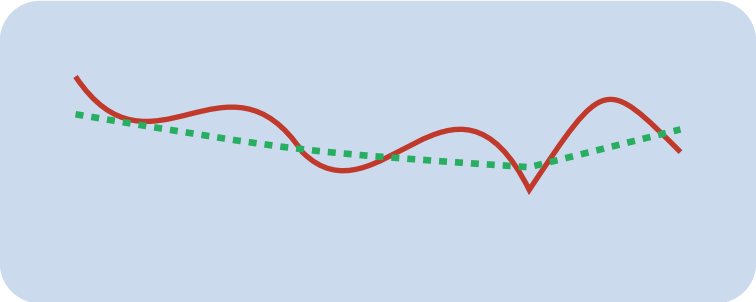
Deep Learning for Perception[0.5cm] Lecture 04: Generalization, Regularization Optimization[1cm]



Learning to Generalize (Not Memorize)

Topics Covered in This Lecture:

- Generalization Gap
- Overfitting Underfitting
- Regularization (L1, L2, Dropout)
- Early Stopping
- Data Augmentation
- Batch Normalization
- K-Fold Cross Validation
- Optimization: Adagrad, RMSprop, Adam

Instructor: Aqsa Younas[0.2cm] Department of Computer Science FAST-NU, CFD Campus

Contents

Advance Organizer — What You'll Learn

Learning Objectives: By the end of this lecture, you will be able to:

1. **Explain** what generalization is and why test performance matters
2. **Recognize** underfitting vs overfitting using errors/curves
3. **Use** regularization methods (L1, L2, Dropout) to reduce overfitting
4. **Apply** early stopping using a validation set
5. **Understand** why data augmentation helps when data is limited
6. **Explain** batch normalization in simple steps
7. **Run** k-fold cross validation for fair evaluation
8. **Compare** Adagrad, RMSprop, Adam at a high level

Prior Knowledge Required:

- Basic neural network training idea (forward + backward)
- Loss (error) and gradient descent concept
- Very basic derivatives (only idea: slope tells direction)

Quick Symbols Cheat Sheet

- w = weight (a number the model learns)
- η = learning rate (step size)
- λ = regularization strength (how strong the penalty is)
- ∇w or $\frac{\partial L}{\partial w}$ = gradient (direction to change w to reduce loss)
- p = dropout probability (chance to drop a neuron)
- m = dropout mask (0 = dropped, 1 = kept)

1 Generalization Gap

Why It Matters

In real life, your model will see **new data**. So the goal is not to get perfect training accuracy. The goal is to perform well on **unseen examples**.

1.1 What is Generalization?

Definition (Beginner Friendly)

Generalization means: after training, the model can still do well on **new data it never saw before**.

Training error = error on training data (data used for learning). **Test error** = error on test data (new unseen data).

Generalization Gap (common definition): [Gap = Test error - Training error]
How to read it:

- Small gap $\Rightarrow$ similar performance on train and test (good).
- Large gap $\Rightarrow$ model does well on train but worse on test (overfitting).

Analogy — Think of It Like This

Exam analogy:

- **Memorizing answers** = great on practice questions, weak on new questions.
- **Understanding ideas** = good on practice and also on new questions.

2 Overfitting and Underfitting

Why It Matters

Most training problems come from one of these two cases. If you can identify which one is happening, you can pick the right fix.

2.1 Underfitting

Definition (Beginner Friendly)

Underfitting means the model is **too simple**, so it cannot learn the real pattern in the data.

Easy sign: it performs badly **even on the training data**.
Symptoms:

- Training error is high
- Test error is also high

Common fixes: make the model stronger (more neurons/layers), train longer, or use better features.

2.2 Overfitting

Definition (Beginner Friendly)

Overfitting means the model learns the training data **too well**, including **noise and small details** that do not repeat in real life.

Easy sign: it performs very well on training data but worse on test data.
Symptoms:

- Training error becomes very low
- Test error stays high (or starts going up later)

Common fixes: regularization, dropout, early stopping, more data, data augmentation.

