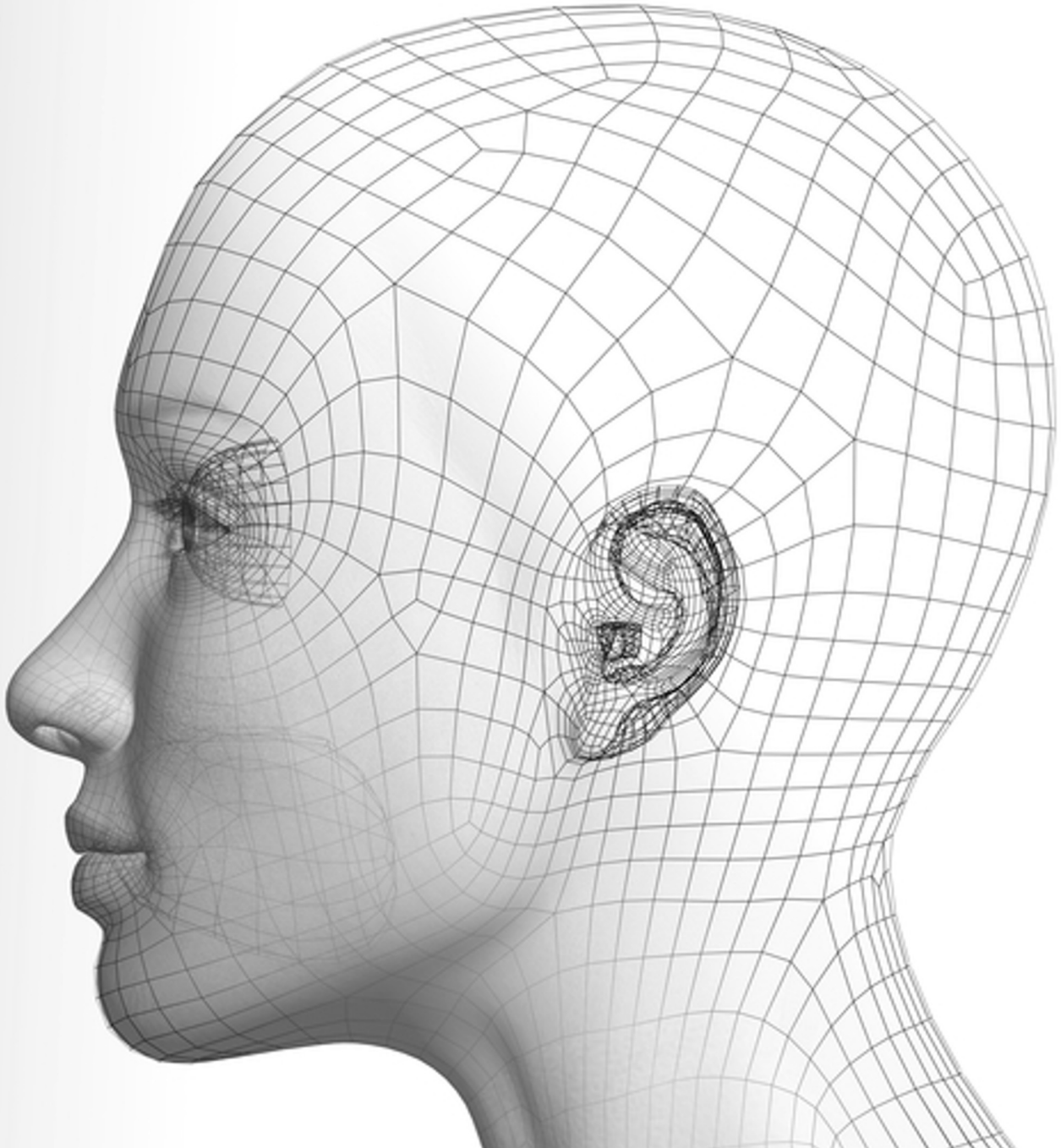
Tutorial 8 Recurrent Neural Networks

Chen Change Loy (Cavan)

https://www.mmlab-ntu.com/person/ccloy/

https://twitter.com/ccloy



Question 1

1. A recurrent neural network (RNN) receives sequences of 3-dimensional inputs and has two hidden neurons and one output neuron. The weight matrix $\boldsymbol{U}$ connecting the input to the hidden layer, the weight matrix $\boldsymbol{V}$ connecting the hidden output to the output layer, the hidden layer bias $\boldsymbol{b}$ and the output layer bias $\boldsymbol{c}$ are given by

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5.$$

Find the output sequence for an input sequence of $(\boldsymbol{x}(t))_{t=1}^{4}$ where

$$\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}, \boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}, \text{ and } \boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix} \text{if}$$

(a) The RNN is of hidden-recurrence (Elman) type with the recurrence weight matrix $\boldsymbol{W}$ connecting the previous hidden output to the current hidden layer input is given by

$$\boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 \\ 1.5 & 0.0 \end{pmatrix}.$$

Assume that the hidden activations are initialized to zero.

