Lecture 5a - Calculus Refresher & Forward Pass

Contents

7 Calculus Refresher & Forward Pass **1**

7.1 Recap . . . 1

7.2 Calculus Refresher . . . 2

7.2.1 Basic Rules of Calculus (Single-variable and Multi-variable) . . . 2

7.2.2 Multi-variable case . . . 3

7.2.3 Chain Rule . . . 3

7.2.4 Distributed Chain Rule: Influence Diagram . . . 4

7.2.5 Example . . . 5

7.3 Forward Pass . . . 6

7.3.1 Input Layer . . . 6

7.3.2 First Hidden Layer (Layer 1) . . . 7

7.3.3 Intermediate Hidden Layers . . . 8

7.3.4 Output Layer . . . 9

7.4 Example of Forward Pass . . . 10

7.4.1 First Hidden Layer Calculations . . . 10

7.4.2 Second Hidden Layer Calculations . . . 11

7.4.3 Output Layer Calculations . . . 11

7.4.4 Calculating the Loss . . . 12

7 Calculus Refresher & Forward Pass

7.1 Recap

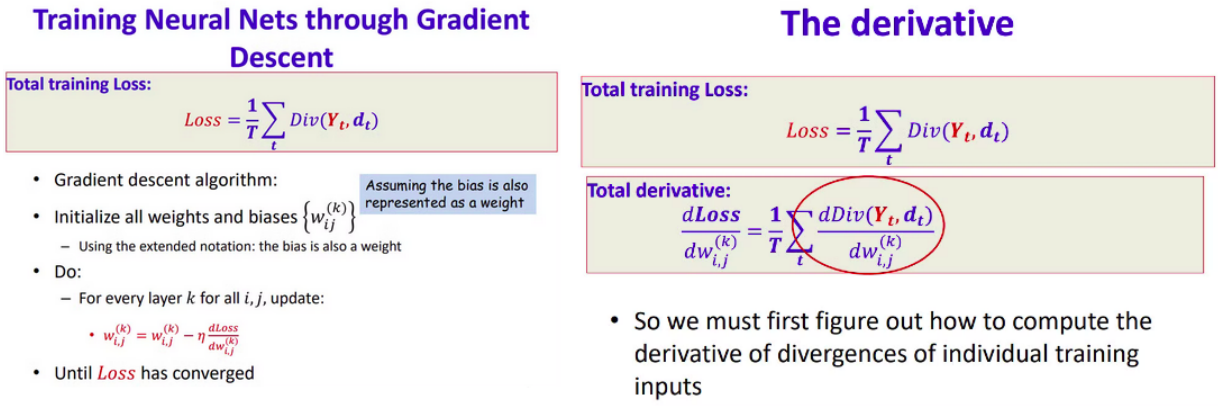
- The problem of training neural networks can be framed as Empirical Risk Minimization (ERM).
- Expected Risk is the shaded area in the plot, representing the average loss over the entire data distribution. In practice, we cannot compute this directly, so we approximate it using the `Empirical Risk`, which is the average loss over a set of training samples:

$$Loss(W) = \frac{1}{T}\sum_{i} div(Y_i, d_i) \tag{1}$$

- Where the divergence on the i-th instance is $div(Y_i, d_i)$ i.e. $Y_i = f(X_i; W)$
- Empirical Risk Minimization: If we minimize the empirical risk (loss over the training samples), we hope that the model generalizes well, meaning it will also minimize the expected

risk (loss over unseen data). In simpler terms, by fitting the model on the training data, we expect it to perform well on new data.

- The loss function measures how far the model's predictions are from the true labels. Our goal is to minimize the loss function.
- The gradient is just a vector comprised of partial derivatives, so for each parameter we're going to take a step against the loss wrt that parameter
- To minimize the loss function, we use the Gradient Descent algorithm. In this algorithm, we iteratively update the parameters of the model by taking small steps in the direction opposite to the gradient of the loss function with respect to the parameters. The gradient points in the direction of the steepest ascent, so moving in the opposite direction leads to minimizing the loss.
- Initialization and Iteration: We initialize the weights randomly and iteratively update them by computing the gradient of the loss with respect to the parameters. We continue this process until the values stop changing, i.e., when convergence is achieved.
- The derivative of the loss wrt any parameter is the average of the derivatives of the losses of the individual training instances wrt that parameter
- So this means we must estimate this circle thing - the derivative of the loss of individual training instances wrt these parameters. If we can compute these we can compute the average derivative and perform the gradient descent



