Machine Learning for Economists

Class 15: Transformers - Attention is All You Need

葛雷

中国人民大学 - 数量经济

2025 年 5 月 21 日

RENMIN UNIVERSITY OF CHINA

Why Large Language Models for Finance?

- Replace us to read and analyze tedious hundreds of thousands financial reports
- Then, extract useful information from these tons of documentations in no time
- Sentiment Analysis, Documents Classification, Information Extraction, Report Generation

What is LLM?

- LLM is Large Language Model
- ChatGPT,KIMI,Qwen,LlaMA, Bert, DeepSeek all use same transformer model
- Transformer from "Attention Is All You Need" (Vaswani et al. 2017) `https://arxiv.org/abs/1706.03762`

How Study LLM?

- LLM not hard if follow step by step
- Codes are convenient to adapt (Please follow me to classA3_bert_model.ipynb)
- Let us begin with some basics

LLM for Finance

Word Embedding

Transformer

Tools

Old Methods: Words Frequency

- How to use text information our finance model?
- How about using the words frequency
- The article with high frequency of positive words $\rightarrow$ positive news to the stock market
- So, we have TFIDF to count the **word frequency** (from sklearn.feature_extraction.text import TfidfVectorizer)

Old Methods: TF-IDF

TF-IDF (*Term Frequency-Inverse Document Frequency*) evaluates word importance in a document relative to a corpus. It combines:

1. Term Frequency (TF)

$$TF(t, d) = \frac{\text{Count of term } t \text{ in document } d}{\text{Total terms in } d}$$

2. Inverse Document Frequency (IDF)

$$IDF(t) = \log\left(\frac{\text{Total documents}}{\text{Documents containing } t + 1}\right)$$

TF-IDF Score

Final Calculation

$$\mathit{TF\text{-}IDF}(t, d) = \mathit{TF}(t, d) \times \mathit{IDF}(t)$$

- **High TF-IDF**: Frequent in document, rare in corpus (e.g., technical terms)
- **Low TF-IDF**: Common words (e.g., "the") or rare words with low TF

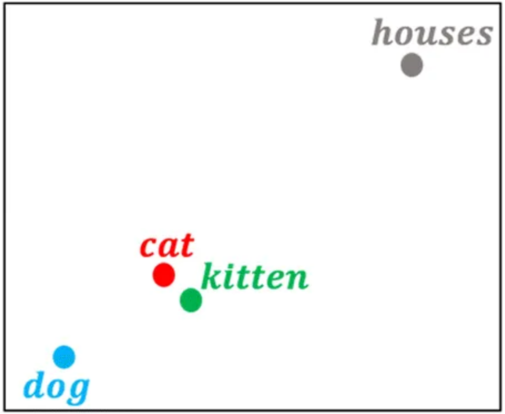
Old Methods are outdated

- However, the word frequency is outdated
- We need to change words to **Meaningful Vectors**
- But how? (Using Word Embedding)

Word Embedding

- Embedding: Word $\rightarrow$ Vector (with meaning)
- Word embeddings convert words into vectors representing their semantic meaning

Why Word Vectors?

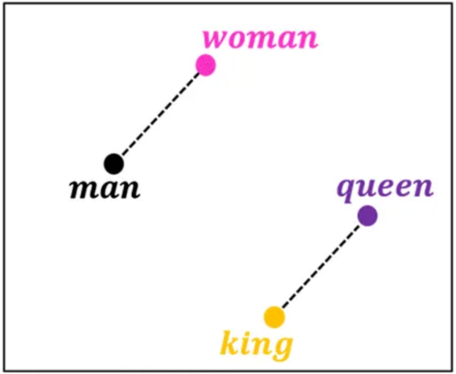


- similar meaning words near
- different meaning words far away

Example

	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

Why Word Vectors?



