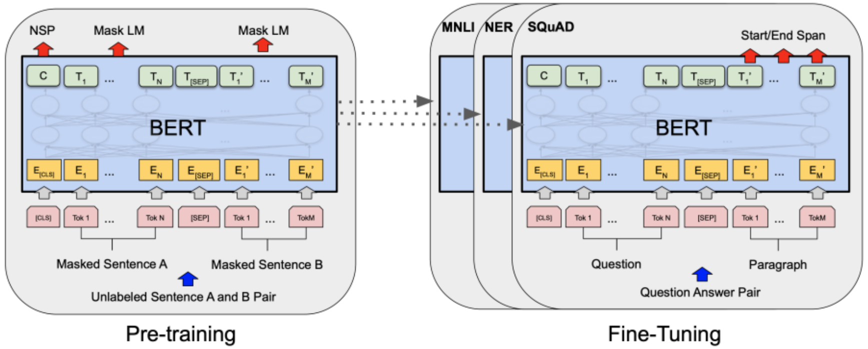


بنام خداوند جان و خرد

More About Transformers

Mohammad Taher Pilehvar

Natural Language Processing 1400
https://teias-courses.github.io/nlp00/

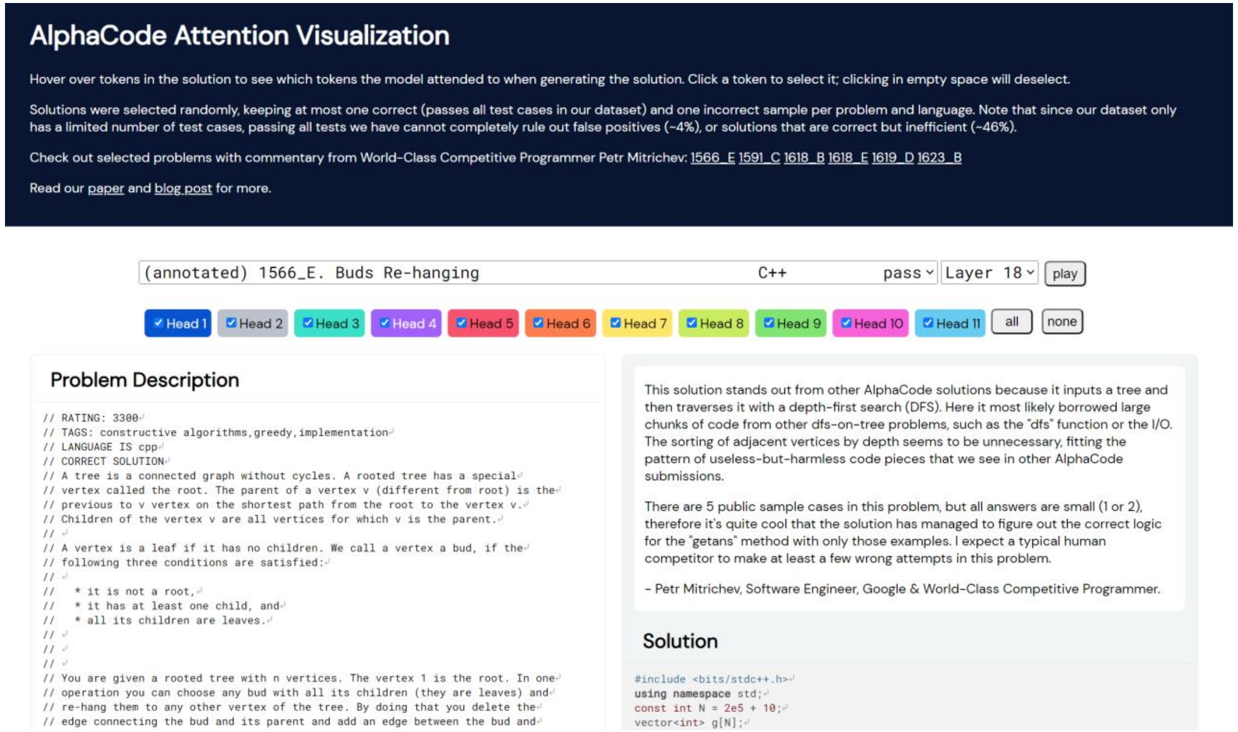
Most materials of these slides are taken from Stanford's CS224N

TeIAS | Tehran Institute for Advanced Studies

Breaking (Transformer) News!

AlphaCode (a pre-trained Transformer-based code generation model) achieved a top 54.3% rating on Codeforces programming competitions (Li et al., 2022)!

https://alphacode.deepmind.com/



Breaking (Transformer) News!

Pre-Trained Transformer-Based theorem prover sets new state-of-the-art (41.2% vs. 29.3%) on the [mniF2F](#) collection of challenging math Olympiad questions (Polu et al., 2022)

https://alphacode.deepmind.com/

PROBLEM 1

Adapted from AMC12 2000 Problem 5

Prove that if $|x-2| = p$, where $x < 2$, then $x - p = 2 - 2p$.

FORMAL | **INFORMAL**

```
theorem amc12_2000_p5       -- ← theorem name
  (x p : ℝ)                 -- ← the statement we want
  (h₀ : x < 2)              --   to prove
  (h₁ : abs (x - 2) = p) :
  x - p = 2 - 2 * p :=
begin                       -- ← formal proof starts here
  -- This first tactic requires that the prover invent
  -- the term: `abs (x - 2) = -(x - 2)`.
  have h₂ : abs (x - 2) = -(x - 2), {
    apply abs_of_neg,
    linarith,
  },
  rw h₁ at h₂,
  -- At this stage the remaining goal to prove is:
  -- `x - p = 2 - 2 * p` knowing that `p = -(x - 2)`.
  linarith,
end
```

