Fuzzy Logic & Neural Networks (CS-514)

Dr. Sudeep Sharma

IIIT Surat

sudeep.sharma@iiitsurat.ac.in

Activation Function Derivatives

The Linear Activation Function Derivative

$$f(z) = y = z$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial z}{\partial z} = 1$$

Activation Function Derivatives

The ReLU Activation Function Derivative

$$f(z) = \mathrm{Re}LU(z) = \max(0, z)$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial \mathrm{Re}LU(z)}{\partial z} = \begin{cases} 1 & z > 0 \\ 0 & z \leqslant 0 \end{cases}$$

Activation Function Derivatives

The Leaky ReLU Activation Function Derivative

$$f(z) \;=\; \text{Leaky ReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases} ; 0 < \alpha < 0.1$$

$$\frac{\partial}{\partial z}\text{Leaky ReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z \leq 0 \end{cases}$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (logsig)

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

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$$f(z) = \frac{g(z)}{h(z)}$$

$$\frac{\partial f(z)}{\partial z} = \frac{g'(z)h(z) - g(z)h'(z)}{[h(z)]^2}$$

$$g(z) = 1,\ h(z) = (1+e^{-z})$$

$$\frac{\partial g(z)}{\partial z} = g'(z) = 0,\ \frac{\partial h(z)}{\partial z} = h'(z) = -e^{-z}$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (logsig)

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

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$$\frac{\partial f(z)}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^{2}}$$

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

$$\frac{\partial f(z)}{\partial z} = f(z)\left(1-f(z)\right)$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (tansig)

$$f(z) = y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)$$

$$f(z) = \frac{g(z)}{h(z)}$$

$$\frac{\partial f(z)}{\partial z} = \frac{g'(z)h(z) - g(z)h'(z)}{[h(z)]^{2}}$$

$$g(z) = (e^{z} - e^{-z}),\ h(z) = (e^{z} + e^{-z})$$

$$\frac{\partial g(z)}{\partial z} = g'(z) = (e^{z} + e^{-z}),\ \frac{\partial h(z)}{\partial z} = h'(z) = (e^{z} - e^{-z})$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (tansig)

$$f(z) = y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)$$

$$\frac{\partial f(z)}{\partial z} = \frac{(e^{z} + e^{-z})^{2} - (e^{z} - e^{-z})^{2}}{(e^{z} + e^{-z})^{2}}$$

$$\frac{\partial f(z)}{\partial z} = (1 - f^{2}(z))$$

Activation Function Derivatives

The Softmax Activation Function Derivative

$$S_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\frac{\partial S_i}{\partial z_m} = \frac{\partial}{\partial z_m}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right);\ j\ =\ 1,2,..m....,K$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when i=m and when i≠m.

Case: 1 i=m

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_i} = \frac{\frac{\partial e^{z_i}}{\partial z_i}\sum_{j=1}^{K} e^{z_j} - e^{z_i}\frac{\partial}{\partial z_i}\sum_{j=1}^{K} e^{z_j}}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when *i*=m and when *i*≠m.

Case: 1 *i*=m

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_i} = \frac{e^{z_i}\sum_{j=1}^{K} e^{z_j} - e^{z_i}e^{z_i}}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when i=m and when i≠m.

Case: 1 i=m

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_i} == \frac{e^{z_i}}{\left(\sum_{j=1}^{K} e^{z_j}\right)} \frac{\left(\sum_{j=1}^{K} e^{z_j} - e^{z_i}\right)}{\left(\sum_{j=1}^{K} e^{z_j}\right)}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when i=m and when i≠m.

Case: 1 i=m

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial}{\partial z_i}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right) = S_i(1 - S_i)$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when *i*=m and when *i*≠m.

Case: 2 *i*≠m

$$\frac{\partial S_i}{\partial z_m} = \frac{\partial}{\partial z_m}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_m} = \frac{\frac{\partial e^{z_i}}{\partial z_m}\sum_{j=1}^{K} e^{z_j} - e^{z_i}\frac{\partial}{\partial z_m}\sum_{j=1}^{K} e^{z_j}}{\left(\sum_{j=1}^{K} e^{z_j}\right)^2}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when i=m and when i≠m.