Figure 1: Visual comparison: Underfitting, Good Fit, and Overfitting

2.3 Training vs Test Error Curves

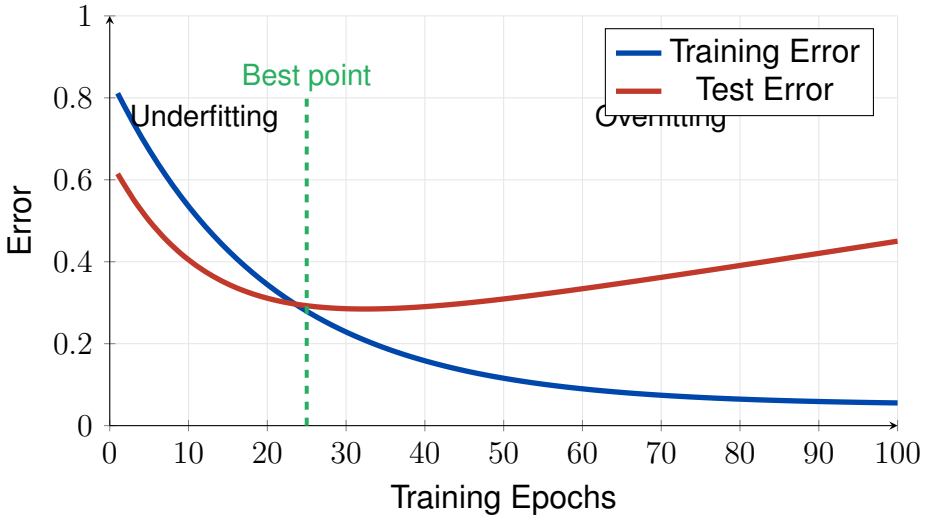


Figure 2: Training vs Test Error: test error can increase when overfitting starts

2.4 How to Avoid Overfitting and Underfitting

Problem	Simple reason	Common solutions
Underfitting		
Model too weak to learn pattern	Make model bigger, train longer, better features	Model memorizes noise/details
Overfitting		
Regularization, dropout, early stopping, more data height		

Table 1: Fitting problems and practical fixes

3 Regularization Techniques

Why It Matters

Regularization is a set of tricks to reduce overfitting. The main goal is to stop the model from becoming **too complex or too sensitive**.

3.1 L2 Regularization (Ridge / Weight Decay)

Definition (Beginner Friendly)

L2 Regularization (Weight Decay) adds an extra "punishment" if weights become too large.

Why it helps: large weights can make the model very sensitive and more likely to overfit. L2 gently pushes weights to stay **smaller**.

Key Formula + Meaning

L2 Regularized Loss: [$\boxed{L_{\text{total}} = L_{\text{original}} + \frac{\lambda}{2}|\mathbf{w}| * 2^2 = L * \text{original} + \frac{\lambda}{2}\sum_i w_i^2}$]

Weight Update with Decay: [$\boxed{w_i \leftarrow w_i - \eta\frac{\partial L}{\partial w_i} - \eta\lambda w_i = w_i(1-\eta\lambda) - \eta\frac{\partial L}{\partial w_i}}$]

Meaning of symbols (simple):

- λ controls **how strong the penalty is** (bigger λ = more shrinking)
- η is learning rate (step size)
- $(1-\eta\lambda)$ makes w a bit smaller each step

Solved Example 1: L2 Weight Decay

Given Data:

- Current weight: $w = 2.0$
- Learning rate: $\eta = 0.1$
- Regularization: $\lambda = 0.1$
- Gradient from loss: $\frac{\partial L}{\partial w} = 0.5$

Solution:
Step 1: Decay factor [1 - $\eta\lambda = 1 - (0.1)(0.1) = 0.99$]
Step 2: Shrink the weight a little [2.0 $\times 0.99 = 1.98$]
Step 3: Do the normal gradient step [1.98 - 0.1 $\times 0.5 = 1.98 - 0.05 = \mathbf{1.93}$]

Answer: $w_{\text{new}} = 1.93$
The weight got smaller because: (1) decay shrunk it a bit, and (2) gradient step reduced it again.

3.2 L1 Regularization (Lasso)

Definition (Beginner Friendly)

L1 Regularization adds a penalty based on the **absolute value** of weights.

Big idea (beginner): L1 can push some weights to become exactly **0**. This means the model can automatically **ignore** some inputs.

In simple words: L1 can make the model simpler by turning off unimportant features.

