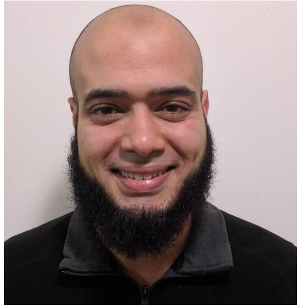
Machine Learning

LSTM in PyTorch

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

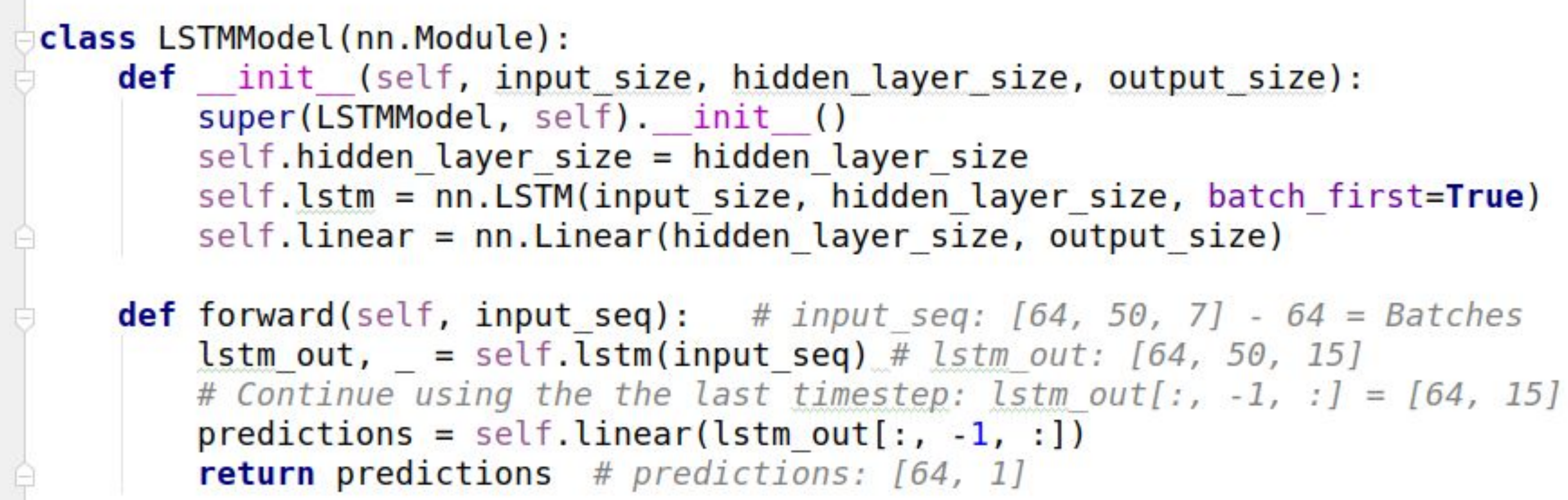


Coding LSTMs

- Tackling LSTMs (in general RNNs) requires that simple question first:
    - Is the input length sequence or variable length sequence
- Fixed length sequence
    - For example, all your videos are 40 frames. Each frame is represented with 140 features
- Variable length sequence
    - For example, Each videos is between 10-40 frames.
    - In theory, nothing wrong or challenging about that
    - However, it imposes efficiency limitation to process batches of inputs
        - As each sequence has different length

