Transformers successes in NLP & Vision

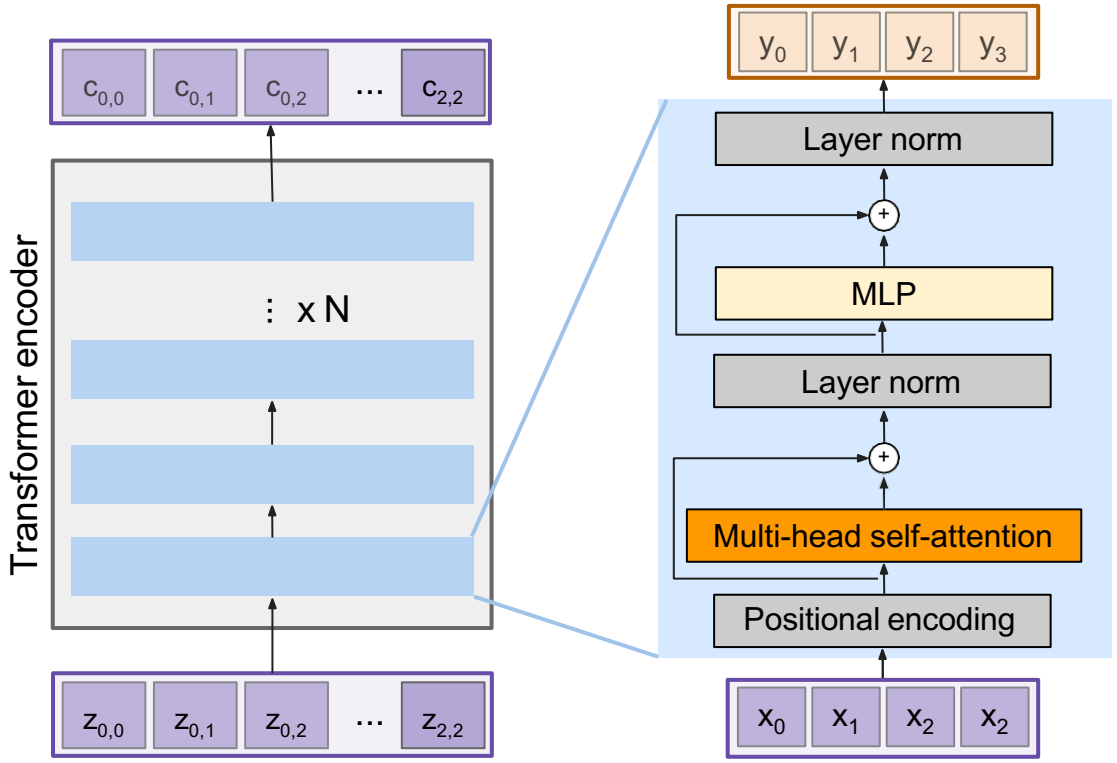
M. Soleymani
Sharif University of Technology
Spring 2024

Most slides have been adopted from Fei Fei Li and colleagues lectures cs231n, Stanford and Manning & colleagues lectures cs224n, Stanford

Outline

- Transformers in NLP
  - Subword modeling
  - Model pretraining three ways:
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Interlude: what do we think pretraining is teaching?
- Transformers in vision

Recap: The Transformer encoder block



Transformer Encoder Block:

Inputs: Set of vectors **x**
Outputs: Set of vectors **y**

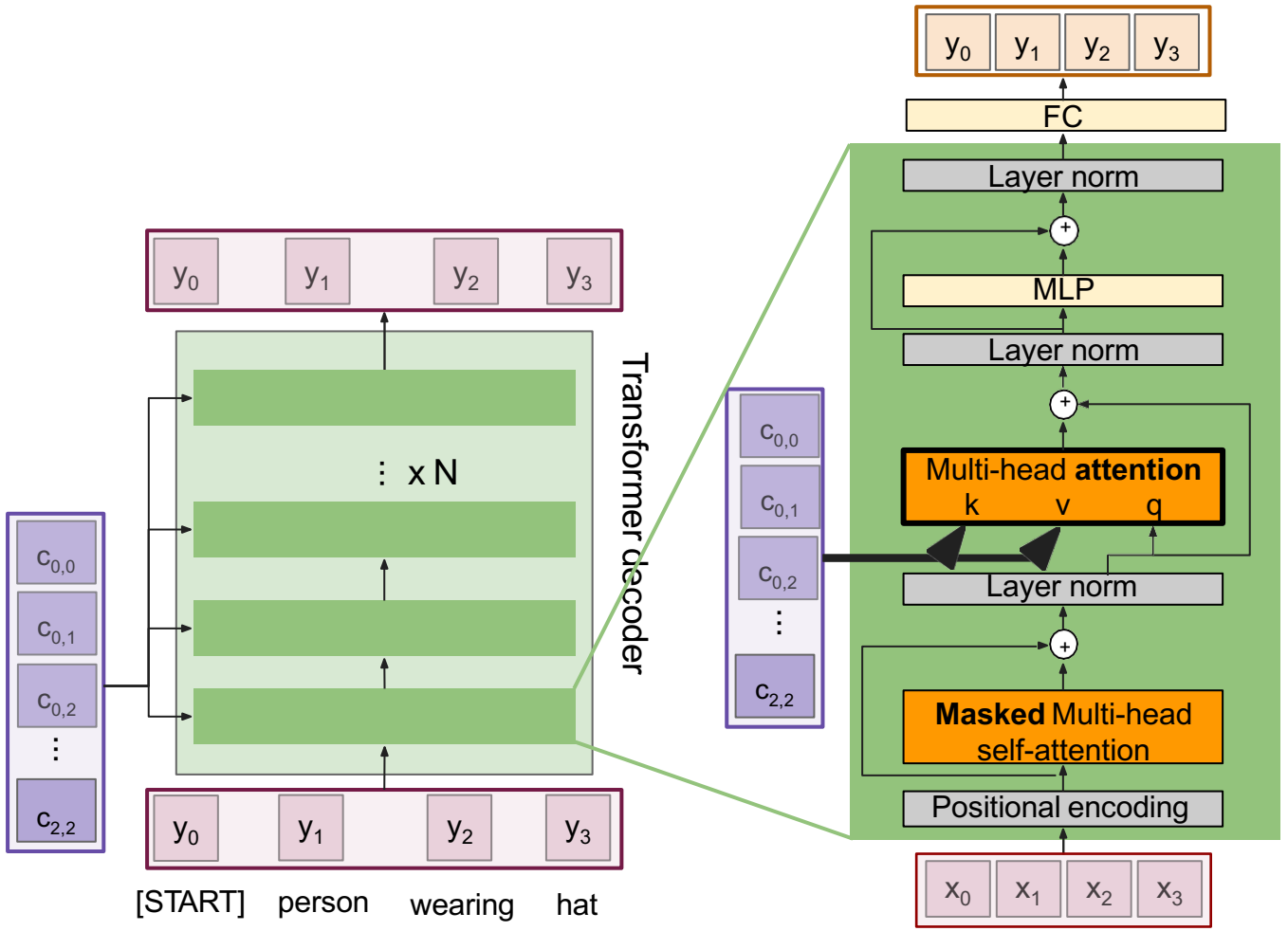
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al, "Attention is all you need", NeurIPS 2017

Recap: The Transformer decoder block



Multi-head attention block attends over the transformer encoder outputs.

For image captions, this is how we inject image features into the decoder.

Vaswani et al, "Attention is all you need", NeurIPS 2017

Recap: The Transformer encoder-decoder

- Transformer Decoder is modified to perform cross-attention to the output of the Encoder.

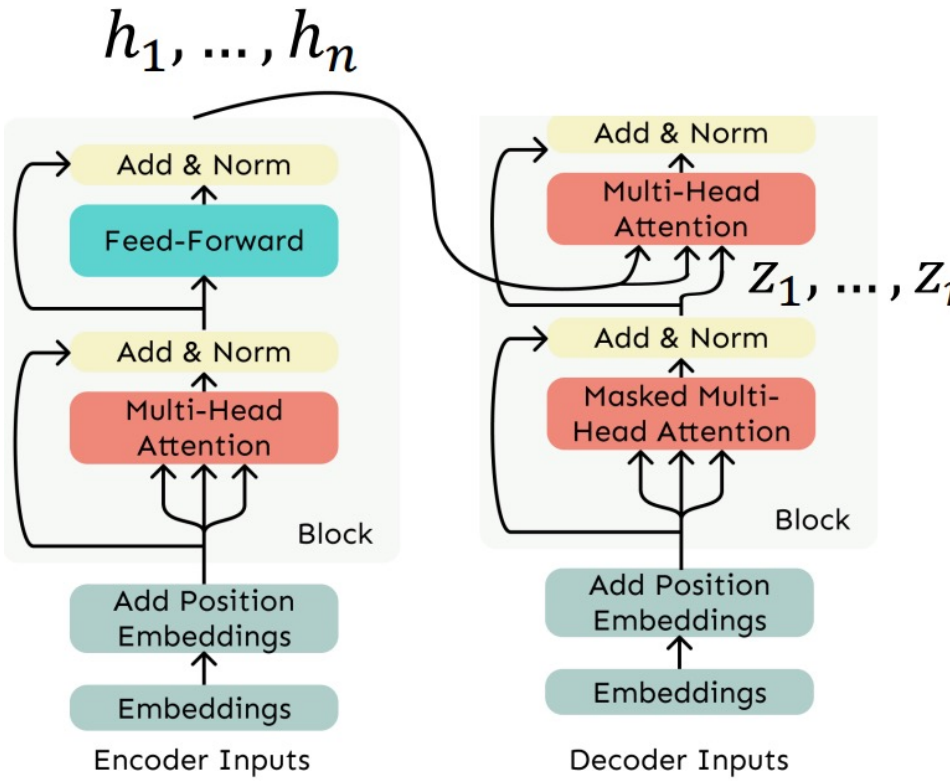
- Let $h_1, \dots, h_n$ be output vectors from the Transformer encoder
- Let $z_1, \dots, z_n$ be input vectors from the Transformer decoder, $z_i \in \mathbb{R}^d$

- Then keys and values are drawn from the encoder (like a memory):

$$k_i = W_k^T h_i \qquad v_i = W_v^T h_i$$

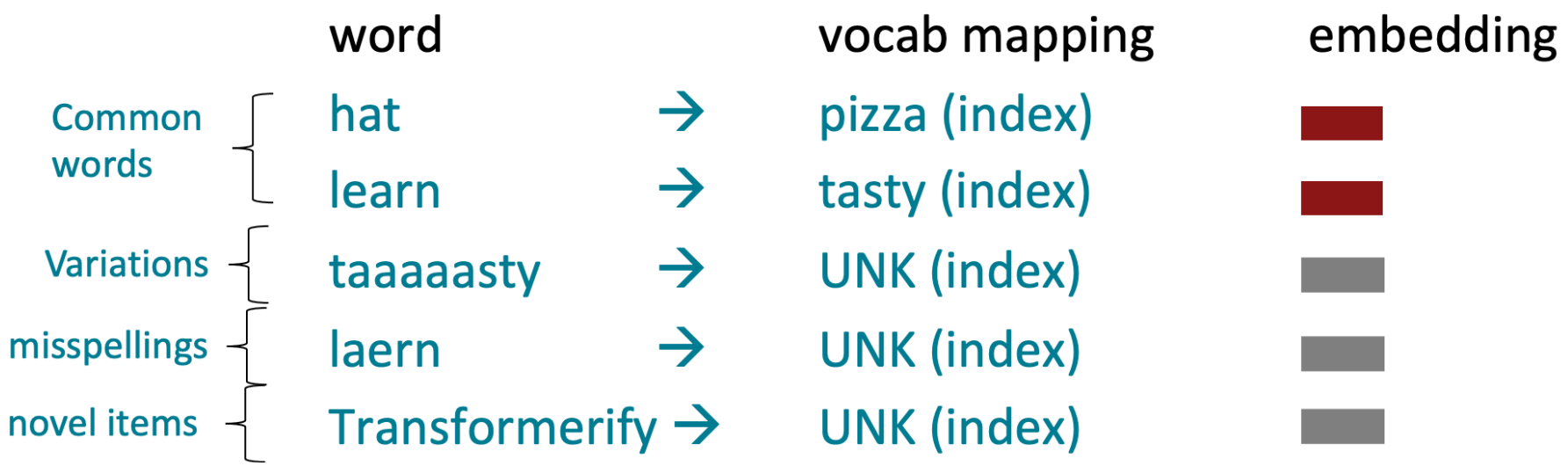
- And the queries are drawn from the decoder

$$q_i = W_Q^T z_i$$



Word structure and subword models

- We assume a fixed vocab of tens of thousands of words, built from the training set.
- All novel words seen at test time are mapped to a single UNK.



The byte-pair encoding algorithm

- Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)
  - The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens).
  - At training and testing time, each word is split into a sequence of known subwords.

- Byte-pair encoding is a simple, effective strategy for defining a vocabulary.
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus, find the most common adjacent characters "a,b"; add "ab" as a subword.
  3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

- Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

[Sennrich et al., 2016, Wu et al., 2016]

Wu et al., Google's Neural Machine Translation System, 2016

Example

- Look for the most frequent pairing, merge them, and perform the same iteration again and again until we reach our token limit

Corpus: aaabdaaabac

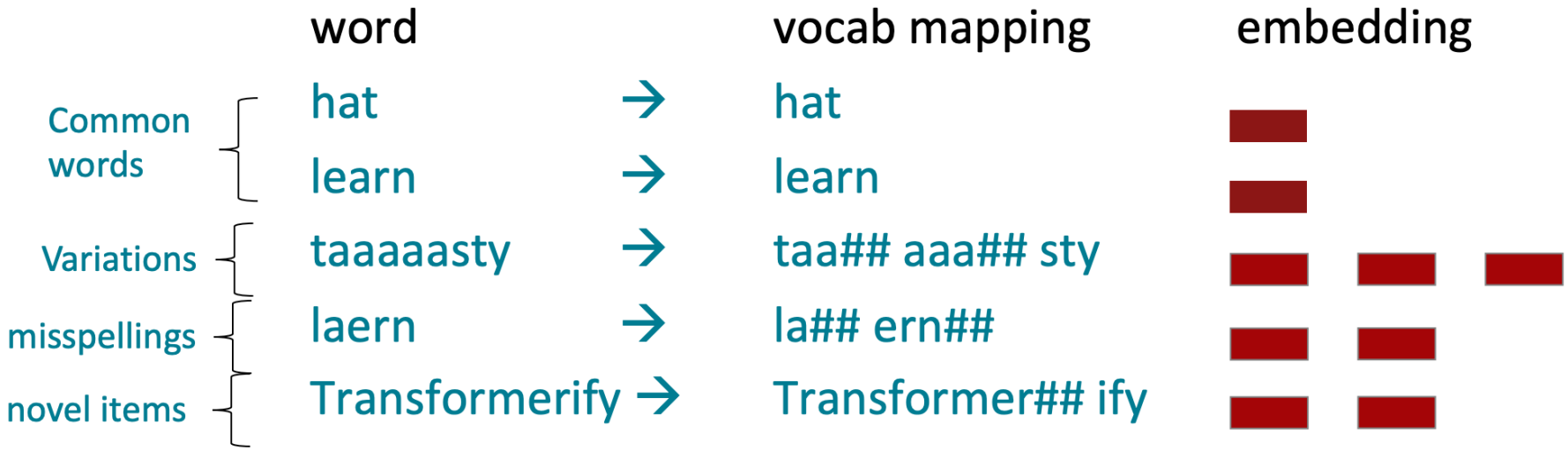
- Z=aa ZabdZabac vocabulary: {a,b,c,d,Z}
- Y=ab ZYdZYac vocabulary: {a,b,c,d,Z,Y}
- X=ZY XdXac vocabulary: {a,b,c,d,Z,Y,X}

Example from wikipedia

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.

