Lab assignment 1: Image recognition using deep networks

Priyanka Singhvi and Fleur Petit

26 April 2019

Exercise one: Identifying handwritten numbers

Question 1

Automatic recognition of hand-written numbers can be useful for digitalising any written source that contains numbers. There are many examples: student numbers on exams, hand-written spreadsheets, phone numbers, dates etcetera.

Load the data:

```
mnist <- dataset_mnist()
```

Data preparation

Use `array_reshape` and `nrow` to convert the train and test set from from 28x28 pixels to a column of 784 pixels for each image. These should have dimensions 60000x784 and 10000x784 respectively.

```
x_train <- array_reshape(x = mnist$train$x, dim = c(60000, 784))
nrow(x_train)
```

```
## [1] 60000
```

```
ncol(x_train)
```

```
## [1] 784
```

```
x_test <- array_reshape(x = mnist$test$x, dim = c(10000, 784))
nrow(x_test)
```

```
## [1] 10000
```

```
ncol(x_test)
```

```
## [1] 784
```

Divide each variable by 255 to scale them to values between 1 and 0.

```
# Test if dividing whole array leads to division of individual numbers.
(test_array <- 1:30)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
## [24] 24 25 26 27 28 29 30
```

```
(test_array/3)
```

```
##  [1]  0.3333333  0.6666667  1.0000000  1.3333333  1.6666667  2.0000000
##  [7]  2.3333333  2.6666667  3.0000000  3.3333333  3.6666667  4.0000000
## [13]  4.3333333  4.6666667  5.0000000  5.3333333  5.6666667  6.0000000
## [19]  6.3333333  6.6666667  7.0000000  7.3333333  7.6666667  8.0000000
## [25]  8.3333333  8.6666667  9.0000000  9.3333333  9.6666667 10.0000000
```

```
# Works

x_train <- x_train/255
x_test <- x_test/255
```

Use `to_categorical` to convert train and test labels to categories. Each number is represent as an array of nine 0s and one 1.

```
y_train <- to_categorical(mnist$train$y)
y_test <- to_categorical(mnist$test$y)

# Check it out
y_train %>% as_tibble %>% head(5) %>% kable
```

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1

Model definition

The pixels of each 28x28 pixels image have been reshaped into a list of 784 values between 0-1 indicating the activation of each pixels. The list of pixels is fed to 784 input nodes that are fully connected to a 256 node hidden layer. The softmax function transforms the output of each picture to the probability of it being each of the digits 0-9.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, input_shape = c(784)) %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model)
```

Compile the model

```
model %>% compile(
 loss = 'categorical_crossentropy',
 optimizer = optimizer_rmsprop(),
 metrics = c('accuracy')
)
```

Training and evaluation

Question 2

See output at `Xs Yus`/sample where

```
history <- model %>% fit(
 x_train, y_train,
 batch_size = 128,
 epochs = 12,
 verbose = 1,
 validation_split = 0.2
)
```

Make a dataframe of the history and look what is in it.

```
history %>%
  as_tibble %>%
  write_csv(path = "histories/history.csv")
```

Question 3

Plot history

