

## Advanced Topic on Neural Machine Translation



Figure 1: Christopher Manning – Image from Web

## Multi-lingual Neural Machine Translation

이제부터는 기계번역의 성능을 끌어올리기 위한 고급 기법들을 설명하고자 합니다. 코드를 직접 구현하고 실습을 해 보기보단, 논문을 소개하는 위주가 될 것 입니다. 앞으로 소개할 기술(논문)들은 일부는 기계번역에만 적용 가능한 기술들도 있지만, 자연어 생성 또는 시퀀스 데이터 생성에 응용될 수 있는 기술들도 있습니다.

기존의 번역 시스템이 Seq2seq를 필두로 어느정도 안정된 성능을 제공함에 따라서, 이를 활용한 여러가지 추가적인 연구 주제가 생겨났습니다. 하나의 end2end 모델에서 여러 언어쌍의 번역을 동시에 제공하는 multi-lingual 기계번역이 그 주제중에 하나입니다.

## Zero-shot Learning

이 흥미로운 방식은 [Johnson et al. 2016]에서 제안 되었습니다. 이 방식의 특징은 여러 언어쌍의 parallel 코퍼스를 하나의 모델에 훈련하면 부가적으로 parallel 코퍼스에 존재하지 않은 언어쌍도 번역이 가능하다는 것 입니다. 즉, 한번도 기계번역 모델에게 데이터를 보여준 적이 없지만, 해당 언어쌍 번역을 처리할 수 있기 때문에, zero-shot learning이라는 이름이 붙었습니다. 뿐만 아니라, parallel 코퍼스가 적은 경우에도 부가적인 효과를 발휘 합니다.

방법은 너무 간단합니다. 아래와 같이 기존 parallel 코퍼스의 맨 앞에 특수 토큰을 삽입함으로써 완성됩니다. 삽입된 토큰에 따라서 target 언어가 결정됩니다.

- Hello, how are you? → Hola, ¿c□mo est□s?
- <2es> Hello, how are you? → Hola, ¿c□mo est□s?

실험의 목표는 단순히 Multi-lingual end2end model을 구현하는 것이 아닌, 다른 언어쌍의 코퍼스를 활용하여 특정 언어쌍 번역기의 성능을 올릴 수 있는가에 대한 관점도 있습니다. 이에 따라 실험은 크게 4가지 관점에서 수행되었습니다.

| 번호 | 방법                    | 설명   |
|----|-----------------------|--|
| 1  | Many to One           | 다수의 언어를 encoder에 넣고 훈련시킵니다.                        |
| 2  | One to Many           | 다수의 언어를 decoder에 넣고 훈련시킵니다.                        |
| 3  | Many to Many          | 다수의 언어를 encoder와 decoder에 모두 넣고 훈련시킵니다.            |
| 4  | Zero-shot Translation | 위의 방법으로 훈련된 모델에서 zero-shot translation의 성능을 평가합니다. |

언어가 다른 코퍼스를 하나로 합치다보면 양이 다르기 때문에 이에 대한 대처 방법도 정의 되어야 합니다. 따라서 아래의 실험들에서 oversampling 기법의 사용 유무도 같이 실험이 되었습니다. Oversampling 기법은 양이 적은 코퍼스를 양이 많은 코퍼스에 양과 비슷하도록 (데이터를 반복시켜) 양을 늘려 맞춰주는 방법을 말합니다.

### Many to One

이 실험에서는 전체적으로 성능이 향상 된 것을 볼 수 있습니다. 하단의 일본어, 한국어, 스페인어, 포르투갈어 실험의 경우에는 모두 oversampling을 기준으로 실험되었습니다.

Many to One: BLEU scores on various data sets for single language pair and multilingual models.

|                                      | Model | Single | Multi | Diff |
|--------------------------------------|-------|--------|-------|------|
| WMT German→English (oversampling)    | 30.43 | 30.59  | +0.16 |      |
| WMT French→English (oversampling)    | 35.50 | 35.73  | +0.23 |      |
| WMT German→English (no oversampling) | 30.43 | 30.54  | +0.11 |      |
| WMT French→English (no oversampling) | 35.50 | 36.77  | +1.27 |      |
| Prod Japanese→English                | 23.41 | 23.87  | +0.46 |      |
| Prod Korean→English                  | 25.42 | 25.47  | +0.05 |      |
| Prod Spanish→English                 | 38.00 | 38.73  | +0.73 |      |
| Prod Portuguese→English              | 44.40 | 45.19  | +0.79 |      |

Figure 2: Many to One

## One to Many

One to Many: BLEU scores on various data sets for single language pair and multilingual models.

|                                      | Model | Single | Multi | Diff |
|--------------------------------------|-------|--------|-------|------|
| WMT English→German (oversampling)    | 24.67 | 24.97  | +0.30 |      |
| WMT English→French (oversampling)    | 38.95 | 36.84  | -2.11 |      |
| WMT English→German (no oversampling) | 24.67 | 22.61  | -2.06 |      |
| WMT English→French (no oversampling) | 38.95 | 38.16  | -0.79 |      |
| Prod English→Japanese                | 23.66 | 23.73  | +0.07 |      |
| Prod English→Korean                  | 19.75 | 19.58  | -0.17 |      |
| Prod English→Spanish                 | 34.50 | 35.40  | +0.90 |      |
| Prod English→Portuguese              | 38.40 | 38.63  | +0.23 |      |

Figure 3: One to Many

이 실험에서는 이전 실험과 달리 성능의 향상이 있다고 보기 힘듭니다. 게다가 oversampling과 관련해서 코퍼스의 양이 적은 영어/독일어 코퍼스는 oversampling의 이득을 본 반면, 양이 충분한 영어/프랑스어 코퍼스의 경우에는 oversampling을 하면 더 큰 손해를 보는 것을 볼 수 있습니다.

## Many to Many

이 실험에서도 대부분의 실험결과가 성능의 하락으로 이어졌습니다. (그렇지만 절대적인 BLEU 수치는 쓸만합니다.)

