Over-Parametrization in Deep Learning

Levent Sagun

November 17, 2017

Paris Sud
Joint work with: Utku Evci, Uğur Güney, Yann Dauphin, Léon Bottou, and Yann LeCun

The Loss Function

1. Take a dataset and split it into two parts: $\mathcal{D}_{train}$ & $\mathcal{D}_{test}$
2. Form the loss using only $\mathcal{D}_{train}$:

$$\mathcal{L}_{train}(w) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} \ell(w;(x,y))$$

3. Find: $w^* = \arg\min \mathcal{L}_{train}(w)$
4. *...and hope that it'll work on* $\mathcal{D}_{test}$.

The Loss Function

Some quantites:

- M : number of parameters $w \in \mathbb{R}^M$
- N : number of examples in the *training* set $|\mathcal{D}_{train}|$
- d : number of dimension in the input $x \in \mathbb{R}^d$
- k : number of classes in the dataset

Question: When do we call a model over-parametrized?

The Loss Function

Old fear:

- M in $w \in \mathbb{R}^M$ is large
- $\mathcal{L}(w)$ is non-convex in w

But SGD works:

- random stating point $\rightarrow$ same loss value
- larger $M \rightarrow$ better performance

Conclusion: Optimization is easy with SGD!

Training

What's special about SGD?

GD is bad use SGD:

- The total gradient (3) converges to a *local minimum* of the cost function. The algorithm then cannot escape this local minimum, which is sometimes a poor solution of the problem.

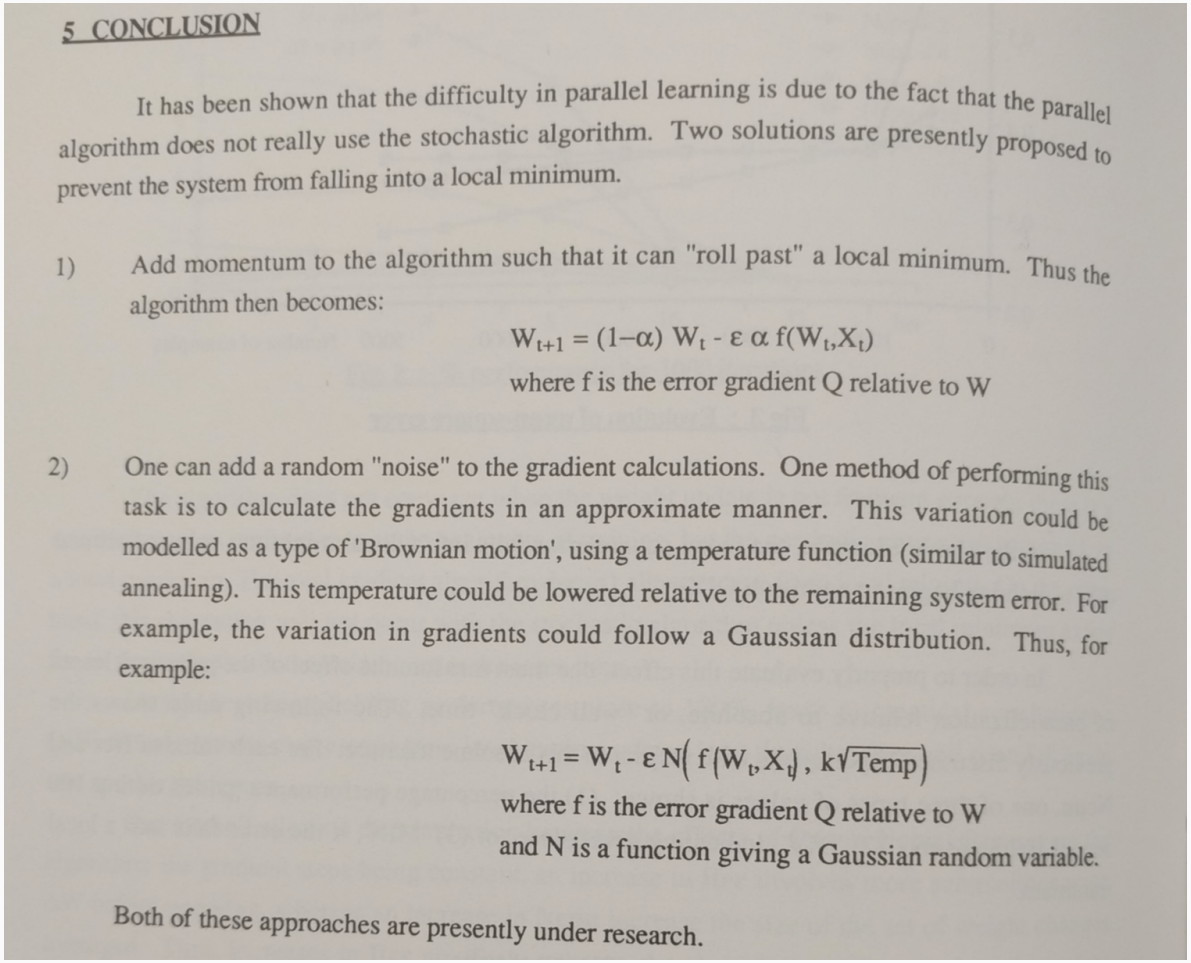
  In practical situations, the gradient algorithm may get stuck in an area where the cost is extremely ill conditionned, like a deep ravine of the cost function. This situation actually is a local minimum in a subspace defined by the largest eigenvalues of the Hessian matrix of the cost.

  The stochastic gradient algorithm (4) usually is able to escape from such bothersome situations, thanks to its random behavior (Bourrely, 1989).

This remark in Bottou (1991) is still widely believed *(with a twist)*!

What's special about SGD?

GD is bad use SGD:



5 CONCLUSION

It has been shown that the difficulty in parallel learning is due to the fact that the parallel algorithm does not really use the stochastic algorithm. Two solutions are presently proposed to prevent the system from falling into a local minimum.

1) Add momentum to the algorithm such that it can "roll past" a local minimum. Thus the algorithm then becomes:

$$W_{t+1} = (1-\alpha)\, W_t - \varepsilon\, \alpha\, f(W_t, X_t)$$

where f is the error gradient Q relative to W

2) One can add a random "noise" to the gradient calculations. One method of performing this task is to calculate the gradients in an approximate manner. This variation could be modelled as a type of 'Brownian motion', using a temperature function (similar to simulated annealing). This temperature could be lowered relative to the remaining system error. For example, the variation in gradients could follow a Gaussian distribution. Thus, for example:

