



**girafe  
ai**

# Lecture 05: Context based models, BERT overview

**Radoslav Neychev**

Fall 2021, Moscow, Russia

1. OpenAI Transformer
2. ELMo
3. BERT
4. GPT-2 & GPT-3
5. Q & A

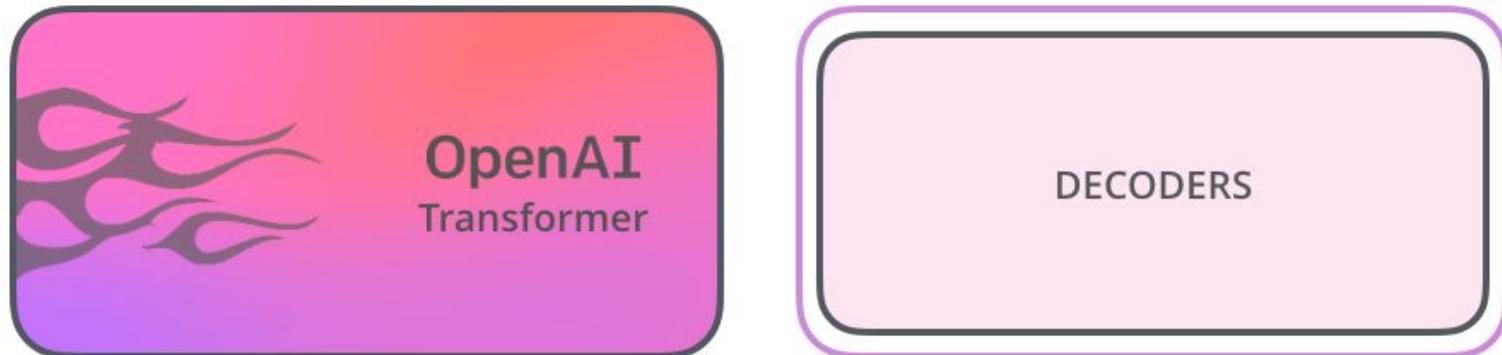
Based on: <http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture13-contextual-representations.pdf>  
<https://jalammar.github.io/illustrated-transformer/>  
<http://jalammar.github.io/illustrated-bert/>  
<https://medium.com/mlreview/understanding-building-blocks-of-ulmfit-818d3775325b>



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

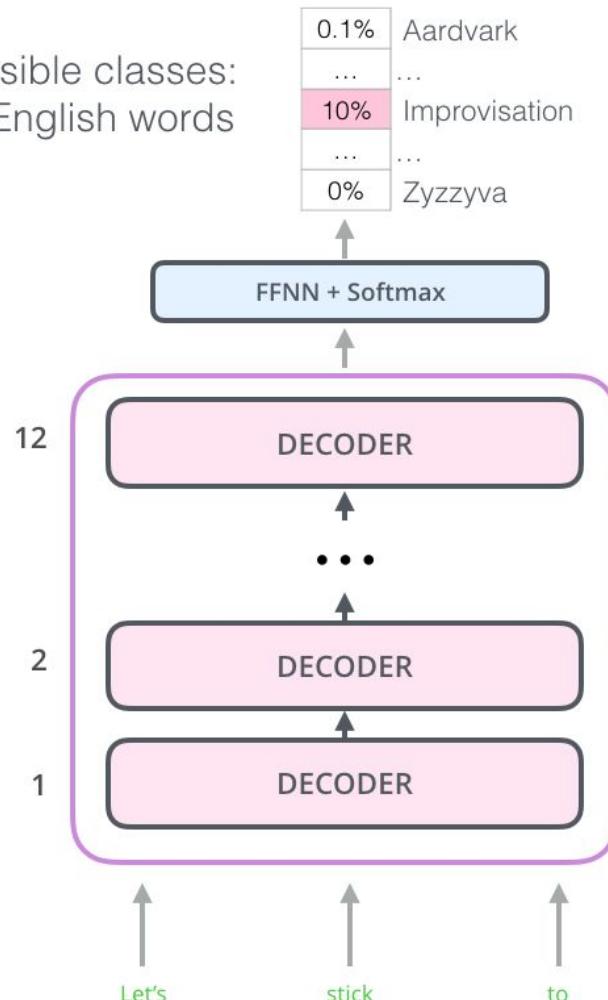
# OpenAI Transformer

- The Encoder-Decoder structure of the transformer made it perfect for machine translation
- But what about sentence classification?
- **Main goal: pre-train a language model that can be fine-tuned for other tasks**



# OpenAI Transformer

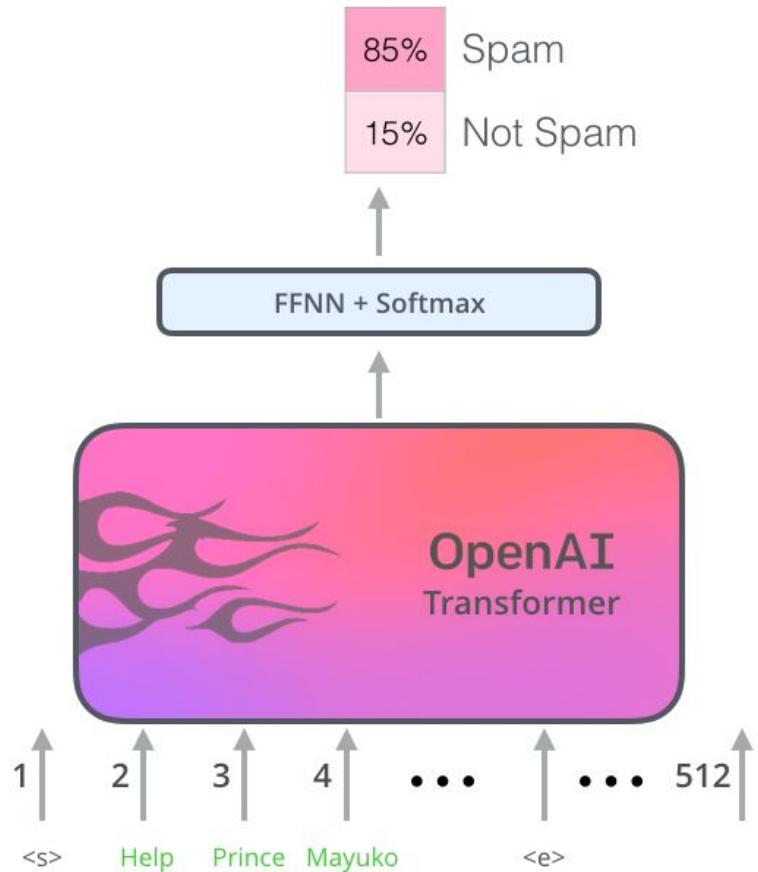
Possible classes:  
All English words



## Differences from vanilla Transformer:

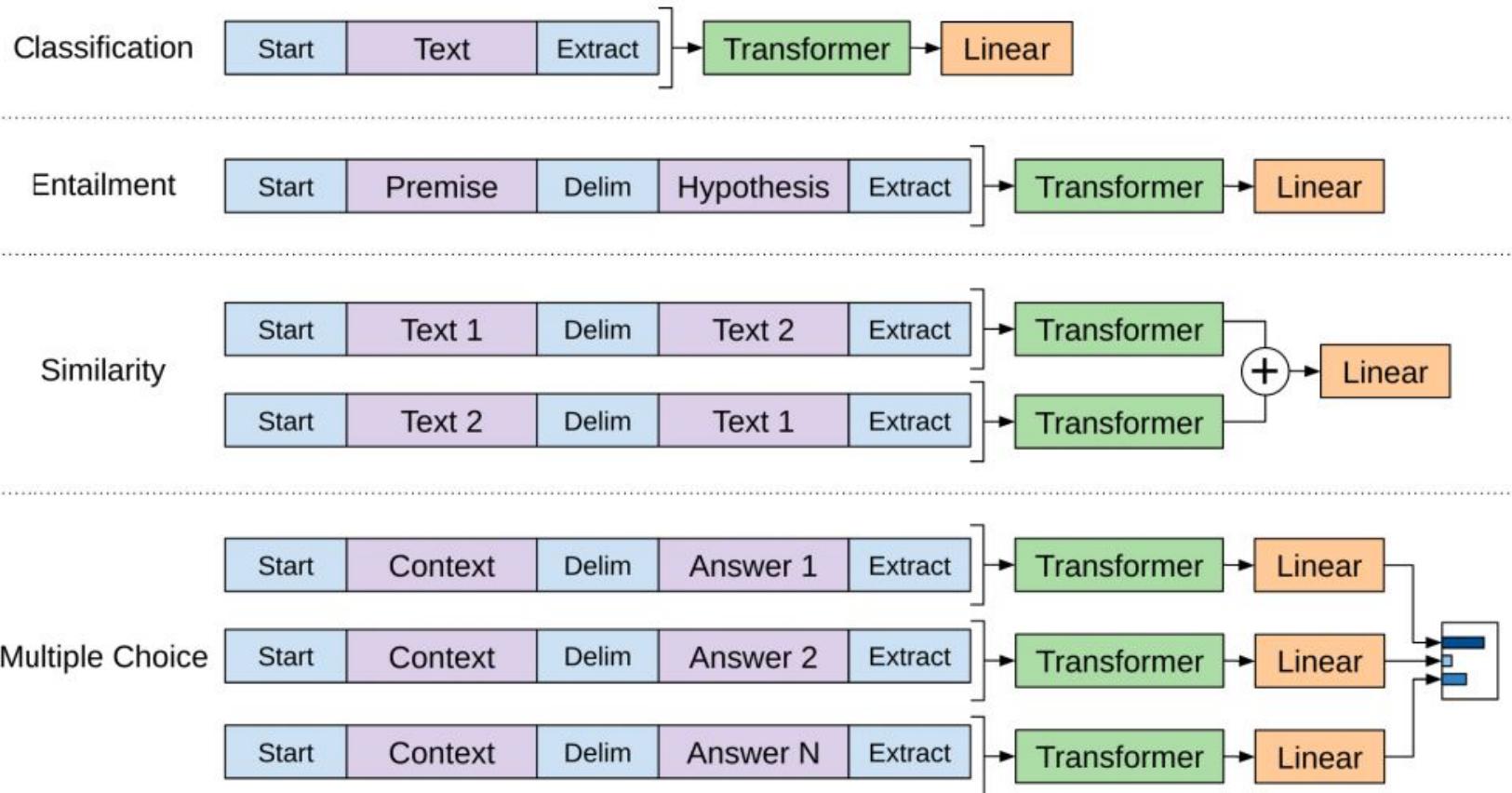
- no encoder
- decoder layers would not have the encoder-decoder attention sublayer
- Pre-train the model on predicting the next word using massive (unlabeled) datasets

# OpenAI Transformer



- During pre-training phase layers have been tuned to reasonably handle language
- Now let's use it for downstream tasks (e.g. sentence classification)

# Input transformations for different tasks





# **ELMo: context that matters**

# ELMo: contextualized word embeddings

*“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?”*



Peters et. al., 2017, McCann et. al., 2017,  
Peters et. al., 2018 in the ELMo paper

ELMo – deep contextualized  
word representations

## What does it stand for?



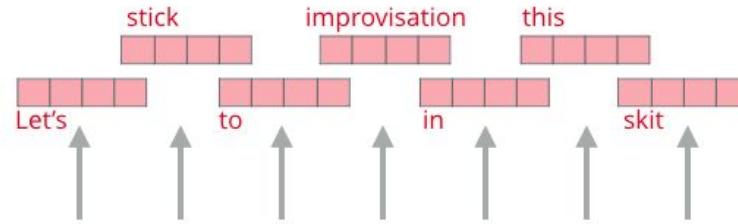
1. Expedited Labour Market Opinion
2. Electric Light Machine Organization
3. Enough Let's Move On
4. Embeddings from Language Models

# ELMo: contextualized word embeddings



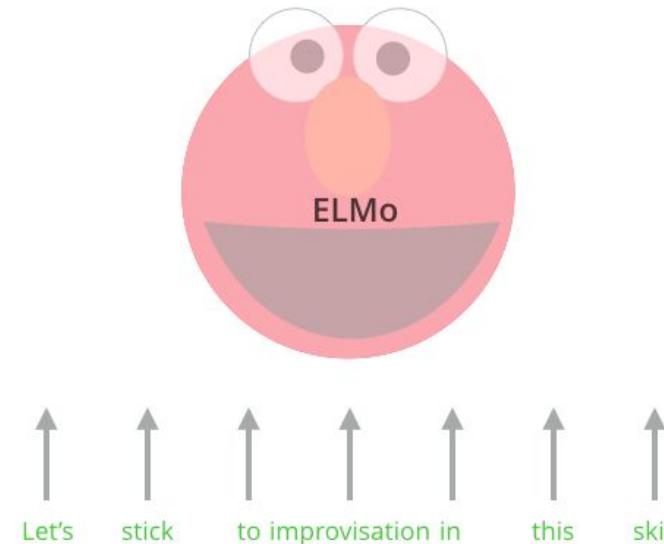
# ELMo: Contextualized word embeddings

ELMo  
Embeddings



- uses a bi-directional LSTM trained on Language Modeling task
- a model can learn without labels

Words to embed



# Bidirectional Language Models (biLMs)

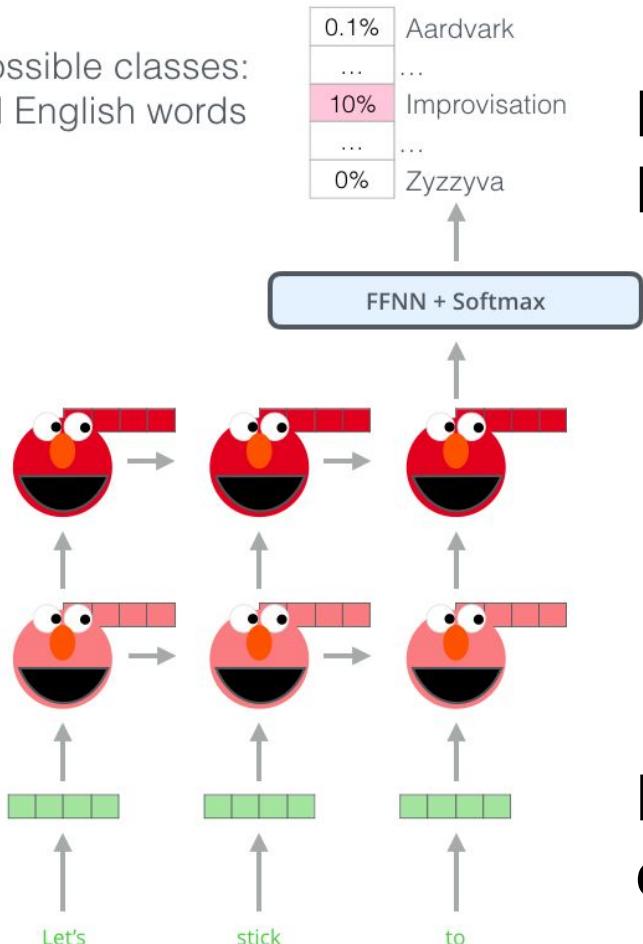
Possible classes:  
All English words

Output  
Layer

LSTM  
Layer #2

LSTM  
Layer #1

Embedding



biLMs consist of forward and backward LMs:

- Forward:

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

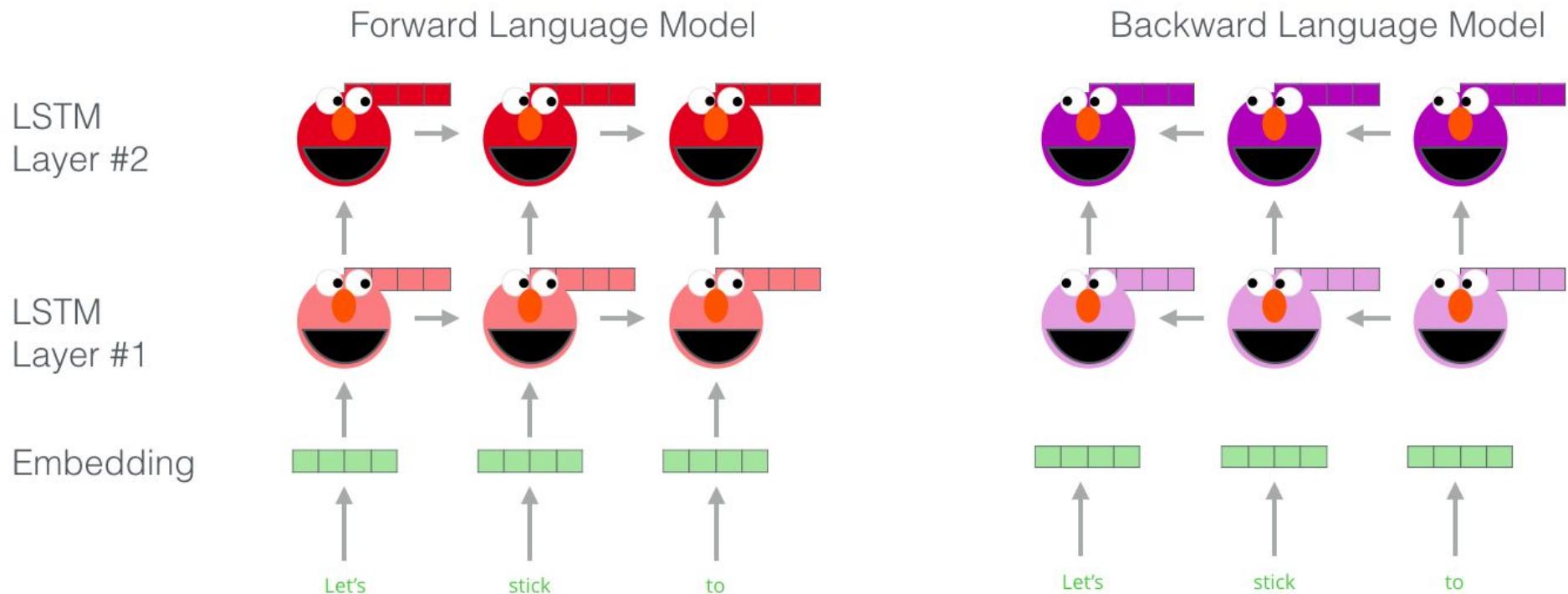
- Backward:

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

LSTM predicts next word in both directions to build biLMs

# ELMo: main pipeline

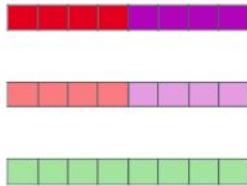
Embedding of “stick” in “Let’s stick to” - Step #1



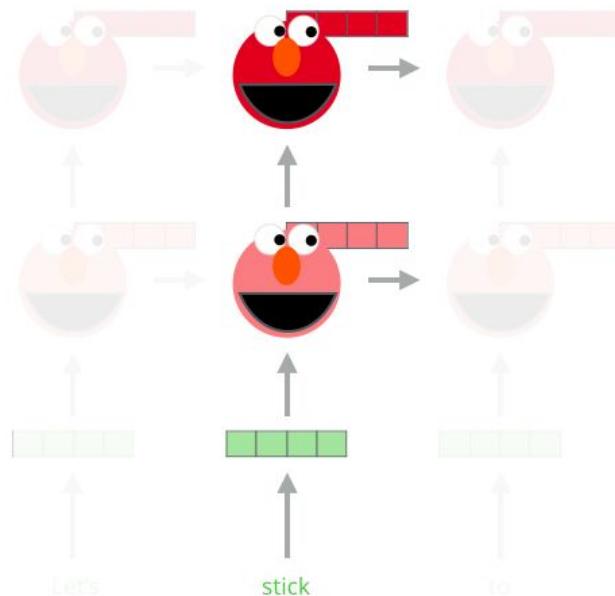
# ELMo: main pipeline

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

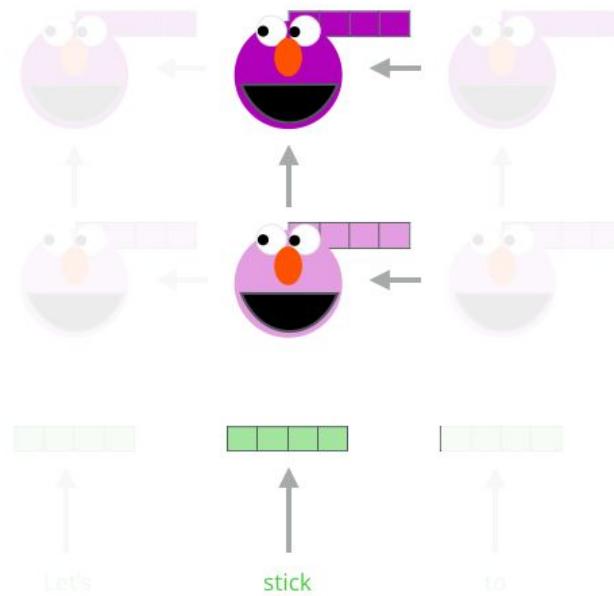
1- Concatenate hidden layers



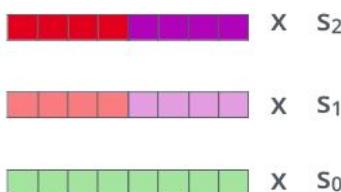
Forward Language Model



Backward Language Model



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



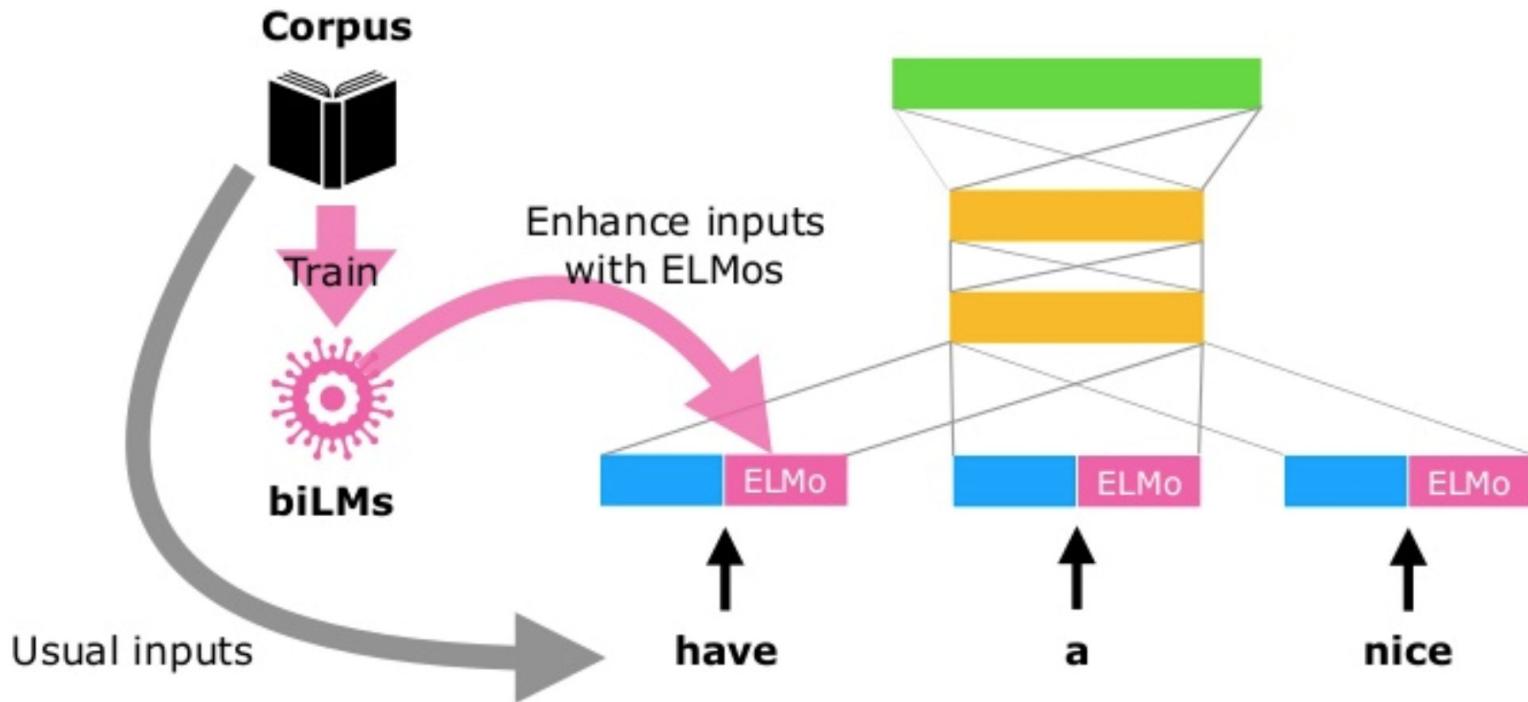
3- Sum the (now weighted) vectors



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

ELMo represents a word as a linear combination of corresponding hidden layers

ELMo can be integrated to almost all neural NLP tasks with simple concatenation to the embedding layer



- Pretrained ELMo models: <http://allennlp.org/elmo>
- AllenNLP is a library on the top of PyTorch
- Higher levels seems to catch semantics while lower layer probably capture syntactic features



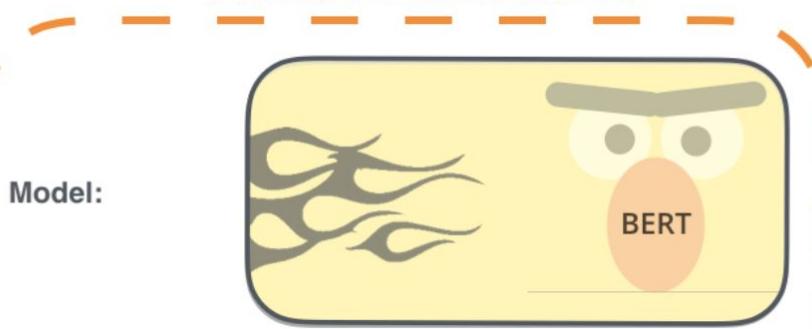
# BERT

Bidirectional Encoder Representations from Transformers

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

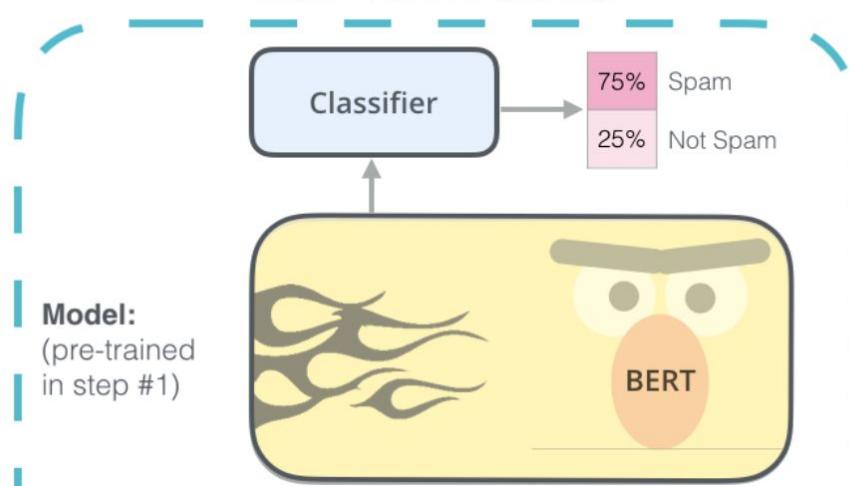
The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

### Semi-supervised Learning Step



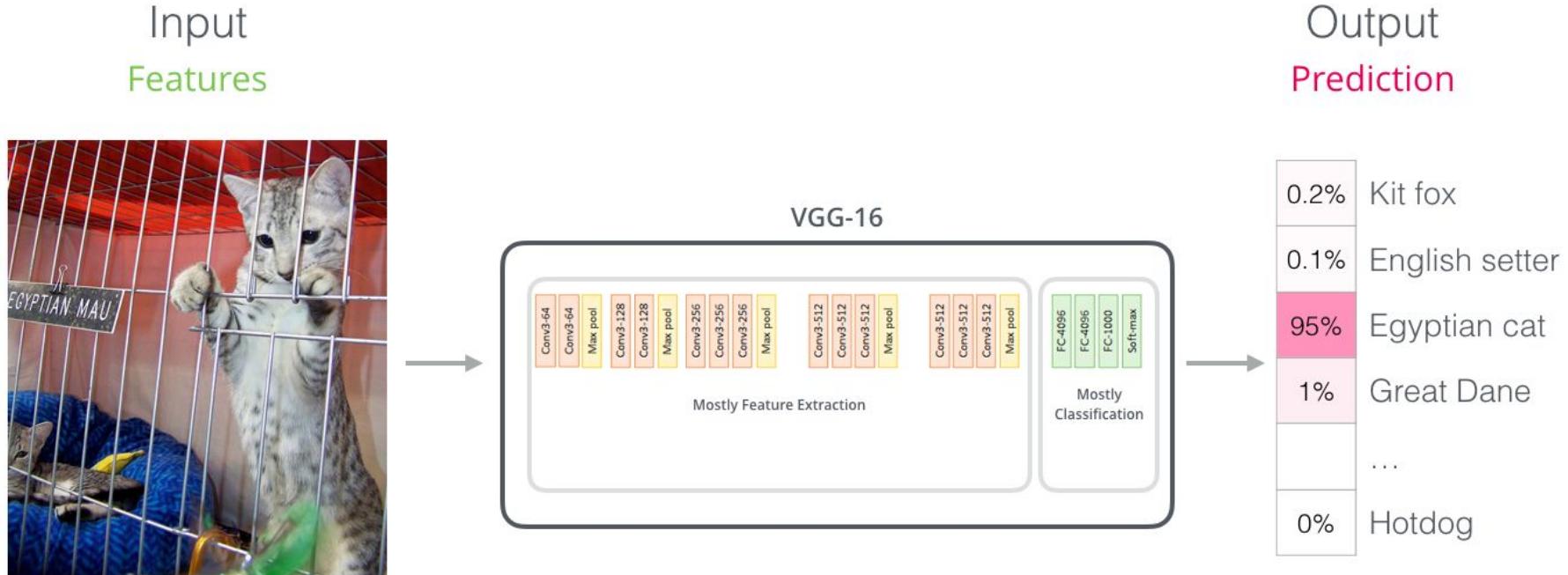
## 2 - Supervised training on a specific task with a labeled dataset.

### Supervised Learning Step



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

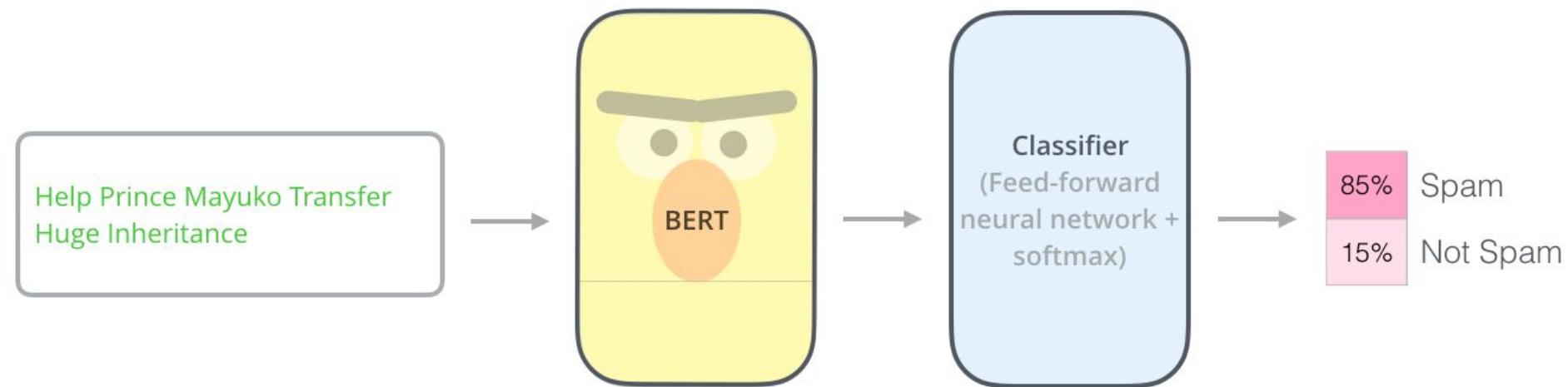
# Similar to CNN concept!