Case: 2 *i*≠m

$$\frac{\partial S_i}{\partial z_m} = \frac{\partial}{\partial z_m}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_m} = \frac{e^{z_i}}{\left(\sum_{j=1}^{K} e^{z_j}\right)} \frac{-e^{z_m}}{\left(\sum_{j=1}^{K} e^{z_j}\right)}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

We will consider two cases: when *i*=m and when *i*≠m.

Case: 2 *i*≠m

$$\frac{\partial S_i}{\partial z_m} = \frac{\partial}{\partial z_m}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right)$$

$$\frac{\partial S_i}{\partial z_m} = \frac{\partial}{\partial z_m}\left(\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}\right) = S_i\,(0 - S_m)$$

Activation Function Derivatives

The Softmax Activation Function Derivative

Combining both the Cases:

$$\frac{\partial S_i}{\partial z_m} = \begin{cases} S_i(1 - S_m) & for\ i = m \\ S_i(0 - S_m) & for\ i \neq m \end{cases}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

For better understanding consider:

General Case:

$$S = \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_K \end{bmatrix}$$

Example Case:

$$S = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

General Case:

$$\frac{\partial S}{\partial z_m} = \frac{\partial}{\partial z_m}\begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_K \end{bmatrix};\ For\ all\ m\!:\!1\ to\ K$$

Example Case:

$$\frac{\partial S}{\partial z_m} = \frac{\partial}{\partial z_m}\begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix};\ For\ all\ m\!:\!1\ to\ 3$$

Activation Function Derivatives

The Softmax Activation Function Derivative

General Case:

$$\frac{\partial S}{\partial z_1} = \frac{\partial}{\partial z_1}\begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_K \end{bmatrix} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} \\ \dots \\ \frac{\partial S_K}{\partial z_1} \end{bmatrix}$$

Example Case:

$$\frac{\partial S}{\partial z_1} = \frac{\partial}{\partial z_1}\begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} \\ \frac{\partial S_3}{\partial z_1} \end{bmatrix}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

For overall Softmax outputs:

General Case:

$$\frac{\partial S}{\partial z} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} & \frac{\partial S_1}{\partial z_2} & \cdots & \frac{\partial S_1}{\partial z_K} \\ \frac{\partial S_2}{\partial z_1} & \frac{\partial S_2}{\partial z_2} & \cdots & \frac{\partial S_2}{\partial z_K} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial S_K}{\partial z_1} & \frac{\partial S_K}{\partial z_2} & \cdots & \frac{\partial S_K}{\partial z_K} \end{bmatrix}$$

The above expression is called Jacobian Matrix

Activation Function Derivatives

The Softmax Activation Function Derivative

For overall Softmax outputs:

Example Case:

$$\frac{\partial S}{\partial z} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} & \frac{\partial S_1}{\partial z_2} & \frac{\partial S_1}{\partial z_3} \\ \frac{\partial S_2}{\partial z_1} & \frac{\partial S_2}{\partial z_2} & \frac{\partial S_2}{\partial z_3} \\ \frac{\partial S_3}{\partial z_1} & \frac{\partial S_3}{\partial z_2} & \frac{\partial S_3}{\partial z_3} \end{bmatrix}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

$$\frac{\partial S_i}{\partial z_m} = \begin{cases} S_i(1 - S_m) & for\ i = m \\ S_i(0 - S_m) & for\ i \neq m \end{cases}$$

For overall Softmax outputs:

General Case:

$$\frac{\partial S}{\partial z} = \begin{bmatrix} S_1(1 - S_1) & -S_1 S_2 & \dots & -S_1 S_K \\ -S_1 S_2 & S_2(1 - S_2) & \dots & -S_2 S_K \\ \dots & \dots & \dots & \dots \\ -S_1 S_K & -S_2 S_K & \dots & S_K(1 - S_K) \end{bmatrix}$$

Activation Function Derivatives

The Softmax Activation Function Derivative

$$\frac{\partial S_i}{\partial z_m} = \begin{cases} S_i(1 - S_m) & for\ i = m \\ S_i(0 - S_m) & for\ i \neq m \end{cases}$$

For overall Softmax outputs:

Example Case:

$$\frac{\partial S}{\partial z} = \begin{bmatrix} S_1(1 - S_1) & -S_1 S_2 & -S_1 S_3 \\ -S_1 S_2 & S_2(1 - S_2) & -S_2 S_3 \\ -S_1 S_3 & -S_2 S_3 & S_3(1 - S_3) \end{bmatrix}$$

Loss Function

What is Loss Function?

