다음으로 본론으로 넘어가서, 모델구조와 학습과정, 그리고 결과값까지 설명하도록 하겠습니다.

우선 트랜스포머 모델의 인코더는 6개의 동일한 레이어로 구성되어 있으며, 각 레이어는 multi-head attention과 feedforward라는 두개의 서브 레이어를 가집니다. 두 서브 레이어 주위에 각각 잔여연결을 해준 후 정규화가 진행되는데, 이때 잔여연결이란 input을 output으로 그대로 전달하는 것을 뜻합니다.

디코더 역시 6개의 동일한 레이어로 구성되어 있으나, 디코더는 세개의 서브 레이어를 가집니다. 인코더의 결과에 multi-head attention을 수행할 서브 레이어를 하나 더 추가해준 것입니다. 마찬가지로 서브 레이어에 잔여연결을 사용한 뒤, 레이어 정규화를 해줍니다.

다코더에서는 인코더와 달리, 순차적으로 결과를 만들어내야 하기 때문에, masking을 해줍니다. Masking을 통해 position i보다 이후에 있는 position에 attention을 주지 못하게 하는데, 이는 position i에 대한 예측은 미리 알고 있는 output에만 의존을 하도록 하는 것입니다. **/**

다음으로 볼 것은 scaled dot-product attention입니다. 먼저 input은 쿼리, 키, value로 구성됩니다.

이때, 모든 query와 key에 대한 내적을 계산하고, 각 키를 나눠준 뒤, softmax함수를 적용함으로써 가중치를 얻을 수 있습니다. key와 value는 attention이 이루어지는 위치에 상관없이 같은 값을 갖게 됩니다. 이때 query와 key에 대한 dot-product를 계산하면 각각의 query와 key 사이의 유사도를 구할 수 있게 됩니다. softmax를 거친 값을 value에 곱해주면, query와 유사한 value일수록, 즉 중요한 value일수록 더 높은 값을 가지게 됩니다. 결론적으로, 중요한 정보에 더 관심을 둔다는 attention의 원리에 알맞는 것입니다. **/**

Key, query, value로 한번의 attention을 수행하는 것보다, 여러 번의 attention을 병렬로 수행하는 것이 더 효과적이라고 합니다. 즉, multi-head attention은 Scaled dot-product attention을 여러 개 만들어 다양한 특징에 대한 attention을 볼 수 있게 한 방법입니다. 여기서는, 동일한 Q,K,V를 헤드 수 만큼 나누어 병렬적으로 계산합니다.

트랜스포머 모델은 인코더 디코더 언텐션 레이어, 인코더의 self-attention layer 그리고, 자동회귀속성을 방지하기 위한 경우에 multi-head attention을 사용합니다. . **/**

트랜스포머 모델의 인코더와 디코더에서 각각의 레이어는 fully connected feed-forward network를 포함하고 있습니다. 각 포지션마다 동일하게 적용되는 이 네트워크는 두개의 선형 transformation과 정류선형유닛으로 구성되어 있습니다.

이 과정을 보시면, 우선x에 linear transformation을 적용한 뒤, 정류선형유닛을 거쳐 다시 한번 더 linear transformation 적용하는 것을 알 수 있습니다. 각 포지션마다 같은 파라미터 W, b를 사용하지만, 레이어가 달라지면 다른 파라미터 사용합니다. 이 과정은 커널 사이즈가 1이고 channel이 layer인 convolution을 두 번 수행한 것으로도 이해할 수 있습니다.  **/**

embedding 값을 고정시키지 않고, 학습을 하면서 embedding값이 변경되는 learned embedding을 사용했습니다. 이때 input과 output은 같은 embedding layer를 사용합니다. 또한 decoder output을 다음 token의 확률로 바꾸기 위해 learned linear transformation과 softmax function을 사용했습니다.

트랜스포머 모델은 반복모델도 아니고 convolution도 아니기 때문에, 단어의 시퀀스를 이용하기 위해서는 단어의 포지션에 대한 정보를 추가해줄 필요가 있었습니다. 따라서 인코더와 디코더의 input embedding에positional encoding을 더해주었습니다. **/**

다른 레이어와 비교하여 레이어당 전체 계산 복잡도와 병렬 연산가능 계산량, 그리고 네트워크에서 장거리 종속성 사이의 경로 길이를 고려해보았을 때, self attention 레이어를 사용하는 것이 가장 효율적임을 알 수 있었습니다. **/**

Transformer 모델에 대한 학습은 영어-독일어, 영어-프랑스어라는 대표적인 두가지 데이터셋을 설정해두고 진행하였습니다. WMT 영독 데이터셋은 약 450만개의 문장쌍 그리고 37,000개의 source 토큰과 target토큰으로 이루어져 있으며, WMT 영프 데이터셋의 경우엔 약 3,600만개의 문장쌍이 32,000개의 단어 토큰으로 나누어져 있습니다. 학습을 위한 하드웨어로는 8개의 엔비디아 P100 GPU를 사용하였고, base모델과 big모델 모두 매우 빠른 학습시간을 보여주었습니다. **/**

다음으로 최적화 과정에서는 Adam optimizer를 사용하였습니다.

