CPEN 455: Deep Learning
Optimization and Regularization

Companion notes to Lecture 4.2: Backpropagation II

Renjie Liao
With assistance from ChatGPT-6 Astra Ultra.
University of British Columbia Winter Term 1, 2026

1 From Backpropagation to Gradient-based Learning

Backpropagation computes gradients. An *optimizer* uses them to update parameters. These notes follow the lecture's order: SGD and momentum, adaptive methods, Adam, learning-rate schedules, weight decay, Muon, and early stopping.

Mini-batch loss and iteration convention

As in the slides, W^{t-1} denotes the parameters *before* update t, and (X^t, Y^t) is a mini-batch of B examples. Define

$$L(W, X^t, Y^t) = \frac{1}{B}\sum_{i=1}^{B} \ell\big(f(W, X^t[i]), Y^t[i]\big), \qquad g^t = \left.\frac{\partial L(W, X^t, Y^t)}{\partial W}\right|_{W=W^{t-1}}. \tag{1}$$

The superscript t indexes iterations; β^t is a scalar power when β is a momentum coefficient. W may collect all parameters. For coordinatewise methods we view it as a vector, with $W[j]$ selecting a coordinate. In the Muon section, W is one layer's matrix and g^t has the same matrix shape. Square roots, squares, and division of vectors below are elementwise; $\odot$ denotes elementwise multiplication.

Stochastic gradient descent (SGD) performs

$$W^t = W^{t-1} - \eta g^t, \tag{2}$$

with learning rate $\eta > 0$. At each iteration: sample a mini-batch, run a forward pass to compute the loss, run backpropagation to compute g^t, then update W. A uniformly sampled mini-batch gives an unbiased empirical-loss gradient, conditional on the current W, but individual updates are noisy.

Why one learning rate can be restrictive

Consider $L(w) = \frac{1}{2}(aw_1^2 + bw_2^2)$ with $a \gg b > 0$. Gradient descent gives $w_1^t = (1-\eta a)w_1^{t-1}$ and $w_2^t = (1-\eta b)w_2^{t-1}$. Stability requires $0 < \eta < 2/a$. A rate small enough for the steep direction can therefore make progress slow in the shallow direction. This is the ravine illustrated in the slides. Momentum smooths the update direction; adaptive methods also rescale coordinates. Neither guarantees a global optimum for a neural network.

2 Momentum and Nesterov Momentum

The slides use the momentum convention

$$M^t = \beta M^{t-1} + g^t, \qquad W^t = W^{t-1} - \eta M^t, \qquad M^0 = 0, \quad 0 \le \beta < 1. \tag{3}$$

Unrolling the recurrence yields $M^t = \sum_{\tau=1}^{t} \beta^{t-\tau} g^\tau$. Consistently signed gradients accumulate, whereas alternating gradients can cancel. For a constant gradient g, $M^t = (1-\beta^t)g/(1-\beta)$. Thus momentum changes the scale as well as the direction of an update.

An exponential moving average (EMA) instead uses $m^t = \beta m^{t-1} + (1-\beta)g^t$. With zero initialization, $m^t = (1-\beta)M^t$ for the same gradient sequence. The learning rate must be rescaled if the two conventions are to give identical updates. Adam and the simplified Muon derivation below use the EMA convention.

The look-ahead idea

One standard form of Nesterov momentum evaluates the gradient after moving in the previous momentum direction [1]. With a fixed learning rate, write

$$\begin{aligned} \bar{W}^{t-1} &= W^{t-1} - \eta\beta M^{t-1}, \\ g_{\mathrm{N}}^t &= \left.\frac{\partial L(W, X^t, Y^t)}{\partial W}\right|_{W=\bar{W}^{t-1}}, \\ M^t &= \beta M^{t-1} + g_{\mathrm{N}}^t, \qquad\qquad W^t = W^{t-1} - \eta M^t. \end{aligned} \tag{4}$$

The look-ahead gradient can correct a momentum step that would overshoot a valley. The subscript distinguishes this gradient from g^t in (1). Implementations often express Nesterov momentum with a different parameterization of the stored weights and velocity; compare the full recurrences rather than one isolated update line.

