

M01S03- Backpropagation in Neural Networks

Introduction to neural networks 6 hrs 55 mins estimate

1. Introduction
2. What Does Training a Network Mean?
3. Complexity of the Loss Function
4. Comprehension - Training a Neural Network
5. Updating the Weights and Biases - I
6. Updating the Weights and Biases - II
7. Updating the Weights and Biases - III
8. Sigmoid Backpropagation
9. Updating the Weights and Biases - IV
10. Updating the Weights and Biases - V
11. Updating the Weights and Biases - VI
12. Batch in Backpropagation
13. Training in Batches
14. Summary
15. Graded Questions

Introduction

Previous session covered how the information flows through a neural network in the forward direction. This session covers the process of training neural networks called **backpropagation**.

Able to **build own neural network from scratch** in Numpy. This is the task of the assignment of this module.

In this Session:

- Training a Neural Network
- Cost Function and its Complexity
- Gradient Descent in Neural Networks
- The backpropagation algorithm

Prerequisites

There are no prerequisites for this session other than knowledge of the matrix multiplication and previous courses on Statistics and ML.

What Does Training a Network Mean?

Previous session covered how the information passes through neural networks.

This session will cover how neural networks are **trained**. Training task is to compute the optimal weights and biases by **minimizing some cost function**. In the upcoming lectures, study all the elements involved in training neural networks in detail - the loss function, backward flow of information, optimisation techniques etc.

Let's start with a quick recap of defining the training task.

Neural Network Topology

1. Architecture is fixed
2. Activation Functions are fixed
3. Train the network for weights and biases.

Training a Network

UpGrad

1. $\vec{h}^{l+1}(w, b, \vec{x}_i)$ → Output of the last layer for a given set of weights w , bias b and input \vec{x}_i

2. Expected output - y_i

$$\text{Cost/ Loss : } L(\vec{h}^{l+1}(w, b, \vec{x}_i), y_i)$$

3. Total cost across training data

$$\sum_{i=1}^n L(\vec{h}^{l+1}(w, b, \vec{x}_i), y_i) \text{ (Number of data points = } n\text{)}$$

4. Training problem → find w, b to minimize total cost

$$\min_{w, b} \sum_{i=1}^n L(\vec{h}^{l+1}(w, b, \vec{x}_i), y_i)$$

The task of training neural networks is the same as that of other ML models such as linear regression, SVMs etc. The desired output (output from the last layer) minus the actual output is the **cost** (or the **loss**), and we tune the parameters w and b such that **the total cost is minimized**.

An important point to note is that if the data is large (which is often the case), loss calculation itself can get messy. For example, if million data points are fed into the network (in batch), the output will be calculated using feedforward and the loss/cost L_i (for i^{th} data point) will be calculated. The total loss is the sum of losses of all the individual data points. Hence,

$$\text{Total Loss} = L = L_1 + L_2 + L_3 + \dots + L_{1000000}$$

The total loss L is a function of w 's and b 's. Once the total loss is computed, the weights and biases are updated (in the direction of decreasing loss). In other words, L is **minimized with respect to the w 's and b 's**.

This can be done using any optimisation routine such as **gradient descent**.

We minimize the average of the total loss and not the total loss. Minimizing the average loss implies that the total loss is getting minimized.

Complexity of the Loss Function

Training refers to the task of finding the optimal combination of weights and biases to minimise the total loss (with a fixed set of hyperparameters). This segment covers the anatomy and complexity of the cost function.

The optimisation is done using the familiar **gradient descent** algorithm, the parameter being optimised is iterated in the direction of reducing cost according to the following rule:

$$W_{\text{new}} = W_{\text{old}} - \alpha \cdot \partial L / \partial W.$$

The same can be written for the biases. Note that the weights and biases are often collectively represented by one matrix called W . Going forward, W will by default refer to the matrix of all the weights and biases.

The main challenge is that W is a huge matrix, and thus, the total loss L as a function of W is a complex function. Let's see how we deal with this complexity.

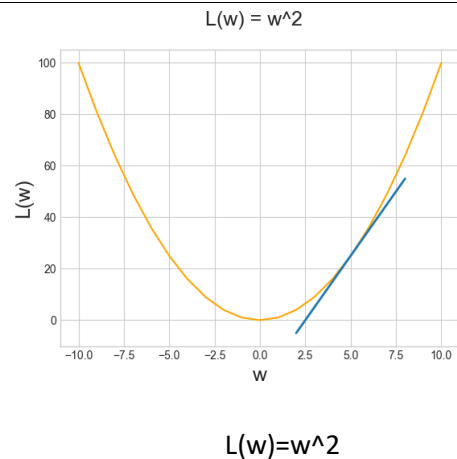
Cost Function	Update
$\text{Loss} = \min_{w,b} \sum_{i=1}^n L(\vec{h}^{l+1}(w, b, \vec{x}_i), y_i)$	Loss function for a very small and simple neural network can be very complex. The best way to minimise this complex loss function is by using gradient descent.
1. Hidden layers - 5	
2. Neurons - 100	
3. Input layers - 100	
4. Output layers - 100	
5. 100 X 100 = 10000 (Interconnections)	
6. Weight matrices - 6	
7. Weights in the entire network - 60000	
8. Biased value for each layer - 100	
9. Number of parameters - 60000 + 600	

Let's revisit some basic concepts of gradient descent optimisation.

Comprehension - Gradient Descent

Gradient descent is an optimisation algorithm used to find the minimum of a function. The basic idea is to use the gradient of the function to find **the direction of steepest descent**, i.e. the direction in which the value of the function decreases most rapidly and move towards the minima iteratively.

Let's take an example of a one-dimensional (univariate) function. Say you have a loss function L which depends only on one variable w : $L(w) = w^2$. The minimum of this function is at $w = 0$.



The algorithm starts with an initial arbitrary guess of w , computes the gradient at that point, and updates w according to the rule iteratively:

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot (\partial L / \partial w)$$

For example, let's take an arbitrary initial value $w_0 = 5$. The gradient $\partial L / \partial w$ is $2w$, so the value of the gradient (the slope) at $w_0 = 5$ is $2 \cdot 5 = 10$, as shown by the blue line in the figure above. The gradient has **two critical pieces of information**:

- The **sign of the gradient** (positive here) is the 'direction' in which the function value increases, and thus, the negative of that sign is the **direction of decrease**.
- The **value of the gradient** (10 here) represents how steeply the function value increases or decreases at that point.

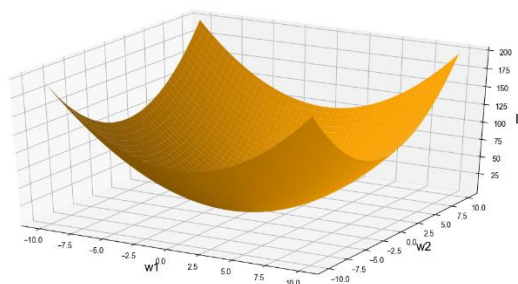
We want to move in the direction of decreasing function value, i.e. in the negative direction of the gradient (towards the origin from $w_0 = 5$). Also, we want to take a larger step if the gradient value is high, and a smaller one if it is low. We control the **step size** through **the learning rate** α . Both these ideas are captured in the term $-\alpha \cdot \partial L / \partial w$.

Let's perform one iteration of gradient descent starting with $w_0 = 5$ and $\alpha = 0.1$:

$$\begin{aligned} w_1 &= w_0 - \alpha \cdot \partial L / \partial w \\ w_1 &= 5 - (0.1) (10) \\ w_1 &= 4 \end{aligned}$$

We have moved closer to the minima. In another iteration, verify that we move closer.

Gradient descent can be easily extended to multivariate functions, i.e. functions of multiple variables. Let's take the bivariate function $L(w_1, w_2) = w_1^2 + w_2^2$. The minimum value of this function is 0 at the point $(w_1, w_2) = (0, 0)$. For convenience, let's represent the two variables together as $W = (w_1, w_2)$.



Bivariate Function

The iterative procedure is the same - start with an arbitrary initial guess of $W_0 = (w_1, w_2)_0$ and move in the direction of decreasing function value. The only change is that the **gradient** $\partial L / \partial W$ is now a **vector**:

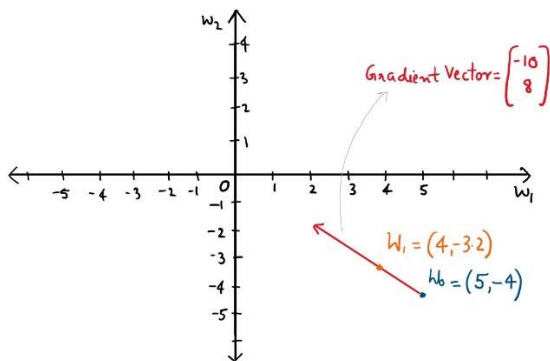
$$\frac{\partial L}{\partial W} = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

Each component of the **gradient vector** conveys the same two pieces of information. For e.g. let's take an initial guess $W_0 = (5, -4)$. The gradient at this point is:

$$\frac{\partial L}{\partial W} = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix} = \begin{bmatrix} 10 \\ -8 \end{bmatrix}$$

The first component ($2w_1 = 10$), being positive, says that the function value increases along the direction of increasing w_1 , and the 'rate of change' along the w_1 axis is 10. Similarly, the second component ($2w_2 = -8$) says that the function decreases along the direction of increasing w_2 with a rate of 8.

Combining both the elements, the **negative of the gradient vector**, $-\begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix} = \begin{bmatrix} -10 \\ 8 \end{bmatrix}$, is the **direction** in which the function value decreases most rapidly. The gradient vector is shown on a w_1 - w_2 plane below:



Gradient Vector

We take a step along that vector according to (assuming the same learning rate $\alpha=0.1$):

$$\begin{aligned} W_1 &= W_0 - \alpha \cdot \frac{\partial L}{\partial W} \\ W_1 &= \begin{bmatrix} 5 \\ -4 \end{bmatrix} - 0.1 \begin{bmatrix} 10 \\ -8 \end{bmatrix} \\ W_1 &= \begin{bmatrix} 4 \\ -3.2 \end{bmatrix} \end{aligned}$$

$$W_1 = \begin{bmatrix} 4 \\ -3.2 \end{bmatrix}$$

$$W_0 = \begin{bmatrix} 5 \\ -4 \end{bmatrix}$$

Notice that the point is closer to the minima (0,0) than the starting point. Perform one more iteration from W_1 to W_2 to verify move closer to the minima.

We can now extend this idea to any number of variables. Say one of the neural network layers has n biases. Represent them in a large vector (b_1, b_2, \dots, b_n) . The gradient vector will also be an n -dimensional vector, each element of which captures two pieces of information - the direction and rate of change of the function with respect to the parameter b_i .

$$\frac{\partial L}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \\ \dots \\ \frac{\partial L}{\partial b_n} \end{bmatrix}$$

Gradient Descent with One Variable

Consider the minimisation of the univariate function $L(w) = w^2$. We have already seen one iteration starting from $w_0 = 5$ which moves to $w_1 = 4$. Continue one more iteration with the same learning rate $\alpha = 0.1$. What is the value of w_2 ?

☐ 4.8

☒ 3.2

Feedback:
 $w_2 = w_1 - \alpha \frac{\partial L}{\partial w} = 4 - 0.1(8) = 3.2$

☐ 3.6

Gradient Descent with Two Variables

Consider the minimisation of the function $L(w_1, w_2) = w_1^2 + w_2^2$. We have already performed one iteration starting from $W_0 = (5, -4)$ which moves to $W_1 = (4, -3.2)$. Continue one more iteration with the same learning rate $\alpha = 0.1$. What is the value of W_2 ?

