

# 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。

---

```
1 # R 3.5.1 で実行確認
2 # import libraries
3 library(nnet)
4 library(MASS)
5
6 # レポート課題 1.1
7 # 1)
8 # 隠れ素子の数を一つずつ増やし、
9 # 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。
10 # 2)
11 # 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。
12 # 3)
13 # 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。
14 # 4)
15 # 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、
16 # 隠れ素子が1個の場合と比較検討しなさい。
17 hidden = c(1:40)
18 iter = c(1:10)
19 trave = rep(0, length(hidden))
20 res_s = c()
21 tr = matrix(0, length(iter), length(hidden))
22 z = 0
23 for (i in 1:length(hidden)) {
24   for (j in 1:length(iter)) {
```

```

25   res_s = c(list(nnet(classes~., data=xor, size=hidden[i], rang=0.1)), res_s)
26   out = predict(res_s[i][[1]], xor, type="class")
27   tr[j, i] = mean(out != xor$classes)
28   }
29   trave[i] = mean(tr[,i])
30   if(trave[i] == 0.0 && z == 0) {
31     z = i
32   }
33 }
34 res_s = rev(res_s)
35 # 1) 37
36 z
37 # 2) 1-1-2.png
38 plot(hidden, trave, type="b", lty=1, lwd=2)
39 # 3) 1-1-3.png
40 hist(res_s[1][[1]]$wts, breaks=seq(-0.1, 0.1, 0.04), freq=TRUE)
41 # 4) 1-1-4.png
42 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。

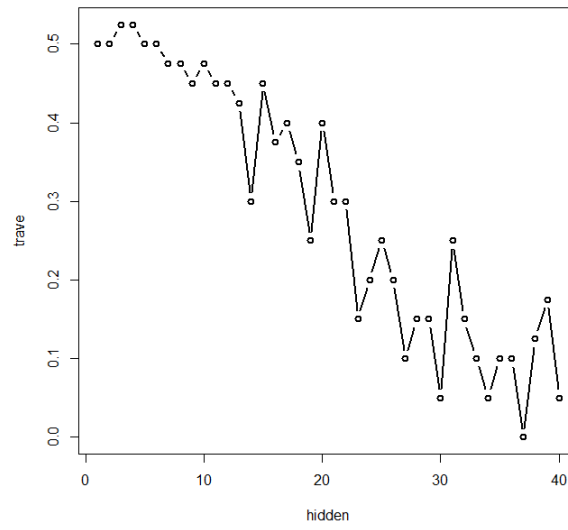


Figure 1: 1-1-2.png 隠れ素子の数と誤認識率の平均のグラフ

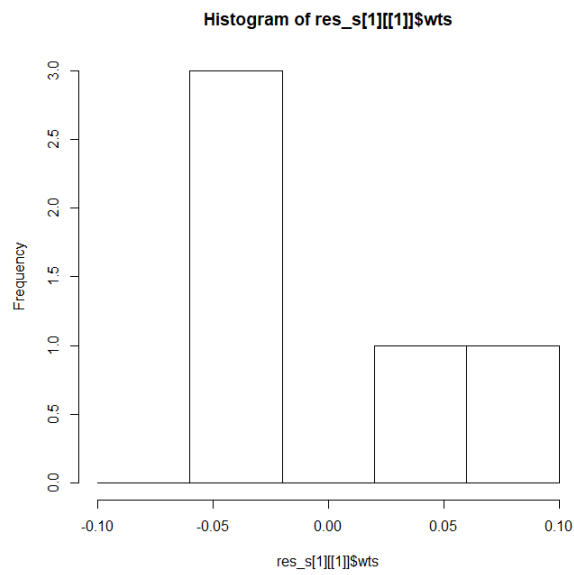


Figure 2: 1-1-3.png 隠れ素子が 1 個の場合の結合係数の大きさの分布

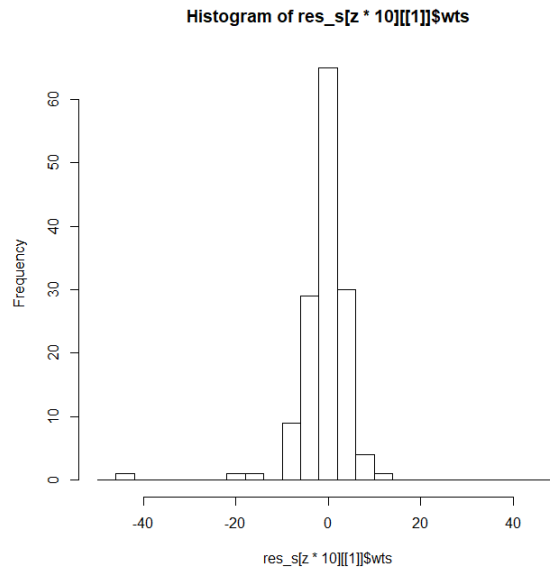


Figure 3: 1-1-4.png 10 回とも正しく識別出来た場合の結合係数の大きさの分布

## 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。

---

