1. 딥러닝의 기초와 자료구조

Hosik Choi

Dept. of Applied Statistics, Kyonggi Univ.

June, 2019

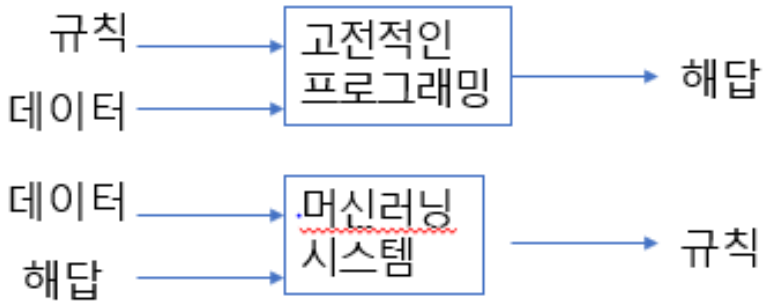
Outline

1. 딥러닝의 소개(머신러닝 역사)

2. 자료구조: 스칼라/벡터/행렬/텐서
   - 연습: 우편번호 다중클래스 분류

3. 실습
   - 실습1: 이진분류
   - 실습2: 회귀분석

딥러닝 $\subset$ 머신러닝 $\subset$ 인공지능

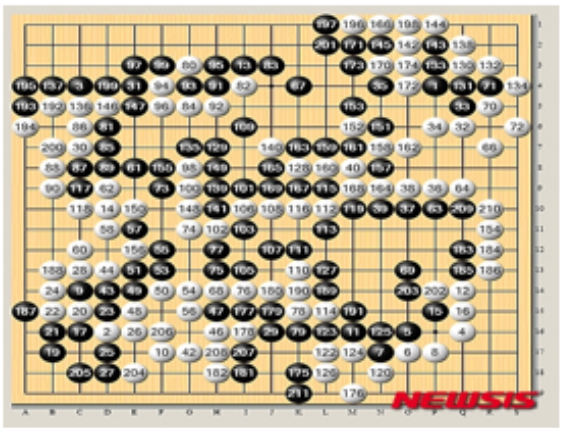


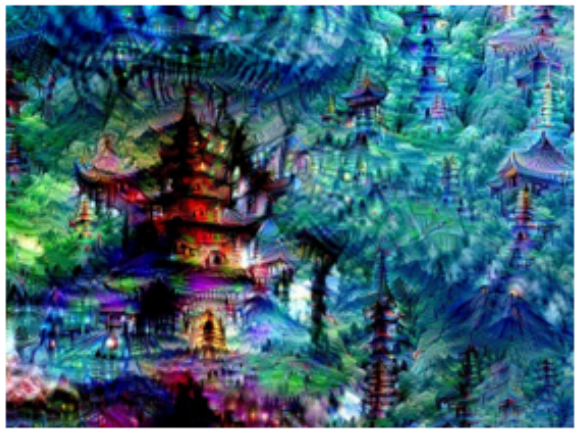
- 딥러닝의 등장
  - 2006년 Science 저널에 발표된 토론토 대학의 Hinton 교수의 논문 이후로 활발하게 관심 받기 시작
  - MIT에서 선정한 2013년을 빛낼 10대 혁신 기술 중 하나로 선정
  - 가트너가 2014년 세계 IT 시장 10대 주요 기술로 예측
  - 니혼게이자이신문 2017년 신년 특집호에서 "올해는 딥러닝 등 최신 AI 위력을 누구나 실감할 수 있는 한 해가 될 것"이라고 관측

- 딥러닝 대가들의 글로벌 업체로의 이동
  - Geoffery Hinton: 캐나다 토론토 대학교 → 구글
  - Yann Lecun: 뉴욕 대학교 교수 겸 페이스북 인공지능 연구소장
  - Andrew Ng: 스탠포드 대학교 교수 겸 바이두 연구소 최고 과학자
  - Yoshua Bengio: Element AI 창업

