April 4, 2021

DRQA PAPER
QA model answers factoid questions
advancing of reading comprehension models results in a two stage framework or (1) context retriever to select sample of passages which may have the answer then (2) a machine reader can read the contexts and the passages to identify the correct answer
reducing open-domain QA to just machine learning will cause a lower accuracy rate however, so retrieval needs to be improved
open-domain QA implemented with TF-IDF (term frequency-inverse document frequency) or BM25 (bag of words model/a sequence of text treated like a string of words instead of a grammar-abiding sentence, which ranks a documents based on the number of query's terms, ignoring the location of the found words in the document)
conversely → Dense Passage Retrieval reads the semantic encodings instead of the actual words by themselves, because words like "bad guy" and "villain" mean the same thing but term based systems will fail to find the matching context
dense encodings are also learnable and you can change the embedding functions
retrieval can be done with maximum inner product searches MIPS, which searches for a vector in a set of vectors given a query, that vector will have the maximum inner product with the query vector
learning dense vector representation requires a large number of queries to context matches which is computationally intensive → don't outperform TF-IDF/BM25, so inverse cloze task is desired, which predicts the blocks that contain a masked sentence for pretraining
open retrieval QA shows dense retrieval can outperform BM25, but (1) ICT pre training is computationally intensive and questionable replacements (2) answers returned may be inaccurate when context encoder does not have fine tuning with query-answer pairs

Question: can we improve dense embedding models with query-answer pairs without more pretraining?
embedding is optimized for maximum inner products of the question and relevant passage vectors in a batch
Results: (1) simply fine tuning question and passage encoders for query-answer pairs produces outputs better than BM25 (2) improving retrieval means improving accuracy

open domain QA has to search through many documents and extract one or more passages.
retriever takes input as a question and filters the whole corpus to a smaller set of possibly paragraphs
dense passage retriever DPR indexes all the passages in low dimensional and continuous space using inner product function to improve the dense passage retriever

the encoders use two BERT(natural language processing model networks for the representative vector
inference → passage encoder so dot-product similarity is used to rank the retrieval relevant questions and passages so that relevant pairs of questions and passages will have smaller distances than the irrelevant ones by learning a better embedding function
positive and negative passages
negative passages for learning dual-encoder model that boosts the number of training examples
wikipedia data pre-processing → extracts texts only and creates 100 word passages and prepended with the document title and a SEP token

question answering datasets:
Natural questions (NQ) → realistic questions and answers mined from google search queries
TriviaQA → trivia questions with answers that were scraped from web
WebQuestions → questions selected with Google suggest API
CuratedTREC → sources from TREC QA
selection of positive passages → used BM25 to choose the positive passages to determine which answers can be attained
retrieval methods → effects of different training schemes and run time efficiency
DPR trained with 1000 examples outperforms BM25 in passage retrieval
results: DPR outperforms BM25 on all, combining DPR with BM25 can be improved with
SQuAD has lower performance because the questions were written with the exact answer from the text
sample efficiency → DPR outperforms the BM25 just by training with 1000 examples
in-batch negative training → adding more negative passages to the training selected by BM25 improves the results for DPR
similarity functions used: dot product, cosine and Euclidean L2 distance
DPR captures lexical variations/semantic relationships, BM25 good for selective words
how different passage retrievers affect the final QA accuracy
end to end question answering system is ideal
conclusion: dense retrieval can outperform and potentially replace the traditional sparse retrieval component in open-domain QA

Pruning the Index Contents for Memory Efficient Open-Domain QA
→ purpose of paper is to find how to exclude unnecessary information from the search
selection of relevant passages without a question, which is opposite of the retriever, which relies on the question to filter the contents

drawback of sorting out the passages without considering the question itself,

extractive → direct from passages

abstractive → model generated vocabulary, not from the article themselves

open-QA Pipeline
Rank Twice, reaD Twice → parameters of each components of DPR passage retriever are evaluated separately
reader one → **extractive** span selection, second reader based on fusion-in-decoder and rerank (more computationally intensive)
passage re-ranker receives passages and redefines the re-ranking function
extractive reader → different probabilities of start/end/boundary/answer token
vector of scores are calculated with the components
component fusions → R2D2 aggregates the log probabilities
score aggregation → combined the scores to maximizes correct answer span probability
datasets and data pre processing
train the re ranker and extractive reader and filter out passages without "ground truth passages annotation" and are not matched to F1 heuristics
models and pipeline→ train the ELECTRA system with 2 negative samples per positive
retriever → we use BERT based DPR
passage re ranker → we use longformer-base encoder
extractive reader → pretrained ELECTRA large to read 128 passages then rerank and narrow down to 24
generative reader → concatenation of question, passages and their respective titles at the fusion-in-decoder's input

Compressing the "image size" → save models and index in half precision without significant loss of performance?? compress pytorch
R1D1 and only retriever has lower performance than FiD large
reranker performance → analyze the performance of our retriever and reranker with Accuracy@K
Pruner → NQ-golden test data 90.63% accuracy
this work proposed R2D2 as a novel pipeline for open-domain QA based on 4 components: retriever, reranker, generative reader, and extractive reader which reduces vast number of passages used within the knowledge base

summary: Memory efficient open domain QA system uses a Rank twice Read twice method in the DPR passage retriever

---

read BERT paper and the DPR paper again
try to run efficient baseline

April 1, 2021
benefit is that it can parallelize the computations in computer

using small answer/question pairs → maximized the product of question and answer pairs to work in small answering