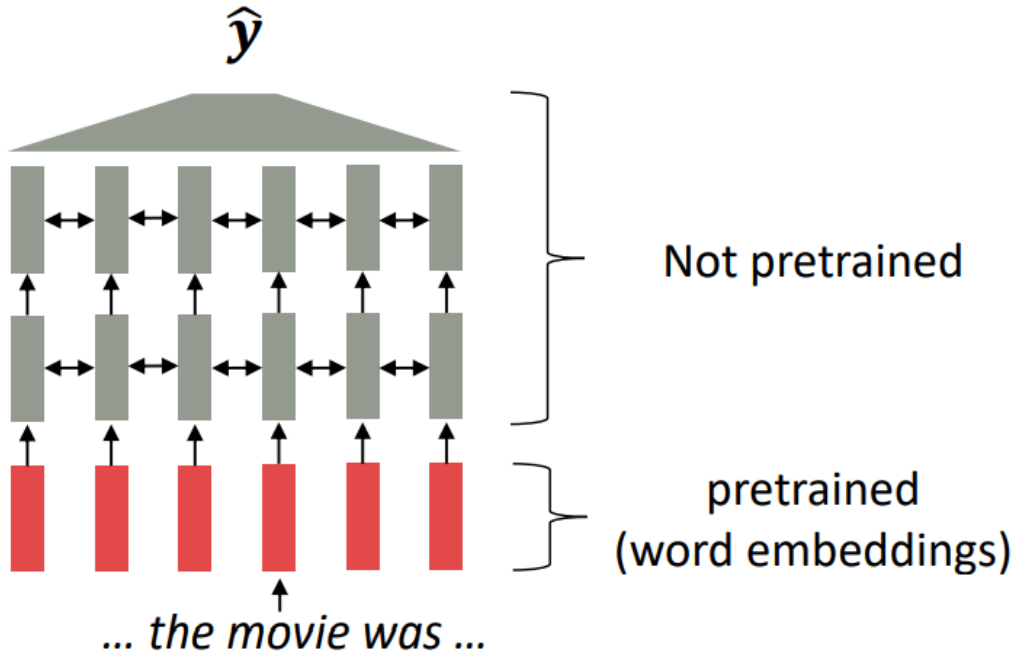
- In the worst case, words are split into as many subwords as they have characters.



Where we were: pretrained word embeddings

- Circa 2017:
  - Start with pretrained word embeddings (no context!)
  - Learn how to incorporate context in an LSTM or Transformer while training on the task.

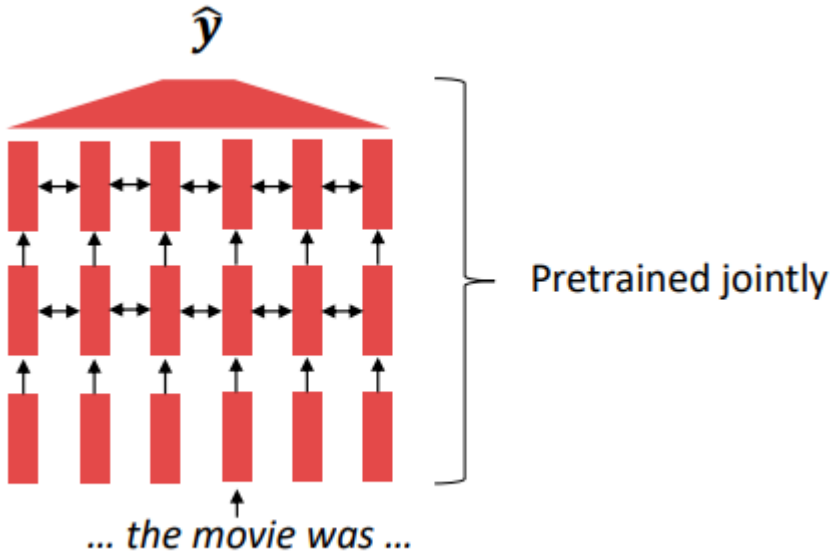
- Some issues to think about:
  - The training data of downstream task (like question answering) must be sufficient to teach all contextual aspects of language.
  - Most of the parameters in our network are randomly initialized!



Pretraining

- In modern NLP:
  - All (or almost all) parameters in networks are initialized via pretraining.
  - Pretraining methods **hide** **parts** of the input from the model, and train the model to **reconstruct** **those parts**.

- This has been exceptionally effective at building strong:
  - representations of language
  - parameter initializations for strong NLP models.
  - Probability distributions over language that we can sample from



This model has learned how to represent entire sentences through pretraining

What can we learn from reconstructing the input?

I put ___ fork down on the table.

I went to the ocean to see the fish, turtles, seals, and _____.

The woman walked across the street, checking for traffic over ___ shoulder.

I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____

What kinds of things does pretraining teach?

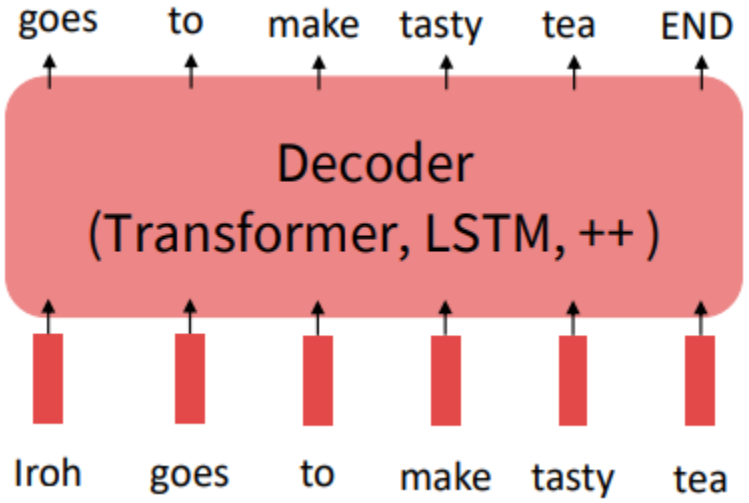
- There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language. Taking our examples from the start of class:

- Stanford University is located in _____, California. [Trivia]
- I put ___ fork down on the table. [syntax]
- The woman walked across the street, checking for traffic over ___ shoulder. [coreference]
- I went to the ocean to see the fish, turtles, seals, and _____. [lexical semantics/topic]
- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___. [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____ [some basic arithmetic; they don't learn the Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.

Pretraining through language modeling [Dai and Le, 2015]

- Recall the language modeling task:
  - Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
  - There's lots of data for this! (In English.)

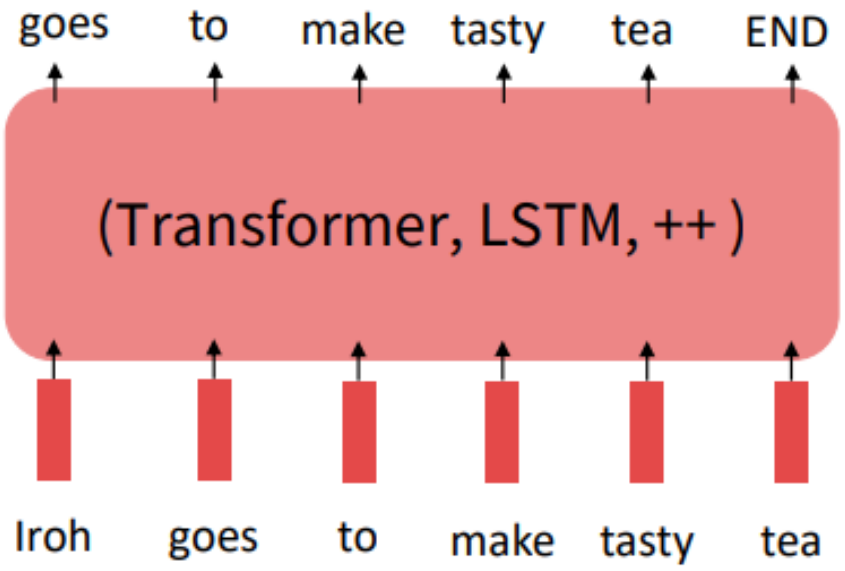
- Pretraining through language modeling:
  - Train a neural network to perform language modeling on a large amount of text.
  - Save the network parameters.



The Pretraining / Finetuning Paradigm

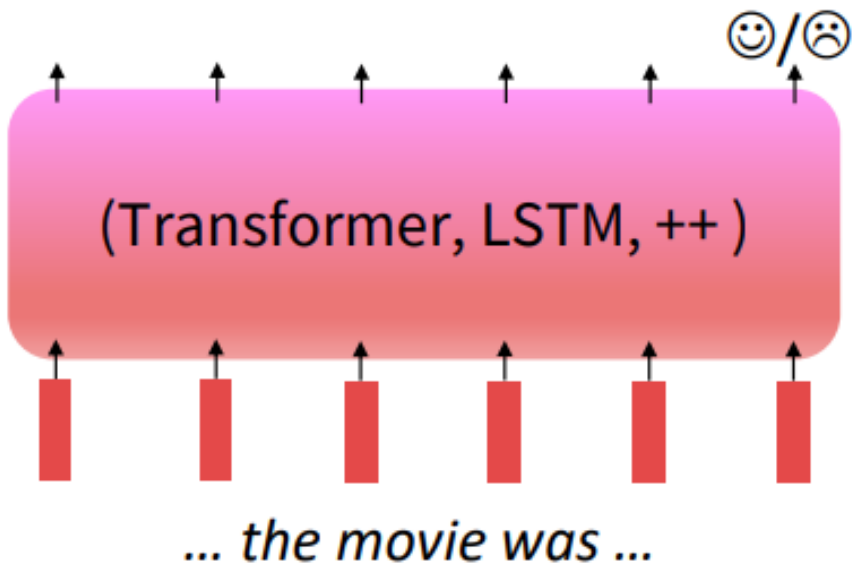
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

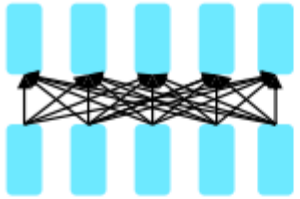
Not many labels; adapt to the task!



- Transformers' parallelizability allows for efficient pretraining, and have made them the de-facto standard.

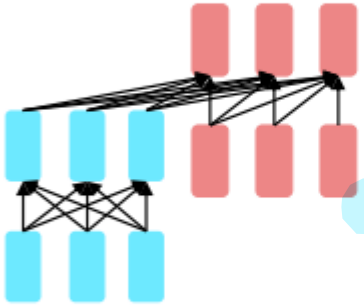
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



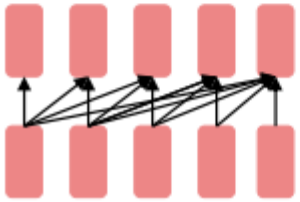
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

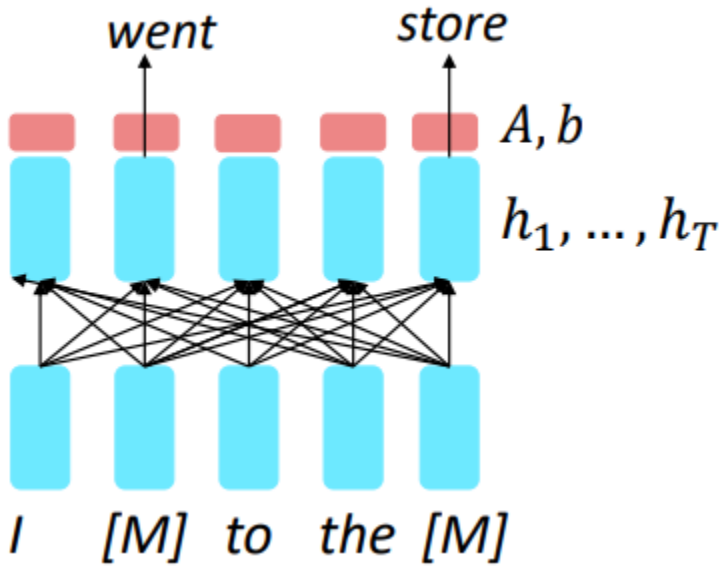
Pretraining encoders: what pretraining objective to use?

- So far, we've looked at language model pretraining. But **encoders get bidirectional context**

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

$$h_1, \dots, h_T = Encoder(w_1, \dots, w_T)$$

$$y_t \sim Ah_t + b$$



- 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**.

- Devlin et al., 2018 proposed the "Masked LM" objective.

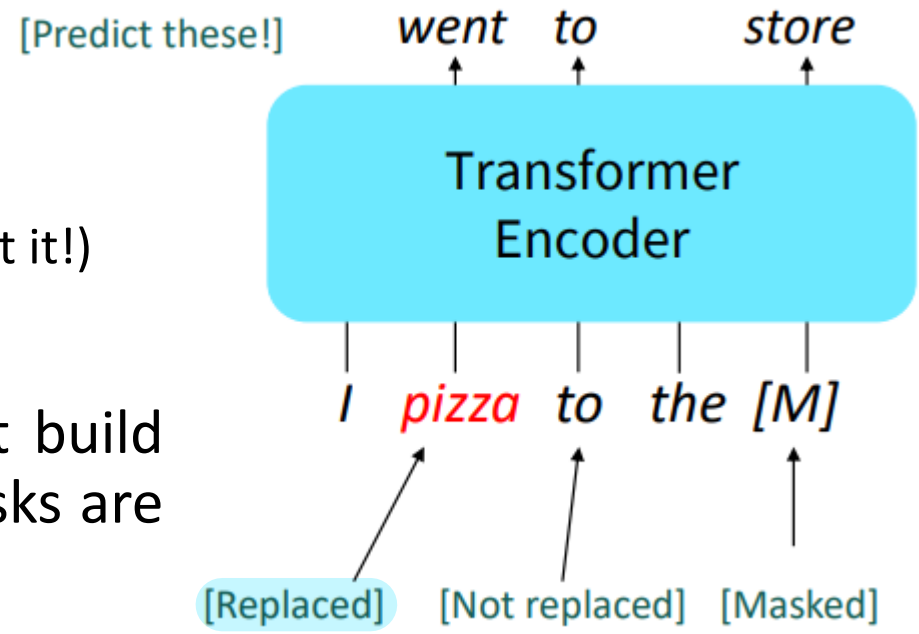
Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

BERT: Bidirectional Encoder Representations from Transformers

- **BERT:** a pretrained Transformer by a the "Masked LM" objective.

