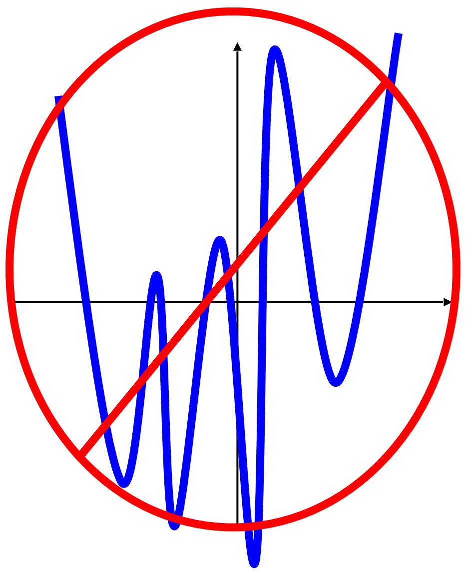
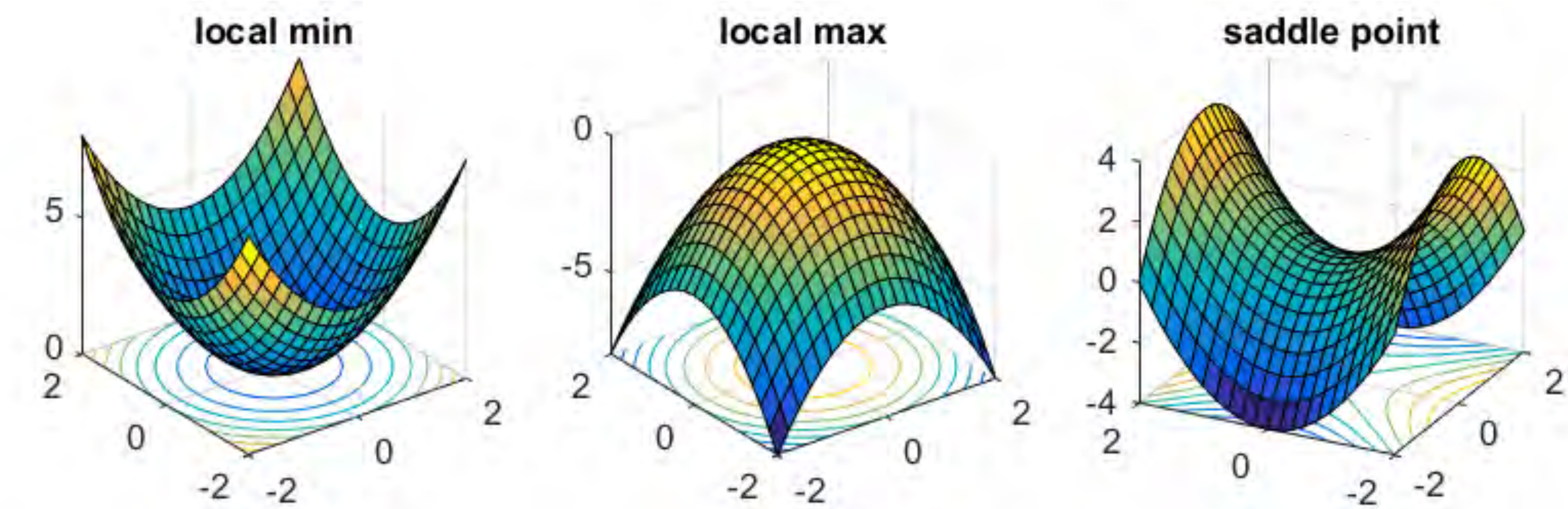
What's so hard about optimizing deep networks?

Benjamin Recht
University of California, Berkeley

What makes optimization of deep models hard?





Recent abstracts on arXiv:

"We prove that recovering the global minimum becomes harder as the network size increases." arXiv:1412.0233

"Difficulty originates from the proliferation of saddle points, not local minima, especially in high dimensional problems of practical interest. " arXiv:1406.2572

It's hard to hit a saddle

$$f(x) = \frac{1}{2}\sum_{i=1}^{d} a_i x_i^2$$

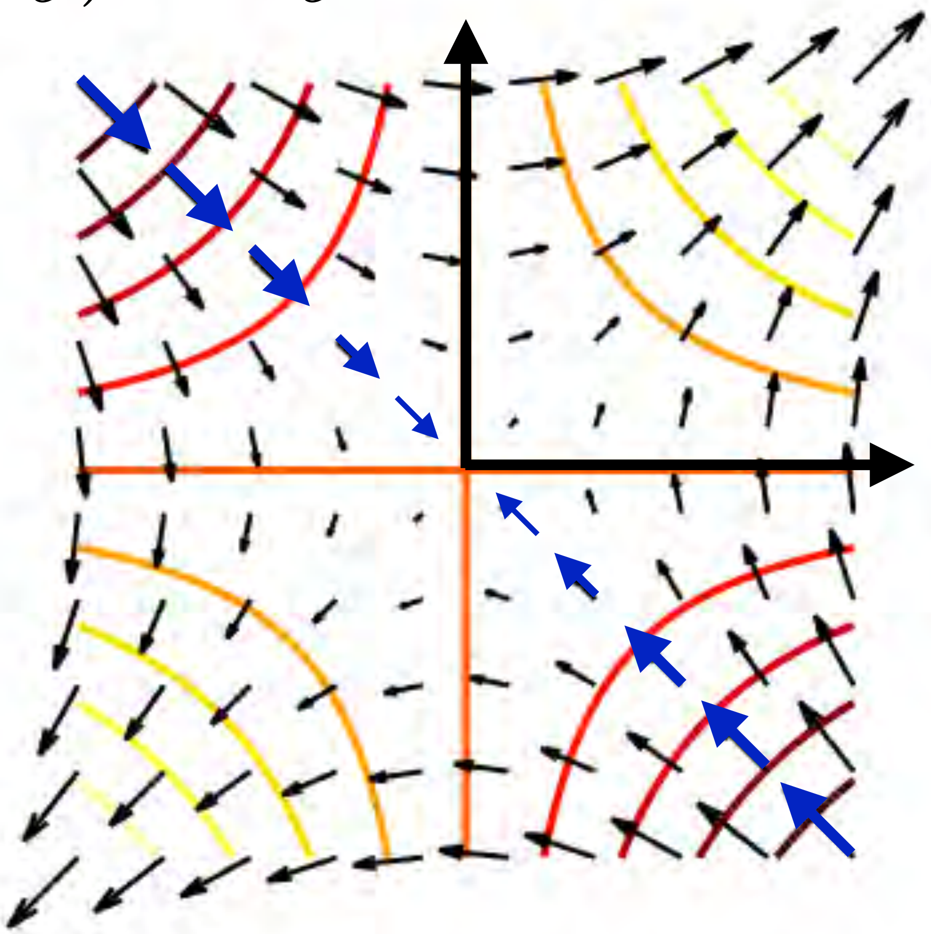
Gradient descent: $x_i^{(k+1)} = (1 - ta_i)x_i^{(k)}$

After k steps $x_i^{(k)} = (1 - ta_i)^k x_i^{(0)}$

If $t|a_i| < 1$ $\begin{cases} \text{converges to 0 if all } a_i \text{ are positive} \\ \text{diverges almost surely if single } a_i \text{ is negative} \end{cases}$

It's hard to hit a saddle

$f(x, y) = xy$



If you are not on the line {*x*=-*y*}, you diverge at an exponential rate

This picture fully generalizes to the nonconvex case

Thm: [Lee et al, 2016] For the short-step gradient method, the basin of attraction of strong saddle points has measure zero.

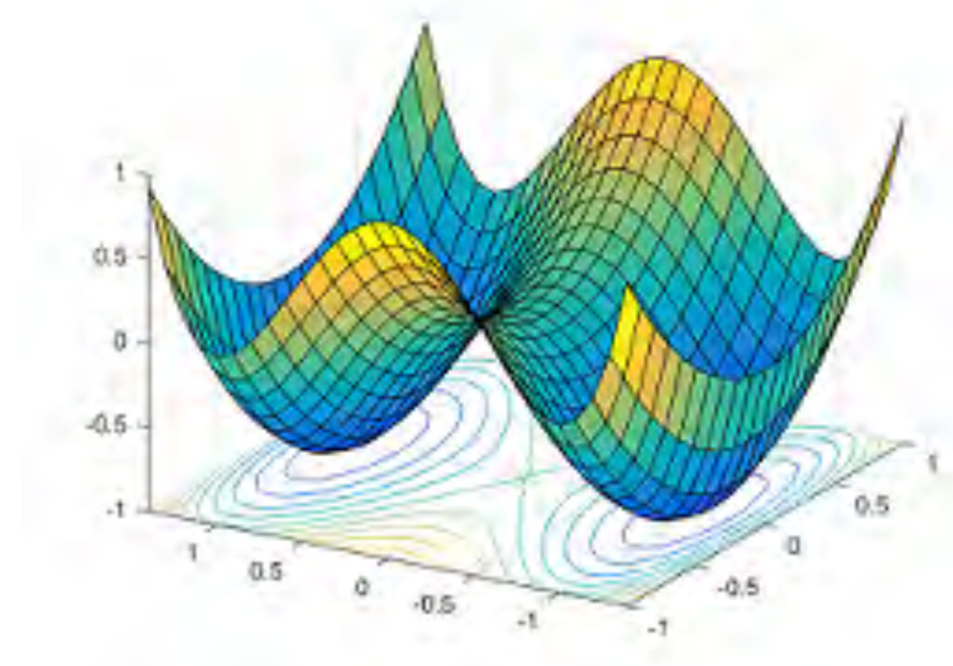
Simple consequence of the *Stable Manifold Theorem* (Smale *et al*)

This is our fault, optimizers.

- Too many fragile examples in text books

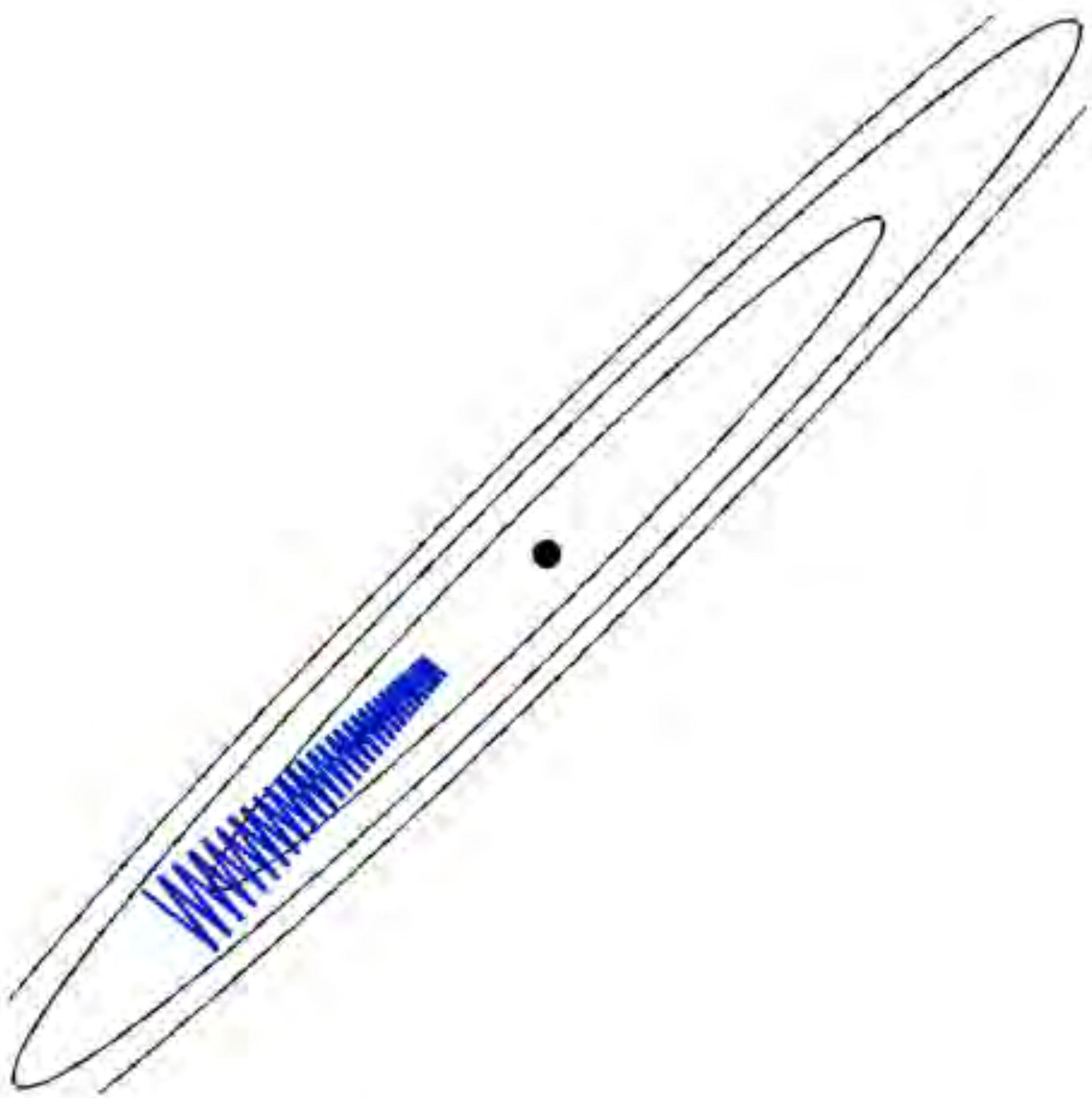
- Minor perturbations in initial conditions always repel you from saddles.

