Teacher Forcing

Winter Vacation Capstone Study

TEAM Kai.Lib

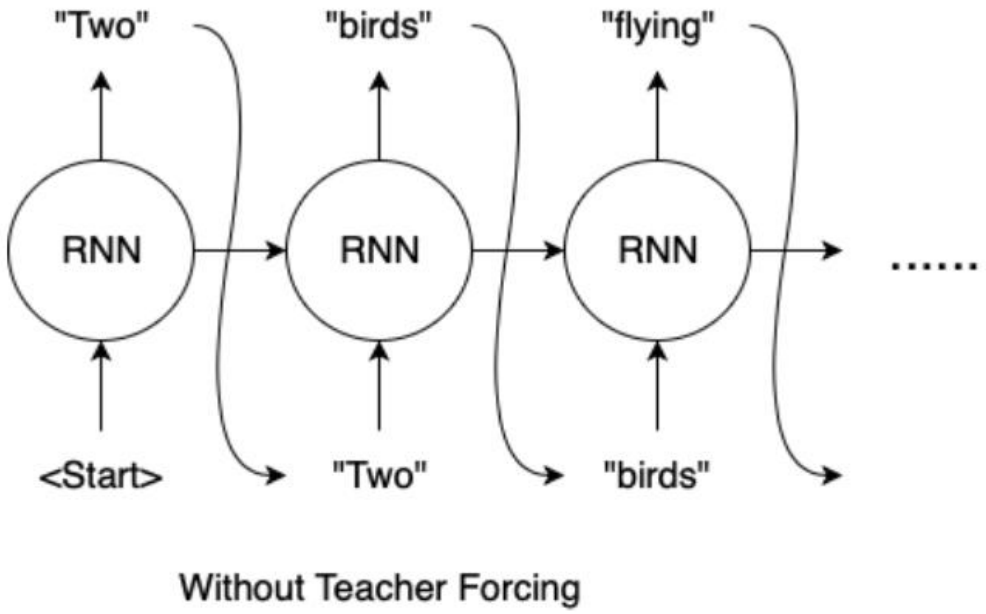
발표자 : 배세영

2020.01.06 (MON)

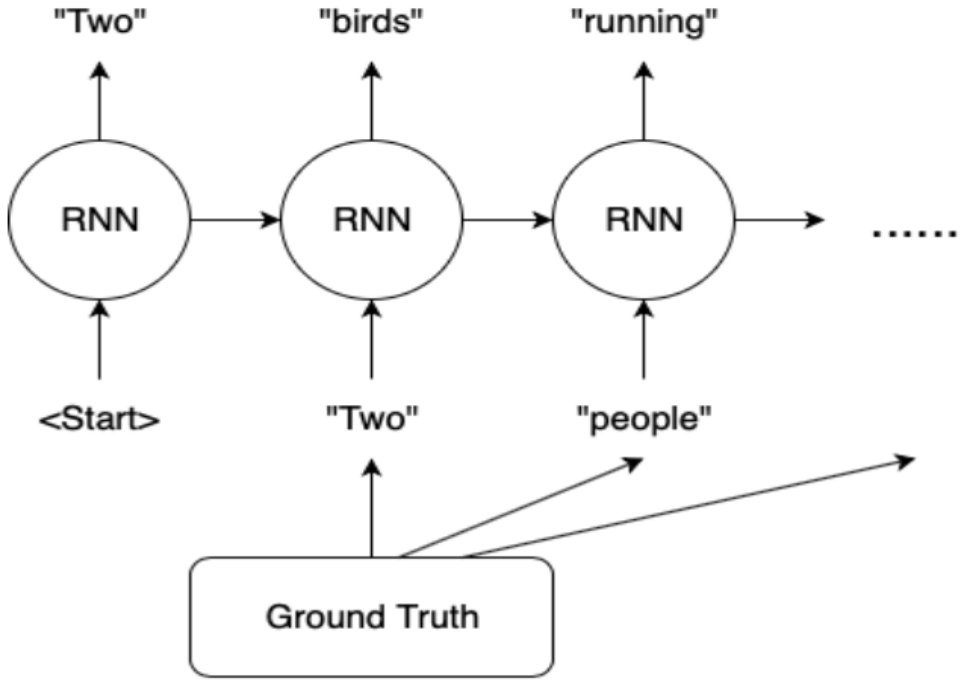
Teacher Forcing의 개요

- "***Teacher Forcing*** **is the technique where the *target word* is passed as the *next input* to the decoder."**

티쳐 포싱은 target word(Ground Truth)를 디코더의 다음 입력으로 넣어 주는 기법



Without Teacher Forcing



With Teacher Forcing

Teacher Forcing의 쉬운 비유

- **문제 A의 답이 문제 B의 계산에 필요하고, 문제 B의 답이 문제 C의 풀이에 이용되는 일련의 문제들에서**

  - **Teacher Forcing 미사용**

    학생은 문제 A, B, C를 순서대로 풀이하고 답 a, b, c를 한꺼번에 작성하여 제출
    교사는 이 답안지를 보고 a, b, c를 한꺼번에 채점하여 점수를 알려 줌

  - **Teacher Forcing 사용**

    학생은 문제 A를 풀이하고 답 a를 제출
    교사는 이 답안지를 보고 a를 채점한 후, 점수와 함께 정답 a′을 알려 줌
    학생은 문제 A의 정답 a′을 가지고 문제 B를 풀이하고 답 b를 제출
    .....

