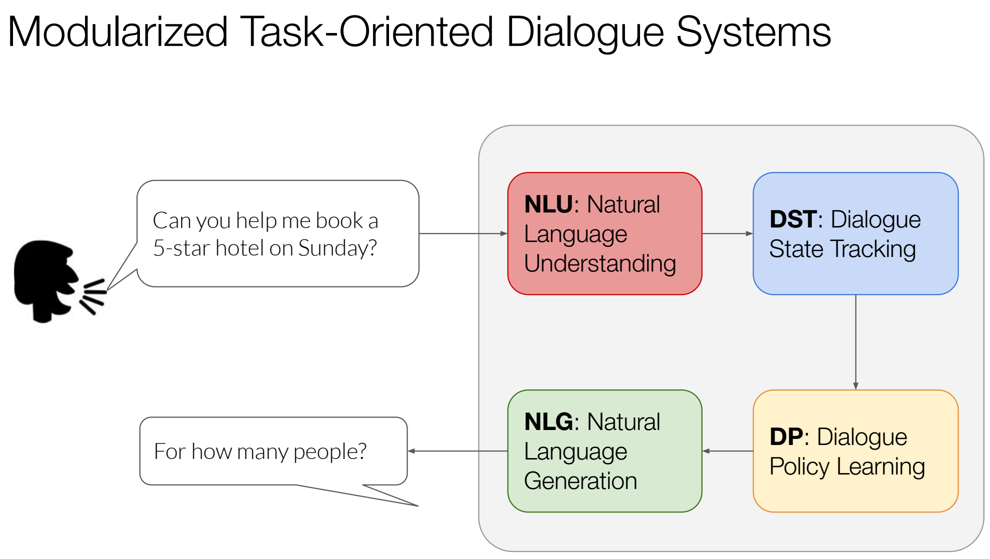
GPT2DST - Task Oriented Dialogue State Tracking with GPT-2

Contents

1 Introduction to the Practical

This exercise is an investigation into Natural Language Understanding (NLU) and Dialogue State Tracking (DST) using Hugging Face GPT-2 Transformer language models [1]. GPT-2 Transformer language models are pre-trained on very large heterogeneous text collections. After pre-training the models may not work well for very specialised tasks, such as Dialogue State Tracking. However pre-trained models can be fine-tuned with relatively small amounts of in-domain data, yielding surprisingly good performance, if care is taken in how the fine-tuning task is formulated. This is the approach to DST taken in this practical. A collection of pre-trained models are provided which you will use as the basis for your investigations. You will be asked to report performance in Natural Language Understanding and Dialogue State Tracking on the MultiWOZ task-oriented dialogue data sets, possibly after further fine-tuning of the models on the in-domain data.


Modularized Task-Oriented Dialogue Systems
Can you help me book a 5-star hotel on Sunday?
NLU: Natural Language Understanding
DST: Dialogue State Tracking
DP: Dialogue Policy Learning
NLG: Natural Language Generation
For how many people?

Recently published results in Dialogue State Tracking are summarised here : `https://github.com/budzianowski/multiwoz#belief-tracking`.

[1]Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing* `https://arxiv.org/abs/1910.03771`, `https://huggingface.co/transformers/`

1.1 HPC: Virtual Environment, Environment Variables, Working Directories

These practical instructions assume you will do your work on the HPC CSD3 cluster.

You can login to the CSD3 cluster using SSH:
`ssh CRSID@login-cpu.hpc.cam.ac.uk`
where `CRSID` is your CRSID. More information on how to connect is here:
`https://docs.hpc.cam.ac.uk/hpc/user-guide/connecting.html`

Working Directories: Please do not use your home directory for large-scale experiments. You can use

- `/rds/user/<username>/hpc-work` – this directory is private to you
- `/rds/project/rds-xyBFuSj0hm0/MLMI.2021-22/$CRSID` – already created for you

See `https://docs.hpc.cam.ac.uk/hpc/user-guide/io_management.html` for more information.

1.2 Virtual Environment for the Practical

Sourcing the following script will activate a Python Anaconda environment with the tools and libraries you will need for this exercise.

```
$ source /rds/project/rds-xyBFuSj0hm0/MLMI8.L2022/envs/README.MLMI8.GPT2DST.activate
```

IMPORTANT: You should deactivate all Anaconda, Miniconda or other virtual environments before running the above command. You can verify that all environments are deactivated by confirming that
`echo $CONDA_DEFAULT_ENV`
returns an empty response. After sourcing the above file, you can verify that the environment is activated:

```
$ which python
/rds/project/rds-xyBFuSj0hm0/MLMI8.L2022/envs/MLMI8.GPT2DST/bin/python
$ python --version
Python 3.8.12
```

1.3 Base Directories

Sourcing the file `README.MLMI8.GPT2DST.activate` should set the BaseDirectory environment variable `BDIR` :

```
$ echo $BDIR
/rds/project/rds-xyBFuSj0hm0/MLMI8.L2022/GPT2DST/
$ ls $BDIR/
checkpoints/ data_preparation/ hyps/
cc-dst.py  gpt2-dst.py multiwoz_dst_score.py sgd_parser.py train-gpt2-dst.py
dstdataset.py metrics.py sgd_dst_score.py test_parser.py utils.py
decode.nlu.mwoz21.slurm.wilkes2 train.nlu.mwoz21.slurm.wilkes2
slurm.example.1job slurm.example.2jobs
```

Please familiarise yourself with these files and directories in `$BDIR`:

- `checkpoints/` : Hugging Face GPT-2 models fine-tuned on MultiWOZ NLU training data

- `hyps/` : NLU hypotheses generated using GPT-2 models on MultiWOZ development and test sets
- `data_preparation/data/multiwoz21/processed/` : NLU training, development, and test sets in JSON format
- `data_preparation/data/multiwoz21/refs/` : dialogue reference annotations for automatic scoring of NLU and DST output
- `*.py` : training, decoding, and scoring scripts
- `slurm.example.1job, slurm.example.2jobs` : example job submission scripts
- `decode.nlu.mwoz21.slurm.wilkes2, train.nlu.mwoz21.slurm.wilkes2` : job submission scripts for model training and decoding

1.4 SLURM Job Submission

Note that the HPC login nodes are not meant for running long jobs. You can use them for compiling and debugging , but long running jobs will be killed automatically.
See `https://docs.hpc.cam.ac.uk/hpc/user-guide/connecting.html#more-on-login-nodes`

Longer running jobs should be submitted to the CSD3 cluster using the 'SLURM' workload management and job scheduling system. Extensive documentation is provided [2], but simple examples of submitting and monitoring jobs are given in Sections 1.4.2 and 1.4.3.