$$W_{t+1} = W_t - \varepsilon\, N\left(f(W_t, X_t),\ k\sqrt{Temp} \right)$$

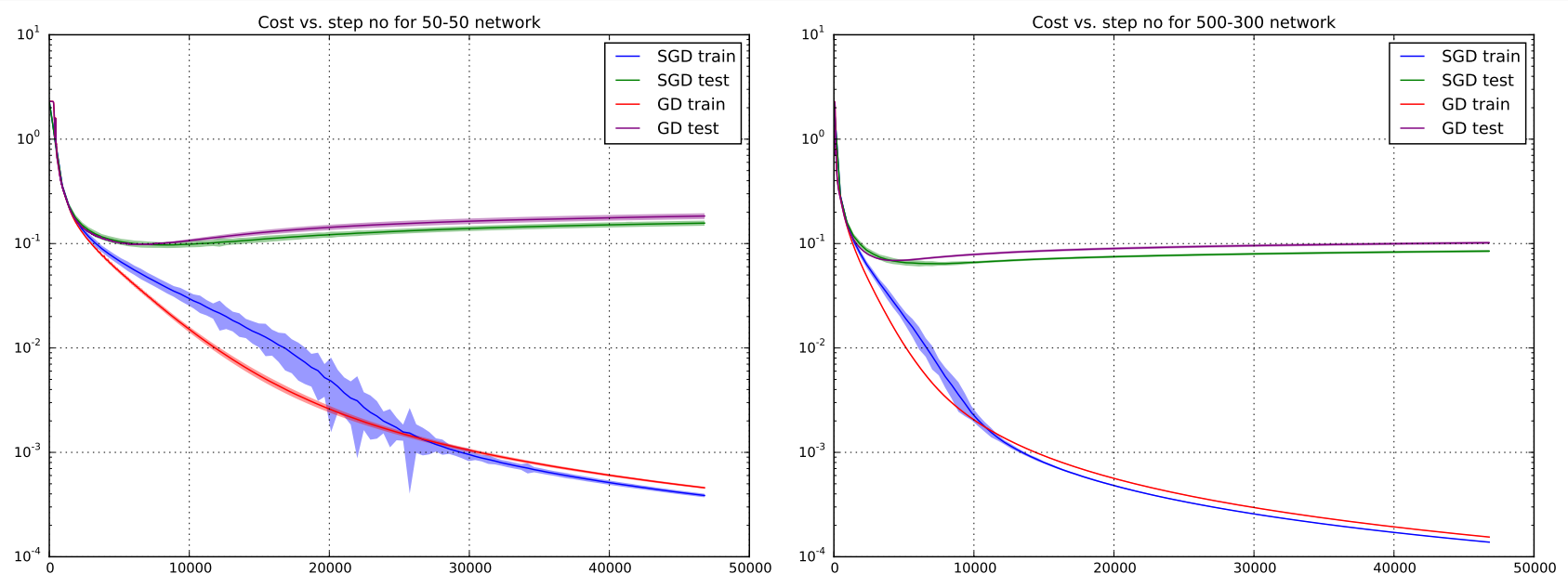
where f is the error gradient Q relative to W
and N is a function giving a Gaussian random variable.

Both of these approaches are presently under research.

Bourelly (1988)

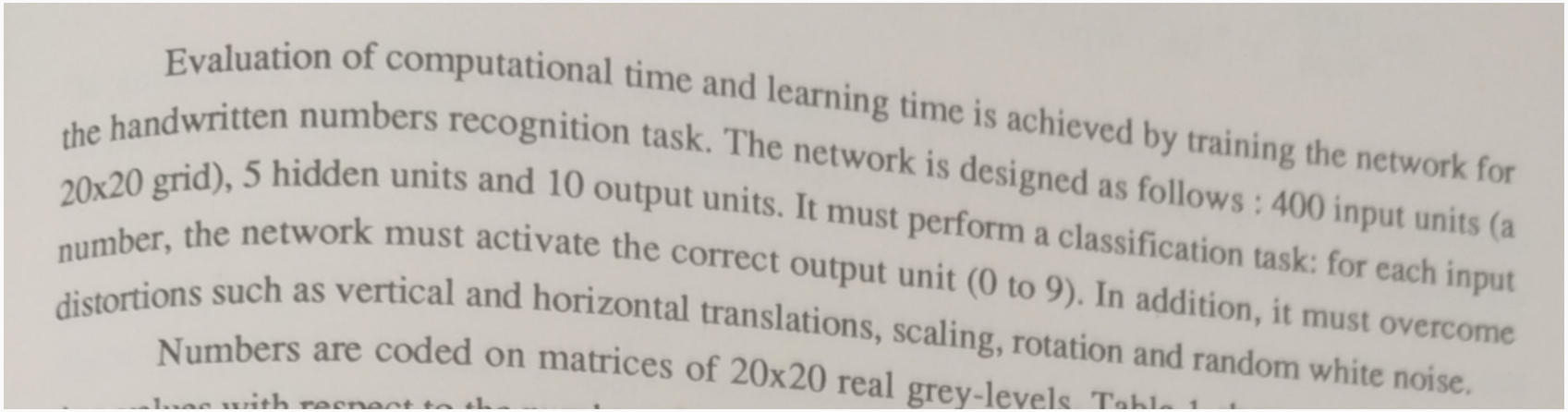
What's special about SGD?

Simple fully-connected network on MNIST (S.-Guney-LeCun, 2014):
$M \sim 43\text{K}$ (left) and $M \sim 450\text{K}$ (right)



What's special about SGD?

GD is bad use SGD:



Evaluation of computational time and learning time is achieved by training the network for the handwritten numbers recognition task. The network is designed as follows : 400 input units (a 20x20 grid), 5 hidden units and 10 output units. It must perform a classification task: for each input number, the network must activate the correct output unit (0 to 9). In addition, it must overcome distortions such as vertical and horizontal translations, scaling, rotation and random white noise.

Numbers are coded on matrices of 20x20 real grey-levels.

In Bourelly (1988), M is tiny, the network has only 5 neurons in the hidden layer!

What's special about SGD?

Assuming good initialization and right pre-processing (non-trivial)

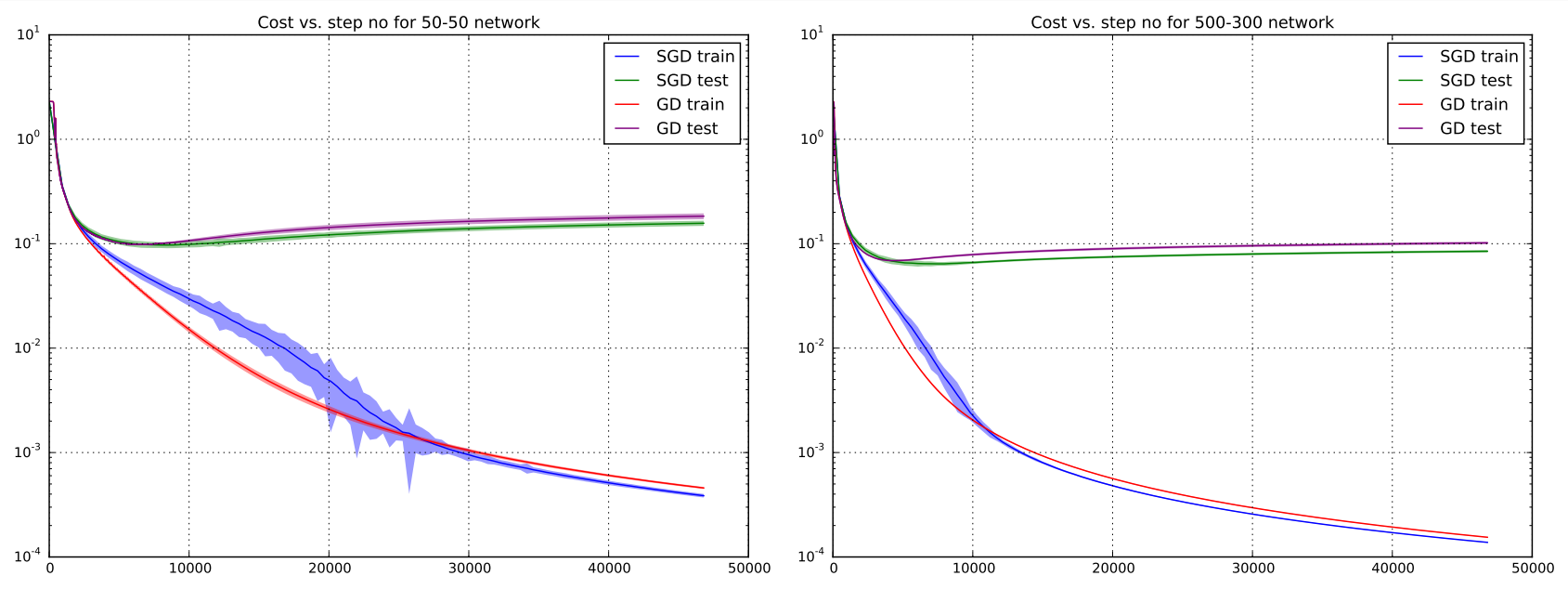
Training is easy when M is large (S.-Bottou 2016, Zhang et. al. 2016)

