Assignment 2: Arithmetic as a language

2024 NTHU Natural Language Processing

Hung-Yu Kao

IKM Lab TAs

Assignment Description

In assignment 2, you will practice training simple sequence generation models. We will treat **arithmetic expressions as a language** and use recurrent neural networks (RNN, LSTM) to train a sequence generation model for this special language.
In this assignment, you will practice training and analyzing a neural network model, as well as reflect on the model's logical understanding of arithmetic operations.

Train a model

Step1: Prepare the dataset

Step2: Construct the model

Step3: Define Optimizer

Step4: Define loss function

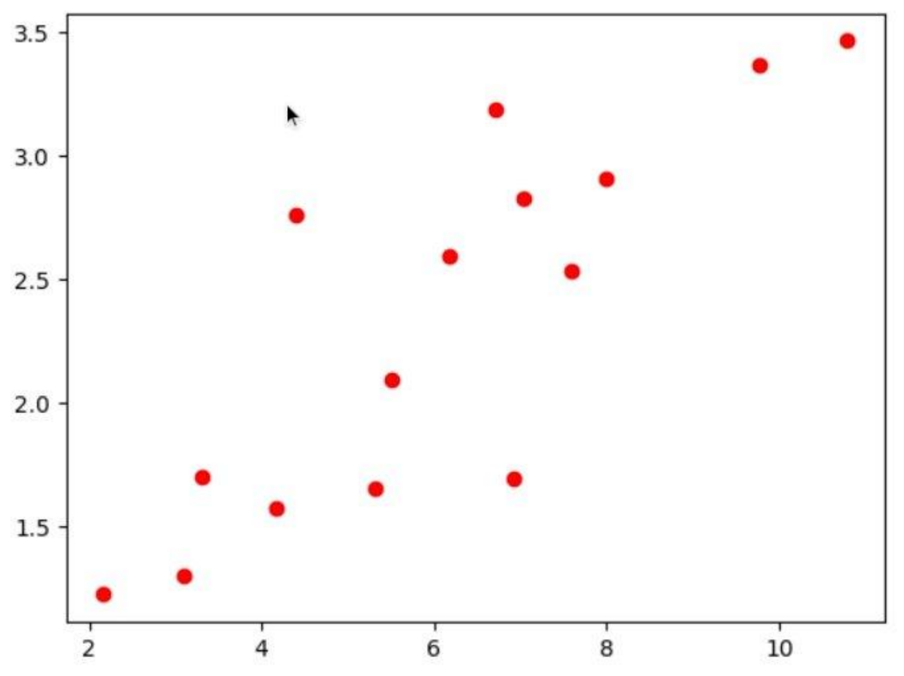
Step5: Train the model

Step6: Evaluate the model

1. Input data to the model
2. compute loss
3. clear gradients
4. compute gradients
5. optimize parameters
6. back to 1.

A simple example of linear regression

- Data distribution:
- Use a line to represent these data



Pytorch code of linear regression

```python
# Toy dataset
x_train = torch.tensor([[3.3], [4.4], [5.5], [6.71], [6.93], [4.168],
                        [9.779], [6.182], [7.59], [2.167], [7.042],
                        [10.791], [5.313], [7.997], [3.1]], dtype=torch.float32)

y_train = torch.tensor([[1.7], [2.76], [2.09], [3.19], [1.694], [1.573],
                        [3.366], [2.596], [2.53], [1.221], [2.827],
                        [3.465], [1.65], [2.904], [1.3]], dtype=torch.float32)
```

initialize data tensor

```python
# Linear regression model
model = nn.Linear(input_size, output_size)
```

model

```python
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

loss & optimizer

```python
# Train the model
for epoch in range(num_epochs):

    # Forward pass
    outputs = model(x_train)
    loss = criterion(outputs, y_train)

    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 20 == 0:
        print ('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.item()))
```

load data

1. Input data to the model
2. compute loss
3. clear gradients
4. compute gradients
5. optimize parameters

Forward & Back-propagation Insight (1/2)

step 1

