Lecture 2: Introduction to Neural Networks; From Single Layer to Deep Networks

Justo E. Karell

September 12, 2024

CS 583: Deep Learning

Overview of Neural Networks

A neural network is a computational model composed of several interconnected layers of nodes (or neurons). Each node represents a mathematical function that processes inputs, applies transformations, and produces outputs. The layers of a neural network are as follows:

1. **Input Layer:** The input layer receives the raw input data as vectors. The input vector $\mathbf{x}$ has dimensions $\mathbb{R}^n$, where n is the number of input features.

2. **Hidden Layers:** These layers process inputs from the previous layer through a series of transformations. Each hidden layer is composed of multiple nodes, each of which applies a weighted sum to the input and then passes it through an activation function to introduce non-linearity.

3. **Output Layer:** The final layer outputs the predictions or classifications of the network. Its structure depends on the task (e.g., regression or classification) and can vary in the number of output nodes.

4. **Weights and Biases:** Each connection between nodes in adjacent layers has an associated weight w, and each node has a bias b. These parameters are learned during training through backpropagation and gradient descent.

5. **Activation Function:** The activation function $\sigma(z)$ introduces non-linearity into the model. Common activation functions include the sigmoid function, hyperbolic tangent, and ReLU (Rectified Linear Unit).

6. **Loss Function:** The loss function measures the difference between the predicted output and the actual target. Common loss functions are Mean Squared Error (MSE) for regression and Cross-Entropy Loss for classification.

7. **Backpropagation and Gradient Descent:** The parameters (weights and biases) are updated through backpropagation. The gradients of the loss function with respect to the parameters are computed and used to update the weights via gradient descent:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

where η is the learning rate and L is the loss function.

Mathematically, for a neural network with L layers, the output of the i-th node in layer l is given by:

$$h_i^{(l)} = \sigma \left(\sum_{j=1}^{n_{l-1}} w_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)} \right)$$

Where:

- $h_j^{(l-1)}$ is the output from node j in the previous layer $(l-1)$.
- $w_{ij}^{(l)}$ is the weight from node j in layer $(l-1)$ to node i in layer l.
- $b_i^{(l)}$ is the bias for node i in layer l.
- σ is the activation function, applied element-wise.

—

Introduction to Nodes in Neural Networks

The most fundamental building block of a neural network is a node. In a mathematical sense, a node is a computational unit that processes data, combining the input with weights, adding a bias, and passing the result through an activation function. Importantly, nodes introduce non-linearity into the system via activation functions, making neural networks powerful tools for modeling complex relationships.

A node is often compared to a function in traditional machine learning models (e.g., linear regression). However, unlike traditional models, nodes in neural networks can incorporate non-linear transformations and dynamic updates to their parameters, which enables them to solve more complex problems.

Node in a Neural Network

Mathematically, a node in a neural network is defined as:

$$h = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

Where:

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n$ is the input vector, a real-valued vector containing n features.
- $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T \in \mathbb{R}^n$ is the weight vector, a real-valued vector of the same dimension as the input.
- $b \in \mathbb{R}$ is the bias term, a scalar.
- σ is the activation function, such as ReLU, sigmoid, or others, applied to the weighted sum.
- $h \in \mathbb{R}$ is the output of the node, a scalar value.

Traditional Model Comparison

In traditional linear regression, the model for predicting an output y is:

$$y = \mathbf{w}^T\mathbf{x} + b$$

This is a simple weighted sum of inputs and is inherently linear.

In contrast, a neural network node extends this idea by adding an activation function σ, introducing non-linearity:

$$h = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

This allows neural networks to capture non-linear relationships between inputs and outputs.

Example: Node with a Linear Activation Function

To preserve linearity in the neural network, we could choose the identity activation function $\sigma(z) = z$. In this case, the node acts similarly to a basic linear model:

$$h = \mathbf{w}^T\mathbf{x} + b$$

Real-World Example with Data

Let's consider a basic input vector with two features:

$$\mathbf{x} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \in \mathbb{R}^2$$

With weights:

$$\mathbf{w} = \begin{bmatrix} 0.5 \\ -0.4 \end{bmatrix} \in \mathbb{R}^2$$

And bias:

$$b = 1 \in \mathbb{R}$$

For a simple node with a sigmoid activation function, the output is:

$$z = \mathbf{w}^T \mathbf{x} + b = 0.5(2) + (-0.4)(3) + 1 = 1.0$$

Applying the sigmoid function:

$$h = \frac{1}{1 + e^{-1.0}} \approx 0.731$$

The output $h \in \mathbb{R}$ is the node's result after one pass through this basic neural network node.

