Introduction to text generation

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Instructor

Text generation and NLP

- Key applications: chatbots, language translation, technical writing
- RNN, LSTM, GRU: remembering past information for better sequential data processing
- Input: The cat is on the m
- Output: The cat is on the m**at**

[1] Image by vectorjuice on Freepik

Building an RNN for text generation

```
import torch
import torch.nn as nn
data = "Hello how are you?"
chars = list(set(data))
char_to_ix = {char: i for i, char in enumerate(chars)}
ix_to_char = {i: char for i, char in enumerate(chars)}
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
```

Forward propagation and model creation

```
    def forward(self, x):
        h0 = torch.zeros(1, x.size(0), self.hidden_size)
        out, _ = self.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        return out

model = RNNmodel(1, 16, 1)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

Preparing input and target data

```
inputs = [char_to_ix[ch] for ch in data[:-1]]
targets = [char_to_ix[ch] for ch in data[1:]]


inputs = torch.tensor(inputs, dtype=torch.long)
              .view(-1, 1)


inputs = nn.functional.one_hot(
       inputs, num_classes=len(chars)).float()


targets = torch.tensor(targets, dtype=torch.long)
```

- Creating indexes
- Tensor conversion
- One-Hot encoding
- Targets preparation

Training the RNN model

```
for epoch in range(100):
    model.train()
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch {epoch+1}/100, Loss: {loss.item()}')
```

Testing the model

```
model.eval()
test_input = char_to_ix['h']
test_input = nn.functional.one_hot(torch.tensor(test_input)
                                   .view(-1, 1), num_classes=len(chars)).float()
predicted_output = model(test_input)
predicted_char_ix = torch.argmax(predicted_output, 1).item()
print(f'Test Input: 10, Predicted Output: {model(test_input).item()}')
```

```
Epoch 10/100, Loss: 3090.861572265625
Epoch 20/100, Loss: 2935.4580078125
...
Epoch 100/100, Loss: 1922.44140625
```

