Rの実習課題

情報学群情報科学類 3 年 江畑 拓哉 (201611350)

1 レポート課題 1.1

排他的論理和を誤差逆伝搬法で学習し、以下の問いに答えなさい。
この課題は試行ごとにばらつきがあるため、ここに書かれた結果を直ちに再現できるとは限らない。
以下にこの課題を実行した R のファイルを示し、その後それぞれの問いについて答える。

```
# R 3.5.1 で実行確認
# import libraries
library(nnet)
library(MASS)

# レポート課題 1.1
# 1)
# 隠れ素子の数を一つづつ増やし、
# 10 回の学習で 10 回とも正しく識別出来るようになった隠れ素子の数を求めなさい。
# 2)
# 隠れ素子の数によって誤識別率の平均がどのように変化するのかをグラフで示しなさい。
# 3)
# 隠れ素子が 1 個の場合に得られた学習結果について、結合係数の大きさの分布を示しなさい。
# 4)
# 10 回とも正しく識別できた場合の学習結果について、結合係数の大きさの分布を示し、
# 隠れ素子が 1 個の場合と比較検討しなさい。
hidden = c(1:40)
iter = c(1:10)
trave = rep(0, length(hidden))
res_s = c()
tr = matrix(0, length(iter), length(hidden))
z = 0
for (i in 1:length(hidden)) {
  for (j in 1:length(iter)) {
```

```
  res_s = c(list(nnet(classes~., data=xor, size=hidden[i], rang=0.1)), res_s)
  out = predict(res_s[i][[1]], xor, type="class")
  tr[j, i] = mean(out != xor$classes)
  }
  trave[i] = mean(tr[,i])
  if(trave[i] == 0.0 && z == 0) {
    z = i
  }
}
res_s = rev(res_s)
# 1) 37
z
# 2) 1-1-2.png
plot(hidden, trave, type="b", lty=1, lwd=2)
# 3) 1-1-3.png
hist(res_s[1][[1]]$wts, breaks=seq(-0.1, 0.1, 0.04), freq=TRUE)
# 4) 1-1-4.png
hist(res_s[z * 10][[1]]$wts, breaks=seq(-0.1, 0.1, 0.04), freq=TRUE)
```

1.1 隠れ素子を一つずつ増やし、**10** 回の学習で **10** 回とも正しく識別できるようになった隠れ素子の数を求めなさい。

37 個

1.2 隠れ素子の数によって誤認識率の平均がどのように変化するのかをグラフで示しなさい。

1.3 隠れ素子が **1** 個の場合に得られた一つの学習結果について、結合係数の大きさの分布を示しなさい。

1.4 **10** 回とも正しく識別できた場合の一つの学習結果について、結合係数の大きさの分布を示し、隠れ素子が **1** 個の場合と比較検討しなさい。

隠れ素子が 1 個の場合は結合係数の数が少ないので断言することはできないが、0 近傍に点が集中していることがわかる。また後者は 0 を中心として山なりに分布していることがわかる。

