Introduction to Artificial Neural Networks [3] - Functional API, Callbacks, TensorBoard

Samkeun Kim <skim@hknu.ac.kr>

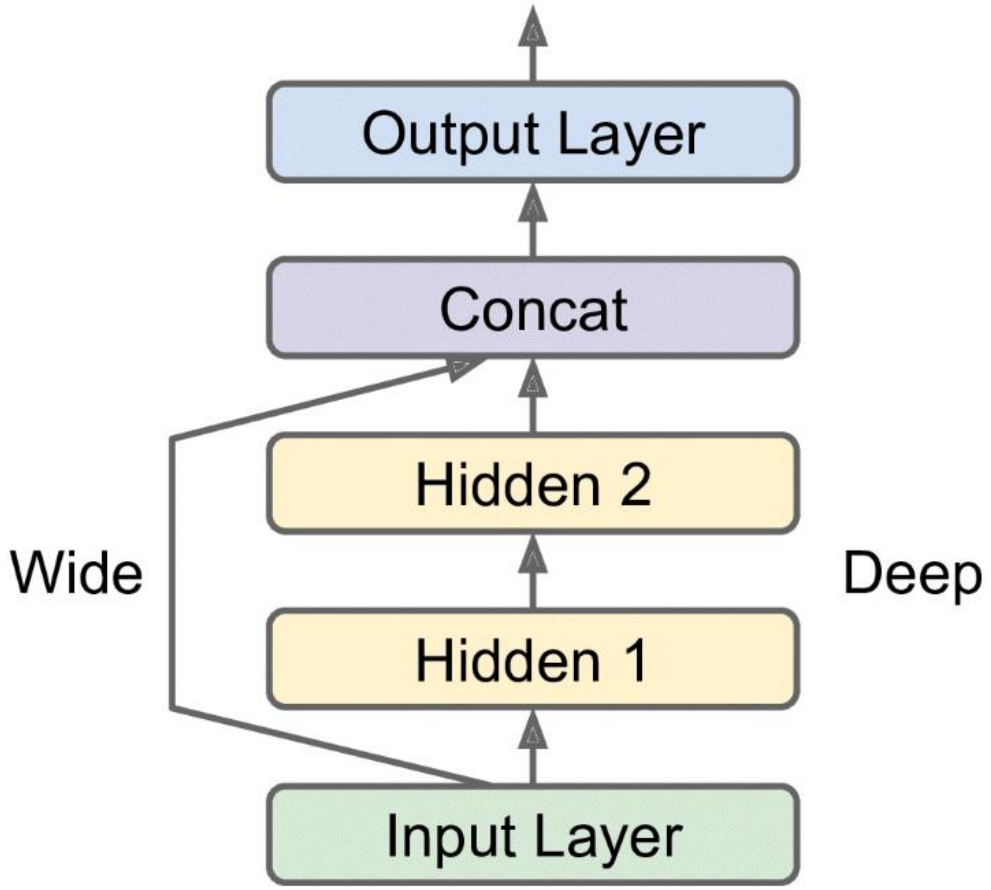
http://cyber.hankyong.ac.kr

국립한경대학교 HANKYONG NATIONAL UNIV.

Building Complex Models Using the Functional API

Nonsequential 신경망의 한 가지 예 => Wide & Deep 신경망

- Heng-Tze Cheng et al.이 제안(2016)
- 입력의 전체 또는 일부분을 출력 레이어에 직접 연결
- Deep 패턴 학습 => deep path를 통해
- 단순 룰 학습 => wide path를 통해



California housing problem에 적용:

Input 객체 생성:
shape/dtype 지정

30개의 뉴런을 갖는
Dense 레이어 생성
⇒ 함수처럼 호출!

두 번째 히든 레이어 생성

Concatenate 레이어 생성
⇒ 입력과 두 번째 히든 레이어의 출력을 병합!

```
input_ = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.Concatenate()([input_, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.Model(inputs=[input_], outputs=[output])
```