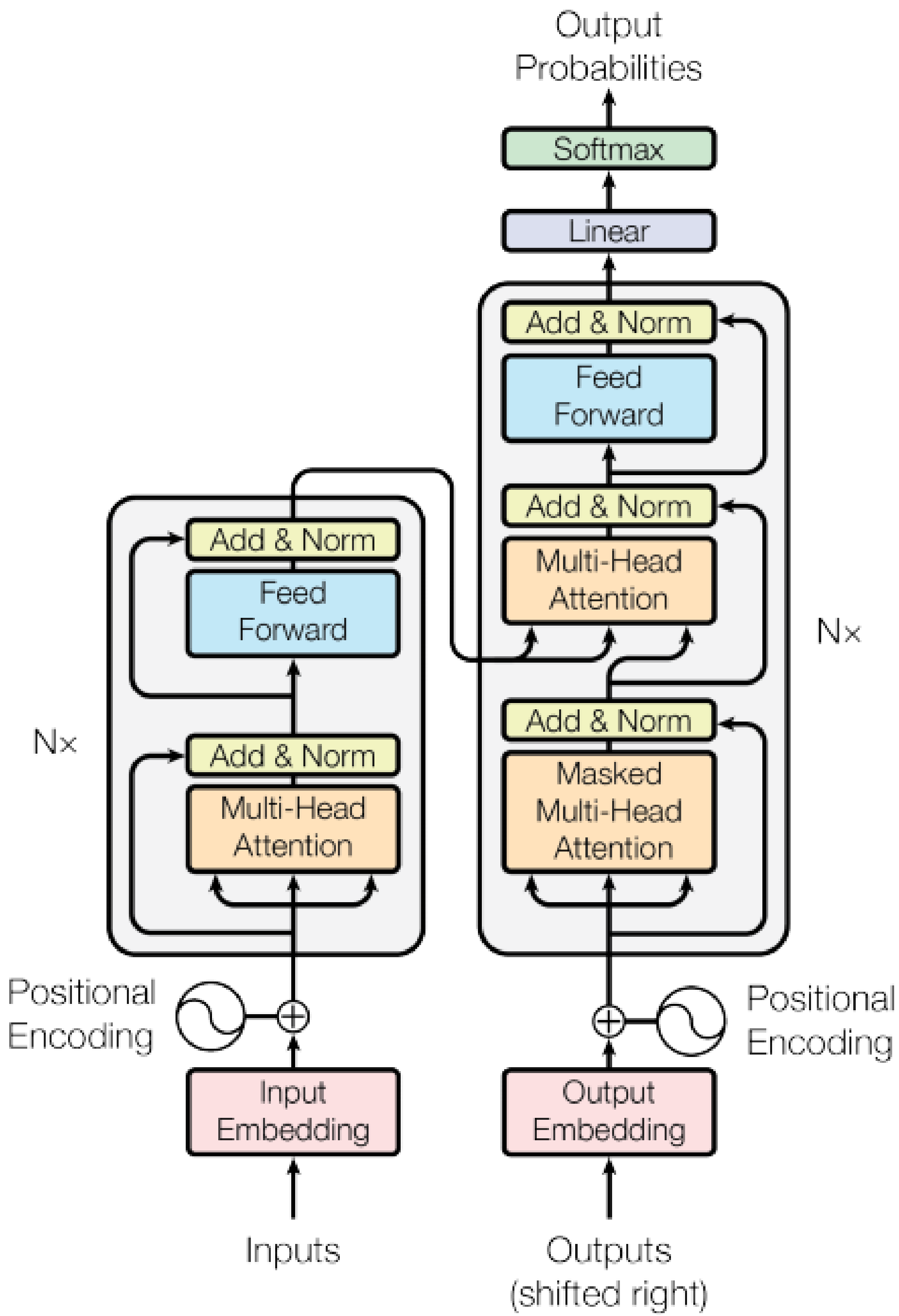
In this Lecture

- Quick reminder of Transformer model
- Subword modeling
- Model types
  - Decoders
  - Encoders
  - Encoder-Decoders

