10-605/10-805: Machine Learning with Large Datasets

Fall 2022

Hyperparameter Tuning

Announcements

- Mini-project Proposals due on 11/11 (tomorrow!)
- Recitation this week: HW4 review
- Guest lecture Thursday (11/17): Krishna Rangasayee SiMa.ai
  - Topic: ML on Embedded Edge Devices)
  - Virtual / Zoom
  - Will not be recorded — please show up!
  - **Will be covered on Exam 2**

Key course topics

Data preparation
- Data cleaning
- Data summarization
- Visualization
- Dimensionality reduction

Training
- Distributed ML
- Large-scale optimization
- Scalable deep learning
- Efficient data structures
- **Hyperparameter tuning**

Inference
- Hardware for ML
- Techniques for low-latency inference (compression, pruning, distillation)

Infrastructure / Frameworks
- Apache Spark
- TensorFlow
- AWS / Google Cloud / Azure

Advanced topics
- Federated learning
- Neural architecture search
- Productionizing ML

MOVING FORWARD

What makes deep learning expensive?

Training requires lots of computation

- Specialized hardware (GPUs, TPUs) can help with matrix computations
- Can use advanced iterative optimization methods
- Can parallelize training

Hyperparameter tuning and neural architecture search (NAS) make this worse

- Lots of knobs to tune!

Resulting models can be large!

- Can be expensive to store model, perform inference

Outline

1. HP Search background
2. HP tuning at scale: Early stopping
3. Formulation as a multi-armed bandit problem

Outline

1. **HP Search background**
2. HP tuning at scale: Early stopping
3. Formulation as a multi-armed bandit problem

What are some hyperparameters we've seen?

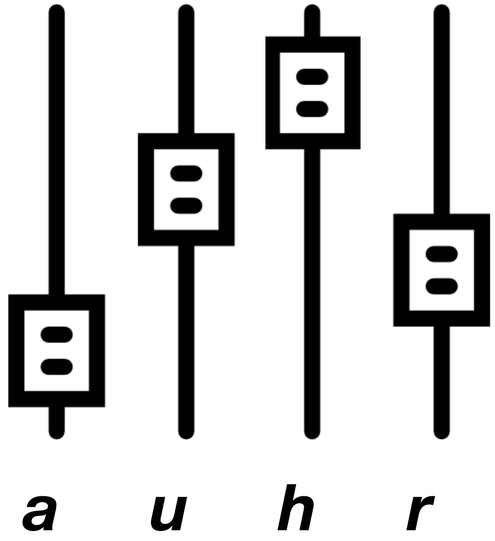
- learning rate
- batch size
- regularization parameter
- size of layers, number of layers
- activation function
- loss function
- optimizer
- momentum, moving average
- data augmentation
- learning rate scheduling

What are some hyperparameters we've seen?

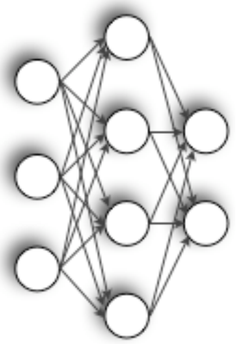
- regularization parameter (ridge regression, regularized logistic regression, etc)
- size of reduced dimension (PCA, t-SNE, random projections)
- perplexity (t-SNE)
- rank (Nystrom method)
- size of hash function / table (feature hashing, count-min sketch)
- depth of tree, # of thresholds (planet, yggdrasil)
- step size, mini-batch size (mb-SGD & variants)
- moving average (RMSProp, Adam)
- momentum (heavy ball method, NAG, Adam)
- architectural parameters (layers, nodes per layer, activations)
- …

Model space defined by

‘hyperparameters’

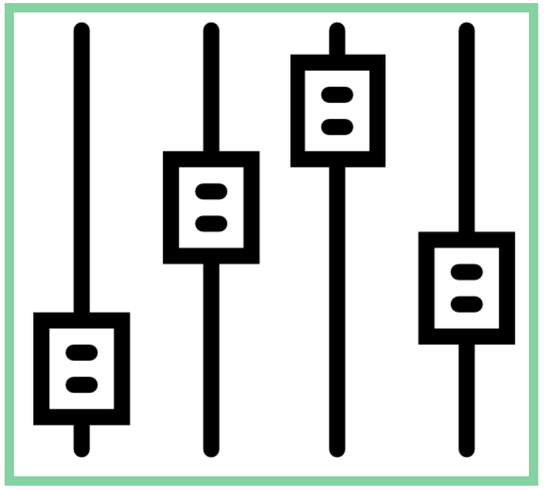


- activation function (***a***)
- # units per layer (***u***)
- # hidden layers (***h***)
- regularization (***r***)

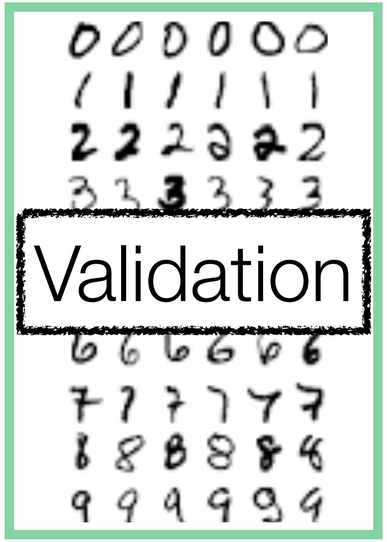


Black-box Solver

$\widehat{f}$



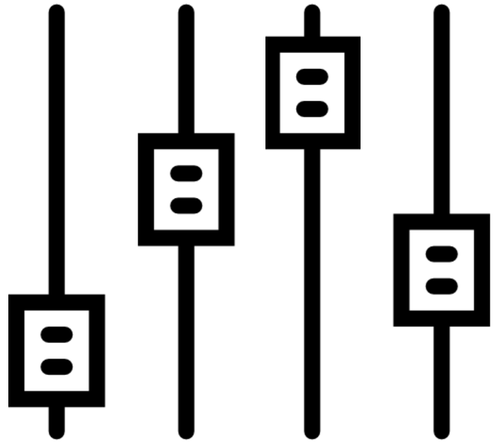
a u h r



$\widehat{f}$

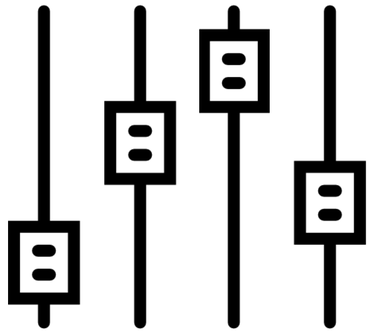
Predictive Error

0.058



a ***u*** ***h*** ***r***

0.058



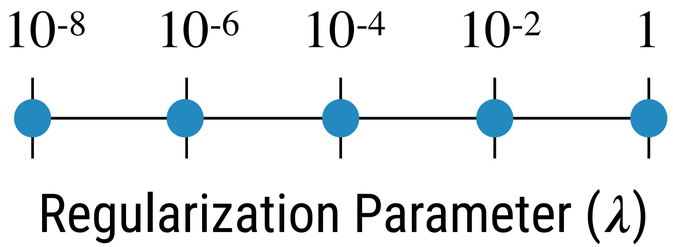
How can we **efficiently** identify **high-quality** hyperparameters?

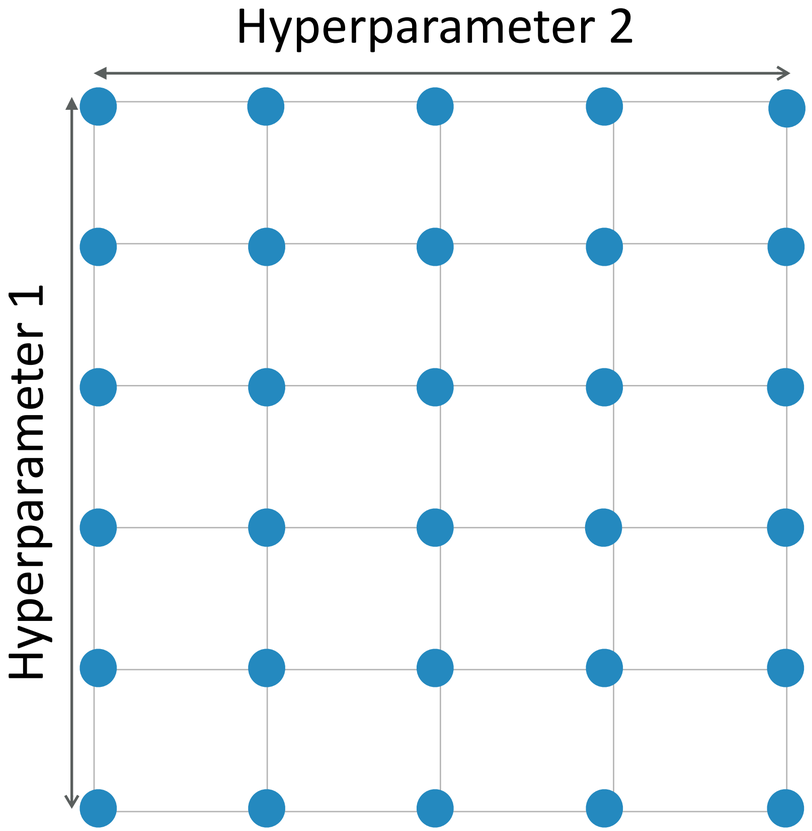
Efficiency ⟺ resources consumed

Quality ⟺ predictive error

RECALL

HP tuning baselines



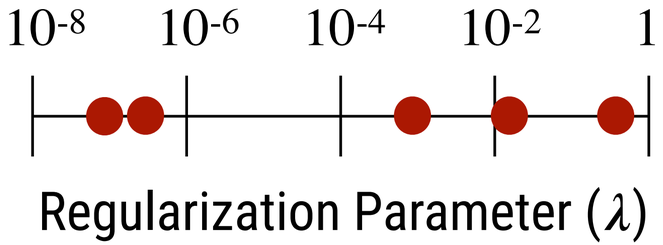


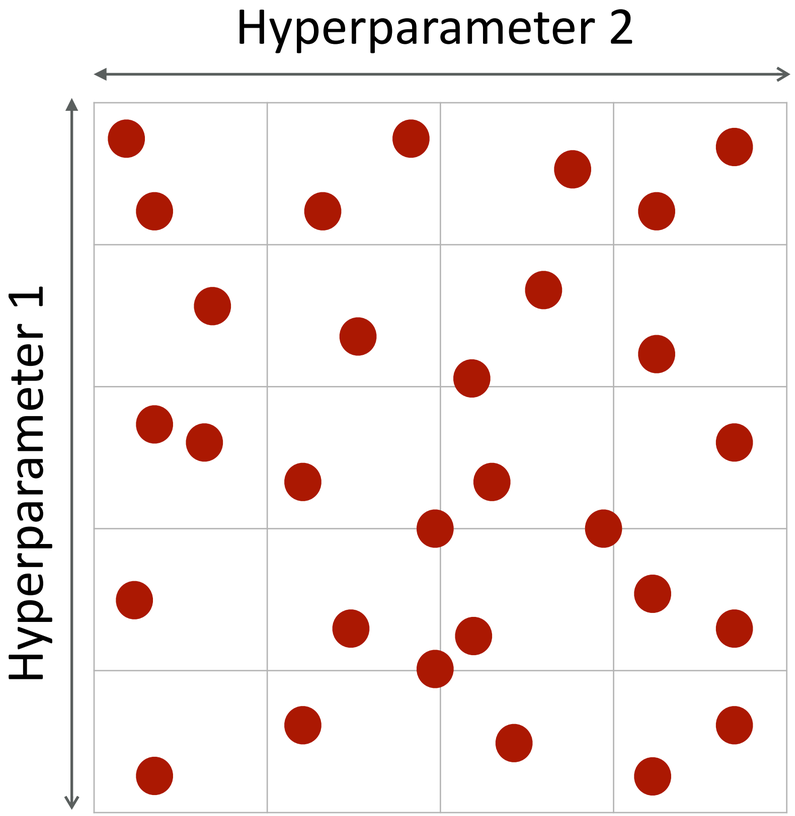
Grid Search:

- Create a 'grid' by evenly discretizing search space (e.g., linear or log scale)

RECALL

HP tuning baselines





Random Search:

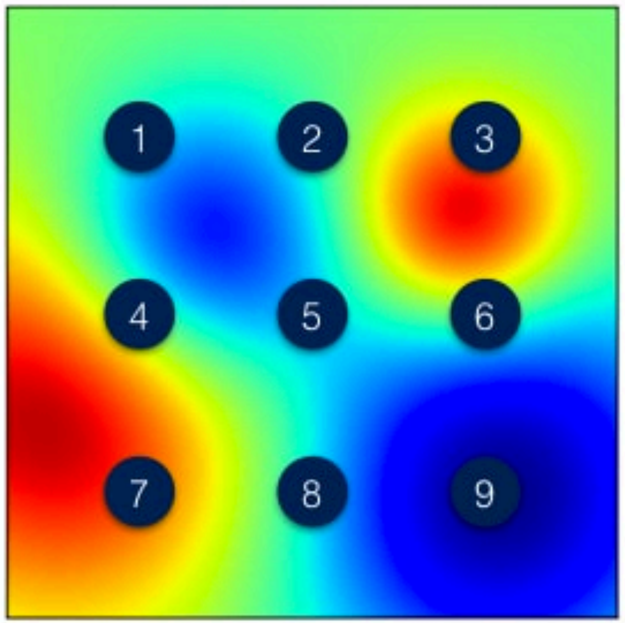
- Randomly select hyperparameters within a range

Grid Search:

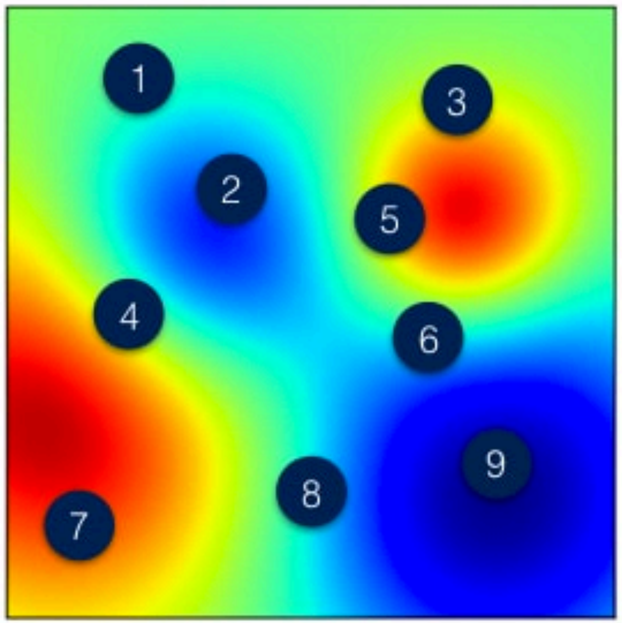
- Create a 'grid' by evenly discretizing search space (e.g., linear or log scale)

*We'll focus in this lecture on **Random Search** as a baseline; you'll explore Grid Search vs Random Search in HW5*