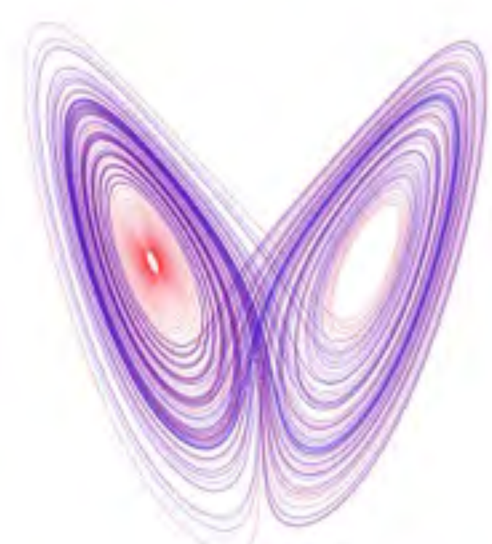
$$f(x,y) = x_1^4 - 2x_1^2 + x_2^2$$

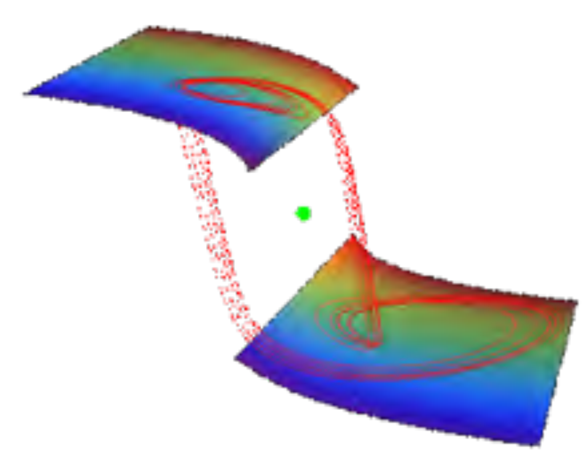


Flatness is what makes things hard

- In convex-land, flat directions (ill-conditioning) slow algorithms down.
- What happens in nonconvex-land?







Flatness is what makes things hard

- What happens in nonconvex-land?

$$\boxed{f(x) = \sum_{i,j=1}^{d} Q_{ij} x_i^2 x_j^2}$$

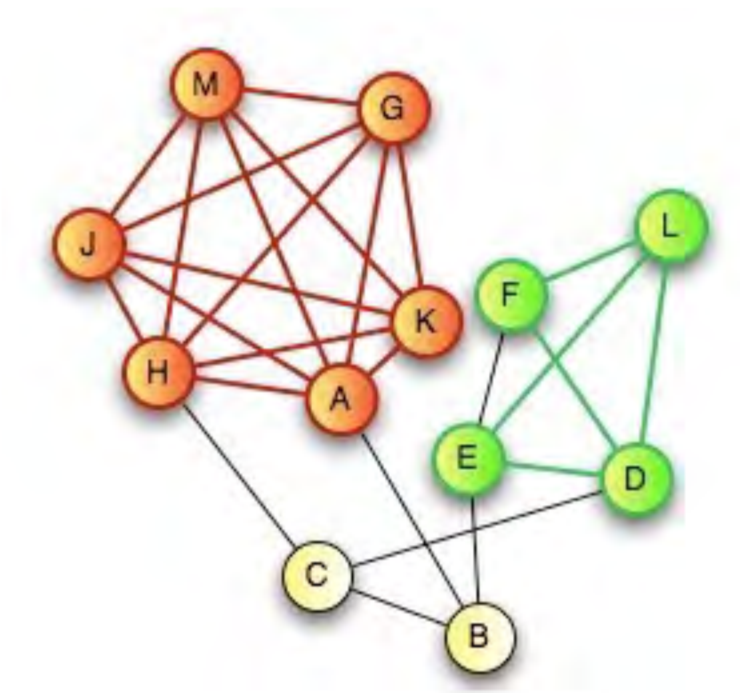
$\nabla f(0) = 0$ Is 0 a global min, saddle, or global max?

$\boxed{\nabla^2 f(0) = 0}$ f is *super flat* at 0.

Deciding if there is a descent direction at 0 is NP-complete

$$f(x) = \sum_{i,j=1}^{d} Q_{ij} x_i^2 x_j^2$$

$$Q = I - A + s \cdot \mathbf{1}\mathbf{1}^T$$



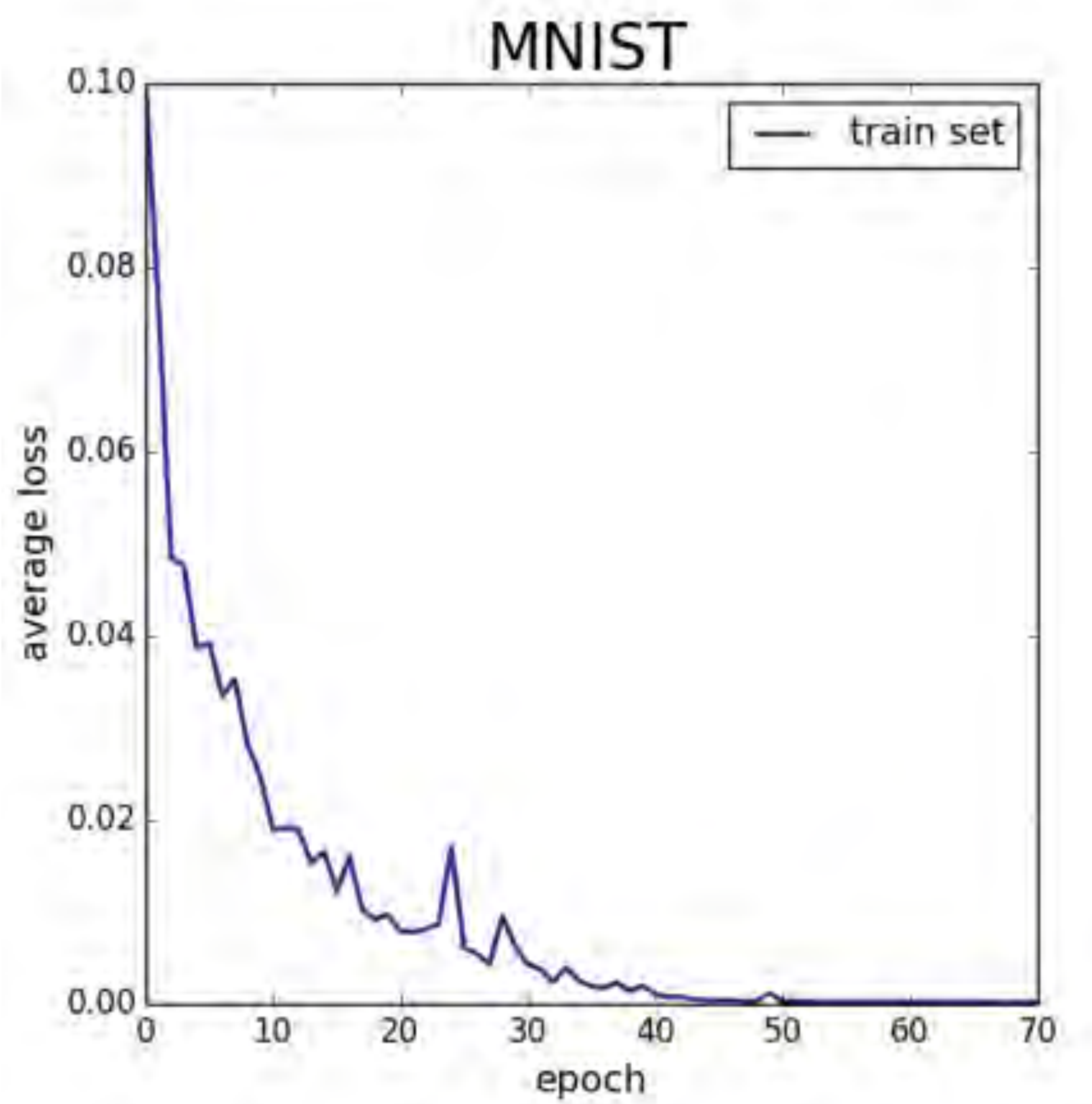
A = adjacency matrix of G

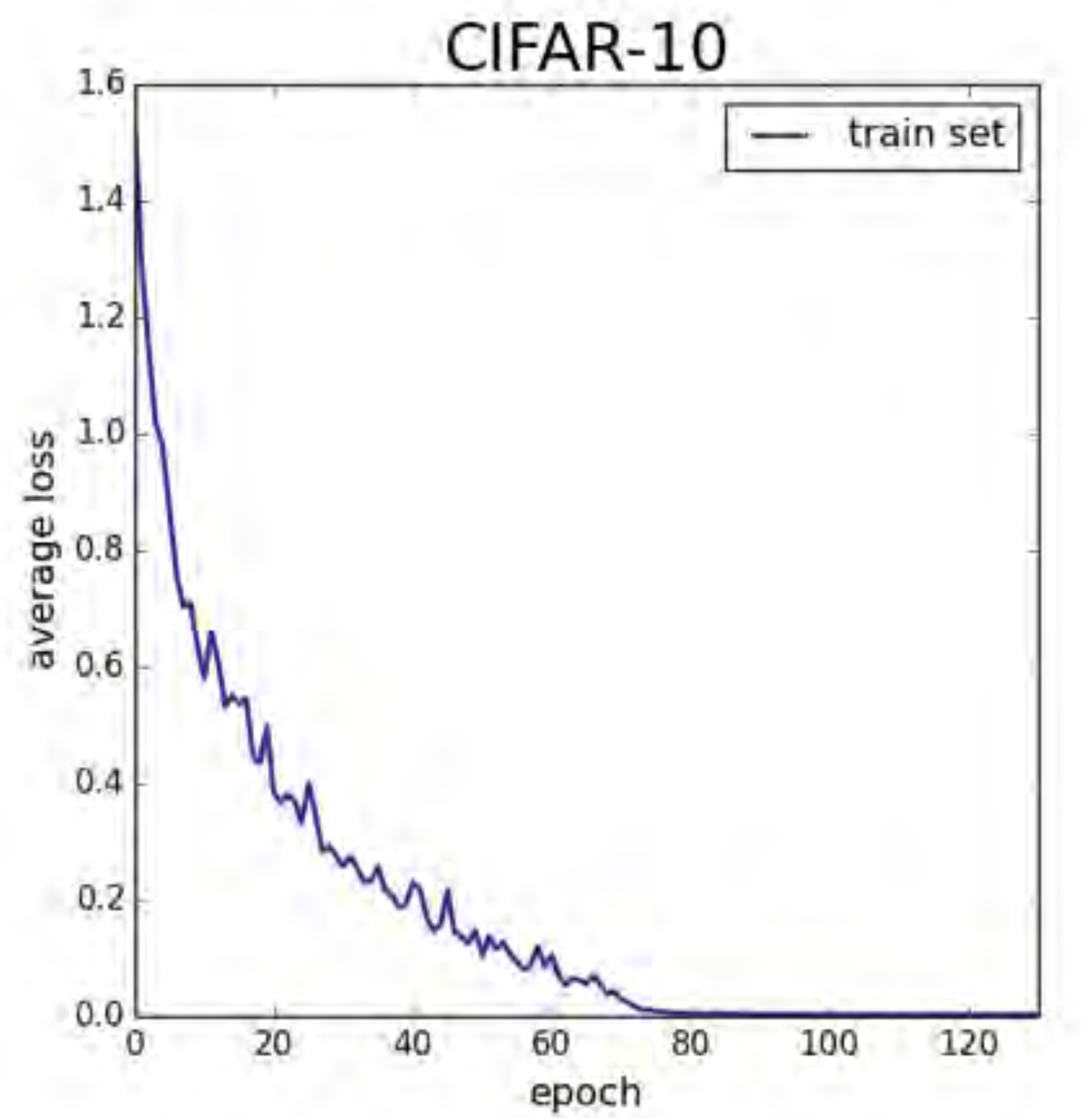
G has an clique of size larger than 1/(1-s) if and only if 0 is not a local minimizer*.

Thm [Barak *et al.* 2016]: Finding a maximum clique is F-hard

* http://www.ti.inf.ethz.ch/ew/lehre/ApproxSDP09/notes/copositive.pdf

Is deep learning as hard as maximum clique?





