Math cluster

The maximum run time limit is currently **220** hours (**9** days).

When the node is not busy, the job will start immediately. Otherwise, it may takes 0.5~2 days.

Common command:

- Check the status: qstat -u Augustin
- Submit job: qsub -l gpus=1 py.job
- Delete job: qdel job_id

```
#!/bin/bash
#$ -cwd
#$ -V                     Inherit environment
#$ -l h=node561           Specify the node ID
#$ -N testpython          Name
#$ -l mem_free=5G         Memory usage
#$ -o tf.out              Output file
#$ -j y                   Join,  write both error messages and output
#$ -l h_rt=780000         Estimated running time
#$ -m be
#$ -M xxx@campus.tu-berlin.de
source ~/.bashrc
--python xx.py
```

Math cluster

```
[augustin@cluster-i ~]$ qstat -u augustin
job-ID  prior   name       user         state submit/start at     queue
    slots ja-task-ID
-----------------------------------------------------------------------------------
-------------------
9685605 1.00728 testpython augustin     r     06/16/2018 07:33:07 long@node562
        1
9685607 1.00728 testpython augustin     r     06/16/2018 07:40:00 long@node561
        1
9685608 1.00724 testpython augustin     r     06/16/2018 08:14:13 long@node560
        1
9685610 1.00723 testpython augustin     r     06/17/2018 12:27:49 long@node560
        1
9685611 1.00723 testpython augustin     r     06/16/2018 08:25:38 long@node562
        1
9686502 1.00439 testpython augustin     r     06/18/2018 08:51:44 long@node563
        1
9690069 1.00308 testpython augustin     r     06/19/2018 07:05:33 long@node561
        1
```

Runtime profiling of one batch

```
2018-06-23 15:20:50,002 [INFO] log1:
===========Run training epoch==========

multi processes: Sat Jun 23 15:20:53 2018
multi processes: Sat Jun 23 15:20:53 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:54 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:55 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
multi processes: Sat Jun 23 15:20:56 2018
Numbe of batch: 1 Sat Jun 23 15:21:04 2018 --------------------------------------------
multi processes: Sat Jun 23 15:21:04 2018
multi processes: Sat Jun 23 15:21:04 2018
multi processes: Sat Jun 23 15:21:04 2018
```

Batch size: 40
Time length: 3000
Time of loading 40 scene instances: 3 s
Time of calling cudnn-lstm: 8s

Compare cv performance

```
df = pd.read_csv('seperate performance.csv',error_bad_lines=False)
result = df

for i in range(1,7):
    s = 'cv_' + str(i)
    t = df.sort_values([str(i)], ascending=False)['index']
    result[s] = t.values
result
```

	index	1	2	3	4	5	6	EPOCH	cv_1	cv_2	cv_3	cv_4	cv_5	cv_6
0	A1	0.802	0.810	0.811	0.811	0.793	0.804	20	B2	B2	B2	B2	B2	E9
1	B2	0.828	0.819	0.830	0.819	0.820	0.820	12	F11	F11	E9	F11	E9	B2
2	C7	0.500	0.500	0.500	0.500	0.500	0.500	2	A1	E9	A1	A1	F11	F11
3	D8	0.500	0.500	0.500	0.500	0.500	0.500	2	E9	A1	F11	E9	A1	A1
4	E9	0.794	0.813	0.813	0.808	0.818	0.826	20	I15	I15	I15	I15	I15	I15
5	F11	0.806	0.819	0.810	0.814	0.807	0.817	30	H14	H14	H14	H14	H14	H14
6	G13	0.729	0.733	0.724	0.731	0.722	0.745	20	G13	G13	G13	G13	G13	G13
7	H14	0.748	0.762	0.750	0.763	0.742	0.768	20	C7	C7	C7	C7	C7	C7
8	I15	0.775	0.785	0.773	0.775	0.761	0.796	20	D8	D8	D8	D8	D8	D8

Random search (LDNN)

- TODO: 3D heatmap.
- Question:

For example: x: dropout rate, y: model complexity, z(color): valid_acc. But there are other parameters really affect the valid_acc, such as learning rate or L2_norm. Same (x,y) could have different valid_acc depends on learning rate. How can we draw conclusion from such heatmap?