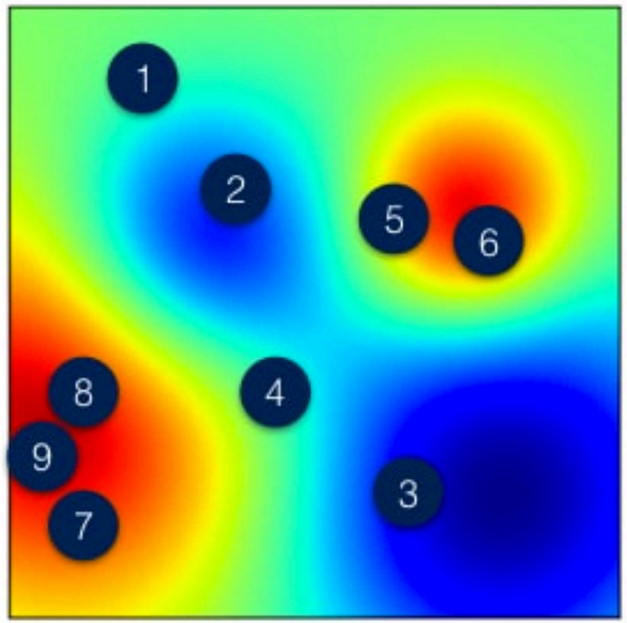
Another approach: adaptive optimization



Grid Search



Random Search

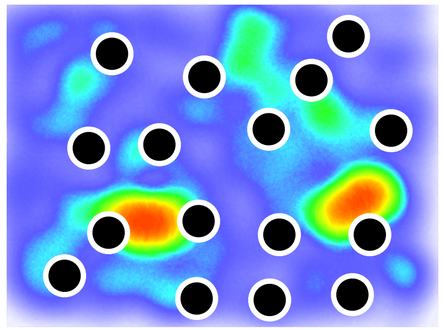


Adaptive Selection

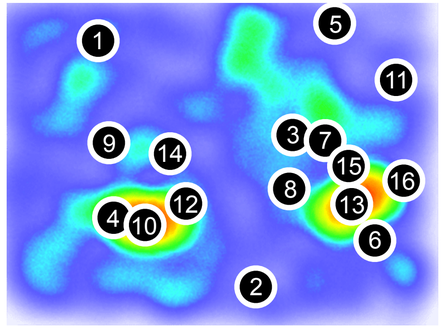
Random vs. adaptive

Assume d hyperparameters, n hyperparameter configurations

Random



Adaptive



Random vs. adaptive

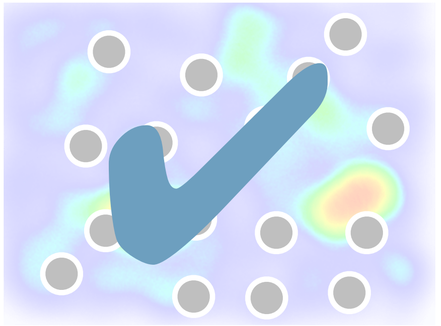
Assume d hyperparameters, n hyperparameter configurations

Case 1: n is ***exponential*** in d

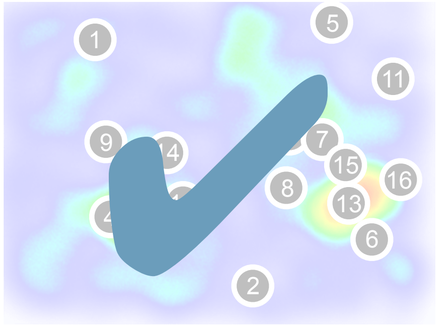
✔ Can **cover** the space (see HW5)

✔ **Adaptivity** works well!

Random



Adaptive



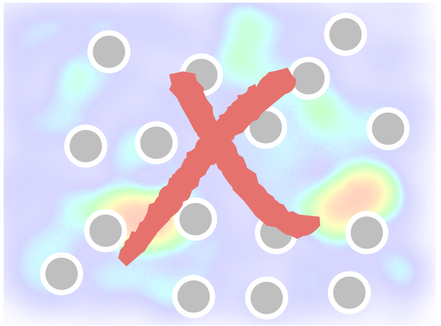
Random vs. adaptive

Assume d hyperparameters, n hyperparameter configurations

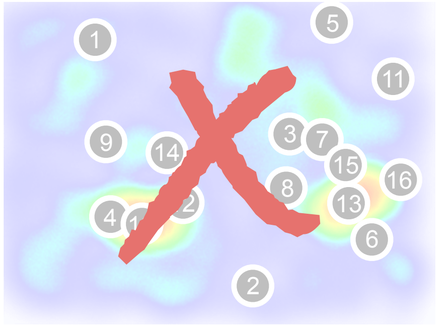
Case 2: n is ***linear*** in d

- ✗ **Cannot cover space**
- ✗ This also **limits adaptivity**

Random



Adaptive



Standard regime for deep learning

Hyperparameter Tuning for Deep Learning

*"One major drawback of current architectures is that they are expensive to train, typically requiring **days to weeks of GPU time**"*

*"We report results … corresponding to over **250,000 GPU hours** on the standard WMT English to German translation task"*

"Massive Exploration of Neural Machine Translation Architectures" (Britz et. al., 2017)

10K Train Models and 29 Compute Years!!

HOW CAN WE DO THIS AT SCALE??

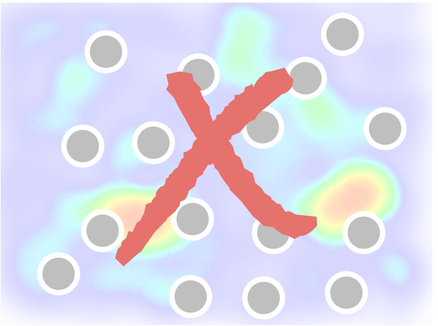
Random vs. adaptive

Assume d hyperparameters, n hyperparameter configurations

Case 2: n is ***linear*** in d

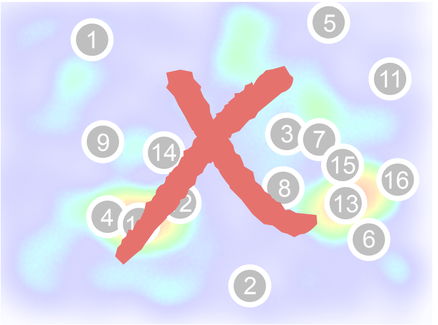
- ✗ **Cannot cover space**
- ✗ This also **limits adaptivity**

Random



Downsample to evaluate more configurations on same budget

Adaptive

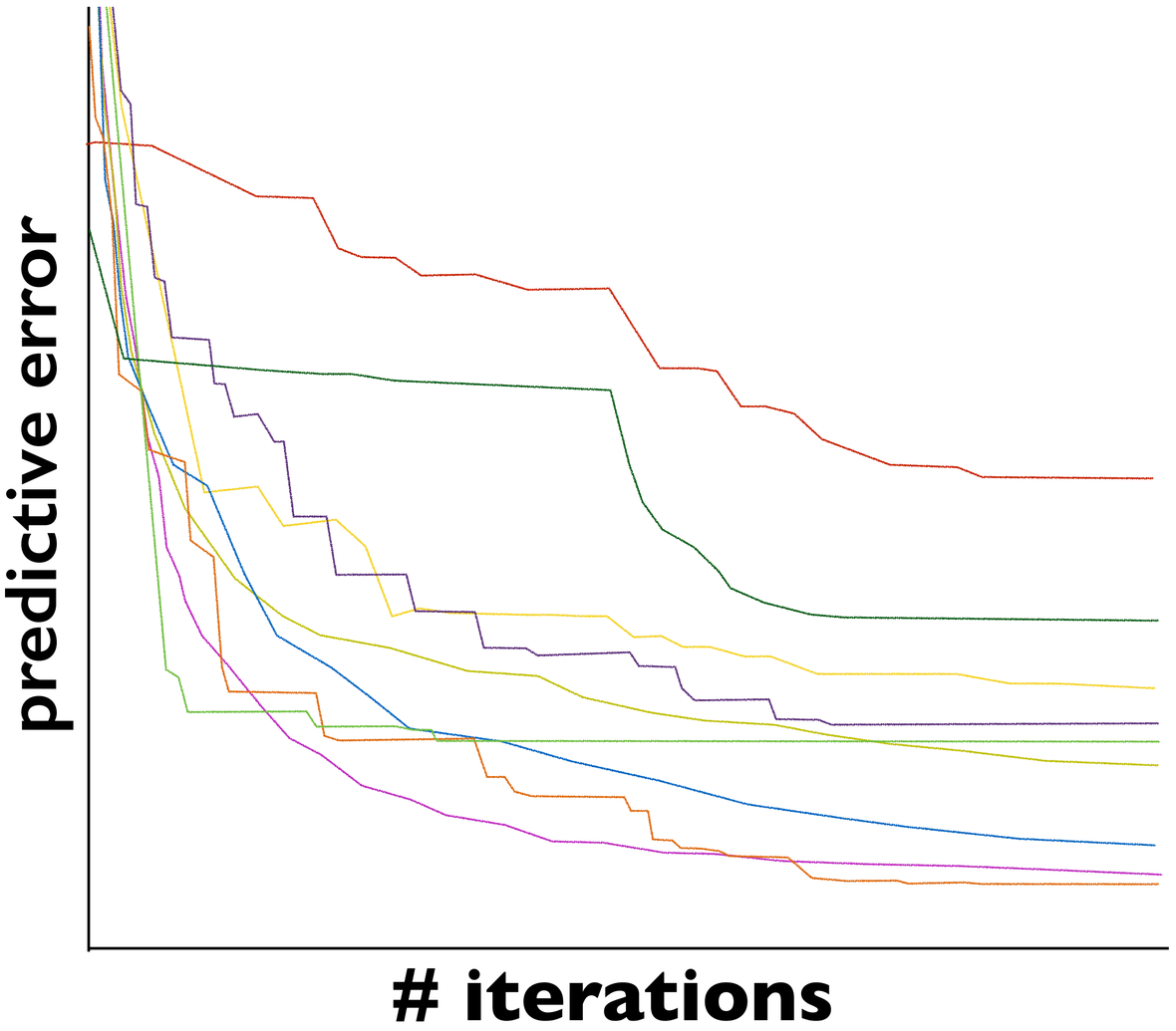


Outline

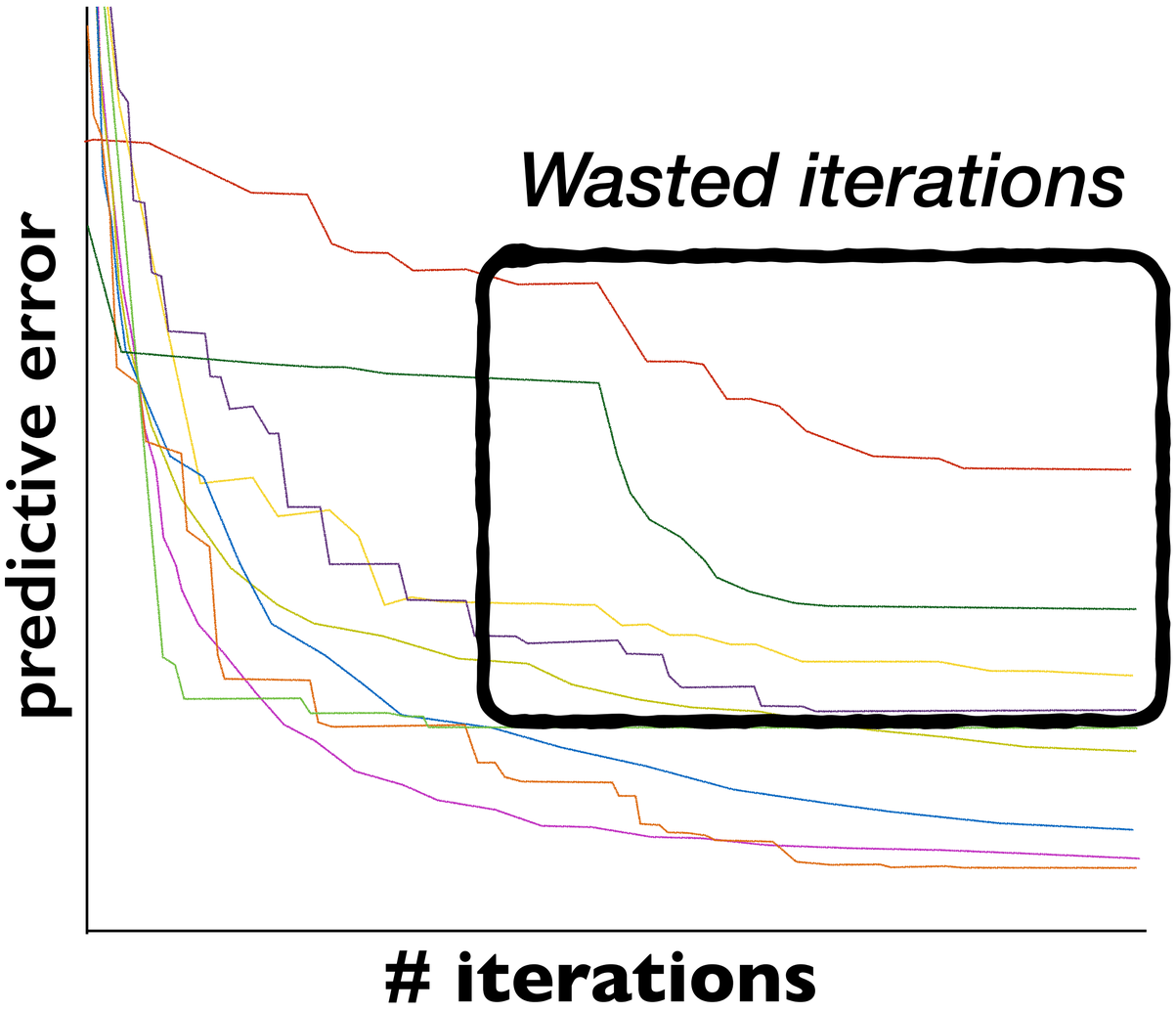
1. HP Search background
2. **HP tuning at scale: Early stopping**
3. Formulation as a multi-armed bandit problem

Many learning algorithms are iterative!

