Deep Learning (CS324)
Assignment 3 Report

Fitria Zusni Farida
Student ID: 12112351

June 6, 2024

1 Introduction

(1) Long Short Term Memory RNN

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) that are well-suited to learning and remembering over long sequences of data. LSTMs are designed to avoid the long-term dependency problem which standard RNNs struggle with. LSTMs are a more sophisticated version of RNNs designed to overcome the limitations of Vanilla RNNs in handling long-term dependencies. The key features of LTSM are cell states and gates.

(2) Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) are a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014. GANs are primarily used for generating synthetic data that resembles real data. They consist of two neural networks: the Generator and the Discriminator, which compete against each other in a game-theoretic setup.

2 Motivation

The primary motivation of this assignment are as follow:

(1) Implement LTSM to predict the last digit of palindrome. Then analyze the loss and accuracy.

(2) Compare the LTSM implementation performance with Vanilla RNN in the Assignment 2.

(3) Build and train GAN in MNIST dataset.

(4) Sample 25 images from the trained GAN.

(5) Sample 2 images from different classes from GAN. Interpolate these two digits in Latent space.

3 Methodology

In this assignment, the task are divided into two parts: (1) PyTorch LSTM, and (2) GAN

(1) **Part I: PyTorch LSTM**

In this task, the task is generate the last digit of palindrome using long short term memory in PyTorch. Compared with previous implementation, which used Vanilla RNN, this LSTM implementation provides more features and advancements, those are the cell and gates. The structure of LSTM is described in the following picture.

This LSTM structure has more complex structure which consist of several gates compare with single neuron in Vanilla RNN. The gates are as following:

- Input gate (i_t)
  controls the extent to which the new information coming from the input x_t and the previous hidden state h_{t-1} should be allowed into the cell state. It helps in deciding which values to update in the cell state.

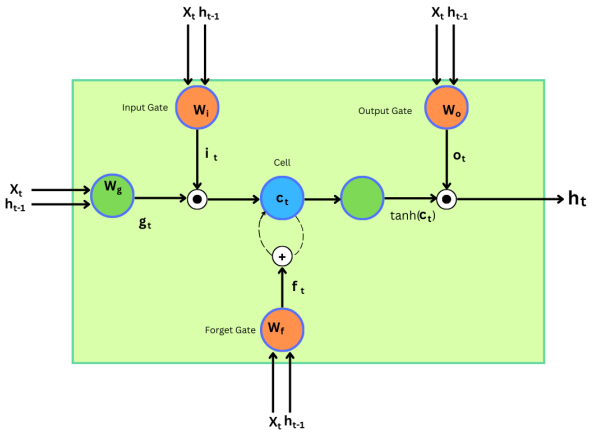

Figure 1: LTSM structure

$$i_t = \sigma(W_{ix}x_t + W_{ih}h_{t-1} + b_i)$$

- W_{ix} and W_{ih} are the weights for the input x_t and the hidden state h_{t-1}, respectively.

- b_i is the bias term.

- σ denotes the sigmoid function, which outputs values between 0 and 1. This range allows the gate to decide how much of each component should be let through.

- Forget gate (f_t)

  The forget gate controls what information from the previous cell state c_{t-1} should be carried forward. It essentially decides what to forget or retain from the past.

$$f_t = \sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f)$$

- W_{fx} and W_{fh} are the weights for the input x_t and the hidden state h_{t-1}, respectively.

- b_f is the bias term.

- The sigmoid function σ outputs a value between 0 and 1 for each number in the cell state c_{t-1}.

- Output gate (o_t)

  The output gate controls what part of the cell state should be output to the next hidden state. It decides what the next hidden state h_t should be.

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o)$$

- W_{ox} and W_{oh} are the weights for the input x_t and the hidden state h_{t-1}, respectively.

- b_o is the bias term.

- The sigmoid function σ provides a gate value that will be multiplied with the tanh of the cell state to decide the output.

- Candidate Cell state (g_t

  The candidate cell state is the new information that could be added to the cell state.

$$g_t = \tanh(W_{gx}x_t + W_{gh}h_{t-1} + b_g)$$

- W_{gx} and W_{gh} are the weights for the input x_t and the hidden state h_{t-1}, respectively.

- b_g is the bias term.

- The tanh function ensures that the values are in the range $[-1, 1]$.

- Cell state (c_t)

  The cell state is a key component of the LSTM that carries information across different time steps. It's modified slightly at each time step depending on the input and forget gates.

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

- $\odot$ denotes element-wise multiplication. - The forget gate f_t determines how much of the previous cell state c_{t-1} should be retained. - The input gate i_t determines how much of the candidate cell state g_t (new information) should be added.