- Corrupt data
- Scramble labels
- More complex data
- Random polynomials

Generalization

SGD is really special

Simple fully-connected network on MNIST (S.-Guney-LeCun, 2014):
$M \sim 43$K (left) and $M \sim 450$K (right)

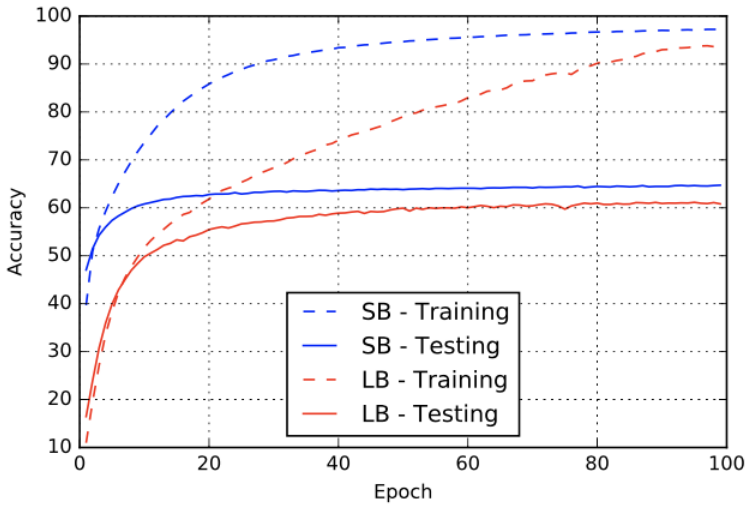


Average number of misclassifications:

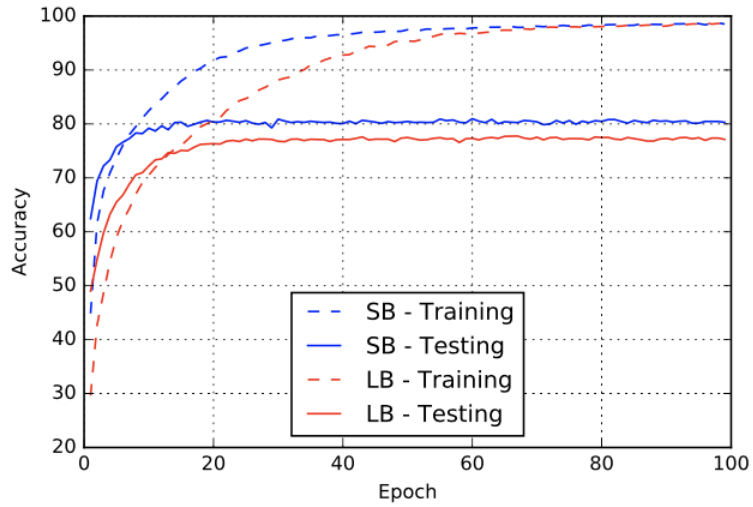
small-SGD: 256, small-GD: 299, large-SGD: 174, large-GD: 194

SGD is really special

Where common wisdom may be true (Keskar et. al. 2016.):



(a) F_2



(b) C_1

Figure 2: Convergence trajectories of training and testing accuracy for SB and LB methods

F2: fully connected, TIMIT ($M = 1.2$M)

C1: conv-net, CIFAR10 ($M = 1.7$M)

- Similar training error, but gap in the test error.

SGD is really special

Moreover, Keskar et. al. (2016) observe that:

- LB $\rightarrow$ sharp minima
- SB $\rightarrow$ wide minima

Considerations around the idea of sharp/wide minima:

$$\tilde{H}_{\Lambda,f}(R) \equiv f^{-1}\Big\{\int S_\Lambda(R-R')f[H(R')]dR'\Big\} \tag{2}$$

where R is a multidimensional vector representing all the coordinates in the molecule.

One of the simplest and most useful forms for S_Λ is a Gaussian

$$\begin{aligned} S_\Lambda(R) &\equiv C(\Lambda)e^{-R\Lambda^{-2}R} \\ C(\Lambda) &\equiv \pi^{-d/2}Det^{-1}(\Lambda) \end{aligned} \tag{3}$$

where d is the total dimensionality of R. The function f included in (2) allows for non-linear averaging. Two choices motivated by physical considerations are $f(x) = x$ and $f(x) = e^{-x/k_BT}$. These choices correspond respectively to the "diffusion equation" and "effective energy" methods which are described below. Wu [77] has presented a general discussion of transformations of the form of (2).

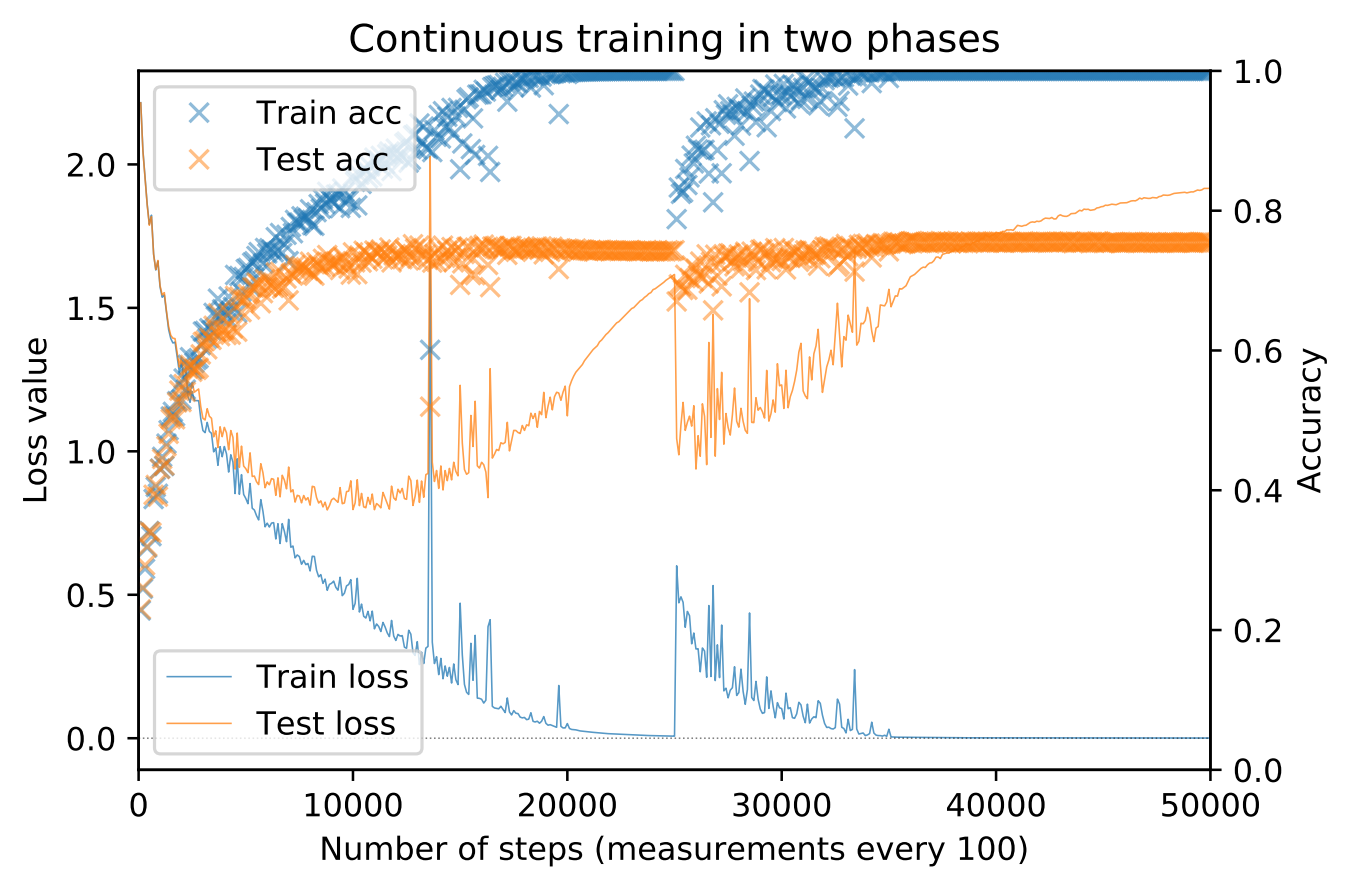
A highly smoothed $\tilde{H}_{\Lambda,f}$ (from which all high spatial-frequency components have been removed) will in most cases have fewer local minima than the unsmoothed ("bare") function, so it will be much easier to identify its global minimum. If the strong spatial-scaling hypothesis is correct, the position of this minimum can then be iteratively tracked by local-minimization as Λ decreases. As $\Lambda \rightarrow 0$, the position will approach the global minimizer of the bare objective function.

