Attention Is All You Need.

Abstract

1. Introduction

2. Background

3. Model Architecture

3.1 Encoder & Decoder Stacks

3.2 Attention

3.2.1 Scaled Dot-Product Attention

3.2.2 Multi-Head Attention

3.2.3 Applications of Attention in our Model

3.3 Position-wise Feed-Forward Networks

3.4 Embeddings and Softmax

3.5 Positional Encoding

4. Why Self-Attention

5. Training

5.1 Training Data and Batching

5.2 Hardware and Schedule

5.3 Optimizer

5.4 Regularization

6 Results

6.1 Machine Translation

6.2 Model Variations

6.3 English Contituency Parsing

7. Conclusion

Abstract

- Simple network architecture ··· Transformer
- based solely on attention mechanisms!
- Experiments (machine translation):
  - superior in quality
  - more parallelizable
  - require significantly less time to train
- WMT 2014, best BLEU
- generalizes well!
  - *1 Constituency parsing

1. Introduction

Recurrent models

- typically factor Computation along the symbol position of the input and output sequences.
- Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as
  - a function of previous hidden state h_{t-1}
  - input for position t.

↓

preclude parallelization within training examples, critical at longer sequence lengths!

(improvements : factorization tricks & conditional computation)

- Attention mechanism
  : allow modeling of dependencies without regard to their distance in the input or output sentences.

Transformer

- rely entirely on an attention mechanism to draw global dependencies between input & output
- more parallelization
- SOTA in translation quality
- 12 hrs training with 8 P100 GPUs.

2. Background

- to reduce sequential computation ; CNN
  - Extented Neural GPU
  - ByteNet
  - ConvS2S

↓

the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions.

- linearly for ConvS2S
- logarithmically for ByteNet

★ Transformer ★

- reduced to a constant number of operations,
- reduced effective resolution due to averaging attention-weighted positions

↓

- counteract with Multi-Head Attention
- Self-Attention (= intra - attention)

  : attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

- 1st transduction model relying entirely on self-attention to compute representations of its input & output without using sequence-aligned RNN or convolution.

