Applied Deep Learning for NLP

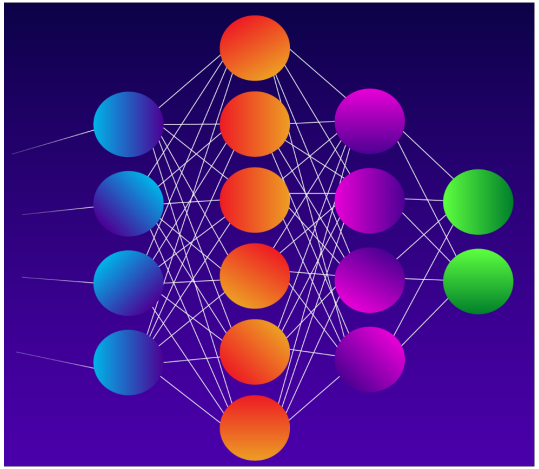
Week 8, 9 - GPT, BERT, and family

Juan Carlos Medina Serrano

Technische Universität München
Hochschule für Politik
Political Data Science

Munich, 10. December 2020

political
data
science

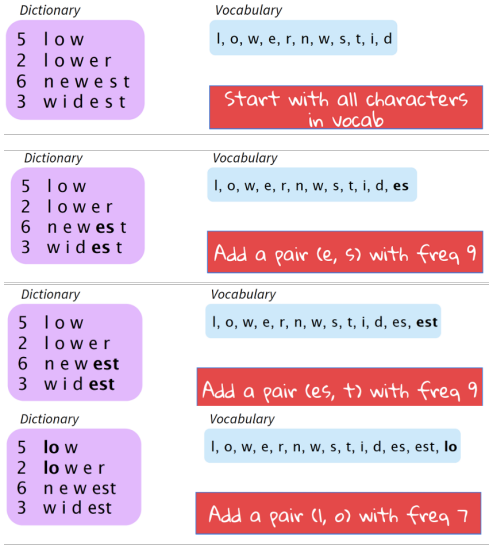


Subword Tokenizers

Subword tokenizers helped Mikolov enhance Word2Vec. They are still the best choice for Transformers.

Here the most important ones:

- **Byte Pair Encoding** (BPE) 1) Start with a vocabulary of characters
  2)Most frequent ngram pairs -> a new ngram
  3) Stop until you reach your defined maximum vocabulary size



Subword Tokenizers

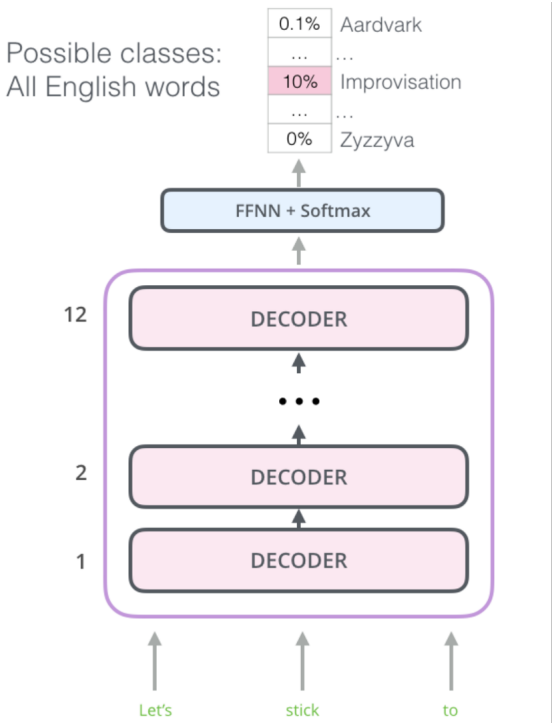
Subword tokenizers helped Mikolov enhance Word2Vec. They are still the best choice for Transformers.

Here the most important ones:

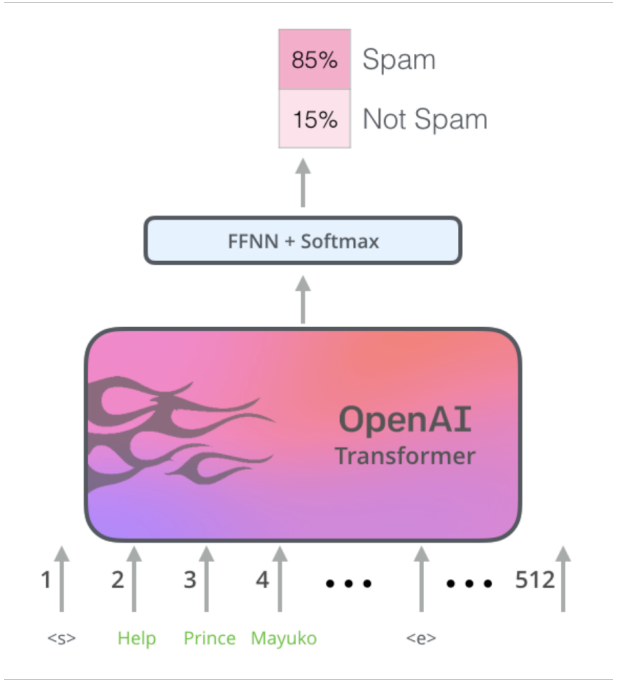
- **Byte Pair Encoding** (BPE)
- **WordPiece** Similar to BPE, but trains a language model and selects ngrams according to the likelihood of the data instead of frequency. (the n-gram that reduces perplexity)
- **Unigram Language Model** Based on a probabilistic language model. (BPE is deterministic). Starts the other way
  around, with a large vocabulary and subtokens, and then prunes the ones that contribute less to decrease the loss.
- **SentencePiece** Treats spaces as special token (_). No pretokenization needed for spaces. It then uses one of the three methods above (most commonly with the Unigram model)