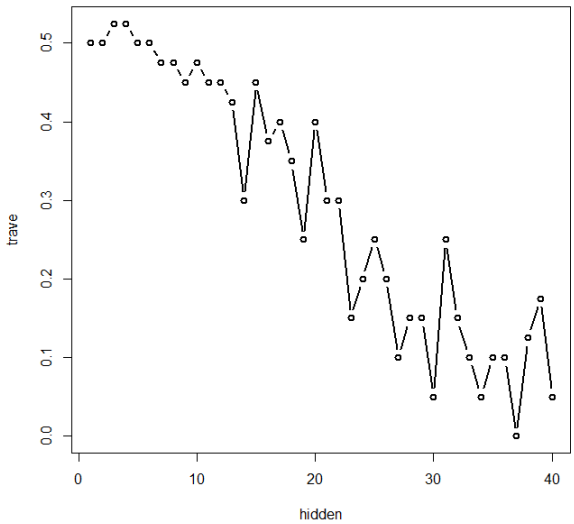


Figure 1: 1-1-2.png 隠れ素子の数と誤認識率の平均のグラフ

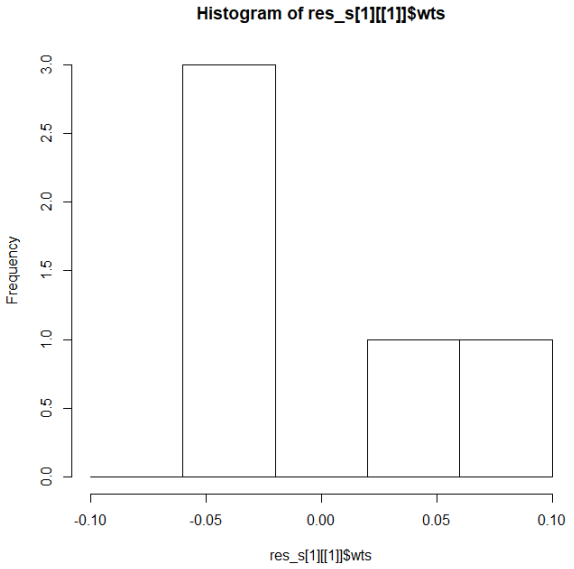


Figure 2: 1-1-3.png 隠れ素子が 1 個の場合の結合係数の大きさの分布

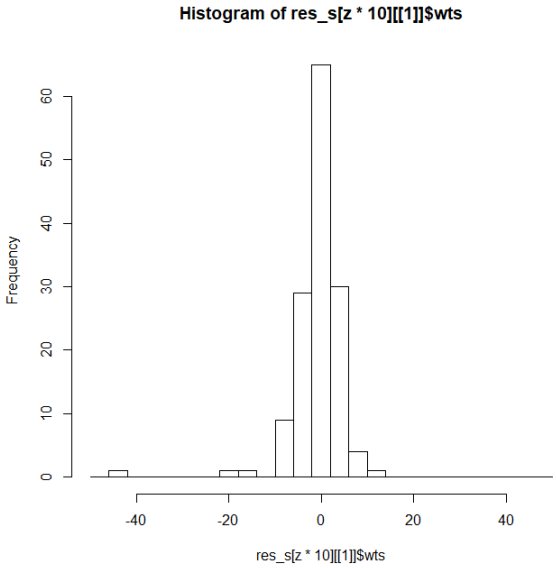


Figure 3: 1-1-4.png 10 回とも正しく識別出来た場合の結合係数の大きさの分布

2 レポート課題 1.2

アヤメデータを用いて誤差逆伝搬法による学習を行い、下記の項目に答えなさい。

以下にこの課題を実行した R のファイルを示し、その後それぞれの問いについて答える。

```
# レポート課題１．２
# データの用意
ir <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
      species=factor(c(rep("sv", 50), rep("c", 50),
      rep("sv", 50))))
samp <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
# 1)
# decay=0 に設定し、隠れ素子の数を１にして学習データを用いて１０回学習し、
# 学習データに対する誤識別率の平均と、テストデータに対する誤識別率の平均を求めなさい。
# 隠れ素子の数を１０まで１つずつ増やして同じことを行い、
# 隠れ素子の数に対する再代入誤りと汎化誤差の変化をグラフ化しなさい。
# 2)
# 再代入誤りが一番小さな場合の結合係数の分布と、
# 汎化誤差が一番小さな場合の結合係数の分布を比較検討しなさい。
# 3)
# decay=0.01 にして同様の実験を行い、
```

```r
# 隠れ素子数に対する再代入誤りと汎化誤差の変化をグラフ化しなさい。
# 4)
# 再代入誤りが一番小さな場合の結合係数の分布と、
# 汎化誤差が一番小さな場合の結合係数の分布を、
# decay=0 の場合と比較しなさい。
hidden =
c(1:10)
iter = c(1:10)
decay = 0
trave_learn = rep(0, length(hidden))
trave_test = rep(0, length(hidden))
res_s = c()
tr_learn = matrix(0, length(iter), length(hidden))
tr_test = matrix(0, length(iter), length(hidden))
for (i in 1:length(hidden)) {
  for (j in 1:length(iter)) {
    res_s = c(list(nnet(species~., data=ir[samp,], size=hidden[i],
              rang=0.5, decay=decay, maxit=200)), res_s)
  out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
  out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
  tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
  tr_test[j, i] = mean(out_test != ir[-samp,]$species)
  }
  trave_learn[i] = mean(tr_learn[, i])
  trave_test[i] = mean(tr_test[, i])
}
res_s = rev(res_s)
# 1) 隠れ素子数が 1 つのときの再代入誤り、汎化誤差
trave_learn[1]
trave_test[1]
# 1) 1-2-1-1.png 再代入誤りの変化
plot(hidden, trave_learn, type="b", lty=1, lwd=2)
# 1) 1-2-1-2.png  汎化誤差の変化
plot(hidden, trave_test, type="b", lty=1, lwd=2)
# 2)
which.min(trave_learn) # 9
which.min(trave_test)  # 10
# 2) 1-2-2-1.png
hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
# 2) 1-2-2-2.png
hist(res_s[which.min(trave_test)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
# 3)
hidden = c(1:10)
iter = c(1:10)
```

```
decay = 0.01
trave_learn = rep(0, length(hidden))
trave_test = rep(0, length(hidden))
res_s = c()
tr_learn = matrix(0, length(iter), length(hidden))
tr_test = matrix(0, length(iter), length(hidden))
for (i in 1:length(hidden)) {
  for (j in 1:length(iter)) {
    res_s = c(list(nnet(species~., data=ir[samp,],
                        size=hidden[i], rang=0.5, decay=decay, maxit=200)), res_s)
  out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
  out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
  tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
  tr_test[j, i] = mean(out_test != ir[-samp,]$species)
  }
  trave_learn[i] = mean(tr_learn[, i])
  trave_test[i] = mean(tr_test[, i])
}
res_s = rev(res_s)
# 3) 1-2-3-1.png
plot(hidden, trave_learn, type="b", lty=1, lwd=2)
# 3) 1-2-3-2.png
plot(hidden, trave_test, type="b", lty=1, lwd=2)
# 4)
which.min(trave_learn) # 4
which.min(trave_test) # 4
# 4) 1-2-4-1.png
hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-6, 6, 1), freq=TRUE)
# 4) 1-2-4-2.png
hist(res_s[which.min(trave_test)][[1]]$wts, breaks=seq(-6, 6, 1), freq=TRUE)
```

2.1 decay=0 に設定し、隠れ素子の数を 1 にして学習データを用いて１０回学習し、学習データに対する誤識別率の平均と、テストデータに対する誤識別率の平均を求めなさい。隠れ素子の数を 10 まで 1 ずつ増やして同じことを行い、隠れ素子の数に対する再代入誤りと汎化誤差の変化をグラフ化しなさい。

