Lab 09 - Character-level Language Model with LSTM

In this lab, your task is to build a character-level language model with LSTM layer.

Reference: [Let's build GPT: from scratch, in code, spelled out (by Andrej Karpathy) (https://www.youtube.com/watch?v=kCc8FmEb1nY)](https://www.youtube.com/watch?v=kCc8FmEb1nY)

In [24]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
ue).
```

In [25]:
```
cd "/content/drive/MyDrive/UCCD3074_Labs/UCCD3074_Lab9"
```

```
/content/drive/MyDrive/UCCD3074_Labs/UCCD3074_Lab9
```

In [26]:
```
import os, time
import torch
import torch.nn as nn
from torch.nn import functional as F
```

In [27]:
```
torch.manual_seed(1234)
device = 'cuda' if torch.cuda.is_available() else "cpu"
```

In [28]:
```
if not os.path.exists('input.txt'):
    !wget 'https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt'
```

Load the dataset

Read the dataset into the string `raw_data` .

In [29]:

```
with open('./input.txt', 'r', encoding='utf-8') as f:
    raw_data = f.read()

# print the length of the datasets
print('Number of characters:', len(raw_data))
print('\n------------------------')

# look at the first 1000 characters
print(raw_data[:1000])
```

Number of characters: 1115394

First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor citizens, the patricians good.
What authority surfeits on would relieve us: if they
would yield us but the superfluity, while it were
wholesome, we might guess they relieved us humanely;
but they think we are too dear: the leanness that
afflicts us, the object of our misery, is as an
inventory to particularise their abundance; our
sufferance is a gain to them Let us revenge this with
our pikes, ere we become rakes: for the gods know I
speak this in hunger for bread, not in thirst for revenge.

Create the vocabulary

Get the the vocabulary `vocab` from the raw data. `vocab` contains all unique characters in the raw data.

In [30]:

```python
vocab = sorted(list(set(raw_data)))
vocab_size = len(vocab)
print('vocab:', vocab)
print('vocab_size:', vocab_size)
```

```
vocab: ['\n', ' ', '!', '$', '&', "'", ',', '-', '.', '3', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
vocab_size: 65
```

Create the vocabulary mapping functions. `stoi` performs the mapping from the character token to its index, and `itos` , vice versa.

In [31]:

```python
stoi = {token : id for id, token in enumerate(vocab)}
itos = {id : token for id, token in enumerate(vocab)}
```

In [32]:

```python
print(stoi)
print(itos)
```

```
{'\n': 0, ' ': 1, '!': 2, '$': 3, '&': 4, "'": 5, ',': 6, '-': 7, '.': 8, '3': 9, ':': 10, ';': 11, '?': 12, 'A': 13, 'B': 1
4, 'C': 15, 'D': 16, 'E': 17, 'F': 18, 'G': 19, 'H': 20, 'I': 21, 'J': 22, 'K': 23, 'L': 24, 'M': 25, 'N': 26, 'O': 27, 'P':
28, 'Q': 29, 'R': 30, 'S': 31, 'T': 32, 'U': 33, 'V': 34, 'W': 35, 'X': 36, 'Y': 37, 'Z': 38, 'a': 39, 'b': 40, 'c': 41,
'd': 42, 'e': 43, 'f': 44, 'g': 45, 'h': 46, 'i': 47, 'j': 48, 'k': 49, 'l': 50, 'm': 51, 'n': 52, 'o': 53, 'p': 54, 'q': 5
5, 'r': 56, 's': 57, 't': 58, 'u': 59, 'v': 60, 'w': 61, 'x': 62, 'y': 63, 'z': 64}
{0: '\n', 1: ' ', 2: '!', 3: '$', 4: '&', 5: "'", 6: ',', 7: '-', 8: '.', 9: '3', 10: ':', 11: ';', 12: '?', 13: 'A', 14:
'B', 15: 'C', 16: 'D', 17: 'E', 18: 'F', 19: 'G', 20: 'H', 21: 'I', 22: 'J', 23: 'K', 24: 'L', 25: 'M', 26: 'N', 27: 'O', 2
8: 'P', 29: 'Q', 30: 'R', 31: 'S', 32: 'T', 33: 'U', 34: 'V', 35: 'W', 36: 'X', 37: 'Y', 38: 'Z', 39: 'a', 40: 'b', 41: 'c',
42: 'd', 43: 'e', 44: 'f', 45: 'g', 46: 'h', 47: 'i', 48: 'j', 49: 'k', 50: 'l', 51: 'm', 52: 'n', 53: 'o', 54: 'p', 55:
'q', 56: 'r', 57: 's', 58: 't', 59: 'u', 60: 'v', 61: 'w', 62: 'x', 63: 'y', 64: 'z'}
```

Create the function to encode and decode the text. `encode_text` encodes a string into its one-hot integer representation while `decode_text` decodes an integer representation back to its text.

In [33]:

```python
encode_text = lambda str : [stoi[s] for s in str]
encoded = encode_text("Hello how do you do?")
print(encoded)
```

```
[20, 43, 50, 50, 53, 1, 46, 53, 61, 1, 42, 53, 1, 63, 53, 59, 1, 42, 53, 12]
```

In [34]:

```python
decode_text = lambda l : ''.join([itos[i] for i in l])
decoded = decode_text(encoded)
print(decoded)
```

```
Hello how do you do?
```

Now, we encode the raw data and then convert it into a 1-D integer tensor `data` .

In [35]:

```python
data = torch.tensor(encode_text(raw_data))
print('Shape of data:', data.shape)
print('Type of data: ', data.dtype)
print('\nFirst 100 characters of data:\n', data[:100])
```

```
Shape of data: torch.Size([1115394])
Type of data:  torch.int64

First 100 characters of data:
 tensor([18, 47, 56, 57, 58,  1, 15, 47, 58, 47, 64, 43, 52, 10,  0, 14, 43, 44,
        53, 56, 43,  1, 61, 43,  1, 54, 56, 53, 41, 43, 43, 42,  1, 39, 52, 63,
         1, 44, 59, 56, 58, 46, 43, 56,  6,  1, 46, 43, 39, 56,  1, 51, 43,  1,
        57, 54, 43, 39, 49,  8,  0,  0, 13, 50, 50, 10,  0, 31, 54, 43, 39, 49,
         6,  1, 57, 54, 43, 39, 49,  8,  0,  0, 18, 47, 56, 57, 58,  1, 15, 47,
        58, 47, 64, 43, 52, 10,  0, 37, 53, 59])
```

To train the language model, the samples will be trained with a block of text with `block_size` characters. The function `get_batch` randomly sample a block of text as input `x` . The label `y` is the block of text shifted by 1 position of `x` . The start position of the first block is 0. The start position of the last block is `len(data) - (block_size + 1)` where each the length of each sample is `block_size+1` (remember that `y` is `x` shifted by 1).

In [36]:
```python
torch.manual_seed(1234)

def get_batch(batch_size, block_size, device):
    ix   = torch.randint(0, len(data) - block_size - 1, (batch_size,))
    x = torch.stack([data[i:i+block_size] for i in ix])
    y = torch.stack([data[i+1:i+block_size+1] for i in ix])
    x, y = x.to(device), y.to(device)
    return x, y
```

In [37]:
```python
batch_size=4
block_size=8
```

In [38]:
```python
device = "cuda" if torch.cuda.is_available() else "cpu"
x_batch, y_batch = get_batch(batch_size, block_size, device)

print(x_batch)
print('----------------')
print(y_batch)
```

```
tensor([[21, 17, 32, 10,  0, 27,  1, 58],
        [ 6,  1, 44, 53, 53, 50, 47, 57],
        [43, 56,  2,  1, 39, 58,  1, 39],
        [53, 59, 56,  1, 43, 63, 43, 57]], device='cuda:0')
----------------
tensor([[17, 32, 10,  0, 27,  1, 58, 46],
        [ 1, 44, 53, 53, 50, 47, 57, 46],
        [56,  2,  1, 39, 58,  1, 39,  1],
        [59, 56,  1, 43, 63, 43, 57,  1]], device='cuda:0')
```

Note that when training on `(x, y)`, the model is learning multiple conditional probabilities `p(target|context)` simultaneously.

In [39]:
```python
x, y = x_batch[0], y_batch[0]

print('x:', x.tolist())
print('y:', y.tolist(), '\n')

for t in range(block_size):
    context = x[:t+1]
    target = y[t]
    print(f'when input is: {context}, the target is: {target}')
```

```
x: [21, 17, 32, 10, 0, 27, 1, 58]
y: [17, 32, 10, 0, 27, 1, 58, 46]

