COL870: Deep Learning
Assignment 2

Harman Singh 2018EE10542 Shubham Mittal 2018EE10957

Q1 Conditional GAN

In order to train the cGAN we need to create a sufficient supervised dataset from the available one-shot data (1 example per class). We describe the different data clustering methods we used in the following section and also report the FID score of cGAN trained on the dataset obtained from these methods.

Data Clustering Methods

First we report the results obtained using K-Means clustering method. Since the dataset (unsupervised) is large (64k digits each in query and target), we use mini-batch K-Means.

A) minibatch - KMeans :

Instead of initialising cluster centers randomly, **we provide extra supervision by initialising them with the available supervised data** (sample images). We saw a boost in the performance because of this initialisation. With random initialisation, we observed that the 0th class is clustered into multiple different classes and did not give good enough (supervised) data to work with.

B) minibatch - Kmeans sampled :

Same as minibatch Kmeans but once clusters are formed, for each cluster we sample closest "n" samples to the corresponding labeled sample image. Results in better dataset but **less diversity** in the dataset (as all the data thus formed is close to one of the labelled sample images).

C) Unsupervised Data Augmentation (UDA):

We also use a semi-supervised learning method to learn the labels using limited supervised data. For this we use LeNet as the classifier, a simple CNN model for our simple dataset (Arabic MNIST). **We implemented the UDA paper from scratch for our data.**

Experimental detail: We use 9k supervised dataset (using sampling from near to given sample (supervised) data) and 9k unsupervised dataset (from the query data + target data randomly sampled). Data augmentation is applied on both and losses were chosen as mentioned in the paper (https://arxiv.org/abs/1904.12848). Our performance of clustering drastically improved using this.

Since we don't have the true labels for the unlabelled dataset, we measure the performance of clustering by using the constraints of sudoku on each board in the target set.

Thus, we define two performance metrics:

1. **8-level accuracy:**
    a. finding the number of correct rows, columns, groups (of 2x4) in each board.
    b. A row or column or group is correct if all the digits (given by clustering method) are distinct
2. **Board-level accuracy:**
    a. finding the number of correct boards
    b. A board is correct if it has all the constraints (i.e. rows, columns, groups) satisfied.

Note: we are not using the rules of sudoku for any learning setting, but only to evaluate (accurately) what the model is learning.

Clustering method	8-level accuracy %	Board-level accuracy %
minibatch-KMeans	22.3	0
minibatch-KMeans -sampled, n=15k	20.1	0
Unsupervised Data Augmentation	**70.51**	**5.47**

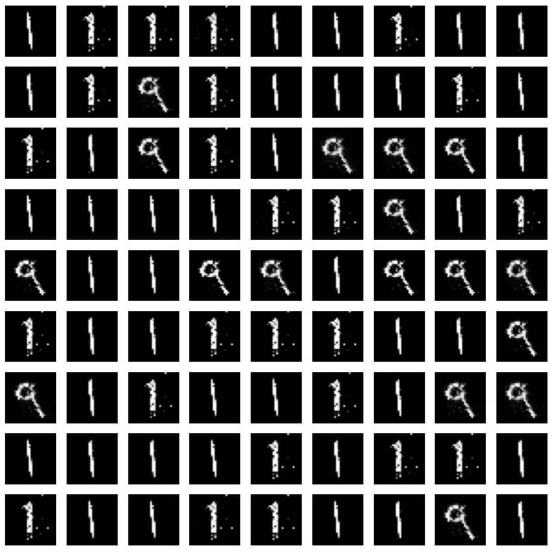
Training the cGAN model and its Generation Quality

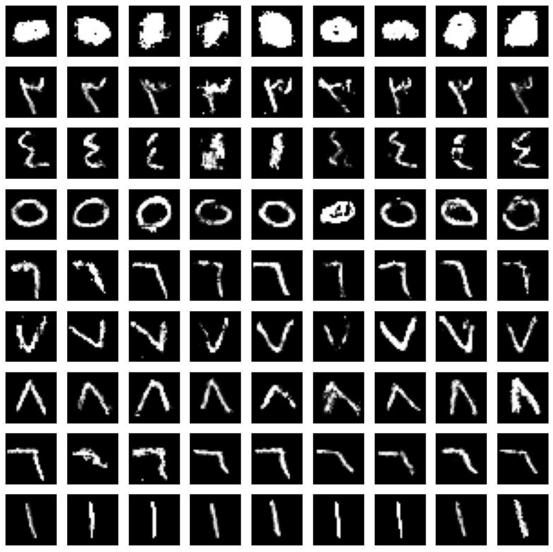
1. We use the exact cGAN architecture (consisting of fully connected layers, maxout layers and dropout), optimizer and hyperparameter settings (like learning rate) as given in [Mirza and Osindero, 2014].
2. It was not clear from the paper if the dropout layer is applied at the input, so we report the results with and without dropout layer. We saw better results with dropout.
3. The generation quality is measured using FID score (b/w 9k real and 9k (1k per class) generated data)

