A4

2023-09-30

Question 4

Algorithms and tools built using neural networks have gain a lot of promenance over the last 10 years. However, this uptake is probably more due to increases in processing power of modern computers as well as in the availability of large datasets rather than in the development of new mathematics. It is interesting to understand how random matrix theory (RMT) can play a role in understanding how neural networks function. Given p-dimensional data observations $x_1, x_2, ..., x_n$, we stack them into a data matrix $X = [x_1, ..., x_n]$ of size $p \times n$. A classic neural network model is given by $f(WX)$ where W is a $r \times p$ matrix of weights and $f : R \to R$ is a function (applied component-wise). This model and its connection to RMT has been recently studied in [A] and [B]. Setting $Z := f(WX)$, the authors in both papers study the Gram matrix

$$\frac{1}{n} ZZ^T$$

where W is a random matrix with univariate normal distributed $N(0, \sigma_w^2/p)$ entries and X has entries with distribution $N(0, \sigma_x^2)$. This is sometimes called a random feature network.

(a) In [A], they consider the case where f is a variant of the rectified linear unit (ReLU) activation function given by f_α; see equation (17) in paper. Implement this function f_{alpha} and plot it for various values of $alpha$ and compare it to the (classic) ReLU function.

The function f_α is defined as below:

$$f_\alpha(x) = \frac{[x]_+ + \alpha[-x]_+ - \frac{1+\alpha}{\sqrt{2\pi}}}{\sqrt{\frac{1}{2}(1+\alpha^2) - \frac{1}{2\pi}(1+\alpha)^2}}$$

Let's write a function to realize $f_\alpha(x)$

```r
f_alpha <- function(x, alpha){
  numerator = pmax(x, 0) + alpha * pmax(-x, 0) - (1 + alpha)/sqrt(2*pi)
  denominator = sqrt( (1+alpha^2)/2 - (1+alpha)^2/2/pi )
  return(numerator / denominator)
}
```

Let's plot the values of $f_\alpha(x)$ against the classical rectified linear unit (ReLU) function, which is simply in the form:

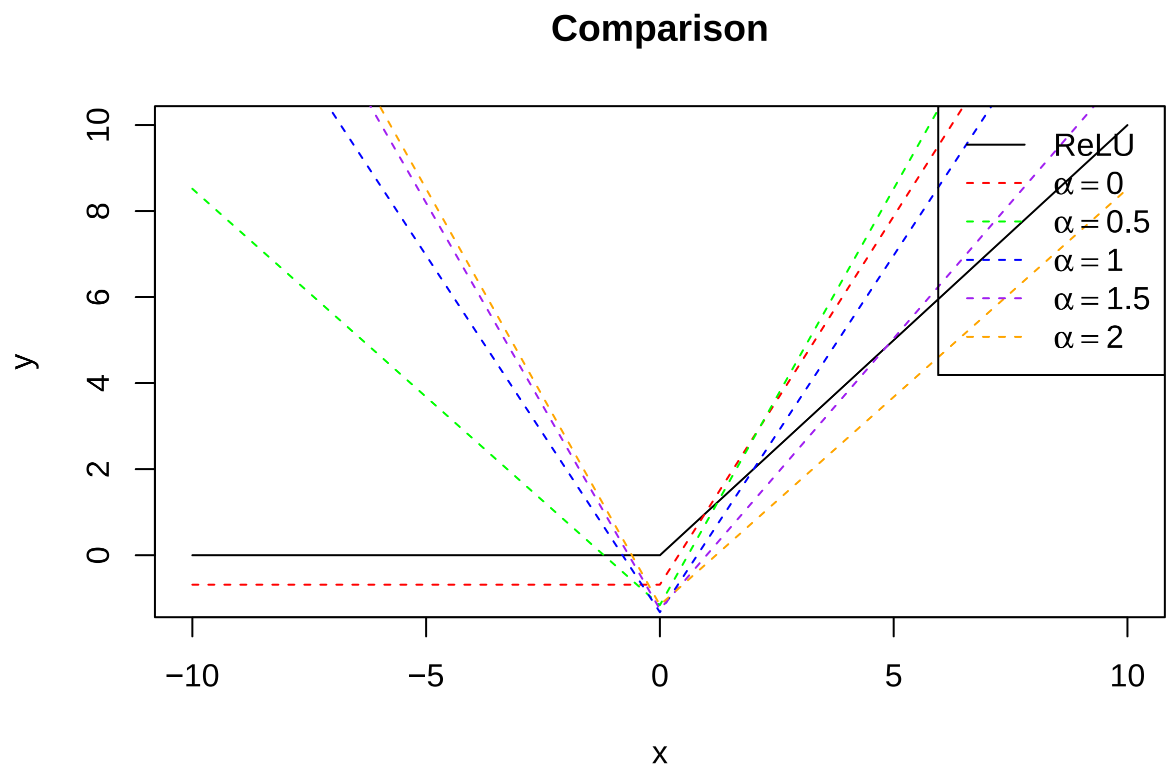
$$f_{ReLU}(x) = max\{0, x\}$$

We can create our own ReLU implementation:

```
f_relu <- function(x) {
  return(pmax(0, x))
}
```