딥러닝 응용분야 I

- 이미지 분석
  - 이미지 인식, 압축, 복원, 생성 등
  - AlphaGo
  - DeepDream (그림을 그리는 인공지능)
  - 무인자동차





- 언어 분석
  - 번역, Q & A 등
  - Google 번역기, 챗봇 등

- 음성 인식, 복원, 생성, 분해 등등
  - 인공지능 스피커, STT (Speech to Text)

딥러닝 입문

다양한 딥러닝 아키텍처 (Architecture, e.g. Model)

- Fully-connected Neural Network: Basic building block
- Deep Belief Network: Generative model
  - e.g. Factor model
- Stacked Auto-Encoder: Dimension reduction and feature learning
  - e.g. Principle component analysis
- Convolutional Neural Network(CNN): Neural network specialized for image recognition
- Recurrent Neural Network(RNN): Neural network specialized for sequence data
  - e.g. 언어, 음성 등

프로그래밍 언어

- 딥러닝을 위해 많이 활용되는 소프트웨어
  - tensorflow: 구글 개발, 현재 가장 많은 사람이 사용
  - Caffe: UC Berkeley에서 관리
  - Theano: 딥러닝 알고리즘을 파이썬으로 쉽게 구현할 수 있도록 해줌
  - torch: 페이스북과 구글 딥마인드가 사용
  - CNTK: 마이크로소프트 개발
  - Matlab: 상용 소프트웨어
  - R: 통계분석, R은 다양한 종류의 통계분석 기능과 세련된 시각화 기능이 있을 뿐만 아니라, 사용의 편리함 때문에 데이터 과학자 및 빅데이터 분석전문가에게 인기. 다수의 딥러닝 관련 패키지를 지원하며 R을 사용하여 딥러닝 및 기계학습 관련 업무를 수행하는 경우가 많아지고 있음.

주요 성능 개선 결과

- 컴퓨터 비전:
  - 얼굴인식 DeepID2: 얼굴 인식 데이터셋 LFW(labeled fraces in the wild)에서 99.47%
- 광학 문자 인식(optical character recognition, OCR) 분야
  - MNIST 자료 99.77%
- 구글의 디지털 인식 시스템(구글맵에 적용): 구글 스트릿뷰에서 SVHN 데이터에서 96%의 정확도
- 음성인식
  - TIMIT 자료 GMM의 오류율을 21.7%에서 17.9%로 개선

자료구조 I

용어

- Batch: 학습 데이터 전체
- Mini-batch: 학습 데이터의 일부
- Epoch: 반복적인 학습 알고리즘을 사용할 때 모든 학습 데이터를 한 번 씩 사용하는 것을 의미.
  - (ex: 10000개의 학습 데이터가 존재하고, 매번 50개의 데이터(i.e. 50개의 mini-batch)를 이용하여 모수를 학습할 때, 이 알고리즘을 200번을 반복하면 1 epoch, 400번을 반복하면 2 epochs라고 함.)

케라스 R패키지 설치

케라스(Keras)의 특징

- 모듈화 (modularity)
- 최소주의 (minimalism)
- 쉬운 확장성
- 파이썬 기반

```
> install.packages("keras") # 케라스 R패키지 설치
> library(keras)
> install_keras()
```

텐서(tensor)

스칼라(scalar)	scalar tensor	zero-dimensional tensor(0D tensor)
벡터(vector)	1D tensor	p-dimensional vector
행렬(matrix)	2D tensor	raw, column
3D tensor	고차원 tensor	

```
> x <- c(12, 3, 6, 14, 10)
> str(x)
 num [1:5] 12 3 6 14 10
> dim(as.array(x))
[1] 5

> x <- array(rep(0, 2*3*2), dim=c(2,3,2))
> str(x)
 num [1:2, 1:3, 1:2] 0 0 0 0 0 0 0 0 0 0 ...
> dim(x)
[1] 2 3 2
```

