Task_7.3

Machine Learning (WiSe 2025/2026)

Author: Suvansh Shukla
Matriculation No. 256245

Assignment 7 Task 7.3

Part A

A neural network looks like web of nodes connected to each other across different layers of nodes.
Information flows in these nodes from one direction to another across these layers. With the input layer handling input,
one or more hidden layers handling processing and output layer. Each of these nodes across layers are further connected
to each other through a weighted connection. Each node (or neuron) also has a bias value.

The formula for the net input to a neuron in the next layer is the sum of the weighted outputs from the previous layer
plus its bias. So something like this:

$$z_j = \left(\sum_i w_{ij} a_i\right) + b_j$$

The output of the neuron is this net input passed through an activation function (σ): $a_j = \sigma(z_j)$

PART B

A neural network classifies data by segregating them into discrete sections or areas. The output layer of the neural network
has a number of neurons equal to the number of classes. The neural network takes input features, processes them through the
hidden layers then outputs an array of numerical scores.

The final layer often uses a **Softmax** activation function that converts the raw output scores into a vector of probabilities
that sum to 1. The class with the highest probability is the network's prediction.

Softmax function formula:

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}}$$

Where C is the number of classes.

PART C

A neural network is learned by comparing the error/deviation received for a single instance using a "loss function"

and then sending it backwards throughout the network, this is called "Backpropagation". This error/deviation is
then used to adjust the weights of each of the nodes until it sufficiently reduces the error a close as possible
to zero. Once the error is reduced to zero for the entire training dataset, the learning is completed.

An example of the Loss Function is the Mean Squared Error:

$$S = \frac{1}{2} \sum_{i=1} \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

The weight update rule or Gradient Descent formula is this:

$$w_{new} = w_{old} - \eta \overline{w}$$

The problem of "vanishing gradients" is a problem that occurs when we try to find the error gradient for nodes in previous nodes.
The problem specifically lies in the gradient generated after the error is propagated backwards throughout the network, is too small
or miniscule to help us find the minimum adjustment required. This happens because Backpropagation calculates gradients by multiplying
the gradients of each layer. If many of these local gradients are small the products of these small numbers across many layers becomes
exponentially tiny, causing the weights in the initial layer to update minimally or not at all.

PART D

Neurons should not all be initialized to the same values for all weights:

1. to prevent neural networks from getting stuck in local minimas, this is to prevent such an occurrence
2. if all neurons start at the same weights then all neurons
   will follow the same gradient and will always end up doing the same thing as one another[1].

More on point 2, is to break symmetry. If all weights are the same, all neurons in the same hidden layer will compute the exact same output, receive the exact same gradient during Backpropagation, and thus update their weights identically. The network would effectively have only one hidden neuron per layer.

We need non-linearity in neural networks because: "without a non-linear activation function in the network, a neural network, no matter how many layers it had, would behave just like a single-layer perceptron, because summing these layers would give you just another linear function"[2]

---

1. David J. Harris (https://stats.stackexchange.com/users/4862/david-j-harris), Danger of setting all initial weights to zero in Backpropagation, URL (version: 2012-04-28): https://stats.stackexchange.com/q/27152 ↩

2. https://stackoverflow.com/a/9783865 ↩