summary

inverted index

inverted index

- to improve search performance
- stores statistics
- bag of words: order is not important
- document data:
    - ids, metadata
    - doc length
    - avg doc length
- term data:
    - term frequency
    - document frequency
- dictionary:
    - posting list:
        - consists of `[doc-id:freq]` pairs for each term
        - doesn't store values with 0 freq

generating inverted index

- i) register metadata, assign doc id
- ii) tokenization
    - split on whitespace, punctuation character
    - keep abbreviations, names, numbers together
- iii) case folding
    - lowercase everything
- iv) stemming
    - reduce terms to 'root' form
    - ie. {changing, changed, change} $\mapsto$ chang
    - sometimes also lemmatization: {am, are, is} $\mapsto$ be
- v) filter stop words
    - ie. 'a', 'the', 'is', 'are'
- vi) add to dictionary, update posting list

querying inverted index

- find most relevant doc ids based on a scoring model (sum of scores for each query term)
- spell checking: can suggest different queries
- lookup on: hash table, prefix tree, b tree, …

scoring models

scoring model

- $\text{score}(q, d)$ = relevance score
  - q query term
  - d document

tf-idf

- https://en.wikipedia.org/wiki/Tf-idf
- weights used as basis for other methods like vector space model VSM
- $TF\_IDF(q, d) = w_{t,d} = tf_{t,d} \cdot idf_t$
- consists of 2 parts
- term frequency:
  - intuition: increases with the number of occurrences within a document, but logistically
  - $tf_{t,d} = \log(1 + f_{t,d})$
  - raw term frequency in this doc
- inverse document frequency:
  - intuition: the more docs contain the term, the less significant it is
  - $idf_t = \log(\frac{|D|}{df_t})$
  - df_t = num of docs with this term

bm25

- $BM25(q, d) = \sum_{t \in T_d \cap T_q} \frac{tf_{t,d}}{k_1 \cdot ((1-b) + b \cdot \frac{dl_d}{avgdl}) + tf_{t,d}} \cdot \log\left(\frac{|D| - df_t + 0.5}{df_t + 0.5}\right)$
- improves upon tf-idf: more saturated than logarithm as the term frequency increases
- variables:
  - $tf_{t,d}$ = term frequency
  - $|D|$ = num of all documents
  - df_t = document frequency of term = number of documents containing term
  - dl_d = doc length
  - $avgdl_d$ = average doc length
- hyperparameters:
  - k_1 = term frequency scaling
  - b = document length normalization

bm25f

- $BM25F(q, d) = \sum_{t \in T_d \cap T_q} \frac{\widetilde{tf}_{t,d}}{k_1 + \widetilde{tf}_{t,d}} \cdot \log\left(\frac{|D| - df_t + 0.5}{df_t + 0.5}\right)$
- $\widetilde{tf}_{t,d} = \sum_{s=1}^{s_d} w_s \cdot \frac{tf_{t,s}}{(1 - b_s) + b_s \cdot \frac{sl_s}{avgsl}}$
- improves upon bm25: can weigh document segments ie. title, abstract, body
- each segment is called a 'stream'
- new variables:
  - sl_s = stream length
  - w_s = stream weight
  - $avgsl$ = average stream length (for that doc index)

evaluation

we want IR models to be effective, efficient (fast, scalable), interpretable.

online vs. offline eval

- online: observing user behavior in production system
  - ie. user study through A/B testing
- offline: prepared dataset
  - documents, queries, judgements (= test data, expected query results)

test collections

- public test-collections: msmarco (sampled), trec (handcrafted), …
- result differences between sparse and dense judgements aren't too large
- consist of:
  - documents
  - queries
  - judgements:
    - binary labels (relevant vs. not relevant) - vs. - graded labels (score usually between 0;3)
    - sparse / dense
    - implicit feedback / explicit feedback

metrics

see: https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)

