Regularization in Multi-Layer Perceptrons

Vaibhav Chourasia

December 16, 2025

1 Introduction

Multi-Layer Perceptrons (MLPs) are powerful function approximators capable of learning highly complex patterns. However, this flexibility makes them prone to overfitting—a situation where the model memorizes noise in the training data and performs poorly on unseen samples.

Regularization refers to a collection of techniques that improve the generalization ability of neural networks. These techniques modify the training process in ways that discourage overly complex models.

2 What is Regularization?

Before exploring specific techniques, it is important to understand why regularization is needed.

Overfitting vs Underfitting

Underfitting: The model is too simple and fails to capture true patterns.

Overfitting: The model is too complex and fits training noise.

A simple mathematical way to understand this is through the bias–variance decomposition. For a prediction $\hat{y}$ of the true value y:

$$\mathbb{E}[(y-\hat{y})^2] = \text{Bias}^2 + \text{Variance} + \sigma^2$$

- High Bias $\rightarrow$ Underfitting
- High Variance $\rightarrow$ Overfitting

Regularization techniques primarily aim to reduce variance while keeping bias manageable.

3 L1 Regularization

3.1 Intuition

L1 regularization encourages the network to use only a small subset of weights by pushing many weights to exactly zero. This creates a "sparse" model, similar to feature selection.

3.2 Mathematical Form

For a model with loss $L(\theta)$, L1 adds a penalty proportional to the sum of absolute values of weights:

$$L_{\text{total}} = L(\theta) + \lambda \sum_i |\theta_i|$$

Here, λ controls the strength of regularization.

3.3 Gradient Derivation

Consider the derivative of $|\theta|$:

$$\frac{d}{d\theta}|\theta| = \begin{cases} +1, & \theta > 0 \\ -1, & \theta < 0 \\ \text{undefined}, & \theta = 0 \end{cases}$$

So the gradient update becomes:

$$\theta \leftarrow \theta - \alpha \left(\frac{\partial L}{\partial \theta} + \lambda \cdot \text{sign}(\theta) \right)$$

This "soft pushes" weights toward 0. If a weight becomes exactly 0, it stays there—producing sparsity.

3.4 Effect on Learning

- Encourages sparse models
- Can improve interpretability

- Good when many input features are irrelevant

4 L2 Regularization

4.1 Intuition

L2 regularization discourages large weights. Instead of eliminating weights, it shrinks them smoothly.

4.2 Mathematical Form

$$L_{\text{total}} = L(\theta) + \lambda \sum_i \theta_i^2$$

4.3 Derivative and Update Rule

Derivative of θ^2 is:

$$\frac{d}{d\theta}(\theta^2) = 2\theta$$

Thus:

$$\theta \leftarrow \theta - \alpha \left(\frac{\partial L}{\partial \theta} + 2\lambda\theta \right)$$

This causes weights to decay toward zero over time.

4.4 Why L2 Helps

- Prevents any weight from dominating
- Smooths the function space
- Reduces variance

4.5 Bayesian Interpretation

L2 regularization is equivalent to assuming weights come from a Gaussian distribution:

$$\theta_i \sim \mathcal{N}(0, \sigma^2)$$

This means "smaller weights are more likely," which encourages smoother models.

5 Weight Decay vs L2 Regularization

The terms "L2 regularization" and "weight decay" are often used interchangeably. However, they are not the same when using adaptive optimizers like Adam.

5.1 Weight Decay

Weight decay directly multiplies weights by a shrinkage factor:

$$\theta \leftarrow (1 - \alpha\lambda)\theta - \alpha\frac{\partial L}{\partial \theta}$$

5.2 Why Adam + L2 is Not True Weight Decay

Adam scales gradients using running averages:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

If L2 is added to the loss:

$$g_t = \frac{\partial L}{\partial \theta} + 2\lambda\theta$$

The L2 term is also scaled by Adam's adaptive denominator:

$$\frac{m_t}{\sqrt{v_t}}$$

Thus L2 interferes with Adam's gradient scaling.

5.3 AdamW Fixes This

AdamW applies weight decay separately:

$$\theta \leftarrow \theta - \alpha\lambda\theta$$

$$\theta \leftarrow \theta - \alpha \frac{m_t}{\sqrt{v_t}}$$

This clean separation makes AdamW behave correctly.