GPT (Generative Pre-trained Transformer)

Forget about translations! Use only the Transformer Decoder to train a **Language Model**. (Using BPE)

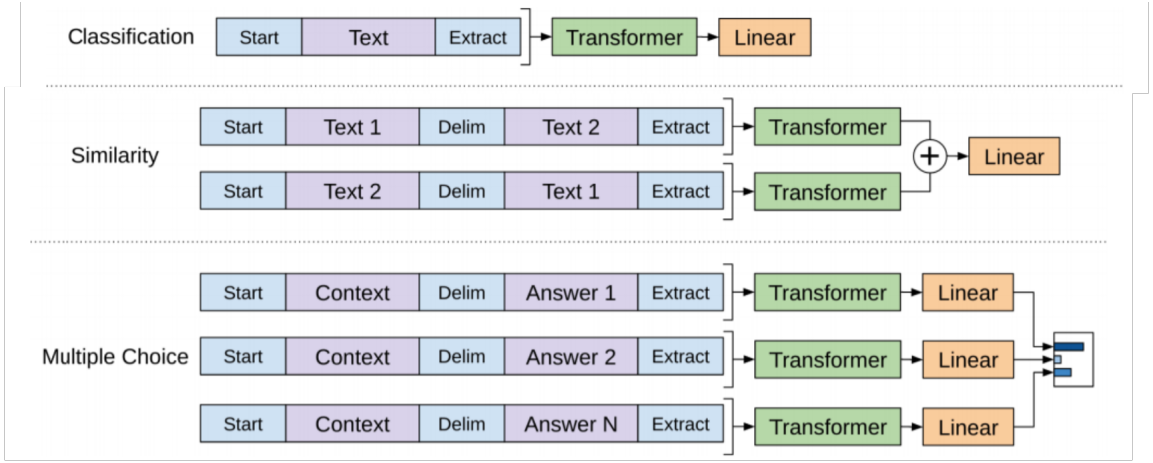


GPT Fine-tuning



Again awesome design from `http://jalammar.github.io/illustrated-bert/`

GPT Fine-tuning Tasks



BERT

Remember ELMO also trained a language model, and was bi-directional?
Can we add bi-directionality to the Transformer?

Enter: BERT: Bidirectional Encoder Representations from Transformers.

Use the **Encoder** instead of the Decoder!

Masked LM

We need to change the prediction task to allow bidirectional learning.

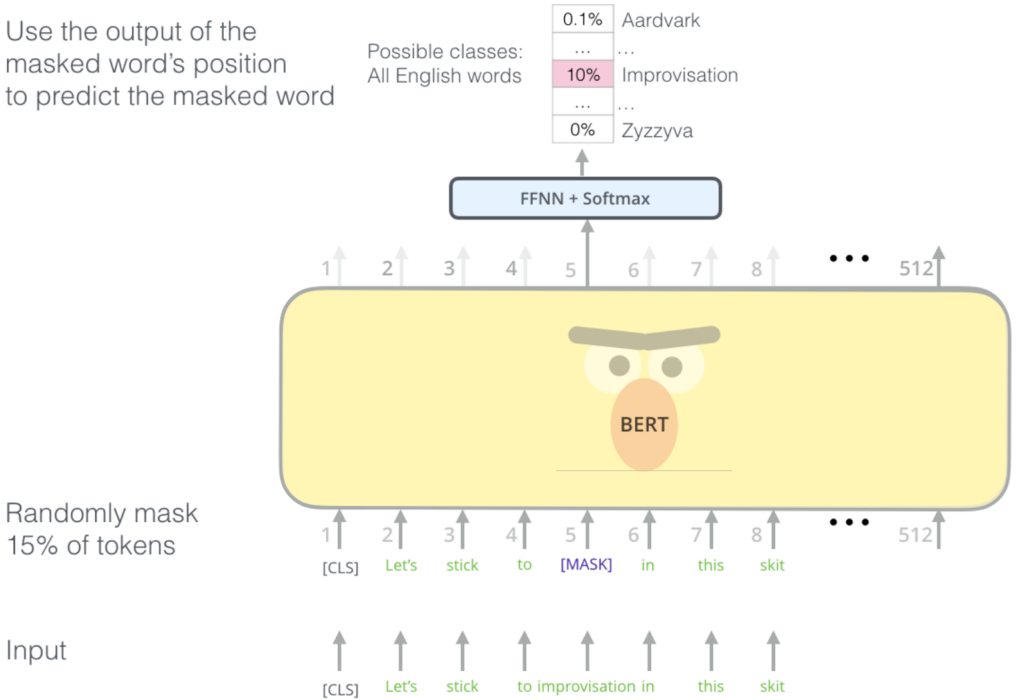
Solution: Mask out 15% of the input words, and predict them.

Using future words is not cheating!

In the implementation for the selected words to predict:

- 80% of the time, replace with mask token ([MASK])
- 10% of the time, replace with random word
- 10% of the time, replace with **the same** word.

