Primers • Bidirectional Encoder Representations from Transformers (BERT)

Background: Pre-Training

- One of the biggest challenges in natural language processing (NLP) is the shortage of training data. Because NLP is a diversified field with many distinct tasks, most task-specific datasets contain only a few

variety of techniques for training general purpose language representation models using the enormous amount of unannotated text on the web (known as pre-training).
- The pre-trained model can then be fine-tuned on small-data NLP tasks like question answering and sentiment analysis, resulting in substantial accuracy improvements compared to training on these datasets from scratch.

Enter BERT

- In 2018, Google open sourced a new technique for NLP pre-training called Bidirectional Encoder Representations from Transformers, or BERT. As the name suggests, it generates representations using an encoder from Vaswani et al.'s Transformer architecture. However, there are notable differences between BERT and the original Transformer, especially in how they train those models.
- With BERT, anyone in the world can train their own state-of-the-art question answering system (or a variety of other models) in about 30 minutes on a single Cloud TPU, or in a few hours using a single GPU. The release includes source code built on top of TensorFlow and a number of pre-trained language representation models.
- In the paper, Devlin et al. (2018) demonstrate state-of-the-art results on 11 NLP tasks, including the very competitive Stanford Question Answering Dataset (SQuAD v1.1).

Word Embeddings

- Word embeddings (or word vectors) are a way to represent words or sentences as vectors of numbers that can be fed into downstream models for various tasks such as search, recommendation, translation and so on. The idea can be generalized to various entities beyond words – for e.g., topics, products, and even people (commonly done in applications such as recommender systems, face/speaker recognition, etc.)

embeddings trained on the Google News corpus are available). Two important learnings from Word2Vec were:
    - Embeddings of semantically similar words are close in cosine similarity.
    - Word embeddings support intuitive arithmetic properties. (An important consequence of this statement is that phrase embeddings can be obtained as the sum of word embeddings.)
- However, since Word2Vec there have been numerous techniques that attempted to create better and better deep learning models for producing embeddings (as we'll see in the later sections).

Contextual vs. Non-contextual Word Embeddings

- It is often stated that Word2Vec and GloVe yield non-contextual embeddings while ELMo and BERT embeddings are contextual. At a top level, all word embeddings are fundamentally non-contextual but can be made contextual by incorporating hidden layers:
    1. The word2vec model is trained to learn embeddings that predict either the probability of a surrounding word occurring given a center word (SkipGram) or vice versa (CBoW). The surrounding words are also called context words because they appear in the context of the center word.

2.

3. The GloVe model is trained such that a pair of embeddings has weights that reflect their co-occurrence probability. The latter is defined as the percentage of times that a given word $\mathbf{y}$ occurs within some context window of word $\mathbf{x}$.
4. If embeddings are trained from scratch in an encoder / decoder framework involving RNNs (or their variants), then, at the input layer, the embedding that you look up for a given word reflects nothing about the context of the word in that particular sentence. The same goes for Transformer-based architectures.

- Word2Vec and GloVe embeddings can be plugged into any type of neural language model, and

particular sentence. Similarly, while hidden layers of an LSTM encoder or Transformer do extract information about surrounding words to represent a given word, the embeddings at the input layer do not.

- **Key takeaways**
  - Word embeddings you retrieve from a lookup table are always non-contextual since you cannot have pre-generated contextualized embeddings. (It is slightly different in ELMo which uses a character-based network to get a word embedding, but it does consider context).
  - However, when people say contextual word embeddings, they don't mean the vectors from the look-up table, they mean the hidden states of the pre-trained model.
  - Here are the key differences between them Word2Vec and BERT embeddings and when to use each:

	BERT	Word2Vec
Central idea	Offers different embeddings based on context, i.e., "contextual embeddings" (say, the game "Go" vs. the verb "go"). The same word occurs in different contexts can thus yield different embeddings.	Same embedding for a given word even if it occurs in different contexts.
Context/Dependency coverage	Captures longer range context/dependencies owing to the Transformer encoder architecture under-the-hood which is designed to capture more context.	While Word2Vec Skipgram tries to predict contextual (surrounding) words based on the center word (and CBOW does the reverse, i.e.,), both are limited in their context to just a static window size (which is a hyperparameter) and thus cannot capture longer range context relative to BERT.
		Does not take into account word

	BERT	Word2Vec
Generating embeddings	Use a pre-trained model and generate embeddings for desired words (rather than using pre-trained embeddings as in Word2Vec) since they are tailor-fit based on the context.	Off-the-shelf pre-trained word embeddings available since embeddings are context-agnostic.

