7. 합성곱 신경망

인공 신경망은 파라미터만 잘 조정해주면 매우 다양한 함수를 나타낼 수 있다. 그래서 우리가 원하는 결과를 얻기 위해 모형의 형태에 제약을 주기도 한다. 이러한 방법 중에 대표적인 것으로 합성곱 신경망과 순환신경망이 있다.

이미지를 기계학습으로 처리할 경우, 이미지의 점 하나가 변수가 된다. 가로 28 X 세로 28인 이미지가 있다면, 이 이미지는 784개의 점으로 이뤄져 있다. 즉 784개의 변수로된 데이터를 분석하는 것과 같게 된다.

이미지는 점들이 모여 작은 형태를 이루고, 작은 형태들이 모여 더 큰 형태를 이루는 특징이 있다. 따라서 이미지들을 단순히 여러 개의 점들로 보는 것보다, 이런 점들이 모여 만드는 형태들을 모형에 포함시킨다면 이미지 기계학습의 성능을 향상시킬 수 있다. 실제로 생물의 시각도 같은 방식으로 작동한다. 생물에서 작은 영역에 반응하는 세포들을 receptive field라고 부른다. 여기에서 착안한 것이 합성곱 신경망(convolutional neural network, CNN)이다.

7.1. 합성곱

합성곱(convolution)은 원래 이미지 처리에 많이 사용되는 방법이다.예를 들어 다음과 같은 이미지에

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

아래와 같은 필터를

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

합성곱 해보자. 위의 필터는 수직선에 반응하는 비슷한 기능을 한다.

계산은 다음과 같이 이뤄진다. 먼저 왼쪽 위의 3x3 부분만을 본다.

$$\begin{bmatrix} 1 & 0 & \cdot \\ 1 & 0 & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

그 다음 같은 위치에 있는 값들끼리 곱해준다.

$$\begin{bmatrix} 1 \times 1 & -1 \times 0 \\ 1 \times 1 & -1 \times 0 \end{bmatrix}$$

그러면 계산 결과가 다음과 같다.

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

마지막으로 이들의 합을 구하면 2가 된다.

다음으로는 한 칸 오른쪽으로 옮겨서 다음 3x3 부분에

$$\begin{bmatrix} \cdot & 0 & 0 \\ \cdot & 0 & 0 \\ \cdot & \cdot & \cdot \end{bmatrix}$$

똑같은 계산을 적용하면 0이 된다. 이렇게 계산을 반복하고 그 결과를 아래와 같은 feature map으로 만든다.

$$\begin{bmatrix} 2 & 0 \\ 1 & 0 \end{bmatrix}$$

위의 행렬을 보면 그림의 왼쪽 위가 수직선 패턴이 가장 강하다는 것을 알 수 있다. 이번에는 수평선 패턴에 반응하는 아래와 같은 필터를

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

똑같은 이미지에 합성곱을 하면 다음과 같은 feature map을 만들 수 있다.

$$\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}$$

위의 행렬을 보면 그림의 오른쪽 아래가 수평선 패턴이 가장 강하다는 것을 알 수 있다.

이런 식으로 다양한 필터를 만들어 합성곱을 해주면 이미지에서 특정 패턴을 추출할 수 있다. 합성곱은 패턴 추출에만 사용되는 것이 아니고 흐리게

하기(blur)나 날카롭게 하기(sharpen) 등 이미지 편집에 사용되는 여러 필터 적용에도 쓸 수 있다.

합성곱 신경망에서 필터 역할을 하는 레이어를 합성곱 레이어라고 한다. 기존의 이미지 처리가 미리 만들어진 필터를 적용하는 방식이라면, CNN은 이러한 필터 자체를 학습한다는 것이 특징이다. 필터라는 것은 결국 그림의 일부에 일정한 계수를 곱해주고 그 결과를 더하는 것이로 일반적인 신경망의 구조와 다를 것이 없다. 다만, 같은 계수를 그림의 부분들에 반복적으로 적용한다는 점만 차이가 있을 뿐이다.

7.2. Pooling

합성곱을 하면 feature map에서 근처의 값들끼리 비슷해지는 경향을 띈다. 그래서 근방의 영역에 있는 값들을 묶어 하나의 값으로 줄이는 pooling을 한다. pooling의 방법에는 다음과 같은 것들이 있다.

