An Empirical Look at the Loss Landscape

HEP AI - September 4, 2018

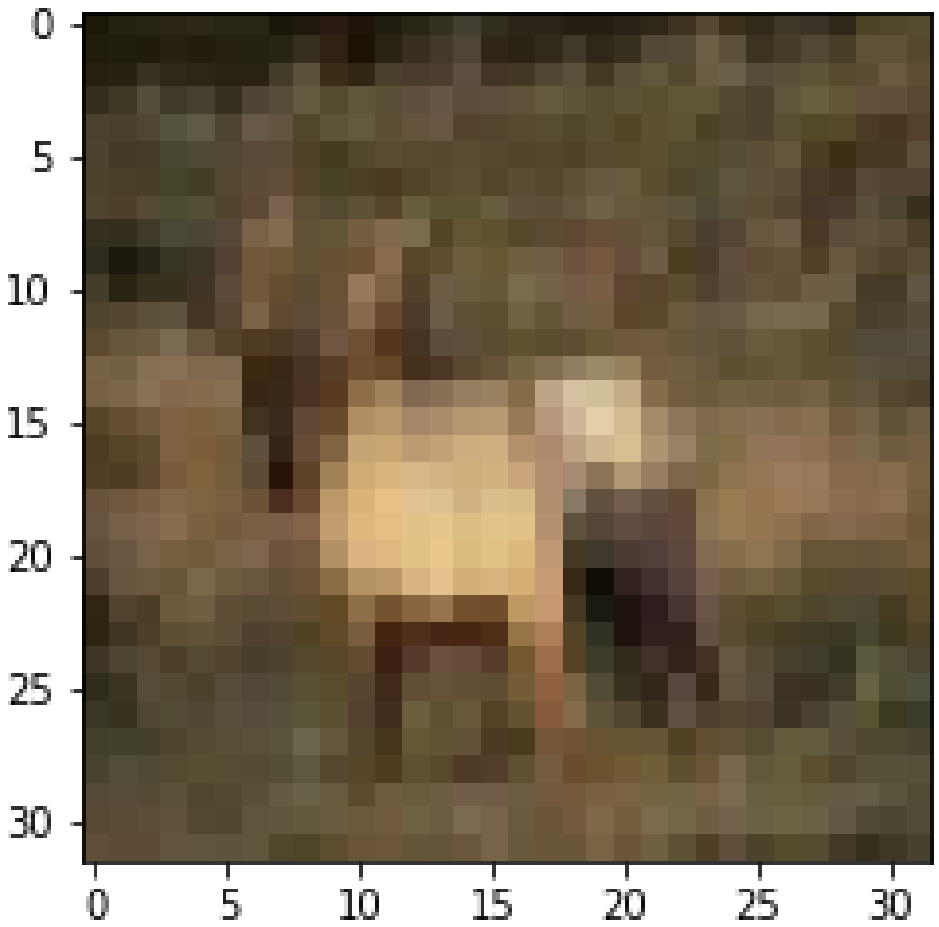
Components of training an image classifier

For fixed architecture of ResNet 56 we have:

1. Preprocessing: normalize, shift and flip (show examples)
2. Momentum
3. Weight decay (aka L^2 regularization)
4. Learning rate scheduling

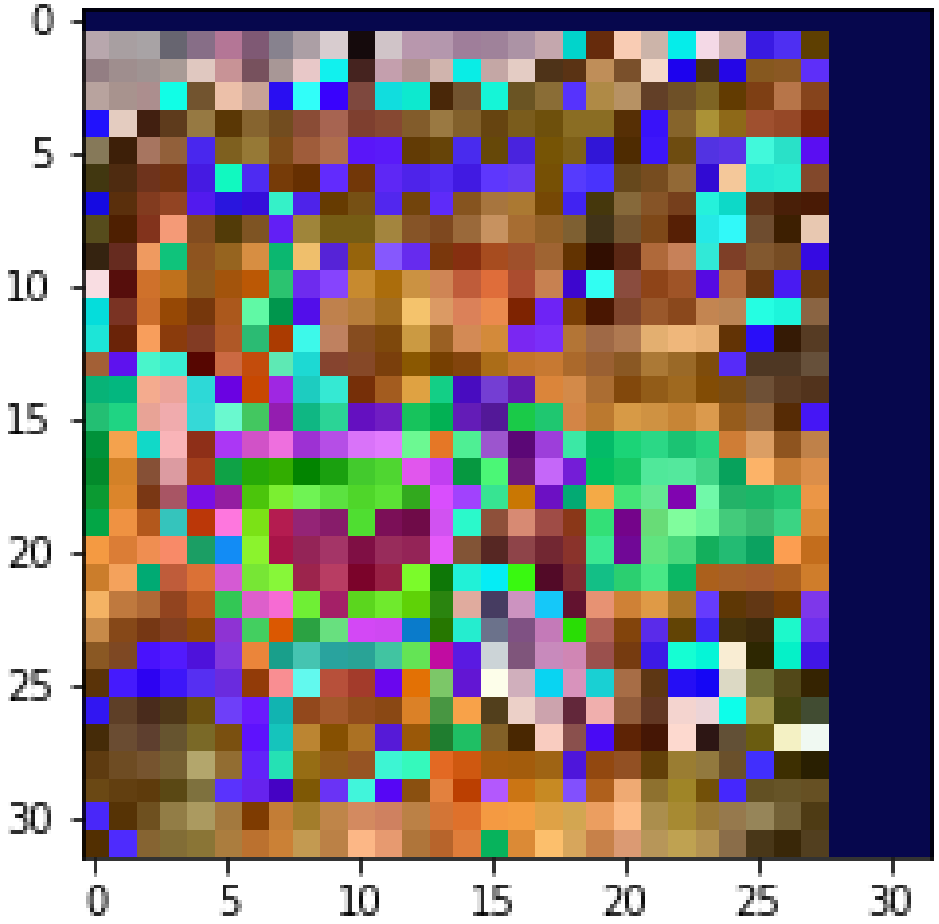
Components of training an image classifier

Dataset: CIFAR10 raw



Components of training an image classifier

Dataset: CIFAR10 processed (normalize, shift and flip)



Components of training an image classifier

With all the ingredients (mom, wd, prep) we get 93.1% accuracy on C10!