The plot is shown below.

```
# Create a sequence of x values
x <- seq(-10, 10, by = 0.1)

# Defined contrasting colors
colors <- c("red", "green", "blue", "purple", "orange")

# Plot ReLU function
plot(x, f_relu(x), type = "l", col = "black", ylim = c(-1, 10), xlab = "x", ylab = "y", main = "Comparis

# Plotting loop with new colors
for (i in seq_along(colors)) {
  alpha <- i *0.5  - 0.5  # Adjust alpha based on index i
  lines(x, f_alpha(x, alpha), col = colors[i], lty = 2)
}

# Updated legend with new colors
legend("topright", legend = c("ReLU", expression(alpha == 0), expression(alpha == 0.5), expression(alpha
```



Immediately, we can see that the function reduces to a quasi-ReLU function with a shift and a scaling factor when the $\alpha = 0$:

$$f_0(x) = \frac{[x]_+ - \frac{1}{\sqrt{2\pi}}}{\sqrt{\frac{1}{2} - \frac{1}{2\pi}}}$$

Also, it can be shown that the function is reduces to a quasi-l1 norm function with a shift and a scaling factor when the $\alpha = 1$:

$$f_\alpha(x) = \frac{|x| - \frac{2}{\sqrt{2\pi}}}{\sqrt{1 - \frac{2}{\pi}}}$$

The plot showcases how the function $f_\alpha(x)$ varies with different values of α compared to the classical ReLU function. As α increases, the function $f_\alpha(x)$ starts to incorporate negative values of x more significantly, deviating further from the classical ReLU which only considers positive values of x. This behavior could imply that the function $f_\alpha(x)$ could provide a smoother transition between negative and positive values of x, possibly enhancing the learning process in certain neural network architectures by handling negative inputs differently.

(b)The paper defines (see Theorem 1) that

$$\eta := E[f(\sigma_w \sigma_x \xi)^2], \zeta := E[\sigma\_w\sigma\_x f'(\sigma_w \sigma_x \xi)]^2,$$

where $\xi \sim N(0,1)$. The paper claims that when $f = f_\alpha$, it is straightforward to check that (see near equation (18) in paper),

$$\eta := 1, \zeta := \frac{(1-\alpha)^2}{2(1+\alpha^2) - \frac{2}{\pi}(1+\alpha)^2}.$$

Proceed to check this is true. Note that $[x]+ = max(0, x) = x1_{\{x>0\}}$.

We firstly write down the formula of η and ζ:

$$\eta = \int dz \frac{e^{-z^2/2}}{\sqrt{2\pi}} f(\sigma_w \sigma_x z)^2, \ \zeta = \left(\sigma_w \sigma_x \int dz \frac{e^{-z^2/2}}{\sqrt{2\pi}} f'(\sigma_w \sigma_x z) \right)^2$$

It is easy to show the result. (See handwritten note for details)

(c) Consider the empirical spectral distribution (ESD) of $\frac{1}{n} Z_\alpha Z_\alpha^T$ for various choices of α where $Z_\alpha := f_\alpha(WX)$ and take $\sigma_w = \sigma_x = 1$. Compare the histogram of the ESD against the Marchenko-Pastur density. Justify and discuss your choices of parameters (e.g., see Section 3.2.2 in [A]). What is the value of α that best matches the Marchenko-Pastur distribution?

Let's write our own Marchenko-Pastur density function:

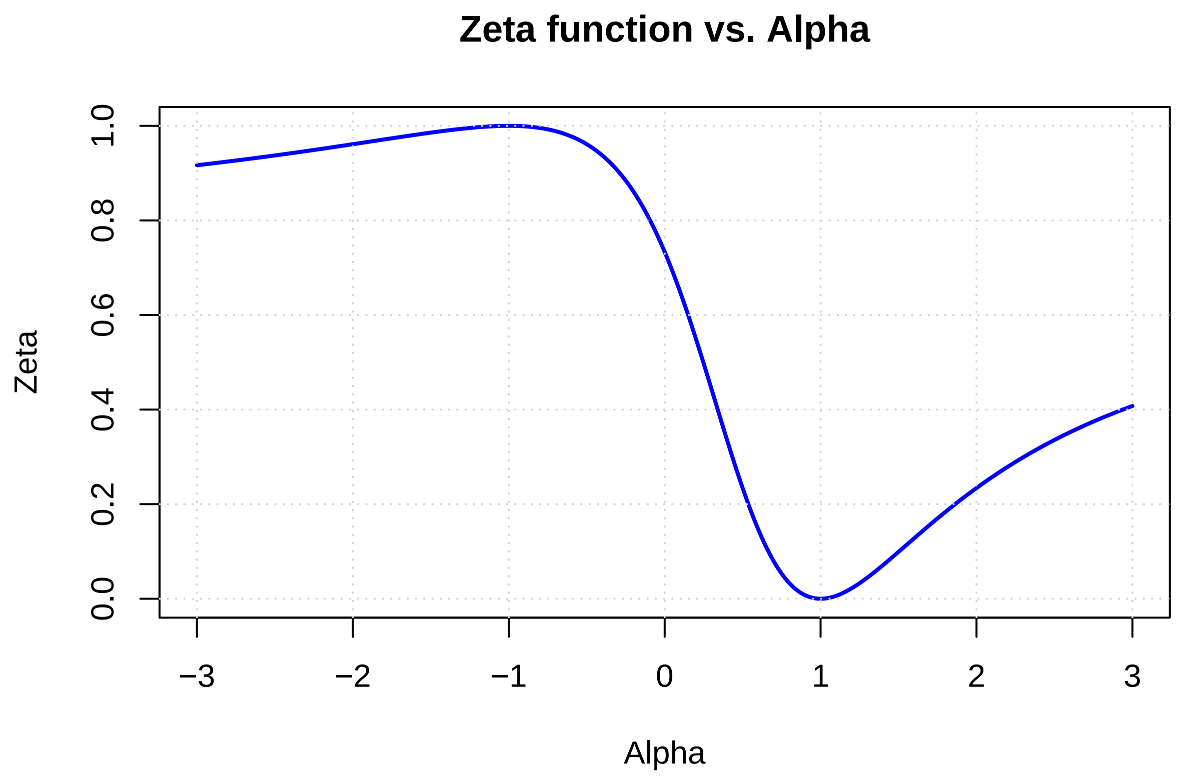
```
dmp = function(x, y, sigma=1) {
  a = (1-sqrt(y))^2
  b = (1+sqrt(y))^2
  suppressWarnings(ifelse((x <= a) | (x >= b), 0, sqrt((x - a) * (b - x))/(2 * pi * sigma * x * y)))
}
```