- 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!)



Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

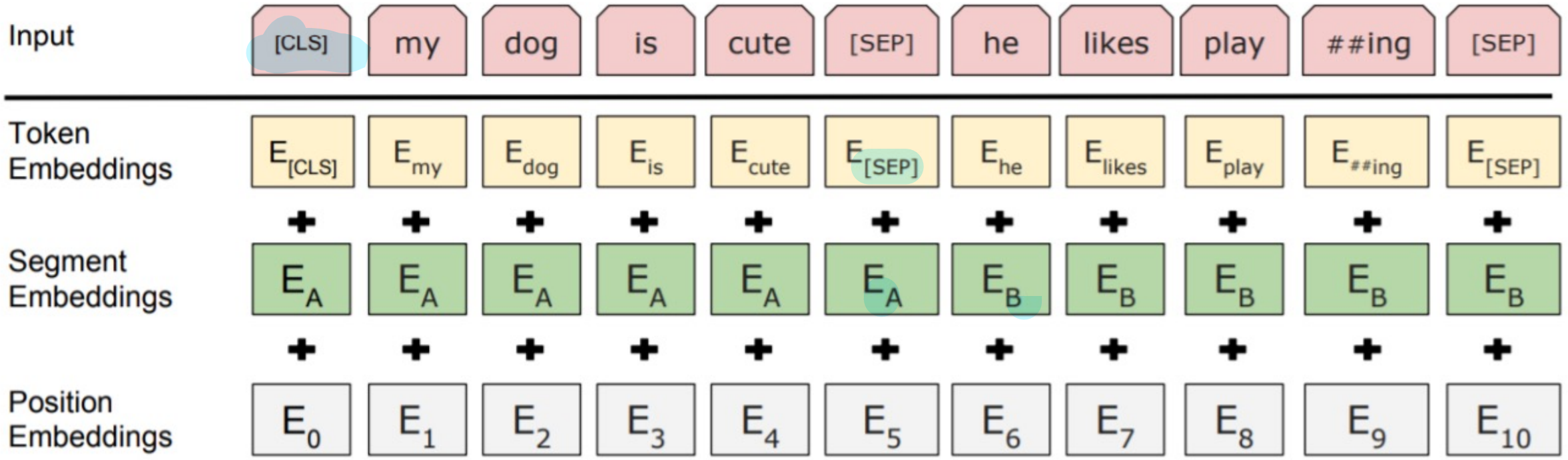
Pretraining BERT

- Task #1: Masked LM
- Task #2: Next Sentence Prediction (NSP)

Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

BERT: Pretraining by Next Sentence Prediction (NSP)

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



- BERT was also trained to predict whether one chunk follows the other or is randomly sampled.
- Later work has argued this "next sentence prediction" is not necessary.

Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

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

- "Small" models like BERT have become general tools in a wide range of settings

Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

BERT: Results

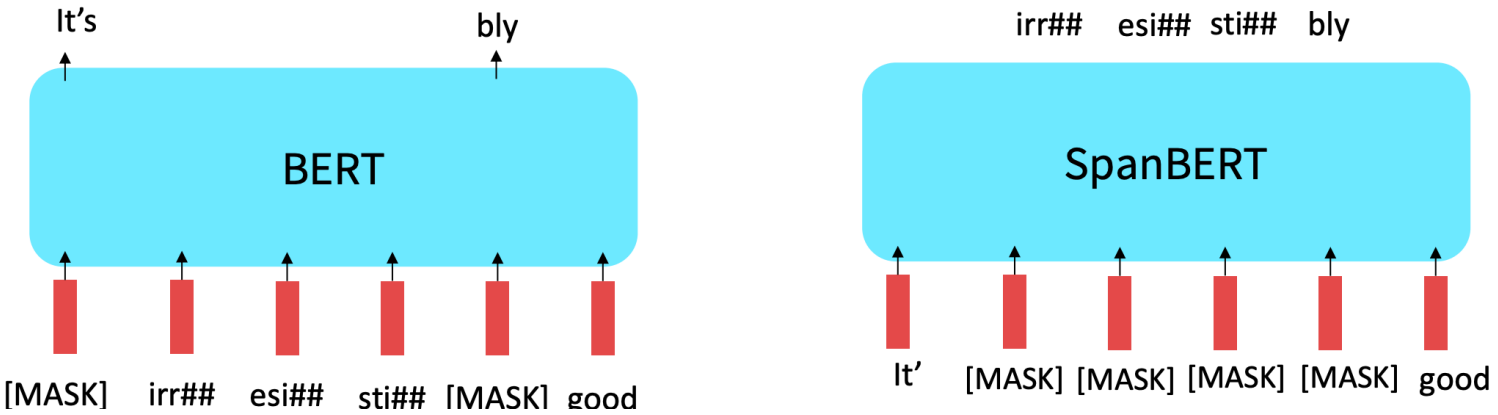
- BERT was massively popular and hugely versatile; finetuning BERT led to new state-of the-art results on a broad range of tasks.

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Devlin et al., BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

Extensions 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



[Liu et al., 2019; Joshi et al., 2020]

Extensions of BERT

- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!

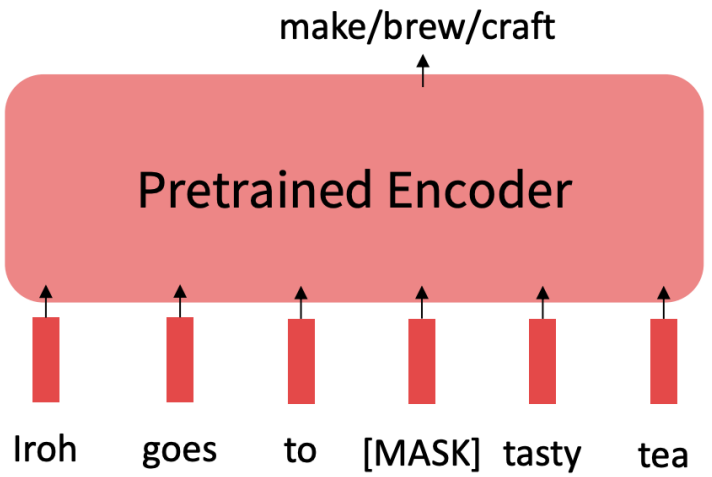
Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$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

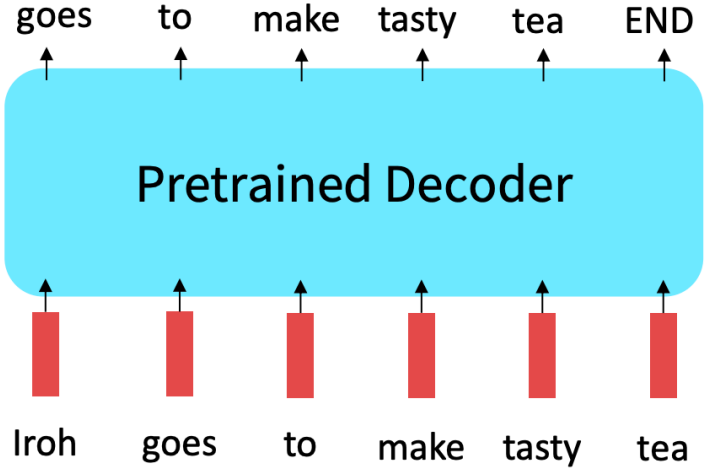
- A takeaway from the RoBERTa paper: more compute, more data can improve pretraining even when not changing the underlying Transformer encoder.

Limitations of pretrained encoders

- Those results looked great! Why not used pretrained encoders for everything?

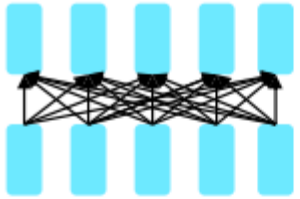
- If your task involves generating sequences, consider using a pretrained decoder; BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods.





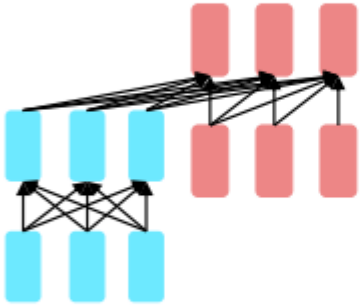
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



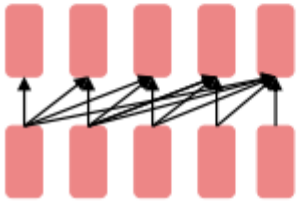
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



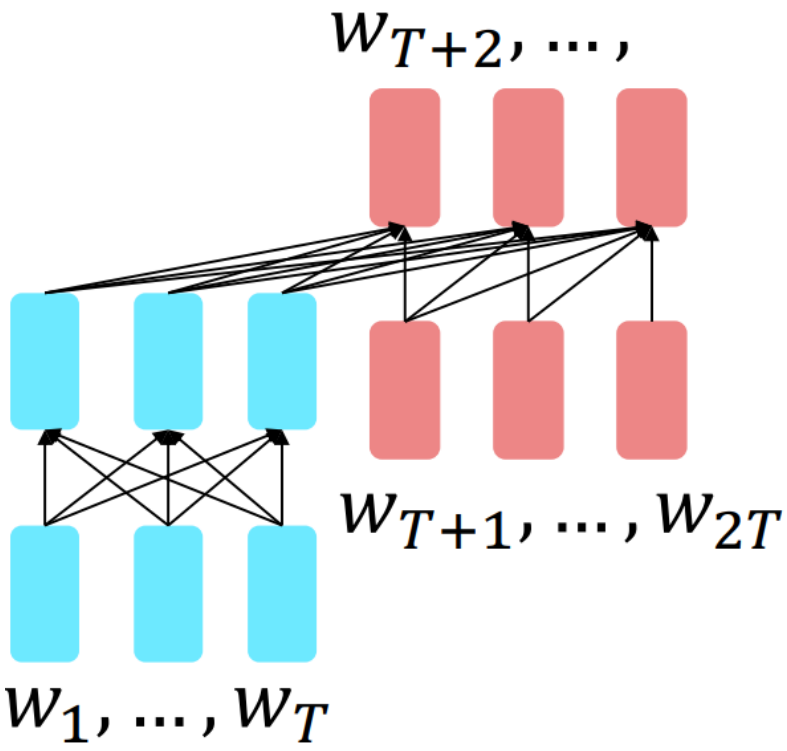
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining encoder-decoders: what pretraining objective 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.

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



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

Pretraining encoder-decoders: what pretraining objective to use?

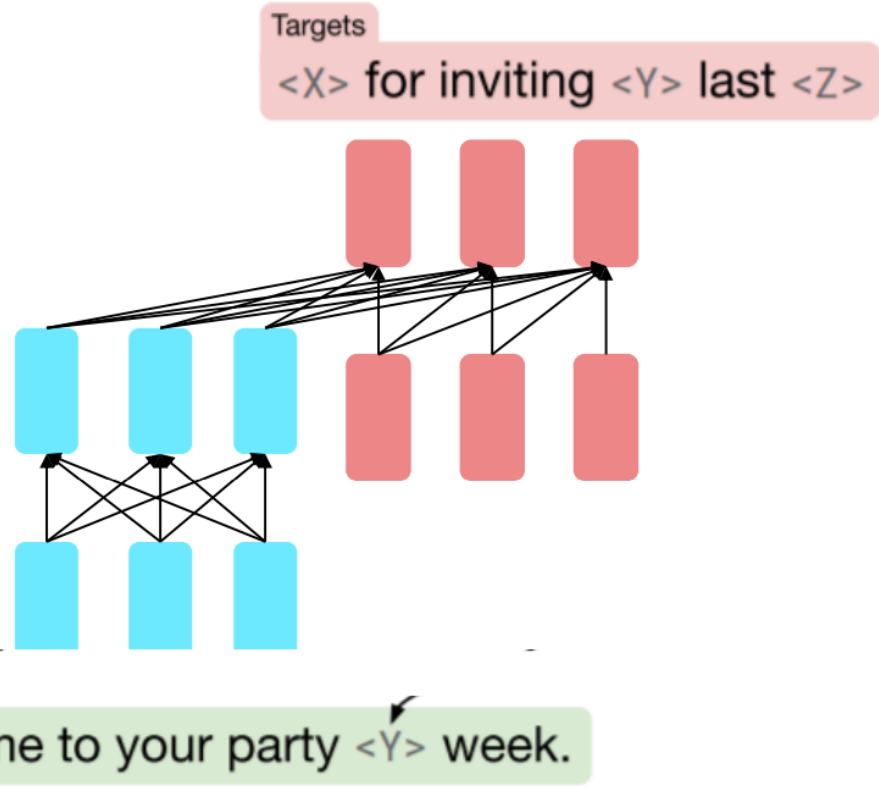
- **Text-to-Text Transfer Transformer (T5)**
  - Every task – including translation, question answering, and classification – is cast to text-to-text.
- **T5 model:** span corruption works best.

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

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

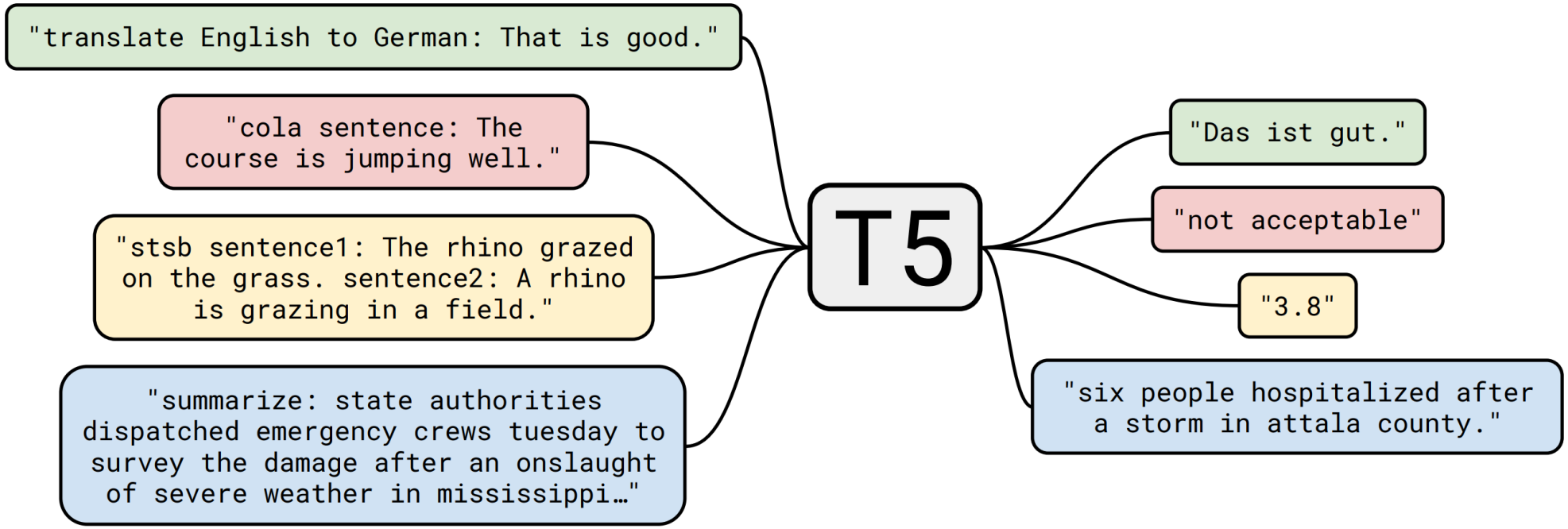
- This is implemented in text preprocessing: it's still an objective that looks like language modeling at the decoder side.