7.2 Calculus Refresher

7.2.1 Basic Rules of Calculus (Single-variable and Multi-variable)

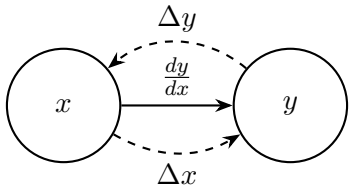
Single-variable case

For any differentiable function $y = f(x)$, the derivative is represented by $\frac{dy}{dx}$. A small change in x (denoted as Δx) leads to a corresponding small change in y (denoted as Δy). This relationship is

approximated by:

$$\Delta y \approx \frac{dy}{dx} \cdot \Delta x$$

The diagram shows an "influence" diagram where changes in x influence changes in y, represented by a simple line with $x \to y$.

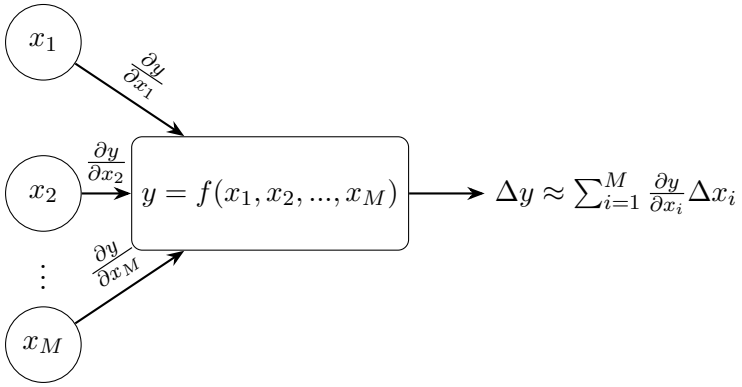


7.2.2 Multi-variable case

For a function of multiple variables, $y = f(x_1, x_2, ..., x_M)$, the derivative involves partial derivatives with respect to each variable. A small change in the multiple variables $\Delta x_1, \Delta x_2, ..., \Delta x_M$ leads to a change in y, approximated by the sum of partial derivatives:

$$\Delta y \approx \frac{\partial y}{\partial x_1}\Delta x_1 + \frac{\partial y}{\partial x_2}\Delta x_2 + \cdots + \frac{\partial y}{\partial x_M}\Delta x_M$$

The diagram visually shows each variable influencing y, with arrows showing partial derivatives.

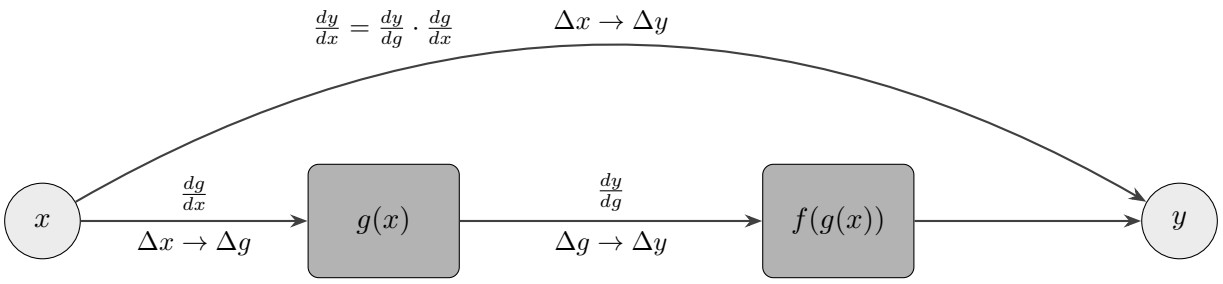


7.2.3 Chain Rule

For a nested function $y = f(g(x))$, the chain rule is used to differentiate. The rule states:

$$\frac{dy}{dx} = \frac{dy}{dg(x)} \cdot \frac{dg(x)}{dx}$$

The diagram shows how x affects g, which in turn affects y. The changes propagate from $\Delta x \to \Delta g \to \Delta y$.

