

# 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

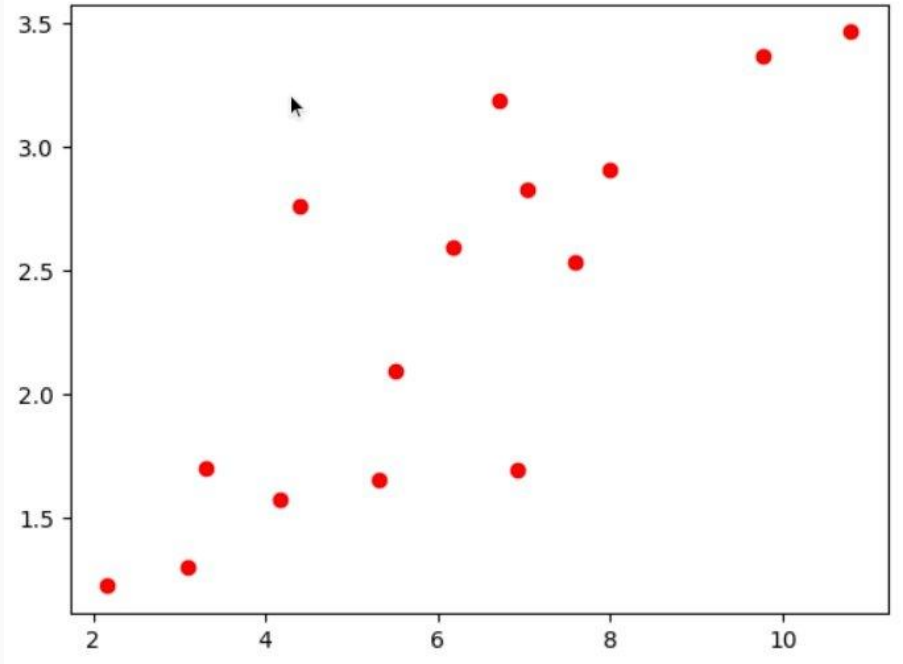
Step1: Prepare the dataset  
Step2: Construct the model  
Step3: Define Optimizer  
Step4: Define loss function  
Step5: Train the model  
Step6: Evaluate the model



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

```
# 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)

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

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

# 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()))
```

initialize  
data tensor

model

loss & optimizer

load data

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

## 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])

## 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

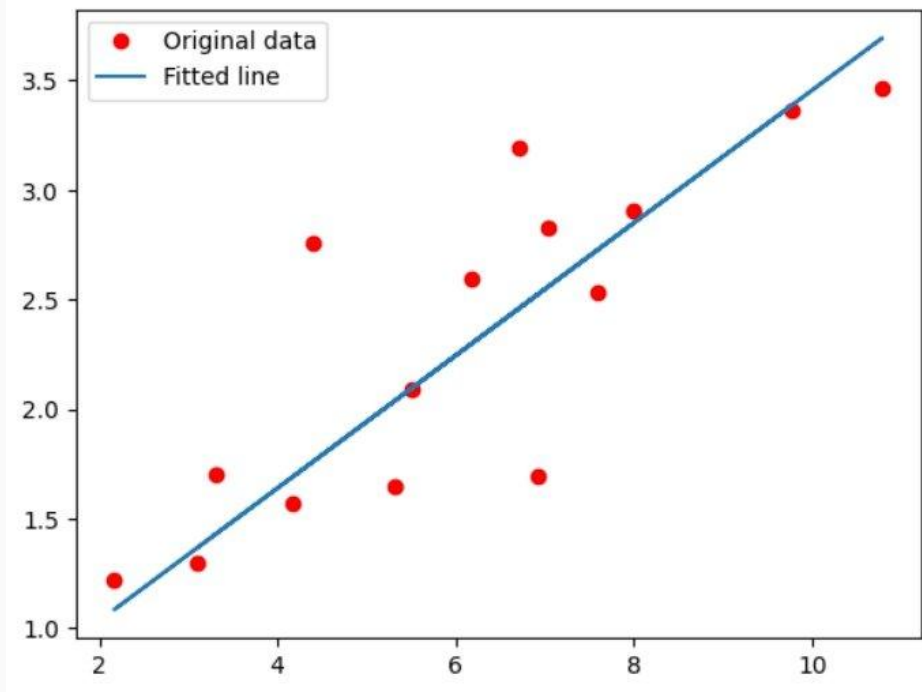
```
optimizer.zero_grad()  
print_grads(model)
```

