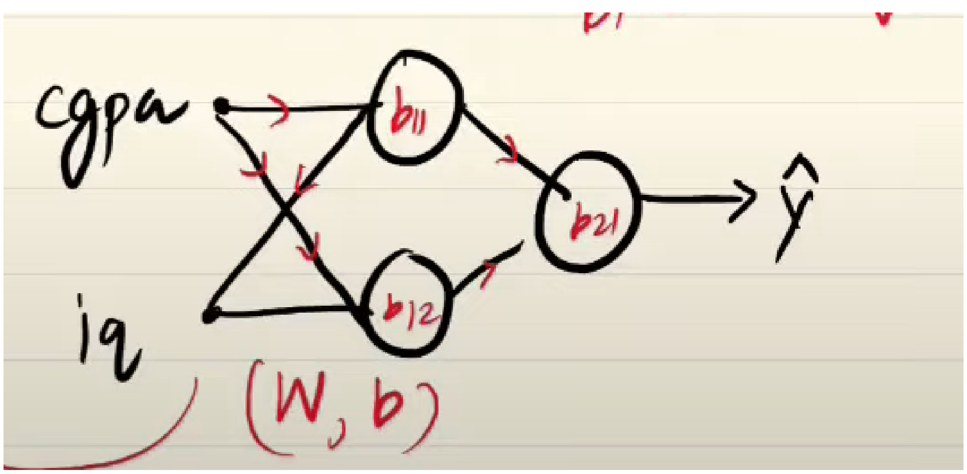
Backpropagation in Deep Learning

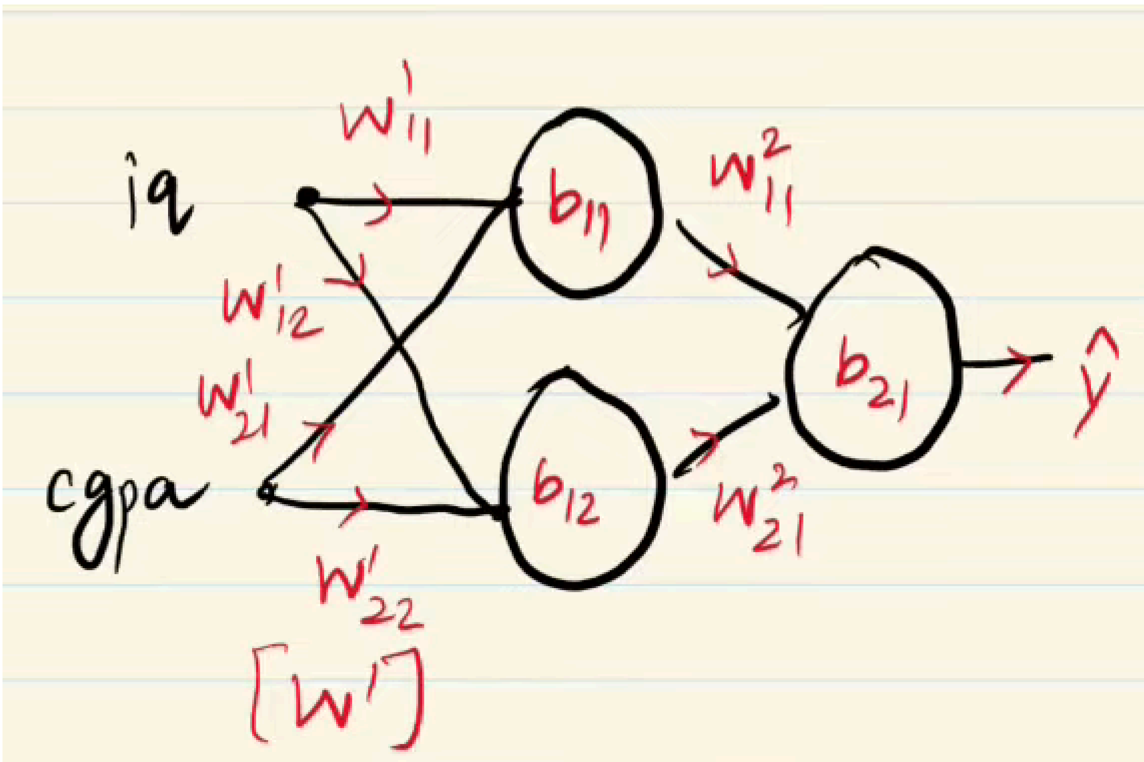
- Backpropagation (short for "backward propagation of errors") is a key algorithm used for training artificial neural networks.
- It's a supervised learning technique that helps the network learn by adjusting the weights of neurons in order to minimize the difference between predicted outputs and actual target values.

