SUSTech CS324 2024 Spring Assignment 3 Report

12110304 Chunhui XU

May 23, 2024

1 Part 1 Task 1

1.1 Simple Introduction

In Assignment 3, we train our model on the same data set, that is, the prediction of the palindrome structure mentioned in Assignment 2.

Of course, in Assignment 2, we tried to use RNN to solve this problem, but the results were not satisfactory: the model still performed better for shorter palindrome strings, but the performance for longer strings was somewhat different. Decline, even training results are getting worse. Therefore, in this exercise we try to use LSTM (Long Short-Term Memory) [2] with better memory ability to solve this problem.

1.2 LSTM Structure

1.2.1 LSTM Formulas

I build the LTSM with the following formulas:

$$g^{(t)} = \tanh(W_{gx}x^{(t)} + W_{gh}h^{(t-1)} + b_g) \tag{1}$$
$$i^{(t)} = \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i) \tag{2}$$
$$f^{(t)} = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f) \tag{3}$$
$$o^{(t)} = \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o) \tag{4}$$
$$c^{(t)} = g^{(t)} \odot i^{(t)} + c^{(t-1)} \odot f^{(t)} \tag{5}$$
$$h^{(t)} = \tanh(c^{(t)} \odot o^{(t)}) \tag{6}$$
$$p^{(t)} = (W_{ph}h^{(t)} + b_p) \tag{7}$$
$$\tilde{y}^{(t)} = \text{softmax}(p^{(t)}) \tag{8}$$

where $\odot$ denotes the element-wise multiplication.

The structure is similar to that of the RNN implemented in Assignment 2, but with the addition of three gates: the input gate i, the forget gate f, and the output gate o (and the input modulation g). I shall initialise $h^{(0)}$ to the vector of all zeros.

Use the cross-entropy loss over the last time-step:

$$\mathcal{L} = -\sum_{k=1}^{K} y_k log(\tilde{y}_k^{(T)}) \tag{9}$$

where k runs over the classes ($K = 10$ digits in total), y_k is a one-hot encoding vector.

1.2.2 `PyTorch` Implementation

When I implement the structure, for each one among g, i, f, o, I use `nn.Linear` with bias to compute the $x^{(t)}$ part, `nn.Linear` without bias to compute the $h^{(t-1)}$ part.

In each `for` loop, for h and c, update them with `torch` notation $*$ and `torch.tanh`.

Finally, use `nn.Linear` with bias to compute the $p^{(t)}$, and use `torch.softmax` to get the output.

2 Part 1 Task 2

2.1 Simple Introduction

In this part, I will train my LSTM on palindrome dataset with default parameters, and compare the different result with changing the `input_length`.

2.2 Default Parameters

- `input_length`: 4, length of input sequence
- `input_dim`: 1, dimension of input data
- `num_classes`: 10, number of classes in the classification task
- `num_hidden`: 128, number of hidden units in the neural network
- `batch_size`: 128, batch size for training
- `learning_rate`: 0.001, learning rate for optimization
- `max_epoch`: 100, maximum number of epochs to train the model
- `max_norm`: 10, maximum norm constraint for gradient clipping
- `data_size`: 100000, size of the dataset
- `portion_train`: 0.8, portion of the dataset used for training

I will tested the training efficiency under different T values (i.e. `input_length` + 1).

2.3 Result Visualization

Table 1 shows different parameters and corresponding figure.

Table 1: Different Parameters and Result

Fig ID	T	**Changed Parameters**
Fig 1	5	Default Parameters
Fig 2	20	Default Parameters
Fig 6	5	Change leaning rate to $1e^{-2}$ and $1e^{-4}$
Fig 7	20	Change leaning rate to $1e^{-2}$ and $1e^{-4}$
Fig 8	30	Change leaning rate to $1e^{-4}$
Fig 9	20	Assignment 2 RNN Network

2.4 Result Analysis

2.4.1 Overall Analysis

We can find that compared to RNN, LSTM not only achieves a better upper limit of accuracy on long inputs, but also has a significantly improved convergence speed. Compared with the original RNN structure, we can reasonably speculate that LSTM can achieve better training results on longer sequence inputs.

