IMPERIAL

Pre-training Strategies (1)

08/12/2025

Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

Idris Abdulmumin
Postdoctoral Research Fellow,
DSFSI, University of Pretoria
https://abumafrim.github.io/

References

Chapter 8 : Transformer: https://web.stanford.edu/~jurafsky/slp3/

The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)(https://jalammar.github.io/illustrated-bert/)

The Illustrated Transformer https://jalammar.github.io/illustrated-transformer/

Reference Transformer Book: ***Natural Language Processing with Transformers***

Transformer Course: https://huggingface.co/learn/nlp-course/en/chapter1/1

LLMs are built out of transformers

Transformer: a fancier feedforward network, but based on attention

---

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

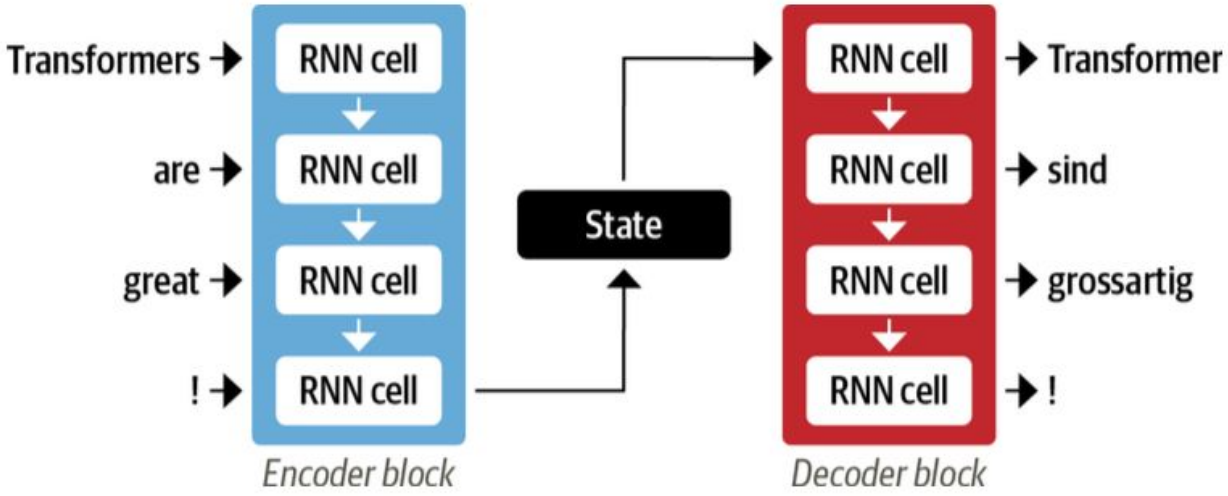
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

A very approximate timeline

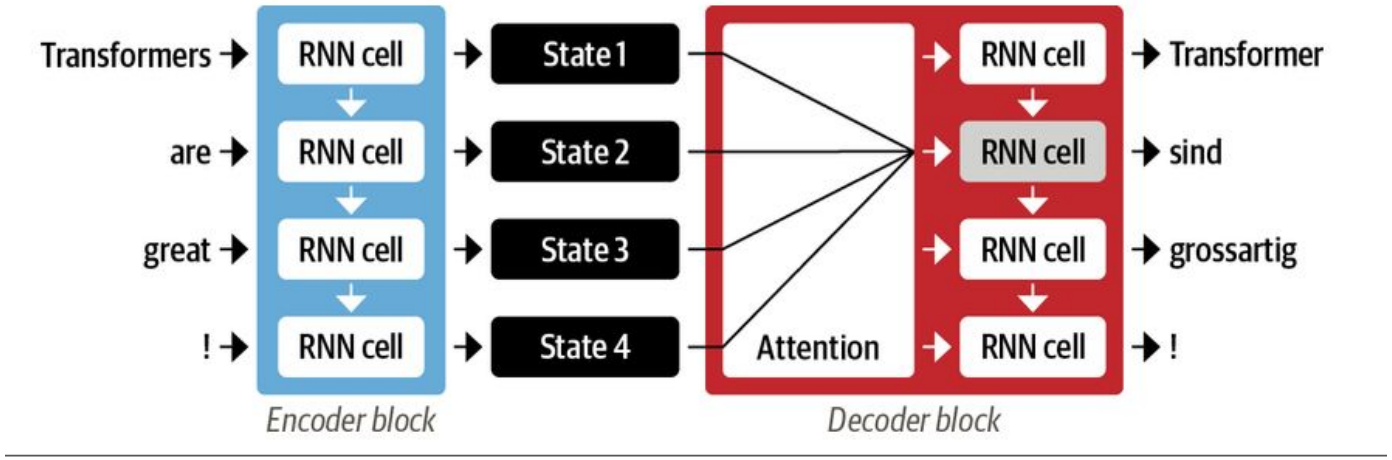
1990 Static Word Embeddings
2003 Neural Language Model
2008 Multi-Task Learning
2015 Attention
2017 Transformer
2018 Contextual Word Embeddings and Pretraining
2019 Prompting

An encoder-decoder architecture



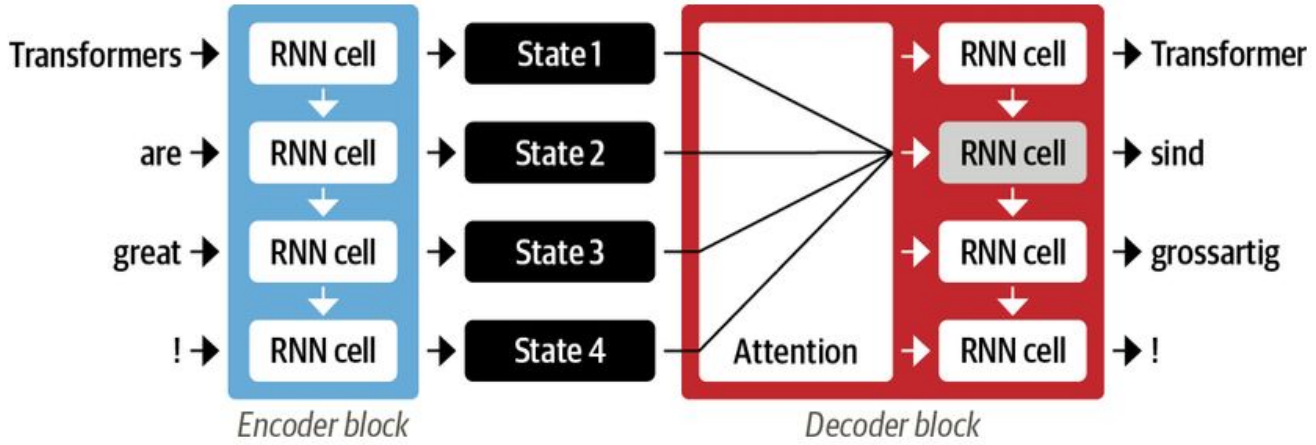
Although elegant in its simplicity, one weakness of this architecture is that the final hidden state of the encoder creates an information bottleneck: it has to represent the meaning of the whole input sequence because this is all the decoder has access to when generating the output.

An encoder-decoder architecture



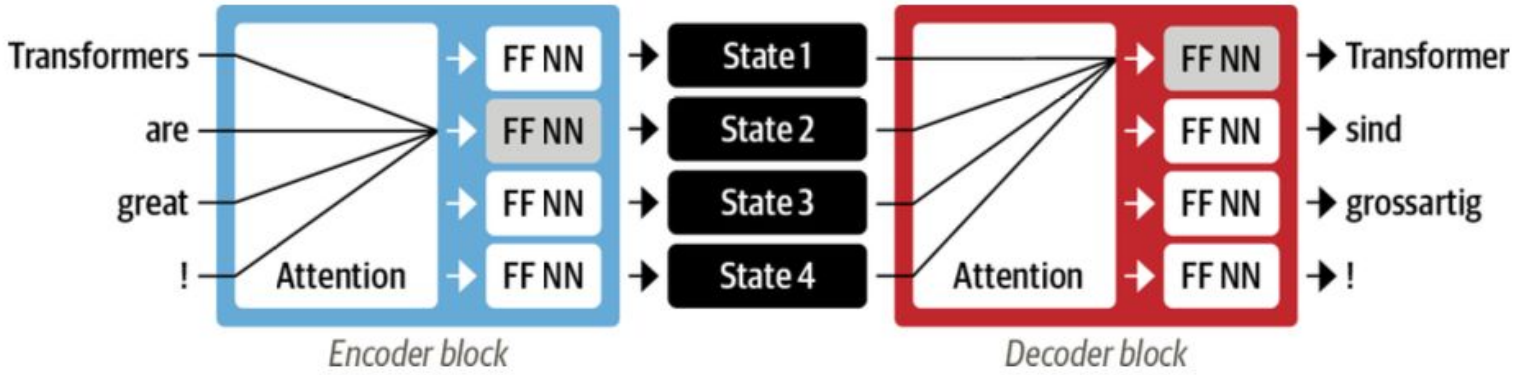
An encoder-decoder architecture with an attention mechanism for a pair of RNN

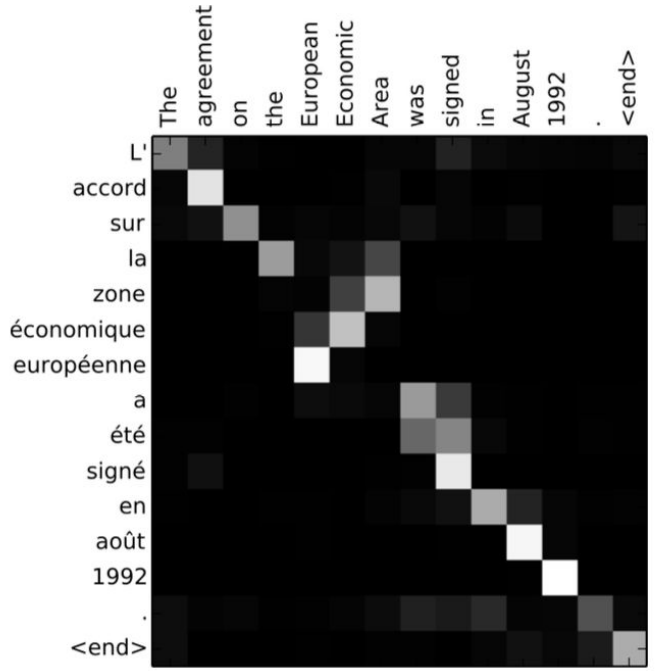
An encoder-decoder architecture



An encoder-decoder architecture with an attention mechanism for a pair of RNN

An encoder-decoder architecture





Encoder-decoder architecture of the original Transformer

Missing Piece: Transfer Learning

In the original Transformer paper, the translation model was trained from **scratch on a large corpus of sentence pairs in various languages.**

However, in many practical applications of NLP we do not have access to large amounts of labeled text data to train our models on. A final piece was missing to get the transformer revolution started: transfer learning.