Experimental results of Q1:

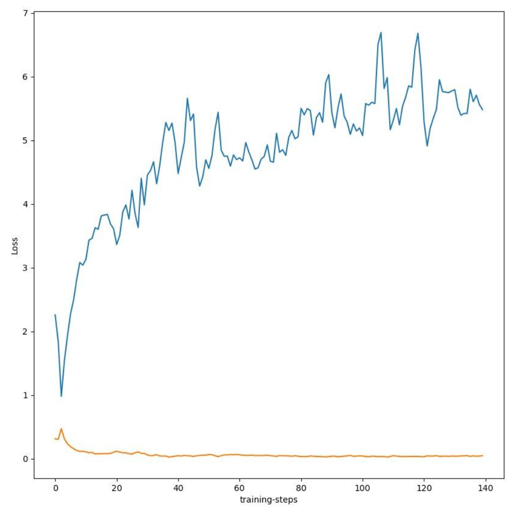
Clustering method	Data for GAN training	Dropout on input of GAN *	Epochs trained for	FID
minibatch-KMeans -sampled, n=15k	query	Yes	100	136.312
minibatch-KMeans -sampled, n=15k	query	No	150	123.441
minibatch-KMeans -sampled, n=15k	query	No	200	95.520
minibatch-KMeans	query	Yes	150	59.911
minibatch-KMeans	**query+target**	**Yes**	**150**	**49.981**
UDA	**query+target**	**Yes**	**150**	**45.37**

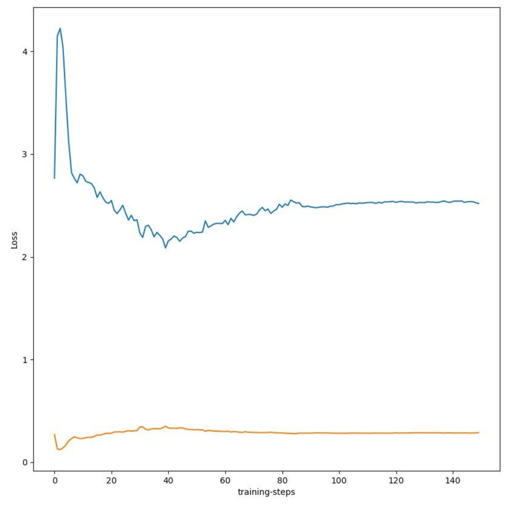
Note : all the above were trained using Adam optimizer. We saw convergence and mode collapse issues while training with SGD. The loss curves for both SGD and Adam are given in the appendix. Few sample data generated using cGAN with SGD (left) and with Adam (right), using KMeans are shown below :-





The loss curves (blue: generator, green:discriminator) for cGAN with SGD(left) and Adam(right) are shown below:





Comments on using clustering methods :

1. Sampled Kmeans, samples images from a particular cluster, which are close to the corresponding labelled image. This improves the clustering (there are images in a cluster that do not belong to that cluster) but reduces diversity in Images. This leads to less diversity and mode collapse like situation when training the cGAN with this data. FID score is less because the diversity in the GAN outputs is less (which is due to the fact that we had less diversity in our input data)
2. Once we get the labels for the unsupervised dataset using KMeans, we create another supervised dataset using cGAN. Upon training a classifier (LeNet) on this generated dataset, we didn't see much difference in the labelling accuracy (metrics defined above) as compared to that with K-Means.
3. UDA helped us increase the clustering accuracy by a lot as mentioned above.

Comments on training cGAN and data generation:

1. Since the training or learning of GANs is tricky, we trained the model for a lot of epochs. The training time was quite high, around 15 hours, but this ensured that model converged properly and giving FID score of **49.9**
2. We also inverted the image color (i.e. there are lot of black pixels like MNIST), and this helped in model convergence of the GAN and in getting better FID score (after inverting the images to white again). Learning is probably easier with black images as 0 is easier to learn than 1.

