MT-DNN 발표

- 모두연 풀림스쿨 NLP Bootcamp 5th
- 백영상

---

0. Abstract

- MT-DNN = Multi-Task Deep Neural Network 의 약자
- 여러 NLU task의 데이터를 통해서 general representation을 표현하는 게 목적
  - 단지 많은 데이터를 활용하는 것이 아니라, 여러 상이한 task를 이용하여 regularization effect를 기대
    - 새로운 task와 domain에 적용하는 데에 효과적으로 대처할 수 있도록 하기 위함 (domain adaptation)
  - 이 논문의 base는 기존 연구 Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval를 BERT 버전으로 업데이트 했다고 보면 됨.

---

1. Introduction

- vector space representation을 학습하는 방법.
  - *Multi-task learning*
  - *Language model pre-training*
- 이 논문에서는 두 가지를 잘 융합하는 것의 힘을 보여줌.
- MTL(multi-task learning)의 효과 비유
  - "사람이 스키타는 법"을 배운다고 하자. 이 때 스케이팅을 타본 사람과 아닌 사람 두 명을 비교해보면, 스케이팅 탄 사람이 배우기 쉽다
  - 즉, 여러 관련 있는 task를 같이 학습하면, 하나의 task에서 얻은 지식으로 인해 다른 task를 잘 수행할수 있지 않을까? -> 여기서 착안
- MTL에 대한 관심 증가 이유 2가지
  - Supervised 학습 (label이 붙어 있는 학습)은 데이터 양이 많이 필요하고 항상 이용가능하지 않다. (cost가 많이 발생)
  - MTL은 특정 task에 overfiitting되어 학습되는 효과를 regularization을 하는 효과가 있다.
- Language Model (pre-training) 의 효과
  - 대표적으로 ELMO, GPT, BERT가 있음.
- 논문의 핵심 주장은 MTL와 Language Model pre-training의 방법을 상호보완적으로 활용하여 학습에 녹이도록 하는 것.
  - MTL의 확장 차원에서 BERT를 도입: BERT를 shared text encoding layer로 사용
  - BERT와 유사점: pre-training과 fine-tunning 두 가지 step으로 학습하고 domain adaptation시 fine-tuning형태로 학습
  - BERT와 차이점: fine-tunning에 대해서는 MTL의 방법을 사용.
  - 새로운 방식의 효과 - label이 적은 task에 대해 BERT보다 MT-DNN을 사용하는 것이 훨씬 유리함

    - GLUE (General Language Understanding Evaluation) 데이터셋 실험 결과: 8 sota / 9 dataset - 82.7% (+2.2% BERT보다 우수함)
    - 훨씬 적은 수의 데이터를 적용해서도 우수한 결과를 얻음: SNLI, SciTail 데이터 0.1%, 1% training data로도 우수한 결과

---

2. Tasks

- MT-DNN은 4가지 NLU task를 결합함: 모두 GLUE 데이터셋에 포함
    - single-sentence classification
    - text similarity scoring
    - pairwise text classification
    - relevance ranking
- 적용 Task별 분류 데이터셋: GLUE