```
optimizer.zero_grad()
print_grads(model)
```

```
weight: Parameter containing:
tensor([[0.4165]], requires_grad=True)
weight grad: None

bias: Parameter containing:
tensor([0.4819], requires_grad=True)
bias grad: None
```

step 2

```
loss.backward()
print_grads(model)
```

```
weight: Parameter containing:
tensor([[0.4165]], requires_grad=True)
weight grad: tensor([[10.0239]])

bias: Parameter containing:
tensor([0.4819], requires_grad=True)
bias grad: tensor([1.3666])
```

Forward & Back-propagation Insight (2/2)

step 2

```
loss.backward()
print_grads(model)
```

```
weight: Parameter containing:
tensor([[0.4165]], requires_grad=True)
weight grad: tensor([[10.0239]])

bias: Parameter containing:
tensor([0.4819], requires_grad=True)
bias grad: tensor([1.3666])
```

step 3

```
optimizer.step()
print_grads(model)
```

```
weight: Parameter containing:
tensor([[0.4065]], requires_grad=True)
weight grad: tensor([[10.0239]])

bias: Parameter containing:
tensor([0.4805], requires_grad=True)
bias grad: tensor([1.3666])
```

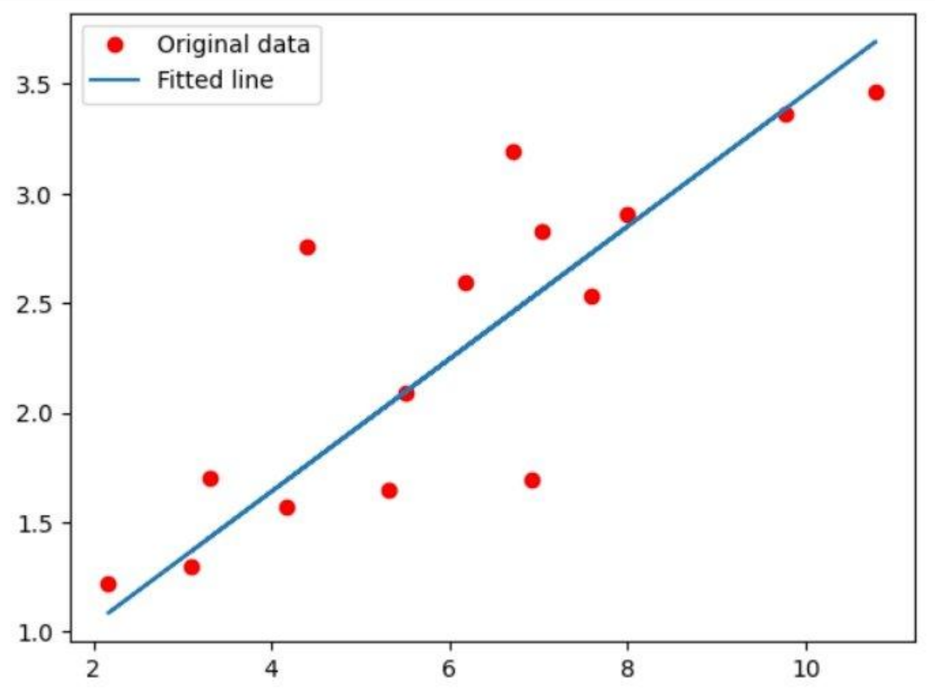
step 4

