Keras를 통한 네트워크 구현 및 활용

정보컴퓨터공학과 권혁동

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

이전 구현

라이브러리를 통한 구현 방법

네트워크 학습 및 활용 시연

결론

이전 구현

- 파이썬 클래스와 메소드를 조합하여 단일 뉴런과 레이어 구현
  - Input layer → Activation 순으로 메소드 호출
- 직접 코드를 작성하여 구현하는 경우, **구현의 난이도가 높음**
  - 다층 레이어를 형성한다면 레이어 간 연결도 어려움
  - 뉴런이 많아질 수록 난이도가 증가
- 일반적으로 사용되는 **라이브러리를 활용**하여 구현
  - 구현의 난이도가 낮아지고 **빠른 속도로 구현**이 가능

라이브러리를 통한 구현 방법

- Keras를 사용한 네트워크 구현
  - Tensorflow의 일종
- 기존과 동일하나, **뉴런 단위가 아닌 레이어 단위로 구현**
- '레이어+활성화 함수'가 1계층 레이어를 형성
  - e.g.) 'input+activation' + 'dense+activation'의 경우 2계층 레이어

모델 선언

Input layer

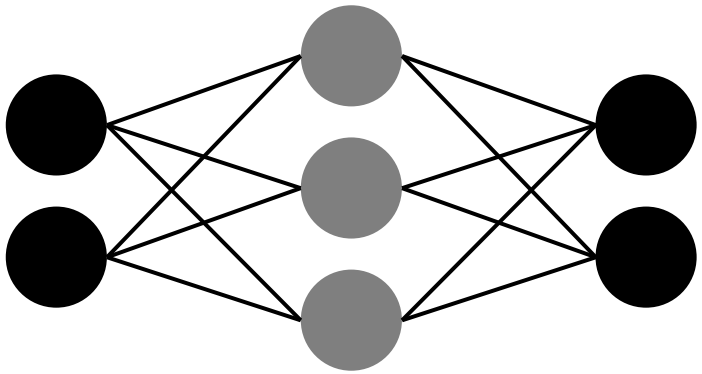
Hidden layer

Output layer

```
model = Sequential()
model.add(Flatten(input_shape = (28, 28)))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(256))
model.add(Activation('softmax'))

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

라이브러리를 통한 구현 방법

- Input layer의 Flatten은 이미지를 1차원 배열로 펼침
  - 28x28 array → 784 array
- Dense layer는 Fully-connected layer를 형성
- Activation function은 Relu 사용
- Loss function은 sparse categorical crossentropy 사용
- Optimizer는 adam 사용



```
model = Sequential()
model.add(Flatten(input_shape = (28, 28)))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(256))
model.add(Activation('softmax'))

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

라이브러리를 통한 구현 방법

- 모델 구성 이후 compile 메소드를 통해서 모델 완성

- Summary 메소드는 모델의 전체적인 형태를 요약

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 flatten_1 (Flatten)          (None, 784)               0

 dense_1 (Dense)              (None, 256)               200960

 activation_4 (Activation)    (None, 256)               0

 dense_2 (Dense)              (None, 256)               65792

 activation_5 (Activation)    (None, 256)               0

=================================================================
Total params: 266,752
Trainable params: 266,752
Non-trainable params: 0
_________________________________________________________________
```

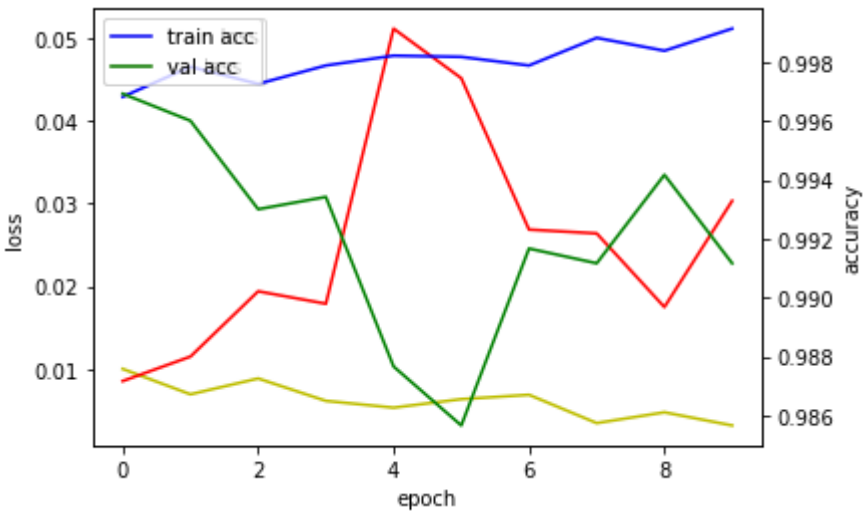
```
model = Sequential()
model.add(Flatten(input_shape = (28, 28)))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(256))
model.add(Activation('softmax'))

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

라이브러리를 통한 구현 방법

- Fit 메소드를 통해서 학습을 진행
  - History에 학습 상태를 기록
- Evaluate 메소드를 통해 학습한 모델을 평가

```
history = model_flat.fit(x_train, y_train, validation_data=(x_valid, y_valid), epochs=10)
```

```
Epoch 1/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0086 - val_accuracy: 0.9969
Epoch 2/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0070 - accuracy: 0.9978 - val_loss: 0.0115 - val_accuracy: 0.9960
Epoch 3/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0089 - accuracy: 0.9973 - val_loss: 0.0194 - val_accuracy: 0.9930
Epoch 4/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0062 - accuracy: 0.9979 - val_loss: 0.0179 - val_accuracy: 0.9934
Epoch 5/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.0511 - val_accuracy: 0.9877
Epoch 6/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0064 - accuracy: 0.9982 - val_loss: 0.0451 - val_accuracy: 0.9857
Epoch 7/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0069 - accuracy: 0.9979 - val_loss: 0.0269 - val_accuracy: 0.9917
Epoch 8/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0035 - accuracy: 0.9988 - val_loss: 0.0264 - val_accuracy: 0.9912
Epoch 9/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0048 - accuracy: 0.9984 - val_loss: 0.0175 - val_accuracy: 0.9942
Epoch 10/10
1500/1500 [==============================] - 3s 2ms/step - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0303 - val_accuracy: 0.9912
```




```
test_loss, test_acc = model_flat.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.1058 - accuracy: 0.9790
```

라이브러리를 통한 구현 방법

- Save 메소드를 통해서 학습이 완료된 **모델을 추출** 가능
  - Load 메소드를 사용하여 모델을 탑재
  - 추출한 모델을 프로그램이나 기기에 탑재하여 활용 가능
- 모델을 탑재하여 활용하는 것으로 작성한 모델을 가시적으로 확인

```
model_flat.save('mnist_mlp_model.h5')
```


```
if fname:
    self.loaded_model = tf.keras.models.load_model(fname)
    self.statusbar.showMessage('Model loaded.')
```

네트워크 학습 및 시연

- 시연은 유튜브 세미나 영상을 참고해주세요
- https://youtu.be/VLXo6XZRqYo

결론

- Tensorflow Keras 라이브러리를 통한 네트워크 구현
  - 네트워크를 구현하기 위해서는 **레이어를 쌓아 올리기**만 진행
  - 나머지는 라이브러리가 전부 처리!
- MNIST 데이터 셋을 사용해서 학습 진행
  - 학습 → 평가 순으로 진행
  - 생성한 모델을 추출 가능
- 모델을 프로그램에 탑재해서 **실제로 활용 가능**함을 보임
  - 동일한 방법으로 특정 기기에도 탑재 가능

Q & A