when input is: tensor([21], device='cuda:0'), the target is: 17
when input is: tensor([21, 17], device='cuda:0'), the target is: 32
when input is: tensor([21, 17, 32], device='cuda:0'), the target is: 10
when input is: tensor([21, 17, 32, 10], device='cuda:0'), the target is: 0
when input is: tensor([21, 17, 32, 10,  0], device='cuda:0'), the target is: 27
when input is: tensor([21, 17, 32, 10,  0, 27], device='cuda:0'), the target is: 1
when input is: tensor([21, 17, 32, 10,  0, 27,  1], device='cuda:0'), the target is: 58
when input is: tensor([21, 17, 32, 10,  0, 27,  1, 58], device='cuda:0'), the target is: 46
```

Create the character-level language model with LSTM

Network structure

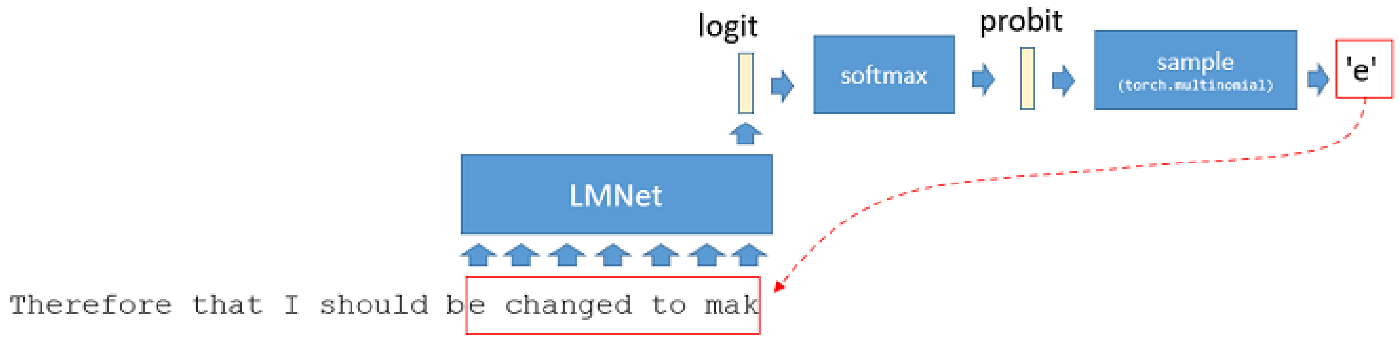
Now, let's build the character-level language model with LSTM. The network `LMNet` has the following layers:

Layer	Configuration	Shape
Input	-	`(B, T)`
Embedding	num_embedding = vocab_size, embedding_dim = n_embd	`(B, T, n_embd)`
LSTM	input_size = n_embd, hidden_size = n_embd num_layers = 1, batch_first = true	`(B, T, n_embd)`
fc	in_features = n_embd, out_features = vocab_size	`(B, T, vocab_size)`

Generating novel text

To generate novel text, we implement the method `generate`. Since the network is trained on a sequence of length `T = block_size`, when generating the text, we feed the most recent `block_size` characters into the network to generate the next character. Here are the steps:

1. Crop the most recent `block_size` characters in the generated text
2. The cropped text is fed to the generative model to generate the next character. The network output the `logit` value.
3. Convert the logit of the network to `probit` value by performing `softmax` operation.
4. Sample a character from the `probit` by using [`torch.multinorm` (https://pytorch.org/docs/stable/generated/torch.multinomial.html)](https://pytorch.org/docs/stable/generated/torch.multinomial.html)
5. Append the sampled character to the end of the generated text.
6. Repeat steps 1-5 for `text_len` times



In [40]:

```
class LMNet(nn.Module):
    def __init__(self, vocab_size, n_embd):
        super().__init__()
        self.embedding = nn.Embedding(num_embeddings=vocab_size, embedding_dim=n_embd)
        self.lstm      = nn.LSTM(input_size=n_embd, hidden_size=n_embd, num_layers=1, batch_first=True, bidirectional=Fal
        self.fc        = nn.Linear(in_features=n_embd, out_features=vocab_size)

    def forward(self, x):

        x    = self.embedding(x)
        x, _ = self.lstm(x)
        x    = self.fc(x)
        return x

    def generate(self, text_len, block_size):

        model.eval()  # set to evaluation mode

        # initialize the text with the first token (newline)
        num_samples = 1
        num_tokens = 1
        text = torch.zeros((num_samples, num_tokens), dtype=torch.long).to(device)

        # repeat until the length of text = "text_len"
        for t in range(text_len):

            # crop text to the last block-size tokens
            inputs = text[:, -block_size:]

            # get the predictions
            with torch.no_grad():
                logit = self(inputs)     # Shape = (B=1, T, F)

                # focus only on the last time step.
                logit = logit[:, -1, :]  # Shape = (F,)

                # apply soft max to get probabilities
                probit = F.softmax(logit, dim=-1)

                # sample from distribution
                next_token = torch.multinomial(probit, num_samples=1)   # (T,) --> (B, T), i.e.,  (1,) --> (1, 1)

