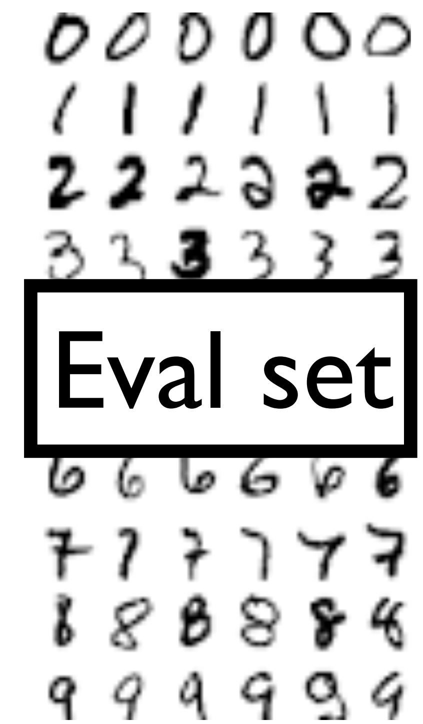
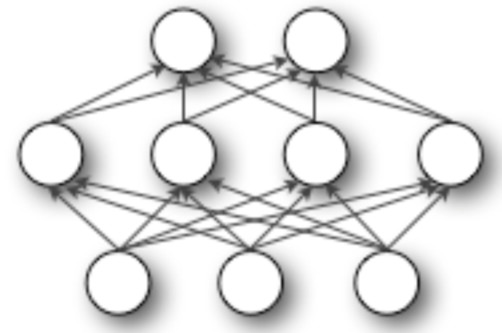
A Novel Bandit-Based Approach to Hyperparameter Optimization

Ameet Talwalkar (UCLA)
December 10, 2016

Collaborators: Lisha Li (UCLA), Kevin Jamieson (UC Berkeley), Giulia DeSalvo (NYU), Afshin Rostamizadeh (Google)


Training set

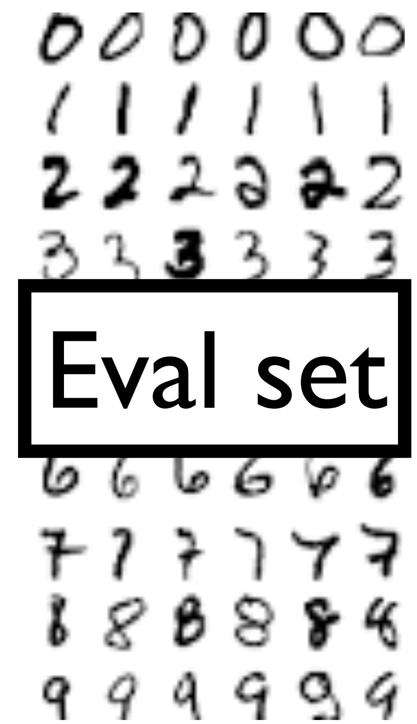

Eval set



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

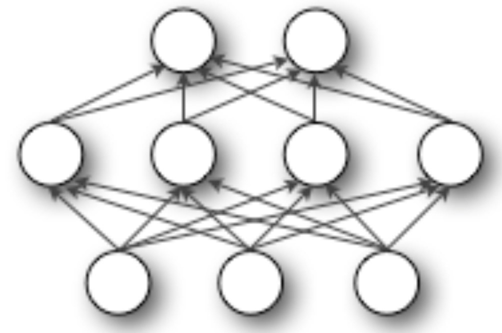


hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

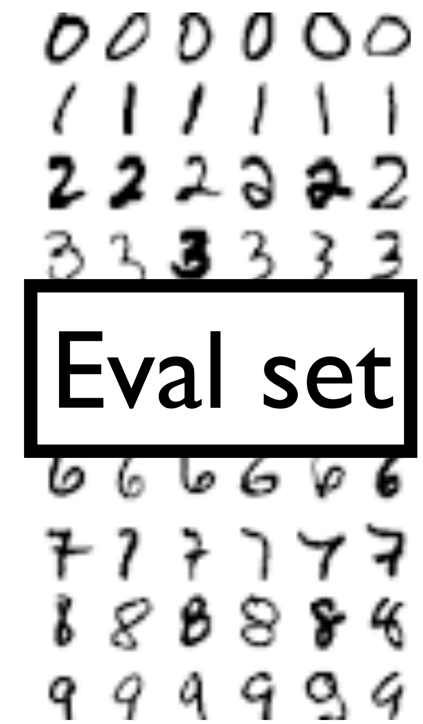
hidden nodes $N_{hid} \in [10^{1}, 10^{3}]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

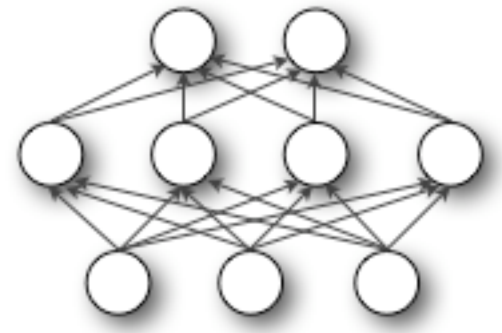
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

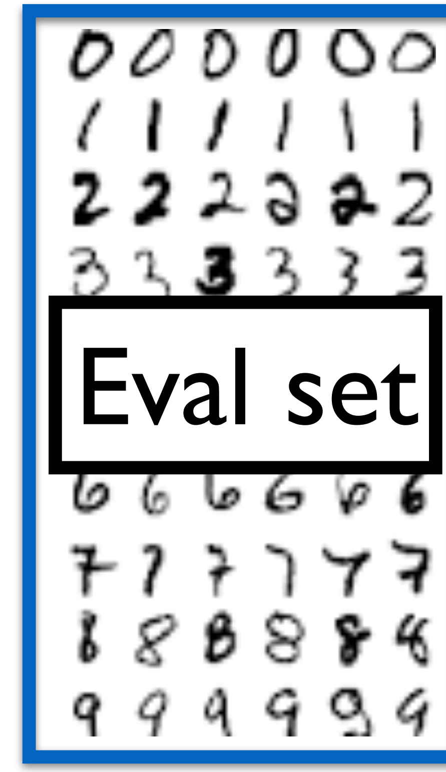
$\#$ hidden nodes $N_{hid} \in [10^{1}, 10^{3}]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

Eval-loss

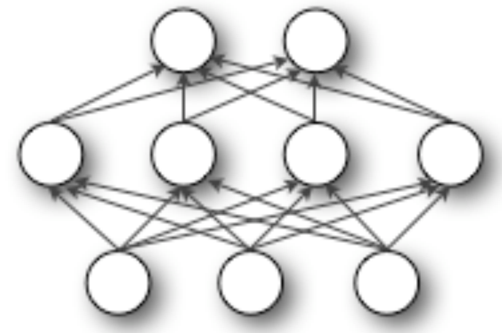
$\boxed{0.0577}$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

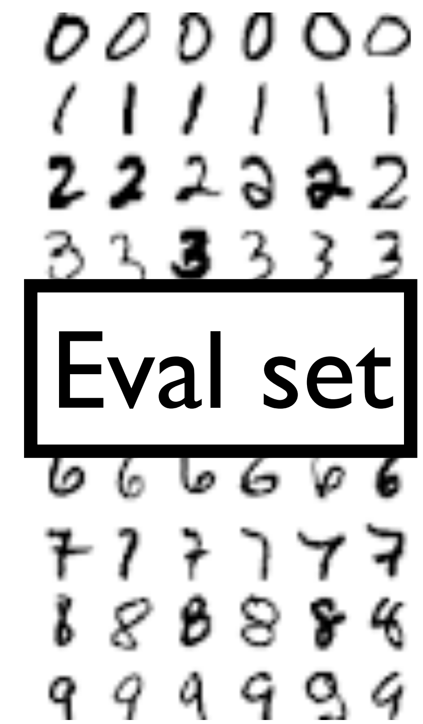
hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



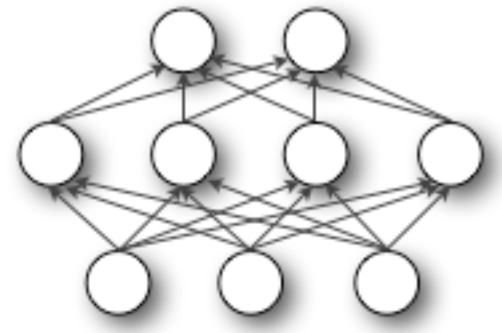
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

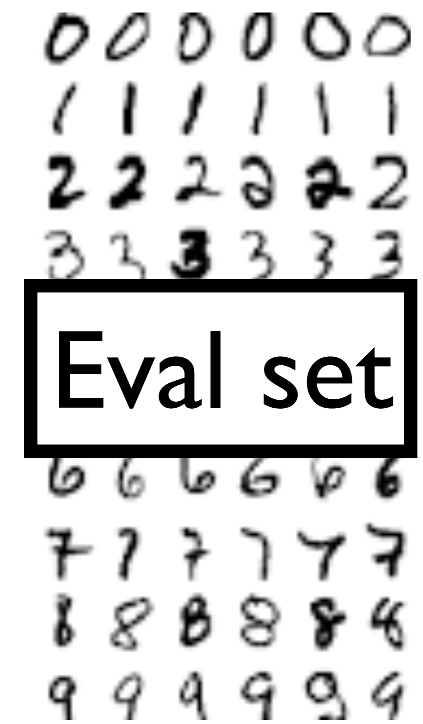
hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

How do we efficiently choose good hyperparameters?

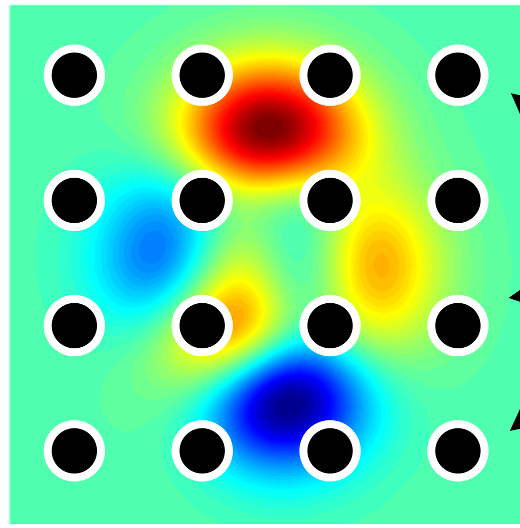
Existing Methods

Hyperband Algorithm

- Experimental Results
- Theory (Briefly)

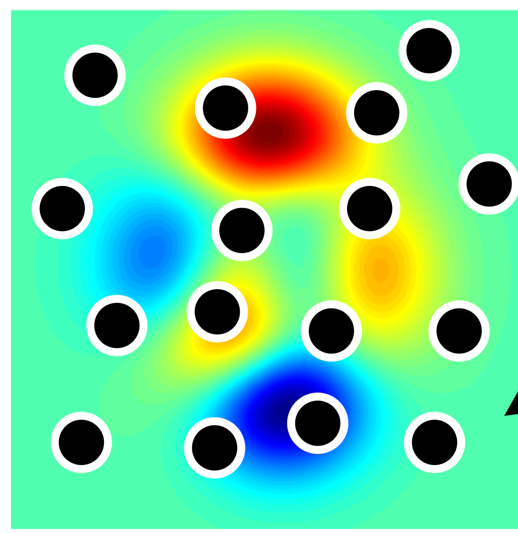
How do we choose hyperparameters?

Grid search:



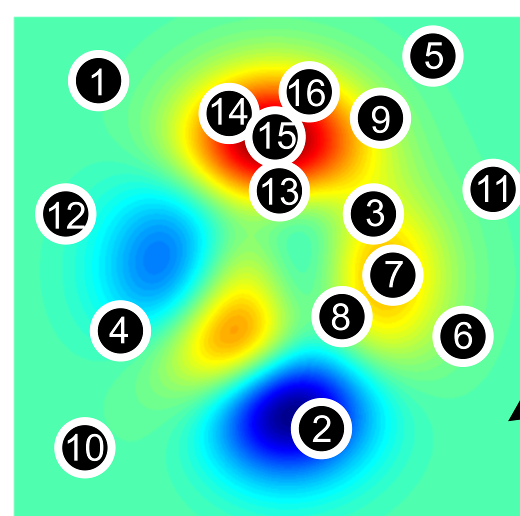
Hyperparameters on 2d uniform grid

Random search:



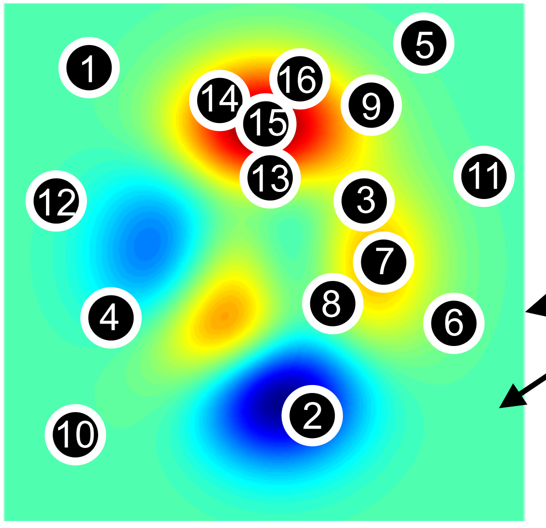
Hyperparameters randomly chosen

Bayesian Optimization:



Hyperparameters *adaptively* chosen

Bayesian Optimization attempt to optimize configuration selection



Hyperparameters *adaptively* chosen

Method is popular for hyperparameter tuning

However…

Sequential (i.e. difficult to parallelize across nodes)

Requires its own hyperparameters

Not guaranteed to find a good setting

Random Search does not suffer any of these downsides but it is often less efficient in number of evaluations

Goal: make random search faster

Intuition: Adaptive Resource Allocation

Assume:

- d hyperparameters to tune
- N total evaluations of configurations

Case 1: $N = O(2^d)$

- We can hope to cover the space
- Black-box optimization is a reasonable option

Case 2: $N = O(d)$

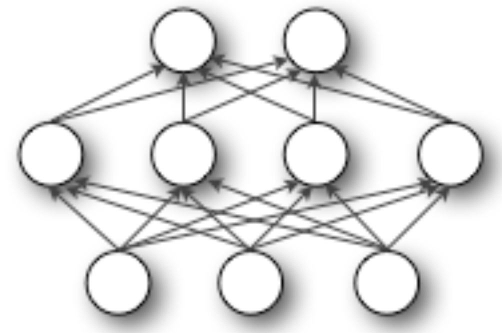
- Hard to cover the space, let alone be adaptive
- Increasingly common regime, e.g., deep learning

Idea: Use adaptive resource allocation in Case 2 to drastically increase # evaluations using same budget!

