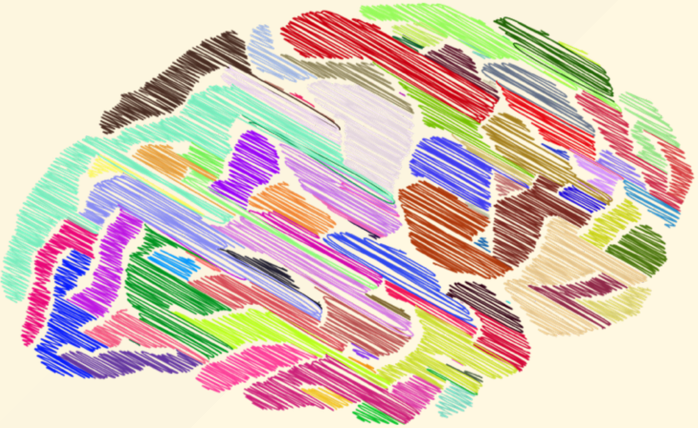
Network Optimisation



Jeremy Bernstein
bernstein@caltech.edu

Agenda for today

1. State of deep learning
2. Optimisation theory
3. Perturbation theory approach

what is the Adam optimiser?

different models of trust in Taylor expansions

an optimisation model based on the neural network structure.
(my research, so be critical!)

The stages of theory

1. Empirical exploration many (most?) ideas that work well in deep learning
2. Modelling
3. Derivation
4. Empirical validation

(In my opinion) deep learning optimisation theory is still at the modelling stage.

— designing theoretically principled deep learning optimisers requires building faithful yet analytically tractable models of neural network loss functions.

Stochastic gradient descent

This is where many treatments of deep learning begin.

loss function $$\boxed{\mathcal{L}(w) = \sum_{i=1}^{n} \underbrace{l_i(w)}}$$ loss on datapoint i

full gradient $$\boxed{\nabla_w \mathcal{L}(w) = \sum_{i=1}^{n} \nabla_w l_i(w)} =: g \text{ (shorthand)}$$

stochastic gradient $$\boxed{\sum_{i \in B} \nabla_w l_i(w)} =: \tilde{g} \text{ (shorthand)}$$

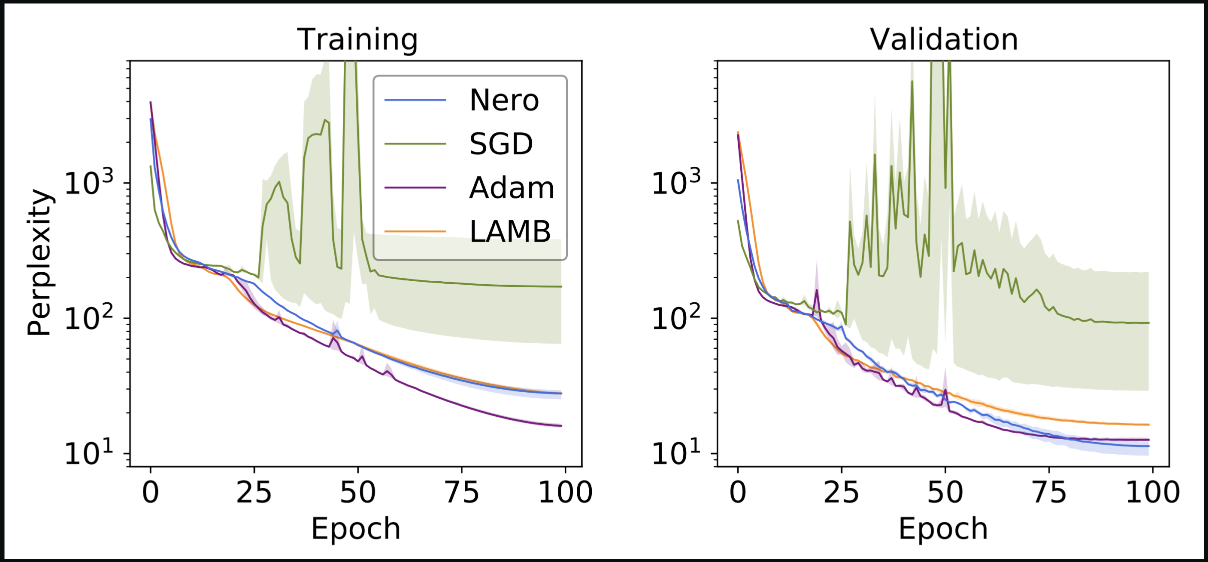
random "mini batch" of datapoints

Stochastic gradient descent perturbs parameters via

$$\boxed{w \longrightarrow w - \eta \tilde{g}}$$

where η is the learning rate. It makes sense, BUT...