```

1 # レポート課題 1. 2
2 # データの用意
3 ir <- data.frame(rbind(iris3[,1], iris3[,2], iris3[,3]),
4                   species=factor(c(rep("sv", 50), rep("c", 50),
5                                   rep("sv", 50))))
6 samp <- c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25))
7 # 1)
8 # 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、
9 # 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。
10 # 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、
11 # 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。
12 # 2)
13 # 再代入誤りが一番小さな場合の結合係数の分布と、
14 # 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。
15 # 3)
16 # decay=0.01 にして同様の実験を行い、

```

```

17 # 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。
18 # 4)
19 # 再代入誤りが一番小さな場合の結合係数の分布と、
20 # 汎化誤差が一番小さな場合の結合係数の分布を、
21 # decay=0 の場合と比較しなさい。
22 hidden =
23 c(1:10)
24 iter = c(1:10)
25 decay = 0
26 trave_learn = rep(0, length(hidden))
27 trave_test = rep(0, length(hidden))
28 res_s = c()
29 tr_learn = matrix(0, length(iter), length(hidden))
30 tr_test = matrix(0, length(iter), length(hidden))
31 for (i in 1:length(hidden)) {
32   for (j in 1:length(iter)) {
33     res_s = c(list(nnet(species~., data=ir[samp,], size=hidden[i],
34                      rang=0.5, decay=decay, maxit=200)), res_s)
35     out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
36     out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
37     tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
38     tr_test[j, i] = mean(out_test != ir[-samp,]$species)
39   }
40   trave_learn[i] = mean(tr_learn[, i])
41   trave_test[i] = mean(tr_test[, i])
42 }
43 res_s = rev(res_s)
44 # 1) 隠れ素子数が1つのときの再代入誤り、汎化誤差
45 trave_learn[1]
46 trave_test[1]
47 # 1) 1-2-1-1.png 再代入誤りの変化
48 plot(hidden, trave_learn, type="b", lty=1, lwd=2)
49 # 1) 1-2-1-2.png 汎化誤差の変化
50 plot(hidden, trave_test, type="b", lty=1, lwd=2)
51 # 2)
52 which.min(trave_learn) # 9
53 which.min(trave_test) # 10
54 # 2) 1-2-2-1.png
55 hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
56 # 2) 1-2-2-2.png
57 hist(res_s[which.min(trave_test)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
58 # 3)
59 hidden = c(1:10)
60 iter = c(1:10)

```

```

61 decay = 0.01
62 trave_learn = rep(0, length(hidden))
63 trave_test = rep(0, length(hidden))
64 res_s = c()
65 tr_learn = matrix(0, length(iter), length(hidden))
66 tr_test = matrix(0, length(iter), length(hidden))
67 for (i in 1:length(hidden)) {
68   for (j in 1:length(iter)) {
69     res_s = c(list(nnet(species~., data=ir[samp,],
70                       size=hidden[i], rang=0.5, decay=decay, maxit=200)), res_s)
71     out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
72     out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
73     tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
74     tr_test[j, i] = mean(out_test != ir[-samp,]$species)
75   }
76   trave_learn[i] = mean(tr_learn[, i])
77   trave_test[i] = mean(tr_test[, i])
78 }
79 res_s = rev(res_s)
80 # 3) 1-2-3-1.png
81 plot(hidden, trave_learn, type="b", lty=1, lwd=2)
82 # 3) 1-2-3-2.png
83 plot(hidden, trave_test, type="b", lty=1, lwd=2)
84 # 4)
85 which.min(trave_learn) # 4
86 which.min(trave_test) # 4
87 # 4) 1-2-4-1.png
88 hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-6, 6, 1), freq=TRUE)
89 # 4) 1-2-4-2.png
90 hist(res_s[which.min(trave_test)][[1]]$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

