

The dictionary is case-sensitive!

Test: I love paris in the spring time I love paris in the fall I love paris in the I love franch songs and to hear them in the afternoon
Train: I went to Paris last year becuae I wanted to have some fun
Valid: yesterday I went home to see my daughter which is called ayalla and also my wife which is called tanya

```
>>> corpus.dictionary.word2idx
{'and': 21, 'love': 25, 'Paris': 3, 'spring': 29, 'is': 18, 'in': 27, 'some': 9, 'see': 14, 'have': 8, 'year': 5, 'home': 13, 'franch': 32, 'also': 22, 'paris': 26, 'to': 2, 'tanya': 24, 'which': 17, 'becuase': 6, 'them': 35, 'I': 0, 'ayalla': 20, 'yesterday': 12, '<eos>': 11, 'hear': 34, 'fall': 31, 'last': 4, 'wanted': 7, 'the': 28, 'daughter': 16, 'wife': 23, 'afternoon': 36, 'time': 30, 'fun': 10, 'went': 1, 'my': 15, 'called': 19, 'songs': 33}

>>> corpus.dictionary.idx2word
['I', 'went', 'to', 'Paris', 'last', 'year', 'becuase', 'wanted', 'have', 'some', 'fun', '<eos>', 'yesterday', 'home', 'see', 'my', 'daughter', 'which', 'is', 'called', 'ayalla', 'and', 'also', 'wife', 'tanya', 'love', 'paris', 'in', 'the', 'spring', 'time', 'fall', 'franch', 'songs', 'hear', 'them', 'afternoon']
```

```
>>> corpus.valid
12
0
1
13
2
14
15
16
17
18
19
20
21
22
15
23
17
18
19
24
11
[torch.LongTensor of size 21]
```

```
>>> corpus.train
0
1
2
3
4
5
6
0
7
2
8
9
10
11
[torch.LongTensor of size 14]
```

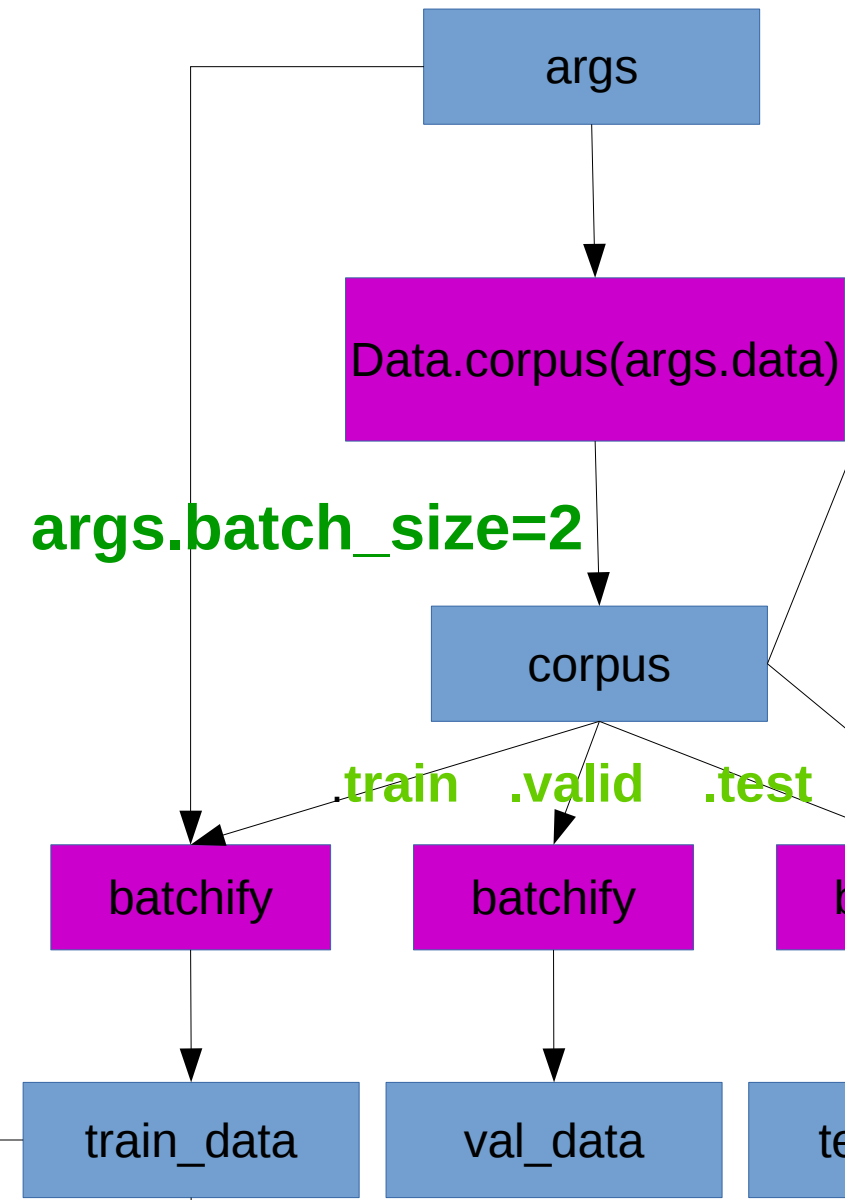
```
>>> corpus.test
0
25
26
27
28
29
30
0
25
26
27
28
0
25
32
33
21
2
34
35
27
28
36
11
[torch.LongTensor of size 30]
```

```
>>> train_data
0 0
1 7
2 2
3 8
4 9
5 10
6 11
[torch.LongTensor of size 7x2]
```

