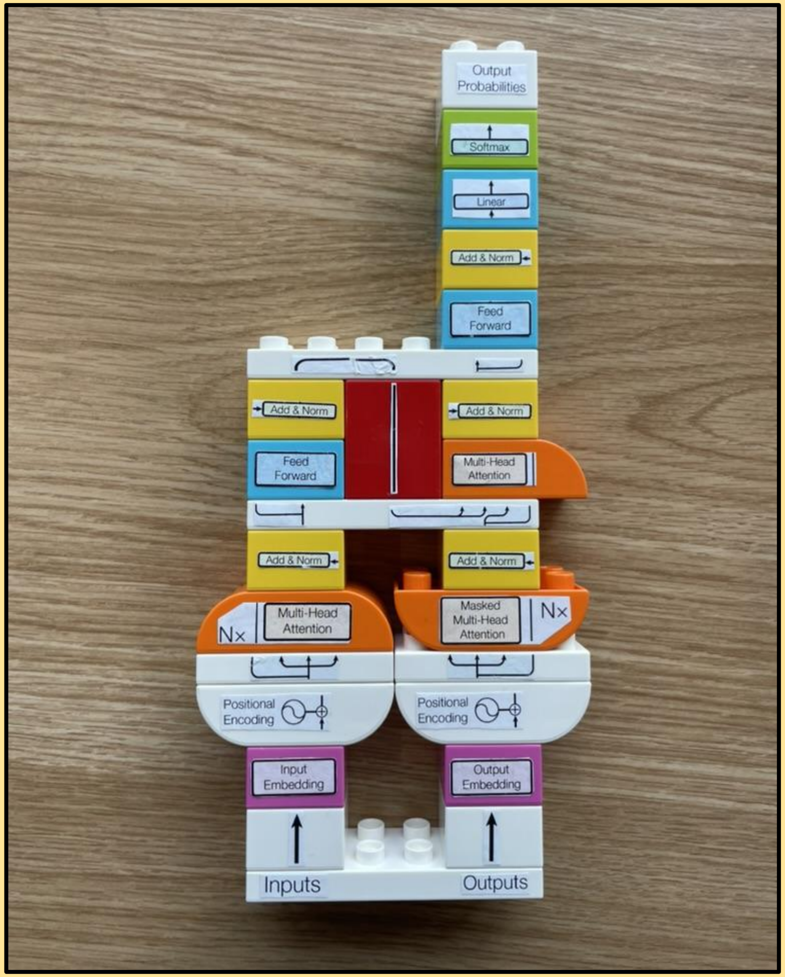
Metrized deep learning

Jeremy Bernstein

https://jeremybernste.in/

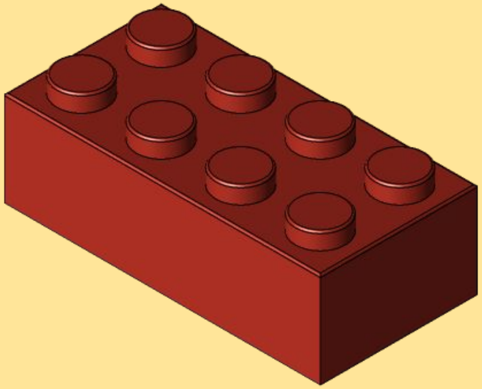
We build neural networks like lego



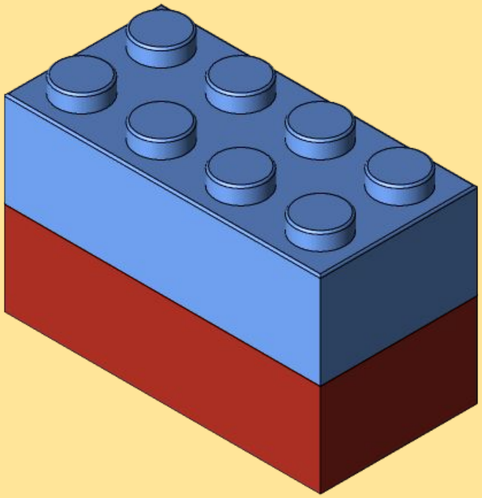
Then why don’t we also build **the theory** like lego?

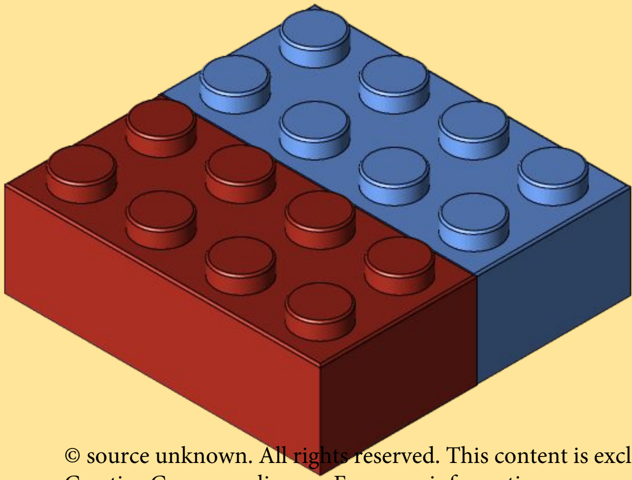
What does that even mean?

Suppose we can characterize the properties of an **individual layer**



Can we characterize the properties of **combinations of layers**?



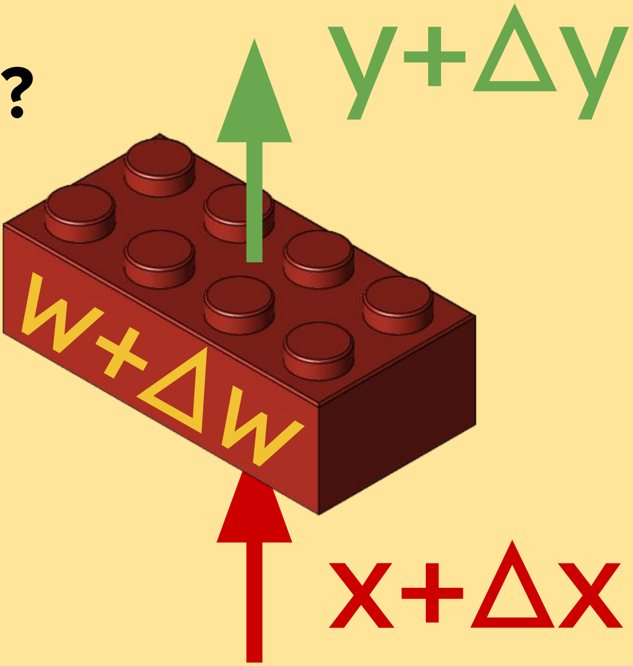


series combination

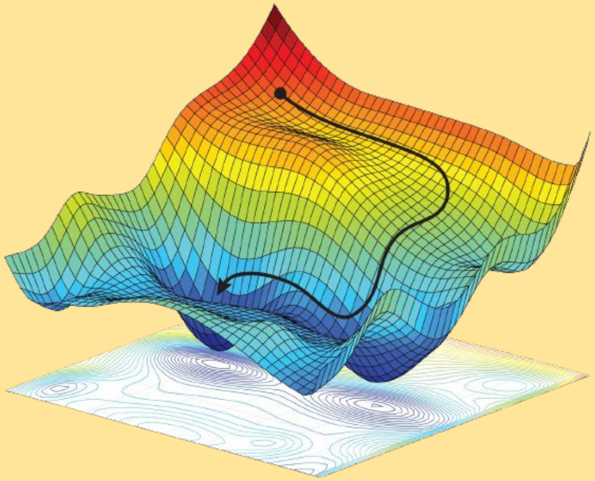
parallel combination

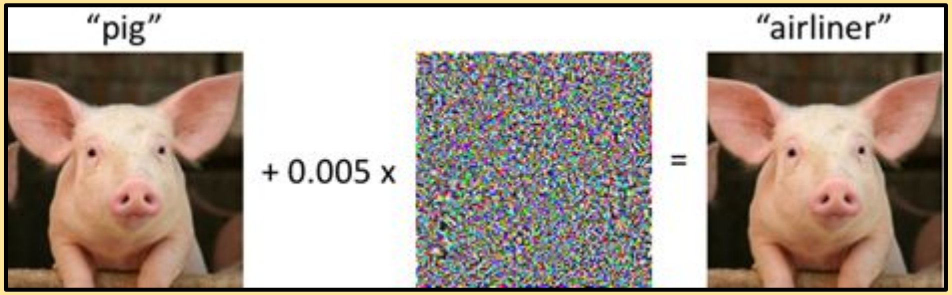
What properties do we care about?

a layer has
- an input **x**
- weights **w**
- an output **y**

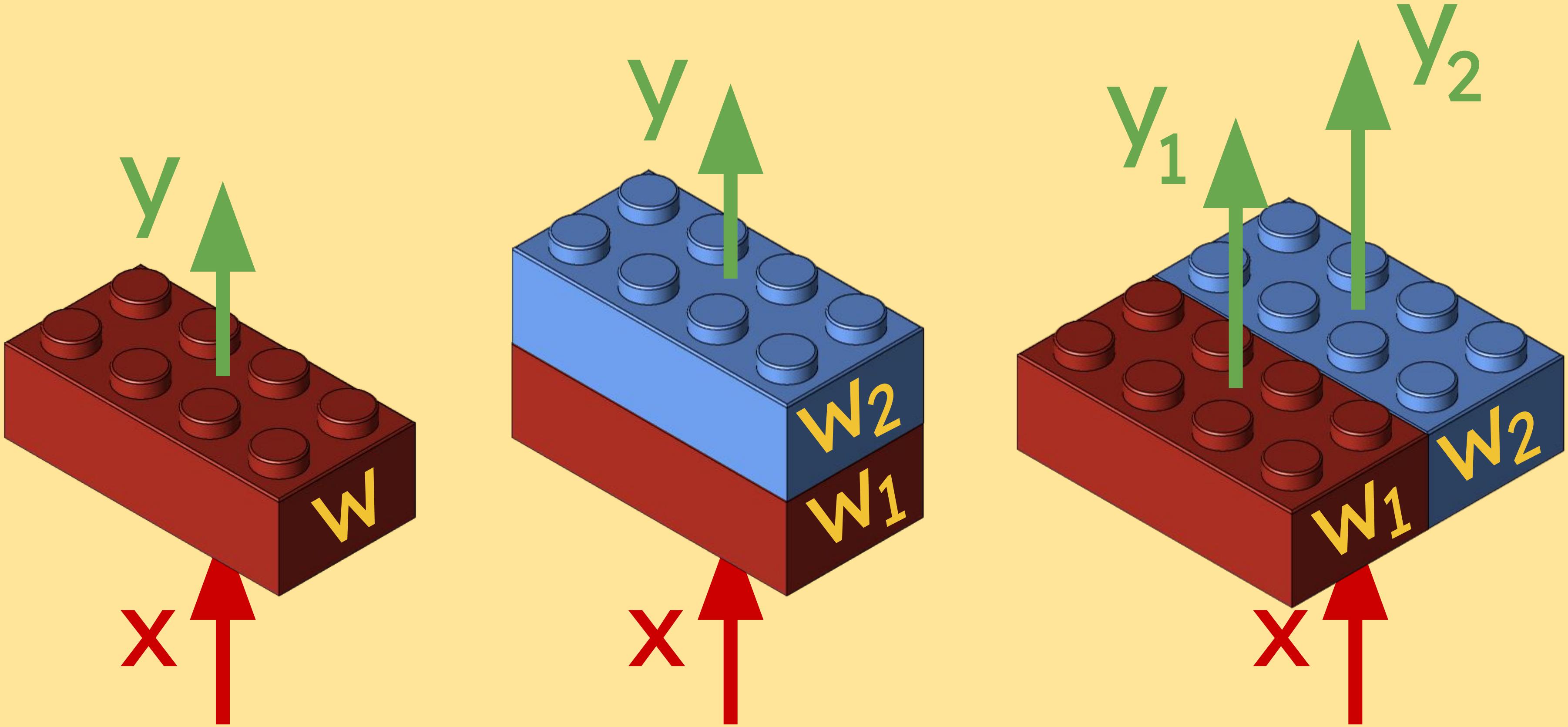


How **sensitive** is the output to the weights and inputs?