(2)
DPR Paper annotations https://arxiv.org/pdf/2004.04906.pdf
Dense Passage Retrieval: matches similar words such as “bad guy” and “villain” together whereas term based systems cannot, DPR requires large number of labelled pairs, so they do not outperform TF-IDF/BM25
paper addresses question: can we train a better dense embedding model using only pairs of questions and passages (or answers),with-out additional pretraining
ablation studies → final solution: embedding is optimized for maximizing inner products of the question and relevant passage vectors
DPR, improving on the retrieval part of the component in the open-domain, DPR goal is to index all the passages in a low dimensional and continuous space so it can retrieve efficiently the top k passages relevant to the input query
encoders: two independent BERT networks and take the CLS token as the output
inference: at inference time, we apply the passage decoder to all the passages and index them using FAISS into a vector space , a library for similarity search and clustering of dense vectors which can also be applied to billions at a time

training the encoders so that a vector space such that relevant pairs of questions and passages will have a smaller distance than the irrelevant ones and learn a better embedding function
positive passages → explicitly available examples,
negative examples are selected from extremely large pool: (1) random passage from corpus (2) BM25: top passages from BM25 which don’t have the question but most question tokens are available, (3) gold: positive passages paired with other questions in the training set

preprocessing of wikipedia to extract pure text from articles then split into 100 word passages and prepended with title of the article

results of DPR performing significantly better than BM25 on all datasets using top-k accuracy, gap is large when k is small, SQuAD has lower performance due to high lexical overlap between passages and questions, and the data set was from a smaller pool of articles → wrote question after they knew the answer in the

(1)
DrQA>drqa>pipeline>drqa.py:
under the DrQA class

Document reader:

```
logger.info('Initializing document reader...')
        reader_model = reader_model or DEFAULTS['reader_model']
        self.reader = reader.DocReader.load(reader_model, normalize=False)
        if embedding_file:
            logger.info('Expanding dictionary...')
            words = reader.utils.index_embedding_words(embedding_file)
            added = self.reader.expand_dictionary(words)
            self.reader.load_embeddings(added, embedding_file)
        if cuda:
            self.reader.cuda()
        if data_parallel:
            self.reader.parallelize()
```

Document retriever:

```
       # Iterate through the predictions, and maintain priority queues for
        # top scored answers for each question in the batch.
        queues = [[] for _ in range(len(queries))]
        for result, ex_ids, batch_size in result_handles:
            s, e, score = result.get()
            for i in range(batch_size):
                # We take the top prediction per
```

```
def process(self, query, candidates=None, top_n=1, n_docs=5,
                return_context=False):
      """Run a single query."""
        predictions = self.process_batch(
            [query], [candidates] if candidates else None,
            top_n, n_docs, return_context
        )
        return predictions[0]
```

DrQA>scripts>pipeline>interactive.py:

```
def process(question, candidates=None, top_n=1, n_docs=5):
    predictions = DrQA.process(
        question, candidates, top_n, n_docs, return_context=True
    )
    table = prettytable.PrettyTable(
        ['Rank', 'Answer', 'Doc', 'Answer Score', 'Doc Score']
    )
    for i, p in enumerate(predictions, 1):
        table.add_row([i, p['span'], p['doc_id'],
                       '%.5g' % p['span_score'],
                       '%.5g' % p['doc_score']])
    print('Top Predictions:')
    print(table)
    print('\nContexts:')
    for p in predictions:
        text = p['context']['text']
        start = p['context']['start']
        end = p['context']['end']
        output = (text[:start] +
                  colored(text[start: end], 'green', attrs=['bold']) +
                  text[end:])
        print('[ Doc = %s ]' % p['doc_id'])
        print(output + '\n')


banner = """
Interactive DrQA
>> process(question, candidates=None, top_n=1, n_docs=5)
>> usage()
"""


def usage():
    print(banner)
```

interactive.py

Open with

```
# ------------------------------------------------------------------------------
# Drop in to interactive mode
# ------------------------------------------------------------------------------


def process(question, candidates=None, top_n=1, n_docs=5):
    predictions = DrQA.process(
        question, candidates, top_n, n_docs, return_context=True
    )
    table = prettytable.PrettyTable(
        ['Rank', 'Answer', 'Doc', 'Answer Score', 'Doc Score']
    )
    for i, p in enumerate(predictions, 1):
        table.add_row([i, p['span'], p['doc_id'],
                       '%.5g' % p['span_score'],
                       '%.5g' % p['doc_score']])
    print('Top Predictions:')
    print(table)
    print('\nContexts:')
    for p in predictions:
        text = p['context']['text']
        start = p['context']['start']
        end = p['context']['end']
        output = (text[:start] +
                  colored(text[start: end], 'green', attrs=['bold']) +
                  text[end:])
        print('[ Doc = %s ]' % p['doc_id'])
        print(output + '\n')


banner = """
Interactive DrQA
>> process(question, candidates=None, top_n=1, n_docs=5)
>> usage()
"""


def usage():
    print(banner)


code.interact(banner=banner, local=locals())
```

```
!cd /content/drive/MyDrive/DrQA/; python scripts/pipeline/interactive.py --tokenizer simple

04/01/2021 06:37:40 PM: [ CUDA enabled (GPU -1) ]
04/01/2021 06:37:40 PM: [ Initializing pipeline... ]
04/01/2021 06:37:40 PM: [ Initializing document ranker... ]
04/01/2021 06:37:40 PM: [ Loading /content/drive/MyDrive/DrQA/data/wikipedia/docs-tfidf-ngram=2-hash=16777216-tokenizer=simple.npz ]
tcmalloc: large alloc 8506228736 bytes == 0x55650a724000 @  0x7fbb34b8f1e7 0x7fbb326cf46e 0x7fbb3271fc7b 0x7fbb326d2ce8 0x556504dbec25 0x556504d7f8e9 0x556504df3ade 0x5
tcmalloc: large alloc 4253114368 bytes == 0x556706250000 @  0x7fbb34b8f1e7 0x7fbb326cf46e 0x7fbb3271fc7b 0x7fbb326d2ce8 0x556504dbec25 0x556504d7f8e9 0x556504df3ade 0x5
04/01/2021 06:38:55 PM: [ Initializing document reader... ]
04/01/2021 06:38:55 PM: [ Loading model /content/drive/MyDrive/DrQA/data/reader/multitask.mdl ]
04/01/2021 06:39:03 PM: [ Initializing tokenizers and document retrievers... ]

Interactive DrQA
>> process(question, candidates=None, top_n=1, n_docs=5)
>> usage()

>>> process('What is question answering?')
04/01/2021 07:14:30 PM: [ Processing 1 queries... ]
04/01/2021 07:14:30 PM: [ Retrieving top 5 docs... ]
04/01/2021 07:14:35 PM: [ Reading 106 paragraphs... ]
04/01/2021 07:14:36 PM: [ Processed 1 queries in 5.3353 (s) ]
Top Predictions:
+------+--------+----------------------------+--------------+-----------+
| Rank | Answer |            Doc             | Answer Score | Doc Score |
+------+--------+----------------------------+--------------+-----------+
|  1   |   QA   | Social information seeking |    2964.4    |   212.63  |
+------+--------+----------------------------+--------------+-----------+

Contexts:
[ Doc = Social information seeking ]
Social information seeking is often materialized in online question-answering (QA) websites, which are driven by a community. Such QA sites have emerged in the past few

>>>
```

