

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][[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。

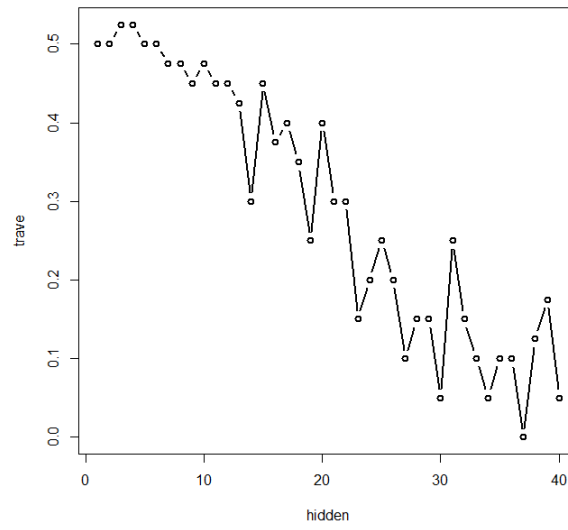


Figure 1: 1-1-2.png

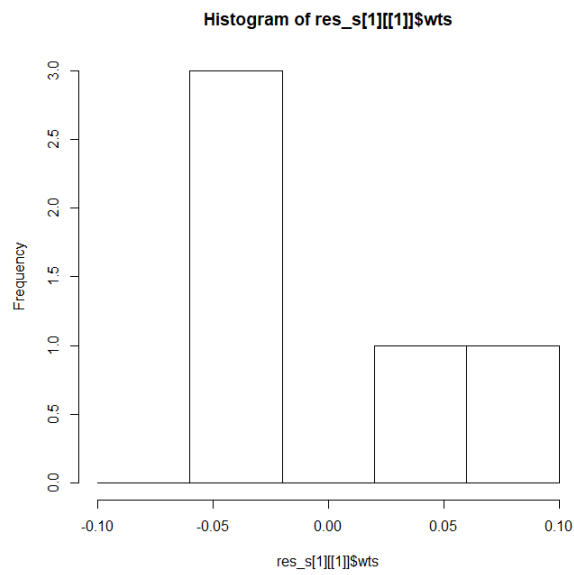


Figure 2: 1-1-3.png

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 = c(1:10)
23 iter = c(1:10)
24 decay = 0
25 trave_learn = rep(0, length(hidden))
26 trave_test = rep(0, length(hidden))
27 res_s = c()
28 tr_learn = matrix(0, length(iter), length(hidden))
29 tr_test = matrix(0, length(iter), length(hidden))
30 for (i in 1:length(hidden)) {
31   for (j in 1:length(iter)) {
32     res_s = c(list(nnet(species~., data=ir[samp,], size=hidden[i],
33       rang=0.5, decay=decay, maxit=200)), res_s)
34     out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
35     out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
36     tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
```

```

37   tr_test[j, i] = mean(out_test != ir[-samp,]$species)
38   }
39   trave_learn[i] = mean(tr_learn[, i])
40   trave_test[i] = mean(tr_test[, i])
41 }
42 res_s = rev(res_s)
43 # 1) 隠れ素子数が1つのときの再代入誤り、汎化誤差
44 trave_learn[1]
45 trave_test[1]
46 # 1) 1-2-1-1.png 再代入誤りの変化
47 plot(hidden, trave_learn, type="b", lty=1, lwd=2)
48 # 1) 1-2-1-2.png 汎化誤差の変化
49 plot(hidden, trave_test, type="b", lty=1, lwd=2)
50 # 2)
51 which.min(trave_learn) # 9
52 which.min(trave_test) # 10
53 # 2) 1-2-2-1.png
54 hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
55 # 2) 1-2-2-2.png
56 hist(res_s[which.min(trave_test)][[1]]$wts, breaks=seq(-20, 20, 5), freq=TRUE)
57 # 3)
58 hidden = c(1:10)
59 iter = c(1:10)
60 decay = 0.01
61 trave_learn = rep(0, length(hidden))
62 trave_test = rep(0, length(hidden))
63 res_s = c()
64 tr_learn = matrix(0, length(iter), length(hidden))
65 tr_test = matrix(0, length(iter), length(hidden))
66 for (i in 1:length(hidden)) {
67   for (j in 1:length(iter)) {
68     res_s = c(list(nnet(species~., data=ir[samp,],
69                       size=hidden[i], rang=0.5, decay=decay, maxit=200)), res_s)
70     out_learn = predict(res_s[i][[1]], ir[samp,], type="class")
71     out_test = predict(res_s[i][[1]], ir[-samp,], type="class")
72     tr_learn[j, i] = mean(out_learn != ir[samp,]$species)
73     tr_test[j, i] = mean(out_test != ir[-samp,]$species)
74   }
75   trave_learn[i] = mean(tr_learn[, i])
76   trave_test[i] = mean(tr_test[, i])
77 }
78 res_s = rev(res_s)
79 # 3) 1-2-3-1.png
80 plot(hidden, trave_learn, type="b", lty=1, lwd=2)

```