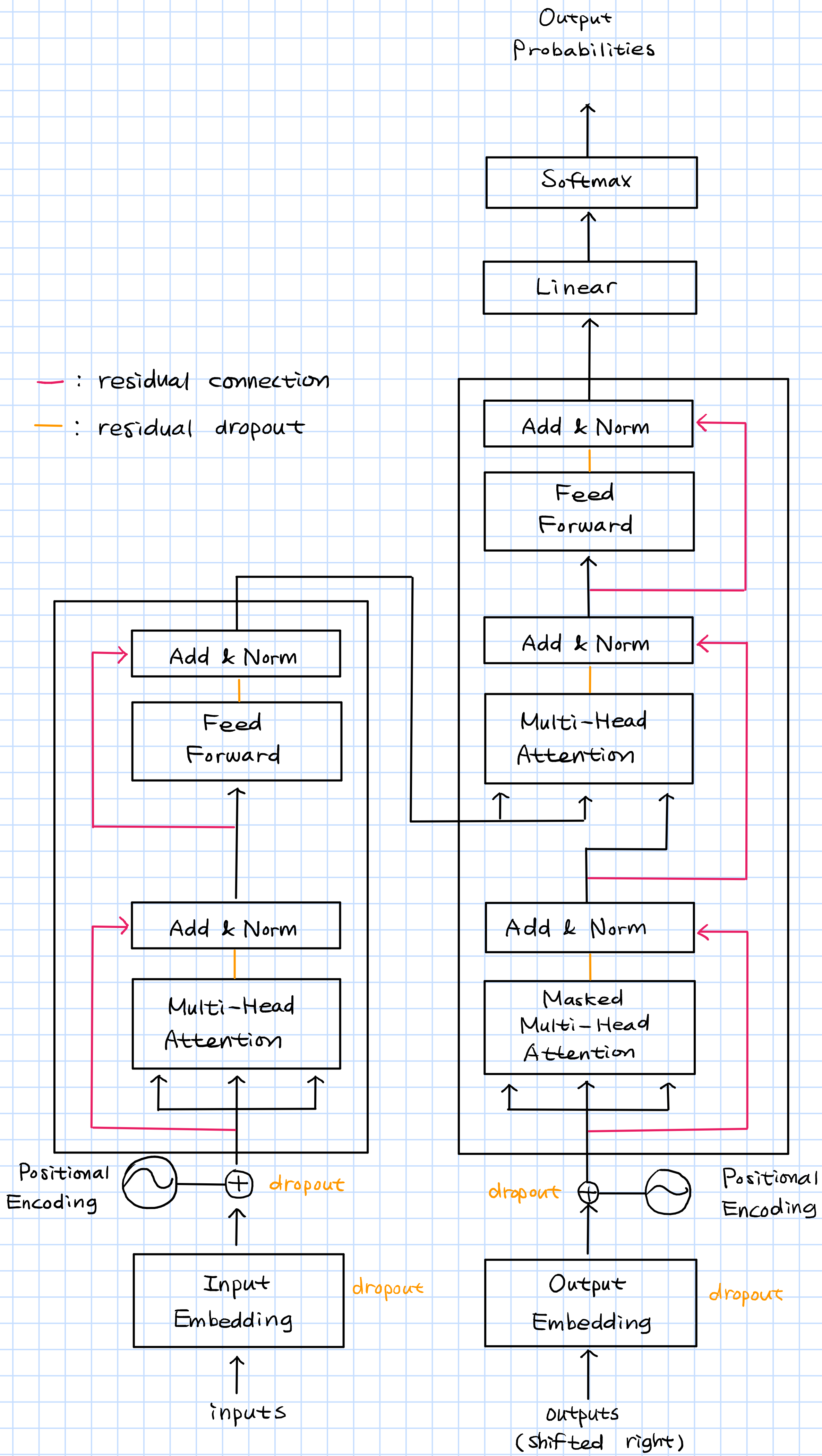
3. Model Architecture

- Encoder - decoder structure
- Encoder : maps an input sequence of symbol representations $(x_1, \ldots, x_n)$
  to a sequence of continuous representations $z = (z_1, \ldots, z_n)$

↓

- Decoder : generates an output sequence $(y_1, \ldots, y_m)$ of symbols
  one element at a time.
- auto - regressive at each step.
  (consuming previously generated symbols as additional input
  when generating the next)

- stacked self - attention
- point - wise, fully connected layers for both encoder & decoder



Output
Probabilities
Softmax
Linear
: residual connection
: residual dropout
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
dropout
dropout
Positional
Encoding
Input
Embedding
dropout
Output
Embedding
dropout
inputs
outputs
(shifted right)

3.1 Encoder & Decoder Stacks

- Encoder

  - stack of $N = 6$ identical layers
  - 1st : multi-head self-attention mechanism
  - 2nd : position-wise fully connected feed-forward network
  - residual connection around each of the two sub-layers, followed by layer normalization.
  - the output of each sub-layer is LayerNorm $(x + \text{Sublayer}(x))$

    function implemented by the sub-layer itself.
  - $d_{model} = 512$

- Decoder

  - stack of $N = 6$ identical layers
  - masking : prevent positions from attending to subsequent positions.

3.2 Attention

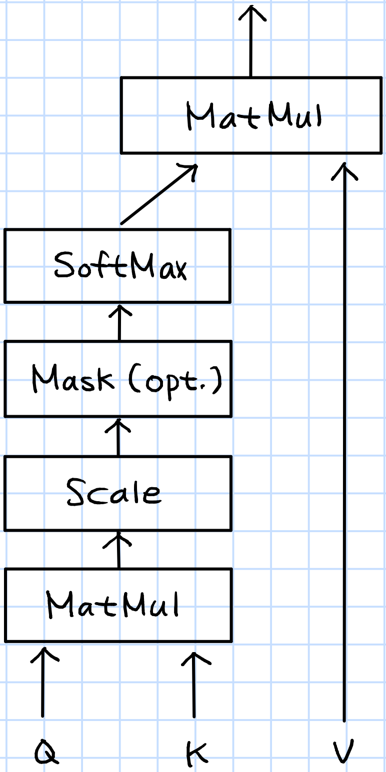
- mapping a query & a set of key-value pairs to an output
  (Q, K, V, output are all vectors)
- output: weighted sum of the values
- the weight assigned to each value is computed by
  a compatibility function of the query with the corresponding key.

