RAISE Center of Excellence

Hyperparameter Optimization for Deep Learning using High Performance Computing

July 18th, 2023

Eric Wulff

CERN

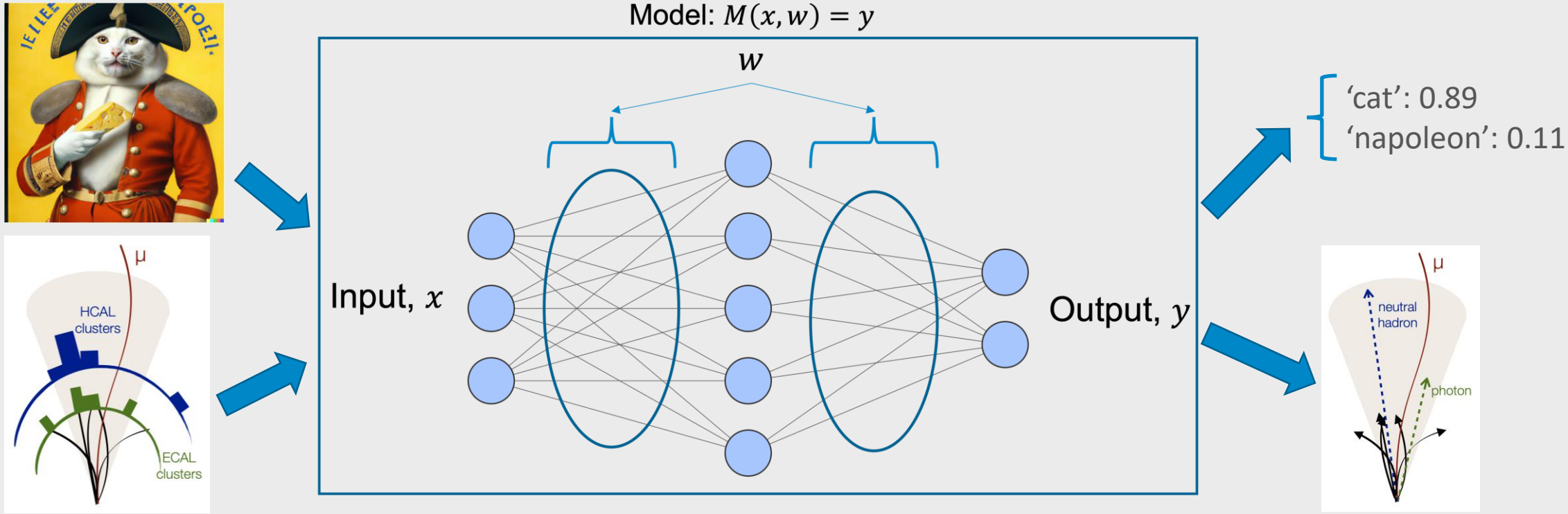
Outline

- A brief introduction to Hyperparameter Optimization
  - Deep Learning
  - Hyperparameter Optimization (HPO)
  - Why we need HPO
  - Intro to some popular HPO algorithms
    - Adaptive configuration selection
    - Adaptive configuration evaluation
- Hyperparameter Optimization in CoE RAISE
  - Machine-Learned Particle-Flow (in collaboration with CMS)
  - HPO on HPC systems
  - Model performance improvements
  - Quantum-assisted Hyperparameter Optimization in CoE RAISE

A brief introduction to Hyperparameter Optimization

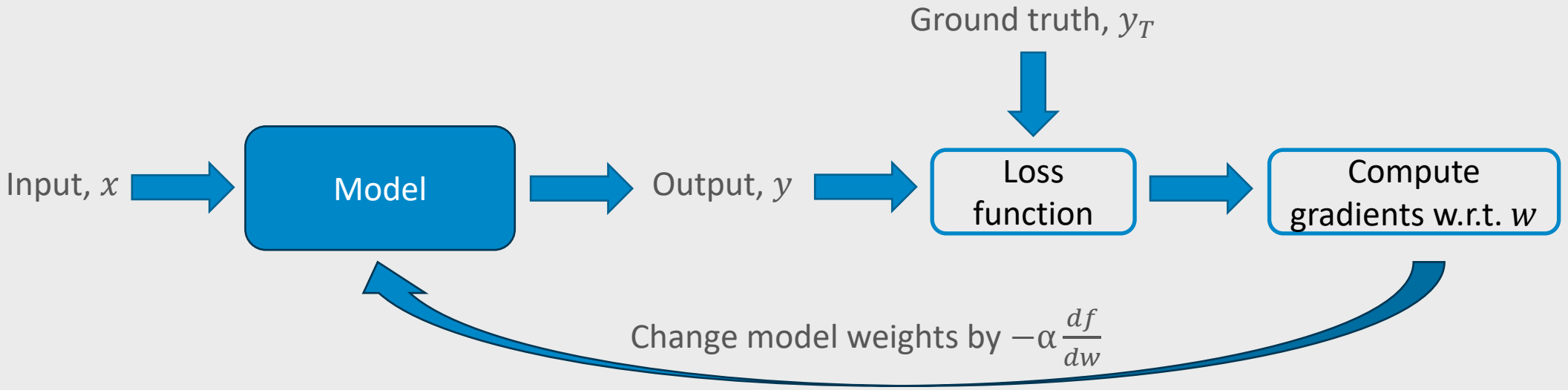
What is Deep Learning? (1/2)

- Deep Learning (DL) is a subset of Machine Learning (ML) where the ML model is a deep Neural Network (NN)
- To be considered deep, the NN should have multiple layers
- Two examples:



What is Deep Learning? (2/2)

- In DL, model parameters w are learned using backpropagation and gradient descent to minimize some objective, or loss, $f(w, \theta)$
- Training:
  - For each x in training data, compute the gradients of the loss and change the model's weights by subtracting from them the gradients multiplied by some learning rate, α
  - Repeat until convergence or other stopping criterion

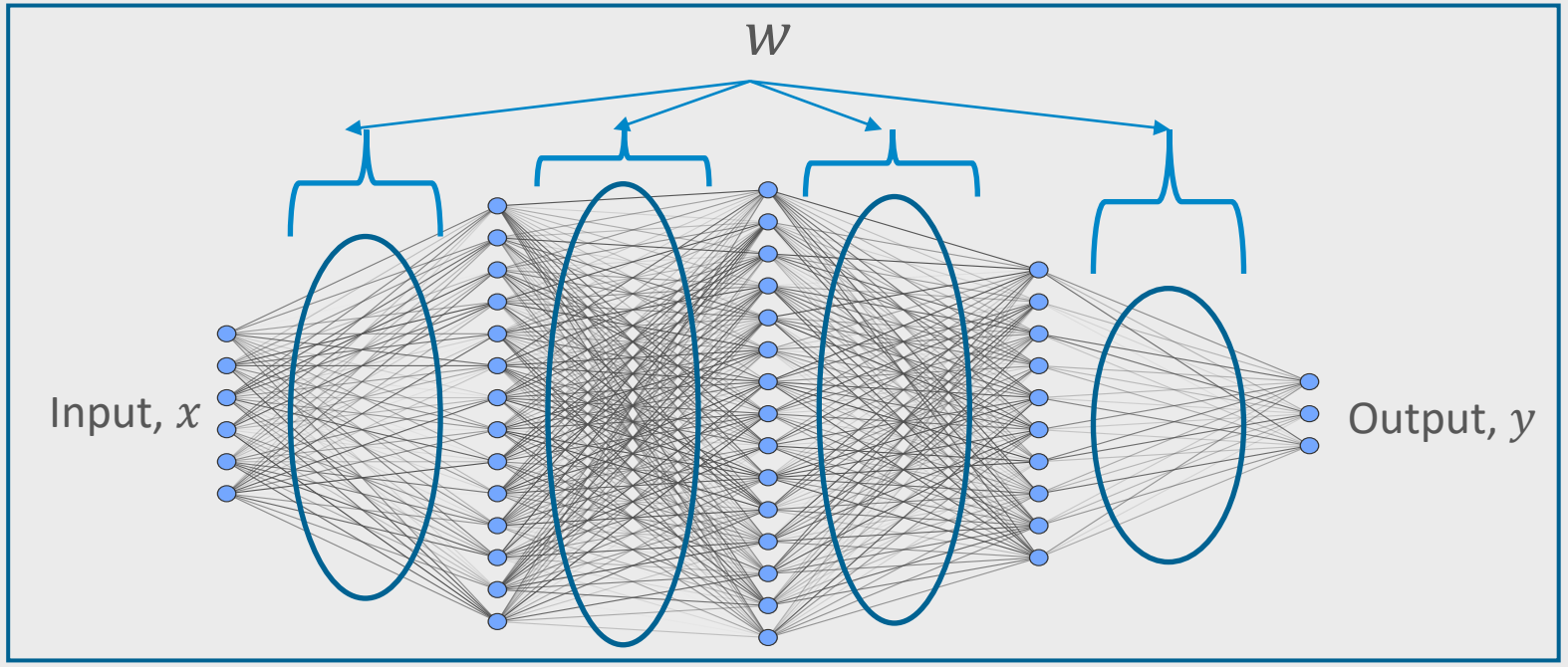


What is Hyperparameter Optimization? (1/3)

- $f(w, \theta)$, depends not only on w, but also on θ
  - w: Model parameters
  - θ: Hyperparameters
    - Number of layers
    - Number of nodes
    - Choice of optimizer, learning rate, batch size, etc.
- Hyperparameter Optimization (HPO) is the process of tuning θ to improve performance

f is the final validation loss after completed training, f is not the model itself

Model: $M(x, w) = y$

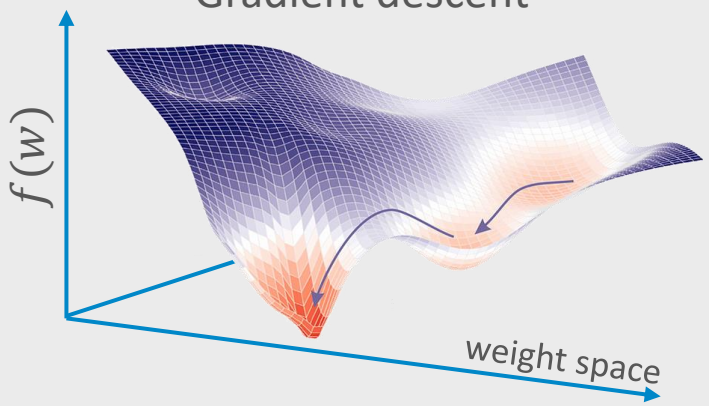


What is Hyperparameter Optimization? (2/3)

- Optimizing the objective $f(w, \theta)$ is done in two different ways
  - 1: Optimize w by gradient descent $\Longrightarrow$ search for $w^* = \arg\min_w f(w, \theta)$
  - 2: Optimize θ by HPO $\Longrightarrow$ search for $\theta^* = \arg\min_\theta f(w, \theta)$

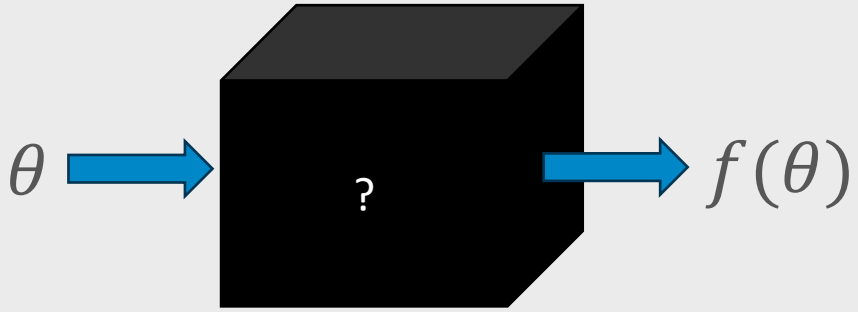
$f(w, \theta)$ is differentiable w.r.t. w

Gradient descent



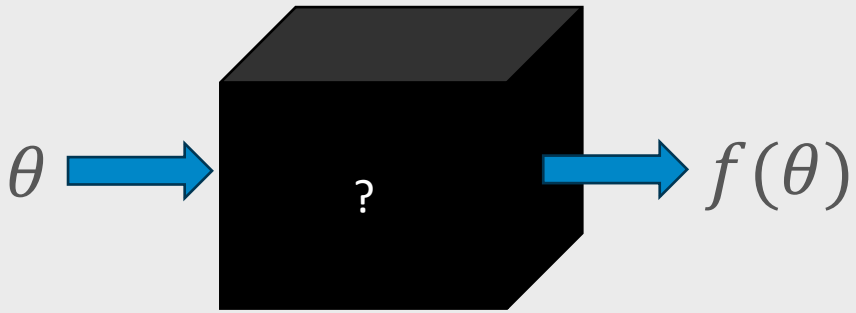
$f(w, \theta)$ non-differentiable w.r.t. θ

- Black-box optimization
- No straightforward update rule for θ



What is Hyperparameter Optimization? (3/3)

- HPO is a black box optimization problem, we want to find $\theta^* = \arg\min_{\theta} f(w, \theta)$ but only get to query values of f, not compute its gradient w.r.t. θ
  - w: Model parameters (learned by gradient descent)
  - θ: Hyperparameters
  - $f(w, \theta)$: What we're trying to minimize, e.g., loss
  - f is non-differentiable w.r.t. θ
- f is often expensive to evaluate
- HPO is compute-resource intensive
  - Benefits greatly from HPC resources
  - In need of smart, efficient search algorithms



Why we need automatic hyperparameter optimization

- Very time consuming for humans, and we do a bad job
- Without good tuning, performance is likely left on the table
- Good tuning is required to accurately compare model architectures

- In 2017, it was shown that then recent advances of STOTA in NLP was not due to the novel architectures, but insufficient tuning of old architectures [1]

On the State of the Art of Evaluation in Neural Language Models

Gábor Melis[†], **Chris Dyer**[†], **Phil Blunsom**[†‡]
{melisgl,cdyer,pblunsom}@google.com
[†]DeepMind
[‡]University of Oxford

Abstract

Ongoing innovations in recurrent neural network architectures have provided a steady influx of apparently state-of-the-art results on language modelling benchmarks. However, these have been evaluated using differing codebases and limited computational resources, which represent uncontrolled sources of experimental variation. We reevaluate several popular architectures and regularisation methods with large-scale automatic black-box hyperparameter tuning and arrive at the somewhat surprising conclusion that standard LSTM architectures, when properly regularised, outperform more recent models. We establish a new state of the art on the Penn Treebank and Wikitext-2 corpora, as well as strong baselines on the Hutter Prize dataset.

https://arxiv.org/abs/1707.05589

Some popular HPO algorithms

- Model-free algorithms
  - Grid search
  - Random search
  - Successive Halving (SHA)
  - Hyperband
  - Asynchronous SHA (ASHA)
  - Evolutionary search
- Model-based algorithms
  - Bayesian optimization
- Hybrid algorithms
  - Combines model-free and model-based methods

Static configuration ***selection***

Adaptive configuration ***evaluation***

Adaptive configuration ***selection***

Selection- and evaluation-based algorithms can easily be combined

Grid and random search

- Grid search
  - Deterministic
  - Exhaustive search (on the grid)
  - Uses same value several times
- Random Search
  - Stochastic
  - Exhaustive search (on the random points)
  - Explores many more values of each HP

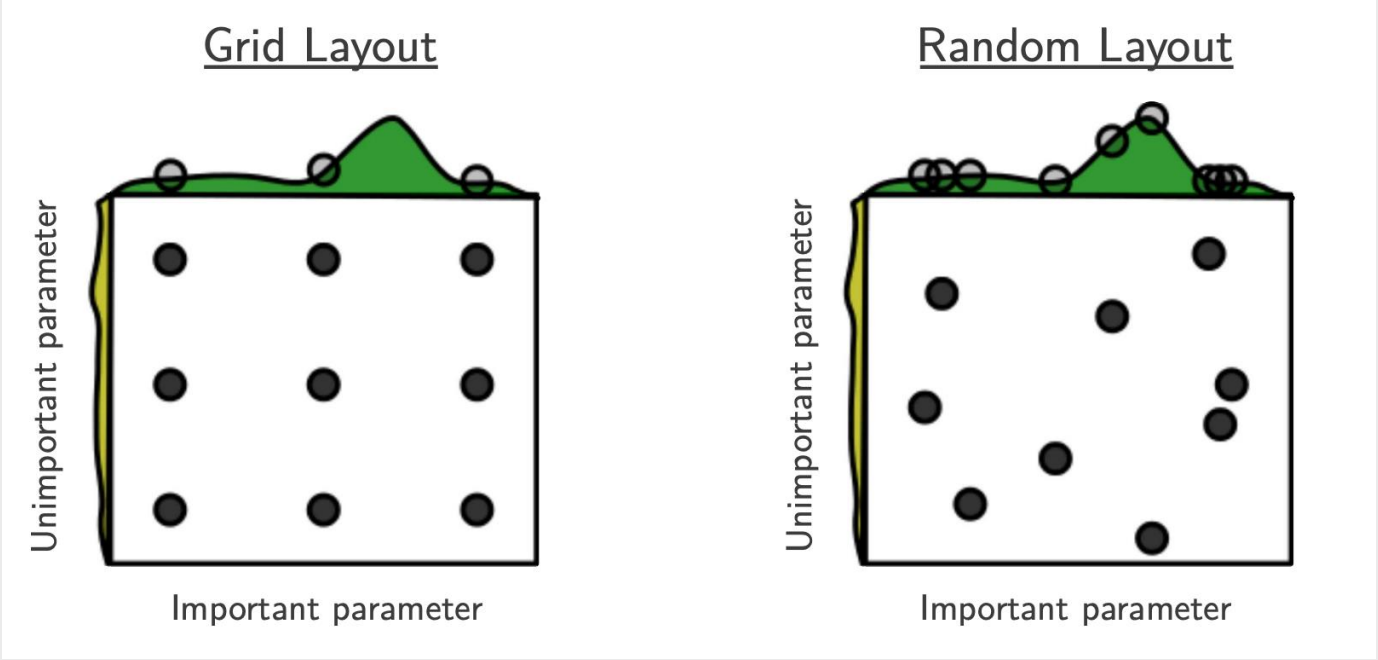


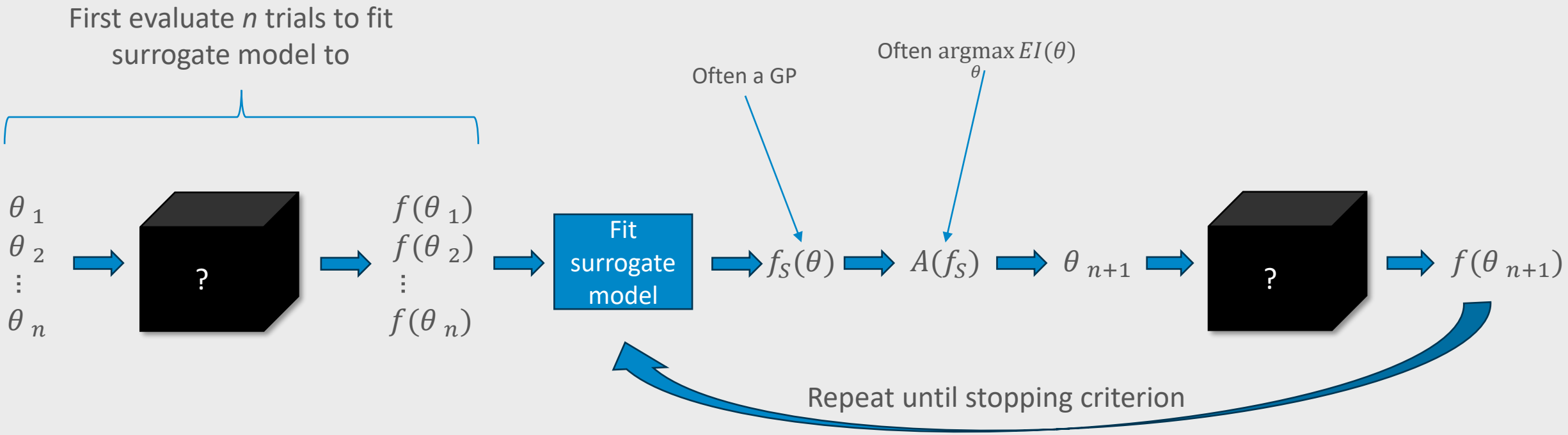
Figure from: [2]James Bergstra and Yoshua Bengio: Random Search for Hyper-Parameter Optimization, https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Adaptive configuration selection

