Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu

JMLR 2020

오지은

index

논문의 목적과 의의

T5: 목표

- idea: 모든 NLP 문제(understanding, generation 무관)를 “text-to-text” 문제로 환원하겠다

→ 같은 모델, objective, training procedure, decoding process를 모든 문제에 적용할 수 있음

→ “unified” approach

- 이 접근법으로 여러 learning objective, dataset 등을 비교해볼 수 있음
- 비교를 통해 transfer learning의 한계를 탐구할 수 있음

T5: 목표

"As such, the bulk of our work comprises of a survey, exploration, and empirical comparison of existing techniques."

- 논문의 목적은 새로운 모델을 제시하는 것이 아니라 기존의 연구와 방법론을 통틀어 정리하고 조합하여 한계를 넓히는 것

Baseline

baseline

- 그간의 연구들을 비교하기 위해 알맞은 baseline을 정함
- “Our goal is to compare a variety of different approaches on a diverse set of tasks while keeping as many factors fixed as possible.”
- 예를 들어, 같은 transformer 구조라도 enocder-only 모델과 decoder-only 모델을 곧바로 비교할 수는 없음
- architecture: **Transformer** (standard: encoder-decoder 구조)
  - 12 layer each
  - d_ff = 3072
  - d_model = 768
  - head = 12
  - 합해서 BERT-base의 2배 정도 parameter를 가지며, 2배인 것은 인코더와 디코더로 되어 있기 때문

baseline

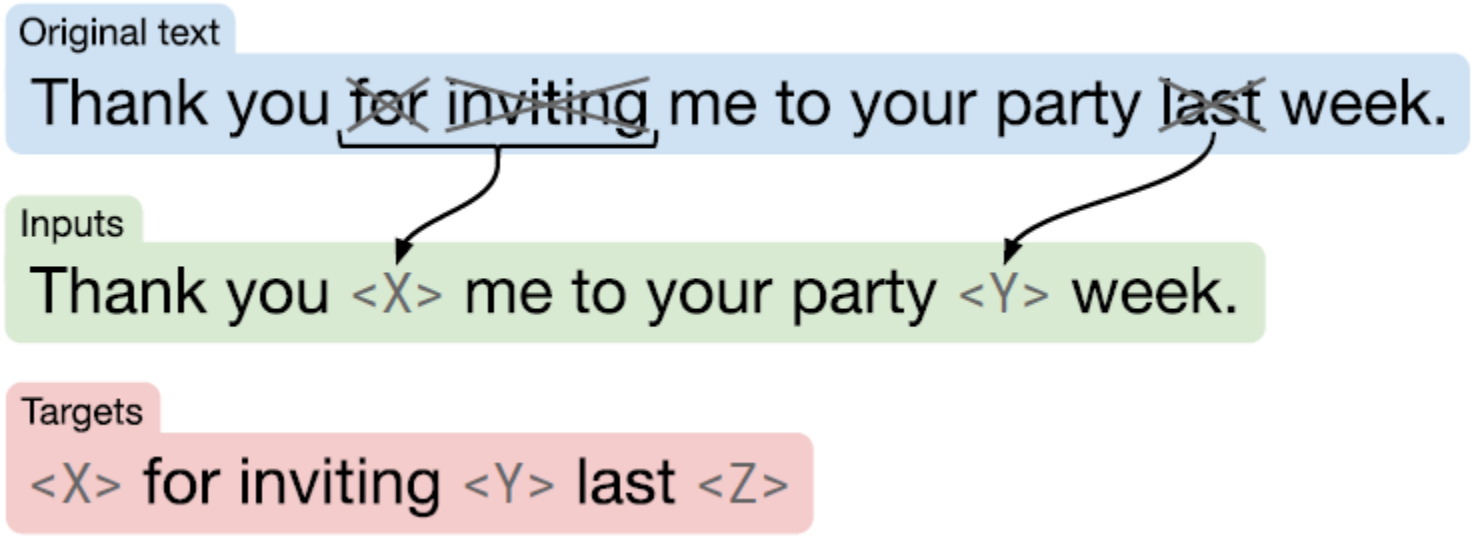
- training:
  - 2^19 = 524,288 step 동안 pre-train
  - 2^18 = 262,144 step 동안 fine-tune
- vocabulary
  - sentencepiece 사용 → wordpiece token
  - vocab size: 32K
  - task들 중에 번역 문제도 있기 때문에, 영어 이외에 독일어, 프랑스어, 루마니아어 단어도 있음
  - shared vocabulary (predetermined, fixed set of languages만 커버 가능)

baseline

- Unsupervised objective
  - pre-train은 unlabeled dataset을 사용하기 때문에 unsupervised objective 필요
  - 채택된 것: denoising objective → 입력 문장의 랜덤 15% 토큰을 drop

