CS 763/CS 764

COMPUTER VISION

REFLECTION ESSAY
LAB 08

Submitted by

Group 05

193079001, 193079005, 193079015

Contents

List of Tables

1 Training Details

1.1 Dataloader:

MNIST dataset is downloaded from `torchvision.datasets`. Dataloader from `torch.utils.data` is used and the tranforms applied are normalization transforms (mean = 0.5 and standard deviation = 0.5) We have used a batchsize of 128 with random shuffling for training the model.
The length of train set = 60000, length of test set = 10000. Dimensions of each image = (28,28).

1.2 Model:

The model for DCGAN consists of two modules.

1. Generator - The generator takes a noise vector of dimension 100, and outputs a 1x28x28 image. The noise vector first passes through a linear layer and outputs feature dimension of size (256*7*7) which is then reshaped to a tensor of dimension 256x7x7 where 256 is along the feature dimension. This is passed through a [conv layer,batch norm and leaky relu] followed by [two upconv,batch norm and leaky relu] and a tanh layer resulting finally into a image of dimension 1x28x28

2. Discriminator - This block consists of two 2D convolutional layers with `LeakyReLU()` as activation function and a 2D Batchnormalization layer, followed by a Linear layer with `Sigmoid()` activation.
   This block takes in the (1,28,28) image as input and outputs a number which is the probability of the input being a real image or a fake one.

The model for CWGAN consists of similarly two modules:

1. Generator - The generator takes a noise vector of size 100, appends a one hot encoded vector of labels to it, and produces a 28x28 image. The noise vector passes through a linear layer which takes the input vector of dimension 100+10, and outputs a feature size 256. This then passes through a LeakyReLu activation function. Then output of this passes through two similar linear + LeakyReLu layers to take the feature size from 256 to 512, and from 512 to 1024. Finally this feature of size 1024 passes through linear layer with output of size 28*28 = 784. The output shape is then changed to 1x28x28.

2. Discriminator - It takes the input of image of size 1x28x28, which is flattened in vector of size 784, and appended with one hot encoded vector of size 10.. This is then passed through combination of three [linear layers and LeakyReLu and Batchnorm], which takes the image to feature map size of 1024, then to 512, then to 256. Finally the output linear layers takes the 256 feature size to output of dimension 1 (fake/real).

1.3 Loss functions:

The loss functions used are exactly as asked in the task:

1. Binary cross entropy loss for both generator and discriminator training.

2. Wasserstein or Earth Movers distance as mentioned in the task. It takes the output of discriminator model for real and fake images and computes generator and discriminator loss as shown in task, and also adds the L2 norm of the gradients instead of direct weight clipping.

1.4 Training script:

- Optimizer - Adam optimizer is used for both generator and the discriminator.
- Input - Inputs to the Discriminator are (28,28) images, while the input to the Generator is a noise vector of size (B,100), where B is the batch size.
- Output - Output of the Generator is a (28,28) image and the Discriminator output is a probability value.
- Loss Criterion - Binary Cross Entropy loss, `BCELoss()` from `nn` module is used.
- Method - The Generator tries to generate images similar to the ones in the MNIST dataset to make the Discriminator believe them to be real images, while the Dicriminator tries to identify whether the image is real or fake. The training of these two blocks is sequential.
  - Discriminator Training - First the Discriminator is fed with real images and the `loss_real` is found. Next Discrimiator is fed with fake samples and `loss_fake` is found. The total loss, `loss_d` is the sum of these two losses for the DCGAN model, where these losses are binary cross entropy losses. For the CWGAN model it also includes a

L2 norm of the gradients, along with the `loss_real` and `loss_fake`, where these losses are earth movers distance or in too simple words (real - fake) kind loss. This loss_d is backpropagated to the Discriminator block to learn the Discriminator weights.

- Generator Training - Generator is fed with a (B,100) noisy vector to generate fake images. These images are fed to Discriminator and the prediction loss, `loss_g` is backpropagated to update the Generator weights.

2 Questions asked

Q.1 Dimensionality of latent space: There are two spaces refered in GANs, latent space corresponding to the low dimensional input space to the generator, and the sample space corresponding to high dimensional output space (image here). The dimensionality basically means how much degree of freedom we are giving to any space, for eg, 2D space has 2 dimensionality, which can be x y, so we can cover any point using these x y. Similarly in GANs, if we try to make a object which has lets say N dimensions (intuition can be nose, mouth, eyes, such), then we need atleast N dimensions in latent space, or more. More the dimensionality of latent space more it can capture the complexity/shape/distribution of sample space, but with too high dimensionality it is observed that mode collapse happens, leaving one or two objects out of possible outputs. So having dimensionality equals to 100 in latent space is needed, as that accounts for more complexity capturing/changing in the sample space. The reference [1] was used for understanding the concept behind dimensionality.

Q.2 Probability distribution of output images generated by GAN: The generator tries to learn the mapping of a random noise vector (latent space) to a output image (sample space). So each dimension of latent space will lead to some specific change in output image. But the value of each dimension is taken randomly with uniform distribution, so the classes of output should be seen with nearly equal probability, but it would be seen given a very large set of output images. We also tried to run the generate function multiple times to verify this, and it gave almost all output classes with equal probabilities.