Bayesian Optimization (1/5)

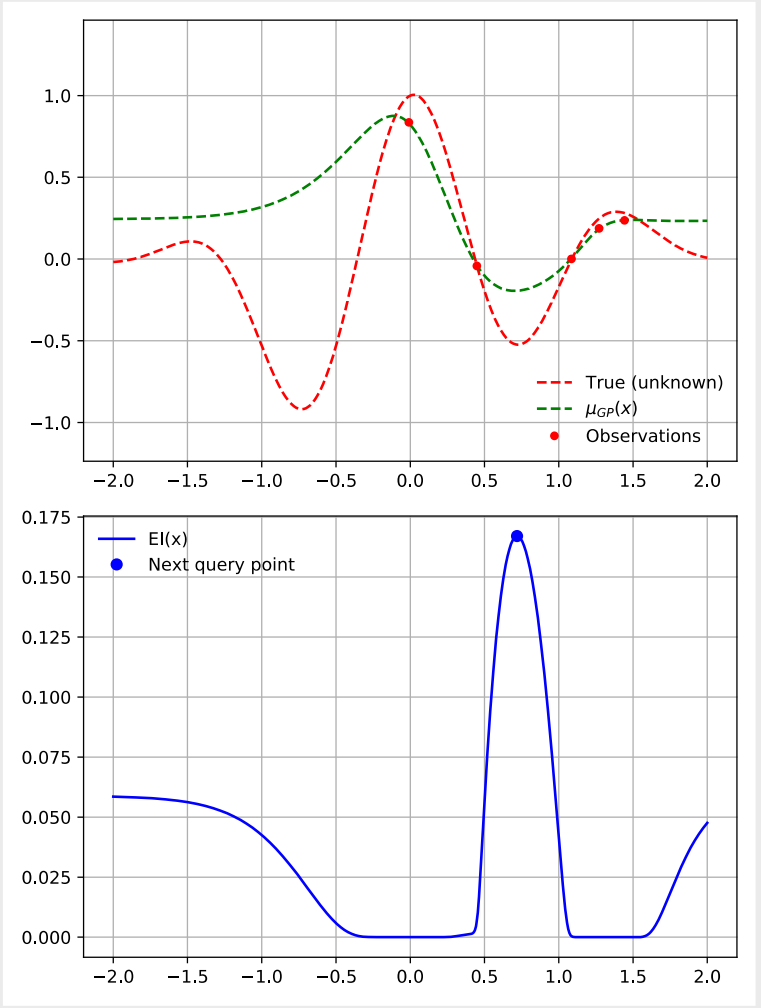
- Bayesian Optimization (BO) is a black-box optimization technique for expensive and/or noisy objectives
- Surrogate model
  - Estimates $f(\theta)$, given some HPs θ
  - Estimates uncertainty of the objective function estimate
  - Must be much faster than evaluating f
- Acquisition function
  - Selects next θ to evaluate
  - Makes exploitation/exploration trade-off
  - Popular choice: Expected Improvement (EI), i.e., how much better is the next observation going to be over our current best?
  - Other choices include Probability of Improvement (PI) and Upper Confidence Bound (UCB)

Bayesian Optimization (2/5)



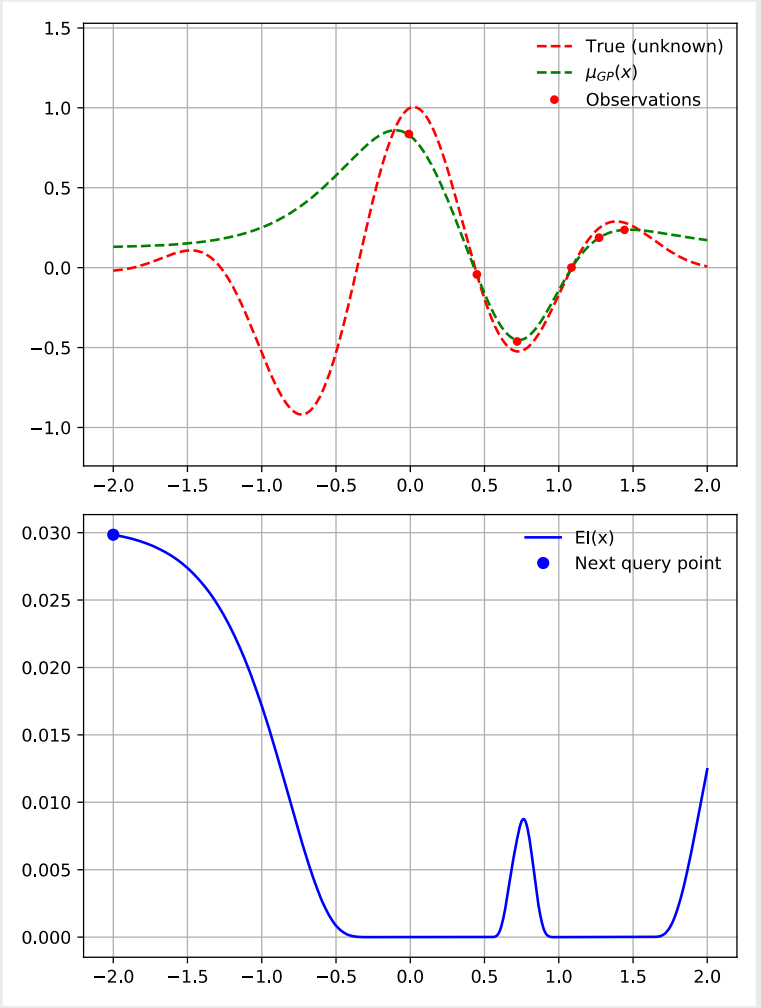
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



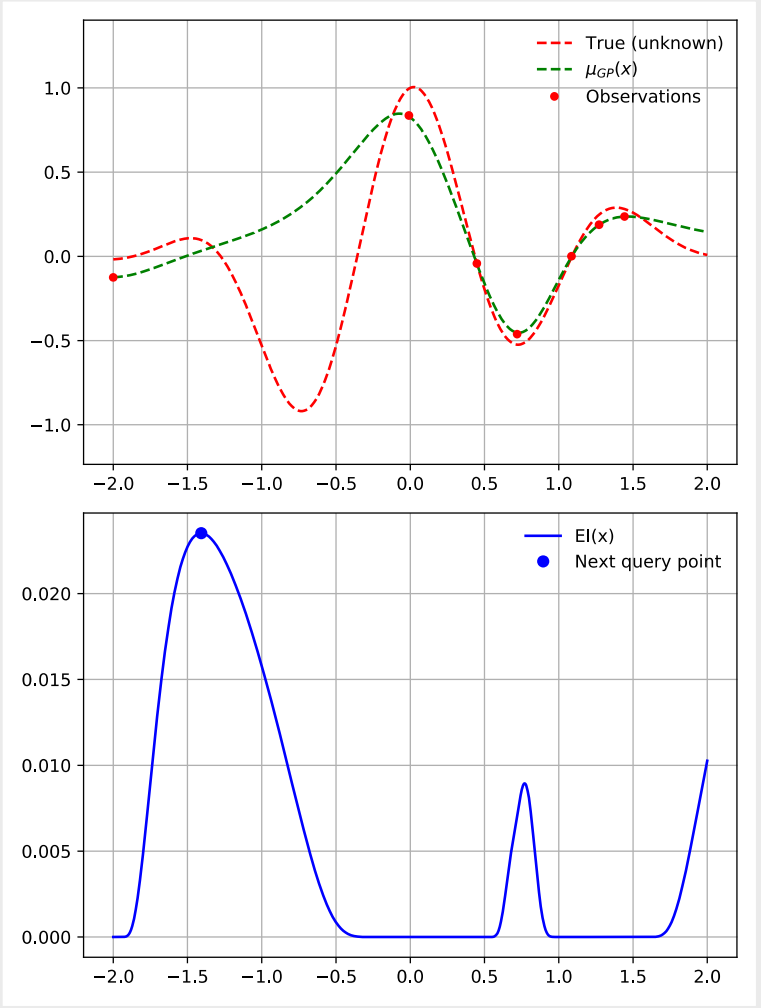
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



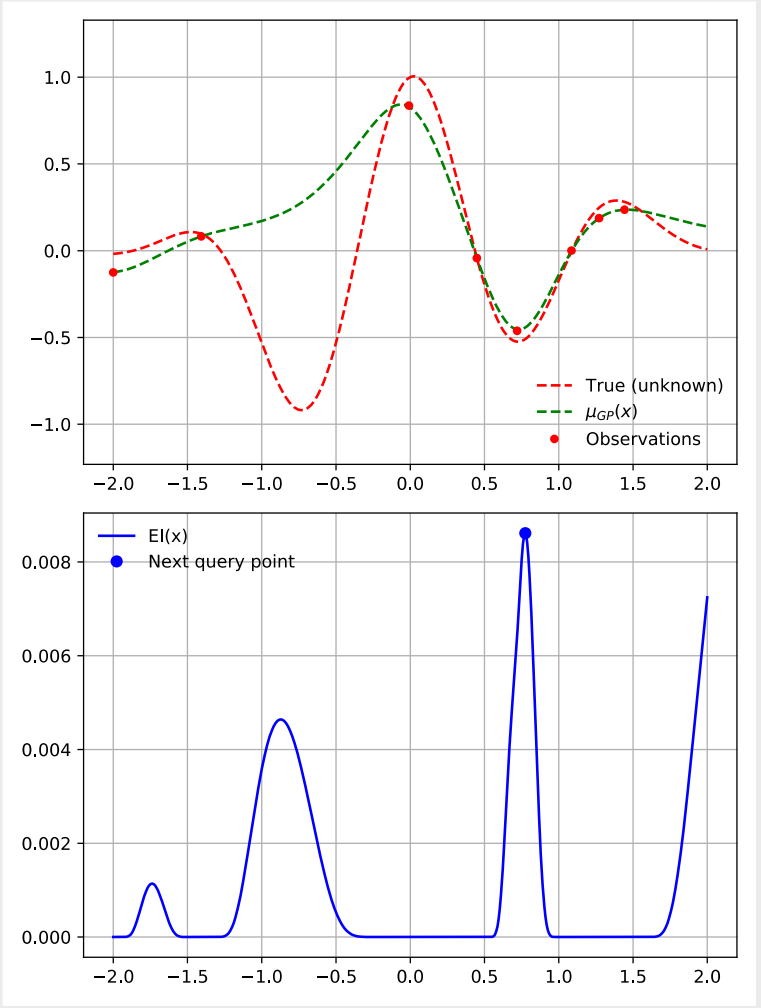
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



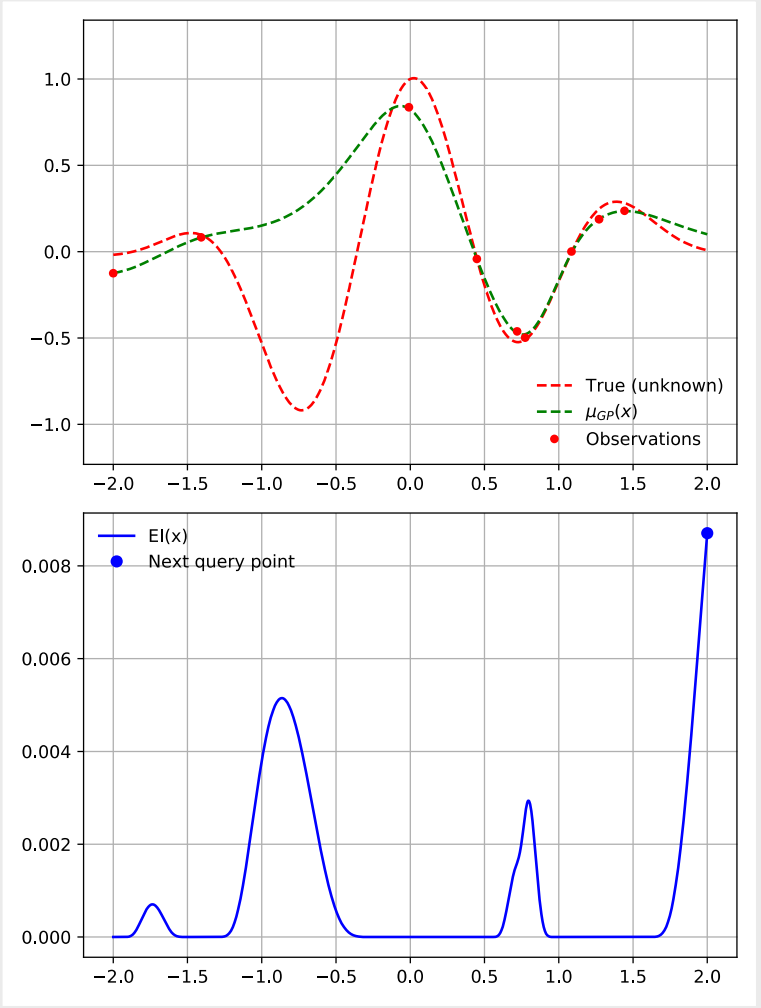
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



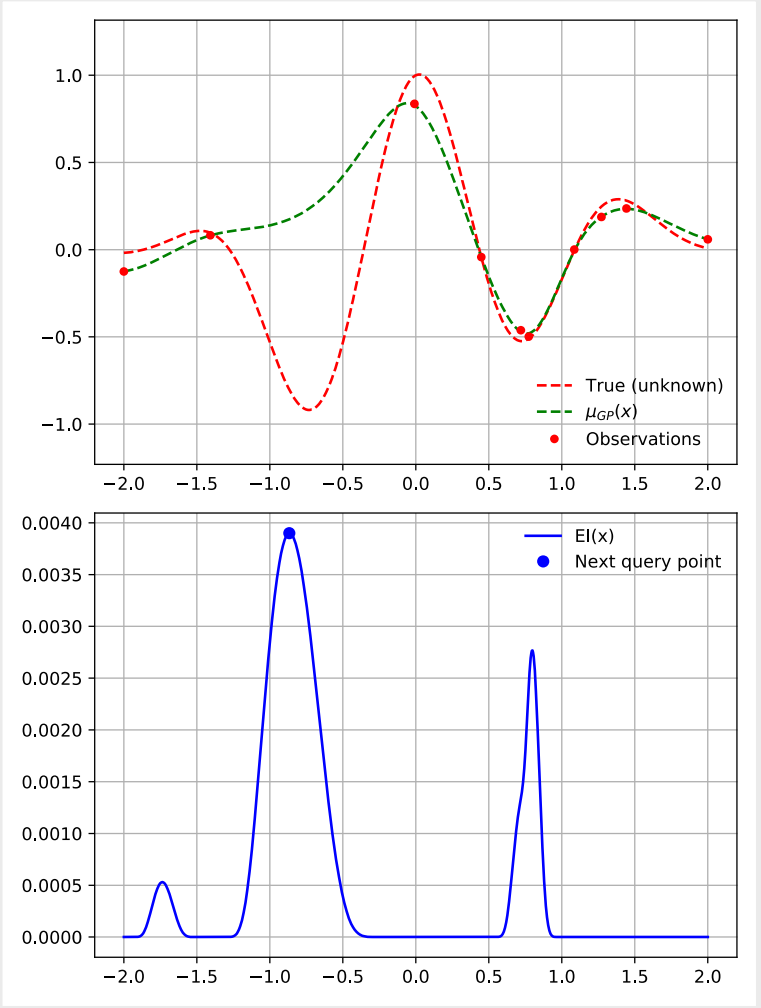
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



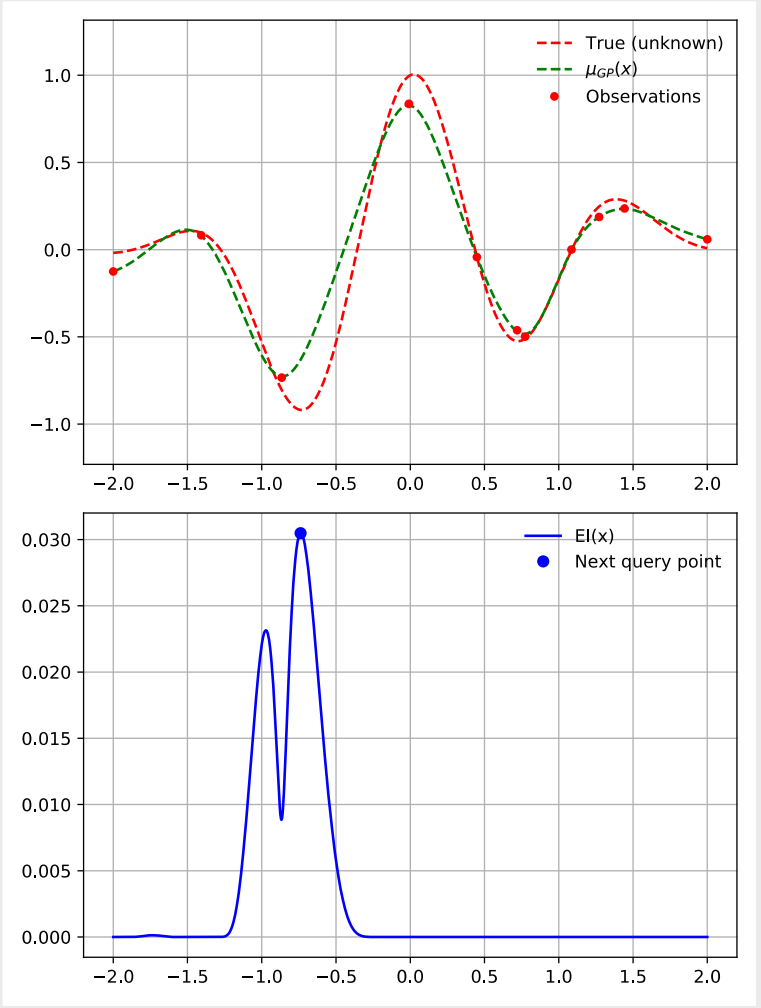
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



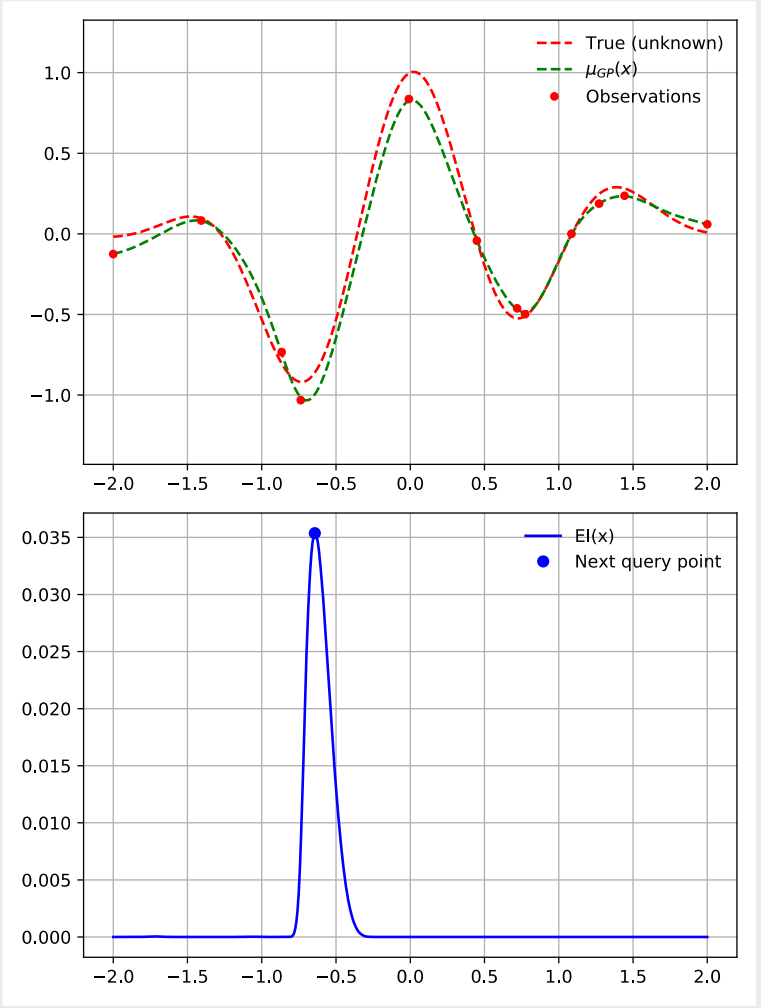
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



