Attention 구조 & 분석

과목명	딥러닝프레임워크 [00]
담당교수	최인엽 교수님
학과	ICT 공학부 소프트웨어학과
학번, 이름	202184050 정가현
학번, 이름	202184031 송민아
제출일	2024. 05. 06

1. 배경지식

인코더 (encoder)

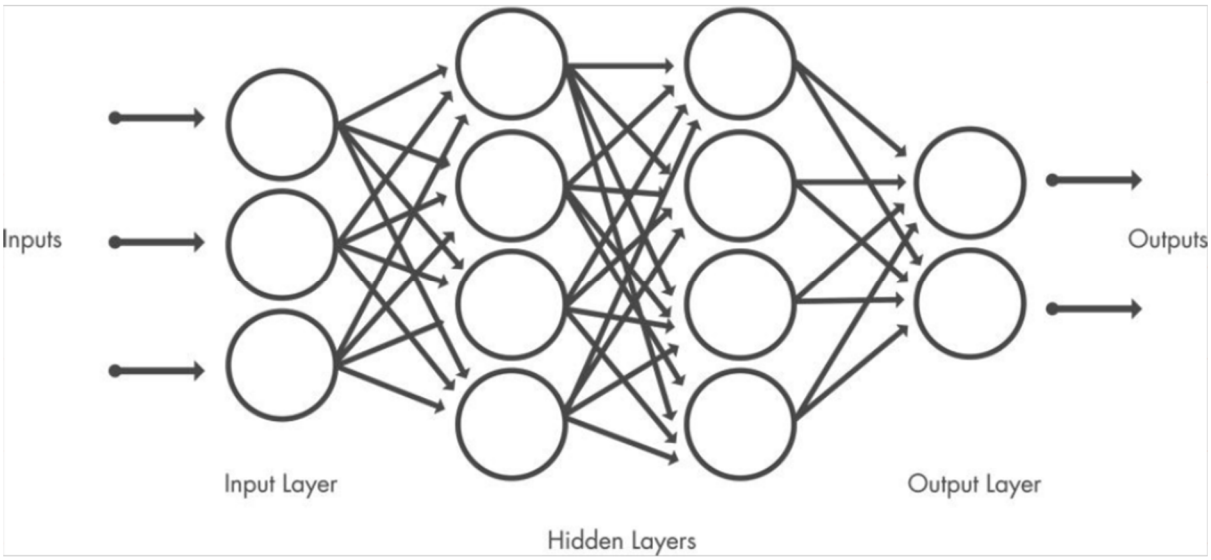
'부호기'라고도 말하며, 부호화를 수행하는 장치나 화로, 컴퓨터 소프트웨어, 알고리즘을 말한다. 즉, 입력 신호를 컴퓨터 내부에서 사용하는 코드로 변경하여 장치가 잘 알아들을 수 있게 바꿔주는 역할을 한다.

디코더 (decoder)

'복호기'라고도 말하며, 인코더와 반대로 복호화를 수행하는 장치나 회로, 컴퓨터 소프트웨어, 알고리즘을 말한다. 즉, 인코더가 한 일들을 해독해서 사람이 읽을 수 있게 바꾸는 해독기 역할이며 컴퓨터 내부의 코드를 일반적인 신호로 변경하여 출력한다.

컨볼루션 신경망 (CNN : Convolutional Neural Networks)

컨볼루션 신경망은 '합성곱 신경망'이라고도 한다. 데이터로부터 직접 학습하는 딥러닝의 신경망 architecture (설계 방식) 이다.

