PRML Assignment 4 Report

Implementation

RNN

Base Formula

RNN

> $$\mathbf{h}_t = tanh(\mathbf{h}_{t-1} * W_h + \mathbf{x}_t * W_x + \mathbf{b})$$

LSTM

> $$\begin{aligned} \mathbf{z} &= [\mathbf{h}_{t-1}, \mathbf{x}_t] \\ \mathbf{f}_t &= \sigma(W_f * \mathbf{z} + b_f) \\ &= \sigma(W_{fh} * \mathbf{h}_{t-1} + W_{fx} * \mathbf{x}_t + b_f) = \sigma(\mathbf{net}_f) \\ \mathbf{i}_t &= \sigma(W_i * \mathbf{z} + b_i) \\ &= \sigma(W_{ih} * \mathbf{h}_{t-1} + W_{ix} * \mathbf{x}_t + b_f) = \sigma(\mathbf{net}_i) \\ \tilde{\mathbf{c}}_t &= tanh(W_c * \mathbf{z} + b_c) \\ &= tanh(W_{ch} * \mathbf{h}_{t-1} + W_{cx} * \mathbf{x}_t + b_f) = \sigma(\mathbf{net}_{\tilde{c}}) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\ \mathbf{o}_t &= \sigma(W_o * \mathbf{z} + b_o) \\ &= \sigma(W_{oh} * \mathbf{h}_{t-1} + W_{ox} * \mathbf{x}_t + b_f) = \sigma(\mathbf{net}_o) \\ \mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t) \end{aligned}$$

Parameters

```python
class  RNN(nn.Module):
    def  __init__(self, vocab_size, input_dim, hidden_dim, output_dim):
    ...
```

Calculation

First, `embed` the input sequences, translate each word into a vector of `input_dim` .
Next, pass a whole sentence into the `RNN` or `LSTM` , get the final $\mathbf{h}_t$.
Finally, apply a `linear` transformation on $\mathbf{h}_t$ to get a vector of `output_dim` .

```python
word_seq =  self.embed(word_seq)
rnn_out, self.hidden =  self.rnn(word_seq, self.hidden)
```

```
y_pred =  self.linear(rnn_out)
```

CNN

Parameters

```
class  CNN(nn.Module):
    def  __init__(self, vocab_size, input_dim, output_dim, in_channels,
     out_channels, kernel_sizes, keep_probab):
    ...
```

Here, `in_channels` is always 1, while you can tune the `out_channels` .
Also, `kernel_sizes` is a list of 3 that can be tuned.

Calculation

First, `embed` the input sequences, translate each word into a vector of `input_dim` .
Next, apply `convolution` with 3 different kernel sizes to the input.
Then, concatenate the results from 3 layers and apply `dropout` function.
Finally, apply a `linear` transformation to get the output.

```
input  =  self.embed(word_seq)
input  =  input.unsqueeze(1)
# max_out: [batch_size, out_channels]
max_out1 =  self.conv_calc(input, self.conv1)
max_out2 =  self.conv_calc(input, self.conv2)
max_out3 =  self.conv_calc(input, self.conv3)
# all_out: [batch_size, num_kernels*out_channels]
all_maxout = torch.cat((max_out1, max_out2, max_out3), 1)
all_dropout =  self.dropout(all_maxout)
# logits: [batch_size, output_dim]
logits =  self.linear(all_dropout)
```

Data Processing

Data Source

I use `get_20newsgroups_data` to acquire 4000 news from 8 different categories as training data, and 400 as development data.

Vocabulary and DataSet

I use `Vocabulary` from `fastNLP` to build the vocabulary with `min_seq=10`.
And I use `DataSet` from `fastNLP` to build the train set and test set.

Also, once the `dataset` is built, it will be saved to disk, so that we don't have to rebuild everything the next time.

```
train, test = get_20newsgroups_data(categories)
train_set = create_dataset(train, train_size)
test_set = create_dataset(test, test_size)
# vocabulary
vocab = Vocabulary(min_freq=10)
test_set.apply(lambda  x: [vocab.add(word) for word in x['word_seq']])
vocab.build_vocab()
# word_seq to int
train_set.apply(lambda  x: [vocab.to_index(word) for word in x['word_seq']],
 new_field_name='word_seq')
test_set.apply(lambda  x: [vocab.to_index(word) for word in x['word_seq']],
 new_field_name='word_seq')
```

Training

I use `Trainer` from `fastNLP` to train the model.

```
trainer = Trainer(model=model, train_data=train_set, dev_data=test_set,
                loss=CrossEntropyLoss(pred='output', target='target'),
                metrics=AccuracyMetric(pred='pred', target='target'),
                optimizer=Adam(lr=lr),
                save_path='../model',
                batch_size=1,
                n_epochs=epochs)
trainer.train()
```

Result

> categories =
> ['comp.os.ms-windows.misc', 'rec.motorcycles', 'sci.space', 'sci.crypt', 'sci.electronics', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',]

LSTM (RNN)

> input_dim = 256
> hidden_dim = 128

acc = 51.25%

CNN

> kernel_sizes = [3, 4, 5]
> keep_proba = 0.5
> input_dim = 512
> out_channels = 256

acc = 85.50%

FastNLP

Pros

- The `vocabulary` and `DataSet` modules are super useful, and they save a lot of code.
- Also, `Trainer` is helpful, while you don't have to write the process of organizing a batch, calculating the loss, updating the weights and measuring the accuracy.

Cons

- For a greenhand, it can be confusing to figure out how the trainer feed the data into the model. `ie. how it organizes a whole batch of data.`
- It is kind of inconvenient to connect output and input with parameter names. `ie. the name of model input has to be word_seq (although you could rename), and the model returns a dict while the key determines what each value is for .`

Suggestion

- Write some examples of building custom model, including how to organize input and output.
- Let user to define how often to save the model, and how to name them.
- Provide some intermediate results in trainer for debugging.