(b) The RNN is of top-down recurrence (Jordan) type with the weight matrix $\boldsymbol{W}$ connecting the previous output to the current state of hidden layer is given by

$$\boldsymbol{W} = (2.0 \quad 1.3).$$

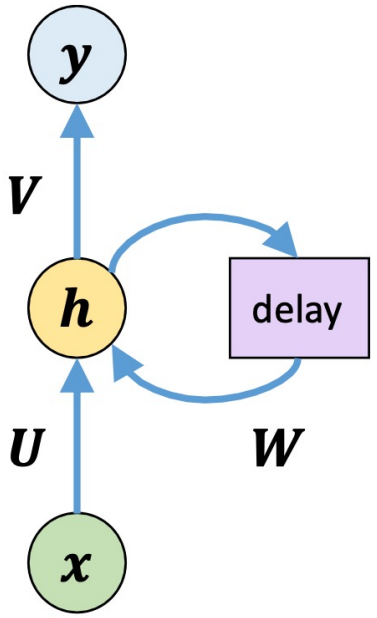
Assume that the output activations are initialized to zero.

t8q1.ipynb

Question 1

Hidden Recurrence:

$$\boldsymbol{U} = \begin{pmatrix} -1.0 & 0.5 \\ 0.5 & 0.1 \\ 0.2 & -2.0 \end{pmatrix}, \boldsymbol{V} = \begin{pmatrix} 2.0 \\ -1.5 \end{pmatrix} \text{ and } \boldsymbol{W} = \begin{pmatrix} 2.0 & 1.3 \\ 1.5 & 0.0 \end{pmatrix}.$$



$$\boldsymbol{b} = \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}, \text{ and } \boldsymbol{c} = 0.5$$

Three input neurons, two hidden neurons, and one output neuron.

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(t) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(t-1) + \boldsymbol{b}\right)$$
$$\boldsymbol{y}(t) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(t) + \boldsymbol{c}\right)$$

$$\phi(u) = \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$
$$\sigma(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}.$$

Assume $\boldsymbol{h}(0) = (0 \quad 0)^{T}$.

Question 1

At t=1, $\boldsymbol{x}(1) = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$:

$$\boldsymbol{h}(1) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(1) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(0) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 \\ 1.3 & 0.0 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix}$$

$$\boldsymbol{y}(1) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5)\begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix} + (0.5)\right) = (0.27)$$

Question 1

At t=2, $\boldsymbol{x}(2) = \begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix}$:

$$\boldsymbol{h}(2) = \phi\left(\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{x}(2) + \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{h}(1) + \boldsymbol{b}\right)$$

$$= \tanh\left(\begin{pmatrix} -1.0 & 0.5 & 0.2 \\ 0.5 & 0.1 & -2.0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \\ -2 \end{pmatrix} + \begin{pmatrix} 2.0 & 1.5 \\ 1.3 & 0.0 \end{pmatrix}\begin{pmatrix} 0.0 \\ 0.99 \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.99 \\ 1.0 \end{pmatrix}$$

$$\boldsymbol{y}(2) = \sigma\left(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{h}(2) + \boldsymbol{c}\right) = \text{sigmoid}\left((2.0 \quad -1.5)\begin{pmatrix} 0.99 \\ 1.0 \end{pmatrix} + (0.5)\right) = (0.73)$$

Similarly,

at t=3, $\boldsymbol{x}(3) = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}$; $\boldsymbol{h}(3) = \begin{pmatrix} 1.0 \\ -0.21 \end{pmatrix}$ and $\boldsymbol{y}(3) = (0.94)$

at t=4, $\boldsymbol{x}(4) = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}$; $\boldsymbol{h}(4) = \begin{pmatrix} -0.54 \\ 0.98 \end{pmatrix}$ and $\boldsymbol{y}(4) = (0.11)$

Question 1

The input sequence $\left(\boldsymbol{x}(1), \boldsymbol{x}(2), \boldsymbol{x}(3), \boldsymbol{x}(4)\right)$:

$$\left(\begin{pmatrix}1\\2\\-1\end{pmatrix} \quad \begin{pmatrix}-1\\1\\-2\end{pmatrix} \quad \begin{pmatrix}0\\3\\1\end{pmatrix} \quad \begin{pmatrix}2\\-1\\0\end{pmatrix}\right)$$

The output sequence $\left(\boldsymbol{y}(1), \boldsymbol{y}(2), \boldsymbol{y}(3), \boldsymbol{y}(4)\right)$:

$$((0.27) \quad (0.73) \quad (0.94) \quad (0.11))$$

Question 2

2. An RNN receives 3-dimensional input sequence and produces 2-dimensional output sequence. It has 5 hidden neurons and receives sequences of 100 time steps.

   - Generate 8 training input sequences by drawing values uniformly between 0 and 1.0.
   - If the sequence $\left(\boldsymbol{y}(t)\right)_{t=1}^{100}$ where $\boldsymbol{y}(t) = \left(y_k(t)\right)_{k=1}^{2} \in \boldsymbol{R}^2$ denotes the output sequence for an input sequence $\left(\boldsymbol{x}(t)\right)_{t=1}^{100}$ where $\boldsymbol{x}(t) = \left(x_i(t)\right)_{i=1}^{3} \in \boldsymbol{R}^3$, generate the corresponding output sequences for the input training sequences as follows:

$$y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon$$
$$y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon$$

   where ε is standard normally distributed random variable.