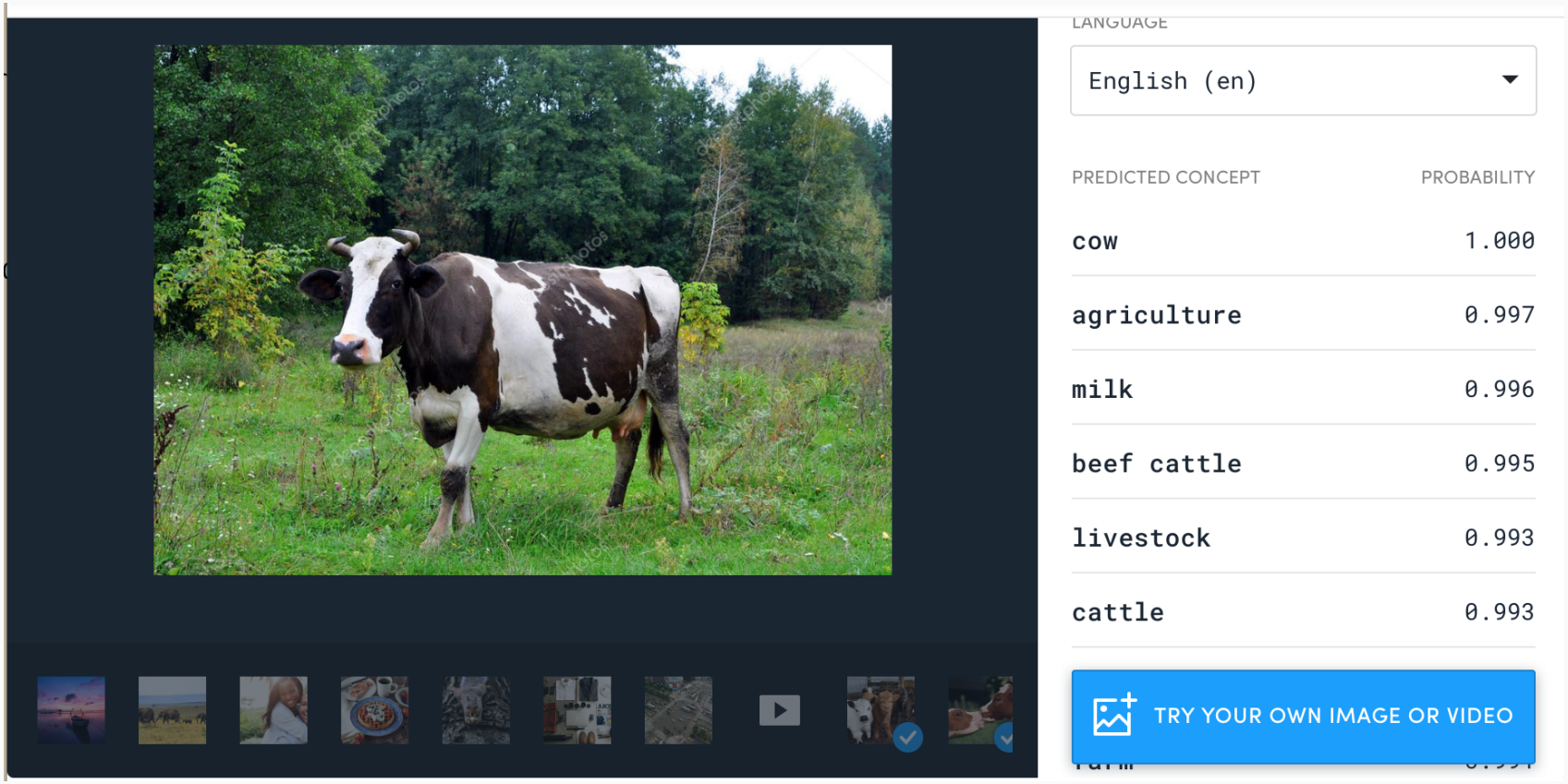
- Remove momentum only: -1.5%
- Remove weight decay only: -3.2%
- Remove preprocessing only: -6.3%
- Remove all three: -12.5%

What components are essentially necessary?

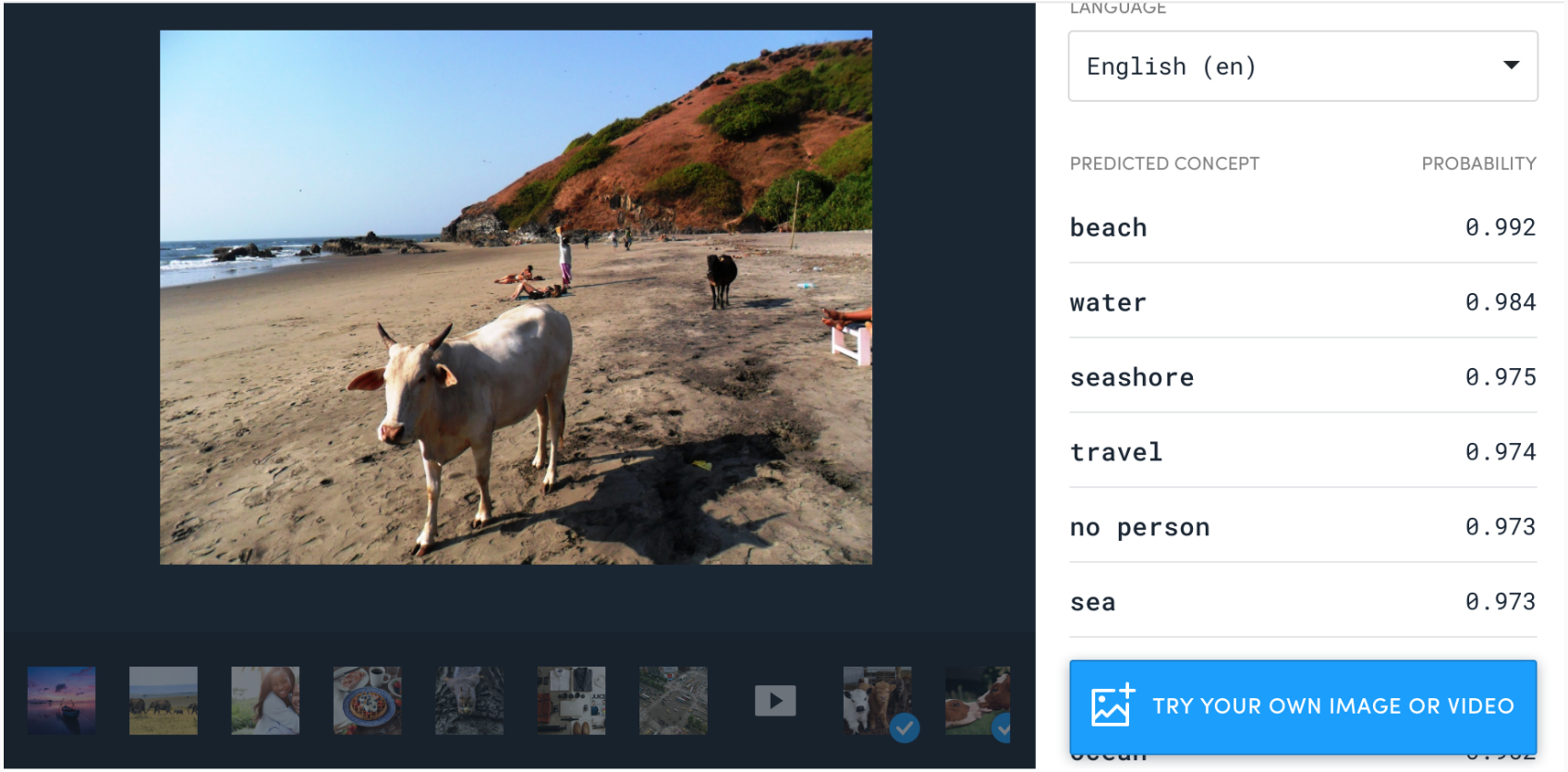
Expressivity and overfitting

- Regression vs. classification is there a fundamental reason that makes one harder?
- Is it always possible to memorize the training set? (9 examples in CIFAR100)
- What's happening to the loss when the accuracy is stable?