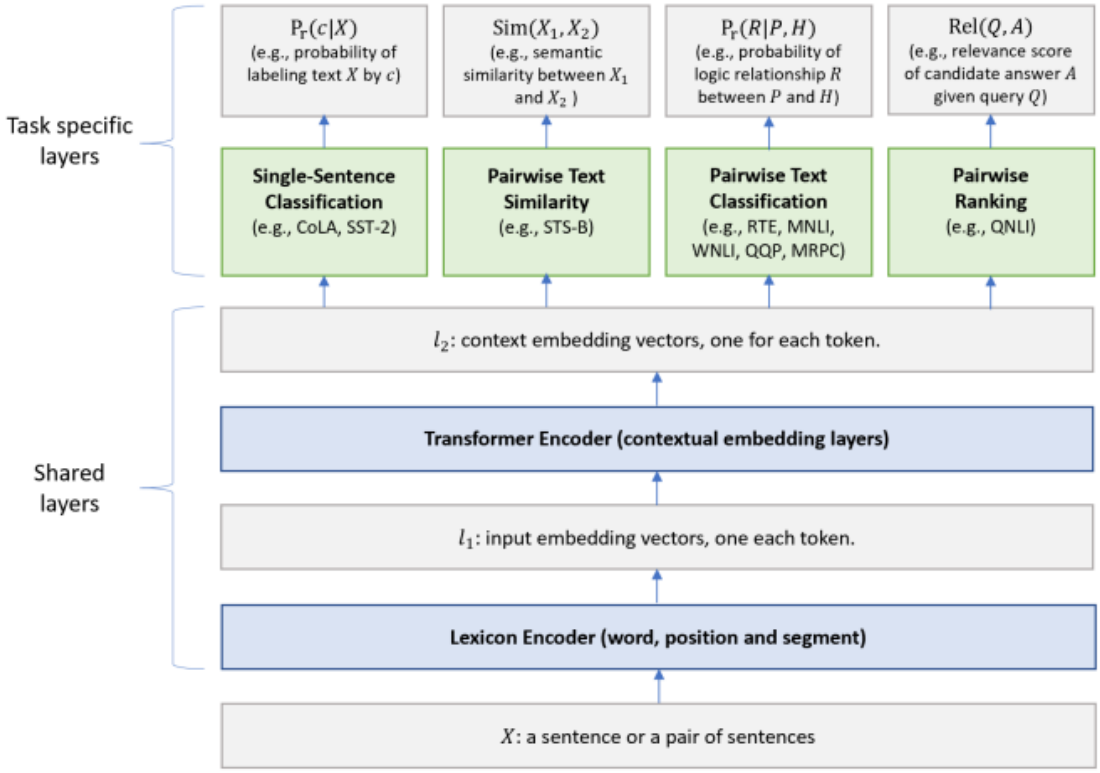
```
## https://github.com/namisan/mt-dnn/blob/master/data_utils/task_def.py
class TaskType(IntEnum):
    Classification = 1
    Regression = 2
    Ranking = 3
    Span = 4
    SeqenceLabeling = 5
```

- 각 type별 설명
    1. Single-sentence classification
        - CoLA: 문법적으로 맞는 지 긍 부정 (binary)
        - SST-2: sentiment 긍부정 문제
    2. Text similarity (regression)
        - STS-B: 문장 유사도 scoring (0~1의 값)
    3. Pairwise Text classification
        - RTE, MNLI: 문 장 두개의 의미 Entailment, contradiction, neutral 세 개 중에 고르기
        - QQP, MRPC: 문 장 두개의 의미가 같은 문장인지 O X
    4. Relevance Ranking
        - QNLI: SQuAD중 하나. Question Paragraph entailment / non-entailment 형태.
            - Question에 대해 올바른 정답을 할 수 있는지에 대한 문제
            - 3번 task에 넣지 않고 따로 pairwise-ranking task로 처리. 직접 처리하는 것보다 성능이 향상됨.
- cf. WNLI: 유일하게 sota를 못찍은 데이터셋. 데이터셋 자체의 문제가 있다는 언급

