Chapter 8. NLP with Transformer Based Model

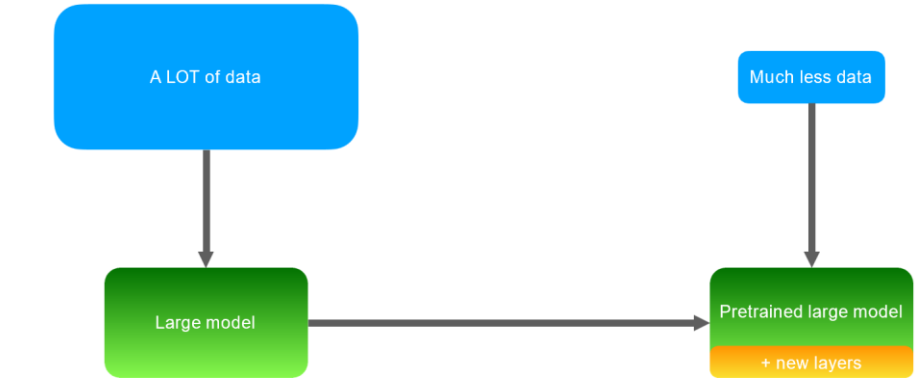
Outline

A. Transfer Learning

B. GPT/GPT2, BERT

C. Natural Language Generation

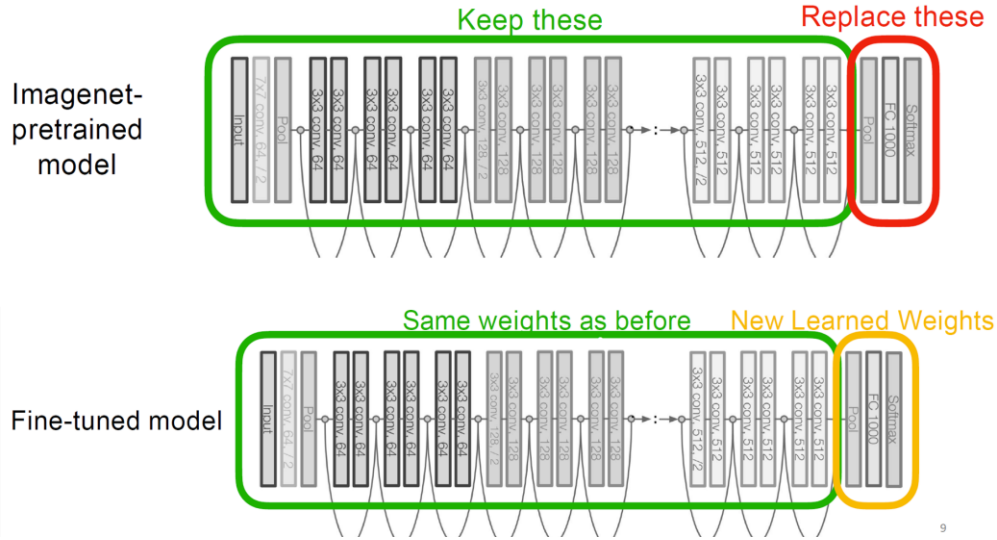
A. Transfer Learning



Language Model은 특히 Transfomer는 학습 시 방대한 양의 Data를 필요로 한다. 만약, 학습 데이터가 부족할 경우 Attention이 제대로 되지 않기 때문에 성능이 저하된다.

하지만, 실제로 Data양이 제한되어 있는 경우가 많다. 이런 경우를 위해 적은 양의 데이터(Less data)로 Pretraining (사전학습)을 진행한 후 fine-tuning (미세 조정)을 거쳐 downstream task에 적용한다.

이처럼 Pretraining + fine-tuning의 학습을 Transfer learning (전이 학습)이라고 한다. 전이 학습은 기존의 방식보다 적은 양의 데이터를 요구하지만, 학습이 더 빠르다.

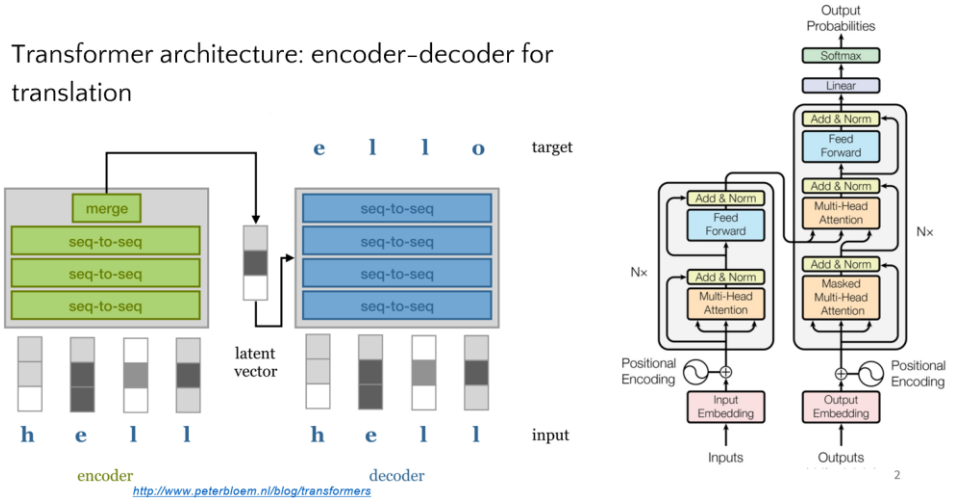


Pretraining은 general한 Task(such as NSP, SOP, MLM)에 초점을 맞춰 학습을 진행한다. 이렇게 generalization된 모델을 좀 더 세부적인 task에 적용하기 위해 parameter를 조절하는 과정이 fine-tuning이다.

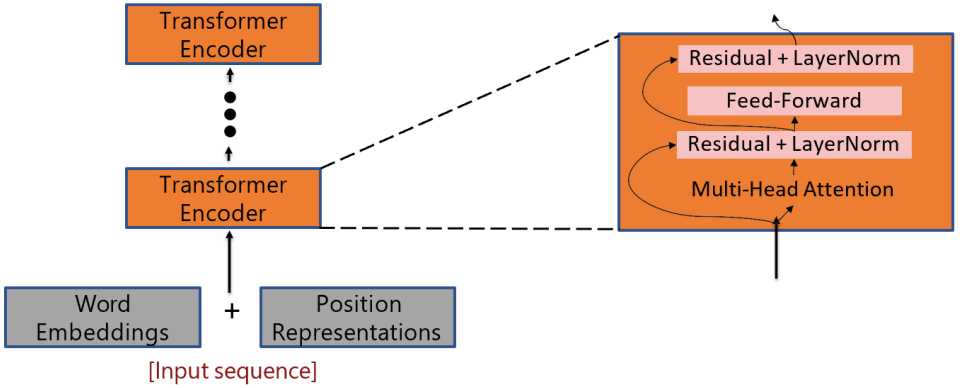
B. GPT/GPT-2/BERT

1) Recap of Transformer

- Transformer architecture: encoder-decoder for translation



- Focus on the encoder block



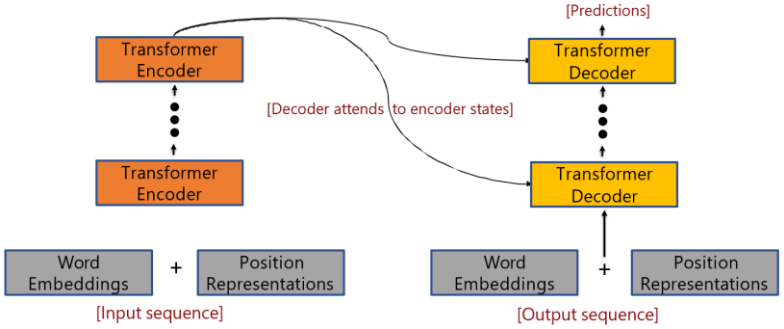
Transformer는 대표적인 Encoder-Decoder 모델이며 Multi-head attention을 기반으로 한다. Transformer에서 하는 Attention은 크게 두 종류가 있다.

a. Self-Attention

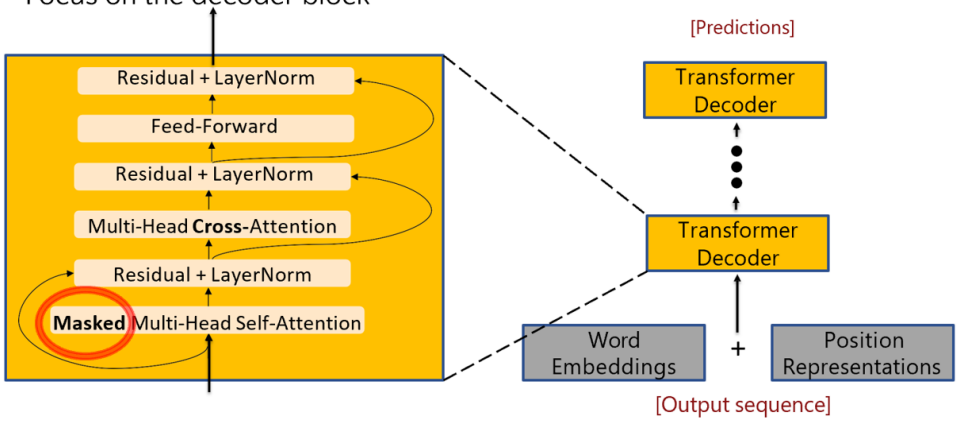
b. Cross-Attention

이 때, Self-Attention은 Weight를 업데이트하는 목적이 아닌, 입력 토큰들 간의 연관성만을 학습하며, Cross-Attention의 경우, 인코더의 최종 출력이 self-attention과 마스킹 작업을 거쳐온 디코더의 입력과의 attention이다.

