UNIVERSITY OF LIVERPOOL-ION SWITCHING

Group: Highlanders

Introduction:

Many diseases, including cancer, are believed to have a contributing factor in common. Ion channels are pore-forming proteins present in animals and plants. They encode learning and memory, help fight infections, enable pain signals, and stimulate muscle contraction. If scientists could better study ion channels, which may be possible with the aid of machine learning, it could have a far-reaching impact.

When ion channels open, they pass electric currents. Existing methods of detecting these state changes are slow and laborious. Humans must supervise the analysis, which imparts considerable bias, in addition to being tedious. These difficulties limit the volume of ion channel current analysis that can be used in research. Scientists hope that technology could enable rapid automatic detection of ion channel current events in raw data.

In this competition, we'll use ion channel data to better model automatic identification methods. One will be able to detect individual ion channel events in noisy raw signals. The data is simulated and injected with real world noise to emulate what scientists observe in laboratory experiments using various deep learning and machine learning algorithms.

Abstract:

The training data is recordings in time. At each 10,000th of a second, the strength of the signal was recorded and the number of ion channels open was recorded. It is our task to build a model that predicts the number of open channels from signal at each time step. Furthermore we are told that the data was recorded in batches of 50 seconds. Therefore each 500,000 rows is one batch. The training data contains 10 batches and the test data contains 4 batches. Let's display the number of open channels and signal strength together for each training batch.

Algorithm Used:

WaveNet is a powerful new predictive technique that uses multiple Deep Learning (DL) strategies from Computer Vision (CV) and Audio Signal Processing models and applies them to longitudinal (time-series) data. The WaveNet proposes an autoregressive learning with the help of convolutional networks with some tricks. Basically, we have a convolution window sliding on the data, and at each step try to predict the next sample value that it did not see yet. In other words, it builds a network that learns the causal relationships between consequtive timesteps. (see below)

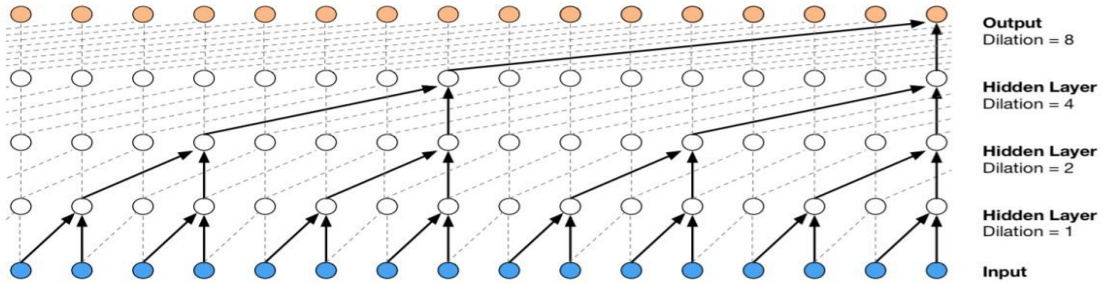


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Formulae:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

What we see below is a Gated Activation. Similar to gates in LSTM or GRUs, the tanh branch is an activation filter, or modifier of the dilated convolution that happened just below. It's the "squashing function" we've seen in CNNs before. The sigmoid branch serves essentially as a binary gate, and is able to cancel everything up to it; it learns which data is important, going back an arbitrary number of periods into the past.

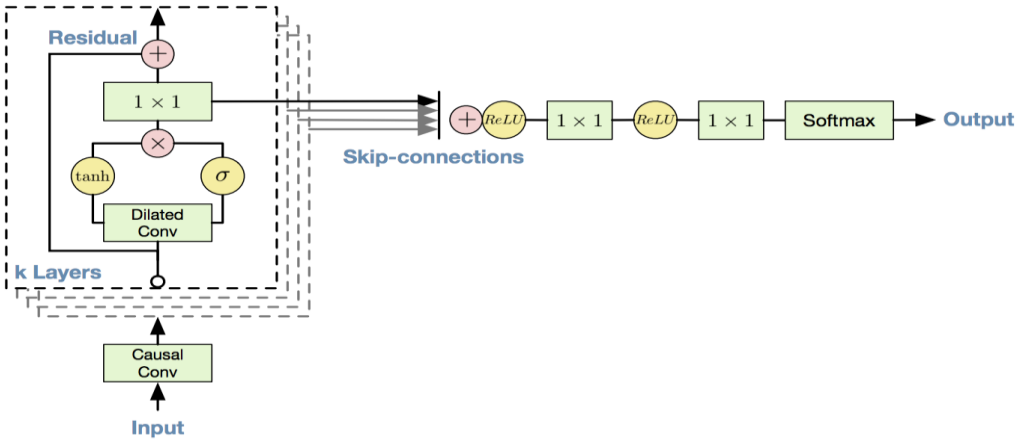
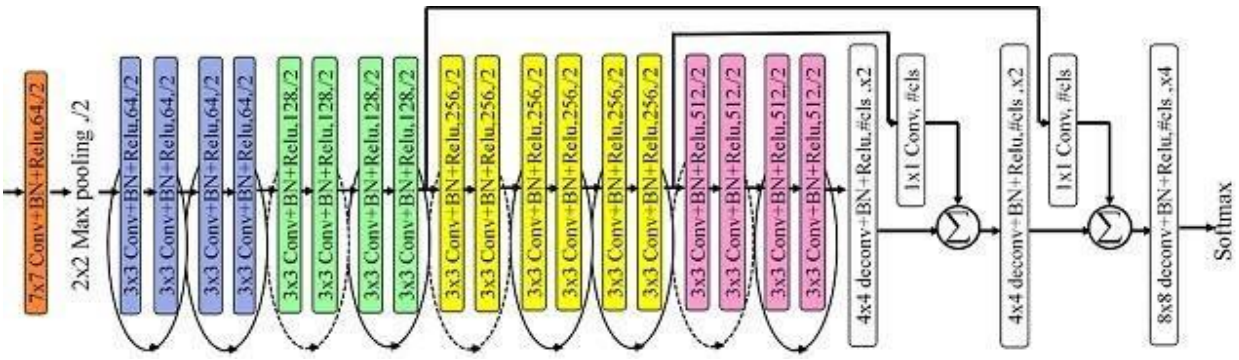


Figure 4: Overview of the residual block and the entire architecture.

Also note the grey arrows pointing right: these are Skip Connections. They allow a complete bypass of convolution layers, and give raw data the ability to influence the formulation of predictions — again — to an arbitrary number of periods into the future. Don't worry, these are hyper-parameters that you can validate on slices of your data. Optimal values depend on the structure and complexity of the sequence you learn.

Remember, in fully-connected NNs, a neuron takes inputs from all neurons in the previous layer: early layers establish later ones via a hierarchy of intermediate computations. This allows NNs to build complex interactions of raw inputs/signals.

But… what if raw inputs are directly useful for prediction, and we want them to directly influence the output? In detail, skip connections allow outputs of any layer to bypass multiple future layers and skip influence dilution! Keras allows us to store the tensor output of each convolutional block — in addition to passing it through further layers — with skips.append(). Note how for each block in the stack above, the output from the gated activations joins the set of skip connections.



The overall model involves some stacks of dilated conv layers, nonlinear filter and gates, residual and skip connections, and last 1x1 convolutions.