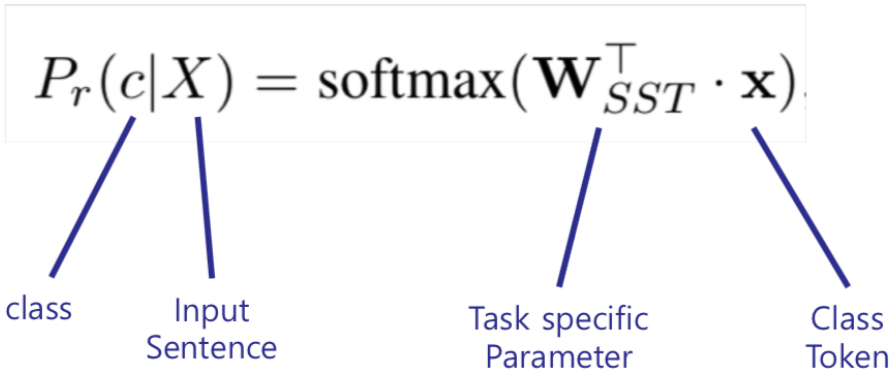
---

3. The Proposed MT-DNN Model

- 모델

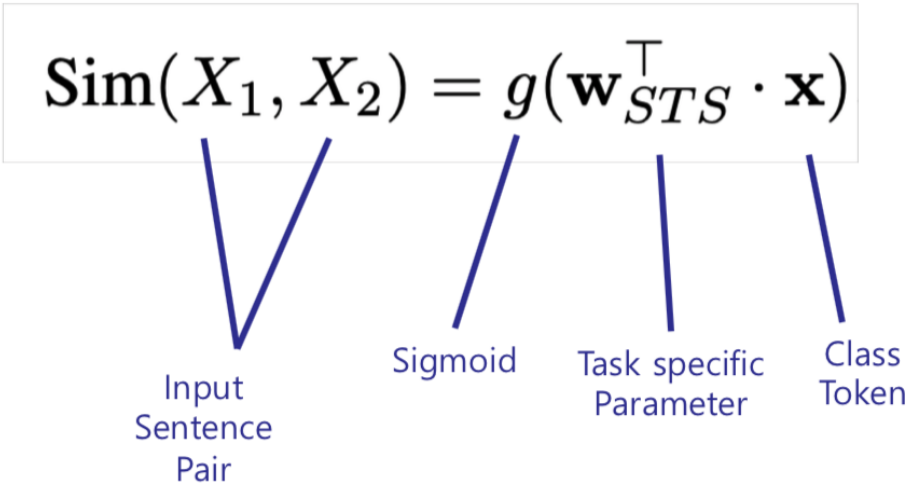


- shared layer 는 BERT, ROBERTA를 사용하였음 (v2) - huggingface BERT 사용
- 모델 부분 코드 참고 github - SANBertNetwork
- Layers
    1. Lexicon Encoder
        - 맨 앞에 [CLS] 토큰을 추가
        - 만약 문장쌍이 들어오면, 문장 사이에 [SEP] 토큰을 추가
        - Lexicon encdoer는 1. word-piece embedding, 2. segment embedding, 3. position embedding 3가지
        - BERT와 동일
        - Input 부분 코드 참고 github - batcher
    2. Transformer Encoder
        - Pre-training을 하는 부분
        - BERT에서 전달되는 embedding을 받아서 씀
        - 모델 부분 코드 참고 github - SANBertNetwork
    3. Task Specific layer
        - MTL로 fine-tunning
        - Loss 부분 코드 참조 github
        1. Single-Sentence Classification Output
            - [CLS]의 representation을 x라 했을 때, softmax로 logistic regression predict를 한다.



2. Text Similarity Output

- [CLS] token의 representation을 x라 했을 때, 다음과 같은 식으로 similarity를 구함.



3. Pairwise Text Classification Output

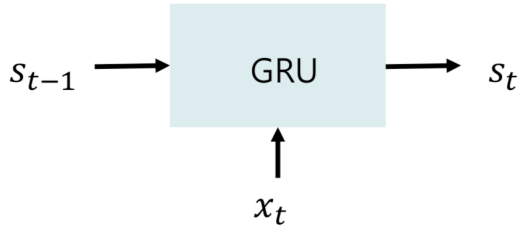
- 두 문장 간의 의미 관계 등을 분류 하기 위해서 Stochastic Answer Network(SAN)라는 것을 사용.
- SAN의 핵심 Idea는 Multi-Step Reasoning
- 즉, 1번에 Classification 결과를 예측하지 않고 여러번의 예측을 통한 Reasoning으로 결과를 예측하고자 하는 것. 즉, RNN으로 time step 마다 Classification 결과를 예측, 해당 결과들을 조합
- Premisie의 embedding 결과와 Hypothesis의 embedding을 잘 연관 짓는 형태
- 두 문장을 각각 Hypothiesis 문장, Premise 문장이라고 할 때, RNN(GRU)의 Input은 Hypothesis 문장을 그리고 Hidden State는 Premise 문장을 Embedding 하는데 사용
- 여기서 문장 Embedding이란 Transformer에 의해 생성된 Token Vector들의 Weighted Sum을 의미.