```
optimizer.zero_grad()
print_grads(model)
```

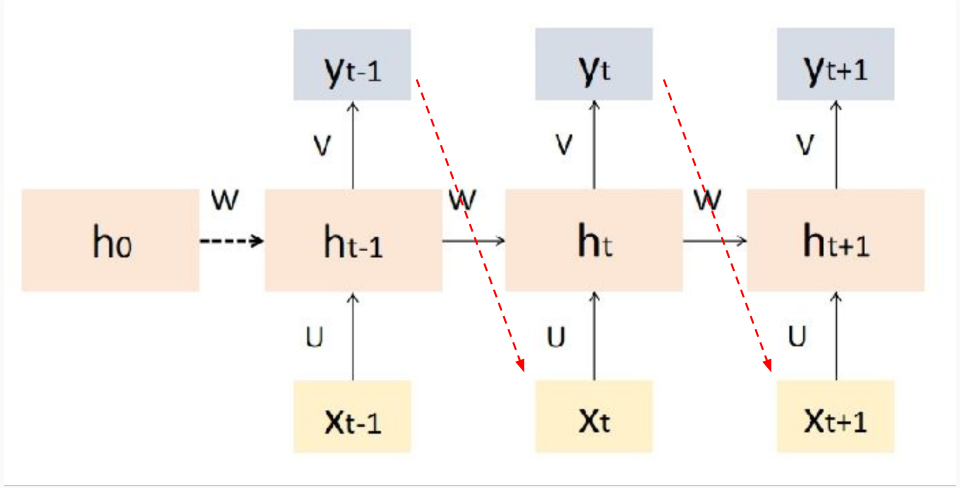
```
weight: Parameter containing:
tensor([[0.4065]], requires_grad=True)
weight grad: None

bias: Parameter containing:
tensor([0.4805], requires_grad=True)
bias grad: None
```

Outputs

