ch3 Design and Implementation

3.1 Dataset description

This is a question an open-domain question answering (QA) system should be able to respond to Question Answer systems (url is at: https://ai.google.com/research/NaturalQuestions/dataset)

Open Domain Question Answering

A core goal in artificial intelligence is to build systems that can read the web, and then answer complex questions about any topic. These question-answering (QA) systems could have a big impact on the way that we access information. Furthermore, open-domain question answering is a benchmark task in the development of Artificial Intelligence, since understanding text and being able to answer questions about it is something that we generally associate with intelligence.

The Natural Questions Dataset

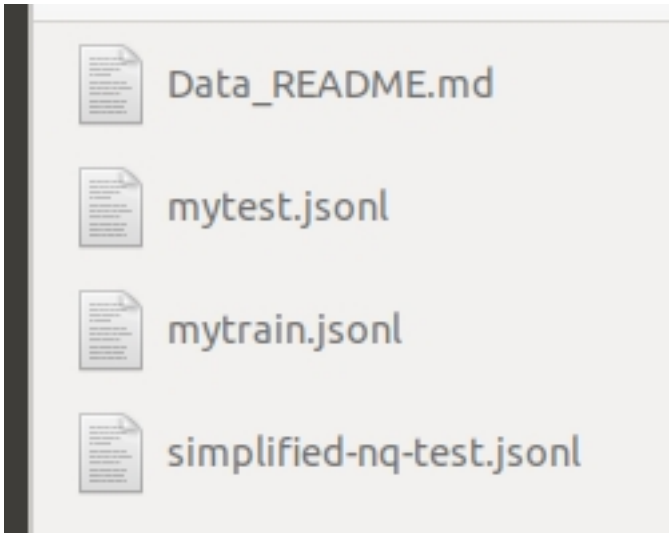
To help spur development in open-domain question answering, google have created the Natural Questions (NQ) corpus, along with a challenge website based on this data. The NQ corpus contains questions from real users, and it requires QA systems to read and comprehend an entire Wikipedia article that may or may not contain the answer to the question. The inclusion of real user questions, and the requirement that solutions should read an entire page to find the answer, cause NQ to be a more realistic and challenging task than prior QA datasets.

Part of the data annotation information of an example displayed by the visualization tool：

URL	https://en.wikipedia.org//w/index.php?title=Trade_winds&oldid=817251427
Question	what purpose did seasonal monsoon winds have on trade
Candidate With Answer	[0]
Short Answer	
Long answer pred	161:285 (6.2730677127838135)
Short answer pred	170:176 (6.2730677127838135) - the steering flow for tropical storms

Index	Contains Answer	Top Level Long Answer Candidate
0(44:161)	True	The trade winds are the prevailing pattern of easterly surface winds found in the tropics , within the lower portion of the Earth 's atmosphere , in the lower section of the troposphere near the Earth 's equator . The trade winds blow predominantly from the northeast in the Northern Hemisphere and from the southeast in the Southern Hemisphere , strengthening during the winter and when the Arctic oscillation is in its warm phase . Trade winds have been used by captains of sailing ships to cross the world 's oceans for centuries , and enabled European empire expansion into the Americas and trade routes to become established across the Atlantic and Pacific oceans .
1(161:285)	False	In meteorology , the trade winds act as the steering flow for tropical storms that form over the Atlantic , Pacific , and southern Indian Oceans and make landfall in North America , Southeast Asia , and Madagascar and eastern Africa , respectively . Trade winds also transport African dust westward across the Atlantic Ocean into the Caribbean Sea , as well as portions of southeastern North America . Shallow cumulus clouds are seen within trade wind regimes , and are capped from becoming taller by a trade wind inversion , which is caused by descending air aloft from within the subtropical ridge . The weaker the trade winds become , the more rainfall can be expected in the neighboring landmasses .
2(343:547)	False	The term trade winds originally derives from the early fourteenth century late Middle English word ' trade , ' meaning `` path '' or `` track . '' The Portuguese recognized the importance of the trade winds (then the Volta do mar , meaning in Portuguese `` turn of the sea '' but also `` return from the sea '') in navigation in both the north and south Atlantic ocean as early as the 15th

URL	Question	Long Answer	Short Answer	Prediction	Parsed Document
Trade winds	what purpose did seasonal monsoon winds have on trade	44 : 161 , The **trade winds** are the prevailing pattern of easterly surface winds found in the tropics, within the lower portion of the Earth's atmosphere, in the lower section of the troposphere near the Earth's equator. The trade winds blow predominantly from the northeast in the Northern Hemisphere and from the southeast in the Southern Hemisphere, strengthening during the winter and when the Arctic oscillation is in its warm phase. Trade winds have been used by captains of sailing ships to cross the world's oceans for centuries, and enabled European empire expansion into the Americas and trade routes to become established across the Atlantic and Pacific oceans.		Long answer: 161 : 285 score: 6.2730677127838135 Short answer: 170 : 176 score: 6.2730677127838135 Yes-no answer: NONE Short answer text: the steering flow for tropical storms	link
High School Musical 2	where did they film high school musical two	: ,		Long answer: 260 : 344 score: 8.831056237220764 Short answer: 288 : 294 score: 8.831056237220764 Yes-no answer: NONE Short answer text: Disneyland , in Anaheim , California	link
List of Nobel laureates in Physics	who got the first nobel prize in physics	233 : 388 , The first Nobel Prize in Physics was awarded in 1901 to Wilhelm Conrad Röntgen, of Germany, who received 150,782 SEK, which is equal to 7,731,004 SEK in December 2007. John Bardeen is the only laureate to win the prize twice—in 1956 and 1972. Maria Skłodowska-Curie also won two Nobel Prizes, for physics in 1903 and chemistry in 1911. William Lawrence Bragg was, until October 2014, the youngest ever Nobel laureate; he won the prize in 1915 at the age of 25.[5] Two women have won the prize: Curie and Maria Goeppert-Mayer (1963).[6] As of 2017, the prize has been awarded to 206 individuals. There have been six years in which the Nobel Prize in Physics was not awarded (1916, 1931, 1934, 1940–1942).	245:251Wilhelm Conrad Röntgen, of Germany	Long answer: 233 : 388 score: 12.41018694639206 Short answer: 245 : 251 score: 12.41018694639206 Yes-no answer: NONE Short answer text: Wilhelm Conrad Röntgen , of Germany	link



our input dataset

Data fields

document_text - the text of the article in question (with some HTML tags to provide document structure). The text can be tokenized by splitting on whitespace.

question_text - the question to be answered

long answer candidates - a JSON array containing all of the plausible long answers.

annotations - a JSON array containing all of the correct long + short answers. Only provided for train.

document url - the URL for the full article. Provided for informational purposes only. This is NOT the simplified version of the article so indices from this cannot be used directly. The content may also no longer match the html used to generate document text. Only provided for train.

example_id - unique ID for the sample.

3.2 Open problem about the dataset

Natural Questions contains (*question*, *wikipedia page*, *long answer*, *short answer*) quadruples where: the question seeks factual information; the Wikipedia page may or may not contain the information required to answer the question; the long answer is a bounding box on this page contain- ing all information required to infer the answer; and the short answer is one or more entities that give a short answer to the question, or a boolean 'yes' or 'no'. Both the long and short answer can be NULL if no viable candidates exist on the Wikipedia page.

All the questions in NQ are queries of 8 words or more that have been issued to the Google search engine by multiple users in a short period of time. From these queries, we sample a subset that either:

1. start with 'who', 'when', or 'where' directly followed by: a) a finite form of 'do' or a modal verb; or b) a finite form of 'be' or 'have' with a verb in some later position;
2. start with 'who' directly followed by a verb that is not a finite form of 'be';
3. contain multiple entities as well as an adjective, adverb, verb, or determiner;
4. contain a categorical noun phrase immediately preceded by a preposition or relative clause;
5. end with a categorical noun phrase, and do not

contain a preposition or relative clause.

1.a	where does the nature conservancy get its funding
1.b	who is the song killing me softly written about
2	who owned most of the railroads in the 1800s
4	how far is chardon ohio from cleveland ohio
5	american comedian on have i got news for you

Above picture gives examples.