Many to Many: BLEU scores on various data sets for single language pair and multilingual models.

|                                      | Model | Single | Multi | Diff |
|--------------------------------------|-------|--------|-------|------|
| WMT English→German (oversampling)    | 24.67 | 24.49  | -0.18 |      |
| WMT English→French (oversampling)    | 38.95 | 36.23  | -2.72 |      |
| WMT German→English (oversampling)    | 30.43 | 29.84  | -0.59 |      |
| WMT French→English (oversampling)    | 35.50 | 34.89  | -0.61 |      |
| WMT English→German (no oversampling) | 24.67 | 21.92  | -2.75 |      |
| WMT English→French (no oversampling) | 38.95 | 37.45  | -1.50 |      |
| WMT German→English (no oversampling) | 30.43 | 29.22  | -1.21 |      |
| WMT French→English (no oversampling) | 35.50 | 35.93  | +0.43 |      |
| Prod English→Japanese                | 23.66 | 23.12  | -0.54 |      |
| Prod English→Korean                  | 19.75 | 19.73  | -0.02 |      |
| Prod Japanese→English                | 23.41 | 22.86  | -0.55 |      |
| Prod Korean→English                  | 25.42 | 24.76  | -0.66 |      |
| Prod English→Spanish                 | 34.50 | 34.69  | +0.19 |      |
| Prod English→Portuguese              | 38.40 | 37.25  | -1.15 |      |
| Prod Spanish→English                 | 38.00 | 37.65  | -0.35 |      |
| Prod Portuguese→English              | 44.40 | 44.02  | -0.38 |      |

Figure 4: Many to Many

Portuguese→Spanish BLEU scores using various models.

|     | Model                          | Zero-shot | BLEU  |
|-----|--------------------------------|-----------|-------|
| (a) | PBMT bridged                   | no        | 28.99 |
| (b) | NMT bridged                    | no        | 30.91 |
| (c) | NMT Pt→Es                      | no        | 31.50 |
| (d) | Model 1 (Pt→En, En→Es)         | yes       | 21.62 |
| (e) | Model 2 (En↔{Es, Pt})          | yes       | 24.75 |
| (f) | Model 2 + incremental training | no        | 31.77 |

Figure 5: Zero-shot Translation

## Zero-shot Translation

이 실험은 Zero-shot learning의 성능을 평가하였습니다. bridged 방법은 중간 언어를 영어로 하여 *Portuguese → English → Spanish* 2단계에 걸쳐 번역을 한 경우를 말합니다. (PBMT 방식은 SMT방식 중의 하나입니다.) *NMT Pt → Es*는 단순 parallel 코퍼스를 활용하여 기존의 방법대로 훈련한 baseline입니다.

모델1은 *Pt → En*, *En → Es*를 한 모델에 훈련한 버전입니다. 그리고 모델2는 *En ↔ Pt*, *En ↔ Es* 코퍼스를 한 모델에 훈련한 버전입니다. 모델2는 총 4가지 코퍼스를 훈련 한 점을 주의해야 합니다.

마지막으로 모델2 + incremental 학습 방식은 (c) 보다 적은양의 parallel 코퍼스를 기준에 훈련한 모델2에 추가적으로 훈련한 모델입니다.

비록 모델1과 모델2는 훈련 중에 한번도 *Pt → Es* 데이터를 보지 못했지만, 20이 넘는 BLEU를 보여주는 것을 알 수 있습니다. 하지만 bridge 방식의 (a), (b) 보다 성능이 떨어지는 것을 알 수 있습니다. 다행히도 (f)의 경우에는 (c)보다 (큰 차이는 아니지만) 성능이 뛰어난 것을 알 수 있습니다. 따라서 우리는 parallel 코퍼스의 양이 얼마 되지 않는 언어쌍의 번역기를 훈련할 때에 위와 같은 방법을 통해서 성능을 끌어올릴 수 있음을 알 수 있습니다.

## Conclusion

앞서 다룬 monolingual 코퍼스를 활용하는 방법의 연장선상으로써 위와 같이 multi-lingual 기계번역 모델로서는 의의가 있지만, 성능에 있어서는 이득이 있었기 때문에 실제 사용에는 한계가 있습니다. 더군다나 훈련 데이터의 양이 적은 언어쌍에 대해서는 성능의 향상이 있지만 뒤 챕터에 설명할 방법들을 사용하면 그다지 좋은 방법은 아닙니다.

## Applications

우리는 합성 토큰을 추가하는 방식을 다른 곳에서도 응용할 수 있습니다. 다른 도메인의 데이터를 하나로 모아 번역기를 훈련시키는 과정에 사용 가능합니다. 예를 들어 코퍼스를 뉴스기사와 미드 자막에서 각각 모았다고 가정하면, 문어체와 대화체로 도메인을 나누어 특수 토큰을 추가하여 우리가 원하는대로 번역문의 말투를 바꾸어줄 수 있을 겁니다. 또는 마찬가지로 의료용과 법률용으로 나누어 번역기의 모드를 바꾸어줄 수 있을 겁니다.

# Improve Neural Machine Translation using Monolingual Corpora

번역 시스템을 훈련하기 위해서는 다양한 Parallel Corpus(병렬 말뭉치)가 필요합니다. 필자의 경험상 대략 300만 문장쌍이 있으면 완벽하지는 않지만 나름 쓸만한 번역기가 나오기 시작합니다. 하지만 인터넷에는 정말 수치로 정의하기도 힘들 정도의 monolingual corpus가 널려 있는데 반해서, 이러한 parallel corpus를 대량으로 얻는 것은 굉장히 어려운 일입니다. 또한, monolingual corpus가 그 양이 많기 때문에 실제 우리가 사용하는 언어의 확률분포에 좀 더 가까울 수 있고, 따라서 언어모델을 구성함에 있어서 훨씬 유리합니다. 즉, 앞으로 소개할 기법들은 다양한 monolingual corpus를 활용하여 해당 언어의 언어모델을 보완하는 것이 주 목적입니다. 이번 섹션은 이러한 값싼 monolingual corpus를 활용하여 신경망 기계번역 시스템의 성능을 쥐어짜는 방법들에 대해서 다룹니다.

