System Support for Automatic Machine Learning Tuning

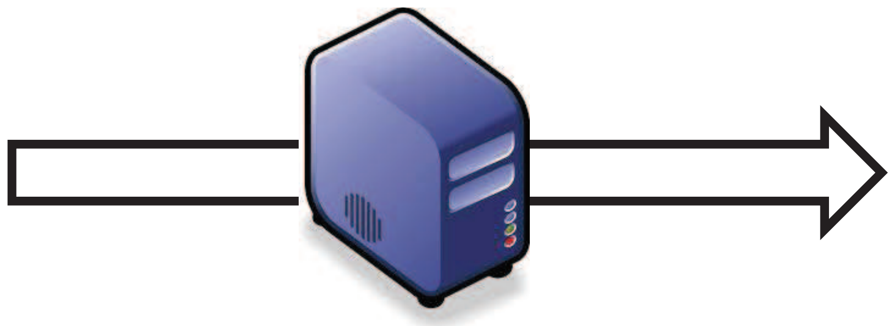
Henggang Cui,
Greg Ganger, and Phil Gibbons

PARALLEL DATA LABORATORY
Carnegie Mellon University

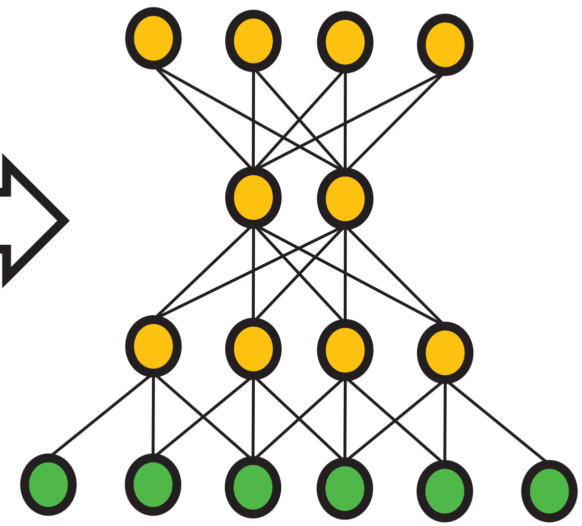
Machine learning

Training data: labelled images

A machine learning task

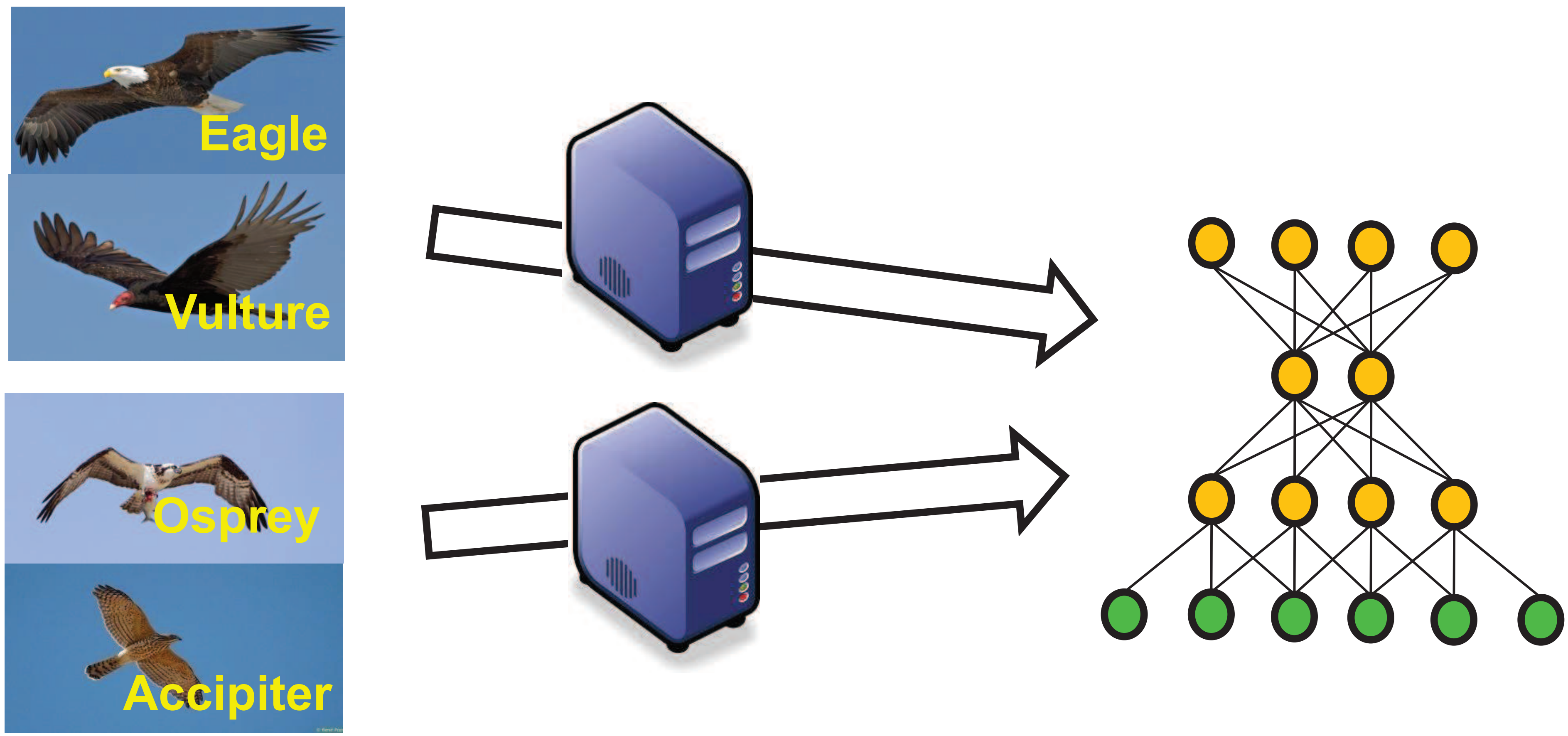


Fits model params to training data, by optimizing the objective function (e.g., via SGD)



Model parameters (solution): neural net weights

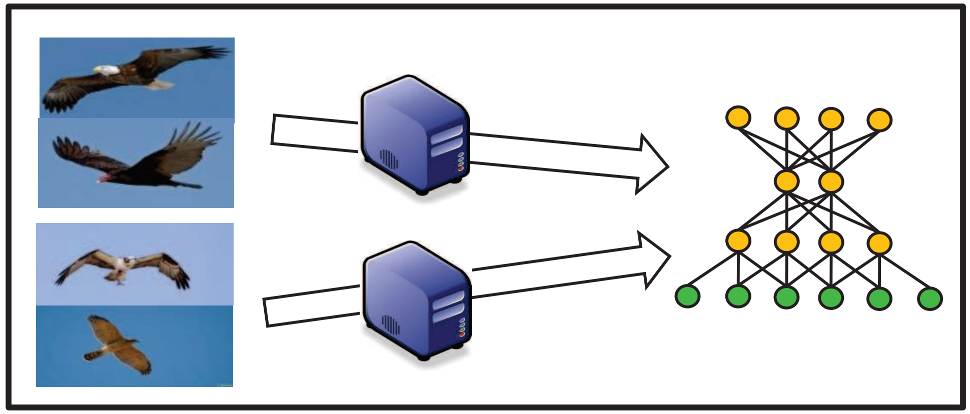
Distributed machine learning

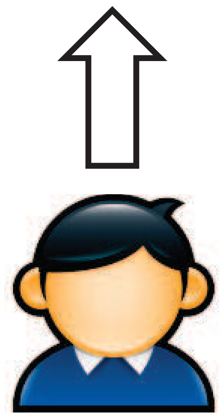


Partitioned
training data

Shared
model parameters

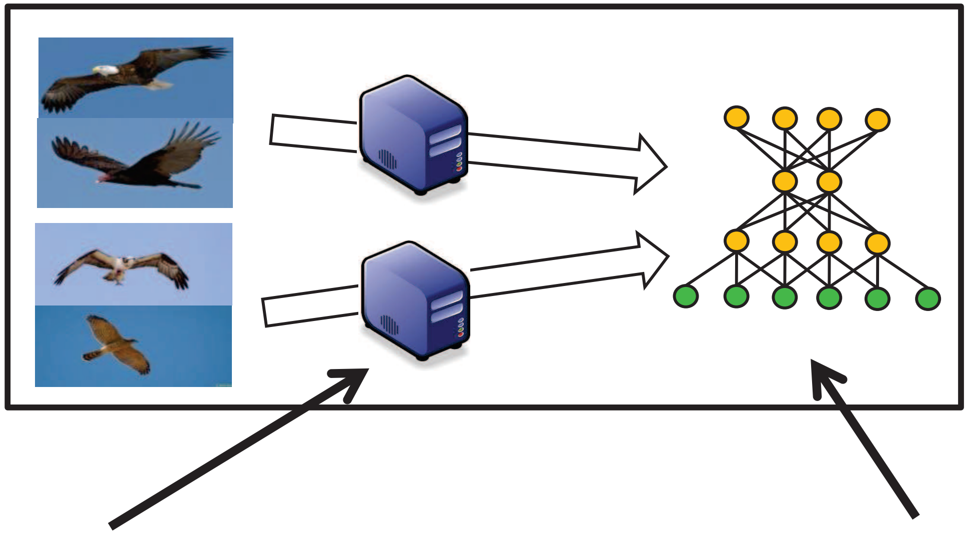
Training tunables in machine learning





- **Training tunables:**
  - learning rate (step size)
  - training batch size
  - data staleness bound
  - ...