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

T5

- Text-to-text framework casts multiple NLP tasks into a common format
  - Every task is cast as feeding the input and training it to generate target text
- The same model, loss function, hyperparameters, etc can be used across diverse set of tasks
- Provides a consistent training objective both for pre-training and fine-tuning



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

Pretraining dataset

- They leveraged Common Crawl as a source of text scraped from the web
- They prepared Colossal Clean Crawled Corpus (C4) by applying several heuristics for cleaning up Common Crawl

Pretraining encoder-decoders: what pretraining objective to use?

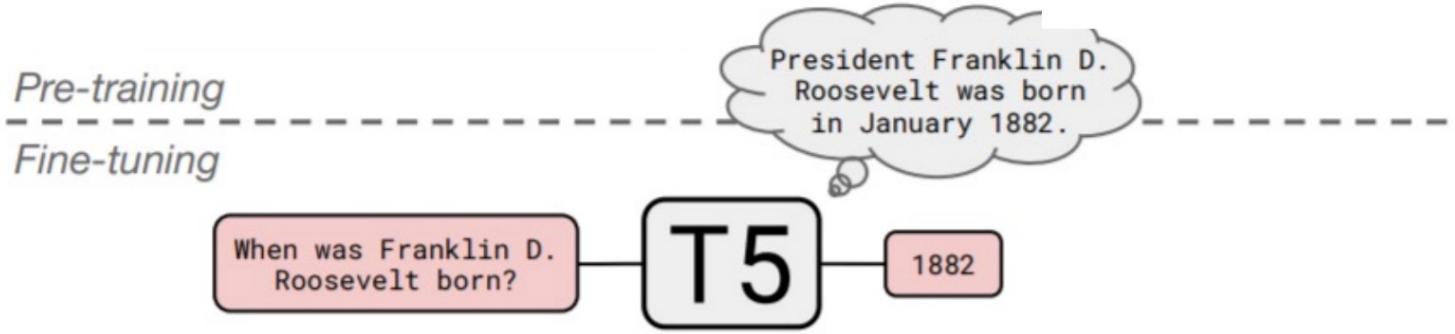
- It found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

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

Pretraining encoder-decoders: what pretraining objective to use?

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



All "open-domain" versions

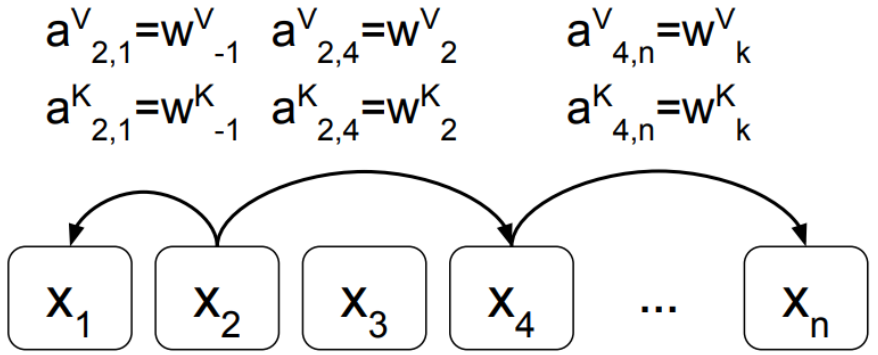
	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL +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

NQ: Natural Questions
WQ: WebQuestions
TQA: Trivia QA

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

Relative Position Representations

- Relative position embeddings produce a different learned embedding according to the offset between the "key" and "query"



$$a^K_{ij} = w^K_{\text{clip}(j-i,k)}$$
$$a^V_{ij} = w^V_{\text{clip}(j-i,k)}$$
$$\text{clip}(x,k) = \max(-k, \min(k,x))$$

We then learn relative position representations $w^K = (w^K_{-k}, \ldots, w^K_k)$ and $w^V = (w^V_{-k}, \ldots, w^V_k)$ where $w^K_i, w^V_i \in \mathbb{R}^{d_a}$.

Shaw et al., Self-Attention with Relative Position Representations, NAACL 2018

Relative Position Representations

- $e_{ij} = \frac{x_i^T W_q (W_k^T x_j + a_{ij}^K)}{\sqrt{d}}$
- $\alpha_{ij} = \frac{e^{e_{ij}}}{\sum_{j'=1}^{T} e^{e_{ij'}}}$
- $z_i = \sum_{j=1}^{n} \alpha_{ij} (W_v^T x_j + a_{ij}^V)$

a_{ij}^V	a_{ij}^K	EN-DE BLEU
Yes	Yes	25.8
No	Yes	25.8
Yes	No	25.3
No	No	12.5

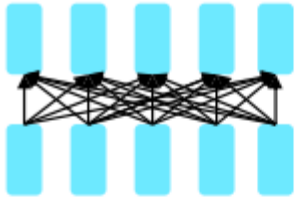
Shaw et al., Self-Attention with Relative Position Representations, NAACL 2018

Positional Embedings

- Absolute PE
- Relative PE
- Rotary PE
- Attention with Linear Biases (ALiBi)

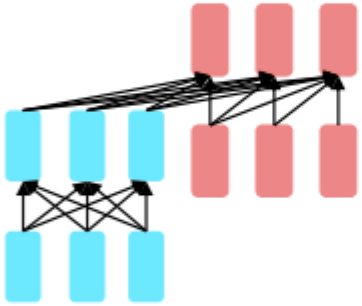
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



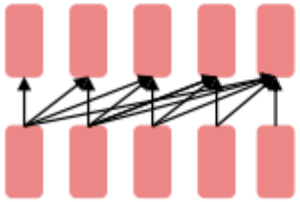
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

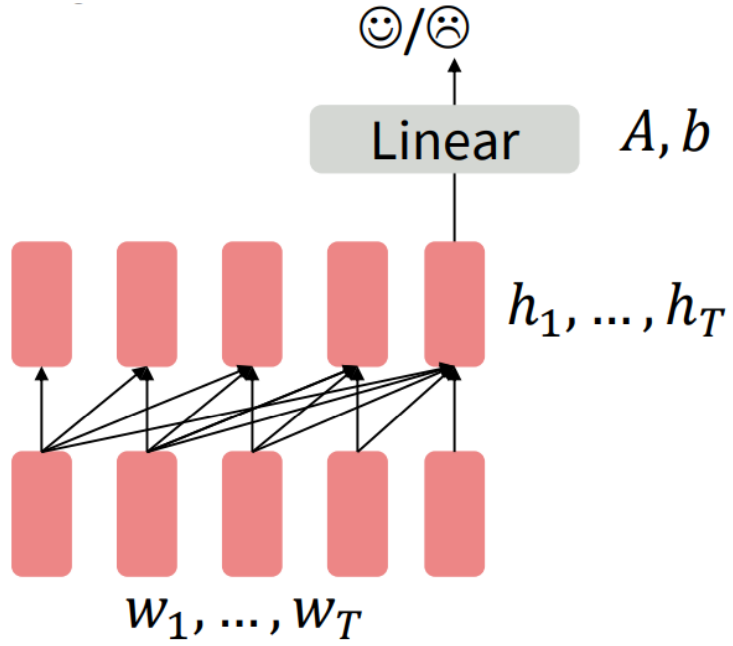
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Pretraining decoders

- When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_1:w_{t-1})$.
- We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$y \sim Ah_T + b$$

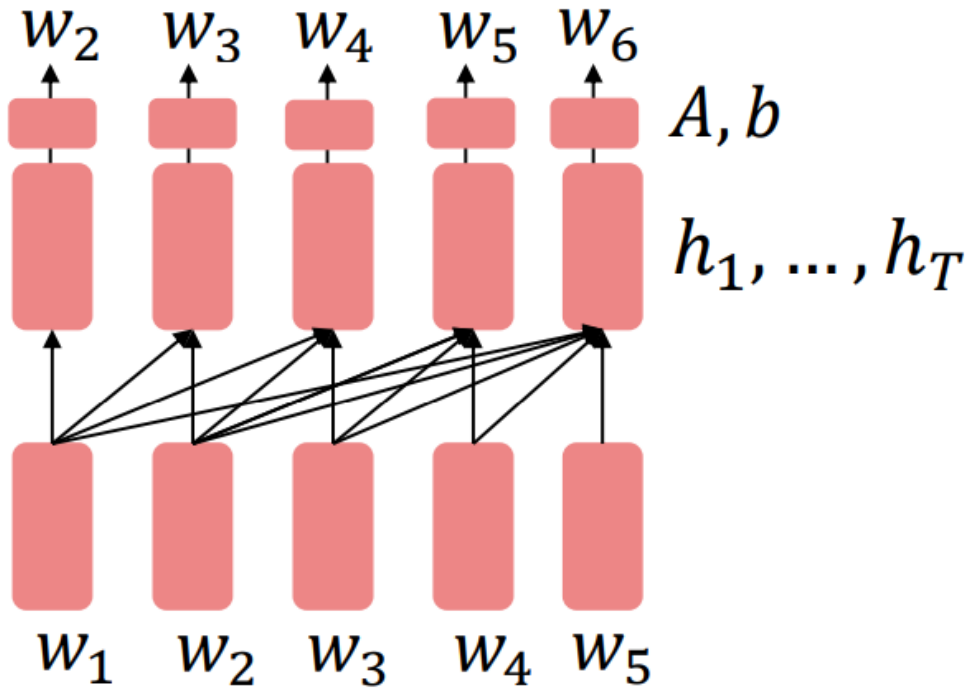
- Where A and b are randomly initialized and specified by the downstream task.
- Gradients backpropagate through the whole network.



[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

- It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!
- This is helpful in tasks where the output is a sequence with a vocabulary like that at pretraining time!
- Where A, b were pretrained in the language model



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer (GPT)

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

- Transformer decoder with 12 layers, 117M parameters.
  - 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.

- Showed LM at scale can be an effective pretraining technique for downstream tasks (like Natural Language Inference)

Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Generative Pretrained Transformer (GPT)

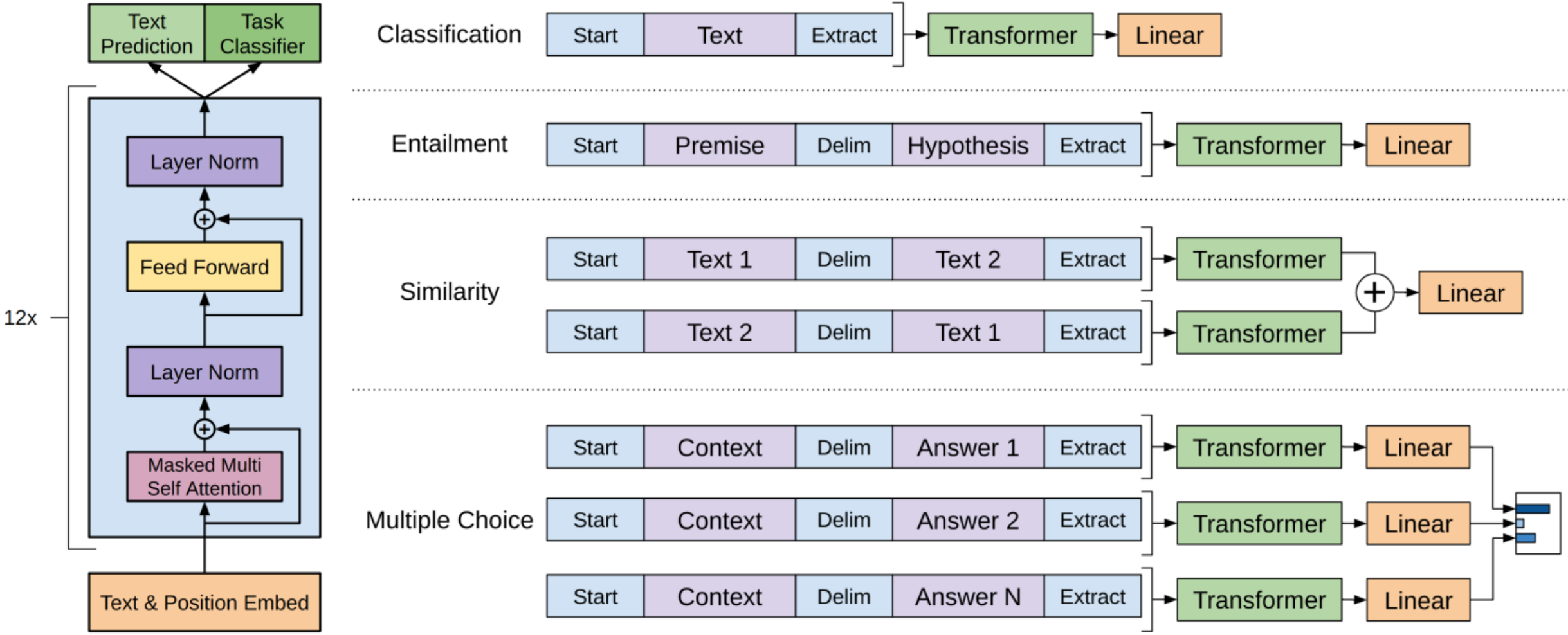
- How do we format inputs to our decoder for finetuning tasks?
- Natural Language Inference: Label pairs of sentences as entailing/contradictory/neutral
  - Premise: The man is in the doorway
  - Hypothesis: The person is near the door

- Radford et al., 2018 evaluate on natural language inference. Here's roughly how the input was formatted, as a sequence of tokens for the decoder.
  - The linear classifier is applied to the representation of the [EXTRACT] token.

  [START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]

Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Generative Pretrained Transformer (GPT)



Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Generative Pretrained Transformer (GPT)

- GPT results on various natural language inference datasets.

