Encoders:
The Art of Packing Text into Vectors

Marek Rei

Imperial College
London
Marek Rei

Lecturer at Imperial College London

Visiting researcher at the University of Cambridge

Scientific advisor for Transformative AI

Previously:
○ Senior Research Associate at Cambridge

○ Researcher at Swiftkey (now Microsoft)
Talk outline

01 Representing words as vectors
02 Pre-trained transformer encoders
03 Putting pre-trained models to work
04 The keys to good encoders
05 Some interesting extensions to these models
Representing words as vectors
Motivation

Language is so highly compositional, we cannot rely on seeing specific word combinations during training.

I bought a beautiful rose
I acquired a pretty flower

We want to represent textual information as feature vectors, such that texts with similar meaning have similar vectors*.

These feature vectors capture the meaning of text for downstream ML models.

* A vector is just a sequence of numbers/features. It can also be thought of as a coordinate.
Word embeddings

First, how can we represent individual words?

**Idea:** Let’s allocate a number of parameters for each word and allow a neural network to automatically learn what the useful values should be.

Often referred to as “**word embeddings**”, as we are embedding the words into a real-valued vector space.
Problem: Often our training datasets are small, not containing enough information about all the words.

Solution: Learn these vectors on some other task with many more examples, then transfer them to our main task.

We can use a task that doesn’t require any manually annotated labels.
Continuous Bag-of-Words (CBOW) model

Assumption:
Words which are similar in meaning occur in similar contexts.

He is reading a magazine
I was reading a newspaper

Learning to predict the current word based on the surrounding words.

Can use any plain text for training.

Mikolov et. al. (2013)
Word embeddings
The vectors are usually not 2 or 3-dimensional. More often 100-1000 dimensions.

For example, a real vector for “bear”:

-0.089383  -0.375981  -0.337130  0.025117  -0.232542  -0.224786  0.148717  -0.154768  -0.260046  -0.156737  
-0.085468  0.180366  -0.076509  0.173228  0.231817  0.314453  -0.253200  0.170015  -0.111660  0.377551  -0.025207  
-0.097520  -0.020041  0.117727  0.105745  -0.352382  0.010241  0.114237  -0.315126  0.196771  -0.116824  -0.091064  
-0.291241  -0.098721  0.297539  0.213323  -0.158814  -0.157823  0.152232  0.259710  0.335267  0.195840  -0.118898  
0.169420  -0.201631  0.157561  0.351295  0.033166  0.003641  -0.046121  0.084251  0.021727  -0.065358  -0.083110  
-0.265997  0.027450  0.372135  0.040659  0.202577  -0.109373  0.183473  -0.380250  0.048979  0.071580  0.152277  
0.298003  0.017217  0.072242  0.541714  -0.110148  0.266429  0.270824  0.046859  0.150756  -0.137924  -0.099963  
-0.097112  -0.110336  -0.018136  -0.032682  0.182723  0.260882  -0.146807  0.502611  0.034849  -0.092219  
-0.103714  -0.034353  0.112178  0.065348  0.161681  0.006538  0.364870  0.153239  -0.366863  -0.149125  0.413624  
-0.229378  -0.396910  -0.023116
The meaning of a word can depend on its context

I deposited some money in the bank

I was camping on the east bank of the river

Having a single vector for every word doesn’t cut it.

Need to take the context into account when constructing word representations.
Contextual word embeddings

We can train a recurrent neural network to predict the next word in the sequence, based on the previous words in the context.

Internally it will learn to encode any given context into a vector.


Represents "brown" while taking "The quick brown" into account
ELMo: Embeddings from Language Models

Take a large corpus of plain text and train two language models:

1. One recurrent language model going forward, left-to-right.
   \[ p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1}) \]

2. A second recurrent language model going backward, right-to-left.
   \[ p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N) \]

ELMo: Embeddings from Language Models

When we need a vector for a word, combine the representations from both directions.
ELMo: Embeddings from Language Models

https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/
ELMo: Embeddings from Language Models

ELMo can be integrated into almost all neural NLP tasks with simple concatenation to the embeddings layer.
Pre-trained transformer encoders
BERT

Bidirectional Encoder Representations from Transformers (Devlin et al., 2019).

Takes in a sequence of words, gives as output a vector for each word.