⟨x⟩, ⟨y⟩: sentinel token
'for', 'inviting'은 연속되므로 같은 토큰 ⟨x⟩로 대체

target: 입력에서 삭제되었던 토큰들. sentinel token ⟨x⟩, ⟨y⟩, ⟨z⟩는 delimiter로 사용됨



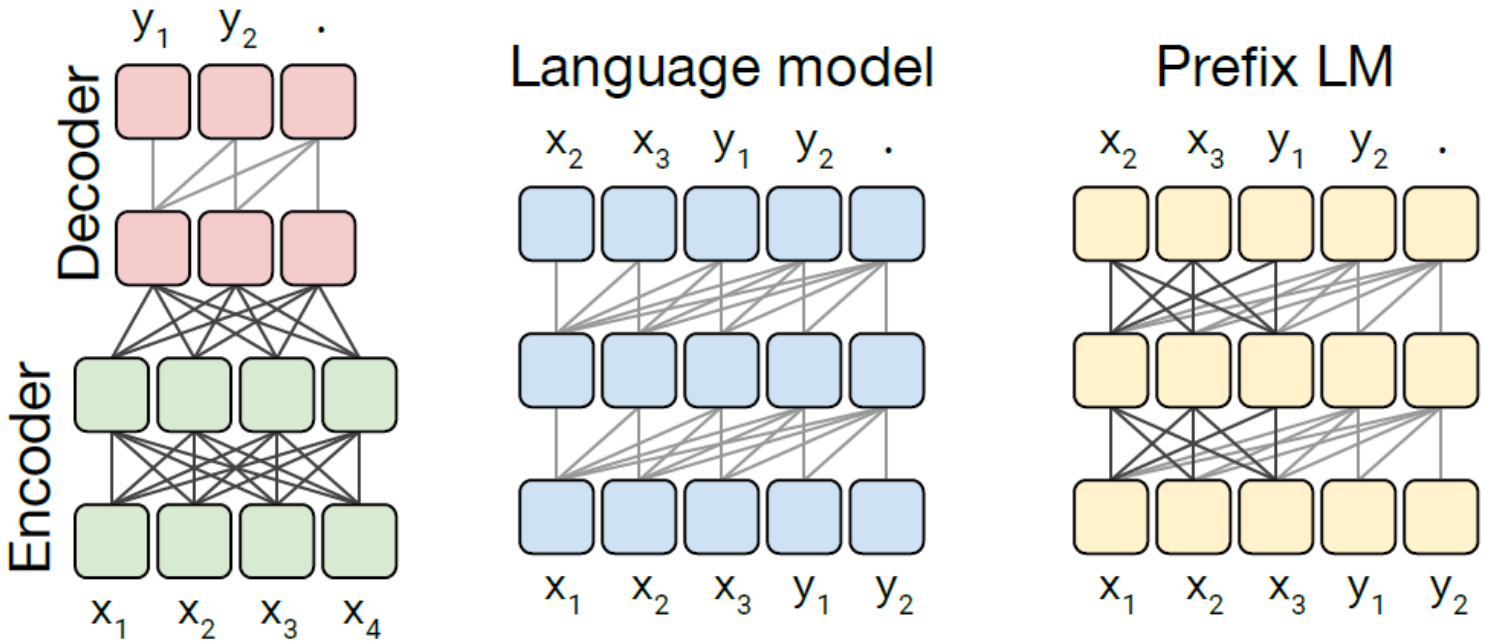
baseline

	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline average	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Baseline standard deviation	0.235	0.065	0.343	0.416	0.112	0.090	0.108
No pre-training	66.22	17.60	50.31	53.04	25.86	**39.77**	24.04

- 열 번 학습(different initialization, shuffling)해서 평균한 결과와 표준편차
- 전반적으로 현존하는 다른 비슷한 크기의 모델들과 비슷한 결과를 보임
- (놀랍지 않게도) pre-training은 성능을 크게 올려준다
  - 예외: WMT en-fr. 데이터가 충분히 크기 때문에 pre-training의 효과가 별로 없음

