CA 6 – Natural Language Processing

Sheyda Eshaghi – Parnian Fazel

CA#6 Part 1

Question Answering

Abstract

One of the fundamental objectives of artificial intelligence is the development of question-answering systems (QA). Deep Learning (DL) approaches have led to significant improvements in QA systems. Despite having excellent QA performance, DL needs a sizable amount of annotated data for training. For the QA work, many annotated datasets have been created; the majority of them are only in English and models doesn't work on Persian Language. So we propose four models on three different datasets, PQuad, PersianQA, and ParsSquad, in order to answer the requirement for a high-quality QA dataset in the Persian language. Finally, we combine datasets from PersianQA and PQuad to create our final model.
For Bert and Albert separately, we train these datasets.
We develop our models using these steps: (using the pars Bert version)

1) Preprocessing datasets
2) Preparing dataset features
3) Tokenizing datasets
4) Training models with Bert and Albert configurations
5) Prediction and Evaluation.

Introduction

Nowadays, many efforts have been performed to design systems to factoid answer the user's queries. In Natural Language Processing (NLP), Question answering (QA) systems can be developed to general and private domains. The QA systems are used in several systems, such as Decision Support systems, Business Intelligence, Interactive systems with a robot-based interface to allow a conversation to imitate human dialogue, community QA systems, and QA systems in biomedical medicine field. In the Persian language, most of these systems have focused on questions with factoid answers, which can answer these questions with relatively little linguistic knowledge.

Our Goal is to measure the performance of Bert and Albert based models on available Persian datasets for question answering task and introduce best model.

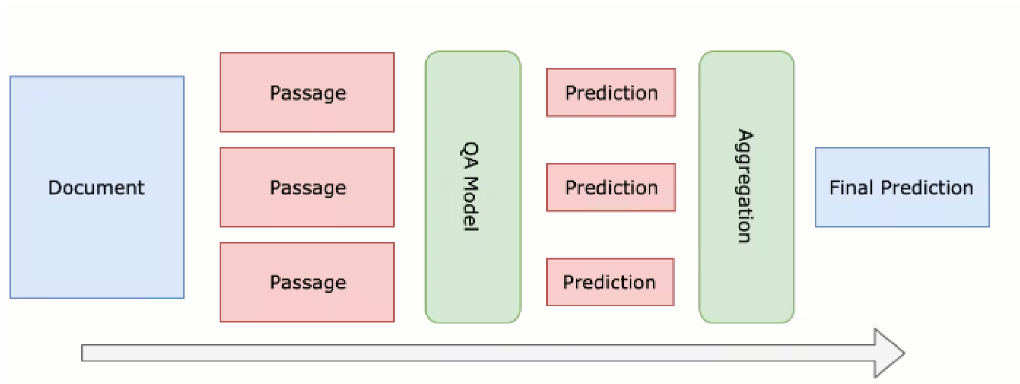

Figure 1 - Workflow of a question answering model

Methodology

The input sequence that is fed into the model will contain tokens from the question, tokens from the passage, model specific special tokens and padding tokens. The length of these four combined must be less than the model's max sequence length and this must be accounted for when we divide up our document into passages.

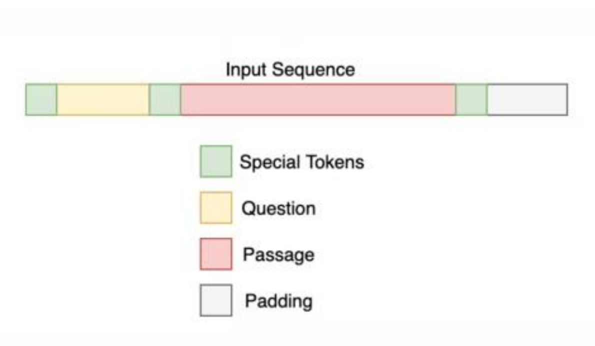

Figure 2 - Input sentence preprocessing

The labels for *positive-answers* are represented in the model by start and end token indices (e.g. [20, 54]). A *no-answer* is represented by a start and end at index 0 of the passage (i.e. [0, 0]). This usually means that start and end will land on the first special token (e.g. [CLS] in BERT).

First we define a preprocessing function for creating data frames from Json files. (All of datasets given in projects are json files.)

In this function based on json parameters in json files, train and test and validation datasets are built. We can access to every part of json file in this function.

```python
def json_to_dataframe(file):
    f = open ( file , "r")
    data = json.loads(f.read())              #Loading the json file.
    iid = []
    tit = []                                 #Creating empty lists to store values.
    con = []
    Que = []
    ans = []

    for i in range(len(data['data'])):       #Root tag of the json file contains 'title' tag & 'paragraphs' list.
        for p in range(len(data['data'][i]['paragraphs'])):  # 'paragraphs' list contains 'context' tag & 'qas' list.
            context = data['data'][i]['paragraphs'][p]['context']
            for q in range(len(data['data'][i]['paragraphs'][p]['qas'])):  # 'qas' list contains 'question', 'Id' tag & 'answer
s' list.
                question = data['data'][i]['paragraphs'][p]['qas'][q]['question']
                Id = data['data'][i]['paragraphs'][p]['qas'][q]['id']
                answer_texts = []
                answer_starts = []
                for a in range(len(data['data'][i]['paragraphs'][p]['qas'][q]['answers'])): # 'answers' list contains 'ans_star
t', 'text' tags.
                    answer_texts.append(data['data'][i]['paragraphs'][p]['qas'][q]['answers'][a]['text'])
                    answer_starts.append(data['data'][i]['paragraphs'][p]['qas'][q]['answers'][a]['answer_start'])
                    if len(answer_texts) == 0:
                      answer_texts.append('')
                      answer_starts.append(0)
                    con.append(context)
                    Que.append(question)                    # Appending values to lists
                    iid.append(Id)
                    ans.append({'text': answer_texts, 'answer_start': answer_starts })

    new_df = pd.DataFrame(columns=['id','context','question','answers']) # Creating empty DataFrame.
    new_df.id = iid
    new_df.context = con
    new_df.question = Que
    new_df.answers = ans

    return new_df
```

Figure 3 Python code of Json to Dataframe function

After defining this function we call it for every train, test and validation samples.

In the following we observe samples from three given datasets.

	id	context	question	answers
0	1	شرکت فولاد مبارکهٔ اصفهان، بزرگ‌ترین واحد صنعتی...	شرکت فولاد مبارکه در کجا واقع شده است	{'text': ['در شرق شهر مبارکه'], 'answer_start'...
1	2	شرکت فولاد مبارکهٔ اصفهان، بزرگ‌ترین واحد صنعتی...	فولاد مبارکه چند بار برنده جایزه شرکت دانشی را...	{'text': ['۶'], 'answer_start': [263]}
2	3	شرکت فولاد مبارکهٔ اصفهان، بزرگ‌ترین واحد صنعتی...	شرکت فولاد مبارکه در سال ۱۳۹۱ چه جایزه ای برد؟	{'text': ['تندیس زرین جایزهٔ ملی تعالی سازمانی'...
3	4	شرکت فولاد مبارکهٔ اصفهان، بزرگ‌ترین واحد صنعتی...	بزرگ ترین مجموعه تولید فولاد ایران چیست؟	{'text': ['شرکت فولاد مبارکهٔ'], 'answer_start'...
4	5	شرکت فولاد مبارکهٔ اصفهان، بزرگ‌ترین واحد صنعتی...	فولاد مبارکه در چه سالی احداث شد؟	{'text': ['۱۳۷۱'], 'answer_start': [504]}

Figure 4 Train dataset – PersianQA

	id	context	question	answers
0	874587	عبّاس میرزا (۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹...	عبّاس میرزا در چه تاریخی به دنیا آمد؟	{'text': ['۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹ م...
1	874588	عبّاس میرزا (۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹...	عبّاس میرزا در چه تاریخی درگذشت؟	{'text': ['۱۰ جمادی‌الثانی ۱۲۴۹ ه.ق/۲۵ اکتبر ...
2	874589	عبّاس میرزا (۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹...	عبّاس میرزا که بود؟	{'text': ['نایب‌السّل... نامدار به عبّاس میرزا...
3	874590	عبّاس میرزا (۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹...	عبّاس میرزا در چه سال‌هایی ولایتعهدی ایران و ن...	{'text': ['سال‌های ۱۷۹۷ تا ۱۸۳۳ م'], 'answer_s...
4	874591	عبّاس میرزا (۴ ذی‌الحجهٔ ۱۲۰۳ ه.ق/۲۶ اوت ۱۷۸۹...	عبّاس میرزا فرزند کدام پادشاه ایران است؟	{'text': ['فتحعلی‌شاه'], 'answer_start': [161]}

Figure 5 Train dataset – PSquad

	id	context	question	answers
0	56be85543aeaaa14008c9063	بیانسه جیزل نوولز-کارتر (/ biːˈjɒnseɪ / bee-YO...	از چه زمانی بیانسه شروع به محبوبیت کرد؟	{'text': ['در اواخر دهه 1990'], 'answer_start'...
1	56be85543aeaaa14008c9066	بیانسه جیزل نوولز-کارتر (/ biːˈjɒnseɪ / bee-YO...	چه موقع بیانسه Destiny's Child را ترک کرد و یک...	{'text': ['2003'], 'answer_start': [476]}
2	56bf6b0f3aeaaa14008c9602	بیانسه جیزل نوولز-کارتر (/ biːˈjɒnseɪ / bee-YO...	در چه دهه ای بیانسه مشهور شد؟	{'text': ['اواخر دهه 1990'], 'answer_start': [...
3	56bf6b0f3aeaaa14008c9605	بیانسه جیزل نوولز-کارتر (/ biːˈjɒnseɪ / bee-YO...	چه کسی گروه Destiny's Child را مدیریت می کرد؟	{'text': ['ماتیو نولز'], 'answer_start': [323]}
4	56d43c5f2ccc5a1400d830a9	بیانسه جیزل نوولز-کارتر (/ biːˈjɒnseɪ / bee-YO...	چه موقع بیانسه به شهرت رسید؟	{'text': ['اواخر دهه 1990'], 'answer_start': [...

