Keras Study Presentation

What is Keras?

Python 기반의 딥러닝 라이브러리

01. Review - Characteristics of Keras

1. 모듈화 (Modularity)

모든 모듈이 독립적이며, 가능한 최소한의 제약 사항으로 서로 연결될 수 있음

신경망 층, 비용함수, 최적화/초기화 기법, 활성화함수, 정규화기법이 모두 독립적인 모듈

새로운 모델을 만들 때 이러한 모듈을 조합할 수 있음

01. Review - Characteristics of Keras

2. 미니멀리즘 (Minimalism)

각 모듈은 짧고 간결함

모든 코드는 한 번 훑어보는 정도로도 이해가 가능한 수준

(반복 속도와 혁신성에서는 부족함이 있을 수 있음)

01. Review - Characteristics of Keras

3. 확장성 (Extensibility)

새로운 클래스나 함수로 모듈을 아주 쉽게 추가할 수 있음

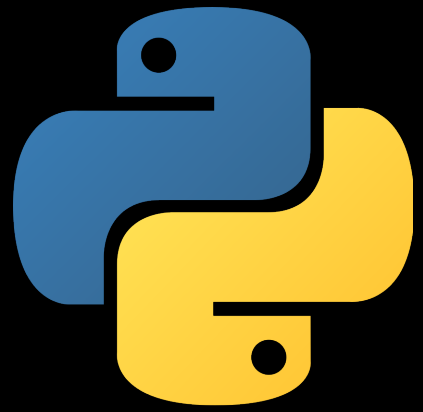
고급 연구에 필요한 다양한 표현을 쉽게 할 수 있음

01. Review - Characteristics of Keras

4. 파이썬 기반 (Based-on Python)

별도의 모델 설정 파일이 필요 없음

많은 사람들이 사용하는 Python 코드로 모델들의 정의되고 구동됨



01. Review - Keras vs. PyTorch

딥러닝 연구에서 새롭게 떠오르는 강자,
PyTorch

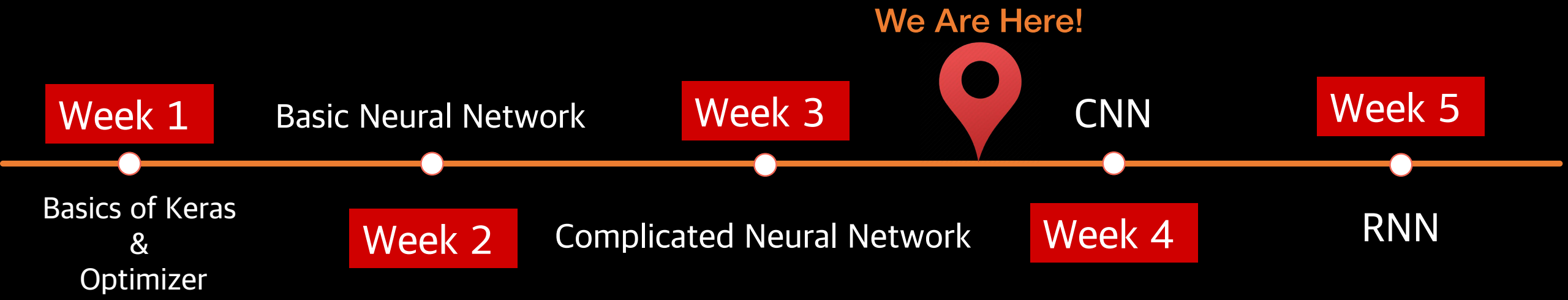
PyTorch

01. Review - Keras vs. PyTorch

최근 딥러닝 연구 대부분에서 PyTorch를 사용 중

Tensorflow, Keras처럼 독자적인 모듈이 아닌,
기초적인 Python 구문으로 제작되어
좀 더 기초적인 부분에서의 변화가 용이함

02. Course of Keras Study



03. Strengths of Keras

2주차 때의 기억을 떠올려 봅시다…

이번 주에 배운 여러 기법을 바탕으로 이전에 봤던 MNIST 데이터셋을 이용하여 딥러닝 구조를 PyTorch로 짜는 문제입니다 :)

Q1-1) 아래에 주어진 주석을 기반으로 하여 코딩을 해주세요.