- precision P:
  - intuition: correctness
  - $P = TP / (TP + FP)$
- recall R:
  - intuition: completeness
  - $R = TP / (TP + FN)$
- mean average precision MAP: (binary labels)
  - intuition: sum of relative rel-doc positions / total num of rel-docs
  - also measures area under precision-recall-curve, hard to interpret
  - $MAP(Q) = \frac{1}{|Q|} * \sum_{q \in Q} \frac{\sum_{i=1}^{k} P(q)_{@i} \cdot rel(q)_i}{|rel(q)|}$
  - $P(q)_{@i}$ = precision metric at $i \rightarrow$ can be interpreted as position of rel-docs among other rel-docs
  - $rel(q)_i$ = relevance (binary)
  - $|rel(q)_i|$ = number of rel-docs in result

- mean reciprocal rank MRR: (binary labels)
  - intuition: 1 / absolute position of first rel-doc
  - $MRR(Q) = \frac{1}{|Q|} * \sum_{q \in Q} \frac{1}{FirstRank(q)}$
  - $FirstRank$ = position of first rel-doc
- discounted cumulative gain DCG: (graded labels)
  - intuition: relevance score / logarithm of absolute position for each doc
  - $DCG(D) = \sum_{d \in D,\, i=1} \frac{rel(d)}{\log(i+1)}$
  - $rel(d)$ = relevance of doc for given query

- i = absolute position in ranking
- discounted relevance means it normalizes score based on rank

- normalized discounted cumulative gain nDCG: (graded labels)
  - intuition: current dcg divided by the dcg of correctly sorted rel-docs
  - $nDCG(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{DCG(q)}{DCG(\text{sorted}(rel(q))}$
  - normalizes dcg again by best possible ranking per query ($DCG(\text{sorted}(rel(q)))$) - meaning the docs being in the correct order based on their relevance for the given query

statistical significance

- goal: proving that difference in two systems isn't by chance
- set a significance level / p-value first
- ie. 5% p-level means there is a 5% chance that the result is just by chance
- types:
  - paired: paired student's t-test, wilcoxson signed-rank test, …
  - non-paired: student's t-test, mann-whitney u test, …

creating test collections

see: https://ir-datasets.com/

- i. create k-cutoff-set of documents (= pooling-process)
  - use many diverse existing models to reduce work / number of initial documents to annotate
  - problem: lowers recall
- ii. let people annotate doc-query pairs
  - usually crowdsourced
  - majority voting for quality assurance = inter-annotator agreement (iaa)
- iii. create a model
  - test model on full dataset and if it retrieves docs outside the k-cutoff-set, then assume that they're false positives
  - be aware of bias:
    - biased dataset → biased results, overfit models
    - word embeddings are trained on biased data from wikipedia
    - gender, racial, lingual, term-position bias
    - you can de-bias data

word representations

n-gram

- word-n-gram = looking at n words at a time
- char-n-gram = looking at n characters at a time

word embeddings

- https://jalammar.github.io/illustrated-word2vec/
- unsupervised learning of vector representation for words
- each float in vector is a weight, describing the relationship between words
- we can do maths like cosine similarity
- word2vec:
  - 1-word-1-vector
  - predictive method (neural networks)
  - variations:
    - skip-gram = predict context from word

- cbow = predict word from context
  - trained against actual context with a sliding window
  - architecture:
    - input: 1-hot encoding of entire vocabulary
    - hidden: embeddings with trained weights
    - output: 1-hot encoding of entire vocabulary → softmax probability of each word being in the context of the input = $p(w_{t+j} \mid w_t)$
    - usually we ignore the output and just store the embeddings
- fastText:
  - 1-word-1-vector with char-n-grams
  - based on word2vec
  - using subwords (or char ngrams) as vocabulary to handle out-of-vocabulary words that are combinations of known words
  - massively improves performance because low-frequency terms are very important
- bert:
  - context dependent
  - transformer

character embeddings

- input: 1-hot encoding of characters
- output: char-n-gram
- doesn't limit you to a vocabulary

query expansion

- add related words to query based on nearest embeddings
- topic-shifting = neighbours of embedding might not make sense because they're from a different topic
- retrofitting = use more data to tune embeddings with unsupervised learning (not common anymore)
- lsi = find latent (hidden) semantic structure in text

