Practical 3. Transformers and Deep Generative Models

University of Amsterdam – Deep Learning Course

December 10, 2021

This part of the assignment is not graded.

Due to time constraints, assignment 3 was reduced to only 50 points. Nonetheless, we do not take away parts of the assignment that we believe can still be helpful for understanding deep generative models. Thus, we include part 3 of the assignment on Generative Adversarial Networks (GAN) here, which is **not graded**. Instead, your TA will discuss this part of the assignment with you during the TA session on Monday, Dec 13, and we encourage you to take a look at it beforehand.

3 Generative Adversarial Networks **(not graded)**

Generative Adversarial Networks (GANs) are another type of deep generative models. Similar to VAEs, GANs can generate images that mimic images from the dataset by sampling an encoding from a noise distribution. In contrast to VAEs, in vanilla GANs there is no inference mechanism to determine an encoding or latent vector that corresponds to a given data point (or image). Figure 1 shows a schematic overview of a GAN. A GAN consists of two separate networks (i.e., there is no parameter sharing or the like) called the generator and the discriminator. Training a GAN leverages an adversarial training scheme. In short, that means that instead of defining a loss function by hand (e.g., cross-entropy or mean squared error), we train a network that acts as a loss function. In the case of a GAN this network is trained to discriminate between real images and fake (or generated) images, hence the name discriminator. The discriminator (together with the training data) then serves as a loss function for our generator network that will learn to generate images to are similar to those in the training set. Both the generator and discriminator are trained jointly. In this assignment, we will focus on obtaining a conditional generative adversarial network (cGAN) that can generate images that are similar to those in the training set yet the output is condition on a given parameter.

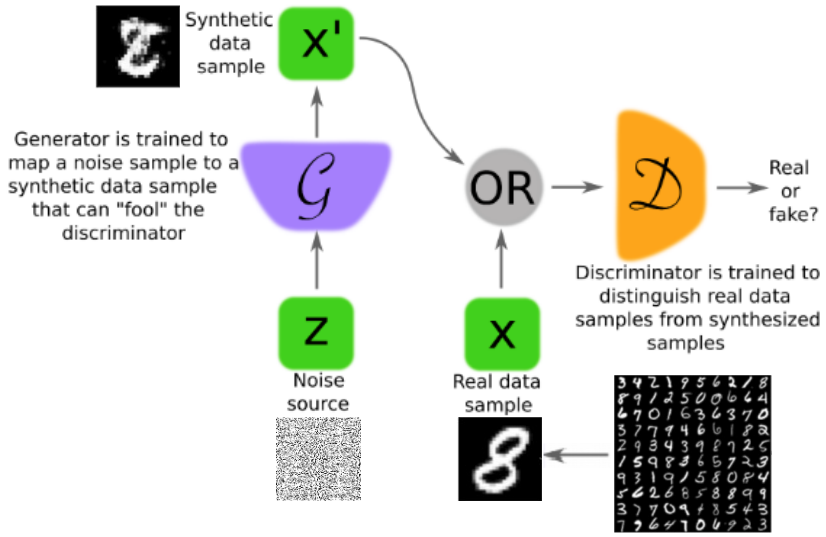


Figure 1. Schematic overview of a Generative Adversarial Network (GAN) [Creswell et al., 2018] on MNIST data. There two networks, generator $\mathcal{G}$ and discriminator $\mathcal{D}$, are learned during optimization for a generative adversarial network. While the generator samples new images, the discriminator has to distinguish between generated images and the real dataset.

3.1 Conditional GAN

Traditional or *vanilla* GANs are unconditional. For this reason [Mirza and Osindero, 2014] proposed a conditional network architecture where both the generator and the discriminator are conditioned on some extra information $\mathbf{y}$. A simple conditioning can be achieved by adding the class label, for example y is the target for the image to be generated: set y=3 if the goal is to only generate 3s from the MNIST data set. More complex conditioning such as other modalities is also possible for further details see [Mirza and Osindero, 2014, Reed et al., 2016].

To obtain a cGAN network architecture the *vanilla* GAN architecture is adjusted by creating a joint representation of the input($\mathbf{x}$) or noise($\mathbf{x}$) and the label $\mathbf{y}$, see Figure 2. In particular, for the generator the prior input noise $p_z(\mathbf{z})$ and $\mathbf{y}$ are combined in a joint hidden representation and there is quite some flexibility in how this hidden representation

is composed. Similarly, for the discriminator now the real or fake input is concatenated with the underlying conditioning $\mathbf{y}$ as well.

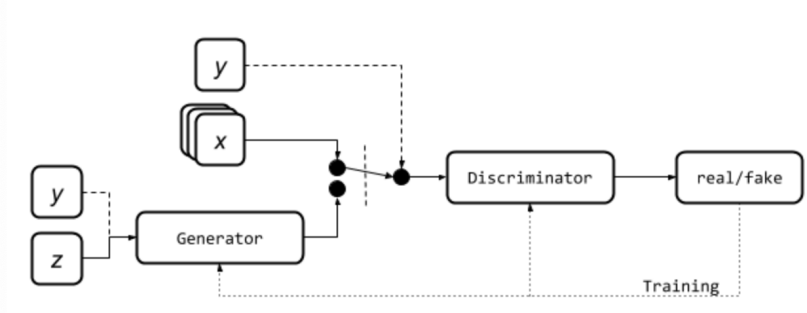


Figure 2. Schematic overview of Conditional Generative Adversarial Network (cGAN) [Mirza and Osindero, 2014] by F. Rodriguez. Both networks, generator $\mathcal{G}$ and discriminator $\mathcal{D}$, are conditioned on a class label y, where x is the real data and z is the latent space.

