1 Neural Networks for Classification and Regression

In this part of the assignment, you will implement a feedforward neural network from scratch. Additionally, you will implement multiple activation functions, loss functions, and performance metrics. Lastly, you will train a neural network model to perform a classification and a regression problem.

1.1 Bank Note Forgery - A Classification Problem

The classification problem we will examine is the prediction of whether or not a bank note is forged. The labelled dataset included in the assignment was downloaded from the UCI Machine Learning Repository. The target $y \in \{0, 1\}$ is a binary variable, where 0 and 1 refer to fake and real respectively. The features are all real-valued. They are listed below:

- Variance of the transformed iamge of the bank note
- Skewness of the transformed iamge of the bank note
- Curtosis of the transformed iamge of the bank note
- Entropy of the image

1.2 Red Wine Quality - A Regression Problem

The task is to predict the quality of red wine from northern Portugal, given some physical characteristics of the wine. The target $y \in [0, 10]$ is a continuous variable, where 10 is the best possible wine, according to human tasters. Again, this dataset was downloaded from the UCI Machine Learning Repository. The features are all real-valued. They are listed below:

- Fixed acidity
- Volatile acidity
- Citric acid
- Residual sugar
- Chlorides
- Free sulfur dioxide
- Total sulfur dioxide
- Density
- pH
- Sulphates
- Alcohol

1.3 Training a Neural Network

In Lecture 14, you learned how to train a neural network using the backpropagation algorithm. In this assignment, you will apply the forward and backward pass to the entire dataset simultaneously (i.e. batch gradient decsent). As a result, your forward and backward passes will manipulate tensors, where the first dimension is the number of examples in the training set, n. When updating an individual weight $W_{i,j}^{(l)}$, you will need to find the average gradient $\frac{\partial E}{\partial W_{i,j}^{(l)}}$ across all examples in the training set to apply the update. Algorithm 1 gives the training algorithm in terms of functions that you will implement in this assignment. Further details can be found in the documentation for each function in the provided source code.

Algorithm 1 Neural network training

Require: $\eta > 0$ ▷ Learning rate
Require: $n_{epochs} \in N^+$ ▷ Number of epochs
Require: $X \in R^{n \times f}$ ▷ Training examples with n examples and f features
Require: $y \in R^n$ ▷ Targets for training examples
  Initiate weight matrices $W^{(l)}$ randomly for each layer. ▷ Initialize `net`
  for $i \in \{1, 2, \ldots, n_{epochs}\}$ **do** ▷ Conduct n_{epochs} epochs
    `A_vals, Z_vals` $\leftarrow$ `net.forward_pass(`X`)` ▷ Forward pass
    $\hat{y} \leftarrow$ `Z_vals[-1]` ▷ Predictions
    $L \leftarrow \mathcal{L}(\hat{y}, y)$
    Compute $\frac{\partial}{\partial \hat{y}} \mathcal{L}(\hat{y}, y)$ ▷ Derivative of error with respect to predictions
    `deltas` $\leftarrow$ `backward_pass(A_vals,` $\frac{\partial}{\partial \hat{y}} \mathcal{L}(\hat{y}, y)$ `)` ▷ Backward pass
    `update_gradients()` ▷ $W_{i,j}^{(\ell)} \leftarrow W_{i,j}^{(\ell)} - \eta E_n \frac{\partial \mathcal{L}}{\partial W_{i,j}^{(\ell)}}$ for each weight
  end for
  return trained weight matrices $W^{(\ell)}$

1.4 Activation and Loss Functions

You will implement the following activation functions and their derivatives:

Sigmoid

$$g(x) = \frac{1}{1 + e^{-kx}}$$

Hyperbolic tangent

$$g(x) = \tanh x$$

ReLU

$$g(x) = \max(0, x)$$

Leaky ReLU

$$g(x) = \max(0, x) + \min(0, kx)$$

You will implement the following loss functions and their derivatives:

Cross entropy loss: for binary classification

Compute the average over all the examples. Note that log() refers to the natural logarithm.

$$\mathcal{L}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Mean squared error loss: for regression

$$\mathcal{L}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y} - y)^2$$