- 모두의 딥러닝 시즌2 github 코드에 힌트가 있습니다 :) https://github.com/deeplearningzerotoall/PyTorch

```
'Fill this blank!
```

03. Strengths of Keras

사용했던 여러 기법들

Batch Normalization
Dropout
Xavier initialization
.
.
.
Train & Test & Predict

K 03. Strengths of Keras

PyTorch의 너무 긴 코드들…

```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import matplotlib.pylab as plt
import random


# 파라미터 설정 (learning rate, training epochs, batch_size)
learning_rate = 0.001
training_epochs = 15
batch_size = 100


#train과 test set으로 나누어 MNIST data 불러오기

mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)


#dataset loader에 train과 test할당하기(batch size, shuffle, drop_last 꼭 설정할 것!)

train_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           drop_last=True)

test_loader = torch.utils.data.DataLoader(dataset=mnist_test,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          drop_last=True)


# Layer 쌓기 (조건: 3개의 Layer 사용, DropOut 사용 (p=0.3), ReLU 함수 사용, Batch normalization하기)

# 각 Layer의 Hidden node 수 : 1st Layer (784,100), 2nd Layer(100,100),3rd Layer(100,10)

linear1 = torch.nn.Linear(784, 100, bias=True)
linear2 = torch.nn.Linear(100, 100, bias=True)
linear3 = torch.nn.Linear(100, 10, bias=True)
relu = torch.nn.ReLU()
bn1 = torch.nn.BatchNorm1d(100)
bn2 = torch.nn.BatchNorm1d(100)
dropout = torch.nn.Dropout(p=0.3)


#xavier initialization을 이용하여 각 layer의 weight 초기화

torch.nn.init.xavier_uniform_(linear1.weight)
torch.nn.init.xavier_uniform_(linear2.weight)
```

```
#model 정의하기(쌓는 순서: linear-Batch Normalization layer - ReLU- DropOut)
bn_model = torch.nn.Sequential(linear1, bn1, relu, dropout,
                               linear2, bn2, relu,dropout,
                               linear3)

# Loss Function 정의하기 (CrossEntropy를 사용할 것!)
criterion = torch.nn.CrossEntropyLoss()

#optimizer 정의하기 (Adam optimizer를 사용할 것!)
bn_optimizer = torch.optim.Adam(bn_model.parameters(), lr=learning_rate)
```

```
#cost 계산을 위한 변수 설정
train_total_batch = len(train_loader)


#Training epoch (cost 값 초기 설정(0으로)과 model의 train 설정 꼭 할 것)
for epoch in range(training_epochs):
    avg_cost = 0
    bn_model.train()  # set the model to train mode


    #train dataset을 불러오고, back propagation을 통해 loss를 최적화하는 과정
    for X, Y in train_loader:
        X = X.view(-1, 28 * 28)
        Y = Y

        bn_optimizer.zero_grad()
        bn_prediction = bn_model(X)
        bn_loss = criterion(bn_prediction, Y)
        bn_loss.backward()
        bn_optimizer.step()

        avg_cost += bn_loss / train_total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning finished')
```

```
with torch.no_grad():
    bn_model.eval()
    X_test = mnist_test.test_data.view(-1, 28 * 28).float()
    Y_test = mnist_test.test_labels

    bn_acc = 0
    bn_prediction = bn_model(X_test)
    bn_correct_prediction = torch.argmax(bn_prediction, 1) == Y_test
    bn_loss += criterion(bn_prediction, Y_test)
    bn_acc += bn_correct_prediction.float().mean()

    print("Accuracy: ", bn_acc.item())



    ##Test set에서 random으로 data를 뽑아 Label과 Prediction을 비교하는 코드
    r = random.randint(0, len(mnist_test)-1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 *28).float()
    Y_single_data = mnist_test.test_labels[r:r + 1]

    print('Label: ', Y_single_data.item())
    single_prediction = bn_model(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())
```

03. Strengths of Keras

모델의 전반적인 과정에서 Keras와 PyTorch를 비교해보자!