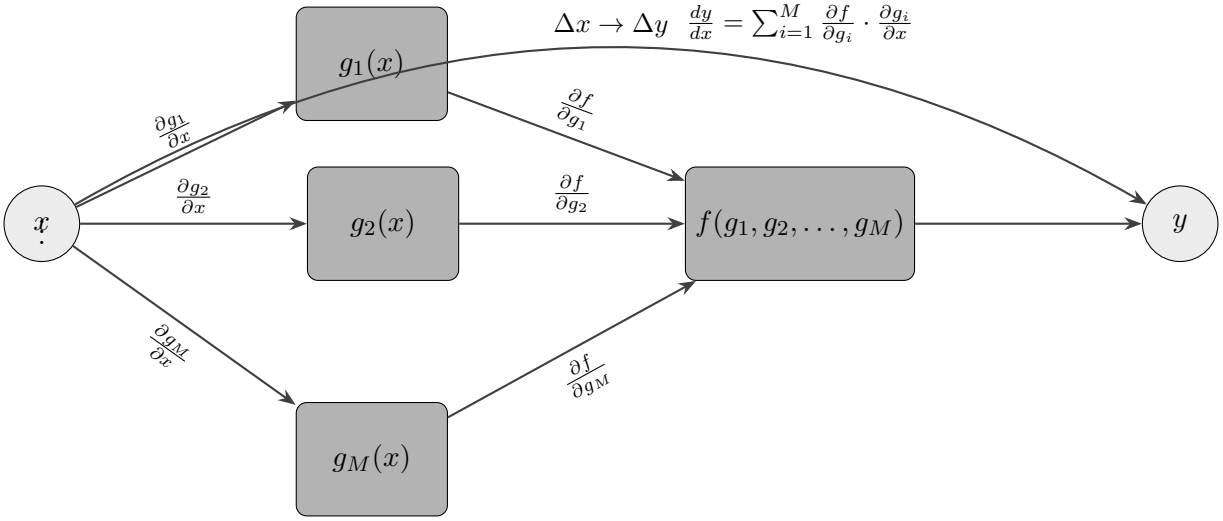


7.2.4 Distributed Chain Rule: Influence Diagram

This extends the chain rule to multiple nested functions. For a function $y = f(g_1(x), g_2(x), ..., g_M(x))$, each nested function $g_i(x)$ depends on x, and each g_i influences y. The derivative rule in this case applies to each chain separately, propagating the changes from x through all intermediate functions $g_1(x), g_2(x), ..., g_M(x)$ to influence y.

$$y = f(g_1(x), g_2(x), ..., g_M(x))$$

The diagram shows a network of influence where x affects $g_1, g_2, ..., g_M$, and these in turn affect y. Arrows indicate the flow of influence and derivative propagation.



Where:

$$\begin{aligned}
\frac{dy}{dx} &= \sum_{i=1}^{M} \frac{\partial f}{\partial g_i} \cdot \frac{\partial g_i}{\partial x} \\
&= \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x} + \frac{\partial f}{\partial g_3} \cdot \frac{\partial g_3}{\partial x} + \cdots \qquad (2) \\
&= \left(\frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x}\right) + \left(\frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x}\right) + \left(\frac{\partial f}{\partial g_3} \cdot \frac{\partial g_3}{\partial x}\right) + \cdots + \left(\frac{\partial f}{\partial g_M} \cdot \frac{\partial g_M}{\partial x}\right)
\end{aligned}$$

7.2.5 Example

In the distributed chain rule, f is the outer function that depends on multiple intermediate functions. Specifically:

- f is a function that takes multiple inputs: $g_1(x), g_2(x), \ldots, g_m(x)$

So we can write f as:

$$y = f(g_1(x), g_2(x), \ldots, g_m(x))$$

Here's a breakdown of what this means:

- x is the initial input variable.
- $g_1, g_2, \ldots, g_m$ are intermediate functions, each depending on x.
- f is a function that takes the outputs of all these g functions as its inputs.
- y is the final output of this composite function.