📌 **Key Idea**:

1. **Forward Propagation** → Compute predictions.
2. **Calculate Loss** → **Calculating the error** between predictions and true values.
3. **Backward Propagation** → Compute ***gradients (derivatives)*** of the loss function w.r.t. weights.
4. **Update Weights** → Adjust weights using Gradient Descent to reduce the error.



💡 **Backpropagation finds out appropriate values of weights & biases.**



Data:

iq	cgpa	lpa
80	8	3
60	9	5
70	5	8
120	7	11

Steps

Note: Here, activation function is Linear.

1. Initialize weights & biases (w, b)

w→**1**

b→ **0**

2. Select a row

- Feed the data
  - eg. ***80 & 8*** (Row 1👆)

3. Predict using the initial weight & bias

- This step is called **forward propagation**.
- eg. The model predicted **18**
  - 👆This is obviously far from the actual value (**3**)

4. Choose a Loss Function

- eg. **MSE** for regression
  - $(3-18)^2 = 225$
- Here, we have to decrease the value.
- To decrease the value, we have to **go back and change the values** of w & b
  - Therefore, the name → **Back propogation**

5. Update the weights & biases

- Use → ***Gradient Descent***

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial L}{\partial b_{old}}$$

η = Learning rate

For above example, we have to calculate these 9 derivatives:

$$\frac{\partial L}{\partial w_{11}^{2}}, \frac{\partial L}{\partial w_{21}^{2}}, \frac{\partial L}{\partial b_{21}} \;\Big|\; \frac{\partial L}{\partial w_{11}^{1}}, \frac{\partial L}{\partial w_{21}^{1}}, \frac{\partial L}{\partial b_{11}} \;\Big|\; \frac{\partial L}{\partial w_{12}^{1}}, \frac{\partial L}{\partial w_{22}^{1}}, \frac{\partial L}{\partial b_{12}}$$

Meaning of Derivative:

> 💡 **How much the L changes when we are changing w or b**

- L is loss
- The loss changes when we make changes in weights and biases. We're finding out→how much does it change?

We cannot directly calculate:

$$\boxed{\frac{\partial L}{\partial \hat{y}}}$$

First we have to calculate

$$\boxed{\frac{\partial \hat{y}}{\partial w^2_{11}}}$$

- After that we'll multiply these 2

$$\frac{\partial L}{\partial w^2_{11}} = \boxed{\frac{\partial L}{\partial \hat{y}}} \times \boxed{\frac{\partial \hat{y}}{\partial w^2_{11}}}$$

👆 This is called **chain rule**.

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} (y - \hat{y})^2 = \boxed{-2(y - \hat{y})}$$

$$\frac{\partial \hat{y}}{\partial w^2_{11}} = \frac{\partial}{\partial w^2_{11}} \left[O_{11} W^2_{11} + O_{12} W^2_{21} + b_{21} \right]$$

$$= O_{11}$$

$$\frac{\partial L}{\partial w^2_{11}} = -2(y-\hat{y})O_{11}$$

Similarly, the second derivative will be

$$\frac{\partial L}{\partial w^2_{21}} = -2(y-\hat{y})O_{12}$$

Bias:

$$\boxed{\frac{\partial L}{\partial b_{21}} = -2(y-\hat{y})}$$

6. Repeat this process

💡 Key Insights

1. **Vanishing Gradients**:

- If gradients become too small (e.g., in sigmoid/tanh), early layers learn **slowly**.
- **Fix**: Use **ReLU** or **batch normalization**.

2. **Exploding Gradients**:
    - If gradients grow too large (common in RNNs), training becomes unstable.
    - **Fix**: Use **gradient clipping**.
3. **Local Minima**:
    - Backpropagation can get stuck in suboptimal solutions.
    - **Fix**: Use **momentum** (e.g., Adam optimizer).

MLP Memoization – Optimizing Neural Network Efficiency

> 💡 **Memoization is already built-in in Keras library**

- **Memoization** (caching intermediate results) can **speed up training & inference** in **Multilayer Perceptrons (MLPs)** by avoiding redundant computations.

📌 What is Memoization in MLPs?

- **Stores layer outputs** during forward/backward passes.
- **Reuses them** instead of recalculating (e.g., in loops or repeated calls).
- **Trade-off**: Saves compute but increases memory usage.

Limitations:

- **Memory usage**: Memoization consumes memory to store intermediate results, which could become an issue when training deep networks with large datasets.
- **Batch processing**: When processing large batches of data, memoization might be less effective because the inputs are often different across samples, leading to fewer cache hits.