Recent Advances in Language Model and Information Retrieval

Ph.D. Dang Tran Thai

Department of Natural Language Processing
Virtual Assistant Technology Division
VinBigdata

September 10, 2023

Outline

N-gram

The task of predicting the next word can be stated as attempting to estimate the probability function:

$$P(w_n|w_1, ..., w_{n-1}) \tag{1}$$

- *Perplexity* is one of the most common metrics for evaluating language models

$$PP(W) = P(w_1, w_2, ..., w_N)^{-\frac{1}{N}} = \sqrt[n]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, ..., w_{i-1})}} \tag{2}$$

- N-gram language model:
  - Unigram: $P(w_1, w_2, ..., w_n) \approx \prod_i P(w_i)$
  - Bigram: $P(w_1, w_2, ..., w_n) \approx \prod_i P(w_i|w_{i-1})$
  - Trigram: $P(w_1, w_2, ..., w_n) \approx \prod_i P(w_i|w_{i-1}, w_{i-2})$
  - Smoothing methods: Add-one (Laplace), backoff, linear interpolation, Good-Turing

Word2Vec

The Skip-gram Model: assumes that a word can be used to generate its surrounding words in a text sequence.

- For each word with index i in the dictionary, denote by $v_i \in \mathbb{R}^d$ and $u_i \in \mathbb{R}^d$ its two vectors when used as a *center* word and a *context* word, respectively. The conditional probability of generating any context word w_o given center word w_c is modeled as below:

$$P(w_o|w_c) = \frac{\exp(u_o^\intercal.v_c)}{\sum_{i\in V}\exp(u_i^\intercal.v_c)} \tag{3}$$

- The likelihood function of the skip-gram model is the probability of generating all context words given any center word:

$$\prod_{t=1}^{T} \prod_{-m\leq j\leq m, j\neq 0} P(w^{(t+j)}|w^{(t)}) \tag{4}$$

where T is the sequence length, m is the context window size.

Word2Vec

The Continuous Bag of Words (CBOW): assumes that a center word is generated based on its surrounding context words in the text sequence.

- Since there are multiple context words, these context word vectors are averaged in the calculation of the conditional probability.
- For any word with index i in the dictionary, denote by $v_i \in \mathbb{R}^d$ and $u_i \in \mathbb{R}$ its two vectors when used a *context* and word and *center* word, respectively. The conditional probability of generating any center word w_c given its surrounding context words $w_{o_1}, ..., w_{o_{2m}}$ can be modeled as below:

$$P(w_c | w_{o_1}, ..., w_{o_{2m}}) = \frac{\exp(\frac{1}{2m} u_c^\intercal (v_{o_1} + ... + v_{o_{2m}}))}{\sum_{i \in V} \exp(\frac{1}{2m} u_i^\intercal (v_{o_1} + ... + v_{o_{2m}}))} \quad (5)$$

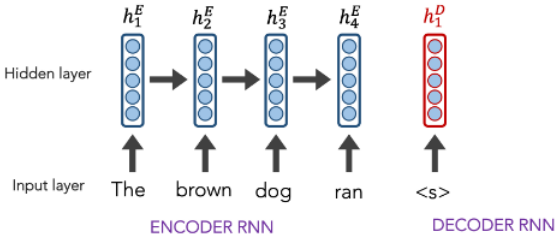
- The likelihood function of the CBOW:

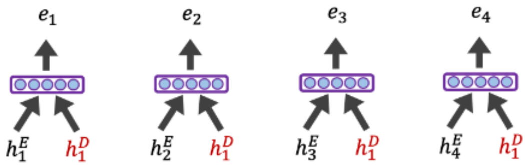
$$\prod_{t=1}^{T} P(w^{(t)} | w^{(t-m)}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+m)}) \quad (6)$$