SOURCE CODE with EXPLANATION:

```
> setwd("C:\\Users\\anush\\OneDrive\\Desktop\\fs\\dar\\finalproject")
> library(data.table)
data.table 1.12.8 using 8 threads (see ?getDTthreads).  Latest news: r-datatable.com
Warning message:
package 'data.table' was built under R version 3.6.3
```

(The data.table offers fast and memory efficient: file reader and writer, aggregations, updates, equi, non-equi, rolling, range and interval joins, in a short and flexible syntax, for faster development.)

According to the paper (on page 7 in section "Data description and dataset construction"), our competition data is synthetic data with real life "electrophysiological" noise and synthetic drift added.

First synthetic data was generated from multiple "stochastic Markovian processes" (i.e. computer programs). Then it was transmitted into the real world and recorded back into digital. (This is like playing music over a speaker and recording it back with microphone). Lastly "In some datasets additional drift was applied to the final data with Matlab". So data = generated + noise + drift.

And here we are using the clean data from kaggle website

```
> train <- fread("traink.csv")
> test <- fread("testk.csv")
```

removing the garbage values to save memory of pc

```
> gc()
         used   (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells   4771700  254.9    7364534  393.4   7364534  393.4
Vcells 152461584 1163.2  232490888 1773.8 170474558 1300.7
```

```
> time <- test$time
> #extending the time frame
> xT1 <- seq(from = -9.9998, to = 0, by = .0001)
> xT2 <- seq(from = 500.0001, to = 510.9999, by = .0001)
```

dividing the time frame into two for better approach

```
> with(train,{
+   xtend1 <<- train[time >=40.0001 & time < 50,]
+   xtend2 <<- train[time >=50.0001 & time < 61,]
+ })
```

```
> #now order the sequence
> xtend1 <- xtend1[order(xtend1$time),]
> xtend2 <- xtend2[order(xtend2$time),]
```

Assigning the sequence and group the values with train data

```
> xtend1$time <- xT1
> xtend2$time <- xT2
> train <- rbind(xtend1, train, xtend2)
```

```
> #sorting the test data
```

```
> xT1 <- seq(from = 480.0001, to = 500.0000, by = .0001)
> xT2 <- seq(from = 700.0001, to = 720.0000, by = .0001)

> with(test,{
+   xtend1 <<- rbind(test[time > 500 & time <= 510,],test[time > 500 & time <= 510,])
+   xtend2 <<- test[time > 680 & time <= 700,]
+ })
> xtend1 <- xtend1
> xtend2 <- xtend2[order(xtend2$time),]
> xtend1$time <- xT1
> xtend2$time <- xT2
> test <- rbind(xtend1, test, xtend2)
 > rm(xtend1, xtend2, xT1, xT2)
```

RcppRoll: Efficient Rolling / Windowed Operations
Provides fast and efficient routines for common rolling / windowed operations. Routines for the efficient com putation of windowed mean, median, sum, product, minimum, maximum, standard deviation and variance are provided.

```
 > library (RcppRoll)
 Warning message:
 package ‘RcppRoll’ was built under R version 3.6.3

  > lags = c(11, 26, 51, 101, 1001)


  > CRoll <- function(DF, lags){
 +
 +   Features = NULL
 +
 +   for (l in lags) {
 +     Start = Sys.time()
 +     #calculating the mean
 +     Mn <-
 +       RcppRoll::roll_mean(DF$signal,
 +                           n = l,
 +                           fill = NA,
 +                           align = "center")
 +     End = Sys.time()
 +     print(End - Start)
 +   #claculating standard deviation
 +     Start = Sys.time()
 +     Std <-
 +       RcppRoll::roll_sd(DF$signal,
 +                         n = l,
 +                         fill = NA,
```

```
+                     align = "center")
+   End = Sys.time()
+   print(End - Start)
+   #calculating the maximum
+   Start = Sys.time()
+   Mx <-
+     RcppRoll::roll_max(DF$signal,
+                        n = l,
+                        fill = NA,
+                        align = "center")
+   End = Sys.time()
+   print(End - Start)
+   #calculating the minimum
+   Start = Sys.time()
+   Mi <-
+     RcppRoll::roll_min(DF$signal,
+                        n = l,
+                        fill = NA,
+                        align = "center")
+   End = Sys.time()
+   print(End - Start)
+  #calculating the weighted mean
+   Start = Sys.time()
+   Wm <-
+     RcppRoll::roll_mean(
+       DF$signal,
+       n = l,
+       weights = c((1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1)
+                   -max((1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1))/((l-1)/2),
+                   max((1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1))*2,
+                   (((l-1)/2):1) / ((l-1)/2 * ((l-1)/2 + 1) / 1)
+                   -max((1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1))/((l-1)/2)),
```

The vector (1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1) is from 1 to l. Each element divided through the half of the sum of 1 to l. It sums up to 0.5. Furthermore, I subtract the max() from each element. This is the left side of the weights.

I do the samefor the right side but reverse from l to 1.

In the middle max((1:((l-1)/2)) / ((l-1)/2 * ((l-1)/2 + 1) / 1))*2, I add the double of the max(). This is because I realized that is is better to overweight the centre.

The whole constuct sums up to one. For example l=11. You have 5 left 1 in the centre and 5 right. Therefore, weights are:
0.03333333 0.06666667 0.10000000 0.13333333 0.33333333 0.13333333 0.10000000 0.06666667 0.03333333

```
+        fill = NA,
+        align = "center"
+      )
+
+    End = Sys.time()
+    print(End - Start)
+    Features <- cbind(Features, Mn, Std, Mx, Mi, Wm)
+
 +
 +    colnames(Features)[c(
 +      (ncol(Features) - 4),
 +      (ncol(Features) - 3),
 +      (ncol(Features) - 2),
 +      (ncol(Features) - 1),
 +      ncol(Features))] <-
 +      c(
 +        paste("Mn", l, sep = "_"),
 +        paste("Std", l, sep = "_"),
 +        paste("Mx", l, sep = "_"),
 +        paste("Mi", l, sep = "_"),
 +        paste("Wm", l, sep = "_")
 +      )
 +
 +    gc()
 +    print(l)
 +  }
 +
 +  return(Features)
 + }
```

Defining a function so that the left and right parameters are aligned statistically proper

```
> LRRolling <- function(DF){
+
+   LR = NULL
+
+   l=100
+   R_W <-
+     RcppRoll::roll_mean(
+       DF$signal,
+       n = l,
+       weights = (1:l) / (l * (l + 1) / 2),
+       fill = NA,
+       align = "right"
```

```
+    )
+  L_W <-
+    RcppRoll::roll_mean(
+      DF$signal,
+      n = l,
+      weights = (l:1) / (l * (l + 1) / 2),
+      fill = NA,
+      align = "left"
+    )
 +  l=1000
 +  R_W <-
 +    RcppRoll::roll_mean(
 +      DF$signal,
 +      n = l,
 +      weights = (1:l) / (l * (l + 1) / 2),
 +      fill = NA,
 +      align = "right"
 +    )
 +  L_W <-
 +    RcppRoll::roll_mean(
 +      DF$signal,
 +      n = l,
 +      weights = (l:1) / (l * (l + 1) / 2),
 +      fill = NA,
 +      align = "left"
 +    )
 +
 +  LR <- cbind(LR, right_W100, right_W1000, left_W100, left_W1000)
 +
 +  return(LR)
 + }

  > SplFeatures <- function(DF){
  +
  +  SFeatures = NULL
  +  #calculating the lags
  +  L1 <- c(NA, DF$signal[1:(nrow(DF)-1)])
  +  L2 <- c(c(NA,NA), DF$signal[1:(nrow(DF)-2)])
  +  L3 <- c(c(NA, NA, NA), DF$signal[1:(nrow(DF)-3)])
  +
  +  F1 <- c(DF$signal[2:nrow(DF)], NA)
  +  F2 <- c(DF$signal[3:nrow(DF)], c(NA, NA))
  +  F3 <- c(DF$signal[4:nrow(DF)], c(NA, NA, NA))
  +  #calculating the special signals
```

```
+   dL1 <- c(NA, diff(DF$signal))
+   dF1 <- c(diff(DF$signal), NA)
+   sigAb <- abs(DF$signal)
+
+   SigSq <- DF$signal^2
+   sigSqR <- sign(DF$signal)*abs(DF$signal)^(1/2)
+   SFeatures <- cbind(SFeatures, L1, L2, L3, F1, F2, F3, dL1, dF1,
+                      sigAb,
+                       SigSq,
+                      sigSqR)
+
+   return(SFeatures)
+ }
```