3.2 Training objective: A Minimax Game

The training objective for cGAN is equivalent to that of *vanilla* GAN: a two-player minimax game. The only difference is that the distributions are now conditioned on $\mathbf{y}$. In our experiments, the prior noise distribution of $p(\boldsymbol{z})$ will be a standard normal distribution.

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(\boldsymbol{x},\mathbf{y})}[\log D(\boldsymbol{x}|\mathbf{y})] + \mathbb{E}_{p_{\boldsymbol{z}}(\boldsymbol{z})p_{\text{data}}(\mathbf{y})}[\log(1 - D(G(\boldsymbol{z}|\mathbf{y})|\mathbf{y}))] \tag{1}$$

Question 3.1 (6 points)

a) Explain the two terms in the GAN training objective defined in Equation 1 based on what part of the adversarial loss they represent.
b) What is the value of $V(D,G)$ after training has converged? Explain it shortly.

Question 3.2 (4 points)

Early on during training, the term $\log(1 - D(G(\boldsymbol{z}|\mathbf{y})|\mathbf{y}))$ can be problematic for training the GAN. Explain why this is the case and how it can be solved.

3.3 Building a cGAN

Now that the objective is specified, and it is clear how the generator and discriminator should behave, we are ready to implement a cGAN. In this part of the assignment, you will implement a cGAN in PyTorch. Training a cGAN includes two steps per iteration:

Generator In the first step, we randomly sample a latent vector $\boldsymbol{z} \in \mathbb{R}^{B \times d_z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ where B is the batch size and d_z the dimensionality of the latent space, as well as a class label $\tilde{y}$. Depending on the dataset, $\tilde{y}$ can be categorical, continuous, or even a more complex structure like an image itself. In our setting, $\tilde{y}$ represents a class label between 0 and 9. Both z and $\tilde{y}$ are embedded before being combined in a single tensor. This latent vector is forwarded through the generator to create $\boldsymbol{x}' = G(\boldsymbol{z}|\tilde{\mathbf{y}})$, and we can use x' to calculate the loss via the Discriminator (see Eq 1).

Discriminator In the second step, you forward a fake batch $\tilde{x}$ and a "real" batch x (the batch from the dataset) with the corresponding generated labels $\tilde{y}$ and "real" labels y through the discriminator. Based on the discriminator's output, you calculate the

gradients for D. Note that $\tilde{x}$ and x' can be the same in a training step, but can also be two different samples from G. In the implementation, we will use the latter version as it is less error-prone to implement (guidance is provided in the templates). It is also recommended using the solution of Question 3.2 to stabilize training.

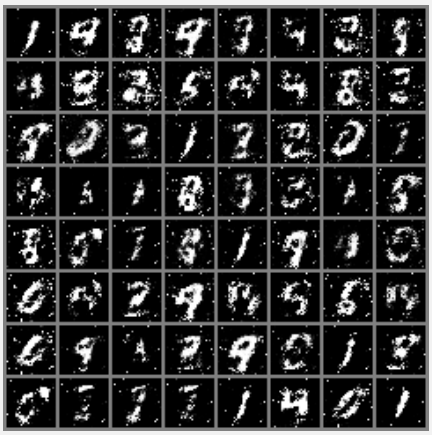
In the code directory `part3`, you find the templates to use for implementing the cGAN. Similarly to the VAE implementation, we provide a training loop template in PyTorch Lightning (`train_pl.py`), and one in plain PyTorch (`train_torch.py`). **You only need to implement one of the two training loop templates**. You also need to implement the generator and discriminator in the file `models.py`. We specified a recommended architecture to start with, but you are allowed to experiment with your own ideas for the models (see e.g. GAN hacks for tips and tricks on training strong GANs). For the sake of the assignment, it is sufficient to use the recommended architecture. Use the provided unittests to ensure the correctness of your generator and discriminator implementation. Read the provided README to become familiar with the code template.

Question 3.3 (12 points)

Implement a cGAN in the provided templates, and train it on the MNIST dataset. Provide a short description (no more than 10 lines) of the used architectures for generator and discriminator, and your training steps.

Additionally, sample 64 images (8×8 grid) from your trained GAN and include these in your report. Do this at the start of training, after 10 epochs, and after 250 epochs when training has terminated. Shortly describe the quality and differences over training.

Hint: You should already see some progress after the first 10 epochs of training. See example figure below. The expected runtime on Lisa (gpu) for complete training should be around 40 min.



After 10 epochs of training

Question 3.4 (3 points)

In cGANs we can sample images by conditioning them on a target we want to generate. For a trained model, sample a grid of images conditioned on the same, given target. Include 3 examples of the samples in your report here, explain how you obtained the result, and what variations you can observe among samples (3-4 sentences) (*See file evaluate.py*).

Question 3.5 (5 points)

In cGANs we can morph between 2 images by interpolating their latent noise and label vectors. For a trained model, sample 2 images randomly from your cGAN given 2 target labels and interpolate between these two digits in the latent space. Include 4 examples of the morphings in your report here, and explain what you see. Use 6 morhping steps, resulting in 8 images for each of the 4 examples (including start and end point)(*See file evaluate.py*).

References

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. IEEE Signal Processing Magazine, 35(1):53–65, 2018. 2

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014. 2, 3

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In International Conference on Machine Learning, pages 1060–1069. PMLR, 2016. 2