3 Adaptive Learning Rates

AdaGrad: accumulate squared gradients

In the slides' vector notation, AdaGrad [2] uses

$$G^t = \sum_{\tau=1}^{t} g^\tau (g^\tau)^\top, \qquad W^t[j] = W^{t-1}[j] - \frac{\eta}{\sqrt{\epsilon + G^t[j,j]}} g^t[j].$$

Only the diagonal is needed. Setting $v^t[j] = G^t[j,j]$ gives the practical form

$$v^t = v^{t-1} + g^t \odot g^t, \qquad W^t = W^{t-1} - \eta \frac{g^t}{\sqrt{\epsilon + v^t}}, \qquad v^0 = 0. \tag{5}$$

The denominator uses *all past squared gradients*, not just $(g^t[j])^2$. Coordinates with larger accumulated gradients receive a smaller effective rate. Storage is linear in the number of parameters; no full outer-product matrix is required. Since $v^t[j]$ never decreases, the effective rate never increases and can become too small during long training.

RMSProp: forget old squared gradients

RMSProp [3] replaces the cumulative sum with an EMA:

$$v^t = \rho v^{t-1} + (1-\rho)(g^t \odot g^t), \qquad W^t = W^{t-1} - \eta \frac{g^t}{\sqrt{\epsilon + v^t}}. \tag{6}$$

A gradient from k steps ago has weight proportional to ρ^k; a larger ρ remembers a longer history. The denominator reflects recent gradient magnitudes instead of growing indefinitely. This is a second *raw moment*, not a variance: variance would also subtract the squared mean.

Adadelta: track the scale of parameter updates as well

Adadelta [4] adds an EMA u^t of squared parameter changes. One standard form, with $u^0 = v^0 = 0$, is

$$v^t = \rho v^{t-1} + (1-\rho)(g^t \odot g^t),$$
$$\Delta W^t = -\frac{\sqrt{u^{t-1}+\epsilon}}{\sqrt{v^t+\epsilon}} \odot g^t, \qquad W^t = W^{t-1} + \Delta W^t,$$
$$u^t = \rho u^{t-1} + (1-\rho)(\Delta W^t \odot \Delta W^t).$$

The previous update scale supplies the numerator, while the current gradient scale supplies the denominator. This historical variant helps motivate adaptive scaling; Adam is the main adaptive optimizer developed next.

4 Adam and Moment Bias Correction

Adam, short for *adaptive moment estimation*, combines a first-moment EMA with a second-moment EMA [5]. The notation in the slides maps to

$$m^t = \tilde{g}^t, \qquad v^t[j] = G^t[j,j].$$

With $m^0 = v^0 = 0$, the state updates are

$$m^t = \beta_1 m^{t-1} + (1-\beta_1) g^t, \tag{7}$$
$$v^t = \beta_2 v^{t-1} + (1-\beta_2)(g^t \odot g^t). \tag{8}$$

Typical starting values in the original algorithm are $\beta_1 = 0.9$ and $\beta_2 = 0.999$; they are hyperparameters, not requirements. Setting $\beta_1 = 0$ removes first-moment averaging, giving an RMSProp-like update with a second-moment bias correction.

Where the correction factors come from

Consider one coordinate. Suppose its gradient has the same first and second moments at each iteration in this calculation: $\mathbb{E}[g^\tau] = \mu$ and $\mathbb{E}[(g^\tau)^2] = \nu$. These assumptions do not require gradients at different iterations to be independent. Unrolling (7) and using linearity of expectation gives

$$m^t = (1-\beta_1)\sum_{\tau=1}^{t} \beta_1^{t-\tau} g^\tau,$$
$$\mathbb{E}[m^t] = (1-\beta_1)\mu \sum_{k=0}^{t-1} \beta_1^k = (1-\beta_1^t)\mu. \tag{9}$$

The same argument gives $\mathbb{E}[v^t] = (1-\beta_2^t)\nu$. The missing initial history makes the EMA weights sum to less than one. Dividing by that sum yields

$$\hat{m}^t = \frac{m^t}{1-\beta_1^t}, \qquad \hat{v}^t = \frac{v^t}{1-\beta_2^t}, \qquad \mathbb{E}[\hat{m}^t] = \mu, \quad \mathbb{E}[\hat{v}^t] = \nu. \tag{10}$$

Under these assumptions, the moment estimates are unbiased. During actual training, the gradient distribution changes as W changes. The correction still normalizes the EMA weights, but it does not make the weighted history equal to the current iteration's moment.

The canonical coordinatewise Adam update is

$$\boxed{W^t = W^{t-1} - \eta \frac{\hat{m}^t}{\sqrt{\hat{v}^t}+\epsilon}.} \tag{11}$$

At $t = 1$, $\hat{m}^1 = g^1$ and $\hat{v}^1 = g^1 \odot g^1$. Consequently the first update in a nonzero coordinate is approximately $-\eta \operatorname{sign}(g^1[j])$ when ϵ is negligible. This is a useful check on the correction factors.

