Training and Finetuning Transformers. Supervised Learning, Transfer Learning, Few-Shot and Zero-shot learning

Venelin Kovatchev

Lecturer in Computer Science

v.o.kovatchev@bham.ac.uk

Outline

- Quick recap
- The encoder-decoder transformer
  - Training a transformer model
- Transfer Learning and Finetuning
  - Decoder transformers: GPT
  - Encoder transformers: BERT
- In-context learning. Zero-shot and Few-shot learning

Quick recap:
Encoder Decoder. Attention. Transformers.

Encoder decoder models

- Mapping between data of different format, size, and structure
  - Two texts of different length and alignment, image and text
- Simple idea
  - Encoder “represents” the source (e.g., English)
  - Decoder “generates” the target (e.g., German)

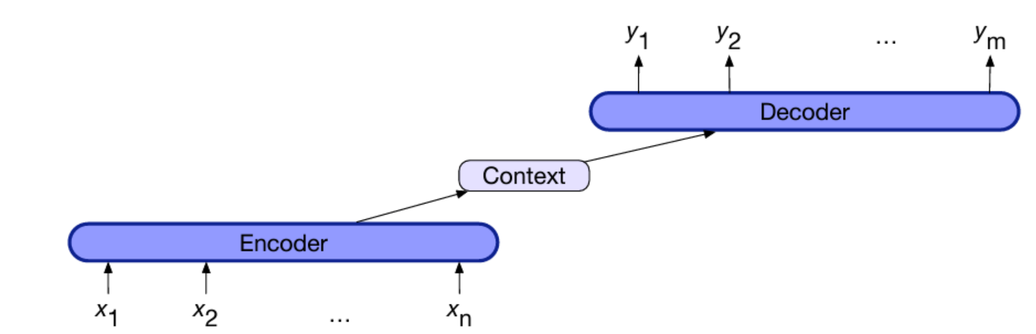


Fig 9.16

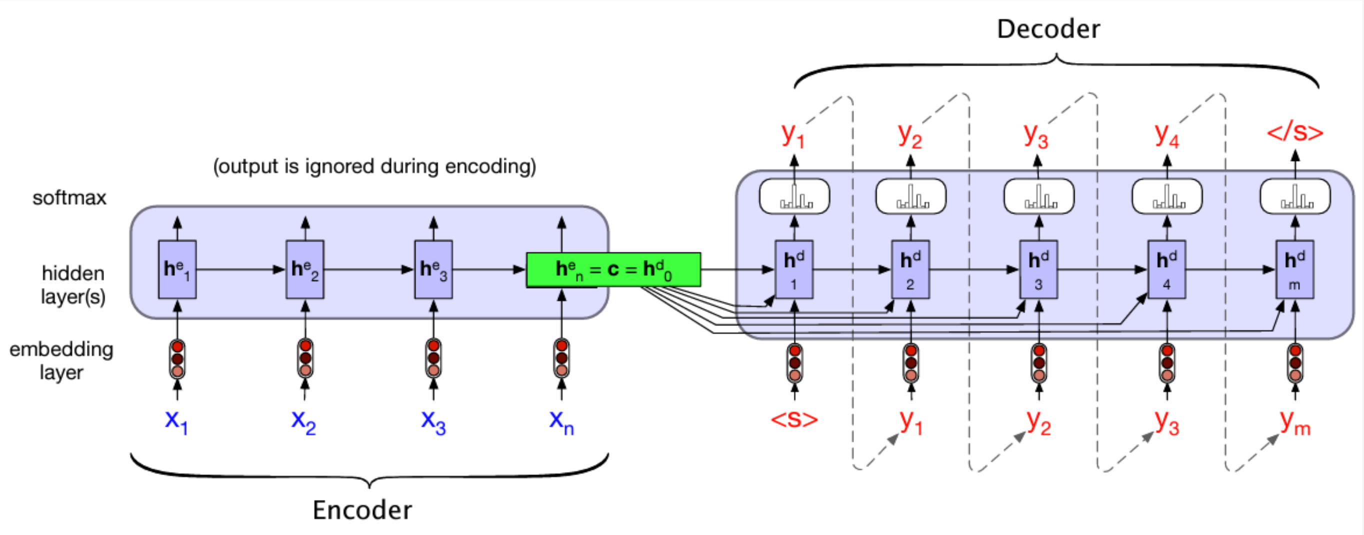
Using RNNs for encoder and decoder

- Encode the input

- Pass the context at every step

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$

- Limitations
  - Single vector representation is a bottleneck
  - Long distance dependencies in source

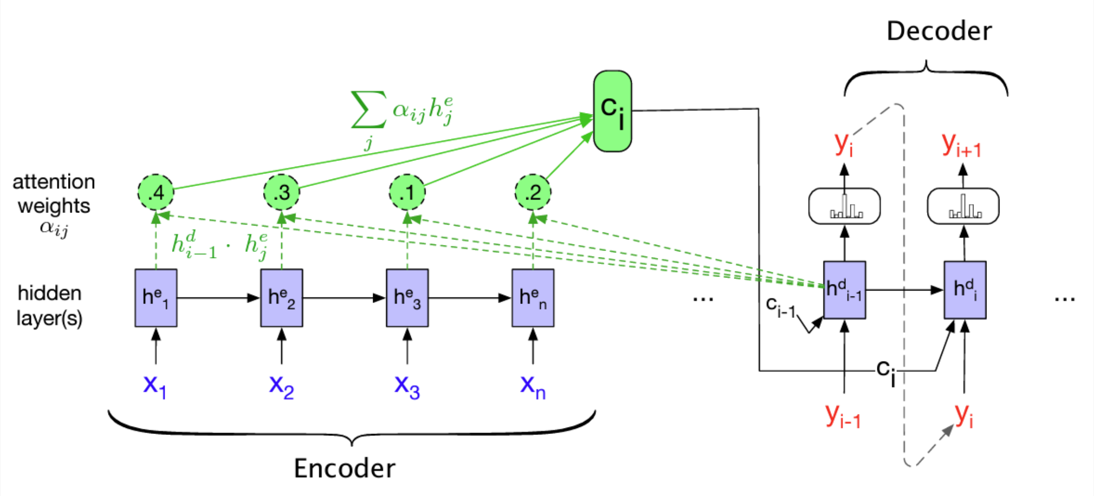


Attention – intuition

- Intuition: each token in the target should use a "personalized" context

- Access all the hidden states in the encoder
  - Still needs to have a fixed length, regardless of variable input length

- The context – weighted sum of all hidden states



Dot product attention (formally)

- Scoring function:

$$\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{i-1} \cdot \mathbf{h}^e_j$$

- Weight vector:

$$\begin{aligned}\alpha_{ij} &= \mathrm{softmax}(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j)) \\ &= \frac{\exp(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j))}{\sum_k \exp(\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_k))}\end{aligned}$$

- Personalized context:

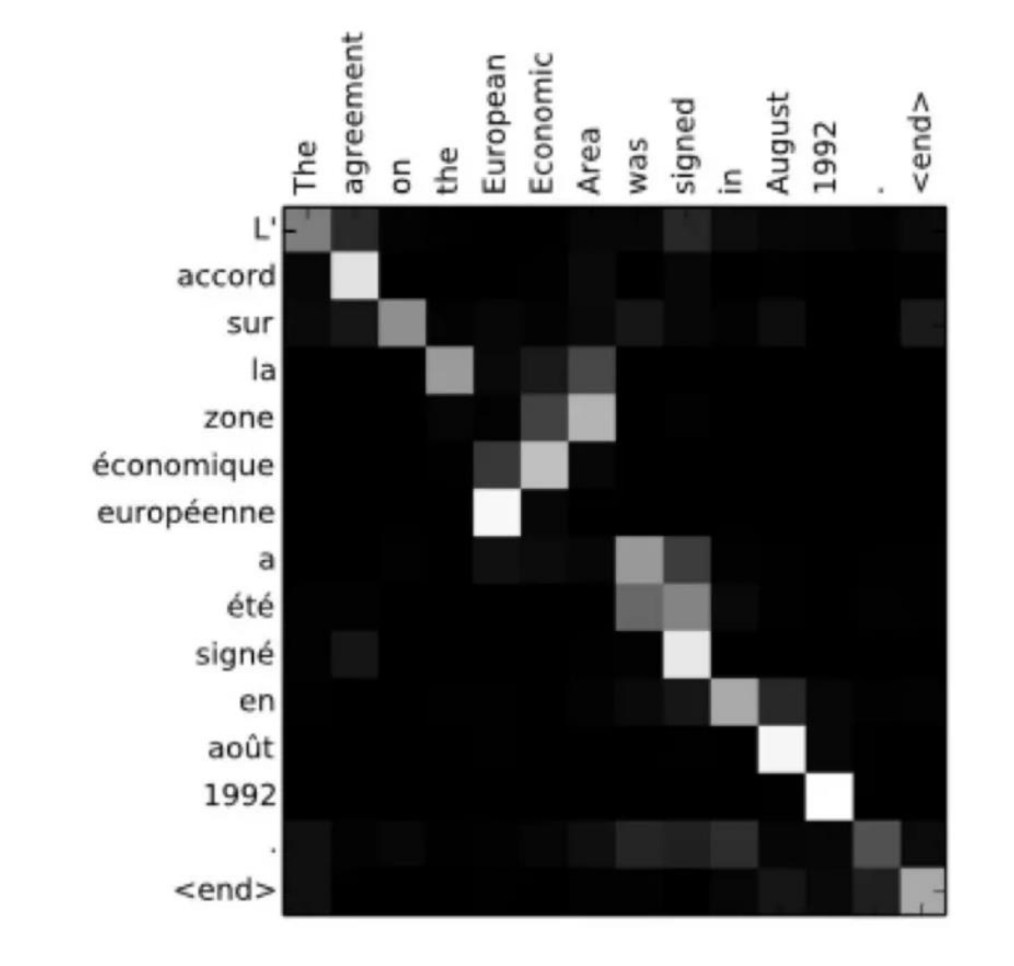
$$\mathbf{c}_i = \sum_j \alpha_{ij}\, \mathbf{h}^e_j$$

- More complex scoring functions:

$$\mathrm{score}(\mathbf{h}^d_{i-1}, \mathbf{h}^e_j) = \mathbf{h}^d_{t-1} \mathbf{W}_s \mathbf{h}^e_j$$

Visualizing attention

- Linear weights are interpretable
- We can see which word is more important
- Can we use attention for explainability?
- How would a context of an RNN/LSTM look for enc-dec?



Do we need LSTM?

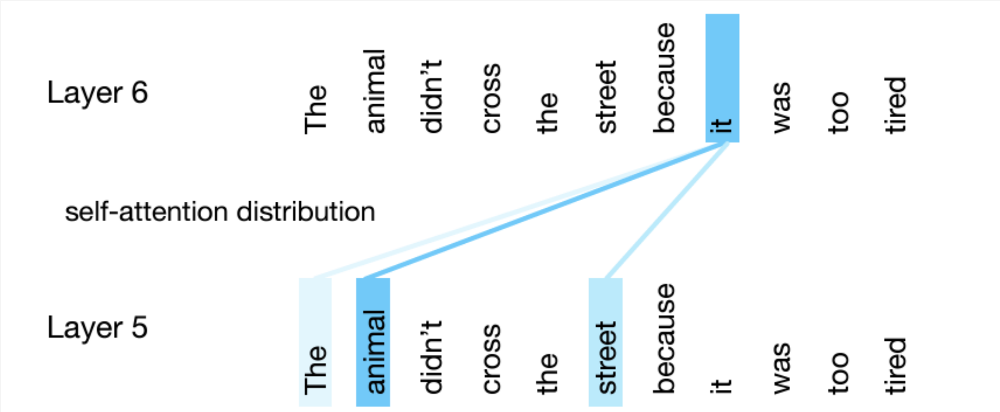
- Original implementation: two LSTMs + attention

- Do we need LSTM?
  - Can process input sequentially
  - Some long-distance dependencies
  - Can’t be parallelized (why?)