- max: 영역의 최대값
- average: 영역의 평균
- L2: 영역의 제곱합의 제곱근

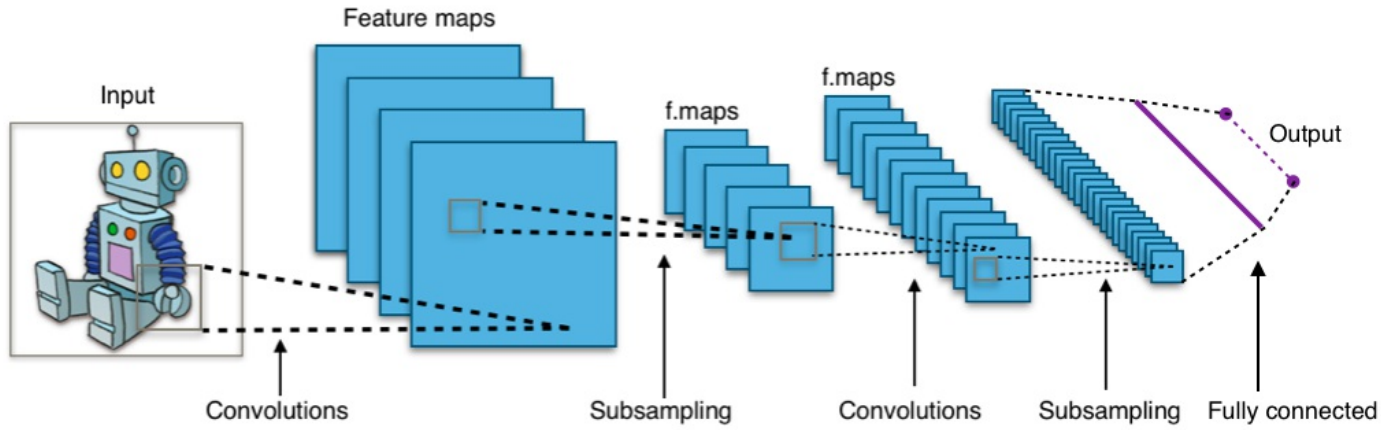
앞에서 예로 들었던 feature map에 2x2 max pooling을 적용하면 똑같이 2가 나오게 된다.

따라서 우리는 정확한 위치는 알 수 없지만 3x3 그림에 수직선과 수평선이 하나 있다는 것을 알 수 있게 된다. 이러한 pooling은 정보를 줄이게 되지만, 과적합도 줄어들 뿐만 아니라 그림이 일부 변형되더라도 학습결과를 유지할 수 있게 해준다. 왜냐하면 그림이 수직이나 수평 방향으로 1픽셀 이동하더라도 pooling을 거친 결과는 동일하기 때문이다.

합성곱 레이어와 달리 풀링 레이어는 별도의 학습이 이뤄지지 않는다.

7.3. 합성곱 신경망

이제 이미지를 입력 받아 그 위로 합성곱 레이어와 풀링 레이어를 반복해서 쌓으면 CNN이 된다. 두 가지 레이어를 반복해서 쌓는 이유는 작은 형태를 바탕으로 다시 큰 형태를 처리하게 하기 위해서다.



7.4. 실습

- `keras` 홈페이지: https://keras.io
- R interface to Keras 홈페이지: https://keras.rstudio.com

케라스를 불러들인다.

```
library(keras)
```

7.4.1. MNIST 데이터

MNIST 손글씨 데이터를 불러온다. 해당 데이터는 케라스에 내장되어 있다.

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

훈련용 데이터와 테스트용 데이터를 불러온다. x는 손글씨 이미지, y는 그 이미지에 해당하는 숫자이다.

```
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

x는 가로 28, 세로 28의 이미지 6만장이다.

```
dim(x_train)
```

```
[1] 60000    28    28
```

x_train과 y_train을 28*28*1의 형태로 바꿔준다. 컬러 이미지의 경우에는 RGB 세 가지 색을 쓰기 때문에 28 *28* 3의 형태가 되어야 한다. MNIST는 흑백이므로 1이 된다.

```
dim(x_train) <- c(60000, 28, 28, 1)
```

```
dim(x_test) <- c(10000, 28, 28, 1)
```

x의 각 점은 0~255범위의 값을 갖는다. 수치가 너무 크기 때문에 255로 나눠서 0~1까지 범위로 바꿔준다.

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

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

y는 6만개의 숫자다.

```
dim(y_train)
```

```
[1] 60000
```

y의 첫 10개 값을 보면 숫자라는 것을 알 수 있다.

