CS324 · Deep Learning

Assignment 3 Report

Sicheng Zhou / 12110644

December 24, 2024

1 PyTorch LSTM

1.1 Model Structure

This LSTM implementation consists of a single LSTM layer followed by a fully connected output layer. It is designed to process sequential data with three key dimensions:

- Batch size: Number of sequences processed in parallel
- Sequence length: The temporal dimension of the input data
- Input dimension: The feature dimension of each time step

Core Components:

1. LSTM Cell Gates
   - Input gate (i_t): Controls what new information will be stored in the cell state
   - Forget gate (f_t): Determines what information to discard from the cell state
   - Cell candidate (g_t): Creates a vector of new candidate values for the cell state
   - Output gate (o_t): Controls what parts of the cell state will be output
2. Weight Matrices
   - Input weights (W_i, W_f, W_c, W_o): Transform input data for each gate
   - Recurrent weights (U_i, U_f, U_c, U_o): Transform previous hidden state
   - Bias terms (b_i, b_f, b_c, b_o): Offset values for each gate
   - Output layer weights (W_{out}, b_{out}): Transform final hidden state to output

1.2 Regularization and Optimization Features

The training implementation incorporates several modern optimization techniques and best practices for neural network training.

Weight initialization. We use Xavier initialization to initialize the weight matrix. If all parameters are initialized with the constant 0, it causes the network to start training with all neurons behaving the same, making it difficult for the network to learn effective features. Xavier initialization helps control the gradient scale when propagating forward and back. Initializing the offset of the forgetting gate to 1 can maintain more long-term dependent information at the beginning of training. Other biases are initialized to 0.0.

Dropout. We applied to hidden states with a dropout rate of 0.5 to prevent overfitting.

Forward pass.

1. Initialize hidden state (h_0) and cell state (c_0) to zeros
2. For each time step:
   - Process input through all gates
   - Update cell and hidden states
   - Apply dropout to hidden state

3. Use final hidden state to generate output through linear layer

Optimizer. The training implementation utilizes RMSprop optimiser for tuning the weights.

Loss function and gradient processing. Cross-entropy loss is employed as the primary objective function, which is particularly suitable for classification tasks. The implementation includes several crucial gradient handling techniques:

- Gradient clipping with a maximum norm of 10.0, implemented via `torch.nn.utils.clip_grad_norm_`
- Proper gradient zeroing before each backward pass using `optimizer.zero_grad()`
- Automated backpropagation through time (BPTT) handled by PyTorch's autograd system

1.3 Experimental Results

The model is obtaining close to perfect accuracy with T = 5 and the default parameters.

The LSTM model demonstrated a robust and successful training process over the course of 100 epochs. The training began with relatively poor performance. However, the model exhibited steady improvement during the early phases of training. A significant breakthrough occurred around epochs 12-15, where the validation accuracy notably improved from approximately 40% to over 55%, indicating the model had begun to effectively learn the underlying patterns in the data.

The middle phase of training, particularly between epochs 30 and 60, showed consistent progress in both training and validation metrics. The model's learning stabilized after epoch 60, with training accuracy maintaining levels above 75%. This period was characterized by gradual improvements and occasional minor fluctuations in validation accuracy, suggesting the model was fine-tuning its parameters and improving its generalization capabilities.

The later stages of training, especially after epoch 80, demonstrated remarkable performance with validation accuracy frequently reaching 100%. This phase showed strong evidence of successful learning and generalization, with validation metrics often surpassing training metrics. The final performance metrics were particularly impressive, with the model achieving a training accuracy of 85.75% and a validation accuracy of 99.50%. The corresponding final loss values were similarly encouraging, with training loss decreasing to 0.3915 and validation loss reaching 0.1444.

In conclusion, the training results indicate a highly successful implementation of the LSTM model, characterized by steady learning progression, strong final performance, and robust generalization capabilities. The model's evolution from poor initial performance to nearly perfect validation accuracy demonstrates the effectiveness of the chosen architecture and training approach.