If we understand the sensitivity of individual layers...



...can we extend this understanding to **combinations**?

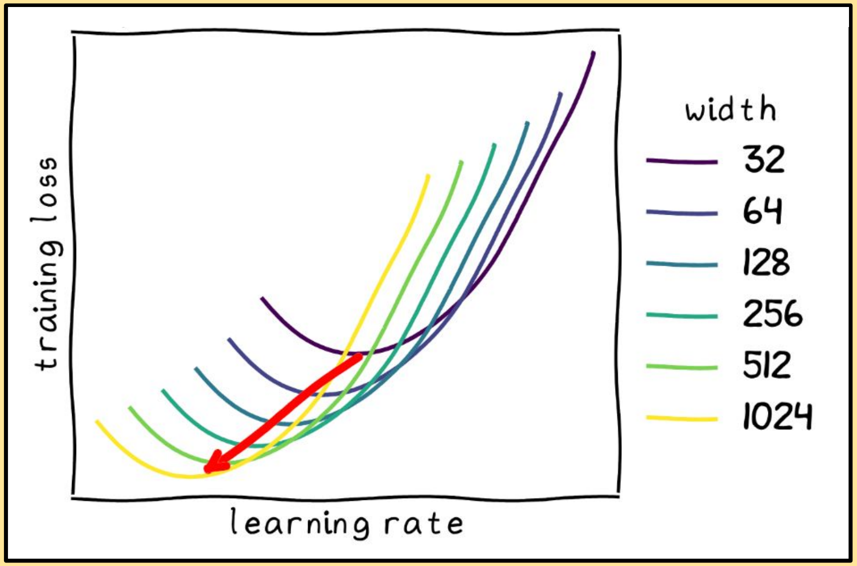
A deep learning library should be like a lego set

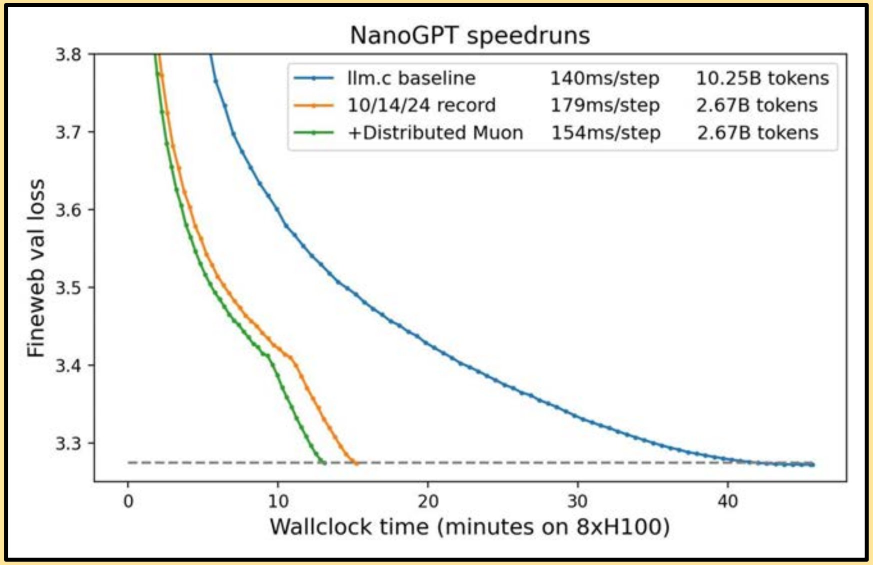
- a collection of layers each with its own theory
- a system of rules for combining layers
- build whatever you want!

The practical payoff... so far

fixing scaling issues

nanoGPT speed records





@kellerjordan0

Part I

Optimization theory

Recall: Steepest descent

Consider a loss function $\mathcal{L}: \mathrm{R}^N \to \mathrm{R}$ and its Taylor expansion:

$$\begin{aligned}\boxed{\mathcal{L}(w+\Delta w)} &= \boxed{\mathcal{L}(w)} + \boxed{\nabla_w \mathcal{L}^T \Delta w} + \boxed{\tfrac{1}{2} \Delta w^T \nabla^2_w \mathcal{L} \Delta w} + \boxed{\dots} \\ &\leq \boxed{\mathcal{L}(w)} + \boxed{\nabla_w \mathcal{L}^T \Delta w} + \boxed{\tfrac{1}{2} \lambda \, \|\Delta w\|^2} + \boxed{\dots}\end{aligned}$$

can we find a **norm $\| \cdot \|$** and a **sharpness λ**
to make this inequality hold tightly?

If so, then we can select an optimization step by solving:

$$\arg\min_{\Delta w} \quad \boxed{\nabla_w \mathcal{L}^T \Delta w} + \boxed{\tfrac{1}{2} \lambda \, \|\Delta w\|^2}$$

How could we produce such a norm? Step 1/3

We need to bound $\Delta w^T \nabla^2_w \mathcal{L} \Delta w \leq \lambda \, ||\Delta w||^2$

Recall that in deep learning, the loss function is a composite

$$\mathcal{L}(w) = \underset{\substack{\uparrow \\ \text{error measure}}}{\ell} \circ \underset{\substack{\uparrow \\ \text{neural net}}}{f(w)}$$

By the Gauss–Newton decomposition, the Hessian satisfies:

$$\Delta w^T \nabla^2_w \mathcal{L} \Delta w = \Delta w \; \nabla^2_w f \; \Delta w \; \nabla_f \ell + \Delta w \; \nabla_w f \; \nabla^2_f \ell \; \nabla_w f \; \Delta w$$

How could we produce such a norm? Step 2/3

We need to bound $\Delta w^T \nabla^2_w \mathcal{L} \Delta w \leq \lambda \, \|\Delta w\|^2$

Now suppose we know a good norm $\| \cdot \|$ on the network output

Then we may bound the Gauss-Newton decomposition:

$$\Delta w^T \nabla^2_w \mathcal{L} \Delta w = \Delta w \; \nabla^2_w f \; \Delta w \;\; \nabla_f \ell + \Delta w \; \nabla_w f \; \nabla^2_f \ell \; \nabla_w f \; \Delta w$$

$$\leq \|\Delta w \; \nabla^2_w f \; \Delta w\| \; \|\nabla_f \ell\| + \|\nabla^2_f \ell\| \; \|\nabla_w f \; \Delta w\|^2$$

dual norm

operator norm

How could we produce such a norm? Step 3/3

We need to bound $\Delta w^T \nabla^2_w \mathcal{L} \Delta w \leq \lambda \, \|\Delta w\|^2$

By the Gauss-Newton decomposition and having an output norm:

$$\begin{aligned}\Delta w^T \nabla^2_w \mathcal{L} \Delta w &= \Delta w \;\nabla^2_w f \;\Delta w \;\; \nabla_f \ell + \Delta w \;\nabla_w f \;\nabla^2_f \ell \;\nabla_w f \;\Delta w \\ &\leq \|\Delta w \;\nabla^2_w f \;\Delta w\| \; \|\nabla_f \ell\| + \|\nabla^2_f \ell\| \; \|\nabla_w f \;\Delta w\|^2\end{aligned}$$

Therefore, our problem reduces to the following:

Can we produce a norm $\| \cdot \|$ on the network weights such that:

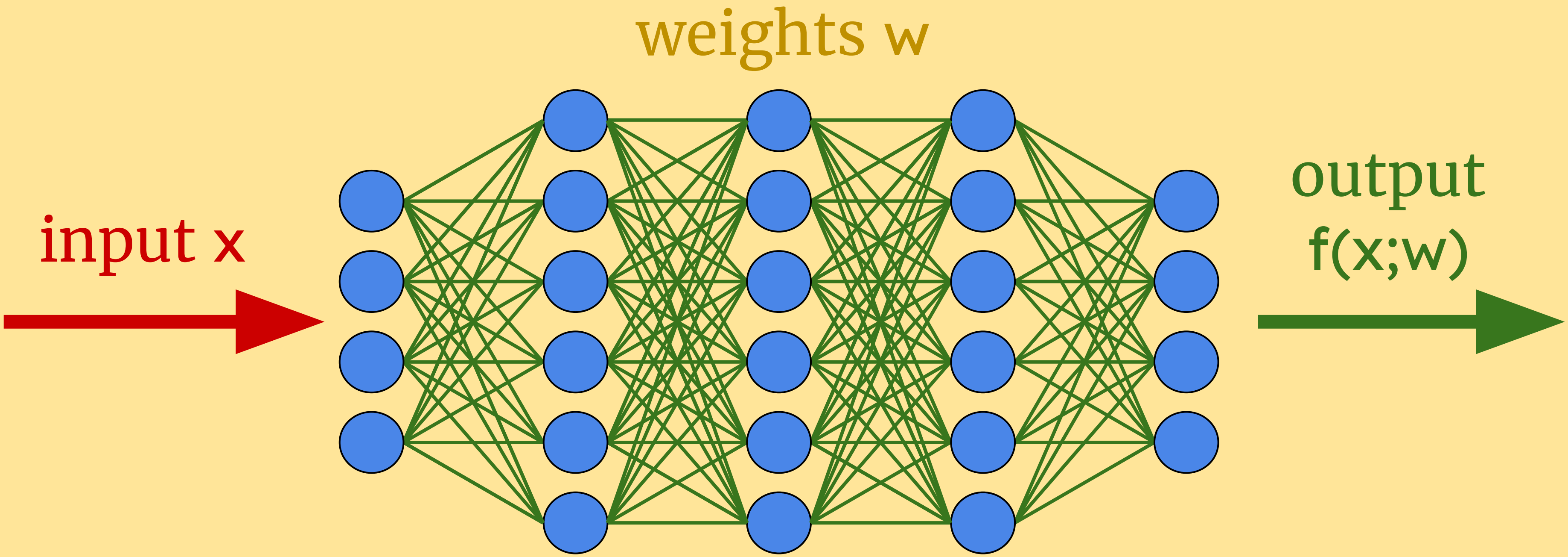
$$\|\Delta w \;\nabla^2_w f \;\Delta w\| \leq \alpha \, \|\Delta w\|^2 \qquad \|\nabla_w f \;\Delta w\|^2 \leq \delta \, \|\Delta w\|^2$$

network is “Lipschitz smooth” network is “Lipschitz”

