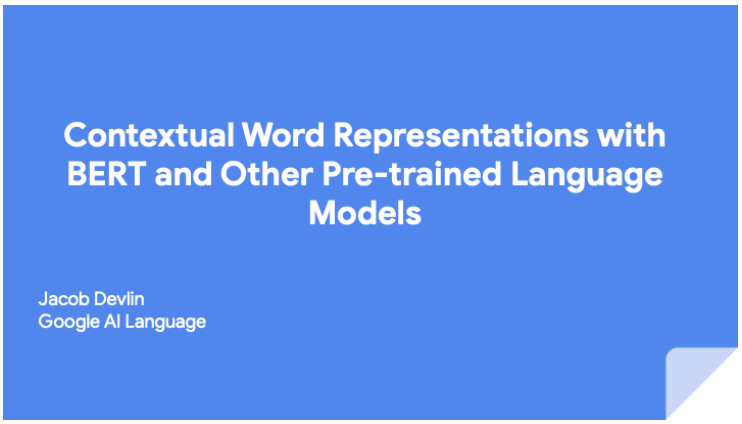
Natural Language Processing with DeepLearning

week 18

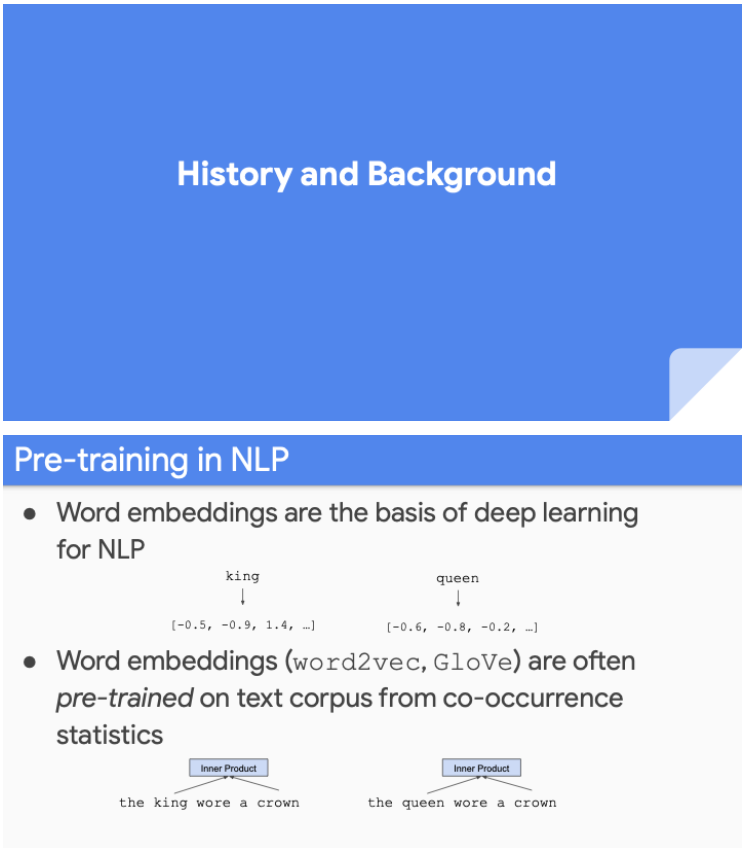
The course

BERT에서 Contextual Word Representation을 어떻게 학습했는지를 중심으로, BERT 이전의 연구와 BERT 이후의 연구들에 대한 내용

- ELMO와 GPT1보다 왜 BERT가 더 좋은 이유: 기본적으로 Bidirectional하게 학습하기 위해 Masked LM, Next sentence prediction task로 pretrain하였기 때문
- BERT 이후의 모델은 ALBERT, XLNET, T5, ELECTRA를 배우고 serving하기 위해서 Distillation이라는 방법을 사용해서 모델을 압축한다는 것을 배움



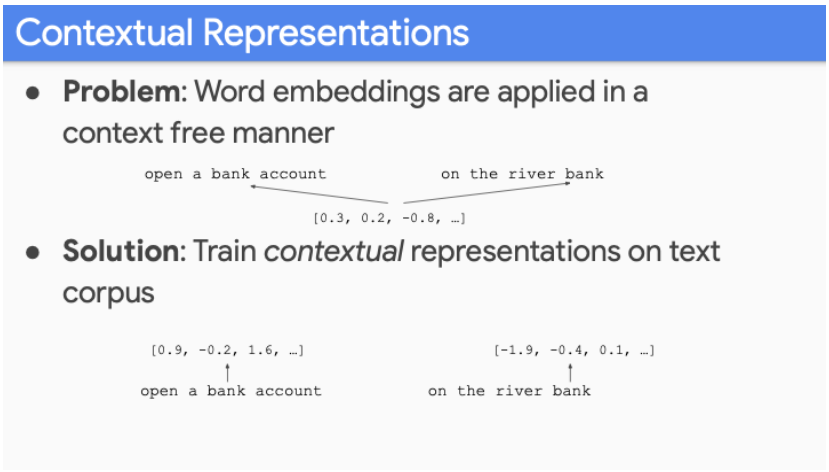
<BERT와 BERT 이전에 있었던 모델, 이후에 있었던 모델들>



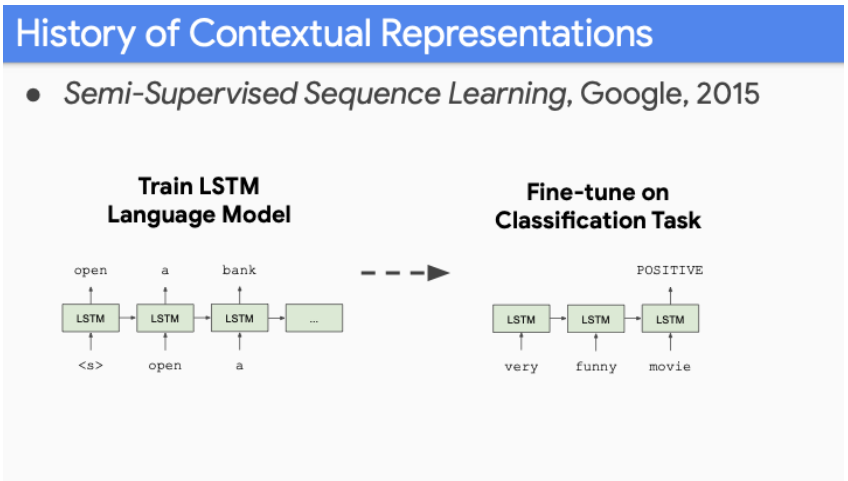
기본적으로, Word embedding이라는 것이 NLP에 Neural Net을 사용하는 것을 가능하게 함

- Neural Net은 continuous space(vectors)에서 동작하지만, text는 discrete space로 표현되므로 이 둘 사이의 gap을 연결시켜 줄 것이 필요!