Keras 모델 생성:
사용할 입력과 출력 지정

출력 레이어 생성:
1개의 뉴런, 활성 함수 없음
함수처럼 호출

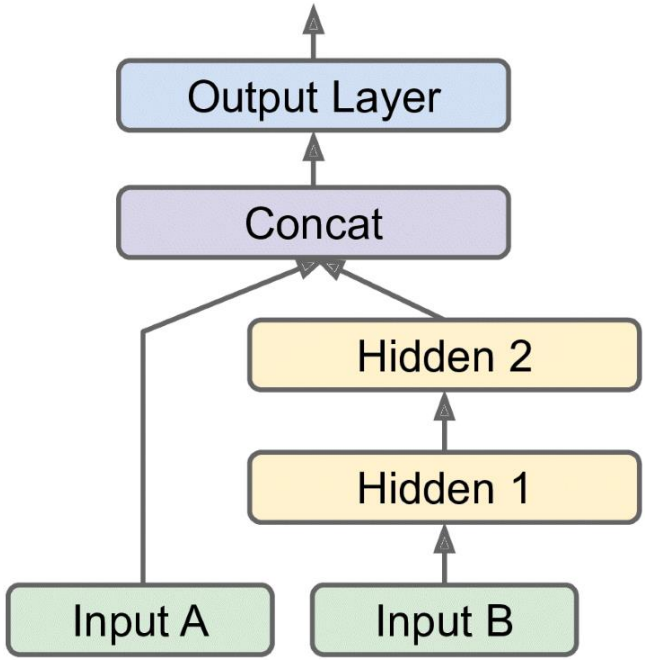
Building Complex Models Using the Functional API

입력 특성의 일부 세트는 wide path를 통과시키고

또 다른 일부 세트는 deep path를 통과시키려면?

- 다중 입력 사용
- 모델 생성할 때

  ⇒ `inputs=[input_A, input_B]` 지정




```
input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="output")(concat)
model = keras.Model(inputs=[input_A, input_B], outputs=[output])
```

fit() 메소드 호출할 때 => 1개의 X_train 행렬이 아니라 (X_train_A, X_train_B)를 전달해야

```
model.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1e-3))

X_train_A, X_train_B = X_train[:, :5], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :5], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :5], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]

history = model.fit((X_train_A, X_train_B), y_train, epochs=20,
                    validation_data=((X_valid_A, X_valid_B), y_valid))
mse_test = model.evaluate((X_test_A, X_test_B), y_test)
y_pred = model.predict((X_new_A, X_new_B))
```

Saving and Restoring a Model

Sequential API 또는 Functional API를 사용할 때 학습된 Keras 모델을 저장하는 방법:

```
model = keras.models.Sequential([...]) # or keras.Model([...])
model.compile([...])
model.fit([...])
model.save("my_keras_model.h5")
```

- Keras => 모델 아키텍처와 모든 레이어의 모든 모델 파라메터의 값을 HDF5포맷으로 저장!
- Optimizer도 저장
- TensorFlow의 SavedModel 포맷을 사용하여 tf.keras 모델을 저장하는 방법 =>Chapter 19.

Saving and Restoring a Model

저장된 모델을 로딩하는 방법:

```
model = keras.models.load_model("my_keras_model.h5")
```

그런데 학습이 몇 시간 동안 걸린다면? 사실 이런 경우는 매우 흔하다. 특히, 대규모 데이터셋에 대해 학습시킬 때

⇒ 보통 학습 종료 후에 모델을 저장하고, 또 학습 중에 정규적인 간격으로 체크포인트를 저장

⇒ 학습 중에 컴퓨터가 다운되었을 때 모든 내용을 잃지 않기 위해

⇒ 그러나, `fit()` 메소드에게 어떻게 체크포인트를 저장하라고 말할 수 있을까?

Using Callbacks

fit() 메소드 => Keras가 호출하게 될 객체 리스트를 지정하게 해주는 `callbacks` 인자를 받아들인다.

- 학습의 시작과 끝에
- 각 epoch의 시작과 끝에
- 각 batch 처리 전후에

예) `ModelCheckpoint` 콜백 => 학습 중 정규적인 간격(디폴트로 각 epoch의 끝에)으로 모델의 체크포인트를 저장

```
[...] # build and compile the model
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5")
history = model.fit(X_train, y_train, epochs=10, callbacks=[checkpoint_cb])
```

Using Callbacks

학습 중 validation set을 사용한다면, `ModelCheckpoint` 생성할 때 save_best_only=True 설정 가능

- Validation set에 대해 성능이 지금까지 최상인 경우만 모델을 저장

  ⇒ 추후 학습 후 저장된 마지막 모델만 restore하면 됨 (즉 validation set에 대한 최상의 모델임)
- Early stopping을 구현하기 위한 단순한 방법:

```
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5",
                                                save_best_only=True)
history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])
model = keras.models.load_model("my_keras_model.h5") # roll back to best model
```

