Project Topic

Question Answering Pipeline using Vertex AI and Haystack

Submitted by:

Teepika Ramasamy Marimuthu

Duncan Inganji

Shreya Goyal

Jagruti Mohanty

Table of Contents

Abstract

Question Answering System model helps generate answers for the questions in an interactive manner. In this project, we have implemented state-of-the-art deep learning models for Question Answering such as transformer models roberta and distilbert For training the model, we have used SQUAD and SELQA data sets. The implementation uses two approaches - Haystack extractive QA pipeline and vertex AI to deploy the endpoint. The project implementation resulted in an end-to-end production-ready pipeline for the Question answering model, i.e., it is a type of conversational AI.

Key Words: DistilBert , SQUAD , SELQA , Vertex AI , QuestionAnswering, Haystack

Introduction

The objective is to build a Question Answering model. One implementation uses Vertex AI, and the other one uses Haystack. Haystack is an end-to-end framework that enables users to have production-ready pipelines for different use cases such as question answering. Haystack is built in a modular manner that enables the use of open-source APIs such as Hugging Face BERT models. Haystack is an end-to-end framework that enables you to build powerful and production-ready pipelines for different search use cases. The implementation is Haystack uses a question answering pipeline consisting of Retriever and Reader to serve the requests. The model used is RoBERTa, and training data is a combination of SQuAD 2.0 and SELQA.

A question answering bot is useful in generating natural language for questions and answers. This facilitates human-like conversation between the machine and the human. With end-to-end continuous integration of the model, the AI model keeps improving. The need for using deep learning-based models for question answering bots is due to the multiple disadvantages of having scripted chatbots. The scripted bots cannot provide accurate responses if a customer uses a different way of questioning than what has been preprogrammed. The context of the question is not accurately determined for similar words. Hence we have implemented transformer-based models, i.e., state-of-the-art implementations in the field of natural language processing. The transformer-based encoder decorder neural networks learn to set context visa attention mechanisms.The training is based on SqUAD 2.0 and SELQA datasets.

There are two approaches that are implemented in the project.The other implementation is using Distil Bert and Vertex AI to deploy the endpoint of the question-answering session. Vertex AI Pipelines help in automating, monitoring, and orchestrating the ML workflow in a serverless manner and storing the artifacts using vertex ML metadata. The project uses the SQuAD dataset for training. Data is stored in the GCP cloud storage, and it is running with GPU devices in the project. The model is trained using distilled Bert for Question Answering, and the endpoint is deployed.

Related Work

In recent years, there has been remarkable progress in this field. Recent technology of natural language processing called BERT has reached superhuman performance in high resource languages for reading comprehension tasks. However, several researchers have stated that multilingual models are not enough for low-resource languages, since they are lacking a thorough understanding of those languages. In one of the research , the study compares both multilingual and Swedish monolingual inherited BERT models for question answering utilizing both an English and a Swedish machine translated SQuADv2 data set during its fine-tuning process. The models are evaluated with SQuADv2 benchmark and within an implemented question answering system built upon the classical retriever-reader methodology. This study introduces a naive and more robust prediction method for the proposed question answering system as well finding a sweet spot for each individual model approach integrated into the system. The question answering system is evaluated and compared against another question answering library at the leading edge within the area, applying a custom crafted Swedish evaluation data set. The results show that the fine-tuned model based on the Swedish pretrained model and the Swedish SQuADv2 dataset were superior in all evaluation metrics except speed.

Data

For vertex AI implementation, the dataset is SQUAD 2.0 dataset. Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or *span*, from the corresponding reading passage, or the question might be unanswerable. SQuAD2.0 combines the 100,000 questions in SQuAD1.1 with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones.

To create a haystack pipeline SELQA dataset is used. The SelQA dataset provides crowdsourced annotation for two selection-based question answer tasks, answer sentence selection and answer triggering. This dataset composes about 8K factoid questions for the top-10 most prevalent topics among Wikipedia articles.