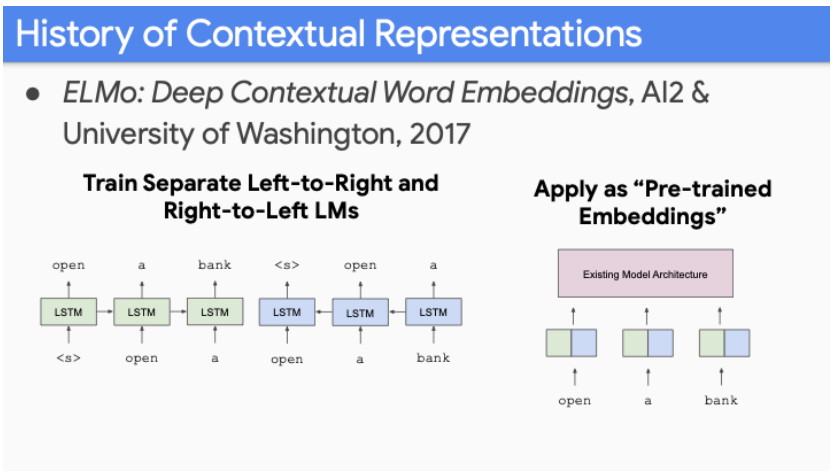
- 그 bridge가 set of discrete vocabulary에서의 look up table을 이용하여 word를 vector로 변환 하는 것
- Bengio 2003 neural language model paper에서 위의 word embedding이 end to end로 학습되는 것을 배웠고, 사람들은 이러한 pretrained representation을 사용
- 이후 word2vec, glove가 나왔고, 이 방법들은 학습할 때, 더 cheaper, scalable way였다. word2vec, glove를 이용하여 billions of token을 하나의 CPU로 돌릴 수 있게 됨



문제: embedding이 context-free 방식으로 적용되었다는 것
- 예를들어, bank가 위의 두 문장에서 같은 embedding값을 갖게 됨 => 단순한 single word가 아닌, word sense embedding과 같은 방식을 시도
- 대부분의 단어가 context에 따라 다른 의미를 가짐
'open the bank account', 'I went to the bank'라는 두개의 문장에 대해서도 semi-different sense가 존재한다. 그래서 우리는 contextual representation이 필요!

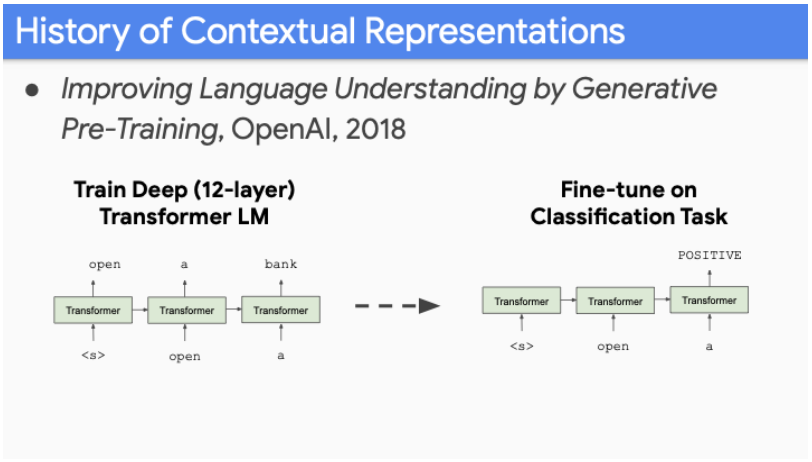


Contextual representation의 역사
- 이것에 대해 첫번째 big paper는 Semi-supervised Sequence Learning (from Andrew Dai)
: sentiment classification on movie review와 같은 classification task을 사용
  - Big corpus of movie review로부터 pretrained embedding을 사용하지 않고, LSTM만을 사용하여 model 전체를 (LM 처럼) pretrain하고, classification을 위해 fine tune한다
  - 좋은 결과가 나왔지만, 매우 좋지는 않음) Corpus, model size가 LM과 같은 매우 크지 않고, 기존의 size를 사용했기 때문)



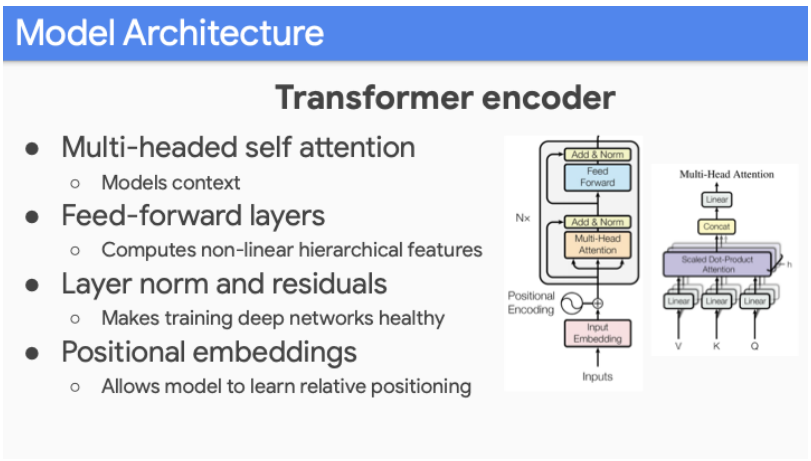
2017년 Washington univ에서의 ELMo

- LM을 big corpus(billion words)에 대해 big model (LSTM with 4000 hidden dimension with bidirectional model (왼쪽에서 오른쪽으로 학습하고, 오른쪽에서 왼쪽으로 학습한것을 concat))을 사용
- 그리고 이것을 contextual pretrained embeddings라 부름
- ELMO에 기저하는 아이디어는 your existing model architecture를 바꾸는 게 아니고, 너가 갖고 있는 task-specific model architecture에 대해 예전엔 Globe embedding을 사용했다면 이제는 ELmo embedding을 input으로 넣는 것
- 그 당시에 모든 task에서 state-of-art 를 달성! (state - of- art model에 ELMO만 embedding으로 넣어주면 되었기 때문)

