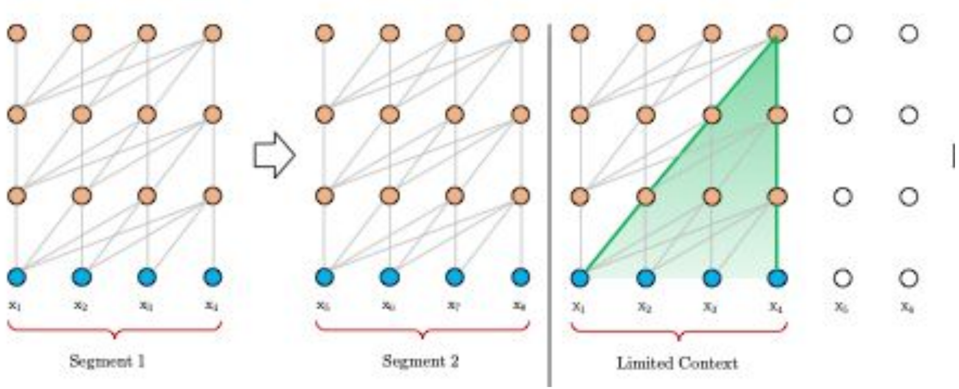
Problem Statement

Attention is not recurrent, it can only deal with fixed-length context

Context fragment: 기존 모델의 fixed length는 long term dependency 해결 못함



TransformerXL

Problem Statement

Transformer보다 긴 Longer dependency 해결

RNN 대비 80%, vanilla Transformer대비 450%

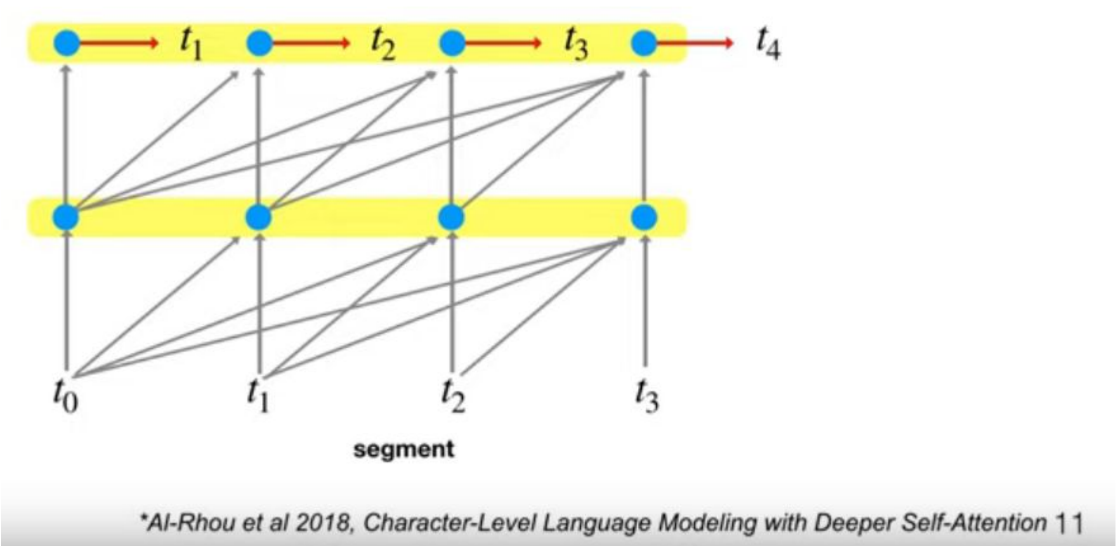
Article Generation으로 천 단어 정도의 토큰 생성을 가능

Related Work

- Truncated Back Propagation Through Time (Truncated BPTT)
  - RNN 기반의 모델
  - 이전 batch에서 hidden states를 가져와 현 batch에 넘겨준다
- Character Transformer Model
  - Transformer를 Char로 접근
  - 매우 깊은 (64 Layer)를 쌓았음.

