BERT Overview

2019년 6월 29일

이동현 차석 (A.I. Plus Lab), ESTsoft Inc.

NLP's ImageNet moment has arrived (2018!)

미리 학습된
Word Embedding

SkipGram
Fasttext
GloVe
+ RNN, CNN, Attentions

미리 학습된
Sentence Emb.

ELMo
ULMFiT
OpenAI GPT
Google **BERT**

2018년 **SQuAD 1.1**
벤치마크에서 큰 차이로 1위

그 이후...

BERT+MTL
BERT+...
BERT+AoA

* 많은 Unlabeled Data로 **임베딩만** 학습, 태스크별 Net(CNN/RNN)
* 대부분의 파라메터가 **적은 태스크별 데이터**로만 학습됨

* **많은 Unlabeled Data**로 DNN 전체를 **미리** 학습, 적은 태스크별 Net
* 대부분의 파라메터가 **많은 태스크 독립적인 데이터**로 학습됨

* **통일된 방법** 위에서 다양하게 진화
* **Big Language Model**을 만드는 것이 거의 모든 태스크에 도움이 됨

http://ruder.io/nlp-imagenet/ (Sebastian Ruder, 2018)

Multi-task NLP (GLUE) Benchmark

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI
1	GLUE Human Baselines	GLUE Human Baselines		87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2
2	Microsoft D365 AI & MSR /	MT-DNN-ensemble		84.2	65.4	96.5	92.2/89.5	89.6/89.0	73.7/89.9	87.9	87.4	96.0
3	王玮	ALICE large (Alibaba DAMO NLP)		83.9	65.3	95.2	92.0/89.3	90.3/89.4	74.1/90.5	88.0	87.7	95.7
… …												
9	Jacob Devlin	BERT: 24-layers, 16-heads, 1024-		80.5	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7	85.9	92.7
10	Neil Houlsby	BERT + Single-task Adapters		80.2	59.2	94.3	88.7/84.3	87.3/86.1	71.5/89.4	85.4	85.0	92.4
11	Alec Radford	Singletask Pretrain Transformer		72.8	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1	81.4	87.4
12	GLUE Baselines	BiLSTM+ELMo+Attn		70.0	33.6	90.4	84.4/78.0	74.2/72.3	63.1/84.3	74.1	74.5	79.8

https://gluebenchmark.com/leaderboard (NYU, U Washington, DeepMind since 2018)

The Stanford Question Answering Dataset 2.0

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Mar 20, 2019	BERT + DAE + AoA (ensemble) *Joint Laboratory of HIT and iFLYTEK Research*	**87.147**	**89.474**
2 Mar 15, 2019	BERT + ConvLSTM + MTL + Verifier (ensemble) *Layer 6 AI*	86.730	89.286
3 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	86.673	89.147
4 Apr 13, 2019	SemBERT(ensemble) *Shanghai Jiao Tong University*	86.166	88.886
5 Mar 16, 2019	BERT + DAE + AoA (single model) *Joint Laboratory of HIT and iFLYTEK Research*	85.884	88.621
6 Mar 05, 2019	BERT + N-Gram Masking + Synthetic Self-Training (single model) *Google AI Language* https://github.com/google-research/bert	85.150	87.715
7 Mar 13, 2019	BERT + ConvLSTM + MTL + Verifier (single model) *Layer 6 AI*	84.924	88.204
8 Apr 16, 2019	Insight-baseline-BERT (single model) *PAII Insight Team*	84.834	87.644
8 Apr 11, 2019	SemBERT (single model) *Shanghai Jiao Tong University*	84.800	87.864
9 Jan 10, 2019	BERT + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	84.292	86.967
10 Dec 21, 2018	PAML+BERT (ensemble model) *PINGAN GammaLab*	83.457	86.122
10 Dec 13, 2018	BERT finetune baseline (ensemble) *Anonymous*	83.536	86.096
11 Dec 16, 2018	Lunet + Verifier + BERT (ensemble) *Layer 6 AI NLP Team*	83.469	86.043

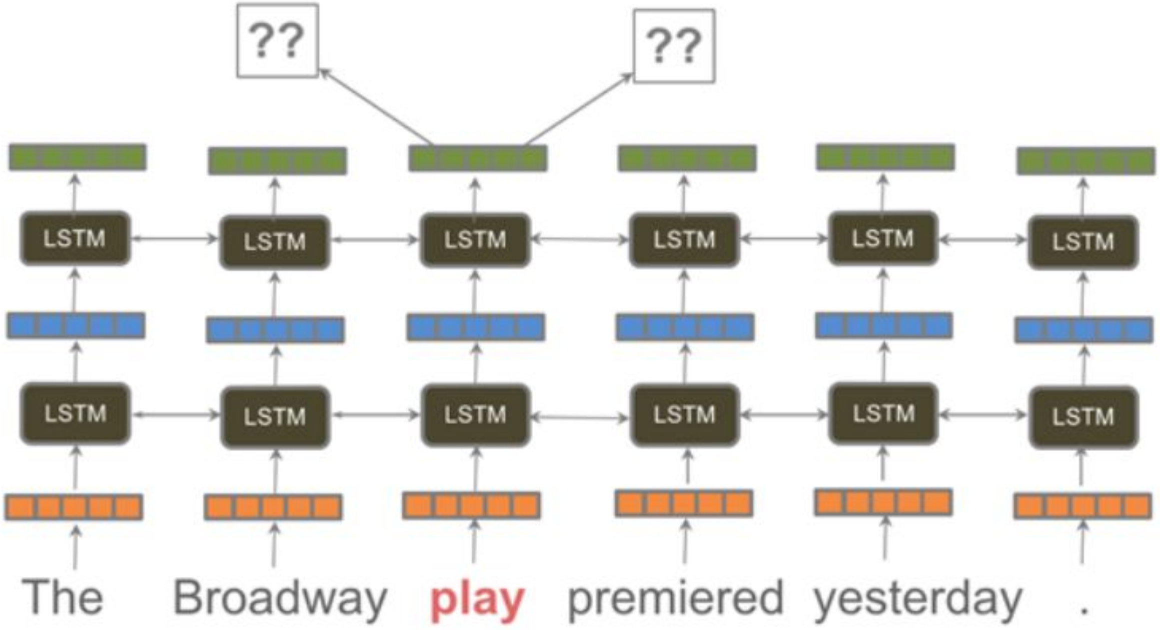
https://rajpurkar.github.io/SQuAD-explorer/ (Stanford, since 2016)

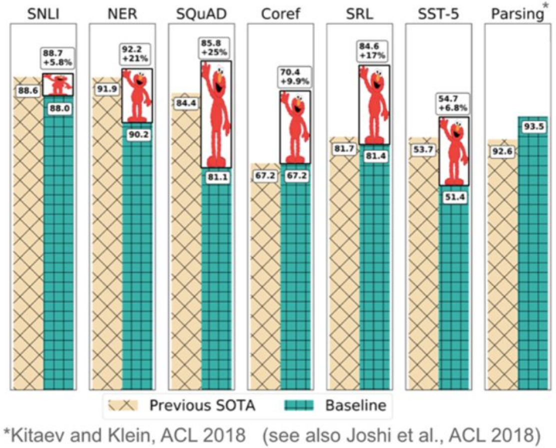
The Korean Question Answering Dataset

-	2018.10.17	**Human Performance**	80.17	91.20
1	2019.06.04	BERT-CLKT-MIDDLE (single model) Anonymous	86.71	**94.55**
2	2019.06.03	LaRva-Kor-Large + CLaF (single) Clova AI LaRva Team (LPT)	**86.79**	94.37
3	2019.03.15	{BERT-CLKT} (single model) Anonymous	86.22	94.08
4	2019.05.07	LaRva-Kor+ + CLaF (single) Clova AI LaRva Team (LPT)	85.35	93.96
5	2019.04.24	LaRva-Kor+ (single) Clova AI LaRva Team (LPT)	85.25	93.94
6	2019.05.24	BBERT fine-tuned(ensemble) Oh Yeon Taek	83.99	92.89
7	2019.04.10	BERT-Kor (single) Clova AI LPT Team	83.79	92.63
8	2019.03.29	BERT insp. by GPT-2 + KHAIII (single) Kakao NLP Team	84.12	92.62

https://korquad.github.io/ (LG CNS, since 2019)