☒ $(3.2, -2.56)$

Feedback:
 $W_2 = W_1 - \alpha \frac{\partial L}{\partial W}$

☐ $(3.2, 2.56)$

☐ $(3.2, -3.84)$

Gradient of Loss Function

Consider an n -dimensional setting where the loss function L depends on n parameters. Choose all correct statements about the gradient of the loss function with respect to the model parameters:

☐ The gradient is an n -dimensional vector

Feedback:

The gradient vector is the direction in which the loss value increases most rapidly.

☒ The gradient is a scalar

Feedback:

The gradient vector is the direction in which the loss value increases most rapidly.

☒ The gradient gives the direction in which the loss decreases most rapidly

Feedback:

The gradient vector is the direction in which the loss value increases most rapidly.

☐ The gradient gives the direction in which the loss increases most rapidly

Feedback:

The gradient vector is the direction in which the loss value increases most rapidly.

Gradient of a Loss with Respect to Multiple Variables

Say a layer in your neural network has a large number of biases - n biases, denoted by the variable \vec{b} . In some iteration of the algorithm, the gradient is computed to be as follows:

$$\frac{\partial L}{\partial \vec{b}} = \begin{bmatrix} \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \\ \vdots \\ \frac{\partial L}{\partial b_{j-1}} \\ \frac{\partial L}{\partial b_j} \\ \frac{\partial L}{\partial b_{j+1}} \\ \vdots \\ \frac{\partial L}{\partial b_n} \end{bmatrix} = \begin{bmatrix} 0.45 \\ -0.30 \\ \vdots \\ 0.20 \\ 0.00 \\ \vdots \\ \frac{\partial L}{\partial b_n} \end{bmatrix}$$

Imagine an n -dimensional space whose each dimension (axis) corresponds to one parameter of the network. Choose all the correct statements:

☒ In the next iteration, the algorithm should move in the direction of decreasing b_1

Feedback:

The loss with respect to b increases if $\frac{\partial L}{\partial b}$ is positive (and vice-versa). If it is zero, it means the loss does not depend (locally) on that variable. Also, the magnitude of the gradient defines how large a step the algorithm takes along that variable.

Correct

☐ In the next iteration, the algorithm should move in the direction of increasing b_1

☒ In the next iteration, the algorithm should move towards increasing b_2

Feedback:

The loss with respect to b increases if $\frac{\partial L}{\partial b}$ is positive (and vice-versa). If it is zero, it means the loss does not depend (locally) on that variable. Also, the magnitude of the gradient defines how large a step the algorithm takes along that variable.

Correct

☒ In the next iteration, the algorithm will take a larger step along the dimension b_1 than b_2

Feedback:

The loss with respect to b increases if $\frac{\partial L}{\partial b}$ is positive (and vice-versa). If it is zero, it means the loss does not depend (locally) on that variable. Also, the magnitude of the gradient defines how large a step the algorithm takes along that variable.

Correct

☐ In the next iteration, the algorithm will take a larger step along the dimension b_2 than b_1

☒ Changing b_{j+1} slightly does not affect the value of the current loss significantly

Feedback:

The loss with respect to b increases if $\frac{\partial L}{\partial b}$ is positive (and vice-versa). If it is zero, it means the loss does not depend (locally) on that variable. Also, the magnitude of the gradient defines how large a step the algorithm takes along that variable.

Correct

Next segment gives an intuitive understanding of the training process using the example of a small neural network simulating an OR gate.

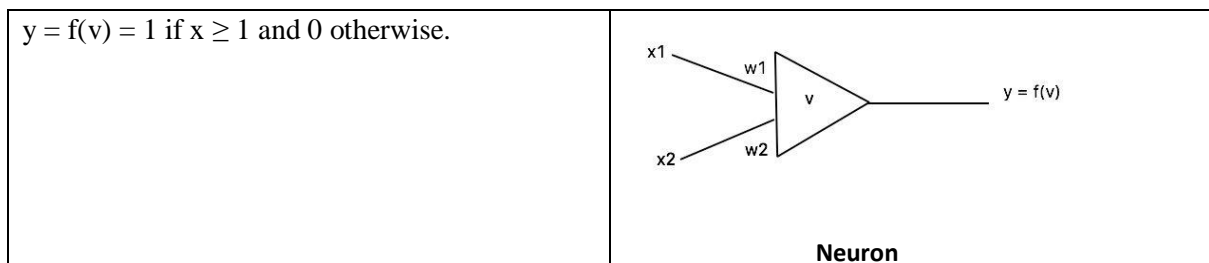
Comprehension - Training a Neural Network

Training a network essentially means to find the optimal set of weights and biases to minimise the total loss. The loss function is the difference between the actual output and the output predicted by the network (aggregated across all training data points).

Take a simple NN acting as an OR gate to understand the training process. Input to the logical OR gate is represented by two features (x_1 , x_2). Output is 1 if at least one of the inputs is 1, else it is 0.

Table 1 - OR Gate				
Input (x_1 , x_2)	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Output(y)	0	1	1	1

The table above is the training dataset, i.e. we have four data points in the training set. Also, our network has a single neuron (in its only hidden layer). The activation function for this neuron is defined as follows:



Assume that the bias of the neuron is 0. Thus, the weight matrix is $W = [w_1, w_2]$.

<p>Training of a Network Mark all the correct option(s):</p> <div><input checked="" type="checkbox"/> The training task is to find the weight vector (w_1, w_2) which minimizes the loss. ✓</div> <p>Feedback : Training a neural network basically implies finding correct values for weights and biases which minimises the loss function. The model starts with a random guess of the initial weights, predict the output using feedforward and change the weights in the direction of reducing loss. This is the gradient descent algorithm.</p> <div><input type="checkbox"/> In training, the network will start with a random guess of the two weights, calculate the predicted output and iterate the weights ✓</div> <p>Feedback : Training a neural network basically implies finding correct values for weights and biases which minimises the loss function. The model starts with a random guess of the initial weights, predict the output using feedforward and change the weights in the direction of reducing loss. This is the gradient descent algorithm.</p> <div><input type="checkbox"/> In training, the network will start with a random guess of the activation function, calculate the predicted output and iterate the function</div> <div><input checked="" type="checkbox"/> The training task is to find the set of outputs which minimizes the loss ✗</div> <p>Feedback : Training is not done to find the output which minimises the loss.</p>	<p>Output of the Network For an initial guess of the weights $w = (0, 0)$ the predicted output:</p> <div><input checked="" type="radio"/> Is 0 irrespective of the input</div> <p>Feedback : Given that $v = w \cdot x$ where w is the weight and x is the input. Multiplying input with weight 0 would result in $v=0$. Activation function is given as $f(v) = 1$ if $v \geq 1$ so $f(0) = 0$ irrespective of input. 0 otherwise</p> <div><input type="radio"/> Is 1 irrespective of the input</div> <div><input type="radio"/> Depends on the input</div>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The following image shows the different combinations of the weights, the inputs (x_1 , x_2), the predicted outputs and the loss for an initial guess of $W = [0, 0]$. The input can be either of the four values in column-2. We can calculate the predicted output for different values of W , compare that with the true output of an OR gate and compute the loss.

Following exercise aims to find the right w_1 and w_2 which minimise the total loss and correctly represent an OR gate.

Weights (w_1, w_2)	Input(i) (x_1, x_2)	Predicted Output (y_{pred})	True Label (y)	Loss (i) = $y - y_{pred}$	Total Loss = $\sum Loss(i)$
(0,0)	(0,0)	0	0	0	3
	(0,1)	0	1	1	
	(1,0)	0	1	1	
	(1,1)	0	1	1	
(1,0)	(0,0)				
	(0,1)				
	(1,0)				
	(1,1)				

(w_1, w_2)	(x_1, x_2)	Y_{pred}	Y	$y - y_{pred}$	Total Loss
(1, 0)	(0,0)	0	0	0	1
	(0,1)	0	1	1	
	(1,0)	1	1	0	
	(1,1)	1	1	0	
(0, 1)	(0,0)	0	0	0	1
	(0,1)	1	1	0	
	(1,0)	0	1	1	
	(1,1)	1	1	0	

Training a Single Neuron with Weights = [w_1, w_2]

Total Loss
For weight $W = [1, 0]$, the total loss is:

☐ 0

☒ 1

Feedback:
The predicted outputs for each input (x_1, x_2) are 0, 0, 1, 1. The true outputs are the same i.e. 0, 1, 1, 1. The loss vector is (0, 1, 0, 0) and hence the total loss is sum of loss(i) = 1.

☐ 2

☐ 3

Gradient Descent
The gradient descent algorithm will:

☐ Move in the direction of reducing weights by changing the loss

☒ Move in the direction of reducing loss by changing the weights

Feedback:
We want to minimize the loss by changing the weights, i.e. move in the direction where $d(loss)/d(W)$ decreases.

☐ Move in the direction of reducing weights by changing the activation function

☐ Move in the direction of reducing value of activation function by changing the loss

Gradient Descent
The gradient can be thought of as:

☐ The slope of a hill whose height represents the weight [w_1, w_2] and each location on the hill represents a unique cost

☐ The slope of a hill whose height represents the total cost and each location on the hill represents a unique activation function

☒ The slope of a hill whose height represents the total cost and each location on the hill represents a unique weight [w_1, w_2]

Feedback:
We can imagine a hill whose height represents the total cost and each location on it represents a unique weight [w_1, w_2]. Then we want to minimize the height (i.e. the cost), i.e. move in the direction of the slope of the hill to a point [w_1, w_2] where the cost / height is minimum.

Gradient Descent
We can imagine a hill whose height represents the total cost and each location on it represents a unique weight vector (w_1, w_2). The gradient, or the slope, can then be defined as $d(Loss) / d(W)$, i.e. change in total cost with respect to the change in $W = (w_1, w_2)$.

While changing the weight vector from the (0,0) to (1, 0):

☐ The value of $d(Loss)$ is positive

☒ The value of $d(Loss)$ is negative

Feedback:
The total costs for $w = (0, 0)$ and $w = (1, 0)$ are 3 and 1 respectively. The cost thus reduces as we move from (0,0) to (1, 0), so $d(Loss)$ is negative.

☐ The value of $d(Loss)$ is 0

Gradient Descent
While changing the weight vector from the (1, 0) to (0, 1):

☐ The gradient is positive

☐ The gradient is negative

☒ The gradient is 0

Feedback:
For weights = (1, 0) and (0, 1), the loss is 1, or $d(Loss)$ is 0. Thus the gradient is $d(Loss)/d(W) = 0$.

Gradient Descent
We can imagine a hill whose height represents the total cost and each location on it represents a unique weight (w_1, w_2). In case of heavy rainfall, the water will get accumulated at the point:

☐ (0, 0)

☐ (0, 1)

☐ (1, 0)

☒ (1, 1)

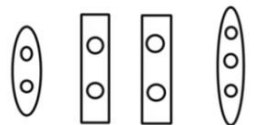
Feedback:
The point where $w = (1, 1)$ minimizes the Loss since, at that point, the predicted output exactly matches the true output of an OR gate.

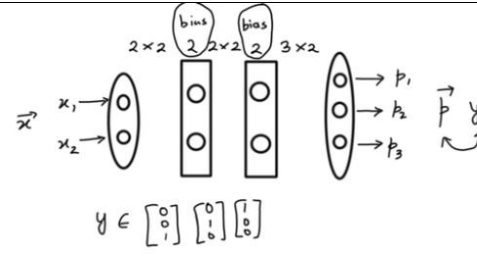
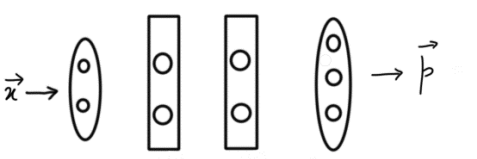
The weights are adjusted in such a way that the total loss is minimized. Now, having understood this, let's study **backpropagation** in detail.