Methods

Haystack is an open-source framework for building search systems that work intelligently over large document collections. Recent advances in NLP have enabled the application of question answering, retrieval and summarization to real world settings and Haystack is designed to be the bridge between research and industry. The objective of the project is to build a haystack index that supports question answering settings, being backed by our custom model fine tuned for our requirements. Our modeling starts with training roberta-base model on the SelQA dataset. The roberta-base model is a language model which is pretrained on SQuAD 2.0 dataset for Extractive QA downstream task. The model is specifically designed for solving English language NLP QA tasks.

We coupled Hugging face model training with Weights & Biases. With the integration, we are able to quickly train and monitor models for full traceability and reproducibility without any extra line of code. Also Weights & Biases help in monitoring and create consolidated dashboards for multiple runs of the same training pipeline which helps in visualizing the performance of each run with different hyperparameters. W&B helps in logging the model's configuration parameters like logging losses and metrics, logging gradients and parameter distributions, keeping track of the source code and also logging the system metrics (GPU, CPU, memory utilization).

The training starts with preprocessing steps of the input text. Some of the questions have lots of whitespace on the left, which is not useful and will make the truncation of the context fail. So we removed that left whitespace. We tokenized our examples with truncation and padding, but kept the overflows using a stride. This resulted in one example possibly giving several features when a context is long, each of those features having a context that overlaps the context of the previous feature. Since one example might give us several features if it has a long context, we need a map from a feature to its corresponding example. So, we created sample_mapping for serving that purpose. Then we created offset_mappings which will give us a map from token to character position in the original context. It helped us compute the start_positions and end_positions of answers. We labelled the questions with impossible answers as CLS to simplify the process. Also the sequence ids are retrieved for examples to understand the question and the context. If no answers are there for a given question, cls_index has been set, else start and end character index are set. With all these text preprocessing steps, start_positions and end_positions are set and tokenized examples are returned as a result of our preprocessing step.

With the tokenized input dataset, the training is initiated with essential arguments like evaluation_strategy, learning_rate, reporting to weights & biases etc. Once the training is completed, we save the model to huggingface hub for future inference from anywhere. As a final step, we built a haystack index of InMemoryDocumentStore type with FARMReader and TfidfRetriever. To validate the performance of our custom model, we implemented two ExtractiveQAPipelines backed by two readers - our fine tuned model and roberta-base-squad2 model from Deepset. The haystack index is written with the wiki articles about Game of Thrones and the index is queried against the relevant questions to test the performance.

Experiments and Results

To experiment with the MLOPS perspective of our implementation, we built a QA pipeline from scratch in Vertex AI. For building this pipeline, we used TFDistilBertForQuestionAnswering. We set all the project, region, authentication token set programmatically since building QA pipeline is not one of the options in Vertex AI cloud console. We utilized the single distribution strategy since only one gpu device - NVIDIA_TESLA_V100 is used for training. All the training code with preprocessing steps are written as a single file, built into a training image with all the required dependencies and pushed to google container repository for training. The same image is used for training using ai_platform. Once the model is trained, it is deployed in an endpoint for future inference.

With the performance validation against roberta-base-squad2 model from Deepset and our fine tuned model, it is shown that our model captures the context of the question correctly for the given question.