Deep Contextualized Word Representations (ELMo)

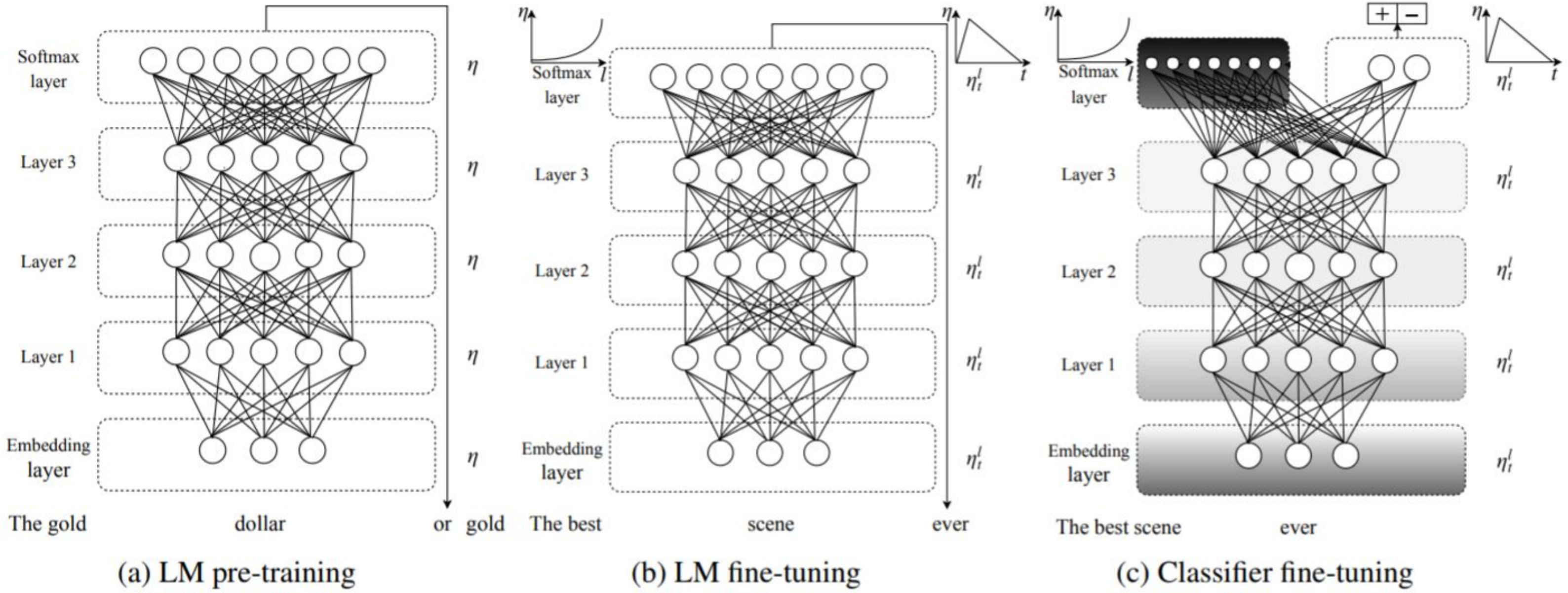




Deep contextualized word representations (Peters et al., AllenAI & UW, NAACL 2018)
https://arxiv.org/pdf/1802.05365.pdf

Beyond Word Embeddings Part 2 (Aaron Bornstein, 2018)
https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec

Universal Language Model Fine-tuning for Text Classification (ULMFiT)

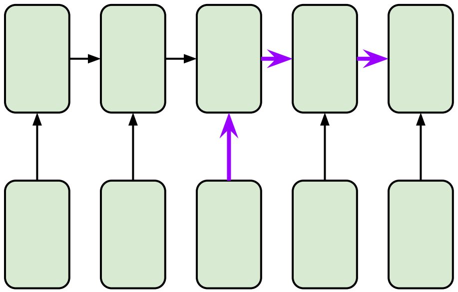


General-Domain LM | Target task LM | Discriminative fine-tuning

Universal Language Model Fine-tuning for Text Classification (Howard, fast.ai & Ruder, DeepMind, ACL 2018)
https://arxiv.org/pdf/1801.06146.pdf

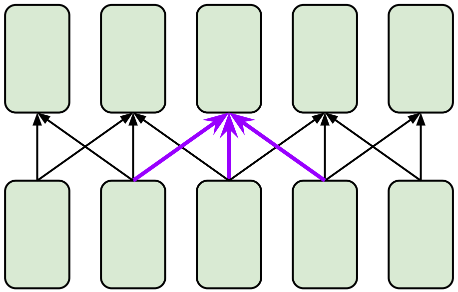
Transformer : Self-Attention Layer !

RNN



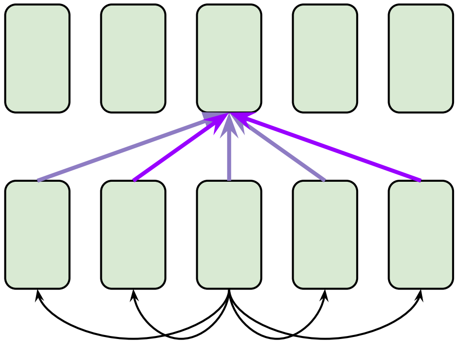
Recurrent Connection

CNN



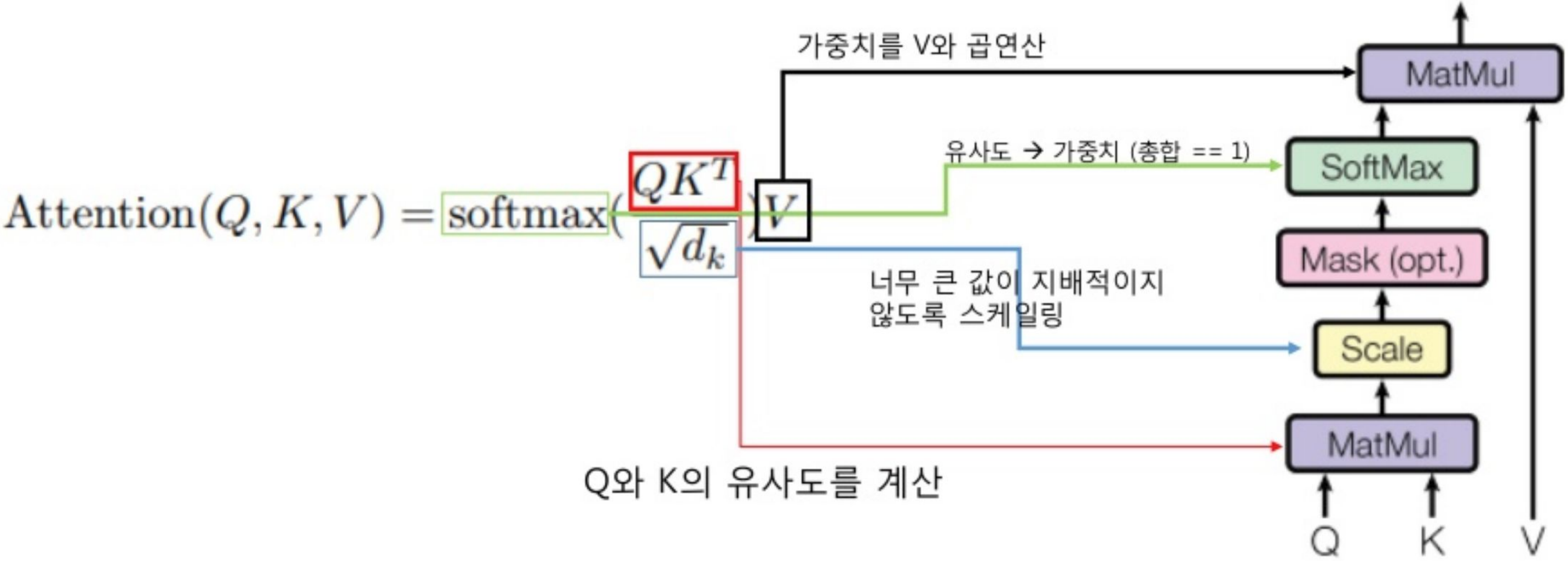
Local Connection

Self-Attention



Content-based Connection
(Dot-Products between units
become connection strength)

