Python 마지막차

1. 일반적인 신경망 학습의 과정은?

- 1. 미니배치
  - 훈련 데이터 무작위 선발
- 2. 기울기 산출
  - 손실함수의 기울기를 산출
- 3. 매개변수 갱신
  - 더 최적의 매개변수로 갱신
- 4. 1~3반복.

2. 활성화 함수가 무엇인지, 그 예시

- 입력신호의 총합을 출력 신호로 변환하는 함수
- 입력신호의 총합이 활성화를 일으키는지를 정하는 역할을 함.

- Ex) 계단함수, 시그모이드 함수, ReLU 함수,

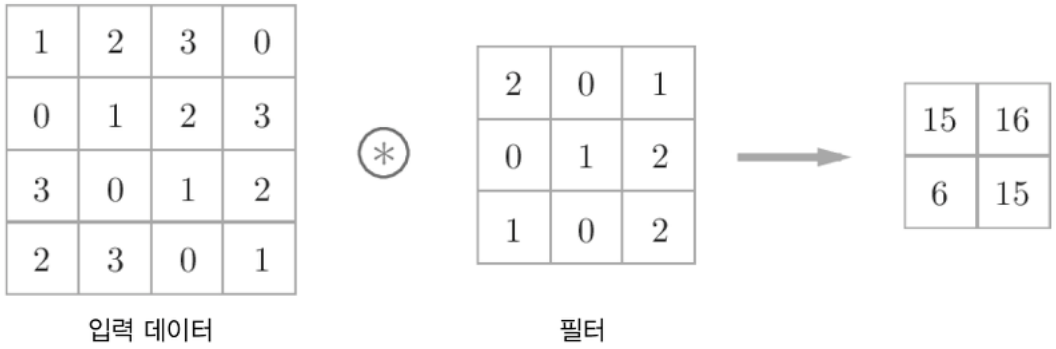
3. 오버피팅 이란?

- 신경망이 훈련 데이터에만 지나치게 적응되어 그 외의 데이터에는 제대로 대응하지 못하는 상태
- 범용 성능이 떨어지므로 가치를 잃음
- 매개변수가 많고 표현력이 높거나, 훈련 데이터가 적은 모델에서 주로 일어남.

- 가중치 감소, 드롭아웃으로 억제 가능.

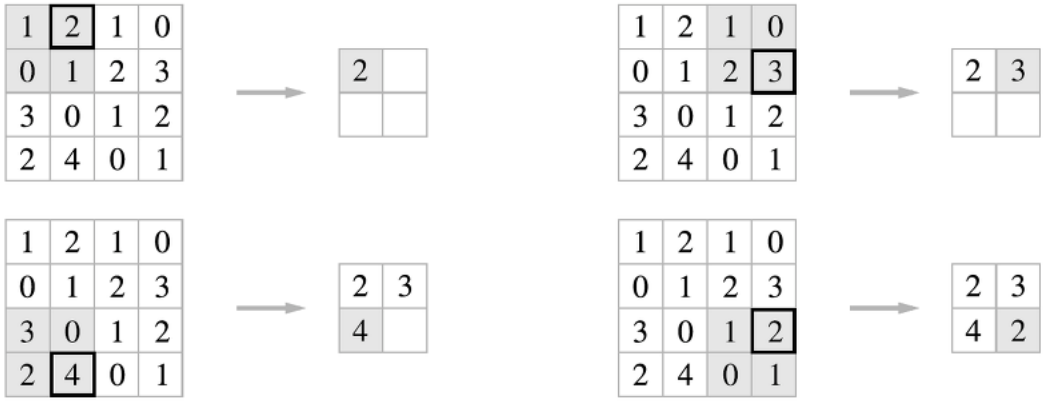
4- CNN의 핵심인 두 계층

- 합성곱(convolutional) 계층
  - 필터를 이용해 합성곱 연산을 수행한다. 필터(커널 = 가중치)의 윈도우를 일정 간격으로 이동해가며 입력 데이터에 적용한다.
  - 딥러닝과 달리 데이터 형상이 유지됨.



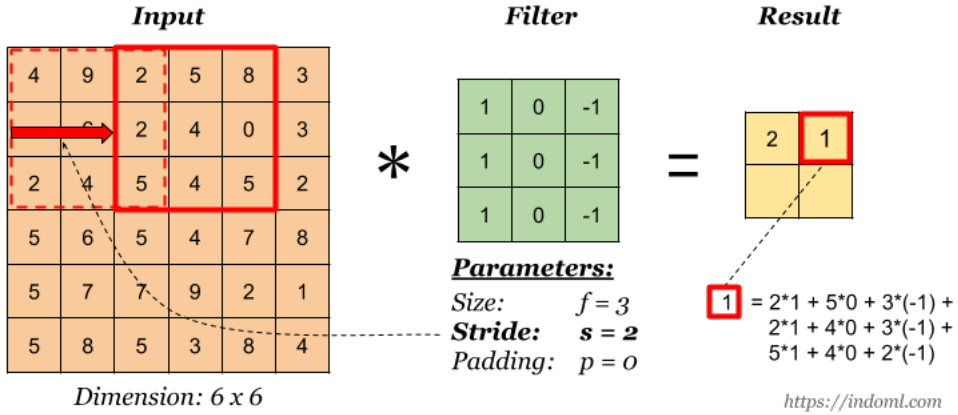
4- CNN의 핵심인 두 계층

- 풀링(pooling) 계층
  - 풀링은 2차원 데이터의 세로 및 가로 방향의 공간(매개변수)을 줄이는 연산
  - 풀링에는 최대 풀링(Max Pooling), 평균 풀링(Average Pooling) 등이 있다.
  - 최대 풀링은 대상 영역에서 최댓값을 취하는 연산, 평균 풀링은 대상 영역의 평균을 계산한다. 이미지 인식 분야에서는 주로 최대 풀링을 사용한다.

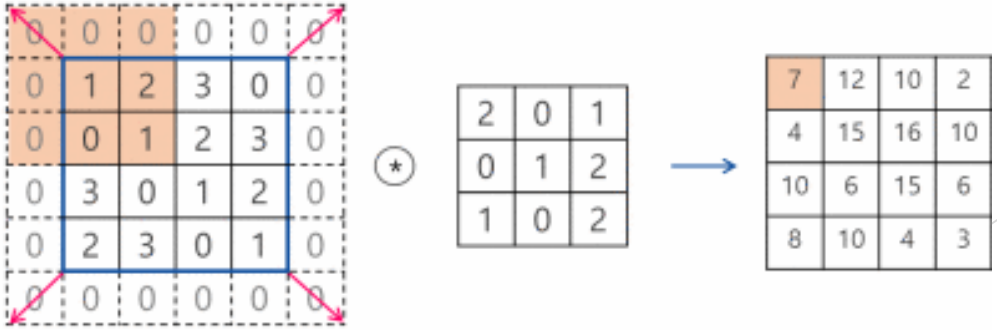


5. CNN

- Stride란?
  - Filter가 이동하는 칸 수.