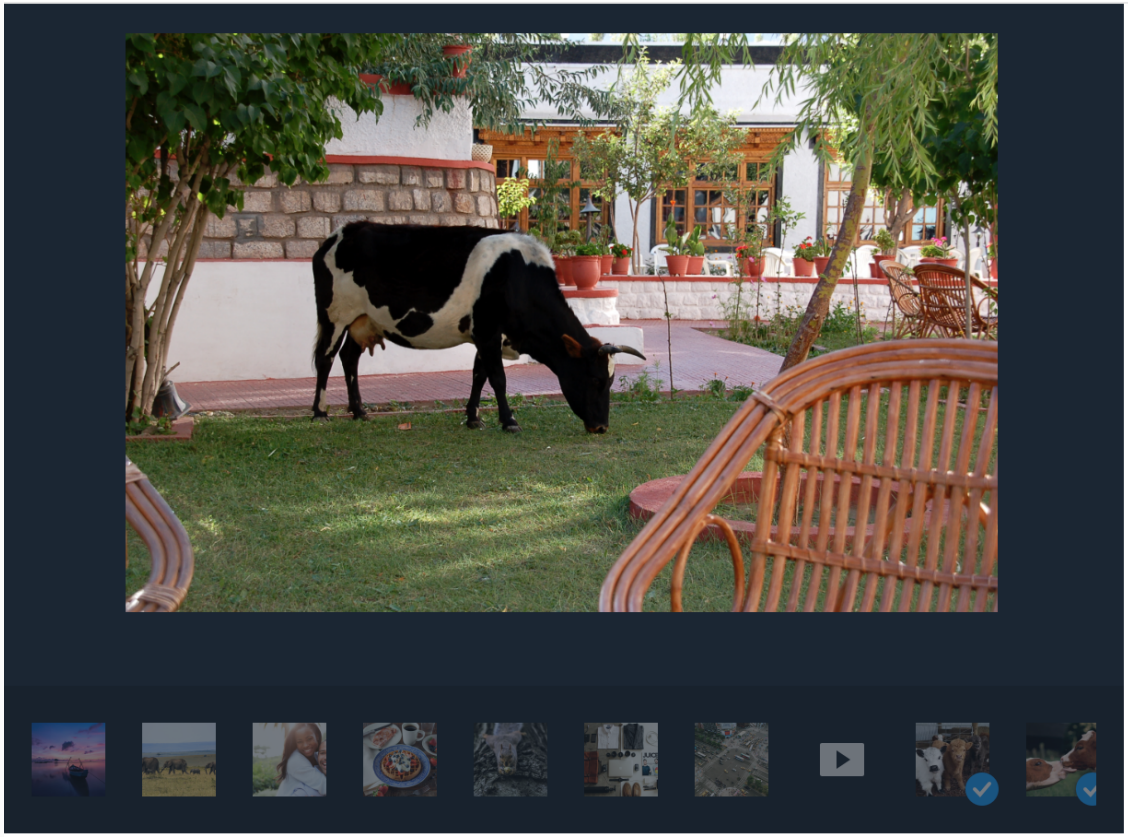
State of Image Recognition - http://clarifai.com/



State of Image Recognition - http://clarifai.com/



State of Image Recognition - http://clarifai.com/



LANGUAGE

English (en)

PREDICTED CONCEPT	PROBABILITY
no person	0.980
summer	0.951
outdoors	0.939
farm	0.922
mammal	0.920
landscape	0.897

TRY YOUR OWN IMAGE OR VIDEO

State of Image Recognition - http://clarifai.com/

Is all we do still just a fancy curve fitting?

Geometry of the training surface

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 $\mathcal{D}_{train}$:

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

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

The Loss Function

Some quantites:

- M : number of parameters $w \in \mathbb{R}^M$
- N : number of neurons in the first layer
- P : 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?

Question: How to minimize the high-dimensional, non-convex loss?

GD is bad use SGD

"Stochastic gradient learning in neural networks", Léon Bottou, 1991

- 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).

GD is bad use SGD

Bourelly (1988)

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:

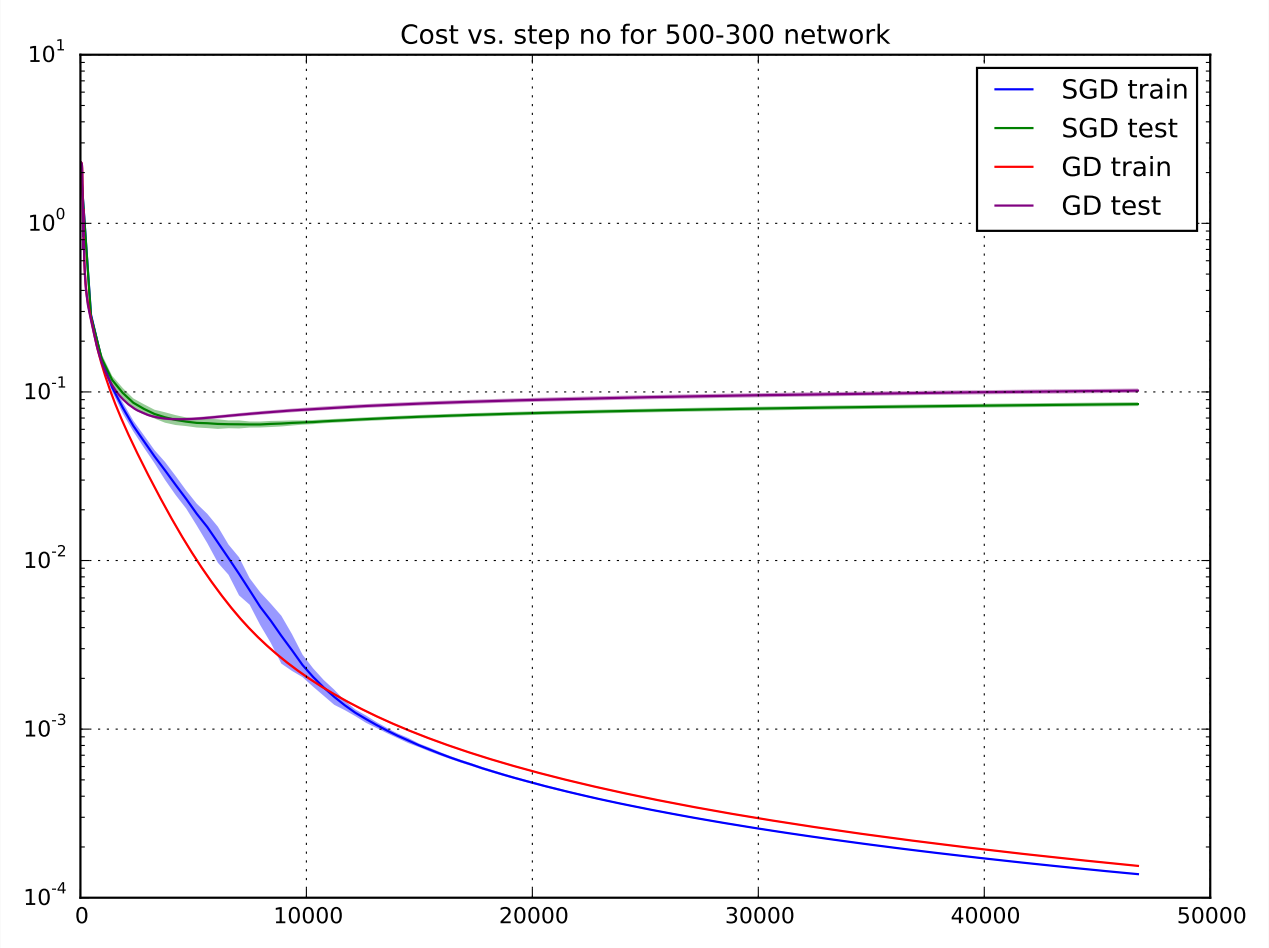
$$W_{t+1} = W_t - \varepsilon\, N\left(f(W_t, X_t),\ k\sqrt{Temp} \right)$$

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.