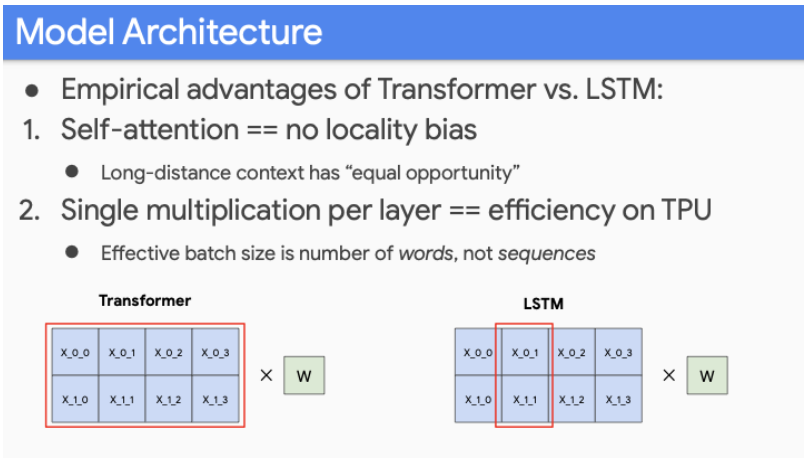


OpenAI가 2018년도에 Improving Language Understanding by Generative Pre-Training (GPT1)

- ELMO와 유사하게 large corpus로 large language model(12-layer)을 학습했다. 그 당시에 open source model중 가장 큰 모델
- 그 당시에 발표자분은 너무 크다고 생각했으나지금와서 생각해보면 이러한 depth(12-layer)가 crucial element 중 하나 였다. LM을 이용하여 fine tune한 후, 마지막 token으로 classification task를 했고, state-of-art 를 달성



BERT로 넘어가기 전에 transformer부터 짚고 넘어가겠다. BERT, GPT를 잘 작동하게 한 precursor 중에 하나



Transformer에서 가장 중요한 것: LSTM에 비해 두가지 advantage가 존재

1) 첫번째로 locality bias가 없음

- Long distance context는 short distance context와 동등한 기회를 가짐
- 보통의 Language Understanding: locality bias of LSTM은 좋은것으로 간주하는데, 이는 local context가 long-distance context보다 더 relevant하기 때문이다.
- 하지만 GPT, BERT 와 다른 model들은 context들을 concatenate 하여 학습
- 예를들어 sentence1이 sentence2 이후에 나오냐 와 같은 것
- LSTM은 이것에 대해 attention을 사용하여 encoder-decoder 구조를 사용
- transformer에서: same sequence로 간주하여 넣어서 각 sentence각 자신의 token 뿐만아니라 다른 sentence의 token에 대해서도 attend할 수 있게 됨

= 즉, 모든 sentence의 token에 대해 attend하기가 쉬워지고, 모델을 simplifying 하는데에 중요함

2) 위 슬라이드 처럼 4개의 단어를 갖는 두개의 문장이 있다면 LSTM에서는 각 단어가 한번에 하나씩만 학습하게 됨

- Modern hardware TPU, GPU에서는 matrix multiplication이 클수록 (값이 아니라 크기) 더 좋음 (빨라진다).
- LSTM에서는 작은 size의 batch를 사용할 수 밖에 없으나(sequence 개수에 따른 batch size) Transformer에서는 batchsize가 total number of words
- 만약 500개의 단어로 이루어진 32개의 문장이 있으면 batch size는 500512가됨 (발표자는 32512이라 했는데, max_length(word size)가 batch size가 되어야 함)
- 그러면 huge matrix multiplication을 하게 되고, modern hardware의 이점을 모두 사용할 수 있게 됨

<BERT>

BERT

Problem with Previous Methods

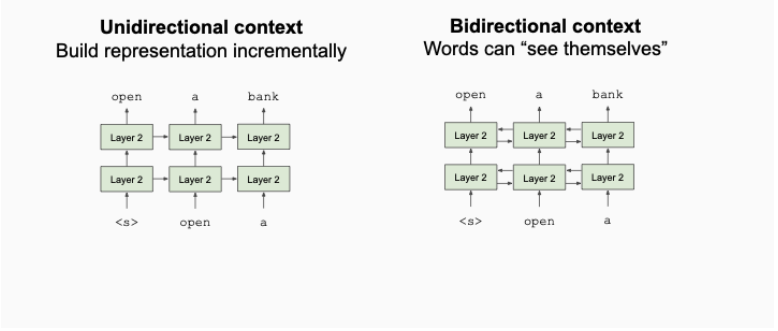
- **Problem**: Language models only use left context *or* right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - We don't care about this.
- Reason 2: Words can "see themselves" in a bidirectional encoder.

ELMO와 GPT가 갖고있던 문제점들

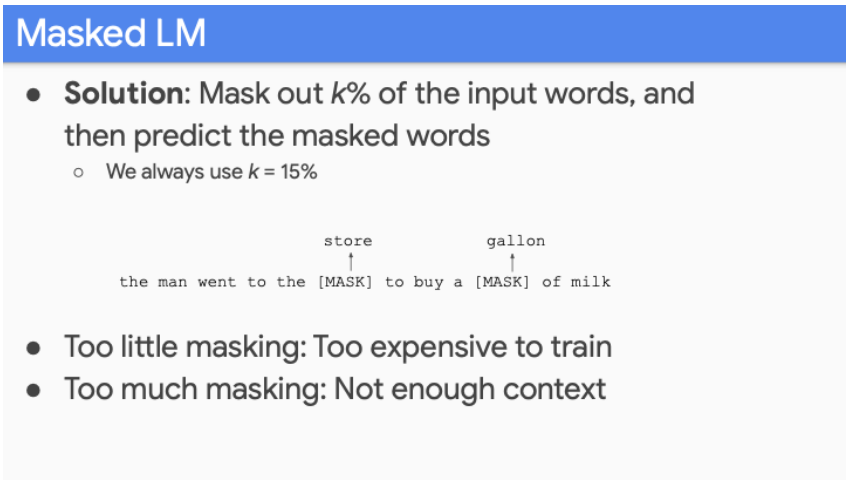
: LM이 left context or right context만을 (또는 두개를 concat) 이용

