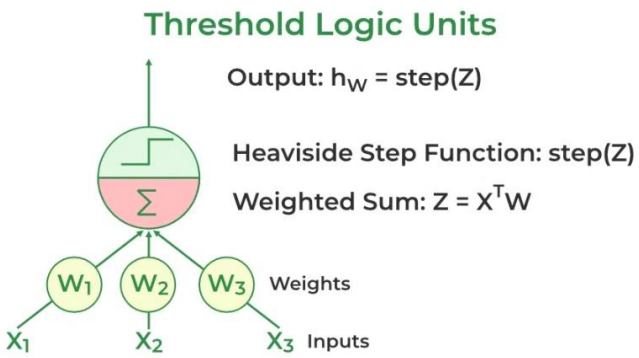
REPORT ABOUT NEURAL NETWORKS

1. Perceptron

- draw of model architecture



- vector representation of data (inputs and outputs)

Inputs: $x = [x_1, x_2, \dots, x_n]^T$
Weights: $w = [w_1, w_2, \dots, w_n]^T$
Bias: b, an additional parameter that adjusts the output independently of the input values.
Output: y, the target value (scalar or vector for multi-class problems).
Predicted output: $\hat{y}$, the model's output after applying the activation function.

- math formulation of linear combination, activation function and loss function

Linear combination

$$z = w^T x + b = \sum_{i=1}^{n} w_i x_i + b$$

Activation function

The activation function $f(z)$ introduces non-linearity. Common choices include:

1) Step function: $f(z) = \begin{cases} 1, if\ z \geq 0 \\ 0, otherwise \end{cases}$
2) Sigmoid $f(z) = \frac{1}{1+e^{-z}}$
3) Rectified Linear Unit: $f(z) = \max(0, z)$

Loss function

The loss function quantifies the difference between the true output y and the predicted output $\hat{y}$.

For classification (Cross-Entropy Loss):

$$L = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

For regression (Mean Squared Error):

$$L = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- math formulation of how neural nets calculate the predictions (y_hat)

The predicted output $\hat{y}$ is computed as:

$$\hat{y} = f(z) = f(w^T x + b)$$

- explanation of gradient descendent algorithm

Gradient descent is used to optimize the weights and biases by minimizing the loss function. The algorithm updates parameters iteratively in the opposite direction of the gradient of the loss with respect to the parameters.

- formulas of gradients and weights/biases updates

Gradients calculation

1) Gradient of the loss with respect to weights:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_i}$$

2) Gradient of the loss with respect to bias:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial b}$$

Parameters updates

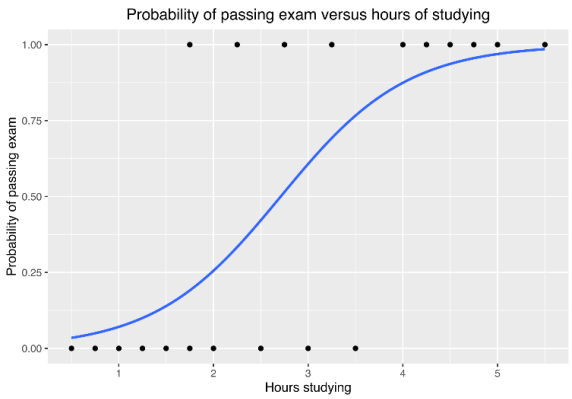
Using learning rate η, the updates are:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{\partial w_i}$$

$$b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

2. Logistic Regression

- draw of model architecture



- vector representation of data (inputs and outputs)

Inputs: $x = [x_1, x_2, \ldots, x_n]^T$
Weights: $w = [w_1, w_2, \ldots, w_n]^T$
Bias: b.
Output (Target): binary value $y \in \{0,1\}$.

Predicted output: $\hat{y}$, a probability that the target class is 1.

- math formulation of linear combination, activation function and loss function

Linear combination

The linear combination of inputs, weights, and bias is:

$$z = w^T x + b = \sum_{i=1}^{n} w_i x_i + b$$

Activation gunction

The sigmoid function transforms z into a probability:

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

Loss function

Logistic regression uses a binary cross-entropy loss to measure the difference between predicted probabilities $\hat{y}$ and true labels y:

$$L = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where m is the number of training samples.

