Why Batch-Norm Parameters Should Not Have Weight Decay

1 Batch-Norm Recap

For a mini-batch $B = \{x_i\}_{i=1}^m$ and weights W of the preceding linear/conv layer,

$$\begin{aligned} z_i &= Wx_i, \\ \hat{z}_i &= \frac{z_i - \mu_B}{\sigma_B}, \\ y_i &= \gamma \hat{z}_i + \beta, \end{aligned}$$

with batch statistics μ_B, σ_B and two trainable scale/shift parameters γ, β.

2 Scale-Invariance Introduced by BN

Multiply all rows of W by any positive scalar c:

$$W' = cW \quad \Longrightarrow \quad \hat{z}_i' = \frac{cWx_i - \mu_B'}{\sigma_B'} = \hat{z}_i,$$

because $\mu_B' = c\mu_B, \sigma_B' = c\sigma_B$.

Hence the network's output does not depend on the norm of W; only the direction of the weights matters. Regularizers that penalize that norm therefore do not constrain the function implemented by the network.

3 What Weight Decay Does in This Setting

Weight decay (L2) adds

$$\mathscr{L}_{\text{WD}} = \frac{\lambda}{2} \left(\|W\|_2^2 + \gamma^2 + \beta^2 \right)$$

to the loss.

Because the data loss is invariant to $\|W\|$, the true optimum of the network lies on an entire ray $\{(cW, \gamma, \beta) \mid c > 0\}$.

Adding $\mathscr{L}_{\text{WD}}$ breaks this symmetry and drives the optimizer to the point where

$$\frac{\partial \mathscr{L}}{\partial W} = 0, \quad \frac{\partial \mathscr{L}}{\partial \gamma} + \lambda \gamma = 0, \quad \frac{\partial \mathscr{L}}{\partial \beta} + \lambda \beta = 0.$$

Because $\partial\mathscr{L}/\partial\gamma, \partial\mathscr{L}/\partial\beta$ are often small near convergence, the last two equations force $\gamma, \beta \to 0$.

That collapses the post-BN activations ($\mathrm{Var}(y) = \gamma^2$) and destroys representational power; the network merely rescales earlier weights to compensate, leaving performance unchanged but training dynamics worse.

4 Effective-Learning-Rate Distortion

With weight decay, each SGD/Adam update for a BN parameter is

$$\Delta\gamma = -\eta\left(\frac{\partial\mathscr{L}}{\partial\gamma} + \lambda\gamma\right).$$

When $\lambda\gamma$ dominates, this behaves like using a much smaller effective learning rate

$$\eta_{\text{eff}} = \eta\frac{\partial\mathscr{L}/\partial\gamma}{\partial\mathscr{L}/\partial\gamma + \lambda\gamma}.$$

The same distortion occurs for W because only its norm (not its direction) is penalized; the term λW competes with the gradient and harms convergence rather than regularizing the function.

5 Practical Takeaway

- Place all BN parameters (γ, β) and often even the weights that feed directly into BN into a separate optimizer group with $\lambda = 0$.
- Apply weight decay only to parameters whose scale does affect the network's output (e.g., convolution/linear weights that are not immediately normalized).

This preserves the intended regularization effect of weight decay while avoiding meaningless shrinkage of scale/shift terms and the associated optimization pathologies.

6 Practical Implementation in PyTorch

6.1 Method 1: Manual Parameter Grouping

```python
def create_param_groups(model, weight_decay=1e-4, lr=0.1):
    """
    Separates parameters into two groups:
    1. Parameters WITH weight decay (conv, linear weights)
    2. Parameters WITHOUT weight decay (BN params, biases)
    """
    decay_params = []
    no_decay_params = []

    for name, param in model.named_parameters():
        if not param.requires_grad:
            continue

        # Check if parameter is from batch norm or is a bias
        if 'bn' in name or 'batch_norm' in name or 'norm' in name:
            no_decay_params.append(param)
        elif 'bias' in name:
            no_decay_params.append(param)  # Often biases are also excluded
        else:
            decay_params.append(param)

    param_groups = [
        {'params': decay_params, 'weight_decay': weight_decay, 'lr': lr},
        {'params': no_decay_params, 'weight_decay': 0.0, 'lr': lr}
    ]

    return param_groups

# Usage
model = torchvision.models.resnet18()
param_groups = create_param_groups(model, weight_decay=1e-4)
optimizer = torch.optim.SGD(param_groups, momentum=0.9)
```