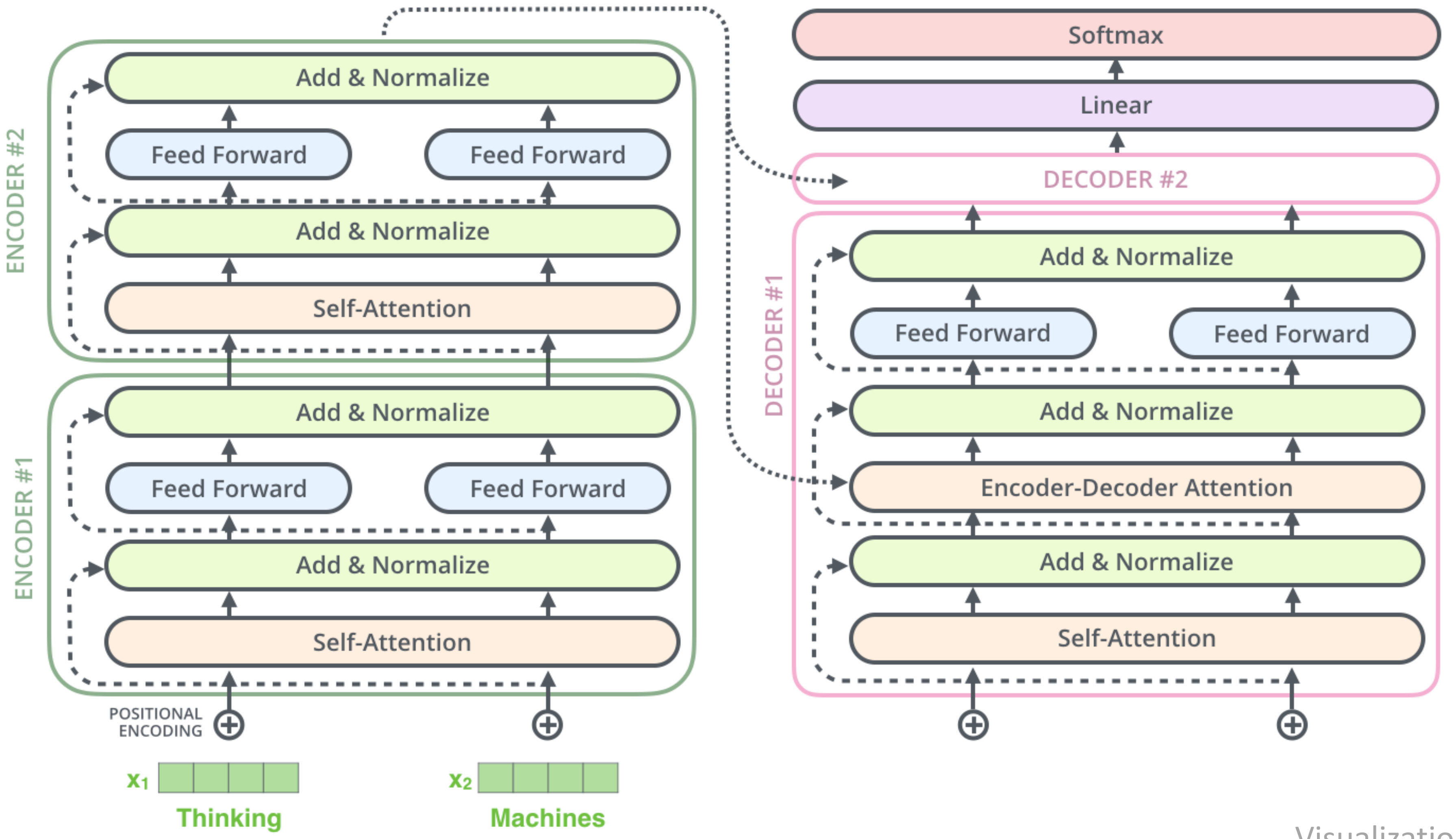
➢ Homework #2 will be out soon

➢ Project proposal is due this week

Transformers

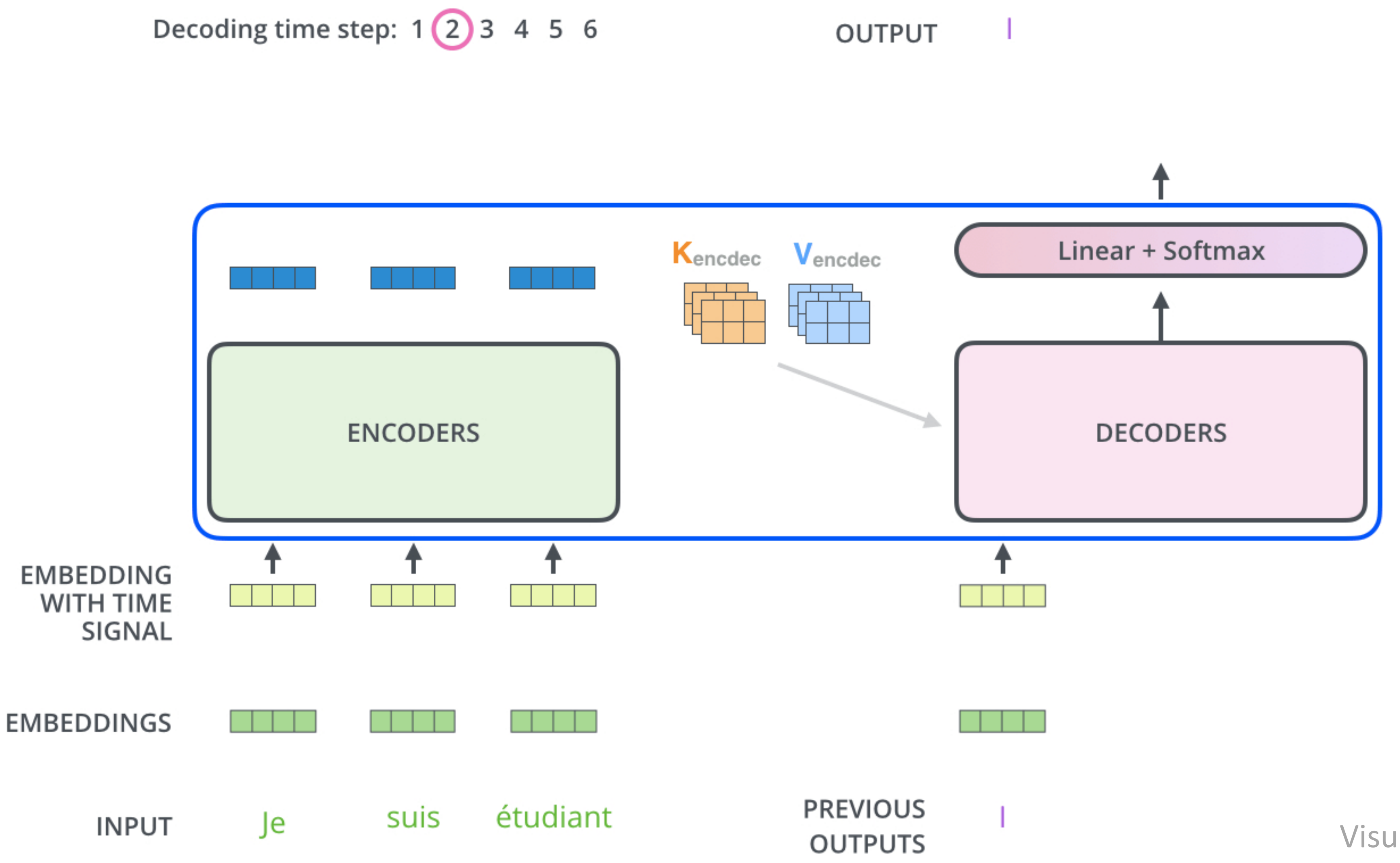


Transformer with 2 stacked encoders and decoders



Visualization by Jay Alammar

Decoder side



Visualization by Jay Alammar

Tokenizing text

- Simplest: split by spaces

```
["Don't", "you", "love", "🤗", "Transformers?", "We", "sure", "do."]
```

```
["Don", "'", "t", "you", "love", "🤗", "Transformers", "?", "We", "sure", "do", "."]
```

- spaCy and Moses are two popular rule-based tokenizers.

```
["Do", "n't", "you", "love", "🤗", "Transformers", "?", "We", "sure", "do", "."]
```

https://huggingface.co/docs/transformers/tokenizer_summary

Tokenizing text

- Hazm (https://github.com/sobhe/hazm)

```
>>> sent_tokenize('ما هم برای وصل کردن آمدیم! ولی برای پردازش، جدا بهتر نیست؟')
['ما هم برای وصل کردن آمدیم!', 'ولی برای پردازش، جدا بهتر نیست؟']
>>> word_tokenize('ولی برای پردازش، جدا بهتر نیست؟')
['ولی', 'برای', 'پردازش', '،', 'جدا', 'بهتر', 'نیست', '؟']
```

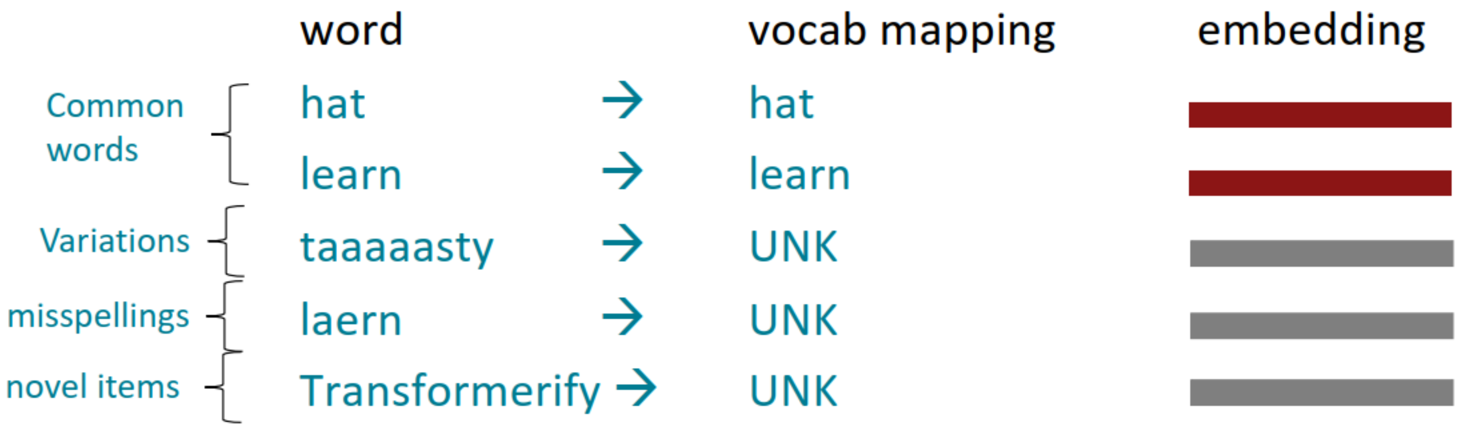
- Dadmatools (https://github.com/Dadmatech/DadmaTools)

https://huggingface.co/docs/transformers/tokenizer_summary

Word structure and subword models

Traditionally, we used to have a fixed vocab of hundreds of thousands of words (often built from the training set).

All *novel* words seen at test time are mapped to a single *UNK*.



Word structure and subword models

Fixed vocab makes *less* sense for languages with rich morphology (more word types, occurring less frequently)

گردانش فارسی							
بن واژه		دیدن					
بن ماضی		دید					
بن مضارع		بین					
شخص		مفرد			جمع		
		اول شخص	دوم شخص	سوم شخص	اول شخص	دوم شخص	سوم شخص
گذشته		من	تو	او/آن	ما	شما	آنها/ایشان
	ساده	دیدم	دیدی	دید	دیدیم	دیدید	دیدند
	استمراری	می‌دیدم	می‌دیدی	می‌دید	می‌دیدیم	می‌دیدید	می‌دیدند
	کامل	دیده‌بودم	دیده‌بودی	دیده‌بود	دیده‌بودیم	دیده‌بودید	دیده‌بودند
	التزامی	دیده‌باشم	دیده‌باشی	دیده‌باشد	دیده‌باشیم	دیده‌باشید	دیده‌باشند
	مستمر	داشتم می‌دیدم	داشتی می‌دیدی	داشت می‌دید	داشتیم می‌دیدیم	داشتید می‌دیدید	داشتند می‌دیدند
حال		من	تو	او/آن	ما	شما	آنها/ایشان
	ساده	بینم	بینی	بیند	بینیم	بینید	بینند
	استمراری	می‌بینم	می‌بینی	می‌بیند	می‌بینیم	می‌بینید	می‌بینند
	کامل	دیده‌ام	دیده‌ای	دیده‌است/دیده	دیده‌ایم	دیده‌اید	دیده‌اند
	ملموس	دارم می‌بینم	داری می‌بینی	دارد می‌بیند	داریم می‌بینیم	دارید می‌بینید	دارند می‌بینند
	التزامی	ببینم	ببینی	ببیند	ببینیم	ببینید	ببینند
آینده		من	تو	او/آن	ما	شما	آنها/ایشان
		خواهم دید	خواهی دید	خواهد دید	خواهیم دید	خواهید دید	خواهند دید
دستوری		-	تو	-		شما	-
	امر		ببین			ببینید	
	نهی		نبین			نبینید	

Word structure and subword models

Big vocab size:

- Enormous embedding matrix (input and output layers)
- Increased memory
- Increased time complexity

In general, transformers models rarely have a vocabulary size greater than 50,000, especially if they are pretrained only on a single language.

