Optimization and Regularization Techniques in Practice

October 22, 2025

Contents

1 Adaptive Optimizers: Adam, RMSprop, and Beyond

Gradient-based optimization adapts learning rates to the geometry of the loss surface. Adaptive optimizers rescale parameter-specific learning rates using running statistics of the gradients, enabling faster convergence on ill-conditioned objectives.

1.1 RMSprop

RMSprop maintains an exponential moving average of squared gradients:

$$\mathbf{v}_t = \rho \mathbf{v}_{t-1} + (1-\rho)\nabla_{\boldsymbol{\theta}}\mathcal{L}_t \odot \nabla_{\boldsymbol{\theta}}\mathcal{L}_t, \tag{1}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\nabla_{\boldsymbol{\theta}}\mathcal{L}_t}{\sqrt{\mathbf{v}_t + \epsilon}}, \tag{2}$$

with decay $\rho \approx 0.9$ and numerical stabilizer $\epsilon \approx 10^{-8}$. The adaptive denominator dampens updates along directions with persistent large gradients.

1.2 Adam and AdamW

Adam augments RMSprop with momentum by tracking both first and second moments:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\mathcal{L}_t, \tag{3}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\nabla_{\boldsymbol{\theta}}\mathcal{L}_t \odot \nabla_{\boldsymbol{\theta}}\mathcal{L}_t. \tag{4}$$

Bias correction compensates for the initialization at zero:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \qquad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}. \tag{5}$$

The parameter update becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}. \tag{6}$$

AdamW decouples weight decay from the adaptive gradient by applying L2 regularization separately:

$$\boldsymbol{\theta}_{t+1} = (1-\eta\lambda)\boldsymbol{\theta}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}, \tag{7}$$

where λ is the weight decay coefficient. Decoupling preserves the intended magnitude control while retaining Adam's adaptive step.

1.3 AdaBelief, AdaFactor, and Yogi

Modern variants tweak moment tracking for stability in large-scale training.

- **AdaBelief** replaces squared gradients with the squared deviation from the first moment, reducing variance in stationary regions.
- **AdaFactor** factorizes the second-moment matrix into row and column statistics, dramatically lowering memory footprint for transformers.
- **Yogi** adds a signed adjustment to the second moment, preventing $\mathbf{v}_t$ from growing unbounded and stabilizing training on sparse gradients.

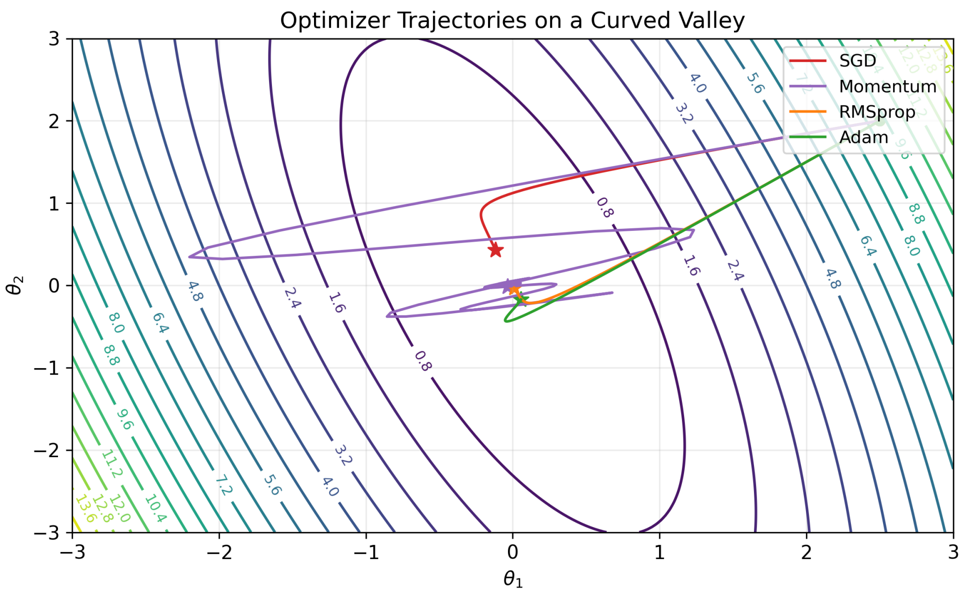

Figure 1: Optimization trajectories on a valley-shaped loss for SGD, momentum, RMSprop, and Adam. Adaptive methods align steps with narrow curvature directions.

