About diagonal rescaling applied to neural nets

JEAN LAFOND, NICOLAS VASILACHE, LÉON BOTTOU

FACEBOOK AI RESEARCH, NEW YORK

Motivation

- Lots of stochastic gradient (SG) variants for training neural nets.
  - with heuristic justifications,
  - and hard-to-interpret experiments because baselines are different.
- How would they compare with a "*well tuned stochastic gradient*" ?
- What is a "*well tuned stochastic gradient*" ?

  ➔ Initialization tricks.

  ➔ Different stepsizes in different layers/units.

Diagonal rescaling: $w_{t+1} = w_t - \gamma_t D_t g(w_t, \xi_t)$ with D_t diagonal

Alternate Motivation

- The convergence rate of SG algorithms already match the lower bounds for optimization algorithms using noisy first order oracles [Agarwal et al, 2011].

  (well, when both are known…)

- Improved algorithms typically reduce #iterations by a multiplicative constant …
  … with a computational overhead that is also a multiplicative constant.
- Diagonal rescaling has very low computational overhead.

How much can we gain with zero overhead?

Summary

1. A pointless remark?
2. Zero overhead reparametrization.
3. Natural gradient with a twist
4. Problems

1- A pointless remark?

Diagonal Hessian …

- If the Hessian of $f(x,y)$ is diagonal for all (x,y) then $f(x,y) = f_1(x) + f_2(y)$
- Proof:

$$f(x,y) = f(0,0) + \int_0^x \frac{\partial f}{\partial x}(t,0)dt + \int_0^y \frac{\partial f}{\partial y}(x,r)\,dr$$

$$\frac{\partial f}{\partial y}(x,r) = \frac{\partial f}{\partial y}(0,r) + \int_0^x \frac{\partial^2 f}{\partial x \partial y}(t,r)\,dt = \frac{\partial f}{\partial y}(0,r)$$

Diagonal approximation

- Conducting the optimization as if the Hessian of $f(x, y)$ were diagonal

  is (locally) like optimizing a separated function $f(x, y) = f_1(x) + f_2(y)$.

- In general the best way to optimize a separated function

  consists of solving two separate optimization problems.

  (e.g. Newton optimization on $f(x, y) = \frac{1}{2}x^2 + \log(e^y + e^{-y})$ does not work well.)

- Rephrasing for iterative optimization algorithms:

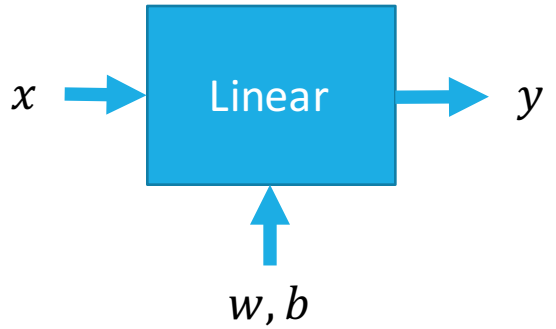
  update x as if y were fixed, update y as if x were fixed, …

Isn't that what a gradient is about?

2- Zero overhead reparametrization

A linear layer

$$\forall j \in \{1 \dots m\} \qquad y_j = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

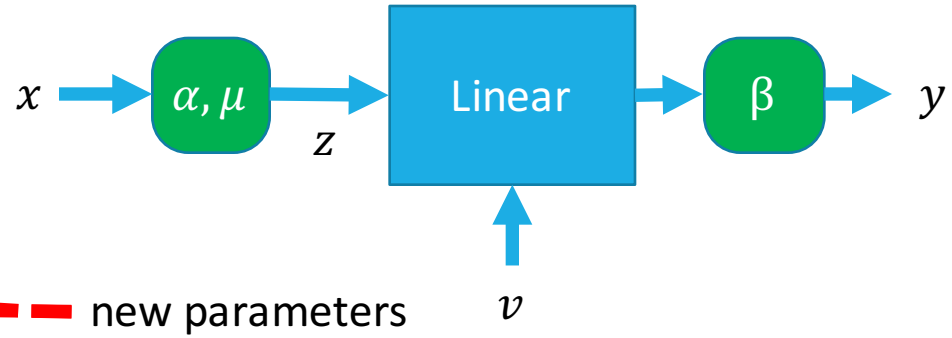


Using the notation $g_j = \dfrac{\partial E}{\partial y_j}$ and the convention $x_0 = 1$,

$$\forall (i,j) \in \{0 \dots n\} \times \{1 \dots m\} \qquad \frac{\partial E}{\partial w_{ij}} = x_i g_j$$

Reparametrization

$$y_j = \beta_j \left(v_{0j} + \sum_{i=1}^{n} \alpha_i (x_i - \mu_i) v_{ij} \right)$$



