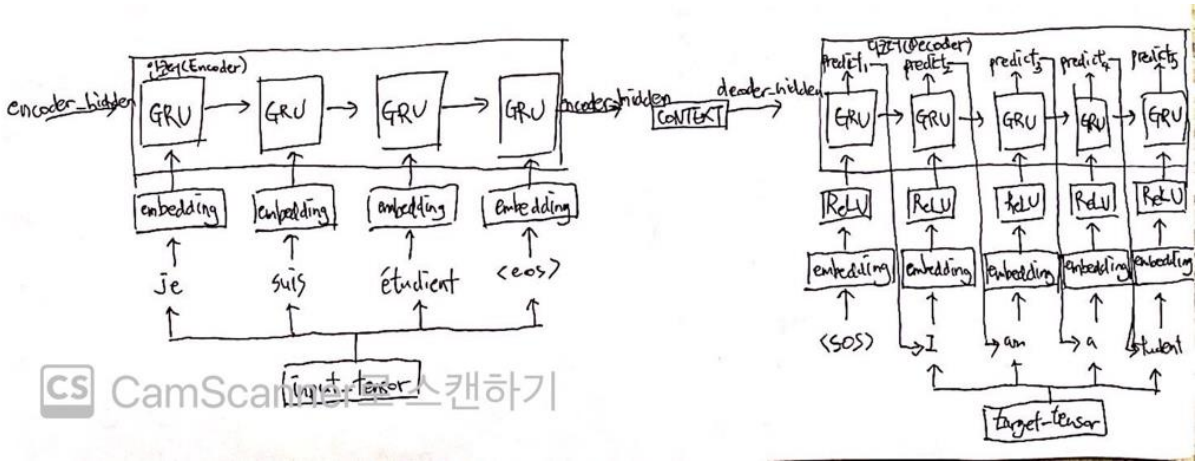
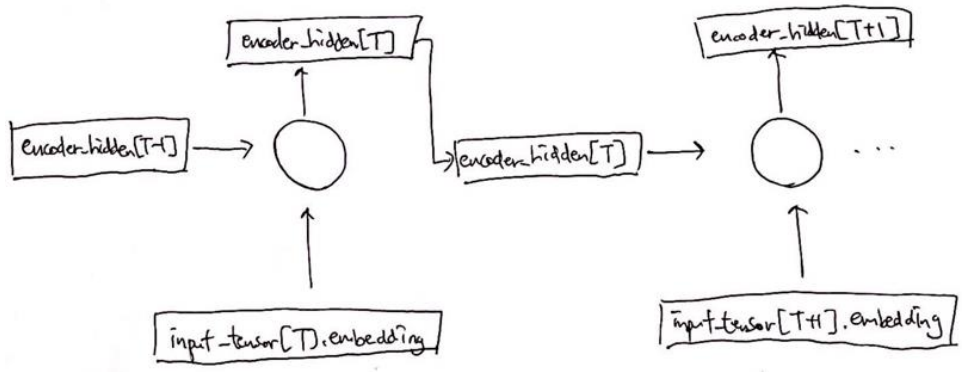
- Seq2Seq Network structure



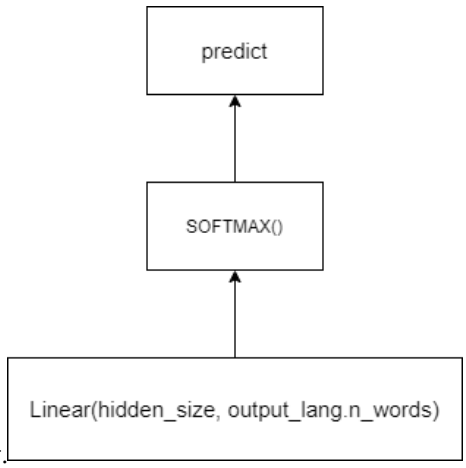
Seq2Seq network structure는 인코더와 디코더로 이루어져 있습니다. 인코더와 디코더 모두 GRU 셀들로 구성되어 있습니다. 인코더의 각 GRU 셀은 input_tensor에 있는 단어들을 순서대로 입력 받습니다. 여기서 input_tensor는 저희 모델의 경우 preprocessing된 프랑스어 문장들과 영어 문장들 중 랜덤하게 하나를 뽑았을 때의 프랑스어 문장이 되겠습니다. 저는 "je suis etudiant"라는 문장으로 가정하였습니다. 이 때 기계는 텍스트가 왔을 때 이를 어떻게 받아들여야 할지 모를 것이기 때문에 텍스트를 숫자로 바꿔줘야할 필요가 있습니다. 그래서 모든 단어들을 embedding을 통해 벡터화 시켜줍니다. 저희 모델의 경우 torch.Size([1])의 dimension을 가지던 단어들이 embedding을 거쳐 torch.Size([1, 1, 256])의 형태가 됩니다. 이제 이 embedding 벡터와 encoder_hidden이 GRU 셀로 들어가 서로 연산이 됩니다. Encoder_hidden은 embedding 벡터와 같은 size를 가지고 있고 초기값은 모두 0으로 되어있습니다. 변화한 encoder_hidden은 다음 단어와 다음 GRU 셀에 계속해서 들어가게 되고, 결국 encoder_hidden은 모든 단어들의 정보를 요약해서 담고 있다고 할 수 있습니다.