```
y_train[1:10]
```

```
[1] 5 0 4 1 9 2 1 3 1 4
```

one-hot encoding을 적용한다.

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

이제 y는 10개의 변수가 6만개 있는 형태로 바뀐다.

```
dim(y_train)
```

```
[1] 60000    10
```

y의 앞부분을 보면 10개의 변수가 순서대로 0~9까지 값을 가리킨다.

```
head(y_train)
```

```
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 0    0    0    0    0    1    0    0    0    0
[2,] 1    0    0    0    0    0    0    0    0    0
[3,] 0    0    0    0    1    0    0    0    0    0
[4,] 0    1    0    0    0    0    0    0    0    0
[5,] 0    0    0    0    0    0    0    0    0    1
[6,] 0    0    1    0    0    0    0    0    0    0
```

7.4.2. 앞먹임 신경망

이제 앞먹임 신경망 모형을 만든다. 앞먹임 신경망은 벡터 형태의 입력을 받는다. 이미지 데이터를 변환할 수도 있지만 데이터의 형태를 바꿔주는 `layer_reshape`을 추가해서 신경망 내에서 형태가 바뀌도록 해준다.

`softmax`는 `sigmoid`의 확장버전으로 여러 개의 출력을 0~1범위로 하고, 출력의 합이 항상 1이 된다. y가 one-hot encoding 되어 있으므로 이 경우에는 각각의 숫자일 확률을 나타낸다고 해석할 수 있다.

```
ff = keras_model_sequential()
ff %>%
  layer_reshape(28 * 28, input_shape=c(28, 28, 1)) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

summary(ff)
```

```
_____________________________________________________________________________
Layer (type)                     Output Shape                  Param #
=============================================================================
reshape_3 (Reshape)              (None, 784)                   0
_____________________________________________________________________________
dense_7 (Dense)                  (None, 256)                   200960
_____________________________________________________________________________
dense_8 (Dense)                  (None, 10)                    2570
=============================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_____________________________________________________________________________
```

선택지가 여러 개이므로 `binary_crossentropy` 대신 `categorical_crossentropy`를 적용한다.

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

학습을 시킨다. 검증 데이터로 손실을 측정해서 그 손실이 이전보다 감소하지 않으면 학습을 중단케 하여 과적합을 방지한다.

```
history = ff %>%  fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2,
  callbacks = c(callback_early_stopping(monitor = "val_loss"))
)
```

기록을 확인해보면 30에포크 보다 일찍 학습이 중단된 것을 볼 수 있다. 왜냐하면 마지막에 검증 데이터의 손실이 증가(정확도가 감소)했기 때문이다.

```
history$metrics$val_loss
```

```
[1] 0.17681224 0.12036056 0.10137217 0.09321289 0.08685864 0.08547075 0.08202487
[8] 0.08324006
```

```
history$metrics$val_acc
```

```
[1] 0.9600833 0.9750833 0.9791667 0.9789167 0.9823333 0.9838333 0.9845833
[8] 0.9829167
```

7.4.3. CNN

이번에는 CNN을 만들어보자. `layer_flatten`은 `layer_reshape`과 비슷하지만 무조건 1차원 벡터로 만든다. 앞의 예에서도 어차피 1차원 벡터로 만들 것이었기 때문에 똑같이 `layer_flatten`을 써도 된다.

```
cnn = keras_model_sequential()
cnn %>%
  layer_conv_2d(32, 3, activation='relu', input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

summary(cnn)
```

```
_________________________________________________________________________________
Layer (type)                         Output Shape                     Param #
=================================================================================
conv2d_11 (Conv2D)                   (None, 26, 26, 32)               320
_________________________________________________________________________________
max_pooling2d_5 (MaxPooling2D)       (None, 13, 13, 32)               0
_________________________________________________________________________________
flatten_2 (Flatten)                  (None, 5408)                     0
_________________________________________________________________________________
dense_13 (Dense)                     (None, 32)                       173088
_________________________________________________________________________________
dense_14 (Dense)                     (None, 10)                       330
=================================================================================
Total params: 173,738
Trainable params: 173,738
Non-trainable params: 0
_________________________________________________________________________________
```

나머지는 앞먹임 신경망과 다르지 않다.

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

history = cnn %>%  fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2,
  callbacks = c(callback_early_stopping(monitor = "val_loss"))
)
```

