Practice with RNNs

Machine Intelligence Lab
Handong Global University

Practice – language modeling

"we 're talking about years ago before anyone heard of asbestos having any questionable pr operties"

"the total of N deaths from malignant <unk> lung cancer and <unk> was far higher than ex pected the researchers said"

"about N workers at a factory that made pape r for the kent filters were exposed to asbestos in the 1950s"

PennTreeBank (PTB) dataset

N sentences generated from Humans
N for trainset
N for validset

what is language modeling?

Estimating the probability of the word given the sequence.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots w_n) = \prod_{i=1}^{n} P(w_n | w_1, \ldots, w_{n-1})$$

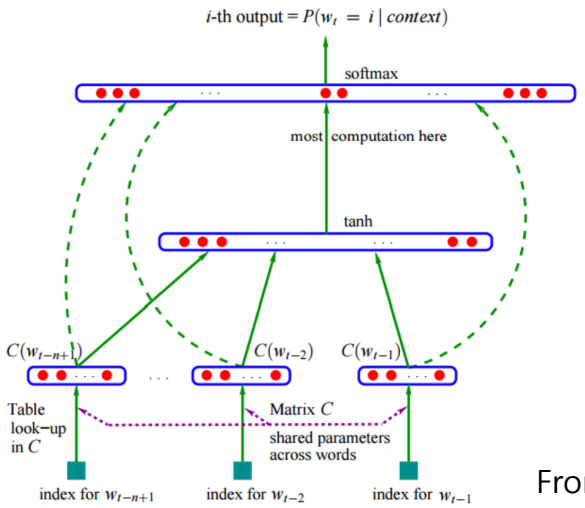
(1) $P(Today\ is\ Wednesday) = 0.001$

(2) $P(Today\ Wednesday\ is) = 0.0000000001$

From ratsgo's blog

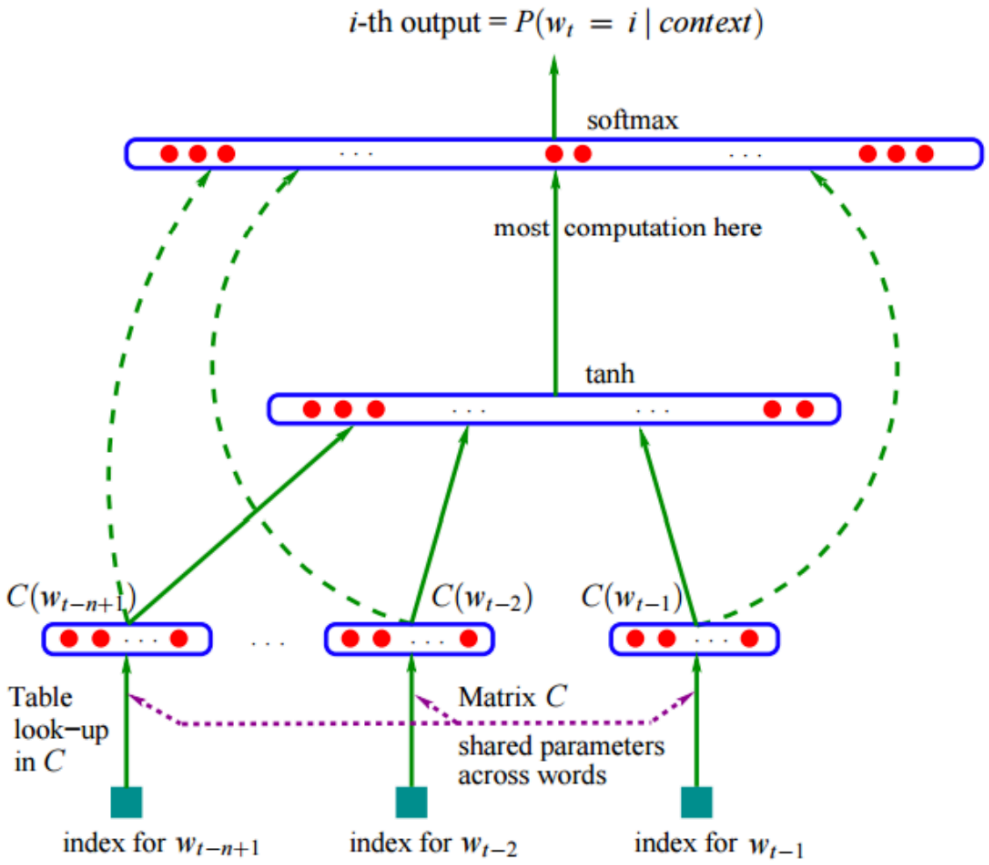
Modeling this conditional probability with neural network..!