어떤 구조가 좋은가?

architectural variants – structure



- prefix LM: language model은 일반적으로 텍스트 생성에 쓰이지만, 입력과 타겟을 concat함으로써 text-to-text 프레임워크로 쓰일 수 있음
- **translate English to German: That is good. target:** Das ist gut. 에서 시작~target까지를 prefix로 고정하면 LM은 Das ist gut.을 출력
- 그런데 트랜스포머 deocer를 LM으로 사용하는 사례에서, 이 prefix 부분에 look-ahead mask를 적용하는 것은 불필요함
- → prefix 부분에 masking을 적용하지 않고 fully-visible self attention을 적용하는 것 = prefix LM
- 이 prefix는 classification에서는 BERT의 CLS 토큰과 같은 역할을 수행함

architectural variants - config for comparing

- 두 모델이 같은 개수의 parameter를 갖거나 (입력, 타겟) 쌍을 처리하는 데 드는 계산량이 비슷하다면 두 모델은 같다고 할 수 있다
- 그렇지만 encoder-decoder 모델을 decoder-only 모델과 바로 비교할 수는 없다
  - enc-dec 모델은 decoder 모델의 2배 파라미터를 가지지만, 계산량은 비슷하기 때문
  - 계산량이 비슷한 이유: 디코더 모델은 디코더 스택에 대해 입력과 타겟이 모두 적용되지만, 인코더-디코더 모델은 인코더는 입력만, 디코더는 타겟만 처리하기 때문
- 타당한 비교를 위해 여러 가지 configuration을 고려해보겠다
  - BERT-base 크기 모델의 레이어 개수를 L, 파라미터 수를 P로 표기
  - 모델이 (입력, 타겟) 쌍을 처리하는 데 든 계산량을 M으로 표기

architectural variants

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

1. 모든 문제에 대해, enc-dec에 denoising으로 학습한 모델이 가장 우수함
2. 인코더와 디코더의 파라미터를 공유한 경우도 1과 거의 비슷한 성능을 보임
3. 레이어 개수를 절반으로 줄일 때는 성능이 크게 떨어짐
4. 2는 deocder-only prefix LM 모델보다 성능이 좋음 → 명시적인 encoder-decoder attention이 성능에 유익
5. autoregressive LM보다 denoising으로 학습하는 쪽이 언제나 더 좋음

어떻게 pre-train할 것인가?

Unsupervised objective

- denoising objective가 LM objective보다 좋다면, noise를 어떻게 줄 것인가?

- high-level approaches
  1. **prefix LM**: 텍스트를 둘로 쪼갠 후, 하나를 입력으로 하나를 예측해야 할 타겟으로 함
  2. **MLM**: BERT 방식. MASK 토큰을 예측하되 BERT와 달리 타겟 문장 전체를 예측함
  3. **deshuffling**: 문장 내 토큰 순서를 뒤바꾼 뒤, 원래 순서의 문장을 예측함

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	**26.86**	39.73	**27.49**
BERT-style [Devlin et al., 2018]	**82.96**	**19.17**	**80.65**	**69.85**	**26.78**	**40.03**	**27.41**
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62

BERT-style이 가장 좋고 deshuffling이 가장 나쁨

denoising objective

- BERT-style objective가 가장 좋다면, 이것을 어떻게 개선할 것인가?

1. MASS-style: random 토큰을 제외하고, 15% 토큰을 마스크 토큰만으로 대체하며, uncorrupted sequence 전체를 재구성하도록 함
2. 타겟 전체를 예측하는 것은 너무 길다. 이러지 않을 방법이 있을까?
    1. consecutive span 전체를 a unique mask token으로 대체함 (앞에서 baseline에 사용된 그 방법)
    2. mask로 대체하는 것이 아니라 아예 삭제하고, 그 삭제된 부분을 예측하도록 함

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
I.i.d. noise, mask tokens	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

denoising objective

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
BERT-style [Devlin et al., 2018]	82.96	19.17	**80.65**	69.85	26.78	**40.03**	27.41
MASS-style [Song et al., 2019]	82.32	19.16	80.10	69.28	26.79	**39.89**	27.55
★ Replace corrupted spans	83.28	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**
Drop corrupted tokens	**84.44**	**19.31**	**80.52**	68.67	**27.07**	39.76	**27.82**

