Backpropagation and Neural Networks Simplified

Introduction to Backpropagation

What is Backpropagation?

Backpropagation is an algorithm used to optimize a neural network by minimizing the error in its predictions. It works by calculating the gradient of the error with respect to the weights of the network, adjusting them iteratively to reduce the overall error.

Key points:

- It uses **gradient descent** to adjust the weights.
- The error is propagated backward, from the output layer to the input layer.

Steps of Backpropagation:

1. **Forward Propagation**: Calculate the output using initial weights.
2. **Calculate Error**: Compare predicted outputs with actual targets.
3. **Backward Propagation**: Compute gradients of the error with respect to weights using the chain rule.
4. **Update Weights**: Use the gradients to update weights.

Process of Building a Neural Network

1. **Perform Forward Propagation (FP):**
    - Input values are passed through the network using initial random weights.
    - Each neuron calculates its weighted sum and applies an activation function to produce an output.
2. **Calculate Gradient:**
    - Compute the partial derivative of the error with respect to each weight using the chain rule.
    - Gradients represent how much each weight contributes to the error.
3. **Update Weights:**
    - Adjust weights using the formula:

      where is the learning rate.

Step-by-Step Learning of a Neural Network

Example:

- 2 input neurons .
- 1 hidden layer with 2 neurons .
- 2 output neurons .

Step 1: Forward Propagation

1. Calculate inputs to the hidden layer:
2. Apply activation function to compute outputs:
3. Calculate inputs to output neurons:
4. Apply activation function to compute final outputs:

Step 2: Calculate Error

For each output neuron, compute the error:

Total error:

Step 3: Backpropagation

Use the chain rule to calculate the gradient of with respect to each weight.

1. **Output layer weights:**
2. **Hidden layer weights:** Similarly, calculate gradients for .

Step 4: Update Weights

Using the learning rate :

Code Example

Here's an example of backpropagation using Python:

```
import numpy as np

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Input data
inputs = np.array([[0.1, 0.05]])

# Target outputs
targets = np.array([[0.02, 0.98]])

# Initialize weights and biases
weights  input  hidden = np.array([[0.25, 0.2], [0.5, 0.15]])
weights_hidden_output = np.array([[0.7, 0.4], [0.6, 0.35]])
```

```
bias  hidden = np.array([0.3])
bias_output = np.array([0.9])

# Forward propagation
hidden_input = np.dot(inputs, weights_input_hidden) + bias_hidden
hidden_output = sigmoid(hidden_input)

output  input = np.dot(hidden  output, weights_hidden_output) + bias_output
output = sigmoid(output_input)

# Calculate error
error = 0.5 * (targets - output)**2

# Backpropagation (gradient calculation)
output  error = targets - output
output_delta = output_error * sigmoid_derivative(output)

hidden_error = np.dot(output_delta, weights_hidden_output.T)
hidden_delta = hidden_error * sigmoid_derivative(hidden_output)

# Update weights
learning_rate = 0.5
weights  hidden  output += learning  rate * np.dot(hidden  output.T, output_delta)
weights_input_hidden += learning_rate * np.dot(inputs.T, hidden_delta)

print("Updated weights (input to hidden):", weights  input  hidden)
print("Updated weights (hidden to output):", weights_hidden_output)
```

Learning Rate ()

- **Large learning rates** lead to large updates, risking overshooting the optimal solution.
- **Small learning rates** result in slow convergence.
- Choose carefully to balance speed and stability of learning.

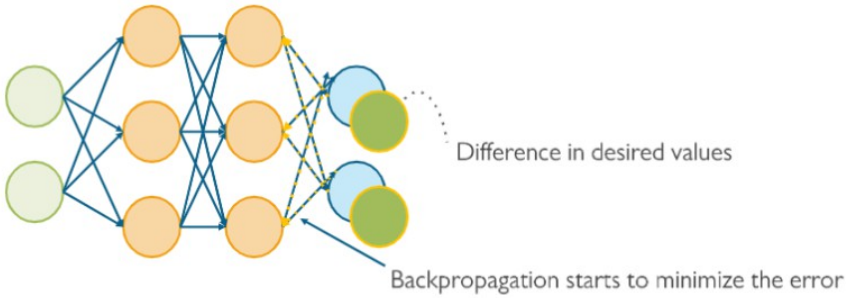
Conclusion

Backpropagation is a powerful algorithm for training neural networks, enabling them to learn complex patterns. By understanding and applying forward propagation, gradient calculation, and weight updates, we can build effective models for various tasks.

