[NESTED LEARNING]

* CORE IDEA:

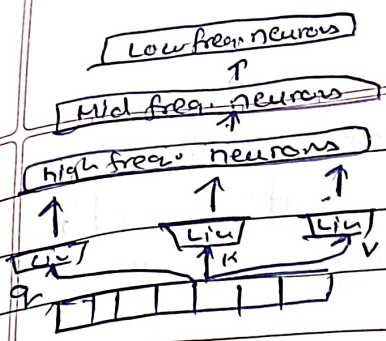
- In deep learning, there is multiple layers of successive memory compression, i.e depth ≠ deep computation
- In NL → multiple optimisers nested inside each other. Every layer, optimizer, memory unit has it's own optimization problem.
  ∴ It is a stack of optimizers & not functions

(1) Deep optimizers
↳ Adam, SGD + Momentum ⇒ memory system that learns to compress gradients usefully

∴ Expressive optimizers with deep memory are designed which has multiple layers of memory inside optimizer itself.
↳ Deep momentum
↳ Better preconditioning
↳ Better gradient compression.

(2) Self modifying model
↳ sequence model that learns to modify itself by learning to update it's own algo.

(3) Continum memory system (CMS)
↳ diff parts of memory update at different freq. like long / mid / short term memory
→ each compresses at own time scale



each level update at different frequency & doesn't rely on centralised clock.

* ASSOCIATIVE MEMORY: ability to form & retrieve connection

2.1 IMP!

~~Learning = neural update~~

Memory = neural update via I/P

Learning = acquiring effective & useful memory.

Every neural net layer & every optimizer is an associative memory system

↳ It stores associations b/w things it has seen.

Let associative memory M as a function

$$M : K \longrightarrow V$$

K: key, V: value

$$\therefore M^* = \arg\min_M \tilde{L}(M(K); V)$$

$\tilde{L}$ → objective that measures quality of mapping

Eg 1: Training 1 Layer MLP = Associative memory

→ model is $y = Wx$

& gradient descent is $W_{t+1} = W_t - \eta \nabla_W L(W_t; x_{t+1})$

↳ standard backprop

now,

$$\min_W \left(\langle Wx_{t+1}, u_{t+1} \rangle + \frac{1}{2\eta} |W - W_t|^2 \right)$$

$\langle Wx, u \rangle$ → dot prod. of what model produces (Wx) & what it should.

$$\cdot\ u_{t+1} = \nabla_y L(W_t; x_{t+1})$$

now, u_{t+1} = error signal aka local surprise signal

If $u_{t+1} \uparrow\uparrow \Rightarrow$ bad pred.

$u_{t+1} \downarrow\downarrow \Rightarrow$ good pred.

$\frac{1}{2}\|W - W_t\|^2$ ↳ stabilizer. like gradient descent.

∴ It is clear from Eg 1 that neural net is storing compressed map b/w each data point x & it's error / surprise signal u. ie training = memory compression

Eg 2: Momentum = 2-level memory

$$m_{t+1} = m_t - \eta \nabla_W L(W_t; x_{t+1})$$

$$W_{t+1} = W_t + m_{t+1}$$

Here m_t is a second associative memory that stores & compresses past gradients.

$$\therefore m_{t+1} = \arg\min_W \left(-\langle m, \nabla L(W_t; x_{t+1}) \rangle + \eta \|m - m_t\|^2 \right)$$

Eg 3: Linear attention = Memory

$$M_{t+1} = M_t + v_t k_t^T$$

aka $$M_{t+1} = M_t - \nabla \tilde{L}(M_t; k_t, v_t)$$

M_t is associative memory mapping key → values by gradient descent

∴ W_K, W_Q, W_V → traind by GD

& M → updated by inner GD, nested optimization

SUMMARY of 2.1

	Component	Key	Value	Memory
①	MLP traing	x	∇L	longterm
②	Momentum	grad index	grad	medium
③	Linear attention	k	v	short
④	Optimizer	weights	grad stats	meta

* NESTED OPTIMIZATION PROBLEMS

2.2

- Update freq → for any compon A:

  f_A = no. of updates of component A per unit time

In transformers:

1. Attention is fast memory
2. Adam momentum is medium memory
3. Weights are slow memory.

~~If A faster than B then A > B~~

↳ If $f_A > f_B$ or $f_A = f_B$ but A must be computed faster

or A ≡ B when no one is faster & are independent to each other

Hierarchy

LVL 1 → fastest → optimation prob. → outer flow

2 → slower — " —

⋮ ⋮

k → slowest — " —

eg: Gradient descent

$W_{t+1} = W_t - \eta \nabla L(W_t)$

only W updates : :

freq. lvl = slow

& only one optimization problem

eg: GD + Moretum

$m_{t+1} = \alpha m_t - \eta \nabla L(W_t)$

$W_{t+1} = W_t + m_{t+1}$

m updates faster & W updates slower

∴ Inner optim. → updates m to compress recent gradients

Outer optim. → updates W based on m.

eg: Linear Attention

$M_{t+1} = M_t + v_t k_t^T$, happens every token → very fast

W_K W_V W_Q → slow

∴ attention M is fastest

optimizer is mid

& weights is long term

* OPTIMIZERS as LEARNING MODULES

2.3 ↳ Optimizer is a multi lvl learner, with it's own memory

Eg 1: GD as a learning Module

$$W_{t+1} = W_t - \eta g_t$$

In NL

$$W_{t+1} = \arg\min_W \left[\langle W, g_t \rangle + \frac{1}{2\eta} \|W - W_t\|^2 \right]$$

→ here optimizer solving tiny problem at ever step
key = gradient dirx^n
value = weight correction

Eg 2: SGD+ Momentum

$$m_{t+1} = \beta m_t + (1-\beta) g_t$$

$$W_{t+1} = W_t - \eta m_{t+1}$$

→ momentum memory m stores a compressed history of past gradients.
→ this is faster memory

Inner optim. (update momentum): $m_{t+1} = \arg\min_m \left[\langle m, g_t \rangle + \frac{1}{2(1-\beta)} \|m - m_t\|^2 \right]$

momentum = lvl 1
weights = lvl 2

Eg 3: Adam

m_t = first moment (mean)
v_t = second moment (variance)
W_t = weights

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$W_{t+1} = W_t - \eta \frac{m_t}{\sqrt{v_t} + \varepsilon}$$

→ m_t = fast surprise memory
v_t = slower variance memory
W_t = slow long term memory

Eg 4: Shampoo / Muon (Preconditioners)

$$P_t \approx \sum_i g_i g_i^T$$

→ ~~preconditioner store correlations b/w GDs.~~
→ It is a rich associative memory
→ Adam stores scalar memory
Shampoo stores matrix memory

DATE:

* CONTINUUM MEMORY SYSTEM (CMS)

3.1

Fast memory → update every token
medium → ever few token
slow → per sequence
v. slow → only during training

$\tau \rightarrow$ time const.

$$\therefore M_i(t+1) = \left(1 - \frac{1}{\tau_i}\right) M_i(t) + \frac{1}{\tau_i} F_i(x_i)$$

$M_i \rightarrow$ memory at i ; $F_i \rightarrow$ update signal ; $\tau_i =$ update speed.

~~Each memory receives I/p, previous o/p & it's own memory state.~~

$$h_i(t) = f(W_i h_{i-1}(t), M_i(t))$$
$$M_i(t+) = \alpha_i M_i(t) + (1-\alpha_i) h_i(t)$$

α_i near 0 → fast memory
α_i near 1 → slow memory

* HOPE Arch. [Hierarchial Optimization Process Engine]

3.2

HOPE = CMS + Self modifying sequence model.

1. CMS → memory hierarchy
2. Titas-style self modifying blocks
3. Linear attention → for long context
4. Deep optimizer inside forward pass
5. Model modifies its own internal fx^{n}s.

① I/p goes ~~through encoder~~

$$h_0(t) = \Sigma \cdot E(x_t)$$

② Feature passes CMS

$$h_i(t) = f_i(h_{i-1}(t), M_i(t))$$
$$M_i(t+1) = (1-\alpha_i)\, h_i(t) + \alpha_i M_i(t)$$

③ Self modifying 'Neural learning module'
↳ predicts how to change part of model's own weigh

$$\Delta W_i(t) = U_i(h_i(t))$$
$$W_i(t+1) = W_i(t) + \Delta W_i(t)$$

' U_i is learned

④ Linear Attention
↳ to handle long sequences

$$A(t+1) = A(t) + v(t)\, k(t)^T$$

This is fastest

⑤ Decoder head produces next token

$$y_t = D(h_{final}(t))$$