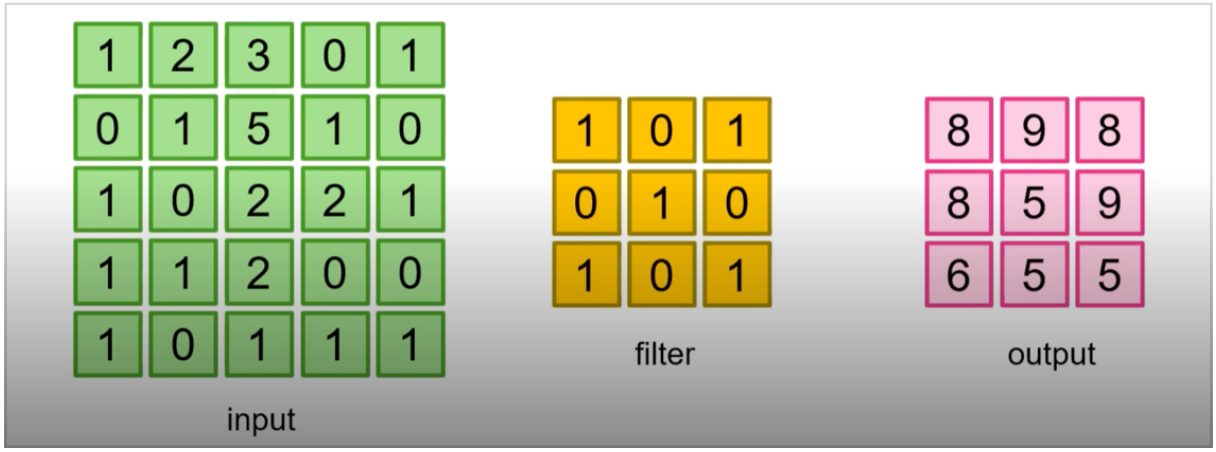
#스터디 #CNN #RNN #머신러닝#

CNN(합성곱신경망)

합성곱신경망 네트워크

- #CNN/Convoultion 텐서의 합성곱 연산 : 필터를 이미지 위에서 stride 값만큼 이동시키면서 겹쳐지는 부분의 각 원소의 값을 곱한것을 모두 더하여 output을 출력하는 연산



(Stride 가 1이 거나 2 일수 있다)

- #CNN/Padding (zero-padding) : 0으로 된 패드가 input의 주위를 둘러쌈
  - 파이토치로 합성곱 연산 구현하기 -

(필터_생성)

커널 사이즈는 따로 지정하지 않으면 정방행렬로나옴 (3 –> 3x3 matrix)

input 의 형태는 Tensor 로 되어있음

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolution의 output 크기

$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

(Output 텐서의 크기)

- #CNN/Cross-correlation

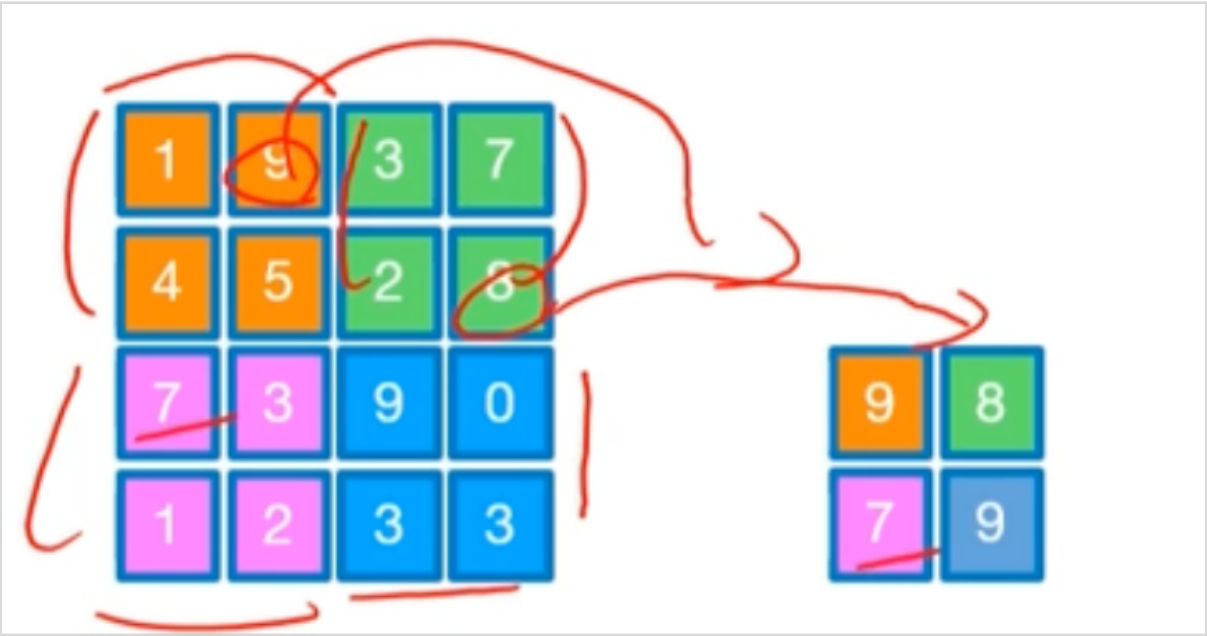
$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.$$

