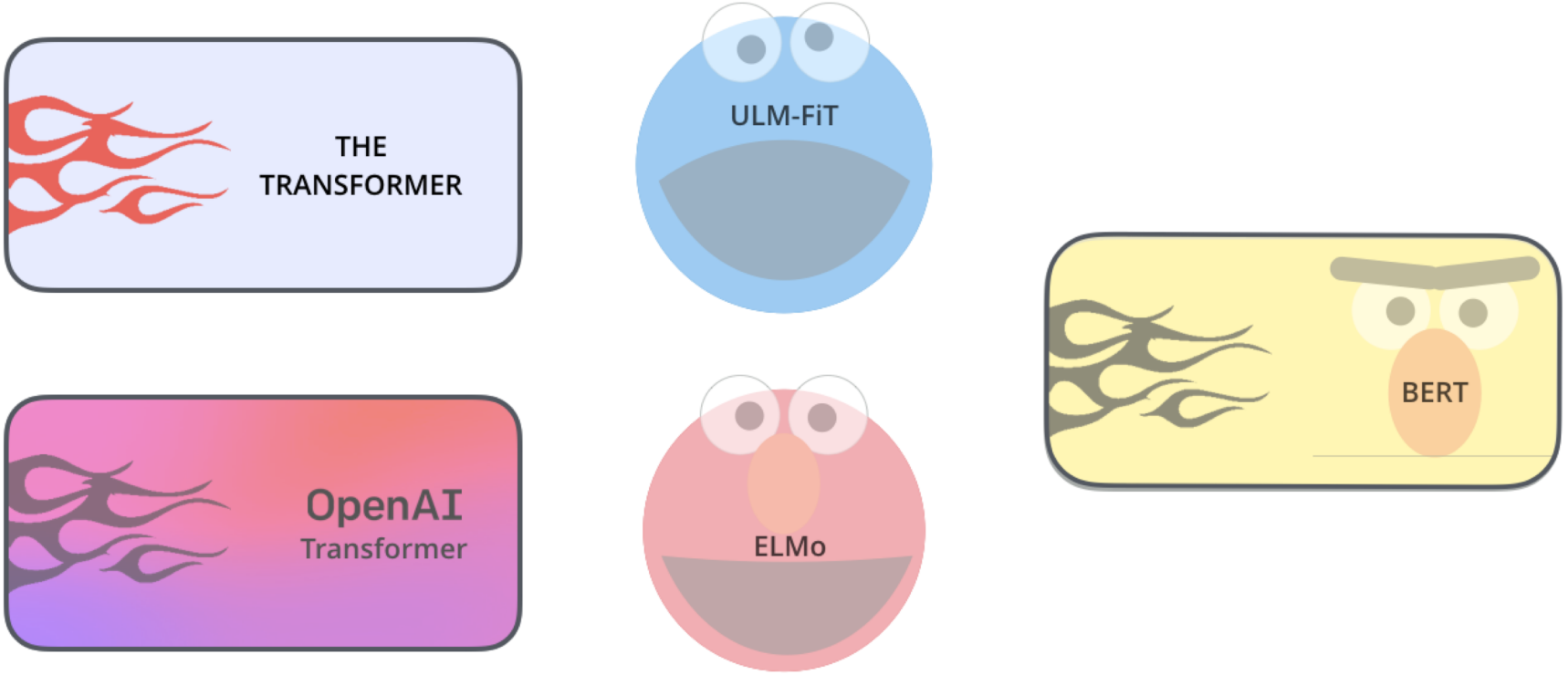
The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)

Adapted from: http://jalammar.github.io/illustrated-bert/

The state of the art in Textual Analysis/NLP ...

- The year 2018 was an inflection point for machine learning models handling text (i.e., Natural Language Processing --- NLP).
- Our conceptual understanding of how best to represent words and sentences in a way that best captures underlying meanings and relationships is rapidly evolving.
- Moreover, the NLP community has been putting forward incredibly powerful components that you can freely download and use in your own models and pipelines
- 2018/2019 has been referred to as "NLP's ImageNet moment" --- referencing how years ago similar developments accelerated the development of machine learning in Computer Vision tasks.