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Method	**Classification**		**Semantic Similarity**			**GLUE**
	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	
Sparse byte mLSTM [16]	-	**93.2**	-	-	-	-
TF-KLD [23]	-	-	**86.0**	-	-	-
ECNU (mixed ensemble) [60]	-	-	-	81.0	-	-
Single-task BiLSTM + ELMo + Attn [64]	35.0	90.2	80.2	55.5	66.1	64.8
Multi-task BiLSTM + ELMo + Attn [64]	18.9	91.6	83.5	72.8	63.3	68.9
Finetuned Transformer LM (ours)	**45.4**	91.3	82.3	**82.0**	**70.3**	**72.8**

Radford et al., Improving Language Understanding by Generative Pre-Training, 2018

Increasingly convincing generations (GPT2)

- Pretrained decoders can be used in their capacities as language models.
- GPT-2, a larger version (**117M**->**1.5B**) of GPT
  - trained on much more data (**4GB**->**40GB**)
  - Scrape links posted on Reddit w/ at least 3 upvotes (rough proxy of human quality)
- It was shown to produce relatively convincing samples of natural language.

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

Increasingly convincing generations (GPT2)

- Pretrained decoders can be used in their capacities as language models.
- GPT-2, a larger version (**117M->1.5B**) of GPT
  - trained on much more data (**4GB->40GB** of WebText)
  - Scrape links posted on Reddit w/ at least 3 upvotes (rough proxy of human quality)
- It was shown to produce relatively convincing samples of natural language.

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]

Emergent abilities of large language models: GPT-3

- GPT-3 (175B parameters; Brown et al., 2020)
- Another increase in size (**1.5B** -> **175B**)
- and data (**40GB** -> over **600GB**)

Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Brown et al., Language Models are Few-Shot Learners, 2020

GPT-3, In-context learning, and very large models

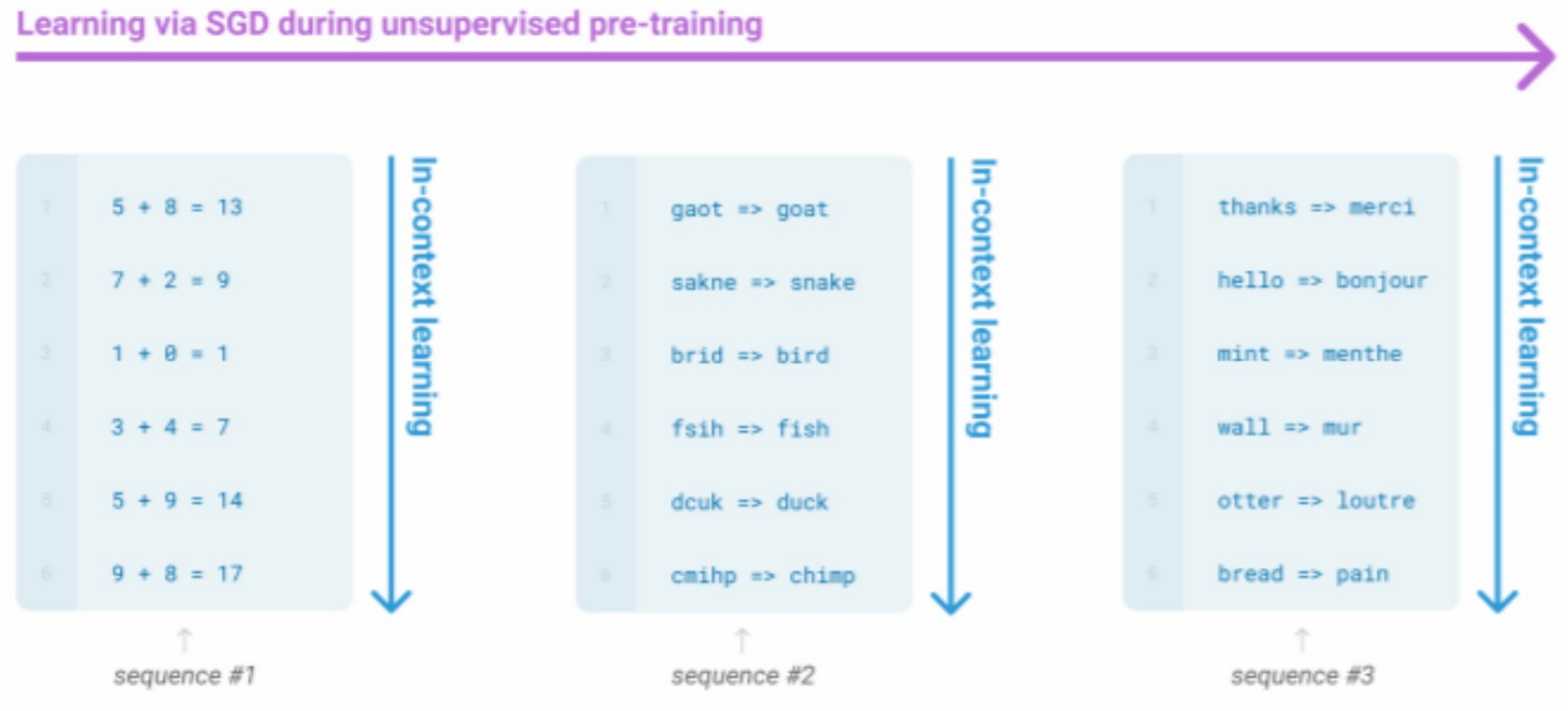
- So far, we've interacted with pretrained models in two ways:
  - Sample from the distributions they define (maybe providing a prompt)
  - Fine-tune them on a task we care about, and take their predictions.

- 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.

GPT-3, In-context learning, and very large models

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



Brown et al., Language Models are Few-Shot Learners, 2020

GPT3: few-shot and zero-shot learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Brown et al., Language Models are Few-Shot Learners, 2020

Scaling Efficiency: how do we best use our compute

- GPT-3 was 175B parameters and trained on 300B tokens of text.
- Roughly, the cost of training a large transformer scales as **parameters*tokens**
- Did OpenAI strike the right parameter-token data to get the best model? No.

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

This 70B parameter model is better than the much larger other models!

Brown et al., Language Models are Few-Shot Learners, 2020

Image Captioning using Transformers

Input: Image **I**

Output: Sequence $\boldsymbol{y} = y_1, y_2, \ldots y_T$

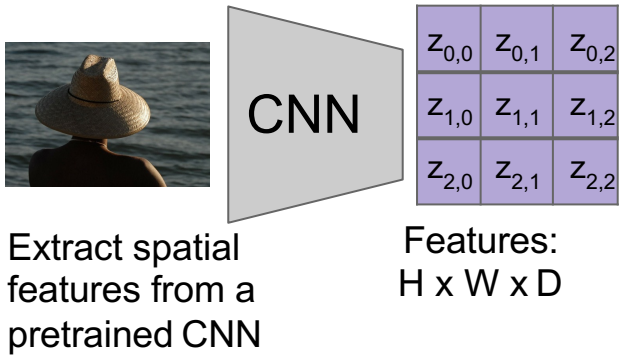


Image Captioning using Transformers

Input: Image **I**

Output: Sequence $\boldsymbol{y} = y_1, y_2, \ldots y_T$

Encoder: $c = T_{\boldsymbol{w}}(\boldsymbol{z})$

where **z** is spatial CNN features

$T_{\boldsymbol{w}}(.)$ is the transformer encoder

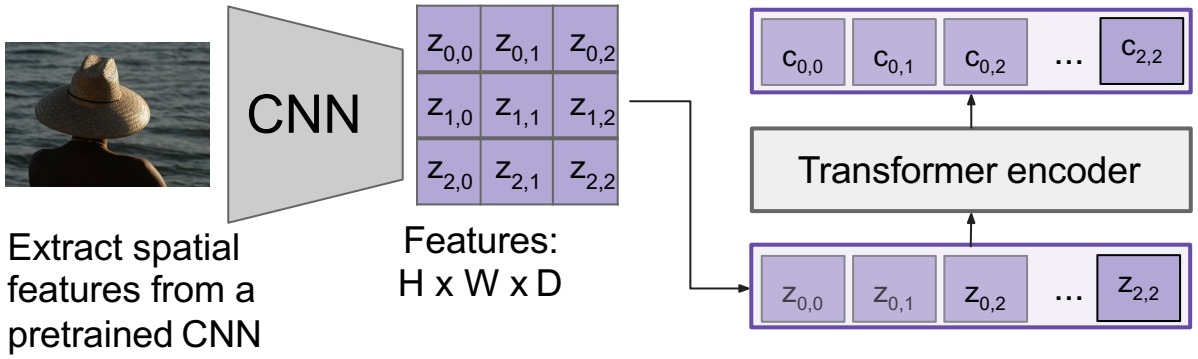


Image Captioning using Transformers

Input: Image **I**

Output: Sequence $\boldsymbol{y} = y_1, y_2, \ldots y_T$

Decoder: $y_t = T_D(y_{0:t-1}, c)$
Where $T_D(.)$ is the transformer decoder

Encoder: $c = T_{\boldsymbol{w}}(\boldsymbol{z})$

where **z** is spatial CNN features

$T_{\boldsymbol{w}}(.)$ is the transformer encoder

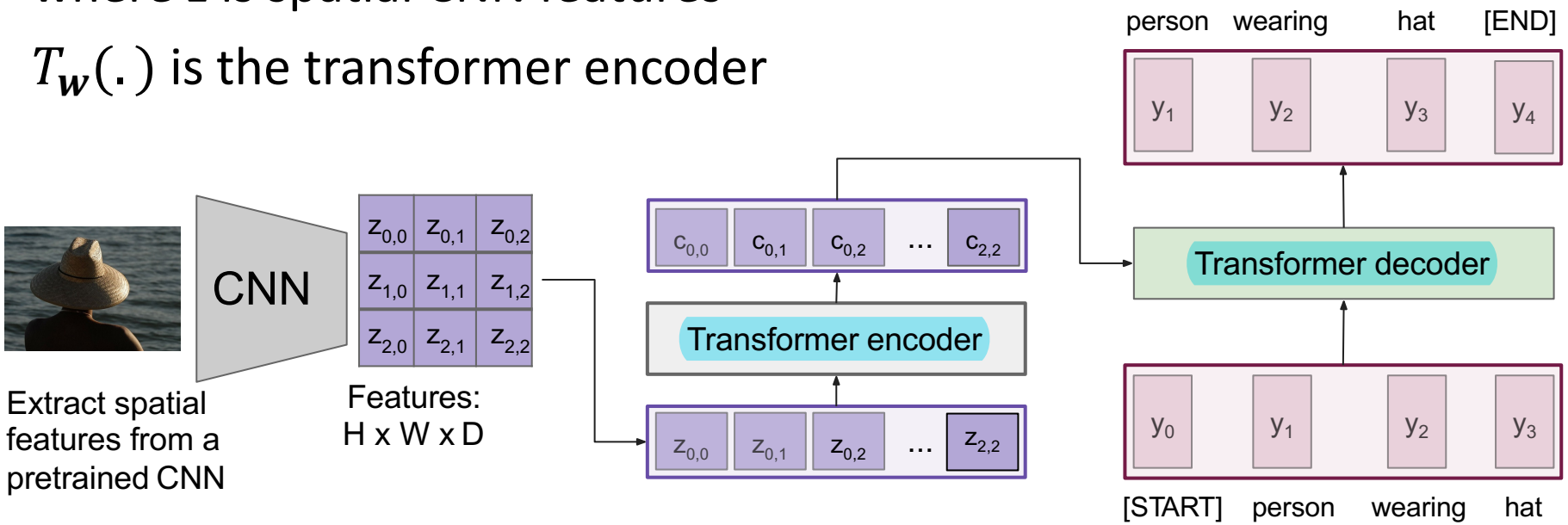


Image Captioning using transformers

No recurrence at all

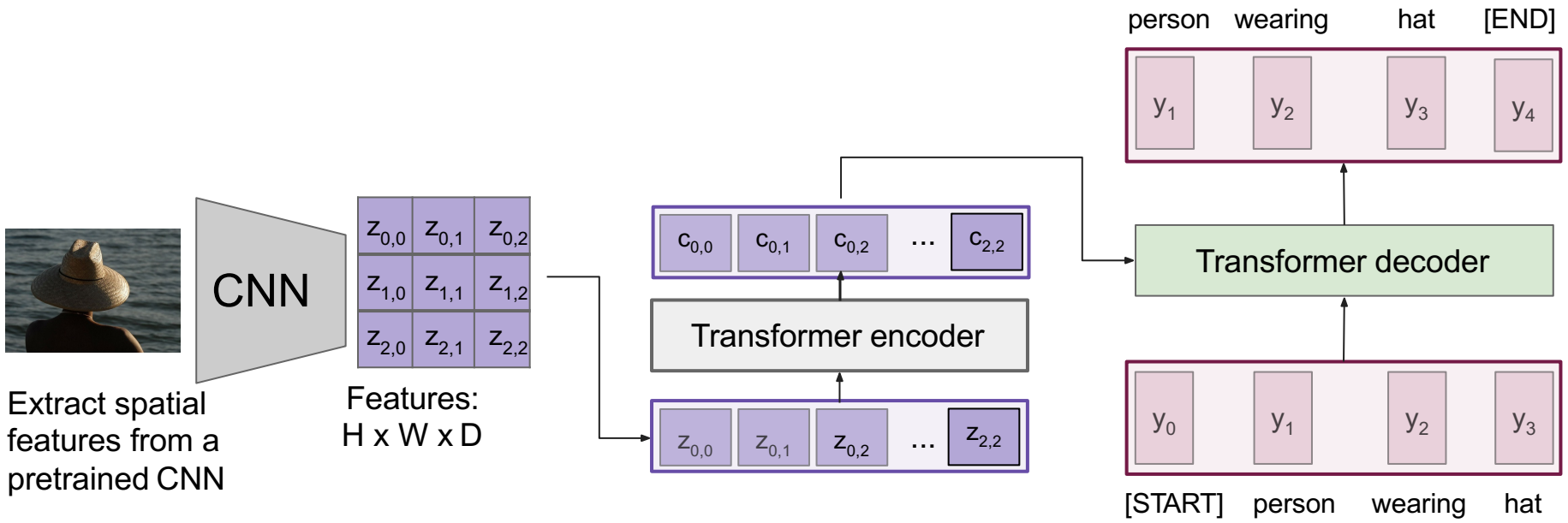


Image Captioning using transformers

Perhaps we don't need convolutions at all?