3.2.1 Scaled Dot-Product Attention

apply a softmax function to obtain the weights on the values

divide each by $\sqrt{d_k}$

compute the dot product of the query with all keys



- input: queries, keys of dimension d_k, values of dimension d_v
- In practice, compute attention function simultaneously,
  packed together into a matrix Q, K (keys), V (Values)
- $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

2 most commonly used attention functions:

① additive attention

② dot-product (multiplicative) attention

① additive attention

- computes compatibility function using a feed-forward network with a single hidden layer.

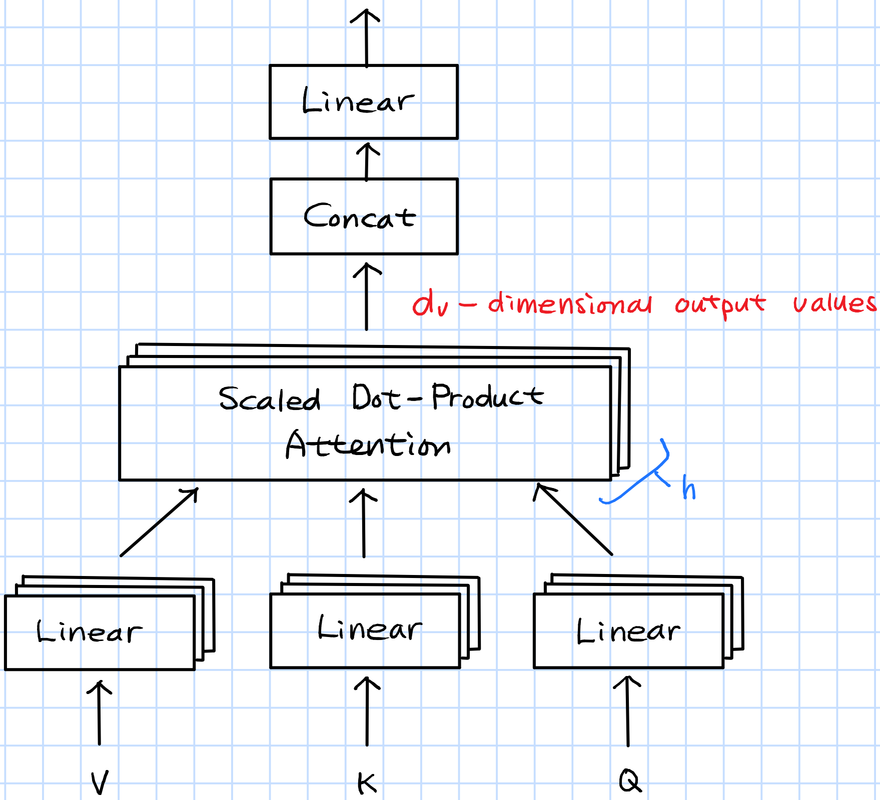
② dot-product attention

- much faster, space-efficient in practice
- for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

$\downarrow$

scale the dot product by $\frac{1}{\sqrt{d_k}}$

3.2.2 Multi-Head Attention



- linearly project the queries, keys, and values h times with different, learned linear projections to d_k, d_k, d_v dimensions, respectively.
- allows models to jointly attend to information from different representation subspace at different positions.

- $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W^O$

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

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

- $h = 8$ parallel attention layers, or heads.
- $d_k = d_v = d_{model} / h = 64$
- Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Application of Attention in our Model

- Transformer uses multi-head attention in 3 ways:

① In encoder – decoder attention layers

  - Queries: come from the previous decoder layer.
  - Memory keys & values: come from the output of the encoder.
  - allows every position in the decoder to attend over all positions in the input sequence.

② Self – attention layers in encoder

  - In a self – attention layer, all of the keys, values, and queries come from the output of the previous layer in the encoder.
  - Each position in the encoder can attend to all positions in the previous layer of the encoder.

③ Self – attention layers in decoder

  - allow each position in the decoder to attend to all positions in the decoder up to & including that position
  - need to prevent leftward information flow in the decoder to preserve the auto – regressive property.

    ↓

  - inside of scaled dot – product attention by masking out all values in the input of the softmax which correspond to illegal connections.