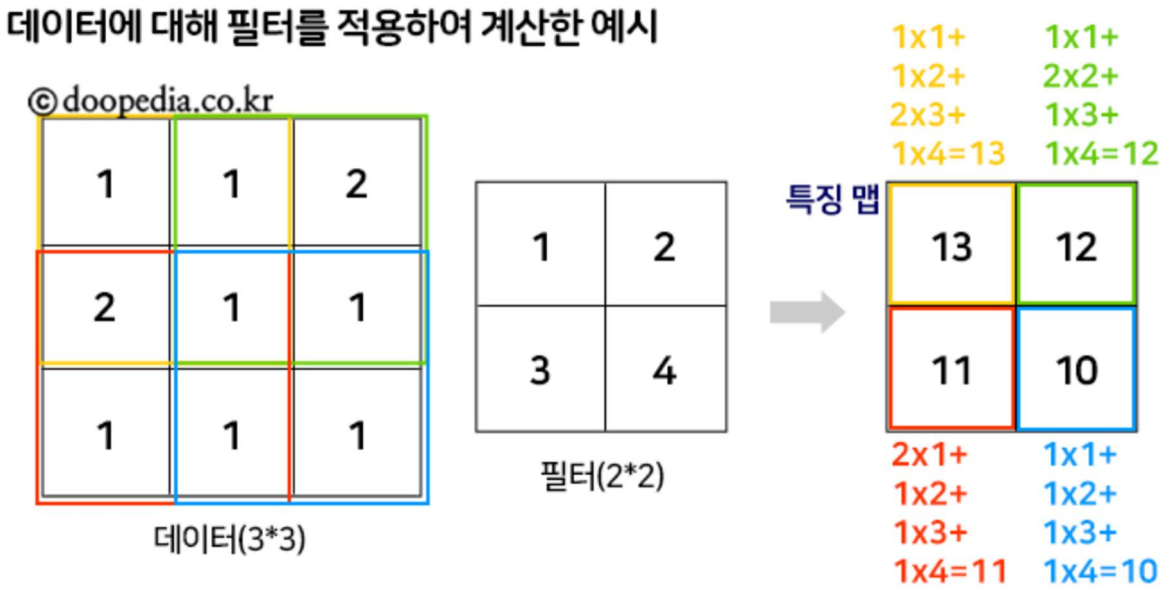


CNN은 입력 계층, 출력 계층, 그리고 그 사이의 여러 은닉 계층으로 구성된다. 각 계층은 해당 데이터의 다양한 특징을 학습하기위해 데이터를 변형시키는 연산을 수행한다.

CNN은 컨볼루션 계층을 통해 특징을 추출하고, ReLU 계층을 사용하여 비선형성을 도입한 후에, 폴링 계층을 사용하여 공간 크기를 줄이고 특징을 강화시킨다.

- 컨볼루션 계층 (합성곱 계층)

입력 영상을 일련의 Convolutional 필터 (Filter) 에 통과시킨다. 각 필터는 영상에서 특정한 특징을 활성화한다. 2차원 필터는 일반적으로 정사각형 형태의 행렬로 정의된다. 이러한 필터는 입력 이미지를 스캔하여 각 위치에서 필터의 가중치와 입력값의 요소별 곱을 계산한 후, 이를 모두 합하여 해당 부분의 특정한 속성을 추출한다. 이 과정을 통해 특징 맵(Feature map)이 생성된다. 필터의 크기와 stride (이동 거리)는 중요한 역할을 한다. 필터의 크기가 클수록 더 큰 영역에 대한 특징을 감지하게 되고, stride가 작을수록 출력 특징 map의 크기가 작아지게 된다.



컨볼루션 계층은 여러 개의 필터를 사용하여 입력 데이터의 다양한 특징을 동시에 감지한다. 각 필터는 해당하는 특징을 추출한 후, 그 결과가 특징 map에 저장된다. 만약 컴퓨터 상에서 점의 색상을 표현하는 RGB 데이터와 같이 한 점의 데이터가 3개의 채널(Channel)로 나눠진다면, 필터 또한 3개의 channel에 대해 각각 정의되어야 하며, 특징 map은 이들의 합으로 만들어진다.

- ReLU (Rectified Linear Unit) 활성화 함수 계층

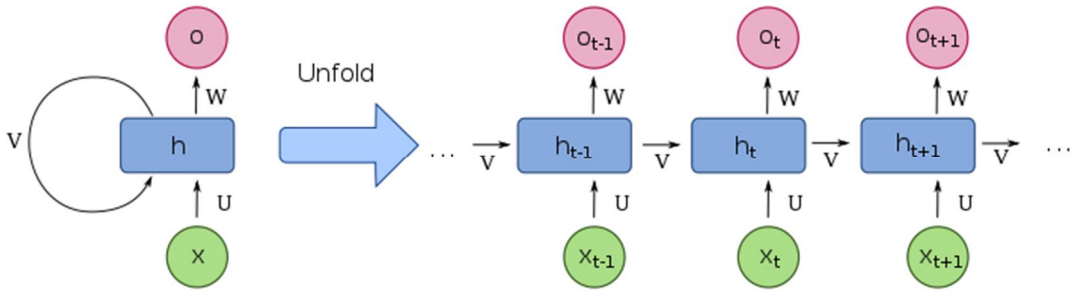
ReLU 활성화 함수 계층은 신경망에서 중요한 역할을 한다. 이 함수는 입력으로 받은 값을 처리하여 음수 값은 0 으로 만들고 양수 값은 그대로 유지한다. 이를 통해 비선형성을 도입하여 신경망이 더 복잡한 패턴을 학습할 수 있게 된다. 또한, ReLU 함수는 신경망의 학습 속도를 높이고, Gradient 소실 문제를 완화하여 신경망의 효율성을 향상시킨다. 계산이 간단하고 빠른 특징 덕분에 신경망의 훈련 과정을 효율적으로 만든다. 활성화 함수를 통과한 결과는 다음 계층으로 전달되며, 이때 활성화된 특징만이 다음 계층으로 유지된다. 이렇게 함으로써 네트워크는 입력 데이터에서 중요한 특징을 추출하고, 이를 다음 계층으로 전달하여 더 복잡한 패턴을 학습할 수 있다.