                # append sampled index to the running sequence
                text = torch.cat((text, next_token), dim=1)
```

```

            # print the sample
            print(itos[next_token.item()], end='')
            time.sleep(0.01)
```

Create the model for testing

In [41]:
```
model = LMNet(vocab_size=len(vocab), n_embd=32).to(device)
```

In [42]:
```
x, y = get_batch(batch_size=4, block_size=8, device=device)
x, y = x.to(device), y.to(device)

yhat = model(x)
```

Train the model

In [43]:
```
batch_size    = 128
block_size    = 256
lr            = 3e-4
max_iters     = 20000
show_interval = 1000
n_embd        = 256
```

Create the model

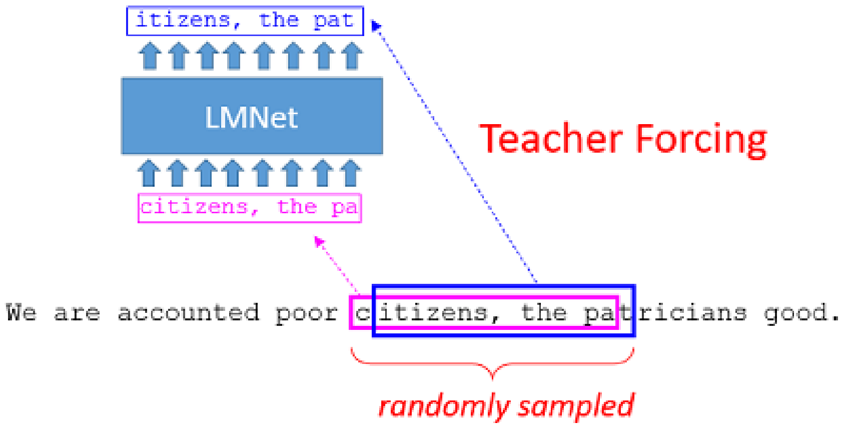
In [44]:
```
model = LMNet(vocab_size=len(vocab), n_embd=n_embd).to(device)
```

Create the optimizer

In [45]:
```
optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
```

Train the model

To train the model, we use **teacher forcing** where the predicted output is simply the 1-shifted sequence of the input sequence. We shall train the network with sentence sequence of length `block_size`. Since the network is trained on sequences of length `block_size`, during inference, the generative model should use input sequence of similar length to get good results.



During training, the network is based on the many-to-many architecture. However, during inference (Figure at generating novel text), the network is based on many-to-one architecture.

In [46]:

```
model.train()  # set to training mode

for steps in range(max_iters):

    # sample a batch of data
    x_batch, y_batch = get_batch(batch_size, block_size, device)

    # forward propagation
    yhat_batch = model(x_batch)

    # compute loss
    B, T, C = yhat_batch.shape
    yhat_batch = yhat_batch.reshape(-1, yhat_batch.size(-1))
    y_batch = y_batch.reshape(-1)
    loss = F.cross_entropy(yhat_batch, y_batch)

    # backpropagation
    loss.backward()
    optimizer.step()

    # reset the optimizer
    optimizer.zero_grad()

    # print the training loss
    if steps % show_interval == 0:
        print(f"Iter {steps}: train loss {loss:.4f}")
```

```
Iter 0: train loss 4.1727
Iter 1000: train loss 1.6298
Iter 2000: train loss 1.4246
Iter 3000: train loss 1.3700
Iter 4000: train loss 1.2928
Iter 5000: train loss 1.2847
Iter 6000: train loss 1.2369
Iter 7000: train loss 1.2231
Iter 8000: train loss 1.2068
Iter 9000: train loss 1.1888
Iter 10000: train loss 1.1635
Iter 11000: train loss 1.1638
Iter 12000: train loss 1.1454
Iter 13000: train loss 1.1550
Iter 14000: train loss 1.1263
Iter 15000: train loss 1.1121
Iter 16000: train loss 1.1081
Iter 17000: train loss 1.0878
Iter 18000: train loss 1.0928
Iter 19000: train loss 1.0768
```

Generate text

In [47]:
```
model.generate(text_len=1000, block_size=block_size)
```

```
Curse in no soily servant fastle heaven!

WARWICK:
Your firm strongs before your palace room use to leave
to strike above thee it, to see your mother's suit:
Speak now. There's many's ask the fully, and to her
beat with keeper. Katharina, sir. Your hand and
A man of ault the obsequio's famous first
By yours behomise: when Marcius doth touchety thou
Theirs' blood with winds, he could support him, not
Loes burthen English eye my father's rob:
Give us strip to loyal sister is my meanal.

QUEEN:
O God! dares, my man I was so trust?

ISABELLA:
Ay, and do guist, no
For her maid to many auptleed their state:
Ah, is the obscuin the ancient fellow.
To thispiteries him for the guilt is shame,
Nor need to wearing from Pisa, was again,
When with the angel guest come to her heart
Of noble man shall but Juliet scorps
That in the throw thy tear.

AUFIDIUS:
Go think;
Any that thou concertanately be foot!
To break with purpose sword freely to fear,
And all to straight, thou tread to undernorater,
I tru
```