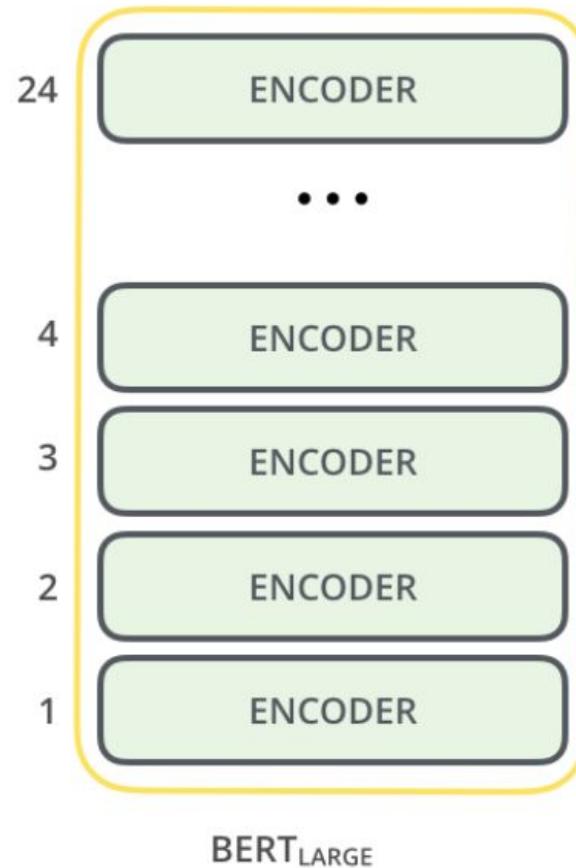
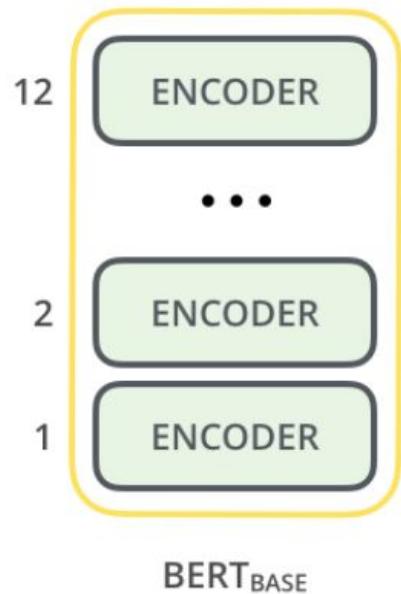
# BERT

Input  
Features

Output  
Prediction



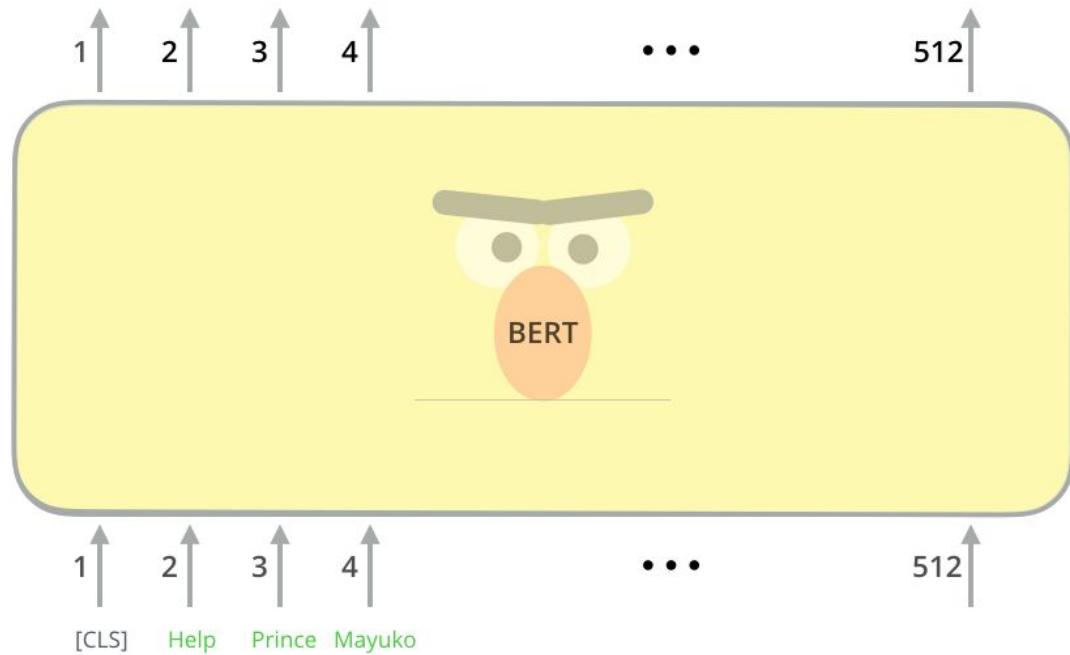
# BERT: base and large



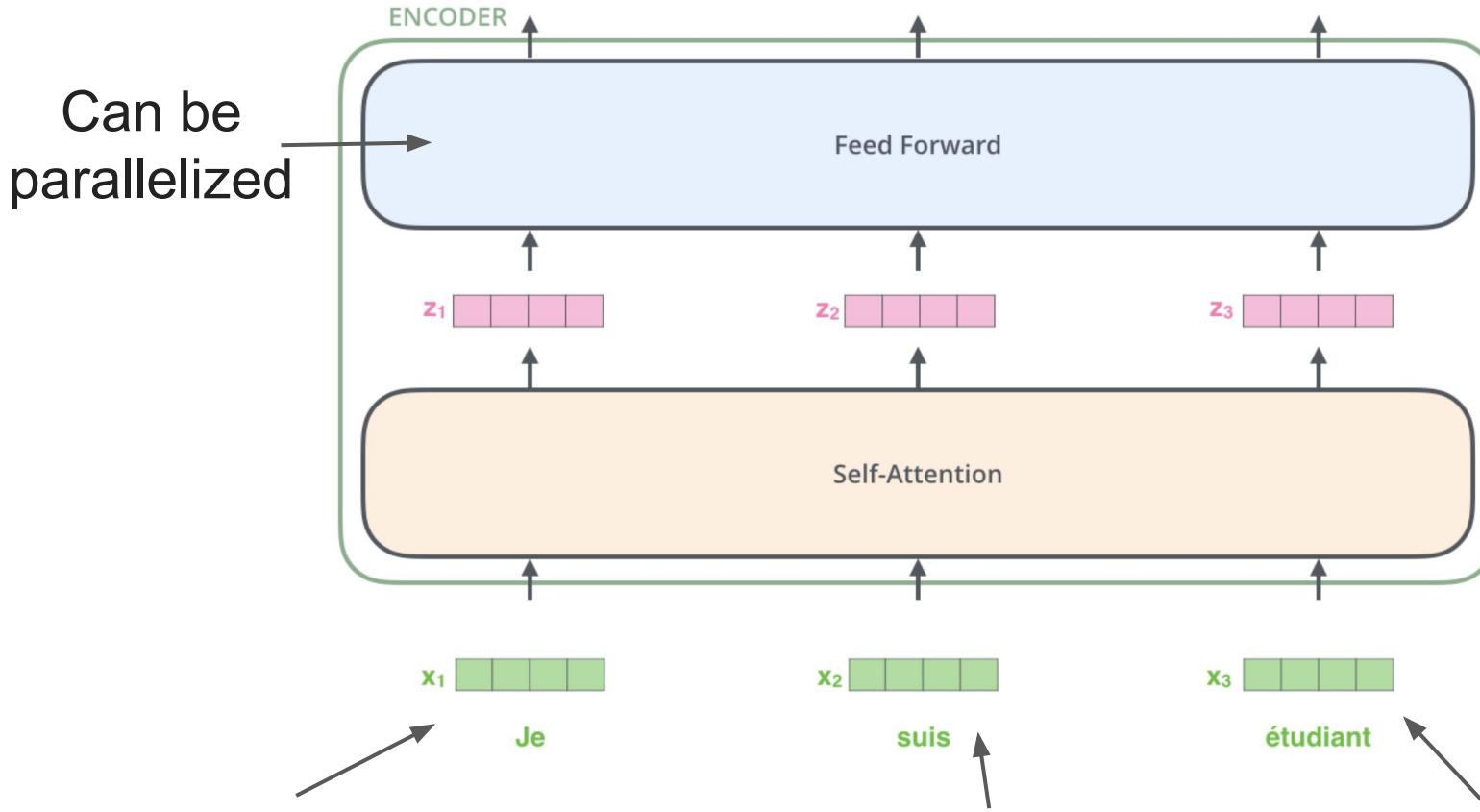
# BERT vs. Transformer

	 THE TRANSFORMER	 BERT	
		<b>Base BERT</b>	<b>Large BERT</b>
Encoders	6	12	24
Units in FFN	512	768	1024
Attention Heads	8	12	16

# Model inputs

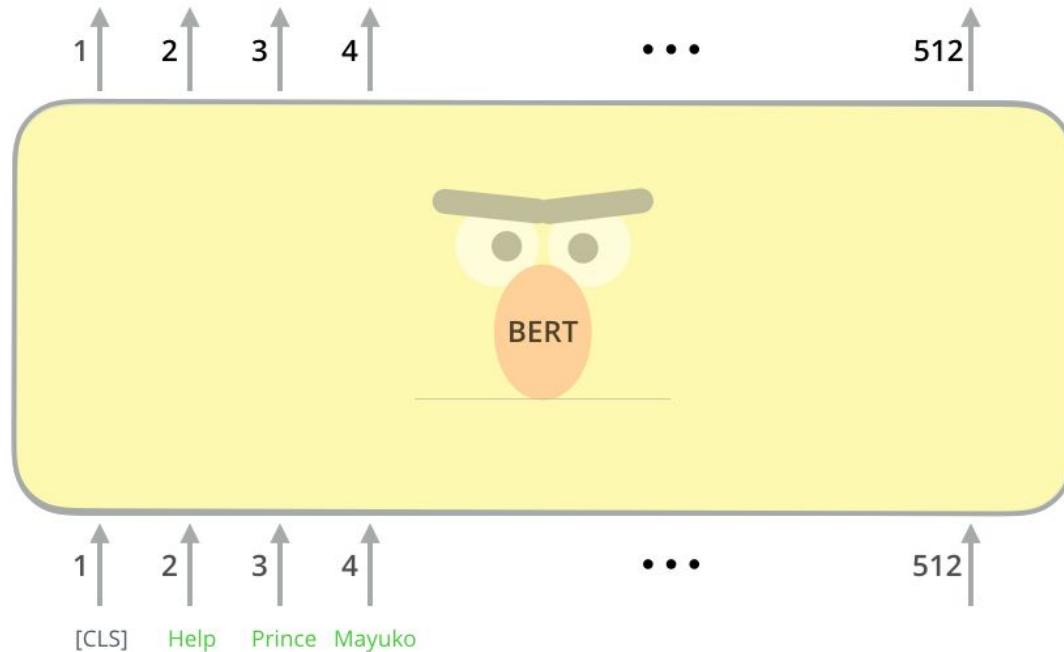


# Transformer Block in BERT



the word in each position flows through its own path in the encoder

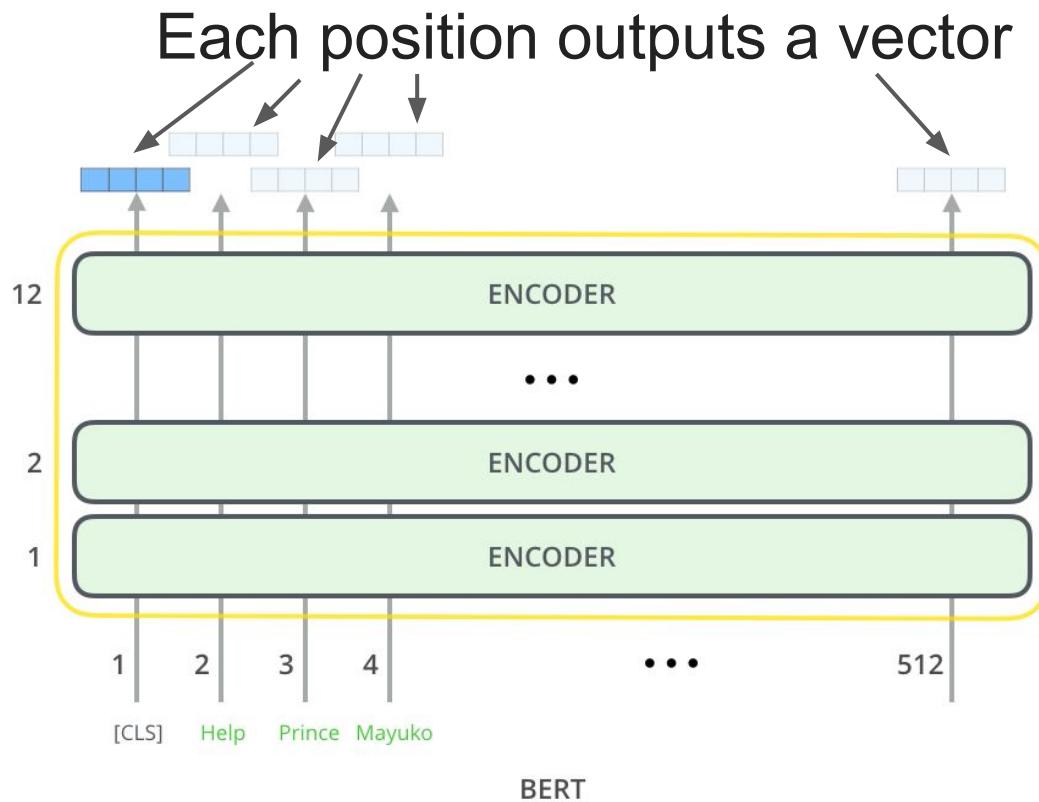
# Model inputs



Identical to the Transformer up until this point

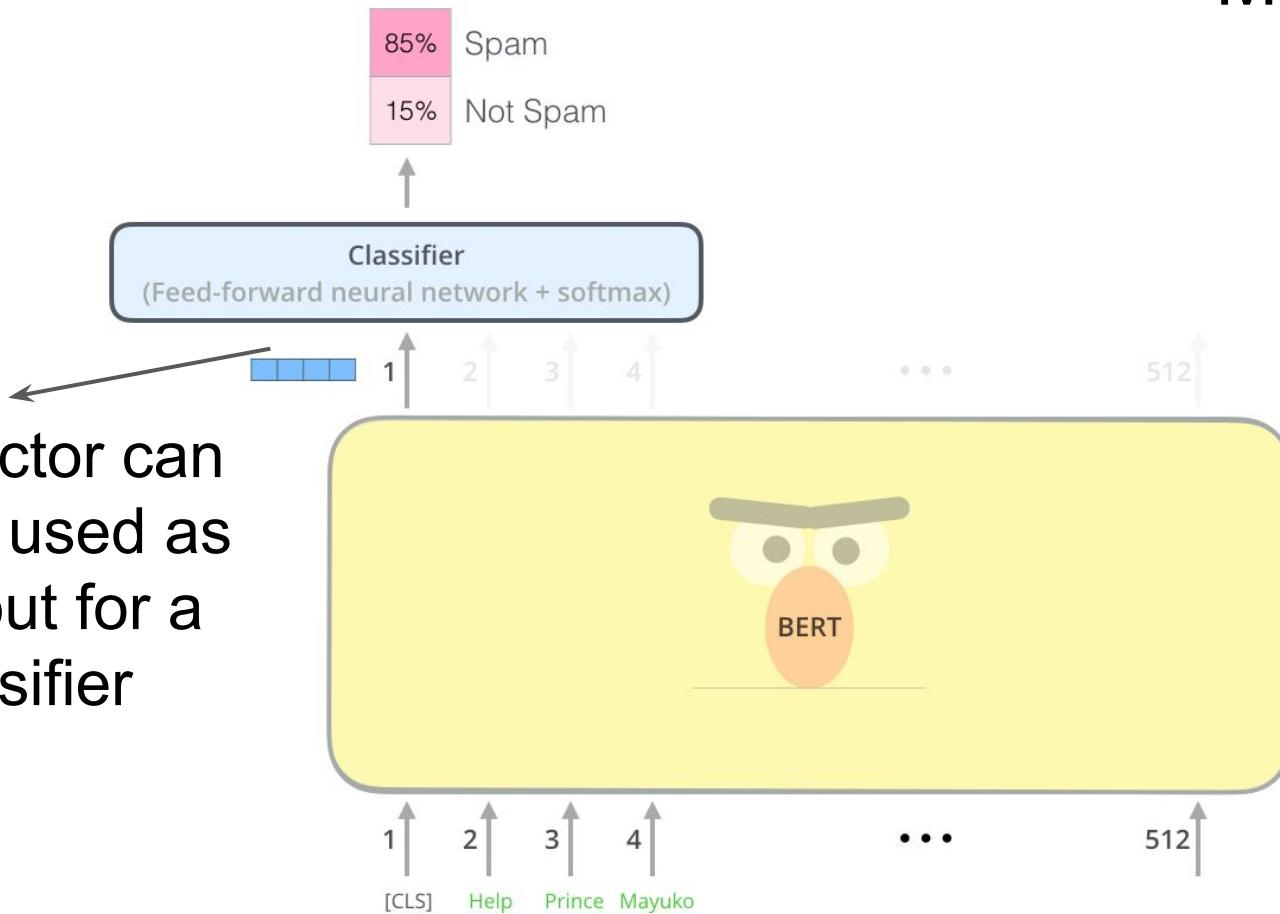
Why is BERT so special?