- Looking back at the whole model, zooming in on an Encoder block

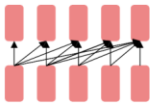


- Focus on the decoder block



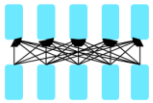
2) Pre-training for Three types of Architecture

Encoder / Decoder / Encoder-Decoder로 총 3가지 타입의 유형이 존재.



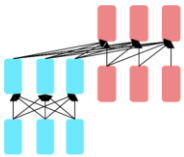
Decoders

- Language models for generating words
- Can't condition on future words
- **Examples:** GPT-2, GPT-3



Encoders

- Gets bidirectional context – can condition on future
- How do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



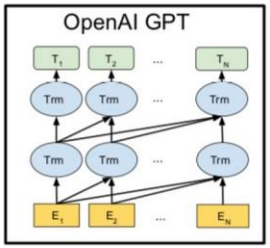
Encoder-Decoders

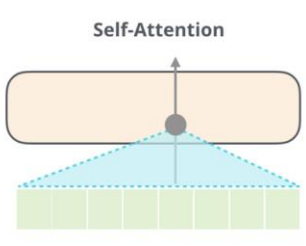
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5

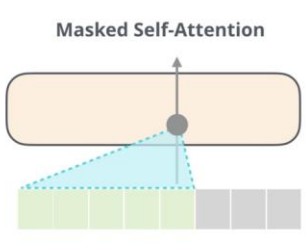
GPT 계열의 모델은 대표적인 Decoder 기반의 모델이다. 반면, BERT는 Transformer의 Encoder이다. 그림에서도 보이듯이, Encoder 기반의 모델인 BERT는 Bidirectional하게 attention을 수행한다. 반면, GPT의 경우는 Uni-directional하다.

3) Generative Pre-trained Transformer (GPT)

- Learns to predict the next word in the sequence
- Since it conditions only on preceding words, it uses masked self-attention



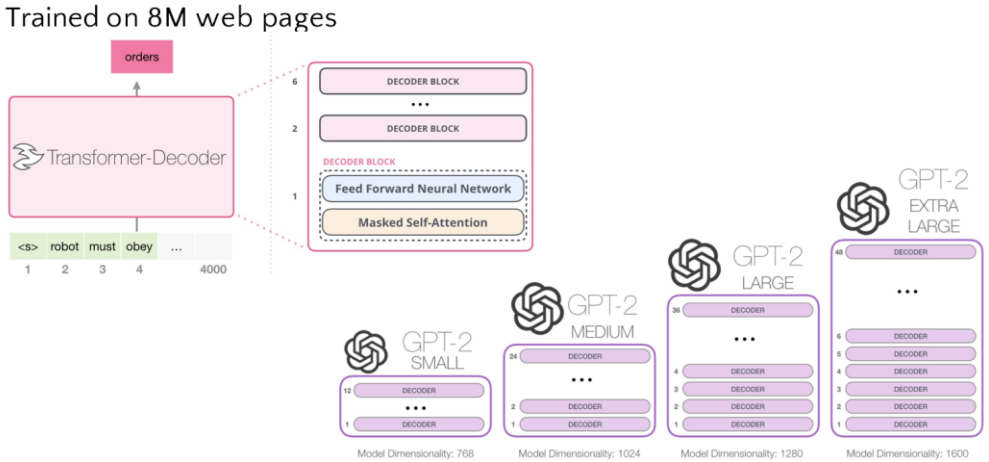




(1) GPT Overview

GPT는 시퀀스에서 **'다음 단어가 무엇인가?'**를 학습한다. 따라서, 미래 시점의 토큰이 학습 중에 등장하며 안되므로, Masking을 이용해 미래 시점의 토큰들을 모두 가려버린다.

GPT-1과 GPT-2는 기본 모델은 동일하나, Pre-training시 사용하는 데이터의 양과 Parameter양에서 차이를 보인다.



(2) Unsupervised Pretraining in GPT

GPT 계열의 핵심은 Pretraining 할 때 , Unsupervised Corpus가 주어졌을 때, Log-likelihood를 최대화하는 것이다.

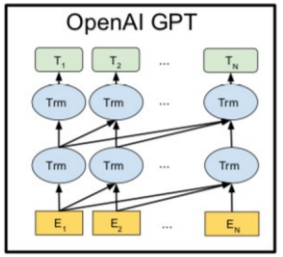
Given an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \dots, u_n\}$, maximize the log-likelihood

$$L_1(\mathcal{U}) = \Sigma_i log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$$

- k is the size of context windows,
  P is modelled as neural network with parameters Θ

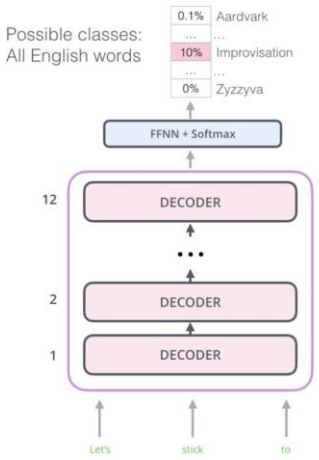
$$h_0 = UW_e + W_p$$
$$h_l = \text{Decoder}(h_{l-1}) \; \forall l \in [1, L]$$
$$P(u) = \text{softmax}(h_L W_e^T)$$

- U is one-hot representation of tokens in the window,
  L is the total number of transformer layers,
  Decoder() denotes the decoder of the transformer model



Pre-training의 목적은 General한 Task를 풀기 위함이다. 따라서, labeling이 되지 않은 corpus에 대해서 학습을 하는 것이다. 이를 통해 사전 학습된 모델을, labeling된 데이터를 이용해 Supervised learning을 하는 과정이 fine-tuning이다.

(3) Supervised Fine-tuning in GPT



- Given labelled data $\mathcal{C}$, including each input as a sequence of tokens $x^1, \ldots, x^m$, and each label as y:

$$P(y|x^1, \ldots, x^m) = \text{softmax}(h_l^m W_y + b_y)$$

  - h_l^m: the final decoder block's activation
  - W_y, b_y: parameters to be learned in finetuning

- Maximize the final objective function

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda L_1(\mathcal{C})$$

  - $L_2(\mathcal{C}) = \Sigma_{(x,y)} log P(y|x^1, \ldots, x^m)$
  - λ is set as 0.5 in the experiment

Pre-training이 끝난 GPT를 labeling된 데이터를 통해 파라미터를 미세 조정하여 특정 Task를 풀기 위한 준비를 하는 것이 바로 Fine-tuning이다. Labeling이 된 데이터를 이용해 Learnable parameter인 W를 학습시키는 과정이다.

수식을 살펴보면 h_l^m은 마지막 decoder layer의 activation이고, W_y와 b_y는 각각 학습 가능한 weight와 bias값이다. 즉, decoder의 최종 출력값을 Weighted SoftMax 한 것이 labeling된 input data $x^1, x^2 \ldots, x^m$에 대한 조건부 확률 값이 된다.

P(y|$x^1, x^2 \ldots, x^m$) = 주어진 supervised data에 대한 label의 출현 확률 값

a. Loss for pre-training = $L_1(u)$

b. Loss for fine-tuning = $L_2(u)$

최종적으로 $L_3(u) = L_1(u) + \lambda L_2(u)$ 를 objective function으로 하며 이를 최대화하는 것을 목표로 한다.