- Hidden state (h_t)

  The hidden state is the output of the LSTM cell at the current time step, which is used as input for the next time step and also for any further layers in the network.

$$h_t = o_t \odot \tanh(c_t)$$

- The output gate o_t decides what part of the cell state should be output.

- $\tanh(c_t)$ transforms the cell state values to be within $[-1, 1]$.

(2) **Part II: GAN**

In this part, the task is to build and train GAN using MNIST dataset and generate some samples from it. GAN consists of two neural networks: the Generator and Discriminator, which compete against each other in a game-theoretic setup.

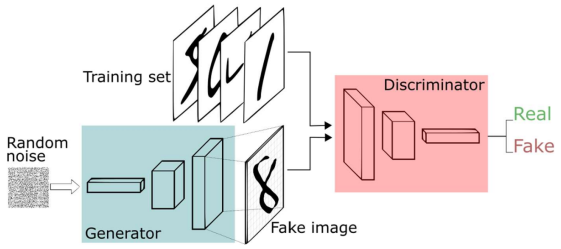

Figure 2: GAN architecture

- **Generator G**

  It generate synthetic data that is similar to the real data. It takes a random noise vector from sampled distribution as input and transforms it into a data sample. It tries to produce data that the Discriminator cannot distinguish from real data and maximize the probability that the the Discriminator makes a mistake.

- **Discriminator D**

  Its job is to distinguish between real data (from the training set) and fake data produced by the Generator. It takes a data sample as input and outputs a probability indicating whether the sample is real or fake. The Discriminator aims to minimize the probability of making a mistake.

Objective functions

- Discriminator Objective Function

  The Discriminator aims to maximize the following objective function:

$$\mathcal{L}_D = \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \quad (1)$$

- Generator Objective Function

  The Generator aims to maximize the following objective function:

$$\mathcal{L}_G = \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))] \quad (2)$$

- Alternative Generator Objective Function

  In practice, an alternative loss for the Generator that avoids vanishing gradients is often used:

$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log D(G(\mathbf{z}))] \quad (3)$$

Training Process

- **Step 1: Initialization**

  Initialize the Generator and Discriminator with random weights.

- **Step 2: Train the Discriminator**

  - Sample a batch of real data from the training set.

  - Sample a batch of random noise vectors and generate fake data using the Generator.

  - Calculate the Loss using both real and fake data using objective value (1) mentioned above.

  - Perform backpropagation and update the Discriminator's weights to minimize the loss.

- **Step 3: Train the Generator**

  - Sample a batch of random noise vectors and generate fake data using the Generator.

  - Calculate the Loss based on the Discriminator's ability to classify the fake data as real using the objective value (3).

  - Perform backpropagation and update the Generator's weights to maximize the Discriminators's classification error on the fake data.

4 Experiments

(1) **Part I: PyTorch LSTM**

Task 1

1) Dataset Preparation

The dataset is palindrome dataset with a digit label which consist of digit numbers with input length N. In this experiment, there are 1,000,000 samples with input length is 19 for each sample. The dataset is divided into training and validation set with ratio 8:2.

2) Network Architecture

As described in the Figure 1, which each of the layer has input gate, forget gate, and output gate.

3) Hyperparameter configuration

input dim = 1

num classes = 10

num hidden = 128

batch size = 128

learning rate = 0.001

max epoch = 100

max norm = 10.0

data size = 100000

portion train = 0.8

4) Result

For the input length 5, the performance result can be obtained as following in Figure 3:

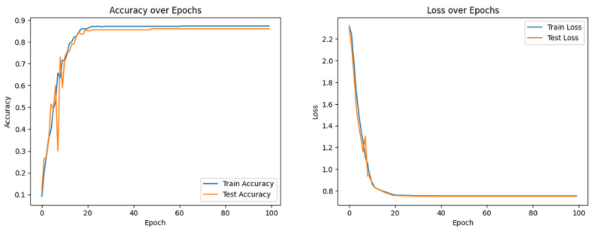

Figure 3: Accuracy and Loss LSTM input length 5

Whereas for input length 19, the result is as follow in Figure 4:

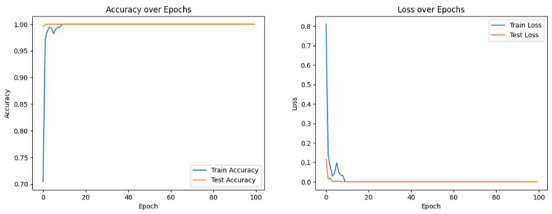

Figure 4: Accuracy and Loss LSTM input length 19

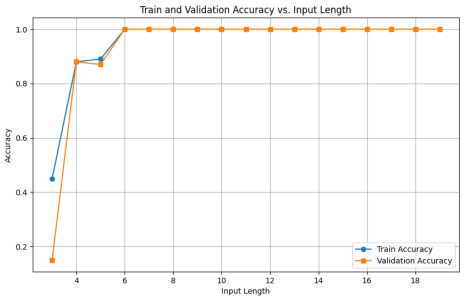

Figure 5: LSTM result accuracy

Moreover, after assessing with input length 3-19, the result can be concluded as following in Figure 5:

From Assignment 2, Vanilla RNN, the performance result for input length 5 and other input length are as follows in Figure 6:

(2) **Part II: GAN**

1) Dataset preparation The dataset used is digit handwritten from MNIST dataset, which consist of 70,000 greyscale images. The image size is 28x28 pixel.

2) Network architecture The architecture consist of the Generator and Discriminator.

- Generator

  The Generator is responsible for generating synthetic images from random noise vectors. The architecture consists of several linear layers, activation functions, and batch normalization layers. The layer is divided into an input

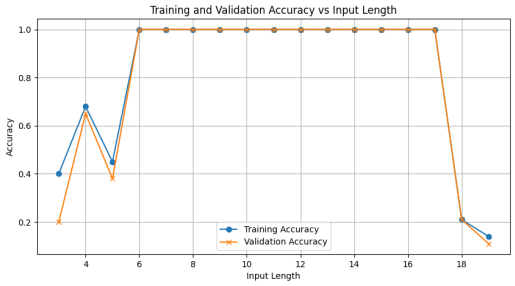

Figure 6: Vanilla RNN result accuracy

layer, 3 hidden layer, an output layer. The input and hidden layer use linear layer as activation function. The reason is it is able to address dying ReLU problem, maintain gradient for negative inputs, faster convergence, and stable training.

$$\text{Leaky ReLU(x)} = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

In this assignment alpha 0.2 is used. Another function implemented is Batch normalization which is used in the hidden layer. The function is used to normalize the output of previous layer to have a mean of 0 and standard deviation of 1, which is able to stabilize and speeding up the training. The output layer uses Tanh as activation function, which scales the output to the range [-1,1], suitable for image pixel values after normalization. The forward pass takes latent vector z, passes it through sequential layers and reshape the output to match the dimensions of MNIST image (1, 28, 28).

- Discriminator

  The Discriminator is responsible for classifying images as real or fake. The architecture consists of several linear layers and activation functions. It is divided into an input layer, a hidden layer, and and an output layer. The input layer and hidden layer consist of linear layer and Leaky ReLU as activation function. Whereas the output layer use sigmoid as activation function to indicate otuput probability.

3) Hyperparameter configuration

input size = 100

hidden size = 256

image size = 28 * 28

batch size = 64

num epochs = 200

learning rate = 0.0002

4) Result

During the training, the result can be observed as following:

- Beginning of training (epoch 1) (Figure 7)

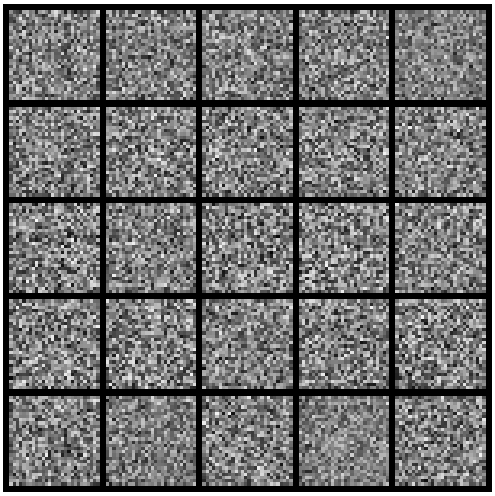

Figure 7: Beginning of training LSTM

- Midway training (epoch 100) (Figure 8)

- End training (epoch 200) (Figure 9)

By sampling 25 images from the trained GAN, it can be obtained following result in Figure 10.