- Can we replace RNN/LSTM with attention?
  - “Attention is all you need”

Self attention

- Attention is used to inform the decoder of relevant context
  - It models relations external to the model



- Self-attention can replace RNN/LSTM for compositionality
  - It models internal relations within the same layer

The query, key, and value

- Single representation can be a bottleneck
  - The representation x_i has multiple roles
  - Dot product is commutative
  - Learning can be inefficient

- Learn different representations (projections) for each role

The query, key, and value (2)

- We project the input vector x to three vectors that serve different purpose: “query”, “key”, and “value”
- Two vector operations in the original attention:
  - “Score”: for indexes i and j, calculate how important is x_j for x_i: score(x_i, x_j)
  - “Scale”: for index i, calculate the hidden state h_i as a weighted sum of x_1 … x_i: $h_i = \sum_{j \le i} \alpha_{ij} x_j$
- Each input vector x can three different roles
  - Argument 1 in score() [“dog” in score(“dog”, ”black”)] -> **query**
  - Argument 2 in score() [“dog” in score(“black”, “dog”)] -> **key**
  - The **value** used in scale to calculate the hidden state

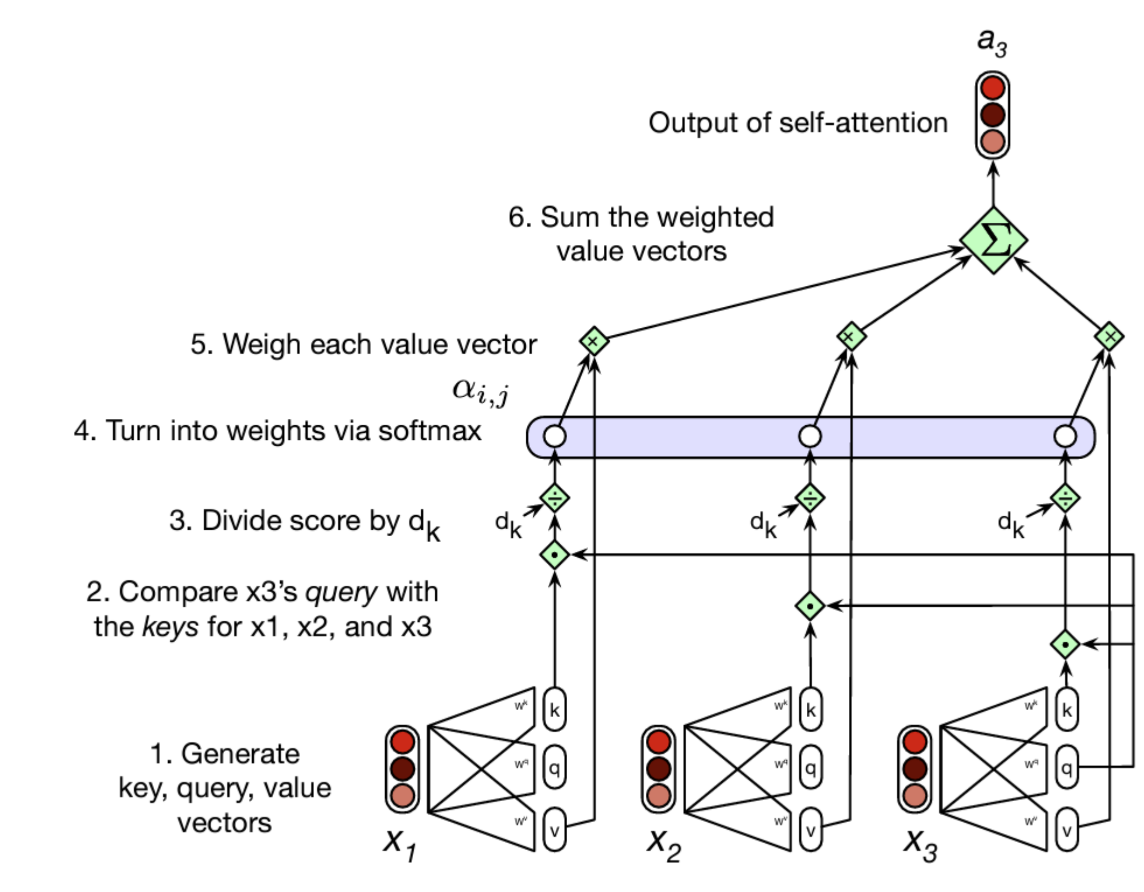
The transformer self attention

1. $$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \mathbf{k}_i \;=\; \mathbf{x}_i \mathbf{W}^{\mathbf{K}}; \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{V}}$$

2. and 3. $$\text{score}(\mathbf{x}_i, \mathbf{x}_j) \;=\; \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

4. $$\alpha_{ij} \;=\; \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\forall j \leq i$$

5. and 6. $$\mathbf{a}_i \;=\; \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

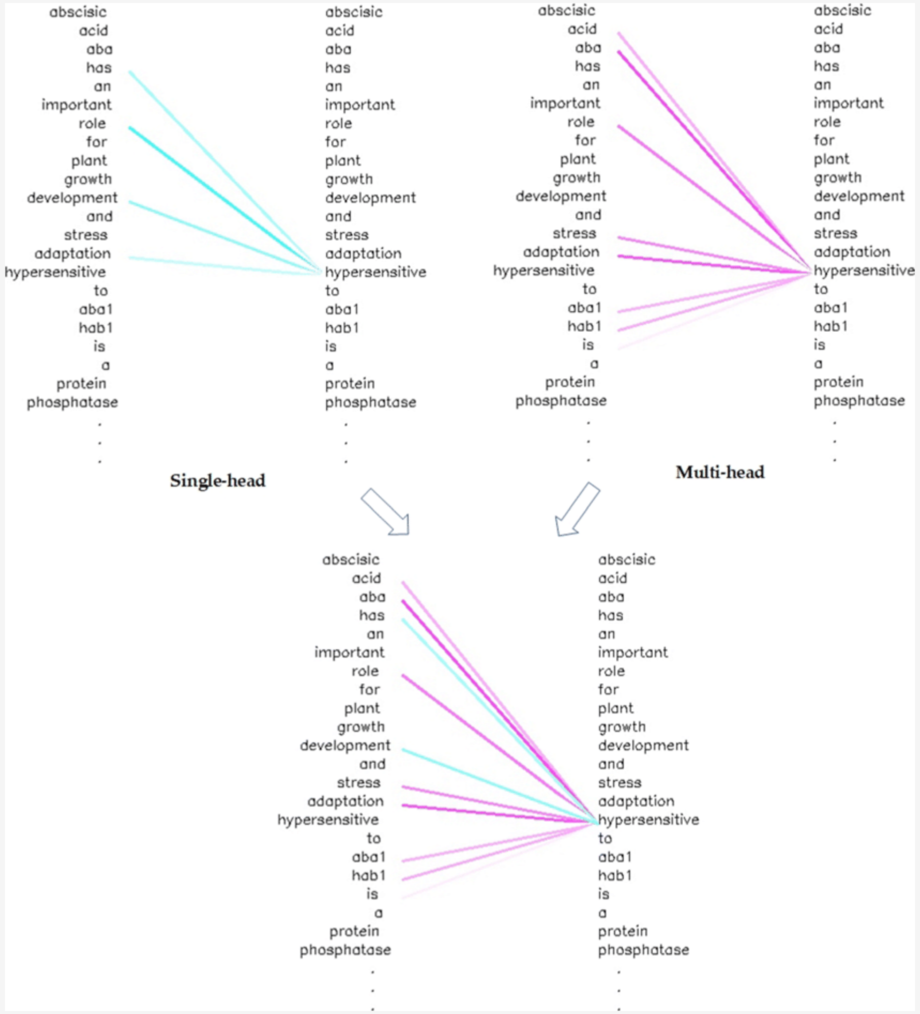


Multiheaded self-attention

- Instead of using a single self attention, we can use multiple
  - Each "head" has its own weights W^Q, W^K, W^V
  - The outputs of all heads are concatenated and projected to input dimensions
- Formally:

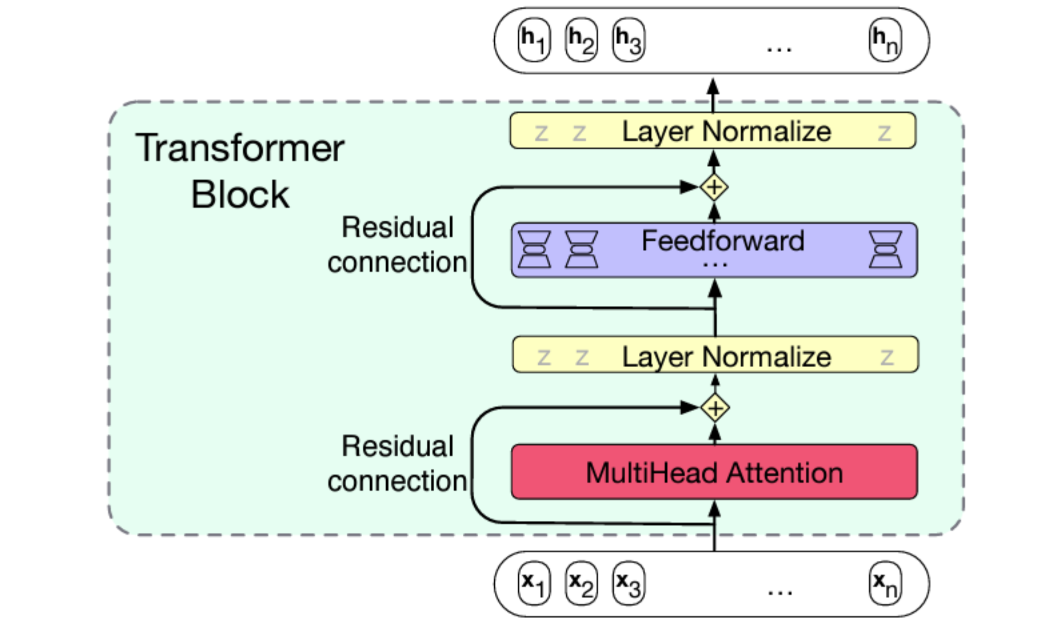
$$\begin{aligned} \mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \,;\, \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \,;\, \mathbf{V} = \mathbf{X}\mathbf{W}_i^V \\ \mathbf{head}_i = \mathrm{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ \mathbf{A} = \mathrm{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O \end{aligned}$$

- Intuition:
  - Similar to using multiple filters in CNN



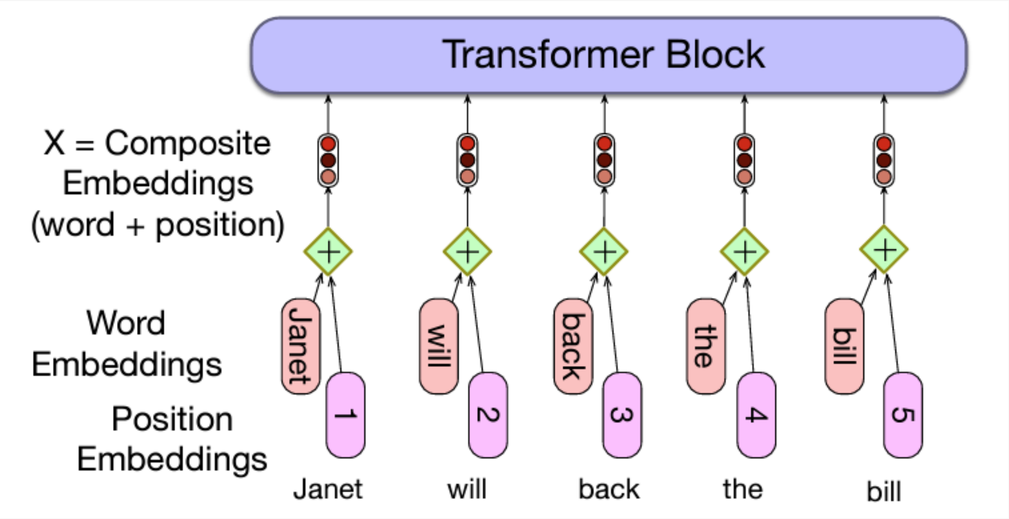
The transformer block

- Residual connection
  - Copy the input of a layer to its output
- Layer normalize
  - Rescale each x vector to 0-mean with STD=1
- Feedforward
  - Apply the same fully connected FFN to each x



Encoding the Input. Positional Embeddings.

