Lecture 5b - Backward Pass

Contents

9 Backpropagation

We have computed intermediate values in the forward computation. We need to remember these values which we will use to compute the derivatives. The last section in 5a was for a single training instance.

Backward Pass

- The goal is to compute the gradients of the loss function with respect to **each** parameter of the network. This process is carried out by propagating gradients from the output layer back to the input layer, applying the chain rule.
- We begin by computing the derivative of the divergence with respect to the network's output. The divergence measures the difference between the network's predicted output and the desired output. Our goal is to determine how perturbing the network's output Y will affect the divergence — that is, we calculate the sensitivity of the divergence with respect to changes in Y.
- To achieve this, we propagate gradients backward, starting from the output layer. First, we assess how perturbing the affine terms at the output layer influences the divergence. Next, we move further back to examine how changes in the weights affect the divergence. Finally, we continue back through the network, determining how perturbing the outputs of the previous layers modifies the overall divergence. This process is repeated layer by layer until we've traced the effects back to the input.

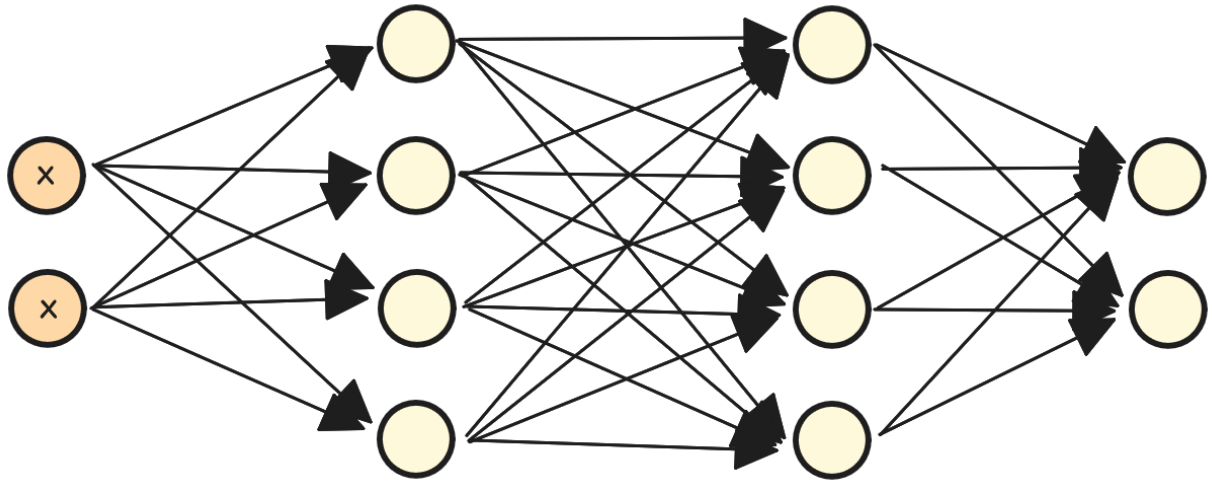


Figure 1: Architecture of NN used in this Lecture

Figure 1 shows the architecture of the Neural Network used in this lecture. The network has 2 inputs, 2 hidden layers with 4 neurons in each of the hidden layer, and 2 neurons in the output layer. Since Each neuron computes 2 operation which are: 1. Affine Function, 2. Activation Function. So we can redraw the above architecture as the one below.

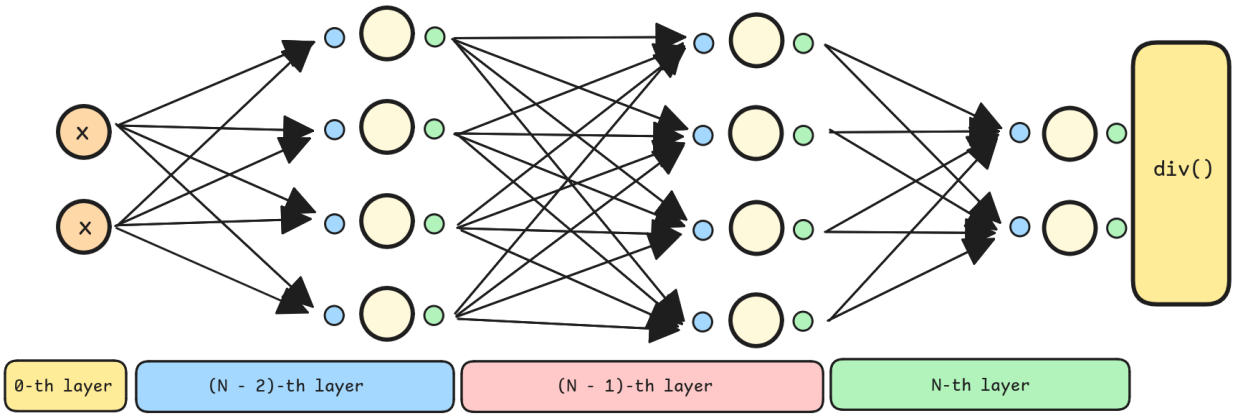


Figure 2: Redrawn architecture of figure 1

Figure 2 is same as figure 1 but it shows the pre-activation terms z and post activation terms y separately. Now for backward pass, we propagate gradients backward, starting from the output layer.

9.1 Gradient of the Divergence with Respect to the Output

The first step is to compute the gradient of the divergence $\mathcal{D}$ with respect to the network's output Y:

$$\frac{\partial \mathcal{D}}{\partial y_i} \quad \text{for each output neuron } i$$

9.2 Gradient with Respect to the Pre-Activation Terms of the Output Layer

Let $Z^{(L)}$ be the pre-activation (affine) terms of the output layer. The goal is to compute the gradient of the divergence with respect to each pre-activation term $z_i^{(L)}$:

$$\frac{\partial \mathcal{D}}{\partial z_i^{(N)}} = \frac{\partial \mathcal{D}}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_i^{(N)}}$$

where $\frac{\partial y_i}{\partial z_i^{(N)}}$ is the derivative of the activation function with respect to the pre-activation terms $z_i^{(N)}$.