$P(w_n | w_1, \ldots, w_{n-1})$



From Bengio et al. 2003

what is language modeling?



From Bengio et al. 2003

Probaility of each word

Ouput score

Hidden state

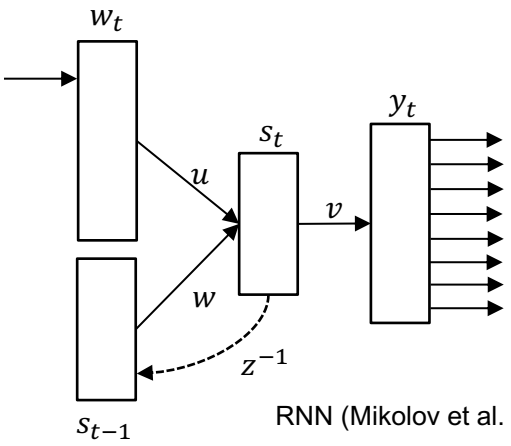
Word Embedding

Word Sequence

word embedding

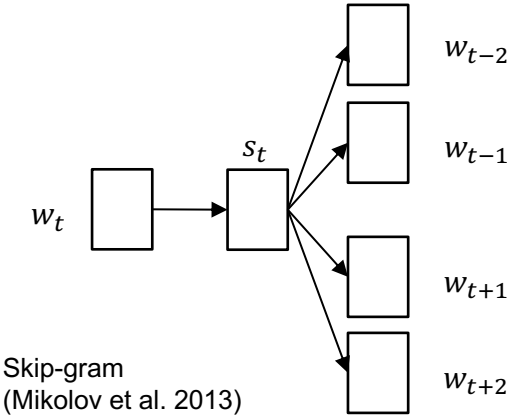
- language is discrete, symbolic, and categorical
- word embedding (semantic)
  - U (or s) is the representation. (distributed representation)

language model



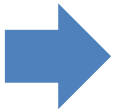
RNN (Mikolov et al. 2013)

only for word embedding



Skip-gram
(Mikolov et al. 2013)

w(t): one hot representation
car = [0, 1, 0]
vehicle = [0, 0, 1]
flower = [1, 0, 0]

