Introduction to Artificial Neural Networks [2] – MLP with Keras

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교 HANKYONG NATIONAL UNIV.

Implementing MLPs with Keras

Keras: high-level Deep Learning API(March 2015)

- 모든 종류의 신경망을 build, train, evaluate, execute를 쉽게 할 수 있도록
- Keras 레퍼런스 구현체: 신경망이 요구하는 과대한 계산을 수행하기 위해 computation backend에 의존
- 현재 3개의 오픈 소스 Deep Learning 라이브러리 제공됨(multibackend Keras라고 함)

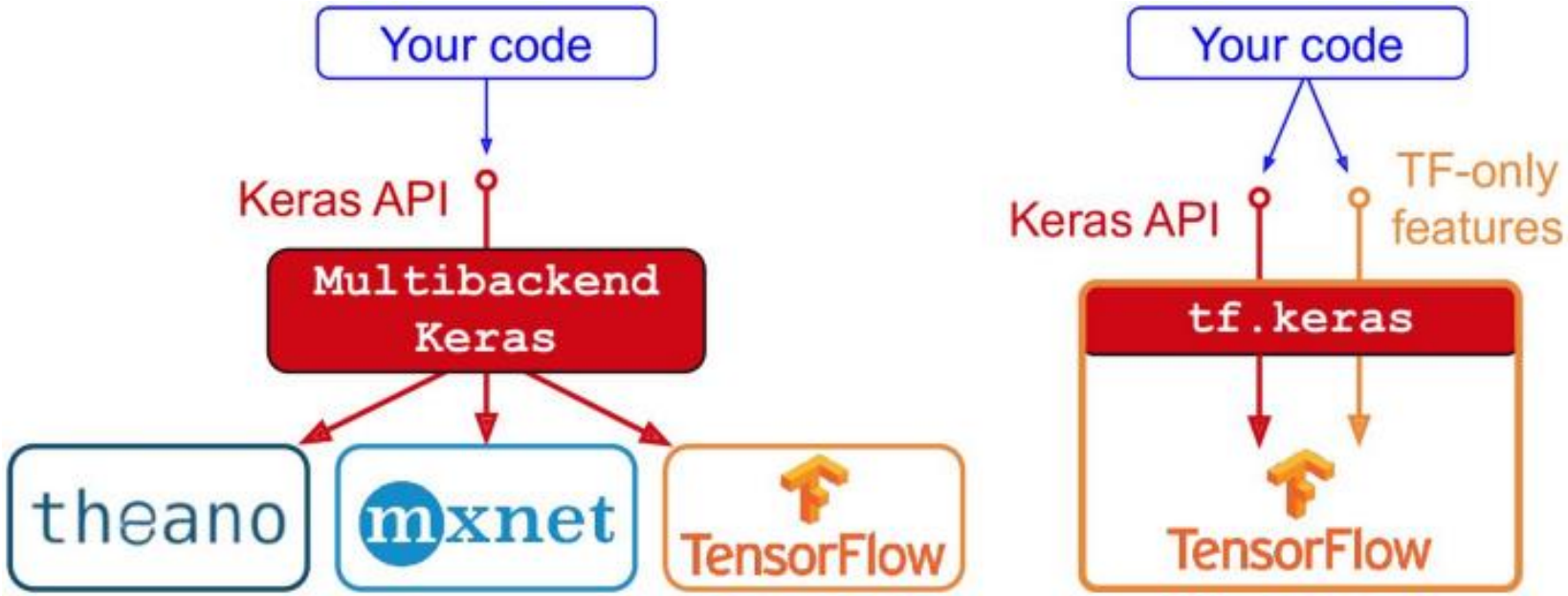
  TensorFlow, Microsoft Cognitive Toolkit(CNTK), Theano.

2016년 후반, 다른 구현체 배포

- Apache MXNet, Apple's Core ML, JavaScript or TypeScript(웹브라우저에서 Keras 실행 가능), PlaidML
- TensorFlow 2 => 자체 구현체에 Keras를 번들로 제공: **tf.keras**

오로지 TensorFlow만 백엔드로 지원 + TF-only 엑스트라 기능 제공 (예: Data API)

- 여기서는 `tf.keras` 사용



Installing TensorFlow 2

1. AVX(CPU) 기능이 없는 경우

$ activate aisam

$ conda install tensorflow==**1.5.0**

2. AVX 기능이 있는 경우

$ activate aisam

$ conda install tensorflow

$ python -c 'import tensorflow; print(tensorflow.__version__)'

2.1.0

⇒ TensorFlow 2.1.0 버전부터는 CPU 패키지와 GPU 패키지가 통합되었다.

