COMP307 – Assignment 2
greenthom – 300536064

Question 1.1
The tables below give the prior distribution P(X), and two conditional distributions P(Y|X) and P(Z|Y). It is also known that Z and X are conditionally independent given Y. All three variables (X, Y, and Z) are binary variables.

X	$P(X)$
0	0.35
1	0.65

X	Y	$P(Y\|X)$
0	0	0.10
0	1	0.90
1	0	0.60
1	1	0.40

Y	Z	$P(Z\|Y)$
0	0	0.70
0	1	0.30
1	0	0.20
1	1	0.80

1. **Compute the table of the joint distribution P(X, Y, Z). Show the rule(s) you used and the steps for calculating each joint probability**

To compute the joint distribution P(X, Y, Z) -> use Product Rule:
- P(X, Y, Z) = P(Z | Y,X) * P(Y|X) * P(X)

Given that Z and X are independent given Y, we can rewrite the equation as:
- P(X, Y, Z) = P(Z | Y) * P(Y | X) * P(X)

X	Y	Z	P(Y|X)	P(Z|Y)	P(X)	P(X,Y,Z)	
0	0	0	0.1	0.7	0.35	P(Z=0|Y=0) * P(Y=0|X=0) * P(X=0)	0.1 * 0.7 * 0.35 = 0.0245
0	0	1	0.1	0.3	0.35	P(Z=1|Y=0) * P(Y=0|X=0) * P(X=0)	0.1 * 0.3 * 0.35 = 0.0105
0	1	0	0.9	0.2	0.35	P(Z=0|Y=1) * P(Y=1|X=0) * P(X=0)	0.9 * 0.2 * 0.35 = 0.063
0	1	1	0.9	0.8	0.35	P(Z=1|Y=1) * P(Y=1|X=0) * P(X=0)	0.9 * 0.8 * 0.35 = 0.252
1	0	0	0.6	0.7	0.65	P(Z=0|Y=0) * P(Y=0|X=1) * P(X=1)	0.6 * 0.7 * 0.65 = 0.273
1	0	1	0.6	0.3	0.65	P(Z=1|Y=0) * P(Y=0|X=1) * P(X=1)	0.6 * 0.3 * 0.65 = 0.117
1	1	0	0.4	0.2	0.65	P(Z=0|Y=1) * P(Y=1|X=1) * P(X=1)	0.4 * 0.2 * 0.65 = 0.052
1	1	1	0.4	0.8	0.65	P(Z=1|Y=1) * P(Y=1|X=1) * P(X=1)	0.4 * 0.8 * 0.65 = 0.208

2. **Create the full joint probability table of X and Y, i.e. the table containing the following four joint probabilities P(X=0, Y=0), P(X=0, Y=1), P(X=1, Y=0), P(X=1, Y=1). Show the rule(s) used, and the steps of calculating each joint probability.**

Independence shows us we can do P(X) * P(X|Y) to find P(X,Y) -> joint probability rule

P(X)	P(X|Y)	P(X,Y)
P(X=0) = 0.35	P(X=0|Y=0) = 0.1	0.35 * 0.1 = 0.035
P(X=0) = 0.35	P(X=0|Y=1) = 0.9	0.35 * 0.9 = 0.315
P(X=1) = 0.65	P(X=1|Y=0) = 0.6	0.65 * 0.6 = 0.390
P(X=1) = 0.65	P(X=1|Y=1) = 0.4	0.65 * 0.4 = 0.260

3. **From the above joint probability table of X, Y, and Z, calculate the following probabilities. Show your derivation.**

a. **P(Z=0)**

Probabilities for combinations of X and Y for Z=0:
P(Z=0)
= P(X=0, Y=0, Z=0) + P(X=0, Y=1, Z=0) + P(X=1, Y=0, Z=0) + P(X=1, Y=1, Z=0)
= 0.0245 + 0.063 + 0.273 + 0.052
= 0.4125

b. **P(X=0, Z=0)**

Probabilities for combination Y for X=0 and Z=0:
P(X=0,Z=0) =
= P(X=0, Y=0, Z=0) + P(X=0, Y=1, Z=0)
= 0.0245 + 0.063
= 0.0875

c. **P(X=1, Y=0|Z=1)**

To calculate we need to calculate p(Z=1) and p(Z=1 | X=1, Y=0)

P(Z=1) =
= P(X=0, Y=0, Z=1) + P(X=0, Y=1, Z=1) + P(X=1, Y=0, Z=1) + P(X=1, Y=1, Z=1)
= 0.0105 + 0.252 + 0.117 + 0.208
= 0.5875

