Building a Diffusion Model from Scratch on MNIST

Arepalli Om Satya Swaroop
CSYE7380 53375 Theory & Prac App AI Gen Model SEC 02

A Complete Implementation Guide

Page 1: Introduction & Project Overview

What is a Diffusion Model?

Diffusion models are state-of-the-art generative AI models that learn to create new data by reversing a gradual noising process. They work in two main phases:

Forward Process (Adding Noise):

- Gradually adds Gaussian noise to real images over T timesteps
- Transforms real data into pure noise following: $q(x_t \mid x_{t-1}) = N(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I)$

Reverse Process (Denoising):

- Learns to remove noise step by step to generate new samples
- Model learns: $p_\theta(x_{t-1} \mid x_t) = N(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$

Project Scope & Implementation

This notebook presents a complete from-scratch implementation featuring:

- **Dataset**: MNIST handwritten digits (28×28 grayscale images)
- **Architecture**: Custom U-Net with time embeddings and class conditioning
- **Training**: 60,000 samples with 10 epochs
- **Model Size**: 6,141,313 parameters
- **Capabilities**: Interactive digit generation, text prompts, conditional sampling

Key Advantages of Diffusion Models

- **Stable Training**: No adversarial training required (unlike GANs)
- **High Quality**: Produces very high-quality samples
- **Flexible Conditioning**: Easy to condition on classes or text prompts
- **Controllable Generation**: Supports guided and classifier-free guidance

Page 2: Architecture & Technical Implementation

Core Components

1. Noise Scheduler

```
class NoiseScheduler:
def __init__(self, timesteps=1000, beta_start=0.0001, beta_end=0.02):
self.betas = torch.linspace(beta_start, beta_end, timesteps)
self.alphas = 1.0 - self.betas
self.alphas_cumprod = torch.cumprod(self.alphas, dim=0)
```

- **Linear Schedule**: β values from 0.0001 to 0.02 over 1000 timesteps
- **Cumulative Products**: Enable direct sampling at any timestep
- **Forward Process**: $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon$

2. U-Net Architecture Components

Time Embeddings:

- Sinusoidal positional encodings for timestep information
- 128-dimensional embeddings expanded to 256 dimensions
- Critical for model to understand denoising step

Network Structure:

- **Input**: 28×28 grayscale images + timestep + class label
- **Encoder**: Downsampling path (28×28 → 14×14 → 7×7)
- **Bottleneck**: Feature processing at lowest resolution
- **Decoder**: Upsampling path with skip connections
- **Output**: Predicted noise at original resolution

3. Class Conditioning

- Embedding layer for 10 digit classes
- Combined with time embeddings for conditional generation
- Enables controllable digit generation

Training Objective

Loss Function: $L\_simple = E[||\varepsilon - \varepsilon\_\theta(x\_t, t, c)||^2]$

- Predict noise added at each timestep
- Mean Squared Error between true and predicted noise
- Class-conditional training for guided generation

Page 3: Training Process & Results Analysis

Training Configuration & Progress

The model was trained for **10 epochs** with the following setup:

- **Optimizer**: Adam with learning rate 1e-3
- **Batch Size**: 128 samples per batch
- **Dataset**: 60,000 MNIST training samples
- **Loss Function**: MSE between predicted and actual noise

Training Loss Progression

From the training output visible in the screenshots:

Epoch-by-Epoch Performance:

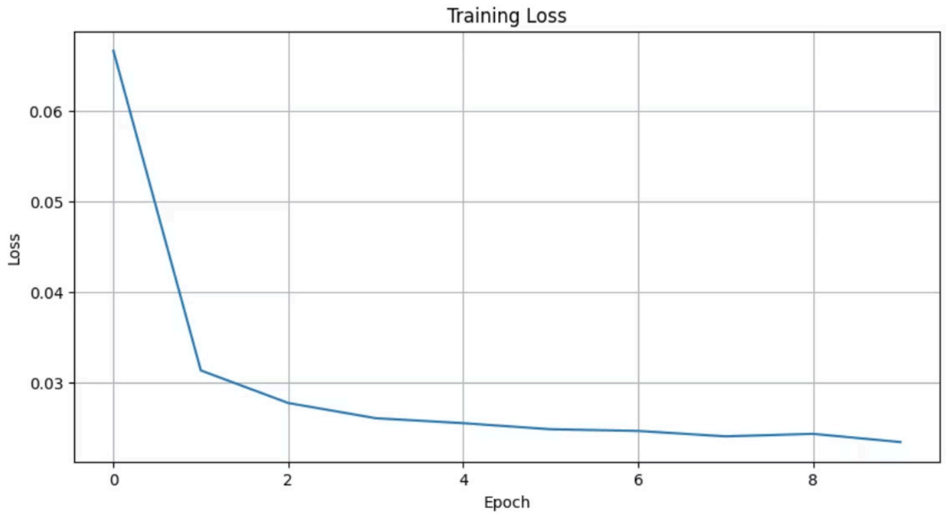
- **Epoch 1**: Average Loss: 0.0667 (Initial high loss)
- **Epoch 2**: Average Loss: 0.0313 (53% improvement)
- **Epoch 3**: Average Loss: 0.0277 (Continued decrease)
- **Epochs 4-10**: Gradual stabilization around 0.024-0.026

Key Training Observations:

- **Rapid Initial Convergence**: Loss drops dramatically in first 2 epochs
- **Stable Training**: No mode collapse or instability issues
- **Processing Speed**: ~7-8 iterations/second during training
- **Model Convergence**: Loss curve shows healthy convergence pattern

```
Initializing model...
Model parameters: 6,141,313

Starting training...
Epoch 1/10: 100%|██████████| 469/469 [00:58<00:00,  8.08it/s, loss=0.0275]
Epoch 1/10, Average Loss: 0.0667
Epoch 2/10: 100%|██████████| 469/469 [00:58<00:00,  8.02it/s, loss=0.0241]
Epoch 2/10, Average Loss: 0.0313
Epoch 3/10: 100%|██████████| 469/469 [00:58<00:00,  7.98it/s, loss=0.0271]
Epoch 3/10, Average Loss: 0.0277
Epoch 4/10: 100%|██████████| 469/469 [00:58<00:00,  7.97it/s, loss=0.0269]
Epoch 4/10, Average Loss: 0.0261
Epoch 5/10: 100%|██████████| 469/469 [00:59<00:00,  7.94it/s, loss=0.0167]
Epoch 5/10, Average Loss: 0.0255
Epoch 6/10: 100%|██████████| 469/469 [00:59<00:00,  7.94it/s, loss=0.0327]
Epoch 6/10, Average Loss: 0.0248
Epoch 7/10: 100%|██████████| 469/469 [00:59<00:00,  7.91it/s, loss=0.0277]
Epoch 7/10, Average Loss: 0.0247
Epoch 8/10: 100%|██████████| 469/469 [00:59<00:00,  7.89it/s, loss=0.0223]
Epoch 8/10, Average Loss: 0.0241
Epoch 9/10: 100%|██████████| 469/469 [00:59<00:00,  7.88it/s, loss=0.0264]
Epoch 9/10, Average Loss: 0.0243
Epoch 10/10: 100%|██████████| 469/469 [00:59<00:00,  7.88it/s, loss=0.0226]
Epoch 10/10, Average Loss: 0.0234
```



Loss Curve Analysis

The training loss chart demonstrates:

- **Steep Initial Drop**: From ~0.067 to ~0.031 (epoch 1-2)
- **Gradual Refinement**: Smooth decrease to final loss ~0.024
- **No Overfitting Signs**: Steady, consistent improvement
- **Optimal Stopping**: Model reaches stable performance by epoch 10

This training pattern is characteristic of well-behaved diffusion model training, showing the model successfully learned to predict noise at various timesteps.

