Notes on Language Models, Attention and Transformers

Kyle Richardson[1]

Allen Institute for AI (AI2)[1]

August 2022

Goals for this lecture

1. What are *language models*? Classical approaches, limitations
2. What are *contextual models*? Differences with classical approaches
3. What is *attention* and how do *transformers* work?
4. How are these models trained and used in practice? Large-scale *pre-training* and *fine-tuning*.

Language modeling basics

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

Basic Language Modeling

- **objective**: for text $w_1, w_2, \ldots, w_m$, give $p(X_1 = w_1, X_2 = w_2, \ldots, X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, \ldots, w_{j-1}}_{\text{what came before}})$$

$$\begin{aligned} p(t) = & \, p(\text{Norway} \mid \overbrace{\texttt{<BOS>}}^{\text{start symbol}}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\ & p(\text{the} \mid \text{Norway is to}) \times p(\text{west} \mid \text{Norway is to the}) \\ & p(\text{of} \mid \text{Norway is to the west}) \times p(\text{Sweden} \mid \text{Norway is to the west of}) \end{aligned}$$

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

$$\begin{aligned} p(t) = & p(\text{Norway} \mid \overbrace{\texttt{<BOS>}}^{\text{start symbol}}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\ & p(\text{the} \mid \text{Norway is to}) \times p(\text{west} \mid \text{Norway is to the}) \\ & p(\text{of} \mid \text{Norway is to the west}) \times p(\text{Sweden} \mid \text{Norway is to the west of}) \end{aligned}$$

Estimation: learning these probabilities from example text.

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

$$p(t) = p(\text{Norway} \mid \overbrace{\texttt{<BOS>}}^{\text{start symbol}}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times$$
$$p(\text{the} \mid \text{Norway is to}) \times p(\text{west} \mid \text{Norway is to the})$$
$$p(\text{of} \mid \text{Norway is to the west}) \times p(\text{Sweden} \mid \text{Norway is to the west of})$$

Is this a hard problem? Involves <u>commonsense</u> knowledge,
$p(\text{Sweden} \mid \text{Norway is.. west of}) > p(\text{Iceland} \mid \text{Norway is..west of})$

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

$$\begin{aligned} p(t) =\, & p(\text{Norway} \mid \overbrace{\texttt{<BOS>}}^{\text{start symbol}}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\ & p(\text{the} \mid \text{Norway is to}) \times p(\text{west} \mid \text{Norway is to the}) \\ & p(\text{of} \mid \text{Norway is to the west}) \times p(\text{Sweden} \mid \text{Norway is to the west of}) \end{aligned}$$

Is this a hard problem? Involves deep linguistic knowledge,

$p(\text{him} \mid \text{John asked Mary to wash}) > p(\text{himself} \mid \text{John ...Mary to wash})$

Basic Language Modeling

- **objective**: for text $w_1, w_2, ..., w_m$, give $p(X_1 = w_1, X_2 = w_2, ..., X_m = w_m)$

  e.g., $t =$Norway is to the west of Sweden

  How to compute? The chain rule:

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

$$\begin{aligned} p(t) = &\, p(\text{Norway} \mid \overbrace{\texttt{<BOS>}}^{\text{start symbol}}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\ &\, p(\text{the} \mid \text{Norway is to}) \times p(\text{west} \mid \text{Norway is to the}) \\ &\, p(\text{of} \mid \text{Norway is to the west}) \times p(\text{Sweden} \mid \text{Norway is to the west of}) \end{aligned}$$

Generation: Can be used to generate text (*decoding*).

Classical Language Modeling

How to estimate? Count-based models, collect counts from large collection of texts (maximum likelihood estimation), *discrete objects*

$$p(w_j \mid w_1, \ldots, w_{j-1}) \propto \underbrace{\text{count}(w_1, \ldots, w_{j-1}, w_j)}_{\text{how often have I seen this \underline{discrete} event in data?}}$$

Classical Language Modeling

How to estimate? Count-based models, collect counts from large collection of texts (maximum likelihood estimation), *discrete objects*

$$p(w_j \mid w_1, ..., w_{j-1}) \propto \underbrace{\text{count}(w_1, ..., w_{j-1}, w_j)}_{\text{how often have I seen this \underline{discrete} event in data?}}$$

Example with Sweden:

$$p(\text{Sweden} \mid \text{Norway is..west of}) = \frac{\text{count}(\text{Norway is...west of Sweden})}{\sum_{w} \text{count}(\text{Norway is...west of } w)}$$

Classical Language Modeling

How to estimate? Count-based models, collect counts from large collection of texts (maximum likelihood estimation), *discrete objects*

$$p(w_j \mid w_1, ..., w_{j-1}) \propto \underbrace{\text{count}(w_1, ..., w_{j-1}, w_j)}_{\text{how often have I seen this \underline{discrete} event in data?}}$$

Example with Sweden:

$$p(\text{Sweden} \mid \text{Norway is..west of}) = \frac{\text{count}(\text{Norway is...west of Sweden})}{\sum_{w} \text{count}(\text{Norway is...west of } w)}$$

Immediate Problems: Hard to obtain reliable counts for large contexts (*sparsity*), not feasible to store *all* possible contexts.

Classical Language Modeling

e.g., $t =$Norway is to the west of Sweden

How to approximate? Markov assumption (order=N) (Shannon, 1948):

$$p(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, \ldots, w_{j-1}}_{\text{what came before}})$$

$$\approx \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_{j-N+1}, \ldots, w_{j-1}}_{\text{up to } N \text{ words before}})$$

Classical Language Modeling

e.g., $t =$ Norway is to the west of Sweden

How to approximate? Markov assumption (order$=N$) (Shannon, 1948):

$$p(w_1, w_2, ..., w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, ..., w_{j-1}}_{\text{what came before}})$$

$$\approx \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_{j-N+1}, ..., w_{j-1}}_{\text{up to } N \text{ words before}})$$

$$p_{N=2}(t) = p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times$$
$$p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the})$$
$$p(\text{of} \mid \text{the west}) \times p(\text{Sweden} \mid \text{west of})$$

Classical Language Modeling

e.g., $t =$Norway is to the west of Sweden

How to approximate? Markov assumption (order=N) (Shannon, 1948):

$$p(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, \ldots, w_{j-1}}_{\text{what came before}})$$

$$\approx \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_{j-N+1}, \ldots, w_{j-1}}_{\text{up to } N \text{ words before}})$$

$$\begin{aligned} p_{N=2}(t) =& p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\ & p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the}) \\ & p(\text{of} \mid \text{the west}) \times p(\text{Sweden} \mid \text{west of}) \end{aligned}$$

Observation: Still many parameters, e.g., for vocabulary of 100,000 words, 10^{15} (*quadrillion*) parameters (see Norvig (2012)).

Classical Language Modeling

e.g., $t =$Norway is to the west of Sweden

How to approximate? Markov assumption (order$=N$) (Shannon, 1948):

$$p(w_1, w_2, \ldots, w_m) = \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_1, \ldots, w_{j-1}}_{\text{what came before}})$$

$$\approx \prod_{j=1}^{m} p(\underbrace{w_j}_{\text{current word}} \mid \underbrace{w_{j-N+1}, \ldots, w_{j-1}}_{\text{up to } N \text{ words before}})$$

$$p_{N=2}(t) = p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times$$
$$p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the})$$
$$p(\text{of} \mid \text{the west}) \times p(\text{Sweden} \mid \text{west of})$$

Data: Google n-gram corpus. E.g., (Lin et al., 2012), 8 million books, 6% of all books published, around 500 billion tokens for English.

Classical Language Modeling: Why this doesn't work well

$$
\begin{aligned}
p_{N=2}(t) = &\, p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\
&\, p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the}) \\
&\, p(\text{of} \mid \text{the west}) \times \underbrace{p(\text{Sweden} \mid \text{west of})}_{\text{what is to the west?}}
\end{aligned}
$$

Classical Language Modeling: Why this doesn't work well

$$
\begin{aligned}
p_{N=2}(t) = & p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\
& p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the}) \\
& p(\text{of} \mid \text{the west}) \times \underbrace{p(\text{Sweden} \mid \text{west of})}_{\text{what is to the west?}}
\end{aligned}
$$

Syntactic Structures Chomsky (1957)

> Despite the undeniable interest and importance of semantic and statistical studies of language, they appear to have no direct relevance to the problem of determining and characterizing the set of grammatical utterances.... I think we are forced to conclude that... **probabilistic models give no particular insight into some of the basic problems of syntactic structure**

Classical Language Modeling: Why this doesn't work well

$$
\begin{aligned}
p_{N=2}(t) = & p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times \\
& p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the}) \\
& p(\text{of} \mid \text{the west}) \times \underbrace{p(\text{Sweden} \mid \text{west of})}_{\text{what is to the west?}}
\end{aligned}
$$

Syntactic Structures Chomsky (1957)

> Despite the undeniable interest and importance of semantic and statistical studies of language, they appear to have no direct relevance to the problem of determining and characterizing the set of grammatical utterances.... I think we are forced to conclude that... **probabilistic models give no particular insight into some of the basic problems of syntactic structure**

Technically: do not *faithfully* model complex joint probability distributions $p(w_1, ..., w_m)$, independence assumptions, limited context.