Why remove this initialization bias? The correction removes a startup distortion caused solely by the zero initial states. It makes the normalized EMA weights sum to one; it does not aim to keep the actual expected update constant throughout training. For a constant nonzero coordinate gradient g, $m^t = (1-\beta_1^t)g$ and $v^t = (1-\beta_2^t)g^2$, so

$$\frac{m^t}{\sqrt{v^t}} = \frac{1-\beta_1^t}{\sqrt{1-\beta_2^t}} \operatorname{sign}(g), \qquad \frac{\hat{m}^t}{\sqrt{\hat{v}^t}} = \operatorname{sign}(g).$$

The raw ratio varies with t even though the gradient is unchanged. With $\beta_1 = 0.9$ and $\beta_2 = 0.999$, its first-step magnitude is $0.1/\sqrt{0.001} \approx 3.16$, compared with 1 after correction, ignoring epsilon.

Relating epsilon and learning-rate conventions in the slides

The lecture first places ϵ inside $\sqrt{\epsilon + G^t[j,j]}$ and later uses $\sqrt{v^t[j]} + \epsilon$ in its algorithm box. These are distinct stability conventions; the same numerical ϵ does not make them identical. For the canonical update (11), an exact rearrangement is

$$\eta \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \epsilon} = \eta \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \frac{m^t}{\sqrt{v^t} + \epsilon\sqrt{1-\beta_2^t}}. \tag{12}$$

Thus pulling the correction into the learning rate also rescales epsilon. Ignoring epsilon recovers the slides' correction multiplier $\eta_t = \eta\sqrt{1-\beta_2^t}/(1-\beta_1^t)$ exactly. These notes use (11) when writing Adam or AdamW; its moment correction is the central idea, independent of the numerical-stability convention.

Motivating the correction factor

Following the coordinatewise calculation in the slides, define the signed amount subtracted from $W^{t-1}[j]$ as

$$\Delta W^t[j] = \eta_t \frac{\tilde{g}^t[j]}{\sqrt{\epsilon + G^t[j,j]}}, \qquad W^t[j] = W^{t-1}[j] - \Delta W^t[j].$$

Use the stationary first and second moments above, now written explicitly for coordinate j: $\mu = \mathbb{E}[g^i[j]]$ and $\nu = \mathbb{E}[(g^i[j])^2] > 0$. Then

$$\mathbb{E}[\tilde{g}^t[j]] = (1-\beta_1^t)\mu, \qquad \mathbb{E}[G^t[j,j]] = (1-\beta_2^t)\nu.$$

For this calculation, make the *simplifying independence assumption* that $\tilde{g}^t[j]$ and $G^t[j,j]$ are independent, and approximate the positive second-moment state as concentrated around its mean, sufficiently far from zero. Assume the inverse moments below exist and epsilon is small relative to $G^t[j,j]$ over the relevant range. These assumptions give

$$\begin{aligned}
\mathbb{E}[\Delta W^t[j]] &\approx \eta_t \mathbb{E}\left[\tilde{g}^t[j](G^t[j,j])^{-1/2}\right] \\
&\approx \eta_t \mathbb{E}[\tilde{g}^t[j]]\, \mathbb{E}[(G^t[j,j])^{-1/2}] \\
&\approx \eta_t \frac{\mathbb{E}[\tilde{g}^t[j]]}{\sqrt{\mathbb{E}[G^t[j,j]]}} \\
&= \eta_t \frac{1-\beta_1^t}{\sqrt{1-\beta_2^t}} \frac{\mu}{\sqrt{\nu}}.
\end{aligned} \tag{13}$$

Under the simplifying independence assumption, the second line is an equality: independence also holds for any function of $G^t[j,j]$, including its inverse square root. In actual Adam, both states use the same gradient history; the factorization therefore models their joint behavior approximately. Stationary gradient moments alone do not imply independence or concentration.

Why concentration permits the inverse-square-root approximation

Write $V = G^t[j,j] > 0$, $a = \mathbb{E}[V] > 0$, and $\delta = (V-a)/a$. Thus $V = a(1+\delta)$ and $\mathbb{E}[\delta] = 0$. A Taylor expansion around $\delta = 0$ gives

$$V^{-1/2} = a^{-1/2}\left(1 - \frac{\delta}{2} + \frac{3\delta^2}{8} + O(|\delta|^3)\right).$$

For example, $|\delta| \le r < 1$ ensures a remainder bound uniform over the possible values of V, with a constant depending on r. Taking expectations,

$$\mathbb{E}[V^{-1/2}] = \frac{1}{\sqrt{a}}\left(1 + \frac{3\,\mathrm{Var}(V)}{8a^2} + O(\mathbb{E}[|\delta|^3])\right). \tag{14}$$

The linear fluctuation cancels because $\mathbb{E}[\delta] = 0$. The leading relative correction is $3\,\mathrm{Var}(V)/(8a^2)$: if fluctuations are small relative to the mean and the remainder is controlled, then $\mathbb{E}[V^{-1/2}] \approx 1/\sqrt{\mathbb{E}[V]}$. Averaging squared gradients can make this approximation useful when enough weakly dependent contributions are averaged. It is not automatic: values near zero produce a large inverse square root. Without the bounded-fluctuation condition above, the contribution of such values and other tails must also be controlled; small variance alone is insufficient.

