University of Cape Town

Department of Statistical Sciences

Analytics

Assignment 2

Author:	*Student Number:*
Christopher Eason	ESNCHR001
Phillip Liu	LXXZHA003

April 29, 2025

Contents

1 Question a

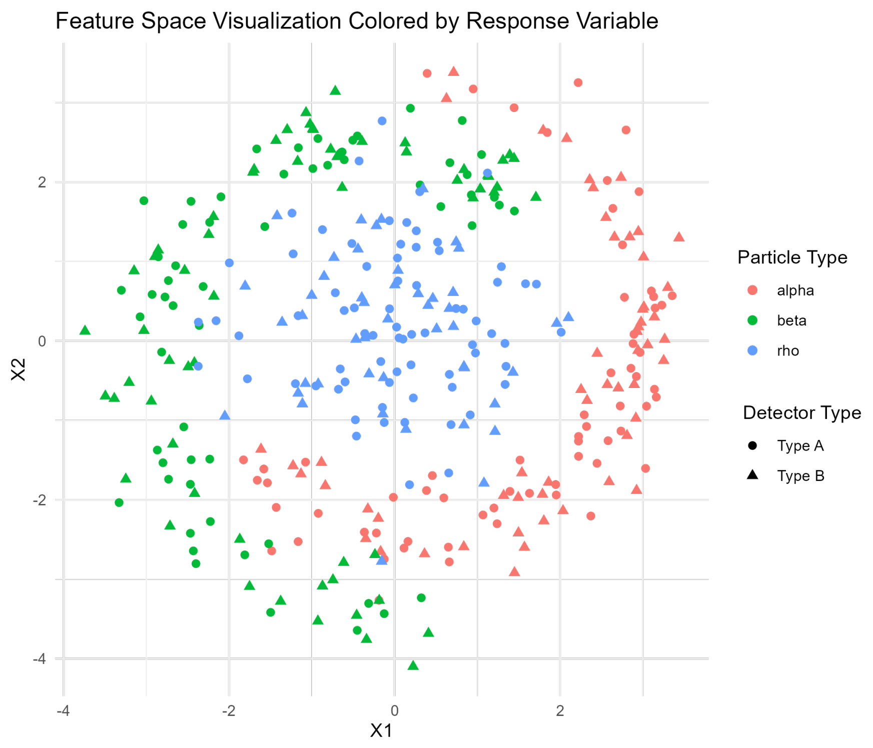

Figure 1: Plot the Coordinates of Each Particle, and Colouring Coding Based on Particle Type

A neural network is an appropriate model class for the present classification task due to the non-linear structure evident in the feature space. While a linear decision boundary might suffice to distinguish between alpha and beta particles, the classification of rho particles presents a greater challenge. As observed in the scatter plot, rho particles occupy a central region around which both alpha and beta particles curve, forming non-linearly separable clusters. This pattern suggests the presence of non-linear class boundaries. Traditional linear classifiers would struggle to capture this complexity, leading to suboptimal performance. Consequently, a model capable of capturing non-linear relationships—such as a neural network—is well-suited to learn the complex boundaries necessary for accurate particle classification.

2 Question b

```
softmax <- function(Z){
  Z_exp <- exp(Z)
  A_1 <- Z_exp * t(as.matrix(1/colSums(Z_exp), ncol = nrow(Z_exp), byrow = FALSE)
  %*% matrix(1,ncol = nrow(Z_exp)))

  return(t(A_1))
}
```

Listing 1: R implementation of the softmax activation function in matrix form, normalizing exponentiated logits across each row to produce class probabilities.