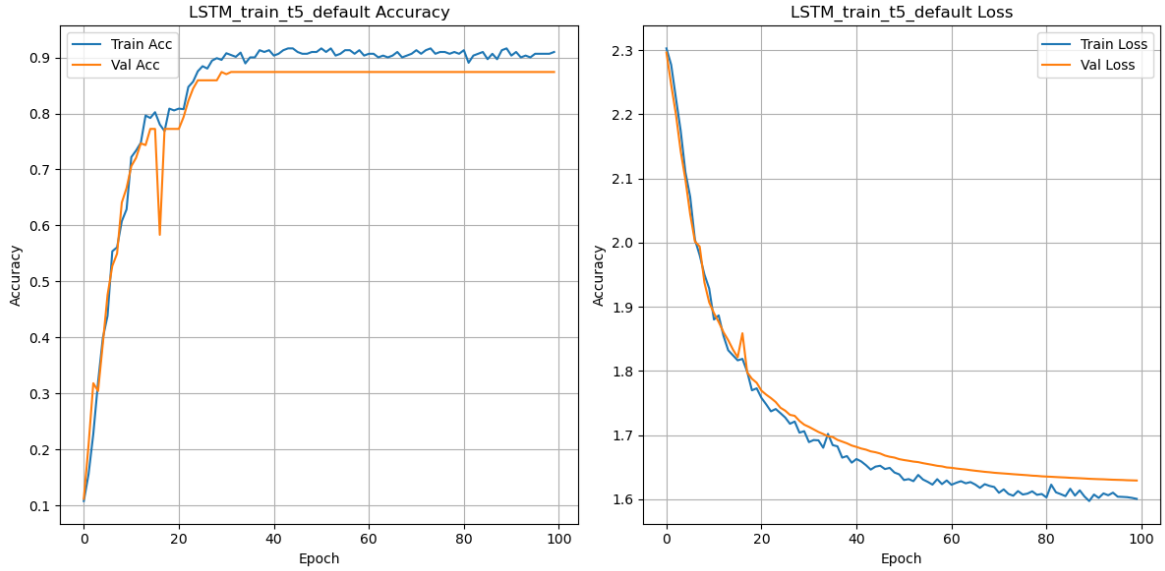


Figure 1: $T = 5$ Default Curve

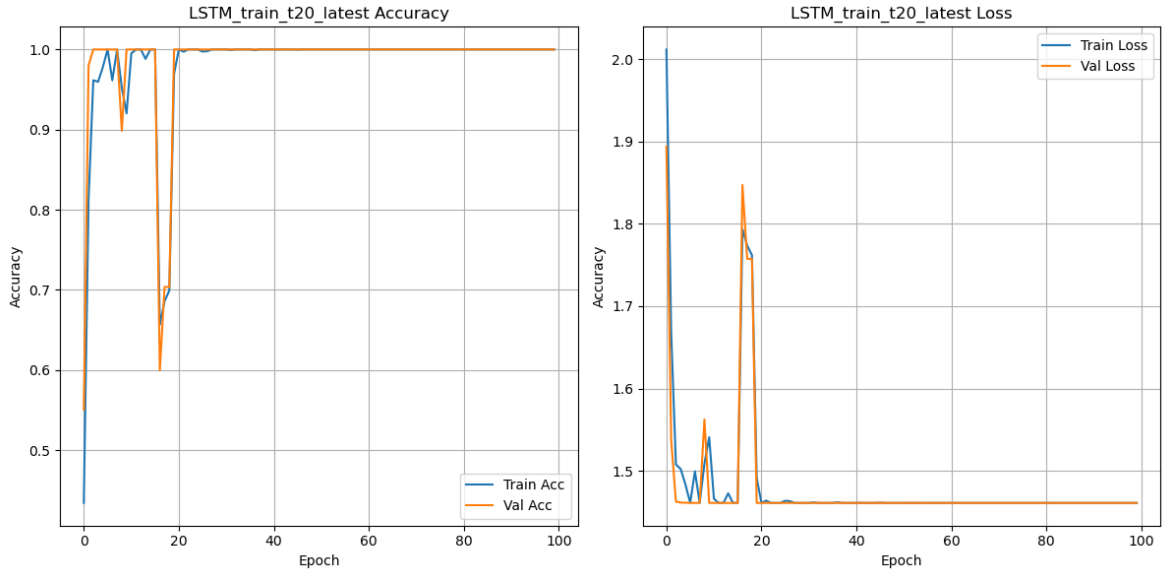


Figure 2: $T = 20$ Default Curve

2.4.2 Learning Rate Analysis

By changing variables, especially the learning rate, we also found more interesting results.