```
Epoch [5/60], Loss: 11.2489
Epoch [10/60], Loss: 4.6657
Epoch [15/60], Loss: 1.9987
Epoch [20/60], Loss: 0.9182
Epoch [25/60], Loss: 0.4805
Epoch [30/60], Loss: 0.3031
Epoch [35/60], Loss: 0.2313
Epoch [40/60], Loss: 0.2021
Epoch [45/60], Loss: 0.1903
Epoch [50/60], Loss: 0.1855
Epoch [55/60], Loss: 0.1835
Epoch [60/60], Loss: 0.1827
```



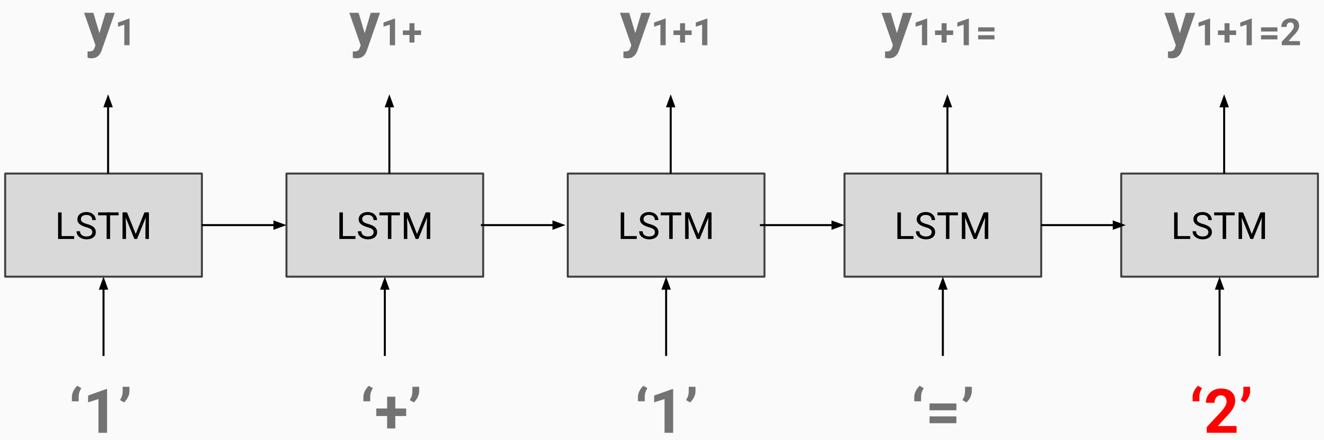
RNN Review



- Each input word embeddings has a corresponding output y.
- In generative tasks, RNN encodes the prior tokens to a vector representation and outputs y, which is the prediction of the next token.

Arithmetic

- You are tasked with training an LSTM recurrent model to enable it to perform arithmetic operations.



Dataset

Arithmetic dataset

- Train split: 2,369,250 pieces
- Eval split: 263,250 pieces
- Each data piece: A 2~3-number equation, each number is in [0, 50),
    - e.g. (10 + 4) * 2 =   and the answer is 28
    - The operations include: +, -, *, ()

Dataset examples

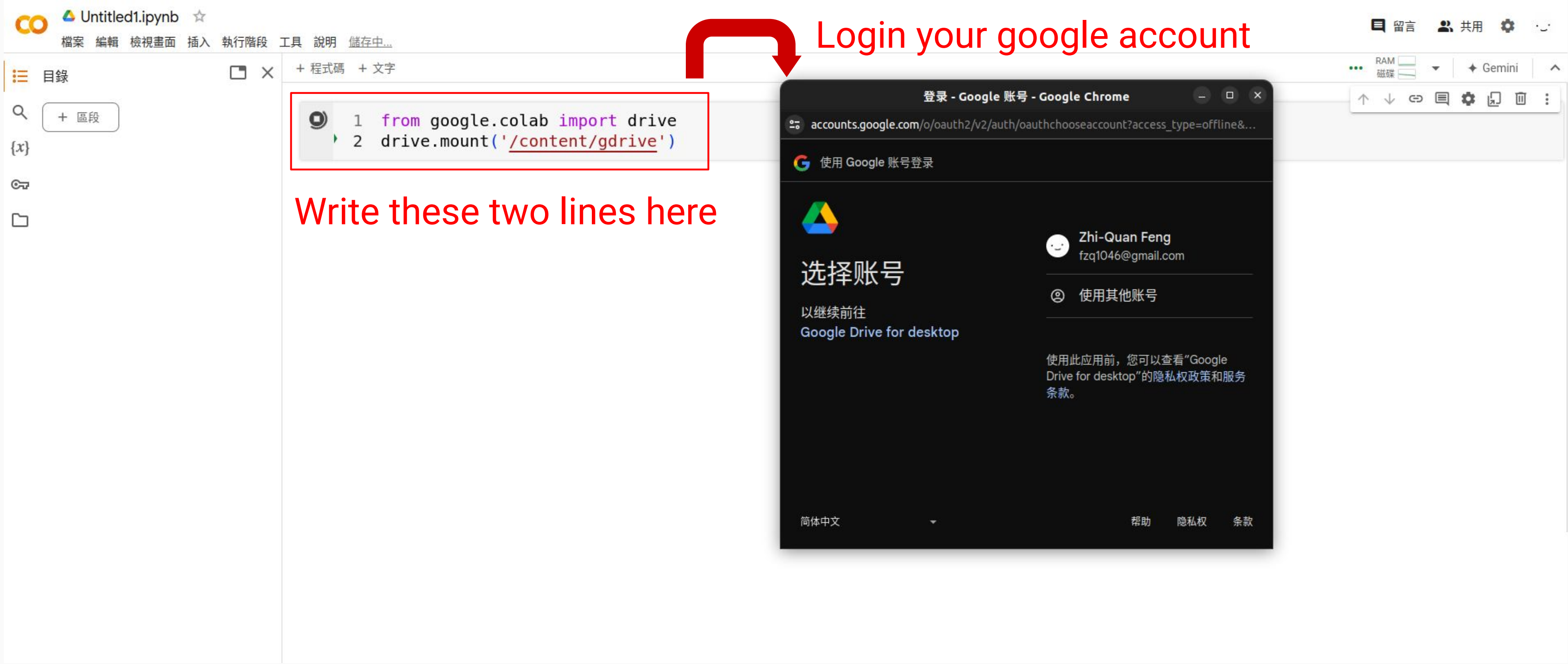
*Answer in red

- Task: **A (+/-/*) B (+/-/*) C =** ?

Example	Inputs	Answer
1 + 2 - 3 = 0	1 + 2 - 3 =	0
(10 + 4) * 2 = 28	(10 + 4) * 2 =	28

Code (hints only)

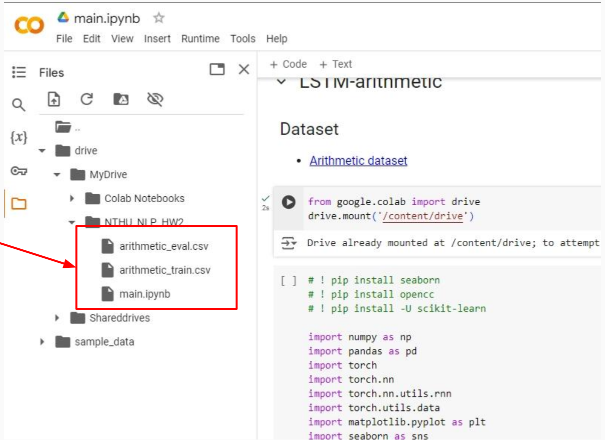
Colab: access google drive (1/2)



Colab: access google drive (2/2)

Upload your data
to google drive

Your google drive is here
(./drive/MyDrive/…)



Check the downloaded file

Arithmetic_train.csv