Train an RNN to learn the above sequences by using gradient descent learning. Use a learning factor $\alpha = 0.01$ and Adam optimizer.

t8q2.ipynb

Question 2

Set random seed and initialize hyper-parameters

0s

```
[2] # Set random seed for reproducibility
    seed = 10
    np.random.seed(seed)
    torch.manual_seed(seed)
```

```
<torch._C.Generator at 0x7eeddd747dd0>
```

0s

```
[3] # Initialize hyper-parameters
    n_in = 3
    n_hidden = 5
    n_out = 2
    n_steps = 100
    n_seqs = 8
    n_iters = 10000
    lr = 0.01
```

Question 2

Generate training data

```
[4]  x_train = np.random.rand(n_seqs, n_steps, n_in)
     y_train = np.zeros([n_seqs, n_steps, n_out])

     y_train[:, 1:, 0] = 5 * x_train[:, 1:, 0] - 0.2 * x_train[:, :-1, 2]
     y_train[:, 3:, 1] = 25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]
     y_train += 0.1 * np.random.randn(n_seqs, n_steps, n_out)

     x_train = torch.tensor(x_train, dtype=torch.float32)
     y_train = torch.tensor(y_train, dtype=torch.float32)
```

0s

1. **Random Input Data (`x_train`)**:
   - `x_train = np.random.rand(n_seqs, n_steps, n_in)` generates random input data from a uniform distribution, with dimensions corresponding to the number of sequences (`n_seqs`), the number of time steps (`n_steps`), and the number of input features (`n_in`).
2. **Zero-initialized Output Data (`y_train`)**:
   - `y_train = np.zeros([n_seqs, n_steps, n_out])` initializes an output tensor with zeros. The shape of `y_train` matches the number of sequences, time steps, and output features (`n_out`).

Question 2

The basic slice syntax is i:j:k where *i* is the starting index, *j* is the stopping index, and *k* is the step.

y_train[:,3:,1] = [$y_2(3), y_2(4), y_2(5) \ \ldots, y_2(99)$]

x_train[:,2:-1,1] = [$x_2(2), x_2(3), x_2(4) \ \ldots, x_2(98)$]

x_train[:, :-3,2] = [$x_3(0), x_3(1), x_3(2) \ \ldots, x_3(96)$]

Assume *n* is the number of elements in the dimension being sliced

If *j* is not given it defaults to *n*

If *k* is not given it defaults to 1

Negative *i* and *j* are interpreted as *n* + *i* and *n* + *j* where *n* is the number of elements in the corresponding dimension

Question 2

Generate training data

```
[4]  x_train = np.random.rand(n_seqs, n_steps, n_in)
     y_train = np.zeros([n_seqs, n_steps, n_out])

     y_train[:, 1:, 0] = 5 * x_train[:, 1:, 0] - 0.2 * x_train[:, :-1, 2]
     y_train[:, 3:, 1] = 25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]
     y_train += 0.1 * np.random.randn(n_seqs, n_steps, n_out)

     x_train = torch.tensor(x_train, dtype=torch.float32)
     y_train = torch.tensor(y_train, dtype=torch.float32)
```

$$\begin{cases} y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon \\ y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon \end{cases}$$

3. **Applying the First Equation** (`y1(t)`):
   - `y_train[:, 1:, 0] = 5 * x_train[:, 1:, 0] - 0.2 * x_train[:, :-1, 2]` corresponds to the first equation for `y1(t)` :

     $$y_1(t) = 5x_1(t) - 0.2x_3(t-1)$$

     This operation assigns the values to the first output feature (index `0` in the last dimension) at time steps `t >= 1` . The term `5 * x_train[:, 1:, 0]` scales the `x1(t)` input, and `-0.2 * x_train[:, :-1, 2]` incorporates the lagged (shifted by one step) values from `x3(t-1)` .

Question 2

Generate training data

```
[4]  x_train = np.random.rand(n_seqs, n_steps, n_in)
     y_train = np.zeros([n_seqs, n_steps, n_out])

     y_train[:, 1:, 0] = 5 * x_train[:, 1:, 0] - 0.2 * x_train[:, :-1, 2]
     y_train[:, 3:, 1] = 25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]
     y_train += 0.1 * np.random.randn(n_seqs, n_steps, n_out)

     x_train = torch.tensor(x_train, dtype=torch.float32)
     y_train = torch.tensor(y_train, dtype=torch.float32)
```

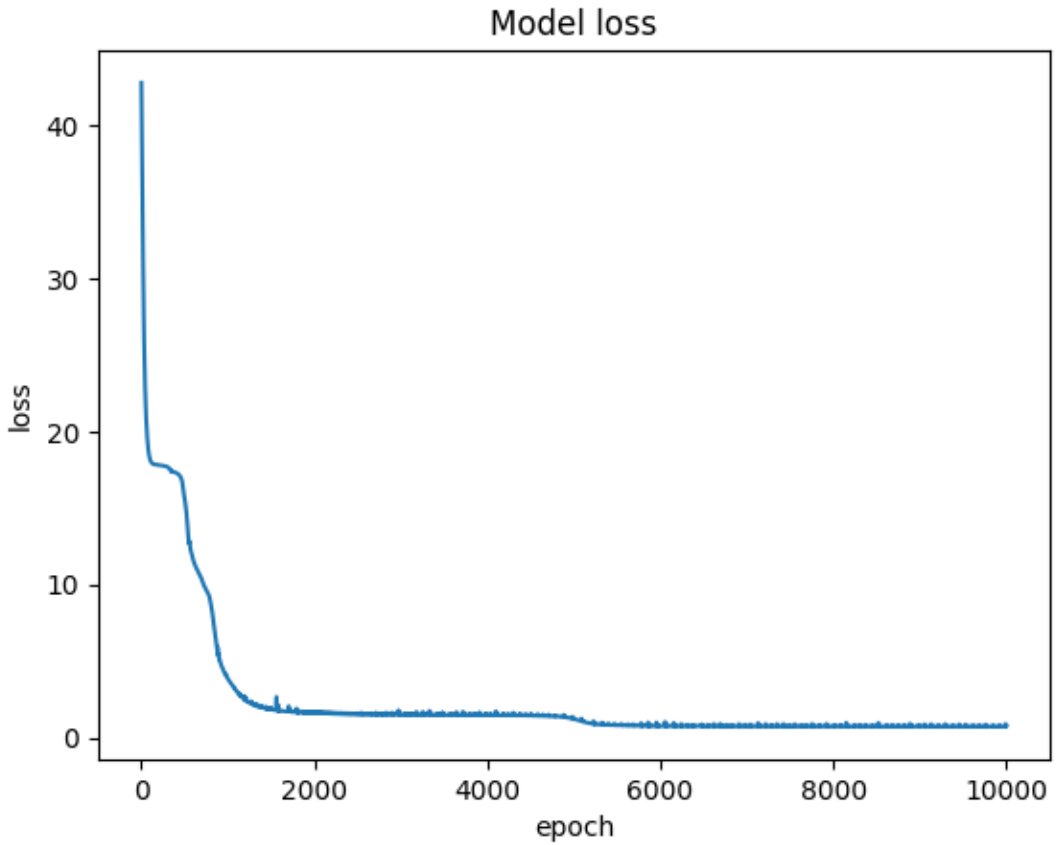
$$\begin{cases} y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon \\ y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon \end{cases}$$

