Language Understanding

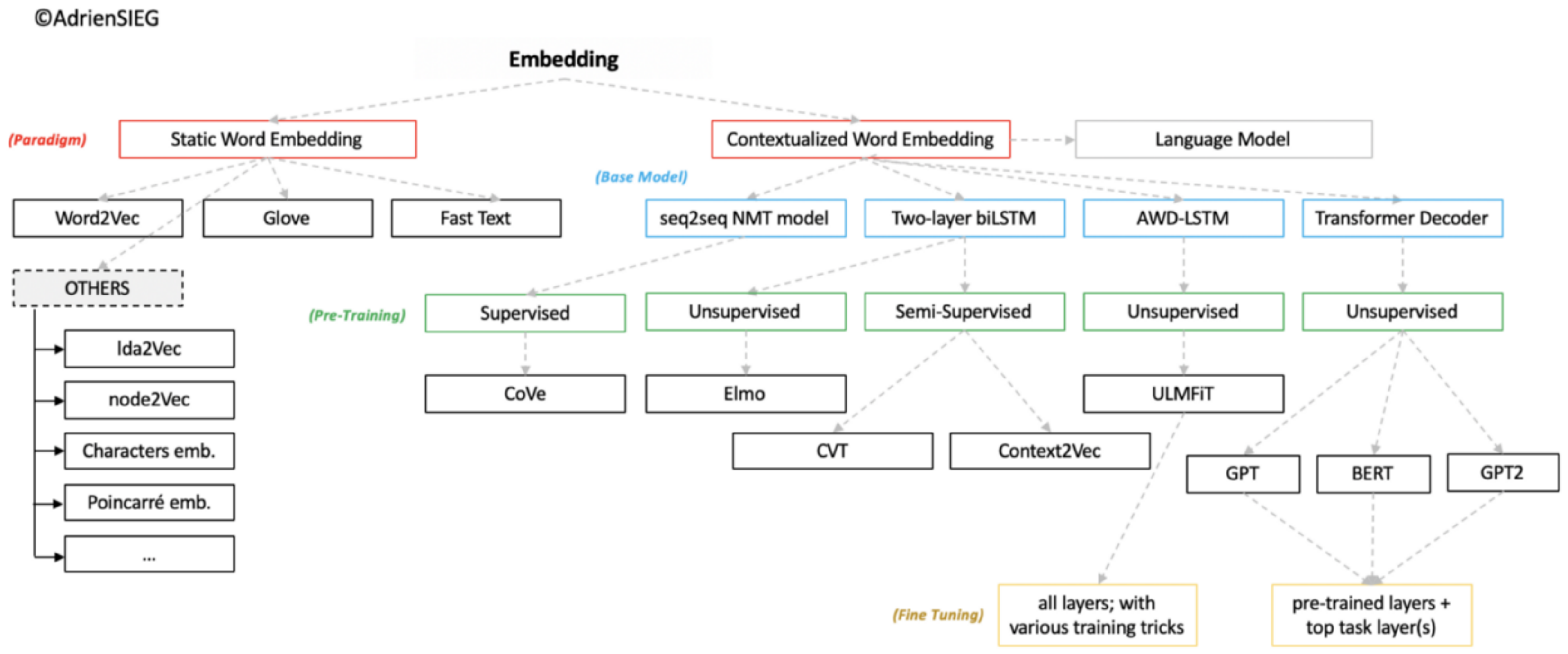
07 - Word Representations and Morphology

Hossein Zeinali

Introduction

- We saw how FFNNs and RNNs can be used for language modeling.
- Each hidden layer of a neural language model is a candidate for representing its input in a new space.
  - So, multiplying the one-hot input vector of word i, w_i, by the weight matrix C, selects a row of C (a vector).
  - We call this vector the word embedding of w_i.
- In most NLP tasks, learned word embeddings have replace hand-engineered word features.

Overview of Word Embeddings



https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598

Static Word Embeddings

- They assign a fixed vector to each word, independent of its context.
- They are used for feature extraction, i.e., to initialize the embedding layer of a target model.
- They are not designed to be used for fine-tuning.
- They are often efficient enough so that training from scratch is possible.
- Word2Vec, Glove, FastText are typical examples.
- Embeddings are available to download for many languages.

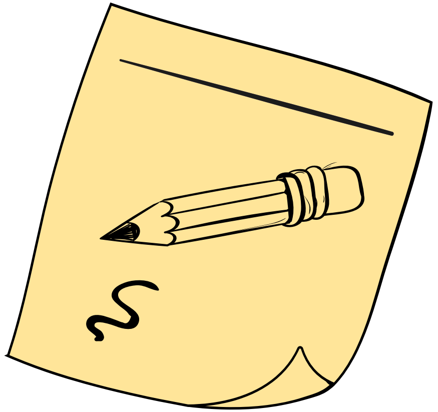
- Here we will introduce Word2Vec as an example of static word embeddings.

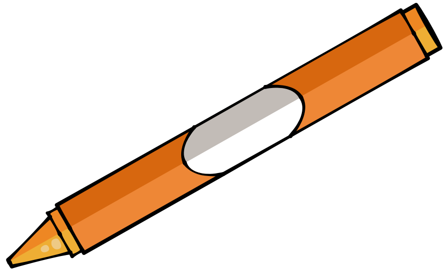
Contextualized (Dynamic) Word Embeddings

- They assign a vector to a word that depends on its context, i.e., on preceding and following words.
- Can be used in a target model just like static embeddings.
- However, they can also be fine-tuned: we re-train some of the weights of the embedding model for the target task.
- Contextualized embeddings take a lot of memory and compute to train from scratch, so fine-tuning is their dominant use.
- **Elmo, Bert, Ulmfit, GPT** are the most prominent examples.
- Pre-trained versions can be downloaded for multiple languages.

- We will see Elmo, Bert and GPTs as examples of contextualized word embeddings.

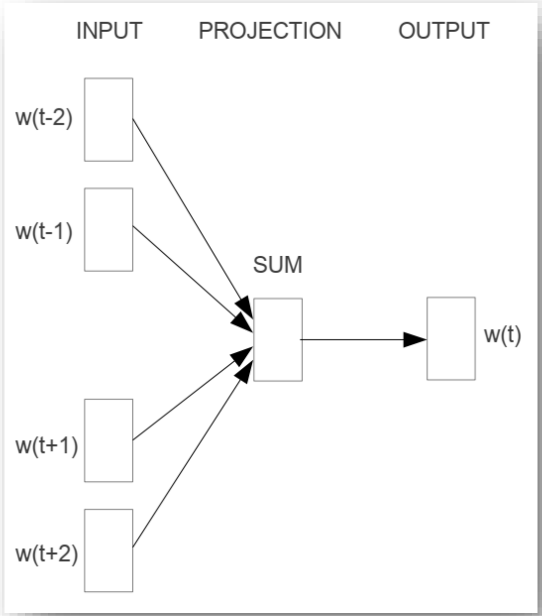
Static Word Embeddings



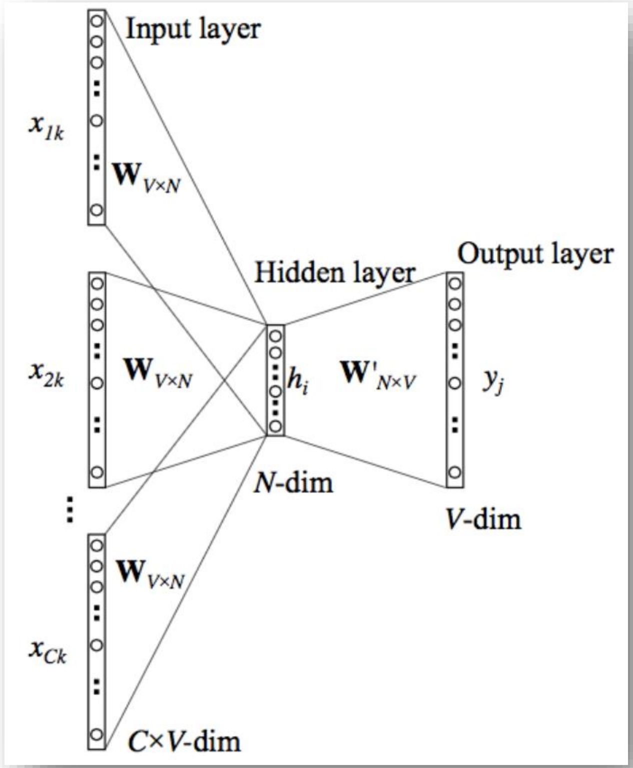


Word2Vec: CBOW

- Continuous Bag-of-Words Model (CBOW) uses the words within a small window to predict the current word.
- Has only a single, linear hidden layer.
- The weights for different positions are shared.
- Window size is five in the original paper (two context words each to the left and right of the target words).
- Computationally very efficient.



Word2Vec: CBOW



- How to generate training examples?

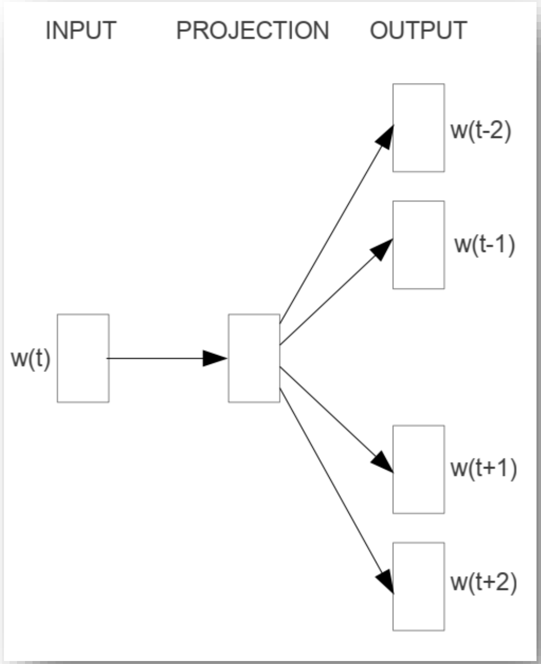
Jay was hit by a ______ bus in...

by	a	red	bus	in

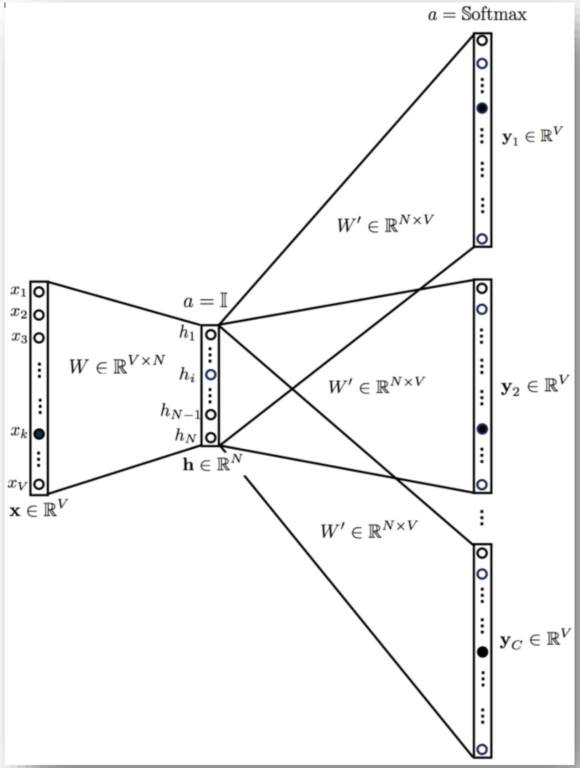
input 1	input 2	input 3	input 4	output
by	a	bus	in	red

Word2Vec: Skipgram

- Skipgram turns CBOW on its head: uses the current word to predict the context words.
- This finds word representations that are useful for predicting surrounding words in a sentence or document.
- Same overall architecture as CBOW but has better performance.
  - In most cases this model is used for word2vec.



Word2Vec: Skipgram



$a = \text{Softmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
x_1
x_2
x_3
x_k
x_V
$W \in \mathbb{R}^{V \times N}$
$a = \mathbb{I}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$\mathbf{x} \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$

- How to generate training examples?