- Semantic embeddings
  - One-hot encoding maps to a row in a matrix
- Positional embeddings
  - One embedding for each position
  - Learnable; Same dimension as semantic
- Add semantic and positional embeddings



- Why do we need positional embeddings?

Using functions for positional embeddings

- Learning representations for positions can be a problem – data sparsity

- Using mathematical functions to encode position as a vector
  - Using the position, calculate (deterministically) a vector representation
  - OG transformer approach – using sine and cosine

- Same underlying concept – use positional embedding to modify the semantic embedding

Positional embeddings with sine and cosine

- Given: Input length L, number of dimensions d, constant n

- For each $k = 0$ to $L - 1$:
  - For each $i = 0$ to $\frac{d_{model}}{2}$:
    - $PE_{(k,2i)} = sin(\frac{k}{n^{\frac{2i}{d_{model}}}})$
    - $PE_{(k,2i+1)} = cos(\frac{k}{n^{\frac{2i}{d_{model}}}})$

$$PE_{(k,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$PE_{(k,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$\overbrace{\hphantom{[sin(\tfrac{0}{10000^{\frac{0}{4}}}), cos(\tfrac{0}{10000^{\frac{0}{4}}}), sin(\tfrac{0}{10000^{\frac{1}{4}}}), cos(\tfrac{0}{10000^{\frac{1}{4}}})]}}^{d\_model = 4}$$

$$P(0) = \left[sin(\frac{0}{10000^{\frac{0}{4}}}), cos(\frac{0}{10000^{\frac{0}{4}}}), sin(\frac{0}{10000^{\frac{1}{4}}}), cos(\frac{0}{10000^{\frac{1}{4}}})\right]$$

$$k \begin{cases} P(1) = \left[sin(\frac{1}{10000^{\frac{0}{4}}}), cos(\frac{1}{10000^{\frac{0}{4}}}), sin(\frac{1}{10000^{\frac{1}{4}}}), cos(\frac{1}{10000^{\frac{1}{4}}})\right] \\ P(2) = \left[sin(\frac{2}{10000^{\frac{0}{4}}}), cos(\frac{2}{10000^{\frac{0}{4}}}), sin(\frac{2}{10000^{\frac{1}{4}}}), cos(\frac{2}{10000^{\frac{1}{4}}})\right] \\ P(3) = \left[sin(\frac{3}{10000^{\frac{0}{4}}}), cos(\frac{3}{10000^{\frac{0}{4}}}), sin(\frac{3}{10000^{\frac{1}{4}}}), cos(\frac{3}{10000^{\frac{1}{4}}})\right] \\ P(4) = \left[sin(\frac{4}{10000^{\frac{0}{4}}}), cos(\frac{4}{10000^{\frac{0}{4}}}), sin(\frac{4}{10000^{\frac{1}{4}}}), cos(\frac{4}{10000^{\frac{1}{4}}})\right] \end{cases}$$

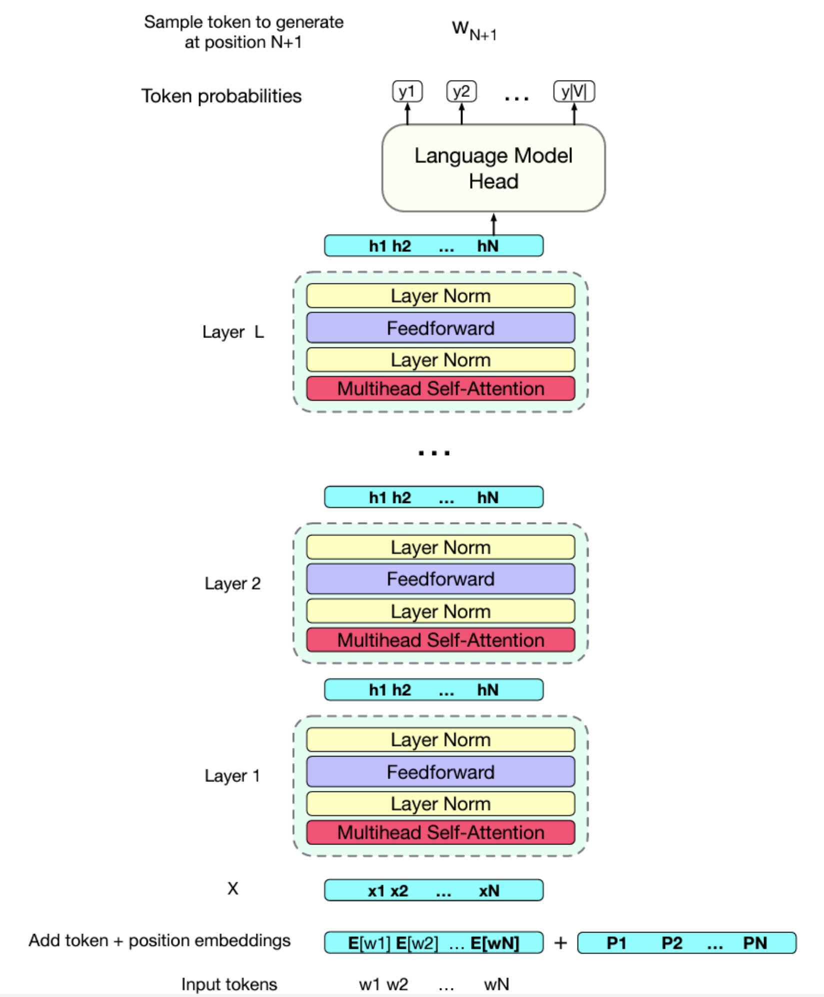
$$P(5) = \left[sin(\frac{5}{10000^{\frac{0}{4}}}), cos(\frac{5}{10000^{\frac{0}{4}}}), sin(\frac{5}{10000^{\frac{1}{4}}}), cos(\frac{5}{10000^{\frac{1}{4}}})\right]$$

$$i{=}0 \qquad i{=}0 \qquad i{=}1 \qquad i{=}1$$

- Generate even dimensions using sin, odd dimensions using cos
  - Example with L=6, d=4, n=10000

A final transformer representation for LM

- Token + positional embedding
- Multiple stacked transformer blocks
- A classification head
  - To train, we need a task and an error function
  - Language modeling with weight tying and sampling



The original transformer

An encoder-decoder architecture using two transformers

- The original transformer is an encoder-decoder used for machine translation
- Both encoder and decoder have 6 stacked layers
- 8 multiheads, 64 dimensions per head, hidden size of 512
- Sin/Cos positional embeddings

Image

- Encoder
  - Bi-directional attention can “see” all tokens
  - Follows the architecture we have seen last week
- Decoder
  - Causal attention
  - Additional Multiheaded Attention
    - Why do we need it?
    - What would be the Q, K, V used by it?

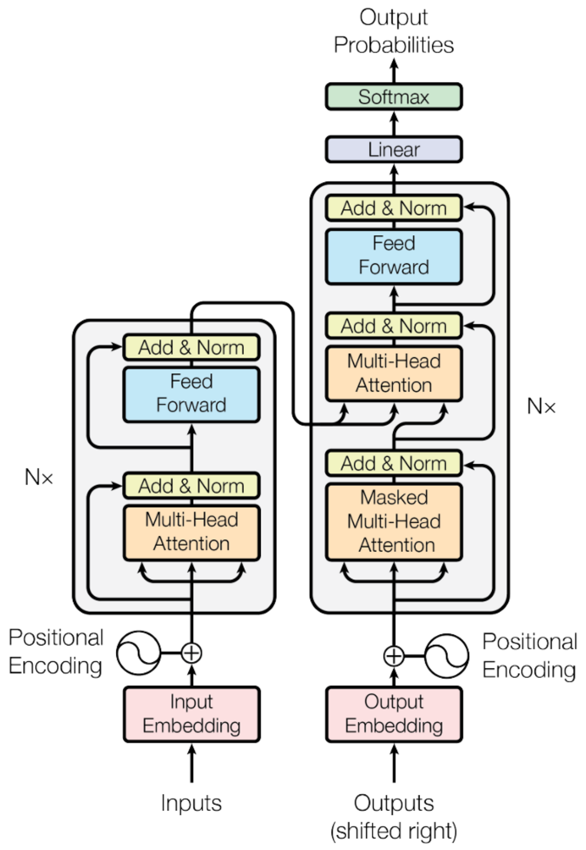


Figure 1: The Transformer - model architecture.

Training configuration

- Training set:
  - 4.5 million En-De sentence pairs; 36 million En-FR sentence pairs
- Hardware and time:
- Parameters and specifications:
  - 65 million parameters for base; 213 million for large
  - 37k BPE token vocabulary for EN-DE; 32k for EN-FR

Text generation from a probability distribution

- The output of a neural LM is a probability distribution
- How do we choose which word to generate?
  - Process called "decoding"
  - Efficient decoding is still a challenge
  - Any suggestions?

Random sampling

- Random sampling – choosing a word at random, according to its probability
- Pop quiz: When do we stop?
- Objectives of sampling:
  - Quality: how good (coherent/likely/factual) is the generated text
  - Diversity: how boring/repetitive/biased is the generated text
  - Can these objectives be aligned?

Restricting the sampling

- Greedy sampling – always pick the most probable word
- Top k sampling – randomly sample one of the k most probable words (re-scaling the probability to sum to 1)
    - How would greedy and top k perform in terms of quality and diversity?
- Top p sampling – select tokens that represent p percentage of the probability mass
- Temperature sampling – modify the probability distribution
    - Low temperature(<1) – more probability to frequent tokens; High temperature – more probability to unfrequent
    - How: divide the logit by temperature: $y = \mathrm{softmax}(u/\tau)$

Training a transformer model

- Original transformer is trained on translation data
  - Uses bilingual data
- Calculate the error at each token (use teacher forcing)
- Aggregate the error across the target language (e.g. French)
- Backpropagate the error through both encoder and decoder