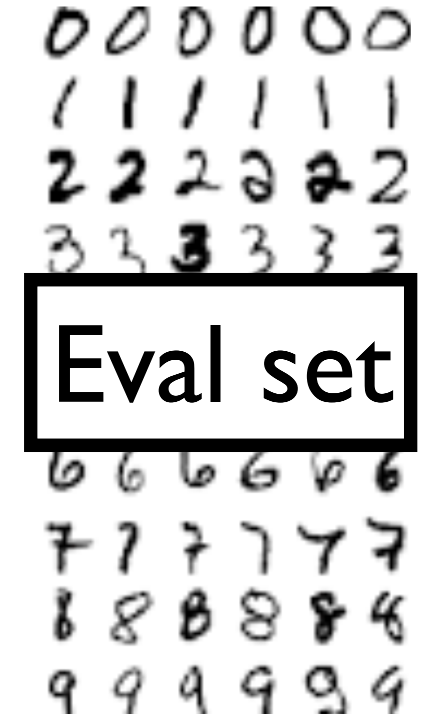
Existing Methods

Hyperband Algorithm

- Experimental Results
- Theory (Briefly)

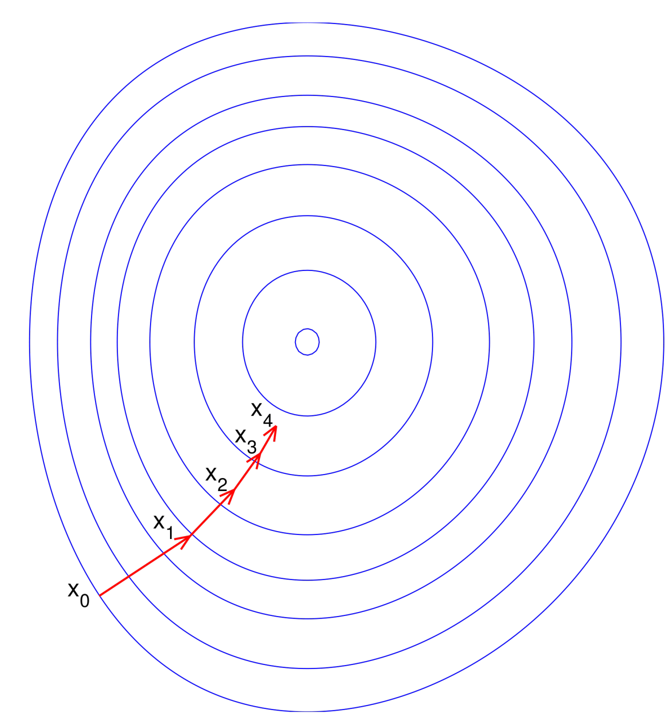


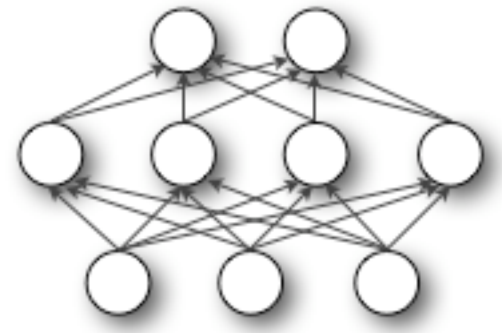
$N_{in} = 784$



Assume we’re using an **iterative learning algorithm**

- Gradient descent
- Newton’s method
- Block coordinate descent
- Decision Trees
- ALS
- …