1.4.1 Your SLURM Project ID

Students registered for MLMI8 have been provided with access to two accounts: one account is for running jobs on the CPU cluster and the other account is running jobs on GPUs. You should be able to type

```
$ mybalance
```

to see how many hours you have in each account. Students are each initially allocated 10 GPU hours and 500 CPU hours. Your accounts should have this form, where `[crsid]` is your CRSID:

```
MLMI-[crsid]-SL2-CPU
MLMI-[crsid]-SL2-GPU
```

To get started, copy the scripts `$BDIR/slurm.example.1job` and `$BDIR/slurm.example.2jobs` to your working directory. Edit each file to replace the dummy account name `MLMI-[crsid]-SL2-CPU` by your own account, i.e. change `[crsid]` to your own CRSID.

Note that you will need to do this for the scripts `decode.nlu.mwoz21.slurm.wilkes2` and `train.nlu.mwoz21.slurm.wilkes2` , as well.

1.4.2 SLURM Job Submission - Single Jobs

A toy job submission script is provided in `$BDIR/slurm.example.1job` . Take a look at this script and edit it, as above. You can submit this script using the `sbatch` command as follows (you will get a unique job id):

[2] `https://docs.hpc.cam.ac.uk/hpc/user-guide/batch.html`

```
$ sbatch slurm.example.1job
Submitted batch job 35354144
```

You can use the the `squeue` command to monitor the job while its queued :

```
$ squeue -u $USER
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
35354144 skylake,c TryMe wjb31 PD 0:00 1 (None)
```

and while its running :

```
$ squeue -u $USER
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
35354144 skylake TryMe wjb31 R 0:36 1 cpu-e-188
```

The script simply waits for a short period and writes to a file `log.slurm.example.35354144` :

```
$ cat log.slurm.example.35354144
Hi. I'm an example SLURM job.
I'm running on cpu-e-351.
I've started at Mon 27 Dec 14:12:18 GMT 2021
And now I'm finished at Mon 27 Dec 14:15:18 GMT 2021
```

The SLURM submission scripts provided for you for this Practical Exercise have default settings that should send jobs to the appropriate queue with appropriate resource requests. The example here requests that a single job be submitted to the skylake and cclake CPU clusters.

Note also the **maximum time limit** set by the `#SBATCH --time=00:05:00` directive: jobs that run for longer than their requested time slot are killed automatically.

1.4.3 SLURM Job Submission - Job Arrays

One of the advantages of a job scheduling system is that multiple jobs can be submitted to the cluster [3]. See the simple example script `$BDIR/slurm.example.2jobs` that submits an array of 2 jobs to the CPU clusters; the option

```
#SBATCH --array=0-1
```

asks for an array of 2 jobs, with array indices 0 and 1.

After editing the project account as described above, the jobs are submitted using the `sbatch` command:

```
$ sbatch slurm.example.2jobs
```

Their progress can be tracked while queueing :

```
$ squeue -u $USER
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
35355348_[0-1] skylake,c TryMe2 wjb31 PD 0:00 1 (None)
```

and while running :

```
$ squeue -u $USER JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
35355348_0 skylake TryMe2 wjb31 R 0:01 1 cpu-e-1076
```

[3] https://docs.hpc.cam.ac.uk/hpc/user-guide/batch.html#array-jobs

```
35355348_1 skylake TryMe2 wjb31 R 0:01 1 cpu-e-1076
```

The outputs are written to separate log files - see the `$SLURM_ARRAY_TASK_ID` environment variable in the script `slurm.example.2jobs` :

```
$ head log.slurm.example.35355350.0 log.slurm.example.35355348.1
==> log.slurm.example.35355350.0 <==
Hi. I'm an example SLURM job JOB0.
I'm running on cpu-e-776.
I've started at Mon 27 Dec 14:16:37 GMT 2021
And now I'm finished at Mon 27 Dec 14:19:37 GMT 2021

==> log.slurm.example.35355348.1 <==
Hi. I'm an example SLURM job JOB1.
I'm running on cpu-e-351.
I've started at Mon 27 Dec 14:16:36 GMT 2021
And now I'm finished at Mon 27 Dec 14:19:36 GMT 2021
```

2 Introduction to MultiWOZ

Multi-Domain Wizard-of-Oz dataset (MultiWOZ) is a fully-labeled collection of human-human written conversations spanning over multiple domains and topics. The description here is drawn from the corpus GitHub page [4]. At the time of its creation, MultiWOZ was one of the largest annotated task-oriented corpora. There are now larger corpora, with different annotation schemes, but the MultiWOZ collections are still widely used in the literature for reporting progress in dialogue belief state tracking.

MultiWOZ has gone through several iterations of 'cleaning' in which corrections to dialogue annotations were made. This practical has been developed on MultiWOZ version 2.0 and 2.1. To limit the scope of the project, **you will be asked to report results only on MultiWOZ version 2.1**.

2.1 MultiWOZ and this Practical

The following sections are an introduction to MultiWOZ. Note that Sections 2.1 through 2.5 are provided as background; Section 2.6 introduces the MultiWOZ annotations in the form they will be used in this Practical.

The distributed versions of the MultiWOZ annotations have been adapted for use in this practical. You can inspect the original data files to get a sense of the conversations and the annotation scheme. The dialogue `MUL0003` can be viewed in its original annotated form in files available from the MultiWOZ GitHub page [5].

Note: Like any large collection of annotated natural language, MultiWOZ does have annotation errors and you will certainly find some if you look even casually. Do not be alarmed.

2.2 Templates for Dialogues

Dialogues are captured as interactions between 'Users' and 'Wizards'. The Users are assigned objectives to achieve, such as arranging for hotel, taxi, and restaurant bookings. The Wizards assist Users by providing information about relevant options , asking clarifying questions of the User, and then confirming bookings once the User's intent is sufficiently clear.