- In neural network training, a loss function (also known as a cost function or objective function) is a mathematical function.
- The Loss function measures the difference between the network's predictions and the actual target values.
- The goal of training a neural network is to minimize this loss function, thereby improving the accuracy of the model's predictions.

Loss Function

What is Loss Function?

- Different types of loss functions are used depending on the task, such as regression, or classification.
- During training, the neural network uses optimization algorithms (like gradient descent) to adjust its weights and biases in order to minimize the loss function.
- The gradients of the loss function with respect to the network's parameters are computed through backpropagation, and these gradients guide the updates to the parameters in each training iteration.

Loss Function

Mean Squared Error (MSE) Loss Function

- MSE loss function is used for Regression Problems.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$

- MSE measures the average squared difference between the actual target values (y_i) and the predicted values $(\hat{y}_i)$.
- Lower MSE indicates better performance.

Loss Function

Mean Absolute Error (MAE) Loss Function

- MAE loss function is used for Regression Problems.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|$$

- MAE measures the average absolute difference between actual (y_i) and predicted values $(\hat{y}_i)$.

- It is less sensitive to outliers compared to MSE.

Loss Function Derivative

(MSE) Loss Function Derivative

$$\frac{\partial}{\partial \hat{y}_i} MSE = \frac{\partial}{\partial \hat{y}_i}\left(\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2\right)$$

$$\frac{\partial}{\partial \hat{y}_i} MSE = \frac{1}{N}\frac{\partial}{\partial \hat{y}_i}\left(y_i - \hat{y}_i\right)^2$$

$$\frac{\partial}{\partial \hat{y}_i} MSE = \frac{2}{N}\left(y_i - \hat{y}_i\right)\frac{\partial}{\partial \hat{y}_i}\left(y_i - \hat{y}_i\right)$$

$$\frac{\partial}{\partial \hat{y}_i} MSE = -\frac{2}{N}\left(y_i - \hat{y}_i\right)$$

Loss Function Derivative

(MAE) Loss Function Derivative

$$\frac{\partial}{\partial \hat{y}_i} MAE = \frac{\partial}{\partial \hat{y}_i}\left(\frac{1}{N}\sum_{i=1}^{N}\left|y_i - \hat{y}_i\right|\right)$$

$$\frac{\partial}{\partial \hat{y}_i} MAE = \frac{1}{N}\frac{\partial}{\partial \hat{y}_i}\left|y_i - \hat{y}_i\right|$$

$$\frac{\partial}{\partial \hat{y}_i} MAE = \frac{1}{N}\begin{cases} -1 & \left(y_i - \hat{y}_i\right) > 0 \\ 1 & \left(y_i - \hat{y}_i\right) < 0 \end{cases}$$

Loss Function

Categorical Cross-Entropy Loss

- It is used for multi-class classification tasks where the labels are one-hot encoded.
- The Loss function measures the difference

$$\mathrm{L}oss = -\sum_{i=1}^{K} y_i \log(S_i)$$

where K is the number of classes, y_i is the true label (1 for the correct class, 0 otherwise), and S_i is the predicted probability for class i.

- This loss value indicates how well the model predicted the correct class. The lower the loss, the better the prediction.

Loss Function

Categorical Cross-Entropy Loss

- **Example Case:** Consider a 3-class classification problem where the true label is class 3 (one-hot encoded as [0,0,1]), and the model predicts probabilities [0.1,0.3,0.6].

$$y = [0, 0, 1]$$

$$S = [0.1, 0.3, 0.6]$$

$$\mathrm{L}oss = -\sum_{i=1}^{3} y_i \log(S_i)$$

$$= -y_1 \log(S_1) - y_2 \log(S_2) - y_3 \log(S_3) = 0.511$$

Loss Function

Categorical Cross-Entropy Loss

- **Example Case:** Consider a 3-class classification problem where the true label is class 3 (one-hot encoded as [0,0,1]), and the model predicts probabilities [0.1,0.3,0.6].

```python
import numpy as np

# True label (y) (one-hot encoded)
true_label = np.array([0, 0, 1])

# Predicted probabilities (S)
predicted_probabilities = np.array([0.1, 0.3, 0.6])

# Compute the categorical cross-entropy loss
Loss = -np.sum(true_label * np.log(predicted_probabilities))

# Output the result
print("Categorical Cross-Entropy Loss:", Loss)
```