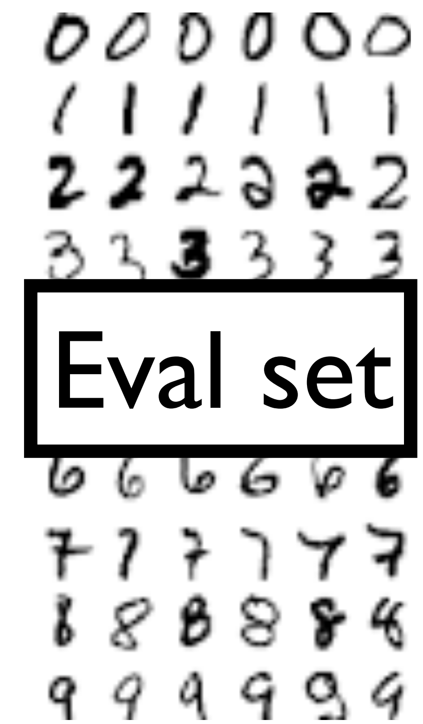




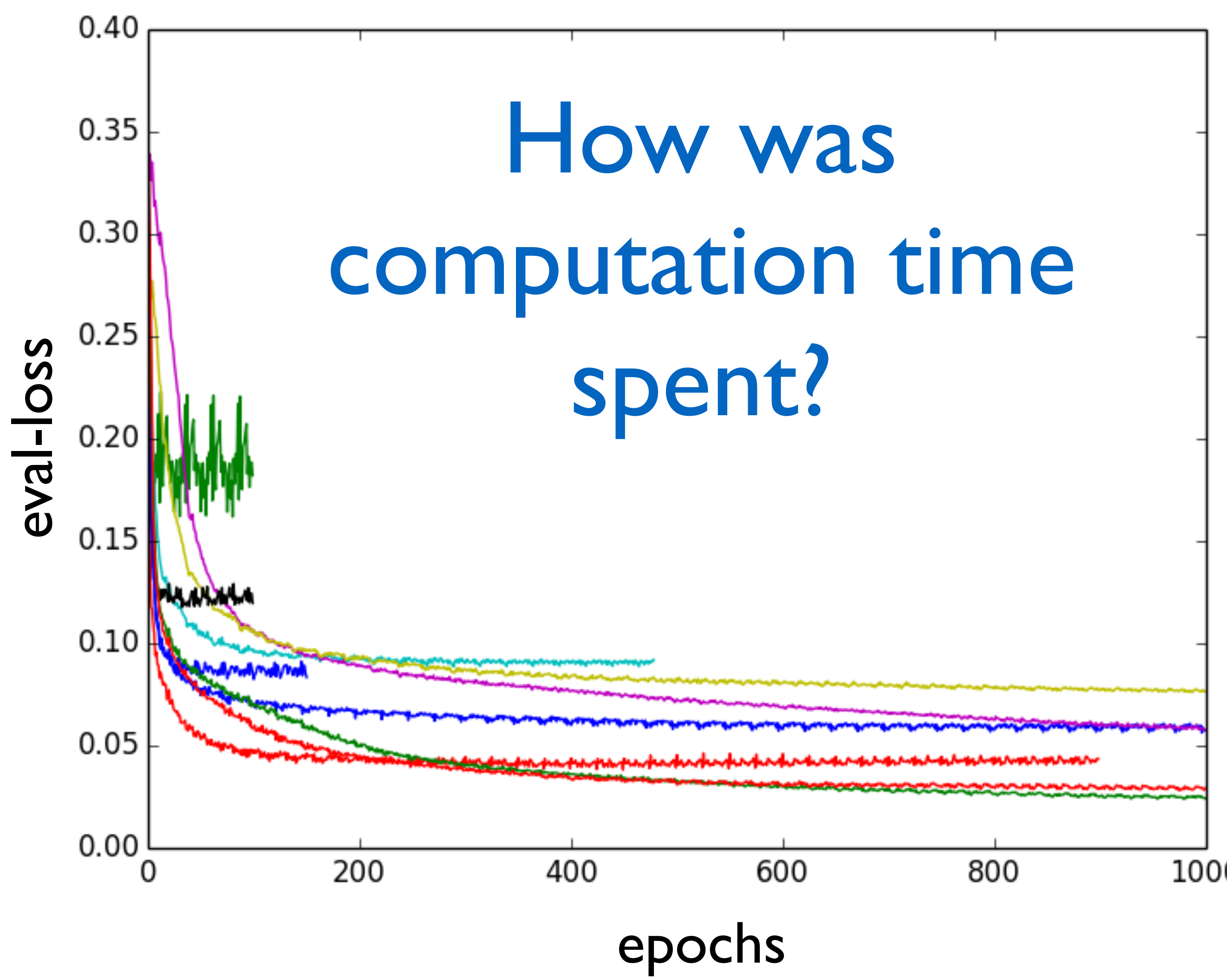
$N_{out} = 10$

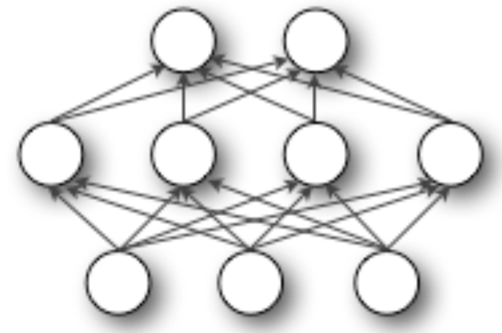
N_{hid}

$N_{in} = 784$



Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

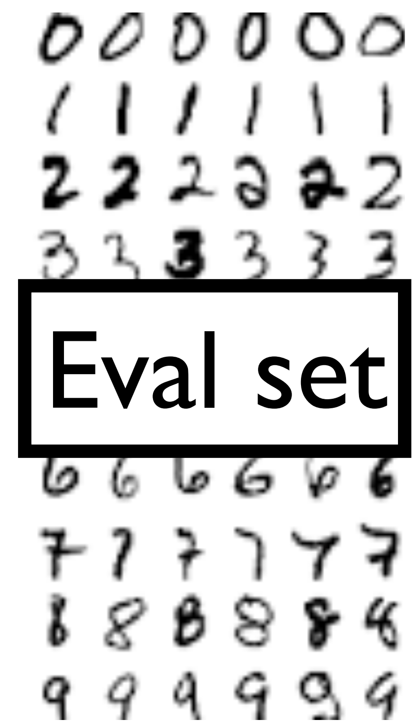




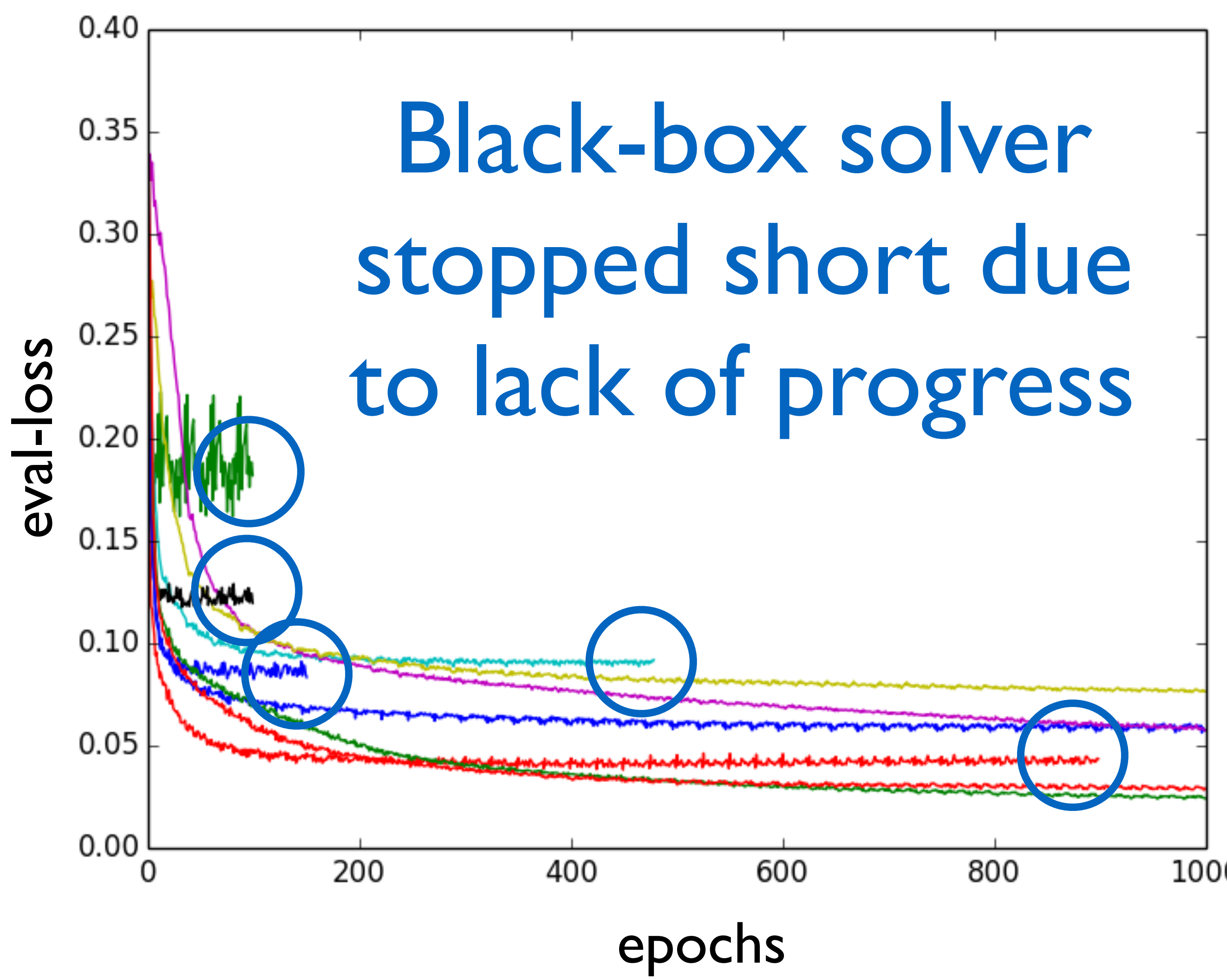
$N_{out} = 10$

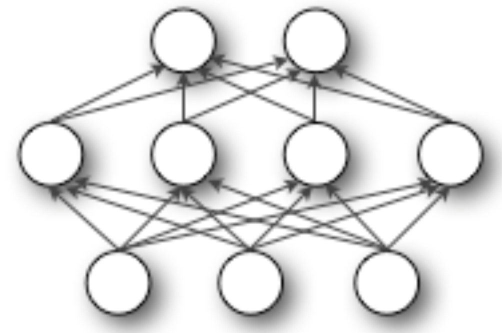
N_{hid}

$N_{in} = 784$



Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

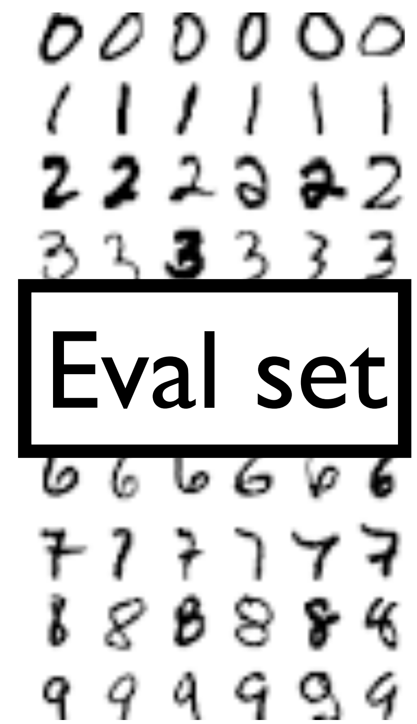




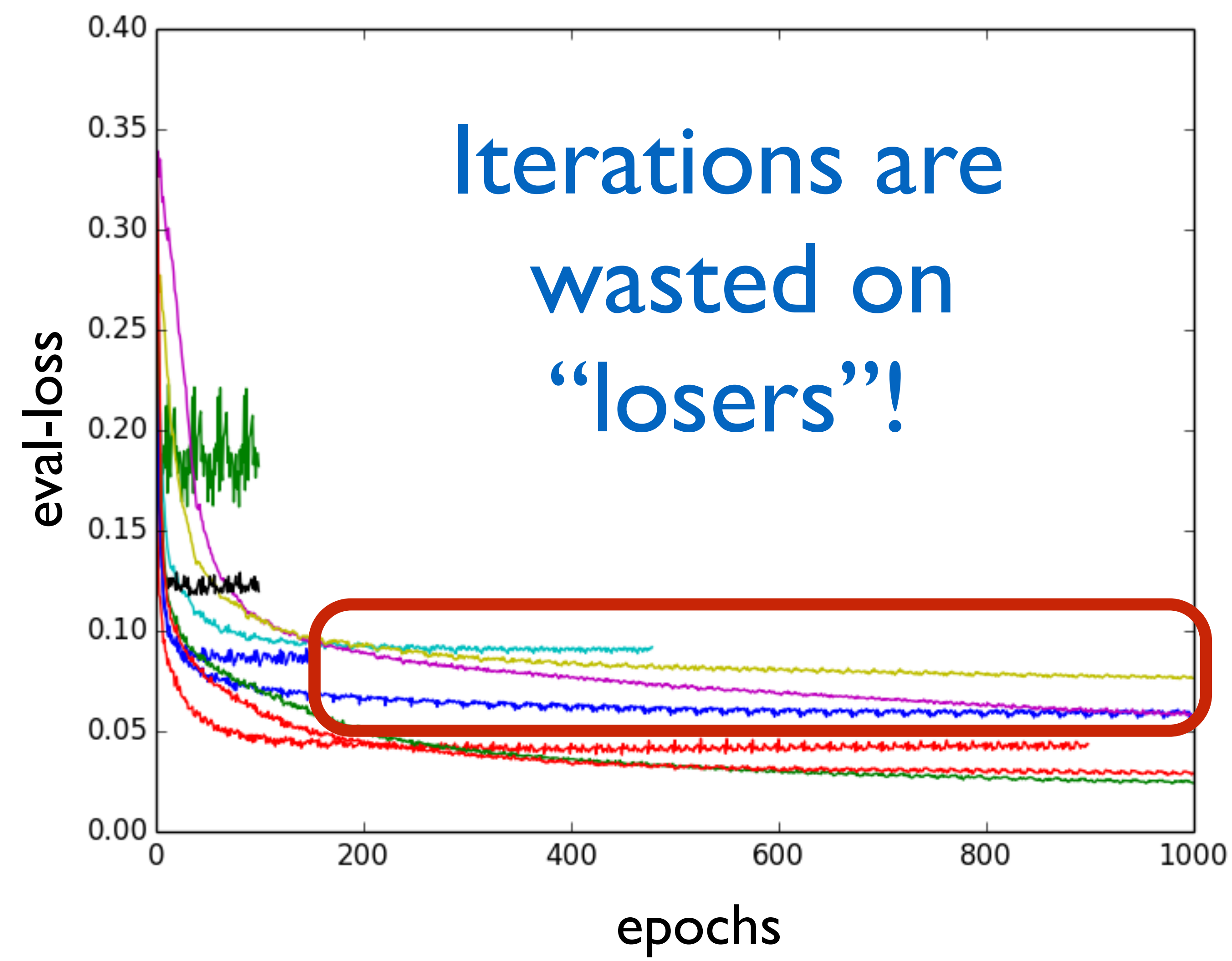
$N_{out} = 10$

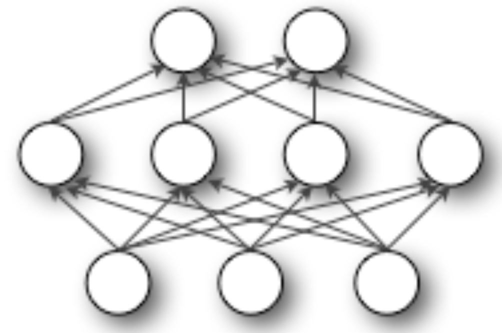
N_{hid}

$N_{in} = 784$



Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

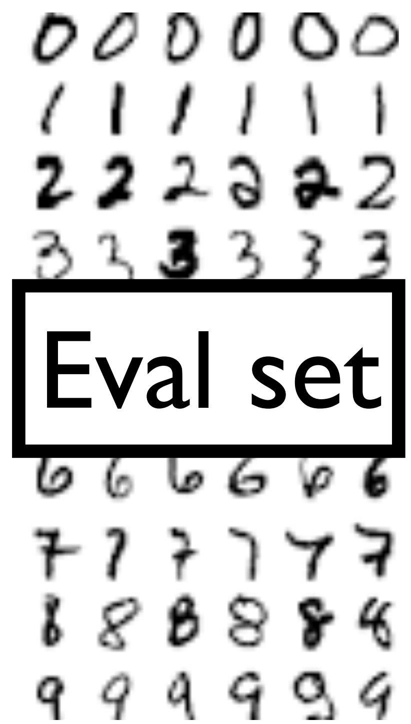




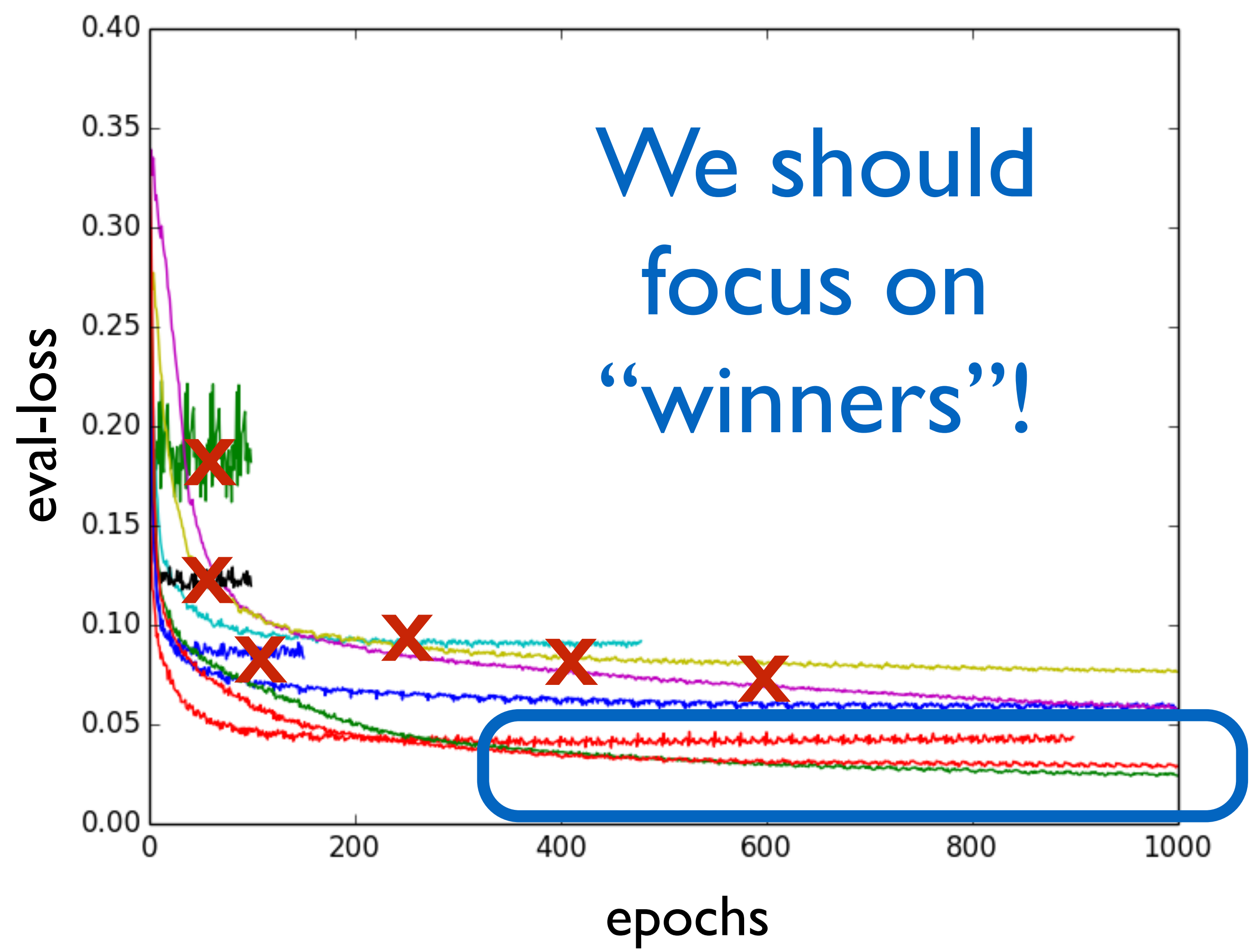
$N_{out} = 10$

N_{hid}

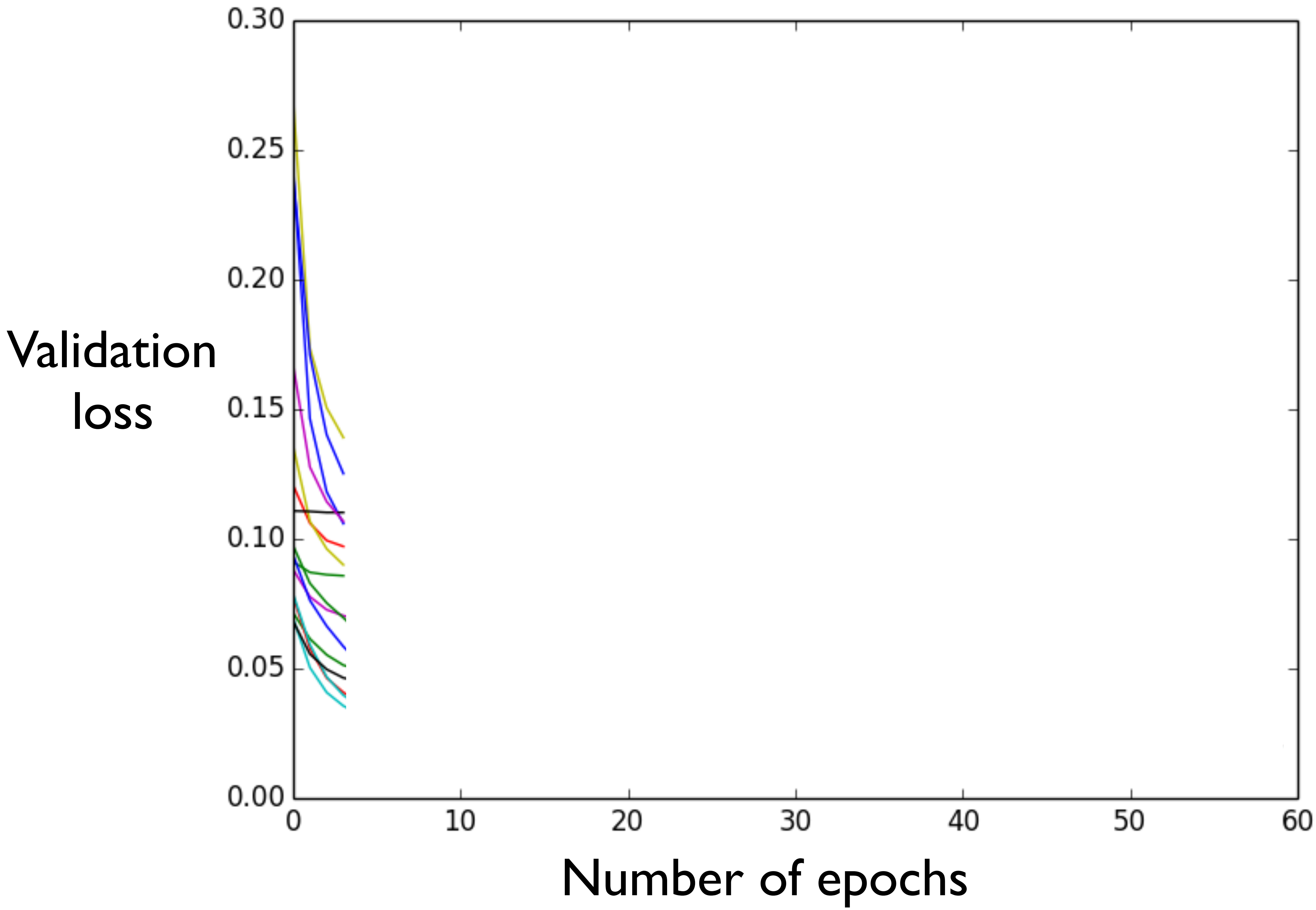
$N_{in} = 784$



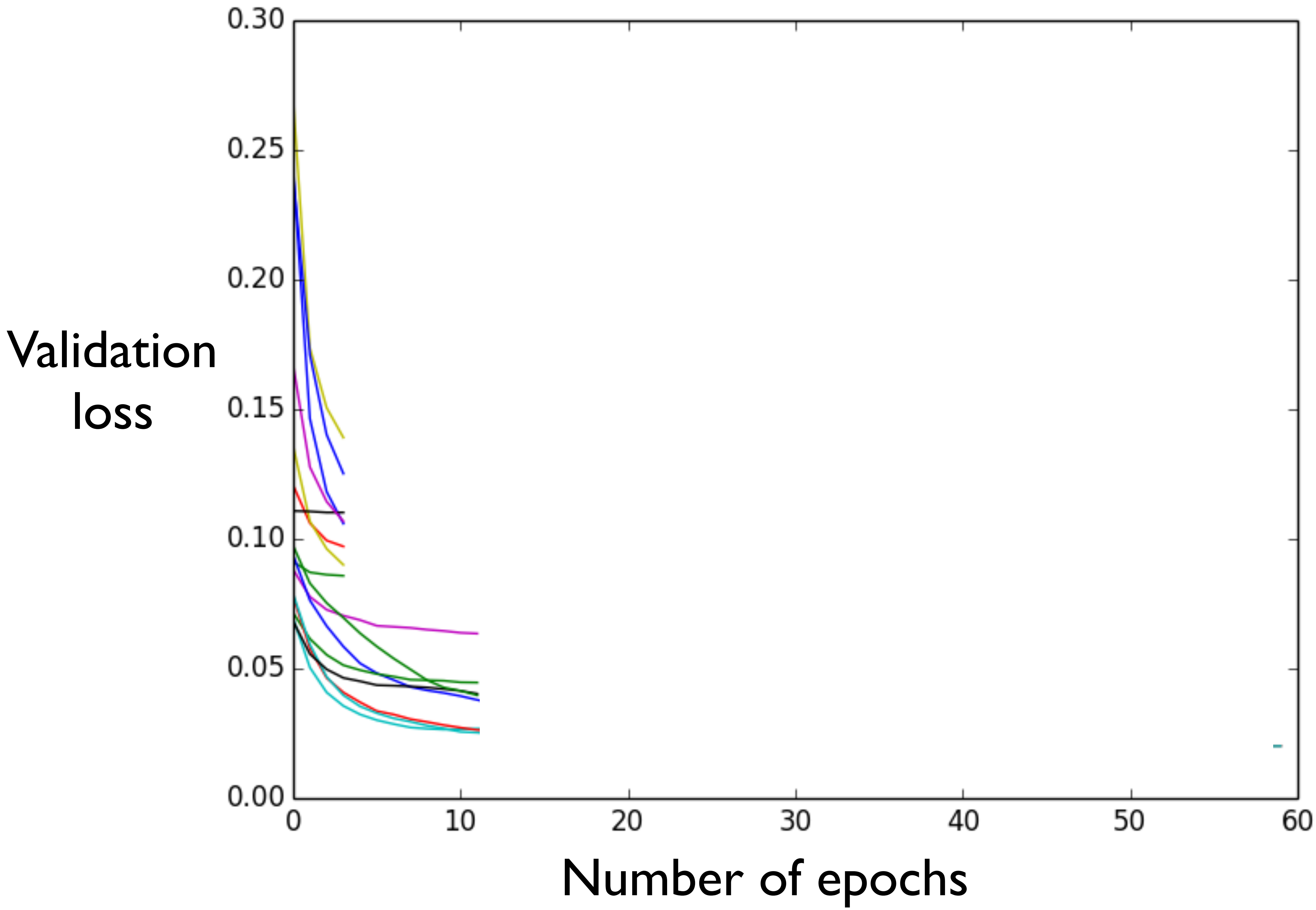
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029



