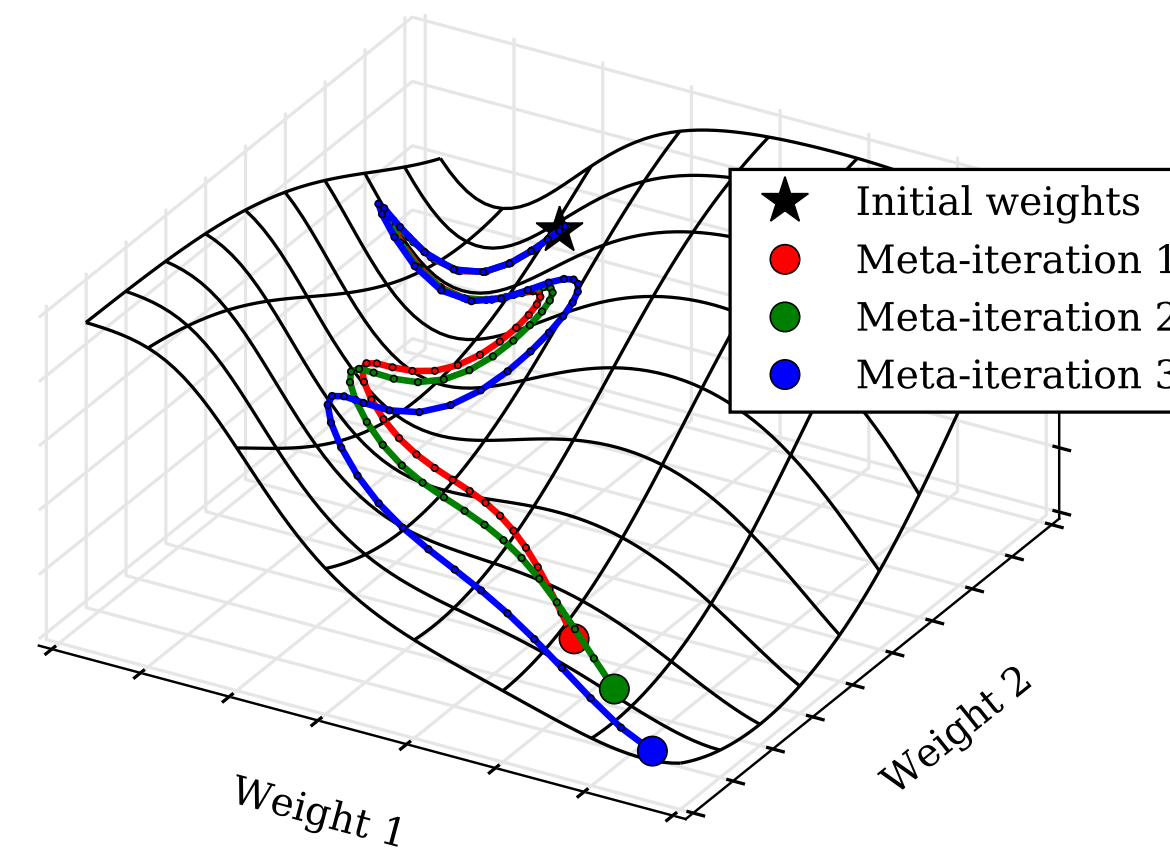


Motivation

- Hyperparameters are everywhere
- Gradient-free optimization fails in high dimensions
- Why not use gradients? Then we could optimize thousands of hyperparameters!