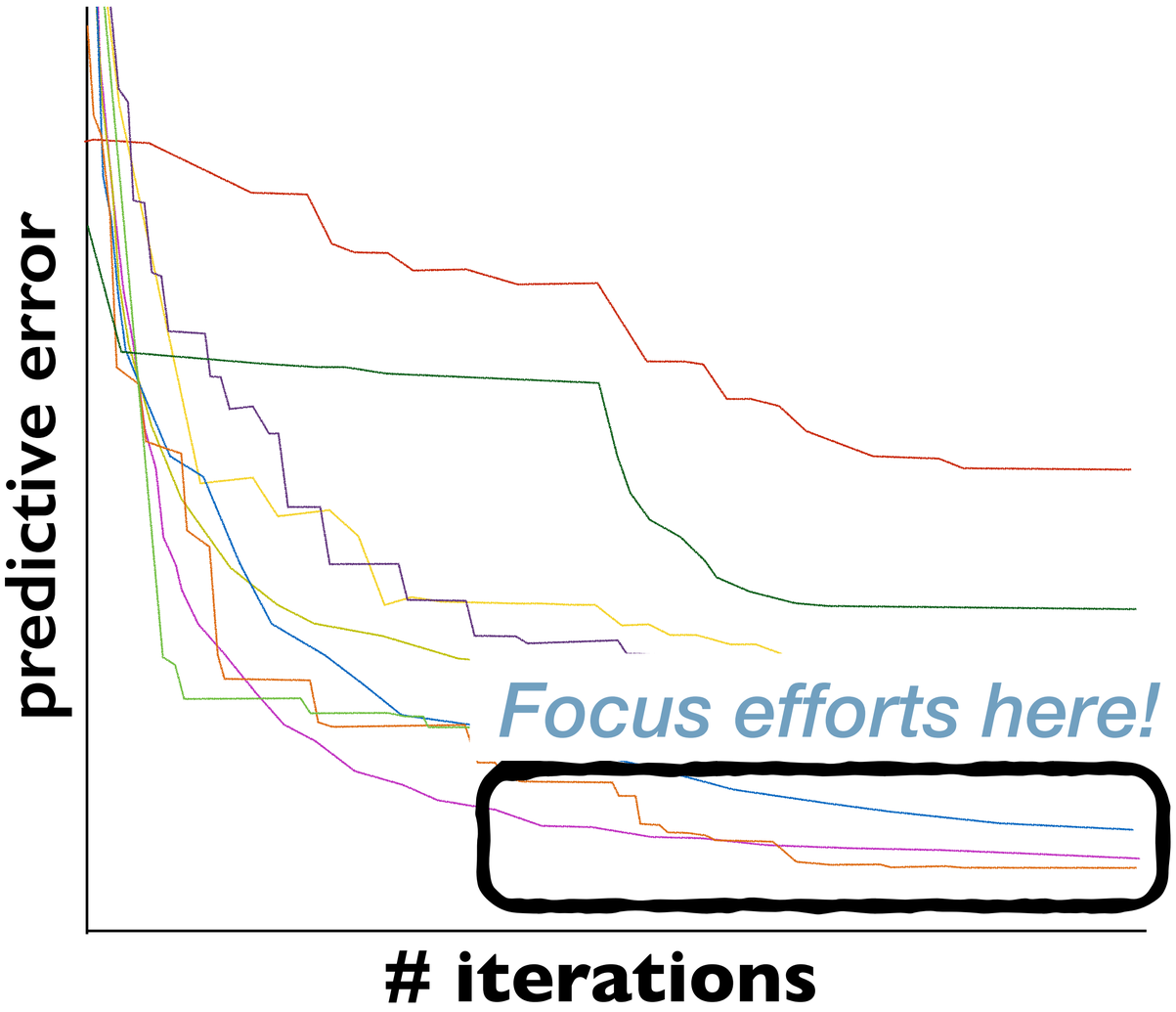
Downsampling Iterations



Downsampling Iterations



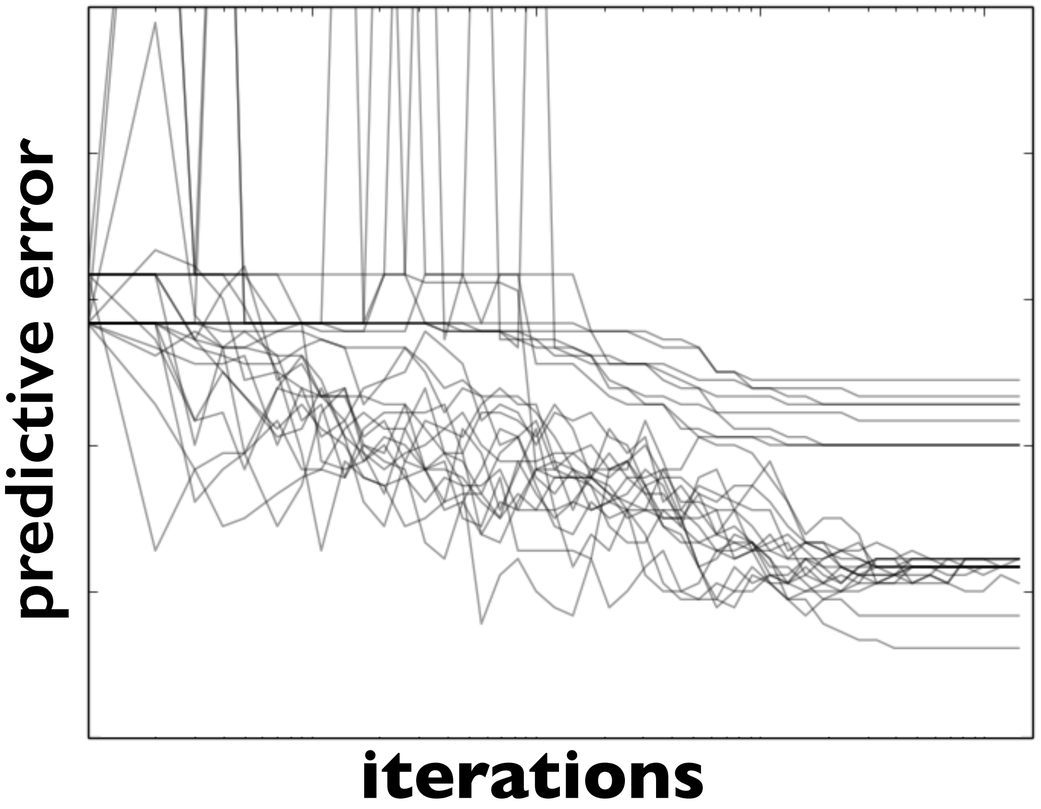
Downsampling Iterations



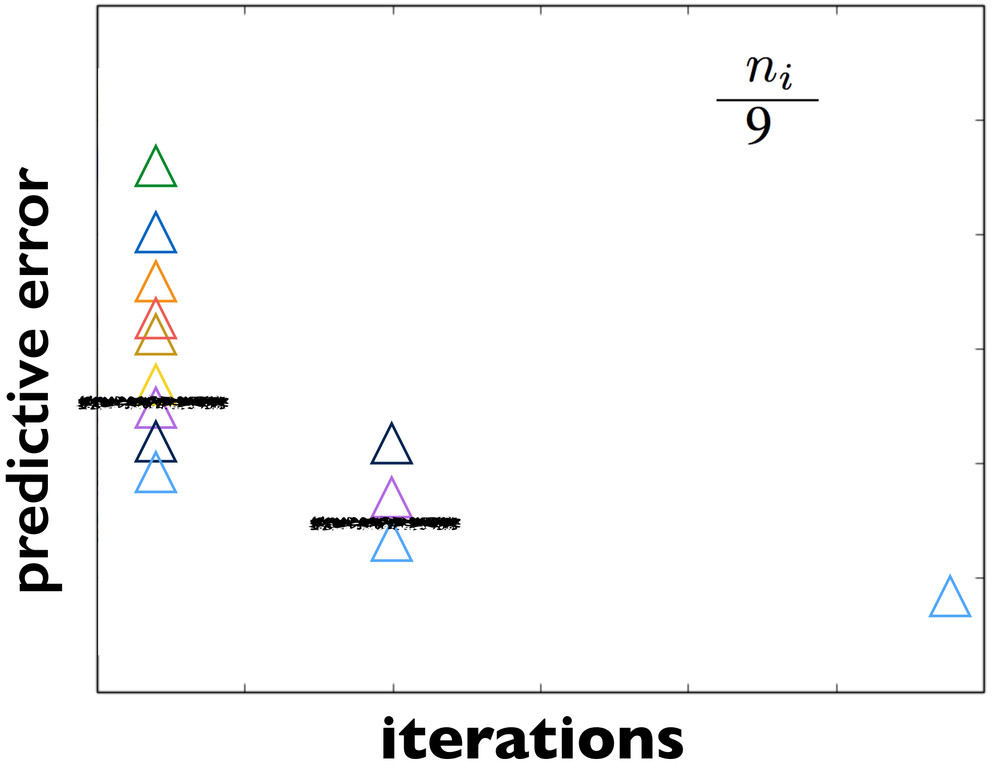
What could go wrong?

Sequences can be **non-monotonic**, **non-smooth**

- How can we “safely” discard a configuration?



1st ingredient: Successive “Halving”



r: Max iterations [**81**]

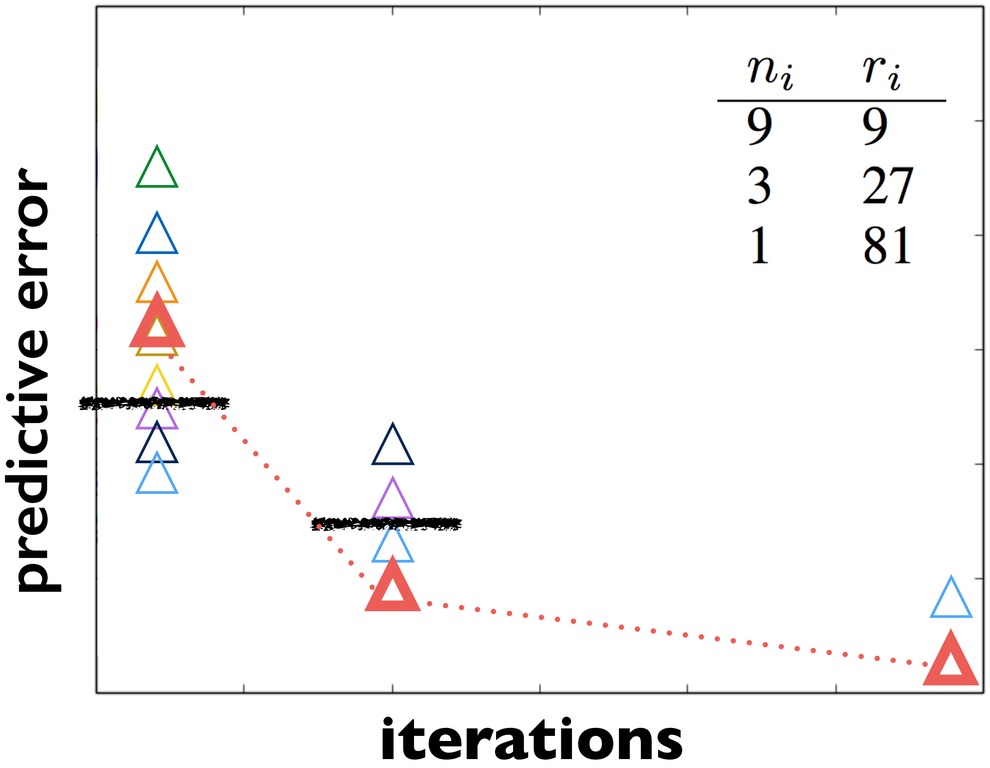
B: Total budget [**3r**]

n: # configurations [**9**]

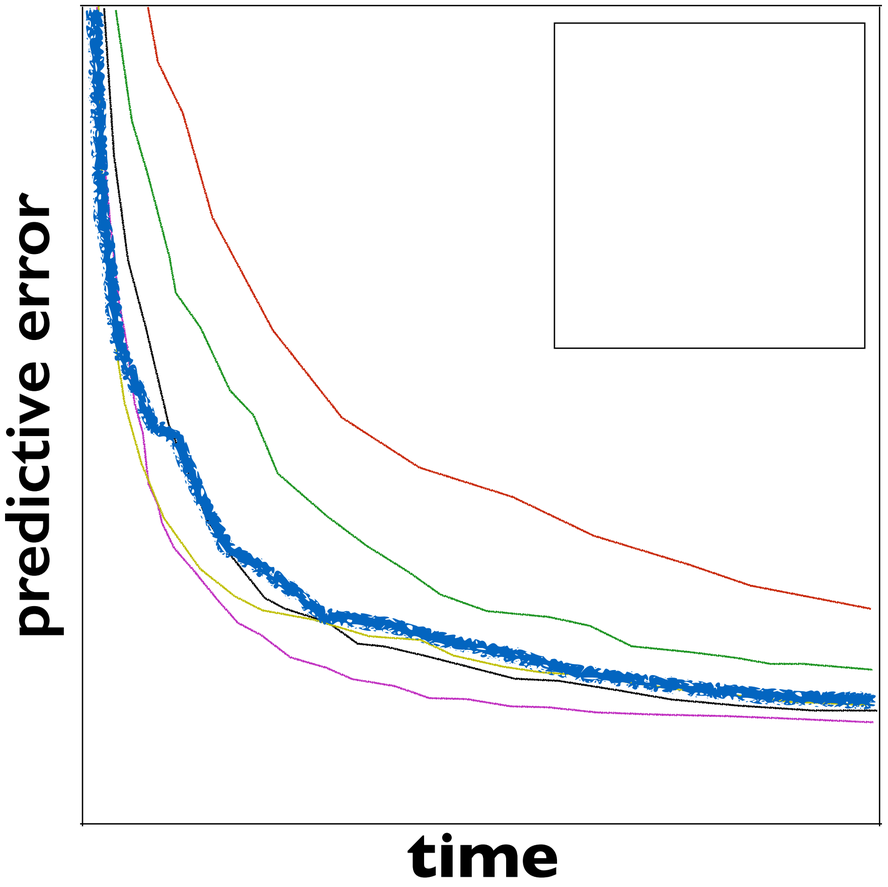
Is Downsampling “Safe”?

It **increases** n but may **introduce errors**

Depends on **unknown convergence properties**



2nd ingredient: Safe Downsampling



Max total budget: **[5r]**

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

Optimal choice is **unknowable** a priori

Try them all!

