Text Preprocessing

In [1]:

```
import sys
sys.path.insert(0, '..')

from mxnet import nd
import random

with open('data/timemachine.txt', 'r') as f:
    lines = f.readlines()
    raw_dataset = ' '.join(' '.join(lines).lower().split())

print('number of characters: ', len(raw_dataset))
print(raw_dataset[0:70])
```

```
number of characters:  178605
the time machine, by h. g. wells [1898] i the time traveller (for so i
```

Character Index

In [2]:

```
idx_to_char = list(set(raw_dataset))
char_to_idx = dict([(char, i) for i, char in enumerate(idx_to_char)])
vocab_size = len(char_to_idx)
print(char_to_idx)
```

```
{',': 0, 'm': 1, 'p': 2, 'y': 3, '[': 4, '?': 5, 'k': 6, 'z': 7, ')': 8, ' ':
9, '!': 10, 'w': 11, 'a': 12, '8': 13, "'": 14, 'o': 15, 'd': 16, '(': 17,
'"': 18, 'h': 19, 's': 20, 'g': 21, '.': 22, 'i': 23, '-': 24, 'j': 25, 'c': 2
6, 'n': 27, 'e': 28, ':': 29, '_': 30, ';': 31, '1': 32, 'x': 33, 'u': 34,
'r': 35, 'v': 36, 'b': 37, 'l': 38, 'f': 39, 'q': 40, '9': 41, ']': 42, 't': 4
3}
```

Converting it back to text

In [3]:

```
corpus_indices = [char_to_idx[char] for char in raw_dataset]
sample = corpus_indices[:20]
print('chars:', ''.join([idx_to_char[idx] for idx in sample]))
print('indices:', sample)
```

```
chars: the time machine, by
indices: [43, 19, 28, 9, 43, 23, 1, 28, 9, 1, 12, 26, 19, 23, 27, 28, 0, 9, 3
7, 3]
```

Random Sampling

In [4]:

```
# This function is saved in the d2l package for future use.
def data_iter_random(corpus_indices, batch_size, num_steps, ctx=None):
    # offset for the iterator over the data for uniform starts
    offset = int(random.uniform(0,num_steps))
    corpus_indices = corpus_indices[offset:]
    # subtract 1 extra since we need to account for the sequence length
    num_examples = ((len(corpus_indices) - 1) // num_steps) - 1
    # discard half empty batches
    num_batches = num_examples // batch_size
    example_indices = list(range(0, num_examples * num_steps, num_steps))
    random.shuffle(example_indices)

    # This returns a sequence of the length num_steps starting from pos.
    def _data(pos):
        return corpus_indices[pos: pos + num_steps]

    for i in range(0, batch_size * num_batches, batch_size):
        # batch_size indicates the random examples read each time.
        batch_indices = example_indices[i:(i+batch_size)]
        X = [_data(j) for j in batch_indices]
        Y = [_data(j + 1) for j in batch_indices]

        yield nd.array(X, ctx), nd.array(Y, ctx)
```

Example

Batch size 2 and time steps is 5 for a sequence of length 30.

In [5]:

```
my_seq = list(range(30))
for X, Y in data_iter_random(my_seq, batch_size=2, num_steps=5):
    print('X: ', X, '\nY:', Y)
```

```
X:
[[10. 11. 12. 13. 14.]
 [ 0.  1.  2.  3.  4.]]
<NDArray 2x5 @cpu(0)>
Y:
[[11. 12. 13. 14. 15.]
 [ 1.  2.  3.  4.  5.]]
<NDArray 2x5 @cpu(0)>
X:
[[ 5.  6.  7.  8.  9.]
 [15. 16. 17. 18. 19.]]
<NDArray 2x5 @cpu(0)>
Y:
[[ 6.  7.  8.  9. 10.]
 [16. 17. 18. 19. 20.]]
<NDArray 2x5 @cpu(0)>
```

Sequential partitioning

Adjacent positioning of minibatches. This way we can retain the latent state between batches.

In [6]:

```
# This function is saved in the d2l package for future use.
def data_iter_consecutive(corpus_indices, batch_size, num_steps, ctx=None):
    # offset for the iterator over the data for uniform starts
    offset = int(random.uniform(0,num_steps))
    # slice out data - ignore num_steps and just wrap around
    num_indices = ((len(corpus_indices) - offset) // batch_size) * batch_size
    indices = nd.array(corpus_indices[offset:(offset + num_indices)], ctx=ctx)
    indices = indices.reshape((batch_size,-1))
    # need to leave one last token since targets are shifted by 1
    num_epochs = ((num_indices // batch_size) - 1) // num_steps

    for i in range(0, num_epochs * num_steps, num_steps):
        X = indices[:,i:(i+num_steps)]
        Y = indices[:,(i+1):(i+1+num_steps)]
        yield X, Y
```

Example partitioning

```
In [7]: for X, Y in data_iter_consecutive(my_seq, batch_size=2, num_steps=6):
            print('X: ', X, '\nY:', Y)
```

```
X:
[[ 4.  5.  6.  7.  8.  9.]
 [17. 18. 19. 20. 21. 22.]]
<NDArray 2x6 @cpu(0)>
Y:
[[ 5.  6.  7.  8.  9. 10.]
 [18. 19. 20. 21. 22. 23.]]
<NDArray 2x6 @cpu(0)>
X:
[[10. 11. 12. 13. 14. 15.]
 [23. 24. 25. 26. 27. 28.]]
<NDArray 2x6 @cpu(0)>
Y:
[[11. 12. 13. 14. 15. 16.]
 [24. 25. 26. 27. 28. 29.]]
<NDArray 2x6 @cpu(0)>
```