—

Iteration Explanation

In the example above, we've computed the result of one iteration or one pass through the neural network. In a full training process, many such passes through the data would be performed. An epoch refers to a full pass over the entire dataset. During an epoch, the model would adjust its weights through back-propagation to minimize the loss function.

—

Example 1: Single Neuron Perceptron (One Pass)

Let's perform a forward pass through a simple single neuron perceptron:

- Input vector: $\mathbf{x} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \in \mathbb{R}^2$ - Weights: $\mathbf{w} = \begin{bmatrix} 0.7 \\ -0.5 \end{bmatrix} \in \mathbb{R}^2$ - Bias: $b = 0.1 \in \mathbb{R}$ - Activation function: Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

Calculation:

$$z = \mathbf{w}^T \mathbf{x} + b = (0.7 \times 2) + (-0.5 \times 3) + 0.1 = 1.4 - 1.5 + 0.1 = 0$$

$$h = \sigma(0) = \frac{1}{1 + e^0} = 0.5$$

The output $h = 0.5 \in \mathbb{R}$.

—

Example 2: Two Neuron Layer (One Pass)

Consider a neural network with two neurons in the hidden layer:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \in \mathbb{R}^2$ - Weights for each neuron:

$$\mathbf{w}_1 = \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} \in \mathbb{R}^2, \quad \mathbf{w}_2 = \begin{bmatrix} -0.4 \\ 0.9 \end{bmatrix} \in \mathbb{R}^2$$

- Biases: $b_1 = 0.2 \in \mathbb{R}, \quad b_2 = -0.1 \in \mathbb{R}$ - Activation function: ReLU

Calculation for neuron 1:

$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1 = 0.6(1) + 0.3(2) + 0.2 = 1.4$$

$$h_1 = \max(0, 1.4) = 1.4$$

Calculation for neuron 2:

$$z_2 = \mathbf{w}_2^T \mathbf{x} + b_2 = (-0.4)(1) + 0.9(2) - 0.1 = 1.3$$

$$h_2 = \max(0, 1.3) = 1.3$$

The output of the layer is:

$$\mathbf{h} = \begin{bmatrix} 1.4 \\ 1.3 \end{bmatrix} \in \mathbb{R}^2$$

—

Example 3: Input Vector of Length 5 (One Pass)

Consider an input vector of length 5 and three neurons in the layer:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 0.5 \\ -1 \\ 2 \\ 0 \end{bmatrix} \in \mathbb{R}^5$ - Weights for each neuron:

$$\mathbf{w}_1 = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.5 \\ 0.3 \\ 0.7 \end{bmatrix} \in \mathbb{R}^5, \quad \mathbf{w}_2 = \begin{bmatrix} -0.3 \\ 0.8 \\ -0.1 \\ 0.4 \\ 0.2 \end{bmatrix} \in \mathbb{R}^5, \quad \mathbf{w}_3 = \begin{bmatrix} 0.5 \\ -0.4 \\ 0.6 \\ 0.1 \\ -0.2 \end{bmatrix} \in \mathbb{R}^5$$

- Biases: $b_1 = 0.1 \in \mathbb{R}, \quad b_2 = -0.2 \in \mathbb{R}, \quad b_3 = 0.05 \in \mathbb{R}$ - Activation function: Sigmoid

Calculation for neuron 1:

$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1 = 0.1(1) - 0.2(0.5) + 0.5(-1) + 0.3(2) + 0.7(0) + 0.1 = 0.3$$

$$h_1 = \sigma(0.3) = \frac{1}{1 + e^{-0.3}} \approx 0.574$$

Calculation for neuron 2:

$$z_2 = \mathbf{w}_2^T \mathbf{x} + b_2 = -0.3(1) + 0.8(0.5) - 0.1(-1) + 0.4(2) + 0.2(0) - 0.2 = 1.0$$

$$h_2 = \sigma(1.0) = \frac{1}{1 + e^{-1.0}} \approx 0.731$$

Calculation for neuron 3:

$$z_3 = \mathbf{w}_3^T \mathbf{x} + b_3 = 0.5(1) - 0.4(0.5) + 0.6(-1) + 0.1(2) - 0.2(0) + 0.05 = 0.0$$

$$h_3 = \sigma(0.0) = 0.5$$

The output of the layer is:

$$\mathbf{h} = \begin{bmatrix} 0.574 \\ 0.731 \\ 0.5 \end{bmatrix} \in \mathbb{R}^3$$

—

Layers in a Neural Network

A layer in a neural network consists of multiple nodes that operate in parallel on the same input. Each node applies a weighted sum, adds a bias, and passes the result through an activation function, producing an output. The collection of outputs from all nodes in the layer forms the layer's output vector.