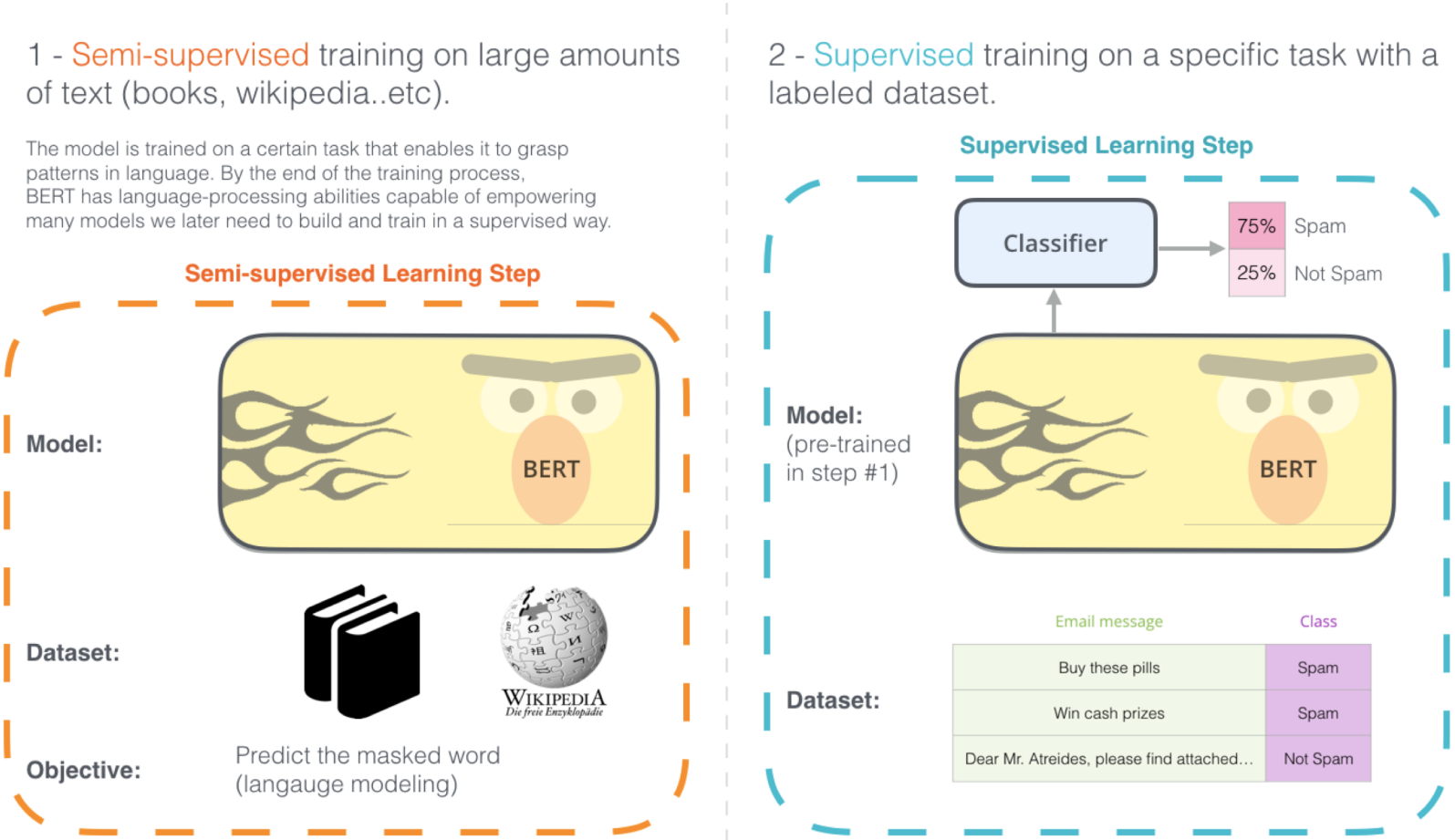
Some State of the Art NLP Models:



The Introduction of BERT

- A major milestones: The release of BERT, an event described as marking the beginning of a new era in NLP.
- BERT is a model that broke several records for how well models can handle language-based tasks.
- The Google BERT team also open-sourced the code of the model --- freely available for download versions of the model already pre-trained on massive datasets.
- Enables anyone building a NLP machine learning model to use “Transfer Learning” methods to save time, energy, knowledge, and resources that would have gone to training a language-processing model from scratch.

Two steps for applying BERT: (1) Download the model that is pre-trained on un-annotated data, and (2) Fine-tune it for your specific application.

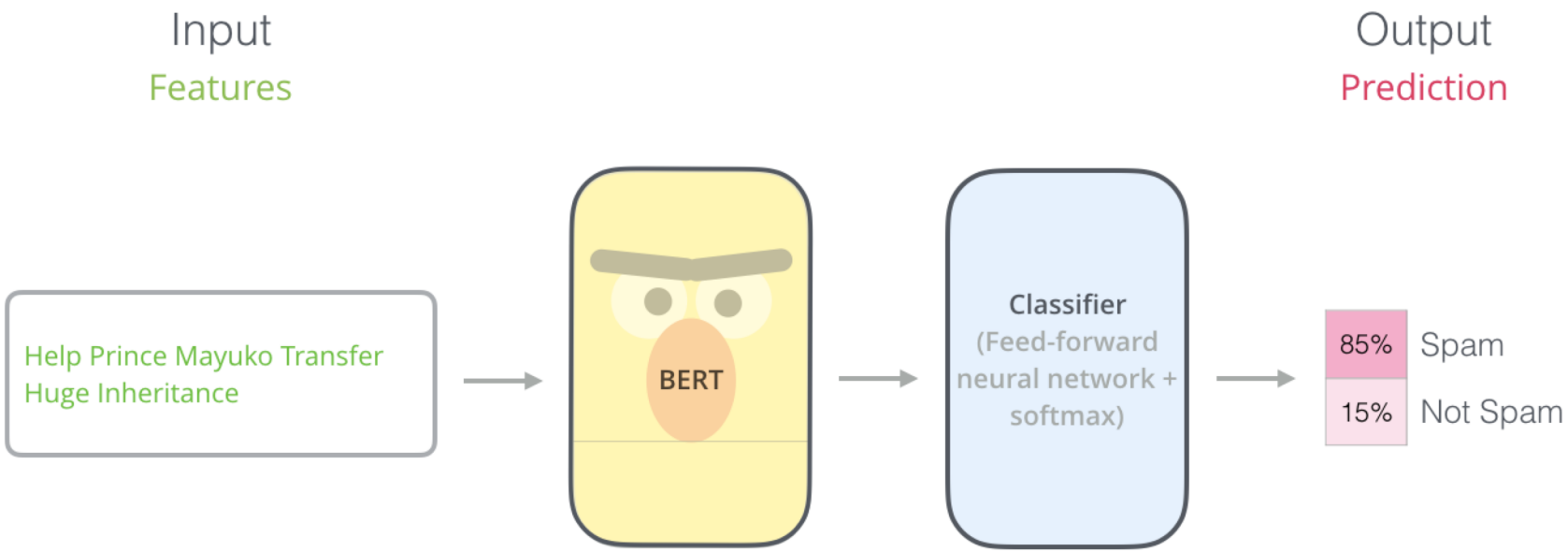


Background behind BERT

- BERT builds on top of a clever NLP developments from other resarchers:
- Semi-supervised Sequence Learning (by Andrew Dai and Quoc Le), ELMo (by Matthew Peters and researchers from AI2 and UW CSE), ULMFiT (by fast.ai founder Jeremy Howard and Sebastian Ruder), the OpenAI transformer (by OpenAI researchers Radford, Narasimhan, Salimans, and Sutskever), and the Transformer (Vaswani et al).
- Let's first look first at ways you can use BERT before looking at the concepts involved in the model itself.