- 뒤집고 계산하면 => (Convolution)
- 안 뒤집고 계산하면 => (Cross-Correlation)

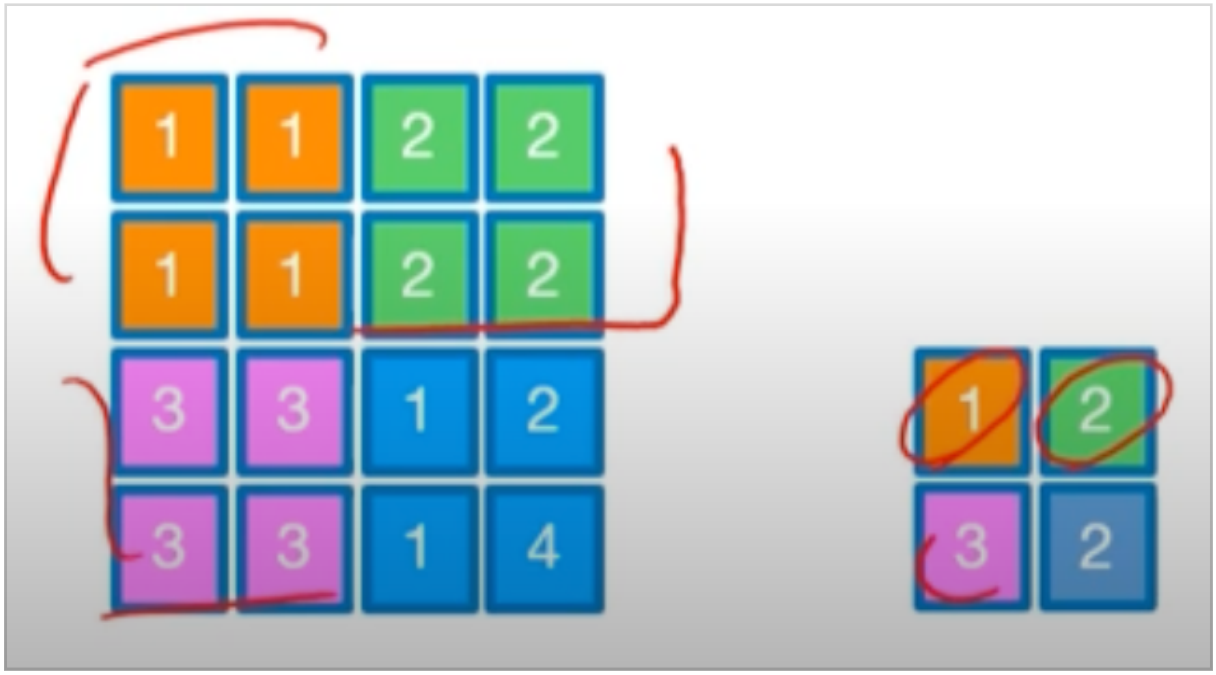
- #CNN/Pooling : 이미지 사이즈 조절 / 연산 대체
  - #CNN/Pooling/MaxPooling

```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)
```