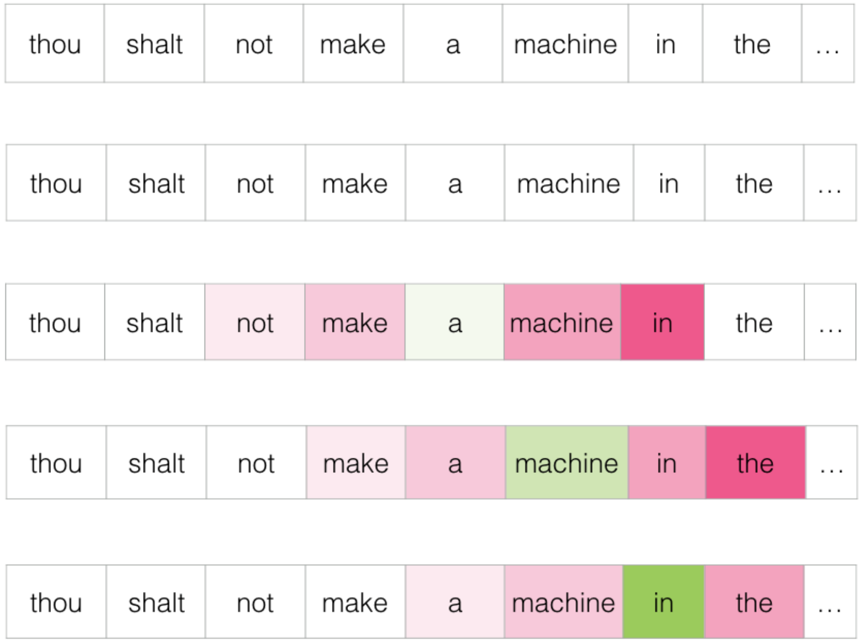
Jay was hit by a red bus in...

by	a	red	bus	in

input	output
red	by
red	a
red	bus
red	in

Word2Vec: Skipgram

Thou shalt not make a machine in the likeness of a human mind



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

Word2Vec: Negative Sampling

- Given a sequence of training words $w_1, w_2, w_3, \dots, w_T$, the objective of the Skip-gram model is to maximize the average log probability:

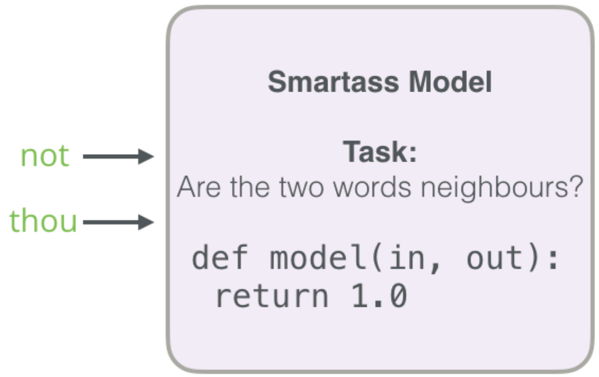
$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0}\log p(w_{t+j}|w_t)$$

where c is the size of the training context and

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)}$$

- This formulation is impractical because its computation cost is proportional to W , which is often large.
- **Negative Sampling**: instead of computing the full output layer based on the hidden layer, evaluate only the output neuron that represents the positive class and a few randomly sampled neurons.

Word2Vec: Negative Sampling



```
def model(in, out):
  return 1.0
```

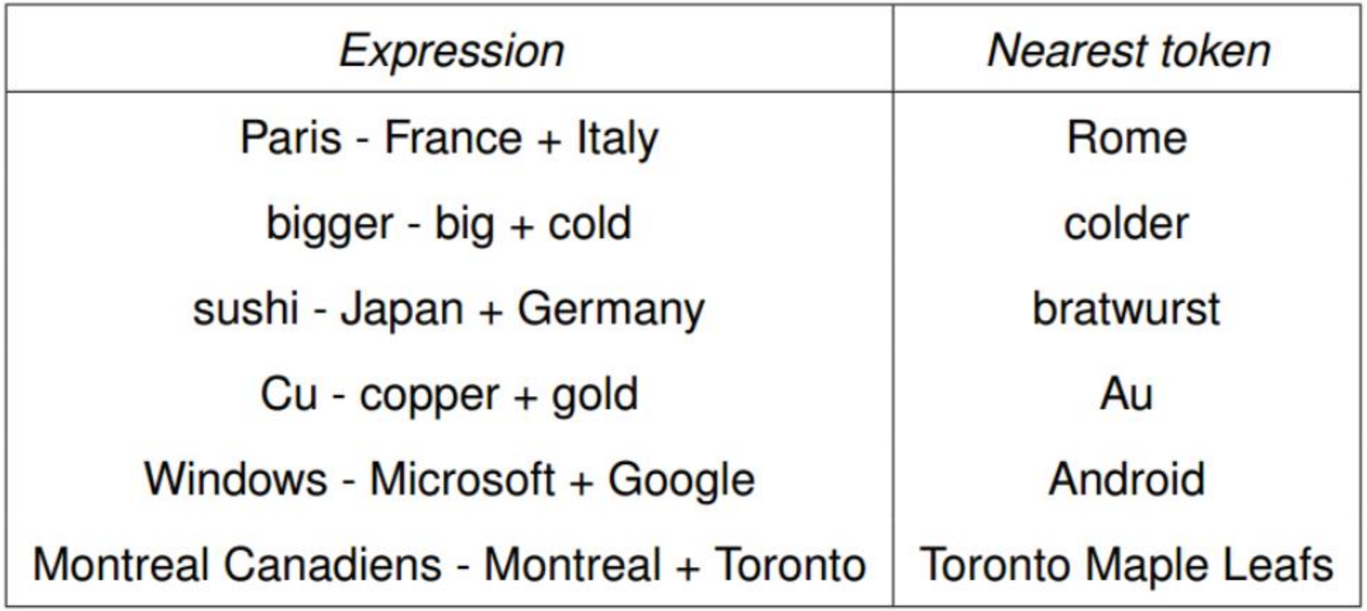
input word	output word	target
not	thou	1
not		0
not		0
not	shalt	1
not	make	1

Negative examples

See the following page for a full word2vec illustration:
https://jalammar.github.io/illustrated-word2vec/

Skipgram Model Evaluation

- Subtracting two word vectors, and add the result to another word.



Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Open Vocabulary Modeling

Language Modeling Assumption

- To have a valid probability distribution, the following rule should be met (V is a finite vocabulary):

$$\sum_{w \in V} p(w|w_{i-n+1}, \dots, w_{i-1}) = 1$$

where

$$p(w|w_{i-n+1}, \dots, w_{i-1}) = \frac{C(w_{i-n+1} \dots w_{i-1}w)}{C(w_{i-n+1} \dots w_{i-1})}$$

- When does this assumption make sense for language modeling?
- What about word2vec?

Words Are Not a Finite Set!

- Most languages do not have a finite set of words
  - New incoming words and its compositions
  - Generative languages like Persian

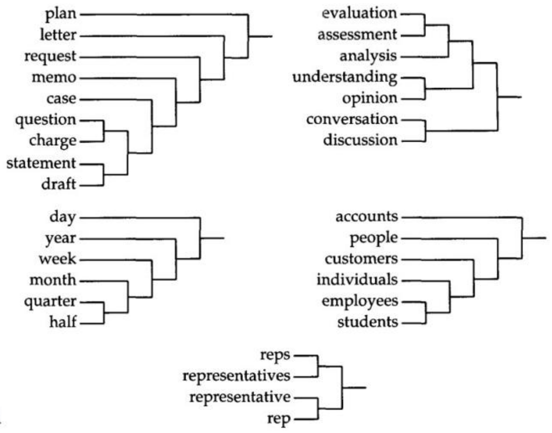
- Some trivial solutions:
  - Bengio et al.: “Rare words with frequency ≤ 3 were merged into a single symbol, reducing the vocabulary size considerably”
  - Bahdanau et al.: “Only use a shortlist of 30,000 most frequent words in each language. Any word not included in the shortlist is mapped to a special token ([UNK]).”

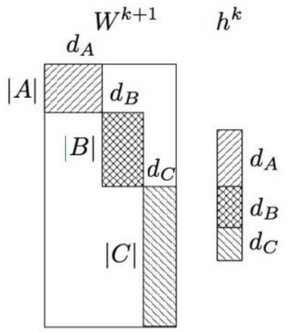
Scale by Partitioning

- Class-based LM: Partition the vocabulary into smaller pieces:

$$p(w_i|h_i) = p(c_i|h_i)p(w_i|c_i, h_i)$$

- Hierarchical softmax: partition the vocabulary into smaller pieces hierarchically
- Differentiated softmax: assign more parameters to more frequent words, fewer to less frequent words.
- And some other methods





Source: Strategies for Training Large Vocabulary Neural Language Models

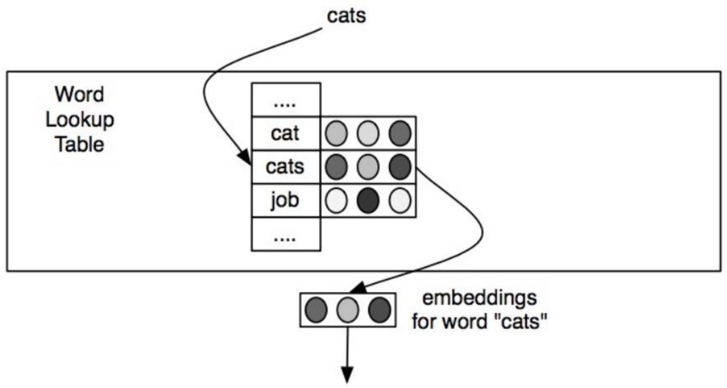
Vocabulary and Possible Finite Sets

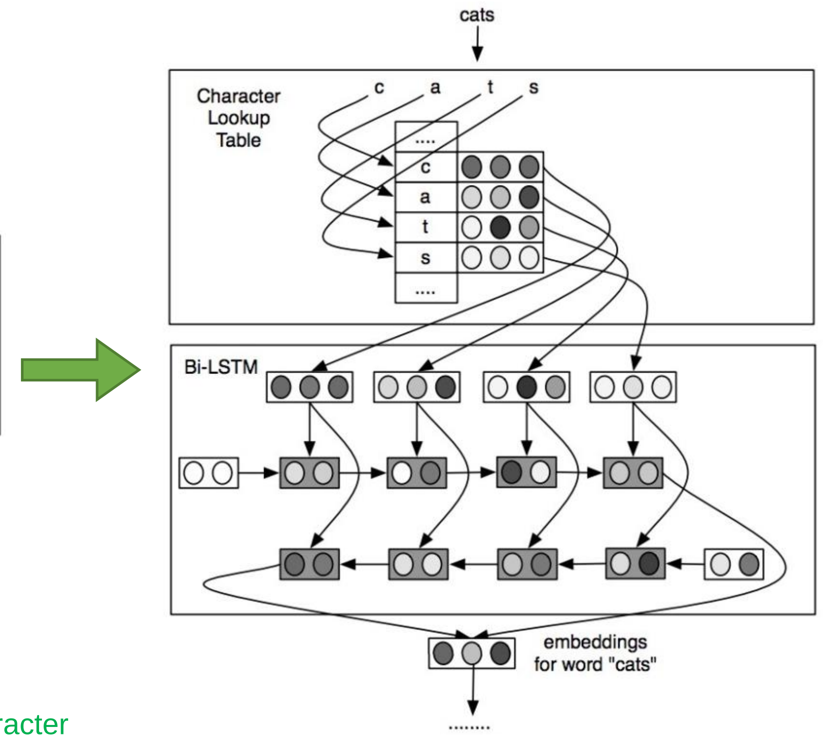
- The vocabulary of word types is infinite
  - Practical problem: softmax computation is linear in vocabulary size.
  - Cannot used trained models for new words

- What set is finite?
  - Characters, or unicodes in multilingual scenario
  - Char-trigrams
  - Word pieces
  - Syllable (difficult to obtain)
  - Morphemes (stems)
  - Etc.