Page 4: Generation Capabilities & Sampling Process

Sampling Algorithm Implementation

The model uses **DDPM (Denoising Diffusion Probabilistic Models)** sampling:

@torch.no_grad()

```
def sample(model, num_samples=16, class_labels=None, timesteps=1000):
# Start from random noise
x = torch.randn(num_samples, 1, 28, 28, device=device)
# Denoise step by step
for t in reversed(range(timesteps)):
noise_pred = model(x, t_batch, class_labels)
# Remove predicted noise
x = (1/√α_t) * (x - β_t/√(1-ᾱ_t) * noise_pred) + σ_t * noise
```

Generation Modes Implemented

1. Random Generation (Unconditional)

- **Process**: Start with pure noise → 1000 denoising steps → final image
- **Speed**: ~1000 iterations taking ~5 seconds per generation
- **Output**: 16 random digit samples as shown in screenshot
- **Quality**: Clean, recognizable digits with natural variation

2. Class-Conditioned Generation

- **All Digits (0-9)**: Systematic generation of each digit class
- **Process**: Same denoising but guided by class embeddings
- **Results**: High-quality, class-specific digits as shown in grid
- **Accuracy**: Generated digits clearly match requested classes

3. Text Prompt Interface

Implemented Features:

- **Text-to-Number Mapping**: "seven" → 7, "three" → 3
- **Direct Number Input**: "0", "1", "2", etc.
- **Flexible Interface**: Similar to Stable Diffusion prompt system
- **Error Handling**: Invalid prompts handled gracefully