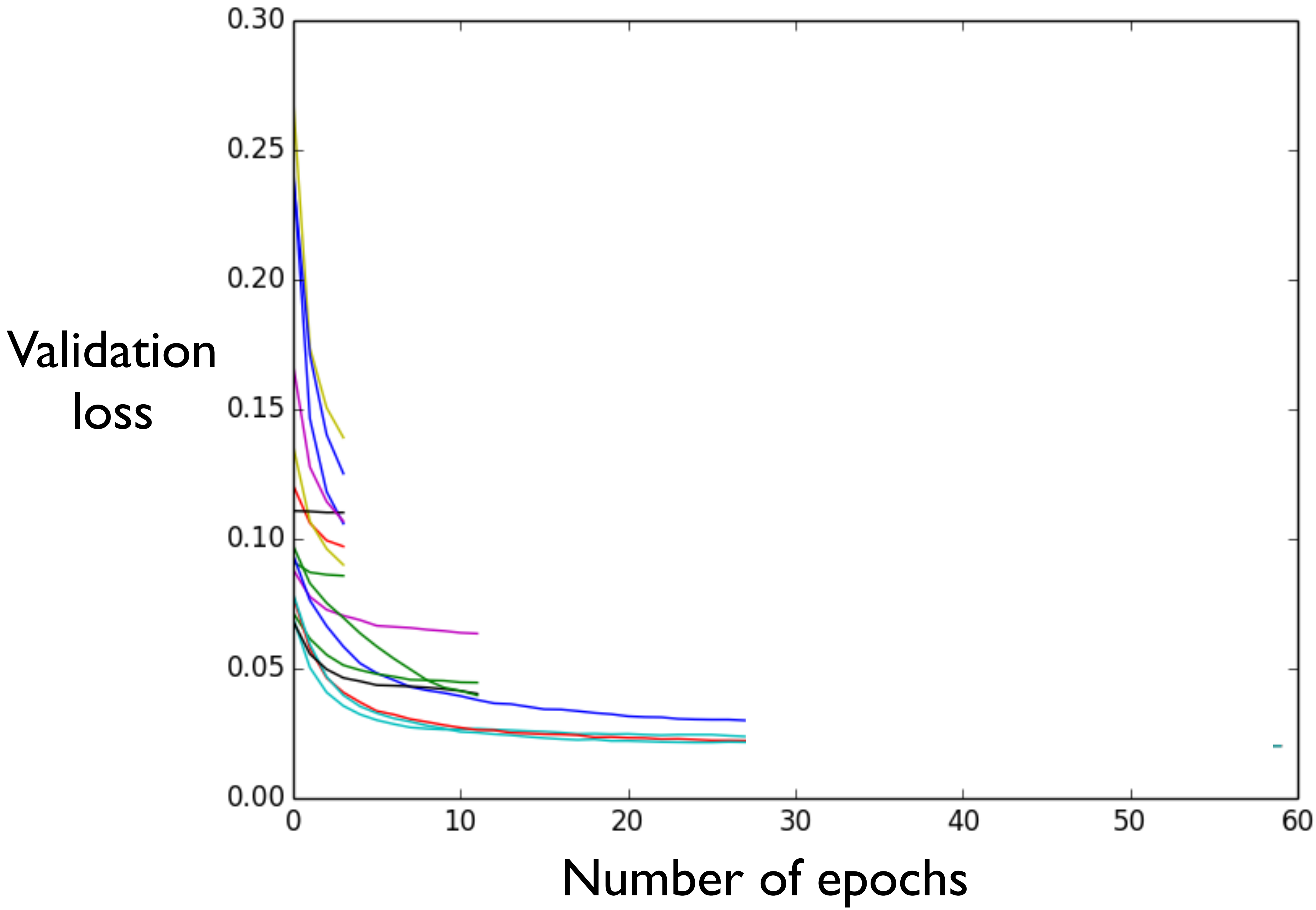
An early-stopping heuristic…



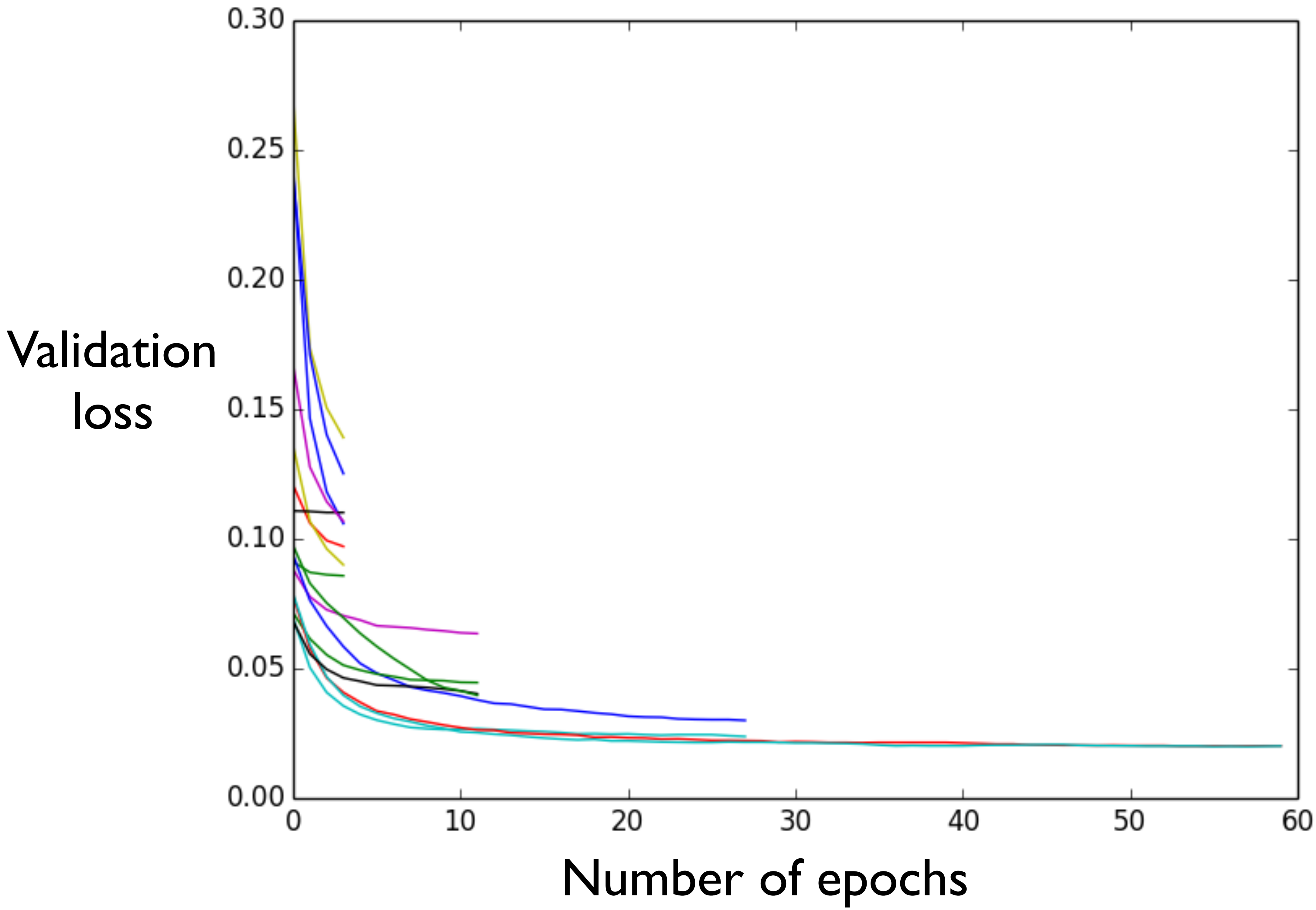
An early-stopping heuristic…



An early-stopping heuristic…

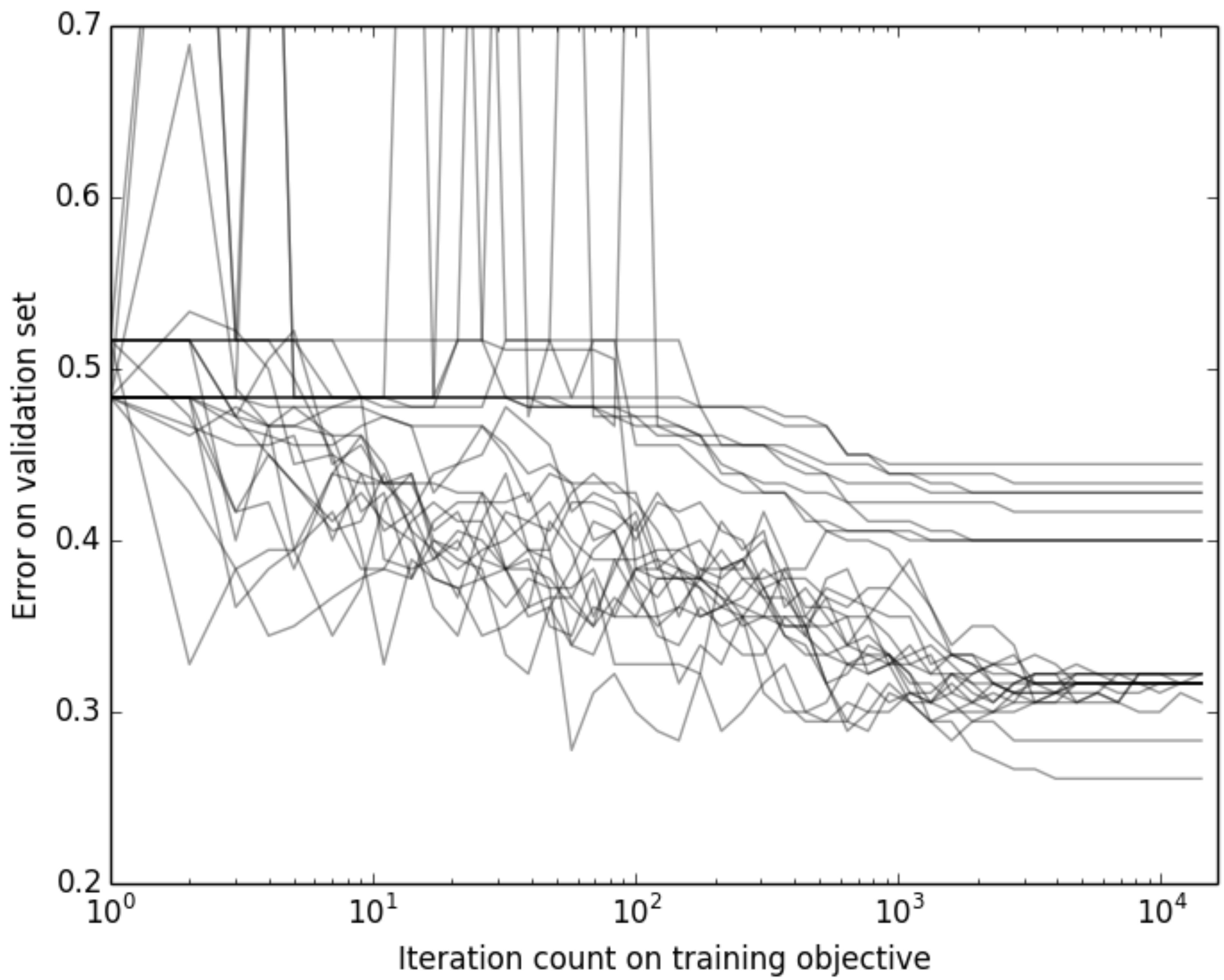


An early-stopping heuristic…



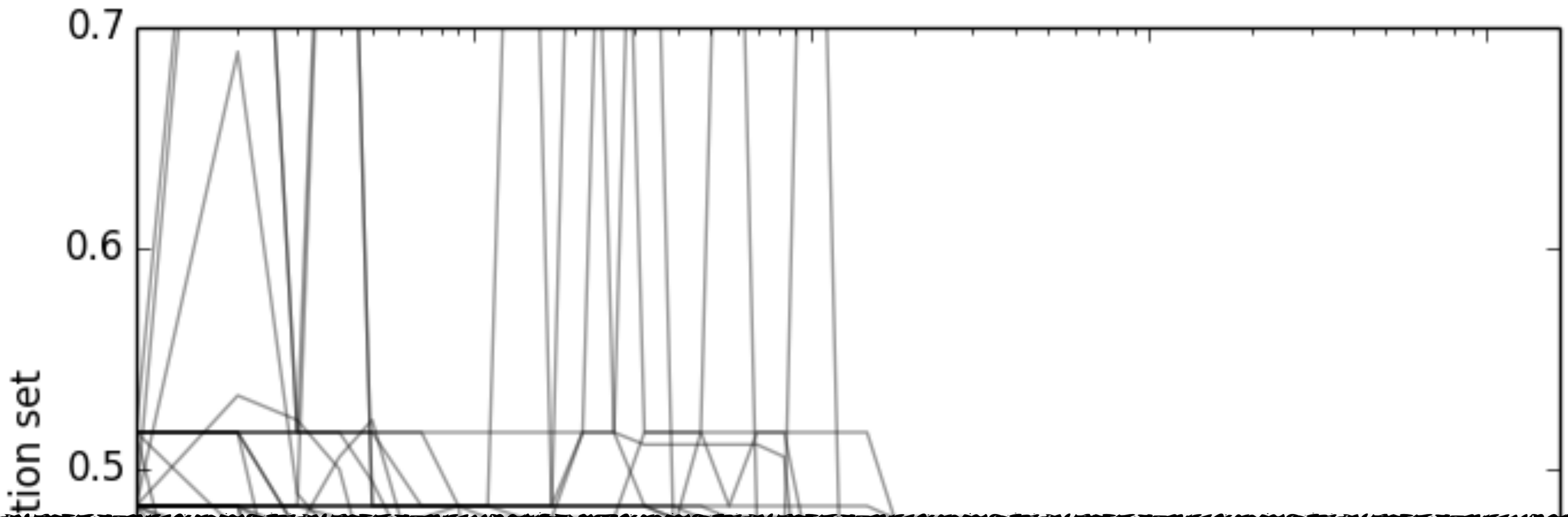
What could go wrong?

Sequences can be non-monotonic, non-smooth, and have different rates of convergence



What could go wrong?

Sequences can be non-monotonic, non-smooth, and have different rates of convergence



Main challenges for an algorithm:

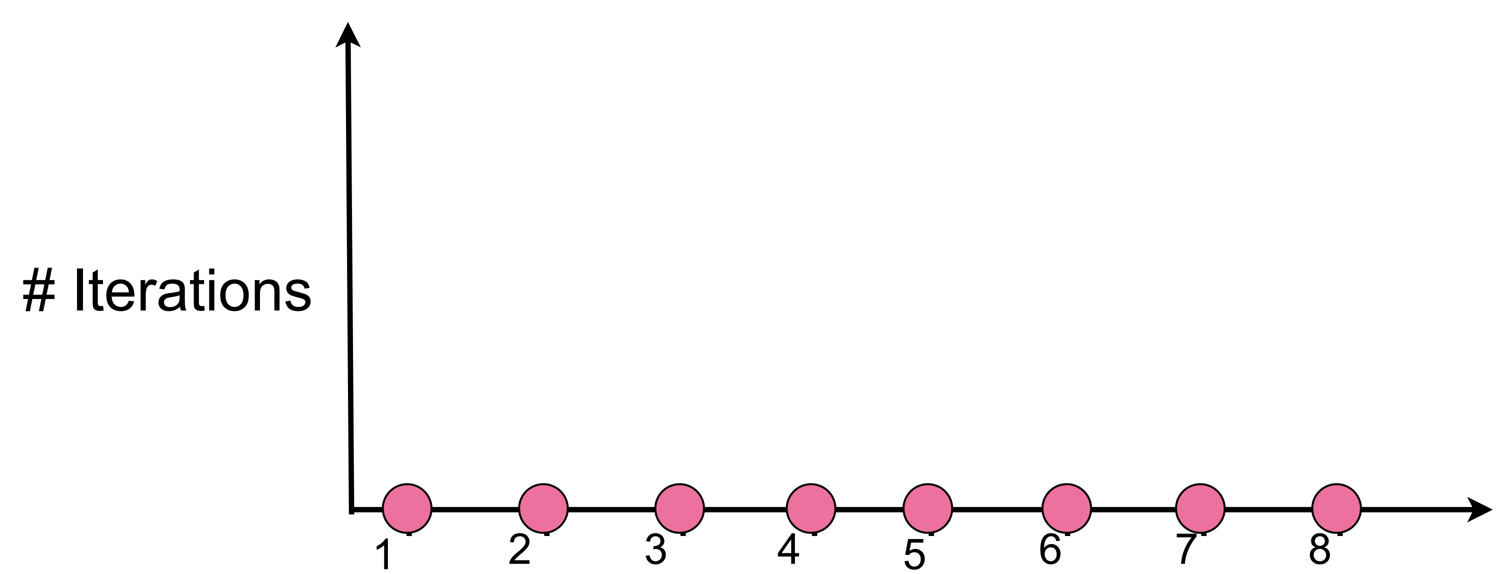
1) What scheme to use to allocate iterations?
2) What is the minimum iteration to throw out configs?

Does there exist an algorithm that provably works and also demonstrates good empirical performance?