Hypothesis 문장 Token의 Vector

Premise 문장 Token의 Vector

$$\mathbf{s}^0 = \sum_j \alpha_j \mathbf{M}_j^h \qquad \mathbf{x}^k = \sum_j \beta_j \mathbf{M}_j^p$$

$$\alpha_j = \frac{\exp(\mathbf{w}_1^\top \cdot \mathbf{M}_j^h)}{\sum_i \exp(\mathbf{w}_1^\top \cdot \mathbf{M}_i^h)} \qquad \beta_j = \text{softmax}(\mathbf{s}^{k-1}\mathbf{W}_2^\top \mathbf{M}^p)$$



- 두 문장 자체의 Embedding Vector, 그리고 두 문장 간 관계(차의 크기와 Dot Product)를 concat하여 구성된 Vector를 활용하여 문장 간 관계를 분류

$$P_r^k = \text{softmax}(\mathbf{W}_3^\top [\mathbf{s}^k; \mathbf{x}^k; |\mathbf{s}^k - \mathbf{x}^k|; \mathbf{s}^k \cdot \mathbf{x}^k])$$

각 문장의 Vector

문장 간 거리

문장 간 Similarity

4. Relevance Ranking Output

- (Question, Paragraph)가 입력이고 이 때의 [CLS] embedding을 x라 할 때 다음과 같은 식을 따름.
- Relevance Ranking은 Question과 Paragraph Pair를 Input으로 넣어 생성한 [CLS] Token에 Sigmoid를 취하여 문장 별로 점수를 Scoring하고 가장 높은 점수만 Question에 해당하는 정답이 있다고 예측하는 방식으로 Classification을 수행

$$\text{Rel}(Q, A) = g(\mathbf{w}_{QNLI}^\top \cdot \mathbf{x})$$

Input Sentence Pair

Sigmoid

Task specific Parameter

Class Token

3.1 The Training Procedure

- 두 가지 학습 과정이 있다.
- Pretraining - Lexicon encoder과 Transformer encoder의 unsupervised 학습

    - BERT 방식
    - pre-trained BERT model을 가져다가 사용
- Multi-task learning state SGD로 학습.
    - 매 epoch마다 mini-batch를 선택하는데, 선택하는 방법은 모든 task(9개의 GLUE)의 데이터 세트를 모아다가 batch를 끄집어내는 식

Algorithm 1: Training a MT-DNN model.

Initialize model parameters Θ randomly.
Pre-train the shared layers (i.e., the lexicon encoder and the transformer encoder).
Set the max number of epoch: $epoch_{max}$.
//Prepare the data for T tasks.
for t *in* $1, 2, ..., T$ **do**
  Pack the dataset t into mini-batch: D_t.
end
for $epoch$ *in* $1, 2, ..., epoch_{max}$ **do**
  1. Merge all the datasets:
    $D = D_1 \cup D_2 ... \cup D_T$
  2. Shuffle D
  for b_t *in* D **do**
    //b_t is a mini-batch of task t.
    3. Compute loss : $L(\Theta)$
      $L(\Theta)$ = Eq. 6 for classification
      $L(\Theta)$ = Eq. 7 for regression
      $L(\Theta)$ = Eq. 8 for ranking
    4. Compute gradient: $\nabla(\Theta)$
    5. Update model: $\Theta = \Theta - \epsilon\nabla(\Theta)$
  end
end

1. Classification task
    - 여기서 1(X,c)는 맞는 class이면 1, 아니면 0
    - Cross-entropy loss 식을 따름
2. Similarity task
    - MSE loss 식을 따름
3. Relevance ranking task
    - Query Q가 주어지고, 후보 A list가 주어짐
    - A = [A+, A-]식으로 주어지고 positive sample A+은 한 개, negative sample A-은 |A|-1 개가 됨.
    - 이렇게 |A|개 sample에 대해 softmax식을 통하여 Negative log-likelihood loss을 정의함.