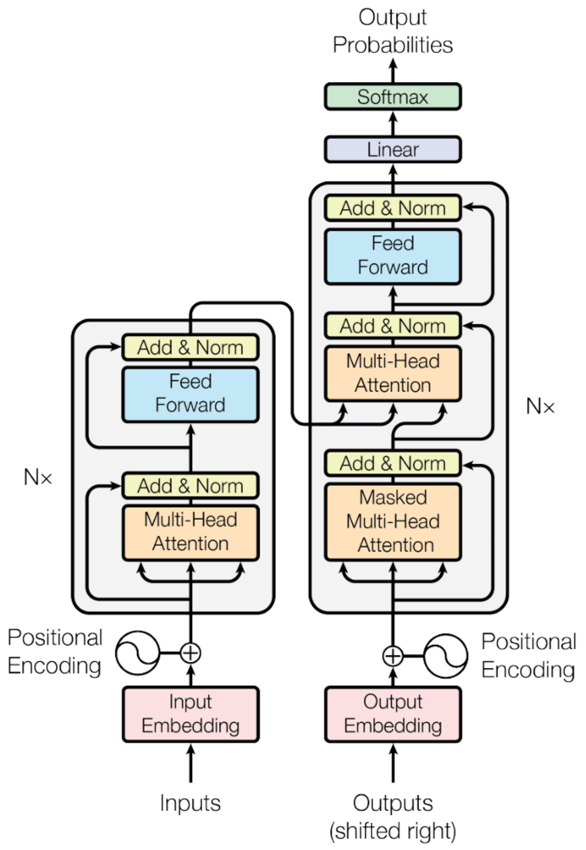
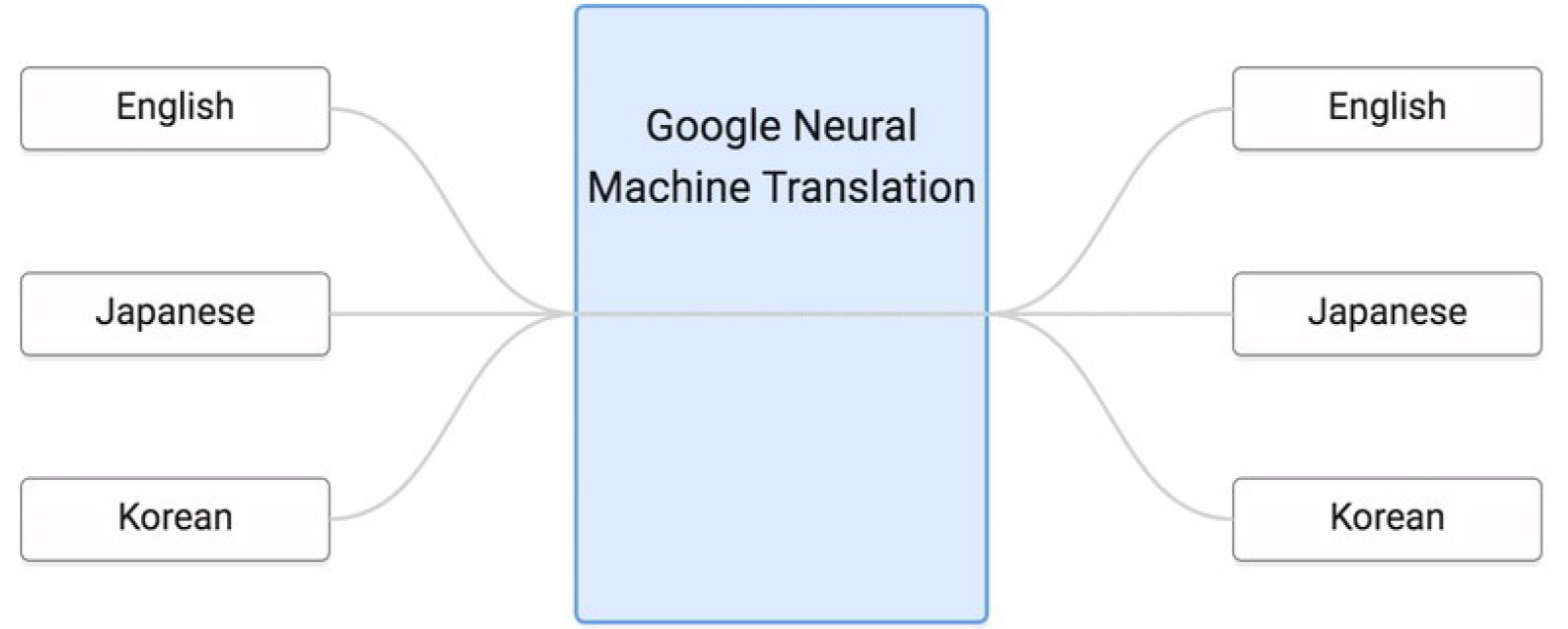


Figure 1: The Transformer - model architecture.

Multilingual NMT

- To train encoder-decoder for EN-DE, you need bilingual data in EN-DE
    - E.g., books translated from EN to DE
- To train encoder-decoder for EN-FR, you need bilingual data in EN-FR
- Do you need bilingual data for every pair that you want to train?

Multilingual NMT



https://research.googleblog.com/2016/11/zero-shot-translation-with-googles.html

Transfer Learning
BERT and GPT

The concept of transfer learning in vision

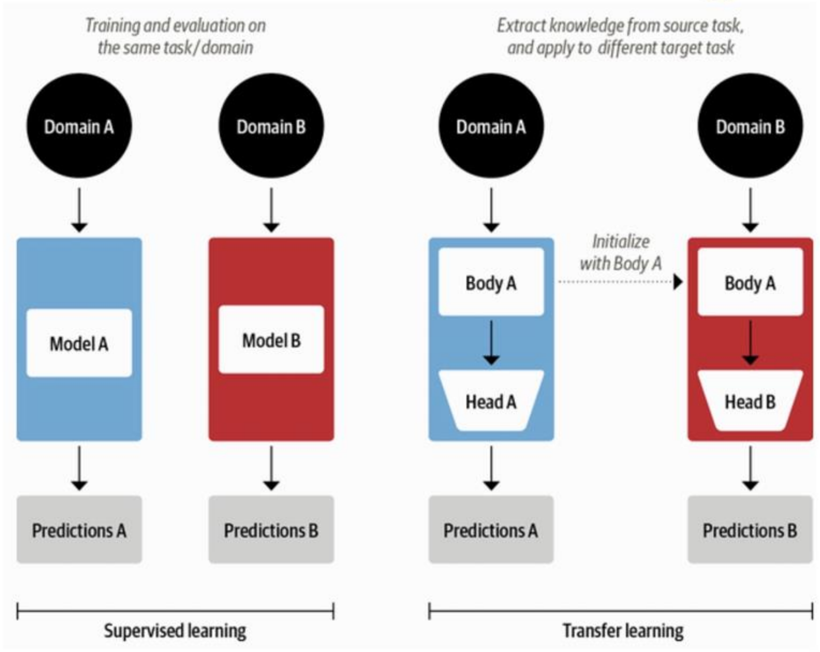
- Transfer learning has ben used in vision for a while
- Pretrain and finetune idea
  - Pretrain to capture general features and properties
  - Finetune to learn task-specific weights and compositions
- Can we do that for language?
  - Embeddings have some limited success

Supervised learning vs Transfer learning

- What are the goals and benefits of transfer learning?

- What are some potential issues or risks?

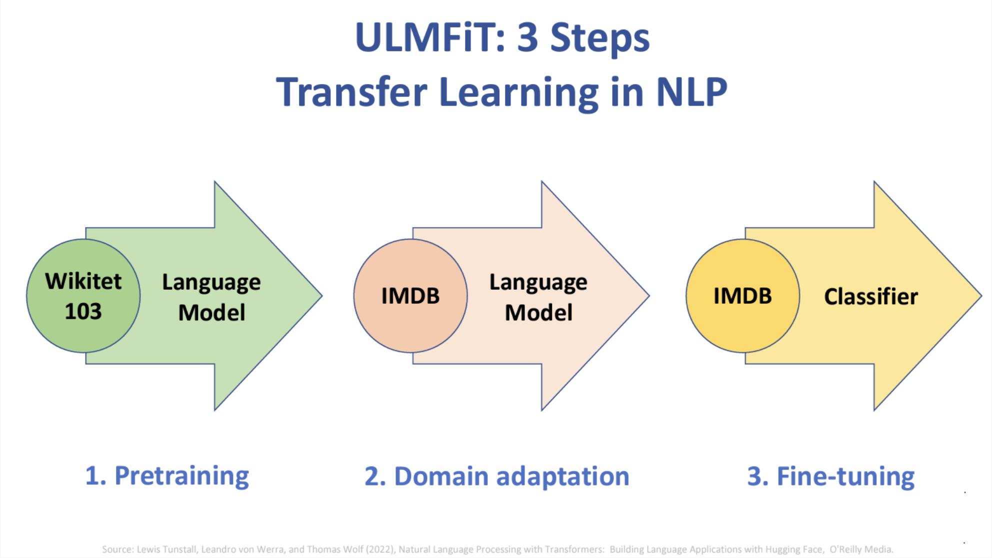
- Are there any other paradigms that you can think of in that area?



Source: Lewis Tunstall, Leandro von Werra, and Thomas Wolf (2022), Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media.

ULMFiT (Howard and Ruder et al., 2018)

- “Universal Language Model Finetuning”
- Training a language model for “inductive transfer learning”
  - Model trained on a source task (language modeling)
  - Finetuned with limited data on target task
- Language modeling – the analogy of ImageNet
- Base model - BiLSTM



ULMFiT Pipeline

- General-domain LM pretraining
  - Train BiLSTM on Wikipedia
- Target-task LM finetuning
  - Change the LM so it predicts words in a specific domain (e.g. IMDB)
  - Use dynamic learning rate techniques to facilitate training
- Target-task classifier finetuning
  - Add additional layers (heads) for classification: batch normalization, dropout, relu, and a final softmax

ULMFiT Pipeline

- Which parts are reused?

- What happened with last layers?

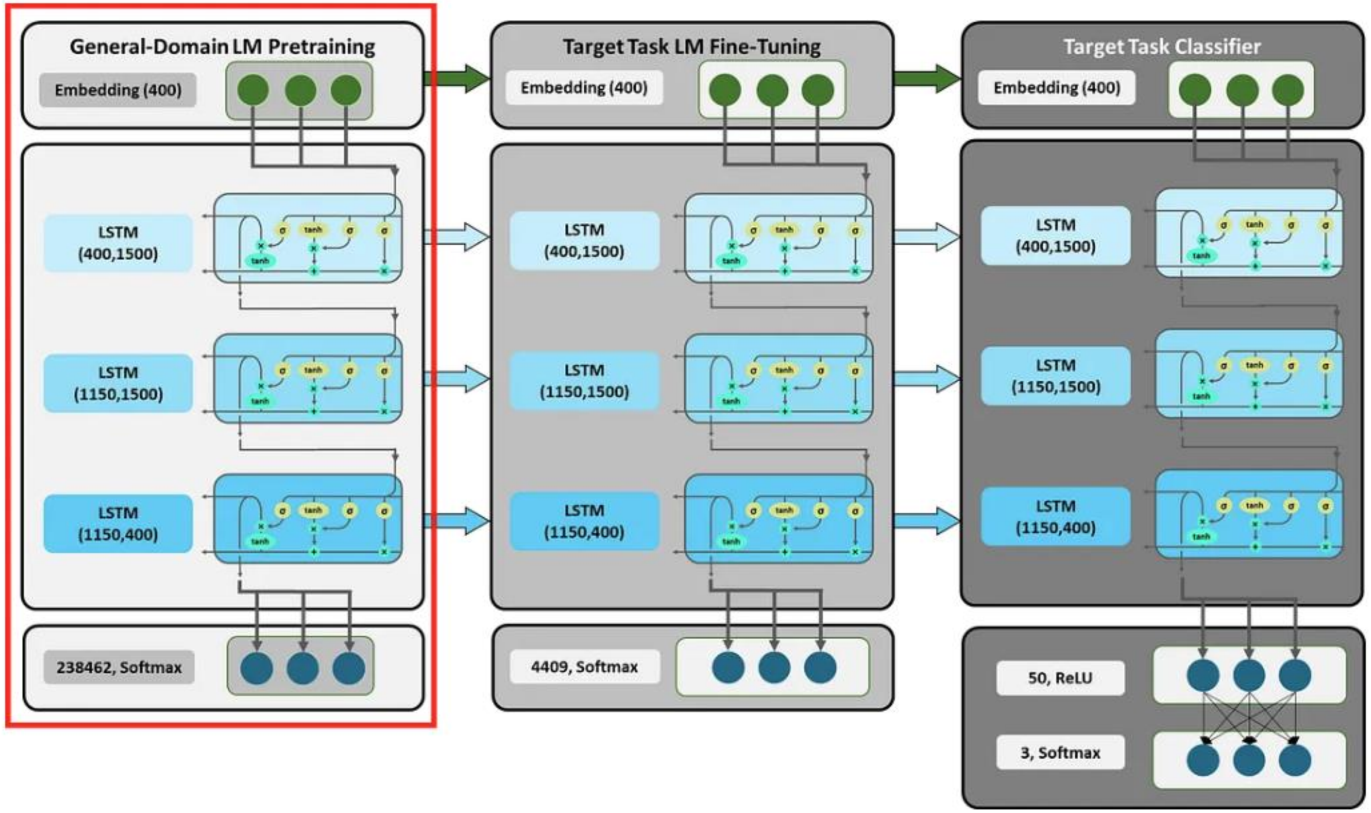


Figure 5: Block Diagram of ULMFiT in Text Analysis — Image from by HU-Berlin from GitHub

As used in https://towardsdatascience.com

Layer Freezing and Catastrophic Forgetting

- Catastrophic forgetting is an (unsolved) challenge in transfer learning
  - How to know what to learn and what to forget?
  - Overfitting to the task

- Freezing lower layers
  - Reducing training time
  - Reducing catastrophic forgetting and overfitting
  - Dynamic un-freezing (typically from top to bottom)