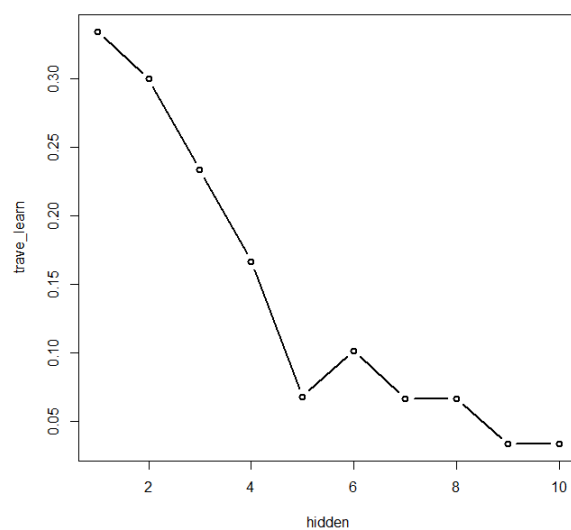


Figure 4: 1-2-1-1.png 再代入誤りの変化のグラフ

- 汎化誤差の変化のグラフ

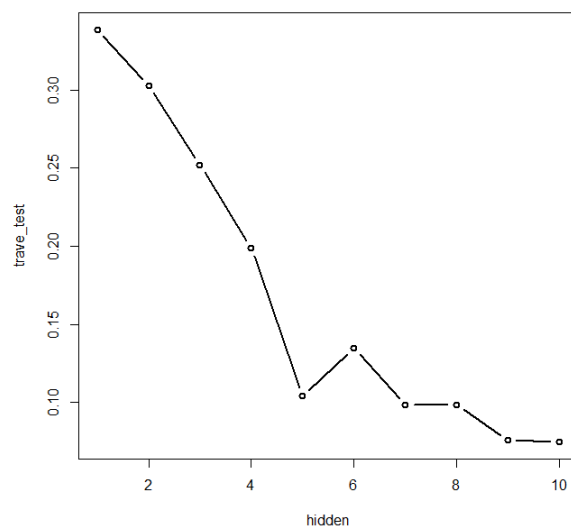


Figure 5: 1-2-1-2.png 汎化誤差の変化のグラフ



## 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

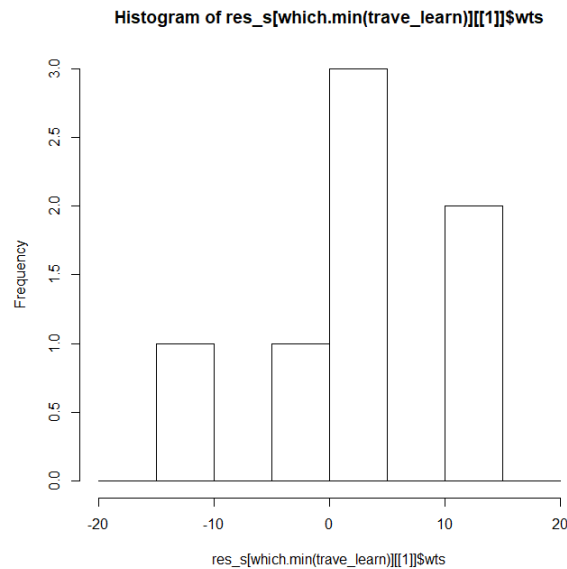


Figure 6: 1-2-2-1.png 再代入誤りが一番小さな場合の結合係数の分布

- 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思像できる。

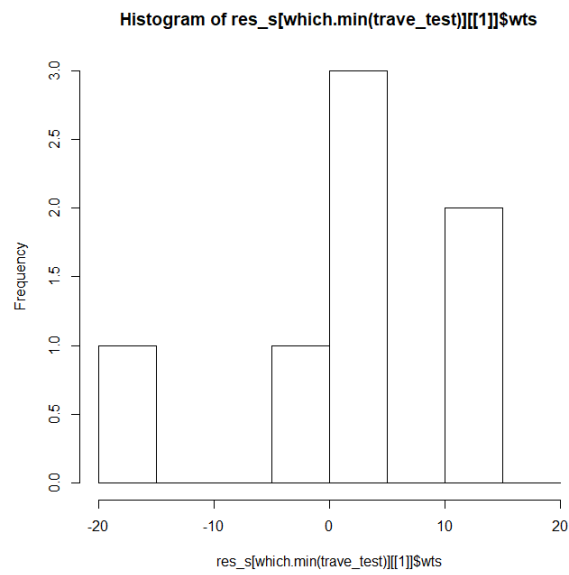


Figure 7: 1-2-2-2.png 汎化誤差が一番小さな場合の結合係数の分布

**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

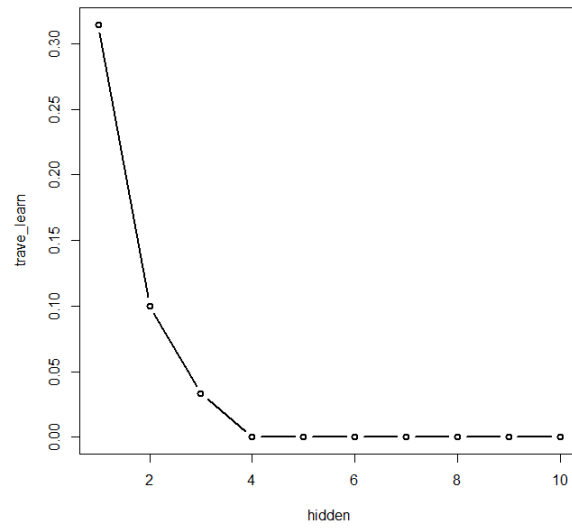


Figure 8: 1-2-3-1.png 再代入誤りの変化のグラフ (decay=0.01)

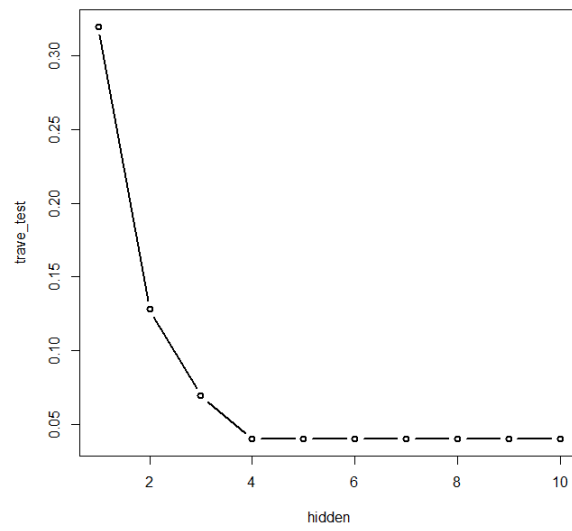


Figure 9: 1-2-3-2.png 汎化誤差の変化のグラフ (decay=0.1)