```
history <- read_csv("histories/history.csv")
```

```
## Parsed with column specification:
## cols(
##   epoch = col_integer(),
##   value = col_double(),
##   metric = col_character(),
##   data = col_character()
## )
```

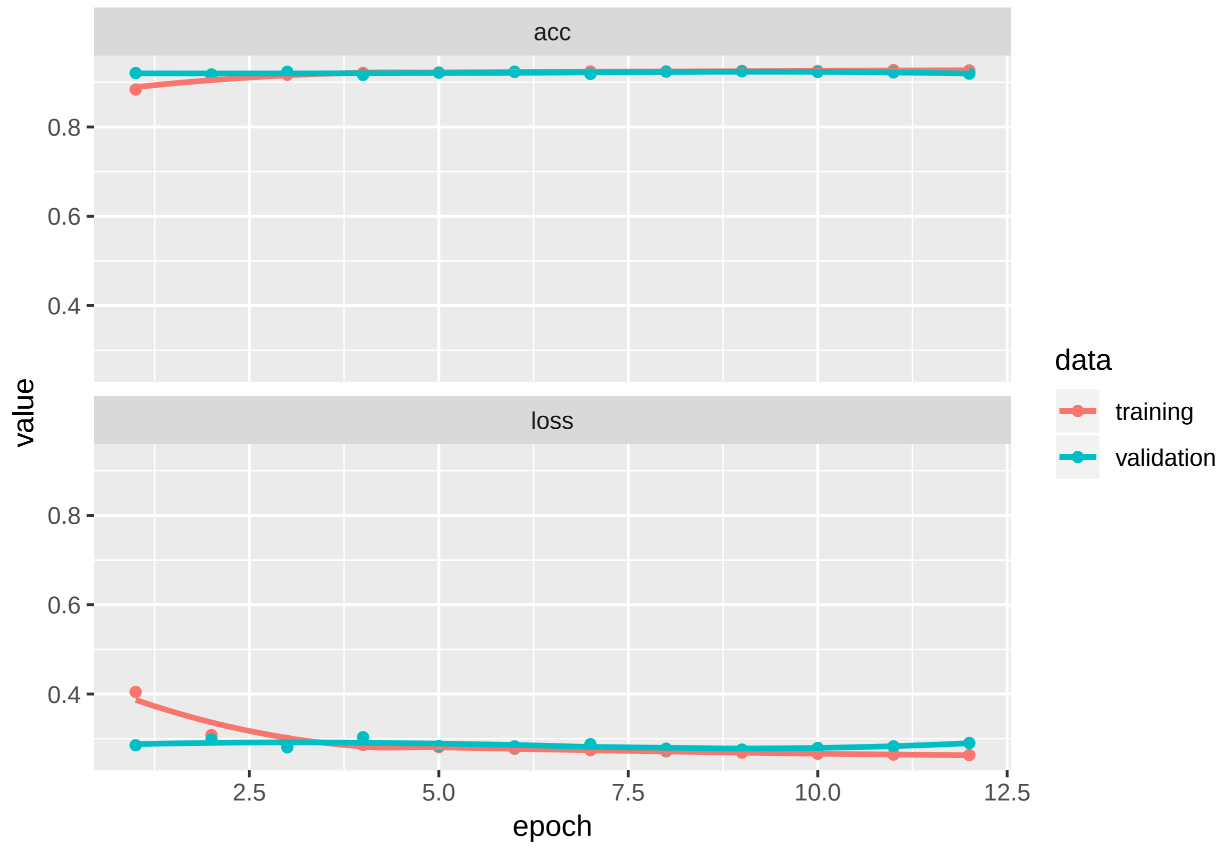
```
history %>% kable
```

epoch	value	metric	data
1	0.4048642	loss	training
2	0.3082619	loss	training
3	0.2952906	loss	training
4	0.2861764	loss	training
5	0.2814019	loss	training
6	0.2774602	loss	training
7	0.2743379	loss	training
8	0.2715181	loss	training
9	0.2688666	loss	training
10	0.2662954	loss	training
11	0.2643621	loss	training
12	0.2633906	loss	training
1	0.8837084	acc	training
2	0.9137500	acc	training
3	0.9167292	acc	training
4	0.9207917	acc	training
5	0.9218541	acc	training
6	0.9230000	acc	training
7	0.9241042	acc	training
8	0.9236042	acc	training
9	0.9252292	acc	training
10	0.9252500	acc	training
11	0.9272083	acc	training
12	0.9266250	acc	training
1	0.2855015	loss	validation
2	0.2981743	loss	validation
3	0.2807852	loss	validation
4	0.3034280	loss	validation
5	0.2838197	loss	validation

epoch	value	metric	data
6	0.2828578	loss	validation
7	0.2877972	loss	validation
8	0.2775686	loss	validation
9	0.2756672	loss	validation
10	0.2793385	loss	validation
11	0.2830024	loss	validation
12	0.2904317	loss	validation
1	0.9206667	acc	validation
2	0.9176667	acc	validation
3	0.9235000	acc	validation
4	0.9165000	acc	validation
5	0.9214166	acc	validation
6	0.9235833	acc	validation
7	0.9187500	acc	validation
8	0.9240834	acc	validation
9	0.9246666	acc	validation
10	0.9231667	acc	validation
11	0.9221666	acc	validation
12	0.9192500	acc	validation

```r
ggplot(history, aes(epoch, value, colour = data)) +
  geom_point() +
  geom_smooth(se = F) +
  facet_wrap(~metric, ncol = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Question 4

The model performs similar on training and out of training data. It generalises well.

Question 5

```
model %>% evaluate(
 x_test, y_test,
 verbose = 0
) %>%
  as_tibble %>%
  write_csv("scores/score.csv")
```

```
score <- read_csv("scores/score.csv")
```

```
## Parsed with column specification:
## cols(
##   loss = col_double(),
##   acc = col_double()
## )
```

```
score %>% kable
```

loss	acc
0.3029577	0.9174

Changing model parameters

Question 6

The accuracy is not high enough for applications that can have big concequences when the highest precision is not achieved. However, for less precise tasks the accuracy may suffice.