weight: Parameter containing:  
tensor([[0.4065]], requires\_grad=True)  
weight grad: None

bias: Parameter containing:  
tensor([0.4805], requires\_grad=True)  
bias grad: None

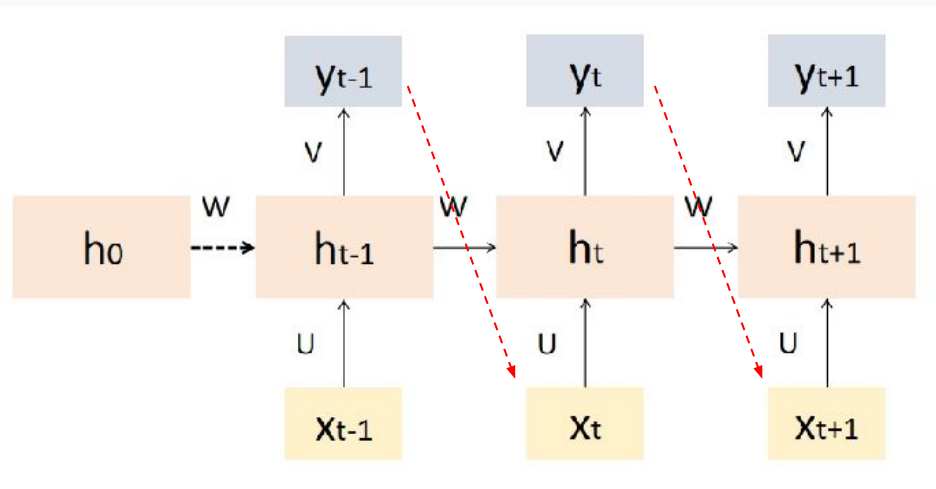
# Outputs

```
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
```





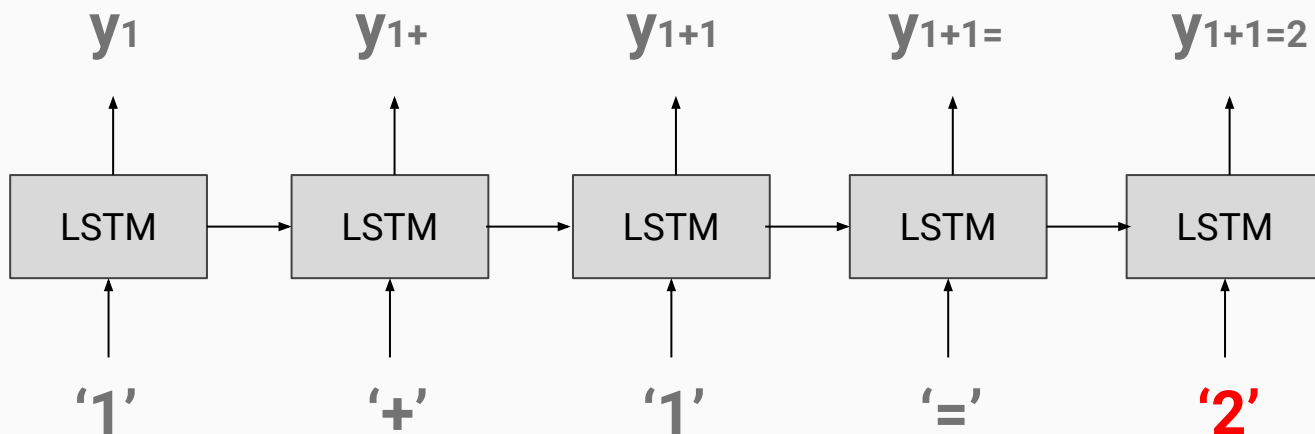
# RNN Review



- 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)