## 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

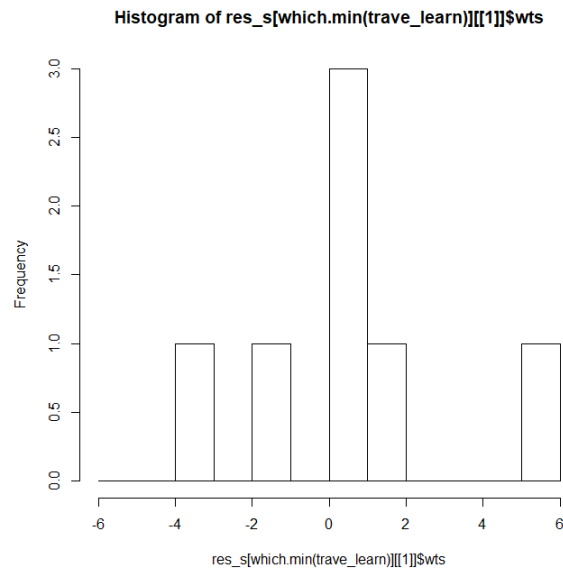


Figure 10: 1-2-4-1.png 再代入誤りが一番小さな場合の結合係数の分布

- 汎化誤差が一番小さな場合の結合係数の分布
- 比較

いずれと比較しても、 $\text{decay} = 0.01$  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  $\text{decay} = 0.01$  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。

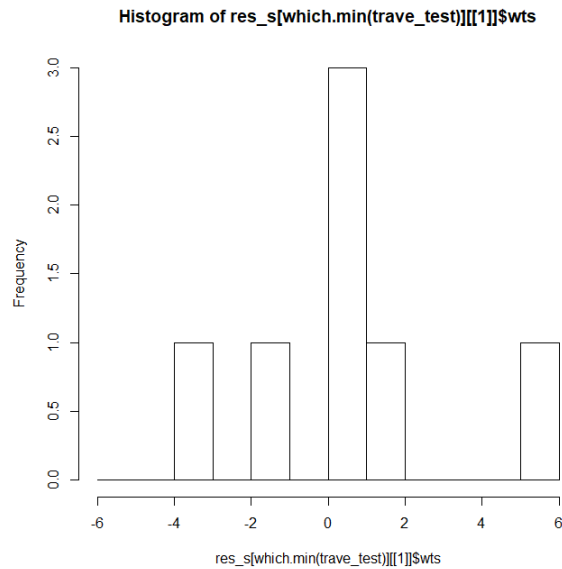


Figure 11: 1-2-4-2.png 汎化誤差が一番小さな場合の結合係数の分布

### 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。

---

```

1 # R 3.5.1 で実行確認
2 # import libraries
3 library(nnet)
4 library(MASS)
5 library(mxnet)
6
7 # create dataset
8 train <- read.csv("data/short_prac_train.csv", header = TRUE)
9 test <- read.csv("data/short_prac_test.csv", header = TRUE)
10 train <- data.matrix(train) test <- data.matrix(test)
11 train.x <- train[,-1]
12 train.y <- train[,1]
13 test_org <- test
14 test <- test[,-1]
15 train.x <- t(train.x/255) # [0, 255] -> [0, 1]
```

```

16 test <- t(test/255)
17 table(train.y)
18
19 # check image
20 image(x=seq(1:28),y=seq(1:28), matrix(train.x[,4], 28, 28)[, 28:1],
21       col = gray(0:255/255))
22
23 # sample
24 # network settings
25 data <- mx.symbol.Variable("data")
26 fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
27 act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
28 fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
29 act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
30 fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
31 softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
32
33 # network training
34 devices <- mx.cpu()
35 mx.set.seed(0)
36 model <- mx.model.FeedForward.create(softmax, X = train.x, y = train.y,
37                                     initializer = mx.init.uniform(0.07),
38                                     ctx = devices,
39                                     num.round = 10, array.batch.size = 100,
40                                     learning.rate=0.05,
41                                     momentum=0.9, wd=0.00001,
42                                     eval.metric = mx.metric.accuracy,
43                                     epoch.end.callback =
44                                     mx.callback.log.train.metric(100))
45
46 preds <- predict(model, test, ctx=devices)
47 pred.label <- max.col(t(preds)) -1
48 sum(diag(table(test_org[,1], pred.label))) / 1000
49 table(test_org[,1], pred.label)
50
51 # レポート課題 2. 1
52 # 1)
53 # 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。
54 # 2)
55 # 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、
56 # 認識率はどのようになるか。
57 # ReLUの場合と同じ条件で実験し、比較しなさい。
58
59 training_mnist <- function(seed, activate_fun) {