Figure 6 Train dataset - Parssquad

- For ParsSquad and PersianQA we don't have any validation files, so we use test files as validation datasets and train the model based on train dataset and validation dataset.
- For Mixed model we append two dataframes for each train, test and validation datasets.
- One specific thing for the preprocessing in question answering is how to deal with very long documents. We usually truncate them in other tasks, when they are longer than the model maximum sentence length, but here, removing part of the the context might result in losing the answer we are looking for. To deal with this, we will allow one (long) example in our dataset to give several input features, each of length shorter than the maximum length of the model (or the one we set as a hyper-parameter). If we just truncate, we will lose information.

```
len(tokenizer(example["question"], example["context"])["input_ids"])
```

```
322
```

Figure 7- Max length of PQuad

```
len(tokenizer(example["question"], example["context"])["input_ids"])
```

```
442
```

Figure 8- Max length of ParsSquad

```
len(tokenizer(example["question"], example["context"])["input_ids"])
```

```
320
```

Figure 9 - Max length of PersianQA

After we finish this step and making dataframes from json samples we change the type of dataset to json. We do this because of indention in json files and after these processing we have same indent json parameters like below.


```
{'answers': {'answer_start': [263], 'text': ['۶']},
 'context': 'بزرگ\u200cترین مجتمع تولید فولاد در کشور ایران است، که در شرق شهر مبارکه، اصفهان، بزرگ\u200cترین واحد صنعتی خصوصی در ایران و بزرگ\u200cشرکت فولاد مبارکهٔ
دستی است. فولاد مبارکه در ۱۱ دوره جایزهٔ ملی تعالی سازمانی و ۶ دوره جایزهٔ شرکت دانشی\u200cاکنون محرک بسیاری از صنایع بالادستی و پایین\u200cقرار دارد. فولاد مبارکه هم
بار به عنوان تنها شرکت ایرانی با کسب امتیاز ۶۵۴ تندیس زرین جایزهٔ ملی تعالی\u200cاست و همچنین این شرکت در سال ۱۳۹۱ برای نخستین\u200cدر کشور رتبهٔ نخست را بدست آورده
ترین واحدهای صنعتی و بزرگترین مجتمع تولید فولاد در ایران است. ای\u200cی سازمانی را از آن خود کند. شرکت فولاد مبارکهٔ اصفهان در ۲۳ دی ماه ۱۳۷۱ احداث شد و اکنون بزرگ
است. مصرف آب این کارخانه در کمترین میزان خ\u200cن شرکت در زمینی به مساحت ۳۵ کیلومتر مربع در نزدیکی شهر مبارکه و در ۷۵ کیلومتری جنوب غربی شهر اصفهان واقع شده
شود\u200cرود شناخته می\u200cآبی زاینده\u200cرود برابر سالانه ۲۳ میلیون متر مکعب در سال است و خود یکی از عوامل کم\u200cوده، ۱٫۵٪ از دبی زاینده.',
 'id': 2,
 'question': 'فولاد مبارکه چند بار برنده جایزه شرکت دانشی را کسب کرده است؟'}
```


Figure 10 - A sample after preprocess json file

In every datasets after processing json file we have these features with different row numbers:

```
Dataset({
    features: ['id', 'context', 'question', 'answers'],
    num_rows: 6306
})
```

We want to figure out every answer end position. Because we have start position and the answer text but we don't have end position and this is important to know this for embedding and feature preparing.

A few preprocessing steps particular to question answering that we assume are:

1. Some examples in a dataset may have a very long context that exceeds the maximum input length of the model. Truncate only the context by setting truncation="only_second".
2. Next, map the start and end positions of the answer to the original context.
3. With the mapping in hand, we can find the start and end tokens of the answer. Use the sequence_ids method to find which part of the offset corresponds to the question and which corresponds to the context.
After that we use HuggingFace Datasets map function to apply the preprocessing function over the entire dataset. You can speed up the map function by setting batched=True to process multiple elements of the dataset at once and we remove the columns we don't need.

```
tokenized_train = train_ds.map(prepare_train_features, batched=True, remove_columns=train_ds.column_names)
```

We use DefaultDataCollator to create a batch of examples. Unlike other data collators in HuggingFace Transformers, the DefaultDataCollator does not apply additional preprocessing such as padding.

```
from transformers import DefaultDataCollator

data_collator = DefaultDataCollator()
```

Before we can feed those texts to our model, we need to preprocess them. This is done by a HuggingFace Transformers Tokenizer which will (as the name indicates) tokenize the inputs (including converting the tokens to their corresponding IDs in the pretrained vocabulary) and put it in a format the model expects, as well as generate the other inputs that model requires.

To do all of this, we instantiate our tokenizer with the AutoTokenizer.from_pretrained method, which will ensure:

- we get a tokenizer that corresponds to the model architecture we want to use,
- we download the vocabulary used when pretraining this specific checkpoint.

Now that our data is ready for training, we can download the pretrained model and fine-tune it. Since our task is question answering, we use the AutoModelForQuestionAnswering class.

The question and answer texts are separated by a [sep] token, and "##" means that the rest of the token should be attached to the previous one, without a space (for decoding or reversal of the tokenization). The usage of "##" ensures that the token with this symbol is directly related to the token just before it.

ParsBert configuration for all of models shown as below.

```
config = AutoConfig.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
tokenizer = AutoTokenizer.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
```

```
model = AutoModelForQuestionAnswering.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
```

Albert configuration for all of models shown as below.

```
config = AutoConfig.from_pretrained("HooshvareLab/albert-fa-zwnj-base-v2")
tokenizer = AutoTokenizer.from_pretrained("HooshvareLab/albert-fa-zwnj-base-v2")
model = AutoModelForQuestionAnswering.from_pretrained("HooshvareLab/albert-fa-zwnj-base-v2")
```

To instantiate a Trainer, we will need to define three more things. The most important is the TrainingArguments, which is a class that contains all the attributes to customize the training. It requires one folder name, which will be used to save the checkpoints of the model, and all other arguments are optional. We will evaluate our model and compute metrics in the next section. Then we just need to pass all of this along with our datasets to the Trainer.

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    "bert-finetuned-squad",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    num_train_epochs=3,
    weight_decay=0.01,

)


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_val,
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

In following we describe our models arguments:

Num Epochs = 3
Instantaneous batch size per device = 8
Total train batch size (w. parallel, distributed & accumulation) = 8

An input sequence can be passed directly into the question answering model as is standardly done in Transfer Learning paradigm. For every token that enters the model, a contextualized word vector is returned.

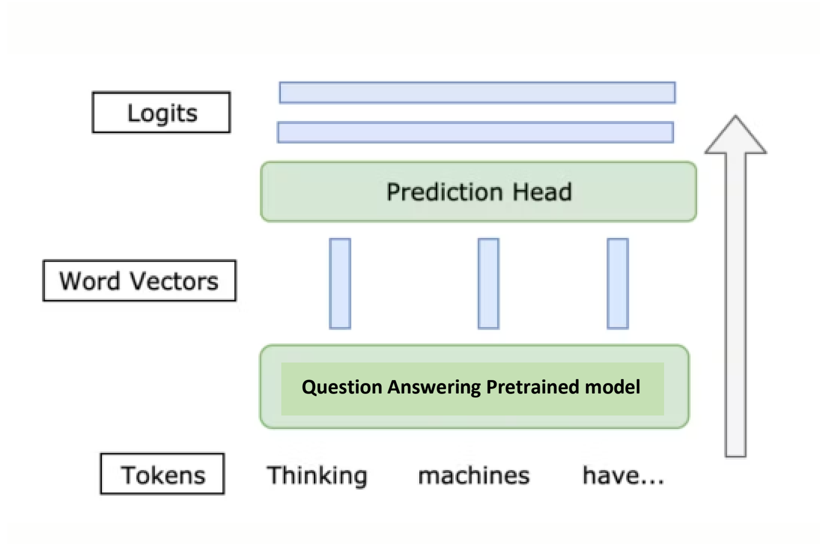

Figure 11 - Model paradigm

Result and evaluation

Evaluating our model will require a bit more work, as we will need to map the predictions of our model back to parts of the context. The model itself predicts logits for the start and en position of our answers: if we take a batch from our validation datalaoder, here is the output our model gives us.

To classify our answers, we will use the score obtained by adding the start and end logits. We won't try to order all the possible answers and limit ourselves to with a hyper-parameter we call n_best_size. We'll pick the best indices in the start and end logits and gather all the answers this predicts. After checking if each one is valid, we will sort them by their score and keep the best one. Here is how we would do this on the first feature in the batch.

And then we can sort the valid_answers according to their score and only keep the best one. The only point left is how to check a given span is inside the context (and not the question) and how to get back the text inside. To do this, we need to add two things to our validation features:

the ID of the example that generated the feature (since each example can generate several features, as seen before);

the offset mapping that will give us a map from token indices to character positions in the context.

That's why we will re-process the validation set with the following function, slightly different from prepare_train_features.

```python
def compute_metrics(start_logits, end_logits, features, examples):
    predicted_answers = []
    for i, example in enumerate(examples):
        example_id = example["id"]
        context = example["context"]
        answers = []
        start_logit = start_logits[i]
        end_logit = end_logits[i]
        offsets = features[i]["offset_mapping"]
        start_indexes = np.argsort(start_logit)[-1 : -n_best - 1 : -1].tolist()
        end_indexes = np.argsort(end_logit)[-1 : -n_best - 1 : -1].tolist()
        start_indexes = np.argsort(start_logit)[-1 : -n_best - 1 : -1].tolist()
        end_indexes = np.argsort(end_logit)[-1 : -n_best - 1 : -1].tolist()
        for start_index in start_indexes:
            for end_index in end_indexes:
                if offsets[start_index] is None or offsets[end_index] is None:
                    continue
                if (
                    end_index < start_index
                    or end_index - start_index + 1 > max_answer_length
                ):
                    continue
                answer = {
                    "text": context[offsets[start_index][0] : offsets[end_index][1]],
                    "logit_score": start_logit[start_index] + end_logit[end_index],
                }
                answers.append(answer)

        if len(answers) > 0:
            best_answer = max(answers, key=lambda x: x["logit_score"])
            predicted_answers.append(
                {"id": example_id, "prediction_text": best_answer["text"]}
            )
        else:
            predicted_answers.append({"id": example_id, "prediction_text": ""})
    theoretical_answers = [{"id": ex["id"], "answers": ex["answers"]} for ex in examples]
    return metric.compute(predictions=predicted_answers, references=theoretical_answers)
```

Figure 12- Model metrics computation

Now we can grab the predictions for all features by using the Trainer.predict method:

```
predictions, a, b = trainer.predict(dataset_test_preprocessed)
```

```
The following columns in the test set don't have a corresponding argument in `BertForQuestionAnswering.forward` and have been i
gnored: id, offset_mapping. If id, offset_mapping are not expected by `BertForQuestionAnswering.forward`,  you can safely ignor
e this message.
***** Running Prediction *****
  Num examples = 8742
  Batch size = 8
```

[1093/1093 11:43]

At the end with prediction and start logits and end logits of answer we can compute f1 score and exact match of our models with following code:

```python
start_logits, end_logits = predictions
compute_metrics(start_logits, end_logits, dataset_test_preprocessed, test_ds)
```