- 풀링 (pooling) 계층

풀링 계층은 데이터의 양을 줄이는 과정에서 단순한 선형적 방법 대신 비선형적인 방법을 적용하는 기법인 비선형 다운샘플링 (nonlinear downsampling)을 수행하여 신경망이 학습해야 하는 파라미터의 개수를 줄임으로써 출력을 단순화한다.

Convolutional 을 통해 얻어진 특징 맵에서, 합성곱 신경망은 특징을 요약하고 차원을 축소하기 위해 poling 연산을 적용한다. poling 연산은 합성곱 계층과 동일하게 필터의 크기가 정해져 있으며, 해당 크기의 필터를 특징 맵에 적용하여 각 부분의 최대값, 최소값, 혹은 평균을 구분한다. 이를 각각 최대 풀링 (max-pooling), 최소 풀링 (min-pooling), 평균 풀링 (average-pooling)이라고 한다.

순환신경망 (RNN : Recurrent Neural Networks)



순환신경망은 입력층에서 출력층으로의 입력값 전달과 동시에 은닉층의 정보가 다음 은닉층으로 이어지는 구조를 가진 신경망이다. 이러한 구조를 통해 순차 데이터나 시계열 데이터와 같이 일정한 순서를 가지는 데이터나 일정 시간 간격으로 배치된 데이터 수열을 처리할 수 있다.

- 시퀀스 데이터 처리

RNN 은 주로 시퀀스 데이터를 처리하는 데 사용된다. 이는 문장, 음성 데이터, 주가 변동 데이터 등과 같이 순서가 있는 데이터를 다룰 때 유용하다. 시퀀스 데이터 처리는 각 단계에서 이전 단계의 출력이 다음 단계의 입력으로 사용되는 것을 의미한다. 이렇게 함으로써 모델은 데이터 내의 패턴 및 관계를 학습하고 예측할 수 있게 된다. 예를 들어, 문장을 다룰 때 각 단어는 이전 단어의 의미와 문맥에 따라 해석되어야 한다. 순환신경망은 이러한 문맥을 파악하여 문장 내의 단어들 간의 관계를 이해하고 문장의 의미를 적절히 해석할 수 있다.

- 단기 기억 메커니즘

RNN 은 과거 정보를 기억하고 현재 입력과 함께 처리하는 데 도움이 되는 단기 메모리를 가지고 있다. 이러한 메커니즘은 모델이 시간에 따라 변화하는 입력 데이터의 패턴을 파악하고 이를 이용하여 예측을 수행할 수 있도록 한다.

단기 기억 메커니즘은 RNN 이 각 시간 단계에서 입력 데이터와 함께 이전 시간 단계의 은닉 상태를 고려하여 현재 상태를 계산하는 과정에서 발생한다. 이전 시간 단계의 정보가 현재 상태에 반영되기 때문에 모델은 과거의 입력에 대한 정보를 일부 유지하고 활용할 수 있다. 이를 통해 RNN 은 시퀀스 데이터의 패턴을 인식하고 이를 기반으로 다음 단계의 출력을 생성할 수 있다. 단기 기억 메커니즘은 자연어 처리 (Natural Language Processing), 음성 인식 (Speech Recognition), 시계열 데이터 분석 등 다양한 응용 분야에서 유용하게 활용된다. 예를 들어, 자연어 처리에서 RNN 은 문장 내의 단어들 간의 의미적 관계를 파악하고 문맥을 이해하는 데 도움이 된다.