Question 7

Keras dense layers use a linear activation function at default. This means that there is no threshold activation. A ReLu function does induce some kind of threshold. I.e. input below a certain value does not activate a neuron. Using this threshold allows one to ignore input that is too small to be relevant, to make the categorisation more efficient. Features of the picture that stand out will be emphasised, while features at the background will be ignored. If node activation is determined by a linear function the prominent features in the picture lack useful emphases.

Question 8

```r
model_relu <- keras_model_sequential()
model_relu %>%
  layer_dense(units = 256, input_shape = c(784), activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

summary(model_relu)
```

```r
model_relu %>% compile(
 loss = 'categorical_crossentropy',
 optimizer = optimizer_rmsprop(),
 metrics = c('accuracy')
)
```

```r
history_relu <- model_relu %>% fit(
 x_train, y_train,
 batch_size = 128,
 epochs = 12,
 verbose = 1,
 validation_split = 0.2
)
```

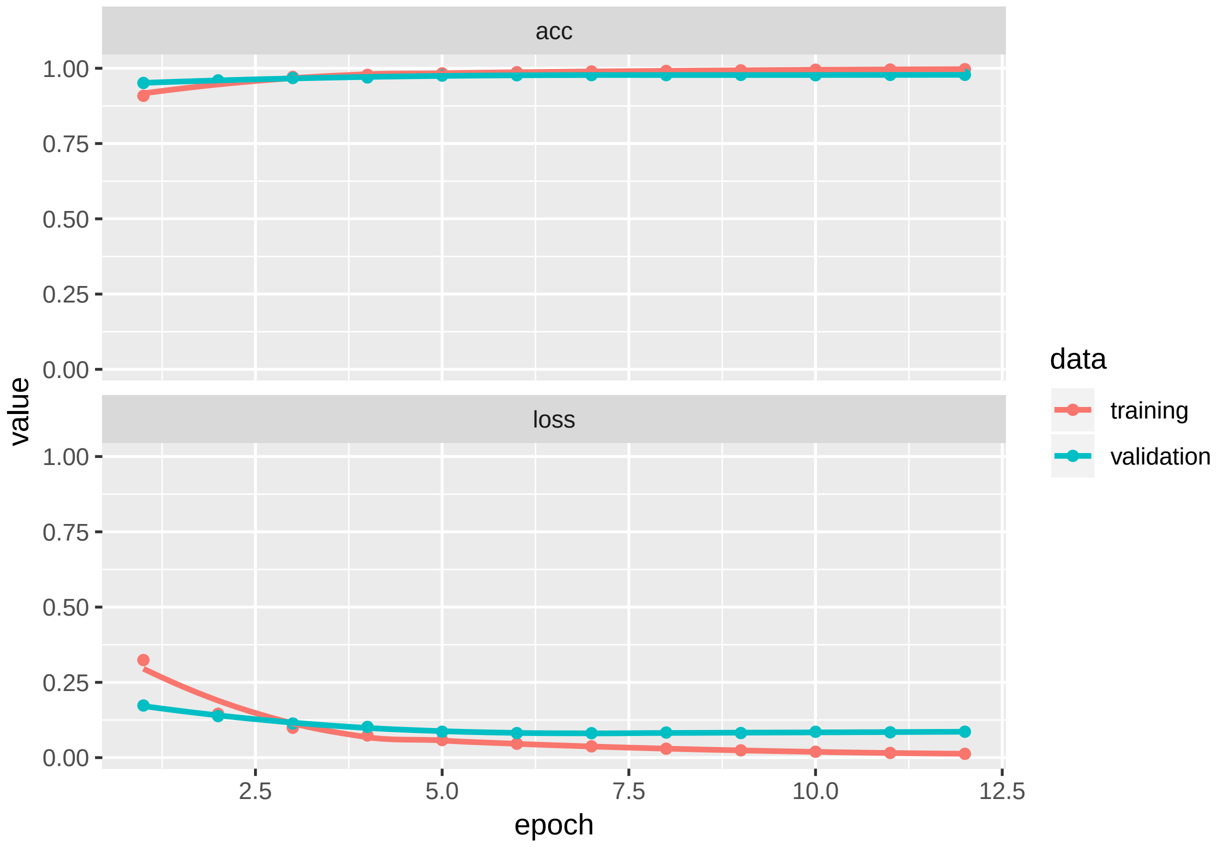
```r
history_relu %>%
  as_tibble %>%
  write_csv(path = "histories/history_relu.csv")
```

```r
history_relu <- read_csv("histories/history_relu.csv")
```

```
## Parsed with column specification:
## cols(
##   epoch = col_integer(),
##   value = col_double(),
##   metric = col_character(),
##   data = col_character()
## )
```

```r
ggplot(history_relu, aes(epoch, value, colour = data)) +
  geom_point() +
  geom_smooth(se = F) +
  facet_wrap(~metric, ncol = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Question 9

The performance on the validation set is a lot lower than the performance on the trainingset. Probably this model does not generalise as well as the previous model.

Question 10

Probably this will perform worse on the test set than the previous model. This can be expected from the performance on the validation set.