Thus in Q1, we got the supervised dataset (symbolic dataset) using KMeans and UDA.

Q2 Sudoku Solver

The RRN model given in Palm et al. [2018] is used to solve the combinatorial problem. In this part we use the symbolic dataset obtained using KMeans. The symbolic dataset created using UDA (which is much better than that with KMeans) is used in Q3.

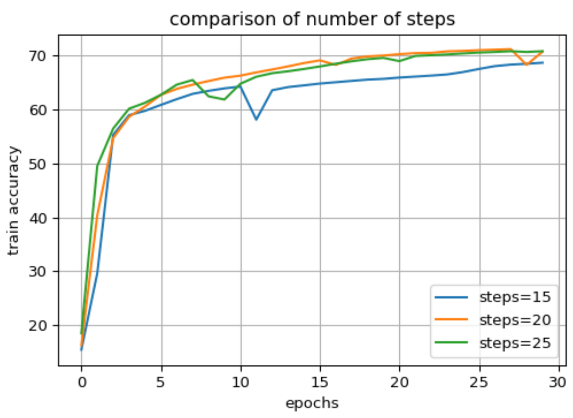
2.1 Recurrent Relational Network

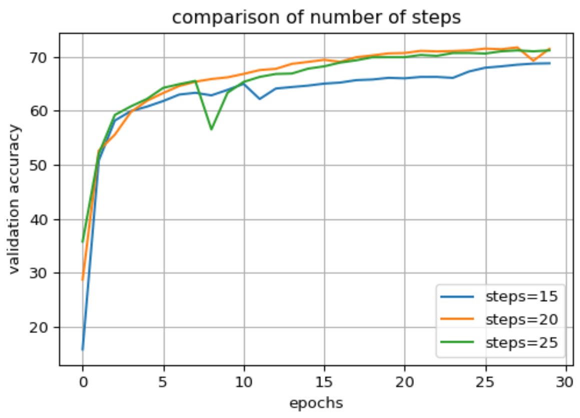
We use the same architectural details as given in the paper with slight modifications (as given below) which gave us considerable boost in the performance. We attempted to reduce the size of the model since we are using a smaller sudoku table.

1. 3 layer MLP (instead of 4 layers) with ReLU activations on each layer except the last layer.
2. The cell state (and not hidden state) of the LSTM cell is used at the softmax layer for final prediction.
3. We experimented with both learnable embeddings and one-hot embeddings for encoding each cell content (digit), row and column information, and both gave very similar performance. Thus, we use one-hot embeddings (to further reduce model size).
    a. There is no notion of similarity in the representation of different digits i.e. they are orthogonal
    b. No problem of high dimensionality like in word embeddings (from vocabulary)

Experimental details

1. Adam optimizer is used starting with 2e-4, with batch size of 64, trained for 60 epochs.
2. Since we are solving 8x8 Sudoku, and not 9x9, we use less number of (time) steps than what is given in the paper. The comparison of performance (digit level accuracy) with different number of steps is given below.
3. Train-val split of 95:5 on dataset of size 10k.



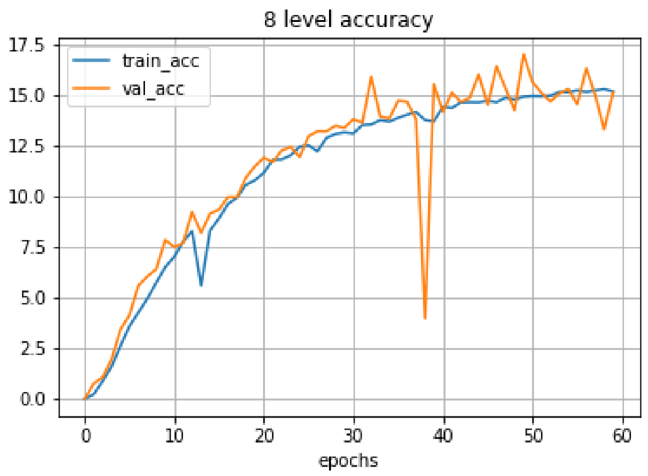


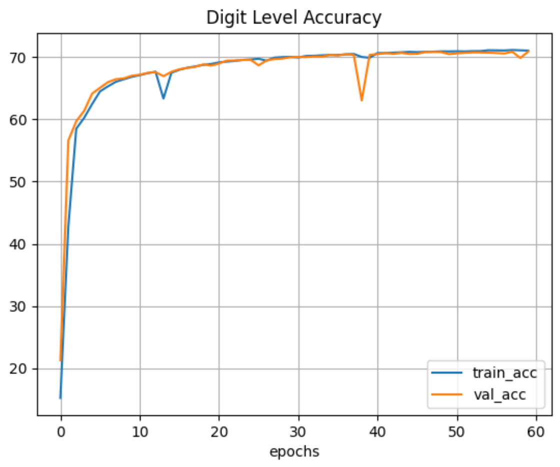
Since we get nearly the same performance for the number of steps being 20 and 25, we choose to work with the former as the training time is lesser.