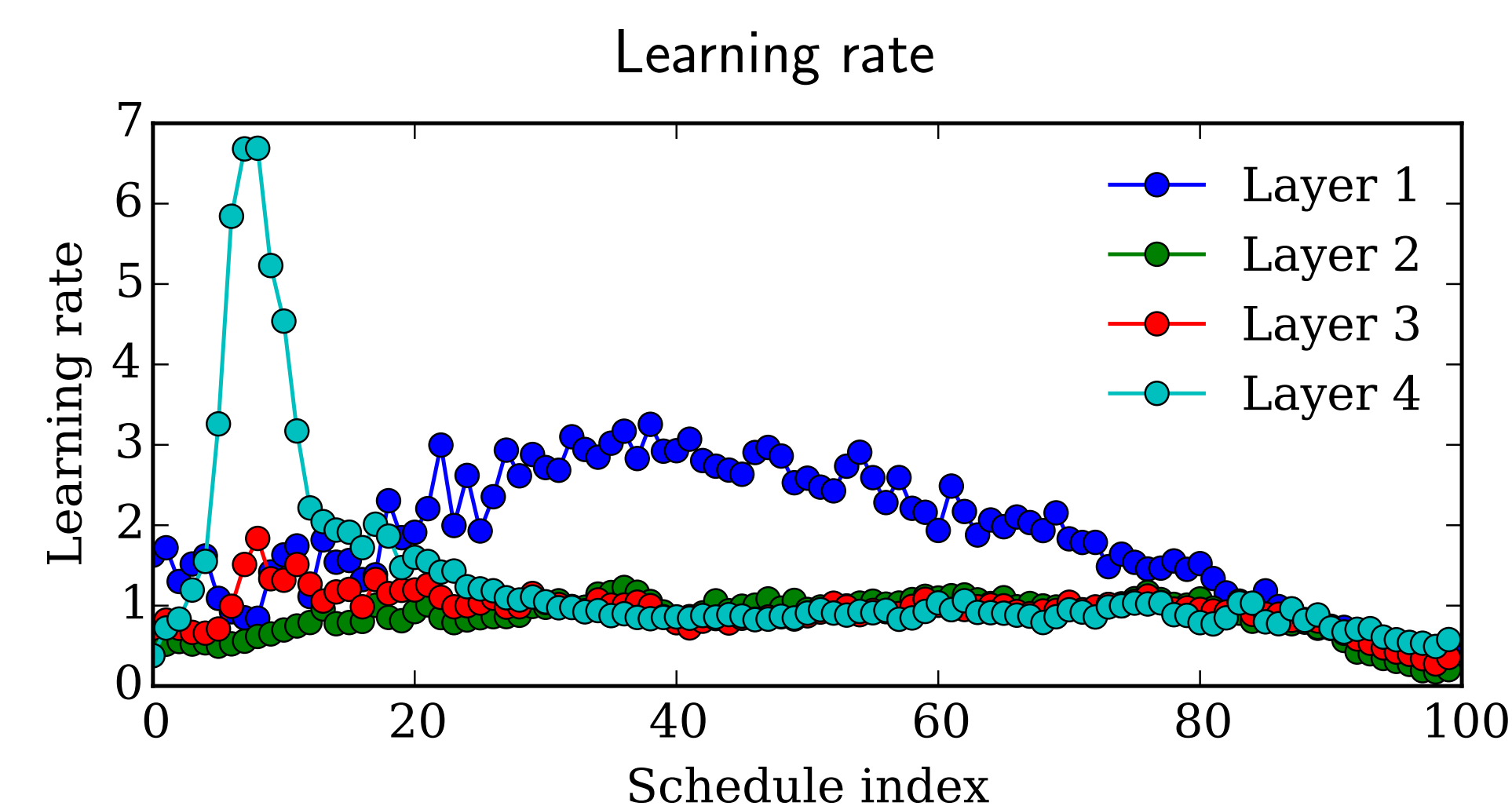
Stochastic gradient descent is a function

- We want to optimize validation loss
- Validation loss is a function of SGD
- SGD is a smooth function mapping (init weights, hypers) \rightarrow trained weights
- Let's compute its gradients!