- 하지만 실제 language understanding은 bidirectional함
- 이유 1) LM은 historically하게 다른 시스템에서의 feature로 사용됨, most direct application of LM은 predictive text와 같은 것이 있음. 다른 application은 MT, speech recognition system과 같은 것으로, translation feature or acoustic feature에 대해 LM에 넣어 sentence의 확률을 계산해볼 수 있음
- 그래서 well-formed distribution이 필요했지만 pre-train model은 사실 이러한 것을 고려하지 않는다.
- 이유 2) bidirectional encoder에서 word들이 자기 자신들을 볼 수 있다는 것

Unidirectional vs. Bidirectional Models

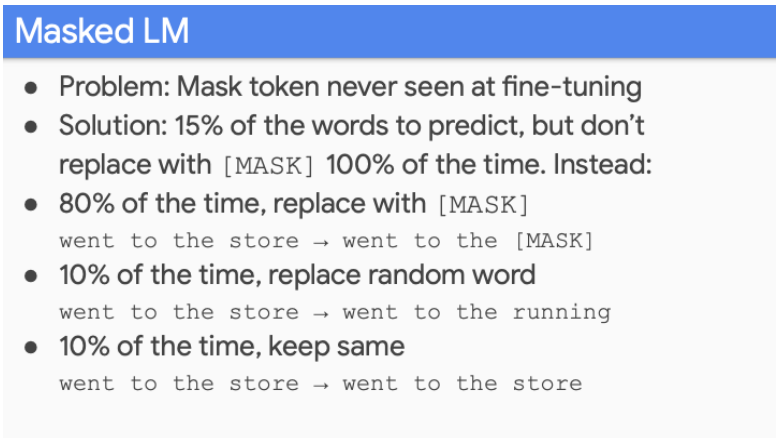


- Language Representation을 한다고 했을때, Unidirectional context에서는 항상 offset by 1이어야 하고, sentenece에 있는 모든 단어를 예측하게 됨. => 좋은 sample efficiency를 보여주는데, 왜냐하면 512 dimension을 갖고 있는 500개의 단어에 대해, 하나의 단어만 예측한다면 500번 반복해야 함
- 하지만 bidirectional-LSTM, transformer를 사용하면, 첫번째 layer이 후에 모든 단어가 자기 자신을 볼 수 있게 됨(path back이 open되어있다.) 그래서 사실상 실제 prediction이 없다.



=> 그래서 간단한 solution으로, normal language model을 이용해서 학습하는 것이 아니라 k percent of words를 mask하여 사용하는 것

- 이렇게 되면 bidirectional model을 사용가능
- input에 대한 정보가없기 때문에 cheat할 수가 없음 이 방법의 단점은 각 단어마다 많은 개수의 prediction은 할 수가 없게 됨.
- k=15%라면 100% word에 대한 prediction이 아닌 15% of words만을 predictive하게 됨
- 하지만 장점은 both direction으로 학습하기 때문에 더 rich한 model을 가짐
- 여기서 k는 emprically하게 결정해야하는 hyperparameter중 하나, 15%가 optimal value였음
- k에 대해 여러 값을 실험하는 이유: 예를들어 50% masking을 사용한다고 했을 때, 더 많은 prediction은 할 수 있겠지만, context의 대부분을 mask out하게 되고, 어떠한 contextual model을 학습할 수 없게 됨
- 하지만 만약에 문장에 대해 하나의 단어씩 mask한다면 optimal이겠지만, data processing, model training의 cost가 expensive하게 됨(compute-bounded) 그래서 %k에 대해 이와 같은 trade-off가 존재한다.



BERT에서 매우 중요한 것 중 하나는, fine-tuning 할때 mask token은 unseen token이라는 것!
=> 이것에 대한 해결 방법: 15% of words에 대해 prediction할 때, [mask] token을 100% replace하지 말라는 것

- 80%는 [mask]로 replace하고, 10%는 random, 10%는 같은 것으로 유지
- went to the 다음에 올 단어에 대해 [mask]로 100% 학습을 하게 되면, [mask]에 대해 예측된 것이 맞는지 틀린지 몰라서 위와 같은 방법을 사용

Next Sentence Prediction

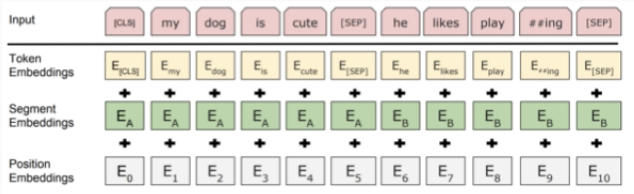
- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

- 다른 종류의 BERT에 관한 detail은, 우리가 하고 있는 많은 task가 words를 학습하는 것이 아니라 sentence 사이의 relationship을 예측하는 것을 학습하길 원함
- 예를들어 주어진 질문에 대해, document의 어떠한 문장이 질문의 답인지 아닌지 물어보는 Q&A task
- word-level이 아닌 sentence-level로 학습하려 하는 것
- 그래서 sentence-level로 학습하기 위해서, next-sentence task 함
- 50%는 같은 document로 부터 두개의 문장을 가져오고, 50%는 random document에서 2개의 문장을 가져와서 was this real next sentence라는 것을 학습

Input Representation



- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

- Input Representation
  : normal transformer와 유사, 여기에 추가적인 embedding, segment embeddings가 있음
- normal transformer와 같이 input에 대해 WordPiece segmentation함(roughly morphological한 값을 갖게) 30,000 word vocab을 사용
- transformer와 동일하게 positional embedding을 사용(LSTM과 다르게 locational awareness가 없기 때문에, 순서를 몰라)
- 이후 Segment Embedding을 하는데 이것이 sentence A인지 B인지 구별

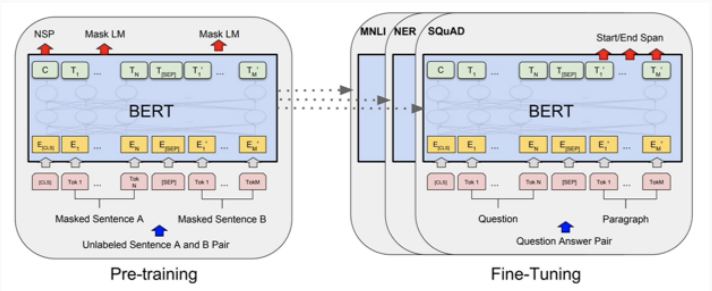
(이것은 older style과 반대되는데, 과거의 것들은 모든 파트에 대해 다른 encoder를 갖고 있는데, BERT는 하나의 sequence를 input으로 받음(하나의 Encoder로!))

