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 on the ai.google.com specially at: research/NaturalQuestions/ dataset)

Open Domain Question and Answering

A key goal of artificial intelligence is to build a system that can be read by itself, a system that can find the corresponding paragraph in the text as an answer. These question-and-answer (QA) systems may make huge changes to the way we obtain information/search for information. In addition, open domain question and answer is an important direction for the development of artificial intelligence, because understanding the text and being able to answer questions about it is what we usually think of as "intelligence" is highly relevant.

Natural problem dataset

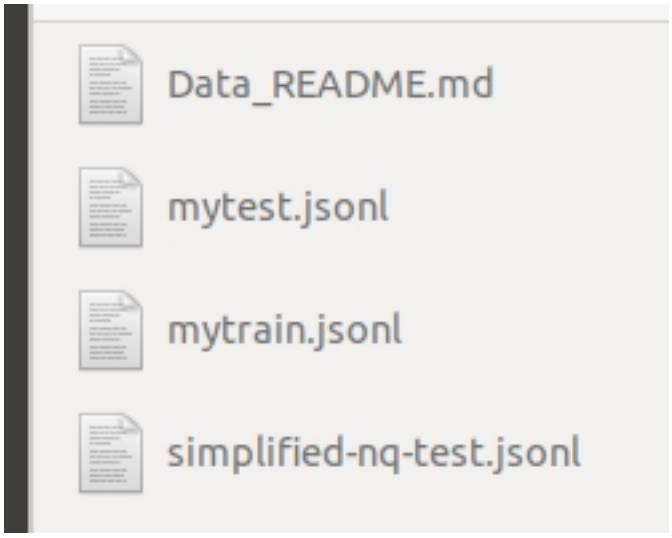
In order to promote the rapid development of open domain question and answer, Google has created a natural question (NQ) corpus. The NQ corpus contains questions from real users, and the corpus requires articles about questions and answers. And there is a marked correct answer and the starting position of the correct answer.

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

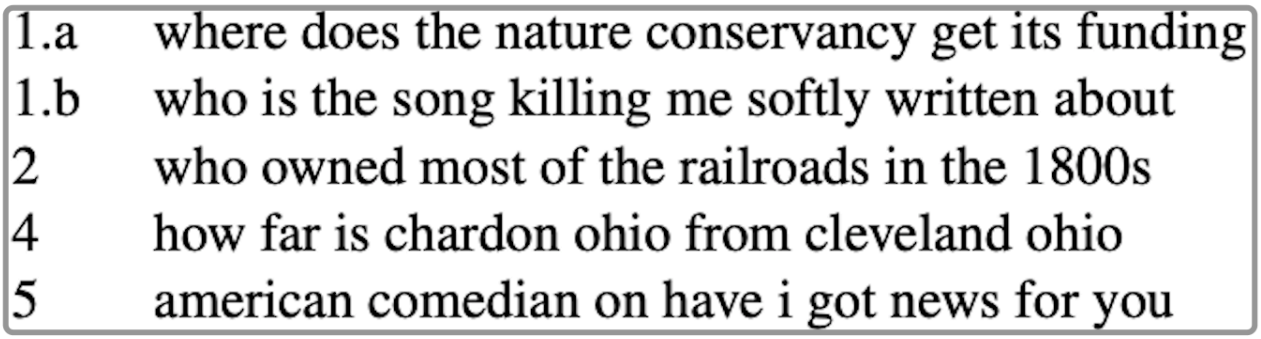
The answer is entirely in the original text;

The answers appear in multiple articles;

Part of the answer appears in the original text and part of the answer appears in the question;

Part of the answer appears in the original text, and the other part is the generated new words;

The answer does not appear in the original text at all (Yes / No type)



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.

During the entire tagging process, the tagger needs to read the entire Wikipedia page to see if there is an answer to this question. Then, on the one hand, find the long answer natural paragraph that contains the required information, and on the other hand, find one or two words or Phrase as a short answer, the accuracy of the entire data set is more than 90%.

All the questions in the data set are asked by users when using Google search. The QA question and answer system needs to read the entire Wikipedia article, and the answer to every question may not be found. Therefore, NQ is better than the previous one. The QA data set is more challenging.

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

My goal is to predict whether the answer to the Wikipedia article question is long or short, what content each is, and where to start. According to the official statement, our final result is to use micro F1 to evaluate the accuracy between the prediction and the expected answer. The predicted long answer and short answer must be exactly the same as the tag index of one of the real tags to be correct . 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) this competition's metric does not use confidence scores to find an optimal threshold for predictions.

Where F1's definition is as following:

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

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

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

precision, that is, the ratio of the correct number to the entire result in the results returned after retrieval.