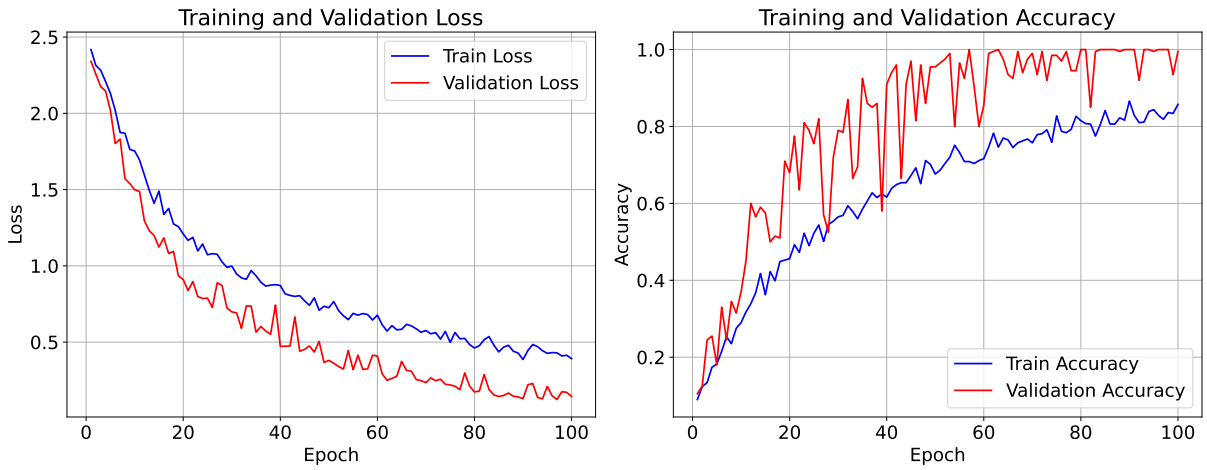

Figure 1: MLP Train and Test Loss on CIFAR10.

2 Generative Adversarial Networks

2.1 Model Structure

The GAN consists of a Generator, a network that generates fake images from random noise vectors sampled from a latent space, and a Discriminator, a network that evaluates images and determines whether they are real (from the dataset) or fake (produced by the generator).

The **Generator** is implemented as a fully connected neural network with a latent space dimension as input.

Input: A random noise vector z of size latent dim. Layers:

- Fully connected layer (Linear) mapping z to a 128-dimensional feature vector.
- Leaky ReLU activation with a negative slope of 0.2.
- Subsequent layers progressively increase the feature dimensions to 256, 512, and 1024, with Batch Normalization applied after each layer.
- Final fully connected layer maps the feature vector to a flattened 28x28 image (784 pixels).
- Tanh activation normalizes pixel values to the range [-1, 1].

Output: Reshaped to (1, 28, 28), representing a grayscale image.

The **Discriminator** is also a fully connected neural network (Discriminator class) that takes a 28x28 image as input and outputs a single probability score indicating its "realness."

Input: A flattened 28x28 image (784 pixels).

Layers:

- The fully connected layer maps the input to 512 features.
- Leaky ReLU activation with a negative slope of 0.2.
- Another fully connected layer reduces the dimensionality to 256.
- The final fully connected layer outputs a single probability value through a Sigmoid activation.

Output: A scalar value between 0 and 1, representing the likelihood that the input image is real.

2.2 Experimental Results

Figure 2, Figure 3, Figure 4 provide a visual representation of how the quality of images evolves throughout the training process. By examining the grids of images corresponding to the start, middle, and end of training, we can analyze the model's progression in learning to generate data that mimics the target distribution.

At the start of training, the images are usually dominated by random noise or unstructured outputs. This is expected because the model has just begun learning and lacks the ability to capture meaningful patterns from the data. These results highlight the initial state of the generator, where it produces outputs with no resemblance to the target dataset.

In the middle of training, the images show significant improvements, with recognizable patterns and partial structures emerging. In the MNIST case, some images start resembling digits, albeit with distortions or incomplete features. This stage demonstrates the generator's growing ability to approximate the data distribution as it begins to learn more complex representations.

By the end of training, the images typically display a high level of detail and accuracy, closely resembling the samples in the target dataset. The noise from earlier stages is replaced by clear, structured outputs.

Figure 5 visualize the smooth transition between two random points in the latent space of a trained GAN generator. By performing linear interpolation in the latent space, the generator produces a sequence of

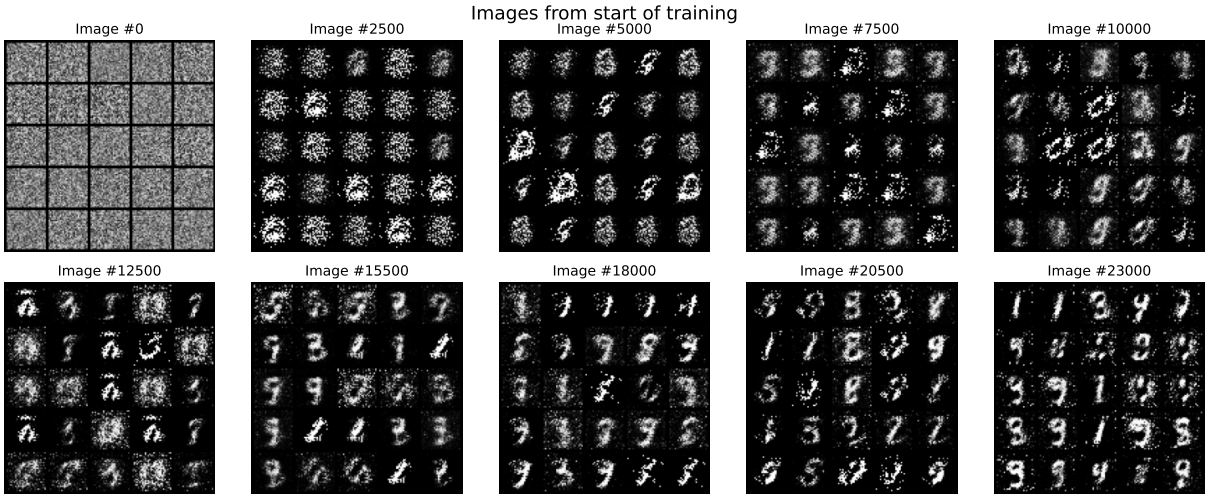

Figure 2: Images generated from GAN from different training stages.

