# 1 Reasoning Under Uncertainty Basics

**Q1.1**

1. Z and X are independent given Y = P(Z|X, Y) = P(Z|Y)

   By using chain rule of probabilities, the joint probability can be:

   - P(X,Y,Z) = P(Z|X,Y) x P(Y|X) x P(X)

   With conditional independence the formula simplifies to:

   - P(X,Y,Z) = P(Z|Y) x P(Y|X) x P(X)

   Given Distributions:

| 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 |

Calculating Joint distribution Table:

P(X = 0, Y = 0, Z = 0)= 0.70 x 0.10 x 0.35 = 0.0245

P(X = 0, Y= 0, Z= 1) = 0.30 x 0.10 x 0.35 = 0.0105

P(X = 0, Y = 1, Z = 0) = 0.20 x 0.90 x 0.35 = 0.063

P(X = 0, Y = 1, Z = 1) = 0.80 x 0.90 x 0.35 = 0.252

P(X = 1, Y = 0, Z = 0) = 0.70 x 0.60 x 0.65 = 0.273

P(X = 1, Y = 0, Z = 1) = 0.30 x 0.60 x 0.65 = 0.117

P(X = 1, Y = 1, Z = 0) = 0.20 x 0.40 x 0.65 = 0.052

P(X = 1, Y = 1, Z = 1) = 0.80 x 0.40 x 0.65 = 0.208

| X | Y | Z | P(X,Y,Z) |
|---|---|---|---|
| 0 | 0 | 0 | 0.0245 |
| 0 | 0 | 1 | 0.0105 |
| 0 | 1 | 0 | 0.063 |
| 0 | 1 | 1 | 0.252 |
| 1 | 0 | 0 | 0.273 |
| 1 | 0 | 1 | 0.117 |
| 1 | 1 | 0 | 0.052 |
| 1 | 1 | 1 | 0.258 |

2. To create the joint probability table of X and Y the following formula can be used:
    - P(X,Y) = P(X) x P(Y|X)

   This comes from the rule of conditional probability which says that the joint probability is the product of the conditional probability and the marginal probability of the event.

   Calculations:

   P(X = 0, Y = 0)
   P(Y = 0 | X = 0) = 0.10
   P(X = 0) = 0.35
   P(X = 0, |Y = 0) = 0.10 x 0.35 = 0.035

   P(X = 0, Y = 1)
   P(Y = 1 | X = 0) = 0.90
   P(X = 0) = 0.35
   P(X = 0, Y = 1) = 0.90 x 0.35 = 0.315

   P(X = 1, Y = 0)
   P(Y = 0 | X = 1) = 0.60
   P(X = 1) = 0.65
   P(X = 1 , Y = 0) = 0.60 x 0.65 = 0.39

P(X = 1, Y = 1)
P(Y = 1 | X = 1) = 0.40
P(X = 1) = 0.65
P(X = 1, Y = 1) = 0.40 x 0.65 = 0.26

| X | Y | P(X,Y) |
|---|---|---|
| 0 | 0 | 0.035 |
| 0 | 1 | 0.315 |
| 1 | 0 | 0.39 |
| 1 | 1 | 0.26 |

**3.** Going based off the joint probability table of X,Y,Z in Q1:

**a)** P(Z=0) = to the sum of all joint probabilities where 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) = sum of probabilities where 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) = Find P(X = 1, Y = 0, Z = 1) and normalize by P(Z = 1):

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

P(X = 1, Y = 0, Z = 1) = 0.117

P(X = 1, Y = 0|Z = 1) = P(X = 1, Y = 0, Z = 1)/P(Z = 1) = 0.117/0.5875 = 0.1991

**d)** P(X = 0|Y = 0, Z = 0) = determine P(X = 0, Y = 0, Z = 0) and normalize by P(Y=0,Z=0)

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(X = 0, Y = 0, Z = 0) = 0.0245)