- 전체적으로 비슷하지만, consecutive span 전체를 a unique mask token으로 대체하는 방식이 가장 나음
- 원래 문장 전체가 아닌 노이즈 부분만 예측하는 쪽이 타겟을 짧게 만들기 때문에 학습 속도에서 유리함

corruption rate

Corruption rate	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
10%	**82.82**	19.00	**80.38**	69.55	**26.87**	39.28	**27.44**
★ 15%	**83.28**	19.24	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
25%	**83.00**	**19.54**	**80.96**	70.48	**27.04**	**39.83**	**27.47**
50%	81.27	19.32	79.80	70.33	**27.01**	**39.90**	**27.49**

- BERT의 기본 비율인 15% 이외의 여러 가지를 실험
- 비율에 따라 성능 차이가 별로 발생하지 않으나, 50%에서는 성능이 떨어짐
- 노이즈의 비율이 크면 타겟의 길이가 길어지기 때문에 계산량 측면에서도 불리함
- BERT의 선례를 따라 계속 15%를 써도 무방하다는 결론

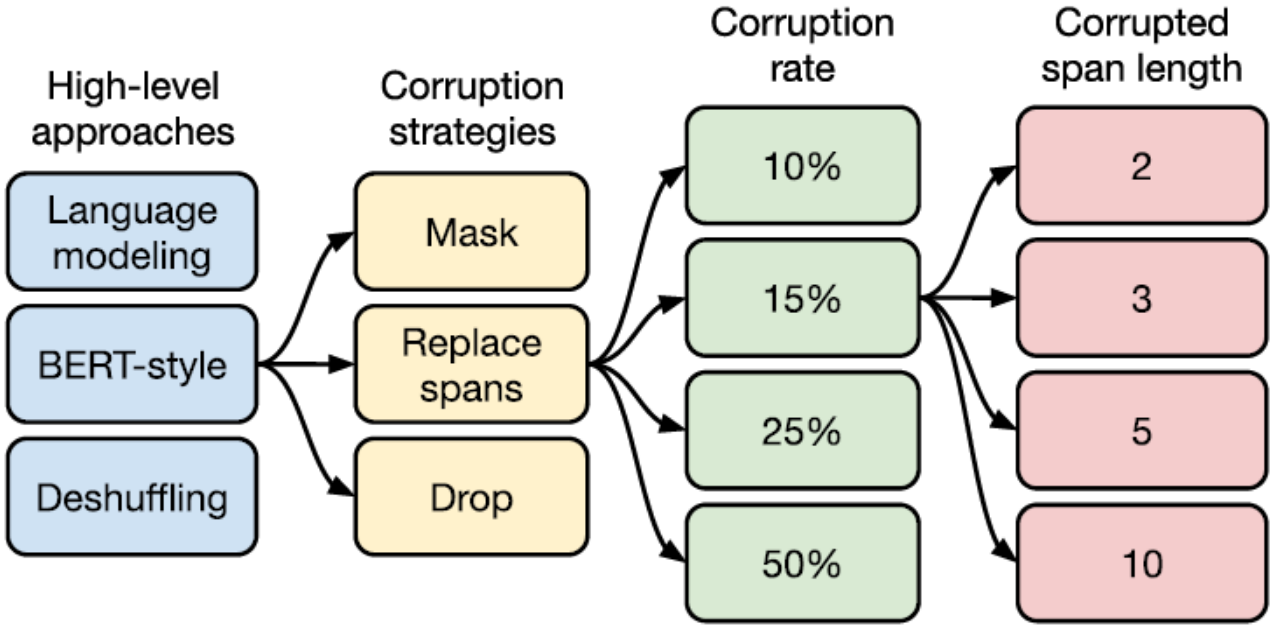
corruption span

- 예측해야 할 타겟의 길이를 줄임으로써 학습 속도를 올려야 함 → span 단위의 corruption
- baseline에선 각각의 토큰을 마스크할지 말지를 하나하나 정해서, 만일 마스크할 토큰이 연속적으로 나타나면 그것을 span으로 했음. 그러나 이런 식으로는 반드시 span이 생긴다는 보장이 없음
- contiguous, randomly-spaced spans of tokens를 corrupt하는 objective 필요
- 비율과 개수를 parameter로 두고 여기에 따라 span length가 정해짐

