Neural Network Hand Calculations

Hanhee Lee

May 2024

1 Introduction

This mini lecture introduces a simple regression models to demonstrate the underlying working mechanics of a shallow and deep neural network.

2 Notation

The following table summarizes the notation used in the neural network models:

Table 1: Detailed Explanation of Variables

Header	Dimension	Explanation
	Superscripts	
$[l]$	1	Current layer
(i)	1	ith training example
	Subscripts	
j	1	jth node of the current layer
k	1	kth input into current layer
	Sizes	
m	1	Number of training examples in the dataset
n_x	1	Number of nodes in the input layer
n_y	1	Number of nodes in the output layer
$n^{[l]}$	1	Number of nodes in the current layer
$n^{[l-1]}$	1	Number of nodes in the previous layer
L	1	Number of layers in the network
	Objects	
$\mathbf{X}$	$n_x \times m$	Input matrix
$\mathbf{x}^{(i)}$	$n_x \times 1$	ith example represented as a column vector
$\mathbf{W}^{[l]}$	$n^{[l]} \times n^{[l-1]}$	Weight matrix of the current layer
$z_j^{[l](i)}$	1	A weighted sum of the activations of the previous layer, shifted by a bias
$w_{j,k}^{[l]}$	1	A weight that scales the kth activation of the previous layer
$b^{[l]}$	$n^{[l]} \times 1$	Bias vector in the current layer
$b_j^{[l]}$	$n^{[l]} \times 1$	Bias in the current layer
$a_j^{[l](i)}$	1	An activation in the current layer
$a_k^{[l-1](i)}$	1	An activation in the previous layer
$g_j^{[l]}$	1	An activation function used in the current layer

3 Neural Network Formulas

This section describes the foundational equations used in neural networks, detailing the computation involved in forward propagation.

Layer-Wise Computation

$$z^{[l]} = \sum_{k} w_{k,j} x_k + b_{l,j} \tag{1}$$

This equation calculates the weighted sum $z_{l,i}$ of layer l, combined with the bias term $b_{l,j}$ for each neuron j in layer l, to produce the pre-activation value for neuron j in layer i for a given input x_k. The term $w_{j,k}$ represents the weight for the given input x_k in neuron j

Note:

- the initial weights are set as 1 in this example for calculation simplicity. During actual training, it's better to have random weight initialization to avoid all neurons learning the same feature, and allow the network to learn more complex patterns

$$L = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2} \tag{2}$$

This equation represents the root mean squared error, a common cost function used to measure the difference between the predicted outputs ($\hat{y}_i$) and the actual targets (y_i) over all m training examples.

4 Simple Regression Models

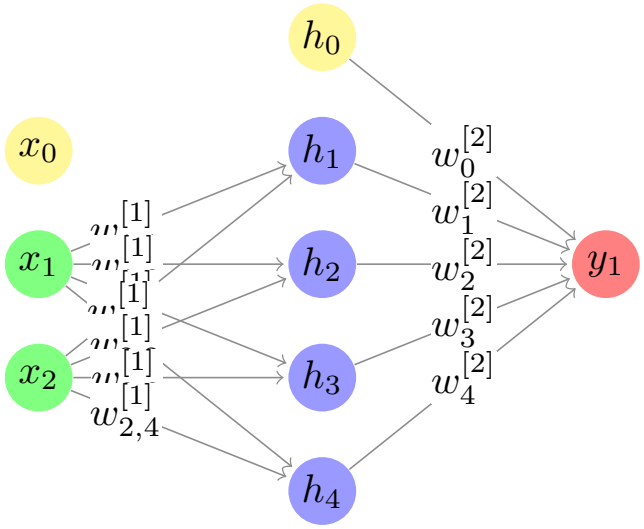
To demonstrate the neural network's learning mechanism, we will use a simple regression model that is described by the following linear relationship:

$$y = x_1^3 - 2x_2 \tag{3}$$

5 Neural Network Diagram

This section details the architecture of a simple neural network, which is diagrammed below (please ignore x0, h0, and w):

- **Input Layer:** Consists of 2 nodes, representing the input features (x_1, x_2).
- **Hidden Layer:** Comprises 4 nodes, which facilitate the learning of non-linear relationships.
- **Output Layer:** Contains a single node, which outputs the prediction y of the network.



x_i is the input data, $w_{i,j}$ is the weight in the hidden layer, W_i is the weight in the output layer, and y_1 demonstrates the output result.