The function $f(v) = v^{-1/2}$ has $f''(v) = 3/(4v^{5/2}) > 0$ for $v > 0$. Jensen's inequality therefore gives the exact *lower* bound

$$\mathbb{E}[V^{-1/2}] \ge (\mathbb{E}[V])^{-1/2},$$

with equality only when V is constant almost surely. It indicates the direction of the approximation: replacing the expectation by $1/\sqrt{a}$ underestimates this positive factor. Closeness comes from concentration, as quantified by the Taylor expansion; Jensen's inequality alone does not establish closeness.

Canceling the zero-initialization factors

The factor $(1-\beta_1^t)/\sqrt{1-\beta_2^t}$ in (13) comes from initializing both moment estimates at zero. Choose the learning-rate multiplier to cancel it:

$$\boxed{\eta_t = \eta\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}} \quad\Longrightarrow\quad \mathbb{E}[\Delta W^t[j]] \approx \eta\frac{\mu}{\sqrt{\nu}}. \tag{15}$$

This choice cancels the transient introduced by zero initialization; changes in the gradient distribution during training can still change the expected update. The calculation motivates the same correction factor as the moment normalization in (10). Normalizing the moments removes their zero-initialization bias exactly under stationary moments, without needing independence or concentration. The expected-update calculation additionally uses those approximations and neglects epsilon where it is small.

5 Learning-rate Schedules

Adaptive coordinate scaling does not remove the need to choose a base learning rate. A schedule changes that global rate over training. Here η_t denotes this *scheduled base rate*, not the bias-correction multiplier called η_t in the earlier Adam derivation. Replace η by η_t in (11) and retain $\hat{m}^t, \hat{v}^t$.

Linear warmup [6] gradually increases the learning rate during the first T_w updates. Cosine decay [7] then decreases it smoothly. One continuous piecewise schedule sampled at iterations $t = 1, \dots, T$ is

$$\eta_t = \begin{cases} \eta_{\max}\, t/T_w, & 0 \le t \le T_w, \\ \eta_{\min} + \dfrac{\eta_{\max} - \eta_{\min}}{2}\left[1 + \cos\left(\pi\dfrac{t - T_w}{T - T_w}\right)\right], & T_w < t \le T. \end{cases} \tag{16}$$

Here $0 < T_w < T$; the plot includes $t = 0$ to show the starting value. Both branches meet at $\eta_{\max}$, and the rate reaches $\eta_{\min}$ at T. Warmup can temper large early updates while optimizer states and activations are settling. Decay makes later updates smaller. The lecture uses one decay segment; SGDR also studies restarting the cosine schedule. The warmup duration, peak rate, and total budget are validation choices, not universal constants.

6 Weight Decay and AdamW

An L_2 penalty changes the loss gradient

Let L be the data loss. Penalizing the selected weights gives

$$\tilde{L}(W) = L(W) + \frac{\lambda}{2}\|\operatorname{Vec}(W)\|_2^2, \qquad \frac{\partial \tilde{L}}{\partial W} = \frac{\partial L}{\partial W} + \lambda W. \tag{17}$$

The factor $1/2$ makes the penalty gradient exactly λW. Without it, the gradient is $2\lambda W$; this is a change in the definition of λ, not a different regularizer.

For plain SGD without momentum, applying this gradient gives

$$W^t = W^{t-1} - \eta_t(g^t + \lambda W^{t-1}) = (1 - \eta_t\lambda)W^{t-1} - \eta_t g^t.$$

Thus an L_2 penalty and multiplicative decay are equivalent here when their coefficients are matched. This simple equivalence does not automatically extend to momentum or adaptive preconditioning.

Coupled Adam: the penalty enters both moment estimates

If Adam optimizes $\tilde{L}$, it receives the penalized gradient

$$r^t = g^t + \lambda W^{t-1},$$
$$m^t = \beta_1 m^{t-1} + (1 - \beta_1)r^t, \qquad v^t = \beta_2 v^{t-1} + (1 - \beta_2)(r^t \odot r^t).$$

Adam then bias-corrects these states and applies (11). The penalty contributes to both the smoothed direction and the denominator. In particular,

$$r^t \odot r^t = g^t \odot g^t + 2\lambda(g^t \odot W^{t-1}) + \lambda^2(W^{t-1} \odot W^{t-1}).$$

Its effect is history-dependent and coordinate-dependent; it is not simply the same shrinkage factor applied to every selected weight.

