Alexa's Utterances and Intent

UC Berkeley MIDS 266

Carla A Prieto Chavez

2024-05-31

Abstract

Voice-activated virtual assistants, like Amazon Alexa, have revolutionized human-computer interaction by responding in real-time to user instructions, known as "utterances." These utterances are mapped to specific "intents," which guide Alexa in executing the correct tasks. This paper explores how Alexa processes and maps utterances to intents, using different NLP techniques.

According to a 2018 paper titled "Efficient Large-Scale Neural Domain Classification with Personalized Attention," Alexa uses LSTM (Long Short-Term Memory) networks and personalized attention techniques to map utterances to intents (Kim et al., 2018). While this paper provides valuable insights, it is uncertain if these methods are still in use today.

AmazonScience, the team within Amazon Alexa responsible for intent prediction, has released the MASSIVE dataset on HuggingFace, providing valuable resources for NLP research (link). By leveraging the MASSIVE dataset, this paper aims to explore how BERT performs on mapping utterances to intent.

Introduction

In this paper, I will explore the use of various BERT (Bidirectional Encoder Representations from Transformers) models to map utterances to intents. This decision stems from my prior experience using BERT for different natural language processing tasks, and a desire to investigate its applicability for multiclass classification, specifically in the context of intent recognition for Alexa.

Having worked as an engineer at Alexa, I am familiar with the intricacies of this problem. My engineering background provided a deep understanding of the challenges in accurately mapping user utterances to the correct intents. However, I wanted to approach this problem from a Machine Learning perspective, leveraging the powerful capabilities of BERT models.

It is important to set realistic expectations for the outcomes of these experiments. The dataset from AmazonScience, known as the MASSIVE dataset, is quite large. Due to constraints in time and computational resources, I am conducting these experiments on Google Colab. Consequently, the results of this study may not match the performance levels achieved in the AmazonScience paper. Nevertheless, this exploration aims to provide insights into the use of BERT for intent recognition and contribute to the ongoing research in this area.

Background

BERT for Multiclass Classification

BERT (Bidirectional Encoder Representations from Transformers) has become a leading model in the field of natural language processing due to its ability to understand the context of words in a sentence by considering the entire sentence bidirectionally. BERT has been successfully applied to a wide range of tasks, including multiclass classification, where it has demonstrated significant improvements in performance compared to previous models [Devlin et al., 2019; Amazon Science, 2023].

For example, a study by Devlin et al. (2019) demonstrated that BERT achieves state-of-the-art results on a variety of sentence classification tasks, such as the Stanford Sentiment Treebank (SST-2) and the Multi-Genre Natural Language Inference (MNLI) benchmarks [Devlin et al., 2019]. This success is attributed to BERT's deep bidirectional representations, which enable it to capture complex dependencies between words and improve classification accuracy.

Alexa's Intent Mapping

Amazon Alexa processes user utterances by mapping them to predefined intents, which represent the actions Alexa should take in response to the user's request. This process involves several steps:

1. **Automatic Speech Recognition (ASR)**: Converts the spoken utterance into text.
2. **Natural Language Understanding (NLU)**: Analyzes the text to determine the user's intent and extract relevant entities.
3. **Intent Mapping**: Matches the analyzed text to a predefined intent using machine learning models [Amazon Science, 2023].

According to the 2018 paper by Kim et al., Alexa initially used LSTM networks combined with personalized attention mechanisms to improve the accuracy of intent recognition. The LSTM networks help in capturing the temporal dependencies in the input sequences, while the personalized attention mechanism allows the model to focus on the most relevant parts of the input for each specific user [Kim et al., 2018]. However, given the rapid advancements in NLP, it is likely that Amazon has continued to innovate and improve upon these methods.

Methodology

Exploratory Data Analysis (EDA)

The first step in our analysis will be conducting exploratory data analysis (EDA) to gain a better understanding of the dataset. This involves the following steps:

1. **Dataset Length**: We will determine the length of the dataset, both for the training and test sets.
2. **Utterance Length**: We will calculate the average length of an utterance in characters and words.
3. **Distribution of Utterances per Intent**: We will examine the distribution of utterances across different intents in both the training and test sets to ensure a roughly equal distribution.

All EDA will be conducted using a Jupyter notebook.

Models

We will approach this problem as a multiclass classification problem. Specifically, we will try three BERT models to evaluate their performance in mapping utterances to intents.

Pretrained BERT Model

For our experiments, we will use a pretrained version of BERT from HuggingFace called `bert-base-cased`. This model is well-suited for a variety of NLP tasks, including multiclass classification.

Tokenization

We will use the equivalent pretrained BERT tokenizer from HuggingFace, also called `bert-base-cased`. The tokenization process will involve the following steps:

1. **Tokenizing the Train and Test Sets**: Both the training and test sets will be tokenized using the `bert-base-cased` tokenizer.
2. **Transforming Labels to Categorical**: The labels in the train and test sets will be transformed into categorical values to facilitate the classification process.