Pardalos et. al. 1993 (*More recently:* Zecchina et. al., Bengio et. al., ...)

SGD is really special

Repeating the LB/SB with a twist (S. et. al. 2017).

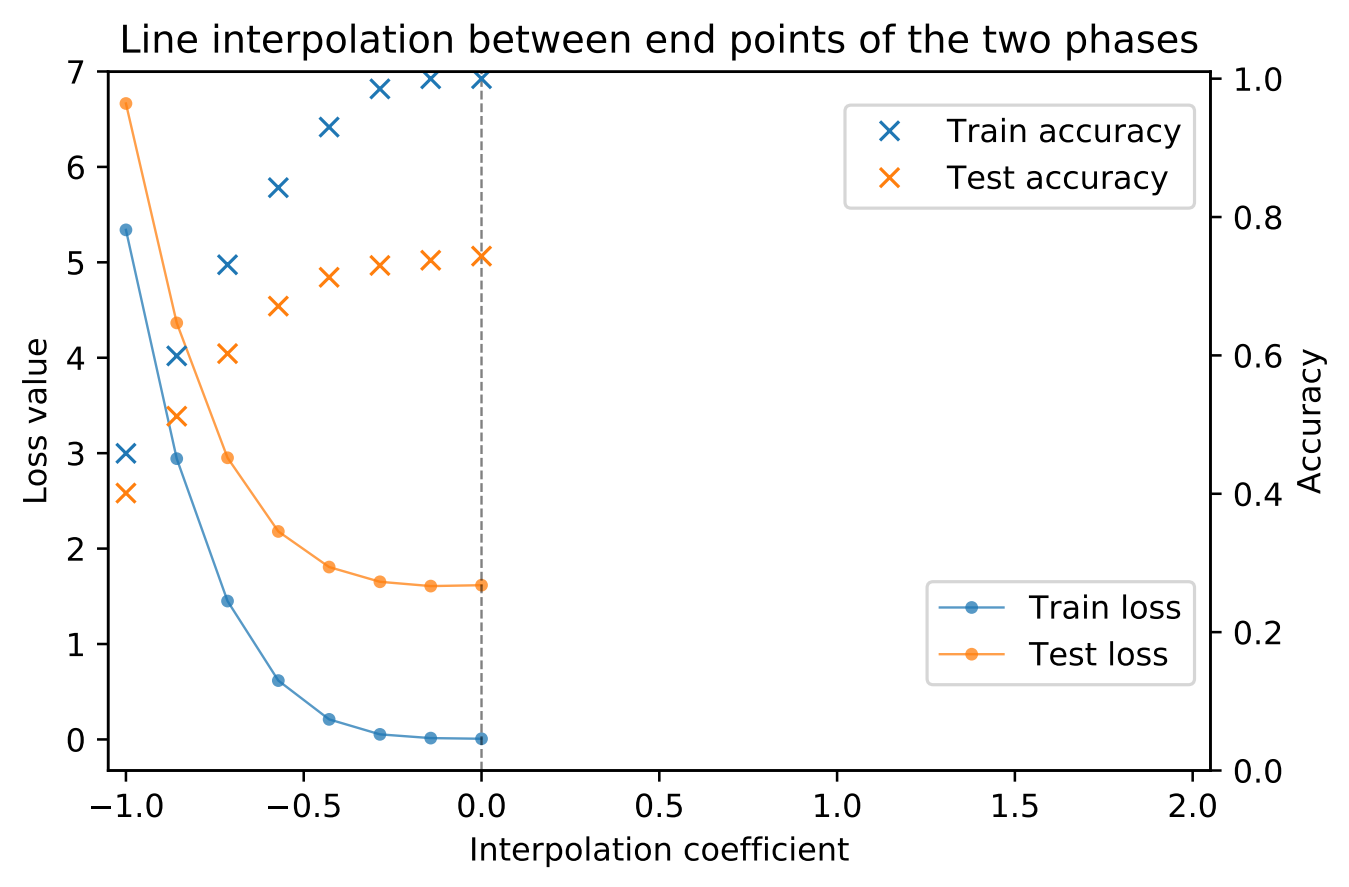
1. Train a large batch CIFAR10 on a *bare* AlexNet
2. At the end point switch to small batch



SGD is really special

Keep the two points: end of LB training and end of SB continuation.