- batch 부분 코드 참고 github - batcher

---

4. Experiments

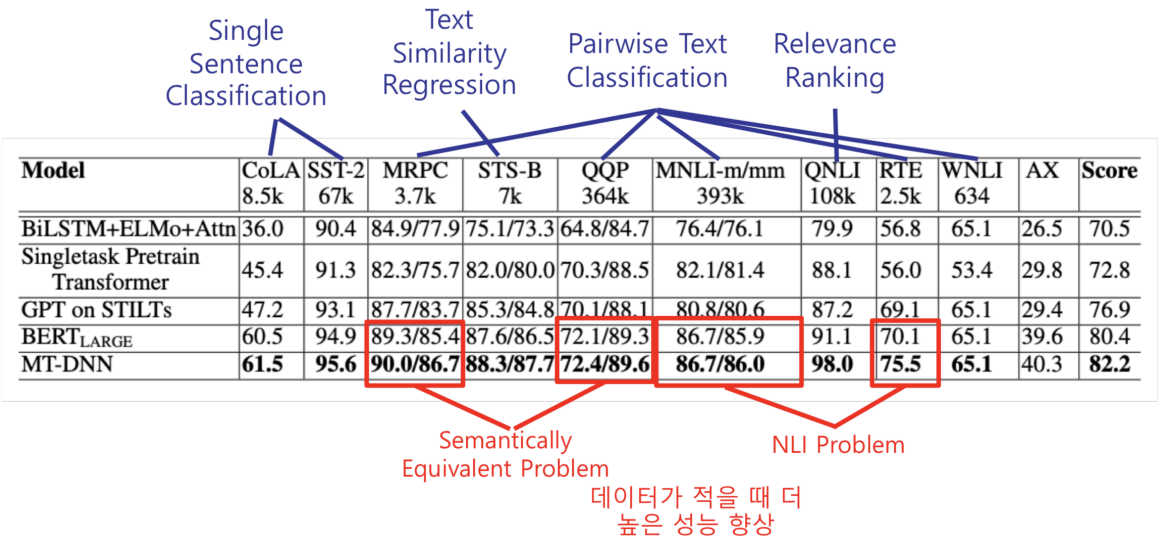
- 3개의 NLU benchmark을 사용
    - GLUE
    - SNLI
    - SciTail
- 각 데이터의 사용 용도
    - GLUE - Language Model fine tuning 대 MT-DNN 방식의 MTL이 효과적임을 보여주는 데이터로 사용

- SNLI 와 SciTail - 다른 domain adaptation을 하는 데에 있어서 MTL의 효과를 보여주는데 사용

4.1 Dataset

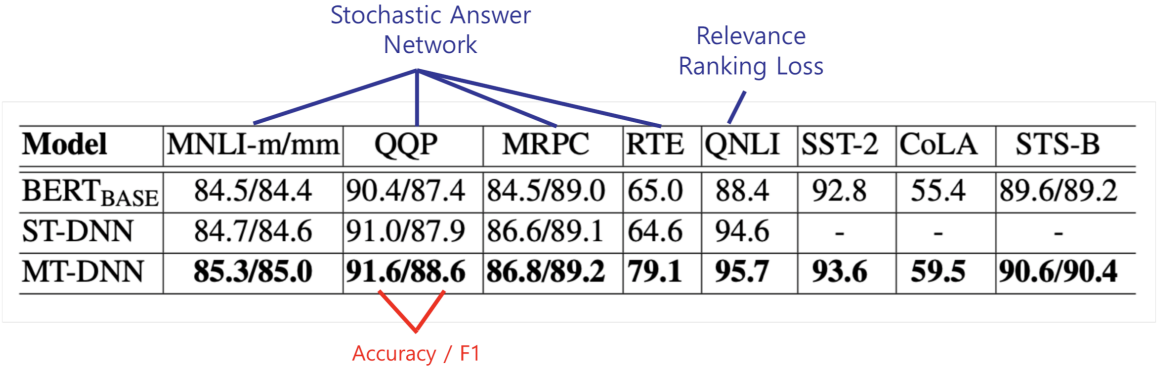
- GLUE 데이터셋 : 위에서 설명
- SNLI (Stanford Natural Language Inference)
  - NLI 데이터세트에서 보편적으로 사용되는 것
- SciTail (Science entailment)
  - Science question Answering(SciQ)에서부터 추출된 entailment dataset
  - P가 H를 entail 하는지 맞추는 문제.
  - 과학적인 문장들이기 때문에 다른 domain이라고 설정
  - 이 논문에서 Domain adaptation을 위한 데이터세트로 사용