- 기울기 소멸 문제 (Vanishing Gradient Problem)

RNN 은 긴 Sequence 를 처리할 때 발생하는 Vanishing Gradient 문제를 가지고 있다. 이 문제는 역전파 과정에서 그래디언트 (Gradient) 가 사라져서 모델이 제대로 학습되지 않을 수 있음을 의미한다.

RNN 은 순환 구조를 가지고 있기 때문에 입력과 출력 사이에 숨겨진 상태를 전달하면서 작동한다. 하지만 긴 시퀀스를 처리할 때, 역전파 과정에서 이전 시간 단계로 거슬러 올라가면서 그래디언트가 곱해지는데, 이 과정에서 Gradient 가 너무 작아지거나 사라지는 경우가 발생할 수 있다. 이는 주로 활성화 함수인 시그모이드 (Sigmoid) 함수의 미분 값이 0 에 가까워져서 발생한다.

Vanishing Gradient 문제로 인해 모델의 학습이 느려지거나 중단되는 경우가 있다. 특히 RNN 이 처리해야 할 시퀀스의 길이가 길어질수록 이 문제가 더 심각해진다. 이를 해결하기 위해 다양한 변형된 RNN 아키텍처나 그래디언트 클리핑, LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Unit) 같은 메모리 셀을 사용하는 방법 등이 제안되어 왔다.

Vanishing Gradient 문제를 극복하기 위한 연구와 개발은 순환신경망의 성능을 향상시키는 데 중요한 역할을 한다. 이를 통해 RNN 이 더 장기적인 의존성을 갖는 시퀀스 데이터를 효과적으로 학습하고 활용할 수 있게 된다.

- 장단기 메모리 (LSTM : Long Short-Term Memory)

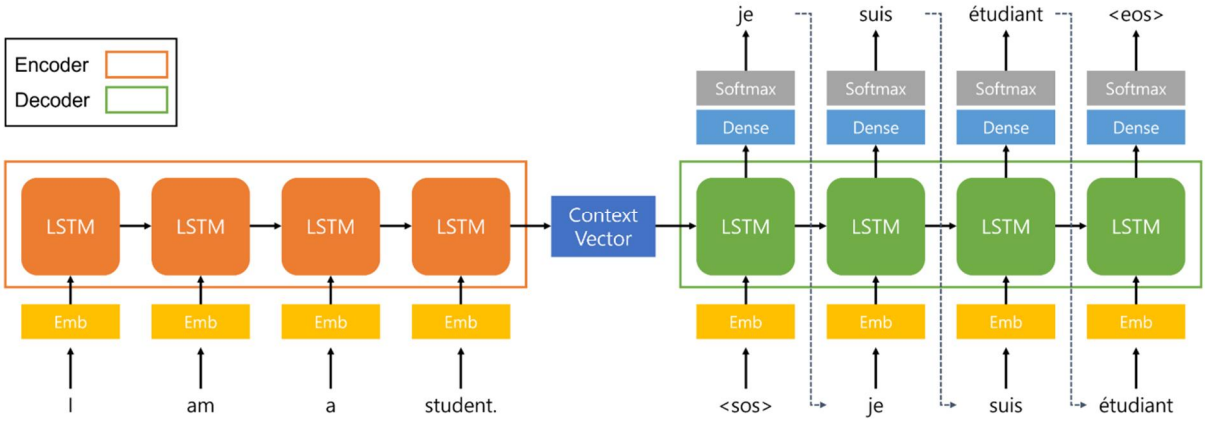
장단기 메모리는 기존의 순환신경망에서 발생하는 기울기 소멸 문제를 해결하기 위해 고안되었다. 이를 위해 LSTM 은 셀, 입력 게이트, 출력 게이트, 망각 게이트라는 네 가지 핵심 구성 요소를 사용한다. 각 구성 요소는 다양한 방식으로 정보를 관리하고 제어하여 모델이 장기적인 의존성을 효과적으로 학습할 수 있도록 도와준다.

셀은 기존의 순환신경망에서 사용되는 은닉 상태와 달리 메모리 셀 상태를 포함한다. 이 메모리 셀은 장기적인 정보를 저장하고 전달하는 역할을 한다. **입력 게이트**는 현재 입력에 대한 정보를 얼마나 기억할지를 결정한다. 이를 통해 새로운 정보를 메모리에 추가하거나 무시하는 역할을 한다. **출력 게이트**는 현재의 메모리 셀 상태를 기반으로 다음 순서의 출력을 조절한다. 이를 통해 모델이 적절한 시기에 출력을 생성할 수 있도록 한다. **망각 게이트**는 이전의 정보 중에서 어떤 정보를 잊을지를 결정한다. 이를 통해 모델이 불필요한 정보를 잊고 새로운 정보를 기억하는 데 도움이 된다.

