Understanding the applications of Probability in Machine Learning

This post is part of my forthcoming book **The Mathematical Foundations of Data Science**. Probability is one of the foundations of machine learning (along with linear algebra and optimization). In this post, we discuss the areas where probability theory could apply in machine learning applications. If you want to know more about the book, follow me on [Ajit Jaokar linked](https://www.linkedin.com/in/ajitjaokar/)

Background

First, we explore some background behind probability theory

Probability as a measure of uncertainty

Probability is a measure of uncertainty. Probability applies to machine learning because in the real world, we need to make decisions with incomplete information. Hence, we need a mechanism to quantify uncertainty – which Probability provides us. Using probability, we can model elements of uncertainty such as risk in financial transactions and many other business processes. In contrast, in traditional programming, we work with deterministic problems i.e. the solution is not affected by uncertainty.

Probability of an event

Probability quantifies the likelihood or belief that an event will occur. Probability theory has three important concepts: **Even**t - an outcome to which a probability is assigned; The **Sample Space** which represents the set of possible outcomes for the events and the **Probability Function** which maps a probability to an event. The probability function indicates the likelihood that the event being a part of the sample space is drawn. The **probability distribution** represents the shape or distribution of all events in the sample space. The probability of an event can be calculated directly by counting all the occurrences of the event and dividing them by the total possible outcomes of the event. Probability is a fractional value and has a value in the range between 0 and 1, where 0 indicates no probability and 1 represents full probability.

Two Schools of Probability

There are two ways of interpreting probability: frequentist **probability** which considers the actual likelihood of an event and the **Bayesian probability** which considers how strongly we believe that an event will occur. frequentist probability includes techniques like **p-values** and **confidence intervals** used in statistical inference and **maximum likelihood** estimation for **parameter estimation**.

Frequentist techniques are based on counts and Bayesian techniques are based on beliefs. In the Bayesian approach, probabilities are assigned to events based on evidence and personal belief. The Bayesian techniques are based on the Bayes’ theorem. Bayseian analysis can be used to model events that have not occurred before or occur infrequently. In contrast, frequentist techniques are based on **sampling** – hence the frequency of occurrence of an event.  For example, the [p­Value](https://www.investopedia.com/terms/p/p-value.asp) indicates a number between 0 and 1. The larger the p-value – the more the data conforms to the null hypothesis. The smaller the p-value, the more the data conforms to the alternate hypothesis. If p-value is less than 0.05, then we reject the null hypothesis i.e. accept the alternate hypothesis.

Applications

With this background, let us explore how probability can apply to machine learning

Sampling - Dealing with non-deterministic processes

Probability forms the basis of sampling. In machine learning, uncertainty can arise in many ways – for example - noise in data. Probability provides a set of tools to model uncertainty. Noise could arise due to variability in the observations, as a measurement error or from other sources. Noise effects both inputs and outputs.

Apart from noise in the sample data, we should also cater for the effects of bias. Even when the observations are uniformly sampled i.e. no bias is assumed in the sampling – other limitations can introduce bias. For example, if we choose a set of participants from a specific region of the country., by definition. the sample is biased to that region. We could expand the sample scope and variance in the data by including more regions in the country. We need to balance the variance and the bias so that the sample chosen is representative of the task we are trying to model.

Typically, we are given a dataset i.e. we do not have control on the creation and sampling process of the dataset. To cater for this lack of control over sampling, we split the data into train and test sets or we use resampling techniques. ***Hence, probability (through sampling) is involved when we have incomplete coverage of the problem domain.***

Pattern recognition

Pattern recognition is a key part of machine learning. We can approach machine learning as a pattern recognition problem from a Bayesian standpoint. In [Pattern Recognition](https://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738) – Christopher Bishop takes a Bayesian view and presents approximate inference algorithms for situations where exact answers are not feasible. For the same reasons listed above, Probability theory is a key part of pattern recognition because it helps to cater for noise / uncertainty and for the finite size of the sample and also to apply Bayesian principles to machine learning.

Training -  use in Maximum likelihood estimation

Many iterative machine learning techniques like [Maximum likelihood estimation](https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1) (MLE) are based on probability theory. MLE is used for training in models like linear regression, logistic regression and artificial neural networks.

Developing specific algorithms

Probability forms the basis of specific algorithms like [Naive Bayes classifier](https://en.wikipedia.org/wiki/Naive_Bayes_classifier)

Hyperparameter optimization

In machine learning models such as neural networks, hyperparameters are tuned through techniques like grid search. Bayesian optimization can be also used for hyperparameter optimization.

Model evaluation

In binary classification tasks, we predict a single probability score. Model evaluation techniques require us to summarize the performance of a model based on predicted probabilities. For example – aggregation measures like [log loss](http://wiki.fast.ai/index.php/Log_Loss) require the understanding of probability theory

**Applied fields of study**

Probability forms the foundation of many fields such as physics, biology, and computer science where maths is applied.

**Inference**

Probability is a key part of inference - MLE for frequentist and Bayesian inference for Bayesian

Conclusion

As we see above, there are many areas of machine learning where probability concepts apply. Yet, they are not so commonly taught in typical coding programs on machine learning. In the last blog, we discussed this trend in context of [correlation vs causation.](https://www.datasciencecentral.com/profiles/blogs/correlation-does-not-equal-causation-but-how-exactly-do-you) I suspect the same is true i.e. the starting point for most developers is a dataset which they are already provided. In contrast, if you conduct a PhD experiment / thesis – you have to typically build your experiment from scratch.

Current Issues with the Transfer Learning

*Natural Language Processing (NLP) has recently witnessed dramatic progress with state-of-the-art results being published every few days. Leaderboard madness is diriving the most common NLP benchmarks such as GLUE and SUPERGLUE with scores that are getting closer and closer to human-level performance. Most of these results are driven by transfer learning from large scale datasets through super large (Billions of parameters) models. My aim in this article is to point out the issues and challenges facing transfer learning and point out some possible solutions to such problems.*

**Computational Intensity**

The most successful form of **Transfer Learning** in NLP today is **Sequential Transfer Learning (STL)**, which is typically employed in the form of [Language Modeling Pre-training](https://arxiv.org/abs/1801.06146). Almost all SOTA results achieved recently have been mainly driven by a two-step scheme:

1. **Pre-train** a monster model for Language Modeling on a large general-purpose corpus (The more data the better).
2. **Finetune** the whole model (or a subset thereof) on the target task.

[ELMO](https://arxiv.org/abs/1802.05365), [BERT](https://arxiv.org/abs/1810.04805), [GPT](https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf), [GPT-2](https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf), [XLNET](https://arxiv.org/abs/1906.08237) and [RoBERTa](https://arxiv.org/abs/1907.11692) are all instances of the same technique. One major problem with these methods is the tremendous resource craveness. What I mean by resources is both *data* and *compute power*. For instance, it has been estimated that it costs around [$250,000](https://twitter.com/eturner303/status/1143174828804857856) to train XLNET on 512 TPU v3 chips with only 1-2% gain over BERT in 3/4 datasets.

This takes us to the next issue:

**Difficult Reproducibility**

Reproducibility is a already becoming a problem in machine learning research. For example, [Dacrema et al.)](https://arxiv.org/pdf/1907.06902) analyzed 18 different proposed Neural-based Recommendation Systems and *found that only 7 of them were reproducible with reasonable effort*. Generally speaking, to be able to use or build upon a particular research idea, it’s imperative for that idea to be easily reproducible. With the substantial computational resources needed to train these huge NLP models and reproduce their results, small tech companies, startups, research labs and independent researchers will not be able to compete.

**Task Leaderboards Are No Longer Enough**

Anna Rogers argues in her [blog post](https://hackingsemantics.xyz/2019/leaderboards/?utm_campaign=NLP%20News&utm_medium=email&utm_source=Revue%20newsletter) why more data & compute = SOTA is NOT research news. She argues that the main problem with leaderboards is that the rank of a model is totally dependent on its task score with no consideration given to the amount of data, compute or training time needed to achieve that score.

**[Anna Rogers](https://twitter.com/annargrs)**[@annargrs](https://twitter.com/annargrs)

Here's a summary post on problems with huge models that dominate [#NLProc](https://twitter.com/hashtag/NLProc?src=hash) these days. I put together several different discussion threads with/by [@yoavgo](https://twitter.com/yoavgo), [@jaseweston](https://twitter.com/jaseweston), [@sleepinyourhat](https://twitter.com/sleepinyourhat), [@bkbrd](https://twitter.com/bkbrd), [@alex\_conneau](https://twitter.com/alex_conneau), [@SeeTedTalk](https://twitter.com/SeeTedTalk). [https://hackingsemantics.xyz/2019/leaderboards/ …](https://t.co/MokmmEYx91)

**[How the Transformers broke NLP leaderboards](https://t.co/MokmmEYx91)**

[With the huge Transformer-based models such as BERT, GPT-2, and XLNet, are we losing track of how the state-of-the-art performance is achieved?](https://t.co/MokmmEYx91)

[hackingsemantics.xyz](https://t.co/MokmmEYx91)

I suggest you check the above thread for various comments on the problem. Rohit Pgarg suggests comparing the performance of models on a two-dimensional scale of both task accuracy and computational resource. See the plot below. I suggest we add another dimension that corresponds to the amount of data the model has been trained on. However, this visualization will not provide an insight into which model is generally better. Also there’s a very interesting comment by Alexandr Savinov where he suggests to use how much of input information the algorithm is able to “pack” to one unit of output (model parameter) representation for one unit of CPU time.

| **https://mohammadkhalifa.github.io/images/scatter-tl.png** |
| --- |
| Using Computational Resource as an additional metric to task accuracy in comparing models performance |

**It is not like How We Learn**

Children learn language through noisy, ambiguous input and minimal supervision. A child can start to pick up the meaning of a word from just a few exposures to that word. This is very unlike the pretraining step used in STL settings where a model need to see millions of contexts including a specific word to grasp the meaning of the word. There is a very important question of whether it is possible to learn semantics from raw text only without any external supervision. If you’re interested in a twitter debate on this topic see this [thread](https://twitter.com/jacobandreas/status/1023246560082063366). If the answer is no, that means that pre-training these models does not actually give them true language understanding. While it’s true that we use transfer learning in our everyday life. For instance, if we know how to drive a manual car, it becomes very easy for us to utilize the acquired knowledge (such as of using the brakes and the gas pedal) to the task of driving an automatic car. But is that the path that we humans take towards language learning? Not likely. One might argue, however, that as long as an approach produces good results, whether it is similar or not to how humans learn does not actually matter. Unfortunately, some of the good results produced by these models are questionable as we will see in the next section.

From another perspective, humans assume a continual (lifelong) learning path towards language learning. Whenever we learn a new task, this learning usually does not interfere with previously learned tasks. On the other hand, the prevalent machine learning models (including transfer learning approaches) that are only trained on a single task usually fail to make use of previously aquired knowledge when the distrubtion of the new training data shifts— a phenomenon known as *catastrophic forgetting* [(McCloskey et al., 1989)](https://www.sciencedirect.com/science/article/pii/S0079742108605368), [(d’Autume et al., 2019)](https://arxiv.org/pdf/1906.01076v1.pdf).