As it is illustrated that each entry X_{iu} and W_{ij} is independently and identically distributed, this immediately enable us to have the simulation function with parameter Ψ, Φ, α:

```r
RandomNN_Eigen <- function(m, Phi, Psi, alpha){
  n0 = m * Phi
  n1 = n0 / Psi

  mu_x <- rep(0, m)
  sigma_x <- diag(m)
  X <- rmvn(n0, mu_x, sigma_x, ncores=8)

  mu_w <- rep(0, n0)
  sigma_w <- diag(x = 1/n0, n0)
  W <- rmvn(n1, mu_w, sigma_w, ncores=8)

  Z_alpha = f_alpha(W %*% X, alpha)
  target = Z_alpha %*% t(Z_alpha) / m
  L = eigen(target, only.values = TRUE)$values
}
```

To make a reasonable guess on the possible α values that align the best with the Marchenko-Pastur distribution, let's investigate the relationship with ζ and α after we adopt the f_α as our activation function.

```r
zeta_function <- function(alpha) {
  numerator <- (1 - alpha)^2
  denominator <- 2 * (1 + alpha^2) - (2/pi) * (1 + alpha)^2
  return(numerator / denominator)
}
# Generate a sequence of alpha values
alpha_values <- seq(-3, 3, by=0.01)

# Compute zeta values for each alpha
zeta_values <- sapply(alpha_values, zeta_function)

# Plot
plot(alpha_values, zeta_values, type="l", col="blue", lwd=2,
     main="Zeta function vs. Alpha", xlab="Alpha", ylab="Zeta")
grid()
```



Applied the L'Hôpital's rule twice, we have the following function when the $\alpha \to \infty$ and $\alpha \to -\infty$.

$$f(\alpha) = (1-\alpha)^2 g(\alpha) = 2(1+\alpha^2) - \frac{2}{\pi}(1+\alpha)^2$$

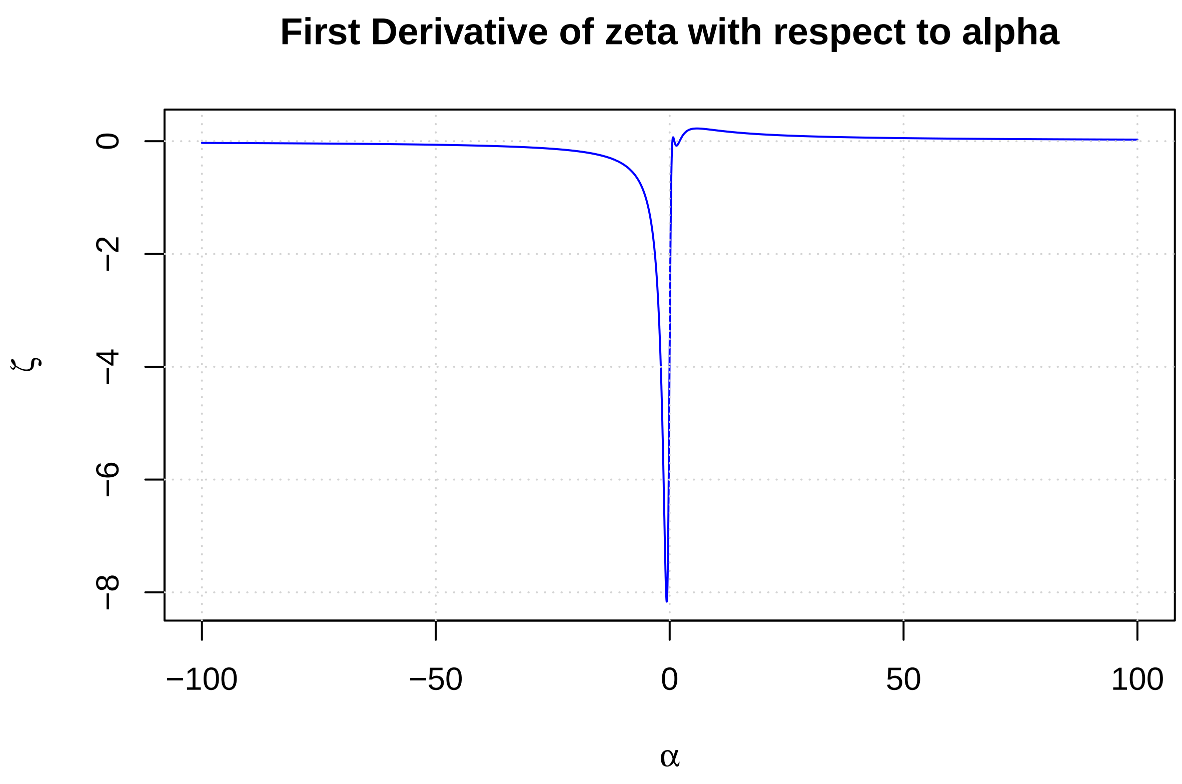
$$\lim_{x\to\pm\infty} \frac{f(x)}{g(x)} = \lim_{x\to\pm\infty} \frac{f'(x)}{g'(x)} = \lim_{x\to\pm\infty} \frac{f''(x)}{g''(x)} = \lim_{x\to\pm\infty} \frac{1}{2-\frac{2}{\pi}} \approx 0.733$$