- math formulation of how neural nets calculate the predictions (y_hat)

The predicted output $\hat{y}$ is the probability that the target class is 1:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-(w^T x + b)}}$$

- explanation of gradient descendent algorithm

Logistic regression uses gradient descent to minimize the loss function by iteratively updating weights and biases. The steps are:

1. Compute the gradient of the loss function with respect to weights w and bias b.
2. Update the parameters in the direction opposite to the gradient, scaled by the learning rate η.

- formulas of gradients and weights/biases updates

Gradients calculation

Gradient with respect to weights:

$$\frac{\partial L}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_{ij}$$

Gradient with respect to bias:

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

Parameters updates

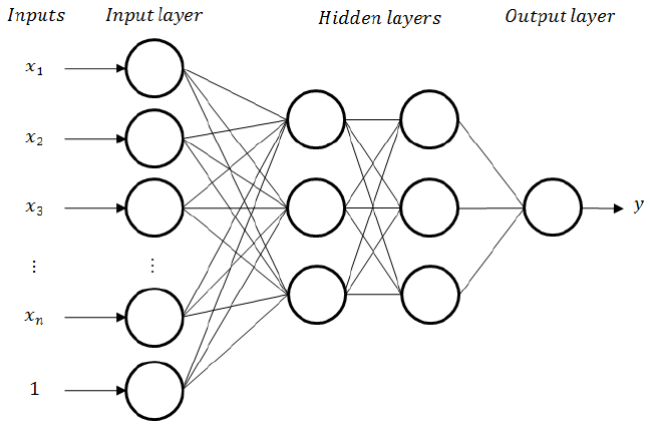
Using the learning rate η, the weights and bias are updated as:

$$w_j \leftarrow w_j - \eta\frac{\partial L}{\partial w_j}$$

$$b \leftarrow b - \eta\frac{\partial L}{\partial b}$$

3. Multilayer Perceptron

- draw of model architecture



- vector representation of data (inputs and outputs)

Inputs: $x = [x_1, x_2, \dots, x_n]^T$, where n is the number of features.
Hidden layer output: $h^{(l)} = [h_1^{(l)}, h_2^{(l)}, \dots, h_m^{(l)}]^T$, where m is the number of neurons in the l-th layer.
Weights matrices: $W^{(l)} \in R^{m \times n}$ for layer l.
Bias: $b^{(l)} \in R^m$.
Output: $\hat{y} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_k]^T$, where k is the number of output classes or predictions.

- math formulation of linear combination, activation function and loss function

Linear combination

For each layer l, the linear combination is:

$$z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)}$$

where $h^{(l-1)}$ is the output of the previous layer (or x for the input layer).

Activation function

Each neuron applies a non-linear activation function f:

$$h^{(l)} = f(z^{(l)})$$

Common activation functions:

1. Rectified linear unit: $f(z) = \max(0, z)$
2. Sigmoid: $f(z) = \frac{1}{1+e^{-z}}$
3. Tanh: $f(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

Loss function

The loss function quantifies the difference between true outputs y and predictions $\hat{y}$.

Cross-Entropy Loss (for classification):

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij})$$

Mean Squared Error (for regression):

$$L = \frac{1}{m}\sum_{i=1}^{m} ||\hat{y}_i - y_i||^2$$

- math formulation of how neural nets calculate the predictions (y_hat)

Predictions are computed by forward propagation:

1) Compute linear combination
2) Apply activation function
3) Final layer outputs $\hat{y} = f(z^{(L)})$, where L is the total number of layers.

- explanation of gradient descendent algorithm

MLP uses backpropagation with gradient descent to optimize the weights and biases. Backpropagation computes gradients layer by layer, starting from the output layer and moving backward.

Steps:

1. Forward pass: calculate predictions $\hat{y}$.
2. Compute loss: use the loss function L.
3. Backward pass: compute gradients of L with respect to $W^{(l)}$ and $b^{(l)}$.
4. Update parameters: use the gradients to update weights and biases.

- formulas of gradients and weights/biases updates

Gradients calculation

Gradient of loss with respect to weights in layer l:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}(h^{(l-1)})^T$$

Gradient of loss with respect to biases in layer l:

$$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

where $\delta^{(l)}$ is the error term for layer l:

$$\delta^{(l)} = \frac{\partial L}{\partial h^{(l)}} * f'(z^{(l)})$$

Parameters updates

Using learning rate η:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}}$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}}$$