Successive Halving (SH)

- Assume training algorithm executes for a maximum number of iterations (R)
- Our toy problem
  - R = 28
  - Budget is B = 96
  - Number of configurations is n = 8

R = 28
B = 96
n = 8

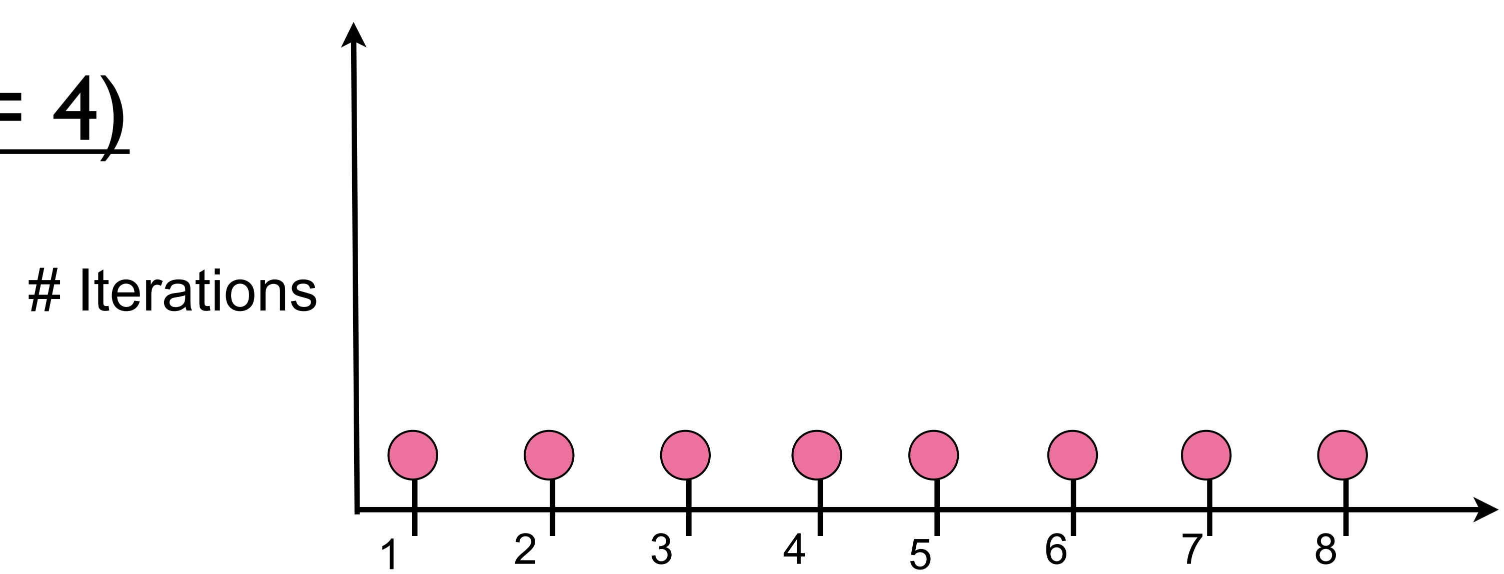


Successive Halving (SH)

1. Uniformly allocate resources among active configurations

round 1 budget is 32 (r_1 = 4)

R = 28
B = 96
n = 8

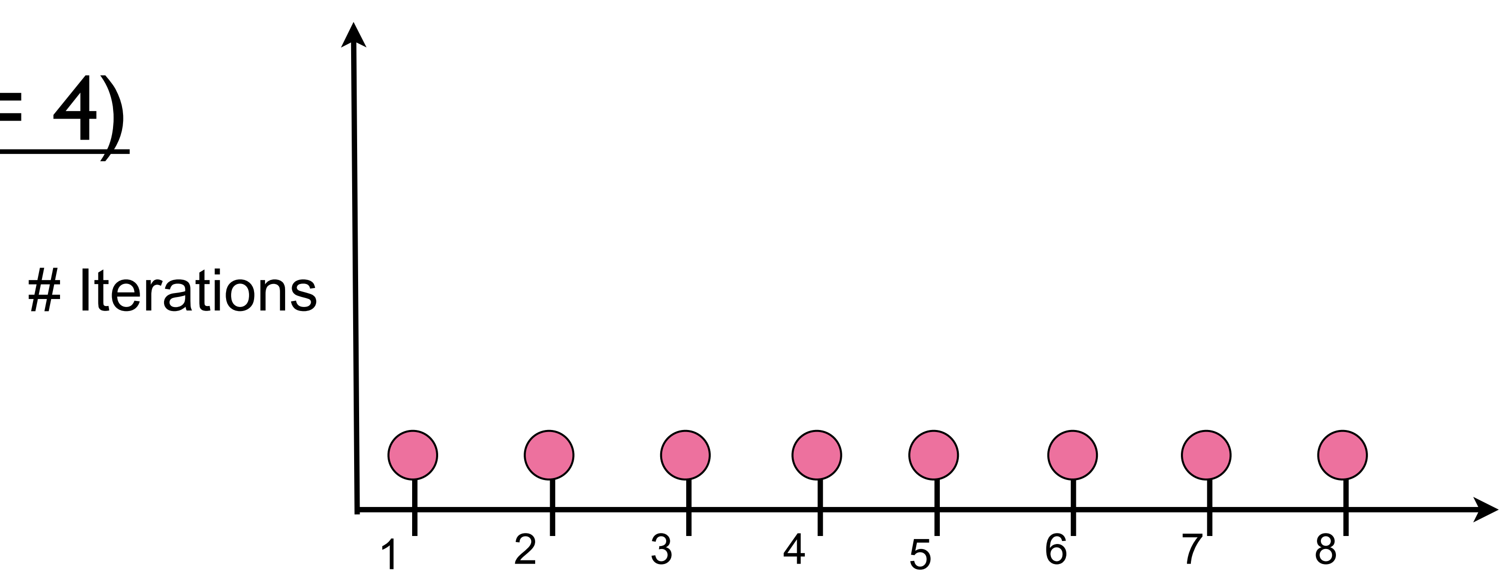


Successive Halving (SH)

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm

<u>round 1 budget is 32 (r_1 = 4)</u>

R = 28
B = 96
n = 8

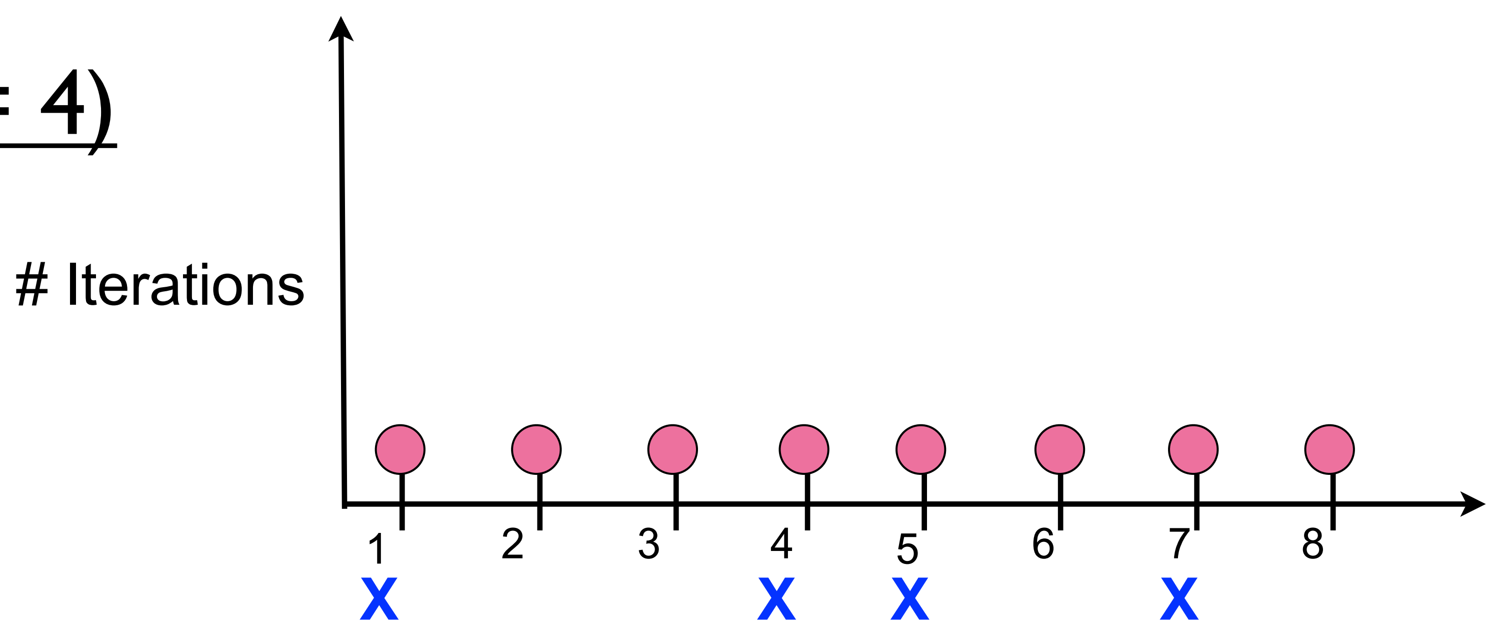


Successive Halving (SH)

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm
3. Throw out the worst half

round 1 budget is 32 ($r_1 = 4$)

R = 28
B = 96
n = 8



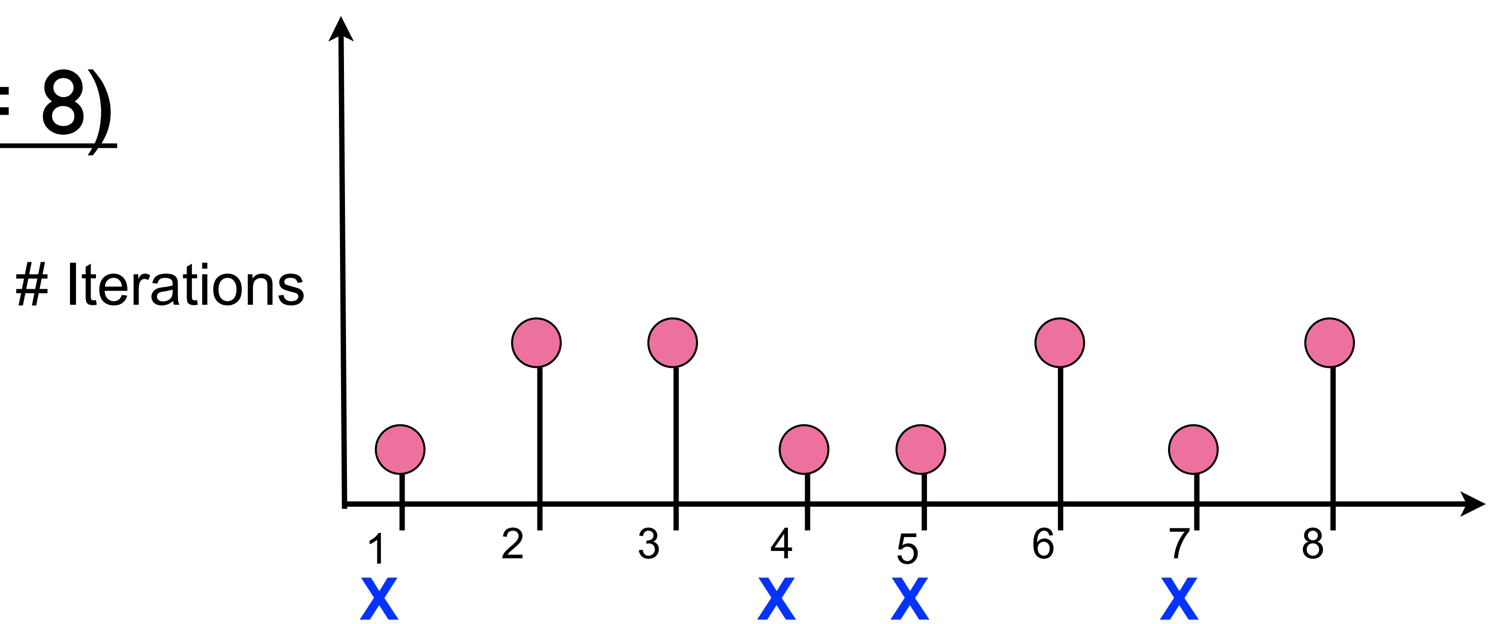
Successive Halving (SH)

Repeat until total budget exhausted

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm
3. Throw out the worst half

round 2 budget is 32 ($r_2 = 8$)

R = 28
B = 96
n = 8



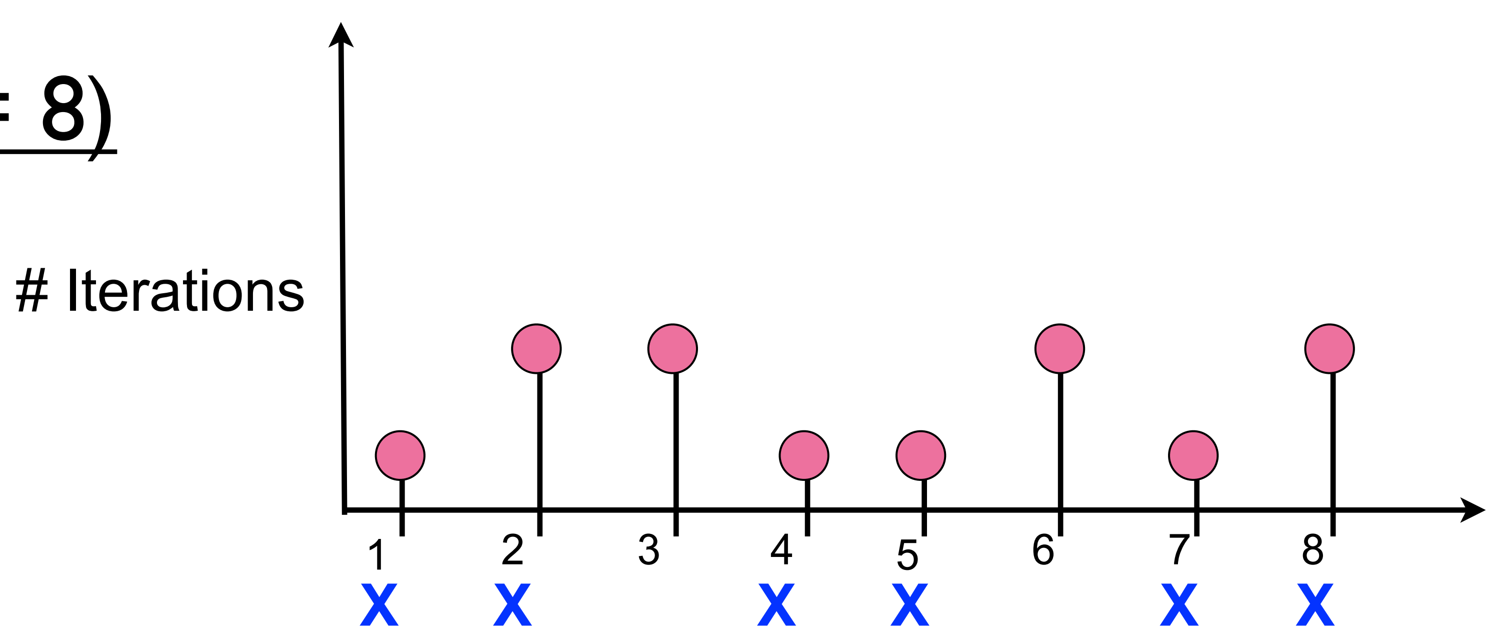
Successive Halving (SH)

Repeat until total budget exhausted

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm
3. Throw out the worst half

round 2 budget is 32 ($r_2 = 8$)