# Model outputs



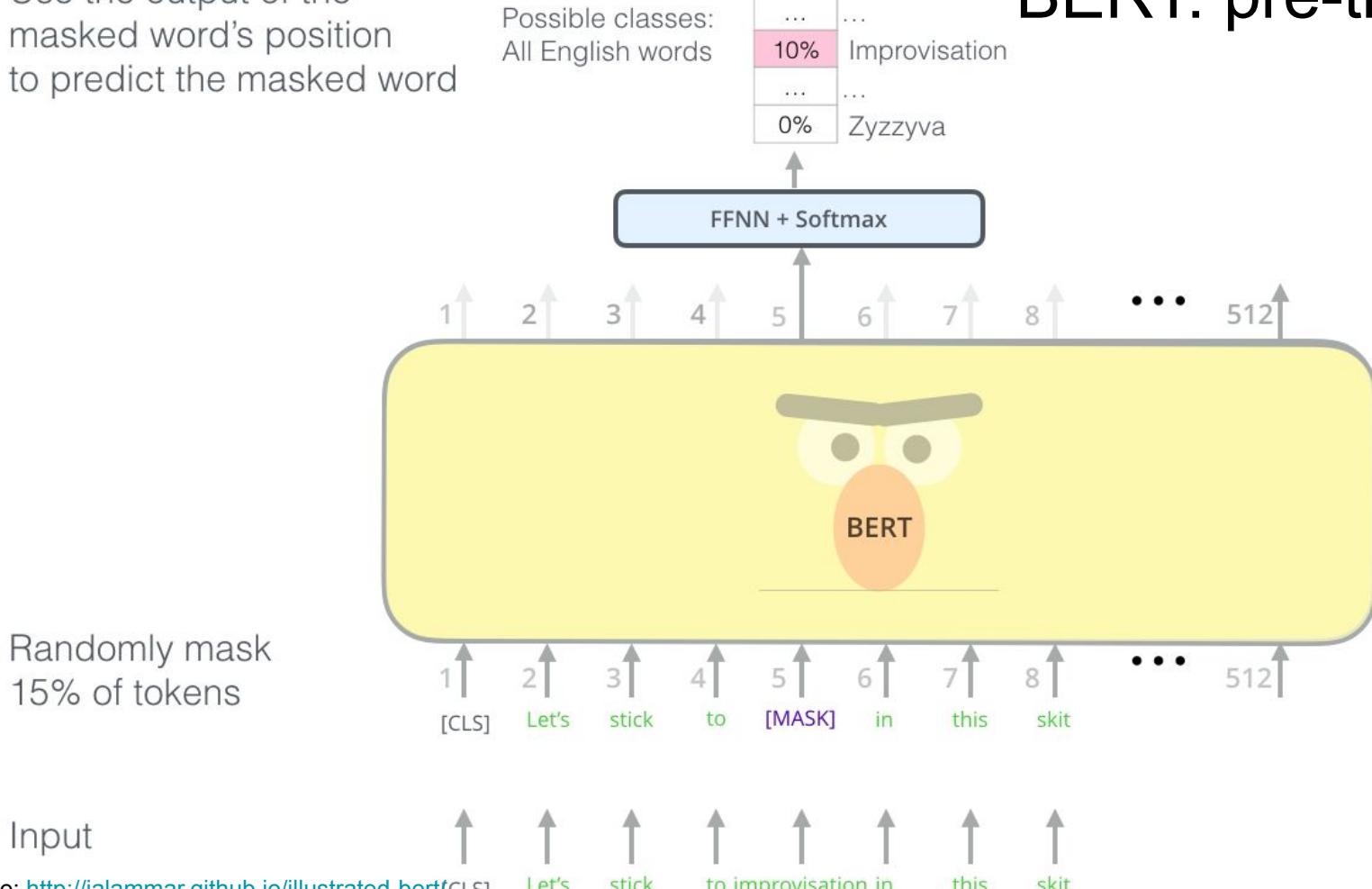
For sentence classification we focus on the first position  
(that we passed [CLS] token to)

# Model inputs



# BERT: pre-training

Use the output of the masked word's position to predict the masked word



Input

# BERT: pre-training

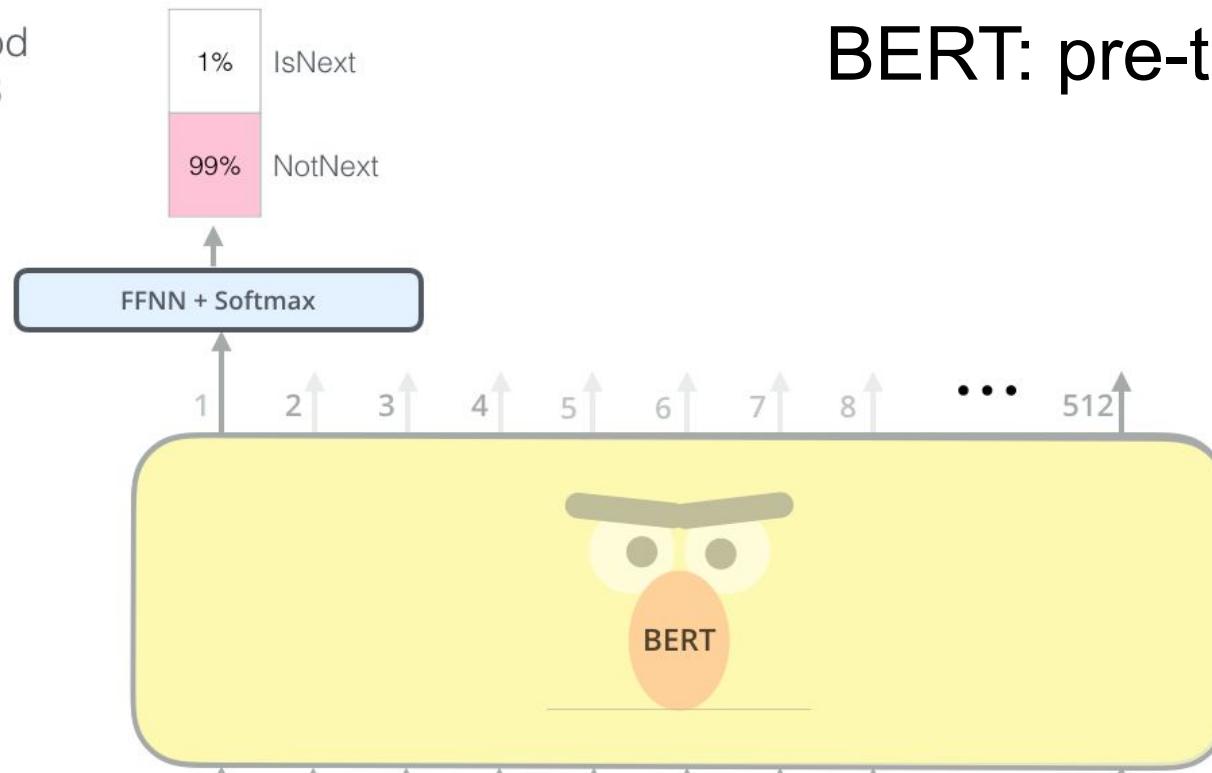
- “Masked Language Model” approach
- To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task:  
*“Given two sentences (A and B), is B likely to be the sentence that follows A, or not?”*

# BERT: pre-training

Predict likelihood  
that sentence B  
belongs after  
sentence A



Tokenized  
Input



Input

[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

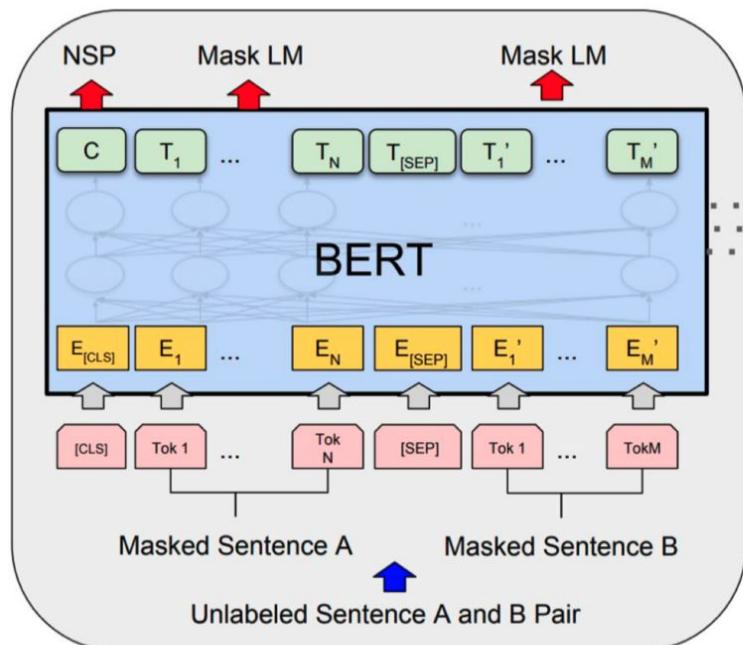
Sentence B

# BERT: input data format

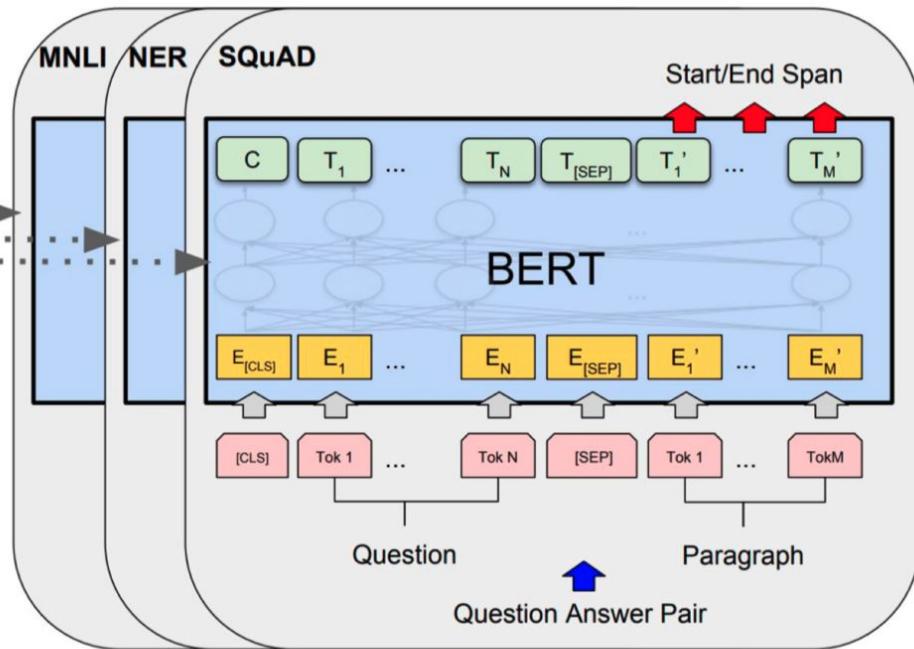
For each tokenized input sentence, we need to create:

- **input ids**: a sequence of integers identifying each input token to its index number in the BERT tokenizer vocabulary
- **segment mask**: a sequence of 1s and 0s used to identify whether the input is one sentence or two sentences long. For one sentence inputs, this is simply a sequence of 0s. For two sentence inputs, there is a 0 for each token of the first sentence, followed by a 1 for each token of the second sentence
- **attention mask**: a sequence of 1s and 0s, with 1s for all input tokens and 0s for all padding tokens

