Dand_Assignment_HW02
Deep Learning

LSTM with the complete data set.

In your report you will include a filled-out table like this:

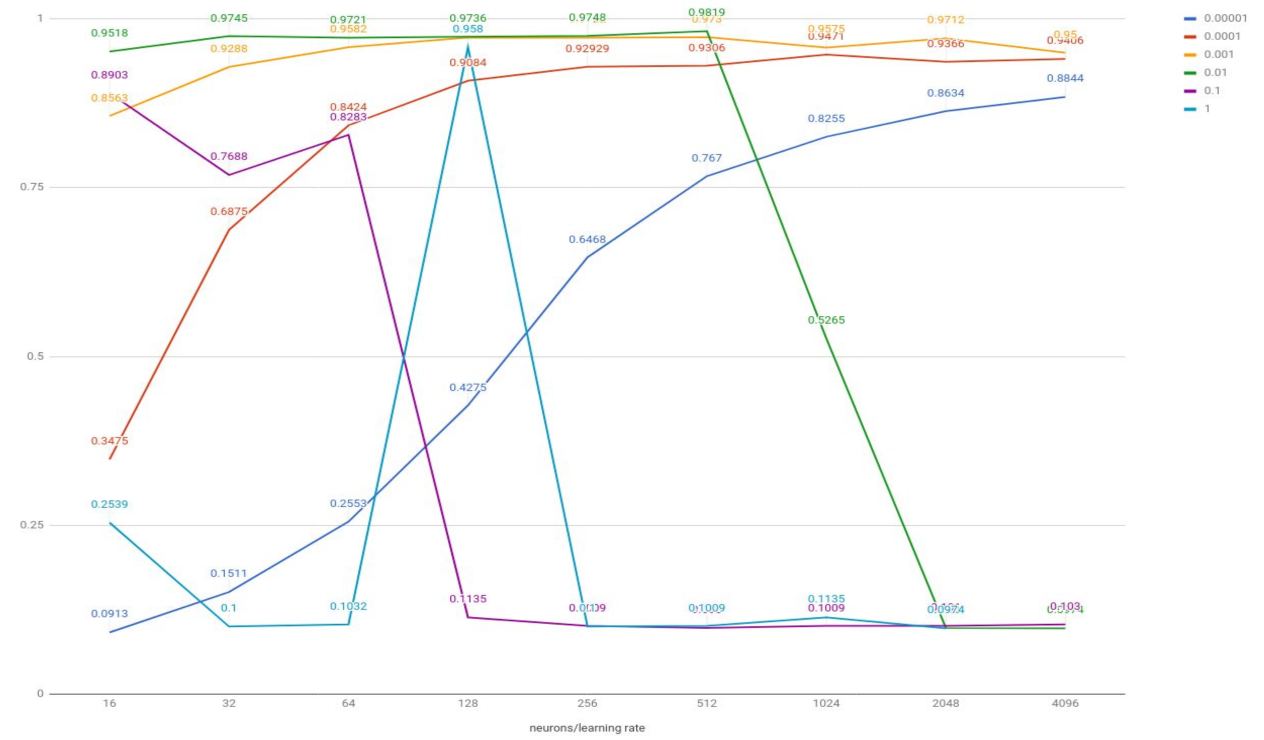
neurons/learning rate	0.00001	0.0001	0.001	0.01	0.1	1.0
16	0.0913	0.3475	0.8563	0.9518	0.8903	0.2539
32	0.1511	0.6875	0.9288	0.9745	0.7688	0.100
64	0.2553	0.8424	0.9582	0.9721	0.8283	0.1032
128	0.4275	0.9084	0.9724	0.9736	0.1135	0.958
256	0.6468	0.9292	0.9722	0.9748	0.1009	0.1
512	0.767	0.9306	0.973	0.9819	0.098	0.1009
1024	0.8255	0.9471	0.9575	0.5265	0.1009	0.1135
2048	0.8634	0.9366	0.9712	0.098	0.101	0.0974
4096	0.8844	0.9406	0.95	0.0974	0.103	0.113

Training LSTM with number of neurons - In this project, we have trained the network with respect to hidden units from 16 to 4096. While having higher number of neurons, we are getting better accuracy even with the learning rate of 0.00001. While comparing the learning rate from 0.00001 to 1.0 with respect to number of neurons, the learning accuracy is in form of sigmoid function that is it increases till learning rate of 0.01 and then starts dropping for most of the cases in the chart above. For this experiment, the learning rate 0.001 somehow has produced the results for all neurons above 90% except for 16 neurons.

Among all the combinations, the best one according to me is with 512 neurons which produced best accuracy of 98.19% with learning rate 0.01.