MaxPool2d



  - #CNN/Pooling/AveragePooling



- #CNN/MnistCNN

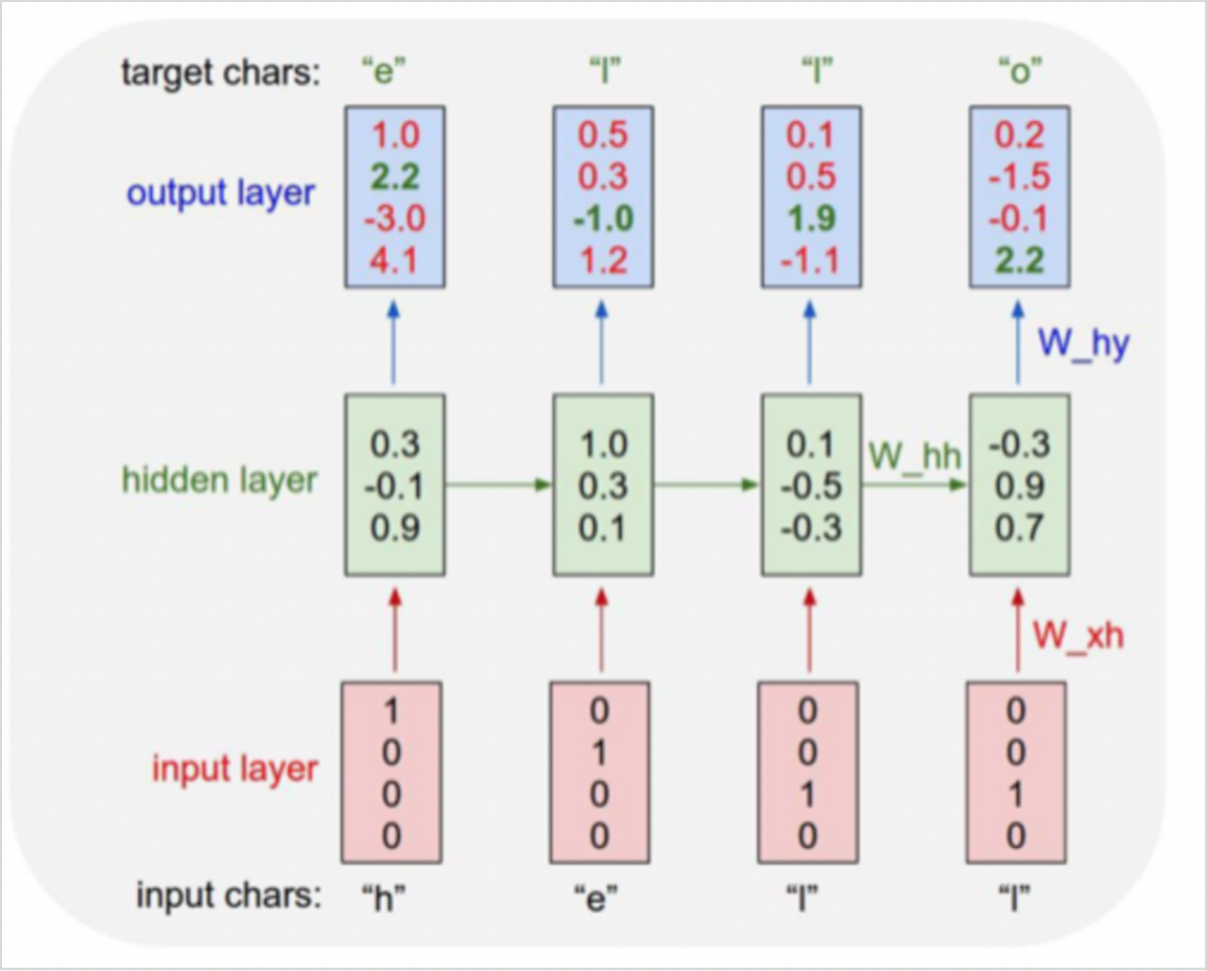
RNN(순환신경망)

순환신경망이란?

- 시간의 흐름에 따라 변화는 데이터들을 학습하는 인공신경망
- 활성함수를 지난 값이 출력층으로만 흐르는 신경망을 피드포워드(feed forward) 방식의 신경망이라고 하는데 RNN은 그렇지 않음
  - 이전 단계에서의 출력이 다음 단계의 입력에 영향을 준다
  - 단어완성 알고리즘 등에 사용됨

    ex) “Hello” 를 입력하려고 하는데 h,e,l 이 순서대로 입력되어있으면

    다음번에 올 문자는 ‘l’ 일 확률이 크다




```
# declare RNN
rnn = torch.nn.RNN(input_size, hidden_size)
# check output
outputs, _status = rnn(input_data)
print(outputs)
print(outputs.size())
```

- Shape = (Batch size , Sequence Length , Input Size)