And the first derivative is shown below:

$$\zeta' = \frac{-2(1-\alpha)\times\left[2(1+\alpha^2) - \frac{2}{\pi}(1+\alpha)^2\right] - (1-\alpha)^2 \times \left[4\alpha - \frac{2}{\pi}(2+2\alpha)\right]}{\left[2(1+\alpha^2) - \frac{2}{\pi}(1+\alpha)^2\right]^2}$$

```r
# Define the function for the first derivative of zeta
zeta_prime <- function(alpha) {
  numerator <- -4*(1 - alpha)*(1 - 1/pi) - 4*alpha^2*(1 - alpha)*(1 - 1/pi) + 8*alpha*(1 - alpha) + 4*al
  denominator <- (2*(1 - 1/pi) + 2*alpha^2*(1 - 1/pi))^2
  return(numerator/denominator)
}

# Generate alpha values
alpha_vals <- seq(-100, 100, by = 0.01)

# Compute zeta_prime values
zeta_prime_vals <- sapply(alpha_vals, zeta_prime)
```

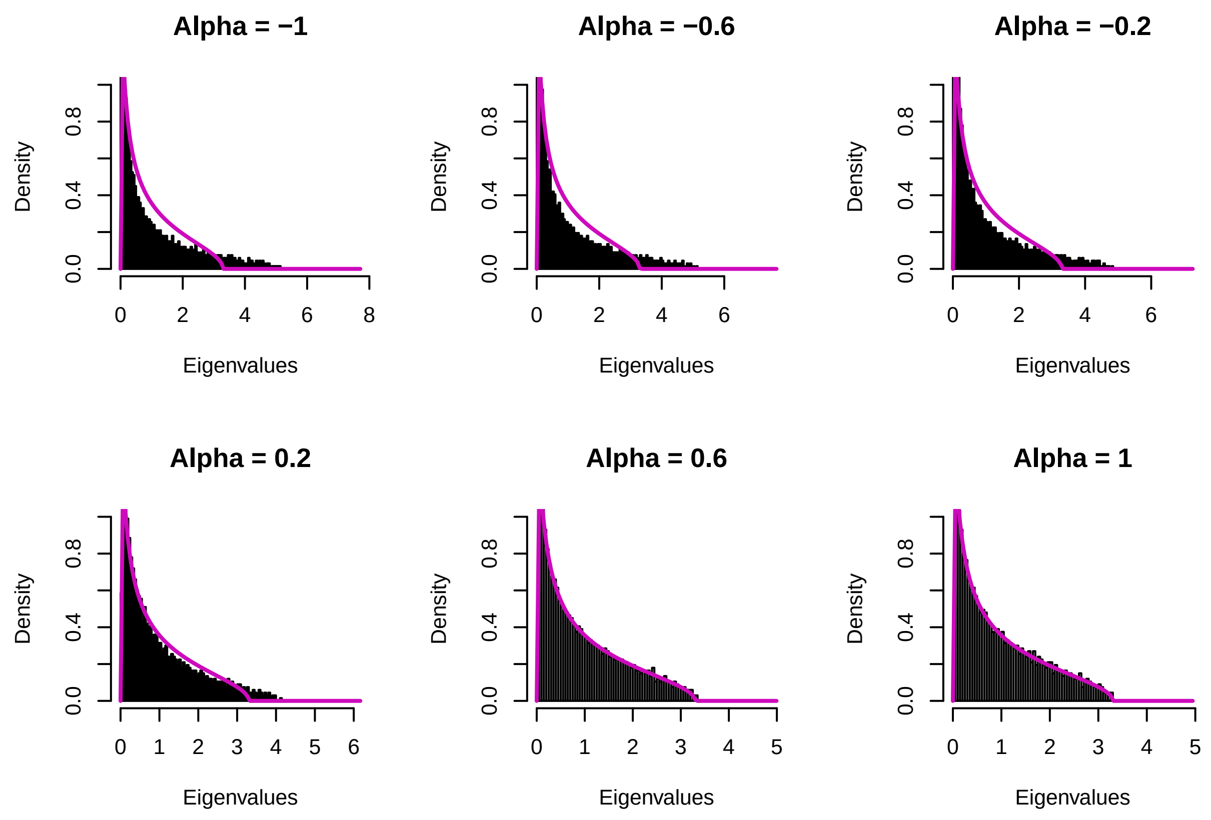
```
# Plot
plot(alpha_vals, zeta_prime_vals, type="l", col="blue", xlab=expression(alpha), ylab=expression(zeta), 
grid()
```