- 学習データに対する誤認識率の平均
  0.338667

- テストデータに対する誤認識率の平均
  0.333333

- 再代入誤りの変化のグラフ

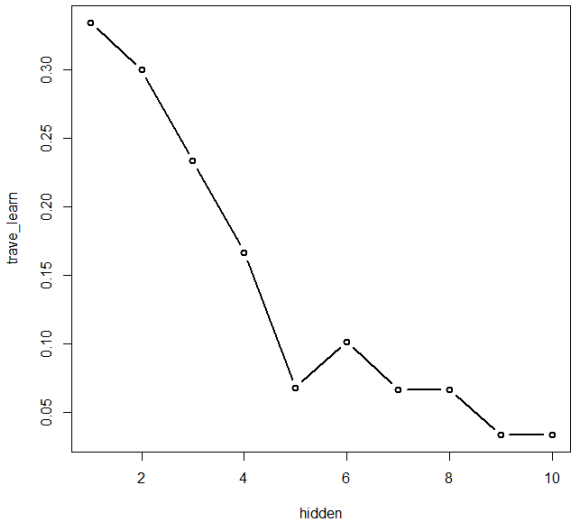


Figure 4: 1-2-1-1.png 再代入誤りの変化のグラフ

- 汎化誤差の変化のグラフ

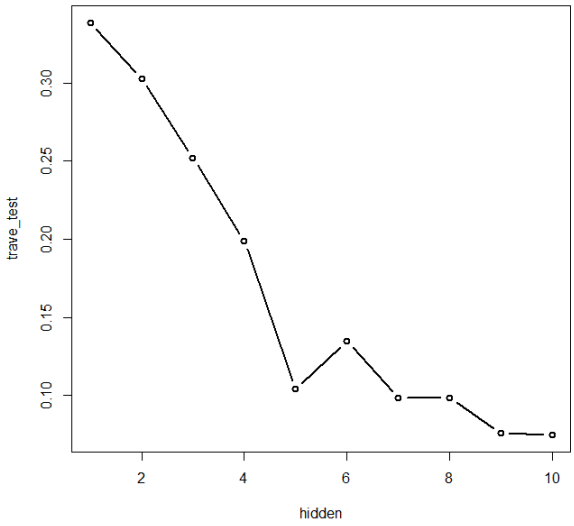


Figure 5: 1-2-1-2.png 汎化誤差の変化のグラフ

2.2 再代入誤りが一番小さな場合の結合係数の分布と、汎化誤差が一番小さな場合の結合係数の分布を比較検討しなさい。

- 再代入誤りが一番小さな場合の結合係数の分布

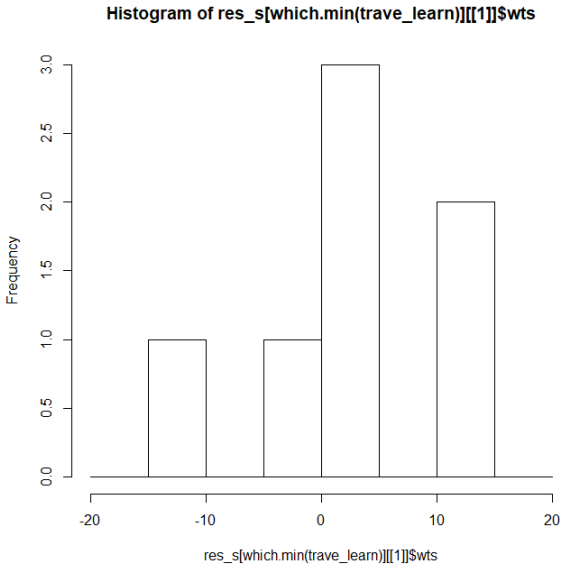


Figure 6: 1-2-2-1.png 再代入誤りが一番小さな場合の結合係数の分布

- 汎化誤差が一番小さな場合の結合係数の分布

- 比較

  ほとんどの試行でこの 2 つが異なることはなかった。また今回のように異なった場合の結合係数の分布も似通った形状をしていることがわかる。これは再代入誤りと汎化誤差、いずれもデータの性質は異なっていないため、極端に結合係数の分布が異なることはないと想像できる。

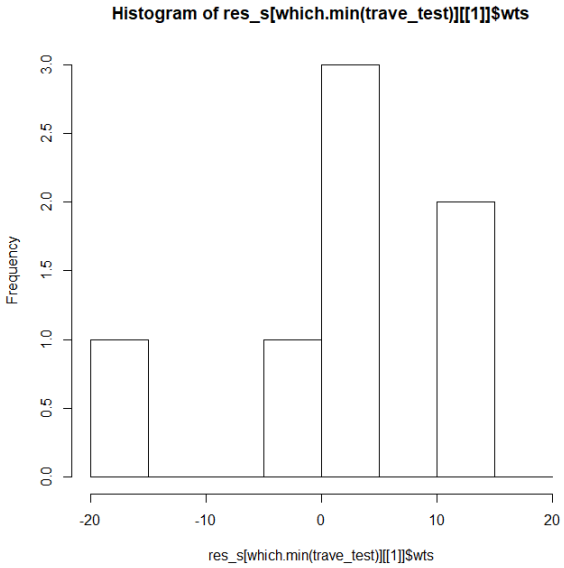


Figure 7: 1-2-2-2.png 汎化誤差が一番小さな場合の結合係数の分布

2.3 decay=0.01 にして同様の実験を行い、隠れ素子数に対する再代入誤りと汎化誤差の変化をグラフ化しなさい。

- 再代入誤りの変化のグラフ

- 汎化誤差の変化のグラフ

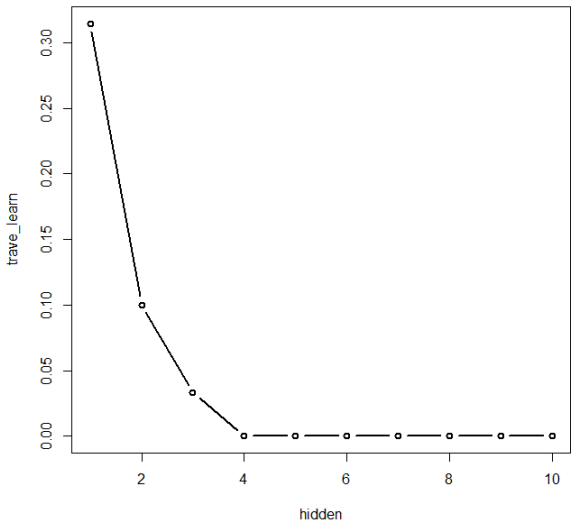


Figure 8: 1-2-3-1.png 再代入誤りの変化のグラフ (decay=0.01)