Fixed Length sequence

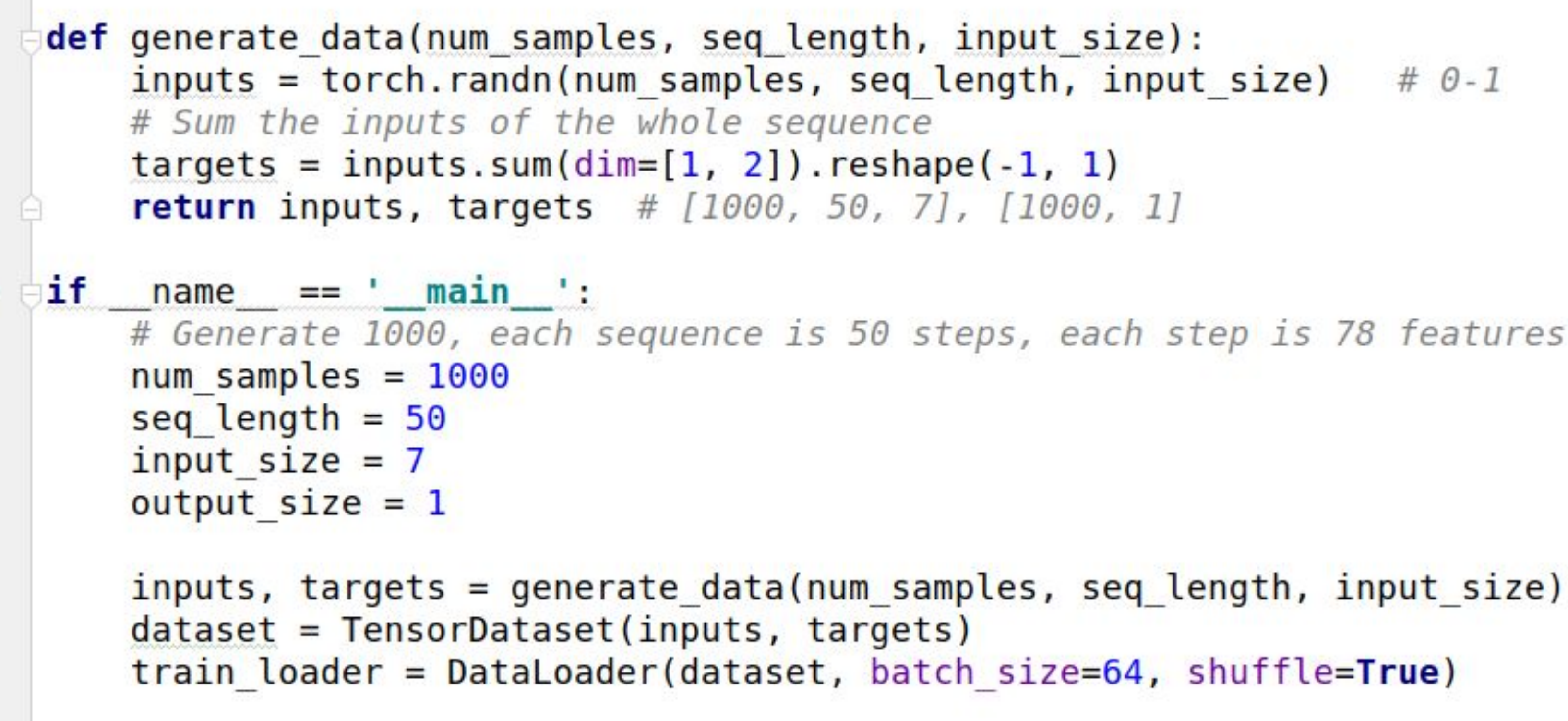
- Straightforward processing class.
- It receives a sequence and feed it to LSTM layer
  - Batch first: input is batch x sequence x features
- Here we use only the last LSTM step



```python
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        super(LSTMModel, self).__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size, batch_first=True)
        self.linear = nn.Linear(hidden_layer_size, output_size)

    def forward(self, input_seq):   # input_seq: [64, 50, 7] - 64 = Batches
        lstm_out, _ = self.lstm(input_seq) # lstm_out: [64, 50, 15]
        # Continue using the the last timestep: lstm_out[:, -1, :] = [64, 15]
        predictions = self.linear(lstm_out[:, -1, :])
        return predictions  # predictions: [64, 1]
```

Normal Data Processing



```python
def generate_data(num_samples, seq_length, input_size):
    inputs = torch.randn(num_samples, seq_length, input_size)   # 0-1
    # Sum the inputs of the whole sequence
    targets = inputs.sum(dim=[1, 2]).reshape(-1, 1)
    return inputs, targets  # [1000, 50, 7], [1000, 1]

if __name__ == '__main__':
    # Generate 1000, each sequence is 50 steps, each step is 78 features
    num_samples = 1000
    seq_length = 50
    input_size = 7
    output_size = 1

    inputs, targets = generate_data(num_samples, seq_length, input_size)
    dataset = TensorDataset(inputs, targets)
    train_loader = DataLoader(dataset, batch_size=64, shuffle=True)
```