```

```

60  # network settings
61  data <- mx.symbol.Variable("data")
62  fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
63  act1 <- mx.symbol.Activation(fc1, name="relu1", act_type=activate_fun)
64  fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
65  act2 <- mx.symbol.Activation(fc2, name="relu2", act_type=activate_fun)
66  fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
67  softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
68
69  devices <- mx.cpu()
70  mx.set.seed(seed)
71
72  # training network
73  model <- mx.model.FeedForward.create(softmax, X = train.x, y = train.y,
74                                     initializer = mx.init.uniform(0.07),
75                                     ctx = devices,
76                                     num.round = 10, array.batch.size = 100,
77                                     learning.rate=0.05,
78                                     momentum=0.9, wd=0.00001,
79                                     eval.metric = mx.metric.accuracy,
80                                     epoch.end.callback =
81                                     mx.callback.log.train.metric(100))
82  preds <- predict(model, test, ctx=devices)
83  pred.label <- max.col(t(preds)) -1
84  return(mean(test_org[,1] == pred.label))
85 }
86
87 # 1)
88 seeds = list(11, 25, 2018)
89
90 training_mnist(seeds[1][[1]], "relu")
91
92 # -----
93 # [1] Train-accuracy=0.41060000102967
94 # [2] Train-accuracy=0.813400003910065
95 # [3] Train-accuracy=0.891999999284744
96 # [4] Train-accuracy=0.911600003242493
97 # [5] Train-accuracy=0.937400004863739
98 # [6] Train-accuracy=0.948400005102158
99 # [7] Train-accuracy=0.966600004434586
100 # [8] Train-accuracy=0.974000008106232
101 # [9] Train-accuracy=0.979200007915497
102 # [10] Train-accuracy=0.984000010490418
103 # [1] 0.938

```

```

104 # -----
105
106 training_mnist(seeds[2][[1]], "relu")
107 # -----
108 # [1] Train-accuracy=0.426600000560284
109 # [2] Train-accuracy=0.818000000715256
110 # [3] Train-accuracy=0.873200000524521
111 # [4] Train-accuracy=0.902800003290176
112 # [5] Train-accuracy=0.933199996948242
113 # [6] Train-accuracy=0.950400000810623
114 # [7] Train-accuracy=0.961600004434586
115 # [8] Train-accuracy=0.96960000872612
116 # [9] Train-accuracy=0.979400007724762
117 # [10] Train-accuracy=0.98080001115799
118 # [1] 0.941
119 # -----
120
121 training_mnist(seeds[3][[1]], "relu")
122 # -----
123 # [1] Train-accuracy=0.44320000231266
124 # [2] Train-accuracy=0.831800000667572
125 # [3] Train-accuracy=0.890200002193451
126 # [4] Train-accuracy=0.921600000858307
127 # [5] Train-accuracy=0.937600003480911
128 # [6] Train-accuracy=0.949200004339218
129 # [7] Train-accuracy=0.960400002002716
130 # [8] Train-accuracy=0.969600001573563
131 # [9] Train-accuracy=0.971800007820129
132 # [10] Train-accuracy=0.967400006055832
133 # [1] 0.931
134 # -----
135
136 # 2)
137 training_mnist(seeds[1][[1]], "sigmoid")
138 # -----
139 # [1] Train-accuracy=0.0967999996244907
140 # [2] Train-accuracy=0.117599999085069
141 # [3] Train-accuracy=0.153000000119209
142 # [4] Train-accuracy=0.275600000321865
143 # [5] Train-accuracy=0.43559999704361
144 # [6] Train-accuracy=0.577199996709824
145 # [7] Train-accuracy=0.684799997806549
146 # [8] Train-accuracy=0.750999997854233
147 # [9] Train-accuracy=0.796199997663498

```



```

148 # [10] Train-accuracy=0.821999995708466
149 # [1] 0.827
150 # -----
151
152 training_mnist(seeds[2][[1]], "sigmoid")
153 # -----
154 # [1] Train-accuracy=0.102400000393391
155 # [2] Train-accuracy=0.106399999856949
156 # [3] Train-accuracy=0.132000000178814
157 # [4] Train-accuracy=0.217799999862909
158 # [5] Train-accuracy=0.385399999022484
159 # [6] Train-accuracy=0.526599999666214
160 # [7] Train-accuracy=0.66940000295639
161 # [8] Train-accuracy=0.76299999833107
162 # [9] Train-accuracy=0.807399994134903
163 # [10] Train-accuracy=0.829199995994568
164 # [1] 0.84
165 # -----
166
167 training_mnist(seeds[3][[1]], "sigmoid")
168 # -----
169 # [1] Train-accuracy=0.0975999997928739
170 # [2] Train-accuracy=0.106199999824166
171 # [3] Train-accuracy=0.129200000017881
172 # [4] Train-accuracy=0.204800001382828
173 # [5] Train-accuracy=0.416599997282028
174 # [6] Train-accuracy=0.578999997973442
175 # [7] Train-accuracy=0.695199999809265
176 # [8] Train-accuracy=0.759400001764297
177 # [9] Train-accuracy=0.807399997711182
178 # [10] Train-accuracy=0.839199997186661
179 # [1] 0.837
180 # -----