What is Backpropagation?

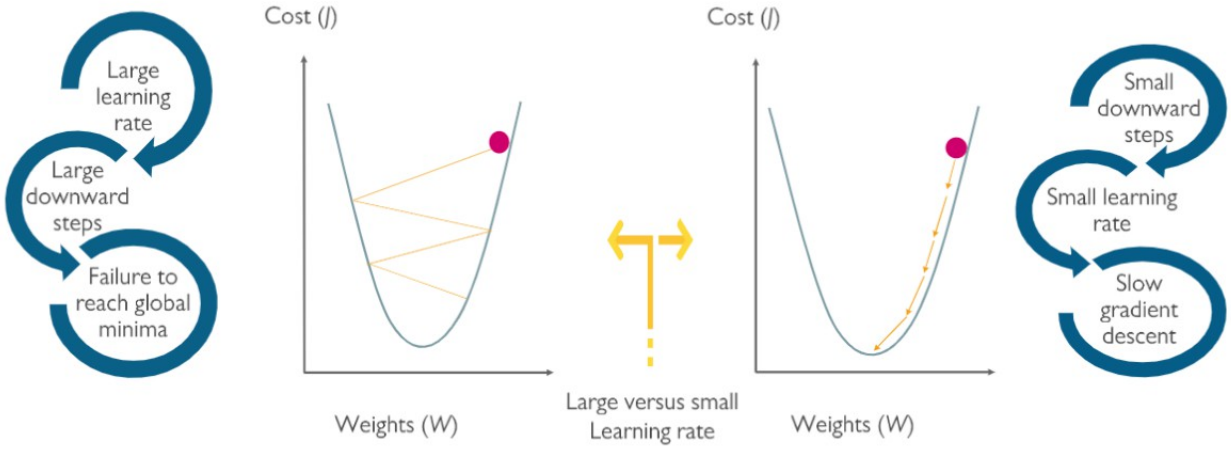
An algorithm to improve the accuracy of prediction and quick calculation of derivatives

- Backpropagation is a learning algorithm to compute gradient descent with respect to weight.
- Weight values are manipulated to get the predicted result.
- Backpropagation is used to update the weights from output to input.

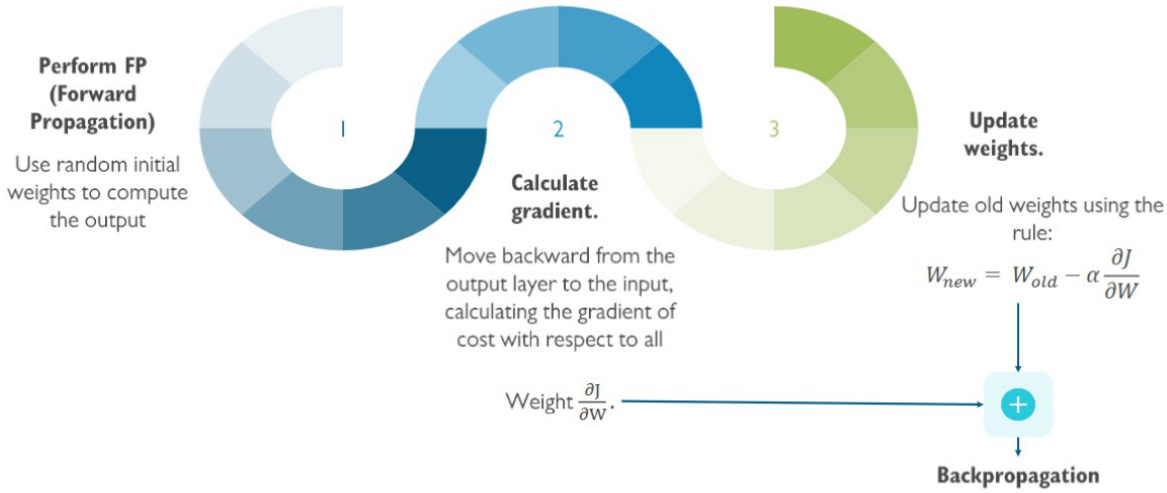


Learning Rate (α)

A hyperparameter that controls how much to update a network each time weights are updated to minimize the calculated cost