This first derivative is approching to 0 from above and below on $x \rightarrow \pm \infty$. Thus, our function is monotonically increasing and monotonically decreasing in $\alpha > 1$ and $\alpha < -1$.

The ζ function have some special values at $\alpha = -1, \zeta = 1$ and $\alpha = 1, \zeta = 0$. When $\zeta = 1$, it also equals to η, which is the special case found in 3.2.1. Now, we can make reasonable guess on α. From the ζ image, we can know that the $\alpha \in [-1, 1]$ covers all the possible value of ζ. Thus, let's plot the graph to compare:

```
# Parameters
m <- 2000
Phi <- 1
Psi <- 3/2
alphas <- seq(-1, 1, by= 0.4)  # different values of alpha

# Plotting
par(mfrow=c(2, 3))
for (alpha in alphas) {

  eigenvalues <- RandomNN_Eigen(m, Phi, Psi, alpha)

  hist(eigenvalues, breaks=100, probability=TRUE, main=paste("Alpha =", alpha), xlab="Eigenvalues", yla
  curve(dmp(x, Phi/Psi), from=0, to=1.5*max(eigenvalues), col=6, lwd=2, add=TRUE)
}
```



The graph shows a good fit between the empirical spectral distribution and the Machenko-Pastur distribution when \$\alpha = 1\$. The $\alpha = 0.6$ might also be a potential candinate. We can compared the *zeta* values, which is very closed.

```r
zeta_function(0.6)
```

```
## [1] 0.1467549
```

```r
zeta_function(1)
```

```
## [1] 0
```

(d) In Section 4.1 of [A], they consider a deep feedforward neural network with lth-layer post-activation matrix given by,

$$Y^l = f(W^l Y^{l-1}), Y^0 = X$$

Implement this model in R.

Denote the list containing the number of nodes on each layer of the neural network as: $[n_1, .., n_l]$ with l length. Give the Φ and Ψ, we can calculate n_1 and randomly generate the layer width after it. Thus, we can generate our random neural nets as below:

```r
RandomNN_l <- function(f, Phi, Psi, m, alpha, l){
  n0 =  m * Phi
  n1 = n0 / Psi
  #set.seed(600) # To make sure it is the same random neural netwrok each times I run
  for (i in 1: l){
    if (i == 1){
      X <- rmvn(n0, rep(0, m), diag(m))
    }
    else{
      X <- Y
    }
    W = rmvn(n1, rep(0, n0),
             diag(x = 1/n0, n0))
    Y = f(W %*% X, alpha)
    n0 <- n1
    #n1 <- sample(100:5000, 1, replace = TRUE)
  }
  return(Y)
}
```

(e) We would like to understand the distance between a limit spectral distribution (LSD) $\bar{\rho}_1$ and the empirical spectral distribution (ESD) of $Y^l(Y^l)^T$. The paper gives a distance metric:

$$d(\bar{\rho}_1, \rho_1) := \int |\bar{\rho}_1(x) - \rho_1(x)| dx$$

Implement this function and apply it to the case $\ell = 1$ and taking $\bar{\rho}_1$ to be the Marchenko-Pastur density.