Long Answer Identification:

for good questions only, annotators select the earliest HTML bounding box containing enough information for a reader to completely infer the answer to the question. Bounding boxes can be paragraphs, tables, list items, or whole lists. Alternatively, annotators mark 'no answer' if the page does not answer the question, or if the information is present but not contained in a single one of the allowed elements.

Short Answer Identification:

for examples with long answers, annotators select the entity or entities within the long answer that answer the question. Alternatively, annotators can flag that the short answer is 'yes', 'no', or they can flag that no short answer is possible.

3.3 What problem about the dataset we want to explore and solve

My goal is to predict short and long answer responses to real questions about Wikipedia articles. The dataset is provided by Google's Natural Questions, but contains its own unique private test set. A visualization of examples shows long and—where available—short answers.

According to google said, our final results are evaluated using micro F1 between the predicted and expected answers. Predicted long and short answers must match exactly the token indices of one of the ground truth labels (*ormatchYES/NOifthequestionhasayes/noshortanswer*. There may be up to five labels for long answers, and more for short. If no answer applies, leave the prediction blank/ null.

The metric in this competition diverges from the original metric in two key respects:

1) short and long answer formats do not receive separate scores, but are instead combined into a micro F1 score across both formats

2) 2) this competition's metric does not use confidence scores to find an optimal threshold for predictions.

Where F1's definition is as following:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

TP = the predicted indices match one of the possible ground truth indices

FP = the predicted indices do NOT match one of the possible ground truth indices, OR a prediction has been made where no ground truth exists

FN = no prediction has been made where a ground truth exists

3.4 Enlightenment from google's model

Google provides an open source Baseline model for this data set. If a worker wants to do well, he must first sharpen his tools. We first study the baseline model, and then learn from experience and useful parts.

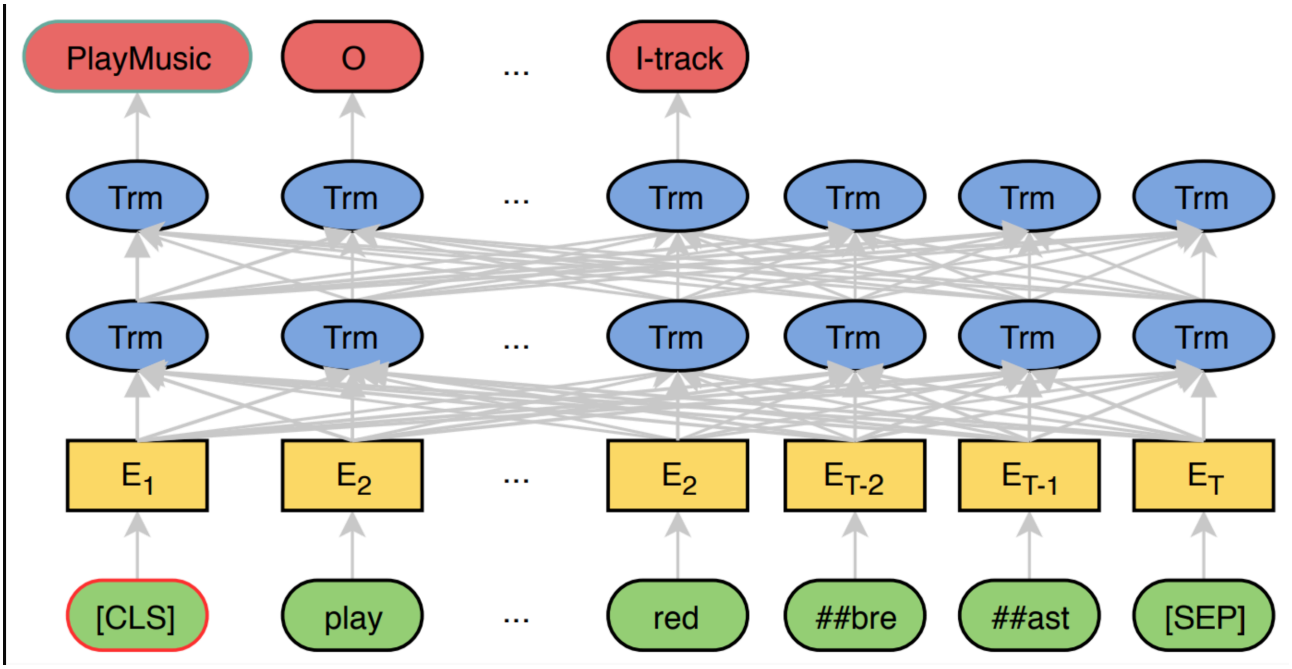


image of JointBERT

The Google 's open source model is called BERTjoint, and the meaning of joint is to use a single model to find long answers and short answers. The basis of this model is BERT, and the data input into BERT is formed by stitching the question and the article body with [sep]. Because the body of the article is very long, a sliding window is used to intercept the body. The default step size is 128 tokens. Since the problem length is almost very small, if the input length of the model is 512 tokens, probably each token will appear in four sliding window samples (chunks) .

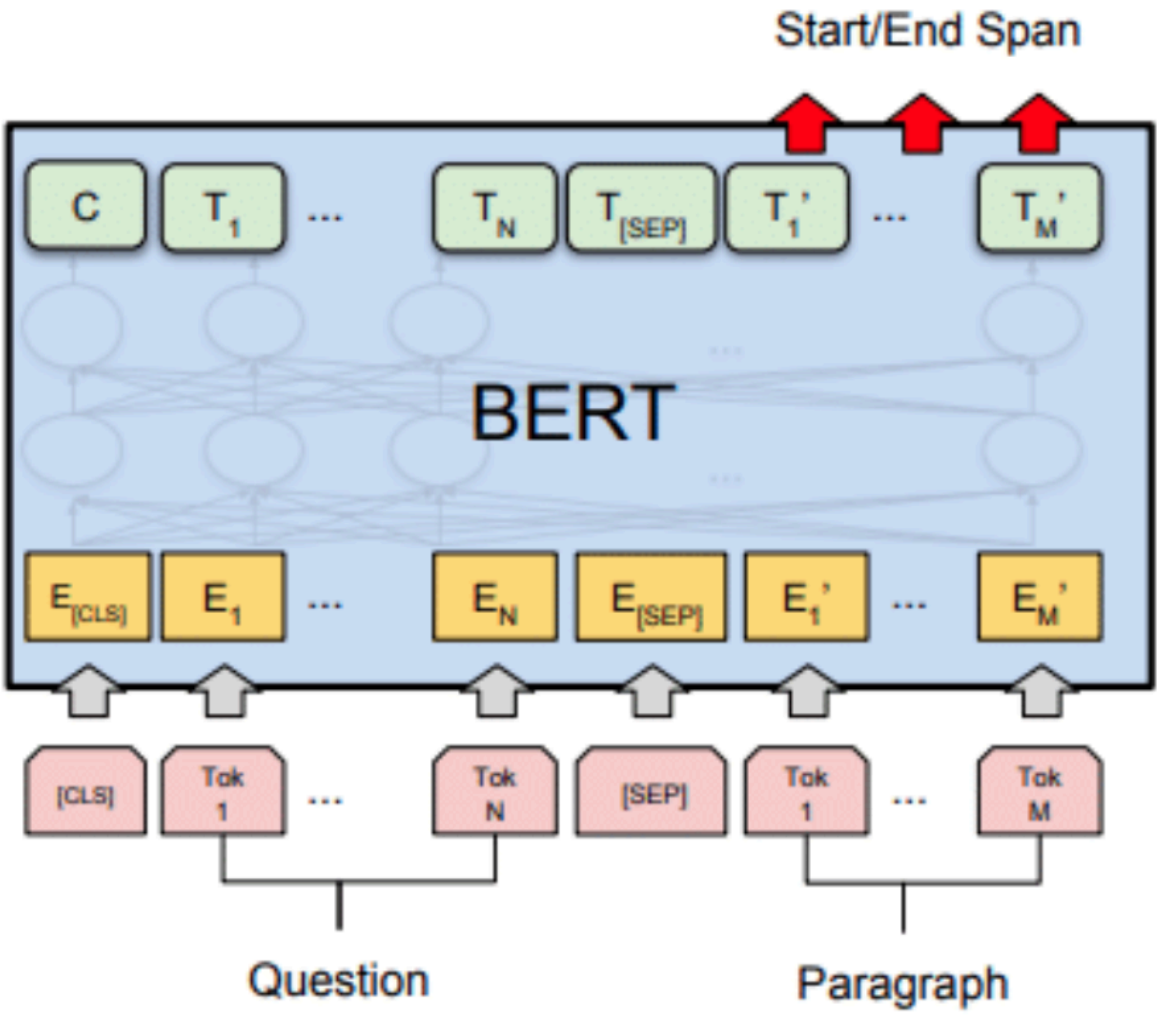
Take out all the top-level candidate long answers, and add a special token to the head to indicate the type (paragraph, table, list) and position of the candidate long answer, that is, which type of long answer candidate is in the full text;

The processed candidate long answers are spliced together for sliding window processing. Combine the text of each window with the question to form an input sample;

The label of the sample contains two parts: the type and the start and end positions.