## Language Model Ensemble

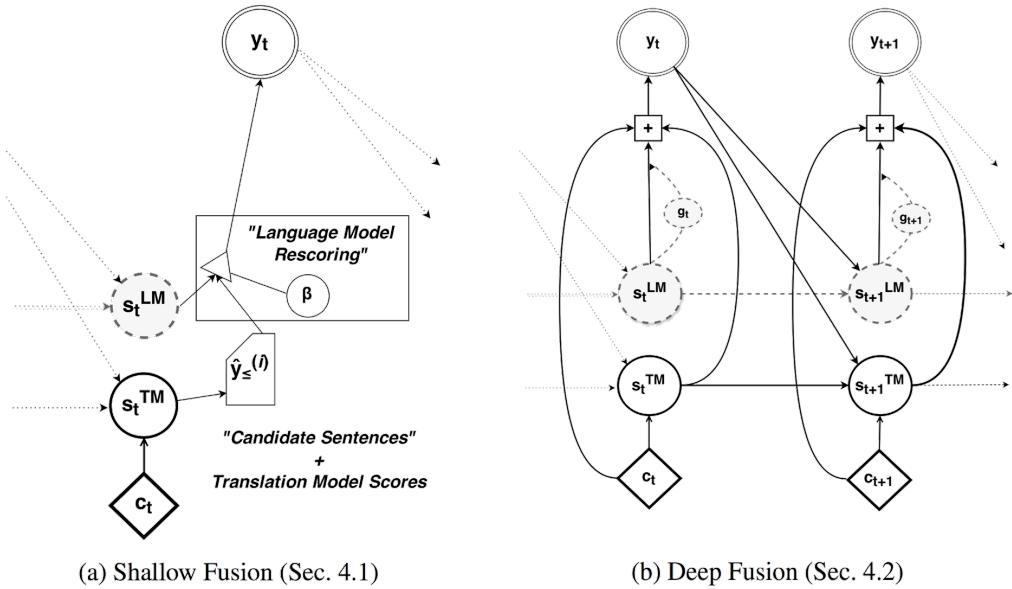


Figure 6: [Gulcehre et al. 2015]

이 방법은 Bengio 교수의 연구실에서 쓴 페이퍼인 [Gulcehre et al. 2015]에서 제안

된 방법입니다. 언어모델을 명시적으로 앙상블하여 디코더의 성능을 올리고자 하였습니다. 두 개의 다른 모델을 쓴 shallow fusion 방법 보다, 언어모델을 Seq2seq에 포함시켜 end2end 학습을 통해 한 개의 모델로 만든 deep fusion 방법이 좀 더 나은 성능을 나타냈습니다. 두 방식 모두 monolingual corpus를 활용하여 언어모델을 학습한 이후 실제 번역기를 훈련시킬 때에는 네트워크 파라미터 값을 고정한 상태로 seq2seq 모델을 훈련합니다.

|                                    | Development Set |              | Test Set     |              |              |              |
|------------------------------------|-----------------|--------------|--------------|--------------|--------------|--------------|
|                                    | dev2010         | tst2010      | tst2011      | tst2012      | tst2013      | Test 2014    |
| <b>Previous Best (Single)</b>      | 15.33           | 17.14        | 18.77        | 18.62        | 18.88        | -            |
| <b>Previous Best (Combination)</b> | -               | 17.34        | 18.83        | 18.93        | 18.70        | -            |
| <b>NMT</b>                         | 14.50           | 18.01        | 18.40        | 18.77        | 19.86        | 18.64        |
| <b>NMT+LM (Shallow)</b>            | 14.44           | 17.99        | 18.48        | 18.80        | 19.87        | 18.66        |
| <b>NMT+LM (Deep)</b>               | <b>15.69</b>    | <b>19.34</b> | <b>20.17</b> | <b>20.23</b> | <b>21.34</b> | <b>20.56</b> |

Figure 7: [Gulcehre et al.2015]

성능상으로는 뒤에 다룰 내용들보다 성능 상의 이득이 적지만, 그 내용이 방법의 장점은 monolingual corpus를 전부 활용 할 수 있다는 것입니다.

## Dummy source sentence translation

아래의 내용들은 전부 Edinburgh 대학의 Nematus 번역시스템에서 제안되고 사용된 내용들입니다. 이 페이퍼[Sennrich et al.2015]의 저자인 Rico Sennrich는 좀 전의 내용처럼 명시적으로 언어모델을 앙상블하는 대신, 디코더로 하여금 monolingual corpus를 학습할 수 있게 하는 방법을 제안하였습니다. 예전 챕터에서 다루었듯이, 디코더는 Conditional Neural Network Language Model이라고 할 수 있는데, source 문장인  $X$ 를 빈 입력을 넣어줌으로써, (그리고 Attention등을 모두 dropout 시켜 끊어줌으로써) condition을 없애는 것이 이 방법의 핵심입니다. 저자는 이 방법을 사용하면 decoder가 monolingual corpus의 language model을 학습하는 것과 같다고 하였습니다.

## Back translation

그리고 같은 [Sennrich et al.2015]에서 좀 더 발전된 다른 방법을 제시하였습니다. 이 방법은 기존의 훈련된 반대 방향 번역기를 사용하여 monolingual corpus를

기계번역하여 합성(synthetic) parallel corpus를 만들어 이것을 훈련에 사용하는 방식입니다. 중요한 점은 기계번역에 의해 만들어진 합성 parallel corpus를 사용할 때, 반대 방향의 번역기의 훈련에 사용한다는 것입니다.