Scaled Dot-Product Attention

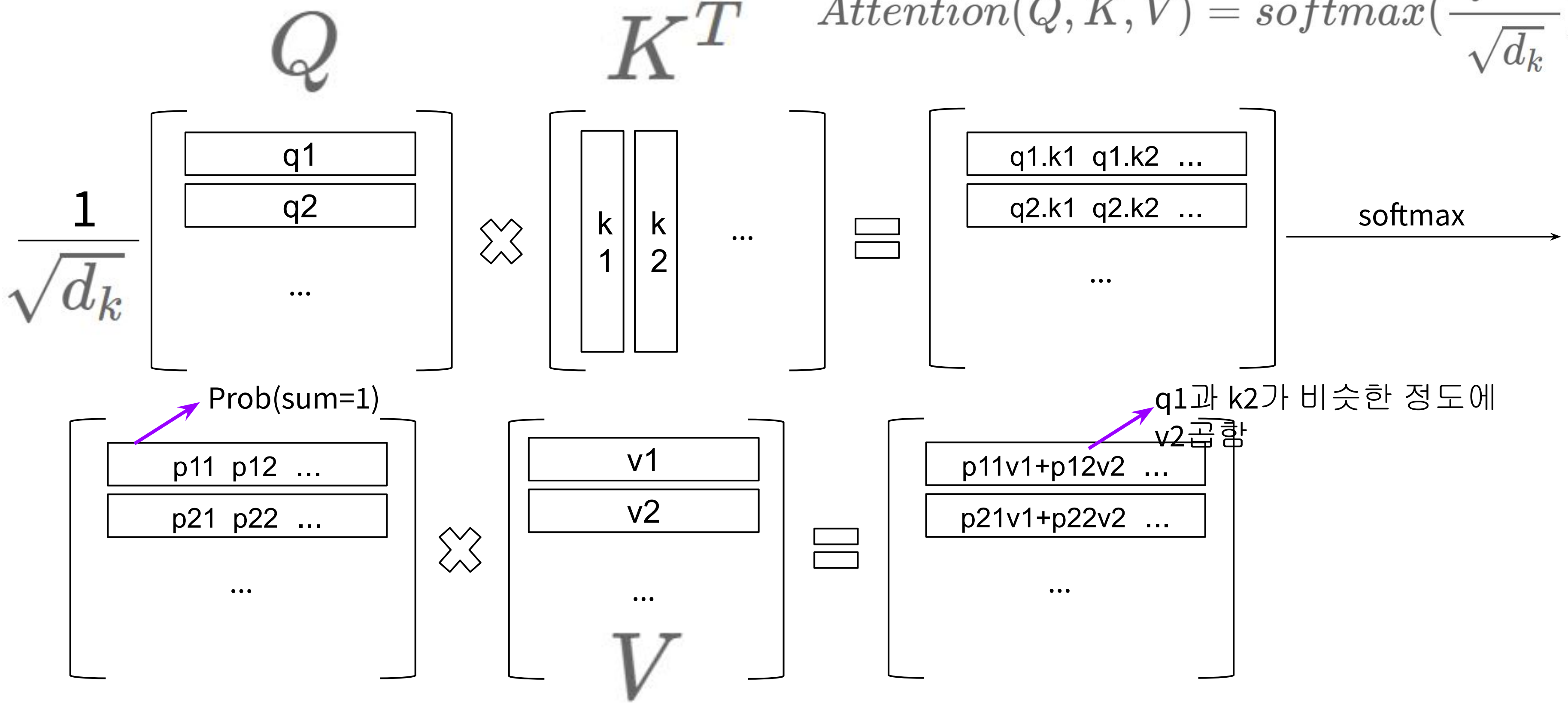


Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

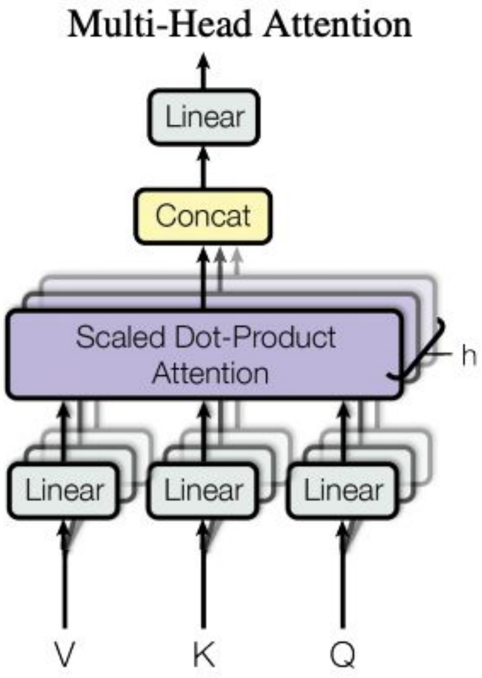
like **K**ey-**V**alue Database(Memory)
Query와 비슷한 **K**ey를 가진 **V**alue를
찾는다.
(실제로는 비슷한 정도로 중첩된 값 리턴)

