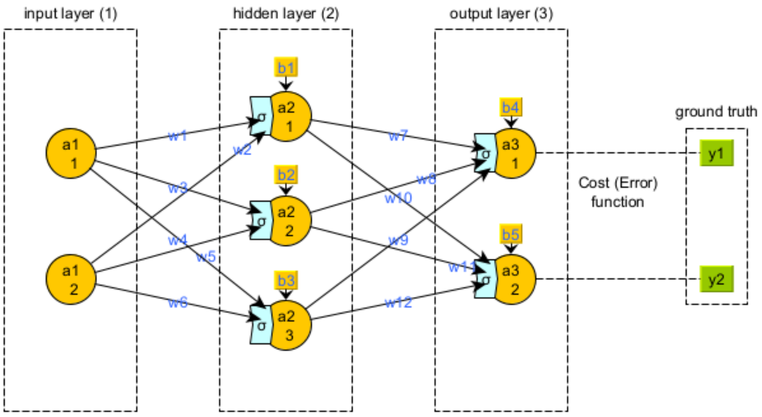
Forward Propagation

Definition: Forward propagation refers to the process where input data is fed through the neural network, layer by layer, to generate an output.

Steps Involved:

1. **Input Layer**: The process begins with the input layer, where the initial data (features) are fed into the network.
2. **Hidden Layers**: Each hidden layer receives inputs from the previous layer. The neurons in each hidden layer calculate a weighted sum of their inputs (from the previous layer) and apply an activation function to produce an output. This output becomes the input for the next layer.
3. **Output Layer**: The final hidden layer's output is passed through the output layer, where it undergoes a final activation function. The output of the output layer depends on the nature of the problem being solved (e.g., regression, binary classification, multi-class classification).
4. **Prediction**: The output layer's activation generates predictions or classifications based on the input data.



$$\sigma\left(\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \end{bmatrix} \times \overset{\text{input}}{\begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right) = \begin{bmatrix} \alpha_1^2 \\ \alpha_2^2 \\ \alpha_3^2 \end{bmatrix} \quad \sigma\left(\begin{bmatrix} w_7 & w_8 & w_9 \\ w_{10} & w_{11} & w_{12} \end{bmatrix} \times \begin{bmatrix} \alpha_1^2 \\ \alpha_2^2 \\ \alpha_3^2 \end{bmatrix} + \begin{bmatrix} b_4 \\ b_5 \end{bmatrix}\right) = \overset{\text{output}}{\begin{bmatrix} \alpha_1^3 \\ \alpha_2^3 \end{bmatrix}}$$

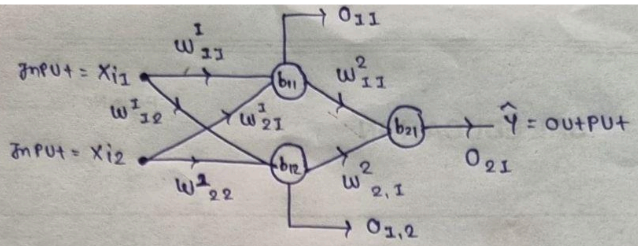
Purpose: Forward propagation serves to compute predictions or classifications based on the current weights and biases of the neural network. It is a deterministic process where the inputs are transformed through the network's layers to produce an output.

Backward Propagation (Backpropagation)

Definition: Backward propagation, or backpropagation, refers to the process where the network computes gradients of the loss function with respect to each weight and bias in the network

Steps Involved:

1. **Loss Calculation**: First, the loss function is calculated using the predicted output and the actual target values. The loss function quantifies how well or poorly the model performed on the training data.
2. **Gradient Calculation**: Starting from the output layer, the gradients of the loss function with respect to the weights and biases of each layer are computed using the chain rule of calculus. These gradients indicate how much each weight and bias contributed to the error in the prediction.
3. **Weight Update**: The gradients computed in the previous step are used to update the weights and biases of the neural network. This step involves using an optimization algorithm (e.g., gradient descent) to adjust the weights in the direction that reduces the error and improves the model's performance.
4. **Iterative Process**: Steps 1-3 are repeated iteratively for each batch of data (in mini-batch gradient descent) or for each data point (in stochastic gradient descent) until the model converges to optimal weights that minimize the loss function.



