-jupyter.scc.kit.edu
  - Drop-down resource selection
- Bare metal access via `ssh`

```
ssh <KIT-ID>@haicore.scc.kit.edu
```

  - Unix shell
- **SLURM** resource scheduling system
- Documentation: Jupyter@KIT



HAICORE – Modules

- Commonly used **software preinstalled**
  - **Why:** no administrator rights
  - Usually multiple versions available
- Modules available in **Jupyter**
  - Blue double cube icon left
  - Entire module list visible
- Troubleshooting
  - Request installation via ticket
  - Self-compilation
- **NOTE:** modules always first

Searching

```
module spider cuda
```

```
Versions:
    devel/cuda/11.8
    devel/cuda/12.0
```

Loading

```
module load devel/cuda/11.8
```

Collections

```
module save <NAME>
module purge
module restore <NAME>
```

HAICORE – Python

- Various **Python** versions (3.6+) as **system packages** available

- **Virtual Environments** for package management

```
virtualenv -p python <PATH>
source <PATH>/bin/activate
pip install ...
```

- Installing an active virtual environment in Jupyter

```
python -m ipykernel install --user --name <NAME> (--display-name <NAME>)
```

- Anaconda possible, but discouraged
  - **Licensing** issue for possible commercial spin-offs
  - **Clashes** with module **binaries**

HAICORE – Job Scripts

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --partition=normal
#SBATCH --gres=gpu:full:1
#SBATCH --time=40:00
#SBATCH --mail-type=BEGIN
#SBATCH --mail-user=markus.goetz@kit.edu

module restore <COLLECTION>
source <PATH>/bin/activate
srun python -u <SCRIPT>
```

- Generally **two parts** for a **SLURM job** script
  - Declarative **header** for **resource** allocation/request
  - Main **body** with actual processing commands
  - **REMEMBER:** restore modules and venv first

HAICORE – Controlling Jobs

- **Submission** to the scheduler

```
sbatch <JOB_SCRIPT>
```

- **Interactive** jobs on the shell, block and spawns remote shell

```
salloc <PARAMS>
salloc --nodes=1 --gres=gpu:1
```

- **Monitoring** job queue

```
squeue
```

- **Cancelling** jobs

```
scancel <JOB-ID>
```

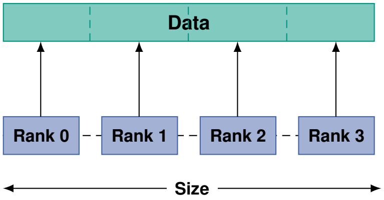
- Querying the **utilization** of the nodes

```
sinfo_t_idle
```

Message Passing

- **SPMD** – Single Program Multiple Data
  - All nodes execute **same program**
  - Diverging behavior using **branching**
  - Data usually roughly equally partitioned
  - Technologies: MPI, Facebook Gloo, NVidia NCCL
  - Focus: computational performance

- Message Passing Interface (**MPI**)
  - Defacto standard on HPC clusters
  - Definition of communication signatures
  - **Communicators** handle **process** groups
  - Multiple implementations



Message Passing

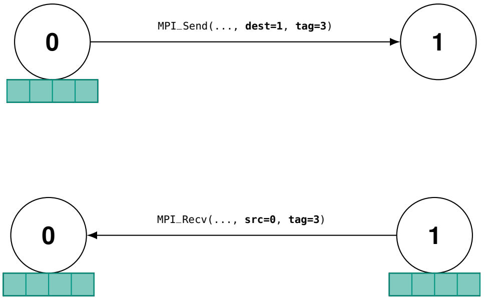
- `mpi4py` for MPI in Python
  - Wraps C/C++ binaries
  - Other modules not maintained
- Two usage modes
  - **Small letter** – flexible, arbitrary object, (de-)serializes data
  - **Capital letter** – buffered, `array`-like objects, highest performance
- Official documentation: https://mpi4py.readthedocs.io/en/stable/

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
```

```python
if rank == 0:
    data = numpy.arange(1000, dtype='i')
    comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
    data = numpy.empty(1000, dtype='i')
    comm.Recv([data, MPI.INT], source=0, tag=77)
```