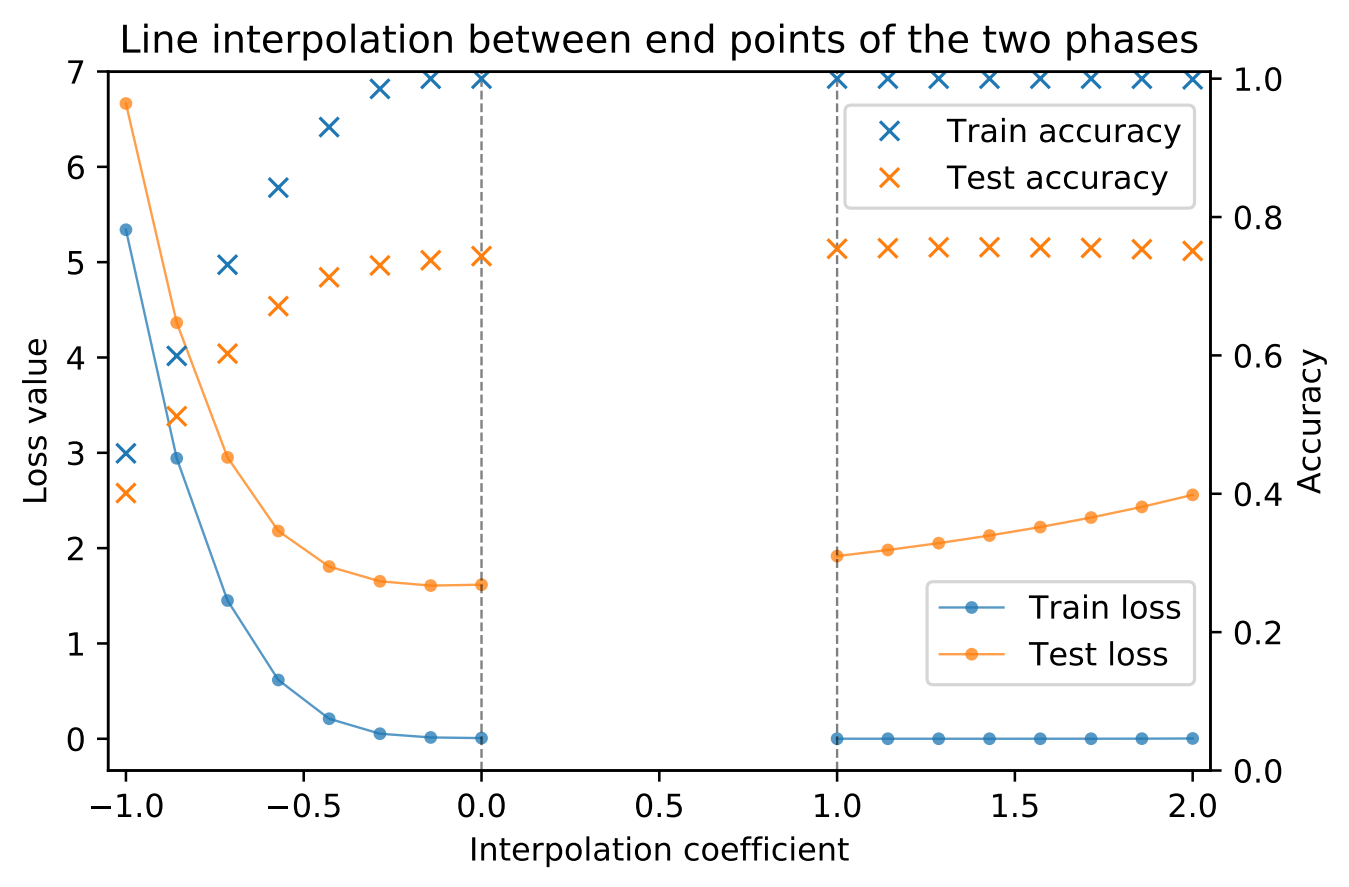
1. Extend a line away from the LB solution



SGD is really special

Keep the two points: end of LB training and end of SB continuation.

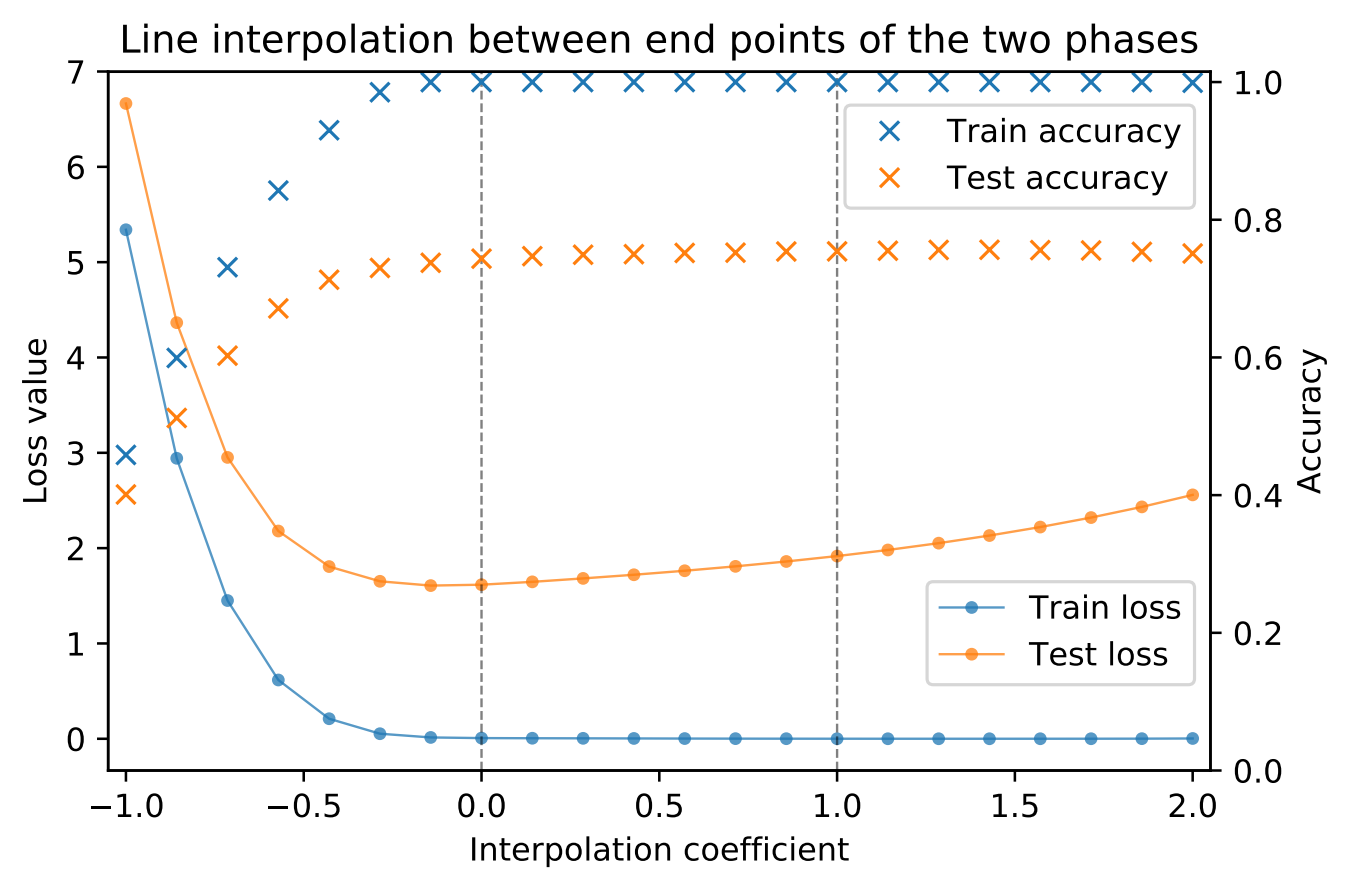
1. Extend a line away from the LB solution
2. Extend a line away from the SB solution



SGD is really special

Keep the two points: end of LB training and end of SB continuation.

1. Extend a line away from the LB solution
2. Extend a line away from the SB solution
3. Extend a line away between the two solutions



Local geometry of the loss

Hessian of the loss

Check out the Taylor expansion for local geometry:

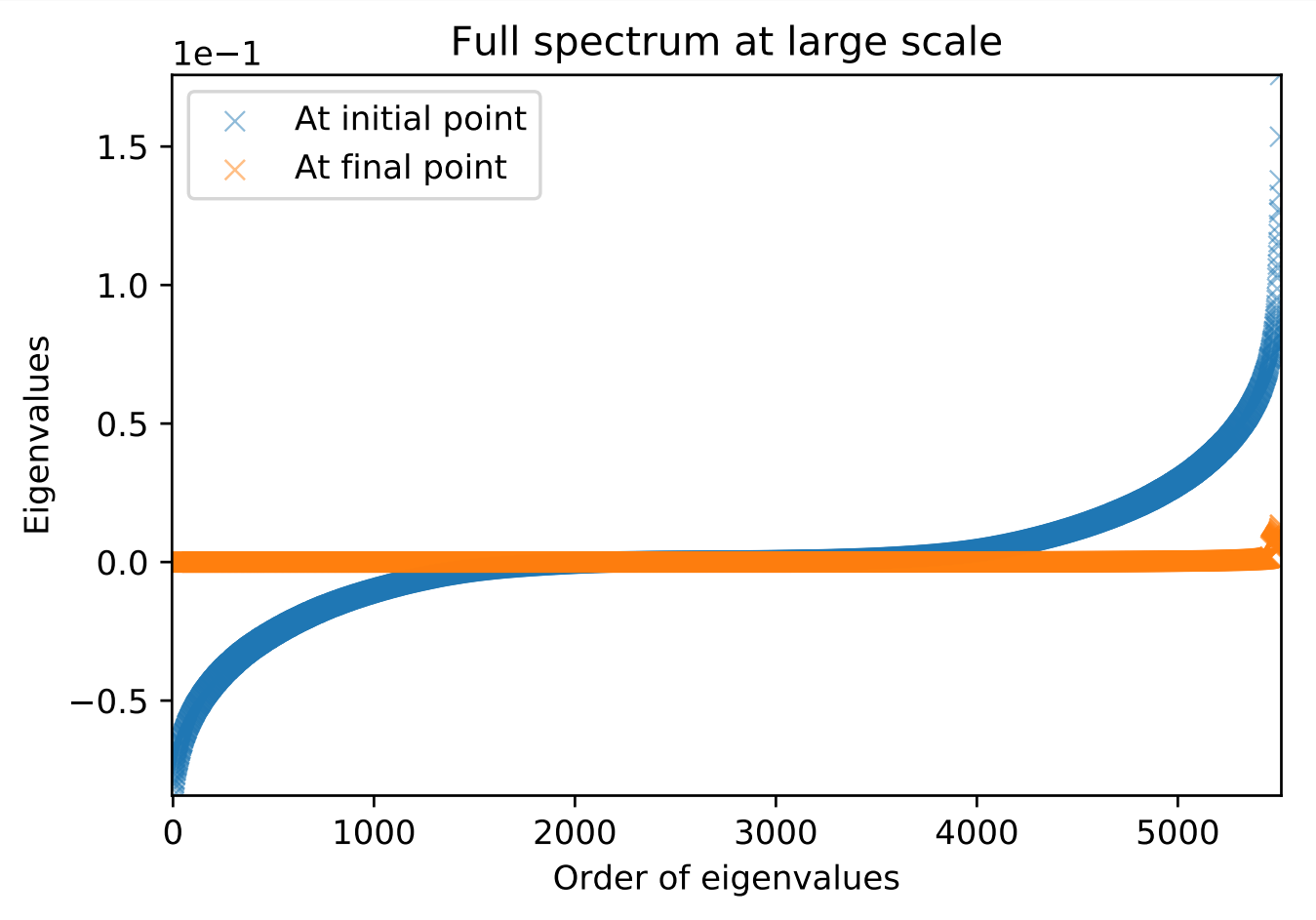
$$\mathcal{L}(w + \Delta w) \approx \mathcal{L}(w) + \Delta w^T \nabla \mathcal{L}(w) + \Delta w^T \nabla^2 \mathcal{L}(w) \Delta w$$

Local geometry at a critical point:

- All positive $\rightarrow$ local min
- All negative $\rightarrow$ local max
- Some negative $\rightarrow$ saddle
- Moving along eigenvectors and sizes of eigenvalues

Hessian of the loss: An example

2-hidden layer ReLU network with 50 categories of Gaussian blobs in 100 dimensions. Trained with SGD constant step size.



All zero, with some outliers...

Gauss-Newton decomposition of the Hessian

Loss functions between the output, s, and label, y

- MSE $\ell(s, y) = (s - y)^2$
- Hinge $\ell(s, y) = \max\{0, sy\}$
- NLL $\ell(s_y, y) = -s_y + \log \sum_{y'} \exp s_{y'}$

are all convex in their output: $s = f(w; x)$

Gauss-Newton decomposition of the Hessian

With $\ell \circ f$ in mind, the gradient and the Hessian per loss:

$$\nabla \ell(f(w)) = \ell'(f(w)) \nabla f(w)$$
$$\nabla^2 \ell(f(w)) = \ell''(f(w)) \nabla f(w) \nabla f(w)^T + \ell'(f(w)) \nabla^2 f(w)$$

then average over the training data:

$$\nabla^2 \mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \ell''(f(w)) \nabla f(w) \nabla f(w)^T + \frac{1}{N} \sum_{i=1}^{N} \ell'(f(w)) \nabla^2 f(w)$$

First term of the decomposition

Let's ignore the second term for the moment and look at the first term:

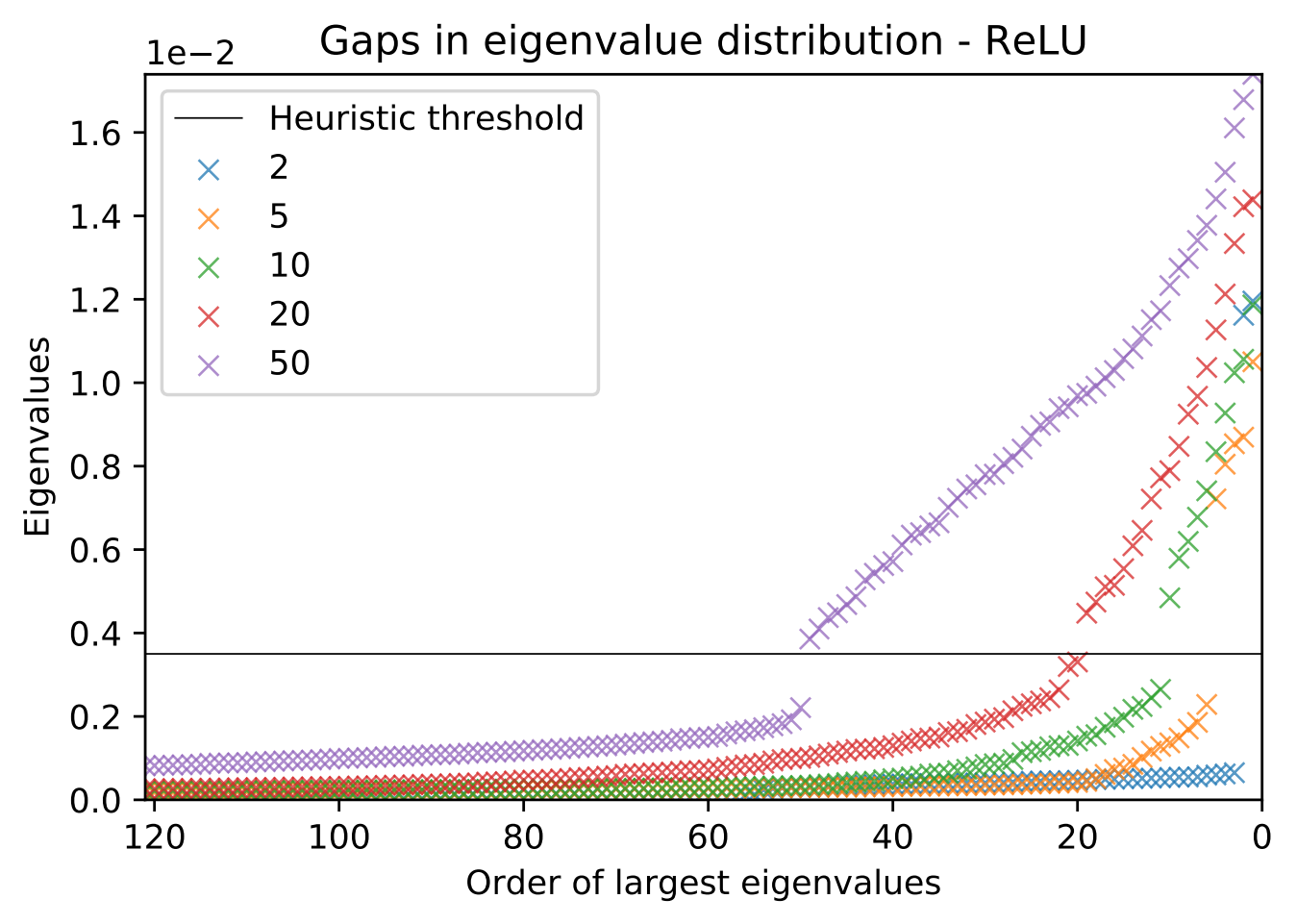
$$\nabla^2 \mathcal{L}(w) \approx \underbrace{\frac{1}{N} \sum_{i=1}^{N} \ell''(f(w)) \nabla f(w) \nabla f(w)^T}_{g(w) g(w)^T}$$

where g is an $M \times N$ matrix with $min\{M, N\}$ non-trivial eigenvalues.