만약 T번째 단어를 볼 때라고 하면, 그림과 같이 계속해서 진행하는데, GRU 셀은 T-1번째에서의 encoder_hidden과 T번째에서의 embedding 벡터를 입력으로 받고, T번째의 encoder_hidden을 만듭니다. 그리고 이것을 T+1번째 GRU 셀의 입력으로 보냅니다.

다음으로 디코더는 인코더의 마지막 GRU 셀까지 거친 encoder_hidden을 decoder_hidden의 초기

상태로서 사용합니다. 이 decoder_hidden과 모든 target_tensor의 첫번째 값인 <sos>로부터, 다음에 등장할 단어를 예측하기 시작합니다. 여기서 target_tensor는 저희 모델의 경우 위에서 input_tensor를 받을 때 같이 뽑았던 영어 문장이 되겠습니다. 디코더도 역시 인코더와 같은 이유로 단어들이 먼저 embedding을 거칩니다. 그 다음 인코더와 다르게 GRU 셀로 들어가기 전 ReLU함수를 거칩니다. 그 후 GRU 셀로 decoder_hidden과 같이 들어가 연산을 합니다. 인코더와 동일하게 decoder_hidden은 이전까지의 단어에 대한 정보를 모두 담고 있고 계속해서 다음 GRU 셀에도 들어가게 됩니다. 하지만 디코더에서는 인코더와 다르게 단어를 예측해야 합니다. 예측된 단어를 predict라고 하면은 predict는 GRU 셀에서 바로 나오는 것이 아니라 추가적으로 다음과 같이 도출합니다.



GRU를 거쳐 나온 output과 decoder_hidden 중 output을 대상으로 Linear layer를 거쳐 원래 사이즈였던 hidden_size(저희 모델의 경우 256)만큼의 parameter들을 예측할 언어의 전체 training 데이터에서 나온 모든 단어의 수로 확장합니다. 그 후 LogSoftmax 함수를 통해 각 단어별 확률 값을 구합니다. Predict는 여기서 가장 높은 확률 값을 가진 단어가 되겠습니다.