P(X = 0|Y = 0, Z = 0) = P(X = 0, Y = 0, Z = 0)/P(Y = 0, Z = 0) = 0.0245/0.2975 = 0.0823

**Q1.2**

1. P(B = t, C = t) = P(B = t| C = t) x P(C = t) (Product Rule)
   = 0.2 x 0.4
   P(B = t, C = t) = 0.08

2. P(A = f|B = t) = 1 - P(A = t| B = t) (Complement rule)
   = 1 - 0.3
   P(A = f|B =t) = 0.7

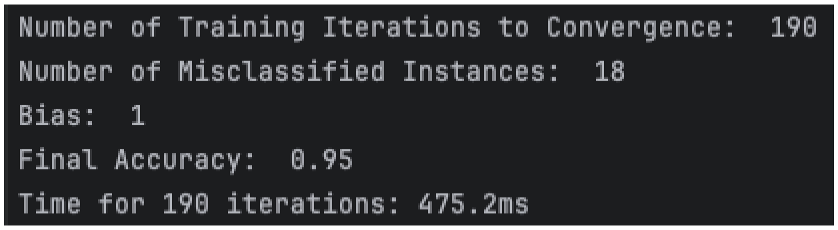
3. P(A = t, B = t|C = t) = P(A = t|C = t) x P(B = t| C = t) (Rule of conditional independence)
   = 0.5 x 0.2
   P(A = t, B= t|C = t) = 0.1

4. P(A = t|B = t, C = t) = P(A = t|C = t) Since A and B are conditionally independent given C.
   P(A = t|B = t, C = t) = 0.5

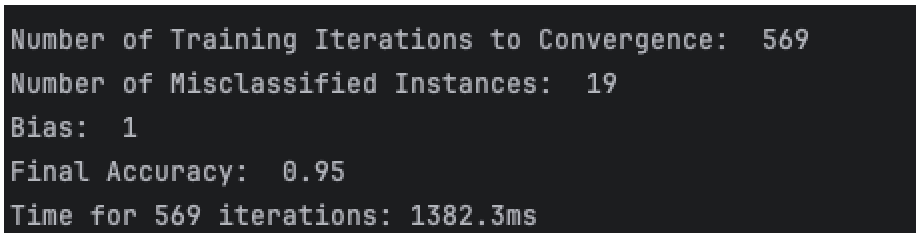
5. P(A = t,B = t, C = t) = P(A = t, B = t|C = t) x P(C) (Rule of conditional independence)
   = 0.1 x 0.4
   P(A = t,B = t, C = t) = 0.04

# 2 Perceptron

1. The Perceptron is reaching an accuracy of about 0.948 which I rounded to 2dp to get 0.95. The accuracy maxes out at the .948 range sometimes halfway through and sometimes after 1-200 epochs, and without the condition to stop the loop it will just stay around .90-.95 and go for the full 1000 iterations.

   ```
   Number of Training Iterations to Convergence:  190
   Number of Misclassified Instances:  18
   Bias:  1
   Final Accuracy:  0.95
   Time for 190 iterations: 475.2ms
   ```

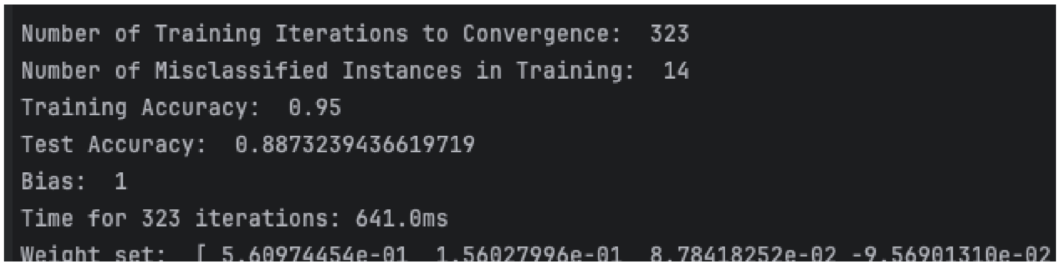
   

   ```
   Number of Training Iterations to Convergence:  569
   Number of Misclassified Instances:  19
   Bias:  1
   Final Accuracy:  0.95
   Time for 569 iterations: 1382.3ms
   ```