Model Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

3B words

Fine-Tuning Procedure

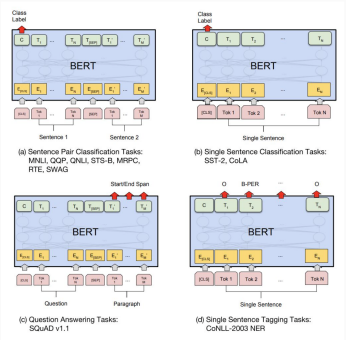


<Fine tuning Procedure>

위 두가지 task에 대해 pretrain

- 다른 embedding 값을 갖고 있는 multiple sentence로 이루어져 있는 input sequence에 대해, transformer model로 feed
- 위에서 언급하지 않은 special embedding [CLS], [SEP]이 있는데, next-sentence prediction task를 학습하기 위해 사용한 것으로, 이 embedding이 intrinsic하게 유용하지는 않음.
- 12 or 24 layer model의 weight가 유용하고, 전체 모델을 fine-tuning 하기 위해서는 silent part만을 골라서 down-stream task에 사용할 수 있음

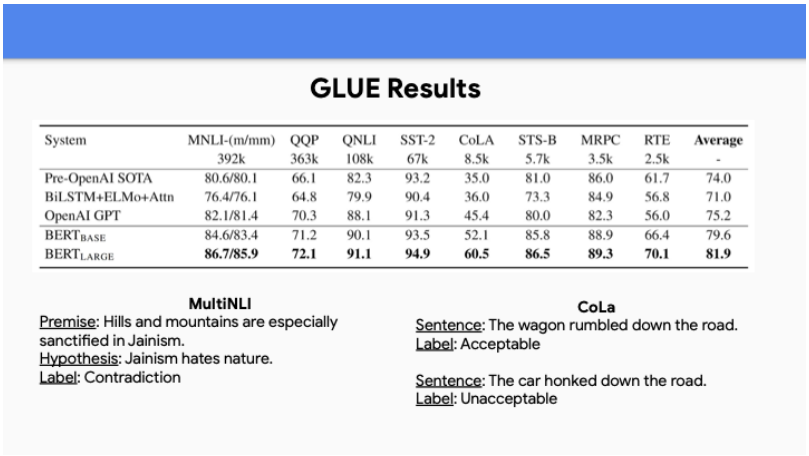
Fine-Tuning Procedure



- 위의 슬라이드가 finetuning에서의 case들

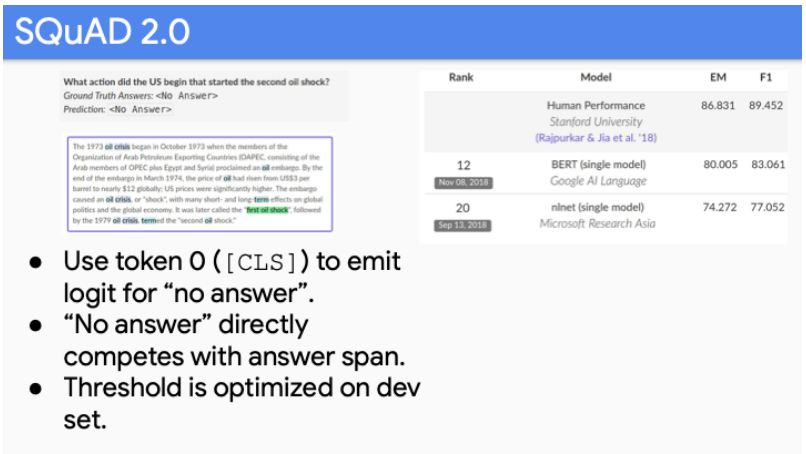
- 만약에 single classifcation task를 한다면, ex. sentiment analysis와 같은, BERT model로 문장을 encode하고나서 우리가 사용하는 것은 final output matrix
- 예를들어 3개의 종류 (Positive, Neutral, Negative)에 대해 3000개에 새로운 Parameter와 기존의 300 million parameter에 대해 jointly train하게 됨
- Span task에 대해서는 start, end 지점에 대한 token을 학습하기 위해 few thousand new parameter를 학습하게 됨
- BERT는 이전의 work에 대해 매우 incremental improvement한 work
- ELMO는 사실 기존의 contextual embedding과의 fundamental한 difference는 없음
- 역사적으로 Deep learning에서는 여러 종류의 layer 들을 합쳐서 새로운 task에 좋은 성능을 달성하는 것에 지나지 않아
- GPT1 또한 left-to-right LM이 었기 때문에 여러 downstream task에서 잘 작동하지 않음, POS와 같은 task에 대해 생각해보면 first word는 context가 없기 때문에 잘 예측할 수 없었음

=> BERT가 매우큰 impact을 갖는 이유를 물리학자 비유해 보면, 많은 research 분야(problem)를 없애버리는 엄청난 theory가 end-goal일 것이고, 이와 유사하게 down-stream task를 해결!



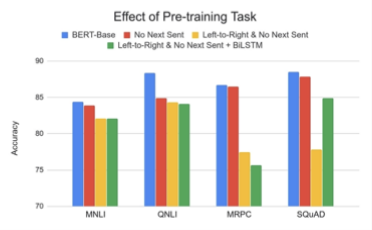
GLUE Result는 대부분 sentence pair 또는 sentence classification task에 관한 것

- BERT Large에 대해서 놀라운 점은 tiny example에 대해서도 가장 성능이 좋아
- 매우 큰 모델을 pretrain하지 않고 작은 데이터 셋에 대해 학습하면 overfit 됨, 그래서 얼마나 큰 pretrain model이 좋은 성능을 보일지 제한이 없는 것처럼 보여



위 슬라이드는 Question Answering dataset인데, 질문에 대해 답이 없는 경우에 "no anwser'라고 대답할 수 있어야 함이 task에 대해 BERT가 그 당시에 state-of-art를 달성했다. 현재는 Human performance를 능가!