```
Test Input: h, Predicted Output: e
```

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH

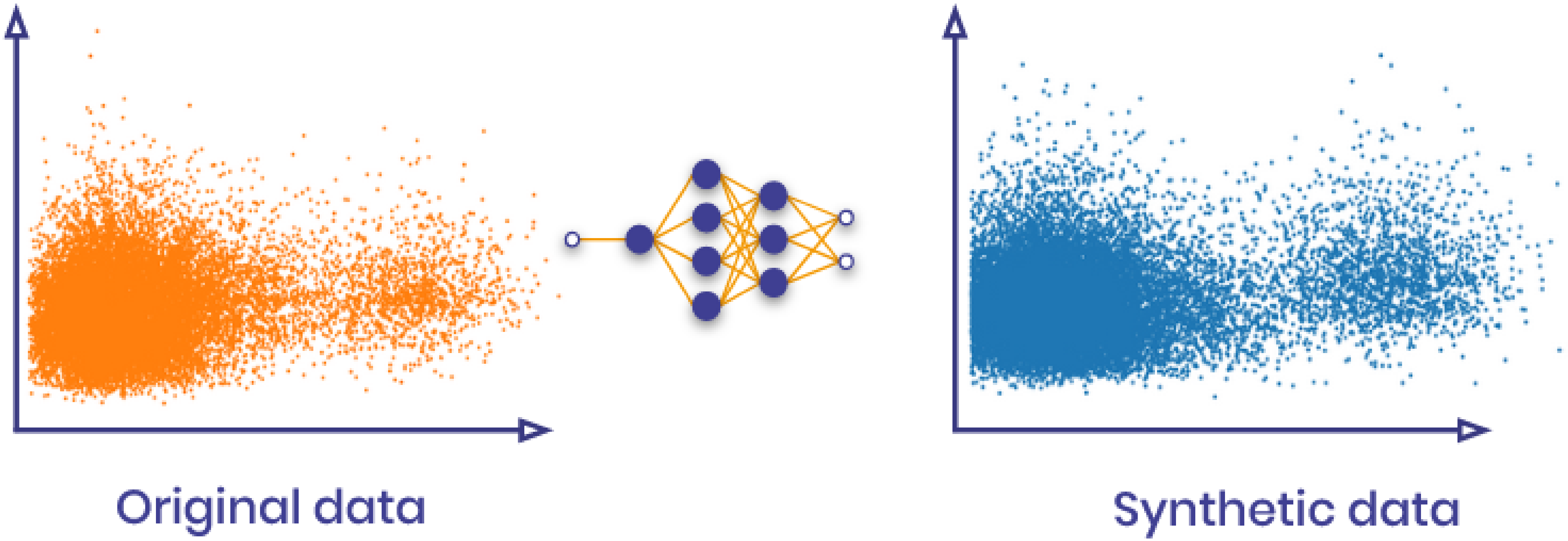
Generative adversarial networks for text generation

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Instructor

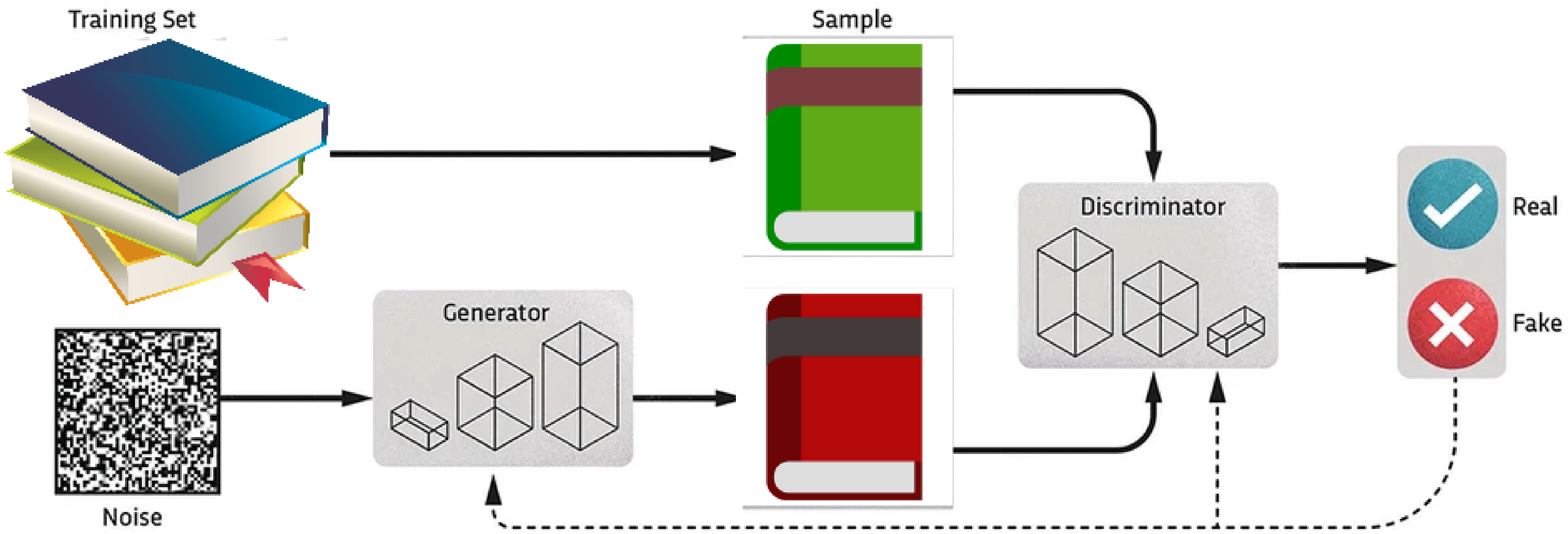
GANs and their role in text generation

- GANs can generate new content that seems original
  - Preserves statistical similarities
- Can replicate complex patterns unachievable by RNNs
- Can emulate real-world patterns



Structure of a GAN

- A GAN has two components:
  - Generator: creates fake samples by adding noise
  - Discriminator: differentiates between real and generated text data



[1] https://www.sciencefocus.com/future-technology/how-do-machine-learning-gans-work/

Building a GAN model in PyTorch: Generator

```
# Embedding reviews
# Convert reviews to tensors
class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(seq_length, seq_length),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)
```

Building the discriminator network

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(seq_length, 1),
            nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)
```