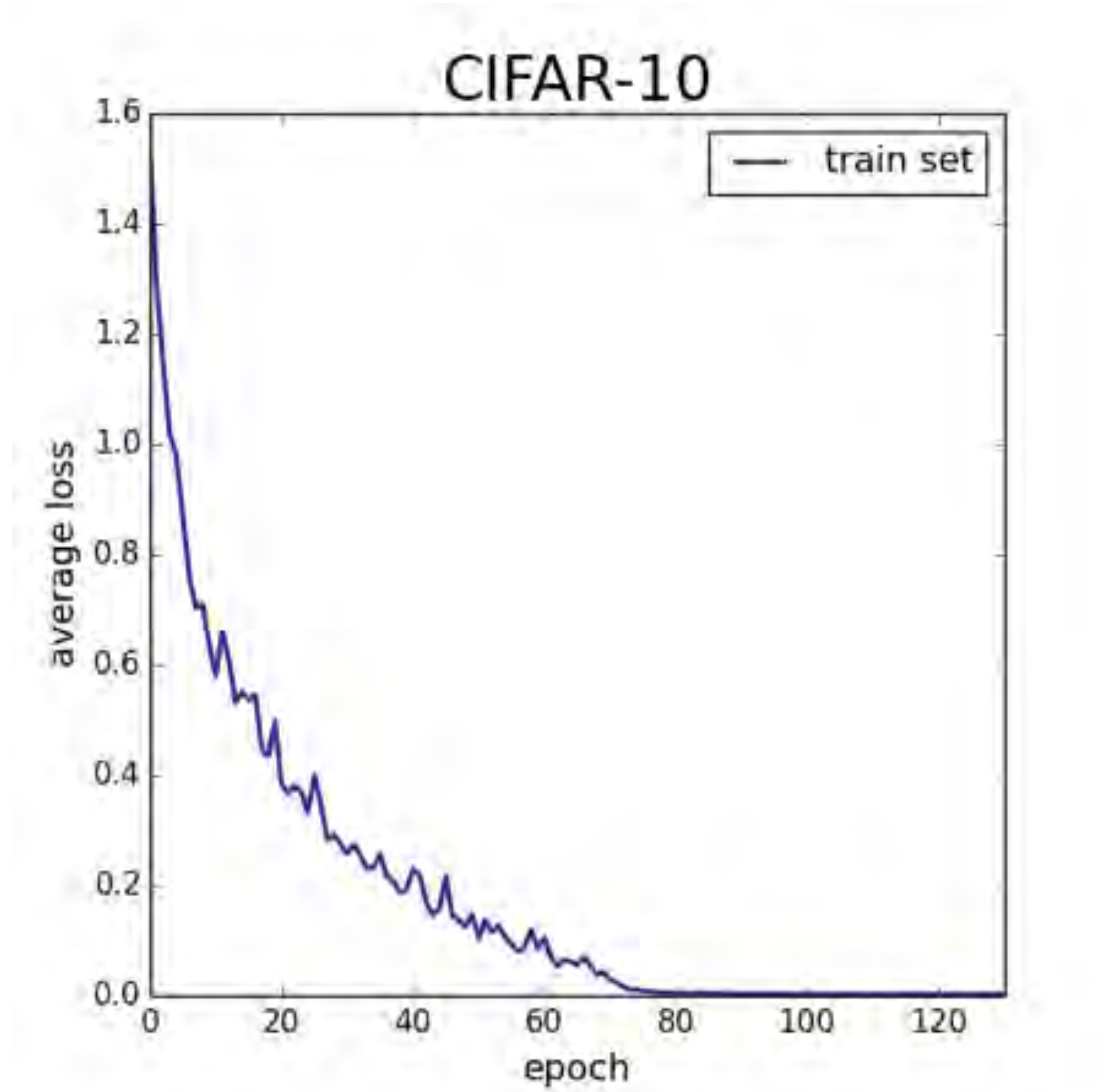
Thm: [Soudry and Carmon 2016]: For an L layer neural network trained with dropout and with $n < d_{L-2}d_{L-1}$, any stable local minimum is a global minimum with loss 0 almost surely.

How can you get to zero with constant stepsize?

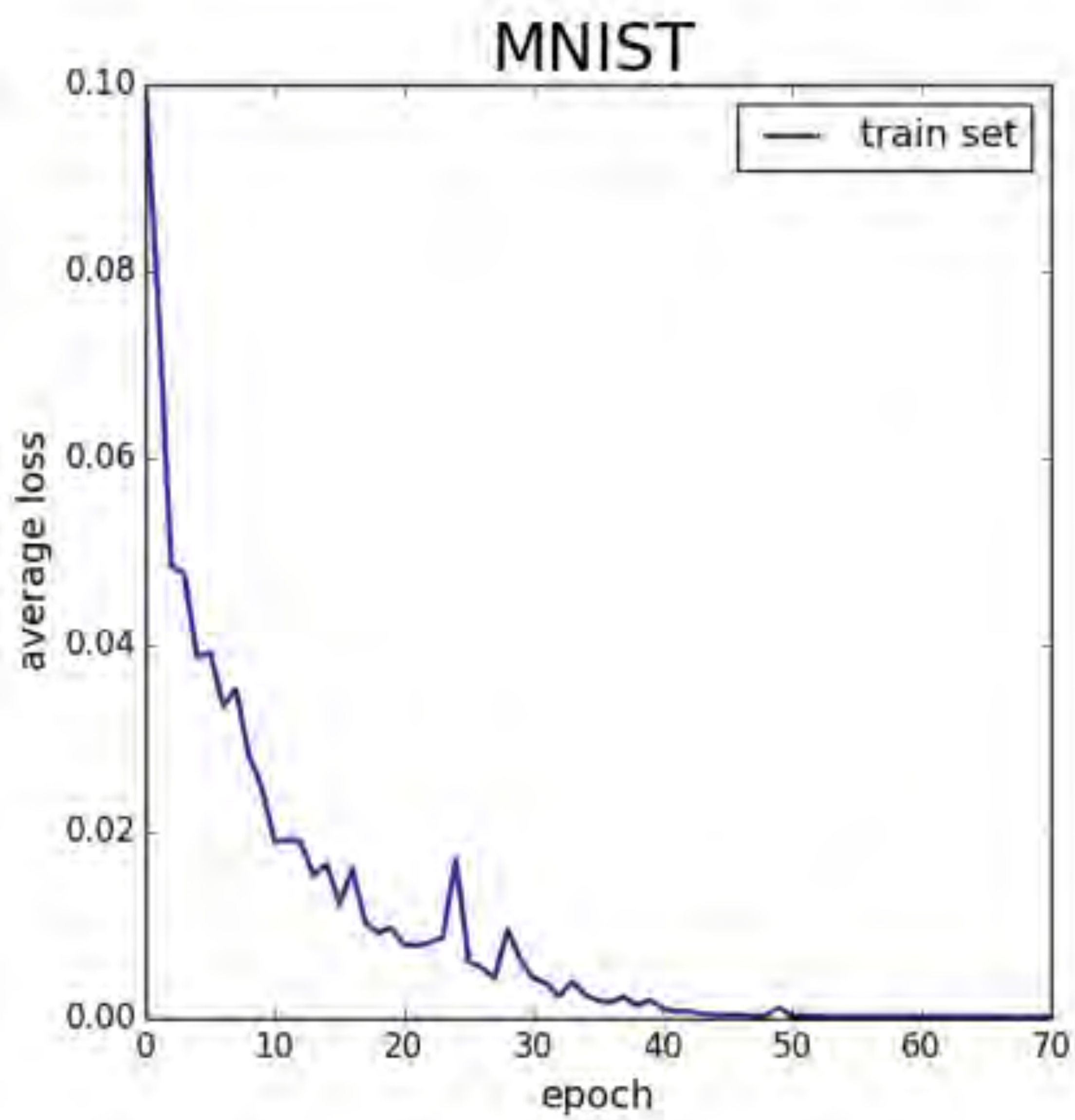
If there is a solution where all gradients vanish, constant stepwise converges linearly

Is there something special about your initialization?

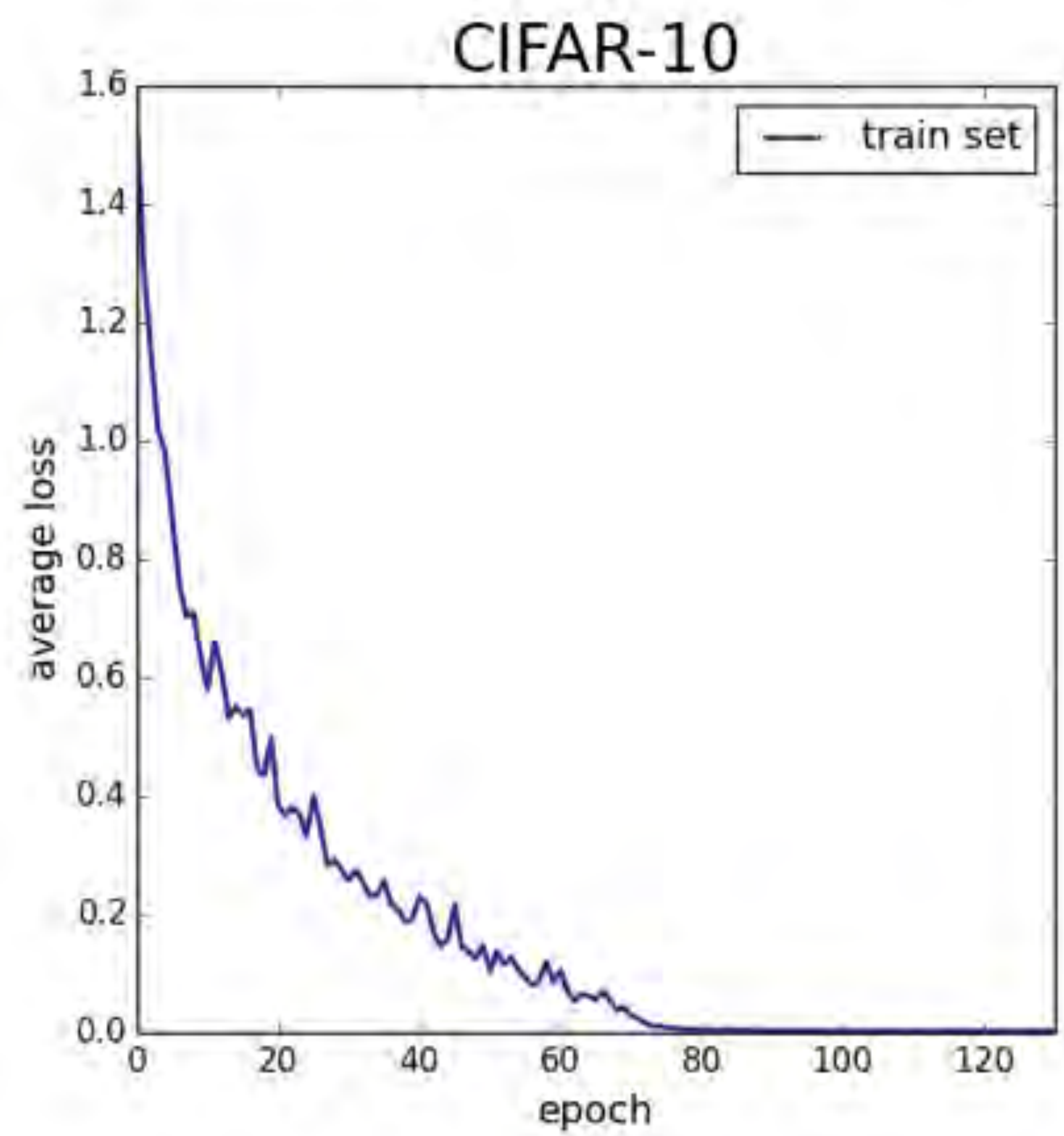
If you have more parameters than unknowns, why would you expect to converge to a saddle?