GD is bad use SGD

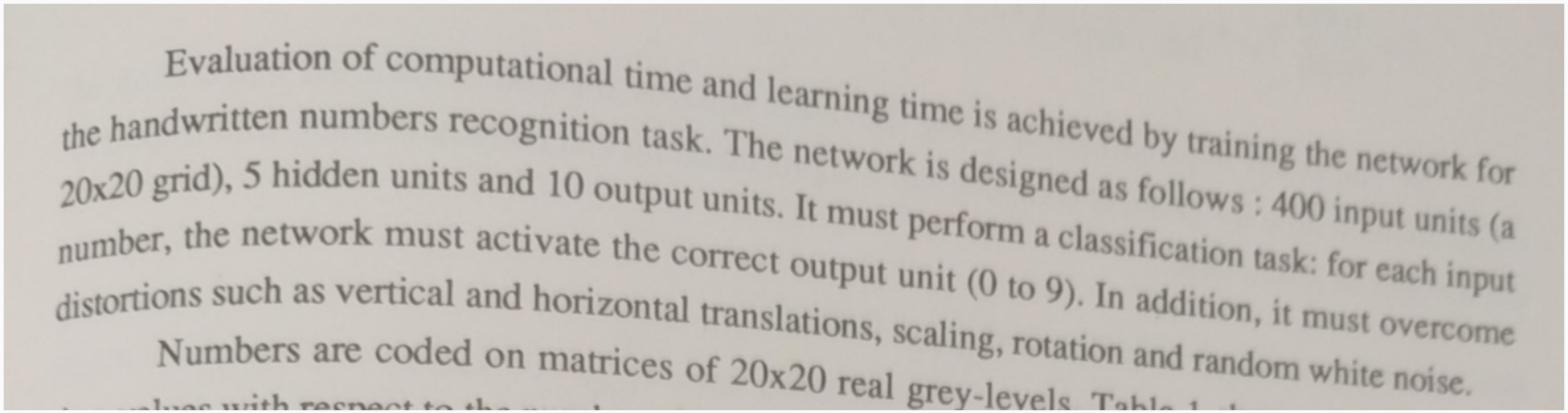
Simple fully-connected network on MNIST: $M \sim 450\text{K}$ (right)



Average number of mistakes: SGD 174, GD 194

GD is bad use SGD

The network has only 5 neurons in the hidden layer!



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. Table 1

GD vs SGD in the mean field approach

Take $\ell(y, f(w;x)) = (y - f(w;x))^2$ where $f(w;x) = \frac{1}{N}\sum_{i=1}^{N} \sigma(w_i, x)$

Expand the square and take expectation over data:

$$\mathcal{L}(w) = Const + \frac{2}{N}\sum_{i=1}^{N} V(w_i) + \frac{1}{N^2}\sum_{i,j=1}^{N} U(w_i, w_j)$$

Population risk in the large N limit:

$$\mathcal{L}(\rho) = Const + 2\int V(w)\rho(dw) + \int U(w_1, w_2)\rho(dw_1)\rho(dw_2)$$

Proposition: Minimizing the two functions are the same

GD vs SGD in the mean field approach

Write the gradient update per example and rearrange:

$$\Delta w_i = 2\eta \nabla_{w_i} \sigma(w_i, x)(y - \frac{1}{N}\sum_{i=1}^{N} \sigma(w_i, x))$$

$$= 2\eta \nabla_{w_i} y\sigma(w_i, x) - 2\eta \nabla_{w_i} \sigma(w_i, x) \frac{1}{N}\sum_{i=1}^{N} \sigma(w_i, x)$$

Taking expectation over (past) data gives the update (ith neuron):

$$\mathbb{E}(\Delta w | past)/2\eta = -\nabla_{w_i} V(w_i) - \frac{1}{N}\sum_{j=1}^{N} \nabla_{w_i} U(w_i, w_j)$$

- Then pass to the large N limit (with proper timestep scaling)
- And write the continuity equation for the density.

GD vs SGD in the mean field approach

References:

1. Mei, Montanari, Nguyen 2018 (above approach)
2. Sirignano, Spiliopoulos 2018 (harder to read)
3. Rotskoff, Vanden-Eijnden 2018 (additional diffusive and noise terms, as well as a CLT)
4. Wang, Mattingly, Lu 2017 (same approach different problems)

Is it really the case that in the large N limit, GD and SGD are the same?

Quick look into Rotskoff and Vanden-Eijnden

Here θ is learning rate / batch size

Dean's equation for correlated noise terms

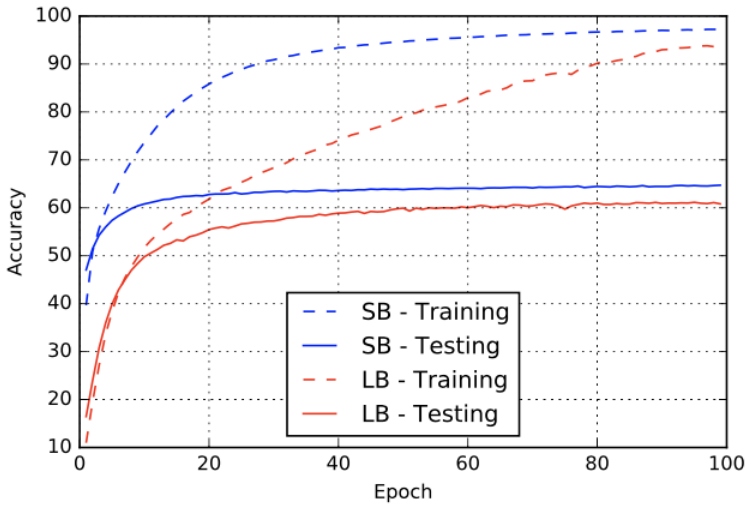
$$\partial_t \rho_n = \nabla \cdot \left(-c \nabla F \rho_n + \int_{D \times \mathbb{R}} c c' \nabla K(\boldsymbol{y}, \boldsymbol{y}') \rho_n' \rho_n d\boldsymbol{y}' dc' \right)$$
$$+ \partial_c \left(-F \rho_n + \int_{D \times \mathbb{R}} c' K(\boldsymbol{y}, \boldsymbol{y}') \rho_n' \rho_n d\boldsymbol{y}' dc' \right)$$
$$+ \tfrac{1}{2}\theta \nabla\nabla : \left(\rho_n c^2 A_2([f_n(t) - f], \boldsymbol{y}, \boldsymbol{y}) \right) + \tfrac{1}{2}\theta \partial_c^2 \left(\rho_n A_0([f_n(t) - f], \boldsymbol{y}, \boldsymbol{y}) \right)$$
$$+ \theta \partial_c \nabla \cdot \left(\rho_n c A_1([f_n(t) - f], \boldsymbol{y}, \boldsymbol{y}) \right)$$
$$+ \sqrt{\theta}\, \dot{\eta}_n(t, \boldsymbol{y}, c)$$

$P = n^2 \implies$

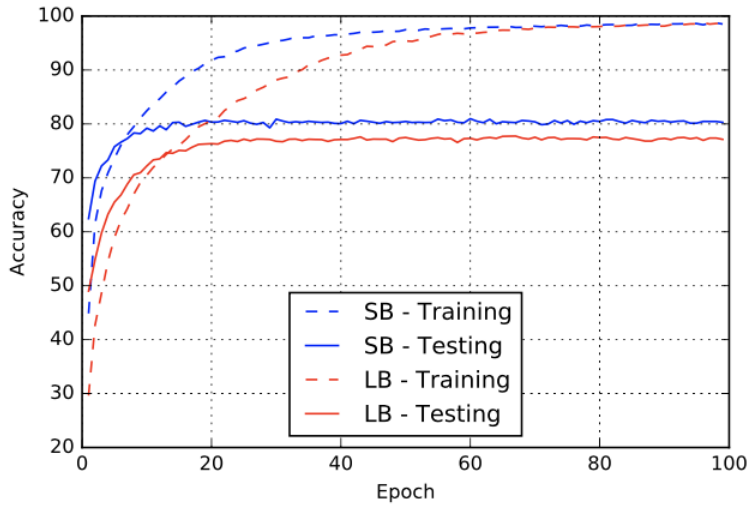
- Same first order term as gradient descent
- Guarantee of convergence
- Recover the error scaling