Initializing networks and loss function

```
generator = Generator()
discriminator = Discriminator()

criterion = nn.BCELoss()

optimizer_gen = torch.optim.Adam(generator.parameters(), lr=0.001)
optimizer_disc = torch.optim.Adam(discriminator.parameters(), lr=0.001)
```

Training the discriminator

```
num_epochs = 50
for epoch in range(num_epochs):
    for real_data in data:
        real_data = real_data.unsqueeze(0)
        noise = torch.rand((1, seq_length))
        disc_real = discriminator(real_data)
        fake_data = generator(noise)
        disc_fake = discriminator(fake_data.detach())
        loss_disc = criterion(disc_real, torch.ones_like(disc_real)) +
                    criterion(disc_fake, torch.zeros_like(disc_fake))
        optimizer_disc.zero_grad()
        loss_disc.backward()
        optimizer_disc.step()
```

Training the generator

```
    # ... (continued from last slide)
     disc_fake = discriminator(fake_data)
     loss_gen = criterion(disc_fake, torch.ones_like(disc_fake))
     optimizer_gen.zero_grad()
     loss_gen.backward()
     optimizer_gen.step()

 if (epoch+1) % 10 == 0:
     print(f"Epoch {epoch+1}/{num_epochs}:\t
             Generator loss: {loss_gen.item()}\t
             Discriminator loss: {loss_disc.item()}")
```

Printing real and generated data

```
print("\nReal data: ")
print(data[:5])

print("\nGenerated data: ")
for _ in range(5):
    noise = torch.rand((1, seq_length))
    generated_data = generator(noise)
    print(torch.round(generated_data).detach())
```

GANs: generated synthetic data

```
Epoch 10/50: Generator loss: 0.8992824673652 Discriminator loss: 1.37682652473
Epoch 20/50: Generator loss: 0.7347183227539 Discriminator loss: 1.39010250568З
...
Epoch 50/50: Generator loss: 0.7019854784011 Discriminator loss: 1.3501529693603
```

Generated data

```
Real data:
tensor([[1., 0., 0., 1., 1.],
        [0., 0., 1., 0., 0.],
        [1., 0., 1., 1., 1.],
        [1., 0., 1., 0., 0.],
        [1., 1., 1., 1., 1.]])
```

```
Generated data:
tensor([[0., 1., 1., 0., 0.]]),
tensor([[0., 1., 1., 1., 1.]])
tensor([[1, 1., 1., 0., 0.]]),
tensor([[1., 1., 1., 0., 0.]])
tensor([[0., 1., 1., 1., 1.]])
```

- Evaluation metric: correlation matrix

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH

Pre-trained models for text generation

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Instructor

Why pre-trained models?

Benefits

1. Trained on extensive datasets
2. High performance across various text generation tasks
   - Sentiments analysis
   - Text completion
   - Language translation

Limitations

1. High computational cost for training
2. Large storage requirements
3. Limited customization options

Pre-trained models in PyTorch

- Hugging Face Transformers: library of pre-trained models
- Pre-trained models:
  - GPT-2
  - T5

Understanding GPT-2 Tokenizer and Model

GPT2LMHeadModel:

- HuggingFace's take on GPT-2
- Tailored for text generation

GPT2Tokenizer:

- Converts text into tokens
- Handles subword tokenization: 'larger' might become ['large', 'r']

GPT-2: text generation implementation

```python
import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')
seed_text = "Once upon a time"
input_ids = tokenizer.encode(seed_text, return_tensors='pt')
```

GPT-2: text generation implementation II

```
output = model.generate(

                                                          )
```

GPT-2: text generation implementation II

```
output = model.generate(input_ids, max_length=40,


                                                        )
```

GPT-2: text generation implementation II

```
output = model.generate(input_ids, max_length=40, temperature=0.7,

                                                    )
```

GPT-2: text generation implementation II

```
output = model.generate(input_ids, max_length=40, temperature=0.7,
                        no_repeat_ngram_size=2,
                                                      )
```

GPT-2: text generation implementation II

```
output = model.generate(input_ids, max_length=40, temperature=0.7,
                        no_repeat_ngram_size=2,
                        pad_token_id=tokenizer.eos_token_id)
```