we use

loss_func = MSE

$$\left[\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2\right]$$

* using forward propagation predict output

* after we calculate loss function
* According to loss function we Reduce loss using formula

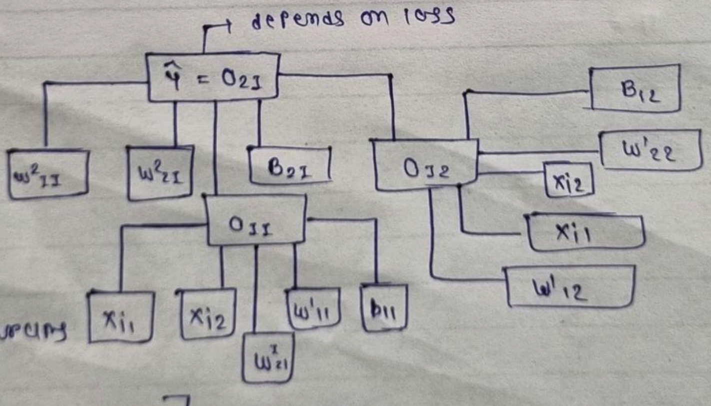
$$\boxed{W_{New} = W_{old} - \eta\frac{\partial L}{\partial \hat{y}}}$$

$$\boxed{B_{New} = B_{old} - \eta\frac{\partial L}{\partial B}}$$

For $\frac{\partial L}{\partial \hat{y}}$ Appliying chain Rule.

$$O_{12} = x_{i1}w^{1}_{12} + x_{i2}w^{1}_{22} + B_{12}$$
$$O_{11} = x_{i1}w^{1}_{11} + x_{i2}w^{1}_{21} + B_{11}$$

$\hat{y} \propto O_{21}$ and $O_{21} = w^{2}_{11}O_{11} + w^{2}_{21}O_{1,2} + B_{21}$



9 learnable params

$$\left[\frac{\partial L}{\partial \hat{y}}, \frac{\partial L}{\partial w^{2}_{11}}, \frac{\partial L}{\partial w^{2}_{21}}, \frac{\partial L}{\partial B_{21}}\right]$$ For layer 2

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

In deep learning, a loss function (or cost function) is a critical component used to guide the training of models. It quantifies how well the model's predictions match the actual data. By minimizing this loss, the model improves its performance. There are various types of loss functions used depending on the type of problem being solved, such as regression, classification, or others.

Types of Loss Functions

Mean Squared Error (MSE) and **Mean Absolute Error (MAE)** are both used in regression tasks, but they have different properties and are used in different scenarios based on the specific needs of the model and the nature of the data.

Mean Squared Error (MSE):-

Use Cases:

- **Sensitive to Outliers:** MSE penalizes larger errors more than smaller errors due to the squaring of differences. This makes it useful when you want your model to be sensitive to outliers.
- **Smooth Gradient Descent:** The squared differences provide a smoother gradient, which can make the optimization process more stable and potentially faster. This is especially important in complex models and deep learning.

Mean Absolute Error (MAE):-

Use Cases:

- **Robust to Outliers:** MAE is less sensitive to outliers because it uses absolute differences rather than squared differences. This makes it a better choice when the data contains outliers or when you want the model to be robust to them.
- **Interpretability:** MAE provides a more interpretable measure of average error, as it represents the average absolute difference between the predicted and actual values.

Binary Cross-Entropy (Log Loss):-

Binary Cross-Entropy (BCE) is used specifically in binary classification tasks. It is most appropriate when your model outputs a probability score between 0 and 1, indicating the likelihood that a given instance belongs to the positive class. Here are specific scenarios and contexts in which BCE is used:

Categorical Cross-Entropy:-

Categorical Cross-Entropy (CCE) is used in multi-class classification tasks where the goal is to classify instances into one of three or more possible categories. This loss function measures the performance of a classification model whose output is a probability distribution over multiple classes.

Huber Loss:-

Huber Loss is used in regression tasks where robustness to outliers is desired. It combines the advantages of Mean Squared Error (MSE) and Mean Absolute Error (MAE) by being less sensitive to outliers than MSE and more stable than MAE in optimization.