... Other optimisers are often substantially better than SGD...



training a transformer network on a machine translation task.

SGD is unstable and Adam works much better.

History of Adam

Rprop → RMSprop → Adam

A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm

Martin Riedmiller   Heinrich Braun

1993

from Hinton's Coursera class

Neural Networks for Machine Learning

Lecture 6e

rmsprop: Divide the gradient by a running average of its recent magnitude

2012

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

2015

Rprop optimiser

i.e. not stochastic

Hinton: Rprop is "used for full batch learning"

gradient descent

$$\boxed{\Delta W = -\eta\, g}$$

global learning rate

can we trust the scale of g to be useful?

Rprop

$$\boxed{\Delta W = -\begin{bmatrix} \eta_1 \,\mathrm{sign}(g_1) \\ \eta_2 \,\mathrm{sign}(g_2) \\ \vdots \\ \eta_d \,\mathrm{sign}(g_d) \end{bmatrix}}$$

Rprop takes the componentwise gradient sign, so it throws away scale information from the gradient.

It also has per component learning rates, with a rule for adapting them over time — but that's not important here.

RMSprop optimiser

RMSprop adapts Rprop to work in the minibatch setting.

Imagine a case where a weight receives 10 gradients of $+0.1$ followed by 1 gradient of -1.0.

The average gradient is $\boxed{10 \times 0.1 + 1 \times (-1) = 0}$ so we don't want the weight to move, but Rprop would move the weight a lot.

RMSprop removes the overall gradient scale while retaining relative scale information between successive stochastic gradients

RMSprop $$\boxed{\Delta W = -\eta \frac{\tilde{g}}{\text{RMS}(\tilde{g})}}$$ (all operations componentwise)

$\text{RMS}(\tilde{g})$ denotes the root mean square gradient over iterations:

$$\boxed{\text{RMS}(\tilde{g}) := \sqrt{(1-\beta)\left[\tilde{g}_t^2 + \beta \tilde{g}_{t-1}^2 + \beta^2 \tilde{g}_{t-2}^2 + \dots\right]}}$$ $\beta \in (0,1)$

Adam optimiser

Adam says, why only smooth the divisor with an exponential moving average?

RMSprop $\boxed{\Delta W = -\eta \, \frac{\tilde{g}}{\mathrm{RMS}(\tilde{g})}}$ (all operations componentwise)

Adam $\boxed{\Delta W = -\eta \, \frac{\mathrm{EMA}(\tilde{g})}{\mathrm{RMS}(\tilde{g})}}$ (all operations componentwise)

where EMA means "exponential moving average":

$$\boxed{\mathrm{EMA}(\tilde{g}) = (1-\beta)\left[g_t + \beta g_{t-1} + \beta^2 g_{t-2} + \dots\right]} \quad \beta \in (0,1)$$

Adam also has a trick for "warming up" the moving average at the start, but that's not important here.

Optimiser zoo

Hinton in 2012: "we really don't have nice clear cut advice for how to train a neural net..... think how much better neural nets will work once we've got this sorted out."