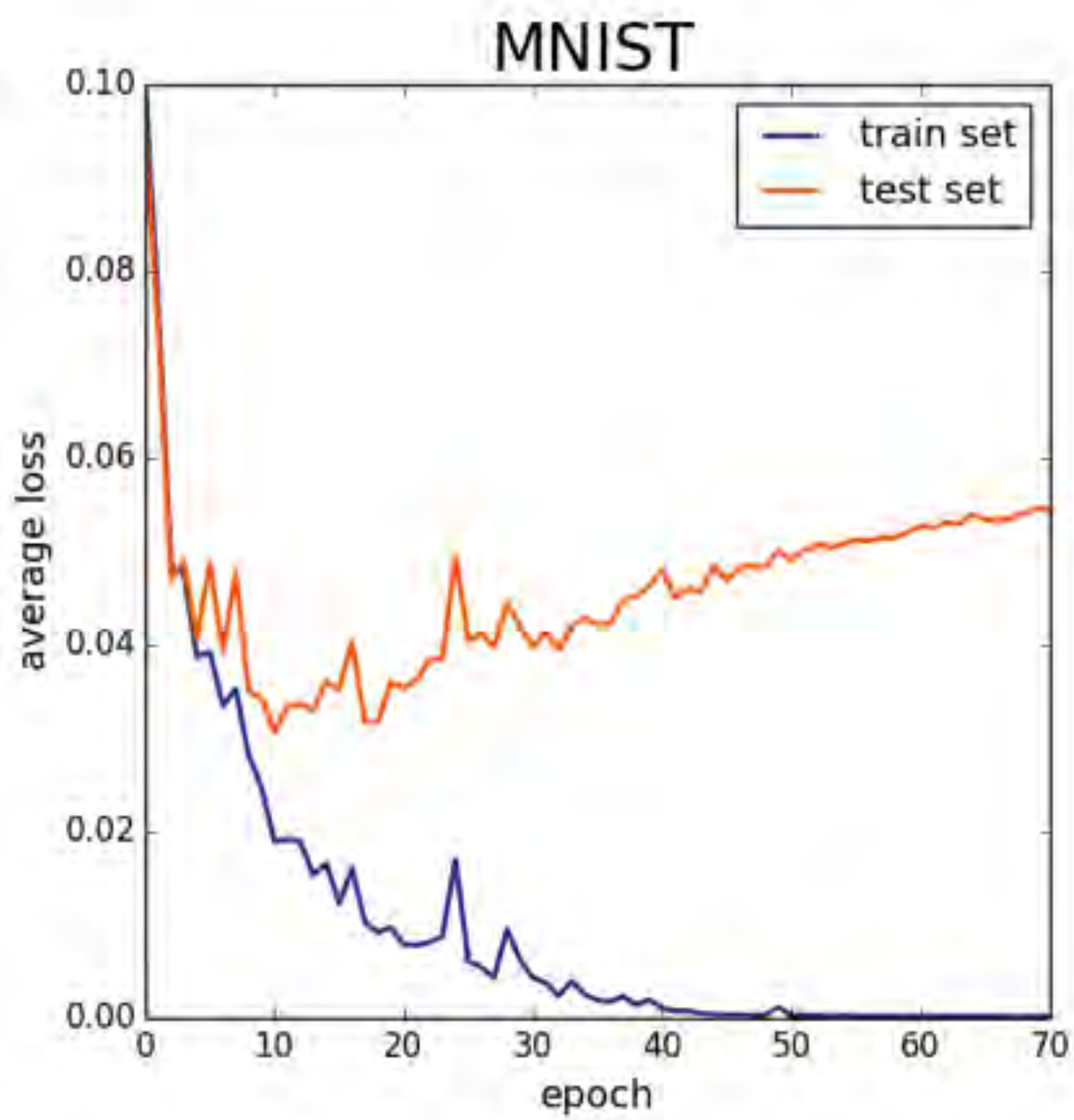
What does the test error look like?

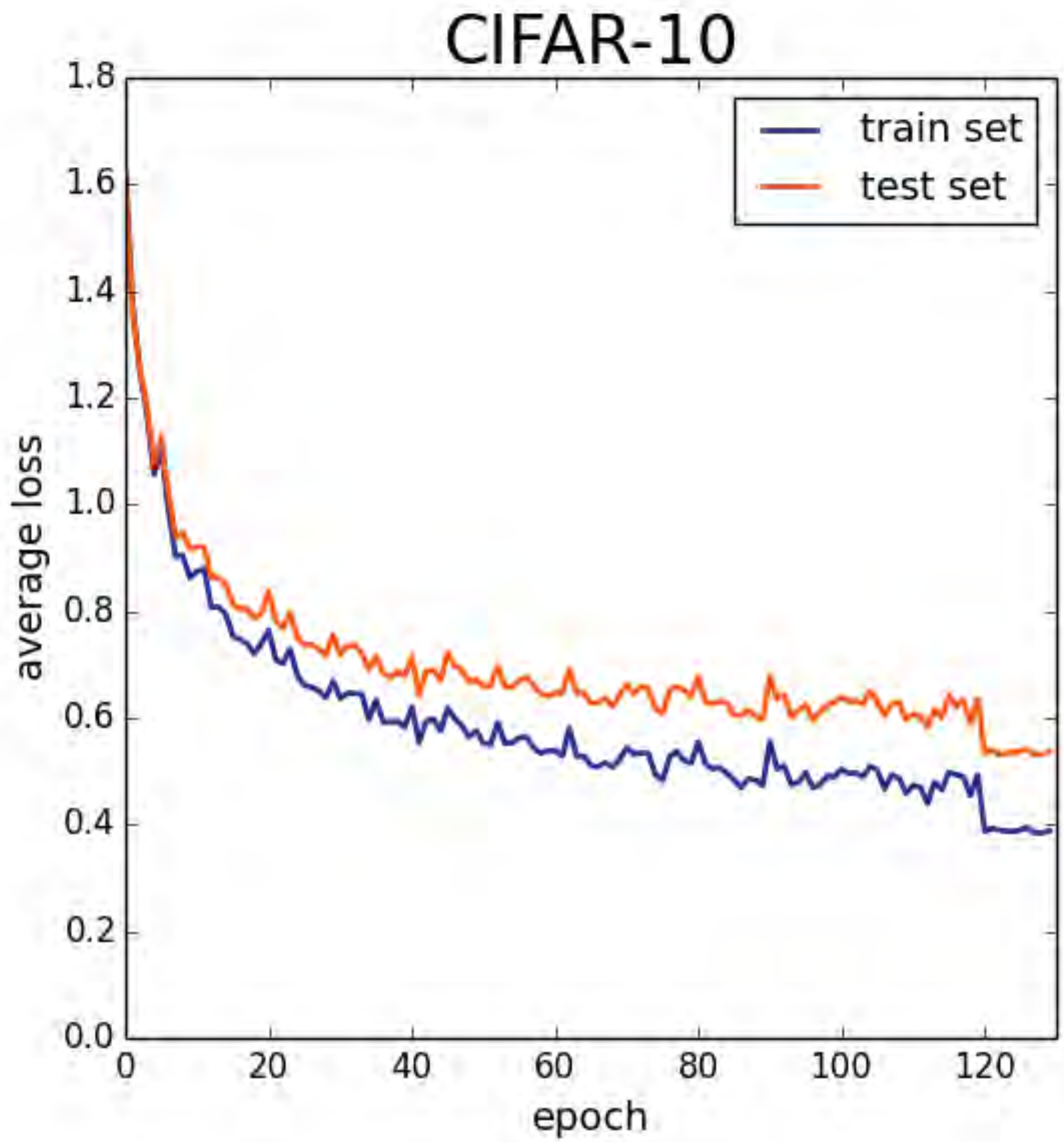


MNIST
train set
average loss
epoch
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
60
70



CIFAR-10
train set
average loss
epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120

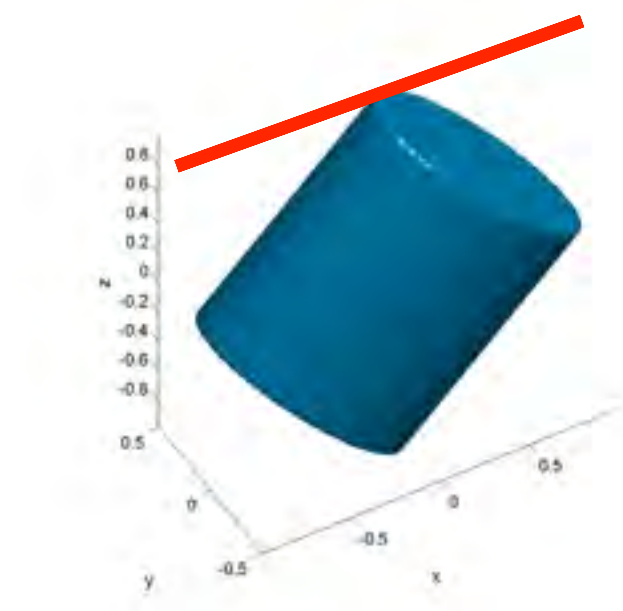

MNIST
CIFAR-10
train set
test set
average loss
epoch
0.00
0.02
0.04
0.06
0.08
0.10
0
10
20
30
40
50
60
70
0.0
0.5
1.0
1.5
2.0
80
100
120


CIFAR-10
train set
test set
average loss
epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120

Avoiding overfitting is hard.

- This is true in the convex case too!

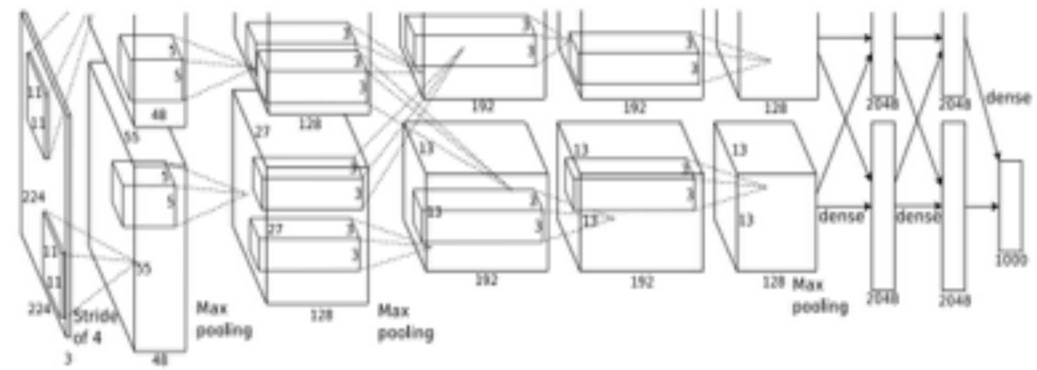
$$\text{minimize } \|y - \Phi x\|^2$$

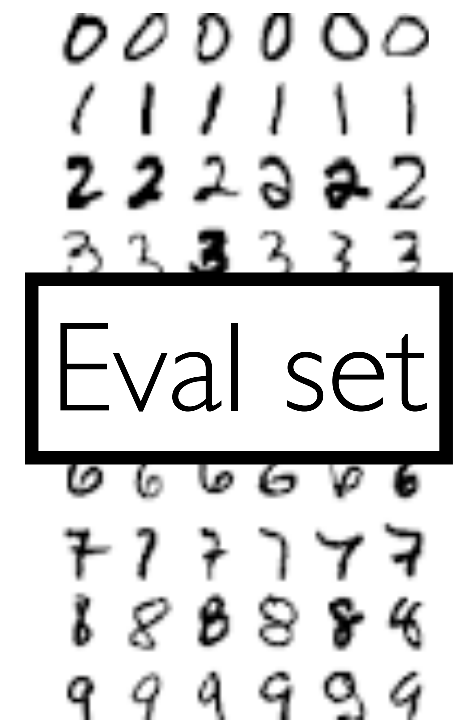


- $\mathbf{\Phi}$ n x p, n<p
- Infinite number of global minima. Which one should we pick?
- Regularize to leverage struture.

Sparsity | Rank | Smoothness | *Architecture*

Can/should we directly minimize overfitting?





$\widehat{f}$

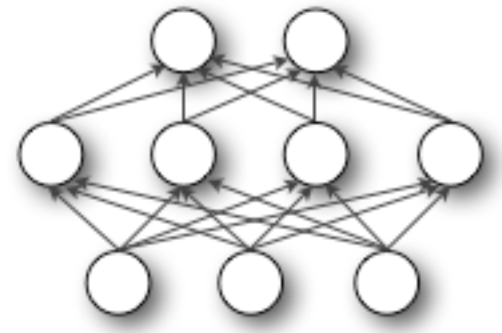
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

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

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

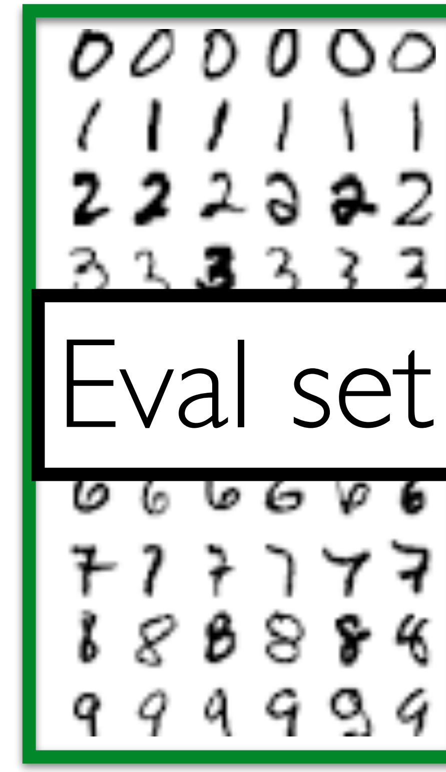
hidden nodes $N_{hid} \in [10^{1}, 10^{3}]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