Transfer learning and Transformers

- The original transformer was designed for encoder-decoder models
  - Initial application: Machine learning
- Soon after:
  - Encoder models: BERT, ROBERTA, DistilBERT
  - Decoder models: GPT, GPT2
- Reusing the concepts of ULMFiT, enabling transfer learning for multiple tasks

The decoder transformer: GPT

- GPT1 combines different concepts we know so far
  - The standard transformer block
  - Neural Language Modeling
  - Transfer learning capabilities

- Intuition:
  - Generative pre-training
  - Discriminative finetuning

Training GPT

- Self-supervision

- Maximizing the likelihood of the text:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

- Minimizing cross entropy loss

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

- Trained on the Book Corpus (7000 books)

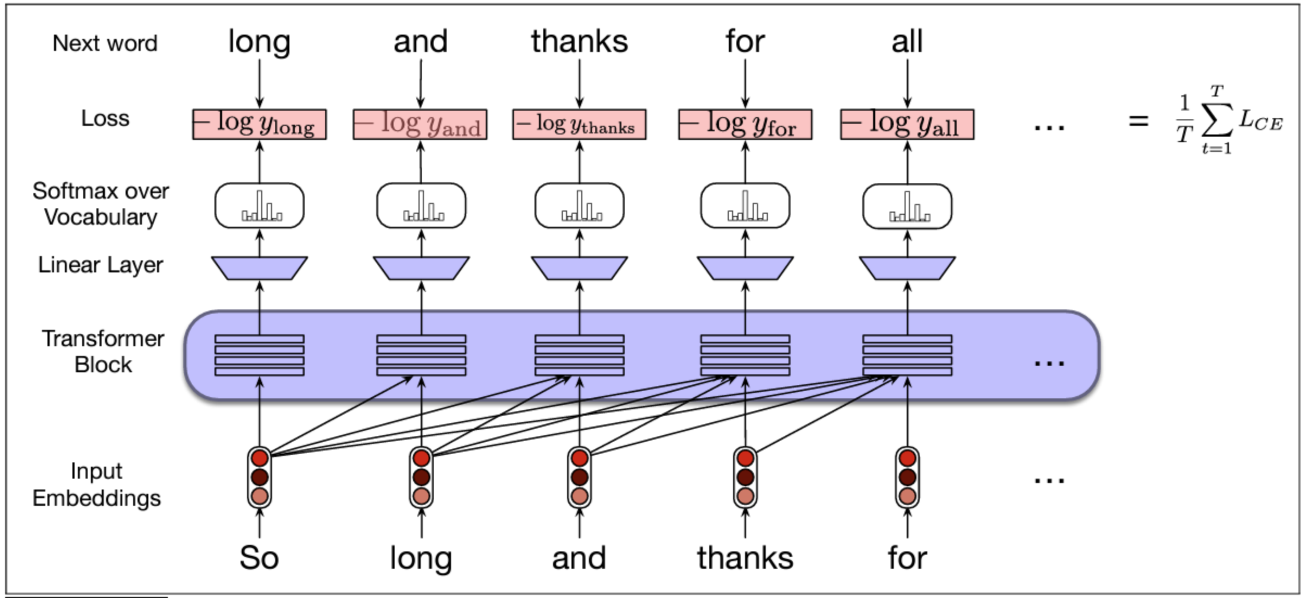


Figure 10.18 Training a transformer as a language model.

Finetuning GPT

- After pretraining, use the hidden state at last layer

- Add a last linear layer with m neurons (m = number of classes)

- Predict the target class:

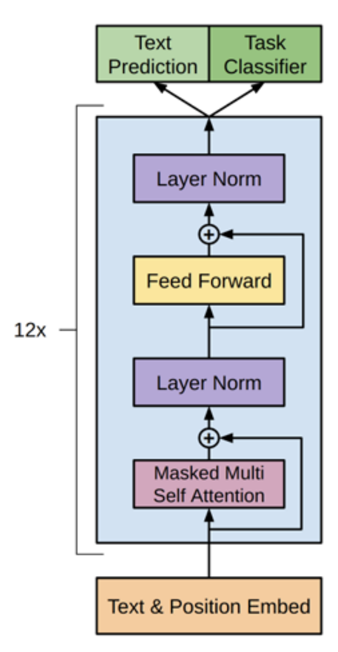
$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

- Maximize the probability of the correct labels (need labeled data)

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

- Combining both losses together

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$



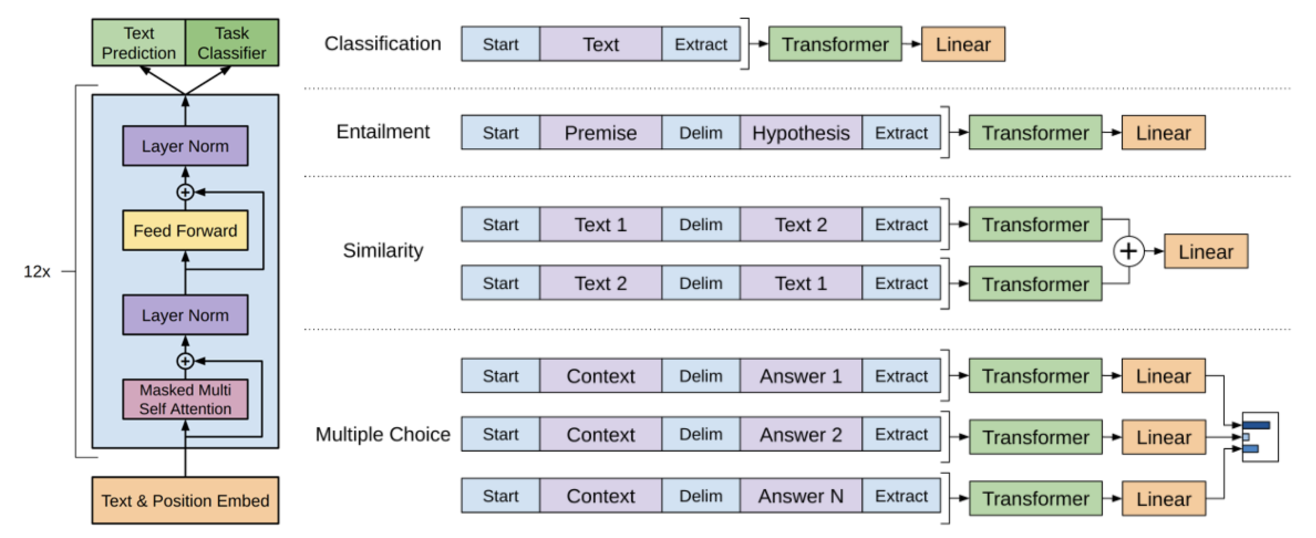
Task specific input transformations

- Out-of-the-box GPT can do:
  - Text generation / Next word prediction
  - Text classification

- How can it perform paraphrase identification?
  - “Is text 1 the same as text 2”
  - Suggestions?

Task specific input transformations

- Task reformulating
- Using special tokens (sep, start/end)
- Comparing separate "streams"
- Task design is a non-trivial task
  - Task formulation; Data format; Metrics and Evaluation



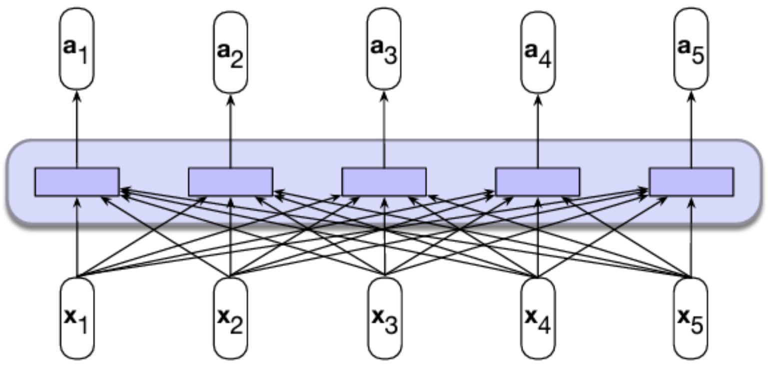
The encoder transformer

- The encoder in “attention is all you need”

- Same architecture as the decoder

- Bi-directional self-attention
    - All key/query values, no masking

- Better for encoding source information



b) A bidirectional self-attention layer

Encoder transformer for classification

- Can an encoder be better for classification?

- Follow the ULMFiT approach
  - Pretrain on a generic task
  - Add task specific layer and finetune

- How do we pretrain?
  - We cannot use “next word prediction”, why?

The encoder transformer: BERT

- The original encoder-only transformer
- An English-only sub-word vocabulary consisting of 30,000 tokens
  - Most of the modern algorithms use subwords tokenizers and embeddings
- 768 hidden size
- 12 layers, 12 heads in each multi-head attention
- 100M parameters
- Trained on two tasks: Masked Language Modeling and Next Sentence Prediction

Masked language modeling objective

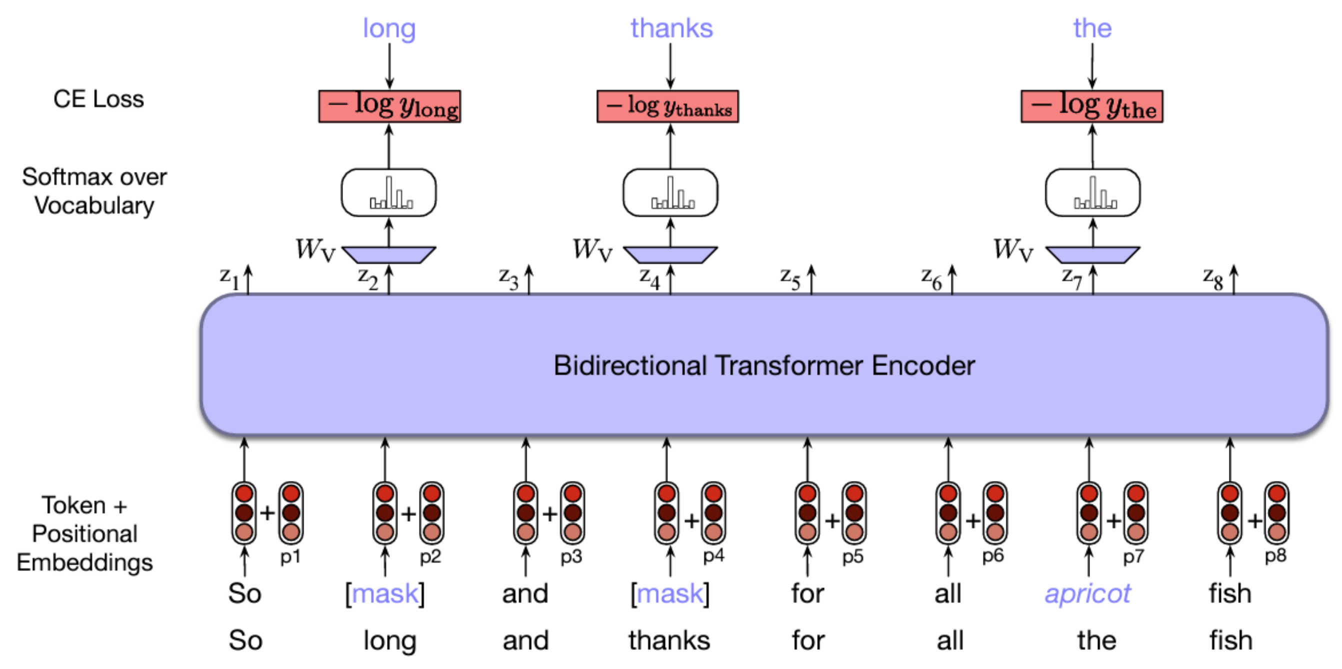
- Based on “cloze” tasks:
  - “Can I have a ____ of water, please?”
  - Does that remind you of something?
- Masked Language Modeling (MLM)
  - Randomly sample tokens from the text and perform alternations
  - Predict the original inputs for each position

MLM and masking

- Rate of sampling – 15% of the input
- Alternations on sampled tokens
    - replace them with [MASK] (80%)
    - replace them with another word (10%)
    - do nothing (10%)
- Use the attention mechanism to calculate the hidden state at all masked positions
  - Calculate cross entropy loss and backpropagate

MLM (visualization)

- In Traditional LMs we predict next token
- In MLM we predict “current” token
- All words participate in attention
- Only “masked” tokens participate in learning



MLM Efficiency and pop quizes

- Is MLM a self supervised approach?