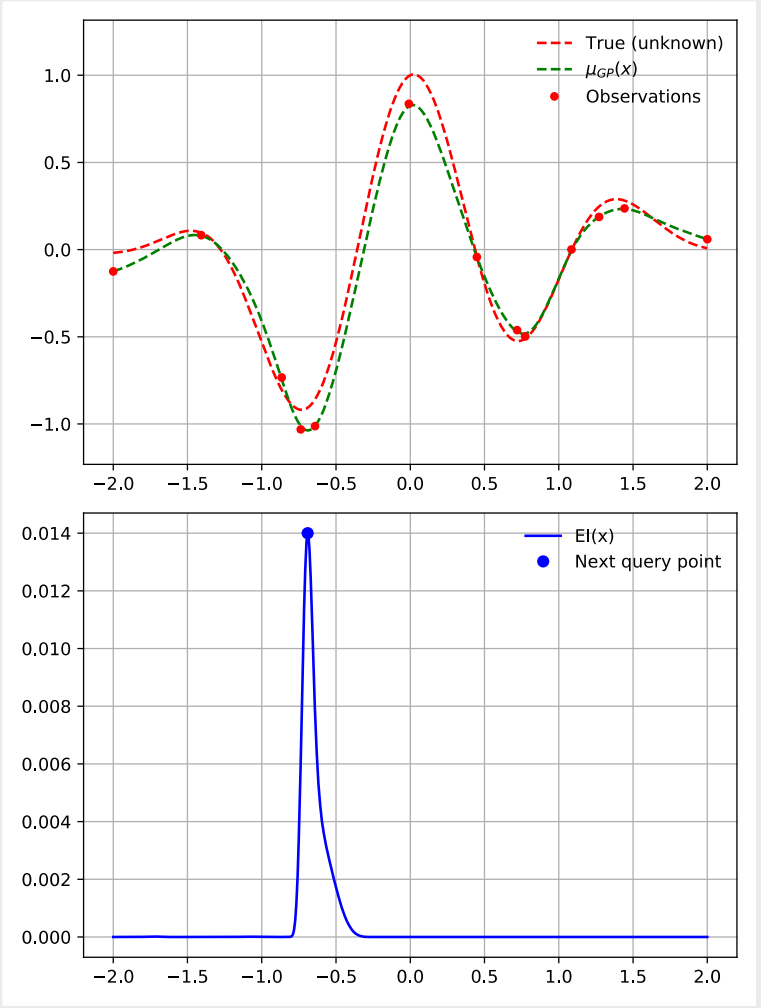
Bayesian Optimization (3/5)

- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



Bayesian Optimization (3/5)

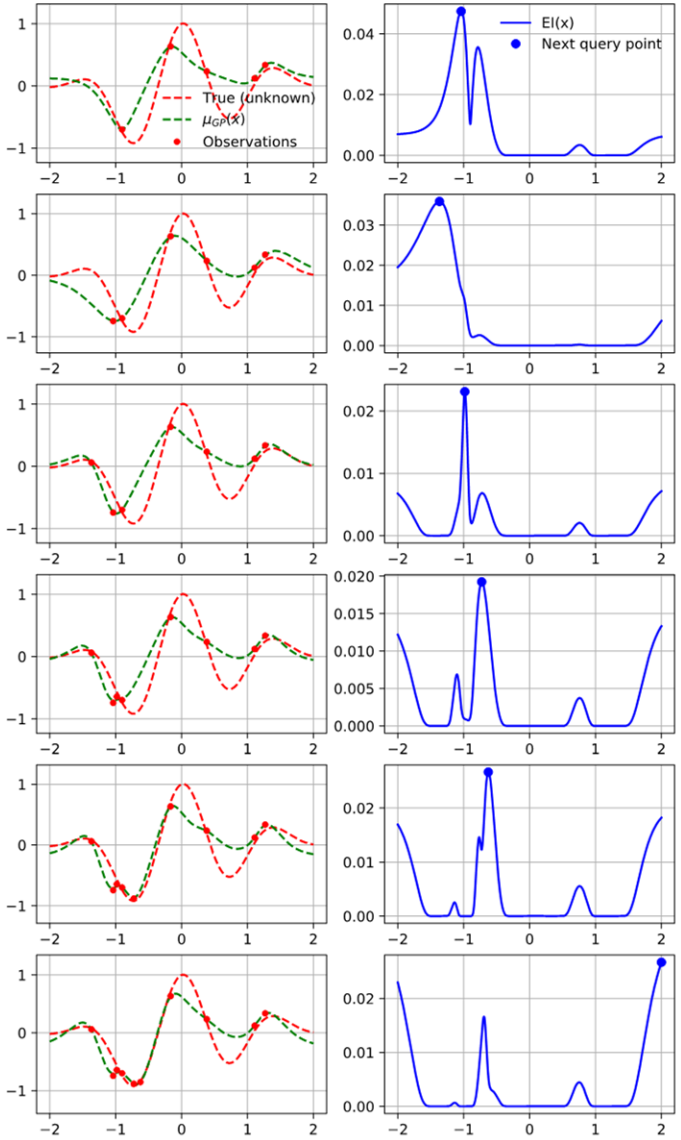
- Let's visualize the BO process
- In this example we have
  - a Gaussian Process as the surrogate model and
  - use EI as the acquisition function



Bayesian Optimization (4/5)

- Choices of surrogate model
  - Gaussian Process (GP)
    - Closed form
    - Runtime complexity: $O(n^3)$
  - Random Forest
    - Ensemble of decision trees
    - Faster than GP
    - Runtime complexity: $O(nlog(n))$
  - Bayesian Neural Network
    - NN with uncertainty estimates built-in
    - Very flexible
    - Requires more training data
  - Tree-structured Parzen Estimator (TPE)
    - Fast
    - Simple and non-parametric
    - Runtime complexity: $O(nlog(n))$

Example using Gaussian Process and EI



Bayesian Optimization (5/5)

Tree-structured Parzen Estimator (TPE) [3]

- Instead of modelling $p(y|x)$, model $p(x|y)$ as
- $p(x|y) = \begin{cases} l(x), if\ y < y^* \\ g(x), if\ y \geq y^* \end{cases}$
- $l(x)$,$(g(x))$, is a Kernel Density Estimator (KDE) formed by all data points $\{x_i\}$ resulting in better, (worse), performance than y^*
- y^* must be chosen to be worse than the best observed
- A common choice is to set y^* to the 15th percentile of all observed points
- Maximizing EI is equivalent to maximizing $\frac{l(x)}{g(x)}$

TPE advantages

- Simple and non-parametric
- Works well with mixed HP spaces
- Runtime complexity: $O(nlog(n))$

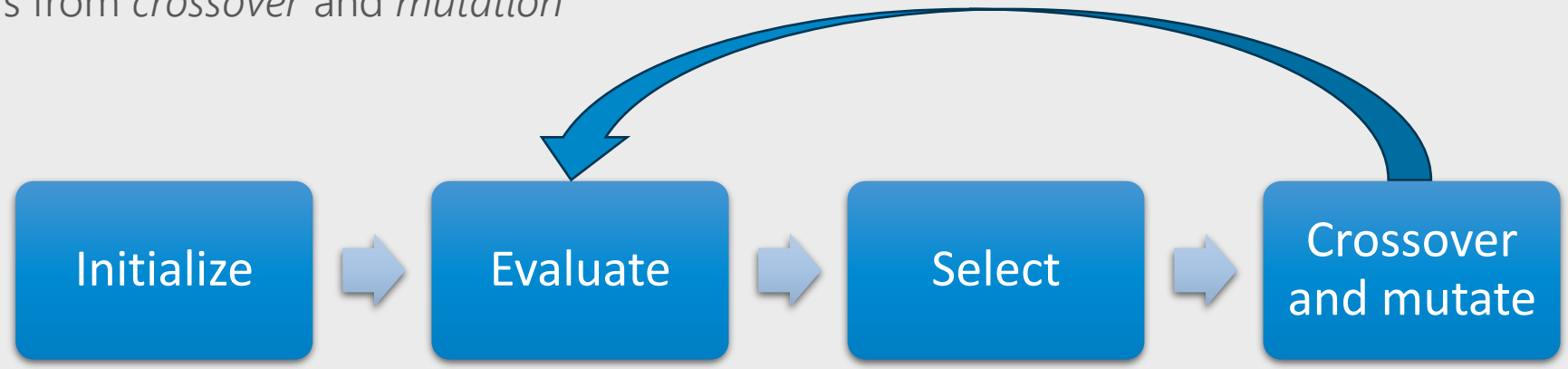
[3] James Bergstra, Rémi Bardenet, Yoshua Bengio and Balázs Kégl, Algorithms for Hyper-Parameter Optimization (2011), *Advances in Neural Information Processing Systems,* https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

Parallel Bayesian Optimization

- BO as discussed up until now is sequential, it waits for an evaluation to complete before selecting a new set of HPs to try
- With modern computing and HPC, we can run many trials in parallel
  - Must ensure to never evaluate same θ more than once since that would be very inefficient
- One strategy is to
  - Evaluate some given number of trials to get a set of observations
  - Pick next θ as described previously
  - If more recourses are available, modify acquisition function to penalize θs that are currently being evaluated but haven't completed yet
    - One way of doing this is by reducing the variance of the surrogate model at those points, θ

Evolutionary Search

- Evolutionary Search (ES) uses mechanisms inspired by biological evolution such as reproduction, mutation, recombination and selection
- A generic ES algorithm includes the following steps
  - Step 1: Train a few initial, random, models
  - Step 2: Repeat the following
    - Use the objective function to evaluate the *fitness* of each model
    - *Select* the fittest models for *reproduction*
    - Create new individuals from *crossover* and *mutation*



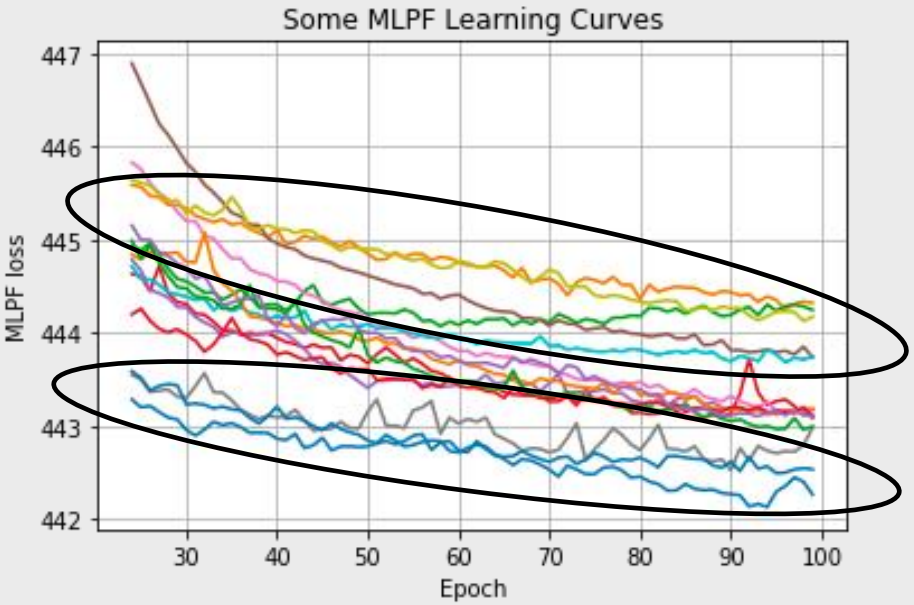
Adaptive configuration evaluation

Adaptive configuration evaluation

- It is possible to identify badly performing trials early so why train them to convergence?
- Adaptive configuration evaluation strategies terminate badly performing trials early

Some examples include:

- Successive Halving Algorithm (SHA)
  - Terminate some fraction of trials according given stopping rate *s*
- Hyperband
  - Loop over SHA using different stopping rates *s*
- Asynchronous Successive Halving Algorithm (ASHA)
  - Async version of SHA



Successive Halving Algorithm – SHA

Bracket

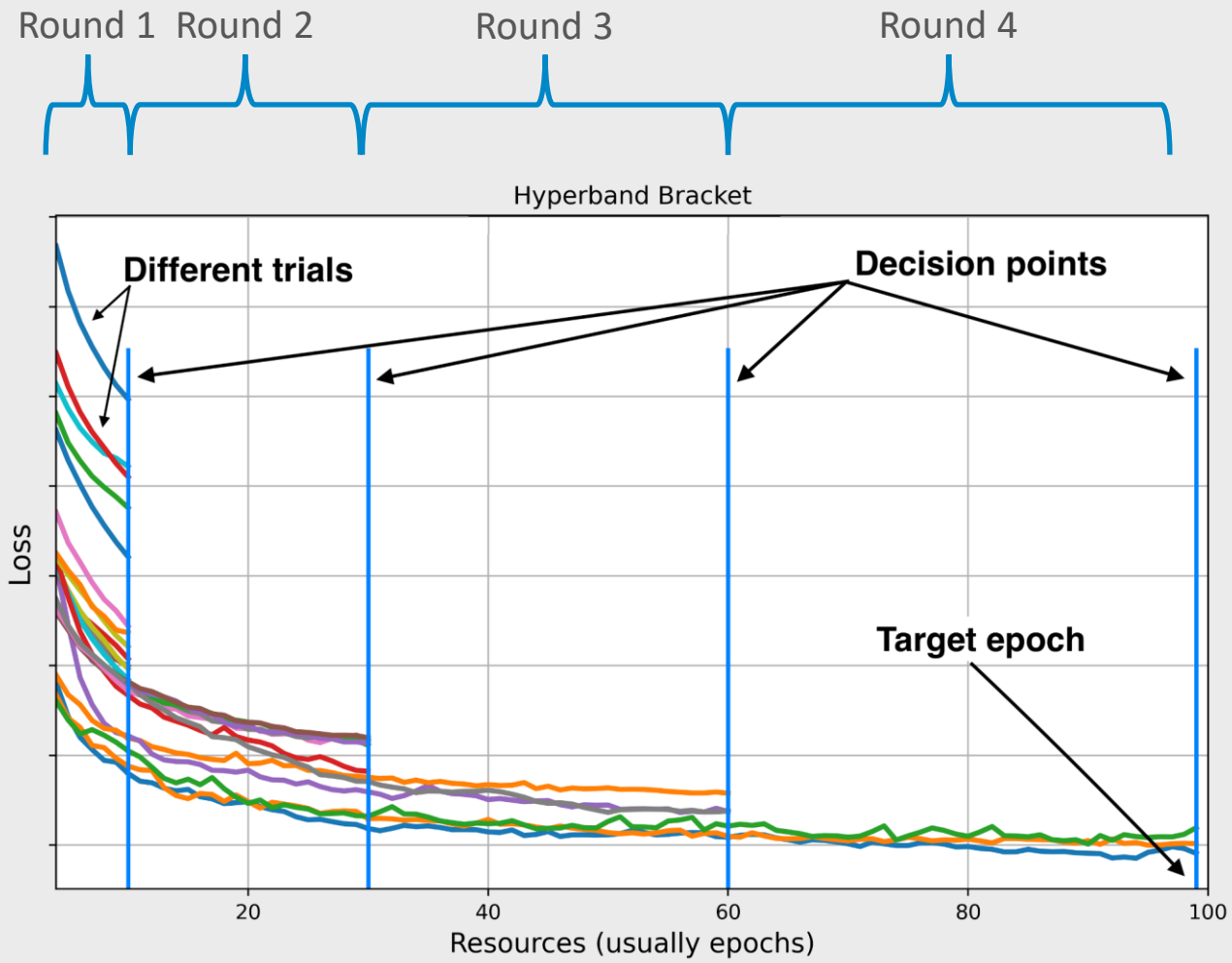
Round

1. Partially train some trials up to a decision point
2. Evaluate performance and throw out worst x%
3. Repeat 1-2 until target epoch is reached or only 1 trial remains

Hyperband adds a 4th step

4. Repeat 1-3 for different sets of decision points

- Drawback: sensitive to stragglers
  - All trials within a round must complete before proceeding to next round
  - Not suitable for large-scale HPC runs



Successive Halving Algorithm – SHA

- SHA [4] requires
  - Number of HP configs, n
  - Max resource, R
  - Min resource, r
  - Reduction factor, η
  - Min early-stopping rate, s
- s determines how many rounds we do, higher s means less rounds
- In this example we have
  $n = 8, R = 100\%, r = 12.5\%, \eta = 2, s = 0$
- Drawback: sensitive to stragglers
  - All trials within a round must complete before proceeding to next round
  - Not suitable for large-scale HPC runs

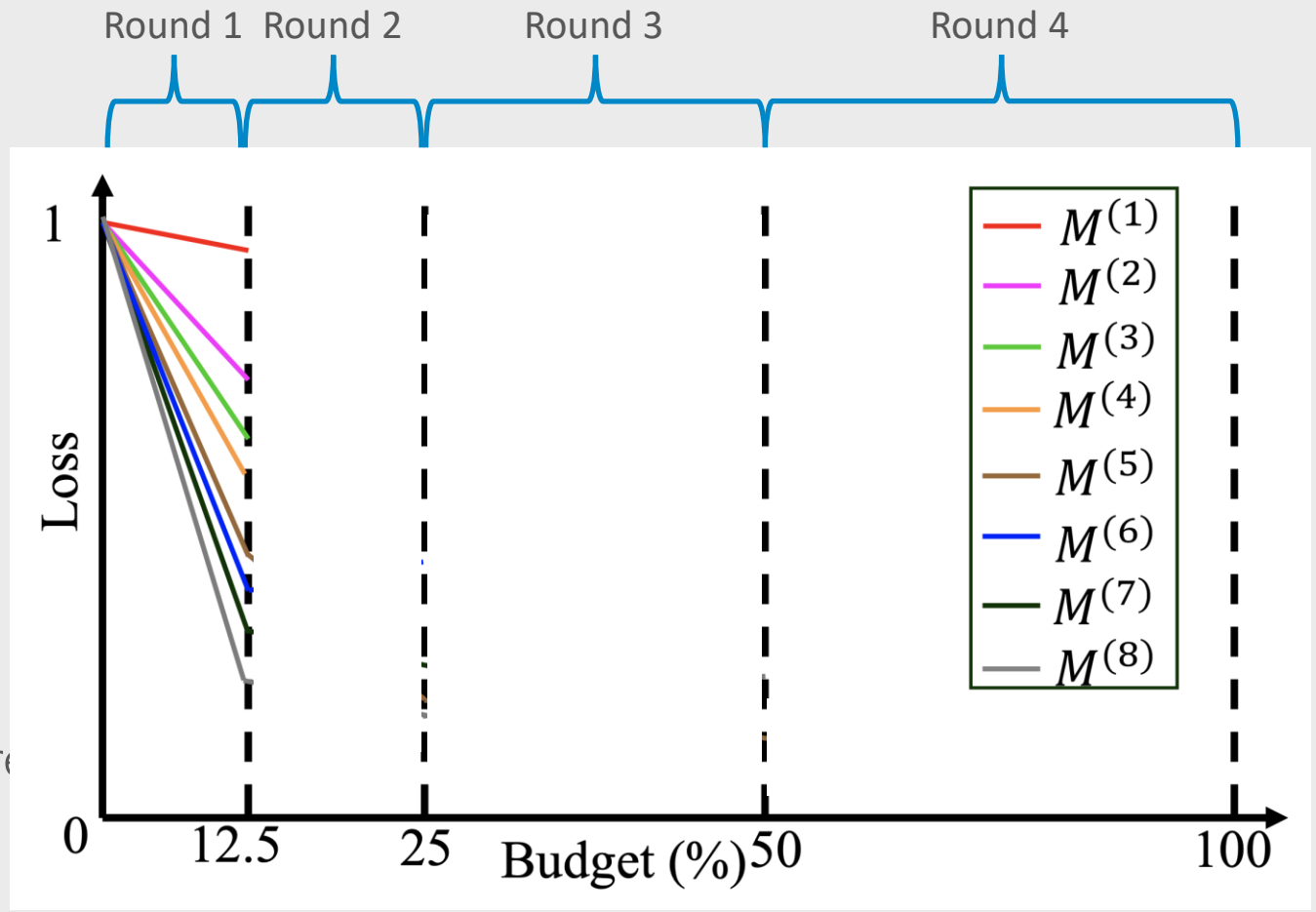


Figure from: Yazdani Abyaneh, Amir Hossein & Krunz, Marwan. (2022). Automatic Machine Learning for Multi-Receiver CNN Technology Classifiers

Successive Halving Algorithm – SHA

Algorithm 1 Successive Halving Algorithm.

input number of configurations n, minimum resource r, maximum resource R, reduction factor η, minimum early-stopping rate s

$s_{\max} = \lfloor \log_\eta(R/r) \rfloor$

assert $n \geq \eta^{s_{\max}-s}$ so that at least one configuration will be allocated R.