Key Formula + Meaning

L1 Regularized Loss: [$\boxed{L_{\text{total}} = L_{\text{original}} + \lambda|\mathbf{w}| * 1 = L * \text{original} + \lambda \sum_i |w_i|}$]

Meaning:

- Bigger weights give a bigger penalty.
- Model tries to keep many weights at 0 when possible.

3.3 L1 vs L2 Comparison

Aspect	L1 (Lasso)	L2 (Weight Decay)
Penalty uses	Absolute value $\|w\|$	Square w^2
Effect	Some weights become exactly 0	Weights shrink but usually not to 0 height
Good when	Many features might be useless	Most features matter a bit

3.4 Dropout Regularization

Definition (Beginner Friendly)

Dropout means: during training, we randomly **turn off some neurons**.

Why it helps:

- Network cannot depend on only a few neurons.
- It learns features that still work if some neurons are missing.

Important: Dropout is used only during training. During testing, we use the full network.

Key Formula + Meaning

Dropout Operation: [$\boxed{h' = \frac{h \odot m}{1-p}}$] Where:

- h = neuron outputs before dropout
- m = mask of 0/1 (0 = dropped, 1 = kept)
- p = probability of dropping a neuron
- $\odot$ = multiply each element
- divide by $(1-p)$ so the average size stays similar

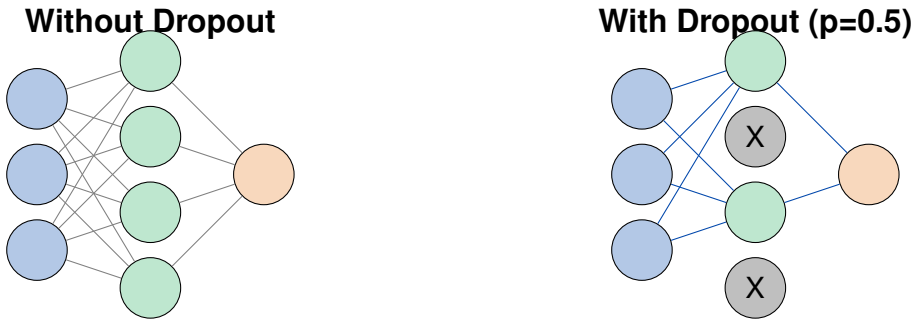


Figure 3: Dropout randomly turns off neurons during training

Solved Example 2: Dropout Calculation

Given Data:

- Activations: $h = [0.5, 0.8, 0.3, 0.9, 0.2]$
- Dropout probability: $p = 0.4$
- Mask: $m = [1, 0, 0, 1, 1]$

Solution:
Step 1: Apply the mask [[0.5, 0.8, 0.3, 0.9, 0.2] $\odot [1, 0, 0, 1, 1] = [0.5, 0, 0, 0.9, 0.2]]$
Step 2: Scale by $\frac{1}{1-p} = \frac{1}{0.6} \approx 1.667$ [[0.5,0,0,0.9,0.2]$_{0.6=[0.833,0,0,1.5,0.333]]}$

Answer: $h' = [0.833, 0, 0, 1.5, 0.333]$
Why scale? Because on average we keep only $1-p$ of neurons. Scaling keeps the average activation size similar between training and testing.

4 Early Stopping

Why It Matters

Early stopping is one of the easiest ways to reduce overfitting. It stops training at the moment validation performance starts getting worse.

Definition (Beginner Friendly)

Early Stopping means we stop training **before overfitting starts**.

We watch the **validation error**:

- If validation error improves, keep training.
- If validation error does not improve for some time, stop.

Patience = how many epochs we wait before stopping (example: 5 epochs).