Updating the Weights and Biases - I

Next few segments cover backpropagation in detail culminating in capability to build own NN in Numpy from the ground up as assignment.

Let's start off this segment by setting the problem statement for backpropagation. We shall use a simple network, a 3-layer network with 2 hidden layers and 1 output layer.

<p>Gradient w.r.t 'parameters' of funcⁿ</p> $G(\vec{w}, \vec{b}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(F(\vec{x}_i), y_i)$ <p>Funcⁿ Parameters</p> 	<p>The loss function is defined as follows:</p> $G(\vec{w}, \vec{b}) = \frac{1}{n} \sum_{i=1}^n L(F(\vec{x}_i), y_i)$ <p>The loss function L is defined in terms of the network output $F(\vec{x}_i)$ and the ground truth y_i. Since $F(\vec{x}_i)$ depends on the weights and biases, the loss L, in turn, is a function of (\vec{w}, \vec{b}). The average loss across all data points is denoted by $G(\vec{w}, \vec{b})$ which we want to minimize.</p> <p>Now that we have defined the problem statement of backpropagation, let's specify all the parts of this network in detail, i.e. the number of parameters, hyperparameters etc.</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

 <p>The network that we'll use to implement backpropagation is shown below:</p>  <p>Topology</p>	<p>We have an input $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.</p> <p>We have three weight matrices and biases as follows:</p> $W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \end{bmatrix}$ $W^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} \\ w_{21}^{[2]} & w_{22}^{[2]} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \end{bmatrix}$ $W^{[o]} = \begin{bmatrix} w_{11}^{[o]} & w_{12}^{[o]} \\ w_{21}^{[o]} & w_{22}^{[o]} \\ w_{31}^{[o]} & w_{32}^{[o]} \end{bmatrix} \quad b^{[o]} = \begin{bmatrix} b_1^{[o]} \\ b_2^{[o]} \\ b_3^{[o]} \end{bmatrix}$
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The last weight matrix $W^{[o]}$ and $b^{[o]}$ (of the output layer) are of a different dimension than the rest. These can also be written as $W^{[3]}$ and $b^{[3]}$ respectively. Even though the last layer does not have a bias, we have defined it here. The only purpose is to show the difference in notations. In general, the last layer does have a bias vector but for the problem setting here, we are not considering the bias vector for the last layer.

$$\vec{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

The output from the last layer is a vector \vec{p} . This shall be compared with the ground truth

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

vector y which can be either of the three one-hot vectors:

Backpropagation

Say you have m data points in the training set. Arrange the following three computation steps as they appear in one iteration of the backpropagation algorithm:

1. Feedforward the i th data point
2. Compute the gradient of loss with respect to weights and biases
3. Update the weights and biases
4. Compute the loss of the i th data point
5. Aggregate (compute the average of) m losses

☐ 1, 4, 2, 5, 3

☒ 1, 4, 5, 2, 3

Q Feedback :

The data points are first fed forward, the loss of each data point is computed, then aggregated (to compute (average loss)), then the gradient of loss is computed, and finally weights and biases are updated once.

☐ 1, 5, 4, 2, 3

☐ 1, 4, 5, 3, 2

Next, let's learn how the propagation of gradient happens from the output layer towards the left.

Updating the Weights and Biases - II

In the previous segment, we defined the architecture of the network, the dimensions of the weight and bias matrices etc. Let's now perform the first two steps before we can propagate the gradients in the backward direction:

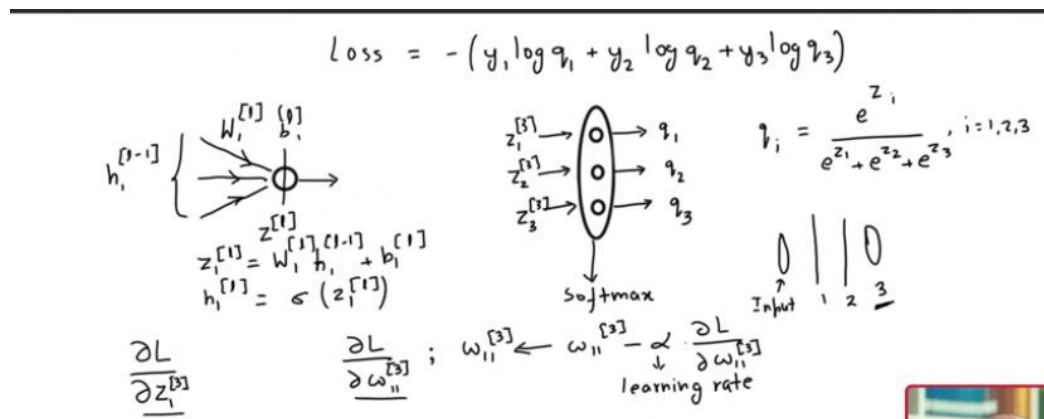
1. Forward propagation (feedforward)
2. Defining the cost/loss function

<p> $h^{[1]} = \sigma(W^{[1]} \cdot \vec{x} + b^{[1]})$ $h^{[2]} = \sigma(W^{[2]} \cdot h^{[1]} + b^{[2]})$ $\vec{p} = \text{normalized}(\exp(W^{[3]} \cdot h^{[2]}))$ </p> <p> cross entropy Loss: $-(y_1 \log p_1 + y_2 \log p_2 + y_3 \log p_3) \leftarrow \text{minimize}$ </p>	<p>Let's summarize the forward propagation equations for our network:</p> <ol style="list-style-type: none"> 1. $h^1 = \sigma(W^1 \cdot x + b^1)$ 2. $h^2 = \sigma(W^2 \cdot h^1 + b^2)$ 3. $p_i = \text{normalized}(e^{W^3 \cdot h^2})$ <p>A commonly used loss function for classification problems is the cross-entropy loss. It is defined as follows:</p> <p>C.E Loss = $-y^T \cdot \log(p) = -(y_1 \log(p_1) + y_2 \log(p_2) + y_3 \log(p_3))$</p> <p>Note that $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$ and $p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$</p> <p>Hence, $\log(p) = \begin{bmatrix} \log(p_1) \\ \log(p_2) \\ \log(p_3) \end{bmatrix}$</p> <p>Thus, the cross-entropy loss can be written concisely as:</p> <p>$-y^T \cdot \log(p) = -[y_1 \ y_2 \ y_3] \cdot \begin{bmatrix} \log(p_1) \\ \log(p_2) \\ \log(p_3) \end{bmatrix} = -(y_1 \log(p_1) + y_2 \log(p_2) + y_3 \log(p_3))$</p> <p>where y^T is the transpose of y.</p> <p>You also understood how minimising this loss function makes sense. Let's try to understand that better through the following questions:</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>C.E Loss</p> <p>The cross-entropy is defined as C.E Loss = $-(y_1 \log(p_1) + y_2 \log(p_2) + y_3 \log(p_3))$. Let us assume the following values</p> <p>$y_1 = 1$. Hence, $y_2 = y_3 = 0$.</p> <p>$p_1 = 0.8, p_2 = 0.1$ and $p_3 = 0.1$</p> <p>What is the value of the C.E loss (upto 1 decimal place)?</p> <p>Note that the log here is wrt to $e = 2.718$.</p> <p> <input checked="" type="radio"/> 0.2 ✓ Feedback: $-1 \ln(0.8) = 0.2$ </p> <p> <input type="radio"/> 0.3 </p>	<p>C.E Loss</p> <p>The Cross-Entropy(C.E) Loss is = $-(y_1 \log(p_1) + y_2 \log(p_2) + y_3 \log(p_3))$. Let us assume the following values</p> <p>$y_1 = 1$. Hence, $y_2 = y_3 = 0$.</p> <p>$p_1 = 0.1, p_2 = 0.1$ and $p_3 = 0.8$</p> <p>What is the value of the C.E loss?(upto 1 decimal places)</p> <p>Note that the log here is wrt to $e = 2.718$.</p> <p> <input checked="" type="radio"/> 2.3 ✓ Feedback: $-1 \ln(0.1) = 2.3$ </p> <p> <input type="radio"/> 2.4 </p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The cross-entropy loss is designed such that when the predicted probability is close to the ground truth, the loss value is close to zero, and vice-versa.

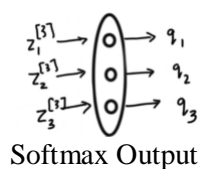
Now, before we move to the next step, let us introduce a new terminology which will make the notations much easier and neater while computing the gradients:



So, the secret element z^l represents the cumulative input into a neuron of layer l :

$$Z^l = W^l \cdot h^{l-1} + b^l$$

We are using the variable z mainly to simplify the gradient computation notations. Having defined z^l , let's now write the output of the last layer in terms of z :



This is a softmax layer with three neurons and gets the following inputs:

$$z^3 = \begin{bmatrix} z_1^3 \\ z_2^3 \\ z_3^3 \end{bmatrix}$$

Thus, the output from the softmax layer in terms of z is $q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$ where:

$$q_1 = \frac{e^{z_1^3}}{e^{z_1^3} + e^{z_2^3} + e^{z_3^3}} \quad q_2 = \frac{e^{z_2^3}}{e^{z_1^3} + e^{z_2^3} + e^{z_3^3}} \quad q_3 = \frac{e^{z_3^3}}{e^{z_1^3} + e^{z_2^3} + e^{z_3^3}}$$

Please note that the superscript '3' in the above expressions stands for the layer number.

We see that for every element of the weight matrix W^3 (for e.g. w_{11}^3), we need to find $\frac{\partial L}{\partial w_{11}^3}$ and update w_{11}^3 according to $w_{11}^3 = w_{11}^3 - \alpha \cdot \frac{\partial L}{\partial w_{11}^3}$. In order to calculate $\frac{\partial L}{\partial w_{11}^3}$, we need to perform a series of steps. You'll study those in the next segment.

Backpropagation Notations

Mark all correct statements about the intermediate variable z :

☒ z^l denotes the cumulative input coming into the layer l ✓

Q Feedback :

z^l denotes the cumulative input going into the layer l , which is computed by multiplying the output of layer $l - 1$ with the weights and adding the bias. The output of layer l is simply the activation function applied to z^l .

☐ z^l denotes the cumulative output from the layer l

☒ $h^l = \sigma(z^l)$ ✓

Q Feedback :

z^l denotes the cumulative input going into the layer l , which is computed by multiplying the output of layer $l - 1$ with the weights and adding the bias. The output of layer l is simply the activation function applied to z^l .

☒ $z^l = W^l \cdot h^{l-1} + b^l$ ✓

Q Feedback :

z^l denotes the cumulative input going into the layer l , which is computed by multiplying the output of layer $l - 1$ with the weights and adding the bias. The output of layer l is simply the activation function applied to z^l .

☐ $z^l = W^l \cdot h^l + b^l$

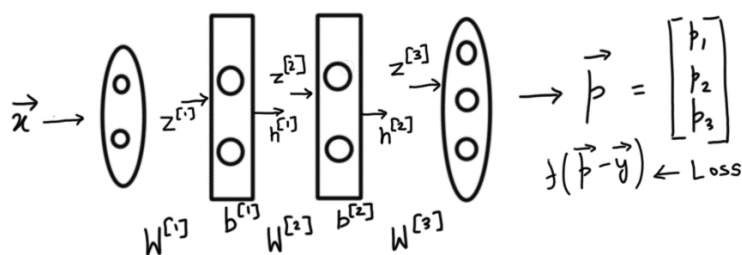
Updating the Weights and Biases - III

Our main objective is to compute the gradient of the loss function wrt the weights $\partial L / \partial W$ so that we can update the weights using gradient descent. The expression for the (cross-entropy) loss L for one data point is:

$$L = -(y_1 \log(p_1) + y_2 \log(p_2) + y_3 \log(p_3))$$

Let's first consider the weights of only the last (output) layer and try to compute the gradient $\partial L / \partial W^3$. Think about how you would compute $\partial L / \partial W^3$ and the challenges in computing it.