**Shallow Language Understanding**

The language modeling task is indeed a complex task. Take for instance the sentence: “The man in the red shirt is running fast. He must be…” In order for the model to complete that sentence, the model has to understand what running fast usually implies i.e being in a hurry. So how deep do these pretrained models actually understand language? Unfortunately, not so much. [Niven et al., 2019](https://www.aclweb.org/anthology/P19-1459) analyze the performance of BERT on the Argument Reasoning and Comprehension task (ARCT) [(Habernal et al., 2018)](https://arxiv.org/abs/1708.01425). ARCT can be described as follows: Given a Claim CC and a Reason RR, the task is to select the correct Warrant WW over another distractor, the alternative warrant AA. The correct warrant satisfies R∧C→WR∧C→W while the alternative warrant satisfies R∧C→¬AR∧C→¬A. See the figure below.

| **https://mohammadkhalifa.github.io/images/arct.PNG** |
| --- |
| Sample of the Argument Reasoning and Comprehension Task. Source: [(Niven et al., 2019)](https://www.aclweb.org/anthology/P19-1459) |

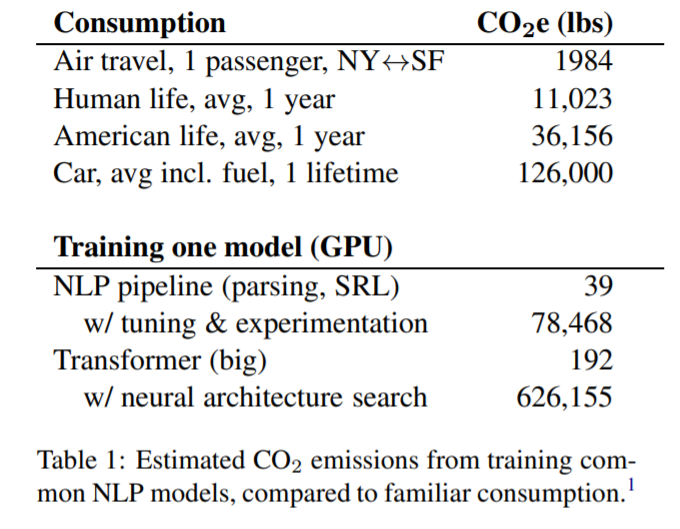
Remarkably, BERT achieves a very competitive accuracy of 77% on this task, which is only 3 points below the human baseline. At first, this would suggest that BERT has a quite strong reasoning ability. To investigate further, [Niven et al., 2019](https://www.aclweb.org/anthology/P19-1459) employed what is known as “probing”. That is, they finetuned BERT on this task, yet the input to BERT was only both the correct and the alternative warrants without exposing it to either the claim or the reason. The hypothesis is that if BERT relies on some statistical cues in the warrants, it should still perfom well even if it has only seen the warrants without any other information. Interestingly, their results show only a drop of 6% in accuracy over using both Reason and Claim. This suggests that BERT is not actually performing any type of reasoning but that the warrants themselves have sufficient cues for BERT to be able to reach such high accuracy. Remarkably, by replacing the test set with an adversarial one that is free of these cues the BERT relies on, BERT was only able to achieve an accuracy of 53%, which is just above random chance.

Another related paper is [(Zellers et al., 2019)](https://arxiv.org/pdf/1905.07830.pdf) titled “Can a Machine Really Finish your Sentence?”. They consider the task of Commonsense Natural Language Inference where a machine should select the most likely follow up to given sentence. For instance,given the sentence: “The team played so well”, the system should select “They won the game” as a follow up. The authors argue that altough BERT was able to achieve almost 86% accuracy (only 2 points below human-baseline), such high accuracy is not due high-level form of reasoning on BERT’s side but due to BERT learning to pick up on dataset-specific distributional biases. They showed that by creating a more difficult dataset (HellaSwag) by means of **Adversarial Filtering** (which is a technique that aims to produce an adversarial dataset for any possible train, test split), BERT accuracy dropped to as low as 53%. The paper discusses the subtle distinction between **Dataset Performance** and **Task Performance**. Performing very well on a dataset for a specific task by no means indicates solving the underlying task.

| **https://mohammadkhalifa.github.io/images/hellaswag.PNG** |
| --- |
| Performance of BERT on SWAG compare to HellaSwag. Source: [(Zellers et al., 2019)](https://arxiv.org/pdf/1905.07830.pdf) |

Clearly, there is something going on here. Is it possible that BERT’s good results are actually driven by its ability to hijack the target datasets by leveraging various distributinoal cues and biases? Can more investigation into BERT’s results lead to other similar findings and conclusions? If so, then I believe we will not only need to build better models, but also better datasets. We need to have datsets that can actually reflect the difficuly of the underlying task rather than make it easy for the model to achieve deceiving accuracies and leaderboard scores.

**High Carbon Footprint**

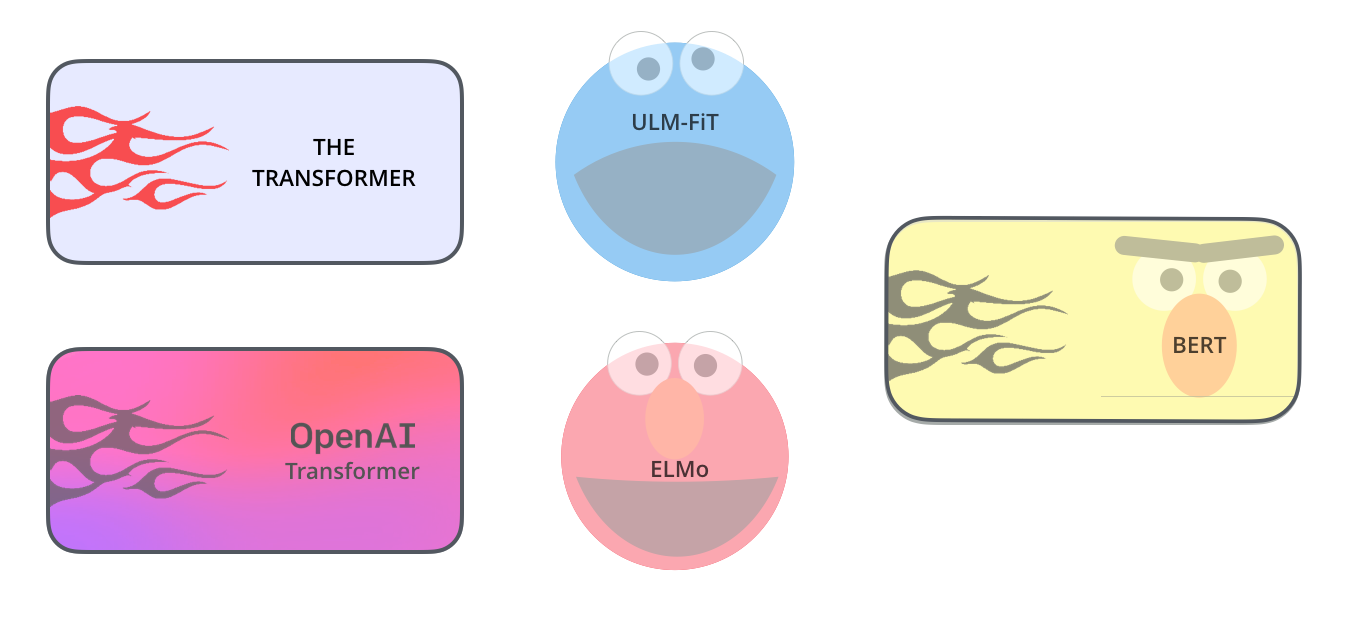
Believe it or not, but training these grandiose models has a negative effect on the environment. [Strubell et al.](https://arxiv.org/pdf/1906.02243.pdf) compare the estimated CO2CO2 emissions from training Big Transformer architecture to emissions caused by other CO2CO2 sources. Suprisingly, training a single Transformer arhcitectue with neural architecture search emits approximately 6.0x the amount of CO2CO2 emitted through the lifetime of a car.

[Schwartz et al.](https://arxiv.org/pdf/1907.10597) introduce what they call *Green AI*, which is the practice of making AI both more *efficient* and *inclusive*. Similar to what we discussed above, they strongly suggest adding efficiency as another metric alongside task accuracy. They also believe it’s necessary for research papers to include the “price tag” or the cost of model training. This should encourage the research towards more efficient and less resource-demanding model architectures.

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

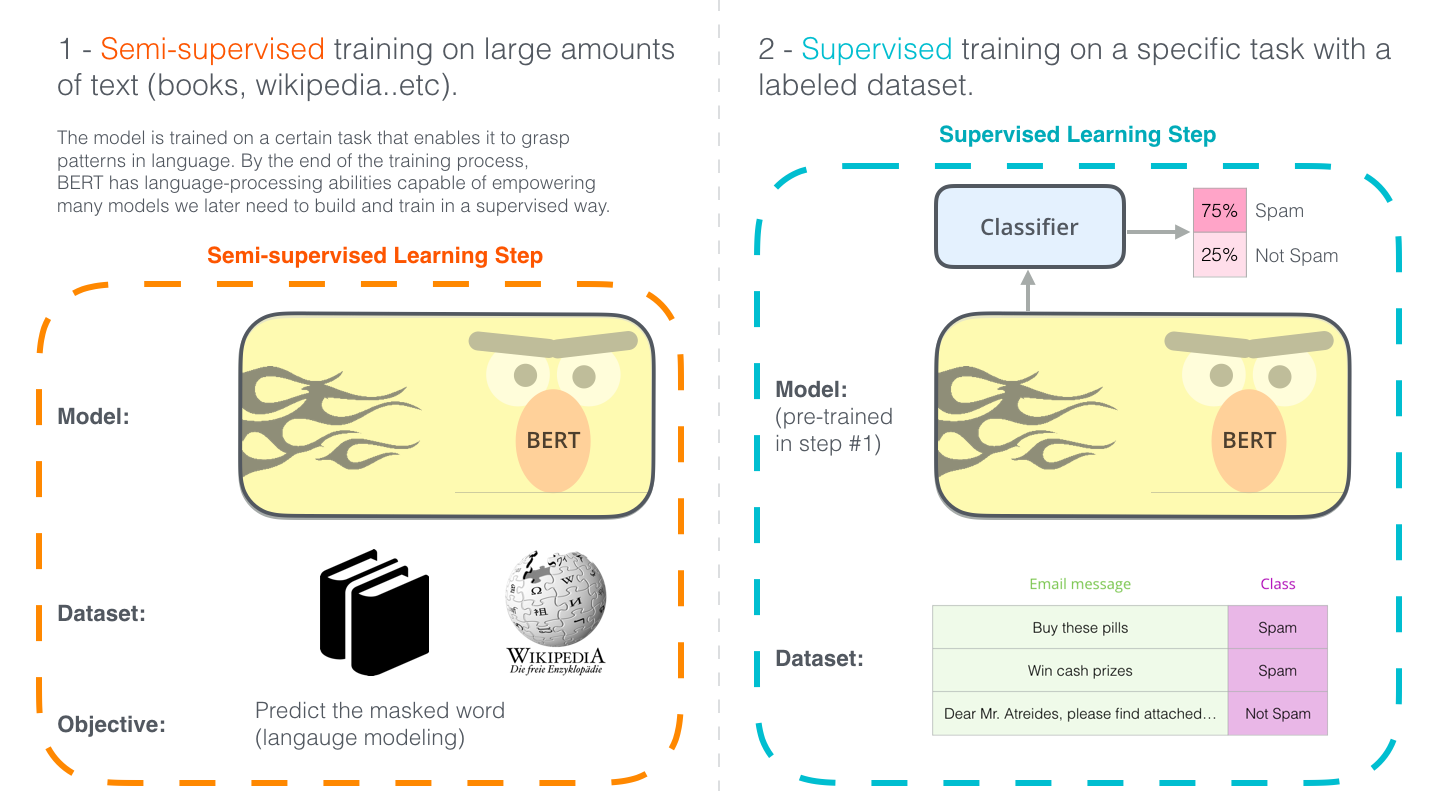
Discussions: [Hacker News (98 points, 19 comments)](https://news.ycombinator.com/item?id=18751469), [Reddit r/MachineLearning (164 points, 20 comments)](https://www.reddit.com/r/MachineLearning/comments/a3ykzf/r_the_illustrated_bert_and_elmo_how_nlp_cracked/)  
Translations: [Chinese (Simplified)](https://blog.csdn.net/qq_41664845/article/details/84787969), [Persian](http://blog.class.vision/1397/09/bert-in-nlp/)

The year 2018 has been an inflection point for machine learning models handling text (or more accurately, Natural Language Processing or NLP for short). 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 (It’s been referred to as [NLP’s ImageNet moment](http://ruder.io/nlp-imagenet/), referencing how years ago similar developments accelerated the development of machine learning in Computer Vision tasks).