- Padding란?
  - Img층의 최소한의 크기를 보장하기 위해 빈 테두리를 얼마나 추가하는지의 칸 수.



6. mnist 프로그래밍 과정

- Dataset loading
- Set up Model
- Session
- learning
- validation
- result

6. mnist 프로그래밍 과정

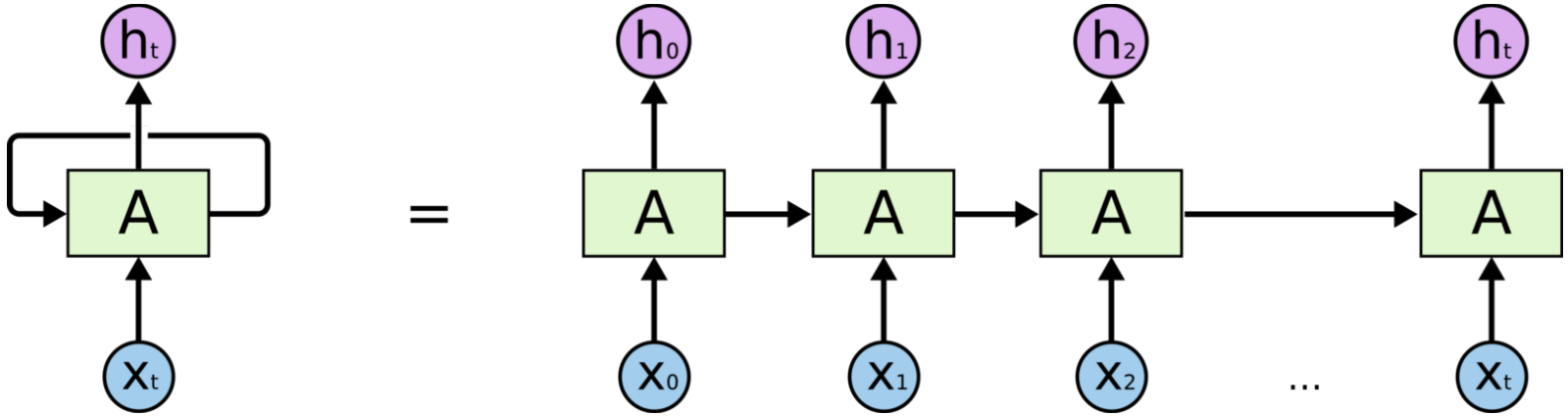
```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

# Dataset loading
mnist = input_data.read_data_sets("./samples/MNIST_data/", one_hot=True)

# Set up model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

6. mnist 프로그래밍 과정

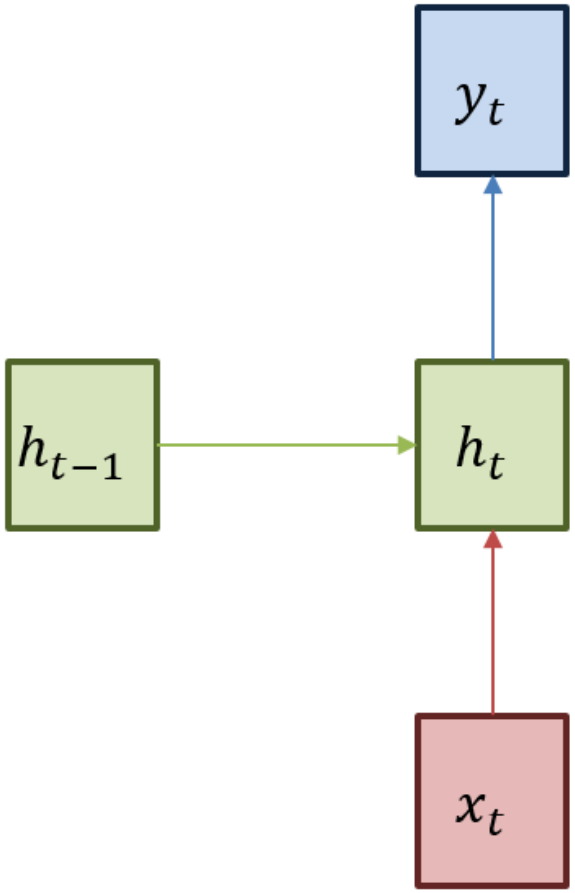
```
# Session
init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)

# Learning
for i in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Validation
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# Result should be approximately 91%.
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

RNN이란

- 순환신경망(recurrent neural network)
- 자연어처리(NLP)에서 주로 쓰임
- **시퀀스데이터**의 모델링에 사용
- 새로운 정보를 접할때 '**갱신**'된다는 아이디어가 RNN을 만듬.
- 과거 상태에 종속적이다.