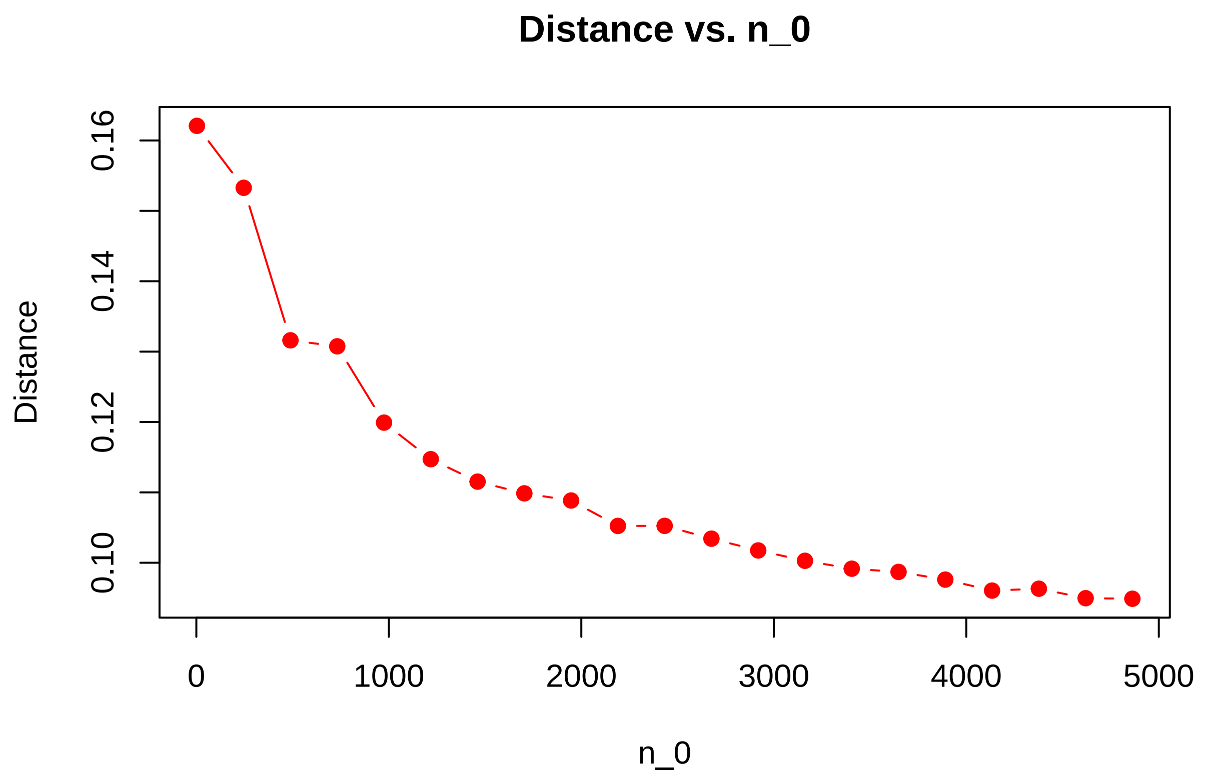
```r
dmp = function(x, y, sigma=1) {
  a = (1-sqrt(y))^2
  b = (1+sqrt(y))^2
  suppressWarnings(ifelse((x <= a) | (x >= b), 0, sqrt((x - a) * (b - x))/(2 * pi * sigma * x * y)))
}

Epdf <- function(x, eigenvalues) {
  dens <- density(eigenvalues,
                  from  = 0, # We consider the positive (semi)definite matrix YY^T/m
                  to = max(eigenvalues),
                  n=512)
  return(approx(dens$x, dens$y, xout=x)$y)
}

distance_metric <- function(eigenvalues, y, dmp, Epdf) {
  d <- function(x) {
    abs(Epdf(x, eigenvalues) - dmp(x, y))
  }

  #print(min(eigenvalues))
  #print(max(eigenvalues))
  #print((1-sqrt(y))^2)
  #print((1+sqrt(y))^2)
  #step_size = 0.001
  #x = seq(0, max(eigenvalues), by=step_size)
  #Integral = sum(d(x) * (step_size))
  #Integral = integrate(d,
  #                      lower = 0, # We consider the positive (semi)definite matrix YY^T/m
  #                     upper = max(eigenvalues),
  #                      subdivisions=10000)$value
  Integral <- quadgk(d,
                     a = 0, # We consider the positive (semi)definite matrix YY^T/m
                     b = max(eigenvalues))
  #print(Integral)
  return(Integral) # Adjust the range as needed
}
```

When $l = 1$, we can write a list of n_0 to compare the result:

```r
l = 1
alpha = 1

dis_list <- numeric()
n0s <- seq(3, 5000, by = 243)

plan(multicore, workers = 8)
```

```
## Warning in supportsMulticoreAndRStudio(...): [ONE-TIME WARNING] Forked
## processing ('multicore') is not supported when running R from RStudio because
## it is considered unstable. For more details, how to control forked processing
## or not, and how to silence this warning in future R sessions, see
## ?parallelly::supportsMulticore
```

```r
compute_distance <- function(f = f_alpha, n0, Phi = 1/80, Psi = 1, alpha = alpha, l = l) {
  n1 = n0 / Psi
  m = n0 / Phi
  RanNN = RandomNN_l(f, Phi, Psi, m, alpha = alpha, l= l)
  #print(dim(RanNN))
  eigenvalues <- eigen(RanNN %*% t(RanNN)/m, only.values = TRUE)$values
  measure <- distance_metric(eigenvalues, Phi/Psi, dmp, Epdf)
  return(measure)
}

set.seed(888)

dis_list <- future_lapply(n0s, FUN = function(n0) compute_distance(f = f_alpha, n0, Phi = 1, Psi = 3/2,

plot(n0s, dis_list, type="b", xlab="n_0", ylab="Distance", main="Distance vs. n_0",col = rainbow(1),  p
```



The distance measure goes to 0 for sufficiently large n_0 for our choice of $\Phi = \frac{1}{80}$ and $\Psi = 1$.