Since in this part of assignment the symbolic dataset is fixed (i.e. the one obtained using Q1 clustering method), we define two more performance metrics:

1. Digit level accuracy: correct here means each digit (in predicted by RRN) equals the corresponding digit in (target) symbolic data
2. Sudoku level accuracy: correct here means complete 64 predicted values by RRN equal the 64 values (in target symbolic data)

Thus, using 20 number of steps and symbolic dataset (created using KMeans), the learning curves of RRN are shown below:





The sudoku level and board level accuracy of the model kept nearly 0, thus the plots are not shown.
Note: the left plot has accuracy in fraction i.e. in 0-1, whereas the right plot has accuracy on scale of 0-100.

Inferences:

1. The digit level accuracy curve tells that RRN has converged and learnt whatever it could from the symbolic data given to it.
2. The 8 level accuracy curve tells that the model keeps on learning the rules of the sudoku, and these are getting incorporated into its weights, or more specifically, the message function.
3. **NOTE:** this learning of rules by RRN forms the motivation for our work in the joint training (Q3).

Thus, in Q2, the RRN model trained on the symbolic dataset (created using KMeans) gave **14%** 8-level accuracy and **0%** board-level accuracy on validation data

Scores on validation data

Clustering method	Rnn 8 level accuracy	Rnn board level accuracy
kmeans-minibatch	14%	0%
UDA	**40%**	**2.9%**

Q3 Joint Training

Our contributions to joint training and improving overall models:

1. UDA classifier (explained in part 1)
2. Kmeans with supervision (ie initializing samples with sample_images.npy data)
3. Inductive biases applied to joint training (see below)
4. Improvement of GAN model using classifier of joint training, UDA classifier

From Q1, we got the classifier (LeNet using UDA) which gave:
70.51% 8-level accuracy and **5.47%** board-level accuracy.

From Q2, we got the RRN model (trained using symbolic data with the help of KMeans) which gave:
14% 8-level accuracy and **0%** board-level accuracy.

From Q2, we got the RRN model (trained using symbolic data with the help of (UDA) which gave:

40% 8-level accuracy and **2.9%** board-level accuracy.

Now we train both the RRN and the classifier in a joint manner, such that the rules learnt by the RRN (using the noisy symbolic data) helps the classifier to improve, and consequently the RRN improves.

Architecture details:

1. The symbolic dataset (query and target) given to RRN for its training is given by the classifier at every step/epoch of learning.
2. The classifier outputs softmax probabilities for each cell, i.e., the output of the classifier is (batch_size,8,8,9). (since we have 9 possible digits in sudoku table)
3. Now instead of passing digits to RRN, i.e., input of shape (batch_size,8,8) (like in Q2), we pass input of shape (batch_size,8,8,9). This means we are passing a 9 dimensional embedding for each digit which is basically soft-argmax. It has another interpretation as well that we replaced one-hot embedding with soft-argmax embedding.
4. We also **add an inductive bias** in the RRN model.
    a. From the visualization of digit 4 and 7, we see that there is a high possibility of the classifier confusing the two. We also saw this experimentally that the classifier is indeed confusing sometime between the two.
    b. Thus we add a fully connected layer in the RRN which allows the model to change the digit to any other digit if the current digit is 4 or 7.
    c. This bias gave us a slight improvement in the performance as we report in the results section.
5. The classifier outputs both the query and target symbolic data for the RRN.
6. We use cross entropy loss at the RRN output.

Joint training:

1. We first train the RRN model for 5-10 epochs and keep the classifier frozen. This allows the RRN to attain Q2's performance i.e. 14% 8-level accuracy.

2. Then we let the classifier fine tune that receive the gradients from both ends of the RRN. This is a crucial step as the classifier now receives the information of rules/constraints of sudoku.
3. Allowing both the RRN and classifier training for further 50 epochs, we see a significant improvement in both RRN and classifier, reported in the next section.