  Pytorch 는 입력된 데이터를 보고 자동적으로 shape 의 형태를 계산할 수 있음
  - Input size : 입력하는 벡터의 차원(사이즈)
  - Hidden size : 출력 및 다음 번 입력에 전달되는 값,출력되는 벡터의 차원과 같음
  - Sequence Length : Sequence 의 개수 ('hello' 를 입력하면 sequence 는 5의 length) 를 가짐
  - Batch size : 한번에 학습시킬 sequence 의 개수

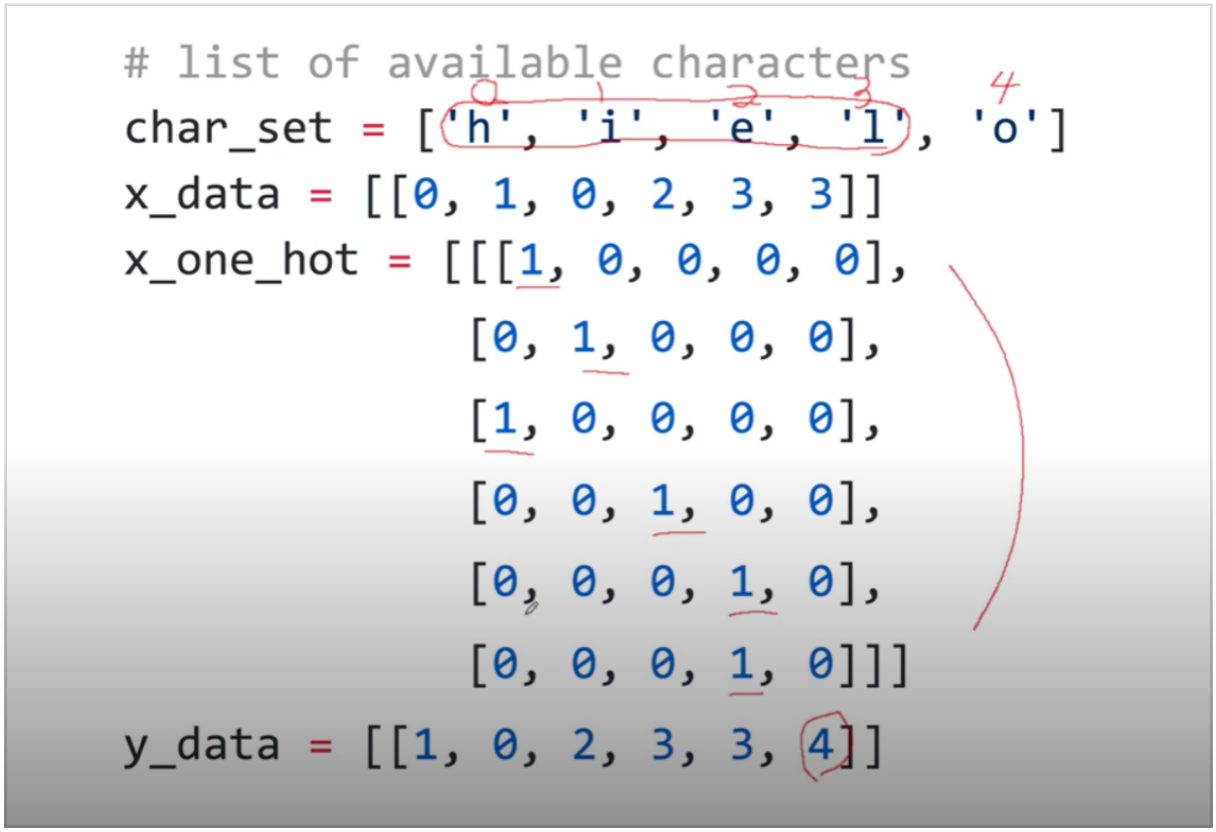
```
# declare dimension
input_size = 4
hidden_size = 2
```

```
# singleton example
# shape : (1, 1, 4)
# input_data_np = np.array([[[1, 0, 0, 0]]])

# sequential example
# shape : (3, 5, 4)
h = [1, 0, 0, 0]
e = [0, 1, 0, 0]
l = [0, 0, 1, 0]
o = [0, 0, 0, 1]
input_data_np = np.array([[h, e, l, l, o], [e, o, l, l, l], [l, l, e, e, l]], dtype=np.float32)
```

```
# check output
outputs, _status = rnn(input_data)
print(outputs)
print(outputs.size())
```

```
torch.Size([3, 5, 2])
```

