Unit 8: Training an Artificial Neural Network
Unit 8 Artefact

e-Portfolio Activity: Gradient Cost Function

I went through this tutorial gradient_descent_cost_function.ipynb after reading the article Mayo, 2017).

Essentially, my understanding of the key concepts in the tutorial is as follows:

Weights = parameters associated with connections between neurons in ANN that determine the strength and direction of the input signal's influence on the neuron's output.

During training, these weights are adjusted ultimately to minimise the error between the actual and predicted values.

Error calculation = Error is the difference between actual output i.e. the true value and the produced output from the model. The error is used for making weight adjustments as it indicates how far off the model's predictions are from the actual values.

Backpropagation of error = the error moves backward through the network, meaning it starts at the output layer and moves toward the input layer. The error is used to update the weights of each connection between neurons in a way that reduces the overall error in subsequent training cycles.

Gradient descent is an optimisation algorithm that minimises the cost function (aka error) by iteratively adjusting the weights. The cost function (aka the loss function) measures how well the ANN predicts the actual values. The ultimate goal of this process is to find the set of weights that minimises the cost function because this means the model predicts the actual values very well.

Updating weights in the ANN involves calculating the error between predicted and actual values, using backpropagation to distribute this error backward through the network, and applying gradient descent to iteratively adjust the weights to minimise the error. Using methods like stochastic gradient descent introduces randomisation into the process that may lead to better generalisation and faster convergence.

I changed the values of iterations and learning_rate to see how the cost changed:

The number of iterations determines how many times the gradient descent algorithm will update the parameters. Each iteration involves computing the gradients, updating the parameters, and recalculating the cost. If the number of iterations is too low, the algorithm may not have enough time to converge to the minimum cost. This means the final parameters might still be far from the optimal values, resulting in a higher cost. If the number of iterations is excessively high, this may be wasteful in terms of excessive computation. The learning rate is essentially the sizes of the steps taken towards the minimum of the cost function. Higher rates mean larger

steps and lower rates mean smaller steps. Finding the optimum rate will make the algorithm more efficient. An appropriate learning rate should be chosen to ensure convergence. Too high can cause divergence, and too low can cause slow convergence (Kubat, 2017; Mayo, 2017).

Ultimately, the goal is to have enough iterations for the algorithm to converge on the minimum. I think of it as the optimal number of iterations is sufficient when the cost function no longer decreases significantly with additional iterations. The global cost minimum is the lowers point on the cost function landscape representing the best possible values (Mayo, 2017).

References:

Kubat, M., 2017. An introduction to machine learning. Springer.

Mayo. Neural Network Foundations, Explained: Updating Weights with Gradient Descent & Backpropagation. 2017. Online at: https://www.kdnuggets.com/2017/10/neural-network-foundations-explained-gradient-descent.html Accessed 29.05.2024.