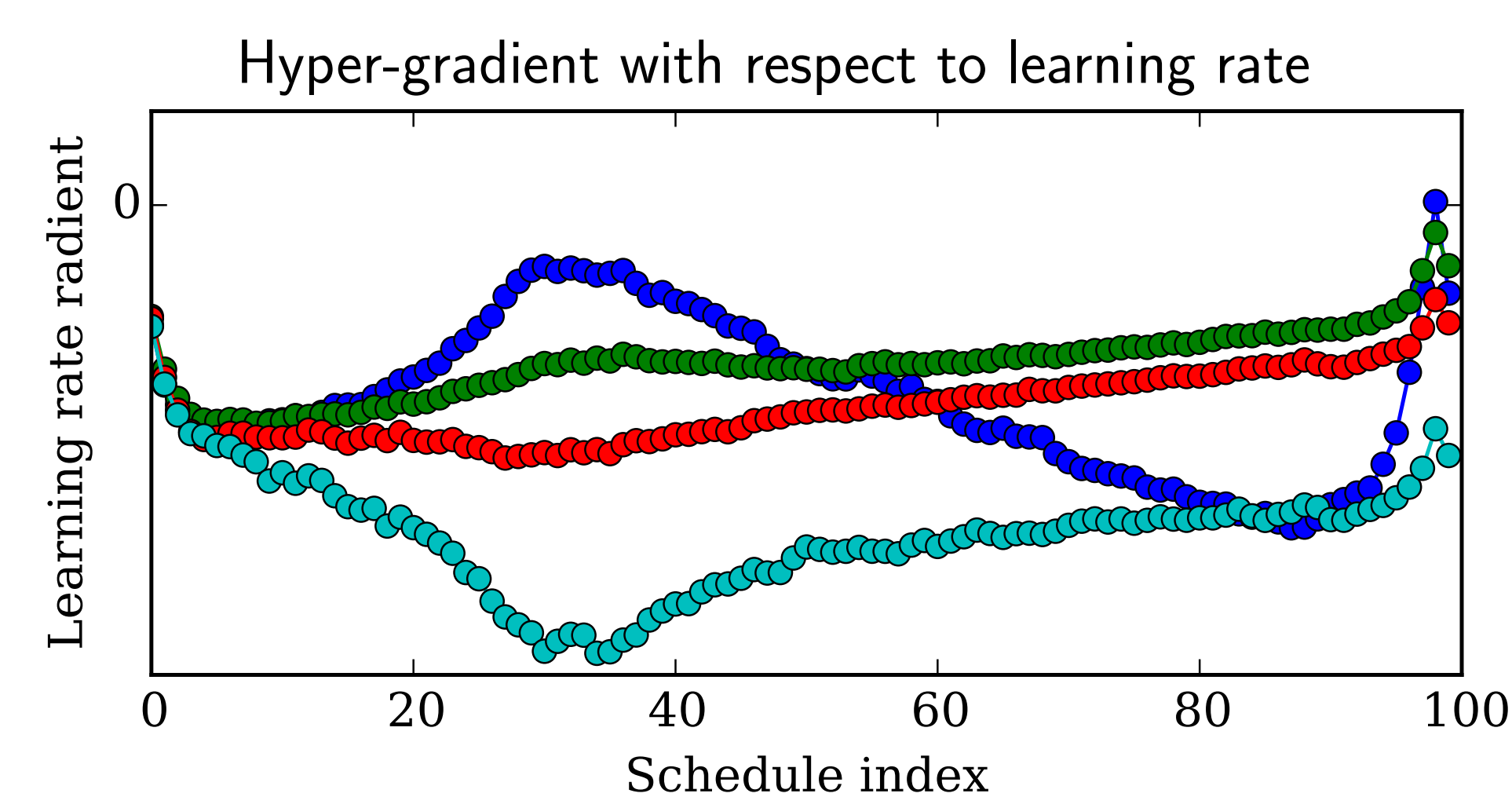


Example: Optimizing learning rate schedules

We can optimize learning rate schedules separately for each layer of a neural network and *each iteration* of training:

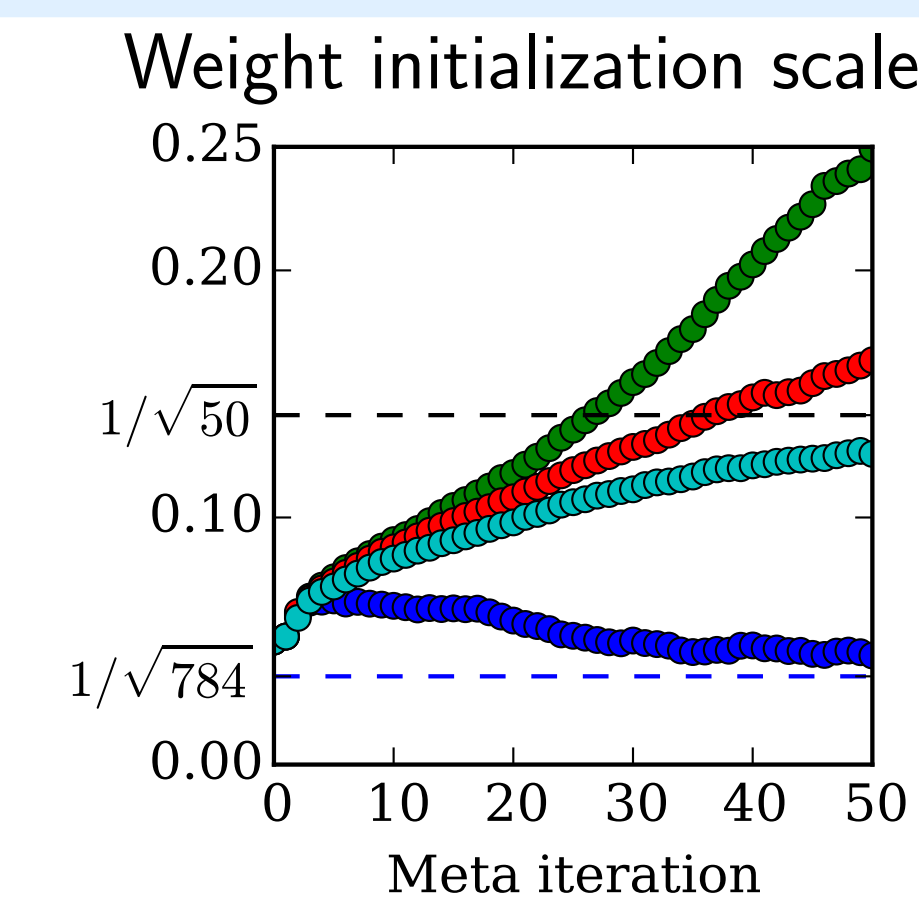


How did we optimize it? By SGD on top of SGD, using meta-gradients:



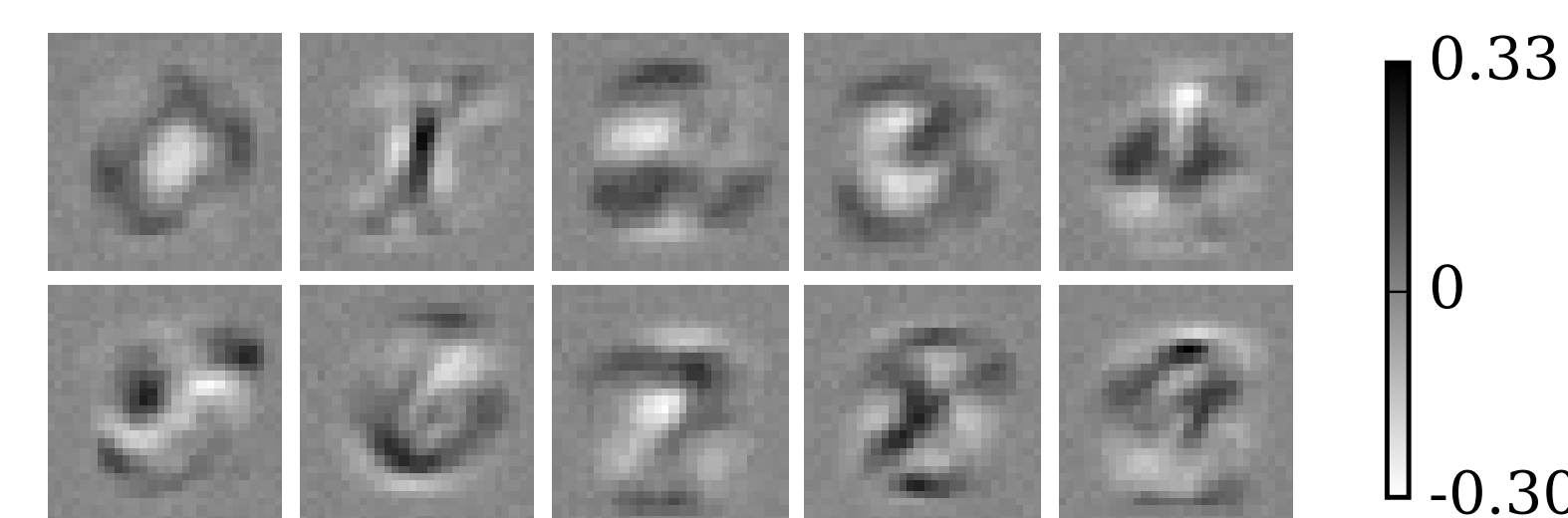
Optimizing initialization distributions

- We can optimize thousands of hyperparameters
- For instance, detailed weight initialization schemes
- Meta-learned values roughly match $1/\sqrt{N}$ heuristic