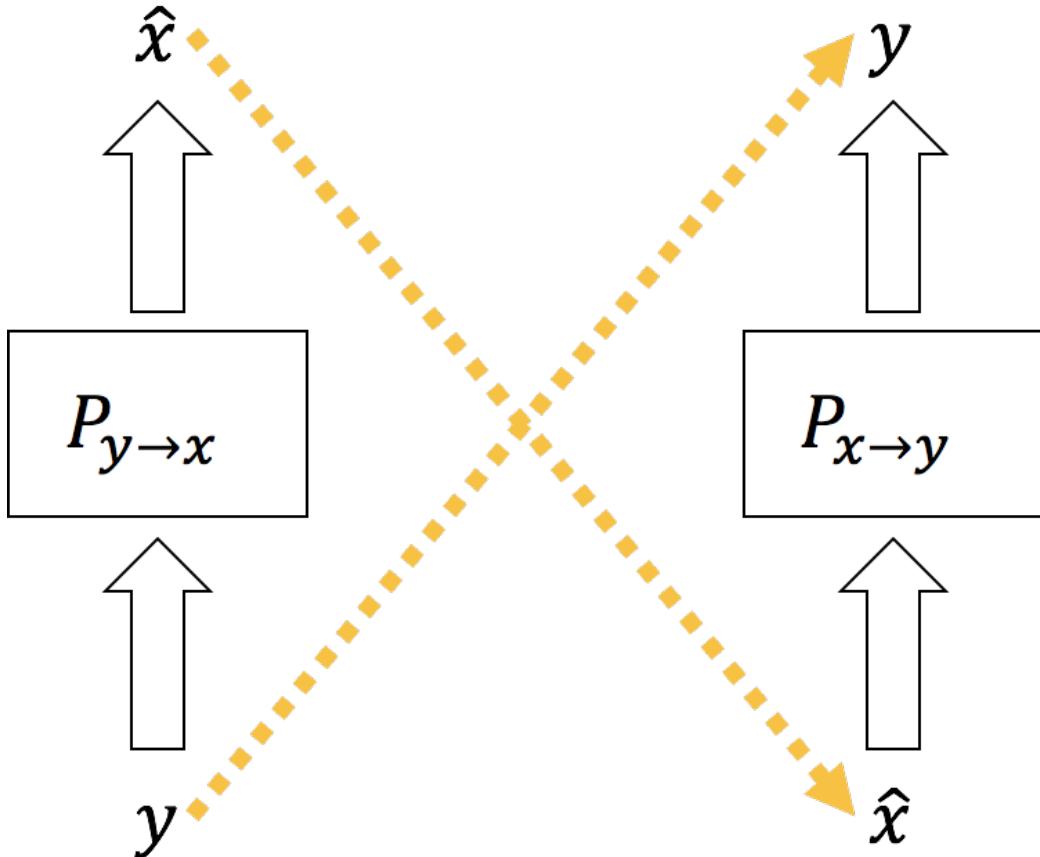


Figure 8: Back Translation 개요

$$\hat{\theta}_e = \underset{\theta}{\operatorname{argmax}} P_{f \rightarrow e}(e | \hat{f}; \theta_e)$$

where  $\hat{f} = \underset{f}{\operatorname{argmax}} P_{e \rightarrow f}(f | e; \theta_f)$

예를 들어, 한국어 monolingual corpus가 있을 때, 이것을 기존에 훈련된 한→영 번역기에 기계번역시켜 한–영 합성 parallel corpus를 만들고, 이것을 영→한

번역기를 훈련시키는데 사용하는 것 입니다. 이러한 방법의 특성 때문에 back translation이라고 명명되었습니다.

| name         | training instances                 | BLEU         |             |              |             |
|--------------|------------------------------------|--------------|-------------|--------------|-------------|
|              |                                    | newstest2014 |             | newstest2015 |             |
|              |                                    | single       | ens-4       | single       | ens-4       |
| parallel     | 37m (parallel)                     | 19.9         | 20.4        | 22.8         | 23.6        |
| +monolingual | 49m (parallel) / 49m (monolingual) | 20.4         | 21.4        | 23.2         | 24.6        |
| +synthetic   | 44m (parallel) / 36m (synthetic)   | <b>22.7</b>  | <b>23.8</b> | <b>25.7</b>  | <b>26.5</b> |

Figure 9: [Sennrich et al.2015]

위의 테이블은 Dummy source translation(=monolingual)과 back translation(=합성) 방식에 대해서 성능을 실험한 결과입니다. 두 가지 방법 모두 사용하였을 때에 성능이 제법 향상된 것을 볼 수 있습니다. Parallel corpus와 거의 같은 양의 corpus가 각각 사용되었습니다. 위에서 언급했듯이, 명시적 언어모델 양상을 방식에서는 코퍼스 사용량의 제한이 없었지만, 이 방식에서는 기존의 parallel corpus와 차이 없이 섞어서 동시에 훈련에 사용하기 때문에, monolingual corpus의 양이 parallel corpus 보다 많아질 경우 주객전도 현상이 일어날 수 있습니다. 따라서 그 양을 제한하여 훈련에 사용하였습니다.

## Copied translation

이 방식은 같은 저자인 Rich Sennrich에 의해서 [Currey et al.2017] Copied Monolingual Data Improves Low-Resource Neural Machine Translation에서 제안 되었습니다. 기존의 Dummy source sentence translation 방식에서 좀 더 나아진 방식입니다. 기존의 방식 대신에 source 쪽과 target 쪽에 똑같은 데이터를 넣어 훈련시키는 것입니다. 기존의 dummy source sentence 방식은 encoder에서 decoder로 가는 길을 훈련 시에 dropout 처리 해주어야 했지만, 이 방식은 그럴 필요가 없어진 것이 장점입니다. 하지만 source 언어의 vocabulary에 target language의 vocabulary가 포함되어야 하는 불필요함을 감수해야 합니다.

## Conclusion

위와 같이 여러 방법들이 제안되었지만, 위의 방법 중에서는 구현 방법의 용이성과 효율성 때문에 back translation이 가장 많이 쓰이는 추세입니다. Back translation 기법은 간단한 방법임에도 불구하고 효과적으로 성능 향상을 얻을 수 있습니다.

| system   | TR→EN |      | EN→TR |      |
|----------|-------|------|-------|------|
|          | 2016  | 2017 | 2016  | 2017 |
| baseline | 20.0  | 19.7 | 13.2  | 14.7 |
| +copied  | 20.2  | 19.7 | 13.8  | 15.6 |

Figure 10: [Sennrich at el.2017]

## Unsupervised Neural Machine Translation

[Artetxe at el.2017]

## Fully Convolutional Seq2seq

Neural Machine Translation의 최강자는 Google이라고 모두가 여기고 있을 때, Facebook이 과감하게 이 논문[Gehring at el.2017]을 들고 도전장을 내밀었습니다. RNN방식의 seq2seq 대신에 오직 convolutional layer만을 이용한 방식의 seq2seq를 들고 나와, 기존의 방식에 대비해서 성능과 속도 두마리 토끼를 모두 잡았다고 주장하였습니다.

### Architecture

#### Position Embedding

이 방식은 RNN을 기반으로 하지 않기 때문에 position embedding을 사용하였습니다. RNN을 사용하면 우리가 직접적으로 위치 정보를 명시하지 않아도 자연스럽게 위치정보가 encoding 되지만, convolutional layer의 경우에는 이것이 없기 때문에 직접 위치 정보를 주어야 하기 때문입니다.

따라서 word embedding vector와 같은 dimension의 position embedding vector를 구하여 매 time-step마다 더해준 뒤, 상위 layer로 feed forward하게 됩니다.

하지만 position embedding이 없다고 이 방식이 동작하지 않는 것은 아닙니다. Position embedding의 유무에 따라서 실험결과 BLEU가 최대 0.5 정도 차이가 나기도

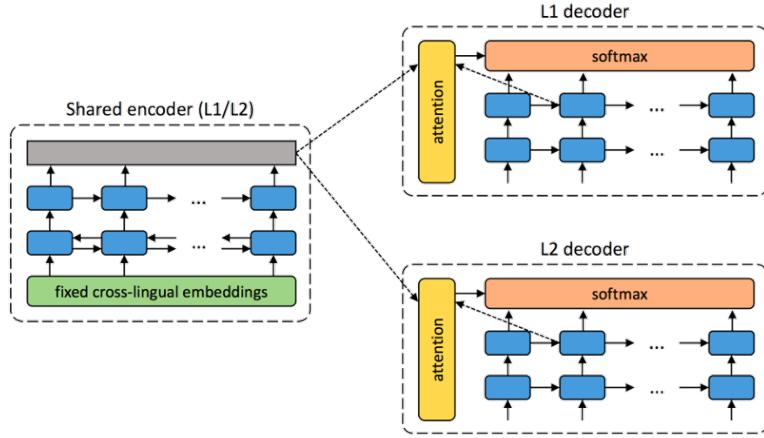


Figure 1: Architecture of the proposed system. For each sentence in language L1, the system is trained alternating two steps: *denoising*, which optimizes the probability of encoding a noised version of the sentence with the shared encoder and reconstructing it with the L1 decoder, and *back-translation*, which translates the sentence in inference mode (encoding it with the shared encoder and decoding it with the L2 decoder) and then optimizes the probability of encoding this translated sentence with the shared encoder and recovering the original sentence with the L1 decoder. Training alternates between sentences in L1 and L2, with analogous steps for the latter.