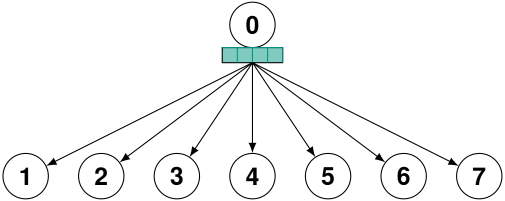
Message Passing

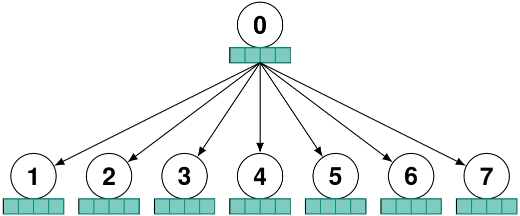
`MPI_Send` allows to transmit buffers with homogeneous data (e.g. `MPI_INT` or `MPI_DOUBLE`) between ranks. With **`MPI_Recv`** messages can be received.



Message Passing

`MPI_Bcast` broadcasts data to all ranks in a `Communicator`





Comparison of `MPI_Send`/`MPI_Recv` with `MPI_Bcast`, 100.000 32 bit integers

Processes	2	4	8	16
Send/Recv in s	0.0344	0.1025	0.2385	0.5109
Broadcast in s	0.0344	0.0817	0.1084	0.1296

Source: https://mpitutorial.com/tutorials/mpi-broadcast-and-collective-communication/

Message Passing

`MPI_Reduce` receives an input array and applies a **reduction operation** element-wise.

`MPI_Allreduce` is analogous to `MPI_Reduce`, but replicated the result on all ranks.

→ **Data-parallel neural networks average the gradients with Allreduce after backpropagation**

Data Parallel Training with PyTorch ddp

Karlsruher Institut für Technologie

```
from torch.nn.parallel import DistributedDataParallel as DDP

def main(): # Will be executed in parallel on all PEs via srun command.
    # SETUP
    world_size = int(os.getenv("SLURM_NPROCS")) # Get overall number of processes.
    rank = int(os.getenv("SLURM_PROCID"))       # Get individual process ID.
    address = os.getenv("SLURM_LAUNCH_NODE_IPADDR")
    port = "29500"
    os.environ["MASTER_ADDR"] = address
    os.environ["MASTER_PORT"] = port
    dist.init_process_group(backend="nccl", rank=rank, world_size=world_size)

    # MODEL
    model = TorchModel().cuda() # Create model and move it to GPU.
    ddp_model = DDP(model)      # Wrap model with DDP.
    optimizer = torch.optim.SGD(ddp_model.parameters())
```

Data Parallel Training with PyTorch ddp

```python
# TRAINING
train_loader = get_data_loader()

for epoch in range(num_epochs):
    train_loader.sampler.set_epoch(epoch) # Pass current epoch to sampler.
    ddp_model.train()

    for batch_idx, (features, targets) in enumerate(train_loader):
        logits = ddp_model(features) # Forward pass.
        loss = torch.nn.functional.cross_entropy(logits, targets)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Logging.
        torch.distributed.all_reduce(loss) # All-reduce local mini-mini-batch losses.
        loss /= world_size

# CLEAN UP
dist.destroy_process_group() # Clean up: Eliminate process group.
```

Data Parallel Training with PyTorch ddp

```python
def get_data_loader():
    train_dataset = ExampleDataset()

    # Specifically for DDP training: DistributedSampler restricts data loading
    # to an exclusive, disjoint subset of the entire dataset.
    train_sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,           # Dataset to sample from
        num_replicas=world_size, # Number of processes in distributed training
        rank=rank,               # Rank of current process within num_replicas
        shuffle=True,            # Shuffle indices
        drop_last=True           # Drop tail to make data evenly divisible
    )

    # Combine dataset and sampler within dataloader
    train_loader = torch.utils.data.DataLoader(
        dataset=train_dataset,
        batch_size=batch_size,
        drop_last=True,
        sampler=train_sampler
    )

    return train_loader
```