A final piece was missing to get the transformer revolution started: **transfer learning.**

Transfer Learning in NLP

Imagine you want to teach someone how to cook jollof rice

Transfer Learning in NLP

Person A — Has Never Cooked Before

They do not know:

- how to boil water
- how to cut onions
- what spices are
- how to measure ingredients

Before they can cook jollof rice, you must teach them:

1. Basic cooking skills
2. Kitchen safety
3. Cutting, boiling, frying
4. Timing and temperature control

This takes **a long time**.

Person B — Already Knows Basic Cooking

This person already understands:

- how to chop vegetables
- how to fry and boil
- how flavors work
- how to follow a recipe

So to teach them jollof rice, you only need to:

- show them the recipe
- adjust spices
- explain the local style

They learn very quickly because they already have foundational cooking knowledge.

Transfer Learning in NLP

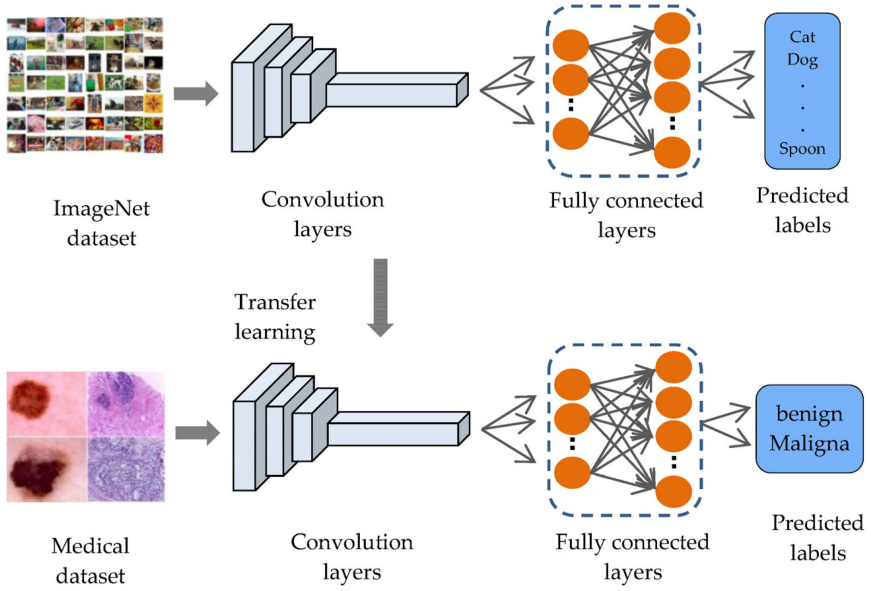
Teaching a model from scratch *is like teaching someone who has never cooked before. Using transfer learning is like teaching someone who already knows how to cook.*

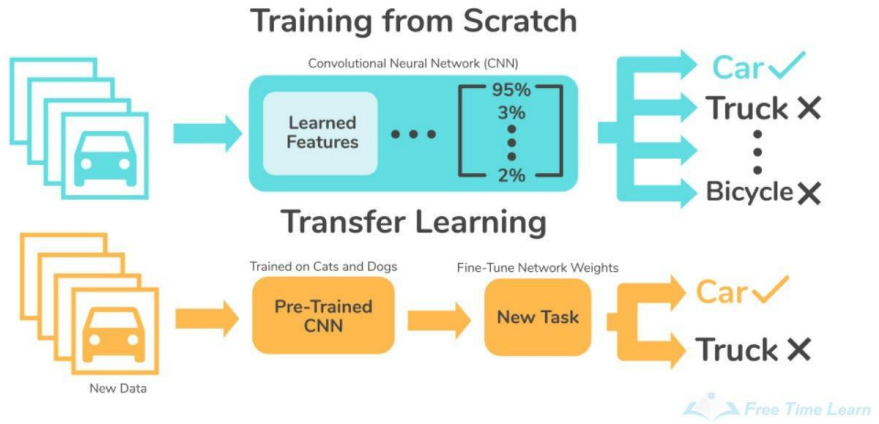
"Transfer learning *means starting with a model that already knows the basics —like hiring someone who already knows how to cook, drive, or read and then teaching them the specific job.*

Transfer Learning in Computer Vision

- It is common practice in **computer vision** to use transfer learning to train a convolutional neural network like ResNet on one task and then adapt it to or fine-tune it on a new task.

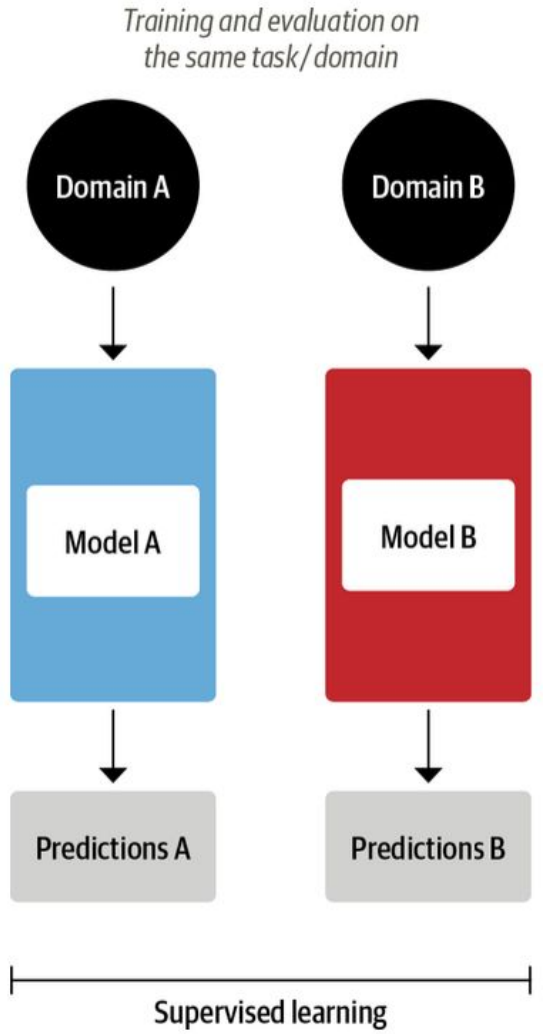
- This allows the network to make use of the knowledge learned from the original task.





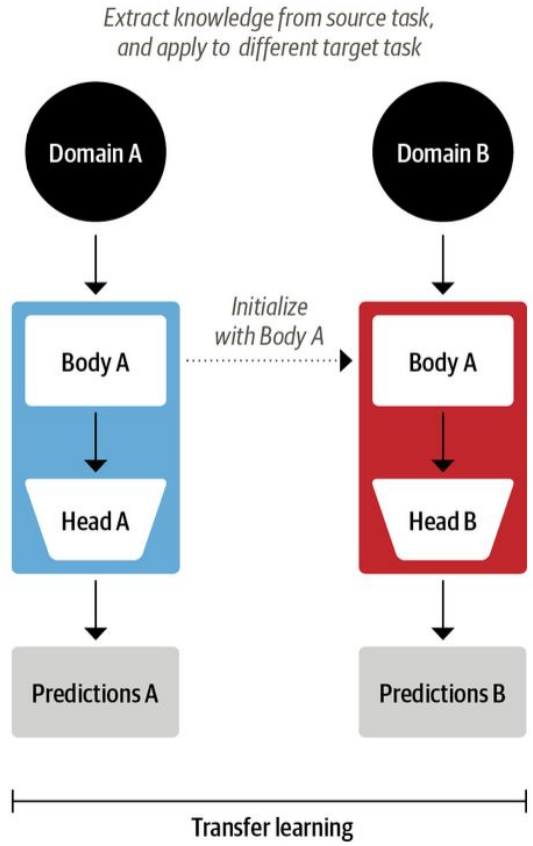
Transfer Learning in Computer Vision

The models are first trained on large-scale datasets such as *ImageNet*, which contain millions of images. This process is called ***pretraining*** and its main purpose is to teach the models the basic features of images, such as edges or colors.



Transfer Learning in Computer Vision

Pretrained models can then be fine-tuned on a *downstream* task such as classifying flower species with a relatively **small number of labeled examples** (usually a few hundred per class).



Fine-tuned models typically achieve a higher accuracy than supervised models trained from scratch on the same amount of labeled data.

Transfer Learning in NLP

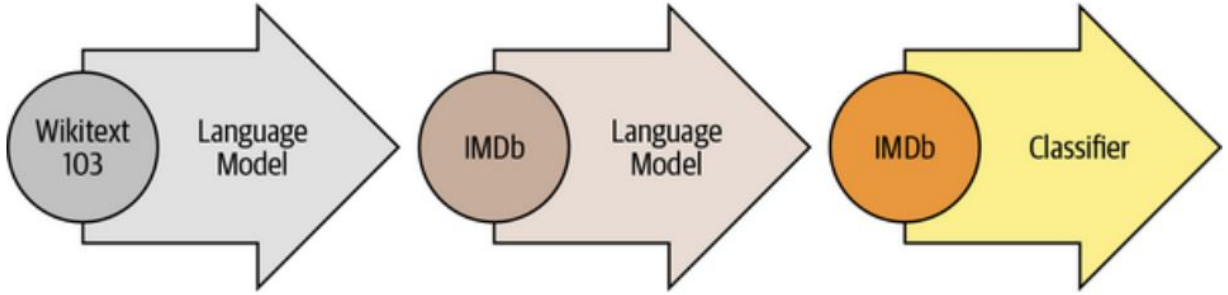
Although transfer learning became the standard approach in computer vision, for many years it was not clear what the analogous pretraining process was for NLP.