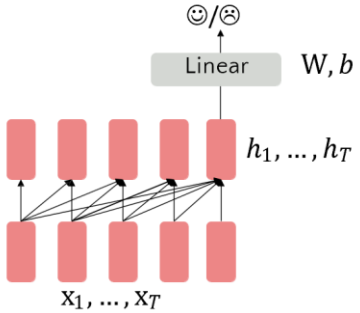
Maximization $L_3(u)$ 인 이유는 두 Loss가 결국 정답 Label에 대한 출현 확률이기 때문이다. 정답을 뽑아 낼 확률 값을 최대하는 방향으로 최적화를 한다.

(4) Fine-tuning GPT

- When using language model pretrained decoders, ignore that they were trained to model $p(x_t|x_{1:t-1})$.
- Finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}(x_1, \ldots, x_T)$$
$$y \sim W h_T + b$$

  - where W and b are randomly initialized and specified by the downstream task.

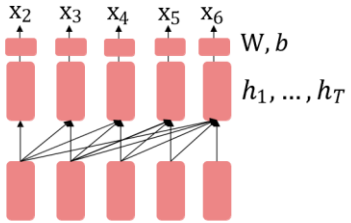


- Gradients backpropagate through the whole network.
- It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(x_t|x_{1:t-1})$

- Pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(x_t|x_{1:t-1})$
- Tasks **where the output is a sequence** with a vocabulary like that at pretraining time!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)

$$h_1, \ldots, h_T = \text{Decoder}(x_1, \ldots, x_T)$$
$$x_t \sim W h_{t-1} + b$$

  - where W, b were pretrained in the language model!



- 2018's GPT was a big success in pretraining a decoder!
- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
- Contains long spans of contiguous text, for learning long-distance dependencies.

(5) GPT-2

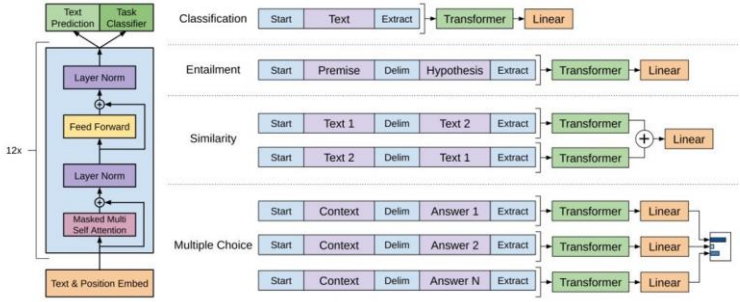
Pretrained decoders can be used **in their capacities as language models.**

GPT-2, a larger version of GPT trained on more data, produced relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

* fine-tuning task

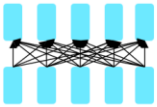
- How do we format inputs to our decoder for finetuning tasks?



- The linear classifier is applied to the representation of the [EXTRACT] token.

4) BERT

(1) Pretraining for encoders



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa

Bidirectional하게 Context를 학습한다. 따라서 미래 시점의 Token을 볼 수 있다. 따라서, Encoder 기반 모델에서는 사전 학습을 어떻게 할지가 Key point이다.

[GPT-1 논문 리뷰]

[BERT 논문 리뷰]

(2) Bidirectional Encoder Representation from Transformer (BERT)

- Encoder blocks only (no masking attentions)

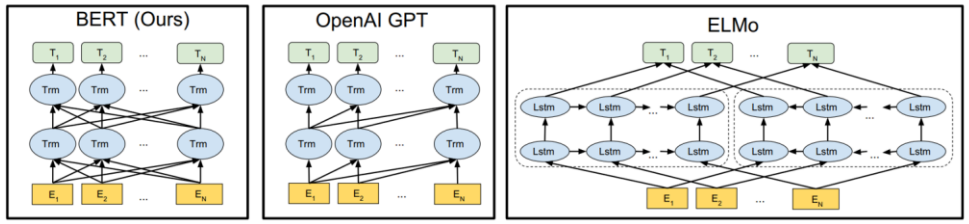
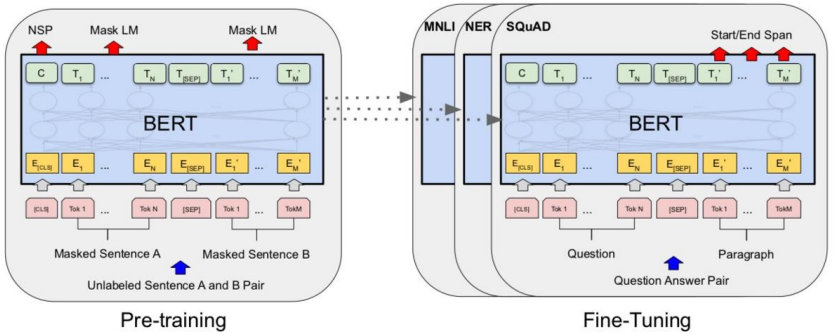


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

Bidirectional Model에서는 Attention시 Masking을 진행하지 않는다. 또한 양방향으로 토큰을 처리하기 때문에, 미래 시점의 토큰도 볼 수 있다. 이를 어떻게 처리해 줄 지가 관건이다.

Unified architecture: minimal differences between the pre-training architecture and the fine-tuned version for each downstream task.



(3) Pretraining Encoders

a. Problem

Encoder가 Bidirectional하게 Context 정보를 학습한다. 이는 Language Modeling을 제대로 수행하지 못함을 의미한다.

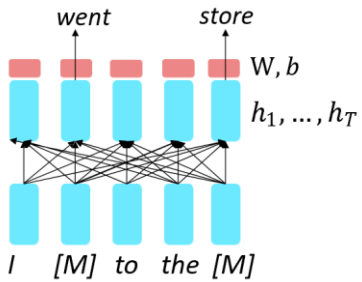
b. Idea

Input sequence에 서 일부분을 [MASK] 토큰으로 대체한다. 즉, 입력 시퀀스의 일부 토큰들을 Masking하는 것이다.

Idea:
replace some fraction of words in the input with a special [MASK] token.
predict these words.

Training tasks

1. **Masked language model (Cloze task)**: only add loss terms from words that are "masked out."
   learn $p_\theta(x|\tilde{x})$ where $\tilde{x}$ is the masked version of x.
2. Next sentence prediction (binary classification)



이런 방식을 Masked Language Modeling이라고 한다. 중요한 것은 마스킹 처리된 단어에서만 Loss term을 추가한다는 것이다.

(4) Masked Language Modeling

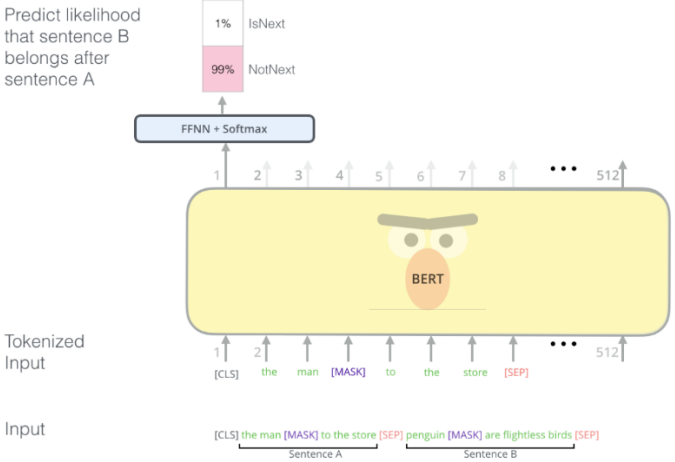
논문에서는 Masking과정을 generalization을 위한 방법을 제시한다. 일반화를 위해서 Masking하도록 정해진 토큰에 대하여 확률적인 variation을 주는 것이다. 먼저 예측을 위해 무작위로 15% 토큰을 선택한다. 단, 토큰들은 아래의 조건을 만족한다.

a. 80%는 [Mask] 토큰으로 교체
b. 10%는 다른 토큰(단어)로 교체
c. 10%는 변경되지 않은 토큰을 사용

여기서 말하는 **확률적인 variation**은 위의 세 가지 방식으로 토큰을 나눈 후, 토큰을 뽑을 때 임의로 한 가지를 뽑는 것이다.

(5) Next Sentence Prediction

Class 토큰 [CLS]의 출력 값으로 binary classification을 하는 task이다. 즉, 출력 토큰의 첫 번째 토큰이 0이냐 1이냐에 따라 결과가 달라짐을 의미한다.