Compact notation

$$z_0 = 1 \text{ and } z_i = \alpha_i (x_i - \mu_i)$$

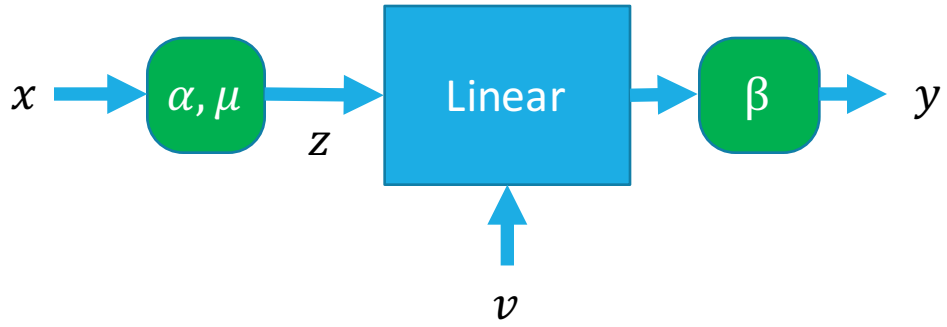
$$y_j = \beta_j \sum_{i=0}^{n} v_{ij} z_i$$

Parameter change

$$w_{ij} = \alpha_i \beta_j v_{ij} \qquad (\text{for } i > 0)$$

$$b_j = \beta_j v_{0j} - \sum_{i=1}^{n} \mu_i w_{ij} = \beta_j v_{0j} - \sum_{i=1}^{n} \mu_i \alpha_i \beta_j v_{ij}$$

SG on the new parameters



$$\delta v_{ij} = \left\langle \frac{\partial E}{\partial v_{ij}} \right\rangle = \langle \beta_j g_j z_i \rangle$$

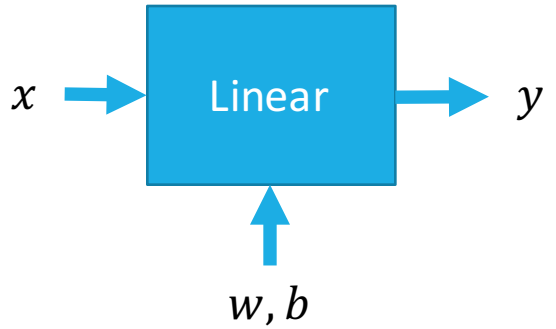
Angle brackets = average over minibatch

SG update is proportional to this quantity.

… expressed on the old parameters

$$\delta w_{ij} = \alpha_i \beta_j \delta v_{ij} \;=\; \left\langle \beta_j^2 g_j \, \alpha_i^2 (x_i - \mu_i) \right\rangle$$

$$\delta b_j = \beta_j \delta v_{0j} - \sum_{i=0}^{n} \mu_i \delta w_{ij} \;=\; \left\langle \beta_j^2 g_j \right\rangle - \sum_{i=1}^{n} \mu_i \delta w_{ij}$$



Remarks

- This is not really diagonal but quasi-diagonal.
- Computational overhead scales like $n + m$, not nm.
- We can choose α_i, μ_i, β_j freely *as long as we change them slowly*.
- How to choose them?

3- Natural gradient with a twist

Classic natural gradient

1. Construct a Riemannian geometry on the optimization domain

$$d(w, w+\delta w) \approx \frac{1}{2}\delta w^T G(w)\ \delta w$$

2. Write steepest descent

$$w_{t+1} = w_t\ + \text{argmin}_{\delta w}\left\{\ \left\langle\frac{\partial E}{\partial w}\right\rangle^T \delta w\ \ \text{s.t.}\ \ \frac{1}{2}\delta w^T G(w_t)\ \delta w\ \leq \eta_t^2\ \right\}$$

3. Use a Lagrange coefficient $1/\gamma_t$

$$w_{t+1} = w_t\ + \text{argmin}_{\delta w}\left\{\ \left\langle\frac{\partial E}{\partial w}\right\rangle^T \delta w\ + \frac{1}{2\gamma_t}\ \delta w^T G(w_t)\ \delta w\ \right\}$$

4. Solve.

$$w_{t+1} = w_t - \gamma_t\ G(w_t)^{-1}\left\langle\frac{\partial E}{\partial w}\right\rangle$$

[looks like (Gauss-)Newton]

Another natural gradient…

- What's the difference between choosing η_t and choosing γ_t ?

$$w_{t+1} = w_t + \text{argmin}_{\delta w} \left\{ \left\langle \frac{\partial E}{\partial w} \right\rangle^T \delta w \quad \text{s.t.} \quad \frac{1}{2} \delta w^T G(w_t)\, \delta w \leq \eta_t^2 \right\}$$

$$w_{t+1} = w_t + \text{argmin}_{\delta w} \left\{ \left\langle \frac{\partial E}{\partial w} \right\rangle^T \delta w + \frac{1}{2\gamma_t} \delta w^T G(w_t)\, \delta w \right\}$$

- Keep solving…

$$w_{t+1} = w_t - \eta_t \frac{1}{\sqrt{\left\langle \frac{\partial E}{\partial w} \right\rangle^T G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle}} G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle$$

Same directions but different lengths.

Singularities

$$w_{t+1} = w_t - \eta_t \frac{1}{\sqrt{\left\langle \frac{\partial E}{\partial w} \right\rangle^T G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle}} G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle$$

Too easily close to zero.

