COVER SHEET

STUDENT NAME: Sudhanshu Kasewa

DEGREE AND YEAR: MSC Machine Learning 2017

MODULE CODE: COMPGI13/COMPM050

MODULE TITLE: Advanced Topics in Machine Learning

COURSEWORK TITLE: Assignment 3

LAB GROUP (if applicable): N/A

DATE OF LAB SESSION (if applicable): N/A

DATE COURSEWORK DUE FOR SUBMISSION: 11:55PM 11TH APRIL

ACTUAL DATE OF SUBMISSION: 11TH APRIL

LECTURER'S NAME (who set courswork): Thore Graepel, Koray Kavukcuoglu, Hado van Hasselt, Joseph Modayil

PERSONAL TUTOR'S NAME: Mark Herbster

DECLARATION BY STUDENT

By submitting this coursework with this information, I am confirming that the coursework is entirely my own work and that I have clearly referenced any quotations, ideas, judgements, data, figures, software or diagrams that are not my own.

I have read the UCL guidance and advice on plagiarism available from http://www.ucl.ac.uk/current-students/guidelines/plagiarism and I understand that any false claim in respect of this work will result in disciplinary action in accordance with the University of London's General Regulations for Internal Students.

Signed

Sudhanshu Kasewa

Assignment 3: Deep reinforcement learning

Sudhanshu Kasewa

Student number: 15115014

April 6, 2017

1 PROBLEM A: CART-POLE

We examine several different algorithms on the Cart-pole problem. For all learning in this section, tensorflow's `GradientDescentOptimizer` was used. Learning rates are reported in each question separately.