Each dialogue between a User and a Wizard is prompted by a *template* that sets out the goals the User should aim to accomplish, along with some guidance as to how the dialogue should proceed. In this way the goals are known for each dialogue, by construction.

Templates are created by a random sampling from the *dialogue ontology* (see Figure 4 - discussed later). The purpose of this random prompting is to create a collection of conversations covering a wide range of domains and goals. Figure 1 shows an example of the random sampling from the ontology that was used in the collection of dialogue `MUL0003` in MultiWoz 2.1.

[4] `https://github.com/budzianowski/multiwoz`

[5] Download `https://github.com/budzianowski/multiwoz/blob/master/data/MultiWOZ_2.1.zip` and search for `MUL0003` in the file `data.json` - be aware that the file is very big.

```
 ...
   "MUL0003.json": {
       "goal": {
           "taxi": {},
           "police": {},
           "hospital": {},
           "hotel": {
               "info": {
                   "pricerange": "cheap",
                   "internet": "yes",
                   "type": "guesthouse",
                   "parking": "yes"
               },
               "book": {
                   "people": "6",
                   "day": "sunday",
                   "invalid": false,
                   "stay": "4"
               },
           },
           "attraction": {},
           "train": {},
           "message": [
               "You are planning your trip in Cambridge",
               "You are looking for a place to stay. The hotel should include free wifi and should be in the type of guesthouse",
               "The hotel should be in the cheap price range and should include free parking",
               "Once you find the hotel you want to book it for 6 people and 4 nights  starting from sunday",
               "Make sure you get the reference number",
               "You are also looking for a restaurant. The restaurant should be in the same price range as the hotel and should be in the centre",
               "The restaurant should serve italian food",
               "Once you find the restaurant you want to book a table for the same group of people at  18:45 on the same day",
               "Make sure you get the reference number"
           ],
           "restaurant": {
               "info": {
                   "food": "italian",
                   "pricerange": "cheap",
                   "area": "centre"
               },
               "book": {
                   "people": "6",
                   "day": "sunday",
                   "invalid": false,
                   "time": "18:45"
               },}},
...
```

Figure 1: Dialogues Prompts are Produced by Random Sampling from the Dialogue Ontology. Sample for Dialogue `MUL0003` from MultiWoz 2.1 `data.json`. User requirements are given in two domains: Hotels (a cheap guesthouses with internet and parking, for 6 people, 4 nights, from Sunday) and Restaurants (a cheap Italian restaurant in the centre, for 6 people at 6PM Sunday).

2.3 Sample Dialogues

Figure 2 presents the dialogue that resulted from the template derived from Figure 1. The dialogue was successful, in that the System was able to satisfy all the Users' requests, and the User was able to satisfy all the given objectives.

System	User
	i am looking for a place to stay . it needs to be a guesthouse and include free wifi .
there are 23 hotel -s that meet your needs . would you like to narrow your search by area and and or price range ?	i would like for it to be cheap and include free parking .
there are 9 guesthouse hotel -s in various area -s . what part of town are you hoping for ?	nothing in particular . i just need it booked for 6 people for a total of 4 nights starting from sunday . i would also like the reference number , please .
sure ! we will book you in a hotel with 4 stars . the reservation will be for 6 people for 4 nights . how many rooms would you like to reserve ?	well , for 6 people i suppose i will be needing 3 rooms , for 4 nights , starting sunday .
you are booked at the alexander bed and breakfast , 517a coldham lane , for 6 people for 4 nights tarting sunday . your reference number is jxvkz7kv .	thank you ! i am also looking for a restaurant . ideally an italian place in the same price range in the centre .
i found ask restaurant would you like to book that ?	is the name of the restaurant ask restaurant ?
yes , the restaurant is called ask restaurant . would you like to book a table there ?	yes . book it for the same group of people on the same day at 18:45 . can i get the reference number please ?
great ! your booking for 6 people at the restaurant ask on sunday at 18:45 was successful ! your reference number is y2y8qyby .	i am all set . thank you .

Figure 2: Transcript of Dialogue `MUL0003` from MultiWoz 2.1 `data.json`.

Points to note about the transcription:

- The dialogue text is tokenized: the text is lower cased, and spaces are added before punctuation.
- The plural forms of some key words such as `hotels` are marked as `hotel -s`.
- The dialogue is 'user initiated' in that the User starts and ends the dialogue.

2.4 Dialogue Act Annotation

MultiWOZ dialogues are annotated with ‘dialogue acts’. Dialogue acts take the form

`<domain>-<type>:[<slot>,<value>].`

In the annotation of dialogue `MUL0003` (see Figure 3), the dialogue act annotation

“dialog_act”: [“Hotel-Inform”: [“Type”, “guesthouse”], [“Internet”, “yes”]]

indicates that the domain is “Hotel”, the type of dialogue act is “Inform”, the slots are “Type” and “Internet”, and their values are “guesthouse” and “yes”. The full ontology of domains, acts, and slots is in Figure 4.

In the example of dialogue `MUL0003`, the User starts off by informing the System that they are looking for a guesthouse with free Wi-Fi. The annotation accompanying this utterance is shown in Figure 3, which indicates both the intent as well as the evidence (the span refers to position 14 in the tokenized utterance (see Figure 2)). The System responds that the User has a choice of 23 hotels and asks for more information.

```
...
"log": [
  {
      "text": "I'm looking for a place to stay. It needs to be a guesthouse and include free wifi.",
      "metadata": {},
      "dialog_act": {
          "Hotel-Inform": [
              [
                  "Type",
                  "guesthouse"
              ],
              [
                  "Internet",
                  "yes"
              ] ]},
      "span_info": [
          [
              "Hotel-Inform",
              "Type",
              "guesthouse",
              14,
              14
          ] ] },
  {
      "text": "There are 23 hotels that meet your needs. Would you like to narrow your search by area and/or price range?",
      "metadata": {
      ...
      "dialog_act": {
      ...
           "Hotel-Inform": [
              [
                  "Choice",
                  "23"
              ],
              [
                  "Type",
                  "hotels"
              ]  ] },
      "span_info": [
          [
              "Hotel-Inform",
              "Choice",
              "23",
              2,
              2
          ],
          [
              "Hotel-Inform",
              "Type",
              "hotels",
              3,
              3
          ] ] },
    ...
```

Figure 3: Excerpts from the Annotation of Dialogue `MUL0003`. Note that this example is illustrative; there are some small differences between these ‘dialogue_acts’ and the DST presented in subsequent sections.