Formatting the train

```
> RollF <- CRoll(train, lags)
Time difference of 0.02833891 secs
Time difference of 1.408231 secs
Time difference of 0.123668 secs
Time difference of 0.120677 secs
Time difference of 0.0339098 secs
[1] 11
Time difference of 0.05784202 secs
Time difference of 1.589755 secs
Time difference of 0.2932169 secs
Time difference of 0.29425 secs
Time difference of 0.07679105 secs
[1] 26
Time difference of 0.1266539 secs
Time difference of 2.179131 secs
Time difference of 0.5296271 secs
Time difference of 0.5345681 secs
Time difference of 0.1725359 secs
[1] 51
Time difference of 0.3071918 secs
Time difference of 2.672798 secs
Time difference of 0.9953361 secs
Time difference of 1.008301 secs
Time difference of 0.362031 secs
[1] 101
Time difference of 4.482025 secs
Time difference of 23.12861 secs
Time difference of 9.690093 secs
```

```
Time difference of 9.760838 secs
Time difference of 4.610689 secs
[1] 1001
> LRF <- LRRolling(train)
 > spls <- SplFeatures(train)
 > DF_train <- cbind(train, RollF, LRF, spls)
  > rm(RollF, LRF, spls)
```

Centering the mean

```
> DF_train$signal_M1001 <- DF_train$signal - DF_train$Mn_1001
> DF_train$signal_M101 <- DF_train$signal - DF_train$Mn_101
```

Slicing the data into batches

```
> DF_train$batch <- DF_train$time %/%10
```

Checking for probability of 75%

```
> b75 <- aggregate(signal~batch, data = DF_train, FUN = quantile, probs = .75)
> colnames(b75)[2] <- "signal75"
> DF_train <- merge(x = DF_train, y = b75, by = "batch", all.x = T)
```

Checking for prob of 25%

```
 > b25 <- aggregate(signal~batch, data = DF_train, FUN = quantile, probs = .25)
 > colnames(b25)[2] <- "signal25"
 > DF_train <- merge(x = DF_train, y = b25, by = "batch", all.x = T)
```

Checking max probability

```
  > bMax <- aggregate(signal~batch, data = DF_train, FUN = max)
  > colnames(bMax)[2] <- "signalMax"
  > DF_train <- merge(x = DF_train, y = bMax, by = "batch", all.x = T)
```

Checking min probability

```
  > bMin <- aggregate(signal~batch, data = DF_train, FUN = min)
  > colnames(bMin)[2] <- "signalMin"
  > DF_train <- merge(x = DF_train, y = bMin, by = "batch", all.x = T)
```

Ordering data

```
  > DF_train <- DF_train[order(DF_train$time),]
```

Calculating upper lower difference

```
> DF_train$UL <- DF_train$Mx_1001 - DF_train$Mi_1001
```

Grouping the channels

```
 > DF_train$DD <- 0
 > DF_train$DD <- ifelse(DF_train$time <= 100, 1, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 100 & DF_train$time <= 150, 1, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 150 & DF_train$time <= 200, 3, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 200 & DF_train$time <= 250, 10, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 250 & DF_train$time <= 300, 5, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 300 & DF_train$time <= 350, 1, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 350 & DF_train$time <= 400, 3, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 400 & DF_train$time <= 450, 5, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 450 & DF_train$time <= 500, 10, DF_train$DD)
 > DF_train$DD <- ifelse(DF_train$time > 500, 1, DF_train$DD)

 >
> DF_train <- DF_train[complete.cases(DF_train),]
> rm(b75, b25, bMax, bMin, train)
> gc()
         used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   4983939  266.2    7364534  393.4    7364534  393.4
Vcells 411441632 3139.1 1179491595 8998.9 1177584321 8984.3
```

Formatting the test data

```
 > RollF <- CRoll(test, lags)
 Time difference of 0.01695395 secs
 Time difference of 0.5924139 secs
 Time difference of 0.05888915 secs
 Time difference of 0.05581188 secs
 Time difference of 0.01498795 secs
 [1] 11
 Time difference of 0.03091502 secs
 Time difference of 0.7639539 secs
 Time difference of 0.1346838 secs
 Time difference of 0.1326401 secs
 Time difference of 0.03590894 secs
 [1] 26
```

```
Time difference of 0.06184602 secs
Time difference of 0.9035661 secs
Time difference of 0.2423539 secs
Time difference of 0.243315 secs
Time difference of 0.08078599 secs
[1] 51
Time difference of 0.1476159 secs
Time difference of 1.24467 secs
Time difference of 0.456774 secs
Time difference of 0.463752 secs
Time difference of 0.173538 secs
[1] 101
Time difference of 2.089388 secs
Time difference of 8.537149 secs
Time difference of 4.449088 secs
Time difference of 4.435129 secs
Time difference of 2.128306 secs
[1] 1001
```

```
> LRF <- LRRolling(test)
> specials <- SplFeatures(test)
> DF_test <- cbind(test, RollF, LRF, specials)
> rm(RollF, LRF, specials)
```

```
> #mean calculation
> DF_test$signal_M1001 <- DF_test$signal - DF_test$Mn_1001
> DF_test$signal_M101 <- DF_test$signal - DF_test$Mn_101
```

```
>
```

```
> #dividing into batches
> DF_test$batch <- DF_test$time %/%10
```

```
> b75 <- aggregate(signal~batch, data = DF_test, FUN = quantile, probs = .75)
> colnames(b75)[2] <- "signal75"
> DF_test <- merge(x = DF_test, y = b75, by = "batch", all.x = T)
> b25 <- aggregate(signal~batch, data = DF_test, FUN = quantile, probs = .25)
> colnames(b25)[2] <- "signal25"
> DF_test <- merge(x = DF_test, y = b25, by = "batch", all.x = T)
> bMax <- aggregate(signal~batch, data = DF_test, FUN = max)
 > bMax <- aggregate(signal~batch, data = DF_test, FUN = max)
 > colnames(bMax)[2] <- "signalMax"
 > DF_test <- merge(x = DF_test, y = bMax, by = "batch", all.x = T)
```

```
> bMin <- aggregate(signal~batch, data = DF_test, FUN = min)
> colnames(bMin)[2] <- "signalMin"
> DF_test <- merge(x = DF_test, y = bMin, by = "batch", all.x = T)
```

Ordering the test data

```
> #order the data
> DF_test <- DF_test[order(DF_test$time),]
```

Upper and lower difference

```
> #upper and lower difference
> DF_test$UL <- DF_test$Mx_1001 - DF_test$Mi_1001
```