1.1 3 TRAJECTORIES UNDER RANDOM POLICY

Three trajectories under random policy are reported below. Since there was no learning here, no learning rate was used. The trajectories are in the form state (a 4-tuple) followed by an action (0 or 1). Rewards are all 0 except for the terminal state where they are -1.

```
 Problem A: 1
[-0.03787321 -0.01932859 0.04120055 -0.01311537] 0
[-0.03825978 -0.21501645 0.04093824 0.29227685] 1
[-0.04256011 -0.02050139 0.04678378 0.0127812 ] 1
[-0.04297014 0.17391949 0.0470394 -0.2647816 ] 0
[-0.03949175 -0.0218412 0.04174377 0.04235909] 1
[-0.03992857 0.17265807 0.04259095 -0.23686655] 1
[-0.03647541 0.36714648 0.03785362 -0.51581664] 1
[-0.02913248 0.5617155 0.02753729 -0.7963348 ] 1
[-0.01789817 0.75644896 0.01161059 -1.0802293 ] 1
```

```
[-0.00276919 0.95141571 -0.00999399 -1.36924628] 1
[ 0.01625912 1.14666127 -0.03737892 -1.66503818] 1
[ 0.03919235 1.34219773 -0.07067968 -1.96912484] 0
[ 0.0660363 1.14788982 -0.11006218 -1.69915321] 0
[ 0.0889941 0.95419521 -0.14404524 -1.44266228] 1
[ 0.108078 1.15076667 -0.17289849 -1.77666875] 0
[ 0.13109334 0.95796295 -0.20843186 -1.54235443] 0
[ 0.1502526 0.765866 -0.23927895 -1.32128168]
Episode finished after 16 timesteps Reward from initial state is -0.8600583546412883

[ 0.02580076 -0.04125614 -0.0221125 0.0319984 ] 0
[ 0.02497563 -0.23605412 -0.02147253 0.31762339] 0
[ 0.02025455 -0.43086376 -0.01512006 0.60345805] 0
[ 0.01163728 -0.62577101 -0.0030509 0.89134038] 1
[-0.00087814 -0.4306078 0.0147759 0.59769997] 0
[-0.0094903 -0.62593336 0.0267299 0.89500028] 1
[-0.02200897 -0.43118387 0.04462991 0.61083804] 1
[-0.03063265 -0.23671321 0.05684667 0.33253936] 0
[-0.03536691 -0.43259627 0.06349746 0.64259325] 0
[-0.04401883 -0.62854313 0.07634932 0.95457691] 1
[-0.0565897 -0.43452667 0.09544086 0.68682494] 1
[-0.06528023 -0.24085002 0.10917736 0.42564914] 1
[-0.07009723 -0.04743019 0.11769034 0.16928122] 1
[-0.07104584 0.14582777 0.12107597 -0.08407929] 0
[-0.06812928 -0.05080295 0.11939438 0.24421699] 1
[-0.06914534 0.14242928 0.12427872 -0.00854929] 0
[-0.06629675 -0.05423559 0.12410773 0.32061676] 1
[-0.06738147 0.13892053 0.13052007 0.06950514] 0
[-0.06460305 -0.05780764 0.13191017 0.40035148] 1
[-0.06575921 0.13522062 0.1399172 0.15199372] 1
[-0.06305479 0.32809071 0.14295708 -0.09348261] 0
[-0.05649298 0.13123994 0.14108742 0.2406678 ] 1
[-0.05386818 0.32409415 0.14590078 -0.00439689] 1
[-0.0473863 0.51685512 0.14581284 -0.24772351] 0
[-0.0370492 0.31998433 0.14085837 0.08716616] 0
[-0.03064951 0.12315379 0.1426017 0.42076337] 1
[-0.02818643 0.31599778 0.15101696 0.17621649] 0
[-0.02186648 0.11907348 0.15454129 0.51247144] 1
[-0.01948501 0.31171966 0.16479072 0.27220054] 1
[-0.01325062 0.50415376 0.17023473 0.03568932] 1
[-0.00316754 0.69647751 0.17094852 -0.19881862] 1
[ 0.01076201 0.88879462 0.16697215 -0.43307498] 1
[ 0.0285379 1.08120773 0.15831065 -0.66881865] 0
[ 0.05016206 0.88428004 0.14493427 -0.33077182] 1
```

```
[ 0.06784766 1.07707357 0.13831884 -0.57446841] 0
[ 0.08938913 0.88031121 0.12682947 -0.24160755] 1
[ 0.10699535 1.07341482 0.12199732 -0.49174845] 1
[ 0.12846365 1.26662381 0.11216235 -0.74362959] 1
[ 0.15379613 1.46003373 0.09728976 -0.99901489] 1
[ 0.1829968 1.65373005 0.07730946 -1.25962731] 0
[ 0.2160714 1.45770889 0.05211691 -0.94376746] 1
[ 0.24522558 1.65209143 0.03324156 -1.21962994] 0
[ 0.27826741 1.45655713 0.00884897 -0.91671931] 0
[ 0.30739855 1.26131665 -0.00948542 -0.6212685 ] 1
[ 0.33262488 1.45656977 -0.02191079 -0.91692367] 1
[ 0.36175628 1.65198102 -0.04024926 -1.21641145] 0
[ 0.3947959 1.45740065 -0.06457749 -0.93660736] 1
[ 0.42394391 1.65333123 -0.08330964 -1.24886322] 0
[ 0.45701054 1.45937019 -0.1082869 -0.98339544] 1
[ 0.48619794 1.65576321 -0.12795481 -1.30803363] 0
[ 0.5193132 1.46247365 -0.15411549 -1.05798566] 0
[ 0.54856268 1.26969187 -0.1752752 -0.81737116] 0
[ 0.57395652 1.07734676 -0.19162262 -0.58454145] 0
[ 0.59550345 0.88535225 -0.20331345 -0.35781386] 0
[ 0.6132105 0.69361319 -0.21046973 -0.13549236]
Episode finished after 54 timesteps Reward from initial state is -0.5870367819374844

[-0.01132796 0.02695163 0.03791375 -0.02881114] 1
[-0.01078893 0.22150994 0.03733752 -0.30929491] 1
[-0.00635873 0.41608056 0.03115162 -0.58997275] 0
[ 0.00196288 0.2205366 0.01935217 -0.28764201] 0
[ 0.00637361 0.02514409 0.01359933 0.01108105] 0
[ 0.0068765 -0.17017022 0.01382095 0.30802351] 1
[ 0.00347309 0.0247521 0.01998142 0.01973119] 1
[ 0.00396813 0.21958188 0.02037604 -0.266581 ] 0
[ 0.00835977 0.02417513 0.01504442 0.03245841] 1
[ 0.00884327 0.21907815 0.01569359 -0.25544013] 0
[ 0.01322484 0.02373568 0.01058479 0.04215121] 0
[ 0.01369955 -0.17153644 0.01142781 0.33815486] 1
[ 0.01026882 0.02342104 0.01819091 0.04909743] 0
[ 0.01073724 -0.17195696 0.01917286 0.34746378] 1
[ 0.0072981 0.02288711 0.02612213 0.06088787] 1
[ 0.00775584 0.21762497 0.02733989 -0.22344024] 1
[ 0.01210834 0.41234571 0.02287109 -0.50737526] 1
[ 0.02035526 0.60713806 0.01272358 -0.79276386] 1
[ 0.03249802 0.80208305 -0.00313169 -1.08141704] 1
[ 0.04853968 0.9972462 -0.02476004 -1.37508105] 0
[ 0.0684846 0.80244227 -0.05226166 -1.09024342] 1
```

```
[ 0.08453345 0.9982127 -0.07406652 -1.39885606] 1
[ 0.1044977 1.1941732 -0.10204365 -1.71374658] 0
[ 0.12838117 1.00035985 -0.13631858 -1.45448765] 0
[ 0.14838837 0.80714938 -0.16540833 -1.20731422] 1
[ 0.16453135 1.00397589 -0.18955462 -1.54693292] 0
[ 0.18461087 0.81156772 -0.22049327 -1.31888613]
Episode finished after 26 timesteps Reward from initial state is -0.7778213593991465
```