Conditional probability rule = P(A|B) = P(A, B)/P(B)
P(Z=1 | X=1, Y=0) =
= (P(X=1, Y=0, Z=1))/P(X=1, Y=0)
= 0.117 / 0.39

= 0.3

Bayes Theorem: P(A|B) = P(B|A) * P(A) / P(B)
(product rule)
P(X=1, Y=0 | Z=1)
= (P(Z=1 | X=1, Y=0) * P(X=1, Y=0)) / P(Z=1)
= (0.3 * 0.39) / 0.5875
= 0.19914

d. P(X=0|Y=0, Z=0)

Bayes Theorem: P(A|B) = P(B|A) * P(A) / P(B)
(product rule)
Conditional probability rule = P(A|B) = P(A, B)/P(B)

P(Y=0, Z=0) =
= P(X=0, Y=0, Z=0) + P(X=1, Y=0, Z=0)
= 0.0245 + 0.273
= 0.2975

P(Y=0, Z=0 | X=0)
= P(X=0, Y=0, Z=0) / P(X=0)
= 0.0245 / 0.35
= 0.07

Put together
P(X=0 | Y=0, Z=0)
= (P(Y=0, Z=0 | X=0) * P(X=0)) / P(Y=0, Z=0)
= (0.07 * 0.35) / 0.2975
= 0.0823

1.2 Question 2
Consider three Boolean variables A, B, and C (can take t or f). We have the following probabilities:

- P(B = t) = 0.7
- P(C = t) = 0.4
- P(A = t | B = t) = 0.3
- P(A = t | C = t) = 0.5
- P(B = t | C = t) = 0.2

We also know that A and B are conditionally independent given C. Calculate the following probabilities. Show your derivation.

1. **P(B = t, C = t)**

Multiplication rule = P(A, B) = P(B|A) * P(A)
P(B = t, C = t)
= P(B = t | C = t) * P(C = t)
= 0.2 * 0.4
= 0.08

2. **P(A = f | B = t)**

A + b are conditionally independent given C
Probability of P(A = t | B = t) = 0.3
= P(A =f | B = t) -> 1 – p(A = t | C = t)
= 1 – 0.3
= 0.7

3. **P(A = t, B = t | C = t)**

Probability of P(A = t, B = t, C = t) would be defined between P(A = t | C = t) multiplied by P(B=t | C = t) -> conditional independence
Multiplication rule = P(A, B) = P(B|A) * P(A)

= P(A = t, B = t, C = t)
= P(A = t | C = t) * P(B = t | C = t) -> multiplication rule
= 0.5 * 0.2
= 0.1

4. **P(A = t | B = t, C = t)**

A and B are conditionally independent given C
Calculate P(A = t, B = t | C = t)
P(A = t, B = t | C = t)
= P(A = t | C = t) * P(B = t | C = t)
= 0.5 * 0.2
= 0.1

Bayes Theorem: P(A|B) = P(B|A) * P(A) / P(B)
Calculate probability of P(A = t | B = t, C = t) using P(A = t, B =t | C = t) / P(B = t | C = t)
P(A = t | B = t, C = t)
= 0.1 / 0.2
= 0.5

5. P(A = t, B = t, C = t)

Calculate P(A = t, B = t, C = t) by P(A = t | C = t) * P(B = t | C = t) * P (C = t)
P(A = t, B = t, C = t)

= P(A = t | C = t) * P(B = t | C = t) * P (C = t)

= 0.5 * 0.2 * 0.4
= 0.04

1.3 Report

1. **Report on the accuracy of your perceptron. For example, did it find a correct set of weights? Did its performance change much between different runs?**

My implementation of perception did not achieve 100% accuracy and optimal weights. Accuracies of my implementation are generally around 92-94%. With all weights set to 0 ('self.weights = np.zeros_like(features[0]) #weights with zeros') I am getting an accuracy of 92.87% (0.9287). With my weights initialized with random values between -1 and 1 ('self.weights = np.random.uniform(-1, 1, num_features)') I am getting returned accuracies between 92 – 94%. I extended the weights to a wider range however it did not improve results with it providing lower accuracies.