Figure 11:

Table 1: BLEU scores in newstest2014. Unsupervised systems are trained in the News Crawl monolingual corpus, semi-supervised systems are trained in the News Crawl monolingual corpus and 100,000 sentences from the News Commentary parallel corpus, and supervised systems (provided for comparison) are trained in the full parallel corpus, all from WMT 2014. For GNMT, we report the best single model scores from Wu et al. (2016).

|                        |                                     | <b>FR-EN</b> | <b>EN-FR</b> | <b>DE-EN</b> | <b>EN-DE</b> |
|------------------------|-------------------------------------|--------------|--------------|--------------|--------------|
| <b>Unsupervised</b>    | 1. Baseline (emb. nearest neighbor) | 9.98         | 6.25         | 7.07         | 4.39         |
|                        | 2. Proposed (denoising)             | 7.28         | 5.33         | 3.64         | 2.40         |
|                        | 3. Proposed (+ backtranslation)     | 15.56        | 15.13        | 10.21        | 6.55         |
|                        | 4. Proposed (+ BPE)                 | 15.56        | 14.36        | 10.16        | 6.89         |
| <b>Semi-supervised</b> | 5. Proposed (full) + 100k parallel  | 21.81        | 21.74        | 15.24        | 10.95        |
|                        | 6. Comparable NMT                   | 20.48        | 19.89        | 15.04        | 11.05        |
| <b>Supervised</b>      | 7. GNMT (Wu et al., 2016)           | -            | 38.95        | -            | 24.61        |

Figure 12:

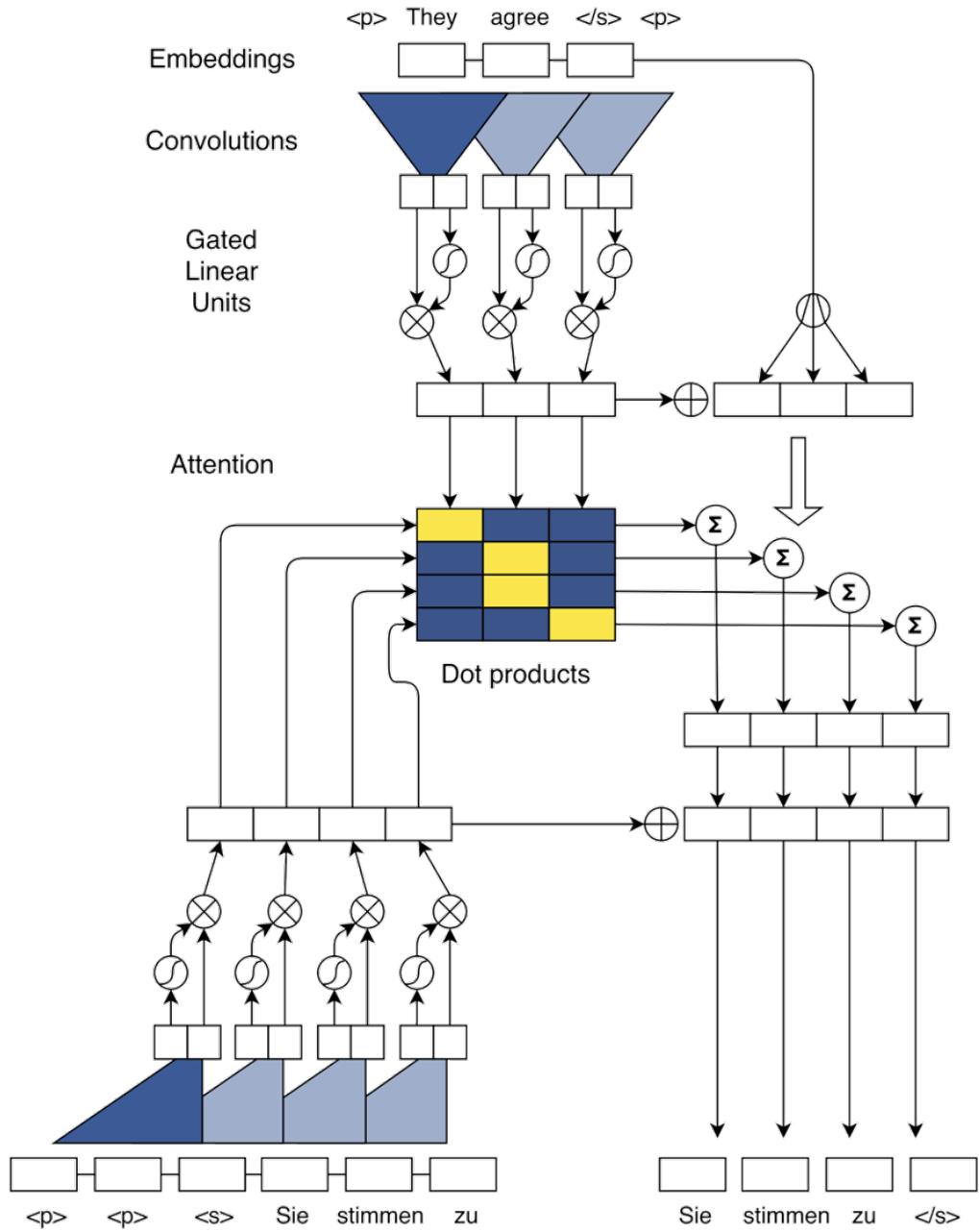


Figure 13: Fully Convolutional Sequence-to-Sequence 아키텍처

합니다.

### Convolutional Layer

Convolutional Layer를 사용한 encoder를 설명하기 이전에, 먼저 [Ranzato et al.2015]에서는 단순히 이전 layer의 결과값을 averaging하는 encoder를 제안하였습니다.

$$e_j = w_j + l_j, \quad z_j = \frac{1}{k} \sum_{t=-\lfloor k/2 \rfloor}^{\lfloor k/2 \rfloor} e_{j+t}$$

where  $w_j$  is word vector and  $l_j$  is position embedding vector.