03. Strengths of Keras

MNIST 데이터 불러오기

PyTorch

```
#train과 test set으로 나누어 MNIST data 불러오기

mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)


#dataset loader에 train과 test할당하기(batch size, shuffle, drop_last 잘 설정할 것!)

train_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           drop_last=True)

test_loader = torch.utils.data.DataLoader(dataset=mnist_test,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          drop_last=True)
```

K

```
from keras.datasets import mnist
(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()
```

```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

03. Strengths of Keras

모델 쌓기

PyTorch

```
linear1 = torch.nn.Linear(784, 100, bias=True)
linear2 = torch.nn.Linear(100, 100, bias=True)
linear3 = torch.nn.Linear(100, 10, bias=True)
relu = torch.nn.ReLU()
bn1 = torch.nn.BatchNorm1d(100)
bn2 = torch.nn.BatchNorm1d(100)
dropout = torch.nn.Dropout(p=0.3)


#xavier initialization을 이용하여 각 layer의 weight 초기화

torch.nn.init.xavier_uniform_(linear1.weight)
torch.nn.init.xavier_uniform_(linear2.weight)



#model 정의하기(쌓는 순서: linear-Batch Normalization layer - ReLU- DropOut)
bn_model = torch.nn.Sequential(linear1, bn1, relu, dropout,
                               linear2, bn2, relu,dropout,
                               linear3)
```

K

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(784, activation='relu', kernel_initializer='glorot_uniform'))
model.add(keras.layers.Dropout(rate=0.3))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Dense(100, activation='relu', kernel_initializer='glorot_uniform'))
model.add(keras.layers.Dropout(rate=0.3))
model.add(keras.layers.Dense(10,activation='softmax'))
```

03. Strengths of Keras

Loss function, Optimizer 설정 후 Training하기

PyTorch

```
# Loss Function 정의하기 (CrossEntropy를 사용할 것!)
criterion = torch.nn.CrossEntropyLoss()

#optimizer 정의하기 (Adam optimizer를 사용할 것!)
bn_optimizer = torch.optim.Adam(bn_model.parameters(), lr=learning_rate)

#cost 계산을 위한 변수 설정
train_total_batch = len(train_loader)


#Training epoch (cost 값 초기 설정(0으로)과 model의 train 설정 꼭 할 것)
for epoch in range(training_epochs):
    avg_cost = 0
    bn_model.train()  # set the model to train mode


    #train dataset을 불러오고, back propagation을 통해 loss를 최적화하는 과
    for X, Y in train_loader:
        X = X.view(-1, 28 * 28)
        Y = Y

        bn_optimizer.zero_grad()
        bn_prediction = bn_model(X)
        bn_loss = criterion(bn_prediction, Y)
        bn_loss.backward()
        bn_optimizer.step()

        avg_cost += bn_loss / train_total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning finished')
```

```
Epoch: 0001 cost = 0.473720551
Epoch: 0002 cost = 0.225975260
Epoch: 0003 cost = 0.184662297
Epoch: 0004 cost = 0.161234275
Epoch: 0005 cost = 0.143860355
Epoch: 0006 cost = 0.134247929
Epoch: 0007 cost = 0.123788722
Epoch: 0008 cost = 0.115821987
Epoch: 0009 cost = 0.111141421
Epoch: 0010 cost = 0.104314081
Epoch: 0011 cost = 0.104165316
Epoch: 0012 cost = 0.100740276
Epoch: 0013 cost = 0.094482362
Epoch: 0014 cost = 0.095930628
Epoch: 0015 cost = 0.090113886
Learning finished
```