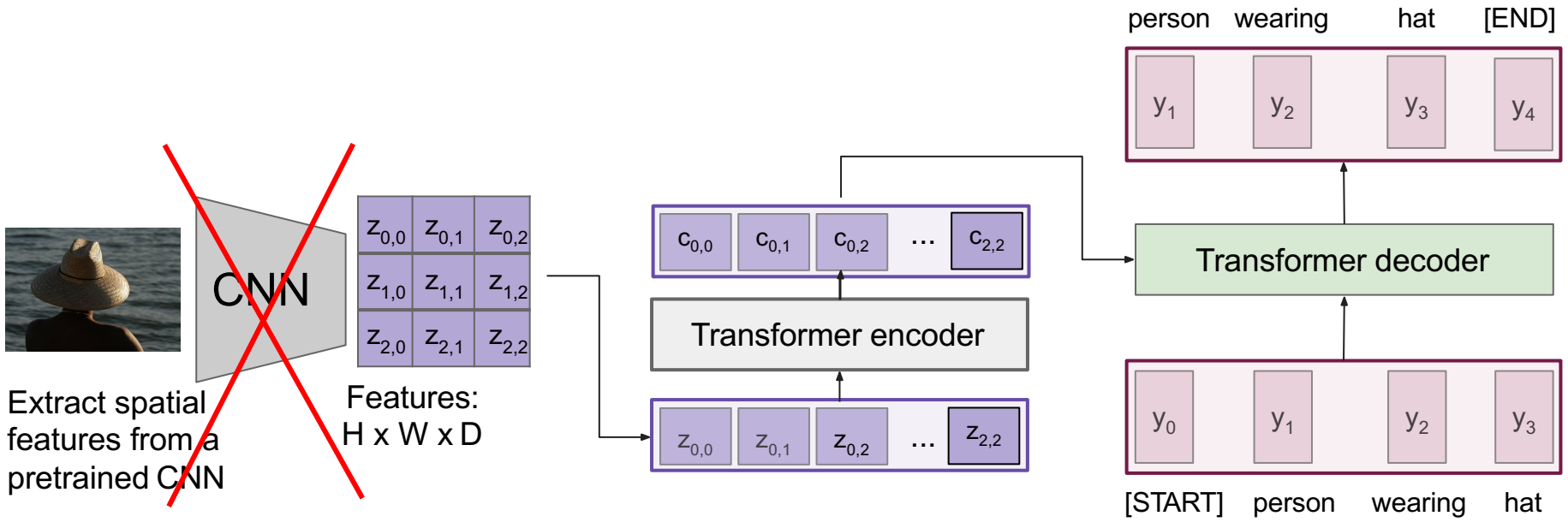
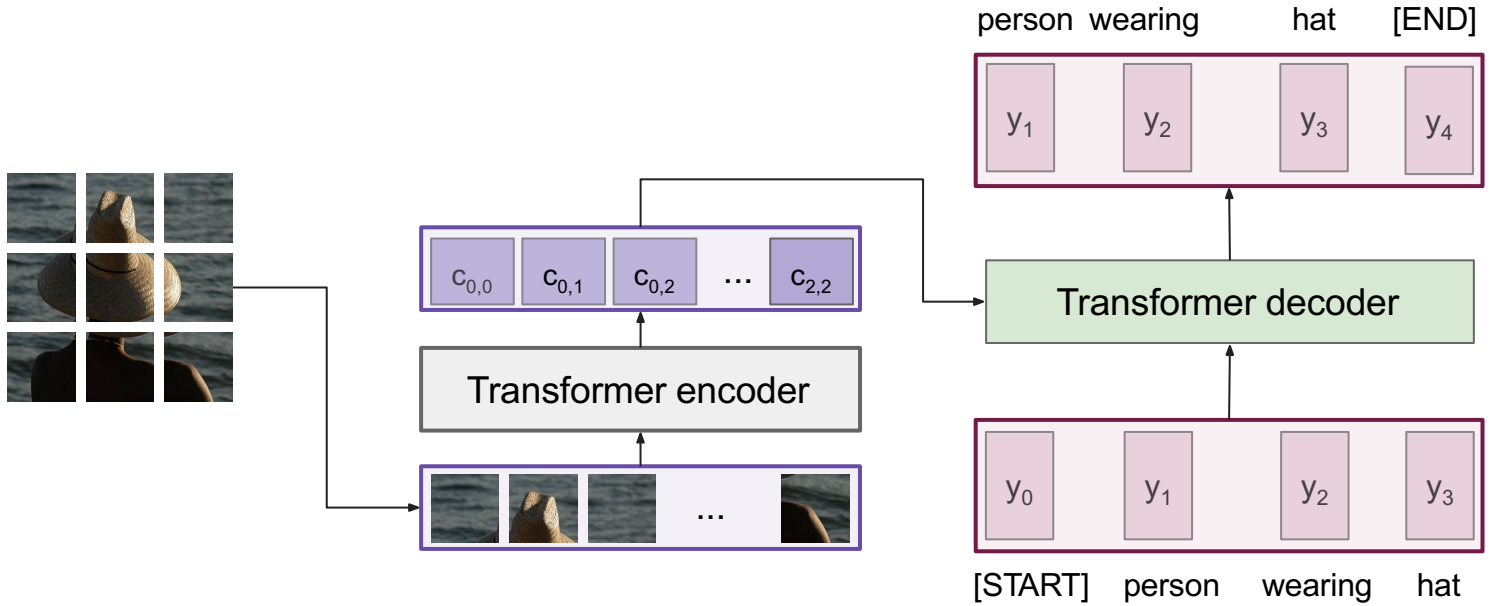


Image Captioning using ONLY transformers

Transformers from pixels to language

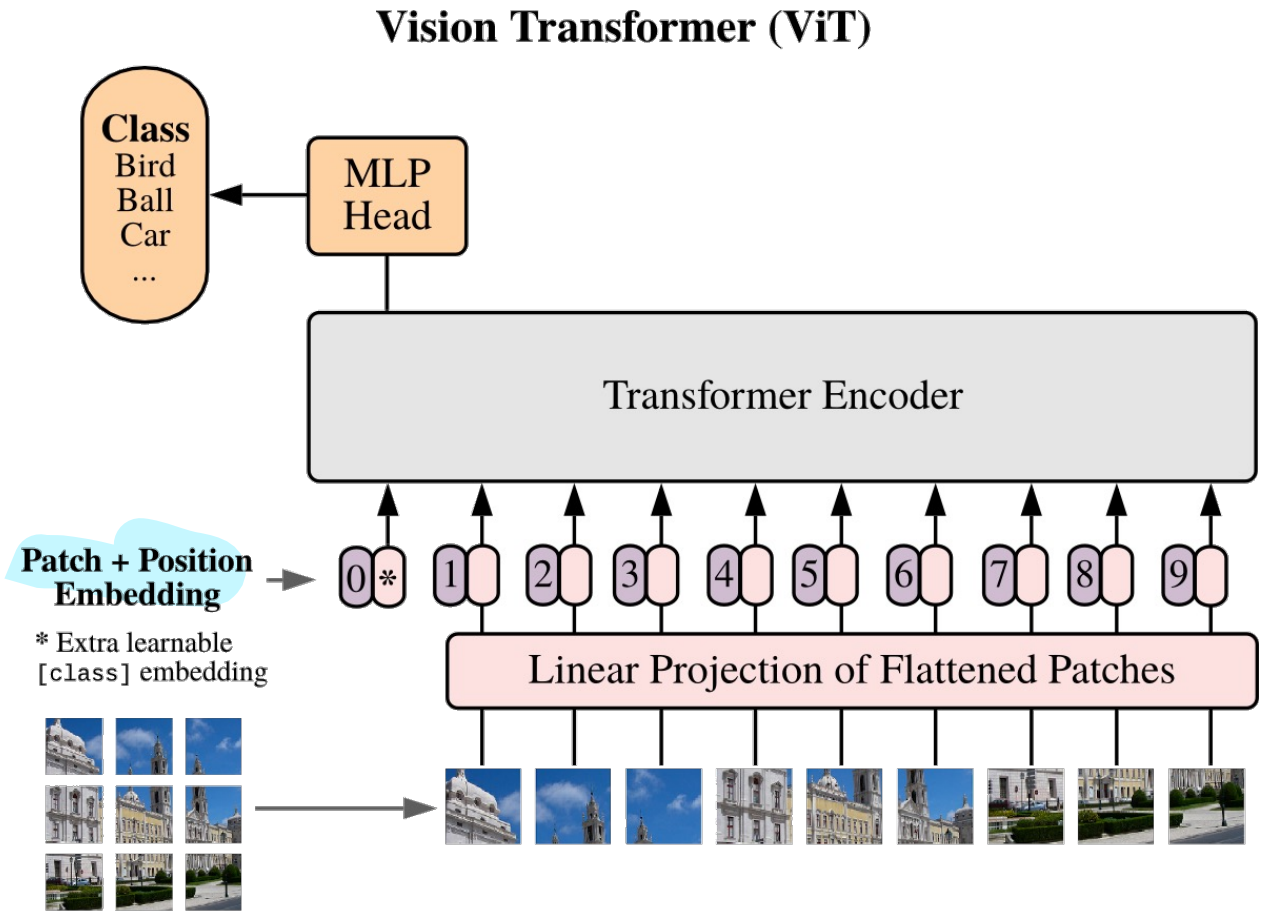


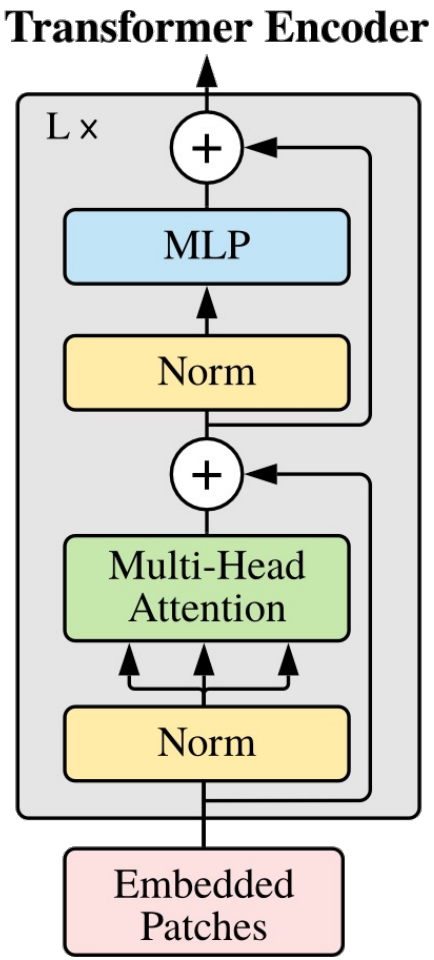
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020
Colab link to an implementation of vision transformers

ViT: Input

- Convert an image of size $H \times W$ to $N = (H \times W)/P^2$ of size P^2
- Patch i is reshaped to a row vector of size $P^2 \times C$ called x_p^i
- Patch is transformed to a d-dimensional vector using the embedding matrix E as $x_p^i E$

ViT: Architecture





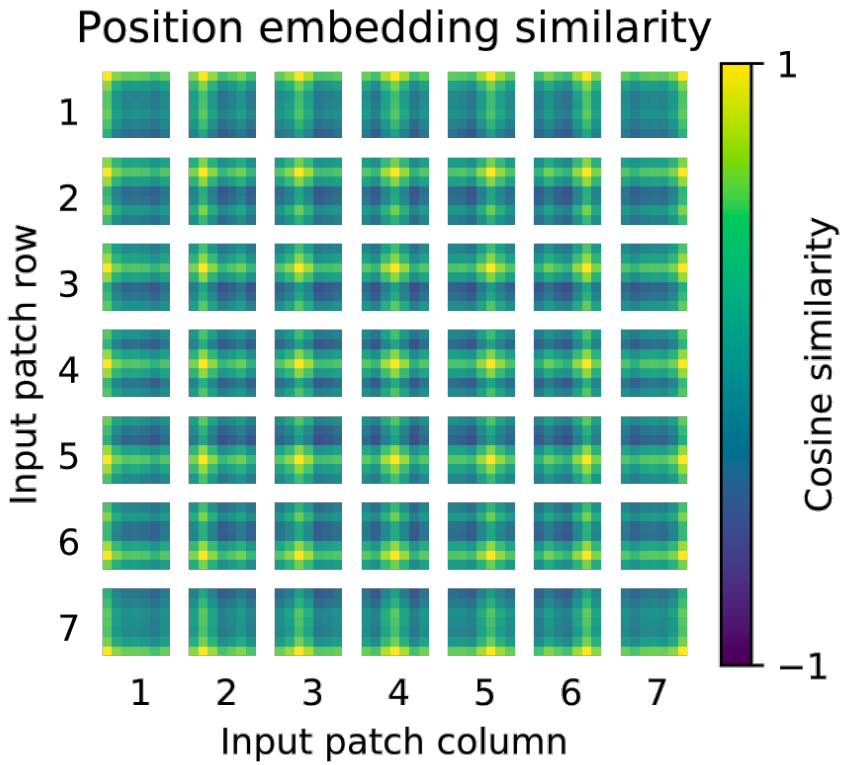
Only MLP layers are local and translationally equivariant, while the self-attention layers are global.

Alexey Dosovitskiy et al., An Image Is Worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

ViT: Positional Embeddings

- Position encodings are added to the patch embeddings

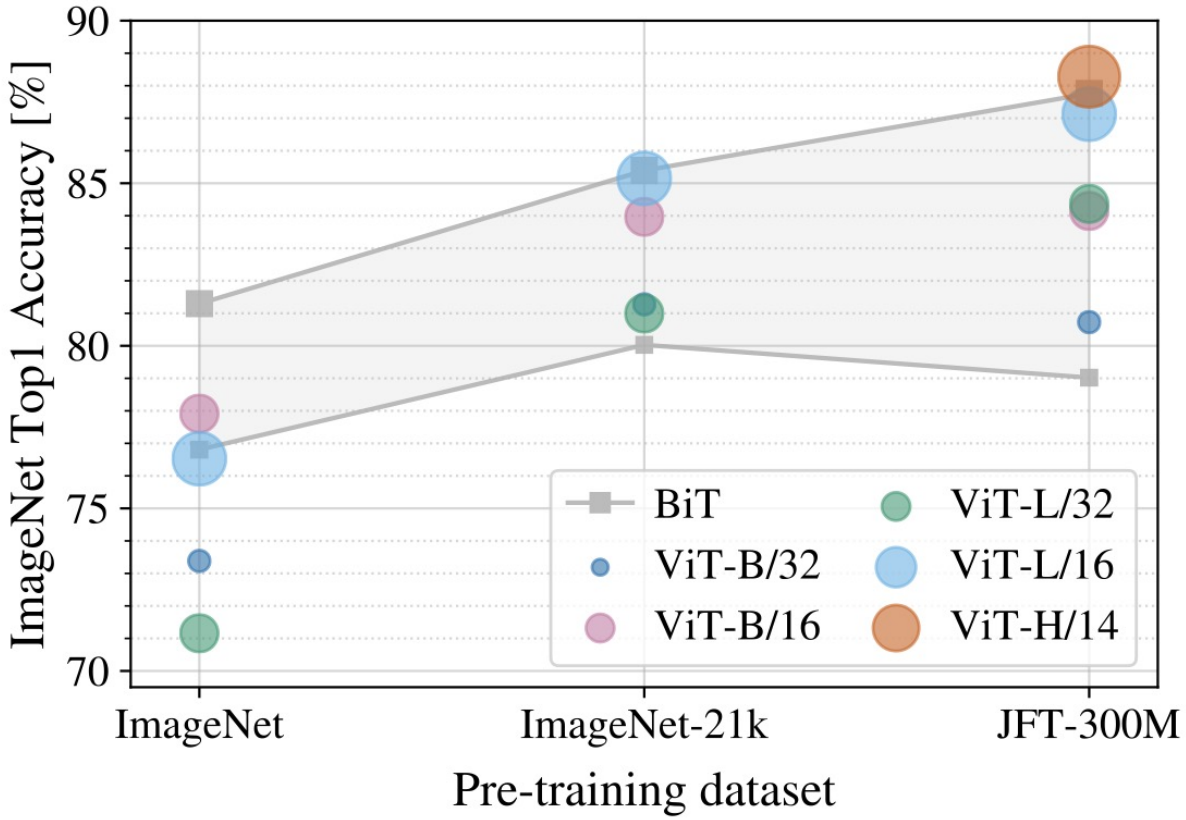
- Standard learnable 1D position embeddings
  - not observed significant performance gains from using more advanced 2D-aware position embeddings.