CNN for text processing

- 1d cnn = one-dimensional convolutional neural networks for nlp, work though a sliding window
- dimensionality reduction = reading multiple embeddings (n-gram), returning a single embedding that captures the context
- filter parameters are learned
- pooling kernel types:
  - max-pooling = keep strongest feature per region
  - avg-pooling = keep average over each feature per region
  - adaptive/dynamic-pooling = change window size of pool (ie. based on sentence length so output is always the same size)

sequence models

= models for sequential data, where order matters

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://colah.github.io/posts/2015-09-NN-Types-FP/
- https://karpathy.github.io/2015/05/21/rnn-effectiveness/
- https://www.youtube.com/watch?v=zxagGtF9MeU&list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1

rnn

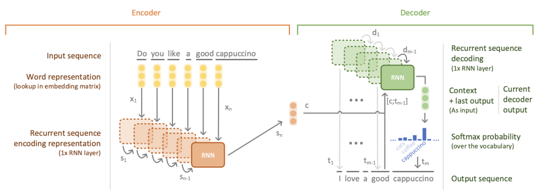
- rnn = recurrent neural network
- vanishing gradient problem: unrolling means adding exponent to weight, leading to information loss
- can be used bidirectionally or hierarchically (multi-layer/stacked)
- $s_i = R_{SRNN}(x_i, s_{i-1}) = \quad g(s_{i-1} \cdot W^s + x_i \cdot W^x + b)$
  - s_i = state at position i (depends on position s_{i-1}, recursive definition)

- g = activation function
- b = bias vector (trainable)
- W^s, W^x = weight matrices (trainable)

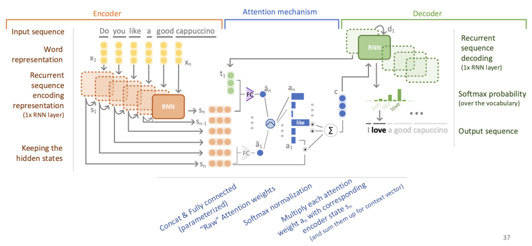
lstm

- lstm = long short term memory
- helps with vanishing gradient problem
- binary-gate is not differentiable, we have to use sigmoid function $\sigma(g')$ to map it to range [0;1]
- $s_j = R_{LSTM}(x_j, s_{j-1}) = \quad [c_j; h_j]$
  - $c_j = f \odot c_{j-1} + i \odot z \longrightarrow$ gated memory
  - $h_j = o \odot \tanh(c_j) \longrightarrow$ hidden state
    - $i = \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \longrightarrow$ input
    - $f = \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \longrightarrow$ forget
    - $o = \sigma(x_j W^{xo} + h_{j-1} W^{hz}) \longrightarrow$ output
    - $z = \tanh(x_j W^{xz} + h_{j-1} W^{hz}) \longrightarrow$ update candidate
  - $\odot$ = hedamard-product, element-wise multiplication
  - $[a;b]$ = `concat(a,b)`

seq2seq



seq2seq + attention



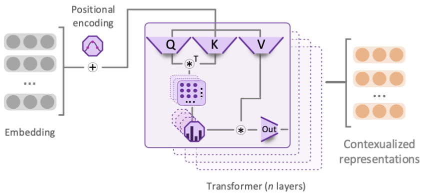
- encoder/decoder architecture used for translation, question answering, chat bots
- i. encoder:
  - $c = RNN_{Enc}(x_{1:n})$
  - get encoding / representation of entire input sequence $x_{1:n}$ as a single embedding
  - for unknown reasons, reversing input works best
- ii. attention mechanism:
  - input: current encoder state, last state from decoder
  - $attend(s_{1:n}, t_j) = c^j = \sum_{i=1}^{n} a^j_{[i]} \cdot s_i$
    - $a^j = \text{softmax}(\bar{a}^j_{[1]}, \ \ldots, \ \bar{a}^j_{[n]}) \longrightarrow$ attention weights sum up to 1
    - $\bar{a}^j_{[i]} = \nu \cdot \tanh([t_j; s_i] \cdot U + b) \longrightarrow$ implemented as neural network
    - j = current rnn iteration number
    - $s_{1:n}$ = all encoder states so far
    - t_j = decoder state at iteration j
    - c^j = context vector at position j
    - $a_{[i]}$ = `a[i]`
    - $[a;b]$ = `concat(a,b)`
    - ν, U, b = learnable params
  - improves encoder output c
  - generates weights (context vector) based on significance of each element
- iii. decoder:
  - $p(t_v \mid t_{1:v-1}) = \text{softmax}(RNN_{Dec}([t_{1:v-1}; \ c]))$