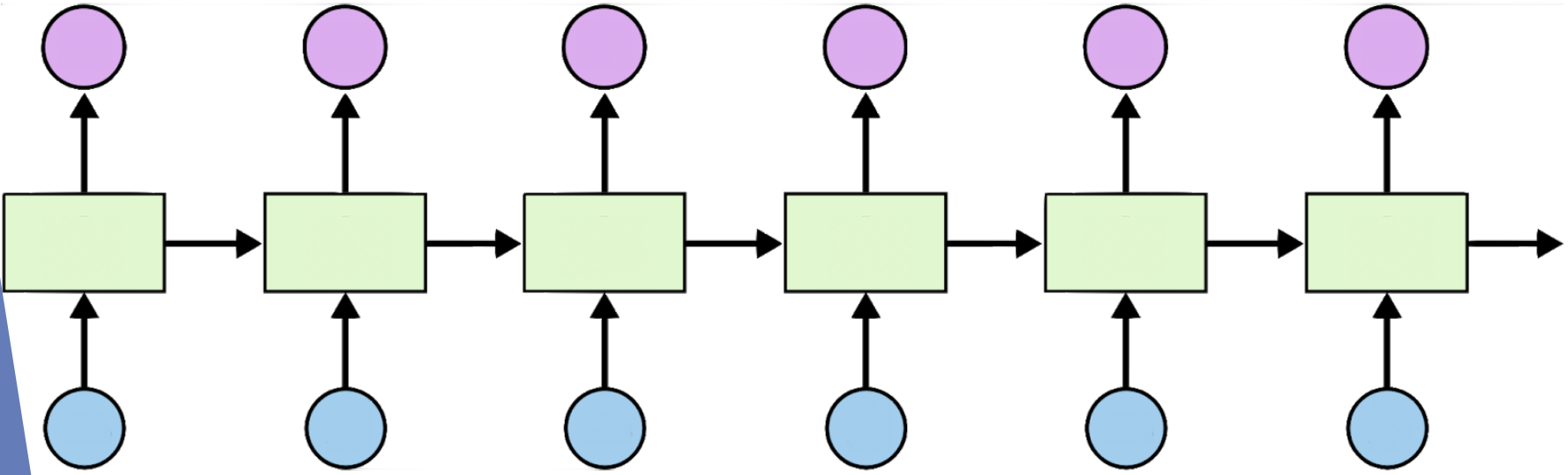
기본 구조



$$y_t = W_{hy} h_t + b_y$$

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

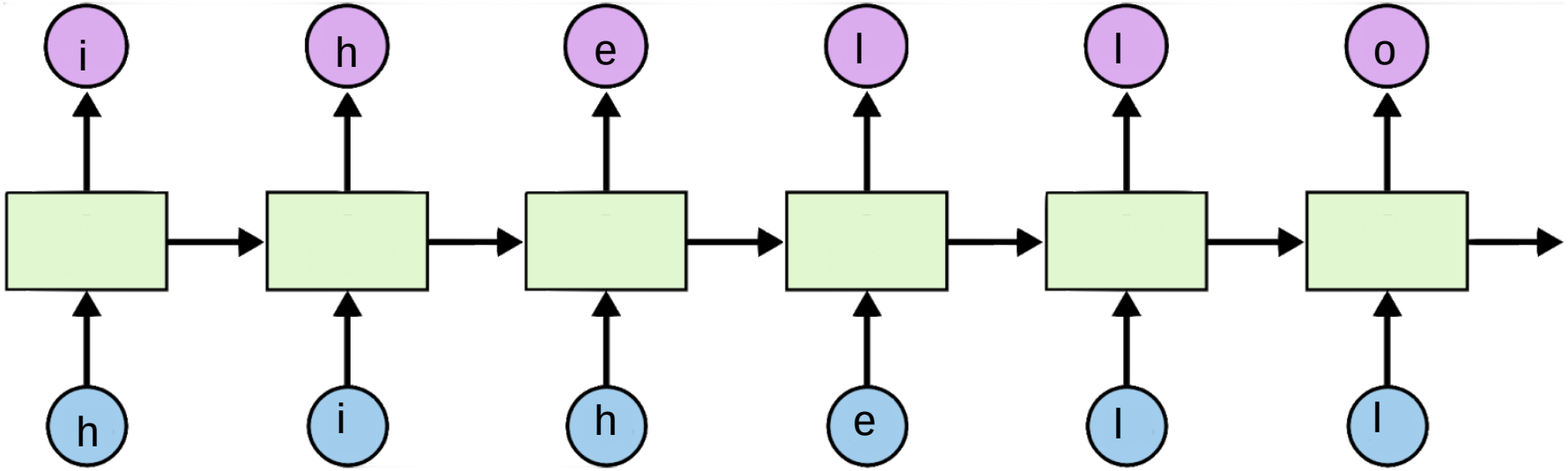
기본구조 - 예시



각각을 **cell** 이라 한다

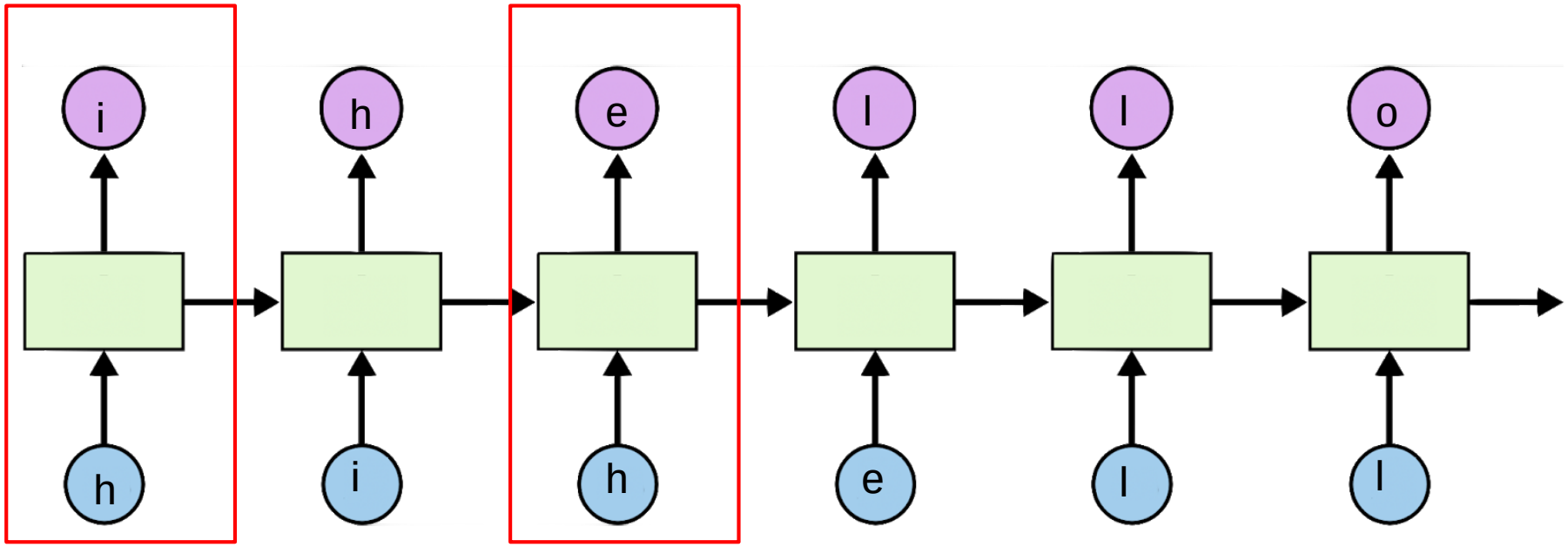
RNN 훈련 예시

- hihello 를 훈련 시키기.
- 하나의 문자를 입력하면, 그 다음 문자를 출력하도록 하는 프로그램 학습.
  - Reference : http://sshkim.tistory.com/153



RNN 훈련 예시

- 같은 h 입력인데, 어떨때는 i, 어떨때는 h 가 나옴.