Batch channels generation

```
> DF_test$DD <- ifelse(DF_test$time <= 500, 1, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 500 & DF_test$time <= 510, 1, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 510 & DF_test$time <= 520, 3, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 520 & DF_test$time <= 530, 5, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 530 & DF_test$time <= 540, 1, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 540 & DF_test$time <= 550, 1, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 550 & DF_test$time <= 560, 10, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 560 & DF_test$time <= 570, 5, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 570 & DF_test$time <= 580, 10, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 580 & DF_test$time <= 590, 1, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 590 & DF_test$time <= 600, 3, DF_test$DD)
> DF_test$DD <- ifelse(DF_test$time > 600, 1, DF_test$DD)
```

```
> DF_test <- DF_test[complete.cases(DF_test),]
> rm(b75, b25, bMax, bMin, test)
> gc()
         used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   4984051  266.2    7364534  393.4    7364534  393.4
Vcells 528952761 4035.6 1179491595 8998.9 1179491379 8998.9
```

```
> #converting into data frames
> DF_train <- as.data.frame(DF_train)
> DF_test <- as.data.frame(DF_test)
```

```
> #modifiying train and test batches
```

```
> DF_train <- DF_train[DF_train$time > 0 & DF_train$time <= 502,-which(colnames(DF_train) %in
% c("time", "batch"))]
> DF_test <- DF_test[DF_test$time > 500 & DF_test$time <= 700,-which(colnames(DF_test) %in% c
("time", "batch"))]


> gc()
         used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   4984368  266.2   10286332  549.4   10286332  549.4
Vcells 490519092 3742.4 1179491595 8998.9 1179491379 8998.9

> head(DF_train)
      signal open_channels     Mn_11    Std_11   Mx_11   Mi_11     Wm_11
99001 -2.7607             0 -2.838145 0.2057949 -2.4243 -3.1449 -2.801317
99002 -2.8480             0 -2.815800 0.2066260 -2.4243 -3.1449 -2.819783
99003 -2.4243             0 -2.799473 0.2169495 -2.4243 -3.1449 -2.745643
99004 -3.1300             0 -2.785845 0.2203233 -2.4243 -3.1449 -2.878600
99005 -3.1449             0 -2.768873 0.2135860 -2.4243 -3.1449 -2.875007
99006 -2.6499             0 -2.798482 0.2360379 -2.4243 -3.1449 -2.761373
        Mn_26    Std_26   Mx_26   Mi_26     Wm_26     Mn_51    Std_51
99001 -2.656204 0.3800587 -1.5989 -3.1449 -2.761150 -2.641963 0.4546064
99002 -2.701538 0.3133384 -1.7197 -3.1449 -2.781579 -2.649571 0.4553045
99003 -2.747892 0.2436836 -2.0320 -3.1449 -2.758762 -2.665257 0.4599054
99004 -2.759177 0.2446672 -2.0320 -3.1449 -2.820347 -2.666990 0.4597742
99005 -2.784858 0.1953618 -2.3962 -3.1449 -2.823323 -2.665245 0.4591451
99006 -2.779150 0.2064649 -2.3320 -3.1449 -2.784099 -2.682690 0.4622729
       Mx_51   Mi_51     Wm_51    Mn_101   Std_101  Mx_101  Mi_101    Wm_101
99001 -1.3135 -3.1889 -2.647373 -2.552321 0.5312239 -1.0974 -3.1998 -2.627265
99002 -1.3135 -3.1889 -2.664884 -2.565490 0.5293765 -1.0974 -3.1998 -2.639293
99003 -1.3135 -3.1890 -2.662085 -2.572870 0.5246996 -1.0974 -3.1998 -2.641035
99004 -1.3135 -3.1890 -2.704153 -2.580395 0.5148677 -1.0974 -3.1998 -2.664642
99005 -1.3135 -3.1890 -2.718558 -2.588039 0.5076603 -1.0974 -3.1998 -2.673820
99006 -1.3135 -3.1998 -2.711512 -2.597642 0.4968815 -1.0974 -3.1998 -2.672119
       Mn_1001  Std_1001 Mx_1001 Mi_1001   Wm_1001       R_W      R_Wt
99001 -2.634075 0.3740345 -0.8661 -3.3863 -2.585678 -2.310664 -2.572372
99002 -2.633807 0.3741414 -0.8661 -3.3863 -2.586153 -2.325129 -2.572797
99003 -2.633869 0.3741563 -0.8661 -3.3863 -2.585606 -2.330986 -2.572376
99004 -2.633843 0.3741450 -0.8661 -3.3863 -2.587316 -2.350705 -2.573366
99005 -2.633145 0.3739492 -0.8661 -3.3863 -2.587646 -2.370364 -2.574385
99006 -2.632637 0.3735969 -0.8661 -3.3863 -2.586955 -2.379881 -2.574415
          L_W      L_Wt      L1      L2      L3      F1      F2      F3
99001 -2.778197 -2.702843 -2.7758 -2.9490 -2.8184 -2.8480 -2.4243 -3.1300
99002 -2.777631 -2.702707 -2.7607 -2.7758 -2.9490 -2.4243 -3.1300 -3.1449
99003 -2.775344 -2.702396 -2.8480 -2.7607 -2.7758 -3.1300 -3.1449 -2.6499
```

```
99004 -2.781475 -2.702932 -2.4243 -2.8480 -2.7607 -3.1449 -2.6499 -2.6971
99005 -2.773506 -2.702058 -3.1300 -2.4243 -2.8480 -2.6499 -2.6971 -2.5961
99006 -2.765079 -2.701152 -3.1449 -3.1300 -2.4243 -2.6971 -2.5961 -2.6685
        dL1     dF1  sigAb    SigSq    sigSqR signal_M1001 signal_M101
99001  0.0151 -0.0873 2.7607 7.621464 -1.661535  -0.12662478 -0.20837921
99002 -0.0873  0.4237 2.8480 8.111104 -1.687602  -0.21419271 -0.28250990
99003  0.4237 -0.7057 2.4243 5.877230 -1.557016   0.20956943  0.14857030
99004 -0.7057 -0.0149 3.1300 9.796900 -1.769181  -0.49615704 -0.54960495
99005 -0.0149  0.4950 3.1449 9.890396 -1.773387  -0.51175534 -0.55686139
99006  0.4950 -0.0472 2.6499 7.021970 -1.627851  -0.01726314 -0.05225842
     signal75 signal25 signalMax signalMin     UL DD
99001  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
99002  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
99003  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
99004  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
99005  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
99006  -2.5225  -2.8486   -0.7027   -3.7003 2.5202  1
> head(DF_test)
       signal     Mn_11    Std_11   Mx_11   Mi_11     Wm_11     Mn_26
199002 -2.6513 -2.414482 0.9801575  0.4430 -3.1217 -2.245577 -2.675938
199003 -2.8466 -2.391718 0.9683786  0.4430 -3.1217 -2.380763 -2.688377
199004 -2.8538 -2.360591 0.9478533  0.4430 -2.8538 -2.490090 -2.673658
199005 -2.4438 -2.373927 0.9540208  0.4430 -2.8538 -2.522063 -2.664635
199006 -2.6125 -2.389182 0.9527086  0.4430 -2.8538 -2.656350 -2.639262
199007 -2.5692 -2.695473 0.1765548 -2.3909 -2.9262 -2.646067 -2.641788
        Std_26 Mx_26   Mi_26     Wm_26     Mn_51    Std_51 Mx_51   Mi_51
199002 0.6796268 0.443 -3.2372 -2.528274 -2.647561 0.6004562 0.443 -3.2372
199003 0.6833651 0.443 -3.2372 -2.551806 -2.671402 0.5649698 0.443 -3.2372
199004 0.6841882 0.443 -3.2372 -2.562834 -2.696433 0.5306135 0.443 -3.2372
199005 0.6843740 0.443 -3.2372 -2.543218 -2.710657 0.5201437 0.443 -3.2372
199006 0.6811196 0.443 -3.2372 -2.569642 -2.709712 0.5201186 0.443 -3.2372
199007 0.6827336 0.443 -3.2372 -2.582297 -2.700159 0.5162966 0.443 -3.2372
         Wm_51    Mn_101   Std_101 Mx_101  Mi_101    Wm_101   Mn_1001
199002 -2.666142 -2.468689 0.6175769  0.443 -3.2372 -2.571991 -2.635480
199003 -2.674992 -2.476621 0.6140215  0.443 -3.2372 -2.583301 -2.635175
199004 -2.673582 -2.483087 0.6060778  0.443 -3.2372 -2.590263 -2.635818
199005 -2.654339 -2.493983 0.5932130  0.443 -3.2372 -2.588101 -2.635543
199006 -2.658196 -2.502803 0.5913752  0.443 -3.2372 -2.597044 -2.635609
199007 -2.654863 -2.504142 0.5918794  0.443 -3.2372 -2.601929 -2.635685
       Std_1001 Mx_1001 Mi_1001   Wm_1001       R_W      R_Wt      L_W
199002 0.3698454   0.443 -3.3789 -2.584994 -2.215690 -2.578807 -2.662793
199003 0.3698547   0.443 -3.3789 -2.585635 -2.230520 -2.579223 -2.663407
199004 0.3696981   0.443 -3.3789 -2.585897 -2.245486 -2.579652 -2.660101
```

```
199005 0.3697702   0.443 -3.3789 -2.585326 -2.252022 -2.579261 -2.656659
199006 0.3697813   0.443 -3.3789 -2.585911 -2.261717 -2.579207 -2.661386
199007 0.3698415   0.443 -3.3789 -2.586074 -2.270423 -2.579068 -2.662811
          L_Wt      L1      L2      L3      F1      F2      F3     dL1
199002 -2.703397  0.4430 -2.2231 -2.6937 -2.8466 -2.8538 -2.4438 -3.0943
199003 -2.703496 -2.6513  0.4430 -2.2231 -2.8538 -2.4438 -2.6125 -0.1953
199004 -2.703204 -2.8466 -2.6513  0.4430 -2.4438 -2.6125 -2.5692 -0.0072
199005 -2.702897 -2.8538 -2.8466 -2.6513 -2.6125 -2.5692 -2.7362  0.4100
199006 -2.703409 -2.4438 -2.8538 -2.8466 -2.5692 -2.7362 -2.7793 -0.1687
199007 -2.703584 -2.6125 -2.4438 -2.8538 -2.7362 -2.7793 -2.8404  0.0433
         dF1  sigAb    SigSq    sigSqR signal_M1001 signal_M101 signal75
199002 -0.1953 2.6513 7.029392 -1.628281  -0.01581998 -0.18261089  -2.4875
199003 -0.0072 2.8466 8.103132 -1.687187  -0.21142527 -0.36997921  -2.4875
199004  0.4100 2.8538 8.144174 -1.689319  -0.21798162 -0.37071287  -2.4875
199005 -0.1687 2.4438 5.972158 -1.563266   0.19174346  0.05018317  -2.4875
199006  0.0433 2.6125 6.825156 -1.616323   0.02310929 -0.10969703  -2.4875
199007 -0.1670 2.5692 6.600789 -1.602872   0.06648501 -0.06505842  -2.4875
      signal25 signalMax signalMin     UL DD
199002  -2.8366     0.443   -3.7794 3.8219  1
199003  -2.8366     0.443   -3.7794 3.8219  1
199004  -2.8366     0.443   -3.7794 3.8219  1
199005  -2.8366     0.443   -3.7794 3.8219  1
199006  -2.8366     0.443   -3.7794 3.8219  1
199007  -2.8366     0.443   -3.7794 3.8219  1
> dim(DF_train)
[1] 5020000      50
> dim(DF_test)
[1] 2000000      49
```