Energy Monitoring

- **Why:** monitoring resource footprint, nice paper addition
- **What:** HAICORE has spezialized sensors, energy consumption report per job

```
...
CPU Efficiency: 1.47% of 12:12:20 core-walltime
Memory Efficiency: 2.95% of 122.00 GB
Energy Consumed: 1204666 Joule / 334.629444444444 Watthours
Average node power draw: 548.323167956304 Watt
```

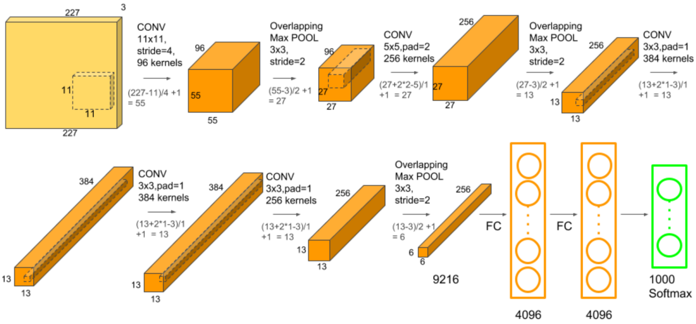
- Querying SLURM

```
sacct -X -o ConsumedEnergy --user <ID>
```

- Alternative `perun`
  - `pip install perun`
  - https://github.com/Helmholtz-AI-Energy/perun

AlexNet

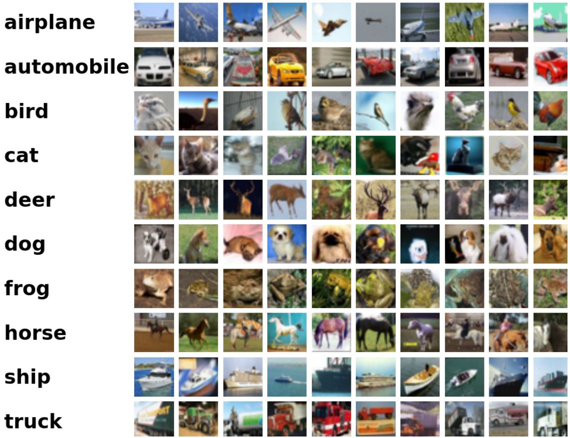
- Classification model
- Convolutional Neural Network
  - 5 convolutional layers (incl. max-pooling)
  - 3 fully connected layers



Source: `https://learnopencv.com/understanding-alexnet/`

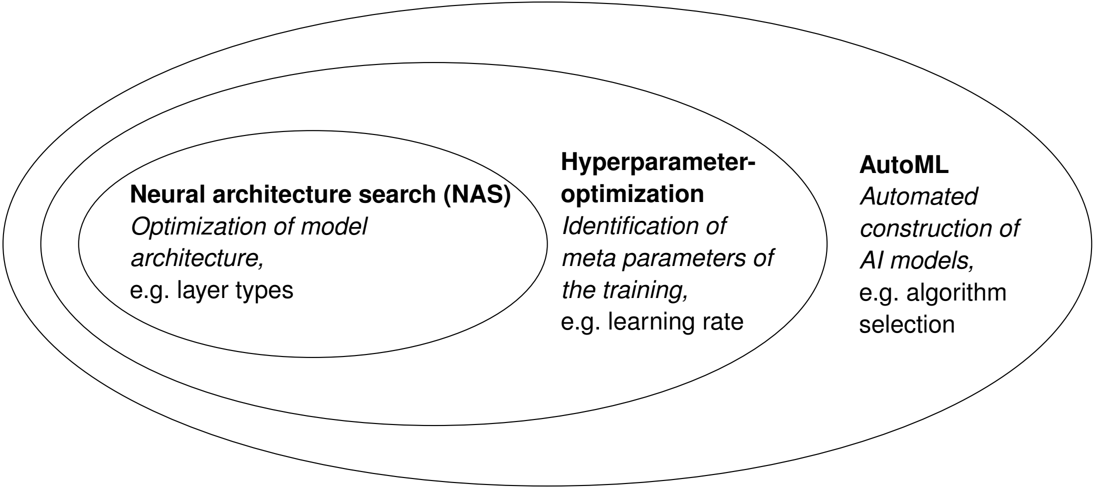
CIFAR-10 Dataset

- 60 000 RGB images, 32 $\times$ 32 pixels
- 10 classes, 6 000 images per class
- 50 000 training samples
- 10 000 test samples