ELMo: Contextualized Embeddings

- ELMo came up with the concept of contextualized embeddings by grouping together the hidden states of the LSTM-based model (and the initial non-contextualized embedding) in a certain way (concatenation followed by weighted summation).

BERT: an Overview

- At the input, BERT (and many other transformer models) consume 512 tokens max —- truncating anything beyond this length. Since it can generate an output per input token, it can output 512 tokens.
- BERT actually uses WordPieces as tokens rather than the input words – so some words are broken down into smaller chunks.
- BERT is trained using two objectives: (i) Masked Language Modeling (MLM), and (ii) Next Sentence Prediction (NSP).

Masked Language Modeling (MLM)

- While the OpenAI transformer (which was a decoder) gave us a fine-tunable (through prompting) pre-

model looks both forward and backwards (in the technical jargon – "is conditioned on both left and right context")?

- However, here's the issue with bidirectional conditioning when pre-training a language model. The community usually trains a language model by training it on a related task which helps develop a contextual understanding of words in a model. More often than not, such tasks involve predicting the next word or words in close vicinity of each other. Such training methods can't be extended and used for bidirectional models because it would allow each word to indirectly "see itself" — when you would approach the same sentence again but from opposite direction, you kind of already know what to expect. A case of data leakage. In other words, bidirectional conditioning would allow each word to indirectly see itself in a multi-layered context. The training objective of Masked Language Modelling, which seeks to predict the masked tokens, solves this problem.
- While the masked language modelling objective allows us to obtain a bidirectional pre-trained model, note that a downside is that we are creating a mismatch between pre-training and fine-tuning, since the `[MASK]` token does not appear during fine-tuning. To mitigate this, BERT does not always replace "masked" words with the actual `[MASK]` token. The training data generator chooses 15% of the token positions at random for prediction. If the $\mathbf{i}^{\text{th}}$ token is chosen, BERT replaces the $\mathbf{i}^{\text{th}}$ token with (1) the `[MASK]` token 80% of the time (2) a random token 10% of the time, and (3) the unchanged $\mathbf{i}^{\text{th}}$ token 10% of the time.

Next Sentence Prediction (NSP)

- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task: Given two sentences ($\mathbf{A}$ and $\mathbf{B}$), is $\mathbf{B}$ likely to be the sentence that follows $\mathbf{A}$, or not?
- More on the two pre-training objectives in the section on Masked Language Model (MLM) and Next Sentence Prediction (NSP).

- BERT comes in two flavors:
  - BERT Base: 12 layers (transformer blocks), 12 attention heads, 768 hidden size (i.e., the size of $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$ vectors), and 110 million parameters.
  - BERT Large: 24 layers (transformer blocks), 16 attention heads, 1024 hidden size (i.e., the size of $\mathbf{q}$, $\mathbf{k}$ and $\mathbf{v}$ vectors) and 340 million parameters.
- By default BERT (which typically refers to BERT-base), word embeddings have 768 dimensions.

Sentence Embeddings with BERT

- To calculate sentence embeddings using BERT, there are multiple strategies, but a simple approach is to average the second to last hidden layer of each token producing a single 768 length vector. You can also do a weighted sum of the vectors of words in the sentence.

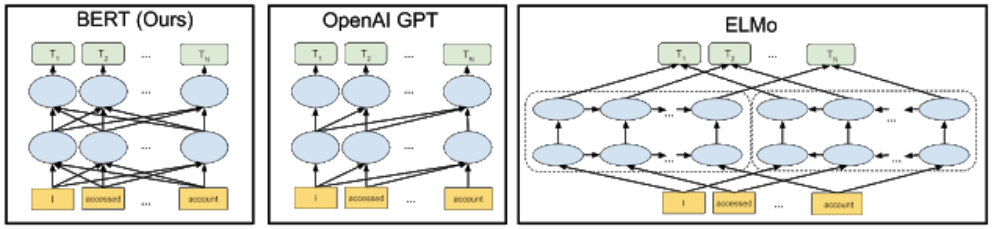
BERT's Encoder Architecture vs. Other Decoder Architectures

- BERT is based on the Transformer encoder. Unlike BERT, decoder models (GPT, TransformerXL, XLNet, etc.) are auto-regressive in nature. As an encoder-based architecture, BERT traded-off auto-regression and gained the ability to incorporate context on both sides of a word and thereby offer better results.
- Note that XLNet brings back autoregression while finding an alternative way to incorporate the context on both sides.
- More on this in the article on Encoding vs. Decoder Models.