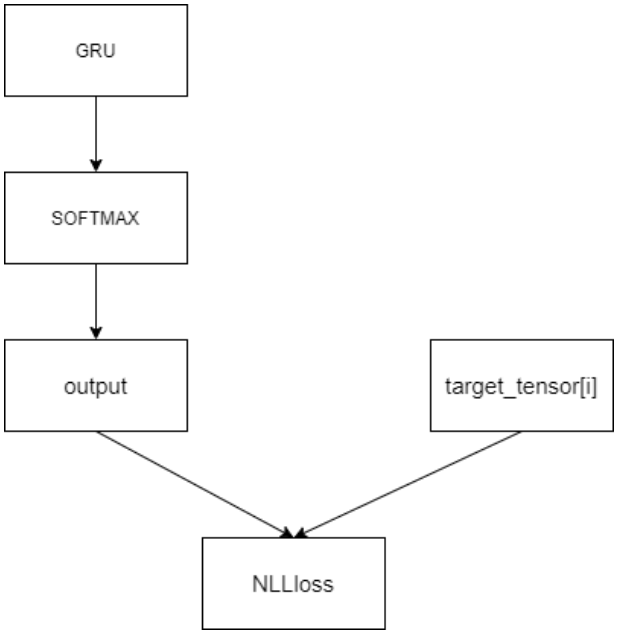
모든 문장의 첫 단어인 <sos>의 경우는 이렇게 예측이 진행되지만 그 뒤로는 약간 달라집니다. Training과 testing에 따라 또 다른데 우선 training시에는 다음 인풋이 될 단어로 그 전에 예측한 단어를 그대로 사용할 수 있고, 아니면 이전 시점의 실제 정답인 값을 사용할 수도 있습니다. 실제 정답을 사용하는 경우를 teacher forcing이라고 하는데, 저희 모델에서는 이 비율을 0.5로 선언했으므로 전체 데이터 문장들 중 반 정도는 그 전에 예측한 단어를 인풋으로 사용하고 나머지 반 정도는 이전 시점의 실제 정답을 인풋으로 사용한다고 보면 되겠습니다. Testing의 경우 모든 경우에 있어서 직전에 예측한 단어를 다음 인풋으로 사용합니다. 디코더의 다음 인풋으로 문장의 끝을 알리는 <eos>가 오게 되면 즉시 디코딩을 종료하게 됩니다.

- How to compute loss

저희 모델에서 loss구하는 기준은 NLLLoss(Negative Log Likelihood Loss) 입니다. Loss를 구하기 전 마지막 layer에서 LogSoftmax를 거치기 때문에 CrossEntropyLoss를 구하는 것과 같다고 할 수 있습니다.

i번째 단어를 예측했을 때의 loss를 구한다고 하면 아래 그림과 같이 gru 셀과 softmax를 거친

output과 실제 정답인 단어의 인덱스로 loss를 구합니다. 만약 정답 단어의 인덱스가 5라면 loss 값은 -log(output에서 5 번째의 값)이 될 것입니다.



- Dimension of tensor

Input / Output tensor of preprocessing)

먼저 preprocessing하기 전 데이터는 총 135842개의 문장 쌍으로 이루어져 있습니다. 그리고 이 문장들 중에 프랑스어 버전과 영어 버전 중 하나라도 MAX_LENGTH인 10개 이상의 단어로 이루어져 있는 버전이 존재한다면 그 쌍은 데이터에서 제외합니다. 또한 영어 버전의 문장이 eng_prefixes라는 리스트에 있는 어떤 하나의 문자열로 시작하지 않는다면 그 쌍도 제외합니다. 이 과정을 거치면 10599개의 문장 쌍들이 각각 2차원 리스트에 남게 됩니다. 이것이 train과 test를 위한 dataset이 됩니다. 모든 문장 쌍들을 탐색하면서 한 번이라도 나왔던 단어를 word2index와 index2word라는 딕셔너리에 저장합니다. 프랑스어는 4345개의 단어, 영어는 2803개의 단어가 나왔으므로 이 단어의 개수가 딕셔너리의 크기임을 알 수 있습니다.

Input / Output / hidden tensor of Encoder)

인코더의 인풋은 단어 하나이기 때문에 torch.Size([1])이 dimension임을 알 수 있습니다. Output tensor는 EncoderRNN의 forward함수를 거쳐 embedding된 벡터이므로 dimension은 torch.Size([1, 1, 256])이 됩니다. Hidden tensor는 forward를 거치기 전부터 initialize를 torch.zeros(1, 1, self.hidden_size, device=device)로 하기 때문에 Output tensor와 마찬가지로 torch.Size([1, 1, 256])이 됩니다.

Input / Output / hidden tensor of Decoder

디코더의 인풋은 처음에 torch.tensor([[SOS_token]], device=device)로 initialize되는데, 이 때는 dimension이 torch.Size([1, 1])이지만, forward가 진행된 후부터는 인풋으로 단어의 인덱스 하나만 들어오게 되면서 torch.Size([1])로 dimension이 바뀝니다. Output tensor는 인코더와 마찬가지로 embedding된 벡터로 dimension은 torch.Size([1, 1, 256])이 됩니다. Hidden tensor는 인코더에서 마지막 gru 셀까지 진행했을 때의 encoder_hidden을 그대로 받아오므로 dimension은 torch.Size([1, 1, 256])이 유지됩니다.