Span length	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline (i.i.d.)	**83.28**	19.24	80.88	71.36	**26.98**	**39.82**	**27.65**
2	**83.54**	19.39	**82.09**	**72.20**	**26.76**	**39.99**	**27.63**
3	**83.49**	**19.62**	**81.84**	**72.53**	**26.86**	39.65	**27.62**
5	**83.40**	19.24	**82.05**	**72.23**	**26.88**	39.40	**27.53**
10	82.85	19.33	**81.84**	70.44	**26.79**	39.49	**27.69**

별 차이 없지만 length 3에서 비교적 좋고, 타겟이 짧아지기 때문에 학습 속도가 약간 빠름

discussion



- LM, deshuflllling, denoising 중에서는 확연하게 denoising이 가장 나았음
- denoising 안에서 objective를 시도해 봤지만, 성능상으로 큰 차이는 없었음
- 유의미한 차이는 계산량과 학습 속도에서 나타남
- 이와 비슷한 방식의 objective를 더 탐구해봐도 성능상의 차이는 발견하기 힘들 것으로 보임

어떤 데이터셋이 좋은가?

dataset

- pre-training에서 데이터셋 자체가 중요한 요소임에도, 새로운 데이터셋은 일반적으로 중요한 contribution으로 취급되지 않았음
- pre-training에서 사용할 'standard' 데이터셋의 부재
- 데이터셋 간의 비교 연구도 많지 않음

→ 이 논문에서 사용된 데이터셋 C4와 다른 데이터셋을 비교해보겠다

+ C4 데이터셋 공개 (https://www.tensorflow.org/datasets/catalog/c4)

Unlabeled datasets

Dataset	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

- C4는 diverse한 데이터셋인데, 이에 따라 일부 task에서는 더 좁은 도메인의 데이터셋에서 더 우수한 성능이 나오기도 한다
  - ex: Wikipedia + TBC의 SuperGLUE 점수는 C4보다 높은데, 이것은 MultiRC 문제에서의 점수가 높기 때문이다. MultiRC는 소설책 데이터로 이루어진 독해력 테스트인데 TBC가 소설책 데이터이다.
  - 즉, in-domain 데이터로 pre-train을 하면 downstream task에서 더 높은 점수를 낼 수 있다

Pre-training dataset size

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full dataset	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

- single domain dataset은 종종 크기가 작음
- 데이터 크기에 따른 차이를 알아보기 위하여 인위적으로 C4의 크기를 줄여 실험
- (예상대로) 크기가 줄어들면 성능도 줄어듦
- 데이터셋이 작아질수록 모델의 loss가 현저히 줄어들며, 답을 암기했을 것으로 추정
- 가능한 한 큰 데이터셋을 사용하는 것이 언제나 좋다

어떻게 fine-tune할 것인가?

Training strategy

- 모델의 파라미터 전체를 fine-tuning하는 것은 (특히 low-resource 문제에서) 나쁜 결과를 낸다는 논란이 있음

→ 모델(인코더-디코더)의 일부만을 업데이트하는 접근에 집중하기로 함

1. **Adapter layers**: 각 블록의 끝에 추가적으로 붙는 additional feedforward network(dense-ReLU-dense). 원 모델의 대부분을 그대로 고정해두며, 이 추가 레이어와 layer normalization만 업데이트됨. 이 ffn의 차원 d가 파라미터
2. **Gradual unfreezing**: 맨 위부터 시작해서 인코더와 디코더를 병렬로 차근차근 unfreeze.

Training strategy

Fine-tuning method	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ All parameters	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Adapter layers, $d = 32$	80.52	15.08	79.32	60.40	13.84	17.88	15.54
Adapter layers, $d = 128$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Adapter layers, $d = 512$	81.54	17.78	79.18	64.30	23.45	33.98	25.81
Adapter layers, $d = 2048$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Gradual unfreezing	82.50	18.95	79.17	**70.79**	26.71	39.02	26.93

- SQuAD처럼 lower-resource task에서는 작은 d에서도 성능이 좋음
- 차원이 task 크기에 알맞게 scale된다면 적은 파라미터로 fine-tuning을 할 수 있음
- gradual unfreezing은 전반적으로 소폭 성능감소를 일으켰지만 어느 정도의 speedup이 있었음
- 더 optimal한 unfreezing schedule을 찾으면 성능 향상이 가능할 것으로 보임

multi-task learning

- multi-task learning: 한 모델을 여러 문제에 대해 한번에 학습하는 것
- 이 연구에서는 한 모델을 학습하되 downstream task에 따라 다른 checkpoint를 사용하는 것으로 함
- 이 text-to-text framework에서 multitask란 단순히 여러 데이터셋을 하나로 섞는 것을 말함

