Neural Machine Translation by Jointly Learning to Align and Translate

전자공학과 220210031 유현우

1. Encoder
   A. Input되는 sentence는 sequence of vector로 이루어져 있음
      i. 여기서 각각의 vector는 각 word를 나타내는 vector 임
      ii. $\mathbf{x} = (x_1, \ldots, x_{T_x}),\ x_i \in \mathbb{R}^{K_x}$
         1. x_i는 word vector를 의미하며 각 word vector들은 K_x의 dimension을 가지며 T_x개가 있음
         2. 이 vector들의 집합인 sentence를 x로 나타냄
         3. 이때 각각의 word를 vector 형태로 tokenize 해주는 과정은 코드상에서 spacy 라이브러리를 이용하여 수행
      iii. 이때 RNN은 아래 그림 1의 예시와 아래 수식에서와 같이 t번째 word vector와 전 layer에서 출력된 hidden state를 input으로 받음
         1. $h_t = f\left(x_t, h_{t-1}\right)$
         2. 이때 함수 f의 역할은 x_t와 h_{t-1} vector를 대상으로 learnable parameter를 이용해 linear projection을 해준 후 tanh activation function을 가해준 것을 의미

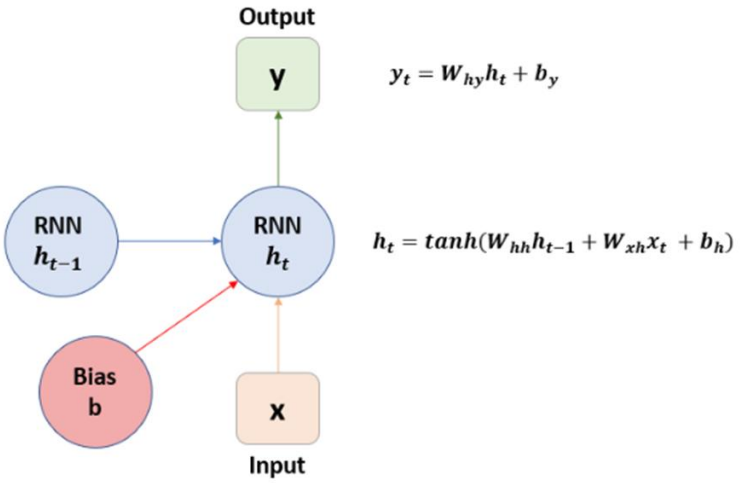

그림 1. RNN encoder layer

iv. 그러나 RNN은 과거 단어를 기반으로 미래 단어를 embedding하므로 문장이 길어지면 초반 input vector의 정보가 소실되는 문제가 있음

v. 이를 극복하기 위해 GRU(Gated Recurrent Unit) cell이 제안됨

1. 이는 아래 그림에서와 같이 forget gate, input gate, output gate를 활용하여 input 된 정보들을 long term과 short term으로 나누어 정밀하게 제어함

$$\overrightarrow{h}_i = \begin{cases} (1-\overrightarrow{z}_i) \circ \overrightarrow{h}_{i-1} + \overrightarrow{z}_i \circ \underline{\overrightarrow{h}}_i & , \text{if } i > 0 \\ 0 & , \text{if } i = 0 \end{cases}$$

2.
$$\underline{\overrightarrow{h}}_i = \tanh\left(\overrightarrow{W}\overrightarrow{E}x_i + \overrightarrow{U}\left[\overrightarrow{r}_i \circ \overrightarrow{h}_{i-1}\right]\right)$$
$$\overrightarrow{z}_i = \sigma\left(\overrightarrow{W}_z\overrightarrow{E}x_i + \overrightarrow{U}_z\overrightarrow{h}_{i-1}\right)$$
$$\overrightarrow{r}_i = \sigma\left(\overrightarrow{W}_r\overrightarrow{E}x_i + \overrightarrow{U}_r\overrightarrow{h}_{i-1}\right).$$