One possible problem here could have been mode collapse, wherein the generator only gives few subset of images as its output. But if that was the case, we

wouldn't have seen all the classes in the output, which we are able to see. Another possible problem might have been if the real dataset was class imbalanced, but that is not the case here, so the learning is uniform in all classes.

Q.3 Original GAN problems: Problems of original GAN are:

1.) Partial Mode Collapse: The generator collapses and provides only output images of certain classes, totally avoiding other classes. This happens generally during early training stage, because when Discriminator model training is paused while generator learns, generator might stumble upon a class output where the discriminator is not able to classify the output of generator as fake, hence the generator easily fools the discriminator. Then the discriminators best bet is to learn to always reject that output. But if the next iteration discriminator gets stuck in local minimum and hence not able to learn, then its too easy for the next generator iteration to find the most easy output for current discriminator as the same class.

2.) Vanishing gradients: If the discriminator is too strong, then the loss function in original gan for generator falls to zero (D(G(z)) =0 and D(x) =1, and generator loss function being E(log(D(x))) + E(log(1-D(G(z))))) and there is no gradient update. Hence generator is not able to learn at all, or learn slowly.

3.) Convergence: As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction. This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

Q.4 Real World Business Problem: We can use it for two purposes:

1.) If working on a ML project, lets say on image dataset, where we don't have much dataset to work on, or where there is a class imbalance of dataset, then GANs can be very useful for augmenting our data, as it provides us with many more samples within the same distribution of sample space, which could help us learn the feature space better with more variation.

2.) We can create artificial data using the generator which will be very similar to real world data, such as human faces, or images of objects like furniture's.

For an application like a gaming environment, where artificial faces are created, or any such application which needs replication of data similar to original one, GANs can be very useful.

Q.5 Process of conditioning The conditioning refers to training of generator and discriminator under a condition i.e. along with input there is also a condition applied. Here the condition is the class label(0-9), so the generator along with noise vector takes one hot encoded class label as input and discriminator along with image takes one hot encoded class label as input. For generator the noise vector dimension is 100 and class label of dimension 10 here is concatenated and passed to generator model. For discriminator the images is flattened from 28x28 to 784 and class label is concatenated along this dimension.

3 Links to files:

Google colab link for DCGAN experiment
Google colab link for WCGAN experiment
Model checkpoint for vanilla GAN
Model checkpoint for WCGAN

4 Results:

We got the following final result images from the generator model at the end of our training. The best way to evaluate GANs is through qualitative evaluation.

1.) The MNIST dataset trained DCGAN:

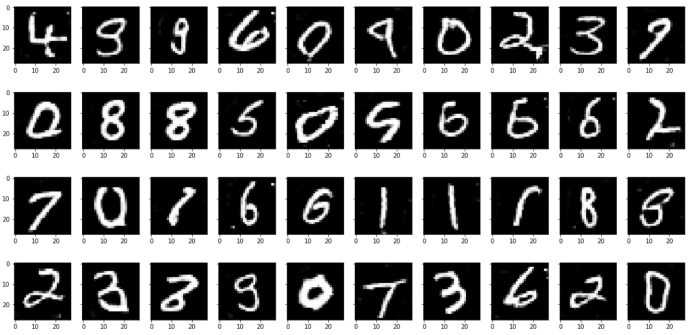

Figure 1: MNIST trained DCGAN results

2.) The Fashion MNIST dataset trained CWGAN:

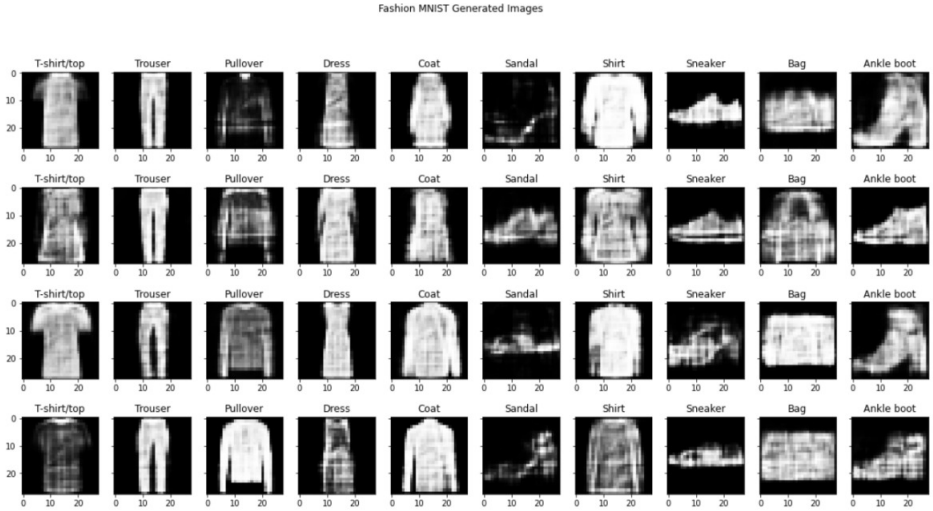

Figure 2: Fashion MNIST trained CWGAN

5 Conclusion

We were able to successfully create images of handwritten numbers from 0-9, using DCGAN and succesfully create images of fashion wear using conditional GAN. Using DCGAN we got random output of classes, and using conditional GAN we got output of a particular class by mentioning the class label.

6 References

[1] Dimensionality of latent space

[2] MNIST-GAN detailed explanation and implementation

[3] GAN for generating MNIST handwritten digits

[4] Generative Adversarial Network