Classical Language Modeling: Why this doesn't work well

$$p_{N=2}(t) = p(\text{Norway} \mid \texttt{<BOS>}) \times p(\text{is} \mid \text{Norway}) \times p(\text{to} \mid \text{Norway is}) \times$$
$$p(\text{the} \mid \text{is to}) \times p(\text{west} \mid \text{to the})$$
$$p(\text{of} \mid \text{the west}) \times \underbrace{p(\text{Sweden} \mid \text{west of})}_{\text{what is to the west?}}$$

elsewhere (see Clark and Lappin (2010); Norvig (2012))

> We cannot seriously propose that a child learns the values of 10^9 parameters in a childhood lasting only 10^8 seconds.

Language Models (LMs): Interim Summary

- **Language models**: Estimate the probability of word sequences, decomposed into next-word prediction, **generation**.

Language Models (LMs): Interim Summary

- **Language models**: Estimate the probability of word sequences, decomposed into next-word prediction, **generation**.

  Classical Methods: words as discrete objects, independence assumptions (otherwise intractable), *do not faithfully model target distributions*.

Contextual Models

Continuous Representations of Linguistic Objects

- A big problem for classical methods: capturing linguistic **similarity** and other types of fuzziness encountered in language.

Continuous Representations of Linguistic Objects

- A big problem for classical methods: capturing linguistic **similarity** and other types of fuzziness encountered in language.

  e.g., *Sweden* is closely related *Norway*, and related (to a lesser degree) to *Italy* and *Portugal*.

Continuous Representations of Linguistic Objects

- A big problem for classical methods: capturing linguistic **similarity** and other types of fuzziness encountered in language.

  > e.g., *Sweden* is closely related *Norway*, and related (to a lesser degree) to *Italy* and *Portugal*.

Old Solution: Represent linguistic objects as continuous vectors (see Widdows (2004)): $\vec{\textbf{sweden}} = [1., 3.], \vec{\textbf{norway}} = [2, 2], \vec{\textbf{italy}} = [6, .7.], ...$

Continuous Representations of Linguistic Objects

- A big problem for classical methods: capturing linguistic **similarity** and other types of fuzziness encountered in language.

  e.g., *Sweden* is closely related *Norway*, and related (to a lesser degree) to *Italy* and *Portugal*.

Old Solution: Represent linguistic objects as continuous vectors (see Widdows (2004)): $\vec{\textbf{sweden}} = [1., 3.], \vec{\textbf{norway}} = [2, 2], \vec{\textbf{italy}} = [6, .7.], \ldots$

$$\textsc{similarity}(\vec{\textbf{sweden}}, \vec{\textbf{norway}}) = \sqrt{(2-1)^2 + (2-3)^2} \approx 1.41$$

$$\textsc{similarity}(\vec{\textbf{sweden}}, \vec{\textbf{italy}}) \approx 2.24$$

Word Vectors

```python
import spacy ## to install: pip install spacy
##download: python -m spacy download en_core_web_md
nlp = spacy.load("en_core_web_md")

norway = nlp("Norway")

### norway word embedding
print(norway.vector)
#array([-7.2509e-01,  2.7890e-01,  2.6907e-01, ...,
#         4.7657e-02, -6.3465e-02,  6.3940e-01, ...,
#        -9.1360e-01, -5.9481e-01, -1.0941e-01,  ...,
#        -3.0875e-01, -4.5818e-01,  1.1814e+00,  ...,
#         4.3488e-01, -9.0963e-02,  5.9045e-01, ...,
#        -4.2597e-01, -3.6353e-01, -8.1080e-02, ...,
#         4.4343e-01,  1.1410e-01,  5.4636e-01,  ...,
#         6.6868e-02, -2.0587e-01, -5.9080e-02, ...,
#         2.7198e-01, -2.7745e-02,  1.4556e-01, ..,
#        -3.6832e-01,  3.1339e-01, -2.3456e-01,  ...,
#        -5.9530e-01,  1.1208e-01, -1.0553e-01, ...,
```

Word Vectors

```
import spacy ## to install: pip install spacy
##download: python -m spacy download en_core_web_md
nlp = spacy.load("en_core_web_md")

norway = nlp("Norway")

### norway word embedding
print(norway.vector)
#array([-7.2509e-01,  2.7890e-01,  2.6907e-01, ...,
#         4.7657e-02, -6.3465e-02,  6.3940e-01, ...,
#        -9.1360e-01, -5.9481e-01, -1.0941e-01,  ...,
#        -3.0875e-01, -4.5818e-01,  1.1814e+00,  ...,
#         4.3488e-01, -9.0963e-02,  5.9045e-01, ...,
#        -4.2597e-01, -3.6353e-01, -8.1080e-02, ...,
#         4.4343e-01,  1.1410e-01,  5.4636e-01,  ...,
#         6.6868e-02, -2.0587e-01, -5.9080e-02, ...,
#         2.7198e-01, -2.7745e-02,  1.4556e-01, ..,
#        -3.6832e-01,  3.1339e-01, -2.3456e-01,  ...,
#        -5.9530e-01,  1.1208e-01, -1.0553e-01, ...,
```

`norway.vector`, general-purpose (non-contextual) representation of *Norway* learned from data (e.g., Mikolov et al. (2013)), useful for similarity.

Word Vectors

```python
import spacy ## to install: pip install spacy
##download: python -m spacy download en_core_web_md
nlp = spacy.load("en_core_web_md")

norway = nlp("Norway")

### norway word embedding
print(norway.vector)
#array([-7.2509e-01,  2.7890e-01,  2.6907e-01, ...,
#        4.7657e-02, -6.3465e-02,  6.3940e-01, ...,
#       -9.1360e-01, -5.9481e-01, -1.0941e-01,  ...,
#       -3.0875e-01, -4.5818e-01,  1.1814e+00,  ...,
#        4.3488e-01, -9.0963e-02,  5.9045e-01, ...,
#       -4.2597e-01, -3.6353e-01, -8.1080e-02, ...,
#        4.4343e-01,  1.1410e-01,  5.4636e-01,  ...,
#        6.6868e-02, -2.0587e-01, -5.9080e-02, ...,
#        2.7198e-01, -2.7745e-02,  1.4556e-01, ..,
#       -3.6832e-01,  3.1339e-01, -2.3456e-01,  ...,
#       -5.9530e-01,  1.1208e-01, -1.0553e-01, ...,
```

semantically: approximates the *intensional* meaning of words, the sense of *Norway*.

Beyond Static Word Vectors

```
## pip install sentence-transformers
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("all-MiniLM-L6-v2")

sentence=["Norway is to the west of Sweden"]
sentence_vector = model.encode(sentence)
print(sentence_vector)
#array([[ 8.98966566e-02,  7.80573040e-02, ...,
#        -5.28105311e-02,  1.03836119e-01, ...,
#        -3.54157202e-02, -2.84450850e-03,  ...,
#         3.65298055e-02,  1.44437077e-02, ...,
#        -1.59231629e-02, -2.63415072e-02, ...,
#        -4.15021786e-03, -3.24299373e-02, ...,
#         4.20359559e-02,  7.27154762e-02, ...,
#       -2.55496521e-02,  4.76241782e-02, ...,
#        1.28677487e-01,  7.31618553e-02,  ...,
#       -5.24449795e-02,  1.07175205e-02,  ...,
#        -4.30012196e-02, -5.51163033e-02,  ....
#.      6.70064837e-02, -6.46066740e-02, ...
```

Recent NLP: learning embedding representations for more complex linguistic objects.

Consequences of moving from discrete to continuous...

- Models can make certain *unforgivable* mistakes.

Consequences of moving from discrete to continuous...

- Models can make certain *unforgivable* mistakes.

Text Normalization (Sproat and Jaitly, 2016)

Input	Correct	Prediction
60	sixty	$\boxed{\text{six}}$
82.55 mm	eighty two point five five millimeters	eighty two one five five $\boxed{\text{meters}}$
2 mA	two milliamperes	two units
£900 million	nine hundred million pounds	nine hundred million $\boxed{\text{euros}}$
16 см	шестнадцати сантиметров	sil сантиметров
16 cm	sixteen centimeters	sil centimeters
-11	минус одиннадцать	минус $\boxed{\text{один}}$ Similar representations

Consequences of moving from discrete to continuous...

- Models can make certain *unforgivable* mistakes.

Text Normalization (Sproat and Jaitly, 2016)

Input	Correct	Prediction
60	sixty	$\boxed{\text{six}}$
82.55 mm	eighty two point five five millimeters	eighty two one five five $\boxed{\text{meters}}$
2 mA	two milliamperes	two units
£900 million	nine hundred million pounds	nine hundred million $\boxed{\text{euros}}$
16 см	шестнадцати сантиметров	`sil` сантиметров
16 cm	sixteen centimeters	`sil` centimeters
-11	минус одиннадцать	минус $\boxed{\text{один}}$

Similar representations

Machine Translation (Arthur et al., 2016)

Input: I come from <u>Tunisia</u>.
Reference: <u>チュニジア</u>の出身です。
<u>Chunisia</u> no shusshindesu.
(I'm from Tunisia.)
System: <u>ノルウェー</u>の出身です。
<u>Noruue-</u> no shusshindesu.
(I'm from Norway.)

Consequences of moving from discrete to continuous...

- Models can make certain *unforgivable* mistakes.

