Sequence models

Author: Juvid Aryaman
Last compiled: November 12, 2021

This document contains my personal notes on sequence models.

1. Embeddings

Embeddings are tensors. You interact with that tensor by indexing into it. It is often used to store encodings of collections of words. For example:

```
>>> nn.Embedding(vocab_sz, n_hidden)
```

creates a set of `vocab_sz` tensors, each of size `n_hidden`.

A common thing to do is to something like:

```
>>> embedding = nn.Embedding(10, 3)
>>> input = torch.LongTensor([[1,2,4,5],[4,3,2,9]])
>>> embedding(input)

tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
         [ 0.9124, -2.3616,  1.1151]]])
```

so you can see that the input is `[sentence1, sentence2]`, where sentence 1 consists of words `[1,2,4,5]`. As an output, we get the corresponding 3-vectors for each word. So the output is:

```
[[[embedding_word_1,    # length 3 vector
   embedding_word_2,
   embedding_word_4,
   embedding_word_5],

  [embedding_word_4,
   embedding_word_3,
   embedding_word_2,
   embedding_word_9]
]]
```

2. Linear layer

Applies a linear transformation to the incoming data: $y = xA^T + b$

```
>>> m = nn.Linear(20, 30)
>>> input = torch.randn(128, 20)
>>> output = m(input)
>>> print(output.size())
torch.Size([128, 30])
```

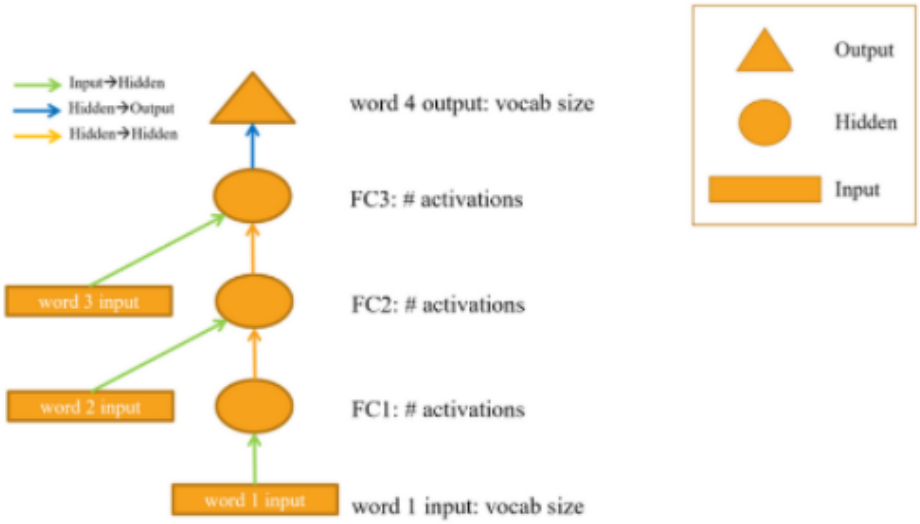

Figure 1. Graphical representation of RNN

3. Recurrent neural network

Torch, by default, applies a multi-layer Elman RNN. This is defined as applying the following function to each element of the input sequence

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{3.1}$$
$$y_t = \sigma_y(W_y h_t + b_y) \tag{3.2}$$

where x_t is an input vector, h_t is a hidden layer vector, y_t is an output vector, W, U, b are parameter matrices and vector, σ_h, σ_y are activation functions.

Bahdanau *et al.* (2014)

References

Bahdanau, D., K. Cho, and Y. Bengio, 2014 Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 .