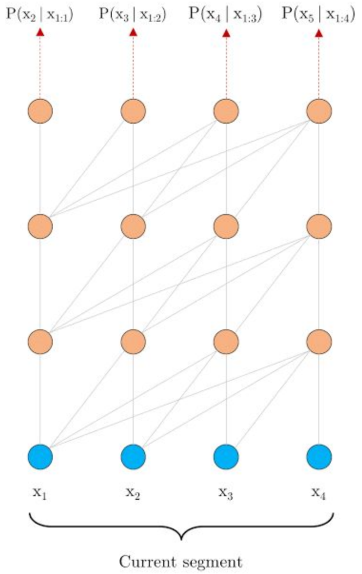
TransformerXL

Vanlia Transformer (Char Transformer)



TransformerXL

Valina Transformer: 정보가 segments 단위로 나눠져 전후 문맥을 볼 수가 없음



TransformerXL

Segment Recurrence

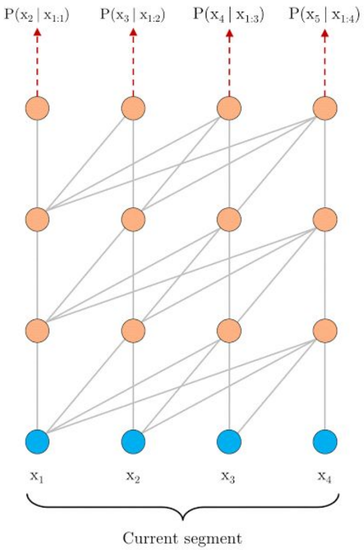
- 기존 Transformer의 단점: 고정된 길이의 문맥 정보만 활용

  -> 긴 컨텍스트를 보기 위해 세그먼트 리커런스라는 기법 제안

Relative Position Embedding

-> Segment Recurrence로 인핸 Absolute Position의 문제를 해결하고자 상대 길이 제안

XL의 문제 해결법: Segment Recurrence



TransformerXL

장점

1. **X9를 예측 할때, 정보가 없어 예측하기 힘든 것을 강화**
2. **기존에 이슈가 되었던 Seg끼리 정보가 전파가 안되는 점을 해결**
3. **No Grad (모든 히든을 Cashe로 가지고 있음) -> 계산이 빠름**

TransformerXL

관련 수식

Stop Gradient 후 Cache화 + 현재와 Concat

$$\widetilde{\mathbf{h}}_{\tau+1}^{n-1} = \left[\boxed{\mathrm{SG}(\mathbf{h}_{\tau}^{n-1})} \circ \mathbf{h}_{\tau+1}^{n-1}\right],$$

[Extended Context]

$$\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} = \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top},$$

[Query, Key, Value Vector]

$$\mathbf{h}_{\tau+1}^{n} = \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right).$$

[Self-attention + FF]

TransformerXL

Code

```python
def _update_mems(self, hids, mems, qlen, mlen):
    # does not deal with None
    if mems is None: return None

    # mems is not None
    assert len(hids) == len(mems), 'len(hids) != len(mems)'

    # There are `mlen + qlen` steps that can be cached into mems
    # For the next step, the last `ext_len` of the `qlen` tokens
    # will be used as the extended context. Hence, we only cache
    # the tokens from `mlen + qlen - self.ext_len - self.mem_len
    # to `mlen + qlen - self.ext_len`.
    with torch.no_grad():
        new_mems = []
        end_idx = mlen + max(0, qlen - 0 - self.ext_len)
        beg_idx = max(0, end_idx - self.mem_len)
        for i in range(len(hids)):

            cat = torch.cat([mems[i], hids[i]], dim=0)
            new_mems.append(cat[beg_idx:end_idx].detach())

    return new_mems
```

```python
def forward(self, w, r, r_w_bias, r_r_bias, attn_mask=None, mems=None):
    qlen, rlen, bsz = w.size(0), r.size(0), w.size(1)
    # W : [36, 4, 200]
    # r : [36, 1, 200] if mems is None else [72, 1, 200]
    # mems : [36, 4, 200]

    if mems is not None:
        cat = torch.cat([mems, w], 0)
        # cat : [72, 4(batch_size), 200]
        if self.pre_lnorm:
            w_heads = self.qkv_net(self.layer_norm(cat))
        else:
            w_heads = self.qkv_net(cat)
        r_head_k = self.r_net(r)

        # w_heads : [72, 4, 12]ㄴ
        w_head_q, w_head_k, w_head_v = torch.chunk(w_heads, 3, dim=-1)
        w_head_q = w_head_q[-qlen:]
```

Relative Position Embedding

기존 **Positional Embedding**으로는 해결이 되지 않음 **(**위치가 겹침**)**

[0,1,2,3]	[0,1,2,3]	[0,1,2,3]
Segment 1	**Segment 2**	**Segment 3**

상대적 위치를 계산하여 문제 해결! **(key vector**와 **query vector (i - j)**

$$\mathbf{A}_{i,j}^{\mathrm{rel}} = \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(b)}$$

$$+ \underbrace{u^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{(c)} + \underbrace{v^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{(d)}.$$