K

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

history = model.fit(X_train, y_train, batch_size=100, epochs=15,validation_data=(X_valid, y_valid))
```

```
Epoch 1/15
1719/1719 [==============================] - 18s 10ms/step - loss: 0.3176 - accuracy: 0.9043 - val_loss: 0.1351 - val_accuracy: 0.9606
Epoch 2/15
1719/1719 [==============================] - 17s 10ms/step - loss: 0.1790 - accuracy: 0.9460 - val_loss: 0.1279 - val_accuracy: 0.9684
Epoch 3/15
1719/1719 [==============================] - 17s 10ms/step - loss: 0.1498 - accuracy: 0.9543 - val_loss: 0.1261 - val_accuracy: 0.9716
Epoch 4/15
1719/1719 [==============================] - 17s 10ms/step - loss: 0.1319 - accuracy: 0.9585 - val_loss: 0.1154 - val_accuracy: 0.9764
Epoch 5/15
1719/1719 [==============================] - 17s 10ms/step - loss: 0.1216 - accuracy: 0.9624 - val_loss: 0.1153 - val_accuracy: 0.9780
Epoch 6/15
1719/1719 [==============================] - 17s 10ms/step - loss: 0.1086 - accuracy: 0.9656 - val_loss: 0.1651 - val_accuracy: 0.9776
Epoch 7/15
1719/1719 [==============================] - 18s 10ms/step - loss: 0.1033 - accuracy: 0.9679 - val_loss: 0.1222 - val_accuracy: 0.9768
Epoch 8/15
1719/1719 [==============================] - 18s 10ms/step - loss: 0.0942 - accuracy: 0.9704 - val_loss: 0.1884 - val_accuracy: 0.9788
Epoch 9/15
1719/1719 [==============================] - 18s 10ms/step - loss: 0.0897 - accuracy: 0.9711 - val_loss: 0.2052 - val_accuracy: 0.9784
Epoch 10/15
1719/1719 [==============================] - 18s 11ms/step - loss: 0.0883 - accuracy: 0.9718 - val_loss: 0.2680 - val_accuracy: 0.9752
Epoch 11/15
1719/1719 [==============================] - 19s 11ms/step - loss: 0.0844 - accuracy: 0.9746 - val_loss: 0.2012 - val_accuracy: 0.9770
Epoch 12/15
1719/1719 [==============================] - 19s 11ms/step - loss: 0.0820 - accuracy: 0.9748 - val_loss: 0.2173 - val_accuracy: 0.9782
Epoch 13/15
1719/1719 [==============================] - 20s 11ms/step - loss: 0.0744 - accuracy: 0.9756 - val_loss: 0.1586 - val_accuracy: 0.9830
Epoch 14/15
1719/1719 [==============================] - 20s 12ms/step - loss: 0.0752 - accuracy: 0.9758 - val_loss: 0.1788 - val_accuracy: 0.9788
Epoch 15/15
1719/1719 [==============================] - 18s 10ms/step - loss: 0.0707 - accuracy: 0.9782 - val_loss: 0.1207 - val_accuracy: 0.9812
```

03. Strengths of Keras

Test data의 Accuracy 체크

PyTorch

```
with torch.no_grad():
    bn_model.eval()
    X_test = mnist_test.test_data.view(-1, 28 * 28).float()
    Y_test = mnist_test.test_labels

    bn_acc = 0
    bn_prediction = bn_model(X_test)
    bn_correct_prediction = torch.argmax(bn_prediction, 1) == Y_test
    bn_loss += criterion(bn_prediction, Y_test)
    bn_acc += bn_correct_prediction.float().mean()

    print("Accuracy: ", bn_acc.item())
```

```
Accuracy:  0.9654999971389771
```

```
model.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1053 - accuracy: 0.9790
[0.10534383356571198, 0.9789999723434448]
```

03. Strengths of Keras

Prediction

PyTorch