Text Normalization (Sproat and Jaitly, 2016)

Input	Correct	Prediction
60	sixty	$\boxed{\text{six}}$
82.55 mm	eighty two point five five millimeters	eighty two one five five $\boxed{\text{meters}}$
2 mA	two milliamperes	two units
£900 million	nine hundred million pounds	nine hundred million $\boxed{\text{euros}}$
16 см	шестнадцати сантиметров	`sil` сантиметров
16 cm	sixteen centimeters	`sil` centimeters
-11	минус одиннадцать	минус $\boxed{\text{один}}$ Similar representations

Machine Translation (Arthur et al., 2016)

Input:	I come from Tunisia.
Reference:	チュニジアの出身です。 Chunisia no shusshindesu. *(I'm from Tunisia.)*
System:	ノルウェーの出身です。 Noruue- no shusshindesu. *(I'm from Norway.)*

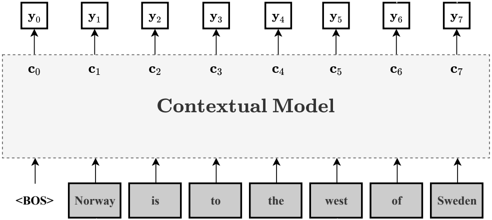
The only way this can work: *learning* good representations from data.

Contextual Models

- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.

Contextual Models

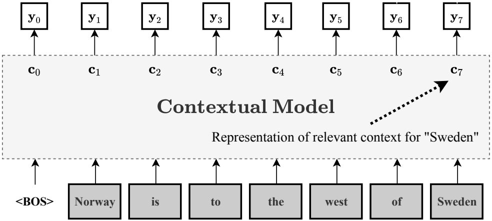
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Operationally: neural network models, often large and opaque.

Contextual Models

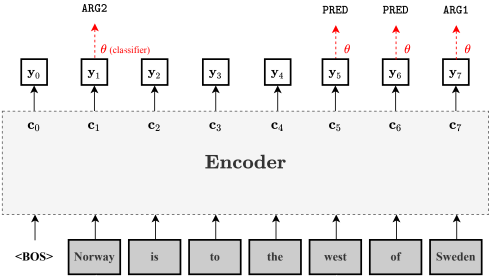
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Semantics: $\mathbf{y}_7$ captures the meaning of *Sweden* grounded in this this particular sentential context, output of a compositional procedure.

Contextual Models

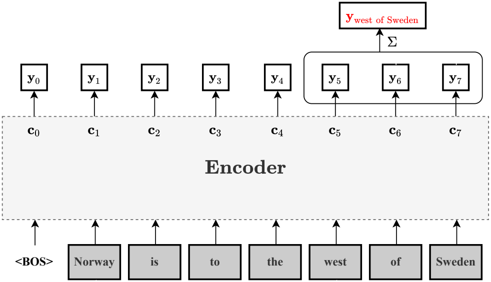
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Why? Build representations that allow you make predictions, **success**: learned representations that effectively solve problems.

Contextual Models

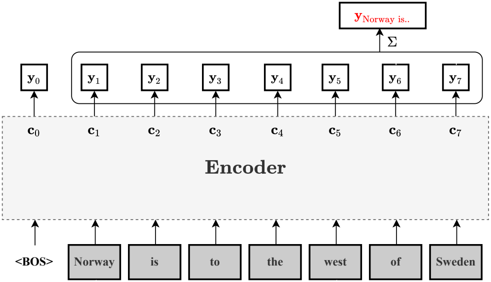
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Why? Build representations that allow you make predictions, **success**: learned representations that effectively solve problems.

Contextual Models

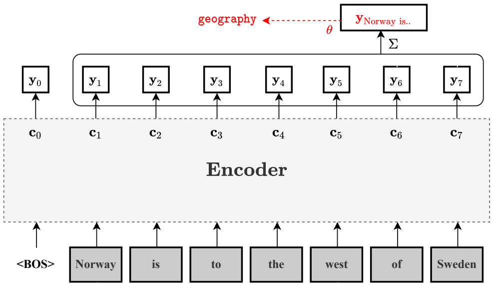
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Why? Build representations that allow you make predictions, **success**: learned representations that effectively solve problems.

Contextual Models

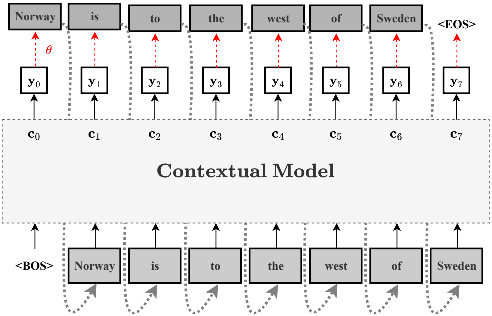
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Why? Build representations that allow you make predictions, **success**: learned representations that effectively solve problems.

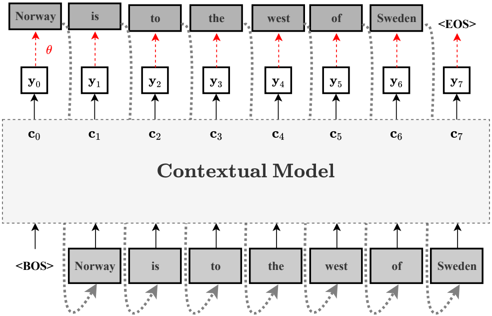
Contextual Models

- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Contextual Models

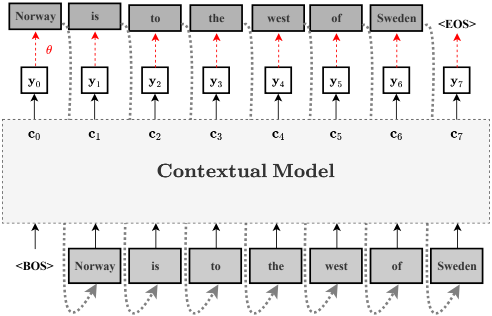
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



as LMs: $p(w_j \mid w_1, ..., w_{j-1}) = p(w_j \mid \mathbf{c}_{j-1})$, **Important:** Can condition on full contexts, *faithfully* model complex joint probability distributions

Contextual Models

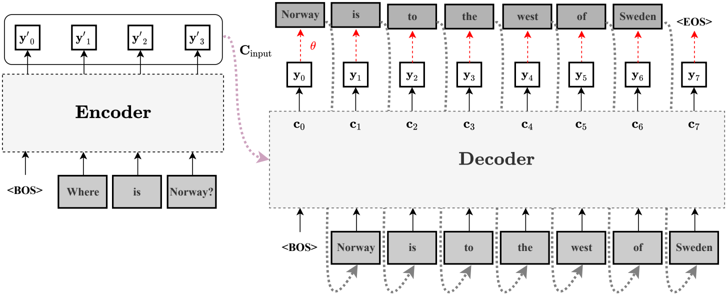
- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Model Architectures: how information is processed, internal representations are constructed. **Common**: RNNs, Transformers.

Contextual Models

- **Role**: assign continuous vectors representations $\mathbf{y}_j \in \mathbb{R}^d$ to elements in a sequence that capture their meaning of those elements in context.



Text2Text models: estimate $p(y^{\text{output}} \mid x^{\text{input}})$, natural way to express many language understanding problems.

Contextual Models: Interim Summary

- **Contextual Model**: builds continuous representations for sequential input, neural networks, faithfully models full context

Contextual Models: Interim Summary

- **Contextual Model**: builds continuous representations for sequential input, neural networks, faithfully models full context

  Used to build *contextual language models* and variants: *text2text models* (**encoders** vs. **decoders**).

Contextual Models: Interim Summary

- **Contextual Model**: builds continuous representations for sequential input, neural networks, faithfully models full context

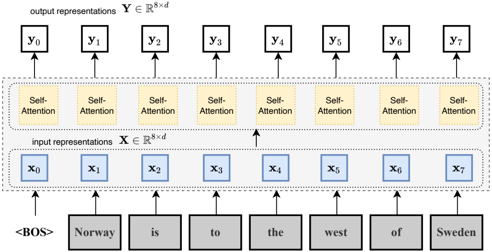
  Used to build *contextual language models* and variants: *text2text models* (**encoders** vs. **decoders**).

essential questions *(architecture)*: how does information flow? how are representations built?

Attention-based Models

Self-Attention: Simple Encoder Example

Attention: Mechanism for building contextual representations (see Vaswani et al. (2017)).