BERT

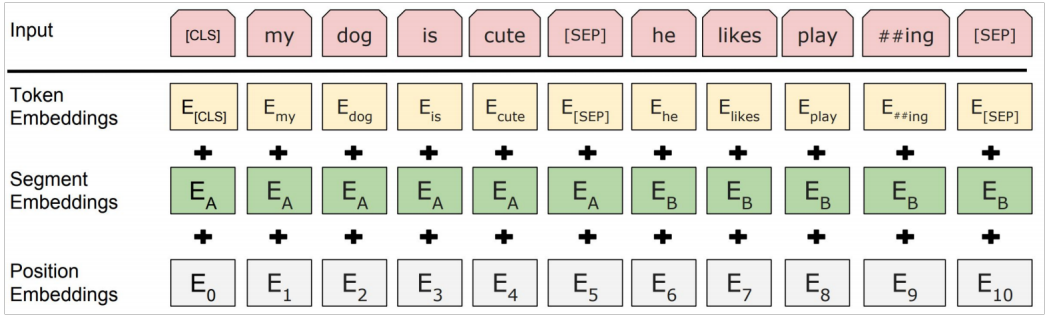

Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
... ...
10% Improvisation
... ...
0% Zyzzyva
FFNN + Softmax
1 2 3 4 5 6 7 8 ••• 512
BERT
Randomly mask 15% of tokens
1 2 3 4 5 6 7 8 ••• 512
[CLS] Let's stick to [MASK] in this skit
Input
[CLS] Let's stick to improvisation in this skit

BERT

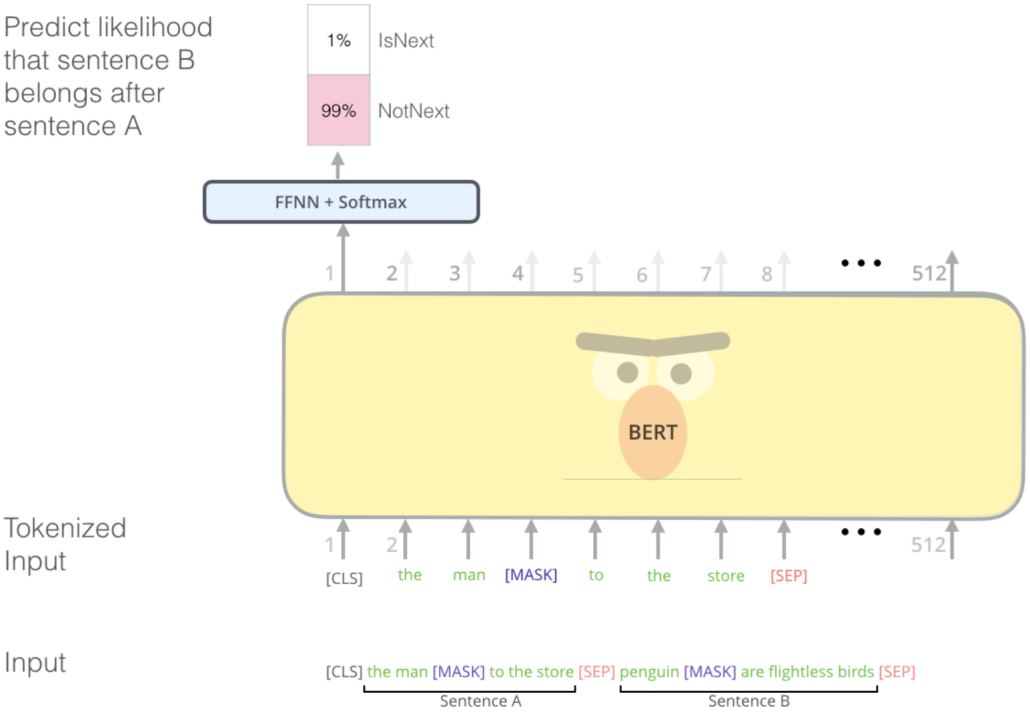
BERT uses a second task to train the model: **NEXT SENTENCE PREDICTION**. To learn relationships between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = `The man went to the store.`	**Sentence A** = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`	**Sentence B** = `Penguins are flightless.`
Label = `IsNextSentence`	**Label** = `NotNextSentence`

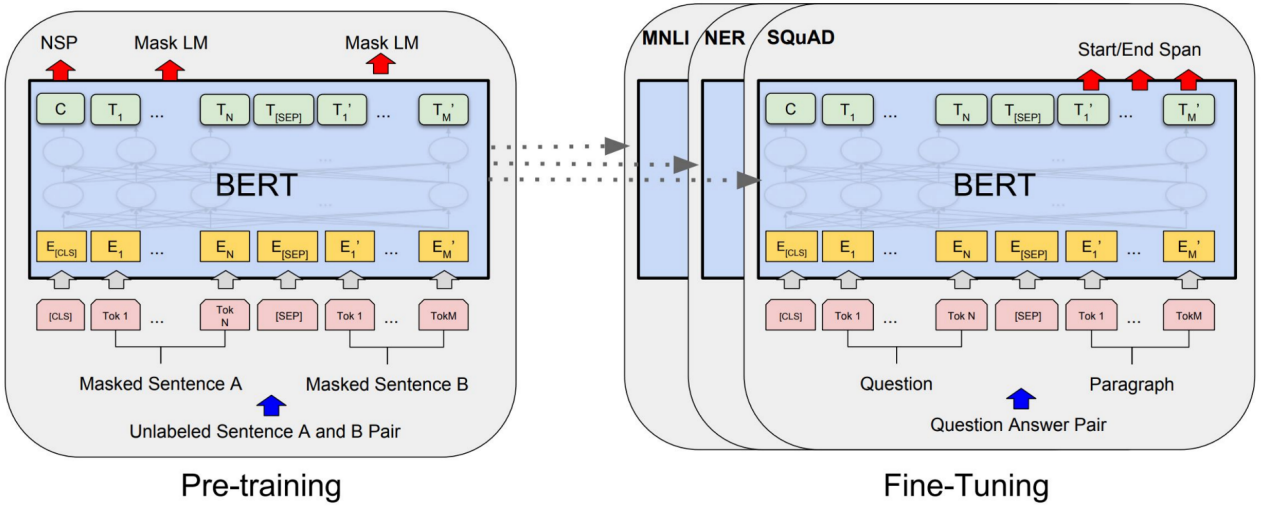
Input Representation consists of WordPiece Embeddings and a [SEP] between sentences:



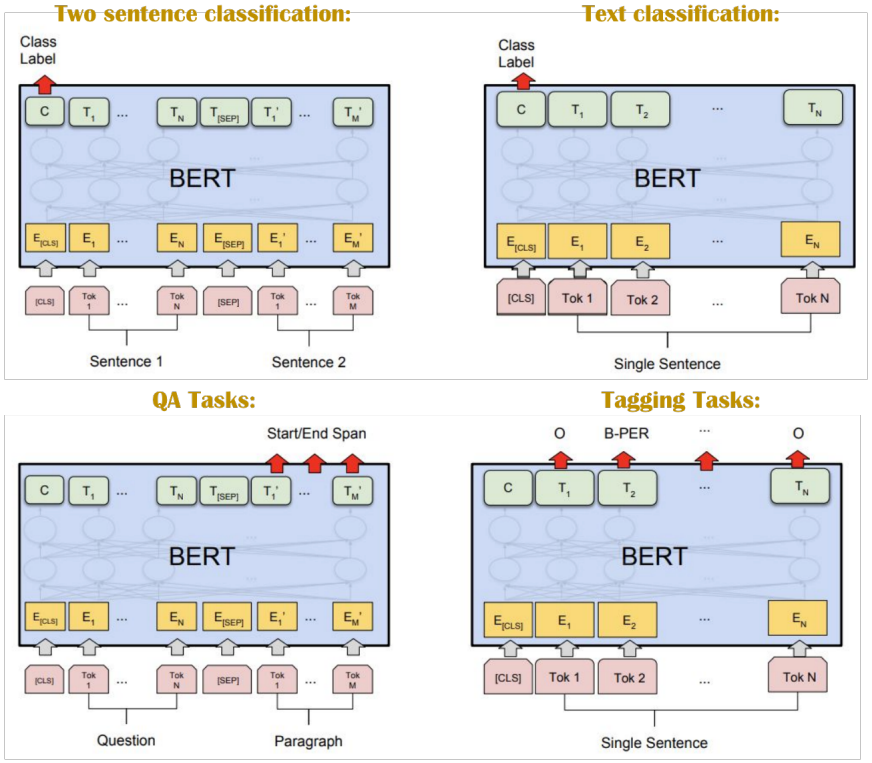
BERT


Predict likelihood that sentence B belongs after sentence A
1%
IsNext
99%
NotNext
FFNN + Softmax
1
2
3
4
5
6
7
8
512
BERT
Tokenized Input
[CLS]
the
man
[MASK]
to
the
store
[SEP]
Input
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B

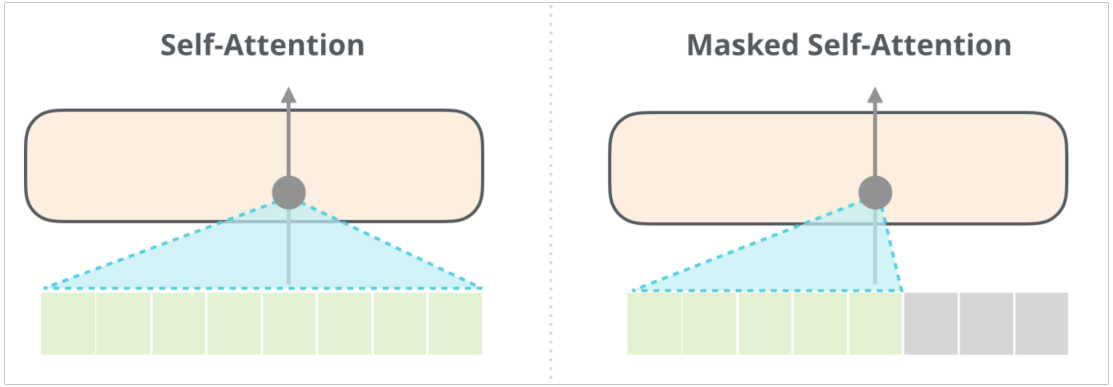
BERT Training



BERT Fine-Tuning



BERT vs GPT Attention

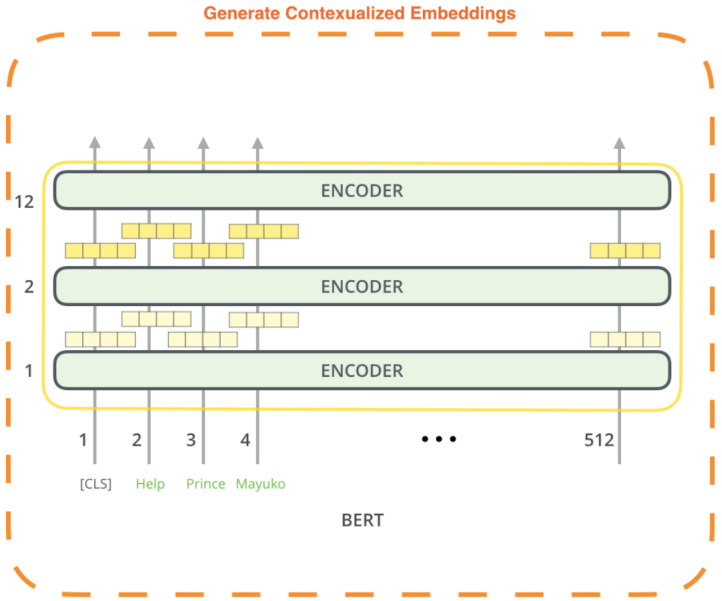


GPT is categorized as an **Autoregressive** (AR) Language Model

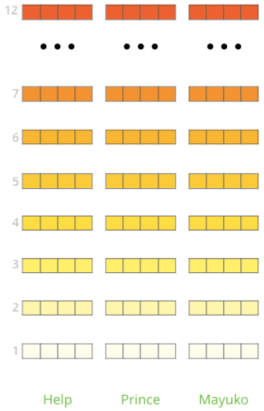
BERT is categorized as an **Autoencoder** (AE) Language Model

