- Neural Networks

Understanding Binary Classification

- Binary classification is a common machine learning task. It involves predicting whether a given example is part of one class or the other. The two classes can be arbitrarily assigned either a ***"0"*** or a ***"1"*** for mathematical representation, but more commonly the object/class of interest is assigned a "***1***"***(positive label)*** and the rest a "***0***"***(negative label)***

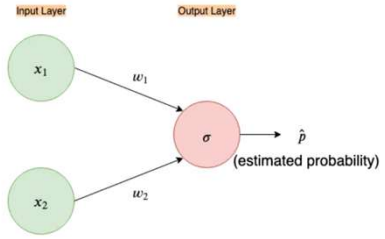


Fig 1. Simple input-output only neural network

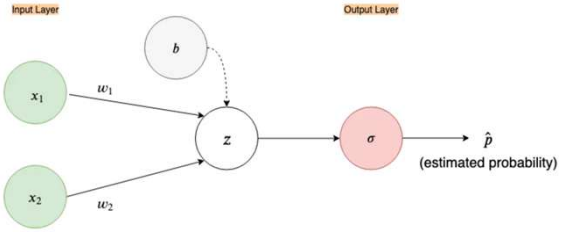


Fig 2. Expanded neural network

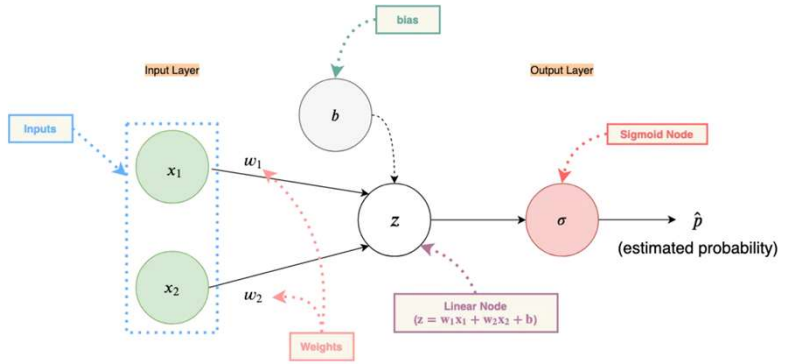
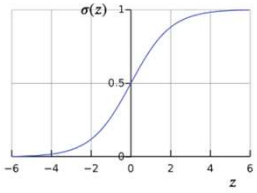


Fig 3. Annotated Expanded Neural Network

- **Inputs:** x_1 and x_2 are the input nodes for two features that represent an example we want our neural network to learn from. Since input nodes form the first layer of the network they are collectively referred to as the "***input layer***".
- **Weights:** w_1 & w_2 represent the weight values that we associate with the inputs x_1 & x_2, respectively**.** Weights control the influence each input has in the calculation of the next node. A neural network "learns" these weights to make accurate predictions. *Initially, weights are randomly assigned.*
- **Linear Node(z):** The "***z***" node creates a linear function out of all the inputs coming into it *i.e* $z = w_1x_1+w_2x_2+b$
- **Bias:** "***b***" represents the bias node. The bias node inserts an additive quantity into the linear function node(***z***). As the name suggests *the bias sways the output so that it may better align with our desired output*. The value of the bias is initialized to ***b=0*** and is also learned during the training phase.

- **Sigmoid Node:** This ***σ*** node, called the Sigmoid node, takes the input from a preceding linear node(***z***) and passes it through the following activation function, called the **Sigmoid function**(because of its S-shaped curve), also known as the **Logistic function**:



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

Fig 4. Output sigmoid/logistic node

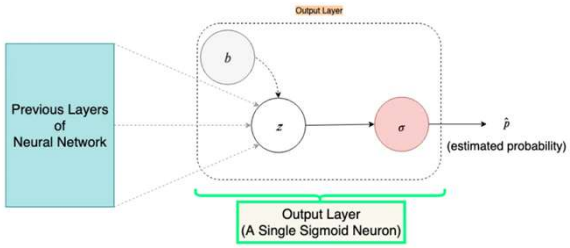


Fig 6. Structure of the output layer of any binary classification neural network

Sigmoid Curve interpreted as a probability distribution

$$P(y \mid x_1, x_2 ; w, b) = \begin{cases} \hat{p} & \text{if } y = 1 \\ 1 - \hat{p} & \text{if } y = 0 \end{cases}$$

Where "P" is the ***probability***,
"$|$" is read "***given***",
"$;$" is read "***parameterized by***"

So the above piecewise probability equation succintly describes:

- If $y = 1$, the chance/probability *of belonging to the postive class* given x_1 & x_2, parametrized by weights(w) and bias(b) is equal to $\hat{p}$
- Similarly, if $y = 0$, the chance/probability of *belonging to the negative class* given x_1 & x_2, parametrized w and b is equal to $1 - \hat{p}$

Note: Beacasue we are dealing in probablilties the combined probability of both the classes(y=1 and y=0) is equal to 1:

$$P(y = 1) + P(y = 0) = \hat{p} + (1 - \hat{p}) = 1$$

Fig 7. Piecewise probability equation

Sigmoid Curve interpreted as a probability distribution

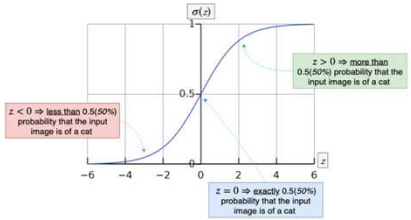
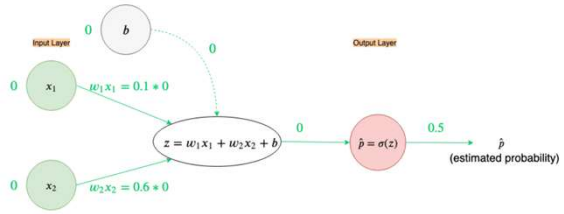


Fig 8. Sigmoid Curve interpreted as a probability distribution

- ***Greater than zero(z>0)*** then the output of the Sigmoid node is ***greater than 0.5****(σ(z)>0.5)*, which can be interpreted as "The probability that the input image is of a cat is ***greater*** than 50%".
- ***Less than zero(z<0)*** then the output of the Sigmoid node is ***less than 0.5(σ(z)<0.5)***, which can be interpreted as "The probability that the input image is of a cat is ***less*** than 50%".
- ***Equal to zero(z=0)*** then the output of the Sigmoid node ***equals 0.5(σ(z)=0.5)***, which means that "The probability that the input image is of a cat is ***exactly*** 50%".