텐서 함수

1. 축의 수(#. of ranks)
2. 모양(shape)
3. 데이터 유형(data type)

```
> library(keras)
> mnist <- dataset_mnist()
> train_images <- mnist$train$x
> train_labels <- mnist$train$y
> test_images <- mnist$test$x
> test_labels <- mnist$test$y
> dim(train_images) #[1] 60000    28    28

#데이터 유형
> typeof(train_images)

#5번째 저장된 숫자 이미지
> digit <- train_images[5, , ]
> plot(as.raster(digit, max = 255))
```

텐서 슬라이싱(tensor slicing)

```
> slice <- train_images[10:99, , ]
> dim(slice) #[1] 90 28 28
> plot(as.raster(slice[1,,], max = 255))

# 오른쪽 하단 이미지 추출
> slice <- train_images[10:99, 15:28, 15:28]
> dim(slice) #[1] 90 28 28
> plot(as.raster(slice[1,,], max = 255)) # range(digit)
```

텐서 모양 변경(tensor reshaping)

```
> train_images <- array_reshape(train_images, c(60000, 28*28))
> x <- matrix(c(0,1,
                2,3,
                4,5), nrow=3, ncol=2, byrow=TRUE)
> x <- array_reshape(x, dim=c(6,1))
> x <- array_reshape(x, dim=c(2,3))
> dim(t(x)) # 전치(transpose)
```

Keras에서 tensor의 활용

- 2D텐서(표본, 특징): 완전연결계층(fully connected layer) or 조밀계층(dense layer), layer_dense
- 3D텐서(표본, 시간대, 특징): layer_lstm
- 4D텐서로 저장된 이미지: 2D 합성곱계층: layer_conv_2d