- Queen = King - Man + Woman
- Semantic meaning => math

Whole Sentence Embedding

We embedding: **a cat catches a mouse**

Word	Embedding Vector
a	[0.1, 0.2, 0.3]
cat	[0.4, 0.5, 0.6]
catches	[0.7, 0.8, 0.9]
a	[0.1, 0.2, 0.3]
mouse	[0.1, 0.3, 0.5]

Whole Sentence Embedding

We embedding: **a cat catches a mouse** in a matrix

$$W_{m,k} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 0.1 & 0.2 & 0.3 \\ 0.1 & 0.3 & 0.5 \end{bmatrix}$$

m is the number of words in the sentence, and k is the dimension of the embedding

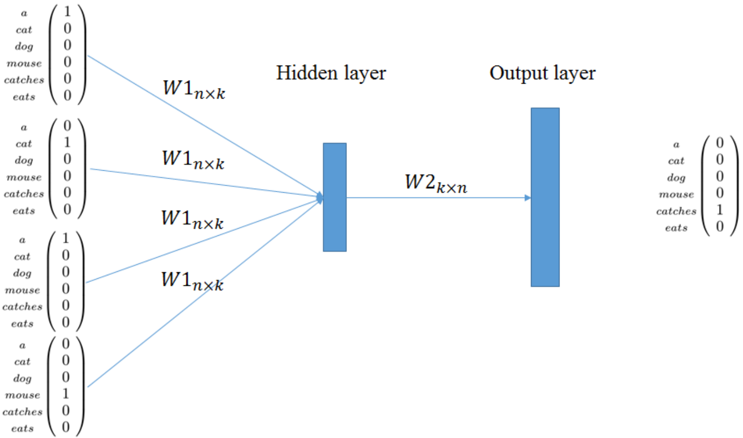
How to train a word embedding Model?

- Please follow me to the python codes for the Word2Vec Model
- Python package of Word2Vec is convenient to use
- How about the math of word embedding model ? Lets us jump into the deep water (math)

Word Embedding Training: Word2Vec model (Math)

- Predicts the target word (center word) from the context words (surrounding words).
- By using simple ANN with one hidden layer

Word Embedding Training: Word2Vec model (Math)



Word Embedding Training: Word2Vec model (Math)

- whole sentence: a cat catches a mouse
- predict "catches" by its nearby words "a","cat", "a", "mouse"

Word Embedding Training: Word2Vec model (Math)

- Step 1 Tokenization: one-hot encoding each word (so each word can be represented by $1 \times n$ dimension tensor)
- Step 2, Linear Transformation: matrix W1 has dimension $n \times k \rightarrow$ vector embedding for each word
- The Step 2 is called word embedding, which maps each n unique word to a k dimension vector

n is number of unique words in corpus, k is the dimension of the embedding vector

Word Embedding Training: Word2Vec model (Math)

- Step 3 Aggregation, sum up the embedding vectors for all nearby words "a","cat", "a", "mouse", and get one summed vector (k dimension)
- Step 4 Linear Transformation: matrix W2 has dimension $k \times n$, map k dimension vector back one n dimension vector
- Step 5 Softmax: apply a Softmax to the n dimension vector and output the probability of the word "catches"

n is number of unique words in corpus, k is the dimension of the embedding vector

Word Embedding Training: Word2Vec model (Math)

- The training of the Word2Vec need: all 5 steps to train the word vectors
- The word embedding process only need step1 and step2

keynotes of the Word Embedding

- It convert each word (token) into a unique vector by extracting it from the embedding matrix $W1$ (like a dictionary)
- This vector has the semantic meaning of the word

Problems of the Word Embedding

- It only consider each word independently
- It does not give the contextual meaning of the whole article
- 我爱你 and 你爱我 has same word vectors but different contextual meaning.
- So we need to add contextual meaning to each word vectors. Let's turn to Transformer

LLM for Finance

Word Embedding

Transformer

Tools

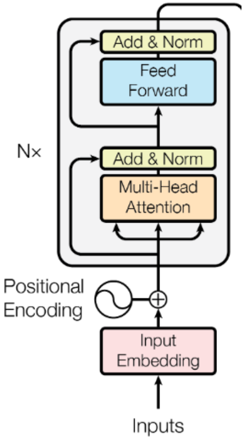
Transformer: new model

- From paper "Attention Is All You Need" by Vaswani et al. (2017)
- New framework, so even the Bible textbook "Deep Learning (2016)" by Goodfellow does not has it
- So, we should use new study material to catch up with the world

Transformer: key elements

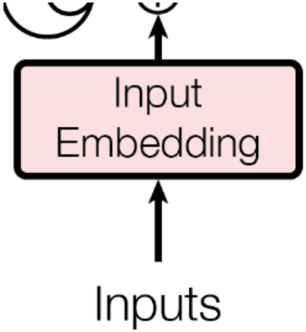
- Word Embedding: convert words (token) into vectors like before
- Positional Encoding: add position information to the word vectors
- Self-Attention Mechanism: add contextual information to the word vectors (Multiple-Heads Self-Attention)