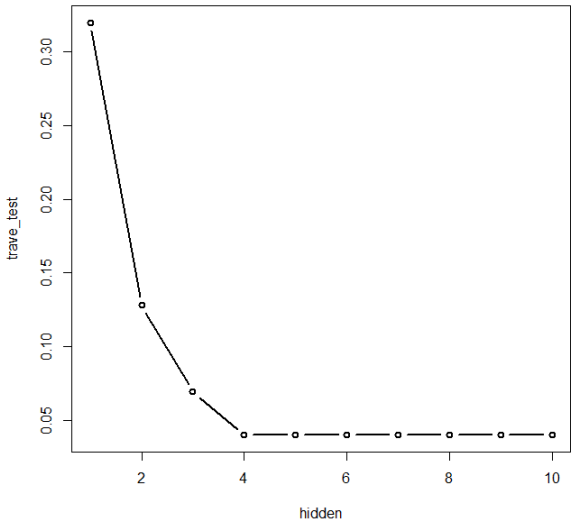
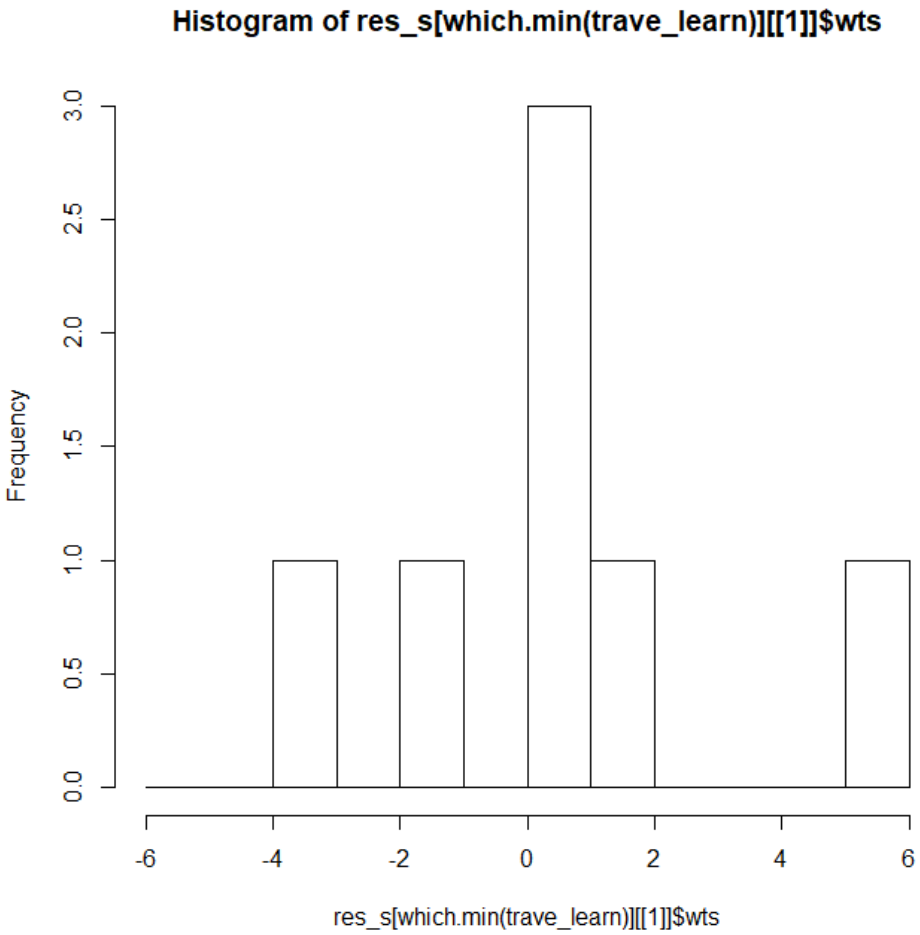


Figure 9: 1-2-3-2.png 汎化誤差の変化のグラフ (decay=0.1)

2.4 再代入誤りが一番小さな場合の結合係数の分布と、汎化誤差が一番小さな場合の結合係数の分布を、**decay=0** の場合と比較しなさい。

- 再代入誤りが一番小さな場合の結合係数の分布



- 汎化誤差が一番小さな場合の結合係数の分布

- 比較
  いずれと比較しても、decay $= 0.01$ の方が分布の幅が縮まっていることが確認できる。また概形としても decay $= 0.01$ の方がより綺麗な (対称な) 山の形を描いていることがわかる。