The image shows a Google Colab notebook titled "Untitled1.ipynb". In the code editor, two lines of Python code are highlighted with a red box:

```
1 from google.colab import drive
2 drive.mount('/content/gdrive')
```

Below the code box, a red arrow points to a browser window showing the Google account login page. The browser window title is "登录 - Google 账号 - Google Chrome". The URL is [accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?access\\_type=offline&...](https://accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?access_type=offline&...). The page displays the Google logo, the text "使用 Google 账号登录", and a "选择账号" (Select account) section. It shows a profile for "Zhi-Quan Feng" with email "fzq1046@gmail.com" and a button to "使用其他账号" (Use another account). Below this, it says "以继续前往 Google Drive for desktop" (Continue to Google Drive for desktop). At the bottom, there are links for "帮助" (Help), "隐私权" (Privacy), and "条款" (Terms).

Write these two lines here

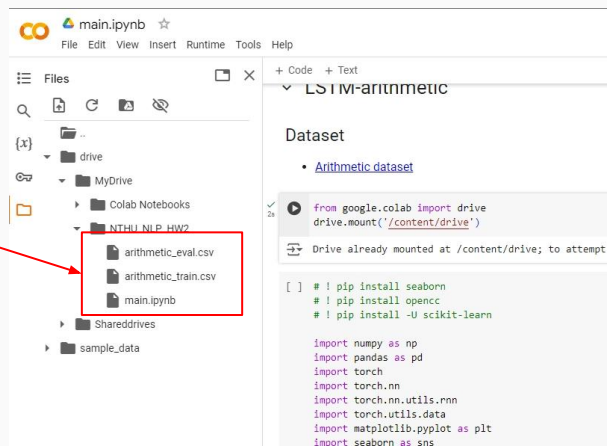
Login your google account

# Colab: access google drive (2/2)

Upload your data to google drive



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



## Arithmetic\_train.csv

Line-by-line manner

```
,src,tgt
2573208,48+43+34=,125
1630340,30-(48+13)=,-31
549277,(21*31)+10=,661
133957,2-27-10=,-35
1279828,(15*20)+24=,324
563918,10*35*26=,9100
153338,2-(45+30)=,-73
2307203,43*41*3=,5289
277302,5-13*17=,-216
59092,(1+6)-5=,2
1560955,19*29+32=,583
1803928,(13*6)+34=,112
617673,11-36*29=,-1033
1014178,(13*6)+19=,97
171066,3+12-22=,-7
2077727,7*(39-23)=,112
2320223,44-(3+21)=,20
1234928,(22*38)-23=,813
566661,10-38+6=,-22
```

id, input, ground truth  
separated by " , "



# Load the data

Read the data from .csv file

```
[ ] 1 df_train = pd.read_csv(os.path.join(data_path, 'arithmetic_train.csv'))
     2 df_eval = pd.read_csv(os.path.join(data_path, 'arithmetic_eval.csv'))
     3 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

```
[ ] 1 # transform the input data to string
     2 df_train['tgt'] = df_train['tgt'].apply(lambda x: str(x))
     3 df_train['src'] = df_train['src'].add(df_train['tgt'])
     4 df_train['len'] = df_train['src'].apply(lambda x: len(x))
     5
     6 df_eval['tgt'] = df_eval['tgt'].apply(lambda x: str(x))
     7 df_eval['src'] = df_eval['src'].add(df_eval['tgt'])
     8 df_eval['len'] = df_eval['src'].apply(lambda x: len(x))
```

# TODO1: Build your dictionary here

## ✓ 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.