Transformer: bird view



Word Embedding

Same as word embedding before, convert words (token) into vectors



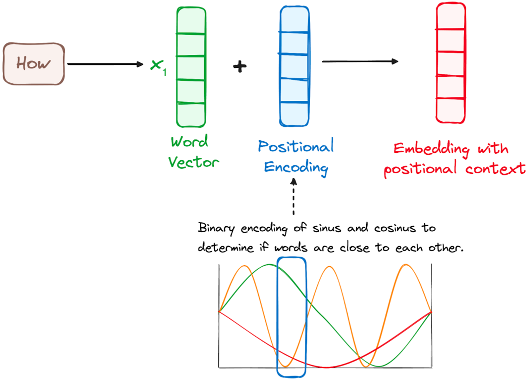
Positional Encoding

We add positional information to each word vectors

Positional Encoding

- The position of each word matters in the context
- Positional encoding add position information to the word vectors
- In practice, the positional encoding use a set of different shape $\sin(.)$ and $\cos(.)$ to add small values to the word embedding vectors

Positional Encoding for a Word



Positional Encoding (Math)

The positional encoding for a position *pos* and dimension *i* is:

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

where:

- *pos* is the position of the word in the sequence.
- *i* is used to indicate the position in word embedding vector
- *d* is the total embedding size (the dimensionality of the model).

Positional Encoding (Example)

Sentence: "Transformers are amazing" Word Embeddings:

$$\text{Transformers} \to [0.1, 0.2, 0.3, 0.4]$$
$$\text{are} \to [0.2, 0.3, 0.4, 0.5]$$
$$\text{amazing} \to [0.3, 0.4, 0.5, 0.6]$$

Positional Encoding (Example)

Word Embeddings + Positional Encoding:

Transformers (position 0) $\rightarrow [0.1 + 0.0, 0.2 + 1.0, 0.3 + 0.0, 0.4 + 1.0]$

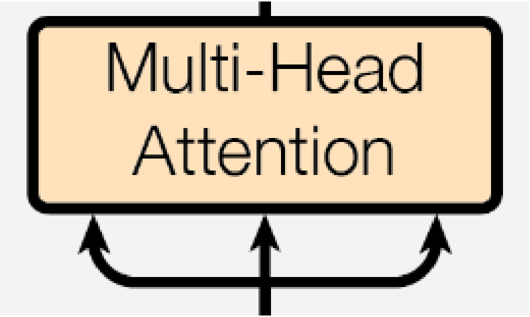
are (position 1) $\rightarrow [0.2 + 0.84, 0.3 + 1.0, 0.4 + 1.0, 0.5 + 1.0]$

amazing (position 2) $\rightarrow [0.3 + 0.91, 0.4 + 1.0, 0.5 + 1.0, 0.6 + 1.0]$

In reality, positional encoding additions are very small because total embedding size is very large

Self-Attention (!!!Heart of LLM!!!)

Self-Attention is the Heart of LLM



Self-Attention, why?

- Add contextualized meaning to the word vectors
- Why? Because one word's meaning depending on all other words
- example:" 你是个学霸 ". the contextual meaning of 霸 depending on all other words

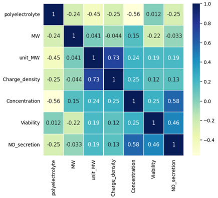
Self-Attention, how?

- However about we give a percentage attention weights represent 霸's dependence on other words
- 你 (10%) 是 (5%) 个 (5%) 学 (50%) 霸 (30%)
- 霸 with contextual meaning = original 霸 + 你 (10%) 是 (5%) 个 (5%) 学 (50%) 霸 (30%)

Bingo, this is self-attention

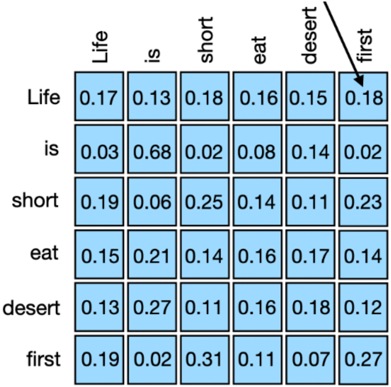
How to get attention weights?

