Learning Feature Representations - Project Module 1

Nils Mechtel

November 30, 2024

1 Introduction

Dataset distillation is a method aimed at condensing a large dataset into a much smaller synthetic set that retains the essential information for training a model. Unlike traditional methods that rely on labeled data, this project focuses on the unsupervised distillation of unlabelled data, where synthetic datasets are optimized to support downstream tasks effectively [Yu et al., 2023].

2 Project Goals and Methodology

The project's primary goal was to develop an intuitive understanding of dataset distillation by engaging with its training process. A shallow feature extractor was trained using proper scoring rules (PSR), with features passed to a task-specific head, a linear classifier in this case. The MNIST dataset, known for its simplicity and accessibility, was chosen for experimentation. Its 60,000 grayscale images of handwritten digits (28×28 pixels) were normalized to $[0, 1]$, with one-hot encoded labels. Training utilized 90% of the data, while 10% was set aside for validation.

3 Model Architecture and Implementation

The baseline provided in Julia comprised a shallow autoencoder and a linear classification head, along with synthetic dataset initialization and a four-term loss function. This baseline was reimplemented in Python using PyTorch and PyTorch-Lightning to facilitate modularity and streamline handling of the bilevel optimization process. A top-level script enabled efficient hyperparameter tuning, and training logs were visualized using Weights and Biases (wandb).

3.1 Model Components

- **Autoencoder (AE):** The AE encoder had one fully connected layer reducing the 784 input dimensions of an MNIST image (28×28 pixels) to 64 latent dimensions, followed by ReLU activation. The decoder reconstructed the input with a 64-to-784 layer and Sigmoid activation.
- **Classification Head:** A single fully connected layer mapped 64 latent dimensions to 10 one-hot encoded output dimensions.
- **Trainable Synthetic Dataset:** An initial synthetic dataset was randomly initialized with values in the range [0, 1]. It consisted of 80 samples, each represented as a flattened vector of 784 dimensions (corresponding to 28×28 pixel images). The dataset was optimized during training to encode the distilled knowledge of the original MNIST dataset.

3.2 Loss Function

The total loss function consisted of a combination of terms designed to balance different objectives in the dataset distillation process:

$$\begin{aligned}\mathcal{L}_{\text{total}} =& \lambda_{\text{classification}}\mathcal{L}_{\text{classification}} + \lambda_{\text{reconstruction}}\mathcal{L}_{\text{reconstruction}} \\ &+ \lambda_{\text{gradient}}\mathcal{L}_{\text{gradient}} - \lambda_{\text{diversity}}\mathcal{L}_{\text{diversity}}.\end{aligned}$$

- $\mathcal{L}_{\text{classification}}$**:** The cross-entropy loss was used to ensure that real data passed through the encoder and classification head produced accurate class predictions. This term guided the feature extractor to learn meaningful representations aligned with downstream tasks.

- $\mathcal{L}_{\textbf{reconstruction}}$**:** The mean squared error (MSE) measured the difference between the input and reconstructed synthetic data. Minimizing this loss encouraged the autoencoder to preserve the structure of the synthetic dataset.

- $\mathcal{L}_{\textbf{gradient}}$**:** This term penalized large gradients of the autoencoder with respect to the synthetic dataset, promoting smoothness and stability in the learned representations. Calculating this term required higher-order derivatives of the reconstruction loss with respect to the synthetic data. Using PyTorch's 'torch.autograd.grad' function with the argument 'create_graph=True' enabled the construction of a computational graph for these gradients, allowing backpropagation through higher-order derivatives.

- $\mathcal{L}_{\textbf{diversity}}$**:** To avoid mode collapse and encourage variation among the synthetic samples, this term maximized the diversity of the distilled data points by calculating the MSE between individual samples and their batch mean.

Each term in the total loss was scaled by a respective weight ($\lambda_{\text{classification}}$, $\lambda_{\text{reconstruction}}$, $\lambda_{\text{gradient}}$, $\lambda_{\text{diversity}}$) to adjust its relative contribution. Hyperparameter tuning was crucial to finding the optimal balance for these weights.

4 Hyperparameter Effects and Challenges

Hyperparameter scaling is crucial for balancing the contributions of the four loss terms, with small or excessive scaling leading to underperformance or distortions in the training process. Achieving the right balance requires trade-offs, as optimizing one term can conflict with others.

For example:

- **High Gradient Scaling:** Excessively scaling the gradient penalty term (e.g., 10^3) caused the synthetic dataset to stagnate, as large penalties discouraged meaningful updates, effectively freezing the optimization.

- **Overemphasized Diversity:** Overemphasizing the diversity term led to binary-like synthetic data (values clustering at 0 and 1). This behavior occurs because maximizing diversity mathematically favors extreme values, which are furthest from the batch mean.

To mitigate these issues, I implemented a strategy to gradually increase the weighting of both the diversity and reconstruction losses over the first few epochs. This allowed the model to learn meaningful features early in training without overly constraining the optimization. Adjusting the number of synthetic samples also required recalibrating loss term scaling factors, as higher sample counts potentially require lower scaling to maintain balance. These strategies highlight the complexity of tuning hyperparameters in bilevel optimization.

5 Results and Observations

The autoencoder pretraining was an essential first step. Early stopping was applied after 47 epochs, with consistent reconstruction loss reduction observed throughout the training process. Figure 1 illustrates the results of the autoencoder pretraining, showing the reconstruction of each class from the MNIST dataset. The original image is shown on top, while the reconstructed version is displayed below, demonstrating the model's ability to capture the essential features of the data.