Two ways to improve this…

- Smooth $\left\langle \frac{\partial E}{\partial w} \right\rangle^T G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle \approx \mathbb{E}\left[\frac{\partial E}{\partial w}\right]^T G(w_t)^{-1} \mathbb{E}\left[\frac{\partial E}{\partial w}\right] \leq \mathbb{E}\left[\frac{\partial E}{\partial w}^T G(w_t)^{-1} \frac{\partial E}{\partial w}\right]$
- Add regularization term $\mu > 0$

The pointless remark matters

If we assume that $G(w_t)$ is block-diagonal

$$w_{t+1} = w_t - \eta_t \frac{1}{\sqrt{\mu + \mathbb{E}\left[\frac{\partial E}{\partial w}^T G(w_t)^{-1} \frac{\partial E}{\partial w}\right]}} G(w_t)^{-1} \left\langle \frac{\partial E}{\partial w} \right\rangle$$

Should we compute this scaling coefficient globally or separately for each block ?

Computing them separately changes the global direction of the update…

Sanity check : $G(w) = I$

- Compute scaling coefficients separately for each weight w_{ij}.
- Estimate $\mathbb{E}\left[\frac{\partial E}{\partial w_{ij}}^T \frac{\partial E}{\partial w_{ij}} \right]$ with a running average $R_{ij} \leftarrow (1-\lambda) R_{ij} + \lambda \left\langle \left(\frac{\partial E}{\partial w_{ij}} \right)^2 \right\rangle$

➔ RMSPROP (Tieleman & Hinton 2011)

$$w_{ij} \leftarrow w_{ij} - \frac{\gamma}{\sqrt{\mu + R_{ij}}} \left\langle \frac{\partial E}{\partial w_{ij}} \right\rangle$$

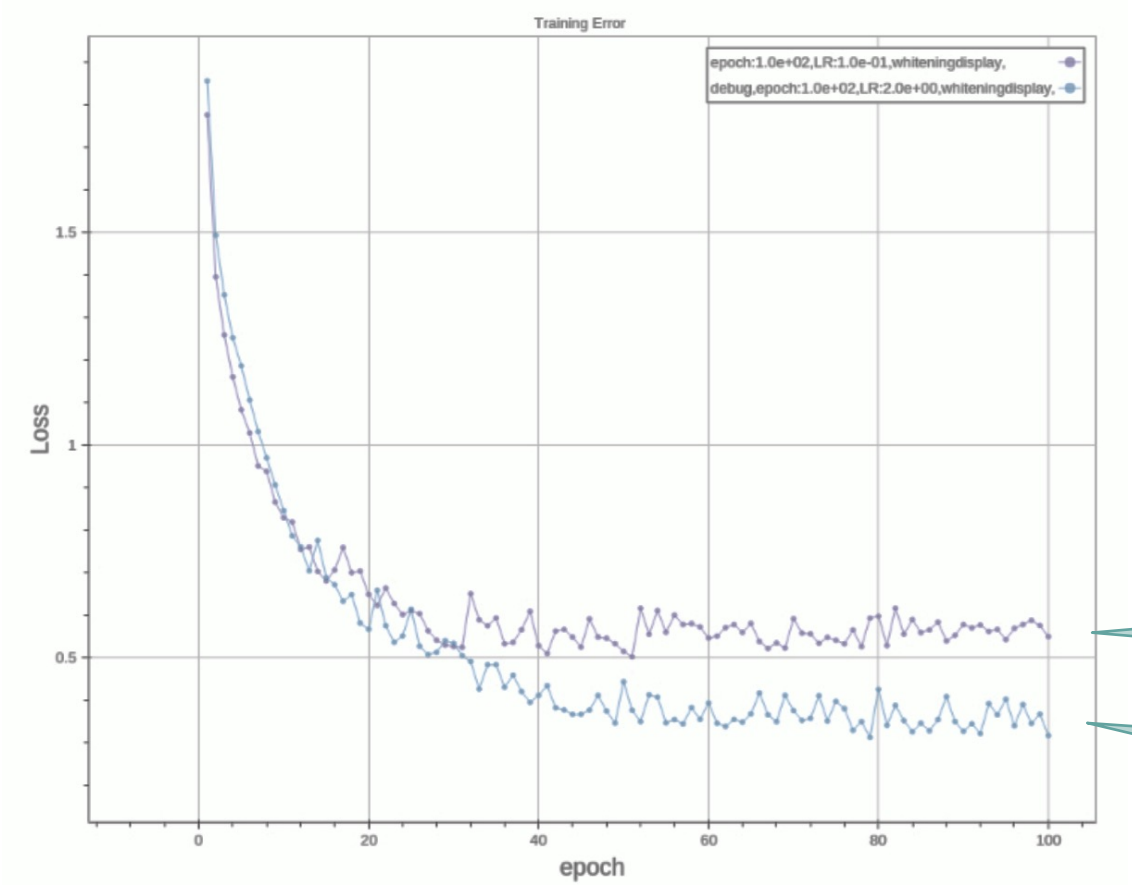
Sanity check : 1990's style network

- Hyperbolic tangent activation function $x_i^{(k+1)} = \tanh\left(x_i^{(k)}\right) \approx \pm 1$
- Gauss-Newton / Empirical Fisher Information : $G(w) = \mathbb{E}\left[\left(\frac{\partial E}{\partial w}\right)\left(\frac{\partial E}{\partial w}\right)^T\right]$
- Block diagonal approximation : $G_j = \mathbb{E}\left[\left(\frac{\partial E}{\partial w_{ij}}\right)\left(\frac{\partial E}{\partial w_{i'j}}\right)\right] = \mathbb{E}\left[g_j^2\, x_i\, x_{i'}\right] \approx \mathbb{E}\left[g_j^2\right] I$
- Compute separate scaling ratio per block (= per neuron)