```
src,tgt
14*(43+20)=,882
(6+1)*5=,35
13+32+29=,74
31*(3-11)=,-248
24*49+1=,1177
3+(25*25)=,628
8*(30+10)=,320
9*38+49=,391
23-17=,6
23-26*15=,-367
18*22-19=,377
(47-23)*42=,1008
7-1+46=,52
(45+2)*25=,1175
47+(29-1)=,75
(27+41)-12=,56
38+(29-46)=,21
32*(19+28)=,1504
37*(35-24)=,407
24*(22-49)=,-648
(4-41)*6=,-222
24-39*6=,-210
38+(0-20)=,18
26-2-35=,-11
```

Line-by-line manner

id, input, ground truth separated by “ , ”

Load the data

Read the data from .csv file

```
df_train = pd.read_csv(os.path.join(data_path, 'arithmetic_train.csv'))
df_eval = pd.read_csv(os.path.join(data_path, 'arithmetic_eval.csv'))
df_train.head()
```

	src	tgt
0	0+0=	0
1	0-0=	0
2	0*0=	0
3	(0+0)*0=	0
4	0+0*0=	0

Transform the output data to string

```
# transform the input data to string
df_train['tgt'] = df_train['tgt'].apply(lambda x: str(x))
df_train['src'] = df_train['src'].add(df_train['tgt'])
df_train['len'] = df_train['src'].apply(lambda x: len(x))

df_eval['tgt'] = df_eval['tgt'].apply(lambda x: str(x))
df_eval['src'] = df_eval['src'].add(df_eval['tgt'])
df_eval['len'] = df_eval['src'].apply(lambda x: len(x))
```

TODO1: Build your dictionary here (5%)

Build Dictionary

- The model cannot perform calculations directly with plain text.
- Convert all text (numbers/symbols) into numerical representations.
- Special tokens
  - '<pad>'
    - Each sentence within a batch may have different lengths.
    - The length is padded with '<pad>' to match the longest sentence in the batch.
  - '<eos>'
    - Specifies the end of the generated sequence.
    - Without '<eos>', the model will not know when to stop generating.