Table 2: Full ontology for all domains in our data-set. The upper script indicates which domains it belongs to. *: universal, 1: restaurant, 2: hotel, 3: attraction, 4: taxi, 5: train, 6: hospital, 7: police.

act type	inform* / request* / select^{123} / recommend/^{123} / not found^{123} request booking info^{123} / offer booking^{1235} / inform booked^{1235} / decline booking^{1235} welcome* /greet* / bye* / reqmore*
slots	address* / postcode* / phone* / name^{1234} / no of choices^{1235} / area^{123} / pricerange^{123} / type^{123} / internet^{2} / parking^{2} / stars^{2} / open hours^{3} / departure^{45} destination^{45} / leave after^{45} / arrive by^{45} / no of people^{1235} / reference no.^{1235} / trainID^{5} / ticket price^{5} / travel time^{5} / department^{7} / day^{1235} / no of days^{123}

Figure 4: MultiWOZ ontology: dialogue acts and slots, with their permitted domains (from Budzianowski et al. "MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling". EMNLP'18.)

2.5 Linearized Dialogue Belief States

Rather than maintain the belief state in the hierarchical structured form as shown in Figure 3, an alternative is to **linearize** the annotation. Linearisation is a commonly used technique in which structured data, such as graphs and trees, is represented as strings. Linearisation may not be perfect, in that it can lose information. In particular, nested hierarchical relationships can 'flattened', and the mapping from structures to strings can be many-to-one (i.e. not invertible). Despite these ambiguities, linearisation makes it straightforward to apply sequence-to-sequence models to structured data.

This practical uses linearised belief states at both the turn level and at the dialogue level. Linearisation is done fairly straightforwardly. Dialogue acts of the form

```
<domain>-<type>:[<slot1>,<value1>,<slot2>,<value2>,...<slotN,valueN>]
```

are linearised into a list of domain / slot / value items, joined via a separator token :

```
domain slot1 value1 <SEP> domain slot2 value2 <SEP> ...  domain slotN valueN
```

As an example, the dialogue act representation

```
["Hotel-Inform":["Type","guest house"],["Internet","yes"]]
```

becomes

```
hotel guest house <SEP> hotel internet yes
```

Note that the 'act type' (e.g. "Inform") is dropped from the linearisation, as is the span information.

2.6 Linearized Dialogue Annotations for this Practical

```
...
"MUL0003": {
       "turn-0": {
            "system_utterance": "",
            "user_utterance": "i am looking for a place to stay . it needs to be a guesthouse and include free wifi .",
            "dst_belief_state": "hotel internet yes <SEP> hotel type guest house",
            "nlu_belief_state": "hotel internet yes <SEP> hotel type guest house",
        },
        "turn-1": {
            "system_utterance": "there are 23 hotel -s that meet your needs . would you like to narrow your search by area and
            and or price range ?",
            "user_utterance": "i would like for it to be cheap and include free parking .",
            "dst_belief_state": "hotel internet yes <SEP> hotel parking yes <SEP> hotel pricerange cheap <SEP> hotel type guest house",
            "nlu_belief_state": "hotel parking yes <SEP> hotel pricerange cheap",
        },
        "turn-2": {
            "system_utterance": "there are 9 guesthouse hotel -s in various area -s . what part of town are you hoping for ?",
            "user_utterance": "nothing in particular . i just need it booked for 6 people for a total of 4 nights starting from sunday .
            i would also like the reference number , please .",
            "dst_belief_state": "hotel day sunday <SEP> hotel internet yes <SEP> hotel parking yes <SEP> hotel people 6 <SEP>
            hotel pricerange cheap <SEP> hotel stay 4 <SEP> hotel type guest house",
            "nlu_belief_state": "hotel day sunday <SEP> hotel people 6 <SEP> hotel stay 4",
        },
...
```

Figure 5: Linearized Belief State Annotations
(`$BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json`).

The first item in `$BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json` is the running example of dialogue `MUL0003` showing the linearised representation of the dialogue belief state. Note that the annotation is maintained at the turn level (`nlu_belief_state`) and at the dialogue level (`dst_belief_state`). The dialogue level belief state is cumulative: it describes the dialogue up to and including the most recent User utterance. The turn level belief state is incremental: it captures what is added to the dialogue by the most recent turn.

Consider the annotations in Figure 5:

- At `turn-0`, the turn level and the dialogue level belief states are identical (both start as blank slates):

  ```
  hotel internet yes <SEP> hotel type guest house
  ```

- At `turn-1`, the User refines their initial request to ask for a cheap hotel with free parking. The turn level belief state reflects this:

  ```
  hotel parking yes <SEP> hotel pricerange cheap
  ```

  The dialogue belief state is cumulative: it adds the information from the current term to what was learned in the previous turn:

  ```
  hotel internet yes <SEP> hotel type guest house <SEP> hotel parking
  yes <SEP> hotel pricerange cheap
  ```

The files `$BDIR/data_preparation/data/multiwoz21/refs/dev/dev_v2.1.json` and `$BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json` contain the reference annotations for the MultiWOZ 2.1 development and test sets. These files will be used as references in scoring the performance of automatically generated belief state annotations at both the turn level and dialogue level.

2.6.1 Allowing for Corrections

In the previous example, the dialogue belief state is found as an accumulation of the turn level belief states. The exception to this is when a User offers a correction. Consider this exchange in dialogue `MUL0071`:

turn-5	the cityroomz hotel is moderately priced , and has free internet and parking . would you like to make a reservation now ?	that sounds nice , yes please book a room for 3 guests staying for 5 nights starting on sunday .
turn-6	i am sorry , those particular dates are filled , if you would like i could try to make the trip shorter or maybe change days ?	how about just for the 1 night ?