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

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

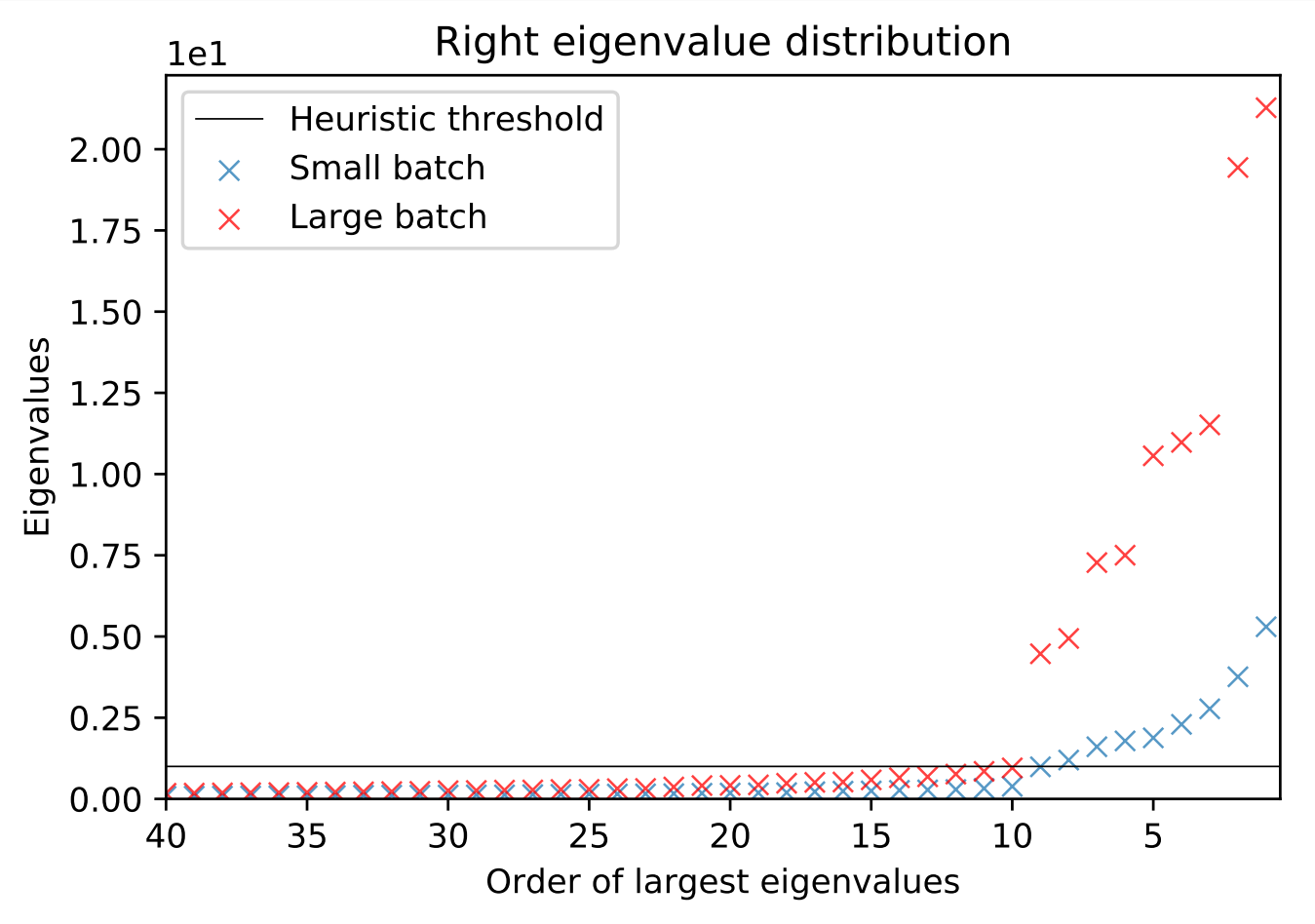
where d is the total dimensionality of R. The function f included in (2) allows for non-linear averaging. Two choices motivated by physical considerations are $f(x) = x$ and $f(x) = e^{-x/k_BT}$. These choices correspond respectively to the "diffusion equation" and "effective energy" methods which are 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., ...)

LB SB and outlier eigenvalues of the Hessian

MNIST on a simple fully-connected network. Increasing the batch-size leads to larger outlier eigenvalues.



Geometry of redundant over-parametrization

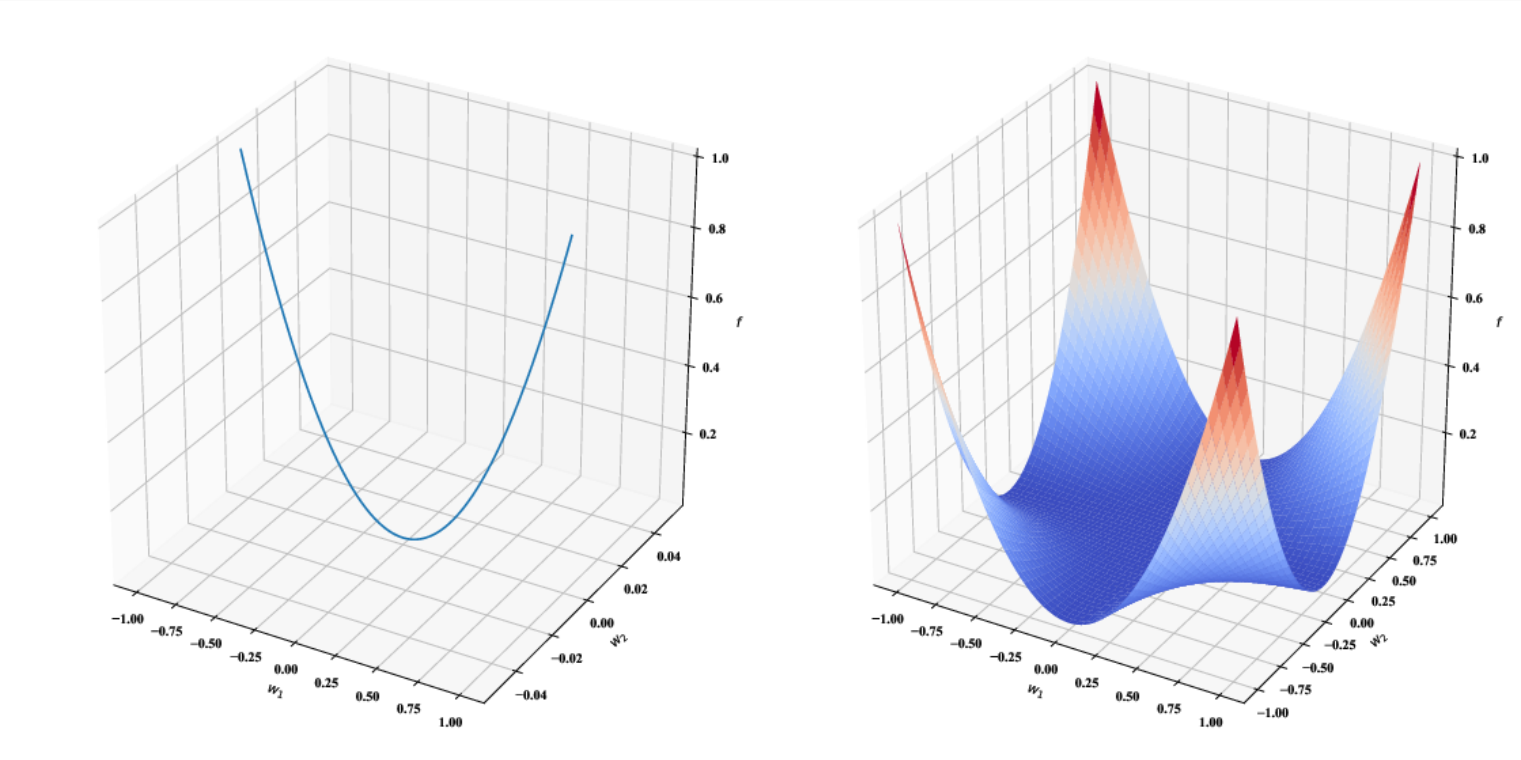
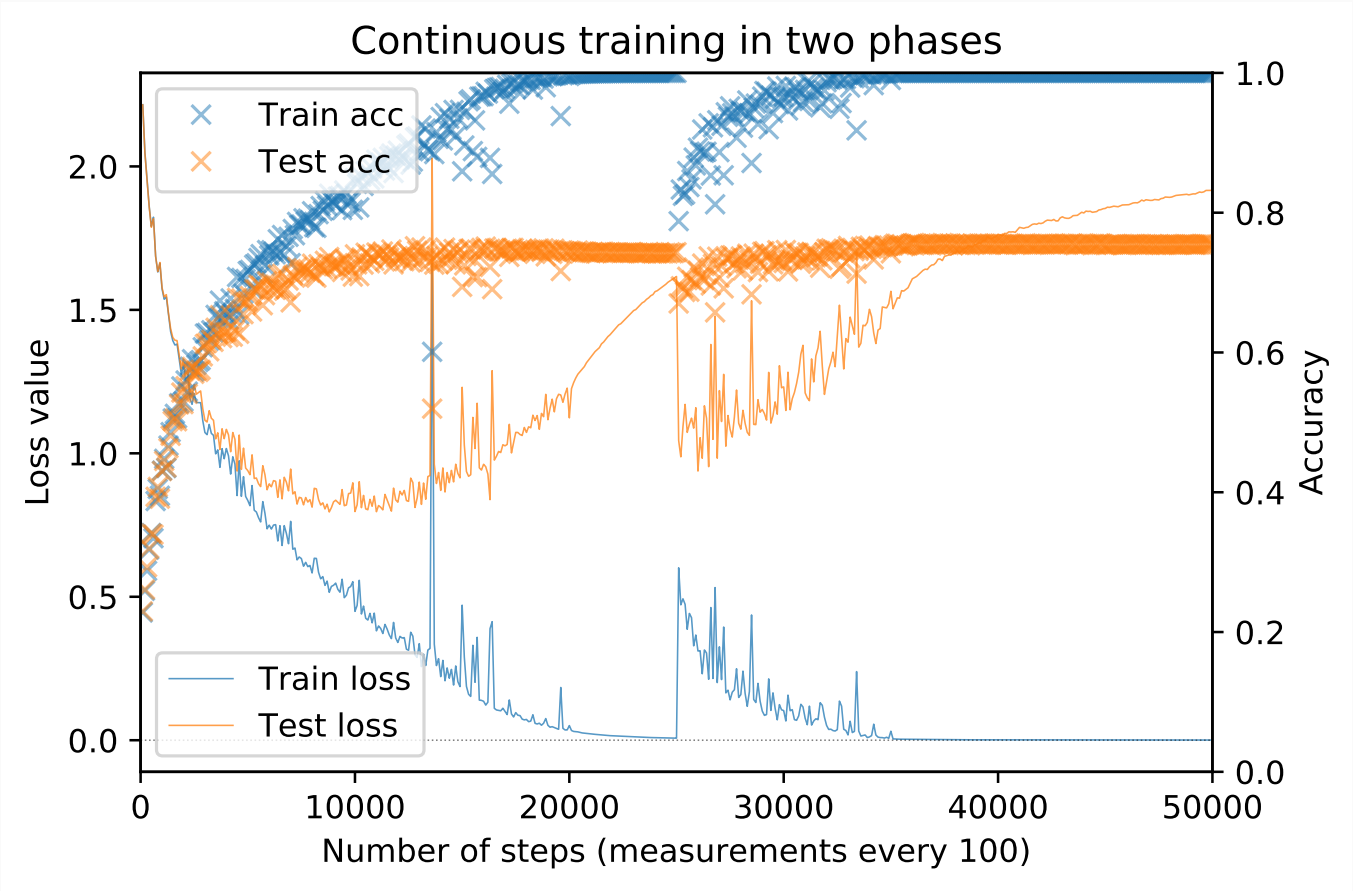