WEIGHTS AT 0

```
tomgreen@Toms-MacBook-Pro perceptron % python3 perceptron.py ionosphere.data
Total amount of iterations performed:  430
Highest accuracy:  0.9287749287749287
Final weights:  [-0.73       0.66       0.         0.0549684 -0.0206025  0.1015194
  0.1050279 -0.0050213  0.1279431  0.1629334  0.0388346 -0.1172435
 -0.0482315 -0.015789   0.0029592  0.1264974 -0.112544   0.0470896
  0.0546583 -0.1728169  0.0236575  0.0140395 -0.1167607  0.1022765
  0.0708433  0.0465803  0.0013153 -0.1442942  0.0148562  0.0510507
  0.0665586  0.0455307  0.0351777 -0.0133719 -0.1187362]
```

RANDOMISED WEIGHTS

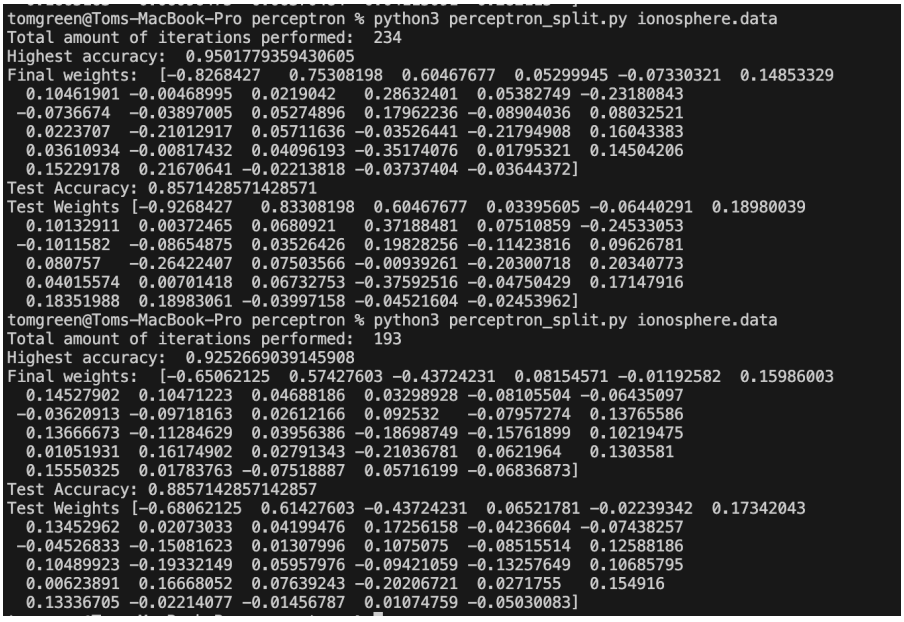
```
tomgreen@Toms-MacBook-Pro perceptron % python3 perceptron.py ionosphere.data
Total amount of iterations performed:  147
Highest accuracy:  0.9401709401709402
Final weights:  [-0.80503282  0.61082691  0.58810129  0.06511234  0.09703289  0.35193566
  0.32055298 -0.15714128  0.18591559  0.40932082  0.00955643 -0.23798602
 -0.01334441 -0.26207027  0.08875311  0.35811742 -0.21389894  0.12068975
  0.1419377  -0.30344412 -0.07942859 -0.03915472 -0.19016264  0.2043049
  0.07616394  0.11755133  0.15278649 -0.39924823 -0.13174602  0.20712719
  0.20428107  0.03055584 -0.01828432  0.085855   -0.12689994]
tomgreen@Toms-MacBook-Pro perceptron % python3 perceptron.py ionosphere.data
Total amount of iterations performed:  196
Highest accuracy:  0.9259259259259259
Final weights:  [-0.70564923  0.61067842  0.60298032  0.06076395 -0.02963604  0.100838
  0.11982184 -0.02673552  0.13931137  0.16166321  0.0396755  -0.11291451
 -0.05766299 -0.02157688 -0.01047871  0.12514861 -0.10684294  0.0535856
  0.06159696 -0.16393285  0.04329003  0.00423395 -0.11343959  0.10652417
  0.08484713  0.02977523  0.00778049 -0.14949176  0.02565286  0.05316962
  0.08354262  0.0661329   0.02447472  0.01074884 -0.09619679]
tomgreen@Toms-MacBook-Pro perceptron %
```

2. **Explain why evaluating the perceptron's performance on the training data is not a good measure of its effectiveness. You should split the dataset, retrain, and perform a fairer evaluation**

Assessing the perceptron's effectiveness based on training data accuracy is insufficient due to overfitting and bias as it may memorize examples rather than learn underlying patterns (overfitting). This results in high performance on training data but poor performance on new, unseen data as well as using same data for both training and test can introduce bias where the model becomes skewed towards specific characteristics of the training data failing to evaluate an unseen dataset.