Models

1. BERT using the pooler output.

We will start with a BERT model using the pooler output for multiclass classification, this as the pooler output it's the standard for using BERT. The model is based on the BERT tokenization, using the inputs, attention mask layers and token type layer. It follows by running the pretrained BERT model and using the pooler output, passing that through a Dense hidden layer, followed by a Dropout layer and a final multiclass classification layer using softmax.

The model architecture is as follows:

Layer Type	**Output Shape**	**Param #**	**Connected to**
InputLayer (input_ids_layer)	(None, 200)	0	[]
InputLayer (attention_mask_layer)	(None, 200)	0	[]
InputLayer (token_type_ids_layer)	(None, 200)	0	[]
TFBertModel (tf_bert_model_2)	(None, 200, 768), (None, 768)	108310272	['input_ids_layer[0][0]', 'attention_mask_layer[0][0]', 'token_type_ids_layer[0][0]']
Dense (dense_10)	(None, 201)	154569	['tf_bert_model_2[0][1]']
Dropout (dropout_119)	(None, 201)	0	['dense_10[0][0]']
Dense (dense_11)	(None, 60)	12120	['dropout_119[0][0]']

Table 1: Model Architecture: BERT using the pooler output

2. BERT using the average of the output.

The second model utilizes the average of the BERT output for multiclass classification. This involves taking the average of the last hidden state outputs instead of using the pooler output.

The model is based on BERT tokenization, using input layers, attention mask layers, and token type layers. It then runs the pretrained BERT model, computes the average of the last hidden state outputs, and passes this through a Dense hidden layer, followed by a Dropout layer, and a final multiclass classification layer using softmax.

The model architecture is as follows:

Layer Type	Output Shape	Param #	Connected to
InputLayer (input_ids_layer)	(None, 200)	0	[]
InputLayer (attention_mask_layer)	(None, 200)	0	[]
InputLayer (token_type_ids_layer)	(None, 200)	0	[]
TFBertModel (tf_bert_model_7)	(None, 200, 768), (None, 768)	108310272	['input_ids_layer[0][0]', 'attention_mask_layer[0][0]', 'token_type_ids_layer[0][0]']
TFOpLambda (tf.math.reduce_mean_3)	(None, 768)	0	['tf_bert_model_7[0][0]']
Dense (dense_17)	(None, 201)	154569	['tf.math.reduce_mean_3[0][0]']
Dropout (dropout_309)	(None, 201)	0	['dense_17[0][0]']
Dense (dense_18)	(None, 60)	12120	['dropout_309[0][0]']

Table 2: Model Architecture: BERT averaging the last layer output

3. BERT + CNN

For the third model, we attach a CNN (Convoluted Neural Networks) to process the output from the last layer of the pretrained BERT model. The CNN consists of 5 layers, each with increasing kernel sizes. This output is then passed through dropout and softmax layers for classification.

The model architecture is as follows:

Layer Type	Output Shape	Param #	Connected to
InputLayer (input_ids_layer)	(None, 200)	0	[]
InputLayer (attention_mask_layer)	(None, 200)	0	[]
InputLayer (token_type_ids_layer)	(None, 200)	0	[]
TFBertModel (tf_bert_model_6)	(None, 200, 768), (None, 768)	108310272	['input_ids_layer[0][0]', 'attention_mask_layer[0][0]', 'token_type_ids_layer[0][0]']
Conv1D (conv1d_10)	(None, 199, 131)	201347	['tf_bert_model_6[0][0]']
Conv1D (conv1d_11)	(None, 198, 127)	292735	['tf_bert_model_6[0][0]']
Conv1D (conv1d_12)	(None, 197, 51)	156723	['tf_bert_model_6[0][0]']
Conv1D (conv1d_13)	(None, 196, 23)	88343	['tf_bert_model_6[0][0]']
Conv1D (conv1d_14)	(None, 194, 17)	91409	['tf_bert_model_6[0][0]']
GlobalMaxPooling1D (global_max_pooling1d_10)	(None, 131)	0	['conv1d_10[0][0]']
GlobalMaxPooling1D (global_max_pooling1d_11)	(None, 127)	0	['conv1d_11[0][0]']
GlobalMaxPooling1D (global_max_pooling1d_12)	(None, 51)	0	['conv1d_12[0][0]']
GlobalMaxPooling1D (global_max_pooling1d_13)	(None, 23)	0	['conv1d_13[0][0]']
GlobalMaxPooling1D (global_max_pooling1d_14)	(None, 17)	0	['conv1d_14[0][0]']
Concatenate (concatenate_2)	(None, 349)	0	['global_max_pooling1d_10[0][0]', 'global_max_pooling1d_11[0][0]', 'global_max_pooling1d_12[0][0]', 'global_max_pooling1d_13[0][0]', 'global_max_pooling1d_14[0][0]']
Dense (hidden_layer)	(None, 201)	70350	['concatenate_2[0][0]']
Dropout (dropout_271)	(None, 201)	0	['hidden_layer[0][0]']
Dense (dense_16)	(None, 60)	12120	['dropout_271[0][0]']