March 31, 2021

```
!cd /content/drive/MyDrive/DrQA/; python scripts/pipeline/interactive.py
--tokenizer simple
```

03/31/2021 11:40:16 PM: [CUDA enabled (GPU -1)]
03/31/2021 11:40:16 PM: [Initializing pipeline...]
03/31/2021 11:40:16 PM: [Initializing document ranker...]
03/31/2021 11:40:16 PM: [Loading
/content/drive/MyDrive/DrQA/data/wikipedia/docs-tfidf-ngram=2-hash=16777216-tokenizer=simple.npz]
tcmalloc: large alloc 8506228736 bytes == 0x55ef0a568000 @ 0x7f8a477581e7
0x7f89f358a46e 0x7f89f35dac7b 0x7f89f358dce8 0x55ef04d78c25 0x55ef04d398e9
0x55ef04dadade 0x55ef04da7b0e 0x55ef04d3a77a 0x55ef04da986a 0x55ef04d3c72b
0x55ef04d7d5e9 0x55ef04d7d55c 0x55ef04e20e59 0x55ef04da8fad 0x55ef04d3a69a
0x55ef04dace50 0x55ef04d3a69a 0x55ef04dace50 0x55ef04da7b0e 0x55ef04d3b02c
0x55ef04d7bd39 0x55ef04d78c84 0x55ef04d3b231 0x55ef04daa1e6 0x55ef04da7b0e
0x55ef04d3b02c 0x55ef04d7bd39 0x55ef04d78c84 0x55ef04d398e9 0x55ef04dadade
tcmalloc: large alloc 4253114368 bytes == 0x55f1060ba000 @ 0x7f8a477581e7
0x7f89f358a46e 0x7f89f35dac7b 0x7f89f358dce8 0x55ef04d78c25 0x55ef04d398e9
0x55ef04dadade 0x55ef04da7b0e 0x55ef04d3a77a 0x55ef04da986a 0x55ef04d3c72b
0x55ef04d7d5e9 0x55ef04d7d55c 0x55ef04e20e59 0x55ef04da8fad 0x55ef04d3a69a
0x55ef04dace50 0x55ef04d3a69a 0x55ef04dace50 0x55ef04da7b0e 0x55ef04d3b02c
0x55ef04d7bd39 0x55ef04d78c84 0x55ef04d3b231 0x55ef04daa1e6 0x55ef04da7b0e
0x55ef04d3b02c 0x55ef04d7bd39 0x55ef04d78c84 0x55ef04d398e9 0x55ef04dadade
03/31/2021 11:41:22 PM: [Initializing document reader...]
03/31/2021 11:41:22 PM: [Loading model
/content/drive/MyDrive/DrQA/data/reader/multitask.mdl]
03/31/2021 11:41:43 PM: [Initializing tokenizers and document retrievers...]

Interactive DrQA
>> process(question, candidates=None, top_n=1, n_docs=5)
>> usage()

>>> process('What is question answering?')
03/31/2021 11:44:04 PM: [Processing 1 queries...]
03/31/2021 11:44:04 PM: [Retrieving top 5 docs...]
03/31/2021 11:44:11 PM: [Reading 106 paragraphs...]
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:202: UserWarning: masked_fill_ received a mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype

```
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  scores.data.masked_fill_(y_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:275: UserWarning: masked_fill_ received a
mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  scores.data.masked_fill_(x_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:242: UserWarning: masked_fill_ received a
mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  xWy.data.masked_fill_(x_mask.data, -float('inf'))
process('What is the answer to life, the universe, and everything?')



quit
quit()
Process ForkPoolWorker-1:
Process ForkPoolWorker-3:
Process ForkPoolWorker-2:
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 110, in worker
    task = get()
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 110, in worker
    task = get()
  File "/usr/lib/python3.7/multiprocessing/queues.py", line 351, in get
    with self._rlock:
  File "/usr/lib/python3.7/multiprocessing/queues.py", line 351, in get
    with self._rlock:
```