Figure 1: Reconstruction of each class from the MNIST dataset after autoencoder pretraining. The original image is on the top and the reconstructed image is on the bottom.

After pretraining, dataset distillation was carried out, during which the training loss (Figure 2, left) and validation accuracy (Figure 2, right) were tracked. The training loss consisted of classification loss (yellow), reconstruction loss (green), diversity penalty (purple), and gradient

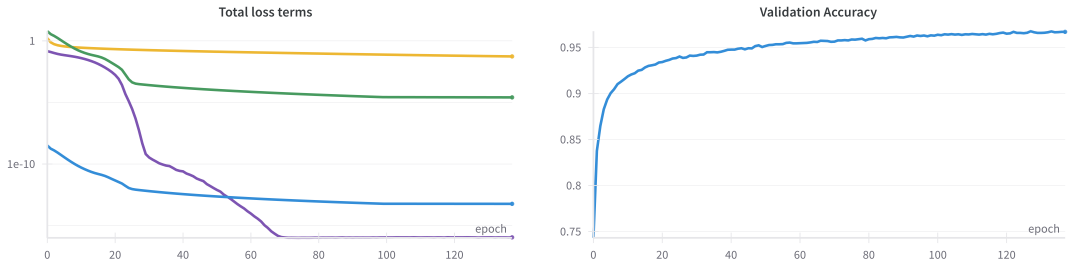

Figure 2: Training loss (left) and validation loss (right) during dataset distillation. The training loss is composed of classification loss (yellow), reconstruction (green), diversity penalty (purple) and gradient penalty (blue).

penalty (blue). The validation accuracy reached 96.7%, indicating that the distilled dataset was able to effectively support the classification task.

The classifier model's performance was further evaluated using a contingency table of ground truth and predicted classes (Figure 3). The heatmap, normalized by counts, demonstrates the classifier's effectiveness in distinguishing between MNIST classes, although with some limitations in diversity within the distilled dataset.

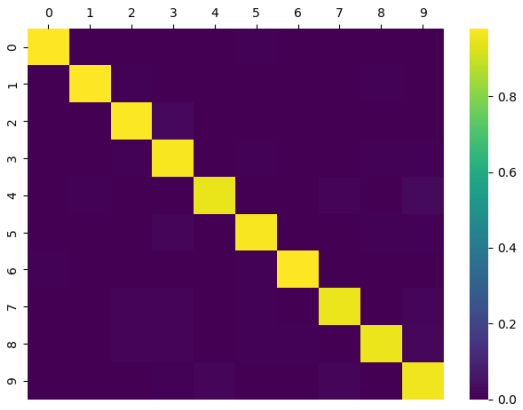

Figure 3: Contingency table of each ground truth class (row) and its predicted class (column). The heatmap is colored by normalized counts.

Despite the high classification accuracy, the distilled dataset exhibited significant homogeneity. To prevent this, the first approach included to prime each image in the synthetic dataset with 25% real MNIST images and 75% added noise. While this approach delayed the onset of homogeneity, it ultimately failed to prevent it. As shown in Figure 4, the distilled dataset became highly uniform by the end of training, with very little variation between samples. The bottom panel of the figure highlights a similar homogeneity in the reconstructions of real MNIST images, suggesting over-reliance on biases in the decoder.

To further address this issue, an autoencoder with shared encoder-decoder weights was tested. This seemed to effectively contribute to heterogeneity in the distilled data, but at the current stage of hyperparameter tuning, the reconstruction still resulted in suboptimal performance, with synthetic data often appearing binary in nature. The accuracy was not effected by the swap of feature extractors. Another approach involved using a Gaussian Restricted Boltzmann Machine (GRBM) with Negative Log-Likelihood (NLL) loss as the feature extractor. While this offered a different perspective on feature extraction, the reconstruction results were inferior to those produced by the autoencoder and were not pursued further.

Figure 4: The distilled dataset with 80 samples is shown on the top. The bottom panel displays the reconstruction of each class from the MNIST dataset after dataset distillation. The original image is on the top and the reconstructed image is on the bottom.

6 Summary and Outlook

This project provided a hands-on opportunity to explore dataset distillation using unsupervised techniques. Through implementation and experimentation, I gained valuable insights into the challenges and trade-offs involved in producing synthetic datasets that are both diverse and effective for downstream tasks.

Although the distilled dataset achieved high classification accuracy, the issue of synthetic data homogeneity highlighted areas for improvement. One potential approach to address this is sampling from both MNIST and synthetic data when calculating the reconstruction loss during data distillation. This could help prevent the model from overly relying on decoder biases by encouraging a balance between reconstructing real and synthetic data.

Further refinements could include testing alternative feature extractors and adjusting the loss function, for example, using the sum instead of the mean during loss reduction to automatically scale the loss with the number of features. This approach, inspired by tools like scVI [Lopez et al., 2018], could help better balance competing objectives such as reconstruction and diversity.

Overall, this project served as a foundation for understanding dataset distillation and highlighted the complexity of optimizing synthetic datasets. Future work will involve exploring these strategies and continuing to address the observed limitations.

7 Code Availability

The code can be found at `https://github.com/nilsmechtel/lfr-2024`.

References

[Lopez et al., 2018] Lopez, R., Regier, J., Cole, M. B., Jordan, M. I. and Yosef, N. (2018). Deep generative modeling for single-cell transcriptomics. Nat Methods *15*, 1053–1058.

[Yu et al., 2023] Yu, R., Liu, S. and Wang, X. (2023). Dataset Distillation: A Comprehensive Review. arXiv .