Training tunables in machine learning



- **Training tunables:**
  - learning rate (step size)
  - training batch size
  - data staleness bound
  - ...
- **NOT in the obj. function**

- **Model hyperparams:**
  - network depth
  - neuron layer sizes
  - neuron activation function
  - ...
- **In the obj. function**

Training tunables are tricky

- Training tunables matter
  - affect ML task completion time
    - e.g., orders of magnitude slower with bad choices
  - affect solution quality
    - e.g., sub-optimal solution with bad choices
- Tuning is hard
  - best choice depends on many factors
    - e.g., app, model, dataset, hardware environment
  - best choice changes during training
    - e.g., large learning rate at the beginning, small at the end
- **Our goal: system for automatic tuning**

Outline

- Motivation
- Traditional tunable tuning approaches
- System design for better tunable tuning
- Experiment results

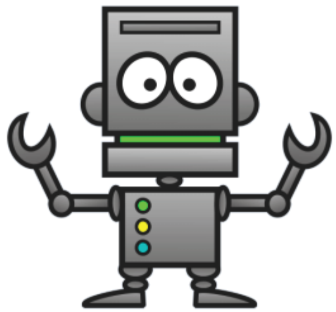
Traditional tunable tuning approaches

- Manual tuning
  - by domain expert, via trial and error
  - slow, expensive and prone to sub-optimal choices

- Hyperparam optimization (e.g., [Snoek et al., 2012])
  - train the model to completion to evaluate a choice
    - useful for model hyperparam tuning
    - not a good idea for the training tunables
  - not able to dynamically change tunables

Our goal: better tuning

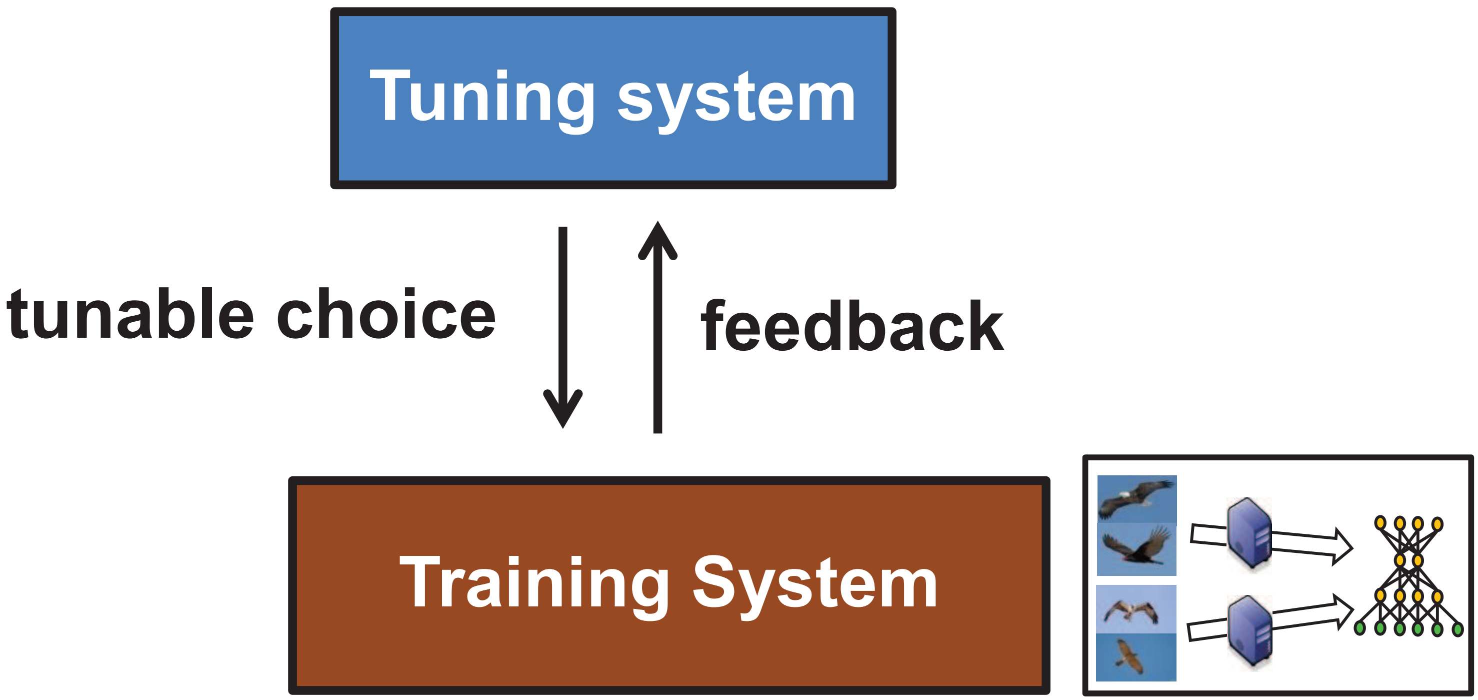
- Tunables should be tuned
  - automatically
    - without the help of domain experts
  - with low overhead
    - no need to train to completion to evaluate a choice
  - dynamically
    - adjust tunable choices during the training



Outline

- Motivation
- Traditional tuning approaches
- System design for automatic tuning
- Experiment results

Automatic tuning system



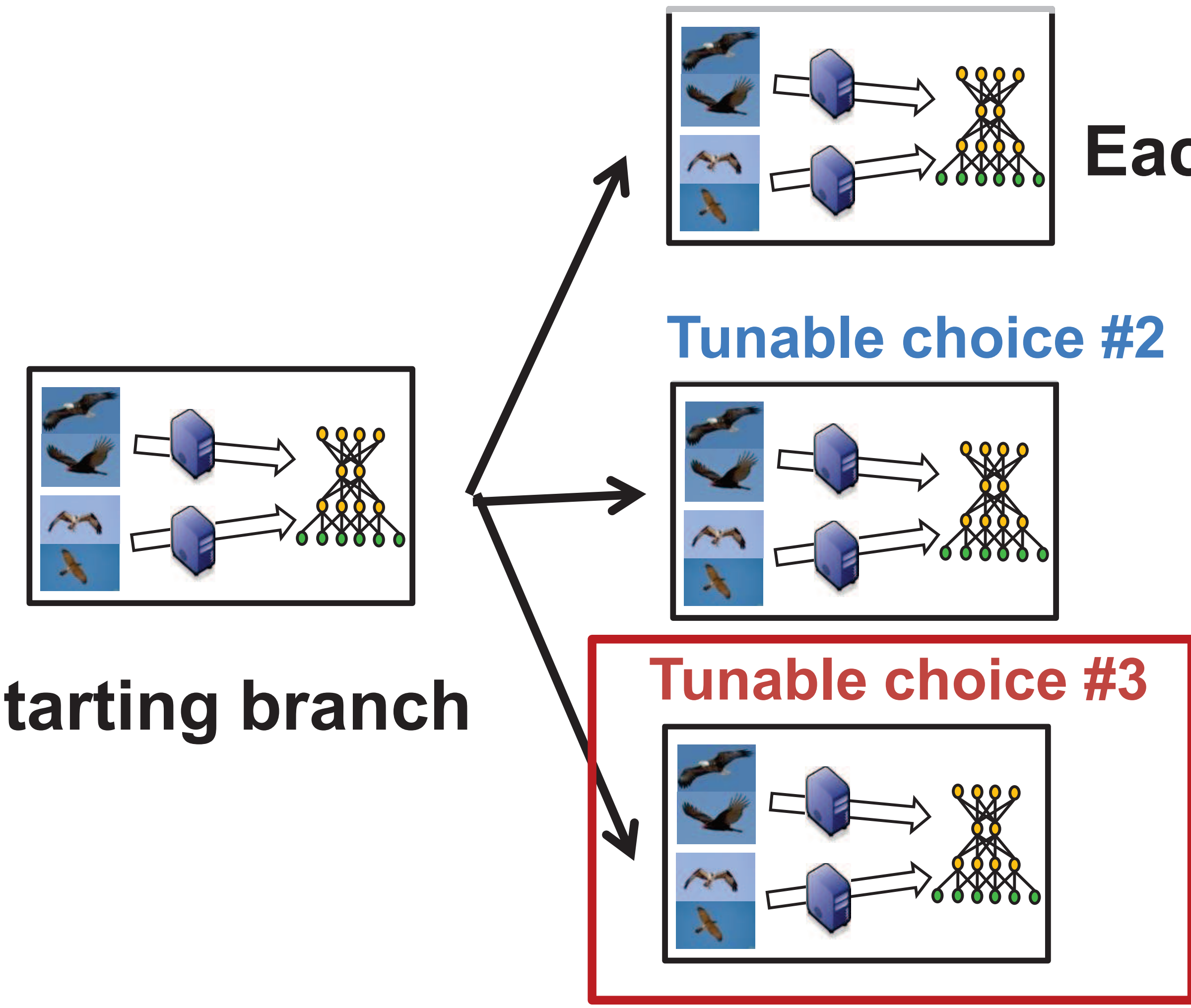
- Automatically tune tunables for training system