AdamW: decay is separate from the moment estimates

AdamW [8] computes m^t, v^t from the *data gradient* g^t using (7)–(8). It applies shrinkage directly to the parameters:

$$\boxed{W^t = (1 - \eta_t\lambda)W^{t-1} - \eta_t \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \epsilon}.} \tag{18}$$

The decisive difference is *what enters the optimizer state*, not merely which line of code is written first. The formula evaluates both terms using the old weights; shrinking the already-updated weights introduces an extra cross-term. The notation here absorbs the learning rate into the shrinkage factor as shown; other conventions name the shrinkage coefficient differently.

A scalar example makes the distinction visible. At the first step, let $W^0 = 2$, $g^1 = 0$, $\lambda = 0.1$, $\eta = 0.01$, and a small positive ϵ. Coupled Adam sees $r^1 = 0.2$, so its normalized update has magnitude $0.01 \times 0.2/(0.2 + \epsilon) \approx 0.01$ and

gives $W^1 \approx 1.99$. AdamW has data-gradient update $0/(0+\epsilon) = 0$ and gives $W^1 = (1 - 0.001)2 = 1.998$. Both can reduce weights, but they implement different rules.

With momentum, an L_2 term added to the gradient is also accumulated in the momentum state, whereas decoupled decay is not; do not assume they coincide. In practice, specify which parameter groups receive decay. Bias and normalization parameters are often excluded, but this is a modeling choice. Tune λ using validation performance; smaller weight norm alone is not a guarantee of better generalization.

7 Muon: Learning with Matrix Updates

Adam rescales individual coordinates. Muon instead uses the matrix structure of a layer's update [9]. Consider

$$W^{t-1}, g^t, m^t \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \qquad m^t = \beta m^{t-1} + (1-\beta) g^t.$$

The capital G^t from the adaptive methods is a second-moment state; it is *not* the matrix gradient here. Muon transforms the momentum matrix m^t before applying an update. It does not constrain the weight matrix itself to stay orthogonal.

Scope in this lecture. We apply Muon only to hidden linear-layer weight matrices. In $y = Wx$, the matrix maps input features to output features, so the SVD/polar transformation acts on meaningful paired directions of the update. Bias and normalization parameters are scalars or vectors without this matrix mixing structure; use AdamW for them. This is our teaching setup: convolution kernels can also be flattened into matrices for Muon [9].

The SVD separates directions from strengths

For a full-rank rectangular matrix m, let $q = \min(d_{\text{out}}, d_{\text{in}})$. Its thin singular value decomposition (SVD) is

$$m = U \Sigma V^\top = \sum_{i=1}^{q} \sigma_i u_i v_i^\top, \qquad U^\top U = V^\top V = I_q, \quad \sigma_i > 0. \tag{19}$$

Here $U \in \mathbb{R}^{d_{\text{out}} \times q}$ and $V \in \mathbb{R}^{d_{\text{in}} \times q}$ have orthonormal columns; $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_q)$. For an input x, $V^\top$ finds its coordinates along input directions, Σ stretches those coordinates, and U maps them to output directions. The pair (v_i, u_i) specifies a mode of the linear map, and σ_i its strength.

The idealized orthogonalized update is

$$Q = UV^\top = \sum_{i=1}^{q} u_i v_i^\top. \tag{20}$$

It retains the paired directions but replaces each positive singular value by one. If the matrix is tall, $Q^\top Q = I$; if it is wide, $QQ^\top = I$. For a square full-rank matrix both identities hold. This is the polar factor of m and the closest such semi-orthogonal matrix in Frobenius distance. Its nonzero update modes therefore have equal strength.

The lecture's 2×2 example

Let

$$m = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad U = V = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}.$$

Along $v_1 = (1,1)^\top/\sqrt{2}$, the map multiplies by three; along $v_2 = (1,-1)^\top/\sqrt{2}$, it multiplies by one. Therefore

$$m = 3u_1 v_1^\top + u_2 v_2^\top, \qquad Q = u_1 v_1^\top + u_2 v_2^\top = I_2.$$

Dividing m by its Frobenius norm gives $m/\sqrt{10}$ and preserves the $3:1$ ratio. Orthogonalization changes that ratio to $1:1$. The matrix entries need not be equal: equal singular values concern *directions*, not individual entries.