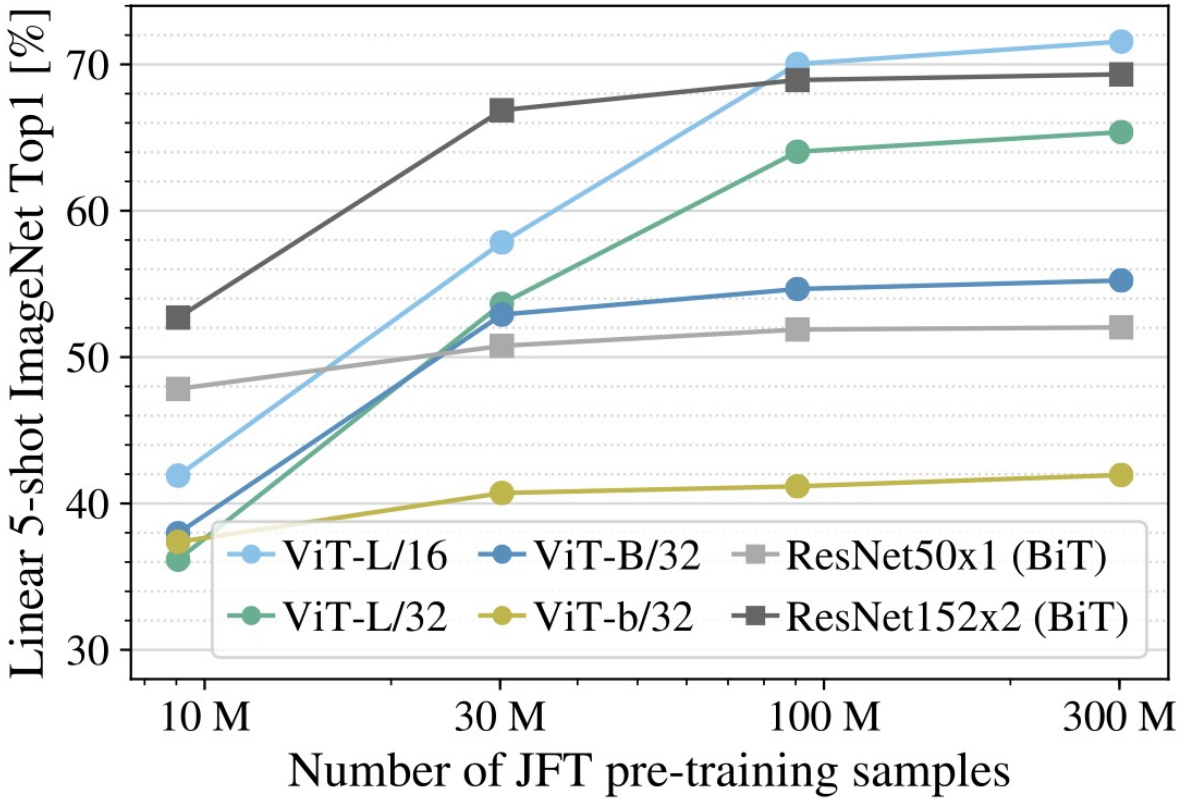


$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

Alexey Dosovitskiy et al., An Image Is Worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

Vision Transformers vs. ResNets





Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Alexey Dosovitskiy et al., An Image Is Worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

Vision Transformers vs. ResNets

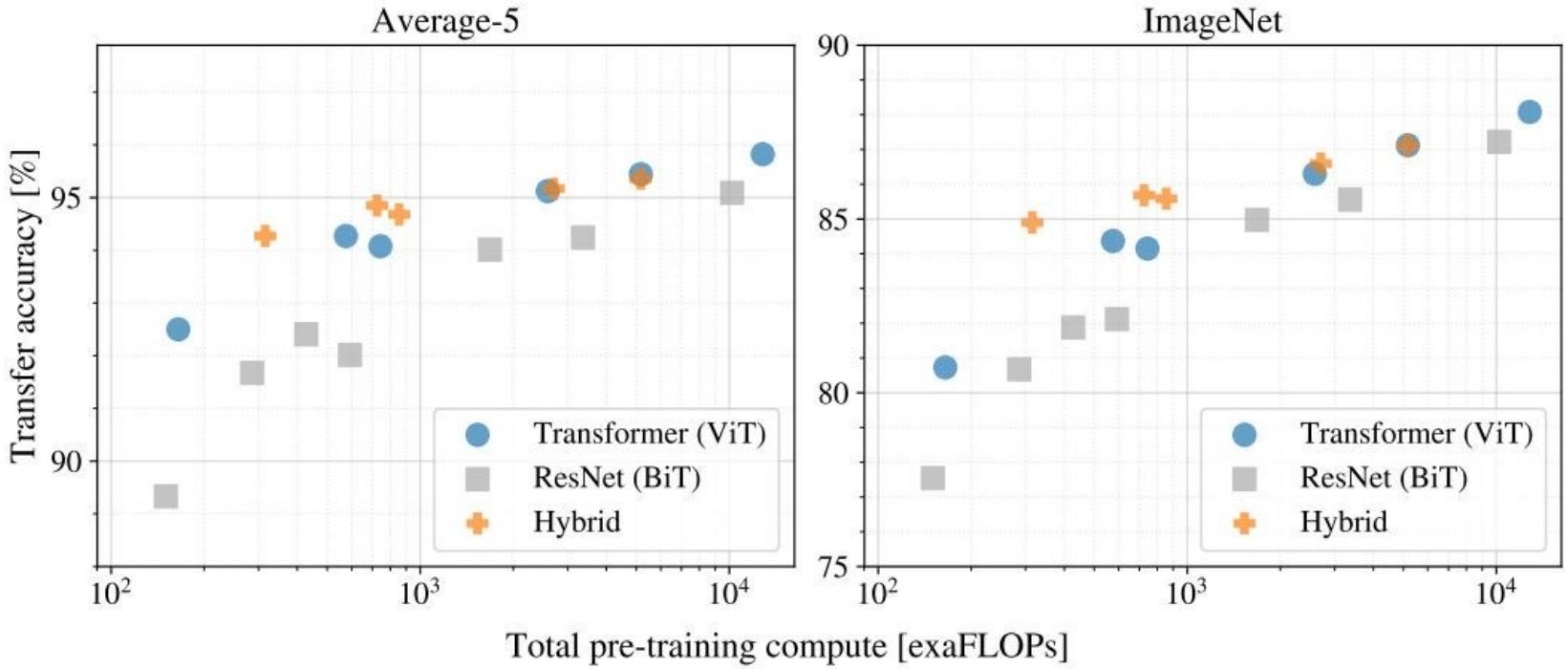
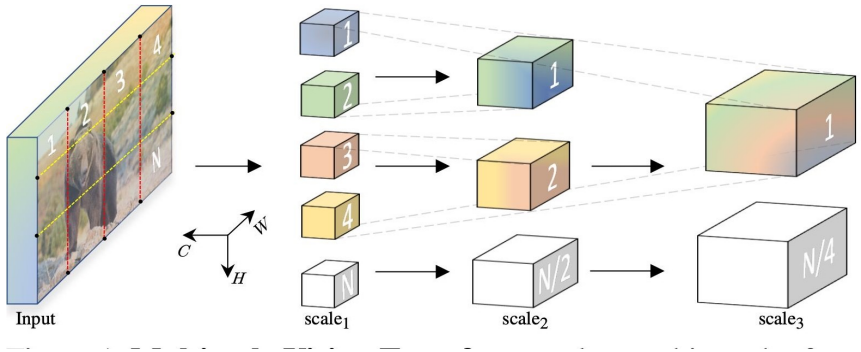


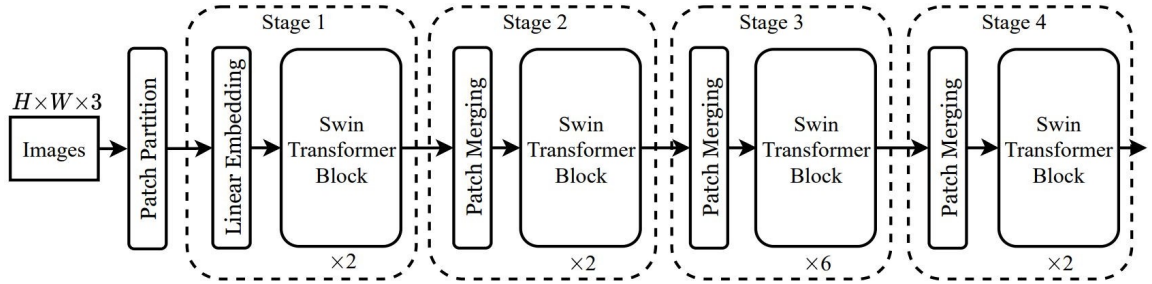
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Alexey Dosovitskiy et al., An Image Is Worth 16X16 Words: Transformers for Image Recognition at Scale, ICLR 2021.

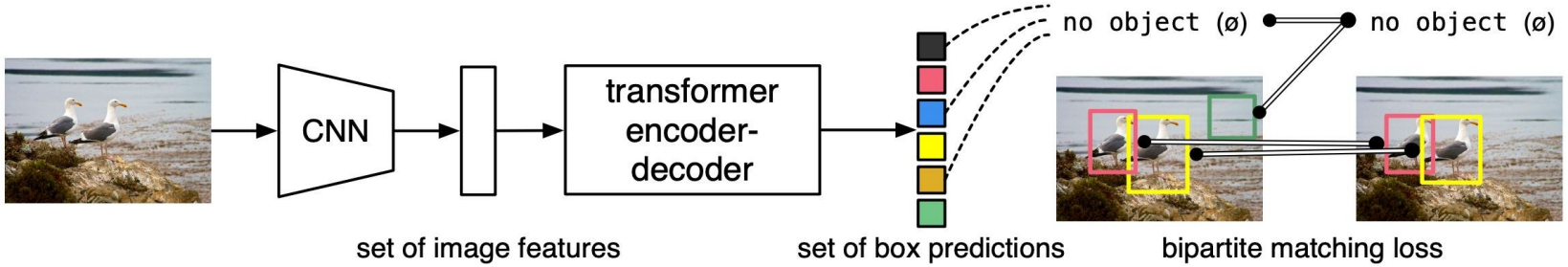
Vision Transformers



Fan et al, “Multiscale Vision Transformers”, ICCV 2021



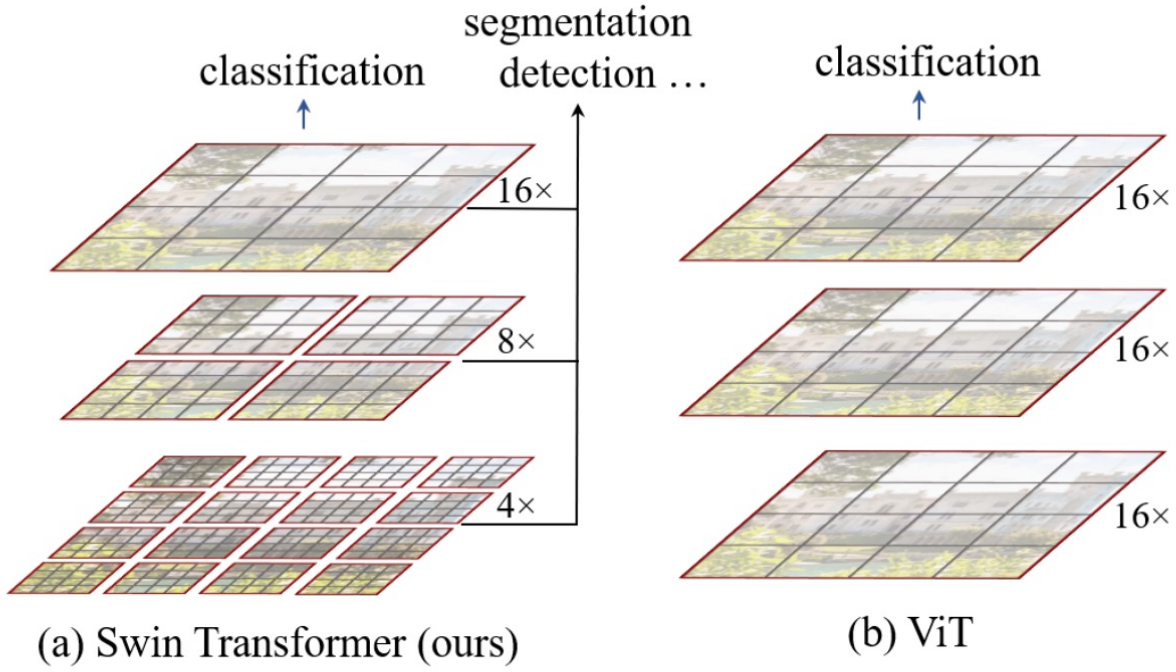
Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021



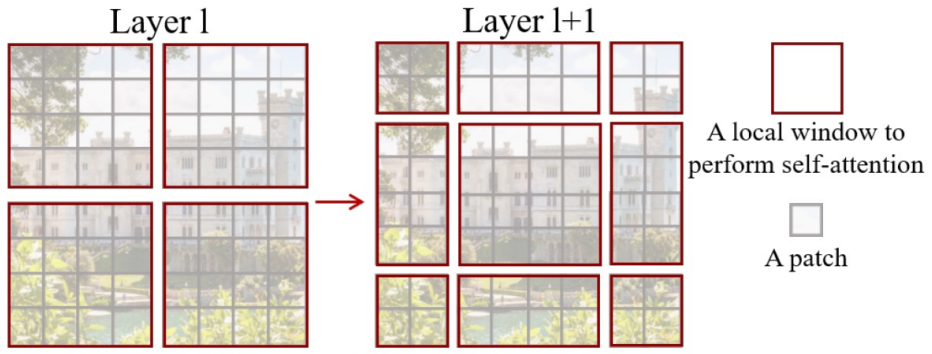
Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

Swin Transformer

- General-purpose backbone for computer vision
- Adapting Transformer from language to vision by considering their differences:
  - large variations in the scale of visual entities
  - high resolution of pixels compared to words
- Hierarchical Transformer with Shifted windows
  - starting from small-sized patches and gradually merging neighboring patches in deeper layers.
  - self-attention only within each local window (shown in red).