손실과 정확도를 확인한다.

```
history$metrics$val_loss
```

```
[1] 0.14397980 0.09219768 0.07285122 0.07239736 0.06129888 0.05795515 0.05627563
[8] 0.05858986
```

```
history$metrics$val_acc
```

```
[1] 0.9600833 0.9750833 0.9791667 0.9789167 0.9823333 0.9838333 0.9845833
[8] 0.9829167
```

7.4.4. caret에서 dummy coding (one-hot encoding)

```
library(caret)
```

```
Loading required package: lattice
Loading required package: ggplot2
Warning message:
: package 'ggplot2' was built under R version 3.3.3
```

내장된 scat 데이터를 가져온다.

```
data(scat)

head(scat)
```

```
Species   Month Year Site Location Age Number Length Diameter Taper   TI
1  coyote January 2012 YOLA     edge   5      2    9.5     25.7  41.9 1.63
2  coyote January 2012 YOLA     edge   3      2   14.0     25.4  37.1 1.46
3  bobcat January 2012 YOLA   middle   3      2    9.0     18.8  16.5 0.88
4  coyote January 2012 YOLA   middle   5      2    8.5     18.1  24.7 1.36
6  coyote January 2012 YOLA     edge   5      4    8.0     20.7  20.1 0.97
7  coyote January 2012 YOLA     edge   5      3    9.0     21.2  28.5 1.34
   Mass   d13C d15N   CN ropey segmented flat scrape
1 15.89 -26.85 6.94  8.5     0         0    0      0
2 17.61 -29.62 9.87 11.3     0         0    0      0
3  8.40 -28.73 8.52  8.1     1         1    0      1
4  7.40 -20.07 5.79 11.5     1         0    0      0
6 25.45 -23.24 7.01 10.6     0         1    0      0
7 14.14 -29.00 8.28  9.0     1         0    0      0
```

`dummyVars` 함수를 이용해 Species 변수의 더미 코딩 규칙을 만든다.

```
dmv = dummyVars('~ Species', scat)
```

규칙을 적용해 더미 코딩을 한다.

```
species = predict(dmv, scat)

head(species)
```

```
Species.bobcat Species.coyote Species.gray_fox
1              0              1                0
2              0              1                0
3              1              0                0
4              0              1                0
6              0              1                0
7              0              1                0
```

한 번에 더미 코딩을 하는 대신 규칙을 만들고 적용하는 방식으로 단계를 나누어 하는 이유는 새로운 데이터가 들어왔을 때 동일한 규칙으로 더미 코딩을 하기 위해서다.

다음으로 모든 변수를 더미 코딩해보자. 이때는 '~ .'이라고 쓰면 된다.

```
dmv.all = dummyVars('~ .', scat)

all.dummy = predict(dmv.all, scat)

head(all.dummy)
```

```
Species.bobcat Species.coyote Species.gray_fox Month.April Month.August
1 "0"            "1"            "0"              "0"         "0"
2 "0"            "1"            "0"              "0"         "0"
3 "1"            "0"            "0"              "0"         "0"
4 "0"            "1"            "0"              "0"         "0"
6 "0"            "1"            "0"              "0"         "0"
7 "0"            "1"            "0"              "0"         "0"
  Month.February Month.January Month.June Month.May Month.November ...   Taper
1 "0"            "1"           "0"        "0"       "0"            "..." "41.9"
2 "0"            "1"           "0"        "0"       "0"            "..." "37.1"
3 "0"            "1"           "0"        "0"       "0"            "..." "16.5"
4 "0"            "1"           "0"        "0"       "0"            "..." "24.7"
6 "0"            "1"           "0"        "0"       "0"            "..." "20.1"
7 "0"            "1"           "0"        "0"       "0"            "..." "28.5"
  TI     Mass    d13C     d15N   CN     ropey segmented flat scrape
1 "1.63" "15.89" "-26.85" "6.94" "8.5"  "0"   "0"       "0"  "0"
2 "1.46" "17.61" "-29.62" "9.87" "11.3" "0"   "0"       "0"  "0"
3 "0.88" "8.4"   "-28.73" "8.52" "8.1"  "1"   "1"       "0"  "1"
4 "1.36" "7.4"   "-20.07" "5.79" "11.5" "1"   "0"       "0"  "0"
6 "0.97" "25.45" "-23.24" "7.01" "10.6" "0"   "1"       "0"  "0"
7 "1.24" "14.14" "-28"    "8.28" "8"    "1"   "0"       "0"  "0"
```