Why equalize the update's singular values?

If a few singular values of m dominate, most of its effect lies in a few paired input/output directions. Dividing by one matrix norm preserves this imbalance. Replacing each positive singular value by one preserves the pairs (v_i, u_i) while increasing the importance of weaker modes *relative to* stronger ones. It need not increase their absolute step size, which also depends on the learning rate and matrix scaling. This can help useful directions that would otherwise receive little emphasis, a motivation suggested by the original Muon report [9]. It can also emphasize noisy directions; equalization is not a universal improvement guarantee.

There is also a local optimization interpretation [14]. For a signed weight change D, define the operator norm

$$\|D\|_{\mathrm{op}} = \max_{x \neq 0} \frac{\|Dx\|_2}{\|x\|_2} = \sigma_{\max}(D).$$

For the lecture's matrix, $m/\sigma_{\max}(m) = m/3$ has singular values $(1, 1/3)$, whereas Q has $(1, 1)$. Both have operator norm one: under the same maximum-stretch budget, Q gives the weaker mode equal strength.

For a linear layer with fixed input x, changing W to $W + D$ changes its output by Dx. Thus $\|D\|_{\mathrm{op}} \leq \eta$ ensures $\|Dx\|_2 \leq \eta\|x\|_2$ for every input. Under this constraint, minimize the first-order change in loss:

$$\min_{\|D\|_{\mathrm{op}} \leq \eta} \langle g, D\rangle_F, \qquad L(W + D) \approx L(W) + \langle g, D\rangle_F, \qquad \langle g, D\rangle_F = \mathrm{tr}(g^\top D). \tag{21}$$

For the compact SVD $g = U_g \Sigma_g V_g^\top$ with positive singular values σ_i, the unit singular vectors give

$$\langle g, D\rangle_F = \sum_i \sigma_i u_{g,i}^\top D v_{g,i} \geq -\eta \sum_i \sigma_i, \qquad D_* = -\eta U_g V_g^\top$$

since $|u_{g,i}^\top D v_{g,i}| \leq \|D\|_{\mathrm{op}} \leq \eta$. The displayed D_* attains equality; choose $D_* = 0$ if $g = 0$. This explains the polar direction as a solution to a local problem with a bound on directional output change. Muon substitutes the smoothed momentum m for g, so it solves a *proxy* problem; the result need not be a descent direction for the current gradient. Neither this linear approximation nor momentum alone guarantees a decrease in the actual loss for a finite step.

Efficient approximation by Newton–Schulz

An SVD gives the polar factor directly, but computing one for every layer at every training step can be expensive. Newton–Schulz uses polynomial matrix products, which GPUs execute efficiently [9]. A small fixed number of iterations can give a useful approximation. It is a computational choice: orthogonalization does not mathematically require Newton–Schulz.

First study the classical cubic iteration. For $m \neq 0$,

$$X_0 = \frac{m}{\|m\|_F}, \qquad X_{k+1} = \frac{3}{2} X_k - \frac{1}{2}(X_k X_k^\top) X_k. \tag{22}$$

Normalization ensures that every singular value of X_0 lies in $[0, 1]$, because $\|m\|_F^2 = \sum_i \sigma_i^2 \geq \sigma_{\max}^2$. The iteration preserves the singular vectors. Substituting $X_k = U\,\mathrm{diag}(s_{i,k})V^\top$ into (22) reduces it to independent scalar maps:

$$s_{i,k+1} = f(s_{i,k}), \qquad f(s) = \frac{3}{2}s - \frac{1}{2}s^3. \tag{23}$$

For $0 < s < 1$,

$$f(s) - s = \frac{1}{2}s(1 - s^2) > 0, \qquad 1 - f(s) = \frac{1}{2}(1 - s)^2(s + 2) \geq 0.$$

Each positive singular value increases and stays at most one, so it has a limit $\ell > 0$. Continuity gives $\ell = f(\ell)$, or $\ell(1-\ell^2) = 0$; hence $\ell = 1$. A singular value already equal to one stays one, while a zero singular value stays zero. Therefore, for rank r_0,

$$\boxed{X_k \longrightarrow U_{r_0} V_{r_0}^\top = Q.} \tag{24}$$

For a full-rank rectangular matrix, this is the semi-orthogonal factor described above. When $r_0 < \min(d_{\text{out}}, d_{\text{in}})$, it is a *partial isometry*: it preserves lengths on the retained input subspace and maps the orthogonal complement to zero. The iteration cannot create a missing singular direction. If $m = 0$, set $Q = 0$ explicitly instead of dividing by its norm.