9.3 Gradient with Respect to Weights and Biases of the Output Layer

9.3.1 For the weights $w_{ij}^{(N)}$:

$$\frac{\partial \mathcal{D}}{\partial w_{ij}^{(N)}} = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}} \cdot \frac{dz_j^{(N)}}{dw_{ij}^{(N)}}$$

When taking the derivative of $z_j^{(N)}$ with respect to the weight $w_{ij}^{(N)}$, you focus on how $z_j^{(N)}$ changes as you change $w_{ij}^{(N)}$:

$$\frac{dz_j^{(N)}}{dw_{ij}^{(N)}} = \frac{d}{dw_{ij}^{(N)}} \left(\sum_k w_{kj}^{(N)} y_k^{(N-1)} + b_j^{(N)} \right)$$

Here, the only term that depends on $w_{ij}^{(N)}$ is the one involving $y_i^{(N-1)}$.
So The derivative of $z_j^{(N)}$ with respect to $w_{ij}^{(N)}$ results in:

$$\frac{dz_j^{(N)}}{dw_{ij}^{(N)}} = y_i^{(N-1)}$$

So: where $y_j^{(N-1)}$ is the activation from the previous layer.

Formula for the weights $w_{ij}^{(N)}$:

$$\frac{\partial \mathcal{D}}{\partial w_{ij}^{(N)}} = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}} \cdot y_j^{(N-1)}$$

9.3.2 For the biases $b_i^{(N)}$:

The derivative of the loss $\mathcal{D}$ with respect to the bias $b_i^{(N)}$ can be computed using the chain rule. Since the bias directly affects $z_i^{(N)}$ and no other terms:

$$\frac{\partial \mathcal{D}}{\partial b_i^{(N)}} = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}} \cdot \frac{\partial z_i^{(N)}}{\partial b_i^{(N)}}$$

$$\frac{\partial b_i^{(N)}}{\partial \mathcal{D}} = \frac{\partial z_i^{(N)}}{\partial \mathcal{D}} \cdot \frac{\partial b_i^{(N)}}{\partial z_i^{(N)}}$$

Derivative of $z_i^{(N)}$ with respect to $b_i^{(N)}$: The pre-activation term $z_i^{(N)}$ depends linearly on the bias, and

$$\frac{\partial z_i^{(N)}}{\partial b_i^{(N)}} = 1 \quad \text{or equivalently} \quad \frac{\partial b_i^{(N)}}{\partial z_i^{(N)}} = 1$$

This is because $b_i^{(N)}$ appears as a simple additive constant in the expression for $z_i^{(N)}$.

So:

$$\frac{\partial \mathcal{D}}{\partial b_i^{(N)}} = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}} \cdot 1 = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}}$$

$$\frac{\partial b_i^{(N)}}{\partial \mathcal{D}} = \frac{\partial z_i^{(N)}}{\partial \mathcal{D}} \cdot 1 = \frac{\partial z_i^{(N)}}{\partial \mathcal{D}}$$

Formula for the biases $b_i^{(N)}$:

$$\frac{\partial \mathcal{D}}{\partial b_i^{(N)}} = \frac{\partial \mathcal{D}}{\partial z_i^{(N)}}$$

Before Starting section 9.4 please see images in the final 3 pages of these notes

9.4 Example Numerical

We have derived all formulas for the backward pass. Now we will be doing a complete forward pass and Backward pass (Backpropagation) on the Neural Network shown in Figure 1. It has 2 inputs, 4 neurons in hidden layer 1, 4 neurons in hidden layer 2 and 2 neurons in the output layer. The Loss Function used will be cross entropy and ReLU will be used as the activation function for hidden layers.

9.4.1 Forward Pass

Architecture

- 2 input features (x_1, x_2)
- 4 neurons in the first hidden layer ($(n-2)$-th layer)
- 4 neurons in the second hidden layer ($(n-1)$-th layer)
- 2 output neurons (output layer, n-th layer)
- Weights and biases for each layer

Input layer (inputs: x_1 and x_2):

Let the input data be:

$$x_1 = 0.5, \quad x_2 = 0.8$$

First Hidden Layer ($(n-2)$-th layer):

Let the weights $w_{ij}^{(n-2)}$ and biases $b_i^{(n-2)}$ be **(These are randomly initialized)**:

$$\begin{aligned} w_{11}^{(n-2)} &= 0.2, \quad w_{12}^{(n-2)} = -0.3 \\ w_{21}^{(n-2)} &= 0.4, \quad w_{22}^{(n-2)} = 0.1 \\ w_{31}^{(n-2)} &= -0.5, \quad w_{32}^{(n-2)} = 0.2 \\ w_{41}^{(n-2)} &= 0.6, \quad w_{42}^{(n-2)} = -0.7 \end{aligned}$$

Biases:

$$\begin{aligned} b_1^{(n-2)} &= 0.1, \quad b_2^{(n-2)} = -0.1, \\ b_3^{(n-2)} &= 0.05, \quad b_4^{(n-2)} = 0.2 \end{aligned}$$

For each neuron in the first hidden layer:

$$\begin{aligned} z_1^{(n-2)} &= w_{11}^{(n-2)} \cdot x_1 + w_{12}^{(n-2)} \cdot x_2 + b_1^{(n-2)} = 0.2 \cdot 0.5 + (-0.3) \cdot 0.8 + 0.1 = 0.01 \\ z_2^{(n-2)} &= w_{21}^{(n-2)} \cdot x_1 + w_{22}^{(n-2)} \cdot x_2 + b_2^{(n-2)} = 0.4 \cdot 0.5 + 0.1 \cdot 0.8 + (-0.1) = 0.3 \\ z_3^{(n-2)} &= w_{31}^{(n-2)} \cdot x_1 + w_{32}^{(n-2)} \cdot x_2 + b_3^{(n-2)} = (-0.5) \cdot 0.5 + 0.2 \cdot 0.8 + 0.05 = -0.06 \\ z_4^{(n-2)} &= w_{41}^{(n-2)} \cdot x_1 + w_{42}^{(n-2)} \cdot x_2 + b_4^{(n-2)} = 0.6 \cdot 0.5 + (-0.7) \cdot 0.8 + 0.2 = -0.16 \end{aligned}$$

Apply an activation function, such as ReLU (Rectified Linear Unit):

$$y_i^{(n-2)} = \max(0, z_i^{(n-2)})$$

Thus, the activations are:

$$\begin{aligned} y_1^{(n-2)} &= \max(0, 0.01) = 0.01 \\ y_2^{(n-2)} &= \max(0, 0.3) = 0.3 \\ y_3^{(n-2)} &= \max(0, -0.06) = 0 \\ y_4^{(n-2)} &= \max(0, -0.16) = 0 \end{aligned}$$

Second Hidden Layer ($(n-1)$-th layer):

