Adversarial Autoencoders in a Semi-Supervised Setting

Franciszek Latała
Delft University of Technology

1 INTRODUCTION

Learning generative models of complex data is a central challenge in machine learning, enabling generation of realistic new data and discovery of latent representations.

Adversarial Autoencoders (AAEs) [6] are generative models that combine autoencoders with adversarial training to impose a specific prior distribution on the latent codes. This allows for matching very complex priors and learning smooth, structured latent representations while maintaining high reconstruction quality. Introduced in 2015, AAEs were an early attempt to integrate adversarial training into latent variable models. The motivation was to enable models to generate high-quality samples from complex priors and to disentangle structured latent features for tasks such as semi-supervised learning, clustering, and dimensionality reduction, without requiring explicit likelihoods or handcrafted priors.

As a group, we found the idea of using adversarial training to shape the latent space to be an interesting and innovative approach. We also appreciated how the original paper provides comprehensive evaluations covering likelihood estimation, supervised and semi-supervised learning, clustering, and dimensionality reduction. However, the paper lacks some details in terms of hyperparameter and training dynamics. The goal of our project was to reproduce the results achieved in the experiments, to confirm the claimed performance. I pursued reproduction of the semi-supervised experiments - I found that area to be particularly interesting because the evaluation in the paper only focuses on classification performance in that setting, thus leaving space for additional analysis.

2 BACKGROUND

In this section the background knowledge relevant for understanding of Adversarial Autoencoders is introduced. Firstly, latent variable models are discussed, later their parametrization in VAEs is explained, followed by GANs and adversarial training.

Latent Variable Models introduce hidden variables z to explain observed data x, modeling $p(x, z) = p(x|z)p(z)$. These models help uncover underlying structures in data, enabling tasks such as dimensionality reduction, clustering, and generative modeling [1]. For example, latent factors can capture topics in text or styles in images. They model the data likelihood as:

$$p(x) = \int p(x|z)p(z)dz \tag{1}$$

In this formulation the integral is typically intractable in complex models (without conjugate priors), making the true posterior $p(z|x)$ hard to compute. To address this, approximations are used, such as variational inference [2], Markov Chain Monte Carlo [9] methods, or parametric approaches as VAEs.

Variational Autoencoders (VAEs) [4] approximate the true posterior $p(z|x)$ with a parametric function $q_\phi(z|x)$, learned by the encoder. The decoder models $p_\theta(x|z)$, the posterior of data given latent representation. This enables efficient, end-to-end learning of both functions, as an alternative to MCMC approximations. Since the data likelihood is intractable, the variational lower bound of $\log p(x)$ is maximized:

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|p(z)) \tag{2}$$

The expectation over the log of $p_\theta(z|x)$ (reconstruction term) is calculated empirically using samples generated from $q_\phi(z|x)$ - a reparameterization trick makes it differentiable. For most common priors, the KL-term can be computed in a closed form.

VAEs have some shortcomings, as they require an explicit form for the assumed latent prior $p(z)$ and they suffer from posterior collapse where the latent representation becomes non-informative. Adversarial Autoencoders (AAEs) attempt to overcome both of these shortcomings by leveraging adversarial training from GANs.

Generative Adversarial Networks (GANs) [3] are used for generating new data samples from learned distributions, often applied in image synthesis, text generation, and data augmentation. GANs consist of two networks: a generator $G(z)$ and a discriminator $D(x)$. The generator G maps latent variables $z \sim p_z(z)$ to data space, aiming to create samples resembling real data. The discriminator D learns to distinguish real data from generated data. Training involves a minimax game:

$$\min_G \max_D \mathbb{E}_{x \sim p_d(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3}$$

This setup encourages G to generate data indistinguishable from real data, while D improves its discrimination ability. GANs are powerful for generating sharp samples but can be unstable to train due to mode collapse and convergence issues.

