CS324 Deep Learning
Assignment 3

Kuang Liang

December 21, 2024

Abstract

For this assignment, I implemented, trained and tested an LSTM and a GAN based on PyTorch. In the report, I will explain how I implemented them and the experiments I conducted with them.

1 LSTM

1.1 Model

An LSTM is an advanced RNN architecture that addresses the vanishing gradient problem and enhances the ability to capture long-term dependencies. Below is a breakdown of the structure:

1. Input Modulation Gate (g_t): This gate computes a candidate memory content using the current input (x_t) and the previous hidden state (h_{t-1}), passed through a *tanh* activation function.

2. Input Gate (i_t): The input gate determines the extent to which new information should be stored in the memory. It applies a sigmoid activation function to control its influence.

3. Forget Gate (f_t): The forget gate regulates which parts of the previous memory (c_{t-1}) are retained. Like the input gate, it uses a sigmoid activation.

4. Cell State Update (c_t): The memory cell updates itself based on the combined influence of the input modulation and forget gates.

5. Output Gate (o_t): The output gate determines how much of the updated cell state (c_t) should contribute to the current output.

6. Hidden State Update (h_t): The updated hidden state is obtained by applying the output gate to the *tanh* of the cell state.

7. Prediction Layer $(p_t, \tilde{y}_t)$: A linear transformation of the hidden state generates logits (p_t), which are then normalized via softmax to produce class probabilities.

In this assignment, our LSTM model is designed such that the input dimension $d_{in} = 1$, the hidden state dimension $d_h = 128$, and the output dimension $d_{out} = 10$.

1.2 Experiment

In this experiment, the evaluation task is predicting the final digit of a palindrome sequence. The model is trained with a batch size of 128, a learning rate of 0.001, and for $E = 20$ epochs. We conducted experiments with input sequence lengths ranging from 3 to 24 and recorded the validation accuracy, as shown in fig. 1. For comparison, the experimental results of an RNN under identical settings are provided in fig. 2. Due to computational resource constraints, experiments with sequence lengths ranging from 25 to 35 were trained for only $E = 5$ epochs, as shown in fig. 3. The experimental results indicate that LSTM significantly outperforms RNN in retaining historical information, as evidenced by the increased effective sequence length. However, a sharp drop in accuracy still occurs sometimes.

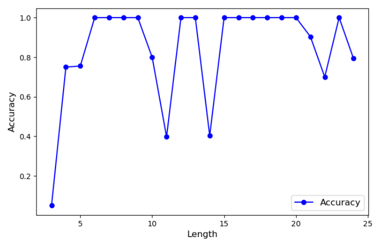

Figure 1: Experiment result on LSTM.

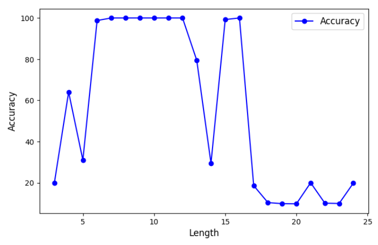

Figure 2: Experiment result on RNN.

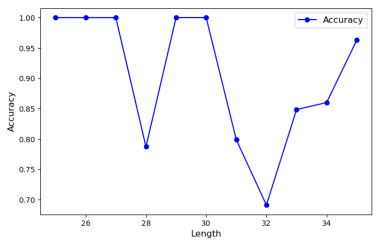

Figure 3: Extra experiment result on LSTM.

2 GAN

2.1 Model

GAN consists of two primary components: a generator and a discriminator, which are trained adversarially.

The Generator (G) is a neural network designed to map a latent space vector (z) sampled from a noise distribution to a realistic data distribution. The architecture is as follows:

- Input: A latent vector of dimension d_z (default: 100).
- Layers:
  - Linear layer: Maps $z \in \mathbb{R}^{d_z}$ to 128-dimensional space.
  - LeakyReLU activation $(\alpha = 0.2)$.