We do not have an expression directly relating L with W^3 , so computing $\partial L / \partial W^3$ is not straightforward. But we do have expressions relating L with p , p with z^3 , z^3 with W^3 , W^3 with h^2 and so on. We'll use this important observation to compute the gradient of L with *all* the weights in the network.



Flow

We used p and q interchangeably to denote the network output. Let's first compute the gradient of the output p (or q) with respect to z_3 , which we will need later to compute further gradients.

NEURAL NETWORKS
Backpropagation

$$Loss = -(y_1 \log q_1 + y_2 \log q_2 + y_3 \log q_3)$$

$z_i^{[1]} = W_i^{[1]} h_i^{[1-1]} + b_i^{[1]}$
 $h_i^{[1]} = \sigma(z_i^{[1]})$

softmax

$q_i = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + e^{z_3}}, i=1,2,3$
 $\leftarrow N$

Input 1 2 3

$\frac{\partial L}{\partial z_1^{[3]}} = \frac{e^{z_1^{[3]}}}{N} - \frac{e^{z_1^{[3]}}}{N^2} \cdot \frac{\partial N}{\partial z_1^{[3]}} = \frac{e^{z_1^{[3]}}}{N} - \frac{e^{z_1^{[3]}} 2}{N^2} = q_1 - q_1^2 = q_1(1 - q_1)$

$\frac{\partial q_2}{\partial z_1^{[3]}} = -\frac{e^{z_2^{[3]}}}{N^2} \cdot e^{z_1^{[3]}} = -q_1 \cdot q_2$

$\frac{\partial L}{\partial \omega_{11}^{[3]}} ; \omega_{11}^{[3]} \leftarrow \omega_{11}^{[3]} - \alpha \cdot \frac{\partial L}{\partial \omega_{11}^{[3]}}$
 learning rate

Thus, we have the following equations relating the gradient of q with respect to z_1^3 :

$$\frac{\partial q_1}{\partial z_1^3} = q_1(1 - q_1) \quad \frac{\partial q_2}{\partial z_1^3} = -q_1 q_2 \quad \frac{\partial q_3}{\partial z_1^3} = -q_1 q_3$$

Similarly, we can write expressions of the gradient of q with respect to z_2^3 and z_3^3 . Thus, we now know the expression for $\frac{\partial q}{\partial z^3}$.

You might wonder how is this going to help us in calculating $\frac{\partial L}{\partial z^3}$. Let's consider $\frac{\partial L}{\partial z_1^3}$. We know that $L = -(y_1 \log(q_1) + y_2 \log(q_2) + y_3 \log(q_3))$. This is where we use the **chain rule of differentiation**:

$$\frac{\partial L}{\partial z_1^3} = -\left(\frac{\partial(y_1 \log(q_1))}{\partial q_1} \frac{\partial q_1}{\partial z_1^3} + \frac{\partial(y_2 \log(q_2))}{\partial q_2} \frac{\partial q_2}{\partial z_1^3} + \frac{\partial(y_3 \log(q_3))}{\partial q_3} \frac{\partial q_3}{\partial z_1^3}\right)$$

Since we already know $\frac{\partial q_i}{\partial z_1^3}$ for $i = 1, 2, 3$, we'll get $\frac{\partial L}{\partial z_1^3}$. Try doing this before the professor demonstrates it in the next lecture.

Gradient Computation

Looking at the expression written using the chain rule,

$$\frac{\partial L}{\partial z_1^3} = -\left(\frac{\partial(y_1 \log(q_1))}{\partial q_1} \frac{\partial q_1}{\partial z_1^3} + \frac{\partial(y_2 \log(q_2))}{\partial q_2} \frac{\partial q_2}{\partial z_1^3} + \frac{\partial(y_3 \log(q_3))}{\partial q_3} \frac{\partial q_3}{\partial z_1^3}\right)$$

what is the expression for $\frac{\partial(y_i \log(q_i))}{\partial q_i}$, $i = 1, 2, 3$ is:

☐ $-\frac{y_i}{q_i}$

☐ $\frac{y_i}{q_i}$

Feedback:
 $\frac{\partial(y_i \log(q_i))}{\partial q_i} = \frac{y_i}{q_i}$. The derivative of $\log(x)$ is $\frac{1}{x}$.

☒ $-y_i \log(q_i)$

Feedback:
The derivative of $\log(x)$ is $\frac{1}{x}$.

Let's now use the chain rule to compute the gradient of loss L with respect to z_3 , i.e. $\partial L / \partial z^3$:

$$\begin{aligned}
\mathcal{L} &= -(y_1 \log q_1 + y_2 \log q_2 + y_3 \log q_3) \leftarrow \\
\frac{\partial q_1}{\partial z_1^{[3]}} &= q_1(1-q_1), \quad \frac{\partial q_2}{\partial z_1^{[3]}} = -q_1 q_2, \quad \frac{\partial q_3}{\partial z_1^{[3]}} = -q_1 q_3 \\
\frac{\partial \mathcal{L}}{\partial z_1^{[3]}} &= -\left(y_1 \frac{1}{q_1} \cdot \frac{\partial q_1}{\partial z_1^{[3]}} + y_2 \frac{1}{q_2} \cdot \frac{\partial q_2}{\partial z_1^{[3]}} + y_3 \frac{1}{q_3} \cdot \frac{\partial q_3}{\partial z_1^{[3]}}\right) \\
&= -\left(y_1 \frac{1}{q_1} q_1(1-q_1) + y_2 \frac{1}{q_2} (-q_1 q_2) + y_3 \frac{1}{q_3} (-q_1 q_3)\right) \\
&= -(y_1 - (y_1 q_1 + y_2 q_1 + y_3 q_1)) \\
&= -(y_1 - q_1(y_1 + y_2 + y_3)) = q_1 - y_1 \\
\frac{\partial \mathcal{L}}{\partial z_i^{[3]}} &= q_i - y_i \quad ; i=1,2,3 \\
\Delta z^{[3]} &= \frac{\partial \mathcal{L}}{\partial z^{[3]}} = \vec{q} - \vec{y}
\end{aligned}$$



Thus, we have now computed the gradient of \mathcal{L} with respect to z^3 , i.e. $\partial \mathcal{L} / \partial z^3$.

In computations until now, loss function $\mathcal{L} = -(y_1 \log(q_1) + y_2 \log(q_2) + y_3 \log(q_3))$ using which we computed $\partial \mathcal{L} / \partial z^3 = (\partial \mathcal{L} / \partial q) (\partial q / \partial z^3)$. Since we had already computed $\partial q / \partial z^3$ earlier, we used the expression for $\partial \mathcal{L} / \partial q$ to get:

$$\frac{\partial \mathcal{L}}{\partial z_1^3} = -\left(\frac{y_1}{q_1} \frac{\partial q_1}{\partial z_1^3} + \frac{y_2}{q_2} \frac{\partial q_2}{\partial z_1^3} + \frac{y_3}{q_3} \frac{\partial q_3}{\partial z_1^3}\right)$$

Substituting for $\partial q / \partial z^3$, we get the following simplified expression:

- $\partial \mathcal{L} / \partial z^3_1 = q_1 - y_1$
- $\partial \mathcal{L} / \partial z^3_2 = q_2 - y_2$
- $\partial \mathcal{L} / \partial z^3_3 = q_3 - y_3$

This can be written in a consolidated vector form as:

$$dz^3 = \frac{\partial \mathcal{L}}{\partial z^3} = \vec{q} - \vec{y}$$

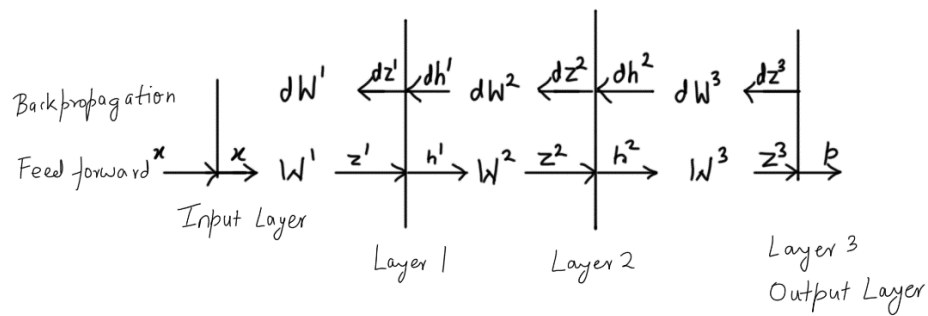
where $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$, $\vec{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$ and $z^3 = \begin{bmatrix} z_1^3 \\ z_2^3 \\ z_3^3 \end{bmatrix}$

Having calculated $\partial \mathcal{L} / \partial z^3$, we can now see how this will help us in calculating, $\partial \mathcal{L} / \partial W^3$ or dW^3 :

$$\frac{\partial \mathcal{L}}{\partial W^3} = \frac{\partial \mathcal{L}}{\partial z^3} \frac{\partial z^3}{\partial W^3}$$

We have defined $\partial \mathcal{L} / \partial z^3$ as dz^3 . We shall follow this representation in the following segments, i.e. gradient of loss with respect to any variable X is $\partial \mathcal{L} / \partial X = dX$.

The general strategy to compute the gradients in a backward direction is as shown:



Backprop

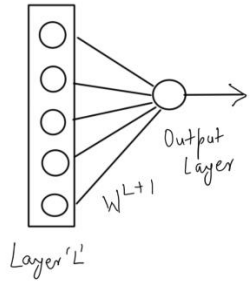
Gradients are calculated in a backward direction starting from dz^3 . Hence, we'll calculate the gradients in the following sequence (dX is short for $\partial L / \partial X$):

1. dz^3
2. dW^3
3. dh^2
4. dz^2
5. dW^2
6. dh^1
7. dz^1
8. dW^1

Process is known as **backpropagation** - we propagate the gradients in a backward direction starting from the output layer. In the next segment, practice computing the gradients for a sigmoid activation function.

Sigmoid Backpropagation

Sigmoid outputs are used for binary classification problems. A sigmoid output layer has only one neuron as shown below. In the following questions, practice computing these gradients in the backward direction: dL/dp , dp/dz^{L+1} and using the previous two expressions, dL/dz^{L+1} .

 <p style="text-align: center;">Sigmoid Layer</p>	<p>Layer Dimension</p> <p>What is the dimension of the weight matrix shown, W^{L+1}?</p> <p><input checked="" type="radio"/> (1, 5)</p> <p>Q Feedback: Dimension of weight matrix = (number of neurons in layer L, number of neurons in layer L-1)</p> <p><input type="radio"/> (5, 1)</p>
------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

We have already seen that the cumulative input into the sigmoid neuron is:

$$z^{L+1} = w_1^{L+1}h_1^L + w_2^{L+1}h_2^L + w_3^{L+1}h_3^L + w_4^{L+1}h_4^L + w_5^{L+1}h_5^L$$

The sigmoid output is given by $p = \frac{1}{1+e^{-z^{L+1}}}$.

The loss is the same cross entropy loss defined as $L = -(y \log(p) + (1 - y) \log(1 - p))$, where y is ground truth label.