From the results in the following section, we will see that the second trajectory is an outlier under the random policy.

1.2 100 RUNS UNDER RANDOM POLICY

The mean and standard deviations of returns and episode lengths over a hundred runs under a uniform random policy are reported below.

```
Mean and stddev.  of episode times:              22.06       10.9414989832
Mean and stddev.  of episode rewards:  -0.805817253927       0.0836435358041
```

It is clear from these results that the second run in section 1.1 of 54 timesteps is an outlier over 3-sigma away from the mean.

1.3 BATCH Q-LEARNING

The version of batch Q-learning we implemented is similar to that as described in Chapter 2 of "Reinforcement Learning: State-of-the-Art.", edited by Wiering, Marco, y Martijn van Otterlo [1], as the algorithm 'fitted-Q-iteration'.

In these experiments, we used learning rates of $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.5]$. We used a mini-batch size of 5000.

1.3.1 LINEAR TRANSFORM

The linear transform did not include any bias. Results are given below. Evaluation was over 10 episodes after every epoch of training.

1.3.2 HIDDEN LAYER WITH 100 UNITS, FOLLOWED BY RELU

The single-layer neural network did not include any biases. Experimentally, biases proved to be much harder to train, and so were dropped from further experiments. All further experiments in part A do not have biases. Evaluation was over 10 episodes after every epoch of training.

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.229.787&rep=rep1&type=pdf

1.4 Online Q-learning

The online Q-learning experiment was run 100 times with the following parameters: Learning rate: 0.01. A graph of the average reward and standard deviation is given below.

1.5 Changing the hidden layer

We modified the above agent and ran the experiment one time each to observe any change in behaviour. The learning rate was 0.01.

1.5.1 Fewer units (30)

Given below is a graph using a hidden layer of 30 units.

1.5.2 More units (1000)

Given below is a graph using a hidden layer of 1000 units.

1.6 Adding experiece replay

Given below is a graph of the behaviour of an agent from Q4 augmented with an experience replay buffer of size 8000. The learning rate is 0.01.

1.7 Adding a target network

The last agent was modified to include a target network, which was updated to the current network every 5 episodes. The learning rate used is 0.01. Its behaviour is graphed below.

1.8 Implementing a SARSA agent

We started with the agent from Q4 and modified it to learn as per the SARSA algorithm. The learning rate used is 0.01. Its behaviour is graphed below.

2 Problem B: Atari Games

For these problems, the following specifications for the world state were implemented.

- Observations were preprocessed from $210 \times 160 \times 3$ to $28 \times 28 \times 1$
- Rewards were clipped to -1, 0 or 1

Further, the following features were implemented.

- An empty experience buffer of 100,000 transitions was created.
- The Q function was approximated using a two-stage convolutional neural network.
- A batch Q-learning agent was implemented, using mini-batch sizes of 32.

- RMSPropOptimizer was used for training the agent.
- Performance of the agent was evaluated every 50,000 steps with a hundred runs of a greedy policy.