Source: `https://www.cs.toronto.edu/~kriz/cifar.html`

AutoML-Hierarchy



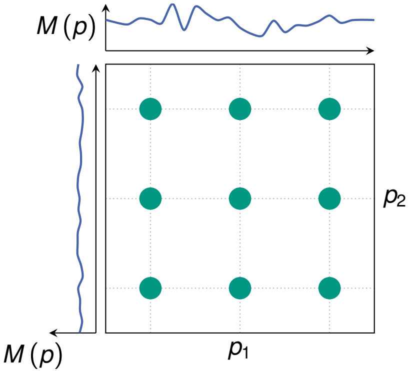
AutoML Example: Spectra Upsampling

- Condition monitoring and predictive maintenance at the US Army
  - Currently: fixed interval maintenance
  - Now: attempt to establish a **demand-oriented maintenance**
  - Pilot project for **cargo choppers**: rotor vibration spectra used for manual classification
  - Only **partial spectra** can be stored during an operation (full spectra multiple TB)
  - Selected helicopters equipped with full recorders

- Approach: **neural upsampling**
  - 241 inputs, 8193 outputs, fully-connected regressional decoder
  - First design (**naive architecture**): $\approx$ **58% precision**
  - Second design (three weeks of **manual optimization** by Facebook): $\approx$ **77% precision**
  - Third design (genetic **hyperparameter optimization**, 72h runtime): $\approx$ **99,8% precision**
  - 100 individuals, 20 generations, 36 NVidia P100 GPUs, $400\,\mathrm{GB\,s^{-1}}$ interconnect

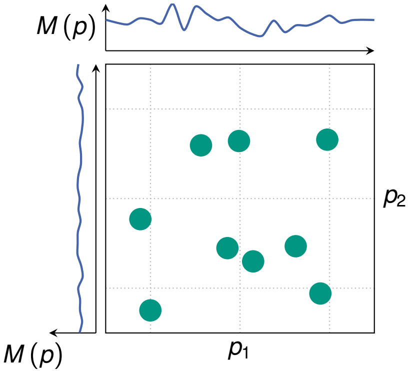
Grid Search

- Grid Search is common nïve approach
  - Manual definition of cartesian grid
  - Try out each candidate.
- **Curse of dimensionality:** Each feature leads to exponential growth of search space
- **Trivial parallelization strategy**
  - Parameter sets are independent
  - Arbitrary uniform partition of space
  - Try out candidates independently, use `Allreduce` across target metric, possibly **checkpoint** intermediate staete



Random Search

- Random testing of solutions
  - Parameter set randomly sample from search space
  - Candidates solution independent from one another
- **Curse of dimensionality** still relevant
- More **robust** towards **uncorrelated features**
- Parallelization strategy analogous to grid search
  - NOTE: select different random seeds
  - Easy resuming

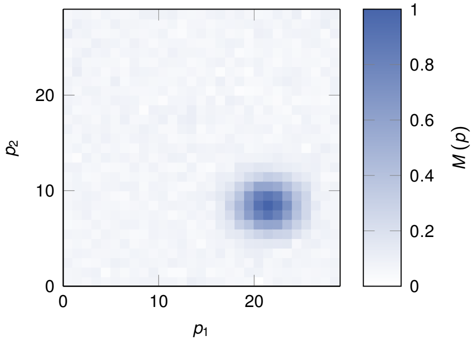


Effiency of Random Search

- Random search has **high coverage**
- **Probability**, of finding a **good solution** with **low sampling** is **high**:

$$P_{\text{miss}} = (1 - P_{\text{viable}})^{\text{trials}}$$
$$P_{\text{hit}} = 1 - P_{\text{miss}}$$