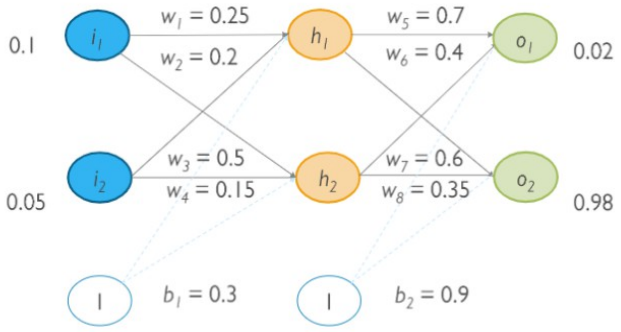
Process of Building a Neural Network: Overview



Step-By-Step Learning of a Neural Network

Consider the following neural network with:

- Two input neurons
- One hidden layer with two neurons
- Two output neurons

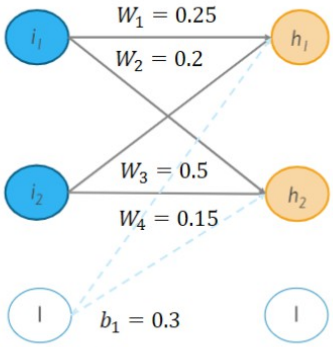


The goal of backpropagation is to optimize the weights so that the network can predict more accurate outputs for the given inputs.

Learning of a Neural Network: FP

Calculate the inputs to h_1 and h_2 = Sum of weighted inputs

$$in_{h_1} = W_1 \cdot i_1 + W_3 \cdot i_2 + b_1 \cdot 1 = 0.25 * 0.1 + 0.5 * 0.05 + 0.3 = 0.35$$
$$in_{h_2} = W_2 \cdot i_1 + W_4 \cdot i_2 + b_1 \cdot 1 = 0.25 * 0.1 + 0.15 * 0.05 + 0.3 = 0.3275$$



Calculate the outputs to h_1 and h_2 = Application of activation function

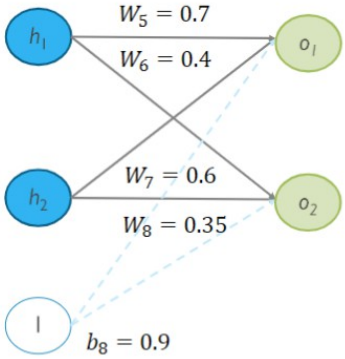
$$out_{h_1} = \frac{1}{1 + e^{-inh_1}} = 0.5866$$

$$out_{h_2} = \frac{1}{1 + e^{-inh_2}} = 0.5812$$

Learning of a Neural Network: FP (contd.)

Calculate the inputs to o_1 and o_2: Sum of weighted outputs from hidden layer

$$in_{o_1} = W_5 \cdot h_1 + W_7 \cdot h_2 + b_2 \cdot 1 = 1.345$$
$$in_{o_2} = W_6 \cdot h_1 + W_8 \cdot h_2 + b_2 \cdot 1 = 1.1546$$



Calculate the outputs to o_1 and o_2: Application of activation function

$$out_{o_1} = \frac{1}{1 + e^{-ino_1}} = 0.7927$$

$$out_{o_2} = \frac{1}{1 + e^{-ino_2}} = 0.7927$$

Learning of a Neural Network: Calculating The Error

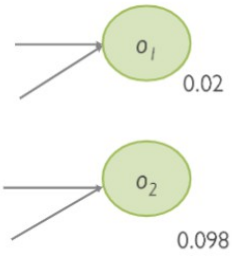
Calculate the error for o_1 and o_2

$$Eo_1 = \frac{1}{2}(target0_1 - outo_1)^2 = 0.2985$$

$$Eo_2 = \frac{1}{2}(target0_2 - outo_2)^2 = 0.0241$$

Calculate the total error as shown below

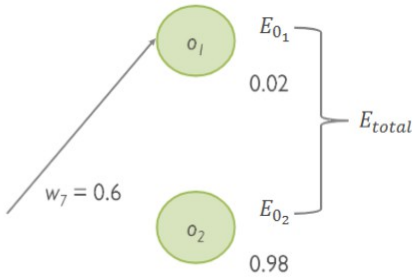
$$E_{total} = Eo_1 + Eo_2 = 0.3226$$



Learning of a Neural Network: Backpropagation

- Calculate the error gradient of E_{total} with respect to W_7.
- Using **chain rule,** it can be calculated as:

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial in_{01}} * \frac{\partial in_{01}}{\partial w_7} = 0.0745$$



- W_7 is updated as follows: $W_{7_{new}} = W_{7_{old}} = -\alpha * \frac{\partial E_{total}}{\partial w_7} = 0.5628 \ldots\ldots\ldots$ (using $x = \alpha = 0.5$)

Similarly, all other weights are updated.