2.1 RANDOM POLICY

An agent following a random policy was evaluated on 100 runs of each game. Mean and standard deviations are reported below. No learning rate was required for this experiment.

```
 Game :  Pong-v3
Possible actions :  6
Evaluation over 100 episodes
Avg steps:  1260.15              Standard deviation:  164.947165783
Avg tot rew:  -20.13              Standard deviation:  0.955562661472
Avg disc rew:  -0.889193101266     Standard deviation:  0.319726787337

 Game :  MsPacman-v3
Possible actions :  9
Evaluation over 100 episodes
Avg steps:  646.66                Standard deviation:  102.025998647
Avg tot rew:  211.9                Standard deviation:  112.718188417
Avg disc rew:  24.5710619061       Standard deviation:  6.80923453538

 Game :  Boxing-v3
Possible actions :  18
Evaluation over 100 episodes
Avg steps:  2381.73                  Standard deviation:  12.9459298623
Avg tot rew:  0.61                  Standard deviation:  4.21163863597
Avg disc rew:  -0.226079547769       Standard deviation:  1.0947763914
```

2.2 UNTRAINED Q-NETWORK

An agent following a greedy policy on an initialised but untrained Q-network was evaluated on 100 runs of each game. Mean and standard deviations are reported below. No learning rate was required for this experiment.

```
 Game :  Pong-v3
Possible actions :  6
Evaluation over 100 episodes
Avg steps:  1020.06                Standard deviation:  8.65658131135
Avg tot rew:  -21.0                Standard deviation:  0.0
Avg disc rew:  -1.12266604771      Standard deviation:  0.0284774340357
```

```
 Game :  MsPacman-v3
Possible actions :  9
Evaluation over 100 episodes
Avg steps:  458.0                   Standard deviation:  5.06557005677
Avg tot rew:  60.0                   Standard deviation:  0.0
Avg disc rew:  21.784172183         Standard deviation:  0.544610419163

 Game :  Boxing-v3
Possible actions :  18
Evaluation over 100 episodes
Avg steps:  2381.73                 Standard deviation:  12.9459298623
Avg tot rew:  -25.0                 Standard deviation:  0.0
Avg disc rew:  -0.373366658935       Standard deviation:  0.0166110046567
```

As we can see, the performance of this agent is different from the agent under a random policy from Q1. Our first observation is that the standard deviation for total reward is always 0. This implies that the untrained Q-learning agent takes the same actions every single time regardless of stochasticity in the environment and the inputs. This is expected because the untrained agent would have weights that cannot give meaningful Q values; it would most likely have Q values that were so far away from each other, that their order from max to min is not affected by input state, and hence would take the same actions every time for any input state. Secondly, the untrained agent seems to perform worse than the random agent. This is because the random agent on average takes several different moves, allowing it to turn down corners in Ms Pacman, or occasionally return the ball and score in Pong, which our Q-learning agent does not do.
The linear transform did not include any bias. Results are given below.
We first examine the functionality of LSTMs and GRUs, and answer three key questions.
For reference, we provide all the equations required for updating LSTMs:

$$i_t = W_{ix}x_t + W_{ih}h_{t-1} + b_i \tag{2.1}$$
$$j_t = W_{jx}x_t + W_{jh}h_{t-1} + b_j \tag{2.2}$$
$$f_t = W_{fx}x_t + W_{fh}h_{t-1} + b_f \tag{2.3}$$
$$o_t = W_{ox}x_t + W_{oh}h_{t-1} + b_o \tag{2.4}$$

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(j_t) \tag{2.5}$$
$$h_t = \sigma(o_t) \odot tanh(c_t) \tag{2.6}$$

Following are the equations for GRUs:

$$f_t = W_{fx}x_t + W_{fh}h_{t-1} + b_f \tag{2.7}$$
$$i_t = W_{ix}x_t + W_{ih}h_{t-1} + b_i \tag{2.8}$$

$$z_t = \sigma(f_t) \tag{2.9}$$
$$r_t = \sigma(i_t) \tag{2.10}$$

$$h = tanh(W_{hx}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{2.11}$$
$$h_t = (1 - z_t) \odot h + z_t \odot h_{t-1} \tag{2.12}$$

2.3 Can they store the current input?

2.3.1 LSTM

In order for an LSTM to store the current inputs x_t for the next time-step, we would want to find weights W and biases b such that $c_t = x_t$. From equation 1.5 we can see that for $c_t = x_t$ we need to have the first part of the RHS equal zero, and the second part equal x_t. We can have $\sigma(f_t) \odot c_{t-1} = 0$ for highly negative values of b_f. However, since the remaining term $\sigma(i_t) \odot tanh(j_t)$ is a combination of σ and $tanh$ operators, it is bounded in the region (-1,1) and inasmuch can never equal x_t that lies outside this region.