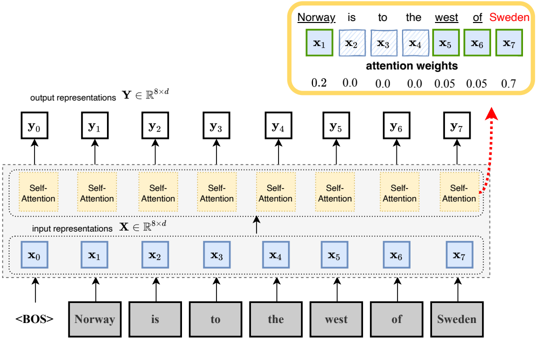


Self-Attention: Embeddings Representations

```
import torch ## to install: pip install pytorch

### word embedding parameters and matrix E
E = torch.nn.Embedding(
    embedding_dim=768,  ##<--- dimensionality
    num_embeddings=3000, ##<--- # words
)

##e.g., Representation of our Input
X = E(torch.tensor([
        0, # <BOS>
        1, # Norway
        2, # is
        3, # to
        4, # the
        5, # west
        6, # of
        7, # Sweden
])) ### => matrix 7 * 768

x1 = X[1] ##-> initial representation of Norway
print(x1)
###=> [-1.1344e+00, -3.0359e-01,  7.3585e-02, .....]
```

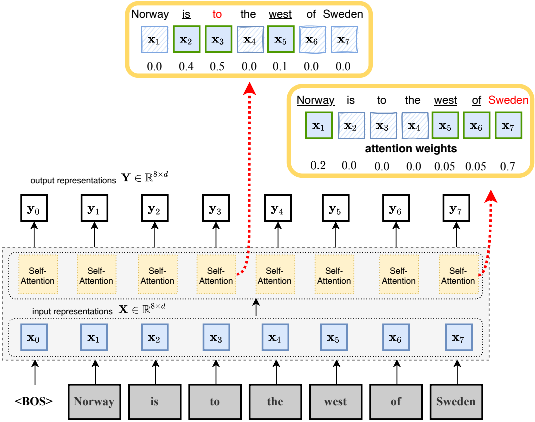
Self-Attention: Simple Encoder Example

Attention: Mechanism for building contextual representations (see Vaswani et al. (2017)).



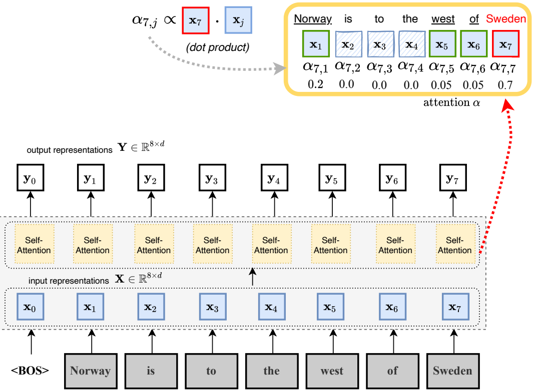
Self-Attention: Simple Encoder Example

Attention: Mechanism for building contextual representations (see Vaswani et al. (2017)).



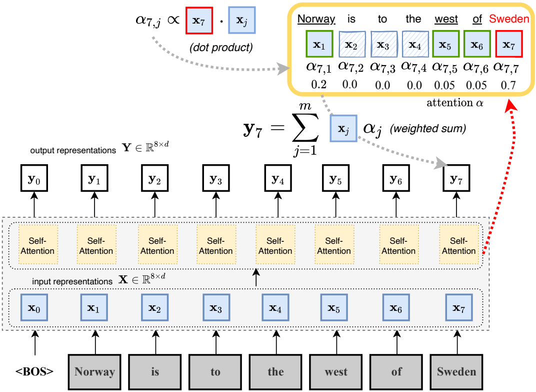
Self-Attention: Simple Encoder Example

Attention: Mechanism for building contextual representations (see Vaswani et al. (2017)).



Self-Attention: Simple Encoder Example

Attention: Mechanism for building contextual representations (see Vaswani et al. (2017)).



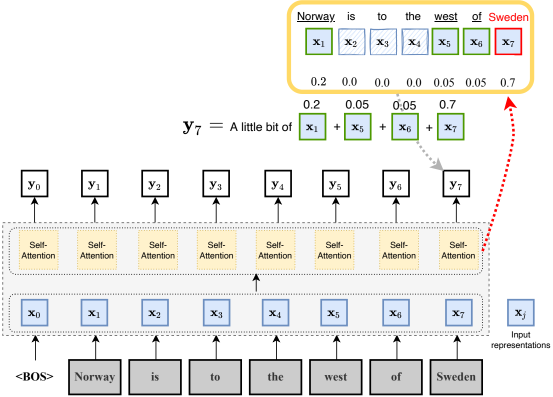
Self-Attention: Computing Final Representations

Reduces to a few lines of PyTorch code.

```
## Input representations (again),
X = E(torch.tensor([0,1,2,3,4,5,6,7]))

### raw weights (dot product / matrix multiplication)
raw_Alpha =  torch.matmul(X,X.transpose(0,1))

### normalized via softmax, probability distribution
alpha =  raw_Alpha.softmax(dim=-1)

### Final self attention representations
Y =  torch.matmul(alpha,X)

### self attention representation of ‘Norway‘
y1 = Y[1]
```

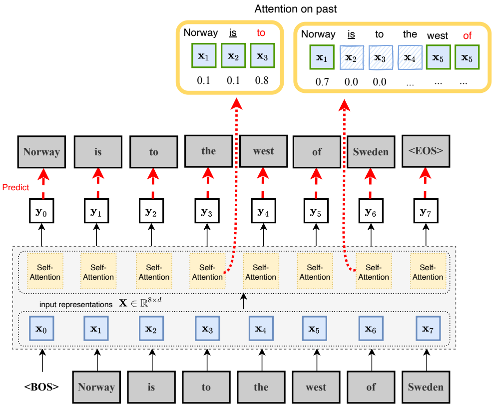
Self-Attention: Intuition

Attention: A kind of brute-force looking around and aggregation of contextual information.



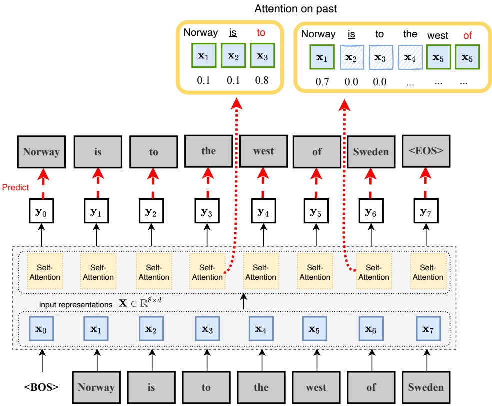
Attention in Other Contexts

Decoders: Attention limited to past context, allows for generation (step-by-step prediction of next word).



Attention in Other Contexts

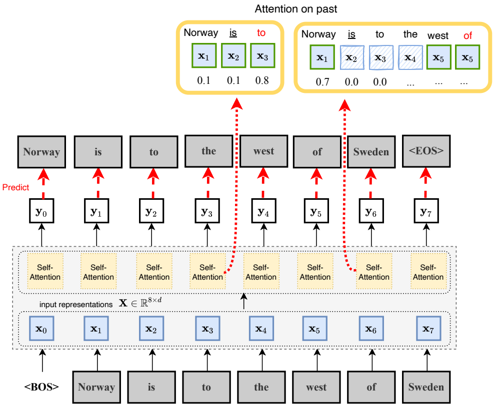
Decoders: Attention limited to past context, allows for generation (step-by-step prediction of next word).



Note: No independence assumptions, full context.

Attention in Other Contexts

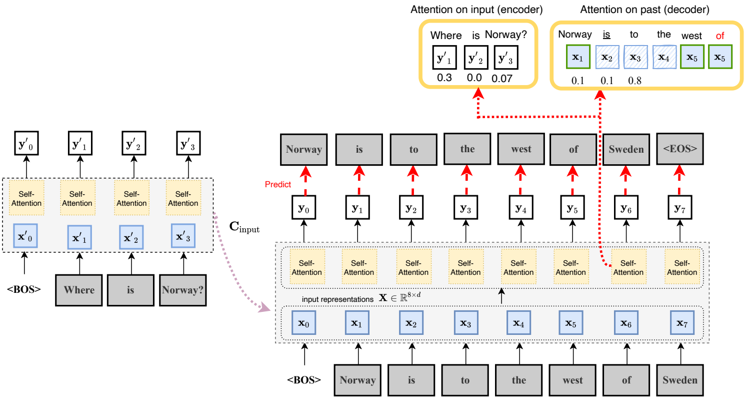
Decoders: Attention limited to past context, allows for generation (step-by-step prediction of next word).



Additional component: Classifier for predicting words from $\mathbf{y}_j$.

Attention in Other Contexts

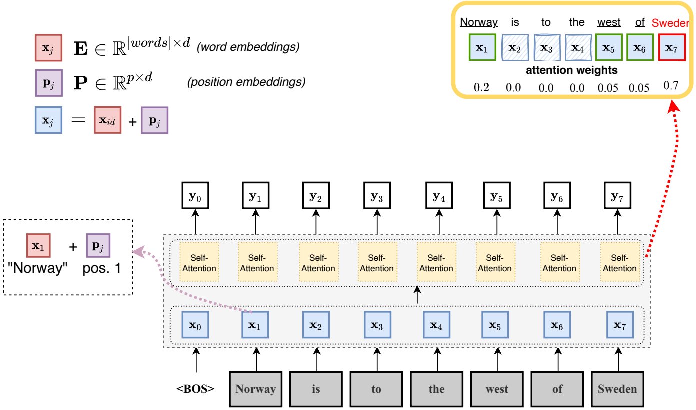
Text2Text: Additional attention on an input (*cross attention*)



Current Transformers

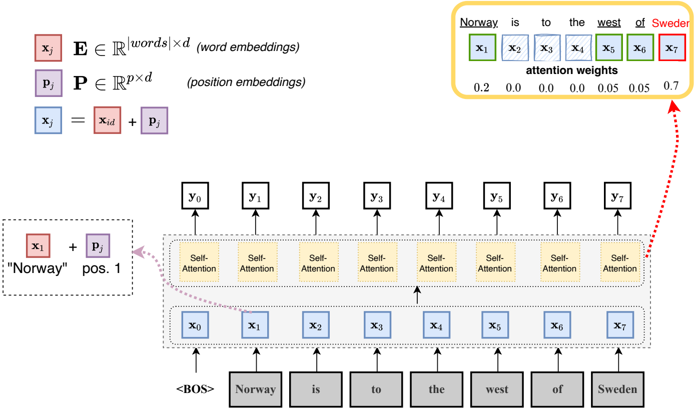
An important detail: positional information

- Word embeddings so far do not encode position information.