Algorithm implementation

```
> #Algorithm implementation
> library(keras)
Warning message:
package ‘keras’ was built under R version 3.6.3
 > y_train <- to_categorical(DF_train$open_channels)


 > #droping open channels
 > DF_train <-DF_train[,-c(which(colnames(DF_train) %in% c("open_channels")))]


 > DF_train <- as.matrix(DF_train)
 > DF_test <- as.matrix(DF_test)
```

Normalizing the data

```
> for(c in 1:ncol(DF_train)){
+   trm <- (DF_train[,c] - mean(c(DF_train[,c], DF_test[,c]))) / sd(c(DF_train[,c], DF_test[,c]))
+   tem <- (DF_test[,c] - mean(c(DF_train[,c], DF_test[,c]))) / sd(c(DF_train[,c], DF_test[,c]))
+
+   DF_train[,c] <- trm
+   DF_test[,c] <- tem
+ }
> rm(tem, trm)
> gc()
          used   (Mb) gc trigger    (Mb)   max used    (Mb)
Ncells  10060270  537.3   23156104  1236.7   11271658   602.0
Vcells 555338419 4236.9 1319258988 10065.2 1319258988 10065.2

> DF_train <- array_reshape(DF_train, c(dim(DF_train), 1))
> gc()
          used   (Mb) gc trigger    (Mb)   max used    (Mb)
Ncells  10060421  537.3   23156104  1236.7   11271658   602.0
Vcells 801318709 6113.6 1319258988 10065.2 1319258988 10065.2

 > DF_test <- array_reshape(DF_test, c(dim(DF_test), 1))
 > gc()
           used   (Mb) gc trigger    (Mb)   max used    (Mb)
 Ncells   7040626  376.1   23156104  1236.7   11271658   602.0
 Vcells 642788871 4904.1 1319258988 10065.2 1319258988 10065.2


 > dim(DF_train)
 [1] 5020000      49       1
 > dim(DF_test)
 [1] 2000000      49       1
```

Epoches calculation:

```
> learning_scheduler = function(epoch, lr) {
+   if (epoch < 25) {
+     return(.001-.00002*epoch)
+   } else if(epoch >= 25 & epoch < 35){
+     return(.0012-.00002*epoch)
+   } else if(epoch >= 35 & epoch < 40){
+     return(.0013-.00002*epoch)
+   } else if(epoch >= 40 & epoch < 50){
+     return(.0014-.00002*epoch)
```

```
+   } else {
+     return(.0015-.00002*epoch)
+   }
+ }
> model <- keras_model_sequential()
> model %>%layer_conv_1d(filters = 10,kernel_size = 8,strides = 1,activation='relu',padding = "same"
, input_shape = c(dim(DF_train)[2],1)) %>%
+   layer_conv_1d(filters = 15,kernel_size = 6,dilation_rate = 8,activation='relu',padding = "same") %>
%
+   layer_conv_1d(filters = 20,kernel_size = 3,dilation_rate = 6,activation='relu',padding = "same") %>
%
+   layer_flatten() %>%
+
+   layer_dense(units = 96,activation = 'relu'
+               #,regularizer_l1_l2(l1 = .01, l2 = .0001)
+   ) %>%
+   layer_dropout(rate = 0.05) %>%
+   layer_dense(units = 32,
+               activation = 'relu'
+   ) %>%
+   layer_dropout(rate = 0.025) %>%
+   layer_dense(units = 11, activation = 'softmax')
```

```
> summary(model)
Model: "sequential_1"
_____________________________________________________________________________
Layer (type)                 Output Shape              Param #
=============================================================================
=======
conv1d_4 (Conv1D)            (None, 49, 10)            90
_____________________________________________________________________________
conv1d_5 (Conv1D)            (None, 49, 15)            915
_____________________________________________________________________________
conv1d_6 (Conv1D)            (None, 49, 20)            920
_____________________________________________________________________________
flatten_1 (Flatten)          (None, 980)               0
_____________________________________________________________________________
dense_3 (Dense)              (None, 96)                94176
_____________________________________________________________________________
dropout_2 (Dropout)          (None, 96)                0
_____________________________________________________________________________
dense_4 (Dense)              (None, 32)                3104
```