Scaled Dot-Product Attention

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



Multi-Headed Attention

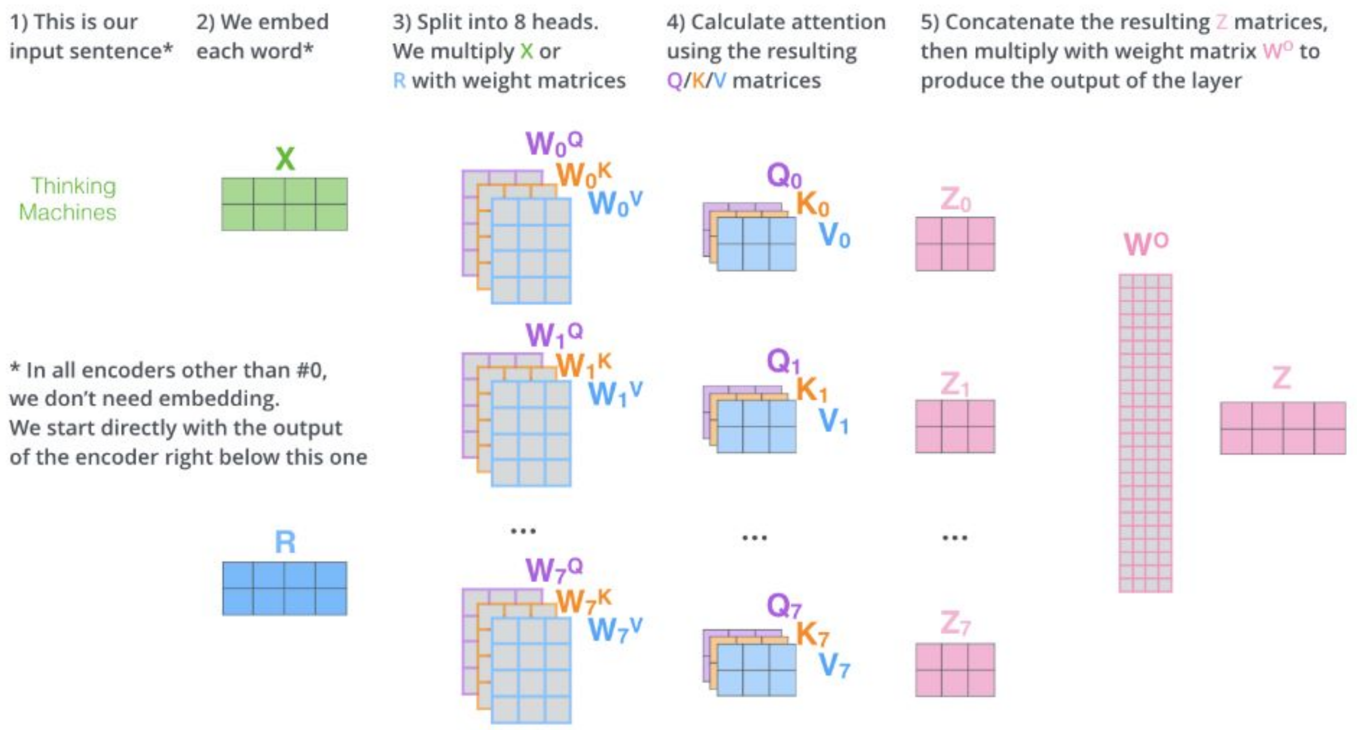


$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O$$

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

Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Multi-Headed Attention



The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

Multi-Headed Attention

```python
class MultiHeadedSelfAttention(nn.Module):
    """ Multi-Headed Dot Product Attention """
    def __init__(self, cfg):
        super().__init__()
        self.proj_q = nn.Linear(cfg.dim, cfg.dim)
        self.proj_k = nn.Linear(cfg.dim, cfg.dim)
        self.proj_v = nn.Linear(cfg.dim, cfg.dim)
        self.drop = nn.Dropout(cfg.p_drop_attn)
        self.scores = None # for visualization
        self.n_heads = cfg.n_heads
```

https://github.com/dhlee347/pytorchic-bert

```python
def forward(self, x, mask):
    """
    x, q(query), k(key), v(value) : (B(batch_size), S(seq_len), D(dim))
    mask : (B(batch_size) x S(seq_len))
    * split D(dim) into (H(n_heads), W(width of head)) ; D = H * W
    """
    # (B, S, D) -proj-> (B, S, D) -split-> (B, S, H, W) -trans-> (B, H, S, W)
    q, k, v = self.proj_q(x), self.proj_k(x), self.proj_v(x)
    q, k, v = (split_last(x, (self.n_heads, -1)).transpose(1, 2)
               for x in [q, k, v])
    # (B, H, S, W) @ (B, H, W, S) -> (B, H, S, S) -softmax-> (B, H, S, S)
    scores = q @ k.transpose(-2, -1) / np.sqrt(k.size(-1))
    if mask is not None:
        mask = mask[:, None, None, :].float()
        scores -= 10000.0 * (1.0 - mask)
    scores = self.drop(F.softmax(scores, dim=-1))
    # (B, H, S, S) @ (B, H, S, W) -> (B, H, S, W) -trans-> (B, S, H, W)
    h = (scores @ v).transpose(1, 2).contiguous()
    # -merge-> (B, S, D)
    h = merge_last(h, 2)
    self.scores = scores
    return h
```

Position-wise FeedForward

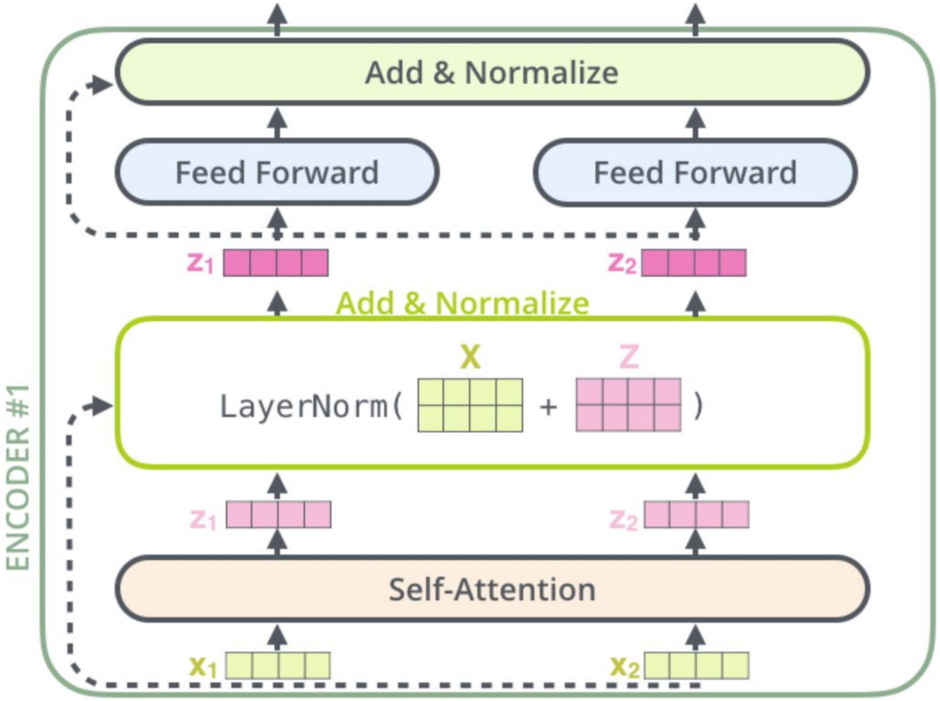
$$FFN(x_i) = max(0, x_i W_1 + b_1) W_2 + b_2$$

For every time steps

```
class PositionWiseFeedForward(nn.Module):
    """ FeedForward Neural Networks for each position """
    def __init__(self, cfg):
        super().__init__()
        self.fc1 = nn.Linear(cfg.dim, cfg.dim_ff)
        self.fc2 = nn.Linear(cfg.dim_ff, cfg.dim)
        #self.activ = lambda x: activ_fn(cfg.activ_fn, x)

    def forward(self, x):
        # (B, S, D) -> (B, S, D_ff) -> (B, S, D)
        return self.fc2(gelu(self.fc1(x)))
```

https://github.com/dhlee347/pytorchic-bert

Transformer Block

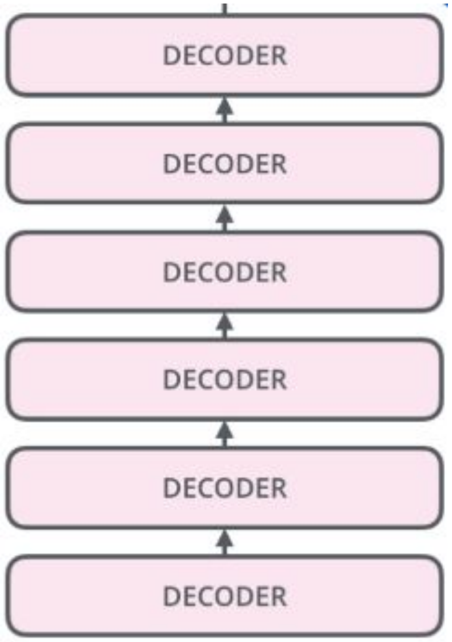



```
class Block(nn.Module):
    """ Transformer Block """
    def __init__(self, cfg):
        super().__init__()
        self.attn = MultiHeadedSelfAttention(cfg)
        self.proj = nn.Linear(cfg.dim, cfg.dim)
        self.norm1 = LayerNorm(cfg)
        self.pwff = PositionWiseFeedForward(cfg)
        self.norm2 = LayerNorm(cfg)
        self.drop = nn.Dropout(cfg.p_drop_hidden)

    def forward(self, x, mask):
        h = self.attn(x, mask)
        h = self.norm1(x + self.drop(self.proj(h)))
        h = self.norm2(h + self.drop(self.pwff(h)))
        return h
```

Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

Transformer

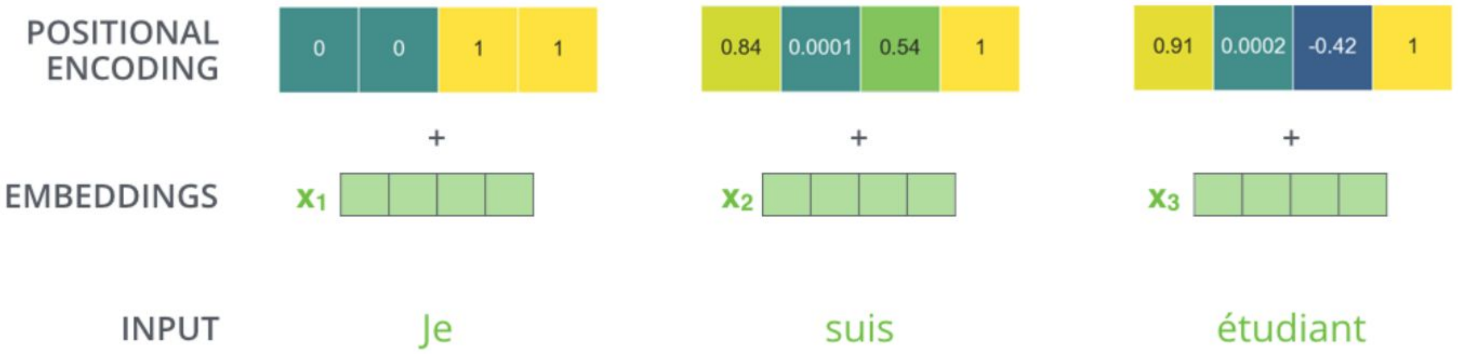



```
class Transformer(nn.Module):
    """ Transformer with Self-Attentive Blocks"""
    def __init__(self, cfg):
        super().__init__()
        self.embed = Embeddings(cfg)
        self.blocks = nn.ModuleList([Block(cfg) for _ in range(cfg.n_layers)])

    def forward(self, x, seg, mask):
        h = self.embed(x, seg)
        for block in self.blocks:
            h = block(h, mask)
        return h
```

Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

Positional Encoding

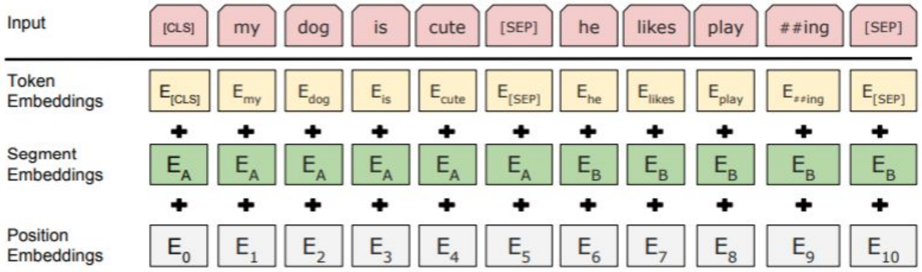


Self-Attention itself can’t deal with positional information !
sine, cosine embedding / learned embedding

Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

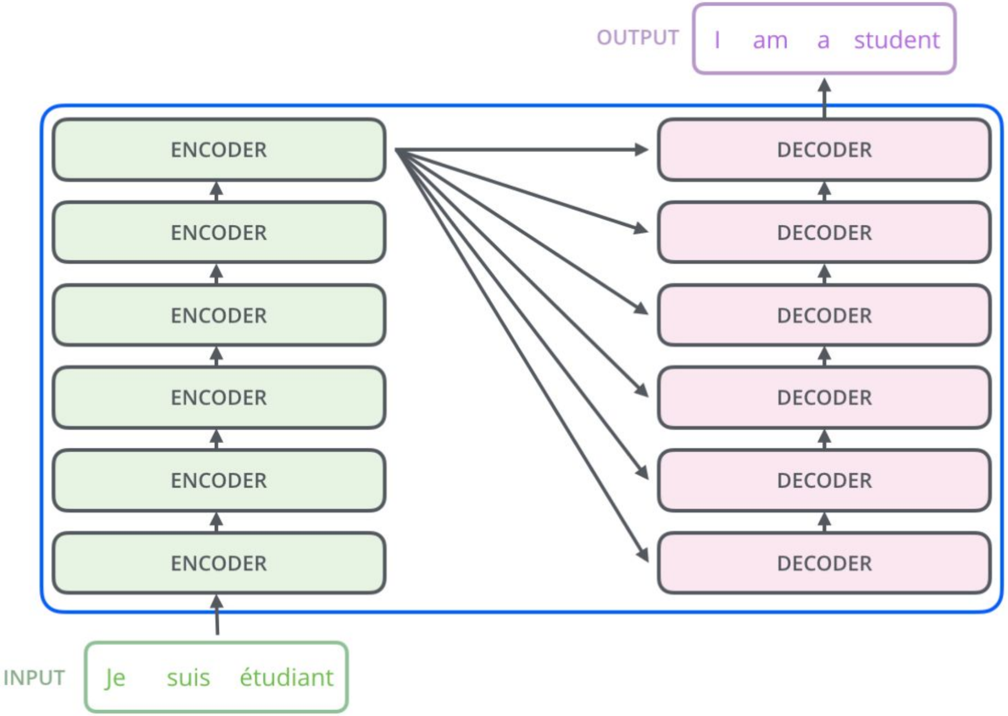
Embedding Layer




```python
class Embeddings(nn.Module):
    "The embedding module from word, position and token_type embeddings."
    def __init__(self, cfg):
        super().__init__()
        self.tok_embed = nn.Embedding(cfg.vocab_size, cfg.dim) # token emb
        self.pos_embed = nn.Embedding(cfg.max_len, cfg.dim) # position emb
        self.seg_embed = nn.Embedding(cfg.n_segments + 1, cfg.dim)  # segm

        self.norm = LayerNorm(cfg)
        self.drop = nn.Dropout(cfg.p_drop_hidden)

    def forward(self, x, seg):
        seq_len = x.size(1)
        pos = torch.arange(seq_len, dtype=torch.long, device=x.device)
        pos = pos.unsqueeze(0).expand_as(x) # (S,) -> (B, S)

        e = self.tok_embed(x) + self.pos_embed(pos) + self.seg_embed(seg)
        return self.drop(self.norm(e))
```

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., Google, NACCL 2019 Best Long Paper)
https://arxiv.org/pdf/1810.04805.pdf