Mathematical Representation For a layer with m nodes, the output vector $\mathbf{h}$ is:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Where:

- $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix, where each row corresponds to the weights for a node in the layer.
- $\mathbf{x} \in \mathbb{R}^n$ is the input vector.
- $\mathbf{b} \in \mathbb{R}^m$ is the bias vector, with one bias per node.

- σ is the activation function, applied element-wise to the vector.
- $\mathbf{h} \in \mathbb{R}^m$ is the output vector of the layer.

—

Example: Layer with 3 Neurons

For a layer with three neurons, the equations for each node are:

$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1, \quad z_2 = \mathbf{w}_2^T \mathbf{x} + b_2, \quad z_3 = \mathbf{w}_3^T \mathbf{x} + b_3$$

Stacking these equations:

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Where:

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \end{bmatrix} \in \mathbb{R}^{3\times 5}, \quad \mathbf{x} \in \mathbb{R}^5, \quad \mathbf{b} \in \mathbb{R}^3$$

The output of the layer is:

$$\mathbf{h} = \sigma(\mathbf{z}) = \begin{bmatrix} \sigma(z_1) \\ \sigma(z_2) \\ \sigma(z_3) \end{bmatrix}$$

—

Example: Layer with 2 Neurons (One Pass)

Consider a layer with two neurons, processing the following inputs:

- Input vector: $\mathbf{x} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} \in \mathbb{R}^2$ - Weights:

$$\mathbf{w}_1 = \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} \in \mathbb{R}^2, \quad \mathbf{w}_2 = \begin{bmatrix} -0.4 \\ 0.9 \end{bmatrix} \in \mathbb{R}^2$$

- Biases: $b_1 = 0.2 \in \mathbb{R}, \quad b_2 = -0.1 \in \mathbb{R}$ - Activation function: ReLU

1. Calculate the outputs for each neuron:

$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1 = 0.6(1) + 0.3(0.5) + 0.2 = 1.05$$

$$z_2 = \mathbf{w}_2^T \mathbf{x} + b_2 = (-0.4)(1) + 0.9(0.5) - 0.1 = 0.35$$

2. Apply the ReLU activation function:

$$h_1 = \max(0, 1.05) = 1.05, \quad h_2 = \max(0, 0.35) = 0.35$$

The output of the layer is:

$$\mathbf{h} = \begin{bmatrix} 1.05 \\ 0.35 \end{bmatrix} \in \mathbb{R}^2$$

—

What Makes Neural Networks Better than Linear Models?

Neural networks provide key advantages over linear models by incorporating non-linear activation functions and allowing for complex relationships between input and output. The primary benefits include:

1. Non-Linearity:

Linear models can only approximate linear relationships, while neural networks can learn non-linear patterns through non-linear activation functions like ReLU, Sigmoid, and Tanh.

2. Higher Capacity:

A neural network with multiple layers can model far more intricate functions, allowing for more flexible decision boundaries.

3. Layered Structure:

Deep networks can decompose complex input features layer by layer, with each layer learning increasingly abstract representations. For example, in image recognition, early layers might learn to detect edges, while deeper layers identify complex objects.

4. Universality:

Neural networks are universal function approximators. This means that given enough data and computational resources, they can approximate any function to arbitrary accuracy.

—

What Makes Neural Networks Better than Non-Linear Models?

While traditional non-linear models (like polynomial regression) can model some non-linearities, neural networks surpass them in several key ways:

1. Flexibility:

Neural networks can adapt to arbitrary complex non-linearities in the data by adjusting weights across many layers, whereas non-linear models are often constrained by their specific functional form (e.g., polynomial degree).

2. Scalability:

Neural networks can handle large datasets and large feature spaces efficiently using parallel computing and modern hardware like GPUs.

3. Feature Engineering:

Non-linear models often require manual feature engineering (e.g., creating polynomial or interaction terms). Neural networks automate this process, learning hierarchical feature representations during training.

4. Depth and Learning:

Deep neural networks can stack multiple non-linear transformations, learning more abstract and useful representations of the data at each layer.