Adversarial Autoencoders (AAEs) [6] combine the latent structure learning from VAEs with the adversarial training strategy of GANs. In AAEs, the encoder maps data x to a latent representation z, similar to a VAE, and the decoder reconstructs x from z. However, instead of enforcing the prior distribution $p(z)$ with the KL-Divergence term (as in Equation 2), AAEs use a discriminator $D(z)$ to distinguish between true prior samples and encoded latent representations. The encoder $q_\phi(z|x)$ acts as the generator in a GAN, trying to fool $D(z)$ into believing that its latent codes come from $p(z)$. The adversarial objective thus becomes:

$$\min_{q_\phi} \max_D \mathbb{E}_{z \sim p(z)}[\log D(z)] + \mathbb{E}_{z \sim q_\phi(z|x)}[\log(1 - D(z))] \tag{4}$$

This encourages the aggregated posterior $q(z) = \mathbb{E}_x[q_\phi(z|x)]$ to match the prior $p(z)$, ensuring that latent representations conform to a desired distribution. Reconstruction is optimized via standard autoencoder loss (e.g., mean squared error or cross-entropy). By decoupling the choice of prior from explicit likelihood assumptions, AAEs mitigate posterior collapse and allow for flexible, often sharper, generative modeling. Moreover, knowing the closed form for the latent prior $p(z)$ is not required, as the KL-term is gone.

3 METHODOLOGY

In my experiments I evaluated the adversarial autoencoder architecture in a semi-supervised setting. This section explains the semi-supervised approach and introduces the experiments that were carried out.

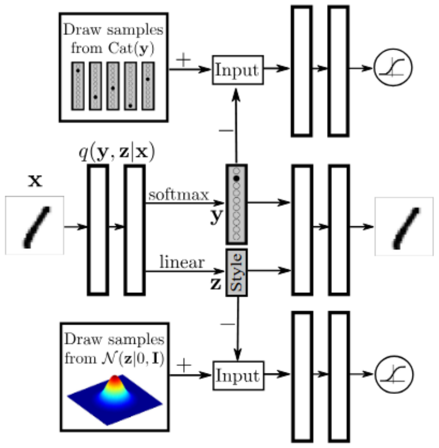


Figure 1: Architecture diagram of the semi-supervised AAE [6].

Semi-Supervised Approach – In the semi-supervised setting the encoder maps the input data x into two parts in the latent space: a categorical vector y, representing class information, and a continuous variable z, capturing style or other variations. In the first stage of training the reconstruction error of the is optimized – encoder and decoder are updated. In the second stage two discriminators are trained adversarially – $D(z)$ to align the aggregated posterior of z with the chosen prior distribution $p(z)$ and $D(y)$ to enforce a categorical distribution over the vector y. The encoder $q_\phi(y, z|x)$ is then updated to fool the discriminators. Finally, the encoder is trained with a small sample of labeled data to predict class labels from y - CrossEntropy loss is used. This setup enables the model to learn meaningful latent structures and improves generalization with limited labeled data.

Code Reproduction – We reproduced the model in Python with PyTorch 2.6.0 – our code can be found in a GitHub repository[1]. The implementation of the semi-supervised architecture closely follows the original paper. The encoder, decoder, and discriminator networks each have two hidden layers of 1000 units with ReLU activation. The encoder outputs both $q_\phi(y|x)$, a 10×1 categorical vector with softmax activation, and $q_\phi(z|x)$ – a 10×1 style vector with linear activation. The decoder reconstructs inputs from both latent codes. Reconstruction loss is squared Euclidean distance, semi-supervised classification uses cross-entropy. Gaussian and categorical priors are imposed on style and labels, respectively. The paper uses SGD with momentum, it doesn't specify the number of optimizers. We used separate optimizers for reconstruction, classification, discriminators, and encoder adversarial training. To prevent the unsupervised data samples from dominating the supervised ones, I matched their batch counts while rotating unlabeled data between batches – a detail based on other reproductions online.

[1]https://github.com/1henrypage/adversarial-autoencoders

Classification Accuracy Experiment – This experiment aimed to reproduce Table 2 from the original paper by training the model semi-supervised on MNIST [5] and evaluating classification accuracy of predicted latent codes y. I tested three scenarios: using 100 labeled samples, 1000 labeled samples, and all labeled samples. To account for variance, all the experiments were repeated three times at both 50 and 500 epochs. The exact same hyper-parameters as in the original paper were used (Table 1).