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

$$\mathrm{L}oss = -\sum_{i=1}^{K} y_i \log(S_i)$$

$$\frac{\partial}{\partial S_i}\mathrm{L}oss = \frac{\partial}{\partial S_i}\left(-\sum_{i=1}^{K} y_i \log(S_i)\right)$$

$$\frac{\partial}{\partial S_i}\mathrm{L}oss = -\frac{y_i}{S_i}$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

$$\frac{\partial}{\partial S_i} \mathrm{L}oss = -\frac{y_i}{S_i}$$

```python
import numpy as np

# Example data
true_label = np.array([0, 1, 0])  # One-hot encoded true label
predicted_probabilities = np.array([0.1, 0.7, 0.2])  # Predicted probabilities

# Calculate the Derivative of the Loss
derivative = -true_label / predicted_probabilities

print("Derivative of Loss:", derivative)
```

```
Derivative of Loss: [ 0.         -1.42857143  0.        ]
```

Loss Function Derivative

Categorical Cross-Entropy Loss

$$\mathrm{L}oss = -\sum_{i=1}^{K} y_i \log(S_i)$$

$$\frac{\partial \mathrm{L}oss}{\partial z_m} = \sum_{i=1}^{K} \frac{\partial \mathrm{L}oss}{\partial S_i} \frac{\partial S_i}{\partial z_m}$$

The summation is used to solve the Jacobian matrix terms involved in the above product.

$$\frac{\partial \mathrm{L}oss}{\partial z_i} = S_i - y_i \; ; \; when \; i = m$$

$$\frac{\partial \mathrm{L}oss}{\partial z_m} = y_i S_m \; ; \; when \; i \neq m$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

- Example case to understand it properly: Let

$$S = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix}$$

$$\frac{\partial S}{\partial z} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} & \frac{\partial S_1}{\partial z_2} & \frac{\partial S_1}{\partial z_3} \\ \frac{\partial S_2}{\partial z_1} & \frac{\partial S_2}{\partial z_2} & \frac{\partial S_2}{\partial z_3} \\ \frac{\partial S_3}{\partial z_1} & \frac{\partial S_3}{\partial z_2} & \frac{\partial S_3}{\partial z_3} \end{bmatrix}$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

$$\frac{\partial \mathrm{L}oss}{\partial z_m} = \sum_{i=1}^{K} \frac{\partial \mathrm{L}oss}{\partial S_i} \frac{\partial S_i}{\partial z_m}$$

$$S = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix}$$

$$\frac{\partial S}{\partial z_1} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} \\ \frac{\partial S_3}{\partial z_1} \end{bmatrix} \qquad \frac{\partial Loss}{\partial S} = \begin{bmatrix} \frac{\partial Loss}{\partial S_1} \\ \frac{\partial Loss}{\partial S_2} \\ \frac{\partial Loss}{\partial S_3} \end{bmatrix}$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

$$\frac{\partial \mathrm{L}oss}{\partial z_1} = \sum_{i=1}^{K} \frac{\partial \mathrm{L}oss}{\partial S_i} \frac{\partial S_i}{\partial z_1} =$$

$$\begin{bmatrix} \frac{\partial Loss}{\partial S_1} & \frac{\partial Loss}{\partial S_2} & \frac{\partial Loss}{\partial S_3} \end{bmatrix} \begin{bmatrix} \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} \\ \frac{\partial S_3}{\partial z_1} \end{bmatrix}$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

$$\frac{\partial \mathrm{L}oss}{\partial z_1} = \sum_{i=1}^{K} \frac{\partial \mathrm{L}oss}{\partial S_i} \frac{\partial S_i}{\partial z_1} =$$

$$\begin{bmatrix} \frac{\partial Loss}{\partial S_1} & \frac{\partial Loss}{\partial S_2} & \frac{\partial Loss}{\partial S_3} \end{bmatrix} \begin{bmatrix} \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} \\ \frac{\partial S_3}{\partial z_1} \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{y_1}{S_1} & -\frac{y_2}{S_2} & -\frac{y_3}{S_3} \end{bmatrix} \begin{bmatrix} S_1(1 - S_1) \\ -S_2 S_1 \\ -S_3 S_1 \end{bmatrix}$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