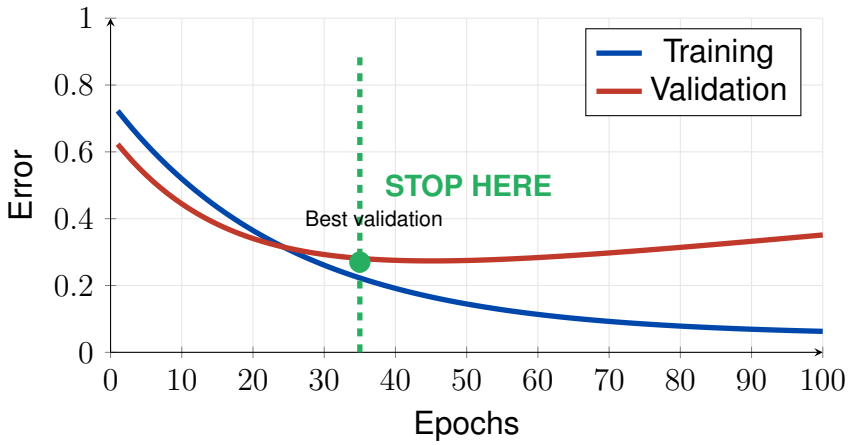


Figure 4: Early stopping: stop when validation error starts increasing

Memory Hook — Remember This!

Early Stopping Checklist:

- Use a validation set (not the test set)
- Patience is often 5–10 epochs
- Keep the best model weights, not necessarily the last epoch

5 Adding Training Data Data Augmentation

Why It Matters

More data almost always helps. If you cannot collect new data, augmentation creates extra training examples by safe transformations.

Definition (Beginner Friendly)

Data Augmentation means we create extra training examples by slightly changing existing data, but **the label stays the same**.

Example (images): rotate a cat photo a little, it is still a cat.

Goal: show the model more variety so it generalizes better.

Memory Hook — Remember This!

Caution: only use changes that keep the meaning!

- Horizontal flip is fine for animals, but not for digits (6 $\leftrightarrow$ 9)
- Big rotations might break street-sign meaning

6 Batch Normalization

Why It Matters

Training can become unstable if numbers inside the network get too large or too small. Batch normalization helps keep values in a stable range.

Definition (Beginner Friendly)

Batch Normalization (BN) makes training more stable by keeping numbers inside the network in a "nice range".

Simple idea:

- For each mini-batch, BN normalizes values (so they are centered and scaled).
- Then it learns two parameters (γ, β) to choose the best scale and shift.

Result: training is often faster and less sensitive to the learning rate.

Key Formula + Meaning

Batch Normalization Steps:
Step 1: Batch mean [$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$]
Step 2: Batch variance [$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$]
Step 3: Normalize [$_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$]
Step 4: Scale and shift (learned) [$\boxed{y_i = \gamma \hat{x}_i + \beta}$]
Meaning:

- m = batch size (how many samples in the mini-batch)
- ϵ = tiny number to avoid division by zero
- γ, β are learned so BN does not harm model power

Solved Example 3: Batch Normalization

Given Data:

- Mini-batch: $x = [1, 2, 3, 4, 5]$
- $\gamma = 1.0$, $\beta = 0.0$
- $\epsilon = 10^{-5}$

Solution:
Mean: [$\mu = \frac{1+2+3+4+5}{5} = \mathbf{3.0}$]
Variance: [$\sigma^2 = \frac{(1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2}{5} = \frac{4+1+0+1+4}{5} = \mathbf{2.0}$]
Normalize: [$_i = \frac{x_i-3}{\sqrt{2+10^{-5}}} \approx \frac{x_i-3}{1.4142}$]
Scale + shift: here $\gamma = 1$, $\beta = 0$, so output equals normalized values.

Answer: $y = [-1.414, -0.707, 0, 0.707, 1.414]$

7 K-Fold Cross Validation

Why It Matters

We should not tune hyperparameters using the test set. K-fold cross validation gives a fair estimate using multiple train/validation splits.

Definition (Beginner Friendly)

K-Fold Cross Validation is a fair way to test a model when data is limited.

How it works:

1. Split training data into k equal parts (folds).
2. Train the model k times.
3. Each time: 1 fold is validation, $k-1$ folds are training.

Final score: average of the k validation results (often shown as mean $\pm$ standard deviation).