- #RNN/One-hot_encoding
  - 벡터의 하나의 축만을 1, 나머지를 0으로 두어 문자열을 코딩함



- #RNN/Hihello
  - #RNN/CEL : 분류문제의 정확도를 향상시키는데 사용하는 손실함수

```
# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(rnn.parameters(), learning_rate)
loss = criterion(outputs.view(-1, dic_size), Y.view(-1))
# 모델의 output 과 정답 label 을 입력함
```

```
char_set = ['h', 'i', 'e', 'l', 'o']
# hyper parameters
input_size = len(char_set)
hidden_size = len(char_set)
learning_rate = 0.1
# data setting
x_data = [[0, 1, 0, 2, 3, 3]]
x_one_hot = [[[1, 0, 0, 0, 0],
              [0, 1, 0, 0, 0],
              [1, 0, 0, 0, 0],
              [0, 0, 1, 0, 0],
              [0, 0, 0, 1, 0],
              [0, 0, 0, 1, 0]]]
y_data = [[1, 0, 2, 3, 3, 4]]
```

```
# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)
```

- #RNN/charseq : hihello 예제를 일반화

```
sample = " if you want you"
# make dictionary
char_set = list(set(sample))
char_dic = {c: i for i, c in enumerate(char_set)}
  # 특정 문자의 인덱스를 자동으로 찾아주는 사전을 선언함
```

```
print(char_dic)

# hyper parameters
dic_size = len(char_dic)
hidden_size = len(char_dic)
learning_rate = 0.1

# data setting
sample_idx = [char_dic[c] for c in sample]
x_data = [sample_idx[:-1]] # 맨 마지막 문자를 제거함
x_one_hot = [np.eye(dic_size)[x] for x in x_data]
  # np.eye : Identity Matrix 를 생성하는 옵션
y_data = [sample_idx[1:]]  # 맨 처음 문자를 제거함

# transform as torch tensor variable
X = torch.FloatTensor(x_one_hot)
Y = torch.LongTensor(y_data)

# declare RNN
rnn = torch.nn.RNN(dic_size, hidden_size, batch_first=True) #output shape 의 첫
번째가 batch_size

# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(rnn.parameters(), learning_rate)

# start training
for i in range(50):
    optimizer.zero_grad()
    outputs, _status = rnn(X)
    loss = criterion(outputs.view(-1, dic_size), Y.view(-1))
    loss.backward()
    optimizer.step()

    result = outputs.data.numpy().argmax(axis=2)
    result_str = ''.join([char_set[c] for c in np.squeeze(result)])
    print(i, "loss: ", loss.item(), "prediction: ", result, "true Y: ", y_data,
"prediction str: ", result_str)
```