Figure 13- Compute metrics of model

Summary of models

F1 Score:

Dataset	Bert	Albert

PSQuad	84.67	73.72
PersianQA	67.28	28.12
ParsSquad	76.29	70.43
PersianQA + PSQuad	76.52	67.04

Exact Match:

Dataset	Bert	Albert
PSQuad	63.93	54.48
PersianQA	41.62	11.05
ParsSquad	67.43	60.62
PersianQA + PSQuad	53.23	45.74

Training and Validation Loss plot and Epoch information for each model:

Bert – PSquad

[2220/2220 12:20, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.865500	1.226043
2	0.979100	1.233559
3	0.397800	1.393670

Albert – PSquad

[2220/2220 11:36, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	2.651000	1.811999
2	1.555000	1.738862
3	0.899900	1.825861

Bert – PersianQA

[2367/2367 20:35, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	3.510500	2.100585
2	1.849300	2.007501
3	1.305600	2.116803

Albert – PersianQA

[2367/2367 19:12, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	5.703300	5.703781
2	5.651000	4.573941
3	4.256100	3.735383

Bert – ParsSquad

[16227/16227 1:44:57, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.395700	1.723292
2	0.870600	1.840423

[962/1093 01:43 < 00:14, 9.27 it/s]

Albert – ParsSquad

[16227/16227 1:45:00, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.655100	1.980053
2	1.262600	1.954766
3	0.958300	2.059170

Bert – PSquad + PersianQA

[4584/4584 42:01, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.818900	1.574881
2	1.168400	1.552404
3	0.725800	1.756013

Albert – PSquad + PersianQA

[4584/4584 39:35, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	2.519300	2.294442
2	1.911000	2.100307
3	1.466500	2.146301

Key Results:

For tasks that require lower memory consumption and faster training speeds, we can use ALBERT. It's a lite version of BERT which uses parameter reduction techniques, and thus redues the number of parameters while running training and inference. This helps in the scalability of the model as well.

The input embeddings in ALBERT consist of an embedding matrix in a relatively low dimension (e.g. 128128), and hidden layer dimensions are higher (768768 as in the BERT case, or more). With reduced matrix size, the projected parameters also reduced, i.e. an 80% reduction can be observed in the parameters. With a major reduction in F1 Score.

As we can see from the above table is the ALBERT model has a smaller parameter size as compared to corresponding BERT models due to the above changes authors made in the architecture. For Example, BERT base has 9x more parameters than the ALBERT base, and BERT Large has 18x more parameters than ALBERT Large. The original BERT (BERT-base) model is made of 12 transformer encoder layers along with a Multi-head Attention.

Cross-layer parameter sharing is the most significant change in BERT architecture that created ALBERT. ALBERT architecture still has 12 transformer encoder blocks stacked on top of each other like the original BERT. Still, it initializes a set of weights for the first encoder that is repeated for the other 11 encoders. This mechanism reduces the number of "unique" parameters, while the original BERT contains a set of unique parameters for every encoder (see Figure 14).

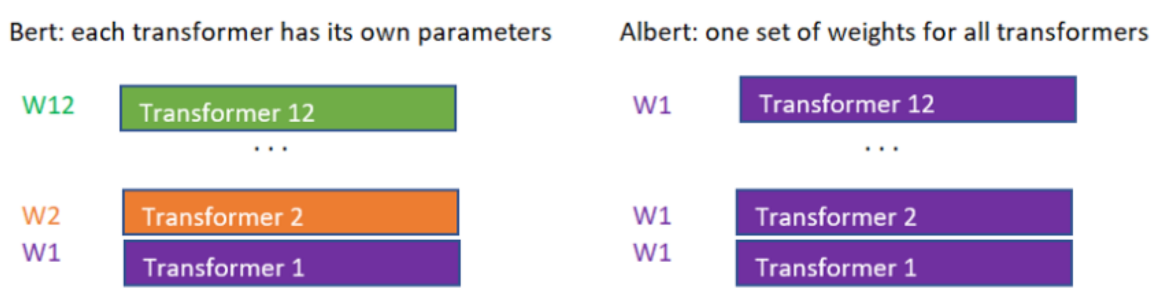

Figure 14

According to all models we assume Pars Bert based model as our base model.

Because of resource and time limits we couldn't get the best answer for our models.

In every model bert works better than albert, because albert is a weak and tiny version of bert using less resources and needs less amount of time.

So Albert needs more epochs for better performance. The validation and training loss in Albert Based models are higher than Bert models.

Bert and Albert works better with smaller datasets.

CA#6 Part 2

Natural Language Understanding

Abstraction

Intent detection and **slot filling** are the main tasks to solve when approaching the problem of **Natural Language Understanding (NLU)** in a **conversational system**. The two tasks are used to obtain a structured representation of the meaning of the utterance, so that it can be processed by a computer. Intent detection deals with identifying the **overall meaning** of the sentence. It is modeled as a **classification** problem, in which we receive an input utterance and we have to classify it as having one intent from a group of known intents. The available intents correspond to the actions that the conversational model can perform, such as adjusting the temperature, controlling the media center or turning the lights on/off in the case of a home assistant. On the other hand, slot filling is modeled as a **sequence labelling problem**, whose purpose is to take the utterance and determine which words **indicate relevant information for the intent**. These slots contain supplementary information about the action and correspond to the parameters of the action.

The task-oriented dialogue system is the basis of virtual assistants like Alexa, Siri, Cortana, and Portal has been increasingly used in modern society; users interact with them across different domains to complete diverse tasks and achieve their specific goals. Key component of these task-oriented dialogue systems is Natural Language Understanding (NLU) which aims to derive the intent of users and fill the value for the slots of the utterance.

Dataset

MASSIVE is a parallel dataset of > 1M utterances across **51 languages** with annotations for the Natural Language Understanding tasks of intent prediction and slot annotation. Utterances span **60 intents** and include **55 slot** types. MASSIVE was created by localizing the SLURP dataset, composed of general Intelligent Voice Assistant single-shot interactions.

In this assignment, we are to use Farsi Dataset (fa-IR.jsonl). This Json file contains train, dev and test sets. Now will study the details of this dataset:

`id` : maps to the original ID in the SLURP collection. Mapping back to the SLURP en-US utterance, this utterance served as the basis for this localization.

`locale` : is the language and country code accoring to ISO-639-1 and ISO-3166.

`partition` : is either `train` , `dev` , or `test` , according to the original split in SLURP.

`scenario` : is the general domain, aka "scenario" in SLURP terminology, of an utterance

`intent` : is the specific intent of an utterance within a domain formatted as `{scenario}_{intent}`

`utt` : the raw utterance text without annotations

`annot_utt` : the text from `utt` with slot annotations formatted as `[{label} : {entity}]`

`worker_id` : The obfuscated worker ID from MTurk of the worker completing the localization of the utterance. Worker IDs are specific to a locale and do *not* map across locales.

`slot_method` : for each slot in the utterance, whether that slot was a `translation` (i.e., same expression just in the target language), `localization` (i.e., not the same expression but a different expression was chosen more suitable to the phrase in that locale), or `unchanged` (i.e., the original en-US slot value was copied over without modification).

`judgments` : Each judgment collected for the localized utterance has 6 keys. `worker_id` is the obfuscated worker ID from MTurk of the worker completing the judgment. Worker IDs are specific to a locale and do *not* map across locales, but *are* consistent across the localization tasks and the judgment tasks, e.g., judgment worker ID 32 in the example above may appear as the localization worker ID for the localization of a different de-DE utterance, in which case it would be the same worker.

Figure 15 Input Jsonl file format details

In this assignment the "judgments" and "worker_id" are not considered.

There are 60 different intents and 56 different slots in this dataset.

```python
import ast
intent_dict = ast.literal_eval(open('data.intents').read())
slot_dict = ast.literal_eval(open('data.slots').read())
intents = list(intent_dict.values())
slots = list(slot_dict.values())
print("Number of All Intents:",len(intents))
print("Number of All  Slots:",len(slots))
print("="*30)
print("----> Intents:")
print('\n'.join(intents))
print("="*30)
print("----> Slots:")
print('\n'.join(slots))
```

```
Number of All Intents: 60
Number of All  Slots: 56
```

Figure 16 Number of slots and intents

Now we take a look at intents list:

----> Intents:
recommendation_locations
play_music
iot_cleaning
email_addcontact
datetime_convert
transport_ticket

qa_stock
lists_query
email_query
datetime_query
calendar_remove
iot_wemo_off
recommendation_events
email_sendemail
qa_maths
general_quirky
calendar_query
iot_hue_lighton
audio_volume_other
takeaway_order
transport_query
weather_query
alarm_set
qa_factoid
play_radio
lists_remove
qa_currency
news_query
lists_createoradd
general_greet
social_query
iot_hue_lightchange
iot_hue_lightdim
calendar_set
iot_hue_lightup
recommendation_movies
play_audiobook
alarm_query
audio_volume_up
cooking_recipe
iot_wemo_on
social_post
qa_definition
audio_volume_mute
general_joke
iot_hue_lightoff
music_dislikeness
transport_traffic
takeaway_query
play_podcasts
iot_coffee
audio_volume_down
play_game
transport_taxi
email_querycontact
music_query
cooking_query
music_likeness
music_settings
alarm_remove

Now we take a look at slots list:

----> Slots:
transport_agency
playlist_name
house_place
media_type
time_zone
time
device_type
business_name
music_album
artist_name
podcast_descriptor
personal_info
email_folder
news_topic
order_type
podcast_name
food_type
transport_type
game_type
general_frequency
list_name
app_name
audiobook_name
sport_type
alarm_type
song_name
place_name
game_name
music_genre
person
change_amount
date
ingredient
radio_name
email_address
meal_type
movie_name
definition_word
transport_name
coffee_type
relation
event_name
currency_name
business_type
music_descriptor
weather_descriptor
timeofday
movie_type
transport_descriptor

joke_type
Other
color_type
player_setting
audiobook_author
cooking_type
drink_type

Here we can compare frequencies of **domains**, **intents** and **slots** count in each partision:

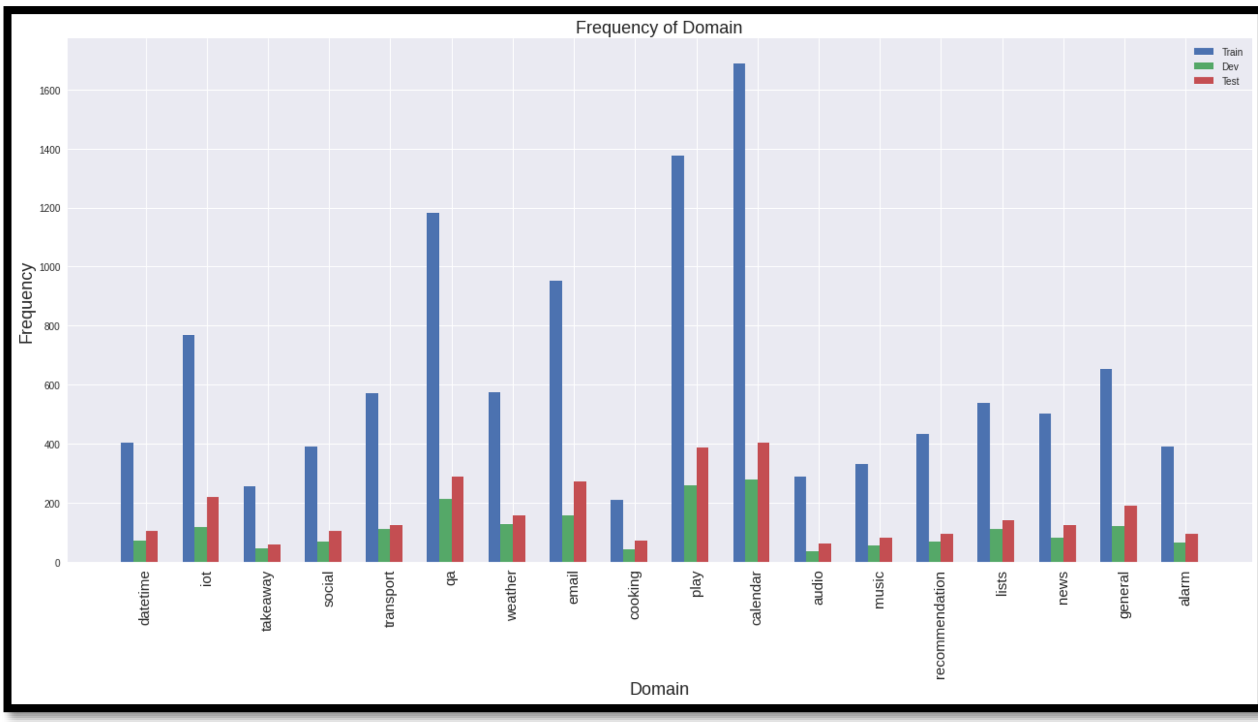

Figure 17 Domain Frequency Bar Plot

As we can see above, the **calendar** domain is the most frequent domain.

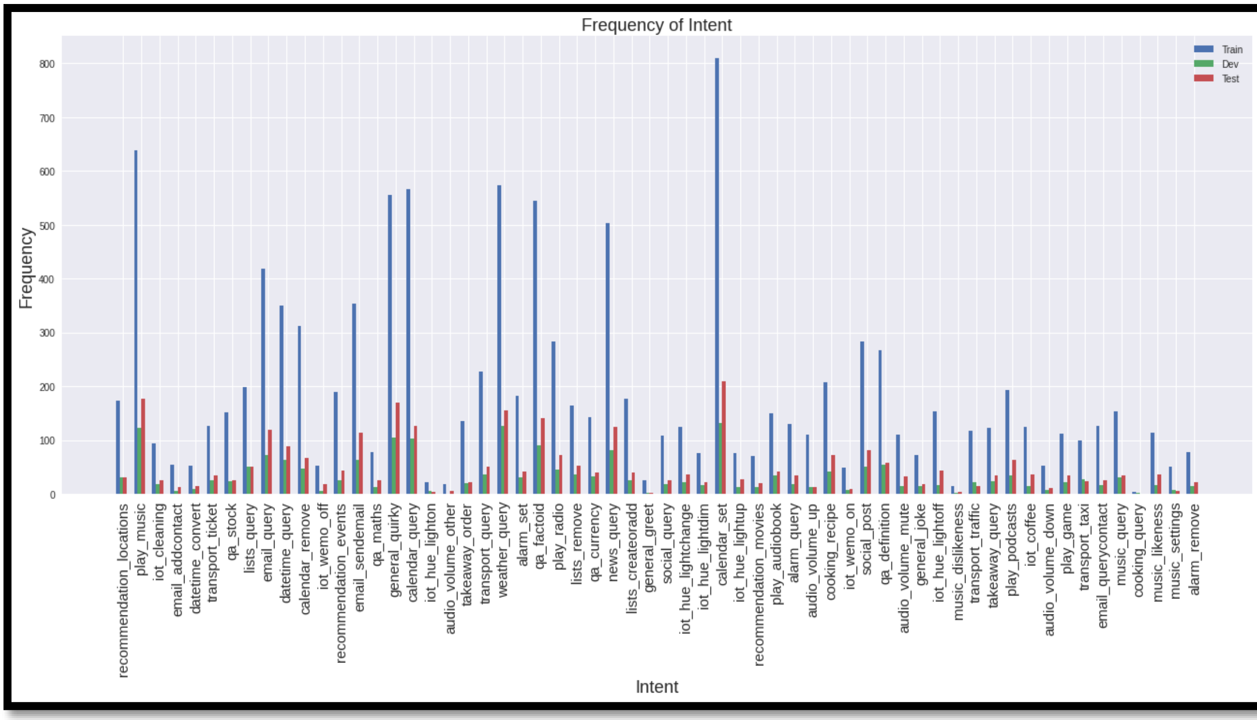

Figure 18 Intent Frequency Bar Plot

As we can see above, the **calendar_set** intent is the most frequent and cooking_query is the least frequent intent.

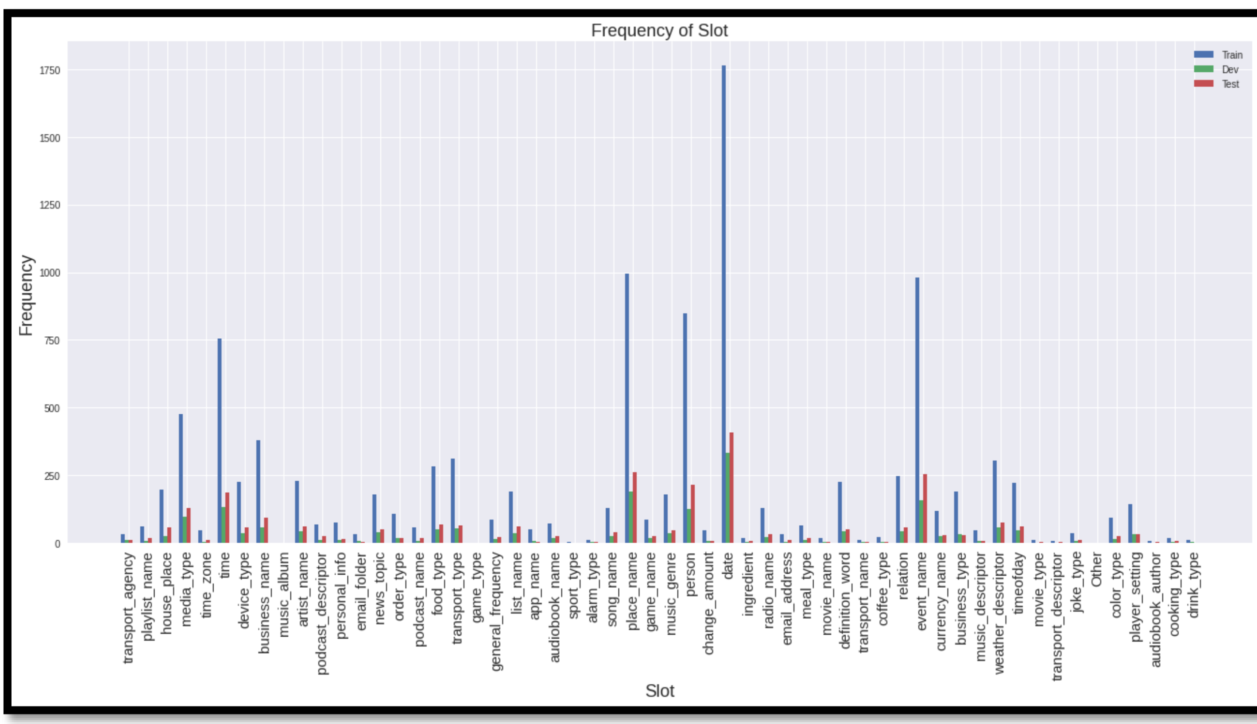

Figure 19 Slot Frequency Bar Plot

As we can see above, the **date** slot is the most frequent slot.

Here are some more details of counts:

```
pd.DataFrame(domain_freq)
```

	train	dev	test
datetime	402	73	103
iot	769	118	220
takeaway	257	44	57
social	391	68	106
transport	571	110	124
qa	1183	214	288
weather	573	126	156
email	953	157	271
cooking	211	43	72
play	1377	260	387
calendar	1688	280	402
audio	290	35	62
music	332	56	81
recommendation	433	69	94
lists	539	112	142
news	503	82	124
general	652	122	189
alarm	390	64	96

Figure 20 Domain frequencies

```
pd.DataFrame(intent_freq)
```

	train	dev	test
recommendation_locations	173	31	31
play_music	639	123	176
iot_cleaning	93	19	26
email_addcontact	54	5	12
datetime_convert	52	9	15
transport_ticket	127	25	35
qa_stock	152	24	26
lists_query	198	50	51
email_query	418	73	119
datetime_query	350	64	88
calendar_remove	312	47	67
iot_wemo_off	52	5	18
recommendation_events	190	26	43
email_sendemail	354	63	114
qa_maths	78	13	25
general_quirky	555	105	169
calendar_query	566	102	126
iot_hue_lighton	22	5	3
audio_volume_other	18	0	6
takeaway_order	135	20	22
transport_query	227	36	51
weather_query	573	126	156
alarm_set	182	31	41

Figure 21 Intent frequencies

```
pd.DataFrame(slot_freq)
```

	train	dev	test
transport_agency	31	9	9
playlist_name	62	7	16
house_place	197	25	57
media_type	474	95	128
time_zone	45	4	11
time	755	132	186
device_type	224	35	57
business_name	379	58	92
music_album	1	0	1
artist_name	228	44	60
podcast_descriptor	66	10	24
personal_info	74	9	14
email_folder	32	6	5
news_topic	179	40	49
order_type	106	19	19
podcast_name	56	8	17
food_type	284	49	69
transport_type	311	54	64
game_type	1	1	0
general_frequency	84	15	20
list_name	190	37	60
app_name	51	8	5
audiobook_name	73	16	23