R = 28
B = 96
n = 8



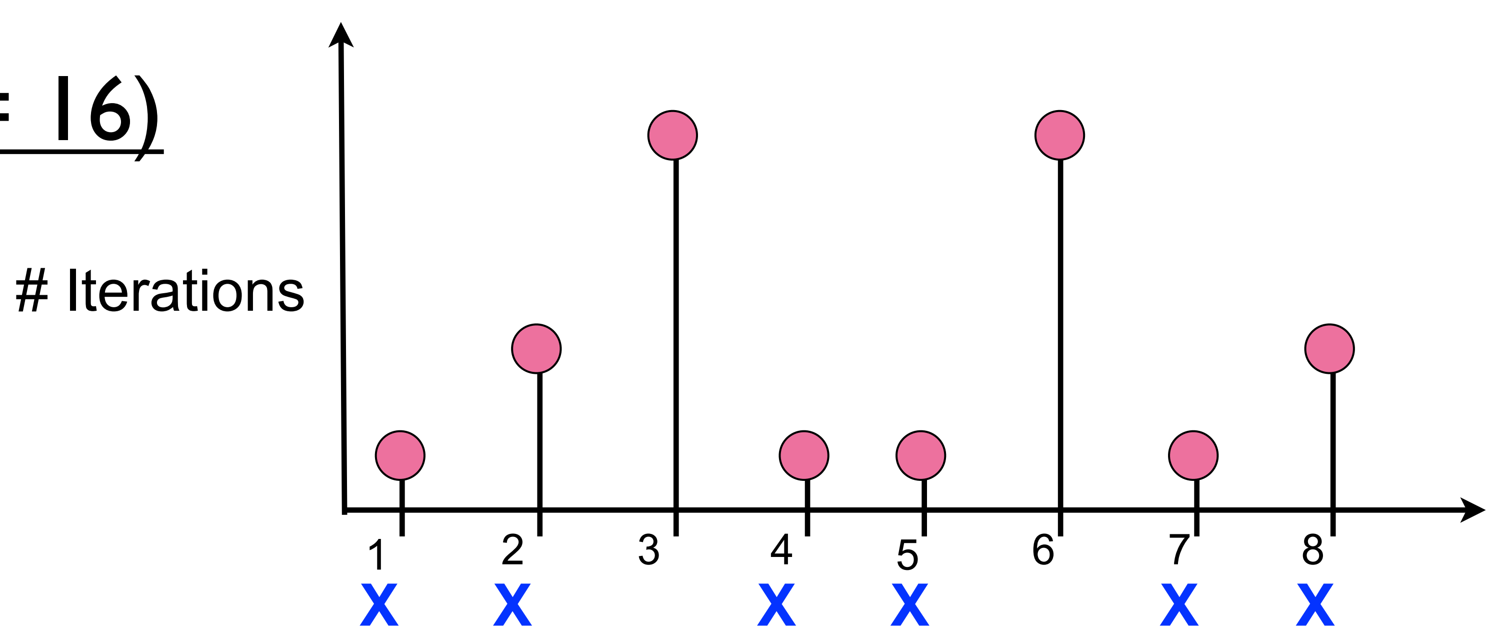
Successive Halving (SH)

Repeat until total budget exhausted

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm
3. Throw out the worst half

round 3 budget is 32 (r_2 = 16)

R = 28
B = 96
n = 8



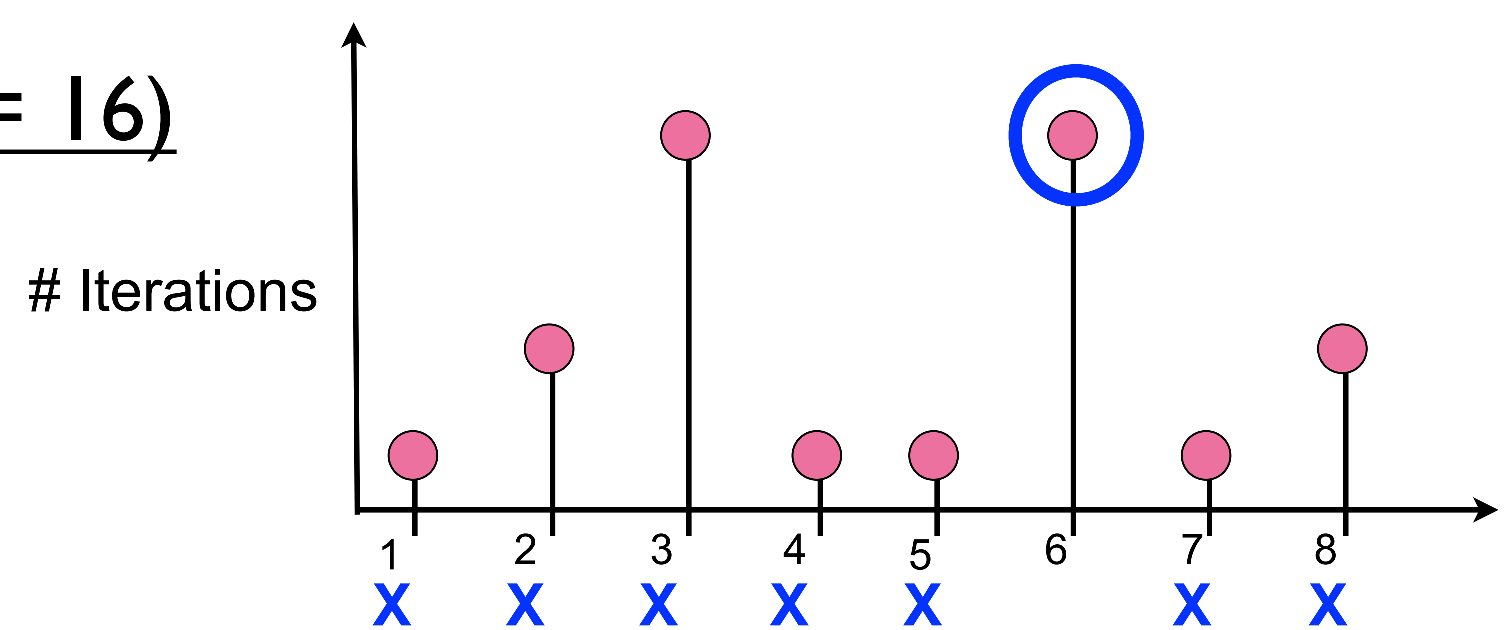
Successive Halving (SH)

Repeat until total budget exhausted

1. Uniformly allocate resources among active configurations
2. Evaluate performance of each arm
3. Throw out the worst half

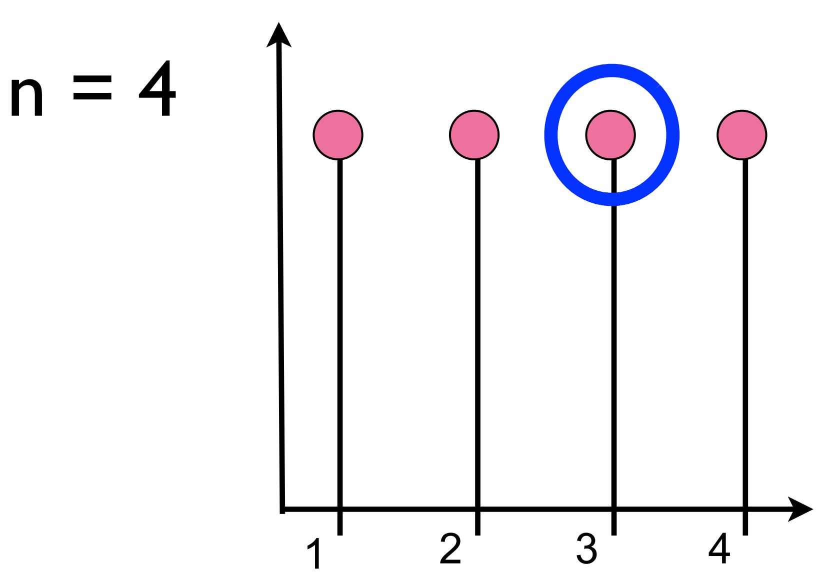
<u>round 3 budget is 32 (r_2 = 16)</u>

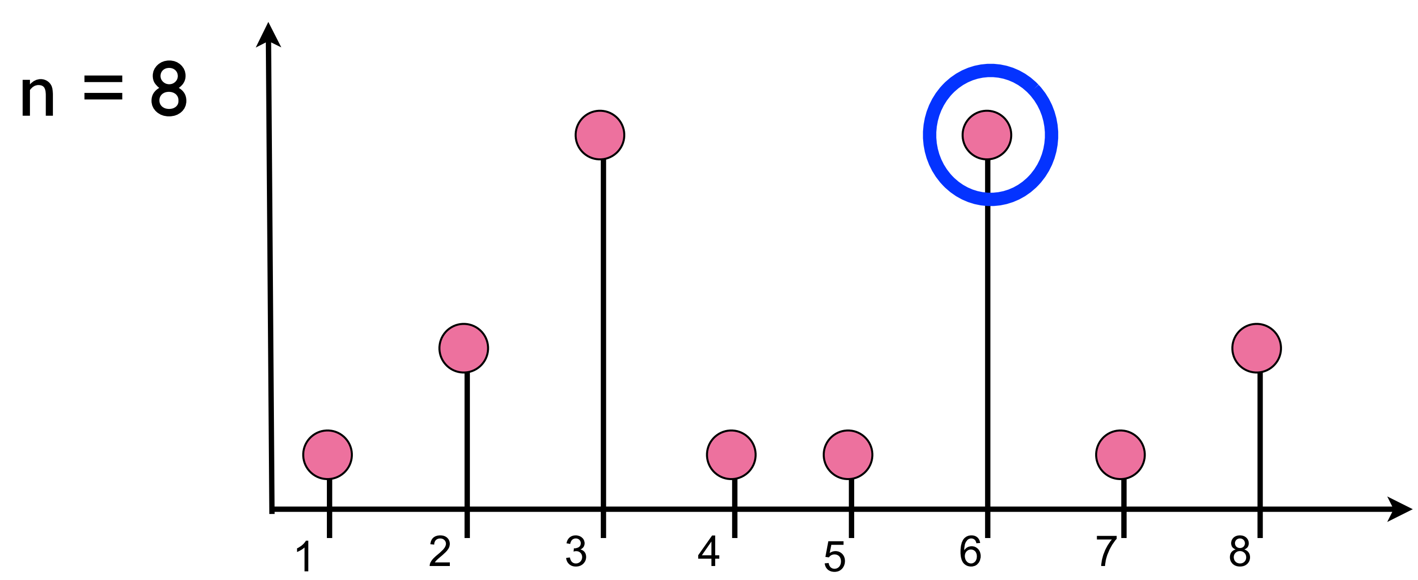
R = 28
B = 96
n = 8



n-versus-B Tradeoff

- For fixed B, we want biggest n possible w/o throwing away a good configuration too quickly
- Problem specific, and depends on underlying (and unknown) convergence properties





n-versus-B Tradeoff

- For fixed B, we want biggest n possible w/o throwing away a good configuration too quickly
- Problem specific, and depends on underlying (and unknown) convergence properties

Hyperband: try 'all' values of n for a given B!

- Max and min values of n determined by R (we require at least one configuration trained on R)
- Perform grid search on this range (in log space)

Algorithm 1: HYPERBAND algorithm for hyperparameter tuning

input : R, η (default $\eta = 3$)

initialization : $s_{\max} = \lfloor \log(R)/\log(\eta) \rfloor$, $B = (s_{\max}+1)R$

for $s \in \{s_{\max}, s_{\max}-1, \ldots, 0\}$ **do**

  $n = \lceil \frac{B\eta^s}{R(s+1)} \rceil$

  $r = R\eta^{-s}$

  $T =$ `get_hyperparameter_configuration`(n)

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

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

    $r_i = r\eta^{i}$

    $L = \{$ `run_then_return_val_loss`(t, r_i) $: t \in T\}$

    $T =$ `top_k`(T, L, n_i)

  end

end

return *Configuration with the smallest loss seen so far.*

Successive "halving", but generalized to arbitrary η

We fix budget B and try different values of n in the outer loop

Algorithm 1: HYPERBAND algorithm for hyperparameter tuning

input : R, η (default $\eta = 3$)

initialization : $s_{\max} = \lfloor \log(R)/\log(\eta) \rfloor$, $B = (s_{\max} + 1)R$

for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**

  $n = \lceil \frac{B\eta^s}{R(s+1)} \rceil$

  $r = R\eta^{-s}$

  $T =$ `get_hyperparameter_configuration`(n)

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

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

    $r_i = r\eta^i$

    $L = \{$ `run_then_return_val_loss`(t, r_i) $: t \in T\}$

    $T =$ `top_k`(T, L, n_i)

  end

end

return *Configuration with the smallest loss seen so far.*

Sample Complexity Guarantees: Pure-exploration Non-stochastic Infinite-armed Bandit Problem

Existing Methods

Hyperband Algorithm

- **Experimental Results**
- Theory (Briefly)

Example: LeNet, SGD on MNIST

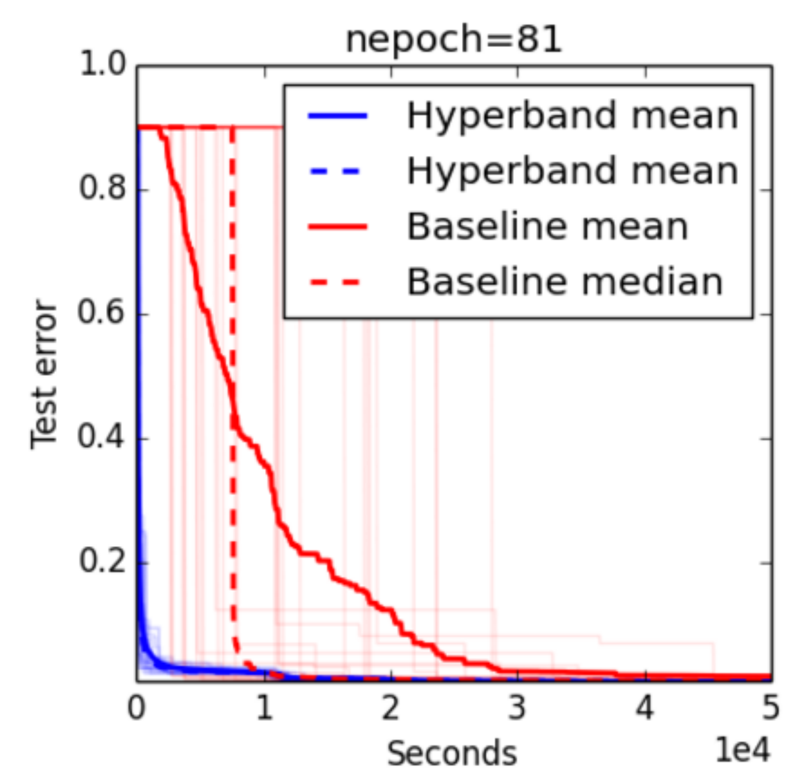
Solver code taken from http://deeplearning.net/tutorial/lenet.html

Hyperparameter	Scale	Min	Max
Learning Rate	log	1e-3	1e-1
Batch size	log	1e1	1e3
Layer 2 Num Kernels (k2)	linear	10	60
Layer 1 Num Kernels (k1)	linear	5	k2

$R = 81$; $B = 5*R$; $\eta = 3$

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
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								

Comparison (low accuracy)

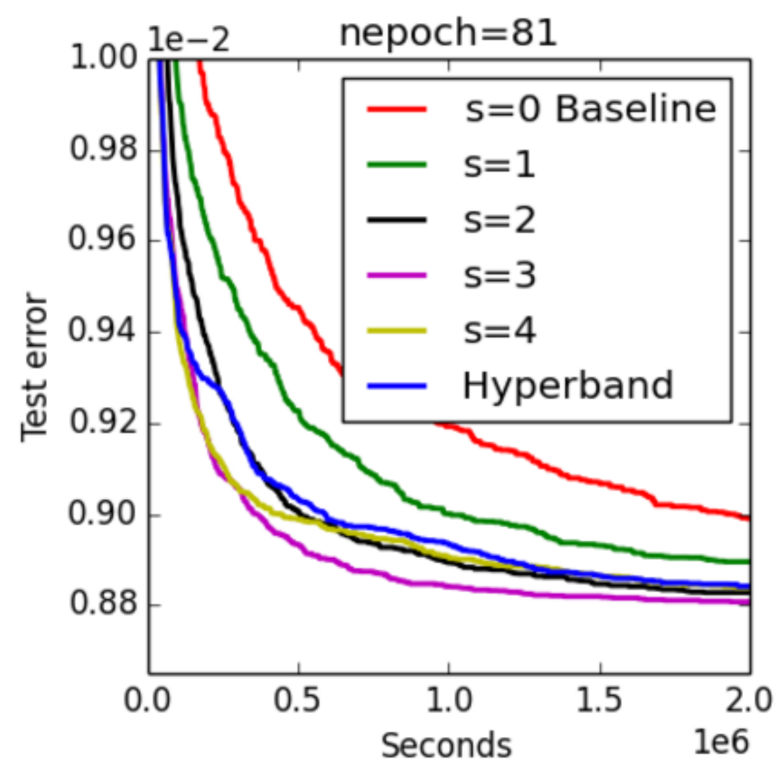


Example: LeNet, SGD on MNIST

How much does s matter?