$$\sqrt{\mathbb{E}\left[\frac{\partial E}{\partial w}^T G(w_t)^{-1} \frac{\partial E}{\partial w}\right]} = \sqrt{\mathbb{E}\left[\frac{\sum_{ii'} g_j^2\, x_i\, x_{i'}}{\mathbb{E}\left[g_j^2\right]}\right]} \approx \sqrt{fanin}$$

Well known heuristic.
Divide stepsizes by
$\sqrt{share \times fanin}$

Divide stepsizes by $\sqrt{share \times fanin}$



- Data : CIFAR 10
  60000 32x32 color images, 10 classes
- Network : CNN
  C6(5x5)-P(2x2)-C16(5x5)-F84-F16-F10
- ReLU : $x_i^{(k+1)} = \max\left(y_i^{(k)}, 0\right)$

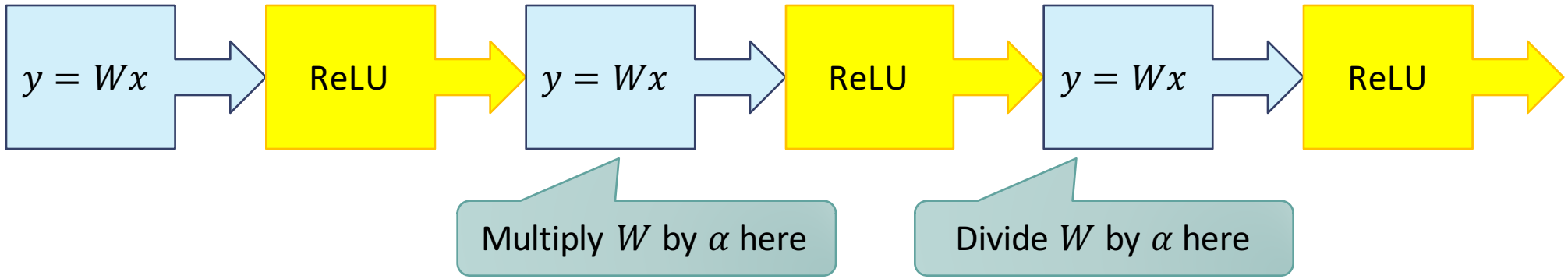
Stochastic gradient
Common stepsize

Per-layer stepsize
$\propto 1/\sqrt{share * fanin}$

Criticism

- With tanh activation, $\mathbb{E}[\![g_j^2 \, x_i \, x_{i'}]\!] \approx \mathbb{E}[g_j^2 \,] \, I$ is a dubious approximation.
- With ReLU activation, this is not credible.

- ReLu networks $x_i^{(k+1)} = \max\left(y_i^{(k)}, 0\right)$ have hyperbolic geometry in parameter space.



Whitening reparametrization

- Make $\mathbb{E}\left[\!\left[g_j^2 \; z_i \, z_{i'} \right]\!\right] \approx \; \mathbb{E}\left[g_j^2 \; \right] I$ with:

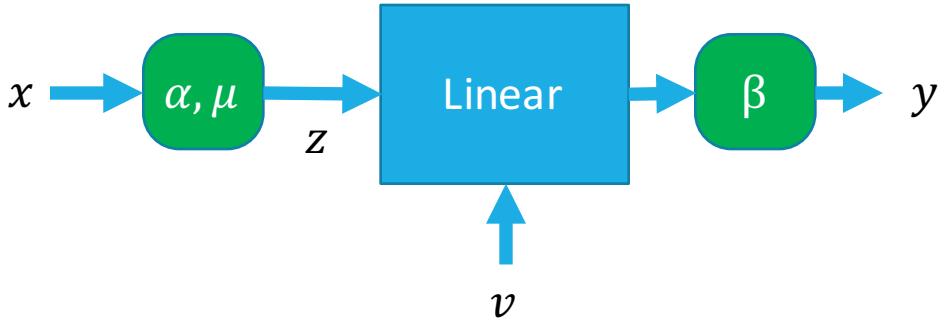
$$\mu_i = \mathbb{E}[x_i] \qquad \alpha_i^2 = \frac{1}{\mathrm{var}[x_i]}$$

estimated with running averages…

- Then take Fisher into account with:

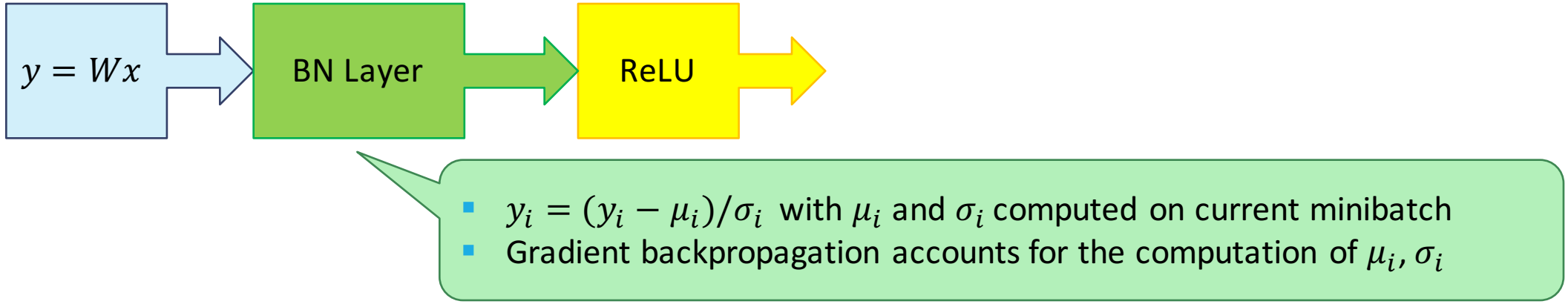
$$\beta_j^2 = \frac{1}{\sqrt{share \times fanin}}$$

(there are cleaner ways to get the same result)



Alternative : batch normalization

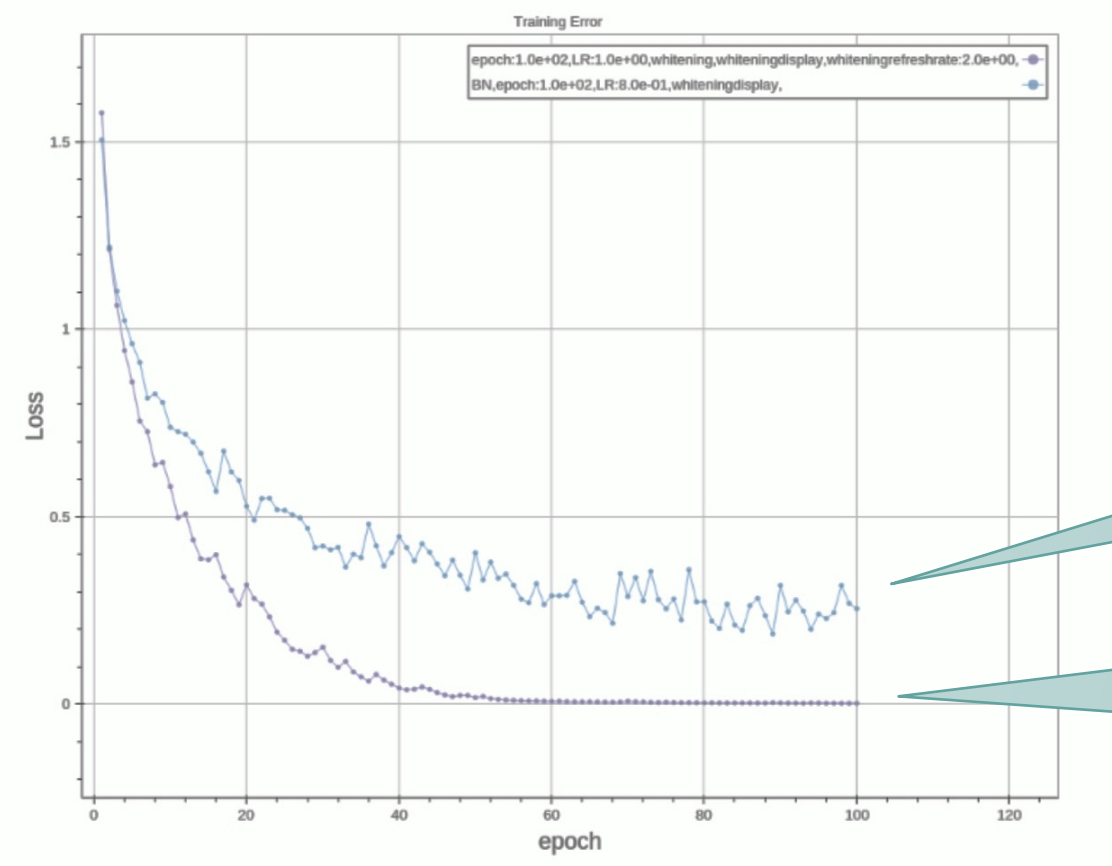
These days, everybody uses batch normalization (Ioffe & Szegedy, 2015)



Batch normalization has strange side effects :

- The network output for a given pattern depends on the other minibatch examples.
- When the minibatch examples are randomly picked, this can be viewed as additional noise…