5.4 Summary

- L2 Regularization = adds penalty to loss
- Weight Decay = shrinks weights during update
- They are the same only for SGD
- They differ for Adam → use AdamW

6 Dropout

6.1 Intuition

Dropout is one of the most influential and effective regularization techniques in deep learning.

During training, a neuron is randomly removed (dropped) with probability p. This forces the network to:

- not depend too heavily on any single neuron,
- learn redundant representations,
- behave like an ensemble of many subnetworks.

This significantly reduces overfitting.

6.2 Mathematical Formulation

Let $\mathbf{h}$ be the activation vector of a layer. Define binary mask $\mathbf{m}$ where each element is:

$$m_i \sim \text{Bernoulli}(1 - p)$$

Then the dropped-out activations are:

$$\tilde{\mathbf{h}} = \mathbf{h} \odot \mathbf{m}$$

To maintain the same expected activation magnitude at test time, we scale by $(1-p)$ during training:

$$\tilde{\mathbf{h}} = \frac{\mathbf{h} \odot \mathbf{m}}{1-p}$$

6.3 Interpretation as Model Averaging

Dropout implicitly trains 2^n possible thinned networks (where n = number of neurons). At inference time, using the full network approximates the average prediction of these subnetworks — improving generalization.

6.4 Effect on Weights

Dropout adds noise to neuron outputs:

$$\tilde{\mathbf{h}} = \mathbf{h} + \epsilon$$

where ϵ is multiplicative noise. This is mathematically similar to L2 regularization:

$$\text{Dropout} \approx \text{adaptive weight decay}$$

7 Batch Normalization

Batch Normalization (BatchNorm) is both an optimization stabilizer and a regularizer. It transforms layer inputs to have:

$$\text{mean} = 0, \quad \text{variance} = 1$$

7.1 The Problem: Internal Covariate Shift

As training progresses, the distribution of activations in each layer keeps changing. This slows down learning.

BatchNorm stabilizes these distributions.

7.2 Mathematical Procedure

Given mini-batch $\{z_1, z_2, ..., z_m\}$:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} z_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_B)^2$$

Normalize:

$$\hat{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Then re-scale using learned parameters γ and β:

$$y_i = \gamma \hat{z}_i + \beta$$

7.3 Why it Helps

- stabilizes gradient flow
- reduces vanishing/exploding gradients
- allows higher learning rates
- provides regularization (acts like dropout-light)

7.4 BN as a Regularizer (Mathematically)

BN introduces noise during training because batch statistics vary across batches:

$$\hat{z}_i = \frac{z_i - \mu_B}{\sigma_B}$$

But μ_B and σ_B depend on the sampled mini-batch $\rightarrow$ they are random variables.

Thus:

$$\hat{z}_i = \frac{z_i - (\mu + \delta_\mu)}{\sigma + \delta_\sigma}$$

This stochasticity creates implicit regularization similar to dropout.

8 Layer Normalization

BatchNorm normalizes across the batch dimension. LayerNorm normalizes across the features of a single sample.

8.1 Mathematical Definition

For a single training example with activations:

$$\mathbf{z} = (z_1, z_2, ..., z_H)$$

Compute mean and variance across features:

$$\mu_L = \frac{1}{H}\sum_{j=1}^{H} z_j$$

$$\sigma_L^2 = \frac{1}{H}\sum_{j=1}^{H}(z_j - \mu_L)^2$$

Normalize:

$$\hat{z}_j = \frac{z_j - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}}$$

Apply scale and shift:

$$y_j = \gamma \hat{z}_j + \beta$$

8.2 BN vs LN

Property	BatchNorm	LayerNorm
Normalizes Across	batch dimension	feature dimension
Batch-size dependent?	Yes	No
Works best for	CNNs	RNNs / Transformers
Regularization strength	Moderate	Lower

8.3 Why LN Helps

- stable for very small batch sizes

- improves training for sequential models
- makes training less sensitive to batch composition

9 Early Stopping

9.1 Motivation

During training, the network typically begins by learning general patterns. After enough epochs, it starts learning noise from the training set → overfitting.