Near one, writing $e = 1 - s$ gives $1 - f(s) = \frac{3}{2}e^2 - \frac{1}{2}e^3$, so the error decreases quadratically once it is small. The lecture's example starts at $(3/\sqrt{10}, 1/\sqrt{10})$; the smaller singular value takes more iterations to approach one. This is convergence of the *inner orthogonalization routine* for a fixed matrix, not a claim that neural network training converges to an optimum.

A teaching algorithm and practical distinctions

For a selected hidden-layer matrix, the simplified update is

$$g^t = \left.\frac{\partial L(W, X^t, Y^t)}{\partial W}\right|_{W=W^{t-1}}, \qquad m^t = \beta m^{t-1} + (1-\beta) g^t,$$
$$Q^t \approx \mathrm{Ortho}(m^t), \qquad W^t = (1 - \eta_t \lambda) W^{t-1} - \eta_t Q^t.$$

Use the zero branch above and a fixed number of polynomial steps for the approximation. The original practical Muon implementation uses a Nesterov-style momentum update, a tuned finite-step polynomial, and parameter-specific conventions [9]. In particular, its quintic polynomial with coefficients $(3.4445, -4.7750, 2.0315)$ targets useful approximate orthogonalization in a small number of steps. It is *not* the convergent cubic in (22); its scalar map sends 1 to 0.701, so exact convergence to one is not its design criterion.

Another practical issue is matrix shape. For a full-rank ideal Q of shape $a \times b$, with $q = \min(a, b)$ as above, $\|Q\|_F = \sqrt{q}$, so its root-mean-square entry is

$$\mathrm{RMS}(Q) = \frac{\|Q\|_F}{\sqrt{ab}} = \frac{1}{\sqrt{\max(a,b)}}.$$

A factor proportional to $\sqrt{\max(a,b)}$ can therefore equalize this entry scale across matrix shapes. The scalable Muon variant of Liu et al. [10] uses a factor $0.2\sqrt{\max(a,b)}$ to target a chosen scale; the constant is an empirical design choice. Such scaling multiplies Q^t in the teaching update. Embedding and output-head matrices commonly also use AdamW [9, 10]. This is a practical parameter-grouping choice, not a mathematical restriction on orthogonalizing those matrices.

Results and extensions

Liu et al. [10] report fitted dense-model scaling curves over 399M–1.5B non-embedding parameters with approximately 52% of AdamW's training FLOPs at matched loss. This is a result for their setup, not a universal twofold wall-clock speedup: matrix operations, hardware, tuning, and target quality matter. *Polar Express* [11] studies improved polynomial constructions for computing polar factors. *MuonClip*, introduced with Kimi K2 [12], addresses attention-score instability in large Transformer training. These extensions build on the same undergraduate idea: separate matrix directions from their strengths, then design a practical transformation of the update.

8 Early Stopping and Checkpoint Selection

Training loss measures the objective being optimized. A separate validation set helps detect when further training no longer improves generalization. Early stopping chooses a training duration using validation performance; it does not use test performance. Common implementations expose patience, a minimum change, and checkpoint-restoration options [13]; the following rule makes each decision explicit.

Choose a validation metric (here lower is better), an evaluation interval, a *patience* integer $p \geq 1$ measured in evaluations, and a minimum meaningful improvement $\delta \geq 0$. Maintain a significant-improvement reference b, a counter c, and a separate best observed validation value b_{best} with its checkpoint:

1. Initialize $b = b_{\text{best}} = +\infty$ and $c = 0$.
2. At each validation evaluation, compute z without training on validation data. If $z < b_{\text{best}}$, save the model and set $b_{\text{best}} = z$.
3. If $z < b - \delta$, set $b = z$ and reset $c = 0$; otherwise increment c.
4. Stop when $c \geq p$, or when the maximum training budget is reached. Restore the checkpoint with the smallest observed validation metric b_{best}.

The two records distinguish the best checkpoint from improvements large enough to reset patience. Tiny cumulative gains can eventually beat b by δ. For example, with $p = 3$, $\delta = 0.01$, and losses $0.500, 0.460, 0.455, 0.456, 0.457$, the counters are $0, 0, 1, 2, 3$. Stop at the fifth evaluation and restore the third checkpoint (loss 0.455). The small improvement at the third evaluation changes the saved checkpoint but does not reset patience. Implementations differ in strictness and whether they save every best value; state the rule being used.

Use evaluation mode and a consistent validation procedure so that dropout, batch-normalization updates, or a changing sample do not obscure the comparison. Patience must allow for noise and the learning-rate schedule; stopping too soon can miss a later gain after rate decay. If this is a concern, train for the full budget and still restore the best validation checkpoint, as suggested in the slides. For resuming training, save optimizer and schedule states too. Use the test set only for the final evaluation after these choices are fixed.