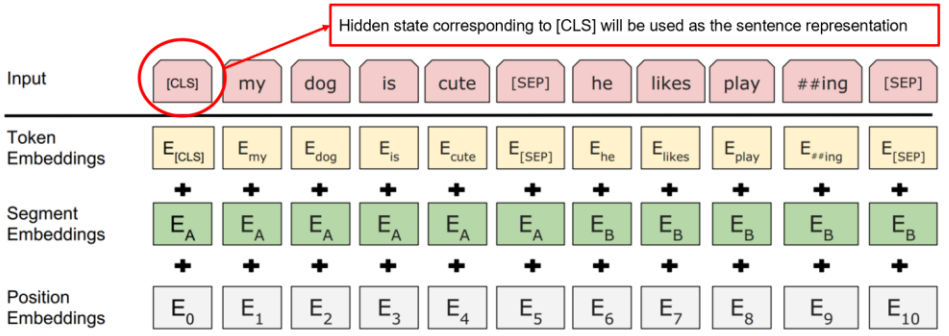
Segment단위, 즉 문장 단위로 랜덤하게 split해서 입력으로 들어온다. 이 때 randomly selected된 두 문장이 입력으로 들어오는데 다음과 같은 조건을 만족한다.

a. Segment A를 고른다.
b. 50% step 중, 다른 문서에서 sentence를 랜덤하게 split해 segment B에 넣는다.
c. 50% step 중, 진짜 정답인 next sentence를 segment b에 넣는다.
d. MASK(Truncate([segment A, segment B]))

BERT는 한 chunk가 다른 chunk를 이어서 오는 것인지, 아니면 무작위로 추출된 것인지를 예측하기 위해 학습되었다.

(6) Input representation of BERT

- Token Embeddings: WordPiece embedding (Wu et al., 2016)
- Segment Embeddings: randomly initialized and learned; single sentence input only adds E_A
- Position embeddings: randomly initialized and learned



3가지의 Input embedding이 더해져서 입력으로 들어가며, 첫 번째 입력 토큰이 Class 토큰인 [CLS]토큰이 된다. 이 [CLS] 토큰은 Next Sentence Prediction에 쓰이며, 다른 Task에 적용시 주로 classification task를 푸는데 사용된다.(Ex) Link Prediction)

* Detail of BERT

Sequence length: **512**

Two models were released:

- BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
- BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

Trained on:

- BooksCorpus (800 million words) about 7,000 unpublished books from a variety of genres
- English Wikipedia (2,500 million words), excluding lists, tables, headers

Pretraining is expensive and impractical on a single GPU.

- BERT was pretrained with 64 TPU chips for a total of 4 days.
- (TPUs are special tensor operation acceleration hardware)

Finetuning is practical and common on a single GPU

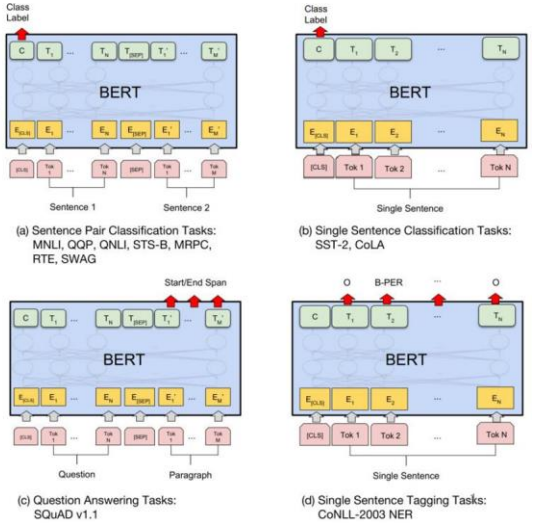
- "Pretrain once, finetune many times."

(7) Finetuning with BERT

Context vector C: take the final hidden state corresponding to [CLS].

Transform to a probability distribution of the class labels:

- $P = \text{softmax}(CW^T)$



(8) Evaluation for BERT: GLUE

General Language Understanding Evaluation (GLUE)는 표준적으로 데이터를 훈련, 검증, 테스트 세트로 분할하는데, 테스트 세트의 라벨은 서버에만 보관된다.

OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

(9) Why does BERT Work

Leveraging huge unlabeled and high-quality data: 7000 books + Wikipedia (together 3300M words)

Multi-head self-attention blocks in Transformer

- Modelling the intra- and extra- word-word relations
- Parallelable within instance and thus efficient

Masked language modelling

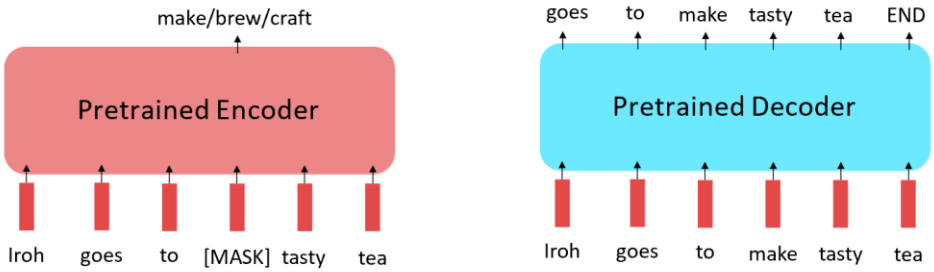
(10) How to improve BERT

- **Pre-training**
  - **Better tasks** for pre-training for more complex usage
  - **Better** (larger, high-quality) **data**
    - Cross-lingual BERT for unsupervised learning (Lample & Conneau, 2019)
  - Even **larger model**, GPT2: zero shot to outperform the SOTA (Radford et al., 2018b)
- **Fine-tuning**
  - **Better loss** in fine-tuning
  - Introduce **new tasks** in fine-tuning

(11) Limitation of BERT

Why not used pretrained encoders for everything?

If your task involves generating sequences, consider using a pretrained decoder;
BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.



BERT를 비롯한 Encoder 기반의 사전 학습 모델은 autoregressive하지 못하다. 양방향으로 학습하기 때문에 바로 다음 단어를 생성해 내는 것이 아닌, 랜덤한 위치에 토큰을 맞추는데 적합하기 때문이다.

(12) Extension of BERT

a. RoBERTa: BERT의 hyperparameter를 최적화하고, 좀 더 많은 양의 데이터로 좀 더 길게 학습하였으며, NSP task를 진행하지 않는다.

b. SpanBERT: 연속된 단어가 Masking된 경우 하나의 Mask로 간주해 좀 더 어려운 Task를 풀게 된다.

C. Natural Language Generation

1) What is Natural Language generation?

NLP = Natural language understanding + Natural Language generation

NLG는 coherent하고 useful한 language를 사람의 이해를 위해 생성해내는 것을 목표로 하는 Task이다.

2) Language Modeling

- **Language modeling**: the task of **predicting the next word**, **given the words so far**

$$P(y_t|y_1, \dots, y_{t-1})$$

- A system that produces this probability distribution is called a **language model**

- **Conditional language modeling**: the task of **predicting the next word**, **given the words so far and other input x**

$$P(y_t|y_1, \dots, y_{t-1}, x)$$

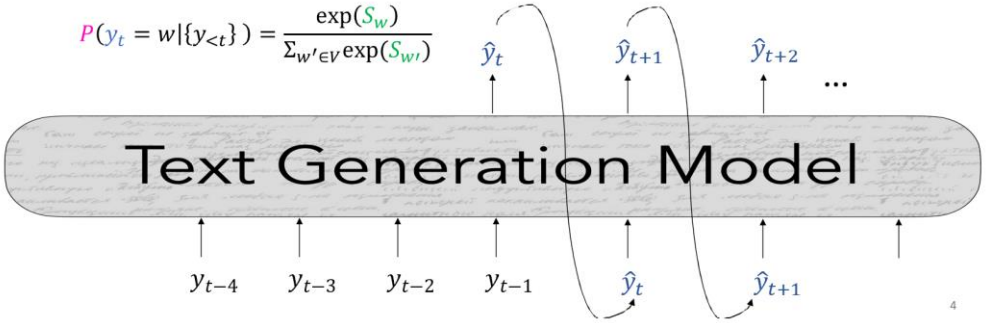
  - Machine translation (x=source sentence, y=target sentence)
  - Summarization (x=input text, y=summarized text)
  - Dialogue (x=dialogue history, y=next utterance)

Language Modeling이란, 주어진 단어들을 이용하여 다음 단어가 무엇인지 맞추는 것이다.(조건부 확률) 즉, 앞선 단어들을 토대로 생성하기 때문에 original한 방식은 Auto-regressive 하다.

