Regularization Techniques

Parameter Norm Penalties & Dataset Augmentation

Deep Learning for Perception — BSCS, FAST-NUCES

1 Parameter Norm Penalties

Why It Matters

Neural networks have millions of parameters that can easily overfit. Parameter norm penalties add a "cost" to having large weights, forcing models to keep parameters small unless truly important. This is THE most common regularization technique in production deep learning.

1.1 The Core Concept

Analogy

Packing for a trip:
Without penalty: Pack everything "just in case" $\rightarrow$ huge heavy suitcase, most items unused.
With penalty ($10/kg): Pack only essentials $\rightarrow$ light efficient suitcase.
Neural networks: Same idea - make the model "pay" for large weights, so it only uses them when necessary!

1.2 Mathematical Foundation

Definition

Parameter Norm Penalty: Add a penalty term to the loss function that penalizes large parameter values.
General Form:

$$\tilde{L}(\theta) = L(\theta) + \alpha\Omega(\theta)$$

Where:

- $L(\theta)$ = Original loss (prediction error)
- $\Omega(\theta)$ = Regularization term (model complexity)
- α = Regularization strength (hyperparameter)
- $\tilde{L}(\theta)$ = Total loss (what we minimize)

Balance: Model must (1) make good predictions AND (2) keep weights small.

1.3 L2 Regularization (Ridge / Weight Decay)

L2 - Most Common

Idea: Penalize the SQUARED magnitude of weights.
Penalty:

$$\Omega(\theta) = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}\sum_i w_i^2$$

Total Loss:

$$\boxed{\tilde{L}(w) = L(w) + \frac{\alpha}{2}\sum_i w_i^2}$$

Properties:

- Penalizes large weights MORE (squared term)
- Drives weights toward zero (rarely exactly zero)
- Smooth, differentiable
- Also called "Ridge" or "Weight Decay"

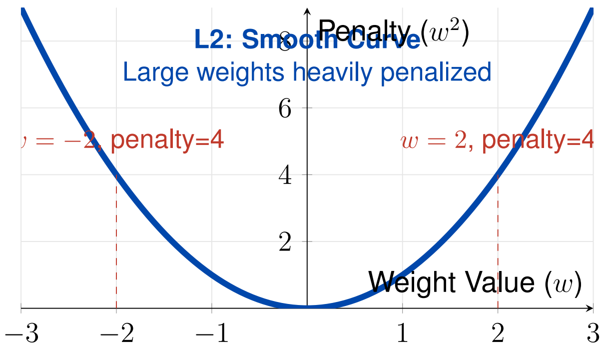

Figure 1: L2 penalty grows quadratically with weight magnitude

1.3.1 Weight Decay Interpretation

Key Formula

Gradient of L2-regularized loss:

$$\frac{\partial \tilde{L}}{\partial w} = \frac{\partial L}{\partial w} + \alpha w$$

Gradient descent update:

$$w \leftarrow w - \eta\left(\frac{\partial L}{\partial w} + \alpha w\right)$$

Rearranging:

$$\boxed{w \leftarrow w(1 - \eta\alpha) - \eta\frac{\partial L}{\partial w}}$$

Interpretation:

- $(1 - \eta\alpha)$ = Decay factor (typically 0.999)
- Each update SHRINKS weight by this factor
- Then applies gradient update
- Weight "decays" toward zero!

Typical values:

- $\alpha = 0.0001$ to 0.01
- $\eta = 0.001$ to 0.1
- Decay: $1 - \eta\alpha \approx 0.9999$ to 0.999

Example

Given: $w = 0.8$, $\eta = 0.01$, $\alpha = 0.5$, gradient $= 0.3$
Solution:
Step 1: Decay factor = $1 - (0.01)(0.5) = 0.995$
Step 2: Apply decay: $0.8 \times 0.995 = 0.796$
Step 3: Apply gradient: $0.796 - 0.01(0.3) = 0.793$
Result: Weight changed from $0.8 \to 0.793$

- Decay contribution: -0.004
- Gradient contribution: -0.003
- Total: -0.007

1.4 L1 Regularization (Lasso)

L1 - Sparse Solutions

Idea: Penalize ABSOLUTE VALUE of weights.
Penalty:

$$\Omega(\theta) = \|w\|_1 = \sum_i |w_i|$$

Total Loss:

$$\boxed{\tilde{L}(w) = L(w) + \alpha \sum_i |w_i|}$$

Properties:

- Pushes weights to EXACTLY zero (sparse)
- Non-smooth at zero (absolute value)
- Automatic feature selection
- Less common in deep learning