Attention Mechanism

Seq2Seq + Attention

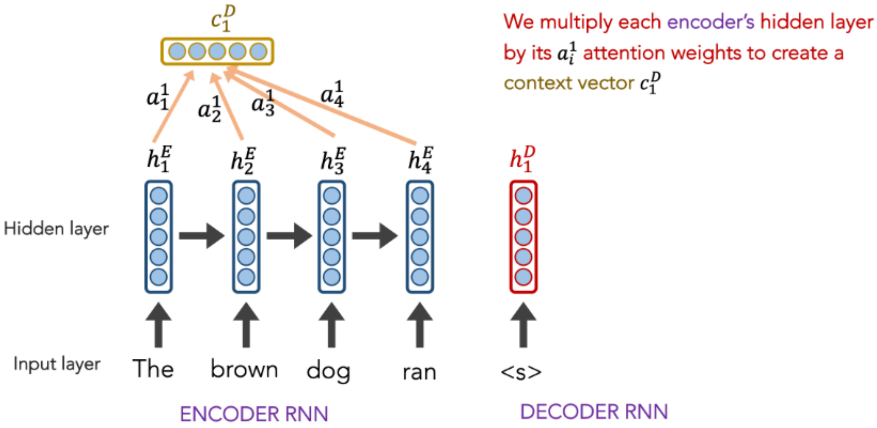
- Input sequence X, encoder f_{enc}, decoder f_{dec}
- $f_{enc}(X)$ produces hidden states $h_1^E, ..., h_N^E$
- On time step t, have decoder hidden state h_t^D
- Attention score: $e_i = h_t^{D\intercal} h_i^E$
- Attention distribution: $a_i^t = \frac{\exp(e_i)}{\sum_i^N \exp(e_i)}$





Attention Mechanism

Seq2Seq + Attention



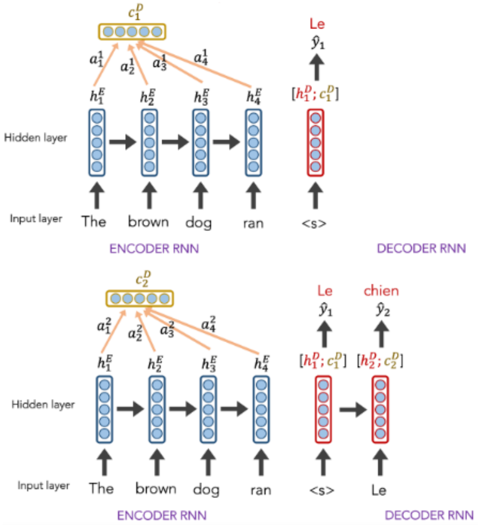
We multiply each encoder's hidden layer by its a_i^1 attention weights to create a context vector c_1^D

Attention Mechanism

Seq2Seq + Attention

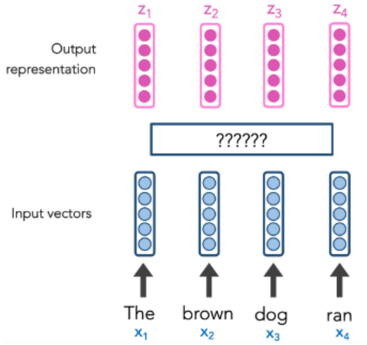
Sample output $\hat{y}_t$ using both h_t^D and c_t^D:

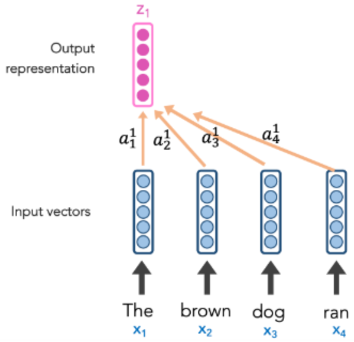
$$\hat{y}_t \sim f_{dec}(h_t^D, c_t^D, \theta) \qquad (7)$$



Attention Mechanism

Self-Attention





- Self-Attention's goal is to create great representations, z_i of the input
- z_i is based on a weighted contribution of each token in the input