Figure 22 slot frequencies

Preparing Dataset

I put the fa-IR-jsonl file in "Dataset" folder and passed it to create_hf_dataset.py.

```
[ ] !python massive/scripts/create_hf_dataset.py -d /content/Dataset -o data

Reading in data from /content/Dataset/fa-IR.jsonl
The following intent labels were detected across all partitions:  {'email_addcontact': 0, 'alarm_query': 1, 'iot_wemo_off': 2, 'music_dislikeness': 3, 'lists_query': 4, 'general_joke': 5
The following slot labels were detected across all partitions:  {'date': 0, 'ingredient': 1, 'transport_agency': 2, 'house_place': 3, 'song_name': 4, 'place_name': 5, 'player_setting': 6
Adding numeric intent and slot labels to the datasets
Parameter 'function'=<function DatasetCreator.add_numeric_labels.<locals>.create_numeric_labels at 0x7f136a92a560> of the transform datasets.arrow_dataset.Dataset._map_single couldn't be
100% 11514/11514 [00:01<00:00, 9639.89ex/s]
100% 2033/2033 [00:00<00:00, 9556.08ex/s]
100% 2974/2974 [00:00<00:00, 9856.36ex/s]
dataset: Dataset({
    features: ['id', 'locale', 'domain', 'intent_str', 'annot_utt', 'utt', 'slots_str', 'slots_num', 'intent_num'],
    num_rows: 11514
})
row 7: {'id': '10', 'locale': 'fa-IR', 'domain': 'iot', 'intent_str': 'iot_hue_lightchange', 'annot_utt': 'لطفا روشنایی را در حالت [color_type : مناسب برای مطالعه] تنظیم کن', 'utt': ['کن' ,'تنظیم' ,'مطالعه
dataset: Dataset({
    features: ['id', 'locale', 'domain', 'intent_str', 'annot_utt', 'utt', 'slots_str', 'slots_num', 'intent_num'],
    num_rows: 2033
})
row 7: {'id': '58', 'locale': 'fa-IR', 'domain': 'takeaway', 'intent_str': 'takeaway_query', 'annot_utt': 'غذای [food_type : تایلندی] [order_type : بیرون\u200cبر] من را اطراف [place_name : تهران ا
dataset: Dataset({
    features: ['id', 'locale', 'domain', 'intent_str', 'annot_utt', 'utt', 'slots_str', 'slots_num', 'intent_num'],
    num_rows: 2974
})
row 7: {'id': '41', 'locale': 'fa-IR', 'domain': 'general', 'intent_str': 'general_quirky', 'annot_utt': 'من وضعیت روشنایی صفحه را می خواهم', 'utt': ['خواهم' ,'می' ,'را' ,'صفحه' ,'روشنایی' ,'وضعیت'
```

Figure 23 Creating Dataset

The create_hf_dataset.py python file has a class which prepare the dataset and parse Jason file. This class if for creating four dataset splits, in the Huggingface Datasets Apache Arrow format from the MASSIVE dataset.

Each dataset split has the following **columns**:

"id", "locale", "utt", "annot_utt", "domain", "intent_str", "intent_num", "slots_str", "slots_num"

Methods:

- ✓ create_datasets(data_path): Creates the dataset splits using the data_path of the MASSIVE set
- ✓ add_numeric_labels(): Create integer versions of intents and slot for modeling
- ✓ investigate_datasets(): Prints out the seventh example from each dataset split as gut check
- ✓ save_label_dicts(prefix): Saves the mappings to the integer versions of the labels
- ✓ save_datasets(out_prefix): Saves the datasets to out_prefix

In this implementation each intent and slot is mapped to a number in order to make it possible to process and use them in training.

Training

I cloned the MASSIVE github to get the needed python files.

```
[5] !git clone https://github.com/alexa/massive

Cloning into 'massive'...
remote: Enumerating objects: 169, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 169 (delta 6), reused 6 (delta 4), pack-reused 142
Receiving objects: 100% (169/169), 120.71 KiB | 7.10 MiB/s, done.
Resolving deltas: 100% (72/72), done.
```

Figure 24 Cloning Massive

Here I used the xlm-roberta-base model for training. For training we use the following script and pass a config file to set the parameters of the model. This file is included in the folder uploaded named “train_config.yml”. Overally, in train.py the following procedure flows:

- ✓ parsing the args
- ✓ creating the massive.configuration master config object
- ✓ Seting up logging

- ✓ Getting all inputs to the trainer
- ✓ Getting the right trainer

As I mentioned before, we need to pass a configuration file. We need to especify **pretrained_weights** and **vocab_file** in this file. For these to fields I used pytorch_model.bin and sentencepiece.bpe.model in xlm-roberta-base of huggingface:

```
[6] !wget https://huggingface.co/xlm-roberta-base/resolve/main/sentencepiece.bpe.model
    !wget https://huggingface.co/xlm-roberta-base/resolve/main/pytorch_model.bin

--2022-06-25 07:28:59--  https://huggingface.co/xlm-roberta-base/resolve/main/sentencepiece.bpe.model
Resolving huggingface.co (huggingface.co)... 184.72.242.196, 34.192.62.10, 2600:1f18:147f:e800:e3ed:f4c9:35c2:ea6b, ...
Connecting to huggingface.co (huggingface.co)|184.72.242.196|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5069051 (4.8M) [application/octet-stream]
Saving to: 'sentencepiece.bpe.model'

sentencepiece.bpe.m 100%[===================>]   4.83M  9.57MB/s    in 0.5s

2022-06-25 07:29:00 (9.57 MB/s) - 'sentencepiece.bpe.model' saved [5069051/5069051]

--2022-06-25 07:29:00--  https://huggingface.co/xlm-roberta-base/resolve/main/pytorch_model.bin
Resolving huggingface.co (huggingface.co)... 184.72.242.196, 34.192.62.10, 2600:1f18:147f:e800:e3ed:f4c9:35c2:ea6b, ...
Connecting to huggingface.co (huggingface.co)|184.72.242.196|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cdn-lfs.huggingface.co/xlm-roberta-base/9d83baaafea92d36de26002c8135a427d55ee6fdc4faaa6e400be4c47724a07e?response-content-disposition=attachment%3B%20filename%3D%22pytc
--2022-06-25 07:29:01--  https://cdn-lfs.huggingface.co/xlm-roberta-base/9d83baaafea92d36de26002c8135a427d55ee6fdc4faaa6e400be4c47724a07e?response-content-disposition=attachment%3B%20fil
Resolving cdn-lfs.huggingface.co (cdn-lfs.huggingface.co)... 65.8.66.15, 65.8.66.95, 65.8.66.50, ...
Connecting to cdn-lfs.huggingface.co (cdn-lfs.huggingface.co)|65.8.66.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1115590446 (1.0G) [application/octet-stream]
Saving to: 'pytorch_model.bin'

pytorch_model.bin   100%[===================>]   1.04G   137MB/s    in 9.2s

2022-06-25 07:29:10 (115 MB/s) - 'pytorch_model.bin' saved [1115590446/1115590446]
```

Figure 25 Getting xlm-roberta-base files

I trained the model with batch_size of 128 and 45 epochs. Configuration file can be fully observed in the uploaded file but I will explain some of the parameters here.

The model and tokenizer parameters are:

```
model:
  type: xlmr intent classification slot filling
  size: base
  pretrained_weights: pytorch_model.bin
  pretrained_weight_substring_transform: ['roberta', 'xlmr']
  strict_load_pretrained_weights: false
  model_config_args:
    attention_probs_dropout_prob: 0.0
    bos_token_id: 0
    eos_token_id: 2
    hidden_act: gelu
    hidden_dropout_prob: 0.45
    hidden_size: 768
    initializer_range: 0.02
    intermediate_size: 3072
    layer_norm_eps: 1e-05
    max_position_embeddings: 514
    num_attention_heads: 12
    num_hidden_layers: 12
    output_past: true
    pad_token_id: 1
    type_vocab_size: 1
    vocab_size: 250002
    use_crf: false
    slot_loss_coef: 4.0
    hidden_layer_for_class: 11
    head_num_layers: 1
    head_layer_dim: 2048
    head_intent_pooling: max

tokenizer:
  type: xlmr base
  tok_args:
    vocab_file: sentencepiece.bpe.model
    max_len: *max_length
```

Figure 26 Parameters

As we can see we use the pytorch_model.bin for pretrained_weights and sentencepiece.bpe.model for vocab_file in xlm-roberta-base. The head layer dimension is 2048 and and number of hidden layers are 12. By setting these parameters we get the following architecture:


```
Model config RobertaConfig {
  "architectures": [
    "XLMRIntentClassSlotFill"
  ],
  "attention_probs_dropout_prob": 0.0,
  "bos_token_id": 0,
  "classifier_dropout": null,
  "eos_token_id": 2,
  "head_intent_pooling": "max",
  "head_layer_dim": 2048,
  "head_num_layers": 1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.45,
  "hidden_layer_for_class": 11,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-05,
  "max_position_embeddings": 514,
  "model_type": "roberta",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "output_past": true,
  "pad_token_id": 1,
  "position_embedding_type": "absolute",
  "slot_loss_coef": 4.0,
  "torch_dtype": "float32",
  "transformers_version": "4.20.1",
  "type_vocab_size": 1,
  "use_cache": true,
  "use_crf": false,
  "vocab_size": 250002
}
```


Figure 27 Model architecture details

Train and validation parameters are:

```yaml
train_val:
  train_dataset: data.train
  dev_dataset: data.dev
  intent_labels: data.intents
  slot_labels: data.slots
  slot_labels_ignore:
    - Other
  eval_metrics: all
  trainer_args:
    save_total_limit: 1
    output_dir: checkpoints/
    save_strategy: epoch
    evaluation_strategy: epoch
    learning_rate: 2.8e-05
    lr_scheduler_type: constant_with_warmup
    warmup_steps: 800
    adam_beta1: 0.9
    adam_beta2: 0.9999
    adam_epsilon: 1.0e-08
    weight_decay: 0.21
    gradient_accumulation_steps: 1
    per_device_train_batch_size: 128
    per_device_eval_batch_size: 128
    num_train_epochs: 45
    remove_unused_columns: false
    label_names:
      - intent_num
      - slots_num
    logging_steps: 100
    log_level: info
    locale_eval_strategy: all and each
    disable_tqdm: false
```