Feature Extraction

Apart from fine-tuning any Transformer-based model, you can also use them to generate contextual word embeddings, and use them in other non-DL tasks. (Remember Week 3)



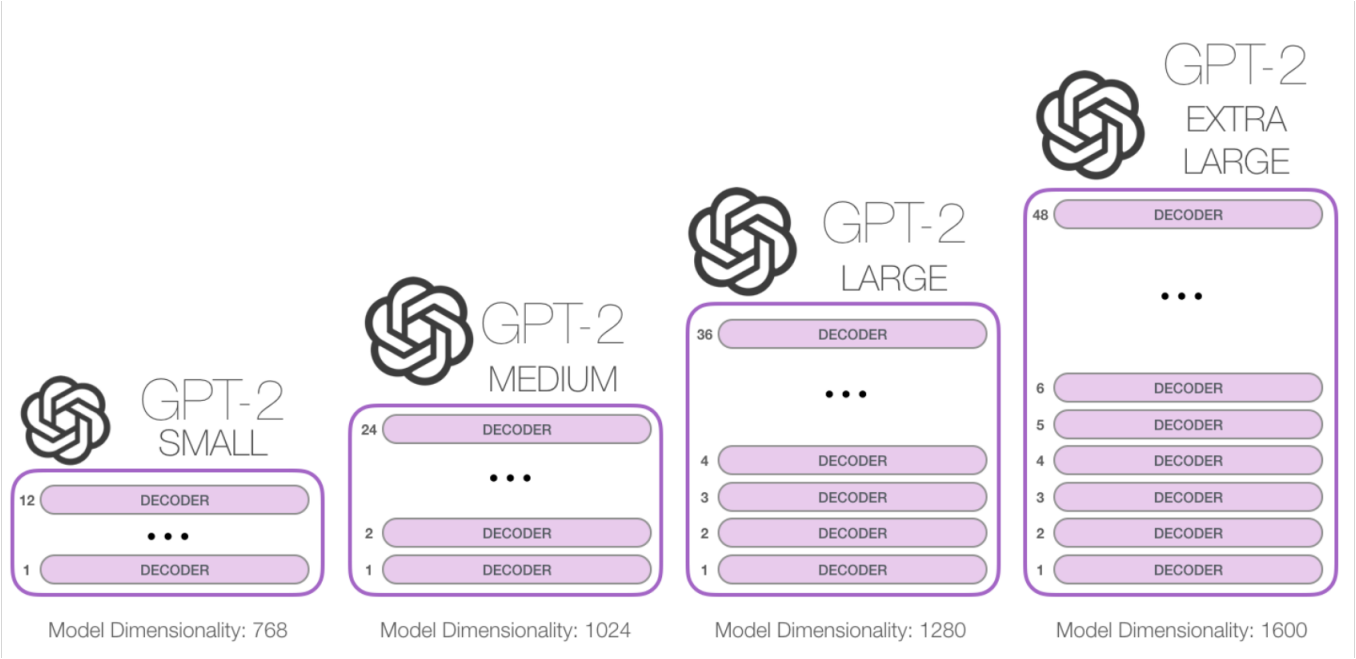
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

GPT-2

Make GPT (great again) larger! Increases the number of parameters



RoBERTa

Robustly Optimized BERT Pretraining Approach: 'BERT on steroids'

Take BERT and

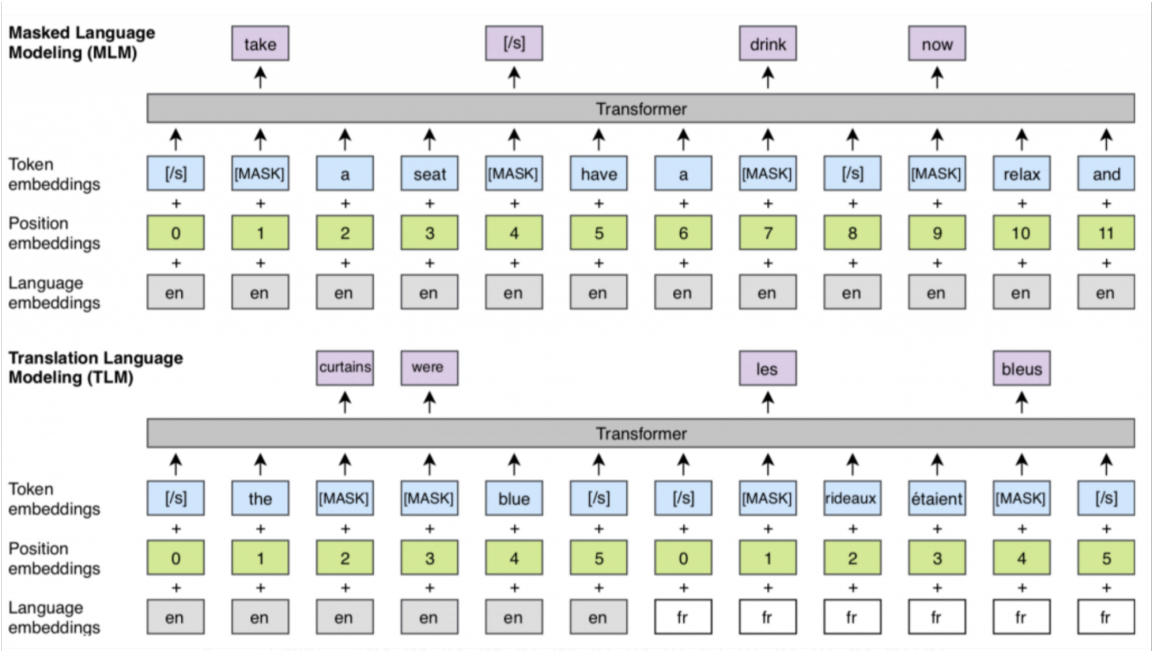
1) train it longer, with more data, with more parameters, with longer sequences!