Word structure and subword models

https://huggingface.co/docs/transformers/tokenizer_summary Hugging Face

Subword tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords.

Examples and demo:

https://colab.research.google.com/drive/1E86l5oGlyZi7vIbzzOcC1RLgbTl8JVzm?usp=sharing
https://github.com/microsoft/SDNet/blob/master/bert_vocab_files/bert-base-uncased-vocab.txt

Word structure and subword models

Byte-Pair Encoding (BPE)

1. Split the training data into words. Create a list of unique words with their frequency.

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

2. Create a base vocabulary consisting of all symbols that occur in the set of unique words

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

3. Learn merge rules to form a new symbol from two symbols of the base vocabulary (until a vocabulary of certain size is reached).

```
("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

```
("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

Hugging Face

Word structure and subword models

WordPiece

- Very similar to BPE
- In contrast to BPE, WordPiece does not choose the most frequent symbol pair, but the one that maximizes the likelihood of the training data once added to the vocabulary.

- Referring to the previous example, maximizing the likelihood of the training data is equivalent to finding the symbol pair, whose probability divided by the probabilities of its first symbol followed by its second symbol is the greatest among all symbol pairs.

  - u followed by g would have only been merged if the probability of ug divided by u (followed by) g would have been greater than for any other symbol pair

Hugging Face

WordPiece

Word structure and subword models

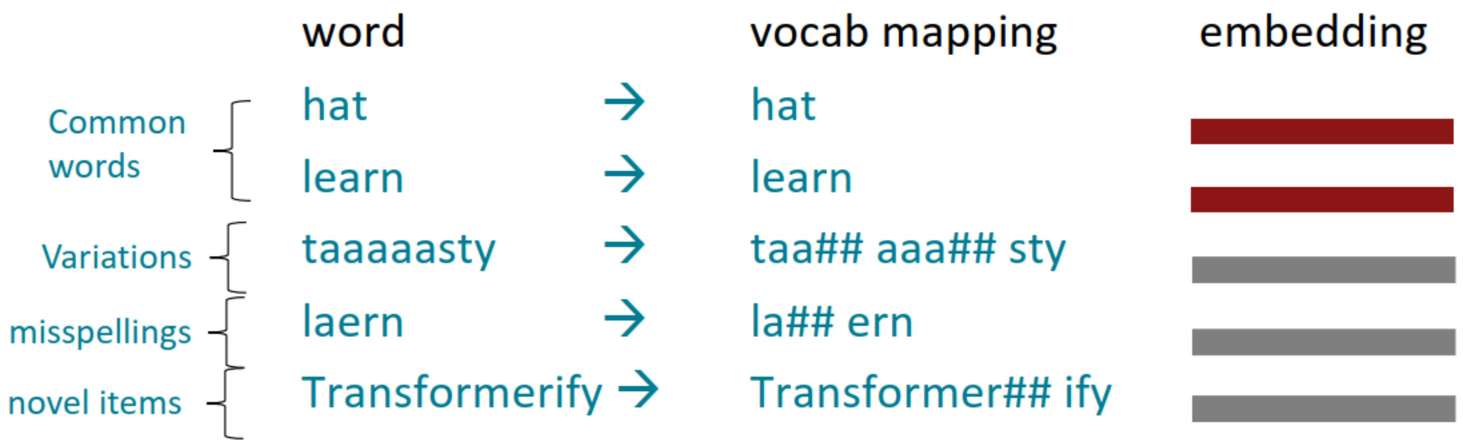
SentencePiece

- Not all languages use spaces to separate words
- SentencePiece treats the input as a raw input stream, thus including the space in the set of characters to use.
- It then uses the BPE or other algorithms to construct the appropriate vocabulary.

Word structure and subword models

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

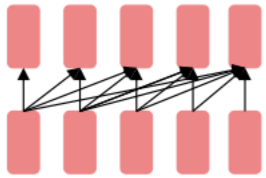
In the worst case, words are split into as many subwords as they have characters.



Outline

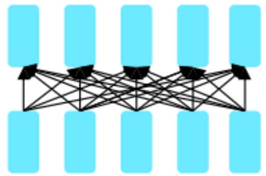
- Quick reminder of Transformer model
- Subword modeling
- Model types
  - Decoders
  - Encoders
  - Encoder-Decoders

Model types



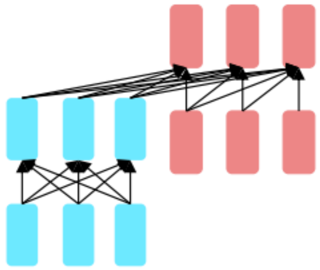
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Decoders

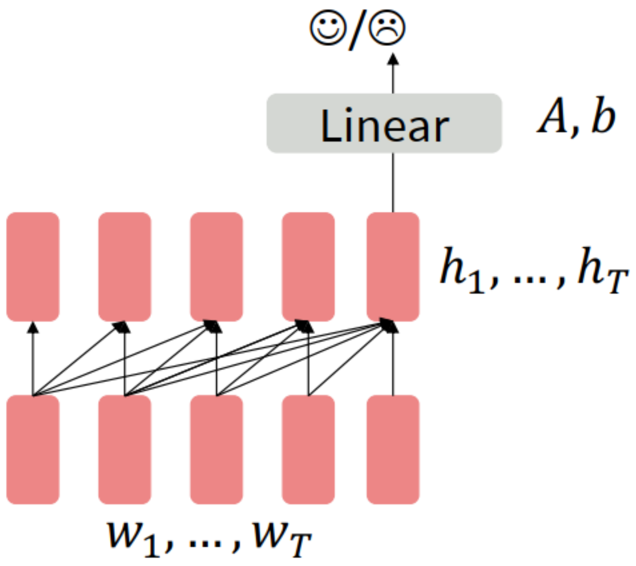
- When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$

- We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \dots, h_t = Decoder(w_1, \dots, w_t)$$

$$y \sim Ah_T + b$$

- Where A and b are randomly initialized and specified by the downstream task. Gradients backpropagate through the whole network.



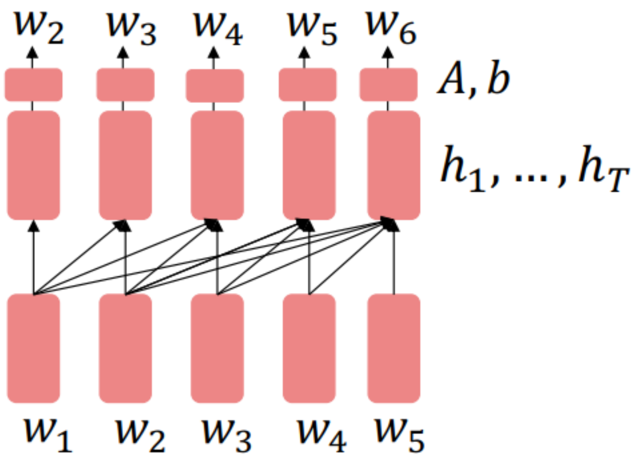
Decoders

- It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p(w_t|w_{1:t-1})$

- This is helpful in tasks where the output is a sequence with a vocabulary like that at pretraining time!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)

$$h_1, \dots, h_t = Decoder(w_1, \dots, w_t)$$

$$w_t \sim Ah_{t-1} + b$$



- Where A, b were pretrained in the language model!