PUTTING IT ALL TOGETHER

Hyperband

Key ideas:

- Use random search to find configurations
- BUT, reduce evaluation cost with downsampling / successive ‘halving’
- Try multiple ‘brackets’ of successive halving to downsample safely

→ *can (approximately) evaluate more configurations within the same budget*

Theoretical results [high-level]:

- if you know convergence behavior *a priori*, there exists a successive ‘halving’ bracket that is optimal
- if you **don’t** know this, Hyperband will do almost as well without these assumptions

Li et. al., 2018

CIFAR-10 - Cuda-Convnet

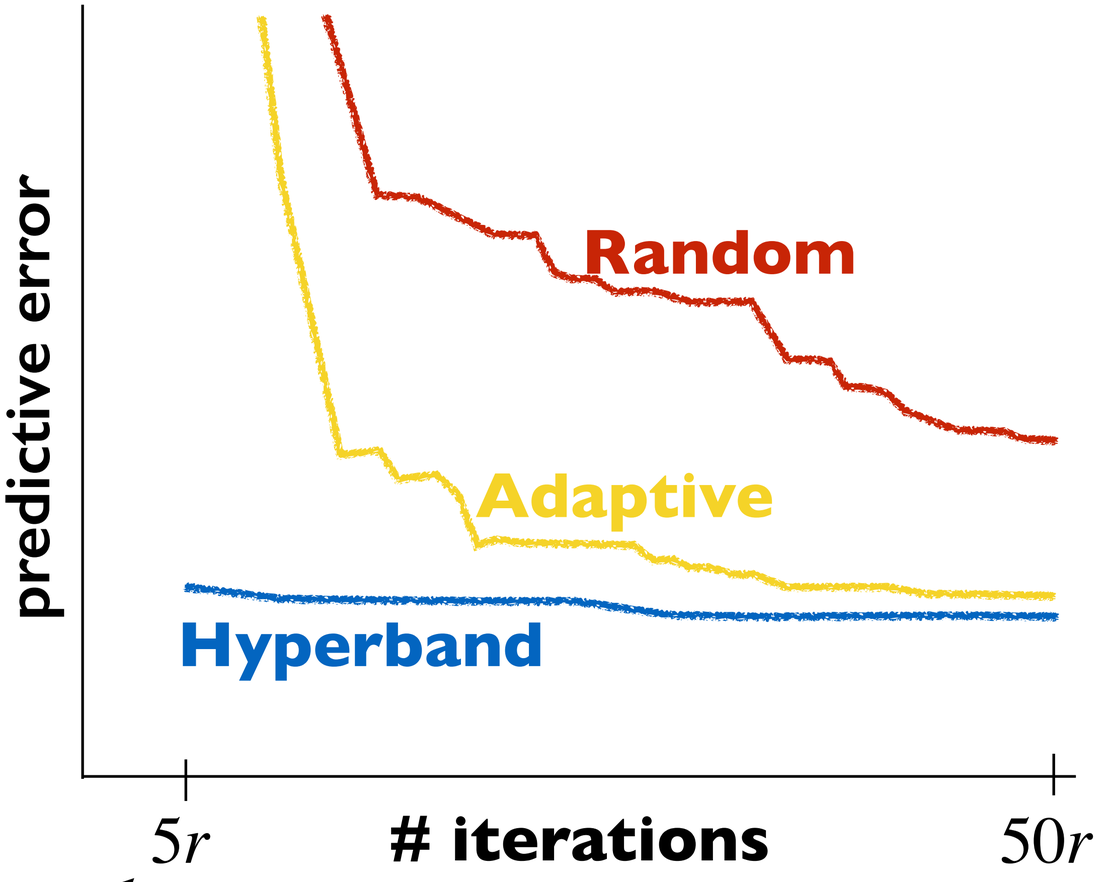
Speedups

>50x over Random

10x over Adaptive

✔ **Lower final error**

✔ **Lower variance**



Hyperband has considered 256 configurations!


Microsoft


Preferred
Networks


OPTUNA


facebook


Neural Network Intelligence


TensorFlow


Katib


PYTORCH


scikit
learn


DASK


Determined AI


K
Keras


RAY


mxnet


Polyaxon


Caffe2


Oríon


R

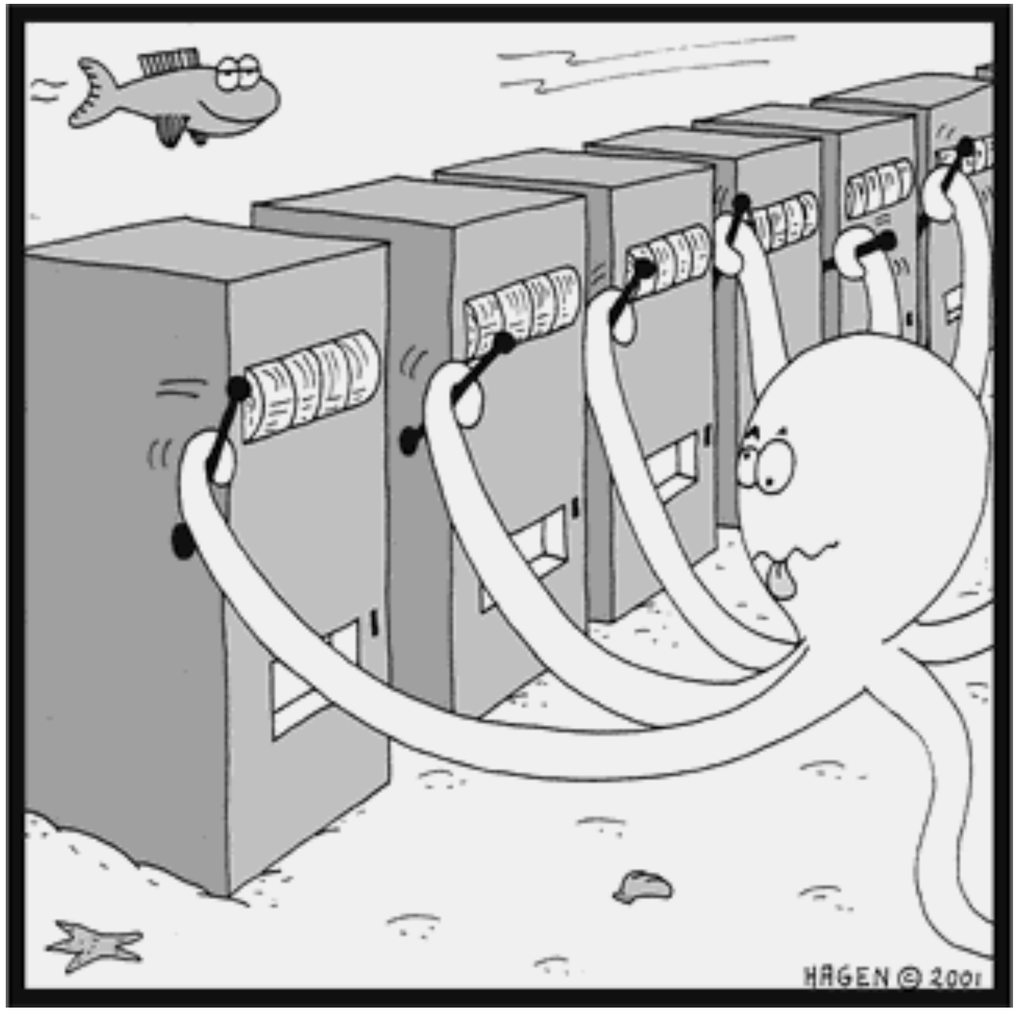

HyperSearch


IBM

Outline

1. HP Search background
2. HP tuning at scale: Early stopping
3. **Formulation as a multi-armed bandit problem**

Multi-armed bandit



- Name comes from problem of trying to determine which slot machine ‘arm’ to pull in a row of slot machines
  - Explore: pick a new arm
  - Exploit: pick the same arm
- **Many uses throughout RL, ML & beyond**

“**Best arm identification**”
Given n configurations/arms, how many resources are required to find the best one?

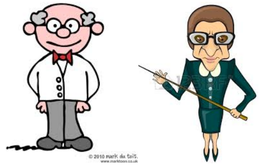
slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

Project 2

Judges



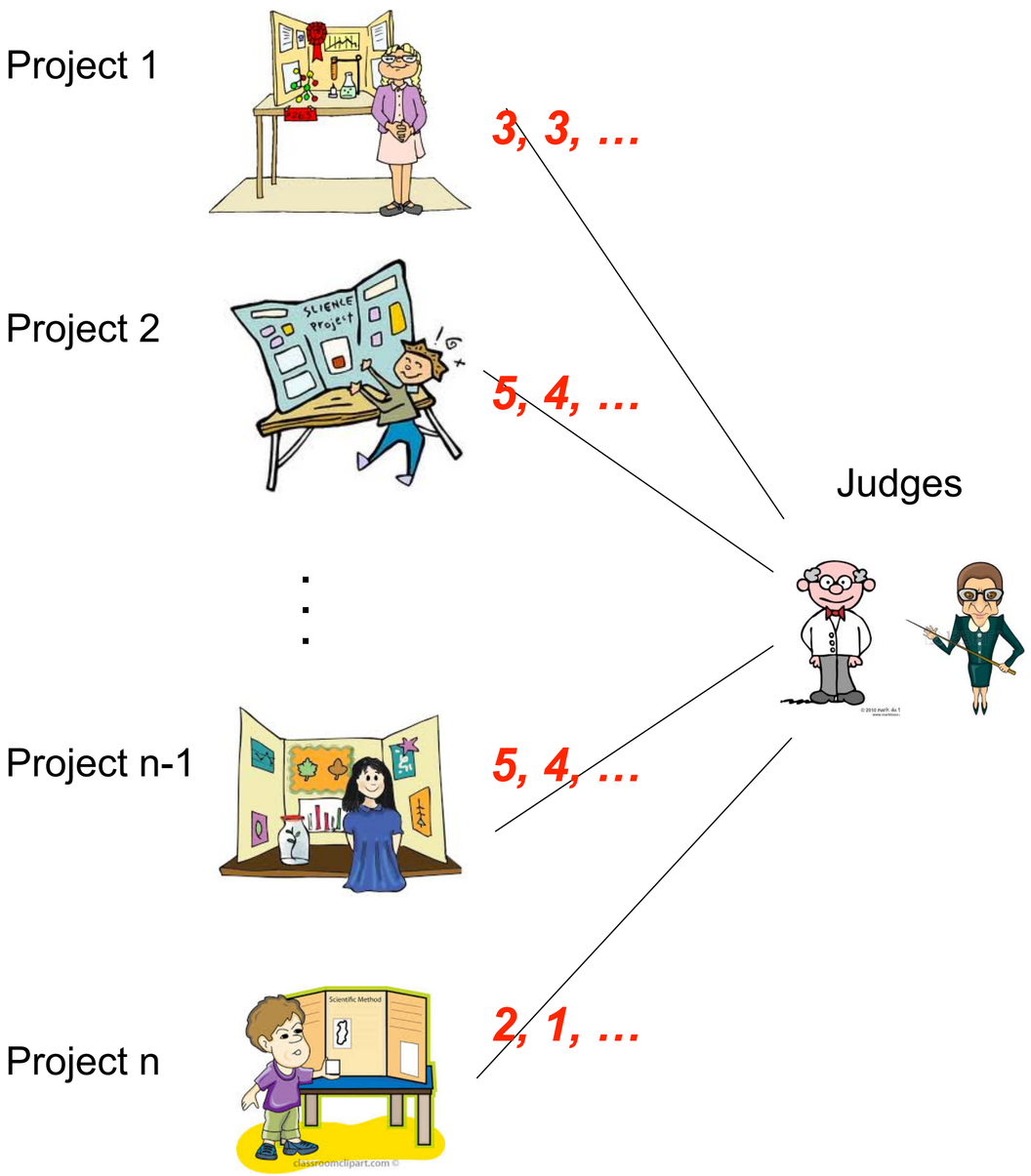
.
.
.

Project n-1

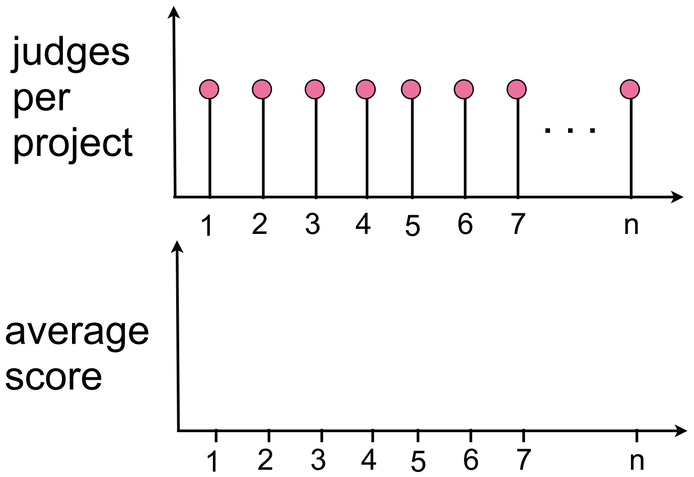
Project n

slide credit: Kevin Jamieson

Choosing the winner of a Science Fair



Uniform Judge Allocation



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

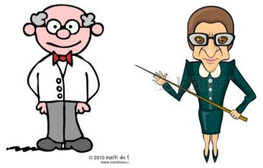
Project 1

3.3

Project 2

4.5

Judges



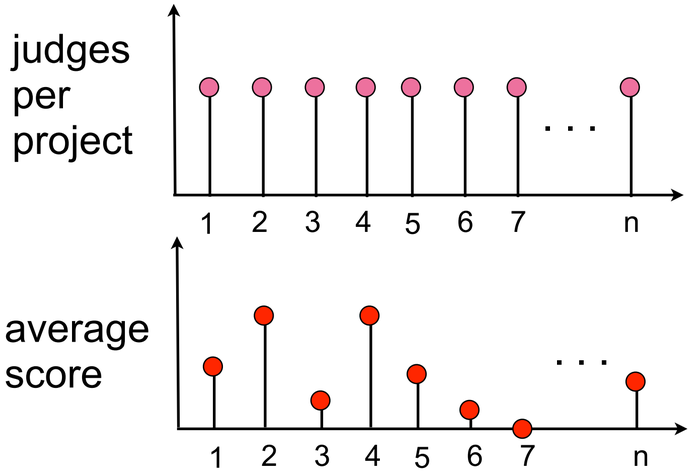
Project n-1

4.5

Project n

2.1

Uniform Judge Allocation



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.3 \pm .3$

Project 2

$4.5 \pm .3$

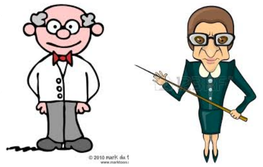
Project n-1

$4.5 \pm .3$

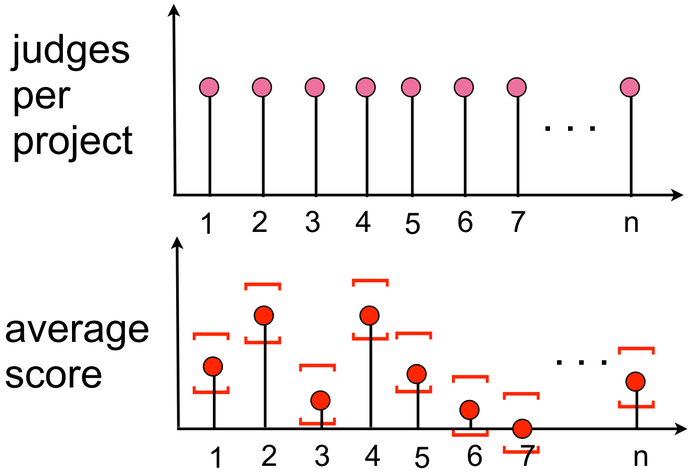
Project n

$2.1 \pm .3$

Judges



Uniform Judge Allocation



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.3 \pm .3$

Project 2

$4.5 \pm .3$

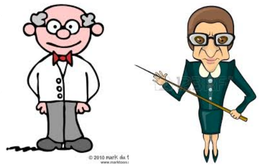
Project n-1

$4.5 \pm .3$

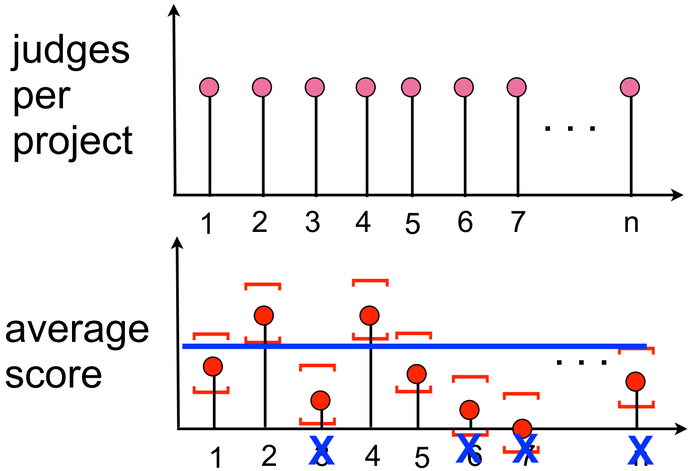
Project n

$2.1 \pm$

Judges



Uniform Judge Allocation



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.3 \pm .3$

Project 2

$4.5 \pm .3$

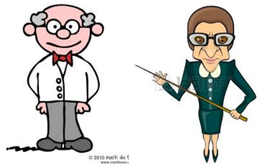
Project n-1

$4.5 \pm .3$

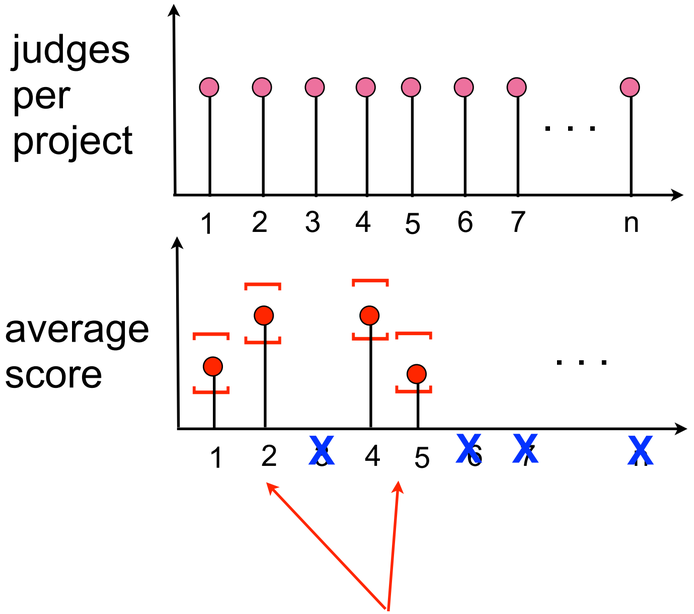
Project n

$2.1 \pm$

Judges



Uniform Judge Allocation



Statistically indistinguishable!