For example, f could be any multivariate function. Let's consider a simple case to illustrate: Suppose we have:

$$g_1(x) = x^2$$
$$g_2(x) = \sin(x)$$
$$f(u, v) = u + v$$

Then our composite function would be:

$$y = f(g_1(x), g_2(x)) = f(x^2, \sin(x)) = x^2 + \sin(x)$$

In this case:

$$\frac{\partial f}{\partial g_1} = \frac{\partial f}{\partial u} = 1$$
$$\frac{\partial f}{\partial g_2} = \frac{\partial f}{\partial v} = 1$$
$$\frac{\partial g_1}{\partial x} = 2x$$
$$\frac{\partial g_2}{\partial x} = \cos(x)$$

Applying the distributed chain rule:

$$\begin{aligned}\frac{dy}{dx} &= \left(\frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial x}\right) + \left(\frac{\partial f}{\partial g_2} \cdot \frac{\partial g_2}{\partial x}\right) \\ &= (1 \cdot 2x) + (1 \cdot \cos(x)) \\ &= 2x + \cos(x)\end{aligned}$$

Intuitively, $\frac{dy}{dx}$ represents the rate at which the output y changes with respect to a small change in the input variable x. It captures how the combined effects of the intermediate functions g_1 and g_2 influence the final output y as x varies.

7.3 Forward Pass

Within each neuron there are 2 operations going on:

1. First you compute an affine function of all of the inputs.
2. Then we put it through an activation

Convention

- Superscripts refer to the Layer number.
  - (1): Refers to the first hidden layer in the neural network.
  - (0): Refers to the input layer (layer before the first hidden layer).
- In Subscript i, j is used:
  - i: Refers to the index of the input or the previous layer's neuron.
  - j: Refers to the index of the current layer's neuron.
- $z_2^{(2)}$ this is the affine term for the second neuron in the second layer.
- $y_2^{(1)}$ Is the output of the activation of the second neuron of the first layer
- The output by the network is Y and the desired output is d, and we compute the divergence between the 2.
- Our job is to compute the derivative of the divergence wrt all the parameters of the network

Convention

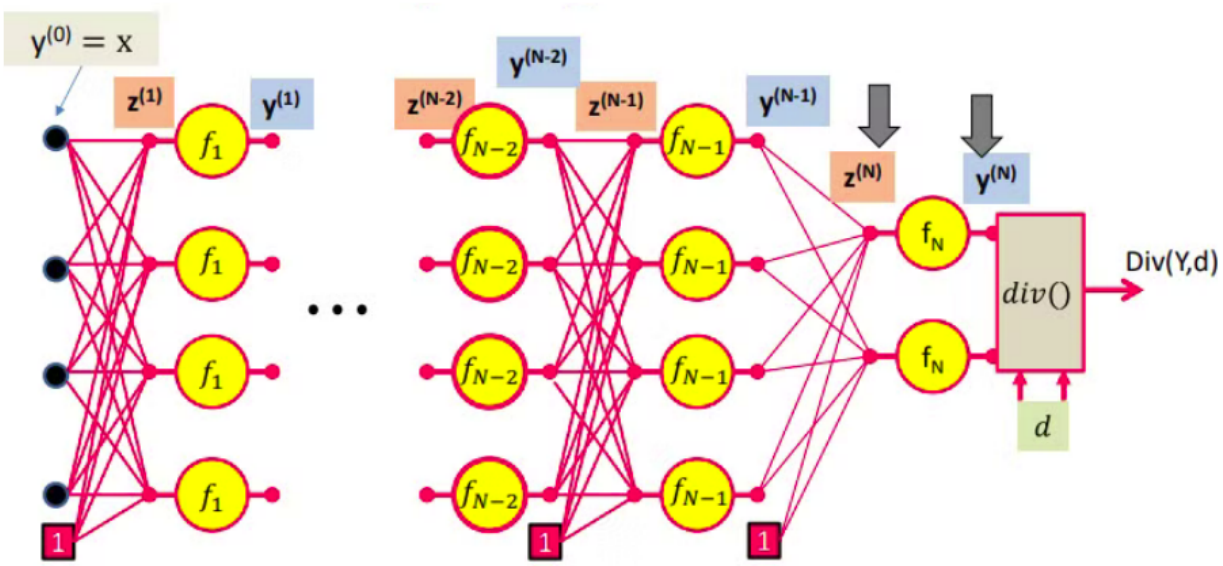
$w_{i,j}^{(1)}$: This refers to the weight connecting the i-th neuron in the previous layer (or input in the input layer) to the j-th neuron in the first hidden layer.

$y_0 = 1$ is chosen because it provides a constant bias input that makes the neural network more flexible in fitting data, and it simplifies both the mathematical formulation and learning process.