- For shorter inputs, a lower learning rate converges slower and has poorer learning effects; a higher learning rate can speed up the convergence and reach a higher upper limit of accuracy;
- For longer inputs, a lower learning rate can eventually reach a higher upper limit of accuracy; while a higher learning rate leads to model degradation, and the prediction effect fluctuates and becomes worse.

Here are the possible explanations I can think of for these situations:

- Increasing the learning rate might help the model converge faster, especially early in training. For shorter inputs, a higher learning rate means faster and more efficient convergence.
- Reducing the learning rate can help improve the stability of the model and reduce instability during the training process. Especially when the sequence length increases, problems such as vanishing/exploding gradient are more likely to occur. By reducing the learning rate, the magnitude of parameter updates can be reduced to better deal with these problems.

3 Part 2 Task 1

3.1 Simple Introduction

In the second part of this assignment you will implement a GAN (Generative Adversarial Network) [1] that generates images similar to those of the training set. The training set is MNIST handwritten digit database.

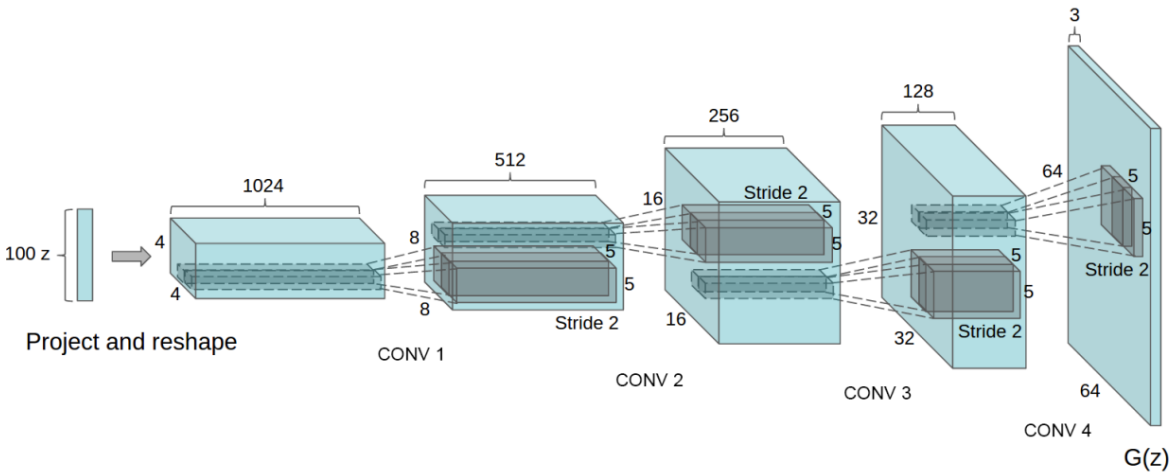


Figure 3: A Typical GAN Structure

3.2 Function Analyze

GAN is a neural network structure composed of a generator and a discriminator, designed to generate realistic data samples, such as images, text, or audio. I show the overview of the structure and training methods:

- Network Structure
  - Generator: The generator accepts a random noise vector as input and outputs new samples similar to the training data. It usually consists of a network that turns low-dimensional noise vectors to high-dimensional data space.

– Discriminator: The discriminator accepts real samples and fake samples generated by the generator as input and tries to distinguish them. It usually consists of a convolutional network that is used to classify whether the input sample is a real sample or a fake sample generated by the generator.

- Training Method

  During the training process, the generator and discriminator compete with each other and improve each other's performance through adversarial training.

  The training process is usually divided into two stages: first, the generator generates fake samples, and then the discriminator evaluates the authenticity of these fake samples. The feedback given by the discriminator is then used to update the parameters of the generator and discriminator.

  The goal of the generator is to generate samples that are as realistic as possible to fool the discriminator into thinking the generated samples are real. And the goal of the discriminator is to distinguish between real samples and fake samples generated by the generator as much as possible, so as to effectively detect fake samples generated by the generator.