(1) Basics of Natural Language Generation

- At each time step t, an autoregressive text generation model takes in a sequence of tokens of text as input $\{y\}_{<t}$ and outputs a new token, $\hat{y}_t$
- For model $f(.)$ and vocab V, get scores $S = f(\{y_{<t}\}, \theta) \in \mathbb{R}^V$

$$P(y_t = w|\{y_{<t}\}) = \frac{\exp(S_w)}{\Sigma_{w' \in V} \exp(S_{w'})}$$



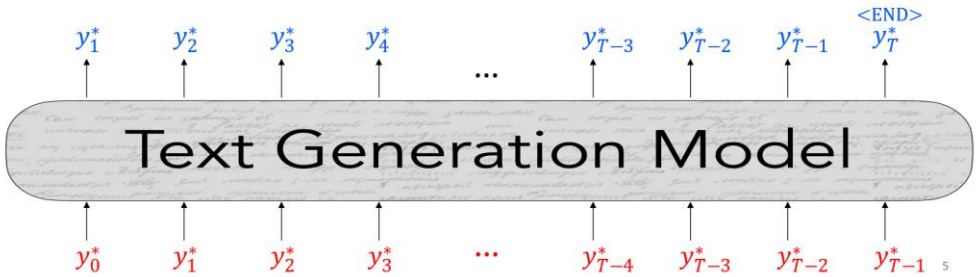
4

(2) Trained One token at a time by Maximum likelihood Teacher Forcing

- Trained to maximize the probability of the next token y^* given preceding words $\{y^*\}_{<t}$

$$\mathcal{L} = -\Sigma_{t=1}^{T} log P(y_t^*|\{y^*\}_{<t})$$

  - A classification task at each time step trying to predict the actual word y_t^* in the training data
  - Doing this is called "teacher forcing" (because you reset at each time step to the ground truth)



5

At inference time, our decoding algorithm defines a function to select a token from this distribution:

- The "obvious" decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

$$\hat{y}_t = g(P(y_t|\{y_{<t}\}))$$

$g(.)$ is your decoding algorithm

While this basic algorithm sort of works, to do better, the two main avenues are to:

1. Improve the decoder
2. Improve the training

Of course, there's also improving your training data or model architecture

3) Decoding From NLG Models

(1) Decoding

At each time step t, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^V$:

$$S = f(\{y_{<t}\})$$

$f(.)$ is your model

Then, we compute a probability distribution P over these scores (usually with a softmax function):

$$P(y_t = w|\{y_{<t}\}) = \frac{\exp(S_w)}{\Sigma_{w' \in V} \exp(S_{w'})}$$

Our decoding algorithm defines a function to select a token from this distribution:

$$\hat{y}_t = g(P(y_t|\{y_{<t}\}))$$

$g(.)$ is your decoding algorithm

(2) Decoding Method

In Neural Machine Translation

Argmax Decoding (Greedy Decoding)

- Selects the highest probability token in $P(y_T|\{y\}_{<T})$

$$\hat{y}_T = argmax_{w \in V} P(y_T = w|\{y\}_{<T})$$

Beam Search

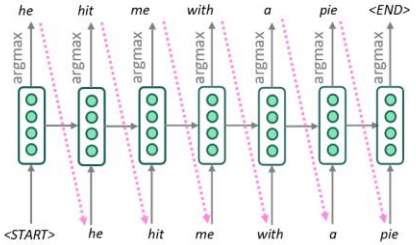
- At heart also a greedy algorithm, but with wider exploration of candidates

Decoding을 하는 방법으로는 크게 두 가지가 있다. Greedy는 매 step 최상의 확률값을 선택하는 방식이다. 하지만 이 방식은 최종적으로 그 값이 진짜 최대인지 확신하기 어렵다. 이를 대체하기 위해 나온 방식이 Beam Search이다. Beam Search는 Greedy와 달리 매 step마다 Top-K개의 확률을 뽑는 방식이다.

Greedy

Generate (or "decode") the target sentence by taking argmax on each step of the decoder

- Take **most probable** word on each step (i.e., argmax)
- Use that as the next word, and feed it as input on the next step
- Keep going until you produce <END> (or reach the max length)



Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

We could try computing all possible sequences y

- This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size

This $O(V^T)$ complexity is far too expensive!

Beam Search

Find a **high-probability sequence** (not necessarily the optimal sequence) by tracking multiple possible sequences at once.

At each step of decoder, keep track of the ***k* most probable** partial translations (called *hypotheses*)

- k is the beam size (in practice around 5 to 10, in NMT)

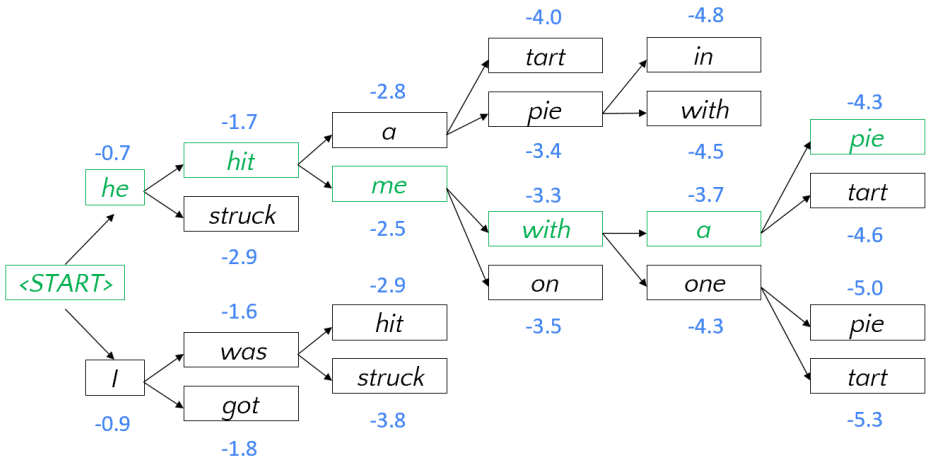
Once you reach a stopping criterion, **choose the sequence with the highest probability**

A hypothesis $y_1, y_2, \ldots, y_T$ has a score which is its log probability:

$$\text{score}(y_1,\ldots,y_t) = \log P_{\text{LM}}(y_1,\ldots,y_t|x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$$

- Scores are all negative, and higher score is better
- We search for high-scoring hypotheses, tracking top k on each step

Beam size = k = 2. Blue numbers = $score(y_1, \dots, y_t) = \Sigma_{i=1}^{t} log P_{\text{LM}}(y_i | y_1, y_2, \dots, y_{i-1}, x)$



- In greedy decoding, we decode until the model produces an <END> token
  - **For example:** *<START> he hit me with a pie <END>*

- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
  - When a hypothesis produces <END>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

- We continue beam search until:
  - We reach timestep *T* (where *T* is some pre-defined cutoff), or
  - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)

Small k

- Similar problem to greedy decoding (k = 1)
- **Ungrammatical, unnatural, nonsensical, incorrect**

Large k: **consider more hypotheses**

- May relieve the above problems
- More **computationally expensive**
- Can introduce new problems
  - For NMT, increasing k too much decreases BLEU score. This is primarily because large-k beam search produces too short translations (even with score normalization)
  - In open-ended tasks, large k can make output **more generic**

Beam Search를 할 때 K값이 작아지면 greedy decoding에 가까워진다는 단점이 있다. 반면 K가 커지면 더 많은 경우를 고려하게 되지만 그와 맞물려 Computational cost가 증가한다.

 # How Can we reduce Repetition?

Given a list of completed hypotheses:

How to select top one with highest score?

Each hypothesis $y_1, y_2, \dots, y_t$ on our list has a score

$$\text{score}(y_1, \dots, y_t) = \log P_{\text{LM}}(y_1, \dots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

Problem with this: longer hypotheses have lower scores

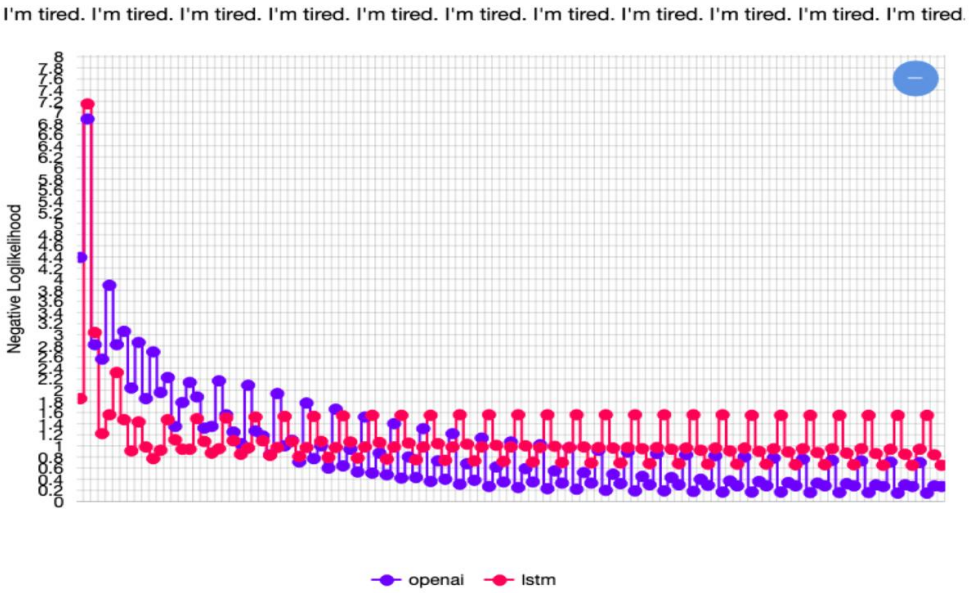
Fix: Normalize by length. Use this to select top one instead:

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

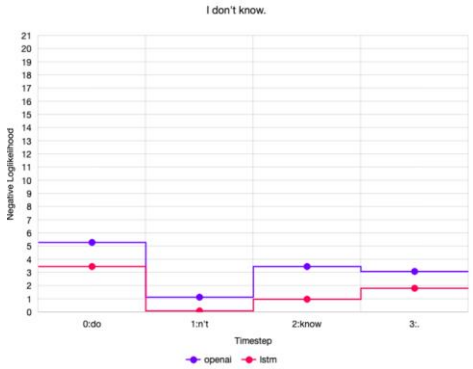
Greedy의 문제점을 보여주는 예시

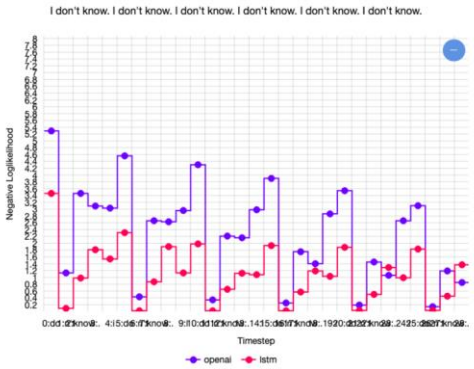
Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México…**



Repetition 이 나타나는 이유





How can we reduce Repetition

Simple option:

Heuristic: Don't repeat *n*-grams

More complex:

Maximize embedding distance between consecutive sentences (Celikyilmaz et al., 2018)

– Doesn't help with intra-sentence repetition

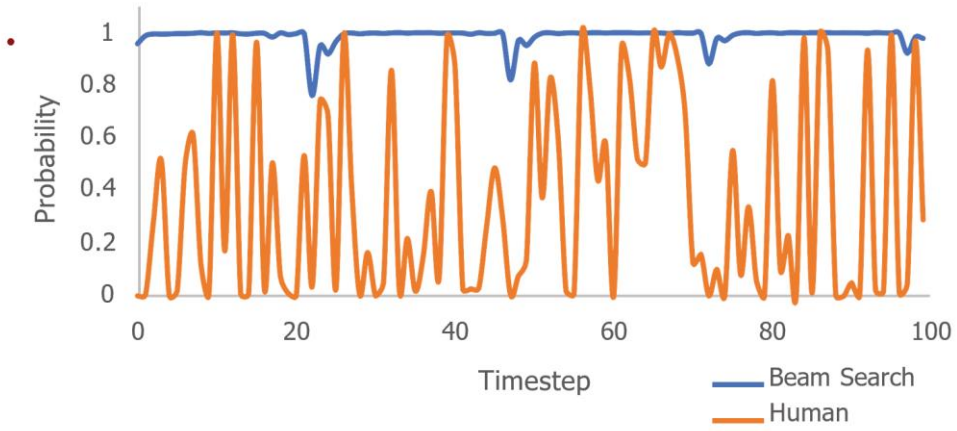
Coverage loss (See et al., 2017)

– Prevents attention mechanism from attending to the same words

Unlikelihood objective (Welleck et al., 2020)

– Penalize generation of already-seen tokens

Are Greedy Methods Reasonable?



Decoding: Top-K Sampling

Problem: Vanilla sampling makes every token in the vocabulary an option

- Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass (statistics speak: we have "heavy tailed" distributions)
- Many tokens are probably *really wrong* in the current context
- Why are we giving them *individually* a tiny chance to be selected?
- Why are we giving them *as a group* a high chance to be selected?

Solution: Top-k sampling

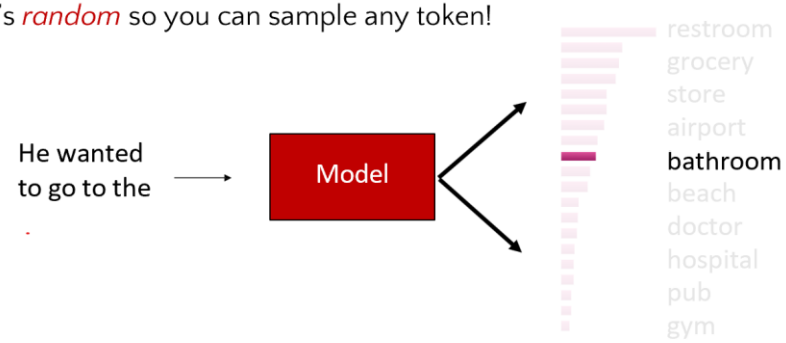
- Only sample from the top k tokens in the probability distribution

Sampling

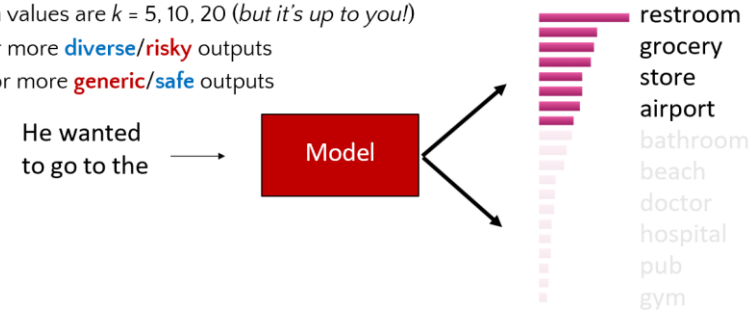
- Sample a token from the distribution of tokens

$$\hat{y} \sim P(y_t = w | \{y\}_{<t})$$

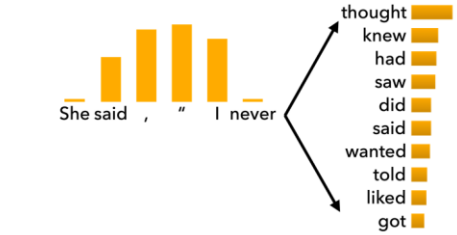
- It's *random* so you can sample any token!



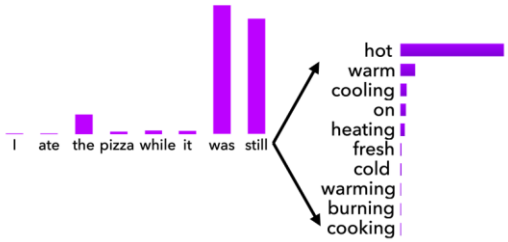
- Solution: Top-k sampling
  - Only sample from the top k tokens in the probability distribution
  - Common values are k = 5, 10, 20 (*but it's up to you!*)
- Increase k for more **diverse**/**risky** outputs
- Decrease k for more **generic**/**safe** outputs



Top-k Sampling의 Issue



Top-k sampling can cut off too ***quickly***!



Top-k sampling can also cut off too ***slowly***!

Decoding: Top-p(Nucleus) Sampling

- Solution: Top-p sampling
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k depending on the uniformity of P_t

$P^1(y_t = w \mid \{y\}_{<t})$ $P^2(y_t = w \mid \{y\}_{<t})$ $P^3(y_t = w \mid \{y\}_{<t})$

(Holtzman et. al., ICLR 2020)

Decoding: Top-p(Nucleus) Sampling

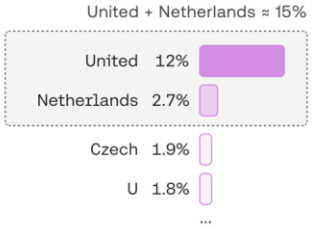
Problem: The probability distributions we sample from are dynamic

- When the distribution P_t is flatter, a limited k removes many viable options
- When the distribution P_t is peakier, a high k allows for too many options to have a chance of being selected

- Solution: Top-p sampling
  - Sample from all tokens in the top p cumulative probability mass (i.e., where mass is concentrated)
  - Varies k depending on the uniformity of P_t

Pick from amongst the top tokens whose probabilities add up to p (e.g., 15%)

1. Consider only the top tokens whose likelihoods add up to 15%. Ignore all others.



2. Sample from them based on their likelihood scores.

Scaling Randomness: SoftMax Temperature

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$

$$P_t(y_t = w) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

- You can apply a *temperature hyperparameter* τ to the softmax to rebalance P_t:

$$P_t(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{w' \in V} \exp(S_{w'}/\tau)}$$

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - **More** diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - **Less** diverse output (probability is concentrated on top words)

Note: softmax temperature is not a decoding algorithm!
It's a technique you can apply at test time, in conjunction with a decoding algorithm (such as beam search or sampling)

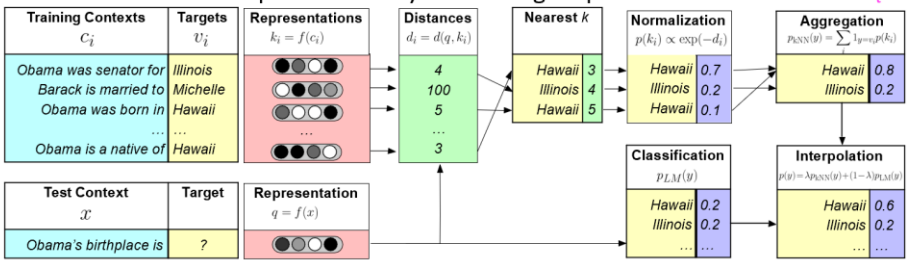
- Problem: What if I decode a bad sequence from my model?