- #RNN/longseq

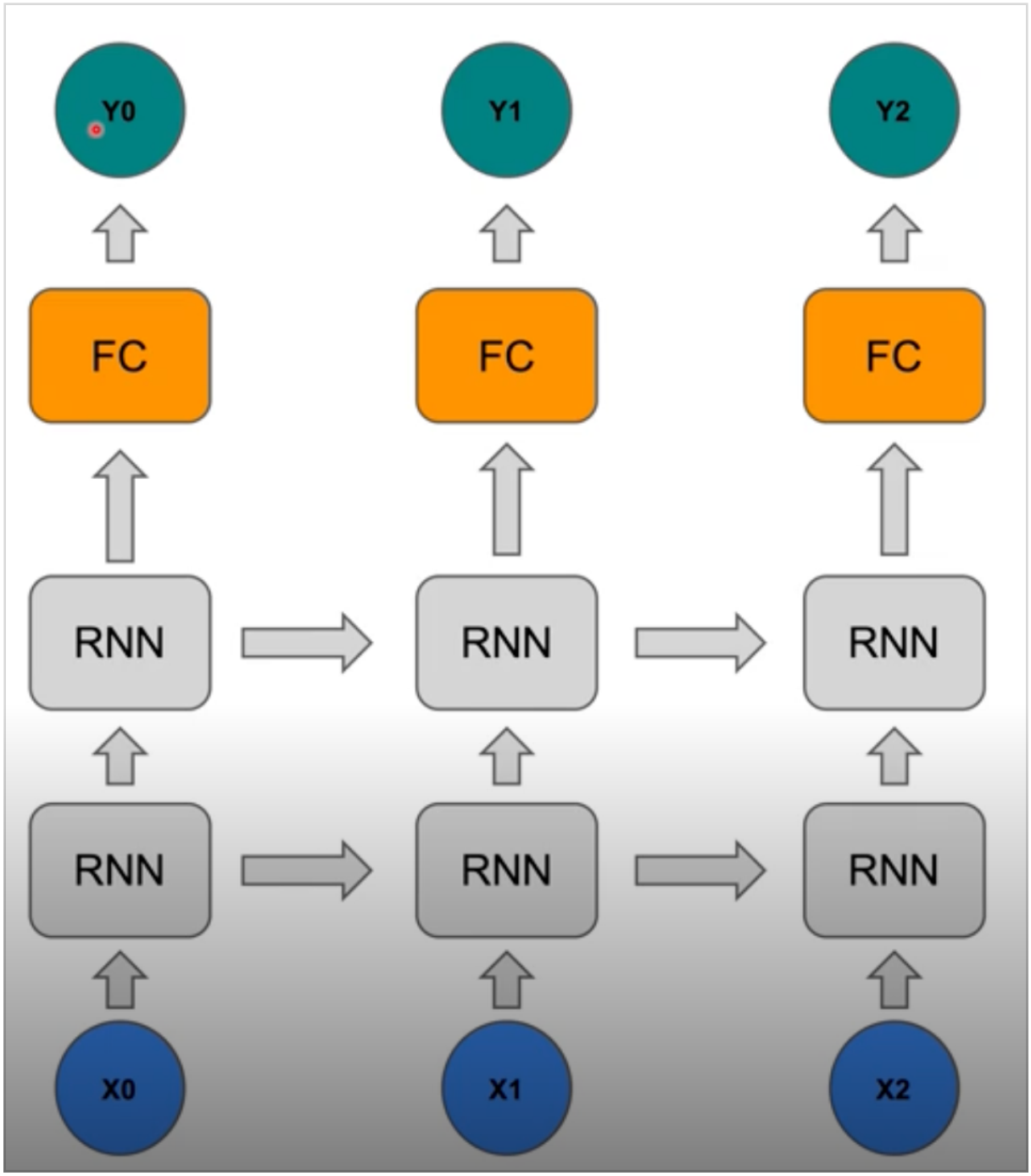
```
sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")
```

(긴 문장을 fixed size 된 sequence로 자르는 게 먼저이다)

- [ ] 합성곱 필터처럼 문장을 공백 포함 특정 크기만큼 선택하고 이동시켜서 sequence 들을 만들자!

```
sentence = ("if you want to build a ship, don't drum up people together to "
            "collect wood and don't assign them tasks and work, but rather "
            "teach them to long for the endless immensity of the sea.")

# hyper parameters
dic_size = len(char_dic)
hidden_size = len(char_dic)
sequence_length = 10  # Any arbitrary number
learning_rate = 0.1

# data setting
x_data = []
y_data = []

for i in range(0, len(sentence) - sequence_length):
    x_str = sentence[i:i + sequence_length]
    y_str = sentence[i + 1: i + sequence_length + 1]
    print(i, x_str, '->', y_str)

    x_data.append([char_dic[c] for c in x_str])  # x str to index
    y_data.append([char_dic[c] for c in y_str])  # y str to index

x_one_hot = [np.eye(dic_size)[x] for x in x_data]
```

- #RNN/FClayer




```
# declare RNN + FC
class Net(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, layers):
        super(Net, self).__init__()
        self.rnn = torch.nn.RNN(input_dim, hidden_dim, num_layers=layers,
batch_first=True)
        self.fc = torch.nn.Linear(hidden_dim, hidden_dim, bias=True)
#Net 모듈이 RNN과 FC layer, 그리고 Linear 모듈을 이용함
    def forward(self, x):
        x, _status = self.rnn(x)
```

```
        x = self.fc(x)
        return x

net = Net(dic_size, hidden_size, 2)

# loss & optimizer setting
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), learning_rate)

# start training
for i in range(100):
    optimizer.zero_grad()
    outputs = net(X)
    loss = criterion(outputs.view(-1, dic_size), Y.view(-1))
    loss.backward()
    optimizer.step()

    results = outputs.argmax(dim=2)
    predict_str = ""
    for j, result in enumerate(results):
        # print(i, j, ''.join([char_set[t] for t in result]), loss.item())
        if j == 0:
            predict_str += ''.join([char_set[t] for t in result])
        else:
            predict_str += char_set[result[-1]]

    print(predict_str)
```