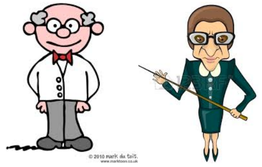
slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

Project 2

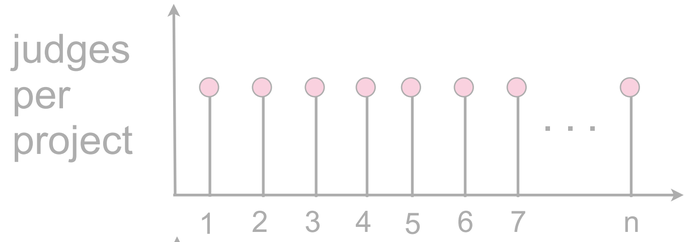
Judges

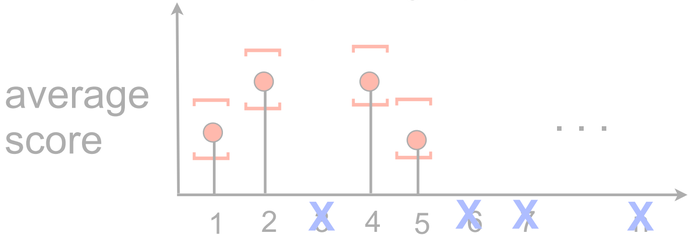


Project n-1

Project n

Uniform Judge Allocation





Adaptive Judge Allocation

slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

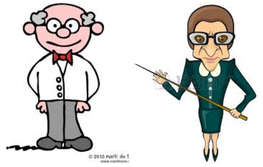
Project 1

2, 4

Project 2

5, 4

Judges



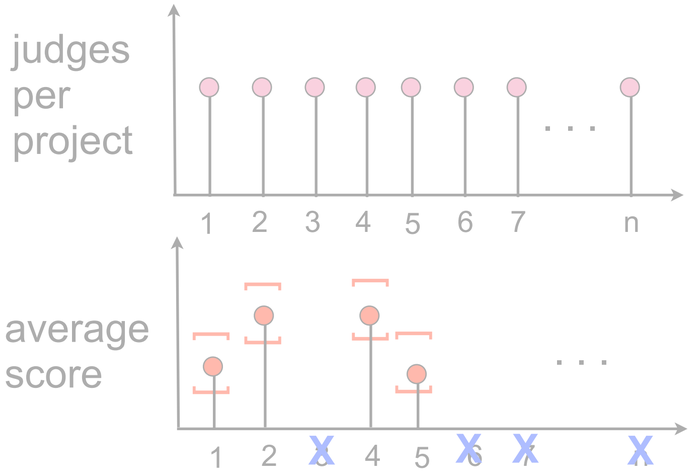
Project n-1

5, 4

Project n

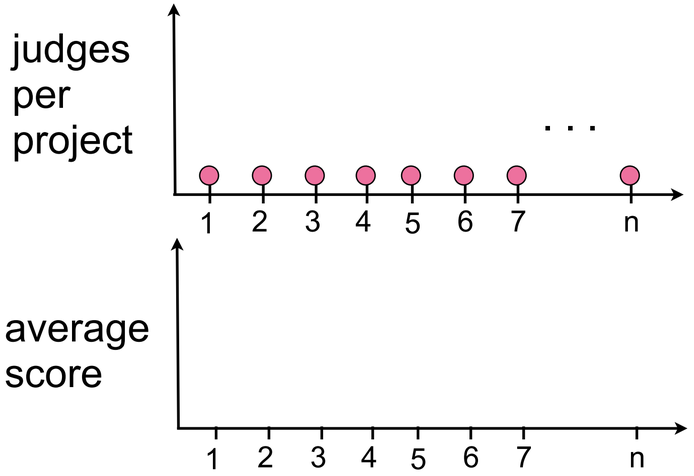
2, 1

Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

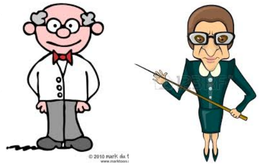
Project 1

3.1

Project 2

4.8

Judges



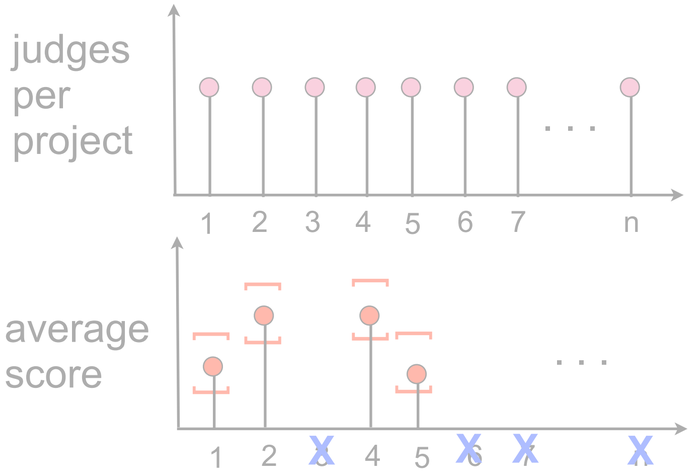
Project n-1

4.7

Project n

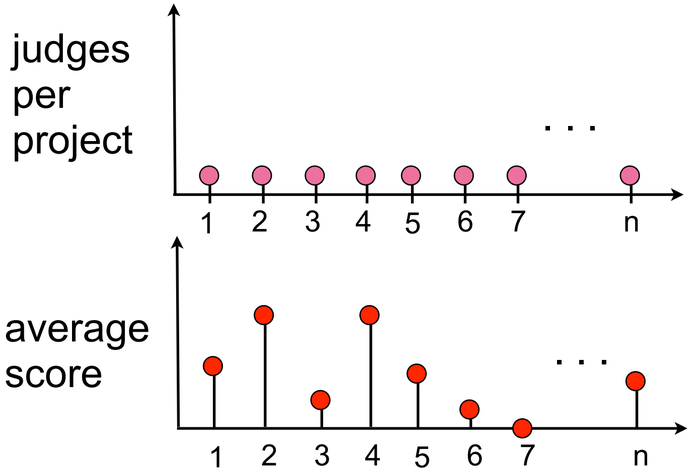
2.2

Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.1 \pm .5$

Project 2

$4.8 \pm .5$

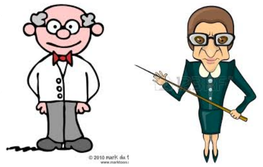
Project n-1

$4.7 \pm .5$

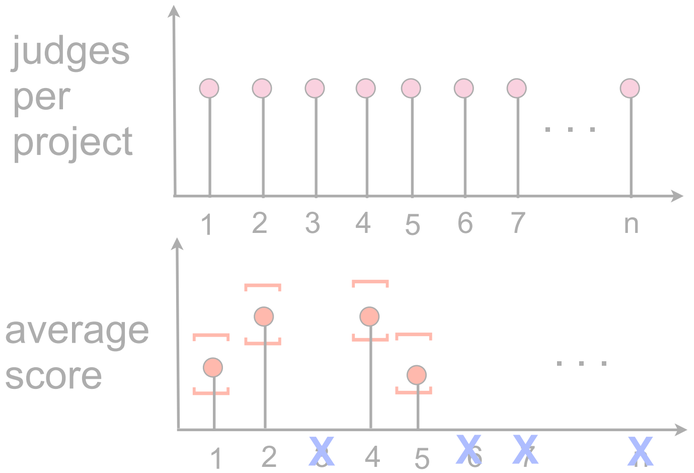
Project n

$2.2 \pm .5$

Judges

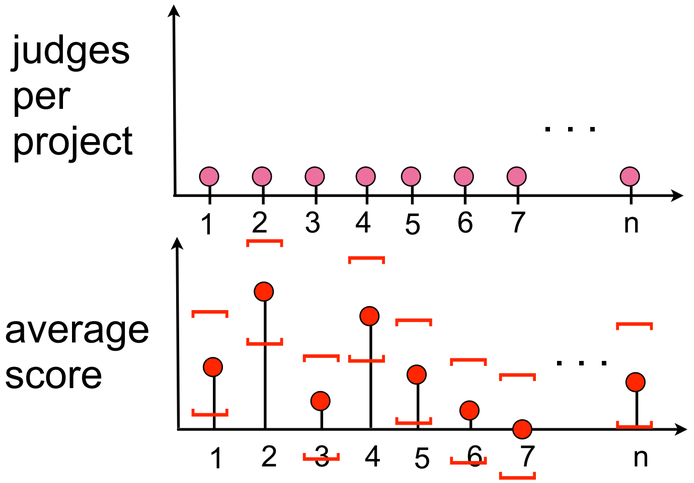


Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.1 \pm .5$

Project 2

$4.8 \pm .5$

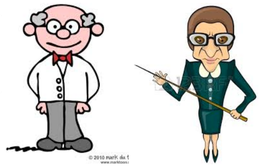
Project n-1

$4.7 \pm .5$

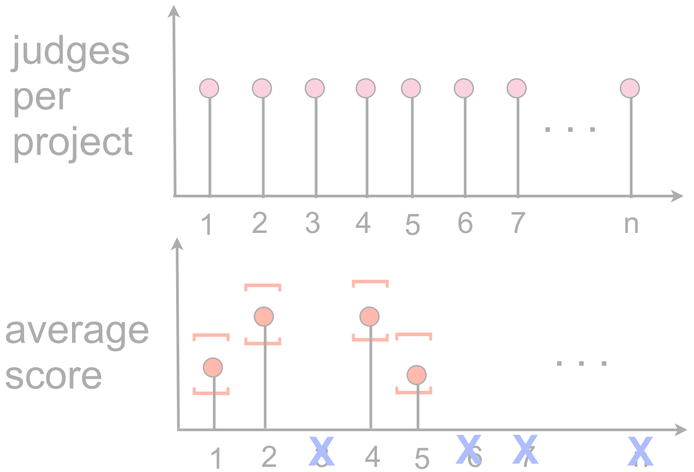
Project n

$2.2 \pm .5$

Judges

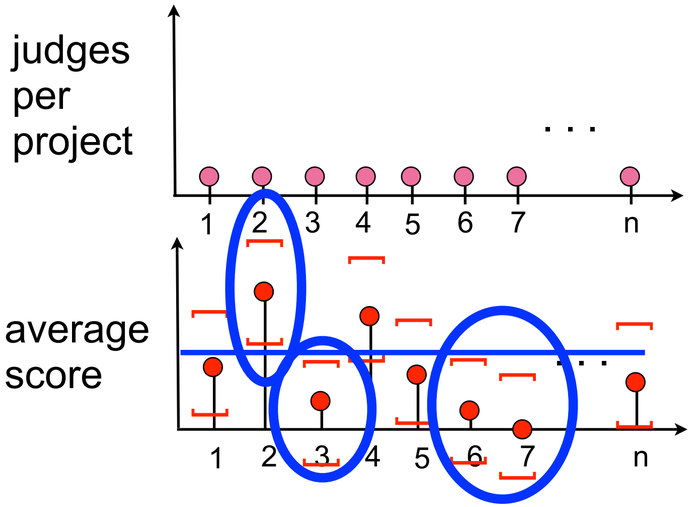


Uniform Judge Allocation



Adaptive Judge Allocation

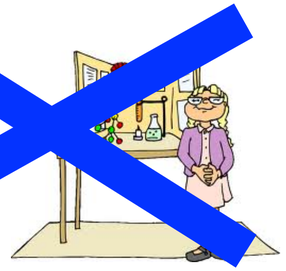
Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

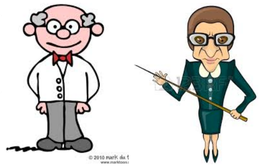
Project 1



Project 2

4.8 ± .5

Judges



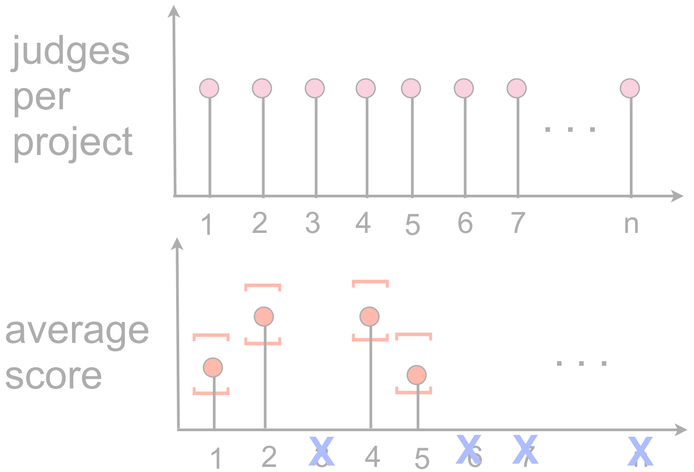
Project n-1

4.7 ± .5

Project n

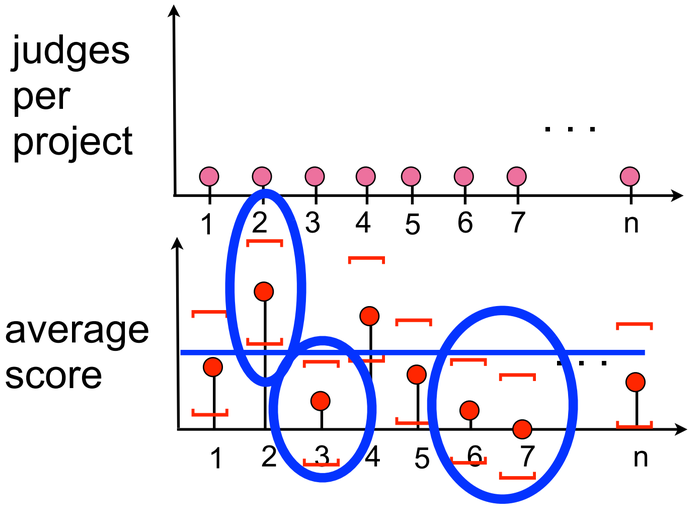
2.2 ±

Uniform Judge Allocation



Adaptive Judge Allocation

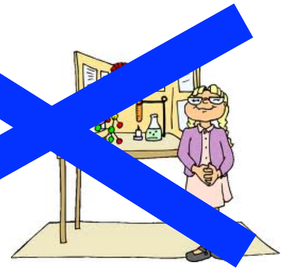
Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