Decoders

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
- Contains long spans of contiguous text, for learning long-distance dependencies

Decoders

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

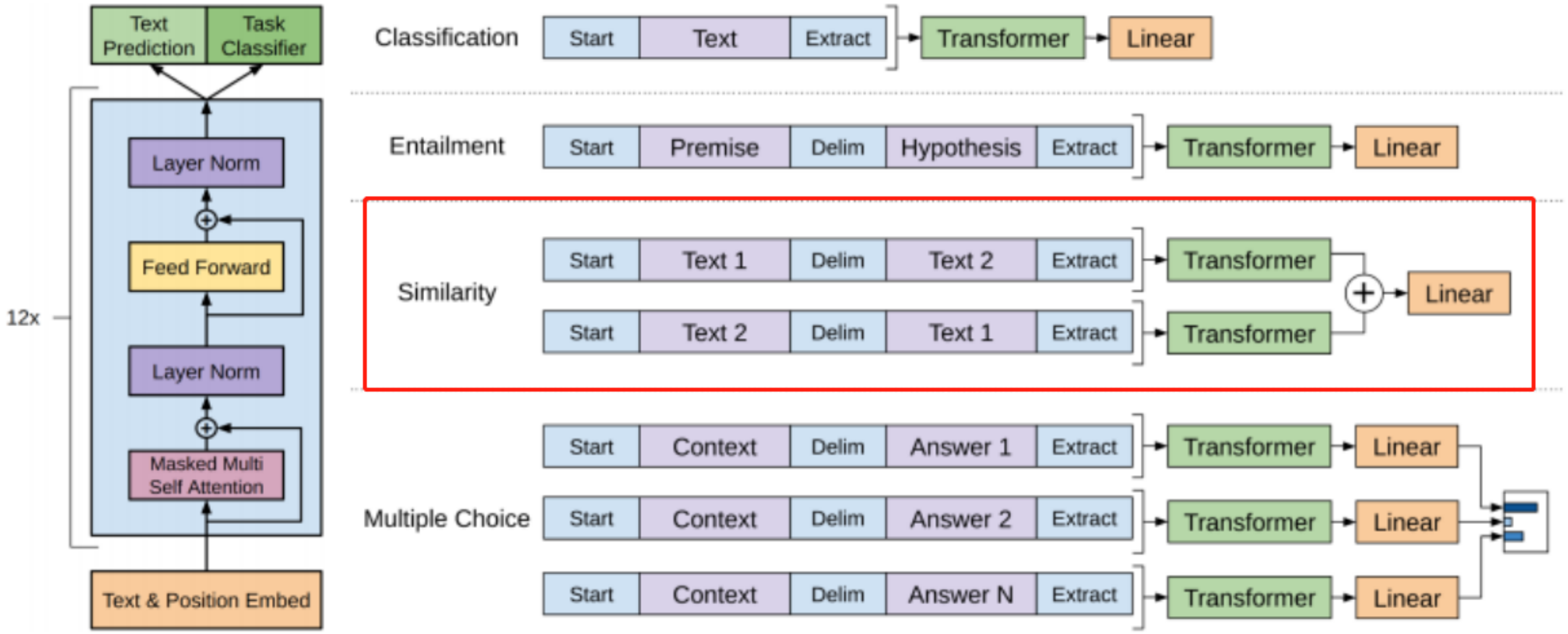
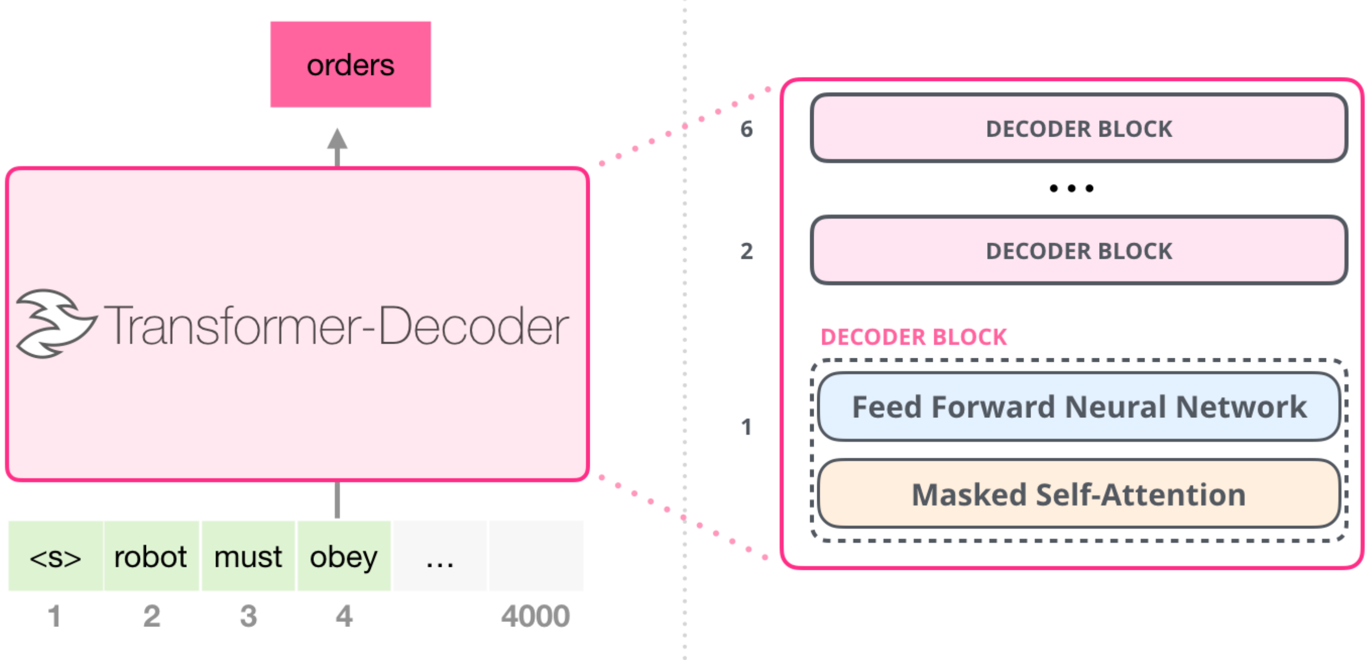


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

GPT-2 (Generative Pretrained Transformer)

- GPT (2018)
- GPT-2 (2019)
- GPT-3 (2020)



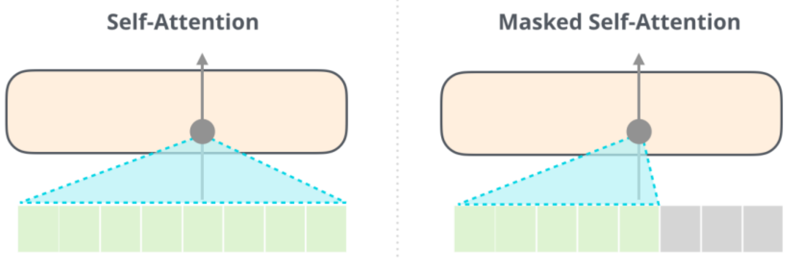


Illustration by [Jay Alammar]