Recall: recall rate, that is, the ratio of the number of true correct numbers in the search results to the number of true correct numbers in the entire data set (retrieved and not retrieved).

FN: False Negative, it is judged as a negative sample, but in fact it is a positive sample.

FP: False Positive, it is judged as a positive sample, but in fact it is a negative sample.

TN: True Negative, is judged as a negative sample, in fact it is also a negative sample.

TP: True Positive, it is judged as a positive sample, in fact it is also a proof sample.

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.

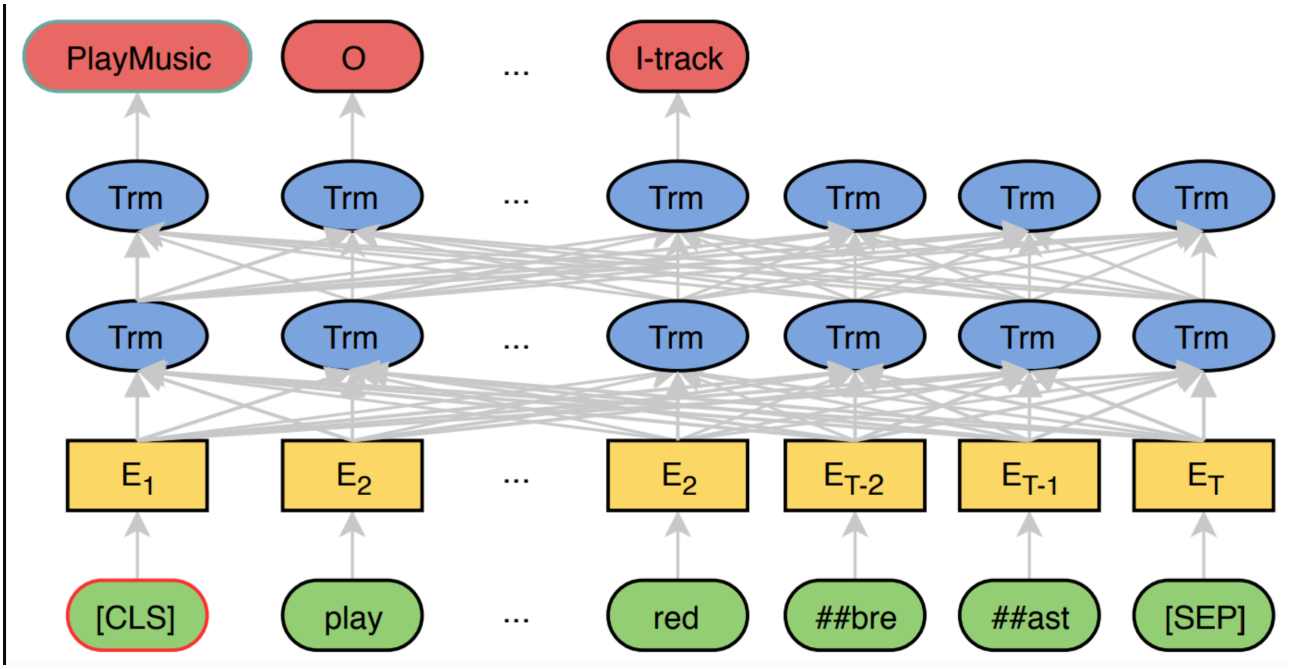


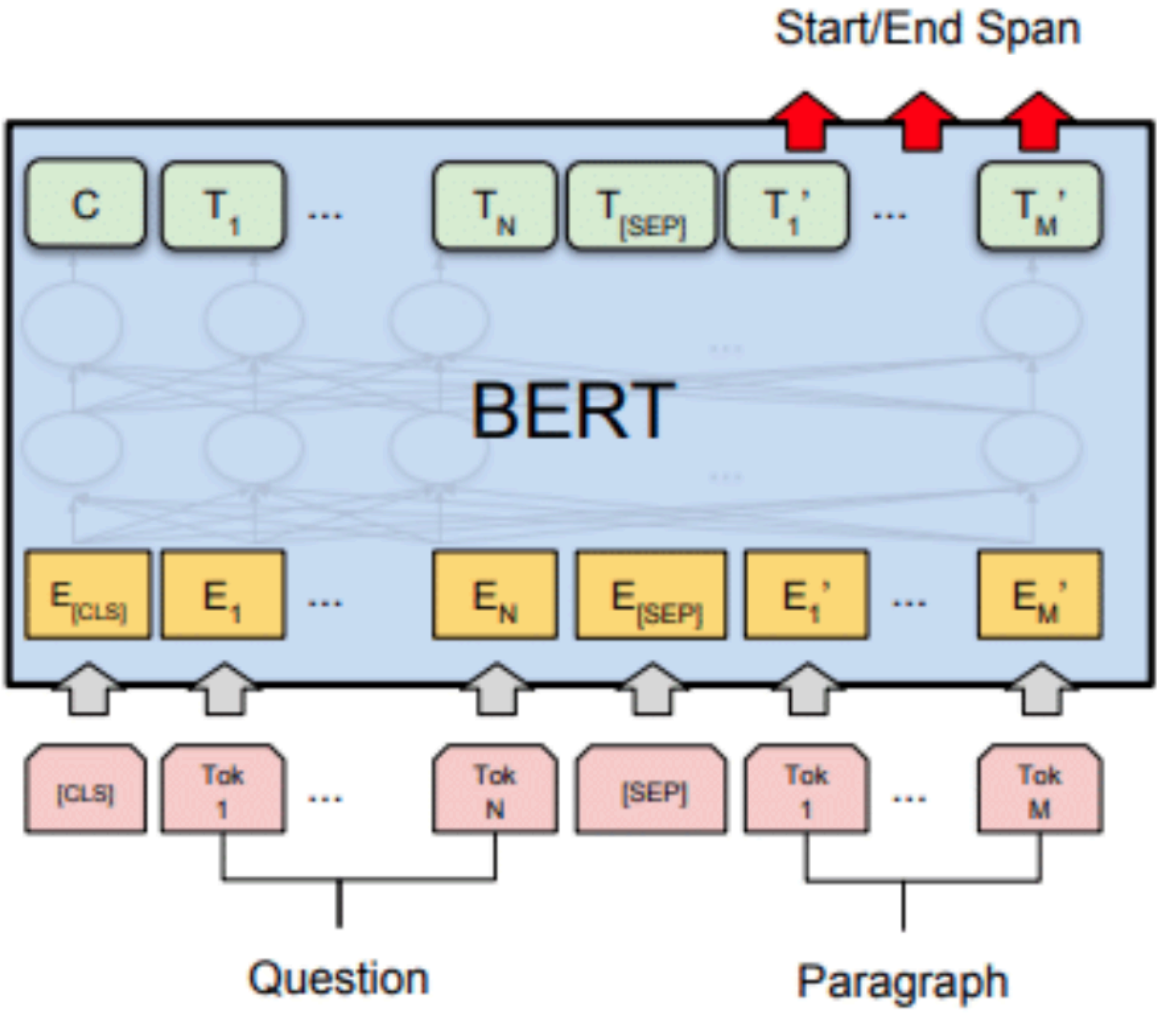
image of JointBERT

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.



In order to be consistent with subsequent tasks, bert enters the original word or a random word in a certain proportion of the word position that needs to be predicted. Of course, because only part of the words in the input text sequence are used for training, the efficiency of BERT will be lower than that of ordinary language models. The author also pointed out that the convergence of BERT requires more training steps. Another innovation of BERT is the addition of a sentence-level continuity prediction task based on the two-way language model. The goal of this task is also very simple. It is to predict whether the text at both ends of the input BERT is continuous text. The author pointed out that introducing this task can better allow the model to learn the relationship between continuous text fragments. During training, the second segment of the input model will be randomly selected from all texts with a probability of 50%, and the subsequent text of the first segment will be selected with a probability of 50%.

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

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

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:

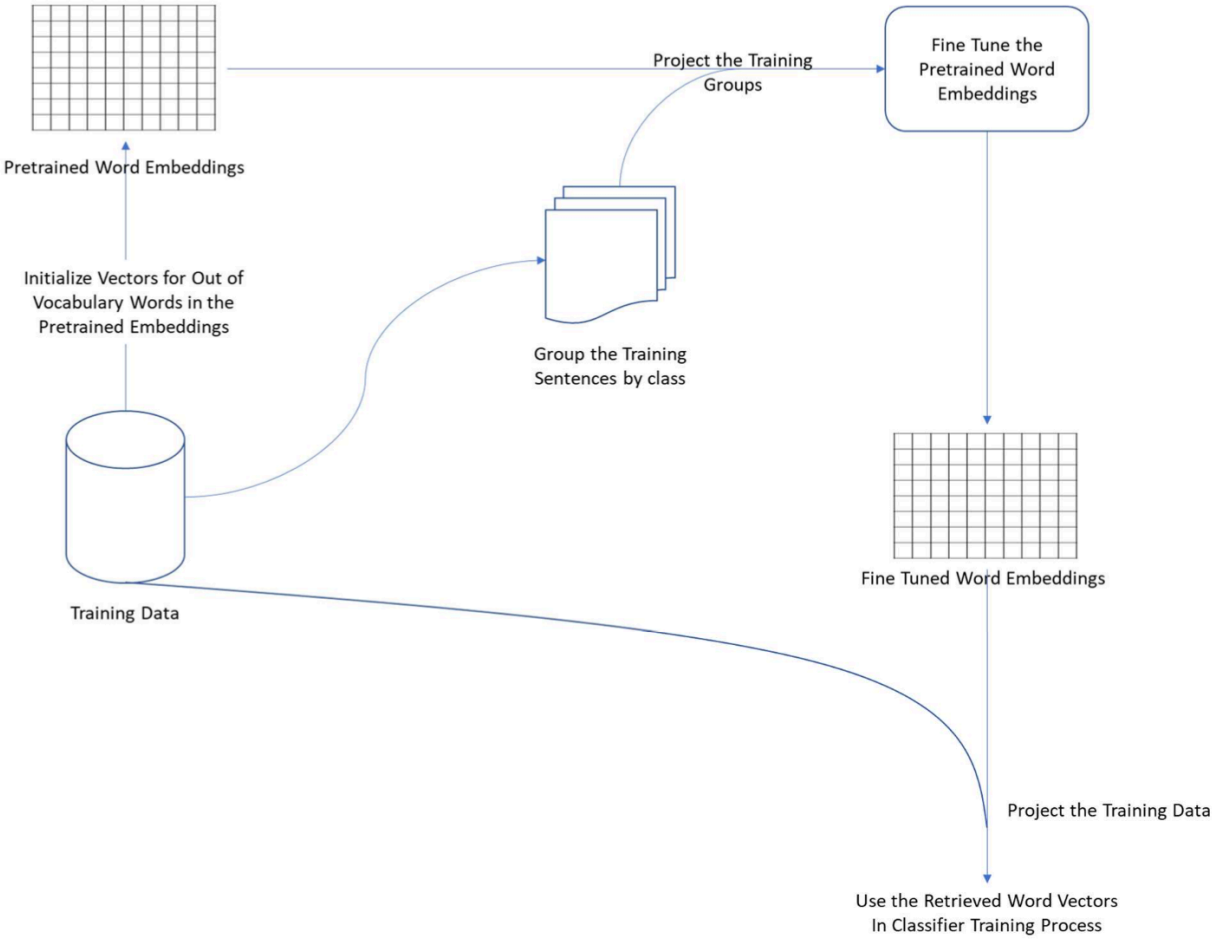
We break down the entire article into small pieces, and then use an LSTM to predict the small tags of the answers.

I will use the correlation model as a new simple method for fine-tuning pre-trained word embeddings for text classification tasks. In this method, the class in which the term appears serves as an additional context variable in the fine-tuning process and contributes to the final word vector of the term. Therefore, words that are specifically used in a particular category will have vectors that are closer to each other in the embedding space, and are more distinguishing for that category.

Use pre-trained word vectors to initialize the wiki-news-300d-1M model before training. Because the computer can only perform numerical calculations. The simplest method is one-hot. If there are 10,000 words in total, then the word vector will have 10,000 dimensions, the dimension corresponding to the word is 1, and the others are 0, but such a representation dimension is too high and too sparse. So later I began to study using a dense vector with a small dimension to represent it. Nowadays, the word vector is generally 128, 200 or 300 dimensions, which is very small. Pre-training means that this kind of word vector is trained in advance, which corresponds to some tasks that can be input. The word id, and then the word vector is trained internally for specific tasks. The word vector obtained in this way is not universal, and the pre-trained word vector is the result of training on a large sample, which has good universality, no matter what All tasks can be used directly.

wiki-news-300d-1M.vec trains a model in the context of each word so that similar words will have similar numerical representations.

Just like a normal feedforward network, where you have a set of independent variables and a target dependent variable that you are trying to predict, you first decompose your sentence into words (tokenize), and create some word pairs based on the window size. One of the combinations can be a pair of words, such as (cat, purr) where cat is the independent variable (X) purr is the target dependent variable we want to predict (Y)



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

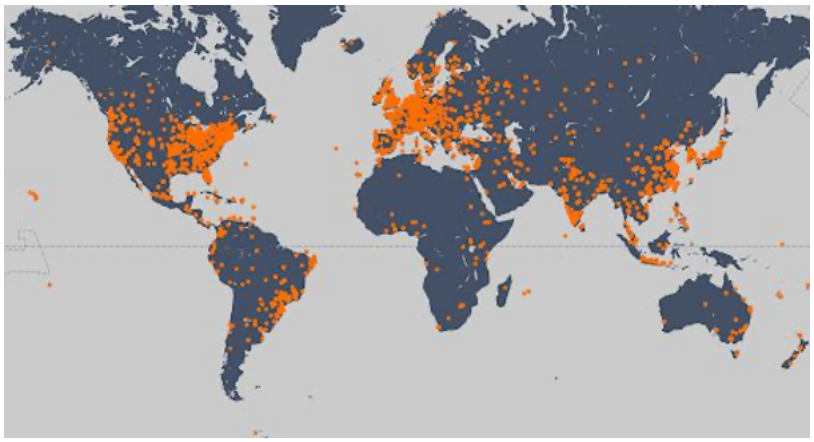
sklearn

seaborn

tensorflow-gpu

Development Tech Stack we choose
Introduction of development technology and tools

a. tensorflow development framework



B. keras

C. NumPy

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:

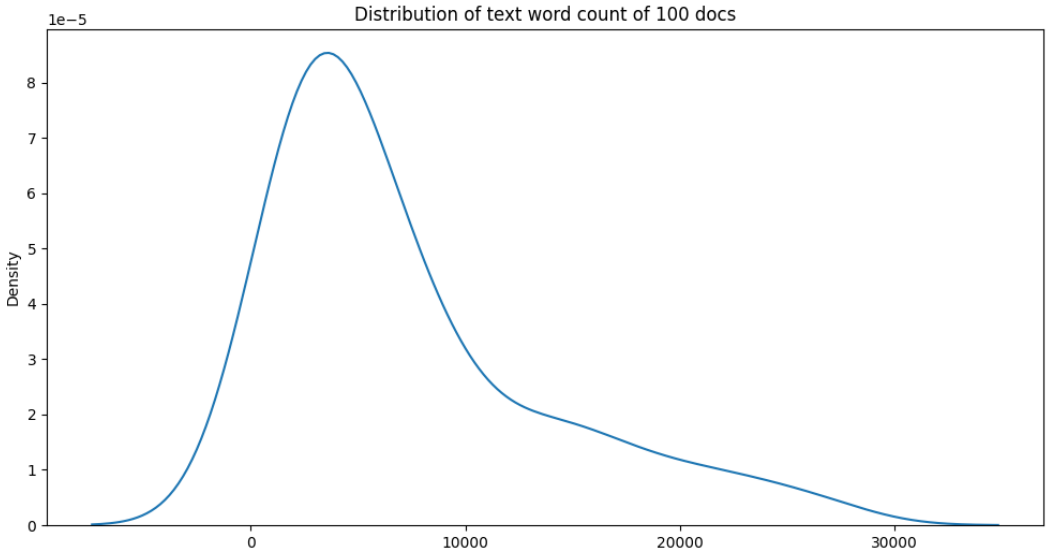
```
from common import get_answer, read_sample


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"
)
```

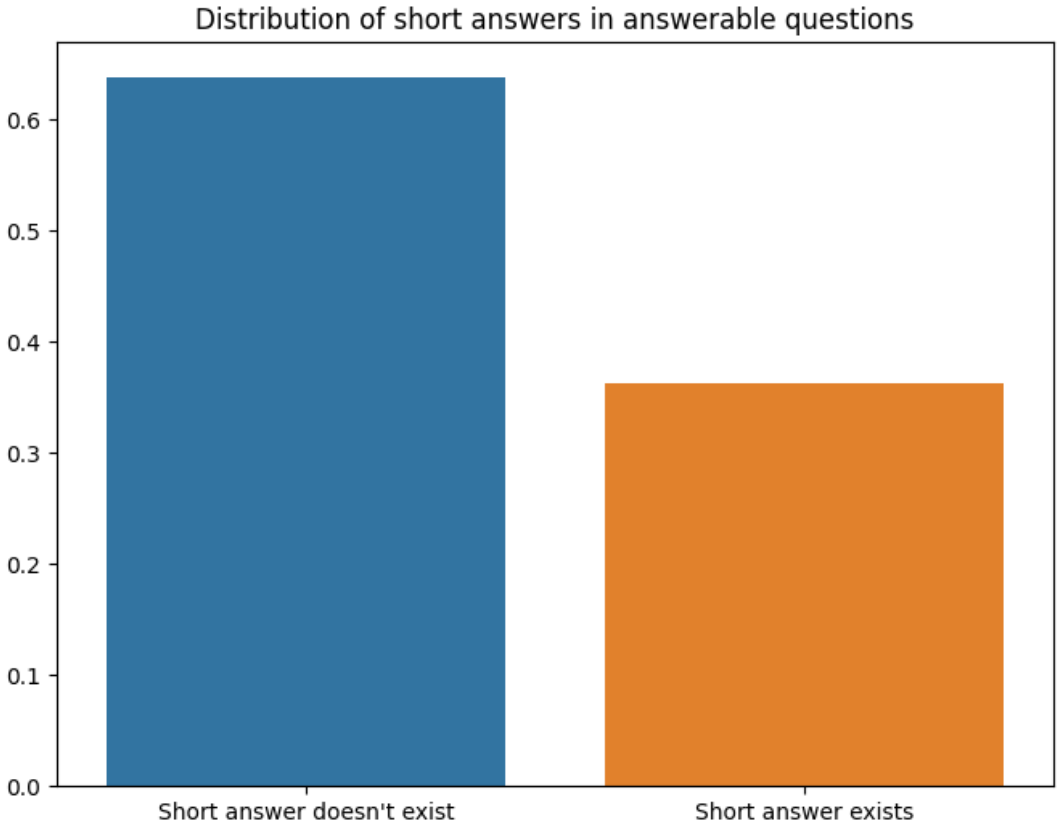
```
plt.savefig("reports/i5.png")



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",
    }
)
```