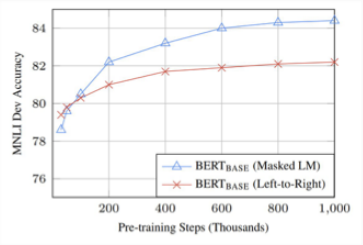
Effect of Pre-training Task



- Masked LM (compared to left-to-right LM) is very important on some tasks, Next Sentence Prediction is important on other tasks.
- Left-to-right model does very poorly on word-level task (SQuAD), although this is mitigated by BiLSTM

- 위의 plot들은 모두 BERT based sized model들이다. 빨간색은 next sentence를 제거한 경우이고, 이러한 task가 중요한지 아닌지 알수있음
- 노란색은 bidirectional을 고려하지 않은 모델이다. MRPC, SQuAD와 같은 task는 span labeling task이고, GPT-1의 경우 context없이 시작하고,끝나는 지점을 알 수 없음

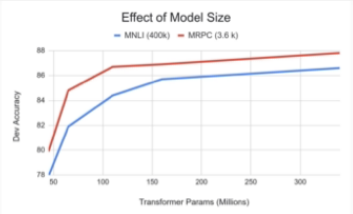
Effect of Directionality and Training Time



- Masked LM takes slightly longer to converge because we only predict 15% instead of 100%
- But absolute results are much better almost immediately

- Masked LM을 사용하면 15%의 단어만 예측하는 반면, Left-to-Right LM 방식을 사용하면 100%의 단어를 예측하여 학습
- 두가지 방법에 대해 converge하는 경향성을 나타내는 그래프, 수렴하는 데 BERT가 더 오래 걸리지만 Bidirectionality가 중요하기 때문에 전체 수렴한 값은 훨씬 큼

Effect of Model Size



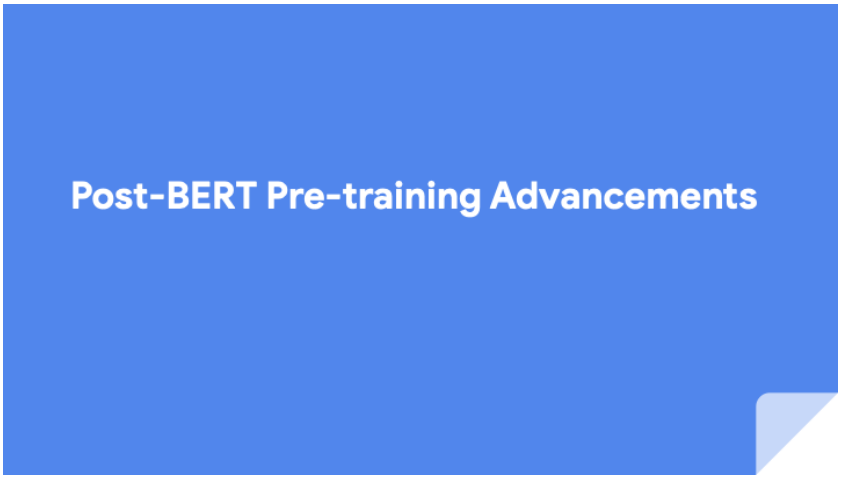
- Big models help *a lot*
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have *not* asymptoted

두가지 그래프를 비교하지말아라 (대응되지 않는 값들이다. 다른 데이터셋을 사용함)
=> 주의해서 봐야할 것은 parameter의 개수, 기존의 rule of thumb은 매우 적은 label에 대해 parameter가 많아지면 overfit한다는 것인데, 여기서 더 이상 그 말이 맞지 않음

Open Source Release

- One reason for BERT's success was the open source release
  - Minimal release (not part of a larger codebase)
  - No dependencies but TensorFlow (or PyTorch)
  - Abstracted so people could including a single file to use model
  - End-to-end push-button examples to train SOTA models
  - Thorough README
  - Idiomatic code
  - Well-documented code
  - Good support (for the first few months)

그리고 BERT가 다른 model에 비해 성공적인 이유가 또한 open source release에 있었다고 한다. 위의 list가 open source release할때 중요한 것들



이 BERT를 다양한 방법으로 향상시킨 5가지 Post-BERT 모델에 대해 설명

RoBERTA

- *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (Liu et al, University of Washington and Facebook, 2019)
- Trained BERT for more epochs and/or on more data
  - Showed that more epochs alone helps, even on same data
  - More data also helps
- Improved masking and pre-training data slightly

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-

1) RoBERTA에 관한 것이다. 여기서 보여준 것은 BERT가 매우 under-trained 되었다는 것, improved mask, 더 많은 Epoch 등을 이용해 학습

XLNet

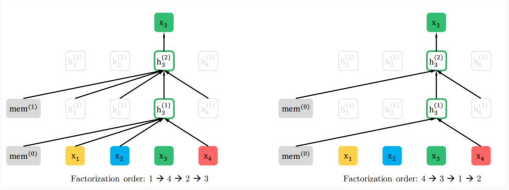
- *XLNet: Generalized Autoregressive Pretraining for Language Understanding* (Yang et al, CMU and Google, 2019)
- Innovation #1: Relative position embeddings
  - Sentence: `John ate a hot dog`
  - Absolute attention: "How much should `dog` attend to `hot`(in any position), and how much should `dog` in position 4 attend to the word in position 3? (Or 508 attend to 507, ...)"
  - Relative attention: "How much should `dog` attend to `hot` (in any position) and how much should `dog` attend to the previous word?"

2) XLNET에서는 Transformer-XL을 사용했는데, 이것은 relative position embedding을 사용한다는 점에서 다름

- Absolute position embedding에서의 문제점은 모든 단어가 위치에 대해 특정한 값을 갖고 이를 실제 현실에서 고려하면 quadratic number of relationship을 갖는다는 것이고, 매우 큰 크기의 문장에서 N^2의 relation ship을 갖게 됨
- Relative position embedding에서는 'how much dog attend to hot'과 같은 것을 고려
- 이 방법이 긴 문장에서 이전 방법보다 더 잘 작동!