Keras 설치:

$ conda install keras

$ python

>>>import keras

Using TensorFlow backend.

인텔 프로세서 AVX 지원 여부 판단 => **지원하지 않는 경우 TensorFlow 2를 설치할 수 없음!!**

https://downloadcenter.intel.com/ko/download/28539/Intel-Processor-Identification-Utility-Windows-Version

Installing TensorFlow 2

AVX 기능이 지원 안될 때:

$ activate **aisam**

$ conda install tensorflow==1.5.0

AVX 기능이 지원될 때:

$ activate **aisam**

$ conda install tensorflow

=> TensorFlow 2.1.0 버전부터는 CPU 패키지와 GPU 패키지가 통합되었다.

GPU가 없을 때:

$ conda install tensorflow-cpu

$ python

>>> import tensorflow as tf

>>> print(tf.__version__)

$ conda install keras

$ python

>>> import keras

Using TensorFlow backend.

Colab

클라우드 기반의 무료 Jupyter 노트북 개발 환경

- 'Colab + Google Drive + Docker + Linux + Google Cloud'로 구성
- GPU를 공짜로 이용할 수 있음
- 최대 세션 유지 시간: 12시간
- Google Colaboratory 공식페이지

Google Colaboratory 환경설정 및 간단한 실습

test.ipynb

Building an Image Classifier Using the Sequential API

사용할 데이터셋: Fashion MNIST

- MNIST와 동일 포맷: 70,000 grayscale images of 28 X 28 pixels each, with 10 classes.
- MNIST 보다 challenging! (Simple linear model: 92% accuracy on MNIST, 83% on Fashion MNIST)

Using Keras to load the dataset

Keras => 데이터셋을 fetch, load 하기 위한 유틸리티 함수 제공

```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

Scikit-Learn이 아니라 Keras를 사용하여 MNIST 또는 Fashion MNIST를 로딩할 때 차이점

- 모든 이미지 => 784 사이즈의 1D 배열이 아니라 28X28 2D 배열로 표현
- 픽셀 강도 => 0.0~255.0 실수형 값이 아니라 0~255 정수형 값으로 표현

```
>>> X_train_full.shape
(60000, 28, 28)
>>> X_train_full.dtype
dtype('uint8')
```

Using Keras to load the dataset

- 데이터셋 => 이미 training set과 test set으로 나뉘어져 있음
- Validation set은 없음 => 생성!
- 신경망은 Gradient Descent로 학습시키므로 입력 특성을 scaling할 필요 있음

```
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
```

- MNIST의 경우: label=5이면 필기 숫자 5임을 의미,

  반면 Fashion MNIST의 경우 현재 다루고 있는 클래스를 알기 위해 클래스명 리스트 필요

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

- 예: training set의 첫 번째 이미지를 알기 위해서는

```
>>> class_names[y_train[0]]
'Coat'
```

Using Keras to load the dataset

Fashion MNIST 데이터셋의 샘플 예:

Creating the model using the Sequential API

Sequential 모델 생성 - 순차적으로 연결된 레이어 스택으로 구성된 신경망의 가장 단순한 형태의 Keras 모델 (Sequential API)

Flatten 레이어: 입력 이미지를 1D 배열로 변환:
입력 데이터 X를 받으면 X.reshape(-1, 1) 계산
입력 shape를 지정해야 함

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

300개의 뉴런을 갖는 Dense 히든 레이어
=> 자신의 가중치 행렬 관리
=> Bias 항목 벡터도 관리

100개의 뉴런을 갖는 Dense 히든 레이어

10개의 뉴런(클래스당 1개)을 갖는 Dense 출력 레이어
Softmax 활성 함수 사용

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

Creating the model using the Sequential API

`summary()` 메소드 => 모델 레이어의 모든 것을 표시

- 레이어 이름, 출력 모양(shape), 파라메터 개수 등

```
>>> model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 300)               235500
_________________________________________________________________
dense_1 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