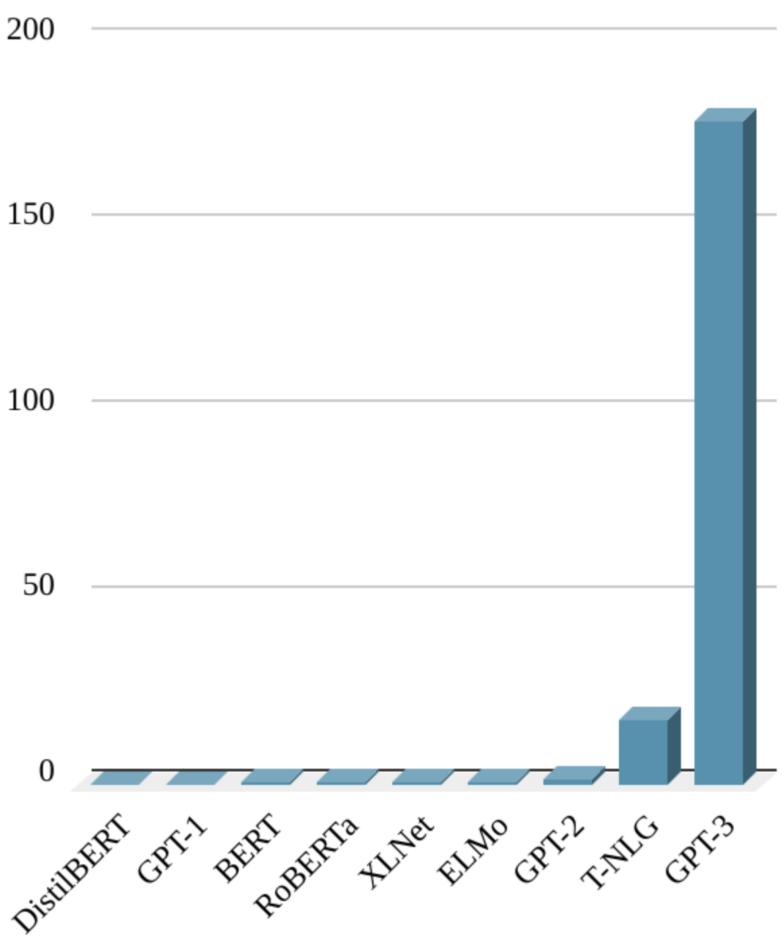
GPT-3

175 billion parameters!

GPT-2 had 1.5B

The largest so far (by Microsoft) had 17B

Training cost: $12M



GPT-3: Conversation

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

GPT-3: Generation

I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a “feeling brain”. But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

debuild.co

Describe your app. Clear Generate

Just describe your app!

debuild.co

Describe your app. Clear Generate

a button that says "Add $3" and a button that says "Withdraw $5". then show me my balance debuild.co

Describe your app.

Clear

Generate

Just describe your app!

```
// a button that says "Add $3" and
a button that says "Withdraw $5".
then show me my balance
class App extends React.Component
{

constructor(props) {
```

Add $3

Withdraw $5

My balance is 4

```
// a button that says "Add $3" and
a button that says "Withdraw $5".
then show me my balance
class App extends React.Component
{

constructor(props) {

super(props)

this.state = {

balance: 0

}

}

onAdd3 = () => {

this.setState({ balance:
this.state.balance + 3 })
```

```
this.setState({ balance:
this.state.balance + 3 })

}

onWithdraw5 = () => {

this.setState({ balance:
this.state.balance - 5 })

}

render() {

return (

<div>

<button onClick={this.onAdd3}>Add
$3</button>
```

```
this.setState({ balance:
this.state.balance - 5 })

}

render() {

return (

<div>

<button onClick={this.onAdd3}>Add
$3</button>

<button onClick=
{this.onWithdraw5}>Withdraw
$5</button>

<p>My balance is
{this.state.balance}</p>
```

debuild.co

Describe your app. Clear Generate

Just describe your app!

```
// a button that says "Add $5" and
a button that says "Withdraw $3"
and a button that says "Give away
all my money". when the give away
button is pressed, only set my
money to 0 if its not negative.
then display my balance.
class App extends React.Component
{
```

Add $5 Withdraw $3

Give away all my money

My balance is -6

```
onGiveMoney = () => {

if (this.state.money > 0) {

this.setState({ money: 0 })

}
```

GitHub Copilot

sentiment.ts | write_sql.go | parse_expenses.py | addresses.rb

```python
import datetime

def parse_expenses(expenses_string):
    """Parse the list of expenses and return the list of triples (date, value, currency).
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2016-01-02 -34.01 USD
        2016-01-03 2.59 DKK
        2016-01-03 -2.72 EUR
    """
    expenses = []
    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split(" ")
        expenses.append((datetime.datetime.strptime(date, "%Y-%m-%d"),
                         float(value),
                         currency))
    return expenses
```

Copilot

GPT-3: in-context learning

Very large language models seem to perform some kind of learning without gradient steps simply from examples you provide within their contexts.

The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing the task to a certain extent.

Input (prefix within a single Transformer decoder context)

```
thanks -> ممنون
hello -> سلام
day -> روز
elephant ->
```

Output (conditional generations): فیل

PaLM: Pathways Language Model

540 billion parameters!

Explaining a joke

Prompt

```
Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for
their TPU team? It showed them how to communicate between two
different pods!
```

Model Response

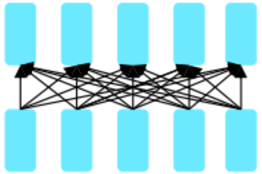
```
Prediction: TPUs are a type of computer chip that Google uses
for deep learning. A "pod" is a group of TPUs. A "pod" is also a
group of whales. The joke is that the whale is able to
communicate between two groups of whales, but the speaker is
pretending that the whale is able to communicate between two
groups of TPUs.
```

https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

Model types

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa

Encoder-Decoders

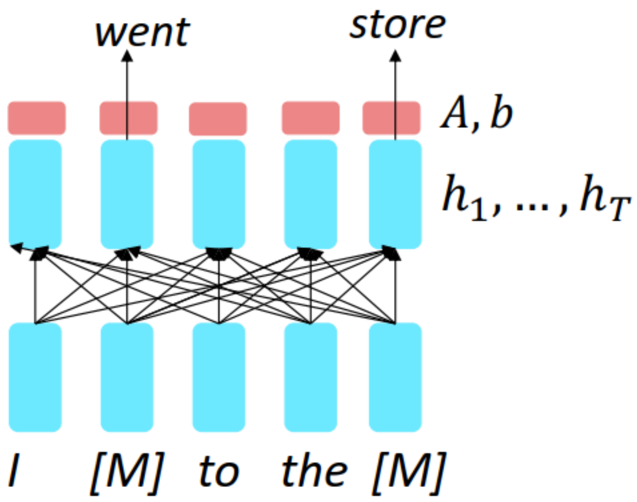
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoders: what objectives to use?

Bidirectional context: we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called Masked LM.



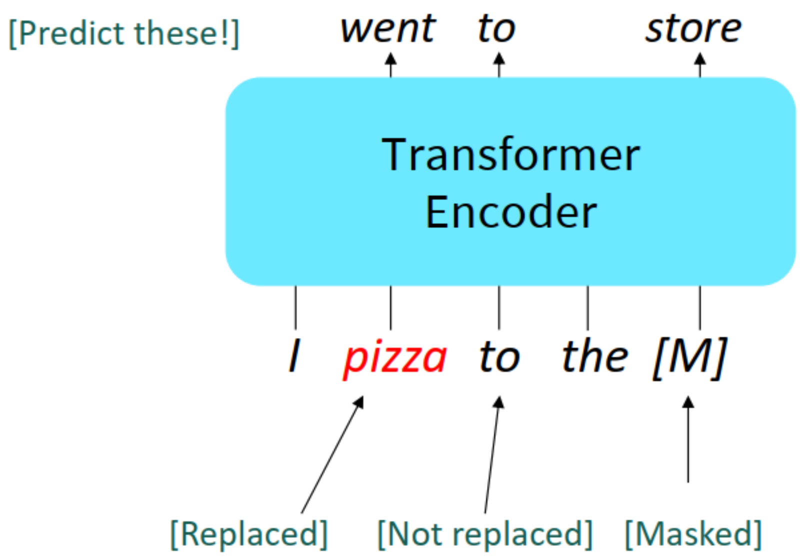
BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective, open-sourced their model as the tensor2tensor library, and released the weights of their pretrained Transformer (BERT).