- Example
  - 3 % of search space close to optimal (P_{viable})
  - 100 optimization steps (*trials*)
  - $P_{\text{miss}} = (1 - 0.03)^{100} \approx 0.048$
  - $P_{\text{hit}} = 1 - 0.048 \approx 0.952$

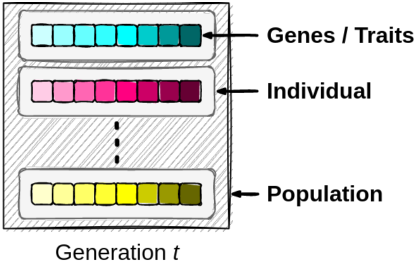


Evolutionary Algorithms (EAs)

WHAT “Survival of the fittest” metaheuristics inspired by biological evolution

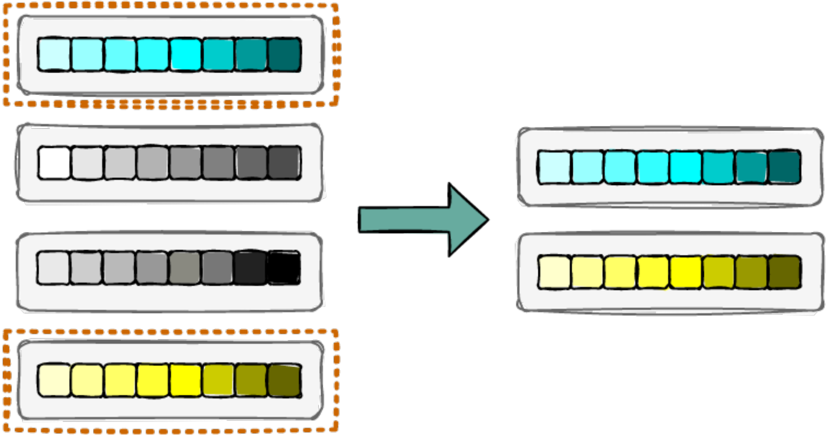
WHY Find good-enough solution for global optimization problems efficiently.
HOW Ingredients

- **Individuals** Representation of candidate solutions in search space, vector of parameters to be optimized
- **Fitness function** Scalar metric to evaluate how good an individual is, metric to optimize on
- **Evolutionary operators** Mechanisms for breeding new (hopefully better) individuals from current ones



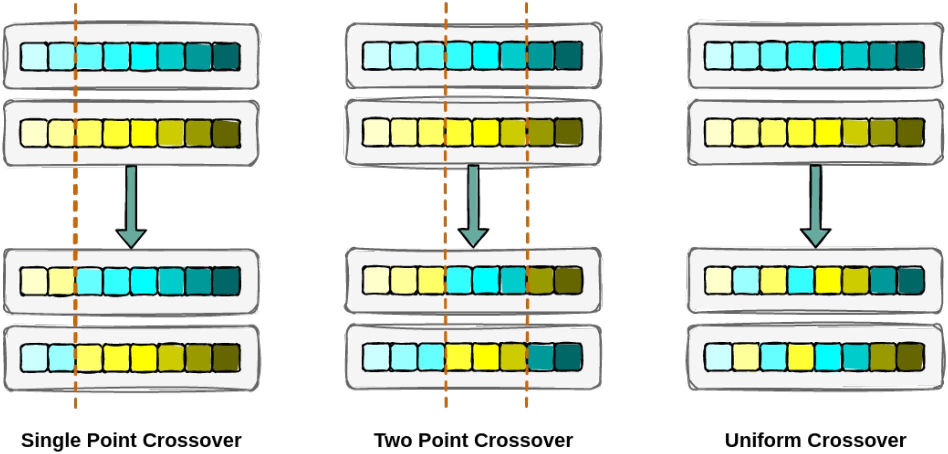
Selection

→ Select individuals from current generation for breeding, usually somehow based on their fitness.



