rnn-gluon

April 9, 2019

0.0.1 Gluon Implementation in Recurrent Neural Networks

```
In [1]: import sys
        sys.path.insert(0, '..')

        import d2l
        import math
        from mxnet import autograd, gluon, init, nd
        from mxnet.gluon import loss as gloss, nn, rnn
        import time

        (corpus_indices, char_to_idx, idx_to_char,
         vocab_size) = d2l.load_data_time_machine()
```

0.0.2 Define the Model

```
In [2]: num_hiddens = 256
        rnn_layer = rnn.RNN(num_hiddens)
        rnn_layer.initialize()
```

Then, we call the `rnn_layer`'s member function `begin_state` to return hidden state list for initialization. It has an element of the shape (number of hidden layers, batch size, number of hidden units).

```
In [3]: batch_size = 2
        state = rnn_layer.begin_state(batch_size=batch_size)
        state[0].shape
```

```
Out[3]: (1, 2, 256)
```

0.0.3 RNN Layer in Action

```
In [4]: num_steps = 35
        X = nd.random.uniform(shape=(num_steps, batch_size, vocab_size))
        Y, state_new = rnn_layer(X, state)
        print(X.shape, len(state), state[0].shape)
        print(Y.shape, len(state_new), state_new[0].shape)
```

```
(35, 2, 43) 1 (1, 2, 256)
(35, 2, 256) 1 (1, 2, 256)
```

0.0.4 RNN Block

```
In [5]: # This class has been saved in the d2l package for future use.
        class RNNModel(nn.Block):
            def __init__(self, rnn_layer, vocab_size, **kwargs):
                super(RNNModel, self).__init__(**kwargs)
                self.rnn = rnn_layer
                self.vocab_size = vocab_size
                self.dense = nn.Dense(vocab_size)

            def forward(self, inputs, state):
                # Get the one-hot vector representation by transposing the input
                # to (num_steps, batch_size).
                X = nd.one_hot(inputs.T, self.vocab_size)
                Y, state = self.rnn(X, state)
                # The fully connected layer will first change the shape of Y to
                # (num_steps * batch_size, num_hiddens).
                # Its output shape is (num_steps * batch_size, vocab_size).
                output = self.dense(Y.reshape((-1, Y.shape[-1])))
                return output, state

            def begin_state(self, *args, **kwargs):
                return self.rnn.begin_state(*args, **kwargs)
```

0.0.5 Prediction

```
In [6]: def predict_rnn_gluon(prefix, num_chars, model, vocab_size, ctx, idx_to_char, char_to_i
            # Use model's member function to initialize the hidden state.
            state = model.begin_state(batch_size=1, ctx=ctx)
            output = [char_to_idx[prefix[0]]]
            for t in range(num_chars + len(prefix) - 1):
                X = nd.array([output[-1]], ctx=ctx).reshape((1, 1))
                (Y, state) = model(X, state)
                # Forward computation does not require incoming model parameters.
                if t < len(prefix) - 1:
                    output.append(char_to_idx[prefix[t + 1]])
                else:
                    output.append(int(Y.argmax(axis=1).asscalar()))
            return ''.join([idx_to_char[i] for i in output])
```

0.0.6 Prediction with Garbage Parameters

```
In [7]: ctx = d2l.try_gpu()
        model = RNNModel(rnn_layer, vocab_size)
```

```
model.initialize(force_reinit=True, ctx=ctx)
predict_rnn_gluon('traveller', 10, model, vocab_size, ctx, idx_to_char,
                  char_to_idx)
```

```
Out[7]: 'travelleruem]huem]h'
```

Next, implement the training function. Its algorithm is the same as in the previous section, but only random sampling is used here to read the data.

```
In [8]: # This function is saved in the d2l package for future use.
def train_and_predict_rnn_gluon(model, num_hiddens, vocab_size, ctx,
                                corpus_indices, idx_to_char, char_to_idx,
                                num_epochs, num_steps, lr, clipping_theta,
                                batch_size, pred_period, pred_len, prefixes):
    loss = gloss.SoftmaxCrossEntropyLoss()
    model.initialize(ctx=ctx, force_reinit=True, init=init.Normal(0.01))
    trainer = gluon.Trainer(model.collect_params(), 'sgd',
                            {'learning_rate': lr, 'momentum': 0, 'wd': 0})

    for epoch in range(num_epochs):
        l_sum, n, start = 0.0, 0, time.time()
        data_iter = d2l.data_iter_consecutive(
            corpus_indices, batch_size, num_steps, ctx)
        state = model.begin_state(batch_size=batch_size, ctx=ctx)
        for X, Y in data_iter:
            for s in state:
                s.detach()
            with autograd.record():
                (output, state) = model(X, state)
                y = Y.T.reshape((-1,))
                l = loss(output, y).mean()
            l.backward()
            # Clip the gradient.
            params = [p.data() for p in model.collect_params().values()]
            d2l.grad_clipping(params, clipping_theta, ctx)
            # Since the error has already taken the mean, the gradient does
            # not need to be averaged.
            trainer.step(1)
            l_sum += l.asscalar() * y.size
            n += y.size

        if (epoch + 1) % pred_period == 0:
            print('epoch %d, perplexity %f, time %.2f sec' % (
                epoch + 1, math.exp(l_sum / n), time.time() - start))
            for prefix in prefixes:
                print(' -', predict_rnn_gluon(
                    prefix, pred_len, model, vocab_size, ctx, idx_to_char,
                    char_to_idx))
```

Train the model using the same hyper-parameters as previously.

```
In [9]: num_epochs, batch_size, lr, clipping_theta = 200, 32, 1e2, 1e-2
        pred_period, pred_len, prefixes = 50, 50, ['traveller', 'time traveller']
        train_and_predict_rnn_gluon(model, num_hiddens, vocab_size, ctx,
                                    corpus_indices, idx_to_char, char_to_idx,
                                    num_epochs, num_steps, lr, clipping_theta,
                                    batch_size, pred_period, pred_len, prefixes)
```

```
epoch 50, perplexity 4.198955, time 0.17 sec
 - traveller.  'not and have not master and the precentions an
 - time traveller.  'not and have not master and the precentions an
epoch 100, perplexity 2.001583, time 0.18 sec
 - traveller.  'you can show black and the latter the pors and
 - time traveller.  'you can show black and the latter the pors and
epoch 150, perplexity 1.509388, time 0.19 sec
 - traveller.  'you cannot move a lear direction, and why shou
 - time traveller ceat in the freeen this is so extension in spocen
epoch 200, perplexity 1.299645, time 0.18 sec
 - traveller. 'but you will never convenient for the historian
 - time traveller, with a seeal they would certainly it traced such
```