Stochastic Gradient Descent



The neural network is going through the following computations(forward computations marked in green):

- Our input for first example $x_1 = 0, x_2 = 0$
- Recall our randomly initialized weights $w_1 = 0.1, w_2 = 0.6$.
- We'll initialize our bias to be zero, $b = 0$
- $z = w_1x_1 + w_2x_2 + b = (0.1 * 0) + (0.6 * 0) + 0 = 0$
- $\hat{p} = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{0}} = 0.5$

Fig 11. Forward propagation on the first example from AND gate table

Binary Cross-Entropy Loss Function

- *Note: in most programming languages "**log**" is the natural logarithm(log with base-**e**), denoted in mathematics as "**ln**". For consistency between code and equations consider "**log**" as natural logarithm and not as "**$\log_{10}$**"(log with base-10).*
- The Binary Cross-Entropy(BCE) Loss function is defined as follows :

$$Loss = L(y, \hat{p}) = -y \log(\hat{p}) - (1 - y) \log(1 - \hat{p})$$

where $y \Rightarrow$ label,
$\hat{p} \Rightarrow$ estimated probability of belonging to the positive class

Fig 12. The Binary Cross-Entropy Loss function

The above equation can be broken down as follows:

when $y = 1$

$$\begin{aligned} L(y=1,\hat{p}) &= -y\log(\hat{p}) - (1-y)\log(1-\hat{p}) \\ &= -1 * \log(\hat{p}) - (1-1) * \log(1-\hat{p}) \\ &= -\log(\hat{p}) - 0 * \log(1-\hat{p}) \\ &= -\log(\hat{p}) \end{aligned}$$

simlarly, when $y = 0$

$$\begin{aligned} L(y=0,\hat{p}) &= -y\log(\hat{p}) - (1-y)\log(1-\hat{p}) \\ &= -0 * \log(\hat{p}) - (1-0) * \log(1-\hat{p}) \\ &= -1 * \log(1-\hat{p}) \\ &= -\log(1-\hat{p}) \end{aligned}$$

resulting in the following ***piecewise equation:***

$$\therefore L(y,\hat{p}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1-\hat{p}) & \text{if } y = 0 \end{cases}$$

Fig 13. Binary Cross-Entropy Loss function broken down into piecewise equation

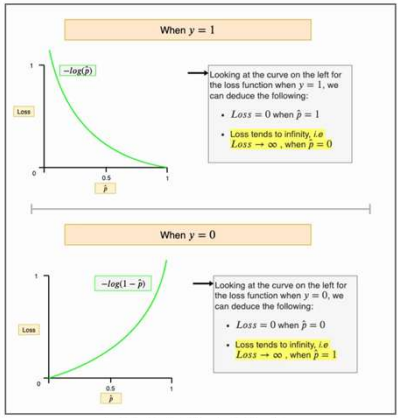


Fig 14. Visualizing the BCE Loss function for each class

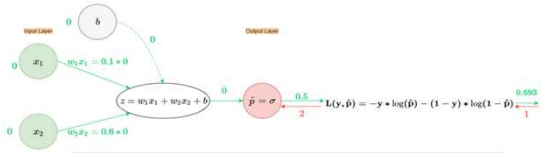
For this derivative we would need the derivative of the ***natural logarithm****(recall it is denoted by* $\log(x)$*, not* $\ln(x)$*)*:

$$\frac{\mathrm{d}}{\mathrm{dx}}\log(x) = \frac{1}{x}$$

Since y is the label of a given example and remains constant for a training example, we'll take the partial derivative(∂) with respect to $\hat{p}$:

$$\begin{aligned} \frac{\partial\, L(y,\hat{p})}{\partial \hat{p}} &= \frac{\partial}{\partial \hat{p}}(-y\log(\hat{p}) - (1-y)\log(1-\hat{p})) \\ &= \frac{\partial}{\partial \hat{p}}(-y\log(\hat{p})) - \frac{\partial}{\partial \hat{p}}((1-y)\log(1-\hat{p})) \\ &= \frac{-y}{\hat{p}} - \left(\frac{1-y}{1-\hat{p}} * -1\right) \\ &= \frac{-y}{\hat{p}} + \frac{1-y}{1-\hat{p}} \end{aligned}$$

Fig 15. The derivative of the Binary Cross-Entropy Loss function



The neural network is going through the following computations(backward computations are marked in red):

- ***The first backward computation, for the most part, is redundant but we'll define it for the sake of completeness.***
- The first backward computation is: $\frac{\partial L}{\partial L} = 1$ - *this forms the first Upstream gradient*
- The *Local gradient* at $L(y,\hat{p}) = -y * \log(\hat{p}) - (1-y) * \log(1-\hat{p})$ is (as derived above):

$$\frac{\partial\, L}{\partial \hat{p}} = \frac{-y}{\hat{p}} + \frac{1-y}{1-\hat{p}}$$

Recall, $\hat{p}$ for current example is $\hat{p} = 0.5$ and the label y is $y = 0$. So, numerical value of local gradient is:

$$\begin{aligned} \frac{\partial\, L}{\partial \hat{p}} &= \frac{-y}{\hat{p}} + \frac{1-y}{1-\hat{p}} \\ &= \frac{-0}{0.5} + \frac{1-0}{1-0.5} \\ &= \frac{1}{0.5} \\ &= 2 \end{aligned}$$

- Finally, we'll combine these and send back to the red node:

$$\frac{\partial L}{\partial \hat{p}} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial L} * \frac{\partial L}{\partial \hat{p}} = 1 * 2 = 2$$

Fig 19.a. Backpropagation on the 1st example

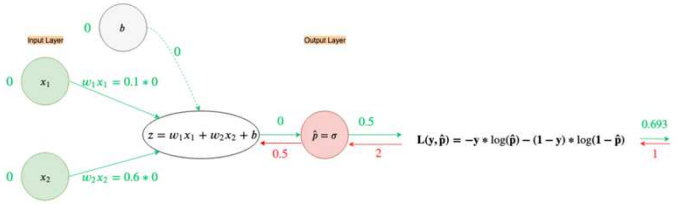
The Sigmoid Function is:

$$\hat{p} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Derivative of the Sigmoid Function is:

$$\frac{\partial \hat{p}}{\partial z} = \hat{p}(1 - \hat{p})$$

Fig 20. The derivative of the sigmoid function

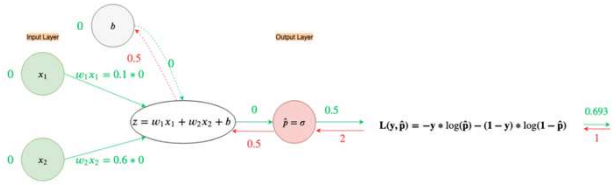


The neural network is going through the following computations(backward computations are marked in red):