3 Question c

$$C_i = \begin{cases} -\log(\hat{y}_{i1}), & \text{if } (y_{i1}, y_{i2}, y_{i3}) = (1,0,0), \\ -\log(\hat{y}_{i2}), & \text{if } (y_{i1}, y_{i2}, y_{i3}) = (0,1,0), \\ -\log(\hat{y}_{i3}), & \text{if } (y_{i1}, y_{i2}, y_{i3}) = (0,0,1). \end{cases}$$

Numerical Advantages
Evaluating only the active term

$$C_i = -\log(\hat{y}_{ij}) \quad \text{for the unique } j \text{ with } y_{ij} = 1$$

has two main benefits:

- Numerical stability: By avoiding the computation of terms of the form $0 \times \log(\hat{y}_{ik})$ for $k \neq j$, we never attempt to evaluate $\log(0)$ or propagate log of extremely small probabilities, which can underflow to $-\infty$ or produce NaN values.
- Computational efficiency: We reduce the number of expensive log and multiplication operations from q (here $q = 3$) to exactly one per observation. This simplification is especially important in large-scale settings.

4 Question d

```
g <- function(Yhat, Y) {
  true_class_col <- max.col(Y)
  prob <- Yhat[1:nrow(Yhat),true_class_col]
  return(-mean(log(prob)))
}
```

Listing 2: R function to compute the cross-entropy loss for multi-class (polytomous) classification, using the predicted probabilities corresponding to the true class labels.

5 Question e

Let

- $p =$ be the number of input features ($\dim(\mathbf{x})$)
- $a =$ be the number of augmented features $= p$
- $m =$ be the number of nodes in each hidden layer
- $q =$ be the number of output classes ($\dim(\mathbf{y})$)

From the input layer to the augmented layer there are p input nodes and a nodes for the augmented features. Hence the parameters in this layer are:

- weights $= p \times a$
- biases $= a$

Total parameters for the augmented layer is $= p \times a + a$

From the augmented layer to the first hidden layer there are $p+a$ input nodes (including the augmented features) and m nodes in the first hidden layer. Hence the parameters in this layer are:

- weights $= (p + a) \times m$
- biases $= m$

Total parameters for the first hidden layer is $= (p + a) \times m + m$

From the first hidden layer to the second hidden layer there are m nodes in the first hidden layer and m nodes in the second hidden layer. Hence the parameters in this layer are:

- weights $= m \times m$
- biases $= m$

Total parameters for the second hidden layer is $= m^2 + m$

From the second hidden layer to the output layer there are m nodes in the second hidden layer and q nodes in the output layer. Hence the parameters in this layer are:

- weights $= m \times q$
- biases $= q$

Total parameters for the second hidden layer is $= m \times q + q$

Therefore the expression to calculate all of the total number of paramters in (m,m)-AFnetwork is:

$$\begin{aligned} & p \times a + a + (p + a) \times m + m + m^2 + m + m \times q + q \\ = & p^2 + p + 2pm + 2m + m^2 + mq + q \end{aligned}$$

6 Question f

```
# Specify activation functions for the hidden and output layers:
sig1 = function(z) {tanh(z)} #1/(1+exp(-z))
sig2 = function(z) {tanh(z)}
sig3 = function(z) {tanh(z)}

## Neural Network
neural_net = function(X, Y, theta, m, nu)
{
   # Relevant dimensional variables:
   N = dim(X)[1]
   p = dim(X)[2]
   q = dim(Y)[2]
   a = p
   Z = matrix(NA, nrow = q, ncol = N)

   # Populate weight-matrix and bias vectors:
   index = 1:(p*a)
   W1 = matrix(theta[index],p,a)
   index = max(index)+1:((p+a)*m)
   W2 = matrix(theta[index],p+a,m)
   index = max(index)+1:(m*m)
   W3 = matrix(theta[index],m,m)
   index = max(index)+1:(m*q)
   W4 = matrix(theta[index],m,q)

   index = max(index)+1:a
   b1 = matrix(theta[index],a,1)
   index = max(index)+1:m
   b2 = matrix(theta[index],m,1)
   index = max(index)+1:m
   b3 = matrix(theta[index],m,1)
   index = max(index)+1:q
   b4 = matrix(theta[index],q,1)

   # Storage
   Yhat = matrix(NA,N,q)
   error = rep(NA,N)

   # Evaluate network:
   for(i in 1:N)
   {
          a0 = matrix(X[i,],p,1)

          z1 = t(W1) %*% a0 + b1
          a1 = sig1(z1)

          z2 = t(W2)%*% rbind(a0,a1) + b2
          a2 = sig2(z2)
```

```
        z3 = t(W3) %*% a2 + b3
        a3 = sig3(z3)

        z4 = t(W4) %*% a3 + b4
        Z[,i] = z4
  }
  A_L = softmax(Z)

  Yhat = A_L
  error = g(Yhat,Y)

  # Calculate error:
  E1 = error
  E2 = E1+nu*(sum(W1^2) + sum(W2^2) + sum(W3^2) + sum(W4^2))/N
  # modified objective/penalised objective

  # Return predictions and error:
  return(list(Yhat = Yhat, E1 = E1, E2 = E2))
}

X <- as.matrix(data[,1:3])
Y <- as.matrix(data[,4:6])
theta <- runif(75,min = -1,max = 1)
results <- neural_net(X,Y,theta,m = 4, nu = 0)
```