$T = \texttt{get\_hyperparameter\_configuration}(n)$

`// All configurations trained for a given` i `constitute a ''rung.''`

for $i \in \{0, \ldots, s_{\max} - s\}$ **do**

$n_i = \lfloor n\eta^{-i} \rfloor$

$r_i = r\eta^{i+s}$

$L = \texttt{run\_then\_return\_val\_loss}(\theta, r_i) : \theta \in T$

$T = \texttt{top\_k}(T, L, n_i/\eta)$

end for

return best configuration in T

SHA algorithm as defined by Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht and Ameet Talwalkar, Massively Parallel Hyperparameter Tuning, *Proceedings of Machine Learning and Systems* (2018) https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf

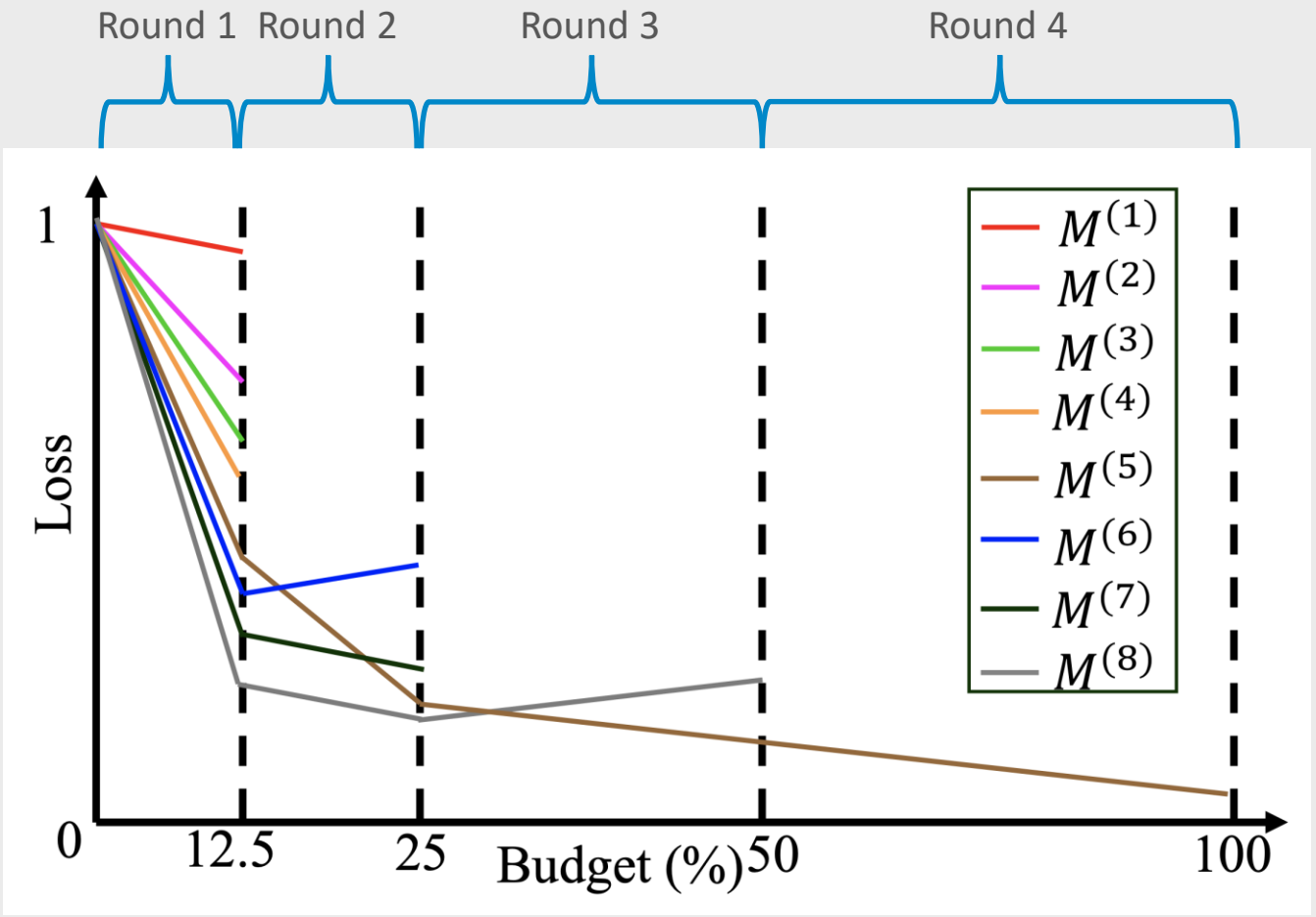


Figure from: Yazdani Abyaneh, Amir Hossein & Krunz, Marwan. (2022). Automatic Machine Learning for Multi-Receiver CNN Technology Classifiers

Hyperband

- Hyperband [5] loops over SHA using different stopping rates s
  - One less parameter to choose
- Choice of s can be important since learning dynamics is highly problem dependent
- Potentially finds better solutions compared to SHA (at cost of additional compute)
- Sensitive to stragglers

Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization

Lisha Li LISHAL@CS.CMU.EDU
Carnegie Mellon University, Pittsburgh, PA 15213
Kevin Jamieson JAMIESON@CS.WASHINGTON.EDU
University of Washington, Seattle, WA 98195
Giulia DeSalvo GIULIAD@GOOGLE.COM
Google Research, New York, NY 10011
Afshin Rostamizadeh ROSTAMI@GOOGLE.COM
Google Research, New York, NY 10011
Ameet Talwalkar TALWALKAR@CMU.EDU
Carnegie Mellon University, Pittsburgh, PA 15213
Determined AI

Editor: Nando de Freitas

Abstract

Performance of machine learning algorithms depends critically on identifying a good set of hyperparameters. While recent approaches use Bayesian optimization to adaptively select configurations, we focus on speeding up random search through adaptive resource allocation and early-stopping. We formulate hyperparameter optimization as a pure-exploration non-stochastic infinite-armed bandit problem where a predefined resource like iterations, data samples, or features is allocated to randomly sampled configurations. We introduce a novel algorithm, HYPERBAND, for this framework and analyze its theoretical properties, providing several desirable guarantees. Furthermore, we compare HYPERBAND with popular Bayesian optimization methods on a suite of hyperparameter optimization problems. We observe that HYPERBAND can provide over an order-of-magnitude speedup over our competitor set on a variety of deep-learning and kernel-based learning problems.

[5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar, *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*, (2016) https://arxiv.org/abs/1603.06560

Asynchronous Successive Halving Algorithm – ASHA

- ASHA's [6] asynchronous nature eliminates straggler problem
  - Promotes trials to next round whenever possible
  - If no promotions are possible, initiate new trials
  - Number of erroneously promoted trials expected to be small for large n
- The authors provide defaults which they claim work well for a wide variety of problems
  - $s = 0, \eta = 4$
- Run for as long as you like
  - Common stopping criteria are the number of trials to evaluate, n, or reaching a given wall time
- Can be used to implement Asynchronous Hyperband by looping over values of s

Algorithm 2 Asynchronous Successive Halving (ASHA)

input minimum resource r, maximum resource R, reduction factor η, minimum early-stopping rate s

function `ASHA()`
  repeat
    for for each free worker **do**
      $(\theta, k) =$ `get_job()`
      `run_then_return_val_loss`$(\theta, r\eta^{s+k})$
    end for
    for completed job (θ, k) with loss l **do**
      Update configuration θ in rung k with loss l.
    end for
  until desired
end function

function `get_job()`
  `// Check if there is a promotable config.`
  for $k = \lfloor \log_\eta(R/r) \rfloor - s - 1, \ldots, 1, 0$ **do**
    candidates $=$ `top_k`(rung k, $|\text{rung } k|/\eta$)
    promotable $= \{t \in \text{candidates} : t \text{ not promoted}\}$
    if $|\text{promotable}| > 0$ **then**
      return promotable$[0], k + 1$
    end if
    `// If not, grow bottom rung.`
    Draw random configuration θ.
    return $\theta, 0$
  end for
end function

ASHA algorithm as defined by Lisha Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht and Ameet Talwalkar, Massively Parallel Hyperparameter Tuning, *Proceedings of Machine Learning and Systems* (2018) https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf

Some practical tips

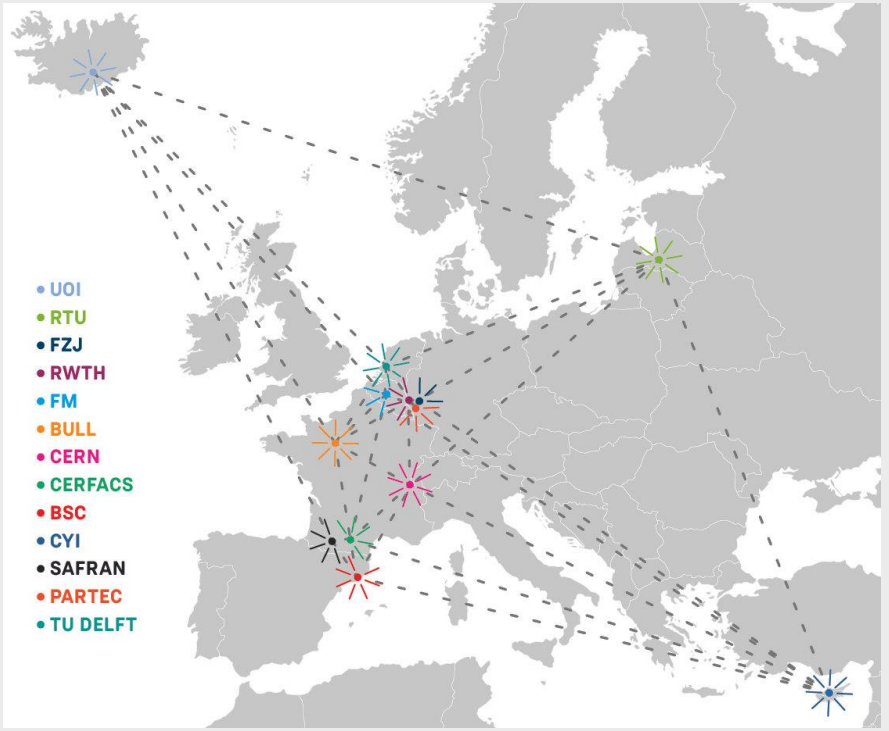
- Don’t use a too narrow search space
- Don’t use grid-search – very inefficient
  - Random search is just as easy to implement but much more efficient
- Eliminate HPs that aren’t important
- Coarse to fine search
- Use appropriate scale when searching for HP values
  - The sensitivity of the LR or the momentum parameter is much higher in certain ranges (i.e., when LR is small, or beta is close to 1)
- Use three dataset splits, train, test, validation, one of which is only used for evaluation after completed HPO

Hyperparameter Optimization in CoE RAISE

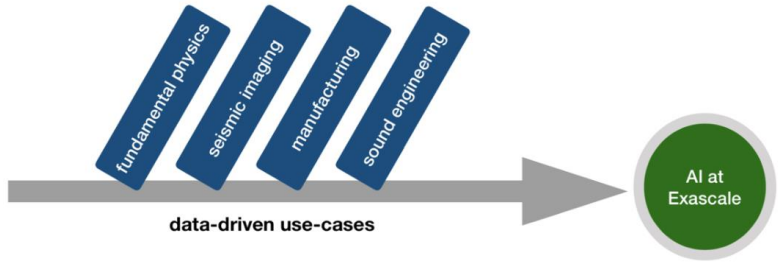
CoE RAISE and Work Packages 2 & 4

- CoE RAISE: Center of Excellence for Research on AI- and Simulation-based Engineering at Exascale
  - Develop novel, scalable Artificial Intelligence technologies
  - Use-cases from Engineering and Natural Sciences
- CERN (Dr. M. Girone) leads WP4: ***Data-Driven Use-Cases towards Exascale***
  - Including Task 4.1 (E. Wulff): *Event reconstruction and classification at the CERN HL-LHC*, more on this shortly
- UOI (Prof. M. Riedel) leads WP2: ***AI- and HPC-Cross Methods at Exascale***
  - Provides expert support on HPC and AI methods to use-cases in WP4

CoE RAISE Partners

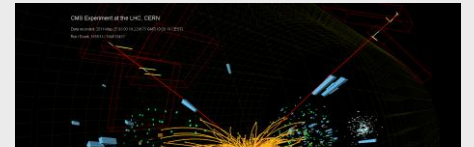


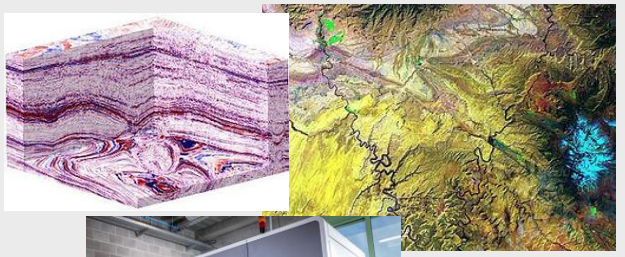
WP4 use-cases



RAISE Center of Excellence

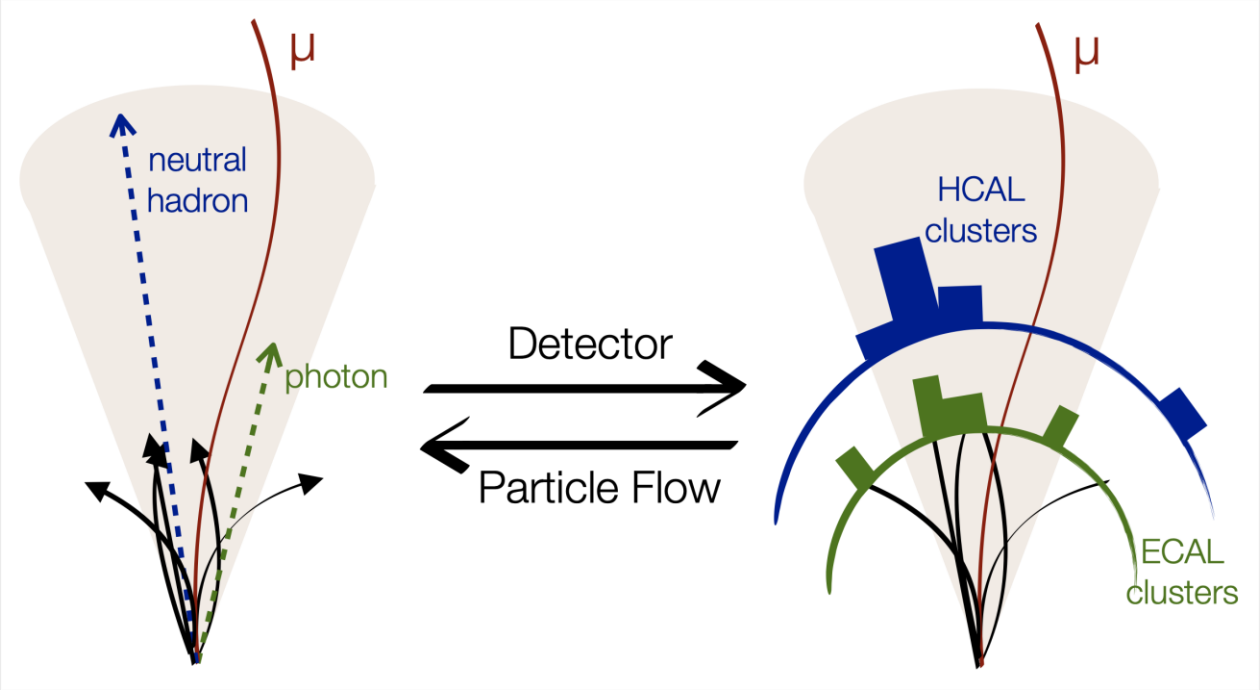
- Representative use-cases from research and industry/SMEs, which have a strong focus on ***data-driven*** technologies, i.e., analyzing data-rich descriptions of physical phenomena
  - *Event reconstruction and classification at the CERN HL-LHC (CERN, RTU)*
    - develop novel approaches for HL-LHC collision event reconstruction replacing traditional algorithms with AI-driven techniques towards HPC-to-Exascale
  - *Seismic imaging with remote sensing for energy applications (FZJ, UOI, CYI)*
    - optimize seismic imaging and remote sensing, enabling AI approaches, combining satellite and airborne data with seismic imaging
  - *Defect-free metal additive manufacturing (UOI, FM)*
    - develop prediction models that detect porosity inside metal parts such that the information is exploited to improve the product quality in additive manufacturing
  - *Sound engineering (FZJ, UOI)*
    - develop a deep-learning-based algorithm that associates individual anatomy to a head-related transfer function (HRTF), for use in spatial audio systems



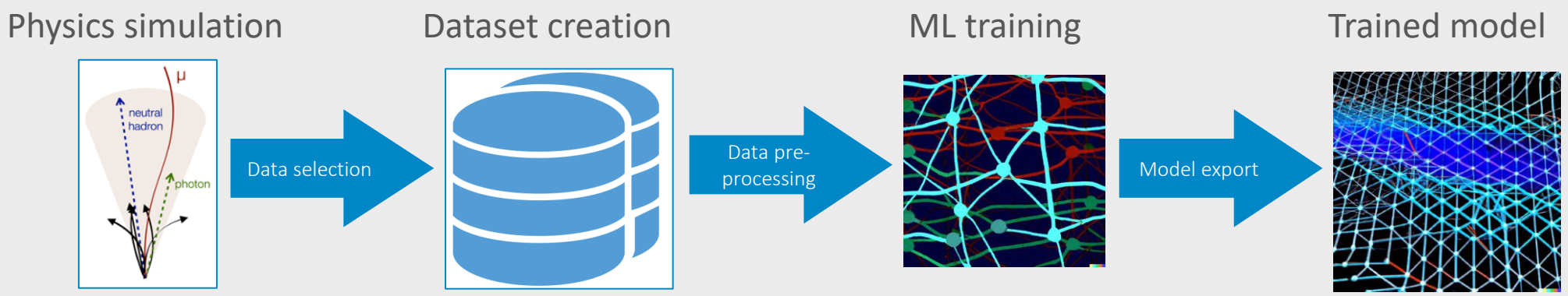


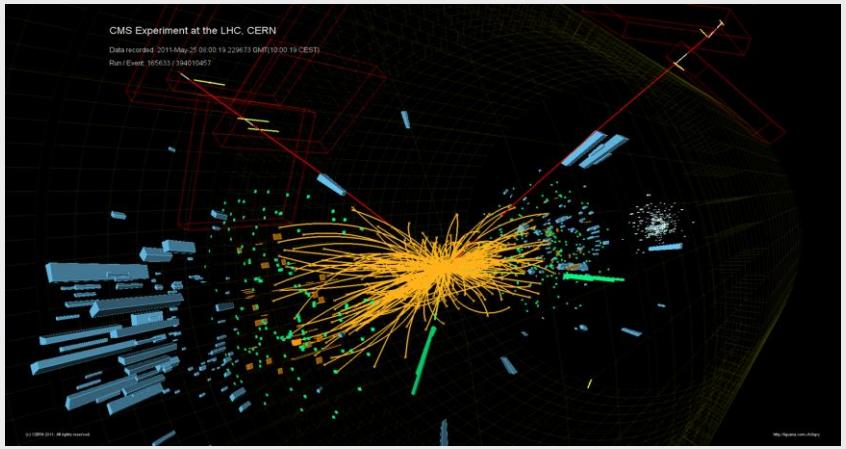
Event reconstruction at the LHC

- Event reconstruction attempts to solve the inverse problem of particle-detector interactions, i.e., going from detector signals back to the particles that gave rise to them
- Particle-flow (PF) reconstruction takes tracks and clusters of energy deposits as input and gives particle types and momenta as output