Relative Position Embedding

기존

$$\mathbf{A}_{i,j}^{\text{abs}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

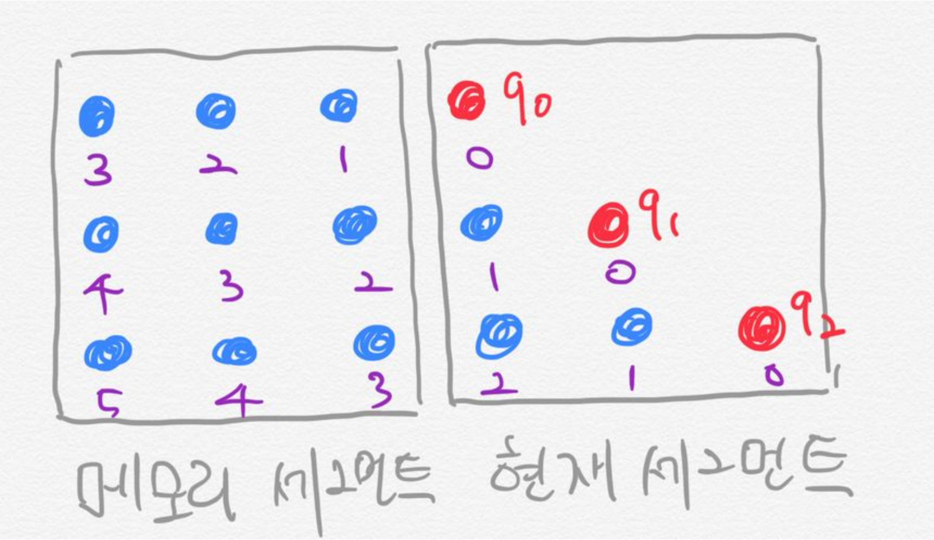
개선

$$\mathbf{A}_{i,j}^{\text{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

TransformerXL

Relative Position Embedding

발 없는 말이 천리 간다 -> [발, 없는, 말, 이, 천리, 간다]



메모리 세그먼트: [발, 없는, 말]

현재 세그먼트: [이, 천리, 간다]