- $p(t_v \mid t_{1:v-1}) = \text{softmax}(RNN_{Dec}(attend(t_{v-1}, s_{1:n})))$ ⟶ alternative with attention mechanism
    - use embedding form previous stage as argument
    - get softmax probability of most likely output sequence $T,\ t_{1:m}$ as $\arg_T \max(p(T|x_{1:n}))$
    - use beam-search to pick highest softmax output → heuristic graph-search-algorithm based on softmax distribution (works better than greedy-search)

pointer generator

- out-of-vocabulary words are replaced with "UNK" by seq2seq
- then using a model they're reconstructed:
    - a) copying
    - b) generating

transformers



- = attention without LSTM, but with matrix-multiplications instead
- transformer types: encoder-only, encoder-decoder
- sequence contextualization = combine multiple surrounding word embeddings (self-attention) without a fixed window size (multi-head)
    - this is the current bottleneck that takes $O(n^2)$
- positional encoding = turning each token (subword) embedding into Q query, K key, V value - where key has d_k dimensions
- $\text{SelfAttention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \cdot V$

bert

- bert = bidirectional encoder representations from transformers
- improvement over word2vec
- limited to <512 tokens
- huge, pretrained, uses multiple layers of stacked transformers, fine-tuned for different use-cases
- masked language modeling:
    - the task to generate the embeddings (we ignore the output) is guessing a masked word from the context by returning a softmax probability over entire vocabulary
    - you can reduce vocabulary size by splitting words up into tokens with the wordPiece or bytePair algorithm
- special tokens:
    - CLS = start of 1-2 segments in sentence
    - MASK = masked word to predict
    - SEP = end of segment in sentence
- huggingface:
    - initially started as tensorflow to pytorch port of BERT
    - share models, datasets via git-lfs

extractive q&a

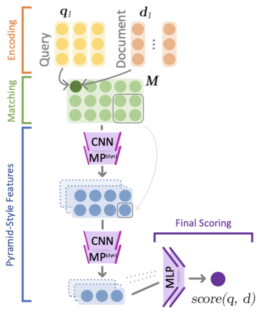
- ≠ generative question answering, chat bots
- find sequences in text that answer question
- can be done with segment start/end token predictions of BERT
- open domain qa = information retrieval + question answering → first retrieve the relevant documents, then find segments that answer question

neural re-ranking

we're looking at neural re-ranking models for content-based ad-hoc retrieval – but there are more efficient ways to re-rank.

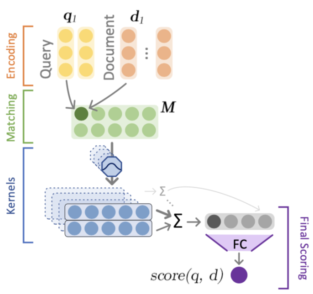
- = learning-to-rank system
- i. first-stage ranker:
    - content-based ad-hoc retrieval (= just use query and document content for ranking)
    - example: bm25@1000
- ii. neural re-ranker:
    - improve ranking order:
        - input triples: (query, relevant doc, non-relevant doc) → hard to find truly non-relevant documents, false negatives confuse the model
        - loss: maximize margin between rel/non-rel doc → $L(Q, P^+, P^-) = MSE(Ms(Q, P^+) - Ms(Q, P^-), M_t(Q, P^+) - Mt(Q, P^-))$
    - online learning: use previous user activity-logs to tune model
    - example: mrr@10