Some more details about Masked LM for BERT:

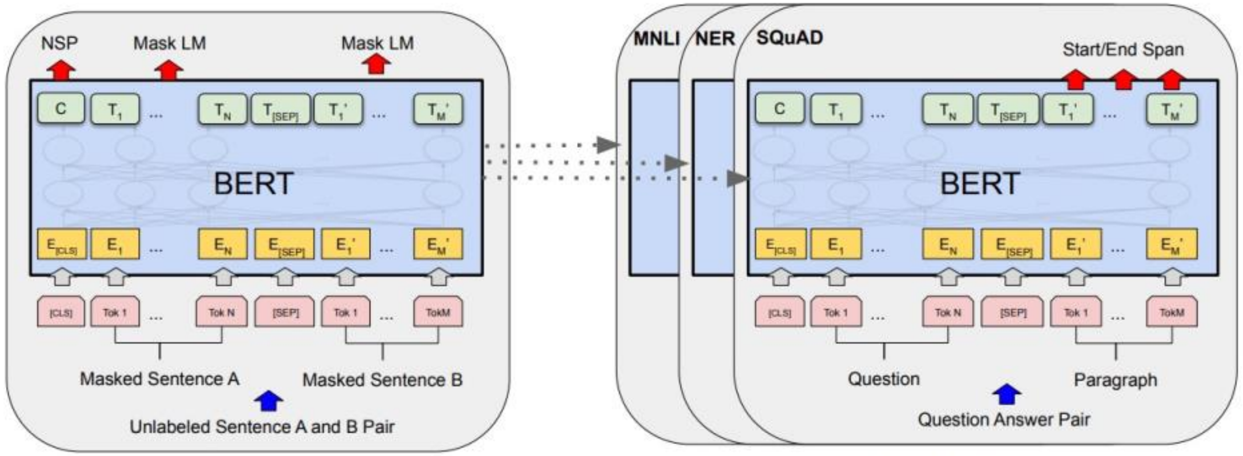
- Predict a random 15% of (sub)word tokens.
- Replace input word with [MASK] 80% of the time
- Replace input word with a random token 10% of the time
- Leave input word unchanged 10% of the time (but still predict it!)

- Why? Doesn't let the model get complacent and not build strong representations of non-masked words.
  (No masks are seen at fine-tuning time!)



BERT: Bidirectional Encoder Representations from Transformers

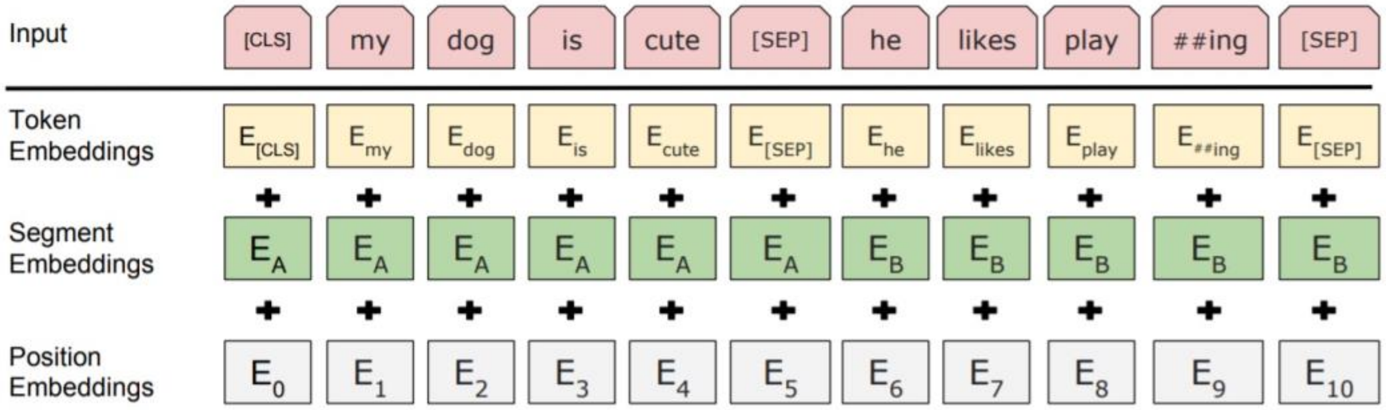
Unified Architecture: As shown below, there are minimal differences between the pre-training architecture and the fine-tuned version for each downstream task.



Pre-training
Fine-Tuning

BERT: Bidirectional Encoder Representations from Transformers

The pretraining input to BERT was two separate contiguous chunks of text:



BERT was trained to predict whether one chunk follows the other or is randomly sampled.

- Later work has argued this “next sentence prediction” is not necessary

BERT: Bidirectional Encoder Representations from Transformers

Details about BERT

- Two models were released
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.

- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)

- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)

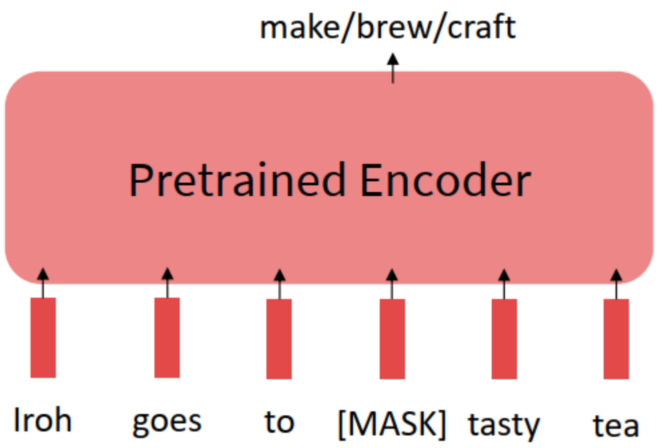
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

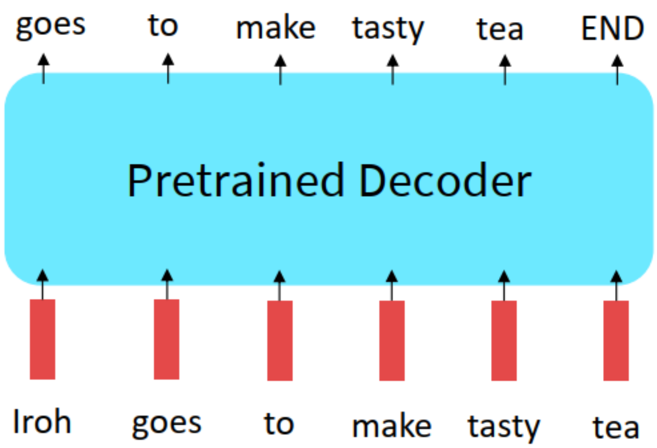
BERT: Bidirectional Encoder Representations from Transformers

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

BERT: Bidirectional Encoder Representations from Transformers

Encoders are not suitable for generation tasks



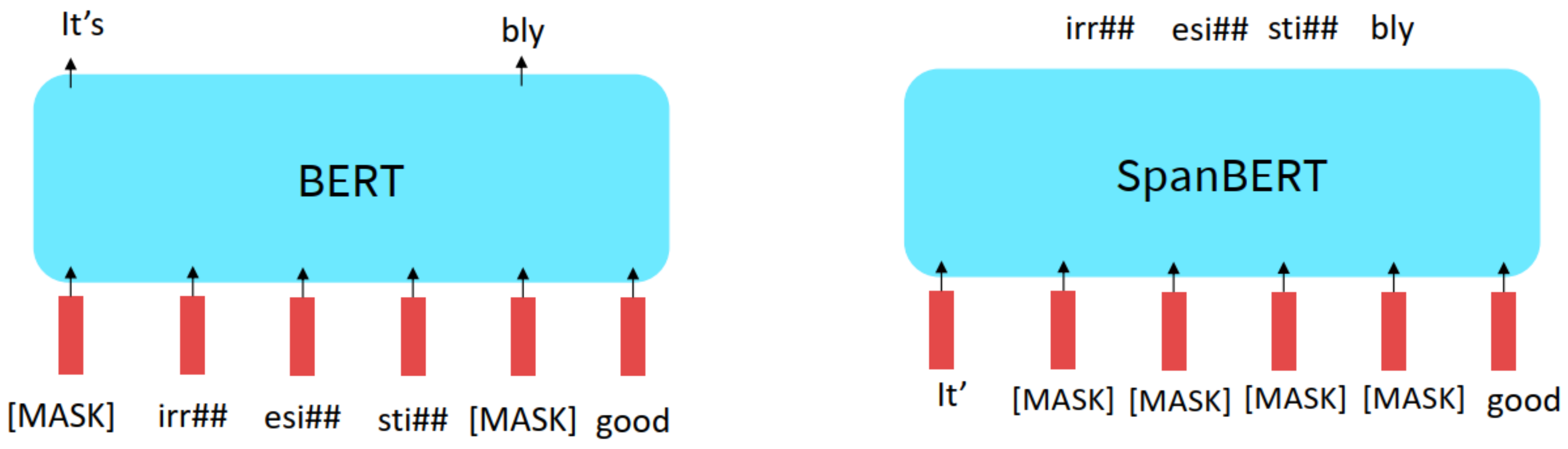


Extenstions of BERT

You'll see a lot of BERT variants like RoBERTa, SpanBERT, +++

Some generally accepted improvements to the BERT pretraining formula:

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task



Extenstions of BERT

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

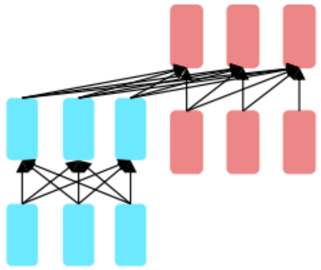
Model types

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- **Examples:** GPT-2, GPT-3, LaMDA

Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- **Examples:** BERT and its many variants, e.g. RoBERTa



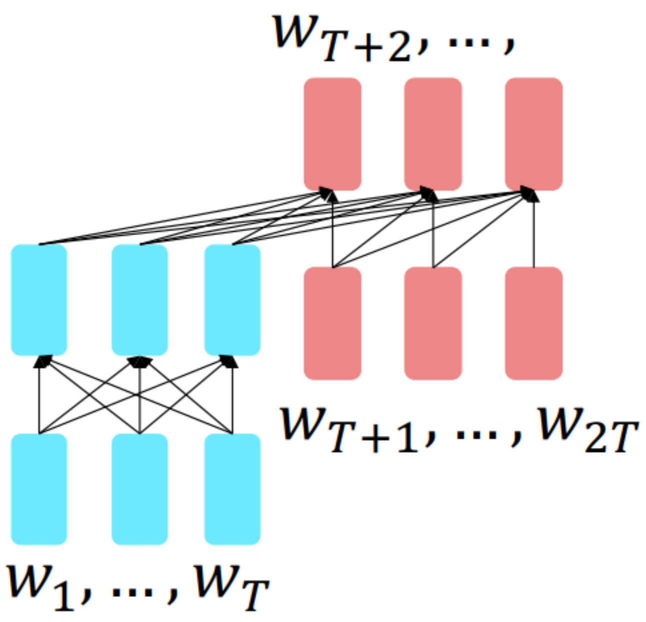
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- **Examples:** Transformer, T5, Meena

Pretraining encoder-decoders: what objectives to use?

For encoder-decoders, we could do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \dots, h_T = Encoder(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Aw_i + b, i > T$$



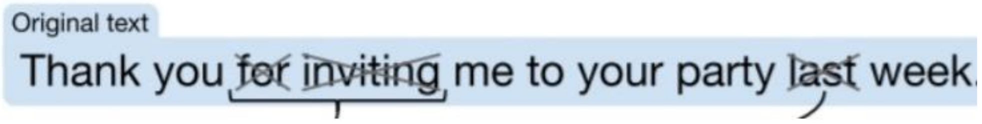
The encoder portion benefits from bidirectional context; the decoder portion is used to train the whole model through language modeling.

Raffel et al., (2018): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

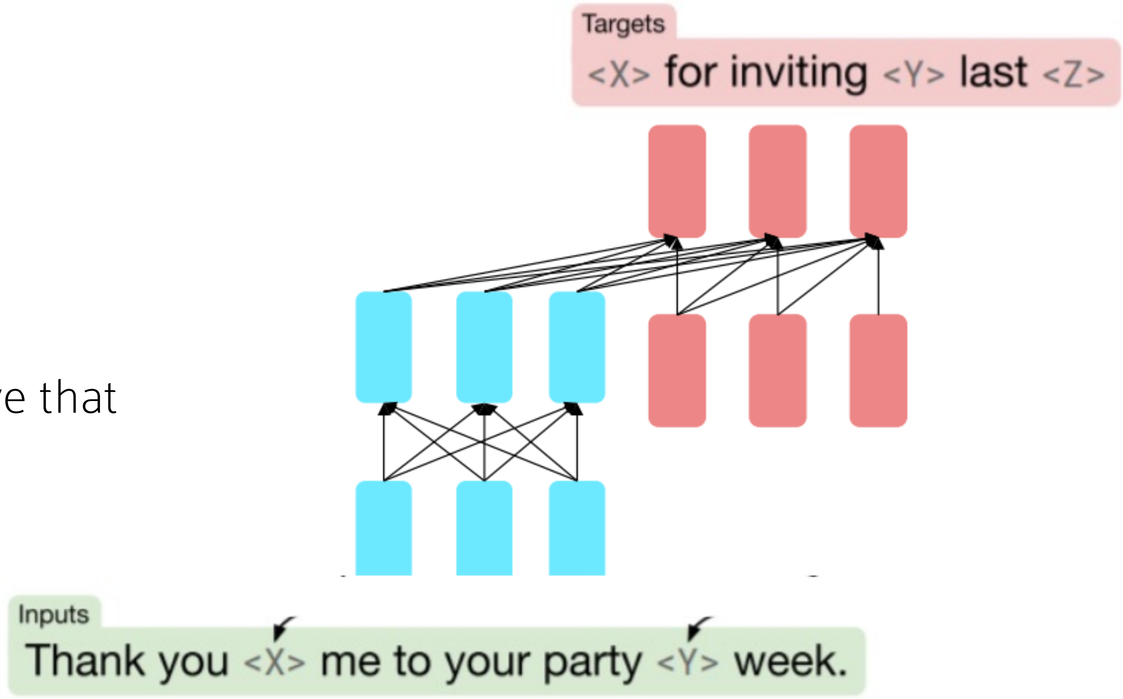
Pretraining encoder-decoders: what objectives to use?

What Raffel et al., 2018 found to work best was span corruption.
Their model: T5.

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



This is implemented in text preprocessing: it's still an objective that looks like language modeling at the decoder side.



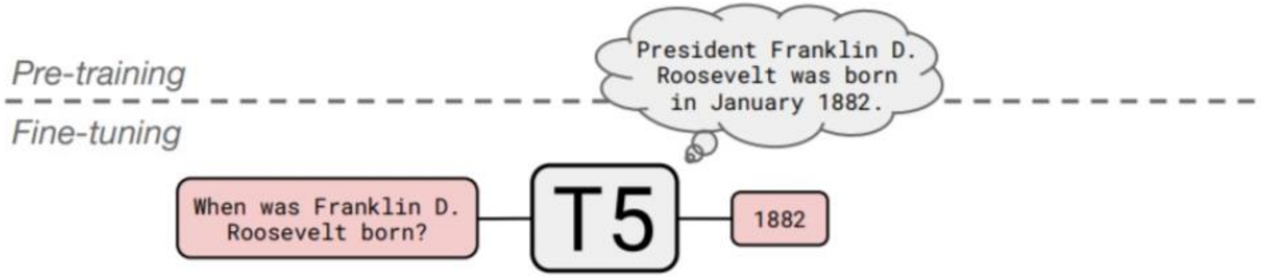
Raffel et al., (2018): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Pretraining encoder-decoders: what objectives to use?

A fascinating property of T5: it can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

All “open-domain” versions



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

Raffel et al., (2018): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Questions