Optimizing training data

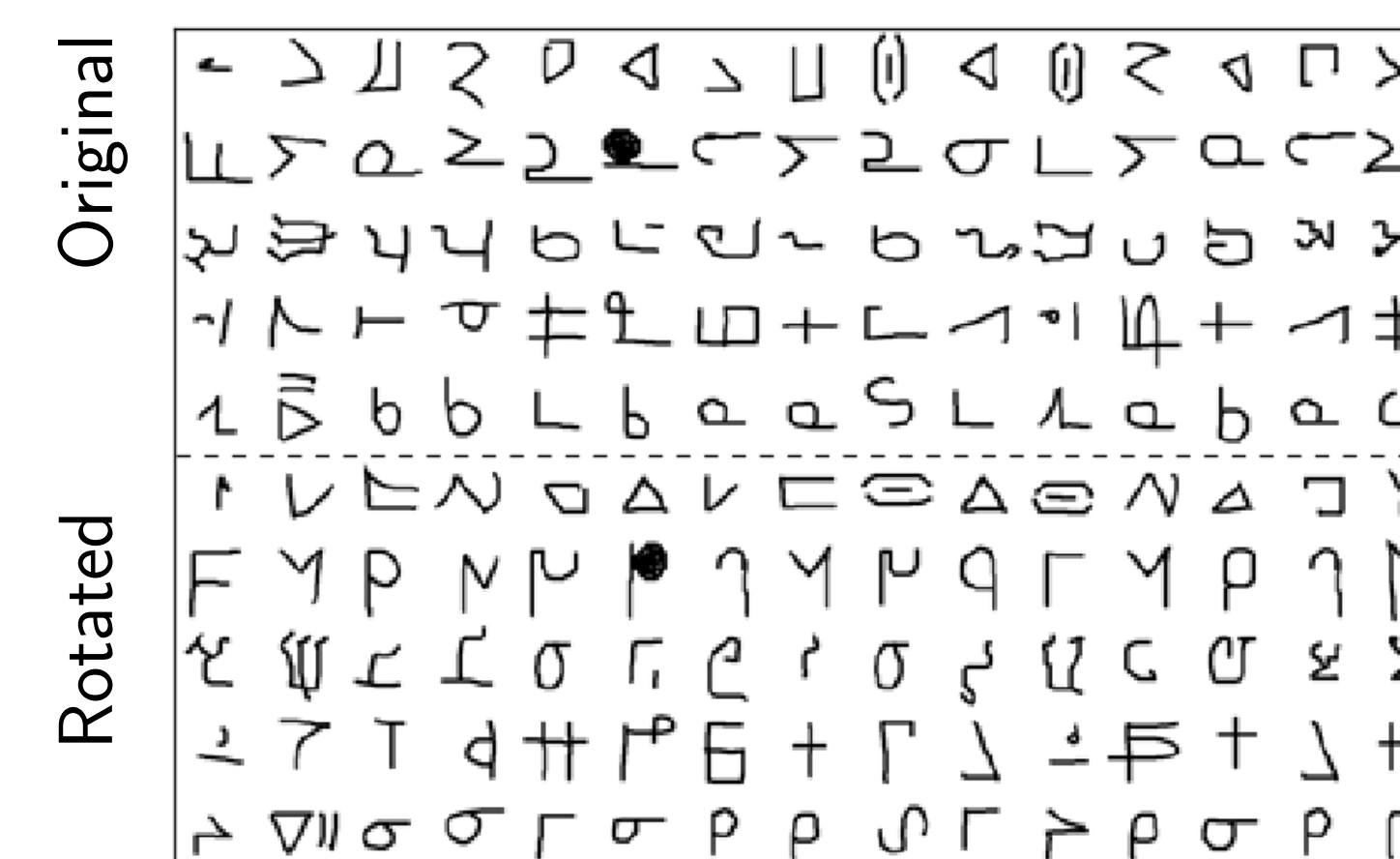
Synthetic MNIST training examples



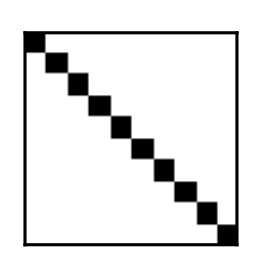
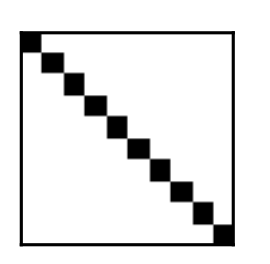
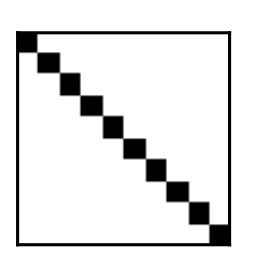
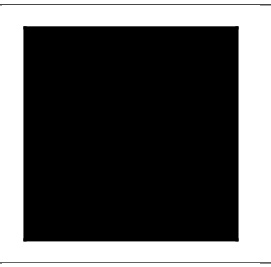
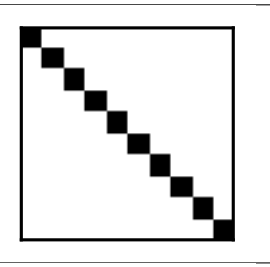
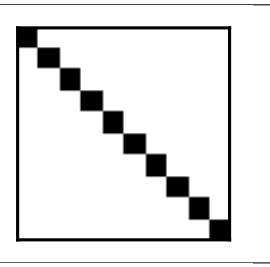
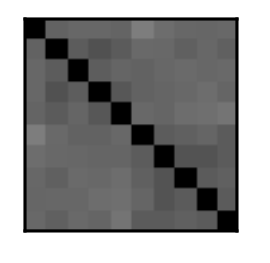
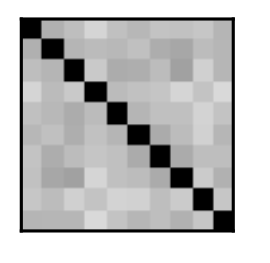
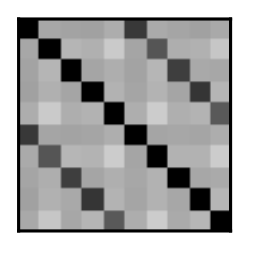
Training data can be viewed as just another hyperparameter. We meta-learned MNIST training examples starting from blank pixels, optimizing validation loss.

