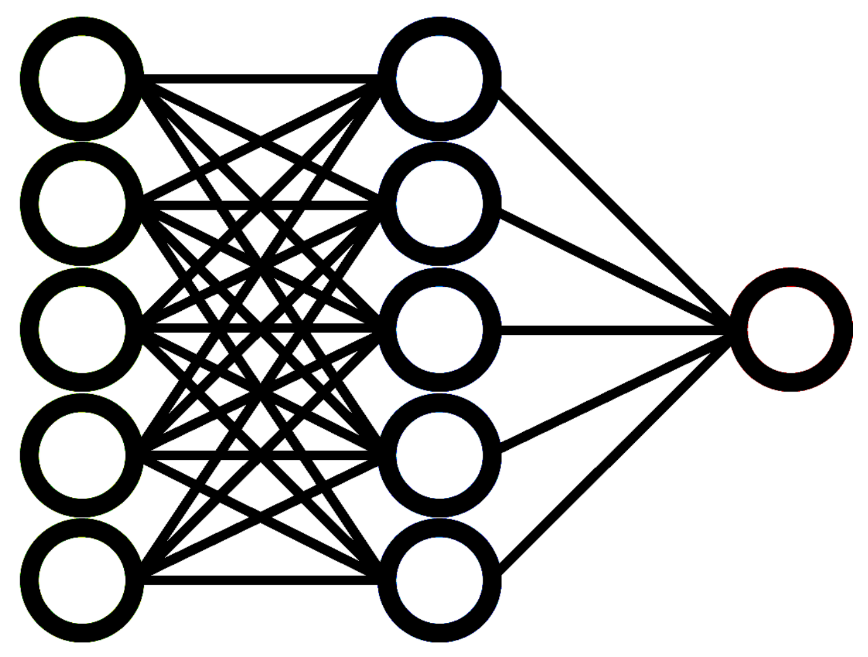
On the distance between two neural networks and the stability of learning

Jeremy Bernstein, Arash Vahdat, Yisong Yue, Ming-Yu Liu

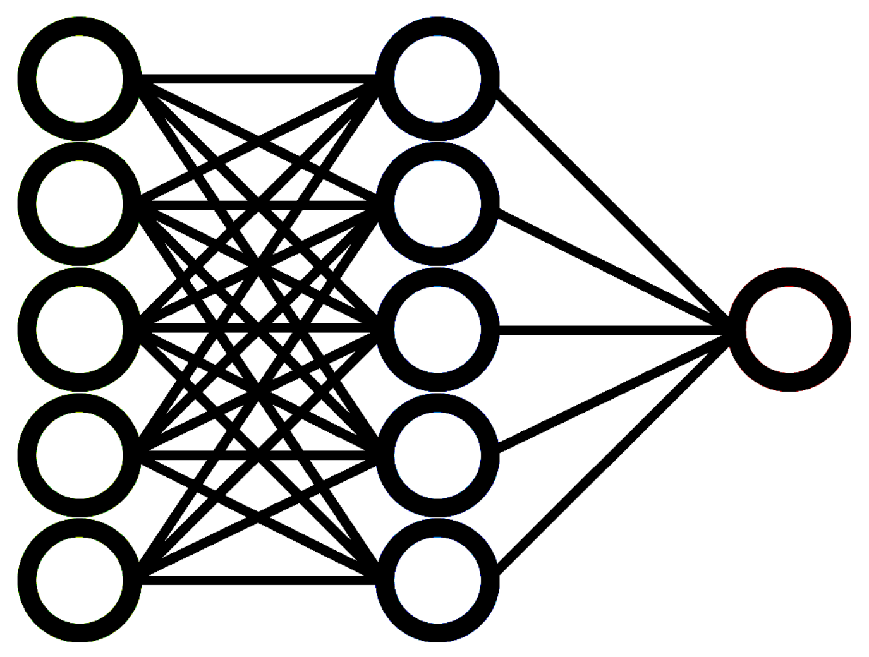
NEURAL INFORMATION PROCESSING SYSTEMS

Distance between networks

How should we measure distance between two neural networks with the same architecture but different weights?



