rnn-scratch

April 9, 2019

1 Building a Recurrent Neural Network from Scratch

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

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

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

1.0.1 One-hot Encoding

In [2]:
```
nd.one_hot(nd.array([0, 2, 30]), vocab_size)
```

Out[2]:
```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 3x43 @cpu(0)>
```

1.0.2 Converting an entire minibatch to one-hot encoded data.

In [3]:
```
def to_onehot(X, size):
    return [nd.one_hot(x, size) for x in X.T]

X = nd.arange(10).reshape((2, 5))
inputs = to_onehot(X, vocab_size)
print(len(inputs), inputs[0].shape)
print(inputs)
```

```
5 (2, 43)
[
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>,
[[0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>,
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>,
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>,
[[0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>]
```

```
In [4]: # Initializing Parameters
        num_inputs, num_hiddens, num_outputs = vocab_size, 512, vocab_size
        ctx = d2l.try_gpu()

        # Create the parameters of the model, initialize them and attach gradients
        def get_params():
            def _one(shape):
                return nd.random.normal(scale=0.01, shape=shape, ctx=ctx)

            # Hidden layer parameters
            W_xh = _one((num_inputs, num_hiddens))
            W_hh = _one((num_hiddens, num_hiddens))
            b_h = nd.zeros(num_hiddens, ctx=ctx)
            # Output layer parameters
            W_hq = _one((num_hiddens, num_outputs))
            b_q = nd.zeros(num_outputs, ctx=ctx)
            # Attach a gradient
            params = [W_xh, W_hh, b_h, W_hq, b_q]
            for param in params:
```

```
            param.attach_grad()
        return params
```

1.0.3 RNN Model

```
In [5]: # return tuples such that we can extend this later
        def init_rnn_state(batch_size, num_hiddens, ctx):
            return (nd.zeros(shape=(batch_size, num_hiddens), ctx=ctx), )
```

```
In [6]: def rnn(inputs, state, params):
            # Both inputs and outputs are composed of num_steps matrices
            # of the shape (batch_size, vocab_size).
            W_xh, W_hh, b_h, W_hq, b_q = params
            H, = state
            outputs = []
            for X in inputs:
                H = nd.tanh(nd.dot(X, W_xh) + nd.dot(H, W_hh) + b_h)
                Y = nd.dot(H, W_hq) + b_q
                outputs.append(Y)
            return outputs, (H,)
```

Let's run a simple test to check whether inputs and outputs are accurate. In particular, we check output dimensions, the number of outputs and ensure that the hidden state hasn't changed.

```
In [7]: state = init_rnn_state(X.shape[0], num_hiddens, ctx)
        inputs = to_onehot(X.as_in_context(ctx), vocab_size)
        params = get_params()
        outputs, state_new = rnn(inputs, state, params)
        print(len(inputs), inputs[0].shape, state[0].shape)
        print(len(outputs), outputs[0].shape, state_new[0].shape)
```

```
5 (2, 43) (2, 512)
5 (2, 43) (2, 512)
```

1.0.4 Prediction Function

```
In [8]: # This function is saved in the d2l package for future use.
        def predict_rnn(prefix, num_chars, rnn, params, init_rnn_state,
                        num_hiddens, vocab_size, ctx, idx_to_char, char_to_idx):
            state = init_rnn_state(1, num_hiddens, ctx)
            output = [char_to_idx[prefix[0]]]
            for t in range(num_chars + len(prefix) - 1):
                # The output of the previous time step is taken
                # as the input of the current time step.
                X = to_onehot(nd.array([output[-1]], ctx=ctx), vocab_size)
                # Calculate the output and update the hidden state.
                (Y, state) = rnn(X, state, params)
                # The input to the next time step is the character in
```

```
            # the prefix or the current best predicted character.
            if t < len(prefix) - 1:
                output.append(char_to_idx[prefix[t + 1]])
            else:
                # This is maximum likelihood decoding, not sampling
                output.append(int(Y[0].argmax(axis=1).asscalar()))
        return ''.join([idx_to_char[i] for i in output])
```