학습률을 training동안 고정하지 않고 다음의 공식에 맞춰 변화시킵니다. 공식에 따르면, 첫 번째 warmup\_steps까지는 학습 속도를 선형적으로 증가시키고, 그 이후에는 step number의 -0.5배 비율로 감소시킵니다. 이렇게 하는 이유는 처음에는 학습이 잘 되지 않은 상태이므로 학습률을 빠르게 증가시켜 변화를 크게 주다가, 학습이 꽤 됐을 시점에는 이를 천천히 감소시켜 변화를 작게 주기 위해서 입니다. **/**

정규화에서는 residual dropout과 label smoothing의 두가지 방법을 사용하였습니다. 우선 Residual dropout에서 dropout은 training data에 overfitting되는 문제를 어느정도 막아줍니다. dropout된 unit들은 training되지 않는 것이니 training data에 값이 조정되지 않기 때문입니다. 따라서 정규화 효과를 위해 residual connection을 수행할 때 dropout을 같이 수행해줍니다. 다음 식을 보시면, 각 sublayer의 아웃풋에 드롭 아웃을 적용한 후, sublayer input과 더해 정규화 함을 알 수 있습니다. 인코더와 디코더에서 embedding에 position 인코딩을 더한 결과에 대해서도 dropout을 적용했습니다.

다음으로 label smoothing입니다. Label smoothing은 잘 알려진 정규화 기법으로, 이미지 분류 등에서도 이미 많이 사용되고 있는 기법 중 하나인데, Label 값을 넣어줄 때 smoothing기법까지 적용하게 해, 모델이 특정 출력 값에 대해서 확신하지 않도록 함으로써 정규화 효과를 더할 수 있습니다. = 0.1인 label smoothing의 도입으로 accuracy와 BLEU score를 높일 수 있었습니다. **/**

결론입니다. 이 표는 최근 모델과 transformer model의 성능 비교 결과를 정리해 놓은 표입니다. 이 표에서 transformer 시스템의 실제 성능을 확인할 수 있습니다. 우선 표를 보시면, 기본적인 base model만 가지고도 기존의 최신 네트워크와 필적하는 좋은 성능을 낸다는 것을 알 수 있었으며, 심지어 학습시간은 다른 모델과 비교해 훨씬 짧았습니다.

또한 big model을 사용했을 때에도 이전 연구와 비교해 학습 효율이 높았으며<FLOPs>, 28.4라는 새로운 최신 BLEU 점수를 확립하며 훨씬 개선된 성능을 보여주었습니다. 결과적으로, 이 실험 결과를 통해 transformer는 영어를 독일어와 프랑스어로 번역하는 항목에서 학습속도와 성능 모두 좋았다는 것을 확인할 수 있습니다. **/**

다음으로, Transformer 시스템 구성에서 어떤 요소가 상대적으로 중요한지에 대한 내용을 확인하기 위해서 model variation 실험을 진행했습니다. 간단하게 헤드 수를 줄여보거나, 특정 파라미터의 수를 변화시키며 구성 요소의 성능을 확인해봤습니다. 첫번째로, 괄호A부분은 헤드의 수를 바꿔가면서 실험한 결과입니다. 다른 요소는 바꾸지 않은 상태에서 attention head의 개수와 attention key, 그리고 value의 크기를 다양하게 설정하였습니다. 결과적으로, 너무 많은 head를 사용하는 것은 오히려 성능을 떨어뜨렸고, 헤드를8개 사용했을 때 성능이 가장 좋은 것을 확인할 수 있었습니다.

B부분은 헤드와 별도로 key 값의 크기를 줄여본 결과입니다. 확인 결과, 당연히 파라미터의 수가 줄어 모델의 품질이 저하되었습니다. 실제로 BLEU결과를 보면 성능이 더 낮아진 것을 확인할 수 있습니다. 결국 key값을 줄이는 것은 모델의 성능을 훼손시키기 때문에 64를 사용한 경우가 16이나 32를 사용한 경우보다 더 좋은 결과값을 보여줍니다.

(C)는 모델의 크기를 키운 경우의 결과 값을 보여주는데, 예상대로 모델의 크기를 키웠을 때 성능이 더 좋아졌습니다. BLEU 값을 보면 d\_model을1024로 높이거나 d\_ff를 4096으로 높였을 때 성능이 좋아진다는 것을 확인할 수 있습니다.

(D)에서는 dropout 기법의 효과를 보여주는데, 결과를 보면 dropout 기법을 사용했을 때 성능이 향상되고, over fitting을 막아주는 데에 매우 효과적인 것을 알 수 있습니다.

마지막으로 E에서는 사인곡선 포지션 encoding 대신 학습된 포지션 embedding을 사용하는 것이 base model과 비교해 성능의 차이가 별로 없다는 것을 확인할 수 있습니다. **/**

transformer 모델은 기계번역 뿐 아니라 다양한 자연어 처리 과정에서 사용할 수 있습니다. 화면의 표는 트랜스포머가 다른 작업을 처리할 수 있는지를 평가하기 위해 영어구문 분석 실험을 진행해본 결과입니다.

네번째 91.7의 경우를 제외한 나머지 최신 네트워크와 비교해보았을 때, transformer의 성능이 더좋은 것을 확인할 수 있습니다. Semi-supervised에서도 마찬가지로, 트랜스포머 모델에서 더 좋은 성능이 나오는 것을 확인하였습니다.