3.3 Position-wise Feed-Forward Networks

- Each of the layers in the encoder & decoder
  contains a fully connected feed-forward network
- consists of 2 linear transformations with a ReLU activation in between

- $FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$

- Linear transformations : same across different positions
  use different parameters from layer to layer.

- Dimensionality :
  - input & output : $d_{model} = 512$
  - inner-layer : $d_{ff} = 2048$

3.4 Embeddings & Softmax

- Embedding : to convert the input tokens & output tokens
  to vectors of dimension d_{model}
- learned linear transformation & softmax function
  : to convert the decoder output to predict next-token probabilities.

- share the same weight matrix between 2 embedding layers &
  the pre-softmax linear transformation
- In the embedding layers, those weights are multiplied by $\sqrt{d_{model}}$

3.5 Positional Encoding

$$PE_{(pos, 2i)} = \sin(pos / 10000^{2i/d_{model}})$$

$$PE_{(\underbrace{pos}_{\text{position}}, \underbrace{2i+1}_{\text{dimension}})} = \cos(pos / 10000^{2i/d_{model}})$$

- each dimension of the positional encoding corresponds to a sinusoid.
- The wavelengths form a geometric progress from 2π to $10000 \cdot 2\pi$

4. Why Self - Attention

- 3 desiderata of self-attention
  1. Total computational complexity per layer
  2. Amount of computation that can be parallelized
  3. Path length between long-range dependencies in the network.
     - key factor affecting the ability to learn such dependencies
       : the length of the paths forward & backward signals have to traverse!
     - the shorter these paths between any combination of positions in the input & output sequences, the easier it is to learn long-range dependencies.

- self-attention layer : connects all positions with a constant number of sequentially executed operations
- recurrent layer : requires $O(n)$ sequential operations.
- In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d

- single convolutional layer with kernel width $k < n$: does not connect all pairs of input & output positions (too expensive!)
- Saparable convolutions : decreases the complexity considerably.
  $O(k \cdot n \cdot d + n \cdot d^2)$

5. Training

5.1 Training Data & Batching

- standard WMT 2014 English-German dataset
  - 4.5M sentence pairs
  - byte-pair encoding
  - shared source-target vocabulary of 37,000 tokens.
- larger WMT 2014 English-French dataset
  - 36 M sentences
  - 32,000 word-piece vocabulary
- Sentence pairs : batched together
  - Each training batch : sentence pairs containing
    25,000 source tokens & 25,000 target tokens.

5.2 Hardware and Schedule

- 8 NVDIA P100 GPUs
- basic model : 100,000 steps (0.4 seconds per step), 12hrs
- big model : 300,000 steps (1.0 seconds per step), 3.5 days

5.3. Optimizer

- *2 Adam optimizer
- $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\epsilon = 10^{-9}$
- varied learning rate according to the formula :

$$lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

  - increasing learning rate linearly for the first warmup_steps
  - decreasing it thereafter to the inverse square root of step number.
- warmup_steps = 4000.

5.4. Regularization

3 types of regularization :

① Residual Dropout

- apply dropout to the output of each sub-layer.
  (before it is added to the sub-layer input & normalized)
- apply dropout to the sums of the embeddings & the positional encodings in both the encoder & decoder stacks.
- base model : $P_{drop} = 0.1$

② Label Smoothing

- $\epsilon_{ls} = 0.1$
- This hurts [*3] perplexity (as the model learns to be more ensure)
- but improves accuracy & BLEU score.

6. Results

6.1 Machine Translation

- Base model : 5 checkpoints
- Big model
  - 20 check points
  - used *beam search with a beam size of 4 & length penalty $\alpha = 0.6$
- hyperparameters were chosen after experimentation on the development set.
- number of floating point operations :

  : training time * number of GPUs used *

  estimate of the sustained single-precision floating-point capacity of each GPU

6.2. Model Variations

- To evaluate the importance of different components of the Transformer

  ⋮

6.3. English Constituency Parsing

- To evaluate if the Transformer can generalize to other tasks
- Challenges :
  - the output is subject to strong structural constraints
  - the output is significantly longer than the input.