```

81 # 3) 1-2-3-2.png
82 plot(hidden, trave_test, type="b", lty=1, lwd=2)
83 # 4)
84 which.min(trave_learn) # 4
85 which.min(trave_test) # 4
86 # 4) 1-2-4-1.png
87 hist(res_s[which.min(trave_learn)][[1]]$wts, breaks=seq(-6, 6, 1), freq=TRUE)
88 # 4) 1-2-4-2.png
89 hist(res_s[which.min(trave_test)][[1]]$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

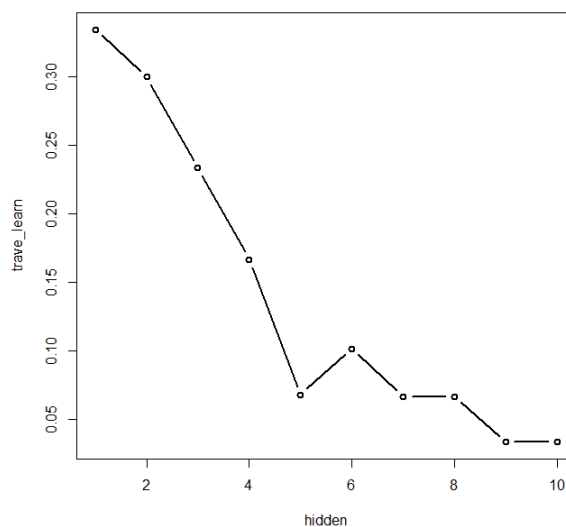


Figure 3: 1-2-1-1.png

- 汎化誤差の変化のグラフ

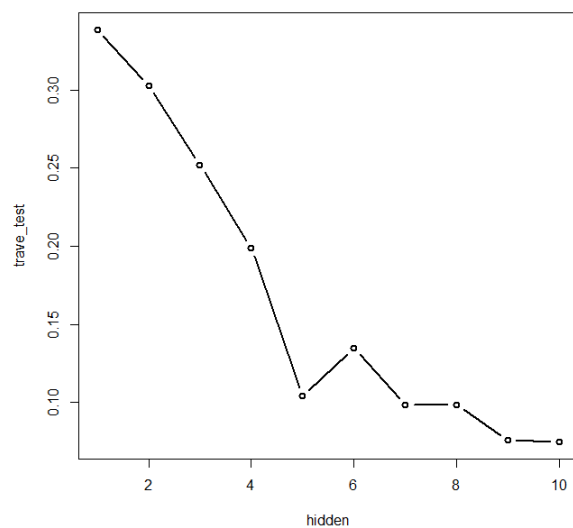


Figure 4: 1-2-1-2.png

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

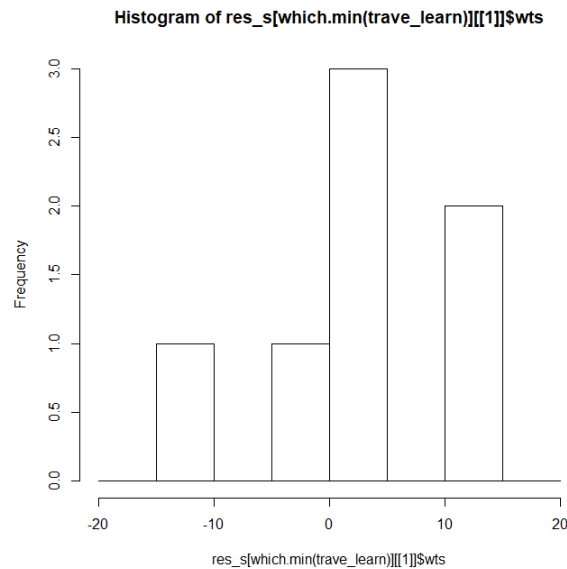


Figure 5: 1-2-2-1.png

- 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。

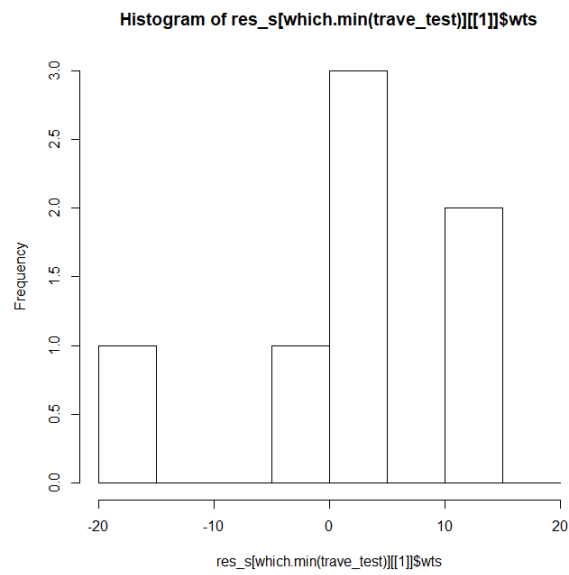


Figure 6: 1-2-2-2.png

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

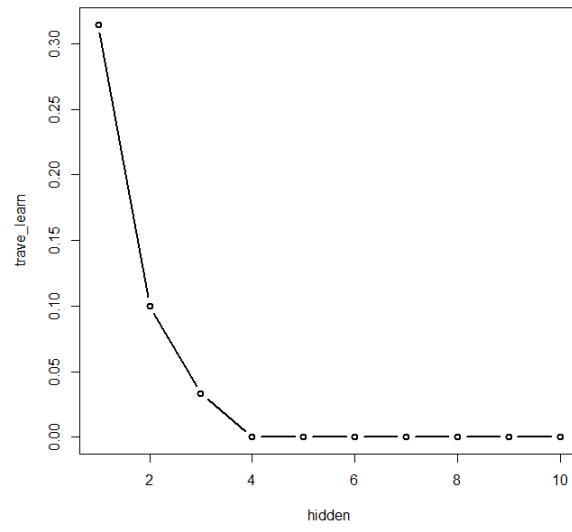


Figure 7: 1-2-3-1.png

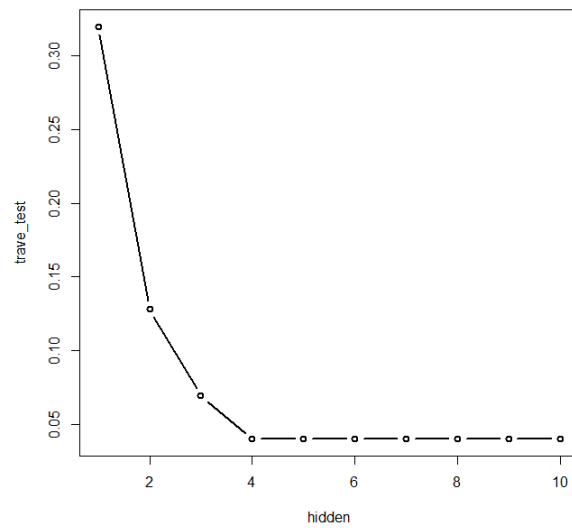


Figure 8: 1-2-3-2.png

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

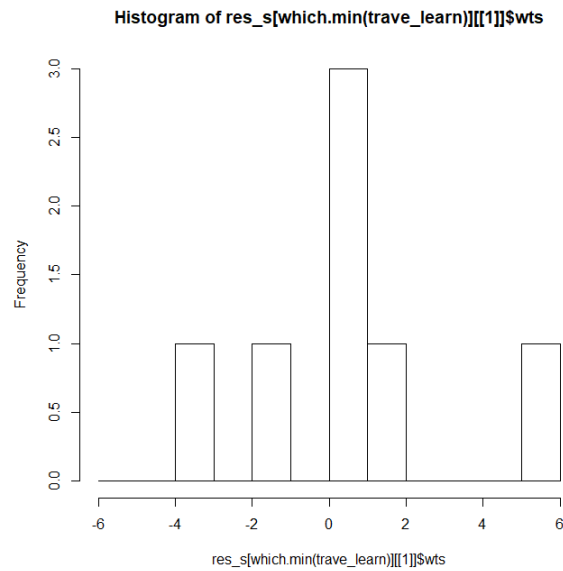


Figure 9: 1-2-4-1.png

- 汎化誤差が一番小さな場合の結合係数の分布
- 比較

どちらも同じ場合の分布なので差は生じない。

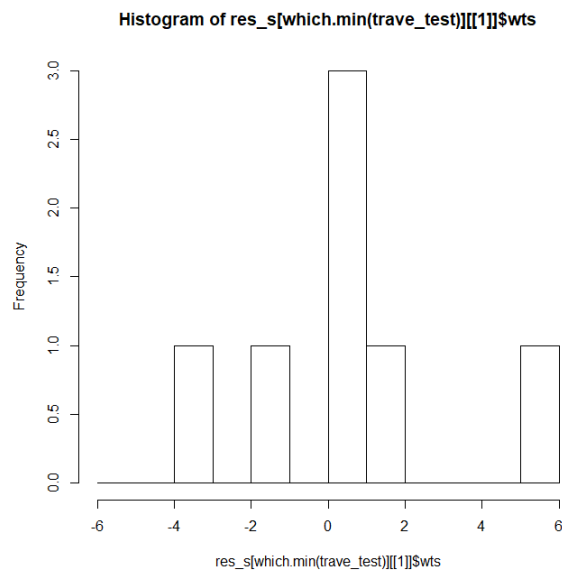


Figure 10: 1-2-4-2.png