- Suppose the data has k clusters each with small variance
- Redundancy in N examples (even when N is too large)
- Would $G(w)$ have order k non-trivial eigenvalues?

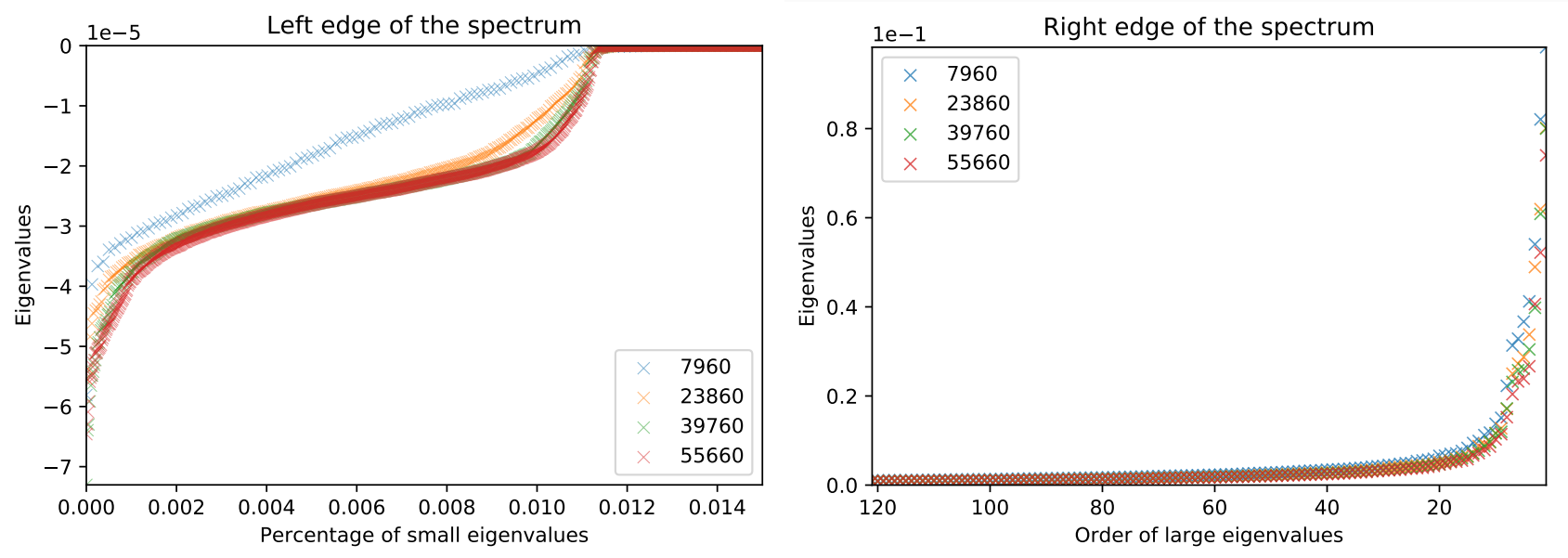
Closer look: (1) data and outliers

Let's train a neural network on artificial data of k-blobs for a fixed ReLU architecture, learning rate, and batch-size. Changing k gives rise to roughly k large eigenvalues:



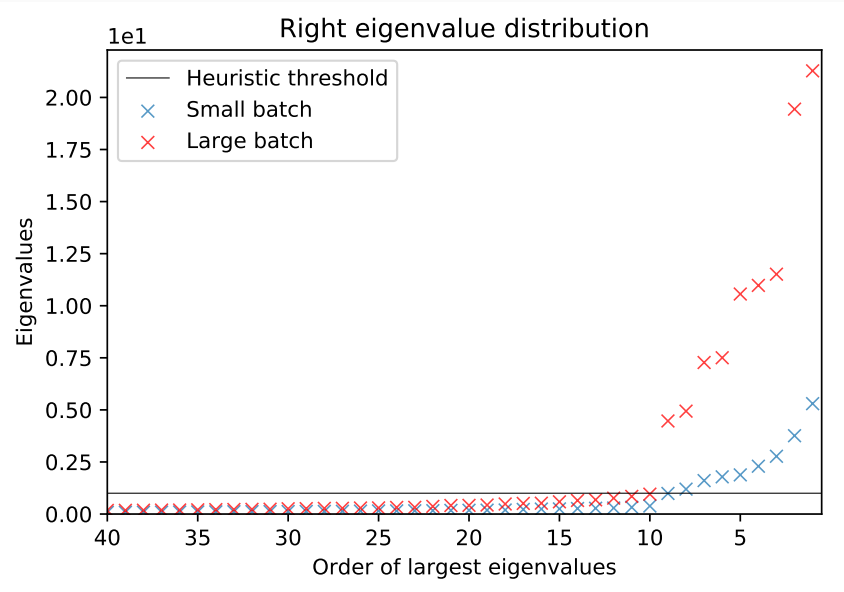
Closer look: (2) number of parameters and outliers

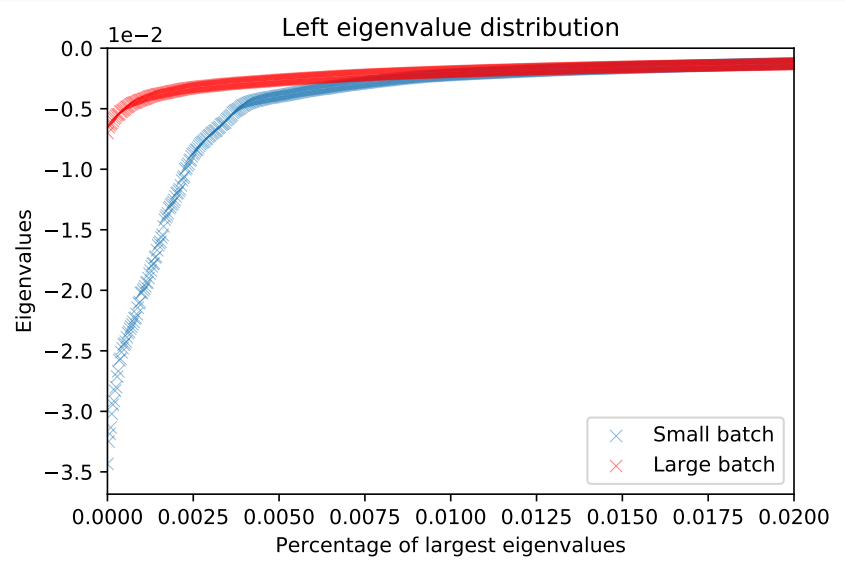
Increasing the size of the network adds more zeros.



