Task 1

Code completed in nn_keras.py with slight alterations to uci_data.py. Multiple runs for the requested tests are shown.

- Training and testing on pendigits dataset, with 2 layers, 10 training rounds.
    - classification accuracy=0.8002
    - classification accuracy=0.7919
    - classification accuracy=0.7936
    - classification accuracy=0.7902
    - classification accuracy=0.7922
- Training and testing on pendigits dataset, with 4 layers, 40 units per hidden layer, 20 training rounds, sigmoid activation for the hidden layers.
    - classification accuracy=0.8805
    - classification accuracy=0.8799
    - classification accuracy=0.8779
    - classification accuracy=0.8851
    - classification accuracy=0.8782

Task 2

$z = h(b+w^T x)$

$h = \sigma(a) = \frac{1}{1+e^{-ka}}$ or $h = \{0,\ a<0 \mid 1,\ a>=0\}$

w = 1 for all

b = -2

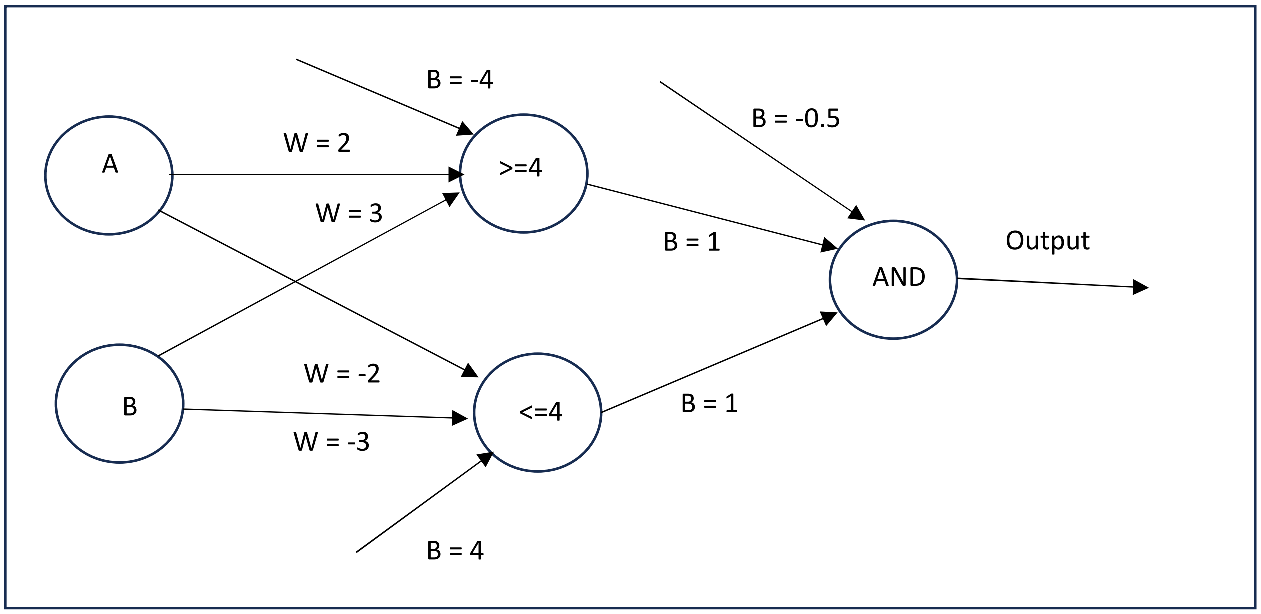
To make a Boolean activation function you can use either a step function or a sigmoid function with k = ∞. For this question a step function would work better due to a=0 returning 1 for the above definition.

For simplicity, the weight of all inputs can be set to 1.

Because we want to return 1 if two or more inputs are 1, b should be set to -2 to offset the inputs. This way if two inputs are 1 then 1 + 1 – 2 = 0 which returns 1 in the defined step function above.

Task 3

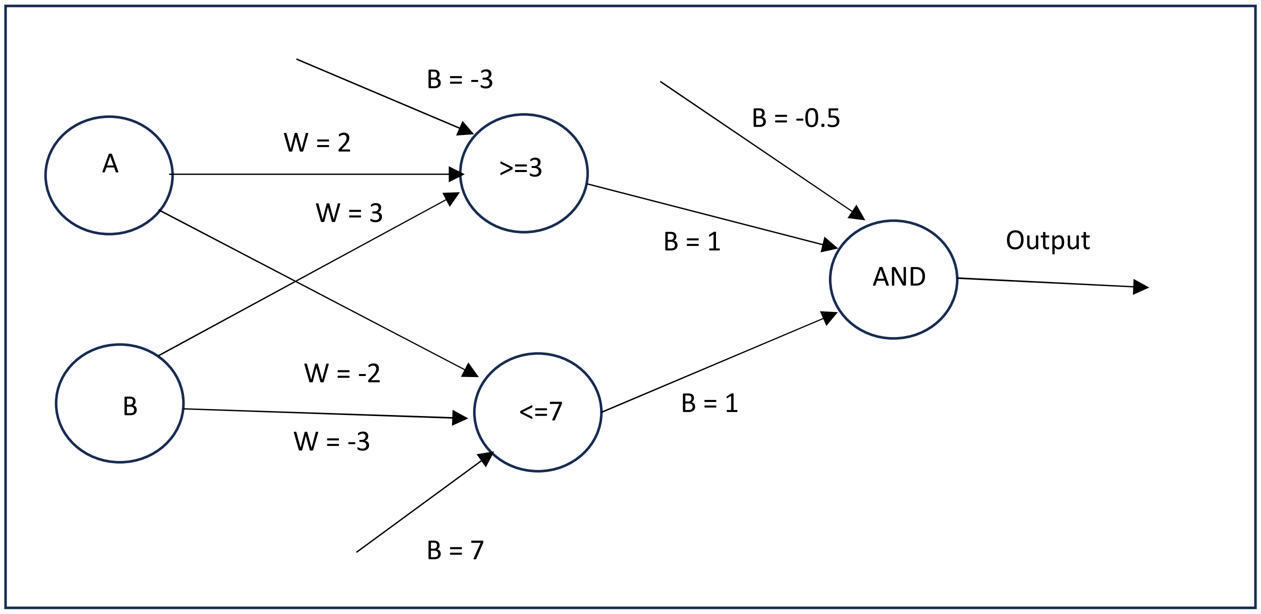
General idea: if 2A+3B >= 4 and 2A+3B <= 4 then 2A+3B = 4



Task 4

Same idea as before but change the values of B.

Yes it is possible.



Task 5

I would expect the classification accuracy to be lower if all the weights are initialized to zero. If all the weights of a perceptron are initially the same, the change of each weight due to training is completely dependent on the initial b. This makes the training focused towards a specific set of weights. This might cause the training to always find a relatively high local minima. By randomizing the weights initially, you reduce the bias towards a single solution with the possibility to find better solutions. By only using weights of zero, you are reducing the possibility that the training can find the best (minima) set of weights. You can always train a model different times and try to get better final weights if the initial weights are random.