The dialogue level belief state at turn 5 is

> `hotel area centre <SEP> hotel day sunday <SEP> hotel name cityroomz <SEP> hotel people 3 <SEP> hotel pricerange moderate <SEP> hotel stars 0 <SEP>` **hotel stay 5** `<SEP> restaurant area centre <SEP> restaurant food dontcare <SEP> restaurant pricerange moderate`

However, at turn-6 the User updates their request (because a room is not available) and the dialogue level belief state changes to

> `hotel area centre <SEP> hotel day sunday <SEP> hotel name cityroomz <SEP> hotel people 3 <SEP> hotel pricerange moderate <SEP> hotel stars 0 <SEP>` **hotel stay 1** `<SEP> restaurant area centre <SEP> restaurant food dontcare <SEP> restaurant pricerange moderate`

Note that the utterance level belief state at turn 6 is `‘hotel stay 1’` Simply concatenating this to the dialogue belief state from turn 5 would lead to a conflict: the dialogue state would contain both `‘hotel stay 1’` and `‘hotel stay 5’` . To avoid this, the turn level annotation allows for a correction:

```
"turn-6": {
 "system_utterance": "i am sorry , those particular dates are filled ,
 if you would like i could try to make the trip shorter or maybe change days ?",
 "user_utterance": "how about just for the 1 night ?",
 "dst_belief_state": "hotel area centre <SEP> hotel day sunday
 <SEP> hotel name cityroomz <SEP> hotel people 3
 <SEP> hotel pricerange moderate <SEP> hotel stars 0 <SEP> hotel stay 1
 <SEP> restaurant area centre <SEP> restaurant food dontcare
 <SEP> restaurant pricerange moderate",
 "nlu_belief_state": "hotel stay 1",
 "nlu_correction": "hotel stay 5"
},
```

The entry `"nlu_correction":"hotel stay 5"` indicates that this dialogue act should be removed from the belief state at this turn. The field `nlu_correction` is used relatively infrequently in the dialogue annotation. It indicates that the User has corrected a particular `domain/slot/value` entry, and that entry should be dropped from the dialogue belief state. In effect, the User asks for previous information to be updated or removed.

3 Natural Language Understanding and GPT-2

In the context of task-oriented dialogue systems, Natural Language Understanding (NLU) is the automatic annotation of dialogue acts within individual utterances or dialogue turns. NLU is performed and evaluated at the turn level within each dialogue. At each turn, the NLU system aims to predict the turn level belief state. Performance is measured against turn level reference annotations. For this Practical we annotate turns with the domains, slots and values mentioned, but omit the intent from the dialogue act.

In its simplest form, the NLU system looks only at the current turn: the NLU system reads the System utterance and User response, and generates a turn level linearised belief state. This will be the form of NLU presented in this section of the Practical. However an NLU system could in principle use the entire dialogue history up to the current turn - you may wish to investigate this later.

This portion of the Practical exercise introduces Hugging Face GPT-2 for NLU. The presentation is at a fairly high level and focuses mainly on data formats and training and decoding configurations. Full source code for the NLU trainer, decoder, and scoring is provided. You are encouraged to investigate the code, but code changes shouldn't be necessary.

3.1 MWOZ 2.1 NLU Data Files

JSON files for the MultiWOZ 2.1 training, development, and test sets are provided:

```
$BDIR/data_preparation/data/multiwoz21/processed/train/version_1/data.json
$BDIR/data_preparation/data/multiwoz21/processed/dev/version_1/data.json
$BDIR/data_preparation/data/multiwoz21/processed/test/version_1/data.json
```

NLU scoring uses annotated references:

```
$BDIR/data_preparation/data/multiwoz21/refs/dev/dev_v2.1.json
$BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json
```

	train	dev	test
dialogues	7888	1000	999
turns	54971	7374	7368

Table 1: MultiWOZ training, development, and test set sizes for this Practical.

3.2 NLU Training and Development Sets

The first step in preparing for fine-tuning GPT-2 models for NLU is to put the training and development data into the appropriate format. The trainer expects to find the following fields [6] in the training and development JSON files:

- `example_id`: the dialogue and turn identifier for each training instance
- `dst_input`: Dialogue State Tracker input. For the simple form of NLU that reads only the current turn, the field `dst_input` is the concatenation of the System utterance and the User utterance. The two special symbols `<SYS>, <USR>` are delimiters that indicate the start of the System and User utterances, respectively. An example of this is:
  `"<SYS> i show many restaurant -s that serve indian food in that price range . what area would you like to travel to ?  <USR> i am looking for an expensive indian restaurant in the area of centre .  "`
- `belief_state`: the linearised belief state is the target used in supervised training. For NLU, this is turn level belief state in linearised form.

See `$BDIR/data_preparation/data/multiwoz21/processed/train/version_1/data.json` as an example:

```
....
{
"example_id": "MUL0001-0",
"dst_input": "<USR> i would really like to take my client out to a nice restaurant
that serves indian food . ",
"belief_state": "restaurant food indian <SEP> restaurant pricerange expensive"
},
{
 "example_id": "MUL0001-1",
 "dst_input": "<SYS> i show many restaurant -s that serve indian food in that price range
 what area would you like to travel to ? <USR> i am looking for an expensive
 indian restaurant in the area of centre . ",
 "belief_state": "restaurant area centre"
},
 {
"example_id": "MUL0001-2",
 "dst_input": "<SYS> might i recommend saffron brasserie ? that is an expensive indian
 restaurant in the center of town . i can book a table for you , if you like .
 <USR> sure thing , please book for 6 people at 19:30 on saturday . ",
"belief_state": "restaurant day saturday <SEP> restaurant name saffron brasserie
<SEP> restaurant people 6 <SEP> restaurant time 19:30"
},
....
```

[6]The field `nlu_correction` is also in the development and test files, but not used for NLU training, decoding, or scoring.

3.3 GPT-2 NLU Fine-Tuning

This section introduces the Hugging Face GPT-2 fine-tuning procedure.

An initial set of fine-tuned Hugging Face GPT-2 models are provided in the directory
`$BDIR/checkpoints/nlu_flat_start/model.60000/`.