Example: Sentence Classification

The most straight-forward way to use BERT is to use it to classify a single piece of text. This model would look like this:



Text Classification

- To train such a classification model, you mainly have to train the classifier, with minimal changes happening to the underlying BERT model during the training phase.
- This training process is called Fine-Tuning, and has roots in Semi-supervised Sequence Learning and ULMFiT.
- Classification tasks involve supervised-learning machine learning.
  - Need a labeled dataset to train such a model.
  - For the spam classifier example, the labeled dataset would be a list of email messages and a label ("*spam*" or "*not spam*" for each message).

NLP

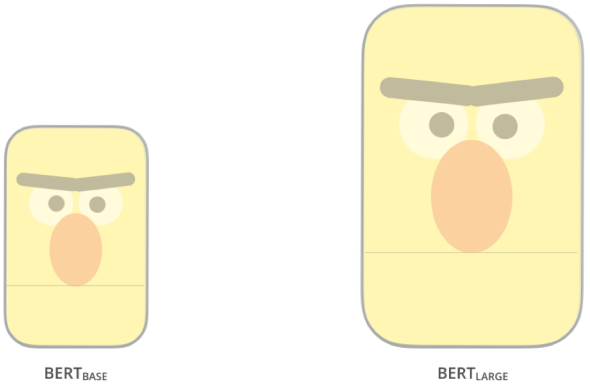
Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached…	Not Spam

Other Classification Examples:

- **Sentiment analysis**
  - Input: Movie/Product review. Output: is the review positive or negative?
  - Example dataset: SST
- **Fact-checking**
  - Input: sentence. Output: “Claim” or “Not Claim”
  - More ambitious/futuristic example:
    - Input: Claim sentence. Output: “True” or “False”
  - Full Fact is an organization building automatic fact-checking tools for the benefit of the public. Part of their pipeline is a classifier that reads news articles and detects claims (classifies text as either “claim” or “not claim”) which can later be fact-checked (by humans now, by with ML later, hopefully).

BERT Model Architecture

How it works



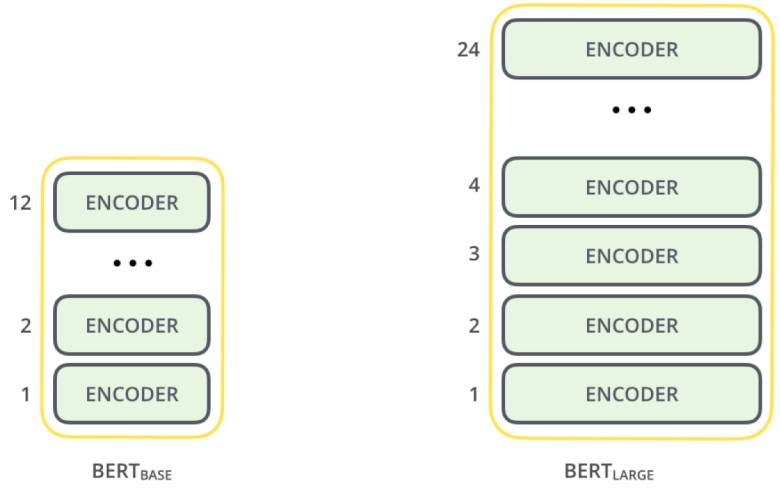
BERT has two model sizes:

- BERT BASE – Comparable in size to the OpenAI Transformer in order to compare performance
- BERT LARGE – A huge model which achieved the state of the art results reported in the paper

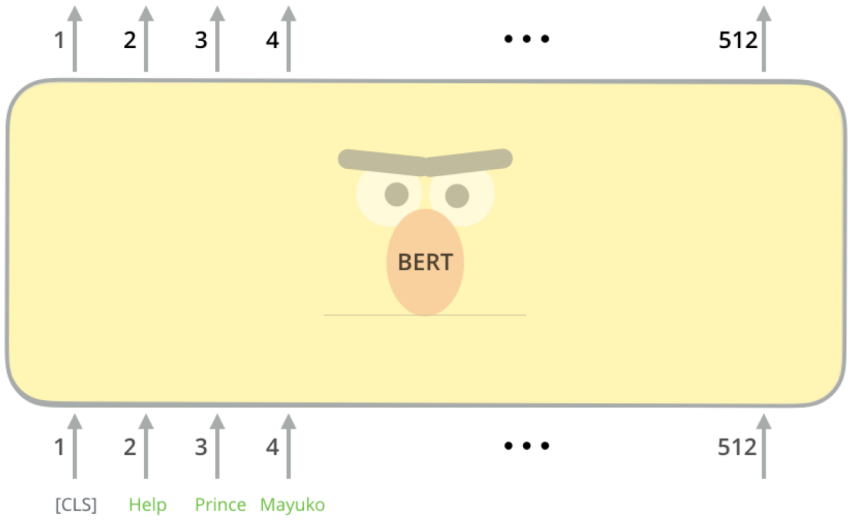
BERT is basically a trained Transformer Encoder stack. See The Illustrated Transformer by Jay Alammaer) which explains the Transformer model – a foundational concept for BERT.