```
[ ]
char_to_id = {}
id_to_char = {}

# write your code here
# Build a dictionary and give every token in the train dataset an id
# The dictionary should contain <eos> and <pad>
# char_to_id is to conver charactors to ids, while id_to_char is the opposite

vocab_size = len(char_to_id)
print('Vocab size{}'.format(vocab_size))
```

```
字典大小： 18
```

For example:

```
char_to_id = {
    '<pad>' : 0,
    '<eos>' : 1,
    '0' : 2,
    …
}
```

And,

```
id_to_char = {
    0 : '<pad>',
    1 : '<eos>',
    2 : '0',
    …
}
```

TODO2: Data preprocessing (10%)

Data Preprocessing

- The data is processed into the format required for the model's input and output. (End with <eos> token)

```
# Write your code here
df.head()
```

	src	tgt	len	char_id_list	label_id_list
0	0+0=0	0	5	[15, 3, 15, 17, 15, 1]	[0, 0, 0, 0, 15, 1]
1	0–0=0	0	5	[15, 7, 15, 17, 15, 1]	[0, 0, 0, 0, 15, 1]
2	0*0=0	0	5	[15, 13, 15, 17, 15, 1]	[0, 0, 0, 0, 15, 1]
3	(0+0)*0=0	0	9	[14, 15, 3, 15, 10, 13, 15, 17, 15, 1]	[0, 0, 0, 0, 0, 0, 0, 0, 15, 1]
4	0+0*0=0	0	7	[15, 3, 15, 13, 15, 17, 15, 1]	[0, 0, 0, 0, 0, 0, 15, 1]

Process the data into the format required for model's input and output.

The model is required to make predictions **only for the tokens following the '=' symbol in the input**.

Any output generated by the model before the '=' symbol is irrelevant and **should be excluded from the loss calculation during training**.

Here we replace them to '<pad>'

TODO3: Data Batching (5%)

Data Batching

- Use `torch.utils.data.Dataset` to create a data generation tool called `dataset`.
- The, use `torch.utils.data.DataLoader` to randomly sample from the `dataset` and group the samples into batches.
- Example: 1+2-3=0
  - Model input: 1 + 2 - 3 = 0
  - Model output: / / / / / 0 <eos> (the '/' can be replaced with <pad>)
  - The key for the model's output is that the model does not need to predict the next character of the previous part. What matters is that once the model sees '=', it should start generating the answer, which is '0'. After generating the answer, it should also generate<eos>

```python
class Dataset(torch.utils.data.Dataset):
    def __init__(self, sequences):
        self.sequences = sequences

    def __len__(self):
        # return the amount of data
        return # Write your code here

    def __getitem__(self, index):
        # Extract the input data x and the ground truth y from the data
        x = # Write your code here
        y = # Write your code here
        return x, y
```

In the DataLoader, data is initially extracted from the Dataset using the `__getitem__(...)` method to construct a batch. This batch is then passed to the collate function for further processing.

During model training, the DataLoader supplies the processed batch, which is the output of the collate function, as input to the model.

Model

```python
class CharRNN(torch.nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super(CharRNN, self).__init__()

        self.embedding = torch.nn.Embedding(num_embeddings=vocab_size,
                                            embedding_dim=embed_dim,
                                            padding_idx=char_to_id['<pad>'])

        self.rnn_layer1 = torch.nn.LSTM(input_size=embed_dim,
                                        hidden_size=hidden_dim,
                                        batch_first=True)

        self.rnn_layer2 = torch.nn.LSTM(input_size=hidden_dim,
                                        hidden_size=hidden_dim,
                                        batch_first=True)

        self.linear = torch.nn.Sequential(torch.nn.Linear(in_features=hidden_dim,
                                                          out_features=hidden_dim),
                                          torch.nn.ReLU(),
                                          torch.nn.Linear(in_features=hidden_dim,
                                                          out_features=vocab_size))
```

We define two LSTM layers (one is also ok).

TODO4: Generation (10%)