What Makes BERT Different?

- BERT builds upon recent work in pre-training contextual representations — including Semi-supervised

- Why does this matter? Pre-trained representations can either be context-free or contextual, and contextual representations can further be unidirectional or bidirectional. Context-free models such as word2vec or GloVe generate a single word embedding representation for each word in the vocabulary.
- For example, the word “bank” would have the same context-free representation in “bank account” and “bank of the river.” Contextual models instead generate a representation of each word that is based on the other words in the sentence. For example, in the sentence “I accessed the bank account,” a unidirectional contextual model would represent “bank” based on “I accessed the” but not “account.” However, BERT represents “bank” using both its previous and next context — “I accessed the … account” — starting from the very bottom of a deep neural network, making it deeply bidirectional.
- A visualization of BERT’s neural network architecture compared to previous state-of-the-art contextual pre-training methods is shown below (source). BERT is deeply **bidirectional**, OpenAI GPT is **unidirectional**, and ELMo is shallowly **bidirectional**. The arrows indicate the information flow from one layer to the next. The green boxes at the top indicate the final contextualized representation of each input word:



Why Unsupervised Pre-Training?

- Vaswani et al. employed supervised learning to train the original Transformer models for language translation tasks, which requires pairs of source and target language sentences. For example, a German-to-English translation model needs a training dataset with many German sentences and corresponding

- We actually can use unsupervised learning to tap into many unlabelled corpora. However, before discussing unsupervised learning, let's look at another problem with supervised representation learning. The original Transformer architecture has an encoder for a source language and a decoder for a target language. The encoder learns task-specific representations, which are helpful for the decoder to perform translation, i.e., from German sentences to English. It sounds reasonable that the model learns representations helpful for the ultimate objective. But there is a catch.
- If we wanted the model to perform other tasks like question answering and language inference, we would need to modify its architecture and re-train it from scratch. It is time-consuming, especially with a large corpus.
- Would human brains learn different representations for each specific task? It does not seem so. When kids learn a language, they do not aim for a single task in mind. They would somehow understand the use of words in many situations and acquire how to adjust and apply them for multiple activities.
- To summarize what we have discussed so far, the question is whether we can train a model with many unlabeled texts to generate representations and adjust the model for different tasks without from-scratch training.
- The answer is a resounding yes, and that's exactly what Devlin et al. did with BERT. They pre-trained a model with unsupervised learning to obtain non-task-specific representations helpful for various language model tasks. Then, they added one additional output layer to fine-tune the pre-trained model for each task, achieving state-of-the-art results on eleven natural language processing tasks like GLUE, MultiNLI, and SQuAD v1.1 and v2.0 (question answering).
- So, the first step in the BERT framework is to pre-train a model on a large amount of unlabeled data, giving many contexts for the model to learn representations in unsupervised training. The resulting pre-trained BERT model is a non-task-specific feature extractor that we can fine-tune quickly to a specific objective.
- The next question is how they pre-trained BERT using text datasets without labeling.

The Strength of Bidirectionality

its previous and next words, since this would allow the word that's being predicted to indirectly "see itself" in a multi-layer model.

- To solve this problem, BERT uses the straightforward technique of masking out some of the words in the input and then condition each word bidirectionally to predict the masked words. In other words, BERT uses the neighboring words (i.e., bidirectional context) to predict the current masked word – also known as the Cloze task. For example (image source):

Input: The man went to the $[MASK]_1$. He bought a $[MASK]_2$ of milk .
Labels: $[MASK]_1$ = store; $[MASK]_2$ = gallon

- While this idea has been around for a very long time, BERT was the first to adopt it to pre-train a deep neural network.
- BERT also learns to model relationships between sentences by pre-training on a very simple task that can be generated from any text corpus: given two sentences A and B, is B the actual next sentence that comes after A in the corpus, or just a random sentence? For example (image source):

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

- Thus, BERT has been trained on two main tasks:
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)

Masked Language Model (MLM)

sequence. It is a left-to-right language model.