Fine tuning for MultiWoz 2.1 was performed from scratch (that is, from the pre-trained GPT-2 model distributed by Hugging Face) using the following command:

```
python $BDIR/train-gpt2-dst.py  \
 --train_data $BDIR/data_preparation/data/multiwoz21/processed/train/version_1/data.json \
 --dev_data $BDIR/data_preparation/data/multiwoz21/processed/dev/version_1/data.json \
 --args $BDIR/train_arguments.nlu.mwoz21.yaml
```

with `train.experiment_name` set to `'nlu_continued_training'` in
`$BDIR/train_arguments.nlu.mwoz21.yaml`.

Model parameters are updated by the Adam optimizer [7], with the following options:

- Training and development sets are read from
  `$BDIR/data_preparation/data/multiwoz21/processed/train/version_1/data.json`
  `$BDIR/data_preparation/data/multiwoz21/processed/dev/version_1/data.json`
- Training proceeds with a `train_batch_size` of 1 (i.e. 1 dialogue turn).
- Gradients are accumulated over 8 batches (i.e. over 8 dialogue turns) between parameter updates.
- The epoch (i.e. one pass through the training set) covers 54971 batches (of 1 dialogue turn each).
- With an `eval_interval` of 2000, a forward pass is computed over the development data every 2,000 batches. After the first epoch, the `eval_interval` is multiplied by 10.

  After each evaluation, models are saved to the checkpoint directories
  `$BDIR/checkpoints/nlu_flat_start/model.step.$count`, where `$count` is the batch counter.

  Note that in your first experiments, the `experiment_name` will be `nlu_continued_training`, and so your models will be saved to directories `checkpoints/nlu_continued_training/model.$count`.
- Training set loss is reported every epoch.

The complete training log is available in `$BDIR/checkpoints/nlu_flat_start/logs/train-gpt2-dst.log`. Training and development set loss (i.e. negative log-likelihood) are provided in Figure 6.

[7] Kingma, Ba. Adam: A Method for Stochastic Optimization `https://arxiv.org/abs/1412.6980v1`

Batch	Train Loss	Dev Loss
2000		0.914
4000		0.421
6000		0.437
8000		0.385
10000		0.340
12000		0.305
14000		0.279
16000		0.271
18000		0.367
20000		0.404
22000		0.345
24000		0.258
26000		0.282
28000		0.243
30000		0.235
32000		0.270
34000		0.242
36000		0.253
38000		0.229
40000		0.264
42000		0.280
44000		0.243
46000		0.236
48000		0.221
50000		0.227
52000		0.220
54000		0.257
54971	0.414	0.218
60000		0.209
80000		0.222
100000		0.212
109942	0.226	0.216
120000		**?**
140000		**?**
160000		**?**
164913	**?**	**?**
180000		**?**
200000		**?**
219884	**?**	**?**

Figure 6: MultiWOZ NLU Flat Start Training Loss. Training loss is reported every epoch (54,971 turns); development loss is reported every 2,000 turns in the first epoch, and every 20,000 turns subsequently. Training is ‘from scratch’, i.e. from the version of GPT-2 distributed by HuggingFace. The initial models are pretrained from large text collections, but not yet trained for MultiWOZ.

3.4 Exercise GPT2DST.1: Fine Tuning and Continued Training for NLU

Note Exercise GPT2DST.1 is straightforward and based on Section 3.3. You will: run a SLURM training job that executes fine-tuning either as continued training (i.e. from the already partially trained model provided) or from a flat start (i.e. 'from scratch'); verify previous training behaviour; and produce models that you will use in later sections.

Replicate the results of Figure 6 of Section 3.3, reporting development set loss every 20,000 batches, up to 200,000 batches, and training set loss every epoch.

Copy the script `$BDIR/train.nlu.mwoz21.slurm.wilkes2` to your working directory and edit it as necessary; remember to modify the `MLMI-[crsid]-SL2-GPU` flag for your own account.

You can train models either:

- **(a)** from models provided at batch 60,000 : `$BDIR/checkpoints/nlu_flat_start/model.60000/`.
- **(b)** from scratch, i.e. a flat-start fine tuning from GPT-2 as distributed by HuggingFace.

Some points to note:

- Option **(b)** should more closely approximate the results of Section 3.3, but may take longer to run than option **(a)**.
- The SLURM script `train.nlu.mwoz21.slurm.wilkes2` is configured for option **(a)**. If the `CHECKPOINT` environment variable is set, it is passed to the `train-gpt2-dst.py` script via the `--restore` option. The training script loads the checkpointed models and continues training.
- For option **(b)**, comment out the line with `CHECKPOINT=` in the SLURM script. The trainer option `--restore` is then not set so that the trainer loads HuggingFace GPT-2 models and continues training 'from scratch'.
- You can change the random seed if you wish, or keep the pre-set value.

In your report you should include your version of Figure 6, along with commentary about any differences in your results. You should also include the training command that you used.

3.5 GPT-2 NLU Decoding

This section explains decoding using Hugging Face GPT-2 models that have been fine-tuned on MultiWOZ 2.1 NLU training sets.

The aim of NLU decoding is to generate a turn level belief state annotation for each dialogue turn in the input file. The NLU decoder takes as its input the concatenation of the System and User utterances at each turn. As an example of the NLU input data format, the file `$BDIR/data_preparation/data/multiwoz21/processed/test/version_1/data.json` has the following entry [8]:

```
...
{
"example_id": "MUL0003-2",
"dst_input": "<SYS> there are 9 guesthouse hotel -s in various area -s .  what
part of town are you hoping for ? <USR> nothing in particular .  i just need it
booked for 6 people for a total of 4 nights starting from sunday .  i would also
like the reference number , please . "
},
...
```

The GPU2DST decoder script is `$BDIR/decode.nlu.mwoz21.slurm.wilkes2` , and here is an example of running the decoder over the data `$BDIR/data_preparation/data/multiwoz21/processed/test/version_1/data.json` using the fine-tuned model `$BDIR/checkpoints/nlu_flat_start/model.60000/` :

```
python $BDIR/gpt2-dst.py --hyp_dir tmp/ \
 --test_data $BDIR/data_preparation/data/multiwoz21/processed/dev/version_1/data.json \
 --args $BDIR/decode_arguments.nlu.mwoz21.yaml \
 --checkpoint $BDIR/checkpoints/nlu_flat_start/model.60000/ -vv --data_size 2
```