Early stopping prevents this by halting training when validation performance stops improving.

9.2 Mathematical Interpretation

Let:

- $L_{\text{train}}(t)$ be the training loss at epoch t
- $L_{\text{val}}(t)$ be the validation loss at epoch t

Empirically:

$$L_{\text{train}}(t) \downarrow \quad \text{always decreases}$$
$$L_{\text{val}}(t) \downarrow \text{ initially, then } \uparrow \text{ when overfitting begins}$$

Define t^* as:

$$t^* = \arg\min_t L_{\text{val}}(t)$$

Training is stopped at or near t^*.

9.3 Early Stopping as Regularization

Early stopping behaves similarly to L2 regularization.

To understand why, consider a simplified quadratic loss:

$$L(\theta) = \frac{1}{2}a\theta^2 - b\theta$$

Gradient descent update:

$$\theta_{t+1} = \theta_t - \alpha(a\theta_t - b)$$

Solving recurrence:

$$\theta_t = (1 - \alpha a)^t \theta_0 + \frac{b}{a}\left(1 - (1 - \alpha a)^t\right)$$

As $t \to \infty$:

$$\theta_\infty = \frac{b}{a}$$

But early stopping uses θ_t for some finite t. This yields:

$$|\theta_t| < |\theta_\infty|$$

This is equivalent to adding weight decay. Therefore:

Early stopping acts like a constraint preventing large weights.

9.4 Practical Implementation

- Monitor validation loss.
- Maintain a patience counter.
- If no improvement for p epochs $\to$ stop.

This technique is simple yet extremely effective.

10 Overfitting and Underfitting

10.1 Definitions

Overfitting: The model learns noise or irrelevant details from the training data. Training accuracy is high, but test accuracy is low.

Underfitting: The model is too simple and fails to capture underlying patterns. Both training and test accuracy are low.

10.2 The Bias–Variance Tradeoff

Generalization error can be decomposed into:

$$\mathbb{E}[(y-\hat{f}(x))^2] = \underbrace{\text{Bias}^2}_{\text{error from wrong assumptions}} + \underbrace{\text{Variance}}_{\text{sensitivity to dataset}} + \underbrace{\sigma^2}_{\text{irreducible noise}}$$

10.3 Derivation of Bias–Variance

Let the true relationship be:

$$y = f(x) + \epsilon, \quad \mathbb{E}[\epsilon] = 0, \quad \text{Var}(\epsilon) = \sigma^2$$

Consider a model trained on dataset D, producing prediction $\hat{f}(x|D)$.

Expected squared error:

$$\mathbb{E}_D\left[(y-\hat{f}(x|D))^2\right]$$

Expand:

$$= \mathbb{E}_D[(f(x)+\epsilon-\hat{f}(x|D))^2]$$

Separate noise:

$$= \mathbb{E}_D[(f(x)-\hat{f}(x|D))^2] + \sigma^2$$

Now add and subtract $\mathbb{E}_D[\hat{f}(x|D)]$:

$$= \mathbb{E}_D\left[(f(x)-\mathbb{E}[\hat{f}]+\mathbb{E}[\hat{f}]-\hat{f})^2\right] + \sigma^2$$

Expanding and simplifying yields:

$$\text{Bias}^2 + \text{Variance} + \sigma^2$$

Thus:

Low bias, high variance $\rightarrow$ overfitting

High bias, low variance $\rightarrow$ underfitting

10.4 Detecting Overfitting vs Underfitting

- Training loss $\downarrow$, Validation loss $\uparrow$ $\rightarrow$ Overfitting
- Training loss high $\rightarrow$ Underfitting

10.5 How Regularization Helps

- L1 reduces variance by inducing sparsity.
- L2 reduces variance by shrinking weights.
- Dropout reduces co-adaptation.
- BN stabilizes gradients and acts as mild regularizer.
- Early stopping reduces effective model capacity.

11 Conclusion

Regularization is essential for training neural networks that generalize well to unseen data. In this report, we began with the fundamental motivation for regularization — the prevention of overfitting — and proceeded through every major modern regularization method used in MLPs.

Overall, regularization is not a single technique but a family of methods that collectively enable neural networks to avoid memorization and learn meaningful, generalizable patterns.