BiLSTMs (Bidirectional Long Short-Term Memory networks)

BiLSTMs are a type of recurrent neural network (RNN) that processes sequence data in both forward and backward directions. This bidirectionality allows the model **to capture context from both past and future time steps**, which is especially useful for tasks like natural language processing (NLP), speech recognition, and more.

Basic Concepts

1. LSTMs:
   - LSTMs are a type of RNN designed to handle long-term dependencies by mitigating the vanishing gradient problem using a memory cell and gates (input, forget, and output).
2. Bidirectional:
   - In a BiLSTM, two LSTM layers are used:
     1. One processes the sequence from start to end (forward).
     2. The other processes the sequence from end to start (backward).
3. Output:
   - The outputs from both directions are concatenated or combined at each time step, providing richer contextual information.

Below is an example of how to implement a BiLSTM in PyTorch:

```
import torch
import torch.nn as nn
```

```
# Define the BiLSTM model
class BiLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=1, bidirectional=True):
        super(BiLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.bidirectional = bidirectional
        self.lstm = nn.LSTM(input_size,
                            hidden_size,
                            num_layers,
                            bidirectional=bidirectional,
                            batch_first=True)
        self.fc = nn.Linear(hidden_size * 2 if bidirectional else hidden_size, output_size)

    def forward(self, x):
        # Initialize hidden and cell states
        h0 = torch.zeros(self.num_layers * (2 if self.bidirectional else 1), x.size(0), self.hidden_size)
        c0 = torch.zeros(self.num_layers * (2 if self.bidirectional else 1), x.size(0), self.hidden_size)

        # Forward propagate LSTM
        out, _ = self.lstm(x, (h0, c0))  # out: (batch_size, seq_length, hidden_size * 2 if bidirectional)

        # Pass the last hidden state to a fully connected layer
        out = self.fc(out[:, -1, :])  # Take the last time-step output
        return out
```

```
# Parameters
input_size = 10    # Number of input features
hidden_size = 20   # Number of hidden units
output_size = 2    # Number of output classes
num_layers = 2     # Number of stacked LSTM layers
seq_length = 5     # Length of input sequence
batch_size = 3     # Batch size
```

```
# Sample input
```

```
x = torch.randn(batch_size, seq_length, input_size)  # Shape: (batch_size, seq_length, input_size)

# Model
model = BiLSTM(input_size, hidden_size, output_size, num_layers)
output = model(x)

print("Output shape:", output.shape)  # Expected: (batch_size, output_size)
```

```
Output shape: torch.Size([3, 2])
```

Explanation of the Code

1. Model Initialization:
    1. input_size: Number of input features per time step.
    2. hidden_size: Number of hidden units in the LSTM.
    3. output_size: Size of the output layer.
    4. num_layers: Number of LSTM layers (stacked).
    5. bidirectional: Enables bidirectionality.
2. LSTM Layer:
    - nn.LSTM is used with the bidirectional=True flag for BiLSTM functionality.
3. Forward Pass:
    - Initializes the hidden and cell states (h0, c0) with zeros.
    - Processes the input sequence through the LSTM layer.
    - The final output is passed through a fully connected layer (fc) to map it to the desired output size.
4. Output:
    - The model outputs the prediction for the last time step of the sequence.

Applications of BiLSTMs

- Text Classification: Captures both past and future context for better understanding.
- Named Entity Recognition (NER): Considers surrounding words for accurate tagging.
- Machine Translation: Improves alignment between input and output sequences.