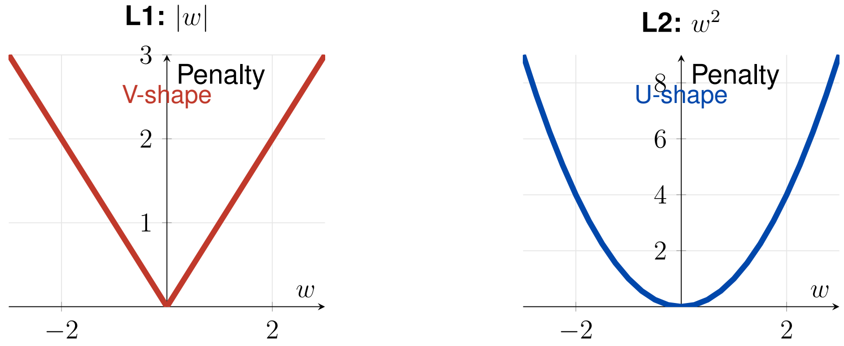

Figure 2: L1 V-shape pushes to zero; L2 U-shape keeps small

1.5 L1 vs L2 Comparison

Aspect	L1 (Lasso)	L2 (Ridge)
Penalty	$\alpha \sum \|w_i\|$	$\frac{\alpha}{2} \sum w_i^2$
Effect	Many $w_i = 0$ exactly	All w_i small
Sparsity	YES	NO
Feature Selection	Automatic	None
Best For	Many irrelevant features	All features contribute
Example	Text (10K words, 100 important)	Images (all pixels)
Popularity	Less common	MOST COMMON

Quick Reference

Quick Guide:
L2 (Default):

- Penalty: $\frac{\alpha}{2} \sum w_i^2$
- Update: $w \leftarrow w(1 - \eta\alpha) - \eta \nabla L$
- Use 90% of the time
- Typical α: 0.01 or 0.001

L1 (Special Cases):

- Penalty: $\alpha \sum |w_i|$
- Creates sparse models
- Use when want feature selection
- Typical α: 0.001

Implementation (PyTorch):

```
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.001,
    weight_decay=0.01  # L2 alpha
)
```

2 Dataset Augmentation

Why It Matters

MORE DATA = BETTER MODELS. But collecting real data is expensive. Dataset augmentation creates NEW training examples by transforming existing data in label-preserving ways. Can effectively 10x-100x your dataset FOR FREE! Used in every state-of-the-art vision model.

2.1 The Core Idea

Analogy

Learning to recognize cars:
Without augmentation: 100 photos, all front view, daylight $\rightarrow$ fails on sideways or night photos.
With augmentation: Same 100 photos rotated, brightness adjusted, flipped $\rightarrow$ sees cars from all angles/lighting $\rightarrow$ recognizes ANY car!
Magic: Created 1000 training examples from 100 originals!

Definition

Dataset Augmentation: Generate new training examples by applying label-preserving transformations.
Key Principles:

1. **Preserves label:** Rotated cat is still a cat
2. **Increases diversity:** Different views/conditions
3. **During training only:** Each epoch sees new versions
4. **No test augmentation:** Use originals for evaluation

Benefits:

- Reduces overfitting
- Improves generalization
- Effectively increases dataset size
- Robust to real-world variations
- FREE!

2.2 Image Augmentation Techniques

2.2.1 Geometric Transformations

Transform	Use When	Avoid When
Rotation ($\pm$5–45°)	Any angle OK	Text, street signs
Horizontal Flip	Symmetric objects	Digits (6 becomes 9!)
Vertical Flip	Satellite images	Natural images
Translation	Object anywhere	Centered only
Zoom (80–120%)	Varying distances	Fixed-size objects
Random Crop	Large images	Small images

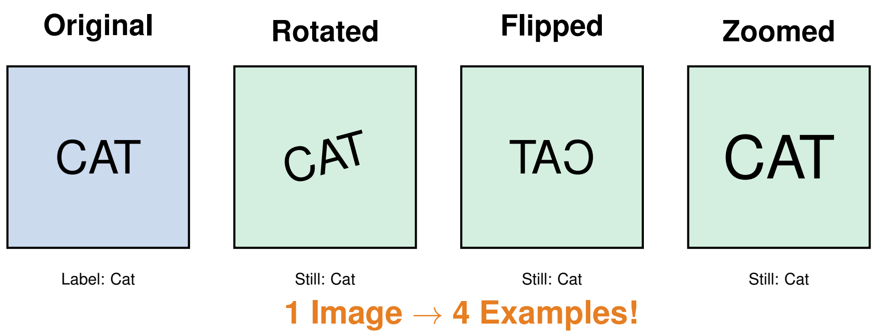

Figure 3: Geometric augmentations preserve labels

2.2.2 Photometric Transformations

Transform	Effect	Typical Range
Brightness	Lighter/darker	$\times$ [0.8, 1.2]
Contrast	More/less contrast	$\times$ [0.7, 1.3]
Saturation	Color intensity	$\times$ [0.5, 1.5]
Hue Shift	Color rotation	$[-10°, +10°]$
Gaussian Noise	Random pixels	$\sigma = 0.01$
Blur	Smoothing	kernel 3$\times$3