(ULM-FiT has nothing to do with Cookie Monster. But I couldn’t think of anything else..)

One of the latest milestones in this development is the [release](https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html) of [BERT](https://github.com/google-research/bert), an event [described](https://twitter.com/lmthang/status/1050543868041555969) 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. Soon after the release of the paper describing the model, the team also open-sourced the code of the model, and made available for download versions of the model that were already pre-trained on massive datasets. This is a momentous development since it enables anyone building a machine learning model involving language processing to use this powerhouse as a readily-available component – saving the time, energy, knowledge, and resources that would have gone to training a language-processing model from scratch.

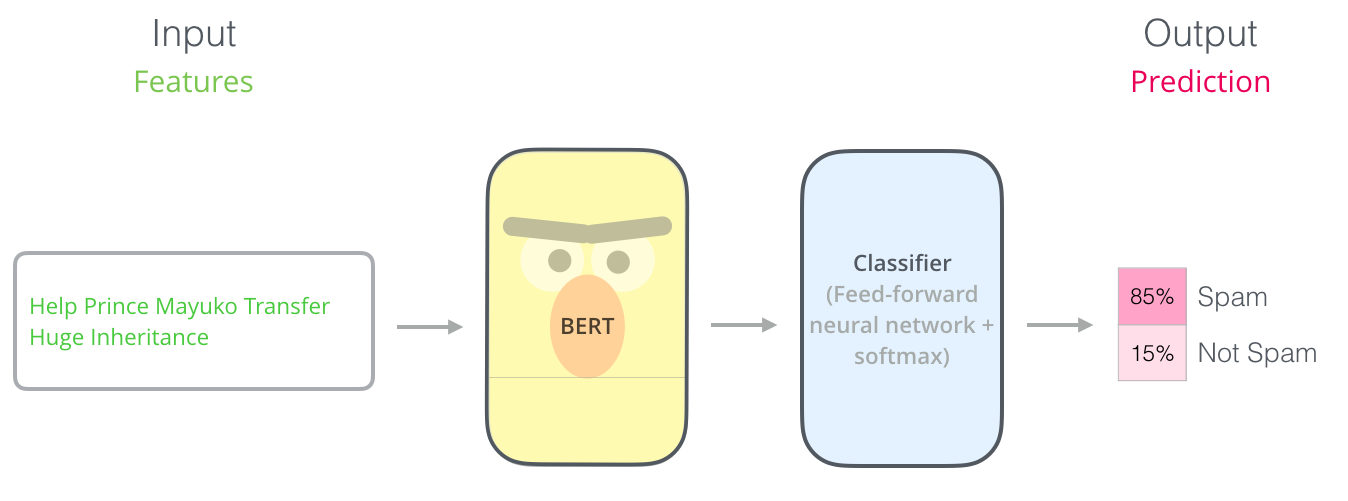
  
The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [[Source](https://commons.wikimedia.org/wiki/File:Documents_icon_-_noun_project_5020.svg) for book icon].

BERT builds on top of a number of clever ideas that have been bubbling up in the NLP community recently – including but not limited to [Semi-supervised Sequence Learning](https://arxiv.org/abs/1511.01432) (by [Andrew Dai](https://twitter.com/iamandrewdai) and [Quoc Le](https://twitter.com/quocleix)), [ELMo](https://arxiv.org/abs/1802.05365) (by [Matthew Peters](https://twitter.com/mattthemathman) and researchers from [AI2](https://allenai.org/) and [UW CSE](https://www.engr.washington.edu/about/bldgs/cse)), [ULMFiT](https://arxiv.org/abs/1801.06146) (by fast.ai founder [Jeremy Howard](https://twitter.com/jeremyphoward) and [Sebastian Ruder](https://twitter.com/seb_ruder)), the [OpenAI transformer](https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf) (by OpenAI researchers [Radford](https://twitter.com/alecrad), [Narasimhan](https://twitter.com/karthik_r_n), [Salimans](https://twitter.com/timsalimans), and [Sutskever](https://twitter.com/ilyasut)), and the Transformer ([Vaswani et al](https://arxiv.org/pdf/1706.03762.pdf)).

There are a number of concepts one needs to be aware of to properly wrap one’s head around what BERT is. So let’s start by looking at ways you can use BERT before looking at the concepts involved in the model itself.

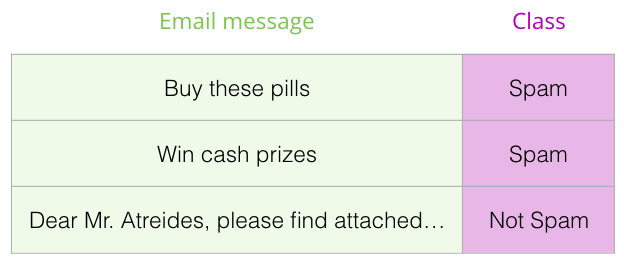
**Example: Sentence Classification**

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:



To train such a model, you mainly have to train the classifier, with minimal changes happening to the BERT model during the training phase. This training process is called Fine-Tuning, and has roots in [Semi-supervised Sequence Learning](https://arxiv.org/abs/1511.01432) and ULMFiT.

For people not versed in the topic, since we’re talking about classifiers, then we are in the supervised-learning domain of machine learning. Which would mean we need a labeled dataset to train such a model. For this spam classifier example, the labeled dataset would be a list of email messages and a labele (“spam” or “not spam” for each message).

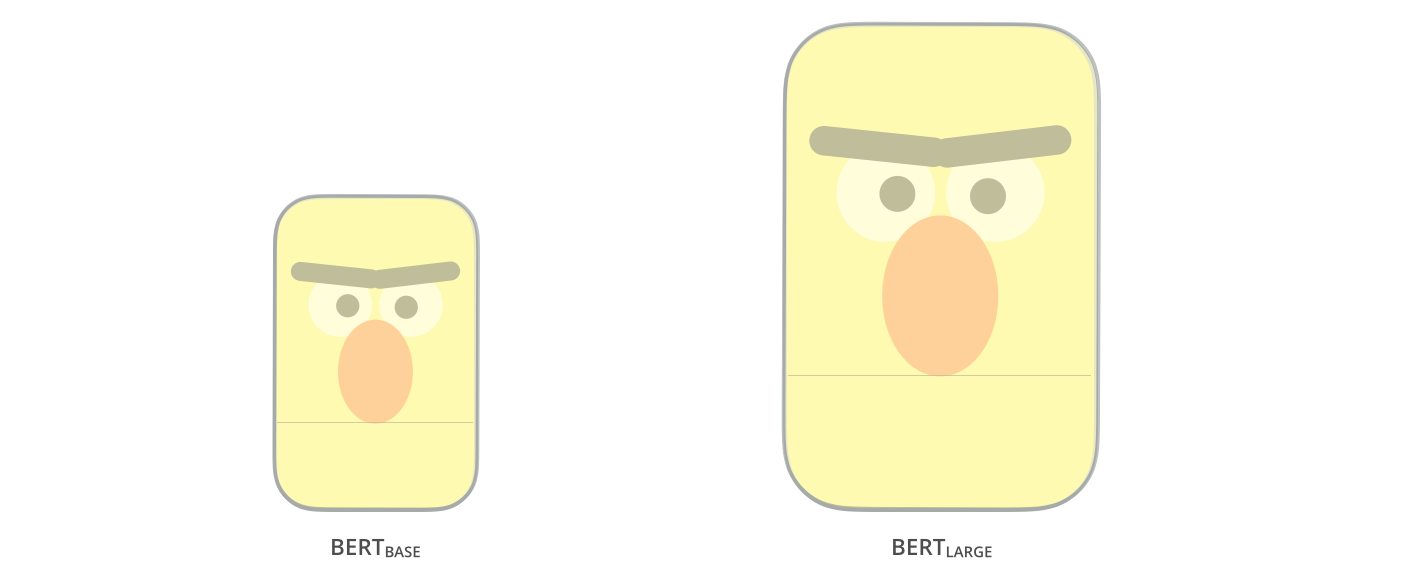


Other examples for such a use-case include:

* **Sentiment analysis**
  + Input: Movie/Product review. Output: is the review positive or negative?
  + Example dataset: [SST](https://nlp.stanford.edu/sentiment/)
* **Fact-checking**
  + Input: sentence. Output: “Claim” or “Not Claim”
  + More ambitious/futuristic example:
    - Input: Claim sentence. Output: “True” or “False”
  + [Full Fact](https://fullfact.org/) 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).
  + Video: [Sentence embeddings for automated factchecking - Lev Konstantinovskiy](https://www.youtube.com/watch?v=ddf0lgPCoSo).

**Model Architecture**

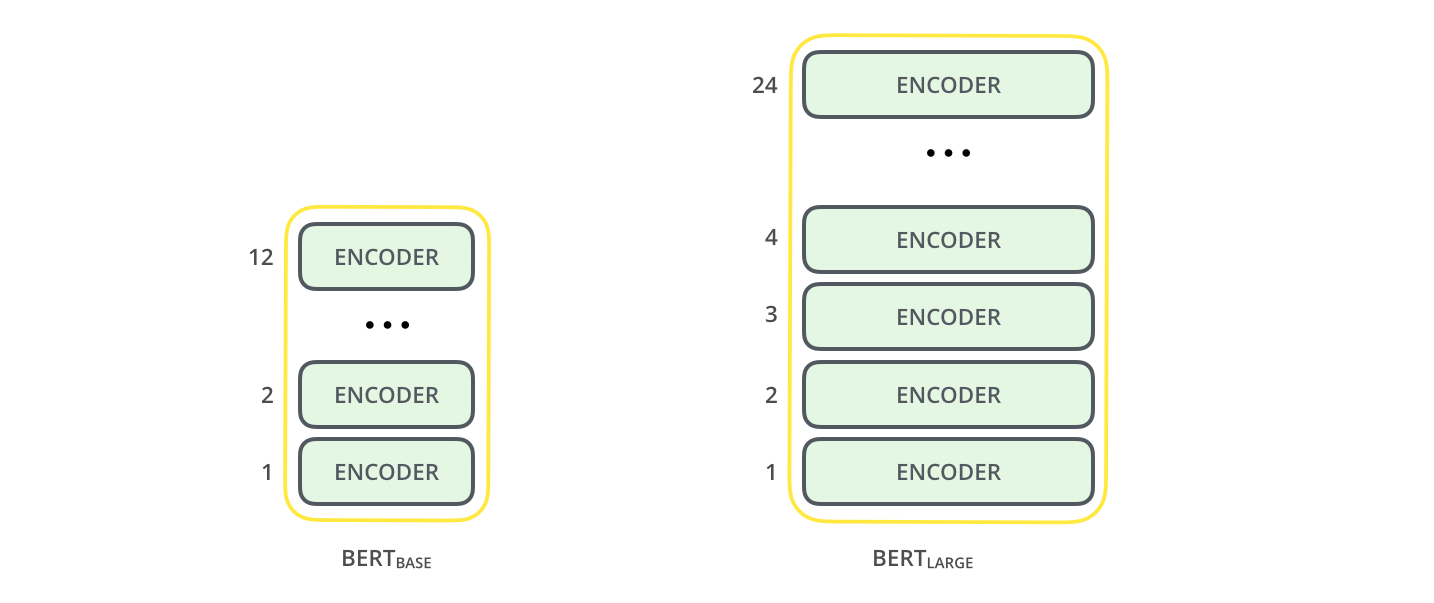
Now that you have an example use-case in your head for how BERT can be used, let’s take a closer look at how it works.