3.3 Structure Analysis

Training the GAN involves playing a minmax game between the generator and discriminator. In other words, our optimization objective is

$$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_p[\log D(X)] + \mathbb{E}_q[\log(1 - D(G(Z)))] \tag{10}$$

3.3.1 Generator Layers

1. 100 input, `latent_dim`
2. $100 \rightarrow 128$ Linear Layer
3. LeakyRelu
4. $128 \rightarrow 256$ Linear Layer
5. BatchNorm, LeakyReLU
6. $256 \rightarrow 512$ Linear Layer
7. BatchNorm, LeakyReLU
8. $512 \rightarrow 1024$ Linear Layer
9. BatchNorm, LeakyReLU
10. $1024 \rightarrow 768$ Linear Layer
11. 768 output, image

3.3.2 Discriminator Layers

1. 784 input, a image
2. $784 \rightarrow 512$ Linear Layer
3. LeakyReLU
4. $512 \rightarrow 256$ Linear Layer
5. LeakyReLU
6. $256 \rightarrow 1$ Linear Layer
7. 1 output, means real or fake

4 Part 2 Task 2

4.1 Simple Introduction

In this part, I need sample 25 images from my trained GAN. And I will do this at the start of training, halfway through training and after training has terminated.

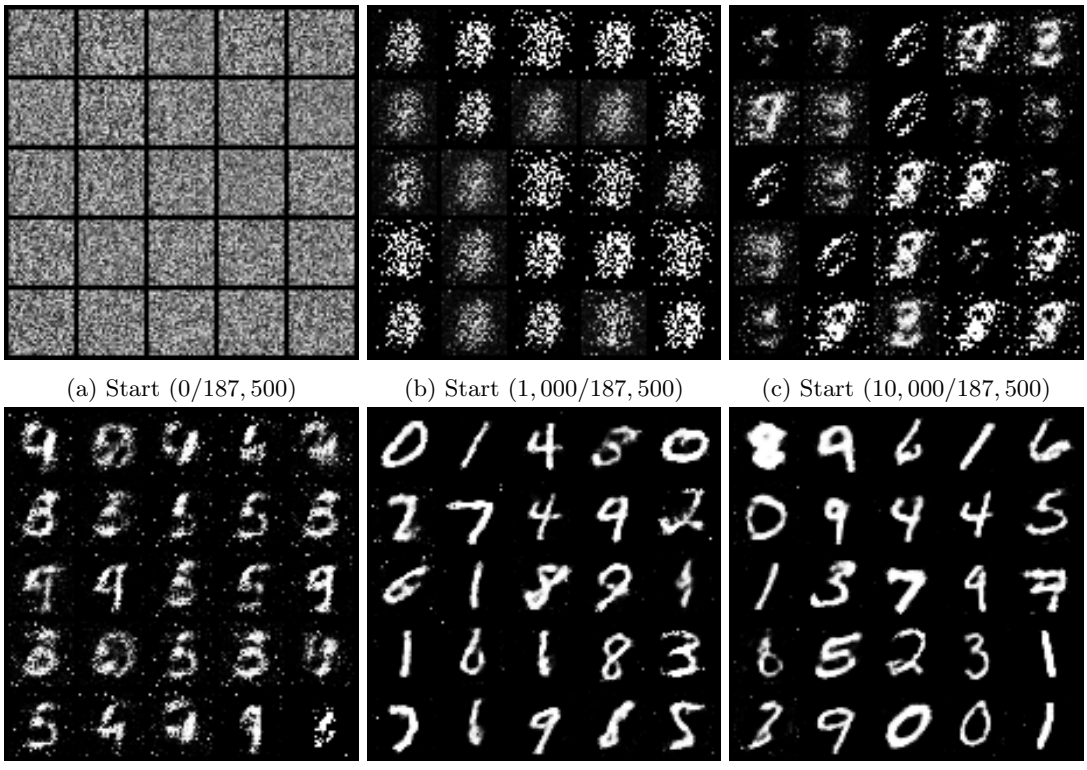
4.2 Default Parameters

- `n_epochs`: 200, maximum number of epochs to train the model
- `batch_size`: 64, batch size for training
- `lr`: 0.0002, learning rate for optimization
- `latent_dim`: 100, dimension of input noise
- `save_interval`: 500, interval for saving images generated.

I use `torch.optim.Adam()` for optimizer, `torch.nn.BCELoss()` for loss function.

4.3 Result Visualization

Fig 4 shows the results of 25 sample images at the start of training, halfway through training and after training has terminated.