Let the weights $w_{ij}^{(n-1)}$ and biases $b_i^{(n-1)}$ be:

$$\begin{aligned} w_{11}^{(n-1)} = 0.1, \quad w_{12}^{(n-1)} = 0.4, \quad w_{13}^{(n-1)} = -0.2, \quad w_{14}^{(n-1)} = 0.3 \\ w_{21}^{(n-1)} = -0.3, \quad w_{22}^{(n-1)} = 0.5, \quad w_{23}^{(n-1)} = 0.2, \quad w_{24}^{(n-1)} = -0.4 \\ w_{31}^{(n-1)} = 0.2, \quad w_{32}^{(n-1)} = -0.1, \quad w_{33}^{(n-1)} = 0.3, \quad w_{34}^{(n-1)} = 0.1 \\ w_{41}^{(n-1)} = -0.4, \quad w_{42}^{(n-1)} = 0.2, \quad w_{43}^{(n-1)} = 0.1, \quad w_{44}^{(n-1)} = -0.2 \end{aligned}$$

Biases:

$$\begin{aligned} b_1^{(n-1)} = 0.05, \quad b_2^{(n-1)} = -0.05, \\ b_3^{(n-1)} = 0.02, \quad b_4^{(n-1)} = -0.01 \end{aligned}$$

For each neuron in the second hidden layer:

$$\begin{aligned} z_1^{(n-1)} &= w_{11}^{(n-1)} \cdot y_1^{(n-2)} + w_{12}^{(n-1)} \cdot y_2^{(n-2)} + w_{13}^{(n-1)} \cdot y_3^{(n-2)} + w_{14}^{(n-1)} \cdot y_4^{(n-2)} + b_1^{(n-1)} \\ &= 0.1 \cdot 0.01 + 0.4 \cdot 0.3 + (-0.2) \cdot 0 + 0.3 \cdot 0 + 0.05 = 0.17 \\ z_2^{(n-1)} &= w_{21}^{(n-1)} \cdot y_1^{(n-2)} + w_{22}^{(n-1)} \cdot y_2^{(n-2)} + w_{23}^{(n-1)} \cdot y_3^{(n-2)} + w_{24}^{(n-1)} \cdot y_4^{(n-2)} + b_2^{(n-1)} \\ &= (-0.3) \cdot 0.01 + 0.5 \cdot 0.3 + 0.2 \cdot 0 + (-0.4) \cdot 0 + (-0.05) = 0.125 \\ z_3^{(n-1)} &= 0.2 \cdot 0.01 + (-0.1) \cdot 0.3 + 0.3 \cdot 0 + 0.1 \cdot 0 + 0.02 = -0.008 \\ z_4^{(n-1)} &= (-0.4) \cdot 0.01 + 0.2 \cdot 0.3 + 0.1 \cdot 0 + (-0.2) \cdot 0 + (-0.01) = 0.045 \end{aligned}$$

Apply ReLU:

$$\begin{aligned} y_1^{(n-1)} &= \max(0, 0.17) = 0.17 \\ y_2^{(n-1)} &= \max(0, 0.125) = 0.125 \\ y_3^{(n-1)} &= \max(0, -0.008) = 0 \\ y_4^{(n-1)} &= \max(0, 0.045) = 0.045 \end{aligned}$$

Output Layer (n-th layer):

Let the weights $w_{ij}^{(n)}$ and biases $b_i^{(n)}$ be:

$$\begin{aligned} w_{11}^{(n)} = 0.2, \quad w_{12}^{(n)} = -0.3, \quad w_{13}^{(n)} = 0.1, \quad w_{14}^{(n)} = -0.2 \\ w_{21}^{(n)} = 0.5, \quad w_{22}^{(n)} = 0.4, \quad w_{23}^{(n)} = -0.1, \quad w_{24}^{(n)} = 0.3 \end{aligned}$$

Biases:

$$b_1^{(n)} = 0.1, \quad b_2^{(n)} = -0.05$$

For each neuron in the output layer:

$$\begin{aligned} z_1^{(n)} &= w_{11}^{(n)} \cdot y_1^{(n-1)} + w_{12}^{(n)} \cdot y_2^{(n-1)} + w_{13}^{(n)} \cdot y_3^{(n-1)} + w_{14}^{(n)} \cdot y_4^{(n-1)} + b_1^{(n)} \\ &= 0.2 \cdot 0.17 + (-0.3) \cdot 0.125 + 0.1 \cdot 0 + (-0.2) \cdot 0.045 + 0.1 = 0.1155 \\ z_2^{(n)} &= 0.5 \cdot 0.17 + 0.4 \cdot 0.125 + (-0.1) \cdot 0 + 0.3 \cdot 0.045 + (-0.05) = 0.179 \end{aligned}$$

Thus, the final output is:

$$\text{Output } z_1^{(n)} = 0.1155, \quad \text{Output } z_2^{(n)} = 0.179$$

Applying Final Activation (Sigmoid)

Usually Softmax is applied as the final activation but using it here will cause additional complexity in backpropagation as it is a **Vector Activation**. We will discuss this complexity in the upcoming section. So we will use sigmoid. Also the below proabilities were calculated using softmax but lets assume the came from sigmoid.

The outputs after applying the sigmoid function (actually softmax but were assuming sigmoid for simplification in upcoming steps) are:

$$p_1 = 0.484, \quad p_2 = 0.516$$

9.4.2 Loss Calculation

The cross-entropy loss function is defined as:

$$L = -\sum_i y_i \log(p_i)$$

Where y_i is the true label and p_i is the predicted probability from the sigmoid.

Given the true target outputs:

$$y_1 = 0.3, \quad y_2 = 0.6$$

And the predicted probabilities:

$$p_1 = 0.484, \quad p_2 = 0.516$$

The cross-entropy loss is calculated as:

$$L = -\left[0.3 \log(0.484) + 0.6 \log(0.516)\right]$$

First, calculate the logarithms:

$$\log(0.484) \approx -0.726, \quad \log(0.516) \approx -0.662$$

Now, substitute these values:

$$L = -\left[0.3 \cdot (-0.726) + 0.6 \cdot (-0.662)\right]$$

$$L = -\left[-0.2178 - 0.3972\right]$$

$$L = 0.615$$

Thus, the cross-entropy loss is approximately:

$$L \approx 0.615$$

9.4.3 Backward Pass

1. Output Layer

We perform the backward pass by calculating the gradients of the loss with respect to the weights and biases, starting from the output layer and propagating backward.