Source: Strategies for Training Large Vocabulary Neural Language Models

Compositional Character Model

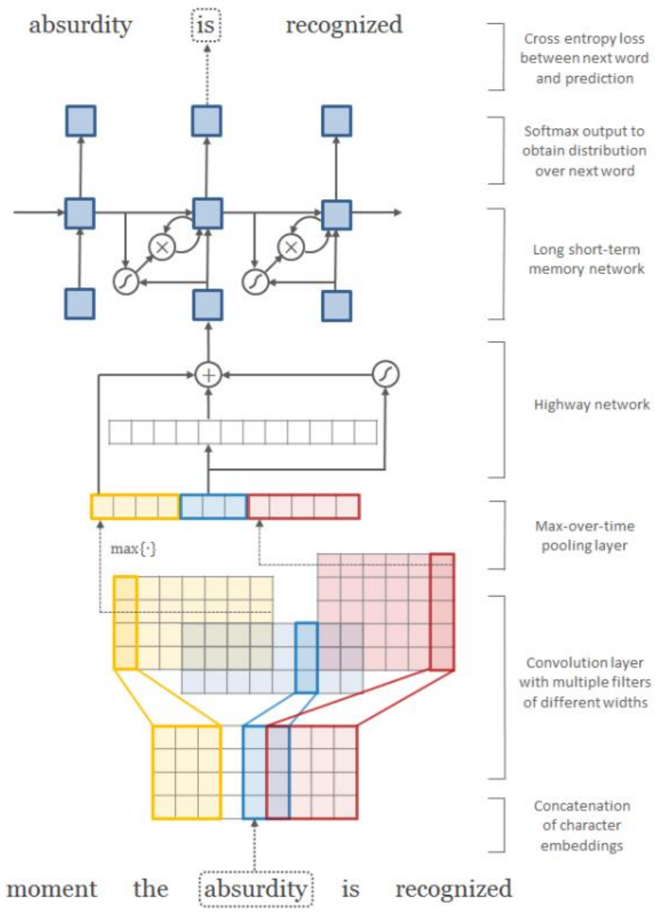




Source: Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation

Character-Aware NLM

- The network is a composition of a normal RNN based LM, CNN for input processing and a highway network as depicted in the figure.



absurdity
is
recognized
Cross entropy loss between next word and prediction
Softmax output to obtain distribution over next word
Long short-term memory network
Highway network
Max-over-time pooling layer
max{·}
Convolution layer with multiple filters of different widths
Concatenation of character embeddings
moment
the
absurdity
is
recognized

Source: Character-Aware Neural Language Models

Subword Tokenization

- The character level representations work fine
  - It is very simple and would save a lot of memory
  - But, does not allow the model to learn representations of texts as meaningful as when using words.
- Can we do a better job?
- Subword tokenization:
  - Relies on the principle that most common words should be left as is, but rare words should be decomposed in meaningful subword units.
    - This is especially useful in agglutinative languages such as Turkish or morphologically rich languages such as Persian and Arabic

Byte-Pair Encoding (BPE)

- Word embedding sometimes is too high level, pure character embedding too low level. For example, if we have learned
  old older oldest
  We might also wish the computer to infer
  smart smarter smartest

- But at the whole word level, this might not be so direct. Thus the BPE idea is to break the words up into pieces like er, est, and embed frequent fragments of words.

Byte-Pair Encoding (BPE)

- Byte Pair Encoding is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte.
- Algorithm:
  - Initialize the symbol vocabulary with the character vocabulary, and represent each word as a sequence of characters, plus a special end-of word symbol ‘</w>’
  - Iteratively count all symbol pairs and replace each occurrence of the most frequent pair (‘A’, ‘B’) with a new symbol ‘AB’ (character n-grams)
  - Keep doing step 2, until it hits the pre-defined maximum number of sub-words or iterations.
  - The final symbol vocabulary size is equal to the size of the initial vocabulary, plus the number of merge operations.

Byte-Pair Encoding (BPE)

- The algorithm can be run on the dictionary extracted from a text, with each word being weighted by its frequency.
- Example (# merges = 10):

  {‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w e s t </w>’: 6, ‘w i d e s t </w>’: 3 }
  - {‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w es t </w>’: 6, ‘w i d es t </w>’: 3 }
  - {‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w est </w>’: 6, ‘w i d est </w>’: 3 }
  - {‘l o w </w>’: 5, ‘l o w e r </w>’: 2, ‘n e w est</w>’: 6, ‘w i d est</w>’: 3 }
  - {‘lo w </w>’: 5, ‘lo w e r </w>’: 2, ‘n e w est</w>’: 6, ‘w i d est</w>’: 3 }
  - {‘low </w>’: 5, ‘low e r </w>’: 2, ‘n e w est</w>’: 6, ‘w i d est</w>’: 3 }
  - …

  {‘low</w>’: 5, ‘low e r </w>’: 2, ‘newest</w>’: 6, ‘wi d est</w>’: 3 }

Byte-Pair Encoding (BPE)

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

WordPiece

- WordPiece is the subword tokenization algorithm used for BERT
- It relies on the same base as BPE
  - The difference is that it does not choose the pair that is the most frequent but the one that will maximize the likelihood on the corpus once merged.
- Algorithm:
  1. Initialize the word unit list with the basic characters
  2. Build a language model on the training data using list from 1
  3. Generate a new word unit by combining two units out of the current word list. Choose the new unit that increases the likelihood on the training data
  4. Go to 2 until a predefined limit of word units is reached

Contextual Word Embeddings

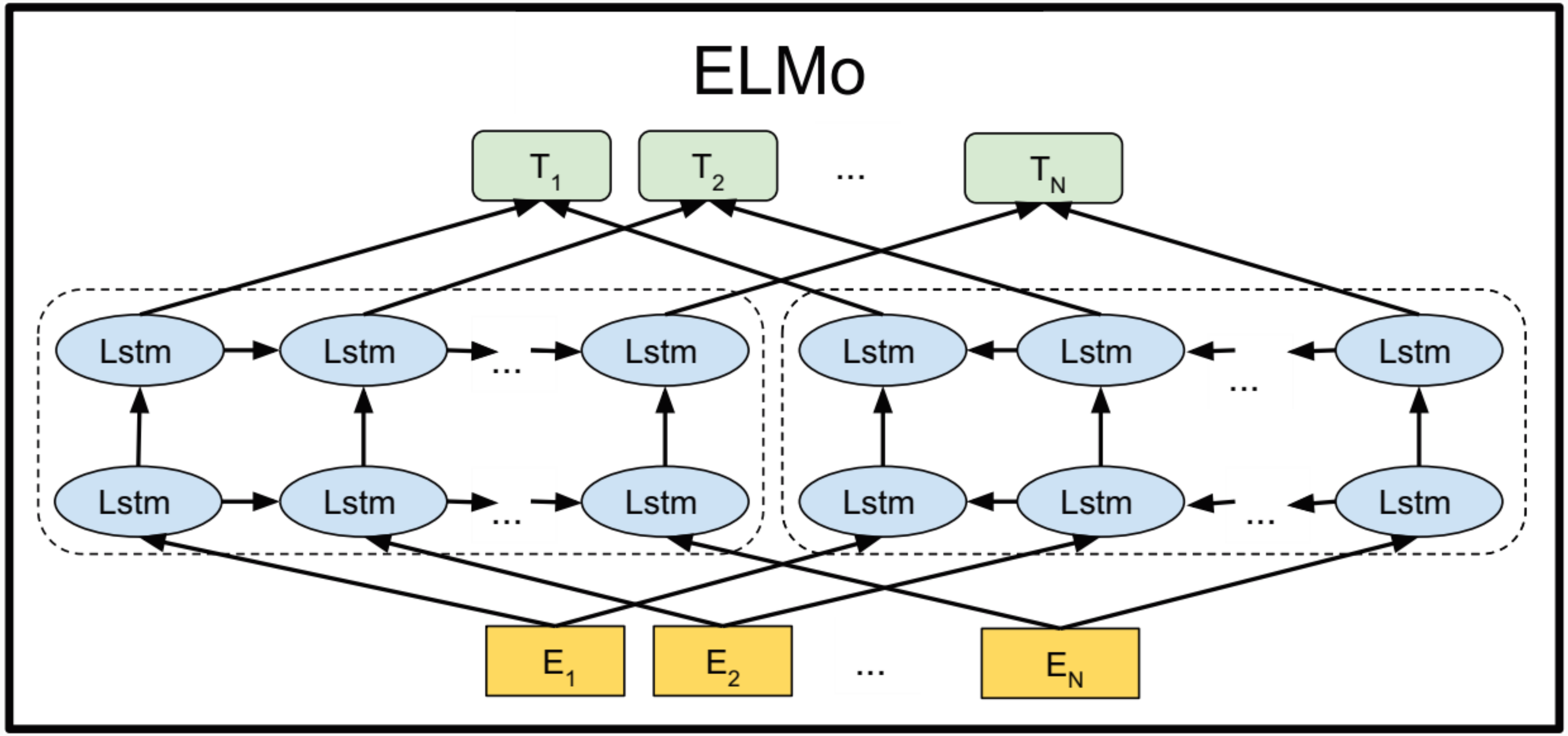
From Transformers to Embeddings

- In Transformer slides, we saw how we can stack transformer blocks and use them for sequence modeling.
  - We can apply mean pooling to the output vectors and map it onto a class vector to do classification.
- We can also use transformers for sequence prediction. For this we need to mask some of the input.
- So far in this lecture, we saw how we can derive embeddings from topologies like neural language models.
- Transformers are language model that can represent context very well: they can learn contextualized embeddings.
- Contextualized language models can be used for feature extraction, but also in a pre-training/fine-tuning setup.

ELMo

- **ELMo:** Embeddings from Language Models
- ELMo is a new type of deep contextualized word representations that model:
  - Complex characteristics of word use
    - (e.g., syntax and semantics)
  - How these uses vary across linguistic contexts (i.e., to model polysemy)
- ELMo can improve existing neural models in various NLP tasks
- ELMo can capture more abstract linguistic characteristics in the higher level of layers
- Learn word embeddings through building *bidirectional language models* (biLMs)

ELMo



ELMo: biLMs

- Given a sequence of N tokens, $(t_1, t_2;, \dots, t_N)$:
  - Forward language model:

$$p(t_1, t_2;, \dots, t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2;, \dots, t_{k-1})$$

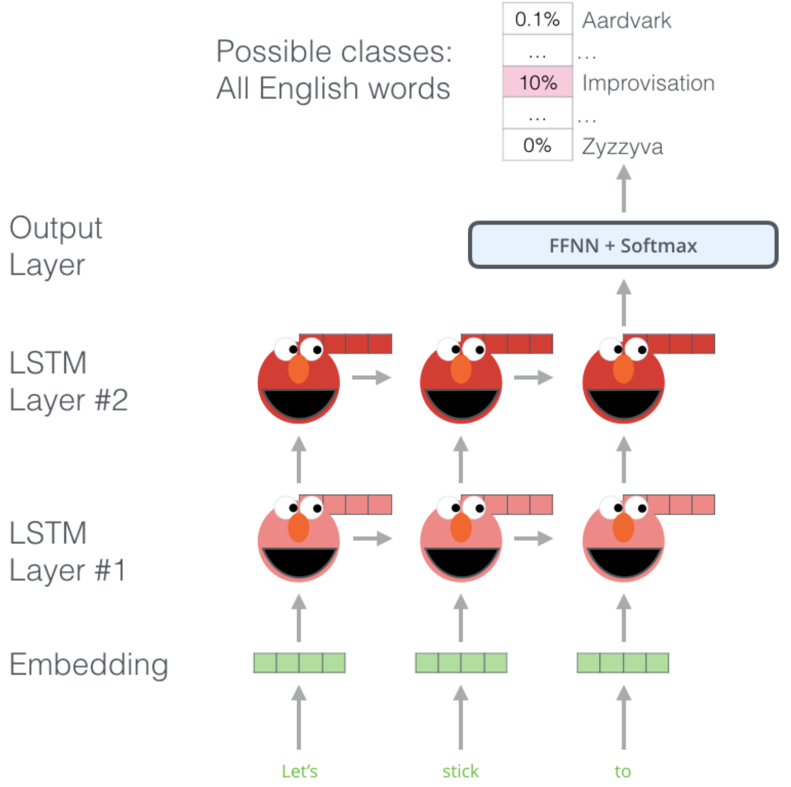
  - Backward language model:

$$p(t_1, t_2;, \dots, t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2};, \dots, t_N)$$

- ELMo is trained to predict the next (previous) word in a sequence of words

ELMo: Training

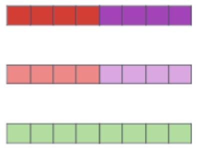
- ELMo is trained to predict the next word in forward LM
- Similarly, backward LM is trained to predict the previous word.