$f(x) = W_2 \cdot \phi(W_1 x)$



$\tilde{f}(x) = \widetilde{W}_2 \cdot \phi(\widetilde{W}_1 x)$

First attempt: flattening

Stack the weights into vectors w and $\widetilde{w}$, and take the norm:

$$\|\widetilde{w} - w\|_2.$$

But this ignores the layered structure of the network! And it ignores the symmetry of $W_1 \to \alpha \cdot W_1$ and $W_2 \to \frac{1}{\alpha} \cdot W_2$.

Second attempt: deep relative trust

Define a new distance function called "deep relative trust":

$$D(\tilde{f}, f) := \prod_{l=1}^{L} \left(1 + \frac{\|\widetilde{W}_l - W_l\|_F}{\|W_l\|_F}\right) - 1.$$

…the relative functional difference satisfies:

$$\frac{\|\tilde{f}(x) - f(x)\|_2}{\|f(x)\|_2} \lessapprox D(\tilde{f}, f).$$

…the Jacobian with respect to the lth hidden layer satisfies:

$$\frac{\|\nabla_l \tilde{f}(x) - \nabla_l f(x)\|_F}{\|\nabla_l f(x)\|_F} \lessapprox D(\tilde{f}, f).$$

$\lessapprox$ hides constants that multiply over layers. The constants depend on matrix condition numbers and the nonlinearity.

Optimisation theory

This panel is intended for optimisation experts. Others should feel free to skip ahead to the Fromage optimiser.

First order optimisation theory relies on having a sensible notion of distance for the function class. How can we use our insights on deep relative trust to this end?

An upper bound on all continuously differentiable functions

Suppose that our loss function $\mathscr{L}(W)$ is continuously differentiable and the parameters fall into L groups. Then the change in loss $\mathscr{L}(W + \Delta W) - \mathscr{L}(W)$ is upper bounded by:

$$-\sum_{l=1}^{L} \|\nabla_{W_l}\mathscr{L}(W)\|_F \|\Delta W_l\|_F \left[\cos\theta_l - \max_{t\in[0,1]} \frac{\|\nabla_{W_l}\mathscr{L}(W_l + t\Delta W_l) - \nabla_{W_l}\mathscr{L}(W_l)\|_F}{\|\nabla_{W_l}\mathscr{L}(W_l)\|_F}\right],$$

where θ_l is the angle between ΔW_l and $-\nabla_{W_l}\mathscr{L}(W)$.

A general descent condition for neural networks

The terms in square brackets compare the relative change in gradient along the step to the cosine of the angle between the step and the negative gradient.

For neural networks, this means that descent is guaranteed provided that for each layer l:

$$\max_{t\in[0,1]} \frac{\|\nabla_{W_l}\mathscr{L}(W_l + t\Delta W_l) - \nabla_{W_l}\mathscr{L}(W_l)\|_F}{\|\nabla_{W_l}\mathscr{L}(W_l)\|_F} < \cos\theta_l .$$

Learning rate tuning in gradient descent would then amount to tuning the size of ΔW_l to match this condition.

Modelling the neural network gradient

The neural network gradient depends on network Jacobians and subnetwork outputs. This suggests using deep relative trust to model the relative change in gradient.

A new learning rule

There is a very simple learning rule that respects the structure of deep relative trust. For each layer l, it sets:

$$W_l \leftarrow W_l - \eta \cdot \frac{\|W_l\|_F}{\|g_l\|_F} \cdot g_l. \qquad \text{(LARS)}$$