```
score_relu <- model_relu %>% evaluate(
 x_test, y_test,
 verbose = 0
) %>%
  as_tibble %>%
  write_csv("scores/score_relu.csv")
```

```
score_relu <- read_csv("scores/score_relu.csv")
```

```
## Parsed with column specification:
## cols(
##   loss = col_double(),
##   acc = col_double()
## )
```

```
score_relu %>% kable
```

loss	acc
0.0779087	0.9801

Reshape `mnist$train$x` to a new variable (x_train) of size 60000, 28, 28, 1. Reshape `mnist$test$x` to a new variable (x_test) of size 10000, 28, 28, 1. Rescale both results to values between zero and one as before.

```
x_train <- array_reshape(x = mnist$train$x, dim = c(60000, 28, 28, 1))
nrow(x_train)
```

```
## [1] 60000
```

```
ncol(x_train)
```

```
## [1] 28
```

```
x_test <- array_reshape(x = mnist$test$x, dim = c(10000, 28, 28, 1))
nrow(x_test)
```

```
## [1] 10000
```

```
ncol(x_test)
```

```
## [1] 28
```

```
x_train <- x_train/255
x_test <- x_test/255

y_train <- to_categorical(mnist$train$y)
y_test <- to_categorical(mnist$test$y)
```

Deep Convolutional Neural Net:

- 2 convolutional layers
- 32 convolutional filters into the first layer
- 64 convulational filters into the second
- 3x3 pixel filters
- ReLu activation
- Pooling to downsample size 2nd layer 2 a quarter of the pixels.
- Flatten to 1-dimensional array and use fully-connected layer to link network to labels.

```
model_cnn <- keras_model_sequential() %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu', input_shape = c(28,28,1)) %>%
 layer_conv_2d(filters = 64, kernel_size = c(3,3),
 activation = 'relu') %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_flatten() %>%
 layer_dense(units = 128, activation = 'relu') %>%
 layer_dense(units = 10, activation = 'softmax')

summary(model_cnn)

model_cnn %>% compile(
 loss = 'categorical_crossentropy',
 optimizer =  optimizer_adadelta(),
 metrics = c('accuracy')
)
```

```
history_cnn <- model_cnn %>% fit(
 x_train, y_train,
 batch_size = 128,
 epochs = 6,
 verbose = 1,
 validation_split = 0.2
)
```

```
history_cnn %>%
  as_tibble %>%
  write_csv(path = "histories/history_cnn.csv")
```

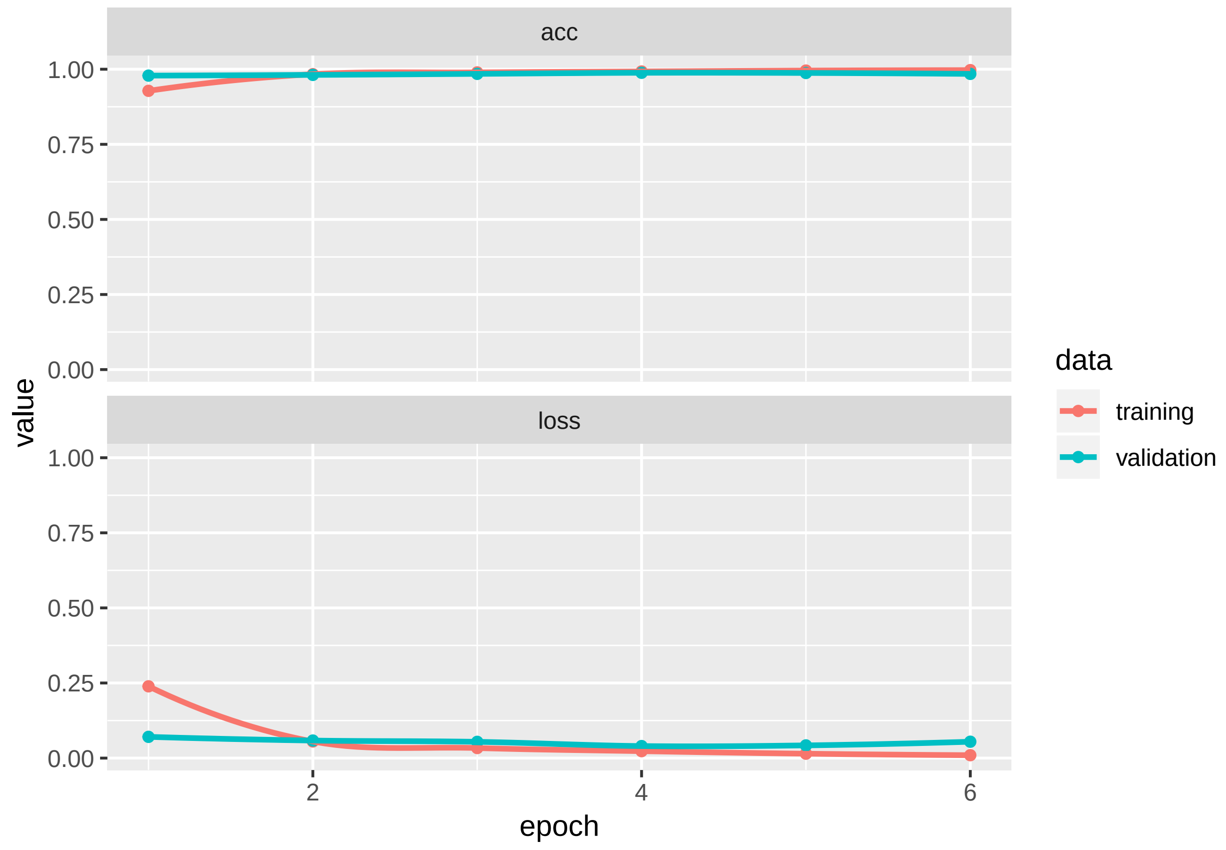
Question 11