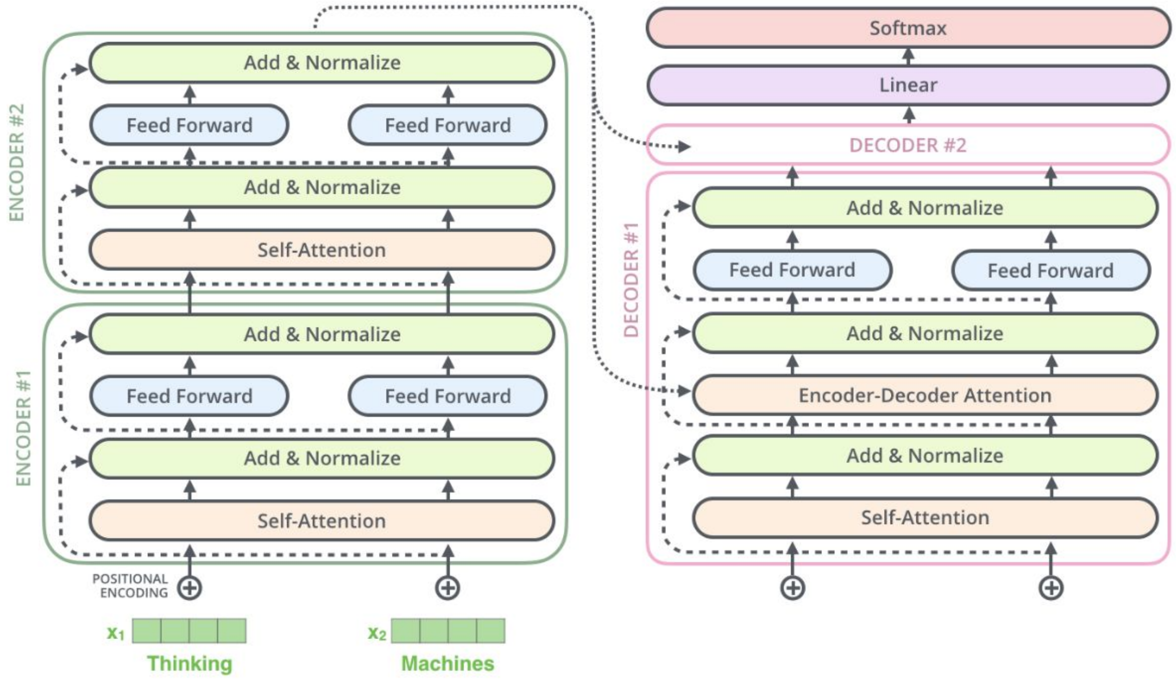
Transformer : Machine Translation



Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

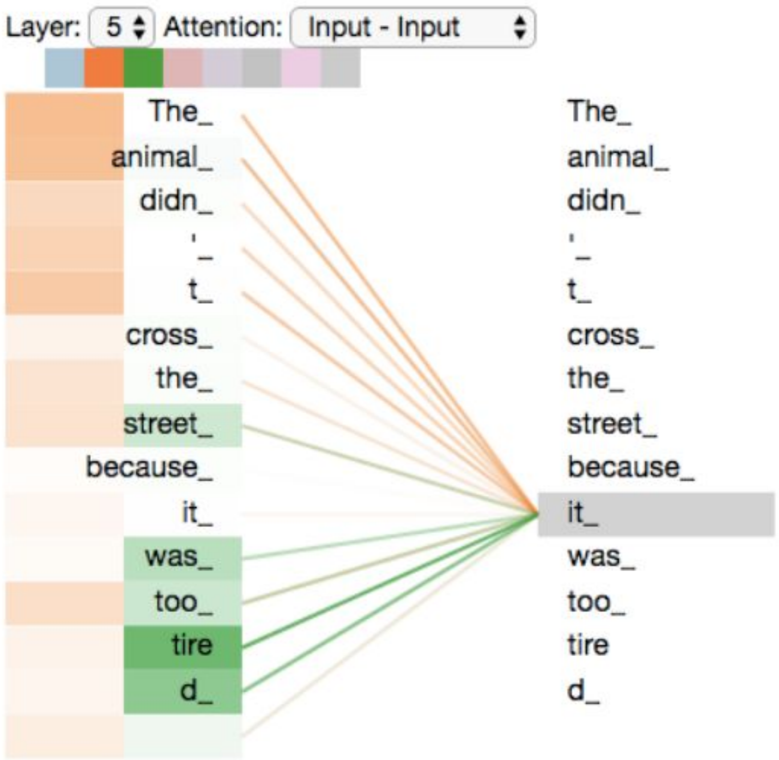
Transformer : Machine Translation



Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

Transformer : How it works



As we encode the word **"it"**, one attention head is focusing most on "the **animal**", while another is focusing on "**tired**" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

Attention Is All You Need (Vaswani et al., Google, NIPS 2017)
https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

The Illustrated Transformer (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-transformer/

Pros & Cons

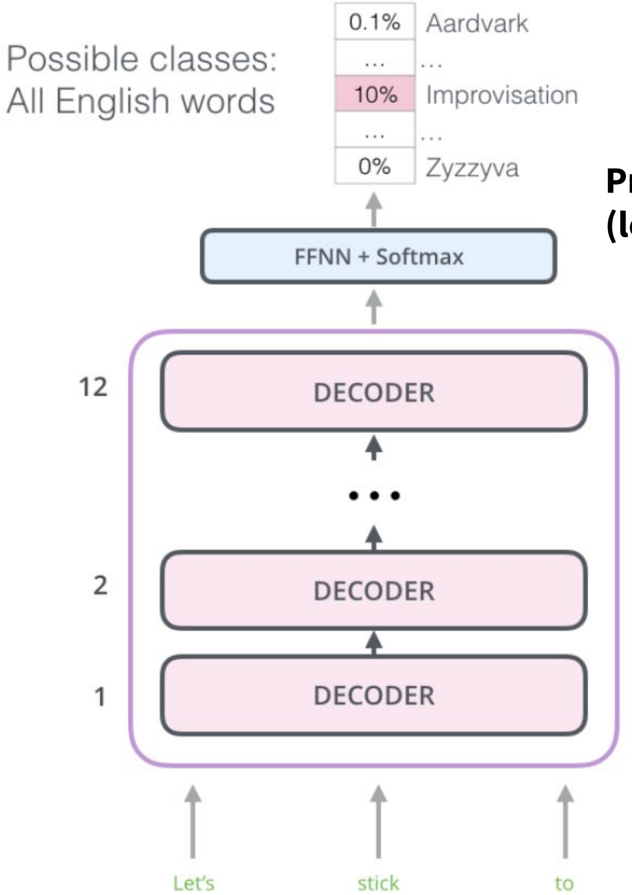
- Long-term dependency
- Parallelization
- Superior Performance

- Time Complexity O(n^2)
- Maximum Lengths

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformer : OpenAI GPT

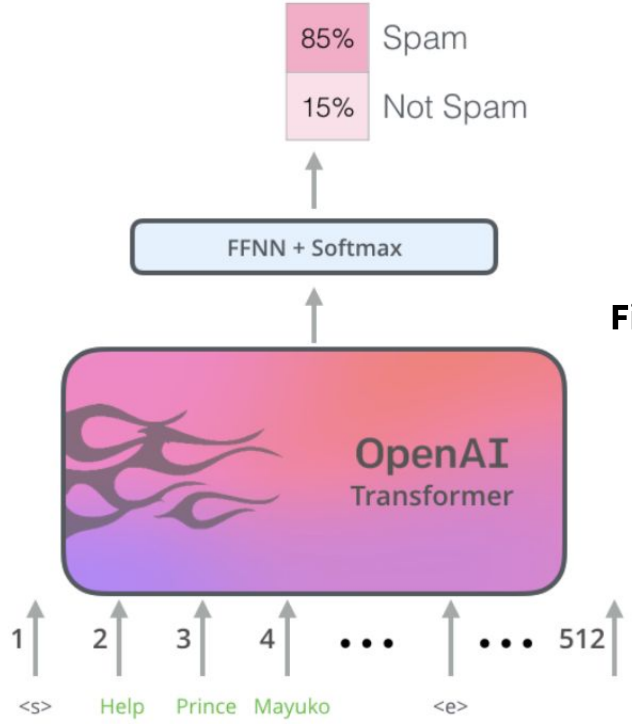


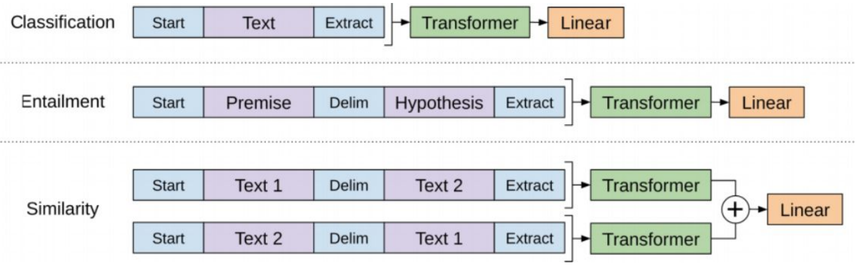
Pretraining :
(left-to-right) Language Modeling

Improving Language Understanding by Generative Pre-Training (Radford, OpenAI et al., 2018)
https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

The Illustrated BERT, ELMo, and co. (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-bert/

Transformer : OpenAI GPT





Finetuning with downstream tasks

Improving Language Understanding by Generative Pre-Training (Radford, OpenAI et al., 2018)
https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf

The Illustrated BERT, ELMo, and co. (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-bert/

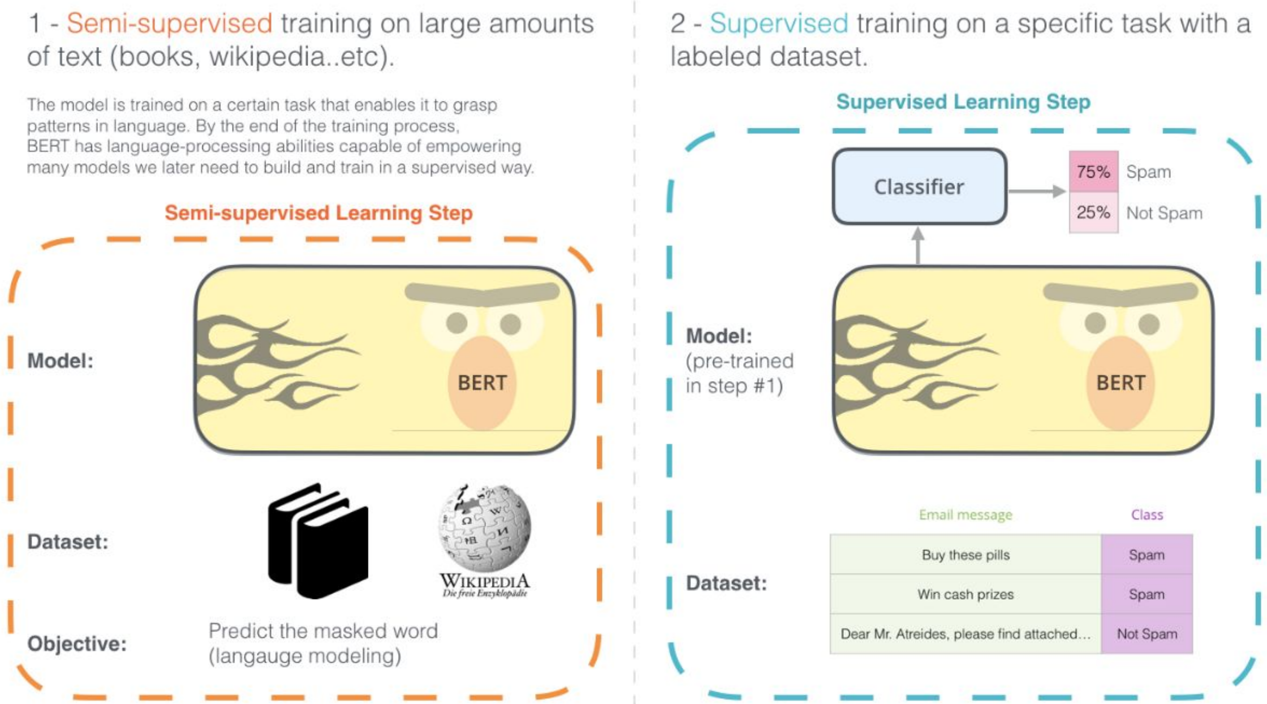
Transformer : Open AI GPT-2

- 10+ times **Larger Data** size (~40GB)
- 10+ times **Larger Model** size (1.5B parameters in 48 Transformer Blocks)
- Much **Better Generation** Capacity
- **Zero-Shot** Transfer Performance

Language Models are Unsupervised Multitask Learners (Radford et al., OpenAI, 2019)
https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

Better Language Models and Their Implications
https://openai.com/blog/better-language-models/#sample8

Transformer : Google BERT



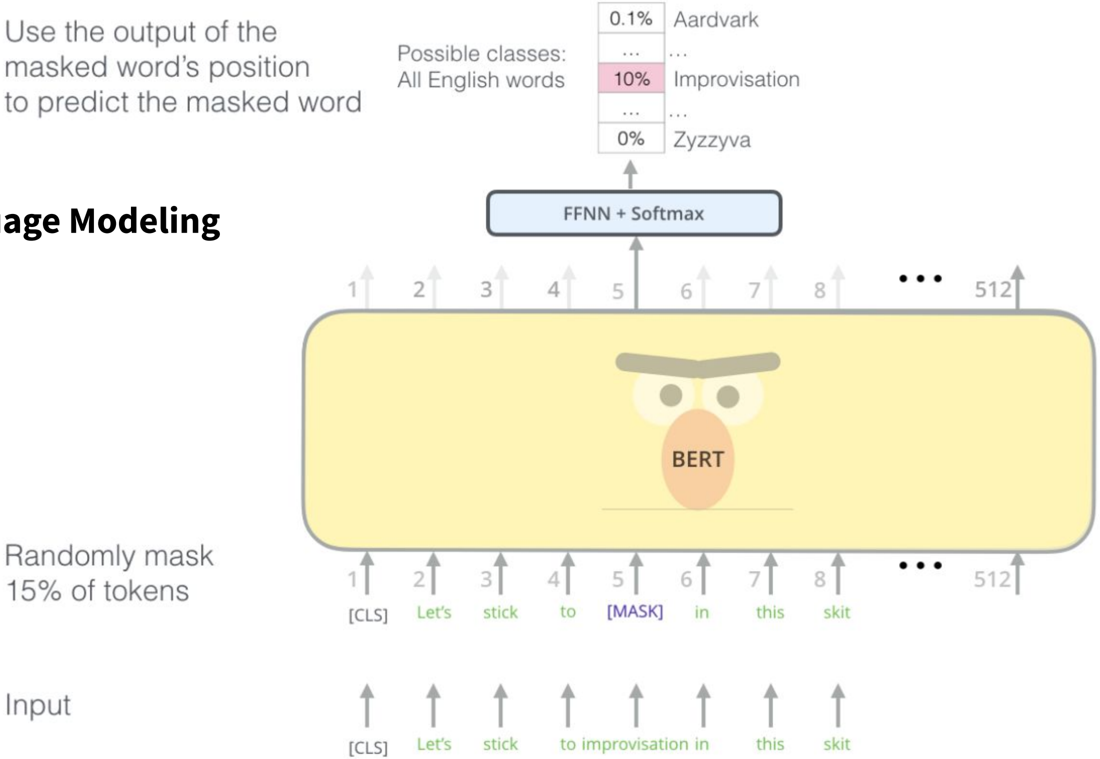
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [Source for book icon].

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., Google, NACCL 2019 Best Long Paper)
https://arxiv.org/pdf/1810.04805.pdf

The Illustrated BERT, ELMo, and co. (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-bert/