Try & evaluate tunables in trial branches

Tunable choice #1

Tunable choice #2

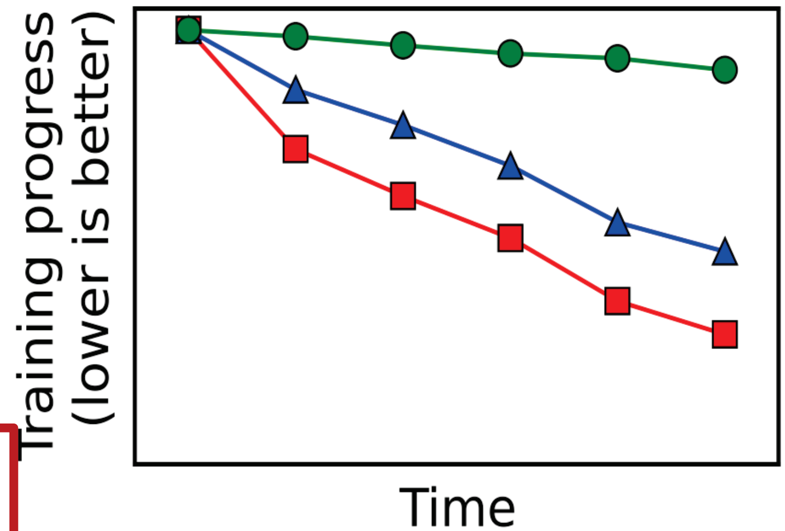
Tunable choice #3



Starting branch

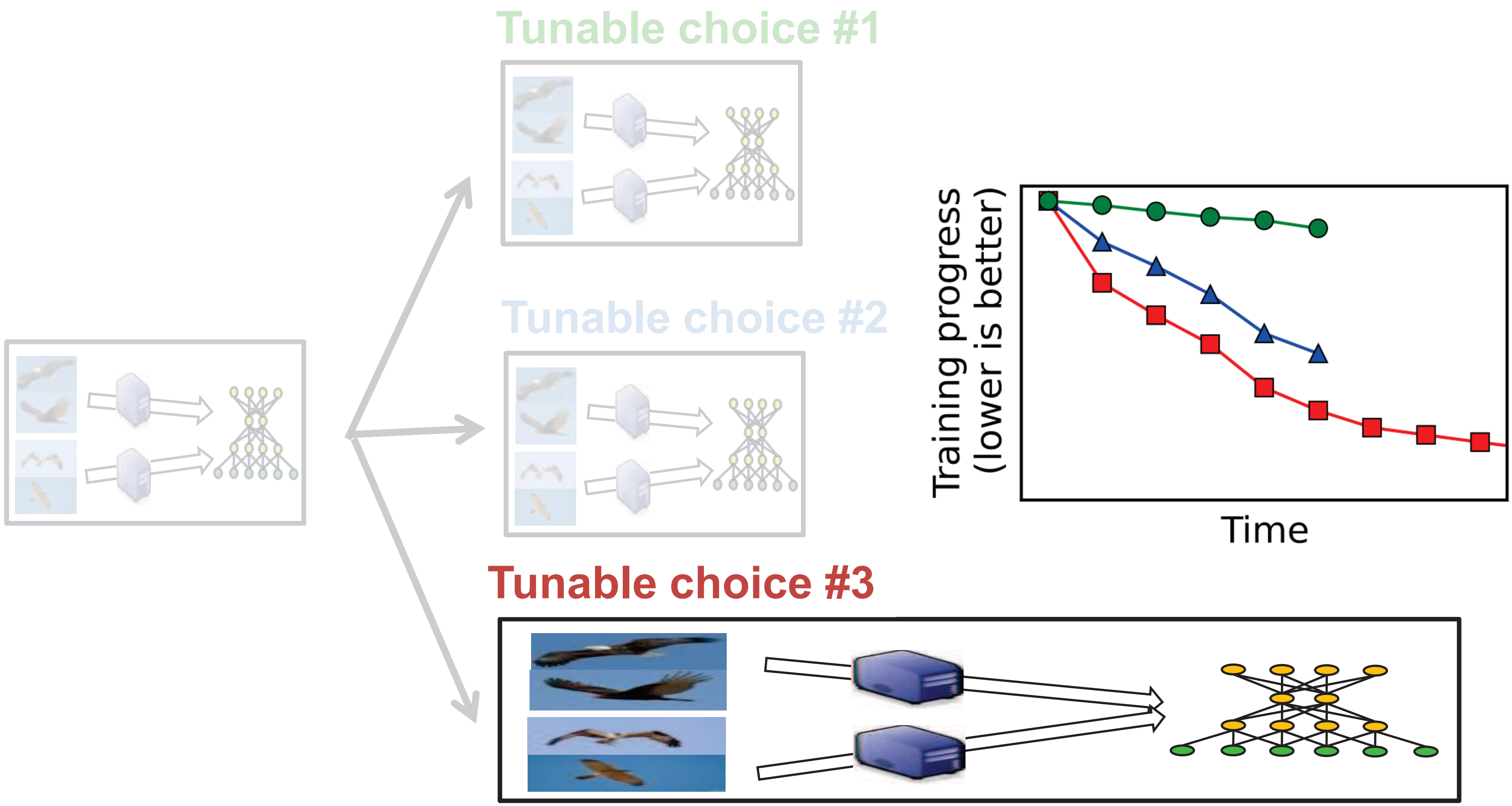
Forked trial training branches

Each running for some time



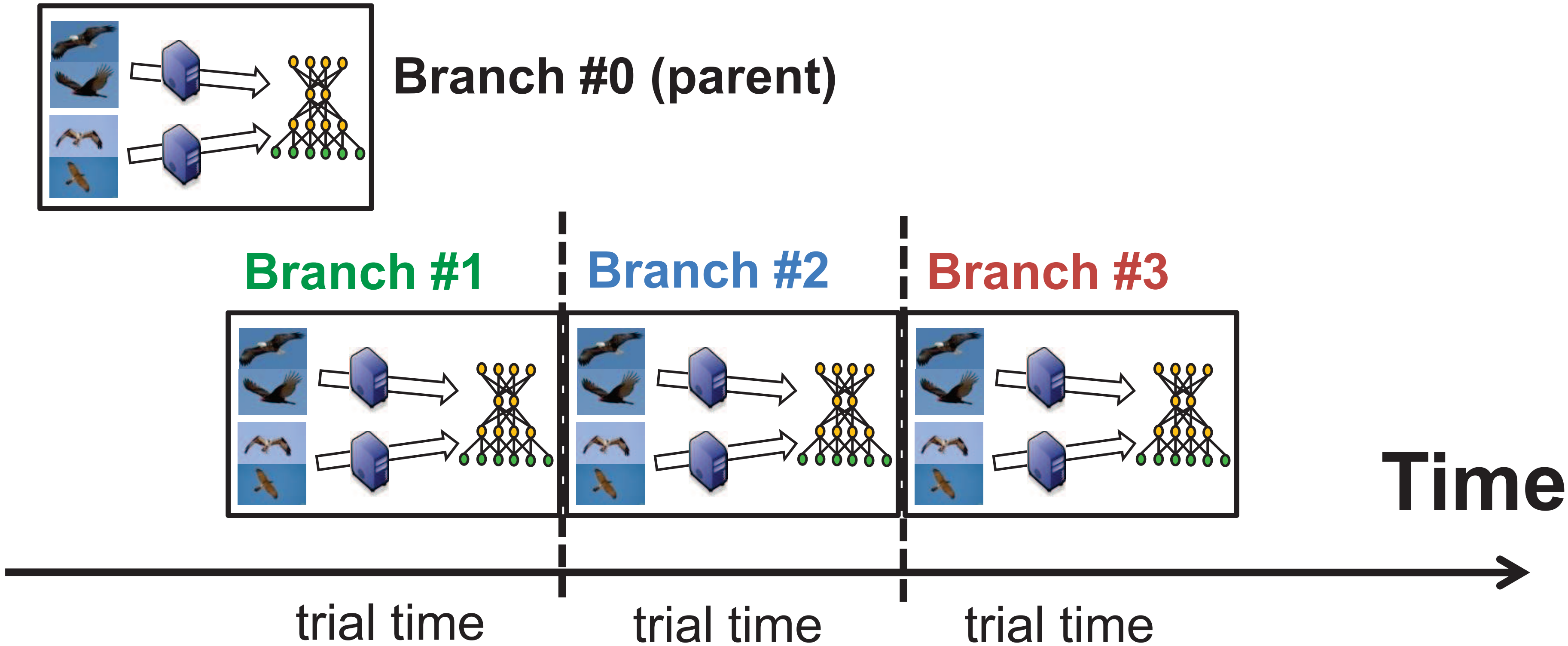
Monitor
convergence speed

Try & evaluate tunables in trial branches



Keep training the best branch

Time sharing for training branches

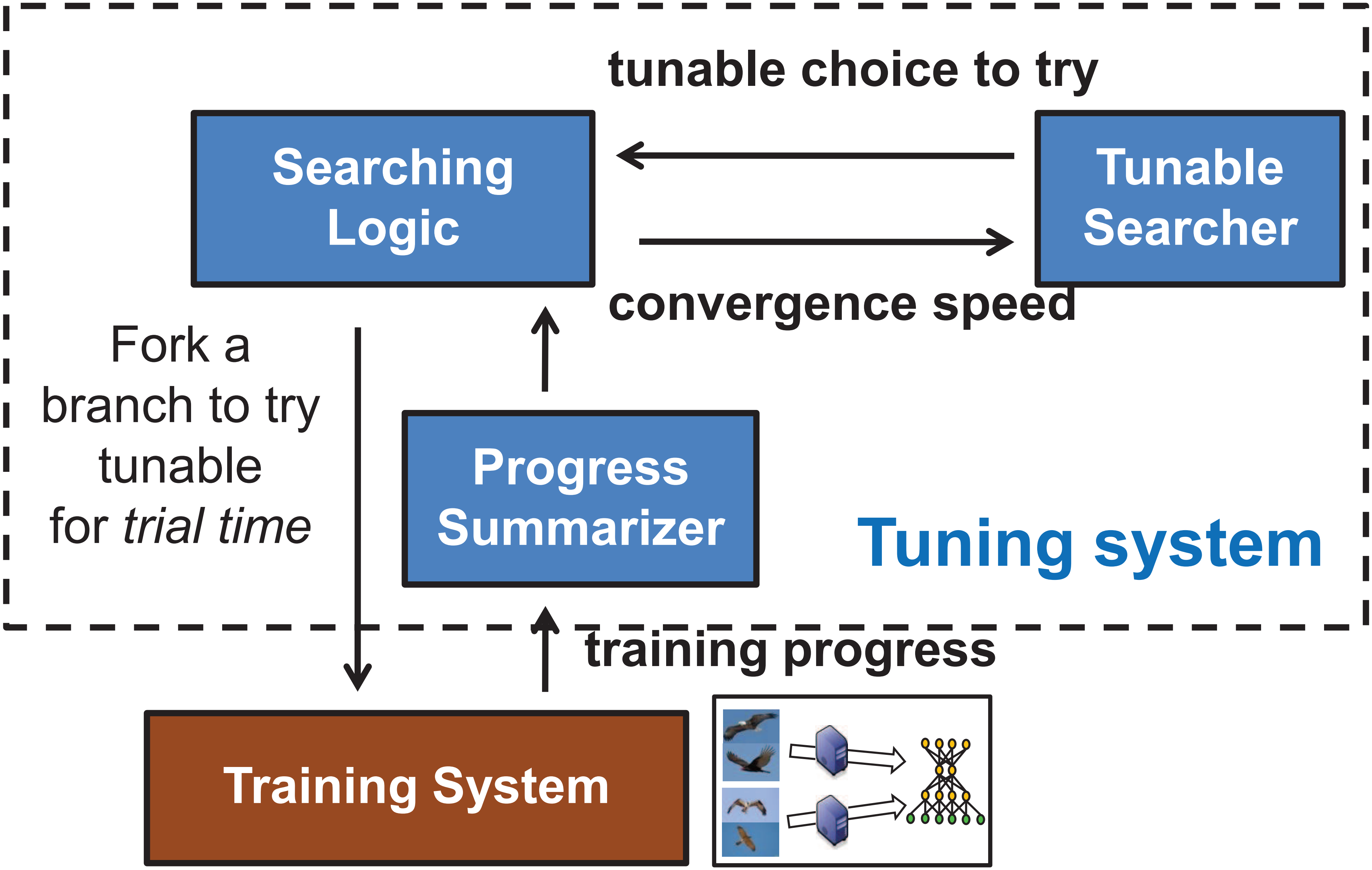


- Not running branches in parallel, because
  - we don’t want to use 3x more machines
    - most of time, trials aren’t going
  - we want to precisely measure their training perf.