- Note: “left-to-right” may imply languages such as English and German. However, other languages use “right-to-left” or “top-to-bottom” sequences. Therefore, “left-to-right” means the natural flow of language sequences (forward), not aiming for specific languages. Also, “right-to-left” means the reverse order of language sequences (backward).
- Unlike left-to-right language models, the masked language models (MLM) estimate the probability of masked words. We randomly hide (mask) some tokens from the input and ask the model to predict the original tokens in the masked positions. It lets us use unlabeled text data since we hide tokens that become labels (as such, we may call it “self-supervised” rather than “unsupervised,” but I’ll keep the same terminology as the paper for consistency’s sake).
- The model predicts each hidden token solely based on its context, where the self-attention mechanism from the Transformer architecture comes into play. The context of a hidden token originates from both directions since the self-attention mechanism considers all tokens, not just the ones preceding the hidden token. Devlin et al. call such representation bi-directional, comparing with uni-directional representation by left-to-right language models.
- Note: IMHO, the term “bi-directional” is a bit misnomer because the self-attention mechanism is not directional at all. However, we should treat the term as the antithesis of “uni-directional”.
- In the paper, they have a footnote (4) that says:

> *We note that in the literature the bidirectional Transformer is often referred to as a “Transformer encoder” while the left-context-only unidirectional version is referred to as a “Transformer decoder” since it can be used for text generation.*

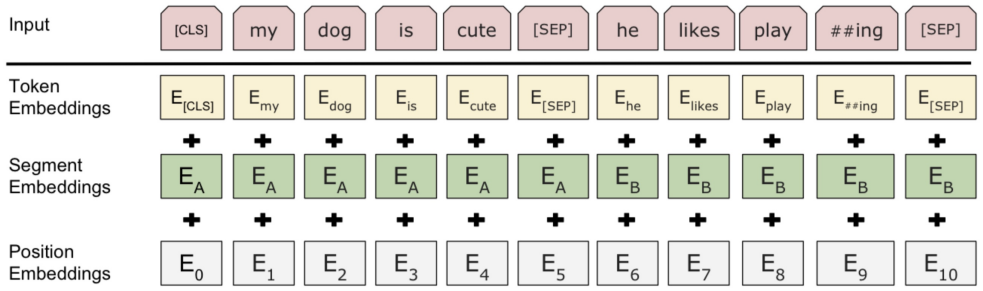
- So, Devlin et al. trained an encoder (including the self-attention layers) to generate bi-directional representations, which can be richer than uni-directional representations from left-to-right language models for some tasks. Also, it is better than simply concatenating independently trained left-to-right and

capture this information.
- We need another unsupervised representation learning task for multi-sentence relationships.

Next Sentence Prediction (NSP)

- A next sentence prediction is a task to predict a binary value (i.e., Yes/No, True/False) to learn the relationship between two sentences. For example, there are two sentences $\mathbf{A}$ and $\mathbf{B}$, and the model predicts if $\mathbf{B}$ is the actual next sentence that follows $\mathbf{A}$. They randomly used the true or false next sentence for $\mathbf{B}$. It is easy to generate such a dataset from any monolingual corpus. Hence, it is unsupervised learning.
- But how can we pre-train a model for both MLM and NSP tasks?
- To understand how a model can accommodate two pre-training objectives, let's look at how they tokenize inputs.
- They used WordPiece for tokenization, which has a vocabulary of 30,000 tokens, based on the most frequent sub-words (combinations of characters and symbols). They also used the following special tokens:
  - `[CLS]` — stands for classification and is the first token of every sequence. The final hidden state is the aggregate sequence representation used for classification tasks.
  - `[SEP]` — a separator token for separating sentences, questions and related passages.
  - `[MASK]` — a token to hide some of the input tokens
- For the MLM task, they randomly choose a token to replace with the `[MASK]` token. They calculate cross-entropy loss between the model's prediction and the masked token to train the model.
- For the NSP task, given sentences $\mathbf{A}$ and $\mathbf{B}$, the model learns to predict if $\mathbf{B}$ follows $\mathbf{A}$. They separated sentences $\mathbf{A}$ and $\mathbf{B}$ using `[SEP]` token and used the final hidden state in place of `[CLS]` for binary classification. The output of the `[CLS]` token tells us how likely the current sentence follows the prior sentence. You can think about the output of `[CLS]` as a probability. The motivation is that the [`CLS`]

- Now you may ask the question – instead of using `[CLS]`'s output, can we just output a number as probability? Yes, we can do that if the task of predicting next sentence is a separate task. However, BERT has been trained on both the MLM and NSP tasks simultaneously. Organizing inputs and outputs in such a format (with both `[MASK]` and `[CLS]`) helps BERT to learn both tasks at the same time and boost its performance.
- When it comes to classification task (e.g. sentiment classification), the output of [CLS] can be helpful because it contains BERT's understanding at the sentence-level.
- Note: there is one more detail when separating two sentences. They added a learned "segment" embedding to every token, indicating whether it belongs to sentence A or B. It's similar to positional encoding, but it is for sentence level. They call it segmentation embeddings. Figure 2 of the paper shows the various embeddings corresponding to the input.