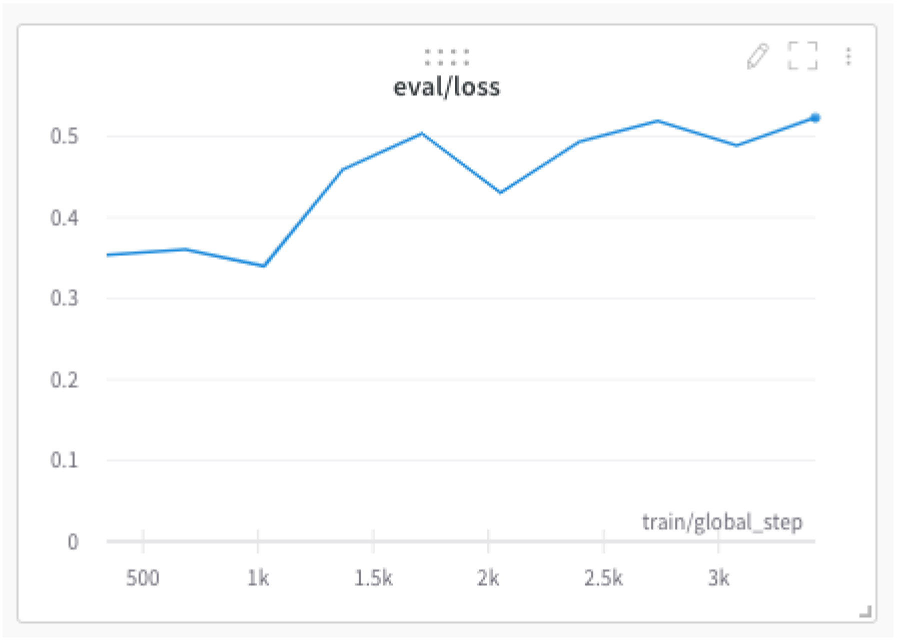


Fig 1. Evaluation Loss recorded during custom model training

As a result of our model training, we achieved a training loss of 0.018600 and evaluation loss of 0.489277. By monitoring the model training, it's clear that our model overfits on the training dataset. The training loss drops gradually but the evaluation loss increases though the model performs better on the training dataset.

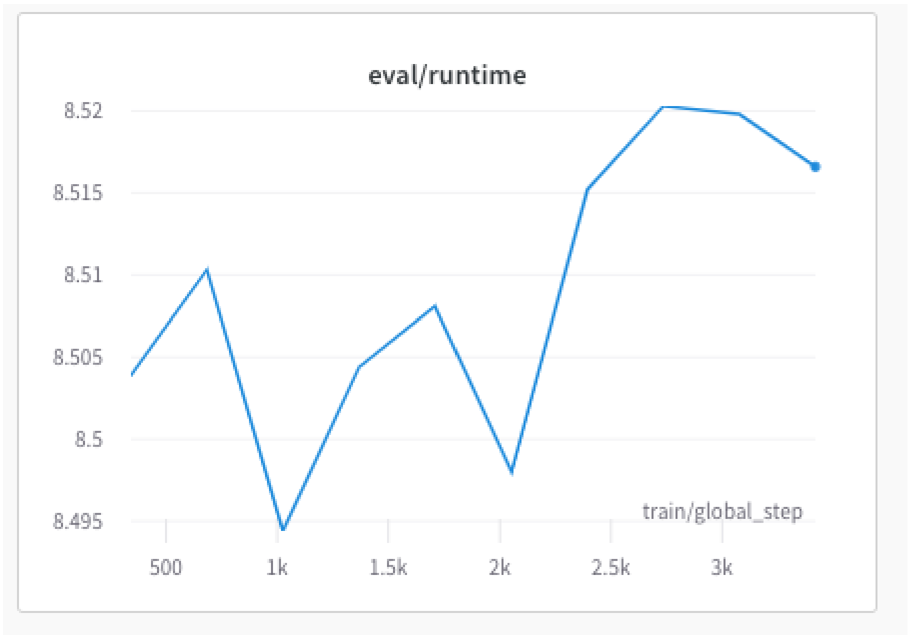
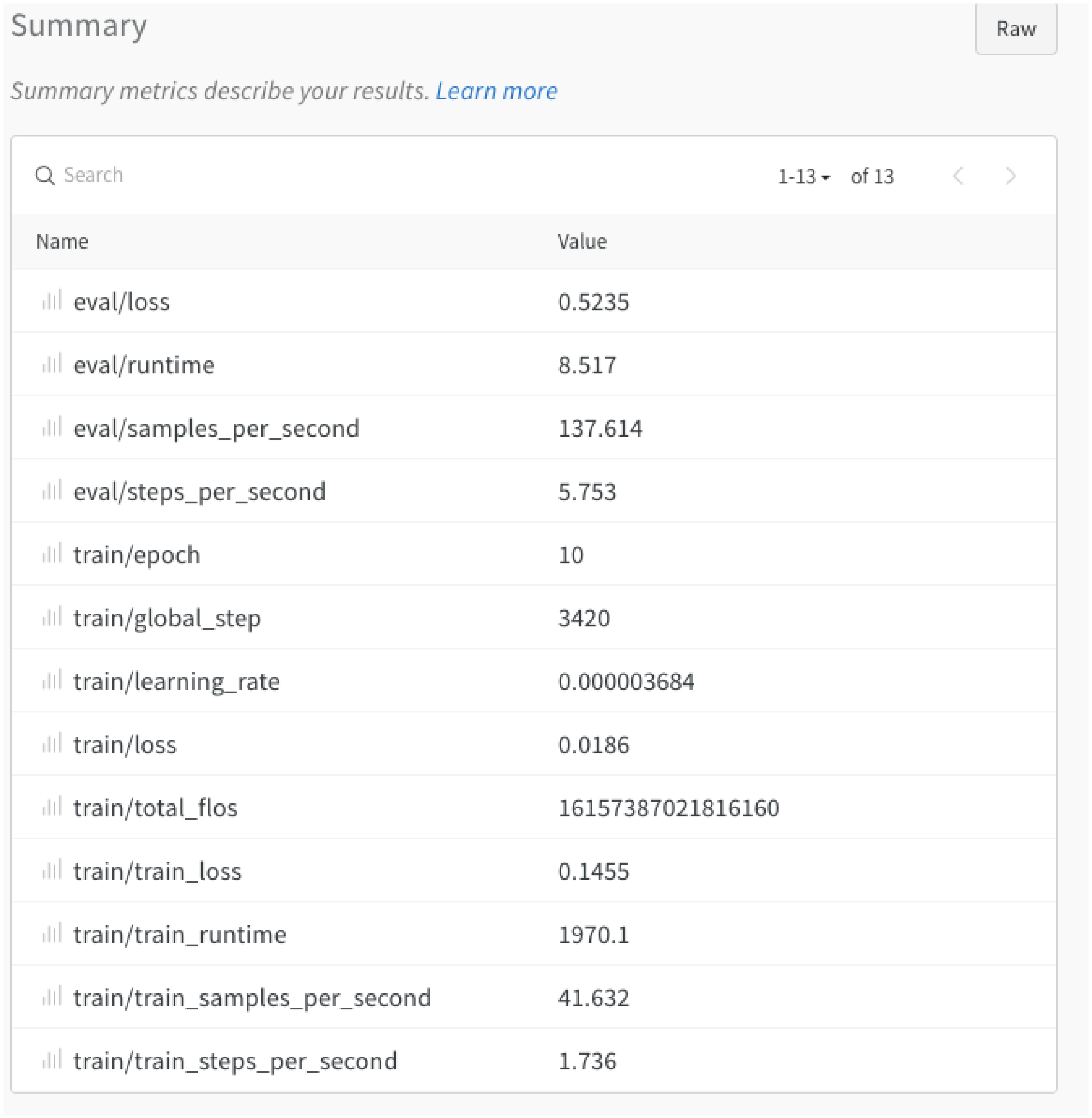