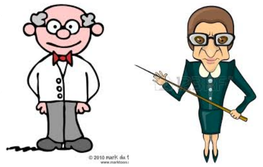
Project 1



Project 2

4.8 ± .5

Judges



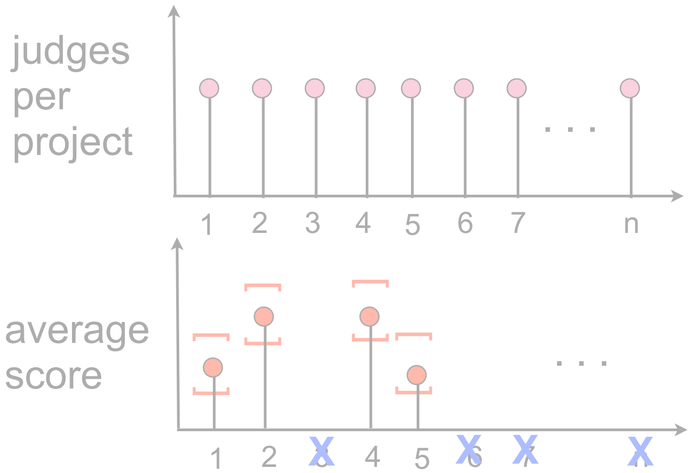
Project n-1

4.7 ± .5

Project n

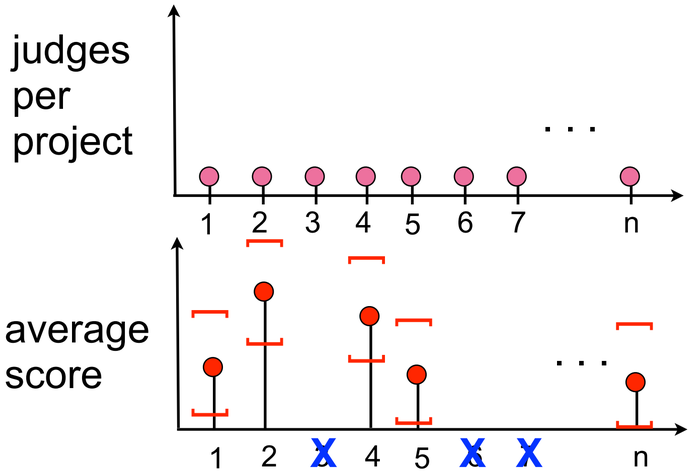
2.2 ±

Uniform Judge Allocation



Adaptive Judge Allocation

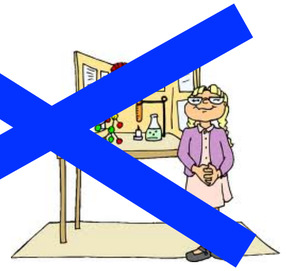
Proceed in rounds: Round 1



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

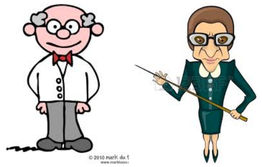
Project 1



Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$

Judges



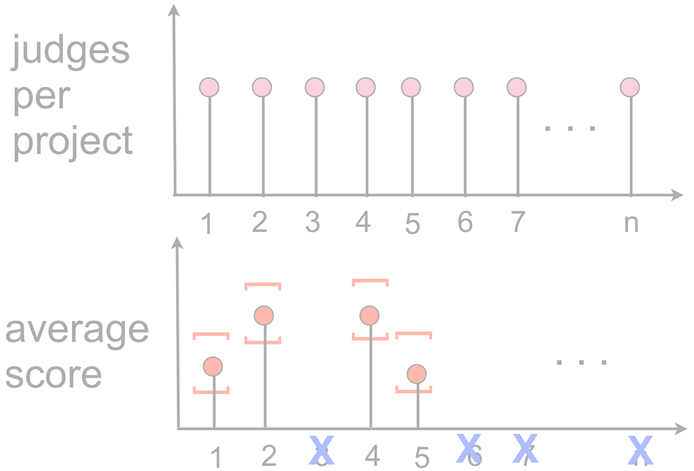
Project n-1

1. $4.7 \pm .5$
2. $4.8 \pm .3$

Project n

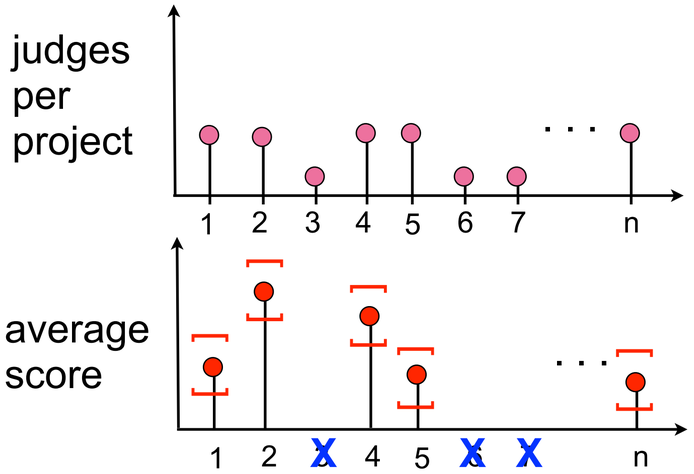
$2.2 \pm$

Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 2

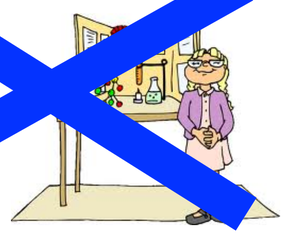


slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

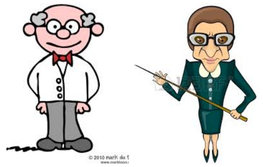
$3.1 \pm$ [illegible]



Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$

Judges



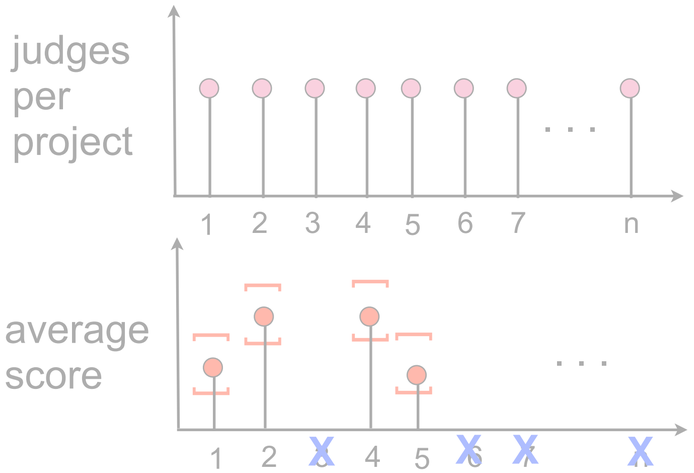
Project n-1

1. $4.7 \pm .5$
2. $4.8 \pm .3$

Project n

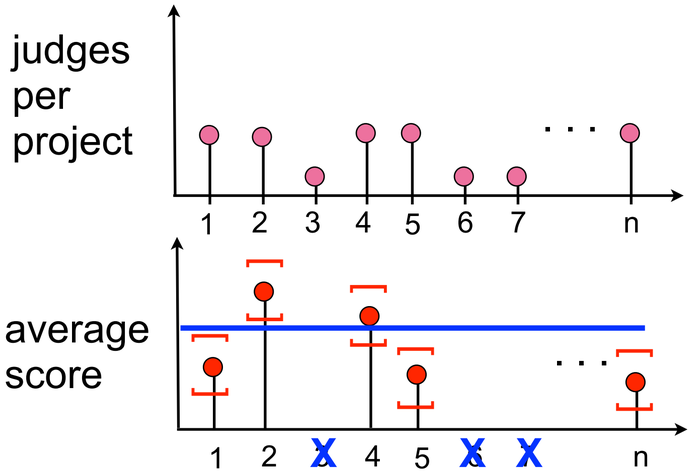
$2.2 \pm$ [illegible]

Uniform Judge Allocation



Adaptive Judge Allocation

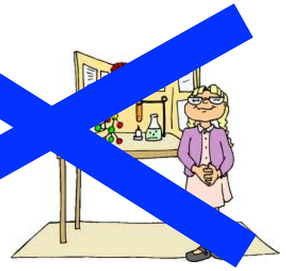
Proceed in rounds: Round 2



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

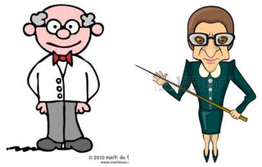
Project 1



Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$

Judges



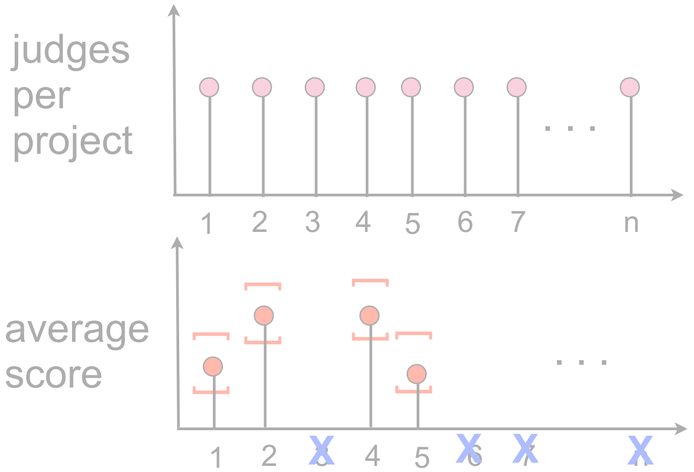
Project n-1

1. $4.7 \pm .5$
2. $4.8 \pm .3$

Project n

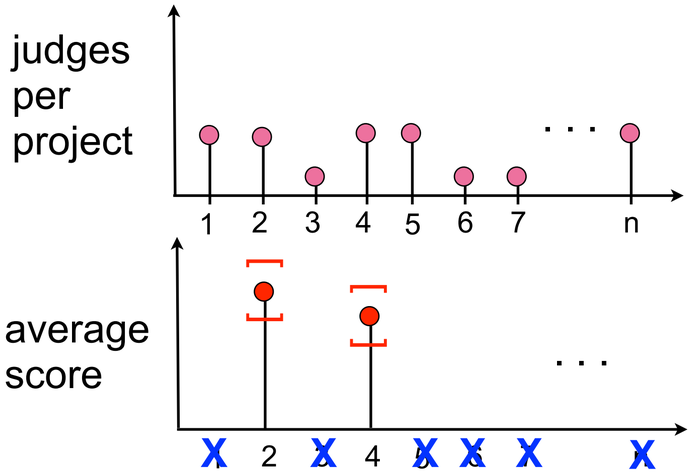
$2.2 \pm$

Uniform Judge Allocation



Adaptive Judge Allocation

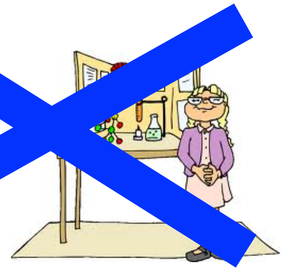
Proceed in rounds: Round 2



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

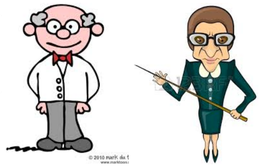
Project 1



Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$
3. $4.9 \pm .1$

Judges



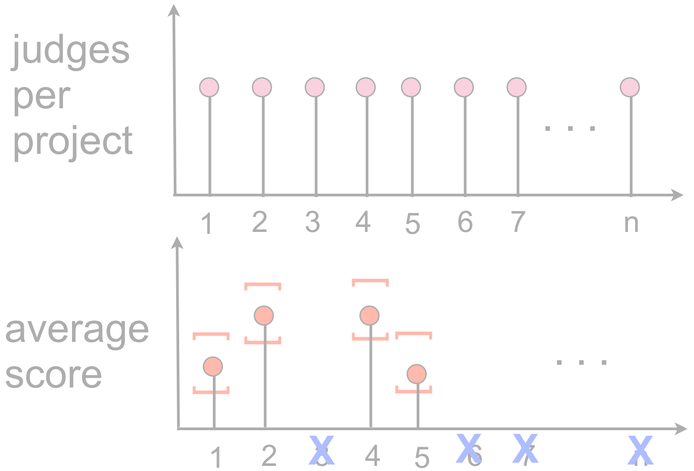
Project n-1

1. $4.7 \pm .5$
2. $4.8 \pm .3$
3. $4.7 \pm .1$

Project n

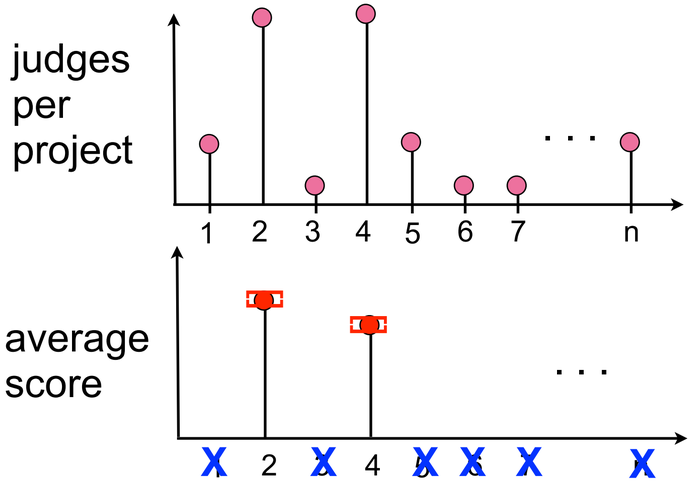
$2.2 \pm$

Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 3

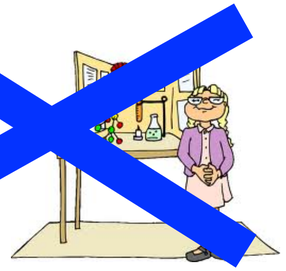


slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

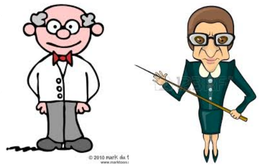
$3.1 \pm$ [illegible]



Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$
3. $4.9 \pm .1$

Judges



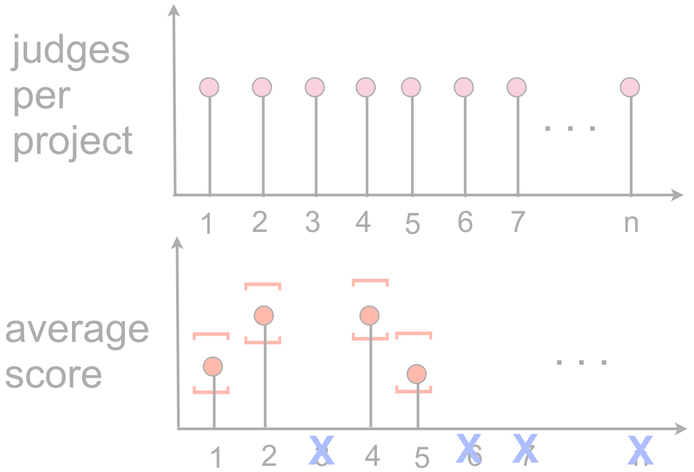
Project n-1

1. $4.7 \pm .5$
2. $4.8 \pm .3$
3. $4.7 \pm .1$

Project n

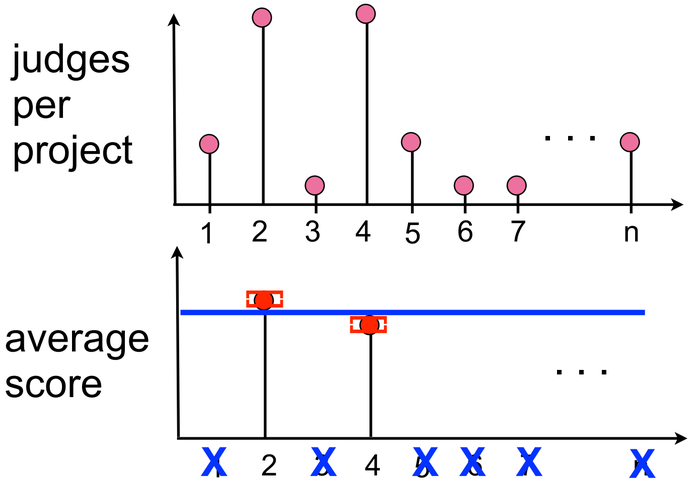
$2.2 \pm$ [illegible]

Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 3



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair

Project 1

$3.1 \pm$

Project 2

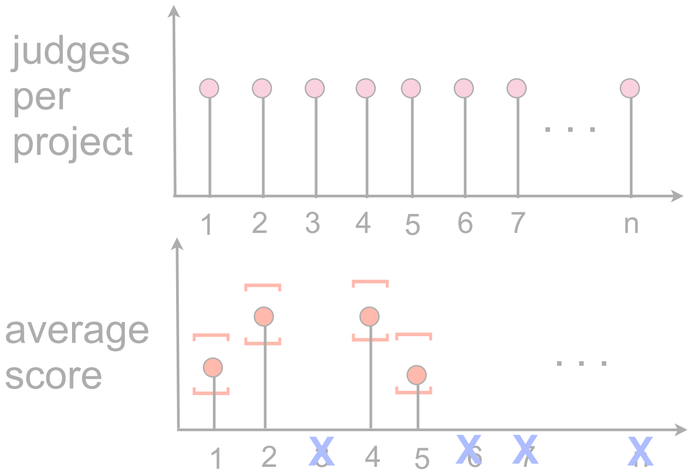
1. $4.8 \pm .5$
2. $4.7 \pm .3$
3. $4.9 \pm .1$

Judges

Project n

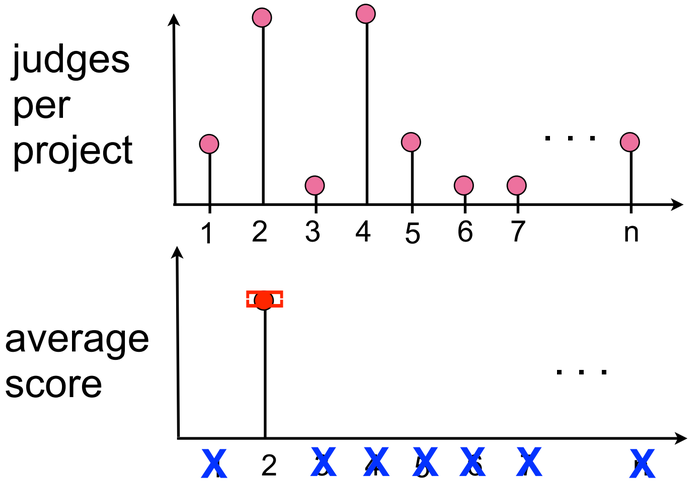
$2.2 \pm$

Uniform Judge Allocation



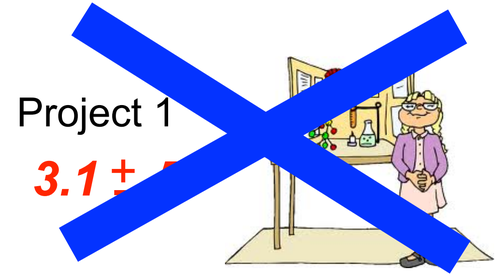
Adaptive Judge Allocation

Proceed in rounds: Round 3



slide credit: Kevin Jamieson

Choosing the winner of a Science Fair



Project 1

$3.1 \pm$ [illegible]

Project 2

1. $4.8 \pm .5$
2. $4.7 \pm .3$
3. $4.9 \pm .1$

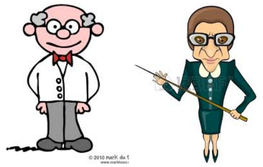
Proje[illegible]1

1. $4.7 \pm .5$
2. $4.8 \pm$ [illegible]
3. [illegible] $\pm .1$

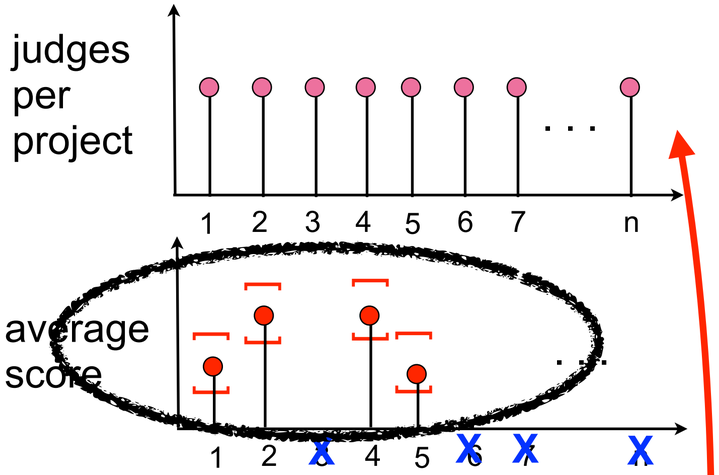
Project n

$2.2 \pm$ [illegible]

Judges

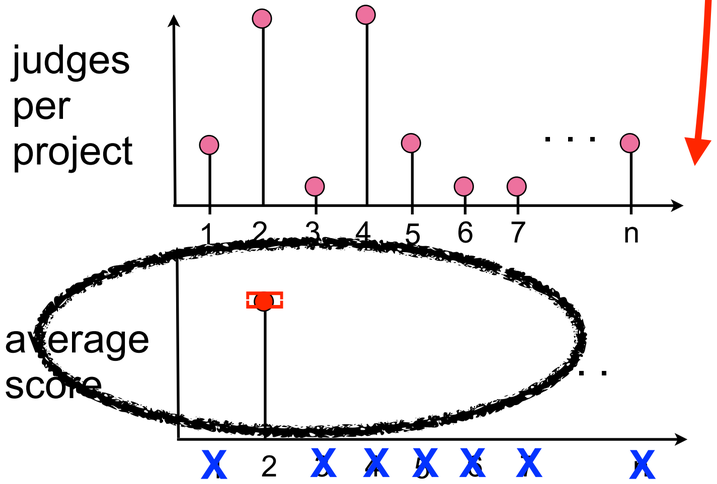


Uniform Judge Allocation



Adaptive Judge Allocation

Proceed in rounds: Round 3



Same number of total judgments

Stochastic Best Arm Identification

n = number of arms — science projects

T_i = number of times i*th* arm is pulled — times i*th* project is judged

$\ell_{i,j}$ = j*th* loss for i*th* arm drawn i.i.d. — j*th* judge's score of i*th* project

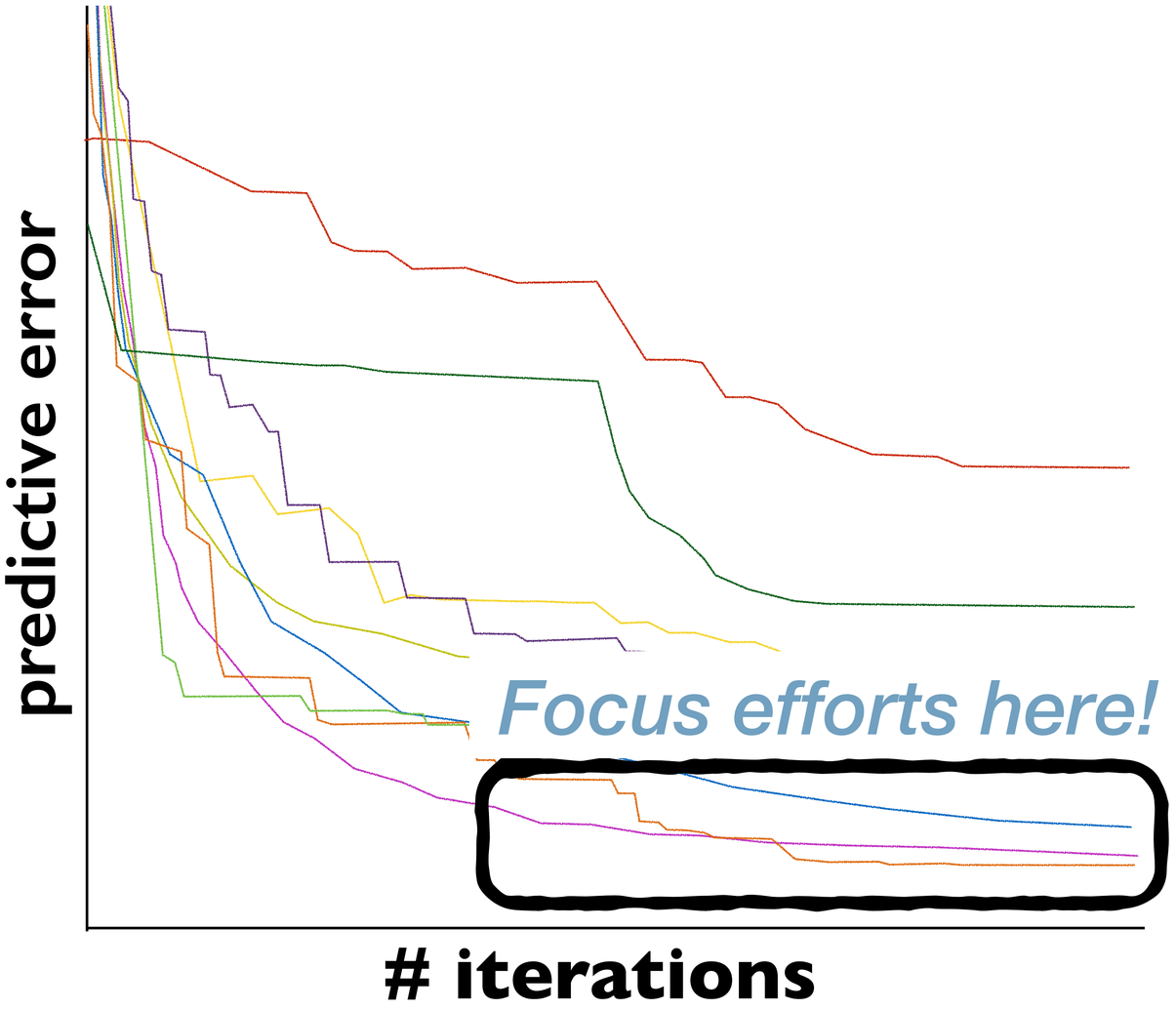
ν_i = expected loss for i*th* arm, $\mathop{\mathbb{E}}_{j} \ell_{i,j}$ — average over all judge scores for i*th* project

Goal: $\min \sum_i T_i$ subject to $\nu_{\hat{i}} = \min_i \nu_i$

"Minimize total judgments" subject to "finding the best project"

Early Stopping Revisited

Isn’t this just a stochastic best arm problem?



Stochastic Best Arm Identification

n = number of arms

distinct models evaluated

T_i = number of times i*th* arm is pulled

training iterations for i*th* model

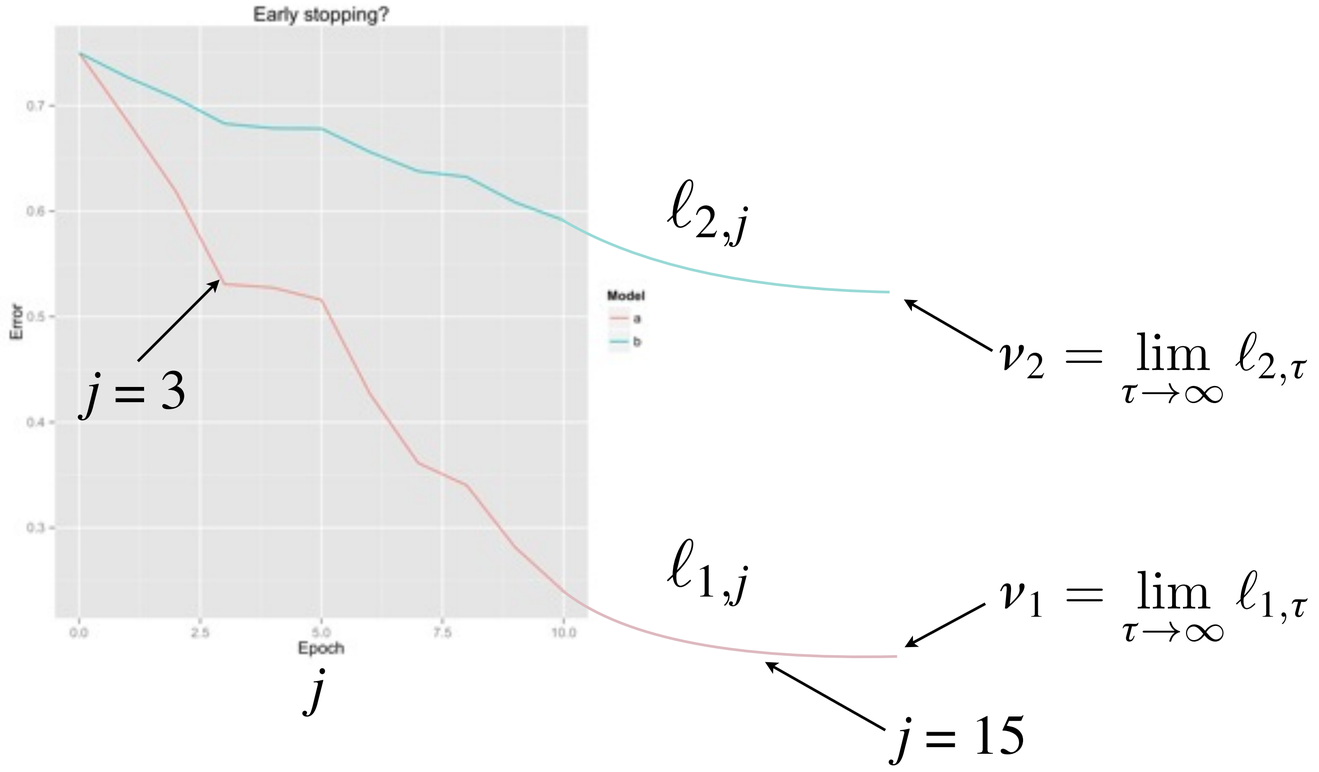
$\ell_{i,j}$ = j*th* loss for i*th* arm drawn i.i.d.

validation error for i*th* model after j*th* iteration

ν_i = expected loss for i*th* arm, $\mathop{\mathbb{E}}_{j} \ell_{i,j}$

average of validation errors for i*th* arm

Goal: $\min \sum_i T_i \quad \text{subject to} \quad \nu_{\hat{i}} = \min_i \nu_i$



Losses are NOT sampled i.i.d.