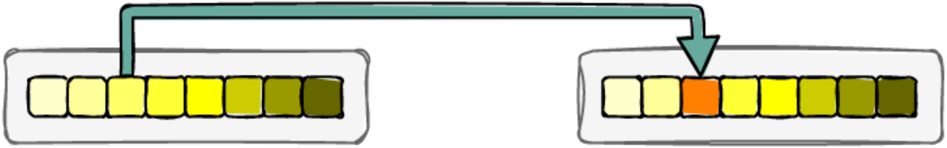
Crossover

→ Generate new child individuals from selected parent individuals by recombining the latters' genes.



Mutation

→ Randomly change an individual's genes to promote genetic diversity.

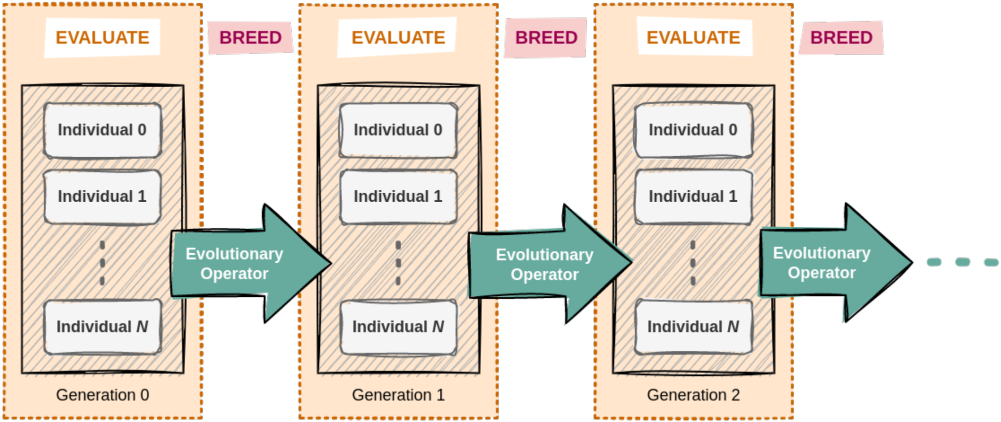


Overall

- Improve population's average fitness by repetitively applying a stochastic combination of selection, crossover, and mutation.

✓ Find near-optimal solution.

Parallel Synchronous EAs



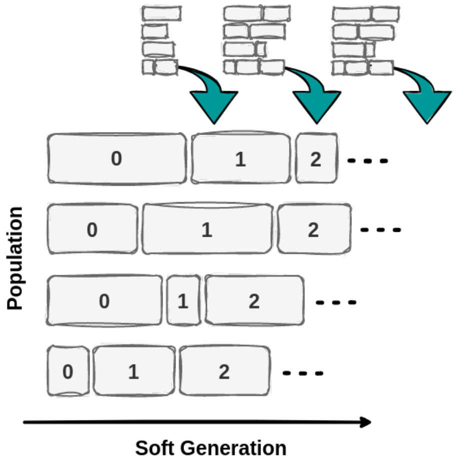
- **Evaluations** easily **parallelizable** within each generation

`propulate` – Asynchronous Parallel Evaluation

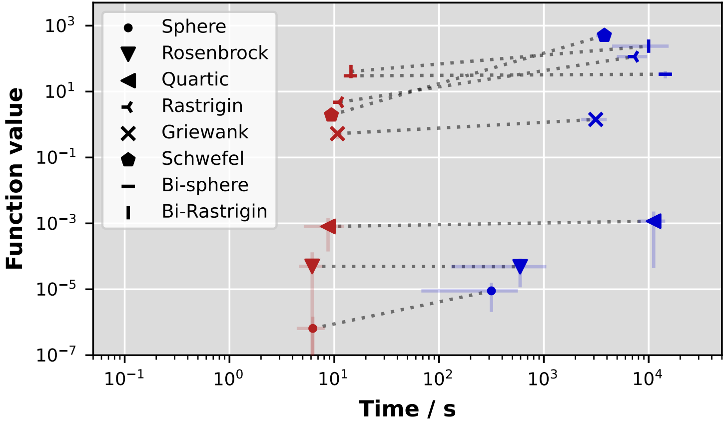
- Breed and evaluate **asynchronously** using **continuous** population of all individuals evaluated so far.