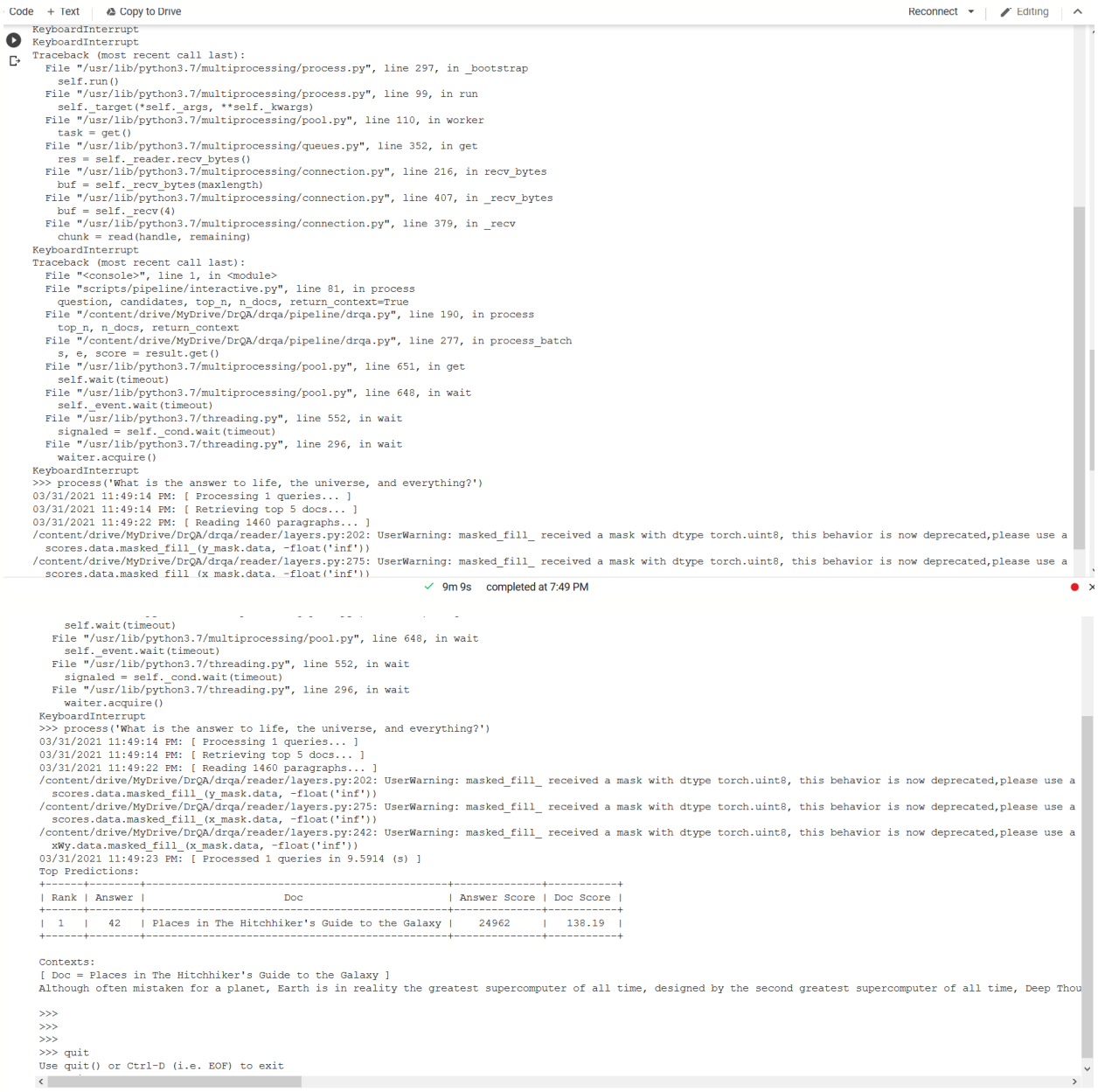
```
  File "/usr/lib/python3.7/multiprocessing/synchronize.py", line 95, in __enter__
    return self._semlock.__enter__()
  File "/usr/lib/python3.7/multiprocessing/synchronize.py", line 95, in __enter__
    return self._semlock.__enter__()
KeyboardInterrupt
KeyboardInterrupt
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 110, in worker
    task = get()
  File "/usr/lib/python3.7/multiprocessing/queues.py", line 352, in get
    res = self._reader.recv_bytes()
  File "/usr/lib/python3.7/multiprocessing/connection.py", line 216, in recv_bytes
    buf = self._recv_bytes(maxlength)
  File "/usr/lib/python3.7/multiprocessing/connection.py", line 407, in _recv_bytes
    buf = self._recv(4)
  File "/usr/lib/python3.7/multiprocessing/connection.py", line 379, in _recv
    chunk = read(handle, remaining)
KeyboardInterrupt
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "scripts/pipeline/interactive.py", line 81, in process
    question, candidates, top_n, n_docs, return_context=True
  File "/content/drive/MyDrive/DrQA/drqa/pipeline/drqa.py", line 190, in process
    top_n, n_docs, return_context
  File "/content/drive/MyDrive/DrQA/drqa/pipeline/drqa.py", line 277, in process_batch
    s, e, score = result.get()
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 651, in get
    self.wait(timeout)
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 648, in wait
    self._event.wait(timeout)
  File "/usr/lib/python3.7/threading.py", line 552, in wait
    signaled = self._cond.wait(timeout)
  File "/usr/lib/python3.7/threading.py", line 296, in wait
    waiter.acquire()
KeyboardInterrupt
>>> process('What is the answer to life, the universe, and everything?')
```

```
03/31/2021 11:49:14 PM: [ Processing 1 queries... ]
03/31/2021 11:49:14 PM: [ Retrieving top 5 docs... ]
03/31/2021 11:49:22 PM: [ Reading 1460 paragraphs... ]
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:202: UserWarning: masked_fill_ received a
mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  scores.data.masked_fill_(y_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:275: UserWarning: masked_fill_ received a
mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  scores.data.masked_fill_(x_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:242: UserWarning: masked_fill_ received a
mask with dtype torch.uint8, this behavior is now deprecated,please use a mask with dtype
torch.bool instead. (Triggered internally at
/pytorch/aten/src/ATen/native/cuda/LegacyDefinitions.cpp:28.)
  xWy.data.masked_fill_(x_mask.data, -float('inf'))
03/31/2021 11:49:23 PM: [ Processed 1 queries in 9.5914 (s) ]
Top Predictions:
+------+--------+----------------------------------------------+--------------+-----------+
| Rank | Answer |                  Doc                         | Answer Score | Doc Score |
+------+--------+----------------------------------------------+--------------+-----------+
|  1   |   42   | Places in The Hitchhiker's Guide to the Galaxy |    24962     |   138.19  |
+------+--------+----------------------------------------------+--------------+-----------+

Contexts:
[ Doc = Places in The Hitchhiker's Guide to the Galaxy ]
Although often mistaken for a planet, Earth is in reality the greatest supercomputer of all time,
designed by the second greatest supercomputer of all time, Deep Thought, to calculate the
Ultimate Question of Life, The Universe and Everything (to which the answer is 42). It was built
by the then-thriving custom planet industry of Magrathea to run a ten-million-year program in
which organic life would play a major role. Slartibartfast, a Magrathean designer, was involved
in the project and signed his name among the fjords of Norway (an area that won him an award).

>>>
>>>
>>>
>>> quit
Use quit() or Ctrl-D (i.e. EOF) to exit
```