Interpreting these conditions

$\boxed{\|\Delta w \; \nabla^2_w f \; \Delta w\|} \leq \boxed{\alpha \, \|\Delta w\|^2}$ $\qquad$ $\boxed{\|\nabla_w f \; \Delta w\|^2} \leq \boxed{\delta \, \|\Delta w\|^2}$

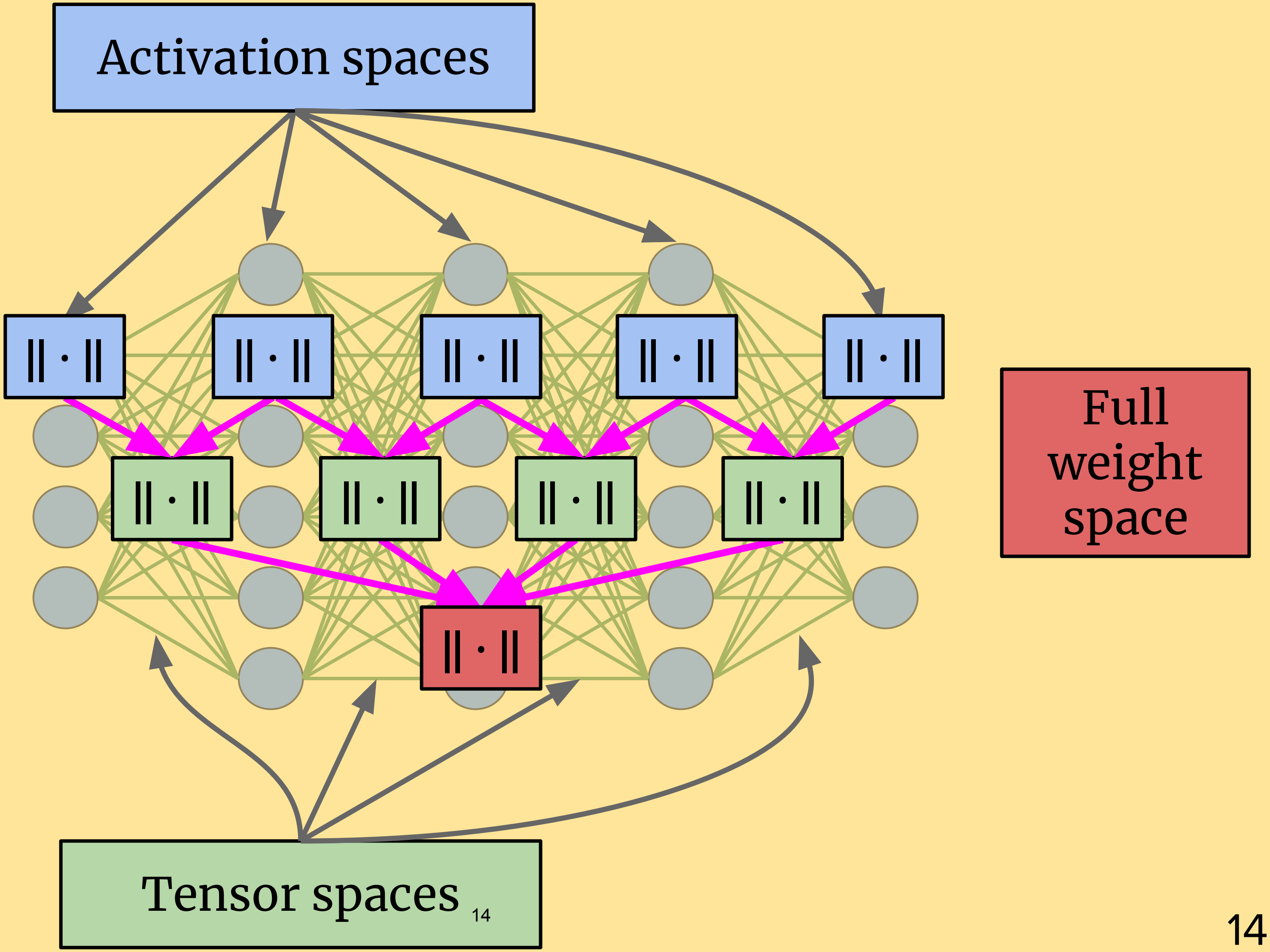
network is “Lipschitz smooth” $\qquad$ network is “Lipschitz”



We seek a weight norm that controls the network’s Taylor expansion

$$f(x;\, w+\Delta w) \;=\; f(x;\, w) \;+\; \boxed{\nabla_w f \; \Delta w} + \boxed{\Delta w \;\, \nabla^2_w f \; \Delta w} \;+\; \ldots$$

Recursively inducing a norm on the weight space



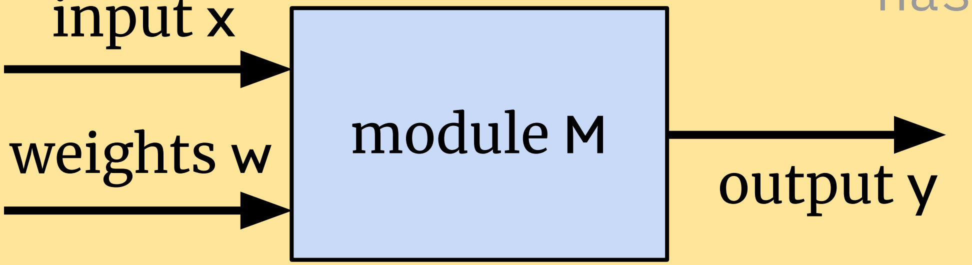
Part II

The theory of modules

Combinator pattern

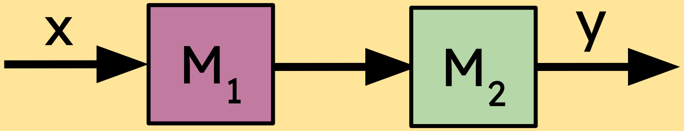
> complex structures are built by defining a small set of very simple “primitives”, and a set of “combinators” for combining them into more complicated structures

Haskell Wiki



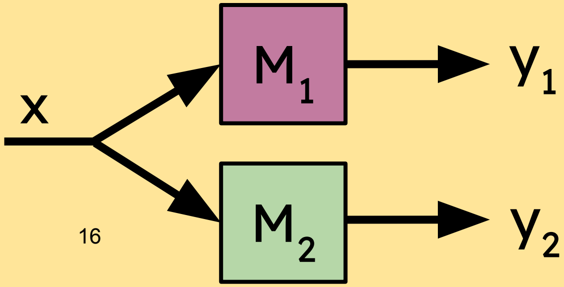
Given two modules M_1 and M_2 we can form their:

composition $M_2 \circ M_1$



modules in series

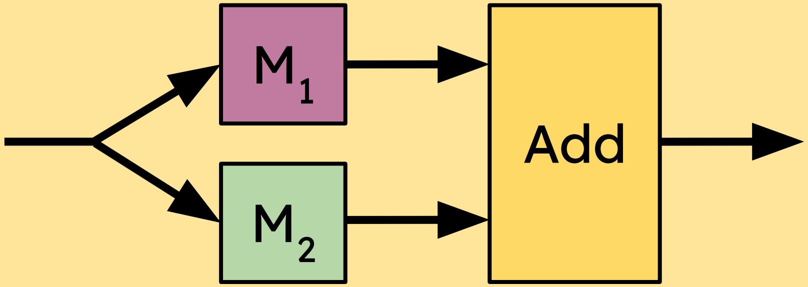
concatenation (M_1, M_2)



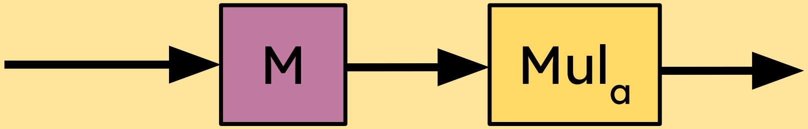
modules in parallel

Some basic circuits

addition $M_1 + M_2$

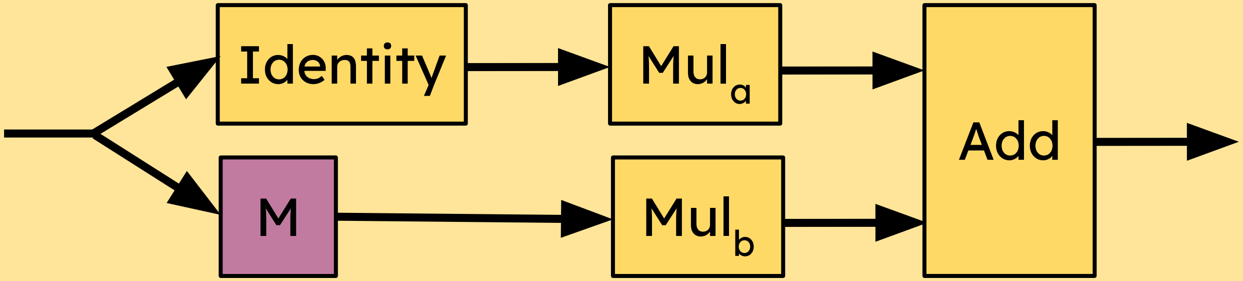


multiplication by scalar $a * M$



where Add and Mul_a are special “utility modules”