- We want to get the attention weights for every words in the sequence
- We need represent the token wise correlation between all tokens
- Similar to features correlation matrix in **df.corr()**



Attention Weights Matrix

Attention Weights Matrix for sequence "Life is short eat dessert first":



	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

Self Attention (Q,K,V)

$$\text{Self Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

- X: is the word vectors matrix (input embeddings)
- Linear transformation of X into Query (Q), Key (K), and Value (V) matrices using learned weight matrices W_Q, W_K, and W_V

Just Linear Transformation

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

- They are just linear transformation in simple ANN model
- Why? Please think

Just Linear Transformation

- X matrix dimension (n_seq,n_emb), n_seq is n of sequence length, n_emb is n of the total embedding dimensionality
- W_Q, W_K, and W_V matrices dimension (n_emb, n_emb)
- So, $Q = XW_Q$ dimension (n_seq, n_emb). Same as K and V
- from X to Q, K, V, the dimension has no change

Attention Scores

$$\text{Attention Scores} = \frac{QK^T}{\sqrt{d_k}}$$

- QK^T dimension (n_seq,n_seq) $\rightarrow$ Words to Words attention dependence
- d_k is n_emb for scaling

Softmax(attention score)

$$\text{Attention Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

- softmax function change each row of attention score to percentage weight which adding up to 100%
- Attention Weights dimension (n_seq,n_seq)

Self-Attention = Attention Weights · Input Embedding

$$\text{Self-Attention Output} = \text{Attention Weights} \cdot V = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

- V representing the original word vectors (input embedding)
- Attention Weights give the dependence percentage weights of one token (word) to all other tokens
- Example: $\boxed{\text{霸 with contextual meaning}}$ = original 霸 + 你 (10%) 是 (5%) 个 (5%) 学 (50%) 霸 (30%)

Self-Attention Output

- Attention Weights dimension (n_seq,n_seq), V dimension (n_seq,n_emb), so Self-Attention Output dimension (n_seq,n_emb)
- Self-Attention Output and input sequence embedding X has same dimension

Self-Attention in one word

$$X_{contextualized} = Self\ Attention(X)$$

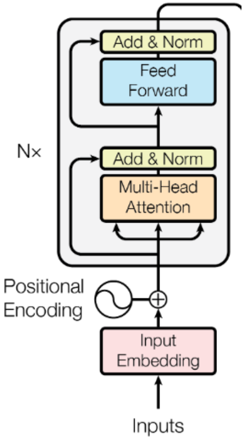
- Use attention weights to add other tokens vectors to one token vector
- from the sequence X to contextualized new $X_{contextualized}$, the embedding is still n_{seq} length words sequence and each word is a n_{emb} dimension vector

Look again our attention weight matrix

Attention Weights Matrix for sequence "Life is short eat dessert first":

	Life	is	short	eat	desert	first
Life	0.17	0.13	0.18	0.16	0.15	0.18
is	0.03	0.68	0.02	0.08	0.14	0.02
short	0.19	0.06	0.25	0.14	0.11	0.23
eat	0.15	0.21	0.14	0.16	0.17	0.14
desert	0.13	0.27	0.11	0.16	0.18	0.12
first	0.19	0.02	0.31	0.11	0.07	0.27

Whole Picture



LLM for Finance

Word Embedding

Transformer

Tools

Advanced Method plz use this way

- Huggingface Transformers (bert, llama,GLM)
- Pytorch

Easy Method

- Ollama (use model)
- Llama Factory (fine tuning)
- Anything LLM (RAG)

Next: Advanced Transformer

- Encoder Only Transformer: Bert
- Decoder Only Transformer: ChatGPT, LlaMA
- Fine-Tuning, RAG, P-tuning, Prefix-tuning

Reference

1. Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition)
2. Wikipedia
3. w3schools
4. geeksforgeeks
5. Kaggle
6. Wikipedia
7. ChatGPT
8. DeepSeek

Appendix: Dimension of Transformer Data is 3-D

python 复制

```python
transformer = nn.Transformer(d_model=512, batch_first=True)
src = torch.rand((32, 10, 512))  # (batch_size, seq_len, d_model)
```