→ 데이터셋을 섞을 때 비율이 중요함

1. **Examples-proportional mixing**: 각 문제의 데이터셋 비율만큼 섞는 것. 단 특정 문제의 비중이 너무 커지지 않게 한도를 정함
2. **Temperature-scaled mixing**: 모델이 low-resource task에도 충분히 학습되도록 하는 것
3. **Equal mixing**: 모든 문제에 대해 같은 비율로 데이터셋을 섞는 것. 가장 나쁨

- multi-task learning 자체는 보통의 pre-train-then-fine-tune보다 성능 낮음

multi-task learning with fine-tuning

- 앞에서는 하나의 모델을 여러 문제에 학습한 후 각 문제에 맞는 checkpoint에서 성능을 평가했음
- 이제 모델이 모든 문제에 대해 pre-train된 후 각 문제에 맞게 fine-tune되는 케이스를 고려해보겠다 (MT-DNN에서 사용했던 방법)

1. Examples-proportional mixing 데이터셋으로 pre-train
2. 1과 같되, downstream task 중에 하나를 빼고 pre-train하고, 그 뺀 하나에 대해 fine-tune
3. 1에서 unsupervised pre-train task를 빼고 pre-train

multi-task learning with fine-tuning

Training strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Unsupervised pre-training + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	27.65
Multi-task training	81.42	**19.24**	79.78	67.30	25.21	36.30	27.76
Multi-task pre-training + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80	**28.07**
Leave-one-out multi-task training	81.98	19.05	79.97	**71.68**	**26.93**	39.79	**27.87**
Supervised multi-task pre-training	79.93	18.96	77.38	65.36	26.81	**40.13**	**28.04**

- multi-task training 후 fine-tuning을 하면 보통의 pre-train-then-fine-tune과 비슷한 성능이 나옴
- leave-one-out의 성능 하락폭이 크지 않음 → 여러 문제에 대해 학습한 모델은 새로운 문제에 적응할 수 있음
- 데이터셋에서 unsupervised 부분을 뺀 결과 성능이 크게 떨어지지만 번역에서는 별로 떨어지지 않음 → english only pretrain은 번역에 별로 도움이 되지 않지만, 다른 문제들에서는 중요함

Scaling strategy

- 일반적으로, 학습의 규모를 키우면 성능은 올라간다
- 그 규모를 어떻게, 어느 방향으로 키울 것인가?

1. 모델 2배로 키우기
2. 모델 4배로 키우기
    1. 4배 학습하기
    2. 2배 학습하고 모델 2배 키우기
    3. 모델 4배 키우기
3. 각각 따로 학습한 모델 4개의 ensemble
4. pre-train은 한 번만 하고 fine-tune을 4가지로 한 ensemble (3에서 계산량을 절약함)

Scaling strategy

Scaling strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× size, 4× training steps	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× size, 4× batch size	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× size, 2× training steps	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× size, 1× training steps	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled, fine-tune only	84.05	19.57	82.36	71.55	27.55	40.22	28.09

- 4배 더 학습하는 것과 batch를 4배 더 키우는 것 둘 다 비슷하게 유익함
- 모델 크기를 늘리는 것이 학습량만 늘리는 것보다 유익함
- ensemble 역시 도움이 됨 (특히 번역에서 다른 방법보다 나음)

벤치마크 최대로 끌어올리기

pushing the limit

- objective: baseline의 토큰 하나하나 noising하던 것을 앞에서 발견한 span-corruption으로 바꿈
- training: pre-training, batch size 키우기, step 수 늘리기 셋 모두 유익
- model size: 모델의 크기가 크면 좋지만, 자원이 한정되어 있다면 작은 모델도 유익
    1. small (transformer-base와 같음)
    2. base (기본. BERT-base와 같음)
    3. large, *d*model: 1,024, *d*ff = 4,096, *d*kv = 64, 16-headed attention, 12 layer
    4. 3B, 11B: *d*model = 1024, 24 layer, *d*ff = 16,384 with 32-head(3B), dff = 65,536 with 128-head(11B)
- beam search: baseline에는 greedy decoding을 사용했으므로, beam search 적용
- optimizer, learning rate, dropout rate 등의 hyperparameter는 baseline과 동일

results

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	**97.1**[a]	**93.6**[b]	**91.5**[b]	**92.7**[b]	**92.3**[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**89.7**	**70.8**	**97.1**	91.9	89.2	92.5	92.1