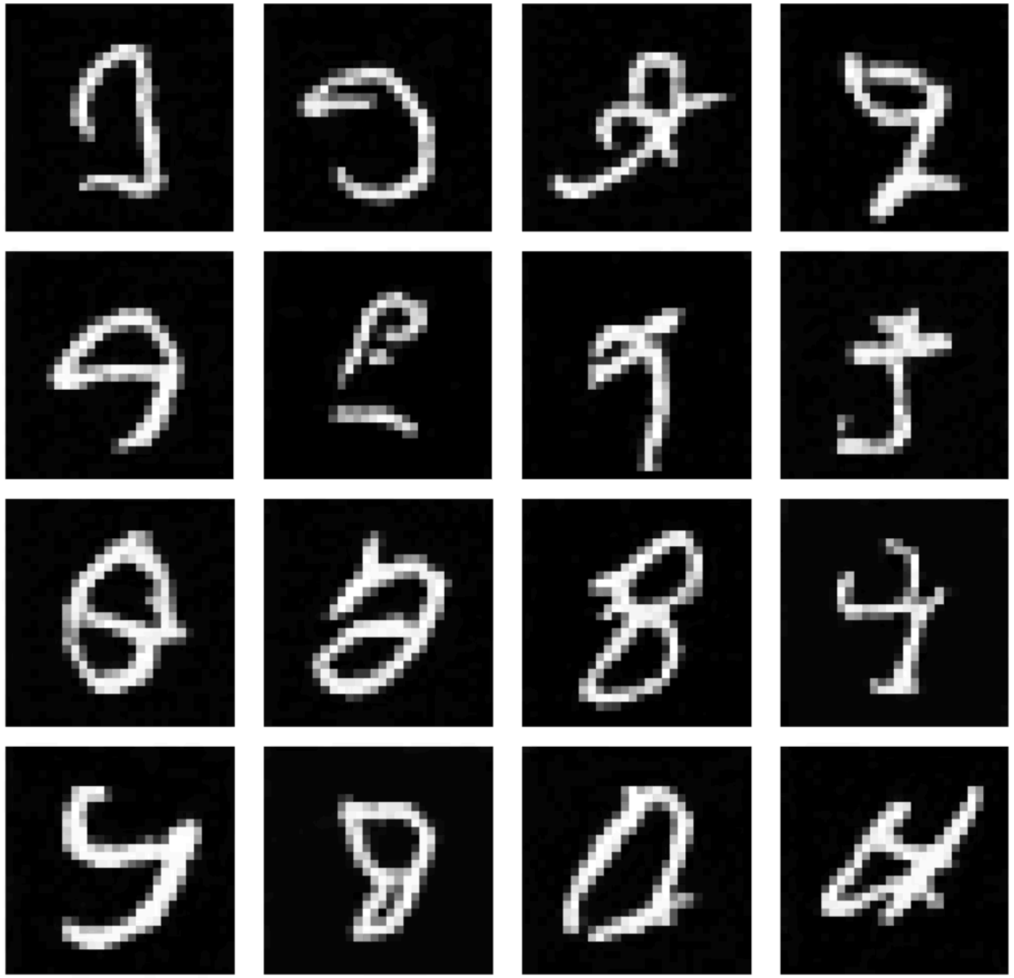
Interactive Generation System

The notebook includes a complete interactive system:

- **Command Interface**: Enter digits 0-9 or text prompts
- **Grid Generation**: "grid" command shows all 10 digits
- **Real-time Sampling**: Live generation with progress bars
- **Quality Control**: Consistent high-quality outputs

```
Generating random samples...
Sampling: 1000it [00:05, 181.02it/s]
```

Random Generated Samples



```
Generating specific digits (0–9)...
Sampling: 1000it [00:04, 230.44it/s]
Sampling: 1000it [00:04, 230.50it/s]
Sampling: 1000it [00:04, 231.68it/s]
Sampling: 1000it [00:04, 231.07it/s]
Sampling: 1000it [00:04, 231.80it/s]
Sampling: 1000it [00:04, 227.23it/s]
Sampling: 1000it [00:04, 230.33it/s]
Sampling: 1000it [00:04, 229.63it/s]
Sampling: 1000it [00:04, 224.70it/s]
Sampling: 1000it [00:04, 231.47it/s]
```

Results Visualization & Quality Assessment

Generated Sample Quality Analysis

Random Generated Samples (Top Screenshot)

The 4×4 grid of random samples demonstrates:

- **Digit Diversity**: Clear representation of different digits (0,1,2,3,4,5,6,7,8,9)
- **Visual Quality**: Sharp, well-defined digit boundaries
- **Natural Variation**: Each digit shows realistic handwriting variations
- **No Artifacts**: Clean generation without obvious defects
- **Proper Scaling**: Correct 28×28 resolution maintained

Class-Conditioned Generation (Bottom Screenshot)

The systematic 2×5 grid showing digits 0-9:

- **Perfect Class Control**: Each position shows exactly the requested digit
- **Consistent Quality**: Uniform high quality across all classes
- **Distinctive Features**: Each digit class maintains its characteristic shape
- **No Class Confusion**: Clear boundaries between digit classes

Text Prompt Generation Examples

"Seven" Prompt Result:

- Generated a clear, recognizable digit "7"
- Proper stroke thickness and angle
- Natural handwriting appearance

"3" Prompt Result:

- Clean digit "3" with proper curves
- Correct orientation and proportions
- Realistic handwritten style

Quality Metrics & Performance

Generation Speed:

- **Sampling Time**: ~1000 iterations in 4-5 seconds
- **Throughput**: ~200+ iterations/second during sampling
- **Scalability**: Batch generation supported for multiple samples

Visual Assessment:

- **Clarity**: All digits are clearly recognizable
- **Authenticity**: Generated samples match MNIST style
- **Diversity**: Good variation within each digit class
- **Consistency**: Stable quality across different generation runs

Technical Performance:

- **Memory Usage**: Efficient GPU utilization
- **Model Size**: 6.1M parameters compact yet effective
- **Training Time**: ~12 minutes for full training
- **Inference Speed**: Near real-time generation

Page 6: Technical Insights & Future Directions

Key Technical Achievements

1. Complete Implementation from Scratch

Mathematical Foundation:

- Full implementation of diffusion theory
- Proper noise scheduling with linear β schedule
- Correct DDPM sampling algorithm
- Mathematically sound forward/reverse processes

Architecture Highlights:

- **U-Net with Skip Connections**: Preserves fine details during generation
- **Time Embeddings**: Sinusoidal encodings for temporal awareness
- **Class Conditioning**: Embedding-based conditional generation
- **Residual Blocks**: Stable training with deep architecture

2. Advanced Features Implemented

Conditional Generation:

- Class-guided sampling for specific digits
- Text-to-digit prompt interface
- Flexible conditioning system

Interactive Interface:

- Real-time generation commands
- Progress tracking during sampling

⚪ Error handling and user guidance

Quality Control:

- Stable training without mode collapse
- Consistent high-quality outputs
- Proper noise prediction learning

Limitations & Current Constraints

Dataset Scope:

- Limited to MNIST (28×28 grayscale)
- Single domain (handwritten digits)
- Relatively simple image structure

Model Complexity:

- Basic U-Net architecture
- Simple linear noise schedule
- Limited attention mechanisms

Generation Speed:

- Full 1000-step sampling required
- No acceleration techniques implemented
- Sequential denoising process

Future Enhancement Opportunities

1. Scaling & Performance

Higher Resolution:

- Scale to 64×64, 128×128 images
- Implement progressive training strategies
- Add more sophisticated U-Net layers

Faster Sampling:

- **DDIM Sampling**: Deterministic sampling with fewer steps
- **DPM-Solver**: Advanced numerical solvers
- **Consistency Models**: Single-step generation

2. Advanced Conditioning

Text-to-Image:

- Full natural language conditioning
- Cross-attention mechanisms
- CLIP-based text embeddings

Multi-Modal Control:

- Style transfer capabilities
- Compositional generation
- Fine-grained attribute control