1.0.5 Testing the prediction function

```
In [9]: predict_rnn('traveller', 10, rnn, params, init_rnn_state, num_hiddens,
                    vocab_size, ctx, idx_to_char, char_to_idx)
```

```
Out[9]: 'traveller;;x(.v9cee'
```

1.0.6 Gradient Clipping

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g}.$$

```
In [10]: # This function is saved in the d2l package for future use.
         def grad_clipping(params, theta, ctx):
             norm = nd.array([0], ctx)
             for param in params:
                 norm += (param.grad ** 2).sum()
             norm = norm.sqrt().asscalar()
             if norm > theta:
                 for param in params:
                     param.grad[:] *= theta / norm
```

1.0.7 Training Loop

```
In [11]: # This function is saved in the d2l package for future use.
         def train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                                   vocab_size, ctx, corpus_indices, idx_to_char,
                                   char_to_idx, is_random_iter, num_epochs, num_steps,
                                   lr, clipping_theta, batch_size, pred_period,
                                   pred_len, prefixes):
             if is_random_iter:
                 data_iter_fn = d2l.data_iter_random
             else:
                 data_iter_fn = d2l.data_iter_consecutive
             params = get_params()
             loss = gloss.SoftmaxCrossEntropyLoss()

             for epoch in range(num_epochs):
                 if not is_random_iter:
                     # If adjacent sampling is used, the hidden state is initialized
                     # at the beginning of the epoch.
```

```
            state = init_rnn_state(batch_size, num_hiddens, ctx)
        l_sum, n, start = 0.0, 0, time.time()
        data_iter = data_iter_fn(corpus_indices, batch_size, num_steps, ctx)
        for X, Y in data_iter:
            if is_random_iter:
                # If random sampling is used, the hidden state is initialized
                # before each mini-batch update.
                state = init_rnn_state(batch_size, num_hiddens, ctx)
            else:
                # Otherwise, the detach function needs to be used to separate
                # the hidden state from the computational graph to avoid
                # backpropagation beyond the current sample.
                for s in state:
                    s.detach()
            with autograd.record():
                inputs = to_onehot(X, vocab_size)
                # outputs is num_steps terms of shape (batch_size, vocab_size)
                (outputs, state) = rnn(inputs, state, params)
                # after stitching it is (num_steps * batch_size, vocab_size).
                outputs = nd.concat(*outputs, dim=0)
                # The shape of Y is (batch_size, num_steps), and then becomes
                # a vector with a length of batch * num_steps after
                # transposition. This gives it a one-to-one correspondence
                # with output rows.
                y = Y.T.reshape((-1,))
                # Average classification error via cross entropy loss.
                l = loss(outputs, y).mean()
            l.backward()
            grad_clipping(params, clipping_theta, ctx)  # Clip the gradient.
            d2l.sgd(params, lr, 1)
            # Since the error is the mean, no need to average gradients here.
            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(
                    prefix, pred_len, rnn, params, init_rnn_state,
                    num_hiddens, vocab_size, ctx, idx_to_char, char_to_idx))
```

1.0.8 Experiments with a Sequence Model

Now we can train the model. First, we need to set the model hyper-parameters. To allow for some meaningful amount of context we set the sequence length to 64. To get some intuition of how well the model works, we will have it generate 50 characters every 50 epochs of the training phase. In particular, we will see how training using the 'separate' and 'sequential' term generation will

affect the performance of the model.

```
In [12]: num_epochs, num_steps, batch_size, lr, clipping_theta = 500, 64, 32, 1e2, 1e-2
         pred_period, pred_len, prefixes = 50, 50, ['traveller', 'time traveller']
```