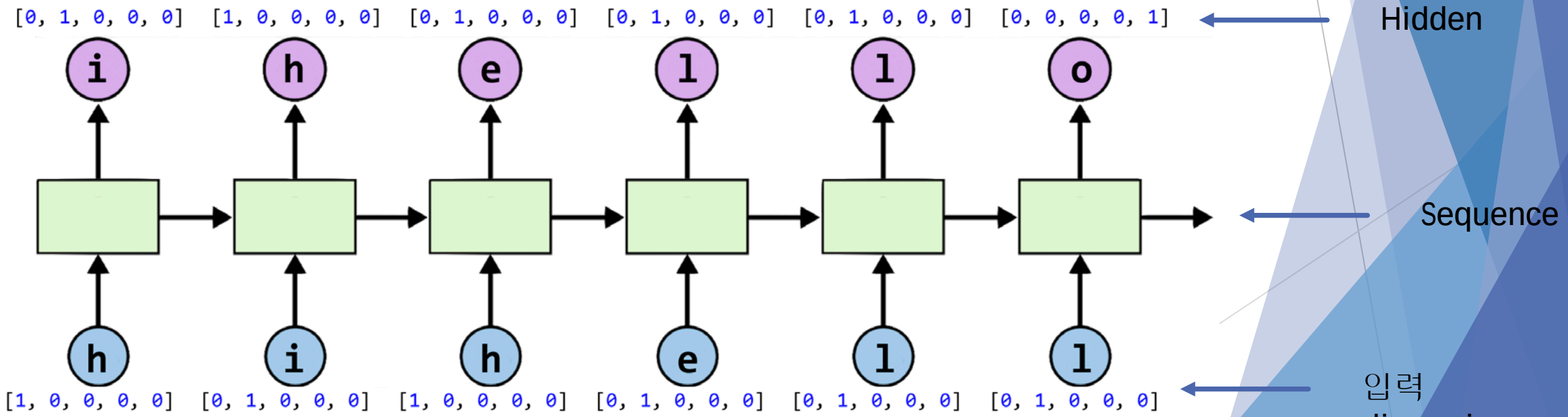
입력 데이터 정리

- 텍스트
  - hihello
- 사용할 문자 종류
  - h, i, e, l, o
- 문자의 index 매기기
  - h:0, i:1, e:2, l:3, o:4
- 문자 one-hot encoding 하기

```
[1, 0, 0, 0, 0],   # h 0
[0, 1, 0, 0, 0],   # i 1
[0, 0, 1, 0, 0],   # e 2
[0, 0, 0, 1, 0],   # l 3
[0, 0, 0, 0, 1],   # o 4
```

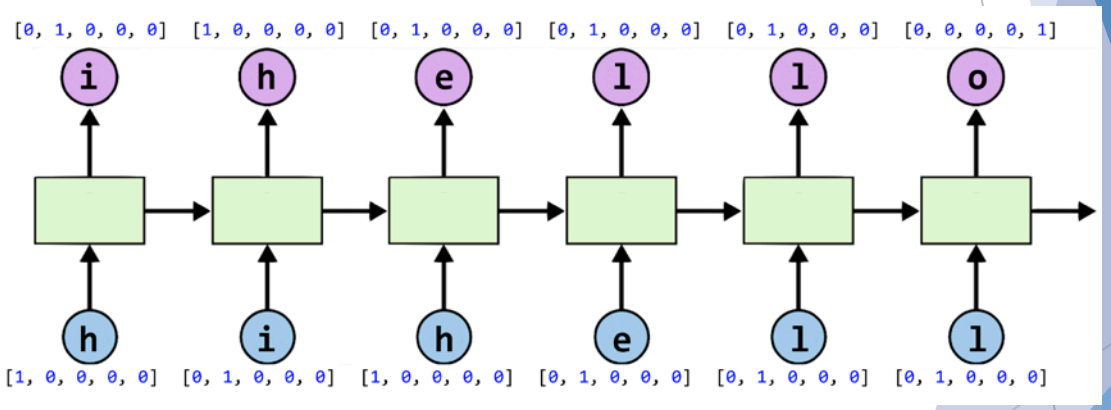
RNN 훈련 개요 파악

- 입력 dimension : 5 (사용 문자 갯수)
- Sequence 길이 : 6
- Hidden 크기 : 5 (출력으로 나오는 단어 크기)
- Batch : 1 (사용하는 입력 data 갯수)



데이터 만들기

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # reproducibility

idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello
```



개요파악 데이터 만들기

num_classes = 5 # hi hello 에서 고유 단어 갯수

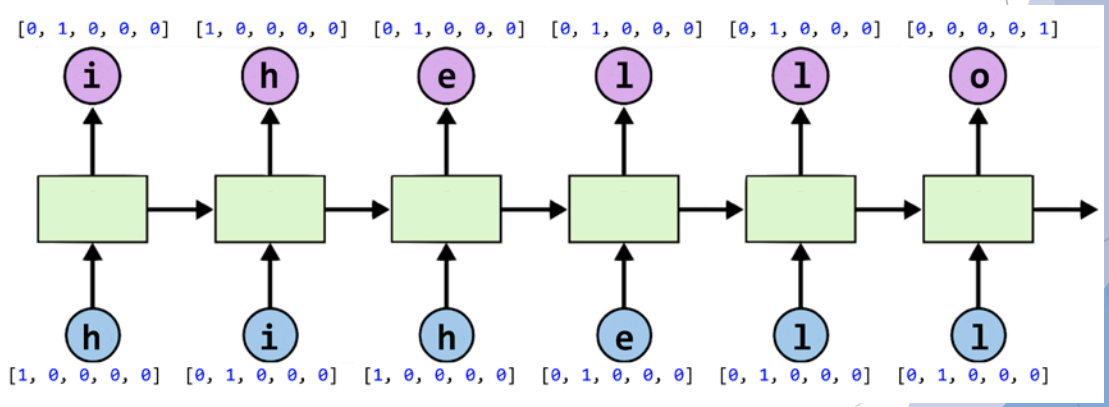
input_dim = 5 # one-hot size

hidden_size = 5 # output from the LSTM. 5 to directly predict one-hot

batch_size = 1   # one sentence

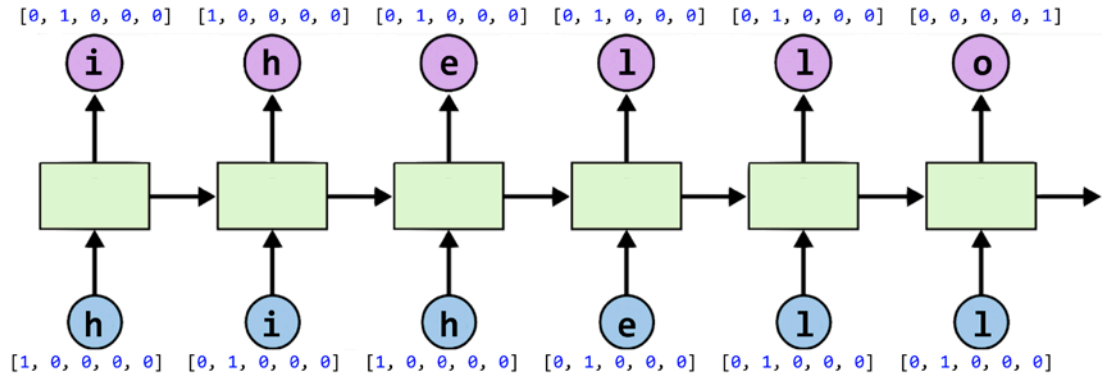
sequence_length = 6 # |ihello| == 6

learning_rate = 0.1



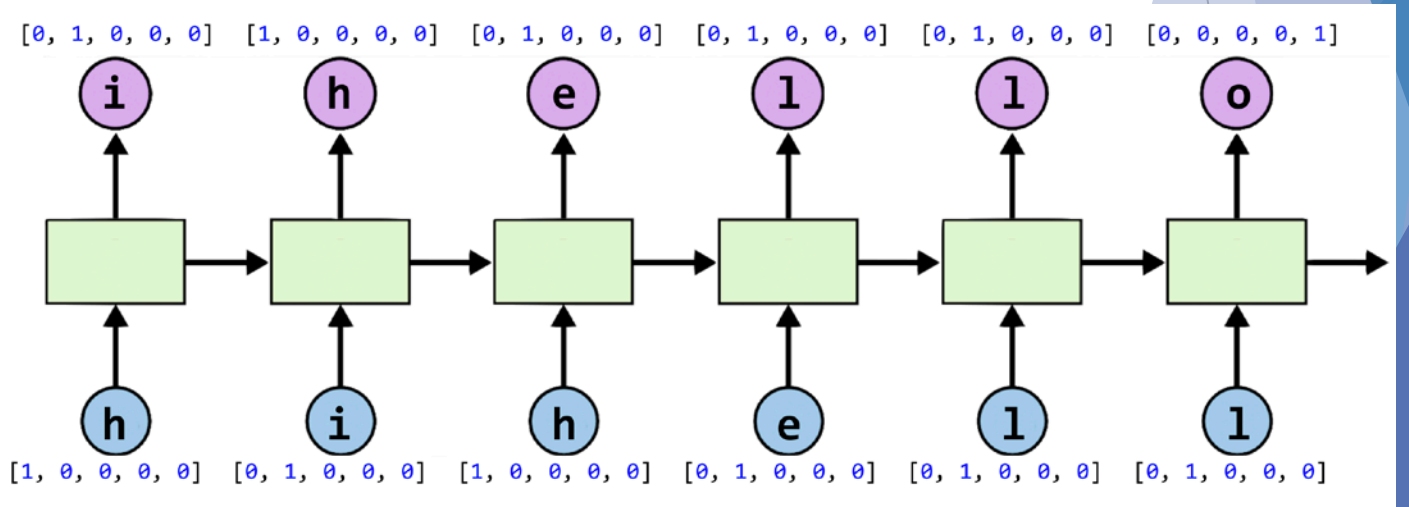
입, 출력 변수 만들기