6.2 Method 2: Using Module Type Checking

```python
def create_param_groups_by_module(model, weight_decay=1e-4, lr=0.1):
    """
    More robust method using module type checking
    """
    bn_types = (nn.BatchNorm1d, nn.BatchNorm2d, nn.BatchNorm3d,
                nn.LayerNorm, nn.GroupNorm, nn.InstanceNorm1d,
                nn.InstanceNorm2d, nn.InstanceNorm3d)

    decay_params = []
    no_decay_params = []

    for module_name, module in model.named_modules():
        for param_name, param in module.named_parameters(recurse=False):
            if not param.requires_grad:
                continue

            full_param_name = f"{module_name}.{param_name}" if module_name else param_name

            if isinstance(module, bn_types):
                # All parameters in normalization layers
                no_decay_params.append(param)
            elif 'bias' in param_name:
                # Bias terms in conv/linear layers
                no_decay_params.append(param)
            else:
                # Weights in conv/linear layers
                decay_params.append(param)

    param_groups = [
        {'params': decay_params, 'weight_decay': weight_decay, 'lr': lr},
        {'params': no_decay_params, 'weight_decay': 0.0, 'lr': lr}
    ]

    print(f"Parameters with weight decay: {len(decay_params)}")
    print(f"Parameters without weight decay: {len(no_decay_params)}")

    return param_groups
```

6.3 Method 3: For SimCLR and LARS Optimizer

```python
def create_param_groups_lars(model, weight_decay=1e-6,
   optimizer_name='lars'):
    """
    Special handling for LARS optimizer (used in SimCLR)
    LARS also excludes biases from weight decay
    """
    def exclude_from_wd_and_adaptation(name):
        # Exclude batch norm parameters
        if 'bn' in name or 'norm' in name:
            return True
        # LARS also excludes all biases
        if optimizer_name == 'lars' and 'bias' in name:
            return True
        return False

    param_groups = [
        {
            'params': [p for name, p in model.named_parameters()
                      if not exclude_from_wd_and_adaptation(name)],
            'weight_decay': weight_decay,
            'layer_adaptation': True,  # LARS specific
        },
        {
            'params': [p for name, p in model.named_parameters()
                      if exclude_from_wd_and_adaptation(name)],
            'weight_decay': 0.0,
            'layer_adaptation': False,  # LARS specific
        },
    ]
    return param_groups
```

6.4 Method 4: Using AdamW (Decoupled Weight Decay)

```python
def create_param_groups_adamw(model, weight_decay=1e-2, lr=1e-3):
    """
    AdamW implements decoupled weight decay which is more stable
    but we still exclude BN parameters
    """
    no_decay = ['bias', 'bn', 'norm', 'BatchNorm', 'LayerNorm', 'GroupNorm']

    optimizer_grouped_parameters = [
        {
            'params': [p for n, p in model.named_parameters()
                       if not any(nd in n for nd in no_decay) and p.requires_grad],
            'weight_decay': weight_decay,
            'lr': lr,
        },
        {
            'params': [p for n, p in model.named_parameters()
                       if any(nd in n for nd in no_decay) and p.requires_grad],
            'weight_decay': 0.0,
            'lr': lr,
        }
    ]

    optimizer = torch.optim.AdamW(optimizer_grouped_parameters)
    return optimizer
```

7 Verification Code

```python
def verify_param_groups(model, optimizer):
    """
    Verify that BN parameters have zero weight decay
    """
    bn_modules = [m for m in model.modules()
                  if isinstance(m, (nn.BatchNorm1d, nn.BatchNorm2d, nn.BatchNorm3d))]

    for group_idx, param_group in enumerate(optimizer.param_groups):
        wd = param_group['weight_decay']
        print(f"\nGroup {group_idx}: weight_decay = {wd}")

        for param in param_group['params']:
            # Find which module this parameter belongs to
            for name, module_param in model.named_parameters():
                if module_param is param:
                    print(f"  - {name}: shape {param.shape}")

                    # Check if this is a BN parameter
                    if 'bn' in name or 'norm' in name:
                        assert wd == 0.0, f"BN parameter {name} has non-zero weight decay!"
                    break

# Example usage
model = torchvision.models.resnet50()
optimizer = create_param_groups_adamw(model)
verify_param_groups(model, optimizer)
```