As aresult, NLP applications typically required large amounts of labeled data to achieve high performance. And even then, that performance did not compare to what was achieved in the vision domain.

ULMFiT

In In 2017 and 2018, people were still training model from scratch in NLP, not using transfer learning

ULMFiT provided the missing piece to make transformers take off. In 2018,



Universal Language Model Fine-tuning for Text Classification

Jeremy Howard*
fast.ai
University of San Francisco
j@fast.ai

Sebastian Ruder*
Insight Centre, NUI Galway
Aylien Ltd., Dublin
sebastian@ruder.io

Abstract

Inductive transfer learning has greatly impacted computer vision, but existing approaches in NLP still require task-specific modifications and training from scratch. We propose Universal Language Model Fine-tuning (ULMFiT), an effective transfer learning method that can be applied to any task in NLP, and introduce techniques that are key for fine-tuning a language model. Our method significantly outperforms the state-of-the-art on six text classification tasks, reducing the error by 18-24% on the majority of datasets. Furthermore, with only 100 labeled examples, it matches the performance of training from scratch on 100× more data. We open-source our pretrained models and code[1].

While Deep Learning models have achieved state-of-the-art on many NLP tasks, these models are trained from scratch, requiring large datasets, and days to converge. Research in NLP focused mostly on *transductive* transfer (Blitzer et al., 2007). For *inductive* transfer, fine-tuning pretrained word embeddings (Mikolov et al., 2013), a simple transfer technique that only targets a model's first layer, has had a large impact in practice and is used in most state-of-the-art models. Recent approaches that concatenate embeddings derived from other tasks with the input at different layers (Peters et al., 2017; McCann et al., 2017; Peters et al., 2018) still train the main task model from scratch and treat pretrained embeddings as fixed parameters, limiting their usefulness.

In light of the benefits of pretraining (Erhan et al., 2010), we should be able to do better than *randomly initializing* the remaining parameters of our models. However, inductive transfer via fine-

08/12/2025

Transfer Learning in NLP

By introducing a viable framework for pretraining and transfer learning in NLP, **ULMFiT** provided the missing piece to make transformers take off.

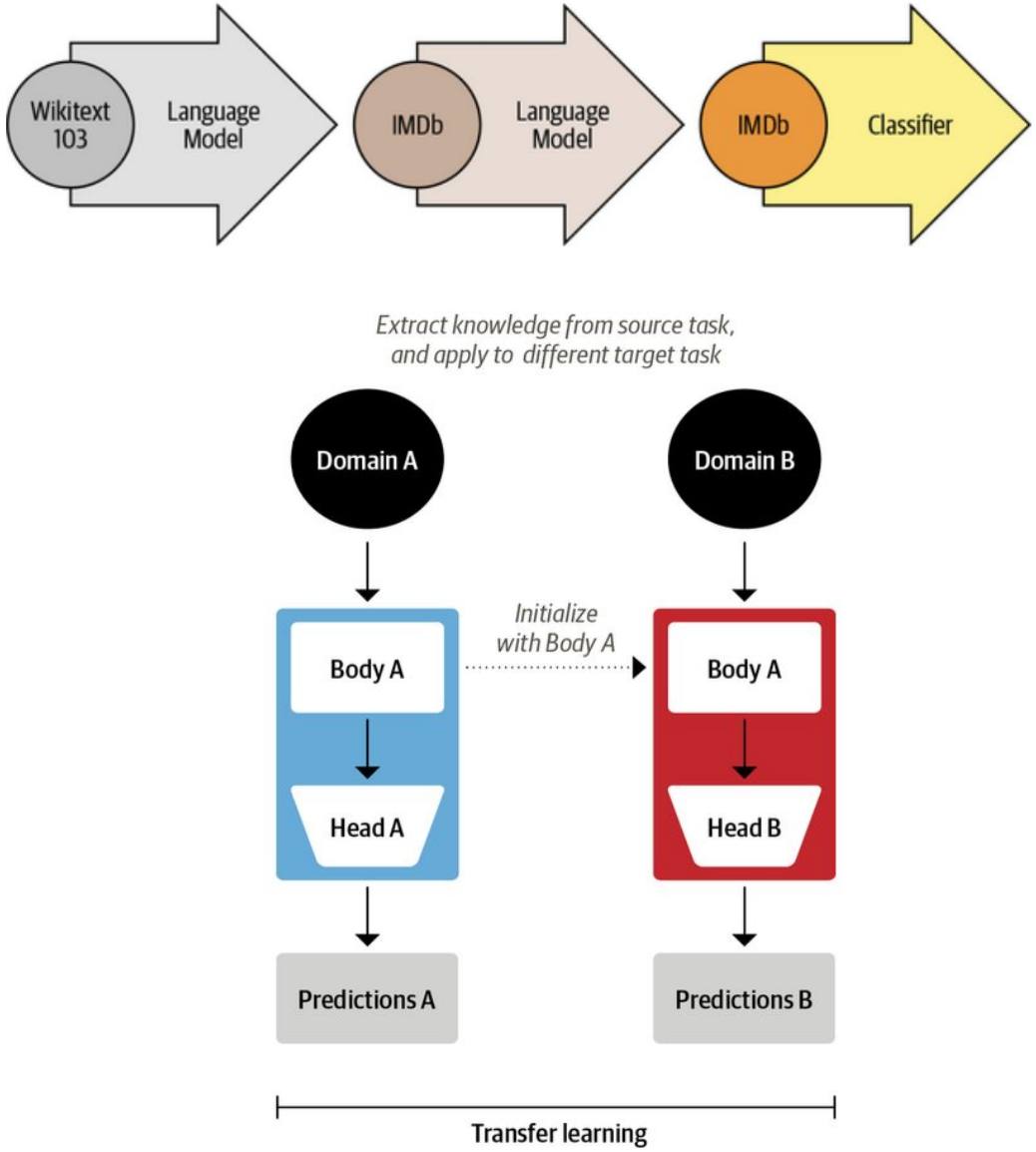
In 2018, *two transformers were released that combined self-attention with transfer learning:* ***GPT and BERT***

GPT and **BERT** set a new state of the art across a variety of NLP benchmarks and ushered in the age of transformers.

ULMFiT

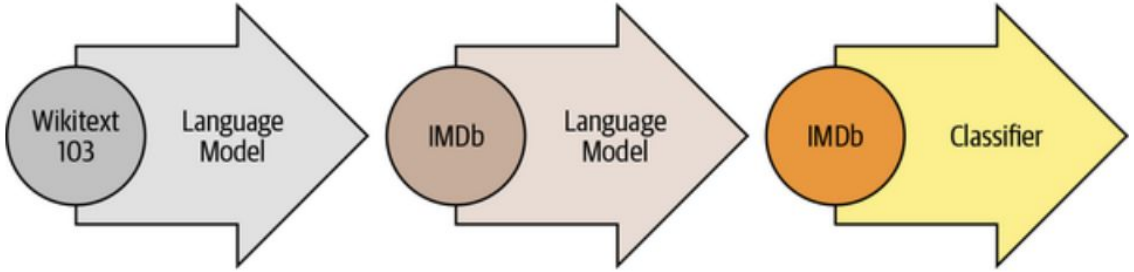
Pretraining : predict the next word based on the previous words. This task is referred to as language modeling. *no labeled data is required*, and one can make use of abundantly available text from sources such as Wikipedia.

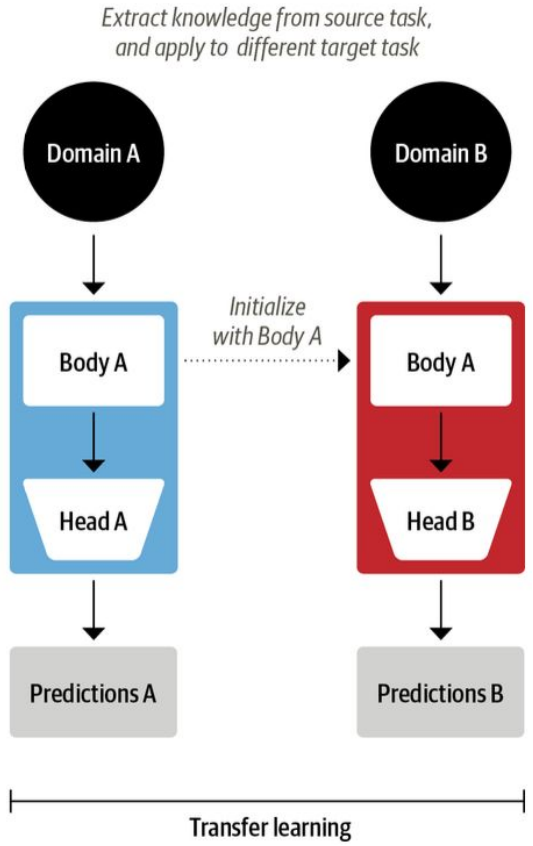
Domain adaptation Once the language model is pretrained on a large-scale corpus, the next step is to adapt it to the in-domain corpus (e.g., IMDB moview review corpus.). This stage still uses ***language modeling***, but now the model has to predict the next word in the target corpus.