Let's now calculate $dz^{L+1} = \frac{dL}{dz^{L+1}}$. Using the chain rule, we know that $\frac{dL}{dz^{L+1}} = \frac{dL}{dp} \cdot \frac{dp}{dz^{L+1}}$.

<p>Gradient Calculation</p> <p>Looking at the below expression, what is $\frac{dL}{dp}$?</p> $L = -(y \log(p) + (1 - y) \log(1 - p))$ <p><input type="radio"/> $p(1 - p)$</p> <p><input checked="" type="radio"/> $\frac{1-y}{1-p} - \frac{y}{p}$</p> <p>Q Feedback: There is a negative sign in the loss function. Hence $-(\frac{y}{p} - \frac{1-y}{1-p}) = \frac{1-y}{1-p} - \frac{y}{p}$.</p> <p><input type="radio"/> $\frac{y}{p} - \frac{1-y}{1-p}$</p> <p><input type="radio"/> $1 - p$</p>	<p>Gradient Calculation</p> <p>Looking at the below expression, what is $\frac{dp}{dz^{L+1}}$?</p> $p = \frac{1}{1+e^{-z^{L+1}}}$ <p><input type="radio"/> $\frac{y}{p} - \frac{1-y}{1-p}$</p> <p><input type="radio"/> $1 - p$</p> <p><input checked="" type="radio"/> $p(1 - p)$</p> <p>Q Feedback: $\frac{dp}{dz^{L+1}} = -e^{-z^{L+1}} \frac{-1}{(1+e^{-z^{L+1}})^2} = \frac{e^{-z^{L+1}}}{1+e^{-z^{L+1}}} \cdot \frac{1}{1+e^{-z^{L+1}}} = (1 - \frac{1}{1+e^{-z^{L+1}}}) \cdot \frac{1}{1+e^{-z^{L+1}}} = p(1 - p)$</p> <p><input type="radio"/> $\frac{1-y}{1-p} - \frac{y}{p}$</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Gradient Calculation

We know that $\frac{dL}{dz^{L+1}} = \frac{dL}{dp} \frac{dp}{dz^{L+1}}$. You have already computed the two terms on the RHS. The expression for $\frac{dL}{dz^{L+1}}$ is:

☐ $p - y$

Q Feedback :

Simply multiply the two expression computed in the previous two questions. You will get

$$\left(\frac{1-y}{1-p} - \frac{y}{p}\right)p(1-p) = p - y.$$

☒ $p(1-p) + y(1-y)$

Q Feedback :

Simply multiply the two expression computed in the previous two questions. You will get

$$\left(\frac{1-y}{1-p} - \frac{y}{p}\right)p(1-p) = p - y.$$

You saw that on substituting the values $\frac{dL}{dp}$, $\frac{dp}{dz^{L+1}}$, you get:

$$\frac{dL}{dz^{L+1}} = \left(\frac{1-y}{1-p} - \frac{y}{p}\right)p(1-p) = p - py - y + py = p - y$$

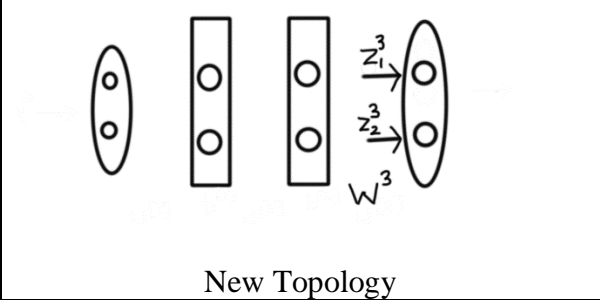
This is the same as what we had arrived at in the previous segment for a softmax layer.

Updating the Weights and Biases - IV

Until now, we have calculated the gradients $\partial L / \partial z^3$ which will now help us in calculating $\partial L / \partial w^3$, i.e. propagating the gradient backwards. Backpropagation strategy is to use the gradients of L wrt the variables towards the right side of the network to compute the gradient wrt the variables towards the left.

In the previous segment, we had calculated the gradients of the softmax layer with three outputs. To simplify the notations a little, from now on we'll use a softmax with only two outputs.

The gradient calculation of the softmax layer gets a bit too messy working with three outputs. To simplify the calculations of the gradients to the left of this layer, we shall consider the following network from now on:



The only difference here is that the last layer now has two neurons instead of three. This won't affect the backpropagation algorithm (in principle) at all.

With the new network, we have the following for the last layer:

$$W^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 \\ w_{21}^3 & w_{22}^3 \end{bmatrix} \text{ and } z^3 = \begin{bmatrix} z_1^3 \\ z_2^3 \end{bmatrix}$$

Let's now see how we can compute $\partial L / \partial w^3$ (the gradient of loss with respect to the last layer's weights) using the gradients we have computed until now. Computing $\partial L / \partial w^3$ will enable the network to make its first weight updates (to the weights of the last layer).

NEURAL NETWORKS
Backpropagation Up

$$\vec{dz}^{[3]} = \vec{q} - \vec{y}$$

$$z_1^{[3]} = w_{11}^{[3]} \cdot h_1^{[2]} + w_{12}^{[3]} \cdot h_2^{[2]} + b_1^{[3]}$$

$$z_2^{[3]} = w_{21}^{[3]} \cdot h_1^{[2]} + w_{22}^{[3]} \cdot h_2^{[2]} + b_2^{[3]}$$

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \left[\frac{\partial \mathcal{L}}{\partial w_{ij}^{[3]}} \right]$$

$$\frac{\partial \mathcal{L}}{\partial w_{11}^{[3]}} = \frac{\partial \mathcal{L}}{\partial z_1^{[3]}} \cdot \frac{\partial z_1^{[3]}}{\partial w_{11}^{[3]}} + \frac{\partial \mathcal{L}}{\partial z_2^{[3]}} \cdot \frac{\partial z_2^{[3]}}{\partial w_{11}^{[3]}}$$

$$= dz_1^{[3]} \cdot \frac{\partial z_1^{[3]}}{\partial w_{11}^{[3]}} + dz_2^{[3]} \cdot \frac{\partial z_2^{[3]}}{\partial w_{11}^{[3]}} \rightarrow 0$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial w_{11}^{[3]}} = dz_1^{[3]} \cdot h_1^{[2]}$$


$$\rightarrow \frac{\partial \mathcal{L}}{\partial w_{12}^{[3]}} = dz_1^{[3]} \cdot \frac{\partial z_1^{[3]}}{\partial w_{12}^{[3]}} + dz_2^{[3]} \cdot \frac{\partial z_2^{[3]}}{\partial w_{12}^{[3]}}$$

$$= dz_1^{[3]} \cdot h_2^{[2]}$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial w_{21}^{[3]}} = dz_2^{[3]} \cdot h_1^{[2]} \rightarrow \frac{\partial \mathcal{L}}{\partial w_{22}^{[3]}} = dz_2^{[3]} \cdot h_2^{[2]}$$

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \begin{bmatrix} dz_1^{[3]} \\ dz_2^{[3]} \end{bmatrix} \begin{bmatrix} h_1^{[2]} & h_2^{[2]} \end{bmatrix}$$

$$= \vec{dz}^{[3]} \cdot (\vec{h}^{[2]})^T$$



<p>Thus, we computed $\frac{\partial L}{\partial W^3}$ which is a matrix representing the gradient of L with respect to each of the weights in W^3:</p> $\frac{\partial L}{\partial W^3} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^3} & \frac{\partial L}{\partial w_{12}^3} \\ \frac{\partial L}{\partial w_{21}^3} & \frac{\partial L}{\partial w_{22}^3} \end{bmatrix}$ <p>To compute the quantity above, we used the chain rule of differentiation again:</p> $\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial z^3} \frac{\partial z^3}{\partial W^3}$ <p>In other words, we are using z^3 as the intermediary variable to compute $\frac{\partial L}{\partial W^3}$.</p> <p>We had already computed $\frac{\partial L}{\partial z^3}$, so now we computed $\frac{\partial z^3}{\partial W^3}$. Note that z_1^3 and z_2^3 each depend only on two weights of W^3 according to the expressions (1) and (2) below.</p> <p>Since the two components of z^3 are:</p> $z_1^3 = w_{11}^3 h_1^2 + w_{12}^3 h_2^2 + b_1^3 \quad (1),$ $z_2^3 = w_{21}^3 h_1^2 + w_{22}^3 h_2^2 + b_2^3 \quad (2),$ <p>we use these expressions to compute $\frac{\partial z^3}{\partial W^3}$.</p> <p>We wrote $\frac{\partial L}{\partial w_{11}^3}$ in terms of both components of z^3 as:</p> $\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial w_{11}^3} + \frac{\partial L}{\partial z_2^3} \cdot \frac{\partial z_2^3}{\partial w_{11}^3}.$	<p>Using equations (1) and (2) above with respect to w_{11}^3, we can see that:</p> $\frac{\partial z_1^3}{\partial w_{11}^3} = h_1^2$ $\frac{\partial z_2^3}{\partial w_{11}^3} = 0$ <p>Hence,</p> $\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial z_1^3} \cdot h_1^2 = dz_1^3 \cdot h_1^2$ <p>Similarly, $\frac{\partial L}{\partial w_{12}^3} = dz_1^3 \cdot h_2^2$ $\frac{\partial L}{\partial w_{21}^3} = dz_2^3 \cdot h_1^2$ $\frac{\partial L}{\partial w_{22}^3} = dz_2^3 \cdot h_2^2$</p> <p>These equations can be combined and written in a matrix form as:</p> $dW^3 = \frac{\partial L}{\partial W^3} = \begin{bmatrix} dz_1^3 \\ dz_2^3 \end{bmatrix} \begin{bmatrix} h_1^2 & h_2^2 \end{bmatrix} = dz^3 \cdot (h^2)^T$ <p>Having calculated dW^3, we can now use it to make updates to the weights W^3 using:</p> $W^3 = W^3 - \alpha \cdot \frac{\partial L}{\partial W^3}.$ <p>This essentially means performing the following update steps:</p> $w_{11}^3 = w_{11}^3 - \alpha \cdot \frac{\partial L}{\partial w_{11}^3}$ $w_{12}^3 = w_{12}^3 - \alpha \cdot \frac{\partial L}{\partial w_{12}^3}$ $w_{21}^3 = w_{21}^3 - \alpha \cdot \frac{\partial L}{\partial w_{21}^3}$ $w_{22}^3 = w_{22}^3 - \alpha \cdot \frac{\partial L}{\partial w_{22}^3}$
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

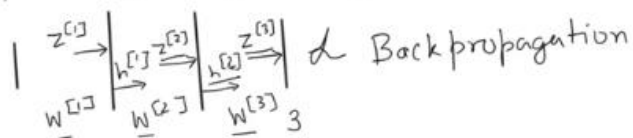
Although the math involved in reaching here was a bit complex, the expression for dW^3 is a simple matrix multiplication.

Now, to update the weights of the previous layer W^2 , we similarly need to compute $dW^2 = \partial L / \partial W^2$. To do that, we will follow the same strategy - compute the gradient with respect to an intermediary variable and use the chain rule.