이러한 구조를 통해 LSTM 은 긴 시퀀스 데이터를 처리하면서도 기울기 소멸 문제를 방지하고, 장기적인 의존성을 효과적으로 학습할 수 있다.

2. Seq2Seq with Attention

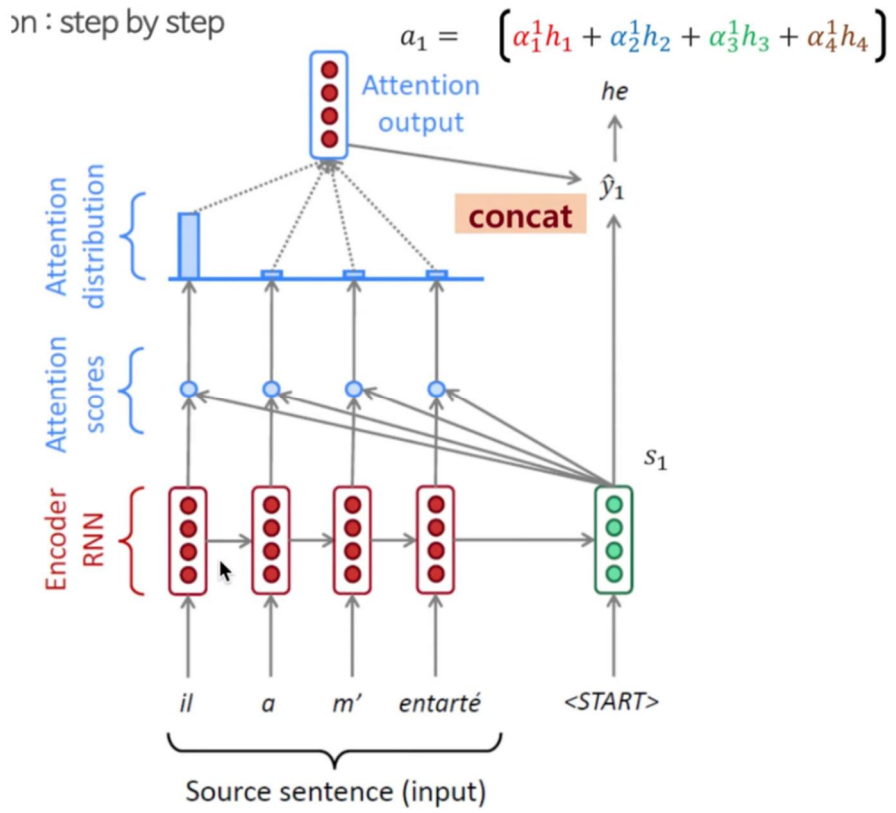
Seq2Seq Model



Seq2Seq 모델은 하나의 Encoder 와 하나의 Decoder 로 구성되며, Encoder 는 입력 시퀀스의 각 아이템을 처리하여 Context Vector 를 생성하고, Decode 는 이를 기반으로 다음 시퀀스의 아이템을 예측하여 전체 Sequence 를 생성한다. 그러나 이 과정에서 Vanishing Gradient Problem 이 발생할 수 있다. Context vector 로 입력된 문장의 길이를 압축할 때 데이터 손실을 방지하기 위해 Attention 기법이 사용된다. Attention 은 Decoder 가 각각의 출력을 생성할 때 입력 Sequence 의 특정 부분에 집중할 수 있도록 도와주는 메커니즘이다.

Attention

Attention 의 기본 아이디어는 " Decoder 에서 각 단어를 생성할 때, Context vector 뿐만 아니라 은닉층의 모든 hidden state 값과 현재 Decoder 에서 생성되는 단어와의 관계를 고려하자 " 이다.



1. 내적값이 낮은 순으로 높은 점수를 부여하는 Attention scores 를 계산하여 Energy (모든 hidden state 에 대한 Attention score 의 집합 벡터) 값을 도출한다.
2. Energy 값에 softmax (들어간 값을 확률로 바꾸는함수)를 적용해 얻은 Attention 분포를 도출하고, Attention 분포의 각각의 값인 Attention 가중치를 도출한다.
3. Attention 가중치에 Encoder 의 hidden state 의 가중 합한 것으로 현재 Decoder 의 시점에 대한 입력 Sequence 의 가중 표현이다. $a^t = \sum_{i=1}^{N} \alpha_i^t h_i$
4. Decoder 의 hidden state 에 Attention value 를 결합하여 Dense Layer 입력값을 구성한다.
   - 단순히 결합(concat) 하는 방법
   - 차원이 일치할 경우, Decoder 의 t 번째 hidden state 에 더하는 방법
5. Dense Layer 통과 이후 Softmax 를 이용해 출력 단어 생성한다.