Now we can build a **residual block** $a * \mathrm{Identity} + b * M$



Three kinds of modules

Atoms — hand-declared attributes

e.g. Linear Conv2d Embed

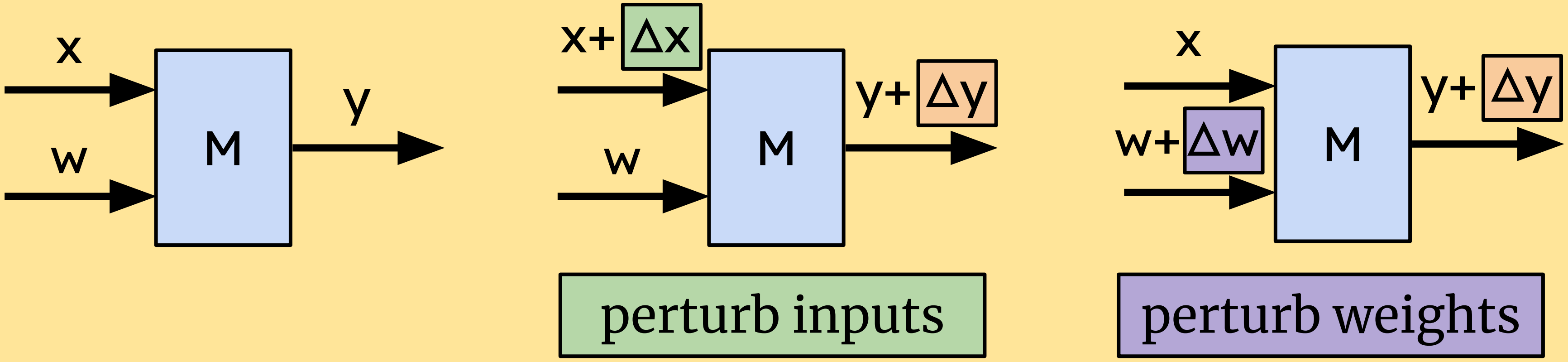
Bonds — hand-declared attributes + no weights

e.g. ReLU FunctionalAttention

Compounds — combinations of atoms and bonds

e.g. MLP = Linear ∘ ReLU ∘ Linear

Sensitivity of a module



perturb inputs

perturb weights

Our major goal:

1. predict size of Δy from size of Δx
2. predict size of Δy from size of Δw

} for any module

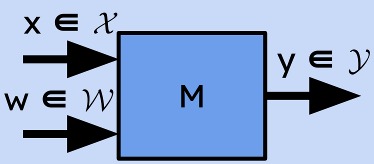
If we can do this for atoms and bonds, what about compounds?

Formal definition of a module

Definition: Module

A module M must have three vector spaces:

1)	input space	$\mathcal{X}$
2)	weight space	$\mathcal{W}$
3)	output space	$\mathcal{Y}$



and four attributes:

I.	a function	M.forward	$\mathcal{X} \times \mathcal{W} \to \mathcal{Y}$
II.	a number	M.sensitivity	$\in \mathbb{R}^+$
III.	a number	M.mass	$\in \mathbb{R}^+$
IV.	a norm	M.norm	$\mathcal{W} \to \mathbb{R}^+$

Definition: Well-normed module

A module M is well-normed if

1)	the input space	$\mathcal{X}$	has norm $\|\cdot\|_{\mathcal{X}}$
2)	the output space	$\mathcal{Y}$	has norm $\|\cdot\|_{\mathcal{Y}}$

and the first derivatives of the module satisfy:

I. $\| \nabla_w M \diamond \Delta w \|_{\mathcal{Y}} \leq$ M.norm(Δw)

II. $\| \nabla_x M \diamond \Delta x \|_{\mathcal{Y}} \leq$ M.sensitivity $* \; \| \Delta x \|_{\mathcal{X}}$

Some atomic modules

Definition: Linear module L

L.forward(W, x)	= W x
L.sensitivity	= 1
L.mass	= 1
L.norm	$= \|\cdot\|_{\text{spectral}} * \text{sqrt(fan-in/fan-out)}$

L <u>well-normed</u> if $\begin{cases} \|\cdot\|_{\mathcal{X}} = \|\cdot\|_{\mathcal{Y}} = \|\cdot\|_{\text{RMS}} \\ \|x\|_{\mathcal{X}} \leq 1 \text{ and L.norm(W)} \leq 1 \end{cases}$

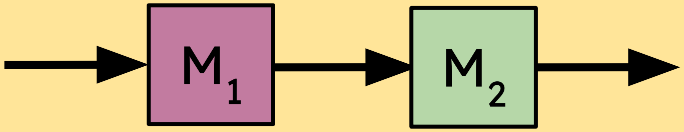
Definition: Embedding module E

E.forward(W, x)	= W x
E.sensitivity	= 1
E.mass	= 1
E.norm	$= \max_i \|\text{column}_i(W)\|_{\text{RMS}}$

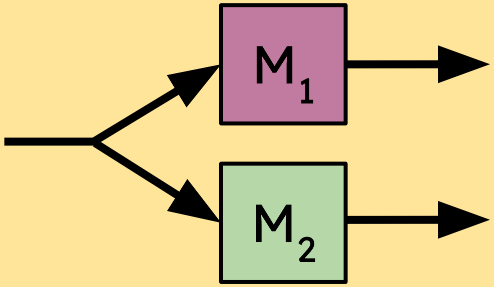
E <u>well-normed</u> if $\begin{cases} \|\cdot\|_{\mathcal{X}} = \|\cdot\|_1 \text{ and } \|\cdot\|_{\mathcal{Y}} = \|\cdot\|_{\text{RMS}} \\ \|x\|_{\mathcal{X}} \leq 1 \text{ and L.norm(W)} \leq 1 \end{cases}$

Can we make compound modules automatically “good”?

We want to be able to prove statements about module combinations



composition



concatenation

Proposition 1 Module combination is associative

Proposition 2 Module combination preserves well-normed-ness

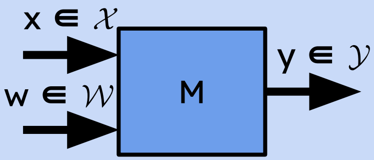
Proposition 3 Feature learning is apportioned by mass

Defining combination rules

Definition: Module

A module M must have

1)	input space	$\mathcal{X}$
2)	weight space	$\mathcal{W}$
3)	output space	$\mathcal{Y}$



and four attributes:

I.	a function	M.forward	$\mathcal{X} \times \mathcal{W} \to \mathcal{Y}$
II.	a number	M.sensitivity	$\in \mathbb{R}^+$
III.	a number	M.mass	$\in \mathbb{R}^+$
IV.	a norm	M.norm	$\mathcal{W} \to \mathbb{R}^+$

Definition: Well-normed module

A module M is well-normed if

1)	the input space	$\mathcal{X}$	has norm $\|\cdot\|_{\mathcal{X}}$
2)	the output space	$\mathcal{Y}$	has norm $\|\cdot\|_{\mathcal{Y}}$

and the first derivatives of the module satisfy:

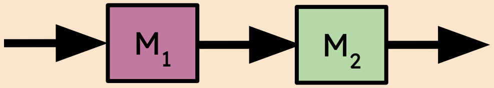
I. $\| \nabla_w M \diamond \Delta w \|_{\mathcal{Y}} \leq \text{M.norm}(\Delta w)$

II. $\| \nabla_x M \diamond \Delta x \|_{\mathcal{Y}} \leq \text{M.sensitivity} * \| \Delta x \|_{\mathcal{X}}$

Definition: Module composition

Given two modules M_1 and M_2 their composite

$M = M_2 \circ M_1$



is the module with attributes:

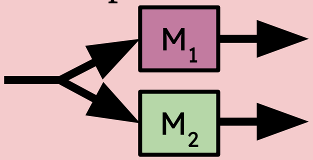
I.	M.forward	$= M_2\text{.forward} \circ M_1\text{.forward}$
II.	M.sensitivity	$= M_1\text{.sensitivity} * M_2\text{.sensitivity}$
III.	M.mass	$= M_1\text{.mass} + M_2\text{.mass}$
IV.	M.norm	$= \max(p * M_1\text{.norm}, q * M_2\text{.norm})$

where $p = \text{M.mass} / M_1\text{.mass} * M_2\text{.sensitivity}$
$q = \text{M.mass} / M_2\text{.mass}$

Definition: Module concatenation

Given two modules M_1 and M_2 their tuple

$M = (M_1, M_2)$



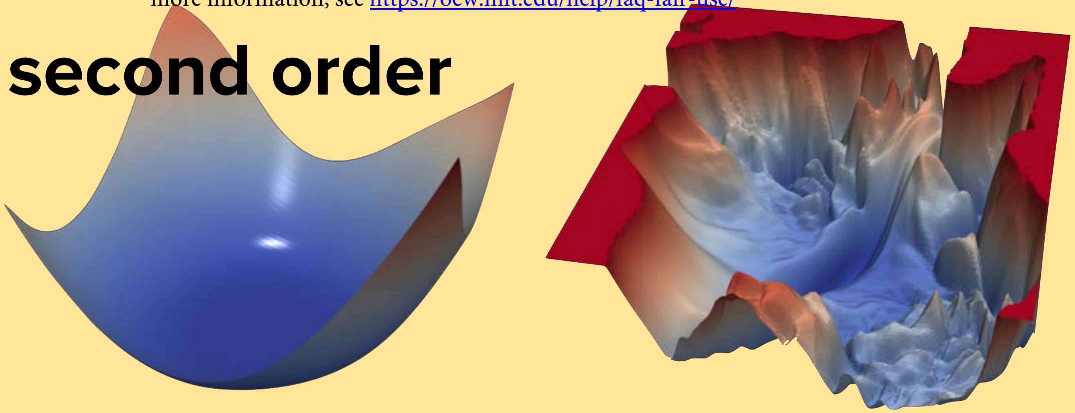
is the module with attributes:

I.	M.forward	$= (M_1\text{.forward}, M_2\text{.forward})$
II.	M.sensitivity	$= M_1\text{.sensitivity} + M_2\text{.sensitivity}$
III.	M.mass	$= M_1\text{.mass} + M_2\text{.mass}$
IV.	M.norm	$= \max(p * M_1\text{.norm}, q * M_2\text{.norm})$

where $p = \text{M.mass} / M_1\text{.mass}$
$q = \text{M.mass} / M_2\text{.mass}$

The theory works to second order

Think: "Generalized top eigenvalues"



Visualizing the loss landscape of neural nets, Li et al (2018)

Definition: Module sharpness

A module M is "(α, β, γ)-sharp" if second derivatives obey:

I. $\| \Delta w \, \nabla_w \nabla_w M \, \Delta\tilde{w} \|_{\mathcal{Y}} \leq \alpha * \text{M.norm}(\Delta w) * \text{M.norm}(\Delta\tilde{w})$

II. $\| \Delta x \, \nabla_x \nabla_w M \, \Delta w \|_{\mathcal{Y}} \leq \beta * \text{M.norm}(\Delta w) * \| \Delta x \|_{\mathscr{X}}$

III. $\| \Delta x \, \nabla_x \nabla_x M \, \Delta\tilde{x} \|_{\mathcal{Y}} \leq \gamma * \| \Delta x \|_{\mathscr{X}} * \| \Delta\tilde{x} \|_{\mathscr{X}}$

- Sharpness tuple (α, β, γ) obeys associative combination laws
- Neural net loss functions are:
  - Lipschitz smooth in the modular norm
  - with non-dimensional Lipschitz constants!
- So long as the error measure is smooth in the module output

Part III

Scaling

Scale is all you need?