```
history_cnn <- read_csv("histories/history_cnn.csv")
```

```
## Parsed with column specification:
## cols(
##   epoch = col_integer(),
##   value = col_double(),
##   metric = col_character(),
##   data = col_character()
## )
```

```
ggplot(history_cnn, aes(epoch, value, colour = data)) +
  geom_point() +
  geom_smooth(se = F) +
  facet_wrap(~metric, ncol = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Question 12

The performance on the trainingset and the validationset are very similar for this model. It is generalises better than the previous model, because it performs similar on training data and out of training data.

Question 13

```
score_cnn <- model_cnn %>% evaluate(
 x_test, y_test,
 verbose = 0
) %>%
  as_tibble %>%
  write_csv("scores/score_cnn.csv")
```

```
score_cnn <- read_csv("scores/score_cnn.csv")
```

```
## Parsed with column specification:
## cols(
##   loss = col_double(),
##   acc = col_double()
## )
```

```
score_cnn %>% kable
```

loss	acc
0.0459091	0.9867

Question 14

The accuracy is sufficient for automatic hand-written digit classification in applications for which 1 mistake in a 100 digits is acceptable. For example postal codes, phone numbers, age etcetera.

Question 15

Applying dropout in the trainingstage means that nodes are dropped with a certain probability. In this manner the chances of good performance on training data and worse performance on test data (overfitting) are reduced.

```r
model_cnn2 <- keras_model_sequential() %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu', input_shape = c(28,28,1)) %>%
 layer_conv_2d(filters = 64, kernel_size = c(3,3),
 activation = 'relu') %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_dropout(rate = .5) %>%
 layer_flatten() %>%
 layer_dense(units = 128, activation = 'relu') %>%
 layer_dense(units = 10, activation = 'softmax')

summary(model_cnn2)

model_cnn2 %>% compile(
 loss = 'categorical_crossentropy',
 optimizer =  optimizer_adadelta(),
 metrics = c('accuracy')
)
```

```r
history_cnn2 <- model_cnn2 %>% fit(
 x_train, y_train,
 batch_size = 128,
 epochs = 6,
 verbose = 1,
 validation_split = 0.2
)
```

```r
history_cnn2 %>%
  as_tibble %>%
  write_csv(path = "histories/history_cnn2.csv")
```