7.3.1 Input Layer

$$y^{(0)} = x \tag{3}$$

This is the input to the network, denoted as $y^{(0)}$, where x is the initial input vector to the neural network.



7.3.2 First Hidden Layer (Layer 1)

Pre-Activation:

$$z_1^{(1)} = w_{i,1}^{(1)} y_i^{(0)} + b^{(1)} \tag{4}$$

The equation $z_1^{(1)} = w_{i,1}^{(1)} y_i^{(0)} + b^{(1)}$ represents the linear combination of the input $y_i^{(0)}$ weighted by $w_{i,1}^{(1)}$ and adjusted by the bias term to compute the input to the first neuron of the current layer (first hidden layer).
In other words $z_1^{(1)}$ is the pre-activation value (or net input) of the first neuron in the first hidden layer. It is calculated by taking the weighted sum of inputs. We can re-write it as:

$$z_1^{(1)} = \sum_i (w_{i,1}^{(1)} y_i^{(0)}) + b^{(1)} \tag{5}$$

we can generalize the equation using summation to represent the pre-activation value for the $j-th$ neuron in the $l-th$ layer of a neural network. The generalized equation is:

$$z_j^{(l)} = \sum_{i=1}^{n} (w_{i,j}^{(l)} y_i^{(l-1)}) + b^{(l)} \tag{6}$$

Activation: Once the z values are computed, the activations can be applied to determine the output y of each neuron in the layer.

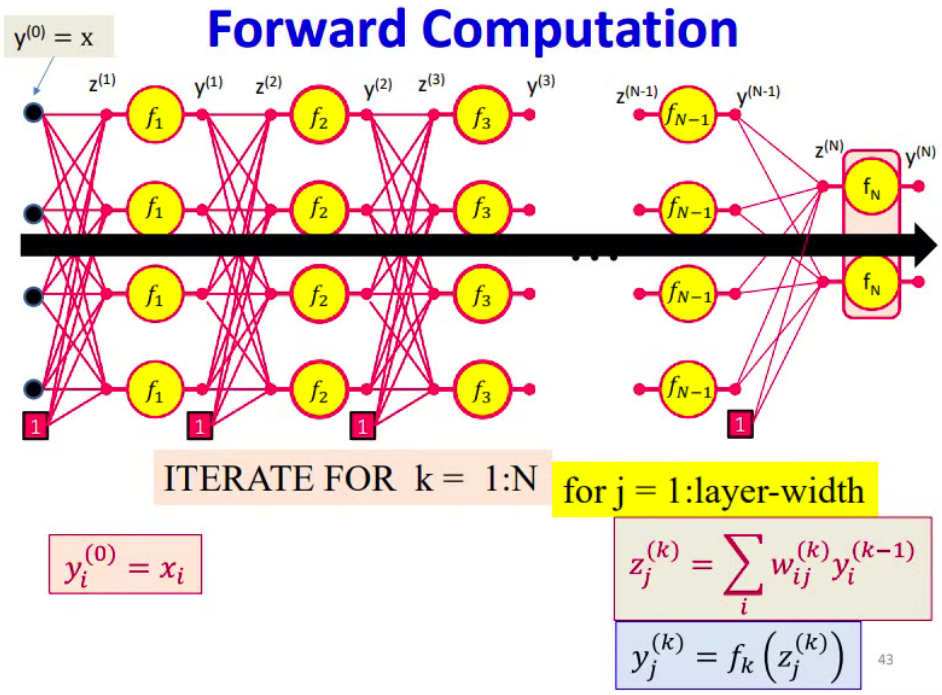
$$y^{(1)} = f_1(z^{(1)}) \tag{7}$$

Here, f_1 is the activation function applied element-wise on $z^{(1)}$.
The equation for the output y of the $j-th$ neuron in the $l-th$ layer, after applying the activation function, can be represented as:

$$y_j^{(l)} = f(z_j^{(l)}) \tag{8}$$

Where, $z_j^{(l)}$ is the Pre-activation value for the $j-th$ neuron in the $l-th$ layer, computed from the weighted sum of inputs and bias.