>>> quit()

```
[ ] !cd /content/drive/MyDrive/DrQA/; python scripts/pipeline/interactive.py --tokenizer simple

    03/31/2021 11:40:16 PM: [ CUDA enabled (GPU -1) ]
    03/31/2021 11:40:16 PM: [ Initializing pipeline... ]
    03/31/2021 11:40:16 PM: [ Initializing document ranker... ]
    03/31/2021 11:40:16 PM: [ Loading /content/drive/MyDrive/DrQA/data/wikipedia/docs-tfidf-ngram=2-hash=16777216-tokenizer=simple.npz ]
    tcmalloc: large alloc 8506228736 bytes == 0x55ef0a568000 @  0x7f8a477581e7 0x7f89f358a46e 0x7f89f35dac7b 0x7f89f358dce8 0x55ef04d78c25 0x55ef04d398e9 0x55ef04dadade
    tcmalloc: large alloc 4253114368 bytes == 0x55f1060ba000 @  0x7f8a477581e7 0x7f89f358a46e 0x7f89f35dac7b 0x7f89f358dce8 0x55ef04d78c25 0x55ef04d398e9 0x55ef04dadade
    03/31/2021 11:41:22 PM: [ Initializing document reader... ]
    03/31/2021 11:41:22 PM: [ Loading model /content/drive/MyDrive/DrQA/data/reader/multitask.mdl ]
    03/31/2021 11:41:43 PM: [ Initializing tokenizers and document retrievers... ]

    Interactive DrQA
    >> process(question, candidates=None, top_n=1, n_docs=5)
    >> usage()

    >>> process('What is question answering?')
    03/31/2021 11:44:04 PM: [ Processing 1 queries... ]
    03/31/2021 11:44:04 PM: [ Retrieving top 5 docs... ]
```

```
03/31/2021 11:44:04 PM: [ Retrieving top 5 docs... ]
03/31/2021 11:44:11 PM: [ Reading 106 paragraphs... ]
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:202: UserWarning: masked_fill_ received a mask with dtype torch.uint8, this behavior is now deprecated,please use a
  scores.data.masked_fill_(y_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:275: UserWarning: masked_fill_ received a mask with dtype torch.uint8, this behavior is now deprecated,please use a
  scores.data.masked_fill_(x_mask.data, -float('inf'))
/content/drive/MyDrive/DrQA/drqa/reader/layers.py:242: UserWarning: masked_fill_ received a mask with dtype torch.uint8, this behavior is now deprecated,please use a
  xWy.data.masked_fill_(x_mask.data, -float('inf'))
process('What is the answer to life, the universe, and everything?')



quit
quit()
Process ForkPoolWorker-1:
Process ForkPoolWorker-3:
Process ForkPoolWorker-2:
Traceback (most recent call last):
Traceback (most recent call last):
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 297, in _bootstrap
    self.run()
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/process.py", line 99, in run
    self._target(*self._args, **self._kwargs)
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 110, in worker
    task = get()
  File "/usr/lib/python3.7/multiprocessing/pool.py", line 110, in worker
    task = get()
  File "/usr/lib/python3.7/multiprocessing/queues.py", line 351, in get
    with self._rlock:
  File "/usr/lib/python3.7/multiprocessing/queues.py", line 351, in get
    with self._rlock:
  File "/usr/lib/python3.7/multiprocessing/synchronize.py", line 95, in __enter__
    return self._semlock.__enter__()
  File "/usr/lib/python3.7/multiprocessing/synchronize.py", line 95, in __enter__
    return self._semlock.__enter__()
KeyboardInterrupt
KeyboardInterrupt
Traceback (most recent call last):
```



March 28, 2021

ReLU → rectified linear unit → if x > 0 return x, else return 0

Natural Language Processing Powerpoint

Recurrent Neural Network → a type of neural network system where the nodes pass information on in a sequential order, “left to right”, thus linearity, cannot look forward into the future