4. **Applying the Second Equation** (`y2(t)`):
   - `y_train[:, 3:, 1] = 25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]` corresponds to the second equation for `y2(t)` :

     $$y_2(t) = 25x_2(t-1)x_3(t-3)$$

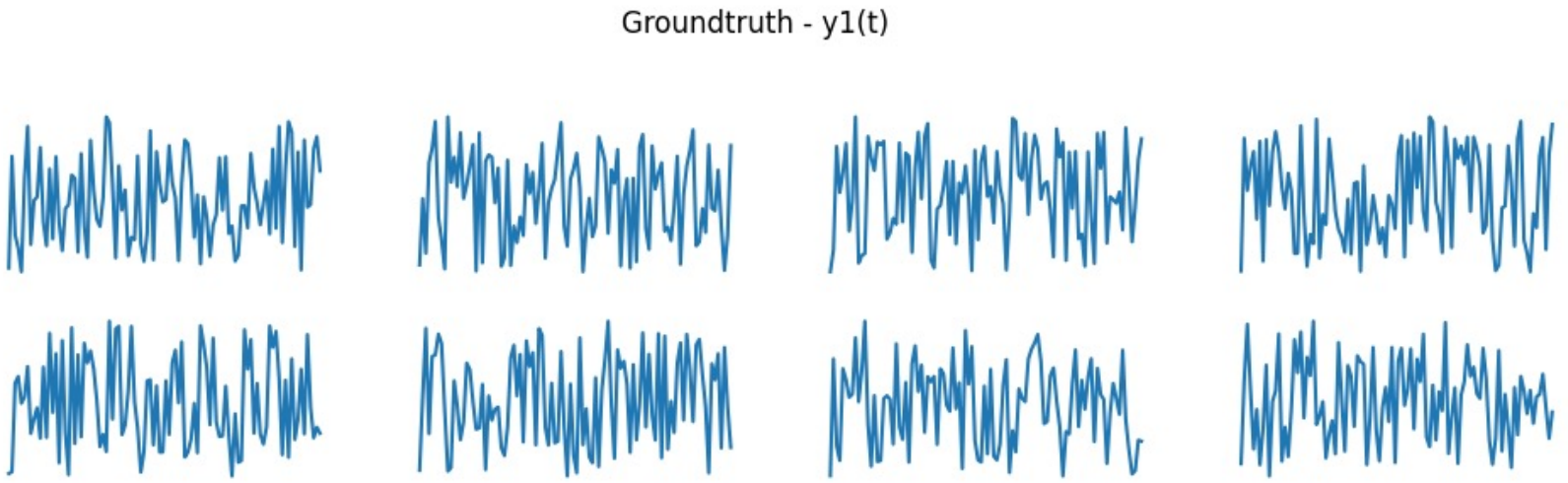
     This assigns values to the second output feature (index `1` in the last dimension) starting from time step `t >= 3` . The term `25 * x_train[:, 2:-1, 1] * x_train[:, :-3, 2]` multiplies the values of `x2(t-1)` and `x3(t-3)` .