- How efficient is MLM compared to traditional LM

- Why didn’t we have MLM in original encoder-decoder?
  - How did we train the encoder there?

Next Sentence Prediction

- MLM predicts relationships between words
- Transformers want to also process sentences
- Next sentence prediction task
  - Given two sentences, predict whether they are a pair of adjacent sentences

Next sentence prediction. The CLS token.

- Next sentence prediction
  - 50 % true adjacent pairs
  - A special [CLS] token added at the beginning
  - A special [SEP] token added between texts
  - Special “sentence position” (first/second) are added to input
- When predicting the sentence relation, we use the CLS as an input to softmax

The CLS token

- Sentence predictions by BERT are based on the CSL token

$$y_i = \text{softmax}(\mathbf{W}_{\mathbf{NSP}} h_i)$$

- Why do we want to use the CLS token?

- Is there any other way to predict the sentence relation?

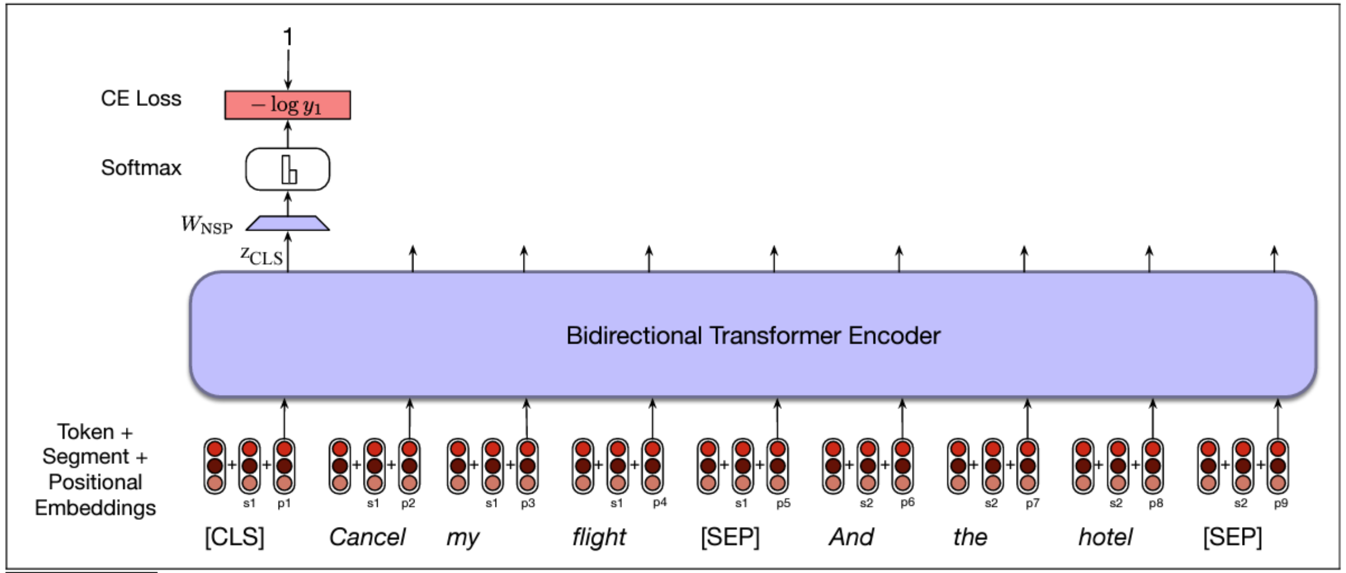


Figure 11.5 An example of the NSP loss calculation.

Representing sentences

- A recurring problem

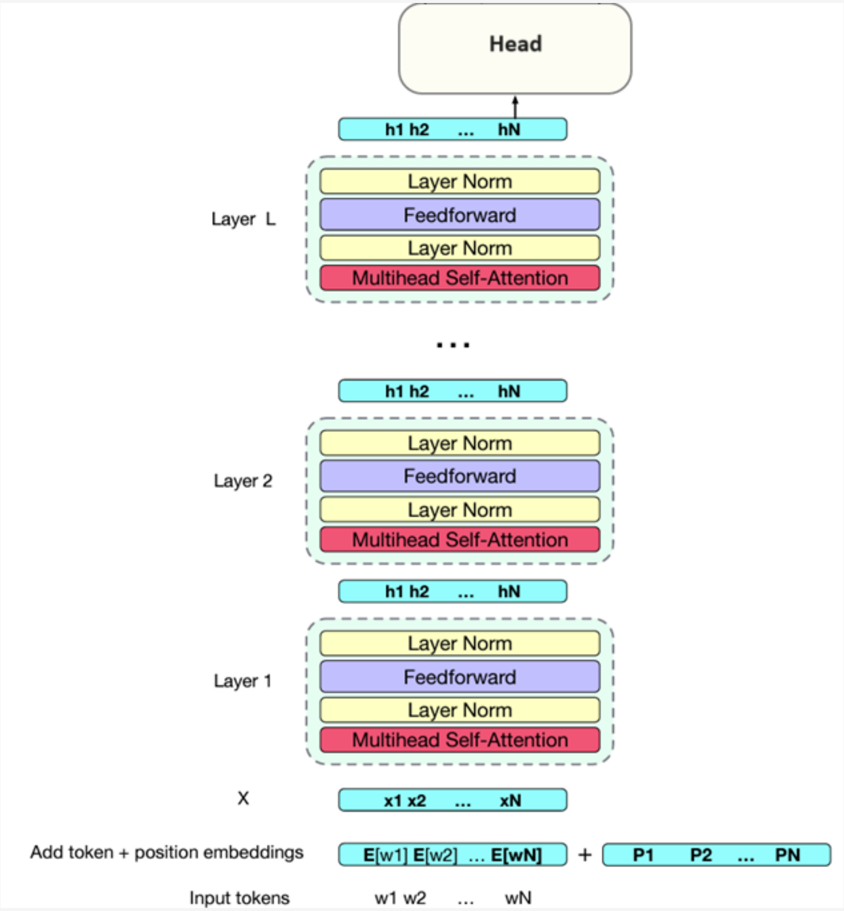
- Word representations
  - Static or contextual

- How do we represent sentences or documents?
  - What strategies have we used before?

Representing sentences

- What strategies can we use to encode the full text?
  - Vector addition
  - Vector concatenation
  - Let the head deal with it
  - Which vector representations are we to use?

- BERT – uses the CLS token instead
  - Learning compositionality as a "special token"



Training BERT

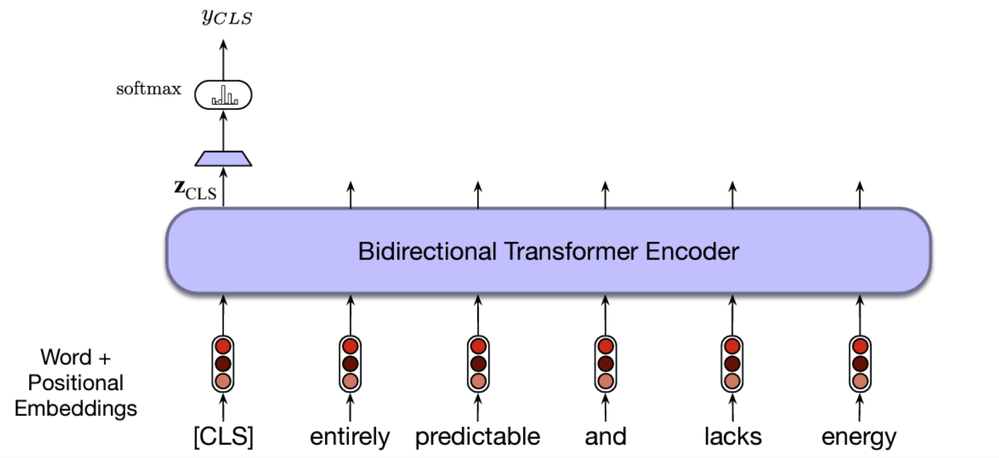
- Combination of Wikipedia and Book Corpus
- Pairing of sentences (true/false pairs)
- Masking of tokens within both sentences
- Combining the MLM loss and NSP loss
    - Which other architecture had two losses? What were they?

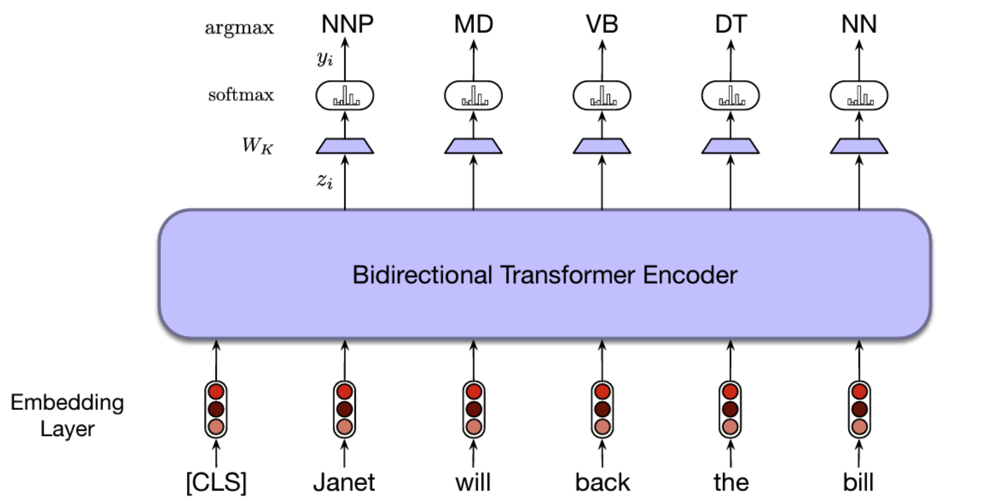
Finetuning BERT

- The same conceptual idea as in ULMFiT and GPT
- Train BERT on web data
- Change the classifier “head” and finetune
  - What would we use as an input to the classifier head?
- What weights are we updating during finetuning?

Adapting other tasks to work with BERT

- How would we perform paraphrase identification?
  - What is the input/output/classification process?
- Performing other tasks:
  - Extractive QA
  - Sequence labeling





Popularity and use of encoder-only models

- Original BERT model achieved SOTA on almost all benchmarks
- Improved models:
  - ROBERTA, DistilBERT, ELECTRA, ALBERT
- Reframing multiple tasks as text classification
- Much better than GPT and GPT2
  - The arrival of GPT3 and increasing scale of training resulted in paradigm shift

What information do transformers capture?

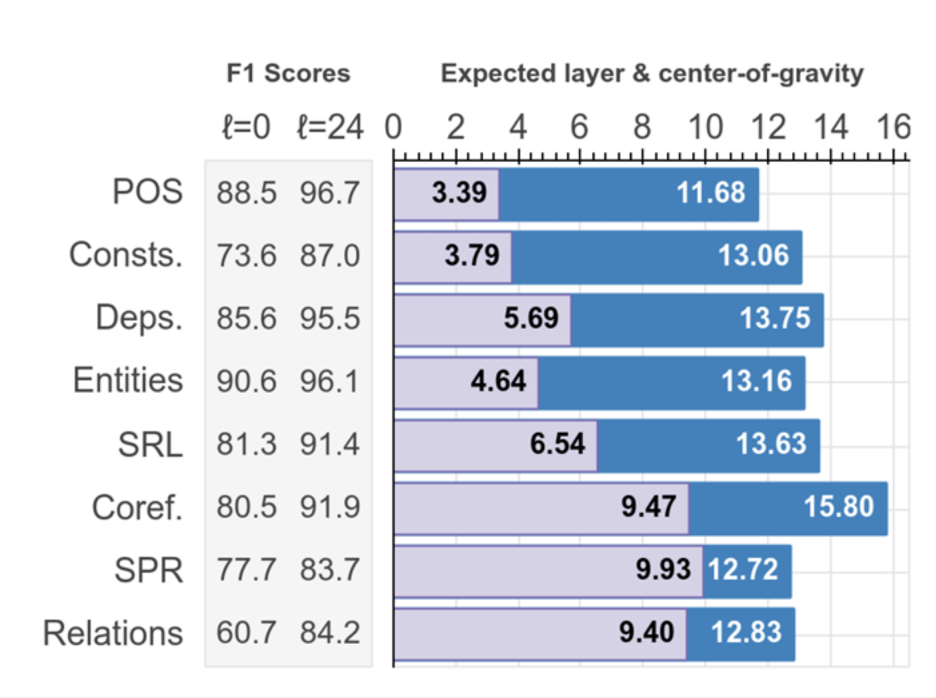
- Pipeline approaches follow a (linguistic) logic
- End-to-end neural models are optimized for a task
  - Difficult to interpret
- Pretraining for CV follows “meaningful” patterns
- What patterns do linguistic pretraining follow?