Optimizing network architecture

- Can optimize thousands of regularization params.
- Architecture can be controlled through regularization.
- We let the network choose which layers to share in a multi-task problem.

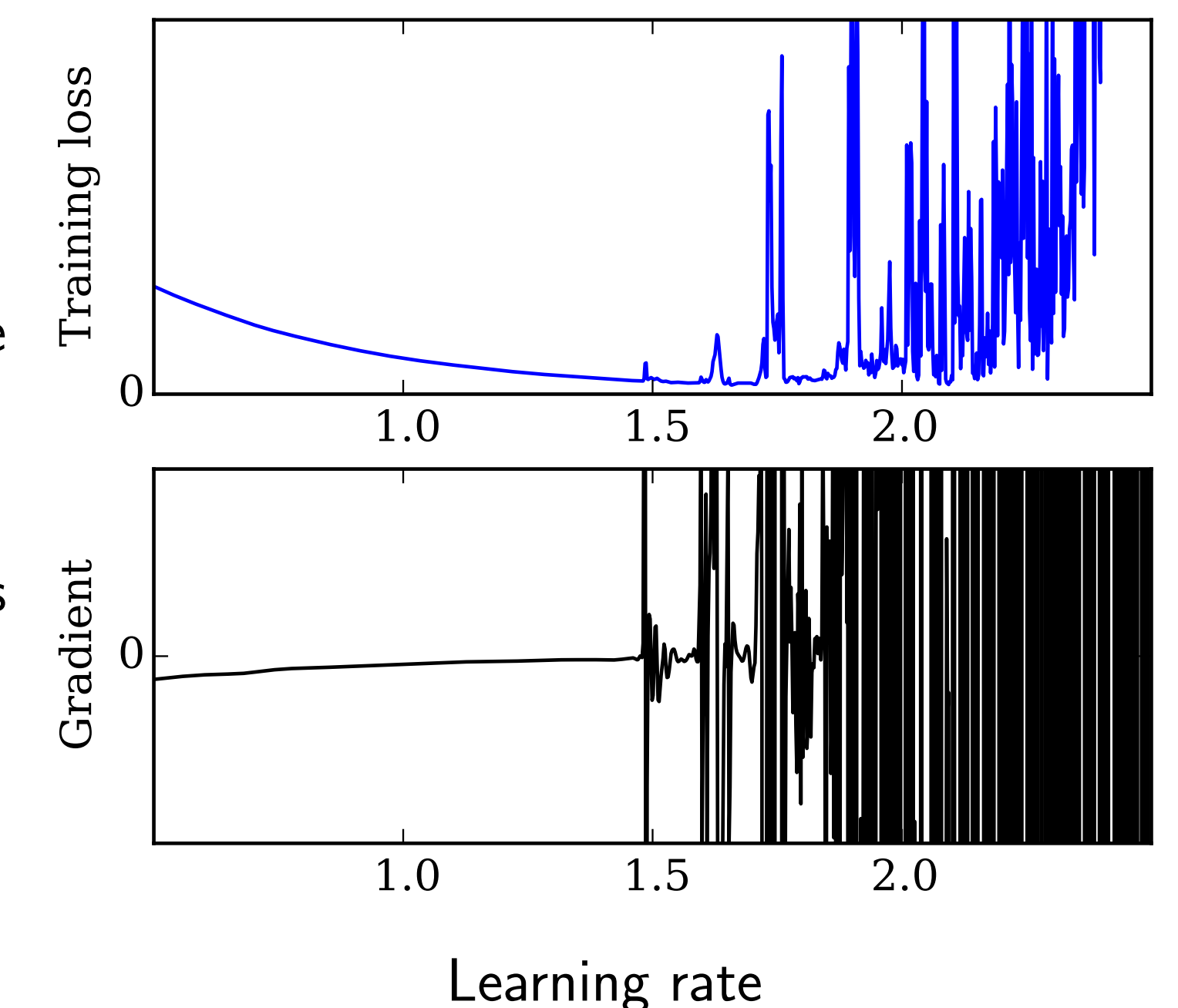


- Network learned to share weights between related tasks.
- Learned sharing works better than all-or-nothing.