GPT-2: text generation output

```
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print(generated_text)
```

```
Generated Text: Once upon a time, the world was a place of great beauty
and great danger. The world of the gods was the place where the great gods were
born, and where they were to live.
```

T5: Language translation implementation

- `t5-small` : Text-to-Text Transfer Transformer
- Pretrained model for language translation tasks

```
import torch
from transformers import T5Tokenizer, T5ForConditionalGeneration
tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small")
input_prompt = "translate English to French: 'Hello, how are you?'"
input_ids = tokenizer.encode(input_prompt, return_tensors="pt")
output = model.generate(input_ids, max_length=100)
```

T5: Language translation output

```
generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
print("Generated text:",generated_text)
```

```
Generated text:
"Bonjour, comment êtes-vous?"
```

Choosing the right pre-trained model

- Many exist!

- **GPT-2**: Text generation
- **DistilGPT-2** (Smaller version of GPT-2): Text generation
- **BERT**: Text classification, question-answering
- **T5** (t5-small is the smaller version of T5): Language translation, summarization

- Find them in HuggingFace and other repositories

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH

Evaluation metrics for text generation

DEEP LEARNING FOR TEXT WITH PYTORCH

Shubham Jain
Instructor

Evaluating text generation

- Text Generation tasks create human-like text
- Standard accuracy metrics such as accuracy, F1 fall short for these tasks
- We need metrics that evaluate the quality of generated text
- BLEU and ROUGE

BLEU (Bilingual Evaluation Understudy)

- Compares the generated text and the reference text
- Checks for the occurrence of n-grams
- In the sentence "**The cat is on the mat**"
  - 1-grams (uni-gram): [the ,cat, is, on, the, mat]
  - 2-grams (bi-gram): ["the cat", "cat is", "is on", "on the", and "the mat"]
  - and so on for n-grams
- A perfect match: Score of 1.0
  - 0 means no match

Calculating BLEU score with PyTorch

```
from torchmetrics.text import BLEUScore

generated_text = ['the cat is on the mat']
real_text = [['there is a cat on the mat', 'a cat is on the mat']]

bleu = BLEUScore()
bleu_metric = bleu(generated_text, real_text)
print("BLEU Score: ", bleu_metric.item())
```

```
BLEU Score: tensor(0.7598)
```

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

- Compares a generated text to a reference text in two ways
- ROUGE-N: Considers overlapping n-grams (N=1 for unigrams, 2 for bigrams, etc.) in both texts
- ROUGE-L: Looks at the longest common subsequence (LCS) between the texts
- ROUGE Metrics:
  - F-measure: Harmonic mean of precision and recall
  - Precision: Matches of n-grams in generated text within the reference text
  - Recall: Matches of n-grams in reference text within the generated text
- 'rouge1', 'rouge2', and 'rougeL' prefixes refer to 1-gram, 2-gram, or LCS, respectively

Calculating ROUGE score with PyTorch

```
from torchmetrics.text import  ROUGEScore

generated_text='Hello, how are you doing?'
real_text= "Hello, how are you?"

rouge = ROUGEScore()

rouge_score = rouge([generated_text], [[real_text]])
print("ROUGE Score:", rouge_score)
```

ROUGE score: output

```
ROUGE Score: {'rouge1_fmeasure': tensor(0.8889),
              'rouge1_precision': tensor(0.8000),
              'rouge1_recall': tensor(1.),
              'rouge2_fmeasure': tensor(0.8571),
              'rouge2_precision': tensor(0.7500),
              'rouge2_recall': tensor(1.),
              'rougeL_fmeasure': tensor(0.8889),
              'rougeL_precision': tensor(0.8000),
              'rougeL_recall': tensor(1.),
              'rougeLsum_fmeasure': tensor(0.8889),
              'rougeLsum_precision': tensor(0.8000),
              'rougeLsum_recall': tensor(1.)}
```

Considerations and limitations

- Evaluate word presence, not semantic understanding
- Sensitive to the length of the generated text
- Quality of reference text affects the scores

Let's practice!

DEEP LEARNING FOR TEXT WITH PYTORCH