Therefore, the value of x_t cannot be stored identically in the cell memory c_t. By the same token, we cannot have $h_t = x_t$ because h_t is bounded by (-1,1).

However, the cell memory c_t can store a **representation** of x_t that is unique to x_t. In particular, if we have all the $W_{.h}$ weights equal 0, all the $W_{.x}$ weights equal 1, and all the biases equal 0, except for a high negative value for b_f, then c_t reduces to a function of x_t.

2.3.2 GRU

The question asks whether the state of the GRU h_t can store the value of the input. Using the same logic as above, we try and get rid of the h_{t-1} term from the RHS of equation 1.12 by setting b_f to have a high negative value, so that z_t tends to 0. However, we run into the same problem again, since now h_t is simply the evalution of a $tanh$ function, and is therefore not able to take on any values outside the bound (-1,1), and in particular, cannot identically ever be equal to any x outside this bound.

We can have a representation of x_t in the hidden state, by setting $W_{hh} = 0$ and $b_h = 0$, and having z_t tend to zero, as before.

2.4 Can they ignore the current input?

2.4.1 LSTM

For an LSTM's states c_{t-1} and h_{t-1} to persist to the next time state, we would need to negate the effect of the input x_t entirely. This is possible in an LSTM cell if input gate $\sigma(i_t) = 0$, the forget gate $\sigma(f_t) = 1$, and the output gate does not change from time $t-1$ to time t, i.e. $\sigma(o_t) = \sigma(o_{t-1})$. In this situation, equations 1.5 and 1.6 become:

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(j_t) = 1 \odot c_{t-1} + 0 \odot tanh(j_t) = c_{t-1} \tag{2.13}$$

$$h_t = \sigma(o_t) \odot tanh(c_t) = \sigma(o_{t-1}) \odot tanh(c_{t-1}) = h_{t-1} \tag{2.14}$$

The output gate condition can be realised by setting b_o to be a large positive value.

2.4.2 GRU

For a GRU to preserve its hidden state h_{t-1} in the next time step, we would need to have $z_t = 1$ in equation 1.12. z_t is the GRU's forget gate, and $z_t = 1$ can be realised if b_f has a very high positive value.

$$b_f \gg 0 \Rightarrow \sigma(f_t) = 1 = z_t$$

And thus, equation 1.12 becomes :

$$h_t = (1 - z_t) \odot h + z_t \odot h_{t-1} = (1-1) \odot h + 1 \odot h_{t-1} = h_{t-1}$$

2.5 Are GRUs a special case of LSTMs?

We interpreted this question to mean whether or not any arbitrary input-output pair that can be learned by a GRU can be learned by an LSTM. If any arbitary input-output pair that can be computed by an GRU can also be computed by an LSTM with suitable weights, then GRUs might be a special case of LSTMs, and further exploration would be required to prove this. However, if there is some input-output pair which can be computed by a GRU which cannot be computed by an LSTM then the question of whether it is a special case is moot.

The inputs are clearly in the form x_t. For clarity of notation, we consider the output of the LSTM to be equation 1.6, $h_t = h_{t,LSTM}$, and for GRUs to be equantion 1.12, $h_t = h_{t,GRU}$.

We propose that the GRU is not a special case of the LSTM. Specifically, we can have the GRU output arbitrarily large values as $h_{t,GRU}$, by selecting the initial state $h_{0,GRU} = k$, and then setting $z_t = 1 \forall t$, persisting this internal state, which is also the output of the GRU.