The BERT Model:

- Both BERT model sizes have a large number of encoder layers (also called Transformer Blocks)
  - 12 for the Base version, and 24 for the Large version.
  - These also have larger feedforward-networks (768 and 1024 hidden units respectively),
  - And more attention heads (12 and 16 respectively) than the default configuration in the reference implementation of the Transformer in the initial paper (6 encoder layers, 512 hidden units, and 8 attention heads)

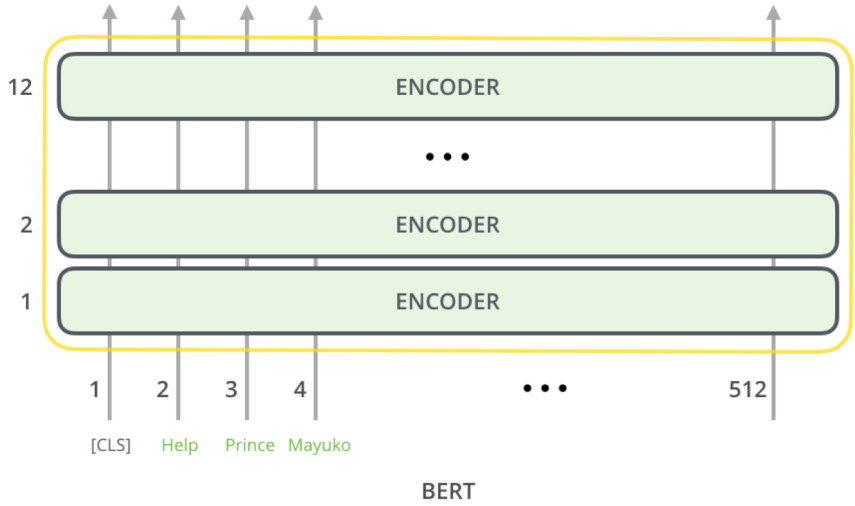


BERT Model Inputs



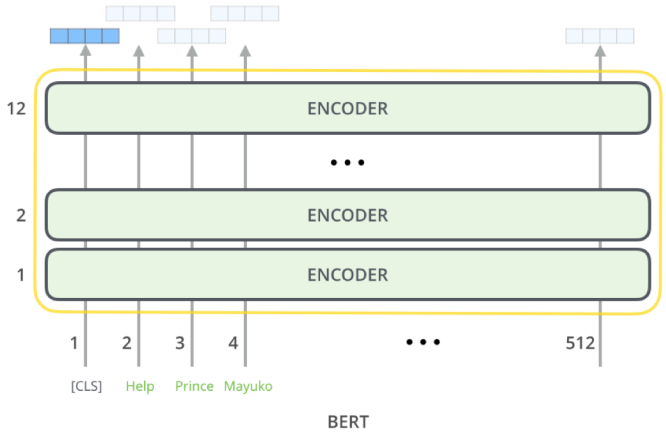
The first input token is supplied with a special [CLS] token for reasons that will become apparent later on. CLS here stands for Classification.

BERT Model



BERT takes a sequence of words as input which keep flowing up the stack. Each layer applies its results through a feed-forward neural network, and then hands it off to the next encoder.

