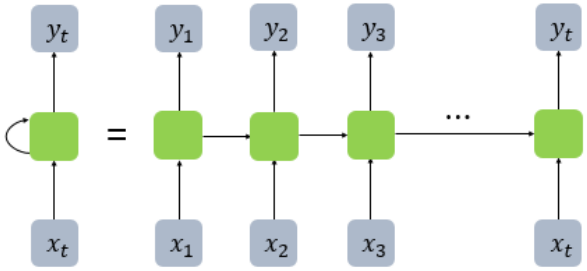
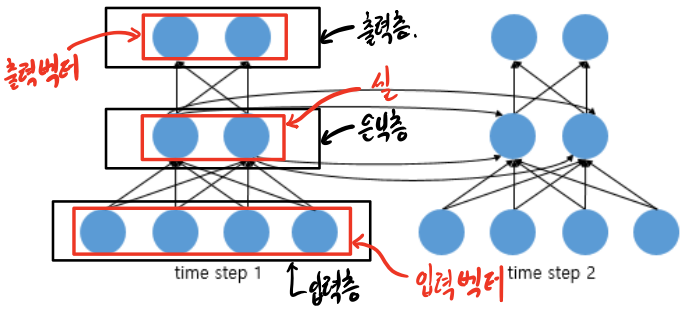
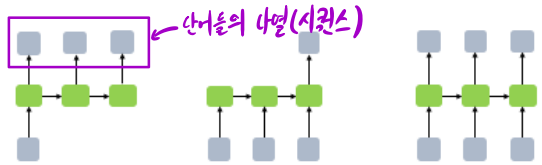


RNN을 표현할 때는 일반적으로 위의 그림에서 좌측과 같이 화살표로 사이클을 그려서 재귀 형태로 표현하기도 하지만, 우측과 같이 사이클을 그리는 화살표 대신 여러 시점으로 펼쳐서 표현하기도 합니다. 두 그림은 동일한 그림으로 단지 사이클을 그리는 화살표를 사용하여 표현하였느냐, 시점의 흐름에 따라서 표현하였느냐의 차이일 뿐 둘 다 동일한 RNN을 표현하고 있습니다.

피드 포워드 신경망에서는 뉴런이라는 단위를 사용했지만 RNN에서는 뉴런이라는 단위보다는 입력층과 출력층에서는 각각 입력 벡터와 출력 벡터, 은닉층에서는 은닉 상태라는 표현을 주로 사용합니다. 그래서 사실 위의 그림에서 회색과 초록색으로 표현한 각 네모들은 기본적으로 벡터 단위를 가정하고 있습니다. 피드 포워드 신경망과의 차이를 비교하기 위해서 RNN을 뉴런 단위로 시각화해보겠습니다.

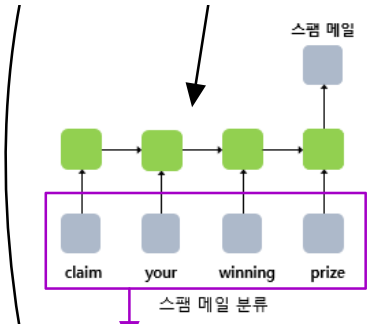


위의 그림은 입력 벡터의 차원이 4, 은닉 상태의 크기가 2, 출력층의 출력 벡터의 차원이 2인 RNN이 시점이 2일 때의 모습을 보여줍니다. 다시 말해 뉴런 단위로 해석하면 입력층의 뉴런 수는 4, 은닉층의 뉴런 수는 2, 출력층의 뉴런 수는 2입니다.