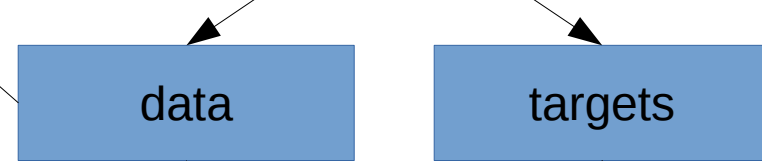
The words in train.txt were divided to batches; every raw is a batch. But the words order are at the columns!

seq_len = 3
Takes the data raw-wise!

args.batch_size=2



data, targets = get_batch(train_data, i)



l=1:
>>> data
Variable containing:
3 8
4 9
5 10
[torch.LongTensor of size 3x2]

l=0:
>>> data
Variable containing:
0 0
1 7
2 2
[torch.LongTensor of size 3x2]

l=1:
>>> targets
Variable containing:
4
9
5
10
6
11
[torch.LongTensor of size 6]

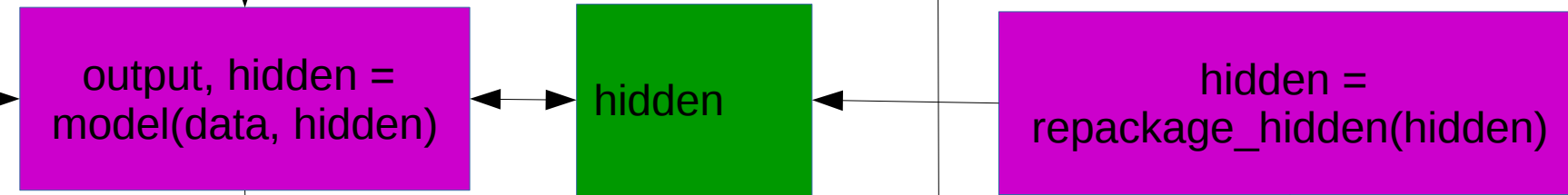
l=0:
>>> targets
Variable containing:
1
7
2
2
3
8
[torch.LongTensor of size 6]

RNNModel (
(drop): Dropout (p = 0.2)
(encoder): Embedding(37, 200)
(rnn): LSTM(200, 200, num_layers=2, dropout=0.2)
(decoder): Linear (200 -> 37)

Criterion:
CrossEntropyLoss ()

Single init @ train

hidden = model.init_hidden(args.batch_size)



```
>>> output.data.shape
(3L, 3L, 37L)
```

```
loss = criterion(output.view(-1, ntokens), targets)
loss.backward()

torch.nn.utils.clip_grad_norm(model.parameters(), args.clip)

for p in model.parameters():
    p.data.add_(-lr, p.grad.data)
```

INPUTS

OUTPUTS

```
I=0:  
>>> data  
Variable containing:  
0 0  
1 7  
2 2  
[torch.LongTensor of size 3x2]
```

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Our dim is 200

data

Self.encoder = nn.embedding

Word Embeddings

```
>>> emb.data.shape  
(3L, 2L, 200L)
```

Self.drop = dropout(0.2)

emb

LSTM

hidden

```
>>> output.data.shape  
(3L, 2L, 200L)
```

output

Self.drop = dropout(0.2)

view

```
[torch.FloatTensor of size 6x200]
```

Decoder = linear(200,ntokens)

decoded

```
>>> decoded.data.shape  
(6L, 37L)
```

