**🎓 Backpropagation (Simple Version)**

**1. What is it?**

* Backpropagation = the way a neural network **learns from its mistakes**.
* It tells us how to adjust each weight so the model gets better next time.

**2. The Two Steps**

1. **Forward pass** →
   * Input goes through the network.
   * We get an output (a prediction).
   * Compare prediction to the real answer → this gives an **error**.
2. **Backward pass** →
   * Start with that error.
   * Send it backward through the network.
   * Figure out which weights caused how much of the error.
   * Update all the weights slightly to reduce the error.

**3. Why Do We Need It?**

* For the **output layer**, it’s easy: error = prediction – target.
* But hidden layers don’t have targets, so it’s not obvious how wrong they are.
* Backprop solves this by spreading the error backward and **assigning responsibility** to every hidden neuron.

**4. The Big Picture**

* Forward = “What’s my guess?”
* Compare = “How wrong was I?”
* Backward = “Who caused the mistake?”
* Update = “Let’s fix it.”

**✅ Takeaway**

👉 Backpropagation is just the network saying:  
“I made a mistake. Let me trace it back through all my layers, see which weights were responsible, and fix them a little bit.”

**🎓 Lecture: Backpropagation Illustrated (Single Hidden Layer)**

**1. Our Simple Network**

We’re looking at a **small network** with:

* **Inputs**:
* **Hidden layer units**:
* **Outputs**:
* **Targets**:

**Weights**:

* From inputs → hidden:
* From hidden → outputs:

We separate them into and so we know **which set of weights belongs to which part of the network**.

**2. Errors at the Output**

* At the output, we have predictions .
* We compare them with targets .
* That gives us two errors:

👉 Updating the -weights is straightforward:

* Each -weight directly contributes to **one output error**.
* Example: only affects , so it only needs .
* Update rule = adjust based on its derivative w.r.t. .

**3. What About the W-Weights?**

Here’s the tricky part.

* Take .
* It affects hidden unit .
* But influences **both** outputs and .
* So contributes to **both errors**: and .

👉 That means updating isn’t as simple as just looking at one error.

**4. The Backpropagation Solution**

* We **backpropagate the errors** from the outputs back into the hidden layer.
* Using the -weights, we can measure **how much each hidden unit contributed to each output error**.
* Then we combine those contributions.
* Once we know how responsible a hidden unit was, we can go back one more step and update its incoming weights ().

👉 In short:

* **Output errors → traced back through U → hidden layer → traced back through W.**

**5. Adjustments**

* If a weight contributed **a lot** to the error → adjust it more.
* If it contributed **a little** → adjust it less.
* This way, every weight update is **proportional to its responsibility** in the error.

**6. Activation Functions Complicate Things**

* So far, we only talked about linear connections.
* But in reality, neurons also have **activation functions** (ReLU, sigmoid, etc.).
* That means when we backpropagate, we must also account for the **derivative of the activation function**.
* This makes the math heavier — but conceptually it’s the same: trace responsibility and update weights.

**✅ Key Takeaway**

Backpropagation is the method that:

1. Starts with errors at the output.
2. Sends those errors backward through the network.
3. Figures out how much each weight (U or W) contributed.
4. Updates weights proportionally to their contribution.

👉 Output weights are easy — they connect directly to errors.  
👉 Input→hidden weights are harder — they affect multiple outputs, so we backpropagate errors through the network to update them correctly.