The derivative of the loss L with respect to $z_i^{(n)}$ for softmax and cross-entropy is given by:

$$\frac{\partial L}{\partial z_1^{(n)}} = \hat{y}_1^{(n)} - y_1^{(n)} = 0.484 - 0.3 = 0.184$$

$$\frac{\partial L}{\partial z_2^{(n)}} = \hat{y}_2^{(n)} - y_2^{(n)} = 0.516 - 0.6 = -0.084$$

Next, we calculate the gradients with respect to the weights and biases:

$$\frac{\partial L}{\partial w_{ij}^{(n)}} = \frac{\partial L}{\partial z_i^{(n)}} \cdot y_j^{(n-1)}$$

For $w_{11}^{(n)}$:

$$\frac{\partial L}{\partial w_{11}^{(n)}} = -0.5035 \cdot y_1^{(n-1)} = -0.5035 \cdot 0.17 = -0.0856$$

For $w_{12}^{(n)}$:

$$\frac{\partial L}{\partial w_{12}^{(n)}} = -0.5035 \cdot y_2^{(n-1)} = -0.5035 \cdot 0.125 = -0.0629$$

Similarly, for $w_{13}^{(n)}$ and $w_{14}^{(n)}$:

$$\frac{\partial L}{\partial w_{13}^{(n)}} = -0.5035 \cdot 0 = 0$$

$$\frac{\partial L}{\partial w_{14}^{(n)}} = -0.5035 \cdot 0.045 = -0.0227$$

Now, for the derivatives related to the second output neuron:
For $w_{21}^{(n)}$:

$$\frac{\partial L}{\partial w_{21}^{(n)}} = 0.5035 \cdot y_1^{(n-1)} = 0.5035 \cdot 0.17 = 0.0856$$

For $w_{22}^{(n)}$:

$$\frac{\partial L}{\partial w_{22}^{(n)}} = 0.5035 \cdot y_2^{(n-1)} = 0.5035 \cdot 0.125 = 0.0629$$

Similarly, for $w_{23}^{(n)}$ and $w_{24}^{(n)}$:

$$\frac{\partial L}{\partial w_{23}^{(n)}} = 0.5035 \cdot 0 = 0$$

$$\frac{\partial L}{\partial w_{24}^{(n)}} = 0.5035 \cdot 0.045 = 0.0227$$

For the bias gradients:

$$\frac{\partial L}{\partial b_1^{(n)}} = -0.5035, \quad \frac{\partial L}{\partial b_2^{(n)}} = 0.5035$$

2. Layer $n-1$

We propagate the gradients backward using the chain rule: Gradient of the loss with respect to $z_j^{(n-1)}$:

$$\frac{\partial L}{\partial z_j^{(n-1)}} = \sum_{i=1}^{2} \frac{\partial L}{\partial z_i^{(n)}} \cdot w_{ij}^{(n)} \cdot f'(z_j^{(n-1)})$$

Where $f'(z_j^{(n-1)})$ is the derivative of the activation function (ReLU, in this case). Since ReLU is applied, $f'(z_j^{(n-1)})$ will be 1 if $z_j^{(n-1)} > 0$ and 0 if $z_j^{(n-1)} \leq 0$.
For $z_1^{(n-1)}$:

$$\frac{\partial L}{\partial z_1^{(n-1)}} = (-0.5035 \cdot 0.2 + 0.5035 \cdot 0.1) \cdot 1 = -0.05035$$

For $z_2^{(n-1)}$:

$$\frac{\partial L}{\partial z_2^{(n-1)}} = (-0.5035 \cdot (-0.1) + 0.5035 \cdot 0.5) \cdot 1 = 0.3015$$

For $z_3^{(n-1)}$:

$$\frac{\partial L}{\partial z_3^{(n-1)}} = (-0.5035 \cdot 0.05 + 0.5035 \cdot (-0.2)) = -0.126$$

For $z_4^{(n-1)}$:

$$\frac{\partial L}{\partial z_4^{(n-1)}} = (-0.5035 \cdot (-0.1) + 0.5035 \cdot 0.3) = 0.2014$$

The gradients with respect to the weights between the first hidden layer (Layer $n-2$) and the second hidden layer (Layer $n-1$) are given by:

$$\frac{\partial L}{\partial w_{ij}^{(n-1)}} = \frac{\partial L}{\partial z_i^{(n-1)}} \cdot y_j^{(n-2)}$$

For $w_{11}^{(n-1)}$ (weight connecting neuron 1 of Layer $n-1$ to neuron 1 of Layer $n-2$):

$$\frac{\partial L}{\partial w_{11}^{(n-1)}} = -0.05035 \cdot y_1^{(n-2)} = -0.05035 \cdot 0.56 = -0.0282$$

For $w_{12}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{12}^{(n-1)}} = -0.05035 \cdot y_2^{(n-2)} = -0.05035 \cdot 0.41 = -0.02067$$

For $w_{13}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{13}^{(n-1)}} = -0.05035 \cdot y_3^{(n-2)} = -0.05035 \cdot 0.23 = -0.01158$$

For $w_{14}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{14}^{(n-1)}} = -0.05035 \cdot y_4^{(n-2)} = -0.05035 \cdot 0.1 = -0.00504$$

For $w_{21}^{(n-1)}$ (weight connecting neuron 2 of Layer $n-1$ to neuron 1 of Layer $n-2$):

$$\frac{\partial L}{\partial w_{21}^{(n-1)}} = 0.3015 \cdot y_1^{(n-2)} = 0.3015 \cdot 0.56 = 0.16884$$

For $w_{22}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{22}^{(n-1)}} = 0.3015 \cdot y_2^{(n-2)} = 0.3015 \cdot 0.41 = 0.123615$$

For $w_{23}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{23}^{(n-1)}} = 0.3015 \cdot y_3^{(n-2)} = 0.3015 \cdot 0.23 = 0.069345$$

For $w_{24}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{24}^{(n-1)}} = 0.3015 \cdot y_4^{(n-2)} = 0.3015 \cdot 0.1 = 0.03015$$

For $w_{31}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{31}^{(n-1)}} = -0.126 \cdot y_1^{(n-2)} = -0.126 \cdot 0.56 = -0.07056$$

For $w_{32}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{32}^{(n-1)}} = -0.126 \cdot y_2^{(n-2)} = -0.126 \cdot 0.41 = -0.05166$$

For $w_{33}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{33}^{(n-1)}} = -0.126 \cdot y_3^{(n-2)} = -0.126 \cdot 0.23 = -0.02898$$

For $w_{34}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{34}^{(n-1)}} = -0.126 \cdot y_4^{(n-2)} = -0.126 \cdot 0.1 = -0.0126$$

For $w_{41}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{41}^{(n-1)}} = 0.2014 \cdot y_1^{(n-2)} = 0.2014 \cdot 0.56 = 0.112784$$

For $w_{42}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{42}^{(n-1)}} = 0.2014 \cdot y_2^{(n-2)} = 0.2014 \cdot 0.41 = 0.082574$$

For $w_{43}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{43}^{(n-1)}} = 0.2014 \cdot y_3^{(n-2)} = 0.2014 \cdot 0.23 = 0.046322$$

For $w_{44}^{(n-1)}$:

$$\frac{\partial L}{\partial w_{44}^{(n-1)}} = 0.2014 \cdot y_4^{(n-2)} = 0.2014 \cdot 0.1 = 0.02014$$

Gradients with Respect to the Biases in Layer $n-1$:

$$\frac{\partial L}{\partial b_1^{(n-1)}} = -0.05035, \quad \frac{\partial L}{\partial b_2^{(n-1)}} = 0.3015$$
$$\frac{\partial L}{\partial b_3^{(n-1)}} = -0.126, \quad \frac{\partial L}{\partial b_4^{(n-1)}} = 0.2014$$

3. Layer $n-2$

We'll now calculate the gradients for the weights and biases in the first hidden layer (Layer $n-2$).