The paper presents two model sizes for BERT:

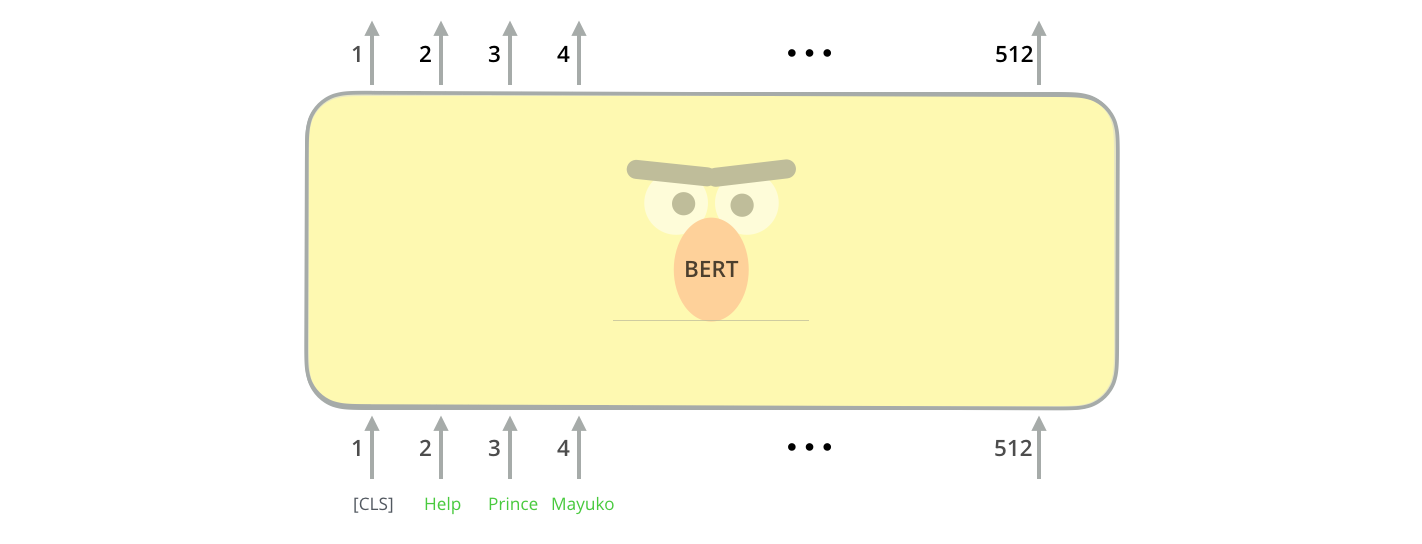
* BERT BASE – Comparable in size to the OpenAI Transformer in order to compare performance
* BERT LARGE – A ridiculously huge model which achieved the state of the art results reported in the paper

BERT is basically a trained Transformer Encoder stack. This is a good time to direct you to read my earlier post [The Illustrated Transformer](https://jalammar.github.io/illustrated-transformer/) which explains the Transformer model – a foundational concept for BERT and the concepts we’ll discuss next.



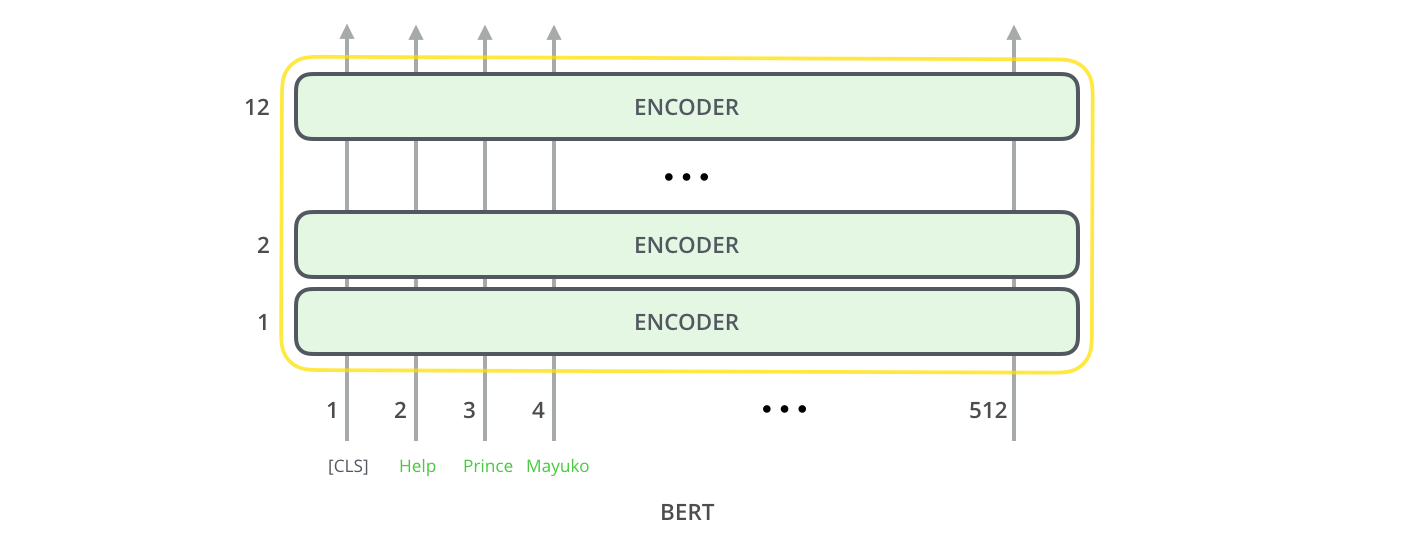
Both BERT model sizes have a large number of encoder layers (which the paper calls Transformer Blocks) – twelve for the Base version, and twenty four for the Large version. These also have larger feedforward-networks (768 and 1024 hidden units respectively), and more attention heads (12 and 16 respectively) 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).

**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.

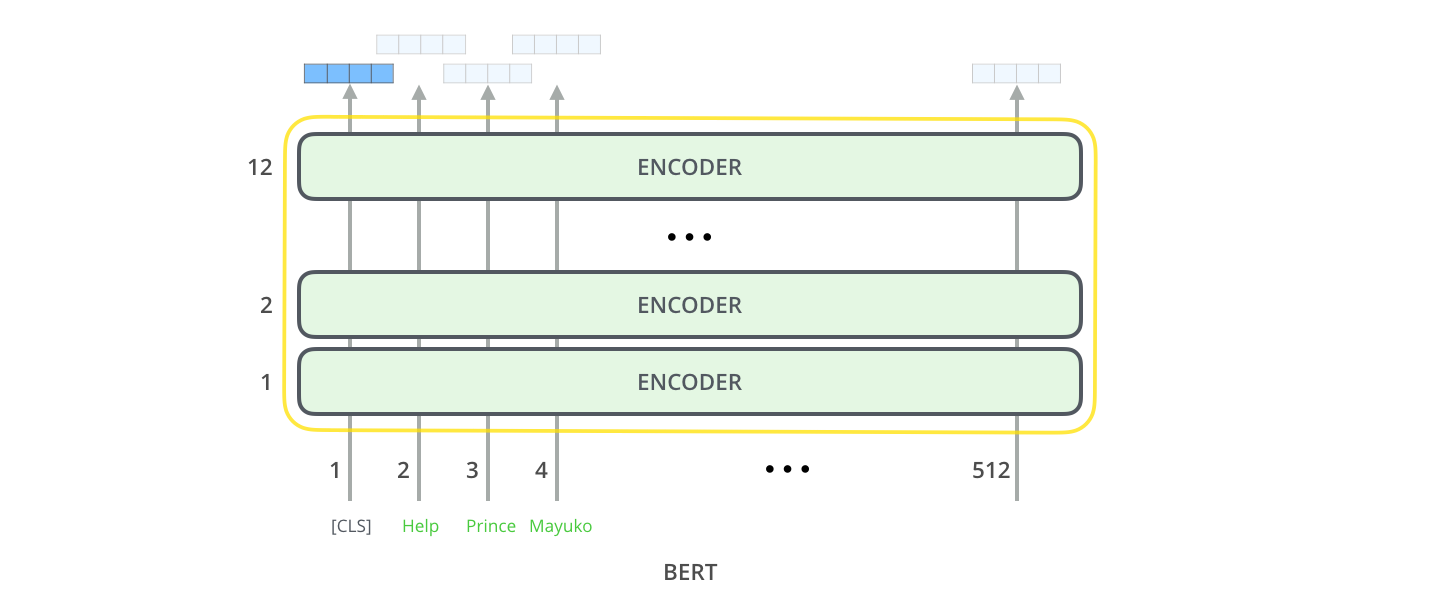
Just like the vanilla encoder of the transformer, BERT takes a sequence of words as input which keep flowing up the stack. Each layer applies self-attention, and passes its results through a feed-forward network, and then hands it off to the next encoder.