- The *Upstream gradient* in this step is $\frac{\partial L}{\partial \hat{p}} = 2$
- The *Local gradient* at the red node(sigmoid node) is:
  $\frac{\partial \hat{p}}{\partial z} = \hat{p} * (1 - \hat{p}) = 0.5 * (1 - 0.5) = 0.5 * (0.5) = 0.25$
- Like previously, we will combine these and send them backwards this time to the white node(**z**):

$$\frac{\partial L}{\partial z} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial \hat{p}} * \frac{\partial \hat{p}}{\partial z} = 2 * (0.25) = 0.5$$

Fig 19.b. Backpropagation on the 1st example



The neural network is going through the following computations(backward computations are marked in red):

- FInally, we have now propogated the upstream gradient back enough to calcalute the gradient of Loss with respect to w_1 ,w_2 and our bias b
- The *Upstream gradient* in this step is $\frac{\partial L}{\partial z} = 0.5$
- The *three Local gratients* are:
  1. $\frac{\partial z}{\partial w_1} = \frac{\partial}{\partial w_1}(w_1x_1 + w_2x_2 + b) = x_1 = 0$
  2. $\frac{\partial z}{\partial w_2} = \frac{\partial}{\partial w_2}(w_1x_1 + w_2x_2 + b) = x_2 = 0$
  3. $\frac{\partial z}{\partial b} = \frac{\partial}{\partial b}(w_1x_1 + w_2x_2 + b) = 1$
- We will again combine these, but this time not send them back to our input nodes as we don't want to actually change our data, instead just figure out how much to change the weights w_1, w_2 and bias b:
  1. $\frac{\partial L}{\partial w_1} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_1} = 0.5 * 0 = 0$
  2. $\frac{\partial L}{\partial w_2} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial w_2} = 0.5 * 0 = 0$
  3. $\frac{\partial L}{\partial b} = UpstreamGradient * LocalGradient = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial b} = 0.5 * 1 = 0.5$

Fig 19.c. Backpropagation on the 1st example

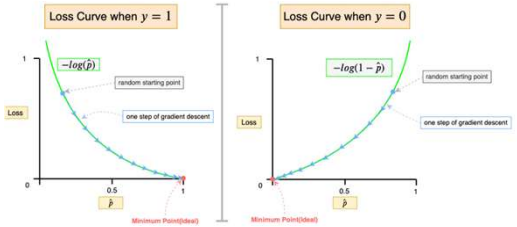


Fig 21 Visualizing Gradient Descent on Binary Cross-Entropy Loss function

General Equation for Gradient Descent

$$w = w - \alpha \frac{\partial L}{\partial w}$$

Learning Rate

Fig 22. The general equation for **gradient descent**

To calculate new weights and bias we move in the negative direction of the gradient

Recall our current weights are $w_1 = 0.1$ and $w_2 = 0.6$ and current bias is $b = 0$

Our gradients for w_1, w_2 and b are $\frac{\partial L}{\partial w_1} = 0$, $\frac{\partial L}{\partial w_2} = 0$ and $\frac{\partial L}{\partial b} = 0.5$, respectively.

Our learning rate is $\alpha = 1$

The new weights are:

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1} = 0.1 - (1 * 0) = \mathbf{0.1}$$

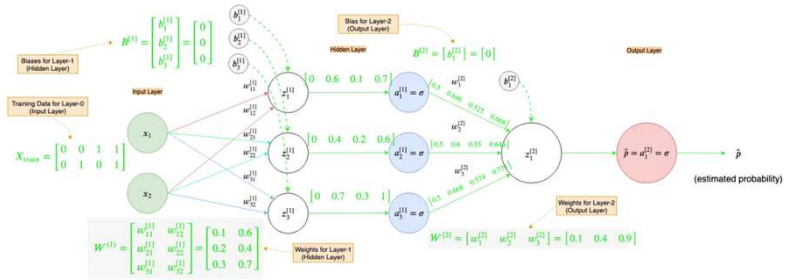
$$w_2 = w_2 - \alpha \frac{\partial L}{\partial w_2} = 0.6 - (1 * 0) = \mathbf{0.6}$$

The new bias is:

$$b = b - \alpha \frac{\partial L}{\partial b} = 0 - (1 * 0.5) = 0 - 0.5 = \mathbf{-0.5}$$

Fig 23. Calculating new weights and bias

Shallow Neural Network



The neural network is going through the following computations(forward computations marked in green):

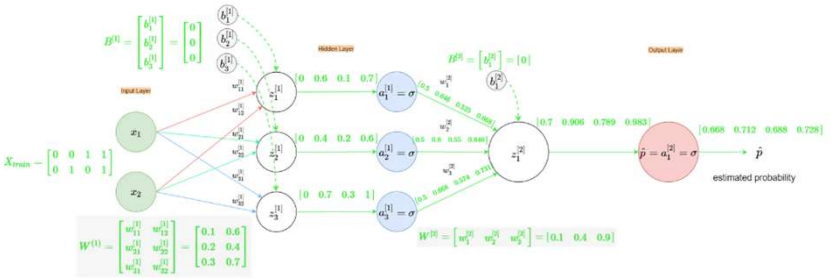
- The first layer(input layer) is passed our data $X_{train} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$ and randomly chosen weights, $W^{[1]} = \begin{bmatrix} 0.1 & 0.6 \\ 0.2 & 0.4 \\ 0.4 & 0.7 \end{bmatrix}$ and zero initialized bias, $B^{[1]} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$
- In our neural network, the ***input layer* is "layer-0", *the hidden layer* is "layer 1" and *the output layer* is "layer-2".** So the superscript "**[1]**" denotes that the data element belongs to ***layer 1***, and connects elements from layer-0 to layer-1.
- All the $z^{[1]}$ nodes are calculated in one go in a vectorized calculation, as follows:

$$Z^{[1]} = W^{[1]} \cdot X_{train} + B^{[1]}$$
$$= \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \cdot \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix}$$
$$= \begin{bmatrix} 0.1 & 0.6 \\ 0.2 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0.6 & 0.1 & 0.7 \\ 0 & 0.4 & 0.2 & 0.6 \\ 0 & 0.7 & 0.3 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0.6 & 0.1 & 0.7 \\ 0 & 0.4 & 0.2 & 0.6 \\ 0 & 0.7 & 0.3 & 1 \end{bmatrix} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & z_1^{[1](3)} & z_1^{[1](4)} \\ z_2^{[1](1)} & z_2^{[1](2)} & z_2^{[1](3)} & z_2^{[1](4)} \\ z_3^{[1](1)} & z_3^{[1](2)} & z_3^{[1](3)} & z_3^{[1](4)} \end{bmatrix}$$

example #1 example #2 example #3 example #4

- Similar to all the z nodes in layer-1 all the activations($a^{[*]}$) of layer-1 are also calculated in one go, as follows:

$$A^{[1]} = \sigma(Z^{[1]})$$
$$= \begin{bmatrix} \sigma(z_1^{[1](1)}) & \sigma(z_1^{[1](2)}) & \sigma(z_1^{[1](3)}) & \sigma(z_1^{[1](4)}) \\ \sigma(z_2^{[1](1)}) & \sigma(z_2^{[1](2)}) & \sigma(z_2^{[1](3)}) & \sigma(z_2^{[1](4)}) \\ \sigma(z_3^{[1](1)}) & \sigma(z_3^{[1](2)}) & \sigma(z_3^{[1](3)}) & \sigma(z_3^{[1](4)}) \end{bmatrix}$$
$$= \begin{bmatrix} \sigma(0) & \sigma(0.6) & \sigma(0.1) & \sigma(0.7) \\ \sigma(0) & \sigma(0.4) & \sigma(0.2) & \sigma(0.6) \\ \sigma(0) & \sigma(0.7) & \sigma(0.3) & \sigma(1) \end{bmatrix}$$
$$= \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix} = \begin{bmatrix} a_1^{[1](1)} & a_1^{[1](2)} & a_1^{[1](3)} & a_1^{[1](4)} \\ a_2^{[1](1)} & a_2^{[1](2)} & a_2^{[1](3)} & a_2^{[1](4)} \\ a_3^{[1](1)} & a_3^{[1](2)} & a_3^{[1](3)} & a_3^{[1](4)} \end{bmatrix}$$



The neural network is going through the following computations(forward computations marked in green):

- Now we'll propagate the values from the ***Hidden Layer(layer-1)*** to the ***Output Layer(layer-2).***
- **The values following into $z_1^{[2]}$ are the activations from the previous layer(the Hidden Layer, in this case).**

$$A_{prev} = A^{[1]} = \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix}$$

- As before we'll perform all the calculations for all the examples in one go.

$$\begin{aligned} Z^{[2]} &= W^{[2]} \cdot A^{[1]} + B^{[2]} \\ &= \begin{bmatrix} w_1^{[2]} & w_2^{[2]} & w_3^{[2]} \end{bmatrix} \cdot \begin{bmatrix} a_1^{[1](1)} & a_1^{[1](2)} & a_1^{[1](3)} & a_1^{[1](4)} \\ a_2^{[1](1)} & a_2^{[1](2)} & a_2^{[1](3)} & a_2^{[1](4)} \\ a_3^{[1](1)} & a_3^{[1](2)} & a_3^{[1](3)} & a_3^{[1](4)} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \end{bmatrix} \\ &= \begin{bmatrix} 0.1 & 0.4 & 0.9 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 & .906 & 0.789 & 0.983 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \\ &= \begin{bmatrix} 0.7 & .906 & 0.789 & 0.983 \end{bmatrix} = \begin{bmatrix} z_1^{[2](1)} & z_1^{[2](2)} & z_1^{[2](3)} & z_1^{[2](4)} \end{bmatrix} \end{aligned}$$

- Now we can calculate the activations of the Output Layer(layer-2) which form the estimated probabilities:

$$\begin{aligned} \hat{P} = A^{[2]} &= \sigma(Z^{[2]}) \\ &= \begin{bmatrix} \sigma(z_1^{[2](1)}) & \sigma(z_1^{[2](2)}) & \sigma(z_1^{[2](3)}) & \sigma(z_1^{[2](4)}) \end{bmatrix} \\ &= \begin{bmatrix} \sigma(0.7) & \sigma(0.906) & \sigma(0.789) & \sigma(0.983) \end{bmatrix} \\ &= \begin{bmatrix} 0.668 & 0.712 & 0.688 & 0.728 \end{bmatrix} = \begin{bmatrix} a_1^{[2](1)} & a_1^{[2](2)} & a_1^{[2](3)} & a_1^{[2](4)} \end{bmatrix} = \begin{bmatrix} \hat{p}^{(1)} & \hat{p}^{(2)} & \hat{p}^{(3)} & \hat{p}^{(4)} \end{bmatrix} \end{aligned}$$

$$\begin{aligned} Cost(Y,Z) &= \frac{1}{m}\sum_{i=1}^{m} max(z^{(i)},0) - z^{(i)}y^{(i)} + \log(1+e^{-|z^{(i)}|}) \\ &= \frac{1}{m}\sum max(Z,0) - ZY + \log(1+e^{-|Z|}) \\ &= \frac{1}{4}\sum max(\begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix},0) - \begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix} \odot \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix} + \log(1+e^{-|\begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix}|}) \\ &= \frac{1}{4}\sum \begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix} - \begin{bmatrix} (0.7*0) & (0.906*1) & (0.789*1) & (0.983*0) \end{bmatrix} + \log(1+e^{-\begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix}}) \\ &= \frac{1}{4}\sum \begin{bmatrix} 0.7 & 0.906 & 0.789 & 0.983 \end{bmatrix} - \begin{bmatrix} 0 & 0.906 & 0.789 & 0 \end{bmatrix} + \log(1+\begin{bmatrix} 0.497 & 0.404 & 0.454 & 0.374 \end{bmatrix}) \\ &= \frac{1}{4}\sum \begin{bmatrix} (0.7-0) & (0.906-0.906) & (0.789-0.789) & (0.983-0) \end{bmatrix} + \log(\begin{bmatrix} (1+0.497) & (1+0.404) & (1+0.454) & (1+0.374) \end{bmatrix}) \\ &= \frac{1}{4}\sum \begin{bmatrix} 0.7 & 0 & 0 & 0.983 \end{bmatrix} + \log(\begin{bmatrix} 1.497 & 1.404 & 1.454 & 1.374 \end{bmatrix}) \\ &= \frac{1}{4}\sum \begin{bmatrix} 0.7 & 0 & 0 & 0.983 \end{bmatrix} + \begin{bmatrix} 0.403 & 0.339 & 0.374 & 0.318 \end{bmatrix} \\ &= \frac{1}{4}\sum \begin{bmatrix} (0.7+0.403) & (0+0.339) & (0+0.374) & (0.983+0.318) \end{bmatrix} \\ &= \frac{1}{4}\sum \begin{bmatrix} 1.103 & 0.339 & 0.374 & 1.301 \end{bmatrix} \\ &= \frac{1}{4}(1.103+0.339+0.374+1.301) = \frac{1}{4}(3.117) \\ &= \mathbf{0.779} \end{aligned}$$