Outline

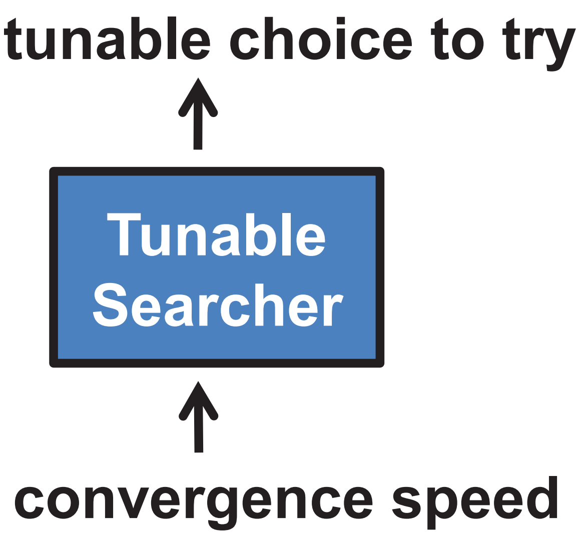
- Traditional tuning approaches
- System design for better tuning
  - Trying tunables in time sharing branches
  - Tuning design
- Experiment results

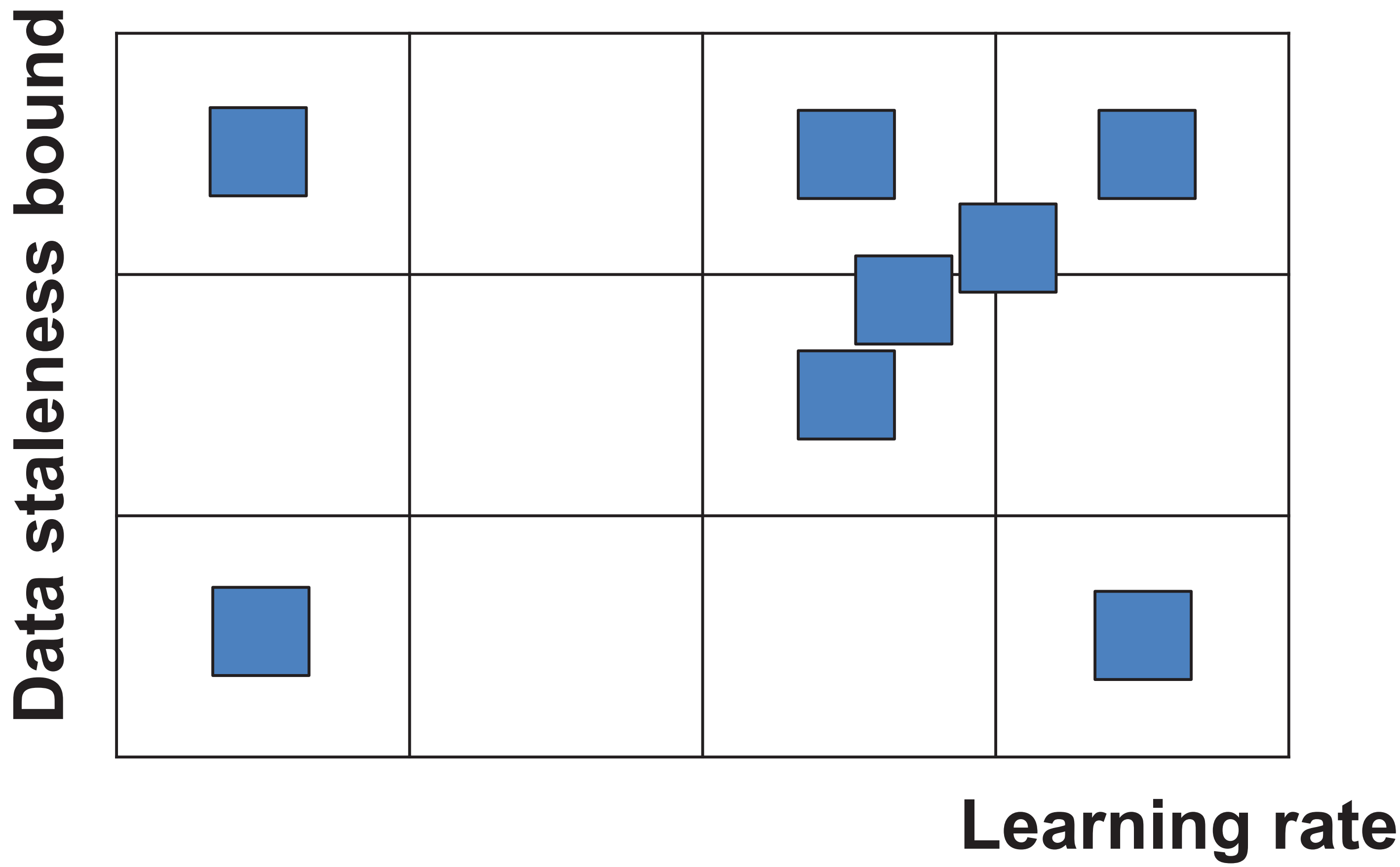
Tuning procedure



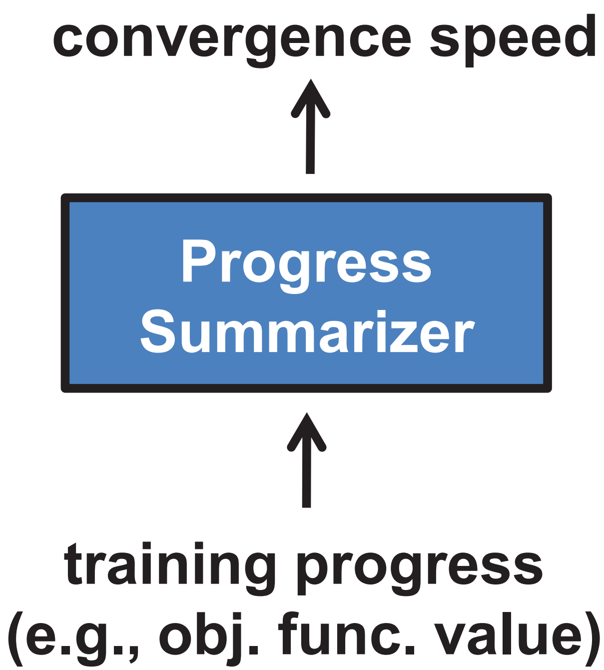
Tunable searcher

- BayesianOpt searcher
  - Decide choice w/ Bayesian optimization

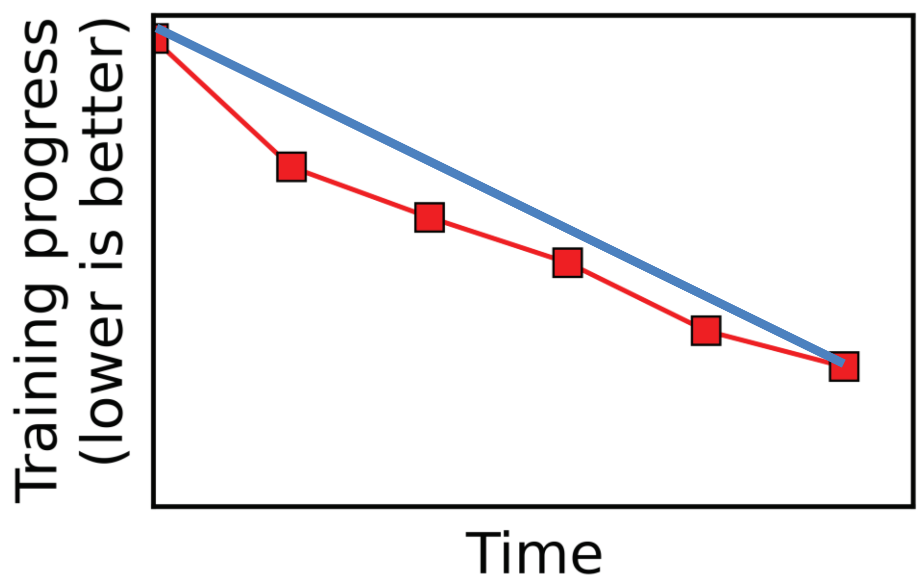




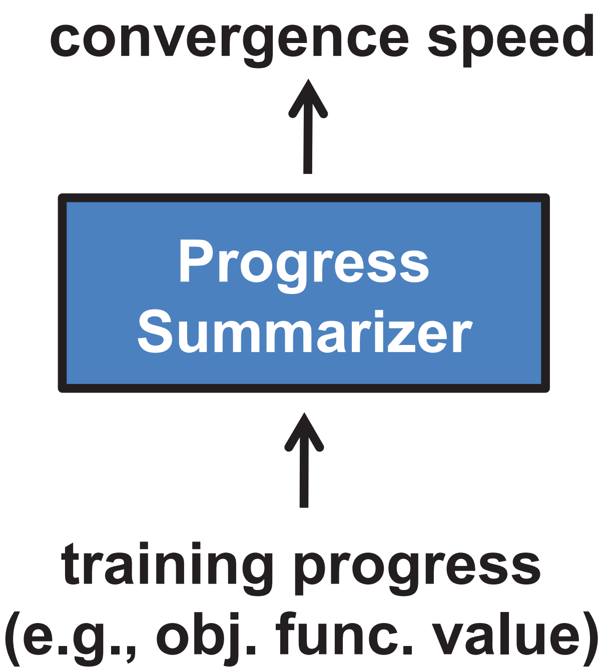
Summarizing progress

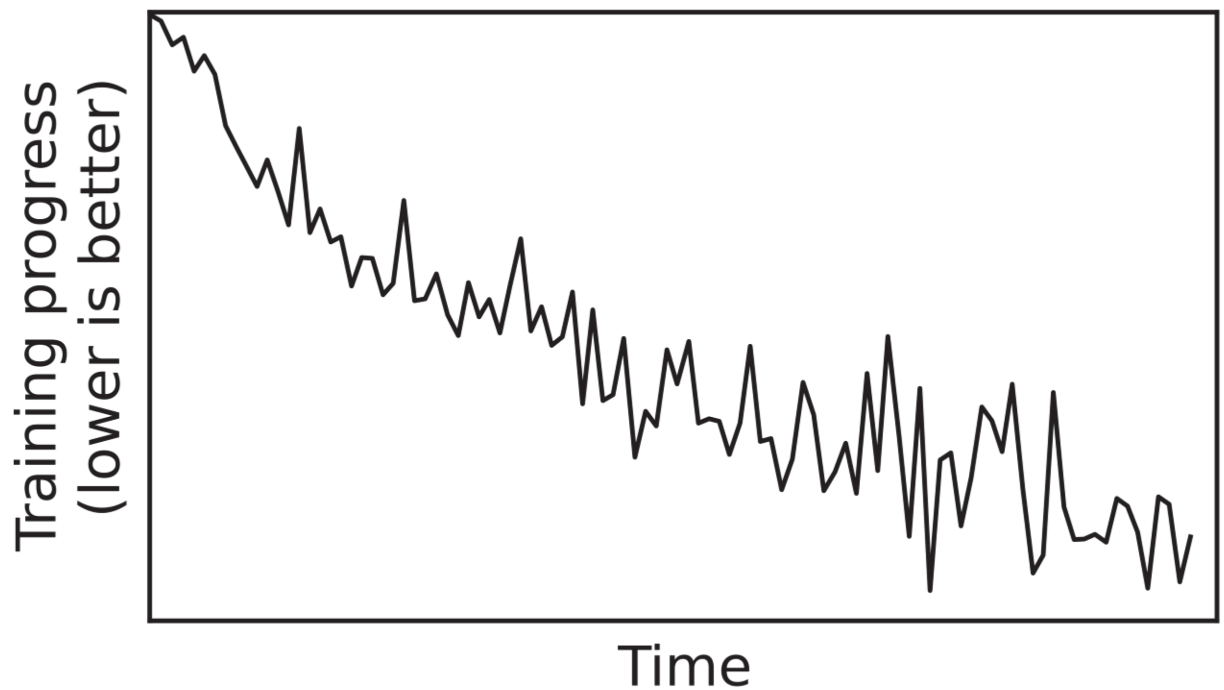