Attention Mechanism

Self-Attention

- Given the input sequence: $x_1, x_2, ..., x_n$
- Each word x_i has 3 associated vectors: *Query vector* q_i, *Key vector* k_i, *Value vector* v_i:

$$q_i = w_q x_i$$
$$k_i = w_k x_i$$
$$v_i = w_v x_i$$

- For word x_i, let's calculate the scores $s_i^1, ..., s_i^n$ which represent how much attention to pay to each respective v_i:

$$s_i^j = q_i k_j \tag{8}$$

  where $j = 1, ..., n$
- Let's divide s_i^j by $\sqrt{d}$ where d is the dimension of k_i and softmax it:

$$a_i^j = softmax(\frac{s_i^j}{\sqrt{d}}) \tag{9}$$

Attention Mechanism

Self-Attention

- Compute z_i as the following:

$$z_i = \sum_{j=1}^{n} a_i^j v_j \tag{10}$$

Self-Attention is powerful that allows to create context-aware representations.

Transformer

- The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$
- Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time.
- Encoder: a stack of 6 identical layers. Each layer contains: multi-head self-attention; position-wise fully connected feed-forward network.
- Decoder: a stack of 6 identical layers. Each layer also contains sub-layers as encoder with adding a multi-head attention over the output of the encoder stack.

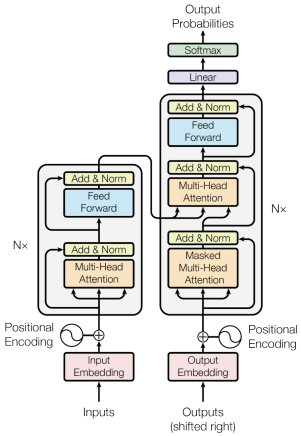


Figure: The Transformer - model architecture

Transformer

Attention

- Scaled Dot-Product Attention

$$Attention(Q, K, V) = softmax(\frac{QK^\intercal}{\sqrt{d_k}})V \tag{11}$$

  where d_k is the dimension of query and key vectors
- Multi-Head Attention: Linearly project queries, keys, and values h times:

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, .., head_h)W^O \\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{12}$$

Positional Encoding

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{13}$$

Bidirectional Encoder Representation from Transformers (BERT)

- **Model Architecture:** A multi-layer bidirectional Transformer encoder
- **Tokenization**: WordPiece with 30,000 token vocabulary. The first token of every sequence is always a special classification token $[CLS]$. Sentences are separated by a special token $[SEP]$
- **Pre-training BERT**:
  - Masked LM: mask 15% of all WordPiece tokens in each sequence at random
  - Next sentence Prediction: Choosing the sentences A and B for each pre-training example, 50% of the time B is actual next sentence that follow A (labeled as IsNext). 50% of the time it is a radom sentence from corpus (labeled as NotNext)

Generative Pre-training Language Model (GPT)

Training procedure[1]:

- **Unsupervised pre-training**: Given $U = \{u_1, ..., u_n\}$, estimate:

$$L_1(U) = \sum_i logP(u_i | u_{i-k}, ..., u_{i-1}; \theta) \tag{14}$$

  In experiments, use **Transformer decoder**.

- **Supervised fine-tuning**: $C = (\{x^1, ..., x^m\}, y)$, estimate:

$$L_2(C) = \sum_{(x,y)} logP(y | x^1, ..., x^m) \tag{15}$$

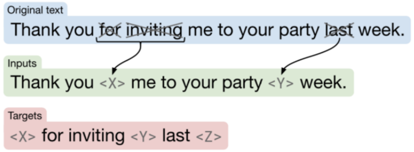
- **Task-specific input transformations**: Converting text pair data (e.g., textual entailment, similarity, question answering) into ordered sequence then feed to pre-trained model

[1]Improving Language Understanding by Generative Pre-training

Text-to-Text Transfer Transformer (T5)

Paper: *"Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer"*[2]

- Use Encoder-Decoder Transformer that includes:
  - Self-attention in encoder blocks and decoder blocks
  - Attention between encoder and decoder
- Unsupervised objective training: "denoising" objective, a.k.a, "mask language modeling"



Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as **<X>** and **<Y>**) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel **<X>**. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token **<Z>**.