# BERT: fine-tuning for different tasks

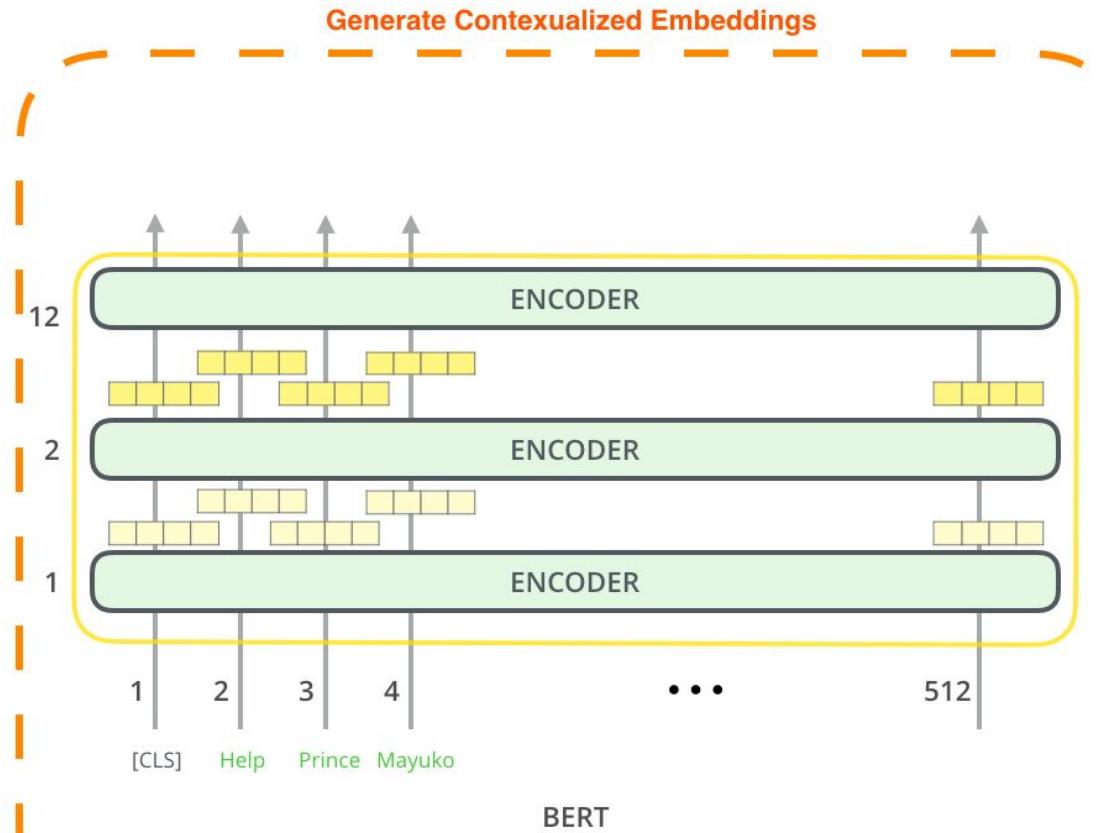


Pre-training

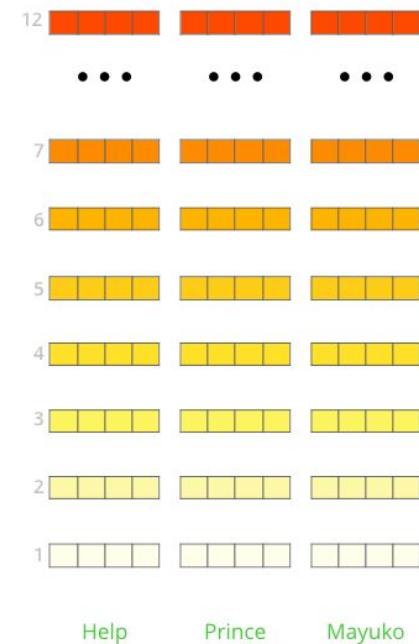


Fine-Tuning

# 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?

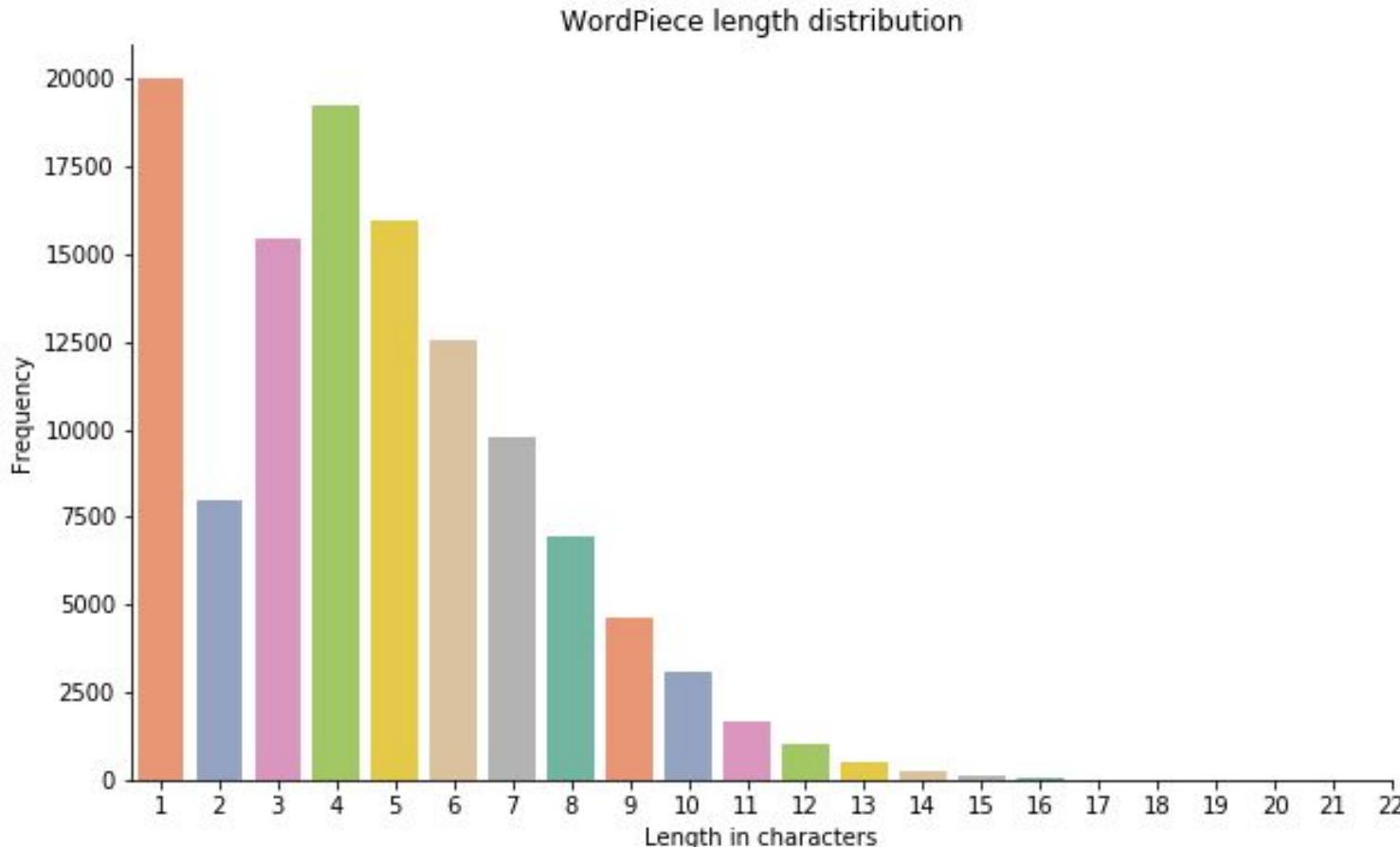
For named-entity recognition task CoNLL-2003 NER

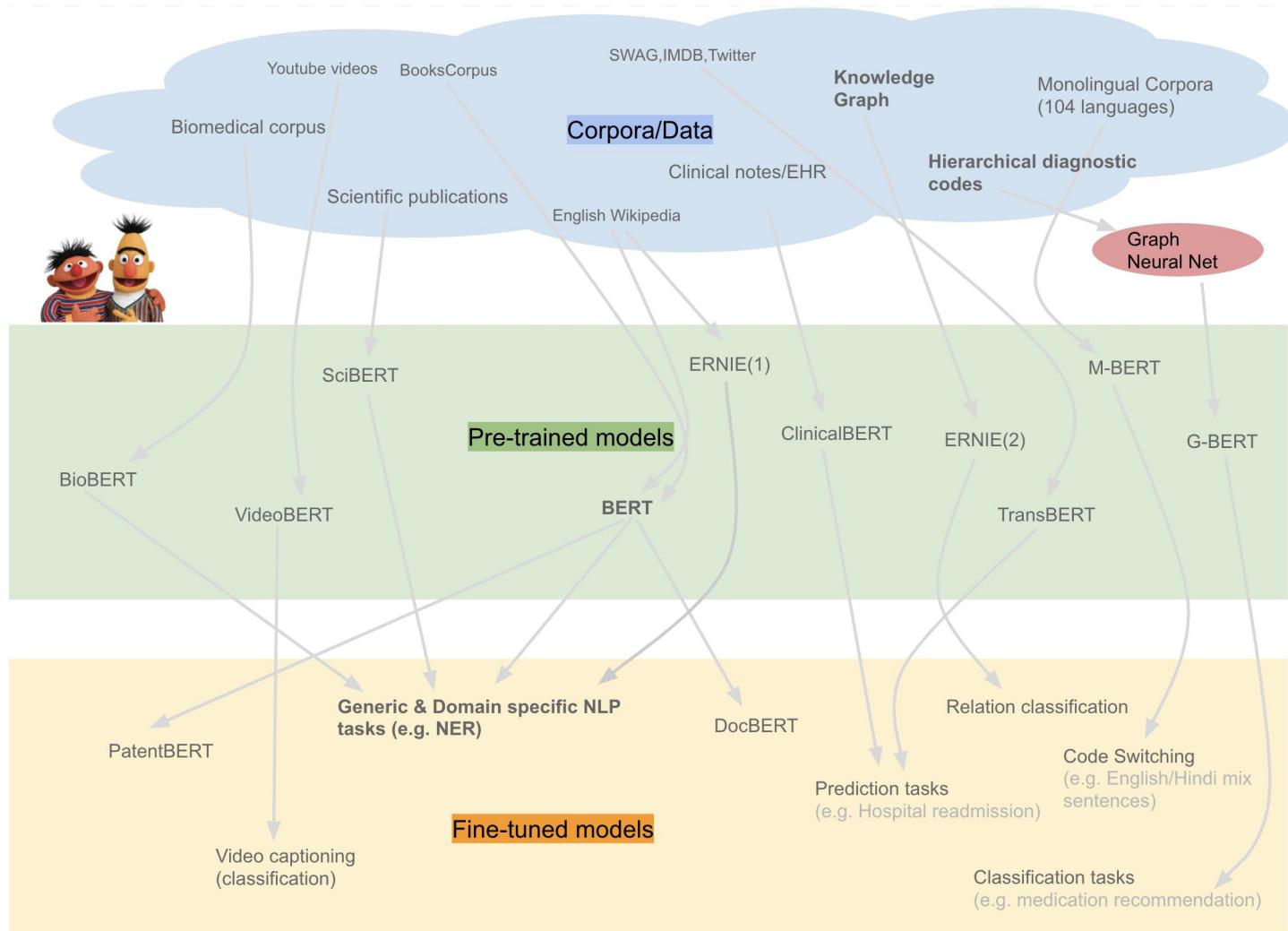
			Dev F1 Score	
12		First Layer	Embedding	91.0
• • •		Last Hidden Layer		94.9
7				
6		Sum All 12 Layers		95.5
5				
4		Second-to-Last Hidden Layer		95.6
3				
2		Sum Last Four Hidden		95.9
1				
		Concat Last Four Hidden		96.1
Help				

## Example: Unaffable -> un, ##aff, ##able

- Single model for 104 languages with a large shared vocabulary (119,547 [WordPiece](#) model)
- Non-word-initial units are prefixed with ##
- The first 106 symbols: constants like PAD and UNK
- 36.5% of the vocabulary are non-initial word pieces
- The alphabet consists of 9,997 unique characters that are defined as word-initial (C) and continuation symbols (##C), which together make up 19,994 word pieces
- The rest are multi character word pieces of various length.

# BERT: tokenization





# BERT: overview

- [BERT repo](#)
- [Try out BERT on TPU](#)
- [WordPieces Tokenizer](#)
- [PyTorch Implementation of BERT](#)

# **GPT-2 & GPT-3**

- Transformer-based architecture
- trained to predict the **next** word
- 1.5 billion parameters
- Trained on 8 million web-pages



On language tasks (question answering, reading comprehension, summarization, translation) works well **WITHOUT** fine-tuning

# GPT-2: question answering

## EXAMPLES

*Who wrote the book the origin of species?*

**Correct answer:** *Charles Darwin*

**Model answer:** Charles Darwin

*What is the largest state in the U.S. by land mass?*

**Correct answer:** *Alaska*

**Model answer:** California

# GPT-2: language modeling

## EXAMPLE

*Both its sun-speckled shade and the cool grass beneath were a welcome respite after the stifling kitchen, and I was glad to relax against the tree's rough, brittle bark and begin my breakfast of buttery, toasted bread and fresh fruit. Even the water was tasty, it was so clean and cold. It almost made up for the lack of...*

**Correct answer:** coffee

**Model answer:** food

# GPT-2: machine translation

## EXAMPLE

### French sentence:

*Un homme a expliqué que l'opération gratuite qu'il avait subie pour soigner une hernie lui permettrait de travailler à nouveau.*

### Reference translation:

*One man explained that the free hernia surgery he'd received will allow him to work again.*

### Model translation:

A man told me that the operation gratuity he had been promised would not allow him to travel.

## New AI fake text generator may be too dangerous to ... - The Guardian

<https://www.theguardian.com/.../elon-musk-backed-ai-writes-convincing-news-fiction>

4 days ago - The Elon Musk-backed nonprofit company OpenAI declines to release research publicly for fear of misuse. The creators of a revolutionary AI system that can write news stories and works of fiction – dubbed "deepfakes for text" – have taken the unusual step of not releasing ...

## OpenAI built a text generator so good, it's considered too dangerous to ...

[https://techcrunch.com/2019/02/17/openai-text-generator-dangerous/ ▾](https://techcrunch.com/2019/02/17/openai-text-generator-dangerous/)

12 hours ago - A storm is brewing over a new language model, built by non-profit artificial intelligence research company OpenAI, which it says is so good at ...

## The AI Text Generator That's Too Dangerous to Make Public | WIRED

[https://www.wired.com/story/ai-text-generator-too-dangerous-to-make-public/ ▾](https://www.wired.com/story/ai-text-generator-too-dangerous-to-make-public/)

4 days ago - In 2015, car-and-rocket man Elon Musk joined with influential startup backer Sam Altman to put artificial intelligence on a new, more open ...

## Elon Musk-backed AI Company Claims It Made a Text Generator ...

[https://gizmodo.com/elon-musk-backed-ai-company-claims-it-made-a-text-gener-183... ▾](https://gizmodo.com/elon-musk-backed-ai-company-claims-it-made-a-text-gener-183...)

Elon Musk-backed AI Company Claims It Made a Text Generator That's Too Dangerous to Release · Rhett Jones · Friday 12:15pm · Filed to: OpenAI Filed to: ...

## Scientists have made an AI that they think is too dangerous to ...

[https://www.weforum.org/.../amazing-new-ai-churns-out-coherent-paragraphs-of-text/ ▾](https://www.weforum.org/.../amazing-new-ai-churns-out-coherent-paragraphs-of-text/)

3 days ago - Sample outputs suggest that the AI system is an extraordinary step forward, producing text rich with context, nuance and even something ...

## New AI Fake Text Generator May Be Too Dangerous To ... - Slashdot

[https://news.slashdot.org/.../new-ai-fake-text-generator-may-be-too-dangerous-to-rele... ▾](https://news.slashdot.org/.../new-ai-fake-text-generator-may-be-too-dangerous-to-rele...)

3 days ago - An anonymous reader shares a report: The creators of a revolutionary AI system that can write news stories and works of fiction – dubbed ...

# GPT-2: fake news and hype

## Top stories



OpenAI built a text generator so good, it's considered too dangerous to release

TechCrunch

11 hours ago



Elon Musk's AI company created a fake news generator it's too scared to make public