```

---

### 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。

---

```
1  # R 3.5.1 で実行確認
2  # import libraries
3  library(nnet)
4  library(MASS)
5  library(mxnet)
6
7  # create dataset
8  train <- read.csv("data/short_prac_train.csv", header = TRUE)
9  test <- read.csv("data/short_prac_test.csv", header = TRUE)
10 train <- data.matrix(train)
11 test <- data.matrix(test)
12 train.x <- train[,-1]
13 train.y <- train[,1]
14 test_org <- test
15 test <- test[,-1]
16 train.x <- t(train.x/255) # [0, 255] -> [0, 1]
17 test <- t(test/255)
18 table(train.y)
19
20 # input layer
21 data <- mx.symbol.Variable("data")
22
23 # hidden layer 1
24 conv1 <- mx.symbol.Convolution(data=data, kernel=c(5, 5), num_filter=20)
25 tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
26 pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2, 2),
27                             stride=c(2, 2))
28 drop1 <- mx.symbol.Dropout(data=pool1, p=0.5)
29
30 # hidden layer 2
31 conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(5,5), num_filter=50)
32 tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
33 pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2, 2),
34                             stride=c(2, 2))
35 drop2 <- mx.symbol.Dropout(data=pool2, p=0.5)
36
37 # fully connected layer 1
38 flatten <- mx.symbol.Flatten(data=drop2)
```

```

39 fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
40 tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
41 drop3 <- mx.symbol.Dropout(data=tanh3, p=0.5)
42
43 # fully connected layer 2
44 fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)
45
46 # output layer
47 lenet <- mx.symbol.SoftmaxOutput(data=fc2)
48
49 # preparing train/test data
50 train.array <- train.x
51 dim(train.array) <- c(28, 28, 1, ncol(train.x))
52
53 test.array <- test
54 dim(test.array) <- c(28, 28, 1, ncol(test))
55
56 # preparing training
57 mx.set.seed(0)
58 devices <- mx.cpu()
59 tic <- proc.time()
60
61 # training model
62 model.CNNtanhDrop <- mx.model.FeedForward.create(lenet, X=train.array,
63                                                    y=train.y, ctx=devices,
64                                                    num.round = 30,
65                                                    array.batch.size = 100,
66                                                    learning.rate=0.05,
67                                                    momentum=0.9, wd=0.00001,
68                                                    eval.metric=mx.metric.accuracy,
69                                                    batch.end.callback =
70                                                    mx.callback.log.train.metric(100))
71 print(proc.time() - tic)
72 preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
73 pred.label <- max.col(t(preds)) -1
74 sum(diag(table(test_org[,1], pred.label))) / 1000
75 # 1)
76 # 1-1)
77 # M1 : 24
78 # N1 : 20
79 # M2 : 12
80 # N2 : 20
81 # 1-2)
82 # M3 : 8

```

```

83 # N3 : 50
84 # M4 : 4
85 # N4 : 50
86 # 1-3)
87 # 3次元配列 2次元配列に変換している
88 # 2)
89 # -----
90 # [1] Train-accuracy=0.0943999997526407
91 # [2] Train-accuracy=0.089199999794364
92 # [3] Train-accuracy=0.0956000000983477
93 # [4] Train-accuracy=0.353200001269579
94 # [5] Train-accuracy=0.814999997615814
95 # [6] Train-accuracy=0.879199998378754
96 # [7] Train-accuracy=0.91200000166893
97 # [8] Train-accuracy=0.918200002908707
98 # [9] Train-accuracy=0.933600000143051
99 # [10] Train-accuracy=0.934600001573563
100 # [11] Train-accuracy=0.940800001621246
101 # [12] Train-accuracy=0.947199996709824
102 # [13] Train-accuracy=0.945600001811981
103 # [14] Train-accuracy=0.943799996376038
104 # [15] Train-accuracy=0.945399998426437
105 # [16] Train-accuracy=0.958400007486343
106 # [17] Train-accuracy=0.956800009012222
107 # [18] Train-accuracy=0.957000002861023
108 # [19] Train-accuracy=0.958199999332428
109 # [20] Train-accuracy=0.958999999761581
110 # [21] Train-accuracy=0.965000002384186
111 # [22] Train-accuracy=0.966000006198883
112 # [23] Train-accuracy=0.966400002241135
113 # [24] Train-accuracy=0.964200004339218
114 # [25] Train-accuracy=0.966800007820129
115 # [26] Train-accuracy=0.964200004339218
116 # [27] Train-accuracy=0.962800003290176
117 # [28] Train-accuracy=0.970400004386902
118 # [29] Train-accuracy=0.969200010299683
119 # [30] Train-accuracy=0.97040000796318
120 # [1] 0.985
121 # -----
122
123 training_mnist_cnn = function(dropout, activate_fn) {
124   # input layer
125   data <- mx.symbol.Variable("data")
126