Question 2



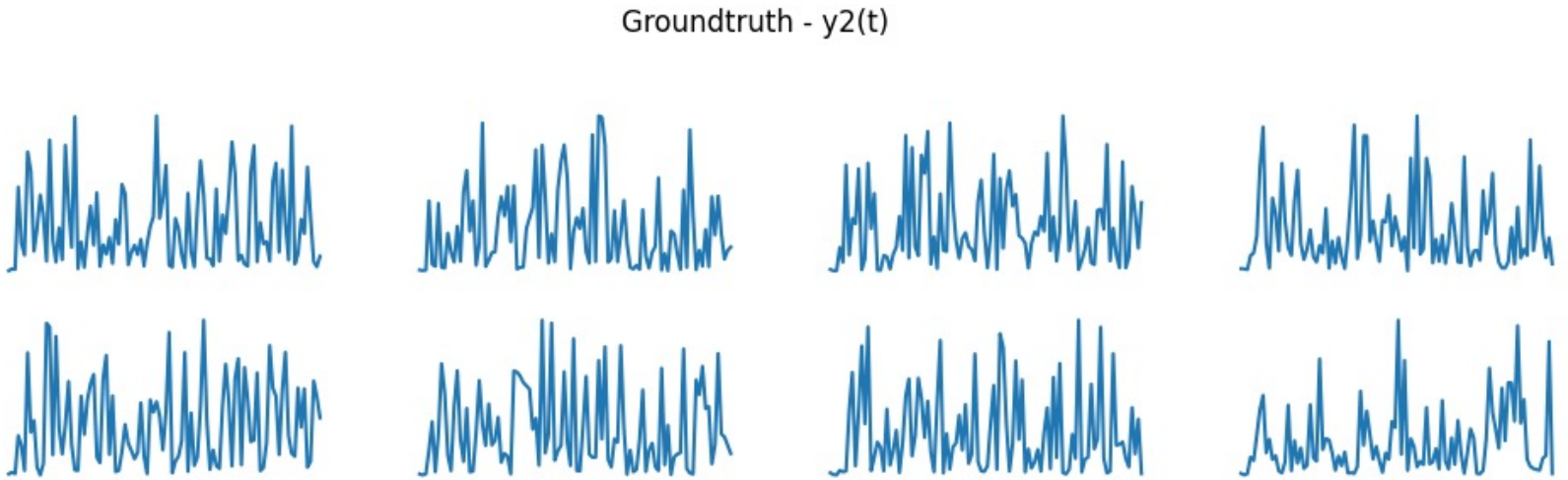
Question 2

$$y_1(t) = 5x_1(t) - 0.2x_3(t-1) + 0.1\varepsilon$$

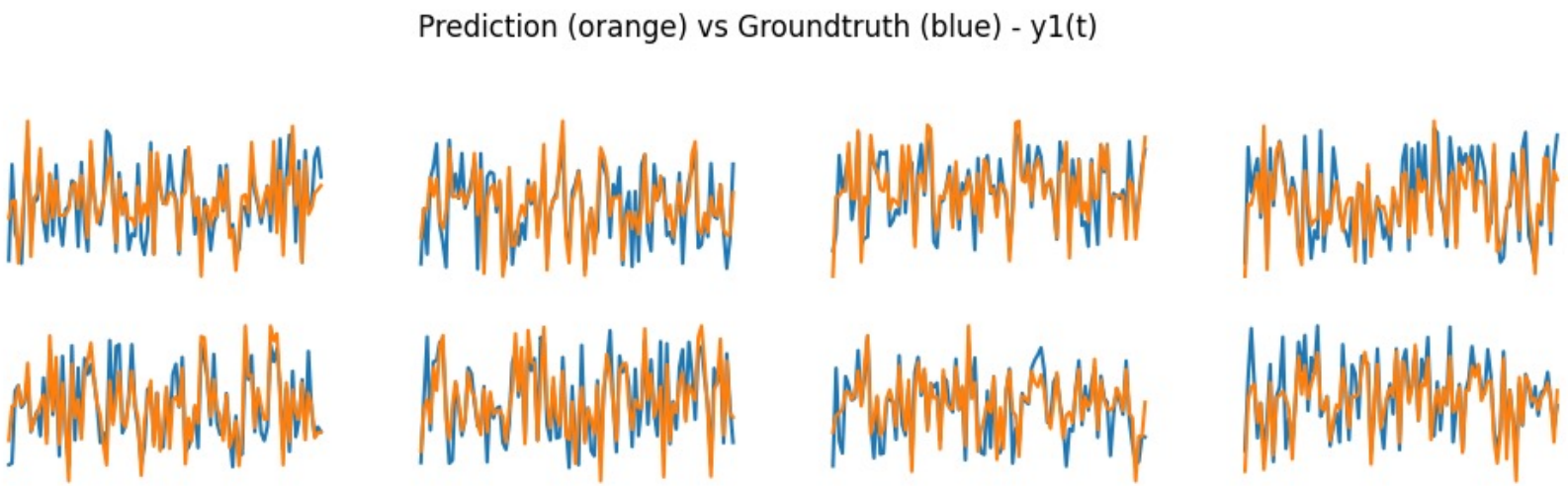


Question 2

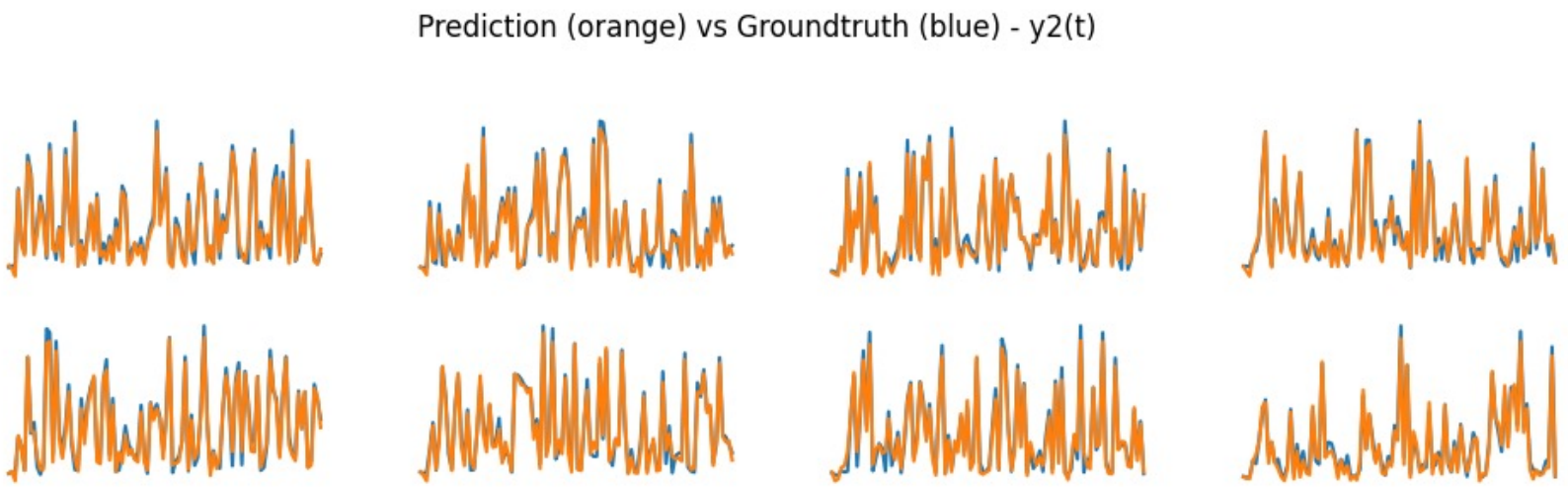
$$y_2(t) = 25x_2(t-1)x_3(t-3) + 0.1\varepsilon$$