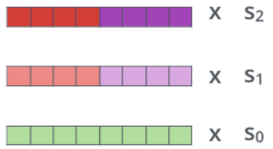
ELMo: Embedding Extraction

Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers

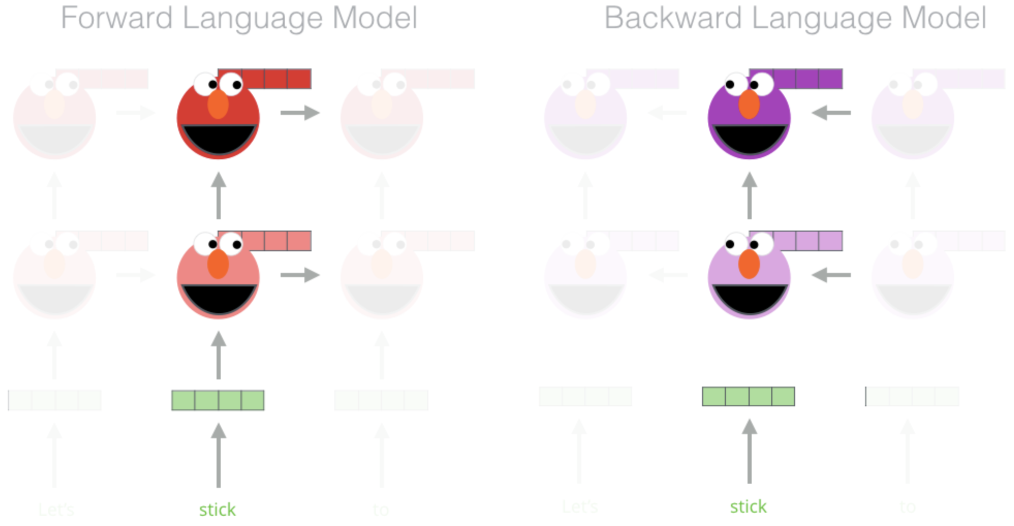


2- Multiply each vector by a weight based on the task



3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context



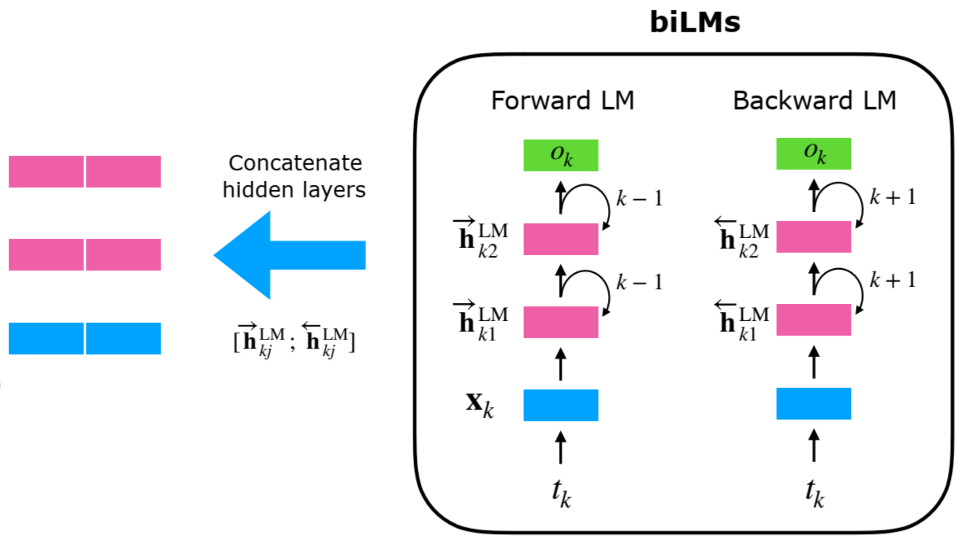
$$ELMo_k^{task} = \gamma^{\text{task}} \sum_{j}^{L} s_j^{task} \boldsymbol{h}_{k,j}^{LM}$$

ELMo: Embedding Extraction

ELMo is a task specific representation. A down-stream task learns weighting parameters

$$\mathbf{ELMo}_k^{\text{task}} = \boxed{\gamma^{\text{task}}} \times \sum \left\{ \begin{array}{l} s_2^{\text{task}} \times \mathbf{h}_{k2}^{\text{LM}} \\ s_1^{\text{task}} \times \mathbf{h}_{k1}^{\text{LM}} \\ s_0^{\text{task}} \times \underset{([\mathbf{x}_k;\, \mathbf{x}_k])}{\mathbf{h}_{k0}^{\text{LM}}} \end{array} \right.$$

Unlike usual word embeddings, ELMo is assigned to every *token* instead of a *type*



Concatenate hidden layers
$[\overrightarrow{\mathbf{h}}_{kj}^{LM}; \overleftarrow{\mathbf{h}}_{kj}^{LM}]$
biLMs
Forward LM
Backward LM
o_k
$k-1$
$k+1$
$\overrightarrow{\mathbf{h}}_{k2}^{LM}$
$\overrightarrow{\mathbf{h}}_{k1}^{LM}$
$\overleftarrow{\mathbf{h}}_{k2}^{LM}$
$\overleftarrow{\mathbf{h}}_{k1}^{LM}$
$\mathbf{x}_k$
t_k

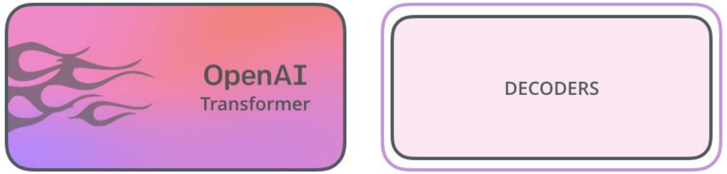
GPT: Generative Pre-Training

GPT: Generative Pre-Training

- Transformers vs GPT vs BERT
- A transformer uses Encoder stack to model input, and uses Decoder stack to model output (using input information from encoder side).
- But if we do not have input, we just want to model the "next word", we can get rid of the Encoder side of a transformer and output "next word" one by one. This gives us GPT.
- If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder of the transformer, that gives us BERT.

GPT: Generative Pre-Training

- There are two subnets in the transforms: encoder and decoder
- It turns out an entire Transformer is not needed to adopt transfer learning and a fine-tunable language model for NLP tasks.
    - It is doable with just the decoder of the transformer.

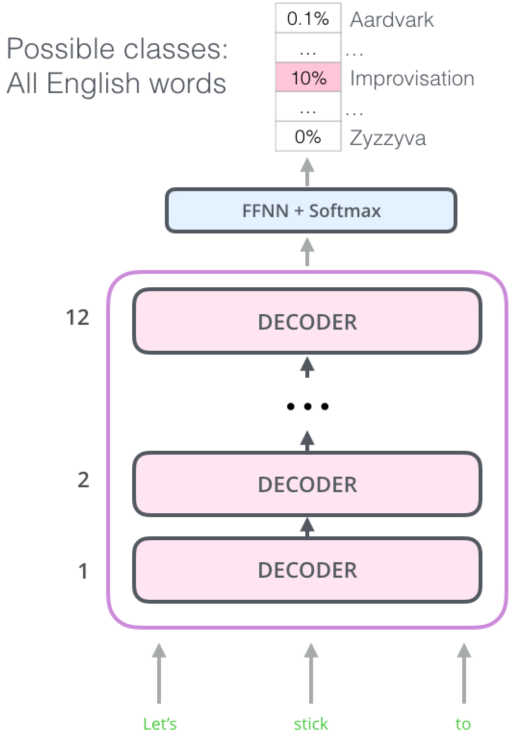


- The task is an unsupervised pre-training:
    - Given an unlabeled corpus of tokens $U = \{u_1, u_2, \dots, u_n\}$, a standard language modeling objective can be used to maximize the following likelihood:

$$L_{lm}(U) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}, \Theta)$$

GPT: Generative Pre-Training

- The model stacked twelve decoder layers
  - The decoder layers would not have the encoder-decoder attention sublayer.
  - It has the self-attention layer
    - masked version, so it doesn't peak the future tokens.
- The model can be trained on the language modeling task: predict the next word using massive (unlabeled) datasets.
- The model parameters will be adapted to the supervised target task.



GPT: Supervised fine-tuning

- Assume there is a labeled dataset C:
  - Each instance consists of a sequence of input tokens, $x_1, \dots, x_m$, along with a label y.
- The inputs are passed through the pre-trained model to obtain the final transformer block's activation h_l^m, which is then fed into an added linear output layer with parameters W_y to predict y

$$P(y|x_1, \dots, x_m) = \text{softmax}\left(h_l^m W_y\right)$$

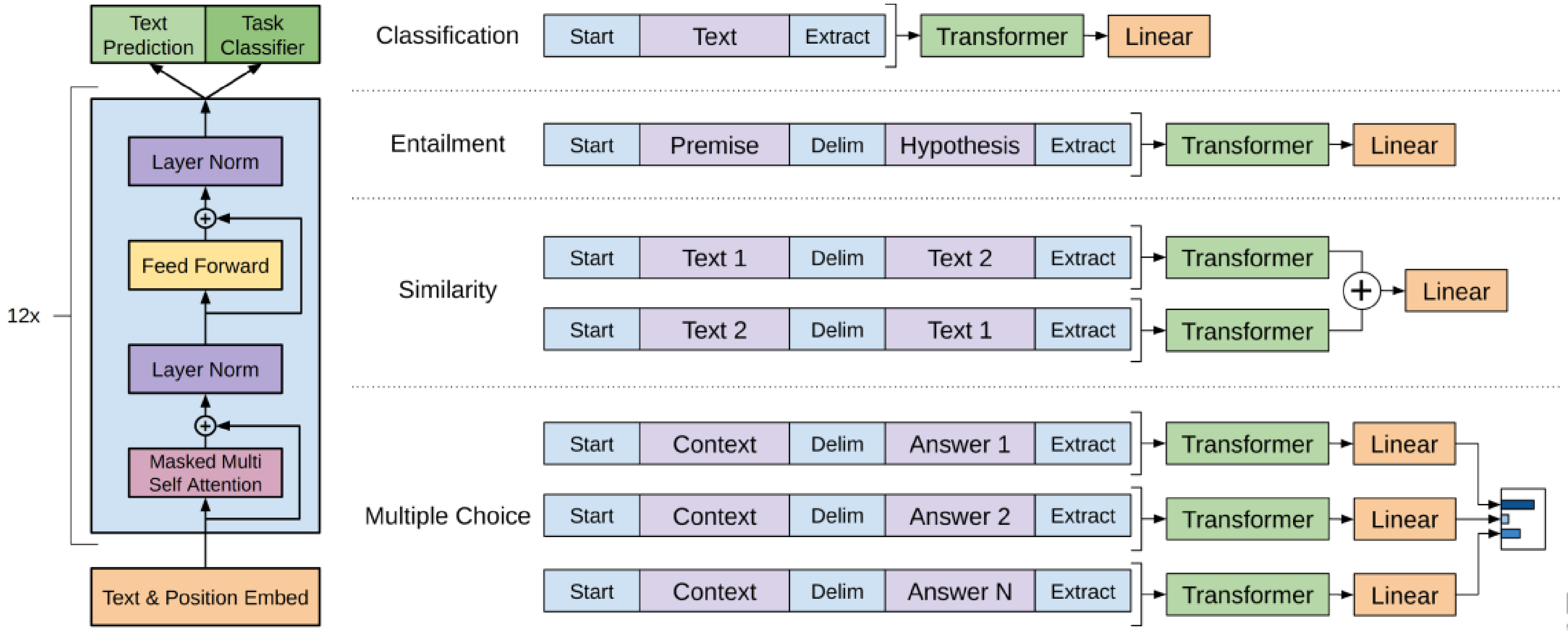
  then the objective to maximize will be:

$$L_{fine}(C) = \sum_{(x,y)} \log P(y|x_1, \dots, x_m)$$

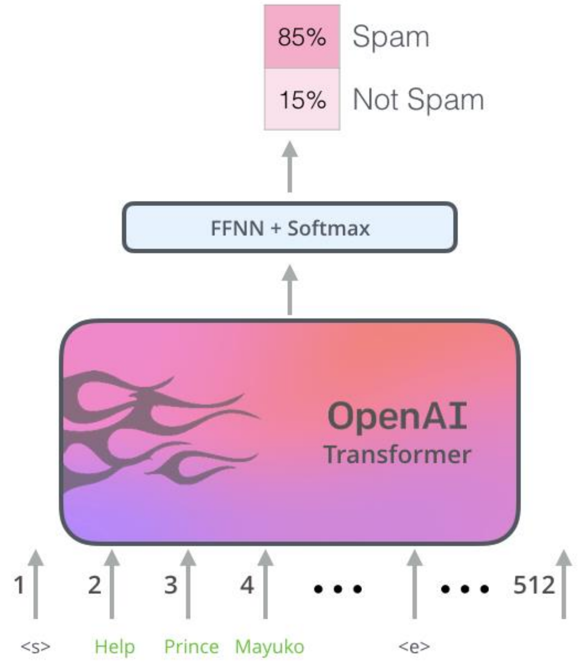
- The final loss is a combination of fine-tuning loss plus language modeling loss as follows:

$$L(C) = L_{fine}(C) + \lambda L_{lm}(C)$$

GPT: Supervised fine-tuning



GPT: Text Classification



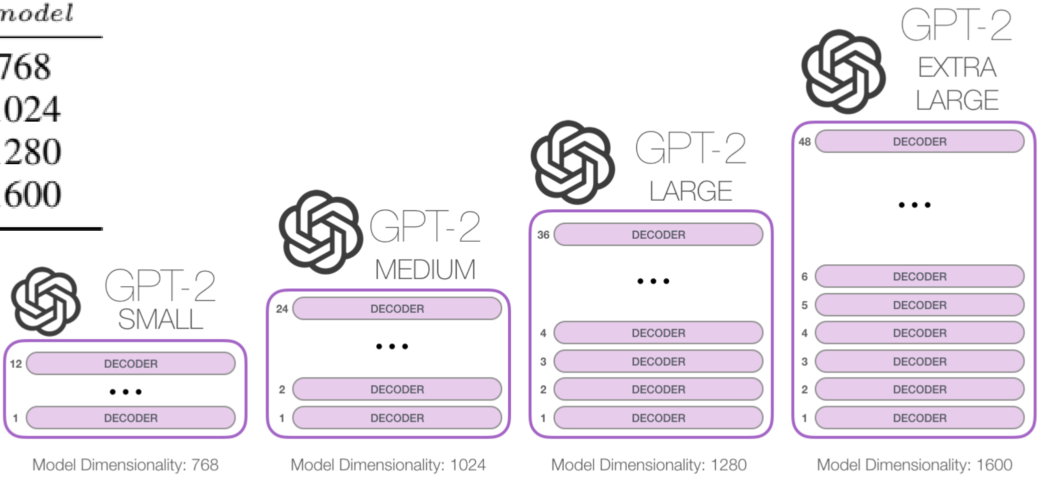
GPT: Supervised fine-tuning

- **Task-specific input transformations**:
  - Convert structured inputs into an ordered sequence that the pre-trained model can process.
- **Textual entailment:** the premise **p** and hypothesis **h** token sequences are concatenated, with a delimiter token ($) in between.
- **Similarity:** the input sequence is modified to contain both possible sentence orderings (with a delimiter in between) and process each independently to produce two sequence representations h_l^m which are added element-wise before being fed into the linear output layer.
- **Question Answering and Commonsense Reasoning**: we are given a context document z, a question q, and a set of possible answers $\{a_k\}$. First generate concatenation input $[z; q; \$; a_k]$ for each answer and process them independently and then normalized via a softmax layer to select the best answer.

GPT-2

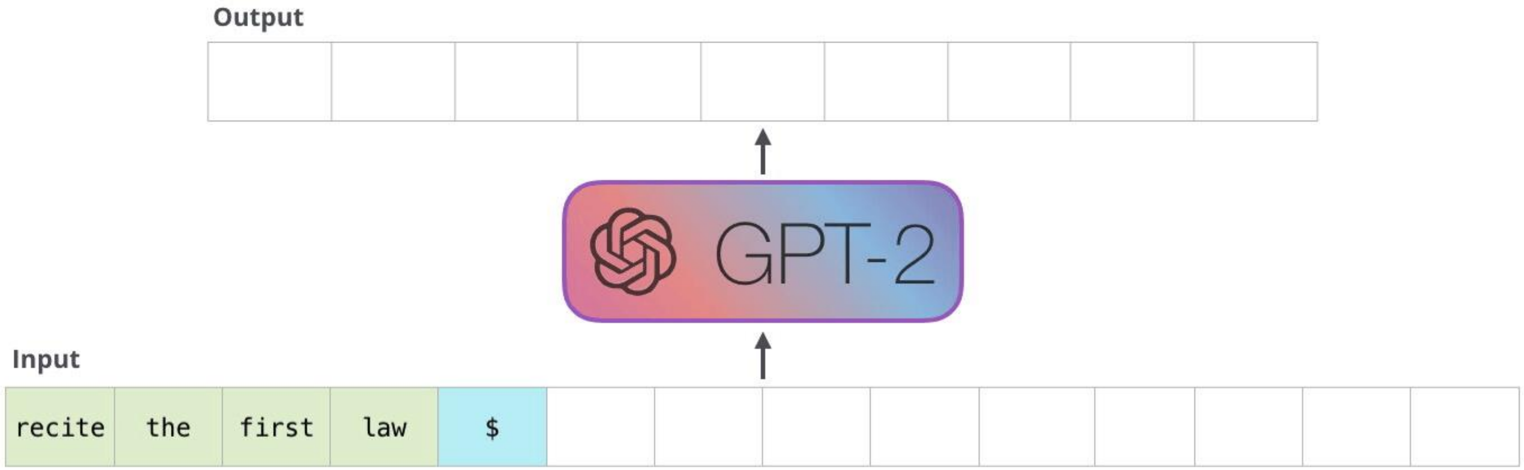
- **GPT-2: Language Models are Unsupervised Multitask Learners**
- The model is based on GPT with some modifications but with more layers

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

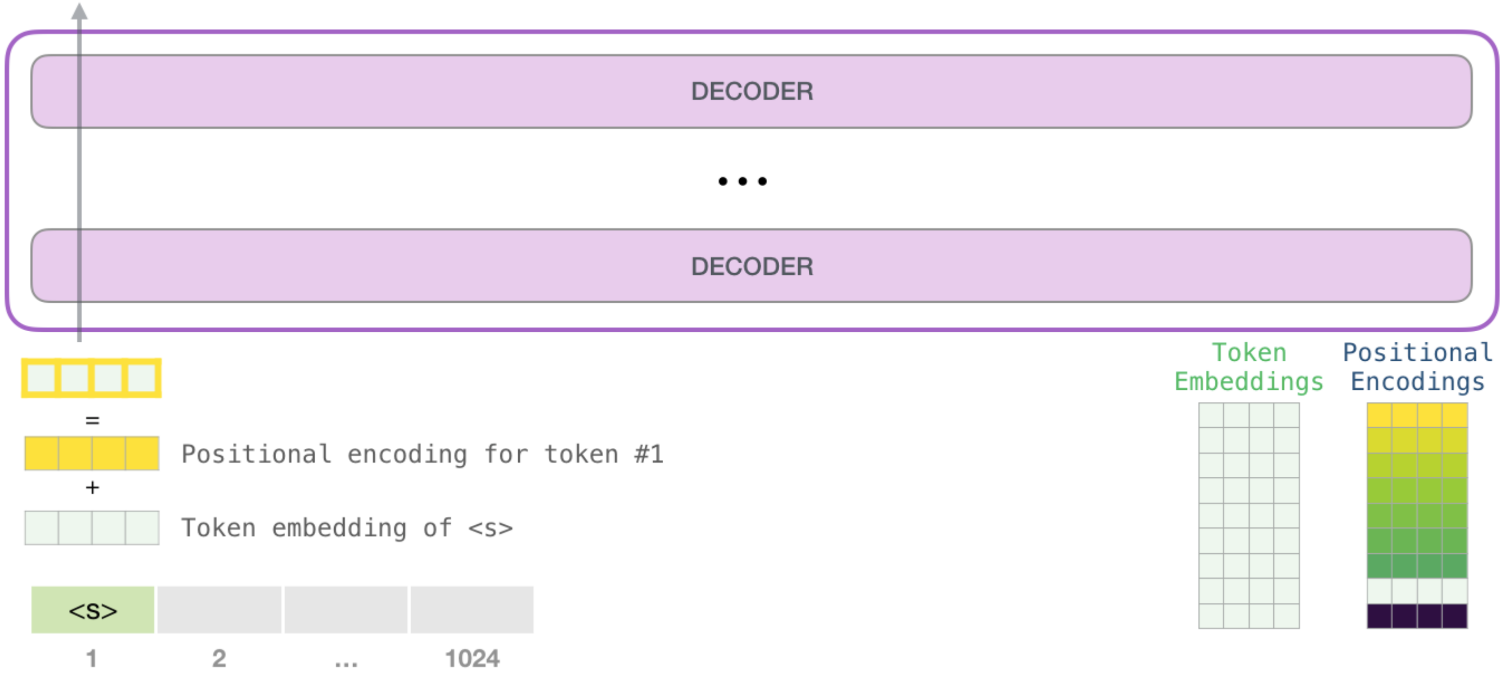


GPT-2

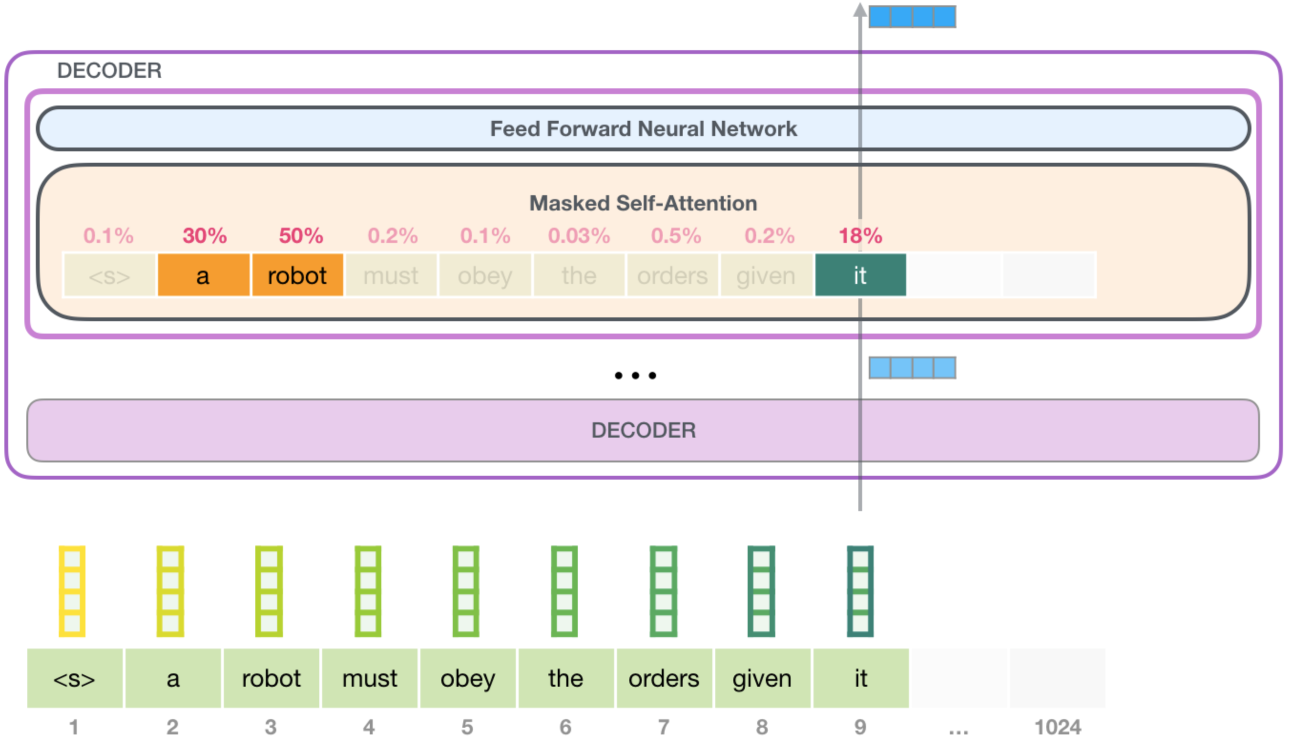
- GPT2 outputs one token at a time unlike BERT (will see latter)
  - Generate the output in a “auto-regressive” mode.