Slope of the training progress (progress per second)



Summarizing progress





Noisy progress

Summarizing progress

convergence speed

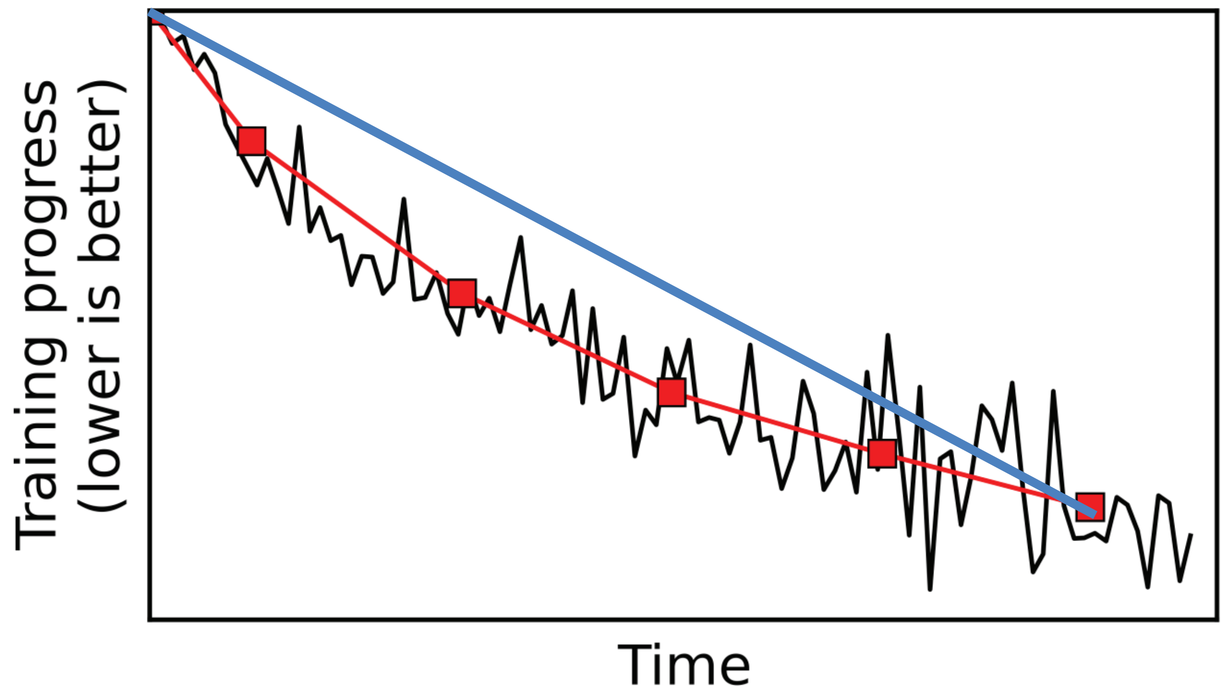
↑

Progress Summarizer

↑

training progress
(e.g., obj. func. value)

Slope of the downsampled progress



Noisy progress

Summarizing progress

- **Convergence and stability checks**

convergence speed

↑

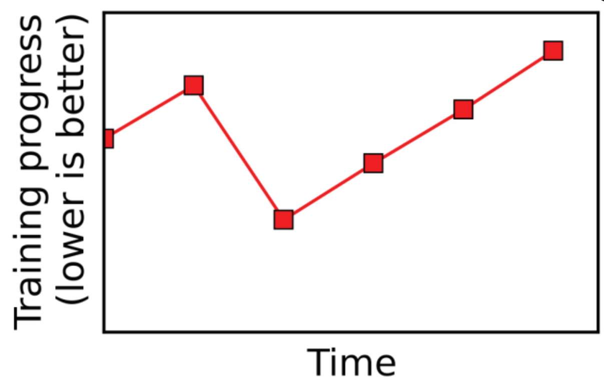
Progress Summarizer

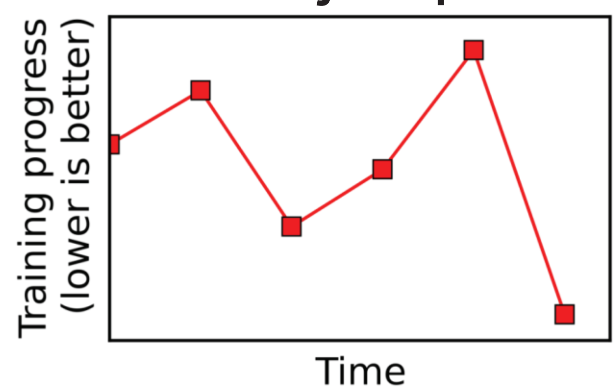
↑

training progress
(e.g., obj. func. value)