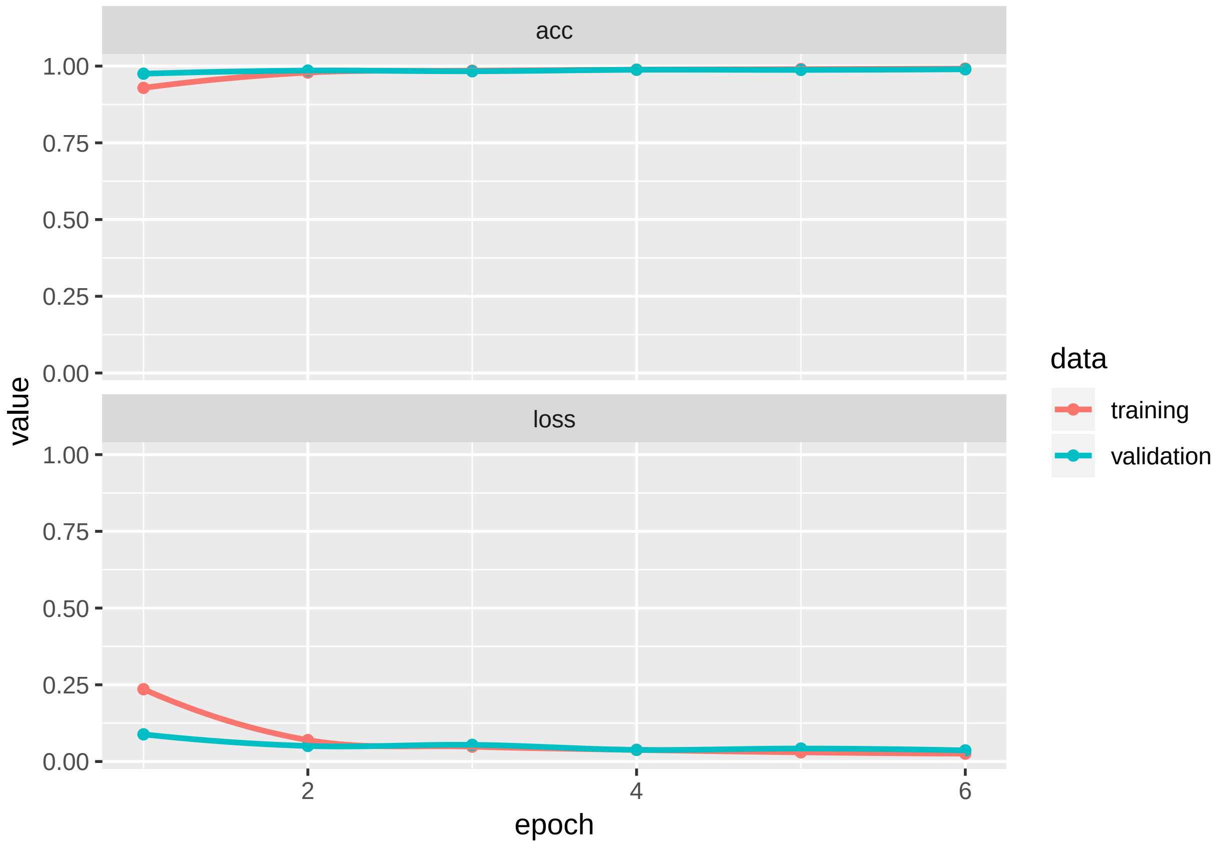
Question 16

```r
history_cnn2 <- read_csv("histories/history_cnn2.csv")
```

```
## Parsed with column specification:
## cols(
##   epoch = col_integer(),
##   value = col_double(),
##   metric = col_character(),
##   data = col_character()
## )
```

```r
ggplot(history_cnn2, aes(epoch, value, colour = data)) +
  geom_point() +
  geom_smooth(se = F) +
  facet_wrap(~metric, ncol = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



History of the training and validation sets are even closer than in the previous models. Training time was about 10 seconds longer on average.

Question 17

The models should generalise well, the 2nd one slightly better than the first.

Exercise two: Identifying objects from images

Prepare data

```
cifar10 <- dataset_cifar10()
```

```
x_train <- cifar10$train$x/255
x_test <- cifar10$test$x/255

y_train <- to_categorical(cifar10$train$y)
y_test <- to_categorical(cifar10$test$y)
```

Question 18

Define the model

```
model_im <- keras_model_sequential() %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu', input_shape = c(32,32,3)) %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu') %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_dropout(rate = .5) %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu') %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3),
 activation = 'relu') %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_dropout(rate = .5) %>%
 layer_flatten() %>%
 layer_dense(units = 512, activation = 'relu') %>%
 layer_dense(units = 10, activation = 'softmax')

summary(model_im)

model_im %>% compile(
 loss = 'categorical_crossentropy',
 optimizer = optimizer_rmsprop(lr = 0.0001, decay = 1e-6),
 metrics = c('accuracy')
)
```

```
history_im <- model_im %>% fit(
 x_train, y_train,
 batch_size = 32,
 epochs = 20,
 verbose = 1,
 validation_data = list(x_test, y_test),
 validation_split = 0.2,
 shuffle = TRUE
)
```