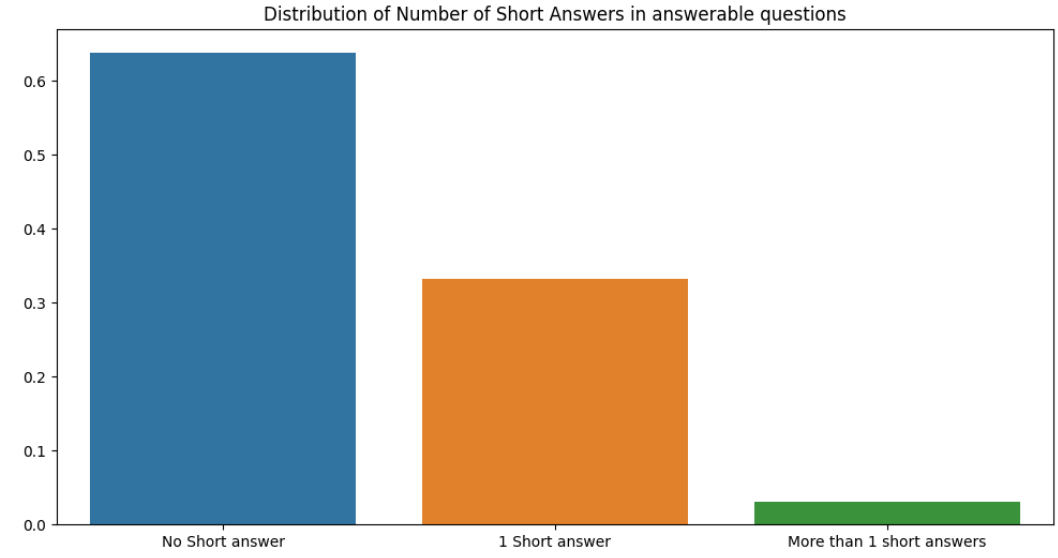
```
plt.figure(figsize=(12, 6))

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:

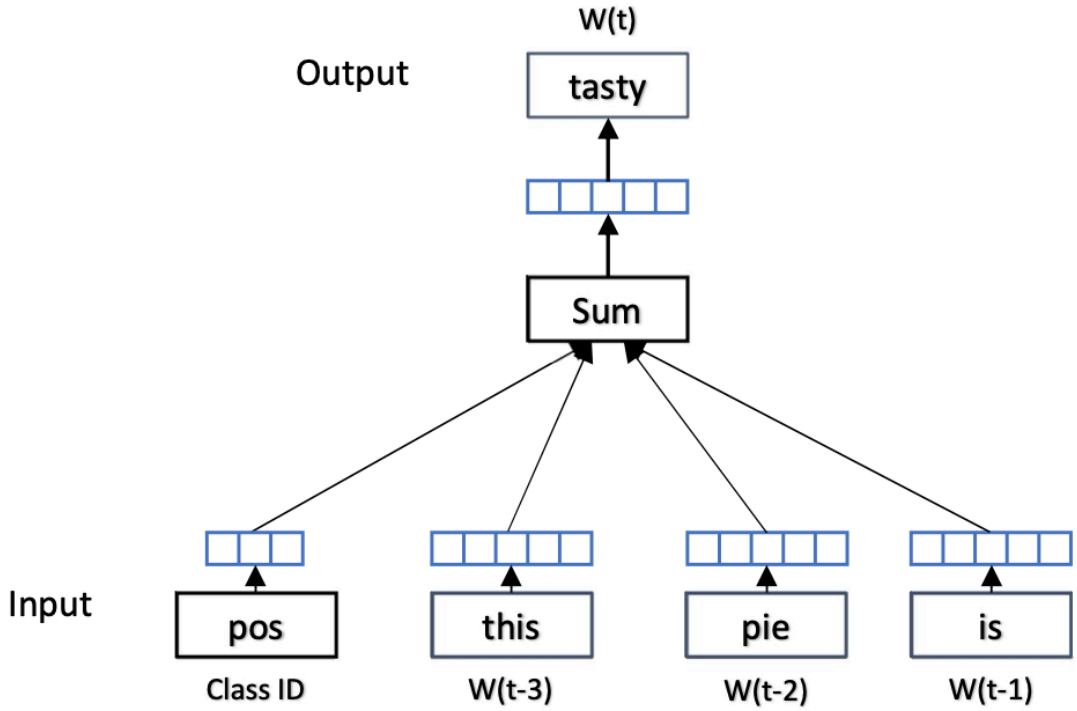
```
def my_build_train(train_path, n_rows=200000, sampling_rate=15):
    with open(train_path) as f:
        processed_rows = []