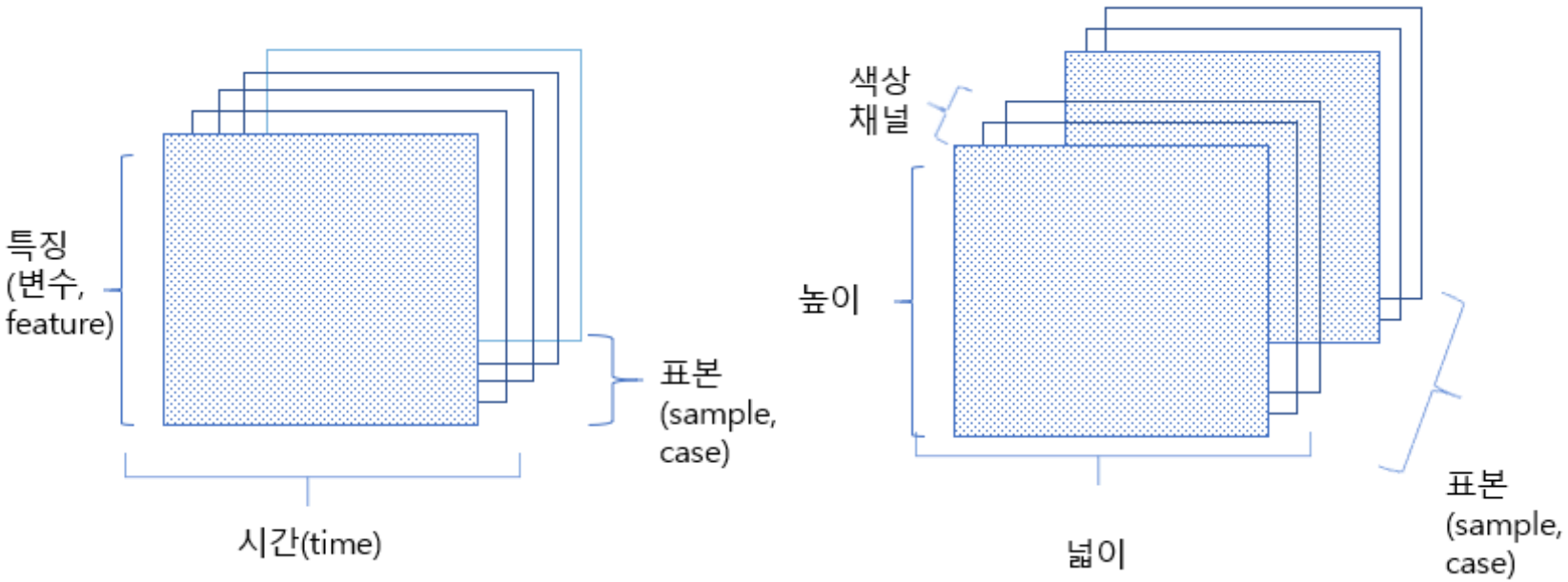


Figure: 3D vs. 4D tensor

계층 호환성(layer compatibility)

특정 모양의 입력텐서, 지정된 출력 텐서

```
# 입력: 2D 텐서, 첫번째 차원이 784
# 출력: 차원이 32인 텐서를 반환
layer <- layer_dense(units=32, input_shape=c(784))
```

연습: 우편번호 다중클래스 분류

- 0-9까지 총 10가지의 수 중 하나를 손으로 직접 쓴 이미지 데이터
- 60000개의 훈련자료와 10000개의 시험자료

훈련자료/평가자료 준비

```
> train_images <- array_reshape(train_images, c(60000, 28*28))
> train_images <- train_images / 255
test_images <- array_reshape(test_images, c(10000, 28*28))
> test_images <- test_images / 255

> train_labels <- to_categorical(train_labels)
> test_labels <- to_categorical(test_labels)
```

망 구성

```
> network <- keras_model_sequential() %>%
    layer_dense(units=512, activation='relu', input_shape = c(28*28)) %>%
    layer_dense(units=10, activation='softmax')

# 망 컴파일 1
> network %>% compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics = c('accuracy')
  )

# 망 구성 및 컴파일 2
#compile(
#  network,
#  optimizer = 'rmsprop',
#  loss = 'categorical_crossentropy',
#  metrics = c('accuracy')
#)
```

모형 적합과 평가

```
# 모형 적합
> network %>% fit(train_images, train_labels, epochs=5, batch_size=128)

# 모형 평가
> metrics <- network %>% evaluate(test_images, test_labels, verbose = 0)
metrics
```

- 참고. % % 는 magrittr 패키지에서 제공되는 연쇄 오른쪽에 있는 함수의 첫번째 인수로 왼쪽에 값을 전달함.

실습1: 이진분류 I

- 리뷰의 텍스트 내용을 기준으로 영화 리뷰를 "긍정적 인" 리뷰와 "부정적인" 리뷰로 분류

```
> library(keras)
# 영화감상평: 긍정 1, 부정 0
> imdb <- dataset_imdb(num_words = 10000) # 출현빈도 기준 상위 1만개

#reuters <- dataset_reuters(num_words=10000)
> c(c(tr_data, tr_labels), c(te_data, te_labels)) %<-% imdb
> str(tr_data[[1]])
> tr_labels[[1]]
> max(sapply(tr_data, max))
```

```
> word_idx <- dataset_imdb_word_index()
> reverse_word_idx <- names(word_idx)
> names(reverse_word_idx) <- word_idx
> decoded_review <- sapply(tr_data[[1]],
    function(idx){
      word <- if(idx >= 3)
                reverse_word_idx[[as.character(idx - 3)]]
      if(!is.null(word))
        word
      else "?"
    }
> )
```

실행에 error가 날 경우

```
#Error in py_call_impl(callable, dots$args, dots$keywords) :
#  ValueError: Object arrays cannot be loaded when #allow_pickle=False
#https://github.com/rstudio/keras/issues/765
library(tensorflow)
install_tensorflow(version="nightly")
```

Table: one hot coding

단어(text)	단어 index	원핫표현
you	1	[1,0,0,0,0,0]
goodbye	3	[0,0,1,0,0,0]

```
> vectorize_sequence <- function(seqs, dims=1e4)
  {
   results <- matrix(0, nrow=length(seqs), ncol=dims)
    for(i in 1:length(seqs)){
      # 특정 index들을 1로 설정
      results[i, seqs[[i]]] <- 1
    }
    return(results)
 }
> x_tr <- vectorize_sequence(tr_data)
> x_te <- vectorize_sequence(te_data)
> str(x_tr[1,])
> y_tr <- as.numeric(tr_labels)
> y_te <- as.numeric(te_labels)
```