```
history_im %>%
  as_tibble %>%
  write_csv(path = "histories/history_im.csv")
```

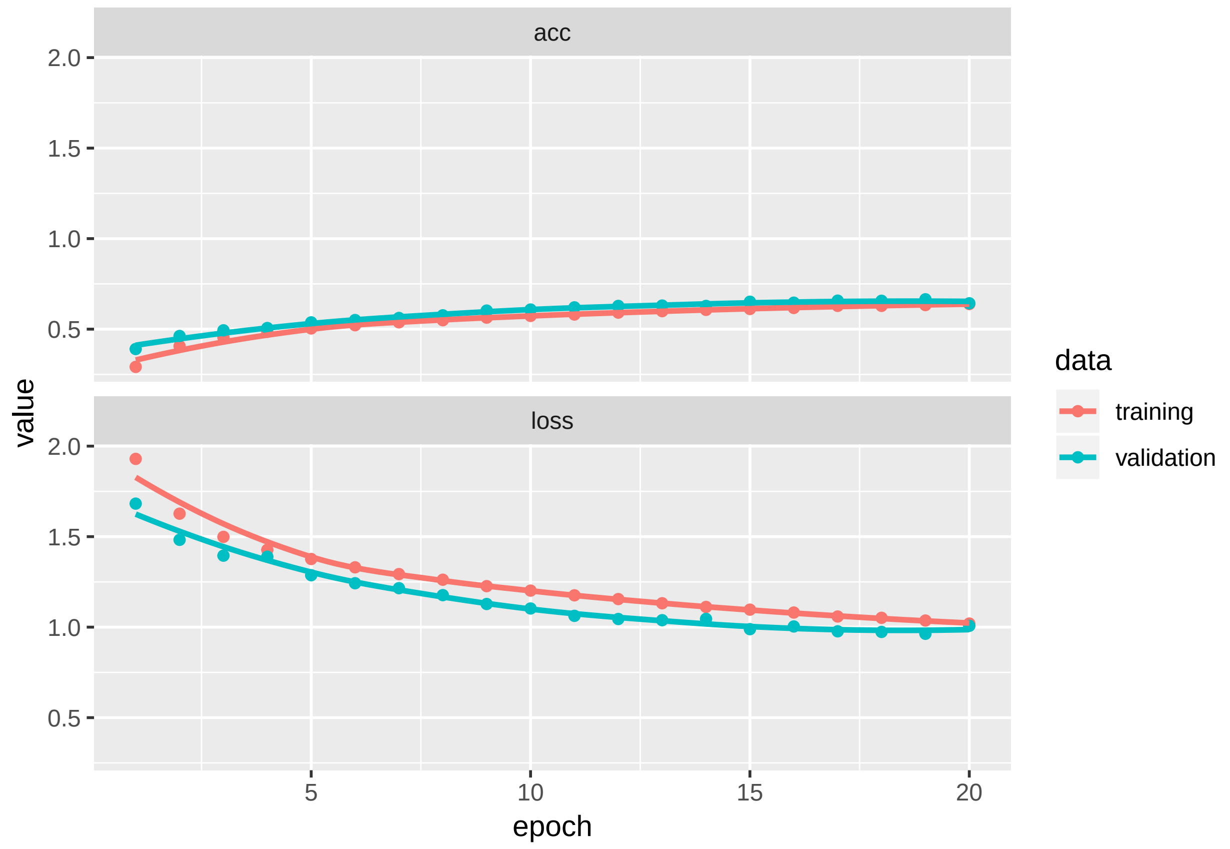
Question 19

```
history_im <- read_csv("histories/history_im.csv")
```

```
## Parsed with column specification:
## cols(
##   epoch = col_integer(),
##   value = col_double(),
##   metric = col_character(),
##   data = col_character()
## )
```

```
ggplot(history_im, aes(epoch, value, colour = data)) +
  geom_point() +
  geom_smooth(se = F) +
  facet_wrap(~metric, ncol = 1)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Question 19

```
score_im <- model_im %>% evaluate(
 x_test, y_test,
 verbose = 0
) %>%
  as_tibble %>%
  write_csv("scores/score_im.csv")
```

```
score_im <- read_csv("scores/score_im.csv")
```

```
## Parsed with column specification:
## cols(
##   loss = col_double(),
##   acc = col_double()
## )
```

```
score_im %>% kable
```

loss	acc
1.007252	0.6429

Question 20

The accuracy and loss function seem to take more epochs before they plateau. Perhaps this is because more information is used for the categorisation and it takes longer to find patterns in more information.

Question 21

Each epoch took more than 2 minutes. The network has more layers, and more imput nodes, and many more connections. More calculations have to done as more information needs to be processed.

Question 22

Goal

Make models that predict IT neuron responses.

Experimental approach

Optimise top-level functional performance of a hierarchical neural net on a image-recognition task.

Results

Directed optimisation of the model was highly correlated with IT predictivity, even though the neural data was not used to optimise the model.

