

# Supplementary material 2 for The Double Descent Behavior in Two Layer Neural Network for Binary Classification

## S.1 Generating the plots in Section 2 and Section 6

Here we present the R/RStudio code used to draw the figures in Section 2 and Section 6.

```
#-----  
# Wisconsin Breast Cancer Dataset Analysis in Section 2  
#-----  
library(tidyverse)  
library(keras)  
library(caret)  
library(fastDummies)  
library(readxl)  
library(dplyr)  
library(tensorflow)  
  
set.seed(42)  
tensorflow::set_random_seed(42) # Ensures reproducibility in Keras  
  
# Load data  
df <- read_excel("wisconsin_data.xlsx")  
df$Y <- ifelse(df$Y == "B", -1, 1) # Convert 'B' to -1 and 'M' to 1  
final <- df  
  
# Normalize the features  
X <- final %>%  
  select(-Y) %>%  
  scale()  
  
y <- to_categorical(final$Y)  
  
# Function to train the model and return test error  
train_and_evaluate <- function(X, y, sample_size) {  
  # Ensure the sample size doesn't exceed the dataset size  
  sample_size <- min(sample_size, nrow(X))  
  
  # Subset data to the current sample size  
  set.seed(42) # Ensure reproducibility for random sampling  
  sample_indices <- sample(1:nrow(X), size = sample_size, replace = FALSE)  
  
  X_train_subset <- X[sample_indices, , drop = FALSE]  
  y_train_subset <- y[sample_indices, ]  
}
```

```

# Define the test set as the rest of the data
X_test <- X[-sample_indices, , drop = FALSE]
y_test <- y[-sample_indices, ]

# Build the model
model <- keras_model_sequential() %>%
  layer_dense(units = 30, activation = 'relu', input_shape = ncol(X),
  kernel_regularizer = regularizer_l2(0.000001)) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'sigmoid')

model %>% compile(
  loss = 'binary_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)

# Train the model
model %>% fit(
  X_train_subset, y_train_subset,
  epochs = 200,
  batch_size = 5,
  validation_data = list(X_test, y_test),
  verbose = 0
)

# Evaluate the model on the test data
test_results <- model %>% evaluate(X_test, y_test, verbose = 0)

# Return the loss (test error)
return(test_results[1]) # Access loss directly
}

# Sample sizes to consider (increasing from 5 to the size of the full dataset)
sample_sizes <- c(seq(10, 40, 5), seq(50, 550, by = 10))

# Record test errors for each sample size
test_errors <- sapply(sample_sizes, function(n) {
  train_and_evaluate(X, y, n)
})

# Create a data frame for plotting
results <- data.frame(
  Alpha = sample_sizes / 30,
  TestError = test_errors
)

require(mass)
library(scales)

```

```

# Plot the test error curve
ggplot(results, aes(x = Alpha, y = TestError, color = "lambda_="0.000001")) +
  geom_line() +
  geom_point() +
  labs(
    title = "Test_Error_vs._Sample_Size/Dimension",
    x = expression(paste(alpha)),
    y = "Test_Error"
  ) +
  scale_x_continuous(n.breaks = 8, limits = c(0, 8)) +
  geom_vline(xintercept = 1, linetype = "dashed", color = "black", size = 0.5) +
# Add vertical line at x = 1
  scale_color_manual(values = c("lambda_="0.000001" = "blue")) +
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(colour = "black"),
    panel.border = element_rect(color = "black", fill = NA, size = 0.5),
    legend.text = element_text(size = rel(1)),
    legend.title = element_blank()
  )

```

---

```

#
# Generating plots in Section 6
#
library(tidyverse)
library(matlib)
library(MASS)
library(quadprog)
library("ggplot2")
set.seed(100)

quad <- function(a, b, c)
{
  a <- as.complex(a)
  answer <- c((-b + sqrt(b^2 - 4 * a * c)) / (2 * a),
              (-b - sqrt(b^2 - 4 * a * c)) / (2 * a))
  if(all(Im(answer) == 0)) answer <- Re(answer)
  if(answer[1] == answer[2]) return(answer[1])
  if(answer[1] <= 0) return(answer[2])
  if(answer[2] <= 0) return(answer[1])
  answer
}

```

```

# Parameters
features <- 60 # Dimension/ number of features

```

```

N<-300 # maximum sample size
eta <- rnorm(features) # Gaussian vector eta
l2_lambda <- 1e-5 # L2 regularization
alpha<-N/features # Ratio of sample size/features, increase alpha from 1 to 10

y <- sample(c(1, -1), size = N, replace = TRUE) # Random binary vector of length N
probabilities <- table(y) / sum(table(y))
rho1<-probabilities[[2]] # probability of getting y=-1
rho2<-probabilities[[1]] # probability of getting y=1
x <- matrix(0, nrow = N, ncol = features) # Initialize matrix for x_i
for (i in 1:N) {
  epsilon <- rnorm(features) # Generate epsilon
  x <- (eta * y)/sqrt(features) + epsilon # Calculate x_i
}

steps<-0.05 # Steps to plot the curve
TestError<-matrix(data=NA, ncol=1, nrow=(alpha/steps))
begin<-0.05
i=0
j=0
for(l in c(l2_lambda)){
  j<-j+1
  for(a in seq(begin, alpha, steps)){
    i<-i+1
    g<-quad(4*l, a+4*l-1, -1)
    g1<-sqrt((a+4*l-1)^2+16*l)
    s<-(8*a*g*rho1*rho2)/(1+g+4*a*rho1*rho2*g)
    b<-(2*rho1-1)*(2-s)
    r<-((a*g^2*(-b^2+(s-2)^2)+(1+g)^2*s^2)/((1+g)^2-a*g^2))
    TestError[i, j]<-1-rho1*pnorm((s+b)/sqrt(r), 0, 1)-rho2*pnorm((s-b)/sqrt(r), 0, 1)
    print(r)
  }

  i=0
}

# Reshape data frame
a = seq(begin, alpha, steps)
l1=TestError[,1]

k<-which.max(TestError[,1])
ggplot() + geom_line(aes(x=a, y=l1, group=1, color='\\u03BB=0.00001')) +
  labs(x=expression(paste(alpha)), y="Test_error")+
  scale_x_continuous(breaks = c(seq(0, 10, by = 1)))+
  theme(legend.position = c(0.8, 0.6), legend.title = element_blank())+
  geom_vline(xintercept = k*steps, linetype="dotted")+
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
    panel.background = element_blank(),
    axis.line = element_line(colour = "black"),

```

```
panel.border = element_rect(color = "black", fill = NA,  
                             size = 0.5))+  
theme(legend.text=element_text(size=rel(1)))
```