ULMFiT

Fine-tuning In this step, the language model is fine-tuned with a ***classification layer for the target task***



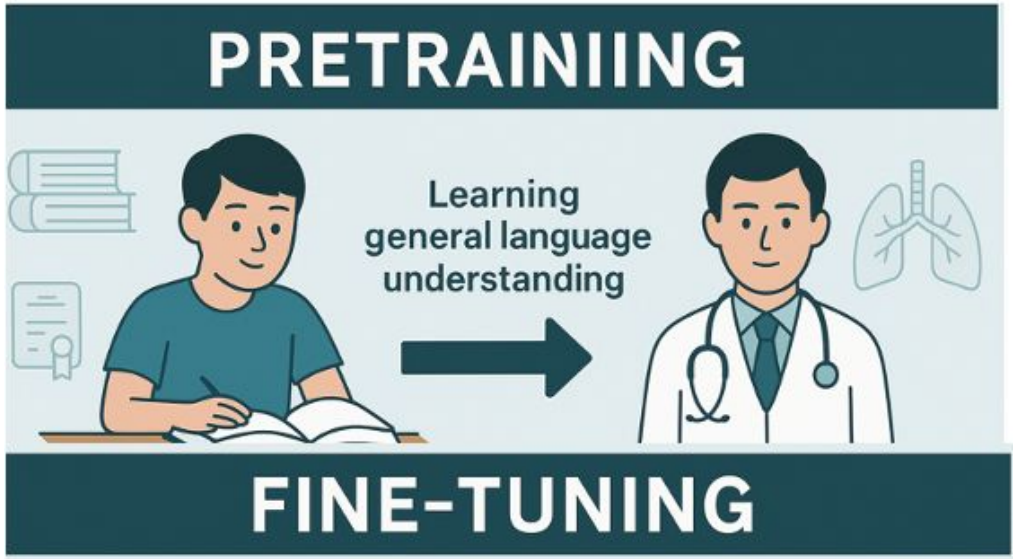


Analogy: Learning to Read and Becoming a Doctor

- **Pretraining** is like **going to school**: you learn how to read, write, and understand general knowledge.

- **Fine-tuning** is like **medical school**: you specialize using your general knowledge to become a doctor who understands anatomy, diseases, and how to treat patients.

Just like you don’t need to relearn how to read in med school, you don’t need to train a language model from scratch every time, you just fine-tune it for the task at hand.

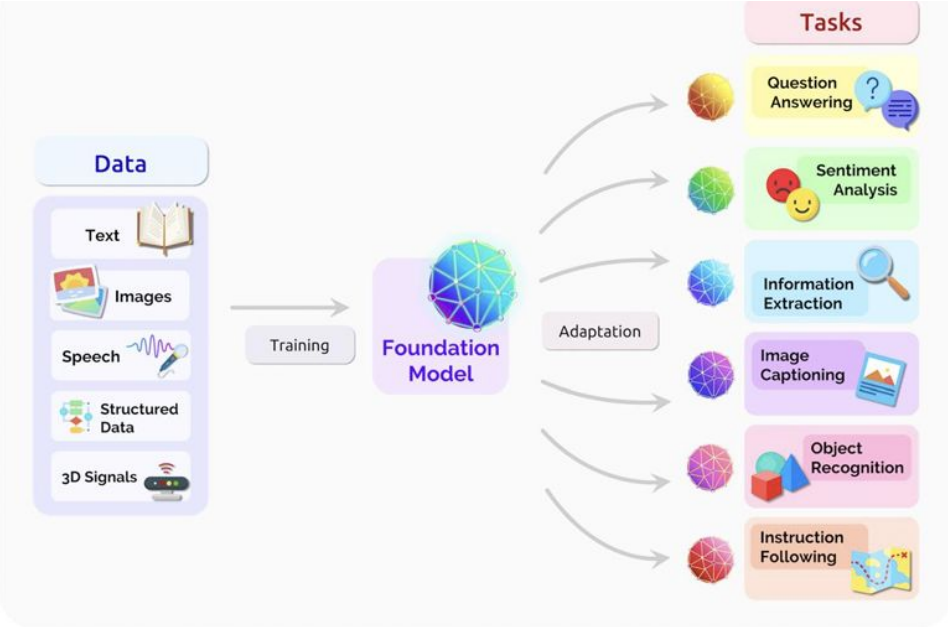


Pretraining vs Fine-tuning

Pretraining is the process where a language model (like BERT or GPT) is trained on a large amount of general text data (e.g., books, Wikipedia) to learn the basics of language, like grammar, vocabulary, and how words relate to each other.

Fine-tuning is when we take that pretrained model and train it a bit more on a specific task, such as sentiment analysis, translation, or question answering, using a smaller, task-specific dataset.

Pretraining – unsupervised learning on the internet



Key ideas in pretraining

- Make sure your model can process large-scale, diverse datasets
- Don't use labeled data (otherwise you can't scale!)
- Compute-aware scaling

Motivating word meaning and context

Recall the adage we mentioned at the beginning of the course:

> *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)

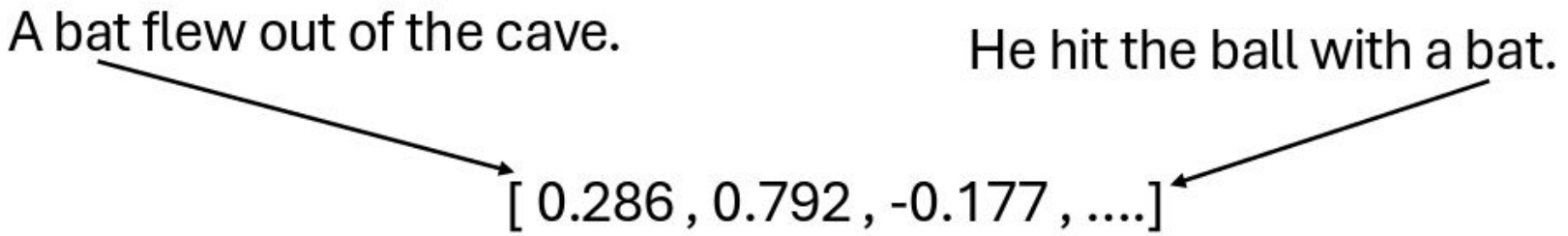
This quote is a summary of **distributional semantics**, and motivated **word2vec**. But:

> "… the complete meaning of a word is always contextual,
> and no study of meaning apart from a complete context
> can be taken seriously." (J. R. Firth 1935)

Consider *I* ***record*** *the* ***record***: the two instances of ***record*** mean different things.

Background: Contextual Representations

- Word embeddings serve as the foundation for deep learning models in natural language processing.
- **Problem :** Word embeddings (word2vec, GloVe) are used without considering the context in which the words appear.



- **Solution :** Train contextual representations on text corpus

The representation of the word should depend on the context in which it appears.

Contextual Representations

Think of a word like **“cute.”**

The meaning of *“cute”* changes depending on the sentence:

- “A **cute** puppy” → cute means adorable
- “A **cute** trick” → clever
- “That was a **cute** comment” → sarcastic

Traditional word embeddings (like Word2Vec, GloVe) give the same meaning of “cute” everywhere.

But human beings don’t do that — we understand words based on context.

Contextual Representations

Non-transformer-based embedding approach called **ELMO**

Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

We introduce a new type of *deep contextualized* word representation that models both (1) complex characteristics of word use (e.g., syntax and semantics), and (2) how these uses vary across linguistic contexts (i.e., to model polysemy). Our word vectors are learned functions of the internal states of a deep bidirectional language model (biLM), which is pre-trained on a large text corpus. We show that these representations can be easily added to existing models and significantly improve the state of the art across six challenging NLP problems, including question answering, textual entailment and sentiment analysis. We also present an analysis showing that exposing the deep internals of the pre-trained network is crucial, allowing downstream models to mix different types of semi-supervision signals.

guage model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors (Peters et al., 2017; McCann et al., 2017), ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer.

Combining the internal states in this manner allows for very rich word representations. Using intrinsic evaluations, we show that the higher-level LSTM states capture context-dependent aspects of word meaning (e.g., they can be used without modification to perform well on supervised word sense disambiguation tasks) while lower-

Background: Contextual Representations

Imagine describing the word **“goal.”**

If you only hear the word alone, you don’t know if it means:

- Football goal
- Life target
- A software objective

But if you hear the **whole sentence**, you immediately know the meaning.

ELMo does exactly that — it listens to the **whole sentence** before deciding what the word means.

ELMo (Embedding from Language Models)

ELMo gives each word a different meaning depending on the sentence it appears in.

It reads the whole sentence *forward* and *backward*:

- **Forward**: A → cute → puppy
- **Backward**: puppy → cute → A

So the model understands what comes before and after the word.

ELMo (Embedding from Language Models)

Step 1: Read the sentence from left to right (forward LM)

This gives one type of hidden representation for each word.

Step 2: Read the sentence from right to left (backward LM)

This gives another representation.

Step 3: Combine both directions

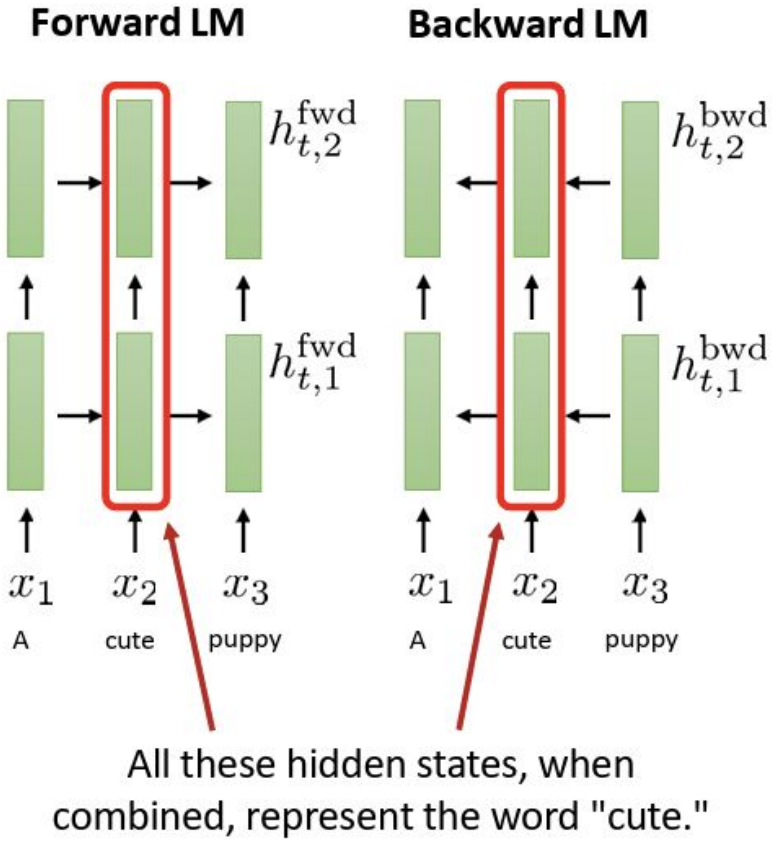
Now each word has a **rich meaning** built from all surrounding words.