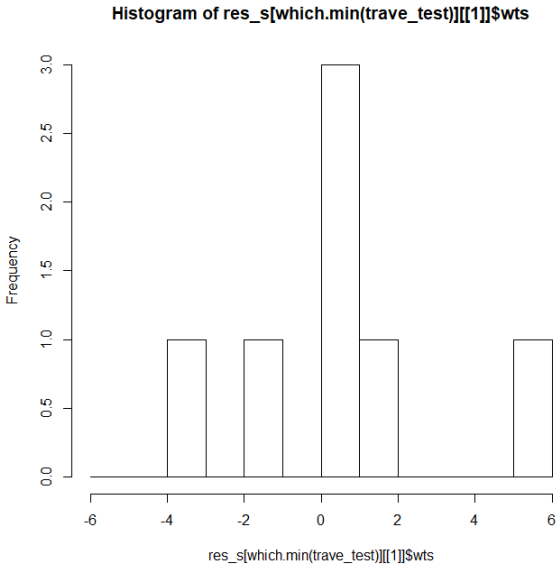


Figure 10: 1-2-4-2.png

3 レポート課題 2.1

例題に従って全結合型 3 層パーセプトロンによる手書き数字認識システムを実装し、下記の問いに答えなさい。

以下にこの課題を実行した R のファイルを示し、その後それぞれの問いについて答える。

```
# R 3.5.1 で実行確認
# import libraries
library(nnet)
library(MASS)
library(mxnet)

# create dataset
train <- read.csv("data/short_prac_train.csv", header = TRUE)
test <- read.csv("data/short_prac_test.csv", header = TRUE)
train <- data.matrix(train) test <- data.matrix(test)
train.x <- train[,-1]
train.y <- train[,1]
test_org <- test
test <- test[,-1]
train.x <- t(train.x/255) # [0, 255] -> [0, 1]
```

```r
test <- t(test/255)
table(train.y)

# check image
image(x=seq(1:28),y=seq(1:28), matrix(train.x[,4], 28, 28)[, 28:1],
      col = gray(0:255/255))

# sample
# network settings
data <- mx.symbol.Variable("data")
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")

# network training
devices <- mx.cpu()
mx.set.seed(0)
model <- mx.model.FeedForward.create(softmax, X = train.x, y = train.y,
                                     initializer = mx.init.uniform(0.07),
                                     ctx = devices,
                                     num.round = 10, array.batch.size = 100,
                                     learning.rate=0.05,
                                     momentum=0.9, wd=0.00001,
                                     eval.metric = mx.metric.accuracy,
                                     epoch.end.callback =
                                       mx.callback.log.train.metric(100))

preds <- predict(model, test, ctx=devices)
pred.label <- max.col(t(preds)) -1
sum(diag(table(test_org[,1], pred.label))) / 1000
table(test_org[,1], pred.label)

# レポート課題２． 1
# 1)
# ３つの異なった乱数の種を用いて、学習データとテストデータに対する認識率を求めなさい。
# 2)
# 最初の２つの隠れ層の非線形出力関数をシグモイド関数 (sigmoid) にした場合、
# 認識率はどのようになるか。
# ReLUの場合と同じ条件で実験し、比較しなさい。

training_mnist <- function(seed, activate_fun) {
```

```
  # network settings
  data <- mx.symbol.Variable("data")
  fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
  act1 <- mx.symbol.Activation(fc1, name="relu1", act_type=activate_fun)
  fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
  act2 <- mx.symbol.Activation(fc2, name="relu2", act_type=activate_fun)
  fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
  softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")

  devices <- mx.cpu()
  mx.set.seed(seed)

  # training network
  model <- mx.model.FeedForward.create(softmax, X = train.x, y = train.y,
                                       initializer = mx.init.uniform(0.07),
                                       ctx = devices,
                                       num.round = 10, array.batch.size = 100,
                                       learning.rate=0.05,
                                       momentum=0.9, wd=0.00001,
                                       eval.metric = mx.metric.accuracy,
                                       epoch.end.callback =
                                         mx.callback.log.train.metric(100))
  preds <- predict(model, test, ctx=devices)
  pred.label <- max.col(t(preds)) -1
  return(mean(test_org[,1] == pred.label))
}

# 1)
seeds = list(11, 25, 2018)

training_mnist(seeds[1][[1]], "relu")

# ------------------------------------------
# [1] Train-accuracy=0.41060000102967
# [2] Train-accuracy=0.813400003910065
# [3] Train-accuracy=0.891999999284744
# [4] Train-accuracy=0.911600003242493
# [5] Train-accuracy=0.937400004863739
# [6] Train-accuracy=0.948400005102158
# [7] Train-accuracy=0.966600004434586
# [8] Train-accuracy=0.974000008106232
# [9] Train-accuracy=0.979200007915497
# [10] Train-accuracy=0.984000010490418
# [1] 0.938
```

```
# ----------------------------------------

training_mnist(seeds[2][[1]], "relu")
# ----------------------------------------
# [1] Train-accuracy=0.426600000560284
# [2] Train-accuracy=0.818000000715256
# [3] Train-accuracy=0.873200000524521
# [4] Train-accuracy=0.902800003290176
# [5] Train-accuracy=0.933199996948242
# [6] Train-accuracy=0.950400000810623
# [7] Train-accuracy=0.961600004434586
# [8] Train-accuracy=0.96960000872612
# [9] Train-accuracy=0.9794000077724762
# [10] Train-accuracy=0.98080001115799
# [1] 0.941
# ----------------------------------------

training_mnist(seeds[3][[1]], "relu")
# ----------------------------------------
# [1] Train-accuracy=0.44320000231266
# [2] Train-accuracy=0.831800000667572
# [3] Train-accuracy=0.890200002193451
# [4] Train-accuracy=0.921600000858307
# [5] Train-accuracy=0.937600003480911
# [6] Train-accuracy=0.949200004339218
# [7] Train-accuracy=0.960400002002716
# [8] Train-accuracy=0.969600001573563
# [9] Train-accuracy=0.971800007820129
# [10] Train-accuracy=0.967400006055832
# [1] 0.931
# ----------------------------------------

# 2)
training_mnist(seeds[1][[1]], "sigmoid")
# ----------------------------------------
# [1] Train-accuracy=0.0967999996244907
# [2] Train-accuracy=0.117599999085069
# [3] Train-accuracy=0.153000000119209
# [4] Train-accuracy=0.275600000321865
# [5] Train-accuracy=0.43559999704361
# [6] Train-accuracy=0.577199996709824
# [7] Train-accuracy=0.684799997806549
# [8] Train-accuracy=0.750999997854233
# [9] Train-accuracy=0.796199997663498
```

```
# [10] Train-accuracy=0.821999995708466
# [1] 0.827
# ----------------------------------------

training_mnist(seeds[2][[1]], "sigmoid")
# ----------------------------------------
# [1] Train-accuracy=0.102400000393391
# [2] Train-accuracy=0.106399999856949
# [3] Train-accuracy=0.132000000178814
# [4] Train-accuracy=0.217799999862909
# [5] Train-accuracy=0.385399999022484
# [6] Train-accuracy=0.526599999666214
# [7] Train-accuracy=0.66940000295639
# [8] Train-accuracy=0.76299999833107
# [9] Train-accuracy=0.807399994134903
# [10] Train-accuracy=0.829199995994568
# [1] 0.84
# ----------------------------------------

training_mnist(seeds[3][[1]], "sigmoid")
# ----------------------------------------
# [1] Train-accuracy=0.0975999997928739
# [2] Train-accuracy=0.106199999824166
# [3] Train-accuracy=0.129200000017881
# [4] Train-accuracy=0.204800001382828
# [5] Train-accuracy=0.416599997282028
# [6] Train-accuracy=0.578999997973442
# [7] Train-accuracy=0.695199999809265
# [8] Train-accuracy=0.7594000017642977
# [9] Train-accuracy=0.807399997711182
# [10] Train-accuracy=0.839199997186661
# [1] 0.837
# ----------------------------------------
```

3.1 3 つの異なった乱数の種を用いて、学習データとテストデータに対する認識率を求めなさい。

乱数の種として、 11, 25, 2018 を用いた。
認識率は以下の通りになった。

	11	25	2018
学習データ	0.984000010490418	0.98080001115799	0.967400006055832
テストデータ	0.938	0.941	0.931

3.2 最初の **2** つの隠れ層の非線形出力関数をシグモイド関数 **(sigmoid)** にした場合、認識率はどのようになるか。**ReLU** の場合と同じ条件で実験し、比較しなさい。

以下の通りになった。

	11	25	2018
ReLU			
学習データ	0.984000010490418	0.98080001115799	0.967400006055832
テストデータ	0.938	0.941	0.931
sigmoid			
学習データ	0.821999995708466	0.829199995994568	0.839199997186661
テストデータ	0.827	0.84	0.837

sigmoid 関数を用いると　ReLU よりもやや精度が低くなったように感じる。しかし、学習データとテストデータの認識率の差を見ると、後者の方が小さいため、より適切なネットワーク構成を考えることができれば、ReLU 以上の精度な汎化性能を得られる可能性があるのかもしれない。