On the other hand, the output for LSTMs $h_t = h_{t,LSTM}$ is always bounded by (-1, 1) as evidenced from equation 1.6, since the expression $h_t = \sigma(o_t) \odot tanh(c_t)$ is the product of two bounded operators and is therefore bounded by (-1, 1). This bound cannot be breached, regardless of the settings of the weights, biases, or the initial values for the hidden state and the

memory of the LSTM.

Specifically, consider any input-output pair $(x, 2)$. Then we can set $h_{0,GRU} = 2$, and choose weights and biases for equation 1.7 such that $z_t = 1$. Therefore, the output of the GRU at any time is $h_{t,GRU} = 2$. However, under no circumstances can $h_{t,LSTM} = 2$ because $h_{t,LSTM}$ is bounded by (-1, 1).

3 Task 1: Classification

We explored the effectiveness of LSTMs and GRUs learning a probability distribution over the digits given the (sequential) input of the image pixels.

We present the results of our experiments below.

Model: LSTM	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layer, 32 units)
Testing Loss	0.5920	0.4936	0.2344	0.5636
Training Loss	0.6035	0.5170	0.2326	0.6556
Testing Accuracy	79.78%	81.55%	93.32%	79.35%
Training Accuracy	79.35%	82.18%	93.30%	76.72%

Model: GRU	(1 layer, 32 units)	(1 layer, 64 units)	(1 layer, 128 units)	(3 layer, 32 units)
Testing Loss	0.2463	0.0824	0.0616	0.1505
Training Loss	0.2454	0.0692	0.0482	0.2015
Testing Accuracy	92.47%	97.70%	98.27%	95.30%
Training Accuracy	92.36%	97.82%	98.50%	93.44%

For comparison, the loss reported is averaged over number of instances. Thus, the total loss can be computed by multiplying by 55,000 for the training loss, and multiplying by 10,000 for the test loss.

Comparison among these models:
Our first observation is that GRUs tend to outperform LSTMs consistently. Not only to do they achieve higher test accuracy than their equivalent LSTM architectures, the weakest GRU model (single layer, size 32) performed just under the best LSTM model (single layer, size 128). GRUs generally trained faster per epoch, improved in accuracy faster, and were more robust to changes in hyperparameters.

Secondly, the best performers in both categories for this task were the single layer, size 128 models.

Thirdly, the stacked variants before better than their single layer cousins, but not by much.

Comparison with feedforward and convolutional models:
It is clear that while some models to achieve high testing accuracy, these models are in general inferior to the regular networks or convolutional networks explored in earlier experiments. 'Seeing' the entire image at once certainly grants those models an edge over our pixel-by-pixel many-to-one classifiers. This is probably because the LSTM and GRU cells need to learn long-term dependencies, which can be difficult; in any case, these networks certainly require more time and training in order to achieve even comparable results. It should be noted that some GRU models did train to very high test accuracy, which are better than the simpler feed-forward models from the previous experiments.

Hyperparameter tuning: We identified the following five hyperparameters to be tuned. We also described how they were tuned.

- Optimisation algorithm, among SGD, Adagrad, Adadelta, and Adam: Using the simplest network, we ran trials up to 10 epochs using these optimisers with learning rates $\in 0.1, 0.01, 0.001, 0.0001$. We selected the best performing optimiser under this setup (Adam) and used it for all further experiments. With a learning rate of 0.0001 Adam steadily learned to an accuracy of 40% on the test set within 10 epochs. Other optimisers and learning rates did not fair as well, often not achieving test accuracy over 30%, and sometimes not learning at all within 10 epochs, e.g. SGD with a learning rate of 0.1.

- Learning rate: After selecting Adam as an optimizer, learning rate was tweaked for each model in the range 0.0001 to 0.001 in increments of 0.0001 till it appeared that the model would successfully learn. For many models, this meant that 0.0001 sufficed; some models learned at 0.0002. LSTM with a hidden size of 32 was trained with a learning rate of 0.0005.

- Batch size: This parameter was increased to the degree allowed by memory, to enable faster training. Its impact on training accuracy was not recorded.

- Dropout percentage (for stacked networks only): This parameter was not tuned, owing to time constraints. For the stacked models we have used an `input_keep_prob=0.75` while training, although we did not validate it against a null hypothesis.

- Gradient clip value: In all experiments, gradients were clippped at -1 and 1 to avoid the exploding gradient problem and large jumps across the loss function surface.