An important detail: positional information

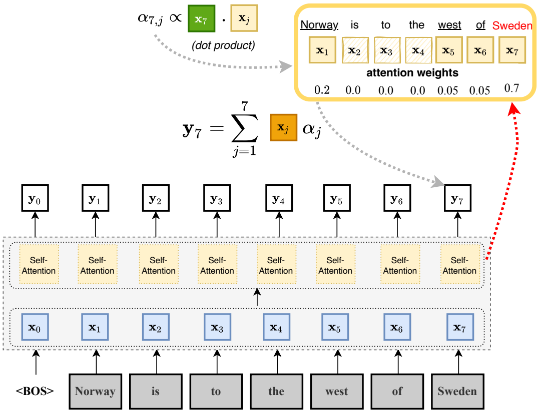
- Word embeddings so far do not encode position information.



Note: we can add any additional information we want, many variations.

More Parameters: Keys, Queries and Values

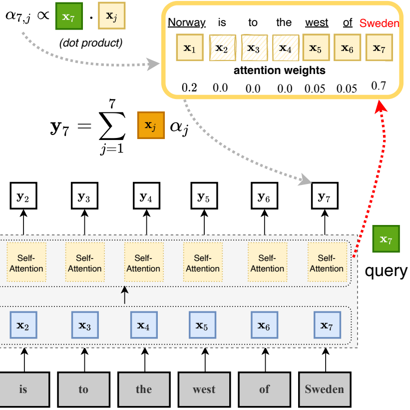
- Three components in our computation.



More Parameters: Keys, Queries and Values

- Three components in our computation.

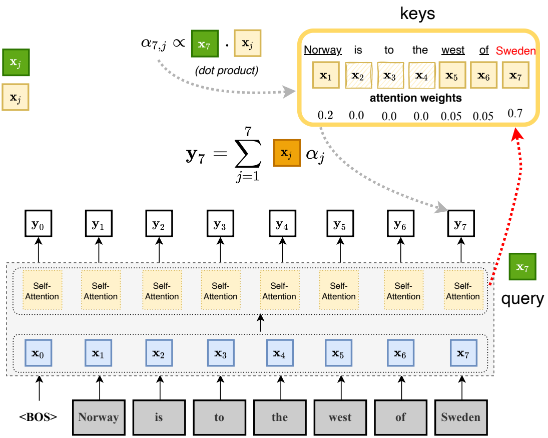
1. **query** $\mathbf{x}_j$



More Parameters: Keys, Queries and Values

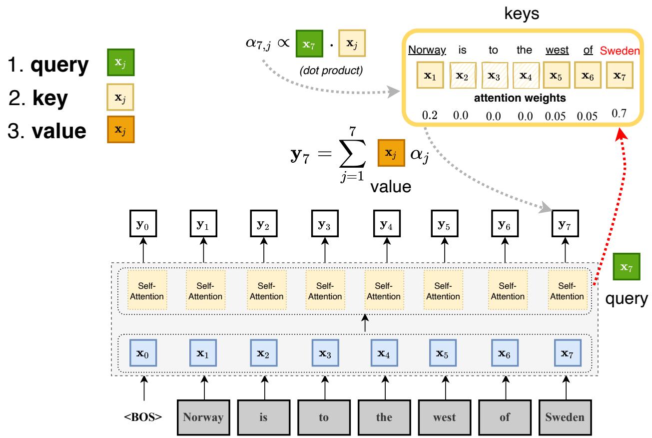
- Three components in our computation.

1. **query** $\mathbf{x}_j$
2. **key** $\mathbf{x}_j$



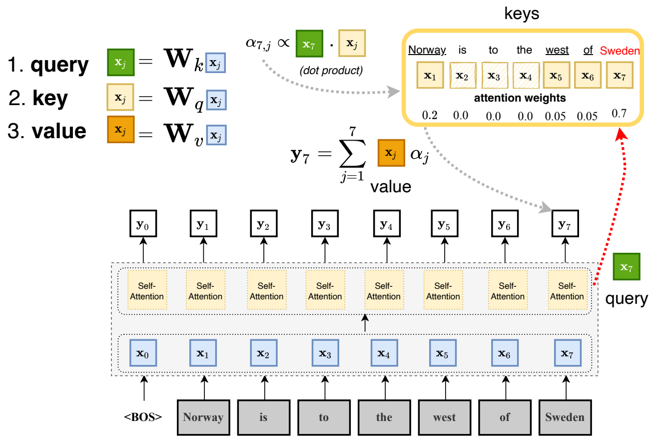
More Parameters: Keys, Queries and Values

- Three components in our computation.



More Parameters: Keys, Queries and Values

- Three components in our computation.



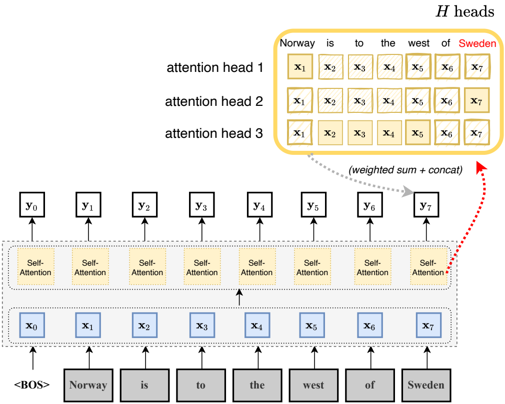
More Parameters: Keys, Queries and Values

Another few additional lines of PyTorch

```
## Input representations (again),
X = E(torch.tensor([0,1,2,3,4,5,6,7]))
D = 768

### key, value, query parameters (linear layer)
W_k   = torch.nn.Linear(D, D, bias=False)
W_q   = torch.nn.Linear(D, D, bias=False)
W_v   = torch.nn.Linear(D, D, bias=False)

key_rep    = W_k(X)  #<-- rep. of X as keys
query_rep  = W_q(X) #<-- rep. of X as queries
value_rep  = W_v(X) #<-- rep of X as values

### sample computation with parameters applied over 'X'
alpha = torch.matmul(
   query_rep,
   key_rep.transpose(0,1)
).softmax(dim=-1)

### same as before
Y = torch.matmul(alpha,value_rep)
```

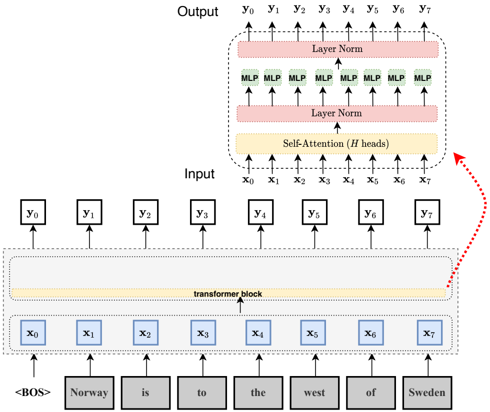
Multi-headed Attention

- Allows the model to simultaneously focus on multiple parts of input.



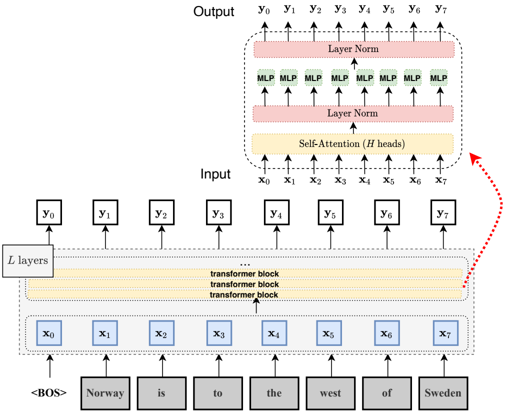
Transformer Blocks and Multiple Layers

- Current models are a bit more complex and multi-layered.



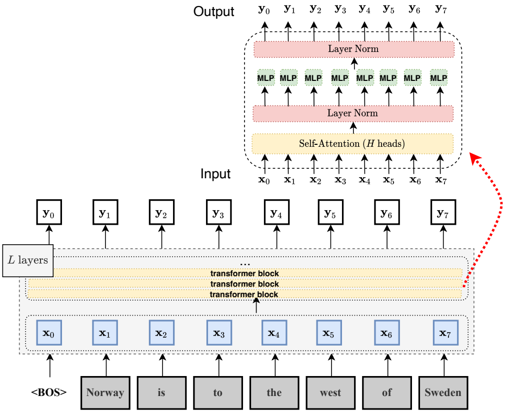
Transformer Blocks and Multiple Layers

- Current models are a bit more complex and multi-layered.