302 neurons

130 thousand neurons

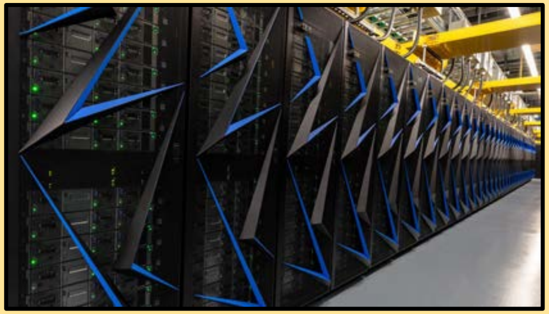
100 billion neurons

250 billion neurons

Fruit fly © Sanjay Acharya. Other images © source unkown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/

Recipe for AGI?

1. get the biggest supercomputer you can

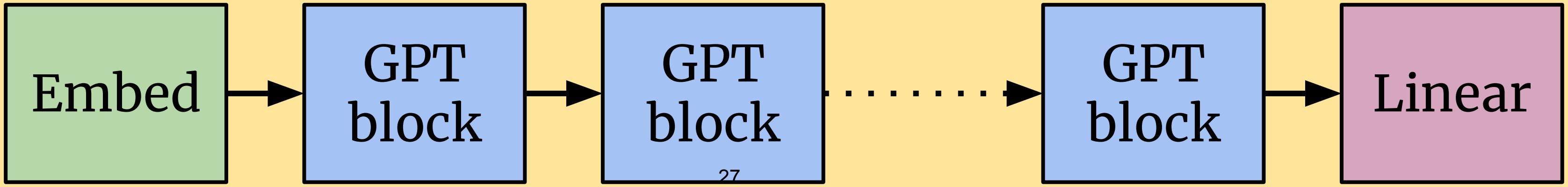


2. scrape as much data as you can (don't get caught!)

The New York Times

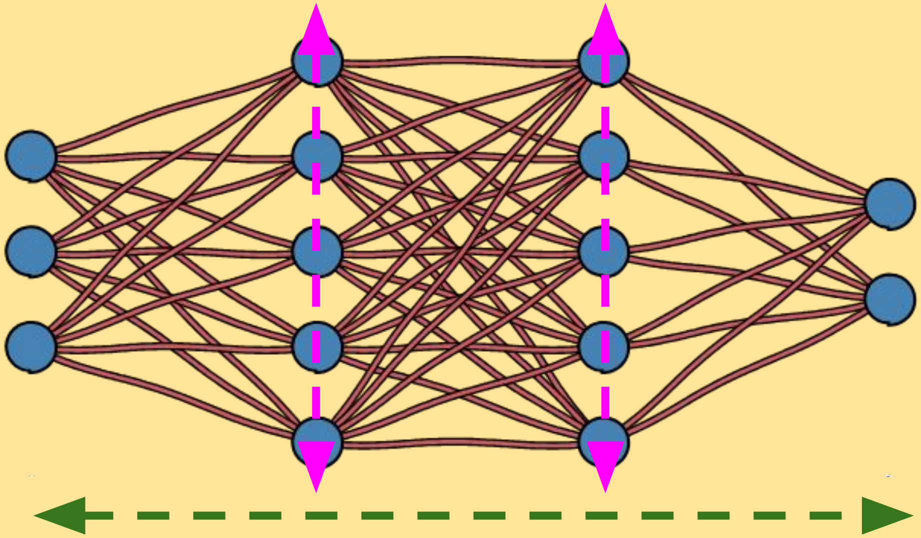
stackoverflow

3. train the biggest transformer you can

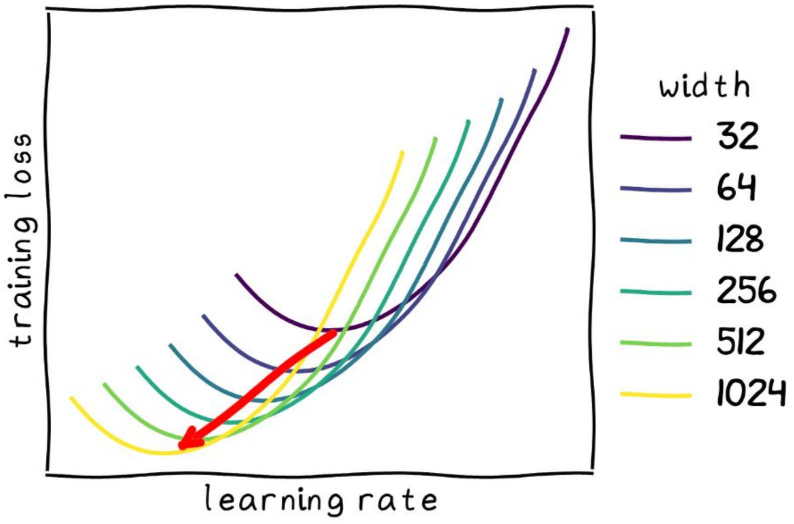


Embed → GPT block → GPT block .... → GPT block → Linear

Problem: Scaling can hurt

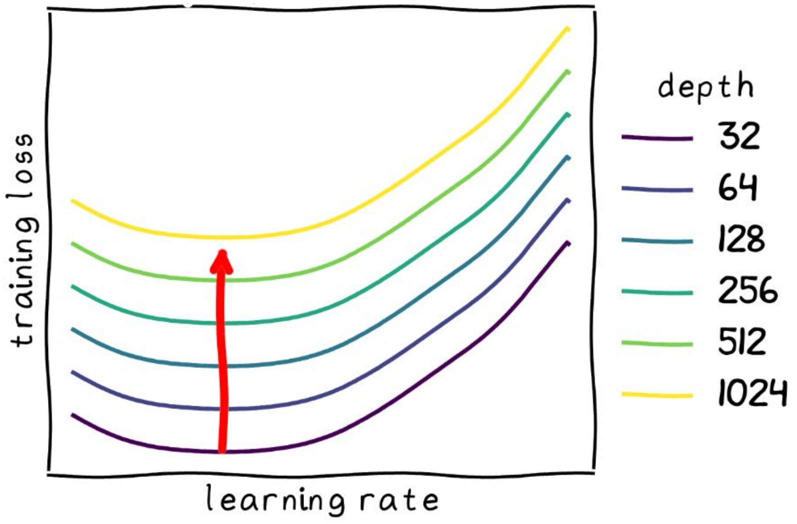


Scale width



❌ optimal learning rate drifts

Scale depth



❌ performance gets worse

Our thesis for good scaling

non-dimensional tight Lipschitzness ⇒ LR transfer

If for generic module M we can achieve:

1. Lipschitz constants independent of width, depth, etc.
2. Bounds stay tight across scale

Then controlling M.norm(Δw) ⇒ control over $\| \Delta y \|_{y}$

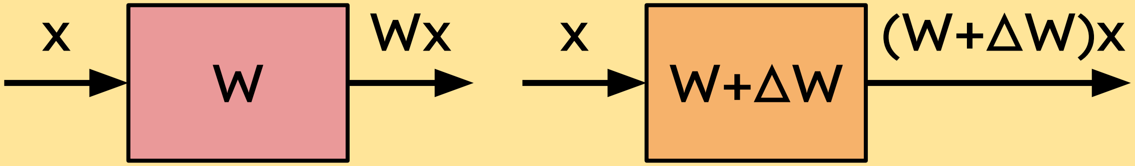
Formally, we want $\boxed{\| \Delta y \|_{y} \leq \text{M.norm}(\Delta w)}$ to hold tightly

Breaking up the problem

What are good properties for an individual layer?

How to keep under composition & concatenation?

Example: linear layer



impose spectral conditions

$$\begin{cases} \text{sqrt(fan-in/fan-out)} * \| W \|_{\text{spectral}} = 1 \\ \text{sqrt(fan-in/fan-out)} * \| \Delta W \|_{\text{spectral}} = \text{LR} \end{cases}$$

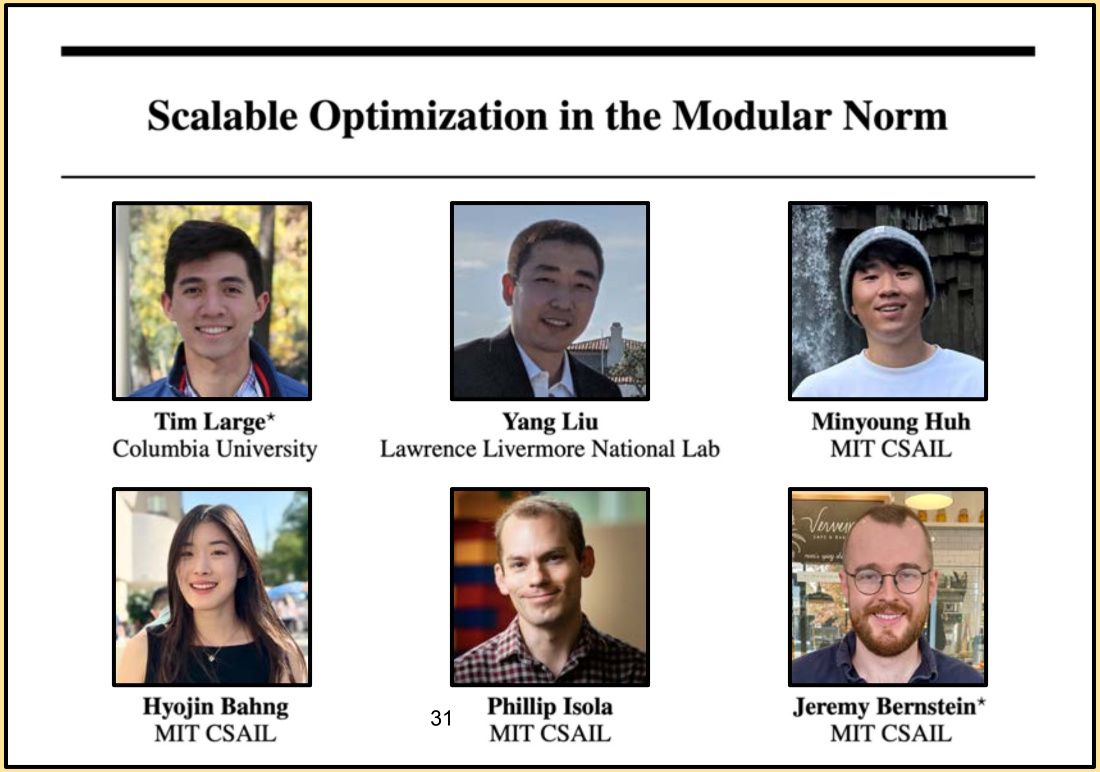
📘 **On the distance between two neural networks and the stability of learning**
Bernstein, Vahdat, Yue, Liu NeurIPS 2020

📗 **A spectral condition for feature learning**
Yang*, Simon*, Bernstein* arXiv 2023

Breaking up the problem

What are good properties for an individual layer?