```
[ ] 1 char_to_id = {}
    2 id_to_char = {}
    3
    4 # write your code here
    5 # Build a dictionary and give every token in the train dataset an id
    6 # The dictionary should contain <eos> and <pad>
    7 # char_to_id is to convert characters to ids, while id_to_char is the opposite
    8
    9 vocab_size = len(char_to_id)
   10 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

## ▼ Data Preprocessing

- The data is processed into the format required for the model's input and output.
- 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>

```
[ ] 1 # Write your code here
    2 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, 15, 1]
4	0+0*0=0	0	7	[15, 3, 15, 13, 15, 17, 15, 1]	[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

## ✓ Data Batching

- Use `torch.utils.data.Dataset` to create a data generation tool called `dataset`.
- Then, use `torch.utils.data.DataLoader` to randomly sample from the `dataset` and group the samples into batches.


```
1 class Dataset(torch.utils.data.Dataset):
2     def __init__(self, sequences):
3         self.sequences = sequences
4
5     def __len__(self):
6         # return the how much data is here in the Dataset object
7         return # Write your code here
8
9     def __getitem__(self, index):
10        # Extract the input data x and the ground truth y from the data
11        x = # Write your code here
12        y = # Write your code here
13        return x, y
14
```

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.

```
1 class CharRNN(torch.nn.Module):
2     def __init__(self, vocab_size, embed_dim, hidden_dim):
3         super(CharRNN, self).__init__()
4
5         self.embedding = torch.nn.Embedding(num_embeddings=vocab_size,
6                                             embedding_dim=embed_dim,
7                                             padding_idx=char_to_id['<pad>'])
8
9         self.rnn_layer1 = torch.nn.LSTM(input_size=embed_dim,
10                                         hidden_size=hidden_dim,
11                                         batch_first=True)
12
13         self.rnn_layer2 = torch.nn.LSTM(input_size=hidden_dim,
14                                         hidden_size=hidden_dim,
15                                         batch_first=True)
16
17         self.linear = torch.nn.Sequential(torch.nn.Linear(in_features=hidden_dim,
18                                                         out_features=hidden_dim),
19                                           torch.nn.ReLU(),
20                                           torch.nn.Linear(in_features=hidden_dim,
21                                                         out_features=vocab_size))
22
```

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



## TODO4: Generation

```
42 def generator(self, start_char, max_len=200):
43
44     char_list = [char_to_id[c] for c in start_char]
45
46     next_char = None
47
48     while len(char_list) < max_len:
49         # Write your code here
50         # Pack the char_list to tensor
51         # Input the tensor to the embedding layer, LSTM layers, linear respectively
52         y = # Obtain the next token prediction y
53
54         next_char = # Use argmax function to get the next token prediction
55
56         if next_char == char_to_id['<eos>']:
57             break
58
59         char_list.append(next_char)
60
61     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 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.**

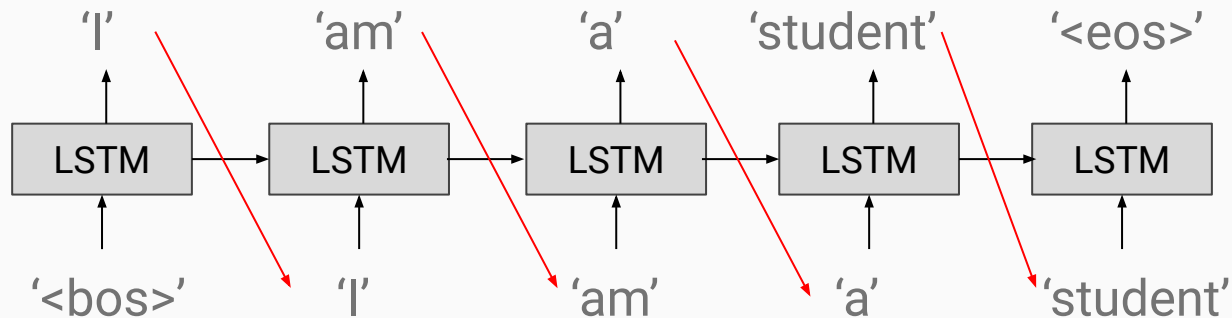
e.g.  $1+2-3=0$

the model input is: "1+2-3="