Hyperparameter	Scale	Min	Max
Learning Rate	log	1e-3	1e-1
Batch size	log	1e1	1e3
Layer 2 Num Kernels (k2)	linear	10	60
Layer 1 Num Kernels (k1)	linear	5	k2

S-comparison



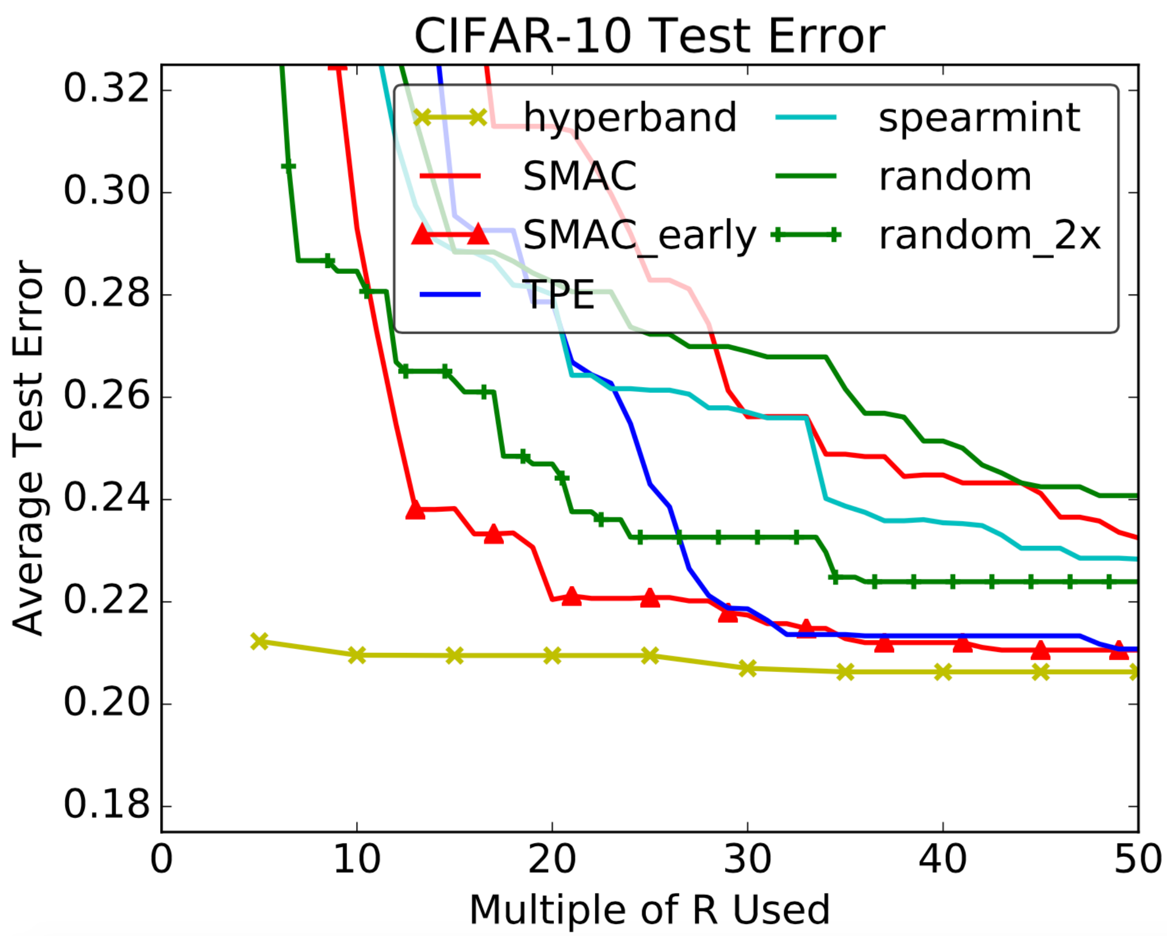
The best value of s is unknowable a priori, so we try them all, and do not lose much

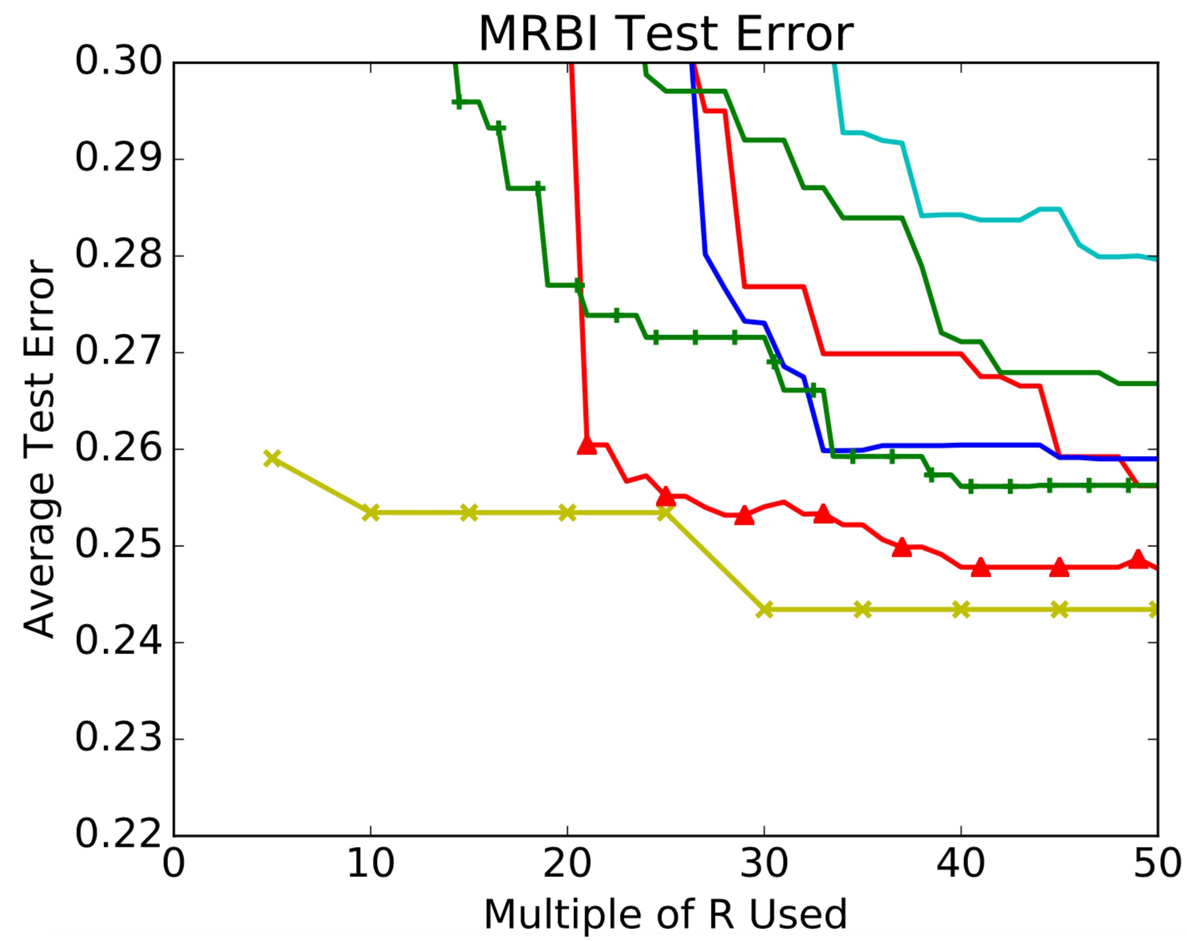
Larger Neural Network Experiments

Setup:

- R=75 epochs over the training set
- Experiments take >2 years in GPU-hours
- Architecture from cuda-convnet (used by Snoek et al. and Domhan et al.)

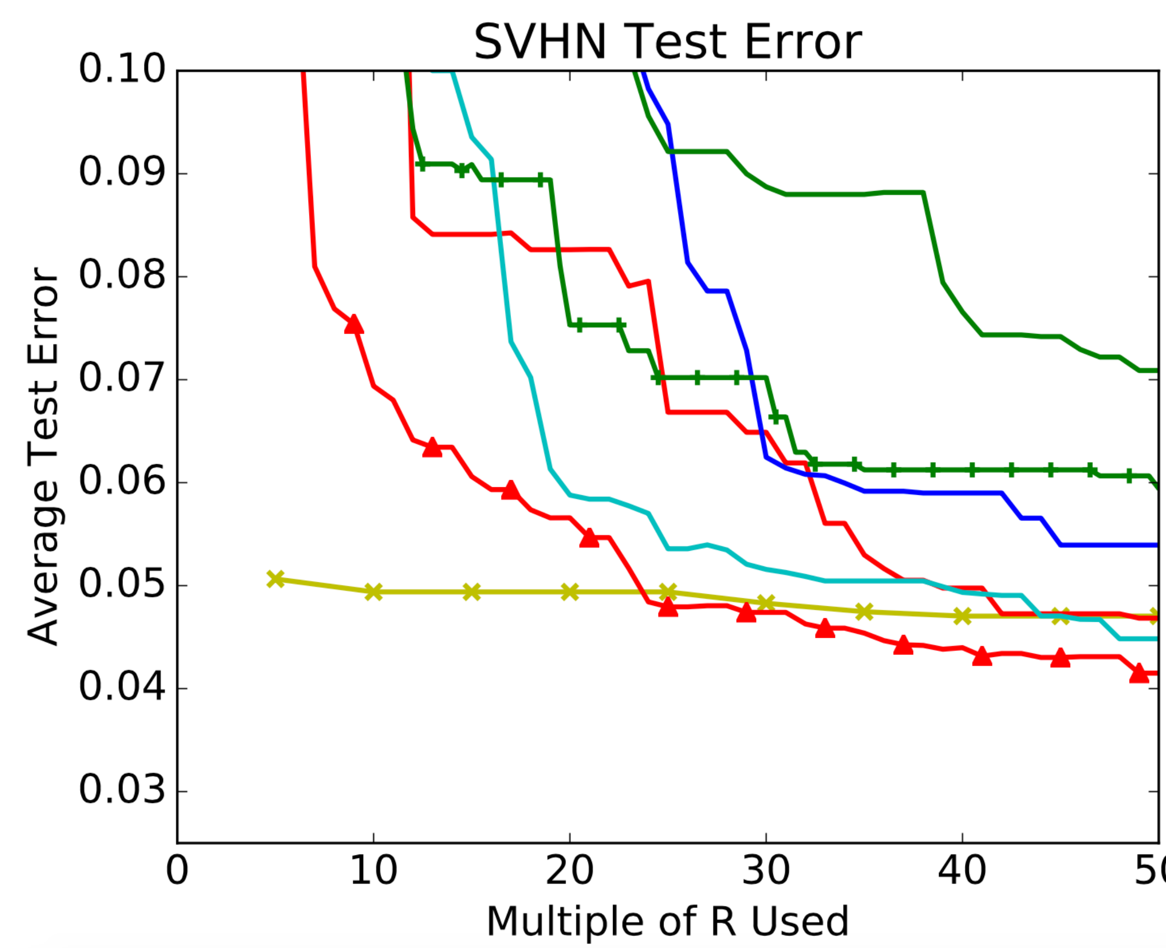
Hyperparameter	Scale	Min	Max
Learning Parameters			
Initial Learning Rate	log	$5 * 10^{-5}$	5
Conv1 l_2 Penalty	log	$5 * 10^{-5}$	5
Conv2 l_2 Penalty	log	$5 * 10^{-5}$	5
Conv3 l_2 Penalty	log	$5 * 10^{-5}$	5
FC4 l_2 Penalty	log	$5 * 10^{-3}$	500
Learning Rate Reductions	integer	0	3
Local Response Normalization			
Scale	log	$5 * 10^{-6}$	5
Power	linear	0.01	3

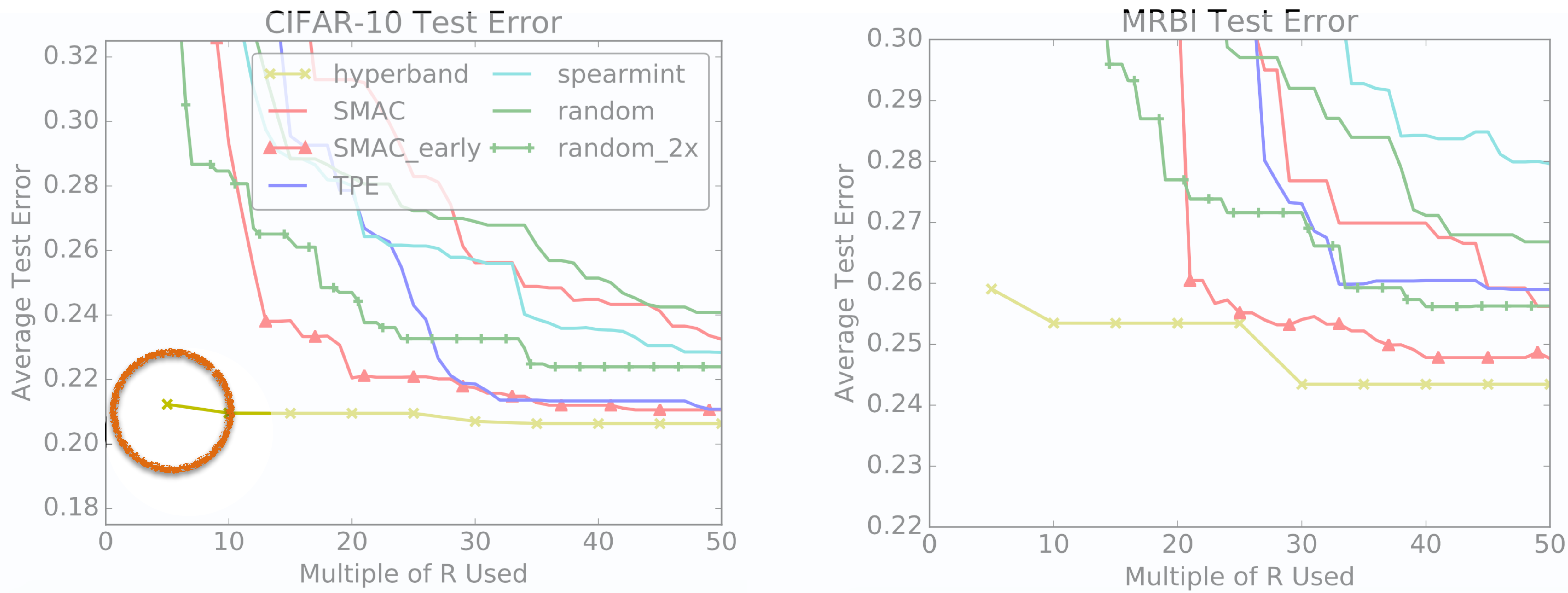




Architecture from Snoek et al. and Domhan et al. from cuda-convnet

Hyperparameter	Scale	Min	Max
Learning Parameters			
Initial Learning Rate	log	$5 * 10^{-5}$	5
Conv1 l_2 Penalty	log	$5 * 10^{-5}$	5
Conv2 l_2 Penalty	log	$5 * 10^{-5}$	5
Conv3 l_2 Penalty	log	$5 * 10^{-5}$	5
FC4 l_2 Penalty	log	$5 * 10^{-3}$	500
Learning Rate Reductions	integer	0	3
Local Response Normalization			
Scale	log	$5 * 10^{-6}$	5
Power	linear	0.01	3





Hyperband exhibits:

- 10x speedup
- Improved final accuracy over purely Bayesian methods
- Lower variance across trials

Hyperband takes 5*R to output anything

- At this point, others have considered **5** configurations, while Hyperband has considered over **256**!