How to keep under composition & concatenation?



Scalable Optimization in the Modular Norm

Tim Large*
Columbia University

Yang Liu
Lawrence Livermore National Lab

Minyoung Huh
MIT CSAIL

Hyojin Bahng
MIT CSAIL

Phillip Isola
MIT CSAIL

Jeremy Bernstein*
MIT CSAIL

mo(du,la)

```python
import torch

from modula.atom import Linear
from modula.bond import ReLU

mlp = Linear(10,10000) @ ReLU() @ Linear(10000, 1000)

weights = mlp.initialize(device="cpu")
data, target = torch.randn(1000), torch.randn(10)

for step in range(steps:=20):
    output = mlp.forward(data, weights)
    loss = (target - output).square().mean()
    loss.backward()

    with torch.no_grad():
        grad = weights.grad()
        mlp.normalize(grad)
        weights -= 0.1 * grad
        weights.zero_grad()
```

Compatible with any array programming package

mo(du,la)

PyTorch

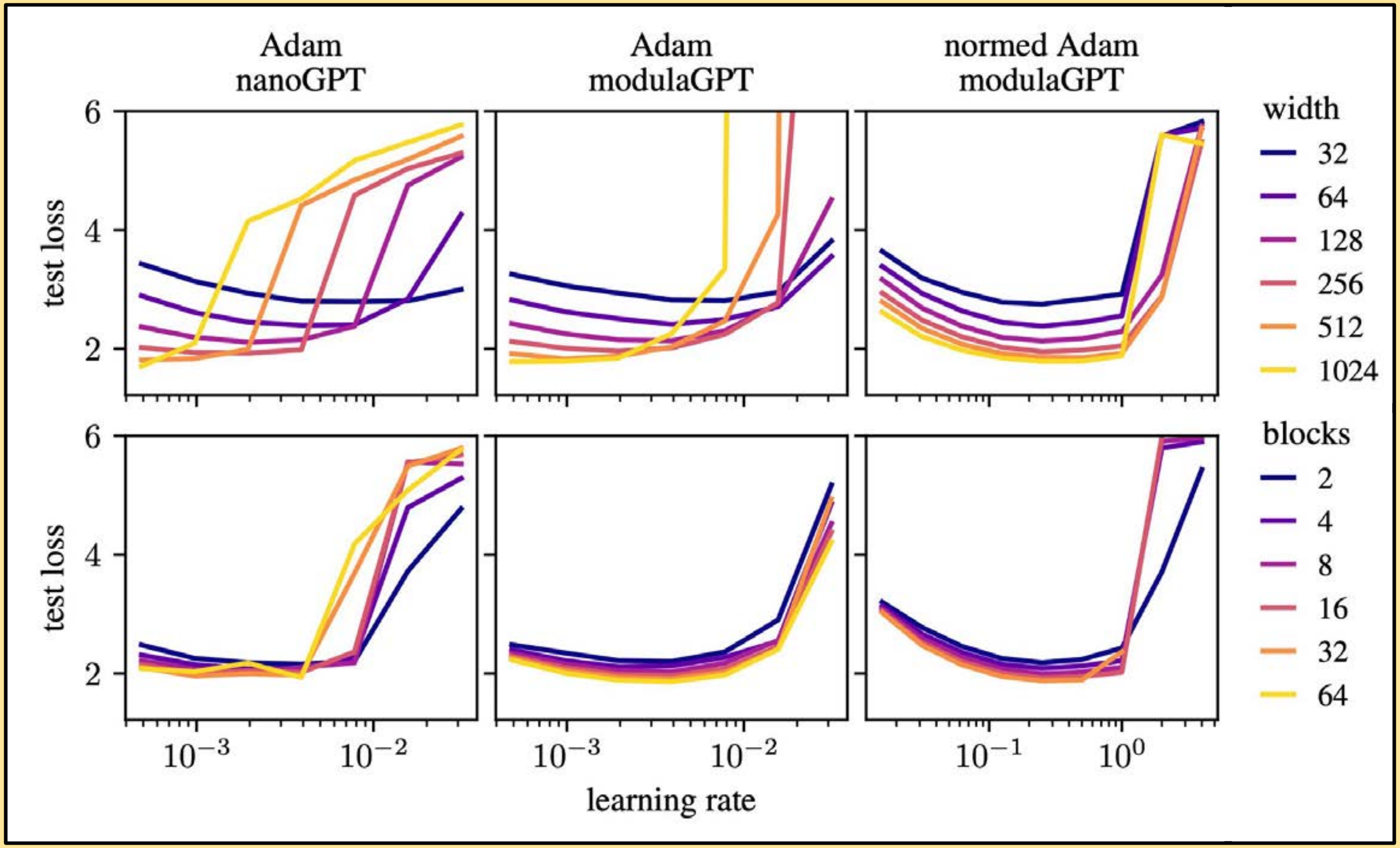
JAX

NumPy

PyTorch:	github.com/jxbz/modula/
JAX:	github.com/GallagherCommaJack/modulax/
NumPy:	open in Colab — best place to start

Learning rate transfers across width and depth

- train GPT for 10k steps on OpenWebText
- normalization {on, off} with Adam as base optimizer



In the paper:

- ❖ enables training GPT using SGD
- ❖ transfers LR across context length

Part IV

Modular duality

Recall: Steepest descent

Consider a loss function $\mathcal{L}: R^N \to R$ that satisfies:

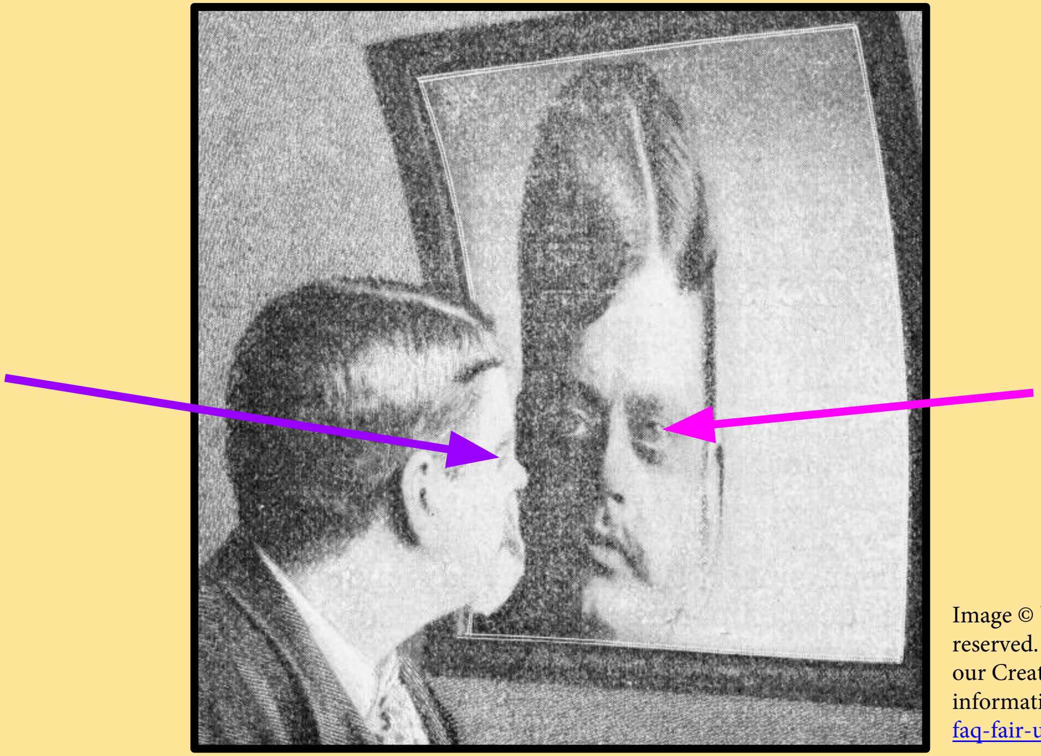
$$\boxed{\mathcal{L}(w+\Delta w)} \leq \boxed{\mathcal{L}(w)} + \boxed{\nabla_w \mathcal{L}^T \Delta w} + \boxed{\tfrac{1}{2} \lambda \, \|\Delta w\|^2}$$

We can select an optimization step by solving:

$$\arg\min_{\Delta w} \boxed{\nabla_w \mathcal{L}^T \Delta w} + \boxed{\tfrac{1}{2} \lambda \, \|\Delta w\|^2}$$

$$= \; - \underbrace{\boxed{\| \nabla_w \mathcal{L} \| \, / \, \lambda}}_{\text{step size}} \; * \; \underbrace{\arg\max_{\|\Delta w\|=1}}_{\text{“duality map”}} \boxed{\nabla_w \mathcal{L}^T \Delta w}$$