For the word **“cute”**, ELMo combines:

- Hidden states from the forward model
- Hidden states from the backward model
- Multiple layers of the neural network

ELMo (Embedding from Language Models)

One of the first contextual embedding



Replace static embeddings (lexicon lookup) with context-dependent embeddings (produced by a deep neural language model)

- Each token's representation is a function of the entire input sentence, computed by a deep (multi-layer) bidirectional language model
- Return for each token a (task-dependent) linear combination of its representation across layers.
- Different layers capture different information

ELMo’s Token Representation

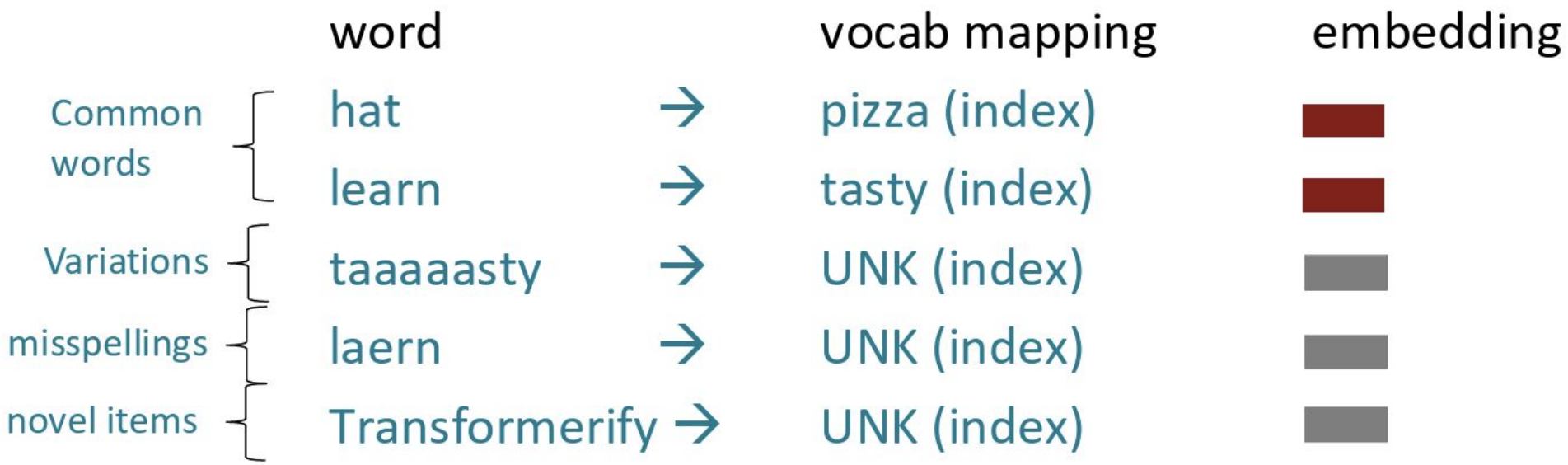
- The input token representations are purely character-based
- Advantage over using fixed embeddings: no **UNK tokens**, any word can be represented

Word structure and subword models

Let's take a look at the assumptions we've made about a language's vocabulary.

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.



Word structure and subword models

Finite vocabulary assumptions make even *less* sense in many languages.

- Many languages exhibit complex **morphology**, or word structure.
  - The effect is more word types, each occurring fewer times.

Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++)

Here's a small fraction of the conjugations for *ambia* – to tell.

Conjugation of *-ambia* [less ▲]

Non-finite forms		
Form	Positive	Negative
Infinitive	kuambia	kutoambia

Simple finite forms		
Positive form	Singular	Plural
Imperative	ambia	ambieni
Habitual	huambia	

Complex finite forms

Polarity	Persons				Persons / Classes		Classes												
	1st		2nd		3rd / M-wa		M-mi		Ma		Ki-vi		N		U	Ku	Pa	Mu	
	Sg.	Pl.	Sg.	Pl.	Sg. / 1	Pl. / 2	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18	
Past																			[less ▲]
Positive	niliambia naliambia	tuliambia twaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia	
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia	
Present																			[less ▲]
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia	
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii	
Future																			[less ▲]
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia	
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia	
Subjunctive																			[less ▲]
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie	
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie	
Present Conditional																			[less ▲]
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia	
Negative	nisingeambia singeambia	tusingeambia hatungeambia	usingeambia hungeambia	msingeambia hamngeambia	asingeambia hangeambia	wasingeambia hawangeambia	usingeambia haungeambia	isingeambia haingeambia	lisingeambia halingeambia	yasingeambia hayangeambia	kisingeambia hakingeambia	visingeambia havingeambia	isingeambia haingeambia	zisingeambia hazingeambia	usingeambia haungeambia	kusingeambia hakungeambia	pasingeambia hapangeambia	musingeambia hamungeambia	
Past Conditional																			[less ▲]
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia	
Negative	nisingaliambia singaliambia	tusingaliambia hatungaliambia	usingaliambia hungaliambia	msingaliambia hamngaliambia	asingaliambia hangaliambia	wasingaliambia hawangaliambia	usingaliambia haungaliambia	isingaliambia haingaliambia	lisingaliambia halingaliambia	yasingaliambia hayangaliambia	kisingaliambia hakingaliambia	visingaliambia havingaliambia	isingaliambia haingaliambia	zisingaliambia hazingaliambia	usingaliambia haungaliambia	kusingaliambia hakungaliambia	pasingaliambia hapangaliambia	musingaliambia hamungaliambia	
Conditional Contrary to Fact																			[less ▲]
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia	
Gnomic																			[less ▲]
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia	
Perfect																			[less ▲]

NLP at AIMS Cameroon 2025

Subword Tokenization

Word-level tokenization suffers from:

- Out-of-Vocabulary (OOV) issues
- Large vocabulary size
- Poor handling of morphological variation

Character-level models lack semantic structure

Need a compromise: subword tokenization

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 subword vocabulary.

1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
2. Using a corpus of text, 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.

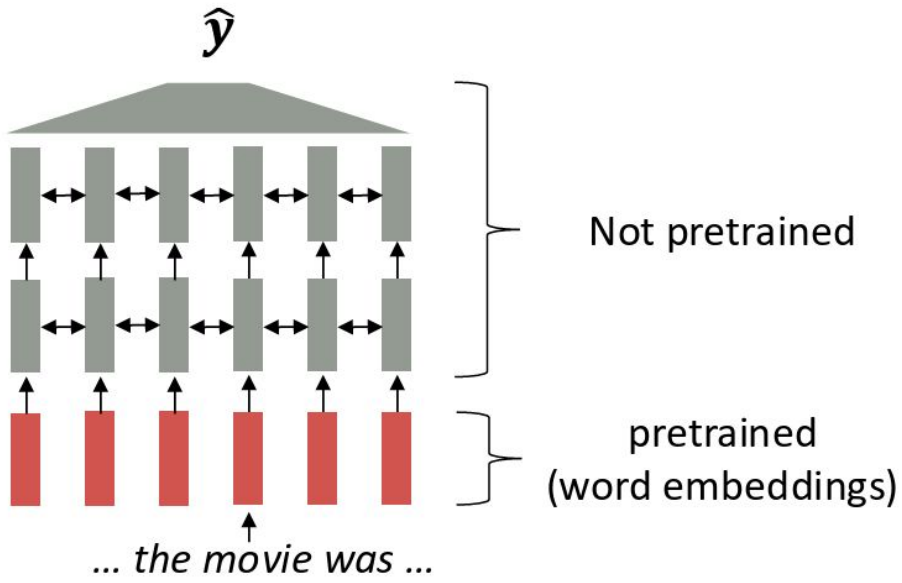
Where we were: pretrained word embeddings/vectors

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 we have for our **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!



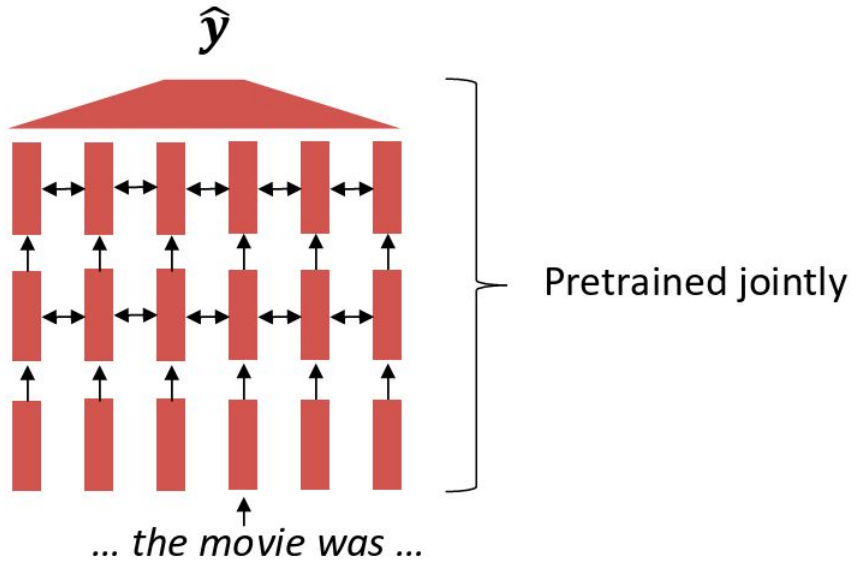
[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we're going: pretraining whole models Pre-trained Word Vectors -> Pre-trained Models

In modern NLP:

- All (or almost all) parameters in NLP 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