↪ Maximizes efficiency by independent workers not waiting for each other!

- `propulate` is an MPI-parallel `Python` for HPO **at scale**
  - `pip install propulate`
  - *github.com/Helmholtz-AI-Energy/propulate*
  - *arxiv.org/abs/2301.08713*



How Good Is It? Propulate vs. Optuna



Evaluation

- Benchmark function minimization accuracy over wallclock time
- Lowest function values found vs. wallclock time to reach them, each averaged over ten runs
- All benchmark functions have their minimum at 0

`propulate` – How To Use It Recipe

- **Search space** of parameters to be optimized as `dict`
  - `int` for ordinal
  - `float` for continuous
  - `str` for categorical
- **Fitness** or loss **function** to optimize upon
  - Inputs: Parameters to be optimized
  - Output: Scalar fitness or loss
  - Can be a black box
  - ! `Propulate` minimizes, so choose sign appropriately
- Adapt `Propulate` hyperparameters as you wish
- **Run** optimization

Source: https://xkcd.com/1639/

Example: Hyperparameter Search with `Propulate`

```python
# IMPORTS
import torch
from torch import nn
from torch.utils.data import DataLoader

from pytorch_lightning import LightningModule, Trainer
from torchmetrics import Accuracy

from torchvision.datasets import MNIST
from torchvision.transforms import Compose, ToTensor, Normalize

from mpi4py import MPI
import random

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

Example: Hyperparameter Search with `Propulate`

Karlsruher Institut für Technologie

```python
# SETTINGS
num_generations = 3                # number of generations
pop_size = 2 * MPI.COMM_WORLD.size # size of breeding population
GPUS_PER_NODE = 4                  # number of GPUs per compute node
# SEARCH SPACE
limits = {
    "convlayers": (2, 10),                    # number of convolutional layers, int for ordinal
    "activation": ("relu", "sigmoid", "tanh"), # activation function, str for categorical
    "lr": (0.01, 0.0001),                     # learning rate, float for continuous
}
```

Example: Hyperparameter Search with `Propulate`

```python
class Net(LightningModule):
    """Neural network class."""
    def __init__(self, convlayers, activation, lr, loss_fn):
        """
        Set up neural network.
        Params
        convlayers [int] : number of convolutional layers
        activation [callable] : activation function to use
        lr [float]: learning rate
        loss_fn [callable] : loss function
        """
        super(Net, self).__init__()
        self.loss_fn = loss_fn # Loss function
        self.lr = lr # Learning rate
        layers = [] # Layers (depending on number of convolutional layers specified)
        layers += [nn.Sequential(nn.Conv2d(1,
                                           10,
                                           kernel_size=3,
                                           padding=1),
                                 activation()),]
```

Example: Hyperparameter Search with `Propulate`

```python
        layers += [nn.Sequential(nn.Conv2d(10,
                                    10,
                                    kernel_size=3,
                                    padding=1),
                                activation())
                   for _ in range(convlayers - 1)]
        self.fc = nn.Linear(7840, 10)
        self.conv_layers = nn.Sequential(*layers)
        self.val_acc = Accuracy()

    def forward(self, x):
        """Forward pass."""
        ...
        return x

    def training_step(self, batch, batch_idx):
        """Calculate loss for training step in Lightning traing loop."""
        ...
        return loss
```

Example: Hyperparameter Search with `Propulate`

```python
    def validation_step(self, batch, batch_idx):
        """Calculate loss and accuracy for validation step in Lightning validation loop during training."""
        x, y = batch                    # Get samples and targets from batch.
        pred = self(x)                  # Compute model predictions for samples.
        val_acc = self.val_acc(torch.nn.functional.softmax(pred, dim=-1), y) # Calculate acc on validation batch.
        # Softmax rescales tensor so that its elements lie within [0,1] and sum to 1.
        if val_acc > self.best_accuracy:
            self.best_accuracy = val_acc # Metric to optimize on in Propulate!
        return self.loss_fn(pred, y)     # Calculate loss value from predictions and actual targets.