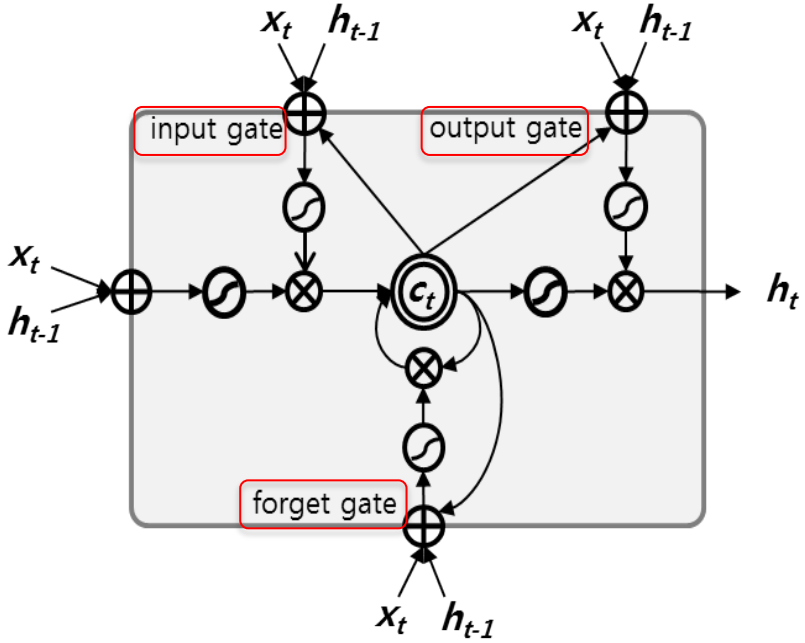


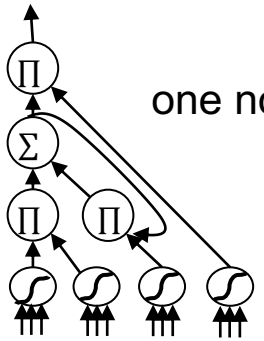
s(t): word embedding
car = [0.17, 0.83]
vehicle = [0.23, 0.77]
flower = [0.81, 0.19]

long short-term memory (LSTM)

- LSTM works successfully with sequential data.
  - hand writing, speech, etc
- LSTM can model very long term sequential patterns.
  - longer memory has a stabilizing effect.

- information is saved in the cell
- gates control information flow.





one node itself is a deep network.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

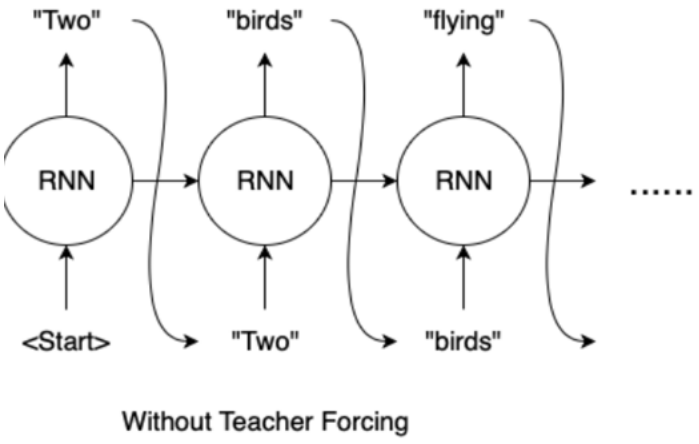
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$