Reduction: Variable to fixed with padding

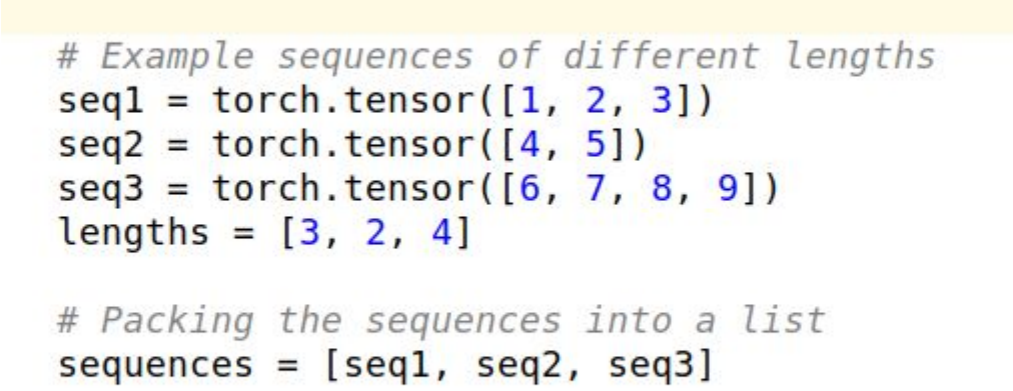
- If the length is fixed, our batch processing can be faster
- One workaround for variable length is to reduce to fixed length
- Simply we can do that by padding the sequence
- Good, however, processing padded items is waste of time
- We can implement the RNN processor in a way that efficiently discard irrelevant padded elements.
    - PyTorch 3 elements: sort, pad and pack

Padding and Packing

- Used for processing **variable**-length sequences in **batch** mode
- It packs all sequences together and performs padding
  - Find the max length and pad with it (zeros / especial token)
- The RNN knows how to skip efficiently these padded elements
  - important to **sort sequences** by their lengths in **descending** order before **packing** them
  - This order will speed the processing
- Procedure
  - First, sort your sequences
  - Second, use rnn.pad_sequence to pad each sequence with zeros
  - Third, Pack these padded sequence using pack_padded_sequence
    - PyTorch knows how to process efficiently these sorted padded sequences