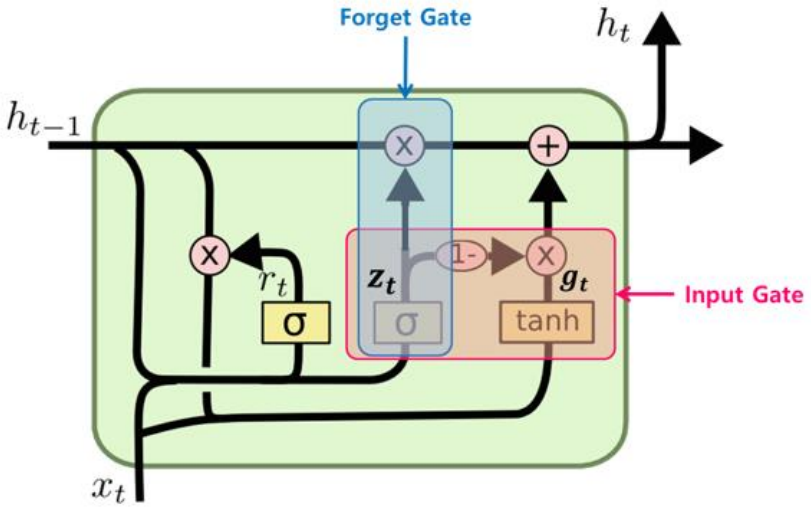

그림 2. GRU cell

vi. 또한 아래 그림3 에서와 같이 bidirectional하게 embedding을 두 방향으로 수행해주는 기법이 제안됨

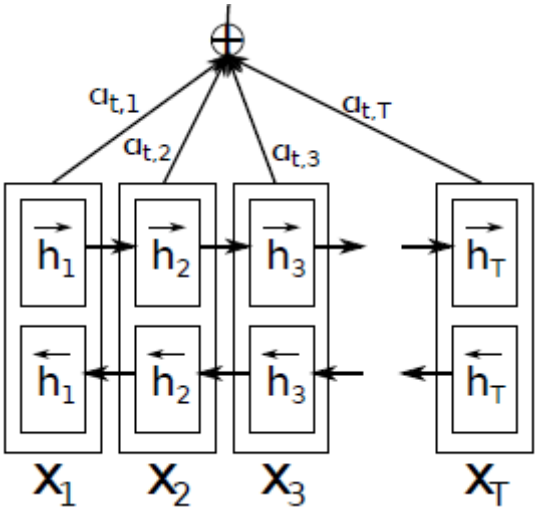

그림 3. Bidirectional RNN encoder architecture

vii. 이 논문에서는 bidirectional GRU cell을 기반으로 encoder를 구성

viii. 이는 코드상으로 아래와 같이 주어져 있음

1. Input word vector(src)를 입력받아 nn.Embedding 연산을 이용해 input_dim 에서 emb_dim으로 embedding을 수행

2. 그리고 bidirectional이 true인 GRU cell을 생성하여 outputs와 hidden을 출력

    A. 이는 각 time step에서 출력된 vector들의 집합 형태로 출력됨

    B. 이때 hidden[-2,:,:]은 forward 방향의 RNN 마지막 hidden feature를 의미하고 hidden[-1,:,:]은 backward 방향의 마지막 hidden feature를 의미

3. 따라서 이 둘을 concat해준 후 한번 더 embedding을 해준 후 tanh function을 이용해 activation 해주어 최종 hidden feature를 출력

```
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout):
        super().__init__()
        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.GRU(emb_dim, enc_hid_dim, bidirectional = True)
        self.fc = nn.Linear(enc_hid_dim * 2, dec_hid_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        embedded = self.dropout(self.embedding(src))
        outputs, hidden = self.rnn(embedded)
        hidden = torch.tanh(self.fc(torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim = 1)))
        return outputs, hidden
```

2. Decoder

    A. Decoder에서는 encoder에서 embedding한 source sentence와 target sentence와의 관계를 이용해 번역을 수행

    B. Decoder에는 target sentence의 word token과 encoder에서 embedding된 feature를 입력으로 받음

        i. Encoder의 출력은 encoder_outputs와 hidden이 있는데, encoder_outputs은 모든 forward와 backward 과정에서 출력된 hidden states이고 hidden은 forward와 backward의 마지막 hidden states

    C. 그리고 attention을 수행

        i. Hidden과 encoder_outputs를 같은 형태의 shape으로 맞춰준 후 dimension 축으로 concat한 후 embedding 수행

            1. 이때 hidden은 같은 token을 이어 붙여 encoder_outputs와 concat하기 위한 크기로 맞추어 줌

ii. 이를 통해 최종적으로 뽑힌 hidden states와 모든 hidden states와의 관계를 고려하여 embedding을 수행해줄 수 있음

    1. 이를 통해 embedding된 최종 feature 'a'를 출력

iii. 그리고 a와 encoder_outputs간의 matrix multiplication을 수행

    1. 이는 encoder에서 추출된 hidden states와 최종 embedding된 값 간의 유사도를 기반으로 attention score를 추출할 수 있음

    2. 이때 matrix multiplication 이 수행될 때 input되는 a의 shape은 [batch size, 1, src len] 이고, encoder_outputs는 [batch size, src len, enc hid dim * 2] 이므로 연산 결과는 [batch size, 1, enc hid dim * 2]의 형태로 출력됨