Transformer Blocks and Multiple Layers

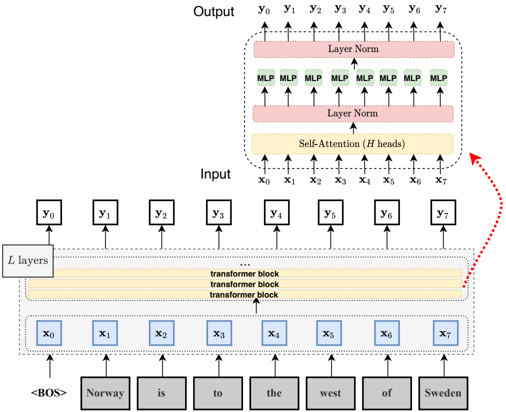
- Current models are a bit more complex and multi-layered.



BERT (Devlin et al., 2018): $L = 24$ layers each with $H = 16$ heads, *340M* parameters, embedding dimension=1024

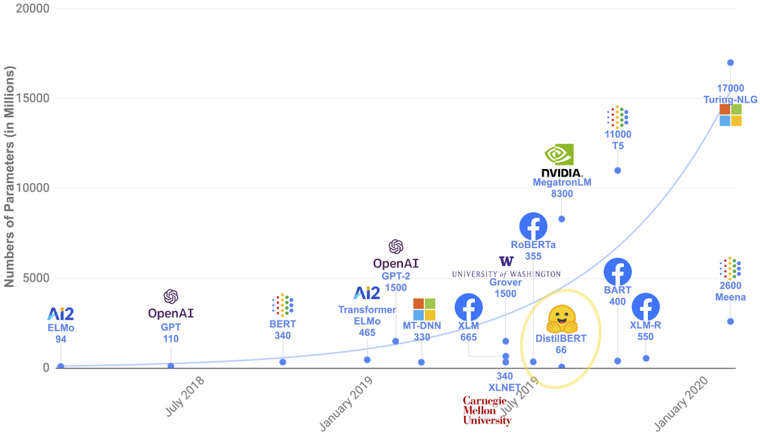
Transformer Blocks and Multiple Layers

- Current models are a bit more complex and multi-layered.



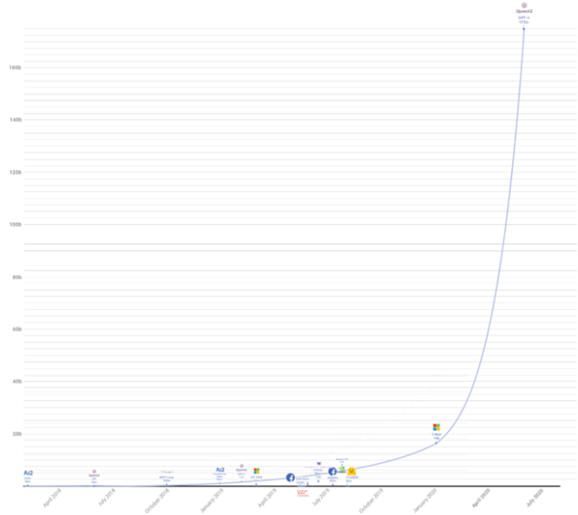
GPT3 (Brown et al., 2020): $L = 96$ layers each with $H = 96$ heads (*175B parameters*), embedding dimension=12288

The Race for Larger Models (Sanh et al., 2019)

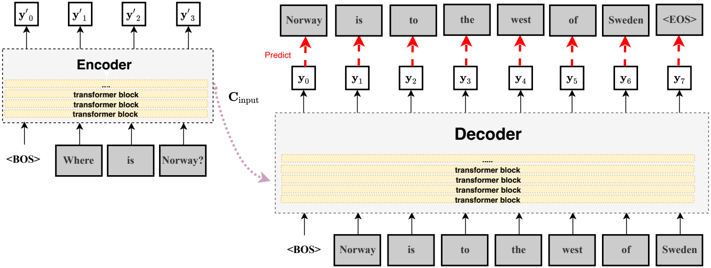


Not (yet) a quadrillion parameters, but models are quickly becoming much larger.

A more updated picture

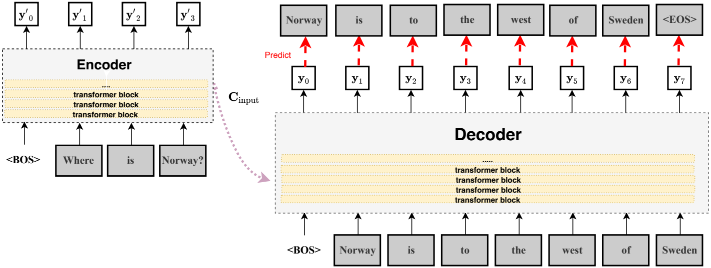


Our Target Model (T5)



T5 model (Raffel et al., 2020), text2text architecture: **encoder** and **decoder**, 12-24 layers, 12-24 heads, cross-attention (220-770 m. parameters).

Our Target Model (T5)



Decoding Procedure: search procedure (usually approximate) for finding the best output, typical solutions: **beam search** or sampling.

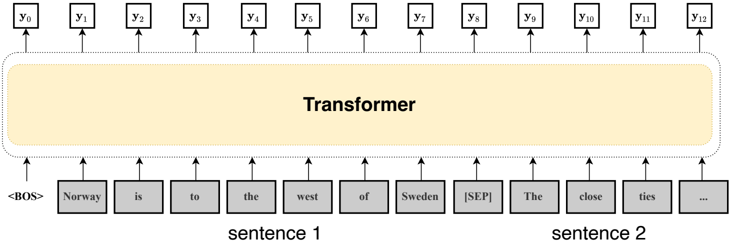
Attention and Transformers: Interim Summary

- **Attention**: mechanism for building contextual representations via brute-force looking around, *variants*: self, causal, cross.

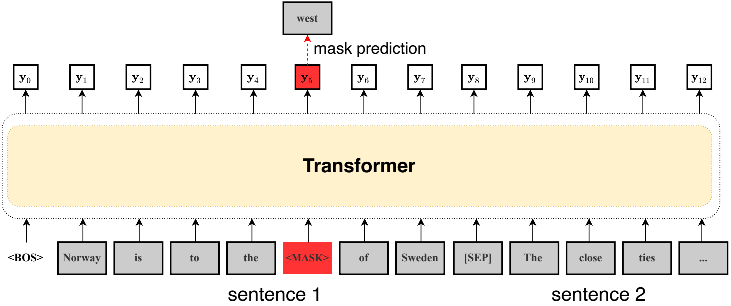
  Transformer: architecture based on attention (*plus other components, transformer block*). **Existing models**: multi-layered, many parameters.

Training and Using Models

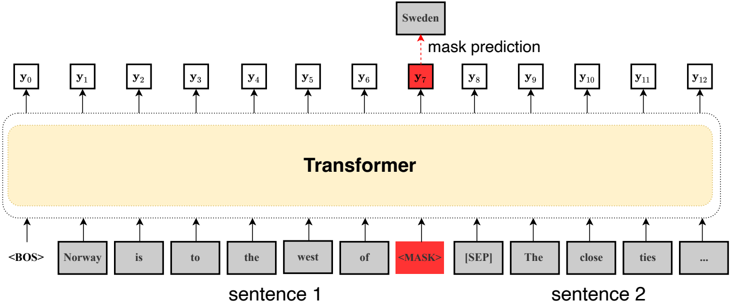
Word Prediction Training: Encoders (Devlin et al. (2018))



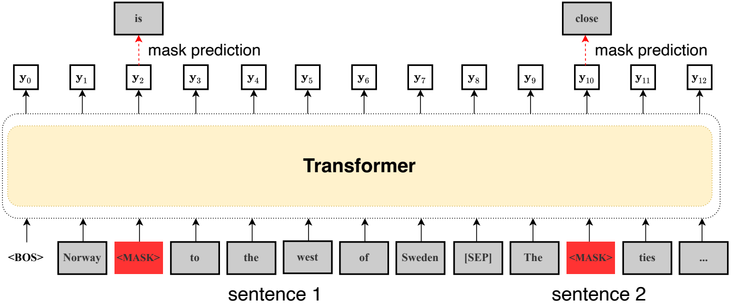
Word Prediction Training: Encoders (Devlin et al. (2018))



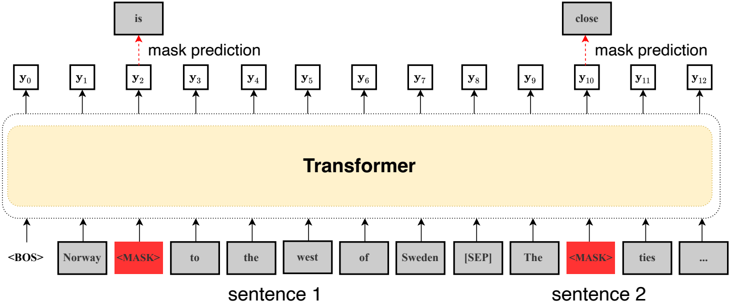
Word Prediction Training: Encoders (Devlin et al. (2018))



Word Prediction Training: Encoders (Devlin et al. (2018))