Sorting and Padding Example

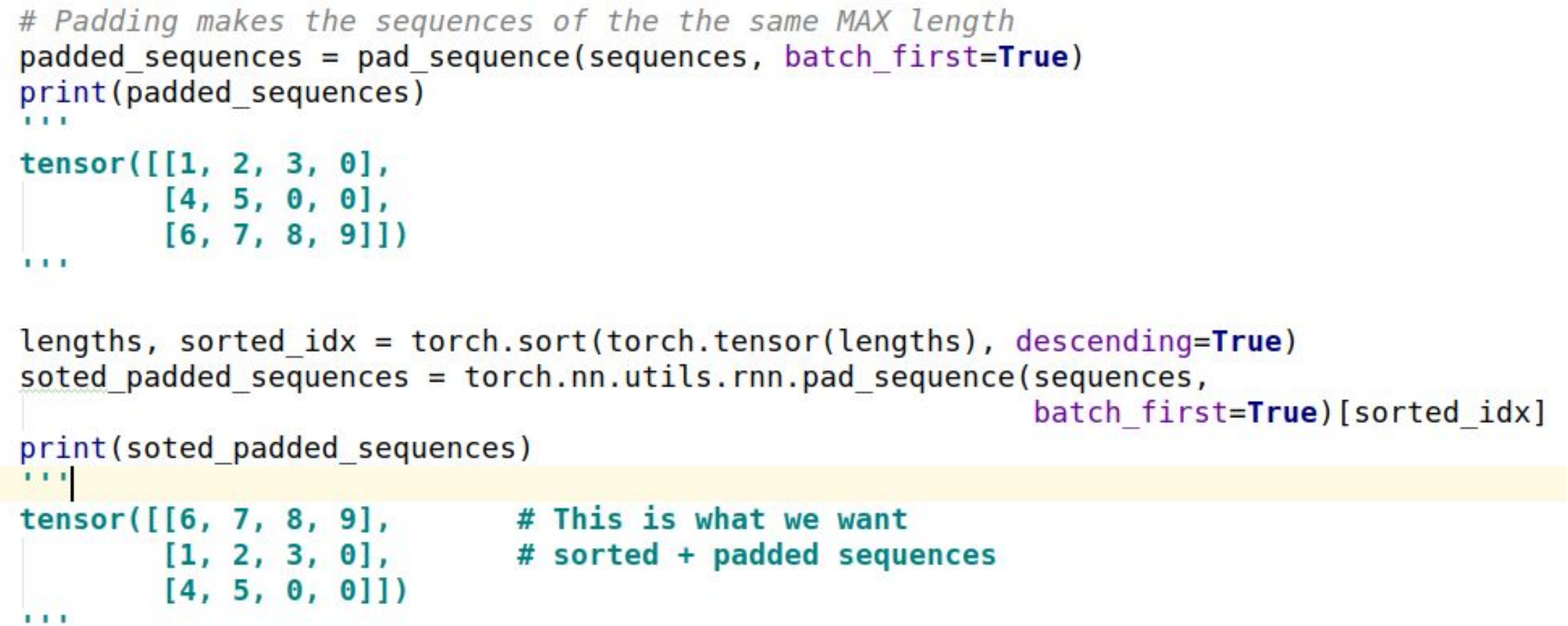
- Assume we have 3 sequences. They have different length



```python
# Example sequences of different lengths
seq1 = torch.tensor([1, 2, 3])
seq2 = torch.tensor([4, 5])
seq3 = torch.tensor([6, 7, 8, 9])
lengths = [3, 2, 4]

# Packing the sequences into a list
sequences = [seq1, seq2, seq3]
```

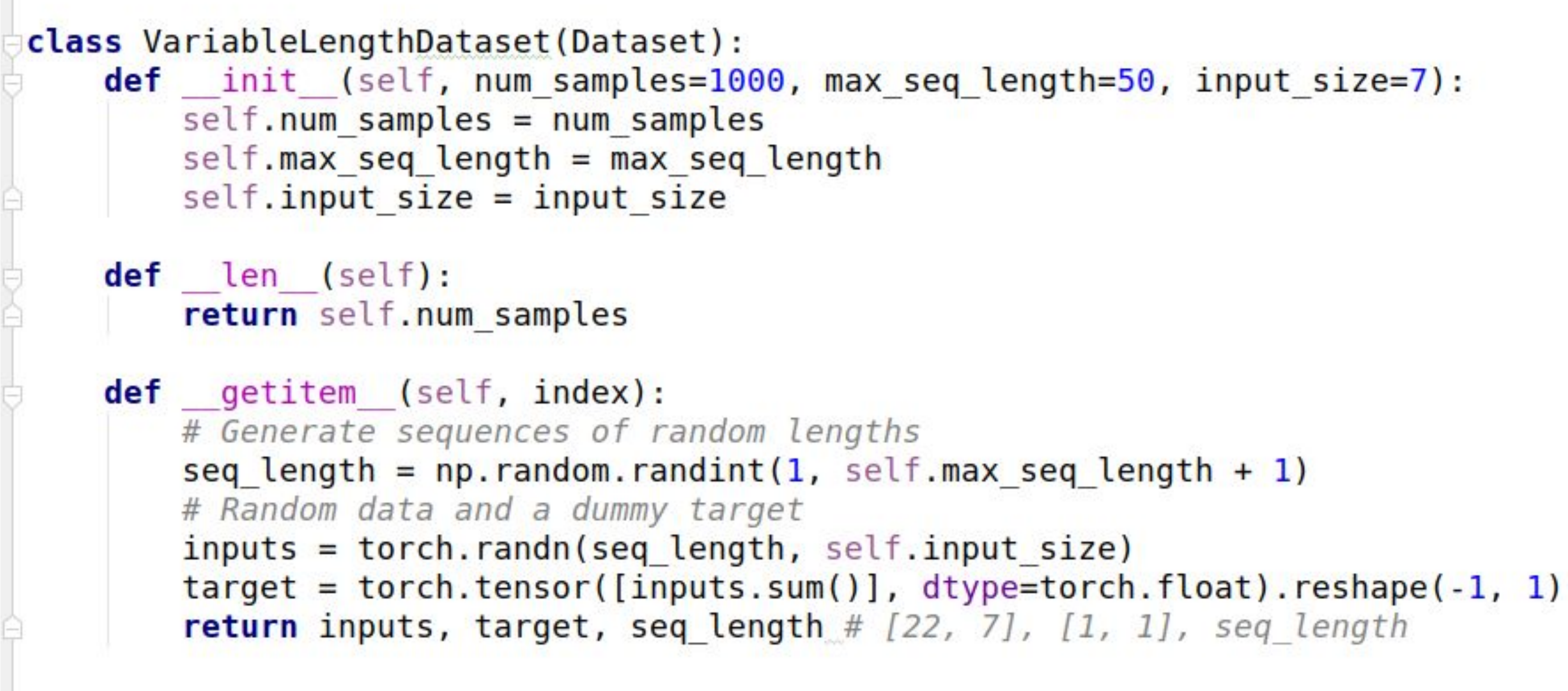
Sorting and Padding Example



```
# Padding makes the sequences of the the same MAX length
padded_sequences = pad_sequence(sequences, batch_first=True)
print(padded_sequences)
'''
tensor([[1, 2, 3, 0],
        [4, 5, 0, 0],
        [6, 7, 8, 9]])
'''

lengths, sorted_idx = torch.sort(torch.tensor(lengths), descending=True)
soted_padded_sequences = torch.nn.utils.rnn.pad_sequence(sequences,
                                                         batch_first=True)[sorted_idx]
print(soted_padded_sequences)
'''
tensor([[6, 7, 8, 9],       # This is what we want
        [1, 2, 3, 0],       # sorted + padded sequences
        [4, 5, 0, 0]])
'''
```