Listing 3: R implementation of the forward pass for an (m,m) augmented feedforward neural network with tanh activations and L2 regularization. The function computes predicted outputs, cross-entropy error, and the penalized objective value for a given parameter set.

7 Question g

```
library(ggplot2)
set.seed(2025)

cross_validation <- function(k,X,Y,theta){
  n_val <- 10
  nu_val <- exp(seq(-6,2,length = n_val ))
  CV_Errors <- c()

  for(i in 1:k){
    errors <- c()
    indices <- 1:nrow(X)
    index_train <- sample(indices,0.8*nrow(X),replace = FALSE)
    training_X <- X[index_train,]
    training_Y <- Y[index_train,]
    index_test <- indices[-index_train]
    test_X <- X[index_test,]
    test_Y <- Y[index_test,]

    for (v in 1:n_val) {
        print(c(i,v))
        theta <- runif(75,min = -1,max = 1)
        params <- optim(theta, fn = \(theta) neural_net(training_X, training_Y,
            theta, m=4, nu=nu_val[v])$E2, method = "BFGS")
        results <- neural_net(X = test_X,Y = test_Y, theta = params$par, m = 4, nu =
             nu_val[v])
        errors[v] <- results$E1
      }
    CV_Errors <- cbind(CV_Errors,errors)
  }
  CV_Errors <- rowMeans(CV_Errors)
  result <- cbind("Nu Values" = nu_val, "CV Errors"= CV_Errors)
  return(result)
}
```

Listing 4: R function for performing K-fold cross-validation to evaluate neural network performance across a range of regularization parameters (). The function computes average validation error over multiple folds to identify the optimal regularization strength.

```
CV_results <- cross_validation(k=3,X,Y, theta)
colnames(CV_results) <- c("Nu_Values","CV_Error")

optimal_nu_index <- which.min(CV_results[,2])
optimal_nu <- unname(CV_results[optimal_nu_index,1])

ggplot() + geom_line(data = as.data.frame(CV_results), aes(x = Nu_Values, y = CV_
    Error)) + theme_minimal()+
  geom_vline(xintercept = optimal_nu, linetype = "dashed", color = "red") +
  annotate("text",x=optimal_nu, y=-Inf, label = paste0("Nu Value = ", round(optimal_
      nu,3)), vjust = -0.8, size = 3, hjust = -0.1, color = "red") +
  labs(
    title = "Three Fold Cross Validation Across\nNu Regularization Parameter",
    x = "Nu Regularised Values",
    y = "CV Error"
  ) +
  theme_minimal() +
  theme(aspect.ratio = 1)
```

Listing 5: R code for performing 3-fold cross-validation to evaluate the effect of regularization parameter on model performance. The plot highlights the optimal that minimizes cross-validation error.