GPT-2: Processing the Input



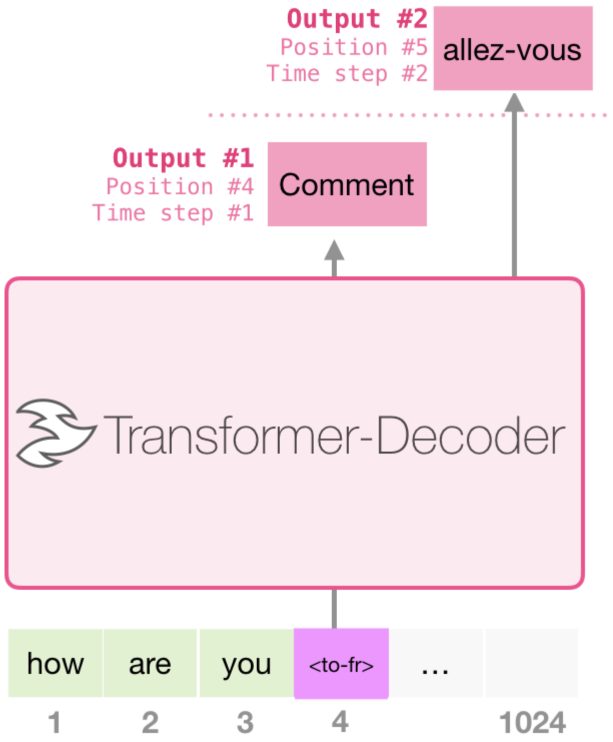
GPT-2: Processing the Input



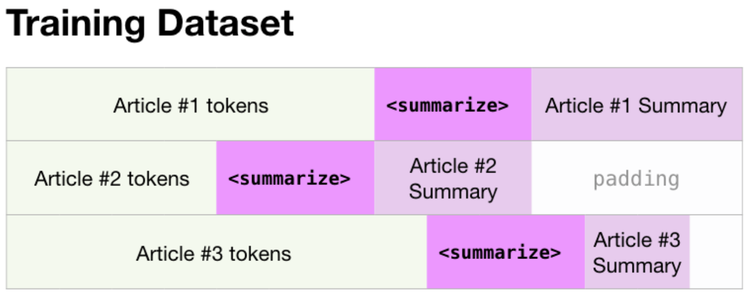
GPT-2 Application: Translation

Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu’ils	mangent	de
good	morning	<to-fr>	Bonjour				

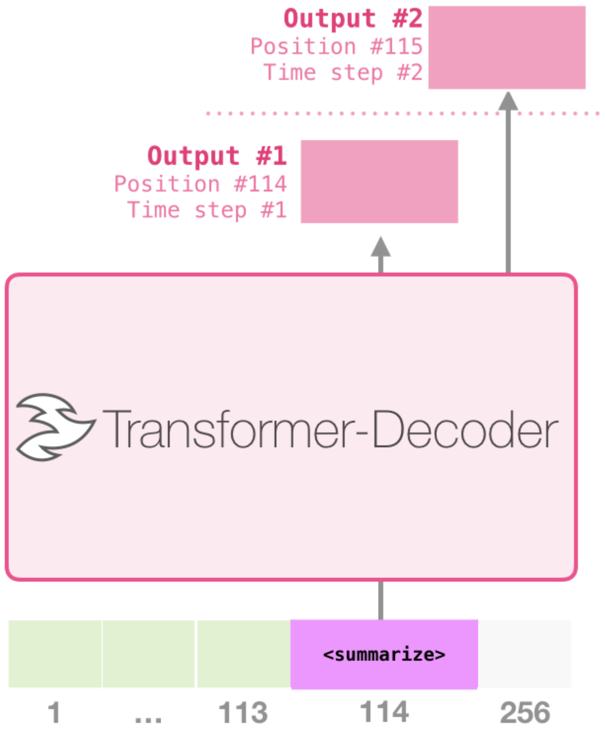


GPT-2 Application: Summarization



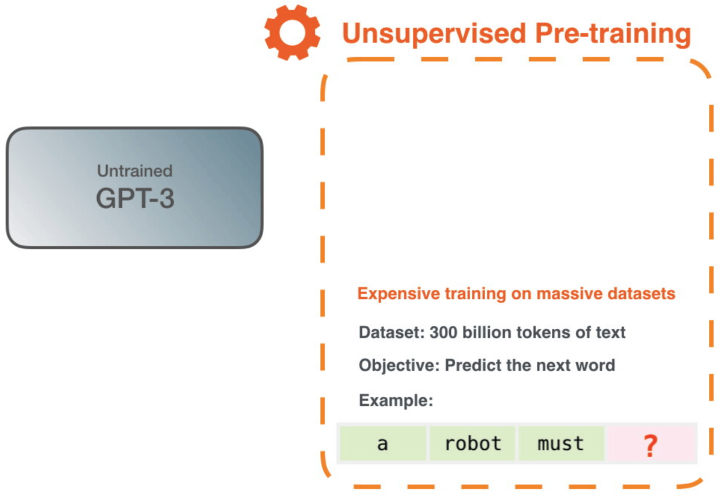
Training Dataset

Article #1 tokens	`<summarize>`	Article #1 Summary	
Article #2 tokens	`<summarize>`	Article #2 Summary	padding
Article #3 tokens	`<summarize>`	Article #3 Summary	



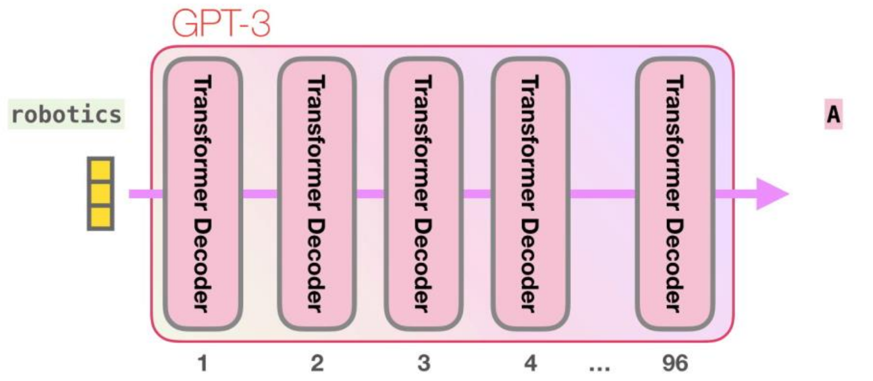
GPT-3

- A very large GPT based model with 175 billion parameters
  - It was estimated to cost 355 GPU years and cost $4.6m.

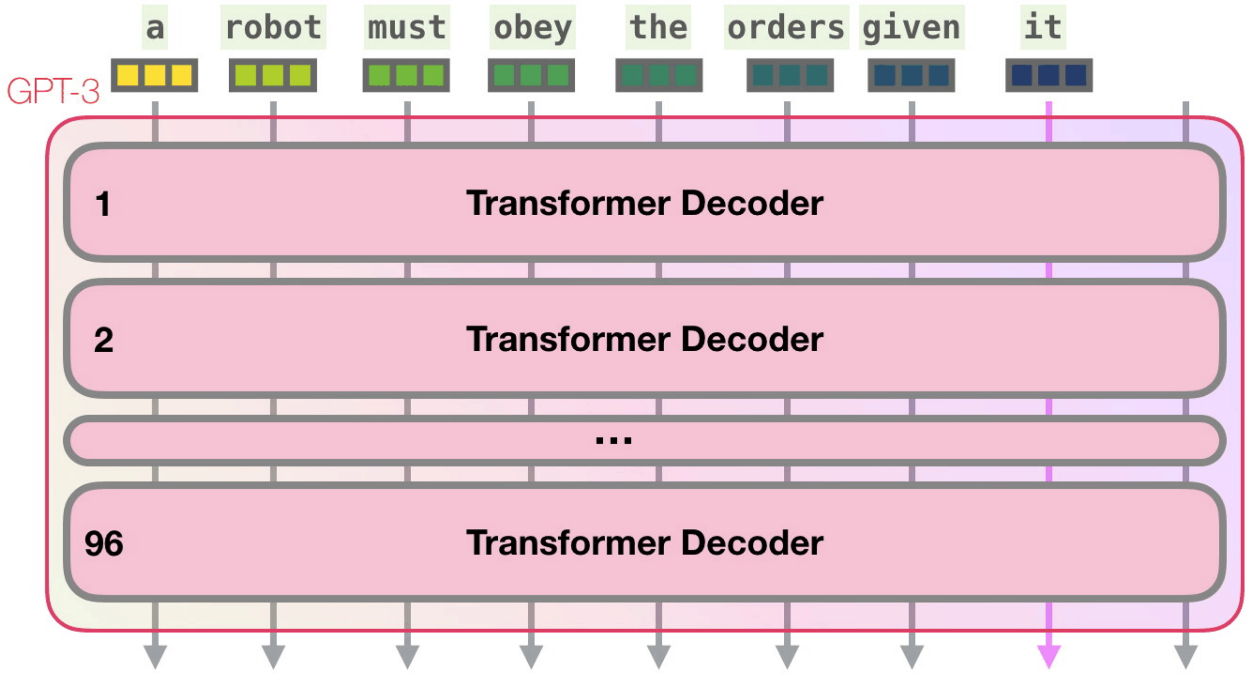

Unsupervised Pre-training
Untrained
GPT-3
Expensive training on massive datasets
Dataset: 300 billion tokens of text
Objective: Predict the next word
Example:
a
robot
must
?

GPT-3

- GPT3 can accept 2048 tokens as an input.
- The main calculations of the GPT3 occur inside its stack of 96 transformer decoder layers.
- GPT-3 was trained with a larger dataset
  - 570GB of text compared to 40GB for GPT-2.

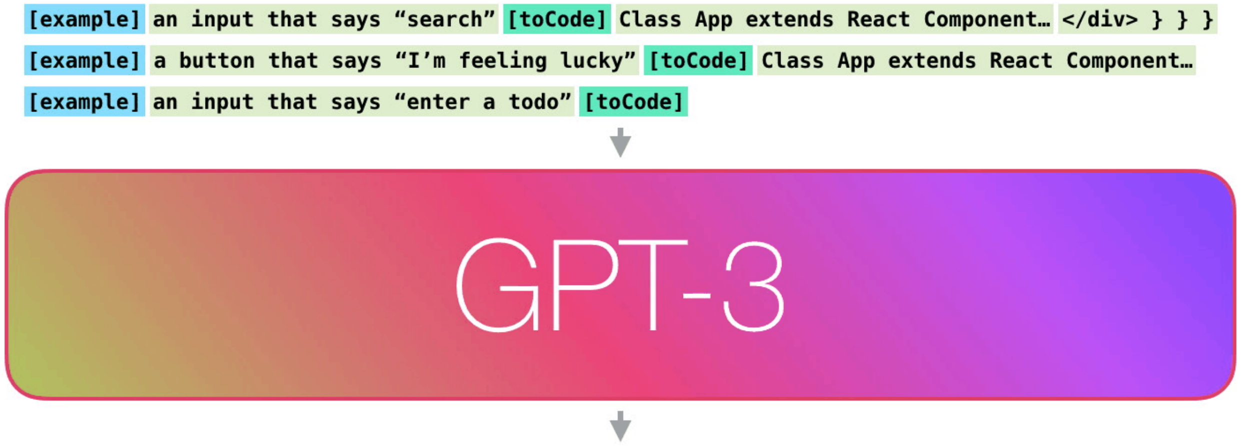


GPT-3


a
robot
must
obey
the
orders
given
it
GPT-3
1 Transformer Decoder
2 Transformer Decoder
...
96 Transformer Decoder