matchPyramid



- i. compute match matrix:
    - cosine-similarity for all query-doc-combinations
    - measures direction of vectors, but not the magnitude
    - $M_{ij} = \cos(q_i, d_j) = \frac{d_j \cdot q_i}{|d_j||q_i|}$
- ii. apply 2D convolution layers on matrix:
    - layers:
        - i. — $z_{ij}^{(1,c)} = \text{2D\_Conv}(M_{ij}) = ReLU\left(\sum_{s=0}^{r_c-1}\sum_{t=0}^{r_c-1} w_{s,t}^{(1,c)} \cdot M_{i+s,j+t} + b^{(1,c)}\right)$
        - ii. — $z_{ij}^{(2,c)} = \text{dyn\_max\_pool}\left(z_{ij}^{(1,c)}\right) = \max_{0 \leq s < d_c} \ \max_{0 \leq t < d_c} z_{i \cdot d_c + s, j \cdot d_c + t}^{(1,c)} \longrightarrow$ makes output size static
        - $\dots$ other kernels, each learning a different feature
        - L. — $z_{ij}^{(l,c)} = \text{max\_pool}\left(\text{2D\_Conv}(z_{ij}^{(l-1)})\right)$
        - $\text{score(q,d)} = MLP(z^l) = W_2 \cdot ReLU(W_1 \cdot z^l + b_1) + b_2 \longrightarrow$ neural-net returns float as final score
    - where:
        - $z^{(n,\_)}$ = sequential variable
        - c = channels
        - d_c = dynamic pool kernel size
        - r_c = channel size
        - W_*, b_* = weights, biases

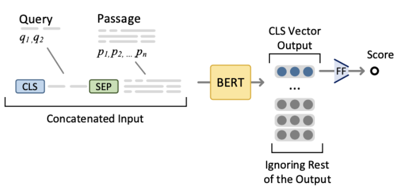
(c)knrm



- knrm = kernel based neural ranking model
- roughly as effective as matchPyramid but a lot faster
- counts similarities

- i. encode docs and queries: (only in conv-krnm variant)
    - apply cnn-kernels to encode multiple words into a single embedding (n-gram embedding)
    - $q^h_{1...n} == 1D\_CNN(q_{1...n})$
    - $d^h_{1...m} = 1D\_CNN(d_{1...m})$
    - where:
        - h = n-gram size
- ii. compute match matrix:
    - $M_{ij} = \cos(q_i, d_j) = \frac{d_j \cdot q_i}{|d_j||q_i|}$
- iii. apply radial-basis-function kernel:
    - $K_k(M) = \sum_{i=1}^{n} \log\left(\sum_j \exp\left(-\frac{(M_{ij}-\mu_k)^2}{2\sigma_k^2}\right)\right) \longrightarrow$ rbf kernel for a single match, summed along document dimension
    - $s = FC(K) = W \cdot K + b$
    - where:
        - K = all kernels
        - μ_k = similarity level
        - σ_k = kernel width / range
        - FC = fully connected neural network
        - W_*, b_* = weights, biases

monobert / bertcat

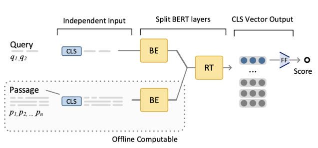


- state-of-the-art by using transformers (since 2018)
- i. concatenate input with special tokens:
    - this needs to be done for every single passage to compute similarity
    - $r = \text{BERT}([\texttt{CLS}]; q_{1..n}; [\texttt{SEP}]; p_{1..m})_{\texttt{CLS}}$
    - where:
        - q = query tokens
        - p = passage tokens
        - $[a; b]$ = `concat(a,b)`
- ii. apply linear layer:
    - get score
    - $s = r \cdot W$

improvements:

- input size: sliding window
    - bert is limited to <512 input tokens (query + document)
- efficiency: using a simpler model
- accuracy: mono-duo implementation
    - mono-phase: compute $\text{score}(q, p)$@1000
    - duo-phase: compute $\text{score}(q, p1, p2)$@50 but $50^2 = 2.500$ times $\longrightarrow$ improves results from mono stage

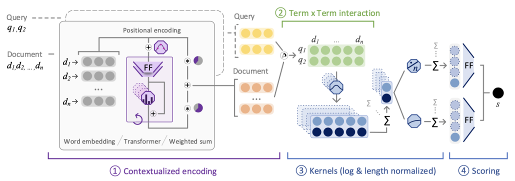
preTTR, colBERT - precomputed embeddings