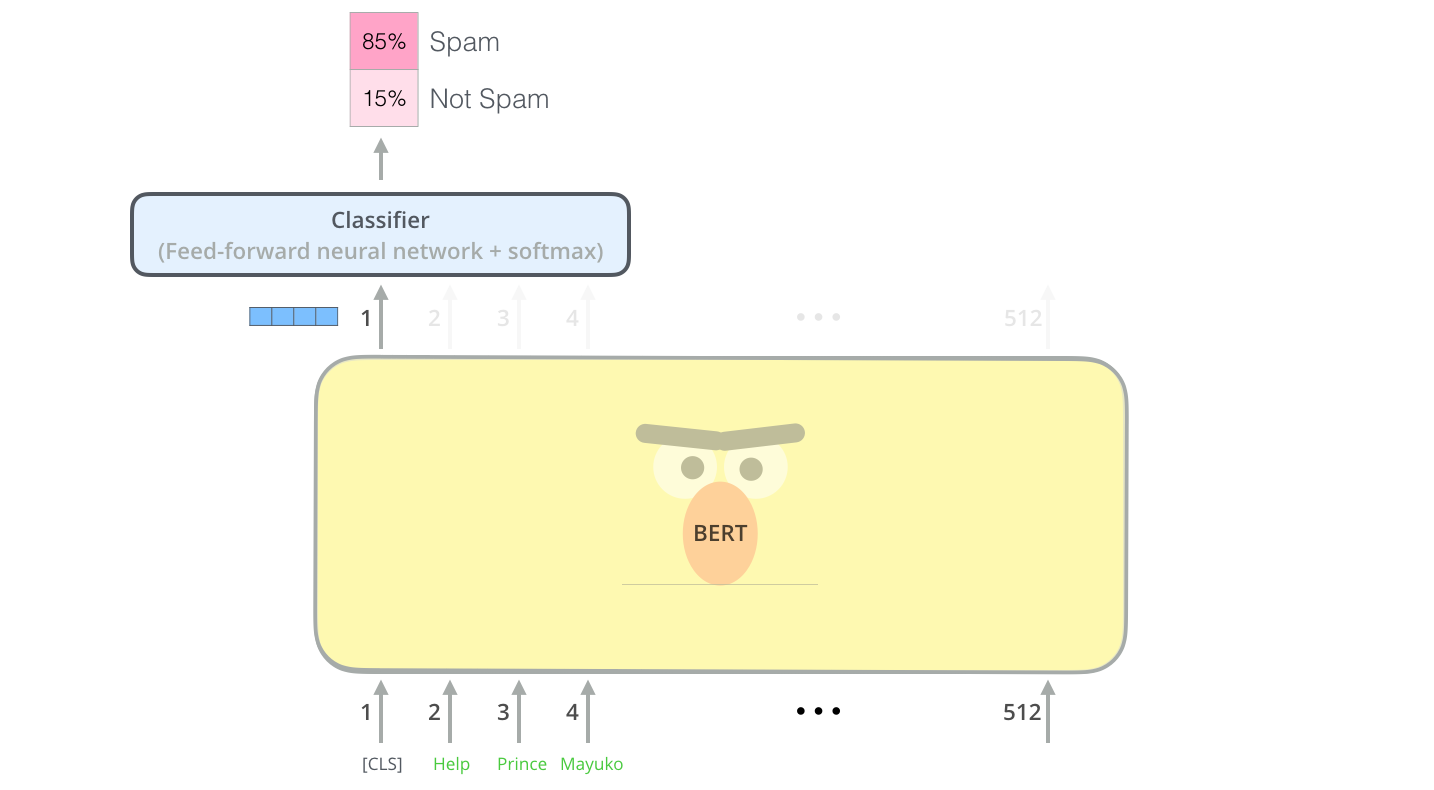
In terms of architecture, this has been identical to the Transformer up until this point (aside from size, which are just configurations we can set). It is at the output that we first start seeing how things diverge.

**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).



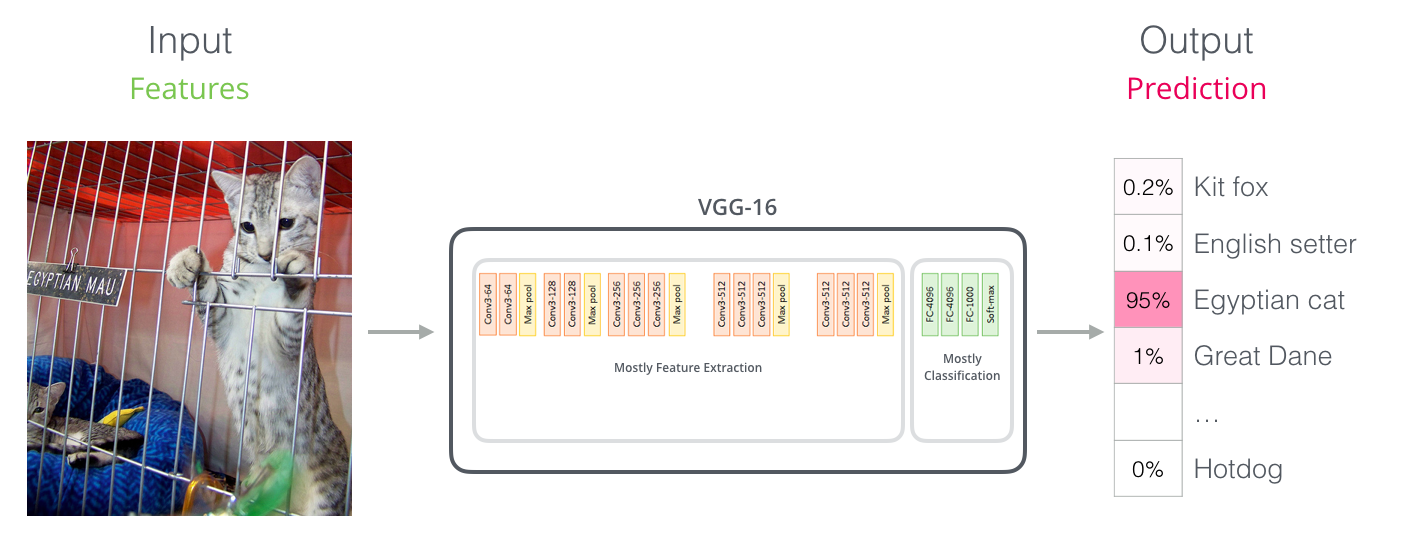
That vector can now be used as the input for a classifier of our choosing. The paper achieves great results by just using a single-layer neural network as the classifier.



If you have more labels (for example if you’re an email service that tags emails with “spam”, “not spam”, “social”, and “promotion”), you just tweak the classifier network to have more output neurons that then pass through softmax.

**Parallels with Convolutional Nets**

For those with a background in computer vision, this vector hand-off should be reminiscent of what happens between the convolution part of a network like VGGNet and the fully-connected classification portion at the end of the network.



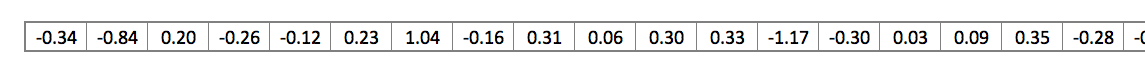
**A New Age of Embedding**

These new developments 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.

**Word Embedding Recap**

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 *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 syntactic, or grammar-based, relationships (e.g. the relationship between “had” and “has” is the same as that between “was” and “is”).

The field quickly realized it’s a 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 it became possible to 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)

  
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 these are large and full of numbers, I use the following basic shape in the figures in my posts to show vectors:

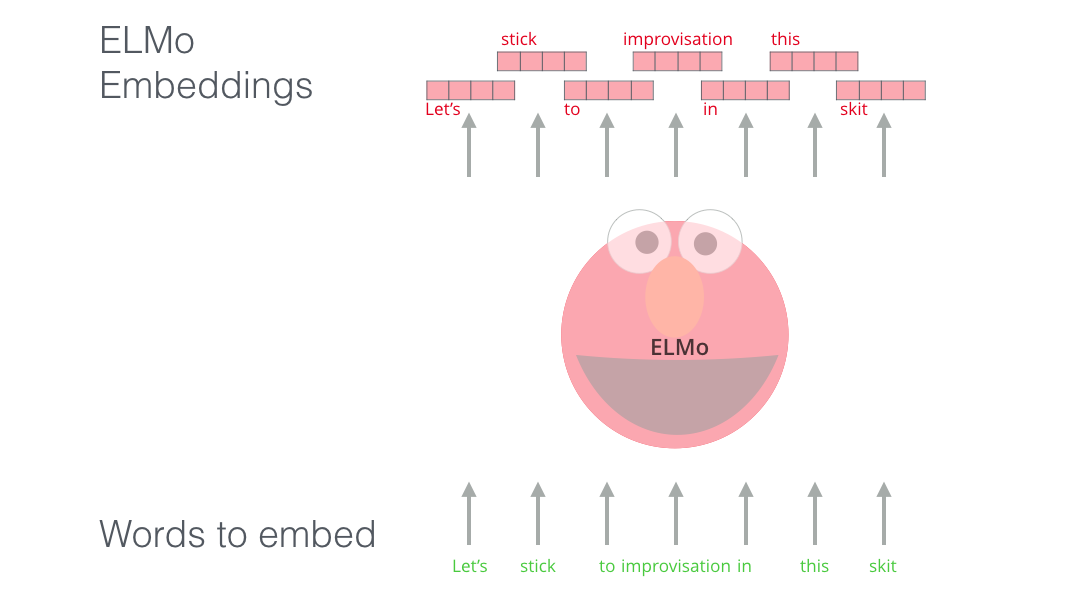
http://jalammar.github.io/images/vector-boxes.png

**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](https://arxiv.org/abs/1705.00108), [McCann et. al., 2017](https://arxiv.org/abs/1708.00107), and yet again [Peters et. al., 2018 in the ELMo paper](https://arxiv.org/pdf/1802.05365.pdf) ), “*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 word-embeddings can give words different embeddings based on the meaning they carry in the context of the sentence. Also, [RIP Robin Williams](https://www.youtube.com/watch?v=OwwdgsN9wF8)

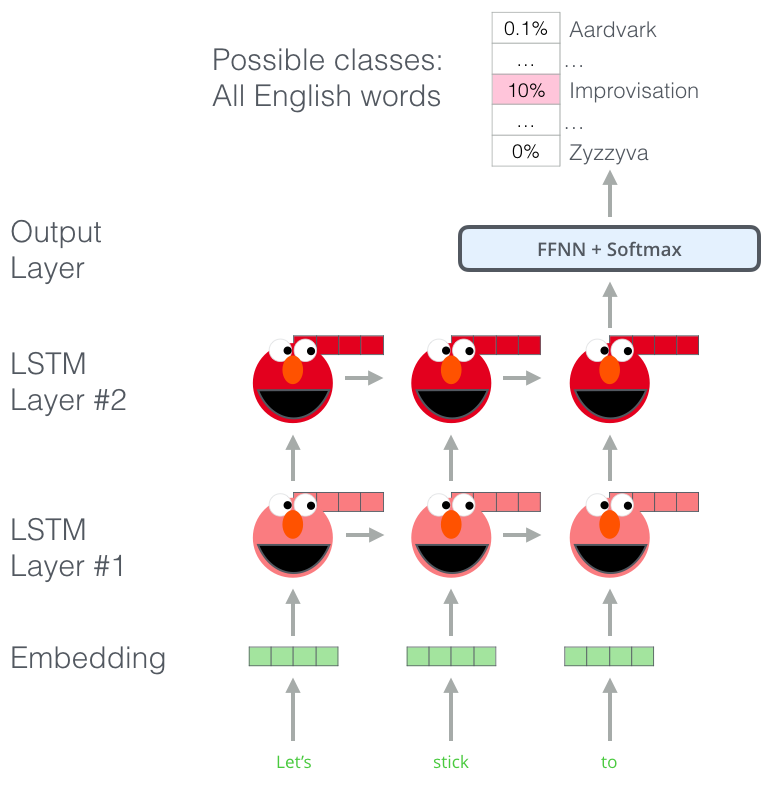
Instead of using a fixed embedding for each word, ELMo looks at the entire sentence before assigning each word in it an embedding. It uses a bi-directional LSTM trained on a specific task to be able to create those embeddings.



ELMo provided a significant step towards pre-training in the context of NLP. The ELMo LSTM would be trained on a massive dataset in the language of our dataset, and then we can use it as a component in other models that need to handle language.