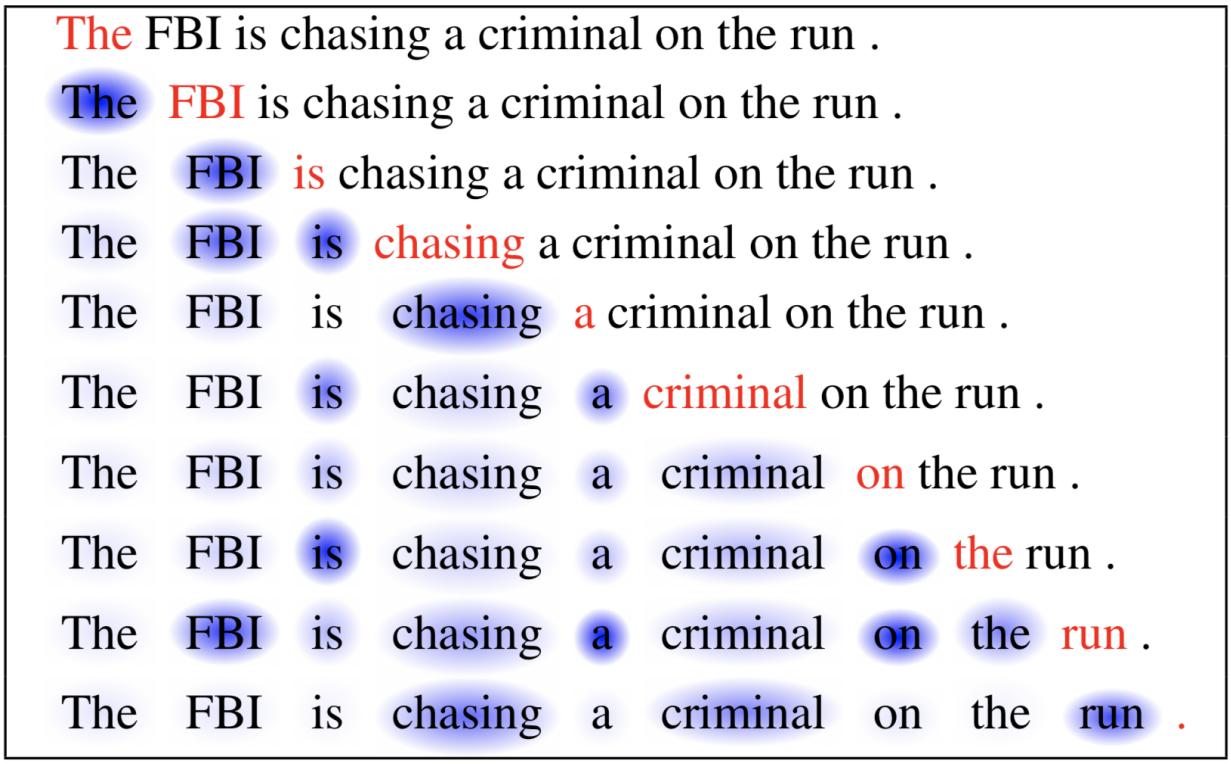
→ problem of this is that there is a literal gap between words and affects the ability for the model to recognize that they are actually related to each other → referred to as **gradient** problems (long distance dependencies)

→ linear order is not the way to think of sentences

word windows: solves these problems somewhat because they relate some of the ideas over longer distances but not extreme distances

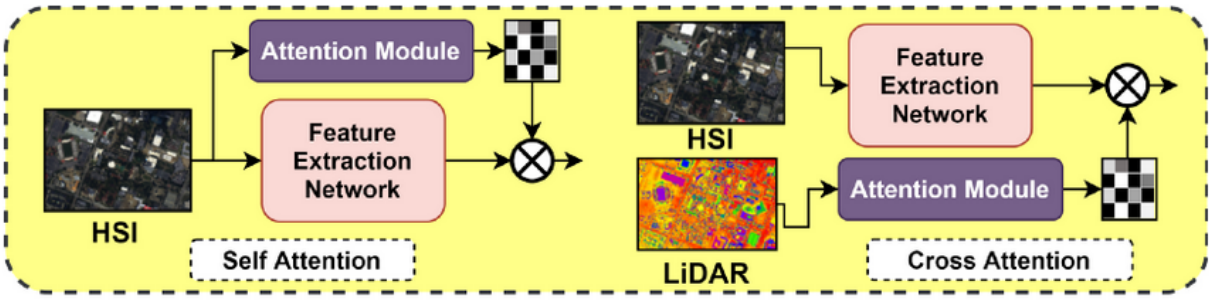
attention: treats the words' values differently, each word is an individual query of its own (queries, keys, and values → calculate a value to weigh the importance of queries and keys)

self attention → interprets a sentence input to create a representation of important ideas and related words in the sentence



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#self-attention

Cross attention → different from self attention, which only uses "one modality", defined as a "method" of analyzing a given input → such as pictures and what objects are in it



Self-attention vs cross-attention for multimodal fusion. The self-attention module (left) works only on single modality where both the hidden representations as well as the attention mask are derived from the same modality (HSIs). On the other hand, in the cross-attention module (right), the attention mask is derived from a different modality (LiDAR) and is harnessed to enhance the latent features from the first modality

[Cross-Attention is what you need! | by Satyam Mohla]
using self attention means there is no order information, so sinusoidal position representations are made to determine the relativity between two words from each other
→ good because this takes all positions into account, bad because this is not learnable and this requires you to go onto longer and longer sequences one by one
self attention: singular modality, give it a specified order because it cannot do it by itself, re-evaluate previous values after passing through a new word (feed forward), masking to prevent future information from interfering with the already analyzed words
feed-forward → information moves forward
→ model that takes word embeddings (word weight vectors) and their position presentations, puts them into a transformer encoder and decoder returns predictions
multiple attention heads → "looks" for different things at once

What is a transformer?
encoder + decoder and connections between them
encoder (feed forward neural network + self attention)
decoder (feed forward, encoder & decoder attention, self attention)
tensors/vectors → scalars and vectors what describes a relationships between objects in vector spaces (so also relationships between different words)
multi headed attention → focuses on different words at once in a sentence

BERT, ELMo
BERT → bidirectional encoder representations from transformers → language learning model that allows fast learning with the use of a left and right context (?) in multiple layers (neural network layers?), it is a trained Transformer encoder stack

→ BERT uses self attention → better than cross because they can see all token information at once in the sequence given
→ train with actual text and check prediction against it for fine tuning

simple task for BERT → predict if an email if spam or not
word embedding → numerical representation of words as vectors
ELMo → NLP model that reads a whole sentence before assigning embeddings to each word, the ELMo LSTM is trained with a large dataset and is trained to predict the next word, has sense of previous word too
ULM-FiT: more methods and an extra language model + process to make language model more precise

masked language model/cloze task and also replaces a word with another word to predict the correct word in that position

