SCC.323 DEEP LEARNING

WEEK 2 LAB SHEET – ACTIVATION FUNCTIONS, PERCEPTRON

EXERCISE 1: ACTIVATION FUNCTIONS

In this exercise, you are asked to write your own “activations.py” file to act as your own library. Add the relu function to your activations file and then add function definitions for the following activation functions. Keep in mind that your input values will probably be arrays rather than single values.

- Heaviside Step
- Sigmoid
- Hyperbolic Tangent (tanh)
- 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.
- Softsign

You can also add the following functions, but if you’ve got the idea, you may wish to do this later.

- Sigmoid Linear Unit (SiLU)
- Softplus
- Scaled Exponential Linear Unit with default $\lambda = 1$.
- Exponential Linear Sigmoid SquasHing (ELiSH)
- Soboleva Modified Hyperbolic Tangent
- Gaussian Error Linear Unit (GeLU)

Test to see if your library works by importing it into an iPython notebook. Create a vector x with 1000 linearly spaced values between -5 and 5, and use this to plot graphs of your activation functions.

EXERCISE 2: PERCEPTRON SOLVER

In this exercise, you are asked to implement your own Perceptron, including the solution algorithm using the update rule:

$$\hat{\boldsymbol{w}} = \boldsymbol{w} + \ell \cdot \delta \cdot \boldsymbol{x}$$

where:

- $\boldsymbol{w}$ is the weights vector and $\hat{\boldsymbol{w}}$ denotes the updated weights
- ℓ is the learning rate
- $\delta = y_i - \hat{y}_i$ is the difference between the ground truth label y_i and the updated label $\hat{y}_i$ for the ith sample
- $\boldsymbol{x}_i$ is in the input data for the ith sample

The following steps will be needed:

1. Generate some training and testing data. You can use AND data from week 1, the Iris dataset, or any other data you prefer.

2. Visualise the data to see if the data is linear separable. It is not essential that your data be linear separable but if it is not then Perceptron will not be able to give a perfect answer. It may however give a good approximation.
3. Train the Perceptron:
   a. Initialise weights and bias. You could include them in the same vector or have the bias as a separate variable. You could use (1,1) for the weights and 0.1 for the bias, as in the lecture, or you could initialize with all zeros, which is common practice.
   b. Set a maximum number of iterations; 1000 would be reasonable. `max_iter` is a common variable name for this.
   c. Set a learning rate. You could try 0.1 or 0.5 initially. `lr` is a common variable name for this.
   d. Use a for loop to cycle over each iteration up to your maximum number of iterations, and through each element of your training samples. NB: you will need to use every sample in each iteration. At each step:
      i. Calculate the forward pass of your training data through the perceptron, i.e.
      $$\widehat{y_i} = \varphi(\boldsymbol{x}_i \cdot w)$$
      where φ is the Heaviside function. You can use the activations.py file that you created above for this.
      ii. Calculate the updated weights, i.e.
      $$\widehat{\boldsymbol{w}} = \boldsymbol{w} + \ell \cdot \delta \cdot \boldsymbol{x}_i, \qquad \delta = y - \hat{y}$$
4. Once your perceptron is trained, use it to make a prediction on your testing data and calculate the mean squared error
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$$
5. Plot your training data on a graph with the decision boundary. To create the decision boundary, you could create a linearly-spaced vector x of values ranging from the minimal to maximal values of the first feature in your training set (appearing on the x-axis of your graph) and calculate
$$y = -\frac{\widehat{w_2}x + \widehat{w_0}}{\widehat{w_1}}$$
where $\widehat{w_0}$ is your bias and $\widehat{w_1}, \widehat{w_2}$ are your weights.

EXERCISE 3: CREATE A MODEL CLASS (OPTIONAL)

Rewrite your Perceptron as a class with fit and predict functions. The inputs for the class should be the learning rate and number of iterations. You can use a fixed initialization of zeros for the weights and bias.

Try training your perceptron with your training data. You will first need to initialize the model by calling the class, and then use the .fit function to train the model and the .predict function to make a prediction on your testing data.