an (incomplete) list of optimisers proposed since then.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KFAC	(Martens & Grosse, 2015)
AdaBatch	(Devarakonda et al., 2017)	KFLR/KFRA	(Botev et al., 2017)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBelief	(Zhuang et al., 2020)	LAMB	(You et al., 2020)
AdaBlock	(Yun et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaBound	(Luo et al., 2019)	LARS	(You et al., 2017)
AdaComp	(Chen et al., 2018)	LookAhead	(Zhang et al., 2019)
Adadelta	(Zeiler, 2012)	M-SVAG	(Balles & Hennig, 2018)
Adafactor	(Shazeer & Stern, 2018)	MAS	(Landro et al., 2020)
AdaFix	(Bae et al., 2019)	MEKA	(Chen et al., 2020b)
AdaFom	(Chen et al., 2019a)	MTAdam	(Malkiel & Wolf, 2020)
AdaFTRL	(Orabona & Pál, 2015)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
Adagrad	(Duchi et al., 2011)	Nadam	(Dozat, 2016)
ADAHESSIAN	(Yao et al., 2020)	NAMSB/NAMSG	(Chen et al., 2019b)
Adai	(Xie et al., 2020)	ND-Adam	(Zhang et al., 2017)
AdaLoss	(Teixeira et al., 2019)	Nesterov	(Nesterov, 1983)
Adam	(Kingma & Ba, 2015)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
Adam^{+}	(Liu et al., 2020b)	NosAdam	(Huang et al., 2019)
AdamAL	(Tao et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdaMax	(Kingma & Ba, 2015)	Padam	(Chen et al., 2020a)
AdamBS	(Liu et al., 2020c)	PAGE	(Li et al., 2020b)
AdamNC	(Reddi et al., 2018)	PAL	(Mutschler & Zell, 2020)
AdaMod	(Ding et al., 2019)	PolyAdam	(Orvieto et al., 2019)
AdamP	(Heo et al., 2020)	Polyak	(Polyak, 1964)
AdamT	(Zhou et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdamW	(Loshchilov & Hutter, 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdamX	(Tran & Phong, 2019)	PStorm	(Xu, 2020)
ADAS	(Eliyahu, 2020)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaS	(Hosseini & Plataniotis, 2020)	RAdam	(Liu et al., 2020a)
AdaScale	(Johnson et al., 2020)	Ranger	(Wright, 2020b)
AdaSGD	(Wang & Wiens, 2020)	RangerLars	(Grankin, 2020)
AdaShift	(Zhou et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AdaSqrt	(Hu et al., 2019)	RMSterov	(Choi et al., 2019)
Adathm	(Sun et al., 2019)	S-SGD	(Sung et al., 2020)
AdaX/AdaX-W	(Li et al., 2020a)	SAdam	(Wang et al., 2020b)
AEGD	(Liu & Tian, 2020)	Sadam/SAMSGrad	(Tong et al., 2019)
ALI-G	(Berrada et al., 2020)	SALR	(Yue et al., 2020)
AMSBound	(Luo et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
AMSGrad	(Reddi et al., 2018)	SDProp	(Ida et al., 2017)
ArmijoLS	(Vaswani et al., 2019)	SGD	(Robbins & Monro, 1951)
ARSG	(Chen et al., 2019b)	SGD-BB	(Tan et al., 2016)
AvaGrad	(Savarese et al., 2019)	SGD-G2	(Ayadi & Turinici, 2020)
BAdam	(Salas et al., 2018)	SGDM	(Liu & Luo, 2020)
BGAdam	(Bai & Zhang, 2019)	SGDP	(Heo et al., 2020)
BRMSProp	(Aitchison, 2020)	SGDR	(Loshchilov & Hutter, 2017)
BSGD	(Hu et al., 2020)	SHAdagrad	(Huang et al., 2020)
C-ADAM	(Tutunov et al., 2020)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
CADA	(Chen et al., 2020c)	SignAdam++	(Wang et al., 2019a)
Cool Momentum	(Borysenko & Byshkin, 2020)	SignSGD	(Bernstein et al., 2018)
CProp	(Preechakul & Kijsirikul, 2019)	SKQN/S4QN	(Yang et al., 2020)
Curveball	(Henriques et al., 2019)	SM3	(Anil et al., 2019)
Dadam	(Nazari et al., 2019)	SMG	(Tran et al., 2020)
DeepMemory	(Wright, 2020a)	SNGM	(Zhao et al., 2020)
DiffGrad	(Dubey et al., 2020)	SoftAdam	(Fetterman et al., 2019)
EAdam	(Yuan & Gao, 2020)	SRSGD	(Wang et al., 2020a)
EKFAC	(George et al., 2018)	SWATS	(Keskar & Socher, 2017)
Eve	(Hayashi et al., 2018)	SWNTS	(Chen et al., 2019c)
Expectigrad	(Daley & Amato, 2020)	TAdam	(Ilboudo et al., 2020)
FRSGD	(Wang & Ye, 2020)	TEKFAC	(Gao et al., 2020)
GADAM	(Zhang & Gouza, 2018)	VAdam	(Khan et al., 2018)
Gadam	(Granziol et al., 2020)	VR-SGD	(Shang et al., 2020)
GOLS-I	(Kafka & Wilke, 2019)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Schmidt, Schneider & Hennig 2021