Slope cannot be positive

Progress cannot jump too much

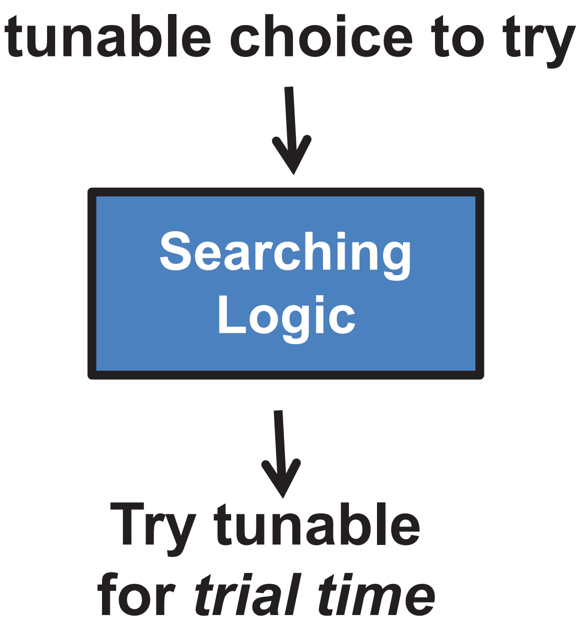


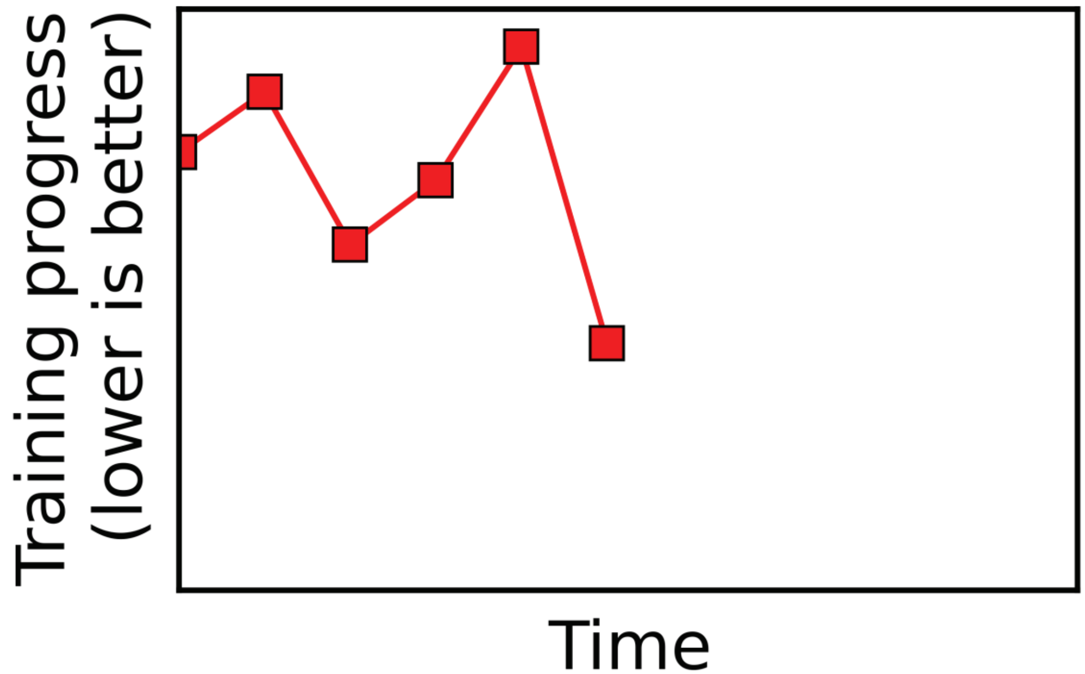


Not converging

Might need to run for longer

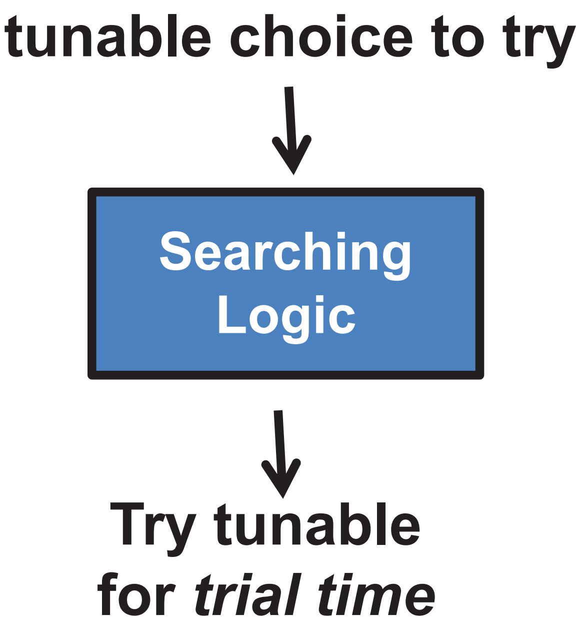
Deciding trial time

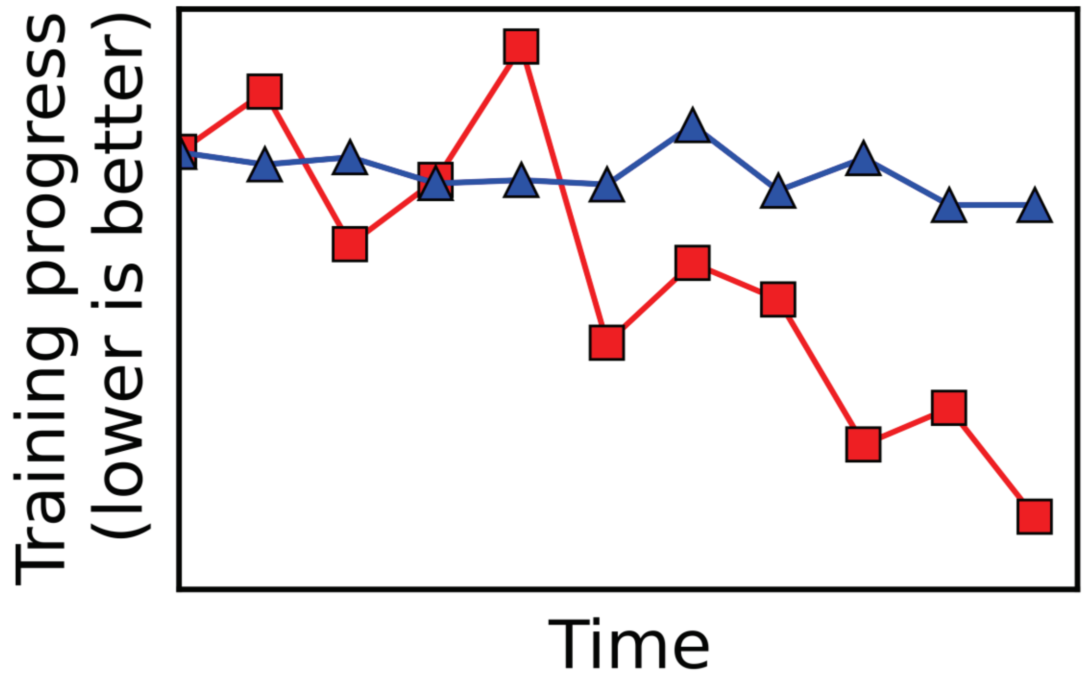




- Need a long enough trial time to get stable progress

Deciding trial time



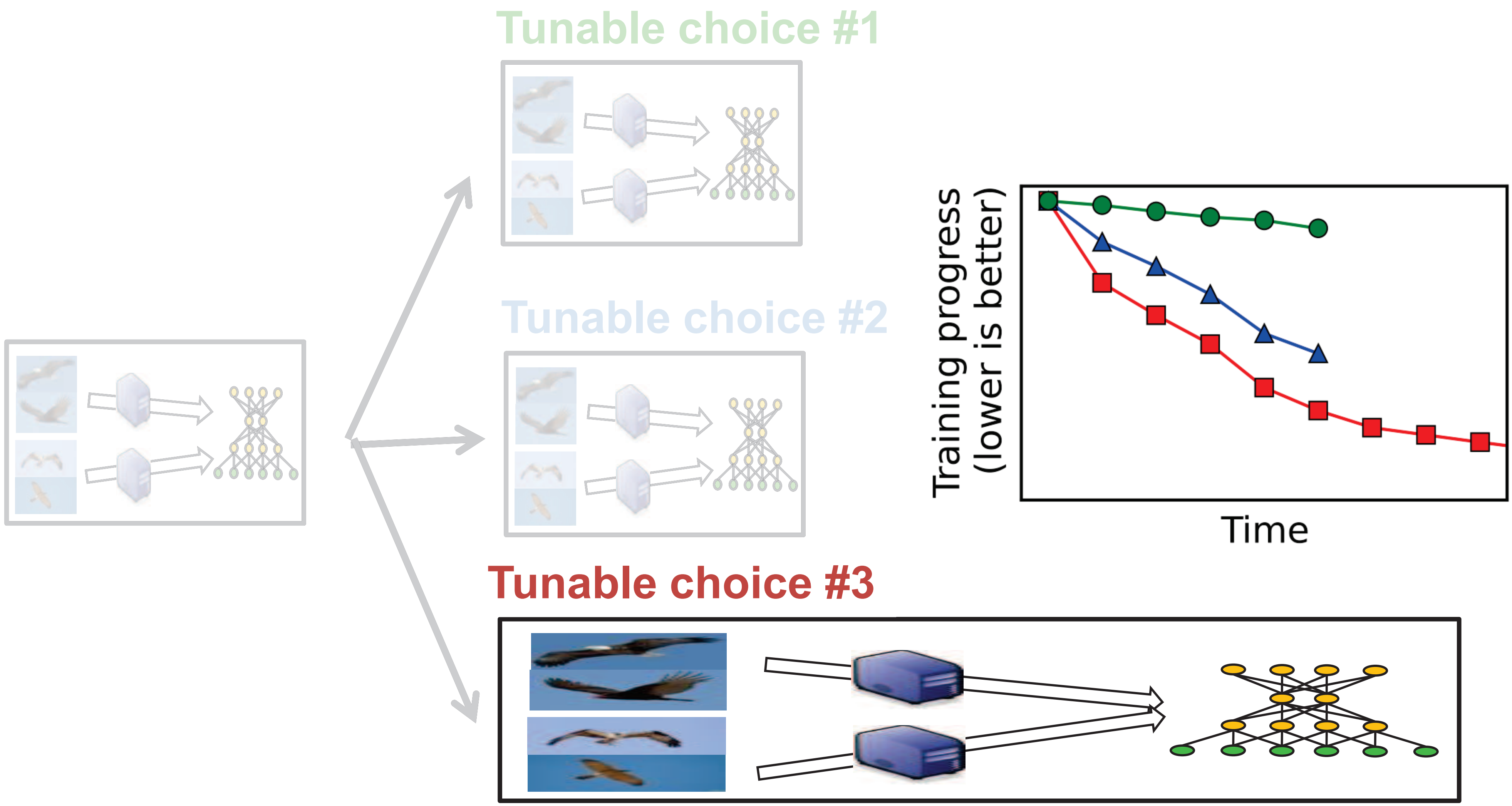


- Need a long enough trial time to get stable progress
- Double trial time until converging branch found

Outline

- Motivation
- Traditional tuning approaches
- System design for better tuning
  - Trying tunables in time sharing branches
  - Tuning design
  - Adjusting tunables
- Experiment results