```
X = tf.placeholder(
    tf.float32, [None, sequence_length, input_dim])  # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label
```



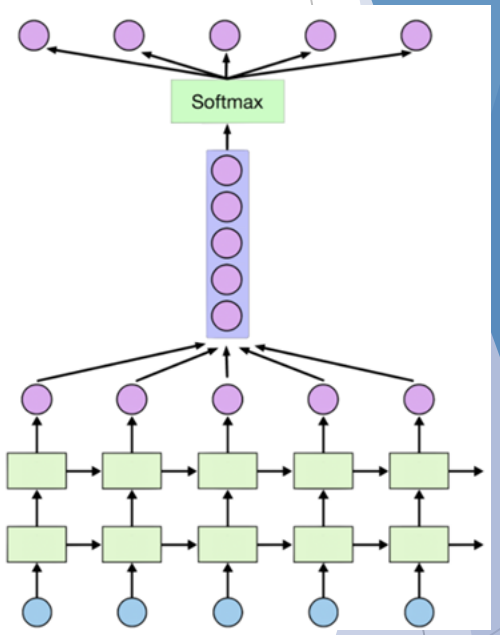
RNN 로직 만들기(Cell, state, output)

```
cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)
#LSTM의 cell 형태를 사용. State_is_tuple은 곧 없어질꺼라는데 return의 형태를 결정짓는 듯
# https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMCell 참조
initial_state = cell.zero_state(batch_size, tf.float32)
# 처음 상태값 0으로 설정
outputs, _states = tf.nn.dynamic_rnn(cell, X, initial_state=initial_state, dtype=tf.float32)
#dynamic_rnn으로 학습.
```



Output 부분을 완전 연결로 만들기

```
# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])


# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
# fc_b = tf.get_variable("fc_b", [num_classes])
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b


outputs = tf.contrib.layers.fully_connected(
    inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)
#참고 : https://www.tensorflow.org/api_docs/python/tf/contrib/layers/fully_connected
# num_outputs : 출력되는 character 수
# activation_fn : default 는 Relu, none 이면 skip됨.
```



Train을 위한 최적화함수 만들기

```
# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

prediction = tf.argmax(outputs, axis=2)
```

훈련 및 결과 출력

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(50):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print("\tPrediction str: ", ''.join(result_str))
```

```
WARNING:tensorflow:From <ipython-input-1-c805804c4287>:28: BasicLSTMCell.__init__ (from tensorflow.python.ops.rnn_cel
l_impl) is deprecated and will be removed in a future version.
Instructions for updating:
This class is deprecated, please use tf.nn.rnn_cell.LSTMCell, which supports all the feature this cell currently has.
Please replace the existing code with tf.nn.rnn_cell.LSTMCell(name='basic_lstm_cell').
0 loss: 1.6078763 prediction:  [[3 3 3 3 3 3]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  llllll
1 loss: 1.5102
        Predic
2 loss: 1.4327
        Predic
3 loss: 1.3489
        Predic
4 loss: 1.2551
        Predic
5 loss: 1.1404
        Predic
6 loss: 1.0167
        Predic
7 loss: 0.8969
        Predic
8 loss: 0.7695
        Predic
9 loss: 0.6550
        Predic
10 loss: 0.542
        Predic
11 loss: 0.428
        Predic
12 loss: 0.334
        Predic
13 loss: 0.247
        Predic
14 loss: 0.181
        Predic
15 loss: 0.132
        Predic
31 loss: 0.003641087 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
32 loss: 0.0032532993 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
33 loss: 0.0029363232 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
34 loss: 0.0026745955 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
35 loss: 0.002456025 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
36 loss: 0.0022713586 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
37 loss: 0.002113483 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
38 loss: 0.0019770935 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
39 loss: 0.0018580654 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
40 loss: 0.0017534181 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
41 loss: 0.0016606627 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
42 loss: 0.0015781594 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
43 loss: 0.0015043863 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
44 loss: 0.0014383355 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
45 loss: 0.001378979 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
46 loss: 0.0013257231 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
47 loss: 0.0012777768 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
48 loss: 0.0012345461 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
49 loss: 0.0011956157 prediction:  [[1 0 2 3 3 4]] true Y:  [[1, 0, 2, 3, 3, 4]]
        Prediction str:  ihello
```