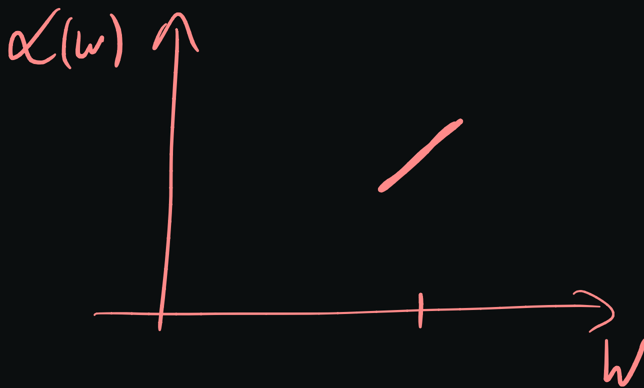
Agenda for today

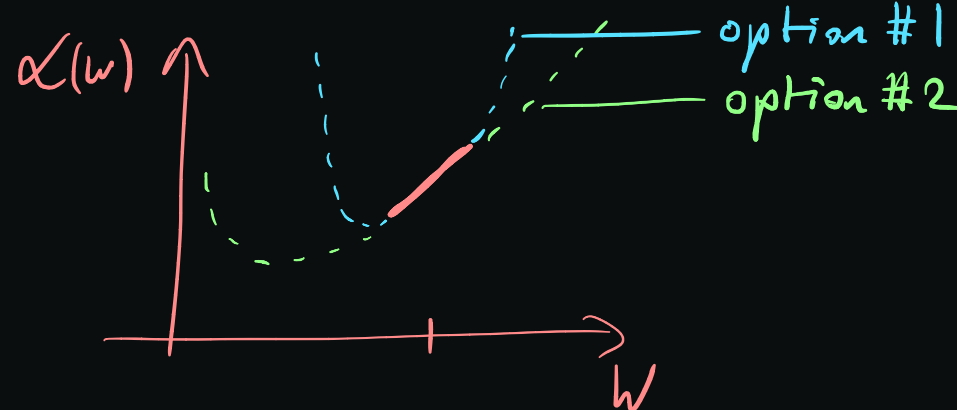
what does optimisation theory have to say?

1. State of deep learning
2. $\boxed{\text{Optimisation theory}}$
3. Perturbation theory approach

Models of smoothness

I computed the full batch gradient $g := \nabla_w \mathcal{L}(w)$ of my loss function $\mathcal{L}(w)$, and it looks like this:





When I choose my step size, I need to know, does $\mathcal{L}(w)$ behave like option #1 or option #2?

the same question:

- how fast does the gradient change?
- how far can I trust the first order Taylor expansion?

Taylor expansions

First order Taylor expansion Higher order terms

$$\mathcal{L}(w + \Delta w) = \boxed{\mathcal{L}(w) + g^T \Delta w} + \boxed{\tfrac{1}{2} \Delta w^T H \Delta w + \ldots}$$

gradient

$$g := \nabla_w \mathcal{L}(w)$$

Hessian

$$H_{ij} := \frac{\partial^2 \mathcal{L}}{\partial w_i \partial w_j}$$

When picking the step size, we want to know how large we can set Δw before the higher order terms start to dominate the first order Taylor expansion.

Optimisation theory —— let's model the higher order terms.

Model #1: Lipschitz smoothness

$$\mathcal{L}(W + \Delta W) \leq \underbrace{\boxed{\mathcal{L}(W) + g^T \Delta W}}_{\text{First order Taylor expansion}} + \underbrace{\boxed{\frac{L}{2} \times \|\Delta W\|_F^2}}_{\text{quadratic penalty}}$$

Lipschitz smoothness models the higher order terms as being bounded by a quadratic term in ΔW.

L is a constant which adjusts the strength of the penalty.

Let's compute the ΔW that minimises this model.

Differentiating with respect to ΔW:

$$g + L \Delta W = 0$$

$$\implies \boxed{\Delta W = -\frac{1}{L} g}$$

this is gradient descent with step size $1/L$

— the steeper the penalty, the smaller the stepsize.