```
r = random.randint(0, len(mnist_test)-1)
X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 *28).float()
Y_single_data = mnist_test.test_labels[r:r + 1]

print('Label: ', Y_single_data.item())
single_prediction = bn_model(X_single_data)
print('Prediction: ', torch.argmax(single_prediction, 1).item())
```

```
Label:  3
Prediction:  3
```

K

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
```

```
array([[0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

```
y_pred = np.argmax(model.predict(X_new), axis=-1)
y_pred
```

```
array([7, 2, 1])
```

```
np.array(class_names)[y_pred]
```

```
array(['7', '2', '1'], dtype='<U1')
```

```
plt.figure(figsize=(7.2, 2.4))
for index, image in enumerate(X_new):
    plt.subplot(1, 3, index + 1)
    plt.imshow(image, cmap="binary", interpolation="nearest")
    plt.axis('off')
    plt.title(class_names[y_test[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
plt.show()
```

03. Strengths of Keras

Visualization on Keras

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
batch_normalization_4 (Batch (None, 784)               3136
_________________________________________________________________
dense_6 (Dense)              (None, 784)               615440
_________________________________________________________________
dropout_4 (Dropout)          (None, 784)               0
_________________________________________________________________
batch_normalization_5 (Batch (None, 784)               3136
_________________________________________________________________
dense_7 (Dense)              (None, 100)               78500
_________________________________________________________________
dropout_5 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1010
=================================================================
Total params: 701,222
Trainable params: 698,086
Non-trainable params: 3,136
_________________________________________________________________
```

model.summary( ) 라는 간단한 문구로

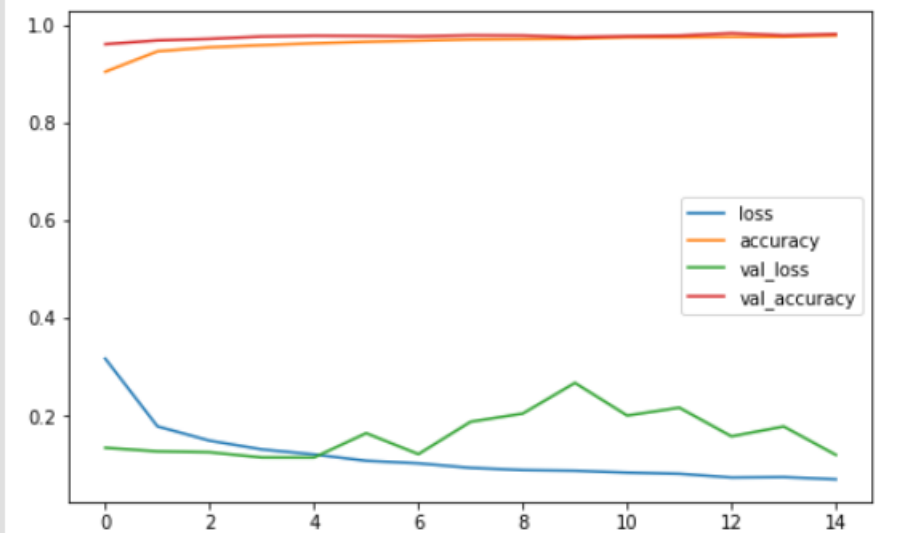
Model의 전체적인 형태를 한 눈에 볼 수 있다.

03. Strengths of Keras

Visualization on Keras

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

history = model.fit(X_train, y_train, batch_size=100, epochs=15,validation_data=(X_valid, y_valid))
```

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.show()
```



Training을 진행하며 각 epoch에서의

Loss, Accuracy, Validation Loss, Validation Accuracy가

자동으로 DataFrame 형태로 저장된다.

이를 이용하여 쉽게 시각화가 가능하다.

