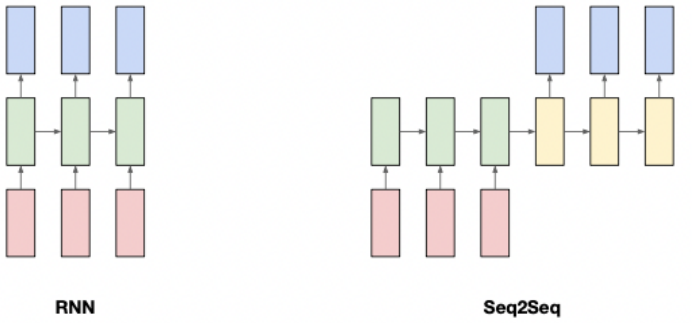
RNN seq2seq

상태	RNN
담당자	

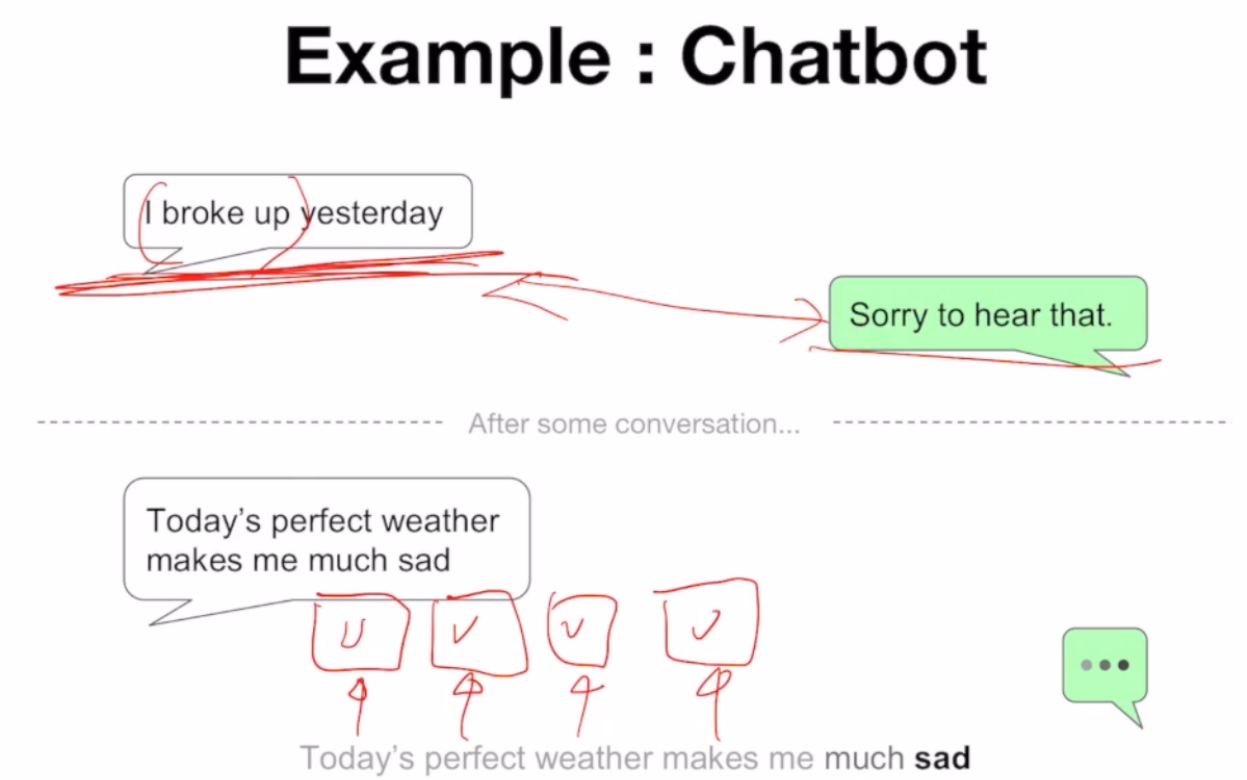
Seq2seq이란?

💡 sequence를 입력받아 sequence를 출력하는 모델로, 기계 번역이나 챗봇에 많이 사용된다.

RNN과 seq2seq은 무엇이 다른가?



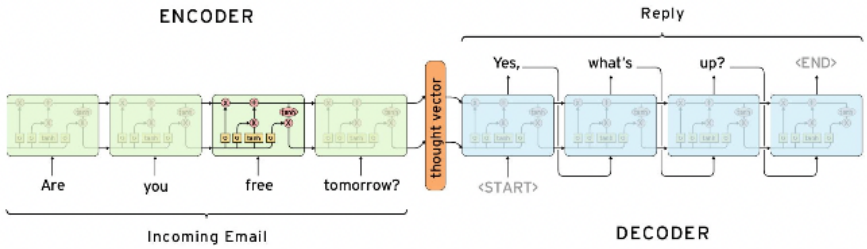
> I broke up yesterday → Today's perfect weather makes me much sad



RNN 기반 모델로 만든 챗봇은 적절한 대답을 해줄 수 있을까? RNN은 문장을 구성하는 단어가 입력되면, 그때마다 단어를 출력해 답변을 만든다. 문장을 다 듣기도 전에 답변을 만드는 셈! 따라서 'Today's perfect weather' 까지 들은 모델은 뒤에 등장하는 'sad'와는 거리가 먼 답변을 할 것이다.