<p>Intermediary Variables</p> <p>We have already computed the gradients with respect to W^3, i.e. $\frac{\partial L}{\partial W^3}$.</p> <p>Now, to update the weights of the previous layer W^2, we need to compute $dW^2 = \frac{\partial L}{\partial W^2}$. To do that, we will follow the same strategy - compute the gradient with respect to intermediary variables and use the chain rule of differentiation. Which intermediary variables do you think will be involved in reaching $\frac{\partial L}{\partial W^2}$?</p> <p>Choose all correct options (note that $dX = \frac{\partial L}{\partial X}$):</p> <div> <input checked="" type="checkbox"/> dh^2 ✓ Feedback: W^3 is directly related to h^2, h^2 is directly related to z^2, z^2 is directly related to W^2. </div> <div> <input type="checkbox"/> dh^1 </div> <div> <input checked="" type="checkbox"/> dz^2 ✓ Feedback: W^3 is directly related to h^2, h^2 is directly related to z^2, z^2 is directly related to W^2. </div> <div> <input type="checkbox"/> dz^1 </div>	<p>Thus, to compute the gradient with respect to W^2, we will use dh^2 and dz^2 as the intermediary variables.</p> <p>Let's now see how we can calculate $\partial L / \partial h^2$ or dh^2.</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

$$\Rightarrow \begin{aligned} z_1^{[3]} &= \omega_{11}^{[3]} \cdot h_1^{[2]} + \omega_{12}^{[3]} \cdot h_2^{[2]} + b_1^{[3]} \\ z_2^{[3]} &= \omega_{21}^{[3]} \cdot h_1^{[2]} + \omega_{22}^{[3]} \cdot h_2^{[2]} + b_2^{[3]} \end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial \omega_{11}^{[3]}}$$



$$\begin{aligned} \rightarrow \frac{\partial \mathcal{L}}{\partial h_1^{[2]}} &= \frac{\partial \mathcal{L}}{\partial z_1^{[3]}} \cdot \frac{\partial z_1^{[3]}}{\partial h_1^{[2]}} + \frac{\partial \mathcal{L}}{\partial z_2^{[3]}} \cdot \frac{\partial z_2^{[3]}}{\partial h_1^{[2]}} \\ &= \delta z_1^{[3]} \cdot \omega_{11}^{[3]} + \delta z_2^{[3]} \cdot \omega_{21}^{[3]} \end{aligned}$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial h_2^{[2]}} = \delta z_1^{[3]} \cdot \omega_{12}^{[3]} + \delta z_2^{[3]} \cdot \omega_{22}^{[3]}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{[2]}} = (\boldsymbol{\omega}^{[3]})^T \cdot \delta \mathbf{z}^{[3]}$$

To calculate $\frac{\partial \mathcal{L}}{\partial h^2}$, we use the chain rule of differentiation and the intermediary variable z^3 , i.e using $\frac{\partial \mathcal{L}}{\partial z^3}$. We already know the expression for z^3 in terms of h^2 from equations (1) and (2) above.

We write dh^2 in terms of its two components as follows:

$$\frac{\partial \mathcal{L}}{\partial h_1^2} = \frac{\partial \mathcal{L}}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial h_1^2} + \frac{\partial \mathcal{L}}{\partial z_2^3} \cdot \frac{\partial z_2^3}{\partial h_1^2} \quad (3)$$

$$\frac{\partial \mathcal{L}}{\partial h_2^2} = \frac{\partial \mathcal{L}}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial h_2^2} + \frac{\partial \mathcal{L}}{\partial z_2^3} \cdot \frac{\partial z_2^3}{\partial h_2^2} \quad (4)$$

Differentiating equations (1) and (2) with respect to h_1^2 , we have $\frac{\partial z_1^3}{\partial h_1^2} = w_{11}^3$ and $\frac{\partial z_2^3}{\partial h_1^2} = w_{21}^3$.

Similarly, differentiating equations (1) and (2) with respect to h_2^2 , we have $\frac{\partial z_1^3}{\partial h_2^2} = w_{12}^3$ and $\frac{\partial z_2^3}{\partial h_2^2} = w_{22}^3$.

Hence, we can write equations (3) and (4) as:

$$\begin{aligned}\frac{\partial L}{\partial h_1^2} &= dz_1^3 \cdot w_{11}^3 + dz_2^3 \cdot w_{21}^3 \\ \frac{\partial L}{\partial h_2^2} &= dz_1^3 \cdot w_{21}^3 + dz_2^3 \cdot w_{22}^3\end{aligned}$$

This can be written in a sweet little matrix form as:

$$\frac{\partial L}{\partial h^2} = dh^2 = (W^3)^T \cdot dz^3$$

where,

$$W^3 = \begin{bmatrix} w_{11}^3 & w_{12}^3 \\ w_{21}^3 & w_{22}^3 \end{bmatrix} \text{ and } dz^3 = \begin{bmatrix} dz_1^3 \\ dz_2^3 \end{bmatrix}$$

Backpropagation Generalisation

We have computed $\frac{\partial L}{\partial h^2} = (W^3)^T \cdot dz^3$, i.e. we have expressed the gradient of loss with respect of the output of the second layer in terms of the weight matrix following that layer and the input to the next layer. This pattern can be extended to any layer further back in the network.

Using this observation, what do you think is the expression for $\frac{\partial L}{\partial h^1}$?

☐ $(W^2)^T \cdot dz^3$

☒ $(W^2)^T \cdot dz^2$



 Feedback :

The gradient of loss with respect of the output of the nth layer depends on the weight matrix following that layer and the input to the next layer.

☐ $(W^1)^T \cdot dz^1$

Now that we have calculated dh^2 , the next step is to calculate dz^2 , i.e. propagate the gradient further back. Let's see how we do this in the next segment.

Updating the Weights and Biases - V

In the previous segment, we calculated dh^2 . In order to calculate dW^2 to calculate the updates to the weight W^2 , we need to calculate dz^2 .

NEURAL NETWORKS
Backpropagation

$$\begin{aligned} \rightarrow d\vec{z}^{[3]} &= \vec{y} - \hat{y} \\ \rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[3]}} &= d\vec{z}^{[3]} \cdot (h^{[2]})^T \\ \rightarrow d\vec{h}^{[2]} &= \frac{\partial \mathcal{L}}{\partial h^{[2]}} = (W^{[3]})^T \cdot d\vec{z}^{[3]} \\ \rightarrow d\vec{z}^{[2]} &= d\vec{h}^{[2]} \otimes \sigma'(z^{[2]}) \\ \rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[2]}} &= d\vec{z}^{[2]} \cdot (h^{[1]})^T \\ \rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[1]}} &= d\vec{z}^{[1]} \cdot (h^{[0]})^T \\ \rightarrow \omega^{[1]} &= \omega^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \omega^{[1]}} \end{aligned}$$

element wise product

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

$dh_1^{[2]} = \frac{\partial \mathcal{L}}{\partial h_1^{[2]}} \cdot \frac{\partial h_1^{[2]}}{\partial z_1^{[2]}} + \frac{\partial \mathcal{L}}{\partial h_2^{[2]}} \cdot \frac{\partial h_2^{[2]}}{\partial z_1^{[2]}}$
 $= dh_1^{[2]} \cdot \sigma'(z_1^{[2]})$
 $dz_2^{[2]} = dh_2^{[2]} \cdot \sigma'(z_2^{[2]})$

We computed the expression for the gradient $\partial \mathcal{L} / \partial z^2$. Thus, we now have expressions for the gradients (from right to left) dz^3 , dW^3 , dh^2 , dz^2 . All these expressions can now be generalised backwards to all the layers.

The process to compute dz^2 is summarised below. To compute dz^2 , we used h^2 as the intermediary variable. We used the relation:

$$h^2 = \sigma(z^2), \text{ i.e., } \begin{bmatrix} h_1^2 \\ h_2^2 \end{bmatrix} = \begin{bmatrix} \sigma(z_1^2) \\ \sigma(z_2^2) \end{bmatrix}$$

to arrive at the following expressions for $\frac{\partial \mathcal{L}}{\partial z^2}$:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_1^2} = dz_1^2 &= \frac{\partial \mathcal{L}}{\partial h_1^2} \cdot \frac{\partial h_1^2}{\partial z_1^2} + \frac{\partial \mathcal{L}}{\partial h_2^2} \cdot \frac{\partial h_2^2}{\partial z_1^2} = dh_1^2 \cdot \sigma'(z_1^2) \\ \text{since } \frac{\partial h_1^2}{\partial z_1^2} &= \sigma'(z_1^2) \text{ and } \frac{\partial h_2^2}{\partial z_1^2} = 0 \end{aligned}$$

$$\text{Similarly, } dz_2^2 = dh_2^2 \cdot \sigma'(z_2^2).$$

We can combine both dz_1^2 and dz_2^2 in a nice vector form as follows:

$$dz^2 = dh^2 \otimes \sigma'(z^2)$$

where the \otimes represents element-wise multiplication (also called the Hadamard product).

Comprehension: Gradients with Sigmoid Activation Function

$$z^2 = \begin{bmatrix} 2 \\ 1 \\ 3 \\ -1 \end{bmatrix}$$

$$\frac{1}{1+e^{-x}}$$

For a particular layer, you have z^2 . The activation function is the sigmoid function $\frac{1}{1+e^{-x}}$. Answer the following questions. Note that you will need to write some basic code for computing the answers.

<p>Output of the Layer</p> <p>Given the value of z^2, what is h^2? (upto 2 decimal places)</p> <p><input type="radio"/> $\begin{bmatrix} 0.781 \\ 0.531 \\ 0.753 \\ 0.469 \end{bmatrix}$</p> <p><input checked="" type="radio"/> $\begin{bmatrix} 0.881 \\ 0.731 \\ 0.953 \\ 0.269 \end{bmatrix}$</p> <p>Feedback : Applying the sigmoid function to all the elements, we have $\begin{bmatrix} 0.881 \\ 0.731 \\ 0.953 \\ 0.269 \end{bmatrix}$.</p>	<p>Differentiation of the sigmoid function</p> <p>You have already seen that $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. What is then $\sigma'(z^2)$? (upto 3 decimal places)</p> <p><input checked="" type="radio"/> $\begin{bmatrix} 0.105 \\ 0.197 \\ 0.045 \\ 0.197 \end{bmatrix}$</p> <p>Feedback : Substituting the values found in the previous question in the formula stated above, we get $\begin{bmatrix} 0.105 \\ 0.197 \\ 0.045 \\ 0.197 \end{bmatrix}$</p> <p><input type="radio"/> $\begin{bmatrix} 0.145 \\ 0.127 \\ 0.035 \\ 0.127 \end{bmatrix}$</p>
<p>Derivative of Loss wrt the cumulative input</p> <p>We know that $dz^2 = dh^2 \otimes \sigma'(z^2)$. Also, we have calculated $\sigma'(z^2) = \begin{bmatrix} 0.105 \\ 0.197 \\ 0.045 \\ 0.197 \end{bmatrix}$. Let us consider $dh^2 = \begin{bmatrix} 1 \\ -1 \\ 2 \\ -1 \end{bmatrix}$. What is dz^2?</p> <p><input checked="" type="radio"/> $\begin{bmatrix} 0.105 \\ -0.197 \\ 0.090 \\ -0.197 \end{bmatrix}$</p> <p>Feedback : Doing element-wise multiplication of the vectors $\sigma'(z^2)$ and dh^2, we get $\begin{bmatrix} 0.105 \\ -0.197 \\ 0.090 \\ -0.197 \end{bmatrix}$.</p> <p><input type="radio"/> $\begin{bmatrix} 0.105 \\ 0.197 \\ 0.090 \\ 0.197 \end{bmatrix}$</p>	

Having calculated dz^2 , we can now easily calculate dW^2 using the expression that we have already derived for $dW^3 = dz^3 \cdot (h^2)^T$.

By symmetry, we can write $dW^2 = dz^2 \cdot (h^1)^T$ and $dW^1 = dz^1 \cdot (h^0)^T = dz^1 \cdot (x)^T$.