3. NEURAL MACHINE TRANSLATION

BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

ABSTRACT & INTRODUCTION

신경망 번역 모델은 최근에 제안된 기계 번역 방법론이다. 과거의 번역 모델은 전통적인 통계 기반의 번역 모델로써 다양한 sub-component로 구성되어 있었고, 각 component는 각각 학습되고 구성되었다. 이후에는 하나의 큰 neural network를 이용한 translation 방법들이 제안되었다. Neural Machine Translation은 encoder-decoder 형식으로 이루어져 있으며, 번역 성능을 최대화하기 위해 하나의 신경망을 조율하는 것을 목표로 한다.

이러한 encoder-decoder 구조는 Bottleneck Problem (병목현상 : 하나의 구성요소로 인해 시스템 내에서 전체 시스템의 성능이나 용량이 제한 받는 현상) 를 발생시킬 수 있다. Bottleneck Problem은 encoder에서 전체 문장을 하나의 고정된 길이의 벡터로 생성할 때 발생한다.

이 논문은 **encoder-decoder 모델에서 새로운 구조를 추가해서 성능을 향상할 수 있는 방법을 제안**한다. 이러한 방식의 가장 큰 장점 중 하나는 입력 문장을 고정된 길이의 벡터로 표현하지 않아도 된다는 것이다. Decoder에서 연산을 진행하면서 Encoder에서 생성한 Context vector를 계속해서 참조하기 때문에, 전체 문장의 정보를 하나의 벡터에 담으려고 하지 않아도 되고, 문장의 길이가 길어지더라도 성능을 유지할 수 있다.

BACKGROUND: NEURAL MACHINE TRANSLATION

확률론적으로 Neural Machine Translation(NMT)을 보면, 입력 문장 s 가 주어졌을 때, 조건부 확률 $p(y|x)$ 를 최대화 하는 출력 문장 yy 를 찾아내는 방식으로 동작한다.

$argmaxy(y|x)$) 최근에는 neural network 를 이용해서 이러한 조건부 확률을 직접적으로 학습하는 방법도 제안되었다. 이러한 방식은 앞서 언급한 RNN을 이용한 encoder-decoder 모델을 예로 들 수 있다. 이후에는 LSTM을 이용해서 RNN 보다 좋은 성능을 내는 모델도 제안되었다.

- RNN을 이용한 encoder-decoder 모델

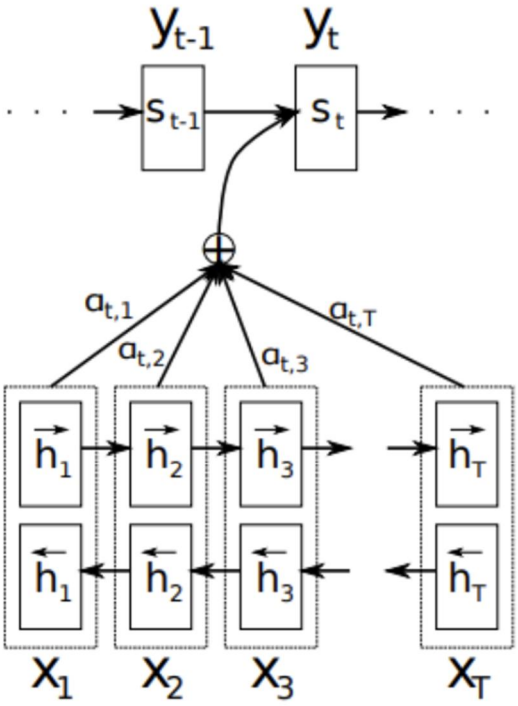
$$h_t = f\left(x_t, h_{t-1}\right) \qquad c = q\left(\{h_1, \cdots, h_{T_x}\}\right)$$

Encoder는 입력으로 제공되는 문장 $\boldsymbol{x=(x1,x2,\ldots,xTx)}$ 을 고정된 길이의 벡터 $\boldsymbol{c}$ 로 변환하게 된다. RNN을 이용하는 경우 위 식을 통해서 벡터 $\boldsymbol{c}$ 를 생성한다. 첫번째 식에서 $\boldsymbol{ht}$는 t (time) 에서의 hidden state를 의미한다. $f()$,$q()$ 는 non-linear function을 의미한다.