2) Forgets the next sentence prediction task

3) Makes masking dynamic (Changes the words chosen as masks ever X number of epochs)

XLM

Remember that the Transformer was originally conceived for doing translations. Why not use BERT to also translate!

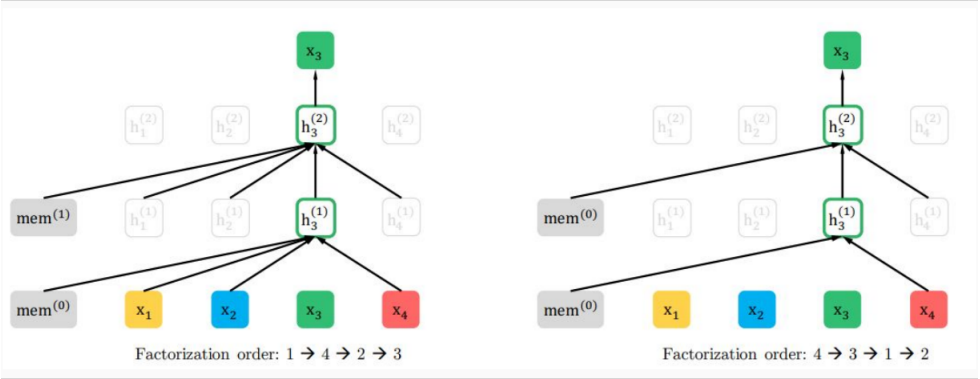


XLM is a **Multi-lingual** model!

XLNet

Take the power of BERT and make it an AR Language Model! How?

1. Use the Encoder from the **Transformer XL** instead of the Transformer.
   - Includes a recurrance mechanism with a recurrent memory in the self attention layer (Takes ideas from RNNs)
   - **Relative positional encoding** instead of absolute.
2. Uses **Permutation Language Modeling**: Randomly permutate the order for every training sentence. Only applies during the calculation of the language model probability distribution.



ALBERT

A Lite BERT. Idea: How to reduce the number of parameters in BERT to have enough compute power (and GPU memory) to scale the model

Changes to BERT:

1) **Factorized Embeddings:** Project the one-hot vectors to Embeddings of a smaller size than the Hidden layer size. In this way, the model parameters (hidden neurons) can increase without increasing the size of vocabulary embeddings.

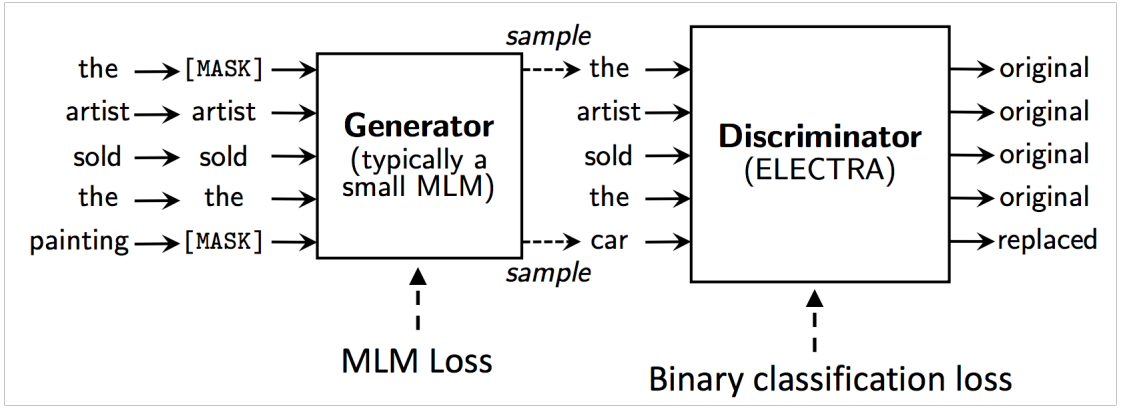
2) **Cross-layer parameter sharing:** Share parameters across all layers (FFN and MultiAttention layers)

3) Instead of next second prediction, it uses **sentence order prediction**: binary classification with two consecutive sentences as positive and the two with the order swapped as negative (Focus on text coherence, not topic prediction).

The large ALBERT model has about 18x fewer parameters compared to BERT-large. However, it has a larger structure and this means way more computational time to train. Also better performance!

ELECTRA

Trains 2 Transformer Encoders. First one is a Generator (Same as BERT) which predicts the masked words. The second one is a Discriminator with a binary classification to find if the token is original or replaced.
Task: **Replaced token detection**



Idea similar to training a GAN discriminator, BUT this is not adversarial task and there is NO backpropagation of loss from Discriminator to Generator! (Sampling in a non-continous vector space is not possible with GANs)

Note: Correct tokens predicted by the generator as marked as original

ELECTRA

Fine tuning only happens with the Discriminator (Opposite to the GAN scenario)

Advantage of ELECTRA: Less computing power needed to train it. (Four times less compute than RoBERTa and XLNet with similar performance). Outperforms these models with the same amount of compute.