최종 코드모음

```
import tensorflow as tf
import numpy as np
tf.set_random_seed(777)  # reproducibility

idx2char = ['h', 'i', 'e', 'l', 'o']
# Teach hello: hihell -> ihello
x_data = [[0, 1, 0, 2, 3, 3]]   # hihell
x_one_hot = [[[1, 0, 0, 0, 0],   # h 0
              [0, 1, 0, 0, 0],   # i 1
              [1, 0, 0, 0, 0],   # h 0
              [0, 0, 1, 0, 0],   # e 2
              [0, 0, 0, 1, 0],   # l 3
              [0, 0, 0, 1, 0]]]  # l 3

y_data = [[1, 0, 2, 3, 3, 4]]    # ihello
```

```
num_classes = 5
input_dim = 5  # one-hot size
hidden_size = 5  # output from the LSTM. 5 to directly predict one-hot
batch_size = 1   # one sentence
sequence_length = 6  # |ihello| == 6
learning_rate = 0.1

X = tf.placeholder(
    tf.float32, [None, sequence_length, input_dim])  # X one-hot
Y = tf.placeholder(tf.int32, [None, sequence_length])  # Y label

cell = tf.contrib.rnn.BasicLSTMCell(num_units=hidden_size, state_is_tuple=True)
initial_state = cell.zero_state(batch_size, tf.float32)
outputs, _states = tf.nn.dynamic_rnn(
    cell, X, initial_state=initial_state, dtype=tf.float32)
```

```
# FC layer
X_for_fc = tf.reshape(outputs, [-1, hidden_size])
# fc_w = tf.get_variable("fc_w", [hidden_size, num_classes])
# fc_b = tf.get_variable("fc_b", [num_classes])
# outputs = tf.matmul(X_for_fc, fc_w) + fc_b
outputs = tf.contrib.layers.fully_connected(
    inputs=X_for_fc, num_outputs=num_classes, activation_fn=None)

# reshape out for sequence_loss
outputs = tf.reshape(outputs, [batch_size, sequence_length, num_classes])

weights = tf.ones([batch_size, sequence_length])
sequence_loss = tf.contrib.seq2seq.sequence_loss(
    logits=outputs, targets=Y, weights=weights)
loss = tf.reduce_mean(sequence_loss)
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

prediction = tf.argmax(outputs, axis=2)
```

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(50):
        l, _ = sess.run([loss, train], feed_dict={X: x_one_hot, Y: y_data})
        result = sess.run(prediction, feed_dict={X: x_one_hot})
        print(i, "loss:", l, "prediction: ", result, "true Y: ", y_data)