- **Example Case:** Consider a 3-class classification problem where the true label is class 3 (one-hot encoded as [0,0,1]), and the model predicts probabilities [S_1, S_2, S_3].

$$\frac{\partial Loss}{\partial z_1} = \begin{bmatrix} -\frac{y_1}{S_1} & -\frac{y_2}{S_2} & -\frac{y_3}{S_3} \end{bmatrix} \begin{bmatrix} S_1(1 - S_1) \\ -S_2 S_1 \\ -S_3 S_1 \end{bmatrix}$$

$$= y_1(S_1 - 1) + y_2 S_1 + y_3 S_1$$

$$as\ y_1 = 0,\ y_2 = 0,\ y_3 = 1$$

$$\frac{\partial Loss}{\partial z_1} = S_1 = S_1 - y_1$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

- **Example Case:** Consider a 3-class classification problem where the true label is class 3 (one-hot encoded as [0,0,1]), and the model predicts probabilities [S_1, S_2, S_3].

$$\frac{\partial Loss}{\partial z_2} = \begin{bmatrix} -\frac{y_1}{S_1} & -\frac{y_2}{S_2} & -\frac{y_3}{S_3} \end{bmatrix} \begin{bmatrix} -S_1 S_2 \\ S_2(1 - S_2) \\ -S_3 S_2 \end{bmatrix}$$

$$= y_1 S_2 + y_2(S_2 - 1) + y_3 S_2$$

$$as\ y_1 = 0,\ y_2 = 0,\ y_3 = 1$$

$$\frac{\partial Loss}{\partial z_1} = S_2 = S_2 - y_2$$

Loss Function Derivative

Categorical Cross-Entropy Loss Derivative

- **Example Case:** Consider a 3-class classification problem where the true label is class 3 (one-hot encoded as [0,0,1]), and the model predicts probabilities [S_1, S_2, S_3].

$$\frac{\partial Loss}{\partial z_3} = \begin{bmatrix} -\frac{y_1}{S_1} & -\frac{y_2}{S_2} & -\frac{y_3}{S_3} \end{bmatrix} \begin{bmatrix} -S_1 S_3 \\ -S_2 S_3 \\ S_3(1 - S_3) \end{bmatrix}$$

$$= y_1 S_3 + y_2 S_3 + y_3 (S_3 - 1)$$

$$as \;\; y_1 = 0, \;\; y_2 = 0, \;\; y_3 = 1$$

$$\frac{\partial Loss}{\partial z_3} = S_3 - 1 = S_3 - y_3$$

Loss Function Derivative

Categorical Cross-Entropy Loss

$$\frac{\partial \mathrm{L}oss}{\partial z_i} = S_i - y_i \;;\; when\ i = m$$

$$\frac{\partial \mathrm{L}oss}{\partial z_m} = y_i S_m \;;\; when\ i \neq m$$

$$the\ \ \mathrm{sec}ond\ term\ y_i S_m = 0\ ;\ when\ i \neq m$$

$$therefore\ \frac{\partial \mathrm{L}oss}{\partial z_i} = S_i - y_i$$

Loss Function Derivative

Categorical Cross-Entropy Loss

$$\frac{\partial \mathrm{L}oss}{\partial z_i} = S_i - y_i$$

```python
import numpy as np

def softmax(x):
    e_x = np.exp(x)
    return e_x / e_x.sum()
```

```python
# Example data
z = np.array([2.0, 1.0, 0.1])
y_true = np.array([0, 1, 0])

# Step 1: Calculate Softmax Output
S = softmax(z)

print("Softmax Output:", S)

# Step 2: Derivative of the Loss with respect to z
derivative_Loss_dz = S - y_true

print("Derivative of Loss with respect to z:", derivative_Loss_dz)
```

```
Softmax Output: [0.65900114 0.24243297 0.09856589]
Derivative of Loss with respect to z: [ 0.65900114 -0.75756703  0.09856589]
```

Activation Function Derivatives

The Linear Activation Function Derivative

$$f(z) = y = z$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial z}{\partial z} = 1$$