Google BERT : Pretraining

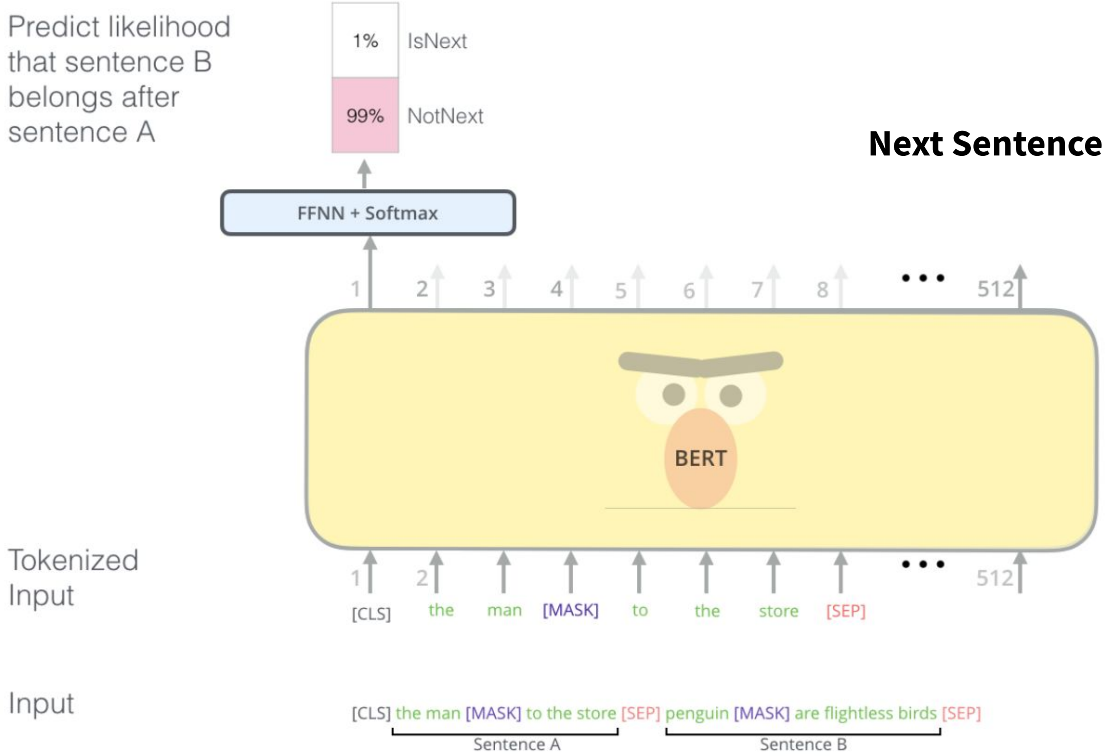
Masked Language Modeling



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., Google, NACCL 2019 Best Long Paper)
https://arxiv.org/pdf/1810.04805.pdf

The Illustrated BERT, ELMo, and co. (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-bert/

Google BERT : Pretraining



Next Sentence Prediction

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al., Google, NACCL 2019 Best Long Paper)
https://arxiv.org/pdf/1810.04805.pdf

The Illustrated BERT, ELMo, and co. (Jay Alammar, 2018)
http://jalammar.github.io/illustrated-bert/

Google BERT : Pretraining

Masked Language Modeling

```
# For masked Language Models
masked_tokens, masked_pos = [], []
# the number of prediction is sometimes less than max_pred when sequence is short
n_pred = min(self.max_pred, max(1, int(round(len(tokens)*self.mask_prob))))
# candidate positions of masked tokens
cand_pos = [i for i, token in enumerate(tokens)
            if token != '[CLS]' and token != '[SEP]']
shuffle(cand_pos)
for pos in cand_pos[:n_pred]:
    masked_tokens.append(tokens[pos])
    masked_pos.append(pos)
    if rand() < 0.8: # 80%
        tokens[pos] = '[MASK]'
    elif rand() < 0.5: # 10%
        tokens[pos] = get_random_word(self.vocab_words)
```

https://github.com/dhlee347/pytorchic-bert

Google BERT : Pretraining

```python
class BertModel4Pretrain(nn.Module):
    "Bert Model for Pretrain : Masked LM and next sentence classification"
    def __init__(self, cfg):
        super().__init__()
        self.transformer = models.Transformer(cfg)
        self.fc = nn.Linear(cfg.dim, cfg.dim)
        self.activ1 = nn.Tanh()
        self.linear = nn.Linear(cfg.dim, cfg.dim)
        self.activ2 = models.gelu
        self.norm = models.LayerNorm(cfg)
        self.classifier = nn.Linear(cfg.dim, 2)
        # decoder is shared with embedding layer
        embed_weight = self.transformer.embed.tok_embed.weight
        n_vocab, n_dim = embed_weight.size()
        self.decoder = nn.Linear(n_dim, n_vocab, bias=False)
        self.decoder.weight = embed_weight
        self.decoder_bias = nn.Parameter(torch.zeros(n_vocab))
```

```python
def forward(self, input_ids, segment_ids, input_mask, masked_pos):
    h = self.transformer(input_ids, segment_ids, input_mask)
    pooled_h = self.activ1(self.fc(h[:, 0]))
    masked_pos = masked_pos[:, :, None].expand(-1, -1, h.size(-1))
    h_masked = torch.gather(h, 1, masked_pos)
    h_masked = self.norm(self.activ2(self.linear(h_masked)))
    logits_lm = self.decoder(h_masked) + self.decoder_bias
    logits_clsf = self.classifier(pooled_h)

    return logits_lm, logits_clsf
```

https://github.com/dhlee347/pytorchic-bert

Google BERT official codes

Introduction

BERT, or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, is a new method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing (NLP) tasks.

Our academic paper which describes BERT in detail and provides full results on a number of tasks can be found here: https://arxiv.org/abs/1810.04805.

- `BERT-Large, Uncased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased (New, recommended)` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)` **(Not recommended, use `Multilingual Cased` instead)**: 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese` : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

* Official Repository
* Tensorflow with TPU
* 코드가 상당히 복잡함
* 여러가지 pre-trained model 제공
* 다국어 버젼있지만 성능 별로
* github star 15000+

https://github.com/google-research/bert

Hugging Face’s Pytorch Implementation

PyTorch Pretrained BERT: The Big & Extending Repository of pretrained Transformers

PASSED

This repository contains op-for-op PyTorch reimplementations, pre-trained models and fine-tuning examples for:

- Google's BERT model,
- OpenAI's GPT model,
- Google/CMU's Transformer-XL model, and
- OpenAI's GPT-2 model.

* 제일 유명한 pytorch impl
* 쉽게 가져다 쓸 수 있음
* 여러 GPU로 큰 모델 학습가능
* FP16 support (V100에서 속도업)
* pretraining code 없음
* github star 6000+

https://github.com/dhlee347/pytorchic-bert

Hugging Face’s Pytorch Implementation

```
import torch
from pytorch_pretrained_bert import BertTokenizer, BertModel, BertForMaskedLM

# OPTIONAL: if you want to have more information on what's happening, activate the
import logging
logging.basicConfig(level=logging.INFO)

# Load pre-trained model tokenizer (vocabulary)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Tokenized input
text = "[CLS] Who was Jim Henson ? [SEP] Jim Henson was a puppeteer [SEP]"
tokenized_text = tokenizer.tokenize(text)

# Mask a token that we will try to predict back with `BertForMaskedLM`
masked_index = 8
tokenized_text[masked_index] = '[MASK]'
assert tokenized_text == ['[CLS]', 'who', 'was', 'jim', 'henson', '?', '[SEP]', 'j

# Convert token to vocabulary indices
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
# Define sentence A and B indices associated to 1st and 2nd sentences (see paper)
segments_ids = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]

# Convert inputs to PyTorch tensors
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])
```

https://github.com/dhlee347/pytorchic-bert

Pytorchic BERT

Pytorchic BERT

This is re-implementation of Google BERT model [paper] in Pytorch. I was strongly inspired by Hugging Face's code and I referred a lot to their codes, but I tried to make my codes **more pythonic and pytorchic style**. Actually, the number of lines is less than a half of HF's.

(It is still not so heavily tested - let me know when you find some bugs.)

```
cuda (8 GPUs)
Iter (loss=0.308): 100%|████████████████████████████████████████| 115/115 [01:19<00:00,  2.07it/s]
Epoch 1/3 : Average Loss 0.547
Iter (loss=0.303): 100%|████████████████████████████████████████| 115/115 [00:50<00:00,  2.30it/s]
Epoch 2/3 : Average Loss 0.248
Iter (loss=0.044): 100%|████████████████████████████████████████| 115/115 [00:50<00:00,  2.33it/s]
Epoch 3/3 : Average Loss 0.068
```

* 매우 간결한 pytorch 구현
* 가져다 쓰기보단 고쳐서 쓰도록 기능 최소화, 미니멀리즘(?)
* Fine-tuning 매우 간단함
* github star 170+

https://github.com/dhlee347/pytorchic-bert

Pytorchic BERT : finetuning code

https://github.com/dhlee347/pytorchic-bert/blob/master/classify.py

https://github.com/dhlee347/pytorchic-bert

감사합니다

Question & Answering