Lipschitz smooth model: $\boxed{\mathcal{L}(W+\Delta W) \leq \mathcal{L}(W) + g^{T}\Delta W + \frac{L}{2}\|\Delta W\|_F^2}$

Convergence rates: deterministic

Our algorithm is gradient descent with step size $1/L$:

$$\boxed{W_{k+1} = W_k - \frac{1}{L} g_k} \qquad g_k := \nabla_W \mathcal{L}(W_k)$$

$\implies$ The change in loss in one step is bounded like:

$$\mathcal{L}(W_{k+1}) - \mathcal{L}(W_k) \leq -\frac{1}{L}\|g_k\|_F^2 + \frac{L}{2} \times \frac{1}{L^2}\|g_k\|_F^2 = -\frac{1}{2L}\|g_k\|_F^2$$

$\Leftrightarrow$ in words: when the gradient is large, the loss will decrease a lot.

Now, assuming the loss is nonnegative, the total possible improvement is bounded, meaning the algorithm must find a point with small gradient:

$$\underbrace{\mathcal{L}(W_0)}_{\text{total possible improvement}} \geq \underbrace{\mathcal{L}(W_0) - \mathcal{L}(W_K)}_{\text{improvement from step 0 to step } K} = \sum_{k=0}^{K-1} \underbrace{\mathcal{L}(W_k) - \mathcal{L}(W_{k+1})}_{\text{improvement from step } k \text{ to step } k+1} \geq \sum_{k=0}^{K-1} \frac{1}{2L}\|g_k\|_F^2$$

$\implies$ convergence rate $\boxed{\frac{1}{K}\sum_{k=0}^{K-1}\|g_k\|_F^2 \leq \frac{2L\,\mathcal{L}(W_0)}{K}}$

in words: the average gradient norm over iterations decays $\sim 1/K$.

Lipschitz smooth model: $\boxed{\mathcal{L}(W+\Delta W) \leq \mathcal{L}(W) + g^T \Delta W + \frac{L}{2}\|\Delta W\|_F^2}$

Convergence rates: stochastic

Now let's consider stochastic gradient descent

$$\boxed{W_{k+1} = W_k - \frac{1}{L}\tilde{g}_k}$$

NOISE MODEL:

$$\mathbb{E}[\tilde{g}_k] = g_k$$

$$\mathbb{E}[\|\tilde{g}_k - g_k\|_F^2] \leq \sigma^2$$

$\Rightarrow$ The change in loss in one step is bounded like:

$$\mathcal{L}(W_{k+1}) - \mathcal{L}(W_k) \leq -\frac{1}{L} g_k^T \tilde{g}_k + \frac{L}{2}\frac{1}{L^2}\|\tilde{g}_k\|_F^2$$

$= \|g_k\|^2 + \|\tilde{g}_k - g_k\|_F^2 + 2 g_k^T(\tilde{g}_k - g_k)$

$\Rightarrow$ The expected improvement in one step is bounded like:

$$\mathbb{E}[\mathcal{L}(W_{k+1}) - \mathcal{L}(W_k) \mid W_k] \leq -\frac{1}{L}\|g_k\|_F^2 + \frac{1}{2L}\|g_k\|_F^2 + \frac{1}{2L}\sigma^2 = \frac{1}{2L}\left(-\|g_k\|_F^2 + \sigma^2\right)$$

$$\mathbb{E}[\mathcal{L}(W_{k+1}) - \mathcal{L}(W_k)] \leq \frac{1}{2L}\left(-\mathbb{E}\|g_k\|_F^2 + \sigma^2\right)$$

Then the total possible improvement (for a nonnegative loss) satisfies:

$$\mathcal{L}(W_0) \geq \mathcal{L}(W_0) - \mathbb{E}\mathcal{L}(W_K) = \sum_{k=0}^{K-1}\mathbb{E}[\mathcal{L}(W_k) - \mathcal{L}(W_{k+1})] \geq \sum_{k=0}^{K-1}\frac{1}{2L}\left(\mathbb{E}\|g_k\|_F^2 - \sigma^2\right)$$

$\Rightarrow$ Convergence rate

$$\boxed{\mathbb{E}\left[\frac{1}{K}\sum_{k=0}^{K-1}\|g_k\|_F^2\right] \leq \frac{2L\mathcal{L}(W_0)}{K} + \sigma^2}$$

$\rightarrow$ gradient converges to a noise ball of radius σ^2. Learning rate decay would overcome this.