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	**74.8**[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	74.6	90.4	**92.0**	**91.7**	96.7	**92.5**	**93.2**

Model	SQuAD EM	SQuAD F1	SuperGLUE Average	BoolQ Accuracy	CB F1	CB Accuracy	COPA Accuracy
Previous best	88.95[d]	94.52[d]	84.6[e]	87.1[e]	90.5[e]	95.2[e]	90.6[e]
T5-Small	79.10	87.24	63.3	76.4	56.9	81.6	46.0
T5-Base	85.44	92.08	76.2	81.4	86.2	94.0	71.2
T5-Large	86.66	93.79	82.3	85.4	91.6	94.8	83.4
T5-3B	88.53	94.95	86.4	89.9	90.3	94.4	92.0
T5-11B	**90.06**	**95.64**	**88.9**	**91.0**	**93.0**	**96.4**	**94.8**

Model	MultiRC F1a	MultiRC EM	ReCoRD F1	ReCoRD Accuracy	RTE Accuracy	WiC Accuracy	WSC Accuracy
Previous best	84.4[e]	52.5[e]	90.6[e]	90.0[e]	88.2[e]	69.9[e]	89.0[e]
T5-Small	69.3	26.3	56.3	55.4	73.3	66.9	70.5
T5-Base	79.7	43.1	75.0	74.2	81.5	68.3	80.8
T5-Large	83.3	50.7	86.8	85.9	87.8	69.3	86.3
T5-3B	86.8	58.3	91.2	90.4	90.7	72.1	90.4
T5-11B	**88.2**	**62.3**	**93.3**	**92.5**	**92.5**	**76.1**	**93.8**

Model	WMT EnDe BLEU	WMT EnFr BLEU	WMT EnRo BLEU	CNN/DM ROUGE-1	CNN/DM ROUGE-2	CNN/DM ROUGE-L
Previous best	**33.8**[f]	**43.8**[f]	**38.5**[g]	43.47[h]	20.30[h]	40.63[h]
T5-Small	26.7	36.0	26.8	41.12	19.56	38.35
T5-Base	30.9	41.2	28.0	42.05	20.34	39.40
T5-Large	32.0	41.5	28.1	42.50	20.68	39.75
T5-3B	31.8	42.6	28.2	42.72	21.02	39.94
T5-11B	32.1	43.4	28.1	**43.52**	**21.55**	**40.69**

results

- 24 task 중 17개에서 SOTA
- (예상대로) 가장 큰 11B 모델이 가장 성능 좋음
  - 하지만 다른 SOTA 모델들도 많은 계산량을 요하는데, 예를 들어 ALBERT 모델은 T5-3B 모델과 크기가 비슷하며 ensemble에 사용하는 비용은 T5-11B 모델을 넘을 수도 있다
- 번역에 대해서는 아무것도 SOTA 성능이 나오지 않았음. pre-training에 영어 데이터만을 사용했기 때문으로 추정
- 또한 SOTA 번역 모델들은 backtranslation 등 정교한 기법을 사용하기 때문에 scale과 pretrain만으로는 그런 기법을 쫓아가지 못했을 것으로 추정

결론

결론

- **Text-to-text**: 이 프레임워크는 하나의 모델로 많은 task에 대해 학습할 방법이다. 이 프레임워크로 classification, generation, regression까지 학습할 수 있다
- **architecture**: 원래의 encoder-decoder 구조가 가장 낫고, 인코더나 디코더 하나만을 쓰는 구조라고 해서 계산량이 줄어들지 않는다. 인코더와 디코더의 parameter를 공유하면 parameter를 반으로 줄이면서 성능은 조금만 낮아진다.
- **objective**: LM보다는 denoising이 좋고, denoising 안에서는 성능이 비슷하며, target sequence의 길이를 줄이는 span 방식이 비용이 적게 든다.
- **dataset**: 일부 task에서는 in-domain 데이터로 학습하는 게 유리하지만, 그럼에도 불구하고 큰 데이터셋(C4)이 좋다.
- **training**: fine-tuning 단계에서 모델의 모든 parameter를 업데이트하는 편이 가장 좋지만 가장 비용이 크다.
- **scaling**: 큰 모델을 더 많은 데이터에 학습하고 ensemble을 사용하면 성능이 올라간다.

End of Document