Figure 28 Parameters

As we can see the batch size is 128, learning rate is initially set to 0.000028. we used the adam optimizer and the epochs are 45. Pay attention since in this implementation there is a “other” slot which corresponds to no slots defined we ignore it.

```
!python massive/scripts/train.py -c train_config.yml

[INFO] 02:54 >> Starting the run at 2022-06-25 02:54:48.691496
[INFO] 02:54 >> Using the following config: {'run_name': 'xlmr_base', 'max_length': 512, 'model': {'type': 'xlmr intent classification slot filling', 'size': 'base', 'pretrained_weig
[INFO] 02:54 >> Caching indices mapping at data.train/cache-985f81c97832a490.arrow
[INFO] 02:54 >> The features of the train dataset: {'id': Value(dtype='string', id=None), 'locale': Value(dtype='string', id=None), 'domain': Value(dtype='string', id=None), 'intent_
[INFO] 02:54 >> Length of the train dataset: 11514
[INFO] 02:54 >> Caching indices mapping at data.dev/cache-fbeb76d932f67e17.arrow
[INFO] 02:54 >> The features of the train dataset: {'id': Value(dtype='string', id=None), 'locale': Value(dtype='string', id=None), 'domain': Value(dtype='string', id=None), 'intent_
[INFO] 02:54 >> Length of the train dataset: 11514
[INFO] 02:54 >> Length of the loaded dev dataset: 2033
[INFO] 02:54 >> Loading pretrained weights from pytorch_model.bin
[INFO] 02:54 >> Finished loading pretrained model. Results: _IncompatibleKeys(missing_keys=['xlmr.embeddings.position_ids', 'intent_classifier.ic_head.1.weight', 'intent_classifier.i
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the Py
  FutureWarning,
***** Running training *****
  Num examples = 11514
  Num Epochs = 40
  Instantaneous batch size per device = 128
  Total train batch size (w. parallel, distributed & accumulation) = 128
  Gradient Accumulation steps = 1
  Total optimization steps = 3600
  0% 0/3600 [00:00<?, ?it/s][INFO] 02:55 >> Collator Check! utt: ['نبینم' ,'را' ,'آن' ,'دوباره' ,'که' ,'کن' ,'حذف' ,'را' ,'من' ,'تقویم' ,'در' ,'بعدی' ,'رویداد']; intent label: 40; slot labels: [15,
[INFO] 02:55 >> Collator Check! utt: ['ها' ,'رستوران' ,'بررسی']; intent label: 18; slot labels: [15, 32, 32], tokenized utt: Encoding(num_tokens=5, attributes=[ids, type_ids, tokens, off
[INFO] 02:55 >> Collator Check! utt: ['بده' ,'نشان' ,'من' ,'به' ,'است' ,'جریان' ,'در' ,'حاضر' ,'حال' ,'در' ,'که' ,'را' ,'رویدادهایی']; intent label: 31; slot labels: [15, 15, 15, 15, 15, 15, 15, 1
[INFO] 02:55 >> Collator Check! utt: ['کن' ,'تبدیل' ,'متن' ,'به' ,'رو' ,'ایمیل' ,'محتوی']; intent label: 23; slot labels: [15, 15, 15, 15, 15, 15, 15], tokenized utt: Encoding(num_tokens=10,
[INFO] 02:55 >> Collator Check! utt: ['کن' ,'پاک' ,'را' ,'تقویم' ,'رویدادهای' ,'تمام']; intent label: 40; slot labels: [15, 15, 15, 15, 15, 15], tokenized utt: Encoding(num_tokens=10, attribut
  2% 90/3600 [00:56<36:25,  1.61it/s]
  0% 0/3 [00:00<?, ?ba/s]
100% 3/3 [00:00<00:00, 29.63ba/s]
***** Running Evaluation *****
  Num examples = 2033
  Batch size = 128

  0% 0/16 [00:00<?, ?it/s]
 12% 2/16 [00:00<00:01,  9.84it/s]
 19% 3/16 [00:00<00:01,  7.08it/s]
 25% 4/16 [00:00<00:02,  5.90it/s]
 31% 5/16 [00:00<00:01,  6.03it/s]
 38% 6/16 [00:00<00:01,  5.67it/s]
 44% 7/16 [00:01<00:01,  4.92it/s]
 50% 8/16 [00:01<00:01,  4.92it/s]
 56% 9/16 [00:01<00:01,  5.11it/s]
```

Figure 29 Training

After training by 45 epoches this is the **results**:

{'training_global_step': 3600, **'training_epoch': 45.0**, 'eval_fa-IR_loss': 1.8478409051895142, **'eval_fa-IR_intent_acc': 0.8494835218888342,** 'eval_fa-IR_intent_acc_stderr': 0.007930504466678288, **'eval_fa-IR_slot_micro_f1': 0.7392102846648301,** 'eval_fa-IR_slot_micro_f1_stderr': 0.0015396802730499027, 'eval_fa-IR_ex_match_acc': 0.6148548942449582, 'eval_fa-IR_ex_match_acc_stderr': 0.010792692898620112, 'eval_fa-IR_runtime': 3.6738, 'eval_fa-IR_samples_per_second': 553.373, 'eval_fa-IR_steps_per_second': 4.355, **'eval_all_loss': 1.8478409051895142**, **'eval_all_intent_acc': 0.8494835218888342**, 'eval_all_intent_acc_stderr': 0.007930504466678288, **'eval_all_slot_micro_f1': 0.7392102846648301**, 'eval_all_slot_micro_f1_stderr': 0.0015396802730499027, **'eval_all_ex_match_acc': 0.6148548942449582**, 'eval_all_ex_match_acc_stderr': 0.010792692898620112, 'eval_all_runtime': 3.669, 'eval_all_samples_per_second': 554.109, 'eval_all_steps_per_second': 4.361, 'eval_highest-locale_intent_acc': 'fa-IR', 'eval_highest-locale-val_intent_acc': 0.8494835218888342, 'eval_lowest-locale_intent_acc': 'fa-IR', 'eval_lowest-locale-val_intent_acc': 0.8494835218888342, 'eval_highest-locale_slot_micro_f1': 'fa-IR', 'eval_highest-locale-val_slot_micro_f1': 0.7392102846648301, 'eval_lowest-locale_slot_micro_f1': 'fa-IR', 'eval_lowest-locale-val_slot_micro_f1': 0.7392102846648301, 'eval_highest-locale_ex_match_acc': 'fa-IR', 'eval_highest-locale-val_ex_match_acc': 0.6148548942449582, 'eval_lowest-locale_ex_match_acc': 'fa-IR', 'eval_lowest-locale-val_ex_match_acc': 0.6148548942449582, 'epoch': 45.0}

Testing

For testing we use the following script and pass a config file to set the parameters of the model. This file is included in the folder uploaded named “test_config.yml”. These parameters are as follows:

```
model:
  type: xlmr intent classification slot filling
  checkpoint: checkpoints/checkpoint-3600

tokenizer:
  type: xlmr base
  tok_args:
    vocab_file: sentencepiece.bpe.model
    max_len: *max_length

collator:
  type: massive intent class slot fill
  args:
    max_length: *max_length
    padding: longest

test:
  test_dataset: data.test
  intent_labels: data.intents
  slot_labels: data.slots
  massive_path: massive/
  slot_labels_ignore:
    - Other
  eval_metrics: all
  predictions_file: /content/preds_xlmr_45_epoch.jsonl
  trainer_args:
    output_dir: checkpoints/
    per_device_eval_batch_size: 128
    remove_unused_columns: false
    label_names:
      - intent_num
      - slots_num
    log_level: info
    logging_strategy: no
    locale_eval_strategy: all only
    disable_tqdm: false
```

The batch size is 128 and I save the predictions in preds_xlmr_45_epoch.jsonl file.

```
[ ] !python massive/scripts/test.py -c /content/test_config.yml
      "head_layer_dim": 2048,
      "head_num_layers": 1,
      "hidden_act": "gelu",
      "hidden_dropout_prob": 0.45,
      "hidden_layer_for_class": 11,
      "hidden_size": 768,
      "initializer_range": 0.02,
      "intermediate_size": 3072,
      "layer_norm_eps": 1e-05,
      "max_position_embeddings": 514,
      "model_type": "roberta",
      "num_attention_heads": 12,
      "num_hidden_layers": 12,
      "output_past": true,
      "pad_token_id": 1,
      "position_embedding_type": "absolute",
      "slot_loss_coef": 4.0,
      "torch_dtype": "float32",
      "transformers_version": "4.20.1",
      "type_vocab_size": 1,
      "use_cache": true,
      "use_crf": false,
      "vocab_size": 250002
    }

    loading weights file checkpoints/checkpoint-3600/pytorch_model.bin
    All model checkpoint weights were used when initializing XLMRIntentClassSlotFill.

    All the weights of XLMRIntentClassSlotFill were initialized from the model checkpoint at checkpoints/checkpoint-3600.
    If your task is similar to the task the model of the checkpoint was trained on, you can already use XLMRIntentClassSlotFill for predictions without further training.
    ***** Running Evaluation *****
      Num examples = 2974
      Batch size = 128
    [INFO] 03:52 >> Collator Check! utt: ['دوشنبه', 'بعدی', 'را', 'علامت', 'بزن']; intent label: 35; slot labels: [0, 15, 15, 15, 15], tokenized utt: Encoding(num_tokens=7, attributes=[ids, typ
    [INFO] 03:52 >> Collator Check! utt: ['لیست', 'بیشترین', 'آهنگ', 'هایی', 'که', 'پخش', 'شده', 'را', 'برای', 'من', 'پخش', 'کن']; intent label: 39; slot labels: [15, 15, 15, 15, 15, 15, 15, 15, 15
    [INFO] 03:52 >> Collator Check! utt: ['زنگ', 'هشدار', 'را', 'برای', 'سه', 'بعدازظهر', 'غیرفعال', 'کن']; intent label: 24; slot labels: [15, 15, 15, 15, 46, 46, 15, 15], tokenized utt: Encoding
    [INFO] 03:52 >> Collator Check! utt: ['زنگ', 'هشدار', 'را', 'کنسل', 'کن']; intent label: 24; slot labels: [15, 15, 15, 15, 15], tokenized utt: Encoding(num_tokens=8, attributes=[ids, typ
    [INFO] 03:52 >> Collator Check! utt: ['برنامه', 'من', 'برای', 'بعد', 'از', 'ظهر', 'را', 'به', 'من', 'یادآوری', 'کن']; intent label: 35; slot labels: [15, 15, 15, 30, 30, 30, 15, 15, 15, 15, 15],
    100% 24/24 [00:05<00:00,  4.28it/s]
```

Figure 30 Testing

Results of testing are:

{ **'test_all_ex_match_acc': 0.6146603900470746,**
'test_all_ex_match_acc_stderr': 0.008924193086447684,
'test_all_intent_acc': 0.8490248823133827,
'test_all_intent_acc_stderr': 0.006565116171480395,
'test_all_loss': 1.9511113166809082,
'test_all_runtime': 6.1897,

'test_all_samples_per_second': 480.475,
'test_all_slot_micro_f1': 0.7281713344316308,
'test_all_slot_micro_f1_stderr': 0.0009614521731479325,
'test_all_steps_per_second': 3.877
}

As we can see the results of the model are very good and close to the accuracies mentioned in the [Massive paper.]