4 レポート課題 3.1

以下にレポート課題 3.1 から 3.6 までを実行した R のファイルを示し、その後それぞれの問いについて答える。

```
# R 3.5.1 で実行確認
# import libraries
library(nnet)
library(MASS)
library(mxnet)

# create dataset
train <- read.csv("data/short_prac_train.csv", header = TRUE)
test <- read.csv("data/short_prac_test.csv", header = TRUE)
train <- data.matrix(train)
test <- data.matrix(test)
train.x <- train[,-1]
train.y <- train[,1]
test_org <- test
test <- test[,-1]
train.x <- t(train.x/255) # [0, 255] -> [0, 1]
test <- t(test/255)
table(train.y)

# input layer
data <- mx.symbol.Variable("data")

# hidden layer 1
conv1 <- mx.symbol.Convolution(data=data, kernel=c(5, 5), num_filter=20)
tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2, 2),
                           stride=c(2, 2))
drop1 <- mx.symbol.Dropout(data=pool1, p=0.5)

# hidden layer 2
conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(5,5), num_filter=50)
tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2, 2),
                           stride=c(2, 2))
drop2 <- mx.symbol.Dropout(data=pool2, p=0.5)

# fully connected layer 1
flatten <- mx.symbol.Flatten(data=drop2)
```

```
fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
drop3 <- mx.symbol.Dropout(data=tanh3, p=0.5)

# fully connected layer 2
fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)

# output layer
lenet <- mx.symbol.SoftmaxOutput(data=fc2)

# preparing train/test data
train.array <- train.x
dim(train.array) <- c(28, 28, 1, ncol(train.x))

test.array <- test
dim(test.array) <- c(28, 28, 1, ncol(test))

# preparing training
mx.set.seed(0)
devices <- mx.cpu()
tic <- proc.time()

# training model
model.CNNtanhDrop <- mx.model.FeedForward.create(lenet, X=train.array,
                                                 y=train.y, ctx=devices,
                                                 num.round = 30,
                                                 array.batch.size = 100,
                                                 learning.rate=0.05,
                                                 momentum=0.9,
                                                 wd=0.000001,
                                                 eval.metric=mx.metric.accuracy,
                                                 batch.end.callback =
                                                   mx.callback.log.train.metric(100))
print(proc.time() - tic)
preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
pred.label <- max.col(t(preds)) -1
sum(diag(table(test_org[,1], pred.label))) / 1000
# 1)
# 1-1)
# M1 : 24
# N1 : 20
# M2 : 12
# N2 : 20
# 1-2)
```

```
# M3 : 8
# N3 : 50
# M4 : 4
# N4 : 50
# 1-3)
# 3次元配列 2次元配列に変換している
# 2)
# ------------------------------------------
# [1] Train-accuracy=0.0943999997526407
# [2] Train-accuracy=0.089199999794364
# [3] Train-accuracy=0.095800000205636
# [4] Train-accuracy=0.353800000697374
# [5] Train-accuracy=0.8151999997425079
# [6] Train-accuracy=0.879399998188019
# [7] Train-accuracy=0.910400002002716
# [8] Train-accuracy=0.919399999380112
# [9] Train-accuracy=0.933800001144409
# [10] Train-accuracy=0.933000004291534
# [11] Train-accuracy=0.939599999189377
# [12] Train-accuracy=0.945799996852875
# [13] Train-accuracy=0.944800004959106
# [14] Train-accuracy=0.945000002384186
# [15] Train-accuracy=0.946200004816055
# [16] Train-accuracy=0.960200003385544
# [17] Train-accuracy=0.956600003242493
# [18] Train-accuracy=0.954999998807907
# [19] Train-accuracy=0.958400005102158
# [20] Train-accuracy=0.961600004434586
# [21] Train-accuracy=0.962000002861023
# [22] Train-accuracy=0.960800007581711
# [23] Train-accuracy=0.964000006914139
# [24] Train-accuracy=0.965400005578995
# [25] Train-accuracy=0.966400007009506
# [26] Train-accuracy=0.968800005912781
# [27] Train-accuracy=0.964800003767014
# [28] Train-accuracy=0.967800005674362
# [29] Train-accuracy=0.965600006580353
# [30] Train-accuracy=0.969400007724762
# [1] 0.986
# ------------------------------------------

training_mnist_cnn = function(dropout, activate_fn) {
  # input layer
  data <- mx.symbol.Variable("data")
```

```
127
128   # hidden layer 1
129   conv1 <- mx.symbol.Convolution(data=data, kernel=c(5, 5), num_filter=20)
130   tanh1 <- mx.symbol.Activation(data=conv1, act_type=activate_fn)
131   pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2, 2),
132                              stride=c(2, 2))
133
134   if (dropout) {
135     drop1 <- mx.symbol.Dropout(data=pool1, p=0.5)
136   } else {
137     drop1 <- pool1
138   }
139
140   # hidden layer 2
141   conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(5,5), num_filter=50)
142   tanh2 <- mx.symbol.Activation(data=conv2, act_type=activate_fn)
143   pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2, 2),
144                              stride=c(2, 2))
145
146   if (dropout) {
147     drop2 <- mx.symbol.Dropout(data=pool2, p=0.5)
148   } else {
149     drop2 <- pool2
150   }
151
152   # fully connected layer 1
153   flatten <- mx.symbol.Flatten(data=drop2)
154   fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
155   tanh3 <- mx.symbol.Activation(data=fc1, act_type=activate_fn)
156
157   if (dropout) {
158     drop3 <- mx.symbol.Dropout(data=tanh3, p=0.5)
159   } else {
160     drop3 <- tanh3
161   }
162
163   # fully connected layer 2
164   fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)