Early stopping을 구현하기 위한 또 다른 방법 => `EarlyStopping` 콜백 사용

- 많은 epoch 동안 validation set에 대해 진전이 없을 때 학습을 인터럽트 시킴 (`patience` 인자로 정의)

```
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,
                                                  restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=100,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

- 자동으로 멈출 수 있으므로 epoch의 개수 지정을 충분히 큰 값으로 설정해도 된다.

Using TensorBoard for Visualization

TensorBoard => Great interactive visualization tool

- 학습 중에 학습 곡선을 보여줄 수 있고, 여러 개의 학습 곡선들을 비교할 수 있고, Computation 그래프를 시각화 등의 복잡한 다차원 데이터를 시각화 하는데 사용할 수 있는 훌륭한 대화형 시각화 도구
- 이를 사용하려면 시각화 하려는 데이터를 이벤트 파일이라는 특수 바이너리 로그 파일로 출력하도록 프로그램을 수정해야 함
- 각 바이너리 데이터 레코드를 'summary'라고 함
- TensorBoard 서버는
  - ✓ Log 디렉토리를 모니터링하고, 변경 사항을 자동으로 선택하여 시각화를 업데이트 해 줌

    ⇒ 이 기능은 학습 중에 생성되는 학습 곡선과 같은 live 데이터를 시각화 가능하게 해 줌

TensorBoard 로그를 위해 사용할 루트 log 디렉토리 + 현재의 date & time에 기반한 서브 디렉토리 생성 함수 정의:

```
import os
root_logdir = os.path.join(os.curdir, "my_logs")

def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

run_logdir = get_run_logdir() # e.g., './my_logs/run_2019_06_07-15_15_22'
```

Using TensorBoard for Visualization

Keras => TensorBoard() 콜백 함수 제공:

```
[...] # Build and compile your model
tensorboard_cb = keras.callbacks.TensorBoard(run_logdir)
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid),
                    callbacks=[tensorboard_cb])
```

몇 초간 수행 후의 디렉토리 구조:

```
my_logs/
├── run_2019_06_07-15_15_22
│   ├── train
│   │   ├── events.out.tfevents.1559891732.mycomputer.local.38511.694049.v2
│   │   ├── events.out.tfevents.1559891732.mycomputer.local.profile-empty
│   │   └── plugins/profile/2019-06-07_15-15-32
│   │       └── local.trace
│   └── validation
│       └── events.out.tfevents.1559891733.mycomputer.local.38511.696430.v2
└── run_2019_06_07-15_15_49
    └── [...]
```

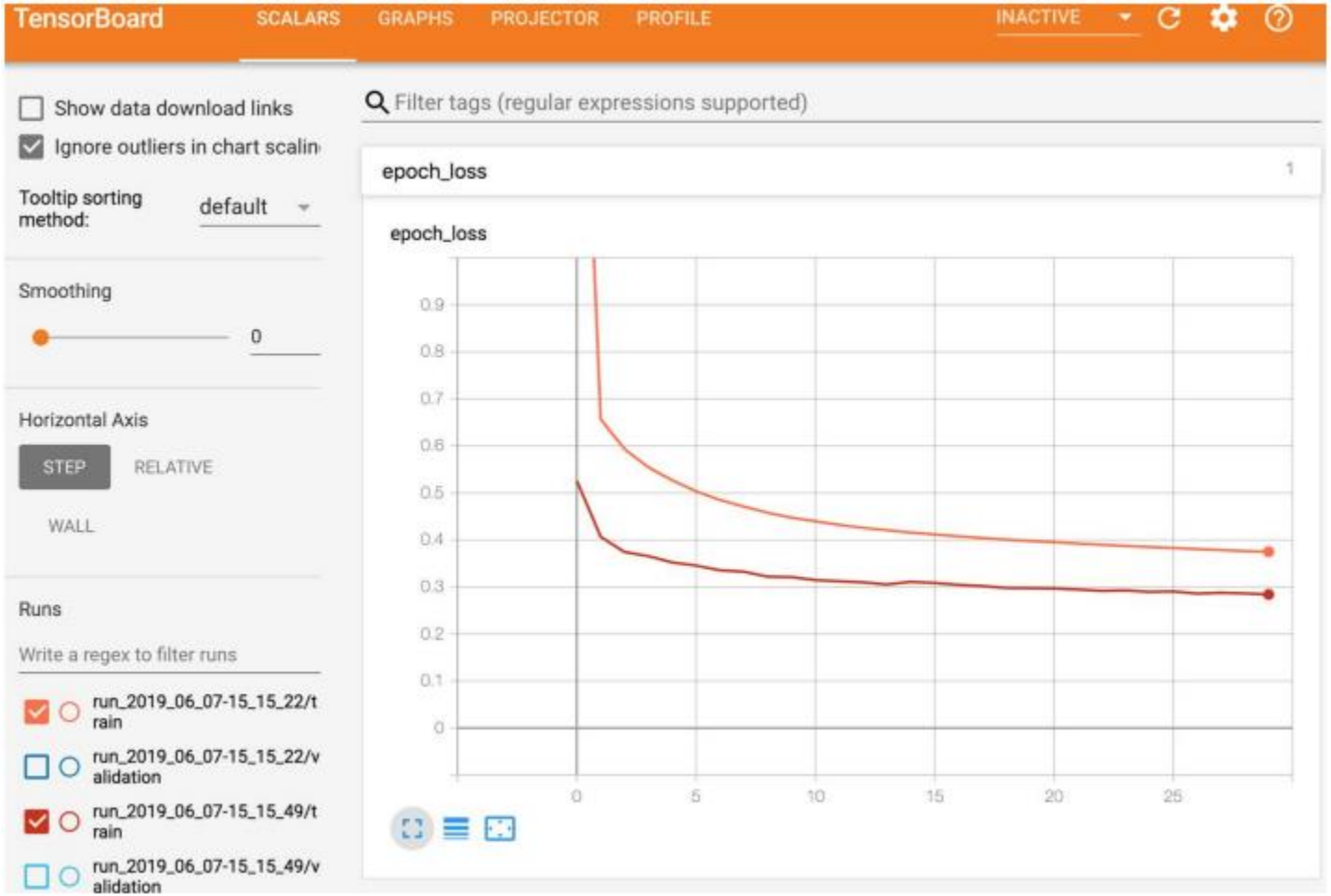
TensorBoard 서버 기동:

```
$ tensorboard --logdir=./my_logs --port=6006
TensorBoard 2.0.0 at http://mycomputer.local:6006/ (Press CTRL+C to quit)
```

웹 브라우저에서 오픈:

- http://localhost:6006

Visualizing learning curves with TensorBoard:



Fine-Tuning Neural Network Hyperparameters

신경망의 유연성 => 주요 단점 중의 하나

- 조정해야 할 하이퍼 파라미터들이 아주 많다.
- 상상할 수 있는 어떤 구조도 가능:

  ✓ 레이어의 개수, 레이어당 뉴런 개수, 각 레이어에서 사용하는 활성 함수 타입, 가중치 초기화 로직 등
- 이런 하이퍼 파라미터들의 어떤 조합이 가장 좋은 성능을 낼까?

옵션 1:

- Validation 세트에 대해 하이퍼 파라미터들을 다양하게 조합시켜 보고 가장 효과적인 조합을 선택하는 방법

예: `GridSearchCV` 또는 `RandomizedSearchCV`를 사용하여 하이퍼 파라미터 공간 탐색 가능(Chapter 2)

Step 1: 하이퍼 파라미터가 주어지면 Keras 모델을 build/compile할 함수 생성:

```
def build_model(n_hidden=1, n_neurons=30, learning_rate=3e-3, input_shape=[8]):
    model = keras.models.Sequential()
    model.add(keras.layers.InputLayer(input_shape=input_shape))
    for layer in range(n_hidden):
        model.add(keras.layers.Dense(n_neurons, activation="relu"))
    model.add(keras.layers.Dense(1))
    optimizer = keras.optimizers.SGD(lr=learning_rate)
    model.compile(loss="mse", optimizer=optimizer)
    return model
```

⇒ Sequential 모델이 생성 됨

Step 2: `build_model()` 함수에 기반한 KerasRegressor 생성:

```
keras_reg = keras.wrappers.scikit_learn.KerasRegressor(build_model)
```

Step 3: Scikit-Learn regressor처럼 사용 가능:

```
keras_reg.fit(X_train, y_train, epochs=100,
              validation_data=(X_valid, y_valid),
              callbacks=[keras.callbacks.EarlyStopping(patience=10)])
mse_test = keras_reg.score(X_test, y_test)
y_pred = keras_reg.predict(X_new)
```

- `fit()` 메소드를 이용하여 학습시킬 수 있고
- `score()` 메소드를 이용하여 평가할 수 있고
- `predict()` 메소드를 이용하여 예측을 수행할 수 있다!

앞의 방법을 다양하게 변형 가능한 모델 중에서 최상의 모델을 찾는 방법에 대해서도 적용해보자:

- 하이퍼 파라미터 조합을 RandomizedSearchCV 사용하여 탐색:

```
from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
    "n_hidden": [0, 1, 2, 3],
    "n_neurons": np.arange(1, 100),
    "learning_rate": reciprocal(3e-4, 3e-2),
}

rnd_search_cv = RandomizedSearchCV(keras_reg, param_distribs, n_iter=10, cv=3)
rnd_search_cv.fit(X_train, y_train, epochs=100,
                  validation_data=(X_valid, y_valid),
                  callbacks=[keras.callbacks.EarlyStopping(patience=10)])
```

학습이 끝난 후 best score 등을 확인해 볼 수 있다:

```
>>> rnd_search_cv.best_params_
{'learning_rate': 0.0033625641252688094, 'n_hidden': 2, 'n_neurons': 42}
>>> rnd_search_cv.best_score_
-0.3189529188278931
>>> model = rnd_search_cv.best_estimator_.model
```

실습과제 13-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

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

참고:

제 13강 실습과제 #13 Introduction to Artificial Neural Networks [3] - Functional API, Callbacks, TensorBoard.pdf