Table 1: Hyperparameters for the classification accuracy experiment.

Parameter	Value
Optimizer	SGD with momentum
Momentum	AE/Semi-sup: 0.9; Adv: 0.1
LR (Reconstruction)	0.01 → 0.001 (at 50 epochs) → 0.0001 (at 1000 epochs)
LR (Adv / Semi-sup)	0.1 → 0.01 (at 50 epochs) → 0.001 (at 1000 epochs)
Noise	Gaussian $\sigma = 0.3$ at input (during training only)
Style prior	Gaussian $\mathcal{N}(0, 1)$

Disentanglement Experiment – The original paper did not evaluate disentanglement in the semi-supervised setting. Since classification accuracy only reflects encoder performance, I generated samples from the latent space to assess style representation and decoder performance. Initially, I used the models from the first experiment, but poor results led me to additionally evaluate models trained with adjusted hyperparameters (Table 2), to test if effective disentanglement and generation were achievable. This involved runs with 1000 labeled samples at 1000 epochs and 2000 labeled samples at 2000 epochs.

Table 2: Hyperparameters for the disentanglement experiment.

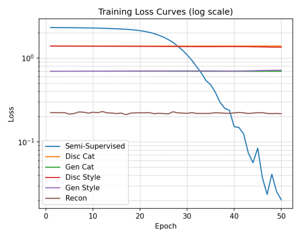
Parameter	Value
Optimizer	AdamW
LR (Same for all optimizers)	0.001 → 0.0001 (at 50 epochs) → 0.00001 (at 1000 epochs)
Noise	Gaussian $\sigma = 0.3$ at input (during training only)
Style prior	Gaussian $\mathcal{N}(0, 1)$

4 RESULTS AND DISCUSSION

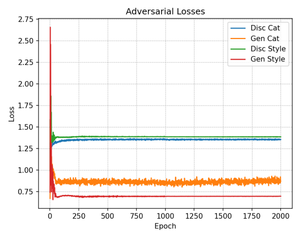
The classification experiment results are shown in Table 3. The first three columns specify the model, the number of labeled samples used for training, as well as the source of the result (mine vs paper), while the last three columns present results at different epochs and the best reported values.

Table 3: Classification accuracy (%) on MNIST depending on the number of labeled samples used for training and epoch count. Table includes my reproduction results, the results claimed in the paper, as well as VAT [8] and NN baselines.

Model	# Labeled Samples	Source	50 epochs	500 epochs	Best
NN Baseline	100	Paper	–	–	74.20
VAT Baseline	100	Paper	–	–	97.67
AAE semi-sup	100	Ours	76.60 ± 1.00	76.50 ± 2.00	76.50 ± 2.00
AAE semi-sup	100	Paper	96.60	97.80	98.10 ± 0.10
NN Baseline	1000	Paper	–	–	91.27
VAT Baseline	1000	Paper	–	–	98.64
AAE semi-sup	1000	Ours	92.30 ± 0.20	93.10 ± 0.10	93.10 ± 0.10
AAE semi-sup	1000	Paper	97.60	97.80	98.40 ± 0.08
NN Baseline	All	Paper	–	–	98.75
VAT Baseline	All	Paper	–	–	99.36 ± 0.04
AAE semi-sup	All	Ours	98.66 ± 0.10	98.96 ± 0.05	98.96 ± 0.05
AAE semi-sup	All	Paper	98.75	98.85	99.15 ± 0.02



(a) All learning curves for 100 labels at 50 epochs.



(b) Adversarial learning curves for 2000 labels at 2000 epochs.

Figure 2: Learning curves for two different training settings.