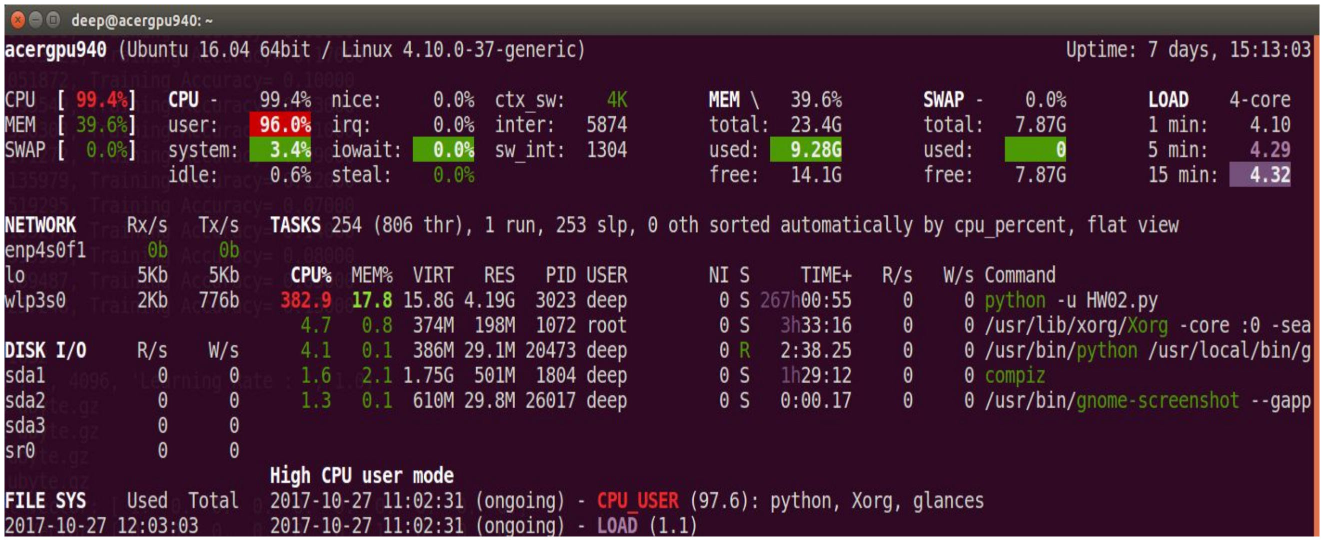
It means that having a smallest learning rate and 4000 number of neurons can still produce decent results but then it takes really long time to finish training the model. A larger training rate and small number of neurons can complete the training soon but results are not decent. A good combination of the learning rate (around 0.01 or 0.001) and range of neurons between 256-512 for this type of problem can work the best.

Dand_Assignment_HW02
Deep Learning



The graph shows number of neurons on x axis and learning rate on y axis.

Time to perform the experiment ~267 hours and counting (since the No of neurons - 4096 and Learning rate 1.0 is not finished yet!)



```
deep@acergpu940: ~
acergpu940 (Ubuntu 16.04 64bit / Linux 4.10.0-37-generic)                           Uptime: 7 days, 15:13:03

CPU  [ 99.4%]   CPU -     99.4%  nice:     0.0%  ctx_sw:    4K     MEM \   39.6%     SWAP -    0.0%      LOAD    4-core
MEM  [ 39.6%]   user:     96.0%  irq:      0.0%  inter:   5874     total:  23.4G     total:   7.87G      1 min:    4.10
SWAP [  0.0%]   system:    3.4%  iowait:   0.0%  sw_int:  1304     used:   9.28G     used:        0      5 min:    4.29
                idle:      0.6%  steal:    0.0%                    free:   14.1G     free:    7.87G      15 min:   4.32

NETWORK    Rx/s   Tx/s   TASKS 254 (806 thr), 1 run, 253 slp, 0 oth sorted automatically by cpu_percent, flat view
enp4s0f1     0b     0b
lo          5Kb    5Kb    CPU%  MEM%  VIRT   RES   PID USER        NI S    TIME+   R/s   W/s Command
wlp3s0      2Kb   776b   382.9  17.8 15.8G 4.19G  3023 deep         0 S 267h00:55    0     0 python -u HW02.py
                           4.7   0.8  374M  198M  1072 root         0 S   3h33:16    0     0 /usr/lib/xorg/Xorg -core :0 -sea
DISK I/O     R/s    W/s    4.1   0.1  386M 29.1M 20473 deep         0 R   2:38.25    0     0 /usr/bin/python /usr/local/bin/g
sda1           0      0    1.6   2.1 1.75G  501M  1804 deep         0 S   1h29:12    0     0 compiz
sda2           0      0    1.3   0.1  610M 29.8M 26017 deep         0 S   0:00.17    0     0 /usr/bin/gnome-screenshot --gapp
sda3           0      0
sr0            0      0
                         High CPU user mode
FILE SYS    Used  Total  2017-10-27 11:02:31 (ongoing) - CPU_USER (97.6): python, Xorg, glances
2017-10-27 12:03:03      2017-10-27 11:02:31 (ongoing) - LOAD (1.1)
```