        # print char using dic
        result_str = [idx2char[c] for c in np.squeeze(result)]
        print("\tPrediction str: ", ''.join(result_str))
```

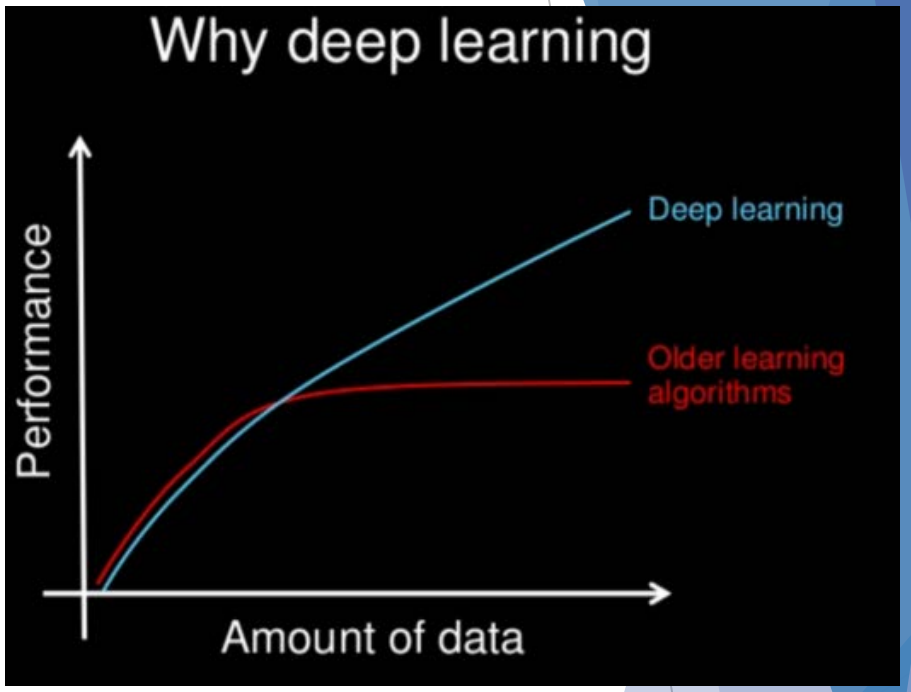
어떻게 써야하는가?

딥러닝이 항상 답은 아니다.

- 기존의 방법이 충분히 좋을 수 있다.
- 기존의 방법으로 하다가 안돼면 돌아오는것이 낫다.

- Ex) 장비병
  - 내가 아는 지식에 빠져서 실질적인 핵심을 놓치고 나의 전문성만을 발휘하려는 습성

딥러닝을 쓰면 좋을 때



- 1. 데이터가 매우 풍부할때
  - 딥러닝은 데이터의 양이 많아야 성능이 나온다.
  - 과제를 이해하는데 매우 큰 데이터가 필요하다.
  - 데이터가 적으면, 수작업이 들어간 머신러닝이 좋다.
- 2. 하드웨어가 좋을때
  - 딥러닝은 고사양 하드웨어가 많은 부분을 차지한다.
  - 많은 양의 행렬을 계산하기 때문에 gpu를 통해 계산하는 것이 좋다.
  - 최근에는 cpu로도 가능하지만 속도가 느리다.
- 3. 내가 하던 방법이 잘 안될때.

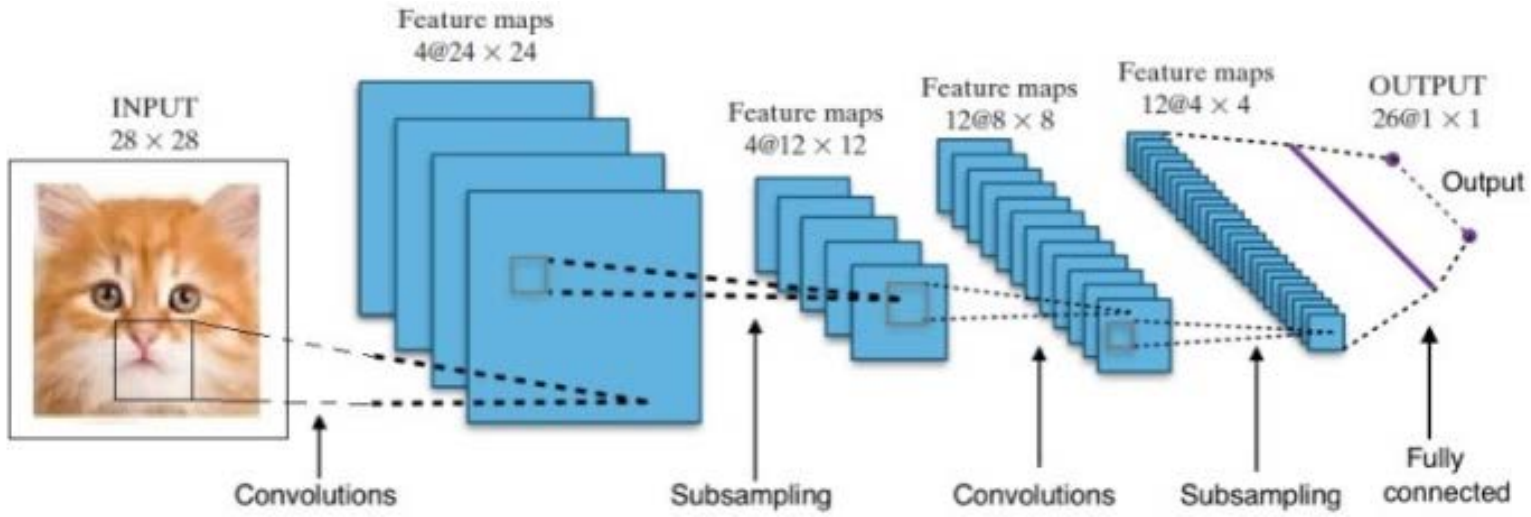
데이터 싸이언스는 디버깅이다.

- 프로그래밍보단 자유도가 적다.
- 처음부터 모델링 하고 세션 실행 하면 원하는 만큼 되지 않는다.
- 최적화를 시키기 위한 과정이 훨씬 오래 필요하다.

딥러닝 이용 사례

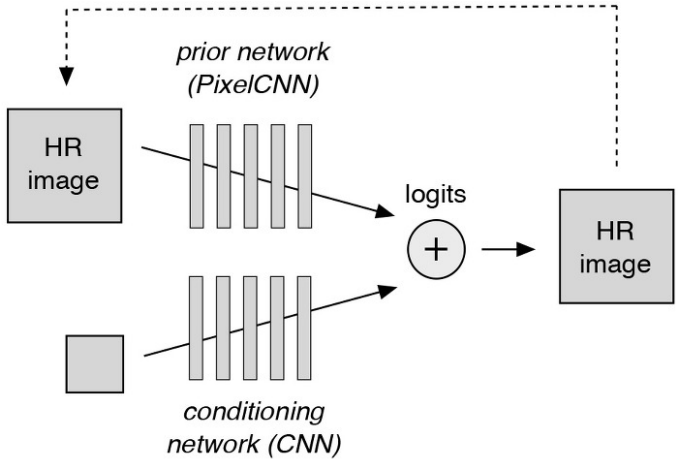
어떤것이 쓰였는지.