	Input weights	Middle weights	Output weights	Train error	Test error
Separate networks				0.61	1.34
Tied weights				0.90	1.25
Learned sharing				0.60	1.13

Chaotic learning dynamics

- Limitation: Gradient can become chaotic
- a.k.a. exploding gradients
- Happens when learning rate is near the optimum.

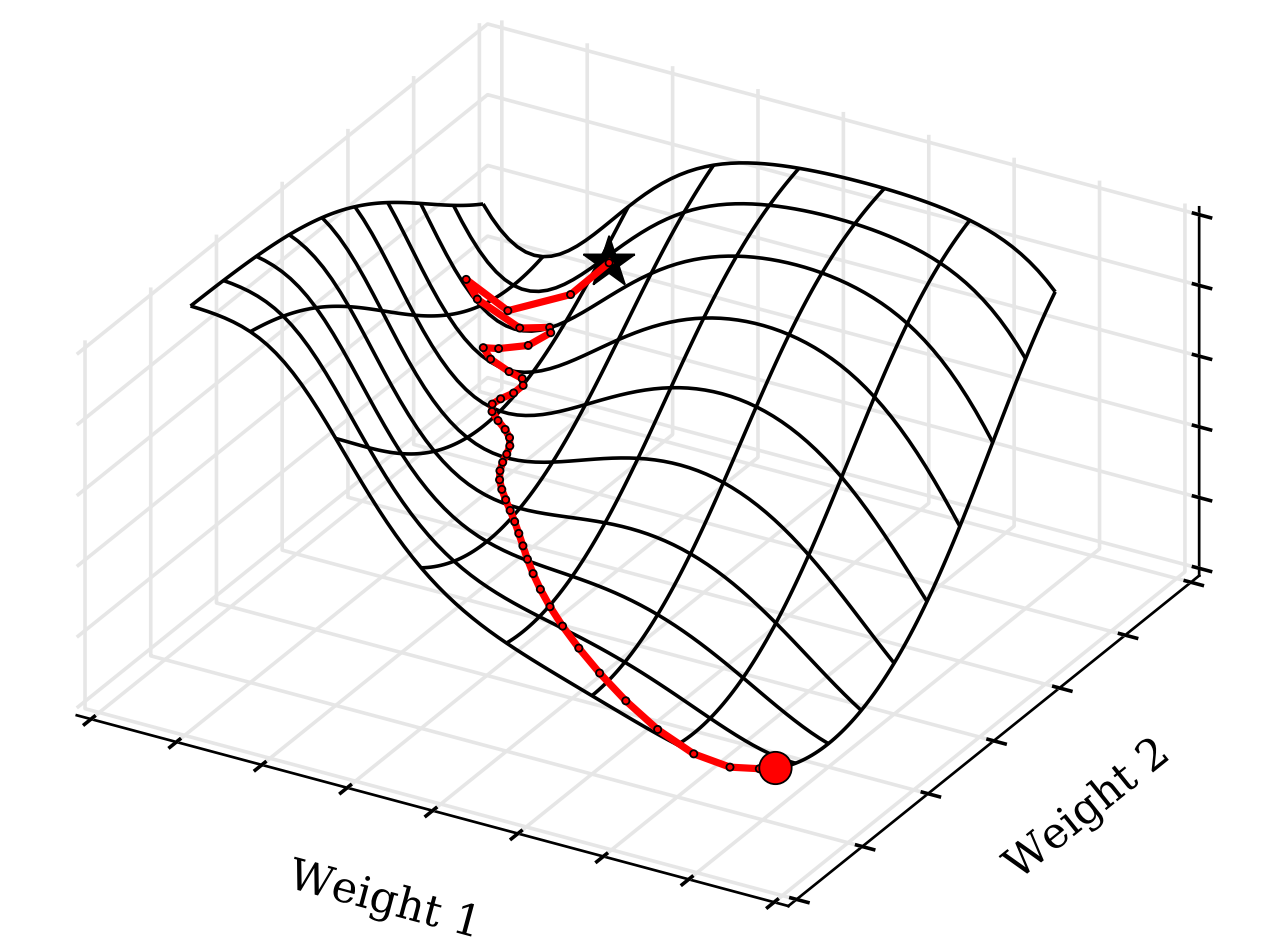


Stochastic gradient descent is reversible

To save memory during reverse-mode differentiation, we run SGD in reverse.