Model #2: cubic regularisation

<u>Lipschitz smoothness model:</u>

$$\mathcal{L}(W + \Delta W) \le \overbrace{\boxed{\mathcal{L}(W) + g^T \Delta W}}^{\text{First order Taylor expansion}} + \overbrace{\boxed{\frac{\lambda}{2} \times \|\Delta W\|_F^2}}^{\text{quadratic penalty}}$$

<u>Cubic regularisation model:</u>

$$\mathcal{L}(W + \Delta W) \le \overbrace{\boxed{\mathcal{L}(W) + g^T \Delta W + \Delta W^T H \Delta W}}^{\text{Second order Taylor expansion}} + \overbrace{\boxed{\frac{\lambda}{6} \|\Delta W\|_F^3}}^{\text{cubic penalty}}$$

PRO: it's a better model of twice differentiable functions

CON: it requires knowing $H_{ij} = \dfrac{\partial^2 \mathcal{L}}{\partial W_i \partial W_j}$: one number for every pair of weights. Modern neural networks have billions of weights!

Model #3: mirror descent

first order Taylor

the thing we need to add to make this an identity.

$$\mathcal{L}(W+\Delta W) = \boxed{\mathcal{L}(W) + g^T \Delta W} + \boxed{\mathcal{L}(W+\Delta W) - \boxed{\mathcal{L}(W) + g^T \Delta W}}$$

Since we don't know the thing in the second box, mirror descent suggests modelling it as:

$$\boxed{h(W+\Delta W) - \left(h(W) + \nabla_W h(W)^T \Delta W\right)}$$

where h is a convex function that we are free to choose.

PRO: convex functions are easy to deal with

PRO: we get to choose h to model our loss function $\mathcal{L}$

CON: there may be no good convex model of $\mathcal{L}$

We will see on HW4 why this is called "mirror" descent.

Gradient descent is a special case

of mirror descent

Take $\boxed{h(W) = \|W\|_F^2}$

Then $\boxed{h(W+\Delta W) - \left(h(W) + \nabla_W h(W)^T \Delta W\right)}$ mirror descent penalty

$= \|W+\Delta W\|_F^2 - \|W\|_F^2 - 2\, W^T \Delta W$

$= \boxed{\|\Delta W\|_F^2}$ Lipschitz smoothness penalty

$\Longrightarrow$ the mirror descent penalty with $\boxed{h(W) = \|W\|_F^2}$ is equivalent to Lipschitz smoothness, which yields gradient descent.

What's missing?

- techniques that involve computing a Hessian don't scale to modern NNs.

- there is something "non-convex" about NNs, but mirror descent models the loss (locally) as convex

- the techniques are <u>generic</u> — they do not exploit knowledge of the neural network architecture.

Agenda for today

1. State of deep learning
2. Optimisation theory
3. Perturbation theory approach

attempt to develop an optimisation theory that makes explicit use of the neural network structure.

NN perturbations bounds

To zeroth order, a neural net is a product of scalars

$$f(x; \underline{a}) = \left(\prod_i a_i\right) x$$

$$\frac{\partial f}{\partial a_j} = \left(\prod_{i \neq j} a_i\right) x$$

both the network output and its gradient take the form of products

We showed in lecture 7 that expressions of this form obey the perturbation result

$$\boxed{\frac{f(x; \underline{a} + \Delta\underline{a}) - f(x; \underline{a})}{f(x; \underline{a})} = \prod_i \left(1 + \frac{\Delta a_i}{a_i}\right) - 1}$$

Trust in a Taylor expansion

We are interested in the region of validity of

$$\boxed{\mathcal{L}(W+\Delta W) \approx \mathcal{L}(W) + g^T \Delta W}$$

That is, how large can we make ΔW before the gradient $\nabla_W \mathcal{L}(W+\Delta W)$ incurs substantial relative change?

For neural nets, our perturbation results suggest the following model:

product over layers

$$\boxed{\frac{\|\nabla_W \mathcal{L}(W+\Delta W) - \nabla_W \mathcal{L}(W)\|_F}{\|\nabla_W \mathcal{L}(W)\|_F} \leq \prod_{l=1}^{L} \left(1 + \frac{\|\Delta W_l\|_F}{\|W_l\|_F}\right) - 1}$$

In words: the relative change in gradient is bounded by the product of the relative change in weights at each layer.

