Question 1

a)

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
# Create column for particle type based on Y1, Y2, Y3
dat$Particle <- ifelse(dat$Y1 == 1, "alpha", ifelse(dat$Y2 == 1, "beta", "rho"))

# Create the scatter plot
ggplot(data = dat, aes(x = X1, y = X2, color = Particle)) +
  geom_point(shape = 16, size = 3) +
  scale_color_manual(values = c("alpha" = "red", "beta" = "lightblue", "rho" = "green"),labels = c("alp
  labs(x = "X1", y = "X2", title = "") +
  theme_minimal() +
  theme(legend.position = "right",axis.line = element_line(color = "black", linewidth = 1))  +
  coord_fixed()  # Ensures 1:1 aspect ratio
```

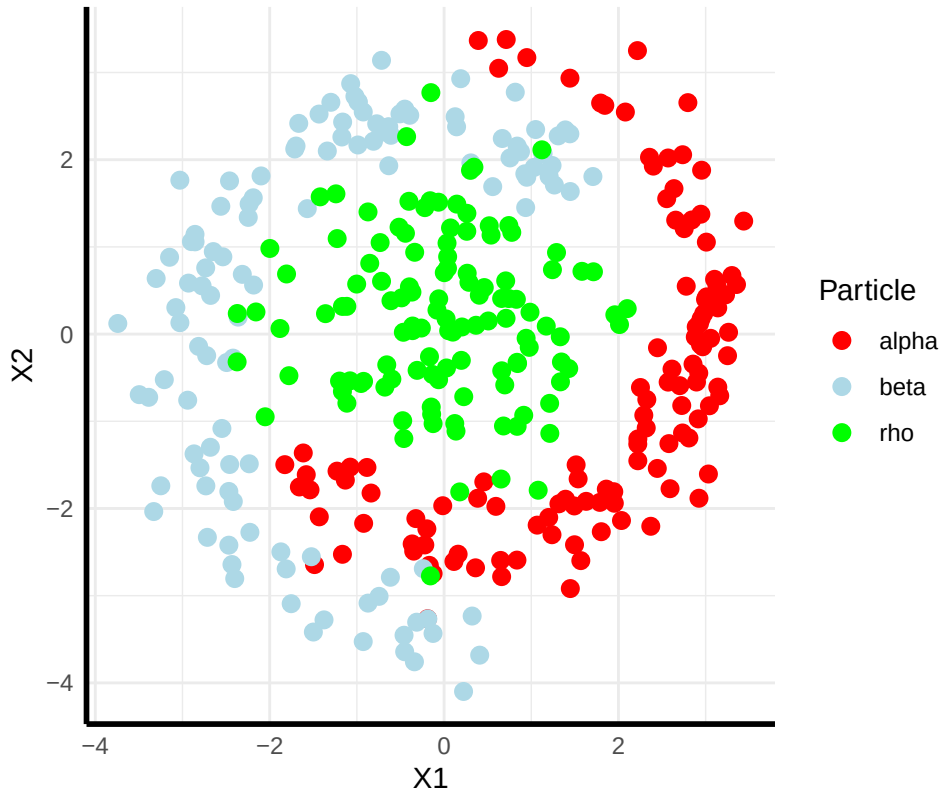

Figure 1: Scatter Plot of coordinates by Particle Type

The rho particles(green) form a circular region.The alpha particles(red) form a concave shape around the

rho and beta particles.The beta particles(blue) also form a concave shape around the rho and alpha particles.There are signs of overlap with all three particles

A neural network is an appropriate model class for this problem as the particles above are not linearly separable and neural networks are good at learning non-linear decision boundaries and capturing complex patterns in the data

b)

```r
soft_max <- function(Z) {
  exp_Z <- exp(Z)
  denom <- colSums(exp_Z)
  probs <- exp_Z / matrix(denom, nrow = nrow(Z), ncol = ncol(Z), byrow = TRUE)
  return(probs)
}
```

c)

```r
cross_ent <- function(y, y_hat) {
  if (y[1] == 1) {
    return(-log(y_hat[1]))
  } else if (y[2] == 1) {
    return(-log(y_hat[2]))
  } else if (y[3] == 1) {
    return(-log(y_hat[3]))
  }
}
```

Since y_i is one-hot encoded,the terms where $y_{ij} = 0$ contribute 0 to C_i.Therefore, it is computationally more efficient to evaluate only the terms corresponding to $y_{i\cdot} = 1$ as computing terms where $y_{i\cdot} = 0$ is computationally a waste of time and resources.

d)

```r
g <- function(Yhat, Y) {
  #Yhat: 3 x N matrix of predicted probabilities
  #Y: 3 x N matrix of one-hot encoded responses

  log_Yhat <- -log(Yhat)

  C_i <- colSums(Y * log_Yhat)

  return(mean(C_i))
}
```

e)

For an (m, m)-AFnetwork with dim(x)=p , dim(y) = q ,

- Augmented features: original p inputs are transformed into p augmented features via a separate hidden layer of size p .
- $2p$ input nodes =(Original p nodes + Augmented features p).
- m nodes in the first hidden layer.
- m nodes in the second hidden layer.
- q ouput nodes.
- Augmentation Layer: transforms original p inputs into p augmented features.

p^2 Weights + p Bias terms = $p^2 + p$ parameters.

- Input to First Hidden Layer:

$2pm$ Weights+ m Bias terms = $2pm + m$ parameters.

- First to Second Hidden Layer:

m^2 Weights + m Bias Terms = $m^2 + m$ paramaters.

- Second Hidden to Output Layer:
  mq weights + q biases = $mq + q$ parameters.
- Total parameters = $p^2 + p + 2pm + m^2 + 2m + mq + q$.

i)(Nesan)

AFnetworks introduce a dedicated hidden layer that learns nonlinear transformations of the input features before they are fed into the main network. These transformations act like automatic, learned features that might capture complex patterns the original inputs could not express on their own leading to better generalization and performance of the model.

Addtionally, as AFnetworks explicitly separate the original inputs from augmented inputs. This allows us to understand what features matter and how they matter. Hence, helps us understand how the network is reshaping the data before making predictions.