Applied Deep Learning with Tensorflow and PyTorch, Modern Recurrent Neural Networks

Gated Recurrent Units (GRU)

Gated RNNs can better capture dependencies for sequences with large time step distances. It contains resent gates and update gates. The reset gates help capture short-term dependencies in sequences. Update gates help capture long-term dependencies in sequences.

GRU from Scratch in PyTorch

```
import torch
from torch import nn

def gru(inputs, state, params):
    W_xz, W_hz, b_z, W_xr, W_hr, b_r, W_xh, W_hh, b_h, W_hq, b_q = params
    H, = state
    outputs = []
    for X in inputs:
        Z = torch.sigmoid((X @ W_xz) + (H @ W_hz) + b_z)
        R = torch.sigmoid((X @ W_xr) + (H @ W_hr) + b_r)
        H_tilda = torch.tanh((X @ W_xh) + ((R * H) @ W_hh) + b_h)
        H = Z * H + (1 - Z) * H_tilda
        Y = H @ W_hq + b_q
        outputs.append(Y)
    return torch.cat(outputs, dim=0), (H,)
```

GRU, Concise Implementation

```
import tensorflow as tf

def gru(inputs, state, params):
    W_xz, W_hz, b_z, W_xr, W_hr, b_r, W_xh, W_hh, b_h, W_hq, b_q = params
    H, = state
    outputs = []
    for X in inputs:
        X = tf.reshape(X, [-1, W_xh.shape[0]])
        Z = tf.sigmoid(tf.matmul(X, W_xz) + tf.matmul(H, W_hz) + b_z)
        R = tf.sigmoid(tf.matmul(X, W_xr) + tf.matmul(H, W_hr) + b_r)
        H_tilda = tf.tanh(tf.matmul(X, W_xh) + tf.matmul(R * H, W_hh) + b_h)
        H = Z * H + (1 - Z) * H_tilda
        Y = tf.matmul(H, W_hq) + b_q
        outputs.append(Y)
    return tf.concat(outputs, axis=0), (H,)
```

GRU from Scratch in TensorFlow

```
    num_inputs = vocab_size
gru_layer = nn.GRU(num_inputs, num_hiddens)
model = d2l.RNNModel(gru_layer, len(vocab))
model = model.to(device)
d2l.train_ch8(model, train_iter, vocab, lr, num_epochs, device)
```

Long Short-Term Memory (LSTM)

- LSTMs have three types of gates: input gates, forget gates, and output gates that control the flow of information.
- The hidden layer output of LSTM includes the hidden state and the memory cell. Only the hidden state is passed into the output layer. The memory cell is entirely internal.
- LSTMs can alleviate vanishing and exploding gradients.

LSTM from Scratch in PyTorch

```
import torch
from torch import nn

def lstm(inputs, state, params):
    [
        W_xi, W_hi, b_i, W_xf, W_hf, b_f, W_xo, W_ho, b_o, W_xc, W_hc, b_c,
        W_hq, b_q] = params
    (H, C) = state
    outputs = []
    for X in inputs:
        I = torch.sigmoid((X @ W_xi) + (H @ W_hi) + b_i)
        F = torch.sigmoid((X @ W_xf) + (H @ W_hf) + b_f)
        O = torch.sigmoid((X @ W_xo) + (H @ W_ho) + b_o)
        C_tilda = torch.tanh((X @ W_xc) + (H @ W_hc) + b_c)
        C = F * C + I * C_tilda
        H = O * torch.tanh(C)
        Y = (H @ W_hq) + b_q
        outputs.append(Y)
    return torch.cat(outputs, dim=0), (H, C)
```

LSTM from Scratch in TensorFlow

```
import tensorflow as tf

def lstm(inputs, state, params):
    W_xi, W_hi, b_i, W_xf, W_hf, b_f, W_xo, W_ho, b_o, W_xc, W_hc, b_c, W_hq, b_q = par
    (H, C) = state
    outputs = []
    for X in inputs:
        X = tf.reshape(X, [-1, W_xi.shape[0]])
        I = tf.sigmoid(tf.matmul(X, W_xi) + tf.matmul(H, W_hi) + b_i)
        F = tf.sigmoid(tf.matmul(X, W_xf) + tf.matmul(H, W_hf) + b_f)
        O = tf.sigmoid(tf.matmul(X, W_xo) + tf.matmul(H, W_ho) + b_o)
        C_tilda = tf.tanh(tf.matmul(X, W_xc) + tf.matmul(H, W_hc) + b_c)
        C = F * C + I * C_tilda
        H = O * tf.tanh(C)
        Y = tf.matmul(H, W_hq) + b_q
        outputs.append(Y)
    return tf.concat(outputs, axis=0), (H, C)
```

Deep Recurrent Neural Networks

Deep RNN in PyTorch

```
import torch
from torch import nn

batch_size, num_steps = 32, 35
vocab_size, num_hiddens, num_layers = len(vocab), 256, 2
num_inputs = vocab_size
device = try_gpu()
lstm_layer = nn.LSTM(num_inputs, num_hiddens, num_layers)
model = RNNModel(lstm_layer, len(vocab))
model = model.to(device)
```

Deep RNN in TensorFlow

```
import tensorflow as tf

vocab_size, num_hiddens, num_layers, num_inputs = len(vocab), 256, 2, vocab_size
strategy = tf.distribute.OneDeviceStrategy(device_name)
rnn_cells = [tf.keras.layers.LSTMCell(num_hiddens) for _ in range(num_layers)]
stacked_lstm = tf.keras.layers.StackedRNNCells(rnn_cells)
lstm_layer = tf.keras.layers.RNN(stacked_lstm, time_major=True,
                                 return_sequences=True, return_state=True)
with strategy.scope():
    model = RNNModel(lstm_layer, len(vocab))
```

Bidirectional Recurrent Neural Networks

In bidirectional RNNs, the hidden state for each time step is simultaneously determined by the data prior to and after the current time step.
In bidirectional RNNs, the hidden state for each time step is simultaneously determined by the data prior to and after the current time step.

Bidirectional RNN

```
import torch
from torch import nn

batch_size, num_steps, device = 32, 35, d2l.try_gpu()
train_iter, vocab = d2l.load_data_time_machine(batch_size, num_steps)

vocab_size, num_hiddens, num_layers = len(vocab), 256, 2
num_inputs = vocab_size
lstm_layer = nn.LSTM(num_inputs, num_hiddens, num_layers, bidirectional=True)
model = RNNModel(lstm_layer, len(vocab))
model = model.to(device)

num_epochs, lr = 500, 1
train(model, train_iter, vocab, lr, num_epochs, device)
```