SCALER School of Technology × Priyansh Saxena

Regularization for Deep Networks

Overfitting • Weight Decay • Dropout • Early Stopping • Data Augmentation

Priyansh Saxena

Scaler School of Technology | Deep Learning I: Neural Networks | Week 8 — Session 1/2

14 May 2026

Recap & Today's Question

Week 2: The *How* (Training)

- Backpropagation computes gradients via chain rule
- Loss functions measure prediction error
- Optimizers (SGD, Adam) update weights

Week 3: The *Art* (Generalization)

- How do we stop a network from **memorizing**?
- How do we make it **generalize**?
- What tools prevent **overfitting**?

Today: regularization — the toolkit that turns memorization into generalization.

The Problem

Model A: Overfitting

- Train loss $\rightarrow$ 0
- Val loss $\rightarrow$ diverges upward
- Memorizes training data

Model B: Well-Regularized

- Train loss decreases steadily
- Val loss tracks train loss
- Generalizes to new data

Same architecture, same data. Only difference: regularization.

Today we learn every trick to get from A to B.

Why Deep Nets Overfit

Parameters vs. Data:

Model	Params	Dataset
ResNet-18	11M	CIFAR-10 (50K)
ResNet-50	25M	ImageNet (1.2M)
GPT-2	1.5B	WebText (40GB)

ResNet-18 on CIFAR-10:
$\frac{\text{11M params}}{\text{50K samples}} = \mathbf{220\ params\ per\ sample}$

Key Insight

When parameters $\gg$ data, the network has enough capacity to **memorize** every training example perfectly.

Overfitting is the *default behavior* of deep networks.

Memorization is Easy

Zhang et al., 2017

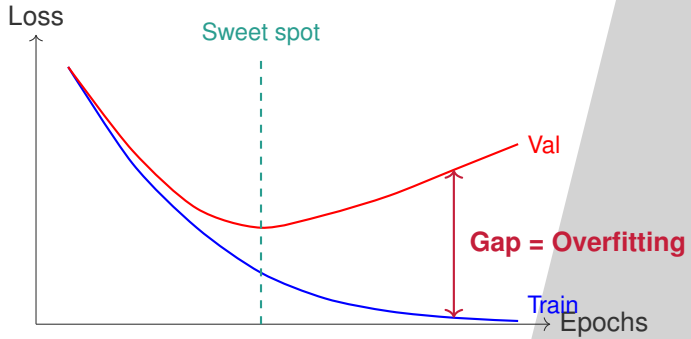
Randomly shuffled ALL labels in CIFAR-10 (cat → car, dog → plane, etc.)

A standard deep network still achieved **zero training loss**.

- The network memorized 50,000 *random* label assignments
- No patterns to learn — pure memorization
- **Implication:** if it can memorize noise, it can easily memorize real data + noise in your dataset

*Deep networks have the capacity to fit anything. Regularization constrains them to learn **patterns** instead.*

Overfitting Visually



The gap between train and val loss is your key diagnostic.

Quiz 1

1. Can a network with 1M parameters memorize 50K *random* labels?

2. What is the **earliest sign** of overfitting you can observe during training?

Quiz 1

1. Can a network with 1M parameters memorize 50K *random* labels?

2. What is the **earliest sign** of overfitting you can observe during training?

Answers

1. **Yes!** Zhang et al. proved it — deep nets can memorize random noise.
2. Validation loss **stops decreasing** while training loss continues to decrease.

The Idea: Penalize Large Weights

Regularized Loss

$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}_{\text{original}}}_{\text{fit the data}} + \underbrace{\lambda \sum_i w_i^2}_{\text{keep weights small}}$$

- λ controls penalty strength (typically 10^{-4} to 10^{-2})
- Small λ = mild regularization
- Large λ = aggressive (risk underfitting)
- **Same idea as Ridge Regression** from your ML course!

Intuition: A Budget Constraint

Without L2

- A few weights become very large
- Network “bets heavily” on specific features
- Prone to memorization

With L2

- All weights stay moderate
- Network uses many features
- More robust, harder to memorize

Analogy: Diversified portfolio vs. betting everything on one stock.

Weight Decay in SGD

Standard SGD update:

$$w \leftarrow w - \eta \cdot \nabla L$$

With weight decay:

$$w \leftarrow \underbrace{(1 - \eta\lambda)}_{\text{decay factor}} \cdot w - \eta \cdot \nabla L$$

Key Observation

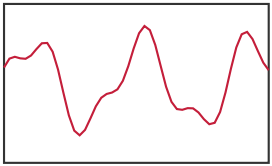
Every step, weights are multiplied by a number slightly less than 1.