If the sample contains a short answer, the answer type of the sample is "short", if there is no short answer but there is a long answer, it is marked as "long", otherwise it is "unknown", and the short type may be differentiated into "yes" And "no" two special types. When there is a short

answer, the start and end positions are the start and end positions of the first short answer; when there is only a long answer and no short answer, the start and end positions are marked as the head and end of the entire long answer; when the answer is not included, the start and end positions are both marked as 0, which points to [CLS]（Definition of CLS: bert also adds some special flags: [CLS] The mark is placed at the top of the first sentence, and the representation vector C obtained by BERT can be used for subsequent classification tasks）



The google Bert model: Formally, we define a training set instance as a four-tuple (c,s,e,t)

where c is a context of 512 word piece ids (including question, document tokens and markup), s, e ∈ {0, 1, . . . , 511} are inclusive indices point- ing to the start and end of the target answer span, and t ∈ {0, 1, 2, 3, 4} is the annotated answer type, corresponding to the labels “short”, “long”, “yes”,

“no”, and “no-answer”. Each probability p is obtained as a softmax over scores computed by the BERT model as follow:

$$
\begin{aligned}
L &= -\log p(s, e, t|c) \\
&= -\log p_{\text{start}}(s|c) - \log p_{\text{end}}(e|c) \\
&\quad - \log p_{\text{type}}(t|c),
\end{aligned}
$$

$$
\begin{aligned}
p_{\text{start}}(s|c) &= \frac{\exp(f_{\text{start}}(s, c; \theta))}{\sum_{s'} \exp(f_{\text{start}}(s', c; \theta))}, \\
p_{\text{end}}(e|c) &= \frac{\exp(f_{\text{end}}(e, c; \theta))}{\sum_{e'} \exp(f_{\text{end}}(e', c; \theta))}, \\
p_{\text{type}}(t|c) &= \frac{\exp(f_{\text{type}}(t, c; \theta))}{\sum_{t'} \exp(f_{\text{type}}(t', c; \theta))},
\end{aligned}
$$

where θ represents the BERT model parameters and fstart, fend, ftype represent three different outputs derived from the last layer of BERT.

After studied google’s model, we can get such inspirations:

Good side:

the training method is relatively simple. For the standard reading comprehension task of finding the beginning and the end, a full connection is added directly after the BERT, and then the probability is softmax in the sentence dimension; for the answer category classification, it is after the output of [CLS] With a full connection, as our picture above showed.

Need to improve side:

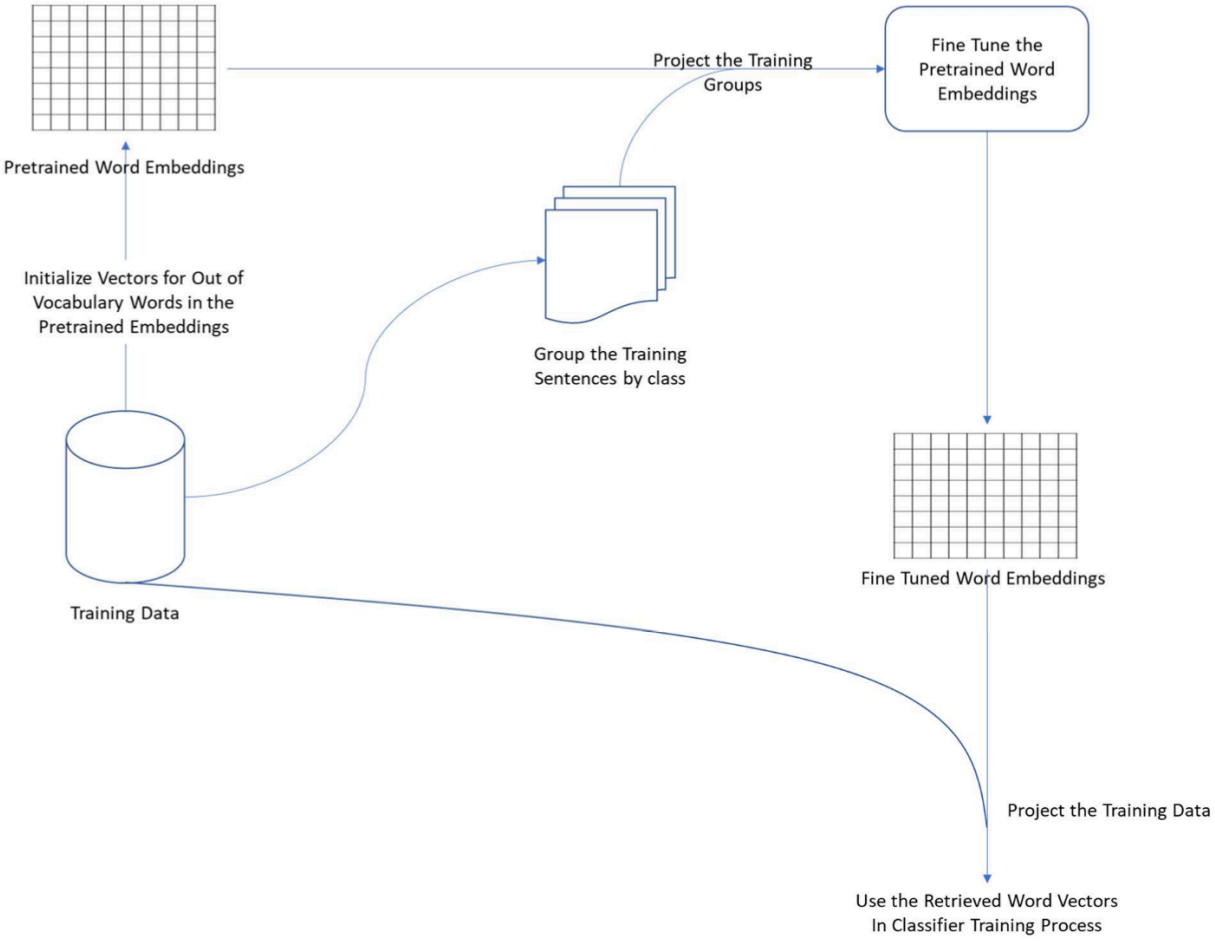
the answer will only be recognized when the answer length is less than a chunk length, which will cause the long answer label with a longer length to be lost; sliding the window after all the candidate long answers are spliced increases the difficulty of finding the long answer.

Inspirations:

I break down each document into parts corresponding to each of the candidate long answers. Then, I label each of the long answer to be 1 if it is the true long answer, or 0 if not. For each of row, I also include the question and the example_id. Then, I train a LSTM to predict that label.

I will use wiki-news-300d-1M.vec , the pertained models as a new and simple approach for fine-tuning pretrained word embeddings for text classification tasks. In this approach, the class in which a term appears, acts as an additional contextual variable during the fine tuning process, and contributes to the final word vector for that term. As a result, words that are used distinctively within a particular class, will bear vectors that are closer to each other in the embedding space and will be more discriminative towards that class.