[2] https://arxiv.org/pdf/1910.10683.pdf

Text-to-Text Transfer Transformer (T5)

Fine-tuning with downstream tasks

- The basic idea is to treat every text processing problem as a “text-to-text”problem
- Tasks:
  - Sentence acceptability judgment
  - Sentiment analysis
  - Paraphrasing/sentence similarity
  - Natural language inference
  - Coreference resolution
  - Sentence completion
  - Word sense disambiguation
  - Question answering
- Flan-T5: finetuning language model with a focus on: (1) scaling the number of tasks, a.k.a, multi-task instruction finetuning; (2) scaling the model size; and (3) finetuning on chain-of-thought data to improve model performance and generalization to unseen tasks.

ChatGPT

- The relevance between the generated text and the instruction is controlled by the *self-attention mechanism of the Transformer decoder*:
- Training steps [3]:
  - **Supervised fine-tuning (SFT):** a pre-trained language model is fine-tuned on a relatively small amount of demonstration data. This represents the baseline model.
  - **"Mimic human preferences" by reward model (RM):** ask labelers to vote a large number of SFT model outputs ($\times 10$ times bigger than data SFT) to make comparison data. The voting is based on the information certainty and the response coherence. Comparison data is used for training RM.
  - **Proximal Policy Optimization (PPO):** The reward model is used to further fine-tune and improve SFT model, is called policy model
- The training steps is repeated many times

[3] https://www.assemblyai.com/blog/how-chatgpt-actually-works/

Direct Preference Optimization (DPO)

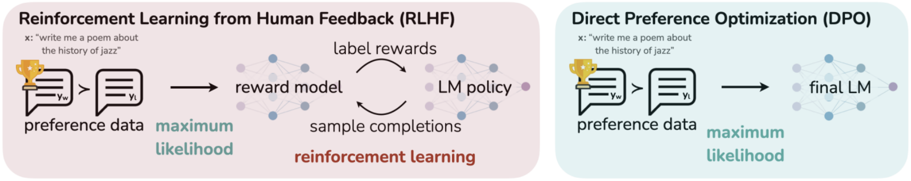
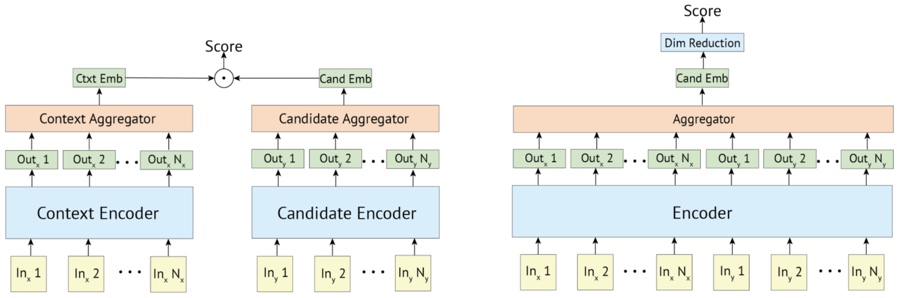

Reinforcement Learning from Human Feedback (RLHF)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
reward model
label rewards
LM policy
sample completions
reinforcement learning
Direct Preference Optimization (DPO)
x: "write me a poem about the history of jazz"
y_w
y_l
preference data
maximum likelihood
final LM

Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, without an explicit reward function or RL.