The annotated output is written to `tmp/nlu_continued_training/model.60000/belief_states.json` [9]:

```
{
  "example_id": "MUL0012-0",
  "predicted_belief_state": "<USR> i need information on a hotel that
   include -s free parking please. <bos>hotel parking yes<eos>",
   "utterance": " i need information on a hotel that include -s free parking please . "
    },
    {
  "example_id": "MUL0012-1",
  "predicted_belief_state": "<SYS> there are 29 hotel -s that meet
   your needs. would you like to narrow your search by area and and
   or price range? <USR> i also need free wifi, and it needs to be a hotel, not a
   guesthouse or anything like that. <bos>hotel internet yes <SEP>
   hotel type hotel<eos>",
   "utterance": " i also need free wifi , and it needs to be a hotel , not a
   guesthouse or anything like that . "
}
```

For each turn the decoder writes its output to the field `predicted_belief_state`. The output includes

[8] Note that the test JSON files do not contain reference annotations.

[9] You can run this script at your command line to see the output; use the argument `--data_size 2` to process only two turns.

the dialogue turn itself, copied directly from the input field `dst_input`. The predicted linearized belief state follows the input, appearing between the `<BOS>` and `<EOS>` tokens, as in the following:

```
dst_input <BOS> domain1 slot1 value1 <SEP> domain1 slot2 value2 ... <EOS>
```

3.6 Scoring GPT2DST NLU Hypotheses

The script `$BDIR/multiwoz_dst_score.py` scores the NLU decoder output against the reference turn level belief states. Two scores are produced: DST Average Slot Accuracy, and DST Average Joint Accuracy. Both are of interest, but for simplicity this Practical will focus on the DST Average Joint Accuracy.

The scoring script is run as:

```
$ python $BDIR/multiwoz_dst_score.py --field nlu_belief_state \
 --dst_ref $BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json \
 --h $BDIR/hyps/test/nlu_flat_start/model.60000/belief_states.json
References: .../data_preparation/data/multiwoz21/refs/test/test_v2.1.json
reference field: nlu_belief_state
Hypotheses: .../hyps/test/nlu_flat_start/model.60000/belief_states.json
DST Average Slot Accuracy: 98.40 %. DST Average Joint Accuracy: 67.13 %
```

The DST Average Joint accuracy is 67.1% on the MWOZ 2.1 test set with the models `nlu_flat_start/model.60000/`.

N.B. The argument `--field nlu_belief_state` tells the scoring programme to compute accuracy with respect to the turn level belief state in the reference file. Recall that the reference files contain annotations at the dialogue level (i.e. cumulative) and at the turn level. **In measuring NLU performance it is important to score relative to the turn level reference belief state.**

3.7 Exercise GPT2DST.2: NLU Decoding on MultiWOZ 2.1

Using the models produced in Exercise GPT2DST.1, either by continued training or from scratch, decode the MultiWOZ 2.1 test set and measure Joint Slot Accuracy to create your own version of Table 2, with missing values filled in.

	Batch	5000	52000	54000	60000	80000	100000	120000	140000	160000	180000	200000
flat start	dev	68.66	70.56	70.90	69.24	72.24	72.09	72.19	72.99	73.27	73.83	74.48
	test	66.30	67.44	69.46	67.13	?	?	?	?	?	?	?
continued training	dev	68.66	70.56	70.90	69.24	71.94	70.22	71.74	72.20	73.27	73.07	74.48
	test	66.30	67.44	69.46	67.13	?	?	?	?	?	?	?

Table 2: NLU Joint Accuracy for MultiWOZ 2.0 and 2.1 found via decoding with GPT2NLU. 'Batch' refers to checkpointed models: for example, at Batch 60000 decoding of MWOZ 2.1 is done with the models `nlu_flat_start/model.60000/`.

4 Dialogue State Tracking

This portion of the practical extends the discussion from turn level Natural Language Understanding to Dialogue State Tracking. The GPT-2 fine-tuning approach introduced for NLU in the previous section will serve as the basis for DST.

4.1 Cheap and Cheerful Dialogue State Tracking with GPT2NLU

'Cheap and Cheerful: (informal) something cheap and cheerful; does not cost a lot but is attractive and pleasant'

The turn level belief state annotations produced by GPT2NLU can be used directly for Dialogue State Tracking. In the simplest case, we can simply measure the DST Joint Accuracy of the turn level annotations against the dialogue level references. The expectation is that this accuracy will be low, since the hypothesized belief state is based only on the current turn and will not carry over any information from earlier turns in the dialogue. As an improvement over this, a Cheap and Cheerful Dialogue State Tracker can be implemented simply by concatenating NLU belief state hypotheses produced at the turn level.

Suppose GPT2NLU generates the following turn level belief states for the first three turns:

```
turn-0: <BOS> hotel internet yes <SEP> hotel type guest house <EOS>
turn-1: <BOS> hotel parking yes <SEP> hotel pricerange cheap <EOS>
turn-2: <BOS> hotel day sunday <SEP> hotel people 6 <SEP> hotel stay 4 <EOS>
```

The Cheap-and-Cheerful DST (CC-DST) system simply concatenates turn level belief states to generate a dialogue level belief state:

```
turn-0: <BOS> hotel internet yes <SEP> hotel type guest house <EOS>
turn-1: <BOS> hotel internet yes <SEP> hotel type guest house
  <SEP> hotel parking yes <SEP> hotel pricerange cheap <EOS>
turn-2: <BOS> hotel internet yes <SEP> hotel type guest house
  <SEP> hotel parking yes <SEP> hotel pricerange cheap
  <SEP> hotel day sunday <SEP> hotel people 6 <SEP> hotel stay 4 <EOS>
```

The CC-DST has obvious limitations. Its overall performance is limited by the quality of independently generated turn level annotations, and at each turn the previous dialogue history is ignored. On the positive side, CC-DST is easy to generate if an NLU system is available.

