How to handle text data?
Language Modelling History

Legend:
- Before Transformer
- After Transformer


- RNN / LSTM
- Word2Vec N-grams
- Transformer
- BERT
- T5
- GPT-3
- PaLM
- Bard / GPT-4

Language Modeling: predict the next word

Assign probabilities to text.

Given a sequence \((x_1, x_2, \ldots, x_T)\), we want to maximize \(P(x_1, x_2, \ldots, x_T)\).

\[
P(x_1, x_2, \ldots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)P(x_4|x_1, x_2, x_3) \ldots P(x_T|x_1, x_2, \ldots x_{T-1})
\]

\(P(I \text{ like cats because they look cute}) = P(I) P(\text{like} | I) P(\text{cats} | I \text{ like}) P(\text{as} | I \text{ like cats}) P(\text{they} | I \text{ like cats because}) P(\text{look} | I \text{ like cats because they}) P(\text{cute} | I \text{ like cats because they look})\)

Predict the next word given current text!
**n-Gram Language Model**

*n-Gram*: chunk of *n* consecutive words

- **Uni-gram**: “I” “like” “cats” “as” “they” “look” “cute”
- **Bi-gram**: “I like” “like cats” “cats as” “as they” …
- **Tri-gram**: “I like cats” “like cats as” “cats as they” …

Count the frequency of each *n*-grams and predict next word!

Assume each word only depends on previous *n - 1* words.

\[
P(x_t | x_1, \ldots, x_{t-1}) = \frac{\text{count}(x_{t-n+1}, \ldots, x_{t-1}, x_t)}{\text{count}(x_{t-n+1}, \ldots, x_{t-1})}
\]

In bi-gram LM

\[
P(\text{I like cats as they look cute}) = P(\text{I}) P(\text{like | I}) P(\text{cats | like}) P(\text{as | cats}) P(\text{they | because}) P(\text{look | they}) P(\text{cute | look})
\]
Discuss: Do you want to have a large $n$ or a small $n$ in a $n$-gram model?
I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

https://koushik1102.medium.com/nlp-bag-of-words-and-tf-idf-explained-fd1f49dce7c4
Bag of Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun...

Drawbacks:
- High dimensionality
- No semantic information
Word Embeddings

- **Motivation**
  - Put words into vectors so we can measure the similarity between words
  - Use cosine similarity

Blitzer et al. 2004
What does ong choi mean?

Suppose you see these sentences:

- Ong choi is delicious sautéed with garlic.
- Ong choi is superb over rice
- Ong choi leaves with salty sauces

And you've also seen these:

- ...spinach sautéed with garlic over rice
- Chard stems and leaves are delicious
- Collard greens and other salty leafy greens

https://web.stanford.edu/~jurafsky/slp3/
Ong choy is a leafy green vegetable with long, hollow stems and slender leaves. It's also known as Chinese water spinach, Chinese water spinach, or hollow stem spinach.
Word2Vec

- We want vectors for words so that the context of a word can suggest the vector of this word, and vice versa.
- Idea: Similar words appear in similar contexts.

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.
Word2Vec - Training

**SkipGram**

- **Context words in window of size 2**
- **Center word**
- **Context words in window of size 2**

- $P(w_{t-2} | w_t)$
- $P(w_{t-1} | w_t)$
- $P(w_{t+1} | w_t)$
- $P(w_{t+2} | w_t)$

**cBOW**

- **Context words in window of size 2**
- **Center word**
- **Context words in window of size 2**

- $P(w_t | w_{t-2})$
- $P(w_t | w_{t-1})$
- $P(w_{t+1} | w_t)$
- $P(w_t | w_{t+2})$
Looking closer…

\[ p(w_c|w_t) = \frac{\exp(w_c^T w_t)}{\sum_j \exp(w_c^T w_j)} \]

(One option is to have two embeddings for each word, one as target and one as context.)

Cross-Entropy Loss:

\[ \mathcal{L}_W = - \sum_{(c,t) \in D} \log (p(w_c|w_t)) \]
Word2Vec Architecture - SkipGram

Predict context word from target word!

coffee → cup

One hot encoding of the context word (dim=|V|)
Word2Vec Architecture - CBOW (continuous bag of words)

(cup, of, is, on) → coffee

- **Input**: \( x \) (vector of dimensions \( d \))
- **Hidden Layer**: \( h \) (vector of dimensions \( d \))
- **Output**: \( y \) (vector of dimensions \( V \))

- \( d \)-dimensional vector (average of vectors of all input words)
  \[ \text{avg}(xW) = h \]

- Matrix \( W \)
- Matrix \( W' \)

- Softmax function
X 2 vec

- Generate vector representations (embeddings) for various data types
- Examples:
  - Word2Vec
  - Doc2Vec
  - Node2Vec
  - Item2Vec
  - Sent2Vec
  - Gene2Vec

Visualize: [https://projector.tensorflow.org/](https://projector.tensorflow.org/)

Explore: [http://epsilon-it.utu.fi/wv_demo/](http://epsilon-it.utu.fi/wv_demo/)
In vector space...
Word embeddings are time-dependent (why?)