XLNet

- Innovation #2: Permutation Language Modeling
  - In a left-to-right language model, every word is predicted based on all of the words to its left
  - Instead: Randomly permute the order for *every training sentence*
  - Equivalent to masking, but many more predictions per sentence
  - Can be done efficiently with Transformers



- 두번째는 pretraining에 대한 것으로, Permutation Language modeling대한 것
- . Left to right LM에서는 모든 단어를 Left to right으로 예측, 하지만 XLNET에서는 모든 단어에 대해 randomly permute order로 예측
- 예를들어 1→4→2→3, 4→3→1→2 순서로 학습할 수 있고 이 방법은 꽤 valid한 way이고, well-formed probability distribution을 얻음
- 그 이유: 원래와 같이 한번에 하나의 단어만 predict하기 때문이다. 이것을 Transformer에서 매우 효율적으로 구현할 수 있는데, 단순히 attention probability를 mask out 하면됨
- => 그래서 각 sentence에 대해 single permutation을 하게 갖게되고, 이것으로 bidirectional model을 더 효율적으로 학습 가능 (왜냐하면 각 단어가 right, left 고르게 condition하게 되기 때문) LSTM에서는 fixed order때문에 이 방법을 적용할 수 없다.

XLNet

- Also used more data and bigger models, but showed that innovations improved on BERT even with same data and model size
- XLNet results:

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
Single-task single models on dev								
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0
RoBERTa [21]	90.2/90.2	94.7	92.2	**86.6**	96.4	**90.9**	68.0	92.4
XLNet	**90.8/90.8**	**94.9**	**92.3**	85.9	**97.0**	90.8	**69.0**	**92.5**

XLNET이 RoBERTa와 거의 비슷한 성능을 보이지만 technique면에서는 좀 더 innovation했다.

ALBERT

- *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations* (Lan et al, Google and TTI Chicago, 2019)
- Innovation #1: Factorized embedding parameterization
  - Use small embedding size (e.g., 128) and then project it to Transformer hidden size (e.g., 1024) with parameter matrix

ALBERT : Lite BERT for self-supervised learning이라는 모델

- 몇가지 멋진 innovation이 있는데, 아이디어는 massive parameter sharing에 관한 것
- Parameter를 share하게 되면 더 좋은 LM은 얻지 못하게 되지만, 더 좋은 sample efficiency를 갖게됨 = 즉, 덜 overfit
- billion parameter를 갖고 있을때, 매우 적은 데이터에 대해 매우 빠르게 overfit 할것
- 하지만 더 적은 parameter를 갖고 있다면 덜 overfit할 것
- 그래서 2가지 major innovation중 1) word embedding을 사용하지 않는것, word embedding은 매우 큰데, 그 사이즈는 vocab_size * num_word * hidden_size이고, 이것은 hidden layer보다 더 커지게 된다. 그래서 factorized embedding data를 사용하는 것
- 위와 같이 matrix multiplication을 통해 100kx1024의 matrix를 갖게된다. 하지만 실제로는 더 적은 parameter를 갖는다. parameter tying은 아니지만, 적절한 방법으로 parameter reduction을 할 수 있게 된다.

ALBERT

- Innovation #2: Cross-layer parameter sharing
  - Share all parameters between Transformer layers
- Results:

Models	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS
Single-task single models on dev								
BERT-large	86.6	92.3	91.3	70.4	93.2	88.0	60.6	90.0
XLNet-large	89.8	93.9	91.8	83.8	95.6	89.2	63.6	91.8
RoBERTa-large	90.2	94.7	**92.2**	86.6	96.4	**90.9**	68.0	92.4
ALBERT (1M)	90.4	95.2	92.0	88.1	96.8	90.2	68.7	92.7
ALBERT (1.5M)	**90.8**	**95.3**	**92.2**	**89.2**	**96.9**	**90.9**	**71.4**	**93.0**

- ALBERT is light in terms of *parameters*, not *speed*

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

- 2) 두번째 innovation: cross-layer parameter sharing
- Universal transformer에서 했던 것인데, 사용하고 있는 bunch of transformer layer에 대해 all 12 layer가 같은 parameter를 공유하는 것
- => 그래서 BERT보다 훨씬 적은 parameter를 갖게되고, 덜 overfit해진다. XLNet이나 RoBERTa에 비해 state-of-art를 달성
- ALBERT에 대해 명심해야할 것 중 하나는 parameter에 대해 light할 뿐, 빠르진 않아 여전히 pretrainnig해야하는 양을 줄이는 방법을 찾진 못했음

T5

- *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* (Raffel et al, Google, 2019)
- Ablated many aspects of pre-training:
  - Model size
  - Amount of training data
  - Domain/cleanness of training data
  - Pre-training objective details (e.g., span length of masked text)
  - Ensembling
  - Finetuning recipe (e.g., only allowing certain layers to finetune)
  - Multi-task training

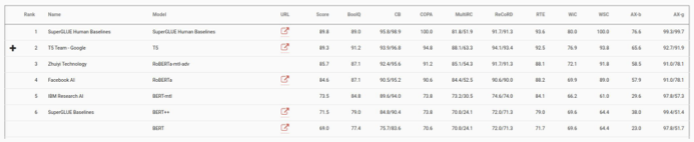
Google brain에서 작성한 논문으로, Pretraining에 여러가지 방법을 적용
=> 엄청난 super clever new pre-training technique을 사용한 것이 아니라, 모든 aspect에 대해 ablate

- How much does model size matter?
- How much does training data matter?
- How much does cleanness of data matter?
- How much does cleanness of data matter?
- How much does the exact way that you do pre-trainig object matter?
- How many do you mask