Fig 69. Calculation of Stable Binary Cross-Entropy Cost

Stable Cross Entropy

$$Loss(y,\hat{p}) = -y\log(\hat{p}) - (1-y)\log(1-\hat{p})$$

$$\begin{aligned} Loss(y,\hat{p}) &= -y\log(\hat{p}) - (1-y)\log(1-\hat{p}) \\ &= -y\log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{-z}}\right) \\ &= -y\log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log\left(\frac{e^{-z}}{1+e^{-z}}\right) \end{aligned}$$

$$\begin{aligned} &= -y\left[\log(1) - \log(1+e^{-z})\right] - (1-y)\left[\log(e^{-z}) - \log(1+e^{-z})\right] \\ &= -y\left[0 - \log(1+e^{-z})\right] - (1-y)\left[-z\log(e) - \log(1+e^{-z})\right] \\ &= \mathbf{y\log(1+e^{-z}) - (1-y)\left[-z - \log(1+e^{-z})\right]} \end{aligned}$$

$$\begin{aligned} &= y\log(1+e^{-z}) - (1-y)\left[-z - \log(1+e^{-z})\right] \\ &= y\log(1+e^{-z}) + z(1-y) + (1-y)\log(1+e^{-z}) \\ &= \cancel{y\log(1+e^{-z})} + z - zy + \log(1+e^{-z}) - \cancel{y\log(1+e^{-z})} \\ &= \mathbf{z - zy + \log(1+e^{-z})} \end{aligned}$$

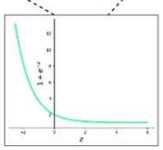


Fig 53. Making the Binary Cross-Entropy Loss Function stable

*In piecewise form our **stable Binary Cross-Entropy Loss function** look as follows:*

$$Loss(y,z) = \begin{cases} z - zy + \log(1+e^{-z}) & \text{if } z \geqslant 0 \\ -zy + \log(e^{z}+1) & \text{if } z < 0 \end{cases}$$

We can elegantly combine this into one equation as follows:

$$Loss(y,z) = max(z,0) - zy + \log(1+e^{-|z|})$$

Where $|z|$ means absolute of z also called mod in mathematics.

*Similarly, the Binary Cross-Entropy **Cost** function becomes:*

$$Cost(Y,Z) = \frac{1}{m}\sum max(Z,0) - ZY + \log(1+e^{-|Z|})$$

***Note**: max here is element-wise maximum compared to zero(in NumPy this is denoted as $np.maximum$)*

Fig 58. Final stable and simplified Binary Cross -Entropy Function

Backward Computations

The neural network is going through the following computations(backward computations are in red):

- As before the first **Upstream Gradient** in redundant. $\frac{\partial Cost}{\partial Cost} = 1$
- We'll employ the optimization technique we've learned so that we don't have to calculate the local gradient of the Cost function or any of the gradients related to the last Sigmoid node. Instead we'll just bypass the last Sigmoid node and pass the following gradient to the last Linear Node($Z^{[2]}$) as the **Upstream Gradient** :

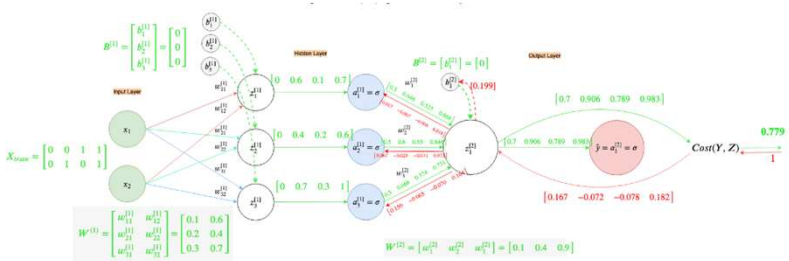
$$\begin{aligned}
\frac{\partial Cost}{\partial Z} &= \frac{1}{m}(\hat{P} - Y) \\
&= \frac{1}{m}(\sigma(Z) - Y) \\
&= \frac{1}{4}(\sigma(\begin{bmatrix}0.7 & 0.906 & 0.789 & 0.983\end{bmatrix}) - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}\sigma(0.7) & \sigma(0.906) & \sigma(0.789) & \sigma(0.983)\end{bmatrix} - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}0.668 & 0.712 & 0.688 & 0.728\end{bmatrix} - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}(0.668-0) & (0.712-1) & (0.688-1) & (0.728-0)\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}0.668 & -0.288 & -0.312 & 0.728\end{bmatrix}) \\
&= \begin{bmatrix}0.167 & -0.072 & -0.078 & 0.182\end{bmatrix}
\end{aligned}$$

$$\frac{\partial Loss(y,\hat{p})}{\partial z} = \frac{y}{\hat{p}} + \frac{1-y}{1-\hat{p}}$$

$$\frac{\partial \hat{p}}{\partial z} = \hat{p}(1-\hat{p})$$

Fig 61. The derivative of the Sigmoid function

$$\begin{aligned}
\frac{\partial Loss}{\partial z} &= UpstreamGradient * LocalGradient \\
&= \frac{\partial Loss}{\partial \hat{p}} * \frac{\partial \hat{p}}{\partial z} \\
&= \left(\frac{-y}{\hat{p}} + \frac{1-y}{1-\hat{p}}\right) * (\hat{p}(1-\hat{p})) \\
&= \left(\frac{-y}{\hat{p}} * \hat{p}(1-\hat{p})\right) + \left(\frac{1-y}{1-\hat{p}} * \hat{p}(1-\hat{p})\right) \\
&= -y(1-\hat{p}) + (1-y)\hat{p} \\
&= -y + y\hat{p} + \hat{p} - y\hat{p} \\
&= -y + \hat{p} \\
&= \hat{\mathbf{p}} - \mathbf{y}
\end{aligned}$$