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

Eval-loss

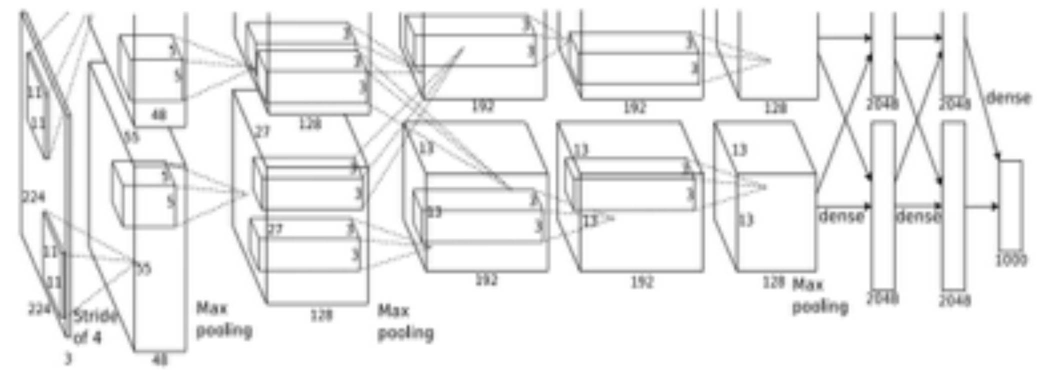
$\boxed{0.0577}$

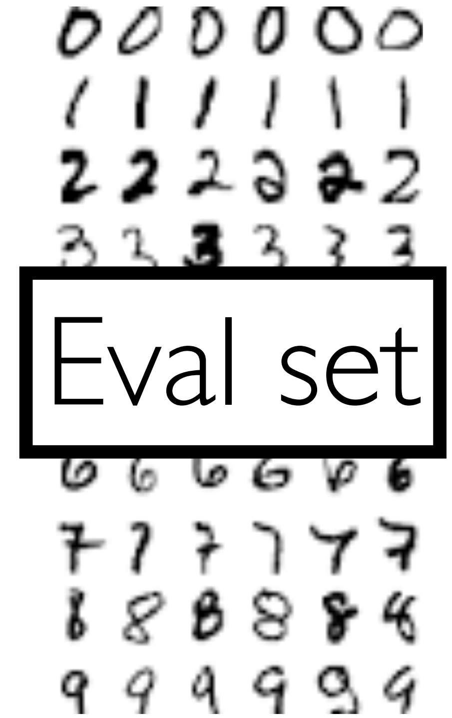
hyperparameters

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

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

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

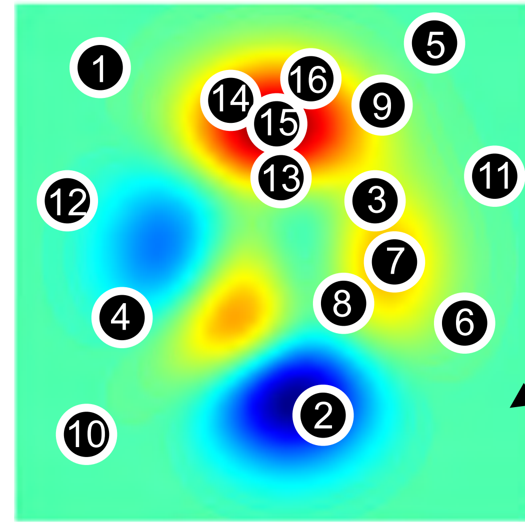




Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	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 choose hyperparameters to train and evaluate?

Bayesian Optimization



Hyperparameters *adaptively* chosen

Very popular for hyperparameter tuning

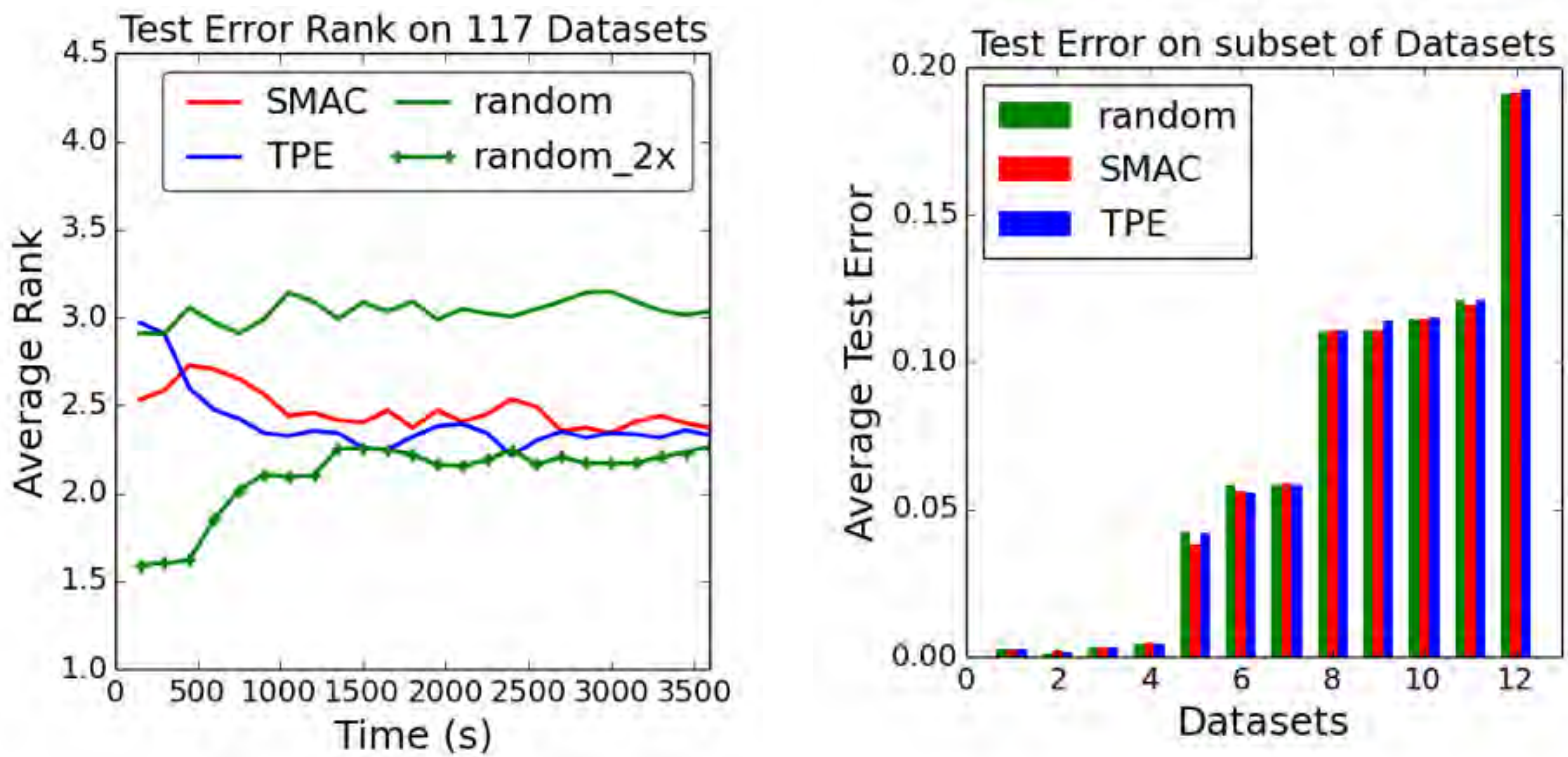
Recent abstracts on arXiv:

"Bayesian optimization has become a **successful tool for hyperparameter optimization** of machine learning algorithms, such as support vector machines or deep neural networks." arXiv:1605.07079

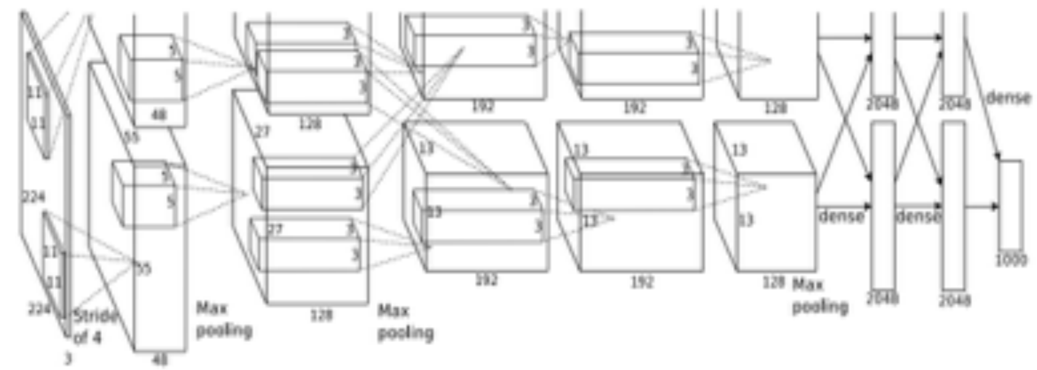
"Bayesian optimization is an **elegant solution to the hyperparameter optimization problem** in machine learning." arXiv:1605.06170

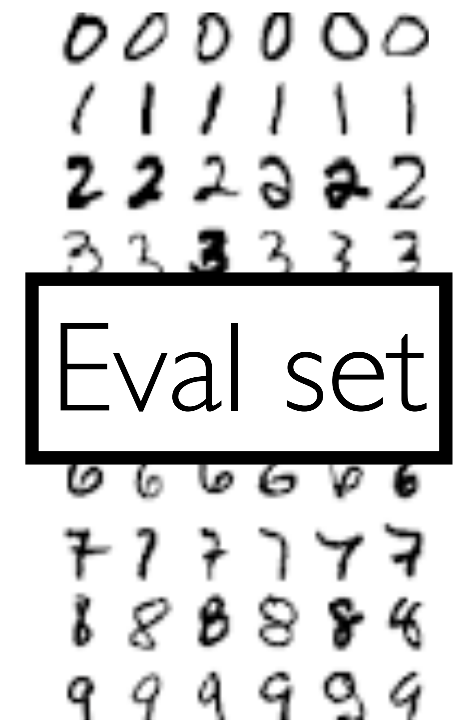
"Bayesian optimization provides a **principled way for searching optimal hyperparameters** for a single algorithm." arXiv:1602.06468

"finds global optima significantly **faster than previous batch Bayesian optimization algorithms … when tuning hyperparameters** of practical machine learning algorithms" arXiv:1606.04414

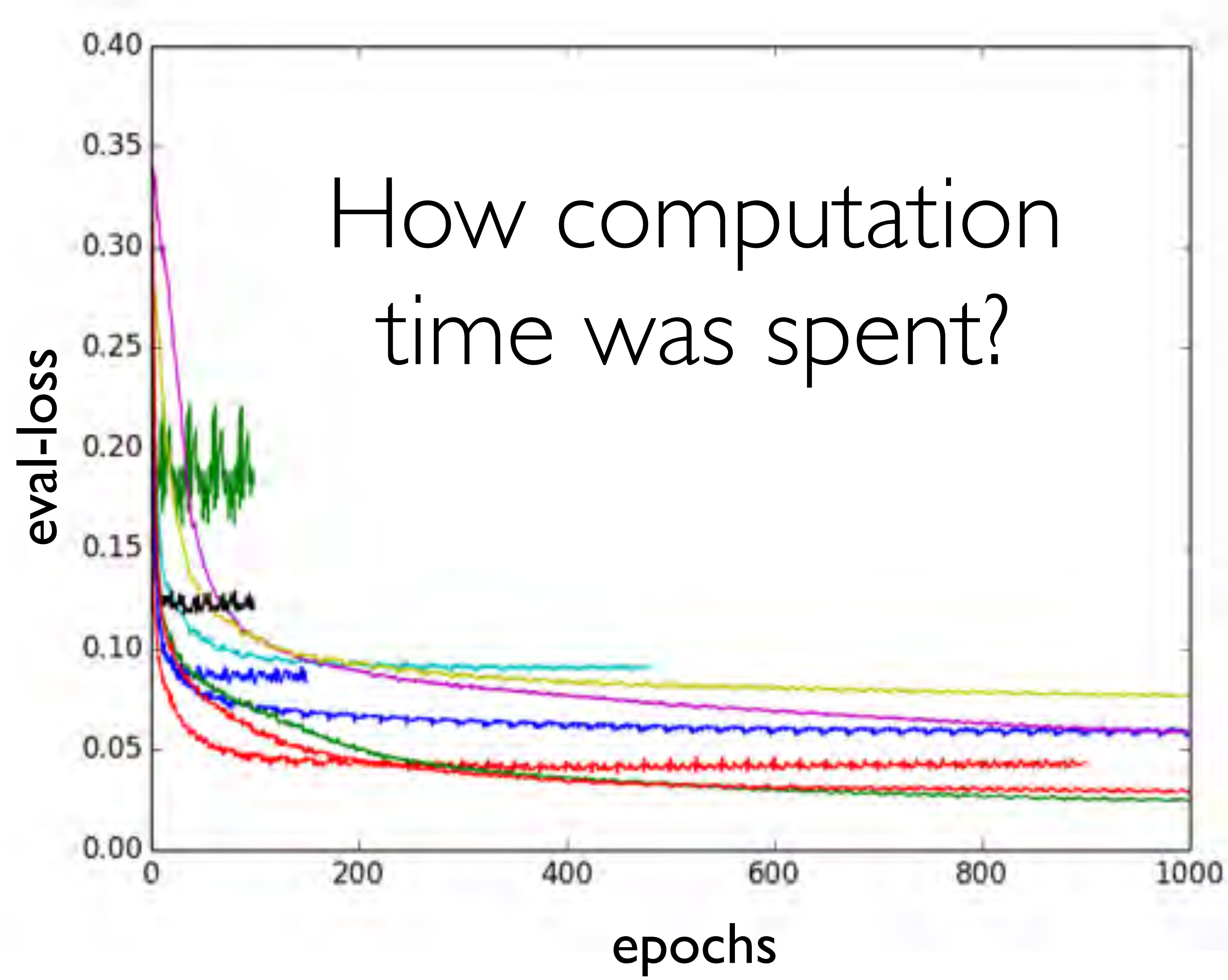


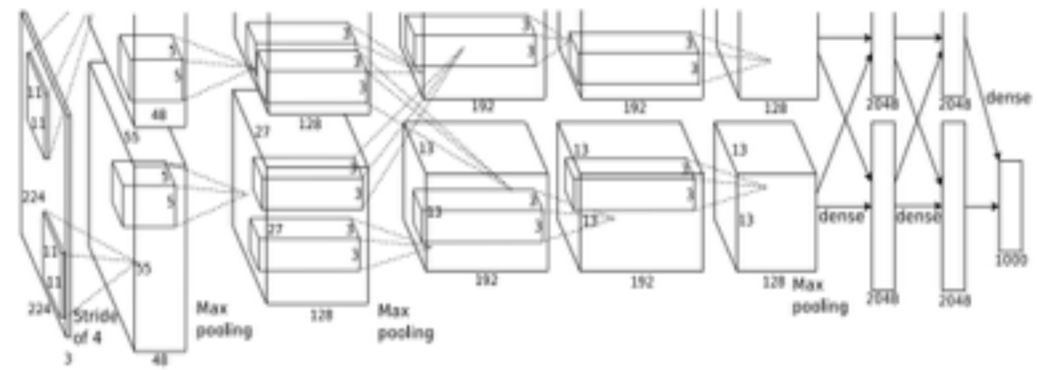
Test Error Rank on 117 Datasets
SMAC
random
TPE
random_2x
Average Rank
Time (s)
Test Error on subset of Datasets
random
SMAC
TPE
Average Test Error
Datasets