그래서 '문장을 끝까지 다 듣고 말하기' 라는 맥락에서 seq2seq의 기본 아이디어가 등장했다.

Encoder-decoder 구조



- Encoder

  : 입력된 단어들의 sequence가 나오며, 이를 어떤 벡터로 압축해 decoder에 전달해 준다.

- Decoder

  : 그 벡터를 첫 셀의 hidden state로 넣어주고, 문장이 시작한다는 start flag와 함께 모델을 시작한다. 셀에서 나온 output을 문장의 첫번째 단어로 두고,

  이 단어가 다시 두 번째 셀의 입력에 들어가, 이전 셀의 hidden state와 함께 다음 단어를 예측한다.

Seq2seq 적용

번역을 위한 seq2seq 코드의 예시를 보면서 전체적인 흐름을 확인해 보자.

- raw : 전체 데이터 셋
- Source Text : 전체 데이터 셋 중 영어 문장
- Target Text : 전체 데이터 셋 중 번역한 한국어 문장
- preprocess : 전체 데이터 셋을 Train과 Test로 나누고, 각각의 단어와 단어의 빈도를 체크한다.
- SOURCE_MAX_LENGTH : Source Text의 최대 길이 제한
- TARGET_MAX_LENGTH : Target Text의 최대 길이 제한

```
SOURCE_MAX_LENGTH = 10
TARGET_MAX_LENGTH = 12
load_pairs, load_source_vocab, load_target_vocab = preprocess(raw, SOURCE_MAX_LENGTH, TARGET_MAX_LENGTH)
print(random.choice(load_pairs))

# Hidden state 정의
enc_hidden_size = 16 # Encoder의 Hidden state 정의
dec_hidden_size = enc_hidden_size # Decoder의 Hidden state 정의 (Encoder와 같다)

# Encoder - Decoder
enc = Encoder(load_source_vocab.n_vocab, enc_hidden_size).to(device) # RNN의 레이어인 Encoder 선언
dec = Decoder(dec_hidden_size, load_traget_vocab.n_vocab).to(device) # RNN의 레이어인 Decoder 선언

# 학습
train(load_pairs, load_source_vocab, load_target_vocab, enc, dec, 5000, print_every=1000) # Encoder의 출력을 Decoder의 입력으로 연결하는 부분 존재

# 모델 평가
evaluate(load_pairs, load_source_vocab, load_target_vocab, enc, dec, TARGET_MAX_LENGTH)
```

Neural net setting

```
# declare simple encoder
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Encoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, x, hidden):
        x = self.embedding(x).view(1, 1, -1)
        x, hidden = self.gru(x, hidden)
        return x, hidden
```

```
# declare simple decoder
class Decoder(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(Decoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x, hidden):
        x = self.embedding(x).view(1, 1, -1)
        x, hidden = self.gru(x, hidden)
        x = self.softmax(self.out(x[0]))
        return x, hidden
```

학습

```
# convert sentence to the index tensor
def tensorize(vocab, sentence):
    indexes = [vocab.vocab2index[word] for word in sentence.split(" ")]
    indexes.append(vocab.vocab2index["<EOS>"])
    return torch.Tensor(indexes).long().to(device).view(-1, 1)
```

문장 → 원핫 벡터 → 파이토치 텐서화 (`tensorize`)

```
# training seq2seq
def train(pairs, source_vocab, target_vocab, encoder, decoder, n_iter, print_every=1000, learning_rate=0.01):
    loss_total = 0

    encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)

    training_batch = [random.choice(pairs) for _ in range(n_iter)] # 전체 데이터 중 랜덤하게 학습 데이터 추출
    training_source = [tensorize(source_vocab, pair[0]) for pair in training_batch] # soucre
    training_target = [tensorize(target_vocab, pair[1]) for pair in training_batch] # target

    criterion = nn.NLLLoss()

    for i in range(1, n_iter + 1):
        source_tensor = training_source[i - 1]
        target_tensor = training_target[i - 1]

        encoder_hidden = torch.zeros([1, 1, encoder.hidden_size]).to(device)
        # 인코더의 첫 hidden state는 아무 값이 없기 때문에 영벡터로 만들어 넣어준다.

        encoder_optimizer.zero_grad()
        decoder_optimizer.zero_grad()

        source_length = source_tensor.size(0)
        target_length = target_tensor.size(0)

        loss = 0

        for enc_input in range(source_length):
            # 인코더의 hidden satate를 꺼내오자
            _, encoder_hidden = encoder(source_tensor[enc_input], encoder_hidden)

        decoder_input = torch.Tensor([[SOS_token]]).long().to(device)
        decoder_hidden = encoder_hidden # connect encoder output to decoder input

        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # teacher forcing

        loss.backward()

        encoder_optimizer.step()
        decoder_optimizer.step()

        loss_iter = loss.item() / target_length
        loss_total += loss_iter

        if i % print_every == 0:
            loss_avg = loss_total / print_every
            loss_total = 0
            print("[{} - {}%] loss = {:05.4f}".format(i, i / n_iter * 100, loss_avg))
```