Small relative updates

This suggests an optimisation algorithm that does

$$\min_{\Delta W} \boxed{g^T \Delta W}$$

such that $\boxed{\prod_{l=1}^{L}\left(1+\frac{\|\Delta W_l\|_F}{\|W\|_F}\right)-1}$ is small.

That is, make updates that are aligned with the negative gradient, but that are small in relative terms per layer.

Per synapse relative updates

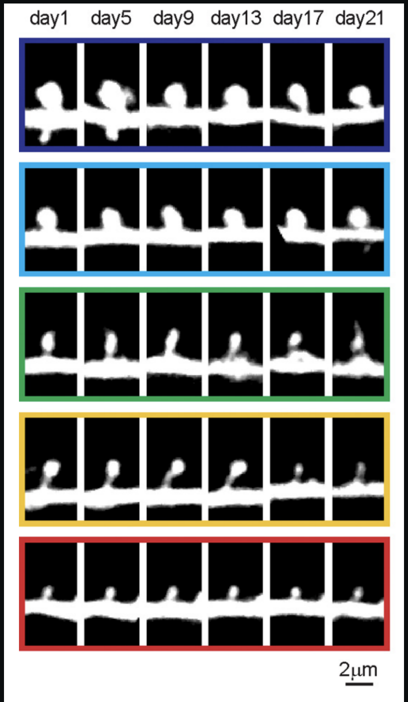
If we do small relative updates per synapse instead of per layer we get a multiplicative weight update.

Each iteration, a synapse either:

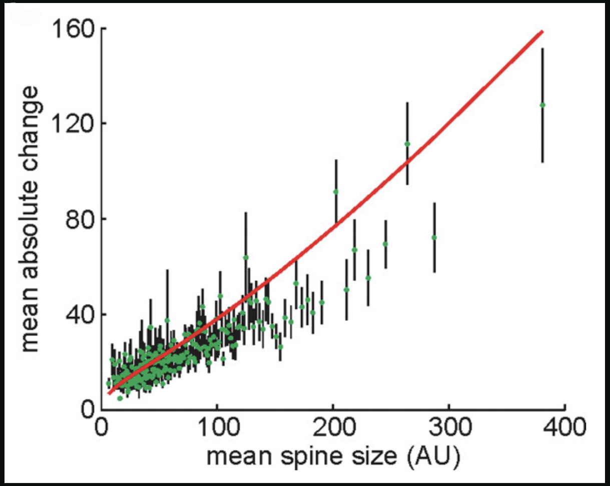
- grows by a factor $1+\varepsilon$
- shrinks by a factor $1-\varepsilon$

Multiplicative Dynamics Underlie the Emergence of the Log-Normal Distribution of Spine Sizes in the Neocortex *In Vivo*

Yonatan Loewenstein,[1] Annerose Kuras,[2†] and Simon Rumpel[2]
[1]Department of Neurobiology, Edmond and Lily Safra Center for Brain Sciences, Interdisciplinary Center for Neural Computation and Center for the Study of Rationality, Hebrew University, Jerusalem 91904, Israel, and [2]IMP—Research Institute of Molecular Pathology, Vienna 1030, Austria