(a) Start $(0/187,500)$ (b) Start $(1,000/187,500)$ (c) Start $(10,000/187,500)$

(d) Middle $(20,000/187,500)$ (e) Middle $(90,000/187,500)$ (f) End $(187,500/187,500)$

Figure 4: Task 2 Training Process

4.4 Result Analysis

4.4.1 Overall Analysis

It can be intuitively seen from the figure that: at the very beginning, the so-called "images" generated are just simple noise points, which have nothing to do with the MNIST handwriting data set

When the training reaches half (approximately $20,000 \sim 90,000$ iterations), the Generator has been able to generate images that are similar to the handwritten digits data set, but there is some noise and the shape of the handwritten digits is not very stable.

When the training is completely completed (approximately $180,000$ iterations), compared with the first two, there are fewer noise points, and the shape of the handwritten numbers is more specific and clear, indicating that the result of training is much better.

4.4.2 Training Analysis

Observing the output image, we can find that in the initial stage of training, the influence of learning is relatively great. The noise that was still meaningless at about $10,000$ iterations, had begun to take on a digital shape by $20,000$ iterations.

But comparing $90,000$ and $180,000$, the marginal diminishing effect appears clearly: compared with the previous training, the gap between them is not very obvious, and even after training for 180000 iterations, the generation of handwritten data still has not a completely clear part.

4.4.3 Network Analysis

One consideration is the structure of my GAN network itself. In fact, in this assignment , following the suggestions on the template, I used linear layers as the main part of the network. According to the comparison of Assignment 2, we already know that in problems related to image processing, convolutional layers often has better results than linear layers. So I think that if I use the convolutional generator structure such as Fig 3, I may get better image generation results.

5 Part 2 Task 3

5.1 Simple Introduction

In this part, I need to sample 2 images from my GAN. Interpolate between these two digits in latent space and include the results in my jupyter notebook. Use 7 interpolation steps, resulting in 9 images (including start and end point).

5.2 Generate Strategy Analysis

5.2.1 Strategy Introduction

In this part of the implementation, I first randomly generate two noises with length `latent_dim` as the begin and end noise, then interpolate 7 noises averagely between the two noises, and finally generate the results through the Generator.

5.2.2 Set notations

- k, `steps`, 9 total steps to generate
- n_1, `noise_begin`, begin noise
- n_k, `noise_end`, end noise

5.2.3 Define Expression

$$\alpha_i = \frac{i-1}{k} (1 \leq i \leq k) \tag{11}$$

$$n_i = (1 - \alpha_i) \cdot n_1 + \alpha_i \cdot n_k \tag{12}$$

Then the generator use noise $n_1 \dots n_k$ to generate k images respectively.

5.3 Result Visualization

Figure 5 shows some results.

(a) Case 1 (7 to 8)

(b) Case 2 (1 to 7)

(c) Case 3 (0 to 5)

Figure 5: Task 3 Sample Results

5.4 Result Analysis

5.4.1 Generate Analysis

We can see that if the first and last noises correspond to two different digital shapes, then the value in the middle will be generated as the shape between the two, showing a transition-like behavior.

In the process of generating images, the interpolation noise generated by my strategy may also be randomly generated directly. In this case, such a image looks like a number and another number. But the number image do not directly exist in the MNIST data set which is used for training. As a result, GAN is like generating a non-existent image out of thin air.

5.5 Application Analysis

With this extension of Assignment, we can imagine that GAN can be applied to generate realistic world images, including human faces, animals, natural scenery, etc. It can even learn a certain artistic style or type of pictures to generate new pictures similar to this style. In this way, GAN can also be used as an implementation method for the recently popular AI painting.

But this "learning" is also controversial. MNIST in this assignment is an academic open source data set, and it is understandable to use it for GAN training. But for works of an artist with obvious style, or facial portraits, will the copyright or portrait rights of others be infringed during the learning process?