1.4 Implementation Considerations

Large-scale training stacks multiple tricks: gradient clipping, mixed precision, and decoupled weight decay. The following snippet shows an AdamW optimizer with gradient clipping and cosine annealing:

Listing 1: AdamW with gradient clipping and cosine learning rate schedule.

```
import torch
from torch.nn.utils import clip_grad_norm_
from torch.optim.lr_scheduler import CosineAnnealingLR

optimizer = torch.optim.AdamW(model.parameters(), lr=3e-4,
                              betas=(0.9, 0.999), eps=1e-8, weight_decay=0.01)
scheduler = CosineAnnealingLR(optimizer, T_max=1000, eta_min=1e-5)

for step, batch in enumerate(dataloader, start=1):
    loss = compute_loss(model, batch)
    loss.backward()
    clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    scheduler.step()
```

```
    optimizer.zero_grad()
```

2 Batch Normalization and Layer Normalization

Normalization stabilizes the distribution of activations, reducing covariate shift between layers and accelerating convergence.

2.1 Batch Normalization

Given a mini-batch $\mathcal{B} = \{\mathbf{h}_i\}_{i=1}^m$, BatchNorm normalizes each feature dimension:

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{h}_i, \qquad \boldsymbol{\sigma}^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{h}_i - \boldsymbol{\mu}_{\mathcal{B}})^{\odot 2}, \tag{8}$$

$$\hat{\mathbf{h}}_i = \frac{\mathbf{h}_i - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}^2_{\mathcal{B}} + \epsilon}}, \qquad \mathbf{y}_i = \boldsymbol{\gamma} \odot \hat{\mathbf{h}}_i + \boldsymbol{\beta}. \tag{9}$$

Running averages of $\boldsymbol{\mu}_{\mathcal{B}}$ and $\boldsymbol{\sigma}^2_{\mathcal{B}}$ provide consistent statistics during inference. BN implicitly regularizes by injecting noise from batch sampling, improving generalization but complicating recurrent models and tiny-batch regimes.

2.2 Layer Normalization

LayerNorm operates across features within a single example, avoiding dependence on batch size. For activations $\mathbf{h}$ of dimension d,

$$\mu = \frac{1}{d}\sum_{j=1}^{d} h_j, \quad \sigma^2 = \frac{1}{d}\sum_{j=1}^{d}(h_j - \mu)^2, \quad \hat{h}_j = \frac{h_j - \mu}{\sqrt{\sigma^2 + \epsilon}}. \tag{10}$$

A shared pair of learnable parameters (γ, β) rescales and recenters the normalized output. LN excels in transformer architectures and autoregressive models where batch statistics vary greatly.

2.3 Comparative Analysis

- **Sensitivity to batch size:** BN degrades when batch statistics are noisy; LN remains stable.
- **Regularization effect:** BN's stochasticity acts as implicit regularization, often reducing the need for dropout.
- **Computation:** BN requires synchronizing statistics across devices for data-parallel training, whereas LN is embarrassingly parallel but adds per-sample overhead.

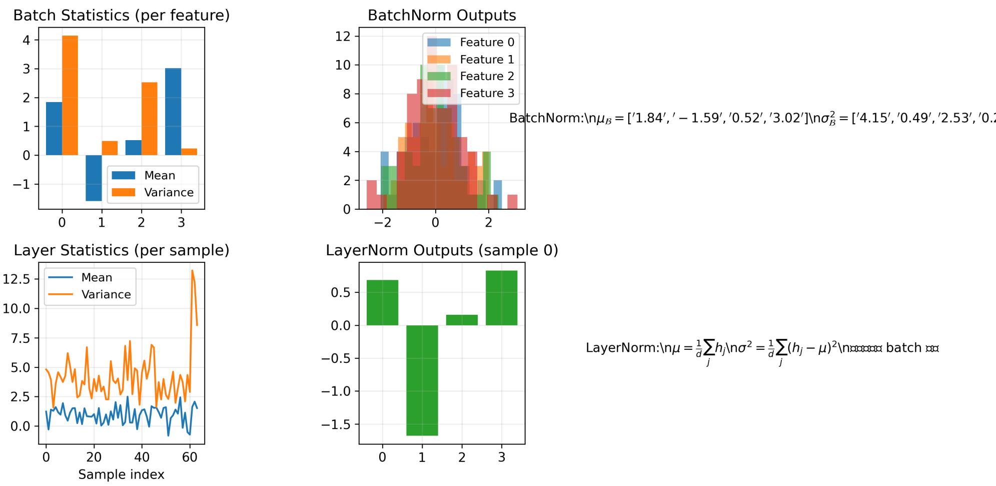

Figure 2: Distribution shift before and after BatchNorm and LayerNorm. BN uses batch-wide statistics, while LN normalizes each token independently.