GPT-3

- It is mainly used in zero/few shot learning scenario



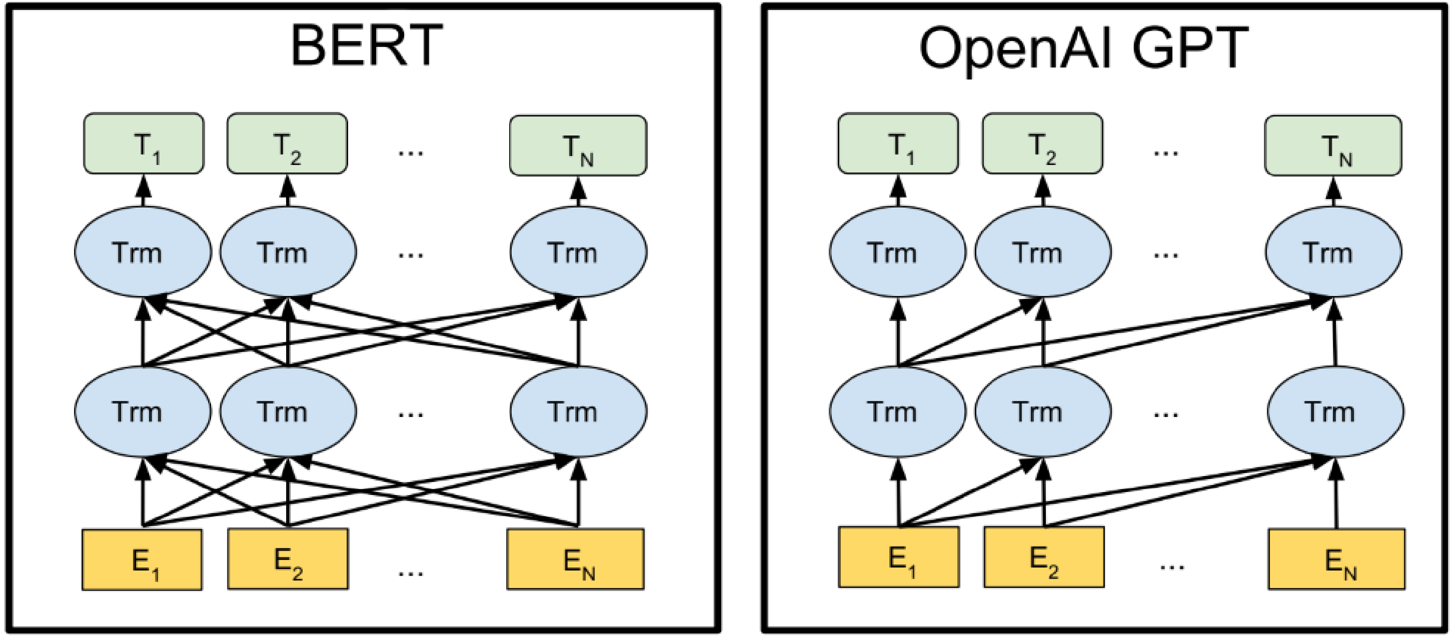
[example] an input that says "search" [toCode] Class App extends React Component… </div> } } }
[example] a button that says "I'm feeling lucky" [toCode] Class App extends React Component…
[example] an input that says "enter a todo" [toCode]
GPT-3

BERT: Bidirectional Encoder Representations from Transformers

BERT: From Decoders to Encoders

- Designed for pre-training deep bidirectional representations from unlabeled text
- Conditions on left and right context in all layers
  - GPTs only uses left context
- Pre-trained model can be finetuned with one additional output layer for many tasks (e.g., NLI, QA, sentiment)
- For many tasks, no modifications to the Bert architecture are required
- At 2019 it achieved SotA results on 11 tasks using pre-training or finetuning approach
  - Main attraction of BERT

BERT vs GPT

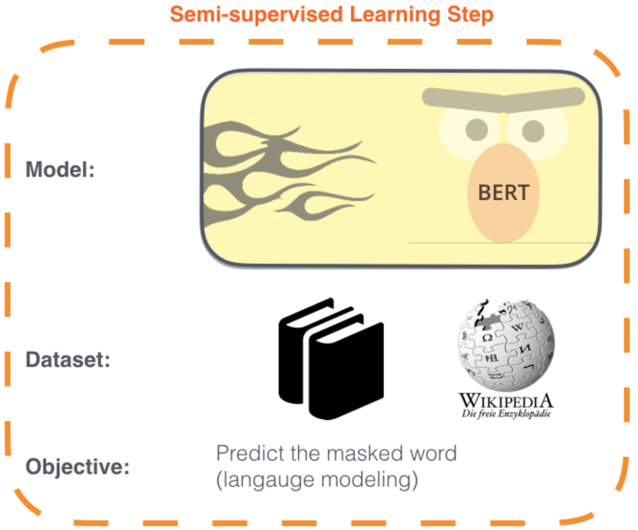


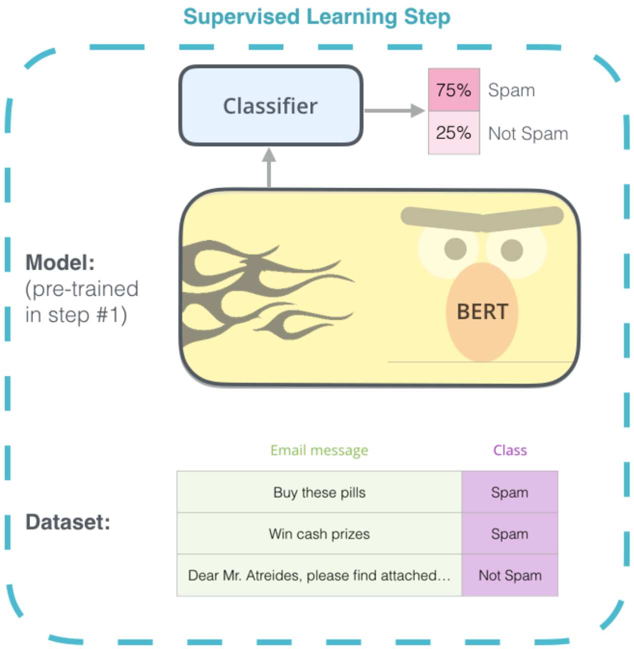
BERT Usage Steps

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

2 - Supervised training on a specific task with a labeled dataset.

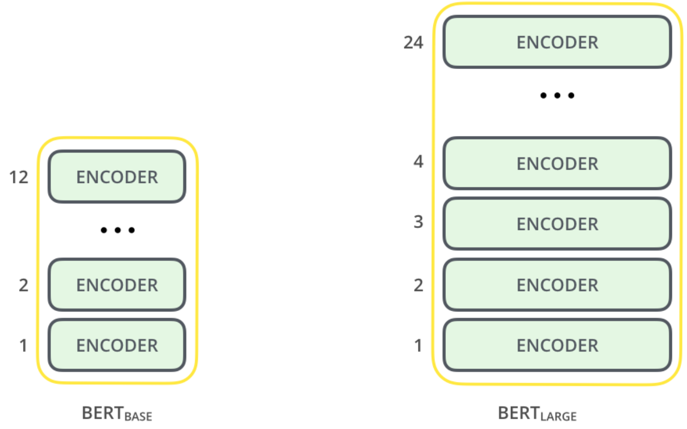




Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached…	Not Spam

Model Architecture

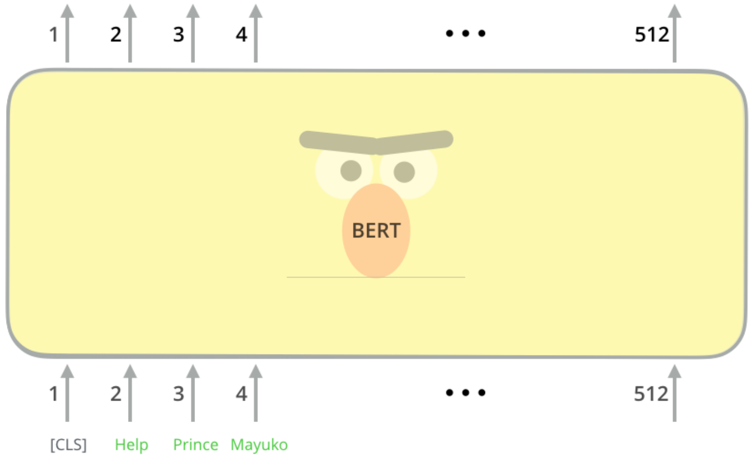
- Model variants:
  - **BERT BASE**: Comparable in size to the OpenAI Transformer (GPT) in order to compare performance
    - 12 encoder layers, 768 hidden units in FFNs, 12 attention heads
  - **BERT LARGE**: A ridiculously huge model which achieved the state of the art results reported in the paper
    - 24 encoder layers, 1024 hidden units in FFNs, 16 attention heads

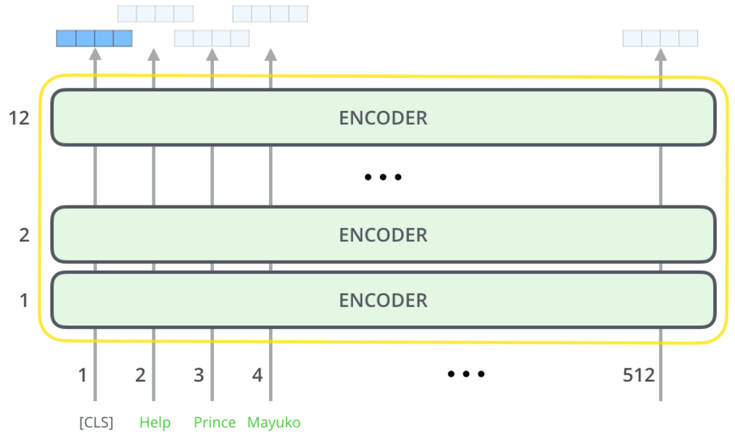


Input/Output Representation

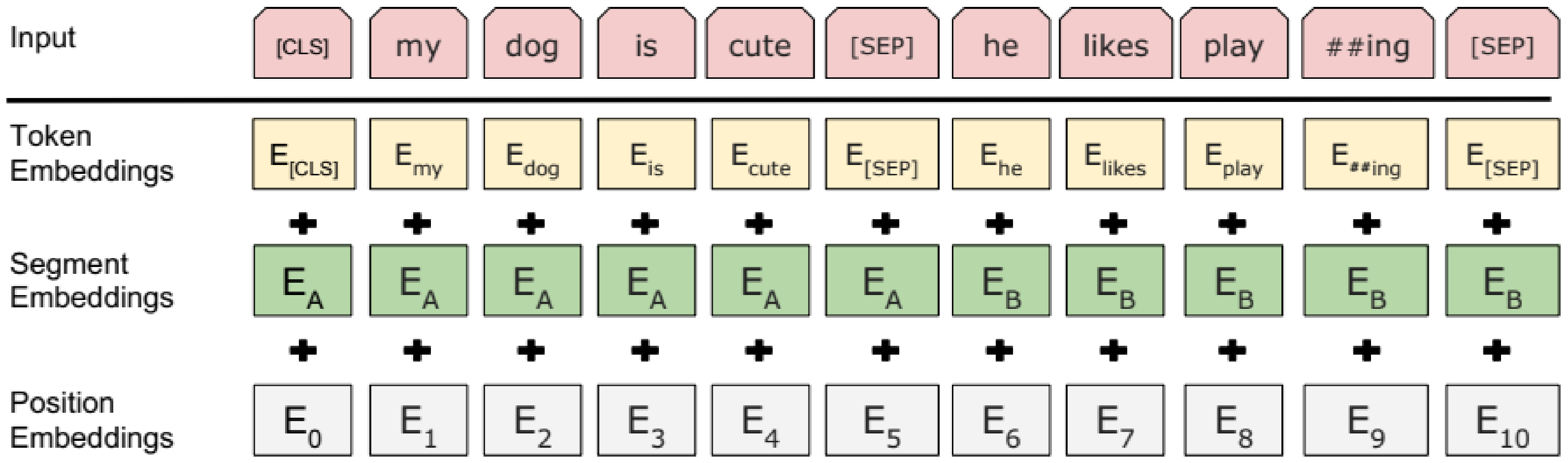
- Input sequence: can be $\langle Question, Answer \rangle$ pair, single sentence, or any other string of tokens
- 30,000 token vocabulary, represented as WordPiece embeddings (handles OOV words)
- First token is always [CLS]: aggregate sentence representation for classification tasks
- Sentence pairs separated by [SEP] token
- BERT is trained in masked language model scenario
    - [MASK] token is used to randomly mask 15% of tokens

