SCC.323 DEEP LEARNING

WEEK 3 LAB SHEET – DERIVATIVE FUNCTIONS, PERCEPTRON AND MLP

EXERCISE 1: ACTIVATION DERIVATIVE FUNCTIONS

We have seen in lectures that to solve models, we need to be able to calculate and use the derivatives of our activation functions. Last week, we created an activation functions python file and we have seen some derivations of the activation derivatives in lectures.

In this exercise, come back to your "activations.py" file and update it to include the derivatives of the following activation functions and any others that you included:

- ReLU
- Sigmoid
- Hyperbolic Tangent (tanh)
- Sigmoid Linear Unit (SiLU)

If you are comfortable with the calculations, you could do the rest outside of the lab:

- Parametric ReLU (PReLU) function with a default parameter of 0.1.
- Leaky ReLU. Hint – you can use your PReLU function here.
- Parametrised Sigmoid with a default α value of 1.
- Softplus
- Scaled Exponential Linear Unit with default $\lambda = 1$.
- Exponential Linear Sigmoid SquasHing (ELiSH)
- Soboleva Modified Hyperbolic Tangent
- Gaussian Error Linear Unit (GeLU)

Plot the derivatives alongside the activation functions to see the relationship between them.

EXERCISE 2: IMPLEMENTING MLP FROM SCRATCH

In this exercise, we will implement from scratch an MLP with 1 hidden layer having n nodes in the hidden layer. This MLP will be used to solve the XOR problem that Perceptron could not solve, so you will only need one output node, but you can extend it to multiple outputs later if you prefer.

1. First, you will need to create a loss function. It would be useful to start building a collection of loss functions in a .py file called, for example, "lossFunctions.py". Add a definition to this file for a loss function called "MSE_loss", given by

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

   where n_s is the number of samples; $y_1, \dots y_{n_s}$ are the ground truth labels of the samples; and $\widehat{y_1}, \dots \widehat{y_{n_s}}$ are the predicted labels. This should take the ground truth label y and predicted label $\hat{y}$ as input and output the mean squared error.
2. Next, you will need to initialize the training by specifying:

   a. The number of input features, or you could determine this using your training data.
   b. The number of nodes that you want to include in your hidden layer.
   c. The number of outputs; use 1 for binary classification or n for non-binary classification with n classes.
   d. The learning rate
   e. The maximum number of iterations
   f. The set of weights $w_{i,j}^{1}$ going from your input layer to your hidden layer. You can initialize these as random values and store them in a matrix (array) of size $m \times n$ where m is the number of input nodes, n is the number of hidden nodes.
   g. The set of weights $w_{i,j}^{2}$ going from your hidden layer to your output layer. You can initialize these as random values and store them in a matrix (array) of size $m \times n$ where m is the number of hidden nodes, n is the number of output nodes.
3. Now, start a loop, which should run to your maximum number of iterations.
   a. Make the forward pass of your input data, storing the linear combination values and activated values (those achieved from using the activation function) in variables for use later.
   b. Calculate the loss.
   c. Determine the weights leading to the output layer using the equations from the lecture.
   d. Determine the weights leading to the hidden layer using the equations from the lecture.
4. Finally, write a code for making a prediction from your testing data. This should be the same as making a forward pass through the model but using your new updated weights.