Also, in in this task also interpolated two digits in the latent space. The first is, z1 and z2 are sampled from a standard normal distribution. The result is as following Figure 11:

The second is, z2 is sampled scaled by 2, which mean z2 is farther away from the origin in the latent space compared to z1. The result is as following in Figure 12:

Figure 8: Midway training LSTM

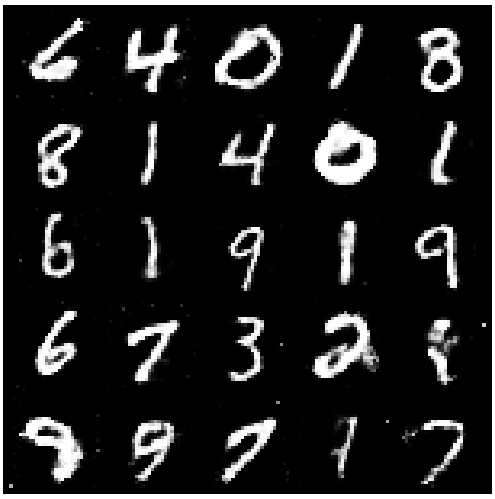

Figure 9: End training LSTM

5 Discussion and Insights

From the experiment result, it can be obtained following analysis:**Part I: PyTorch LSTM** From the Using long short term memory, it can be seen from Figure 3 and 6, that for input shorter input length (3-5) the accuracy improve slightly better. However, for the input reaches 18 or more, LSTM is able to obtain perfect accuracy, in which the Vanilla RNN's accuracy drops significantly. This is a proof that LSTM model perform better especially for handling long dependency. The Vanilla RNN may encounter issue while handling long sequence such as vanishing gradient. **Part II: GAN** From training result, it can be seen that in the first epoch, it obviously did not draw any digit of numbers as can be seen in the Figure 7. However, during the training the generated image appear to obtain clearer digits (as can be seen in the midway training in the Figure 8). Finally, in the epoch 200, which is the final training, the image can been see much cleared to draw the digits of number as can be seen in the Figure 9. From the 25 images sampled from the trained GAN, it can be seen that the images appear to be blurry and distorted. Some digits are partially recoginizeable, but many of the images show artifacts and noise. Overall, the quality suggest that the Generator has learned some features of the digits but is not yet producing high-quality, clear images. This is possibly because of insufficient training, unstable training, too small network architecture. To obtain better result, the model may need more complex and deeper network architecture, larger amount of epochs, and adjusting hyper parameters. Moreover, in the image interpolation task, images are generated from random vectors sampled from a latent space (a high-dimensional space). Each vector in this latent space corresponds to a different generated image. Interpolating between vectors involves creating intermediate vectors that are linear combinations of the two original vectors. For two latent vectors z1 and z2, and the interpolation factor alpha ranging from 0 to 1, the interpolated vector can be calculated as:

$$z_{\text{interpolated}} = (1 - \alpha) \cdot z_1 + \alpha \cdot z_2 \quad \textbf{(2)}$$

- When $\alpha = 0$, $z_{\text{interpolated}} = z_1$.
- When $\alpha = 1$, $z_{\text{interpolated}} = z_2$.
- For α values between 0 and 1, $z_{\text{interpolated}}$ represents points on the straight line between z_1 and z_2 in the latent space.

By feeding the interpolated vectors into the Generator, it can be produced a sequence images that

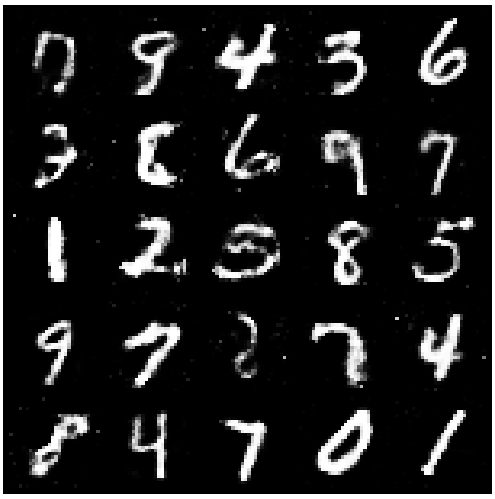

Figure 10: sampling 25 images from the trained GAN

Figure 11: Interpolation sampled from normal distribution

smoothly transition from the image corresponding to z1 and to the image corresponding to z2. As shown in the Figure 9, which the z1 and z2 are sampled from standard normal distribution, the interpolation between them result in smoother transitions between the generated images because both vectors are within a similar range of the latent space. On the other hand in the Figure 8, which z2 is scaled is scaled by factor of 2. This means z2 is farther away from the origin in the latent space compared to z1. The interpolation between z1 and z2 will cover a wider range in the latent space, resulting in more significant and noticeable changes between the generated images.

6 Conclusions

In this series of experiments, I explored the Long Short Term Memory (LSTM) model which is able to overcome long dependecy in the Vanilla RNN. The results clearly demonstrated that LSTM models significantly outperform Vanilla RNNs, especially for longer input sequences, achieving perfect accuracy where Vanilla RNNs struggled. Additionally, I delved into the application of Generative Adversarial Networks (GANs) for image generation. GANs have shown remarkable potential in creating realistic images.

Figure 12: Interpolation which z2 scaled by 2