Question 2



Question 2



Question 3

3. Design an LSTM layer with 10 units to map the following input and output sequences:

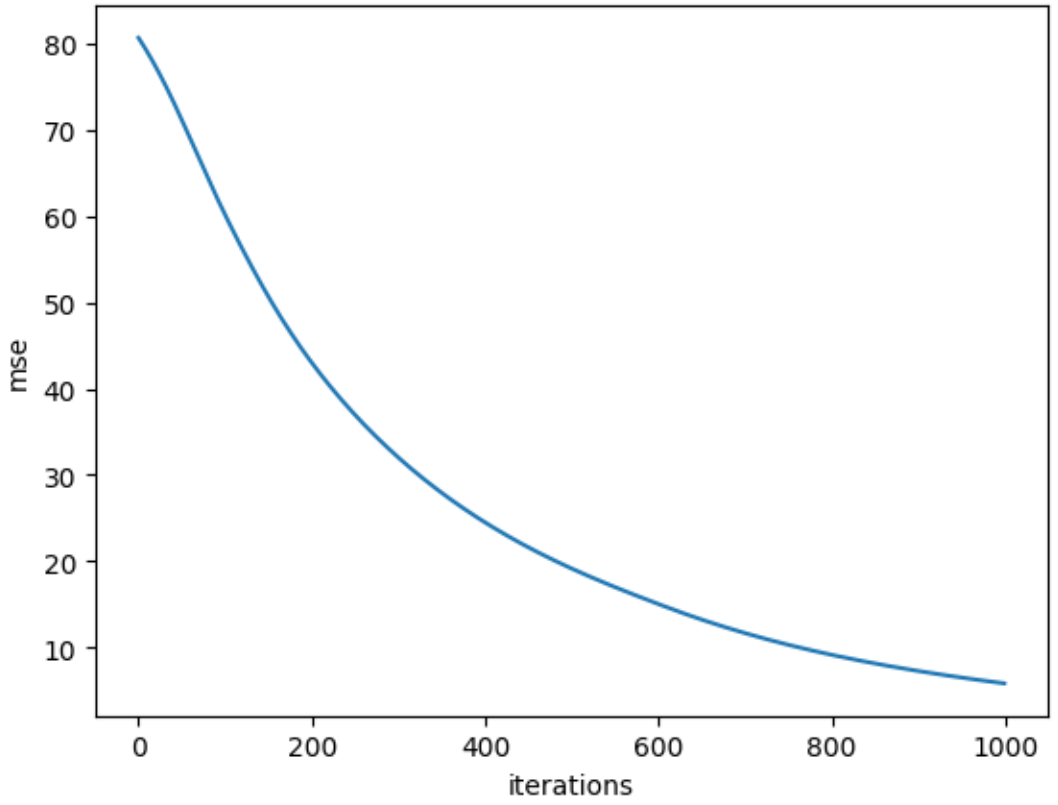
Input $\boldsymbol{x}$	Output $\boldsymbol{y}$
(1 2 5 6)	(1 3 7 11)
(5 7 7 8)	(5 12 14 15)
(3 4 5 7)	(3 7 9 12)

Plot the learning curves at a rate $\alpha = 0.001$.

Find the output sequences for the following input sequences:

(1 2 3 4)
(4 5 6 7)

Question 3




```
test_x = torch.tensor(
    [[[1], [2], [3], [4]],
    [[4], [5], [6], [7]]], dtype=torch.float32)
out = predictor(test_x)
print(out)
```

```
tensor([[0.9988, 2.7404, 6.3064, 8.8883], [4.6289,
8.4632, 9.4166, 9.5583]], grad_fn=<SqueezeBackward1>)
```

Question 4

4. Design a character RNN layer with 10 GRU units to learn the sentiments about a movie, given in table 1.

Convert the input text into character ids and set the maximum text length to be 40. Train the network using gradient descent learning with the Adam optimizer and at a learning rate $\alpha = 0.001$. Use dropouts at the hidden layer of GRU at a probability $p = 0.7$.

Plot the cost and the accuracies against epochs during training.

Question 4

Table 1

Input	*Sentiment*
I did not like the movie	negative
The movie was not good	negative
I watched the movie with great interest	positive
I have seen better movies	negative
Good to see that movie	positive
I am not a fan of movies	negative
I liked the movie great	positive
The movie was of interest to me	positive
I thought they could show interesting scenes	negative
The movie did not have good scenes	negative
Family did not like the movie at all	negative

After training, find the likelihoods of the sentiments of the following statements:

The movie was not interesting to me

I liked the movie with great interest

Question 4

```
[ ] import numpy as np
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import matplotlib.pyplot as plt

    import os
    if not os.path.isdir('figures'):
        print('creating the figures folder')
        os.makedirs('figures')
```

```
creating the figures folder
```

Create Training and Test Data

```
data = ['I did not like the movie',
        'The movie was not good',
        'I watched the movie with great interest',
        'I have seen better movies',
        'Good to see that movie',
        'I am not a fan of movies',
        'I liked the movie great',
        'The movie was of interest to me',
        'I thought they could show interesting scenes',
        'The movie did not have good scenes',
        'Family did not like the movie at all']

targets = [0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0]

test_data = ['The movie was not interesting to me',
        'I liked the movie with great interest']
```

Question 4

```
[ ]  def vocabulary(strings):
         chars = sorted(list(set(list(''.join(strings).lower()))))
         char_to_ix = {ch: i for i, ch in enumerate(chars)}
         vocab_size = len(chars)
         return vocab_size, char_to_ix
```

`vocabulary` Function:

This function constructs a vocabulary from a list of strings.