5-Fold Cross Validation:

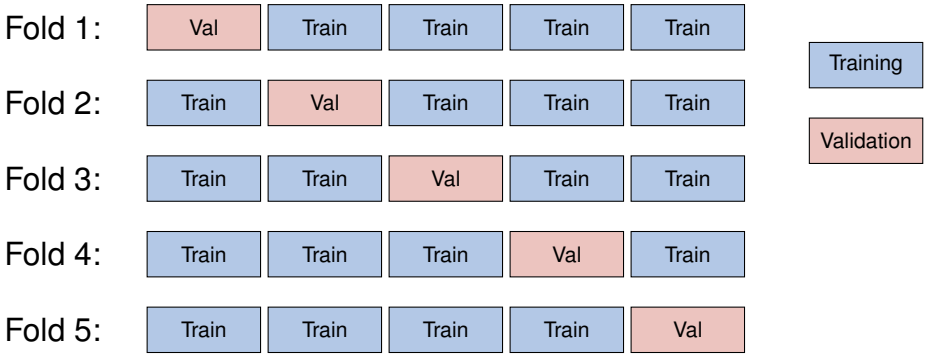


Figure 5: 5-Fold CV: each fold becomes validation once

Solved Example 4: K-Fold Cross Validation

Given Data:
A 5-fold cross validation was done. The accuracies are:

Fold	1	2	3	4	5 height**Accuracy**
0.85	0.88	0.82	0.87	0.83 height	

Find mean and standard deviation.

Solution:
Mean: [$= 0.85+0.88+0.82+0.87+0.83 \frac{}{5=\frac{4.25}{5}=\mathbf{0.85}}$]
Variance (population form): [$\sigma^2 = \frac{\sum(x_i-\bar{x})^2}{5} = \frac{0+0.0009+0.0009+0.0004+0.0004}{5} = 0.00052$]
Std: [$\sigma = \sqrt{0.00052} = \mathbf{0.0228}$]

Answer: Mean accuracy $= 85.0$

8 Optimization Methods

Why It Matters

Basic SGD can be slow and may bounce around. Modern optimizers adjust step sizes automatically and often train faster.

8.1 Problems with Vanilla SGD

Issues with Standard SGD (Simple View)

1. **One learning rate for everything** — but different weights may need different step sizes
2. **Can oscillate** — may bounce in narrow valleys
3. **May converge slowly** — needs careful tuning

8.2 Adagrad

Definition (Beginner Friendly)

Adagrad changes the learning rate for each weight automatically.

Simple rule:

- If a weight has had big gradients many times, Adagrad makes its steps smaller.
- If a weight rarely changes, it can get bigger steps.

Main drawback: steps can become too small after many updates, so learning may slow down too much.

Key Formula + Meaning

Adagrad Update: [$\boxed{\text{cache} \leftarrow \text{cache} + (\nabla w)^2}$] [$\boxed{w \leftarrow w - \frac{\eta}{\sqrt{\text{cache}} + \epsilon} \cdot \nabla w}$]

Meaning:

- cache grows with squared gradients
- bigger cache $\Rightarrow$ smaller effective step size
- ϵ avoids division by zero

8.3 RMSprop

Definition (Beginner Friendly)

RMSprop is like Adagrad, but it avoids the "step size goes to zero" problem.

Instead of storing all past gradients forever, it keeps a **recent average**. So it does not shrink learning rate too much.

Key Formula + Meaning

RMSprop Update: [$\boxed{\text{cache} \leftarrow \rho, \text{cache} + (1-\rho)(\nabla w)^2}$] [$\boxed{w \leftarrow w - \frac{\eta}{\sqrt{\text{cache}} + \epsilon} \cdot \nabla w}$]

Meaning:

- ρ is a decay rate (often 0.9 or 0.99)
- old gradients slowly "fade out"

8.4 Adam

Definition (Beginner Friendly)

Adam is a strong default optimizer because it is usually fast and stable.

Beginner view: Adam combines two ideas:

- **Momentum:** remembers the direction to move (smoother updates)
- **Adaptive step sizes:** like RMSprop, adjusts learning rate per weight

Key Formula + Meaning

Adam Update:

Step 1: Update moving averages [m $\leftarrow \beta_1 m + (1-\beta_1)\nabla w][v \leftarrow \beta_2 v + (1-\beta_2)(\nabla w)^2]$

Step 2: Bias correction (early steps) [= m$\frac{}{1-\beta_1^t, \quad \hat{v} = \frac{v}{1-\beta_2^t}]$

Step 3: Update [$\boxed{w \leftarrow w - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}, \hat{m}}$]

Common defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Solved Example 5: Adam (One Step)

Given Data:

- $w = 2.0$, $\eta = 0.1$, $\nabla w = 0.5$
- $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
- Start: $m = 0$, $v = 0$, $t = 1$