3 Learning Rate Scheduling and Warm-up

Learning rate schedules balance fast initial progress with stable convergence. Warm-up mitigates the mismatch between randomly initialized parameters and adaptive optimizers that rely on accumulated statistics.

3.1 Step, Exponential, and Polynomial Decay

Step decay multiplies η_t by $\gamma < 1$ every k epochs:

$$\eta_t = \eta_0 \gamma^{\lfloor \frac{t}{k} \rfloor}. \tag{11}$$

Exponential decay adjusts the rate continuously:

$$\eta_t = \eta_0 \exp(-\lambda t). \tag{12}$$

Polynomial decay interpolates between η_0 and η_{end} over T steps:

$$\eta_t = \eta_{\text{end}} + (\eta_0 - \eta_{\text{end}}) \left(1 - \frac{t}{T}\right)^p. \tag{13}$$

3.2 Cosine Annealing and Cyclical Policies

Cosine annealing smoothly decays the learning rate to $\eta_{\min}$:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min}) \left(1 + \cos \frac{\pi t}{T}\right). \tag{14}$$

Cyclical learning rates oscillate between bounds, promoting exploration of the loss landscape. For example, triangular schedules increase linearly to $\eta_{\max}$ and decay back to $\eta_{\min}$ within a cycle.

3.3 Warm-up Strategies

Warm-up ramps the learning rate from zero to the target value over T_w steps:

$$\eta_t = \begin{cases} \eta_{\text{target}} \frac{t}{T_w}, & 0 \le t \le T_w, \\ \text{Schedule}(t - T_w), & t > T_w. \end{cases} \tag{15}$$

Transformers often employ linear warm-up followed by inverse square root decay, $\eta_t \propto t^{-1/2}$. Warm-up prevents adaptive optimizers from overreacting to initial gradient noise when moment estimates are unreliable.

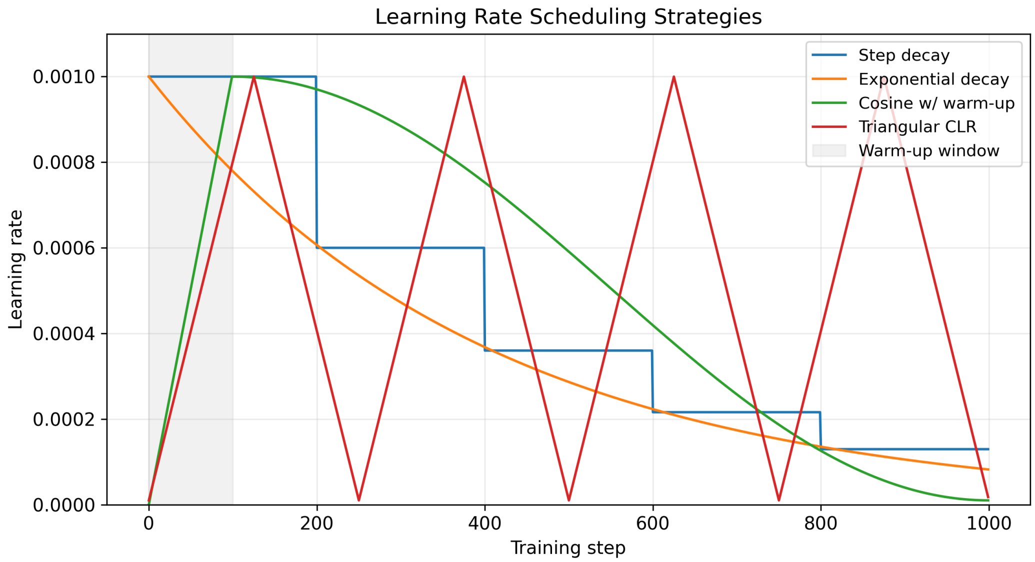

Figure 3: Comparison of step, cosine, cyclical, and warm-up learning rate policies. Warm-up smooths the start of training before the decay schedule begins.