    3. 이를 통해 각 word vector간의 유사도를 기반으로 출력한 score임을 다시 한번 알 수 있음

iv. 최종적으로 encoder의 정보와 decoder에 입력되는 target sentence와의 관계를 한번 더 고려하여 출력해주어야 함

    1. 추출한 attention score인 weighted를 decoder에 input된 target의 첫 token에 dimension 축으로 concat한 후 embedding을 수행

    2. 이러한 방식으로 attention weights를 target token vector에 적용

v. 그림 4에서와 같이 이렇게 생성된 target token과 hidden을 encoder에서와 마찬가지로 GRU cell에 입력

vi. GRU cell mechanism에 의해 출력된 결과가 source를 기반으로 출력된 번역 문장

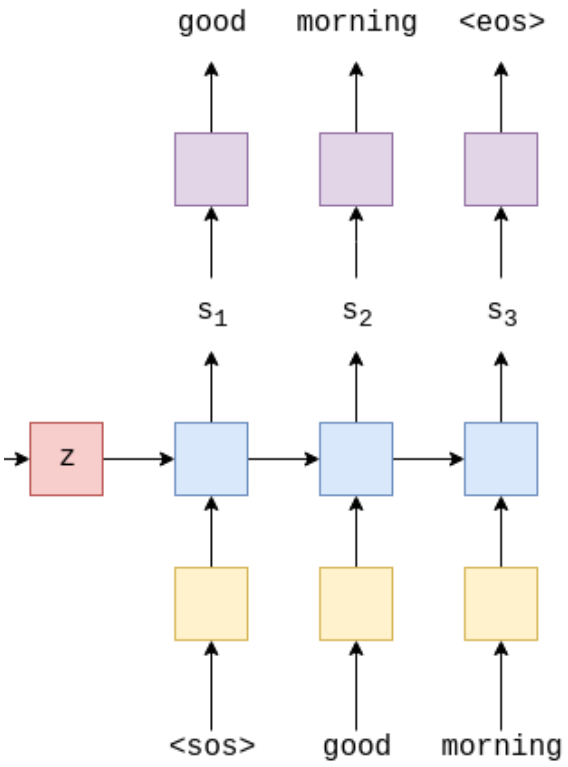

그림 4. GRU decoder

3. Train
   A. Loss
      i. Loss는 cross entropy loss를 사용
         1. $H_p(q) = -\sum_{c=1}^{C} q(y_c) log(p(y_c))$
         2. 이때 모델을 통해 출력된 분포가 p(x)이고 실제 분포는 q(x)
         3. 따라서 위의 수식을 통해 p(x)와 q(x)간의 분포 차이를 효과적으로 계산할 수 있음
   B. Optimizer는 adam optimizer를 사용
   C. Dataset은 Multi30k 사용하였으며 10 epoch 학습
   D. 학습 결과를 살펴보면 train loss와 validation loss가 줄어드는 것을 확인할 수 있음
      i. 이를 통해 학습이 제대로 수행되는 것을 확인

```
Epoch: 01 | Time: 1m 20s
        Train Loss: 5.020 | Train PPL: 151.397
         Val. Loss: 4.814 |  Val. PPL: 123.229
Epoch: 02 | Time: 1m 23s
        Train Loss: 4.128 | Train PPL:  62.080
         Val. Loss: 4.593 |  Val. PPL:  98.815
Epoch: 03 | Time: 1m 22s
        Train Loss: 3.472 | Train PPL:  32.208
         Val. Loss: 3.745 |  Val. PPL:  42.310
Epoch: 04 | Time: 1m 23s
        Train Loss: 2.916 | Train PPL:  18.459
         Val. Loss: 3.403 |  Val. PPL:  30.052
Epoch: 05 | Time: 1m 22s
        Train Loss: 2.524 | Train PPL:  12.475
         Val. Loss: 3.284 |  Val. PPL:  26.673
Epoch: 06 | Time: 1m 23s
        Train Loss: 2.231 | Train PPL:   9.309
         Val. Loss: 3.245 |  Val. PPL:  25.667
Epoch: 07 | Time: 1m 22s
        Train Loss: 1.991 | Train PPL:   7.325
         Val. Loss: 3.169 |  Val. PPL:  23.774
Epoch: 08 | Time: 1m 22s
        Train Loss: 1.774 | Train PPL:   5.897
         Val. Loss: 3.230 |  Val. PPL:  25.268
Epoch: 09 | Time: 1m 22s
        Train Loss: 1.598 | Train PPL:   4.943
         Val. Loss: 3.268 |  Val. PPL:  26.262
Epoch: 10 | Time: 1m 22s
        Train Loss: 1.492 | Train PPL:   4.447
         Val. Loss: 3.309 |  Val. PPL:  27.346
```