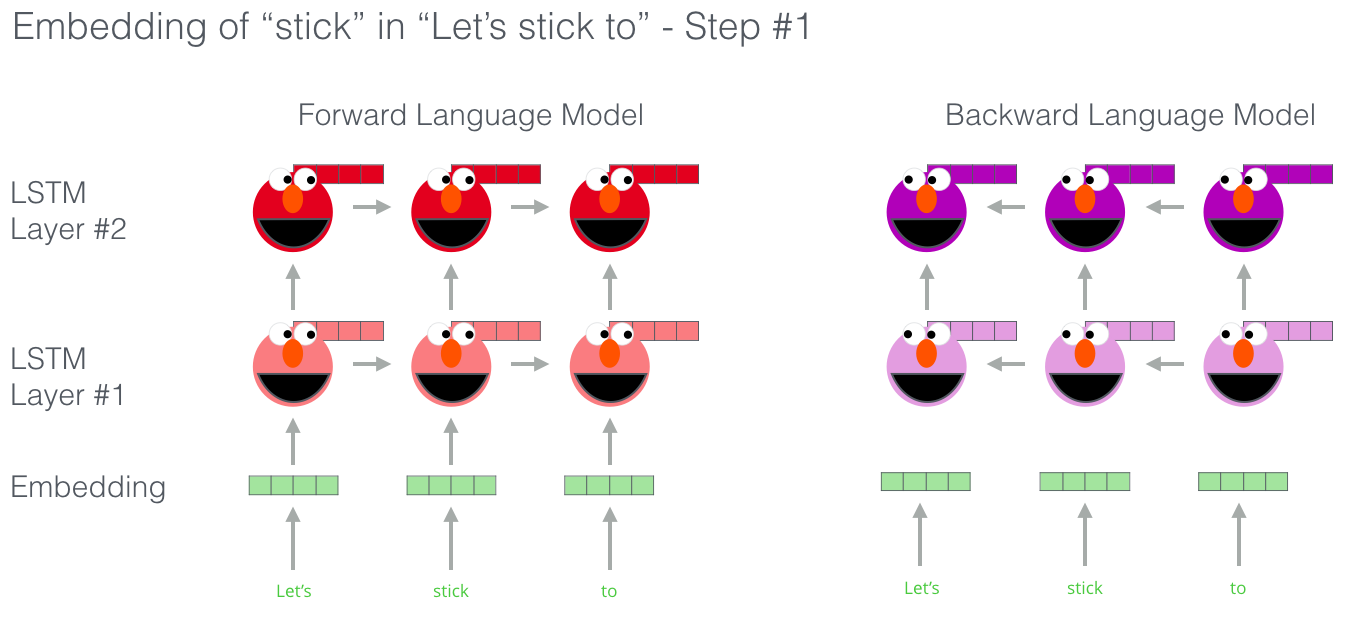
What’s ELMo’s secret?

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.

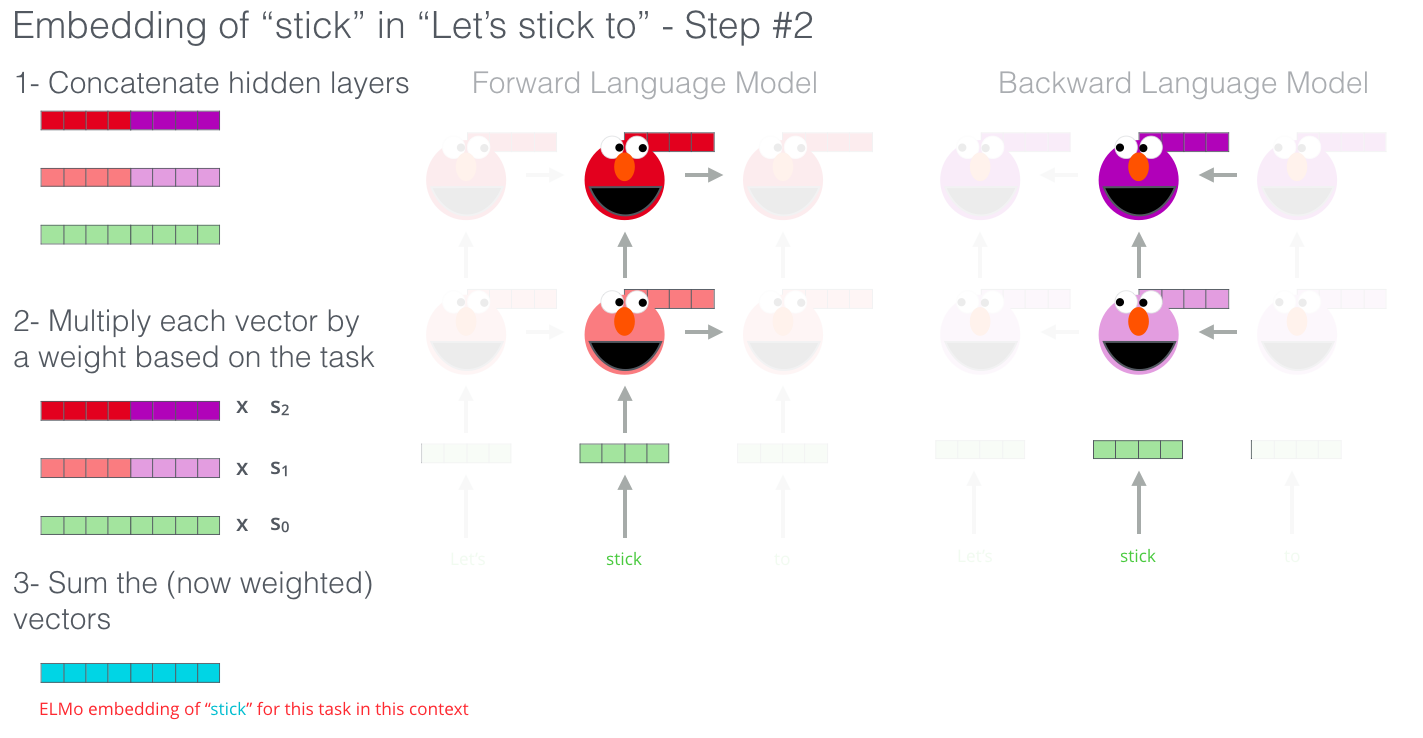
  
A step in the pre-training process of ELMo: Given “Let’s stick to” as input, predict the next most likely word – a *language modeling* task. When trained on a large dataset, the model starts to pick up on language patterns. It’s unlikely it’ll accurately guess the next word in this example. More realistically, after a word such as “hang”, it will assign a higher probability to a word like “out” (to spell “hang out”) than to “camera”.

We can see the hidden state of each unrolled-LSTM step peaking out from behind ELMo’s head. Those come in handy in the embedding proecss after this pre-training is done.

ELMo actually goes a step further and trains a bi-directional LSTM – so that its language model doesn’t only have a sense of the next word, but also the previous word.

  
[Great slides](https://www.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018) on ELMo

ELMo comes up with the contextualized embedding through grouping together the hidden states (and initial embedding) in a certain way (concatenation followed by weighted summation).



**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.

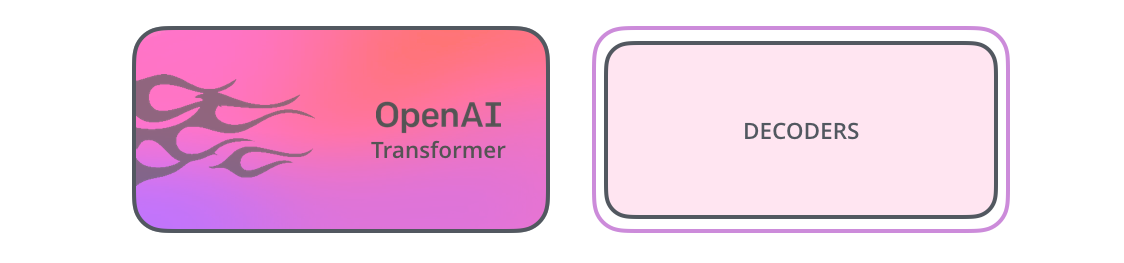
**The Transformer: Going beyond LSTMs**

The release of the Transformer paper and code, and the results it achieved on tasks such as machine translation started to make some in the field think of them as a replacement to LSTMs. This was compounded by the fact that Transformers deal with long-term dependancies better than LSTMs.

The Encoder-Decoder structure of the transformer made it perfect for machine translation. But how would you use it 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).

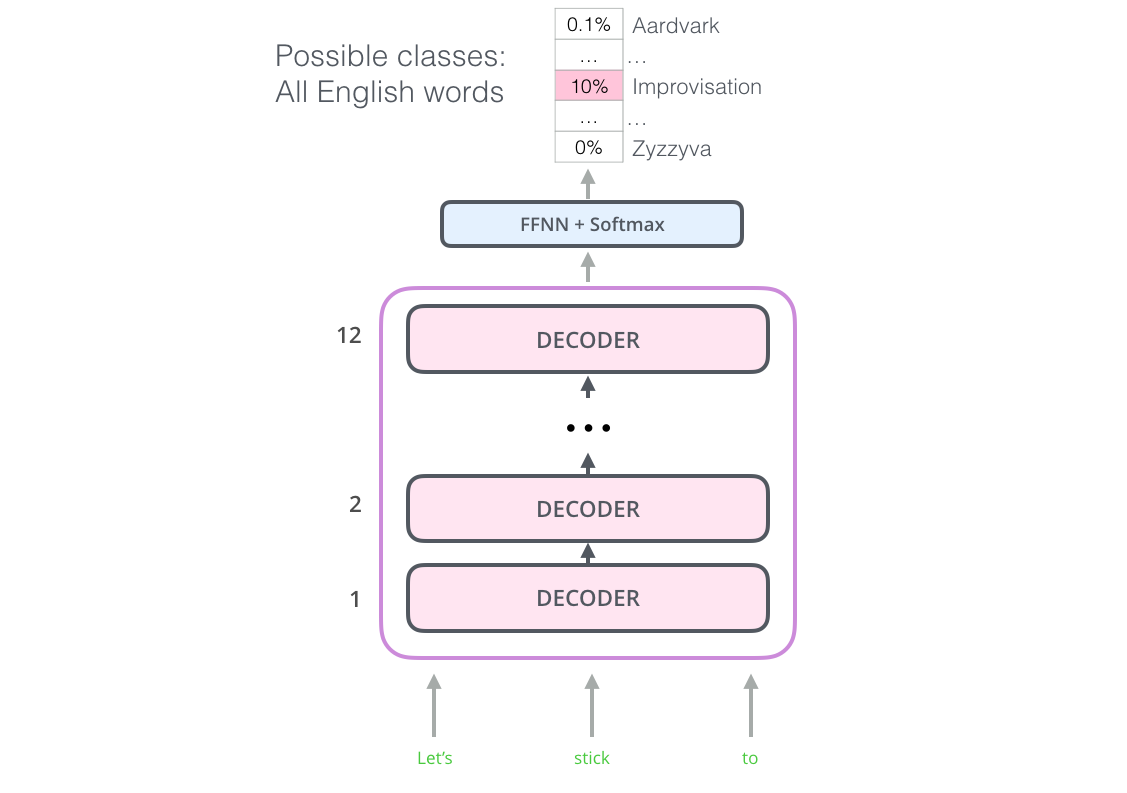
**OpenAI Transformer: Pre-training a Transformer Decoder for Language Modeling**

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.