Gradient of Loss with Respect to $z_j^{(n-2)}$

First, we need to calculate the gradient of the loss with respect to the outputs of this layer:

$$\frac{\partial L}{\partial z_j^{(n-2)}} = \sum \left(\frac{\partial L}{\partial z_i^{(n-1)}} \cdot w_{ij}^{(n-1)} \right) \cdot f'(z_j^{(n-2)})$$

Where $f'(z_j^{(n-2)})$ is the derivative of the ReLU activation function:

$$f'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

For $z_1^{(n-2)}$:

$$\frac{\partial L}{\partial z_1^{(n-2)}} = (-0.05035 \cdot 0.1 + 0.3015 \cdot -0.3 + -0.126 \cdot 0.2 + 0.2014 \cdot -0.4) \cdot 1 = -0.201245$$

For $z_2^{(n-2)}$:

$$\frac{\partial L}{\partial z_2^{(n-2)}} = (-0.05035 \cdot 0.4 + 0.3015 \cdot 0.5 + -0.126 \cdot -0.1 + 0.2014 \cdot 0.2) \cdot 1 = 0.18349$$

For $z_3^{(n-2)}$:

$$\frac{\partial L}{\partial z_3^{(n-2)}} = (-0.05035 \cdot -0.2 + 0.3015 \cdot 0.2 + -0.126 \cdot 0.3 + 0.2014 \cdot 0.1) \cdot 0 = 0$$

(since $z_3^{(n-2)} = -0.06$, which is negative, so the ReLU derivative is 0)

For $z_4^{(n-2)}$:

$$\frac{\partial L}{\partial z_4^{(n-2)}} = (-0.05035 \cdot 0.3 + 0.3015 \cdot -0.4 + -0.126 \cdot 0.1 + 0.2014 \cdot -0.2) \cdot 0 = 0$$

(since $z_4^{(n-2)} = -0.16$, which is negative, so the ReLU derivative is 0)

Gradients with Respect to Weights in Layer $n-2$

Now we can calculate the gradients for the weights:

$$\frac{\partial L}{\partial w_{ij}^{(n-2)}} = \frac{\partial L}{\partial z_i^{(n-2)}} \cdot x_j$$

For $w_{11}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{11}^{(n-2)}} = -0.201245 \cdot 0.5 = -0.1006225$$

For $w_{12}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{12}^{(n-2)}} = -0.201245 \cdot 0.8 = -0.160996$$

For $w_{21}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{21}^{(n-2)}} = 0.18349 \cdot 0.5 = 0.091745$$

For $w_{22}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{22}^{(n-2)}} = 0.18349 \cdot 0.8 = 0.146792$$

For $w_{31}^{(n-2)}$ and $w_{32}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{31}^{(n-2)}} = \frac{\partial L}{\partial w_{32}^{(n-2)}} = 0$$

(Since $\frac{\partial L}{\partial z_3^{(n-2)}} = 0$ due to ReLU)

For $w_{41}^{(n-2)}$ and $w_{42}^{(n-2)}$:

$$\frac{\partial L}{\partial w_{41}^{(n-2)}} = \frac{\partial L}{\partial w_{42}^{(n-2)}} = 0$$

(Since $\frac{\partial L}{\partial z_4^{(n-2)}} = 0$ due to ReLU)

Gradients with Respect to Biases in Layer $n-2$

The gradients for the biases are simply the gradients of the loss with respect to the corresponding z values:

$$\frac{\partial L}{\partial b_1^{(n-2)}} = -0.201245$$

$$\frac{\partial L}{\partial b_2^{(n-2)}} = 0.18349$$

$$\frac{\partial L}{\partial b_3^{(n-2)}} = 0$$

$$\frac{\partial L}{\partial b_4^{(n-2)}} = 0$$

These gradients complete the backward pass through the entire network. They can now be used to update the weights and biases using an optimization algorithm like gradient descent.

9.5 Next Steps in Neural Network Training

After completing one forward pass and one backward pass, the next step in training a neural network is to perform multiple iterations of forward and backward passes over the training dataset. This process is crucial for several reasons:

Learning Representation

During each forward pass, the network generates predictions based on the current weights and biases. The backward pass then computes gradients that indicate how much each weight and bias should be adjusted to minimize the loss. Repeating this process allows the network to iteratively learn and refine its parameters, effectively improving its ability to make accurate predictions.

For instance, in our earlier computation, we found the output values $z_1(n)$ and $z_2(n)$ along with the corresponding softmax outputs. The backward pass then computes gradients that indicate how much each weight and bias should be adjusted to minimize the loss. For example, we derived the gradients for the weights in the output layer:

$$\frac{\partial L}{\partial w_{11}(n)} = -0.0856, \quad \frac{\partial L}{\partial w_{12}(n)} = -0.0629, \quad \frac{\partial L}{\partial w_{21}(n)} = 0.16884, \quad \text{and others}$$

These gradients provide essential information on how to update each weight. Repeating this process allows the network to iteratively learn and refine its parameters based on the computed derivatives, effectively improving its ability to make accurate predictions.

The weights in a neural network are updated using gradient descent or its variants. The update rule typically involves the following steps:

1. Calculate the Gradients

After performing a backward pass, we obtain the gradients of the loss with respect to each weight. These gradients tell us the direction and rate of change needed to minimize the loss function.

2. Apply the Update Rule

The weights are updated using the following formula:

$$w_{ij}^{(new)} = w_{ij}^{(old)} - \eta \frac{\partial L}{\partial w_{ij}}$$

where:

- $w_{ij}^{(new)}$ is the updated weight.

- $w_{ij}^{(old)}$ is the current weight.
- η (eta) is the learning rate, a hyperparameter that determines the size of the step taken in the direction of the gradient.
- $\frac{\partial L}{\partial w_{ij}}$ is the gradient of the loss with respect to the weight w_{ij}.

3. Repeat

This process is repeated for each weight in the network and across multiple epochs. The learning rate can be adjusted dynamically, and various optimization techniques can be applied, such as momentum, Adam, or RMSprop, to improve convergence.