Bert rediscovers the classical NLP pipeline

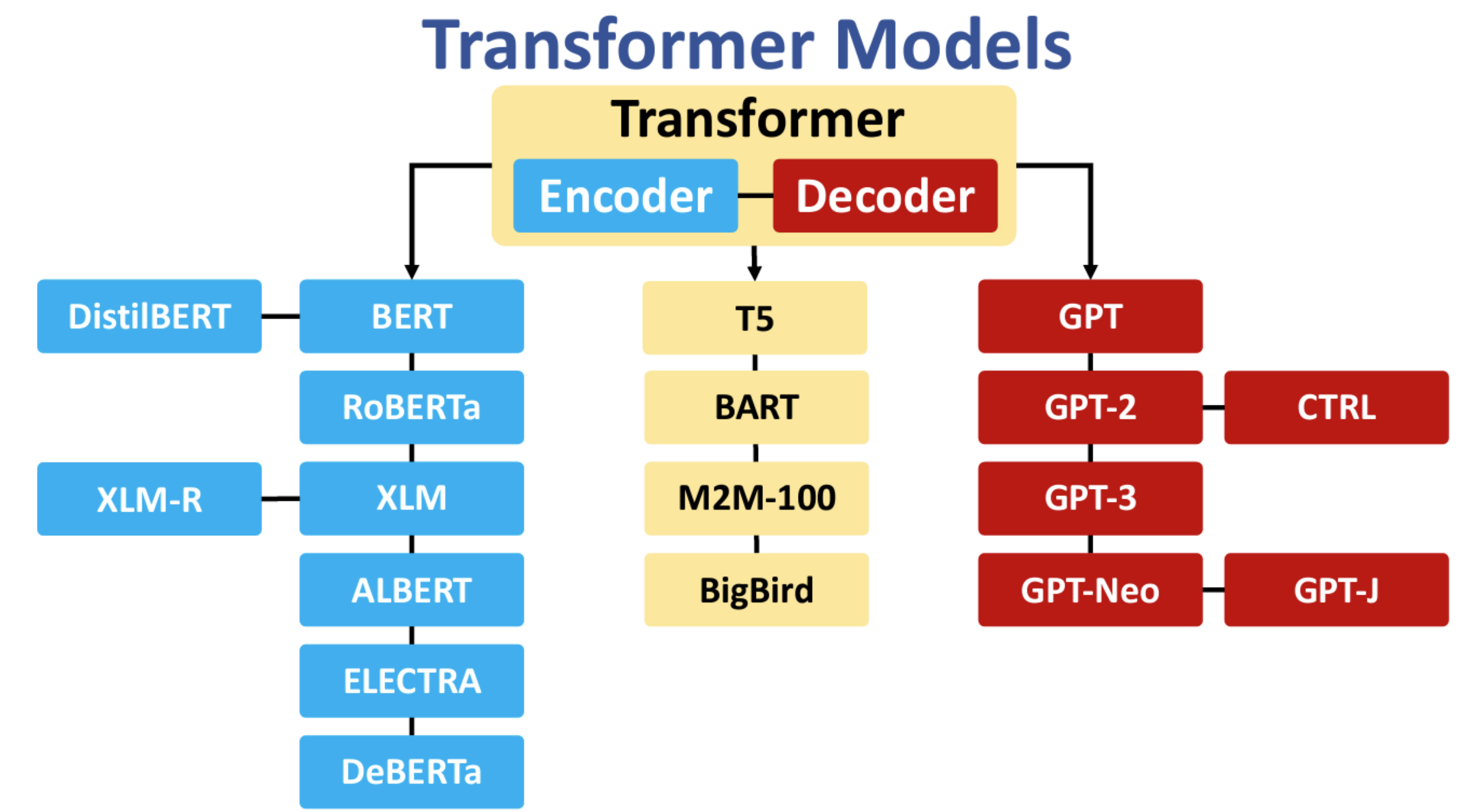
- Probing different layers of BERT for linguistic information
  - Lower layers capture local information; Higher layers capture complex structures
  - Sometimes information can be spread across multiple layers
- Tenney et al. (2019)
  - Does BERT encode traditional NLP preprocessing steps: POS tagging, syntactic parsing…
  - Does BERT follow the same order of operations?
  - What happens with a sentence as it goes through BERT?

Bert rediscovers the classical NLP pipeline (2)

- Two different evaluation criteria
  - Purple – at which layer is most of the information encoded
  - Blue – average layer “used”
- Different linguistic properties are “discovered” in order
  - “Early” properties have a long “tail”



The transformer family tree



Source: Lewis Tunstall, Leandro von Werra, and Thomas Wolf (2022), Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media.

Scaling laws

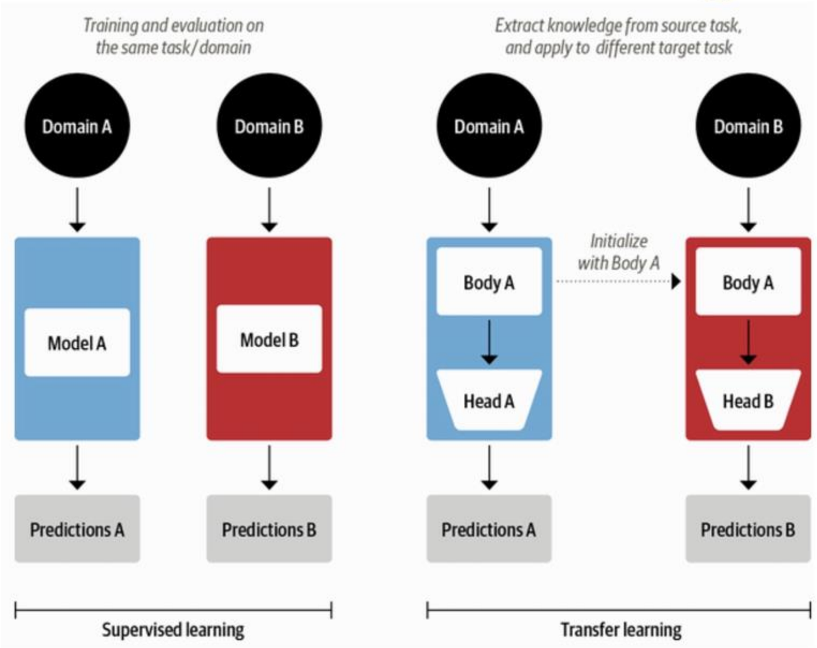
- What makes LLMs better and how much better can they be?

- The performance of LLMs depends on three factors:
    - Model size
    - Data size
    - Compute power
- The (expected) performance of different models can be represented as a function of those factors
    - Kaplan et al. (2020) proposes “scaling laws” for LLMs

Path towards AGI?
Zero-shot and Few-shot learning

Supervised Learning vs Transfer Learning

- In supervised learning, we train from scratch
- In transfer learning, we only change the head
- Can we go further in reducing need for data?
  - How do humans perform tasks?



Source: Lewis Tunstall, Leandro von Werra, and Thomas Wolf (2022), Natural Language Processing with Transformers: Building Language Applications with Hugging Face, O'Reilly Media.

NLP Paradigm shifts

- Several key papers driving paradigm shifts in NLP
  - Word2Vec – Mikolov et al. (2013)
  - End-to-end LSTM/BiLSTM and GRU – multiple papers in 2014
  - Attention – Bhadanau et al. (2014)
  - Transformer – Vaswani et al. (2017)
  - BERT – Devlin et al. (2019)
  - GPT 3 – Brown et al. (2020)
  - Instruct-GPT – Ouyang et al. (2022)

GPT3

- A major shift in NLP paradigm
  - Arguably the largest shift since moving from pipeline to end-to-end models
- Introduced few-shot and zero-shot learning
  - Teaching a model to perform a task without changing the weights (!)
- In-context learning and prompt engineering

GPT3 and current LLMs

- Almost all of current chat-enabled LLMs are based off the concepts in GPT3
- Newer models are performing better
  - More parameters; larger and better datasets
  - Additional training: supervised and RLHF finetuning
  - Human-driven and machine-driven prompt engineering
- GPT3 can, in principle, do anything that modern LLMs can
  - The difference is in the implementation, not in the core concepts (!)

Few-shot and Zero-shot learning

- The problem
  - Getting training data is complex and expensive (there are far more tasks than datasets)
  - Overfitting (to spurious correlations)
  - Humans don’t need large training data for all tasks
- The goal
  - One model that can perform multiple tasks
  - In-context learning
  - AGI?

In-context learning vs supervised/transfer learning

- Using the input to specify the task
- Consider the following inputs to a transformer model:
  - “I like this movie, it’s the best in the Avengers series!”
  - “I bike to work every day. <SEP> I drive to work every day.”
- What is the task? What is the output?
  - The task is what you train the model to do
  - The first sentence can be an input to a NER model
  - The second sentence can be an NLI task or a similarity task

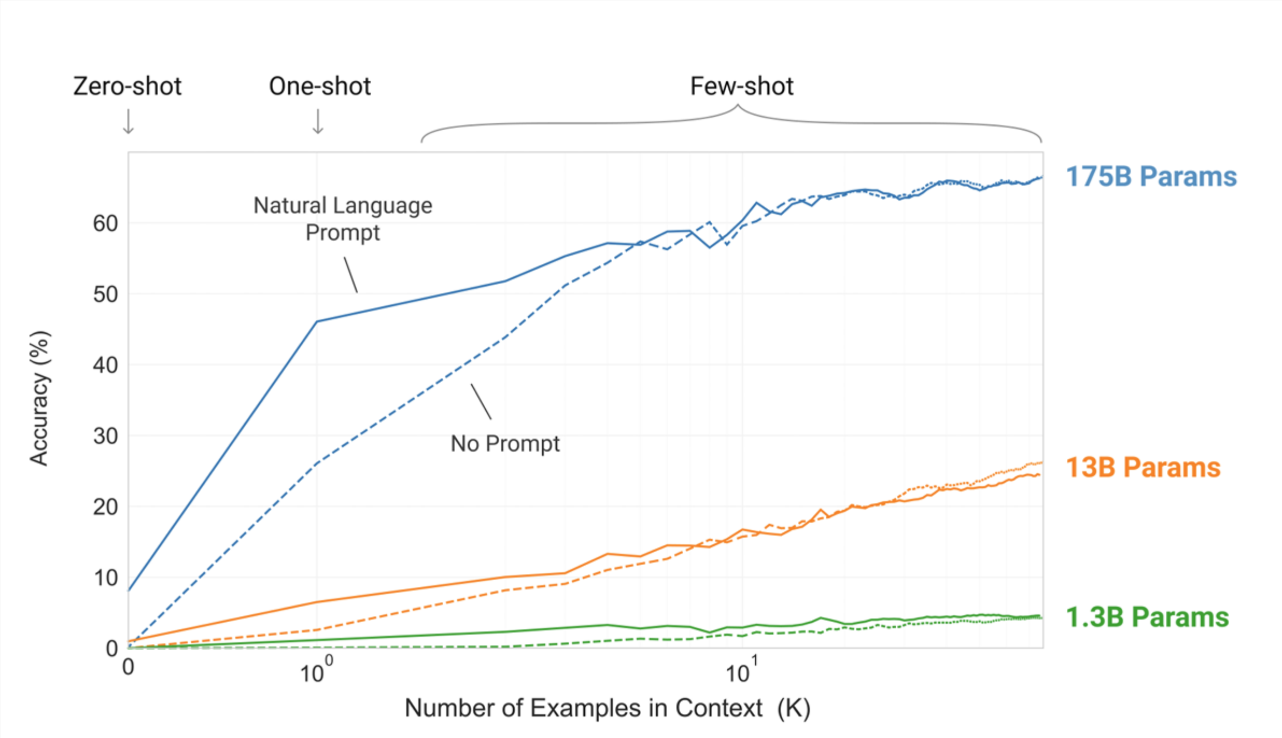
In context learning

- Now consider the following inputs:
  - “What is the sentiment of the following text: I like this movie, it’s the best in the Avengers series!”
  - “Do those sentences contradict each other: I bike to work every day. <SEP> I drive to work every day.”