Weights literally **decay** toward zero over time $\rightarrow$ "weight decay"

In PyTorch: `optim.SGD(params, lr=0.01, weight_decay=0.01)`

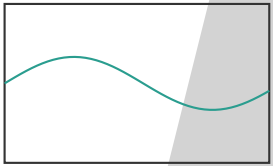
Effect on Decision Boundaries

No Regularization



Complex, wiggly

With L2 Regularization



Smooth, generalizes

L2 penalizes the complexity of the boundary $\rightarrow$ smoother decisions.

Gotcha: Weight Decay $\neq$ L2 in Adam

Important Practical Distinction

In **SGD**: weight decay and L2 penalty are mathematically equivalent.

In **Adam**: they are *not* the same (adaptive scaling breaks equivalence).

Solution: Use AdamW

AdamW = Adam with *decoupled* weight decay.

```
optim.AdamW(params, lr=0.001, weight_decay=0.01)
```

One letter difference. Always use Adam**W**.

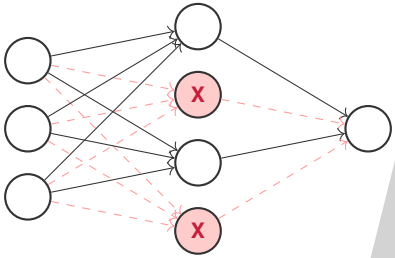
Coming Up Next

Part 2: Dropout — The Most Famous Deep Learning Regularizer

Dropout: The Concept

Definition

During each **training** forward pass, randomly set each neuron's output to zero with probability *p*.



Different neurons dropped each pass

The Team Project Analogy

Analogy

Imagine a group project where **every meeting**, 2 random team members don't show up.

- If anyone might be absent $\rightarrow$ **everyone** must understand the material
- Nobody can be the single point of failure
- Nobody can "slack off" while one person does all the work
- Result: a more **robust team** with distributed knowledge

In Neural Net Terms

Dropout prevents **co-adaptation**: neurons cannot rely on specific other neurons being present. Features must be independently useful.

Training vs. Inference (Inverted Dropout)

Training Mode

1. Drop neurons with probability p
2. Scale survivors by $\frac{1}{1-p}$
3. Different mask every forward pass

Inference Mode

1. Use **all** neurons
2. No scaling needed
3. Deterministic output

Why scale by $\frac{1}{1-p}$?

If $p = 0.5$: half the neurons are zeroed $\rightarrow$ output magnitude halves.
Multiply survivors by 2 to compensate $\rightarrow$ expected output stays the same.

Common Bug

Forgetting `model.eval()` at test time $\rightarrow$ dropout still active $\rightarrow$ noisy predictions!

Dropout as an Ensemble

Key Insight

Each forward pass with dropout = a **different sub-network**.

- With n neurons and $p = 0.5$: there are 2^n possible sub-networks
- Training with dropout $\approx$ training an **exponential number** of models simultaneously
- Inference (all neurons active) $\approx$ **averaging** predictions of all sub-networks

Connection to Classical ML

Remember Random Forests? Same ensemble principle!

But here we get the ensemble **for free** inside a single network.

Dropout: Practical Tips

Setting	Recommendation
Hidden layers	$p = 0.5$ (good default)
Input layer	$p = 0.2$ (don't drop too much input)
After conv layers	$p = 0.25$ or skip entirely
With BatchNorm	Usually don't combine

Modern notes:

- ResNets: dropout used sparingly (BatchNorm handles regularization)
- Transformers: dropout on attention weights + residual connections
- Higher p = more regularization (but too high $\rightarrow$ underfitting)

Quiz 2

1. At **test time**, do you drop neurons?

2. Why does dropout help prevent overfitting? (Give 2 reasons)

Quiz 2

1. At **test time**, do you drop neurons?

2. Why does dropout help prevent overfitting? (Give 2 reasons)

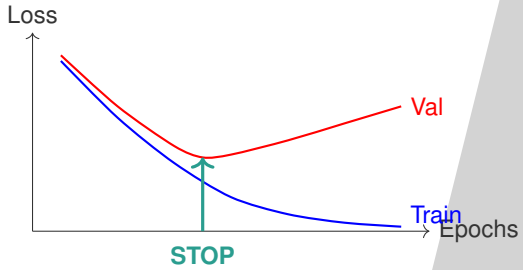
Answers

1. **No!** All neurons are active at test time. Dropout is only during training.
2. (a) Forces **distributed representations** — no co-adaptation
   (b) Acts as an **ensemble** of exponentially many sub-networks

Early Stopping

The Idea

Monitor validation loss every epoch. When it hasn't improved for **patience** epochs → stop training. Restore the best checkpoint.



Patience = 5–30 epochs typically. Almost free — no extra hyperparameters!

Early Stopping = Implicit Regularization

- Weights start **small** (from initialization)
- Training makes them **larger** over time
- Early stopping $=$ limiting how large weights can grow
- Mathematically similar effect to L2 regularization!