Batch normalization vs whitening



- Data : CIFAR 10

  60000 32x32 color images, 10 classes

- Network : CNN

  C6(5x5)-P(2x2)-C16(5x5)-F84-F16-F10

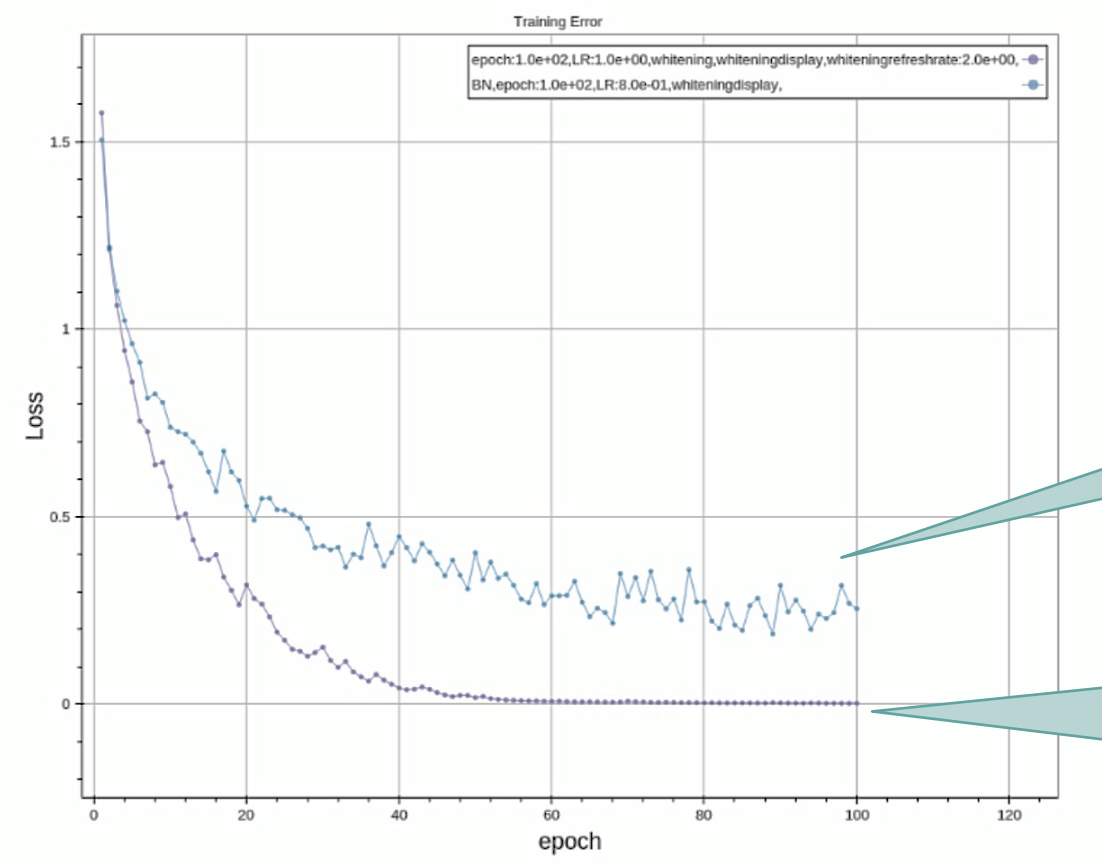
SG with common stepsize and batch normalization

Whitening reparametrization

$\mu_i = E(x_i) \quad \alpha_i^2 = 1/Var(x_i)$

$\beta_j^2 = 1/\sqrt{share * fanin}$

Batch normalization vs whitening



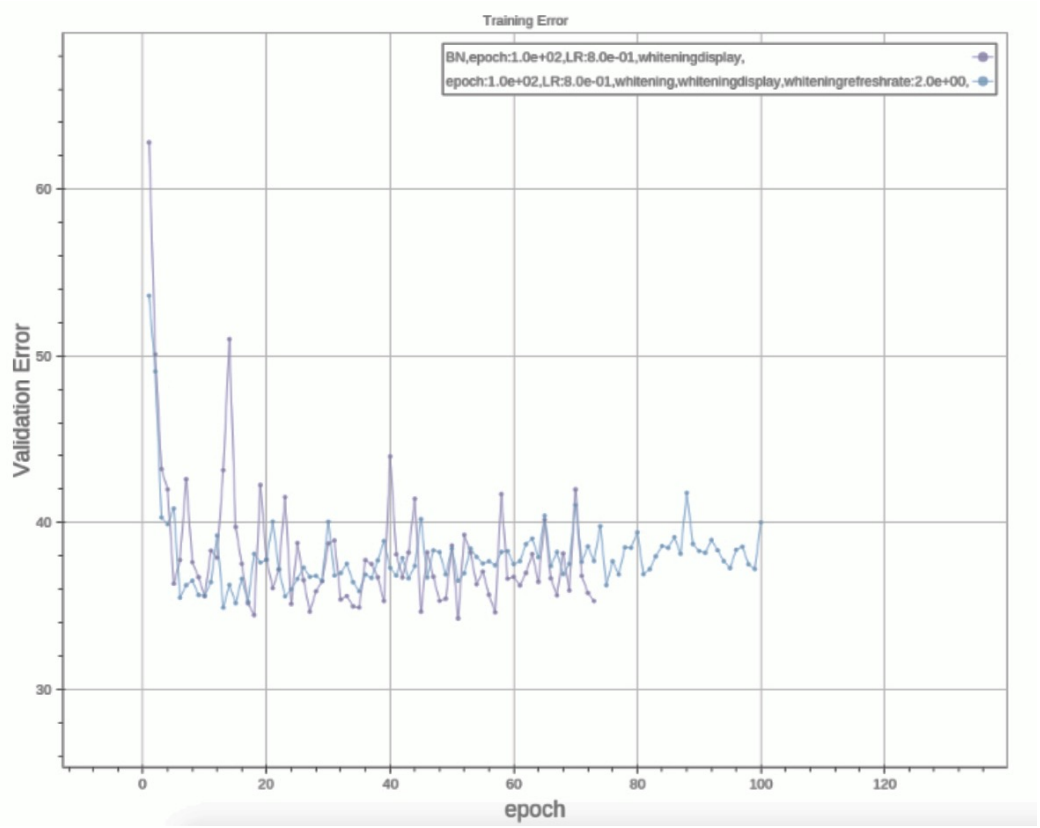
Network structure was copied from Soumith Chintala's Torch tutorial.