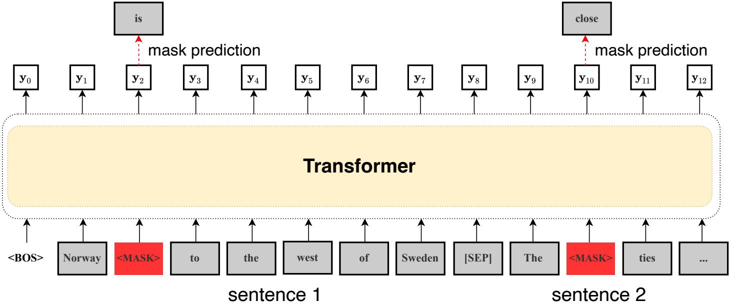


Word Prediction Training: Encoders (Devlin et al. (2018))



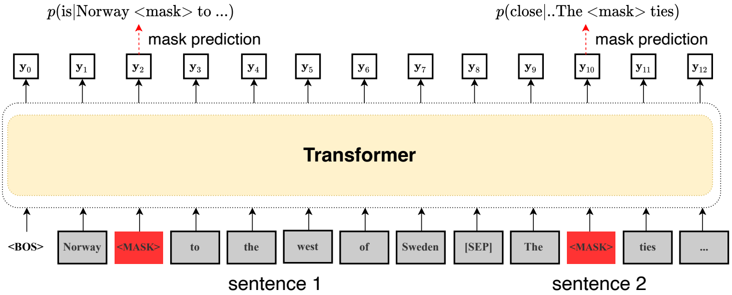
Does this make sense to do? Has long been used as a teaching and assessment technique for humans (**cloze test**) (Taylor (1953)).

Word Prediction Training: Encoders (Devlin et al. (2018))



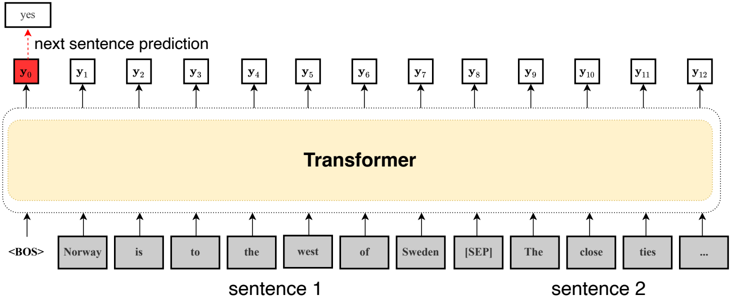
Self-supervised learning: no need for data annotation, can be scaled to large amounts of unlabeled data!

How this works technically



maximum likelihood training (**mask language objective**), trained using first-order optimization, *stochastic gradient descent*.

How this works technically



maximum likelihood training (**mask language objective**), trained using first-order optimization, *stochastic gradient descent*.

How do we find the right representations?

- This is the part that involves **machine learning** and optimization.

Machine Learning

From xkcd

How do we find the right representations?

- This is the part that involves **machine learning** and optimization.

Machine Learning

From xkcd

Important: with these tools, not much more needed, all parameters are tuned to these simple objectives.

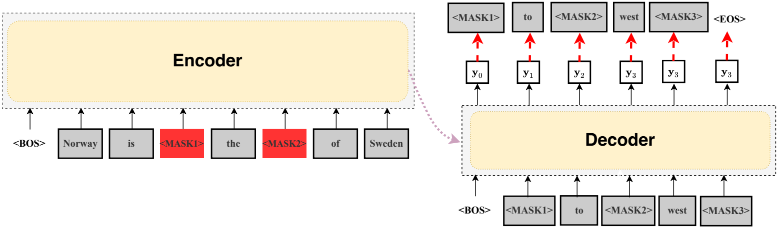
Mask prediction in action

- An example using **RoBERTa** (Liu et al., 2019).

```
## install: pip install transformers
from transformers import pipeline

mask_predictor = pipeline(
  "fill-mask", model="roberta-base"
)

mask_predictor("This lecture is really <mask> .")
#[{'score': 0.392790824174881,
#   'token_str': ' good',}
# {'score': 0.09650349617004395,
#   'token_str': ' interesting',}
# {'score': 0.0915081575512886,
#   'token_str': ' great',}
# {'score': 0.03177962079644203,
#   'token_str': ' important',}
# {'score': 0.03151006624102592 5,
# 'token_str': ' long',} ...]
```

Word Prediction Training: Text2Text Models

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>



(Raffel et al., 2020)

- From perturbed input, reconstruct target sentence via generation; related to *generative* pre-training for decoders-only models (Radford et al., 2018).

Mask Generation in action: T5 (text2text)

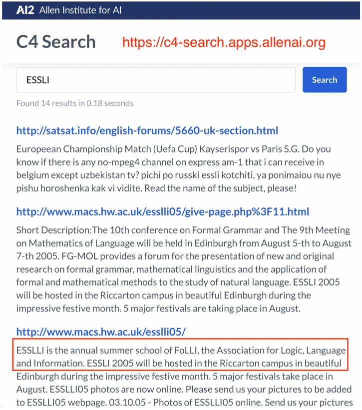
```
## install: pip install transformers
from transformers import (
   T5ForConditionalGeneration as T5,
   T5Tokenizer
)
## model and tokenizer
tokenizer = T5Tokenizer.from_pretrained("t5-large")
model = T5.from_pretrained("t5-large")

###input with a mask
model_input=tokenizer(
  "The Palace is a famous <extra_id_0> in Dublin.",
  return_tensors="pt"
).input_ids

### run generator, decoding
outputs = model.generate(
   input_ids,num_return_sequences=5,
   max_length=4,num_beams=10 #<-- beam search params.
)
### output predictions
[tokenizer.decode(i,skip_special_tokens=True) \
        for i in outputs]
### ["landmark", "hotel", "building", "restaurant",
### "tourist attraction"]
```

Large-scale pre-training

- **Pre-training**: training models (e.g., using masking objectives) on a large corpus of text, learn general-purpose representations.

Large-scale pre-training

Web-scale resources for training e.g., C4 (Raffel et al. (2020); Dodge et al. (2021)), English (*raw*): 1.4Tr tokens, 1.1B documents (2.3 TB raw text).



AI2 Allen Institute for AI

C4 Search https://c4-search.apps.allenai.org

ESSLI

Search

Found 14 results in 0.18 seconds

http://satsat.info/english-forums/5660-uk-section.html

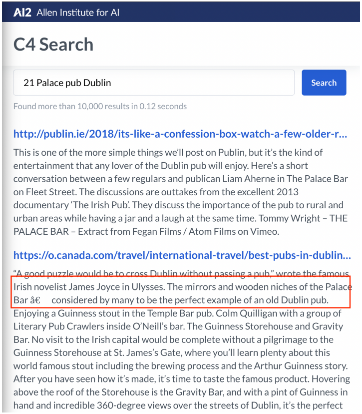
European Championship Match (Uefa Cup) Kayserispor vs Paris S.G. Do you know if there is any no-mpeg4 channel on express am-1 that i can receive in belgium except uzbekistan tv? pichi po russki essli kotchiti, ya ponimaiou nu nye pishu horoshenka kak vi vidite. Read the name of the subject, please!

http://www.macs.hw.ac.uk/esslli05/give-page.php%3F11.html

Short Description:The 10th conference on Formal Grammar and The 9th Meeting on Mathematics of Language will be held in Edinburgh from August 5-th to August 7-th 2005. FG-MOL provides a forum for the presentation of new and original research on formal grammar, mathematical linguistics and the application of formal and mathematical methods to the study of natural language. ESSLI 2005 will be hosted in the Riccarton campus in beautiful Edinburgh during the impressive festive month. 5 major festivals are taking place in August.

http://www.macs.hw.ac.uk/esslli05/

ESSLLI is the annual summer school of FoLLI, the Association for Logic, Language and Information. ESSLI 2005 will be hosted in the Riccarton campus in beautiful Edinburgh during the impressive festive month. 5 major festivals take place in August. ESSLLI05 photos are now online. Please send us your pictures to be added to ESSLLI05 webpage. 03.10.05 - Photos of ESSLLI05 online. Send us your pictures



AI2 Allen Institute for AI

C4 Search

21 Palace pub Dublin

Search

Found more than 10,000 results in 0.12 seconds

http://publin.ie/2018/its-like-a-confession-box-watch-a-few-older-r...

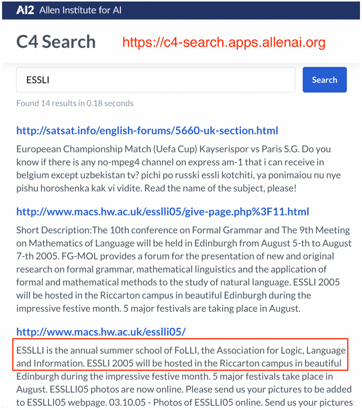
This is one of the more simple things we'll post on Publin, but it's the kind of entertainment that any lover of the Dublin pub will enjoy. Here's a short conversation between a few regulars and publican Liam Aherne in The Palace Bar on Fleet Street. The discussions are outtakes from the excellent 2013 documentary 'The Irish Pub'. They discuss the importance of the pub to rural and urban areas while having a jar and a laugh at the same time. Tommy Wright – THE PALACE BAR – Extract from Fegan Films / Atom Films on Vimeo.