BGR.com

9 hours ago



The AI That Can Write A Fake News Story From A Handful Of Words

NDTV.com

2 hours ago

## When Is Technology Too Dangerous to Release to the Public?

Slate · 2 days ago



## Scientists Developed an AI So Advanced They Say It's Too Dangerous to Release

ScienceAlert · 6 days ago



# Different models (2019)

number of parameters, millions

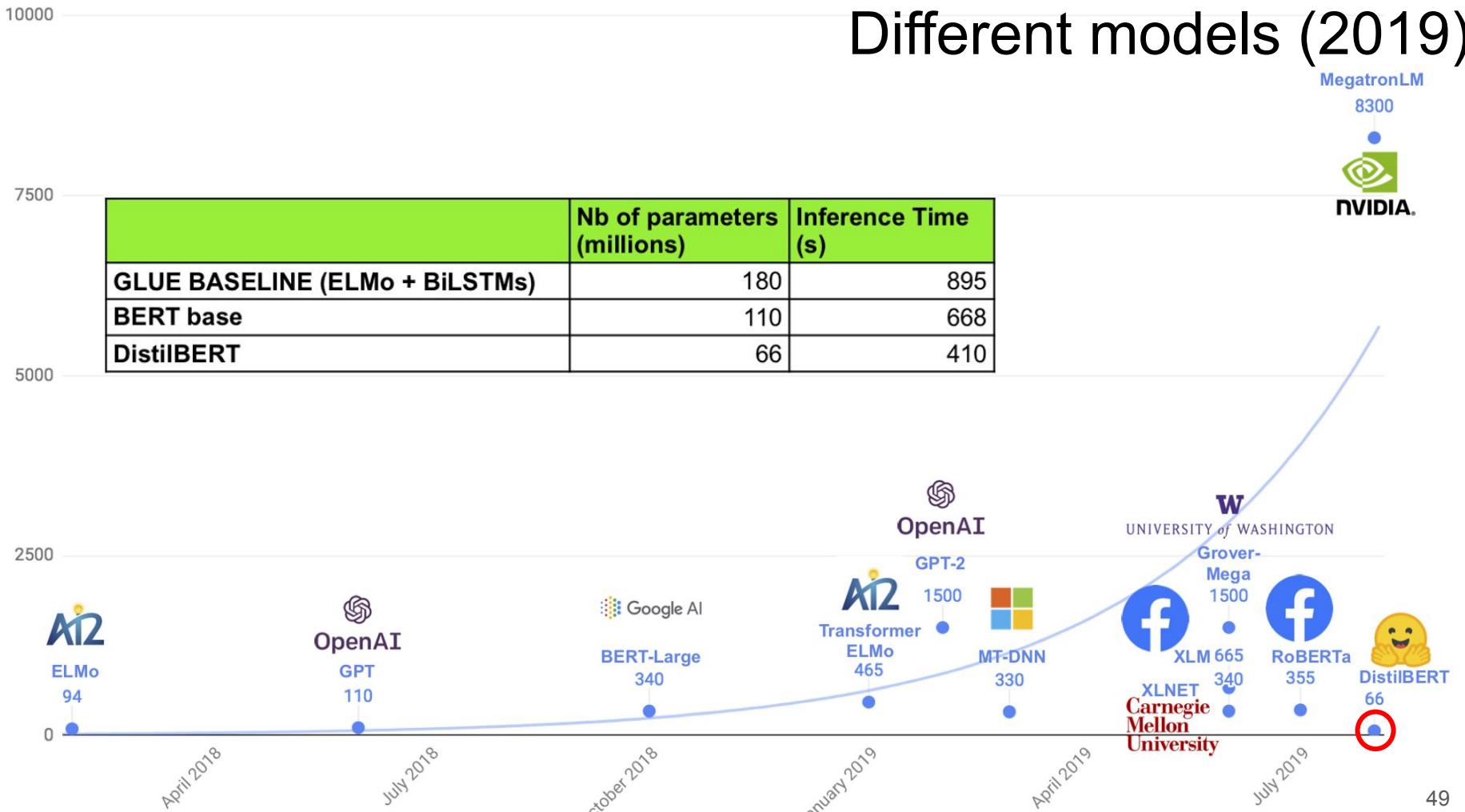
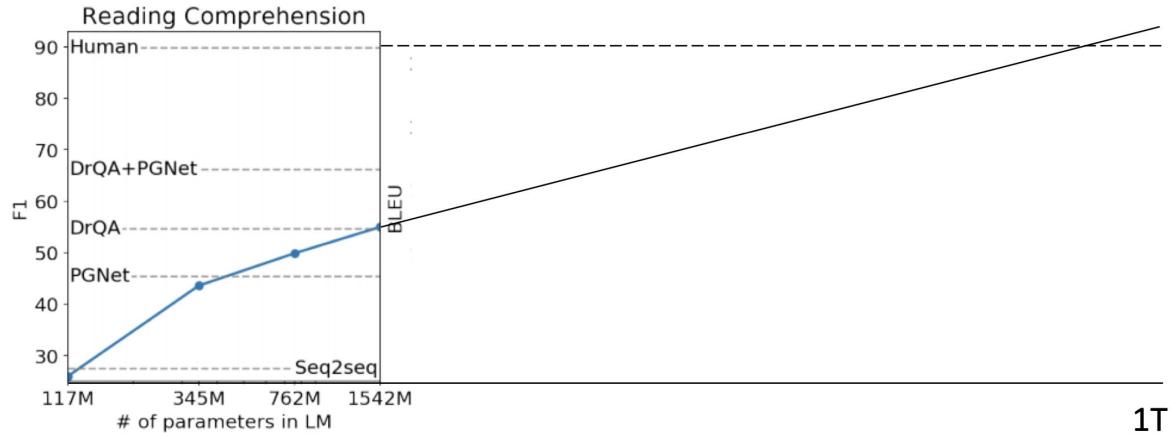
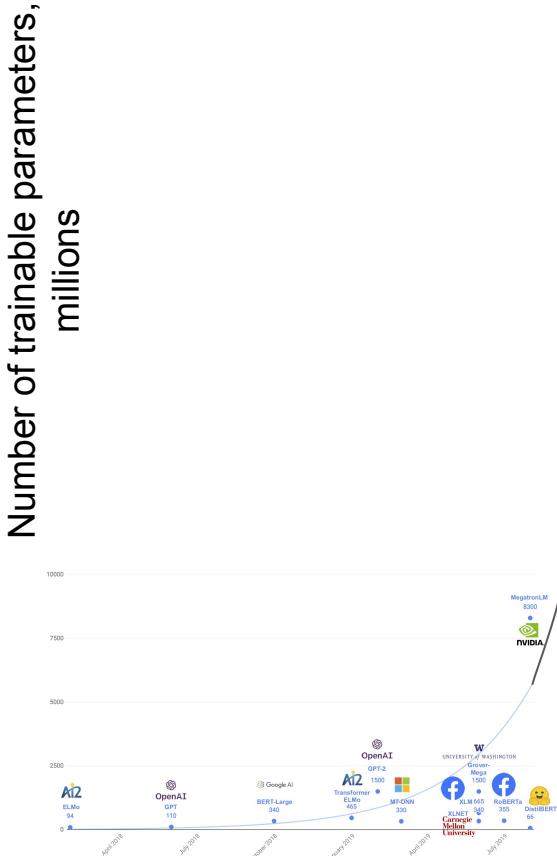


Image source: [Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT](#)

# New achievements: GPT-3

GPT-3, May 2020

Proportions are not preserved for visual sake



Hypothesis from Stanford CS224N Lecture 20 (2019)

- GPT-2: 1.5 billion parameters
- GPT-3: **175 billion** parameters



**Geoffrey Hinton** @geoffreyhinton · Jun 10

Extrapolating the spectacular performance of GPT3 into the future suggests that the answer to life, the universe and everything is just 4.398 trillion parameters.

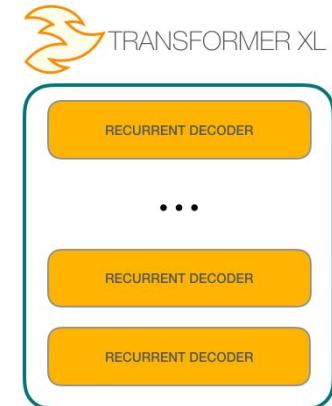
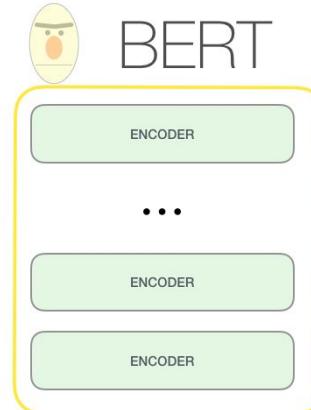
62

643

3.4K



- Transformer
- OpenAI Transformer
- ELMO
- BERT
- BERTology
- GPT
- GPT-2
- GPT-3



- Transfer learning caused a giant leap in Computer Vision and Natural Language Processing
  - Which domain is next?
- Using pre-trained BERT might be a good idea in many tasks
  - Or even using DistillBERT
- Better problem statement leads to better results
  - What information is hidden within the data and can be retrieved?

# Backup

# **ULM-fit: Universal Language Model Fine-tuning for Text Classification**

# ULMFiT: architecture

- Encoder: AWD-LSTM (ASGD  
Weight-Dropped LSTM)

```
SequentialRNN(  
    (0): MultiBatchEncoder(  
        (module): AWD_LSTM(  
            (encoder): Embedding(60003, 300, padding_idx=1)  
            (encoder_dp): EmbeddingDropout(  
                (emb): Embedding(60003, 300, padding_idx=1)  
            )  
            (rnns): ModuleList(  
                (0): WeightDropout(  
                    (module): LSTM(300, 1150, batch_first=True)  
                )  
                (1): WeightDropout(  
                    (module): LSTM(1150, 1150, batch_first=True)  
                )  
                (2): WeightDropout(  
                    (module): LSTM(1150, 300, batch_first=True)  
                )  
            )  
            (input_dp): RNNDropout()  
            (hidden_dps): ModuleList(  
                (0): RNNDropout()  
                (1): RNNDropout()  
                (2): RNNDropout()  
            )  
        )  
    )  
    (1): PoolingLinearClassifier(  
        (layers): Sequential(  
            (0): BatchNorm1d(900, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
            (1): Dropout(p=0.4)  
            (2): Linear(in_features=900, out_features=50, bias=True)  
            (3): ReLU(inplace)  
            (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
            (5): Dropout(p=0.1)  
            (6): Linear(in_features=50, out_features=2, bias=True)  
        )  
    )  
)
```

What's the main difference from all the other Sesame street models?

What's the main difference from all the other Sesame street models?



# ULMFiT: architecture

- Encoder: AWD-LSTM (ASGD Weight-Dropped LSTM)
- AWD-LSTM literally has dropout at all the possible layers as long as it makes sense.

```
SequentialRNN(  
    (0): MultiBatchEncoder(  
        (module): AWD_LSTM(  
            (encoder): Embedding(60003, 300, padding_idx=1)  
            (encoder_dp): EmbeddingDropout(  
                (emb): Embedding(60003, 300, padding_idx=1)  
            )  
            (rnns): ModuleList(  
                (0): WeightDropout(  
                    (module): LSTM(300, 1150, batch_first=True)  
                )  
                (1): WeightDropout(  
                    (module): LSTM(1150, 1150, batch_first=True)  
                )  
                (2): WeightDropout(  
                    (module): LSTM(1150, 300, batch_first=True)  
                )  
            )  
            (input_dp): RNNDropout()  
            (hidden_dps): ModuleList(  
                (0): RNNDropout()  
                (1): RNNDropout()  
                (2): RNNDropout()  
            )  
        )  
    )  
    (1): PoolingLinearClassifier(  
        (layers): Sequential(  
            (0): BatchNorm1d(900, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
            (1): Dropout(p=0.4)  
            (2): Linear(in_features=900, out_features=50, bias=True)  
            (3): ReLU(inplace)  
            (4): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True,  
track_running_stats=True)  
            (5): Dropout(p=0.1)  
            (6): Linear(in_features=50, out_features=2, bias=True)  
        )  
    )  
)
```

# ULMFiT: encoder dropout

1. **Encoder Dropout (*EmbeddingDropout()*):** Zeroing embedding vectors randomly.

ENCODER DROPOUT				
	before dropout			after dropout
token	d1	d2	...	token
I	0.399	0.75379	0.62616	I
love	0.88533	0.29449	0.15856	love
cats	0.48927	0.04071	0.21427	cats
dogs	0.72918	0.86882	0.77136	dogs

# ULMFiT: input dropout

2. Input Dropout (`RNNDropout()`): Zeroing embedding lookup outputs randomly.

INPUT DROPOUT				
batch: [I love cats, I love dogs]				
				<b>before dropout</b>
I	0	0	0	
love	0.88533	0.29449	0.15856	
cats	0.48927	0.04071	0.21427	
				<b>after dropout</b>
I	0	0	0	
love	0	0.29449	0.15856	
cats	0	0.04071	0.21427	
				<b>before dropout</b>
I	0	0	0	
love	0.88533	0.29449	0.15856	
dogs	0.72918	0.86882	0.77136	
				<b>after dropout</b>
I	0	0	0	
love	0.88533	0.29449	0	
dogs	0.72918	0.86882	0	

# ULMFiT: dropout

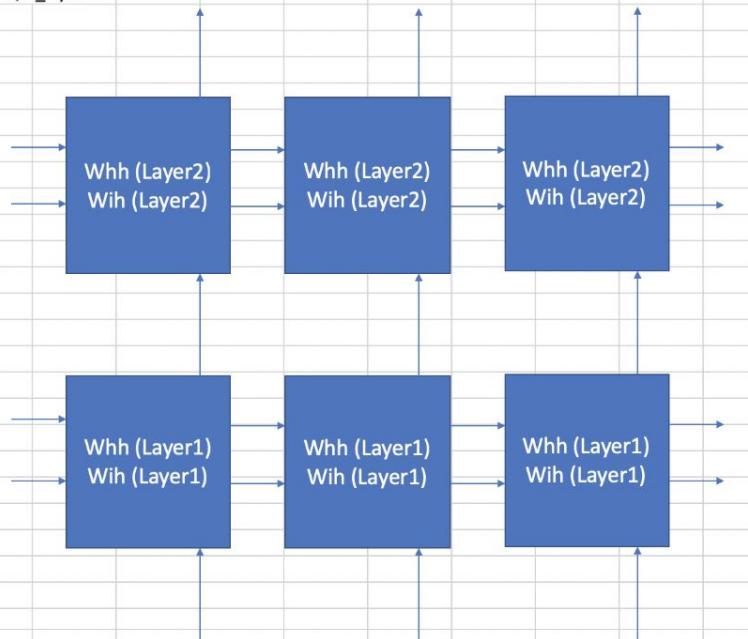
### 3. Weight Dropout(*WeightDropout()*) : Apply dropout to LSTM weights.

Random dropout at hidden-to-hidden weights for each LSTM layer

```
rnn = nn.LSTM(5,10,1, bidirectional=False, batch_first=True)  
WeightDropout(rnn, weight_p=0.5, layer_names=['weight_hh_10'])
```

# ULMFiT: weight dropout

WEIGHT DROPOUT  
 $n_{hid}=5, n_{layers}=2$



I	I
0	0
0	0
0	0

love	love
0	0.88533
0.29449	0.29449
0.15856	0

cats	dogs
0	0.72918
0.04071	0.86882
0.21427	0

before dropout

Whh (Layer1)				
0.30546	0.94294	0.66715	0.5655	0.31191
0.06246	0.36674	0.35672	0.53357	0.04652
0.84278	0.17049	0.92283	0.97827	0.67913
0.94699	0.02277	0.79537	0.09479	0.16284
0.34098	0.09256	0.69661	0.22605	0.57153
0.09455	0.78172	0.48226	0.82222	0.25151
0.92516	0.50419	0.82879	0.67049	0.84304
0.11741	0.09362	0.69367	0.5633	0.34755
0.98025	0.73131	0.21491	0.86319	0.31456
0.0289	0.90899	0.30697	0.75105	0.62107
0.96808	0.4712	0.77992	0.83567	0.09754
0.15539	0.94866	0.51977	0.51167	0.08723
0.10944	0.88881	0.74007	0.96823	0.03516
0.62014	0.4363	0.02097	0.61126	0.30466
0.10465	0.02462	0.2334	0.25372	0.54579
0.14384	0.962	0.70157	0.39285	0.41064
0.42081	0.64236	0.06941	0.41805	0.78772
0.45229	0.9871	0.49831	0.17108	0.48868
0.42142	0.66105	0.53273	0.30901	0.45095
0.33421	0.77472	0.33966	0.42693	0.4567

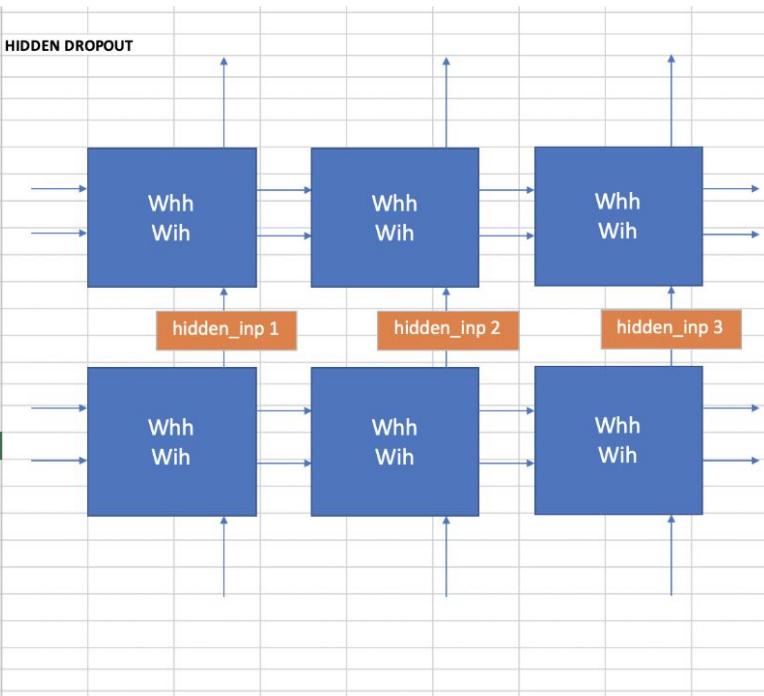
after dropout

Whh (Layer1)				
0.30546	0	0.66715	0.5655	0
0	0.36674	0.35672	0.53357	0.04652
0.84278	0.17049	0.92283	0.97827	0.67913
0.94699	0.02277	0.79537	0.09479	0.16284
0.34098	0.09256	0.69661	0.22605	0.57153
0	0.78172	0.48226	0.82222	0.25151
0	0.50419	0	0.67049	0.84304
0	0.09362	0.69367	0.5633	0
0.98025	0.73131	0	0	0.31456
0	0	0.30697	0.75105	0.62107
0	0	0.77992	0.83567	0.09754
0.15539	0	0.51977	0.51167	0.08723
0.10944	0.88881	0.74007	0.96823	0.03516
0.62014	0.4363	0.02097	0.61126	0.30466
0.10465	0.02462	0.2334	0	0.54579
0	0.962	0.70157	0	0.41064
0	0	0.06941	0.41805	0.78772
0.45229	0.9871	0.49831	0	0.48868
0.42142	0.66105	0.53273	0.30901	0.45095
0	0.77472	0.33966	0.42693	0.4567

same word, different vectors

# ULMFiT: hidden dropout

**4. Hidden Dropout (`RNNDropout()`):** Zeroing outputs of LSTM layers. This dropout is applied except for the last LSTM layer output.