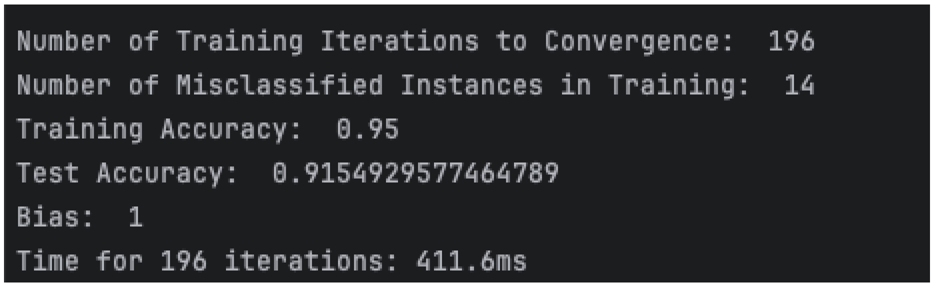
   I'm using these two runs as an example for performance variation between different runs. When Initializing my weights i use np.random.rand so the weights are randomly selected between 0 and 1 and change each run. This leads to variation in the number of iterations/epochs to reach .95 accuracy which I have chosen as the ("Or some limit").

2. Evaluating the perceptron's performance using only the training data isn't a good idea because it can lead to overfitting. This is when the model learns the training data too well, to the point where it becomes very effective on the training data but can fail on new data.

   Using a train/split test is a way to counteract the negative effects of using just training data. I made a copy of my original code and added a new function to split the data and also edited the other functions to use the train and test splits. This is the results after doing two runs using the test/train split:

   ```
   Number of Training Iterations to Convergence:  323
   Number of Misclassified Instances in Training:  14
   Training Accuracy:  0.95
   Test Accuracy:  0.8873239436619719
   Bias:  1
   Time for 323 iterations: 641.0ms
   Weight set:  [ 5.60974454e-01  1.56027996e-01  8.78418252e-02 -9.56901310e-02
   ```

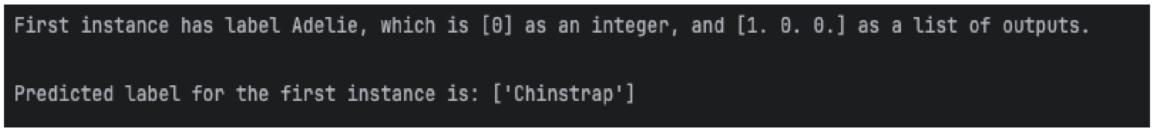


```
Number of Training Iterations to Convergence:  196
Number of Misclassified Instances in Training:  14
Training Accuracy:  0.95
Test Accuracy:  0.9154929577464789
Bias:  1
Time for 196 iterations: 411.6ms
```

This shows a high accuracy with the test accuracy being slightly lower than the training accuracy. Showing that the Perceptron isn't overfitting or underfitting. The way my convergence is calculated is not ideal as it is just using the point where the fit function reaches 0.95 accuracy. In reality the model converges quite quickly which I will work on for the next time, but the number of misclassified instances in training is staying constant at 14.

# 3 Classifying Penguins with NN

1. The output and predicted class of the first instance in the dataset using the provided weights is as follows:

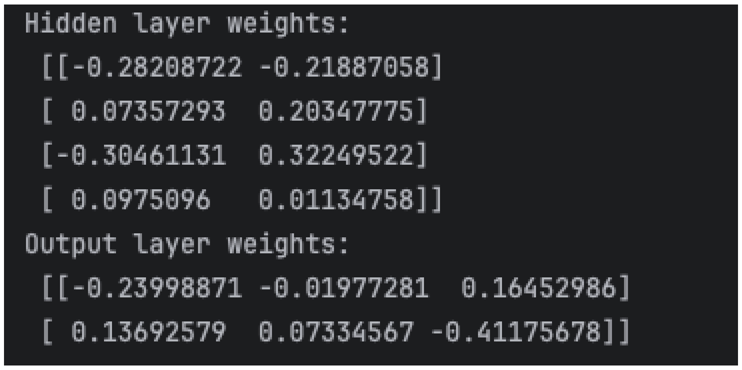


```
First instance has label Adelie, which is [0] as an integer, and [1. 0. 0.] as a list of outputs.

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

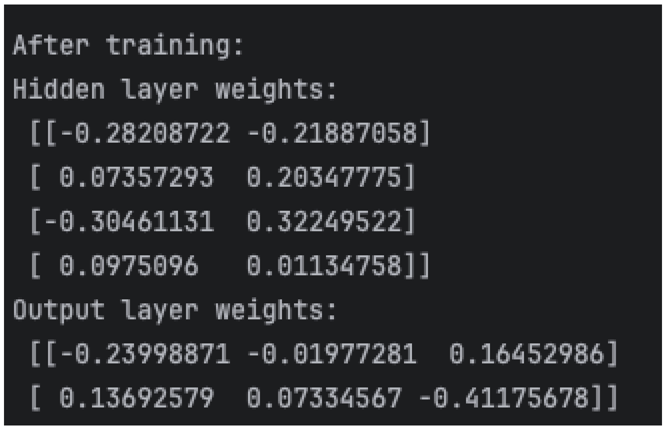
   This shows the first instance has the label Adelie, which [0] as an integer and [1. 0. 0.] as a list of outputs. The predicted class of the first instance is ‘Chinstrap’, which is a type of penguin.

2. After applying a single back propagation based on only the first instance in the dataset these are the outputs:
   Outputs before:



```
 Hidden layer weights:
  [[-0.28208722 -0.21887058]
  [ 0.07357293  0.20347775]
  [-0.30461131  0.32249522]
  [ 0.0975096   0.01134758]]
 Output layer weights:
  [[-0.23998871 -0.01977281  0.16452986]
  [ 0.13692579  0.07334567 -0.41175678]]
```

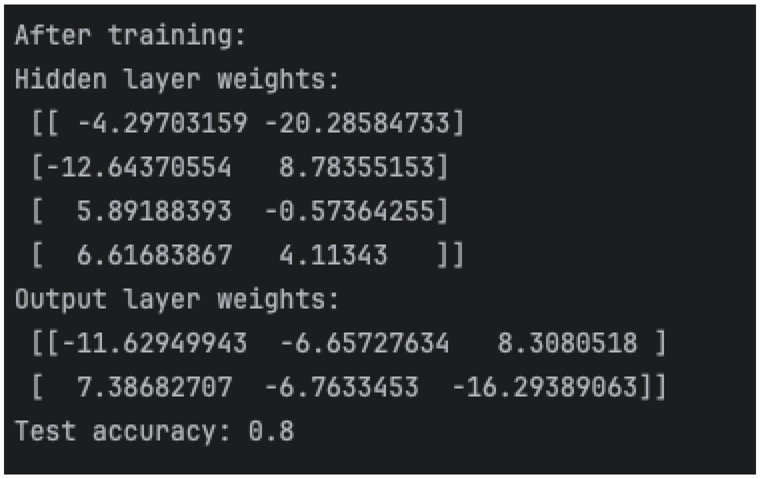
   After



```
After training:
Hidden layer weights:
 [[-0.28208722 -0.21887058]
 [ 0.07357293  0.20347775]
 [-0.30461131  0.32249522]
 [ 0.0975096   0.01134758]]
Output layer weights:
 [[-0.23998871 -0.01977281  0.16452986]
 [ 0.13692579  0.07334567 -0.41175678]]
```

   As you can see there was no change after the single back propagation.

3. The finals weights after training look like this:



```
After training:
Hidden layer weights:
 [[ -4.29703159 -20.28584733]
 [-12.64370554   8.78355153]
 [  5.89188393  -0.57364255]
 [  6.61683867   4.11343   ]]
Output layer weights:
 [[-11.62949943  -6.65727634   8.3080518 ]
 [  7.38682707  -6.7633453  -16.29389063]]
Test accuracy: 0.8
```

There is an accuracy of 0.8, meaning it correctly classified 80% of the instances in the test set. This shows that the model has learned patterns from the training data and can make relatively accurate predictions on unseen data.

Some improvements could be to fine tune the learning rate, epochs, and other parameters to improve the overall performance.

Adding more layers to the model could also increase the ability to learn the pattern but should be done carefully to not cause over fitting.

Overall I think the accuracy of 0.8 without bias nodes shows that the model has learned and been able to make predictions on unseen data.

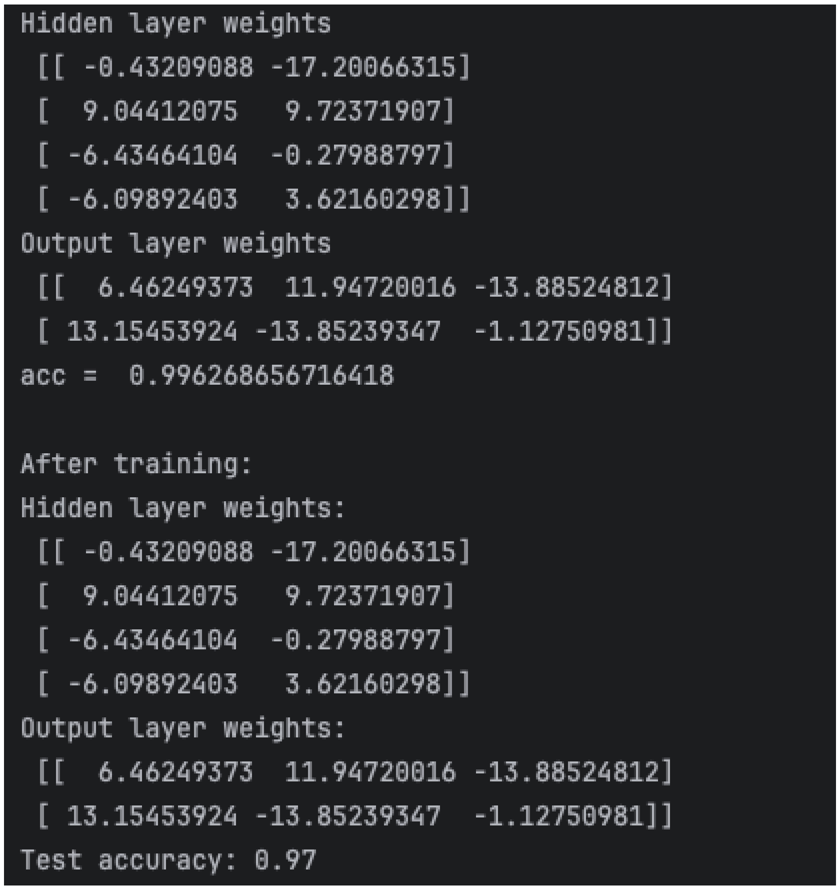
4. The network performed reasonably well with an accuracy of 0.8 at the end of only 100 epochs of training. These are my observations:

   The network looks to converge relatively quickly in the first few epochs. The weights start changing a lot from their starting values in the early epochs. The training accuracy reaches the maximum value of about 0.8 very early on. After that it stays similar with small changes, which shows that it has converged and can’t make any further progress.

   Since the training and test accuracies are equal, it is unlikely that there is any severe over or underfitting because:
   - Overfitting normally has training accuracy > test accuracy
   - Underfitting normally has low training and test accuracy

   Overall the network seems to have predicted the classes and does so while avoiding Severe overfitting or underfitting issues. But it has reached a point where the current Process for configuration/optimization cannot improve.

5. In this part I have made a copy of the program and edited the code to use bias nodes using the initial weights which are shown in Table 2. Using the same training parameters as before this is my new test accuracy:



```
Hidden layer weights
 [[ -0.43209088 -17.20066315]
 [  9.04412075   9.72371907]
 [ -6.43464104  -0.27988797]
 [ -6.09892403   3.62160298]]
Output layer weights
 [[  6.46249373  11.94720016 -13.88524812]
 [ 13.15453924 -13.85239347  -1.12750981]]
acc =  0.996268656716418

After training:
Hidden layer weights:
 [[ -0.43209088 -17.20066315]
 [  9.04412075   9.72371907]
 [ -6.43464104  -0.27988797]
 [ -6.09892403   3.62160298]]
Output layer weights:
 [[  6.46249373  11.94720016 -13.88524812]
 [ 13.15453924 -13.85239347  -1.12750981]]
Test accuracy: 0.97
```

There is now a test accuray of 0.97. This is a significant improvement from the original network which only had an accuracy of 0.8. Also the training accuracy is 0.996, the difference between the test and train accuracies is something that seems more realistic compared to the previous one where they were both equal.

This shows that there still isn't any significant over or underfitting, as the training accuracy is not that much higher than the test, and they are both high accuracies showing they aren't underfitting.

It also takes a bit longer to reach convergence where I need about 50 epochs to reach the maximum accuracy, which is much better than before.

This reason for this is because bias nodes give the network extra flexibility to change the activation functions output along the axis. Which gives the network the ability to model a more complex relationship between input/outputs.