Understanding Neural Networks from Scratch

-Karan Sehgal

Neural networks are computational models inspired by the human brain. They consist of connected nodes (neurons) organized in layers that transform input data into predictions. In this guide, we'll break down how neural networks work from the ground up.

Core Components of a Neural Network

1. Neurons

Neurons are the basic units of a neural network. Each neuron:

- Receives inputs from previous layers
- Applies weights to those inputs
- Sums the weighted inputs and adds a bias
- Passes the result through an activation function
- Produces an output that's sent to the next layer

2. Layers

Neural networks are organized into layers:

- **Input Layer**: Receives the initial data
- **Hidden Layers**: Intermediate layers that perform computations
- **Output Layer**: Produces the final prediction

3. Weights and Biases

- **Weights**: Determine the strength of connections between neurons
- **Biases**: Allow the network to shift the activation function

4. Activation Functions

Activation functions introduce non-linearity, allowing the network to learn complex patterns:

- **Sigmoid**: Maps values to a range between 0 and 1
    - Formula: σ(x) = 1 / (1 + e^(-x))
- **ReLU (Rectified Linear Unit)**: Returns x if x > 0, else 0
    - Formula: f(x) = max(0, x)
- **Tanh**: Maps values to a range between -1 and 1
    - Formula: tanh(x) = (e^x - e^(-x)) / (e^x + e^(-x))

How Neural Networks Work: Step by Step

1. Forward Propagation

This is how the network makes predictions:

1. **Input Layer**: The network receives input data
2. **Hidden Layers**: For each neuron in each hidden layer:
    - Calculate the weighted sum of inputs: $z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$
    - Apply the activation function: $a = activation(z)$
3. **Output Layer**: The final layer produces the prediction

2. Loss Calculation

After making a prediction, the network calculates how far off it was:

- **Mean Squared Error (MSE)**: For regression problems
    - Formula: $MSE = (1/n) \Sigma(y - \hat{y})^2$
- **Cross-Entropy Loss**: For classification problems
    - Formula: $-\Sigma(y \log(\hat{y}))$

3. Backpropagation

This is how the network learns:

1. Calculate the error at the output layer
2. Compute how much each weight contributed to the error
3. Update the weights and biases using gradients
4. Propagate the error backward through the network

4. Gradient Descent

The optimization algorithm that adjusts weights and biases:

1. Calculate gradients (the direction of steepest increase in error)
2. Move in the opposite direction of the gradient
3. Update weights and biases: w_new = w_old - learning_rate * gradient

XOR Problem: A Classic Example

The XOR (exclusive OR) problem is a classic example used to demonstrate neural networks because it cannot be solved with a single-layer network.

XOR Logic:

- 0 XOR 0 = 0
- 0 XOR 1 = 1
- 1 XOR 0 = 1
- 1 XOR 1 = 0

This problem requires a hidden layer because the data is not linearly separable.

Key Neural Network Terminology

- **Epoch**: One complete pass through the entire training dataset
- **Batch Size**: Number of training examples used in one iteration
- **Learning Rate**: How quickly the model updates its parameters
- **Overfitting**: When a model learns the training data too well, including noise

- **Regularization**: Techniques to prevent overfitting
- **Dropout**: Randomly disabling neurons during training to prevent overfitting
- **Hyperparameters**: Settings that control the learning process (like learning rate)

Types of Neural Networks

1. **Feedforward Neural Networks**: Information flows in one direction (our implementation)
2. **Convolutional Neural Networks (CNNs)**: Specialized for grid-like data such as images
3. **Recurrent Neural Networks (RNNs)**: Have connections that form cycles for processing sequences
4. **Long Short-Term Memory (LSTM)**: A type of RNN that can learn long-term dependencies
5. **Transformers**: Use attention mechanisms for tasks like natural language processing

Visual Understanding

In our interactive visualization:

1. **Network Architecture**: Shows the layers and connections between neurons
2. **Training Data**: The XOR problem visualized as points in 2D space
3. **Decision Boundary**: How the network divides the input space for classification
4. **Training Progress**: How loss decreases and predictions improve over time

Mathematical Foundations

Forward Propagation Math

For a neuron with inputs x_1, x_2, ..., x_n, weights w_1, w_2, ..., w_n, and bias b:

1. Calculate weighted sum: $z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$
2. Apply activation function: $a = \text{activation}(z)$

In matrix form for a layer with m neurons and n inputs:

- $Z = WX + B$
- $A = \text{activation}(Z)$

Backpropagation Math

For the output layer:

- Error: $\delta_j = (a_j - y_j) \times \text{activation}'(z_j)$

For hidden layers:

- Error: $\delta_j = (\Sigma\ w_{jk}\delta_k) \times \text{activation}'(z_j)$

Weight updates:

- $\partial C/\partial w_{jk} = a_j \times \delta_k$
- $w_{new} = w_{old} - \text{learning\_rate} \times \partial C/\partial w$

Practical Neural Network Tips

1. **Data Preprocessing**: Normalize or standardize inputs
2. **Weight Initialization**: Properly initializing weights is crucial for training
3. **Learning Rate Selection**: Too high = unstable, too low = slow convergence
4. **Batch Size Tuning**: Affects training speed and stability
5. **Architecture Design**: The number of layers and neurons impacts performance
6. **Regularization**: Apply techniques like L1/L2 regularization or dropout
7. **Monitoring**: Track training and validation metrics to avoid overfitting

Conclusion

Neural networks are powerful tools for pattern recognition and prediction tasks. By understanding how they work from first principles, you can better design, implement, and debug them for your specific problems.

With our interactive visualization and from-scratch implementation, you can see exactly how each component works together to form a complete neural network system.