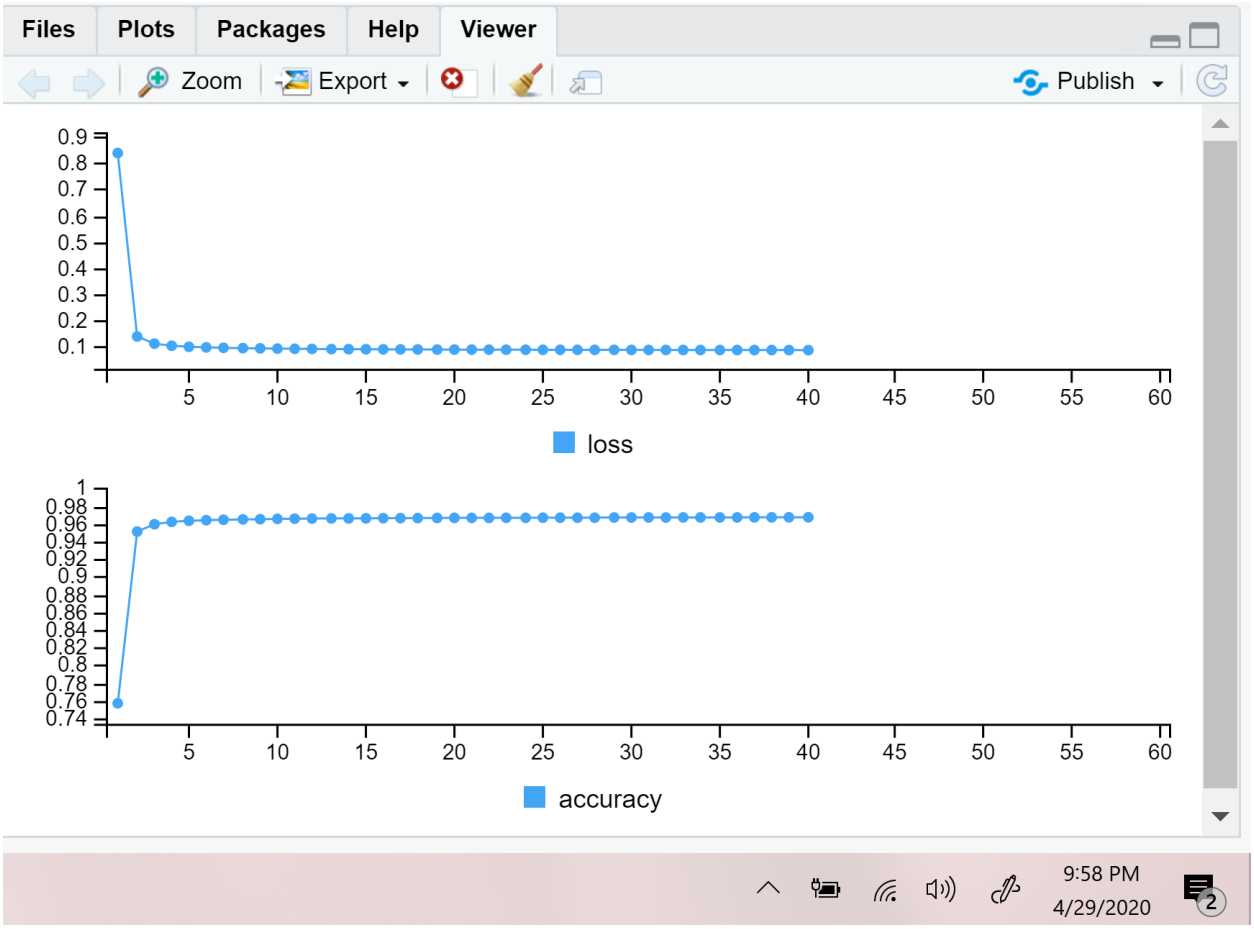
```
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 11)                363
=================================================================
=======
Total params: 99,568
Trainable params: 99,568
Non-trainable params: 0
_________________________________________________________________


> model %>% compile(
+   loss = 'categorical_crossentropy',
+   optimizer = optimizer_adam(beta_1 = .9, beta_2 = .99),
+   metrics = c('accuracy')
+ )
> gc()
          used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   5025340  268.4   17031806  909.6   13071441  698.1
Vcells 535990849 4089.3 1275985808 9735.0 1063254840 8112.0
> history <- model %>% fit(
+   DF_train, y_train,
+   epochs = 60,
+   batch_size = 50000,
+   callbacks = callback_learning_rate_scheduler(learning_scheduler)
+ )
 Train on 5020000 samples
 Epoch 1/60
 5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.8390 - acc
 uracy: 0.7569
 Epoch 2/60
 5020000/5020000 [==============================] - 326s 65us/sample - loss: 0.1364 - acc
 uracy: 0.9514
 Epoch 3/60
 5020000/5020000 [==============================] - 325s 65us/sample - loss: 0.1091 - acc
 uracy: 0.9597
 Epoch 4/60
 5020000/5020000 [==============================] - 318s 63us/sample - loss: 0.1010 - acc
 uracy: 0.9624
 Epoch 5/60
```

5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.0971 - acc
uracy: 0.9636
Epoch 6/60
5020000/5020000 [==============================] - 321s 64us/sample - loss: 0.0948 - acc
uracy: 0.9643
Epoch 7/60
5020000/5020000 [==============================] - 321s 64us/sample - loss: 0.0931 - acc
uracy: 0.9648
Epoch 8/60
5020000/5020000 [==============================] - 319s 63us/sample - loss: 0.0919 - acc
uracy: 0.9652
Epoch 9/60
5020000/5020000 [==============================] - 318s 63us/sample - loss: 0.0910 - acc
uracy: 0.9654
Epoch 10/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0900 - acc
uracy: 0.9658
Epoch 11/60
5020000/5020000 [==============================] - 324s 65us/sample - loss: 0.0894 - acc
uracy: 0.9660
Epoch 12/60
5020000/5020000 [==============================] - 319s 64us/sample - loss: 0.0888 - acc
uracy: 0.9661
Epoch 13/60
5020000/5020000 [==============================] - 323s 64us/sample - loss: 0.0883 - acc
uracy: 0.9663
Epoch 14/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0880 - acc
uracy: 0.9663
Epoch 15/60
5020000/5020000 [==============================] - 311s 62us/sample - loss: 0.0876 - acc
uracy: 0.9664
Epoch 16/60
5020000/5020000 [==============================] - 314s 62us/sample - loss: 0.0873 - acc
uracy: 0.9666
Epoch 17/60
5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.0871 - acc
uracy: 0.9666
Epoch 18/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0869 - acc
uracy: 0.9667
Epoch 19/60