```

```

127 # hidden layer 1
128 conv1 <- mx.symbol.Convolution(data=data, kernel=c(5, 5), num_filter=20)
129 tanh1 <- mx.symbol.Activation(data=conv1, act_type=activate_fn)
130 pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2, 2),
131                             stride=c(2, 2))
132
133 if (dropout) {
134     drop1 <- mx.symbol.Dropout(data=pool1, p=0.5)
135 } else {
136     drop1 <- pool1
137 }
138
139 # hidden layer 2
140 conv2 <- mx.symbol.Convolution(data=drop1, kernel=c(5,5), num_filter=50)
141 tanh2 <- mx.symbol.Activation(data=conv2, act_type=activate_fn)
142 pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2, 2),
143                             stride=c(2, 2))
144
145 if (dropout) {
146     drop2 <- mx.symbol.Dropout(data=pool2, p=0.5)
147 } else {
148     drop2 <- pool2
149 }
150
151 # fully connected layer 1
152 flatten <- mx.symbol.Flatten(data=drop2)
153 fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
154 tanh3 <- mx.symbol.Activation(data=fc1, act_type=activate_fn)
155
156 if (dropout) {
157     drop3 <- mx.symbol.Dropout(data=tanh3, p=0.5)
158 } else {
159     drop3 <- tanh3
160 }
161
162 # fully connected layer 2
163 fc2 <- mx.symbol.FullyConnected(data=drop3, num_hidden=10)
164
165 # output layer
166 lenet <- mx.symbol.SoftmaxOutput(data=fc2)
167
168 # preparing train/test data
169 train.array <- train.x
170 dim(train.array) <- c(28, 28, 1, ncol(train.x))

```

```

171
172 test.array <- test
173 dim(test.array) <- c(28, 28, 1, ncol(test))
174
175 # preparing training
176 mx.set.seed(0)
177 devices <- mx.cpu()
178 tic <- proc.time()
179
180 # training model
181 model.CNNtanhDrop <- mx.model.FeedForward.create(lenet, X=train.array,
182                                                    y=train.y, ctx=devices, num.round = 30, array.batch
183                                                    learning.rate=0.05, momentum=0.9, wd=0.00001,
184                                                    eval.metric=mx.metric.accuracy,
185                                                    batch.end.callback = mx.callback.log.train.metric(
186 print(proc.time() - tic)
187 preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
188 pred.label <- max.col(t(preds)) -1
189 sum(diag(table(test_org[,1], pred.label))) / 1000
190 }
191
192 # 3)
193 training_mnist_cnn(FALSE, "tanh")
194 # -----
195 # [1] Train-accuracy=0.0951999997347593
196 # [2] Train-accuracy=0.0893999997526407
197 # [3] Train-accuracy=0.0931999997794628
198 # [4] Train-accuracy=0.351199997663498
199 # [5] Train-accuracy=0.840799999237061
200 # [6] Train-accuracy=0.919000004529953
201 # [7] Train-accuracy=0.951199996471405
202 # [8] Train-accuracy=0.962000002861023
203 # [9] Train-accuracy=0.970400011539459
204 # [10] Train-accuracy=0.979800009727478
205 # [11] Train-accuracy=0.984800012111664
206 # [12] Train-accuracy=0.991200008392334
207 # [13] Train-accuracy=0.992800006866455
208 # [14] Train-accuracy=0.994800004959106
209 # [15] Train-accuracy=0.996200003623962
210 # [16] Train-accuracy=0.995400004386902
211 # [17] Train-accuracy=0.997000002861023
212 # [18] Train-accuracy=0.997800002098084
213 # [19] Train-accuracy=0.999200000762939
214 # [20] Train-accuracy=1

```

```

215 # [21] Train-accuracy=1
216 # [22] Train-accuracy=1
217 # [23] Train-accuracy=1
218 # [24] Train-accuracy=1
219 # [25] Train-accuracy=1
220 # [26] Train-accuracy=1
221 # [27] Train-accuracy=1
222 # [28] Train-accuracy=1
223 # [29] Train-accuracy=1
224 # [30] Train-accuracy=1
225 # user system elapsed
226 # 549.00 251.10 163.95
227 # [1] 0.982
228 # -----
229 # comment: over fitting
230 # 4)
231 training_mnist_cnn(FALSE, "relu")
232 # -----
233 # [1] Train-accuracy=0.0957999996095896
234 # [2] Train-accuracy=0.0893999997526407
235 # [3] Train-accuracy=0.0903999998420477
236 # [4] Train-accuracy=0.110599999576807
237 # [5] Train-accuracy=0.418600002527237
238 # [6] Train-accuracy=0.869200000762939
239 # [7] Train-accuracy=0.947000004053116
240 # [8] Train-accuracy=0.965000007152557
241 # [9] Train-accuracy=0.973400008678436
242 # [10] Train-accuracy=0.975400009155273
243 # [11] Train-accuracy=0.980800009965897
244 # [12] Train-accuracy=0.986800010204315
245 # [13] Train-accuracy=0.99100000500679
246 # [14] Train-accuracy=0.991400008201599
247 # [15] Train-accuracy=0.994200005531311
248 # [16] Train-accuracy=0.99300000667572
249 # [17] Train-accuracy=0.991000006198883
250 # [18] Train-accuracy=0.995400004386902
251 # [19] Train-accuracy=0.996800003051758
252 # [20] Train-accuracy=0.995800004005432
253 # [21] Train-accuracy=0.998200001716614
254 # [22] Train-accuracy=0.999800000190735
255 # [23] Train-accuracy=1
256 # [24] Train-accuracy=1
257 # [25] Train-accuracy=1
258 # [26] Train-accuracy=1