Results:(on validation set after joint training**)**

Experiment	Rnn 8 level accuracy (%)	Rnn board level accuracy(%)	Classifier 8 level accuracy (%)	Classifier board level accuracy(%)
Simple RRN	80.26	39.32	79.24	29.29
RRN using Inductive Bias	**89.23**	**45.12**	**82.24**	**33.34**

1. *Inductive bias of having a linear layer in RRN to allow the change from 7 to 4; without the inductive bias we saw drop in 10% acc (i.e. 80%) as compared to with inductive bias (89%)*

Minute training details:

1. We reduced the learning rate for the classifier as the gradients were of high magnitude, and the loss associated with the classifier (and its accuracy) went down hill.
2. The RRN learns pretty fast in few early epochs, so we didn't train it for more than 15 epochs (while classifier kept frozen). There was not much difference in performance if early warm up of RRN is done for 5 or 15 epochs.

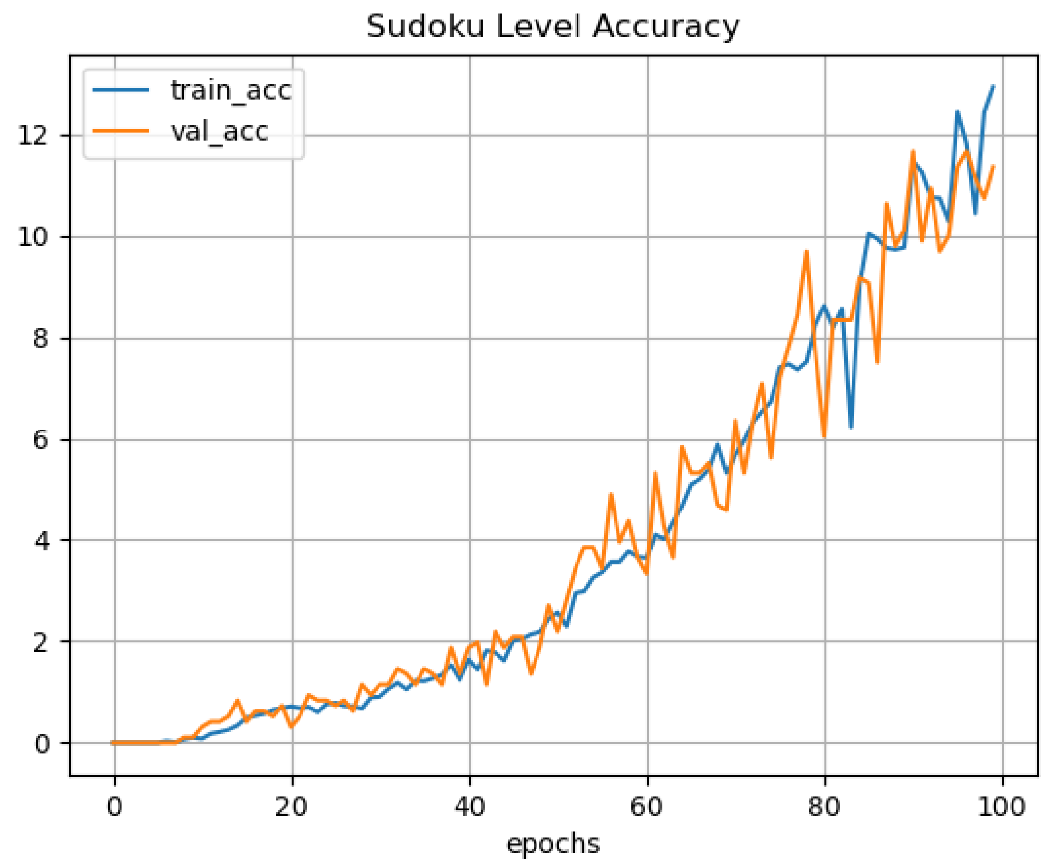
We trained the GAN again using our improved classifier after joint training and saw that our FID improved to 43.21.

Final scores for our trained GAN’s are:

Clustering method	Epochs trained for	FID
minibatch-KMeans	150	49.981
UDA	150	45.37
Improved UDA classifier after joint training	**150**	**43.21**

Appendix

Typical training plot (incomplete due to lack of compute resources):



The train_acc and val_acc are representing how many boards output by RRN , match the output (even if outputs and inputs are noisy)