Builds on previous work (ELMo and others), combines some good ideas and scales it up further.

https://medium.com/@gabell/encoder-decoder-models-and-transformers-5c1500c22c22
What’s inside the transformer encoder

http://jalammar.github.io/illustrated-transformer/
Self-attention

The new representation of each word is calculated based on all the other words.
Self-attention

Unlike RNNs and LSTMs, every word is just one step away from every other word.
Multi-head self-attention

Each head learns to focus on different type of information.
What’s inside the encoder
Input embeddings

Transformers have no concept of sequence order.

Positional embeddings allow it to capture the order of the input words.

Devlin et al. (2018)
Masked Language Modeling

We want to train this model on huge amounts of plain text, so it learns general-purpose language understanding.

Use masked language modeling as the training task - it requires no labeled data!

Hide k% of the input words behind a mask, train the model to predict them.
Masked Language Modeling

Too little masking: too expensive to train
Too much masking: not enough context to make predictions

**Actual strategy used:**
Pick 15% of the input words as training targets
- 80% of those are replaced with the [MASK] token
  
  went to the store -> went to the [MASK]

- 10% are replaced with a random word
  
  went to the store -> went to the running

- 10% are the left as the original word
  
  went to the store -> went to the store

If we masked all the chosen words, the model wouldn’t necessarily learn to construct good representations for non-masked words.
Next sentence prediction

BERT used a second training objective.

Given two sentences, predict whether they appeared in the original order in the source text.

Sentence A = The man went to the store.  
Sentence B = He bought a gallon of milk.  
Label = IsNextSentence

Sentence A = The man went to the store.  
Sentence B = Penguins are flightless.  
Label = NotNextSentence

Later work found that this did not provide much additional benefit, so it has not been used in newer versions.
Putting pre-trained models to work
How do we use these models?

BERT-like models give us a representation vector for every input token. Just have to chop off the masked LM head, it is no longer needed.

We can use those vectors to represent individual tokens or full sentences.

Option 1:
- Freeze BERT, use it to calculate informative representation vectors.
- Train another ML model that uses these vectors as input.

Option 2 (more common these days):
- Put a minimal neural architecture on top of BERT (e.g. a single output layer)
- Train the whole thing end-to-end (called fine-tuning).
Sentence classification

We can add a special token in the input that represents the whole sentence

Devlin et al. (2018)
Token labeling

Putting a classification layer on top of token vectors

Devlin et al. (2018)
Sentence pair classification

Giving multiple sentences as input
Question answering

Labeling the tokens in the candidate answer span

Devlin et al. (2018)
Text generation

The **encoder** learns to process the source sentence and produce an informative vector representation.

The **decoder** learns to generate a sentence in a different language based on that vector.

Bahdanau et al. (2014), figure by Stephen Merity.
Performance improvements

Convincingly helps on pretty much every natural language task

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

Devlin et al. (2018)
Many spin-off models since then

**BERT**
- Trained with masked language modelling and next sentence prediction
- First large pre-trained transformer model
- Outperformed everything before it

**RoBERTa**
- Got rid of next sentence prediction, optimized hyperparameters
- Trained on much more data than BERT
- Outperformed BERT

**DeBERTa**
- Focused on improvements to positional encodings
- Trained on much more data than RoBERTa
- Outperformed RoBERTa

Many spin-off models since then
A model for every occasion

**ALBERTA** and **DistilBERT**

Smaller and faster versions, while retaining performance (or even outperforming BERT)

**Big Bird** and **LongFormer**

For very long textual input (longer than BERT can handle)

**ClinicalBert, MedBert, PubMedBert, BEHRT**

Trained specifically on medical data for medical applications
Something for every language

German BERT
  For German applications

CamemBERT
  For French applications

EstBERT
  For Estonian applications

M-BERT
  Trained jointly on 104 languages
Multimodal models
Combining visual and textual information into the same transformer. LXMERT, VisualBERT, ImageBERT, ViLBERT

An old man swimming in a pool.