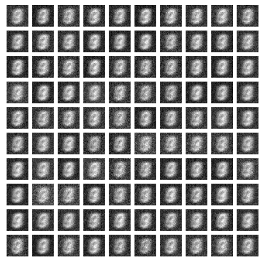
The reproduced results did not exactly match the original paper's claims, especially with only 100 labeled samples, where I got accuracies around 76%, while the paper claims above 96% for all epoch counts. When training with 1000 labeled samples they were much closer however (93% vs 98%), and with all samples labeled, the results were almost exactly matched to the ones stated in the paper, at around 99% accuracy. Overall, our implementation outperformed a neural network baseline in every evaluation setting, however the VAT [8] baseline was stronger

The weaker performance with 100 labeled samples likely stems from differences in training dynamics - the paper lacks detailed explanation as to how precisely were the different training stages interleaved and what were the proportions of unlabeled and labeled samples. As shown by Figure 2a, the semi-supervised loss dropped sharply within 50 epochs, suggesting overfitting to the small (100 data points) labeled set. This also explains why performance didn't improve after 500 epochs.

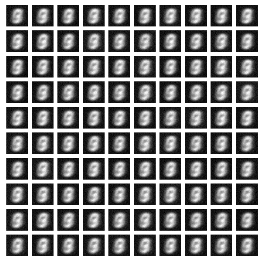
When investigating the disentanglement in the model variants used for the classification experiment the decoder generated nearly identical, noisy images for different digit codes y (from 0 to 9, as seen in Figure 3), indicating poor latent space regularization and insufficient decoder training. This was quite similar in both the 100 and 1000 label scenarios, however at 500 epochs and 1000 labels the images did get less noisy, as shown in Figure 3b. This might be due the decoder being better fitted after more epochs elapsed.

As specified in section 3, to further investigate disentanglement in a semi-supervised setting, I ran further experiments with various parameters. The best results came from training with 2000 labeled samples for 2000 epochs using AdamW. Figure 4a shows that each latent code reliably produces its target digit, while Figure 4b shows how different latent vectors z impact the style - the representations were selected from a 2.0 by 2.0 grid centered at the origin of the probability space. One can see that moving horizontally corresponds to a tilt in the generated digit while moving vertically affects how squeezed the image is. The generation results with SGD momentum (as used in the paper) were significantly weaker - the best ones achieved are shown in Figure 5 (Appendix A).

When training for 2000 epochs, the adversarial losses decreased and then oscillated around the same level, which is shown in Figure 2b. This was true for both the categorical and style adversarial losses and is also consistent with GAN training loss patterns in practice [7]. Combining this observation with the improved latent representations, as demonstrated by Figure 4, one can conclude that disentanglement in AAEs is also achievable in semi-supervised training scenarios.

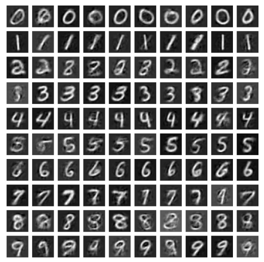


(a) 50 epochs

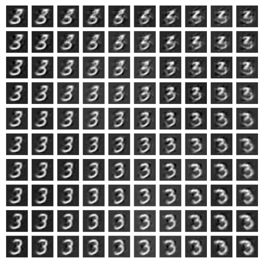


(b) 500 epochs

Figure 3: Images generated using the 1000 label model by fixing code y and randomly sampling style z from $\mathcal{N}(0, 1)$.



(a) Digits generated with z sampled from $\mathcal{N}(0, 1)$.



(b) Digit 3 generated by selecting style z from a 2.0×2.0 grid.

Figure 4: Digit images generated by sampling from the model trained with 2000 labeled samples for 2000 epochs.

Summarizing the results one can state that the semi-supervised AAEs performed worse than claimed by the paper in terms of classification performance, especially with only 100 labeled samples, but further experiments demonstrated that good generation and disentanglement performance are achievable even with limited labeled samples.

5 CONCLUSION

During the project, our group faithfully reproduced all experiments from the original Adversarial Autoencoder paper. I specifically focused on implementing and evaluating the semi-supervised experiments. The obtained classification accuracies, while better than a typical neural network, fell short of the ones claimed by the authors of the Adversarial Autoencoders paper. Nevertheless, the additional experiments demonstrated that achieving well regularized latent representations and good image generation performance are possible even in the semi-supervised scenario. Finally, it is worth noting that the performance gaps could stem from differences in training dynamics or optimizer setup, as these were not very well described in the original paper. Despite this, the results still show that AAEs in a semi-supervised scenario deliver solid classification performance, real latent representation learning and regularization, and a good generative performance. All this makes AAEs quite an efficient and flexible model architecture.