The use of pretrained word vectors for initializing the wiki-news-300d-1M model before training it, has the advantage of capturing previously learnt features and only modifying these features so as to reflect their association with classes in the training data. This ensures that words that have not appeared in the training data, due to its limited size, are still likely to have vectors in the final embeddings model. So, during the testing phase (or prediction) words that have not been seen before in the training phase, will still be able to contribute to the overall performance of the system



We then use an inference kernel that shows how to use the trained model, predict the final results that google 's test environment can accept.

3.5 design and implementation

We choose to use:

fasttext

gensim

numpy

pandas

tqdm

sklearn

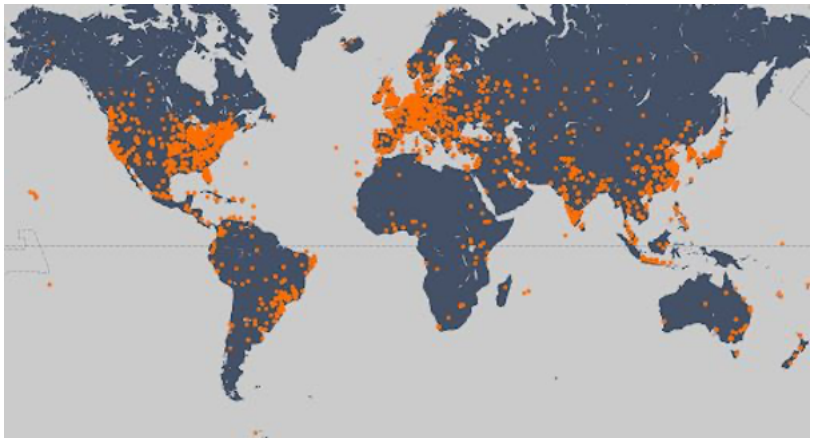
seaborn

tensorflow-gpu

Development Tech Stack we choose
Introduction of development technology and tools

a. tensorflow development framework

TensorFlow is an end-to-end open source machine learning platform. Provide comprehensive and flexible professional tools to enable individual developers to easily create machine learning applications, help researchers to promote the development of cutting-edge technology, and support enterprises to establish stable large-scale applications.



Since its release in 2015, TensorFlow has had 41 million downloads worldwide. As a widely used machine learning framework around the world, TensorFlow has passed its third anniversary, and TensorFlow has gradually become an end-to-end mature platform with a complete ecosystem. With the release of TensorFlow 2.0, it marks the arrival of a new era of TensorFlow, which is easier to use, more flexible and powerful, and more usable in production environments. I hope TensorFlow can become a machine learning platform suitable for all users and can help everyone use it. Machine learning solves real-world problems.

Machine learning is mainly divided into two steps: training and deployment.

In the training phase, TensorFlow not only supports Python, but also provides support for Swift and JS languages. You can choose the language you are familiar with for development.

In the deployment phase, TensorFlow models can run on different platforms, support TensorFlow Serving deployed on the server side, TensorFlow Lite deployed on end-side platforms such as Android, iOS and embedded devices, and TensorFlow.js deployed on browsers and Node servers. And many languages including C language, Java language, Go language, C# language, Rust and R

B. keras

Keras is a neural network library written in pure Python, focusing on deep learning, running on TensorFlow or Theano.

TensorFlow and Theano are currently two popular deep learning libraries, but they are relatively complicated for beginners.

Keras is simple to use and has a clear structure. The underlying computing platform can be based on TensorFlow or Theano with powerful functions.

Keras can run in Python 2.7 or 3.5 environment, perfectly combined with GPU and CPU, released under the MIT license.

Keras is developed and maintained by Google engineer François Chollet.

The following are the design principles of Keras:

o Modularity: A model can be understood as an independent sequence or graph. The models are independent of each other and can be freely combined.

o Minimalism: Each module should be as concise as possible. Each piece of code should appear intuitive and easy to understand when it is first read. There is no black magic, because it will cause trouble for iteration and innovation.

o Extensibility: Adding a new module is super simple and easy. You only need to imitate the existing module to write a new class or function. The convenience of creating new modules makes Keras more suitable for advanced research work.

o Collaboration with Python: Keras does not have a separate model configuration file type (for comparison, caffe has), the model is described by Python code, making it more compact, easy to debug and easy to expand.

C. NumPy

NumPy is the basic package of scientific computing in Python. It is a Python library that provides multi-dimensional array objects, various derived objects (such as masked arrays and matrices), and various APIs for fast array operations, including mathematics, logic, shape operations, sorting, selection, input and output , Discrete Fourier Transform, basic linear algebra, basic statistical operations and random simulation, etc.

The core of the NumPy package is the ndarray object. It encapsulates python's native n-dimensional array of the same data type. In order to ensure its excellent performance, many operations are executed after the code is compiled locally.

There are several important differences between NumPy arrays and native Python Arrays (arrays):

• NumPy arrays have a fixed size when created, which is different from Python's native array objects (which can grow dynamically). Changing the size of the ndarray will create a new array and delete the original array.

• The elements in the NumPy array all need to have the same data type, so they have the same size in memory. Exception: When Python's native array contains NumPy objects, in this case arrays with elements of different sizes are allowed.

• NumPy arrays facilitate advanced math and other types of operations on large amounts of data. Generally, these operations are performed more efficiently and less code than using Python's native arrays.

- More and more Python-based science and mathematics software packages use NumPy arrays; although these tools usually support Python's native arrays as parameters, they will still convert the input arrays to NumPy arrays before processing, and It is also usually output as a NumPy array. In other words, in order to use today's science/mathematics Python-based tools (most of the scientific computing tools) efficiently, it is not enough that you only know how to use Python's native array types

- -you also need to know how to use NumPy arrays.

Development Tech Stack we choose

Experiment introduction and data reading

We hope to use deep learning to predict sentiment and try to surpass the sentiment analysis performance of traditional machine learning algorithms.

We mainly explore the following aspects:

*Use adaptive deep learning model

*Identify and deal with overfitting

*Use word embedding

* Built on a pre-trained model

Data explore

Let you know more about the data situation. Verify that it conforms to the logic you want.

By observing the distribution relationship between features and labels. Preliminarily verify whether this feature has an obvious relationship with the label.

For the later stage of the game, when everyone has tried similar features and solutions that are

reasonable in the practical significance of the data, data exploration is especially important.

Main code is as following:

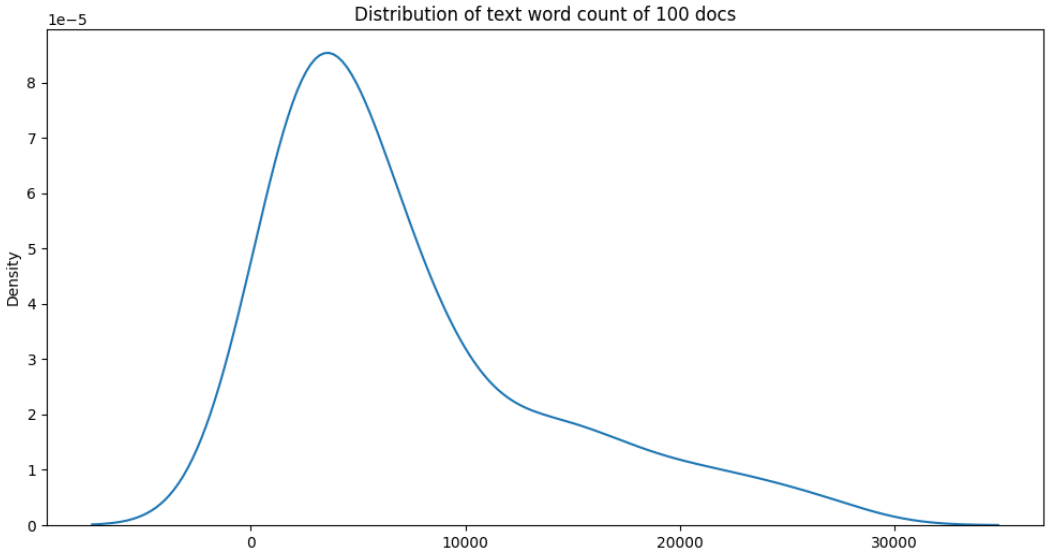
```
import numpy as np  # linear algebra

import pandas as pd  # data processing, CSV file I/O (e.g. pd.read_csv)

import seaborn as sns

import matplotlib.pyplot as plt

from IPython.display import display

import json
```

```
from common import get_answer, read_sample


print("---- 1. let us check the first document in the dataset ----")

df = read_sample(n=3)

print(df.head())


print("---- 2. Distribution of text word count of 100 docs ----")

df = read_sample(n=100)

doc_text_words = df["document_text"].apply(lambda x: len(x.split(" ")))

plt.figure(figsize=(12, 6))

sns.distplot(doc_text_words.values, kde=True, hist=False).set_title(

    "Distribution of text word count of 100 docs"

)

plt.savefig("reports/i2.png")


print("---- 3. let us check the first document in the dataset ----")




def myprocess(n=10):

    df = read_sample(n=n, ignore_doc_text=True)

    df["yes_no"] = df.annotations.apply(lambda x: x[0]["yes_no_answer"])

    df["long"] = df.annotations.apply(
```

```
        lambda x: [x[0]["long_answer"]["start_token"], x[0]["long_answer"]["end_token"]]
    )
    df["short"] = df.annotations.apply(lambda x: x[0]["short_answers"])
    return df



df = myprocess(500)
display(
    df.long.apply(
        lambda x: "Answer Doesn't exist" if x[0] == -1 else "Answer Exists"
    ).value_counts(normalize=True)
)

mask_answer_exists = (
    df.long.apply(lambda x: "Answer Doesn't exist" if x == -1 else "Answer Exists")
    == "Answer Exists"
)

print("---- 4. Distribution of Yes and No Answers ----")
yes_no_dist = df[mask_answer_exists].yes_no.value_counts(normalize=True)
print(yes_no_dist)
```

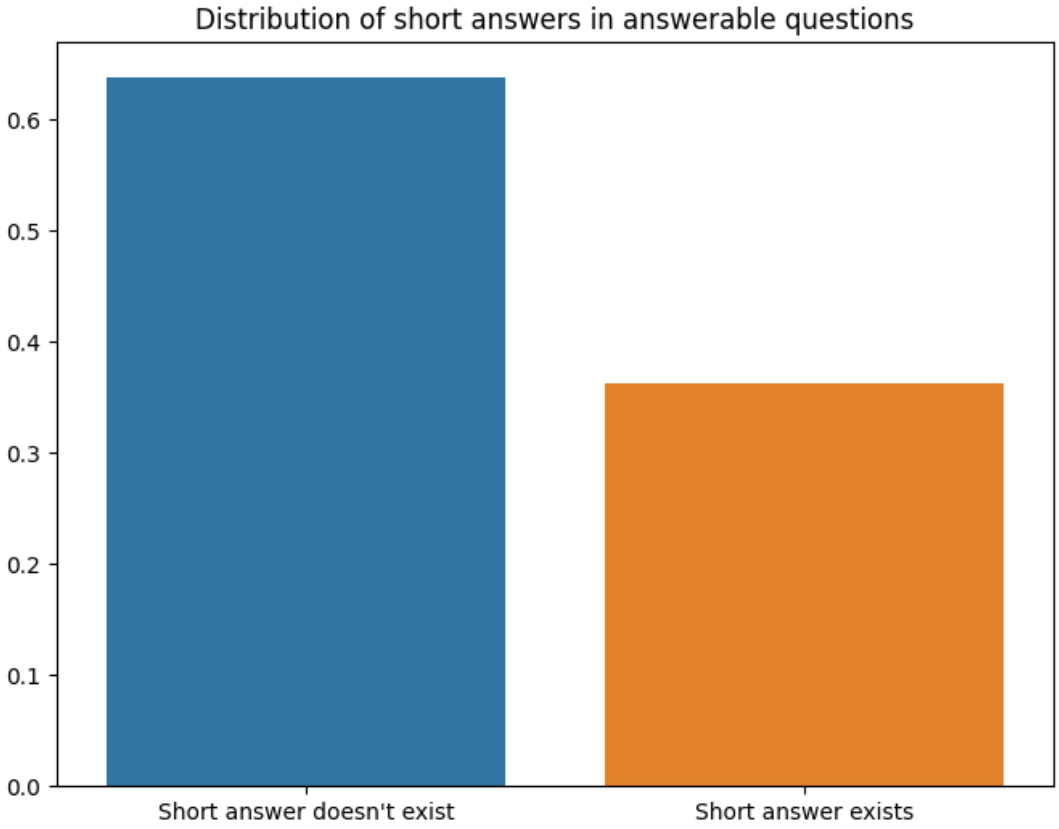
```
print("---- 5. short answers ----")
short_dist = (
    df[mask_answer_exists]
    .short.apply(
        lambda x: "Short answer exists" if len(x) > 0 else "Short answer doesn't exist"
    )
    .value_counts(normalize=True)
)
plt.figure(figsize=(8, 6))
sns.barplot(x=short_dist.index, y=short_dist.values).set_title(
    "Distribution of short answers in answerable questions"
)
plt.savefig("reports/i5.png")



print("---- 6. multiple short answers to a question ----")
short_size_dist = df[mask_answer_exists].short.apply(len).value_counts(normalize=True)
short_size_dist_pretty = pd.concat(
    [
        short_size_dist.loc[
            [
                0,
                1,
```

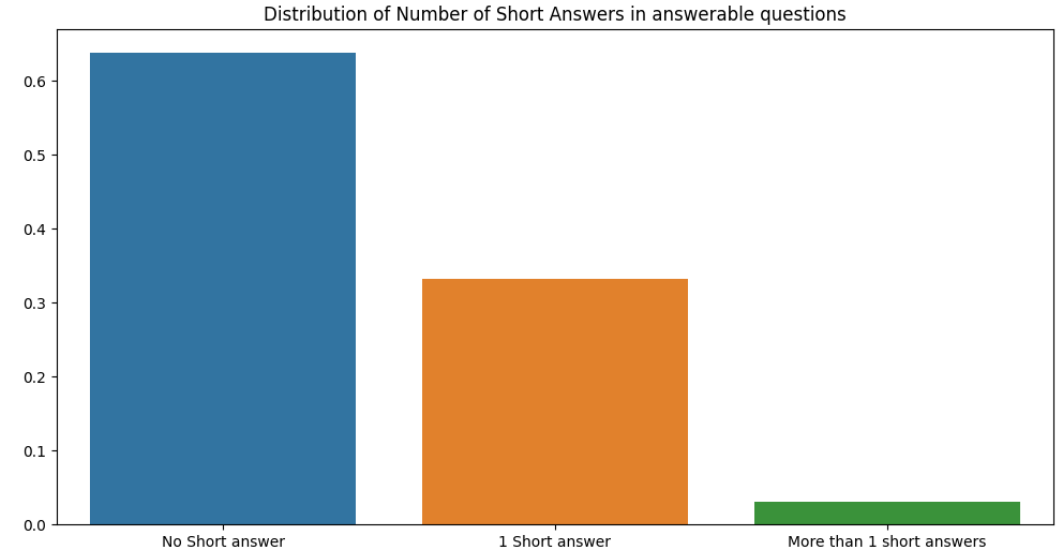
```
            ],
        ],
        pd.Series(short_size_dist.loc[2:].sum(), index=[">=2"]),
    ]
)
short_size_dist_pretty = short_size_dist_pretty.rename(
    index={
        0: "No Short answer",
        1: "1 Short answer",
        ">=2": "More than 1 short answers",
    }
)
plt.figure(figsize=(12, 6))
sns.barplot(x=short_size_dist_pretty.index, y=short_size_dist_pretty.values).set_title(
    "Distribution of Number of Short Answers in answerable questions"
)
plt.savefig("reports/i6.png")
```




```
---- 3. let us check the first document in the dataset ----
  0%|          | 0/499 [00:00<?, ?it/s]
Answer Exists           0.53
Answer Doesn't exist    0.47
```

```
---- 4. Distribution of Yes and No Answers ----
NONE    0.98
NO      0.01
YES     0.01
```





Model training

Main code is as following:

First we input related library we will use to train the model:

```
import os
import json
import gc
import pickle

import numpy as np
import pandas as pd
from tqdm import tqdm_notebook as tqdm
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input,
    Dense,
    Embedding,
    SpatialDropout1D,
    concatenate,
    Masking,
)
from tensorflow.keras.layers import LSTM, Bidirectional, GlobalMaxPooling1D, Dropout
from tensorflow.keras.preprocessing import text, sequence
from tqdm import tqdm_notebook as tqdm
import fasttext
import gensim
```

Then we build our train process pipeline , and split the dataset into train / test .