The script `$BDIR/cc-dst.py` is a simple implementation of this concatenation. It takes the turn level annotation generated by GPT2NLU and produce a dialogue level annotation [10]:

```
$ python $BDIR/cc-dst.py \
 --nlu $BDIR/hyps/test/nlu_flat_start/model.60000/belief_states.json \
 --dst hyps/test/cc_dst/nlu_continued_training/model.60000/belief_states.json
```

Dialogue level annotations are written to the file
`hyps/test/cc_dst/nlu_continued_training/model.60000//belief_states.json`.
These are a concatenation of the turn-level NLU belief states taken from the input file,
`$BDIR/hyps/test/nlu_flat_start/model.60000/belief_states.json`.

[10]Create `hyps/test/cc_dst/nlu_continued_training/model.60000/` before running this command.

The DST annotation accuracy can be computed for both the turn level and dialogue level annotations. This is an indication of the effectiveness of CC-DST as a first attempt at dialogue state tracking.

```
$ python $BDIR/multiwoz_dst_score.py --field dst_belief_state  \
 --dst_ref $BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json  \
 --h $BDIR/hyps/test/nlu_flat_start/model.60000/belief_states.json
References: .../data_preparation/data/multiwoz21/refs/test/test_v2.1.json
reference field: dst_belief_state
Hypotheses: .../hyps/test/nlu_flat_start/model.60000/belief_states.json
DST Average Slot Accuracy: 83.63 %. DST Average Joint Accuracy: 12.87 %
```

```
$ python $BDIR/multiwoz_dst_score.py --field dst_belief_state  \
 --dst_ref $BDIR/data_preparation/data/multiwoz21/refs/test/test_v2.1.json  \
 --h hyps/test/cc_dst/nlu_continued_training/model.60000/belief_states.json
References: .../data_preparation/data/multiwoz21/refs/test/test_v2.1.json
reference field: dst_belief_state
Hypotheses: hyps/test/nlu_flat_start/cc-dst/model.6000/belief_states.json
DST Average Slot Accuracy: 94.16 %. DST Average Joint Accuracy: 31.79 %
```

These results show that CC-DST improves the belief state tracking at the dialogue level from an Average Joint Accuracy of 12.87% to 31.79%.

4.2 Exercise GPT2DST.3: Dialogue State Tracking with GPT2NLU

Using the GPT2NLU annotations from the earlier exercises, compute the dialogue level Average Joint Accuracy to generate your own version of Table 3, with missing entries completed. You can report results with NLU models trained either from a flat-start or via continued training, depending on your choice in Exercise GPT2DST.2.

	Batch	5000	52000	54000	60000	80000	100000	120000	140000	160000	180000	200000
flat start	dev	34.26	39.03	41.36	35.72	42.85	41.51	41.35	?	?	?	?
	test	31.46	34.22	37.76	31.79	?	?	?	?	?	?	?
continued training	dev	34.26	39.03	41.36	35.72	41.69	37.75	40.29	?	?	?	?
	test	31.46	34.22	37.76	31.79	?	?	?	?	?	?	?

Table 3: MultiWOZ 2.1 DST Joint Accuracy: GPT2NLU and Cheap and Cheerful DST.

4.3 Exercise GPT2DST.4: Build Your Own GPT-2 DST System

In this Exercise you will synthesise and expand on the material developed in this Practical. Your objectives in this exercise are to design, build, and evaluate a MultiWOZ 2.1 Dialogue State Tracker.

Here are a range of possibilities to explore, ranging from easy to very ambitious:

- DST with one of the pre-trained/fine-tuned NLU systems provided for you, without further refinement.
- DST with an NLU system refined via continued training on NLU training data.
- Use one of the above NLU systems, but with the complete dialogue history as its input.
- Cheap-and-Cheerful DST using the `cc-dst.py` script as provided with one of the above model sets.
- Improve the CC-DST system `cc-dst.py` and use it with one of the above NLU systems. If you improve the CC-DST system, explain your improvements and consider providing contrastive experiments that validate whether your improvements are effective.
- Train and evaluate a **GPT2DST** model that merges NLU and DST into a single system. This will require creating your own dialogue level training set from the turn level data provided. In preparing the dialogue level training data, consider what linearisation strategies would work best: for example, whether dialogue acts should appear in temporal order, or at random, or in some other way. In fine-tuning, consider whether to start from scratch (from GPT-2 as distributed by HuggingFace) or whether to continued training from one of the NLU models. If you train a GPT2DST system, consider assessing its performance as an NLU system relative to systems from Section 3.
- Either of the following can be attempted as **refinements** to GPT2DST:
  - Latency and Memory: Consider DST in which at each turn the full dialogue history is replaced by the hypothesized dialogue state. This leads to a more compact history, which should improve latency and memory use, but DST model training and decoding will have to change.
  - Sparsely Supervised Training: Turn-level dialogue state annotations as used thus far are expensive. An alternative is 'training is at the entire dialogue level, i.e. including only the dialogue state labels at the final turn without their intermediate states during the session, but with the previous dialogue turns included as history.' [11]. Will there be a loss in performance?

Your report should present a rationale for your design decisions, along with any additional experimental investigations you performed to support your decision. Include in your report DST Joint Accuracy for the MultiWOZ 2.1 development and test sets.

You must not use open source data and models beyond what is provided to you. Your report should also include a **model card** for the system you choose as your final system. Use your own judgement on what to include in the model card, but be concise. Note that you are using models for which model cards already exist, and so you can refer to these in presenting your own work:

- HuggingFace: `https://huggingface.co/gpt2`
- OpenAI: `https://github.com/openai/gpt-2/blob/master/model_card.md`

[11] Lin, Tseng, Byrne. Knowledge-Aware Graph-Enhanced GPT-2 for Dialogue State Tracking, EMNLP'21, `https://aclanthology.org/2021.emnlp-main.620/`

5 Report Organization

Please organise your report into sections as follows:

- Overview
- Exercise GPT2DST.1
- Exercise GPT2DST.2
- Exercise GPT2DST.3
- Exercise GPT2DST.4
  - Presentation of your system, along with figures and tables of results.
  - A comparison of your chosen system to UBAR (typical length: two paragraphs)
  - A Model Card for your system
- Summary and Conclusion

There is no word count limit, but marking will consider both precision and recall.
You should aim for a presentation that is clear, concise, and correct.

Contributors
Alex Coca, Lent 2021, 2022
Andy Tseng, Lent 2021, 2022
Bill Byrne, Lent 2021, 2022