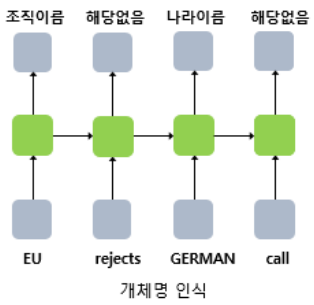


RNN은 입력과 출력의 길이를 다르게 설계 할 수 있으므로 다양한 용도로 사용할 수 있습니다. 위 그림은 입력과 출력의 길이에 따라서 달라지는 RNN의 다양한 형태를 보여줍니다. 위 구조가 자연어 처리에서 어떻게 사용될 수 있는지 예를 들어봅시다. RNN 셀의 각 시점 별 입, 출력의 단위는 사용자가 정의하기 나름이지만 가장 보편적인 단위는 '단어 벡터'입니다.

예를 들어 하나의 입력에 대해서 여러개의 출력(one-to-many)의 모델은 하나의 이미지 입력에 대해서 사진의 제목을 출력하는 이미지 캡셔닝(Image Captioning) 작업에 사용할 수 있습니다. 사진의 제목은 단어들의 나열이므로 시퀀스 출력입니다.



또한 단어 시퀀스에 대해서 하나의 출력(many-to-one)을 하는 모델은 입력 문서가 긍정적인지 부정적인지를 판별하는 감성 분류(sentiment classification), 또는 메일이 정상 메일인지 스팸 메일인지 판별하는 스팸 메일 분류(spam detection)에 사용할 수 있습니다. 위 그림은 RNN으로 스팸 메일을 분류할 때의 아키텍처를 보여줍니다. 이러한 예제들은 **11챕터에서 배우는 텍스트 분류**에서 배웁니다.



다 대 다(many-to-many)의 모델의 경우에는 입력 문장으로 부터 대답 문장을 출력하는 챗봇과 입력 문장으로부터 번역된 문장을 출력하는 번역기, 또는 **12챕터에서 배우는 개체명 인식이나 품사 태깅과 같은 작업** 또한 속합니다. 위 그림은 개체명 인식을 수행할 때의 RNN 아키텍처를 보여줍니다.

2. 케라스(Keras)로 RNN 구현하기

케라스로 RNN 층을 추가하는 코드는 다음과 같습니다.

```
# RNN 층을 추가하는 코드.
model.add(SimpleRNN(hidden_size)) # 가장 간단한 형태
```

인자를 사용할 때를 보겠습니다.

```
# 추가 인자를 사용할 때
model.add(SimpleRNN(hidden_size, input_shape=(timesteps, input_dim)))

# 다른 표기
model.add(SimpleRNN(hidden_size, input_length=M, input_dim=N))
# 단, M과 N은 정수
```

hidden_size = 은닉 상태의 크기를 정의. 메모리 셀이 다음 시점의 메모리 셀과 출력층으로 보내는 값의 크기(output_dim)와도 동일. RNN의 용량(capacity)을 늘린다고 보면 되며, 중소형 모델의 경우 보통 128, 256, 512, 1024 등의 값을 가진다.
timesteps = 입력 시퀀스의 길이(input_length)라고 표현하기도 함. 시점의 수.
input_dim = 입력의 크기.

① "한 문장 내 단어 수만큼 RNN을 돌아."
② 해당 매개변수는 한 데이터(벡터)의 dimension과 같다.

다음에 등장할 숫자를 맞추는 many-to-one을 구현해보겠습니다.

$$[1, 2, 3] \rightarrow [4]$$
$$[2, 3, 4] \rightarrow [5]$$
$$[3, 4, 5] \rightarrow [6]$$