7.3.3 Intermediate Hidden Layers



For intermediate hidden layers in a neural network, we can generalize the formulas for the pre-activation values z and the outputs y of the $j-th$ neuron in the $l-th$ layer as follows:

$$z_j^{(l)} = \sum_{i=1}^{n} w_{i,j}^{(l)} y_i^{(l-1)} + b^{(l)} \tag{9}$$

$$y_j^{(l)} = f_l(z_j^{(l)}) \tag{10}$$

Where: **Pre-activation** $z_j^{(l)}$:

- $z_j^{(l)}$: Pre-activation value for the j-th neuron in the l-th hidden layer.
- $w_{i,j}^{(l)}$: Weight connecting the i-th neuron from the previous layer $(l-1)$ to the j-th neuron in the current layer (l).
- $y_i^{(l-1)}$: Output of the i-th neuron from the previous layer $(l-1)$.
- $b^{(l)}$: Bias term for the j-th neuron in the l-th layer.
- n: Number of neurons in the previous layer $(l-1)$.

Output $y_j^{(l)}$:

- $y_j^{(l)}$: Output of the j-th neuron in the l-th hidden layer.
- f_l: Activation function applied in the l-th layer (can vary for different layers).

Forward "Pass"

- Input: D dimensional vector $\mathbf{x} = [x_j,\ j = 1 \ldots D]$
- Set:
  - $D_0 = D$, is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j,\ j = 1 \ldots D;$ $\quad y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$ D_k is the size of the kth layer
    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k\left(z_j^{(k)}\right)$
- Output:
  - $Y = y_j^{(N)}, j = 1..D_N$

44

7.3.4 Output Layer

1. Pre-Activation Value The pre-activation value for the j-th neuron in the output layer (L) is calculated as:

$$z_j^{(L)} = \sum_{i=1}^{n} w_{i,j}^{(L)} y_i^{(L-1)} + b^{(L)}$$

2. Activation Function The output of the j-th neuron in the output layer after applying the activation function is given by:

$$y_j^{(L)} = f_L(z_j^{(L)})$$

Common activation functions include:

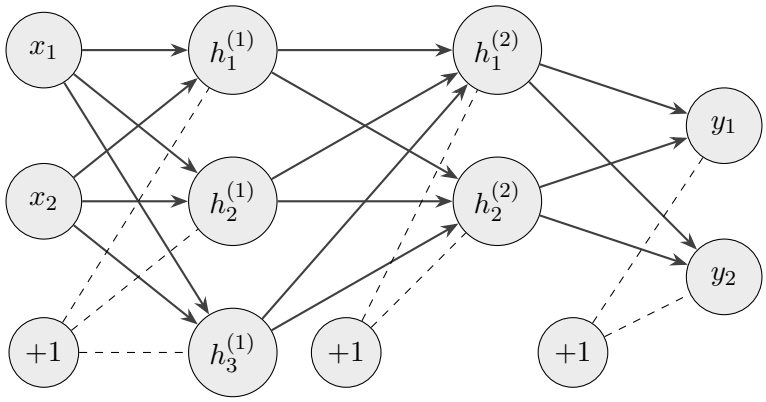
- Softmax for multi-class classification.
- Sigmoid for binary classification.

3. Loss Function (Divergence) The cross-entropy loss for measuring divergence between predicted outputs and true labels is given by:

$$L = -\sum_{j=1}^{C} y_{\text{true},j} \log\left(y_j^{(L)}\right)$$

Where C is the number of classes.

7.4 Example of Forward Pass



Inputs = 2, First Hidden Layer = 3 neurons, Second Hidden Layer = 2 Neurons, Output Layer = 2 neurons

7.4.1 First Hidden Layer Calculations

For the first hidden layer, each neuron j computes:

$$z_j^{(1)} = \sum_{i=1}^{n} w_{i,j}^{(1)} x_i + b^{(1)} \quad (j = 1, 2, 3)$$

Where: - x_i are the inputs to the network.