Example of Weight Update

Using the previously computed gradients for the output layer:

$$\frac{\partial L}{\partial w_{11}(n)} = -0.0856, \quad \frac{\partial L}{\partial w_{12}(n)} = -0.0629, \quad \frac{\partial L}{\partial w_{21}(n)} = 0.16884$$

Assuming a learning rate $\eta = 0.01$:

1. **Update** w_{11}:

$$w_{11}^{(new)} = w_{11}^{(old)} - 0.01 \cdot (-0.0856) = w_{11}^{(old)} + 0.000856$$

2. **Update** w_{12}:

$$w_{12}^{(new)} = w_{12}^{(old)} - 0.01 \cdot (-0.0629) = w_{12}^{(old)} + 0.000629$$

3. **Update** w_{21}:

$$w_{21}^{(new)} = w_{21}^{(old)} - 0.01 \cdot (0.16884) = w_{21}^{(old)} - 0.0016884$$

Convergence

Neural networks often require many passes through the training data to converge to a set of weights that minimizes the loss function. This convergence occurs as the weights are adjusted incrementally over many iterations, allowing the network to explore the error surface and find an optimal solution.

Generalization

By performing multiple passes, the network can learn to generalize from the training data to unseen data. This is important for ensuring that the model does not just memorize the training examples but instead captures the underlying patterns.

Epochs and Batch Size

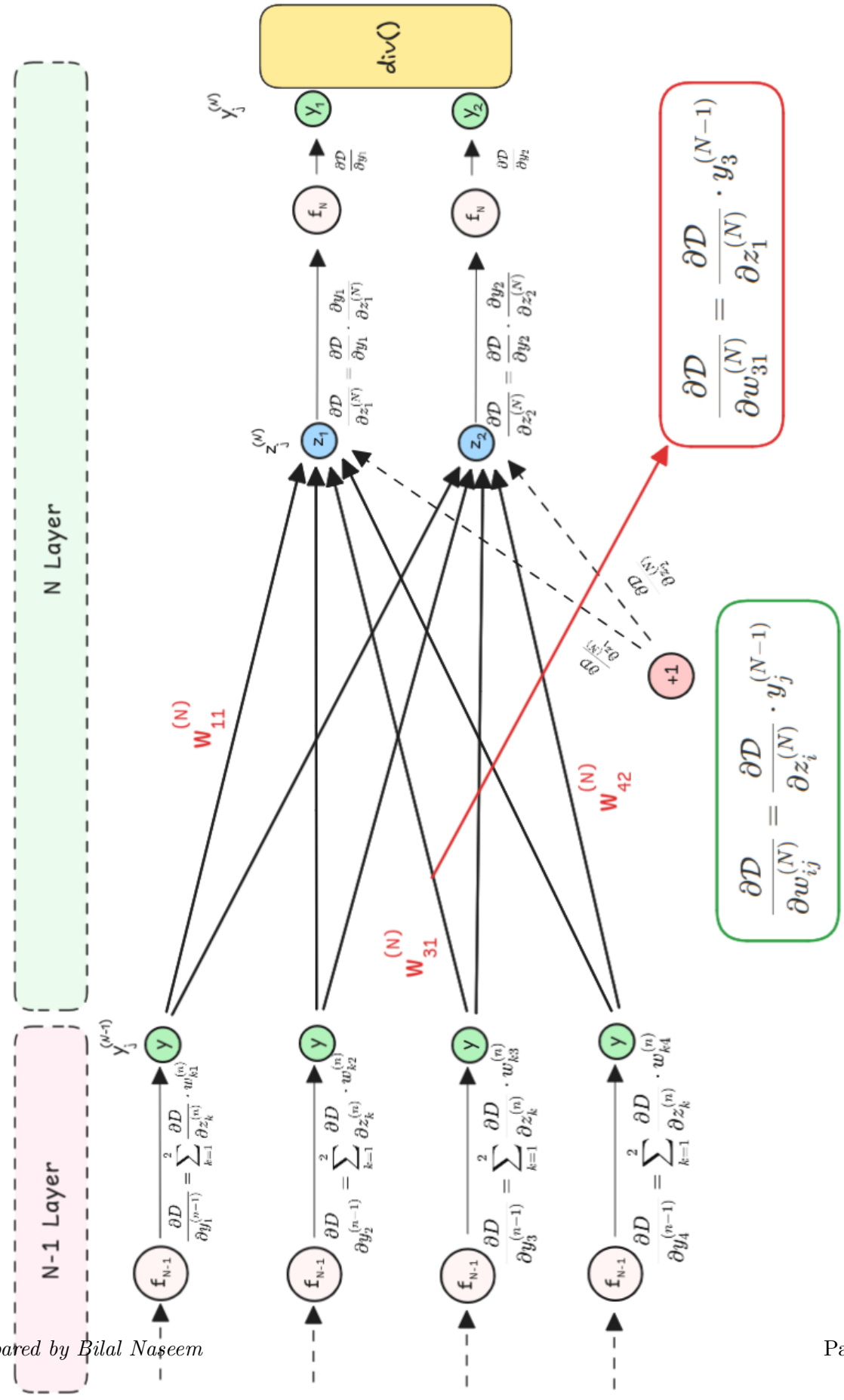
Two critical concepts that influence the number of forward and backward passes are **epochs** and **batch size**:

Epochs: An epoch refers to one complete pass through the entire training dataset. During training, multiple epochs are typically used to allow the model to learn from the data thoroughly. The number of epochs is a hyperparameter that can be adjusted; too few epochs might lead to underfitting, while too many can lead to overfitting.

Batch Size: The batch size determines the number of training examples used in one iteration of training. Instead of using the entire dataset for each update, which can be computationally expensive, the dataset is divided into smaller batches. Each batch is fed through the network, and after processing the batch, the weights are updated based on the gradients calculated from that batch. Smaller batch sizes can lead to more noisy gradients but allow for more frequent updates, while larger batch sizes provide smoother estimates of the gradient but require more memory and can lead to slower convergence.