The main objective of the model is to generalize effectively to unseen data so evaluating my perceptron's performance on a separate test dataset showed how it handles itself predicting new instances. My perceptron_bias.py implementation selects one-fifth of the data for testing created by a random split, where it then removes these selected rows from the original dataset to create a distinct test set. It leaves the remaining data for training.

My implementation provided me with test accuracies varying between 79 to 94 and train accuracies varying from 92-97% with the weights being randomized between -1 to 1. A reason why its performance and consistency of accuracies might not be ideal could be due to issues like overfitting since there may be less data for actual training if a split was not performed and it is now being used for testing.



```
tomgreen@Toms-MacBook-Pro perceptron % python3 perceptron_split.py ionosphere.data
Total amount of iterations performed:  234
Highest accuracy:  0.9501779359430605
Final weights:  [-0.8268427   0.75308198  0.60467677  0.05299945 -0.07330321  0.14853329
  0.10461901 -0.00468995  0.0219042   0.28632401  0.05382749 -0.23180843
 -0.0736674  -0.03897005  0.05274896  0.17962236 -0.08904036  0.08032521
  0.0223707  -0.21012917  0.05711636 -0.03526441 -0.21794908  0.16043383
  0.03610934 -0.00817432  0.04096193 -0.35174076  0.01795321  0.14504206
  0.15229178  0.21670641 -0.02213818 -0.03737404 -0.03644372]
Test Accuracy: 0.8571428571428571
Test Weights [-0.9268427   0.83308198  0.60467677  0.03395605 -0.06440291  0.18980039
  0.10132911  0.00372465  0.0680921   0.37188481  0.07510859 -0.24533053
 -0.1011582  -0.08654875  0.03526426  0.19828256 -0.11423816  0.09626781
  0.080757   -0.26422407  0.07503566 -0.00939261 -0.20300718  0.20340773
  0.04015574  0.00701418  0.06732753 -0.37592516 -0.04750429  0.17147916
  0.18351988  0.18983061 -0.03997158 -0.04521604 -0.02453962]
tomgreen@Toms-MacBook-Pro perceptron % python3 perceptron_split.py ionosphere.data
Total amount of iterations performed:  193
Highest accuracy:  0.9252669039145908
Final weights:  [-0.65062125  0.57427603 -0.43724231  0.08154571 -0.01192582  0.15986003
  0.14527902  0.10471223  0.04688186  0.03298928 -0.08105504 -0.06435097
 -0.03620913 -0.09718163  0.02612166  0.092532   -0.07957274  0.13765586
  0.13666673 -0.11284629  0.03956386 -0.18698749 -0.15761899  0.10219475
  0.01051931  0.16174902  0.02791343 -0.21036781  0.0621964   0.1303581
  0.15550325  0.01783763 -0.07518887  0.05716199 -0.06836873]
Test Accuracy: 0.8857142857142857
Test Weights [-0.68062125  0.61427603 -0.43724231  0.06521781 -0.02239342  0.17342043
  0.13452962  0.02073033  0.04199476  0.17256158 -0.04236604 -0.07438257
 -0.04526833 -0.15081623  0.01307996  0.1075075  -0.08515514  0.12588186
  0.10489923 -0.19332149  0.05957976 -0.09421059 -0.13257649  0.10685795
  0.00623891  0.16668052  0.07639243 -0.20206721  0.0271755   0.154916
  0.13336705 -0.02214077 -0.01456787  0.01074759 -0.05030083]
```

3.2.1. Report the output and predicted class of the first instance in the dataset using the provided weights.

First instance predicted class: Chinstrap

```
tomgreen@Toms-MacBook-Pro neural % python3 a2Part1.py
First instance has label Adelie, which is [0] as an integer, and [1. 0. 0.] as a list of outputs.

Predicted label for the first instance is: ['Chinstrap']
```

3.2.2. **Report the updated weights of the network after applying a single back-propagation based on only the first instance of the dataset**

```
Weights after performing BP for first instance only:
Hidden layer weights:
 [[-0.26293231 -0.38229496]
 [ 0.18978343  0.4868215 ]
 [-0.17585303  0.15099507]
 [ 0.23102589 -0.11557548]]
Output layer weights:
 [[-0.97267967 -0.50843807 -0.11240592]
 [-0.66959968 -0.6057472  -0.65137609]]
```