Assuming you have n inputs $x_1, x_2, \ldots, x_n$ and specific weights and biases, let's say:

$$w^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.3 & 0.5 & 0.7 \end{bmatrix} \quad \text{(for 2 inputs and 3 neurons)}$$

$$b^{(1)} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}$$

Assuming the input $x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$:

1. For $j = 1$:
$$z_1^{(1)} = (0.2 \cdot 1 + 0.3 \cdot 2) + 0.1 = 0.2 + 0.6 + 0.1 = 0.9$$

2. For $j = 2$:
$$z_2^{(1)} = (0.4 \cdot 1 + 0.5 \cdot 2) + 0.2 = 0.4 + 1.0 + 0.2 = 1.6$$

3. For $j = 3$:
$$z_3^{(1)} = (0.6 \cdot 1 + 0.7 \cdot 2) + 0.3 = 0.6 + 1.4 + 0.3 = 2.3$$

Now apply the activation function f_1: - Assume $f_1(z) = \text{ReLU}(z)$, which gives $y_j^{(1)} = \max(0, z_j^{(1)})$. So,

- $y_1^{(1)} = \max(0, 0.9) = 0.9$
- $y_2^{(1)} = \max(0, 1.6) = 1.6$
- $y_3^{(1)} = \max(0, 2.3) = 2.3$

7.4.2 Second Hidden Layer Calculations

Now move to the second hidden layer, with weights $w^{(2)}$ and biases $b^{(2)}$.
Assume:

$$w^{(2)} = \begin{bmatrix} 0.5 & 0.3 \\ 0.4 & 0.6 \\ 0.7 & 0.2 \end{bmatrix} \quad \text{(for 3 inputs and 2 neurons)}$$

$$b^{(2)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

1. For $j = 1$:

$$z_1^{(2)} = (0.5 \cdot 0.9 + 0.4 \cdot 1.6 + 0.7 \cdot 2.3) + 0.1 = 0.45 + 0.64 + 1.61 + 0.1 = 2.8$$

2. For $j = 2$:

$$z_2^{(2)} = (0.3 \cdot 0.9 + 0.6 \cdot 1.6 + 0.2 \cdot 2.3) + 0.2 = 0.27 + 0.96 + 0.46 + 0.2 = 1.89$$

Apply the activation function f_2 (assuming it is also ReLU):

- $y_1^{(2)} = \max(0, 2.8) = 2.8$
- $y_2^{(2)} = \max(0, 1.89) = 1.89$

7.4.3 Output Layer Calculations

Now compute for the output layer with weights $w^{(3)}$ and biases $b^{(3)}$.
Assume:

$$w^{(3)} = \begin{bmatrix} 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \quad \text{(for 2 inputs and 2 neurons)}$$

$$b^{(3)} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

1. For $j = 1$:

$$z_1^{(3)} = (0.3 \cdot 2.8 + 0.5 \cdot 1.89) + 0.1 = 0.84 + 0.945 + 0.1 = 1.885$$

2. For $j = 2$:

$$z_2^{(3)} = (0.4 \cdot 2.8 + 0.6 \cdot 1.89) + 0.2 = 1.12 + 1.134 + 0.2 = 2.454$$

Finally, apply the activation function f_3 (assuming it is a sigmoid). Simoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{11}$$

- $y_1^{(3)} = \frac{1}{1+e^{-1.885}} \approx 0.868$
- $y_2^{(3)} = \frac{1}{1+e^{-2.454}} \approx 0.921$

The sum of probabilities in coming out to be greater than 1, which is why we usually apply a softmax instead of sigmoid. it's common to use a softmax activation function for the output layer instead of the sigmoid function. The softmax function normalizes the output to ensure that the probabilities sum to 1, which is particularly important for multi-class classification tasks. Softmax Function:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad \text{for } j = 1, \ldots, K \tag{12}$$

Calculating the output probabilities:

$$e^{z_1^{(3)}} \approx e^{1.885} \approx 6.588$$
$$e^{z_2^{(3)}} \approx e^{2.454} \approx 11.661$$

Sum:

$$\text{sum} \approx 6.588 + 11.661 \approx 18.249$$

Softmax outputs:

$$y_1^{(3)} = \frac{6.588}{18.249} \approx 0.361$$
$$y_2^{(3)} = \frac{11.661}{18.249} \approx 0.639$$

7.4.4 Calculating the Loss

The cross-entropy loss is given by:

$$L = -\sum_{i=1}^{C} y_i \log(p_i)$$

For our case with two classes:

$$L = -\left(y_1 \log(p_1) + y_2 \log(p_2)\right)$$

If the true label for the first class is $y_1 = 1$ (indicating that it is the correct class), and for the second class, $y_2 = 0$, the cross-entropy loss can be calculated as follows - Substituting the values:

$$L = -\left(1 \cdot \log(0.361) + 0 \cdot \log(0.639)\right) = -\log(0.361) \approx 1.018$$