The results reported in the paper are:

	Exact Match Accuracy (%)					
	mT5 T2T Full	mT5 Enc Full	XLM-R Full	mT5 T2T Zero	mT5 Enc Zero	XLM-R Zero
th-TH	73.4 ± 1.6	72.3 ± 1.6	70.1 ± 1.6	33.5 ± 1.7	40.8 ± 1.8	46.3 ± 1.8
en-US	72.5 ± 1.6	72.0 ± 1.6	69.7 ± 1.7			
sv-SE	71.2 ± 1.6	70.6 ± 1.6	69.7 ± 1.7	53.2 ± 1.8	44.3 ± 1.8	57.9 ± 1.8
da-DK	70.2 ± 1.6	70.3 ± 1.6	68.2 ± 1.7	47.6 ± 1.8	41.0 ± 1.8	54.4 ± 1.8
my-MM	70.1 ± 1.6	69.4 ± 1.7	65.5 ± 1.7	24.4 ± 1.5	22.2 ± 1.5	33.1 ± 1.7
nb-NO	70.0 ± 1.6	68.8 ± 1.7	66.8 ± 1.7	48.5 ± 1.8	41.0 ± 1.8	53.7 ± 1.8
nl-NL	69.4 ± 1.7	68.1 ± 1.7	66.6 ± 1.7	52.4 ± 1.8	41.0 ± 1.8	51.7 ± 1.8
ru-RU	69.2 ± 1.7	67.2 ± 1.7	66.2 ± 1.7	50.5 ± 1.8	42.6 ± 1.8	52.8 ± 1.8
fi-FI	69.1 ± 1.7	68.8 ± 1.7	66.9 ± 1.7	41.3 ± 1.8	35.8 ± 1.7	49.8 ± 1.8
ms-MY	69.1 ± 1.7	67.3 ± 1.7	65.6 ± 1.7	39.3 ± 1.8	33.1 ± 1.7	45.5 ± 1.8
de-DE	69.0 ± 1.7	68.9 ± 1.7	65.7 ± 1.7	52.0 ± 1.8	40.0 ± 1.8	45.4 ± 1.8
ko-KR	68.8 ± 1.7	68.0 ± 1.7	67.5 ± 1.7	33.7 ± 1.7	24.1 ± 1.5	44.8 ± 1.8
ro-RO	68.6 ± 1.7	65.1 ± 1.7	64.5 ± 1.7	45.4 ± 1.8	35.7 ± 1.7	51.6 ± 1.8
id-ID	68.6 ± 1.7	67.2 ± 1.7	64.8 ± 1.7	46.0 ± 1.8	37.4 ± 1.7	50.7 ± 1.8
af-ZA	68.3 ± 1.7	66.8 ± 1.7	64.9 ± 1.7	39.9 ± 1.8	34.9 ± 1.7	43.9 ± 1.8
tr-TR	68.1 ± 1.7	67.7 ± 1.7	65.2 ± 1.7	37.2 ± 1.7	27.4 ± 1.6	43.8 ± 1.8
el-GR	67.8 ± 1.7	66.7 ± 1.7	64.0 ± 1.7	43.5 ± 1.8	36.8 ± 1.7	41.9 ± 1.8
pt-PT	67.6 ± 1.7	66.0 ± 1.7	64.6 ± 1.7	47.6 ± 1.8	39.8 ± 1.8	48.6 ± 1.8
hu-HU	67.2 ± 1.7	67.7 ± 1.7	65.4 ± 1.7	38.7 ± 1.8	33.7 ± 1.7	44.7 ± 1.8
az-AZ	67.2 ± 1.7	66.2 ± 1.7	65.2 ± 1.7	28.3 ± 1.6	20.2 ± 1.4	37.2 ± 1.7
is-IS	67.1 ± 1.7	66.8 ± 1.7	64.3 ± 1.7	28.5 ± 1.6	23.4 ± 1.5	32.7 ± 1.7
ml-IN	67.1 ± 1.7	67.2 ± 1.7	64.9 ± 1.7	32.5 ± 1.7	27.2 ± 1.6	40.1 ± 1.8
lv-LV	67.0 ± 1.7	67.0 ± 1.7	66.6 ± 1.7	34.3 ± 1.7	27.4 ± 1.6	37.8 ± 1.7
it-IT	66.8 ± 1.7	64.8 ± 1.7	63.1 ± 1.7	45.1 ± 1.8	38.1 ± 1.7	45.2 ± 1.8
all	66.6 ± 0.2	65.9 ± 0.2	63.7 ± 0.2	34.7 ± 0.2	28.8 ± 0.2	38.7 ± 0.2
jv-ID	66.6 ± 1.7	65.4 ± 1.7	59.3 ± 1.8	19.0 ± 1.4	15.3 ± 1.3	11.7 ± 1.2
sq-AL	66.5 ± 1.7	65.1 ± 1.7	63.6 ± 1.7	35.5 ± 1.7	28.9 ± 1.6	35.1 ± 1.7
he-IL	66.2 ± 1.7	65.9 ± 1.7	64.5 ± 1.7	28.1 ± 1.6	26.6 ± 1.6	37.8 ± 1.7
es-ES	66.2 ± 1.7	64.3 ± 1.7	62.8 ± 1.7	50.4 ± 1.8	39.7 ± 1.8	47.6 ± 1.8
fr-FR	66.2 ± 1.7	65.1 ± 1.7	62.2 ± 1.7	47.2 ± 1.8	39.5 ± 1.8	48.6 ± 1.8
bn-BD	66.2 ± 1.7	66.0 ± 1.7	63.4 ± 1.7	27.3 ± 1.6	21.6 ± 1.5	36.3 ± 1.7
hy-AM	66.1 ± 1.7	65.8 ± 1.7	63.1 ± 1.7	34.8 ± 1.7	26.3 ± 1.6	39.0 ± 1.8
mn-MN	66.0 ± 1.7	65.4 ± 1.7	63.4 ± 1.7	24.3 ± 1.5	16.4 ± 1.3	33.3 ± 1.7
fa-IR	65.9 ± 1.7	67.3 ± 1.7	67.0 ± 1.7	38.7 ± 1.8	31.5 ± 1.7	49.6 ± 1.8
sl-SI	65.9 ± 1.7	65.6 ± 1.7	64.3 ± 1.7	36.3 ± 1.7	29.9 ± 1.6	38.4 ± 1.7
tl-PH	65.6 ± 1.7	65.6 ± 1.7	61.1 ± 1.8	34.3 ± 1.7	26.9 ± 1.6	26.9 ± 1.6

Figure 31 Exact Match Accuracy in the paper

Intent Accuracy (%)						
	mT5 T2T Full	mT5 Enc Full	XLM-R Full	mT5 T2T Zero	mT5 Enc Zero	XLM-R Zero
en-US	87.9 ± 1.2	89.0 ± 1.1	88.3 ± 1.2			
sv-SE	87.8 ± 1.2	88.5 ± 1.1	87.9 ± 1.2	77.1 ± 1.5	76.0 ± 1.5	85.2 ± 1.3
nb-NO	87.6 ± 1.2	87.7 ± 1.2	87.3 ± 1.2	76.3 ± 1.5	72.8 ± 1.6	83.6 ± 1.3
da-DK	87.5 ± 1.2	88.0 ± 1.2	86.9 ± 1.2	76.8 ± 1.5	73.4 ± 1.6	83.1 ± 1.3
ro-RO	87.2 ± 1.2	87.0 ± 1.2	86.9 ± 1.2	73.0 ± 1.6	70.1 ± 1.6	80.8 ± 1.4
nl-NL	87.2 ± 1.2	87.6 ± 1.2	86.8 ± 1.2	79.9 ± 1.4	76.4 ± 1.5	82.1 ± 1.4
ru-RU	87.0 ± 1.2	86.8 ± 1.2	87.2 ± 1.2	76.2 ± 1.5	73.8 ± 1.6	81.3 ± 1.4
id-ID	87.0 ± 1.2	86.8 ± 1.2	87.1 ± 1.2	77.0 ± 1.5	74.1 ± 1.6	83.1 ± 1.3
fr-FR	86.9 ± 1.2	87.2 ± 1.2	86.3 ± 1.2	76.9 ± 1.5	74.1 ± 1.6	80.8 ± 1.4
it-IT	86.8 ± 1.2	87.6 ± 1.2	86.6 ± 1.2	72.3 ± 1.6	71.5 ± 1.6	76.4 ± 1.5
ms-MY	86.8 ± 1.2	86.9 ± 1.2	86.1 ± 1.2	69.9 ± 1.6	66.0 ± 1.7	76.7 ± 1.5
es-ES	86.7 ± 1.2	86.8 ± 1.2	86.9 ± 1.2	76.6 ± 1.5	75.9 ± 1.5	78.8 ± 1.5
pt-PT	86.7 ± 1.2	86.9 ± 1.2	86.7 ± 1.2	74.0 ± 1.6	74.5 ± 1.6	79.5 ± 1.5
fa-IR	86.3 ± 1.2	87.2 ± 1.2	87.0 ± 1.2	69.0 ± 1.7	66.3 ± 1.7	81.1 ± 1.4
pl-PL	86.3 ± 1.2	87.1 ± 1.2	85.8 ± 1.3	76.4 ± 1.5	74.1 ± 1.6	80.7 ± 1.4
de-DE	86.2 ± 1.2	86.8 ± 1.2	85.7 ± 1.3	77.3 ± 1.5	73.9 ± 1.6	77.6 ± 1.5
az-AZ	86.2 ± 1.2	86.4 ± 1.2	86.2 ± 1.2	57.0 ± 1.8	55.5 ± 1.8	70.9 ± 1.6
tr-TR	86.1 ± 1.2	87.1 ± 1.2	86.3 ± 1.2	66.5 ± 1.7	63.7 ± 1.7	78.4 ± 1.5
ko-KR	86.1 ± 1.2	86.4 ± 1.2	86.5 ± 1.2	60.0 ± 1.8	61.9 ± 1.7	77.0 ± 1.5
af-ZA	86.0 ± 1.2	86.9 ± 1.2	85.6 ± 1.3	68.5 ± 1.7	66.5 ± 1.7	71.7 ± 1.6
ml-IN	86.0 ± 1.2	86.5 ± 1.2	85.1 ± 1.3	60.6 ± 1.8	57.8 ± 1.8	70.1 ± 1.6
sq-AL	85.9 ± 1.3	86.4 ± 1.2	86.4 ± 1.2	62.9 ± 1.7	62.0 ± 1.7	67.6 ± 1.7
sl-SL	85.9 ± 1.3	86.8 ± 1.2	86.3 ± 1.2	61.5 ± 1.7	59.8 ± 1.8	69.5 ± 1.7
el-GR	85.8 ± 1.3	86.6 ± 1.2	86.2 ± 1.2	71.9 ± 1.6	69.8 ± 1.6	74.0 ± 1.6