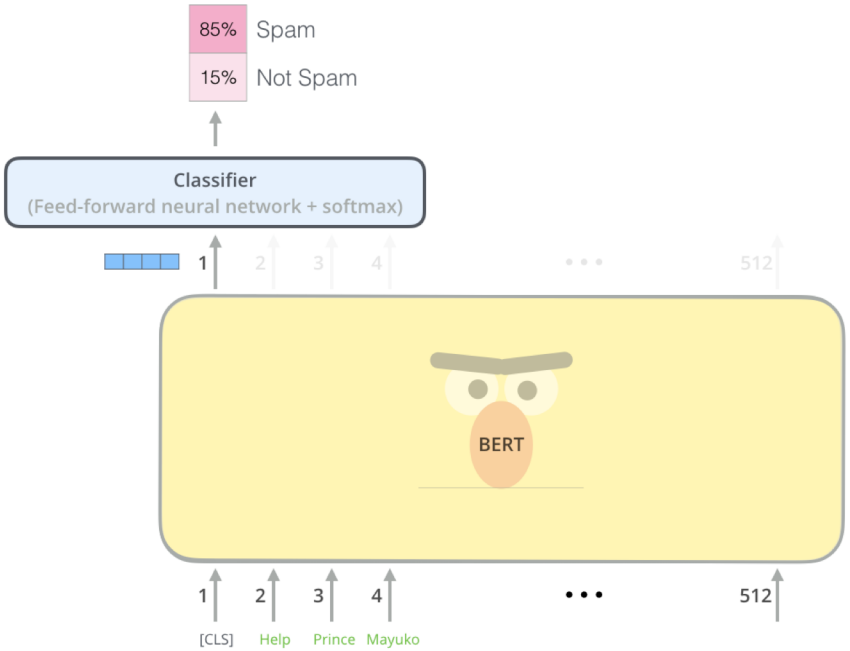
Model Outputs



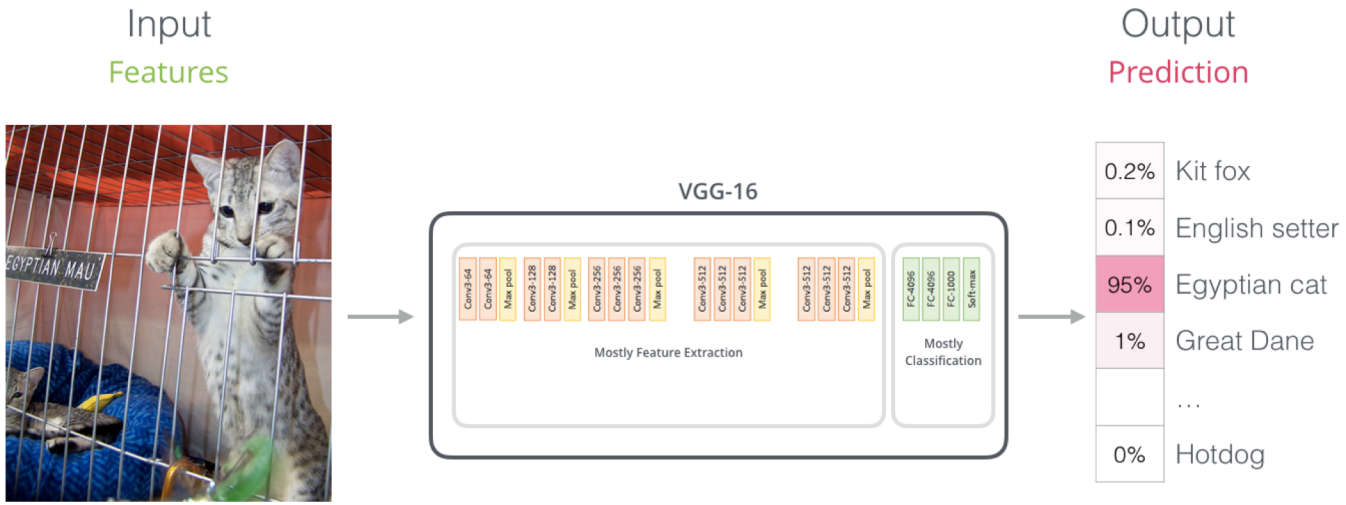
Each position outputs a vector of size *hidden_size* (768 in BERT Base). For the sentence classification example we've looked at above, we focus on the output of only the first position (that we passed the special [CLS] token to).

That vector can now be used as the input for a classifier of our choosing. The original BERT model achieves great results by just using a single-laye neural network as the classifier.

If you have more labels (for example if you're an email service that tags email with "spam", "not spam", "social", and "promotion"), you just tweak the classifier network to have more outputs



Parallels with Convolutional Nets for Images



This vector hand-off is similar to what happens between the convolution part of a image analysis network like ***VGGNet*** and the fully-connected classification portion at the end of the network.

The New Age of Embedding

- These new developments with models like BERT carry with them a new shift in how words are encoded.
- Up until now, word-embeddings have been a major force in how leading NLP models deal with language.
- Methods like Word2Vec and Glove have been widely used for such tasks. Let’s recap how those are used before pointing to what has now changed.

Aside: What are Word Embeddings?

- For words to be processed by machine learning models, they need some form of numeric representation that models can use in their calculation.
- Word2Vec showed that we can use a vector (a list of numbers) to properly represent words in a way that captures:
  1. *Semantic* or meaning-related relationships (e.g. the ability to tell if words are similar, or opposites, or that a pair of words like “Stockholm” and “Sweden” have the same relationship between them as “Cairo” and “Egypt” have between them), as well as
  2. *Syntactic*, or grammar-based, relationships (e.g. the relationship between “had” and “has” is the same as that between “was” and “is”).

Word Embeddings ...

- Great idea to use embeddings that were pre-trained on vast amounts of text data instead of training them alongside the model on what was frequently a small dataset.
- So, download a list of words and their embeddings generated by pre-training with ***Word2Vec*** or ***GloVe***. This is an example of the GloVe embedding of the word “stick” (with an embedding vector size of 200).

-0.34	-0.84	0.20	-0.26	-0.12	0.23	1.04	-0.16	0.31	0.06	0.30	0.33	-1.17	-0.30	0.03	0.09	0.35	-0.28	-(

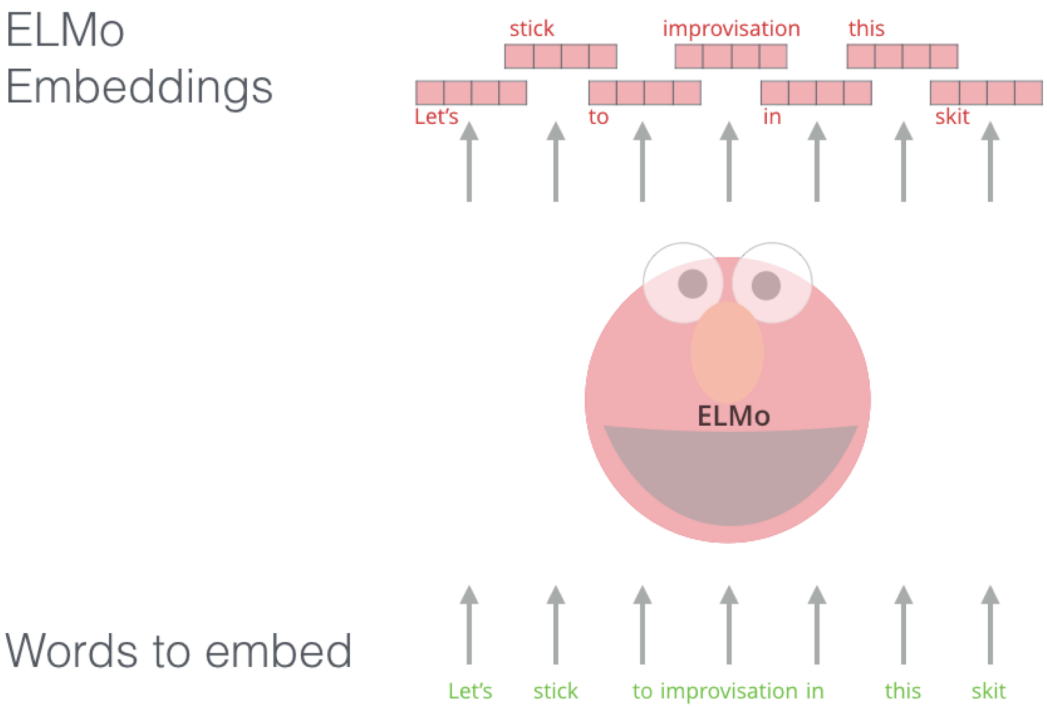
The GloVe word embedding of the word "stick" - a vector of 200 floats (rounded to two decimals). It goes on for two hundred values.