(f) Use the model implemented in (d) to perform the experiment given in Figure 1 of [A] in the case where $f = f_\alpha$ and we are close to the Marchenko-Pastur distribution for the first-layer limiting distribution. Reproduce the figure in the cases $\ell = 1$ and $\ell = 3$ (Note that $\ell = L$ in Figure 1).

Let's create of list of alpha values to carry out experiments.

```
alphas = c(-1, -1/4, 0, 1/4, 1)
```

Let's create a function to let us carry out experiments easily.

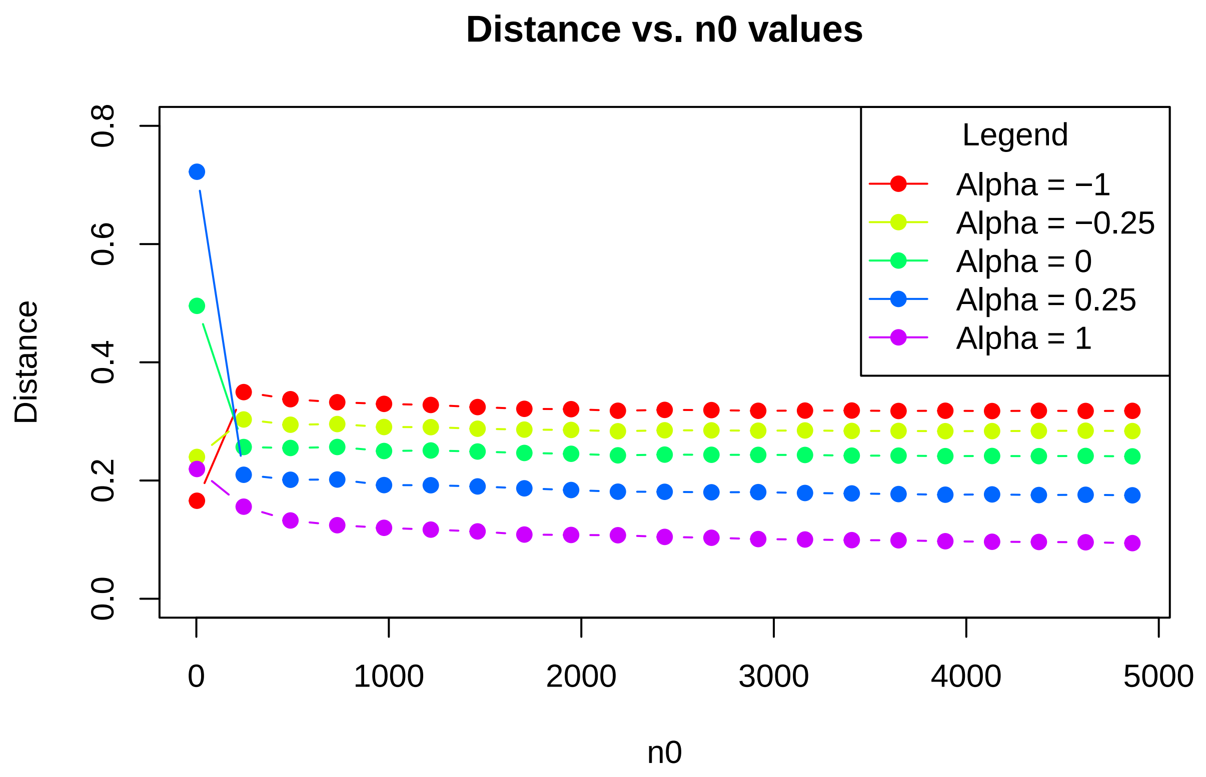
```
Random_NN_exp <- function(alpha = alpha, l = 1, Phi = 1, Psi = 3/2, n0s = n0s){
  dis_list <- numeric()
  set.seed(12)
  dis_list <- future_lapply(n0s, FUN = function(n0) compute_distance(f = f_alpha, n0, Phi = Phi, Psi = 
  return(dis_list)
}
```

For $\ell = 1$, the distance measures align with each others and decrease to 0 when $n_0 \to \infty$ for every α in our list.

```
plan(multicore, workers = 8)
n0s <- seq(3, 5000, by = 243)
#result1 <- future_lapply(alphas, FUN = function(alpha) Random_NN_exp(alpha, l = 1, Phi = 1, Psi = 3/2,
#View(data.frame(result1))

df1 <- data.frame()

set.seed(600)

for (alpha in alphas){

  row_df <- as.data.frame(t(Random_NN_exp(alpha, l = 1, Phi = 1, Psi = 3/2, n0s = n0s)))

  df1 <- rbind(df1, row_df)

}
```

Let's plot the result:

```
colors <- rainbow(5)

# Plotting
plot(n0s, unlist(df1[1,]), type="b", xlab=expression(n0), ylab="Distance", main="Distance vs. n0 values
lines(n0s, unlist(df1[2,]), col=colors[2], type="b", pch=19)
lines(n0s, unlist(df1[3,]), col=colors[3], type="b", pch=19)
lines(n0s, unlist(df1[4,]), col=colors[4], type="b", pch=19)
lines(n0s, unlist(df1[5,]), col=colors[5], type="b", pch=19)
legend_labels <- paste("Alpha =", alphas)
legend("topright", legend=legend_labels, col=colors, pch=19, lty=1, title="Legend")
```



For every case of α values, the distance between the empirical spectral distribution of the random neural network and the Marchenko-Pastur distribution goes to 0 when $n_0 \to \infty$.