  - Linear layer: Expands to 256 dimensions, followed by Batch Normalization and LeakyReLU.
  - Linear layer: Expands to 512 dimensions, followed by Batch Normalization and LeakyReLU.
  - Linear layer: Expands to 1024 dimensions, followed by Batch Normalization and LeakyReLU.
  - Linear layer: Maps 1024-dimensional space to 784 (flattened 28×28) dimensions, followed by a *tanh* activation to normalize pixel values to $[-1, 1]$.

- Output: A 784-dimensional vector reshaped into a 28×28 image.

The Discriminator (D) is a binary classifier designed to distinguish real images from fake images generated by G. The architecture is:

- Input: A flattened $28 \times 28 = 784$-dimensional image vector.
- Layers:
  - Linear layer: Maps 784-dimensional input to 512 dimensions.
  - LeakyReLU activation ($\alpha = 0.2$).
  - Linear layer: Reduces to 256 dimensions, followed by LeakyReLU.
  - Linear layer: Reduces to 1 dimension, followed by a Sigmoid activation to produce a probability.
- Output: A scalar in $[0, 1]$, representing the likelihood of the input being a real image.

2.2 Training

The GAN is trained using a min-max optimization objective:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

- Discriminator Training: D is trained to maximize the likelihood of correctly classifying real and fake images.

$$L_D = -\mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

- Generator Training: G is trained to minimize $\log(1 - D(G(z)))$, which is equivalent to maximizing $\log D(G(z))$.

$$L_G = -\mathbb{E}_{z \sim p_z}[\log D(G(z))]$$

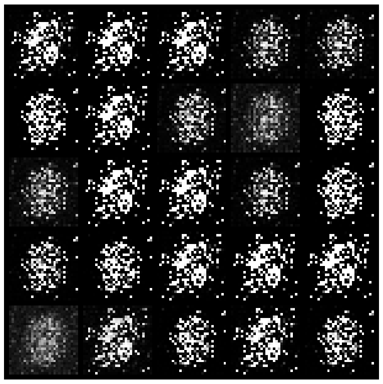

Figure 4: G before training.

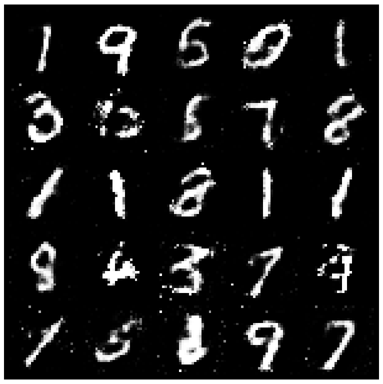

Figure 5: G after training for 100 epochs.

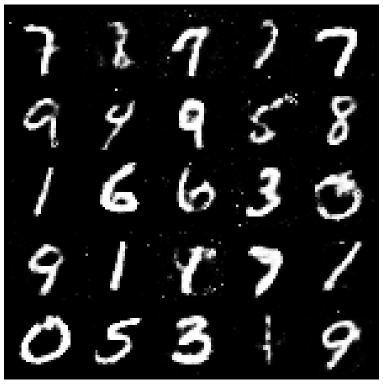

Figure 6: G after training for 200 epochs.

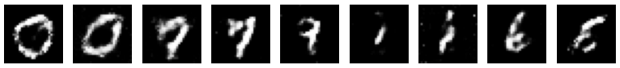

Figure 7: Results of the interpolation experiment.

2.3 Experiment

The experiment is conducted on the MNIST dataset. Both G and D are trained with an Adam optimizer with learning rate 0.0002 for 200 epochs. figs. 4 to 6 show the performance of the generator G before, during and after the training.

The experimental results show that the generator can effectively mimic the samples from the dataset. Additionally, we conducted an interpolation experiment with 7 steps between two different digits in the latent space. The generated images fig. 7 in each step not only resemble valid digits but also exhibit a smooth transition effect. These two experiments demonstrate the effectiveness of the trained generator.