6 Hyperparameters

Hyperparameters are configurations external to the model that influence how the network is structured and trained. Hyperparameters play a crucial role in determining the model's performance by affecting how quickly and effectively it learns from the training data.

Table 2: Detailed Explanation of Variables

Hyperparameter	**Description**
Number of hidden layers	1
Optimizer	Gradient descent backpropagation
Number of nodes in the hidden layer	4
Activation function of the hidden layer	ReLU function
Activation function of the output layer	Sigmoid function
Loss function	Root mean squared error
Learning rate	1
Number of epochs	1

7 Example Process: Mapping Function in Neural Network

Now we are going to work through the process how this neural network is trained with specific inputs and expected output. The inputs for this example are chosen with feature dimensions of 2 and 1, respectively.

Input Layer: The network receives a single training point with features:

- $x_1 = 2$
- $x_2 = 1$

These inputs are fed into the network to process through the neural architecture.

8 Forward Propagation

Forward Propagation involves processing input data through the network from the input to the output layer using current weights and biases, generating predictions that are used to calculate the error against actual targets.

1. **Input to Hidden Layer:**

   As indicated before, we have the input vector X as a 2x1 matrix $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, where $x_1 = 2$ and $x_2 = 1$

   - Here, the input vector $\mathbf{X} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ to the network consists of data points that we want the network to learn from (x_1 and x_2)

Let $\mathbf{W}_1 = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} \\ w_{1,3}^{[1]} & w_{2,3}^{[1]} \\ w_{1,4}^{[1]} & w_{2,4}^{[1]} \end{bmatrix}$, be the weight matrix connecting the input to the hidden layer

- $\mathbf{W}^{[1]}$ is the weight matrix associated with the first layer. Each element, such as $w_{1,1}$, represents the weight connecting the input node x_1 to the first neuron in the hidden layer.

Let $\mathbf{b}^{[1]} = \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \end{bmatrix}$, for the hidden layer.

- $\mathbf{b}^{[1]}$ represents the bias vector for the hidden layer, with each entry like $b_{1,1}$ adding a bias term to the corresponding neuron's output. This helps to adjust the threshold at which the neuron activates.

$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$

- This equation computes the linear combination of inputs and weights, adjusted by the bias. The result, $\mathbf{z}^{[1]}$, is the pre-activation output of the hidden layer.

$$\begin{bmatrix} z_{1,1} \\ z_{1,2} \\ z_{1,3} \\ z_{1,4} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} \\ w_{1,3}^{[1]} & w_{2,3}^{[1]} \\ w_{1,4}^{[1]} & w_{2,4}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \end{bmatrix}$$

$$\begin{bmatrix} z_{1,1} \\ z_{1,2} \\ z_{1,3} \\ z_{1,4} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This step shows the explicit matrix multiplication and addition for the given example. It is a practical computation where each neuron's input is the sum of products of each input feature and the corresponding weight plus a bias term.

$$\mathbf{z}^{[1]} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}$$

- The result is a vector of pre-activation values for each neuron in the hidden layer.

2. **Activation in Hidden Layer:**

$\mathbf{a}^{[1]} = \text{ReLU}(\mathbf{z}^{[1]})$

- The ReLU function is applied to each pre-activation value. This non-linear function outputs the input directly if it is positive; otherwise, it outputs zero.

$$\begin{bmatrix} a_{1,1} \\ a_{1,2} \\ a_{1,3} \\ a_{1,4} \end{bmatrix} = \text{ReLU}\left(\begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} \right)$$

- Since all inputs are positive (6), the ReLU output matches the input. This vector represents the activated output of the hidden layer.

$$\begin{bmatrix} z_{2,1} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} \\ w_{1,3}^{[1]} & w_{2,3}^{[1]} \\ w_{1,4}^{[1]} & w_{2,4}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \end{bmatrix}$$

$$\begin{bmatrix} z_{1,1} \\ z_{1,2} \\ z_{1,3} \\ z_{1,4} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{a}^{[1]} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}$$

- These are the activation values from the hidden layer that will be fed to the next layer.