Let's use random sampling to train the model and produce some text.

```
In [13]: train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                               vocab_size, ctx, corpus_indices, idx_to_char,
                               char_to_idx, True, num_epochs, num_steps, lr,
                               clipping_theta, batch_size, pred_period, pred_len,
                               prefixes)
```

```
epoch 50, perplexity 8.954775, time 0.56 sec
 - travellere than ' said the than ' said the than ' said the
 - time travellere than ' said the than ' said the than ' said the
epoch 100, perplexity 7.555069, time 0.50 sec
 - travellere there there there there there there there there
 - time traveller.  'ican an ally ano dimensions ale there there th
epoch 150, perplexity 5.574858, time 0.55 sec
 - traveller said the payce for the this this as the pais the
 - time traveller said the payce for the this this as the pais the
epoch 200, perplexity 3.716279, time 0.53 sec
 - traveller spoendif the thing. be' sere whe reples, have a a
 - time traveller s of some than it meat a ther that allis batwerid
epoch 250, perplexity 2.434598, time 0.53 sec
 - traveller. 'ice as the thing thee iline thin to seas in tha
 - time traveller smiled round at us sciplling way bl beaconle is t
epoch 300, perplexity 1.829528, time 0.60 sec
 - traveller proceeded, 'any real body must have long magham
 - time traveller proceeded, 'any real body must have long magham
epoch 350, perplexity 1.563005, time 0.58 sec
 - traveller smiled round at us. then, still smilidireaid tiee
 - time traveller smiled round at us. then, still smilidireaid tiee
epoch 400, perplexity 1.423053, time 0.54 sec
 - traveller pere hos has stofit ais y uthids!'  'no 'rof oo t
 - time traveller heme the fimen it us in there hest ps pack to the
epoch 450, perplexity 1.373718, time 0.51 sec
 - traveller.  'it's against reason,' said filby, an argumenta
 - time traveller smiled round at us. the wist expeas expspthat sur
epoch 500, perplexity 1.347638, time 0.50 sec
 - traveller.  'it's against reason,' said the time traveller.
 - time traveller smiled round at us. then, still smiling faintly,
```

1.0.9 Sequential Sampling

```
In [14]: train_and_predict_rnn(rnn, get_params, init_rnn_state, num_hiddens,
                               vocab_size, ctx, corpus_indices, idx_to_char,
```

```
                                char_to_idx, False, num_epochs, num_steps, lr,
                                clipping_theta, batch_size, pred_period, pred_len,
                                prefixes)
```

```
epoch 50, perplexity 8.766011, time 0.52 sec
 - traveller and the the the the the the the the the the t
 - time traveller and the the the the the the the the the the t
epoch 100, perplexity 7.066054, time 0.68 sec
 - traveller.  'the the the the the the the the the the th
 - time traveller.  'that in ton the time trought ons of the the th
epoch 150, perplexity 4.490671, time 0.50 sec
 - traveller.  'experiment onve iftly the gat anllyou the time
 - time traveller.  'experiment onve iftly the gat anllyou the time
epoch 200, perplexity 2.663523, time 0.52 sec
 - traveller sathi ghis tho gension anowe racked anded, 'the e
 - time traveller s movelly urfelw and the mather ave a thit here w
epoch 250, perplexity 1.589984, time 0.54 sec
 - traveller come and asmodi wist as our eathe bedingo menite
 - time traveller some on theremole have g tile fored annoul ant of
epoch 300, perplexity 1.253643, time 0.54 sec
 - traveller cfour hime some than gpint a for the hrmame that
 - time traveller cfore whie wime te sive yot cas of thacerefalled
epoch 350, perplexity 1.250476, time 0.55 sec
 - traveller cald focre mathes thing that fichris dimensions,
 - time traveller, with a slight accession of cheerfulness. 'row he
epoch 400, perplexity 1.169369, time 0.50 sec
 - traveller cabe back, and filby's anecdote collapsed.  the t
 - time traveller came back, and filby's anecdote collapsed.  the t
epoch 450, perplexity 1.139071, time 0.49 sec
 - traveller (for so it will be convenient to speak of him) wa
 - time traveller, with a slight accession of cheerfulness. 'really
epoch 500, perplexity 1.166729, time 0.58 sec
 - traveller that furmed wo ht it our oumengat of hemuthras io
 - time traveller, after the pause required for the proper assimila
```