Batch normalization seem stuck. (normal SG was stuck even higher)

Whitening quickly reaches zero training cost! (overparametrized network)

Test error sanity check

- Whitening optimizes better
  → but also overfits!
- We could reduce the network size and train even faster…
  → but how to compare?
- Sanity check:
  Use the same network size with L2 regularization
  → No longer overfits.



4- Problems

Scaling up to ImageNet

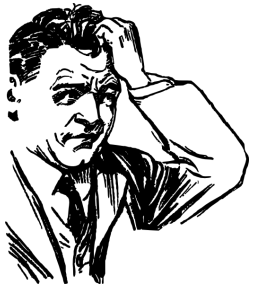
- Data: ImageNet – 1 million 224x224 color images, 1000 classes
- Network: Alexnet (Krizhevsky et al, 2012)

First try 10% of ImageNet (100000 randomly picked examples, 1000 classes)

➔ Same result as CIFAR10: Whitening quickly reaches near zero training cost. BN stuck higher.

Then try 100% of ImageNet (1M examples)

➔ Whitening now trains slower than batch-normalization. Slower than plain SG too!



"This is very strange! With SG, we do not usually expect training error convergence to depend on the training set size."

Investigation

- Good old investigation method : look at the numbers during training.
  This is hard when the network has 60M weights.

- Key observation : during the first epochs, $E(x_i)$ and $Var(x_i)$ can change **very quickly** !

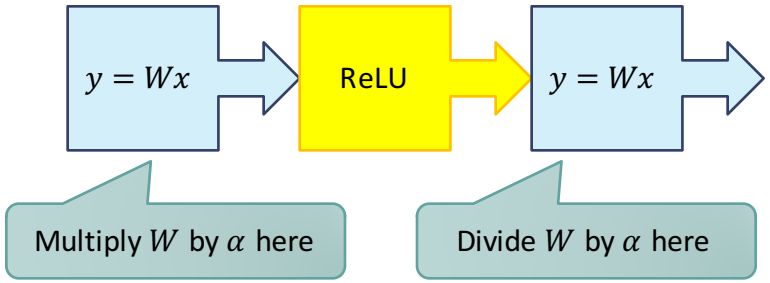
➔ The running estimates of $E(x_i)$ and $Var(x_i)$ are behind.
They are sometimes so wrong that we make a big unwarranted update that takes the y_j in places where the ReLU has no gradient… (plateau)

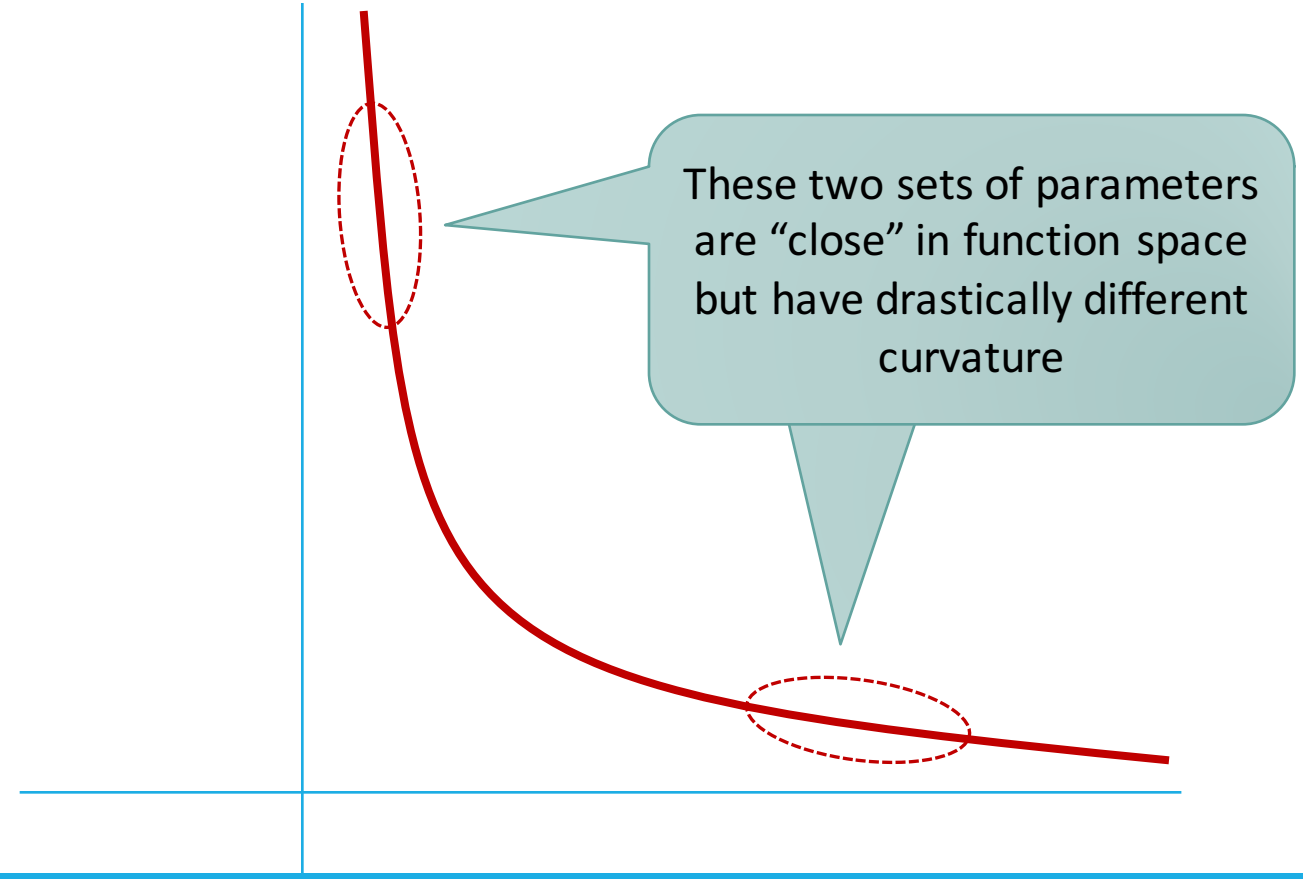
➔ Speeding up the running estimates becomes unstable.