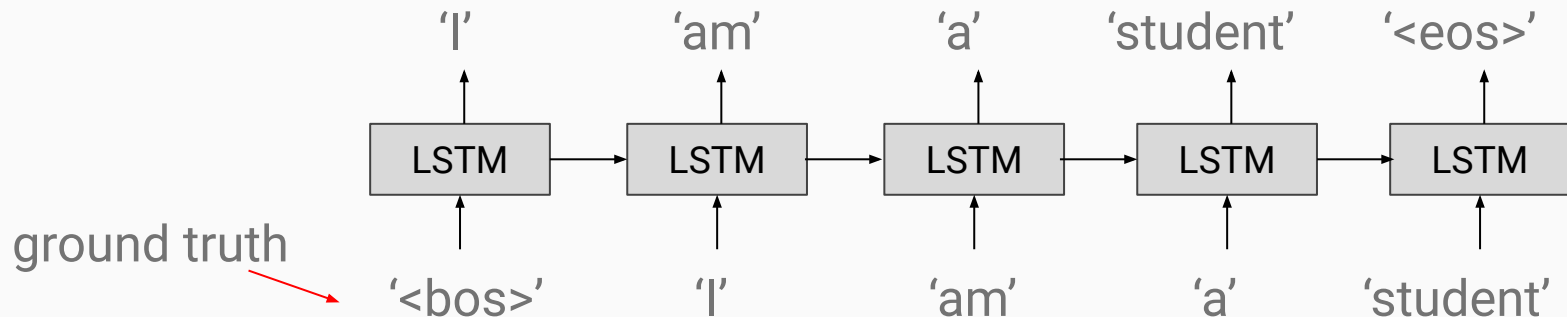
instead of running several forward pass to generate the whole sequence, we input: "1+2+3=0" and predict  $p('0'|'1+2+3=')$  and  $p('<eos>|'1+2+3=0')$

# Teacher forcing

## Without teacher forcing



## Teacher forcing





- 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

Coding work : **60%** (10% for each of the six TODOs)

Report: **40%**

- When training, which hyperparameter affects the learning rate?
- If you use RNN or GRU instead of LSTM, what will happen to the quality of your answer generation? Why?
- If we construct an evaluation set using three-digit numbers while the training set is constructed from two-digit numbers, what will happen to the quality of your answer generation?
- If some numbers never appear in your training data, what will happen to your answer generation?
- Why do we need gradient clipping during training?
- ... **Anything that can strengthen your report.**

# 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 your assignment.



# Delivery policies: Filenames

	Filename rule	Filename example
Coding work	NLP_HW2_ <b>school</b> _student_ID.py	NLP_HW2_ <b>NTHU</b> _12345678.py
Report	NLP_HW2_ <b>school</b> _student_ID.docx	NLP_HW2_ <b>NTHU</b> _12345678.docx
Package list	requirements.txt	
Zipped file	NLP_HW2_ <b>school</b> _student_ID.zip	NLP_HW2_ <b>NTHU</b> _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.**

# 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](mailto:nthuikmlab@gmail.com)**



# Problem Definition

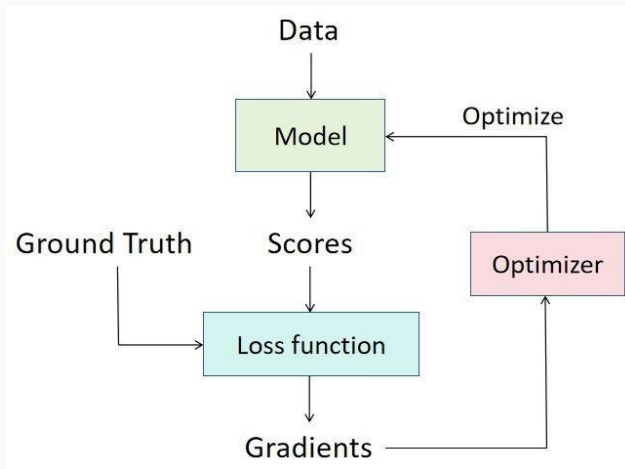
In a deep learning project, we need to:

1. Define a model
  - Input a batch of data, and compute the results.
2. Train the model
  - Record the derivation procedures.
  - Back propagate & Compute the gradients.
  - Optimize the parameters.
  - GPU acceleration.