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。

```

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 以上の精度を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,
68                                                    wd=0.000001,
69                                                    eval.metric=mx.metric.accuracy,
70                                                    batch.end.callback =
71                                                    mx.callback.log.train.metric(100))
72 print(proc.time() - tic)
73 preds <- predict(model.CNNtanhDrop, test.array, ctx=devices)
74 pred.label <- max.col(t(preds)) -1
75 sum(diag(table(test_org[,1], pred.label))) / 1000
76 # 1)
77 # 1-1)
78 # M1 : 24
79 # N1 : 5000
80 # M2 : 12
81 # N2 : 5000
82 # 1-2)

```

```

83 # M3 : 8
84 # N3 : 5000
85 # M4 : 4
86 # N4 : 5000
87 # 1-3)
88 # 3次元配列 2次元配列に変換している
89 # 2)
90 # -----
91 # [1] Train-accuracy=0.0943999997526407
92 # [2] Train-accuracy=0.089199999794364
93 # [3] Train-accuracy=0.095800000205636
94 # [4] Train-accuracy=0.353800000697374
95 # [5] Train-accuracy=0.815199997425079
96 # [6] Train-accuracy=0.879399998188019
97 # [7] Train-accuracy=0.910400002002716
98 # [8] Train-accuracy=0.919399999380112
99 # [9] Train-accuracy=0.933800001144409
100 # [10] Train-accuracy=0.933000004291534
101 # [11] Train-accuracy=0.939599999189377
102 # [12] Train-accuracy=0.945799996852875
103 # [13] Train-accuracy=0.944800004959106
104 # [14] Train-accuracy=0.945000002384186
105 # [15] Train-accuracy=0.946200004816055
106 # [16] Train-accuracy=0.960200003385544
107 # [17] Train-accuracy=0.956600003242493
108 # [18] Train-accuracy=0.954999998807907
109 # [19] Train-accuracy=0.958400005102158
110 # [20] Train-accuracy=0.961600004434586
111 # [21] Train-accuracy=0.962000002861023
112 # [22] Train-accuracy=0.960800007581711
113 # [23] Train-accuracy=0.964000006914139
114 # [24] Train-accuracy=0.965400005578995
115 # [25] Train-accuracy=0.966400007009506
116 # [26] Train-accuracy=0.968800005912781
117 # [27] Train-accuracy=0.964800003767014
118 # [28] Train-accuracy=0.967800005674362
119 # [29] Train-accuracy=0.965600006580353
120 # [30] Train-accuracy=0.969400007724762
121 # [1] 0.986
122 # -----
123
124 training_mnist_cnn = function(dropout, activate_fn) {
125   # input layer
126   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