165
166   # output layer
167   lenet <- mx.symbol.SoftmaxOutput(data=fc2)
168
169   # preparing train/test data
170   train.array <- train.x
```

```
171   dim(train.array) <- c(28, 28, 1, ncol(train.x))
172
173   test.array <- test
174   dim(test.array) <- c(28, 28, 1, ncol(test))
175
176   # preparing training
177   mx.set.seed(0)
178   devices <- mx.cpu()
179   tic <- proc.time()
180
181   # training model
182   model.CNNtanhDrop <- mx.model.FeedForward.create(lenet, X=train.array,
183                                         y=train.y, ctx=devices, num.round = 30,
184                                         array.batch.size = 100,
185                                         learning.rate=0.05, momentum=0.9, wd=0.000001,
186                                         eval.metric=mx.metric.accuracy,
187                                         batch.end.callback =
188                                           mx.callback.log.train.metric(100))
189   print(proc.time() - tic)
190   preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
191   pred.label <- max.col(t(preds)) -1
192   sum(diag(table(test_org[,1], pred.label))) / 1000
193 }
194
195 # 3)
196 training_mnist_cnn(FALSE, "tanh")
197 # ------------------------------------------------------------------------
198 # [1] Train-accuracy=0.0951999997347593
199 # [2] Train-accuracy=0.0893999997526407
200 # [3] Train-accuracy=0.0931999997794628
201 # [4] Train-accuracy=0.353799997121096
202 # [5] Train-accuracy=0.841599998474121
203 # [6] Train-accuracy=0.91860000371933
204 # [7] Train-accuracy=0.951199996471405
205 # [8] Train-accuracy=0.96200000166893
206 # [9] Train-accuracy=0.970600011348724
207 # [10] Train-accuracy=0.980000009536743
208 # [11] Train-accuracy=0.984800010919571
209 # [12] Train-accuracy=0.991400008201599
210 # [13] Train-accuracy=0.992800006866455
211 # [14] Train-accuracy=0.994400005340576
212 # [15] Train-accuracy=0.996200003623962
213 # [16] Train-accuracy=0.995200004577637
214 # [17] Train-accuracy=0.996800003051758
```

```
# [18] Train-accuracy=0.998200001716614
# [19] Train-accuracy=0.999400000572205
# [20] Train-accuracy=1
# [21] Train-accuracy=1
# [22] Train-accuracy=1
# [23] Train-accuracy=1
# [24] Train-accuracy=1
# [25] Train-accuracy=1
# [26] Train-accuracy=1
# [27] Train-accuracy=1
# [28] Train-accuracy=1
# [29] Train-accuracy=1
# [30] Train-accuracy=1
# user  system elapsed
# 549.00  251.10  163.95
# [1] 0.982
# ------------------------------------------------------------------------
# comment: over fitting
# 4)
training_mnist_cnn(FALSE, "relu")
# ------------------------------------------------------------------------
# [1] Train-accuracy=0.0957999996095896
# [2] Train-accuracy=0.0893999997526407
# [3] Train-accuracy=0.0903999998420477
# [4] Train-accuracy=0.112800000011921
# [5] Train-accuracy=0.434800001382828
# [6] Train-accuracy=0.876400001049042
# [7] Train-accuracy=0.945999997854233
# [8] Train-accuracy=0.963000003099442
# [9] Train-accuracy=0.964600006341934
# [10] Train-accuracy=0.971200007200241
# [11] Train-accuracy=0.978200010061264
# [12] Train-accuracy=0.987200009822846
# [13] Train-accuracy=0.989800009727478
# [14] Train-accuracy=0.991600008010864
# [15] Train-accuracy=0.992000007629395
# [16] Train-accuracy=0.996000003814697
# [17] Train-accuracy=0.997800002098084
# [18] Train-accuracy=0.999400000572205
# [19] Train-accuracy=0.999400000572205
# [20] Train-accuracy=0.998400001525879
# [21] Train-accuracy=0.999800000190735
# [22] Train-accuracy=0.999000000953674
# [23] Train-accuracy=0.999400000572205
```

```
# [24] Train-accuracy=0.999800000190735
# [25] Train-accuracy=0.999800000190735
# [26] Train-accuracy=0.999800000190735
# [27] Train-accuracy=1
# [28] Train-accuracy=1
# [29] Train-accuracy=1
# [30] Train-accuracy=1
# user  system elapsed
# 513.99  243.98  154.81
# [1] 0.982
# ------------------------------------------------------------------
# 5)
training_mnist_cnn(TRUE, "relu")
# [1] Train-accuracy=0.0949999997764826
# [2] Train-accuracy=0.0893999997526407
# [3] Train-accuracy=0.0899999997764826
# [4] Train-accuracy=0.100199999809265
# [5] Train-accuracy=0.359799997210503
# [6] Train-accuracy=0.774200001955032
# [7] Train-accuracy=0.879999998807907
# [8] Train-accuracy=0.90300000667572
# [9] Train-accuracy=0.920800005197525
# [10] Train-accuracy=0.927600004673004
# [11] Train-accuracy=0.94440000295639
# [12] Train-accuracy=0.945800001621246
# [13] Train-accuracy=0.945199998617172
# [14] Train-accuracy=0.949000002145767
# [15] Train-accuracy=0.953400001525879
# [16] Train-accuracy=0.960600000619888
# [17] Train-accuracy=0.955800002813339
# [18] Train-accuracy=0.963600004911423
# [19] Train-accuracy=0.960200004577637
# [20] Train-accuracy=0.964400005340576
# [21] Train-accuracy=0.966000009775162
# [22] Train-accuracy=0.967400008440018
# [23] Train-accuracy=0.964800004959106
# [24] Train-accuracy=0.968200006484985
# [25] Train-accuracy=0.969400004148483
# [26] Train-accuracy=0.970800008773804
# [27] Train-accuracy=0.971400009393692
# [28] Train-accuracy=0.966400008201599
# [29] Train-accuracy=0.968200007677078
# [30] Train-accuracy=0.972400006055832
# user  system elapsed
```

```
# 543.69  238.85  157.29
# [1] 0.98
# ------------------------------------------------------
```