Gradient descent does not type check



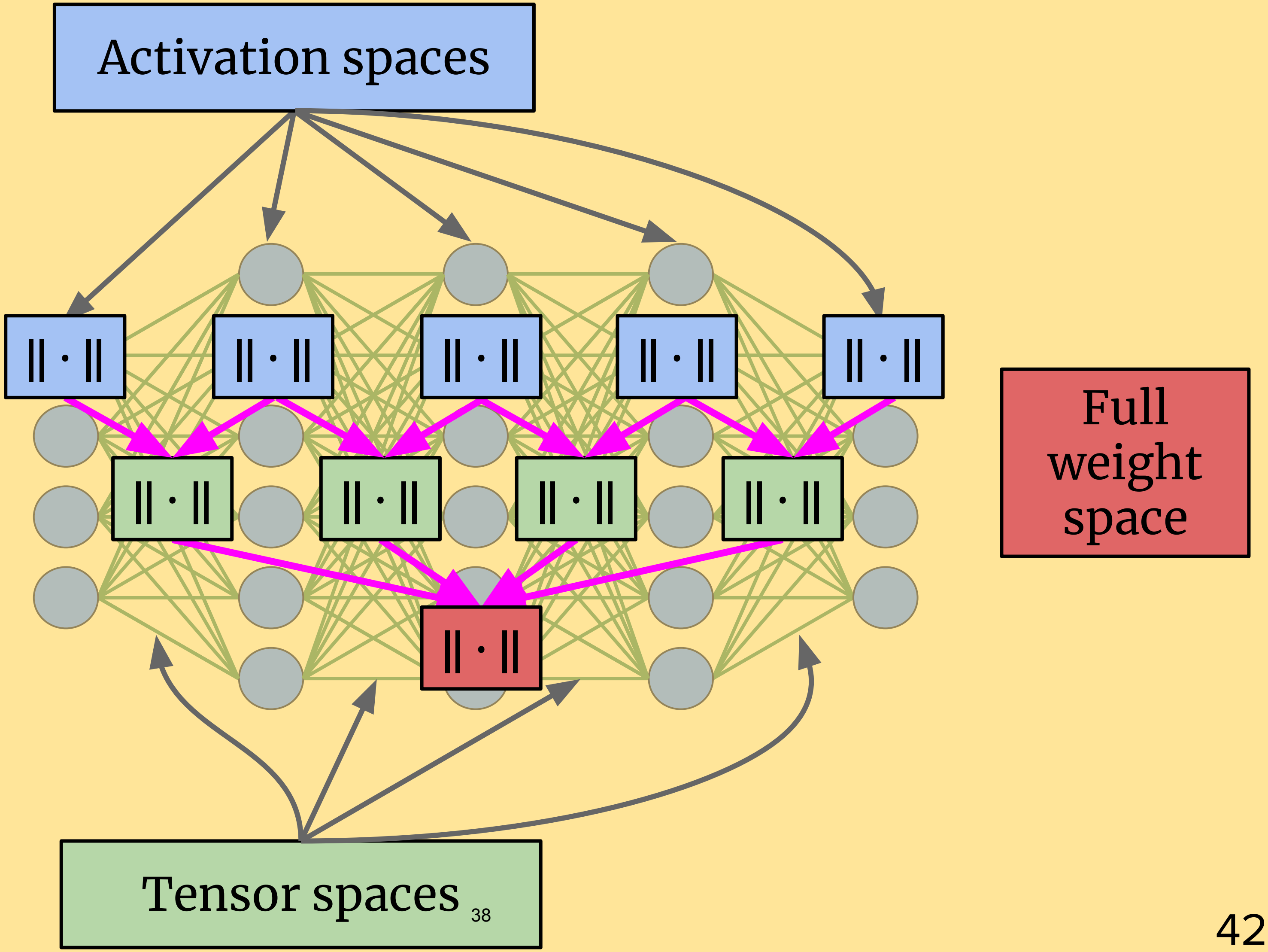
weight space

gradient space

Image © "ilbusca" at iStock. All rights reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/

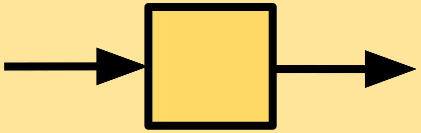
```
weight - LR * gradient              ✗
weight - LR * dualize(gradient)     ✓
```

Recall: Inducing a norm on the full weight space



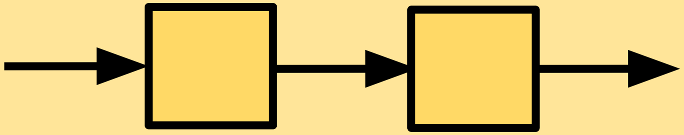
We propose modular dualization

1. solve duality map for each layer

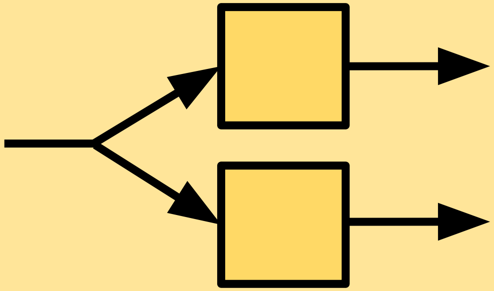


$$\mathtt{dualize}(G) = \arg\max_{\|A\|=1} \langle A, G \rangle$$

2. recursively solve duality map for full network

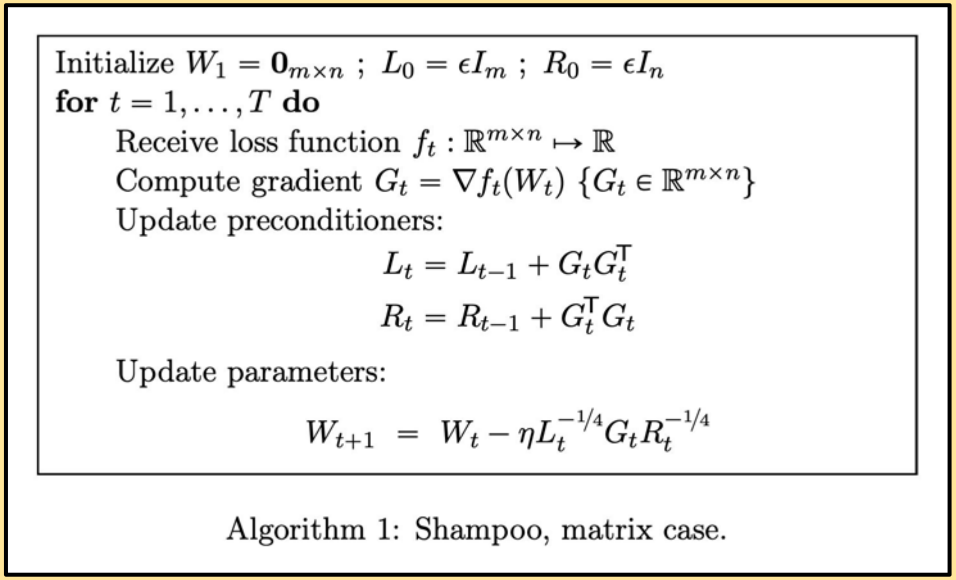


modules in series



modules in parallel

Faster training with Shampoo



```
Initialize $W_1 = \mathbf{0}_{m\times n}$ ; $L_0 = \epsilon I_m$ ; $R_0 = \epsilon I_n$
for $t = 1, \ldots, T$ do
    Receive loss function $f_t : \mathbb{R}^{m\times n} \mapsto \mathbb{R}$
    Compute gradient $G_t = \nabla f_t(W_t)$ $\{G_t \in \mathbb{R}^{m\times n}\}$
    Update preconditioners:
        $L_t = L_{t-1} + G_t G_t^{\mathsf{T}}$
        $R_t = R_{t-1} + G_t^{\mathsf{T}} G_t$
    Update parameters:
        $W_{t+1} = W_t - \eta L_t^{-1/4} G_t R_t^{-1/4}$
```

Algorithm 1: Shampoo, matrix case.

Core primitive: ΔW = $-\eta$ * $(GG^{\mathsf{T}})^{-1/4}\ G\ (G^{\mathsf{T}}G)^{-1/4}$

= $-\eta$ * $\operatorname{argmax}_{\|A\|\leq 1} \langle G, A\rangle$

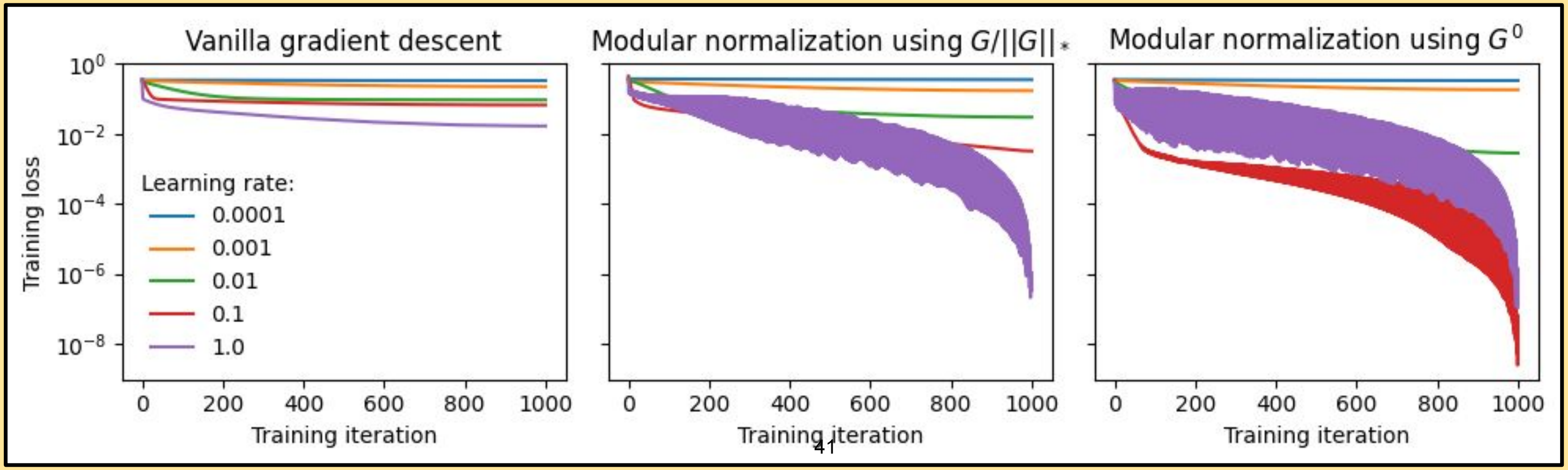
i.e. "steepest descent under the spectral norm"

Implement in Modula just by overriding Linear

$\boxed{(GG^T)^{-¼}\ G\ (G^TG)^{-¼}} = \boxed{G^0}$ i.e. set all singular values to one

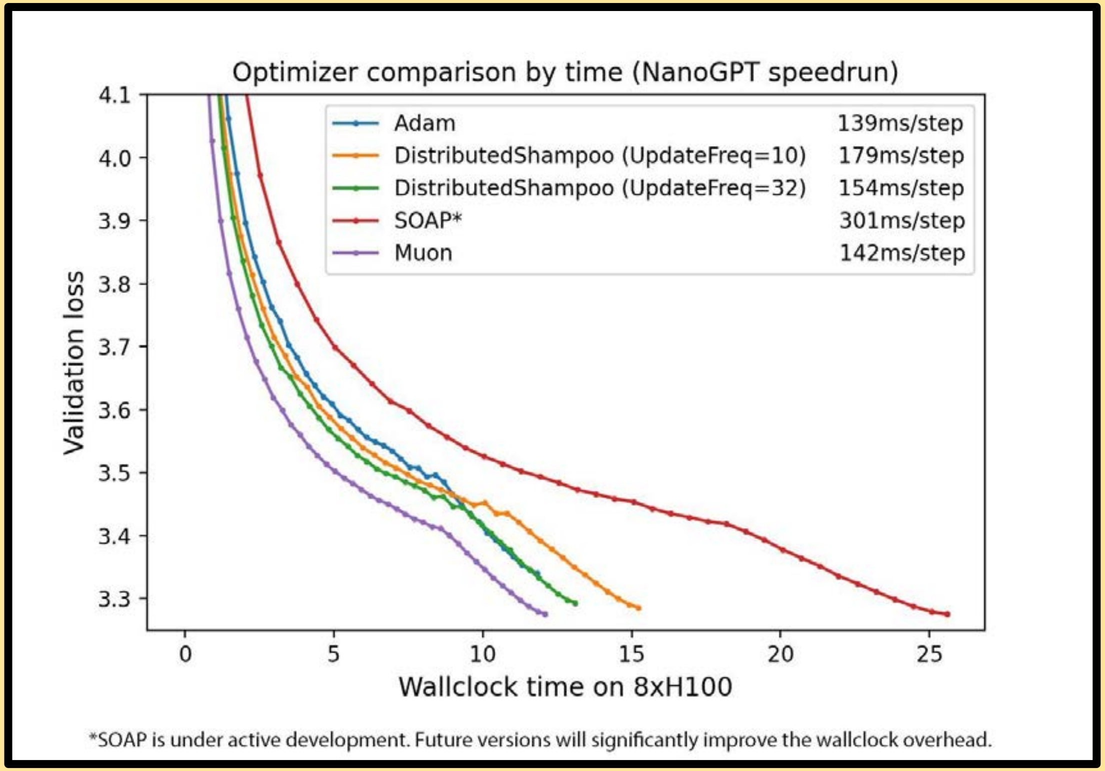
```
class ShampooLinear(Linear):
    def __init__(self, fanout, fanin):
        super().__init__(fanout, fanin)

    def normalize(self, grad_w, target_norm=1.0):
        grad_weight = grad_w[0]
        U, S, Vt = np.linalg.svd(grad_weight, full_matrices=False)
        return [U @ Vt * target_norm]
```

open in Colab



NEWS FLASH

¡New NanoGPT speed record!