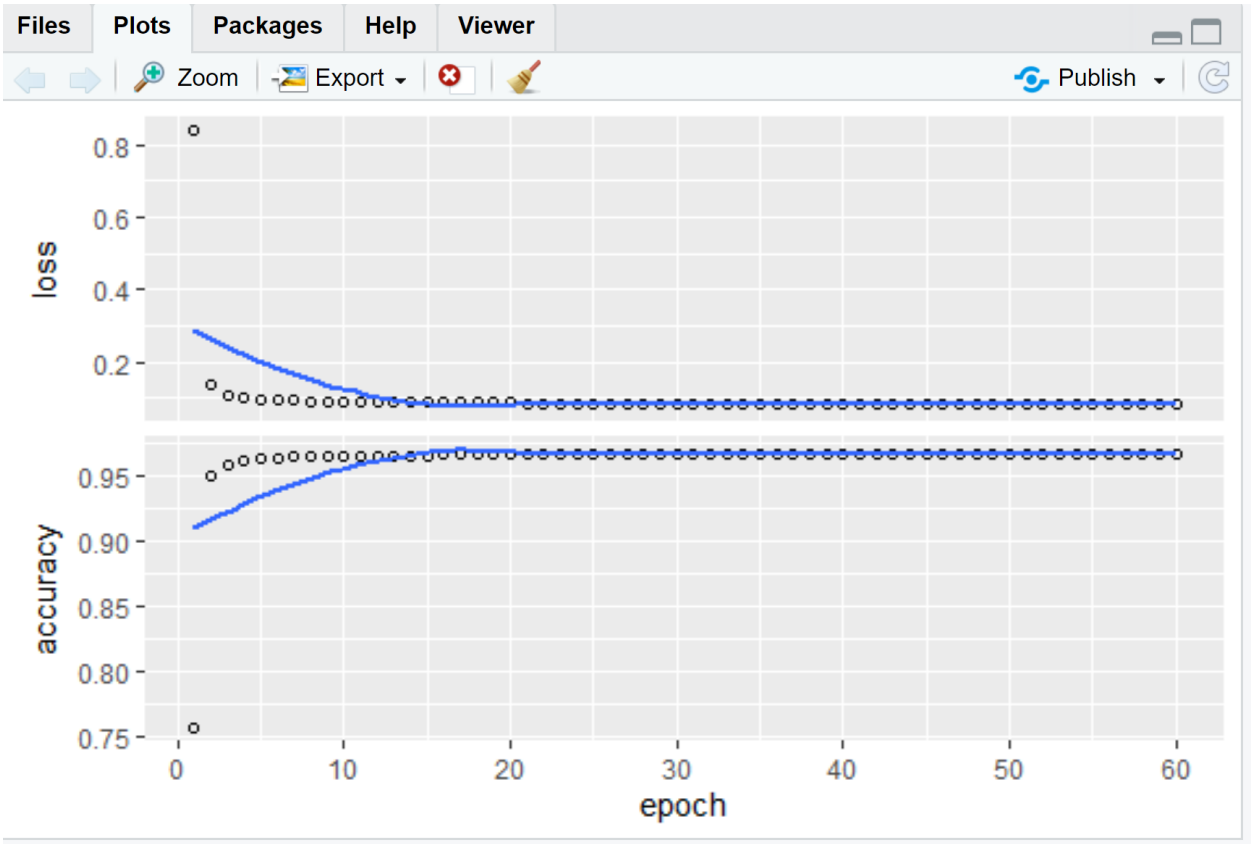
5020000/5020000 [==============================] - 320s 64us/sample - loss: 0.0865 - acc
uracy: 0.9668
Epoch 20/60
5020000/5020000 [==============================] - 319s 64us/sample - loss: 0.0864 - acc
uracy: 0.9669
Epoch 21/60
5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.0863 - acc
uracy: 0.9669
Epoch 22/60
5020000/5020000 [==============================] - 319s 63us/sample - loss: 0.0861 - acc
uracy: 0.9669
Epoch 23/60
5020000/5020000 [==============================] - 343s 68us/sample - loss: 0.0859 - acc
uracy: 0.9670
Epoch 24/60
5020000/5020000 [==============================] - 367s 73us/sample - loss: 0.0858 - acc
uracy: 0.9670
Epoch 25/60
5020000/5020000 [==============================] - 354s 70us/sample - loss: 0.0856 - acc
uracy: 0.9671
Epoch 26/60
5020000/5020000 [==============================] - 342s 68us/sample - loss: 0.0856 - acc
uracy: 0.9671
Epoch 27/60
5020000/5020000 [==============================] - 351s 70us/sample - loss: 0.0855 - acc
uracy: 0.9671
Epoch 28/60
5020000/5020000 [==============================] - 351s 70us/sample - loss: 0.0854 - acc
uracy: 0.9671
Epoch 29/60
5020000/5020000 [==============================] - 322s 64us/sample - loss: 0.0853 - acc
uracy: 0.9673
Epoch 30/60
5020000/5020000 [==============================] - 314s 62us/sample - loss: 0.0851 - acc
uracy: 0.9673
Epoch 31/60
5020000/5020000 [==============================] - 318s 63us/sample - loss: 0.0850 - acc
uracy: 0.9674
Epoch 32/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0850 - acc
uracy: 0.9673
Epoch 33/60

5020000/5020000 [==============================] - 318s 63us/sample - loss: 0.0848 - acc
uracy: 0.9674
Epoch 34/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0848 - acc
uracy: 0.9674
Epoch 35/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0847 - acc
uracy: 0.9674
Epoch 36/60
5020000/5020000 [==============================] - 313s 62us/sample - loss: 0.0847 - acc
uracy: 0.9674
Epoch 37/60
5020000/5020000 [==============================] - 317s 63us/sample - loss: 0.0846 - acc
uracy: 0.9675
Epoch 38/60
5020000/5020000 [==============================] - 318s 63us/sample - loss: 0.0845 - acc
uracy: 0.9675
Epoch 39/60
5020000/5020000 [==============================] - 313s 62us/sample - loss: 0.0845 - acc
uracy: 0.9675
Epoch 40/60
5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.0842 - acc
uracy: 0.9676
Epoch 41/60
5020000/5020000 [==============================] - 553s 110us/sample - loss: 0.0844 - ac
curacy: 0.9675
Epoch 42/60
5020000/5020000 [==============================] - 299s 60us/sample - loss: 0.0842 - acc
uracy: 0.9676
Epoch 43/60
5020000/5020000 [==============================] - 305s 61us/sample - loss: 0.0842 - acc
uracy: 0.9676
Epoch 44/60
5020000/5020000 [==============================] - 304s 61us/sample - loss: 0.0840 - acc
uracy: 0.9676
Epoch 45/60
5020000/5020000 [==============================] - 310s 62us/sample - loss: 0.0840 - acc
uracy: 0.9677
Epoch 46/60
5020000/5020000 [==============================] - 304s 60us/sample - loss: 0.0838 - acc
uracy: 0.9677
Epoch 47/60