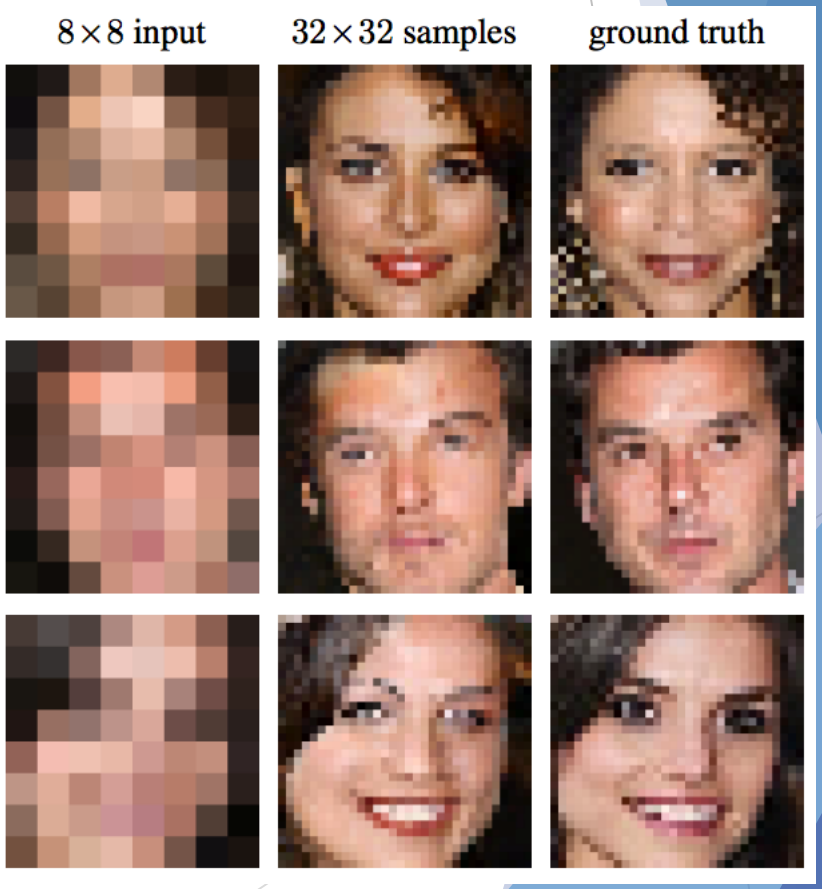
1. 이미지& 영상 인식 - CNN



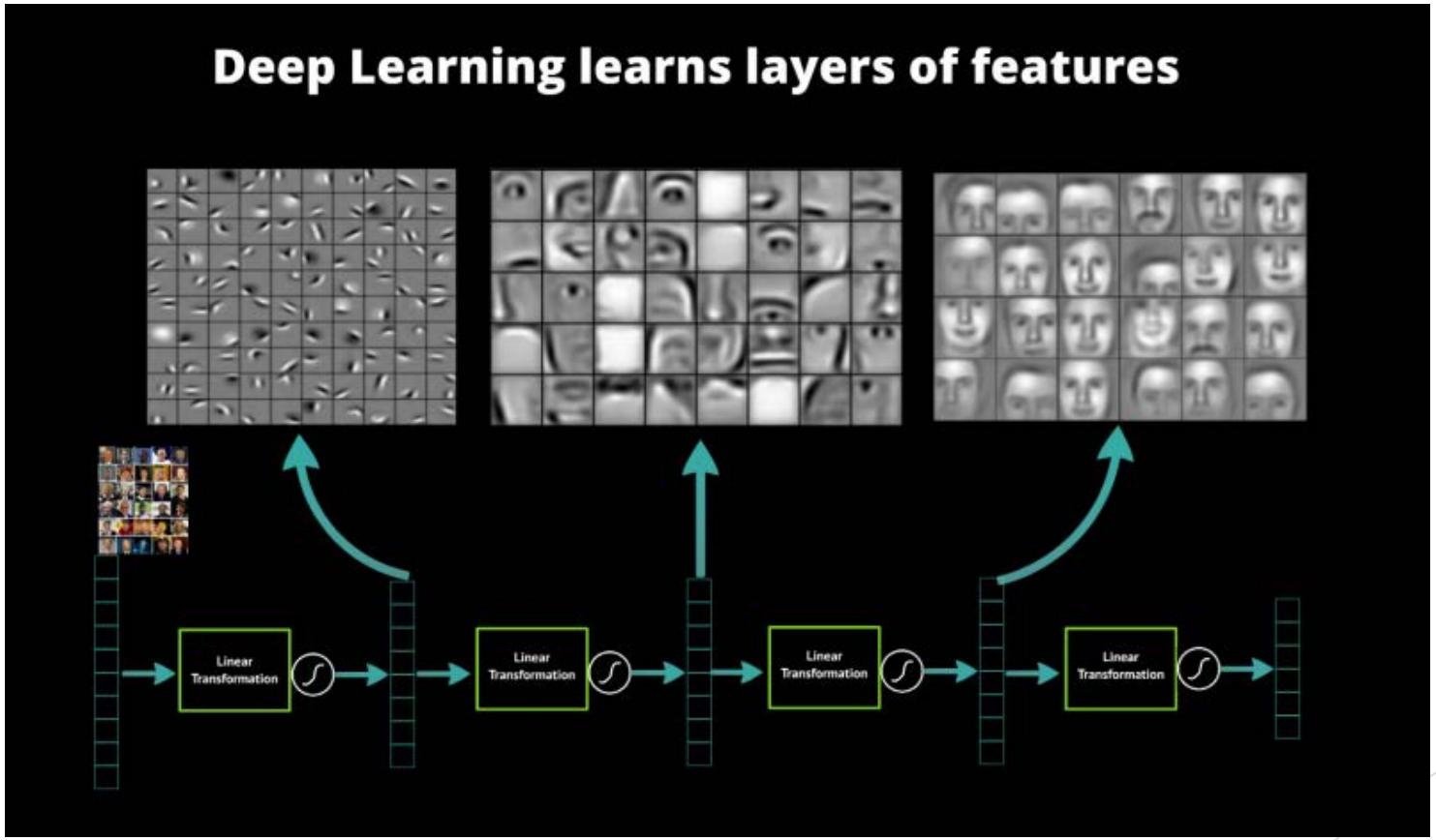
2. 패턴 인식 – 흑색 컬러화, 픽셀복원

- 흑백사진 컬러화
  - https://www.youtube.com/watch?v=ys5nMO4Q0iY
  - Automatic image colorization
- 저화질의 사진 픽셀 복원하기
  - https://www.youtube.com/watch?v=3uoM5kfZIQ0
  - **Pixel Recursive Super Resolution**



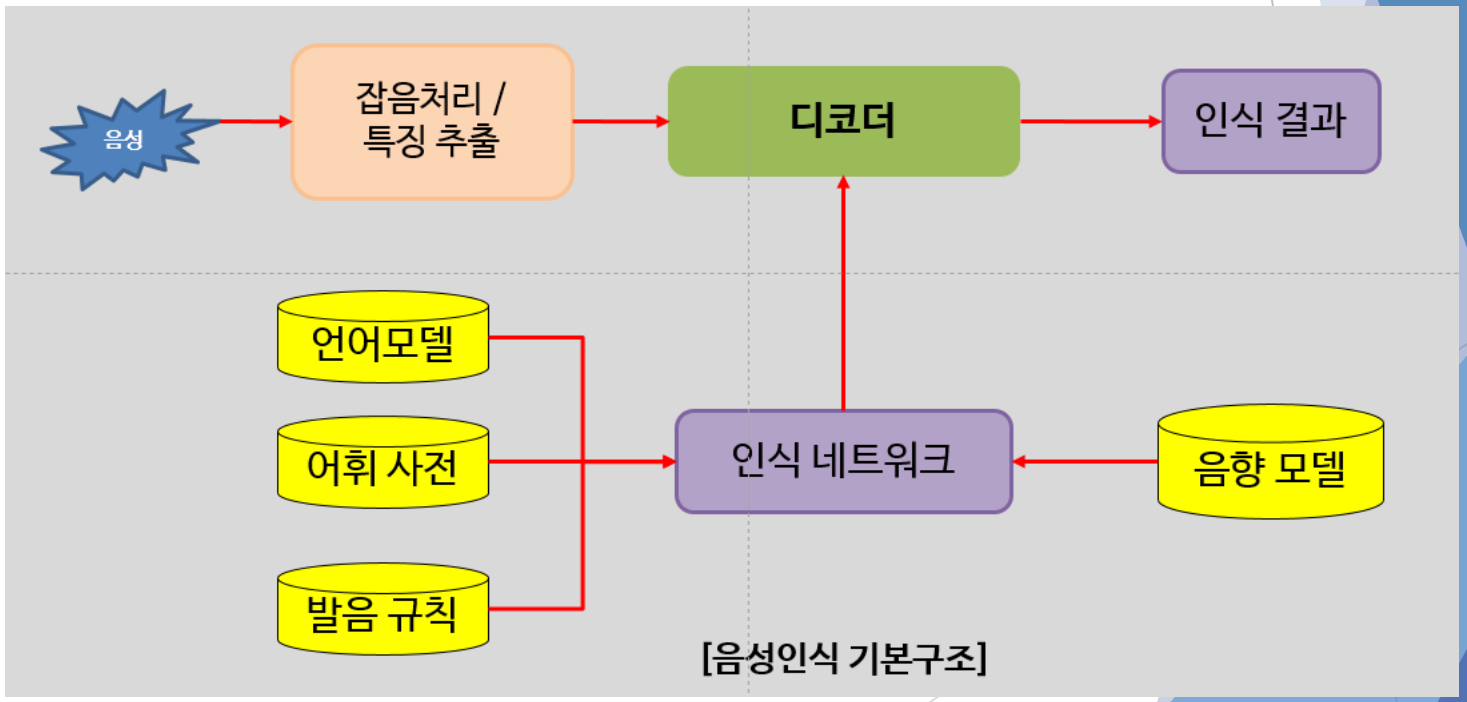


3. 특징점 추출


Deep Learning learns layers of features
Linear Transformation
Linear Transformation
Linear Transformation
Linear Transformation

4. 음성인식 & 음악 자동분류 - 태깅

- 음고, 음량, 음가, 음색, 화성의 다섯 가지 기본 요소를 사용
- http://s-space.snu.ac.kr/handle/10371/142424



[음성인식 기본구조]

5. 자연어처리(NLP) – 챗봇, 사기 판별

- Word2Vec, RNN, LSTM 이용.
- 주요 글로벌 IT 회사에서 챗봇을 위한 프레임 워크를 개발해 놓기도 함.
- 맥락, 기억등을 이용.

- 이상, 사기 의도 판별 – 패턴 학습으로 정확도 향상
  - http://intothedata.com/02.scholar_category/anomaly_detection/