Intuition

Think of it as limiting the **distance traveled** in parameter space.

The longer you train, the more capacity the model uses $\rightarrow$ more likely to overfit.

Stop early $=$ use less capacity $=$ simpler model.

Data Augmentation: Free Data

The Idea

Apply **label-preserving** transformations to training data → larger effective dataset.

- A horizontally flipped cat is still a cat
- A slightly rotated digit is still the same digit
- A cropped photo of a dog is still a dog

One image → many variants (flip, rotate, crop, color jitter, blur, cutout)

Rule

This should be the **first** thing you try when overfitting — before dropout, before L2.
It's free and almost always helps.

Common Augmentations

Images:

- Horizontal flip
- Random crop / resize
- Rotation (± 15)
- Color jitter
- Cutout / Random erasing
- Mixup / CutMix

Text:

- Synonym replacement
- Back-translation
- Random deletion

Audio:

- Time stretching
- Pitch shift
- Noise injection

Constraint

Augmentations must be **label-preserving**!
Don't flip a "6" vertically (becomes "9").

Discussion

Your model is overfitting.

You've already tried `dropout=0.5`.

What do you try next?

Discussion

Your model is overfitting.

You've already tried `dropout=0.5`.

What do you try next?

Some Options

- More data augmentation
- Increase weight decay
- Reduce model size
- Get more data
- Lower learning rate
- Ensemble multiple models

Quick BatchNorm Recap

What BatchNorm Does

For each mini-batch, normalize activations to zero mean and unit variance, then apply learned scale (γ) and shift (β):

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \rightarrow \quad y_i = \gamma \hat{x}_i + \beta$$

Primary purpose: Stabilize training, allow higher learning rates.

But... it also has a regularization side-effect!

(We'll cover BatchNorm in full detail next session.)

Why BatchNorm Regularizes

The Noise Mechanism

- μ_B and σ_B^2 are computed from a **mini-batch** (e.g., 32 samples)
- These are *noisy estimates* of the true mean/variance
- Each sample gets slightly different normalization depending on its batch neighbors

- **Smaller batch** = noisier statistics = more regularization
- At test time: use **running averages** (deterministic) — just like dropout!
- This is why models with BatchNorm often need *less* dropout

Key Takeaway

BatchNorm introduces randomness during training (from mini-batch noise) $\rightarrow$ acts as a mild regularizer.

The Regularization Toolkit

Technique	How It Works	When to Use
L2 / Weight Decay	Penalize large weights	Always (1e-4 to 1e-2)
Dropout	Randomly zero neurons	Large FC layers (p=0.3–0.5)
Early Stopping	Stop when val loss rises	Always (safety net)
Data Augmentation	Label-preserving transforms	Always for images
BatchNorm	Mini-batch noise	When training is unstable

These work together — combine them!

Decision Framework: Which Regularizer When?

Always apply first:

Step 1 Data Augmentation

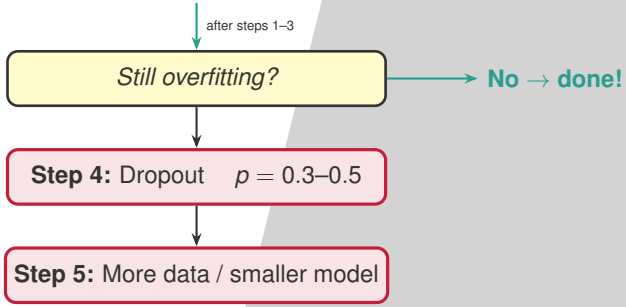
Flip, crop, rotate, colour jitter.
Free diversity from existing data.

Step 2 Weight Decay

AdamW, $\lambda = 10^{-4}$.
Penalises large weights cheaply.

Step 3 Early Stopping

Patience = 10–30 epochs.
Halt before overfitting deepens.



Rule of thumb: Try steps 1–3 before reaching for dropout.
Dropout adds training complexity.

Final Quiz

1. What's the difference between weight decay and L2 regularization in Adam?
2. Train accuracy = 99%, val accuracy = 75%. Name 3 things to try.
3. Why do we scale by $\frac{1}{1-p}$ during training with dropout?

Final Quiz

1. What's the difference between weight decay and L2 regularization in Adam?
2. Train accuracy = 99%, val accuracy = 75%. Name 3 things to try.
3. Why do we scale by $\frac{1}{1-p}$ during training with dropout?

Answers

1. They're **not equivalent** in Adam! Use **AdamW** for proper decoupled weight decay.
2. Data augmentation, dropout, weight decay, early stopping, more data, smaller model.
3. To keep expected output magnitude the same, so inference doesn't need scaling.

Next session: Initialization, Normalization, and Debugging

Thank You!

Questions?

Priyansh Saxena

All lecture materials available at:

www.priyanshsaxena.com/sst-deep-learning-nn