before dropout										
I	0.11587	0.89354	0.18537	0.13159	0.04328	0.36997	0.68241	0.21505	0.07696	0.1405
love	0.51854	0.94156	0.80541	0.49411	0.44335	0.31453	0.20678	0.95815	0.70224	0.2766
cats	0.41255	0.70882	0.76667	0.93813	0.89034	0.1676	0.92944	0.32216	0.11675	0.99425

after dropout										
hidden_inp 1	I	0.11587	0	0.18537	0.13159	0	0.36997	0.68241	0	0
hidden_inp 2	love	0.51854	0	0.80541	0.49411	0	0.31453	0.20678	0	0
hidden_inp 3	cats	0.41255	0	0.76667	0.93813	0	0.1676	0.92944	0	0

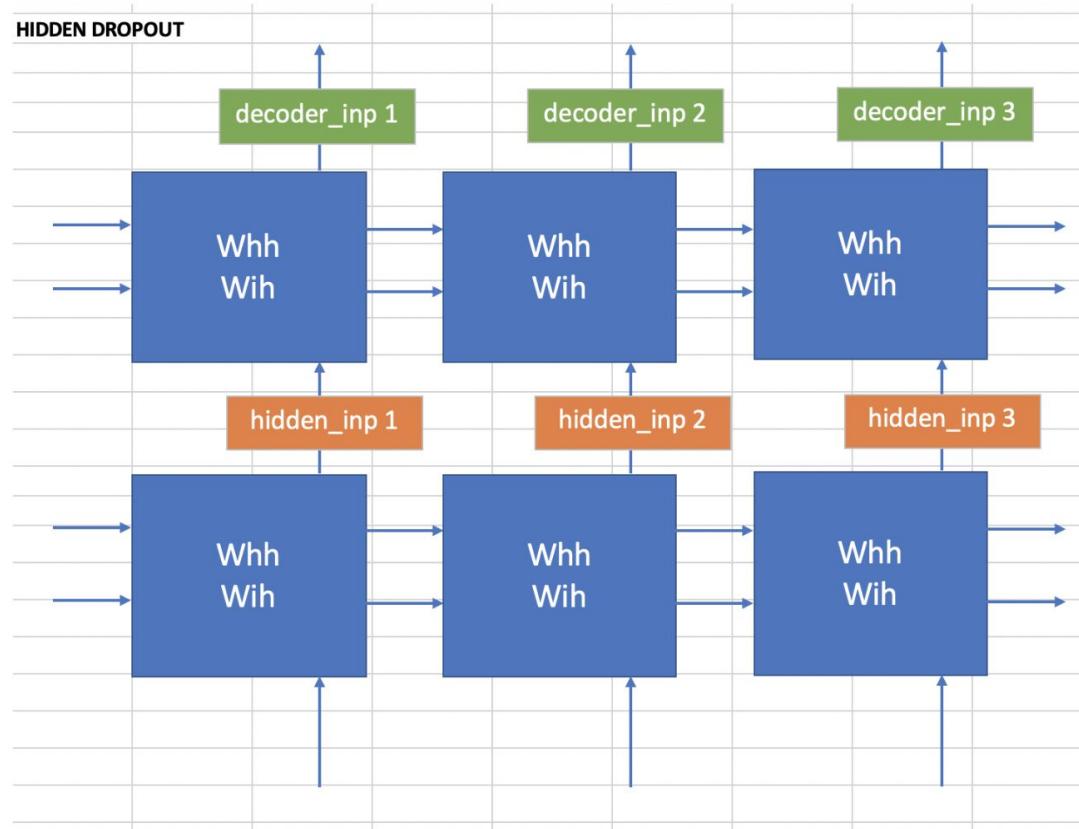
before dropout										
I	0.90644	0.01673	0.38366	0.86887	0.26845	0.332	0.93804	0.34769	0.83711	0.66387
love	0.82539	0.09143	0.32444	0.35099	0.64491	0.79199	0.24797	0.56466	0.46948	0.33358
dogs	0.38805	0.56548	0.1424	0.31113	0.11645	0.36759	0.73068	0.05727	0.7315	0.01498

after dropout										
hidden_inp 1	I	0	0	0.38366	0.86887	0	0.332	0	0.34769	0.83711
hidden_inp 2	love	0	0	0.32444	0.35099	0	0.79199	0	0.56466	0.46948
hidden_inp 3	dogs	0	0	0.1424	0.31113	0	0.36759	0	0.05727	0.7315

# ULMFiT: output dropout

## 5. Output Dropout

(*RNNDropout()*): Zeroing final sequence outputs from encoder before feeding it to decoder.



# Variable Length BPTT

Fixed window to back-propagate will always have the same words contributing for the update with same weight of gradients flowing from last word to the first

Let's randomize windows selected at each step!

More about BPTT: <https://machinelearningmastery.com/gentle-introduction-backpropagation-time/>

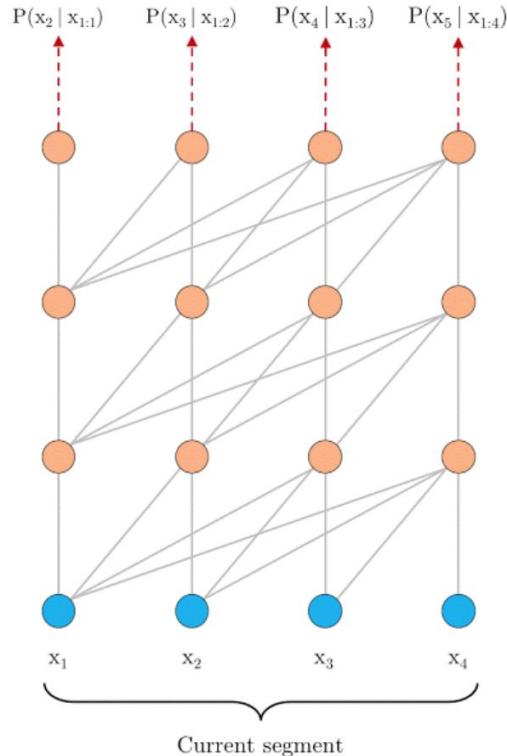
# Transformer-XL

- Vanilla Transformer works with a fixed-length context at training time. That's why:
  - the algorithm is not able to model dependencies that are longer than a fixed length.
  - the segments usually do not respect the sentence boundaries, resulting in context fragmentation which leads to inefficient optimization.

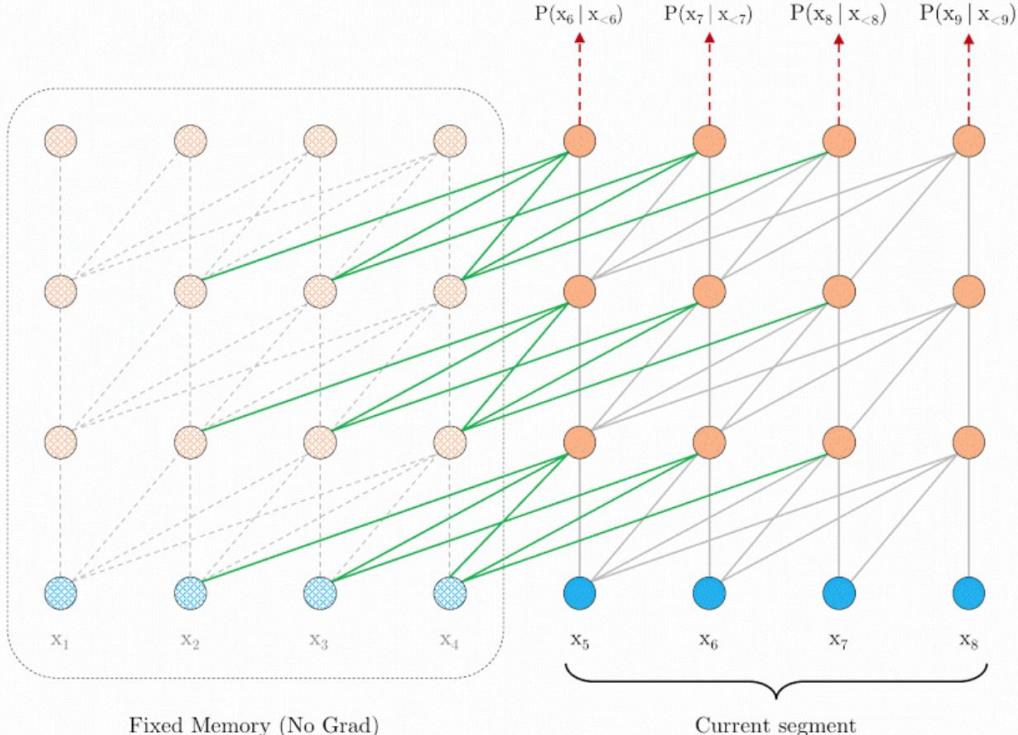
# Segment-level Recurrence

- During training, the representations computed for the previous segment are fixed and cached to be reused as an extended context when the model processes the next new segment.
- Contextual information is now able to flow across segment boundaries.
- Recurrence mechanism also resolves the context fragmentation issue, providing necessary context for tokens in the front of a new segment.

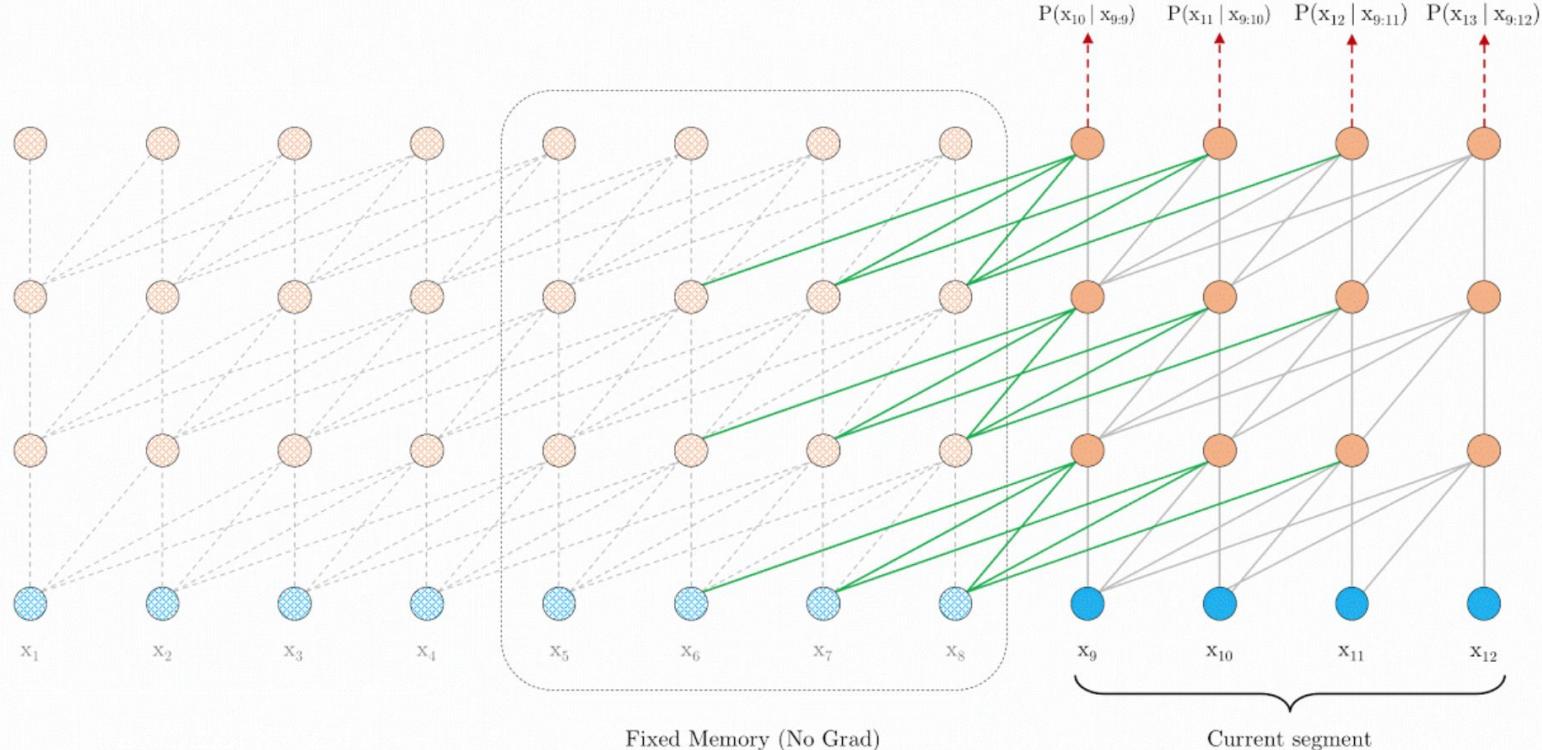
# Segment-level Recurrence



# Segment-level Recurrence



# Segment-level Recurrence



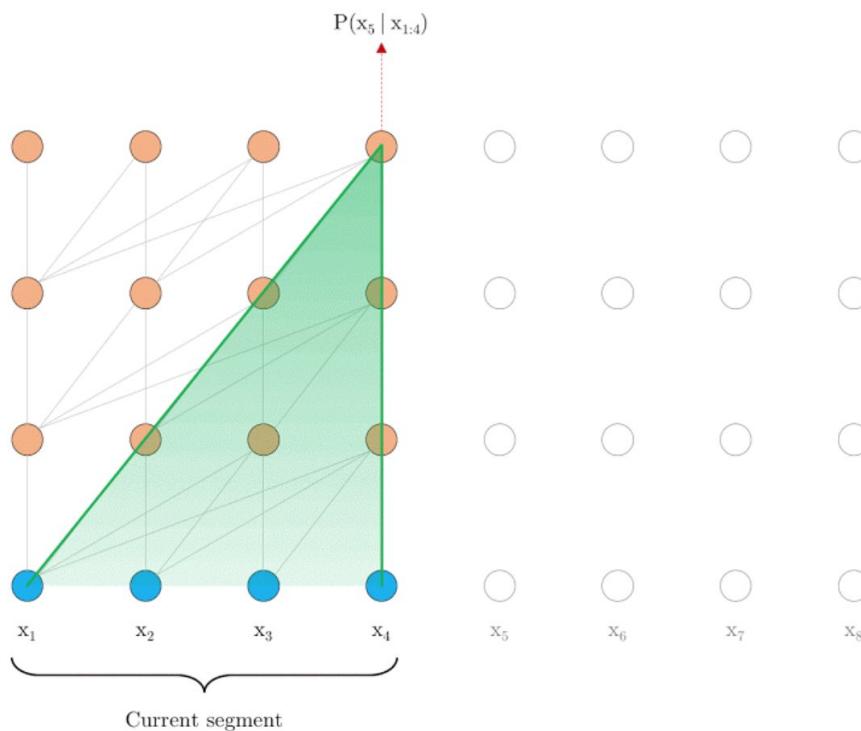
# Relative Positional Encodings

- Fixed embeddings with learnable transformations instead of learnable embeddings

As a result:

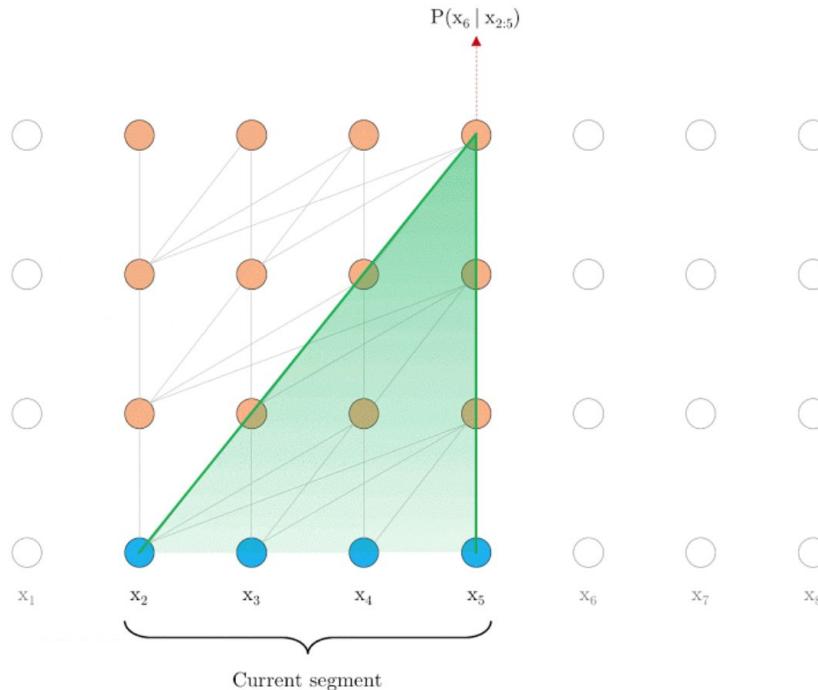
- more generalizable to longer sequences at test time
- longer effective context

# Vanilla Transformer vs. Transformer-XL



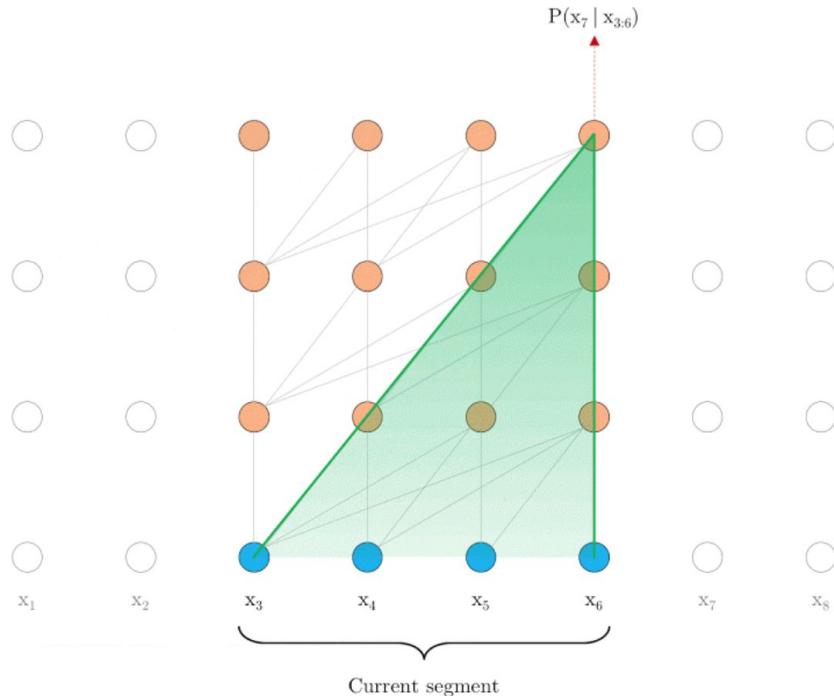
Vanilla Transformer with a fixed-length context at evaluation time

# Vanilla Transformer vs. Transformer-XL



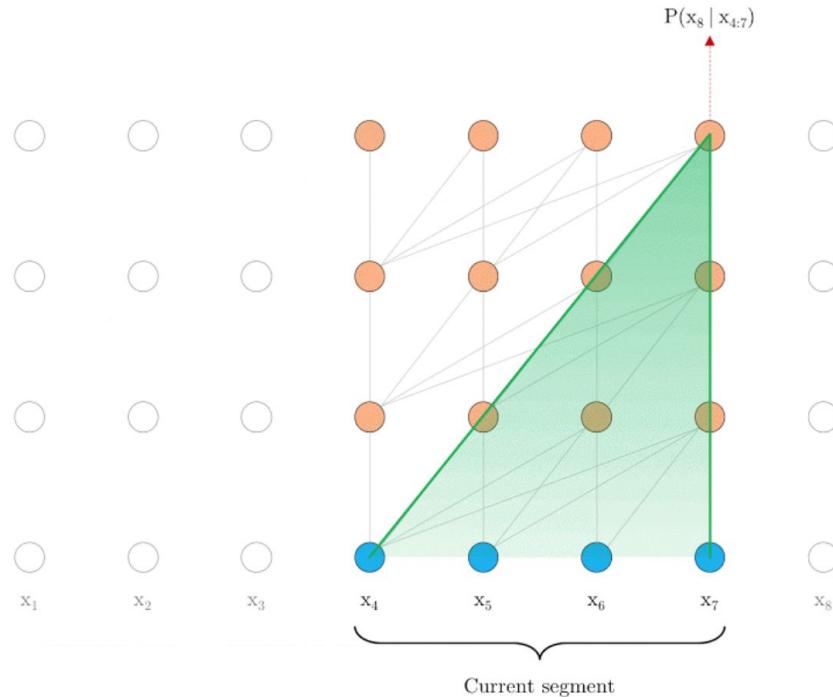
Vanilla Transformer with a fixed-length context at evaluation time

# Vanilla Transformer vs. Transformer-XL



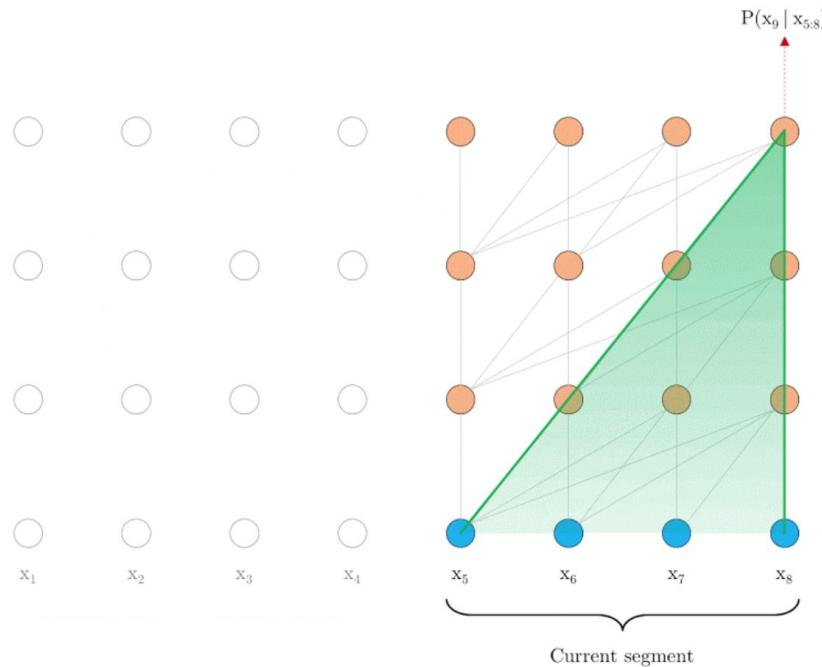
Vanilla Transformer with a fixed-length context at evaluation time

# Vanilla Transformer vs. Transformer-XL



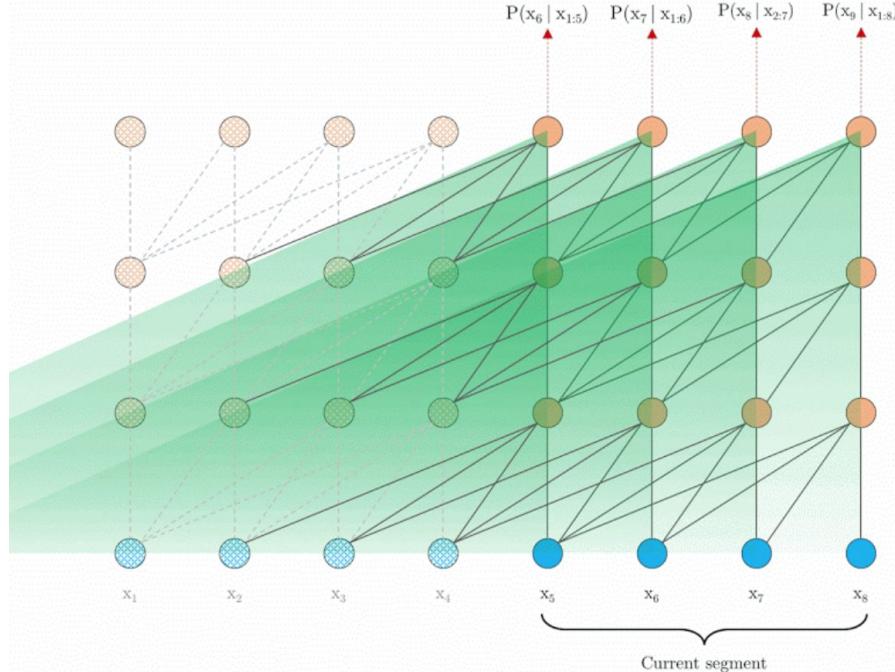
Vanilla Transformer with a fixed-length context at evaluation time

# Vanilla Transformer vs. Transformer-XL



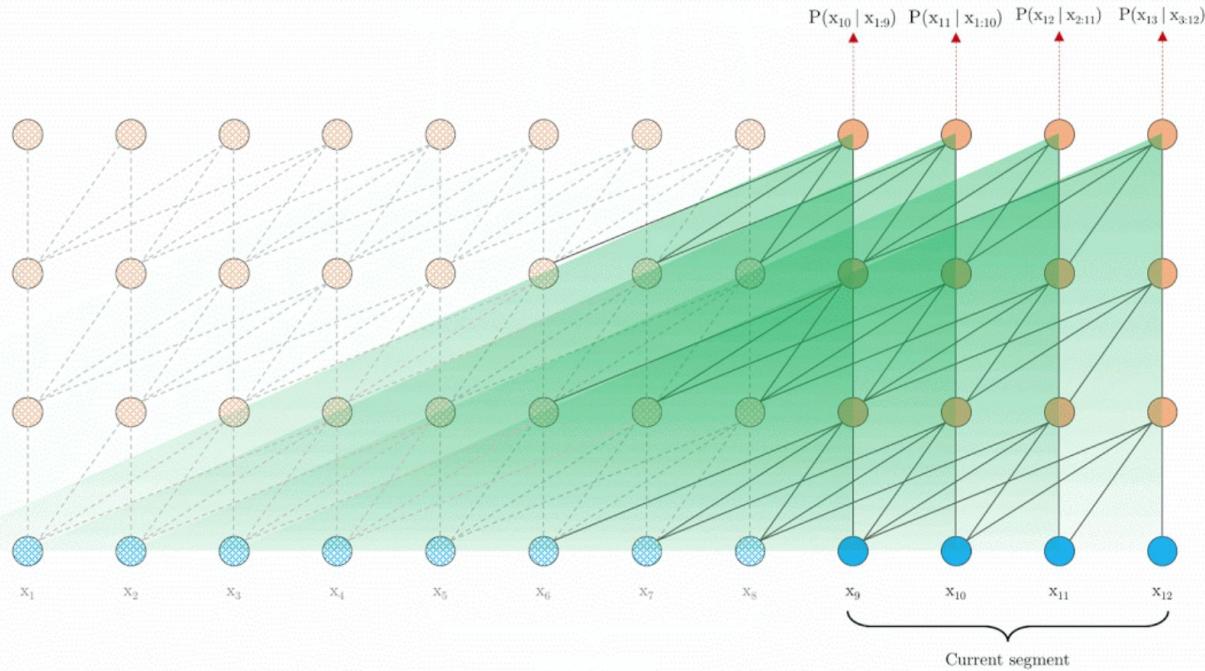
Vanilla Transformer with a fixed-length context at evaluation time

# Vanila Transformer vs. Transformer-XL



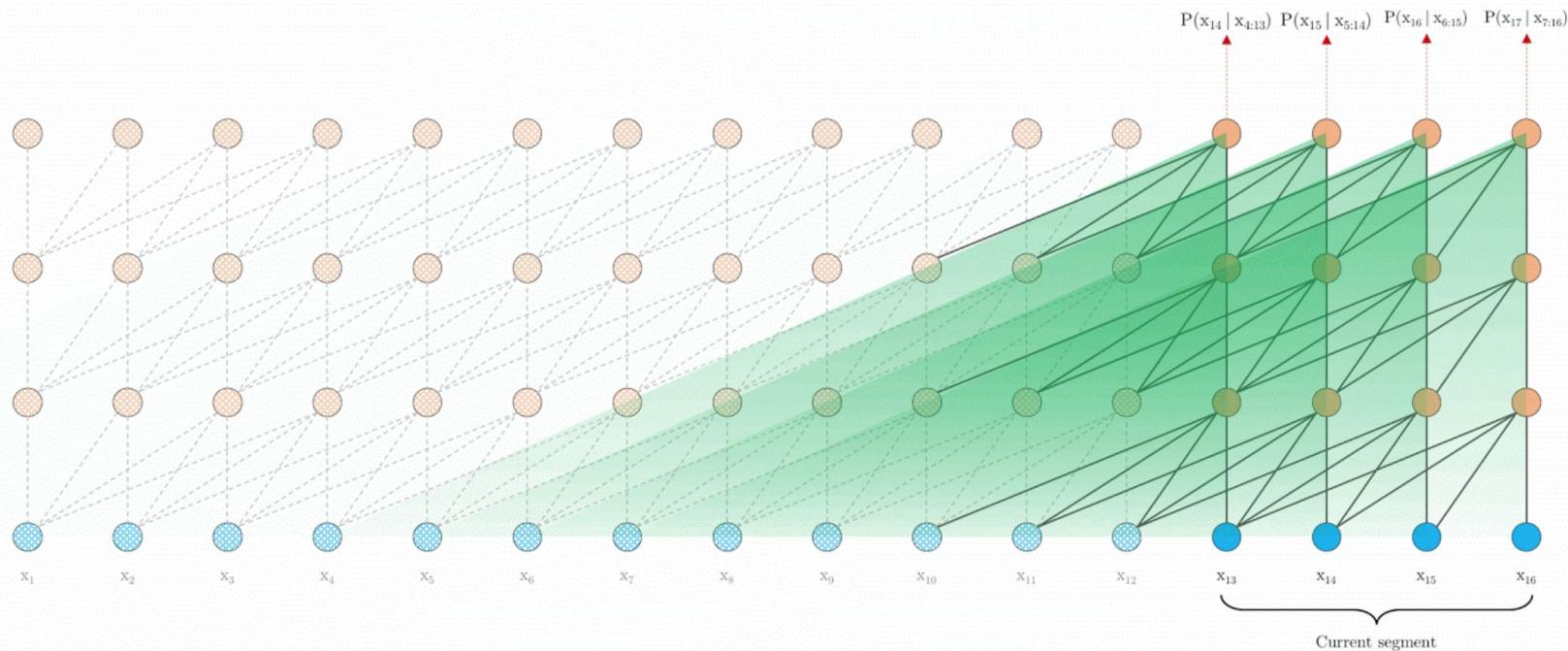
Transformer-XL with segment-level recurrence at evaluation time

# Vanila Transformer vs. Transformer-XL



Transformer-XL with segment-level recurrence at evaluation time

# Vanila Transformer vs. Transformer-XL



Transformer-XL with segment-level recurrence at evaluation time

# Vanila Transformer vs. Transformer-XL

- Transformer-XL learns dependency that is about 80% longer than RNNs and 450% longer than vanilla Transformers
- Transformer-XL is up to 1,800+ times faster than a vanilla Transformer during evaluation on language modeling tasks, because no re-computation is needed