- Decode a bunch of sequences
  - 10 candidates is a common number, but it's up to you
- Define a score to approximate quality of sequences and re-rank by this score
  - Simplest is to use perplexity!
    - Careful! Remember that repetitive methods can generally get high perplexity.
  - Re-rankers can score a variety of properties:
    - style (Holtzman et al., 2018), discourse (Gabriel et al., 2021), entailment/factuality (Goyal et al., 2020), logical consistency (Lu et al., 2020), and many more …
    - Beware poorly-calibrated re-rankers
  - Can use multiple re-rankers in parallel

Improving Decoding: Re-Balancing Distribution

- Problem: What if I don't trust how well my model's distributions are calibrated?
  - Don't rely on **ONLY** your model's distribution over tokens

- One Approach: Re-balance P_t using retrieval from n-gram phrase statistics!
- Cache a database of phrases from your training corpus and use to rebalance P_t



(Khandelwal et. al., ICLR 2020)

- Decoding is still a challenging problem in NLG – there's a lot more work to be done!
- A major realization of the last couple of years is that many of the problems that we see in neural NLG are not really problems with our learned language model probability distribution, but problems with the decoding algorithm
- Human language production is a subtle presentation of information and can't be modeled by simple properties like *probability maximization*
- Different decoding algorithms can allow us to inject biases that encourage different properties of coherent natural language generation
- Some of the most impactful advances in NLG of the last few years have come from simple but effective modifications to decoding algorithms

D. Training NLG Models

1) Exposure Bias

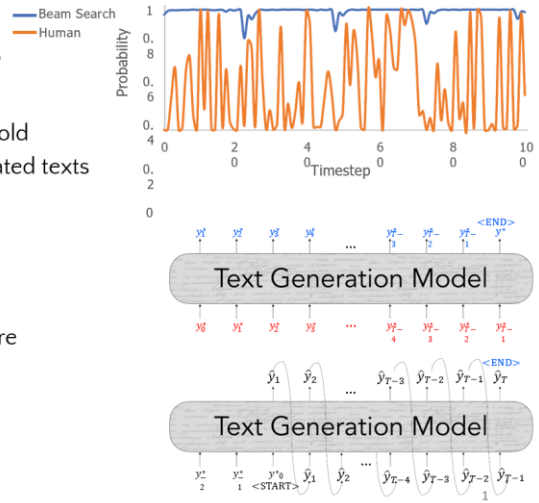
Training with teacher forcing leads to exposure bias at generation time

- During training, our model's inputs are gold context tokens from real, human-generated texts

$$L_{MLE} = -logP(y_t^*|\{y_{<t}^*\})$$

- At generation time, our models' inputs are previously-decoded tokens

$$L_{dec} = -logP(\hat{y}_t|\{\hat{y}_{<t}\})$$



- Teacher forcing is still the main algorithm for training text generation models
- Exposure bias causes text generation models to lose coherence easily.
  - Models must learn to recover from their own bad samples
    - e.g., scheduled sampling
  - Alternatively, not be allowed to generate bad text to begin with (e.g., retrieval + generation)

E. Evaluating NLG Models

1) Content Overlap Metrics

(1) Overview

- Given a reference sentence and a generated sentence:

Reference: They walked to the grocery store .

Generated: The woman went to the hardware store .

- Compute a score that indicates the similarity between *generated* and *gold-standard (human-written) text*
- Fast and efficient and widely used
- Two broad categories:
  - *N*-gram overlap metrics (e.g., **BLEU**, ROUGE, METEOR, CIDEr, etc.)
  - Semantic overlap metrics (e.g., PYRAMID, SPICE, SPIDEr, etc.)

(2) Bilingual Evaluation Understudy (BLEU)

BLEU score is defined as: Brevity penalty N-gram overlap

$$BLEU = \min\left(1, \exp\left(1 - \frac{|reference|}{|output|}\right)\right)\left(\Pi_{i=1}^{4} precision_i\right)^{\frac{1}{4}}$$

with

$$precision_i = \frac{\Sigma_{s \in cand-corpus} \Sigma_{i \in s} \min\left(m_{cand}^i, m_{ref}^i\right)}{w_t^i}$$

$$w_t^i = \Sigma_{s' \in cand-corpus} \Sigma_{i' \in s'} m_{cand}^{i'}$$

where

- m_{cand}^i: the count of i-gram in candidate matching the reference translation
- m_{ref}^i: the count of i-gram in the reference translation
- w_t^i: the total number of i-grams in candidate translation

Brevity penalty

$$BLEU = \boxed{\min\left(1, \exp\left(1 - \frac{|reference|}{|output|}\right)\right)}\left(\Pi_{i=1}^{4} precision_i\right)^{\frac{1}{4}}$$

with

$$precision_i = \frac{\Sigma_{s \in cand-corpus} \Sigma_{i \in s} \min\left(m_{cand}^i, m_{ref}^i\right)}{w_t^i}$$

$$w_t^i = \Sigma_{s' \in cand-corpus} \Sigma_{i' \in s'} m_{cand}^{i'}$$

where

- m_{cand}^i: the count of i-gram in candidate matching the reference translation
- m_{ref}^i: the count of i-gram in the reference translation
- w_t^i: the total number of i-grams in candidate translation

- **Brevity penalty** penalizes generated translations that are too short compared to the closest reference length with an exponential decay.
- This compensates for the fact that the BLEU score has no **recall** term.

N-gram overlap

$$BLEU = \min\left(1, \exp\left(1 - \frac{|reference|}{|output|}\right)\right)\boxed{\left(\Pi_{i=1}^{4} precision_i\right)^{\frac{1}{4}}}$$

with

$$precision_i = \frac{\Sigma_{s \in cand-corpus} \Sigma_{i \in s} \min\left(m_{cand}^i, m_{ref}^i\right)}{w_t^i}$$

$$w_t^i = \Sigma_{s' \in cand-corpus} \Sigma_{i' \in s'} m_{cand}^{i'}$$

where

- m_{cand}^i: the count of i-gram in candidate matching the reference translation
- m_{ref}^i: the count of i-gram in the reference translation
- w_t^i: the total number of i-grams in candidate translation

- N-graph overlap counts how many unigrams, bigrams, trigrams, and four-grams (i=1,...,4) match their n-gram counterpart in the reference translations. This term acts as a precision metrics.
- Unigram account for adequacy while longer n-grams account for fluency of the translation.
- To avoid overcounting, the n-gram counts are clipped to the maximal n-gram count occurring in the reference (m_{ref}^i)

Reference: the cat is on the mat
Candidate: the the the cat mat

- Calculating $precision_1$: count the occurrences of each unigram in the reference and the candidate.
- Total number of unigrams in the candidate w_t^1 is 5, so $precision_1 = \frac{2+1+1}{5} = 0.8$

Unigram	m_{cand}^i	m_{ref}^i	$\min(m_{cand}^i, m_{ref}^i)$
the	3	2	2
cat	1	1	1
is	0	1	0
On	0	1	0
mat	1	1	1

Reference: The NASA Opportunity rover is battling a massive dust storm on Mars .
Candidate 1: The Opportunity rover is combating a big sandstorm on Mars .
Candidate 2: A NASA rover is fighting a massive storm on Mars .

- N-gram precisions

Metric	Candidate 1	Candidate 2
$precision_1$ (1 gram)	8/11	9/11
$precision_2$ (2 grams)	4/10	5/10
$precision_3$ (3 grams)	2/9	2/9
$precision_4$ (4 grams)	0/8	1/8
Brevity Penalty	0.83	0.83
BLEU Score	**0.0**	**0.27**

- **Brevity penalty**: the same for candidate 1 and candidate 2 since both sentences consist of 11 tokens.
- **BLEU score**: at least one matching 4-gram is required to get a BLEU score > 0. Since candidate translation 1 has no matching 4 gram, it has a BLEU score of 0.