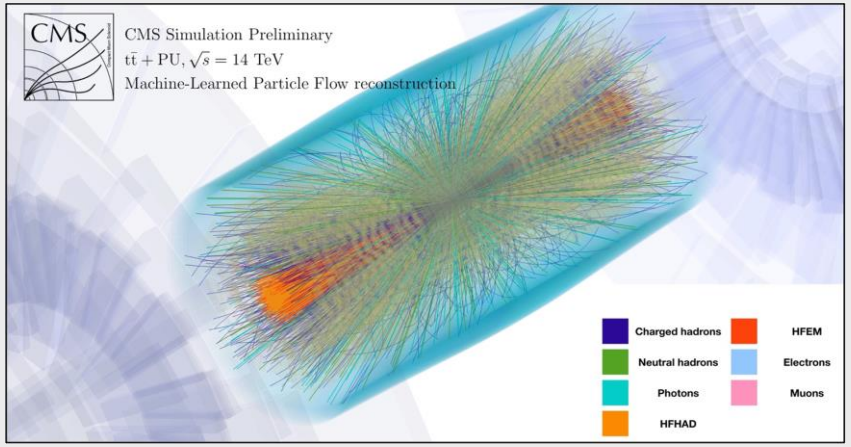
AI-based particle flow reconstruction workflow





CMS Collision event

Event reconstruction



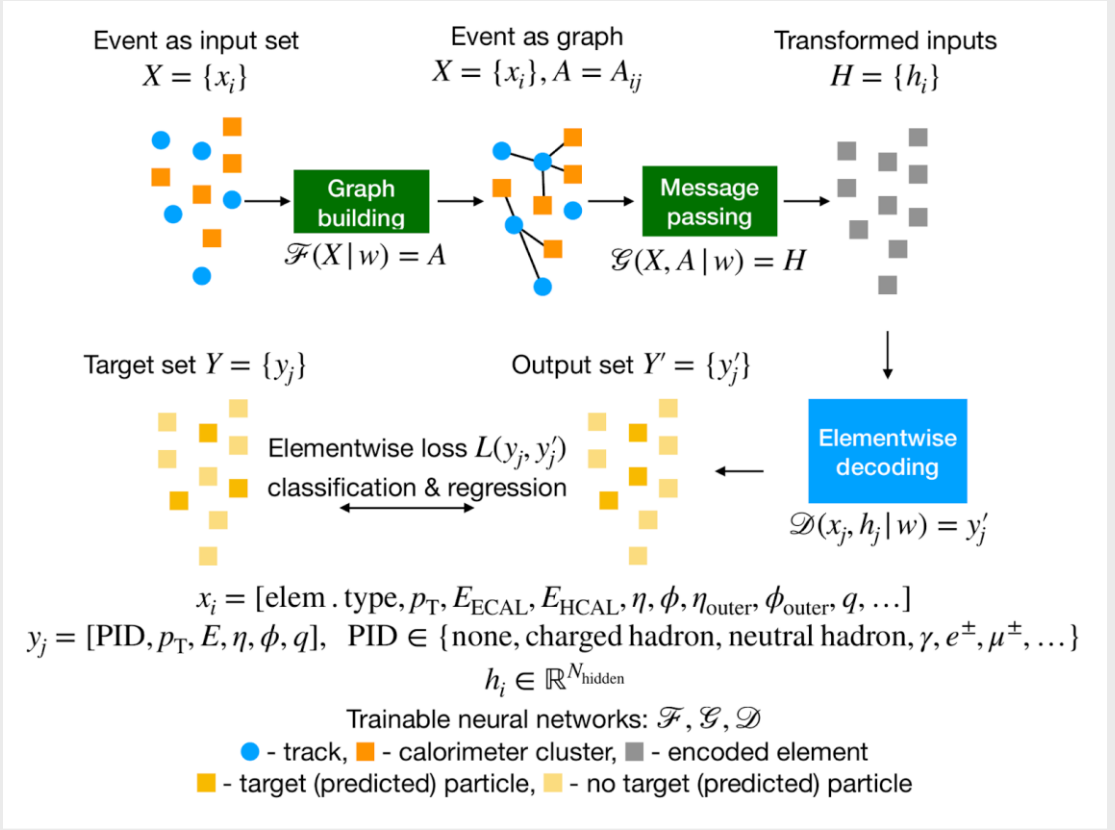
MLPF event reconstruction [7]

[7] Pata, J., Duarte, J., Mokhtar, F., Wulff, E., Yoo, J., Vlimant, J.-R., Pierini, M., Girone, M. (2022). *Machine Learning for Particle Flow Reconstruction at CMS*. Retrieved from http://arxiv.org/abs/2203.00330

Machine-Learned Particle-Flow (MLPF)

- The Particle Flow (PF) Algorithm [8]
  - Tries to identify and reconstruct all stable individual particles from collision events by combining information from different subdetectors (tracks, calorimeter clusters)
- Machine-Learned Particle-Flow (MLPF) [9]
  - GPU accelerated, GNN-based algorithm for PF
  - Code available on GitHub
  - See ACAT2021 talk by J. Pata (and proceedings) for more MLPF model details and ACAT 2021 talk by E. Wulff (and proceedings) for more details on the hypertuning of MLPF
  - See ACAT2022 poster for latest results

The MLPF model



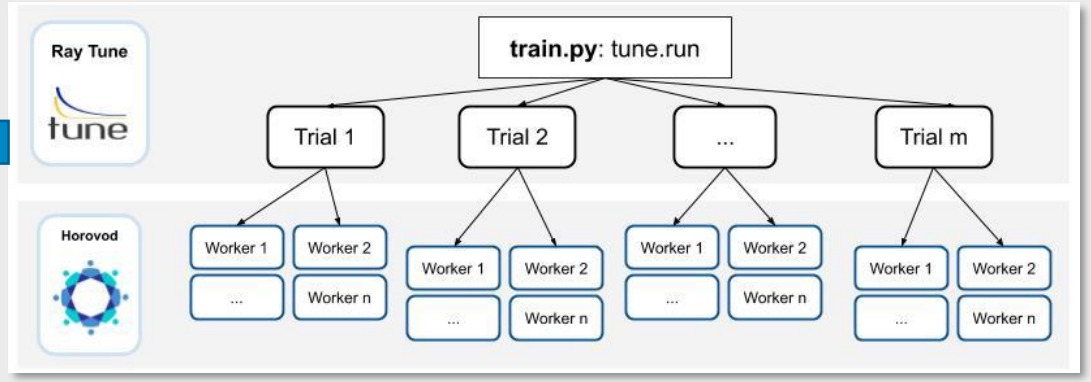
Based on Eur. Phys. J. C 81, 381 (2021)
https://arxiv.org/abs/2101.08578

[8] CMS Collaboration https://cds.cern.ch/record/1194487?ln=en

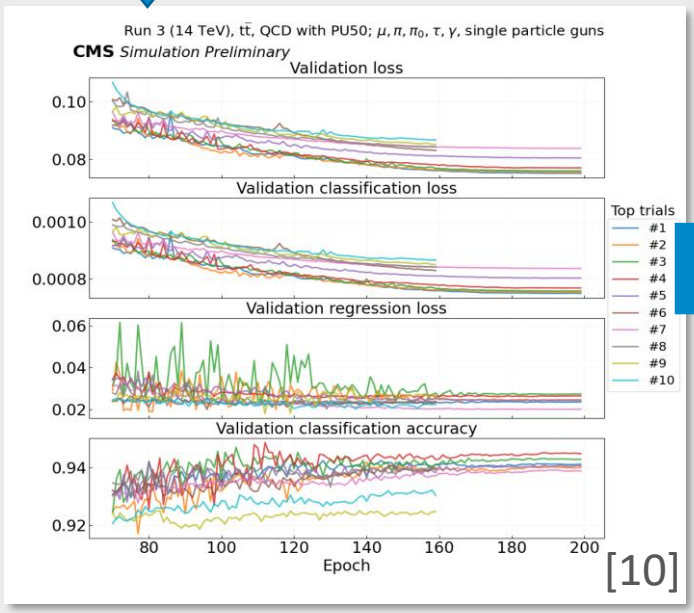
[9] Pata, J., Duarte, J., Vlimant, JR. *et al.* MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. *Eur. Phys. J. C* **81,** 381 (2021). https://doi.org/10.1140/epjc/s10052-021-09158-w

Large-scale distributed Hyperparameter Optimization

Distributed HPO

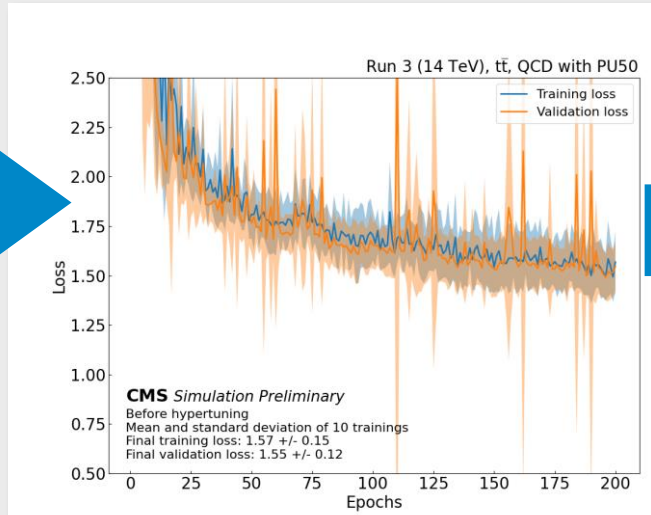


Model selection

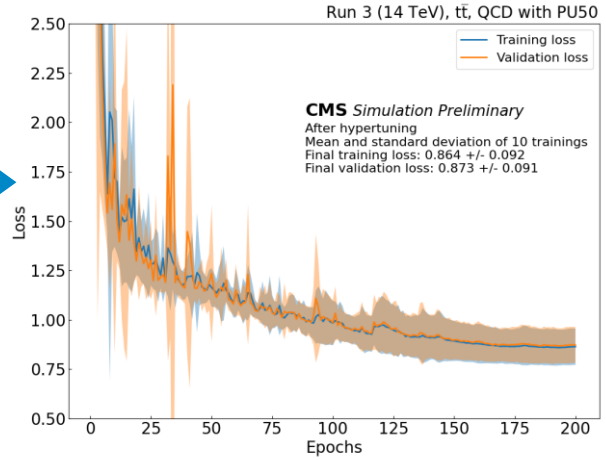


[10]

Assess learning variability



Better learning



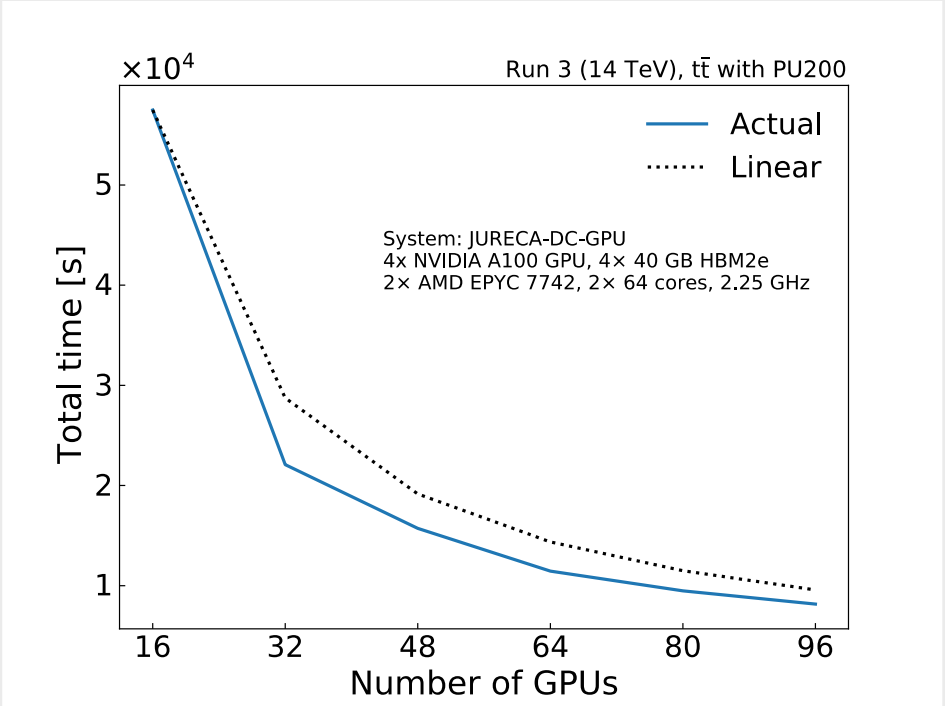
[10]

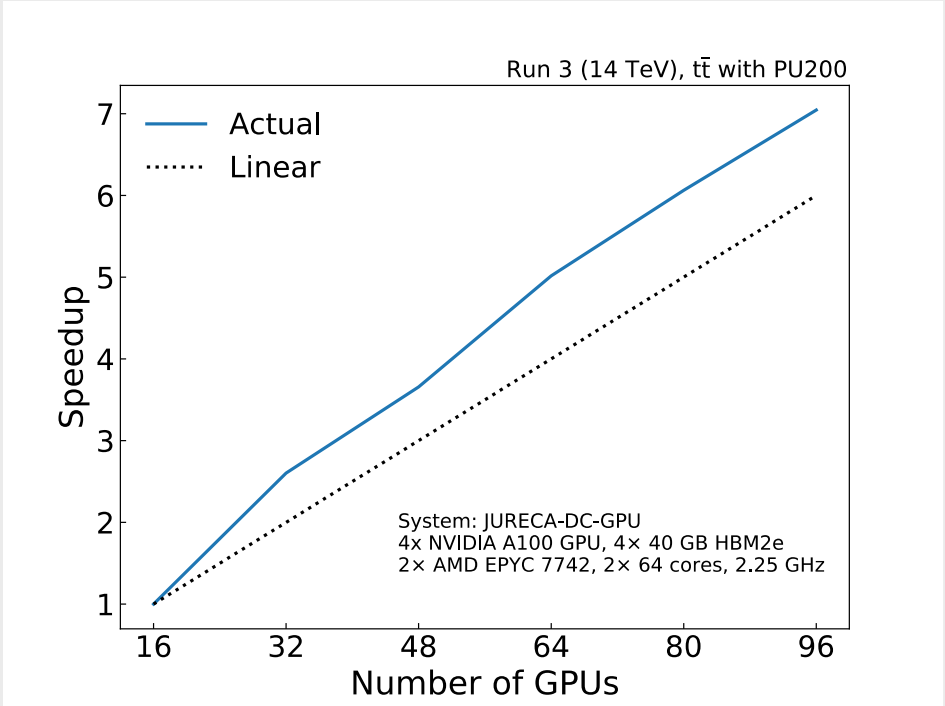
- Using ASHA + Bayesian Optimization
- Scalable up to hundreds of GPUs
- Mean validation loss decreased by ~44% giving a significant performance improvement

[10] E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

Scaling of MLPF hypertuning on multiple compute nodes

- Scaling of a hypertuning run of MLPF on the JURECA-DC-GPU system at the Jülich Supercomputer Centre (JSC), 4 NVIDIA A100 and 2× 64 cores AMD EPYC 7742 per node
- **Superlinear scaling** due to excessive re-loading of models when using fewer nodes
- Using the ASHA algorithm to schedule and terminate trials, in combination with Bayesian optimization





Data used: [11] *Simulated particle-level events of ttbar and QCD with PU200 using Pythia8+Delphes3 for machine learned particle flow (MLPF),* https://doi.org/10.5281/zenodo.4559324

Hypertuning tool of choice: Ray Tune

- Open-source tool for multi-node distributed hyperparameter optimization
- Many built-in SOTA search algorithms
  - ASHA
  - Hyperband
  - Bayesian Optimization
  - Population Based Training
- Supports TensorFlow, PyTorch and others
- Supports integration of many other hypertuning tools such as Scikit-Optimize, HyperOpt, Optuna, SigOpt, etc.

Using Ray Tune on SLURM clusters

- Ray expects a head-worker architecture with a single point of entry
  - Start a head node and multiple worker nodes before running the Ray Tune script on the head node
- Sometimes tricky to set up but once done, it works great


```
#!/bin/sh

#SBATCH ...
#SBATCH ...

# Get the node names
nodes=$(scontrol show hostnames $SLURM_JOB_NODELIST)
nodes_array=( $nodes )

# Get the head node
node_1=${nodes_array[0]}
ip=$(srun --nodes=1 --ntasks=1 -w $node_1 host ${node_1}i | awk '{ print $4 }')
port=6379
ip_head=$ip:$port
echo "IP Head: $ip_head"

echo "STARTING HEAD at $node_1"
srun --nodes=1 --ntasks=1 -w $node_1 \
  ray start --head --node-ip-address="$node_1"i --port=$port \
  --num-cpus "${SLURM_CPUS_PER_TASK}" --num-gpus "${SLURM_GPUS_PER_TASK}" --block &
sleep 5

worker_num=$(($SLURM_JOB_NUM_NODES - 1)) #number of nodes other than the head node
for (( i=1; i<=$worker_num; i++ ))
do
  node_i=${nodes_array[$i]}
  echo "STARTING WORKER $i at $node_i"
  srun --nodes=1 --ntasks=1 -w ${node_i} \
    ray start --address "$node_1"i:"$port" \
    --num-cpus "${SLURM_CPUS_PER_TASK}" --num-gpus "${SLURM_GPUS_PER_TASK}" --block &
  sleep 5
done

# Run the Ray Tune script
python3 tune_script.py --cpus "${SLURM_CPUS_PER_TASK}" --gpus "${SLURM_GPUS_PER_TASK}"
exit
```

Hypertuning MLPF on HPC systems

- Thanks to Forschungszentrum Jülich (FZJ), San Diego Supercomputing Center (SDSC), Flatiron Institute (collaboration with CMS and CERN openlab)
- Using multiple compute nodes with 4 GPUs per node
  - Both systems: 4 NVIDIA A100 40GB per node
  - @CoreSite: 64 core Intel Icelake Platinum 8358
  - @JUWELS: 2x 24 core AMD EPYC Rome 7402
- We did 2 stages of hypertuning:
  - BOHB [12] - Bayesian Optimization combined with Hyperband – using JUWELS Booster
  - ASHA [2] + Bayesian Optimization [3] – using CoreSite
  - ~76000 core-hours in total
- Back of the envelope calculation shows that it would have taken ~6 months on a single GPU instead of ~83 hours using HPC systems

Run in part on the JUWELS Booster [2] Image: JÜLICH Forschungszentrum

```
search_space = {
   "lr": loguniform(1e-4, 3e-2),
   "expdecay_decay_steps": quniform(10, 2000, 10),
   "dropout": uniform(0.0, 0.5),
   "clip_value_low": uniform(0.0, 0.2),
   "dist_mult": uniform(0.01, 0.2),
}
```

```
search_space = {
       "layernorm": samp([False, True]),
       "ffn_dist_hidden_dim": samp([32, 64, 128, 256]),
       "ffn_dist_num_layers": samp([1, 2, 3, 4]),
       "distance_dim": samp([32, 64, 128, 256]),
       "num_node_messages": samp([1, 2, 3, 4]),
       "num_graph_layers_common": samp([1, 2, 3, 4]),
       "num_graph_layers_energy": samp([1, 2, 3, 4]),
       "bin_size": samp([16, 32, 40, 64, 80]),
       "normalize_degrees": samp([True, False]),
       "output_dim": samp([32, 64, 128, 256]),
}
```

HPO studies done in: [10] E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

[12] Falkner S, Klein A and Hutter F BOHB: robust and efficient hyperparameter optimization at scale (2018), https://arxiv.org/abs/1807.01774

HPO results

- Using BOHB, a set of best hyperparameters were found in the first search space
- ASHA + Bayesian Optimization (BO) were used in the second, larger, search space
- Final best hyperparameters shown below
- Variability check: best trial was re-trained 10 times and achieved similar results