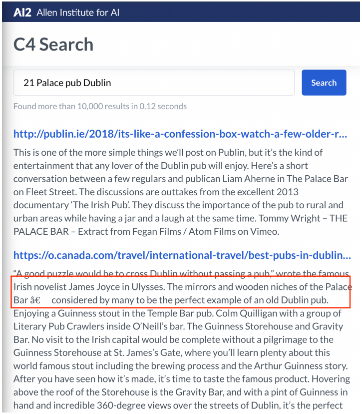
https://o.canada.com/travel/international-travel/best-pubs-in-dublin...

"A good puzzle would be to cross Dublin without passing a pub," wrote the famous Irish novelist James Joyce in Ulysses. The mirrors and wooden niches of the Palace Bar â€ considered by many to be the perfect example of an old Dublin pub. Enjoying a Guinness stout in the Temple Bar pub. Colm Quilligan with a group of Literary Pub Crawlers inside O'Neill's bar. The Guinness Storehouse and Gravity Bar. No visit to the Irish capital would be complete without a pilgrimage to the Guinness Storehouse at St. James's Gate, where you'll learn plenty about this world famous stout including the brewing process and the Arthur Guinness story. After you have seen how it's made, it's time to taste the famous product. Hovering above the roof of the Storehouse is the Gravity Bar, and with a pint of Guinness in hand and incredible 360-degree views over the streets of Dublin, it's the perfect

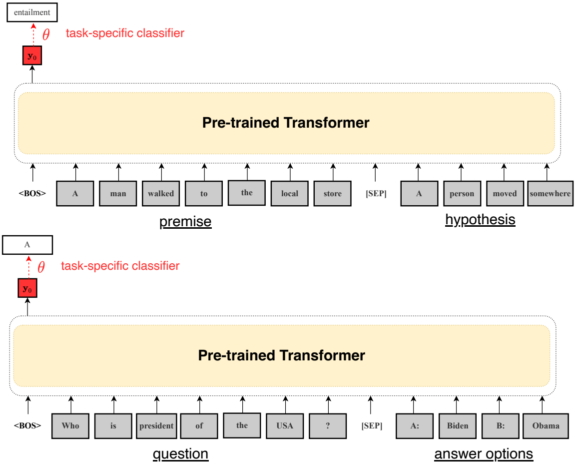
Large-scale pre-training

What's in this corpus: we don't really know, a little of everything (see Dodge et al. (2021)), blackbox (*similar to our models*).



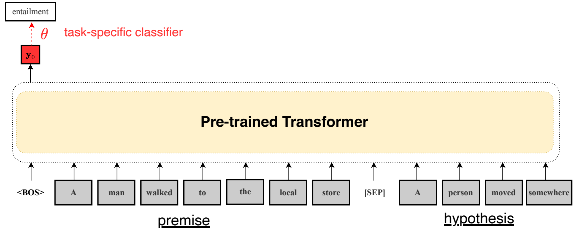


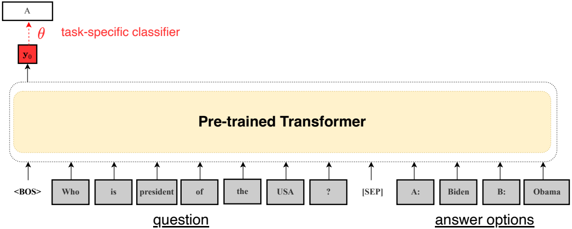
How models are used: fine-tuning for classification



Fine-tuning: customizing models to target tasks using additional parameters; **idea**: bootstrap off of pre-training knowledge.

Typical experiment

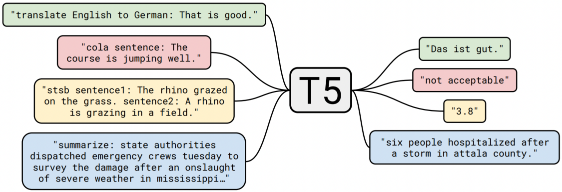




Supervised Learning: fine-tune using labeled training set, measure and report prediction accuracy on unseen set.

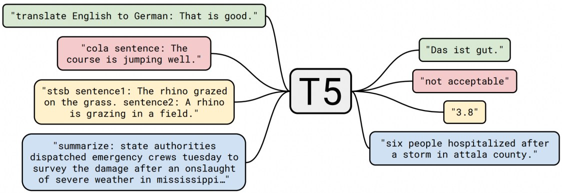
Fine-tuning Text2Text Models

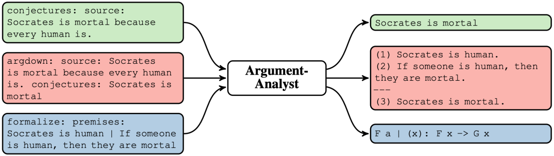
- Treat all problems as a (text2text) translation problem (Raffel et al. (2020)), (*as before*) fine-tune on specialized tasks recast in this way.



Fine-tuning Text2Text Models

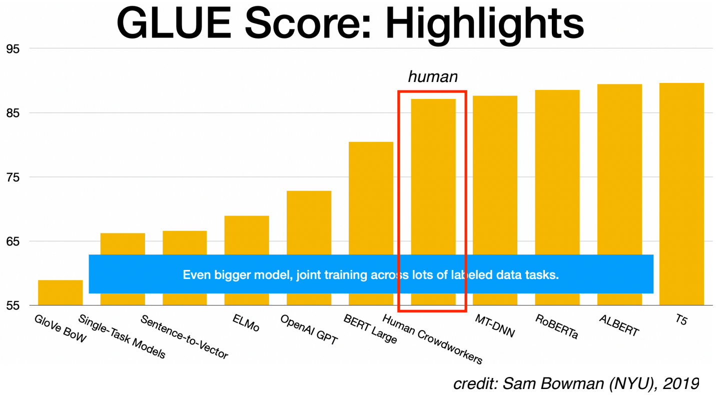
- Treat all problems as a (text2text) translation problem (Raffel et al. (2020)), (*as before*) fine-tune on specialized tasks recast in this way.





Spectacular Results

- GLUE benchmark (Wang et al., 2018), large collection of diverse 9 NLU tasks; *models outperforming estimates of human performance*.



State-of-the-art fine-tuned models in a few lines

- **Huggingface Transformers**[1]: large Python library of transformer code with corresponding models, datasets and other utilities.

```python
from transformers import pipeline
## https://huggingface.co/docs/transformers/v4.21.1/
##en/main_classes/pipelines

### sentiment analysis
sentiment_model = pipeline("sentiment-analysis")

sentiment_model("This is a terrible lecture")
# [{'label': 'NEGATIVE', 'score': 0.999454915523529}]

### translation
en_fr_translator = pipeline("translation_en_to_fr")
en_fr_translator("How old are you?")

## summarization
summarizer = pipeline("summarization")
summarizer("An apple a day, keeps the doctor away",
        min_length=5, max_length=20)
```

[1] https://huggingface.co/docs/transformers/index

Conclusion

- **Language models**: assign probabilities to sequences, generation.
- **Recent Advancess**: novel neural architectures (*transformers*, *attention*), large-scale pre-training of neural representations, enormous datasets.

  State-of-the-art results on many NLP problems, many exciting areas for future research and new applications.

Credits and Additional Resources

- Many examples taken from **Peter Bloem**'s excellent blogpost[2] on transformers: https://peterbloem.nl/blog/transformers. *Also*: **Sasha Rush**'s *Annotated Transformer* https://blog.rush-nlp.com/the-annotated-transformer.html, **Jay Alammar**'s *Illustrated Transformer*: https://jalammar.github.io/illustrated-transformer/.

- *Code resources*: **Huggingface Transformers**: https://github.com/huggingface/transformers, **PyTorch**: https://pytorch.org/, **spaCY**: https://spacy.io/

  Topics glossed over: **Decoding** and **Beam Search**, see https://huggingface.co/blog/how-to-generate, General machine learning and optimization, consult Murphy (2012); Daumé (2017); Goodfellow et al. (2016)[3]

[2] Code examples here adapted from his corresponding code available at https://github.com/pbloem/former.
[3] http://ciml.info/, see especially Chapter 7 and 10.

References I

Arthur, P., Neubig, G., and Nakamura, S. (2016). Incorporating discrete translation lexicons into neural machine translation. *arXiv preprint arXiv:1606.02006*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Chomsky, N. (1957). Syntactic structures. In *Syntactic Structures*. De Gruyter Mouton.

Clark, A. and Lappin, S. (2010). *Linguistic Nativism and the Poverty of the Stimulus*. John Wiley & Sons.

Daumé, H. (2017). *A course in machine learning*. Hal Daumé III.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dodge, J., Sap, M., Marasovic, A., Agnew, W., Ilharco, G., Groeneveld, D., and Gardner, M. (2021). Documenting the english colossal clean crawled corpus.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Lin, Y., Michel, J.-B., Lieberman, E. A., Orwant, J., Brockman, W., and Petrov, S. (2012). Syntactic annotations for the google books ngram corpus. In *Proceedings of the ACL 2012 system demonstrations*, pages 169–174.

References II

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Norvig, P. (2012). Colorless green ideas learn furiously: Chomsky and the two cultures of statistical learning. *Significance*, 9(4):30–33.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Sproat, R. and Jaitly, N. (2016). Rnn approaches to text normalization: A challenge. *arXiv preprint arXiv:1611.00068*.

References III

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Widdows, D. (2004). *Geometry and meaning*. CSLI.