Teacher Forcing 기법의 장단점

- 장점
  - 학습이 빠르다

    학습 초기 단계에서는 모델의 예측 성능이 나쁘다.
    때문에 Tearcher Forcing을 이용하지 않으면 잘못된 예측 값을 토대로 hidden state값이 update되고,
    이 때문에 모델의 학습 속도를 더뎌지게 한다.

- 단점
  - 노출 편향 문제 (Exposure Bias Problem)

    추론 (Inference) 과정에서는 제공할 수 있는 Ground Truth가 없다.
    때문에 모델은 전 단계의 자기 자신의 출력값을 기반으로 다음 예측을 이어가야 한다.
    이러한 학습과 추론 단계에서의 차이 (discrepancy) 가 존재하여 모델의 성능과 안정성을 떨어뜨릴 수 있다.

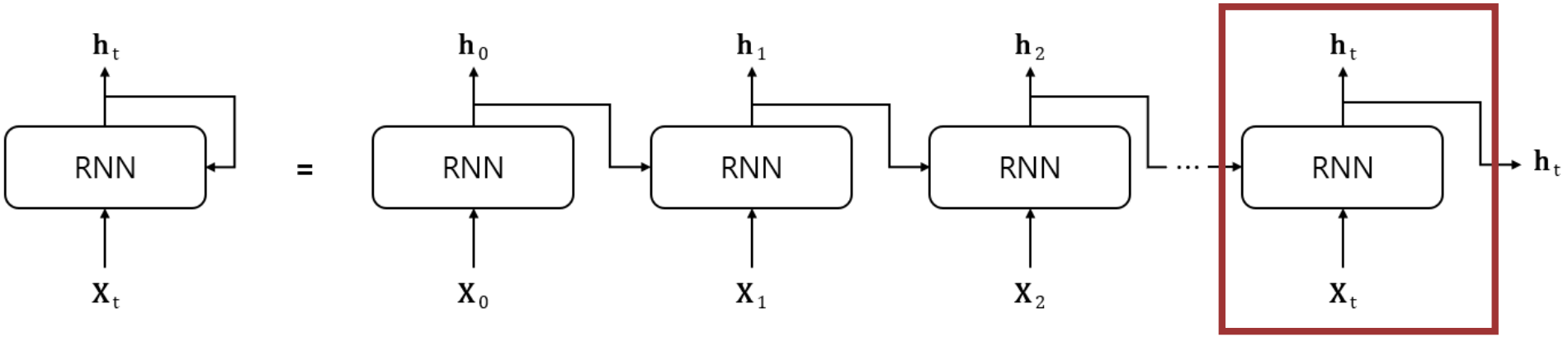
    다만 노출 편향 문제가 생각만큼 큰 영향을 미치지는 않는다는 연구 결과가 나와 있다고 한다.
    (T. He, J. Zhang, Z. Zhou, and J. Glass. Quantifying Exposure Bias for Neural Language Generation (2019), arXiv.)

Teacher Forcing FAQ

Q. Ground Truth를 모두 넘겨준다는 특징 때문에,
모델이 이 Ground Truth를 단순히 암기하여 추론하는 문제가 발생하지는 않는가?

A. 그런 문제는 일어나지 않는다.

시점 t에서 모델의 입력은 t-1 시점에서의 ground truth와, 시점 1에서 t-2까지의 참값을 바탕으로 update된 hidden state이다. 시점 t에서의 참값을 입력으로 주기 전이므로, 모델이 이를 단순히 암기하여 추론하는 경우는 발생할 수 없다.