Having derived dW^3 , dW^2 and dW^1 , all we need to do is update the weights using the formula:

$$W^i = W^i - \alpha \cdot \frac{\partial L}{\partial W^i} \text{ where } i \text{ is the layer number } = 1, 2, 3$$

We have been through so much trouble of calculating the gradients. But do we really need to do all this when modern libraries such as Tensorflow etc. provide convenient APIs?

NEURAL NETWORKS
Backpropagation UpGrad

$\rightarrow \vec{dZ}^{[3]} = \vec{y} - \hat{y}$
 $\rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[3]}} = \vec{dZ}^{[3]} \cdot (\vec{h}^{[2]})^T$
 $\rightarrow \vec{dh}^{[2]} = \frac{\partial \mathcal{L}}{\partial \vec{h}^{[2]}} = (\omega^{[3]})^T \cdot \vec{dZ}^{[3]}$
 $\rightarrow \vec{dZ}^{[2]} = \vec{dh}^{[2]} \otimes \sigma'(z^{[2]})$
 $\rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[2]}} = \vec{dZ}^{[2]} \cdot (\vec{h}^{[1]})^T$
 $\rightarrow \frac{\partial \mathcal{L}}{\partial \omega^{[1]}} = \vec{dZ}^{[1]} \cdot (\vec{h}^{[0]})^T$
 $\rightarrow \omega^{[1]} = \omega^{[1]} - \alpha \frac{\partial \mathcal{L}}{\partial \omega^{[1]}}$ (learning rate)

Diagram: $x \rightarrow h^{[0]} \rightarrow z^{[1]} \rightarrow h^{[1]} \rightarrow z^{[2]} \rightarrow h^{[2]} \rightarrow z^{[3]} \rightarrow \hat{y} \rightarrow \mathcal{L}$
 $z_1^{[2]} \quad h_1^{[2]} \quad h_1^{[2]} = \sigma(z_1^{[2]})$

$dZ_1^{[2]} = \frac{\partial \mathcal{L}}{\partial h_1^{[2]}} \cdot \frac{\partial h_1^{[2]}}{\partial z_1^{[2]}} + \frac{\partial \mathcal{L}}{\partial h_2^{[2]}} \cdot \frac{\partial h_2^{[2]}}{\partial z_1^{[2]}}$
 $= dh_1^{[2]} \cdot \sigma'(z_1^{[2]}) \cdot 0$
 $dZ_2^{[2]} = dh_2^{[2]} \cdot \sigma'(z_2^{[2]})$

element wise product
 $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \otimes \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$

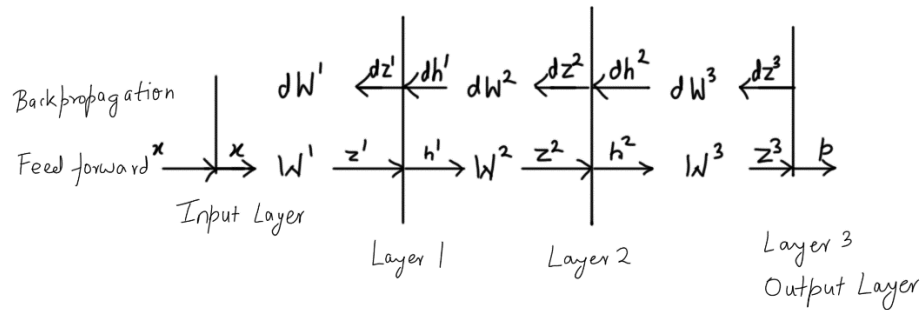
$x^2 \leftarrow \text{minimize w.r.t } x$
 $x \leftarrow x' - \alpha \cdot \frac{\partial x^2}{\partial x} \Big|_{x'} \Rightarrow x \leftarrow x' - \alpha \cdot 2x'$

It is important to conceptually understand how the backpropagation algorithm works, although in practice libraries such as Keras, Tensorflow etc. are used to train neural networks.

Let's summarize the steps done so far in the next segment.

Updating the Weights and Biases - VI

Let's summarize what we have computed until now for the network below.



Backprop

Recollect that earlier, we had stated the order in which we compute the backpropagation gradients. Let's write them in that order:

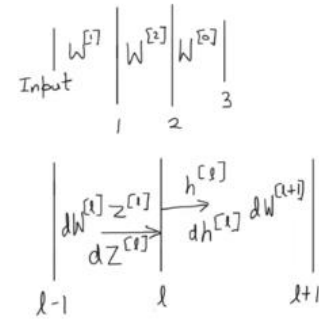
1. $dz^3 = q - y = p - y$	5. $dW^2 = dz^2 \cdot (h^1)^T$
2. $dW^3 = dz^3 \cdot (h^2)^T$	6. $dh^1 = (W^2)^T \cdot dz^2$
3. $dh^2 = (W^3)^T \cdot dz^3$	7. $dz^1 = dh^1 \otimes \sigma'(z^1)$
4. $dz^2 = dh^2 \otimes \sigma'(z^2)$	8. $dW^1 = dz^1 \cdot (x)^T$

We have written this for the specific network we have considered.

<p>Gradients For a particular layer number 5, what is the expression for dW^5?</p> <p><input type="radio"/> $dz^5 \cdot (h^5)^T$</p> <p><input checked="" type="radio"/> $dz^5 \cdot (h^4)^T$</p> <p>Q Feedback: It is $dz^L \cdot (h^{L-1})^T$ for a layer L.</p> <p><input type="radio"/> $dz^4 \cdot (h^4)^T$</p> <p><input type="radio"/> $dz^4 \cdot (h^5)^T$</p>	<p>Gradients For a particular layer number 7, what is the formula of dh^7?</p> <p><input type="radio"/> $(W^6)^T \cdot dz^6$</p> <p><input type="radio"/> $(W^7)^T \cdot dz^7$</p> <p><input checked="" type="radio"/> $(W^8)^T \cdot dz^8$</p> <p>Q Feedback: It is $(W^{L+1})^T \cdot dz^{L+1}$ for a layer L.</p> <p><input type="radio"/> $(W^7)^T \cdot dz^6$</p>	<p>Gradients For a particular layer number 4, what is the formula of dz^4?</p> <p><input type="radio"/> $dh^3 \otimes \sigma'(z^3)$</p> <p><input checked="" type="radio"/> $dh^4 \otimes \sigma'(z^4)$</p> <p>Q Feedback: It is $dh^l \otimes \sigma'(z^l)$ for a particular layer L.</p> <p><input type="radio"/> $dh^5 \otimes \sigma'(z^5)$</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Let's see how we generalise these formulae for a network with L hidden layers. The output layer is the $L+1$ -th layer and the weight and bias matrices for the same are denoted as W^0 and b^0 .

$$\begin{aligned}
 &\vec{x} \\
 &h^{[0]} = \vec{x} \\
 &\text{for } l \text{ in } [1, \dots, L]: \\
 &\quad h^{[l]} = \sigma(W^{[l]} \cdot h^{[l-1]} + b^{[l]}) \\
 &\quad p = \text{normalize}(\exp(W^{[0]} \cdot h^{[L]} + b^{[0]})) \\
 &\quad \mathcal{L} = -\vec{y}^T \cdot \log \vec{p} \quad \leftarrow \text{Cross Entropy Loss} \\
 &\quad dZ^{[0]} = \vec{p} - \vec{y} \quad \leftarrow \text{Specific to Cross Entropy Loss} \\
 &\quad dW^{[0]} = dZ^{[0]} \cdot (h^{[L]})^T \\
 &\quad \text{for } l \text{ in } [L, \dots, 1]: \\
 &\quad \quad d\vec{h}^{[l]} = (W^{[l+1]})^T \cdot d\vec{z}^{[l+1]} \\
 &\quad \quad d\vec{z}^{[l]} = d\vec{h}^{[l]} \otimes \sigma'(z^{[l]}) \quad \rightarrow \text{element wise} \\
 &\quad \quad dW^{[l]} = \frac{\partial \mathcal{L}}{\partial W^{[l]}} = d\vec{z}^{[l]} \cdot (h^{[l-1]})^T
 \end{aligned}$$



We went through one full trip through the network for a single data point, that is:

1. Feedforward
2. Define the loss function
3. Backpropagation

Let's write down the pseudocode of this algorithm:

<ol style="list-style-type: none"> 1. $h^0 = x$ 2. for l in $[1, 2, \dots, L]$: <ol style="list-style-type: none"> 1. $h^l = \sigma(W^l \cdot h^{l-1} + b^l)$ 3. $p = \text{normalize}(\exp(W^0 \cdot h^L + b^0))$ 4. $L = -y^T \cdot \log(p)$ 5. $dz^0 = p - y$ 	<ol style="list-style-type: none"> 6. $dW^0 = dz^0 \cdot (h^L)^T$ 7. for l in $[L, L-1, \dots, 1]$: <ol style="list-style-type: none"> 1. $dh^l = (W^{l+1})^T \cdot dz^{l+1}$ 2. $dz^l = dh^l \otimes \sigma'(z^l)$ 3. $dW^l = dz^l \cdot (h^{l-1})^T$
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


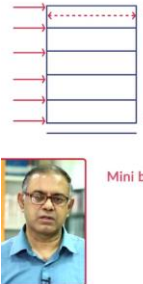
This is the consolidated algorithm for a single data point. We haven't yet updated any of the weights! We shall do the update step in while after processing the above steps for a **batch**, since making a single update for one data point will be extremely slow.

Let's go through that in the next segment.

Batch in Backpropagation

Until now, we worked with a single data point for doing feedforward and backpropagation. Doing this for a large number of training data points will be extremely inefficient.

This segment will modify the feedforward and backpropagation algorithms to work with **batches of multiple data points**.

Pseudo Code - Stochastic Gradient Descent UpGrad	
 <pre>Batch size = m for epoch = [1, ..., L] { Reshuffle the data set number of batch = $\frac{n}{m}$ for batch = [1, ..., number of batches] { Compute gradients for each input in the batch [batch_{1,1} x m, batch_{1,m} x m] Average gradient $\nabla w = \frac{\text{Sum of gradient } \nabla w}{m}$ Average gradient $\nabla b = \frac{\text{Sum of gradient } \nabla b}{m}$ $w = w - \lambda \nabla w$ $b = b - \lambda \nabla b$ } }</pre>	 <pre>Batch size = m for epoch = [1, ..., L] { Reshuffle the data set number of batch = $\frac{n}{m}$ for batch = [1, ..., number of batches] { Compute gradients for each input in the batch [batch_{1,1} x m, batch_{1,m} x m] Average gradient $\nabla w = \frac{\text{Sum of gradient } \nabla w}{m}$ Average gradient $\nabla b = \frac{\text{Sum of gradient } \nabla b}{m}$ $w = w - \lambda \nabla w$ $b = b - \lambda \nabla b$ } }</pre>

For updating weights and biases using plain backpropagation, we need to scan through the entire data set to make a single update to the weights. This is computationally very expensive for large datasets. Thus, we use multiple batches (or **mini-batches**) of data points, compute the **average gradient** for a batch, and update the weights based on that gradient.

But there is a danger in doing this - we are making weight updates based only on gradients computed for small batches, not the entire training set. Thus, we make **multiple passes** through the entire training set using **epochs**. An **epoch is one pass** through the entire training set, and we use multiple epochs (typically 10, 20, 50, 100 etc.) while training. In each epoch, we **reshuffle** all the data points, divide the reshuffled set into m batches, and update weights based on gradient of each batch.