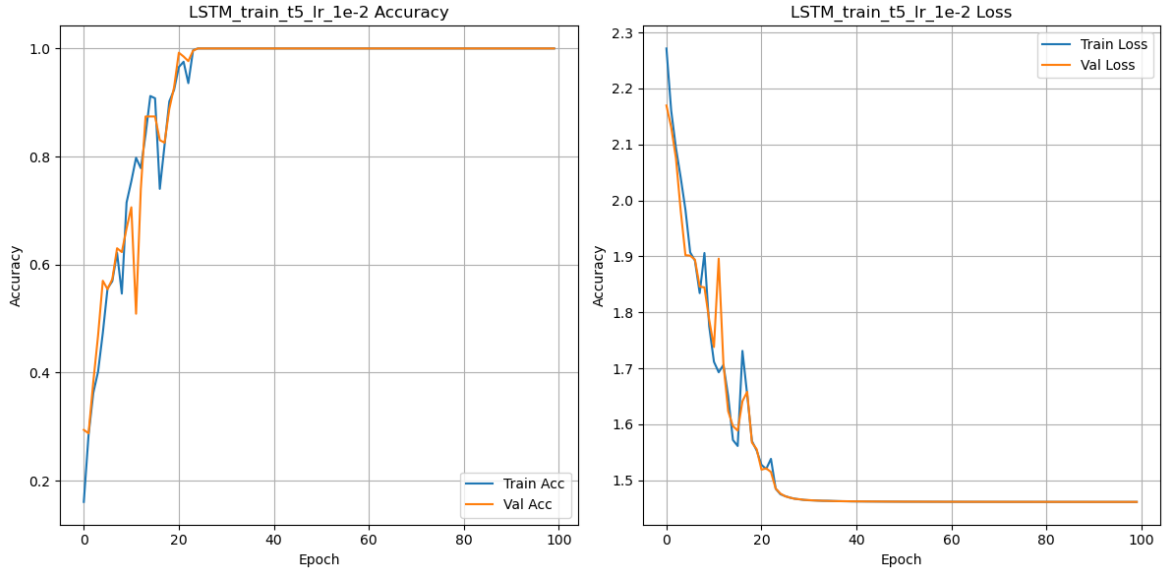
Although these problems are not directly related to the academic content of deep learning, it is also a point that has to make people think deeply.

References

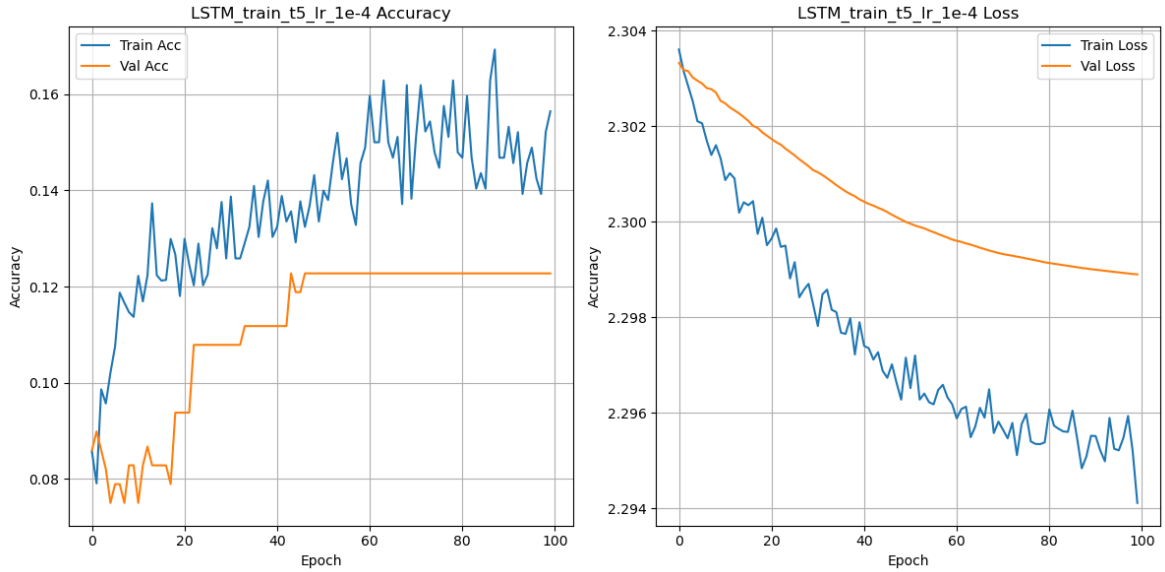
[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

