Generative Adversarial Networks for MNIST Digit Synthesis

Internship Project

Author: Pascual Merita Torres

December 29, 2023

0.1 General Project Statistics

- **Time Employed on Code:** 7 hours.
- **Time Employed on Report:** 1 hour.
- **Tasks Completed:**
  - Conditional GAN trained and tested successfully.
  - Attempted to modify code for standard GAN integration. Encountered a bug during training, leading to zero error in the discriminator and high error in the generator. Due to time constraints, this issue remains unresolved.
- **Remarks:** The focus was on demonstrating the successful implementation of the CGAN, utilizing best practices. The Encoder-Conditional GAN was not implemented due to time constraints. Future work will involve debugging and optimizing the standard GAN implementation apart from other potential improvements detailed in the next section.

0.2 Conditional GAN Implementation

0.2.1 Implementation Strategy

The code was not implemented in Jupyter Notebook or Google Colab, despite their preference stated in the project outline. The main reason for such decision is to enhance scalability and reusability. Large codebases in Jupyter Notebook tend to become unwieldy and prone to errors. The adopted approach aims to facilitate ease of use and maintainability.

0.2.2 Architecture Considerations

Convolutional layers were omitted as the focus was on Conditional GANs (CGANs), not Deep Convolutional GANs (DCGANs). We followed the architecture outlined in [1], which offers a conventional yet sensible structure.

0.2.3 Noise Distribution and Sampling

Experimental observations suggest the noise distribution's impact on performance is minimal. Therefore, a uniform distribution was chosen for simplicity, as it is straightforward to sample from.

Figure 1: Two-dimensional grid of digits from 0 to 9 generated by the CGAN model after 130 epochs of training. Each digit is represented by two distinct generated images, demonstrating the model's variety in generation.

0.2.4 Loss Function

Binary Cross Entropy Loss was employed for its simplicity. However, other alternatives could offer potential improvements:

- Hinge Loss may produce sharper images [2].
- Wasserstein Loss is known for its stability in training and addressing mode collapse in GANs [3]. Note that using Wasserstein Loss often entails removing the sigmoid activation in the discriminator's final layer. Additionally, the discriminator (or critic) in Wasserstein GANs should be a Lipschitz function, which is typically enforced through weight clipping or gradient penalty.

0.2.5 Analysis of CGAN Training and Performance

Figure 2 illustrates the training loss curves for both components of the CGAN: the discriminator and the generator. It is observed that convergence is achieved for both components, with marginal improvements post epoch 60. This indicates a stabilization in the learning process of the model. Furthermore, Figure 1 showcases the generative capabilities of the trained model using the generator obtained after epoch 130.

0.2.6 Overfitting Mitigation and Further Improvements

Several strategies could be adopted to control overfitting and enhance model performance:

- Incorporating validation data during training, which would involve processing the validation set at the end of each epoch.
- Implementing additional training metrics such as discriminator loss on both fake and real data, which could be instrumental in selecting the best-performing model and reducing overfitting.

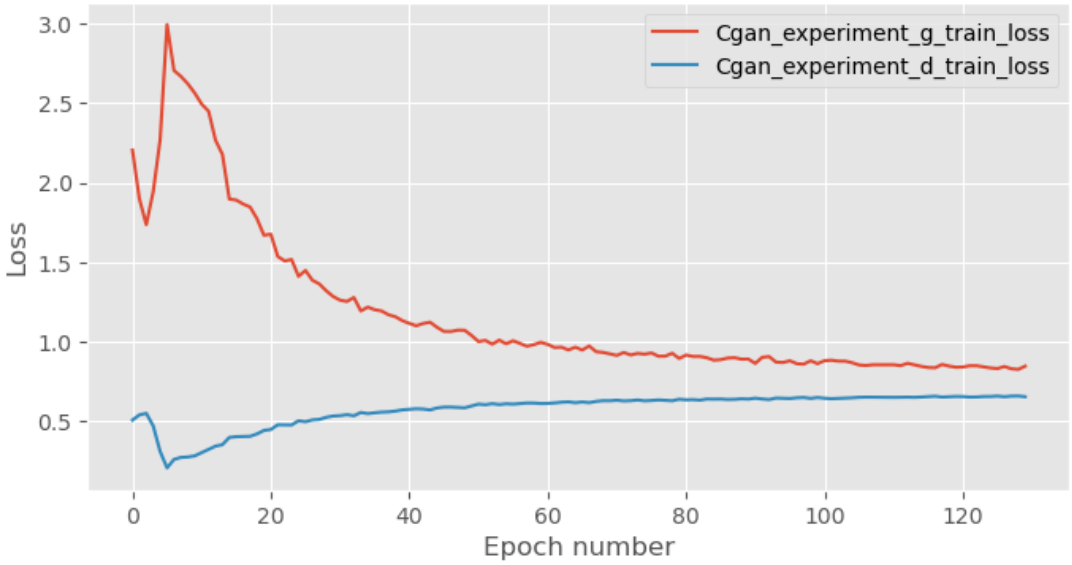

Figure 2: Training loss curves for the CGAN model. *g_train_loss* represents the training loss of the generator, while *d_train_loss* indicates the training loss of the discriminator. The graph shows the trajectory of both losses over the training epochs, highlighting points of significant change and convergence.

- Utilizing data augmentation techniques.
- Conducting more extensive hyperparameter optimization, like grid-search for the optimizers' learning rates.

References

[1] N. Varshney, "Step by step implementation of conditional generative adversarial networks," Dec 2020. [Online]. Available: https://medium.com/analytics-vidhya/step-by-step-implementation-of-conditional-generative-adversarial-networks-54e4b47497d6

[2] J. H. Lim and J. C. Ye, "Geometric gan," 2017.

[3] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.