Since embeddings are large & full of numbers, use this to show vectors:

ELMo: Context Matters

- If we're using this GloVe representation, then the word "stick" would be represented by this vector no-matter what the context was.
- "Wait a minute" said a number of NLP researchers (Peters et. al., 2017, McCann et. al., 2017, and yet again Peters et. al., 2018 in the ELMo paper), "*stick*"" has multiple meanings depending on where it's used.
- Why not give it an embedding based on the context it's used in – to both capture the word meaning in that context as well as other contextual information?.
    - And so, *contextualized* word-embeddings were born.

Contextualized Embeddings



Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding. It uses a bi-directional LSTM trained on a specific task to be able to create those embeddings.

ELMo

- ELMo provided a significant step towards pre-training in the context of NLP
  - The ELMo LSTM is trained on a massive text dataset (pre-training), and then it can be used as a component in other models that need to handle language.
- What's ELMo's secret?
  - ELMo gained its language understanding from being trained to predict the next word in a sequence of words - a task called *Language Modeling*.
  - This is convenient because we have vast amounts of text data that such a model can learn from **without needing labels**.

Next Word Prediction:
OpenAI GPT-2 Model Example

GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a very large dataset. OpenAI created the dataset which emphasizes diversity of content, by scraping content from 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text.

https://demo.allennlp.org/next-token-lm

https://transformer.huggingface.co/doc/gpt2-large

ULM-FiT: Nailing down Transfer Learning in NLP

- ULM-FiT introduced methods to effectively utilize a lot of what the model learns during pre-training – more than just embeddings, and more than contextualized embeddings.
- ULM-FiT introduced a language model and a process to effectively fine-tune that language model for various tasks.
- NLP finally had a way to do transfer learning probably as well as Computer Vision could.

The Transformer: Going beyond LSTMs

The release of the ***Transformer*** paper and code, and the results it achieved on tasks such as machine translation made it state of the art for NLP. Transformers also deals with long-term word dependancies better than other models.

But how would you use ***Transformer*** for sentence classification? How would you use it to pre-train a language model that can be fine-tuned for other tasks (*downstream* tasks is what the field calls those supervised-learning tasks that utilize a pre-trained model or component).

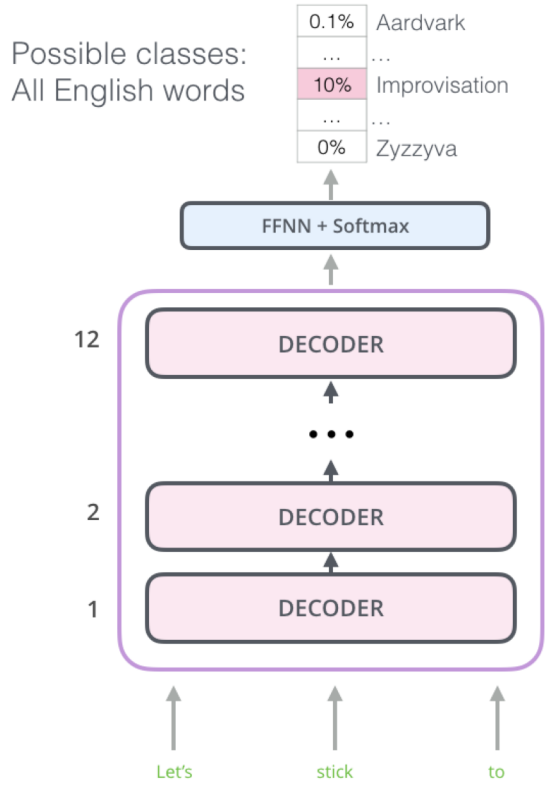
OpenAI Transformer: Pre-training a Transformer Decoder for Language Modeling

- It turns out we don't need an entire Transformer to adopt transfer learning and a fine-tunable language model for NLP tasks.
- We can do with just the decoder of the transformer.
- The decoder is a good choice because it's a natural choice for language modeling (predicting the next word) since it's built to mask future tokens – a valuable feature when it's generating a translation word by word.