A Extra Pictures

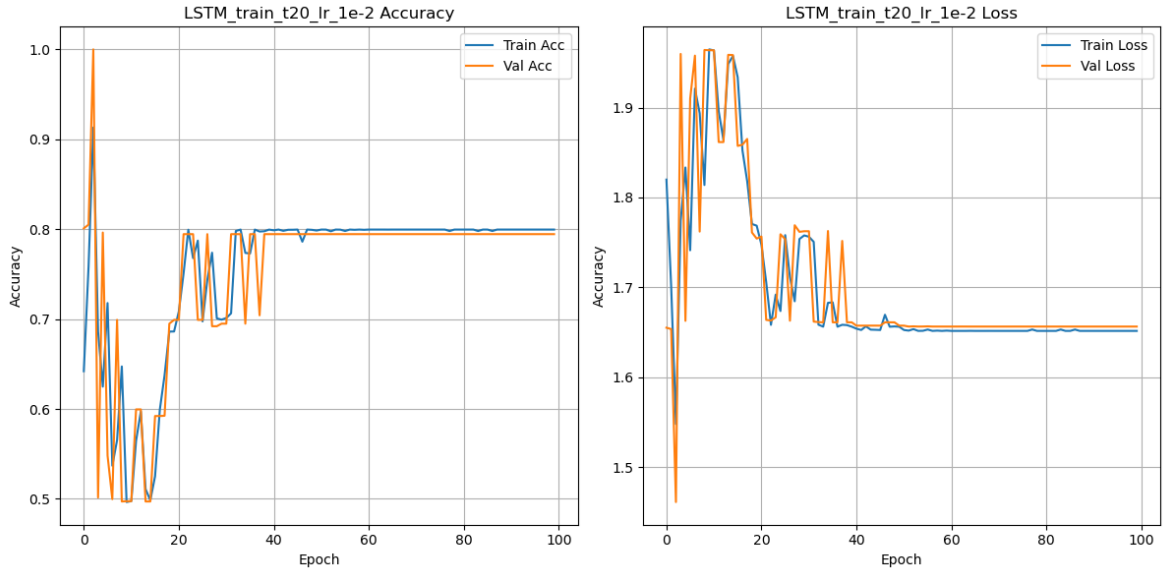


(a) Learning Rate $1e^{-2}$

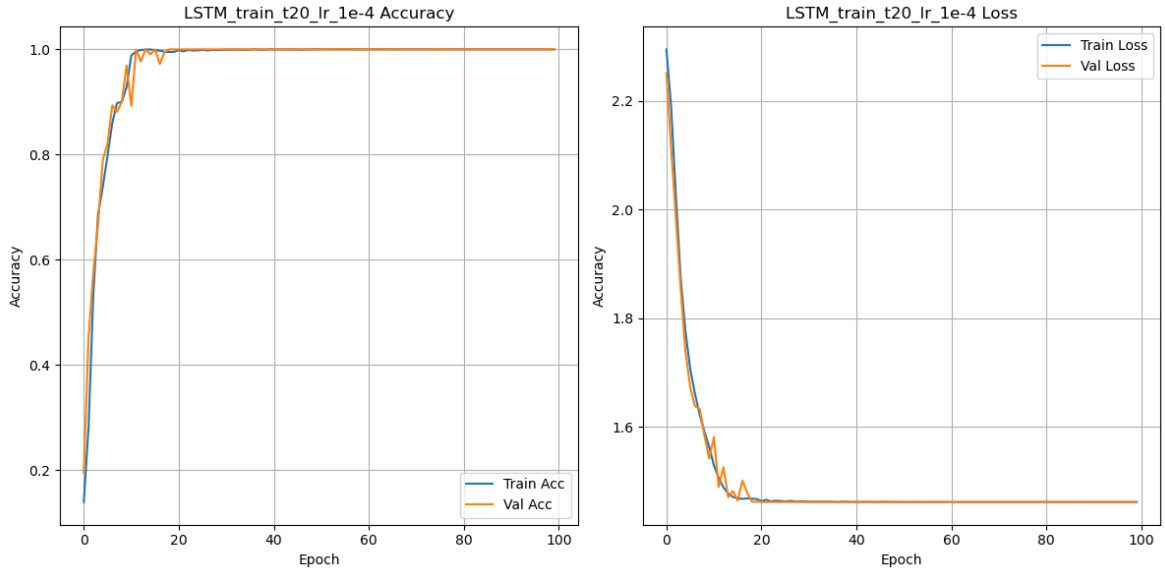


(b) Learning Rate $1e^{-4}$

Figure 6: $T = 5$ Change Learning Rate Curve



(a) Learning Rate $1e^{-2}$