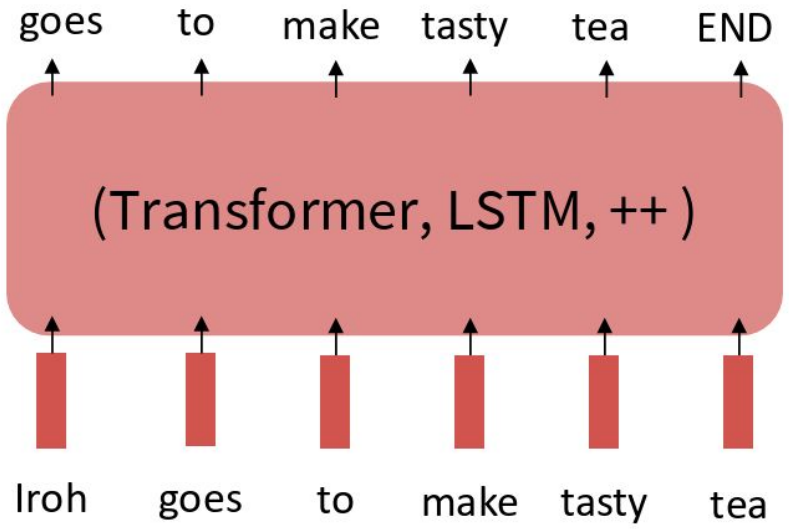
[This model has learned how to represent entire sentences through pretraining]

The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

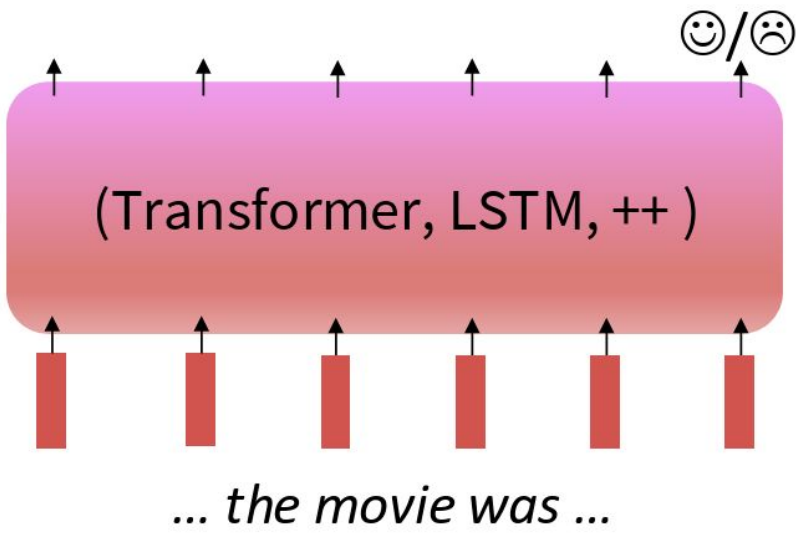
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!



Why is it called pre-training?

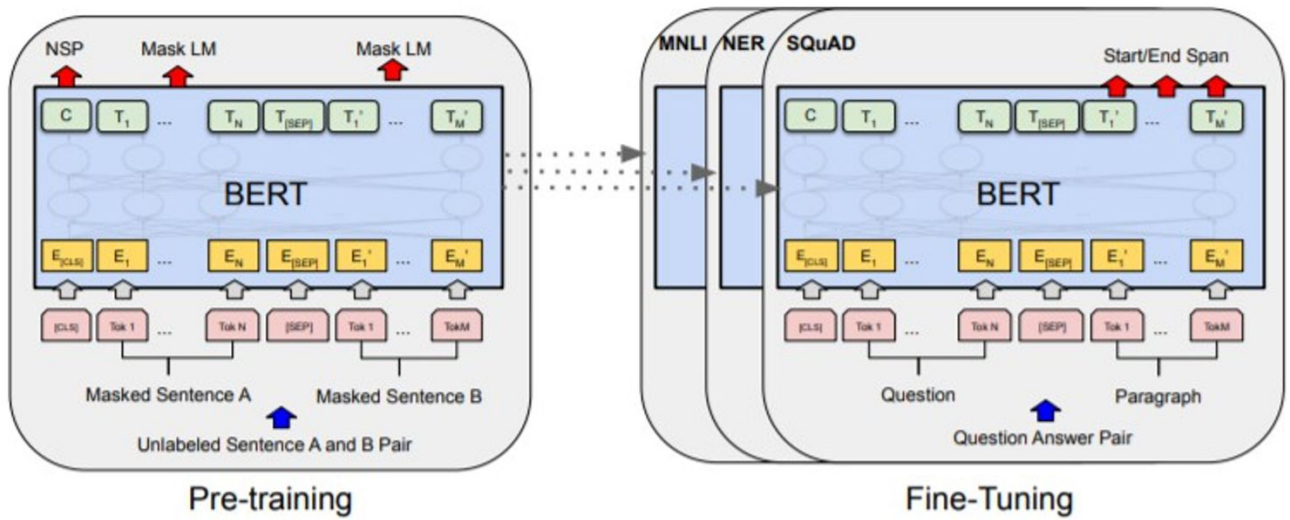


Image Credit: BERT Paper

“Pre-”training happens before training (fine-tuning)!

Why Is It Called Pre-training for LLMs?

The **term pre-training** is used because the large initial training phase is not the final training objective of the model.

Instead, it is the first stage in a **two-stage learning pipeline** in modern NLP architectures.

Pre-training = Learning General Knowledge: syntax, semantics, world knowledge, general reasoning patterns, broad linguistic representations

Fine-tuning = Task-Specific Training: classification, NER, QA, etc

Why Not Just Call It “Training”?

Because in modern NLP:

- **Training** refers to task-level optimization.
- **Pre-training** refers to corpus-wide representation learning.

Calling both “training” would make it unclear which stage is being discussed

Large language models

Computational agents that can interact conversationally with people using natural language

LLMS have revolutionized the field of NLP and AI

Large language models

- Remember the simple n-gram language model
  - Assigns probabilities to sequences of words
  - Generate text by sampling possible next words
  - Is trained on *counts computed from lots of text*

- Large language models are similar and different:
  - Assigns probabilities to sequences of words
  - Generate text by sampling possible next words
  - ***Are trained by learning to guess the next word***

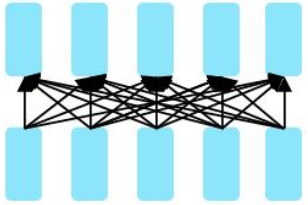
Fundamental intuition of large language models

Text contains enormous amounts of knowledge

Pretraining on lots of text with all that knowledge is what gives language models their ability to do so much

Pretraining LLM: three types of architectures

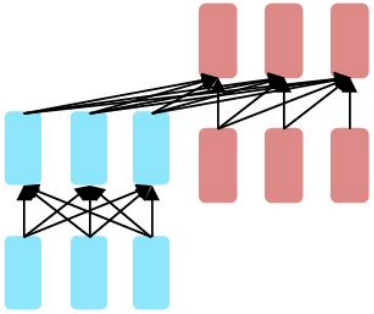
The neural architecture influences the type of pretraining, and natural use cases.



Encoders

e.g., BERT

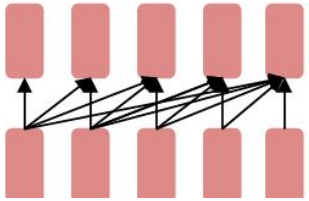
- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

e.g., BART, T5

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

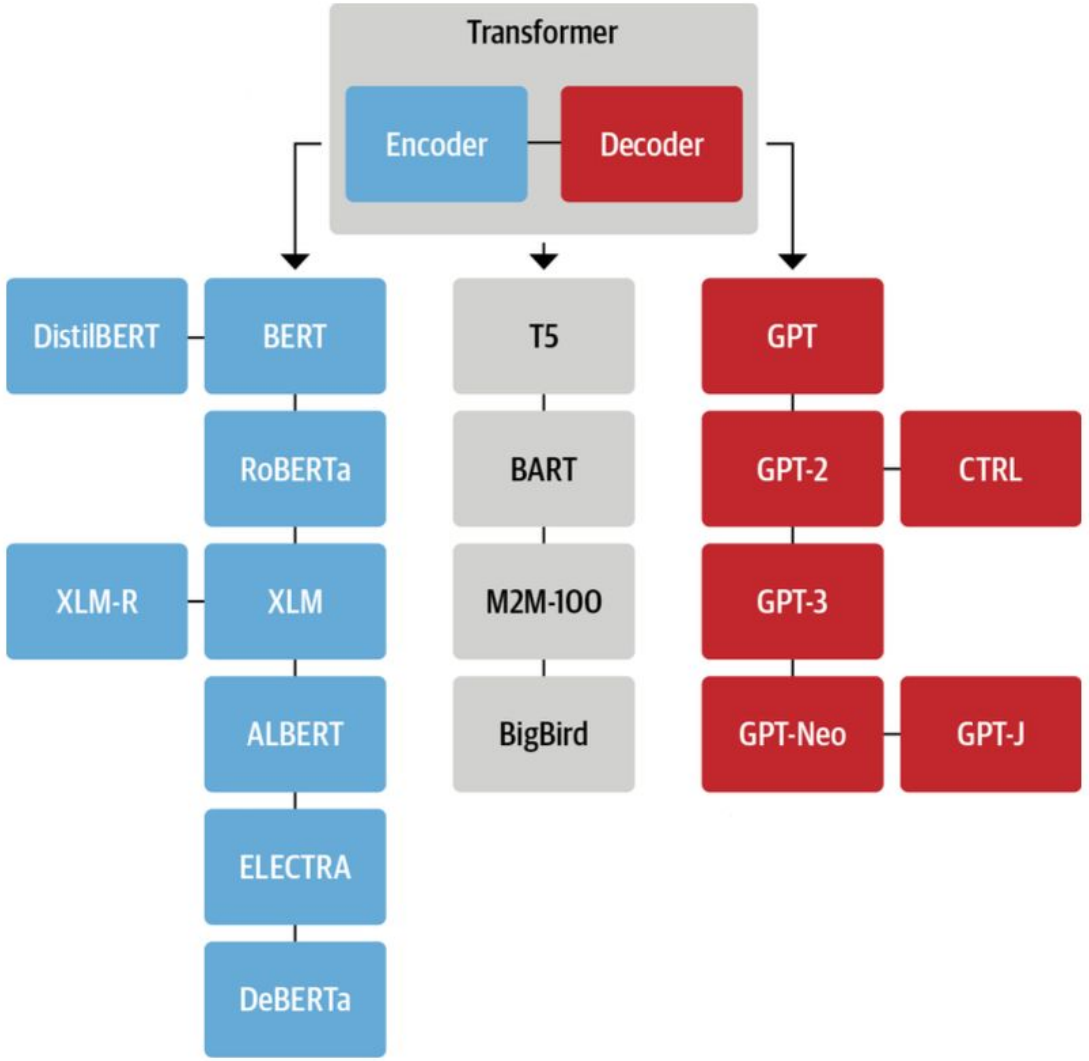


Decoders

e.g., GPT, Llama

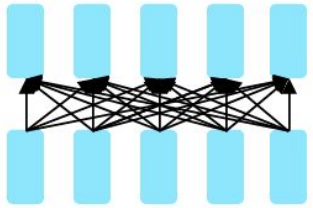
- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Transformers