Solution:

Update moments: [m = 0.9$\cdot 0 + 0.1 \cdot 0.5 = 0.05][v = 0.999 \cdot 0 + 0.001 \cdot (0.5)^2 = 0.00025]$

Bias correction: [= 0.05$\frac{}{1-0.9=0.5, \quad \hat{v} = \frac{0.00025}{1-0.999} = 0.25]}$

Update weight: [$w_{\text{new}} = 2.0 - \frac{0.1}{\sqrt{0.25}+10^{-8}} \cdot 0.5 = 2.0 - \frac{0.1}{0.5} \cdot 0.5 = 2.0 - 0.1 = \mathbf{1.9}]$

Answer: After 1 step, $w = 1.9$.

8.5 Comparison of Optimizers

Optimizer	Main idea (simple)	Pros	Cons
SGD			
Fixed step size	Simple	Needs careful learning rate	Per-weight step size shrinks over time
Adagrad			
Helps sparse features	Can stop learning (steps too small)	Recent average of squared gradients	Keeps learning
RMSprop			
Has decay parameter ρ height	Momentum + adaptive steps	Fast, stable default	Sometimes generalizes slightly worse
Adam			

Table 2: Optimizer comparison (high-level)

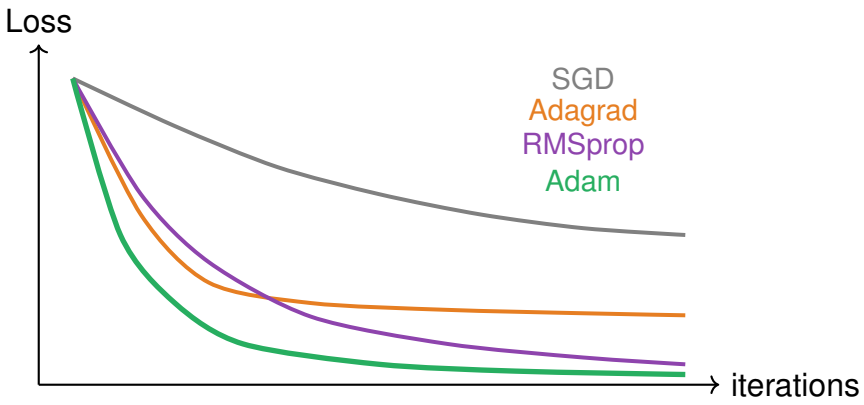


Figure 6: Typical convergence behavior (illustration)

9 Summary

Key Takeaways (Beginner Summary)

1. Generalization

- Goal: perform well on **new unseen data**
- Gap = Test error − Training error (large gap means overfitting)

2. Underfitting vs Overfitting

- Underfitting: too simple → increase capacity, train longer
- Overfitting: memorizes noise → regularize, dropout, early stop, more data

3. Regularization Tools

- L2: shrink weights gently
- L1: can push some weights to 0
- Dropout: randomly turn off neurons during training
- BatchNorm: stabilize internal values to train faster

4. Early Stopping

- Stop when validation error stops improving

5. K-Fold CV

- Train k times, each fold becomes validation once
- Report mean $\pm$ std

6. Optimizers

- Adagrad: steps shrink over time
- RMSprop: uses recent average so it keeps learning
- Adam: momentum + adaptive steps (common default)

Self-Test — Check Your Understanding

Quick Quiz:

1. What is the generalization gap formula used here?
2. If dropout probability is $p = 0.3$, what is the scaling factor for kept neurons?
3. In 10-fold cross validation, how many times do you train the model?

Answers:

1. Gap = Test error − Training error
2. $\frac{1}{1-p} = \frac{1}{0.7} \approx 1.43$
3. 10 times

10 Glossary

Term	Beginn Friendl Mean- ing
	Genera
Doing well on new data, not only on training data height**Generalization Gap**	How much worse test er- ror is com- pared to train- ing er- ror (Test − Train)
	Overfitt
Training looks great, but test performance is poor (memorizing noise) height**Underfitting**	Model is too sim- ple, so it per- forms poorly even on train- ing data
	Regula
Tricks to reduce overfitting and improve real-world performance height**Weight Decay (L2)**	Penalty that slowly shrinks weights so they do not