The neural network is going through the following computations(backward computations are in red):

- As before the first **Upstream Gradient** in redundant. $\frac{\partial Cost}{\partial Cost} = 1$
- We'll employ the optimization technique we've learned so that we don't have to calculate the local gradient of the Cost function or any of the gradients related to the last Sigmoid node. Instead we'll just bypass the last Sigmoid node and pass the following gradient to the last Linear Node($Z^{[2]}$) as the **Upstream Gradient** :

$$\begin{aligned}
\frac{\partial Cost}{\partial Z} &= \frac{1}{m}(\hat{P} - Y) \\
&= \frac{1}{m}(\sigma(Z) - Y) \\
&= \frac{1}{4}(\sigma(\begin{bmatrix}0.7 & 0.906 & 0.789 & 0.983\end{bmatrix}) - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}\sigma(0.7) & \sigma(0.906) & \sigma(0.789) & \sigma(0.983)\end{bmatrix} - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}0.668 & 0.712 & 0.688 & 0.728\end{bmatrix} - \begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}(0.668-0) & (0.712-1) & (0.688-1) & (0.728-0)\end{bmatrix}) \\
&= \frac{1}{4}(\begin{bmatrix}0.668 & -0.288 & -0.312 & 0.728\end{bmatrix}) \\
&= \begin{bmatrix}0.167 & -0.072 & -0.078 & 0.182\end{bmatrix}
\end{aligned}$$

- The **Upstream Gradient** in this step is:

$$\frac{\partial Cost}{\partial Z^{[2]}} = \begin{bmatrix} 0.167 & -0.072 & -0.078 & 0.182 \end{bmatrix}$$

- The ***three* Local Gradients** associated with the $Z^{[2]}$ node are :

1. $\frac{\partial Z^{[2]}}{\partial W^{[2]}} = \left(A^{[1]}\right)^T = \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.646 & 0.6 & 0.668 \\ 0.525 & 0.55 & 0.574 \\ 0.668 & 0.646 & 0.731 \end{bmatrix}$
2. $\frac{\partial Z^{[2]}}{\partial A^{[1]}} = \left(W^{[2]}\right)^T = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.9 \end{bmatrix}$
3. $\frac{\partial Z^{[2]}}{\partial B^{[2]}} = [1]$

- Finally, we'll combine all the upstream gradients with the local ones and send them back to their respective nodes, or store them for the gradient descent update after finishing backpropagation:

1. $W^{[2]}$ is a (1×3) row vector so the derivative of Cost w.r.t $W^{[2]}$ should also be a (1×3) row vector:

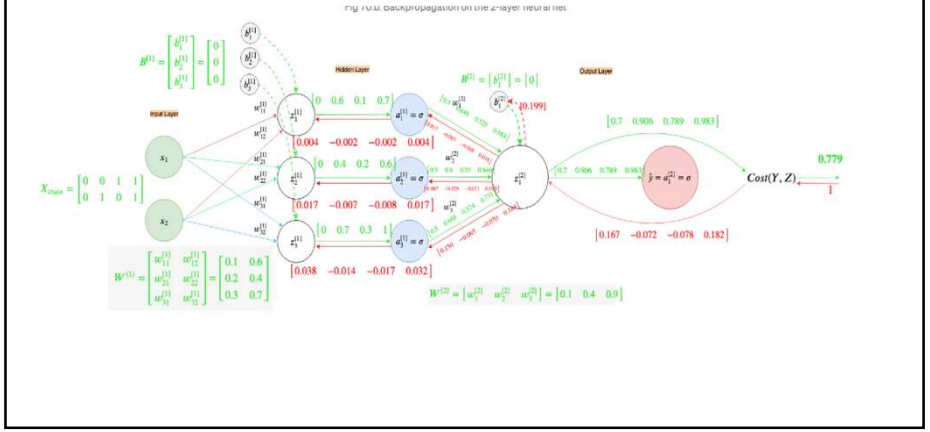
$$\begin{aligned} \frac{\partial Cost}{\partial W^{[2]}} &= UpstreamGradient \cdot LocalGradient \\ &= \frac{\partial Cost}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}} \\ &= \begin{bmatrix} \frac{\partial Cost}{\partial z_1^{[2](1)}} & \frac{\partial Cost}{\partial z_1^{[2](2)}} & \frac{\partial Cost}{\partial z_1^{[2](3)}} & \frac{\partial Cost}{\partial z_1^{[2](4)}} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial z^{[2](1)}}{\partial w_1^{[2]}} & \frac{\partial z^{[2](1)}}{\partial w_2^{[2]}} & \frac{\partial z^{[2](1)}}{\partial w_3^{[2]}} \\ \frac{\partial z^{[2](2)}}{\partial w_1^{[2]}} & \frac{\partial z^{[2](2)}}{\partial w_2^{[2]}} & \frac{\partial z^{[2](2)}}{\partial w_3^{[2]}} \\ \frac{\partial z^{[2](3)}}{\partial w_1^{[2]}} & \frac{\partial z^{[2](3)}}{\partial w_2^{[2]}} & \frac{\partial z^{[2](3)}}{\partial w_3^{[2]}} \\ \frac{\partial z^{[2](4)}}{\partial w_1^{[2]}} & \frac{\partial z^{[2](4)}}{\partial w_2^{[2]}} & \frac{\partial z^{[2](4)}}{\partial w_3^{[2]}} \end{bmatrix} \\ &= \begin{bmatrix} 0.167 & -0.072 & -0.078 & 0.182 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.5 & 0.5 \\ 0.646 & 0.6 & 0.668 \\ 0.525 & 0.55 & 0.574 \\ 0.668 & 0.646 & 0.731 \end{bmatrix} \\ &= \begin{bmatrix} 0.118 & 0.115 & 0.124 \end{bmatrix} = \begin{bmatrix} \frac{\partial Cost}{\partial w_1^{[2]}} & \frac{\partial Cost}{\partial w_2^{[2]}} & \frac{\partial Cost}{\partial w_3^{[2]}} \end{bmatrix} \end{aligned}$$