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

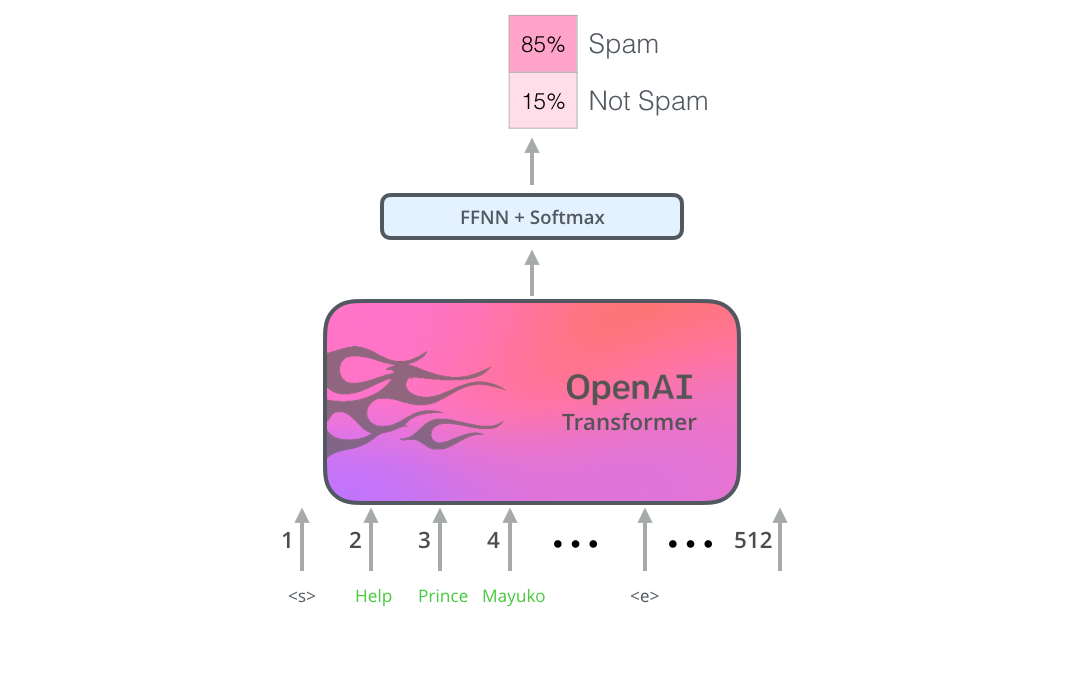
The model stacked twelve decoder layers. Since there is no encoder in this set up, these decoder layers would not have the encoder-decoder attention sublayer that vanilla transformer decoder layers have. It would still have the self-attention layer, however (masked so it doesn’t peak at future tokens).

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.

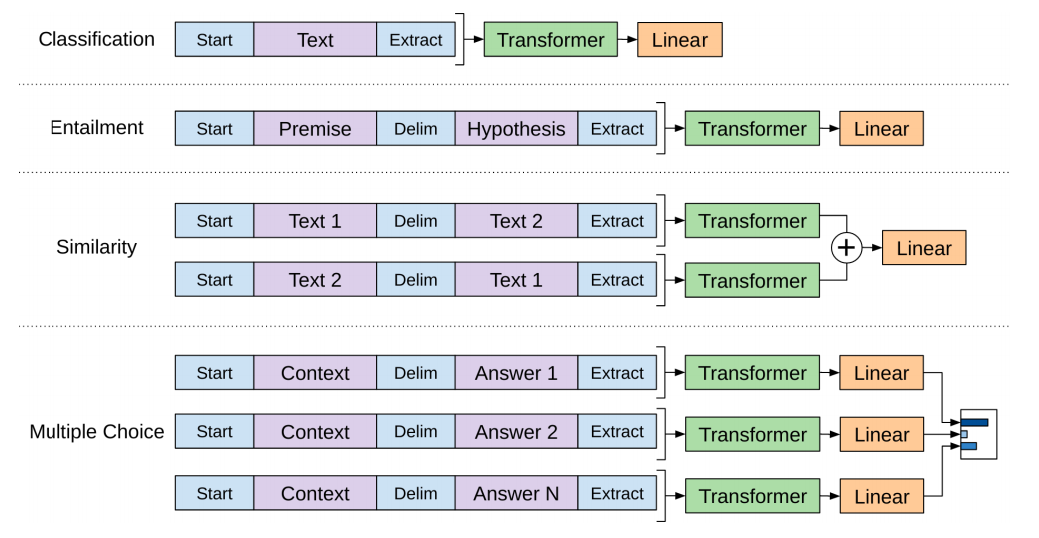
  
The OpenAI Transformer is now ready to be trained to predict the next word on a dataset made up of 7,000 books.

**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 clasification

The OpenAI paper outlines a number of input transformations to handle the inputs for different types of tasks. The following image from the paper shows the structures of the models and input transformations to carry out different tasks.



Isn’t that clever?

**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”)?

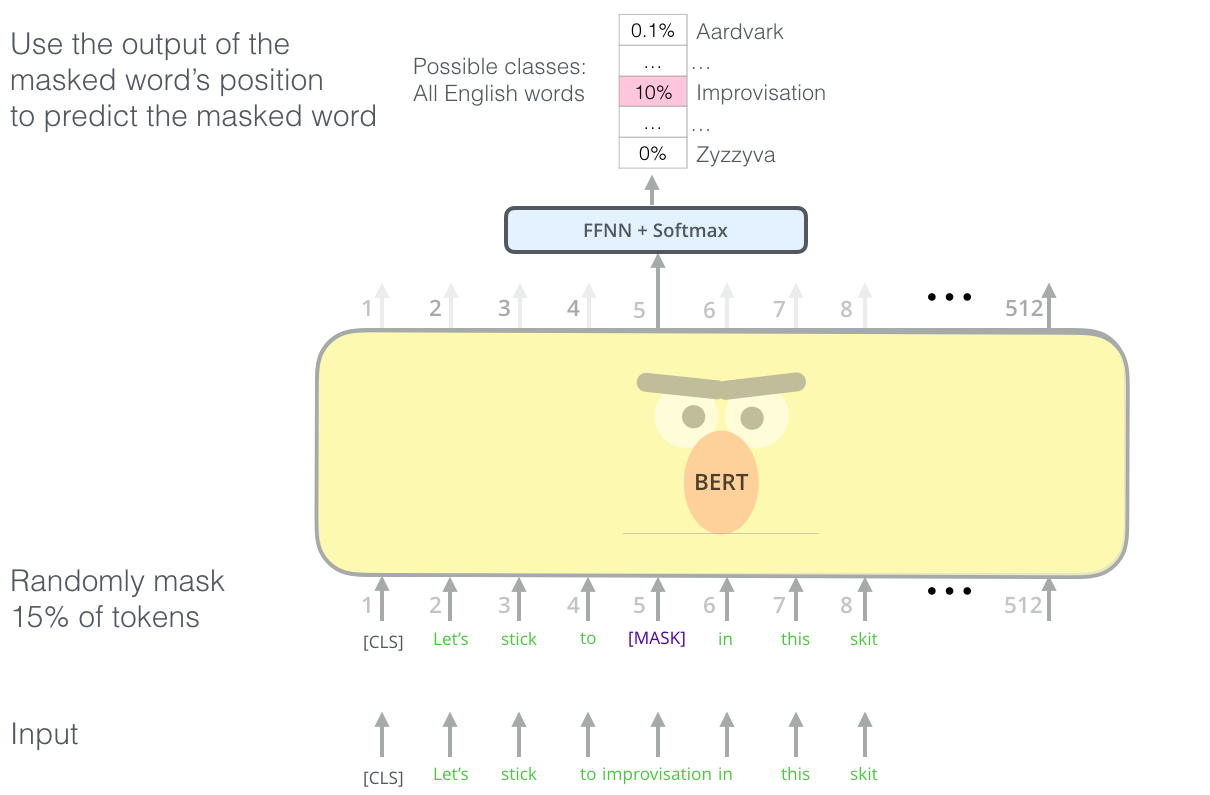
“Hold my beer”, said R-rated BERT.

**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.

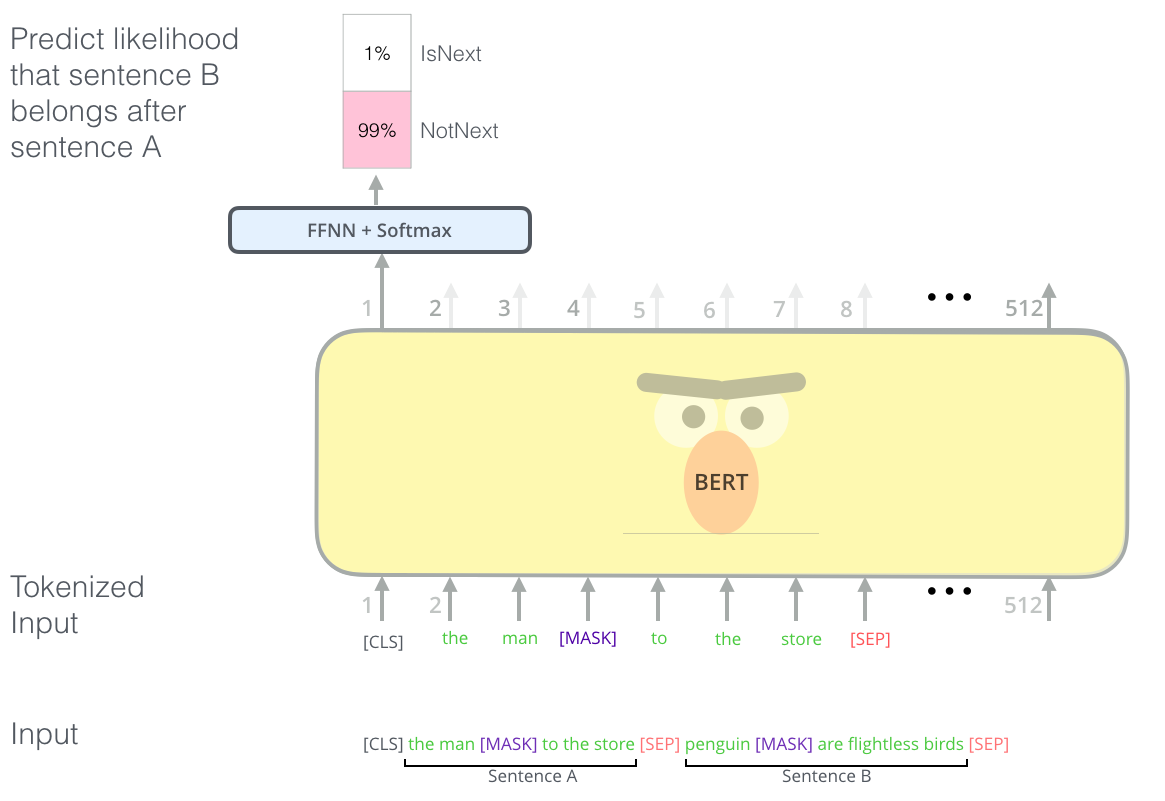
Finding the right task to train a Transformer stack of encoders is a complex hurdle that BERT resolves by adopting a “masked language model” concept from earlier literature (where it’s called a Cloze task).