Gains are particular strong for smaller models.

Sidenote: Replaced token detection is part of **Contrastive Learning** methods. Distinguish observed points from ficticious negative points. It is actually similar to CBOW (word2vec) with negative sampling! However, ELECTRA uses transformers! (And a generator instead of unigram token frequencies)

DistillBERT

Train a small BERT model with the power of a large trained model! How?

DISTILLATION: It is a technique of model compression. A full trained model (teacher) is used to train a smaller model (student). The student tries to mimic the output of the teacher. (Approximate the output distribution)

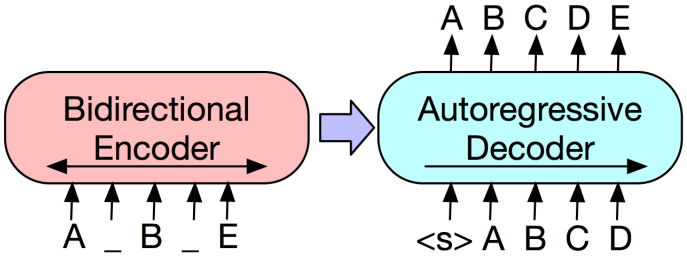
Three losses:

- Masked LM loss- Original loss from BERT
- Cosine embedding loss- The token embeddings should stay similar between teacher and student
- Distillation loss- Compares output distribution between teacher and student. The more similar, the higher the loss. (Similar to the Kullback-Leibler loss that compares two distributions (Think GANS, VAEs)

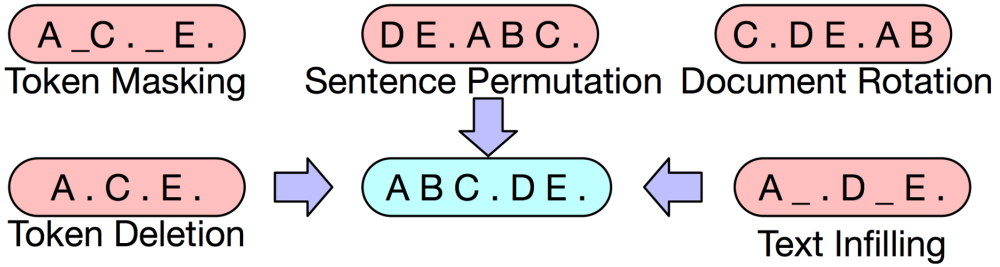
DistillBERT retains 97% performance from BERT but using only half the number of parameters (half number of layers)

BART

Encoder or Decoder Transformer? Let's put them together again! (Keeping BERT architecture on the Encoder side)

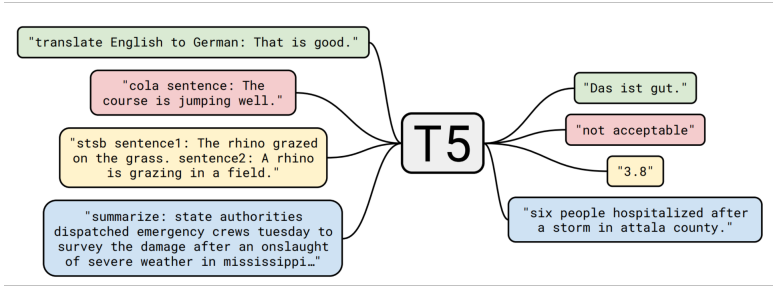


Task: Use a Seq2Seq model to reconstruct the original text from corrupted text using an arbitrary noise function:

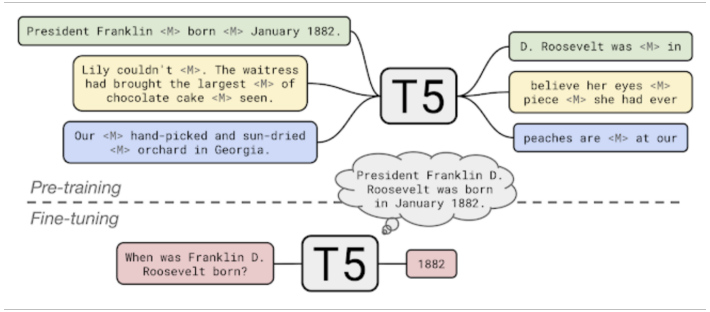


T5: Text-to-Text Transfer Transformer

As BART it uses the complete Transformer architecture with BERT-type Encoder. It goes from seq2seq to text-to-text by including the task in the sequence. Supports all types of NLP tasks!



Training task: **Sized fill-in-the-blank**: Learns to fill in dropped-out spans of text

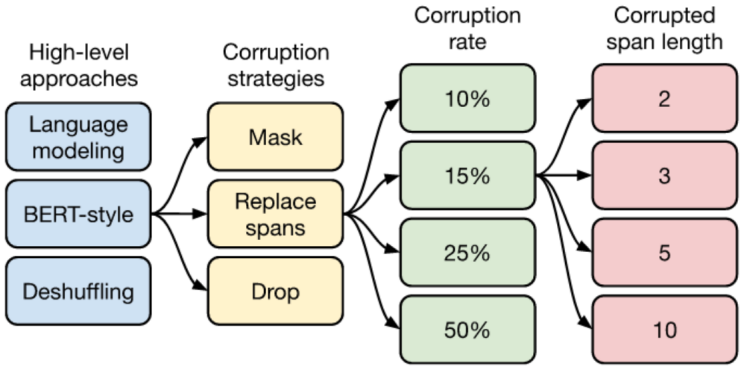


T5: Text-to-Text Transfer Transformer

The paper is a large-scale empirical survey to determine which transfer learning techniques work best and apply these insights at scale. It tries all possible configurations and models.