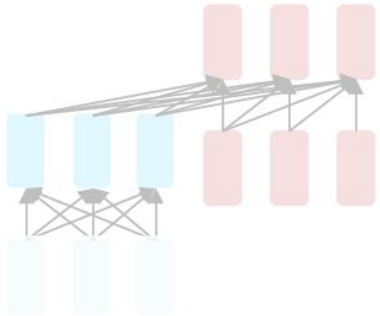
Pretraining for three types of architectures

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



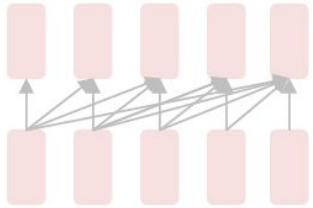
Encoders

- 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

The first encoder-only model based on the Transformer architecture was BERT

BERT: Bidirectional Encoder Representations from Transformers

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

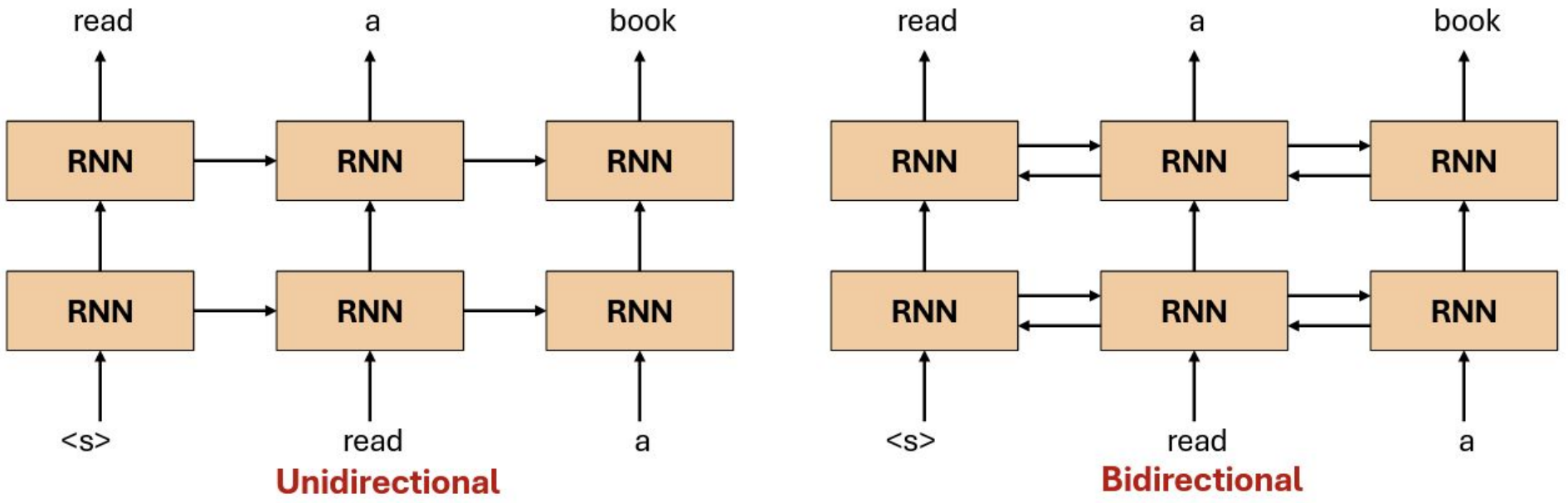
Motivation

- **Problem with previous methods:**
  - Language models only use left context or right context.
  - But language understanding is **bidirectional**.
- **Possible Issue:**
  - Directionality is needed to generate a well-formed probability distribution.
  - Words can see themselves in a bidirectional model.

Bidirectional Context

- Bidirectional context, unlike unidirectional context, takes into account both the left and right contexts.

Unidirectional vs. Bidirectional Models



Encoders Cannot Be Used for Language Modeling

A standard language model (**LM**) predicts the next word using only previous words:

$$p(w_t \mid w_1, w_2, \ldots, w_{t-1})$$

This means:

- the model must **not look ahead**,
- future words must remain **invisible** during prediction.

This is a ***causal*** / ***unidirectional*** constraint.

What encoders do

Encoders (like in BERT) process the entire sentence at once using *self-attention*.

Self-attention in an encoder layer is **bidirectional**:

- word *i* attends to all other words — both before and after it.
- there is **no restriction** preventing a token from seeing “future” positions.

Thus the encoder has **access to the full context**:

$$h_t = f(w_1, \ldots, w_t, \ldots, w_T)$$

This violates the causal LM requirement.

What encoders do

Encoders (like in BERT) process the entire sentence at once using *self-attention*.

Self-attention in an encoder layer is **bidirectional**:

- word *i* attends to all other words — both before and after it.
- there is **no restriction** preventing a token from seeing “future” positions.

Thus the encoder has **access to the full context**:

$$h_t = f(w_1, \ldots, w_t, \ldots, w_T)$$

This violates the causal LM requirement.

A *causal* language model predicts the next word using only the words that come before it.

Why this makes language modeling impossible

If the model can see the future word (e.g., *“cake”*), it cannot meaningfully learn to predict it:

Example:

- Input: *“I love [MASK] cake”*
- An encoder sees *“cake”* on the right — so predicting “cake” is trivial.

For **LM,** this would leak the answer.
Therefore:

> Encoders leak future information, so they cannot be trained to predict the next word in a left-to-right manner.

Encoder-only models → use Masked Language Modeling (MLM)

Instead of predicting the next word, they predict masked words with bidirectional context.

This is why BERT is trained with **Masked LM**, not standard LM.

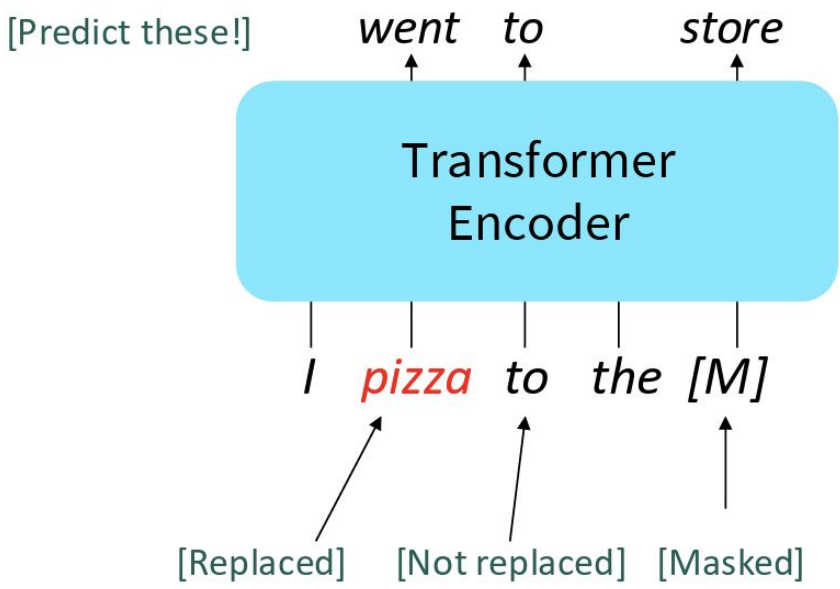
Encoder models cannot be used for language modeling because their bidirectional attention lets each token see future words, which violates the causal constraint required for predicting the next word.

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective and **released the weights of a pretrained Transformer**, a model they labeled 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!)



Masked Language Modelling

- Mask out k% of the input words, and then predict the masked words (Usually k = 15%). Example :

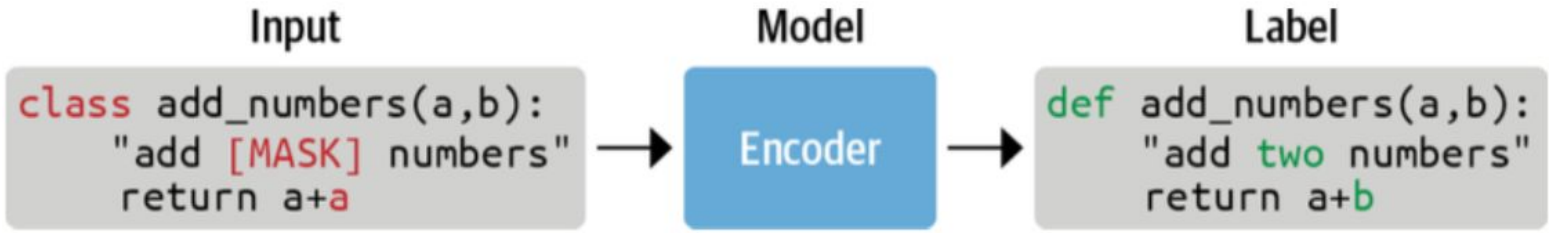
I like going to the [MASK] in the evening

↓

park

  - Too little masking: Too expensive to train
  - Too much masking: Not enough context
- The model needs to predict 15% of the words, but we don't replace with [MASK] 100% of the time. Instead:
  - 80% of the time, **replace with [MASK]**
    - Example : like going to the park → like going to the [MASK]
  - 10% of the time, **replace random word**
    - Example : like going to the park → like going to the store
  - 10% of the time, **keep same**
    - Example : like going to the park → like going to the park

Masked Language Modelling



Self-supervised training objective and is commonly called m***asked language modeling or the denoising objective***. Denoising is generally a good pretraining task to learn general representations for later downstream task.