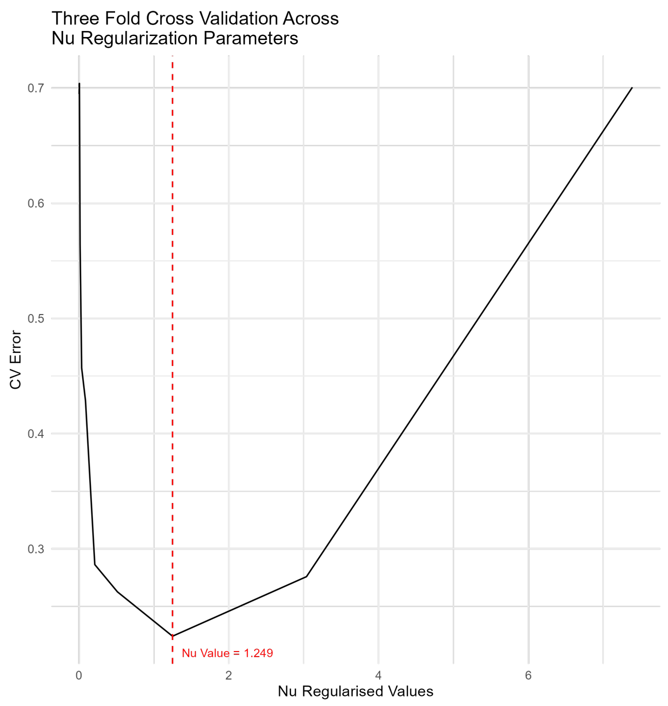

Figure 2: Three Fold Cross Validation AcrossRegularization Parameters

The selected regularization level, = 1.249, is justified by its position at the minimum of the cross-validation (CV) error curve shown in Figure 2. This value corresponds to the point where the model achieves the lowest average prediction error across the three folds of the validation process, indicating an optimal balance between underfitting and overfitting. At lower values of , the model is likely too flexible, capturing noise in the training data, while higher values impose excessive regularization, reducing the model's capacity to fit the underlying patterns. By selecting the value of that minimizes CV error, we ensure that the model generalizes well to unseen data while maintaining an appropriate level of complexity.

8 Question h

```
res <- 1000
x1_seq <- seq(min(data$X1), max(data$X1), length.out = res)
x2_seq <- seq(min(data$X2), max(data$X2), length.out = res)

# full grid
grid <- expand.grid(
  X1 = x1_seq,
  X2 = x2_seq,
  X3 = c(0,1)
)

# Find the optimal parameters given the optimised regularized parameter using
    training data
theta_final <- runif(75, -1, 1)
optimal_theta <- optim(theta_final,
           fn = \(theta) neural_net(X, Y, theta, m=4, nu=optimal_nu)$E2,
           method = "BFGS")

# Find predicted probabilities using the grid of inputs and dummy y variable for
    storage
dummy_Y <- matrix(0, nrow = nrow(grid), ncol = 3)
Yhat_final <- neural_net(
  X = as.matrix(grid[, c("X1", "X2", "X3")]),
  Y = dummy_Y,
  theta = optimal_theta$par,
  m = 4,
  nu = optimal_nu
)

probs <- Yhat_final$Yhat # matrix nrow(grid)  3
```

Listing 6: R code for evaluating predicted class probabilities over a grid of input variables using the optimized neural network model with softmax output. The network is trained with L2 regularization and evaluated on a dense grid for visualization.

```r
# Assign predicted class
grid$pred <- factor(
  apply(probs, 1, which.max),
  levels = 1:3,
  labels = c("alpha","beta","rho")
)

# Split grid by X3 and plot separately
grid0 <- subset(grid, X3 == 0)
grid1 <- subset(grid, X3 == 1)

# Plot for X3 = 0
p0 <- ggplot(grid0, aes(x = X1, y = X2, fill = pred)) +
  geom_tile() +
  labs(
    title = "Predicted Response Regions By Input Variables\nAnd Detector Type B (X3
        = 0)",
    x = "X1",
    y = "X2",
    fill = "Class"
  ) +
  theme_minimal()+
  theme(aspect.ratio = 1)

# Plot for X3 = 1
p1 <- ggplot(grid1, aes(x = X1, y = X2, fill = pred)) +
  geom_tile() +
  labs(
    title = "Predicted Response Regions By Input Variables\nAnd Detector Type A (X3
        = 1)",
    x = "X1",
    y = "X2",
    fill = "Class"
  ) +
  theme_minimal()+
  theme(aspect.ratio = 1)

# display
print(p0)
print(p1)
```

Listing 7: "R code for generating predicted response region plots over input space (X1, X2), split by detector type (X3). Predicted classes are derived from the regularized neural network's softmax outputs.