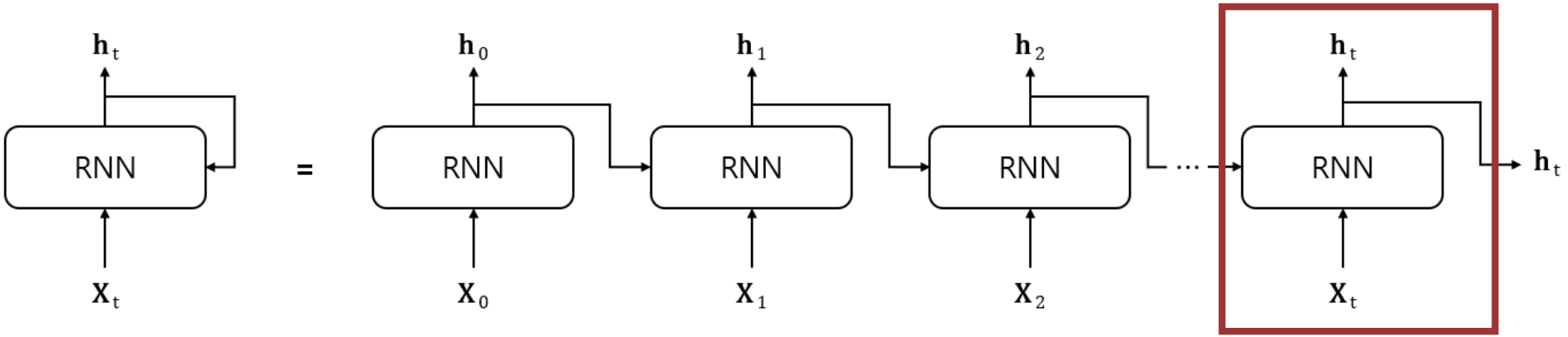


Teacher Forcing FAQ

Q. Teacher Forcing 기법이 자연어 처리 (Natural Language Processing) 이외의 분야에서도 활용되는가?

A. 그렇다.

타임시리즈 예측 (Time Serise Forecasting, 데이터의 시간적 패턴을 탐지하는 것) 등 NLP 이외의 분야에도 활용된다.

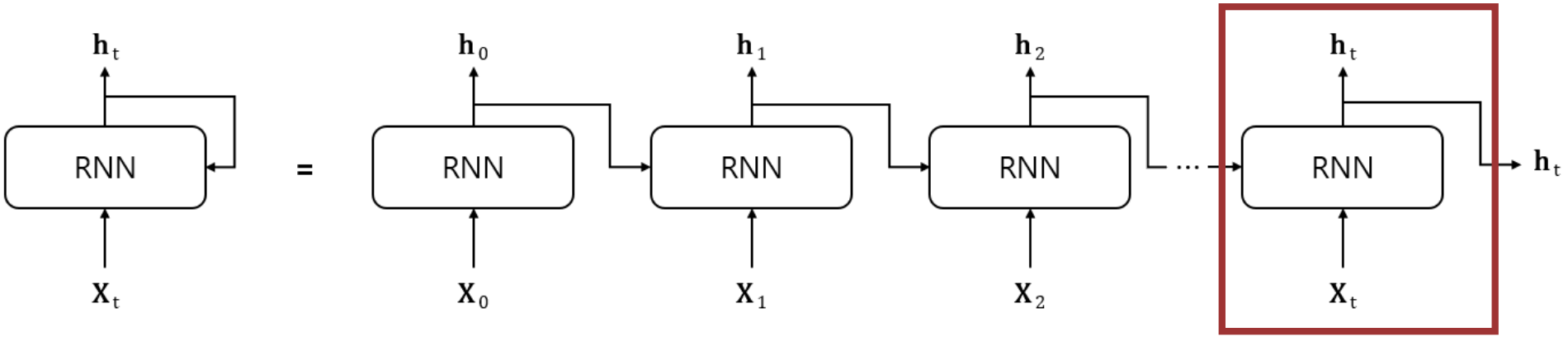


Teacher Forcing FAQ

Q. Teacher Forcing 기법이 순환신경망 (Recurrent Neural Network) 이외의 분야에서도 활용되는가?

A. 그렇다.

트랜스포머 (Transformer)와 같이 자기회귀성 (Autoregressive) 을 가진 다른 모델에도 적용된다.



Teacher Forcing Code의 구현

```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer,
decoder_optimizer, criterion, max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size,
device=device)

    loss = 0
```

Teacher Forcing Code의 구현

```
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)

    decoder_hidden = encoder_hidden

    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else
False
```

Teacher Forcing Code의 구현

```
    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach()  # detach from history as
input

            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
```

Teacher Forcing Code의 구현

```
loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.item() / target_length
```

torch.topk()

```
torch.topk(input, k, dim=None, largest=True, sorted=True, out=None) ->
(Tensor, LongTensor)
```

Returns the `k` largest elements of the given `input` tensor along a given dimension.

If `dim` is not given, the last dimension of the *input* is chosen.

If `largest` is `False` then the *k* smallest elements are returned.

A namedtuple of (*values*, *indices*) is returned, where the *indices* are the indices of the elements in the original *input* tensor.

The boolean option `sorted` if `True`, will make sure that the returned *k* elements are themselves sorted

torch.topk()

```
>>> x = torch.arange(1., 6.)
>>> x
tensor([ 1.,  2.,  3.,  4.,  5.])
>>> torch.topk(x, 3)
torch.return_types.topk(values=tensor([5., 4., 3.]), indices=tensor([4,
3, 2]))
```

torch.Tensor.detach()

- WARNING

`torch.tensor()` always copies `data`. If you have a Tensor `data` and want to avoid a copy, use `torch.Tensor.requires_grad_()` or `torch.Tensor.detach()`. If you have a NumPy `ndarray` and want to avoid a copy, use `torch.as_tensor()`.