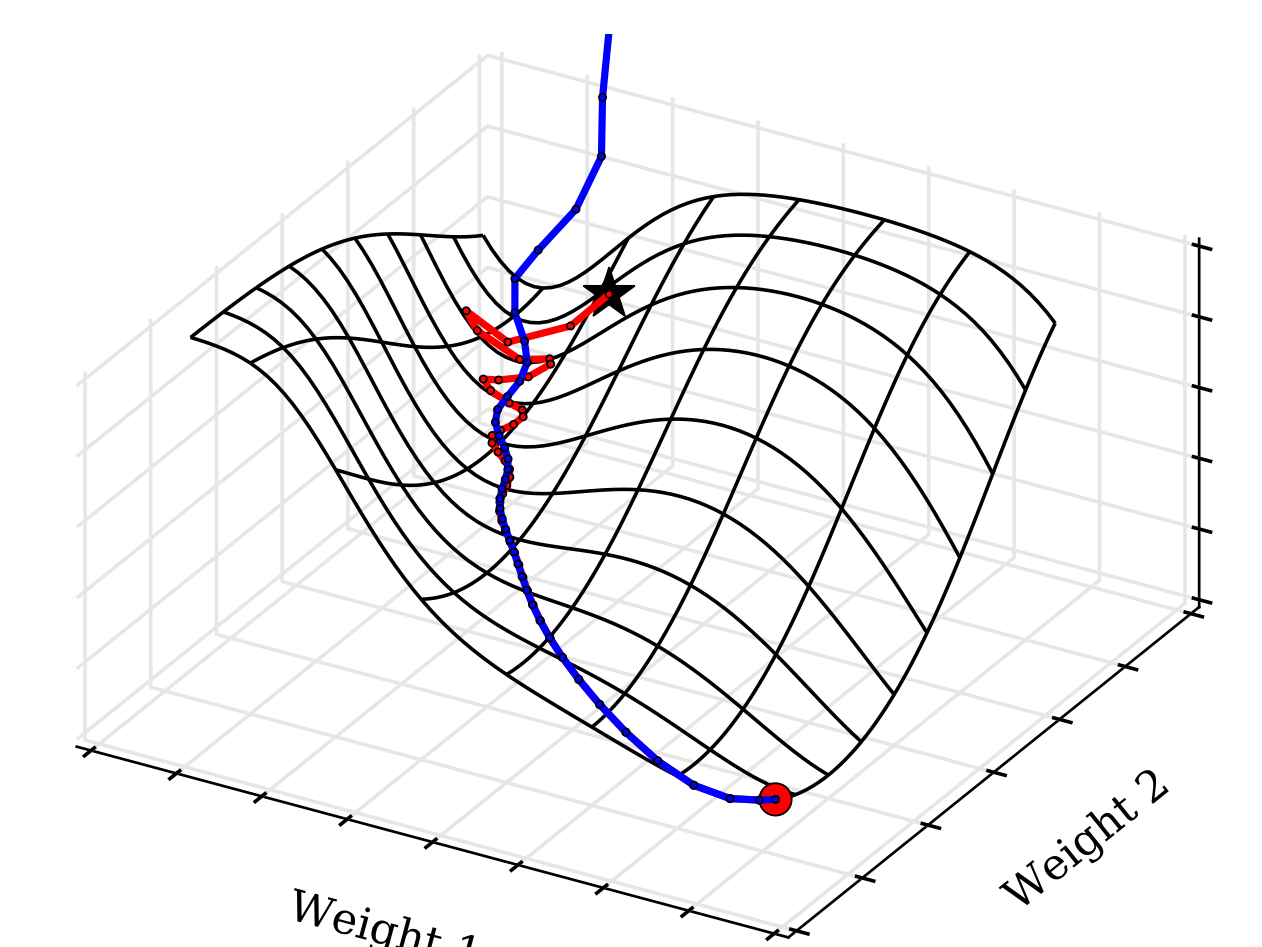
Forward update rule:

$$\begin{aligned}\mathbf{x}_{t+1} &\leftarrow \mathbf{x}_t + \alpha \mathbf{v}_t \\ \mathbf{v}_{t+1} &\leftarrow \beta \mathbf{v}_t - \nabla L(\mathbf{x}_{t+1})\end{aligned}$$



Reverse update rule:

$$\begin{aligned}\mathbf{v}_t &\leftarrow (\mathbf{v}_{t+1} + \nabla L(\mathbf{x}_{t+1})) / \beta \\ \mathbf{x}_t &\leftarrow \mathbf{x}_{t+1} - \alpha \mathbf{v}_t\end{aligned}$$



Need to store tiny corrections to each reversal step to ensure exact reversal.

Conclusion

- We can compute gradients of learning procedures...
- This lets us optimize thousands of hyperparameters!
- All code for experiments at github.com/HIPS/hypergrad
- We also wrote an autodiff package that works on standard Numpy code: github.com/HIPS/autograd