Individual Contribution – I implemented the semi-supervised AAE code (specifically the `semisupervised.py` file in our repo), I conducted the classification experiments to try reproducing the Table 2 from the original paper, and finally I designed and executed the additional qualitative evaluation to assess generation and latent disentanglement in the semi-supervised setting.

REFERENCES

[1] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

[2] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational Inference: A Review for Statisticians. *J. Amer. Statist. Assoc.* 112, 518 (2017), 859–877.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. (2014). arXiv:stat.ML/1406.2661 https://arxiv.org/abs/1406.2661

[4] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. (2022). arXiv:stat.ML/1312.6114 https://arxiv.org/abs/1312.6114

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[6] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. 2016. Adversarial Autoencoders. (2016). arXiv:cs.LG/1511.05644 https://arxiv.org/abs/1511.05644

[7] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. 2018. The Numerics of GANs. (2018). arXiv:cs.LG/1705.10461 https://arxiv.org/abs/1705.10461

[8] Takeru Miyato, Shin ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. 2016. Distributional Smoothing with Virtual Adversarial Training. (2016). arXiv:stat.ML/1507.00677 https://arxiv.org/abs/1507.00677

[9] Radford M Neal. 1993. *Probabilistic Inference Using Markov Chain Monte Carlo Methods*. Technical Report CRG-TR-93-1. University of Toronto.

A APPENDIX

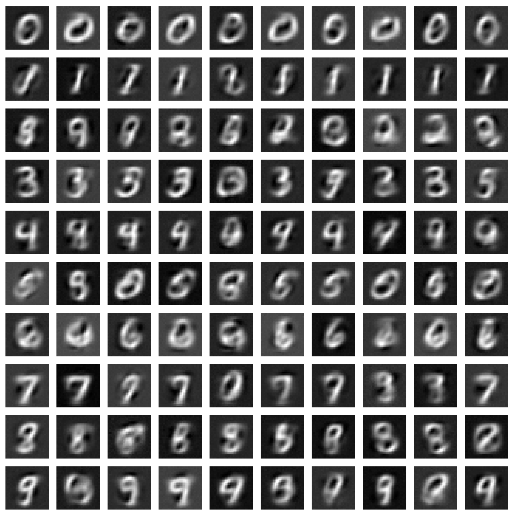


Figure 5: Digits generated from the model trained for 500 eepochs with the SGD Momentum optimizer using the labels for all the samples. Training time was longer than with 2000 semi-supervised epochs with 2000 samples, due to labeled / unlabeled batch-count matching in the second scenario.

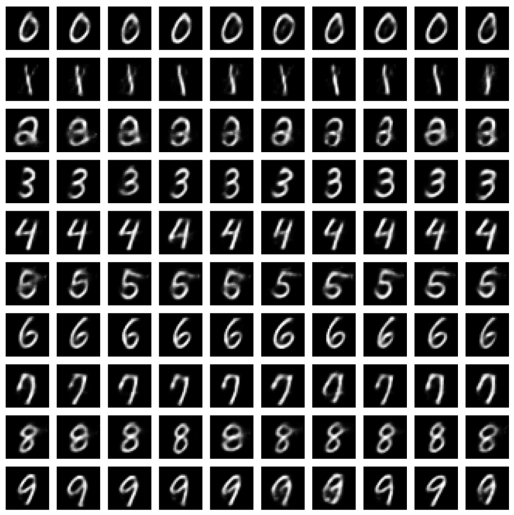


Figure 6: Digits generated from the model trained for a 1000 epochs with the AdamW optimizer using the labels for all the samples. Training time was above 2 hours. As this was a fully supervised scenario (all data was labeled) I didn't include it in the analysis, but it demonstrates that the generation performance can be improved even further. However, the style representation seems to be less informative here, as the samples are more similar to each other despite random sampling of the style code.