Main code is as following:

```
def my_build_train(train_path, n_rows=200000, sampling_rate=15):
    with open(train_path) as f:
        processed_rows = []

        for i in tqdm(range(n_rows)):
            line = f.readline()
            if not line:
                break

            line = json.loads(line)

            text = line["document_text"].split(" ")
            question = line["question_text"]
            annotations = line["annotations"][0]

            for i, candidate in enumerate(line["long_answer_candidates"]):
                label = i == annotations["long_answer"]["candidate_index"]

                start = candidate["start_token"]
                end = candidate["end_token"]
```

```
            if label or (i % sampling_rate == 0):
                processed_rows.append(
                    {
                        "text": " ".join(text[start:end]),
                        "is_long_answer": label,
                        "question": question,
                        "annotation_id": annotations["annotation_id"],
                    }
                )

        train = pd.DataFrame(processed_rows)

        return train


def my_build_test(test_path):
    with open(test_path) as f:
        processed_rows = []

        for line in tqdm(f):
            line = json.loads(line)

            text = line["document_text"].split(" ")
            question = line["question_text"]
            example_id = line["example_id"]

            for candidate in line["long_answer_candidates"]:
                start = candidate["start_token"]
                end = candidate["end_token"]

                processed_rows.append(
                    {
                        "text": " ".join(text[start:end]),
                        "question": question,
                        "example_id": example_id,
                        "sequence": f"{start}:{end}",
                    }
                )

        test = pd.DataFrame(processed_rows)

    return test
```

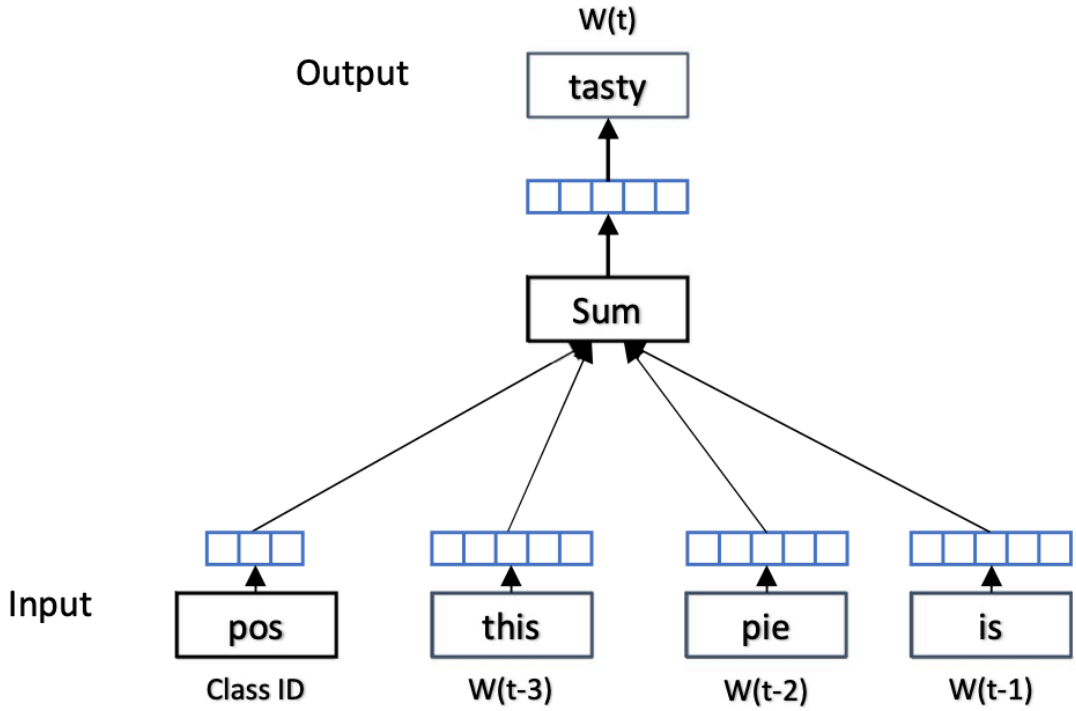
Then we build embedding matrix , which is curial to our training pipeline.

```
def build_embedding_matrix(tokenizer, path):
    embedding_matrix = np.zeros((tokenizer.num_words + 1, 300))

    # only this way worked in new version
    ft_model = gensim.models.KeyedVectors.load_word2vec_format(path, binary=False)

    # ft_model = fasttext.load_model(path)

    for word, i in tokenizer.word_index.items():
        if i >= tokenizer.num_words - 1:
            break
        # embedding_matrix[i] = ft_model.get_word_vector(word)
        if word in ft_model:
            r = ft_model[word]
            embedding_matrix[i] = r

    return embedding_matrix
```

Basically, the process of learning a document representation is not only conditioned by words appearing within a document, but since a document vector is used to serve as an additional context parameter during training, the process also has an effect on the values of resulting word vectors as the document parameter contributes to the prediction process of each word in the learning iterations. In other words, a document vector act as a memory of the topic of the current document, and

this information about the topic has an effect on the resulting word embeddings. The result is that vectors of words that are present more frequently in a document are highly affected by that document.

In the context of this work, classes or categories replace documents as input to the fine-tuning model; in a sense, this could be thought of as ‘class2Vec’.



embeddings model using Doc2Vec, and using the label/class of the document as an additional input in place of a document or paragraph id, words that are indicative of a given class, will have that information encoded in their embeddings.

Consequently, we have the following observations about learning word embeddings with distributed representation of documents (Doc2Vec):

If a certain word is present in a class exclusively, its embedding will be highly affected by its class in the training phase.

Similarly, if a certain word is present among several classes, the effect of individual classes on the embedding of that word will be diminished.

If a word is present in two or more classes, its embedding will be more influenced by the class in which it appeared with a higher frequency.

Finally, the learned word embeddings are domain specific, so the word vectors that were trained for a particular domain (e.g. sentiment analysis or emotion detection) may not be suitable for other domains (e.g. question classification).

According to the previous observations, the distributed representation of documents (Doc2Vec) method can be employed to learn word embedding features that are more relevant to the task at hand.

So, in the proposed method, all text fragments (e.g. tweets) related to a specific class (e.g. a given topic, and emotion or sentiment class (love, sadness, fear, etc.)) are labeled with the name of the class to which they belong. Each text fragment is then passed to the Doc2Vec model with its class identifier instead of a document identifier in order to learn the vectors of both words and classes. In this case, only word vectors are used after the training process, while class vectors are discarded. The exact steps for carrying out this process, are formally described as follows:

Given some input training corpus T containing input instances ti to tn where n is the number of all training instances in T and where each instance ti has a label c where c ? C and given a set of pre-trained word embeddings W where each word wi is represented with a vector wvi of length m, the proposed approach can be summarized in the following steps:

1. Load a pre-trained word embeddings model W which has a vocabulary set V

2. Load training data T

3. Extract vocabulary set VT from T (this will contain all unique words that appeared in all documents in T)

4. Identify words that appear in VT but not V (previously unseen words in the pre-trained model)
   Vunseen = VT - V

5. For each word wi in Vunseen, initialize a random word vector wvunseen_i of length m, and append it to W

6. Initialize a doc2vec model of vocabulary size | V ? VT| with weights from W (as updated in the previous step)

7. Set the number_of_epochs parameter for training the doc2Vec model to a number that is large enough to fine tune pre-trained embeddings, but which is not too large as to significantly change the pre-trained word vectors. In the experiments presented later in this paper, this parameter was empirically set to 10.