➔ BN does not have this problem because it gets $E(x_i)$ and $Var(x_i)$ on the current minibatch.

Why do statistics change so quickly?

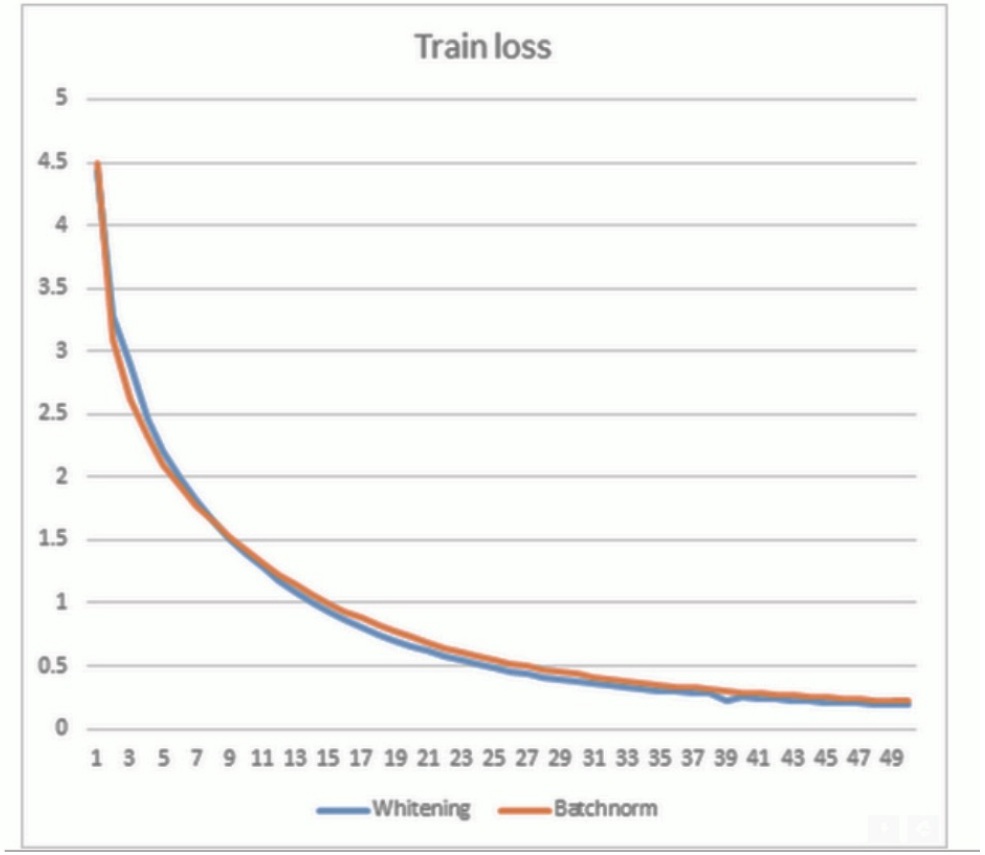
The parameters of a ReLU networks have hyperbolic geometry





Mitigation

- Statistics seem more stable after 2 epochs.
- Running 2 BN epochs before whitening fixes the problem →
- Same speed as BN in iterations, but 20-25% faster in compute time (whitening has less overhead!)
- Looking for a better fix.
  Work in progress...



Conclusion

Conclusions

- We were looking for something *robust, efficient, and principled*.

- This is still work in progress but…

➔ We now understand why RMSPROP works.

➔ We now understand why Batch-Normalization works.