q0: 이

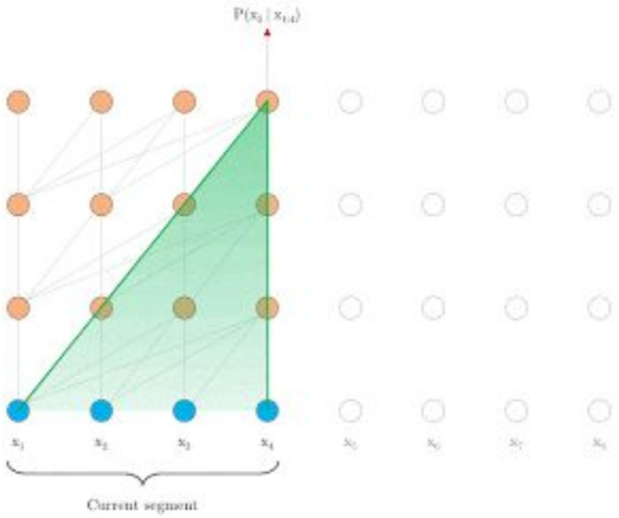
q1: 천리

q2: 간다

TransformerXL

Vanilla Prediction

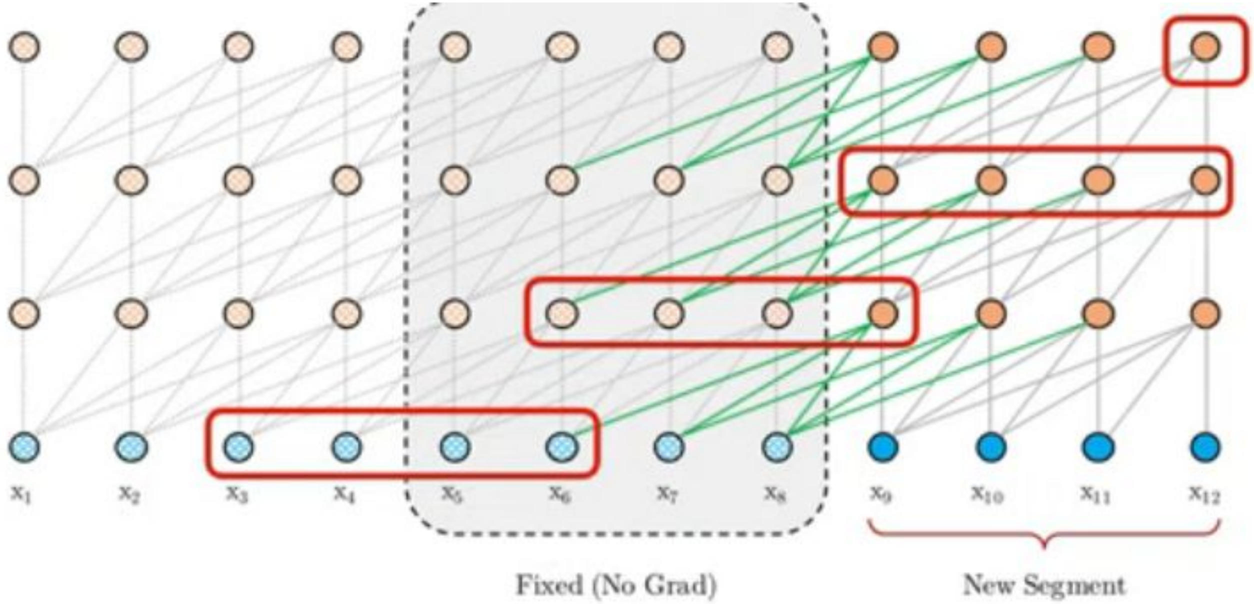
기존은 포지션 당 1개의 예측만 가능 (Extremly Expensive)



TransformerXL

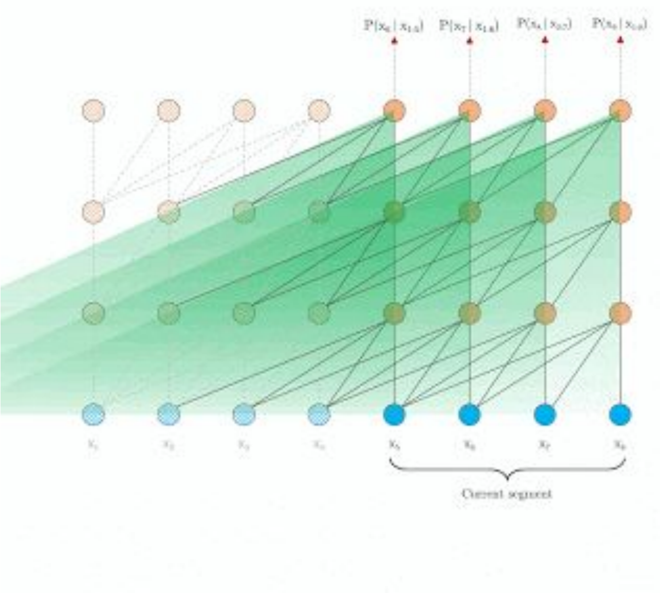
TransformerXL Prediction

이전 segments에서 메모리를 사용하기 때문에, 한 segments에 대한 결과가 한꺼번에 연산 가능



TransformerXL Prediction

이전 segments에서 메모리를 사용하기 때문에, 한 segments에 대한 결과가 한꺼번에 가능



Dataset 및 결과

- WikiText-103*
  - **Word-level** dataset with long-term dependency
  - 103M training tokens from 28K articles, average length of 3.6K tokens per article
- enwiki-8
  - 100M bytes of unprocessed Wikipedia text
- text-8
  - 100M processed Wikipedia **characters**
- One Billion Word
  - Shuffled sentences (**No long-term dependency**)

평가방법

- Perplexity (PPL)

$$PPL(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

- Bit Per Character (bpc)

$$BPC = Average(-\log_2 P(x_{t+1} \mid y_t))$$

- **Relative Effective Context Length (RECL)**
  - Effective Context Length* : longest length to which increasing the context span would lead to a gain more than a threshold
  - RECL : relative improvement over the best short context model

TransformerXL

평가 결과 (1)

Model	#Param	PPL
Grave et al. (2016b) - LSTM	-	48.7
Bai et al. (2018) - TCN	-	45.2
Dauphin et al. (2016) - GCNN-8	-	44.9
Grave et al. (2016b) - LSTM + Neural cache	-	40.8
Dauphin et al. (2016) - GCNN-14	-	37.2
Merity et al. (2018) - QRNN	151M	33.0
Rae et al. (2018) - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	**24.0**
Baevski and Auli (2018) - Adaptive Input◇	247M	20.5
Ours - Transformer-XL Large	257M	**18.3**

Table 1: Comparison with state-of-the-art results on WikiText-103. ◇ indicates contemporary work.

