# Day 20: How Neural Networks Learn – The Secret of Weight Updates

## 🎯 Today's Focus:

We dived deep into the *engine room of neural networks*: **how weights get updated during training**. This process is what actually transforms a "random" network into a smart model capable of accurate predictions.

---

### □ What Are Weights Really Doing?

- Every connection between neurons carries a **weight**, which tells the model *"how important this input is"*.
- Initially, these weights are random — which is why early predictions are usually terrible.
- Training gradually tweaks these weights so the network's outputs align closer with real-world data.

💡 **Think of it like cooking:**
At first, you may add random salt/spices (random weights). Tasting the dish (loss) tells you how far you are from perfect. With each taste-test-feedback cycle, you adjust proportions (weight updates) until it's just right.

---

### 🔄 The Learning Cycle in 4 Simple Steps

1 **Forward Pass:** Data flows through the network to generate a prediction.
2 **Loss Measurement:** Compare prediction vs. actual value using a loss function (e.g., MSE for regression).
3 **Backpropagation:** Calculate how much each weight contributed to the error using derivatives.
4 **Weight Update (Gradient Descent):**

$$w_{new} = w_{old} - \eta \times \frac{\partial L}{\partial w}$$

---

### 🔬 Mini Python Demo – Watch a Weight Learn

```
import numpy as np

# Training data: simple linear relation y = 2x

x = np.array([1, 2, 3])

y = np.array([2, 4, 6])

w = 0.0        # start with a random weight

lr = 0.01      # learning rate

def mse(y, y_pred):

   return ((y - y_pred) ** 2).mean()

for epoch in range(50):

   y_pred = w * x           # Forward pass

   loss = mse(y, y_pred)      # Loss calculation

   grad = -2 * np.dot((y - y_pred), x) / len(x)  # Gradient

   w -= lr * grad           # Weight update
```

```
if epoch % 10 == 0:
 print(f"Epoch {epoch}: Loss={loss:.4f}, Weight={w:.4f}")
```

🔍 Over epochs, weight w steadily moves closer to 2 (the true relationship).

---

## 🌟 Extra Nuggets We Discussed

- **Small learning rate** → safe but slow (like baby steps).
- **Large learning rate** → risky, might "jump over" the target.
- **Bias terms** shift activation functions to fit data better.
- Advanced optimizers like **Adam & RMSProp** bring auto-tuning and momentum to speed things up.

---

## 🎯 Quick Analogy – Learning to Throw Darts

You throw a dart (initial prediction) → miss the bullseye (loss) → adjust your hand slightly (weight update) → repeat. Eventually, your throws consistently hit the center. That's exactly how weight updates refine a neural network!

---

## ✅ Today's Takeaways

- Weights are the **knobs** a neural network turns to learn patterns.
- Backprop + Gradient Descent = the "trial-and-error" feedback loop.
- Proper learning rates and optimizers ensure fast yet stable training.