3.2.3. Report the final weights and accuracy on the test set after 100 epochs. Analyze the test accuracy and discuss your thoughts.

```
After training:
Hidden layer weights:
 [[ 0.9386876  -9.84076342]
 [-7.30590284  5.21616974]
 [ 2.3914977  -1.4072903 ]
 [ 2.47644876  1.44156165]]
Output layer weights:
 [[ -9.71047409  -2.44670946   3.24345684]
 [  4.91368601  -2.87797267 -11.68759154]]
Accuracy test set:  81.53846153846153 %
```

After 100 epochs my test accuracy is 81.5384% showing that the neural network was successful in classifying most of the instances. From here we can improve on this by introducing bias nodes into the code as it would help shift the activation function allowing a higher accuracy.

3.2.4. **Discuss how your network performed compared to what you expected. Did it converge quickly? Do you think it is overfitting, underfitting, or neither?**

```
epoch =  0
epoch acc =  0.48880597014925375 %
epoch =  1
epoch acc =  0.5 %
epoch =  2
epoch acc =  0.5783582089552238 %
epoch =  3
epoch acc =  0.664179104477612 %
epoch =  4
epoch acc =  0.75 %
epoch =  5
epoch acc =  0.7873134328358209 %
epoch =  6
epoch acc =  0.7947761194029851 %
epoch =  7
epoch acc =  0.7947761194029851 %
epoch =  8
epoch acc =  0.7947761194029851 %
epoch =  9
```

Significant improvements were made in the starting six epochs, transitioning from 0.4888% to 0.78%. Subsequent epochs displayed less dramatic enhancements probably due to the initial weight adjustments targeting large errors with later adjustments focusing more on maintaining the current weights which is a common process/trait within neural network training. It finished on an accuracy of 0.82835 at the final epoch. Further training will not benefit the model as it seems improved to its optimal performance level. There is no overfitting or underfitting issues suggested by the test set accuracy of 81.538 and final epoch of 0.82835 which shows that our model is good at generalizing to unseen data.

We are only testing four features in the data so we can improve our model if there were more features as there would be more to learn about each type. This would result in deeper understanding/identification for the model to understand trends in the class data to achieve a higher accuracy.

3.3.1. Report the output and predicted class of the first instance in the dataset using the provided weights. Using bias nodes

```
tomgreen@Toms-MacBook-Pro neural % python3 a2Part1_biases.py
First instance has label Adelie, which is [0] as an integer, and [1. 0. 0.] as a list of outputs.

Predicted label for the first instance is: ['Chinstrap']
```

3.3.2. Report the updated weights of the network after applying a single back-propagation based on only the first instance in the dataset. Using bias nodes:

```
Weights after performing BP for first instance only:
Hidden layer weights:
 [[-0.27424401 -0.30975817]
 [ 0.07886907  0.35534647]
 [-0.21717106  0.19378445]
 [ 0.18085654 -0.09354912]]
Output layer weights:
 [[-0.686051   -0.05057703 -0.08993949]
 [-0.3191158  -0.07758915 -0.59407451]]
```

3.3.3. Train it using the same parameters as before, and report your test accuracy. Compare the accuracy achieved to that of the original network and discuss possible reasons for any performance differences.

```
After training:
Hidden layer weights:
 [[ -1.41778542 -11.19788948]
 [ -6.20381026   5.37974232]
 [  4.21321351  -0.82347839]
 [  4.66569156   2.52785166]]
Output layer weights:
 [[-2.71994508 -7.05540073  7.57204528]
 [ 9.27217128 -8.21007394 -5.47554287]]
Accuracy test set:  100.0 %
```

Adding bias nodes enabled the neural network to obtain 100% test accuracy which improved on our original network as it allowed better fitting of the training data to identify class label patterns with the final epoch reaching up to 0.9925% which could potentially indicate overfitting. A way to test if overfitting is indeed occurring with biases added to nn is by using a larger test set but it seems that the model has improved its ability to recognize patterns on unseen data.

REFERENCES

- COMP307/420 VUW Lecture Neural Networks 2: Backpropagation (equations for forward and backward propagations)
- ICS 440 Lecture 26: Linear Classifiers (understanding of perceptron learning rule and update weights methods) https://courses.grainger.illinois.edu/cs440/fa2019/Lectures/lect26.html
- Youtube: Perceptron -> (understanding of how perceptron works in visual graph format) https://www.youtube.com/watch?v=4Gac5I64LM4