위의 것들 이상에 대해 명확하게 하기 위해 노력

T5

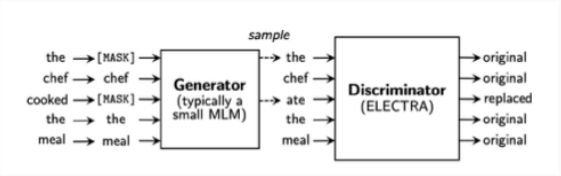
- Conclusions:
  - Scaling up model size and amount of training data helps a lot
  - Best model is 11B parameters (BERT-Large is 330M), trained on 120B words of cleaned common crawl text
  - Exact masking/corruptions strategy doesn't matter that much
  - Mostly negative results for better finetuning and multi-task strategies
- T5 results:



그래서 모든 task에서 state-of-art를 달성
=> 그래서 결론은 bigger model, training more data, more clean data가 중요하다는 것(다른 aspect이 아니라)

ELECTRA

- *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators* (Clark et al, 2020)
- Train model to discriminate locally plausible text from real text



- Newest paper 중 하나인 ELECTRA
- Instead of training to generate output, just train discriminator. Local Language model으로 generate하고 discriminator로 original인지 replace인지 학습
- pre-training을 위해 더 좋은 sample efficiency를 가짐, weak model에 대해 위의 방법을 사용하여 strong model로 만들 수 있어

ELECTRA

- Difficult to match SOTA results with less compute

Model	Train FLOPs	Params	SQuAD 1.1 EM	SQuAD 1.1 F1	SQuAD 2.0 EM	SQuAD 2.0 F1
BERT-Base	6.4e19 (0.09x)	110M	80.8	88.5	–	–
BERT	1.9e20 (0.27x)	335M	84.1	90.9	79.0	81.8
SpanBERT	7.1e20 (1x)	335M	88.8	94.6	85.7	88.7
XLNet-Base	6.6e19 (0.09x)	117M	81.3	–	78.5	–
XLNet	3.9e21 (5.4x)	360M	**89.7**	**95.1**	87.9	**90.6**
RoBERTa-100K	6.4e20 (0.90x)	356M	–	94.0	–	87.7
RoBERTa-500K	3.2e21 (4.5x)	356M	88.9	94.6	86.5	89.4
ALBERT	3.1e22 (44x)	235M	89.3	94.8	87.4	90.2
BERT (ours)	7.1e20 (1x)	335M	88.0	93.7	84.7	87.5
ELECTRA-Base	6.4e19 (0.09x)	110M	84.5	90.8	80.5	83.3
ELECTRA-400K	7.1e20 (1x)	335M	88.7	94.2	86.9	89.6
ELECTRA-1.75M	3.1e21 (4.4x)	335M	**89.7**	94.9	**88.1**	**90.6**

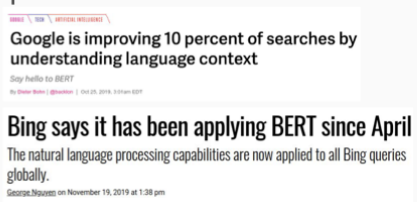
FLOPs를 보면 BERT-Large 유사하게 많은 양을 연산해야 했음
즉 state-of-art의 성능을 내려면 sota 만큼의 연산을 해야한다는 뜻, 그래서 pretrained model을 cheaper하게 아직은 사용할 수 없었다.

Distillation

머자막으로 이러한 모델을 어떻게 serve할까? incredibly expensive to train, nobody has been to figure how to make that faster.

Applying Models to Production Services

- BERT and other pre-trained language models are extremely large and expensive
- How are companies applying them to low-latency production services?



Google Search , Bing Search에 BERT와 같은 model을 사용, 이것을 어떻게 하고 있을까?

Distillation

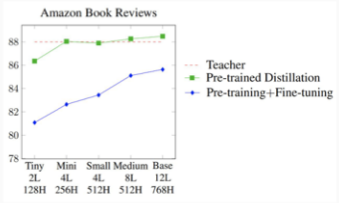
- Answer: Distillation (a.k.a., model compression)
- Idea has been around for a long time:
  - *Model Compression* (Bucila et al, 2006)
  - *Distilling the Knowledge in a Neural Network* (Hinton et al, 2015)
- Simple technique:
  - Train "Teacher": Use SOTA pre-training + fine-tuning technique to train model with maximum accuracy
  - Label a large amount of unlabeled input examples with Teacher
  - Train "Student": Much smaller model (e.g., 50x smaller) which is trained to mimic Teacher output
  - Student objective is typically Mean Square Error or Cross Entropy

그것에 대한 해답은 Distillation 또는 model compression에 !
pretrained LM에 사용한 Distillation은 매우 간단한 techniqued임(하지만 그것이 의미하는 바를 오해하기 쉽다.)

- pretrained model 자체를 압축하는 것이 아니라, task에 대해 fine-tune한 후에 압축하는 것

Distillation

- Example distillation results
  - 50k labeled examples, 8M unlabeled examples



Well-Read Students Learn Better: On the Importance of Pre-training Compact Models (Turc et al, 2020)

- Distillation works *much* better than pre-training + fine-tuning with smaller model

하지만 여전히 처음엔 big model을 학습시켜야 하므로 training cost를 줄여주지는 않음
Inference time에서는 매우 작은 cost로 서비스를 할 수 있게 됨

Distillation

- Why does distillation work so well? A hypothesis:
  - Language modeling is the "ultimate" NLP task in many ways
    - I.e., a perfect language model is also a perfect question answering/entailment/sentiment analysis model
  - Training a massive language model learns millions of latent features which are useful for these other NLP tasks
  - Finetuning mostly just picks up and tweaks these existing latent features
  - This requires an oversized model, because only a subset of the features are useful for any given task
  - Distillation allows the model to only focus on those features
  - Supporting evidence: Simple self-distillation (distilling a smaller BERT model) doesn't work

- 그렇다면 질문은 왜 이렇한 distillation이 잘 작동하는 걸까? LM은 NLP task의 궁국적인 task
- 매우큰 LM을 학습할떄, 많은 개수의 latent feature를 학습하게 되고, latent feature가 finetuning에서의 weight에 반영됨
- Distillation이 이러한 feature에만 focus하게 도와줌

Conclusions

Conclusions

- Pre-trained bidirectional language models work incredibly well
- However, the models are extremely expensive
- Improvements (unfortunately) seem to mostly come from even more expensive models and more data
- The inference/serving problem is mostly “solved” through distillation

But training time에서 더 빠르게 하는 문제는 해결되지 않음