Transformers

The OpenAI Transformer is made up of the decoder stack from the Transformer

OpenAI Transformer

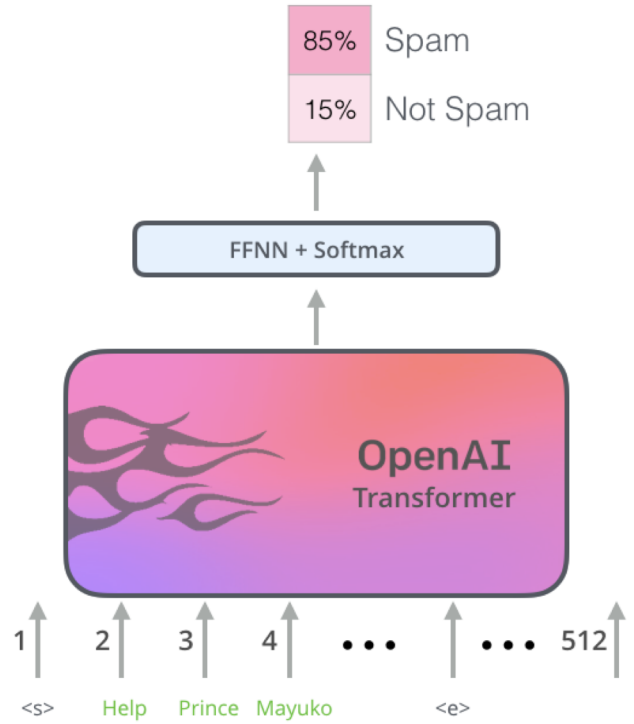


The OpenAI Transformer is now ready to be trained to predict the next word on a dataset made up of 7,000 books

With this structure, we can proceed to train the model on the same language modeling task: predict the next word using massive (unlabeled) datasets. Just, throw the text of 7,000 books at it and have it learn! Books are great for this sort of task since it allows the model to learn to associate related information even if they're separated by a lot of text – something you don't get for example, when you're training with tweets, or articles.

Transfer Learning to Downstream Tasks

Now that the OpenAI transformer is pre-trained and its layers have been tuned to reasonably handle language, we can start using it for downstream tasks. Let's first look at sentence classification (classify an email message as "spam" or "not spam"):



How to use a pre-trained OpenAI transformer to do sentence classification

BERT: From Decoders to Encoders

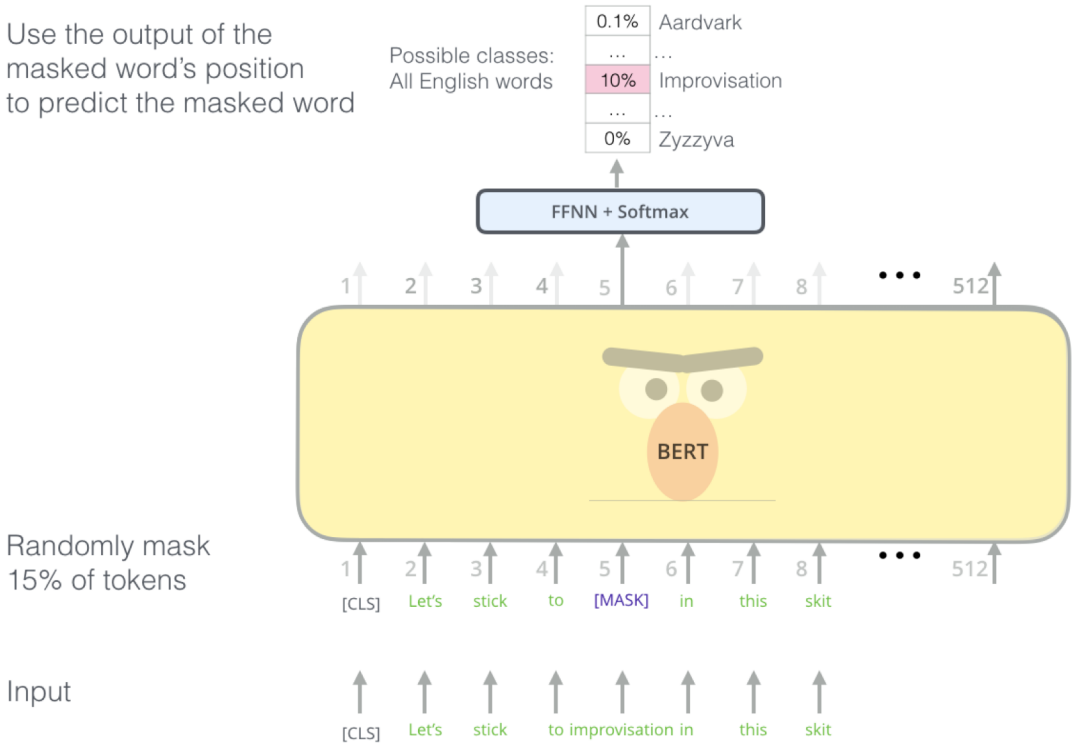
- The OpenAI Transformer gave us a fine-tunable pre-trained model based on the Transformer
- But something went missing in this transition from LSTMs to Transformers.
- ELMo's language model was bi-directional, but the openAI transformer only trains a forward language model.
- Could we build a transformer-based model whose language model looks both forward and backwards (in the technical jargon – "is conditioned on both left and right context")?

Masked Language Model

"We'll use transformer encoders", said BERT.

"This is madness", replied Ernie, "Everybody knows bidirectional conditioning would allow each word to indirectly see itself in a multi-layered context."

"We'll use masks", said BERT confidently.



BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

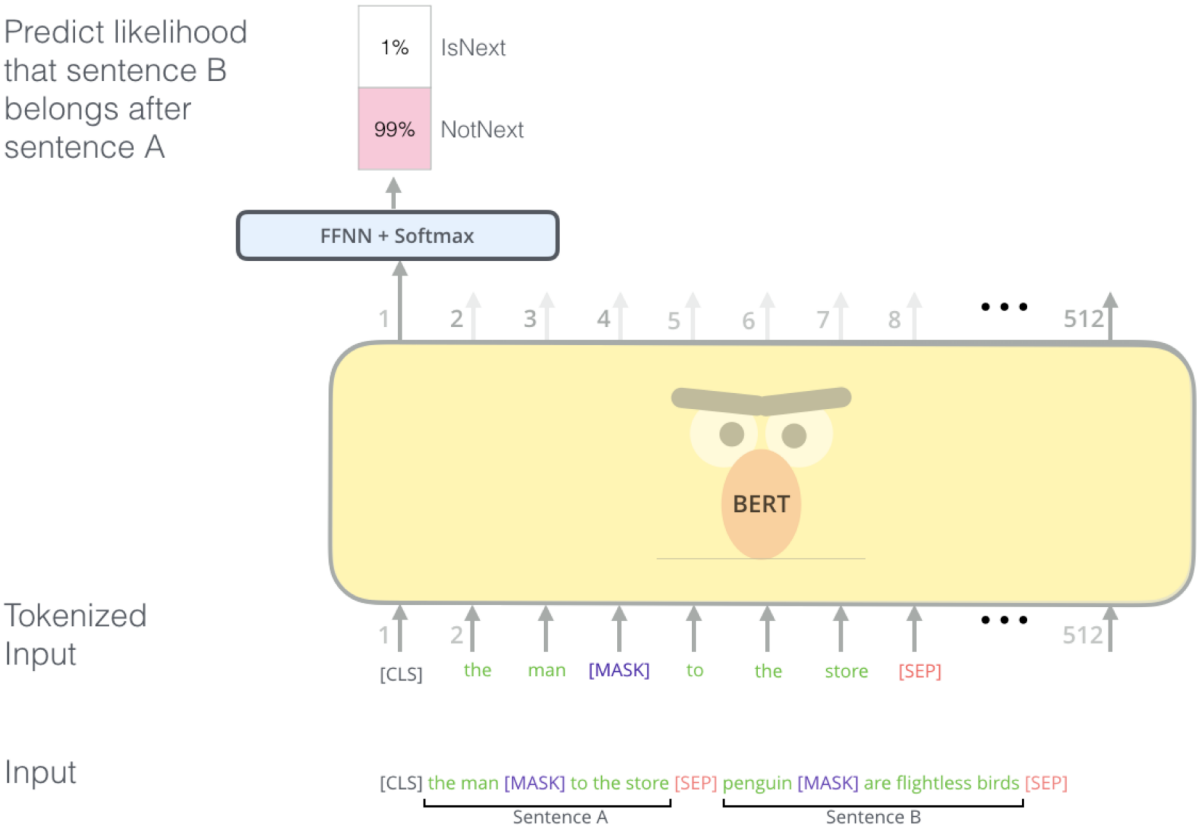
Masking in BERT

- Finding the right task to train a Transformer stack of encoders is a complex hurdle that BERT resolves by adopting a “masked language model” concept.
- Beyond masking 15% of the input, BERT also mixes things a bit in order to improve how the model later fine-tunes.
- Sometimes it randomly replaces a word with another word and asks the model to predict the correct word in that position.

Two-sentence tasks

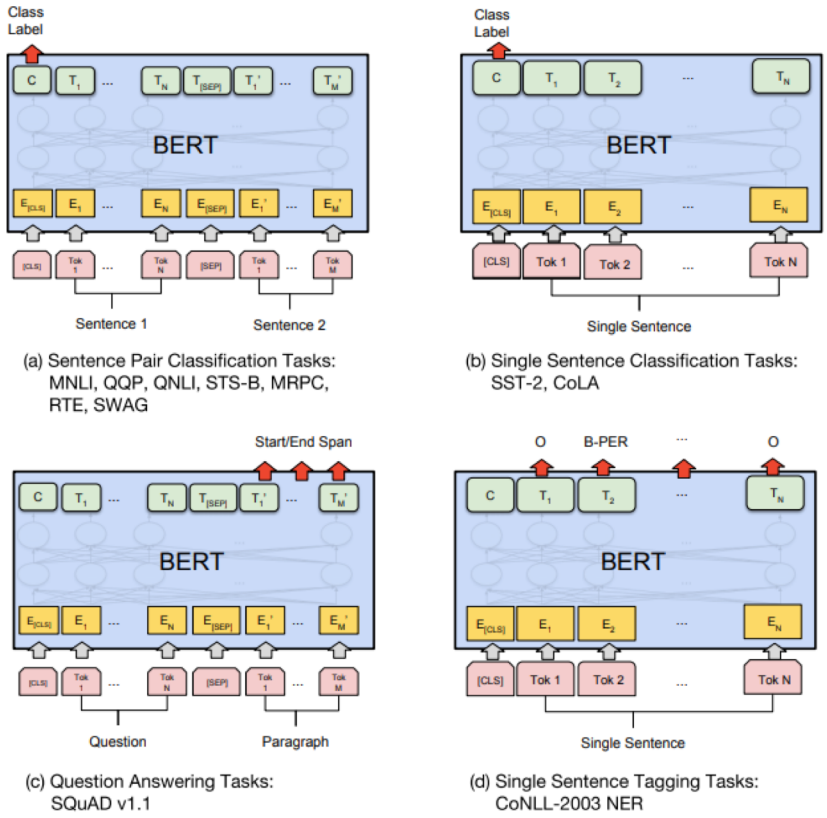
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:
- Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

NLP



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

Task-Specific Models: BERT can be used for a number of tasks:



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Acknowledgements and Attribution

Thanks to Jacob Devlin, Matt Gardner, Kenton Lee, Mark Neumann, and Matthew Peters for providing feedback on earlier drafts of this post.

Written on December 3, 2018