Figure 32 Intent Accuracy in the paper

Micro-Averaged Slot F1 (%)						
	mT5 T2T Full	mT5 Enc Full	XLM-R Full	mT5 T2T Zero	mT5 Enc Zero	XLM-R Zero
th-TH	86.8 ± 0.7	85.7 ± 0.7	83.5 ± 0.7	34.5 ± 0.9	59.5 ± 1.0	57.4 ± 1.0
my-MM	82.2 ± 0.7	82.1 ± 0.7	79.0 ± 0.7	26.0 ± 0.8	38.0 ± 0.9	48.9 ± 0.9
en-US	81.6 ± 0.5	80.4 ± 0.5	78.7 ± 0.6			
km-KH	81.0 ± 0.8	81.9 ± 0.8	77.9 ± 0.8	27.9 ± 0.9	58.2 ± 1.0	53.6 ± 1.0
sv-SE	80.9 ± 0.6	79.6 ± 0.6	78.5 ± 0.6	64.2 ± 0.7	56.8 ± 0.7	68.4 ± 0.7
nb-NO	80.0 ± 0.6	77.8 ± 0.6	76.0 ± 0.6	58.8 ± 0.7	56.0 ± 0.7	65.1 ± 0.7
ko-KR	79.6 ± 0.7	78.9 ± 0.7	77.8 ± 0.7	46.8 ± 0.8	36.0 ± 0.8	56.0 ± 0.8
da-DK	79.4 ± 0.6	79.1 ± 0.6	77.7 ± 0.6	58.5 ± 0.7	54.6 ± 0.7	64.6 ± 0.7
fi-FI	79.4 ± 0.7	79.2 ± 0.7	77.2 ± 0.7	49.1 ± 0.8	48.9 ± 0.8	62.1 ± 0.8
de-DE	78.8 ± 0.6	78.6 ± 0.6	76.2 ± 0.6	64.3 ± 0.7	55.6 ± 0.7	60.0 ± 0.7
ru-RU	78.7 ± 0.6	76.3 ± 0.6	74.9 ± 0.6	61.6 ± 0.7	55.4 ± 0.7	63.3 ± 0.7
ms-MY	78.4 ± 0.6	77.4 ± 0.6	75.5 ± 0.6	51.5 ± 0.7	48.2 ± 0.7	55.9 ± 0.7
af-ZA	78.3 ± 0.6	76.5 ± 0.6	74.6 ± 0.6	51.9 ± 0.7	52.3 ± 0.7	57.3 ± 0.7
is-IS	78.2 ± 0.6	77.7 ± 0.6	75.2 ± 0.6	39.3 ± 0.7	37.9 ± 0.7	45.2 ± 0.7
nl-NL	78.1 ± 0.6	76.5 ± 0.6	75.5 ± 0.6	61.6 ± 0.7	54.3 ± 0.7	62.4 ± 0.7
jv-ID	78.1 ± 0.6	76.1 ± 0.6	70.9 ± 0.7	29.6 ± 0.7	26.7 ± 0.7	24.7 ± 0.6
hu-HU	78.0 ± 0.6	77.5 ± 0.6	75.3 ± 0.6	46.1 ± 0.7	45.8 ± 0.7	56.8 ± 0.7
tr-TR	77.9 ± 0.6	76.1 ± 0.7	74.9 ± 0.7	48.8 ± 0.8	41.9 ± 0.8	52.8 ± 0.8
lv-LV	77.8 ± 0.6	77.1 ± 0.6	76.3 ± 0.6	47.2 ± 0.8	41.6 ± 0.7	53.0 ± 0.8
ka-GE	77.6 ± 0.7	77.1 ± 0.7	76.8 ± 0.7	43.5 ± 0.9	48.6 ± 0.9	55.9 ± 0.9
ro-RO	77.6 ± 0.6	74.1 ± 0.6	72.4 ± 0.6	56.3 ± 0.7	48.6 ± 0.7	60.8 ± 0.7
el-GR	77.0 ± 0.6	75.5 ± 0.6	73.4 ± 0.6	54.8 ± 0.7	51.7 ± 0.7	54.4 ± 0.7
id-ID	76.9 ± 0.6	75.6 ± 0.6	73.6 ± 0.6	55.6 ± 0.7	51.0 ± 0.7	59.7 ± 0.7
all	76.8 ± 0.1	75.4 ± 0.1	73.6 ± 0.1	44.8 ± 0.1	41.6 ± 0.1	50.3 ± 0.1
az-AZ	76.8 ± 0.6	75.6 ± 0.7	74.1 ± 0.7	40.4 ± 0.7	33.8 ± 0.7	46.6 ± 0.8
he-IL	76.7 ± 0.6	75.1 ± 0.7	74.0 ± 0.7	30.6 ± 0.7	35.5 ± 0.7	49.3 ± 0.8
pt-PT	76.6 ± 0.6	74.9 ± 0.6	73.3 ± 0.6	56.3 ± 0.7	46.6 ± 0.7	58.2 ± 0.7
ml-IN	76.6 ± 0.7	76.1 ± 0.7	74.8 ± 0.7	42.1 ± 0.8	45.5 ± 0.8	52.5 ± 0.8
it-IT	76.4 ± 0.6	73.7 ± 0.6	72.3 ± 0.6	58.7 ± 0.7	50.0 ± 0.7	57.3 ± 0.7
bn-BD	76.4 ± 0.6	75.1 ± 0.6	73.4 ± 0.6	39.6 ± 0.7	37.2 ± 0.7	52.3 ± 0.7
cy-GB	76.3 ± 0.6	73.5 ± 0.6	71.2 ± 0.6	21.8 ± 0.6	21.5 ± 0.5	30.1 ± 0.6
sq-AL	75.9 ± 0.6	73.7 ± 0.6	72.0 ± 0.6	48.3 ± 0.7	41.9 ± 0.7	50.0 ± 0.7
tl-PH	75.8 ± 0.6	74.6 ± 0.6	71.6 ± 0.6	44.7 ± 0.6	37.1 ± 0.6	36.1 ± 0.6
mn-MN	75.8 ± 0.6	74.1 ± 0.6	73.7 ± 0.7	36.6 ± 0.7	26.9 ± 0.7	45.0 ± 0.7
ar-SA	75.7 ± 0.7	75.4 ± 0.7	73.8 ± 0.7	39.7 ± 0.8	44.6 ± 0.8	48.4 ± 0.8
fr-FR	75.6 ± 0.6	73.5 ± 0.6	70.9 ± 0.6	54.2 ± 0.7	51.2 ± 0.7	59.1 ± 0.7
es-ES	75.5 ± 0.6	72.8 ± 0.6	71.0 ± 0.6	61.1 ± 0.7	50.4 ± 0.7	57.1 ± 0.7
fa-IR	75.4 ± 0.6	76.6 ± 0.6	76.6 ± 0.6	49.4 ± 0.7	46.9 ± 0.7	60.2 ± 0.6
sl-SL	75.4 ± 0.6	74.3 ± 0.6	72.2 ± 0.7	49.0 ± 0.7	45.6 ± 0.7	53.1 ± 0.7
hy-AM	75.3 ± 0.7	74.1 ± 0.7	72.4 ± 0.7	41.7 ± 0.7	39.1 ± 0.7	50.0 ± 0.8
hi-IN	75.0 ± 0.6	73.5 ± 0.6	72.3 ± 0.6	49.6 ± 0.7	45.1 ± 0.7	54.6 ± 0.7

Figure 33 Micro-Averaged Slot F1 in the paper

Now, lets look at some prediction by more details:

Figure 34 Id 6725 - Actual

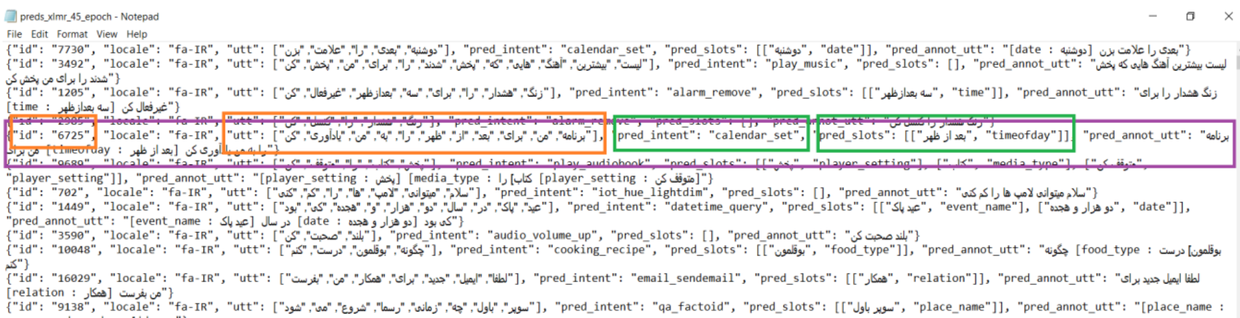

Figure 35 Id 6725 - Predicted

As we can see the prediction is correct and this confirms the results scores I showed above.

I also trained the model on mt5_enc_base. The results are:

'test_all_ex_match_acc': 0.594371217215870881,
 'test_all_ex_match_acc_stderr': 0.0012097048826613399,
 'test_all_intent_acc': 0.699394754539341,
 'test_all_intent_acc_stderr': 0.008407927933758254,
 'test_all_samples_per_second': 518.046,
 'test_all_slot_micro_f1': 0.70055148853099072,
 'test_all_steps_per_second': 8.187
}
Which we can see the results with xlm-roberta-base were better.

Some problems I encountered during completing this part:

Since creating the environment given in massive GitHub was not easy I tried to run the codes without that environment. So I installed packages and dependencies manually and by a req.txt file wich is included in the folder uploaded. Without the given environment I got some errors during running the train.py and test.py so, I needded to change some parts of the code especially the importing parts. After trying to handle these errors I realized by using this command all the erros will be gone.

```
%env PYTHONPATH=massive/src/
```

```
env: PYTHONPATH=massive/src/
```

Another problem was the disk capacity of google colab which leaded to errors. In order to handle this I used an filed named save_total_limit = 1 in the training config file which will save just the last checkpoint not all the checkpoints.

*** for running the note book file make sure you have the "Dataset" folder which included the "fa-IR.jsonl" file, the req.txt file and train_config.yml and test_config.yml.

References:

https://arxiv.org/pdf/2204.08582.pdf

https://arxiv.org/pdf/2010.11934.pdf

https://arxiv.org/abs/1911.02116

https://mdpi-res.com/d_attachment/sensors/sensors-21-01230/article_deploy/sensors-21-01230-v3.pdf?version=1614215625