9 Self-Check Questions

Answers are in Appendix A.

1. Why does diagonal AdaGrad require only one extra number per parameter even though the slides introduce a matrix G^t?
2. For zero-initialized Adam, compute $\hat{m}^1$ and $\hat{v}^1$. Using the expected-update calculation, derive the correction factor in η_t. Which additional approximations does this calculation use?
3. Which gradient enters m^t and v^t in coupled Adam with an L_2 penalty? Which gradient enters them in AdamW?
4. For $m = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$, compare the singular values of $m/\|m\|_F$ and its ideal orthogonalized update.
5. What happens to a zero singular value under the cubic Newton–Schulz iteration? What should an implementation do when the whole momentum is zero?
6. With patience $p = 2$, $\delta = 0.01$, and validation losses $0.50, 0.495, 0.493$, when does the stated early-stopping rule stop, and which checkpoint does it restore?

A Self-Check Answers

1. Only $G^t[j,j] = \sum_{\tau=1}^{t}(g^\tau[j])^2$ appears in the diagonal update. Store these entries as v^t and update them elementwise; do not construct the full outer-product matrix.

2. $\hat{m}^1 = g^1$ and $\hat{v}^1 = g^1 \odot g^1$. The expected-update calculation gives $\mathbb{E}[\Delta W^t[j]] \approx \eta_t(1-\beta_1^t)\mu/(\sqrt{1-\beta_2^t}\sqrt{\nu})$. Choosing $\eta_t = \eta\sqrt{1-\beta_2^t}/(1-\beta_1^t)$ cancels the zero-initialization factors. Besides stationary gradient moments, this calculation assumes independence between the two moment states, concentration of the second-moment state away from zero with controlled fluctuations, and negligible epsilon. Independence is a simplifying model: the actual states share a gradient history. Exact normalization of the stationary moment estimates does not require these additional assumptions.

3. Coupled Adam uses $r^t = g^t + \lambda W^{t-1}$ in both states, including $r^t \odot r^t$ in the second moment. AdamW uses g^t and separately applies $-\eta_t\lambda W^{t-1}$ in the weight update.

4. Normalization gives $(3/\sqrt{10}, 1/\sqrt{10})$, retaining the $3:1$ ratio. The orthogonalized update is I_2, with singular values $(1,1)$.

5. Since $f(0) = 0$, a zero singular value stays zero. If the whole matrix is zero, set the transformed update to zero instead of normalizing it.

6. The first value initializes $b = b_{\text{best}} = 0.50$ and $c = 0$. The next two values improve b_{best} but do not beat $b - \delta = 0.49$, so c reaches two at the third evaluation. Stop there and restore the third checkpoint, whose loss is 0.493, the smallest observed value. Patience and checkpointing serve different purposes.

References

[1] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning." *ICML*, PMLR 28(3):1139–1147, 2013. Paper and PDF.

[2] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *JMLR*, 12(61):2121–2159, 2011. Paper and PDF.

[3] T. Tieleman and G. Hinton. "Lecture 6.5: rmsprop." *Neural Networks for Machine Learning*, 2012. Lecture slides.

[4] M. D. Zeiler. "ADADELTA: An adaptive learning rate method." 2012. arXiv:1212.5701.

[5] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." *ICLR*, 2015. arXiv:1412.6980.

[6] P. Goyal, P. Dollár, R. Girshick, et al. "Accurate, large minibatch SGD: Training ImageNet in 1 hour." 2017. arXiv:1706.02677.

[7] I. Loshchilov and F. Hutter. "SGDR: Stochastic gradient descent with warm restarts." *ICLR*, 2017. arXiv:1608.03983.

[8] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization." *ICLR*, 2019. arXiv:1711.05101.

[9] K. Jordan, Y. Jin, V. Boza, et al. "Muon: An optimizer for hidden layers in neural networks." 2024. Original article and implementation.

[10] J. Liu, J. Su, X. Yao, et al. "Muon is scalable for LLM training." 2025. arXiv:2502.16982.

[11] N. Amsel, D. Persson, C. Musco, and R. M. Gower. "The Polar Express: Optimal matrix sign methods and their application to the Muon algorithm." 2025. arXiv:2505.16932.

[12] Kimi Team. "Kimi K2: Open agentic intelligence." 2025. arXiv:2507.20534.

[13] Keras Team. "EarlyStopping." *Keras API documentation*, accessed September 2026. Callback options.

[14] J. Bernstein. "Deriving Muon." 7 March 2025. Derivation of the update geometry.