4.3 Results



Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	Score
BiLSTM+ELMo+Attn	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	79.9	56.8	65.1	26.5	70.5
Singletask Pretrain Transformer	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	88.1	56.0	53.4	29.8	72.8
GPT on STILTs	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	87.2	69.1	65.1	29.4	76.9
$BERT_{LARGE}$	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	91.1	70.1	65.1	39.6	80.4
MT-DNN	**61.5**	**95.6**	**90.0/86.7**	**88.3/87.7**	**72.4/89.6**	**86.7/86.0**	**98.0**	**75.5**	**65.1**	40.3	**82.2**

- GLUE Task에 대한 성능 평가 결과
- MT-DNN은 BERT 보다 전체 성능이 약 1.8% 향상되어 82.2%로 현재(2019.05 기준) GLUE의 SOTA (표의 성능 지표 Accuracy 혹은 Accuracy/F1-score를 의미)
- 시사점: Dataset이 적은 Task(MRPC, RTE) 비교적 높은 성능 향상 실현.
  - 다른 Task를 통해 학습한 정보를 모델이 활용한 결과로 당연한 결과.



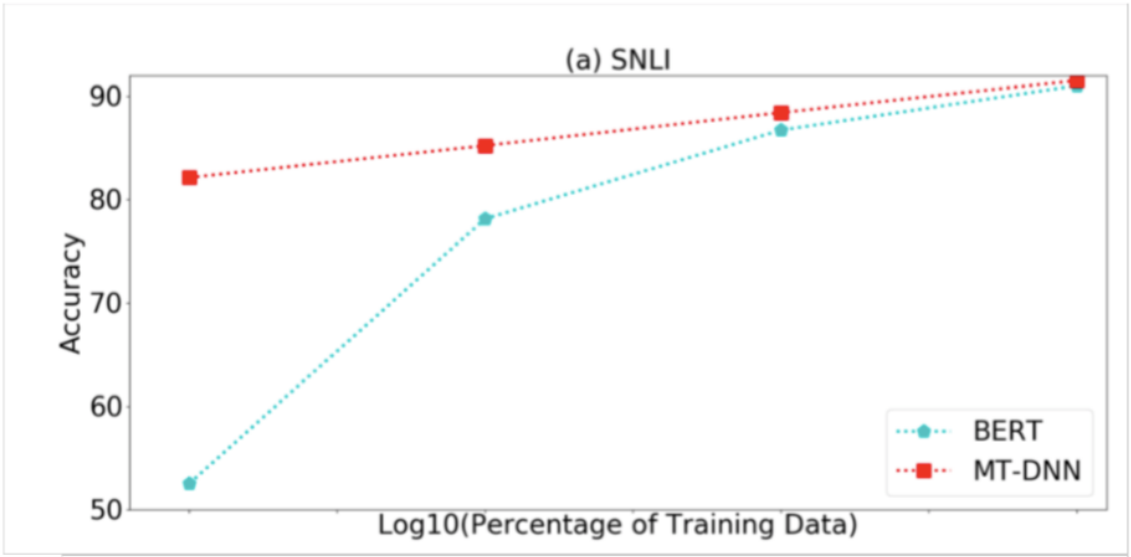
Model	MNLI-m/mm	QQP	MRPC	RTE	QNLI	SST-2	CoLA	STS-B
$BERT_{BASE}$	84.5/84.4	90.4/87.4	84.5/89.0	65.0	88.4	92.8	55.4	89.6/89.2
ST-DNN	84.7/84.6	91.0/87.9	86.6/89.1	64.6	94.6	-	-	-
MT-DNN	**85.3/85.0**	**91.6/88.6**	**86.8/89.2**	**79.1**	**95.7**	**93.6**	**59.5**	**90.6/90.4**