4.1 第 1 隠れ層の **conv1** の出力素子数は $M_1 \times M_1 \times N_1$ である。また、**pool1** の出力素子数は $M_2 \times M_2 \times N_2$ である。 M_1, N_1 と M_2, N_2 はいくつか。

- M_1 24
- N_1 20
- M_2 12
- N_2 20

4.2 第 2 隠れ層の **conv2** の出力素子数は $M_3 \times M_3 \times N_3$ である。また、 **pool2** の出力素子数は $M_4 \times M_4 \times N_4$ である。M_3, N_3 と M_4, N_4 はいくつか。

- M_1 8
- N_1 50
- M_2 4
- N_2 50

4.3 第 1 結合層への入力を作っている **mx.symbol.Flatten()** 関数の役割は何か。

(バッチを考慮するならば) 3 次元配列を、バッチを表す次元を除いた 2 つの次元をまとめることで、2 次元配列に変換している。

一般に深層学習を行う際にはバッチという学習データをいくつかの袋に入れた単位で学習が行われるため、例えば Tensorflow などで実装を行う際にはこの “一次元増える” 現象について理解しておく必要がある。

5 レポート課題 3.2

学習データとテストデータに対する正答率はいくつになったか。

学習データ	0.969400007724762
テストデータ	0.986

6 レポート課題 3.3

dropout 正則化を外した場合、学習データとテストデータに対する正答率はいくつになったか。

学習データ	1
テストデータ	0.982

7 レポート課題 3.4

dropout 正則化を外した状態で、出力関数を tanh から ReLU に変えた場合、学習データとテストデータに対する正答率はいくつになったか。

学習データ	1
テストデータ	0.982

8 レポート課題 3.5

dropout 正則化と ReLU と用いた場合、学習データとテスト d セータに対する正答率はいくつになったか。

学習データ	0.972400006055832
テストデータ	0.98

9 レポート課題 3.6

以上の比較実験から、 dropout 正則化は有効といえるか？また、出力関数はどちらがよいといえるか。

テストデータの値の差から、有効と言える。また出力関数は ReLU の方が適切であるように考えられる。しかし、この値の差は非常に軽微であるように見え、もう少し難しい問題を用いて性能比較を行わなければ明言することはできないだろう。

10 レポート課題 3.7

以上の中で、テストデータに対する正答率が最も良い組み合わせのネットワークに Kaggle の学習データで学習させなさい。Kaggle のテストデータに対する識別結果を下記の手順で作成し、 Kaggle に submit しなさい。正答率と順位はいくつになったか。

この問題に対しては特別に R のプログラムを作成し実験を行った。以下にそれを示す。尚、最も良い組み合わせは活性化関数を “tanh” にして dropout を “有効” にしたものであった。

```
# R 3.5.1 で実行確認
# import libraries
library(nnet)
library(MASS)
library(mxnet)

train <- read.csv("data/train.csv", header = TRUE)
test <- read.csv("data/test.csv", header = TRUE)
train <- data.matrix(train)
test <- data.matrix(test)
train.x <- train[,-1]
train.y <- train[,1]
test_org <- test
# test <- test[,-1]
train.x <- t(train.x/255) # [0, 255] -> [0, 1]
test <- t(test/255)


# preparing train/test data
train.array <- train.x
dim(train.array) <- c(28, 28, 1, ncol(train.x))

test.array <- test
dim(test.array) <- c(28, 28, 1, ncol(test))

# preparing training
mx.set.seed(0)
devices <- mx.cpu()
tic <- proc.time()


training_mnist_cnn = function(dropout, activate_fn) {
  # input layer
```

```
34  data <- mx.symbol.Variable("data")
35
36  # hidden layer 1
37  conv1 <- mx.symbol.Convolution(data=data, kernel=c(5, 5), num_filter=20)
38  tanh1 <- mx.symbol.Activation(data=conv1, act_type=activate_fn)
39  pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2, 2),
40                             stride=c(2, 2))
41
42  if (dropout) {
43    drop1 <- mx.symbol.Dropout(data=pool1, p=0.5)
44  } else {
45    drop1 <- pool1
46  }
47
48  # hidden layer 2
49  conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(5,5), num_filter=50)
50  tanh2 <- mx.symbol.Activation(data=conv2, act_type=activate_fn)
51  pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2, 2),
52                             stride=c(2, 2))
53
54  if (dropout) {
55    drop2 <- mx.symbol.Dropout(data=pool2, p=0.5)
56  } else {
57    drop2 <- pool2
58  }
59
60  # fully connected layer 1
61  flatten <- mx.symbol.Flatten(data=drop2)
62  fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
63  tanh3 <- mx.symbol.Activation(data=fc1, act_type=activate_fn)
64
65  if (dropout) {
66    drop3 <- mx.symbol.Dropout(data=tanh3, p=0.5)
67  } else {
68    drop3 <- tanh3
69  }
70
71  # fully connected layer 2
72  fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)
73
74  # output layer
75  lenet <- mx.symbol.SoftmaxOutput(data=fc2)
76
77  # preparing train/test data
```

```
78    train.array <- train.x
79    dim(train.array) <- c(28, 28, 1, ncol(train.x))
80
81    test.array <- test
82    dim(test.array) <- c(28, 28, 1, ncol(test))
83
84    # preparing training
85    mx.set.seed(0)
86    devices <- mx.cpu()
87    tic <- proc.time()
88
89    # training model
90    model.CNNtanhDrop <- mx.model.FeedForward.create(lenet, X=train.array, y=train.y,
91                                                     ctx=devices, num.round = 30,
92                                                     array.batch.size = 100,
93                                                     learning.rate=0.05,
94                                                     momentum=0.9, wd=0.000001,
95                                                     eval.metric=mx.metric.accuracy,
96                                                     batch.end.callback = mx.callback.log.t
97    print(proc.time() - tic)
98
99    # sum(diag(table(test_org[,1], pred.label))) / 1000
100
101   model.CNNtanhDrop
102 }
103
104 model.CNNtanhDrop = training_mnist_cnn(TRUE, "tanh")
105
106 preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
107 pred.label <- max.col(t(preds)) -1
108
109 submission <- data.frame(ImageId=1:ncol(test),
110                          Label=pred.label)
111
112 write.csv(submission, file='submission.csv', row.names=FALSE, quote=FALSE)
113
114 # 結果
115 # 1411 位 elect 12/18/2018
116 # 0.98600
```

正答率は 0.98600 (98.6%) であった。しかし順位に関しては時間と共に変化するようであるため、正確には示すことが出来なかった。 1411 位というのは結果をアップロードした直後のものである。