2. $B^{[2]}$ is a (1×1) matrix(*actually, just a scalar*) so the derivative of Cost w.r.t $B^{[2]}$ should also have the shape (1×1):

$$\begin{aligned} \frac{\partial Cost}{\partial B^{[2]}} &= \sum UpstreamGradient * LocalGradient \\ &= \sum \frac{\partial Cost}{\partial Z^{[2]}} * \frac{\partial Z^{[2]}}{\partial B^{[2]}} \\ &= \sum \begin{bmatrix} \frac{\partial Cost}{\partial z_1^{[2](1)}} & \frac{\partial Cost}{\partial z_1^{[2](2)}} & \frac{\partial Cost}{\partial z_1^{[2](3)}} & \frac{\partial Cost}{\partial z_1^{[2](4)}} \end{bmatrix} * \begin{bmatrix} \frac{\partial z^{[2](1)}}{\partial b_1^{[2]}} \end{bmatrix} \\ &= \sum \begin{bmatrix} 0.167 & -0.072 & -0.078 & 0.182 \end{bmatrix} * \begin{bmatrix} 1 \end{bmatrix} \\ &= [0.167 - 0.072 - 0.078 + 0.182] \\ &= [0.199] = \begin{bmatrix} \frac{\partial Cost}{\partial b_1^{[2]}} \end{bmatrix} \end{aligned}$$

3. $A^{[1]}$ has the shape (3×4) so the derivative of Cost w.r.t $A^{[1]}$ should also have the shape (3×4), this is the gradient we'll send backwards:

$$\begin{aligned} \frac{\partial Cost}{\partial A^{[1]}} &= LocalGradient \cdot UpstreamGradient \\ &= \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial Cost}{\partial Z^{[2]}} \\ &= \begin{bmatrix} \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \\ \frac{\partial z_1^{[2]}}{\partial a_2^{[1]}} \\ \frac{\partial z_1^{[2]}}{\partial a_3^{[1]}} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial Cost}{\partial z_1^{[2](1)}} & \frac{\partial Cost}{\partial z_1^{[2](2)}} & \frac{\partial Cost}{\partial z_1^{[2](3)}} & \frac{\partial Cost}{\partial z_1^{[2](4)}} \end{bmatrix} \\ &= \begin{bmatrix} 0.1 \\ 0.4 \\ 0.9 \end{bmatrix} \cdot \begin{bmatrix} 0.167 & -0.072 & -0.078 & 0.182 \end{bmatrix} \\ &= \begin{bmatrix} 0.017 & -0.007 & -0.008 & 0.018 \\ 0.067 & -0.029 & -0.031 & 0.073 \\ 0.150 & -0.065 & -0.070 & 0.164 \end{bmatrix} = \begin{bmatrix} \frac{\partial Cost}{\partial a_1^{[1](1)}} & \frac{\partial Cost}{\partial a_1^{[1](2)}} & \frac{\partial Cost}{\partial a_1^{[1](3)}} & \frac{\partial Cost}{\partial a_1^{[1](4)}} \\ \frac{\partial Cost}{\partial a_2^{[1](1)}} & \frac{\partial Cost}{\partial a_2^{[1](2)}} & \frac{\partial Cost}{\partial a_2^{[1](3)}} & \frac{\partial Cost}{\partial a_2^{[1](4)}} \\ \frac{\partial Cost}{\partial a_3^{[1](1)}} & \frac{\partial Cost}{\partial a_3^{[1](2)}} & \frac{\partial Cost}{\partial a_3^{[1](3)}} & \frac{\partial Cost}{\partial a_3^{[1](4)}} \end{bmatrix} \end{aligned}$$



The neural network is going through the following computations(backward computations are in red):

- The **Upstream Gradient** in this step is:

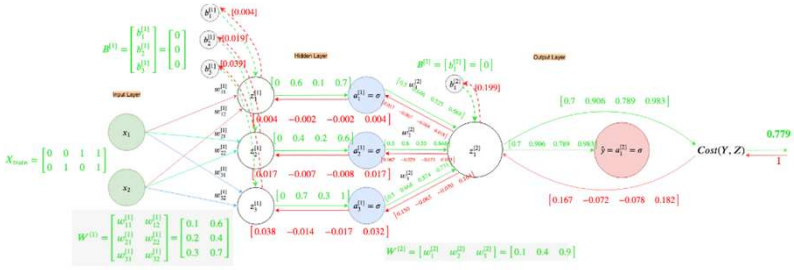
$$\frac{\partial Cost}{\partial A^{[1]}} = \begin{bmatrix} 0.017 & -0.007 & -0.008 & 0.018 \\ 0.067 & -0.029 & -0.031 & 0.073 \\ 0.150 & -0.065 & -0.070 & 0.164 \end{bmatrix}$$

- The **Local Gradient** is the derivative of the Sigmoid($a^{[1]}$ nodes):

$$\begin{aligned} \frac{\partial A^{[1]}}{\partial Z^{[1]}} &= A^{[1]}(1 - A^{[1]}) \\ &= \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix} \left(1 - \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0.5 & 0.646 & 0.525 & 0.668 \\ 0.5 & 0.6 & 0.55 & 0.646 \\ 0.5 & 0.668 & 0.574 & 0.731 \end{bmatrix} * \begin{bmatrix} 0.5 & 0.354 & 0.475 & 0.332 \\ 0.5 & 0.4 & 0.45 & 0.354 \\ 0.5 & 0.332 & 0.426 & 0.296 \end{bmatrix} \\ &= \begin{bmatrix} 0.25 & 0.229 & 0.249 & 0.222 \\ 0.25 & 0.24 & 0.248 & 0.229 \\ 0.25 & 0.222 & 0.244 & 0.197 \end{bmatrix} \end{aligned}$$