Qi et al. (2020)
<table>
<thead>
<tr>
<th>Task</th>
<th>Models</th>
<th>Libraries</th>
<th>Datasets</th>
<th>Languages</th>
<th>Licenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Classification</td>
<td>42,887</td>
<td>PyTorch, TensorFlow</td>
<td>wikipedia, common_voice, squad, glue</td>
<td>English, Spanish, French, German, Chinese, Russian, Arabic, Japanese</td>
<td>apache-2.0, mit, afi-3.0</td>
</tr>
<tr>
<td>Translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image Segmentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic Speech Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Token Classification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sentence Similarity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Classification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Question Answering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summarization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero-Shot Classification</td>
<td>18 Tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Models**

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Author(s)</th>
<th>Updated Date</th>
<th>Downloads</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>microsoft/deberta-base</td>
<td></td>
<td>Jan 13</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>bert-base-uncased</td>
<td></td>
<td>Jun 8</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>Jean-Baptiste/camembert-ner</td>
<td></td>
<td>Apr 4</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>gpt2</td>
<td></td>
<td>May 10</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>distilbert-base-uncased</td>
<td></td>
<td>Mar 31</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>SpanBERT/spanbert-large-cased</td>
<td></td>
<td>May 19</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>xlm-roberta-base</td>
<td></td>
<td>Jun 6</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>distilroberta-base</td>
<td></td>
<td>Jun 6</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>distillm2</td>
<td></td>
<td>Jun 6</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>distilbert-base</td>
<td></td>
<td>Jun 6</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>roberta-base</td>
<td></td>
<td>Jun 6</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>distilbert-base-uncased-finetuned-sst-2-english</td>
<td></td>
<td>Jul 6</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>hfl/chinese-macbert-base</td>
<td></td>
<td>Aug 19</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>tals/albert-xlarge-vitamin-c-mnli</td>
<td></td>
<td>Aug 8</td>
<td>78</td>
<td></td>
</tr>
<tr>
<td>deepset/roberta-base-squad2</td>
<td></td>
<td>Jul 26</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>bert-base-chinese</td>
<td></td>
<td>Jul 26</td>
<td>111</td>
<td></td>
</tr>
</tbody>
</table>
# Text classification example with BERT
# Created by Marek Rei
# Based on https://colab.research.google.com/github/huggingface/notebooks/blob/master/course/chapter3/section4.ipynb
# Training the model for binary sentiment detection, using the SST2 dataset,

# Some settings
# Which pre-trained model to use.
# See https://huggingface.co/models for options.
checkpoint = "bert-base-uncased"

# How much training data to use.
# 1.0 uses the whole training set but it can take a bit of time to train.
train_data_sample_ratio = 0.1

# Example sentence to use
# We print out predictions for this sentence before and after training
example_sentence = "this was by far the best movie of the year"

[197] # Install the necessary libraries
!pip install datasets evaluate transformers[sentencepiece]

[198] # Import the libraries
import torch
import evaluate
Investigating and extending BERT
Generalisation in noisy settings

Adding 0-50% label noise in the training data for Named Entity Recognition. Testing on clean data.

Darker colours correspond to higher levels of noise.

**Finding #1:** Training gets separated into three distinct stages
Generalisation in noisy settings

Adding 0-50% label noise in the training data, testing on clean data

Darker colours correspond to higher levels of noise.

Finding #2: The model is quite robust to noise.
Adding 30% noise to the CoNLL03 dataset causes only a 0.9% decrease of validation performance in the second phase.
Generalisation in noisy settings

Adding 0-50% label noise in the training data, testing on clean data

Darker colours correspond to higher levels of noise.

Finding #3: Early stopping at a specific point is less important than before, as long as it’s somewhere in phase 2.
Detecting noisy examples

Given that noisy and clean examples are learned in different stages, can we use that to automatically detect the noisy datapoints?

We collect the losses for each training example after a short fine-tuning process (4 epochs).

We then assume that the losses form two clusters (noisy vs clean) and find the optimal threshold $T$ to separate them:

$$\arg \min_T \sum_{x < T} \|x - \mu_c\|^2 + \sum_{x \geq T} \|x - \mu_n\|^2$$

where elements denoted as $x$ are the losses extracted from the training set, $\mu_c$ is the mean of all $x < T$, and $\mu_n$ is the mean of all $x \geq T$. 
Detecting noisy examples

Visualising loss values on the CoNLL03 dataset.
Detecting noisy examples

Noise detection results on the CoNLL03 dataset, with different levels of inserted noise.