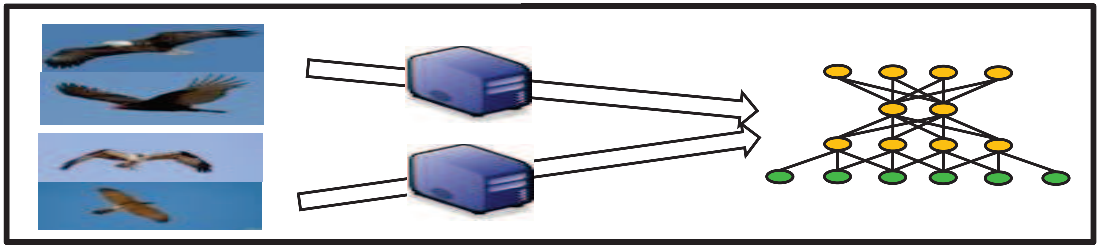
Adjusting tunables during the training



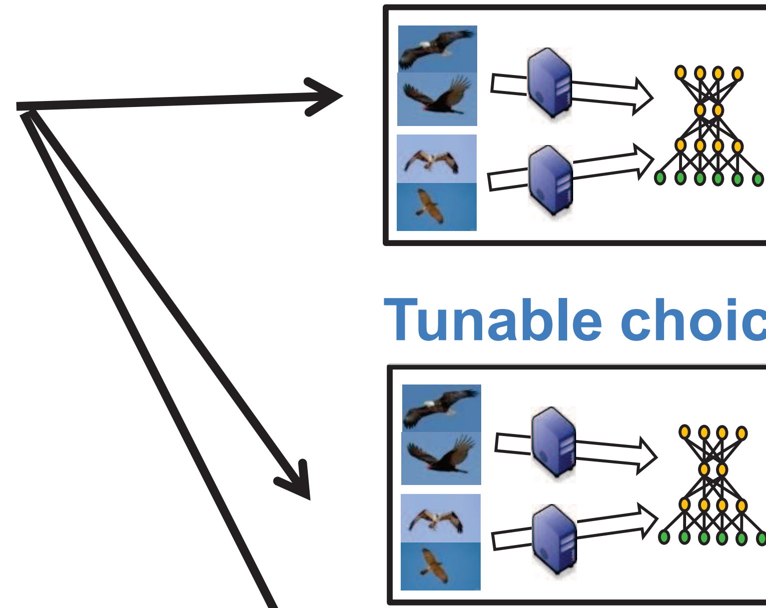
Keep training the best branch

Adjusting tunables during the training

Tunable choice #3

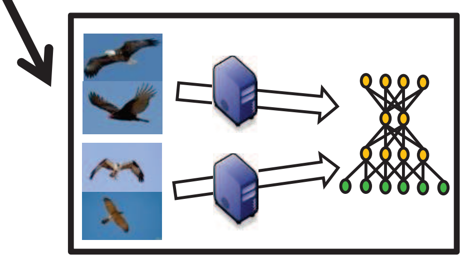


Tunable choice #4



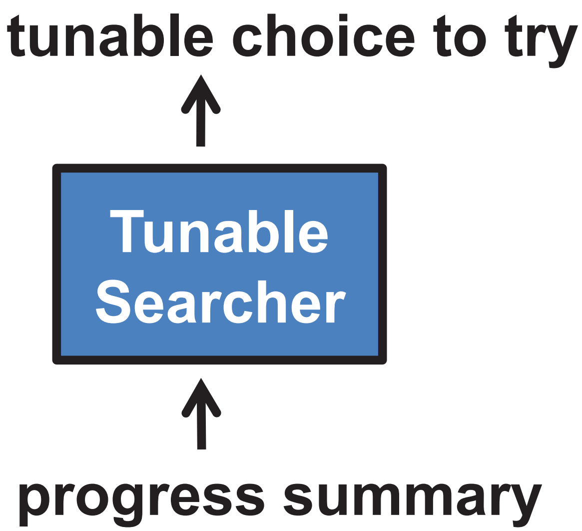
Tunable choice #5

Tunable choice #3

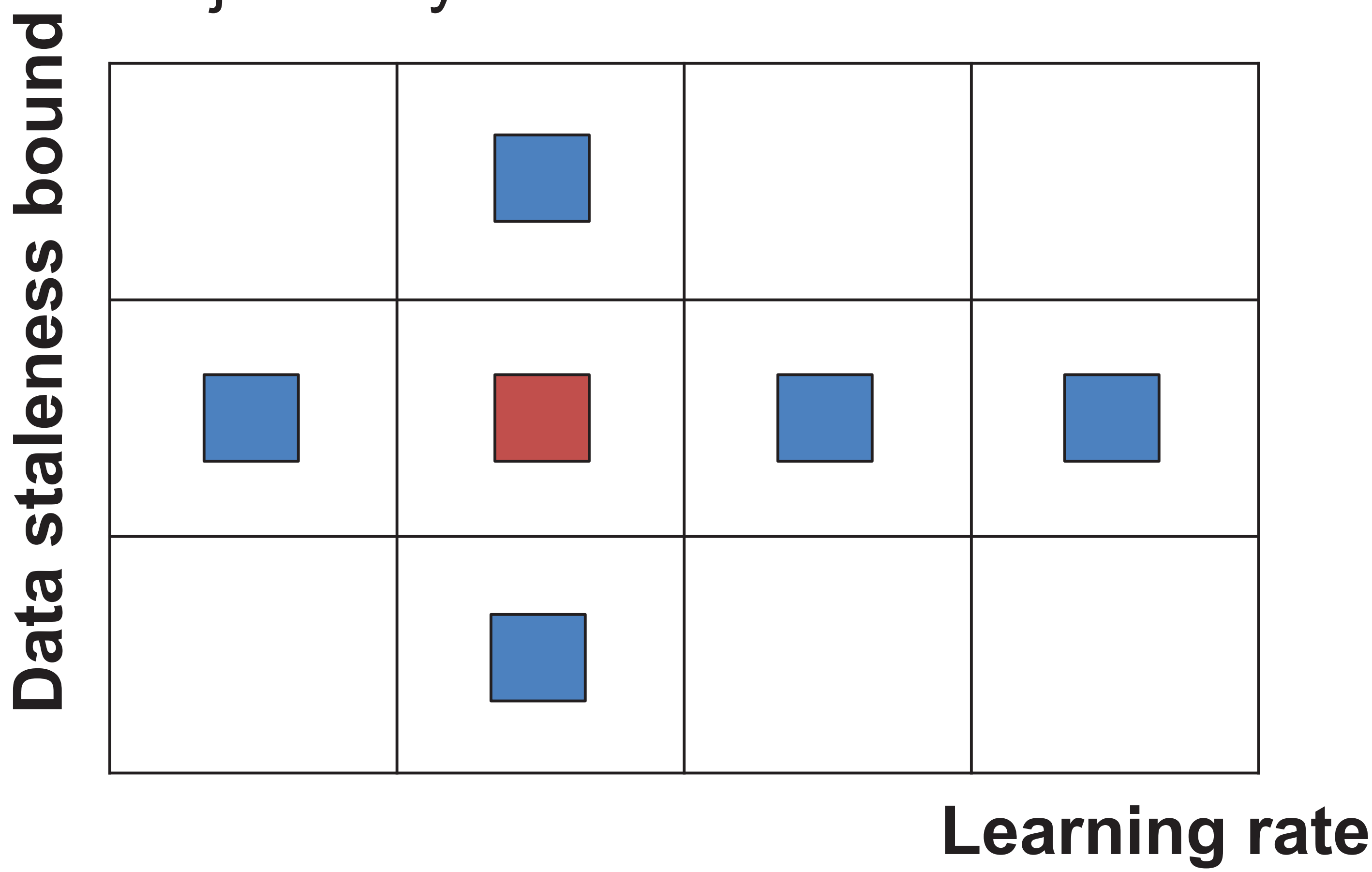


- How to adjust tunables
  - just fork and search again
- When to adjust tunables
  - when progress slows
    - e.g., accuracy stops improving
  - or after running for 10x searching time

MarginalSearcher for adjusting tunables



- MarginalSearcher
  - start from an initial tunable choice
  - adjust only one dimension



Outline

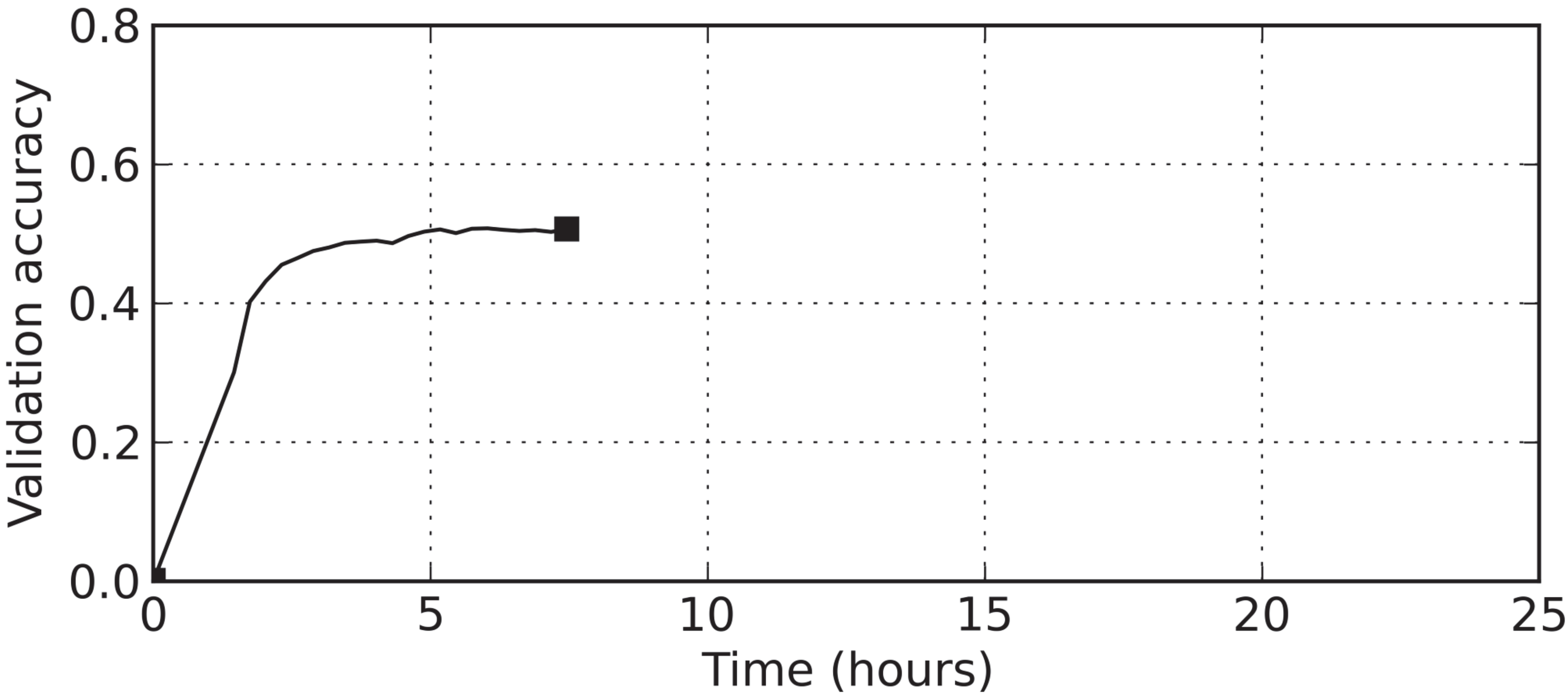
- Motivation
- Traditional tuning approaches
- System design for better tuning
  - Try tunable choices in time sharing branches
  - Tuning design
- Experiment results

Experimental setups

- Image classification w/ deep neural network
- Datasets & models
  - ImageNet ILSVRC12 with Inception-BN
    - 1.3 million images, 1000 classes
- Tunables
  - learning rate
  - training batch size
  - data staleness bound
- Hardware
  - 8 machines, each with one Titan X GPU