        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
```

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

```python
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
```

The pre-trained word vector will contain certain semantic information, and the specific semantic information depends on the objective function used in the pre-training. For example, the objective function used in skipgram training is "observe a word, predict the words in the window around it, and this prediction is as close as possible to the sample (that is, maximize log likelihood)", so the word vector meaning of word2vec is "context information ". (This is why word2vec can't distinguish antonyms well, because their contexts are very close). Word vectors like BERT get more data and use more complex objective functions to contain richer word meaning information. In terms of specific usage, what are the advantages of pre-training word vectors over random word vectors.

The impact of unsupervised pre-training on deep learning is summarized empirically from experiments. Pre-training plays a role similar to regularization, that is, the performance in the training set may not be better than random initialization, but it can be very Effectively prevent overfitting, so it will perform better on the test set. As for the convergence speed, I think that if the pre-training data set has some commonalities with the data set and task you use, it can help the position of the word vector in the parameter space be closer to the local optimum.



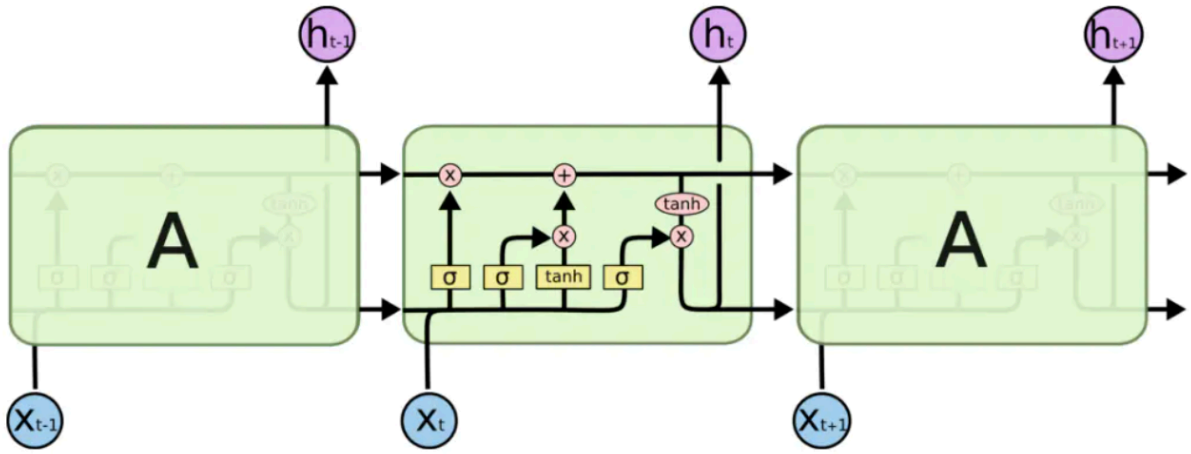
—-

We then begin to construct LSTM model .

LSTM mainly solves the long-term dependent learning problem through the gate mechanism. LSTM removes or adds information to the cell state through a well-designed gate structure. The gate is a way to let the information selection through, it contains a sigmoid neural network layer and a bitwise multiplication operation. The sigmoid layer can output a value from 0 to 1, which is used to control the amount of output of each part to the cell state. Among them, 0 means "no amount is allowed to pass", and 1 means "any amount is allowed to pass". The LSTM has three gates to control the cell state.

LSTM controls the transmission of information by introducing three gate structures. First, the forget gate is used to control the input level of cell state C (long-term information); then the input gate is used to control the new input level; finally, the output gate is used to control the final output level

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 :

```
Epoch 1/2
4/4 [==============================] - 25s 5s/step - loss: 0.5107 - val_loss: 0.2052
Epoch 2/2
4/4 [==============================] - 17s 4s/step - loss: 0.2849 - val_loss: 0.1835
--------------------
```