Finds that the Encoder-Decoder model is better than the Encoder or Decoder separately.

Also to decide which learning task was better to train the model, they tried different ideas:

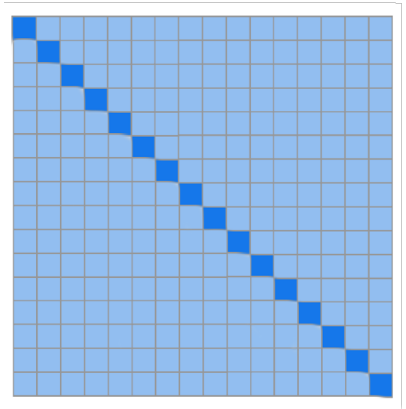


Sparse Attention

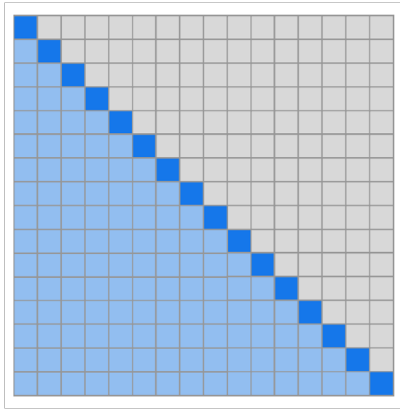
Self-attention requires time and memory that grows quadratically with the sequence length. Long sequences are hard to train. No long memory!

Think of who attends who as bipartite graph, and its adjancency matrix with 1 if attends and 0 if it doesn't.

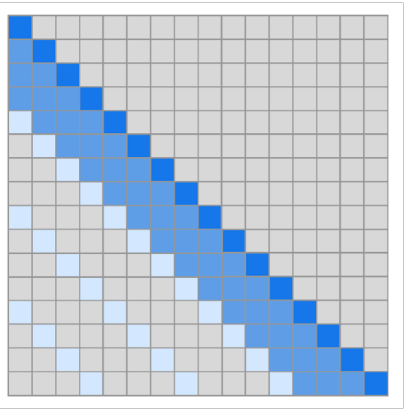
Sparse attention: Don't attend to all other words in sentence:



BERT



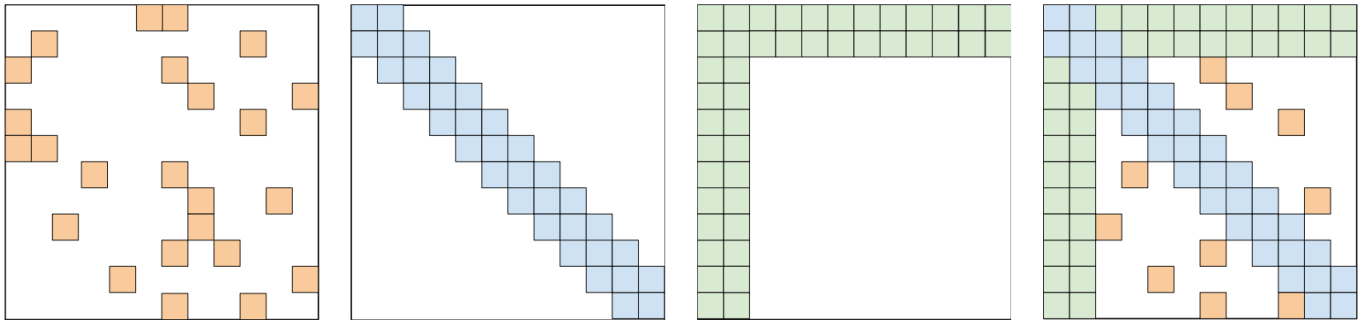
GPT



Sparse Transformer

BigBird

Sparse attention has to approximate the full attention well. BigBird uses graph theory to achieve this (graph sparsification problem). They reduce attention to linear time!



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Can handle sequences 8 times longer than previous architectures (512 tokens for BERT)

GPT-3

Still the same GPT architecture but LAAAAAAAAARGER. Otherwise the only major chance is that they also use sparse attention (which one? they don't say, but similar approach from their Sparse Transformers paper)

Parameter increase: BERT-Large (2018) has 355M parameters, GPT-2 reaches 1.5B, T5 further streches to 11B, GPT-3 finally gets to 175B

GPT-3 would require 355 years and $4,600,000 to train on a simple GPU instance

Memory: The 175 Billion parameters needs $175 \times 4 = 700$GB memory to store in FP32 (each parameter needs 4 Bytes). This is one order of magnitude larger than the maximum memory in a single GPU (48 GB of Quadro RTX 8000). To train the larger models without running out of memory, the OpenAI team uses a mixture of model parallelism within each matrix multiply and model parallelism across the layers of the network.

Language Models are Few-Shot Learners

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
```

PERFORMERS

Reformers -> Linformers -> Performers

Main idea: Make attention faster (linear)! It is a bottleneck for large Transformer models. (quadratic computation time and quadratic space to store the attention scores)

Didn't sparse attention solve this problem? Yes, but not completely (they have limitations such as requiring efficient sparse-matrix multiplication operations, stack more attention layers to compensate for sparse representations, and the softmax still takes long to compute).

Solution: Approximate the softmax! Instead of

$$softmax(Q * K^T) * V -> Q' * K'^T * V$$

How? This is a breakthrough, you need lots of math and kernel methods! Remember kernels allow us to do inner products in high dimensional space, with simple dot products. Here the softmax is the high dimensional space and Q' and K'^T the original K and Q transformed with a kernel

Stores the implicit attention matrix with linear, rather than quadratic, memory complexity. No need for sparcity to have similar performance as normal transformers with longer sequences.