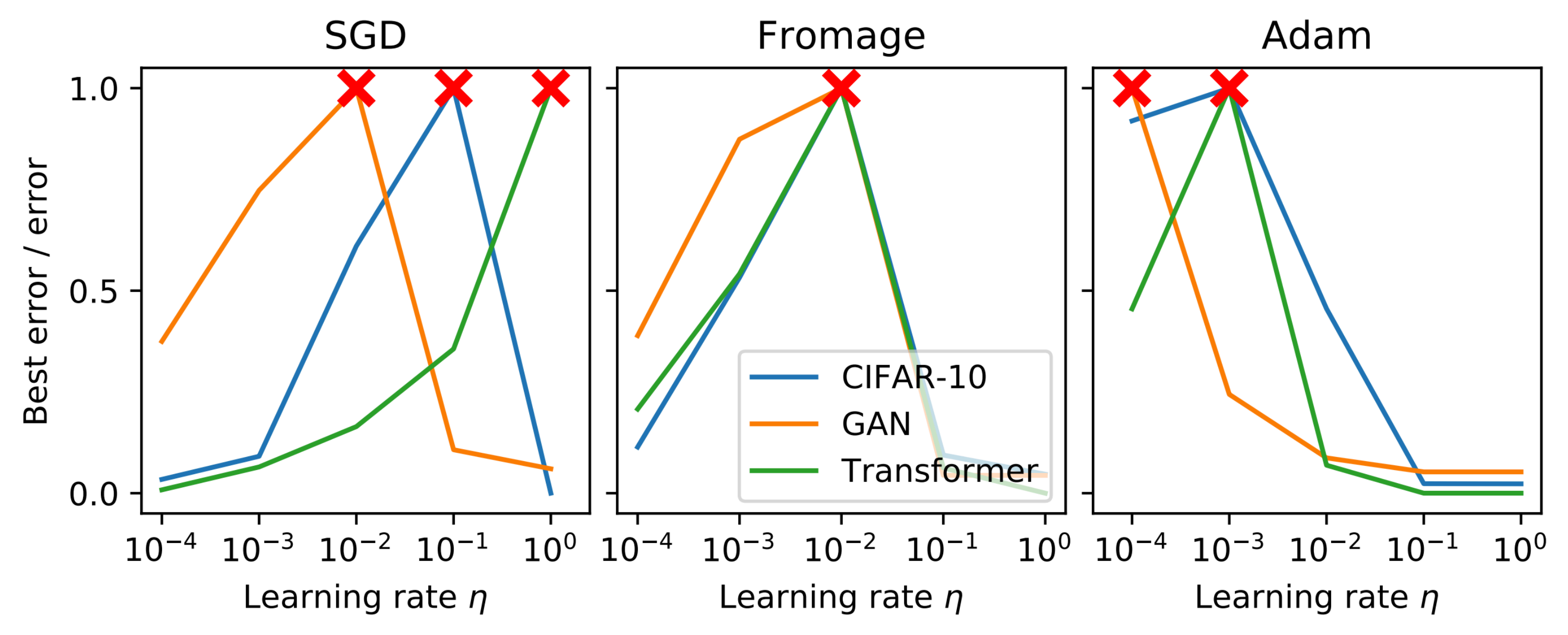
The learning rate η directly controls the relative change to each layer. LARS was proposed in 2017 by Yang You, Igor Gitman and Boris Ginsburg on empirical grounds.

Frobenius matched gradient descent

We propose a minor correction to LARS that stabilises training in scale invariant networks:

$$W_l \leftarrow \frac{1}{\sqrt{1+\eta^2}} \left(W_l - \eta \cdot \frac{\|W_l\|_F}{\|g_l\|_F} \cdot g_l\right). \qquad \text{(Fromage)}$$

We found that Fromage worked across many deep learning experiments with the same learning rate of $\eta = 0.01$.



Also, the training performance of Fromage was often an order of magnitude better than that of SGD.

Benchmark	SGD η	Fromage η	Adam η	SGD	Fromage	Adam
CIFAR-10	0.1	0.01	0.001	$(1.5 \pm 0.2) \times 10^{-4}$	$\mathbf{(2.5 \pm 0.5) \times 10^{-5}}$	$(6 \pm 3) \times 10^{-5}$
ImageNet	1.0	0.01	0.001	2.020 ± 0.003	$\mathbf{2.001 \pm 0.001}$	2.02 ± 0.01
GAN	0.01	0.01	0.0001	34 ± 2	$\mathbf{16 \pm 1}$	23.7 ± 0.7
Transformer	1.0	0.01	0.001	150.0 ± 0.3	66.1 ± 0.1	$\mathbf{36.8 \pm 0.1}$

bernstein@caltech.edu

github.com/jxbz/fromage

youtu.be/dUm8hZFtbLg