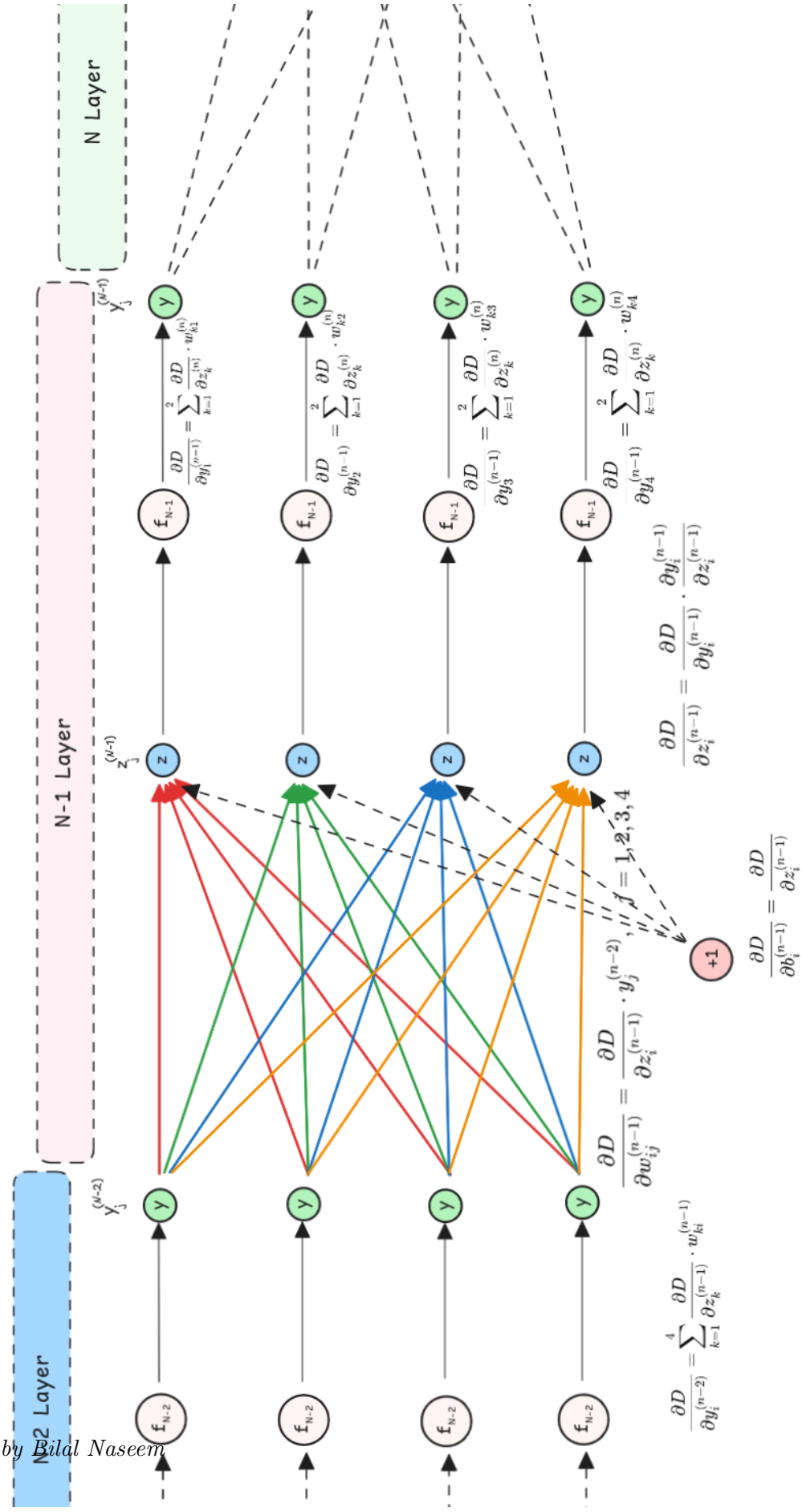
Overall, the interplay between epochs and batch size, along with the learning rate and other hyperparameters, plays a significant role in determining the effectiveness and efficiency of the training process.