2.2.3 Advanced Techniques

Modern Augmentations

1. CutOut / Random Erasing

- Randomly mask rectangular regions
- Forces use of all image parts

2. MixUp

- Blend two images: $x_{new} = \lambda x_1 + (1 - \lambda)x_2$
- Label: $y_{new} = \lambda y_1 + (1 - \lambda)y_2$

3. CutMix

- Cut region from image 1, paste into image 2
- Label proportional to areas

4. AutoAugment

- Automatically learn best policy
- State-of-the-art results

2.3 Domain-Specific Guidelines

When to Use What

Medical Imaging:

- YES: Rotation, scaling, elastic deformation
- NO: Color changes (diagnostic!), flips (left $\neq$ right)

Text/Documents:

- YES: Small rotation ($\pm 2^{\circ}$), brightness, perspective
- NO: Horizontal flip (unreadable!), large rotation

Natural Images:

- YES: Rotation ($\pm 15^{\circ}$), h-flip, crop, color jitter
- NO: Vertical flip (usually)

2.4 Implementation

PyTorch Implementation

```
from torchvision import transforms

# Training augmentation
train_transform = transforms.Compose([
    transforms.RandomRotation(15),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2,
                           contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# NO augmentation for validation/test
val_transform = transforms.Compose([
    transforms.Resize(256),
```

```
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

2.5 Best Practices

Quick Reference

Golden Rules:
DO:

- Augment TRAINING only (not test!)
- Apply multiple together
- Start conservative, increase gradually
- Visualize samples (check they make sense)
- Combine with dropout and L2

DON'T:

- Augment test data (biases evaluation!)
- Change the label
- Over-augment (unrecognizable)
- Use same every epoch

Safety Check: "Would a human still label this correctly?"

- YES $\rightarrow$ Safe
- NO $\rightarrow$ Don't use

Augmentation Strength:

- Light: Rotation $\pm 5°$, brightness $\pm 10\%$
- Medium: Rotation $\pm 15°$, brightness $\pm 20\%$, flip
- Heavy: Rotation $\pm 30°$, crop, flip, color, cutout

Start with Medium, adjust if:

- Still overfitting $\rightarrow$ increase
- Training accuracy too low $\rightarrow$ decrease

Impact Example

Scenario: Cat vs Dog classifier, 1000 images

Setup	Train	Val	Gap
No augmentation	95%	72%	23%
Light (flip)	92%	78%	14%
Medium (flip+rotate+crop)	88%	83%	5%
Heavy (all)	85%	84%	1%

Best: Medium augmentation

- +11% validation (72 $\rightarrow$ 83%)
- 78% gap reduction (23 $\rightarrow$ 5%)
- Equivalent to 5-10x more data!

3 Summary

Key Takeaways

Parameter Norm Penalties

L2 (Weight Decay) - DEFAULT:

- Penalty: $\frac{\alpha}{2}\sum w_i^2$
- Update: $w \leftarrow w(1-\eta\alpha) - \eta\nabla L$
- All weights $\rightarrow$ small
- Use 90% of the time
- Typical α: 0.01 or 0.001

L1 (Lasso) - SPARSE:

- Penalty: $\alpha\sum|w_i|$
- Many weights $\rightarrow$ exactly 0
- Feature selection
- Typical α: 0.001

Dataset Augmentation

Common Techniques:

- Geometric: Rotation, flip, crop, zoom
- Photometric: Brightness, contrast, color
- Advanced: CutOut, MixUp, CutMix

Rules:

- Training only (not test!)
- Preserve labels
- Human should still recognize
- Combine with L2 + dropout

Impact:

- 5-15% accuracy improvement
- Dramatically reduces overfitting
- Equivalent to 5-10x more data
- FREE!

Standard Recipe

```
Data Augmentation (flip+rotate+crop+color)
+ L2 Regularization (alpha=0.01)
+ Dropout (p=0.5)
+ Early Stopping
= Robust Model
```

End of Notes

Deep Learning for Perception — FAST-NUCES