Closer look: (3) algorithm and outliers

Increasing the batch-size leads to larger outlier eigenvalues.

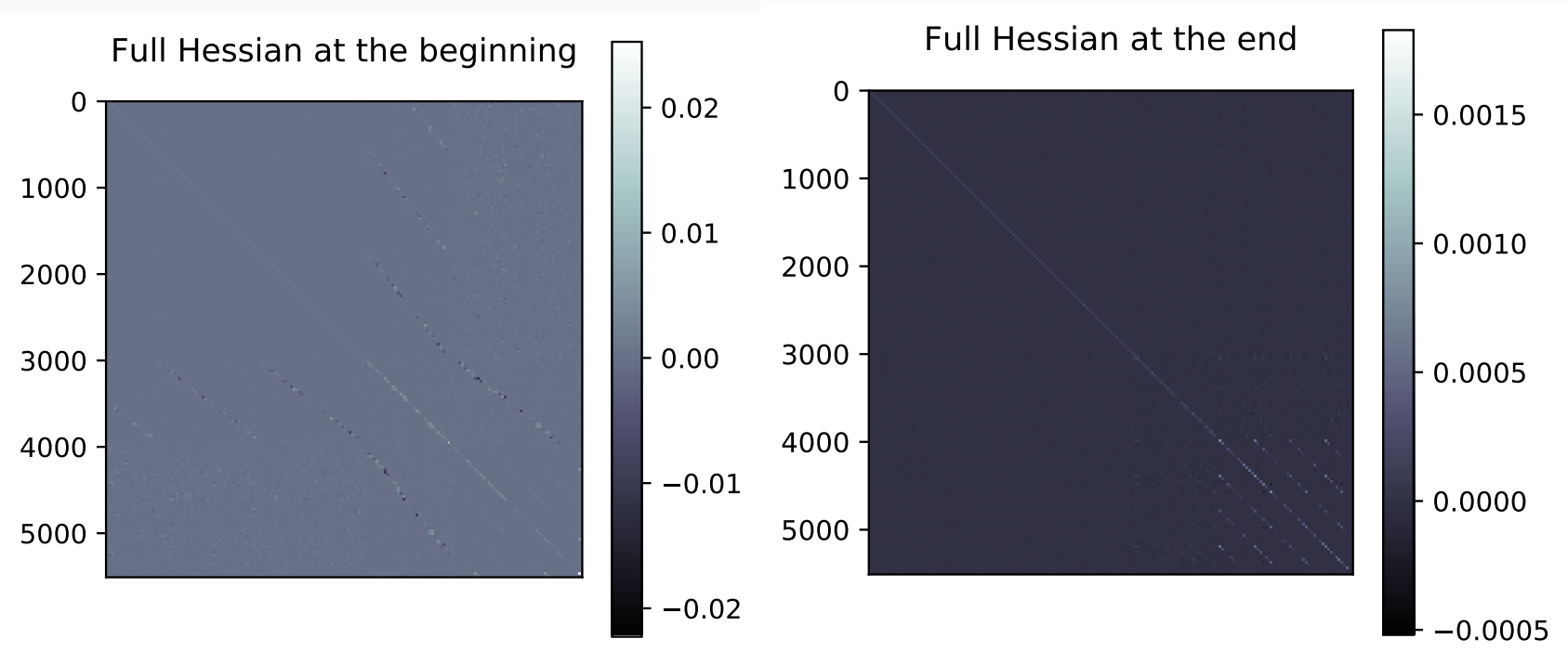




Further observations

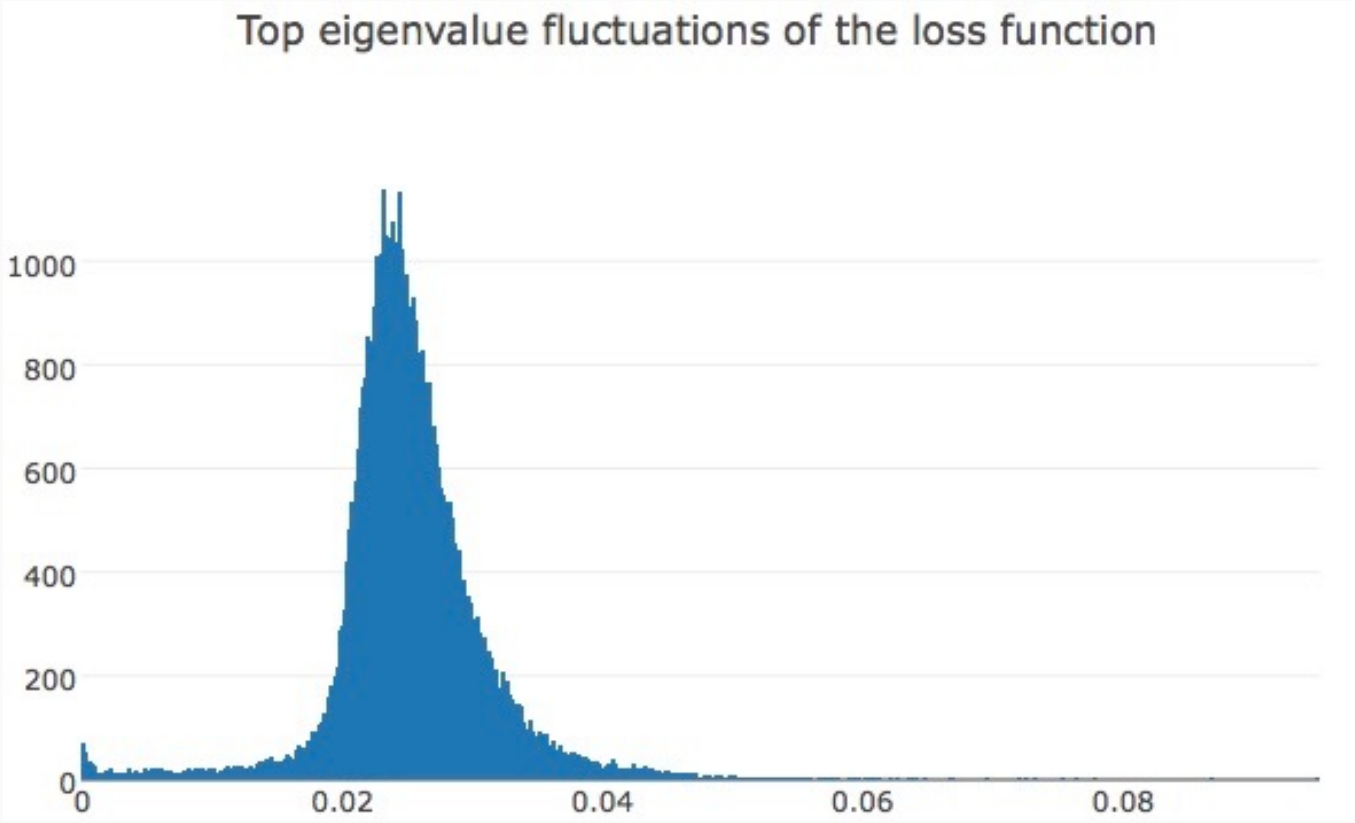
QUESTION: What's the fluctuations of the top eigenvalue?

- What should we expect?
- Not many ev's that push the top one.
- The Hessian is correlated



Further observations

Fix the dimension of the system, calculate the top eigenvalue and repeat training with the same fixed number of iterations 50K times.



Remark: Kurt Johansson "From Gumbel to Tracy-Widom" (2005)

Final thoughts and conclusion

- Existing tools require non-degenerate functions
- Existing intuituion may be misleading
- Maybe this flatness is *the* feature

Thank you!