3. **Hidden Layer to Output Layer:**

   Let $\mathbf{W}^{[2]} = \begin{bmatrix} w_{1,1}^{[2]} & w_{1,2}^{[2]} & w_{1,3}^{[2]} & w_{1,4}^{[2]} \end{bmatrix}$ and $\mathbf{b}^{[2]} = \begin{bmatrix} b_{2,1} \end{bmatrix}$

   - $\mathbf{W}^{[2]}$ and $\mathbf{b}^{[2]}$ are the weight vector and bias for the output layer. Here, we're transitioning from a hidden layer with multiple neurons to an output layer with potentially one neuron.

   $\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$

   - This equation calculates the linear combination of the activated outputs from the hidden layer, weighed by $\mathbf{W}^{[2]}$, and adjusted by the bias $\mathbf{b}^{[2]}$. The result is the input to the output layer's activation function.

$$\begin{bmatrix} z_{2,1} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[2]} & w_{1,2}^{[2]} & w_{1,3}^{[2]} & w_{1,4}^{[2]} \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} + \begin{bmatrix} b_{2,1} \end{bmatrix}$$

$$\begin{bmatrix} z_{2,1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix}$$

$\mathbf{z}^{[2]} = \begin{bmatrix} 12 \end{bmatrix}$

4. **Activation in Output Layer:**

   $a^{[2]} = \hat{y} = \sigma(\mathbf{z}^{[2]})$

   - The sigmoid function σ is used at the output layer to map the input value into a (0,1) range, which is typical for binary classification tasks or probability estimation.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

   $\hat{y} = \sigma(\begin{bmatrix} 12 \end{bmatrix}) = 0.99$

   - Given the high input value of 12, the sigmoid function outputs a value close to 1, indicating a high confidence level in the positive class, assuming a binary classification context.

9 Backward Propogation

Backward Propagation adjusts the network's parameters by calculating the loss function's gradient and updating weights and biases to minimize prediction errors, optimizing network performance over training iterations.

1. **Calculate Loss:**

   $y_i = x_1^3 - 2x_2$
   $y_i = 2^3 - 2$
   $y_i = 6$
   $$E = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2}$$
   $E = \sqrt{(0.99-6)^2} = \sqrt{25.1001}$
   $E = 5.01$

   - This formula calculates the Root Mean Squared Error (RMSE) between the predicted values ($\hat{y}_i$) and the actual values (y_i). Here, $\hat{y}_i = 0.99$ and $y_i = 6$.
   - The RMSE provides a measure of how well the model is predicting the output, quantifying the difference in terms of the model's accuracy. In this case, an RMSE of 11.01 indicates a significant error, showing that the model's prediction is far from the actual value.

2. **Output to Hidden Layer:**

   Using the chain rule, we first calculate the gradient of the loss function with respect to the output predictions:
   $$\frac{\partial L}{\partial \hat{y}} = \frac{2}{m}(\hat{y} - y)\frac{1}{2\sqrt{\text{mean squared error}}}$$
   For the next layer's pre-activation output, we derive the gradient with respect to the sigmoid function:
   $$\frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \sigma'(z^{[2]})$$

   - $\sigma'(z^{[2]})$ is the derivative of the sigmoid activation function, applied to the pre-activation outputs at the output layer.

3. **Hidden Layer to Input Layer:**

   The gradients of the weights and biases are calculated as follows:
   $$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \cdot a^{[1]}$$
   $$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}}$$
   For the activations of the previous layer, we use the transpose of the weights:
   $$\frac{\partial L}{\partial a^{[1]}} = W^{[2]T} \cdot \frac{\partial L}{\partial z^{[2]}}$$
   The derivative through the ReLU function is computed next:
   $$\frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[1]}} \cdot \text{ReLU}'(z^{[1]})$$

   - $\text{ReLU}'(z^{[1]})$ is the derivative of the ReLU function, which is 1 for positive inputs and 0 otherwise.

4. **Update Parameters:**

   Finally, we update the weights and biases using the calculated gradients and a learning rate α:
   $$W^{[l]} = W^{[l]} - \alpha \cdot \frac{\partial L}{\partial W^{[l]}}$$
   $$b^{[l]} = b^{[l]} - \alpha \cdot \frac{\partial L}{\partial b^{[l]}}$$

- These updates adjust the weights and biases to minimize the loss, thereby improving the model with each iteration.

10 References