- i.e., $\mathbb{E}[\ell_{1,3}] \neq \mathbb{E}[\ell_{1,15}]$
- Rather, they usually decrease (non-monotonically)

We should be reasoning about loss at convergence

NON-Stochastic Best Arm Identification

n = number of arms

T_i = number of times i*th* arm is pulled

$\ell_{i,j}$ = j*th* loss for i*th* arm ~~drawn i.i.d.~~

ν_i = expected loss for i*th* arm, $\mathop{\mathbb{E}}_{j} \ell_{i,j}$

NON-Stochastic Best Arm Identification

n = number of arms — distinct models evaluated

T_i = number of times i*th* arm is pulled — training iterations for i*th* model

$\ell_{i,j}$ = j*th* loss for i*th* arm ~~drawn i.i.d.~~ — validation error for i*th* model after j*th* iteration

$\nu_i = \lim_{\tau \to \infty} \ell_{i,\tau}$ — final validation error for i*th* arm

Goal: $\min \sum_i T_i$ subject to $\nu_{\hat{i}} = \min_i \nu_i$

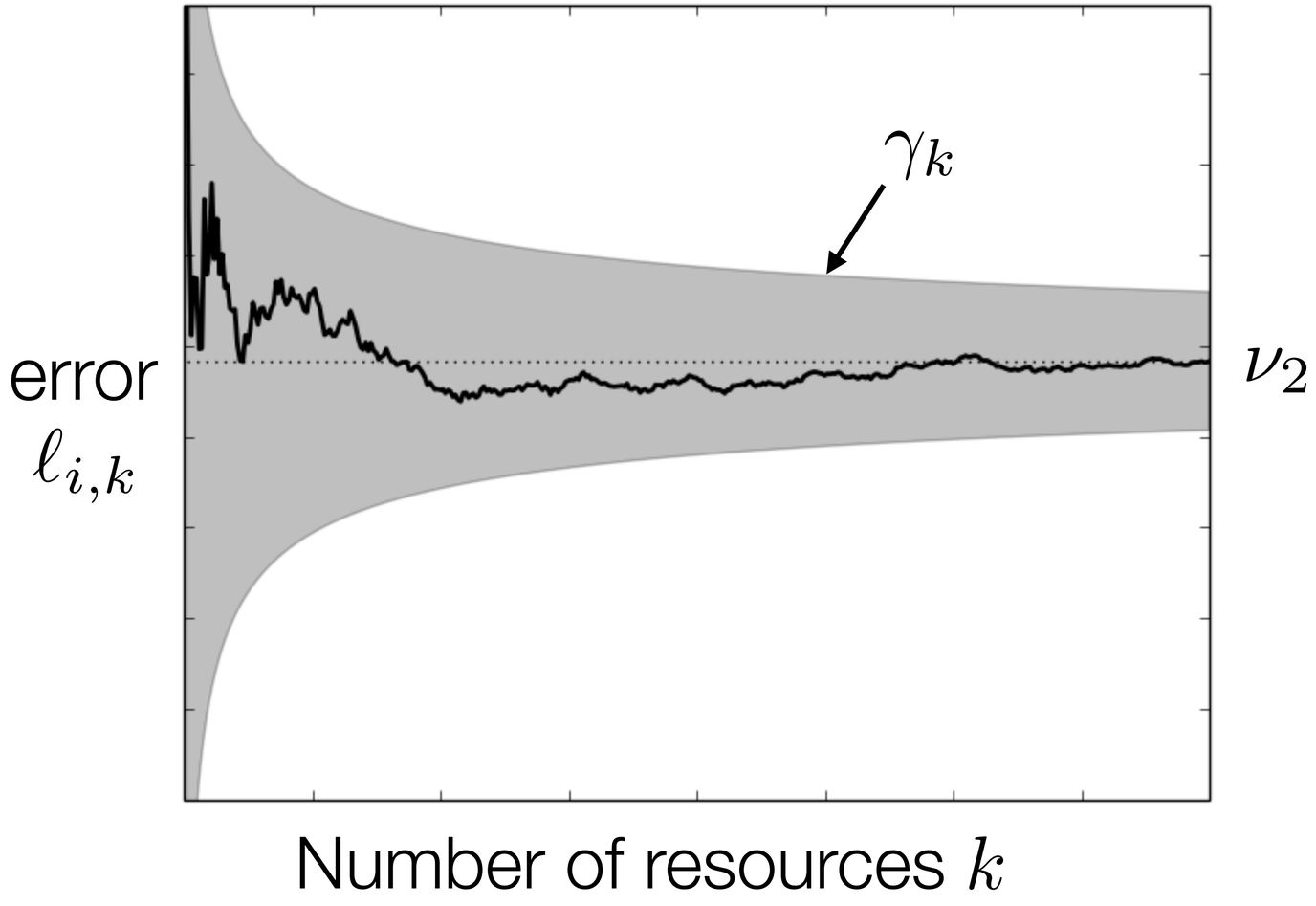
"Minimize total training iterations" subject to "finding best model"

$$\lim_{k \to \infty} \ell_{i,k} = \nu_i$$

$$|\ell_{i,k} - \nu_i| \leq \gamma_k \qquad \forall i \in [n], k \geq 1$$

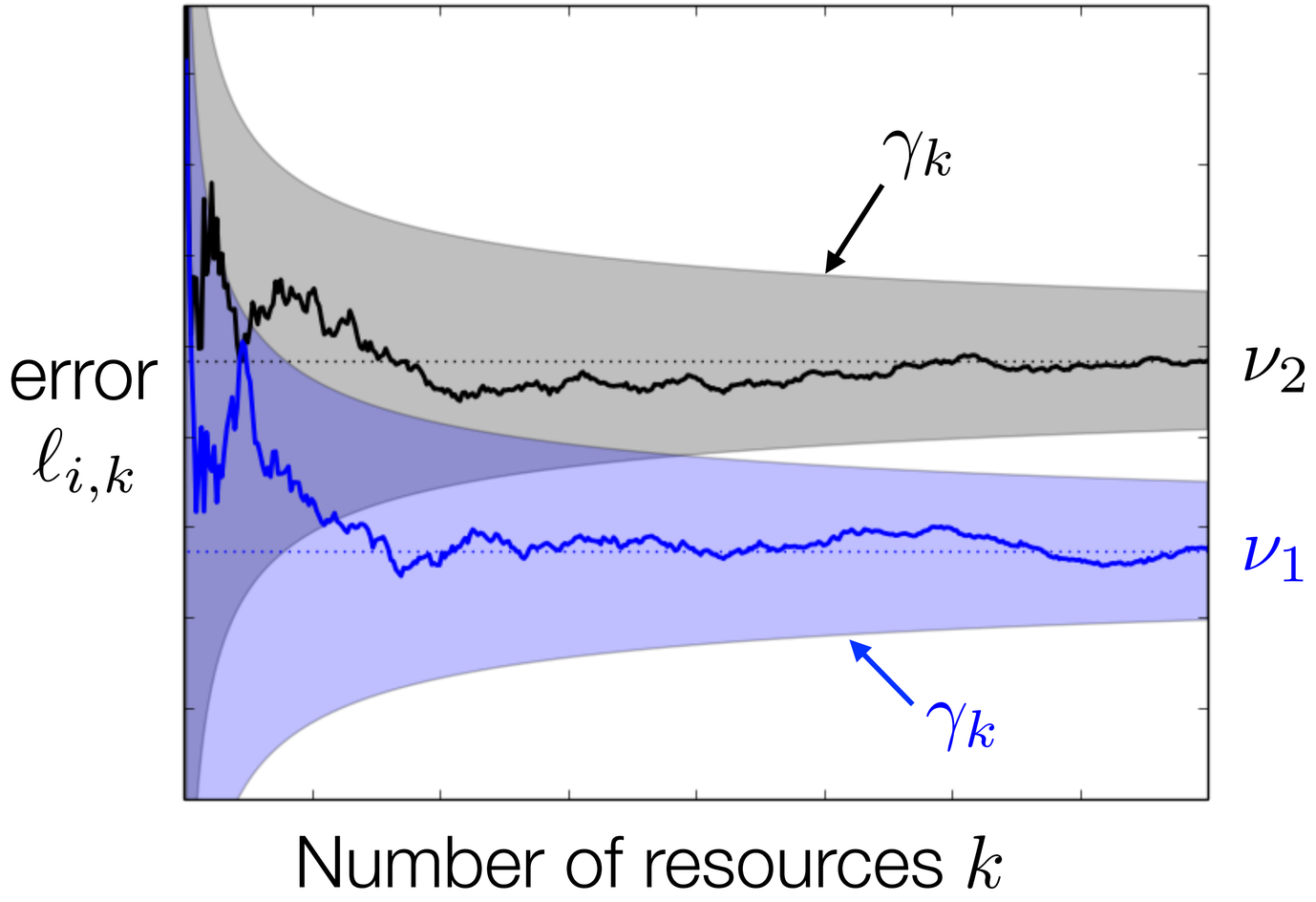
$$\lim_{k \to \infty} \ell_{i,k} = \nu_i$$

$$|\ell_{i,k} - \nu_i| \leq \gamma_k \qquad \forall i \in [n], k \geq 1$$



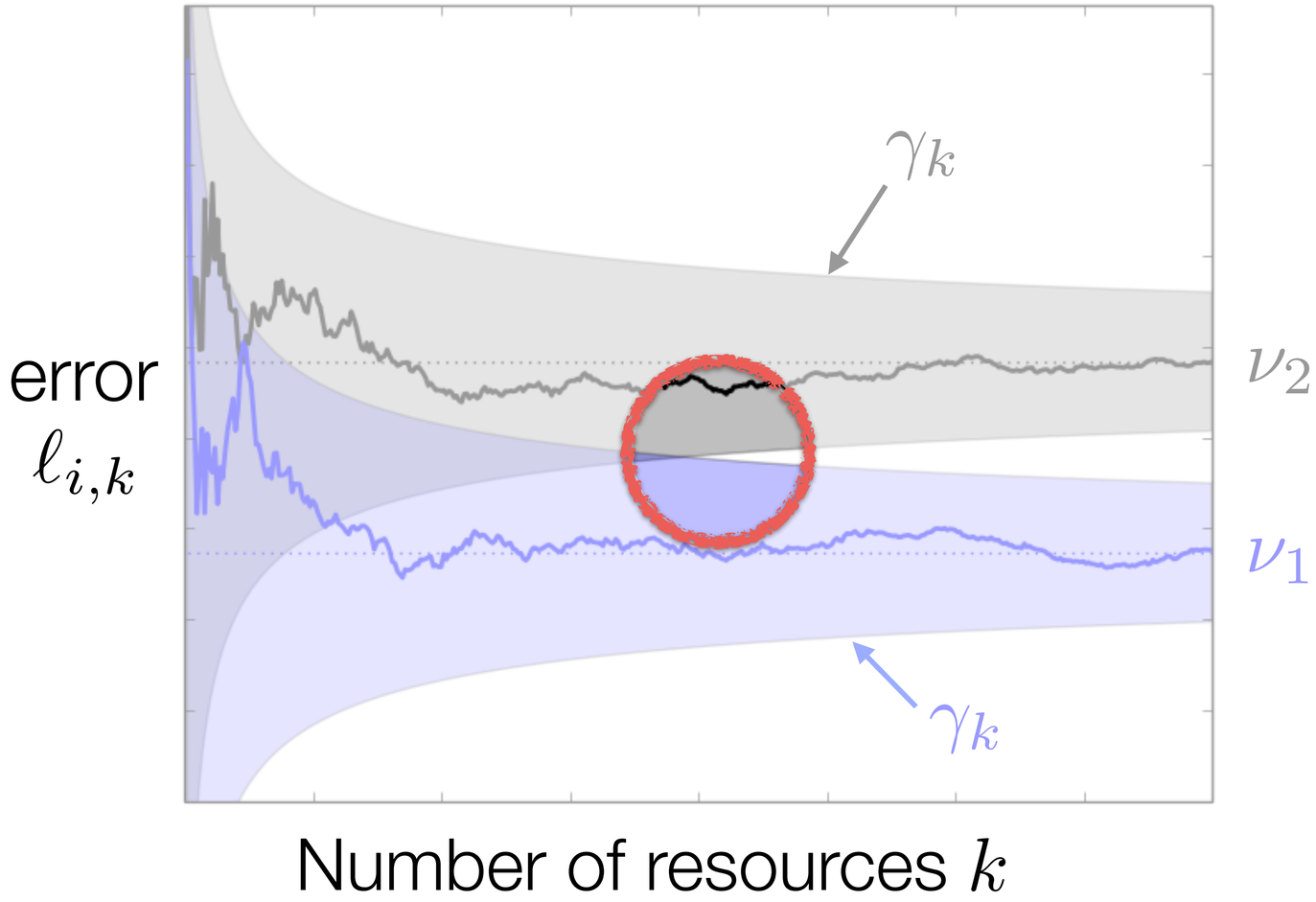
$$\lim_{k\to\infty} \ell_{i,k} = \nu_i$$

$$|\ell_{i,k} - \nu_i| \leq \gamma_k \qquad \forall i \in [n], k \geq 1$$



$$\lim_{k\to\infty} \ell_{i,k} = \nu_i$$

$$|\ell_{i,k} - \nu_i| \leq \gamma_k \qquad \forall i \in [n], k \geq 1$$



Take-aways

Baselines: grid search, random search — why are these expensive?

Hyperband key ideas:

- Use random search to find configurations
- BUT, reduce evaluation cost with downsampling / successive ‘halving’
- Try multiple ‘brackets’ of successive halving to downsample safely

Connections between Hyperband/successive halving and multi-armed bandits

Additional Resources

- Hyperband: https://jmlr.org/papers/volume18/16-558/16-558.pdf
- Blog: https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/