Fig 2. Evaluation runtime recorded during custom model training


Summary

Summary metrics describe your results. Learn more

Name	Value
eval/loss	0.5235
eval/runtime	8.517
eval/samples_per_second	137.614
eval/steps_per_second	5.753
train/epoch	10
train/global_step	3420
train/learning_rate	0.000003684
train/loss	0.0186
train/total_flos	16157387021816160
train/train_loss	0.1455
train/train_runtime	1970.1
train/train_samples_per_second	41.632
train/train_steps_per_second	1.736

Fig 3. Summary of the hyperparameters used during custom model training

Conclusion

The Haystack model is stored in memory, and the saved model has reader-FARMReader and TFidfRetriever for model retrieval. The model used for the reader is the roberta model with SQUAD 2.0 dataset. Then there is a job initiated for training and saving the model in google cloud storage. The other parameters like number of GPUs, epoch, steps to take are set, and training strategy is set for the model. The environment variables are authenticated in the cloud account, and the model is saved. The saved model in vertex AI is utilized for creating an endpoint for future usage. The future direction of work will be to integrate it with a full-stack web application to utilize the functionality of the question-answering feature and improve the model performance by hyperparameter tuning or using other state-of-the-art models in natural language processing.

Code link

https://colab.research.google.com/drive/1_svHGyR3oWqwRP6CVkWERU91pFXN_LZf#scrollTo=K0Br3ZrgkJY6

https://colab.research.google.com/drive/1KpTPzwYu5EoZKxq6cdqQQQeyH0S5tBUi#scrollTo=EZeOWJLvVEbL