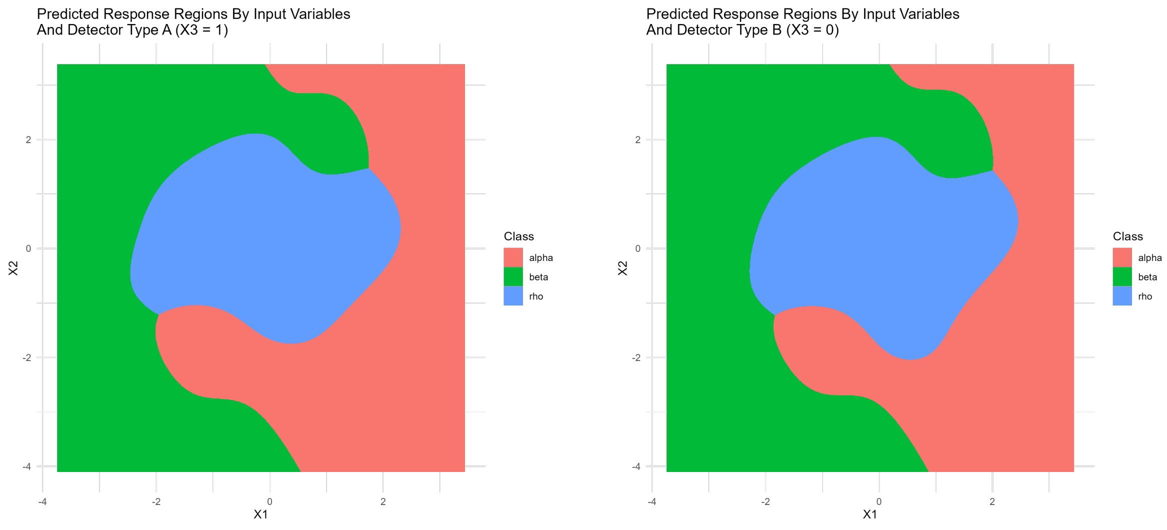

Figure 3: Side-by-side comparison of Detector A and B predictions.

The near-identical response regions for Detector Types A and B indicate that X3 (detector type) has little influence on classification and is thus a weak predictor for particle type. In contrast, X1 and X2 strongly affect predictions, as shown by the distinct and non-linear decision boundaries shaped across their values. This suggests particle type is primarily determined by spatial coordinates, rather than detector type.

9 Question i

Reduced Need for Manual Feature Engineering
Augmented Feature Networks (AFNs) automatically expand the feature space using learned transformations (e.g., polynomials or interactions), effectively performing automated feature engineering. In contrast, Feedforward Neural Networks (FNNs) rely on hidden layers to implicitly learn these representations without explicitly enriching the input space. This makes AFNs particularly useful in domains like particle physics, where raw inputs (X1, X2, X3) may not reveal key patterns such as angular dependencies—patterns that AFNs can uncover without manual intervention.

Expanding The Feature Space Systematically
AFNs expand the feature space by explicitly including transformed versions of each input which provide more structured representations of the interactions between inputs, thus allowing the model to better capture the underlying non-linear patterns. In contrast, FNNs rely on hidden layers to implicitly learn these interactions, which may limit their ability to uncover complex relationships without extensive deep archited chtures. This makes AFNs more effective in capturing intricate dependencies that might otherwise be overlooked in traditional feedforward networks

In summary, AFNs reduce the need for manual intervention for feature generations while also expanding the feature space leading to better generalization and performance, especially when the relationship between inputs and outputs is nonlinear or not easily separable in the original feature space.