Table 3: Model Architecture: BERT and CNN

Results and Discussion

EDA Results

In this subsection, we present the results from the exploratory data analysis (EDA) conducted on the MASSIVE dataset. This includes insights into the dataset length, utterance length, and distribution of utterances per intent for both the training and test sets.

Dataset Length

We examined the length of the dataset to understand its scale. The dataset is divided into training and test sets. Here are the details:

- **Training Set Length**: 11514
- **Test Set Length**: 2974

The test dataset represents 25.83% of the train dataset.

Utterance Length

To gain insights into the structure of the utterances, we calculated the average length of an utterance in both characters and words. The average lengths are as follows:

- **Average Utterance Length in Characters**: 35.039

- **Average Utterance Length in Words**: 6.925

This is important as our pretrained BERT model can only handle a sequence of max 512 tokens. While the words are not strictly equal to a token it gives us a good estimate that the pretrained model will work. This also used as a guide to establish the variable "MAX_SEQUENCE_LENGTH" which is used in the model setup. I left it to be 200 as it covers all utterances.

Distribution of Utterances per Intent

We analyzed the distribution of utterances across different intents to ensure a roughly equal distribution in both the training and test sets. The distribution is visualized in the following charts:

- **Training Set Distribution**:

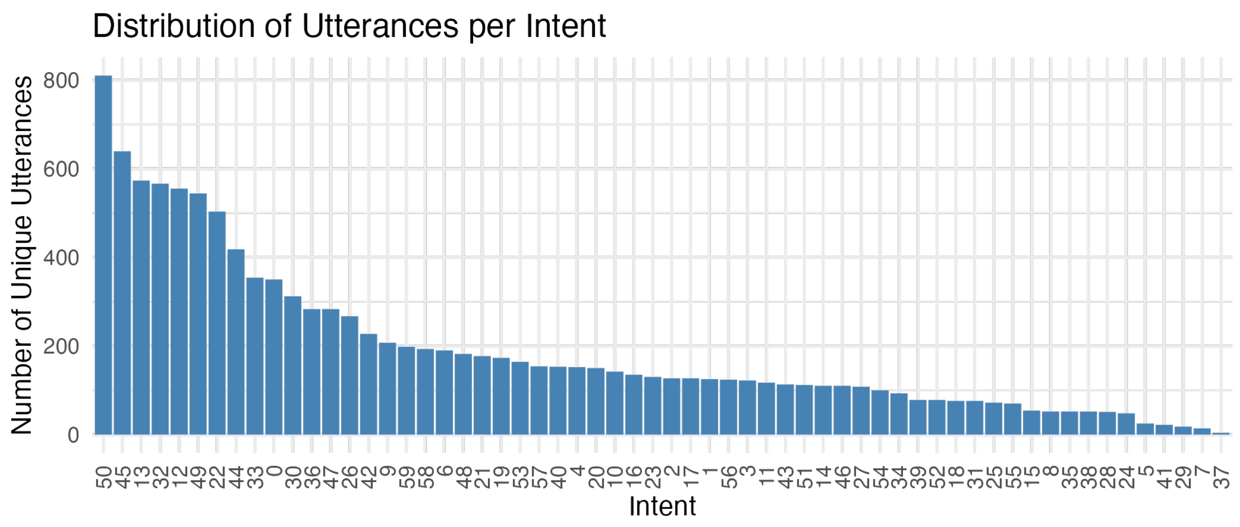
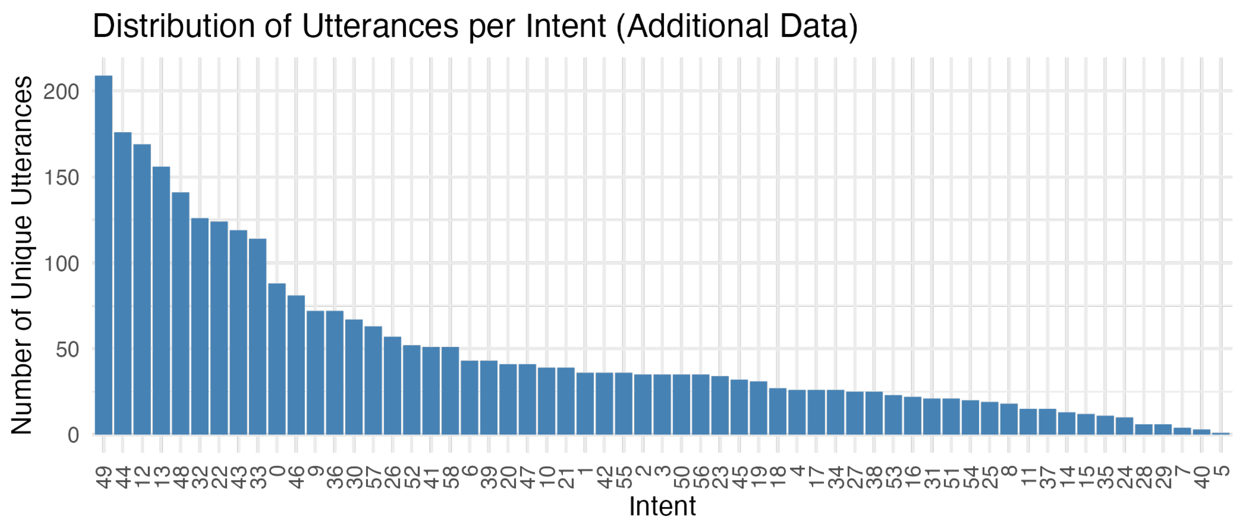