- So, Devlin et al. pre-trained BERT using the two unsupervised tasks and empirically showed that pre-trained bi-directional representations could help execute various language tasks involving single text or text pairs.
- The final step is to conduct supervised fine-tuning to perform specific tasks.

parameters. As a result, we can quickly train a model for each specific task without heavily engineering a task-specific architecture.

- The pre-trained BERT model can generate representations for single text or text pairs, thanks to the special tokens and the two unsupervised language modeling pre-training tasks. As such, we can plug task-specific inputs and outputs into BERT for each downstream task.
- For classification tasks, we feed the final `[CLS]` representation to an output layer. For multi-sentence tasks, the encoder can process a concatenated text pair (using `[SEP]`) into bi-directional cross attention between two sentences. For example, we can use it for question-passage pair in a question-answering task.
- By now, it should be clear why and how they repurposed the Transformer architecture, especially the self-attention mechanism through unsupervised pre-training objectives and downstream task-specific fine-tuning.

Training with Cloud TPUs

- Everything that we've described so far might seem fairly straightforward, so what's the missing piece that made it work so well? Cloud TPUs. Cloud TPUs gave us the freedom to quickly experiment, debug, and tweak our models, which was critical in allowing us to move beyond existing pre-training techniques.
- The Transformer model architecture, developed by researchers at Google in 2017, gave BERT the foundation to make it successful. The Transformer is implemented in Google's open source release, as well as the tensor2tensor library.

Results with BERT

- To evaluate performance, we compared BERT to other state-of-the-art NLP systems. Importantly, BERT achieved all of its results with almost no task-specific changes to the neural network architecture

SQuAD1.1 Leaderboard

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) *Google AI Language* https://arxiv.org/abs/1810.04805	**87.433**	**93.160**
2 Sep 09, 2018	nlnet (ensemble) *Microsoft Research Asia*	85.356	91.202
3 Jul 11, 2018	QANet (ensemble) *Google Brain & CMU*	84.454	90.490

- BERT also improves the state-of-the-art by 7.6% absolute on the very challenging GLUE benchmark, a set of 9 diverse Natural Language Understanding (NLU) tasks. The amount of human-labeled training data in these tasks ranges from 2,500 examples to 400,000 examples, and BERT substantially improves upon the state-of-the-art accuracy on all of them:

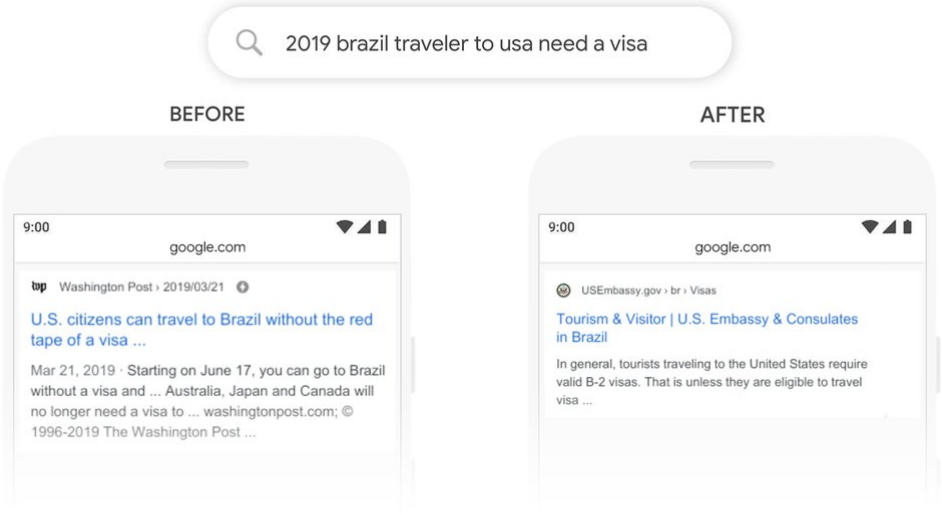
Rank	Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	QNLI	RTE
1	BERT: 24-layers, 1024-hidden, 16-heads	80.4	60.5	94.9	85.4/89.3	87.6/86.5	89.3/72.1	86.7	91.1	70.1
2	Singletask Pretrain Transformer	72.8	45.4	91.3	75.7/82.3	82.0/80.0	88.5/70.3	82.1	88.1	56.0
3	BiLSTM+ELMo+Attn	70.5	36.0	90.4	77.9/84.9	75.1/73.3	84.7/64.8	76.4	79.9	56.8