위와 같이 단순히 평균을 내는 것만으로도 어느정도의 성능을 낼 수 있었습니다. 만약 여기서 convolution filter를 사용하여 averaging 대신에 convolution 연산을 한다면 어떻게 될까요?

위의 물음에서 출발한 것이 이 논문의 핵심입니다. 따라서 kernel(or window) size  $k$ 인 convolution filter가  $d$ 개 channel의 입력을 받아서 convolution 연산을 수행하여  $2d$ 개 channel의 출력을 결과값으로 내놓습니다.

### Gated Linear Unit

이 논문에서는 [Dauphine et al.2016]에서 제안한 Gated Linear Unit(GLU)을 사용하였습니다.

$$v([A; B]) = A \otimes \sigma(B)$$

where  $A \in \mathbb{R}^d$  and  $B \in \mathbb{R}^d$   
thus  $[A; B] \in \mathbb{R}^{2d}$

GLU를 사용하여 직전 convolution layer에서의 결과값인 vector( $\in \mathbb{R}^{2d}$ )를 입력으로 삼아 gate 연산을 수행합니다. 이 연산은 LSTM이나 GRU에서의 gate들과 매우 비슷하게 동작을 수행합니다.

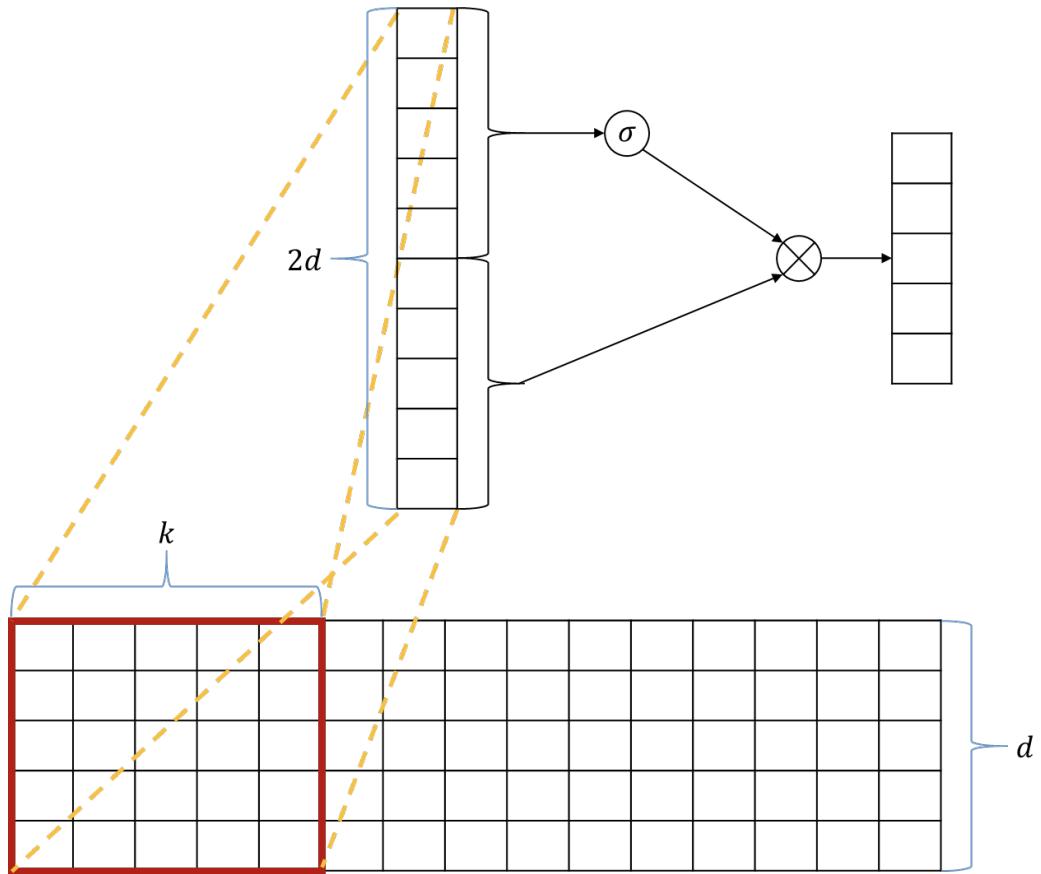


Figure 14: fconv 내에서 연산 과정을 도식화

## Attention

$z^u$ 를 인코더의 출력값,  $h_i^l$ 을 디코더의  $l$ 번째 layer의  $i$ 번째 결과값이라고 하고,  $g_i$ 를  $i - 1$ 번째 디코더의 출력값이라고 할 때, attention의 동작은 아래와 같습니다.

$$\begin{aligned} d_i^l &= W_d^l h_i^l + b_d^l + g_i \\ a_{ij}^l &= \frac{\exp(d_i^l z_j^u)}{\sum_{t=1}^m \exp(d_i^l z_t^u)} \\ c_i^l &= \sum_{j=1}^m a_{ij}^l (z_j^u + e_j) \end{aligned}$$

이렇게 구해진 context vector  $c_i^l$ 을 (기본적인 attention은 concatenate 하였던 것에 비해서) 아래와 같이  $h_i^l$ 에 그냥 더합니다. 그리고 이것을 다음 decoder layer의 입력으로 사용합니다.

$$\tilde{h}_i^l = h_i^l + c_i^l$$

이렇게  $l$ 번째 layer의 attention이 이루어지게 되는데, 이것을 매 layer마다 넣어주게 됩니다. 이전까지의 seq2seq는 attention layer가 전체 구조에서 한 개만 있었던 것에 비해서 Facebook이 제안한 이 구조에서는 모든 layer마다 나타날 수 있습니다. # Transformer (Attention is All You Need)

Facebook에서 CNN을 활용한 번역기에 대한 논문을 내며, 기존의 GNMT 보다 속도나 성능면에서 뛰어남을 자랑하자, 이에 질세라 Google에서 바로 곧이어 발표한 Attention is all you need [Vaswani et al. 2017] 논문입니다. 실제로 ArXiv에 Facebook이 5월에 해당 논문을 발표한데 이어서 6월에 이 논문이 발표되었습니다. 이 논문에서 Google은 아직까지 번역에 있어서 자신들의 기술력 우위성을 주장하였습니다.