Figure: w^2 (left) vs. $(w_1 w_2)^2$ (right)

Searching for sharp basins

Repeating the LB/SB with a twist

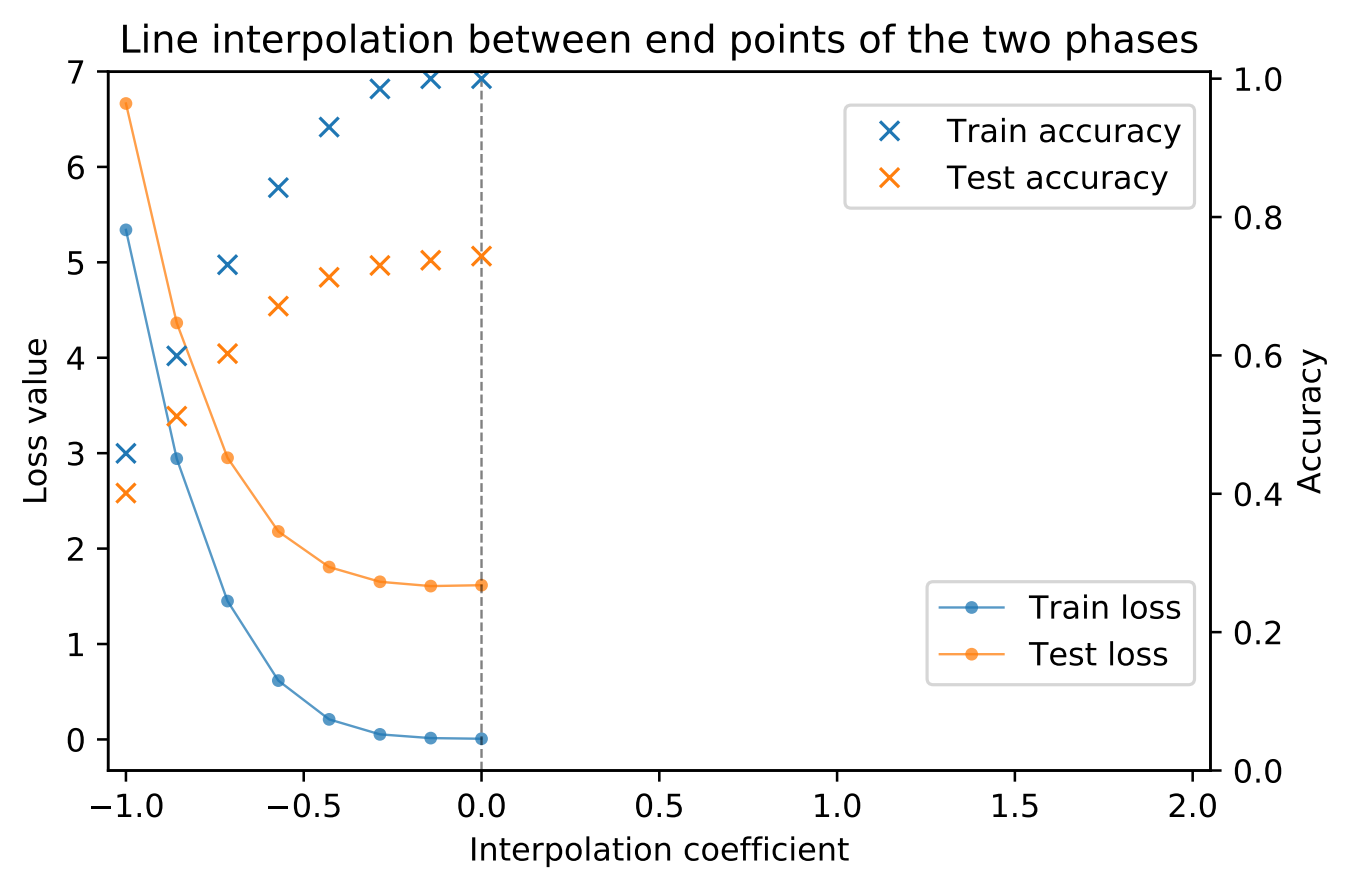
1. Train a large batch CIFAR10 on a *bare* AlexNet
2. At the end point switch to small batch