Model	#Param	bpc
Ha et al. (2016) - LN HyperNetworks	27M	1.34
Chung et al. (2016) - LN HM-LSTM	35M	1.32
Zilly et al. (2016) - RHN	46M	1.27
Mujika et al. (2017) - FS-LSTM-4	47M	1.25
Krause et al. (2016) - Large mLSTM	46M	1.24
Knol (2017) - cmix v13	-	1.23
Al-Rfou et al. (2018) - 12L Transformer	44M	1.11
Ours - 12L Transformer-XL	41M	**1.06**
Al-Rfou et al. (2018) - 64L Transformer	235M	1.06
Ours - 18L Transformer-XL	88M	1.03
Ours - 24L Transformer-XL	277M	**0.99**

Table 2: Comparison with state-of-the-art results on enwik8.

LM모델

Best

Char-Level도 좋은 성능

평가 결과 (2)

Model	#Param	bpc
Cooijmans et al. (2016) - BN-LSTM	-	1.36
Chung et al. (2016) - LN HM-LSTM	35M	1.29
Zilly et al. (2016) - RHN	45M	1.27
Krause et al. (2016) - Large mLSTM	45M	1.27
Al-Rfou et al. (2018) - 12L Transformer	44M	1.18
Al-Rfou et al. (2018) - 64L Transformer	235M	1.13
Ours - 24L Transformer-XL	277M	**1.08**

Table 3: Comparison with state-of-the-art results on text8.

Model	#Param	PPL
Shazeer et al. (2014) - Sparse Non-Negative	33B	52.9
Chelba et al. (2013) - RNN-1024 + 9 Gram	20B	51.3
Kuchaiev and Ginsburg (2017) - G-LSTM-2	-	36.0
Dauphin et al. (2016) - GCNN-14 bottleneck	-	31.9
Jozefowicz et al. (2016) - LSTM	1.8B	30.6
Jozefowicz et al. (2016) - LSTM + CNN Input	1.04B	30.0
Shazeer et al. (2017) - Low-Budget MoE	∼5B	34.1
Shazeer et al. (2017) - High-Budget MoE	∼5B	28.0
Shazeer et al. (2018) - Mesh Tensorflow	4.9B	24.0
Baevski and Auli (2018) - Adaptive Input◇	0.46B	24.1
Baevski and Auli (2018) - Adaptive Input◇	1.0B	23.7
Ours - Transformer-XL Base	0.46B	23.5
Ours - Transformer-XL Large	0.8B	**21.8**

Table 4: Comparison with state-of-the-art results on One Billion Word. ◇ indicates contemporary work.

Char-level 모델

Long-term + Short-term 보다 좋음

TransformerXL

Inference 속도의 차이 - 학습과 모델 크기의 한계는 있지만 Generation으로는 최적화 모델이 아닐까?

Model	$r = 0.1$	$r = 0.5$	$r = 1.0$
Transformer-XL 151M	**900**	**800**	**700**
QRNN	500	400	300
LSTM	400	300	200
Transformer-XL 128M	**700**	**600**	**500**
- use Shaw et al. (2018) encoding	400	400	300
- remove recurrence	300	300	300
Transformer	128	128	128

Table 8: Relative effective context length (RECL) comparison. See text for the definition of RECL and r. The first three models and the last four models are compared as two *model groups* when we calculate RECL (RECL is computed on a model group rather than a single model). Each group has the same parameter budget.

How Long

기존 Char-transformer

Attn Len	How much Al-Rfou et al. (2018) is slower
3,800	1,874x
2,800	1,409x
1,800	773x
800	363x

Table 9: Slowdown in terms of running time during evaluation. Evaluation is based on per-token time on one GPU.

How Fast

TransformerXL

Limitation (Paper 거절 이유)

Better language 모델이지만, downstream task에서의 자료가 없음

Document 생성 잘 된다고 했는데 없음..

OpenGPT2가 관련 모델을 이겼음

↓

XLNet