*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

@kellerjordan0

Uses “Newton-Schulz” to do **`Linear.dualize`** fast

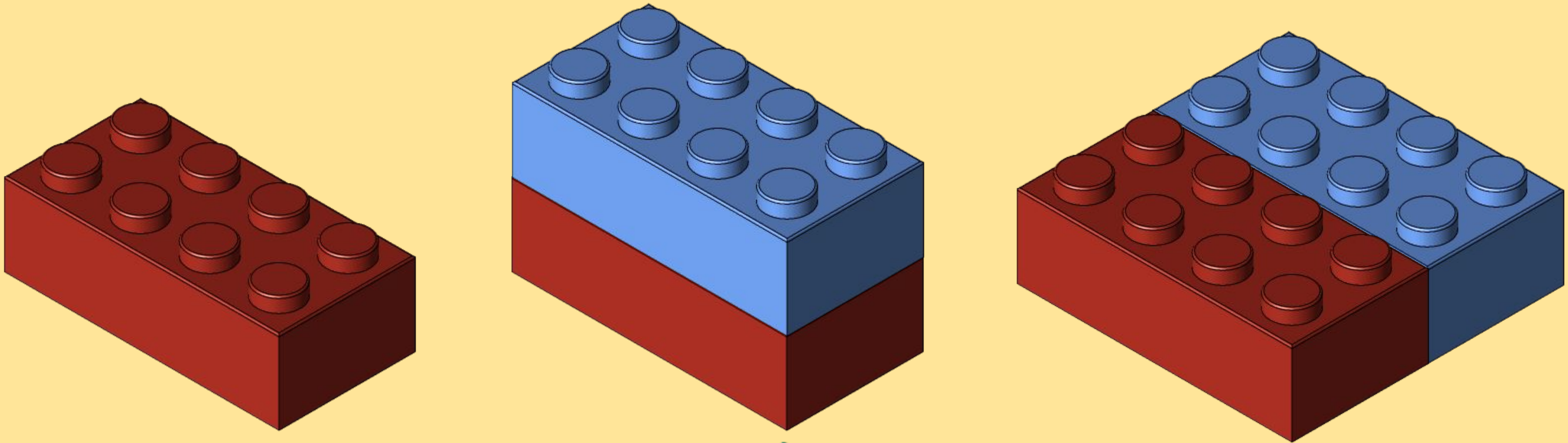
$$X_{t+1} = a\, X_t - b\, X_t X_t^T X_t + c\, X_t X_t^T X_t X_t^T X_t$$

mo(du,la)

```
import torch

from modula.atom import Linear
from modula.bond import ReLU

mlp = Linear(10,10000) @ ReLU() @ Linear(10000, 1000)

weights = mlp.initialize(device="cpu")
data, target = torch.randn(1000), torch.randn(10)

for step in range(steps:=20):
    output = mlp.forward(data, weights)
    loss = (target - output).square().mean()
    loss.backward()

    with torch.no_grad():
        grad = weights.grad()
        ~~mlp.normalize(grad)~~ mlp.dualize(grad)
        weights -= 0.1 * grad
        weights.zero_grad()
```

Conclusion

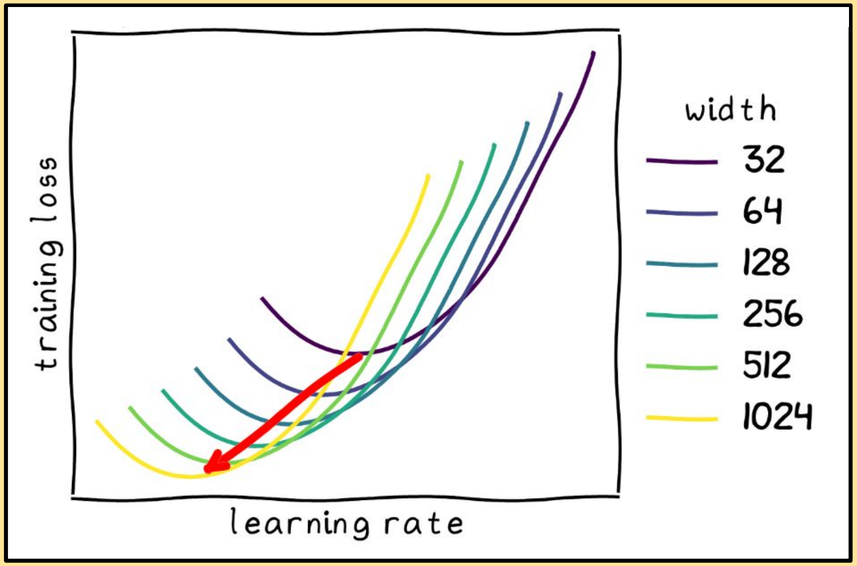
A deep learning library should be like a lego set

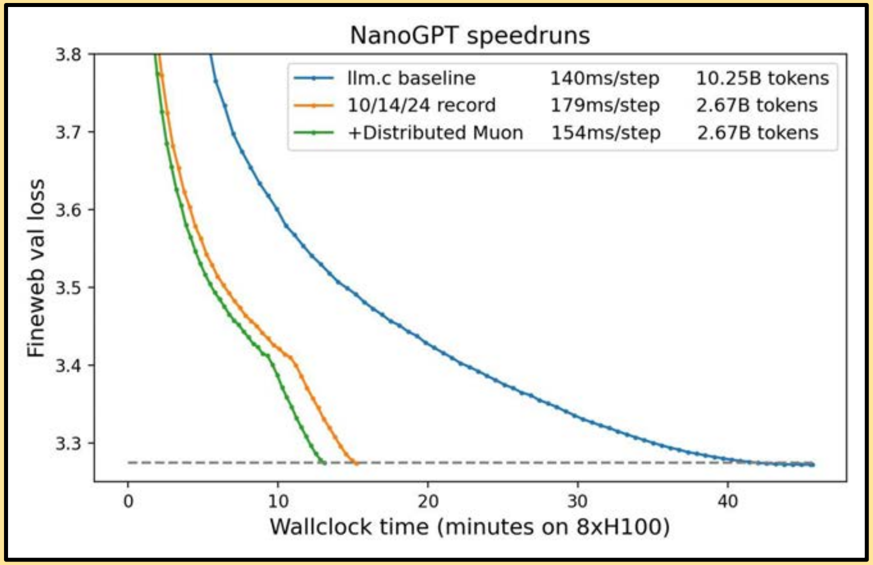


The practical payoff… so far

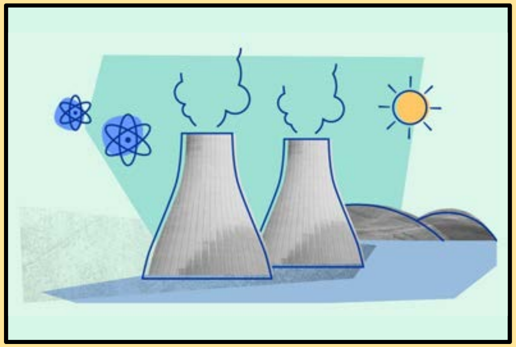
fixing scaling issues

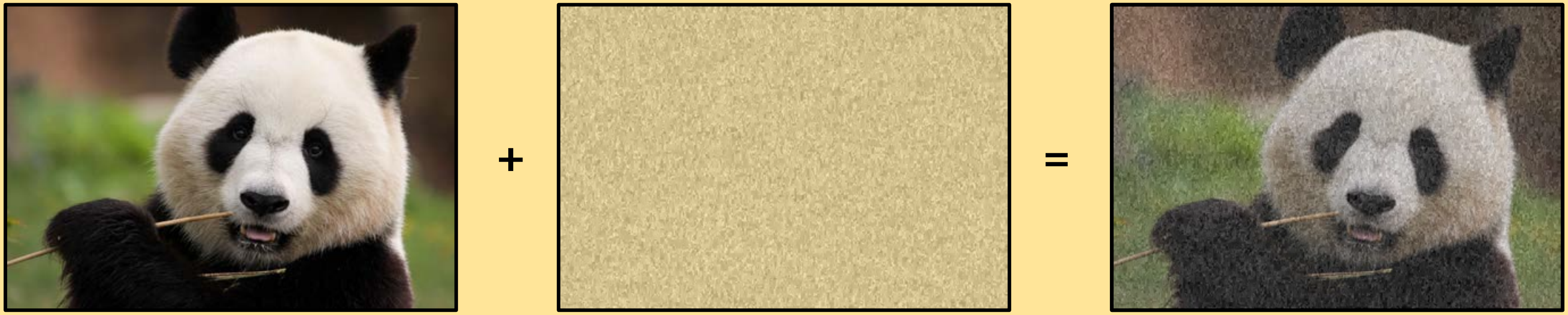
nanoGPT speed records



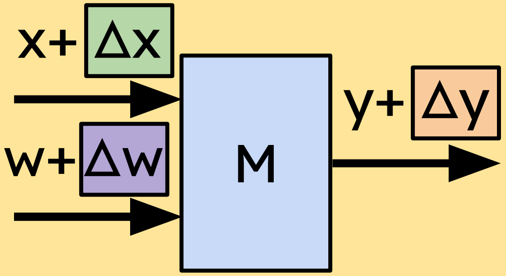


The future: Robust, low-precision models





We believe are questions
of module sensitivity



mo(du,la)

https://modula.systems/