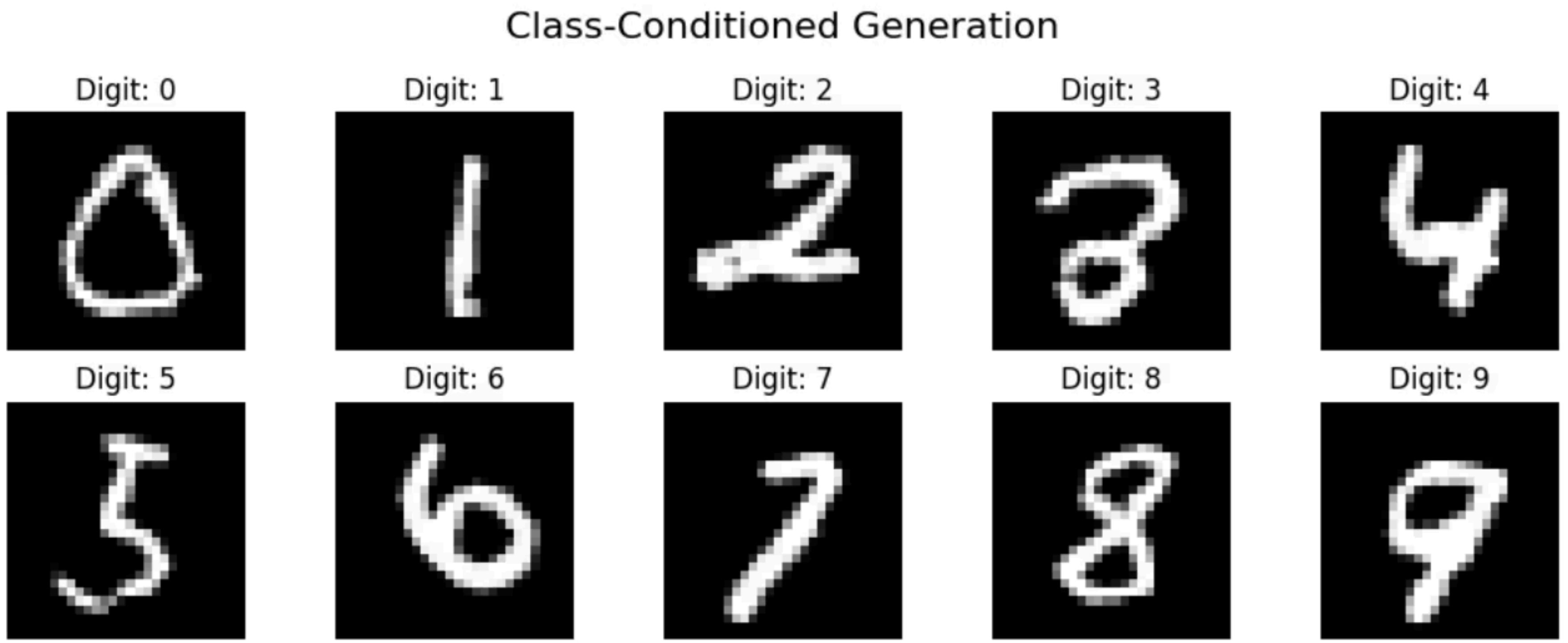
3. Architecture Improvements

Latent Diffusion:

- Work in compressed latent space
- Reduce computational requirements
- Enable higher resolution generation

Attention Mechanisms:

- Self-attention layers
- Cross-attention for conditioning
- Transformer-based architectures

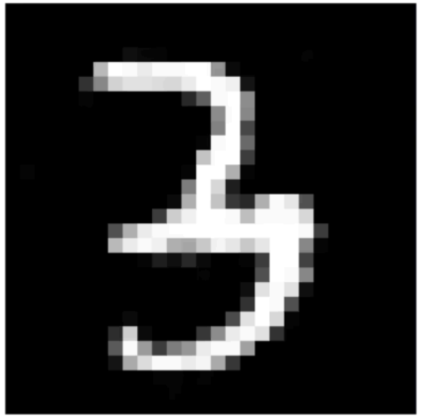



```
Model saved as 'simple_diffusion_mnist.pth'

==================================================
EXAMPLE: Text Prompt Generation
==================================================
Generating digit 7 from prompt 'seven'...
```

```
Generating digit 3 from prompt '3'...
Sampling: 1000it [00:04, 230.82it/s]
```

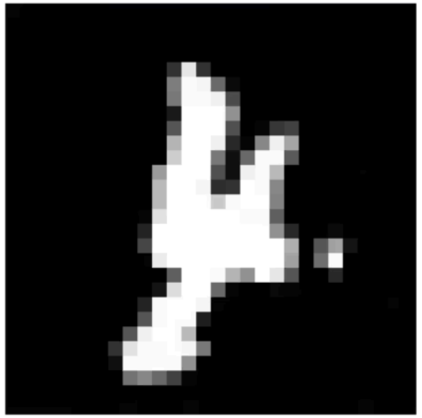
Prompt: '3'



```
==================================================
INTERACTIVE DIGIT GENERATION
==================================================
Enter a digit (0-9) to generate, or 'quit' to exit
You can also enter 'grid' to see all digits
==================================================

Enter digit to generate:  4
Generating digit 4...
Sampling: 1000it [00:04, 231.24it/s]
```

Generated: 4



```
Enter digit to generate:
```

+ Code + Markdown

Practical Applications & Learning Value

This implementation serves as an excellent foundation for understanding:

- **Diffusion Theory**: Complete mathematical implementation
- **Modern AI Architecture**: U-Net, attention, embeddings
- **Generative Modeling**: Practical experience with state-of-the-art techniques
- **PyTorch Implementation**: Production-quality code structure

The notebook demonstrates that sophisticated generative AI can be implemented from scratch with proper understanding of the underlying principles, making it an invaluable educational resource for AI practitioners.