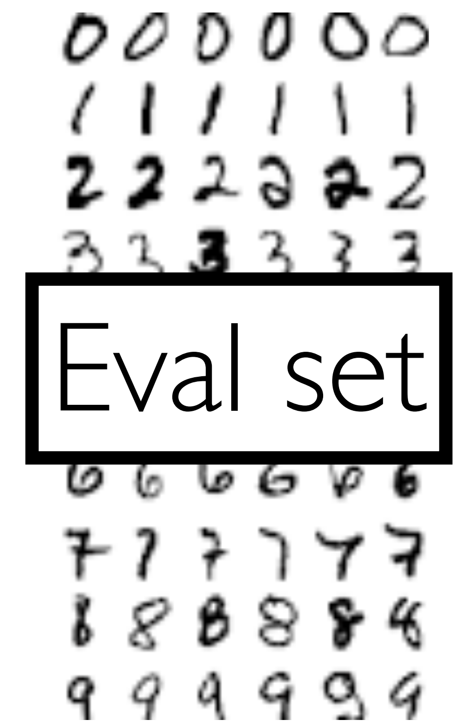




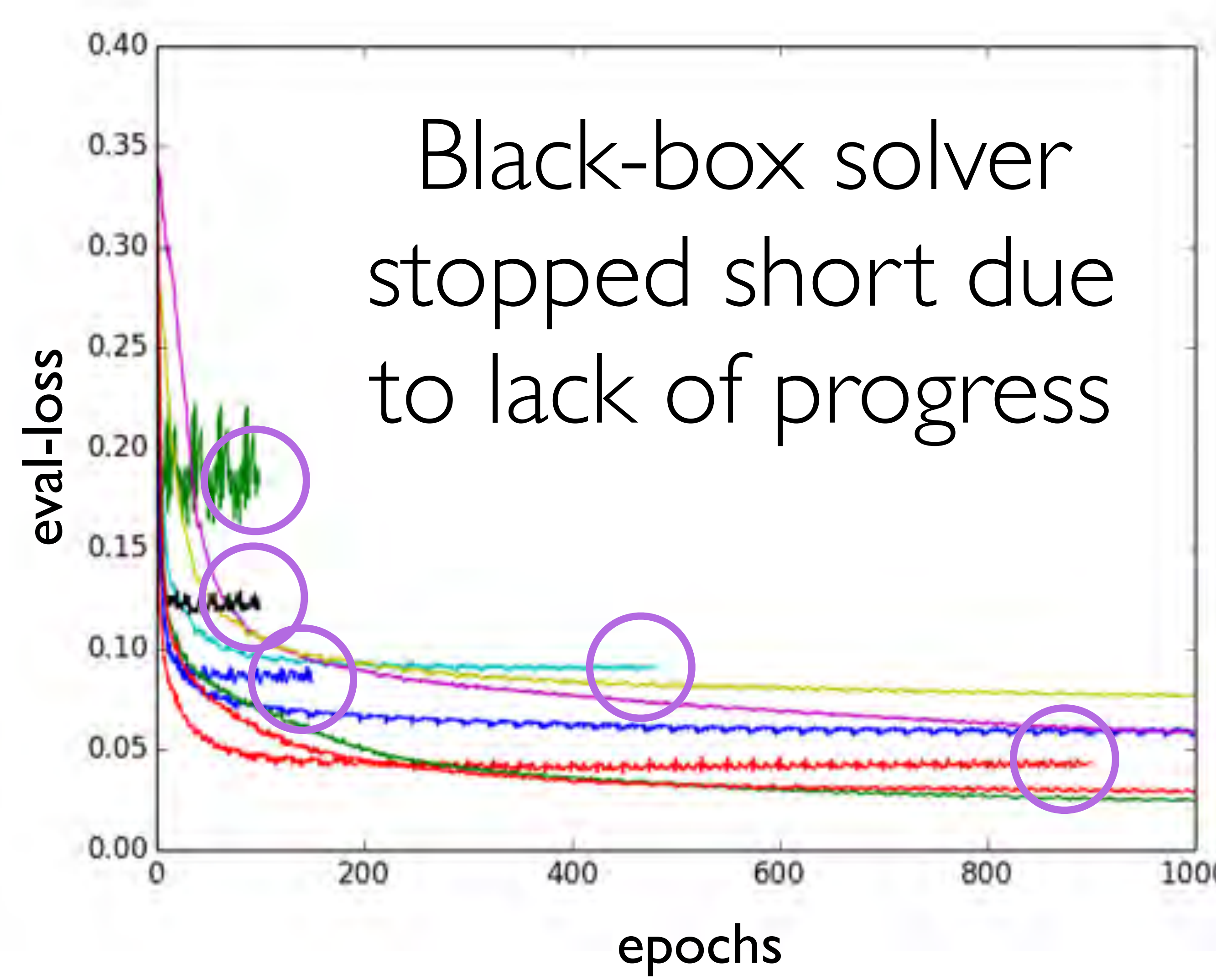
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4},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

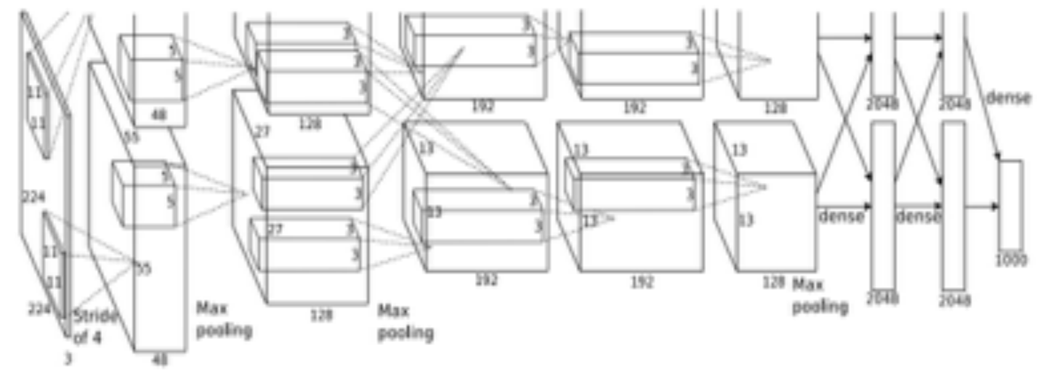


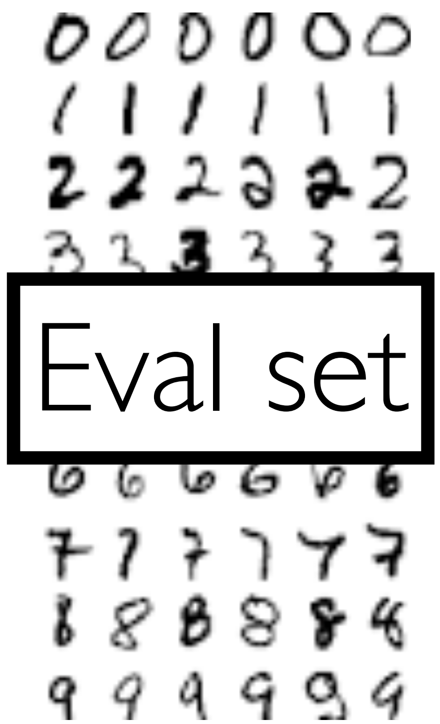




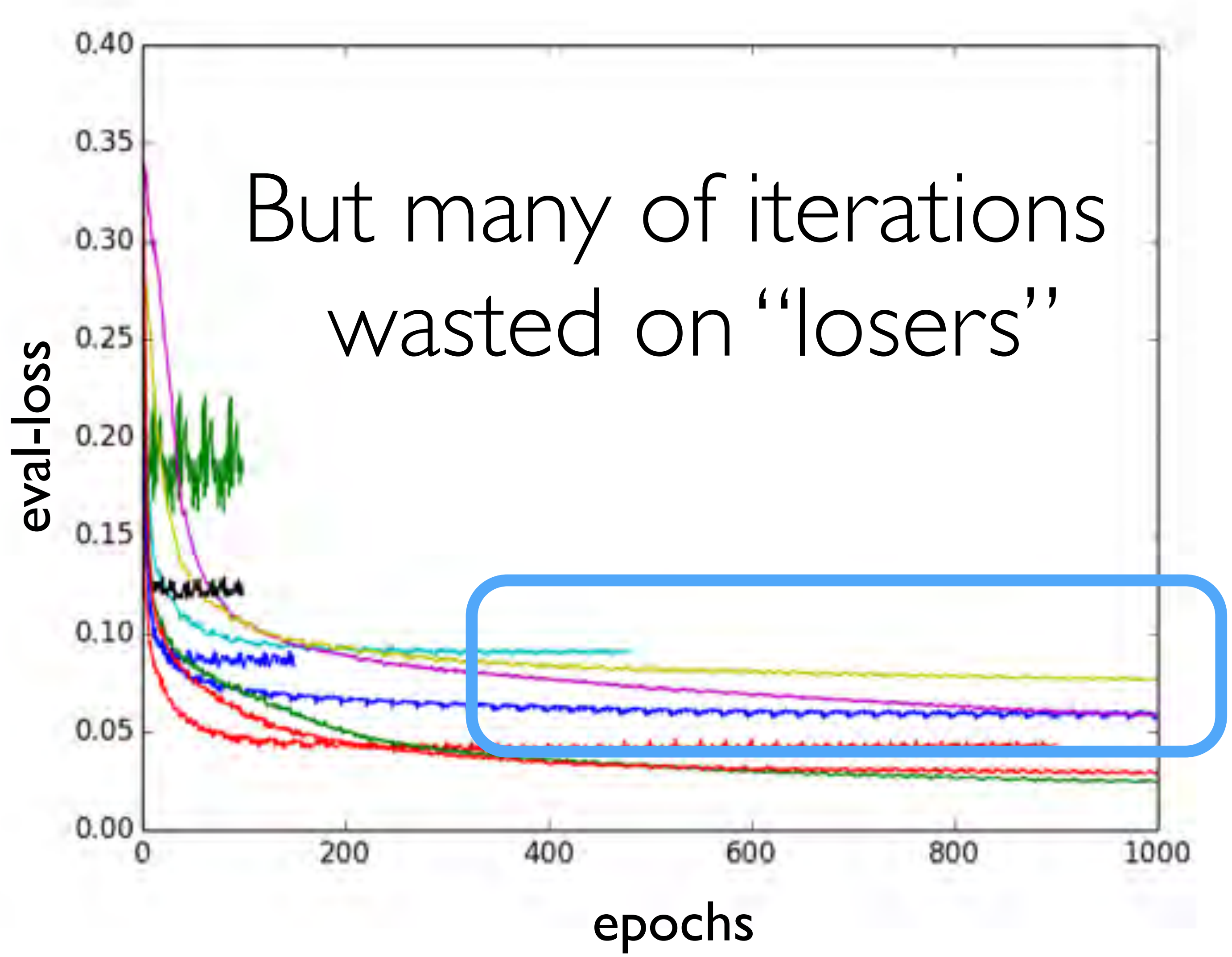
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4},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