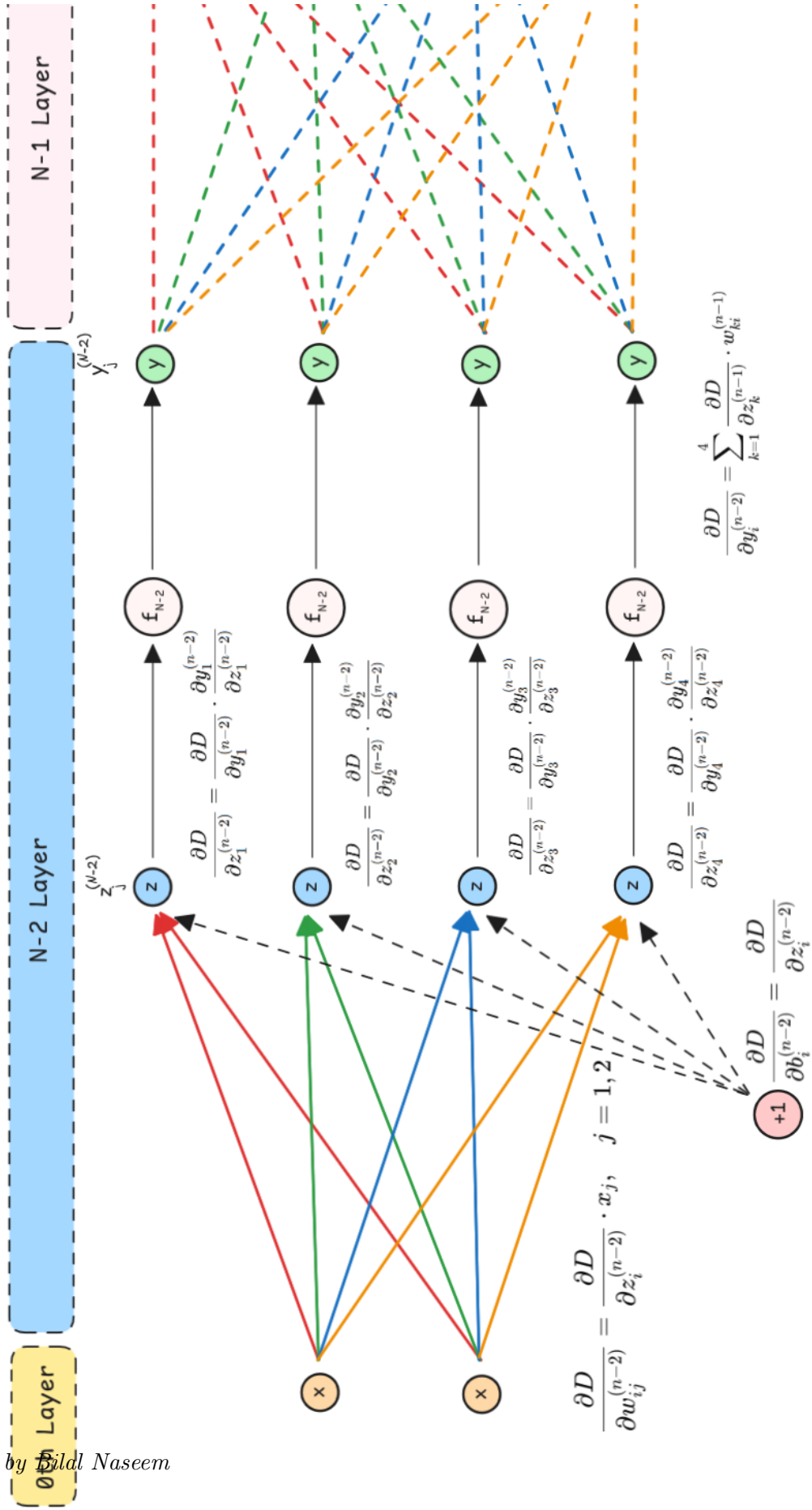
xxxxxxxxxxxxxx



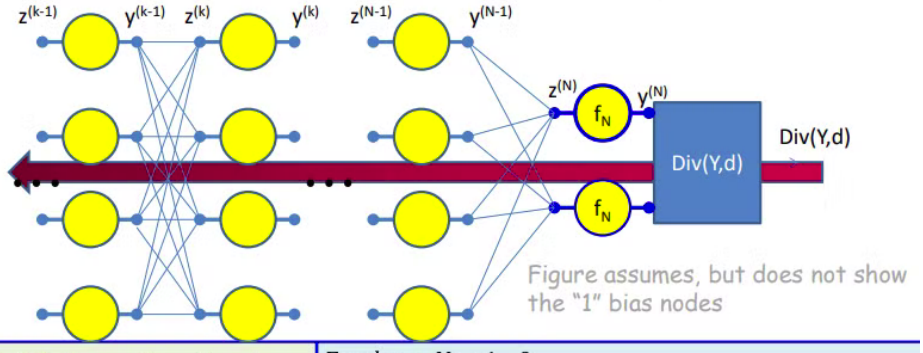
$$\frac{\partial D}{\partial w_{31}^{(N)}} = \frac{\partial D}{\partial z_1^{(N)}} \cdot y_3^{(N-1)}$$

$$\frac{\partial D}{\partial w_{ij}^{(N)}} = \frac{\partial D}{\partial z_i^{(N)}} \cdot y_j^{(N-1)}$$


N Layer
N-1 Layer
N-2 Layer
+1


N-1 Layer
N-2 Layer
0th Layer
z
y
x
+1
f_{N-2}
$y_j^{(N-2)}$
$z_j^{(N-2)}$
$\frac{\partial D}{\partial z_1^{(n-2)}} = \frac{\partial D}{\partial y_1^{(n-2)}} \cdot \frac{\partial y_1^{(n-2)}}{\partial z_1^{(n-2)}}$
$\frac{\partial D}{\partial z_2^{(n-2)}} = \frac{\partial D}{\partial y_2^{(n-2)}} \cdot \frac{\partial y_2^{(n-2)}}{\partial z_2^{(n-2)}}$
$\frac{\partial D}{\partial z_3^{(n-2)}} = \frac{\partial D}{\partial y_3^{(n-2)}} \cdot \frac{\partial y_3^{(n-2)}}{\partial z_3^{(n-2)}}$
$\frac{\partial D}{\partial z_4^{(n-2)}} = \frac{\partial D}{\partial y_4^{(n-2)}} \cdot \frac{\partial y_4^{(n-2)}}{\partial z_4^{(n-2)}}$
$\frac{\partial D}{\partial y_i^{(n-2)}} = \sum_{k=1}^{4} \frac{\partial D}{\partial z_k^{(n-1)}} \cdot w_{ki}^{(n-1)}$
$\frac{\partial D}{\partial w_{ij}^{(n-2)}} = \frac{\partial D}{\partial z_i^{(n-2)}} \cdot x_j, \quad j = 1, 2$
$\frac{\partial D}{\partial b_i^{(n-2)}} = \frac{\partial D}{\partial z_i^{(n-2)}}$

Gradients: Backward Computation



Initialize: Gradient w.r.t network output

$$\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$$

$$\frac{\partial Div}{\partial z_i^{(N)}} = f_k'\left(z_i^{(N)}\right)\frac{\partial Div}{\partial y_i^{(N)}}$$

For $k = N-1..0$

For $i = 1$: layer width

$$\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)}\frac{\partial Div}{\partial z_j^{(k+1)}}$$

$$\frac{\partial Div}{\partial z_i^{(k)}} = f_k'\left(z_i^{(k)}\right)\frac{\partial Div}{\partial y_i^{(k)}}$$

$$\forall j \quad \frac{\partial Div}{\partial w_{ij}^{(k+1)}} = y_i^{(k)}\frac{\partial Div}{\partial z_j^{(k+1)}}$$

86

Backward Pass

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(N)}} = y_i^{(N-1)}\frac{\partial Div}{\partial z_j^{(N)}}$ for $j = 0 \ldots D_{N-1}$
- For layer $k = N-1$ *downto* 1
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)}\frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ij}^{(k)}} = y_i^{(k-1)}\frac{\partial Div}{\partial z_j^{(k)}}$ for $j = 0 \ldots D_{k-1}$

Called "Backpropagation" because the derivative of the loss is propagated "backwards" through the network

Very analogous to the forward pass:

Backward weighted combination of next layer

Backward equivalent of activation

88

Using notation $\dot{y} = \frac{\partial Div(Y,d)}{\partial y}$ etc (overdot represents derivative of Div w.r.t variable)

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\dot{y}_i^{(N)} = \frac{\partial Div}{\partial y_i}$
    - $\dot{z}_i^{(N)} = \dot{y}_i^{(N)} f_N'\left(z_i^{(N)}\right)$
    - $\frac{\partial Div}{\partial w_{ji}^{(N)}} = y_j^{(N-1)}\dot{z}_i^{(N)}$ for $j = 0 \ldots D_{N-1}$
- For layer $k = N-1$ *downto* 1
  - For $i = 1 \ldots D_k$
    - $\dot{y}_i^{(k)} = \sum_j w_{ij}^{(k+1)}\dot{z}_j^{(k+1)}$
    - $\dot{z}_i^{(k)} = \dot{y}_i^{(k)} f_k'\left(z_i^{(k)}\right)$
    - $\frac{\partial Div}{\partial w_{ji}^{(k)}} = y_j^{(k-1)}\dot{z}_i^{(k)}$ for $j = 0 \ldots D_{k-1}$

Called "Backpropagation" because the derivative of the loss is propagated "backwards" through the network

Very analogous to the forward pass:

Backward weighted combination of next layer

Backward equivalent of activation

Summary

Backpropagation is used to compute the derivatives of the divergence with respect to the model parameters, to be used in gradient descent.