Reading wikipedia to Answer Open-Domain Questions

abstract → task is to have a machine learn how to retrieve a relevant article and have the machine understand the article → result combine search component which is based on bigram hashing and TF-IDF matching with multi-layer RNN → bigram hashing and TF-IDF have best results combined
*** wikipedia is the sole knowledge source for DrQA system → outperforms wikipedia's own search engine
DrQA and MRS (machine reading at scale) →2 components (1) find relevant articles (2) document reader to extract answers from a small group of articles or a single article

documental retrieval system first used to narrow the search space from the start
inverted index lookup (index based on content and redirects to different documents), then TF-IDF weighted bag of words (BoW, extract features from a sample of text to use as the input), then use local word order with n-gram features

n-gram → a sequence of n items' attributes from a sample of text (bi-gram is 2 words from a sample)

the bigram (n-gram, n = 2) is used to preserve speed and memory efficiency
hashing: the use of a function to assign data of any specificity to a singular number (maybe each digit position is a different category) → 2^24 bins
murmur3 hash → general hash-based lookup system

Glove (global vectors for word representation) word embedding obtains vector representations for words
glove word embeddings used from sources all across internet to narrow down the most common question words for training
token features → small individual aspects of a word
question encoding → evaluate the importance of questions words in the query

take tokens → RNN to give out the tokens for the right answer
basic idea: first present question/ documents into vectors, then take the tokens
perform encoding on questions/documents → to get answer →
represent question and passage then classify the article +

3 data types → wikipedia as knowledge source, SQuAD dataset (Stanford question answering dataset) for training, and QA datasets (CuratedTREC,WebQuestions, WikiMovies) to test the open domain
short-term memory network (LSTM) → special RNN with the ability to pass on previous inputs/outputs
LSTM → document reader evaluation → with 3 layer bidirectional LSTMs and hidden units (tokens?) → for both paragraph and question encoding with SQuAD
p as tokens, dropout with p =0.3 threshold?
trained with SQuAD then fine-tune (DS),, then multitask with (DS) → single document reader model jointly trained
MRS is a challenge of its own
future steps → document reader should read a few paragraphs at a time, as it currently reads only one, perform end to end training with document retriever and document reader instead of independent training
best single model is the multitask (DS) system
large drop in SQuAD performance when given less direct questions, good results when a singular paragraph is in mind to be the answer, because it narrows to a paragraph even without context

LSTM → solution to the vanishing gradient problem, preserves error that can be back propagated through time and layers (containers of a weighted input and has a transformation of functions)
LSTMs have a forget gate for clearing memory at the end of documents and is a cell of its own layers → input layer, hidden layers with different types of activation functions, and output layer

basically → RNN, sequence can be anything, they only have the word information, but not the context overall, or how they interact with each other (adjacent words), new feature token after

each new word analysis, that's the idea of recurrent, lstm can capture context information THIS IS THE DIFFERENCE, NEW REPRESENTATIONS
How do classifiers know the end and start condition → it learns the introduction from being trained in other samples, also

LSTM examples → used to identify handwriting, speech, and anomalies in traffic or intrusion detection systems

tech report of open domain QA
→ build self contained systems that contain all of the knowledge stored in documents/databses/parameters of a neural network, etc
4 tracks → best accuracy with no storage limit, accuracy + under 6GiB, best under 500MiB. smallest system with 25% accuracy
corpus → collection of authentic text/audio into data sets

Word2vec

assigns various categories of vector values to words, or word embeddings

neural language modeling example --> output most common words' probability after typing a word

training language models is easy because they can be simply paired with articles or any text

create a sliding "window" of samples of words in phrases

skip gram architecture --> analyze before and after gap in phrase

negative samples for non-related words that come after or before the target word

heuristic approach: non traditional way of solving an problem with a different algorithm

recurrent neural networks (RNN): allows previous outputs to be used as inputs

beam width --> parameter for beam search B is number of top likely words

GloVe --> global vectors for word representation --> word imbedding technique for calculating cost (?)

t-SNE: word vector visualizer

bleu score: quantifies how good a machine translation is

attention model --> emphasis on the various ideas in the sentence (teddy bear and book)

RNNs forget information/context as sentences go on, LSTM or long short term memory are RNNs that are designed to avoid long term dependency problem

summarize with own words and take notes

go through the paper again → explain yourself

bag of words → treating words in a sentence as related ideas and helps with training models for predicting missing words

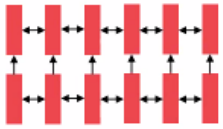
Prepare for not time
1 hour read

1) mapping bigrams to hash → why and what is the hash?
2) what is inverted index?
3) why/what word embedding **assignment of vectors to words to create more dimensional meaning**
4) what is the neural network trying to do/examples (LSTM)
5) explain results and why:
6) what is self / cross attention?

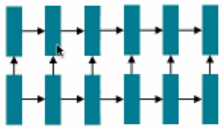
how do we convert a paragraph into a vector → word embeddings
token feature → identifier of what type of word/part of speech the word may be
aligned question embedding → basically considers features and context of both question and the sample paragraph
predict the start of the answer within the paragraph by using dot product
use retriever to find relevant articles, then find the answers from relevant ones

As of last week: recurrent models for (most) NLP!

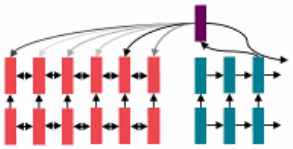
- Circa 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM: (for example, the source sentence in a translation)



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.