- impove performance of monobert by merging precomputed embeddings instead of tokens

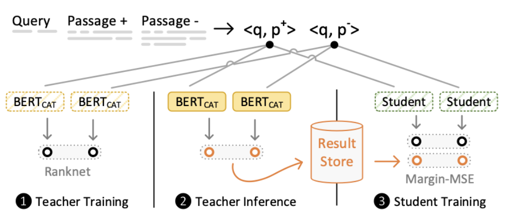
- similar accuracy
- i. precompute embeddings for all passages and common queries
  - $\hat{q}_{1..n} = \text{BERT}_{1..b}([\texttt{CLS}]; q_{1..n})$
  - $\hat{p}_{1..m} = \text{BERT}_{1..b}([\texttt{CLS}]; p_{1..m})$
- ii.a. combine: preTTR
  - $s = W \cdot \text{BERT}_{b..l}(\hat{q}_{1..n}[\texttt{SEP}]; \hat{p}_{1..m})$
- ii.b. combine: colBert
  - max-pool a match-matrix of query-passage-terms
  - $s = \sum_{i=1..n} \max_{t=1...m} \hat{q}_i \cdot \hat{p}_t$

tk



- tk = transformer kernel ranking
- tkl = tk for long documents
- simple and fast
- can be faster removing stopwords from input to use less compute, through a seperate module
- i. precompute embeddings-match-matrix:
  - $\hat{q}_{1..n} = TF(q_{1..n})$ = query tokens after applying transformer
  - $\hat{d}_{1..m} = TF(d_{1..m})$ = passage tokens after applying transformer
  - $M_{ij} = \mathbf{cos}(\widehat{q}_i, \widehat{d}_j)$ = match matrix
- ii. apply radial-basis-function kernel:
  - $K_k(M) = \sum_{i=1}^{n} \log \left(\sum_j \exp \left(- \frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2} \right) \right) \longrightarrow$ rbf kernel for a single match, along all passages
  - $s = FC(K) = W \cdot K + b$

idcm



- knowledge distillation / distilled training = passing knowledge from one model to the other
  - we want to improve effectiveness of small efficient models
  - ie. distilBert is a lot smaller than bert but has similar effectiveness
- hybrid architecture:
  - teacher / teacher ensemble = powerful, slow, generates some initial labels and scores
  - student = weak, fast, use labels from teacher to improve margin-mse score (between rel-passages and non-rel-passages)
- how it works:
  - i. teacher training = train teacher on binary loss
  - ii. teacher inference = get teacher scores (just once)
  - iii. student training = use teacher scores to train students

idcm = intra document cascade

- etm = effective teacher model (bertCat)
- estm = effective student model (ck)
- ps = passage store

dense retrieval

neural approach to first-stage retrieval, competing with traditional methods like bm25.

dense retrieval lifecycle

- i) training: tune or train bertDot to generate passage embeddings
- ii) indexing: use nearest-neighbor-index to store all passage embeddings
- iii) searching: use bertDot to encode query to look up closest passage neighbors

bertDot

- trained similar to re-ranking model by taking triples (query, rel-passage, non-rel-passage)
    - non-rel-passages can be either chosen by bm25 or generated by the model itself
    - ANCE = Approximate Nearest Neighbor Negative Contrastive Learning
- query and passage embeddings are completely independent → relevance can be computed just with cosine-similarity

nearest-neighbor search NN

- cosine similarity is not precise but very cheap, because it's just a dot-product
- useful library: faiss

TAS-balancing

- TAS = topic aware sampling
- i. cluster queries/passages by topic
    - not to expensive in practice
- ii. train by topic → this way the margin between rel-passages and non-rel-passages is a lot more significant

zero-shot benchmarking

- BEIR benchmark
- dense retrival (DR) models don’t generalize well to other query distributions
- neural models can have false-positives that don't make sense (bm25 must have lexical overlap as a constraint) but mostly only on sparse-judgements