Searching for sharp basins

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

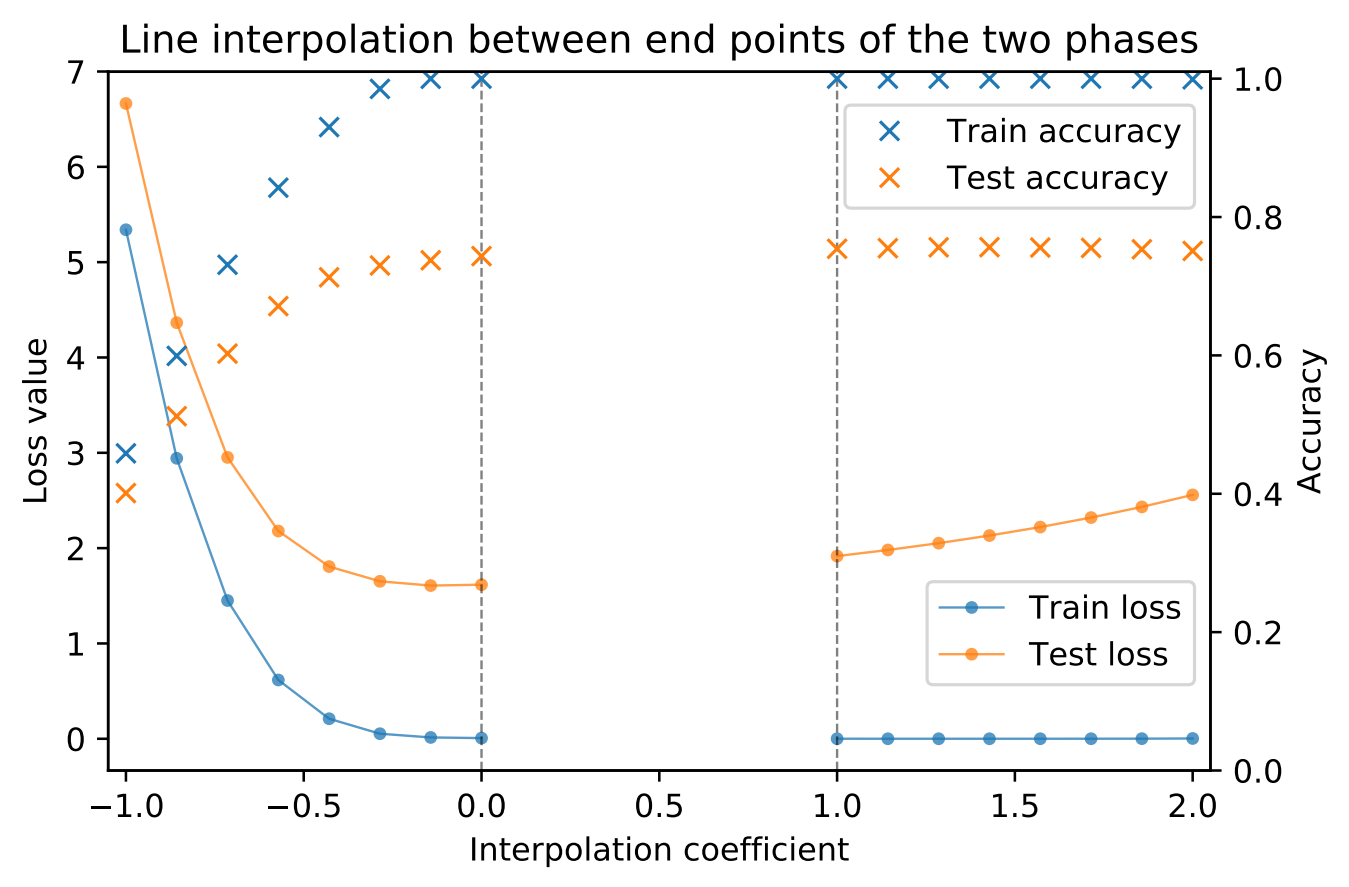
1. Extend a line away from the LB solution



Searching for sharp basins

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

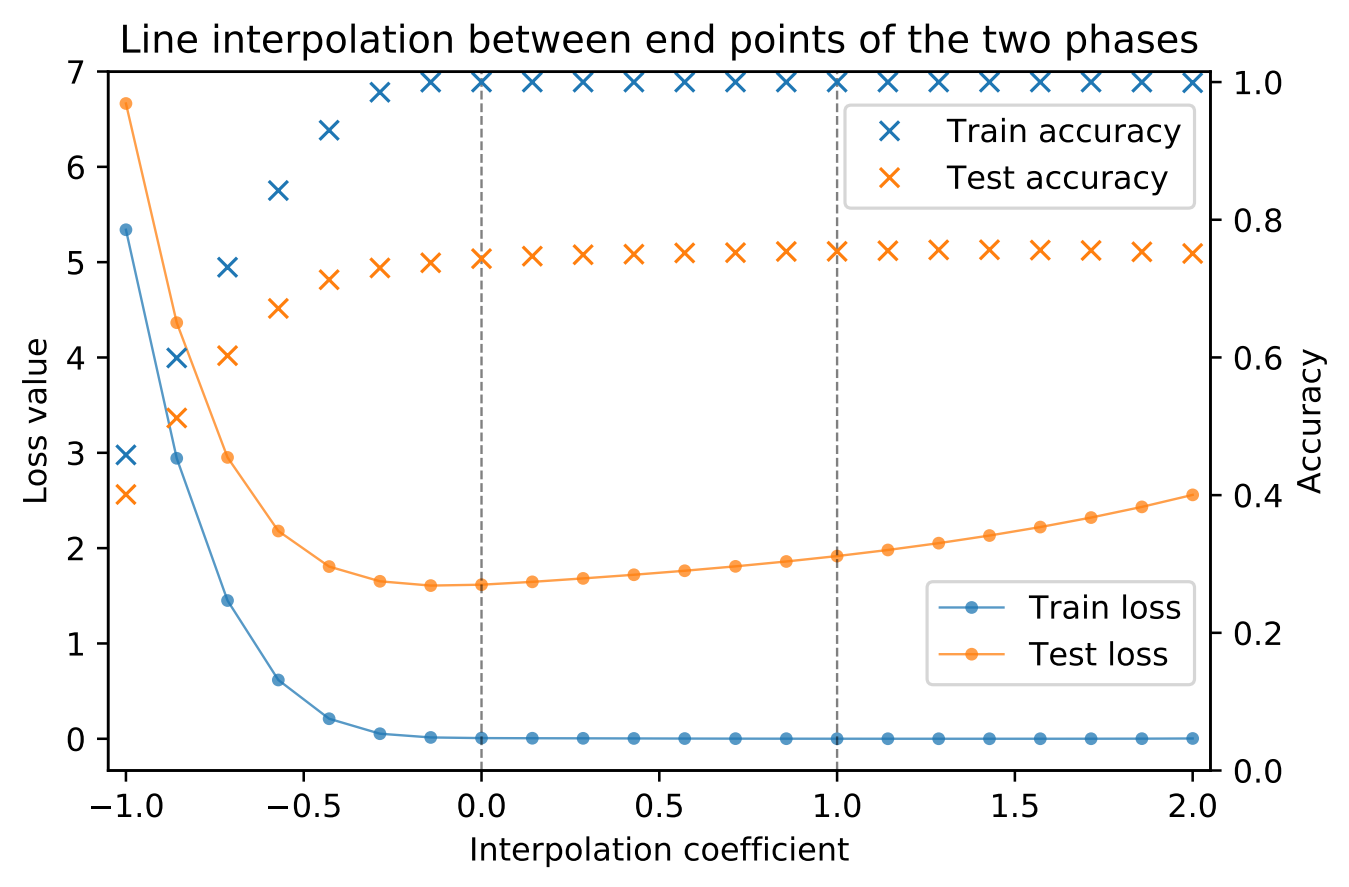
1. Extend a line away from the LB solution
2. Extend a line away from the SB solution