- Semantic similarity of words depends on *time*.
Problems with word2vec

- Words with multiple meanings only have one representation
  - eg. bank of river or bank of money
  - Need contextual information
- Limited Context
  - only trained on words within the context window
Language Modelling History

2008
2013
2014
2016
2018
2019
2020
2022
2023

Legend:
- Before Transformer
- After Transformer

Word2Vec
N-grams
RNN / LSTM
Transformer
BERT
T5
GPT-3
PaLM
Bard / GPT-4

A bat flew out of the dugout, startling the baseball player and making him drop his bat.
Self-Attention

Each word pays attention to the words around it, focusing more on the words which provide important information about its meaning.

A bat flew out of the dugout, startling the baseball player and making him drop his bat.
Transformer Architecture

Encoder:
Computes contextual word embeddings from context.

Decoder:
Predicts the next word given context.
Self-Attention
Self-Attention

\[ s_2 = \frac{q_2 K^\top}{\sqrt{d}} \]

\[ \alpha_2 = \text{softmax}(s_2) \]

\[ x_2 = \sum_{i=1}^{4} \alpha_{2,i} v_i \]

Queries (Q) \hspace{2cm} Keys (K) \hspace{2cm} Values (V)

Input Sequence \hspace{8cm} Output Sequence
Discuss:

- Q, K, V are all \((n \times d)\) matrices.

- What is the shape of \(QK^T\)?
  - What does this matrix represent?

- What is the shape of the final output?
  - What does this matrix represent?

\[
Attention(Q, K, V) = \text{Softmax}(\frac{QK^T}{d_k})V
\]
Multi-Head Attention

What if I want to pay attention to different things at the same time!?

Content-based        This is my big red dog, Clifford.
Description-based     This is my big red dog, Clifford.
Reference-based       This is my big red dog, Clifford.

What’s useful depends on the task. How do I pick what to do?
Multi-Head Attention

- The Scaled Dot-Product Attention attends to one or few entries in the input key-value pairs.
- Idea: apply Scaled Dot-Product Attention multiple times on the linearly transformed inputs.

\[
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O
\]

where \( \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \)
Transformer Architecture

Encoder

- Positional Encoding
- Embedding
- xN
- Self-Attention
- Feed Forward

Decoder

- Positional Encoding
- Embedding
- xN
- Masked Self-Attention
- Cross-Attention
- Feed Forward

I like black coffee

Kaffee

Linear Classifier

<START> Ich mag schwarzen
Cross-Attention

\[ s_1 = \frac{q_1 K^T}{\sqrt{d}} \]
\[ \alpha_1 = \text{softmax}(s_1) \]
\[ x_1 = \sum_{i=1}^{d} \alpha_{1,i} V_i \]

Queries (Q) → Keys (K) → Values (V) → Output Sequence

Decoder Input

Encoder Output

Ich mag schwarzen Kaffee

Linear Classifier

Feed Forward

Self-Attention

Positional Encoding

I like black coffee

<START> Ich mag schwarzen
Transformer Architecture

Encoder
- xN
- Feed Forward
- Self-Attention
- Positional Encoding
- Embedding

I like black coffee

Decoder
- xN
- Feed Forward
- Cross-Attention
- Masked Self-Attention
- Positional Encoding
- Embedding

<KSTART> Ich mag schwarzen

Kaffee
- Linear Classifier
Self-Attention vs. Masked Self-Attention

In masked self-attention each word only pays attention to words from the past, but not the future.

**Quiz:** Why is this advantageous in the decoder?
Transformer Architecture

Encoder
- Positional Encoding
- Embedding
- Self-Attention
- Feed Forward

Decoder
- Positional Encoding
- Embedding
- Masked Self-Attention
- Cross-Attention
- Feed Forward

Kaffee
- Linear Classifier

I like black coffee

<START> Ich mag schwarzen
Point-wise Feed-forward Networks

- **Purpose**
  - Applies non-linear transformations to the output of the attention layer

- **Equation**
  - \( FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \)
    - where \( W \) and \( b \) are learned weights and biases
  - These FFN is applied separately to each position
  - Followed by Layer Normalization
Transformer Architecture

I like black coffee

<START> Ich mag schwarzen
Input Embeddings

- Replace tokens with continuous vectors
- Made up of two components:
  - token embeddings
  - positional encoding depends on position and dimension index as follows (next slide)
- Word order is important:
  - “Sally stole money from John”
  - “John stole money from Sally”
Positional Embedding

Encodes the position of each token in the sentence.

\[ PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]

\[ PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \]
Transformer Summary
Discuss:

- How does the transformer scale with sequence length?
  - Which part of the transformer scales the worst?
BERT
Encoder only

Encoder - Decoder

T5/BART

Decoder only

GPT

I like black coffee

I like black coffee

<START> Ich mag schwarzen
<START> Ich mag schwarzen
Scaling Laws

- Performance improves predictably with increased compute, data, and parameters
  - Can actually fit power laws!
  - Predict performance before training!

\[ C = C_0 N D \]
\[ L = \frac{A}{N^\alpha} + \frac{B}{D^\beta} + L_0 \]

Figure 1  Language modeling performance improves smoothly as we increase the model size, dataset size, and amount of compute\(^2\) used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.
Emergent Capabilities
Is Emergence a Mirage?

- Look at four digit addition under different metrics
  - Emergence can be an artefact of the evaluation metric
- Addition capabilities improve smoothly when using a more granular metric
  - Exact match accuracy -> Token edit distance