```
def generator(self, start_char, max_len=200):

    char_list = [char_to_id[c] for c in start_char]

    next_char = None

    while len(char_list) < max_len:
        # Write your code here
        # Pack the char_list to tensor
        # Input the tensor to the embedding layer, LSTM layers, linear respectively
        y = # Obtain the next token prediction y

        next_char = # Use argmax function to get the next token prediction

        if next_char == char_to_id['<eos>']:
            break

        char_list.append(next_char)

    return [id_to_char[ch_id] for ch_id in char_list]
```

The `start_char` is fed into the model. Each time a sequence is input into the model, it generates a prediction for the next token. The prediction for the next token **corresponds to the last element in the model's output sequence**.

We the output is ‘<eos>’, the generation should be stopped.

Teacher forcing

Teacher forcing is a training technique commonly used in sequence-based models.

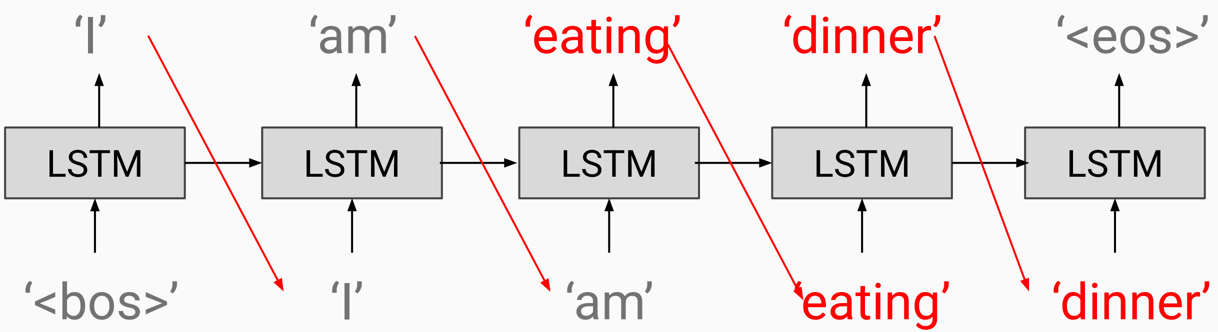
In teacher forcing, during training, instead of using the model's predicted output as input for the next time step, the true target (the ground truth) from the training data is fed as the next input.

e.g. 1+2-3=0
the model input is: "1+2-3="
instead of running several forward pass to generate the whole sequence,
we input: "1+2+3=0" and predict p('0'|'1+2+3=') and p('<eos>'|'1+2+3=0')

Teacher forcing

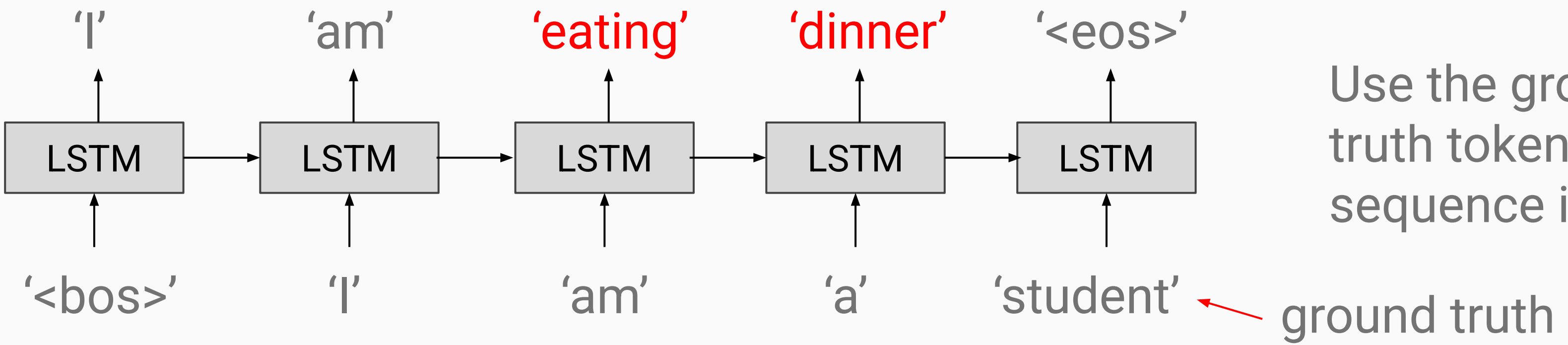
Without teacher forcing

For example, the ground truth is: “**I am a student**”,
but the model’s output is: “**I am eating dinner**”



Use the previous prediction as the next input token.

Teacher forcing



Use the ground truth tokens as sequence input

TODO5: Training (10%) and TODO6: Evaluation (10%)

- You are required to train the LSTM model using teacher forcing.
  - Make sure that you are training your model on gpu.

- You are required to compute accuracy (Exact Match) of the evaluation set.
  - You must generate the whole answers and check whether they match the ground truths.

Submission

Scoring

Coding work : 45%

TODOs	Scores
TODO1: Build your dictionary here	5%
TODO2: Data preprocessing	5%
TODO3: Data Batching	5%
TODO4: Generation	10%
TODO5: Training	10%
TODO6: Evaluation	10%

Scoring

Coding work : 45%

Score: 10% (The higher the accuracy achieved, the higher the score awarded.)

- A screenshot must be included at the end of the report.

Report: 45%

- Present your hyper-parameters in training, including learning rate, batch size, hidden size, epochs(steps), etc. (5%)
- If you use RNN or GRU instead of LSTM, what will happen to the quality of your answer generation? Why? (10%)