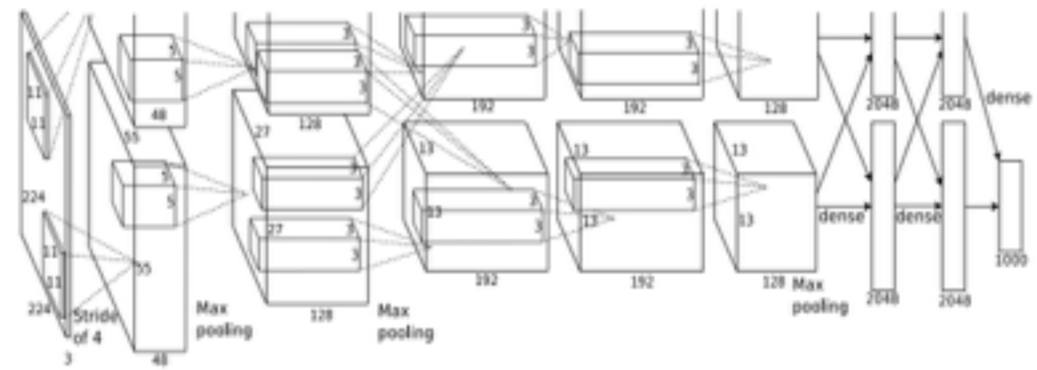


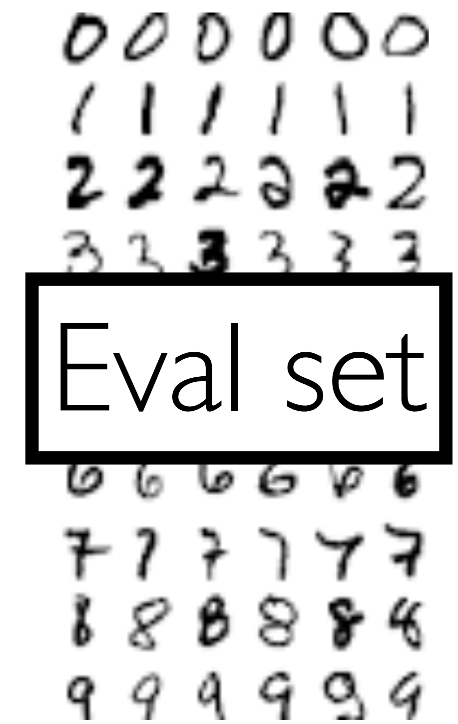




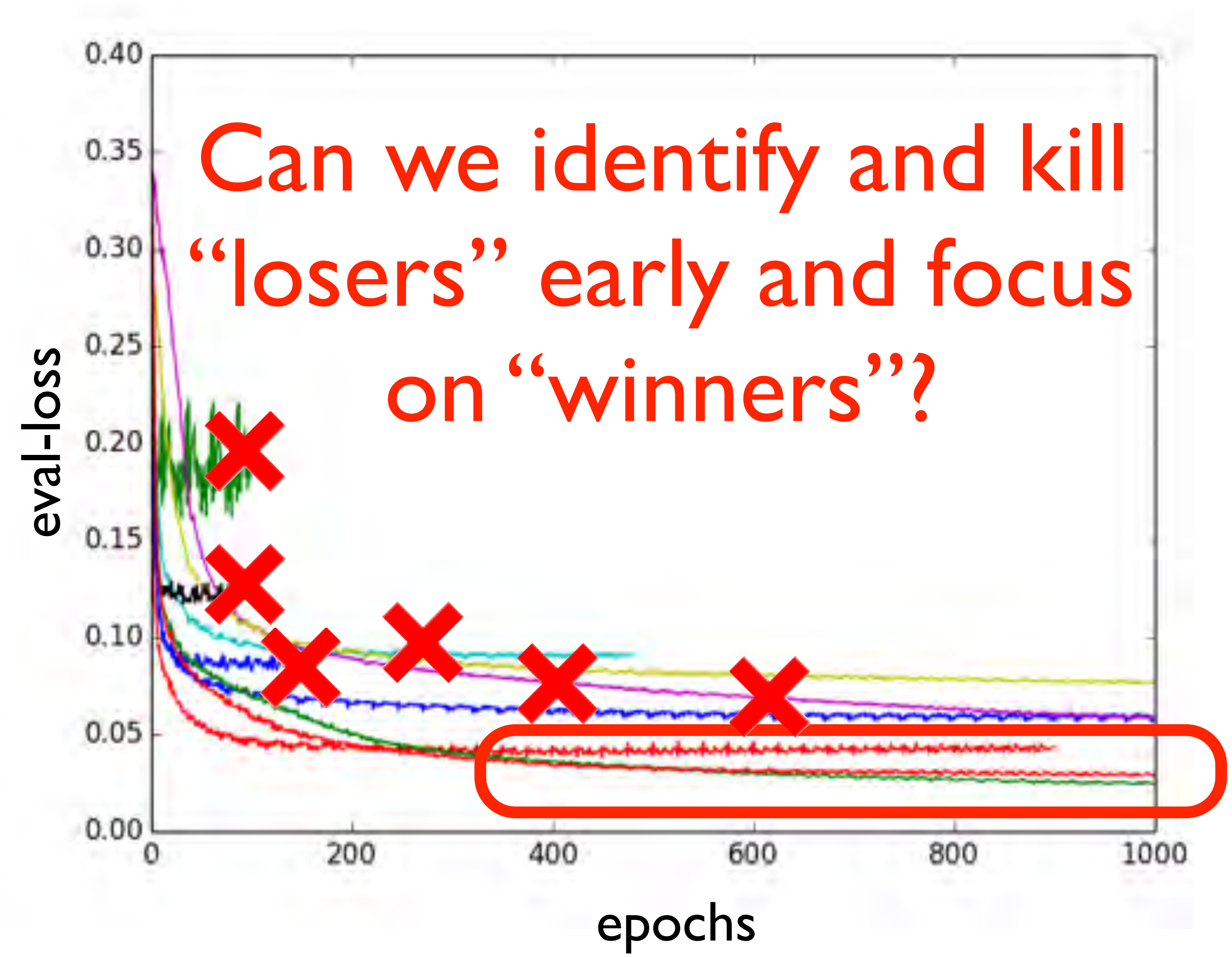
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4},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



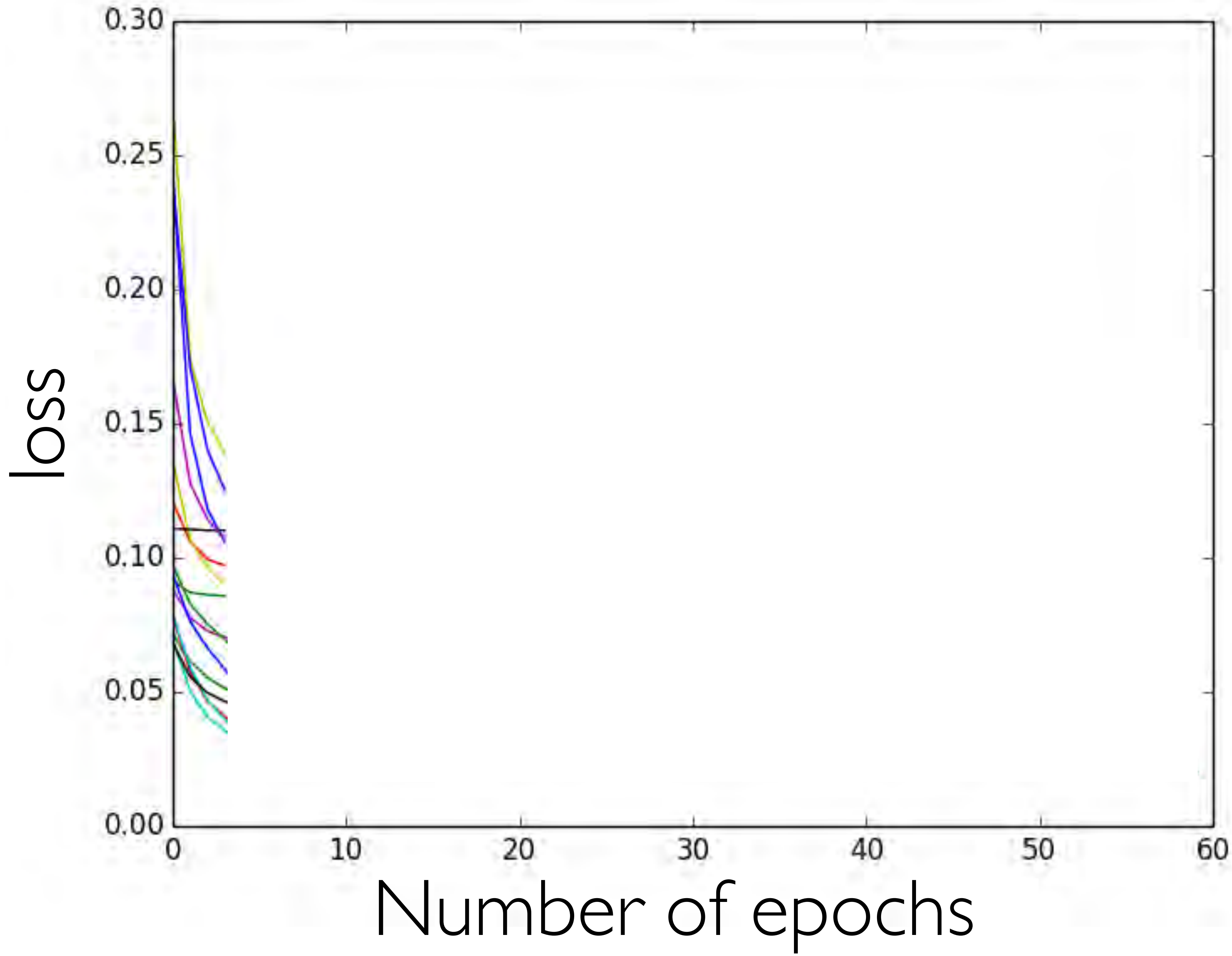




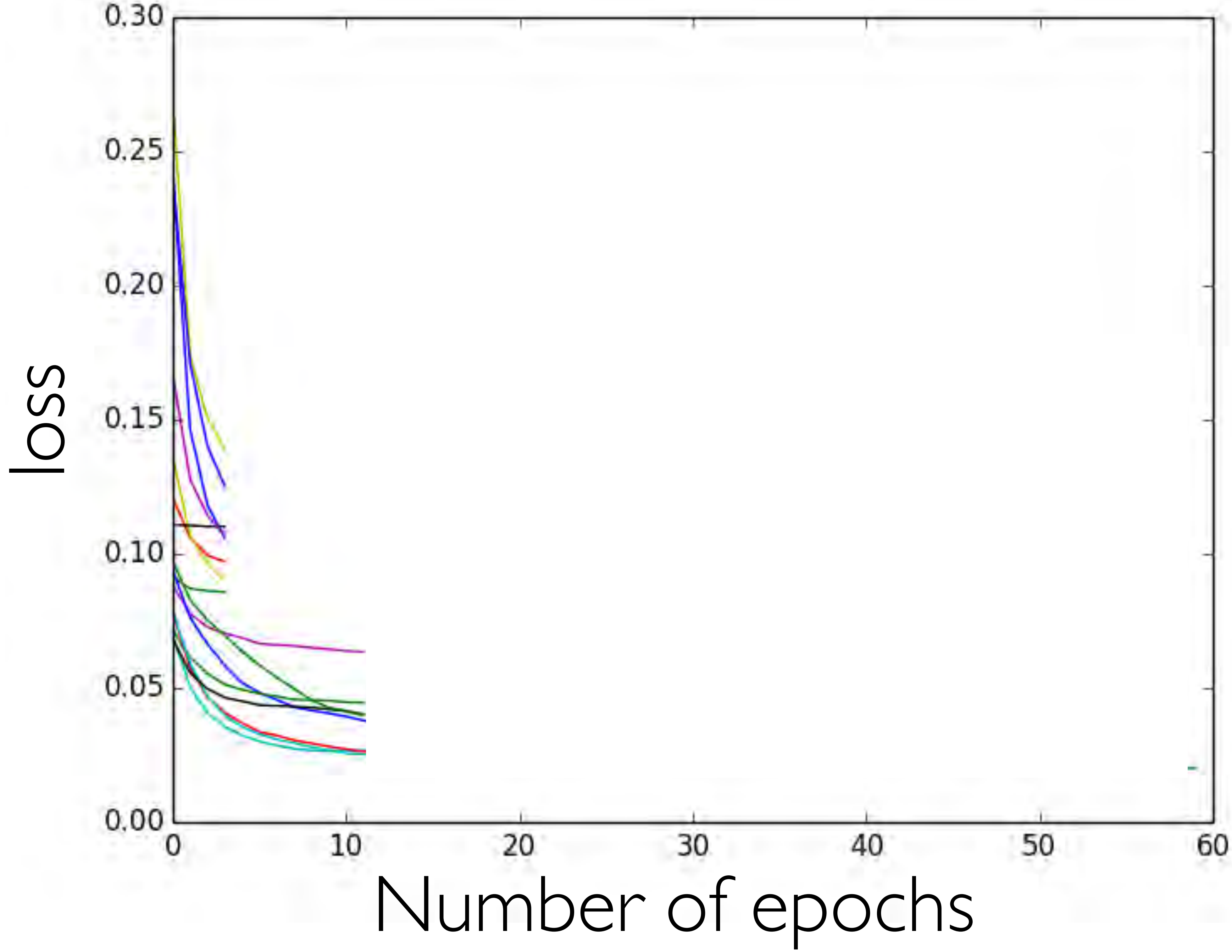
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4},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