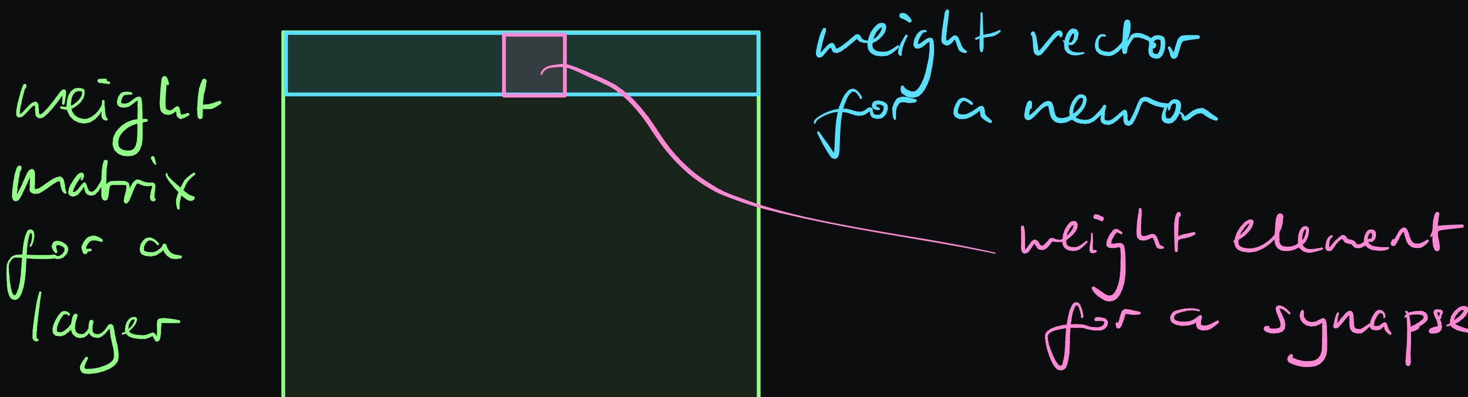


This has been observed in neuroscience!



Per neuron relative updates

We tested per layer, per neuron and per synapse relative updates.



We found that small relative updates per neuron worked best.

Architectural constraints

Recall that in lecture 7 we introduced the following constraints on a neuron's weights:

(1) $$\boxed{\sum_{i=1}^{d} w_i = 0}$$ — balanced excitation & inhibition

(2) $$\boxed{\sum_{i=1}^{d} w_i^2 = 1}$$ — hyperspherical constraint

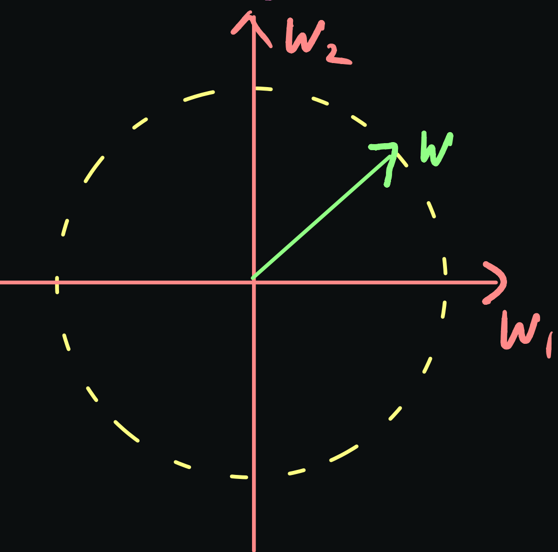
Formally, the optimisation domain is $\boxed{\Omega = \mathbb{S}^{d-2} \times \mathbb{S}^{d-2} \times \ldots \times \mathbb{S}^{d-2}}$

— one hypersphere per neuron in the network.

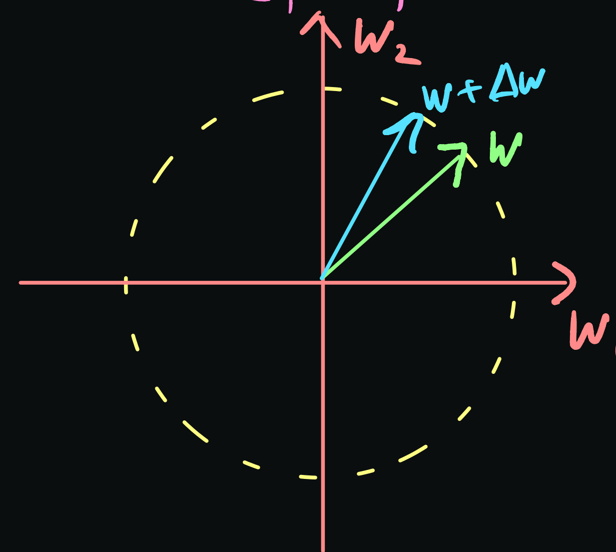
What does small relative change look like under these constraints?

Nero: neuronal rotator

The hyperspherical constraint says that for each neuron, the weight vector is constrained to the unit hypersphere:



so small relative change is just rotation by a small angle.



We tested this algorithm (with a couple of extra tricks) and found that it worked across many deep learning problems with the same "learning rate" of

$\frac{1}{2}$ a degree of rotation per neuron per iteration

Summary

- optimisation theory requires a model of how the first order Taylor expansion breaks down
- popular models (like mirror descent) are not well suited to neural nets
- by directly studying the perturbation properties of neural nets, we can design optimisers that require less tuning.

Next lecture

We'll start GENERALISATION THEORY

by looking at VC theory.