- How do we achieve in-context learning?
  - GPT3 paper argues that scale and emerging properties are the answer
  - Increasing model size from 17B to 175B

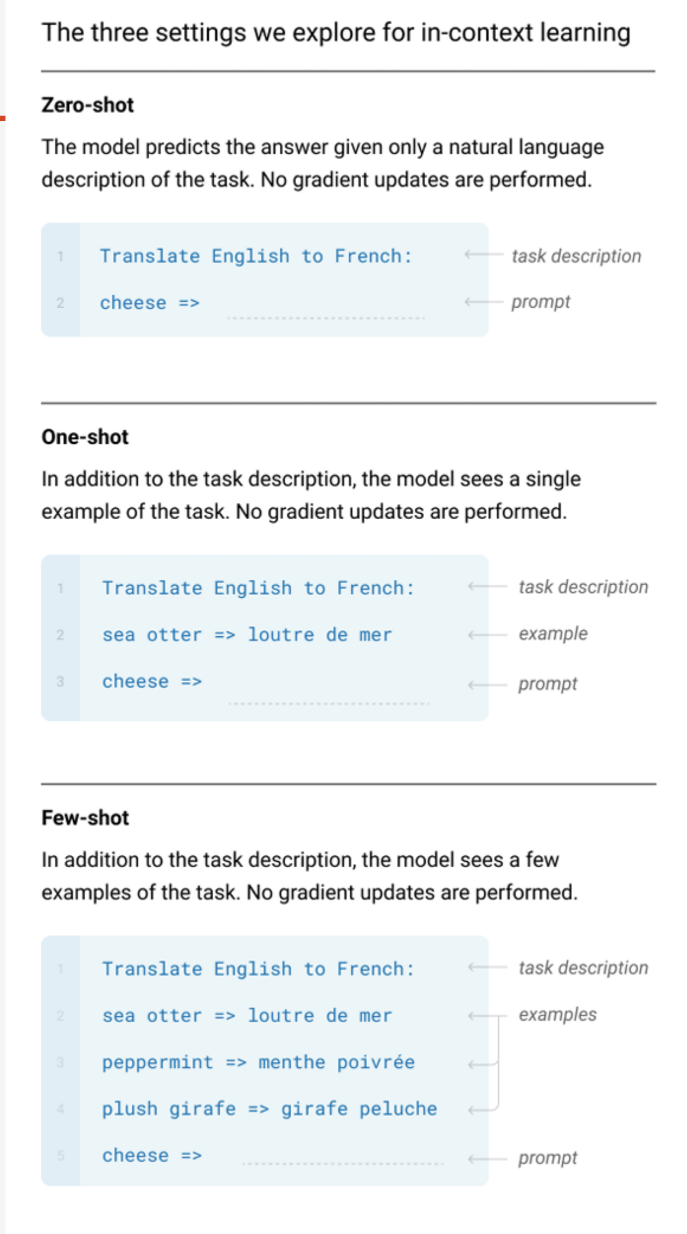
In-context learning and model size

- Two key factors
  - Model size
  - Number of examples
- Model size -> “emerging properties”



Zero- One- and Few-shot learning

- Three different experimental conditions
- No gradient update or finetuning
- The only difference – number of examples

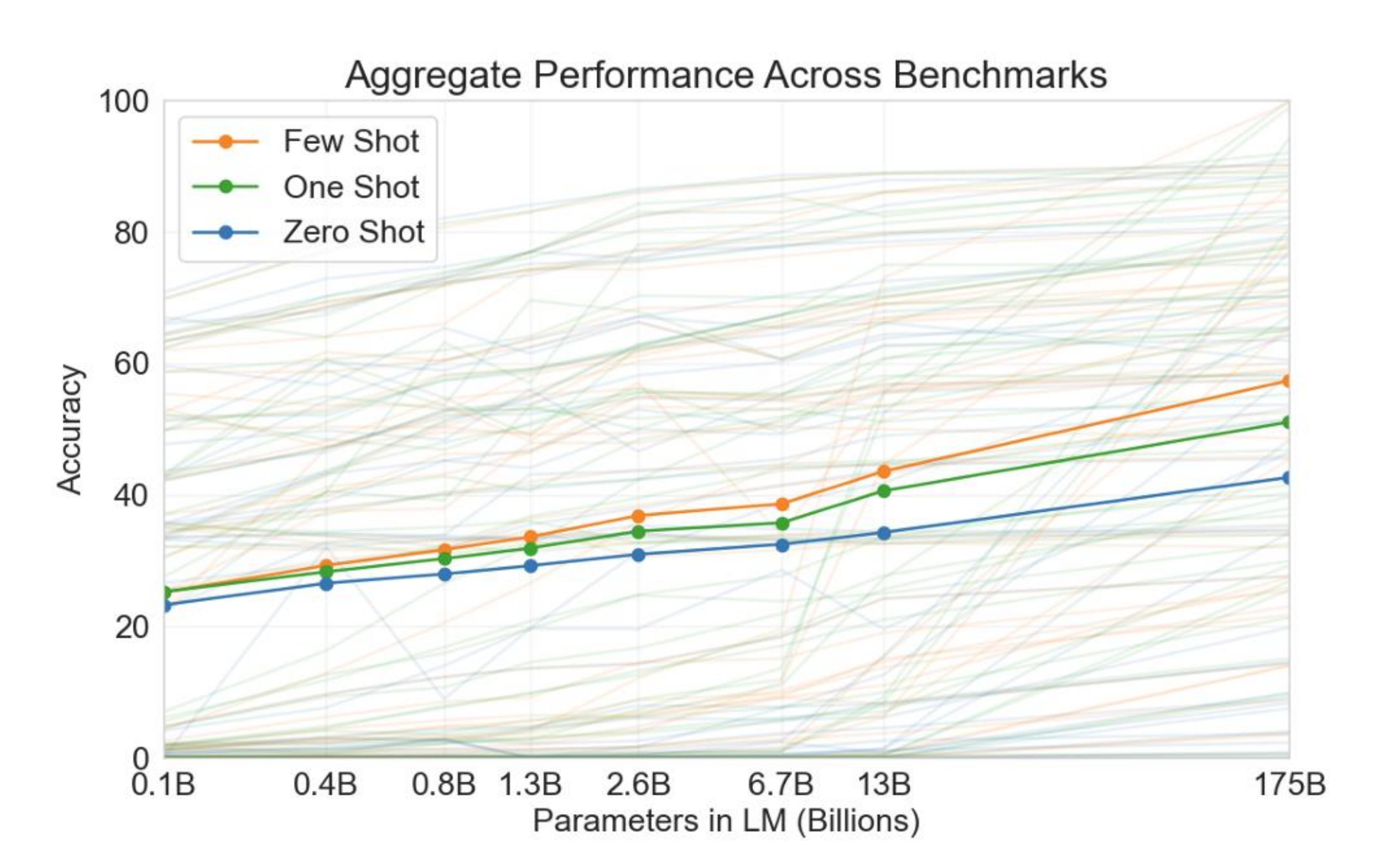


In-context learning and model size

- Same model

- Different sizes

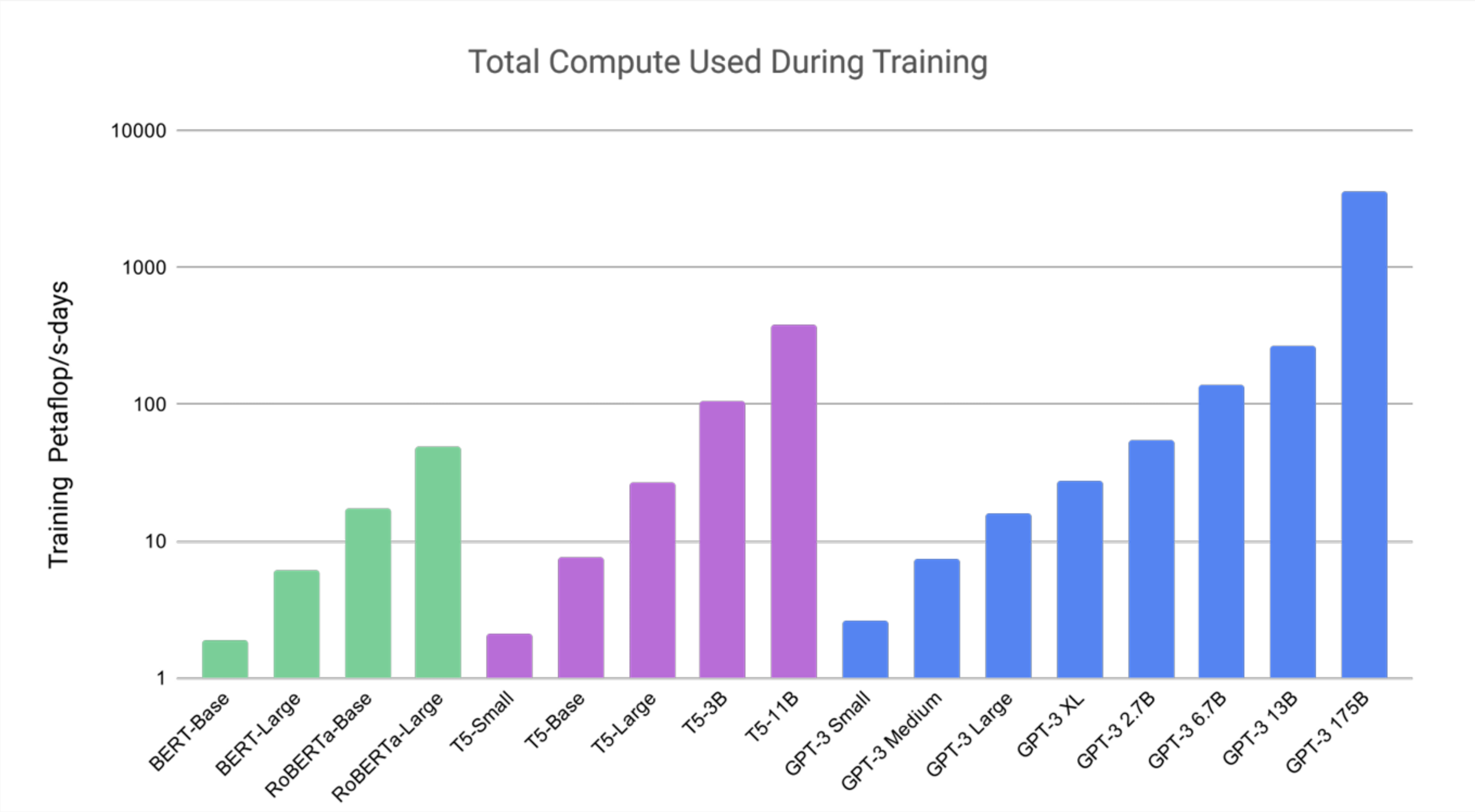
- Different number of examples



Mode architecture

- Same as GPT2, scaled to 175B params
  - 96 layers
  - 96 attention heads
  - 128 d per head -> 12k d for the hidden state
- Causal attention, trained on LM task
  - Trained on Common Crawl (1 trillion words) + data filtering
  - Additional curated high-quality datasets

Compute used for training



GPT and contemporary LLMs (2)

- GPT3 paved the path towards contemporary LLMs

- Modern LLMs build upon GPT3 focusing on several key directions
  - Improved training practices (instruction tuning)
  - Increased model size
  - Increased data size and data quality
  - Multimodality
  - Prompt engineering