- 4-layer transformer with $d_{model} = 1024$
  - Wall Street Journal (WSJ)
  - about 40k training sentences.
  - vocabulary of 16K tokens for the WSJ only setting

- semi-supervised setting
  - larger high-confidence & BerkleyParser corpora from with 17M sentences.
  - vocabulary of 32K tokens for the semi-supervised setting.

- maximum output length to input length +300
- beam size : 21, $\alpha = 0.3$

- Despite the lack of task-specific tuning,

  Transformer performs surprisingly well!

7. Conclusion

- Transformer : the first sequence transduction model
  based entirely on attention, replacing the recurrent layers
  with multi-headed self-attention.

- significantly faster than architectures based on
  recurrent or convolutional layers.

- plan to extend the Transformer to problems involving input & out
  modalities other than text
- investigate local, restricted attention mechanisms
  to effectively handle large inputs & outputs such as images, audio & video.

*1. Constituency parsing

: 문장이 구 단위로 묶여가면서 구조를 이뤄가는 방법

어순이 고정적인 영어에서 쓰임.

*2. Adam Optimizer

- Adaptive Movement Estimation
- Adagrad + momentum
- RMSProp 과 momentum을 섞어서 씀.

- Adagrad : 파라미터별 필요한만큼의 lr 설정

$$W = W - aglr * grads$$

$$aglr = \frac{lr}{\sqrt{g} + e}$$

lr을 점차 줄임.

$$g = g + grads * grads$$

↓

- 각각의 기울기에 따라 차별적으로 학습률이 정해짐.
- 학습을 반복할수록 g의 값 ↑ ⇒ lr ↓, 학습이 멈춤. "학습률 소실 문제"

↓

- RMSProp
  - Adagrad를 개선한 방식
  - Adagrad의 root square에 가중평균 개념 추가
  - g 값을 구하는 방식이 달라짐.
  - $g = pg + (1-p) \cdot (grads)^2$

하이퍼 파라미터. ex) 0.25 : 기존 25%, 현 grads 75%

Adam

- 모멘텀의 운동량 V
- RMSProp 의 수치 S

- $V = p * V + (1-p) * grads$
- $S = p * S + (1-p) * grads^2$

↓ 편향 보정

- $VC = \frac{V}{(1-p^t)}$
- $SC = \frac{S}{(1-p^t)}$

↓

- $W = W - lr * \frac{VC}{\sqrt{SC} + e}$

*3. Perplexity (PPL)

- 모델 내에서 자신의 성능을 수치화하여 내놓는 내부평가 (intrinsic evaluation)
- '헷갈리는 정도' , 낮을수록 언어모델의 성능이 좋다는 것을 의미.
- 단어의 수로 정규화된 테스트 데이터에 대한 확률의 역수

- $$PPL(\underbrace{W}_{\text{문장}}) = P(w_1, w_2, w_3, \dots, w_{\underbrace{N}_{\text{문장 } W\text{의 길이}}})^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \cdots, w_N)}}$$

↓ 문장의 확률에 chain rule 적용

- $$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i \mid w_1, w_2, \dots, w_{i-1})}}$$

n-gram 적용

- $$PPL(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i \mid w_{i-1})}}$$

※4. Beam Search

- 디코딩 타임스텝에서 전체 단어 사전에 대한 확률 분포 예측.
- 전체공간을 탐색하는 것은 계산적으로 불가능.

↓ 가장 많이 사용되는 방법 두가지

① Greedy Search Decoder

② Beam Search Decoder

① Greedy Search Decoder

- Output sequence 에서 생성된 각각의 확률분포에서 가장 값이 높은 토큰을 선택.
- 직관적, 구현간단, 속도 빠름.

② Beam Search Decoder

- 각 스텝에서 가능도가 가장 높은 k개의 토큰 유지 (k : hyperparameter)

- length penalty
  - 현재 time-step까지의 확률을 모두 곱하기 때문에 문장이 길어질수록 확률이 낮아짐.
  - 이러한 현상을 방지하기 위해 length penalty를 주어 search가 조기 종료되는 것을 막음.

$$\log \tilde{P}(\hat{Y} \mid X) = \log P(\hat{Y} \mid X) * \text{penalty}$$

$$\text{penalty} = \frac{(1+\text{length})^{\alpha}}{(1+\beta)^{\alpha}}$$

where β is hyper parameter of minimum length.