784 inputs X 300 weights +
300 bias terms
= 235,500 parameters

Creating the model using the Sequential API

모델의 레이어 리스트 얻기:

```
>>> model.layers
[<tensorflow.python.keras.layers.core.Flatten at 0x132414e48>,
 <tensorflow.python.keras.layers.core.Dense at 0x1324149b0>,
 <tensorflow.python.keras.layers.core.Dense at 0x1356ba8d0>,
 <tensorflow.python.keras.layers.core.Dense at 0x13240d240>]
>>> hidden1 = model.layers[1]
>>> hidden1.name
'dense'
>>> model.get_layer('dense') is hidden1
True
```

get_weights()/set_weights() 메소드를 이용하여 레이어의 모든 파라메터에 접근:

```
>>> weights, biases = hidden1.get_weights()
>>> weights
array([[ 0.02448617, -0.00877795, -0.02189048, ..., -0.02766046,
         0.03859074, -0.06889391],
       ...,
       [-0.06022581,  0.01577859, -0.02585464, ..., -0.00527829,
         0.00272203, -0.06793761]], dtype=float32)
>>> weights.shape
(784, 300)
>>> biases
array([0., 0., 0., 0., 0., 0., 0., 0., 0., ...,  0., 0., 0.], dtype=float32)
>>> biases.shape
(300,)
```

Compiling the model

compile() 메소드 호출: 사용할 loss 함수와 optimizer 지정

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

- "sparse_categorical_crossentropy" loss: Sparse labels & 클래스 : exclusive일 때
  - ✓ for each instance, there is just a target class index, from 0 to 9 in this case
- "categorical_crossentropy" loss: 각 인스턴스가 클래스당 1개의 타깃 확률을 가질 때
  - ✓ such as one-hot vectors, e.g. [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.] to represent class 3
- "binary_crossentropy" loss: 1개 이상의 binary 레이블을 가지는 binary 분류 수행 시
  - ✓ 출력 레이어에 "softmax" 대신 "sigmoid" 활성 함수 사용

"sgd" : Stochastic Gradient Descent를 사용하여 모델 학습

Training and evaluating the model

이제 모델이 학습될 준비가 되었다 => `fit()` 메소드 호출

```
>>> history = model.fit(X_train, y_train, epochs=30,
...                     validation_data=(X_valid, y_valid))
...
Train on 55000 samples, validate on 5000 samples
Epoch 1/30
55000/55000 [======] - 3s 49us/sample - loss: 0.7218     - accuracy: 0.7660
                                      - val_loss: 0.4973 - val_accuracy: 0.8366
Epoch 2/30
55000/55000 [======] - 2s 45us/sample - loss: 0.4840     - accuracy: 0.8327
                                      - val_loss: 0.4456 - val_accuracy: 0.8480
[...]
Epoch 30/30
55000/55000 [======] - 3s 53us/sample - loss: 0.2252     - accuracy: 0.9192
                                      - val_loss: 0.2999 - val_accuracy: 0.8926
```