Figure 1: Distribution of Utterances per Intent

- **Test Set Distribution**:



With this EDA we conclude that we have enough data to train and validate our models. We also corroborate that the train and testing datasets are evenly distributed.

Model Results

Model setup

In the face of resource constraints (lack of computing power). Each model was run with the following parameters: - Epochs = 1 - Batch Size = 10 - Learning rate = 0.00005

All the models were run using 4TPUs using Google Colab. See references for link to the code.

In this subsection, we present the results of the three different BERT models I tested for intent recognition. The table below summarizes the runtime, loss, accuracy, validation loss, and validation accuracy for each model.

Table 4: Model Results

Model	Runtime	Loss	Accuracy	Validation.Loss	Validation.Accuracy
BERT w pooler output	25m	1.6619	0.5906	3.6678	0.4469
BERT w/ averaged output	24.76m	1.0586	0.7440	3.4003	0.4839
BERT and CNN	47.35m	1.2624	0.6950	3.2038	0.4785

The first thing to note about this results is that the validation accuracy is lacking. Since this point I defined success by increasing validation accuracy above 0.50.

I noticed a few things in order to prepare for a second round of models. First the loss is still improvable, so for a second round I decided to push the learning rate to 0.0001. I also decided to try more training epochs.

Table 5: Model Results: 2nd Round

Model	Epochs	Runtime	Loss	Accuracy	Validation.Loss	Validation.Accuracy
BERT w pooler output	2	19.41m	0.5706	0.8688	3.4729	0.4778
BERT w/ averaged output	3	73.44m	0.3624	0.9114	4.7088	0.4667
BERT and CNN	2	4.25h	0.8156	0.8016	0.7981	0.7946

From this results we can gather a few learnings:

- For the BERT w pooler output and BERT w/ average output the extra epoch seems to be overfitting the model, since the accuracy significantly increased but the validation accuracy barely did.
- The best model in this experiment proved to be BERT + CNN. The validation accuracy significantly improved with a second epoch and it's matching the training accuracy better.

Conclusion

The best model from these experiments was the BERT + CNN model. Despite taking roughly 4 hours to complete, it achieved a validation accuracy above 0.50, making this experiment a success. This model's performance demonstrates the potential benefits of combining BERT with CNN for complex multiclass classification tasks.

Throughout this experiment, we learned that increasing the number of epochs does not necessarily lead to better results. It's crucial to evaluate models using test or validation data to get a reliable measure of their performance. This insight underscores the importance of using validation metrics to guide model tuning and selection.

If the classification task had been simpler, with fewer classes and a more even distribution among the classes, the BERT model with averaged output might have been a suitable option. In fact, BERT with averaged output is likely a good starting point for classification tasks using BERT, given its balance of performance and computational efficiency.

It's important to note that we don't have access to the actual model used by Alexa, and our experiment was conducted with more limited resources. Therefore, it would be unfair to compare our results directly with those of the actual Alexa model. Nonetheless, this study provides valuable insights into the use of BERT for intent recognition and lays the groundwork for future research in this area.

References

1. Kim, Y.-B., Kim, D., Han, K., and Lee, Y. (2018). Efficient Large-Scale Neural Domain Classification with Personalized Attention. Retrieved from https://pages.cs.wisc.edu/~ybkim/paper/acl2018.pdf
2. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Retrieved from https://arxiv.org/abs/1810.04805
3. Amazon Science (2023). AmazonScience/massive Dataset. Retrieved from https://huggingface.co/datasets/AmazonScience/massive
4. Github repo with calculations and models: https://github.com/Carla08/alexa_utt_intention_bert