BERT was trained using two objectives:

(1) Masked Language Modeling (MLM)

Predict masked words → teaches *token-level* understanding.

(2) Next Sentence Prediction (NSP)

Predict sentence relationships → teaches *sentence-level* understanding.

BERT was trained using two objectives:

(1) Masked Language Modeling (MLM)

Predict masked words, teaches *token-level* understanding.

(2) Next Sentence Prediction (NSP)

Predict sentence relationships, teaches *sentence-level* understanding.

These objectives make BERT powerful for tasks involving:

- sentence pairs
- paragraph comprehension
- cross-sentence reasoning

which is why BERT became the state-of-the-art model for **QA** and **NLI** upon release.

What NSP trains BERT to do

NSP teaches BERT to determine whether **sentence B logically follows sentence A**, or if it is **a random, unrelated sentence**.

Example:

- A = “I enjoy reading books.”
  a. B = “I finished a novel yesterday.” → **IsNext**
  b. B = “The dog ran down the street.” → **NotNext**

BERT receives both sentences at once using:

[CLS] **Sentence A** [SEP] **Sentence B** [SEP]

The model must classify: **IsNext** or **NotNext**.

Next Sentence Prediction

- To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence.

Input = [CLS] I enjoy read [MASK] book ##s [SEP]
I finish ##ed a [MASK] novel [SEP]
Label = IsNext

Input = [CLS] I enjoy read ##ing book [MASK] [SEP]
The dog ran [MASK] the street [SEP]
Label = NotNext

- Important for many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI)
- How to choose sentences A and B for pretraining?
  - 50% of the time B is the actual next sentence that follows A (labeled as IsNext)
  - 50% of the time it is a random sentence from the corpus (labeled as NotNext)

BERT Training Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

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: Evaluation

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

- **QQP:** Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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**

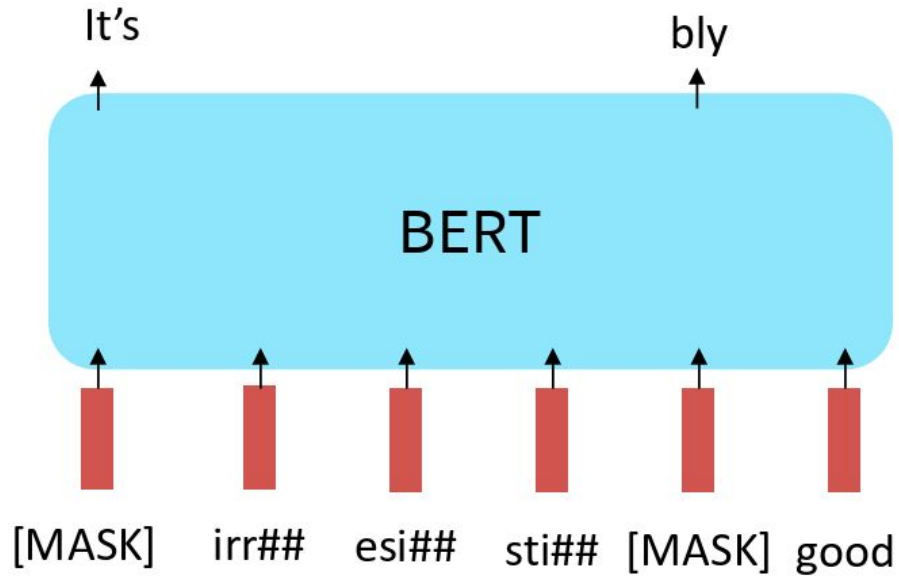
BERT is primarily a **text understanding model**, not a generation model. which is why BERT became the state-of-the-art model for QA and NLI upon release.

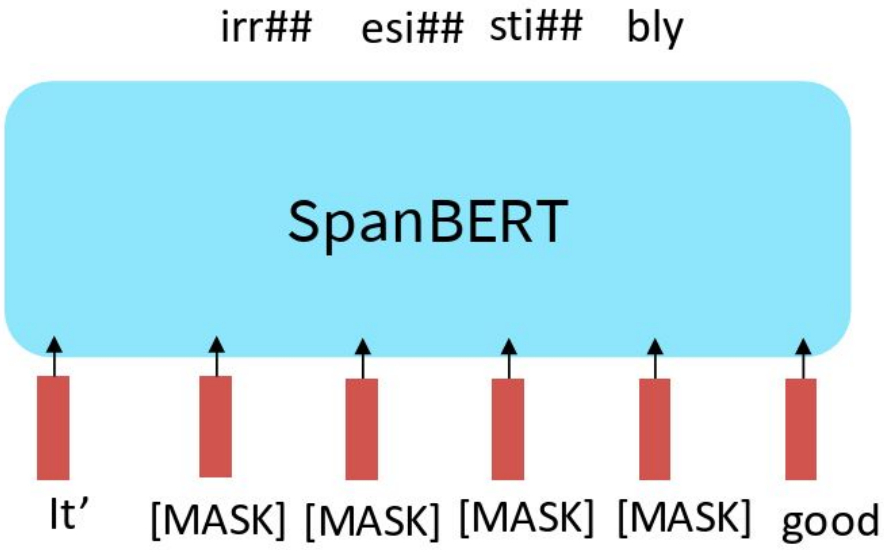
Extensions of BERT

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





Extensions of BERT

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

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

BERT Family

Foundation & Evolution

ORIGINAL MODELS

- **BERT-Base** (Cased/Uncased)
- **BERT-Large** (Cased/Uncased)

KEY IMPROVEMENTS

- **RoBERTa** Enhanced training
- **ALBERT** Parameter reduction
- **DeBERTa** Disentangled attention

EFFICIENT VARIANTS

- **DistilBERT** Distilled (40% smaller)
- **MobileBERT** Mobile-optimized

Specialized Models

DOMAIN-SPECIFIC

- **SciBERT** Scientific papers
- **BioBERT** Biomedical text
- **LegalBERT** Legal documents
- **FinBERT** Finance & trading
- **ClinicalBERT** Healthcare records

MULTILINGUAL

- **mBERT** 104 languages
- **XLM / XLM-R** 100+ languages
- **RemBERT** Enhanced multilingual
- **LaBSE** Sentence embeddings

Africa-Focused Models

CULTURALLY INCLUSIVE AI

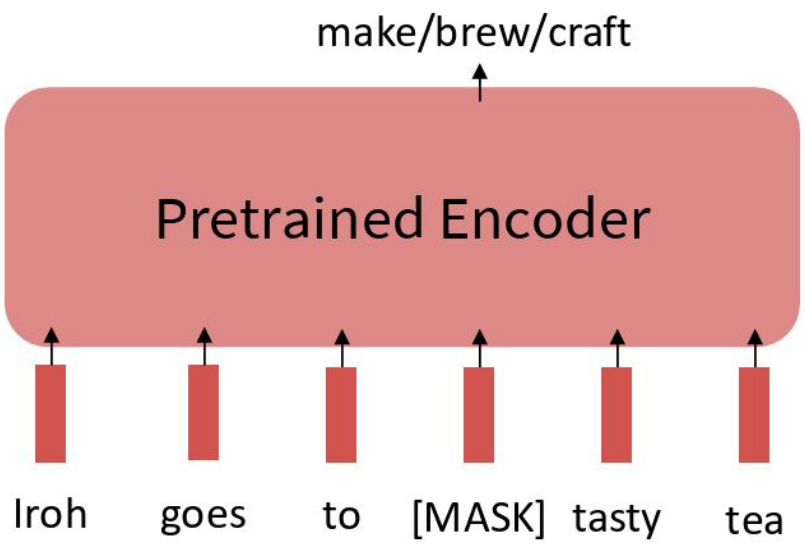
- **AfriBERTa** 11 African languages
- **AfriBERTa-v2** Expanded coverage
- **AfroXLM-R** African-adapted XLM-R

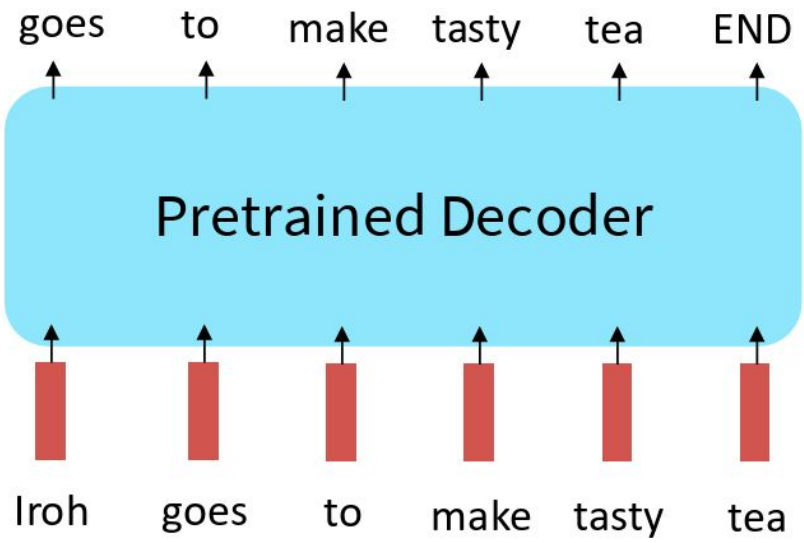
These models address the critical need for culturally-aware NLP systems that perform equitably across diverse African languages and contexts

Limitations of pretrained encoders

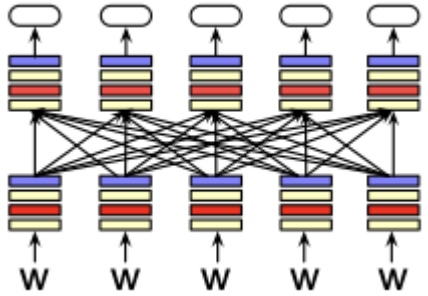
Those results looked great! Why not used pretrained encoders for everything?

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.





Encoders



Strengths

- Learns rich, bidirectional representations of text.
- Excellent for understanding tasks, such as:
  - Text classification
  - Named entity recognition
  - Semantic similarity
  - Question answering (non-generative)

Limitations

- Cannot generate long text naturally (no decoder).
- MLM does not fully reflect natural text generation.

IMPERIAL

Q and A