What if my learning algorithm is **not iterative**?

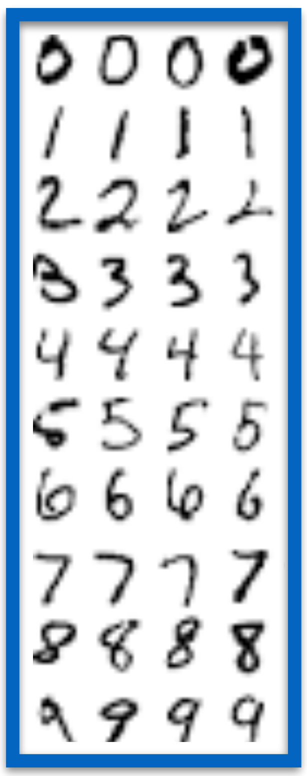
Recall our black-box solver from earlier…

$$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$$

$$\widehat{f}$$

What if my learning algorithm is **not iterative**?

Recall our black-box solver from earlier…

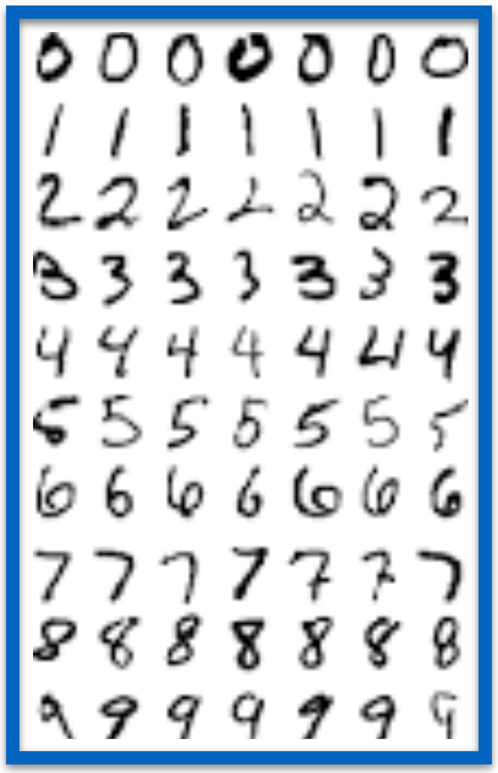


$$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$$

$$\hat{\hat{f}}$$

What if my learning algorithm is **not iterative**?

Recall our black-box solver from earlier…



$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

$\hat{\hat{f}}$

What if my learning algorithm is **not iterative**?

Recall our black-box solver from earlier…

$$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$$

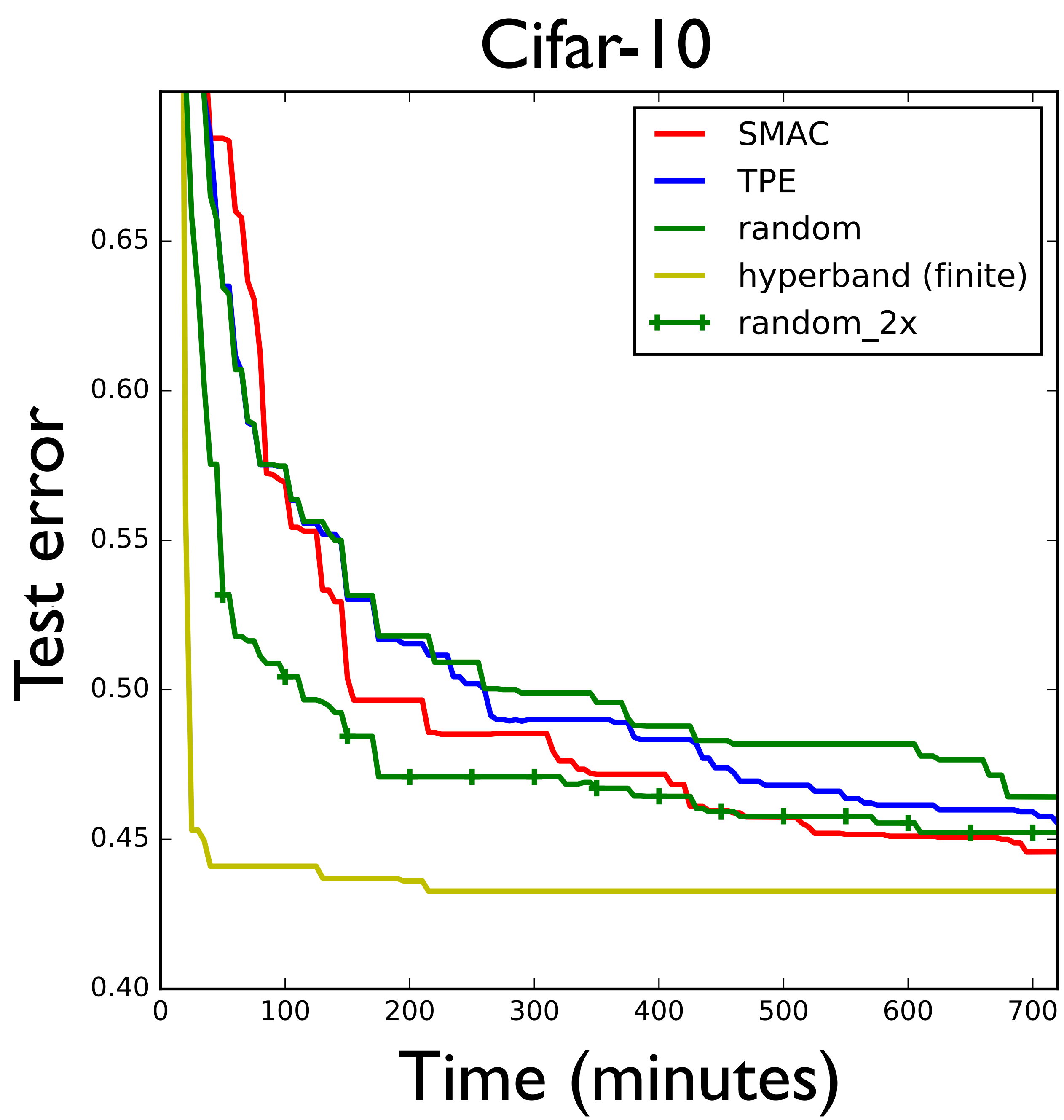
$$\widehat{\widehat{f}}$$

Hyperband with Data Downsampling

Multiclass classification via Kernel LS Regression

Hyperband exhibits:

- 60x speedup over random
- 30x speedup over Bayesian
- improved accuracy



Existing Methods

Hyperband Algorithm

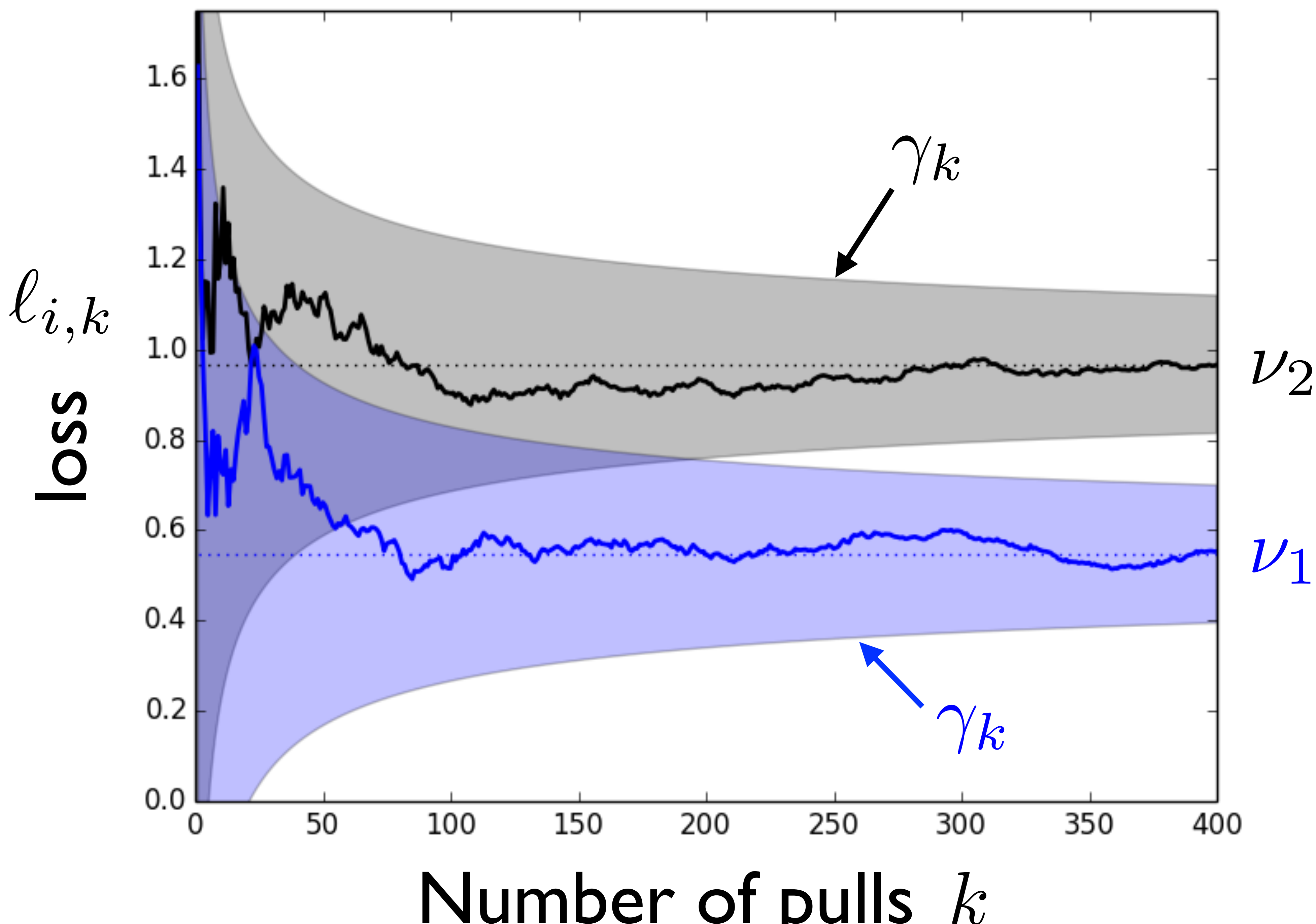
- Experimental Results
- **Theory (Briefly)**

Extensions

What are the relevant quantities? (Neither of which known to the algorithm)

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

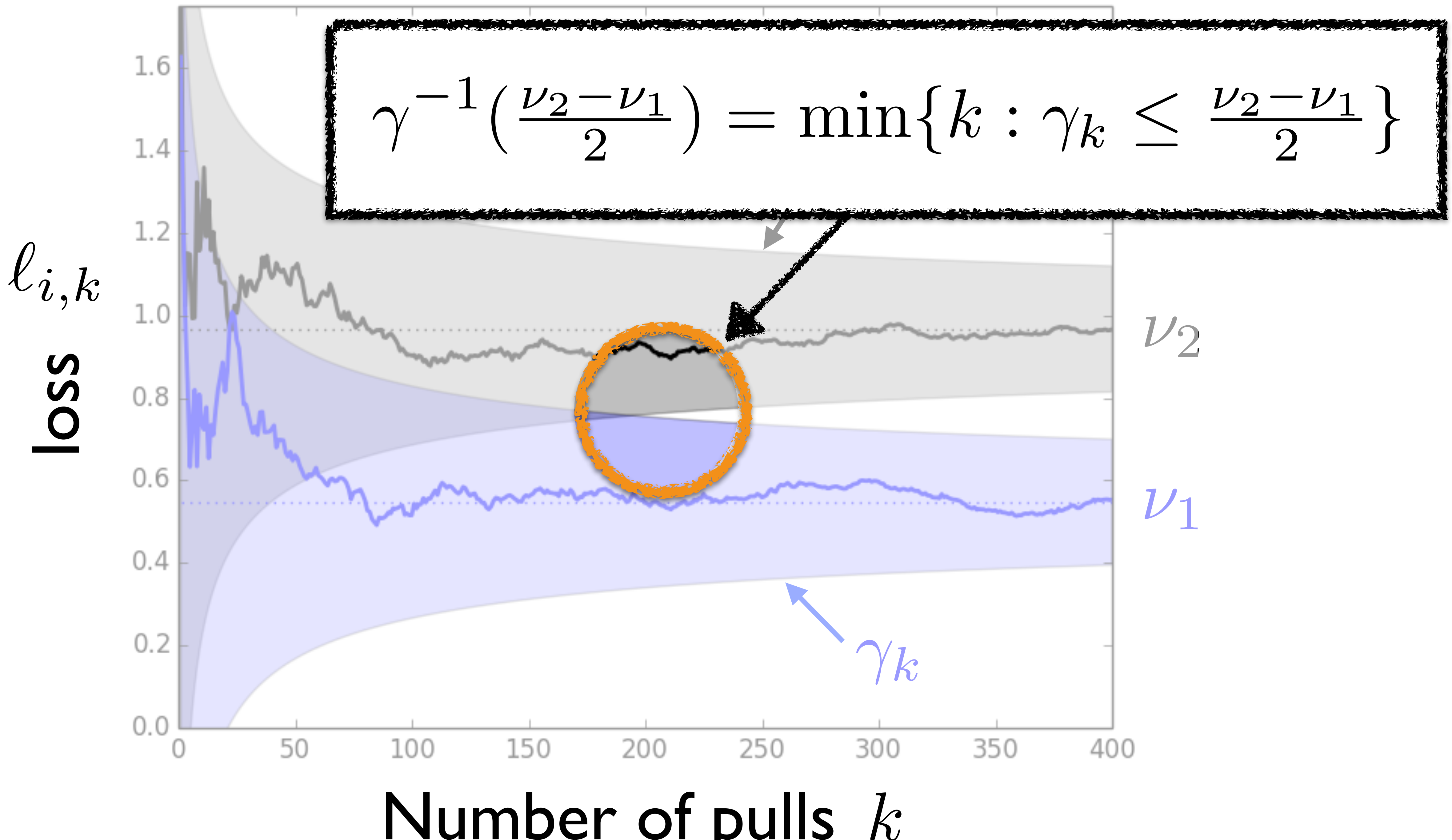
$$|\ell_{i,k} - \nu_i| \leq \gamma_{i,k} \quad \forall i \in [n], k \geq 1$$



What are the relevant quantities? (Neither of which known to the algorithm)

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

$$|\ell_{i,k} - \nu_i| \le \gamma_{i,k} \quad \forall i \in [n], k \ge 1$$



The best arm is identified if the budget is at least

$$2\log(n)\sum_{i=2}^{n}\gamma^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)$$

Successive Halving

$$n\max_{i=2,\ldots,n}\gamma^{-1}\left(\frac{\nu_i-\nu_1}{2}\right)$$

Uniform allocation

Difference between sum and n*max can be large!

More realistic setting: find a 'good' arm

- Assume arms sampled from some unknown distribution
- Can derive similar results comparing SH to Uniform
- Can generalize to *Hyperband*
- See paper for details…

Early stopping is not a new idea

Hyper-parameter optimization / model selection

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. COLT, 2012.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

Non-convex optimization via random-initializations

András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. JAIR, 41, 2011.

Previous works assume **explicit convergence behavior**

Hyperband adapts to it (doesn't rely on knowledge of γ_k !)

Immediate Extensions of Hyperband

Hyperband applies to **general resources**:

- iterations
- dataset subsampling
- feature subsampling: useful when using *random features* to approximate kernels
- time: similar to iterations; useful in distributed setting to kill stragglers

Don't want to set R?

- See paper for 'infinite horizon' version of Hyperband

Hyperband Summary

Looks at more configurations to speed up random search

- Particularly useful when # evaluations linear in number of hyperparameters

Up to 70X faster than random search

General purpose: no assumptions on convergence rates

Papers with theory and some extensions

- AISTATS16: http://arxiv.org/abs/1502.07943
- More recently on arXiv: http://arxiv.org/abs/1603.06560