```



```

259 # [27] Train-accuracy=1
260 # [28] Train-accuracy=1
261 # [29] Train-accuracy=1
262 # [30] Train-accuracy=1
263 # user system elapsed
264 # 513.99 243.98 154.81
265 # [1] 0.979
266 # -----
267 # 5)
268 training_mnist_cnn(TRUE, "relu")
269 # [1] Train-accuracy=0.0949999997764826
270 # [2] Train-accuracy=0.0893999997526407
271 # [3] Train-accuracy=0.0899999997764826
272 # [4] Train-accuracy=0.100200000032783
273 # [5] Train-accuracy=0.35619999974966
274 # [6] Train-accuracy=0.773400000333786
275 # [7] Train-accuracy=0.879000000953674
276 # [8] Train-accuracy=0.908400002717972
277 # [9] Train-accuracy=0.922000004053116
278 # [10] Train-accuracy=0.927400002479553
279 # [11] Train-accuracy=0.938999998569489
280 # [12] Train-accuracy=0.948400005102158
281 # [13] Train-accuracy=0.951800004243851
282 # [14] Train-accuracy=0.953199998140335
283 # [15] Train-accuracy=0.955200002193451
284 # [16] Train-accuracy=0.959800000190735
285 # [17] Train-accuracy=0.95600000500679
286 # [18] Train-accuracy=0.956400002241135
287 # [19] Train-accuracy=0.959000000953674
288 # [20] Train-accuracy=0.961200001239777
289 # [21] Train-accuracy=0.966800006628036
290 # [22] Train-accuracy=0.968200008869171
291 # [23] Train-accuracy=0.969600011110306
292 # [24] Train-accuracy=0.968600010871887
293 # [25] Train-accuracy=0.970800009965897
294 # [26] Train-accuracy=0.970800012350082
295 # [27] Train-accuracy=0.968200005292892
296 # [28] Train-accuracy=0.971000007390976
297 # [29] Train-accuracy=0.969400010108948
298 # [30] Train-accuracy=0.973800009489059
299 # user system elapsed
300 # 543.69 238.85 157.29
301 # [1] 0.986
302 # -----

```

---

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)4次元配列を、バッチを表す次元を除いた3つの次元 (x, y, filter) 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 などを実装を行う際にはこの“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	0.97040000796318
テストデータ	0.985

## 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	0.979

## 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	0.973800009489059
テストデータ	0.986

## 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。テストデータと訓練データで正答率が大きく変化してしまっている時、それは過学習が行われている可能性が高い(実用上ではテストデータと訓練データのデータ性質が乖離してしまった場合や、学習が完了していない場合などにも乖離してしまうことがあるため、訓練データとテストデータの正答率の変化をプロットした上でこの二つの値が離れ始める部分に注目したほうが正しいと考えている。)

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る。この実験の結果は、OS ごと・記法ごと・R のバージョンごとにやや変化するため、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 “relu” にして dropout を “有効” にしたものであった。

---

```
1 # R 3.5.1 で実行確認
2 # import libraries
3 library(nnet)
4 library(MASS)
5 library(mxnet)
6
7 train <- read.csv("data/train.csv", header = TRUE)
8 test <- read.csv("data/test.csv", header = TRUE)
9 train <- data.matrix(train)
10 test <- data.matrix(test)
11 train.x <- train[,-1]
12 train.y <- train[,1]
13 test_org <- test
14 # test <- test[,-1]
15 train.x <- t(train.x/255) # [0, 255] -> [0, 1]
16 test <- t(test/255)
17
18
19 # preparing train/test data
20 train.array <- train.x
21 dim(train.array) <- c(28, 28, 1, ncol(train.x))
22
23 test.array <- test
24 dim(test.array) <- c(28, 28, 1, ncol(test))
25
26 # preparing training
27 mx.set.seed(0)
28 devices <- mx.cpu()
29 tic <- proc.time()
30
31
32 training_mnist_cnn = function(dropout, activate_fn) {
```

```

33  # 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=0.00001,
95                                                    eval.metric=mx.metric.accuracy,
96                                                    batch.end.callback =
97                                                    mx.callback.log.train.metric(100))
98  print(proc.time() - tic)
99
100  # sum(diag(table(test_org[,1], pred.label))) / 1000
101
102  model.CNNtanhDrop
103 }
104
105 model.CNNtanhDrop = training_mnist_cnn(TRUE, "tanh")
106
107 preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
108 pred.label <- max.col(t(preds)) -1
109
110 submission <- data.frame(ImageId=1:ncol(test),
111                           Label=pred.label)
112
113 write.csv(submission, file='submission.csv', row.names=FALSE, quote=FALSE)
114
115 # 結果
116 # 1000 位 elect 12/19/2018
117 # 0.99157

```

---

正答率は 0.99157 (99.157%) 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。1000 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。