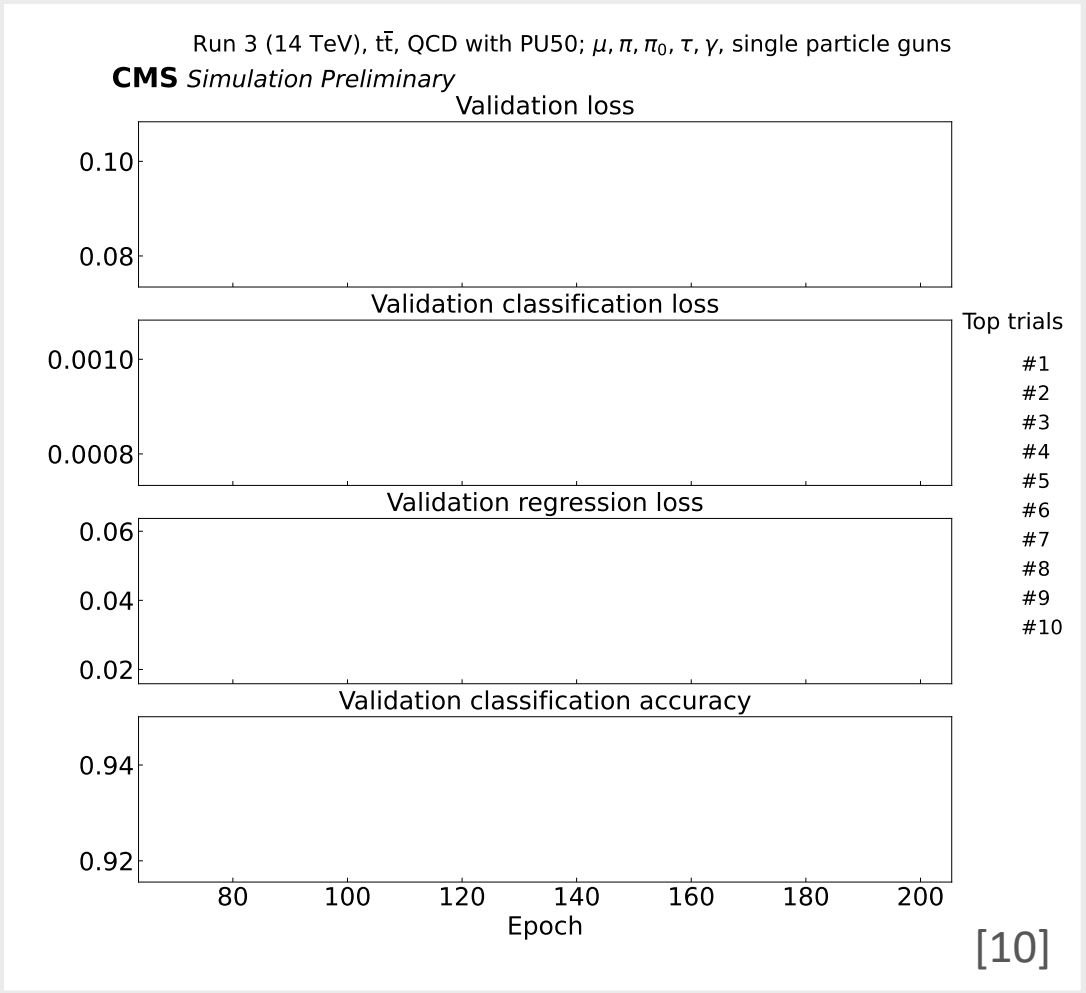
BOHB

```
best_config =
{'clip_value_low': 0.001998,
 'dist_mult': 0.120898,
 'dropout': 0.016312,
 'lr': 0.001129}
```

ASHA + BO

```
best_config =
{'bin_size': 64,
 'distance_dim': 64,
 'ffn_dist_hidden_dim': 128,
 'ffn_dist_num_layers': 3,
 'layernorm': 'False',
 'normalize_degrees': 'True',
 'num_graph_layers_common': 3,
 'num_graph_layers_energy': 2,
 'num_node_messages': 3,
 'output_dim': 64}
```

From the final ASHA + BO search

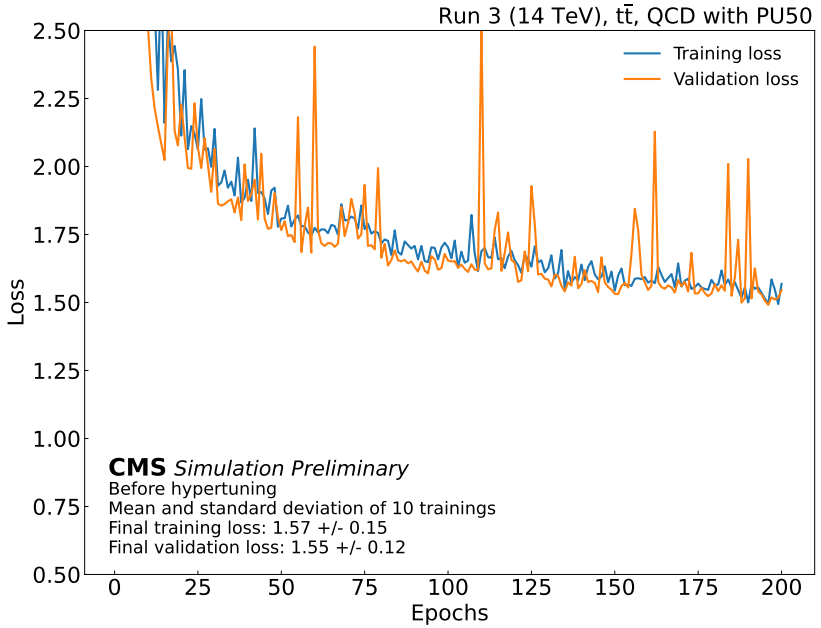


[10]

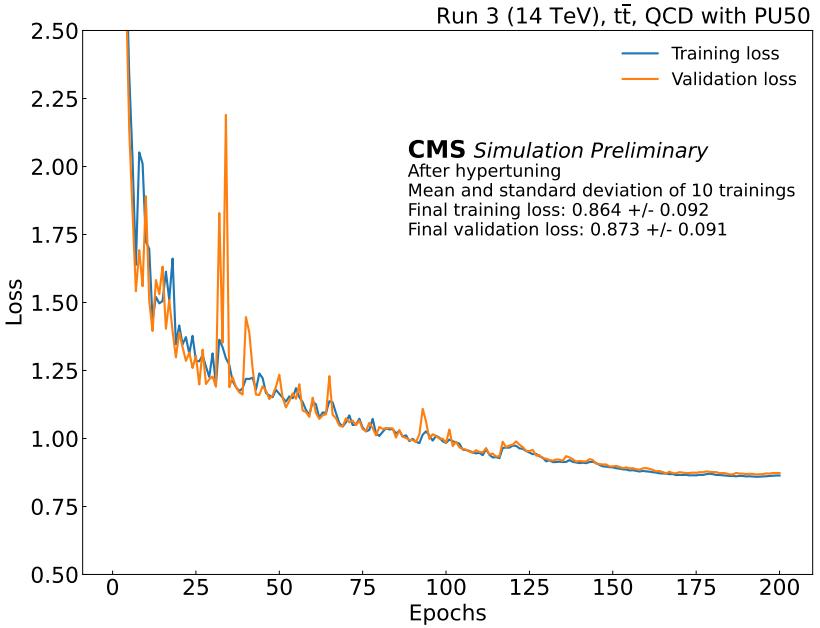
[10] E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

Improvements from HPO

- Loss curves before (left) and after (right) hypertuning
- Only the physical datasets, no single particle gun samples
- Mean and standard deviation of 10 trainings with identical hyperparameters
- Mean validation loss decreased by ~44%



[1]



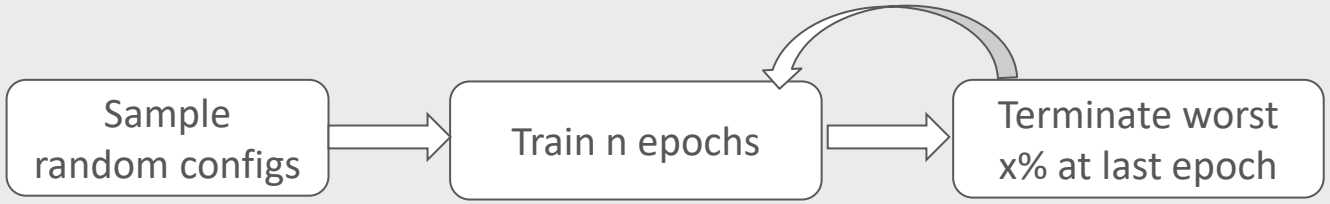
[1]

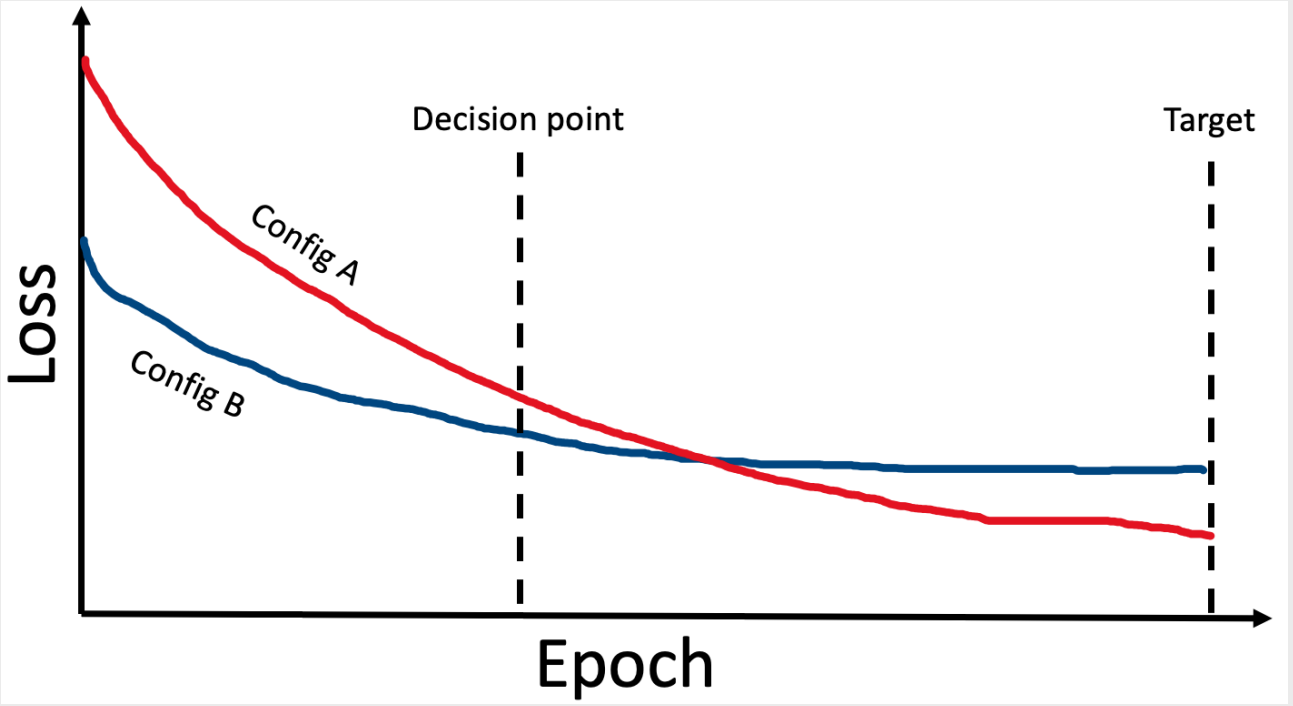
HPO studies done in 2021 in: E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

Quantum-assisted HPO in CoE RAISE

Model performance prediction using QSVR

- Current STOTA hypertuning algorithms rely on early stopping
- Stopping criterion: ranking according to a single metric (e.g., validation loss)
- Potential problem: loss curves are not linear
- Idea 1: Use a non-linear stopping criterion
  - For instance, an SVR model, inspired by [1]
- Idea 2: Use quantum computing to fit a Quantum-SVR (QSVR)

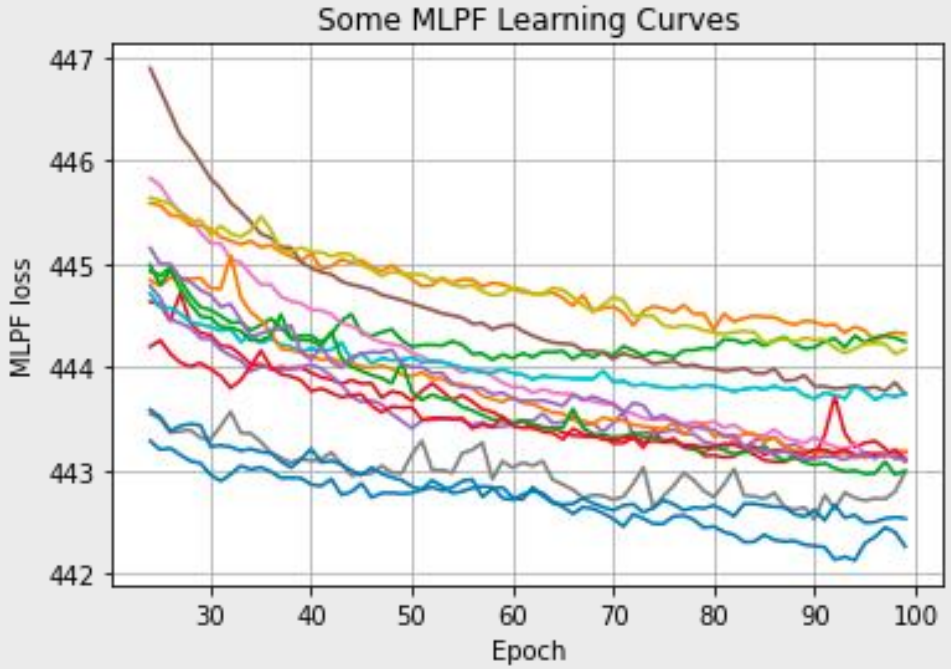




Approach inspired by [13] Bowen Baker, Otkrist Gupta, Ramesh Raskar, Nikhil Naik, Accelerating Neural Architecture Search using Performance Prediction (2017) https://arxiv.org/abs/1705.10823

Dataset creation

- Generated dataset consisting of learning curves and HP configs
  - Run 300 MLPF trainings
  - For each training, sample HPs from a 7-dimensional search space
  - Train for 100 epochs on the publicly available Delphes dataset [11]
- Inputs:
  - HP configuration
  - Partial learning curve
  - 1st and 2nd order differences of the partial learning curve
- Targets
  - Final validation loss



[11] Pata J et al. Simulated particle-level events of t̄t and QCD with PU200 using PYTHIA8+DELPHES3 for machine learned particle flow (MLPF), (2021), https://zenodo.org/record/4559324

Accessing D-Wave Quantum Annealer in CoE RAISE

- A quantum annealer is a particular kind of quantum computer
  - Solves QUBO problems (Quadradic Unconstrained Binary Optimization)
- SVR can be formulated as a QUBO problem [14]
- The annealer returns multiple solutions
  - Quantum annealing is a stochastic process
- Challenges
  - We can only fit 20 training samples
  - Unstable results, quantum noise
  - Limited quantum computing time available

$$\text{Minimize:} \quad f(x) = \sum_{i<j}^{N} Q_{i,j} x_i x_j + \sum_{i}^{N} Q_{i,i} x_i$$

$x_i \in \{0, 1\}$ and Q is a $N \times N$ symmetric matrix

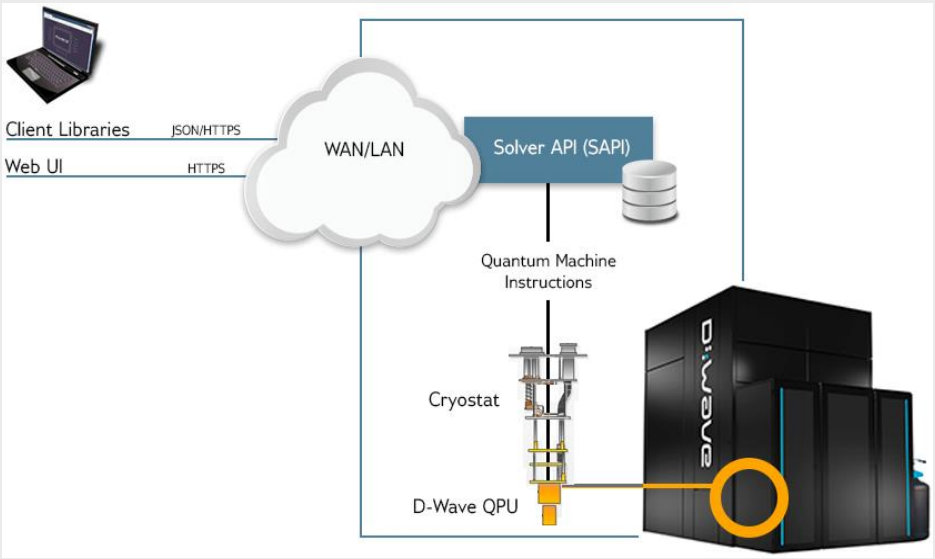


Image from D-Wave documentation

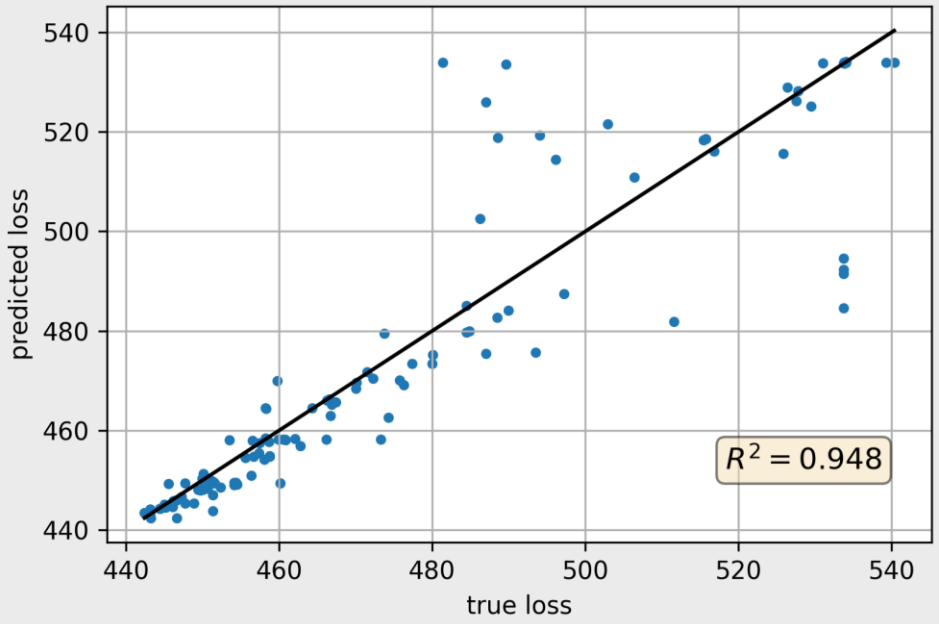
QSVR results

- Predicting final loss from fraction of loss curve (25%)
- QSVR results comparable to classical SVR and to simulated quantum annealing

R^2 scores

	Best	Worst	Mean	Std	Number of trainings
SVR	0.959	0.318	0.889	0.050	1000
Sim-QSVR	0.949	0.383	0.901	0.045	100
QSVR	0.948	0.742	0.880	0.056	10
QSVR Ensemble	0.927	0.857	0.899	0.019	10

Best QSVR results



Summary

Summary

- Hyperparameter optimization could benefit any data-driven AI-based algorithm
- CoE RAISE develops novel, scalable AI methods towards Exascale
  - Use-cases from a wide range of sciences and industry
- Large-scale distributed HPO significantly increased model performance in the example use-case of Machine-Learned Particle Flow (MLPF)
  - Would not have been possible without access to HPC resources
- The disruptive technology of Quantum Computing is already here and can be integrated in hybrid Quantum-HPC workflows
  - The technology is still very early-stage and is likely to improve greatly in the future

References

- [1] Gábor Melis, Chris Dyer and Phil Blunsom, On The State of The Art of Evaluation in Neural Language Processing (2017), https://arxiv.org/abs/1707.05589
- [2] James Bergstra and Yoshua Bengio: Random Search for Hyper-Parameter Optimization, *Journal of Machine Learning Research* 13 (2012) 281-305 https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf
- [3] James Bergstra, Rémi Bardenet, Yoshua Bengio and Balázs Kégl, Algorithms for Hyper-Parameter Optimization (2011), *Advances in Neural Information Processing Systems, https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf*
- [4] K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization (2015), *In International Conference on Artificial Intelligence and Statistics (AISTATS)*
- [5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, Ameet Talwalkar, Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization, (2016) https://arxiv.org/abs/1603.06560
- [6] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Katya Gonina, Moritz Hardt, Benjamin Recht and Ameet Talwalkar, Massively Parallel Hyperparameter Tuning, *Proceedings of Machine Learning and Systems* (2018) https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf
- [7] Pata J, Duarte J, Mokhtar F, Wulff E, Yoo J, Vlimant J, Pierini M and Girone M 2022 Machine learning for particle flow reconstruction at CMS, (2023) *J. Phys.: Conf. Ser. 2438 012100*, https://doi.org/10.1088/1742-6596/2438/1/012100
- [8] The CMS Collaboration, Particle-Flow Event Reconstruction in CMS and Performance for Jets, Taus, and MET (2009), https://cds.cern.ch/record/1194487
- [9] Pata, J., Duarte, J., Vlimant, JR. *et al.* MLPF: efficient machine-learned particle-flow reconstruction using graph neural networks. *Eur. Phys. J. C* **81,** 381 (2021). https://doi.org/10.1140/epjc/s10052-021-09158-w
- [10] E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092
- [11] Pata J et al. Simulated particle-level events of $t\bar{t}$ and QCD with PU200 using PYTHIA8+DELPHES3 for machine learned particle flow (MLPF), (2021), https://zenodo.org/record/4559324
- [12] Falkner S, Klein A and Hutter F BOHB: robust and efficient hyperparameter optimization at scale (2018), https://arxiv.org/abs/1807.01774
- [13] Bowen Baker, Otkrist Gupta, Ramesh Raskar, Nikhil Naik, Accelerating Neural Architecture Search using Performance Prediction (2017) https://arxiv.org/abs/1705.10823
- [14] E. Pasetto, M. Riedel, F. Melgani, K. Michielsen and G. Cavallaro, Quantum SVR for Chlorophyll Concentration Estimation in Water With Remote Sensing, (2022) *IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, Art no. 1505705*, doi: 10.1109/LGRS.2022.3200325

drive. enable. innovate.