```

```

215 # [18] Train-accuracy=0.998200001716614
216 # [19] Train-accuracy=0.999400000572205
217 # [20] Train-accuracy=1
218 # [21] Train-accuracy=1
219 # [22] Train-accuracy=1
220 # [23] Train-accuracy=1
221 # [24] Train-accuracy=1
222 # [25] Train-accuracy=1
223 # [26] Train-accuracy=1
224 # [27] Train-accuracy=1
225 # [28] Train-accuracy=1
226 # [29] Train-accuracy=1
227 # [30] Train-accuracy=1
228 # user system elapsed
229 # 549.00 251.10 163.95
230 # [1] 0.982
231 # -----
232 # comment: over fitting
233 # 4)
234 training_mnist_cnn(FALSE, "relu")
235 # -----
236 # [1] Train-accuracy=0.0957999996095896
237 # [2] Train-accuracy=0.0893999997526407
238 # [3] Train-accuracy=0.0903999998420477
239 # [4] Train-accuracy=0.112800000011921
240 # [5] Train-accuracy=0.434800001382828
241 # [6] Train-accuracy=0.876400001049042
242 # [7] Train-accuracy=0.945999997854233
243 # [8] Train-accuracy=0.963000003099442
244 # [9] Train-accuracy=0.964600006341934
245 # [10] Train-accuracy=0.971200007200241
246 # [11] Train-accuracy=0.978200010061264
247 # [12] Train-accuracy=0.987200009822846
248 # [13] Train-accuracy=0.989800009727478
249 # [14] Train-accuracy=0.991600008010864
250 # [15] Train-accuracy=0.992000007629395
251 # [16] Train-accuracy=0.996000003814697
252 # [17] Train-accuracy=0.997800002098084
253 # [18] Train-accuracy=0.999400000572205
254 # [19] Train-accuracy=0.999400000572205
255 # [20] Train-accuracy=0.998400001525879
256 # [21] Train-accuracy=0.999800000190735
257 # [22] Train-accuracy=0.999000000953674
258 # [23] Train-accuracy=0.999400000572205

```

```

259 # [24] Train-accuracy=0.999800000190735
260 # [25] Train-accuracy=0.999800000190735
261 # [26] Train-accuracy=0.999800000190735
262 # [27] Train-accuracy=1
263 # [28] Train-accuracy=1
264 # [29] Train-accuracy=1
265 # [30] Train-accuracy=1
266 # user system elapsed
267 # 513.99 243.98 154.81
268 # [1] 0.982
269 # -----
270 # 5)
271 training_mnist_cnn(TRUE, "relu")
272 # [1] Train-accuracy=0.0949999997764826
273 # [2] Train-accuracy=0.0893999997526407
274 # [3] Train-accuracy=0.0899999997764826
275 # [4] Train-accuracy=0.100199999809265
276 # [5] Train-accuracy=0.359799997210503
277 # [6] Train-accuracy=0.774200001955032
278 # [7] Train-accuracy=0.879999998807907
279 # [8] Train-accuracy=0.90300000667572
280 # [9] Train-accuracy=0.920800005197525
281 # [10] Train-accuracy=0.927600004673004
282 # [11] Train-accuracy=0.94440000295639
283 # [12] Train-accuracy=0.945800001621246
284 # [13] Train-accuracy=0.945199998617172
285 # [14] Train-accuracy=0.949000002145767
286 # [15] Train-accuracy=0.953400001525879
287 # [16] Train-accuracy=0.960600000619888
288 # [17] Train-accuracy=0.955800002813339
289 # [18] Train-accuracy=0.963600004911423
290 # [19] Train-accuracy=0.960200004577637
291 # [20] Train-accuracy=0.964400005340576
292 # [21] Train-accuracy=0.966000009775162
293 # [22] Train-accuracy=0.967400008440018
294 # [23] Train-accuracy=0.964800004959106
295 # [24] Train-accuracy=0.968200006484985
296 # [25] Train-accuracy=0.969400004148483
297 # [26] Train-accuracy=0.970800008773804
298 # [27] Train-accuracy=0.971400009393692
299 # [28] Train-accuracy=0.966400008201599
300 # [29] Train-accuracy=0.968200007677078
301 # [30] Train-accuracy=0.972400006055832
302 # user system elapsed

```


303 # 543.69 238.85 157.29

304 # [1] 0.98

305 # -----

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 5000
- M_2 12
- N_2 5000

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 5000
- M_2 4
- N_2 5000

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。
3次元配列を、第1次元と第2次元をまとめることで、2次元配列に変換している。

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	0.972400006055832
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。