<table>
<thead>
<tr>
<th>Noise</th>
<th>Precision</th>
<th>Recall</th>
<th>F₁ score</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>92.18%</td>
<td>95.90%</td>
<td>94.00%</td>
</tr>
<tr>
<td>20%</td>
<td>96.19%</td>
<td>96.33%</td>
<td>96.26%</td>
</tr>
<tr>
<td>30%</td>
<td>98.02%</td>
<td>96.35%</td>
<td>97.17%</td>
</tr>
<tr>
<td>40%</td>
<td>98.27%</td>
<td>96.95%</td>
<td>97.60%</td>
</tr>
<tr>
<td>50%</td>
<td>98.64%</td>
<td>97.27%</td>
<td>97.94%</td>
</tr>
</tbody>
</table>

Finding #4: The loss values of datapoints after a short fine-tuning process can be used to detect examples with noisy or incorrect labels.

“Memorisation versus Generalisation in Pre-trained Language Models”
Tanzer, Ruder, Rei. ACL 2022
Zero-shot token labeling

Learning to perform token labeling only based on sentence-level annotation

It was so long time to wait in the theatre.
I look forward to receiving reply to my enquiry.
This is a great opportunity to learn more about whales.
Therefore, houses will be built on high supports.

I like to playing the guitar and sing very louder.
Zero-shot token labeling

"Zero-shot Sequence Labeling for Transformer-based Sentence Classifiers"
Bujel, Yannakoudakis, Rei. RepL4NLP 2021
Soft attention weights

Based on softmax:

\[
    a_i = \frac{\exp(\tilde{e}_i)}{\sum_{k=1}^{N} \exp(\tilde{e}_k)}
\]

Based on sigmoid + normalisation:

\[
    \tilde{a}_i = \frac{1}{1 + \exp(-\tilde{e}_i)} \quad a_i = \frac{\tilde{a}_i}{\sum_{k=1}^{N} \tilde{a}_k}
\]
We can constrain the attention values based on the sentence-level label.

1. Only some, but not all, tokens in the sentence can have a positive label.

   \[ L_2 = \sum_j (\min_j (\tilde{a}_i) - 0)^2 \]

2. There are positive tokens in a sentence only if the overall sentence is positive.

   \[ L_3 = \sum_j (\max_j (\tilde{a}_i) - \tilde{y}(j))^2 \]
Zero-shot token labeling

Improves performance over other interpretability approaches

<table>
<thead>
<tr>
<th></th>
<th>FCE</th>
<th></th>
<th></th>
<th>BEA 2019</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sent $F_1$</td>
<td>$F_1$</td>
<td>MAP</td>
<td>Sent $F_1$</td>
<td>$F_1$</td>
<td>MAP</td>
</tr>
<tr>
<td>Random baseline</td>
<td>-</td>
<td>23.19</td>
<td>33.95</td>
<td>-</td>
<td>16.73</td>
<td>27.01</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>84.51</td>
<td>-</td>
<td>-</td>
<td>83.66</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Rei and Søgaard (2018)</td>
<td>84.75</td>
<td>28.73</td>
<td>48.56</td>
<td>81.27</td>
<td>18.53</td>
<td>31.69</td>
</tr>
<tr>
<td>LIME</td>
<td>84.51</td>
<td>24.60</td>
<td>37.90</td>
<td>83.66</td>
<td>2.09</td>
<td>31.41</td>
</tr>
<tr>
<td>Attention heads</td>
<td>84.51</td>
<td>24.34</td>
<td>48.04</td>
<td>83.66</td>
<td>19.69</td>
<td>40.55</td>
</tr>
<tr>
<td>Soft attention</td>
<td>85.62</td>
<td>32.16</td>
<td>48.90</td>
<td>83.41</td>
<td>22.92</td>
<td>35.79</td>
</tr>
<tr>
<td>Weighted soft attention</td>
<td>85.62</td>
<td>33.31</td>
<td>53.91</td>
<td>83.68</td>
<td>24.35</td>
<td>41.07</td>
</tr>
</tbody>
</table>
Zero-shot token labeling

Visualising where the model is focusing for different tasks

“Zero-shot Sequence Labeling: Transferring Knowledge from Sentences to Tokens”
Rei and Søgaard. NAACL 2018.
Possible applications

01 Token labeling without data

02 Data exploration and feature analysis

03 Model visualisation and interpretation
Supervised Attention

Instead of having the model figure out where it should focus, we can supervise the attention function and teach it to focus on the same areas as humans.