Dense Retrieval



(a) Bi-encoder **(b) Cross-encoder**

Dense Retrieval

Dense Passage Retriever

- Define the similarity between the question and the text passage using dot product of their vectors

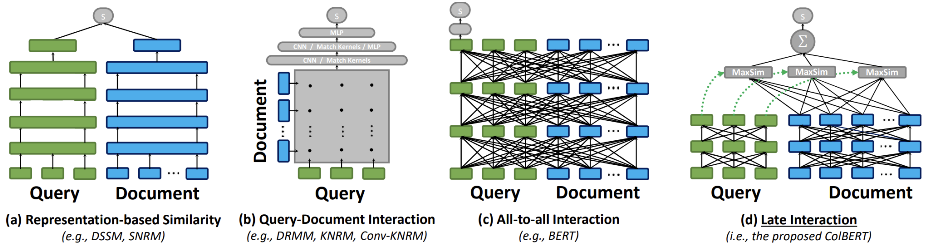
$$sim(q,p) = E_Q(q)^\intercal E_P(p) \tag{16}$$

where q is the question and $E_Q(x)$ is the question encoder that map input question to d-dimensional vector. The text passage p with its encoder $E_P(x)$

- Training: Let $\mathcal{D} = \{\langle q_i, p_i^+, p_{i,1}^-, ..., p_{i,n}^- \rangle\}_1^m$ be the training dataset containing m instances. In each instance, q_i this the input question, p_i^+ is one relevant positive passage along with n irrelevant passages $p_{i,j}^-$. The training loss function is as below:

$$L(q_i, p_i^+, p_{i,1}^-, ..., p_{i,n}^-) = -\log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + \sum_{j=1}^n e^{sim(q_i, p_{i,j}^-)}} \tag{17}$$

ColBERT



S
Query
Document
(a) Representation-based Similarity
(e.g., DSSM, SNRM)
MLP
CNN / Match Kernels / MLP
CNN / Match Kernels
Document
Query
(b) Query-Document Interaction
(e.g., DRMM, KNRM, Conv-KNRM)
Query
Document
(c) All-to-all Interaction
(e.g., BERT)
Σ
MaxSim
MaxSim
MaxSim
Query
Document
(d) Late Interaction
(i.e., the proposed ColBERT)

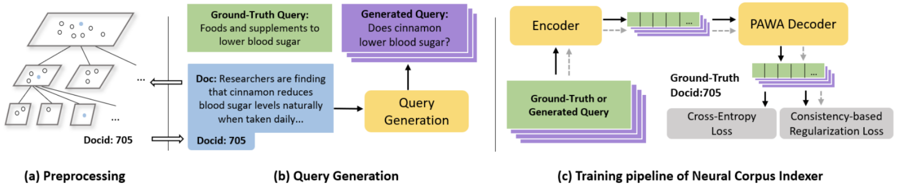
Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

Late Interaction: Given query q and a document d, the relevance score of d to q, denoted as $S_{q,d}$ is estimated as below:

$$S_{q,d} := \sum_{i \in [|E_q|]} \max_{j \in [|E_d|]} E_{q_i} . E_{d_j}^{\intercal} \tag{18}$$

where E_q and E_d are embedding of the query and document,

A Neural Corpus Indexer for Document Retrieval (NCI)



Ground-Truth Query: Foods and supplements to lower blood sugar
Generated Query: Does cinnamon lower blood sugar?
Doc: Researchers are finding that cinnamon reduces blood sugar levels naturally when taken daily...
Docid: 705
Docid: 705
Query Generation
Encoder
PAWA Decoder
Ground-Truth or Generated Query
Ground-Truth Docid:705
Cross-Entropy Loss
Consistency-based Regularization Loss
(a) Preprocessing
(b) Query Generation
(c) Training pipeline of Neural Corpus Indexer

Figure 1: Overview of Neural Corpus Indexer (NCI). (a) Preprocessing. Each document is represented by a semantic identifier via hierarchical k-means. (b) Query Generation. Queries are generated for each document based on the content. (c) The training pipeline of NCI. The model is trained over augmented <*query, docid*> pairs through a standard transformer encoder and the proposed Prefix-Aware Weight-Adaptive (PAWA) Decoder.