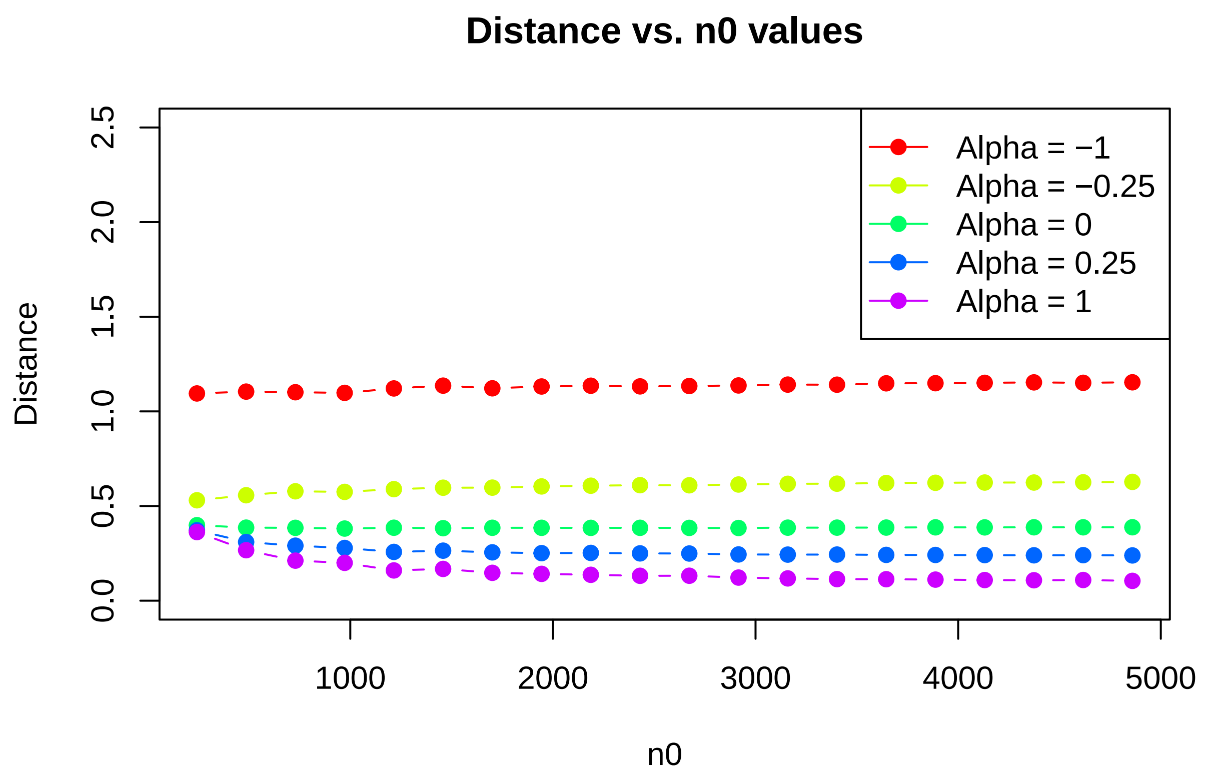
For $\ell = 3$, the distance measures are different for every choice of α.

```
plan(multicore, workers = 8)
n0s <- seq(243, 5000, by = 3^5)
#result1 <- future_lapply(alphas, FUN = function(alpha) Random_NN_exp(alpha, l = 1, Phi = 1, Psi = 3/2,
#View(data.frame(result1))
df3 <- data.frame()

set.seed(123)

for (alpha in alphas){

  row_df <- as.data.frame(t(Random_NN_exp(alpha, l = 10, Phi = 1, Psi = 3/2, n0s = n0s)))

  df3 <- rbind(df3, row_df)

}
```

Let's plot the result:

```
colors <- rainbow(5)

# Plotting
```

```
plot(n0s, unlist(df3[1,]), type="b", xlab=expression(n0), ylab="Distance", main="Distance vs. n0 values
lines(n0s, unlist(df3[2,]), col=colors[2], type="b", pch=19)
lines(n0s, unlist(df3[3,]), col=colors[3], type="b", pch=19)
lines(n0s, unlist(df3[4,]), col=colors[4], type="b", pch=19)
lines(n0s, unlist(df3[5,]), col=colors[5], type="b", pch=19)
legend_labels <- paste("Alpha =", alphas)
legend("topright", legend=legend_labels, col=colors, pch=19, lty=1)
```



We have covered the Figure 1 from the paper. We can see that the empirical spectral distribution of the random neural network matches the Marchenko-Pastur distribution when $n_0 \to \infty$. However, for values that is less than 1, namely $\xi > 0$, the alignment is not very strong.

Reference

[A] Pennington, Worah (2017). Nonlinear random matrix theory for deep learning. NeuroIPS 2017.

[B] Louart, Liao, Couillet (2018). A random matrix approach to neural networks. Annals of Applied Probability, Vol 28., No. 2.