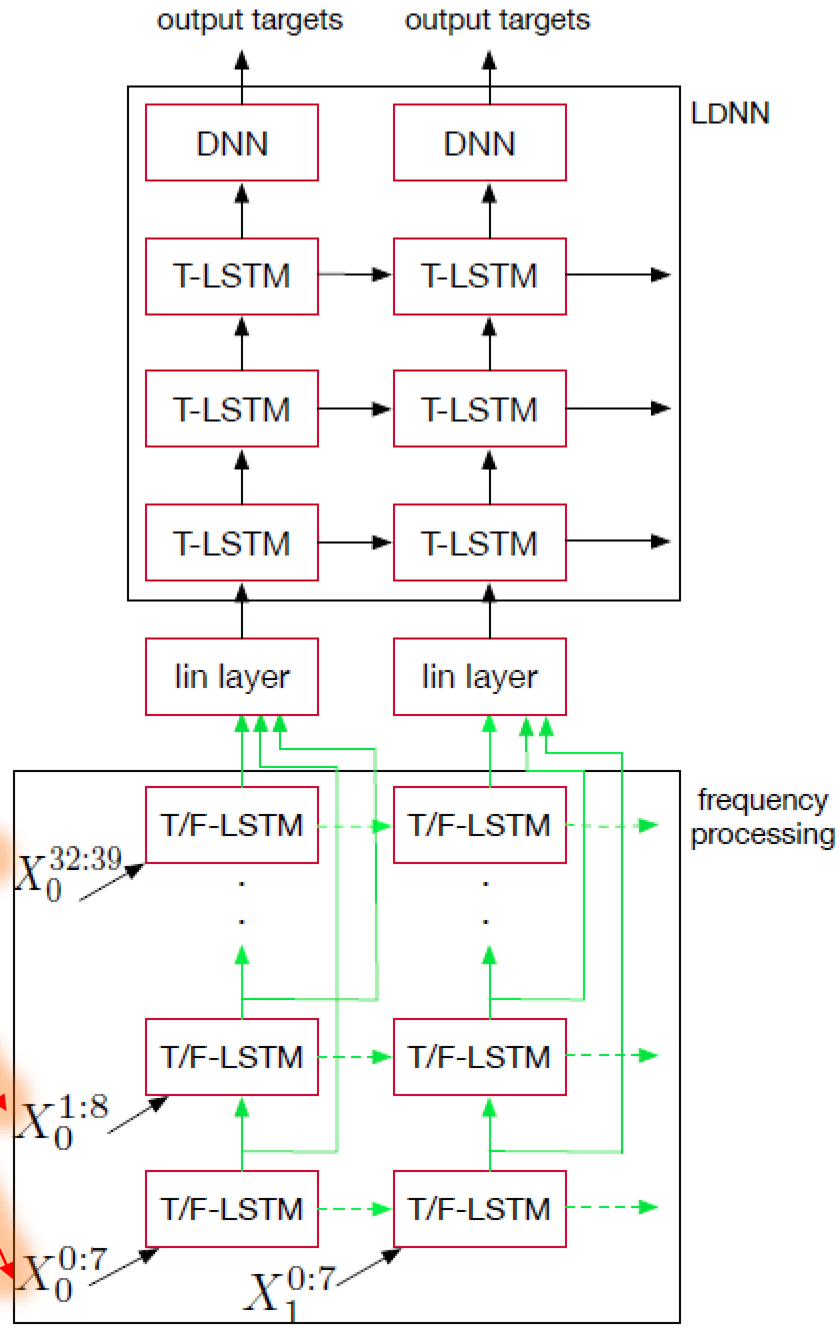
Frequency block Gird LDNN

- Reducing variations within a **small frequency region**
  - Reason: the behavior in low-frequency and high-frequency is very different.

and give this as input to the LSTM. At the next step, we stride the window over the input by S, and take the next F features, denoted by $x_1 = x_t^{S:(F+S)} \in \Re^F$, and pass this to the LSTM.

- Benefits: run **multiple** Grid-LSTMs in **parallel**
- Question: how to segment AMS feature to 8 small frequency block?

0:20	20:40	40:60	60:80	80:100	100: 120	120: 140	140: 160



Frequency block Gird LDNN

Tensorflow function:

```
__init__(
    num_units,    num of cells
    use_peepholes=False,
    share_time_frequency_weights=False,
    cell_clip=None,
    initializer=None,
    num_unit_shards=1,
    forget_bias=1.0,
    feature_size=None,  filtersize
    frequency_skip=None,  stride
    num_frequency_blocks=None,  [L1,L2,...,L8]   Li = (ith_start-ith_end - filtersize)/stride + 1
    start_freqindex_list=None,  [0,20,40,...,140]
    end_freqindex_list=None,    [20,40,...,160]
```

```
num_shifts = int((160 - FILTESIZE) / STRIDE + 1)
grid_lstm_cell = tf.contrib.rnn.BidirectionalGridLSTMCell(num_units=NUM_GRID,
                                                           feature_size=FILTESIZE,
                                                           num_frequency_blocks=[num_shifts],
                                                           frequency_skip=STRIDE,
                                                           share_time_frequency_weights= SHARING
                                                           )
grid_lstm_cell = rnn_cell.DropoutWrapper(grid_lstm_cell, output_keep_prob=0.9)
grid_output, _ = tf.nn.dynamic_rnn(cell=grid_lstm_cell,
                                   inputs=input,
                                   time_major=False,
                                   dtype=tf.float32)
```

Frequency block Gird LDNN- current state

- Grid LSTM is processing in both **time** and **frequency**.

- The amount of Grid LSTMs unrolled over frequency
  - **linear increase** the need of GPU memory
  - Increase computational complexity

- Trade off between memory & batch_size (time)

- Example: one frequency block. One layer LSTM with 128 neurons and one DNN layer with 128 neurons.

	Memory usuage	1 epoch for 1 scene
Stride=40, filtersize=80	8.5GB （batch size 40)	240s
Stride=20, filtersize=80 Stride=20, filtersize=20 Stride=20, filtersize=40 Stride=80, filtersize=80	8.5GB （batch size 40) 11.7GB 11.7GB	380s 550s 440s 170s
LDNN	1.6GB	38s

Bidirectional frequency block Gird LDNN

- The amount of LSTM unrolled over frequency for each block:

$$L = (N - F)/S + 1.$$

- **Small** stride size S will make L **large**. Then too much need of gpu memory and too slow running time.

- Increase the stride size S can reduce computation cost, but decrease accuracy.
    - Their hypothesis [1] of bidirectional GLDNN which hope large stride can reduce computational cost without the loss in accuracy failed on their dataset.
    - fbGrid-LSTM can reduce computational cost by a factor of the number of frequency block.
    - Also achieve same performance compared to conventional Grid-LSTM.

- Question: depend on my experiment, it is extremely memory consuming.
  Even the simplest network requires a lot of memory to unrolled grid-lstm over frequency.
  how to trade off between the usage of gpu memory and batch_size (time)?

[1]Reducing the Computational Complexity of Two-Dimensional LSTMs. *Bo Li, Tara N. Sainath*