RAISE Center of Excellence

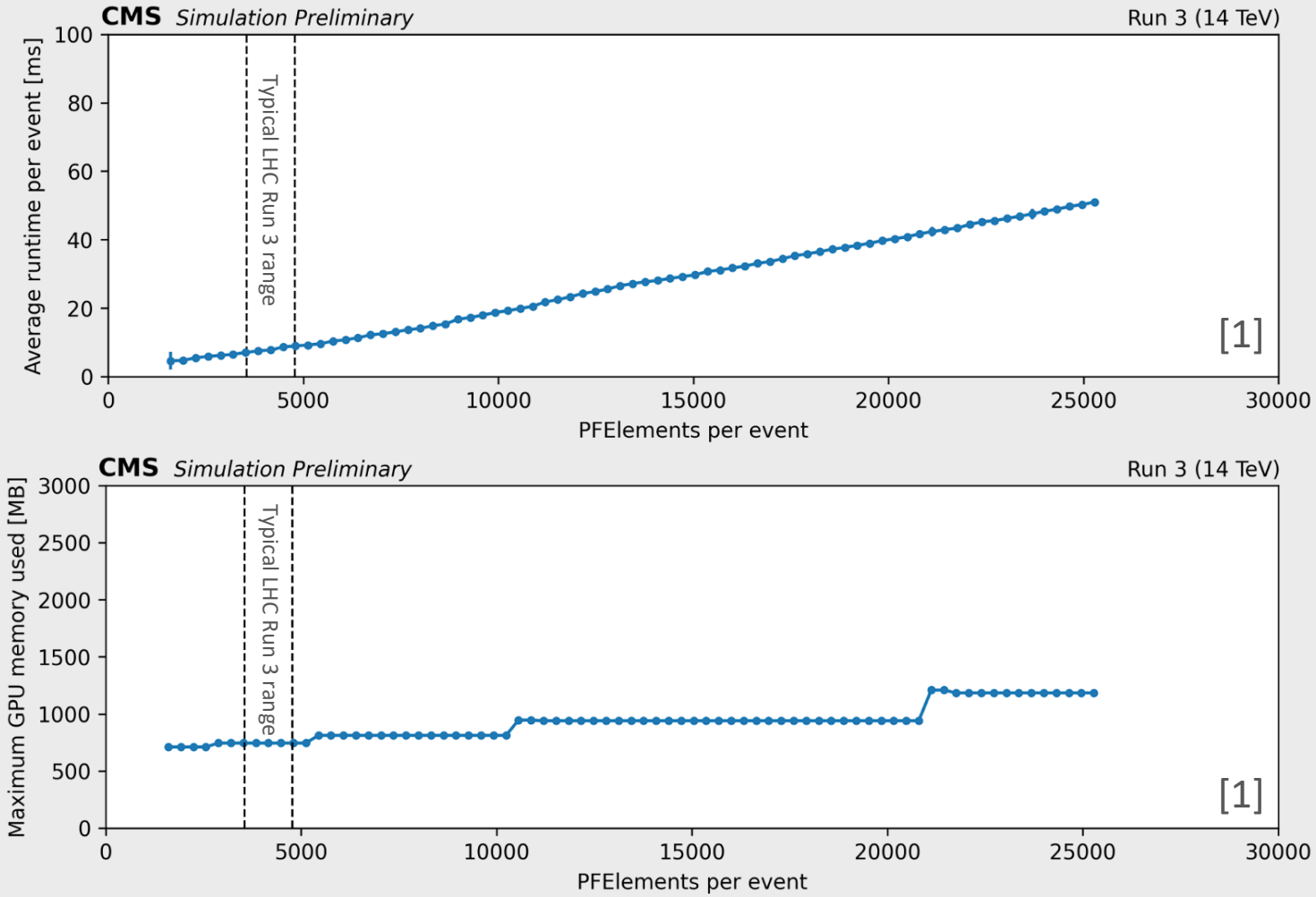
The CoE RAISE project has received funding from
the European Union's Horizon 2020 –
Research and Innovation Framework Programme
H2020-INFRAEDI-2019-1 under grant agreement no. 951733

Follow us:

Backup

Computational performance

- Key feature: avoid quadratic bottleneck, linear scaling of runtime and memory usage with input size
- Test was done using a single stream on 1 GPU with one event at a time (this is not a production setting)
- GPU: NVIDIA RTX 2060S



[1] Pata, J., Duarte, J., Mokhtar, F., Wulff, E., Yoo, J., Vlimant, J.-R., Pierini, M., Girone, M. (2022). *Machine Learning for Particle Flow Reconstruction at CMS*. Retrieved from http://arxiv.org/abs/2203.00330

TPE Algorithm [1]

1. Define a hyperparameter search space
2. Create an objective function, $f(\theta)$
3. Get some of observations using randomly selected hyperparameters
4. Divide the scores into two groups based on some quantile. The first group (x1) contains observations that gave the best scores and the second one (x2) - all other observations,
5. Two densities $l(x)$ and $\mathrm{g}(x)$ are modeled using Parzen Estimators (also known as KDEs) which are a simple average of kernels centered on existing data points,
6. Draw sample hyperparameters from $l(x)$, evaluating them in terms of $\mathrm{g}(x)/\mathrm{l}(x)$, and returning the set that yields the minimum value under $\mathrm{g}(x)/\mathrm{l}(x)$ corresponding to the greatest expected improvement. These hyperparameters are then evaluated on the objective function.
7. Update the observations from step 3
8. Repeat steps 4-7 with a fixed number of trials or until time limit is reached

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio and Balázs Kégl, Algorithms for Hyper-Parameter Optimization (2011), *Advances in Neural Information Processing Systems,* https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

PF-based ground truth datasets for MLPF

- Simulated samples
  - Inputs: PFElements
  - Targets PFCandidates (reconstructed by current baseline PF-algorithm)
  - Using a generator level truth would be better but is left for a future study
- Mixture of physical samples with pile-up (PU) and gun samples
- Generated under Run 3 conditions
  - tag: auto:phase1_2021_realistic
  - with CMSSW_12_1_0_pre3
- Data is split in train/test sets after shuffling using an 80:20 split

sample fragment	PU configuration	MC events
top-antitop pairs	flat 55-75	20k
$Z \to \tau\tau$ all-hadronic	flat 55-75	20k
single electron flat $p_T \in [1, 100]$ GeV	no PU	400k
single muon flat $p_T \in [0.7, 10]$ GeV	no PU	400k
single π^0 flat $p_T \in [0, 10]$ GeV	no PU	400k
single π flat $p_T \in [0.7, 10]$ GeV	no PU	400k
single τ flat $p_T \in [2, 150]$ GeV	no PU	400k
single γ flat $p_T \in [10, 100]$ GeV	no PU	400k

Table 1: MC simulation samples used for optimizing the MLPF model.

Generator-level-based ground truth datasets for MLPF

- Simulated samples
  - Inputs: PFElements
  - Targets: Gen-level ground truth
- Mixture of physical samples with pile-up (PU) and gun samples
- Generated under Run 3 conditions

Datasets

physics process	PU configuration	MC events
top quark-antiquark pairs	flat 55–75	100 k
QCD $\hat{p_T} \in [15, 3000]$ GeV	flat 55-75	100 k
QCD $\hat{p_T} \in [3000, 7000]$ GeV	flat 55–75	100 k
$Z \rightarrow \tau\tau$ all-hadronic	flat 55–75	100 k
single e flat $p_T \in [1, 1000]$ GeV	no PU	10 k
single μ log-flat $p_T \in [0.1, 2000]$ GeV	no PU	10k
single π^0 flat $p_T \in [0, 1000]$ GeV	no PU	10 k
single $\pi^\pm$ flat $p_T \in [0.7, 1000]$ GeV	no PU	10 k
single τ flat $p_T \in [1, 1000]$ GeV	no PU	10 k
single γ flat $p_T \in [1, 1000]$ GeV	no PU	10 k
single p flat $p_T \in [0.7, 1000]$ GeV	no PU	10 k
single n flat $p_T \in [0.7, 1000]$ GeV	no PU	10 k

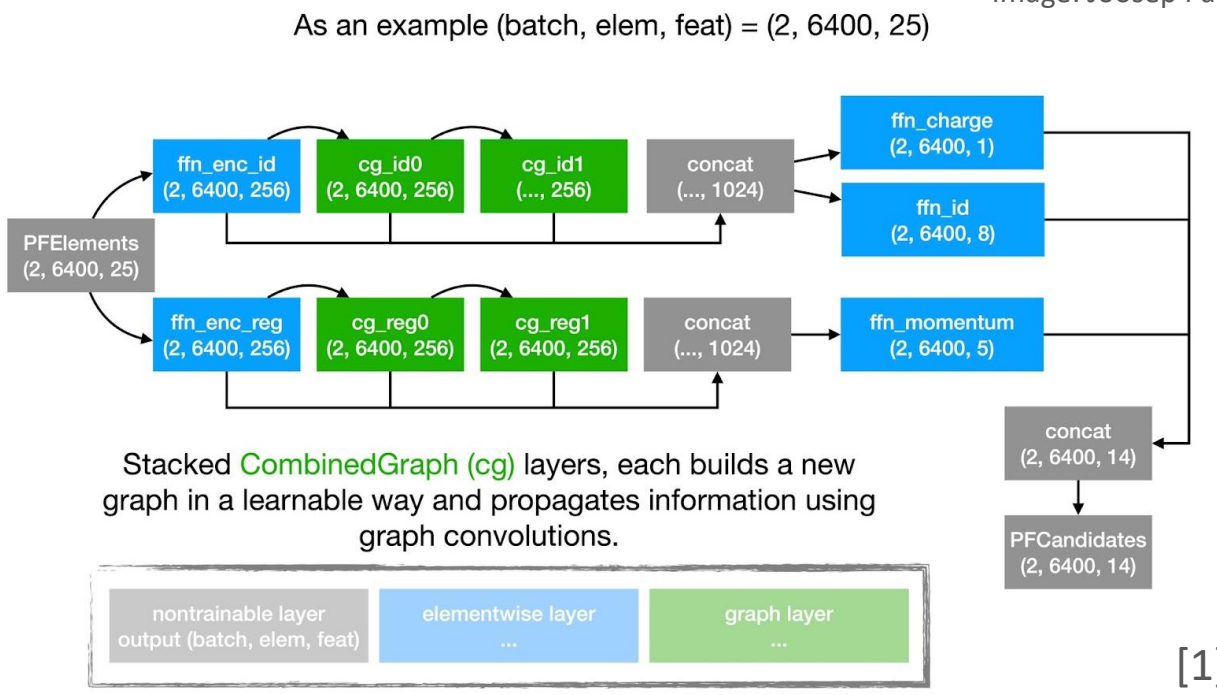
Table 1: MC simulation samples used for optimizing the MLPF model.

The training datasets for MLPF are generated using CMSSW under Run3 conditions, saving the PF elements as inputs, the MLPF truth as the target, and reconstructed PF candidates as an additional cross-check that is not used for ML optimization. A mix of samples with physical pileup conditions, as well as single particle gun samples are used for training and validation.

MLPF architecture overview

- Fully connected feedforward networks are used for encoding and decoding
- Custom CombinedGraph layers are used to dynamically build graphs and perform message passing
- We use two stacks of CombinedGraph layers:
  - one for momentum regression
  - one for particle classification

Image: Joosep Pata



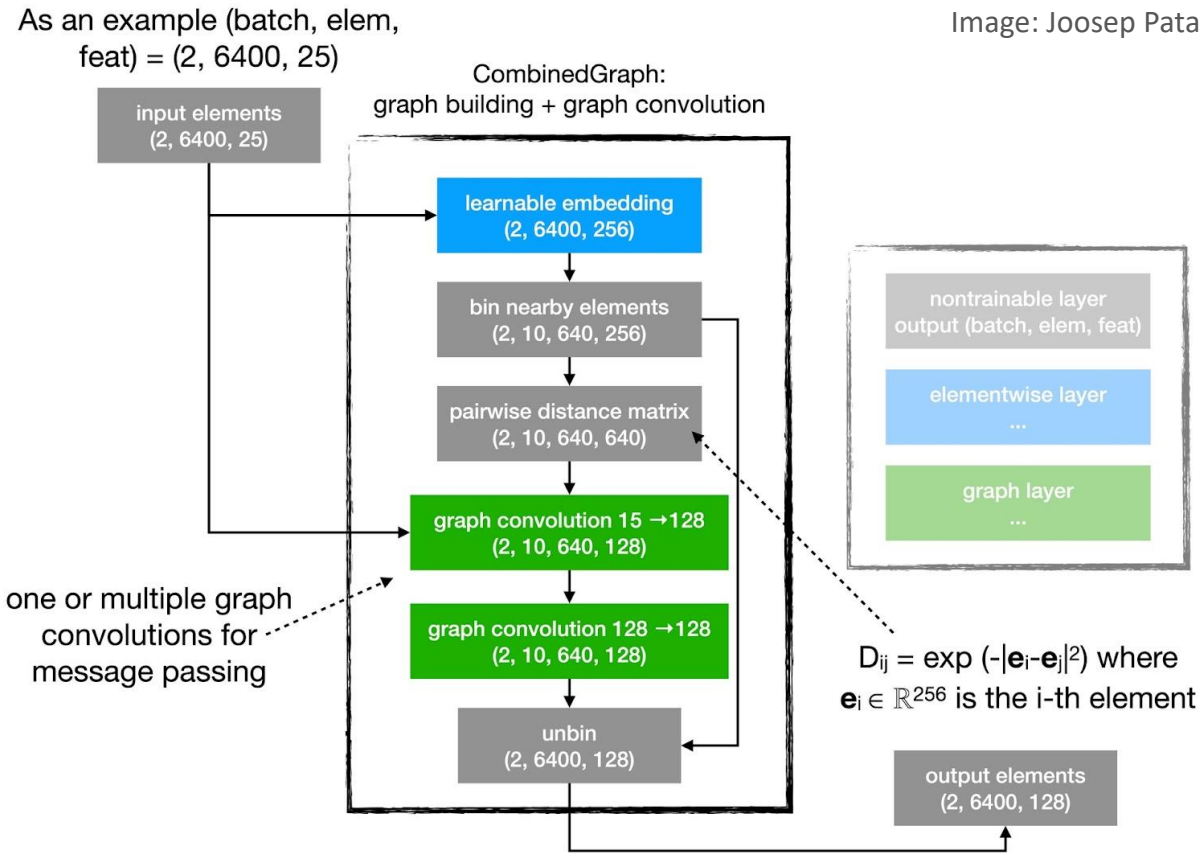
[1]

[1] Pata, J., Duarte, J., Mokhtar, F., Wulff, E., Yoo, J., Vlimant, J.-R., Pierini, M., Girone, M. (2022). *Machine Learning for Particle Flow Reconstruction at CMS*. Retrieved from http://arxiv.org/abs/2203.00330

The CombinedGraph layer

- A learnable embedding allows flexibility in graph building
- Elements are binned into local context areas
- A graph is built within each bin
- The graphs are disjoint but together they represent the entire event
- One or more graph convolutions are applied for message passing
- Elements are unbinned and returned

Image: Joosep Pata



[1]

[1] Pata, J., Duarte, J., Mokhtar, F., Wulff, E., Yoo, J., Vlimant, J.-R., Pierini, M., Girone, M. (2022). *Machine Learning for Particle Flow Reconstruction at CMS*. Retrieved from http://arxiv.org/abs/2203.00330

BOHB (Bayesian Optimization Hyperband)

- Combination of Bayesian Optimization with early stopping functionality of Hyperband [12]
- Using 12 compute nodes with:
  - 4x NVIDIA A100 SXM4 40GB
  - 2× AMD EPYC Rome 7402, 2× 24 cores @ 2.8 GHz
- Two different hypertuning runs:
  - With ExponentialDecay
    - Approximately 105.5 node-hours, or 10128 core-hours
  - With CosineDecay
    - Approximately 98.4 node-hours, or 9446 core-hours

Run on JUWELS Booster

Image: JÜLICH Forschungszentrum

```
num_samples = 200
search_space = {
    "lr": loguniform(1e-4, 3e-2),
    "expdecay_decay_steps": quniform(10, 2000, 10),
    "dropout": uniform(0.0, 0.5),
    "clip_value_low": uniform(0.0, 0.2),
    "dist_mult": uniform(0.01, 0.2),
}
```

HPO studies done in 2021 in: E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

[12] Falkner S, Klein A and Hutter F BOHB: robust and efficient hyperparameter optimization at scale (2018), https://arxiv.org/abs/1807.01774

ASHA + Bayesian Optimization

- Thanks to SDSC and Flatiron Institute
- Using 12 compute nodes with 4 GPUs per node
  - 4x NVIDIA A100 SXM4 40GB
  - 64 core Intel Icelake Platinum 8358 CPU @ 2.60GHz
- Approximately 886.4 node-hours or equivalently 56730 CPU core-hours
- Back of the envelope calculation: it would have taken ~6 months on a single GPU instead of ~83h using HPC systems (for both stages)
- Access to HPC resources are essential for Hypertuning complex AI models on large datasets