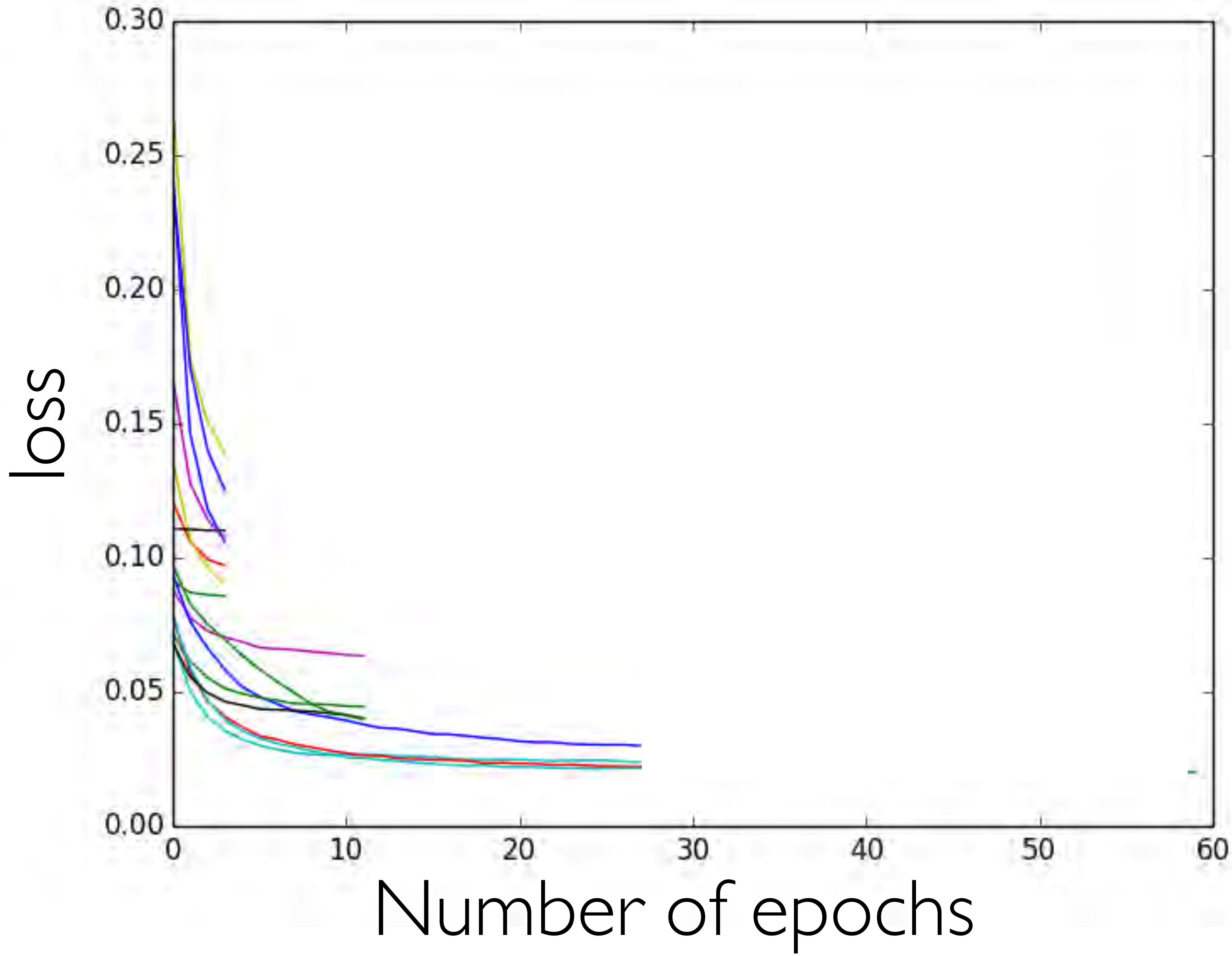
Sparks and Talwalkar heuristic…



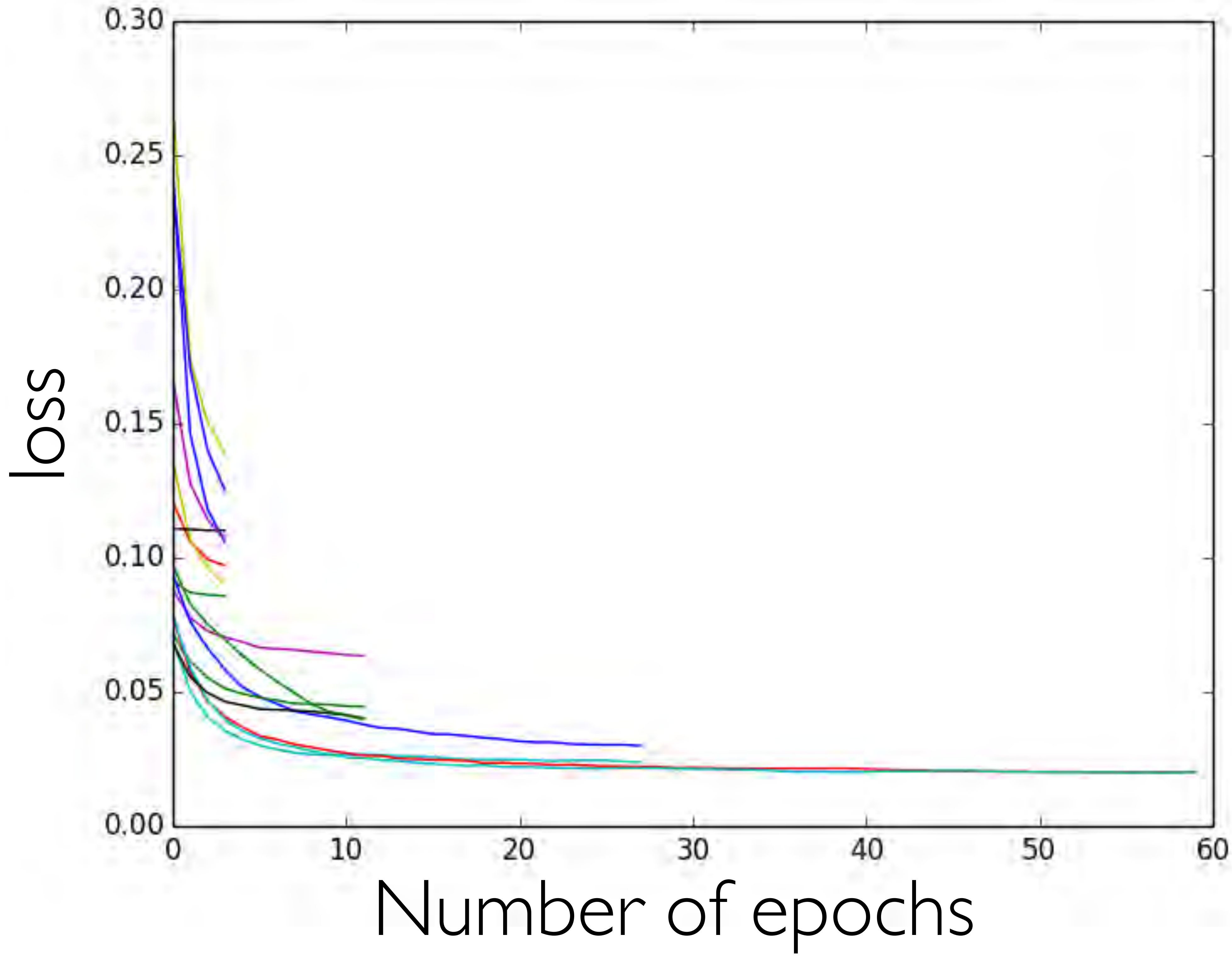
Sparks and Talwalkar heuristic…



Sparks and Talwalkar heuristic…



Sparks and Talwalkar heuristic…



HYPERBAND

Input: `max_iter`

for $s = \log_3(\texttt{max\_iter}), \ldots, 1, 0$:

    $n = 3^s (\log_\eta(\texttt{max\_iter}) + 1)/(s+1)$, $r = \texttt{max\_iter}\, 3^{-s}$

    batch = [get_hyperparameter_config() for $i = 1, \ldots, n$]

    for $i = 0, \ldots, \log_3(\texttt{max\_iter}/r)$:

        $n_i = n\, 3^{-i}$, $r_i = r\, 3^i$

        for params in batch:

            run_and_get_val_loss(config=params, iters= r_i)

        Throw out worst $2/3\ n_i$ configurations

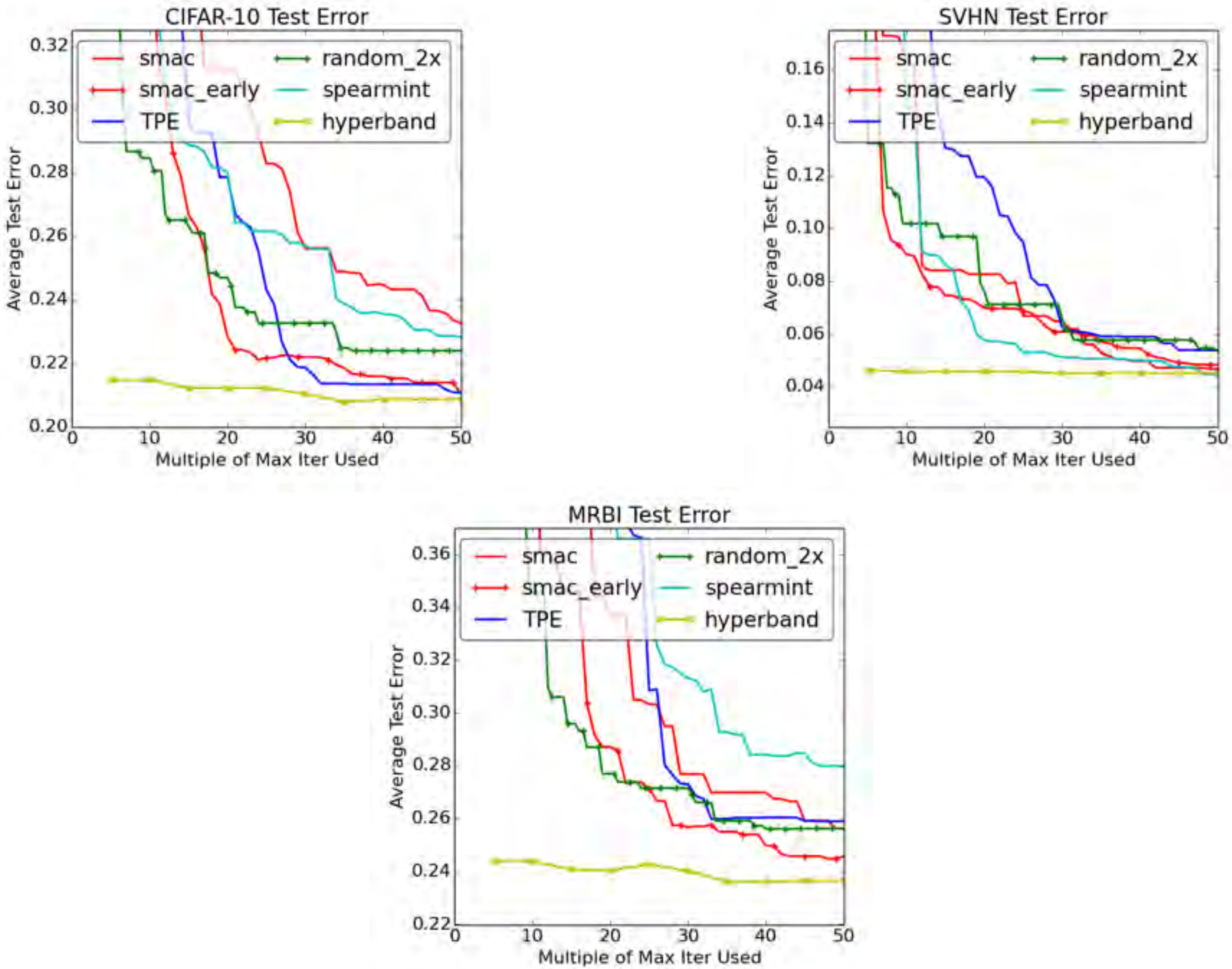
    Return remaining configs in batch

HYPERBAND

$$n > \frac{B}{(\log_3(\texttt{max\_iter}) + 1)^2}$$

UNIFORM ALLOCATION

$$n = \frac{B}{\texttt{max\_iter}}$$


CIFAR-10 Test Error
SVHN Test Error
MRBI Test Error
smac
smac_early
TPE
random_2x
spearmint
hyperband
Average Test Error
Multiple of Max Iter Used
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.36
0.34
0.22
0
10
20
30
40
50

"*The balance between theory and practice in nonlinear programming is particularly delicate, subjective, and problem dependent*" - D. Bertsekas

References

- argmin.net
- "Gradient Descent Converges to Minimizers." Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. COLT 2016, arXiv: 1602.04915
- Lecutre Notes on Approximation Algorithms and Semidefinite Programming. Bernd Gärtner and Jiří Matoušek. 2009. http://www.ti.inf.ethz.ch/ew/lehre/ApproxSDP09/index.html
- "A Nearly Tight Sum-of-Squares Lower Bound for the Planted Clique Problem." Boaz Barak, Samuel B. Hopkins, Jonathan Kelner, Pravesh K. Kothari, Ankur Moitra, Aaron Potechin. arXiv:1604.03084
- "No bad local minima: Data independent training error guarantees for multilayer neural networks." Daniel Soudry and Yair Carmon. arXiv: 1605.08361
- "Efficient Hyperparameter Optimization and Infinitely Many Armed Bandits." Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. arXiv:1603.06560