- Let's combine these two and send them back to the z nodes of the Hidden Layer:

$$\begin{aligned} \frac{\partial Cost}{\partial Z^{[1]}} &= UpstreamGradient * LocalGradient \\ &= \frac{\partial Cost}{\partial A^{[1]}} * \frac{\partial A^{[1]}}{\partial Z^{[1]}} \\ &= \begin{bmatrix} 0.017 & -0.007 & -0.008 & 0.018 \\ 0.067 & -0.029 & -0.031 & 0.073 \\ 0.150 & -0.065 & -0.070 & 0.164 \end{bmatrix} * \begin{bmatrix} 0.25 & 0.229 & 0.249 & 0.222 \\ 0.25 & 0.24 & 0.248 & 0.229 \\ 0.25 & 0.222 & 0.244 & 0.197 \end{bmatrix} \\ &= \begin{bmatrix} 0.004 & -0.002 & -0.002 & 0.004 \\ 0.017 & -0.007 & -0.008 & 0.017 \\ 0.038 & -0.014 & -0.017 & 0.032 \end{bmatrix} = \begin{bmatrix} \frac{\partial Cost}{\partial z_1^{[1](1)}} & \frac{\partial Cost}{\partial z_1^{[1](2)}} & \frac{\partial Cost}{\partial z_1^{[1](3)}} & \frac{\partial Cost}{\partial z_1^{[1](4)}} \\ \frac{\partial Cost}{\partial z_2^{[1](1)}} & \frac{\partial Cost}{\partial z_2^{[1](2)}} & \frac{\partial Cost}{\partial z_2^{[1](3)}} & \frac{\partial Cost}{\partial z_2^{[1](4)}} \\ \frac{\partial Cost}{\partial z_3^{[1](1)}} & \frac{\partial Cost}{\partial z_3^{[1](2)}} & \frac{\partial Cost}{\partial z_3^{[1](3)}} & \frac{\partial Cost}{\partial z_3^{[1](4)}} \end{bmatrix} \end{aligned}$$



The neural network is going through the following computations(backward computations are in red):

- The **Upstream Gradient** in this last step is:

$$\frac{\partial Cost}{\partial Z^{[1]}} = \begin{bmatrix} 0.004 & -0.002 & -0.002 & 0.004 \\ 0.017 & -0.007 & -0.008 & 0.017 \\ 0.038 & -0.014 & -0.017 & 0.032 \end{bmatrix}$$

- We have propagated the gradients back enough to calculate the weights and biases associated with the hidden layer the **two Local Gradients** are:

1. $\frac{\partial Z^{[1]}}{\partial W^{[1]}} = X_{train}^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$
2. $\frac{\partial Z^{[1]}}{\partial B^{[1]}} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Again, note that we will not claulate the local gradient w.r.t X_{train} as we do not intend to change the training data through our gradient descet update.

- Finally, let's combine the upstream and local gradients:

1. $W^{[1]}$ is a (3×2) matrix so the derivative of Cost w.r.t $W^{[1]}$ should also be of the same shape:

$$\begin{aligned} \frac{\partial Cost}{\partial W^{[1]}} &= UpstreamGradient \cdot LocalGradient \\ &= \frac{\partial Cost}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}} \\ &= \begin{bmatrix} 0.004 & -0.002 & -0.002 & 0.004 \\ 0.017 & -0.007 & -0.008 & 0.017 \\ 0.038 & -0.014 & -0.017 & 0.032 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.002 & 0.002 \\ 0.009 & 0.010 \\ 0.015 & 0.018 \end{bmatrix} = \begin{bmatrix} \frac{\partial Cost}{\partial w_{11}^{[1]}} & \frac{\partial Cost}{\partial w_{12}^{[1]}} \\ \frac{\partial Cost}{\partial w_{21}^{[1]}} & \frac{\partial Cost}{\partial w_{22}^{[1]}} \\ \frac{\partial Cost}{\partial w_{31}^{[1]}} & \frac{\partial Cost}{\partial w_{32}^{[1]}} \end{bmatrix} \end{aligned}$$

2. The bias vector, $B^{[1]}$, has the shape (3×1) so its derivative should also have the same shape:

$$\begin{aligned} \frac{\partial Cost}{\partial B^{[1]}} &= \sum(UpstreamGradient \cdot LocalGradient, axis = 1) \\ &= \sum\left(\frac{\partial Cost}{\partial Z^{[1]}} \odot \frac{\partial Z^{[1]}}{\partial B^{[1]}}, axis = 1\right) \\ &= \sum\left(\begin{bmatrix} \frac{\partial Cost}{\partial z_1^{[1](1)}} & \frac{\partial Cost}{\partial z_1^{[1](2)}} & \frac{\partial Cost}{\partial z_1^{[1](3)}} & \frac{\partial Cost}{\partial z_1^{[1](4)}} \\ \frac{\partial Cost}{\partial z_2^{[1](1)}} & \frac{\partial Cost}{\partial z_2^{[1](2)}} & \frac{\partial Cost}{\partial z_2^{[1](3)}} & \frac{\partial Cost}{\partial z_2^{[1](4)}} \\ \frac{\partial Cost}{\partial z_3^{[1](1)}} & \frac{\partial Cost}{\partial z_3^{[1](2)}} & \frac{\partial Cost}{\partial z_3^{[1](3)}} & \frac{\partial Cost}{\partial z_3^{[1](4)}} \end{bmatrix} \odot \begin{bmatrix} \frac{\partial Z^{[1]}}{\partial b_1^{[1]}} \\ \frac{\partial Z^{[1]}}{\partial b_2^{[1]}} \\ \frac{\partial Z^{[1]}}{\partial b_3^{[1]}} \end{bmatrix}, axis = 1\right) \\ &= \sum\left(\begin{bmatrix} 0.004 & -0.002 & -0.002 & 0.004 \\ 0.017 & -0.007 & -0.008 & 0.017 \\ 0.038 & -0.014 & -0.017 & 0.032 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, axis = 1\right) \\ &= \sum\left(\begin{bmatrix} 0.004 & -0.002 & -0.002 & 0.004 \\ 0.017 & -0.007 & -0.008 & 0.017 \\ 0.038 & -0.014 & -0.017 & 0.032 \end{bmatrix}, axis = 1\right) \\ &= \begin{bmatrix} 0.004 - 0.002 - 0.002 + 0.004 \\ 0.017 - 0.007 - 0.008 + 0.017 \\ 0.038 - 0.014 - 0.017 + 0.032 \end{bmatrix} \\ &= \begin{bmatrix} 0.004 \\ 0.019 \\ 0.039 \end{bmatrix} = \begin{bmatrix} \frac{\partial Cost}{\partial b_1^{[1]}} \\ \frac{\partial Cost}{\partial b_2^{[1]}} \\ \frac{\partial Cost}{\partial b_3^{[1]}} \end{bmatrix} \end{aligned}$$