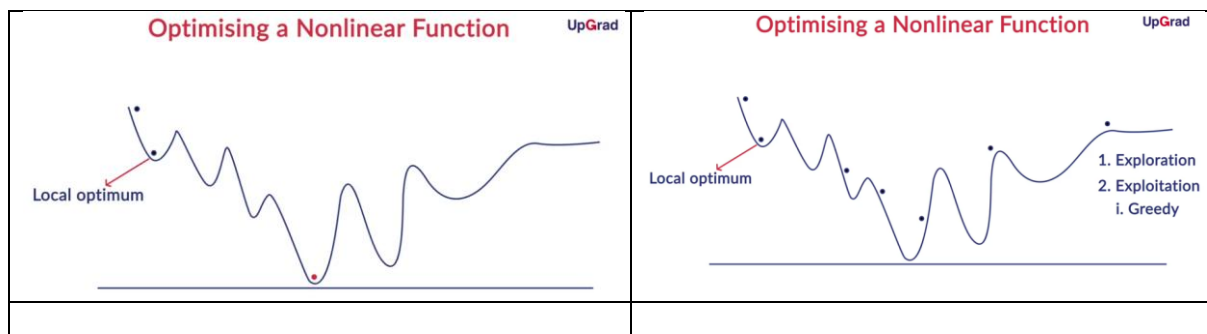
This training technique is called **stochastic gradient descent**, commonly abbreviated as **SGD**.

In most libraries such as Tensorflow, the **SGD training procedure** is as follows:

- Specify the number of epochs (typical values are 10, 20, 50, 100 etc.) - more epochs require more computational power
- Specify the number of batches m (typical values are 32, 64, 128, etc.)
- At the start of each epoch, the data set is **reshuffled** and divided into m batches.
- The **average gradient of each batch** is then used to make a weight update.
- The training is complete at the end of all the epochs.

<p>Number of batches</p> <p>Consider a dataset of 1,00,000 data points. You decide to perform a mini batch/ stochastic gradient descent with the batch size of 50. How many batches will be there?</p> <p><input type="radio"/> 1,00,000</p> <p><input type="radio"/> 1</p> <p><input checked="" type="radio"/> 2000</p> <p>Feedback: The number of batches = $n/m = 100000/50 = 2000$</p>	<p>Number of Updates</p> <p>Consider a dataset of 1,00,000 data points. You decide to perform a mini batch/ stochastic gradient descent with the batch size of 50 and for 3 epochs. How many updates will you make at the end of 3 epochs?</p> <p><input type="radio"/> 1</p> <p><input checked="" type="radio"/> 6000</p> <p>Feedback: You make number of epochs X number of mini batch updates = $3 \times 2000 = 6000$</p> <p><input type="radio"/> 2000</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Apart from being computationally faster, the SGD training process helps to reach the **global minima** (instead of being stuck at a **local minima**).



To avoid the problem of getting stuck at a local optimum, we need to strike a balance between **exploration** and **exploitation**.

Exploration tries to minimise the loss function with different starting points of W and b , i.e., initialise W and b with different values. **Exploitation** tries to reach the global minima starting from a W and b and do not explore the terrain at all. That may lead to the lowest point locally, but not necessarily the global minimum.

Having understood batches and how they help in efficient training, let's now write the pseudocode for batch training in the next segment.

Training in Batches

Until now, we learned the backpropagation algorithm for a single data point. This segment writes the backpropagation algorithm for a **batch** (or **mini batch**) of data points.

NEURAL NETWORKS

$$H^0 = B$$

for l in $[1, \dots, L]$:

$$H^l \leftarrow \sigma(W^l \cdot H^{l-1} + b^l)$$

$$P \leftarrow \text{colnormalize}(\exp(W^0 \cdot H^L + b^0))$$

$$DZ^{L+1} \leftarrow P - Y \quad Z^{L+1} = Z^0 \text{ (specific to (E Loss))}$$

$$\nabla_{\omega}^{L+1} = DW^{L+1} \leftarrow DZ^{L+1} (H^L)^T$$

for l in $[L, \dots, 1]$:

$$dH^l \leftarrow (W^{l+1})^T \cdot DZ^{l+1}$$

$$DZ^l \leftarrow dH^l \otimes \sigma'(Z^l)$$

$$\nabla_{\omega}^l \leftarrow DZ^l \odot_T H^{l-1}$$

$$\nabla_b^l \leftarrow DZ^l \quad Z^l = W^l h^{l-1} + b^l$$

Backpropagation UpGrad

$dZ \rightarrow$ vector

$\rightarrow DZ \rightarrow$ matrix of dZ

$Z^l \rightarrow$ matrix of z vectors one for each datapoint in the batch

width of previous layer

width of following layer

batch size

3-d Tensor

How the backpropagation algorithm can be written for batches of multiple data points. The important points are:

Loss L is computed over a batch. In a way, the batch acts as a proxy for the whole dataset. Hence, for

$$\frac{1}{m} \sum_{x \in B} L(N(x), y)$$

a batch size of m , the **average loss** is:

This is the average loss over the m data points of the batch. $N(x)$ is the network output (the vector p). Let's denote a batch of input data points by the matrix B . The backpropagation algorithm for a batch is:

<ol style="list-style-type: none"> 1. $H^0 = B$ 2. for l in $[1, 2, \dots, L]$: <ol style="list-style-type: none"> 1. $H^l = \sigma(W^l \cdot H^{l-1} + b^l)$ 3. $P = \text{normalize}(\exp(W^0 \cdot H^L + b^0))$ 4. $L = -\frac{1}{m} Y^T \cdot \log(P)$ 5. $DZ^{L+1} = P - Y$ 6. $DW^{L+1} = DZ^{L+1} \cdot (H^L)^T$ 7. for l in $[L, L-1, \dots, 1]$: <ol style="list-style-type: none"> 1. $dH^l = (W^{l+1})^T \cdot DZ^{l+1}$ 2. $DZ^l = dH^l \otimes \sigma'(Z^l)$ 3. $DW^l = DZ^l \cdot (H^{l-1})^T$ 4. $Db^l = DZ^l$ 	<p>Earlier dz^l represented a vector while DZ^l now represents the matrix consisting of the dz^l vectors of all the data points in B (each dz^l being a column of DZ^l).</p> <p>Similarly, dH^l, Y, P and H^l are all matrices of the corresponding individual vectors stacked side by side.</p> <p>DW^l is a tensor while Db^l is a matrix. This is something we don't want because, in the update step, when we write:</p> $W_{new}^l = W_{old}^l - \alpha \cdot \frac{\partial L}{\partial W^l},$ $b_{new}^l = b_{old}^l - \alpha \cdot \frac{\partial L}{\partial b^l},$ <p>we want the shapes of $\partial L / \partial W^l$ (currently a tensor DW^l) and W^l (a matrix) to be the same. Similarly, we want the shapes of $\partial L / \partial b^l$ (currently a matrix Db^l) to be the same as that of b^l (a vector). Let's see how the tensor DW^l and the matrix Db^l can be converted to a matrix and a vector respectively:</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

NEURAL NETWORKS

Backpropagation

$$\nabla_w^l = DZ^l \otimes_T H^{l-1}$$

2-d matrix tensor product

$\nabla_w^l \leftarrow \text{average on batch dimension}(\nabla_w^l)$

$\nabla_b^l \leftarrow \text{average on batch dimension}(\nabla_b^l)$

$W^l \leftarrow W^l - \alpha \cdot \nabla_w^l$

$b^l \leftarrow b^l - \alpha \cdot \nabla_b^l$

end of for loop vector

batch size

batch size

$(h^{l-1})^T$ for a single datapoint

DZ

3-D Tensor


width of the subsequent layer

width of previous layer

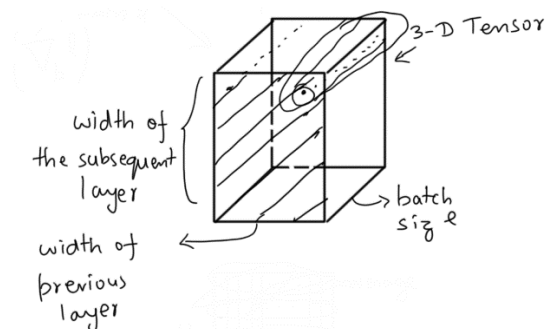
batch size

batch

db



Dot product $DW^l = DZ^l \cdot (H^{l-1})^T$ is basically a **tensor**. This can be visualised as follows:



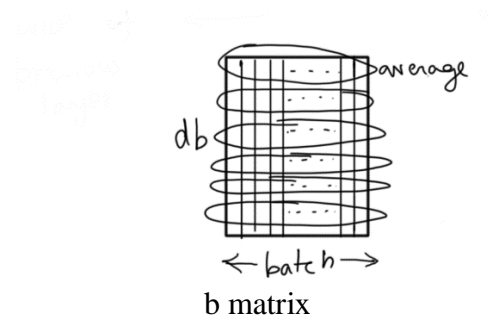
W tensor

Each 'layer' along the 'batch size' dimension of this matrix contains the dW^l matrix for each data

point. Loss function we are minimizing is the average loss of data points in B, i.e. $\frac{1}{m} \sum_{x \in B} L(p, y)$.

Hence, we take the average of all the dW^l matrices across the batch to get the final weight gradient matrix for the update step. In other words, we collapse the tensor along the batch dimension using the average.

Similarly, Db^l is a matrix as shown below - each column corresponds to the vector db^l for one data point in the batch:



This matrix consists of the gradient vectors db^l of each data point in the batch stacked side by side. Since we minimize the average loss of the batch, we take the average of these 'bias gradient vectors' to finally create a single vector $\partial L / \partial b$.

Summary

How **backpropagation** happens in neural networks and how the parameters are updated using the **gradient descent** algorithm.

Task is to **minimize the loss function** wrt a large number of parameters and that can be done efficiently using gradient descent. Idea of training using a simple example of an OR gate. Learnt to derive the expressions for the gradient of loss with respect to the variables Z, W, b, H of the various layers for a single data point.

Concept of **minibatch gradient descent** (also called **stochastic gradient descent, SGD**). It helps in reaching the global minimum faster (or at all) than using the vanilla gradient descent (done for an entire batch). SGD conducts the training in **multiple epochs** where each epoch consists of multiple mini-batches of data points.

How the final gradients with respect to the weights and biases are computed by collapsing a tensor and a matrix respectively. Here is the summarized complete pass through a neural network having L hidden layers:

```
1.  $H^0 = B$ 
2. for  $l$  in  $[1, 2, \dots, L]$  :
    1.  $H^l = \sigma(W^l \cdot H^{l-1} + b^l)$ 
    3.  $P = \text{normalize}(\exp(W^L \cdot H^L + b^L))$ 
    4.  $L = -\frac{1}{m} Y^T \cdot \log(P)$ 
    5.  $DZ^{L+1} = P - Y$ 
    6.  $DW^{L+1} = DZ^{L+1} \cdot (H^L)^T$ ;  $Db^{L+1} = DZ^{L+1}$ 
7. for  $l$  in  $[L, L-1, \dots, 1]$  :
    1.  $dH^l = (W^{l+1})^T \cdot DZ^{l+1}$ 
    2.  $DZ^l = dH^l \otimes \sigma'(Z^l)$ 
    3.  $DW^l = DZ^l \cdot (H^{l-1})^T$ 
    4.  $Db^l = DZ^l$ 
    5.  $\frac{\partial L}{\partial W^l} = \frac{1}{m} DW^l$ 
    6.  $\frac{\partial L}{\partial b^l} = \frac{1}{m} Db^l$ 
8. for  $l$  in  $[1, 2, \dots, L]$  :
    1.  $W_{new}^l = W_{old}^l - \alpha \cdot \frac{\partial L}{\partial W^l}$ 
    2.  $b_{new}^l = b_{old}^l - \alpha \cdot \frac{\partial L}{\partial b^l}$ 
```