(b) Learning Rate $1e^{-4}$

Figure 7: $T = 20$ Change Learning Rate Curve

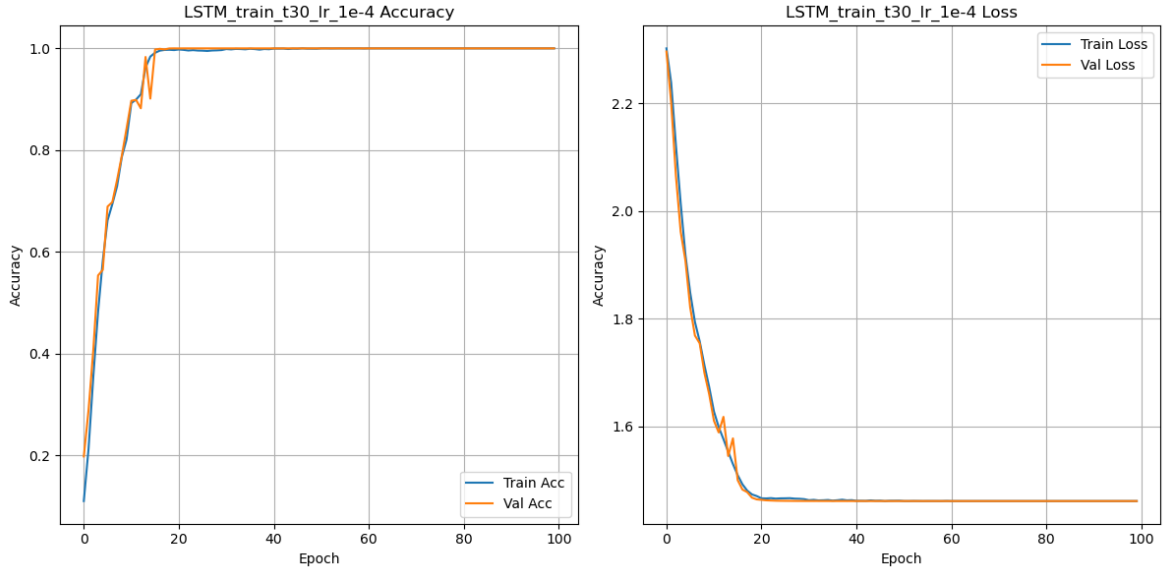


Figure 8: $T = 30$ Learning Rate $1e^{-4}$ Curve

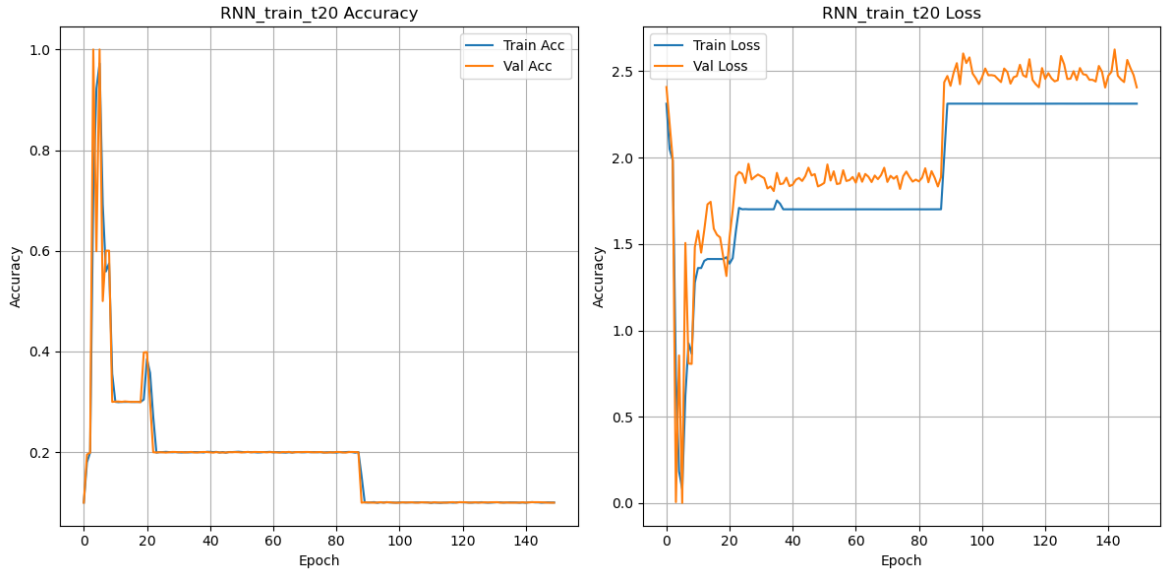


Figure 9: $T = 20$ RNN Curve