8. Use each training instance ti with class ci, as a training instance for doc2Vec

9. Save the resulting embeddings as W'. W' now contains the fine tuned set of embeddings.

—-

We then begin to construct LSTM model .

There are 3 main stages inside LSTM:

1. Forgetting the stage. This stage is mainly to selectively forget the input from the previous node. To put it simply, "forget the unimportant and remember the important".

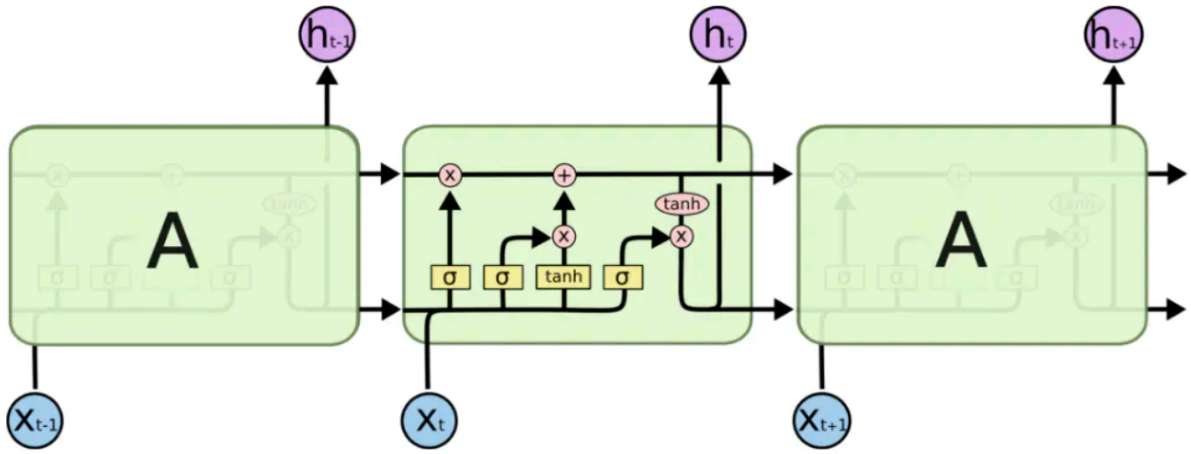
Specifically, the calculated [formula] (f means forget) is used as the forget gate to control the [formula] of the previous state which needs to be left and which needs to be forgotten.

2. Select the memory stage. This stage selectively "memorizes" the input of this stage. The main purpose is to select and memorize the input [formula]. Record what is important, and write less of what is not. The current input content is represented by the [formula] calculated above. The selected gating signal is controlled by [formula] (i stands for information).

Add the results obtained in the above two steps to get the [formula] that is transferred to the next state. This is the first formula in the figure above.

3. Output stage. This stage will determine which will be regarded as the output of the current state. It is mainly controlled by [formula]. It also scales the [formula] obtained in the previous stage (changes through a tanh activation function).

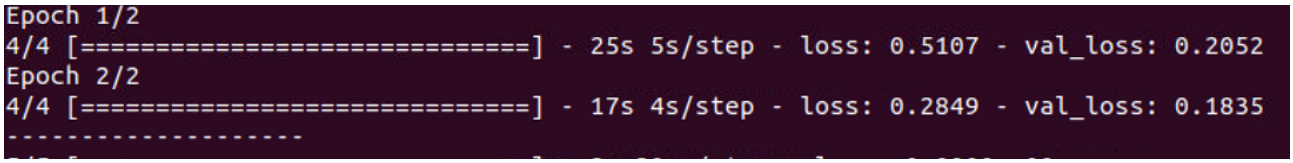
Similar to ordinary RNNs, the output [formula] is often finally obtained through [formula] changes.

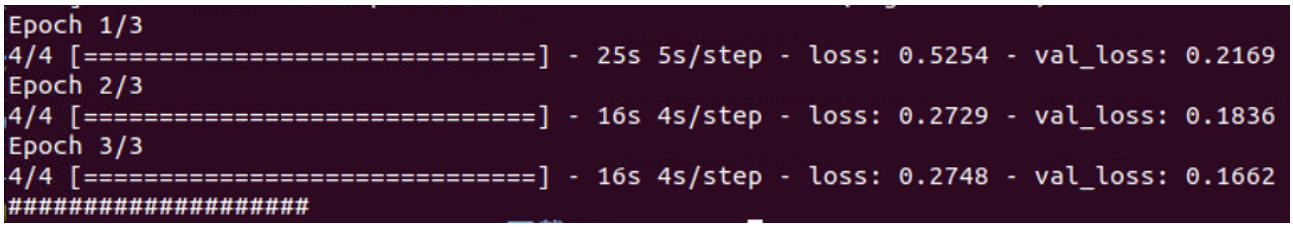


We use Keras to construct the LSTM model by init with embedding matrix , which is describe above.

```
def construct_model(embedding_matrix):
    embedding = Embedding(
        *embedding_matrix.shape,
        weights=[embedding_matrix],
        trainable=False,
        mask_zero=True,
    )

    q_in = Input(shape=(None,))
    q = embedding(q_in)
    q = SpatialDropout1D(0.2)(q)
    q = Bidirectional(LSTM(100, return_sequences=True))(q)
    q = GlobalMaxPooling1D()(q)

    t_in = Input(shape=(None,))
    t = embedding(t_in)
    t = SpatialDropout1D(0.2)(t)
    t = Bidirectional(LSTM(150, return_sequences=True))(t)
    t = GlobalMaxPooling1D()(t)

    hidden = concatenate([q, t])
    hidden = Dense(300, activation="relu")(hidden)
    hidden = Dropout(0.5)(hidden)
    hidden = Dense(300, activation="relu")(hidden)
    hidden = Dropout(0.5)(hidden)

    out1 = Dense(1, activation="sigmoid")(hidden)

    model = Model(inputs=[t_in, q_in], outputs=out1)
    model.compile(loss="binary_crossentropy", optimizer="adam")

    return model
```

Then we train all the epochs :





After we finished training, we then save the model into local cache, we can use latter in the other part:

```
with open("tokenizer.pickle", "wb") as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
model.save(“model.h5")
```

We then construct test pipeline:

```
def construct_test(test_path):
    with open(test_path) as f:
        processed_rows = []

        for line in tqdm(f):
            line = json.loads(line)

            text = line["document_text"].split(" ")
            question = line["question_text"]
            example_id = line["example_id"]
            for candidate in line["long_answer_candidates"]:
                start = candidate["start_token"]
                end = candidate["end_token"]

                processed_rows.append(
                    {
                        "text": " ".join(text[start:end]),
                        "question": question,
                        "example_id": example_id,
                        "PredictionString": f"{start}:{end}",
                    }
                )

        test = pd.DataFrame(processed_rows)

    return test
```

We can then test our previous saved model use this test pipeline:

model = load_model("model.h5")

```
__________________________________________________________________________
_________________________
dense (Dense)                (None, 300)               150300      concaten
ate[0][0]
__________________________________________________________________________
_________________________
dropout (Dropout)            (None, 300)               0           dense[0]
[0]
__________________________________________________________________________
_________________________
dense_1 (Dense)              (None, 300)               90300       dropout
[0][0]
__________________________________________________________________________
_________________________
dropout_1 (Dropout)          (None, 300)               0           dense_1
[0][0]
__________________________________________________________________________
_________________________
dense_2 (Dense)              (None, 1)                 301         dropout_
1[0][0]
```

```
    print("-> " * 10, model.summary())


    with open("tokenizer.pickle", "rb") as f:
        tokenizer = pickle.load(f)


    test_text, test_questions = compute_text_and_questions(test, tokenizer)


    # Evaluate the model on the test data using `evaluate`
    print("Evaluate on test data")
    results = model.evaluate([test_text, test_questions], batch_size=512)
    print("test loss, test acc:", results)


    test_target = model.predict([test_text, test_questions], batch_size=512)


    test["target"] = test_target
```

Then we get test results:

```
#                                   text  ... PredictionString
# 0  <Table> <Tr> <Th_colspan="2"> High Commission ...  ...          18:136
# 1  <Tr> <Th_colspan="2"> High Commission of South...  ...          19:30
# 2  <Tr> <Th> Location </Th> <Td> Trafalgar Square...  ...          34:45
# 3  <Tr> <Th> Address </Th> <Td> Trafalgar Square ...  ...          45:59
# 4  <Tr> <Th> Coordinates </Th> <Td> 51 ° 30 ′ 30 ...  ...          59:126
```

And the acc is 0.021 which is 2.1%.

