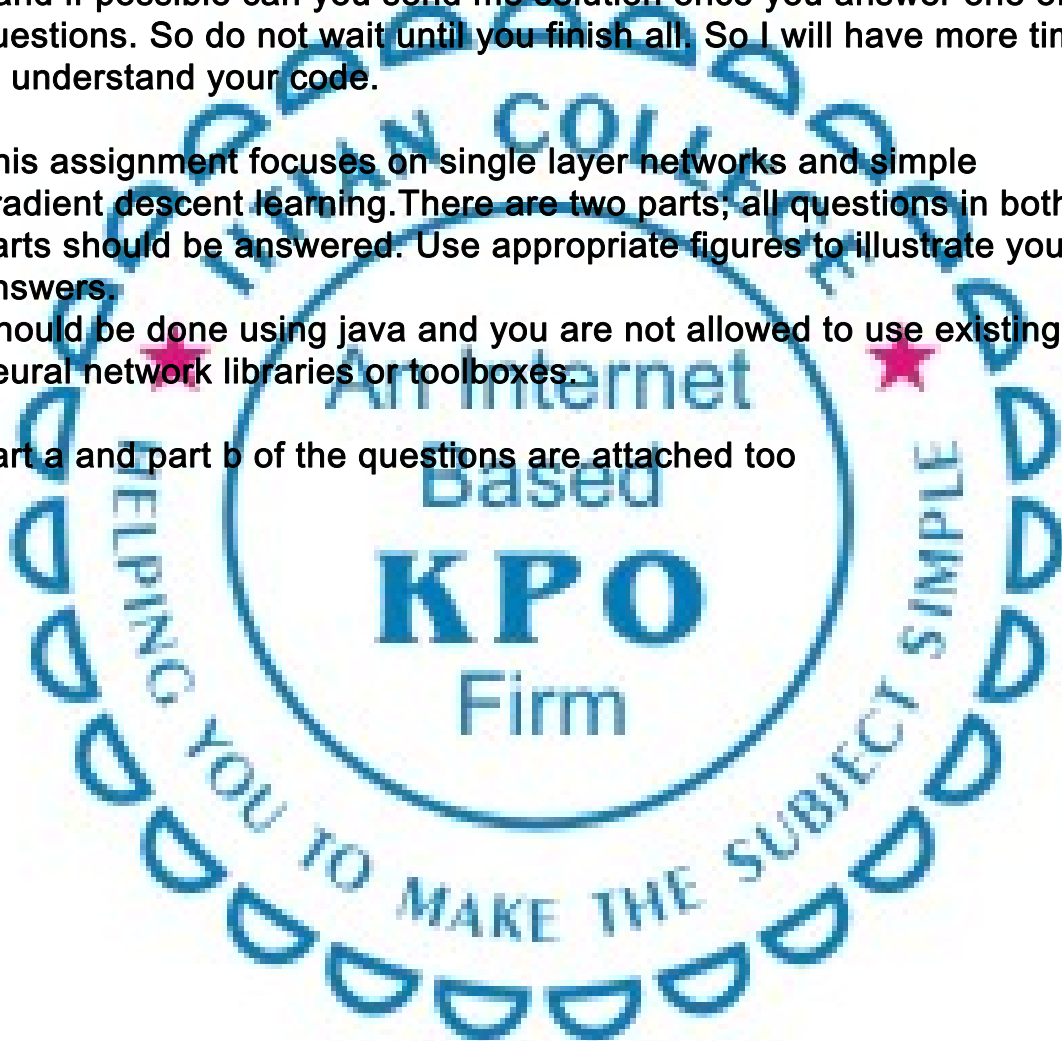


- it will be in java
- you will not use any neural network libraries or tools
- it will be your own code, if you use any other code you will reference it
- you will answer everything not just coding for each question and part.
- you will submit solution within 20 hours after hiring
- and if possible can you send me solution once you answer one of questions. So do not wait until you finish all. So I will have more time to understand your code.

This assignment focuses on single layer networks and simple gradient descent learning. There are two parts; all questions in both parts should be answered. Use appropriate figures to illustrate your answers.

Should be done using java and you are not allowed to use existing neural network libraries or toolboxes.

part a and part b of the questions are attached too



Part B: Using single layer networks for regression

(1) Generate data, using inputs (x-values) between 1 and 200 in increments of 2, from the following linear function with small random fluctuations: $y = 0.4^*x + 3 + \Delta$, where Δ is a small random value (different for each data point) drawn from a uniform distribution between -10 and +10. Using these data points and target values, train a single linear neuron. Comment on the weight and bias values. How do they relate to the original function? What happens to the training error, and the weight and bias values, if the size of the random fluctuations increases? Modify the original function and train a new network: what happens to the weight and bias values now? On the basis of your results, comment on what the network is actually doing and demonstrate this.

(2) Adopting a similar approach, generate data with two input variables (each between 1 and 200 in increments of 20, using each combination of values of the two input variables giving 100 inputs in total) using the following function $y = 0.4^*x1 + 1.4^*x2 + \Delta$ where Δ is a small random value (different for each data point) drawn from a uniform distribution now between -100 and +100. Again, train the network, and comment on the weight and bias values, saying how they relate to the original function. Once again, modify the original function, and train a new network; what happens to the weight and bias values now? What is this multiple-input network actually doing?

Finally, sketch a design for a network which not only has multiple inputs but also has multiple outputs, and suggest what such a network might be used for. **NB**: you do not need to program anything for this particular question only.

Part A: Using single layer networks for classification

(1) Initialise a Perceptron with zero weights. Use the input patterns $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$ as training data for a 2-class problem and assign 2 patterns to each class (use targets $+1$, -1). Illustrate the training procedure for one set of patterns. How many of the combinations of target values of patterns can the Perceptron learn? How many iterations does it take?

(2) Using target values $+1$ and -1 , generate 20 two-dimensional training data from the following Gaussian distributions, ensuring that the data is linearly separable (a way to ensure this can be through visualisation). The two classes of data are: class C1: $\mu = (0, 0)$, $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$; class C2: $\mu = (4, 0)$, $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. Here, the variables μ , Σ represent the centre and the covariance matrix of the Gaussian distribution, respectively. Train a single layer Perceptron to discriminate these two classes of data by using the available 20 training examples. Describe the detail of your training procedure. What happens if you use a sigmoidal activation function $g = \tanh(a)$? Also, generate some non-linearly separable Gaussian data (for example, change the mean μ of C2 to $(2, 0)$). What happens to the decision boundary when training with this data?