Input/Output Representation

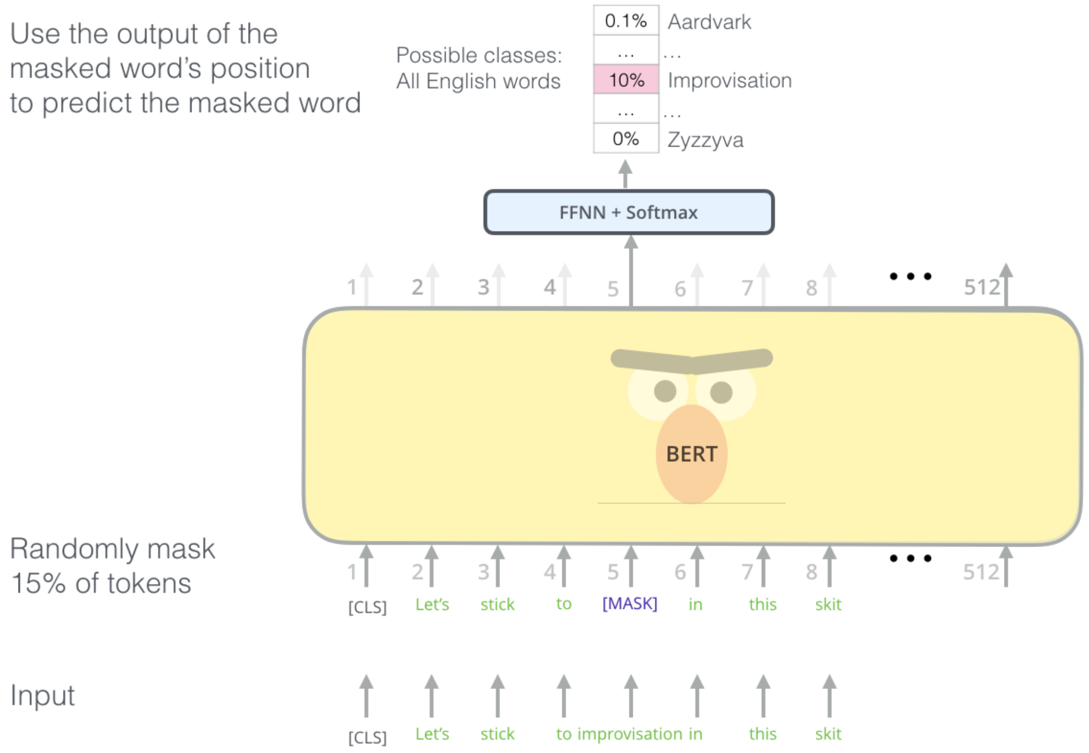




Input/Output Representation


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
E[CLS] E my E dog E is E cute E[SEP] E he E likes E play E##ing E[SEP]
Segment Embeddings
E A E A E A E A E A E A E B E B E B E B E B
Position Embeddings
E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10

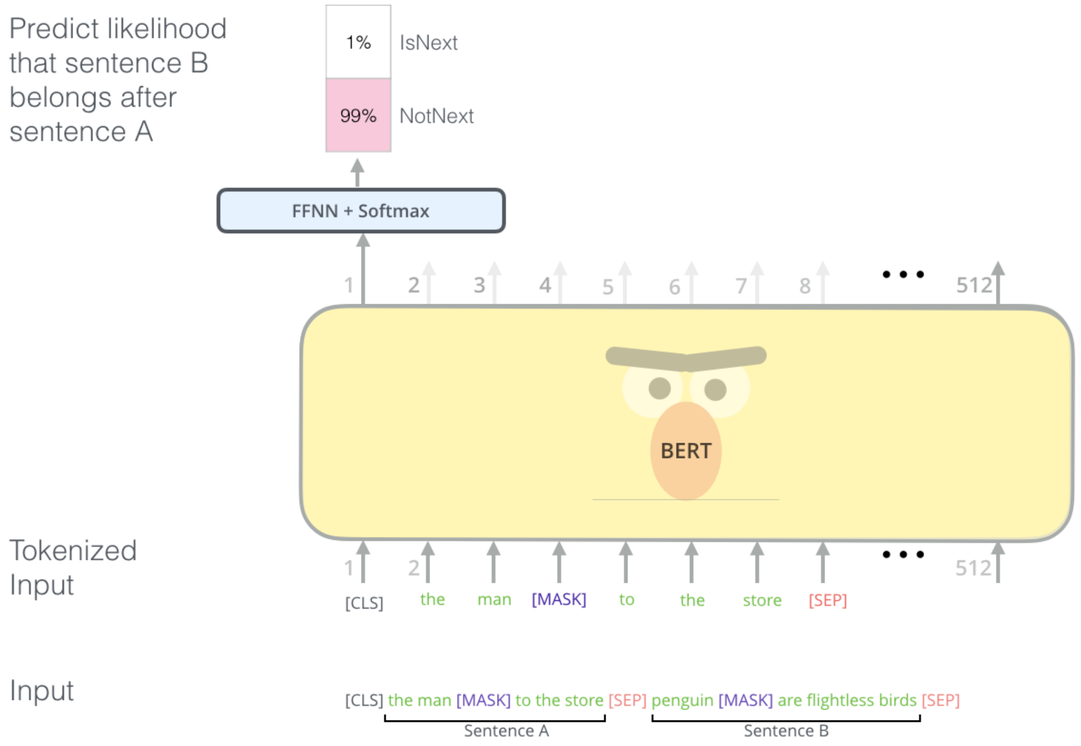
BERT: Masked Language Model


Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
...
10% Improvisation
...
0% Zyzzyva
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
BERT
Randomly mask 15% of tokens
[CLS] Let's stick to [MASK] in this skit
Input
[CLS] Let's stick to improvisation in this skit

BERT: Masked Language Model

- Mask 15% of the tokens in the input sequence; train the model to predict these
- Problem: masking creates mismatch between pre-training and finetuning: [MASK] token is not seen during fine-tuning. Solution:
  - Do not always replace masked words with [MASK]. If i^{th} token is chosen, we replace the i^{th} token with:
    1. The [MASK] token 80% of the time;
    2. A random token 10% of the time;
    3. The unchanged i^{th} token 10% of the time.
  - Now use i^{th} output to predict original token with cross entropy loss.

BERT: Next Sentence Prediction

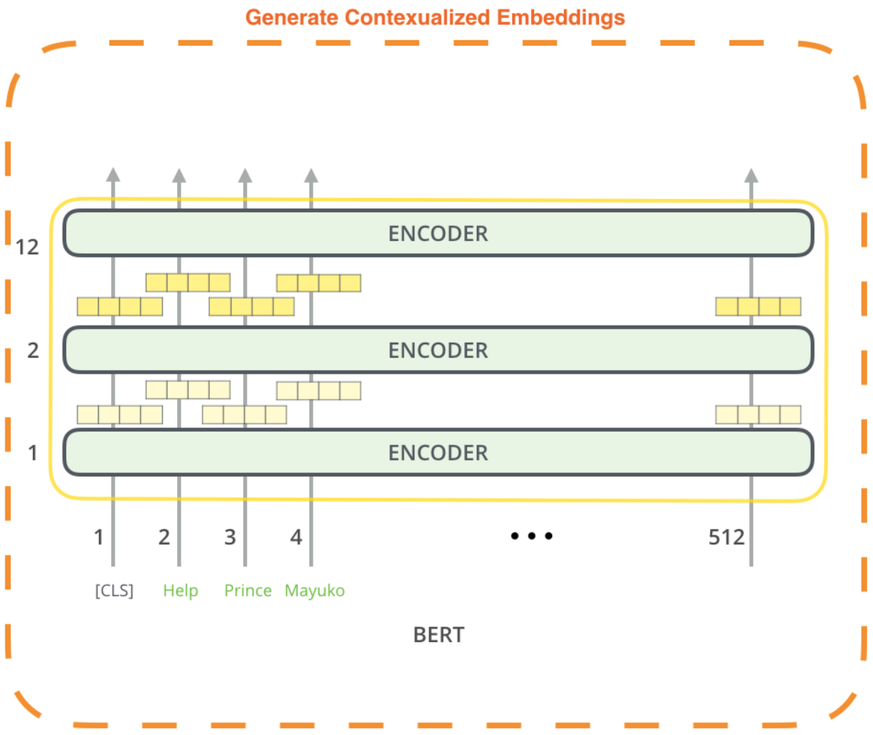


Predict likelihood that sentence B belongs after sentence A
1%
IsNext
99%
NotNext
FFNN + Softmax
1
2
3
4
5
6
7
8
...
512
BERT
Tokenized Input
[CLS]
the
man
[MASK]
to
the
store
[SEP]
Input
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B

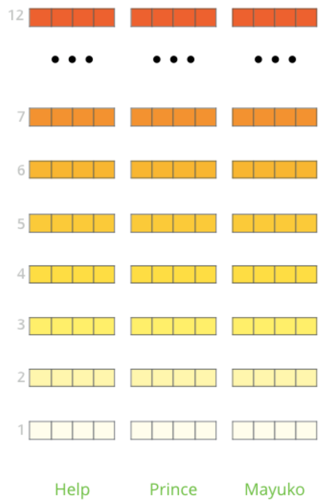
BERT: Next Sentence Prediction

- Bert is designed to be used for tasks such a question answering (QA) and natural language inference (NLI) that require sentence pairs
- Bert uses a special mechanism to capture this: pre-training on next sentence prediction task
- Generate training data: chose two sentences A and B, such that 50% of the time B is the actual next sentence of A, and 50% of the time a randomly selected sentence.

BERT for Feature Extraction

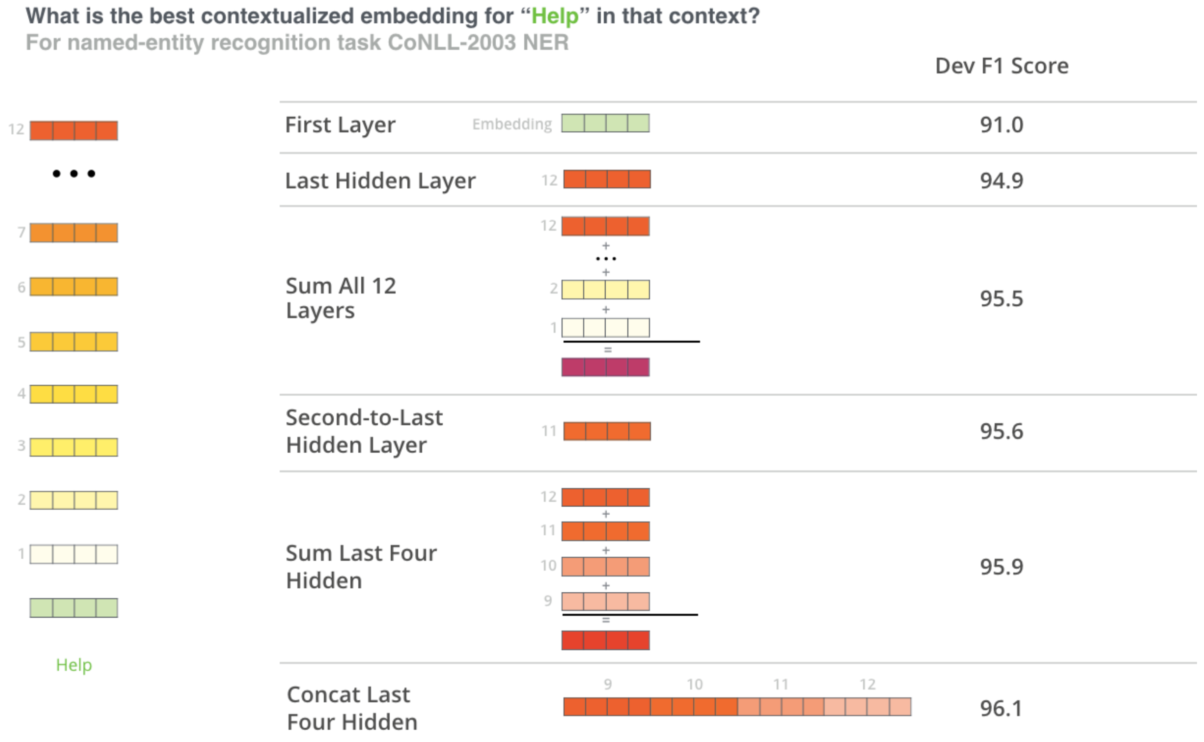


The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

BERT for Feature Extraction


What is the best contextualized embedding for "Help" in that context?

For named-entity recognition task CoNLL-2003 NER

	Dev F1 Score
First Layer	91.0
Last Hidden Layer	94.9
Sum All 12 Layers	95.5
Second-to-Last Hidden Layer	95.6
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1

Other BERT Variants

- **RoBERTa**: A Robustly Optimized BERT Pretraining Approach
  - Optimizing BERT training by selecting proper hyper-parameters and training data size
- **ALBERT**: A Lite BERT for Self-supervised Learning of Language Representations
  - Presents two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT.
  - The best model has better results on the several NLP benchmarks while having fewer parameters compared to BERT-large.
- And others: SemBERT, BERTSP, SpanBERT, AMBERT, DistilBERT, etc.

Thanks for your attention

References and IP Notice

- [1] Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." *arXiv preprint arXiv:1508.07909* (2015).

- Most of the figures are selected from https://jalammar.github.io web site.