1. **Parameters**:
   - `strings`: A list of strings from which the vocabulary is constructed.
2. **Functionality**:
   - All characters from the input strings are extracted, converted to lowercase, and then a set is created to remove duplicates.
   - The unique characters are sorted to create a consistent ordering.
   - A dictionary (`char_to_ix`) is created to map each character to a unique integer ID.
3. **Return**:
   - `vocab_size`: The total number of unique characters in the vocabulary.
   - `char_to_ix`: A dictionary mapping each character to its unique integer ID.

```
my_set = set()  # Creates an empty set
    my_set = set([1, 2, 3, 2])  # Creates a set {1, 2, 3} (duplicates are removed)
```

Example:

If `chars` was the list `['a', 'b', 'c']`, then the `char_to_ix` dictionary would be:

```
{'a': 0, 'b': 1, 'c': 2}
```

Use code with caution

This dictionary can then be used to easily convert characters to their corresponding numerical IDs, which is often necessary for processing text data in machine learning.

Question 4

```python
def preprocess(strings, char_to_ix):
    data_chars = [list(d.lower()) for _, d in enumerate(strings)]
    for i, d in enumerate(data_chars):
        if len(d) > MAX_LENGTH:
            d = d[:MAX_LENGTH]
        elif len(d) < MAX_LENGTH:
            d += [' '] * (MAX_LENGTH - len(d))
    data_ids = np.zeros([len(data_chars), MAX_LENGTH], dtype=np.int64)
    for i in range(len(data_chars)):
        for j in range(MAX_LENGTH):
            data_ids[i, j] = char_to_ix[data_chars[i][j]]
    return torch.tensor(data_ids, dtype=torch.int64)
```

`preprocess` Function:

This function converts a list of strings into a tensor of integer IDs based on a given character-to-ID mapping.

1. **Parameters**:
   - `strings`: A list of strings to be preprocessed.
   - `char_to_ix`: A dictionary mapping characters to their unique integer IDs.
2. **Functionality**:
   - Each string in the input list is converted to lowercase.
   - Strings are truncated or padded to match the `MAX_LENGTH`:
     - If a string's length exceeds `MAX_LENGTH`, it's truncated.
     - If a string's length is less than `MAX_LENGTH`, it's padded with spaces.
   - Each character in the processed strings is then converted to its corresponding integer ID using the `char_to_ix` mapping.
3. **Return**:
   - A tensor of shape `[number of strings, MAX_LENGTH]`, where each entry represents the integer ID of a character.

Here's a step-by-step explanation:

1. **`len(d)`**: This calculates the length of the current string `d`.
2. **`(MAX_LENGTH - len(d))`**: This subtracts the string's length from `MAX_LENGTH`, determining how many spaces are needed to pad the string to the desired length.
3. **`[' '] * (MAX_LENGTH - len(d))`**: This creates a list of spaces. The `*` operator is used to repeat the single-element list `[' ']` by the number of spaces calculated in the previous step.
4. **`d += ...`**: This is the core of the operation. The `+=` operator is used to append the list of spaces created in the previous step to the current string `d`. This effectively pads the string with spaces to reach the `MAX_LENGTH`.

Why having fixed length input in RNN?

- RNNs theoretically can handle variable-length sequences, yet training frameworks usually require fixed-length inputs.
- During training, we process many sequences together in mini-batches for GPU efficiency. However, GPUs require tensors of uniform shape. They can’t handle batches where one sequence is length 10 and another is 120.
- Frameworks like PyTorch and TensorFlow allocate computation graphs statically per batch:
  - The graph structure depends on the sequence length.
  - Fixed lengths allow the same graph to be reused efficiently.