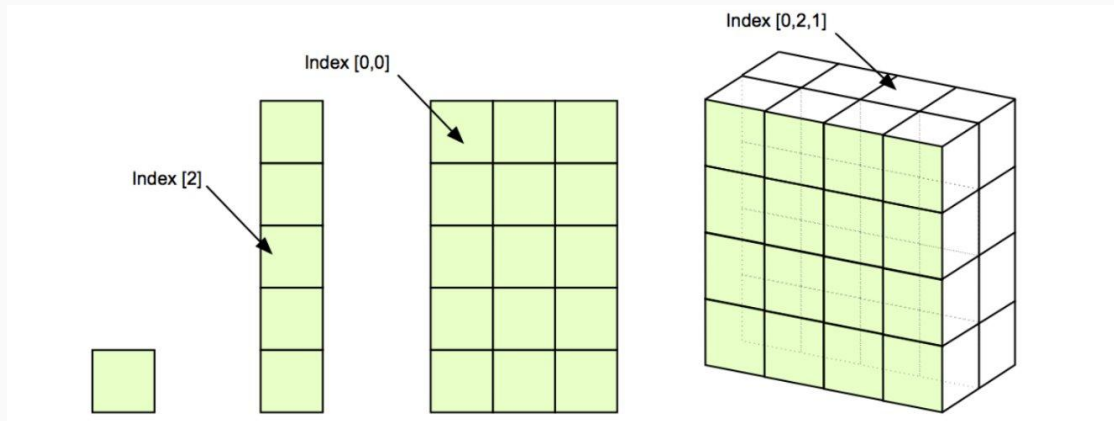
A special data structure was invented.

Called "**Tensor**"(張量)



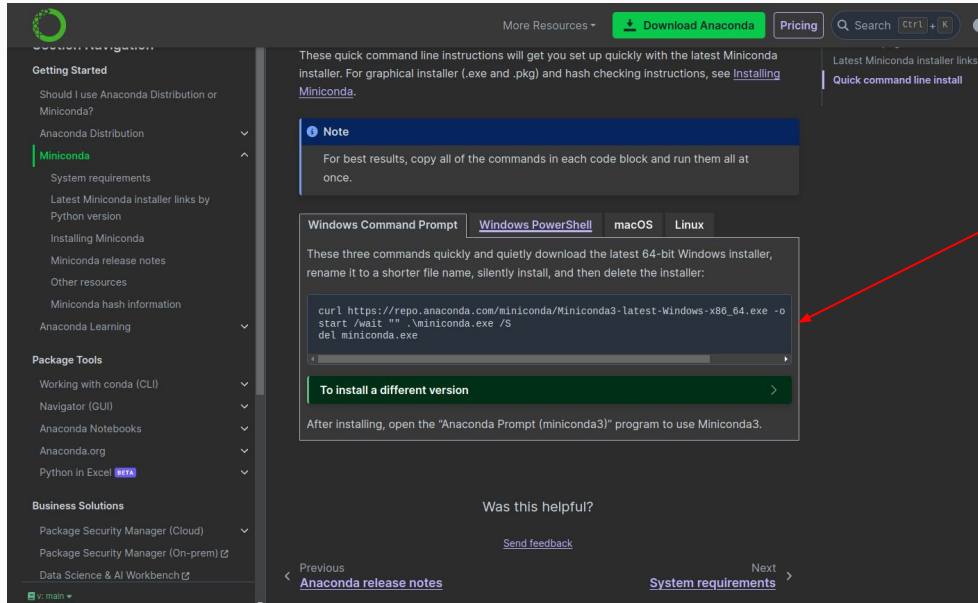
# Tensors (Pytorch) [Link](#)

- Tensors are similar to arrays and matrices.
- Tensors can run on GPUs or other hardware accelerators.



# Coding environment

[Miniconda Link](#)



Open shell and use the command line to install miniconda

If you run the code on your own computer, you can build environment by yourself.