```
Epoch 1/3
4/4 [==============================] - 25s 5s/step - loss: 0.5254 - val_loss: 0.2169
Epoch 2/3
4/4 [==============================] - 16s 4s/step - loss: 0.2729 - val_loss: 0.1836
Epoch 3/3
4/4 [==============================] - 16s 4s/step - loss: 0.2748 - val_loss: 0.1662
####################
```

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

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

The LSTM we use to predict answer type and start / end index is improve the F-score of test data. I think the problem that LSTM has better performance than other networks mainly has the following characteristics:

It belongs to time series data (time dependent) and requires global processing;

The correspondence between input and output element levels (such as between words and words) may have a large time span;

The data is not too short but not too long, for example, less than 1000 steps.

Predict the binary output using Sigmoid activation.

Optimize using Adam and binary cross-entropy loss.

The advantage of LSTM is that a language model is implicitly established when modeling, which is a strong distinction between word order. Two-way lstm is stronger, and a language model is also built for reverse, which not only distinguishes between forward word order and reverse word order, so that for each step of the LSTM, it is equivalent to having a complete context modeling.

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

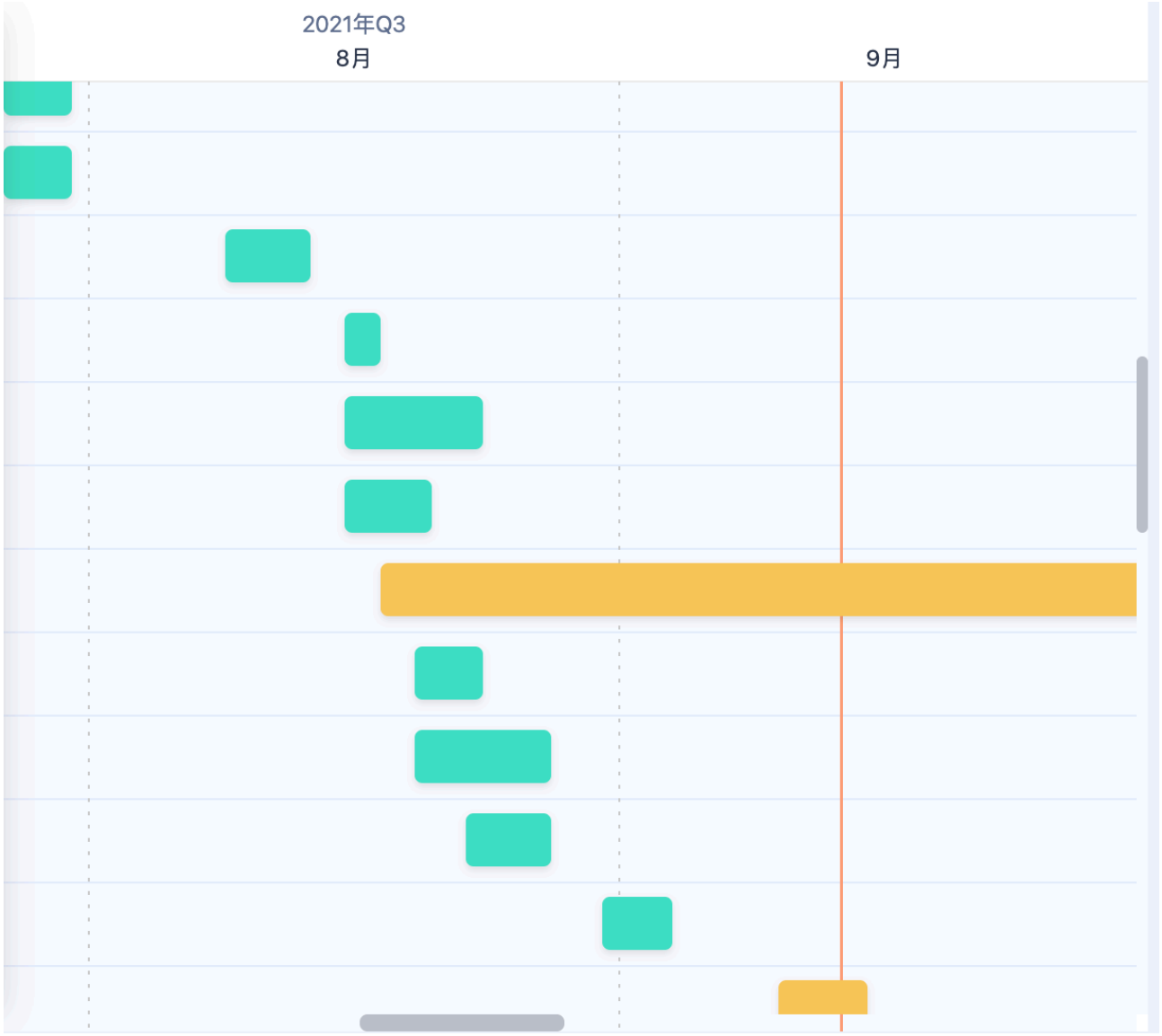
Siva Reddy, Danqi Chen, and Christopher D Manning. 2018. Coqa: A conversational question answering challenge. arXiv preprint arXiv:1808.07042.

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.