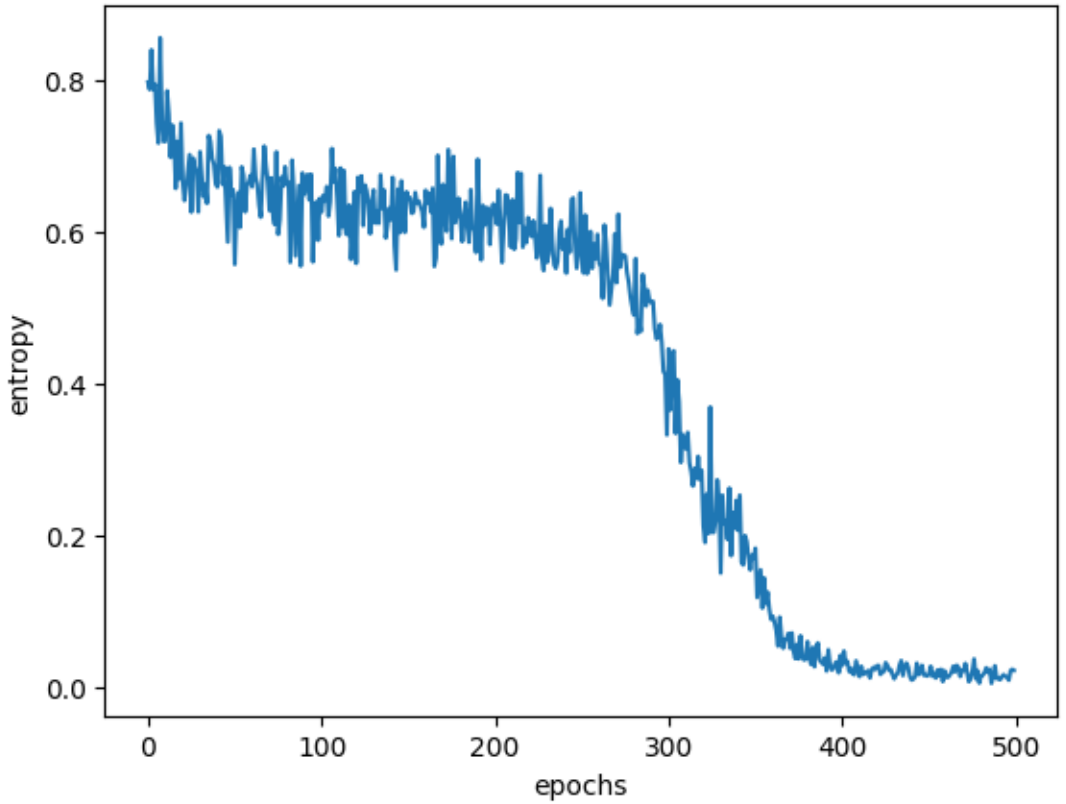
Question 4

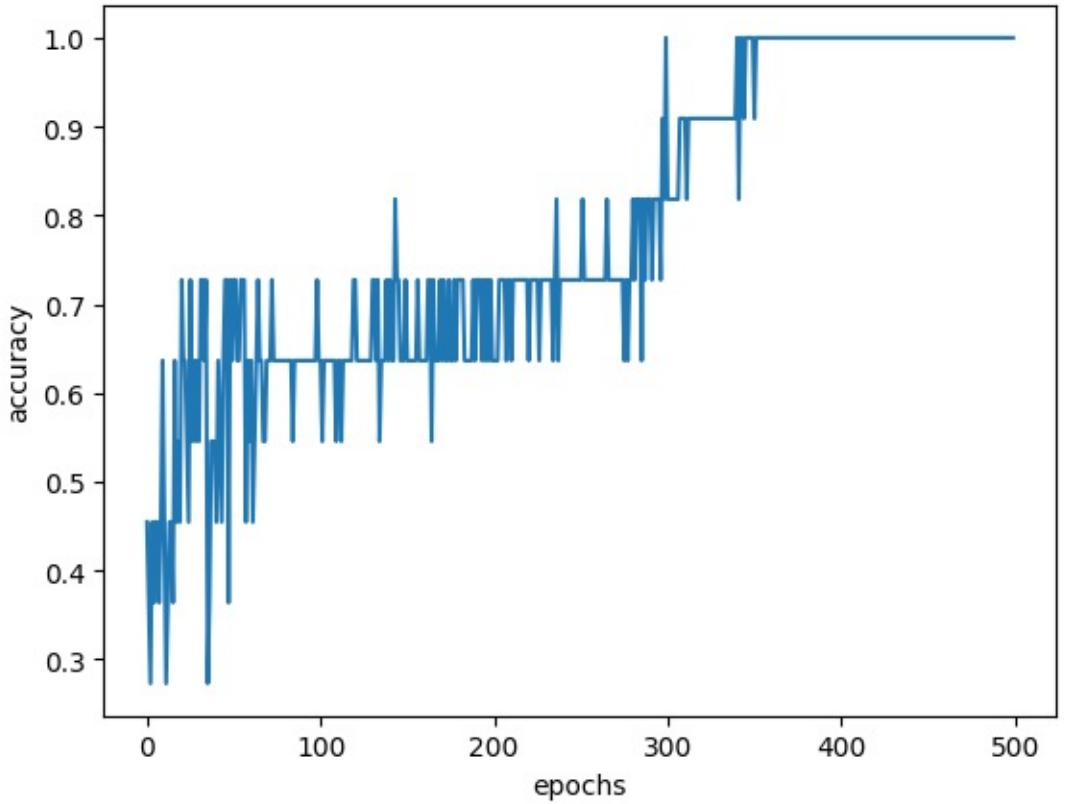
Note the use of the loss function

- CrossEntropyLoss combines nn.LogSoftmax and nn.NLLLoss (negative log-likelihood loss), meaning you don't need to explicitly apply nn.Softmax as a final layer. You can remove the nn.Softmax layer, and nn.CrossEntropyLoss will take care of both the softmax activation and the log-likelihood calculation.

- If the network ends with 'nn.Softmax', you can pair it with nn.NLLLoss, which takes the log probabilities as input, so you would need to take the log of the output of the nn.Softmax layer before passing it to the loss function.

Question 4


entropy
0.0
0.2
0.4
0.6
0.8
0
100
200
300
400
500
epochs


accuracy
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
100
200
300
400
500
epochs

Question 4

```
test_data = ['The movie was not interesting to me',
'I liked the movie with great interest']
```

```
test_logits = model(x_test, drop_rate=0)
probs = nn.functional.softmax(test_logits, dim=1)
print(probs)
```

```
tensor(
[[0.0119, 0.9881],
[0.4359, 0.5641]])
```

‘The movie was not interesting to me’ is a negative sentiment with a probability of 0.9881

‘I liked the movie with great interest’ is a positive sentiment with a probability of 0.4359 (hasn’t trained well)