Attention from [CLS] token (for head h):

\[
\begin{align*}
  d_1 &= 0 \\
  d_2 &= 0 \\
  d_3 &= 0.25 \\
  d_4 &= 0.25 \\
  d_5 &= 0 \\
  d_6 &= 0 \\
  d_7 &= 0.25 \\
  d_8 &= 0 \\
  d_9 &= 0.25 \\
  d_{10} &= 0
\end{align*}
\]

```
[CLS] \rightarrow \text{[CLS]}
\text{The} \rightarrow \text{The}
\text{dog} \rightarrow \text{dog}
\text{swims} \rightarrow \text{swims}
\text{The} \rightarrow \text{The}
\text{dog} \rightarrow \text{dog}
\text{is} \rightarrow \text{is}
\text{sleeting} \rightarrow \text{sleeting}
\text{[SEP]} \rightarrow \text{[SEP]}
```

```
e-SNLI explanation:
A dog cannot be sleeping while he swims
```

\[
\text{Loss}_{Total} = \text{Loss}_{NLI} + \frac{\lambda}{H} \sum_{h=1}^{H} \sum_{i=1}^{n} (a_{h_i} - d_i)^2
\]
## Supervised Attention

<table>
<thead>
<tr>
<th></th>
<th>Dev</th>
<th>Test</th>
<th>Hard</th>
<th>MNLI mi</th>
<th>MNLI ma</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT baseline</td>
<td>90.05</td>
<td>89.77</td>
<td>79.36</td>
<td>72.52</td>
<td>72.28</td>
</tr>
<tr>
<td>Ours (extra layer)</td>
<td>90.40</td>
<td>90.09</td>
<td>79.96</td>
<td>73.03</td>
<td>73.10</td>
</tr>
<tr>
<td>Improvement</td>
<td>+0.35†</td>
<td>+0.32‡</td>
<td>+0.60‡</td>
<td>+0.51†</td>
<td>+0.82‡</td>
</tr>
<tr>
<td>Ours (existing attention)</td>
<td>90.45</td>
<td>90.17</td>
<td>80.15</td>
<td>73.36</td>
<td>73.19</td>
</tr>
<tr>
<td>Improvement</td>
<td>+0.40†</td>
<td>+0.40‡</td>
<td>+0.79‡</td>
<td>+0.84‡</td>
<td>+0.91‡</td>
</tr>
</tbody>
</table>

### Performance when different heads are supervised

![Graph showing performance improvements](image-url)
Happy to take questions!
The keys to good encoders
Transformer uprising

Pre-trained transformer models have been a revolution in NLP.

After only a couple of years it is difficult to find any model that doesn’t use one of the pre-trained transformer models.

This is thanks to a few properties that can be applied to other ML tasks as well, beyond language.
1. Transfer learning (model pre-training)

Pre-training the model gives us better performance even with fewer downstream training examples.
2. Very large models

Training bigger models is giving better performance, although diminishing returns.
2. Very large models

Still growing. GPT-4 is expected to have 100 trillion parameters (same as the number of synapses in a human brain).
3. Loads of data

Huge amounts of training data.

<table>
<thead>
<tr>
<th>Model name</th>
<th>Amount of training data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELMo</td>
<td>800M tokens</td>
</tr>
<tr>
<td>BERT</td>
<td>3.3B words, 16GB</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>160 GB, ~33B words</td>
</tr>
<tr>
<td>DeBERTa</td>
<td>78GB</td>
</tr>
<tr>
<td>T5</td>
<td>750GB</td>
</tr>
<tr>
<td>GPT-3</td>
<td>45TB</td>
</tr>
</tbody>
</table>

Only plausible using unsupervised learning objectives, with unlabeled data.
4. Fast computation

In order to process huge amounts of data, the models need to be fast.

Transformers are not particularly fast.
But they are fast for their size.

Vectors for the whole sentence can be calculated in parallel. Particularly good for running on GPUs!

In contrast, RNNs and LSTMs would processing each word in sequence.

https://pytorch.org/tutorials/beginner/transformer_tutorial.html
5. A difficult learning task

The prediction of missing words is a very difficult task:

- Tens of thousands of possible options to choose from. 
  ~0.002% chance of guessing the correct word by accident.

- The model needs to take all types of language information into account

- The task is so difficult that even humans would have trouble

The model can’t just memorise the correct answers.

It keeps learning more and more.
Being applied on **images**

Masking parts of an image.

AlphaFold is an AI system that predicts a protein's 3D structure from its amino acid sequence. Essentially trains a large unsupervised protein language model which takes as input a set of MSA sequences.