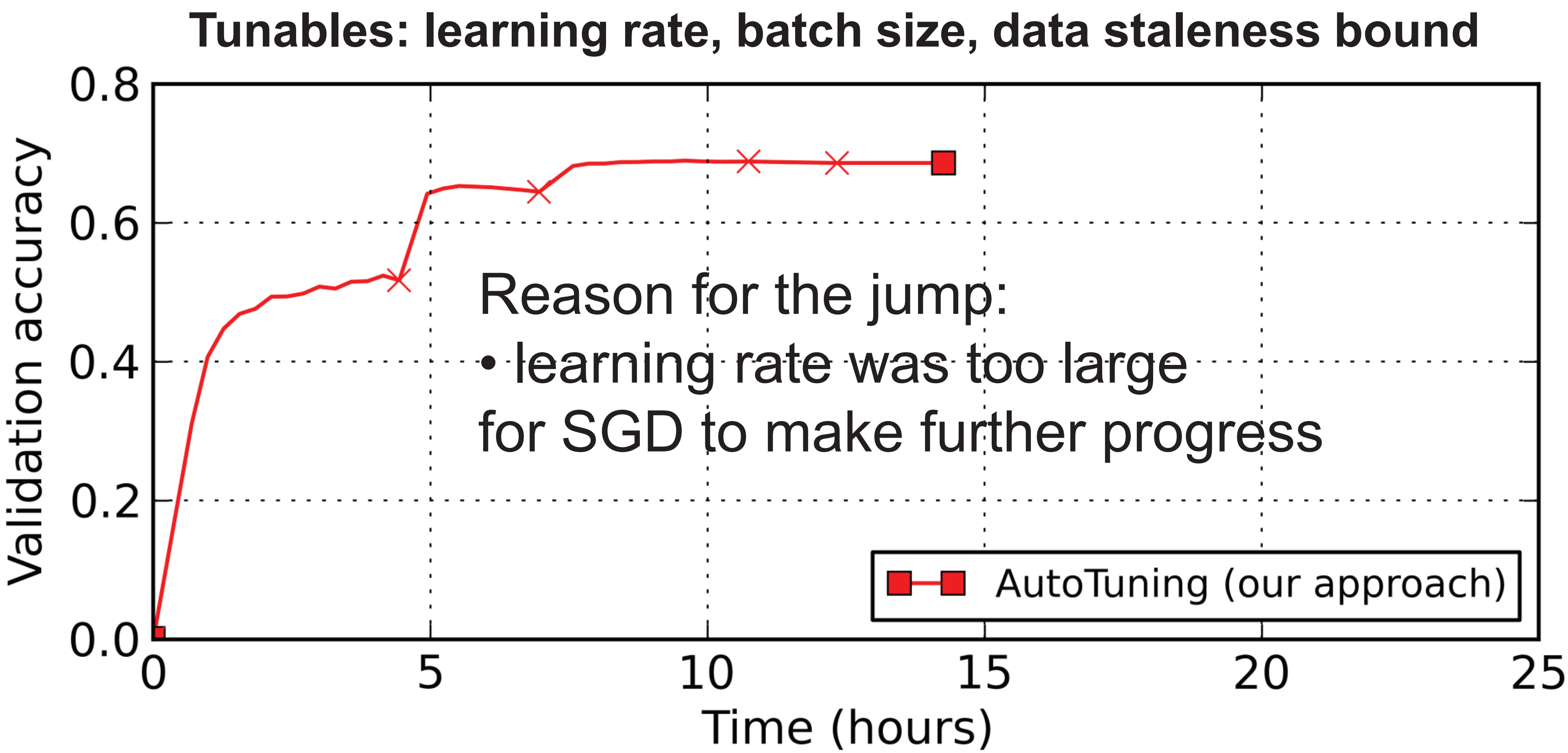
Experimental setups

Tunables: learning rate, batch size, data staleness bound



- **Test on validation set for every training data pass**
- **Convergence condition**
  - **validation accuracy stops increasing for 5 tests**

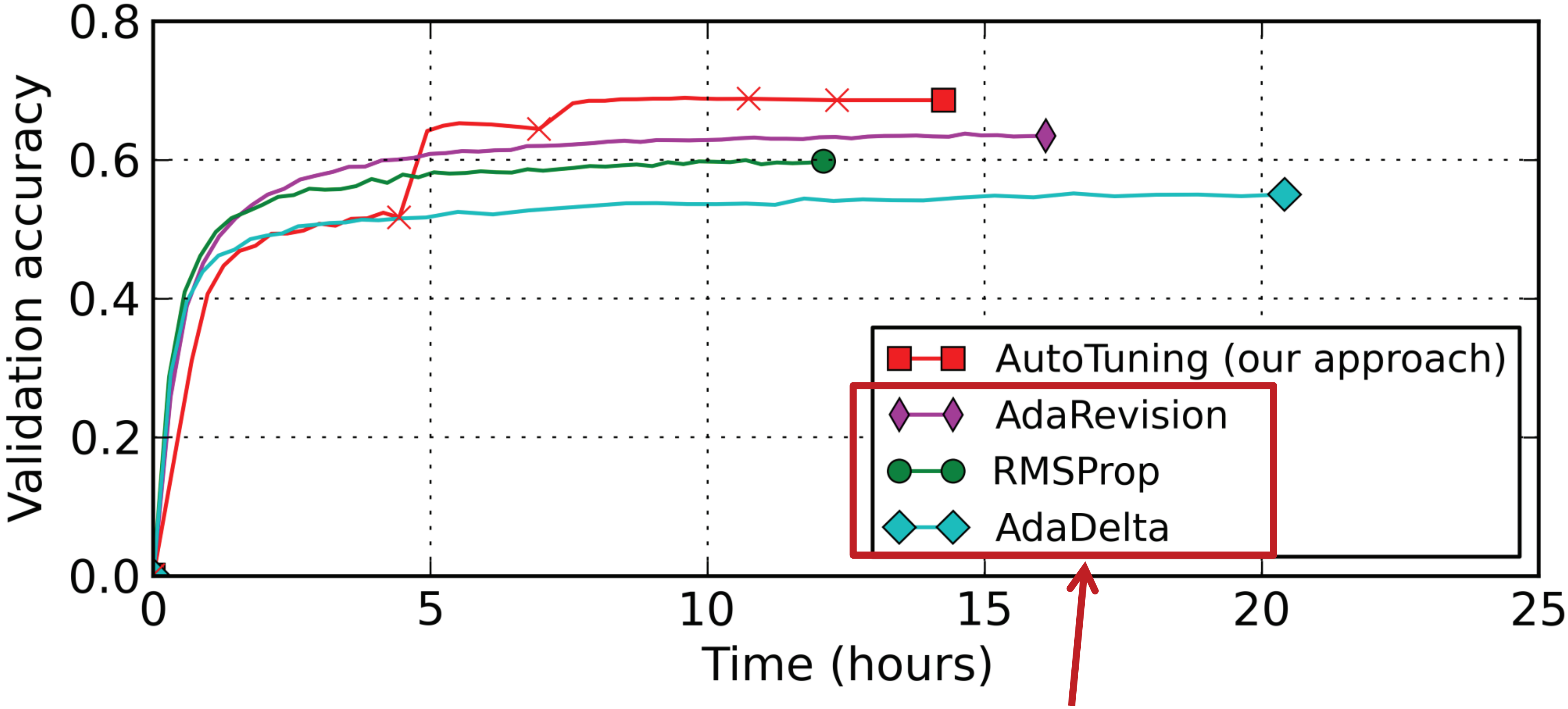
Automatic tuning



- **Search tunables at the beginning**
- **Adjust tunables when convergence condition met**
    - **and test for one more time**

Compare with SGD LR tuning algorithms

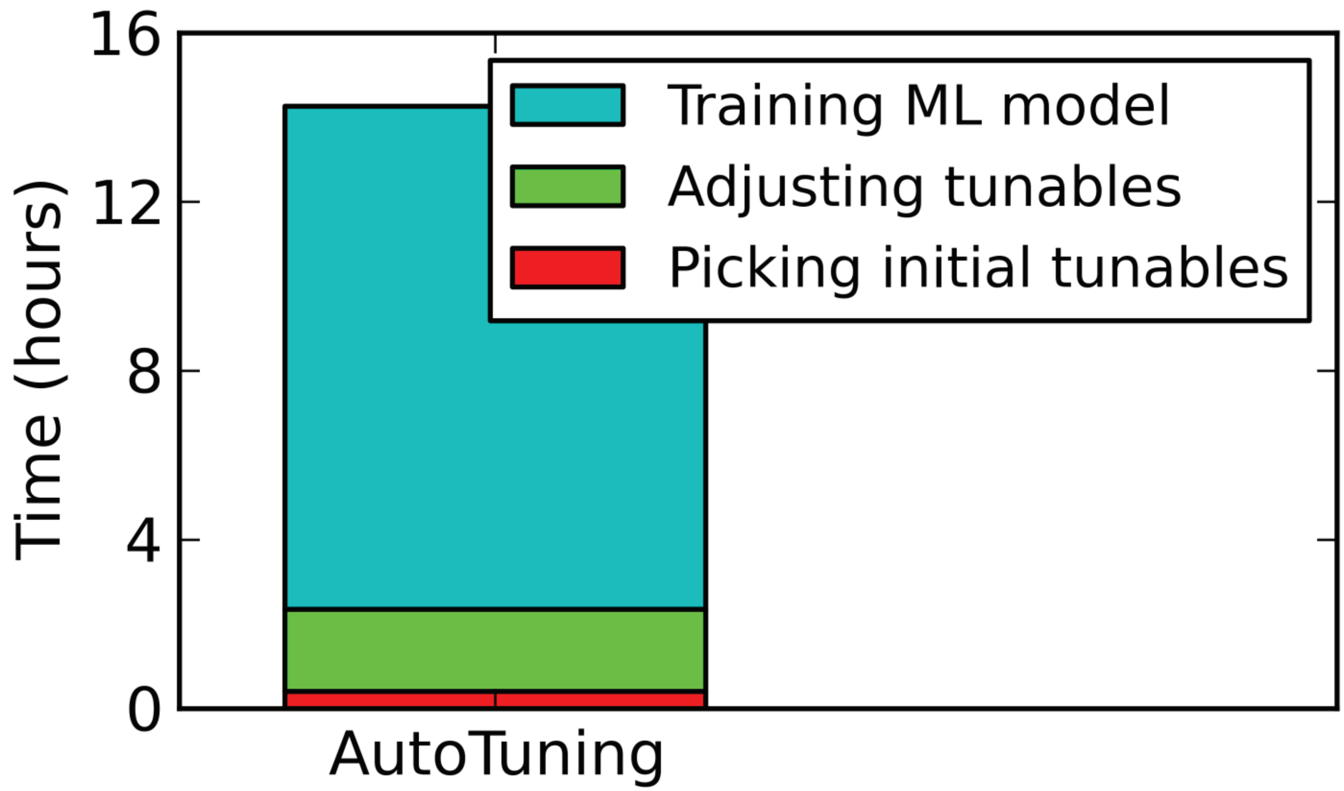
Tunables: learning rate, batch size, data staleness bound



- **State-of-art SGD learning rate tuning algorithms**
  - **initial learning rate still needs to be picked**
  - **converge to lower model accuracies than our approach**

Auto-tuning has low overhead

Tunables: learning rate, batch size, data staleness bound



- **2 hours tuning, 12 hours training**
  - **only 20% overhead**

Other apps/models

- Cifar10 data with AlexNet model
  - by adjusting tunables, faster than best constant one
- Matrix factorization
  - robustly identifies good tunable choices
  - among reasonable by-hand options
    55% are over 10x slower

Conclusions

- A system for automatic ML tuning
  - try & evaluate tunable choices in training branches
- Automatically pick and adjust tunables
  - work on many apps/models
  - without too much overhead

References

- **[HyperparamOpt]** J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In NIPS, 2012.
- **[AdaRevision]** B. McMahan and M. Streeter. Delay-tolerant algorithms for asynchronous distributed online learning. In NIPS, 2014.
- **[RMSProp]** T. TielemanWang and G. Hinton. Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.
- **[AdaDelta]** M. D. Zeiler. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.