Searching for sharp basins

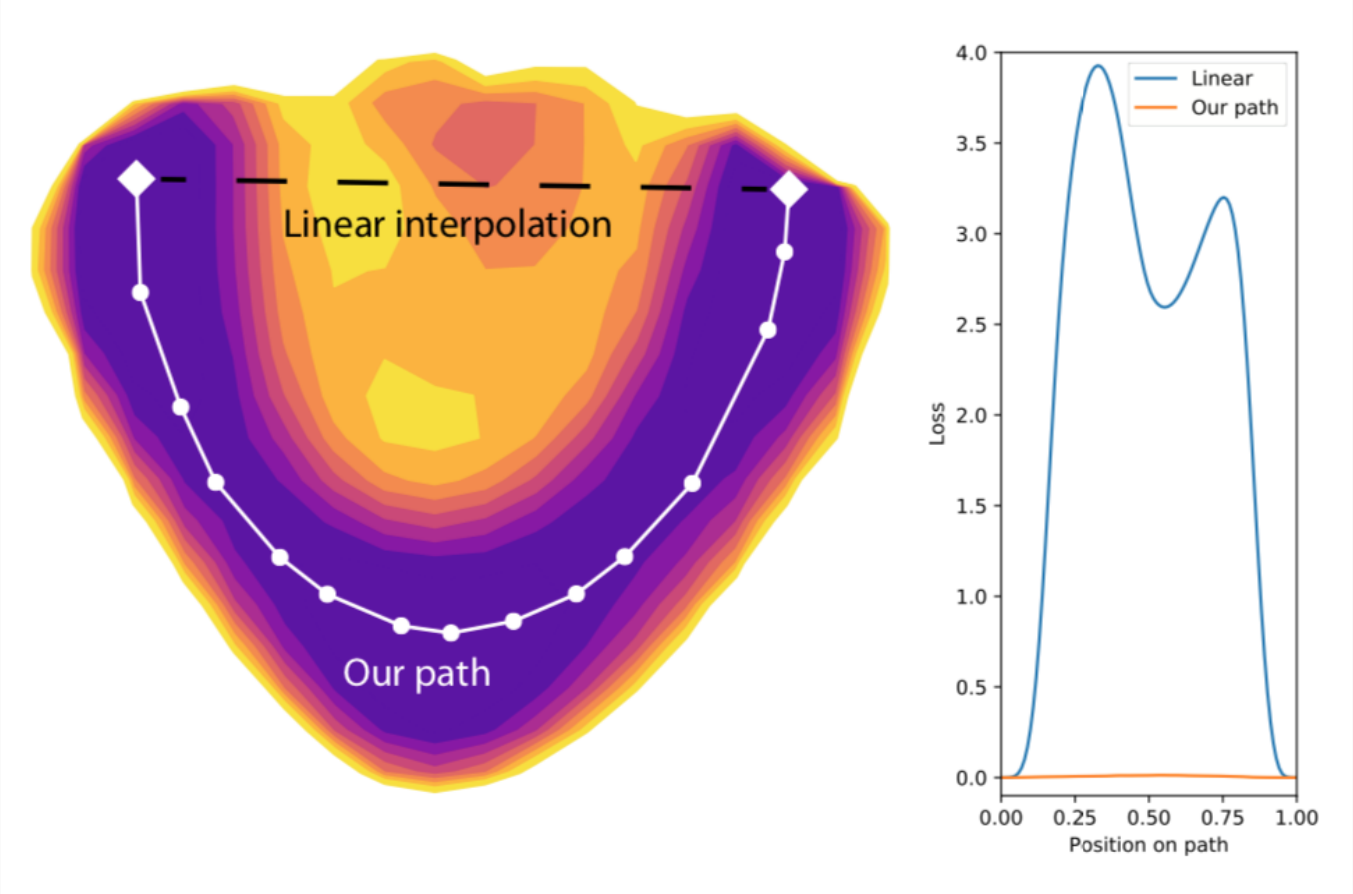
Keep the two points: end of LB training and end of SB continuation.

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



Connecting arbitrary solutions

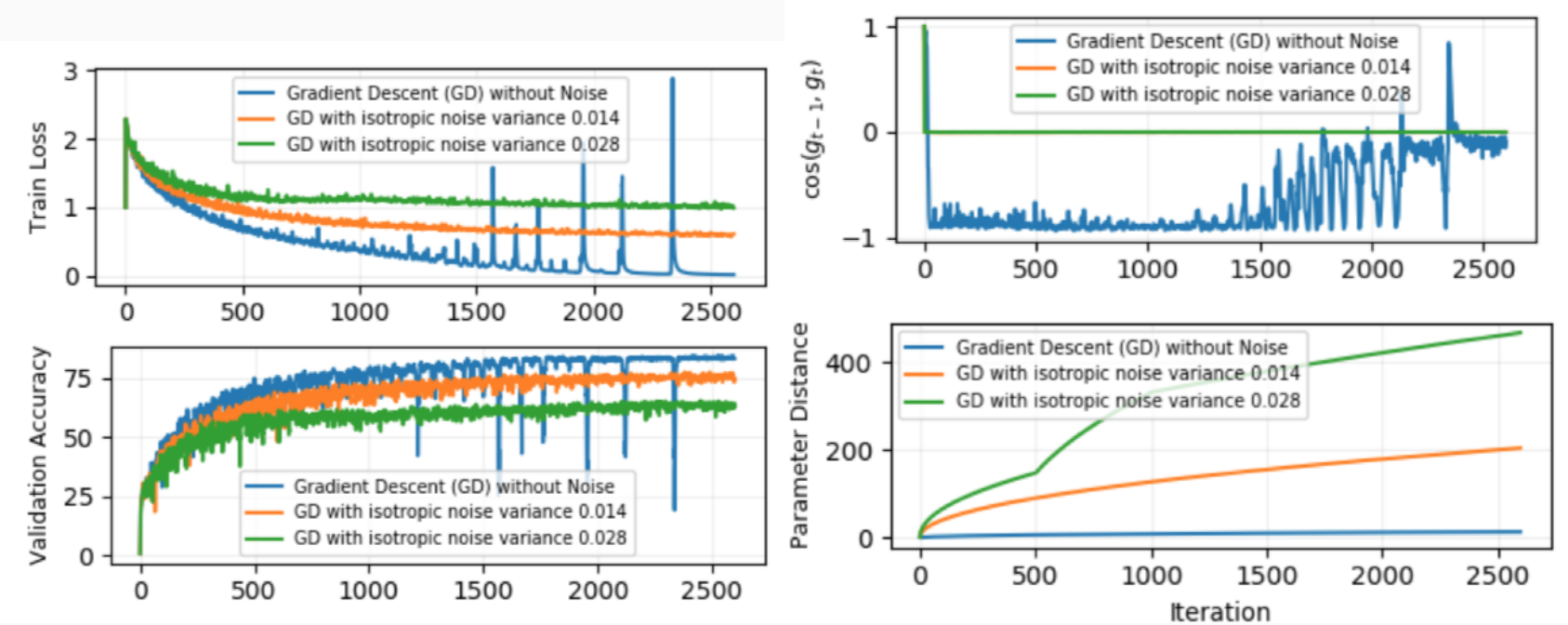
1. Freeman and Bruna 2017: barriers of order $1/M$
2. Draxler et. al. 2018: no barriers between solutions



String method video: `https://cims.nyu.edu/~eve2/string.htm`

What about GD + noise vs SGD

A walk with SGD, Xing et. al. 2018



String method video: `https://cims.nyu.edu/~eve2/string.htm`

Back to the beginning

Does this mean any solution, obtained by any method is in the same basin?

1. Different algorithms
2. Pre-processing vs not pre-processing
3. MSE vs log-loss

- If so, what's the threshold for M?

- Is there an under-parametrized regime in which solutions are disconnected?

The End

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:

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