5020000/5020000 [==============================] - 306s 61us/sample - loss: 0.0838 - acc
uracy: 0.9677
Epoch 48/60
5020000/5020000 [==============================] - 453s 90us/sample - loss: 0.0839 - acc
uracy: 0.9677
Epoch 49/60
5020000/5020000 [==============================] - 454s 90us/sample - loss: 0.0836 - acc
uracy: 0.9678
Epoch 50/60
5020000/5020000 [==============================] - 313s 62us/sample - loss: 0.0837 - acc
uracy: 0.9677
Epoch 51/60
5020000/5020000 [==============================] - 445s 89us/sample - loss: 0.0838 - acc
uracy: 0.9677
Epoch 52/60
5020000/5020000 [==============================] - 306s 61us/sample - loss: 0.0836 - acc
uracy: 0.9677
Epoch 53/60
5020000/5020000 [==============================] - 294s 59us/sample - loss: 0.0835 - acc
uracy: 0.9678
Epoch 54/60
5020000/5020000 [==============================] - 299s 59us/sample - loss: 0.0836 - acc
uracy: 0.9679
Epoch 55/60
5020000/5020000 [==============================] - 297s 59us/sample - loss: 0.0835 - acc
uracy: 0.9678
Epoch 56/60
5020000/5020000 [==============================] - 889s 177us/sample - loss: 0.0833 - ac
curacy: 0.9679
Epoch 57/60
5020000/5020000 [==============================] - 309s 62us/sample - loss: 0.0834 - acc
uracy: 0.9678
Epoch 58/60
5020000/5020000 [==============================] - 315s 63us/sample - loss: 0.0834 - acc
uracy: 0.9679
Epoch 59/60
5020000/5020000 [==============================] - 307s 61us/sample - loss: 0.0832 - acc
uracy: 0.9679
Epoch 60/60
5020000/5020000 [==============================] - 316s 63us/sample - loss: 0.0832 - acc
uracy: 0.9679




```
> gc()
         used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   5054734  270.0   17031806  909.6   17031806  909.6
Vcells 536240697 4091.2 1275985808 9735.0 1063254840 8112.0

> plot(history)
`geom_smooth()` using formula 'y ~ x'
```




```
> #changing to data frame
> predictKeras <- as.data.frame(predictKeras)
> predictKeras <- predictKeras
```

Reading the sample file from the website

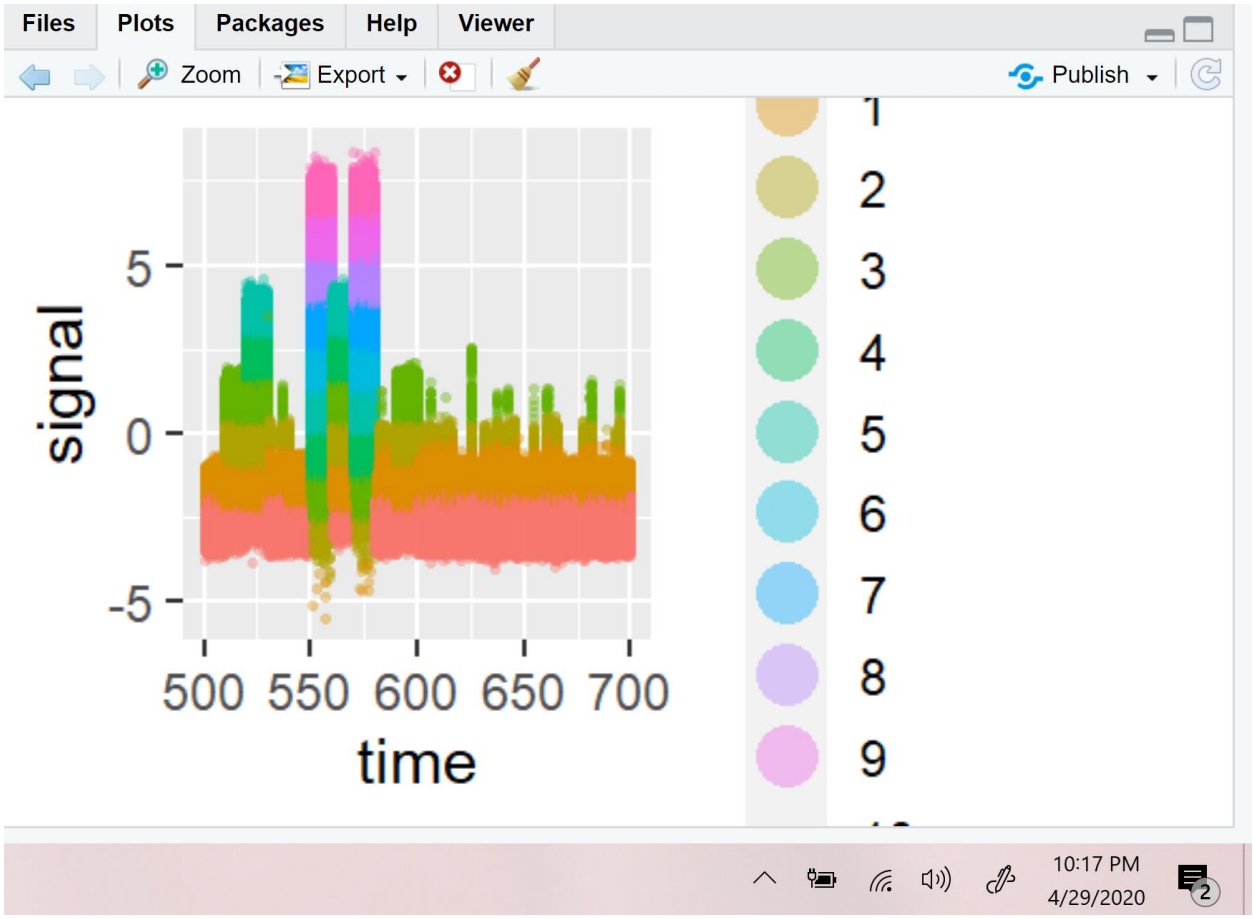
```
> library(data.table)
> sample_submission <- fread("sample_submission.csv", colClasses = "character")
```

Writing the data intosubmission file

```
 > sample_submission$open_channels <- predictKeras
 > fwrite(sample_submission, "submission.csv")
```

```
 > gc()
          used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   7312205  390.6   17031806  909.6   17031806  909.6
Vcells 547230755 4175.1 1275985808 9735.0 1063254840 8112.0
```

```
> test <- fread("testk.csv")
> library(ggplot2)
RStudio Community is a great place to get help:
https://community.rstudio.com/c/tidyverse
> sample_submission$open_channels <- as.factor(sample_submission$open_channels)
> sample_submission$time <- test$time[test$time > 500 & test$time <= 700]
> sample_submission$signal <- test$signal[test$time > 500 & test$time <= 700]
> options(repr.plot.width = 15, repr.plot.height = 10)
> ggplot(sample_submission, aes(x=time, y=signal)) +
+   geom_point(shape=16, aes(color = open_channels), alpha = 0.4) +
+   theme_grey(base_size = 22) +
+   guides(colour = guide_legend(override.aes = list(size=10)))
```




```
> gc()
          used   (Mb) gc trigger   (Mb)   max used   (Mb)
Ncells   5427614  289.9   17031806  909.6   17031806  909.6
Vcells 574852368 4385.8 1275985808 9735.0 1170114783 8927.3
```

REFERENCES:

https://towardsdatascience.com/how-wavenet-works-12e2420ef386

https://neurosciencenews.com/vrac-ion-channel-virus-16144/

https://medium.com/@evinpinar/wavenet-implementation-and-experiments-2d2ee57105d5

https://www.kaggle.com/kei96kag/ion-switching-expriment-2-lag-and-lead-features

https://www.kaggle.com/c/liverpool-ion-switching/discussion/146268

https://www.kaggle.com/frankmollard/rf-weighted-features-kalman

https://www.kaggle.com/friedchips/clean-removal-of-data-drift