Exercise three: Play time

Question 23

Settings:

- Learning rate:
  - With what rate are the weights adapted? How large is the effect of the error on each weight. You don't want the learning rate to be to high; every mistake will bring about large changes in the network. If the learning rate is very low however, it takes the network longer to adapt to the feedback.
- Activation:
  - The activation function. Defines relation between input of a neuron and the output. A linear relation leads to an increase in output equal to increase in input. ReLu, Sigmoid, or Tanh, impose certain thresholds. The input needs to exceed this threshold to lead to activation of the neuron. Sigmoid and tanh have a upper limit for the input strength to lead to activation of the neuron, in addition to the lower threshold.
- Regularization:
  - Kind of smooths the model prediction. Reduces the variability of the model and consequently can prevent overfitting.
- Regularization rate:
  - How much the model is regularised.
- Problem type:
  - Classification/regression
- Ratio of training to test data:

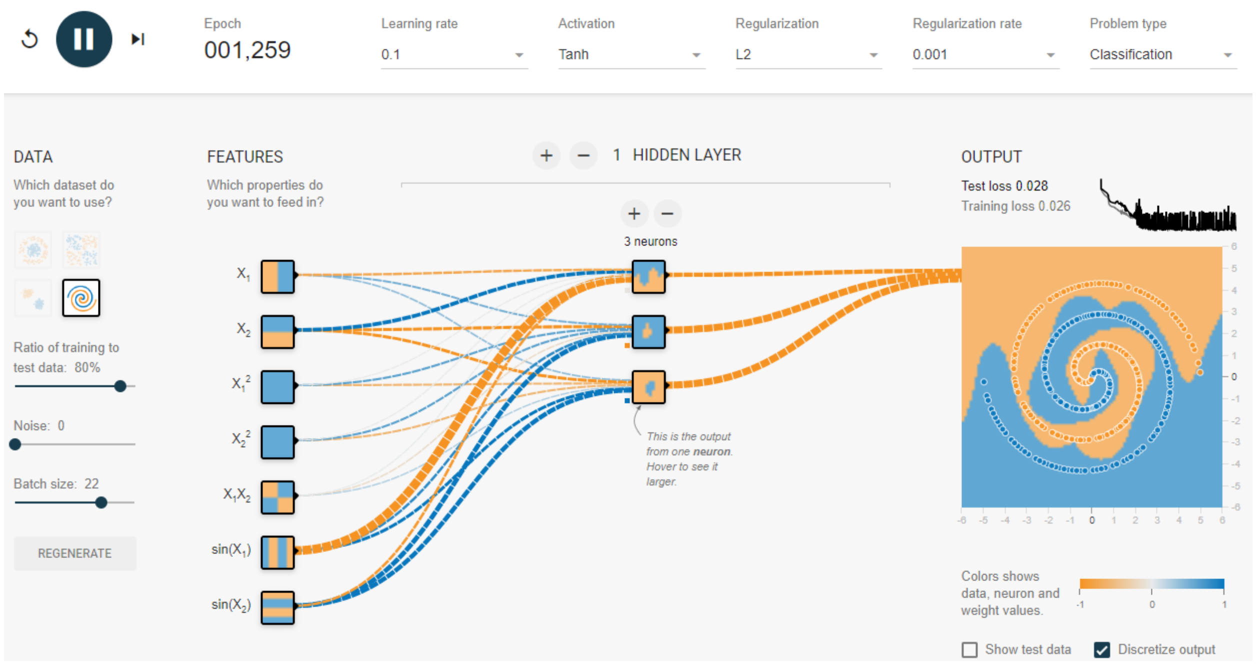

Epoch 001,259
Learning rate 0.1
Activation Tanh
Regularization L2
Regularization rate 0.001
Problem type Classification
Ratio of training to test data: 80%
Noise: 0
Batch size: 22
1 HIDDEN LAYER
3 neurons
Test loss 0.028
Training loss 0.026
DATA
Which dataset do you want to use?
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
This is the output from one neuron. Hover to see it larger.
OUTPUT
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Figure 1: Spiral recognition model

  - How much of the data should be used for training the model and how much should be used to test the model?
- Noise:
  - How large should the irreducible error of the data be?
- Batch size:
  - The number of datapoints that are used per iteration to train the network.
- Input features:
  - What features do we use to categorise the x and y coordinates?
- Hidden layers:
  - How many hiddenlayers do we use.
- n neurons:
  - How many neurons does each layer have?

Question 24

The minimum spiral recognition network to obtain a test loss of $< .1$ that I tested so far consisted of all features and a hidden layer of 3 neurons. I tweaked the learning rate on the go. At first it was 0.1, once a low loss rate was achieved I reduced the learning rate to 0.01 to create a stable network that would not change much anymore in reaction to the error.