- If we construct an training set using three-digit numbers while the evaluation set is constructed from two-digit numbers, what will happen to the quality of your answer generation? (10%)
- If we construct a training set that includes 20% incorrect answers, how will this affect the quality of the generated responses? Present some examples. (5%)
- Why do we need gradient clipping during training? (5%)
- **… Anything that can strengthen your report.** (5%)

For ease of grading, you are encouraged to present data in textual form rather than as images.

Delivery policies: File formats

- Coding work: Python file (.py)
  - Download your script via Colab.
- Package list: requirements.txt
  - E.g., numpy==1.26.3
- Report: Microsoft Word (.docx)
- No other formats are allowed.
- Zip the files above before uploading you assignment.

Delivery policies: Filenames

	Filename rule	Filename example
Coding work	NLP_HW2_school_student_ID.py	NLP_HW2_NTHU_12345678.py
Report	NLP_HW2_school_student_ID.docx	NLP_HW2_NTHU_12345678.docx
Package list	requirements.txt	
Zipped file	NLP_HW2_school_student_ID.zip	NLP_HW2_NTHU_12345678.zip

Delivery policies: Things You should include

- In your report:

	Example	
Environment types	**If Colab or Kaggle**	**If local**
Running environment	Colab	System: Ubuntu 22.04, CPU: Ryzen 7-7800X3D
Python version	Colab	Python 3.10.1

Delivery policies: Rules of coding

- If you use ChatGPT or Generative AI, please specify your usage **both** in:
  - **Code comments**
  - **Reports**
- **No plagiarism**. You should not copy and paste from your classmates. **Submit duplicate code or report will get 0 point !**
- Please provide links if you take the code from the Internet as reference.
- The following behaviors will cause loss in the score of the assignment: **(1) Usage with Generative AI without specifications (2) Internet sources without specifications (3) Plagiarism.**

Punishments

Rule	Name your code: NLP_HW2_school_student_ID.py (only .py is acceptable)	Name your report: NLP_HW2_school_student_ID.docx	Name your file: NLP_HW2_school_student_ID.zip	Include requirements.txt
Punishment	-5	-5	-5	-5
Rule	Include python version in your report	Do not modify the code template (only changes to data loading are allowed).	Do not modify the report template	Your code or report should not shows a high degree of similarity to another student's submission.
Punishment	-5	-5	-5	-100 for both

If you are using Colab, go to **File** → **Download** → **Download .py** to obtain the Python file.

Uploading the zipped file

- Please upload your file to NTU COOL.
- You will have three weeks to finish this assignment.
- If you have any question, please e-mail to **nthuikmlab@gmail.com**