망 구성

2개의 은닉 계층

```
> library(keras)
> model <- keras_model_sequential() %>%
    layer_dense(units=16, activation="relu",
      input_shape=10000) %>%
    layer_dense(units=16, activation="relu") %>%
    layer_dense(units=1, activation="sigmoid")
```

망 컴파일1

```
> model %>% compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=c("accuracy"))
```

또는
망 컴파일2

```
> model %>% compile(
    optimizer=optimizer_rmsprop(lr=0.001),
    loss="binary_crossentropy",
    metrics=c("accuracy"))
```

망 컴파일3(사용자 정의함수 활용)
-toyfunc1, toyfunc2는 사용자정의함수

```
> model %>% compile(
    optimizer=optimizer_rmsprop(lr=0.001),
    loss=toyfunc1,
    metrics=toyfunc2
  )
```

검증집합 설정

```
> val_idx <- 1:1e4
> x_val <- x_tr[val_idx,]
> partial_x_tr <- x_tr[-val_idx,]
> y_val <- y_tr[val_idx]
> partial_y_tr <- y_tr[-val_idx]
```

모델 훈련

- 20 epoch $\longrightarrow$ 미니배치(mini-batch, 512개의 표본)을 20번 반복

```
> model %>% compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=c("accuracy"))

> history <- model %>% fit(
    partial_x_tr, partial_y_tr,
    epochs=20,
    batch_size=512,
    validation_data = list(x_val, y_val))
```

```
> str(history)

> plot(history)

> history_df <- as.data.frame(history)

> str(history_df)
```

훈련결과 정리

- 훈련손실은 에포크마다 줄어들며, 훈련의 정확도는 증가
- 과적합 방지하기 위해 epoch=4에서 훈련종료

재훈련

```
> model %>% fit(x_tr, y_tr, epochs=4, batch_size=512)
> results <- model %>% evaluate(x_te, y_te)
> results
```

평가성능 측정

```
> model %>% predict(x_te[1:10,])
```

자율학습

1. 은닉 계층을 3개로 늘려보자.
2. 32 units, 64 units 등 units의 개수를 늘리거나 줄여보자.
3. 손실함수 binary_crossentropy 대신 mse를 사용해보자.
4. 활성함수로 relu 대신에 tanh를 활용해보자.