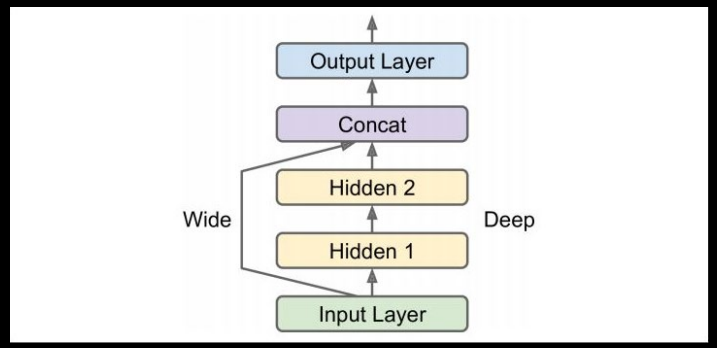
03. Strengths of Keras

독립적이고 직관적인 모듈들로 이루어진 Keras

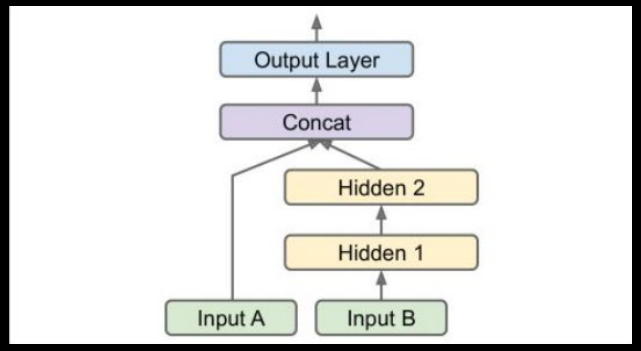
덕분에 완전 초보도 금방 간단한 모델을 구축할 수 있다!

https://keras.io/ko/

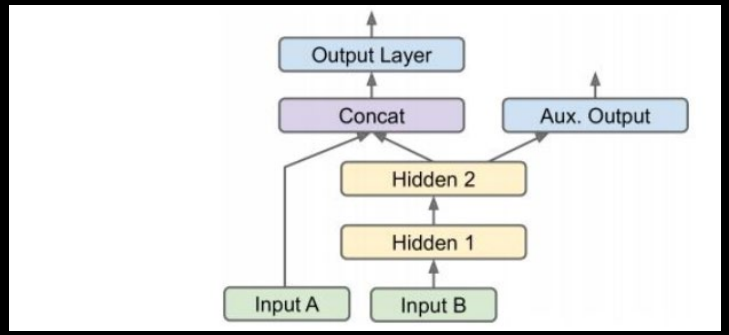
04. And More…




```
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.models.Model(inputs=[input_], outputs=[output])
```




```
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.models.Model(inputs=[input_A, input_B], outputs=[output])
```




```
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="main_output")(concat)
aux_output = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.models.Model(inputs=[input_A, input_B],
                           outputs=[output, aux_output])
```

05. Conclusion

우리가 어떤 건축물을 디자인한다고 생각해보자

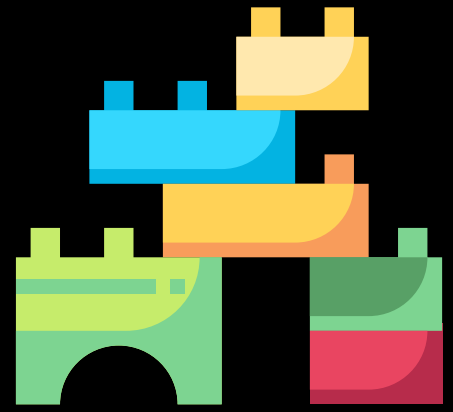
PyTorch

누구나 연필을 쓸 줄 안다.

화려하고 복잡한 건축물을 디자인 할 수 있다.

하지만 실현이 쉽지 않다.

K



레고 자체에 대해 알아야 한다.

엄청 복잡하고 화려한 건축물을 만들기는 어렵다.

그러나 방법만 알면 쉽게 건축물을 만들 수 있다.

05. Conclusion

간단한 모델 구축

PyTorch

05. Conclusion

복잡한 모델 구축

PyTorch

Q&A

끝!