Simple Dataset Generator for variable length



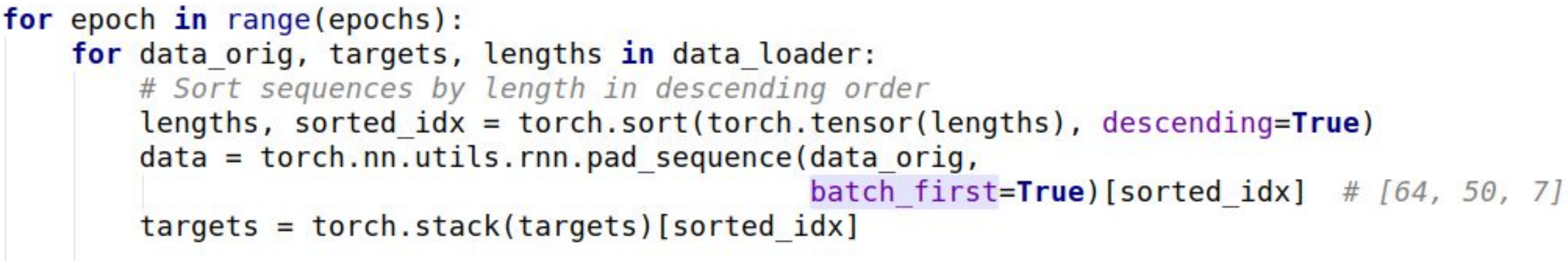
```python
class VariableLengthDataset(Dataset):
    def __init__(self, num_samples=1000, max_seq_length=50, input_size=7):
        self.num_samples = num_samples
        self.max_seq_length = max_seq_length
        self.input_size = input_size

    def __len__(self):
        return self.num_samples

    def __getitem__(self, index):
        # Generate sequences of random lengths
        seq_length = np.random.randint(1, self.max_seq_length + 1)
        # Random data and a dummy target
        inputs = torch.randn(seq_length, self.input_size)
        target = torch.tensor([inputs.sum()], dtype=torch.float).reshape(-1, 1)
        return inputs, target, seq_length # [22, 7], [1, 1], seq_length
```