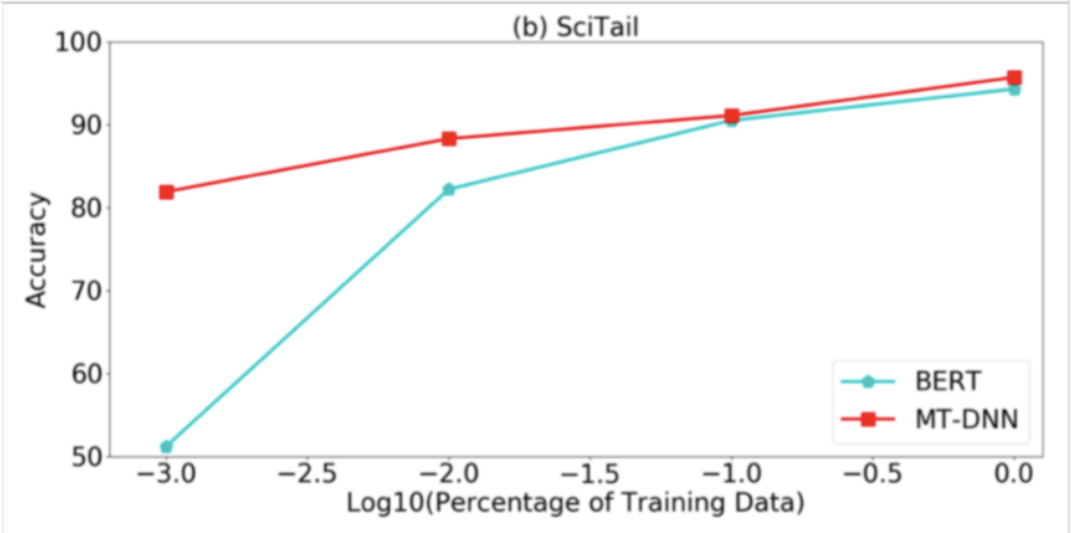
- SAN / Relavance Ranking 효과 검증 및 MT-DNN 검증
- ST-DNN은 Multi-task learning은 수행하지 않고 Task Speicific Layer만 바꾸어 수행한 결과

    - 따라서 Single Task만 돌렸을 경우 이론적으로 BERT 결과와 동일/유사 해야 함 - 3개 테스트 미실행
    - MNLI, QQP, MRPC에서 성능이 약간 차이 - SAN 효과
    - QNLI 성능 향상 - Relevance ranking 효과
- ST-DNN vs MT-DNN
    - 같은 task 기준으로 하나는 data가 많고 하나는 data가 적은 상황임.
    - 즉, unfair한 상황이라고 볼 수도 있는 상황
    - RTE vs MNLI을 보면 RTE에서는 MT-DNN이 훨씬 좋은데, Single과 Multi가 학습한 데이터량이 더 도드라질수록 점수차이가 확연함
- cf. 내부적으로 QNLI --> BioTail을 진행

4.4 SNLI and SciTail Results

- MT-DNN의 핵심 가치 : 새로운 domain task을 연구할 때, 빠르게 적용해보는 것이 중요
- 실전에서 당면하는 문제는, 실질적으로 label을 구축하여 training data를 확보하는 기간이 오래 걸리고 공수가 많이 드는 문제





- MT-DNN에서는 소량의 데이터 사용에도 우수한 결과가 나
- 다른 Natural Language Inference Supervised Task (Network과 Loss까지 같은!)로 MT-DNN은 이미 학습했고, BERT는 그렇지 않으므로 차이가 나는 것은 당연
- 같은 이유로 적은 데이터량으로 Fine Tuning 시 MT-DNN의 성능이 높은 것은 당연 - 학습 데이터량 차이

---

cf. 주목할만한 Multi-Task Learning 프로젝트

- Huggingface HMTL:
  - HMTL: Hierarchical Multi-Task Learning
  - Medium
    - MT-DNN처럼 Model을 사실상 고정해 놓고 데이터셋만 변형하는 식이 아니라, 모델과 데이터셋을 추가로 등록시킬 수 있음.
    - 한 모델의 결과를 받아 그 결과에 따라 2차적으로 결과를 내는 꼬리에 꼬리를 무는 형태의 multi-task leanrning 구현 가능
- NYU jiant:
  - jiant
    - A software toolkit for research on general-purpose text understanding models