    def configure_optimizers(self):
        """Choose optimizers and LR scheduler in Lightning."""
        return torch.optim.SGD(self.parameters(), lr=self.lr)

def get_data_loaders(batch_size, root="."):
    """Get PyTorch dataloaders for training and validation."""
    ...
    return train_loader, val_loader
```

Example: Hyperparameter Search with `Propulate`

```python
def ind_loss(params):
    """
    Fitness function. Use the model's validation accuracy as metric to optimize on.
    Params
    ------
    params [dict] : parameter combination
    """
    # Extract HP values from input dict.
    convlayers = params["convlayers"] # number of convolutional layers
    activation = params["activation"] # activation function
    lr = params["lr"]                 # learning rate

    # Set number of epochs to train.
    epochs = 2

    activations = {"relu": nn.ReLU, "sigmoid": nn.Sigmoid, "tanh": nn.Tanh}
    activation = activations[activation]              # Get activation function.
    loss_fn = torch.nn.CrossEntropyLoss()             # Use cross-entropy loss for multi-class classification.
```

Example: Hyperparameter Search with `Propulate`

Karlsruher Institut für Technologie

```python
    model = Net(convlayers, activation, lr, loss_fn) # Set up neural network with specified HPs.
    model.best_accuracy = 0.0                        # Initialize the model's best validation accuracy.

    train_loader, val_loader = get_data_loaders(batch_size=8) # Get training and validation dataloaders.

    # Under the hood, the Lightning Trainer handles the training loop details.
    trainer = Trainer(max_epochs=epochs,         # Stop training once this number of epochs is reached.
                      accelerator='gpu',         # Pass accelerator type.
                      devices=[                  # Devices to train on.
                          MPI.COMM_WORLD.Get_rank() % GPUS_PER_NODE
                              ],
                      enable_progress_bar=False) # Disable progress bar.

    trainer.fit(      # Run full optimization routine.
        model,        # model
        train_loader, # data loader for training samples
        val_loader)   # data loader for validation samples

    # Return negative best validation accuracy as an individual's fitness.
    return -model.best_accuracy.item()
```

Example: Hyperparameter Search with `Propulate`

```python
if __name__ == "__main__":
    rng = random.Random(MPI.COMM_WORLD.rank) # random number generator specifically for optimization
    propagator = get_default_propagator( # default evolutionary operator
        pop_size, # breeding population size
        limits,   # search space
        0.7,      # crossover probability
        0.4,      # mutation probability
        0.1,      # random-initialization probability
        rng=rng   # random number generator
    )
    islands = Islands(
        ind_loss,                     # Fitness function
        propagator,                   # Evolutionary operator
        rng,                          # Random number generator for optimization
        generations=num_generations,  # Number of generations
        num_isles=1,                  # Number of evolutionary islands (not relevant here)
        load_checkpoint="bla",        # Do not start from checkpoint.
        save_checkpoint="pop_cpt.p"   # Save checkpoint to file.
    )
    islands.evolve(top_n=1, logging_interval=1, DEBUG=2) # Run optimization.
```

References I

[1] T. B. Brown, B. Mann, N. Ryder u. a., „Language models are few-shot learners," *arXiv preprint arXiv 2005.14165*, 2020.

[2] OpenAI, *OpenAI Five*, `https://blog.openai.com/openai-five/`, 2018.

[3] J. Jumper, R. Evans, A. Pritzel u. a., „Highly accurate protein structure prediction with AlphaFold," *Nature*, Jg. 596, Nr. 7873, S. 583–589, 2021.

[4] W. Ertel und N. T. Black, *Grundkurs Künstliche Intelligenz*. Springer, 2016.

[5] Martin Abadi, Ashish Agarwal, Paul Barham u. a., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, [accessed at 2021-08-04], 2015. Adresse: `http://tensorflow.org/`.

[6] A. Paszke, S. Gross, F. Massa, A. Lerer u. a., „PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, S. 8024–8035. Adresse: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.