Iterating on batches

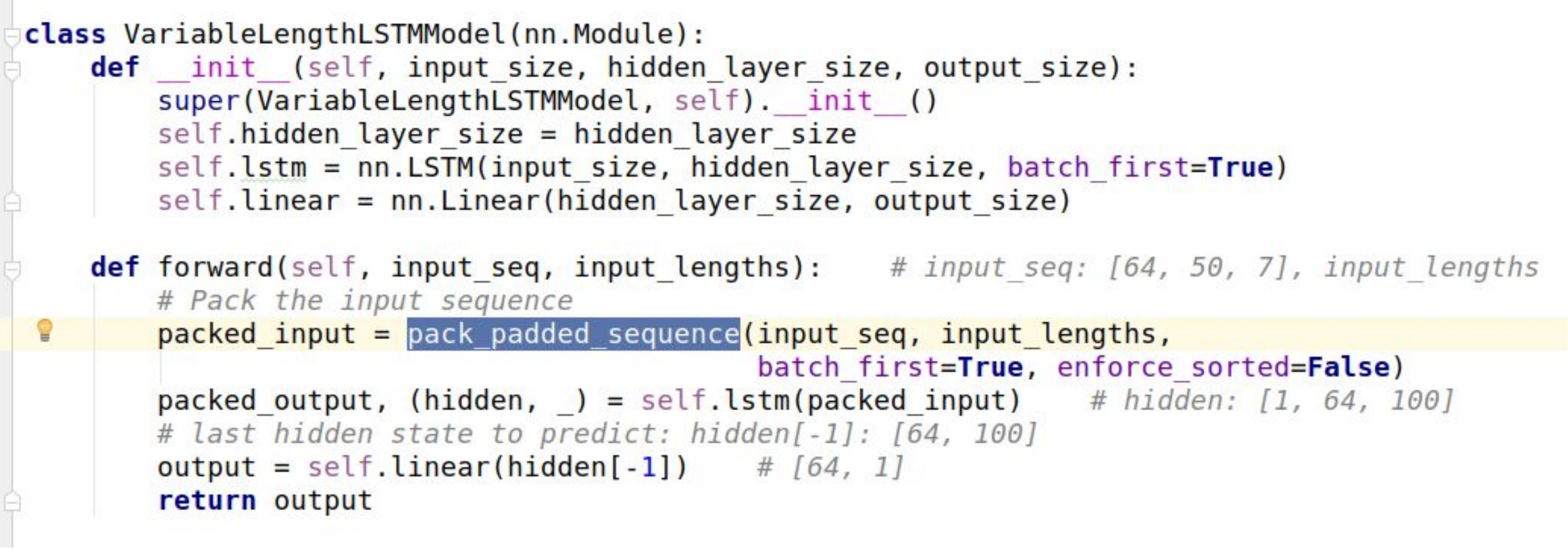
- Get batch of sequence
- Sort and pad it



```
for epoch in range(epochs):
    for data_orig, targets, lengths in data_loader:
        # Sort sequences by length in descending order
        lengths, sorted_idx = torch.sort(torch.tensor(lengths), descending=True)
        data = torch.nn.utils.rnn.pad_sequence(data_orig,
                                               batch_first=True)[sorted_idx]  # [64, 50, 7]
        targets = torch.stack(targets)[sorted_idx]
```

Variable LSTM

- Pack the sorted padded sequence in an object that PyTorch knows how to use to discard padded elements .



```
class VariableLengthLSTMModel(nn.Module):
    def __init__(self, input_size, hidden_layer_size, output_size):
        super(VariableLengthLSTMModel, self).__init__()
        self.hidden_layer_size = hidden_layer_size
        self.lstm = nn.LSTM(input_size, hidden_layer_size, batch_first=True)
        self.linear = nn.Linear(hidden_layer_size, output_size)

    def forward(self, input_seq, input_lengths):    # input_seq: [64, 50, 7], input_lengths
        # Pack the input sequence
        packed_input = pack_padded_sequence(input_seq, input_lengths,
                                            batch_first=True, enforce_sorted=False)
        packed_output, (hidden, _) = self.lstm(packed_input)    # hidden: [1, 64, 100]
        # last hidden state to predict: hidden[-1]: [64, 100]
        output = self.linear(hidden[-1])    # [64, 1]
        return output
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."