Activation Function Derivatives

The Linear Activation Function & Derivative

Implementation

```python
import numpy as np

# Linear activation function
def linear_activation(z):
    return z

# Derivative of the linear activation function
def linear_derivative(z):
    return np.ones_like(z)

# Example
z = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])

# Calculate the activation
activation = linear_activation(z)
print("Linear Activation:", activation)

# Calculate the derivative
derivative = linear_derivative(z)
print("Derivative:", derivative)
```

Activation Function Derivatives

The ReLU Activation Function Derivative

$$f(z) = \mathrm{Re}LU(z) = \max(0, z)$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial \mathrm{Re}LU(z)}{\partial z} = \begin{cases} 1 & z \geqslant 0 \\ 0 & z < 0 \end{cases}$$

Activation Function Derivatives

The ReLU Activation Function Derivative

```python
import numpy as np

# ReLU activation function
def relu_activation(x):
    return np.maximum(0, x)

# Derivative of the ReLU activation function
def relu_derivative(x):
    return np.where(x > 0, 1, 0)

# Example usage
x = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])

# Calculate the activation
activation = relu_activation(x)
print("ReLU Activation:", activation)

# Calculate the derivative
derivative = relu_derivative(x)
print("Derivative:", derivative)
```

Activation Function Derivatives

The Leaky ReLU Activation Function Derivative

$$f(z) = \text{Leaky ReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases} ; 0 < \alpha < 0.1$$

$$\frac{\partial}{\partial z} \text{Leaky ReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z \leq 0 \end{cases}$$

Activation Function Derivatives

The Leaky ReLU Activation Function Derivative

```python
import numpy as np

# Leaky ReLU activation function
def leaky_relu_activation(x, alpha=0.01):
    return np.where(x > 0, x, alpha * x)

# Derivative of the Leaky ReLU activation function
def leaky_relu_derivative(x, alpha=0.01):
    return np.where(x > 0, 1, alpha)

# Example usage
x = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])

# Calculate the activation
activation = leaky_relu_activation(x)
print("Leaky ReLU Activation:", activation)

# Calculate the derivative
derivative = leaky_relu_derivative(x)
print("Derivative:", derivative)
```

Activation Function Derivatives

The Sigmoid Activation Function Derivative (logsig)

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

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right)$$

$$\frac{\partial f(z)}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^{2}}$$

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

$$\frac{\partial f(z)}{\partial z} = f(z)\,(1-f(z))$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (logsig)

```python
import numpy as np

# Log-sigmoid (standard sigmoid) activation function
def log_sigmoid_activation(x):
    return 1 / (1 + np.exp(-x))

# Derivative of the log-sigmoid activation function
def log_sigmoid_derivative(x):
    sigmoid = log_sigmoid_activation(x)
    return sigmoid * (1 - sigmoid)

# Example usage
x = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])

# Calculate the activation
activation = log_sigmoid_activation(x)
print("Log-Sigmoid Activation:", activation)

# Calculate the derivative
derivative = log_sigmoid_derivative(x)
print("Derivative:", derivative)
```

Activation Function Derivatives

The Sigmoid Activation Function Derivative (tansig)

$$f(z) = y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\frac{\partial f(z)}{\partial z} = \frac{\partial y}{\partial z} = \frac{\partial}{\partial z}\left(\frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}\right)$$

$$\frac{\partial f(z)}{\partial z} = \frac{(e^{z} + e^{-z})^{2} - (e^{z} - e^{-z})^{2}}{(e^{z} + e^{-z})^{2}}$$

$$\frac{\partial f(z)}{\partial z} = (1 - f^{2}(z))$$

Activation Function Derivatives

The Sigmoid Activation Function Derivative (tansig)

```python
import numpy as np

# tan_sigmoid activation function
def tan_sigmoid_activation(x):
    return np.tanh(x)

# Derivative of the tan_sigmoid activation function
def tan_sigmoid_derivative(x):
    return 1 - np.tanh(x)**2

# Example usage
x = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])

# Calculate the activation
activation = tan_sigmoid_activation(x)
print("Tanh Activation:", activation)

# Calculate the derivative
derivative = tan_sigmoid_derivative(x)
print("Derivative:", derivative)
```