## Architecture

“Attention is all you need”라는 제목의 논문답게 이 논문은 정말로 Attention만 구현해서 모든것을 해냅니다. 그리고 저자는 이 모델 구조를 Transformer라고 이름 붙였습니다. Encoder와 decoder를 설명하기에 앞서, sub-module부터 소개하겠습니다. Encoder와 decoder를 이루고 있는 sub-module은 크게 3가지로 나뉘어 집니다.

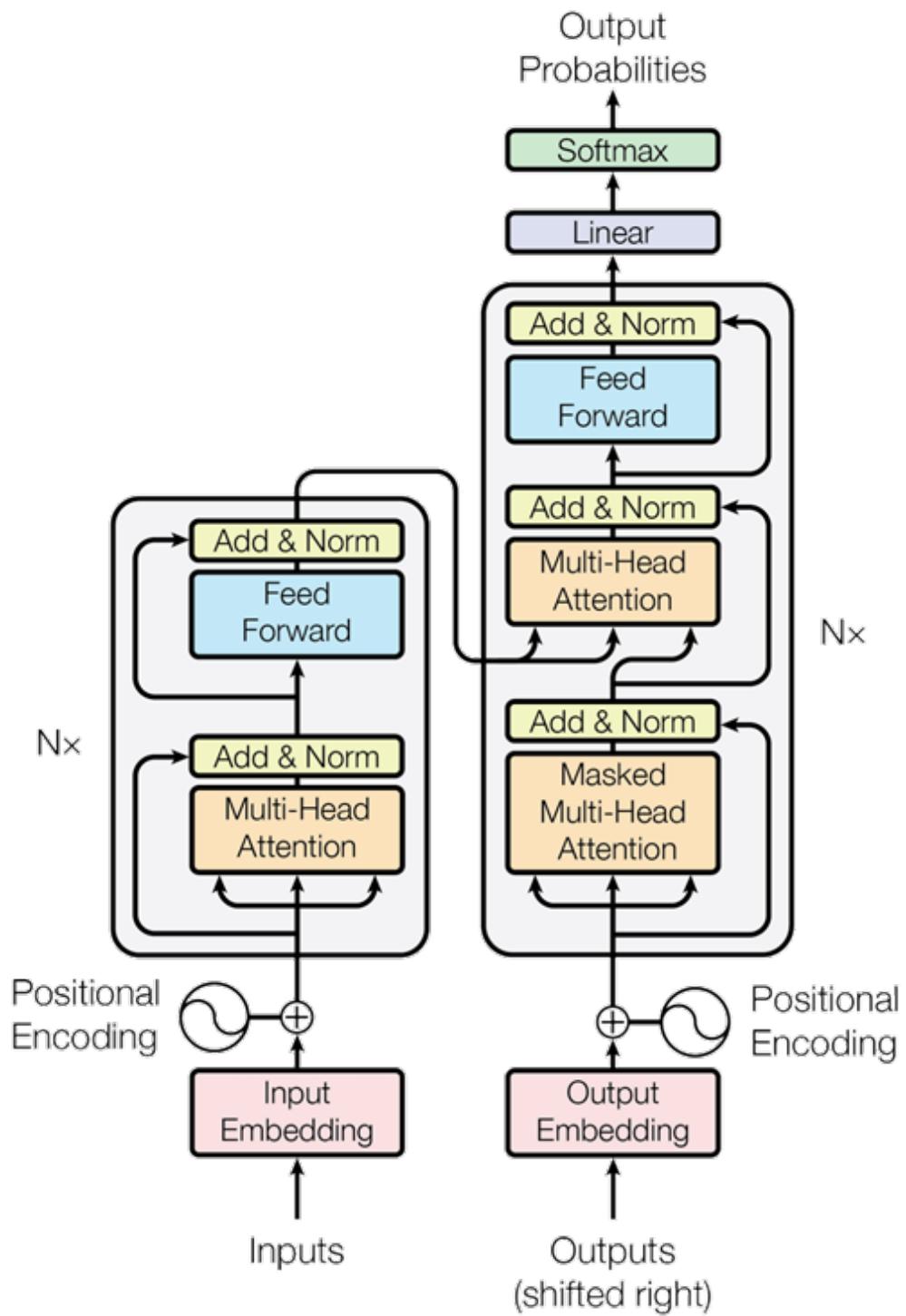


Figure 15: Transformer 아키텍처

---

| 명칭                 | 역할   |
|--------------------|--|
| Self-attention     | 이전 layer의 output에 대해서 attention을 수행합니다.                |
| Attention          | Encoder의 output에 대해서 기존의 seq2seq와 같이 attention을 수행합니다. |
| Feed Forward Layer | attention layer을 거쳐 얻은 결과물을 최종적으로 정리합니다.               |

---

Encoder는 다수의 self-attention layer와 feed forward layer로 이루어져 있습니다. Decoder는 다수의 self-attention과 attention이 번갈아 나타나고, feed forward layer가 있습니다. 이처럼 Transformer는 구성되며 각 모듈에 대한 자세한 설명은 아래와 같습니다.

### Position Embedding

이전 Facebook 논문과 마찬가지로, RNN을 이용하지 않기 때문에, 위치정보를 단어와 함께 주는 것이 필요합니다. 따라서 Google에서도 마찬가지로 position embedding을 통해서 위치 정보를 나타내고자 하였으며, 그 수식은 약간 다릅니다.

$$\begin{aligned} \text{PE}(\text{pos}, 2i) &= \sin(\text{pos}/10000^{2i/d_{\text{model}}}) \\ \text{PE}(\text{pos}, 2i + 1) &= \cos(\text{pos}/10000^{2i/d_{\text{model}}}) \end{aligned}$$

Position embedding의 결과값의 dimension은 word embedding의 dimension과 같으며, 두 값을 더하여 encoder 또는 decoder의 입력으로 넘겨주게 됩니다.

### Attention

이 논문에서의 Attention 방식은 여러개의 attention으로 구성된 multi-head attention을 제안합니다. 마치 Convolution layer에서 여러개의 filter가 있어서 여러가지 다양한 feature를 뽑아 내는 것과 같은 원리라고 볼 수 있습니다.