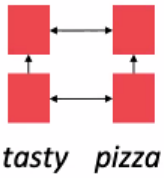
- Use attention to allow flexible access to memory



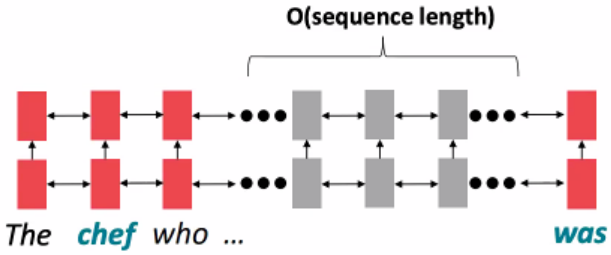
red is bidirectional
attention used to focus on a more important idea and have it sent to the purple box
stronger links and weights to different cells (attention mechanism)

Issues with recurrent models: Linear interaction distance

- RNNs are unrolled "left-to-right".
- This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings

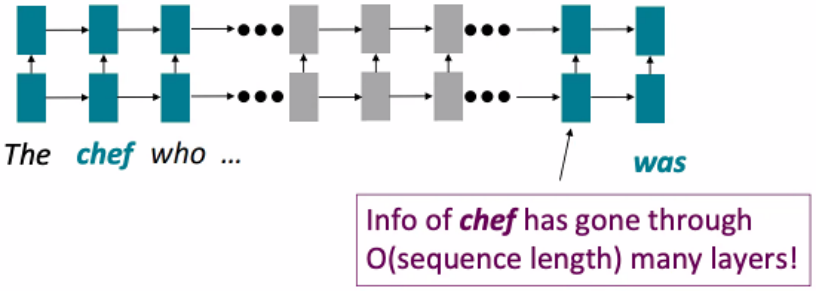


- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



5

- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences...

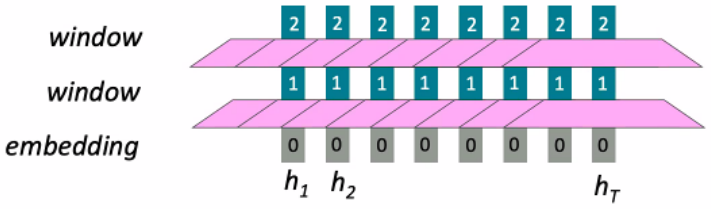


6

gradient managing problem → combining the vectors (dot product) of many vector eventually returns a tiny result, inversely, strongly correlated word embedding results in huge result and approaches infinity

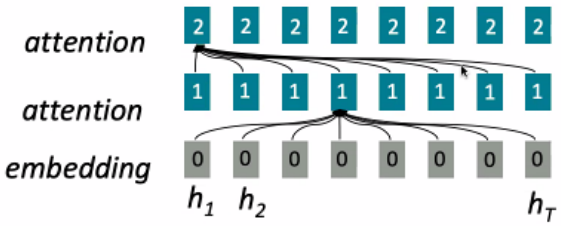
If not recurrence, then what? **How about word windows?**

- **Word window models aggregate local contexts**
  - (Also known as 1D convolution; we'll go over this in depth later!)
  - Number of unparallelizable operations does not increase sequence length!



Numbers indicate min # of steps before a state can be computed

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!



All words attend to all words in previous layer; most arrows here are omitted

Self-Attention

- Recall: Attention operates on **queries**, **keys**, and **values.**
  - We have some **queries** $q_1, q_2, \dots, q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, \dots, k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, \dots, v_T$. Each value is $v_i \in \mathbb{R}^d$

The number of queries can differ from the number of keys and values in practice.

- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, \dots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:

$$e_{ij} = q_i^\top k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})} \qquad \text{output}_i = \sum_j \alpha_{ij} v_j$$

Compute **key-** Compute attention Compute outputs as

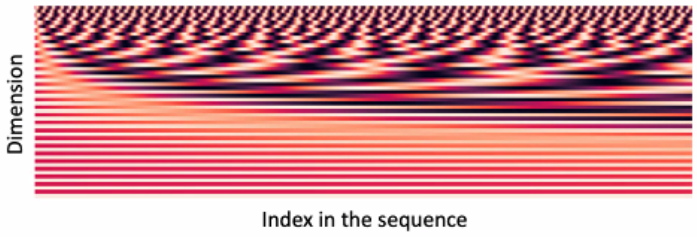
self attention → attention into queries, keys, values from the same source → see how close the queries, keys, values are connected, similarity

$a_{i,j}$ is the attention

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:

cons: not learnable

sinusoids → LSTM does not need this ^ but self attention does not

Position representation vectors learned from scratch

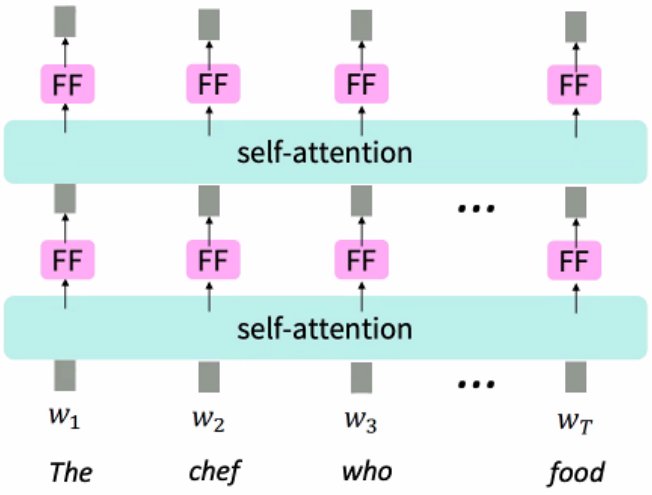
- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $p \in \mathbb{R}^{d \times T}$, *and let each* p_i *be a column of that matrix!*
- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, T$.
- Most systems use this!

- Sometimes people try more flexible representations of position:

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors
- Easy fix: add a **feed-forward network** to post-process each output vector.

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



Intuition: the FF network processes the result of attention

ReLU → rectified Linear Unit →x if x> 0, else 0

heads → use different keys and values to create various heads of values
don't worry about how to implement this yet

Homework

1) re-read research paper again on DrQA
2) annotate