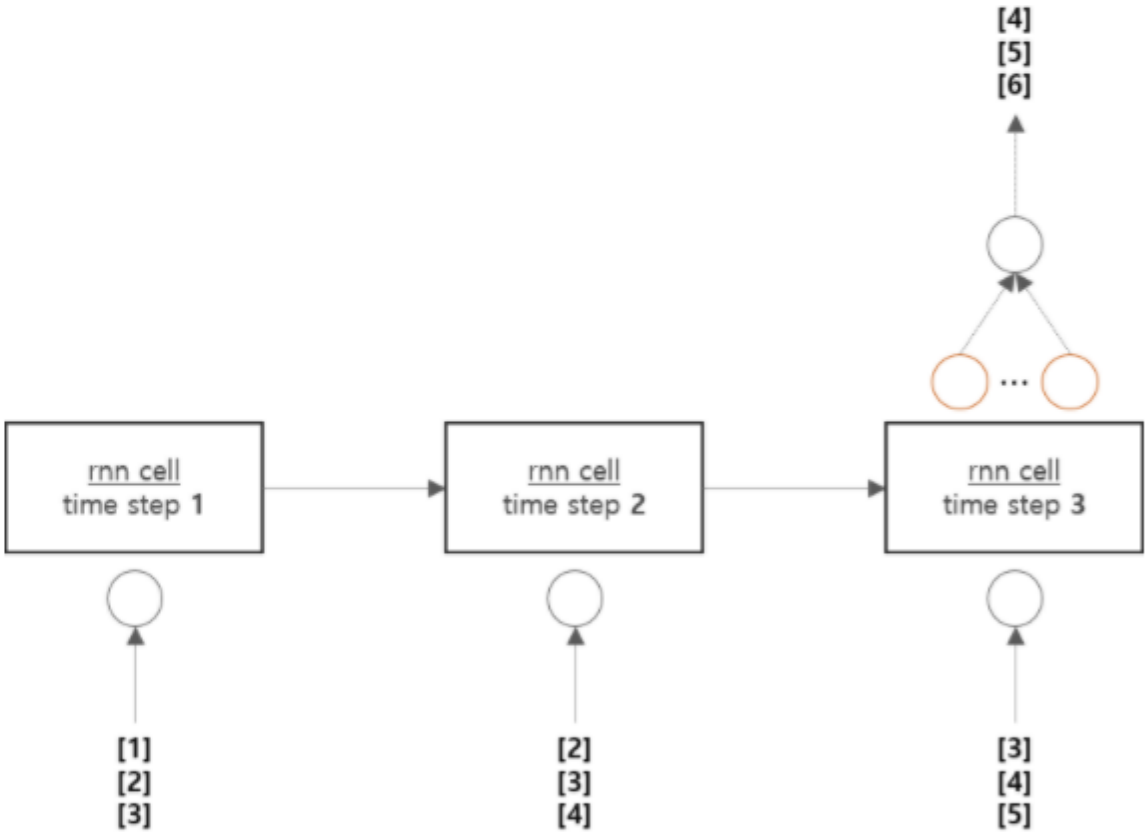


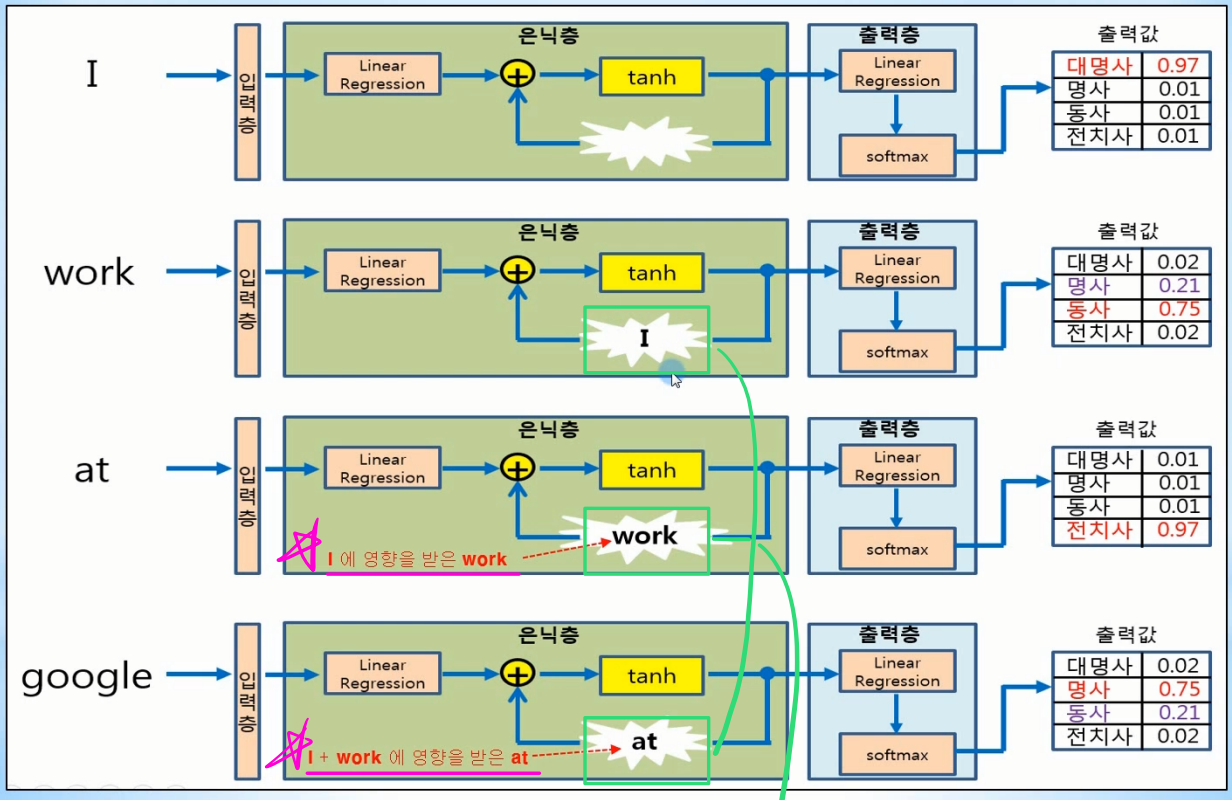
Figure 1: Many-to-One RNN

이어서 Tensorflow의 keras를 이용하여 모형 선언을 해주겠습니다. SimpleRNN 함수를 이용하여 은닉 노드가 100개인 RNN Cell을 쉽게 구축할 수 있습니다. RNN에서 100개의 노드가 출력되면 이를 Dense를 이용하여 하나로 묶어줍니다. 이어서 학습을 위한 손실 함수와 최적화기는 각각 mean squared errror와 adam을 적용해줍니다.

```
# Make Model
layer_input  = keras.Input(shape=(3, 1), name='input')
layer_rnn    = keras.layers.SimpleRNN(100, name='RNN')(layer_input)
layer_output = keras.layers.Dense(1, name='output')(layer_rnn)

model = keras.Model(layer_input, layer_output)
print(model.summary())

# Complier
model.compile(loss = 'mse', optimizer='adam')
model._name = 'many_to_one'

# [ Output ]
# Model: "many_to_one"
# _________________________________________________________________
# Layer (type)                 Output Shape              Param #
# =================================================================
# input (InputLayer)           [(None, 3, 1)]            0
# _________________________________________________________________
# RNN (SimpleRNN)              (None, 100)               10200
# _________________________________________________________________
# output (Dense)               (None, 1)                 101
# =================================================================
# Total params: 10,301
# Trainable params: 10,301
# Non-trainable params: 0
# _________________________________________________________________
```

다음으론 위에서 선언한 모형을 학습하고 학습데이터와 [4, 5, 6]을 검증 데이터 (test data)를 적용해보겠습니다. 예측값은 predict 메소드를 이용하면 구할 수 있습니다. (*데이터가 적고, 최적화된 구조도 아니라서 결과가 좋진 않습니다. Many-to-One 구조를 이렇게 구현할 수 있구나.. 정도만 알고 가셔도 충분할 것 같습니다.*)

RNN 동작원리 – 정성적 분석 (I work at google)



5/7

hidden state.