기본적인 attention의 수식은 아래와 같습니다. 기본적인 attention은 원래 그냥 dot-product attention인데 scaled라는 이름이 붙은 이유는 key의 dimension인  $\sqrt{d_k}$ 로 나누어주었기 때문입니다. 이외에는 이전 섹션에서 다루었던 attention과 같습니다.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

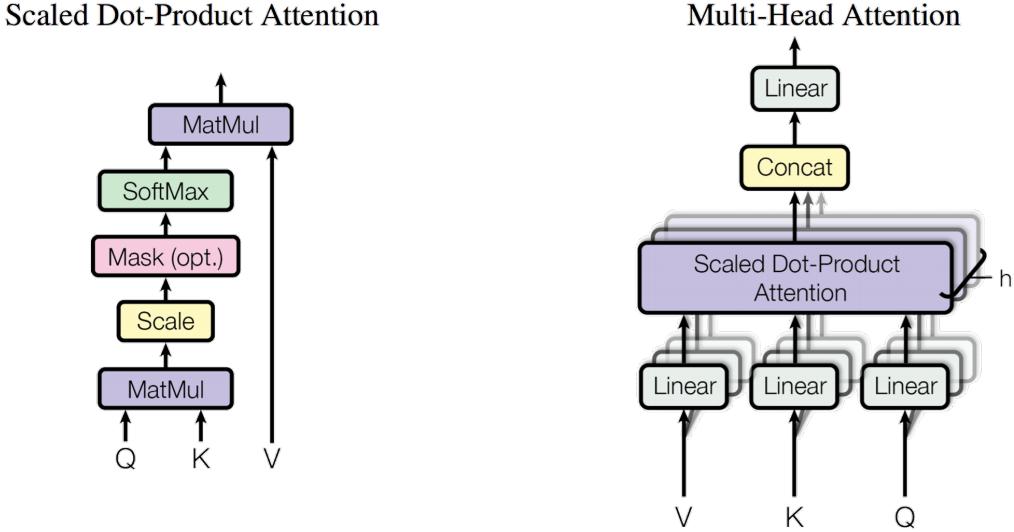


Figure 16: Transformer의 Attention 구성

이렇게 구성된 attention을 하나의 head로 삼아 Multi-Head Attention을 구성합니다.

$$\text{MultiHead}(Q, K, V) = [\text{head}_1; \text{head}_2; \dots; \text{head}_h]W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

where  $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  
 $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$  and  $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

$$d_k = d_v = d_{\text{model}}/h = 64$$

$$h = 8 \text{ and } d_{\text{model}} = 512$$

이때에 각 head의 Q, K, V 마다 다른 W를 곱해줌으로써 각각 linear transformation 형태를 취해 줍니다. 즉, head마다 필요한 다른 정보(feature)를 attention을 통해 encoding 할 수 있게 됩니다. 해당 논문에서는 hidden size를 512로 하고 이를 8개의 head로 나누어 각 head의 hidden size는 64가 되도록 하였습니다.

실제 구현을 할 때에는 self attention의 경우에는 이전 layer의 출력값이 모두 Q, K, V를 이루게 됩니다. 같은 값이 Q, K, V로 들어가지만 linear transform을 해주기 때문에 상관이 없습니다. Decoder에서 수행하는 encoder에 대한 attention을 할

때에는, Q는 decoder의 이전 layer의 출력값이 되지만, K, V는 encoder의 출력값이 됩니다.

### Self Attention for Decoder

Decoder의 self-attention은 encoder의 그것과 조금 다릅니다. 이전 레이어의 출력값을 가지고 Q, K, V를 구성하는 것은 같지만, 약간의 제약이 더해졌습니다. 그 이유는 inference 할 때, 다음 time-step의 값을 알 수 없기 때문입니다. 따라서, self-attention을 하더라도 이전 time-step에 대해서만 접근이 가능하도록 해야 합니다. 이를 구현하기 위해서 scaled dot-product attention 계산을 할 때에 masking을 추가하여, 미래의 time-step에 대해서는 weight를 가질 수 없도록 하였습니다.

### Position-wise Feed Forward Layer

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \text{ where } d_{ff} = 2048$$

사실 여기에서 소개한 이 layer는 기존의 fully connected feed forward layer라기보단, kernel size가 1인 convolutional layer라고 볼 수 있습니다. Channel숫자가  $512 \rightarrow 2048$  으로 가는 convolution과,  $2048 \rightarrow 512$ 로 가는 convolution으로 이루어져 있는 것입니다.

### Evaluation

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model                           | BLEU        |              | Training Cost (FLOPs)                 |                     |
|---------------------------------|-------------|--------------|---------------------------------------|---------------------|
|                                 | EN-DE       | EN-FR        | EN-DE                                 | EN-FR               |
| ByteNet [18]                    | 23.75       |              |                                       |                     |
| Deep-Att + PosUnk [39]          |             | 39.2         |                                       | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38]                  | 24.6        | 39.92        | $2.3 \cdot 10^{19}$                   | $1.4 \cdot 10^{20}$ |
| ConvS2S [9]                     | 25.16       | 40.46        | $9.6 \cdot 10^{18}$                   | $1.5 \cdot 10^{20}$ |
| MoE [32]                        | 26.03       | 40.56        | $2.0 \cdot 10^{19}$                   | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] |             | 40.4         |                                       | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38]         | 26.30       | 41.16        | $1.8 \cdot 10^{20}$                   | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9]            | 26.36       | <b>41.29</b> | $7.7 \cdot 10^{19}$                   | $1.2 \cdot 10^{21}$ |
| Transformer (base model)        | 27.3        | 38.1         | <b><math>3.3 \cdot 10^{18}</math></b> |                     |
| Transformer (big)               | <b>28.4</b> | <b>41.8</b>  | $2.3 \cdot 10^{19}$                   |                     |

Figure 17: Transformer의 성능 비교

Google은 transformer를 통해서 State of the Art의 성능을 달성했다고 보고하였습니다. 뿐만 아니라, 기존의 RNN 및 Facebook의 ConvS2S보다 훨씬 빠른 속도로 훈련이 가능하다고 하였습니다. 실제로 위의 table을 보면, transformer의 training cost의 magnitude는  $10^{18}$ 으로, 대부분의 다른 방식  $10^{19}$ 와 급격한 차이를 보이는 것을 알 수 있습니다.

또 하나의 속도 개선의 원인은 input feeding의 부재입니다. RNN기반의 방식은 input feeding이 도입되면서 decoder를 훈련할 때 모든 time-step을 한번에 할 수 없게 되었습니다. 이로 인해서 대부분의 병목이 decoder에서 발생합니다. 하지만 transformer는 input feeding이 없기 때문에 한번에 모든 time-step에 대해서 계산할 수 있게 되었습니다.

비록 transformer가 최고 성능을 달성하기 했지만 그 모델 구조의 과격함 때문인지 (Facebook의 모델과 함께) 아직 주류로 편입되지 않았습니다. 아직 대부분의 논문들은 이 구조를 비교대상으로 논하기보다, RNN구조의 seq2seq를 대상으로 실험을 비교/진행 하곤 합니다.

## 읽을거리