Run on Flatiron Institute's Iron Cluster

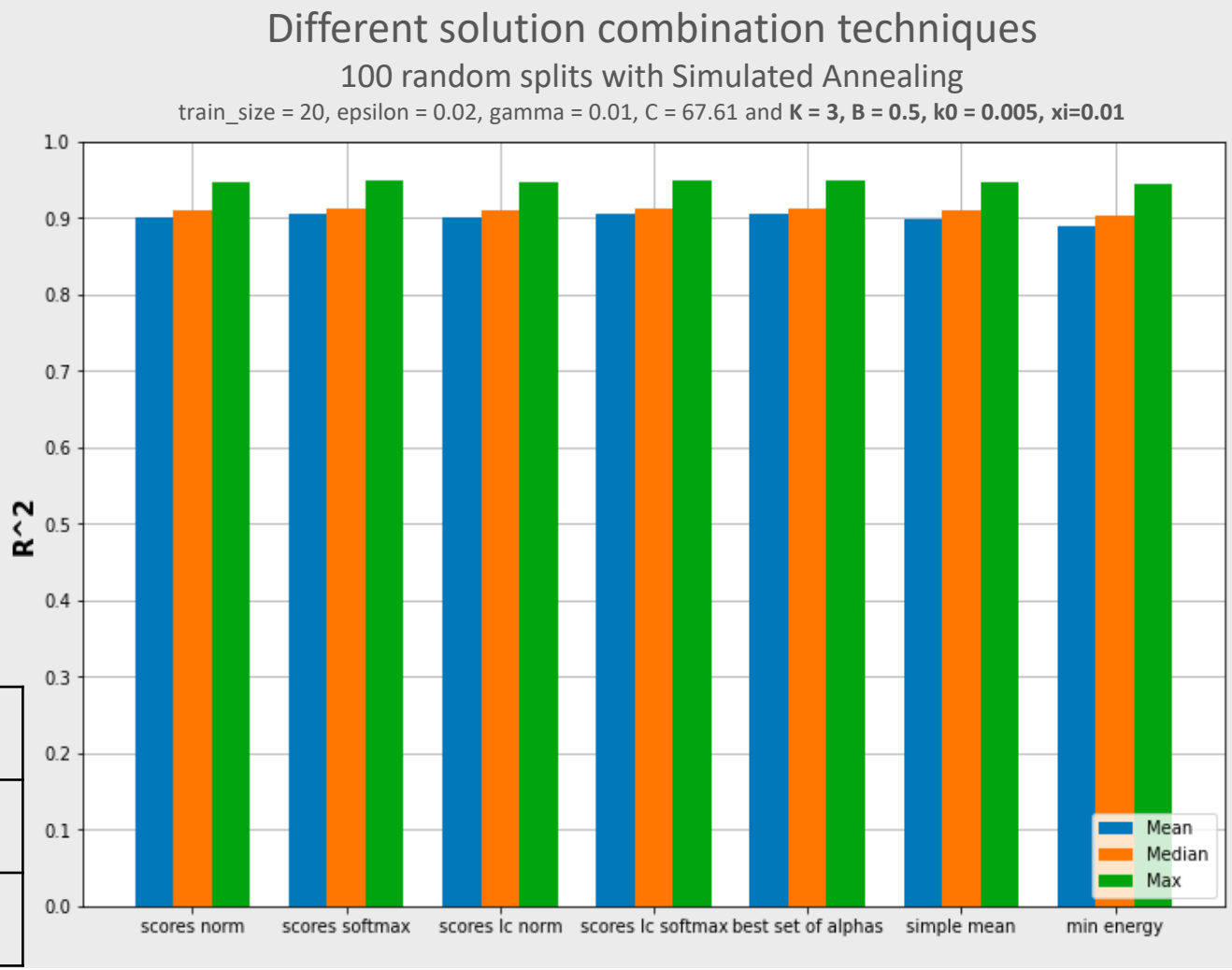
```
num_samples = 200
search_space = {
        "layernorm": choice([False, True]),
        "ffn_dist_hidden_dim": choice([32, 64, 128, 256]),
        "ffn_dist_num_layers": choice([1, 2, 3, 4]),
        "distance_dim": choice([32, 64, 128, 256]),
        "num_node_messages": choice([1, 2, 3, 4]),
        "num_graph_layers_common": choice([1, 2, 3, 4]),
        "num_graph_layers_energy": choice([1, 2, 3, 4]),
        # "dropout": samp([0.0]),
        "bin_size": choice([16, 32, 40, 64, 80]),
        # "clip_value_low": samp([0.0]),
        "normalize_degrees": choice([True, False]),
        "output_dim": choice([32, 64, 128, 256]),
}
```


HPO studies done in 2021 in: E.Wulff, M. Girone, J. Pata, Hyperparameter optimization of data-driven AI models on HPC systems, *J. Phys.: Conf. Ser.* 2438 012092 (2023) https://doi.org/10.1088/1742-6596/2438/1/012092

D-Wave simulated annealing

- D-Wave's Ocean software provides
  - `neal.SimulatedAnnealingSampler()`
  - Allows to test our algorithm in a controlled way without using quantum computing time
- The annealer returns many solutions, some of which are combined to produce final QSVR model
  - Techniques described in [14]
- Results are good – comparable to classical SVR

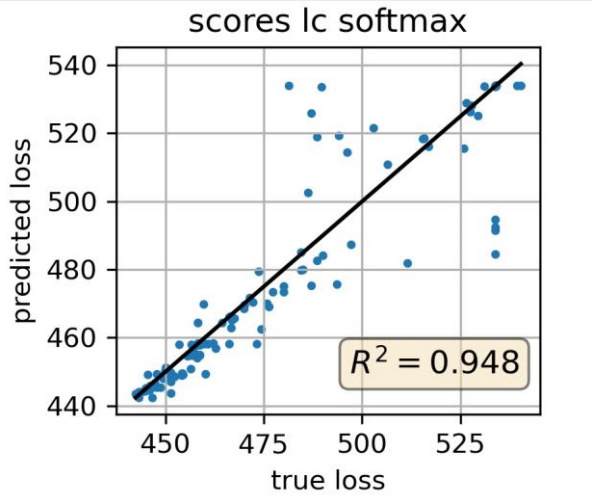
	Best	Worst	Mean	Std	Median	Number of trainings
Classical SVR R^2	0.959	0.318	0.889	0.050	0.900	1000
Simulated annealing R^2	0.949	0.383	0.901	0.045	0.910	100



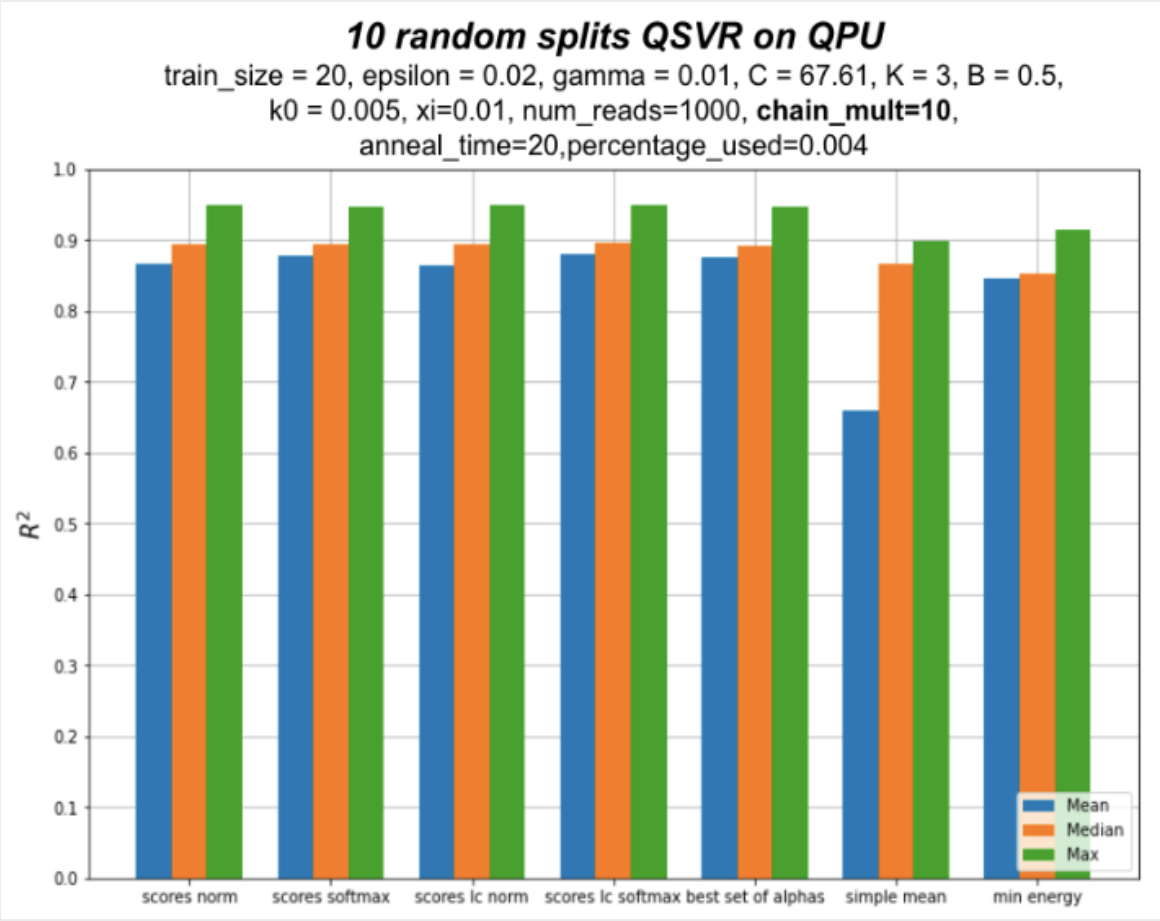
Quantum-SVR

- Predicting final loss from fraction of loss curve
- Comparable results to classical SVR and to simulated annealing
- More stable results

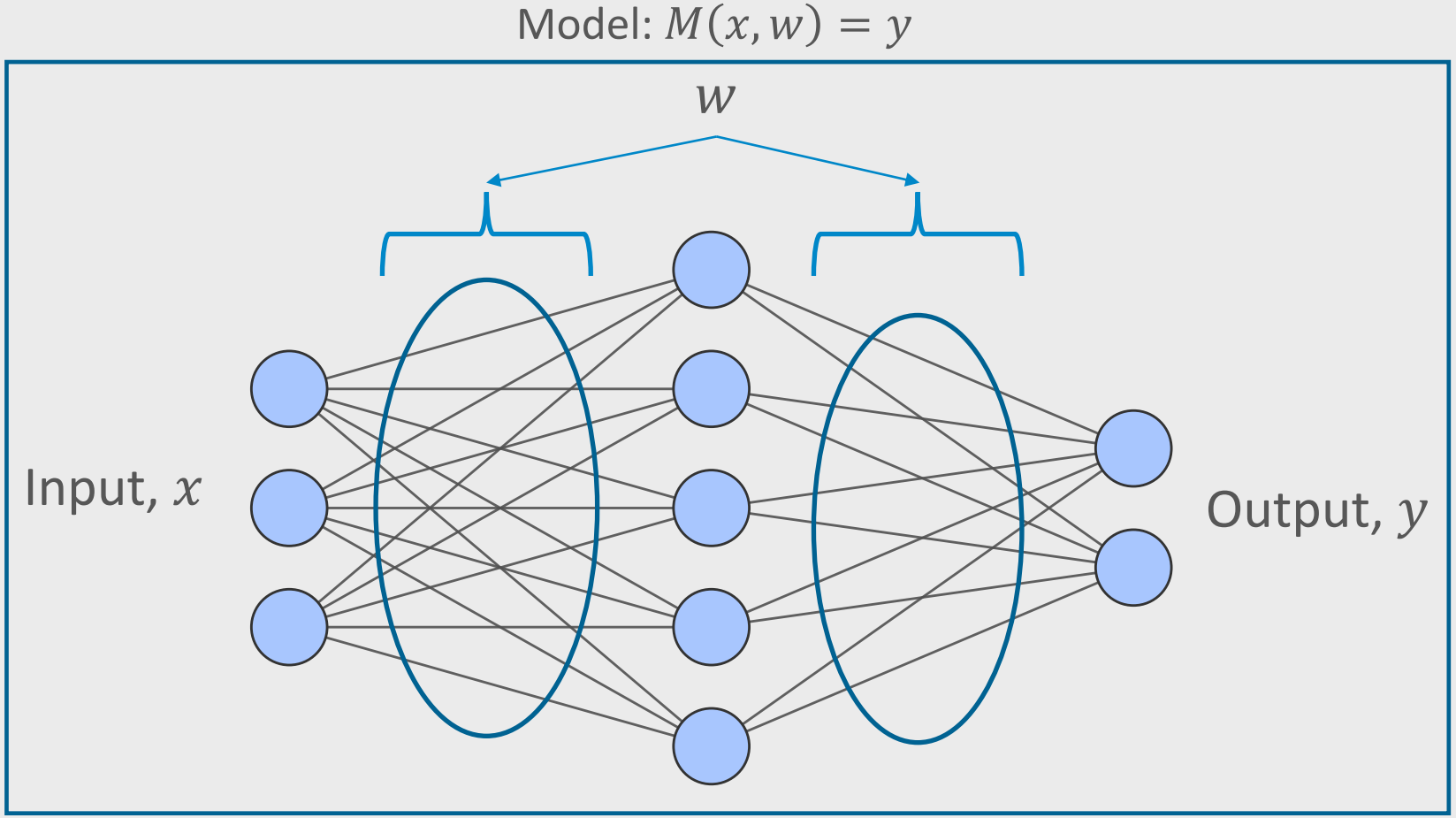
Best performing QSVR



	Best	Worst	Mean	Std	Median	Number of trainings
Classical SVR R^2	0.959	0.318	0.889	0.050	0.900	1000
Simulated annealing R^2	0.949	0.383	0.901	0.045	0.910	100
Quantum annealing R^2	0.948	0.742	0.880	0.056	0.897	10

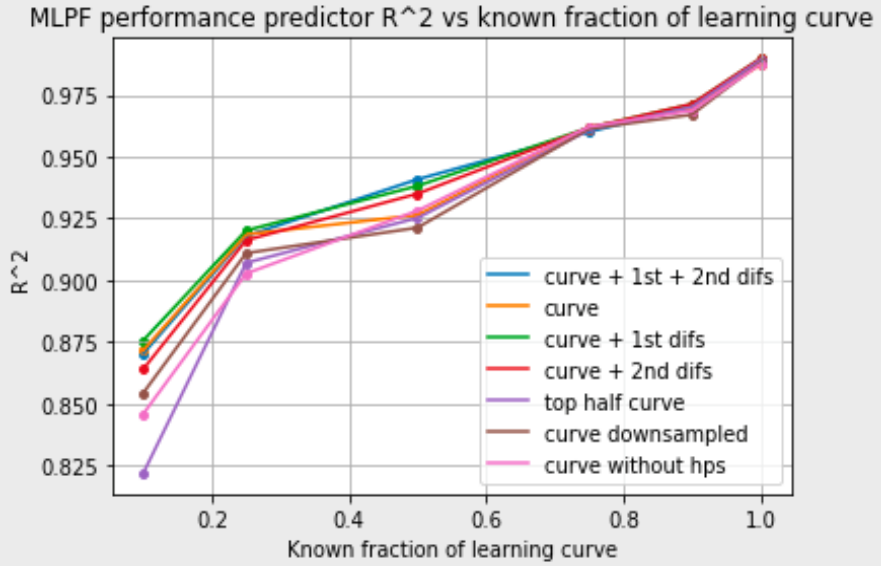


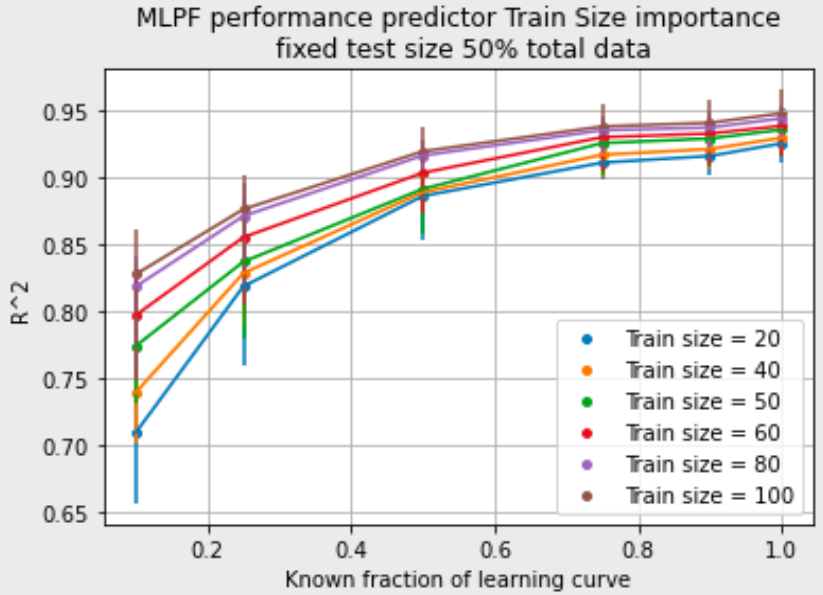
A two-layer fully connected feed-forward network



Classical SVR as performance predictor

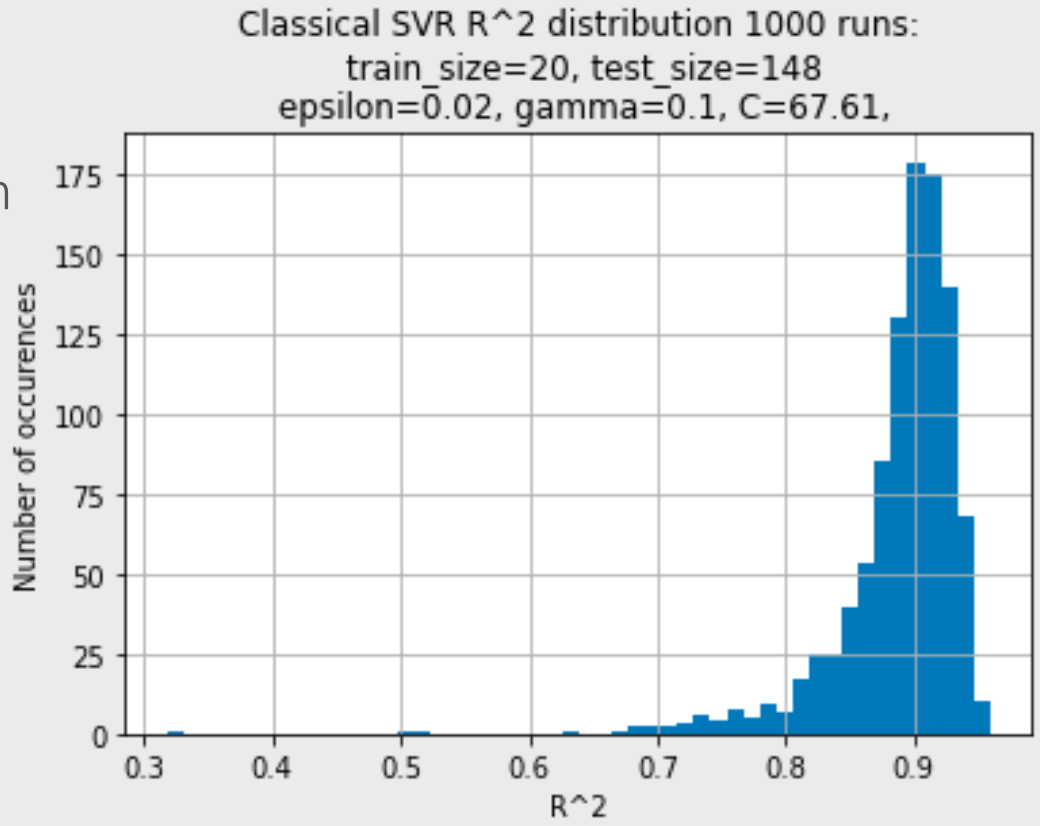
- We first studied the problem using classical SVR
- Varying the know fraction of learning curve and size and contents of feature vector
- With ≥25% of epochs observed, we see an $R^2 > 0.9$
- We also studied the performance dependence on training set size
- $R^2 = 1 - \frac{\sum_i (y_i - \widehat{y_i})^2}{\sum_i (y_i - \overline{y})^2} = 1 - \frac{\sum_i {e_i}^2}{\sum_i (y_i - \overline{y})^2}$





Classical SVR under restrictions

- We evaluate how well a classical SVR performs when meeting same restrictions as the QSVR
  - Reduced training set: randomly draw 20 samples from full training set
  - Reduced feature vector: using only downsampled learning curve as input
- Fit 1000 SVRs using same HPs



	Best	Worst	Mean	Std	Median
R^2	0.959	0.318	0.889	0.050	0.900