실습2: 회귀분석 I

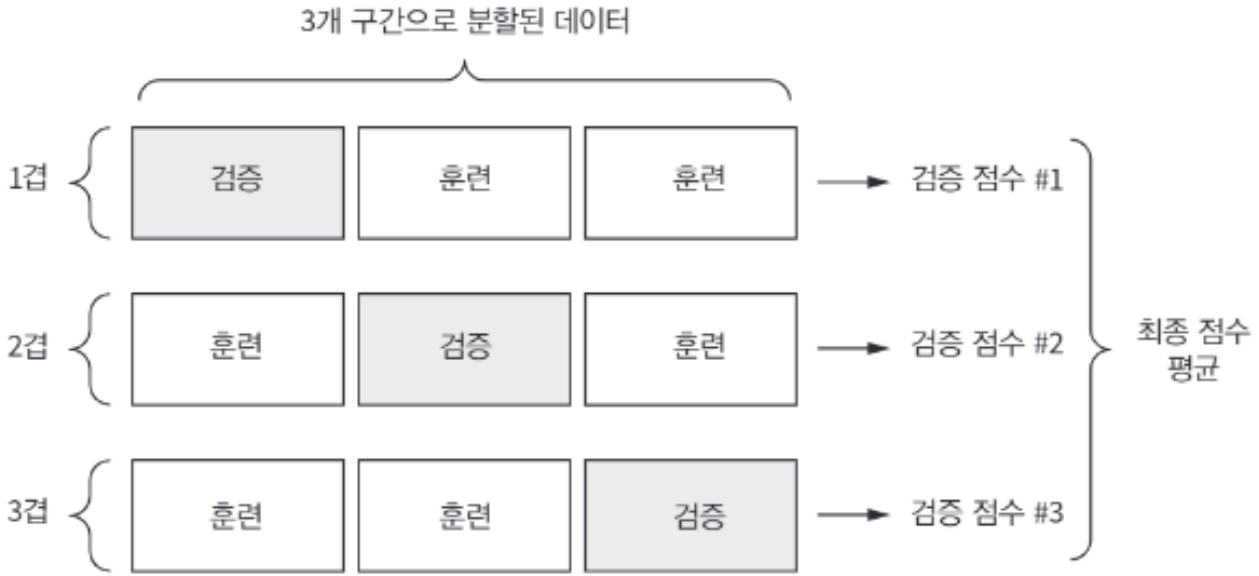
- 1970년대 보스턴 교외 지역의 주택 평균 가격 예측
- 범죄율, 지방 재산세율 등 13개의 변수

```
> library(keras)
> d <- dataset_boston_housing() # class(d); names(d)
> c(c(tr_data, tr_targets), c(te_data, te_targets)) %<-% d
> str(tr_data)
> str(te_data)
> str(tr_targets)
# 데이터 표준화(정규화)
> m <- apply(tr_data, 2, mean)
> s <- apply(tr_data, 2, sd)
> tr_data <- scale(tr_data, m, s)
> te_data <- scale(te_data, m, s)
```

망 구성

- 모델생성 함수 구성
- 마지막 계층은 선형계층으로 구성
- 손실함수로 제곱손실함수 활용(mse, mean squared error)
- 평가측도로 절대오차(mae, mean absolute error) 활용

```
> build_model <- function(){
  model <- keras_model_sequential() %>%
    layer_dense(units=64, activation="relu",
                input_shape=ncol(tr_data)) %>%
    layer_dense(units=64, activation="relu") %>%
    layer_dense(units=1)
  model %>% compile(
    optimizer="rmsprop",
    loss="mse",
    metrics=c("mae")
  )
}
```



k-fold cross-validation

```
> k <- 4 # 4-fold
> indices <- sample(1:nrow(tr_data))
> folds <- cut(1:length(indices), breaks = k, labels = F)
> num_epochs <- 100
> for (i in 1:k) {
  cat("processing fold #", i, "\n")

  # 검증자료
  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- tr_data[val_indices,]
```

```
  val_targets <- tr_targets[val_indices]

  # 훈련자료
  partial_tr_data <- tr_data[-val_indices,]
  partial_tr_targets <- tr_targets[-val_indices]

  # 모형 생성
  model <- build_model()

  # 모형 적합
  history <- model %>% fit(
    partial_tr_data, partial_tr_targets,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 1, verbose = 0
  )
  mae_history <- history$metrics$val_mean_absolute_error
  all_mae_histories <- rbind(all_mae_histories, mae_history)
}
```

```
> average_mae_history <- data.frame(
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories, 2, mean)
)
> library(ggplot2)
> ggplot(average_mae_history, aes(x = epoch, y = validation_mae)) +
  geom_line()
> ggplot(average_mae_history, aes(x = epoch, y = validation_mae)) +
  geom_smooth()
```

전체자료를 모두 활용한 최종망 구축

```
> model <- build_model()
> model %>% fit(train_data, train_targets,
          epochs = 80, batch_size = 16, verbose = 0)
> result <- model %>% evaluate(test_data, test_targets)
> result
```

Reference

- 프랑수와 숄레(2019). Deep learning with R, 프랑수와 숄레, 박진수 옮김, Jpub.
- 이미지 파일: `https://github.com/Jpub/Deep_Learning_R/`
- 예제코드: `https://github.com/jjallaire/deep-learning-with-r-notebooks`