        - 768-dimensional embedding vectors
        - 12 attention heads
    - The large BERT uses 340M parameters in total:
        - 24 encoder blocks
        - 1024-dimensional embedding vectors
        - 16 attention heads

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Google Search Improvements

- Google search deployed BERT for understanding search queries in 2019.
- Given an input query, say “2019 brazil traveler to usa need a visa”, the following image shows the difference in Google’s search results before and after BERT. Based on the image, we can see that BERT (“after”) does a much better job at understanding the query compared to the keyword-based match (before). A keyword-based match yields articles related to US citizens traveling to Brazil whereas the intent behind the query was someone in Brazil looking to travel to the USA.

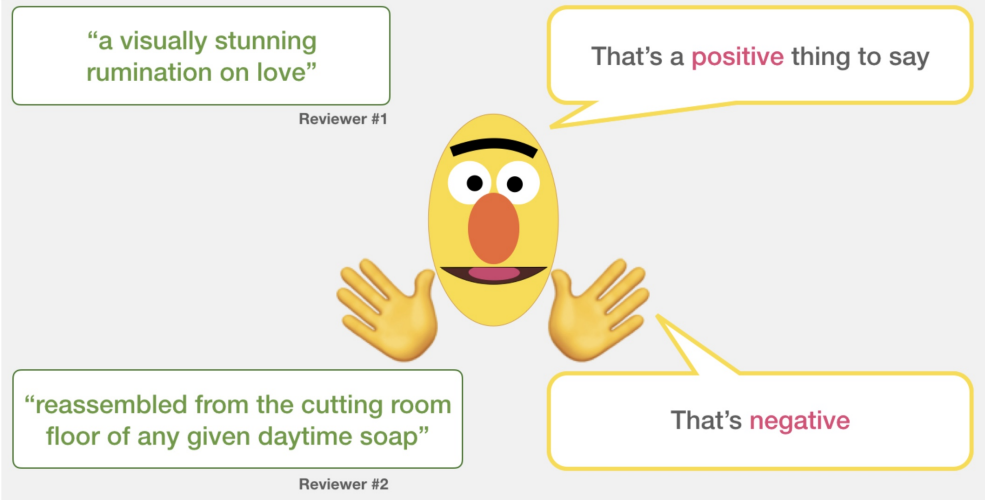


Making BERT Work for You

- The models that Google has released can be fine-tuned on a wide variety of NLP tasks in a few hours or less. The open source release also includes code to run pre-training, although we believe the majority of NLP researchers who use BERT will never need to pre-train their own models from scratch. The BERT models that Google has released so far are English-only, but they are working on releasing models which have been pre-trained on a variety of languages in the near future.
- The open source TensorFlow implementation and pointers to pre-trained BERT models can be found here. Alternatively, you can get started using BERT through Colab with the notebook "BERT FineTuning with Cloud TPUs".

- This Jupyter notebook by Jay Alammar offers a great intro to using a pre-trained BERT model to carry out sentiment classification using the Stanford Sentiment Treebank (SST2) dataset.

A Visual Notebook to Using BERT for the First TIme.ipynb



Further Reading

- Generating word embeddings from BERT
- How are the TokenEmbeddings in BERT created?
- BERT uses WordPiece, RoBERTa uses BPE
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)

References

- What is the vector value of [CLS] [SEP] tokens in BERT
- Difference between non-contextual and contextual word embeddings

Citation

If you found our work useful, please cite it as:

```
@article{Chadha2020DistilledBERT,
  title   = {BERT},
  author  = {Chadha, Aman},
  journal = {Distilled AI},
  year    = {2020},
  note    = {\url{https://aman.ai}}
}
```

www.amanchadha.com