Decoder는 context vector $\boldsymbol{c}$ 가 Encoder로부터 주어졌을 때, $\boldsymbol{c}$ 와 이전에 예측한 결과 $\boldsymbol{y1,y2,\ldots,yt'-1}$ 을 기반으로 다음 단어 $\boldsymbol{yt'}$ 를 예측하게 된다. 번역된 결과 $\boldsymbol{y=(y1,\ldots,yTy)}$ 는 아래의 첫번째 식과 같은 조건부 확률을 기반으로 생성된다. 조건부 확률은 바로 직전 time인 $\boldsymbol{(t-1)}$ 에서 예측한 결과 $\boldsymbol{yt-1}$ 과, RNN의 hidden state $\boldsymbol{st}$, 그리고 non-linear function $\boldsymbol{g()}$ 를 이용해서 구할 수 있다.

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) \qquad p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

LEARNING TO ALIGN AND TRANSLAT



Encoder : 입력 문장 $x=(x1,...,xTx)$ 는 Encoder 의 입력으로 제공된다. 입력 문장에 대해서 이전에 나타는 내용과 함께, 이후에 나타나는 내용도 알아야지 좋은 번역이 가능하기 때문에, 이번 모델에서는 **bidirectional RNN(BiRNN)**을 사용한다. BiRNN 은 두 개의 RNN**(forward RNN, backward RNN)**으로 구성된다. Forward RNN 은 처음부터 순차적으로 입력을 읽어서 순전파 hidden state 생성한다. Backward RNN 은 입력의 제일 뒤에서부터 역방향으로 입력을 읽어서 역전파 hidden state 를 생성한다.

$$h_j = \left[\overrightarrow{h}_j^\top ; \overleftarrow{h}_j^\top\right]^\top$$

Time j 에서의 입력 xj 에 대해서 forward hidden state 순방향 hj 와 backward hidden state 역방향 hj 를 연결해서 **j**-번째 hidden state hj 를 생성한다. 인접한 state 의 정보를 더 많이 가지고 있는 RNN 의 특성상, hj는 입력 단어 xj 의 가까운 위치에 있는 단어들의 정보를 더 많이 보유하게 된다.

$$e_{ij} = a(s_{i-1}, h_j)$$

Decoder : Time i 에서 decoder의 hidden state를 si 이라고 한다. Alignment model에서는

decoder의 time i 에서의 정보다 encoder의 time j 에서의 정보와 얼마나 연관성이 있는지 score 를 계산한다. 해당 Score는 decoder 의 바로 전 time $(i-1)$ 에서의 hidden state $st-1$ 과 encoder 의 time j 에서의 hidden state hj 를 이용해 위의 식과 같은 연산을 수행한다. 이 때 alignment model 인 $a()$ 는 feedforward neural network 라고 할 수 있다.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

이후, time i 에서의 context vector 인 ci 를 생성한다. ci 는 encoder의 모든 hidden state 의 weighted sum으로 구성된다. Encoder의 hidden state 들은 모든 입력에 대한 정보들을 담고 있다.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

이 때, encoder의 각 hidden state에 대한 weight aij 는 alignment model에서 구한 score eij 를 이용해서 위의 식과 같이 구해진다.

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

이후에는 위의 식과 동일한 방법으로 아래 식인 time i 에서의 decoder RNN의 hidden state si 를 계산한다.

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

EXPERIMENT SETTINGS

Model	All	No UNK°
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

데이터셋 : WMT'14 의 English-French parallel corpus 를 사용했다. 전체 corpus의 단어를 348M 개로 제한했고, monolingual data는 하나도 사용하지 않았다. 전체 단어 중 가장 많이 사용되는 30,000개의 단어를 사용했고, 추가적으로 [UNK] 토큰도 함께 사용했다.