Ch4 Conclusions and future work.

Review our process

I break down each document into parts corresponding to each of the candidate long answers. Then, I label each of the long answer to be `1` if it is the true long answer, or `0` if not. For each of row, I also include the question and the `example_id`. Then, I train a LSTM to predict that label. Used the embed matrix tuning well give me advantage.

Build Dataset: Two different utility functions for creating ready-to-use dataframes for both training and testing data. The build_train function only loads a subset of all training data, and set a sampling rate S, such that we only keep 1/S of all the negative-labelled data (and keep all positive-labelled data).

Preprocessing: Train a keras Tokenizer to encode the text and questions into list of integers (tokenization), then pad them to a fixed length to form a single numpy array for text and one for questions.

Modelling:

Generate a fasttext embedding (directly using FAIR's Official Python API) based on the index of the tokenizer.

Build two 2-layer bidirectional LSTM; one to read the questions, and one to read the text.

Concatenate the output of the LSTM and feed in 2-layer fully-connected neural networks.

Predict the binary output using Sigmoid activation.

Optimize using Adam and binary cross-entropy loss.

Save Model: Due to the submission time limit, it is better to import the model we just trained in a separate kernel to infer and submit. In the inference kernel, we first remove all the rows with less

than 0.5 confidence, then for each example_id we only keep the one with the highest confidence to be the output.

After we done all these improvements, our model is better than the Google benchmark model, as our ACC is acc is 0.021.

where we can improve

If I no longer concatenate candidate long answers, but directly apply a sliding window to each candidate long answer. In other words, each of my samples will not contain text from multiple candidate long answers, so I can use [CLS] to do classification tasks for a specific long answer.

In the future, a very important step is hard negative sampling. If I first used a model to obtain the predicted probabilities of all candidate long answers in the training set, and then sampled negative samples based on this probability. Each model was trained for 3-4 epochs. Compared to our direct random sampling of negative samples (the baseline only retains 2% of the negative samples), this method can retain training valuable data. I think this will improve the model.

References

Chris Alberti, Kenton Lee, and Michael Collins. 2019. A BERT Baseline for the Natural Questions. arXiv preprint

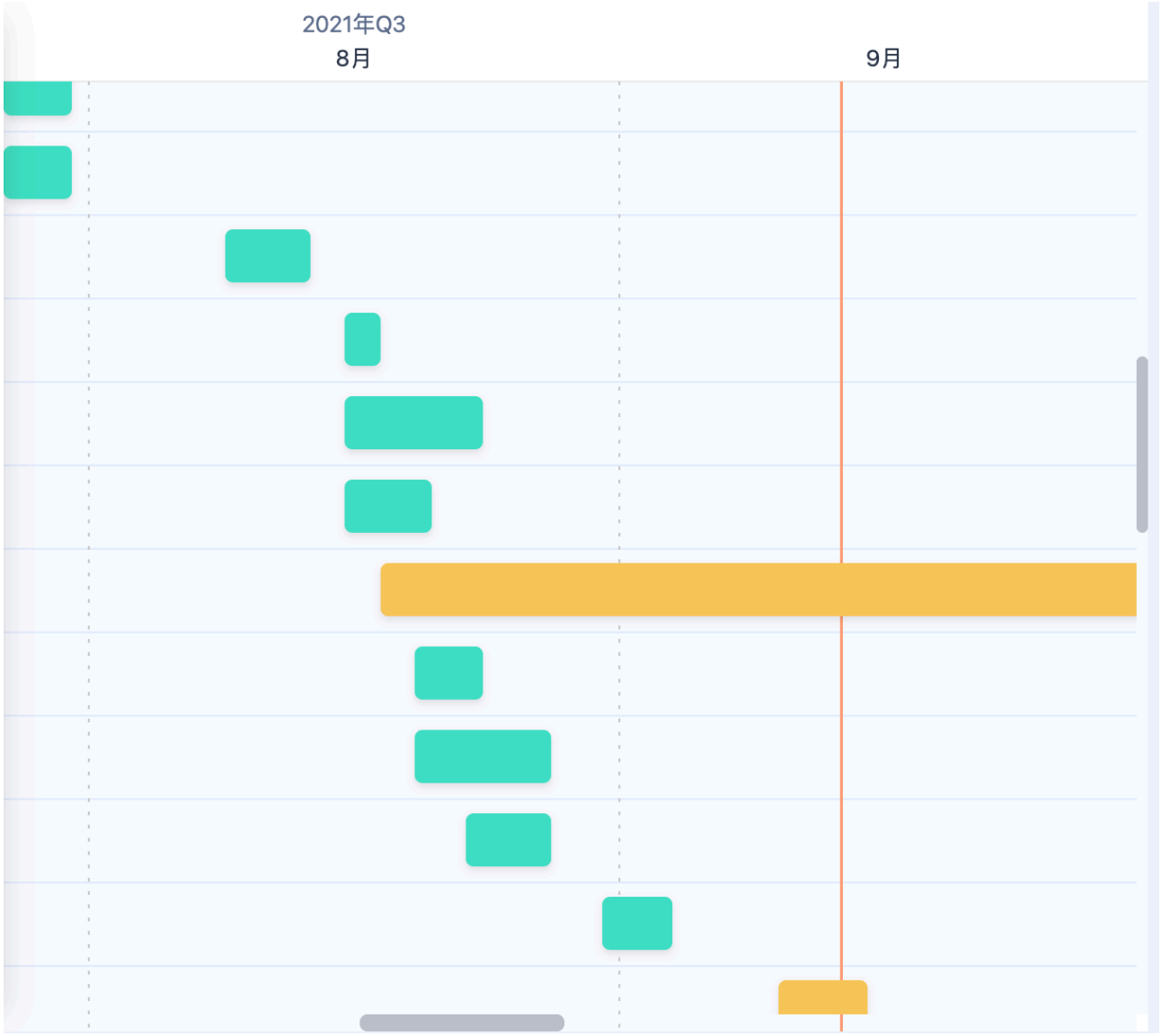
Siva Reddy, Danqi Chen, and Christopher D Manning. 2018. Coqa: A conversational question answering challenge. arXiv preprint arXiv:1808.07042.

Hands-on Question Answering Systems with BERT (https://www.apress.com/gp/book/9781484266632)

Young T, Hazarika D, Poria S, Cambria E. Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. IEEE Computational Intelligence Magazine. 2018 Aug; 13(3).

Wang, J. Xu, B. Xu, Liu, Zhang, Wang, Hao. Semantic Clustering and Convolutional Neural Network for Short Text Categorization. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers); 2015; Beijing. p. 352–357

Ch5 Project management.



We first plan my project to be :
Data explore
Algorithm explore
Find right model to use to process NLP task

Then we divide the 3 projects into stories:

1. Handle all the data-preprocess
2. Install all related libuaries
3. Use numpy and pandas to process dataset
4. Use statics and marplot to visual dataset
5. Find right dataset and algorithm related to NLP
6. Try CNN/LSTM/BERT/Bayes method/model to do question-answer system
7. Test all models and model construct on your chosen dataset

8. Turning our algorithm and models to get better results

Ch6 Self-review

evaluation of my progress, development and how the project went

We have finished the NLP project to predict answers (long and short) from wiki articles.

The implementation process is tortuous, and the time has exceeded my original expectation by half a month. The main time is spent on model tuning, and the combination of various models to complete a task. Model adjustment and optimization take more time than expected.The machine training time is also much longer than expected, often 17G data needs to run overnight in rtx3090ti to complete one-time completed training .

All the hard work is worth it: as understanding wiki-articles, find answers in article to answer the question, there is still a huge space waiting for everyone to explore. At present, machine learning is still lagging behind human beings.