4 Data Augmentation and Transfer Learning

Augmentation and transfer learning expand effective data coverage and reuse pretrained representations to accelerate convergence and boost performance on scarce datasets.

4.1 Classical and Advanced Augmentations

Spatial and photometric augmentations preserve labels while perturbing inputs:

- **Geometric:** random crops, flips, rotations, and elastic deformations.
- **Photometric:** color jitter, histogram equalization, Cutout.
- **Mix-based:** Mixup forms convex combinations of pairs, $\tilde{\mathbf{x}} = \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j$, with targets $\tilde{\mathbf{y}} = \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j$.
- **Distributional:** RandAugment samples augmentation chains; AugMix blends multiple randomized augmentations with Jensen-Shannon consistency loss.

4.2 Transfer Learning Workflow

Transfer learning initializes a target model with pretrained weights $\boldsymbol{\theta}_{\text{pre}}$ obtained on a source dataset $\mathcal{D}_{\text{src}}$. Fine-tuning adapts the model to the target dataset $\mathcal{D}_{\text{tgt}}$:

$$\boldsymbol{\theta}_0 = \boldsymbol{\theta}_{\text{pre}}, \tag{16}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} \big[\mathcal{L}_{\text{tgt}}(\boldsymbol{\theta}_t) + \lambda \mathcal{R}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{\text{pre}})\big], \tag{17}$$

where $\mathcal{R}$ regularizes deviation from pretrained features (e.g., L2-SP, Fisher regularization). Layer-wise adaptive rate scaling (LARS/LAMB) can assign smaller learning rates to lower layers to preserve useful representations.

4.3 Self-Supervised Pretraining

Contrastive and masked-prediction objectives learn transferable features without labels. SimCLR maximizes agreement between augmented views via the NT-Xent loss:

$$\mathcal{L}_{\mathrm{NT-Xent}} = -\sum_i \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_i'/\tau)}{\sum_j \mathbf{1}_{[j \neq i]} \exp(\mathbf{z}_i \cdot \mathbf{z}_j/\tau)}. \tag{18}$$

Fine-tuning leverages these representations on downstream tasks with minimal labeled data.

4.4 Practical Pipeline

Figure **??** summarizes a typical workflow that chains data augmentation, backbone pretraining, and task-specific heads.

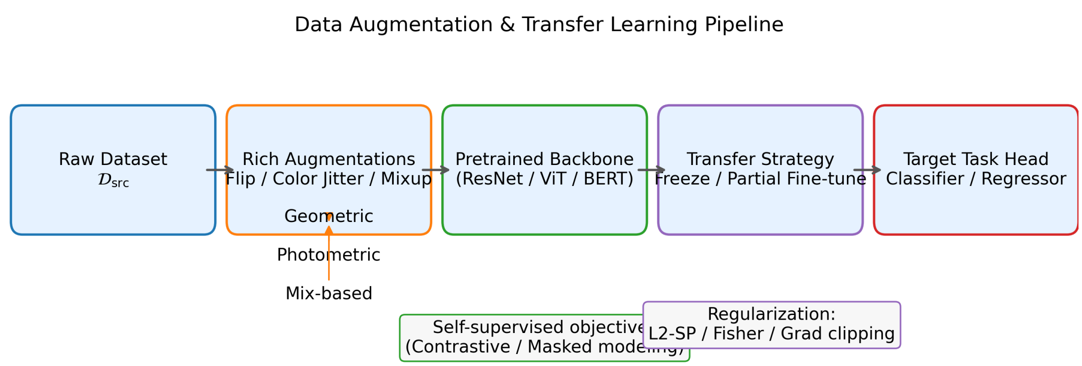

Figure 4: Pipeline combining rich data augmentation with transfer learning. Augmented samples feed a pretrained backbone, optionally frozen, before adaptation on the target task.

Further Reading

- Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization.” ICLR 2019.
- Sergey Ioffe and Christian Szegedy. “Batch Normalization.” ICML 2015.
- Leslie N. Smith. “Cyclical Learning Rates for Training Neural Networks.” WACV 2017.
- Tong He et al. “Bag of Tricks for Image Classification with Convolutional Neural Networks.” CVPR 2019.