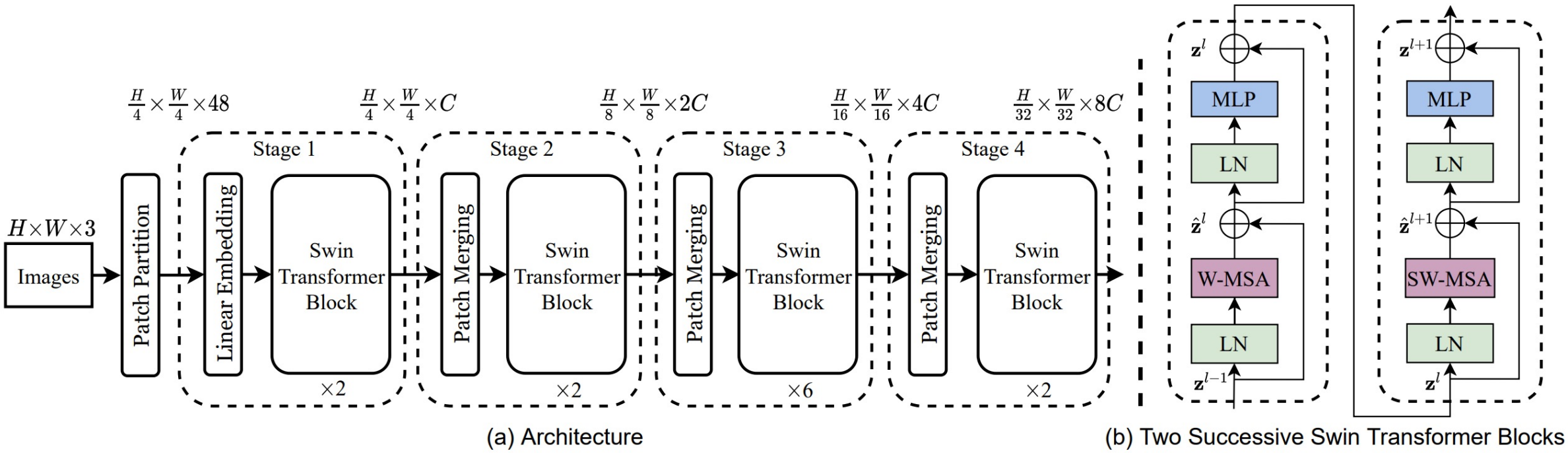


Shifted window approach



Liu et al., Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, 2021

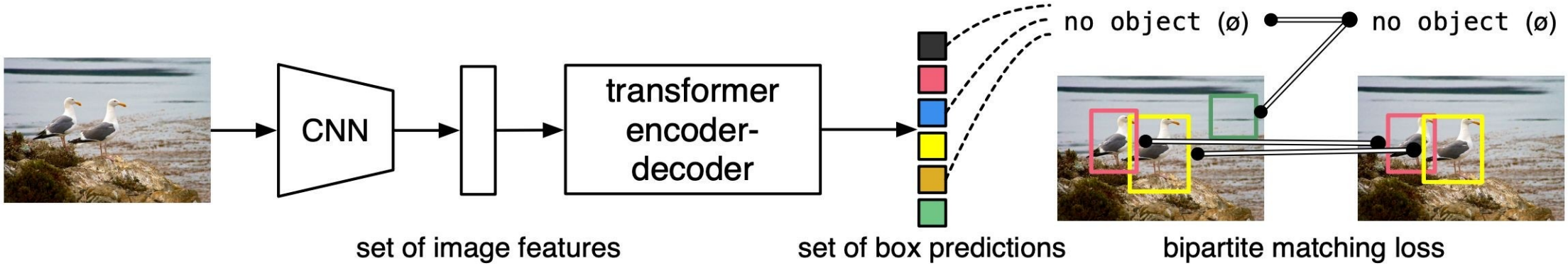
Swin Transformer



Liu et al., Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, 2021

DETR (DEtection TRansformer)

- views object detection as a direct set prediction problem



Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

DETR: Bipartite Matching

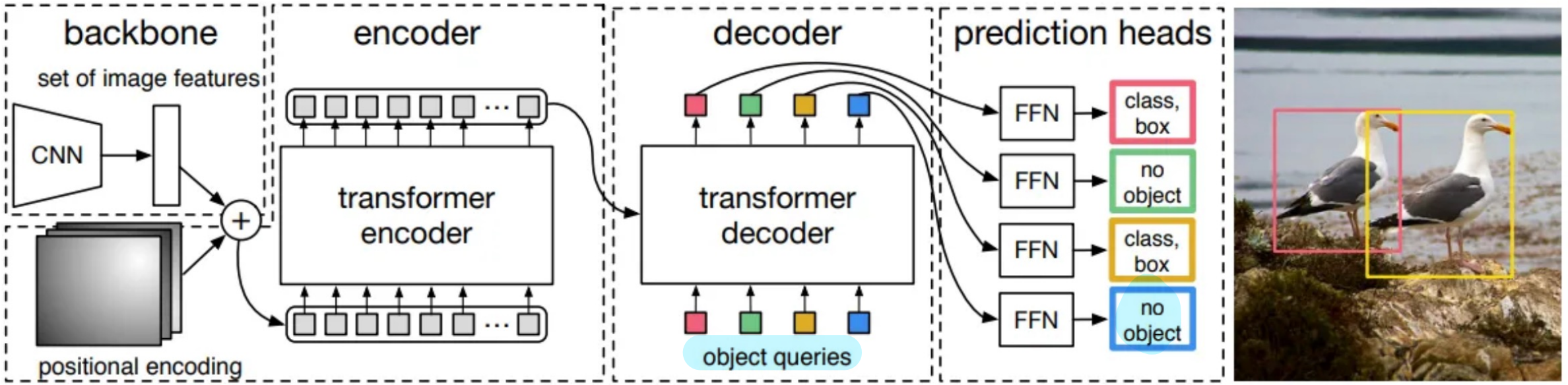
- Set prediction by the Hungarian algorithm to find a bipartite matching between ground-truth and prediction
  - Permutation-invariant and guarantees that each target has a unique match
  - Assuming # of predicted boxes, N, is larger than # of objects in the image, we consider y also as a set of size N padded with ∅ (no object)
  - Search for a permutation of N elements σ with the lowest cost is found

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}),$$

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

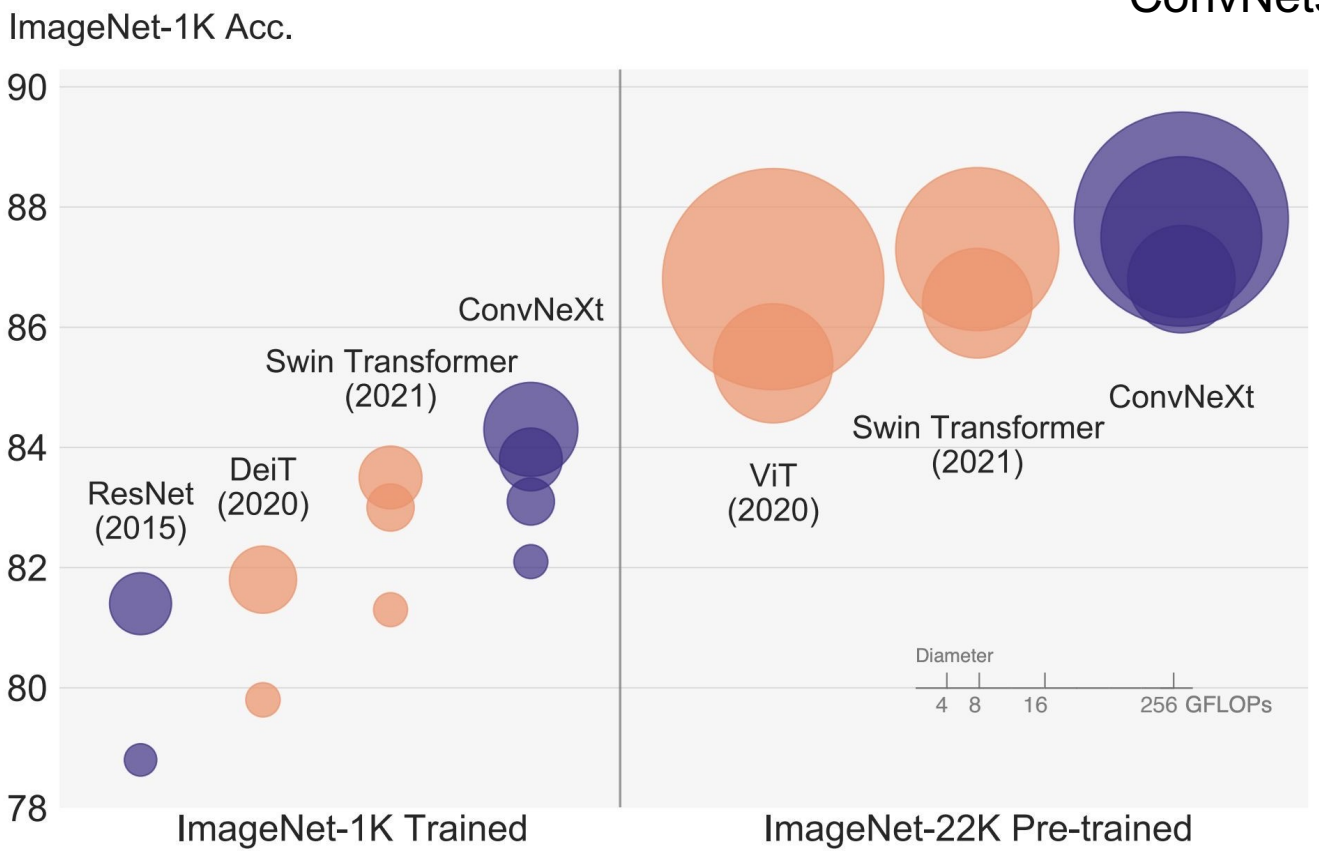
DETR (DEtection TRansformer)



Carion et al, “End-to-End Object Detection with Transformers”, ECCV 2020

Vision Transformers

ConvNets strike back!

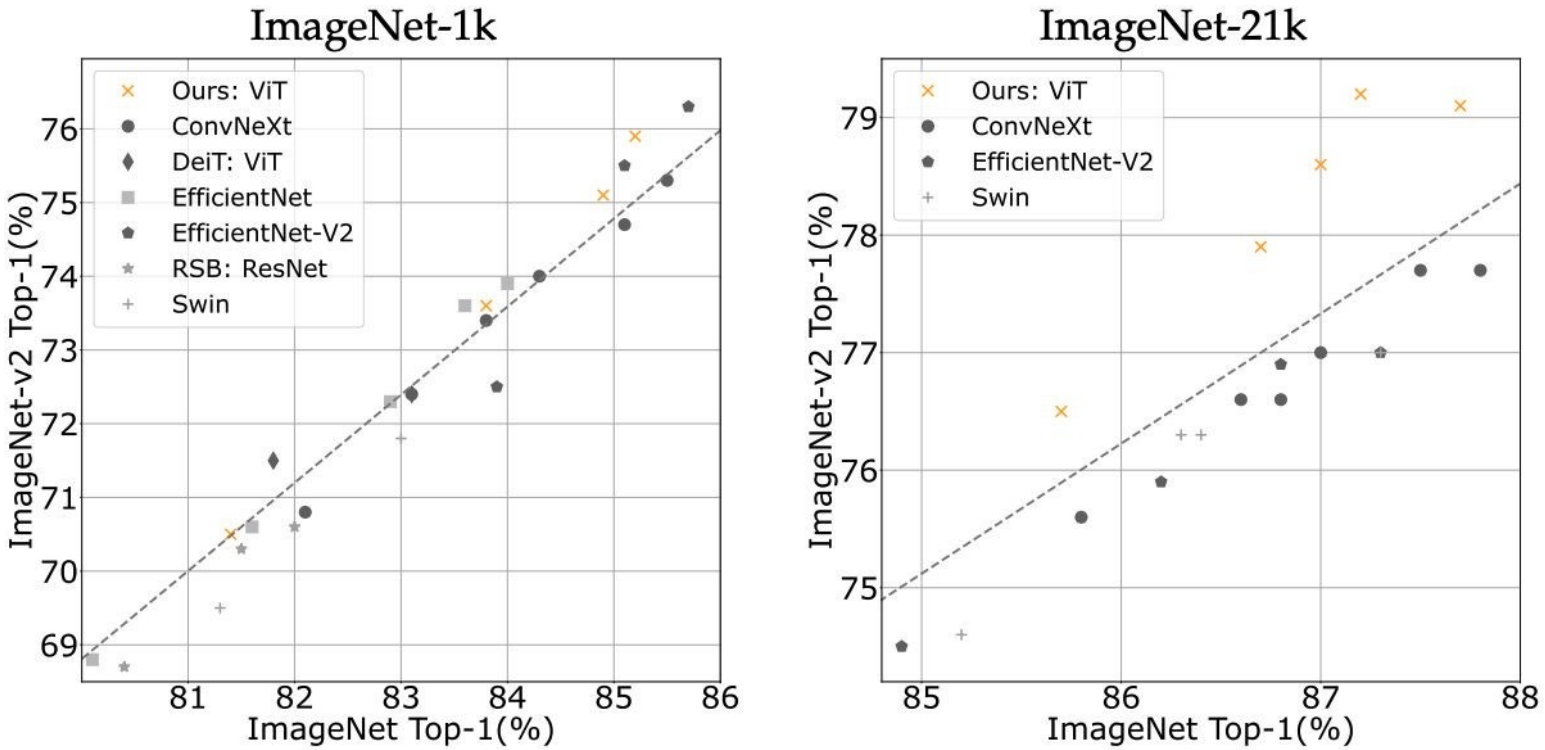


Liu et al., A ConvNet for the 2020s. CVPR 2022

Vision Transformers

DeiT III: Revenge of the ViT

Hugo Touvron*,† Matthieu Cord† Hervé Jégou*



Touvron et al., DeiT III: Revenge of the ViT, 2022

Summary

- Tokenization (BPE)
- Transforms in NLP
  - Encoder: BERT
  - Encode-decoder: T5
  - Decoder: GPTs
- Transformers in Vision
  - ViT, Swin, DETR