모델 : 총 2개의 모델을 학습했다. 첫 번째 모델은 RNN Encoder-Decoder 모델이고, 두 번째 모델은 이번 논문에서 제안한 모델(RNNsearch 라고 부른다.) 이다. 두 모델을 각각 두 번씩 학습한다. 첫 학습에서는 최대 30개의 단어들로 구성된 문장들에 대해서 학습을 진행하고, 두 번째 학습에서는 최대 50개의 단어들로 구성된 문장들로 학습을 진행한다. 30개의 단어로 구성된 문장으로 학습된 모델을 RNNencdec-30, RNNsearch-30 이라고 부르고, 50개의 단어로 구성된 문장으로 학습된 모델을 RNNencdec-50, RNNsearch-50 이라고 부른다. RNNencdec는 1000개의 hidden unit을 보유한다. RNNsearch의 경우, encoder는 forward/backward 각각 1000개의 hidden unit을 보유하고, decoder에서도 1000개의 hidden unit을 보유한다. minibatch SGD를 사용하고 Adadelta를 사용했다. minibatch는 80개의 문장으로 구성되었고, 총 5일 동안 학습을 진행했다.

RESULTS

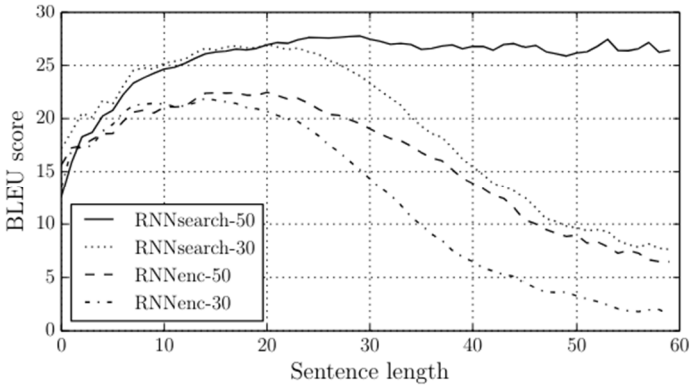
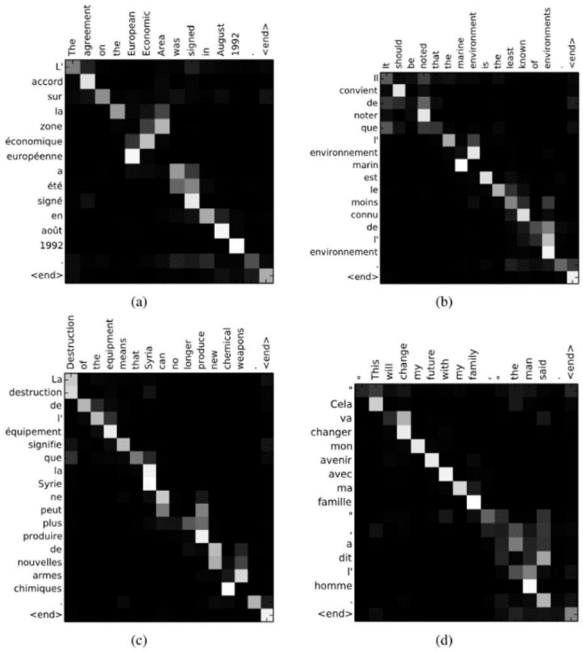


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

BLEU score를 기반으로 translation 성능을 평가했다. RNNsearch 가 모든 경우 더 뛰어난 성능을 제공했다. 또한, 기존의 phrase-based translation system(Moses) 와 유사한 성능을 제공할 수 있었다. Moses의 모델의 경우 추가적인 monolingual corpus를 사용했다는 점에서 이번 모델이 뛰어난 성능을 제공한다는 것을 알 수 있다.

이번 모델의 큰 장점 중 하나는 source sentence를 하나의 고정된 길이의 벡터로 변환하지 않아도 된다는 점이다. Source sentence를 하나의 고정된 길이의 벡터로 변환하는 경우, 문장의 길이가 길어지면 성능이 떨어진다는 것을 알 수 있다. 하지만, RNNsearch-30, RNNsearch-50의 경우, source sentence 가 길어져도 어느정도 일관된 성능을 제공한다. RNNsearch-50은 특히 문장이 길이가 계속 길어져도 성능 저하가 없다는 것을 볼 수 있다.



왼쪽은 annotation weight αij 를 시각화한 것이다. 해당 그림을 통해서 번역된 결과가 어떠한 입력에 (soft-)align 되었는지 확인할 수 있다. Hard-alignment의 경우, 한 단어가 무조건 다른 한 단어와 매핑되어야 하는데, 실제로 번역을 하는 경우 이렇게 1:1로 모든 단어를 매핑하는데 한계가 있다. 반면에 이번 모델에서 제안하는 soft-alignment의 경우, 한 단어가 여러 단어와의 관계를 나타낼 수 있으므로 번역을 하는데 더 좋은 성능을 제공할 수 있다.