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**

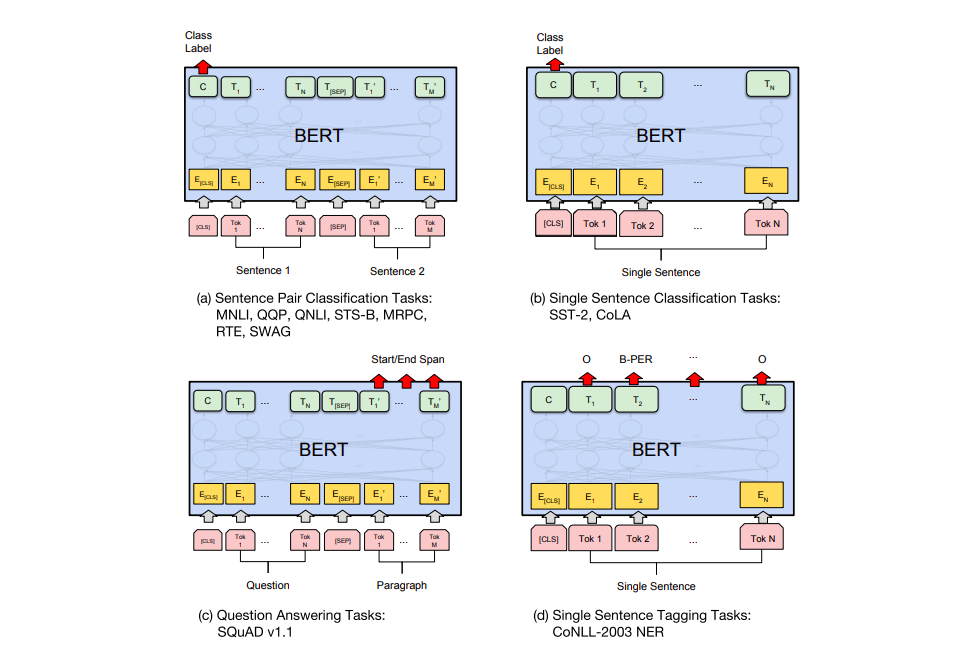
If you look back up at the input transformations the OpenAI transformer does to handle different tasks, you’ll notice that some tasks require the model to say something intelligent about two sentences (e.g. are they simply paraphrased versions of each other? Given a wikipedia entry as input, and a question regarding that entry as another input, can we answer that question?).

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?

  
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.

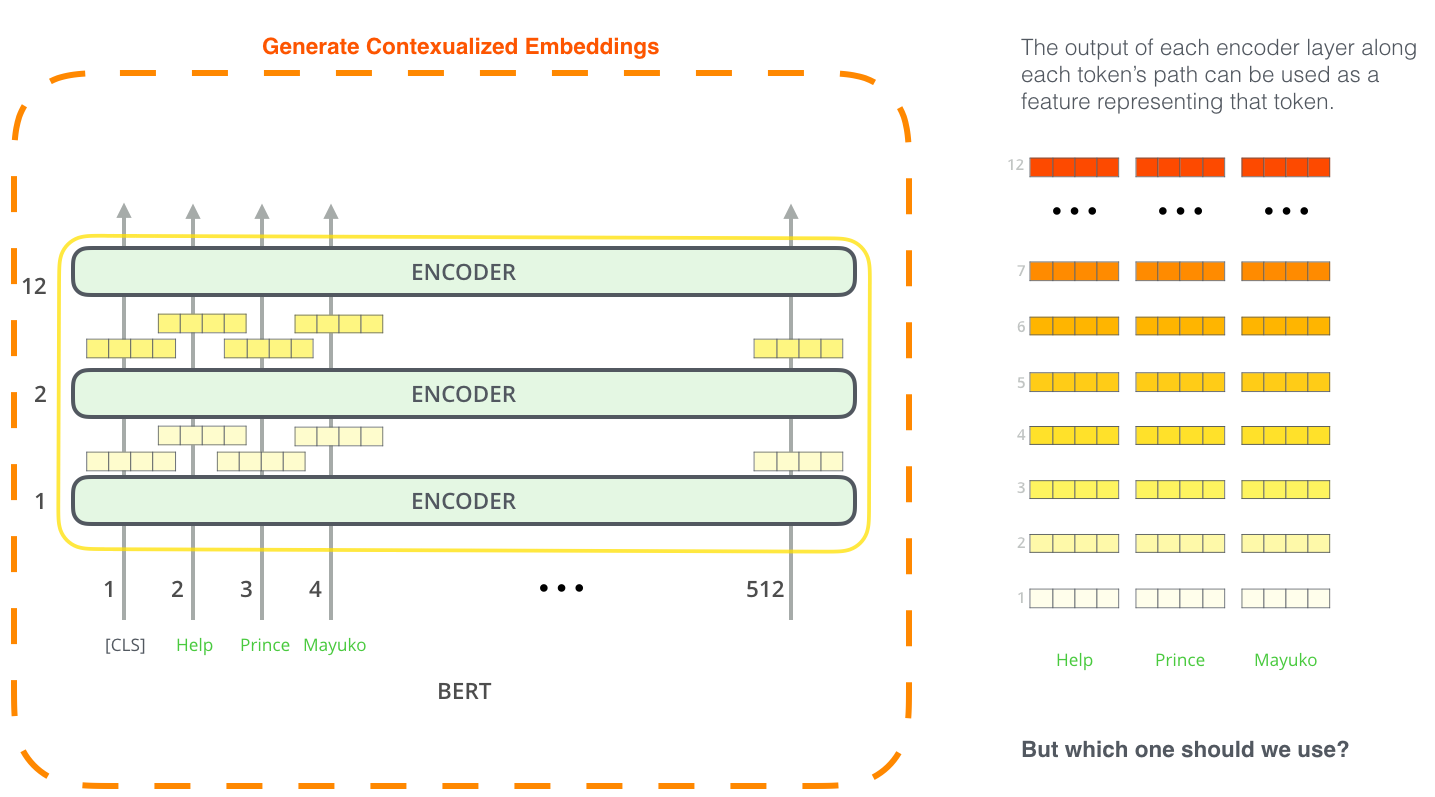
**Task specific-Models**

The BERT paper shows a number of ways to use BERT for different tasks.

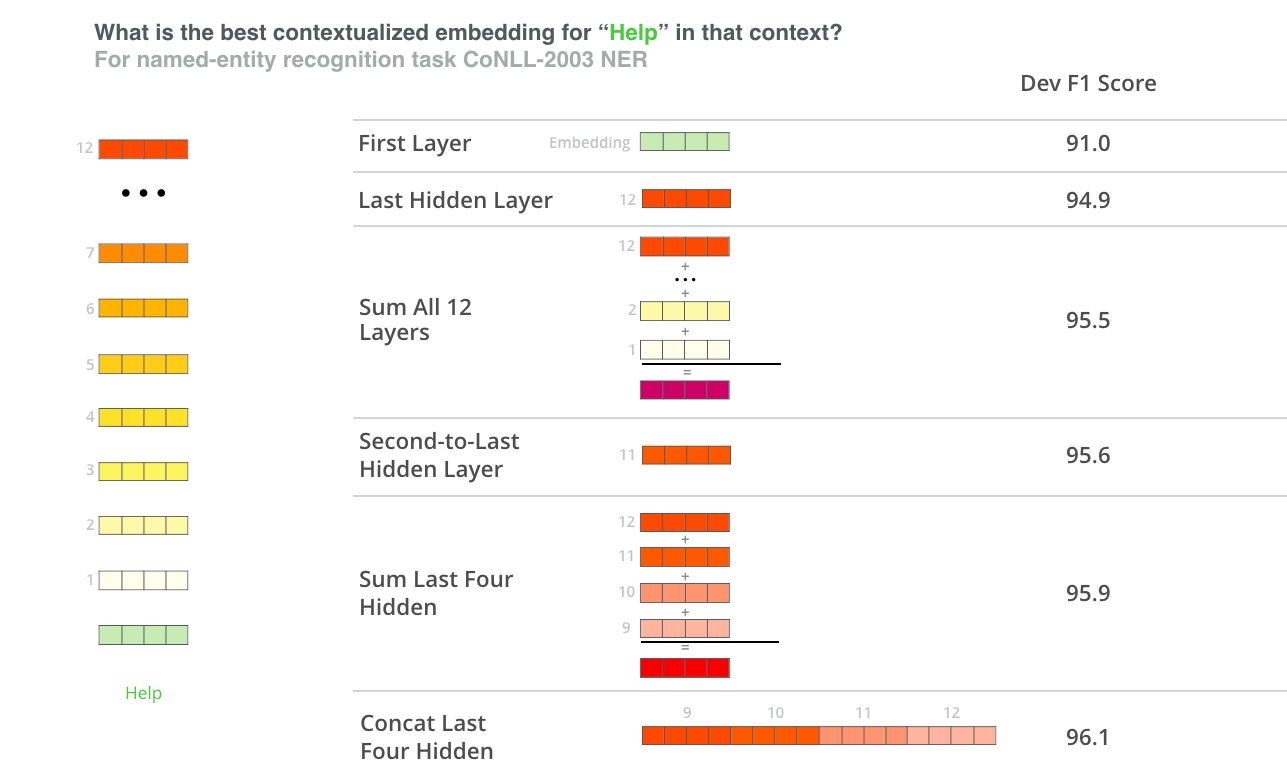


**BERT for feature extraction**

The fine-tuning approach isn’t the only way to use BERT. Just like ELMo, you can use the pre-trained BERT to create contextualized word embeddings. Then you can feed these embeddings to your existing model – a process the paper shows yield results not far behind fine-tuning BERT on a task such as named-entity recognition.



Which vector works best as a contextualized embedding? I would think it depends on the task. The paper examines six choices (Compared to the fine-tuned model which achieved a score of 96.4):



**Take BERT out for a spin**

The best way to try out BERT is through the [BERT FineTuning with Cloud TPUs](https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/bert_finetuning_with_cloud_tpus.ipynb) notebook hosted on Google Colab. If you’ve never used Cloud TPUs before, this is also a good starting point to try them as well as the BERT code works on TPUs, CPUs and GPUs as well.

The next step would be to look at the code in the [BERT repo](https://github.com/google-research/bert):

* The model is constructed in [modeling.py](https://github.com/google-research/bert/blob/master/modeling.py) (class BertModel) and is pretty much identical to a vanilla Transformer encoder.
* [run\_classifier.py](https://github.com/google-research/bert/blob/master/run_classifier.py) is an example of the fine-tuning process. It also constructs the classification layer for the supervised model. If you want to construct your own classifier, check out the create\_model() method in that file.
* Several pre-trained models are available for download. These span BERT Base and BERT Large, as well as languages such as English, Chinese, and a multi-lingual model covering 102 languages trained on wikipedia.
* BERT doesn’t look at words as tokens. Rather, it looks at WordPieces. [tokenization.py](https://github.com/google-research/bert/blob/master/tokenization.py) is the tokenizer that would turns your words into wordPieces appropriate for BERT.

You can also check out the [PyTorch implementation of BERT](https://github.com/huggingface/pytorch-pretrained-BERT). The [AllenNLP](https://github.com/allenai/allennlp) library uses this implementation to [allow using BERT embeddings](https://github.com/allenai/allennlp/pull/2067) with any model.

**Acknowledgements**

Thanks to [Jacob Devlin](https://github.com/jacobdevlin-google), [Matt Gardner](https://twitter.com/nlpmattg), [Kenton Lee](https://github.com/kentonl), [Mark Neumann](https://twitter.com/markneumannnn), and [Matthew Peters](https://twitter.com/mattthemathman) for providing feedback on earlier drafts of this post.

*Written on December 3, 2018*

Top of Form

**Subscribe to get notified about upcoming posts by email**

Bottom of Form