(3) Content Overlap Metrics

Word overlap–based metrics (BLEU, ROUGE, METEOR, CIDEr, etc.)

They're not ideal for machine translation

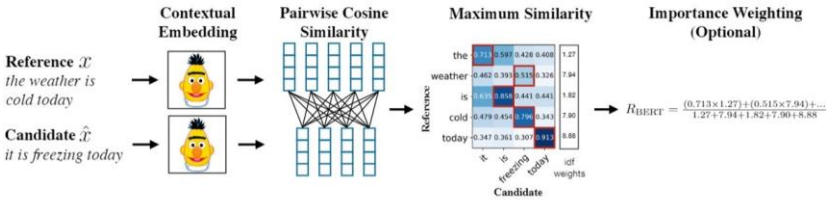
They get progressively much worse for tasks that are more open-ended than machine translation

- **Worse** for summarization, as longer output texts are harder to measure
- **Much worse** for dialogue, which is more open-ended that summarization
- **Much, much worse** story generation, which is also open-ended, but whose sequence length can make it seem you're getting decent scores!

2) Model Based Metrics

BERTSCORE (Zhang et.al., 2020)

- Uses pretrained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity



3) Human Evaluation

- Automatic metrics fall short of matching human decisions
- Human evaluation is most important form of evaluation for text generation systems
  - >75% generation papers at ACL 2019 included human evaluations
- Gold standard in developing new automatic metrics
  - New automated metrics must correlate well with human evaluations!
- Ask *humans* to evaluate the quality of generated text
  - fluency
  - coherence / consistency, …

4) Summary

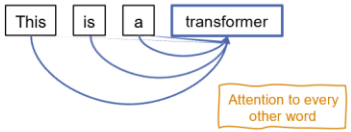
- *Content overlap metrics* provide a good starting point for evaluating the quality of generated text. You will need to use one but they're not good enough on their own.
- *Model-based metrics* can be more correlated with human judgment, but behavior is not interpretable
- *Human judgments* are critical
  - Only thing that can directly evaluate *factuality* – is the model saying correct things?
  - But humans are inconsistent!

F. Generalization of Transformer to Graphs

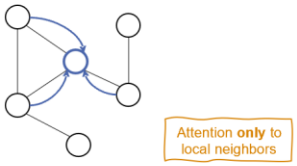
1) Objective

Goal: generalize transformers to arbitrary graphs, i.e., design Graph Transformer

Extend key principles behind Transformer's success to graph structured datasets



Idea: use attention to local neighbors in a graph instead of attending to all nodes, like GATs



2) Graph Transformer

Some attempts to generalize transformer's attention mechanism to graphs with several focused on specialized cases such as:

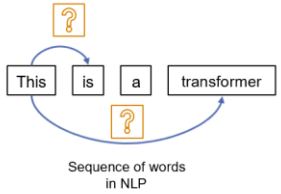
- Temporal networks
- Heterogeneous graphs

Open questions

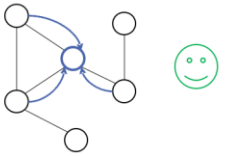
- Local vs. global attention
- Sparse structure vs. full graph
- Positional encoding candidates
- Scalability and pre-training

3) Key Design Aspects

- **Graph Sparsity**
  - Transformers consider fully connected graph and this choice can be justified:
    - Difficult to find meaningful sparse connections between words, i.e., **absence of sparse structure**
    - Transformer sentences often contain less than tens or hundreds of words, hence **scalable to consider full attention**.
  - **But graphs have arbitrary connectivity structure**
  - Full attention is not feasible (millions/billions of nodes)
  - GNNs are state-of-the-art on several application domains since they consider **sparse structure** into account while learning feature representations.
  - Sparsity is a good **inductive bias** for generalization.



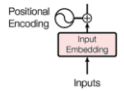
Sequence of words in NLP



Graph with arbitrary structure

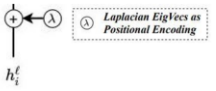
- **Positional Encodings (PE)**
  - Transformers are supplied with a positional encoding for each word

<Sequence> This is a transformer v/s <Multi-set> {This, is, a, transformer}



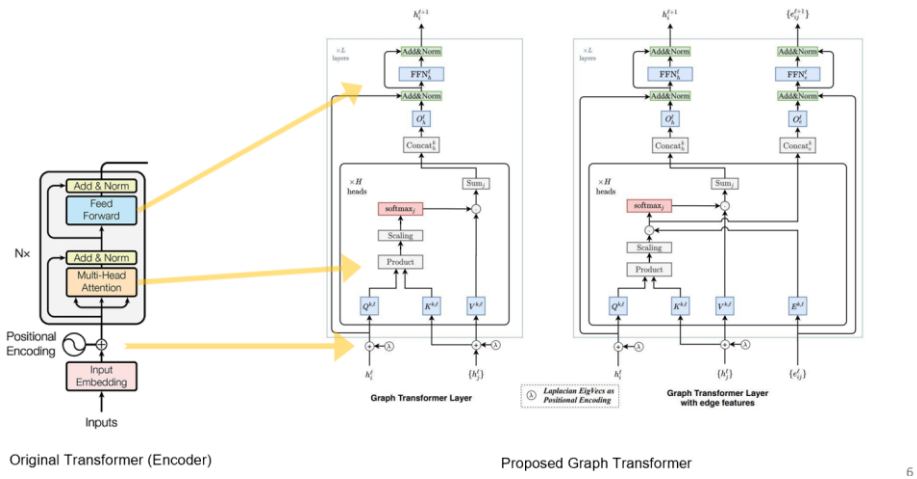
PE in Original Transformer

  - It is natural to use a mechanism to encode node positions in graph transformer, as explored in recent research for position integration in GNNs.
  - PEs in graph is challenging and hard to get canonical node position information.
  - Use **Laplacian Eigenvectors** corresponding to a node, as the node positional encoding **Laplacian PE (λ)**, which generalizes the sinusoidal PE used in Transformers



PE in Proposed Graph Transformer

4) Architecture of Graph Transformer



Original Transformer (Encoder) Proposed Graph Transformer

Compared to the original transformer, the highlights of the proposed architecture are:

a. The attention mechanism is a function of neighborhood connectivity for each node.

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\Vert_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

Node — Node neighborhood

$$\text{where, } w_{ij}^{k,\ell} = \text{softmax}_j \Big(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \Big),$$

b. Positional encoding is represented by Laplacian PE – k smallest non-trivial eigenvector of a node.

c. The layer normalization is replaced by a batch normalization layer.

d. The architecture is extended to have edge representation, which can be critical to tasks with rich information on the edges.

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\Vert_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big),$$

$$\hat{e}_{ij}^{\ell+1} = O_e^{\ell} \Big\Vert_{k=1}^{H} \Big(\hat{w}_{ij}^{k,\ell} \Big), \quad \text{where,}$$

$$w_{ij}^{k,\ell} = \text{softmax}_j(\hat{w}_{ij}^{k,\ell}),$$

$$\hat{w}_{ij}^{k,\ell} = \Big(\frac{Q^{k,\ell} h_i^{\ell} \cdot K^{k,\ell} h_j^{\ell}}{\sqrt{d_k}} \Big) \cdot E^{k,\ell} e_{ij}^{\ell},$$

Available edge information

- **Evaluated** on **ZINC, CLUSTER, PATTERN** from the Benchmarking GNNs.
- **Results**
  - GT outperforms vanilla GCNs and GATs.
  - GT emerges as a fresh powerful attention ba GNN baseline.
  - GT reaches competitive performance with GatedGCN that report the best performance on these datasets.
  - This work only targeted for a straightforward generalization of transformers, and not SOTA.

Model	ZINC	CLUSTER	PATTERN
GNN BASELINE SCORES from (Dwivedi et al. 2020)			
GCN	0.367±0.011	68.498±0.976	71.892±0.334
GAT	0.384±0.007	70.587±0.447	78.271±0.186
GatedGCN	0.214±0.013	76.082±0.196	86.508±0.085
OUR RESULTS			
GT (Ours)	0.226±0.014	73.169±0.622	84.808±0.068

Table: Comparison of performance on each dataset against baseline vanilla GNN (GCN) and baselines anisotropic GNNs (GAT, GatedGCN). Performance Measure for ZINC is MAE (lower is better), and for CLUSTER, PATTERN is Accuracy (higher is better).