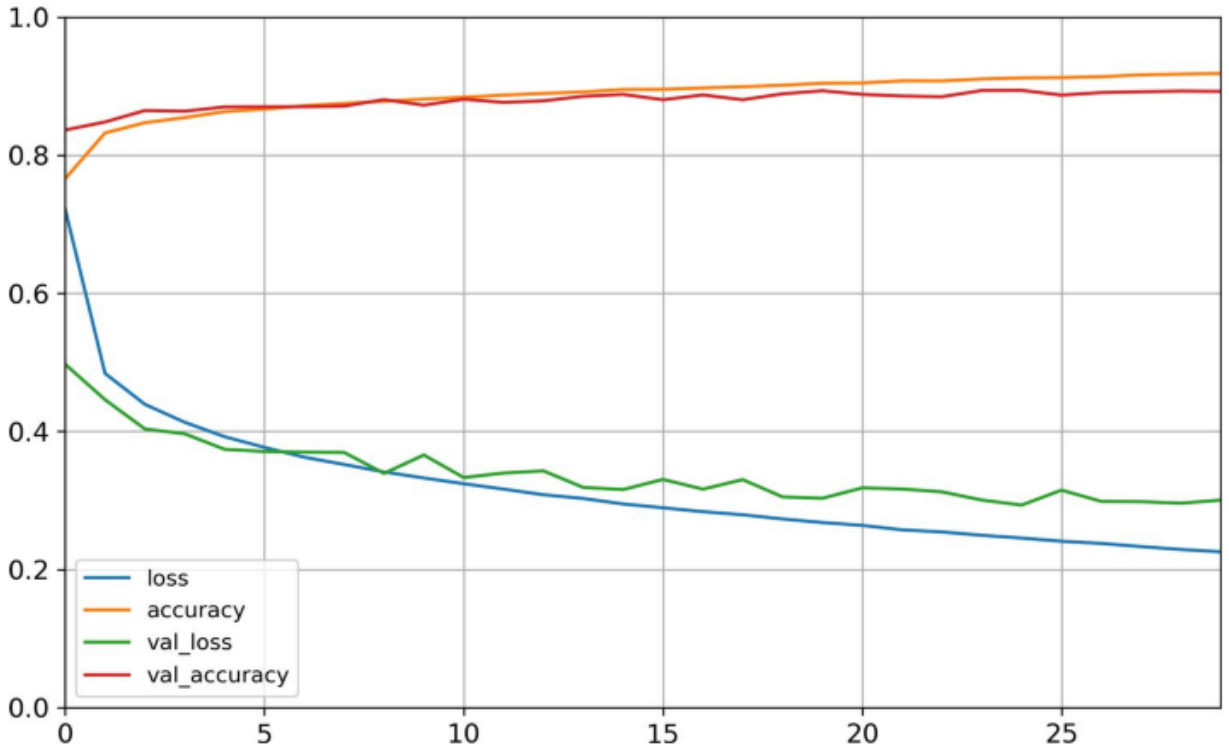
Training and evaluating the model

`fit()` 메소드가 반환하는 내용:

- (`history.params`) 학습 파라메터를 포함하는 History 객체
- (`history.epoch`) 수행한 epoch 리스트
- (`history.history`) 각 epoch 끝에 측정한 loss 및 엑스트라 메트릭을 포함하는 dictionary
- Dictionary를 이용하여 pandas DataFrame을 생성하고 `plot()` 메소드 호출 => 학습 곡선

```
import pandas as pd
import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
plt.show()
```



Training and evaluating the model

모델 성능이 만족스럽지 못하다면 여러 가지 hyperparameter들을 조정해 볼 수 있다.

- 학습률 조정
- Optimizer 변경해 보기 => 학습률 재조정
- 모델 hyperparameter 조정: 레이어 개수, 레이어당 뉴런 개수, 활성 함수 종류 변경

모델의 validation 정확도가 만족스럽다면 => test set에 대해 generalization 에러 추정

```
>>> model.evaluate(X_test, y_test)
10000/10000 [==========] - 0s 29us/sample - loss: 0.3340 - accuracy: 0.8851
[0.3339798209667206, 0.8851]
```

Using the model to make predictions

`predict()` 메소드를 이용하여 새 인스턴스에 대해 예측 수행

- 각 인스턴스에 대해 클래스당 1개의 확률 추정 (클래스 0 ~9까지)

```
>>> X_new = X_test[:3]
>>> y_proba = model.predict(X_new)
>>> y_proba.round(2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.03, 0.  , 0.01, 0.  , 0.96],
       [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

- 가장 높은 추정 확률을 갖는 클래스를 알고 싶다면 => `predict_classes()` 메소드 출력

```
>>> y_pred = model.predict_classes(X_new)
>>> y_pred
array([9, 2, 1])
>>> np.array(class_names)[y_pred]
array(['Ankle boot', 'Pullover', 'Trouser'], dtype='<U11')
```

Using the model to make predictions

Test set에 적용 => 3개의 이미지에 대해 정확하게 분류!

```
>>> y_new = y_test[:3]
>>> y_new
array([9, 2, 1])
```

Ankle boot

Pullover

Trouser

Building a Regression MLP Using the Sequential API

California housing problem을 Regression 신경망으로!

- `fetch_california_housing()` 함수 이용 => 데이터 로딩
- 2장에서 사용했던 것보다 더 단순 => ocean_proximity 특성을 제외한 숫자 특성만으로 구성, no missing value

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()

X_train_full, X_test, y_train_full, y_test = train_test_split(
    housing.data, housing.target)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

Building a Regression MLP Using the Sequential API

Regression MLP 이용 => 예측

- Classification MLP와 흡사
- 차이점: 출력 레이어 => 단 1개의 뉴런, 활성 함수 사용 안함, Loss 함수 => MSE

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer="sgd")
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3] # pretend these are new instances
y_pred = model.predict(X_new)
```

- Sequential API => Extremely COMMON

그러나 복잡한 문제에 대해서는 미흡 => Functional API 이용 !

실습과제 12-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

또는 (실습환경이 AVX 등이 지원되지 않을 경우) Google Colab 이용 가능

참고: Google Colaboratory 환경설정 및 간단한 실습

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고: 제 12강 실습과제 #12 Introduction to Artificial Neural Networks [2] - MLP with Keras.pdf