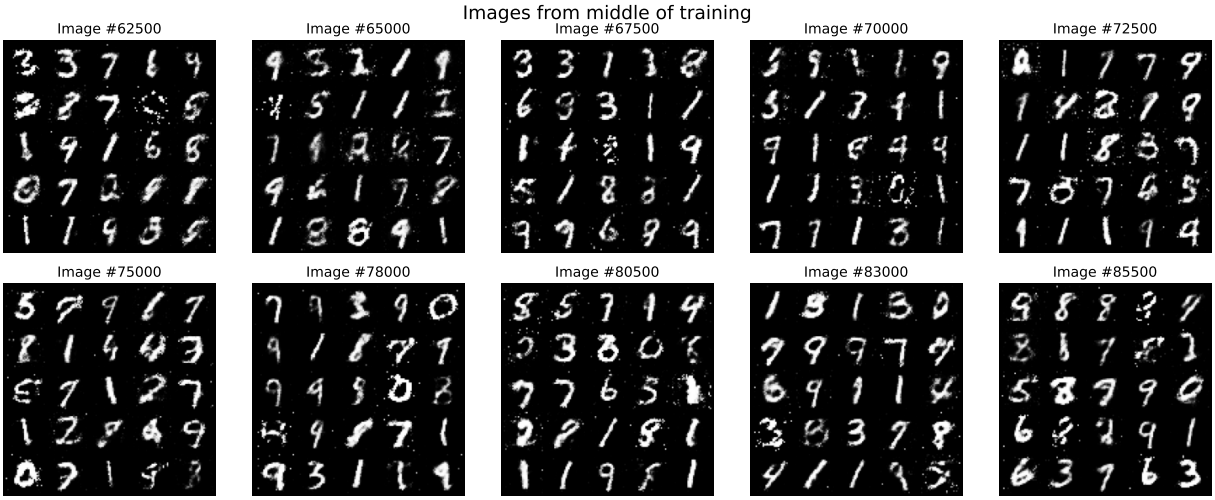

Figure 3: Images generated from GAN from different training stages.

images that gradually morph from one representation to another. This demonstrates the continuity and meaningfulness of the generator's learned latent space.

If the generator has been trained well, the transition between the two images is visually coherent, and each intermediate image appears realistic and meaningful. This result highlights the quality of the generator's learned representation, where nearby points in the latent space correspond to semantically similar images. Irregularities or abrupt changes during the interpolation might indicate areas where the generator's latent space lacks smoothness or consistency, suggesting potential issues with the training process or model design.

The final visualization provides an intuitive understanding of how the GAN encodes information in the latent space and serves as a qualitative measure of the generator's capability to create diverse and realistic outputs. By saving the grid as a PDF file (interpolation.pdf), users can further analyze and document the interpolation results for future reference.

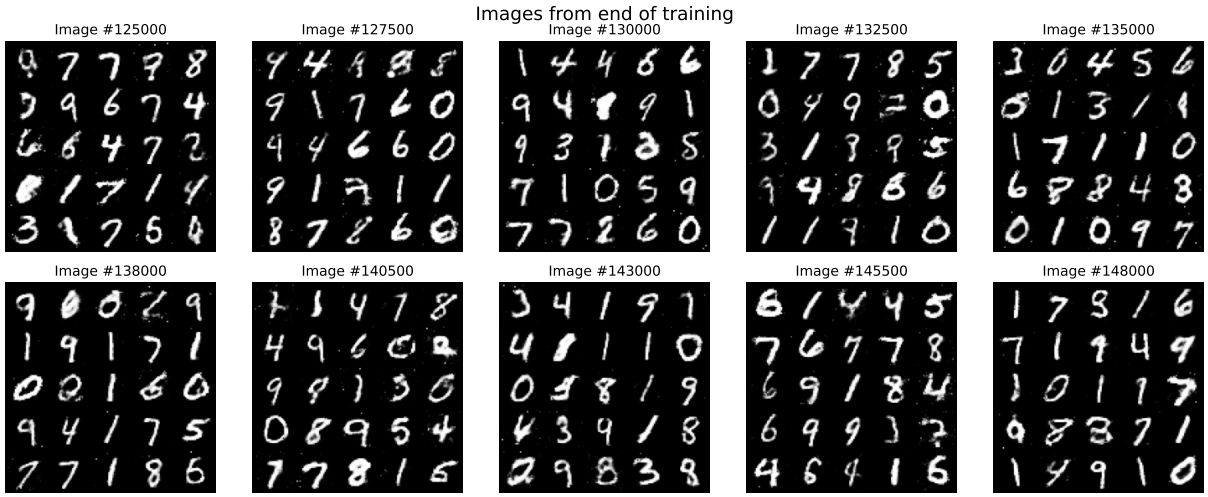

Figure 4: Images generated from GAN from different training stages.

Figure 5: Sample two different images from GAN and interpolate between these two digits in latent space.