$$h_t = o_t \tanh(c_t)$$

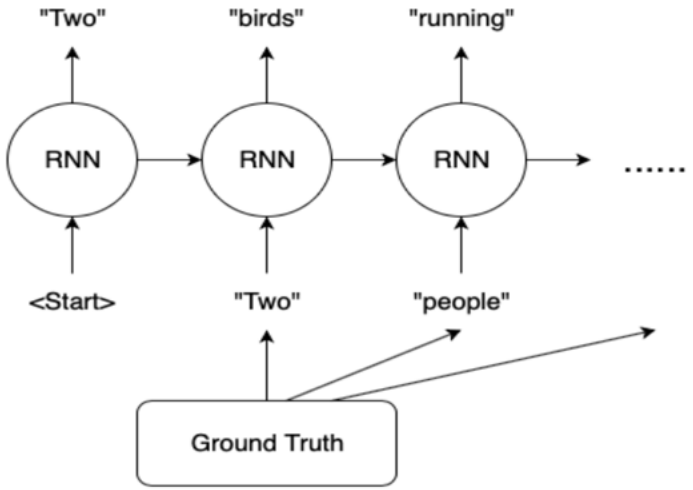
many to many RNN cell operations

Inferencing



Without Teacher Forcing

Training



With Teacher Forcing

From Sooftware's blog

mount, libraries, and util functions

```
### Import the libraries

import os
import time
import math
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable

import six; from six.moves import cPickle
import numpy as np

print("Importing libraries done!")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
BOS_token = 0 # Beginning Of Sentence token
EOS_token = 1 # End Of Sentence token
UNK_token = 2 # UNKnown token

### Define the util functions
def ids2words(dict_map, raw_data, sep=' ', eos_id=0, unk_sym='<unk>'):
    str_text = ''
    raw_data = raw_data.squeeze().tolist()

    # Make the dict to inverse for translate unique number to word
    dict_map_inv = dict()
    for kk, vv in dict_map.items():
        dict_map_inv[vv] = kk

    for vv in raw_data:
        if vv == eos_id:
            break
        if vv in dict_map_inv:
            str_text = str_text + sep + dict_map_inv[vv]
        else:
            str_text = str_text + sep + unk_sym
    return str_text.strip()
```

```
def timeSince(since):
  now = time.time()
  s = now - since

  h = math.floor(s / 3600)
  m = math.floor((s-3600*h) / 60)
  s = s - h*3600 - m*60

  return '{}h {}m {:.3f}s'.format(h,m,s)
```

model

```
### Make my Language Model
class LM(nn.Module):
    def __init__(self, dict_len, dim_enc, dim_wemb, device):
        super(LM, self).__init__()
        self.dim_enc = dim_enc
        self.wemb = dim_wemb
        self.dict_len = dict_len
        self.device = device

        self.dropout = nn.Dropout(0.2)
        self.src_emb = nn.Embedding(dict_len, dim_wemb)
        self.rnn_enc = nn.LSTMCell(dim_wemb, dim_enc)

        self.readout = nn.Linear(dim_enc, dim_wemb)
        self.dec = nn.Linear(dim_wemb, dict_len)

        self.criterion = nn.CrossEntropyLoss()
        self.softmax = nn.Softmax(dim=1)
```

Word embedding layer (word to emb.)
LSTM layer

Fully connected layer (hidden state to emb.)
Fully connected layer (emb. to word)

CrossEntropy Loss function

```
def forward(self, data, mask=None):
    # data : (Timeseq, Batch)
    x_data = data[:-1] # input (the last word is not the input)
    y_data = data[1:]  # label (the first word is not the label)
    if mask is not None:
        x_mask = mask[1:]
        y_mask = mask[1:]

    Tx, Bn = x_data.size()

    x_emb = self.src_emb(torch.reshape(x_data, (Bn*Tx,1)))
    x_emb = x_emb.view(Tx,Bn,-1)
    x_emb = self.dropout(x_emb)
    # x_emb : (Timeseq, Batch, dim_wemb)
```

'forward' function is model's forward propagation

Input sequence become the label

Words to word embedding operation

model

```
    ht = torch.zeros(Bn,self.dim_enc)
    ct = torch.zeros(Bn,self.dim_enc)
    ht = Variable(ht).to(self.device)
    ct = Variable(ct).to(self.device)

    gen_sentence = x_data[0].unsqueeze(1) # (Batch, 1)
    loss = 0
    for i in range(Tx):
        ht, ct = self.rnn_enc(x_emb[i,:,:],(ht, ct))
        # ht, ct : (Batch, dim_enc)
        output = self.readout(ht)
        output = self.dropout(output)
        # output : (Batch, dim_wemb)
        logit = self.dec(output)
        # logit : (Batch, dict_len)
        loss_tmp = self.criterion(logit, y_data[i])

        probs = self.softmax(logit)
        topv, yt = probs.topk(1) # Choose top 1 prob. word

        gen_sentence = torch.cat((gen_sentence, yt), dim=1)
        if mask is not None:
            loss += torch.sum(loss_tmp*y_mask[i])/Bn
        else:
            loss += torch.sum(loss_tmp)/Bn

    return loss, gen_sentence
```

LSTM states initialization

LSTM operation

Hidden state to word embedding
Dropout operation

Word embedding to words

Compute loss

Choose the best scored word

Return the loss and generated sentence

functions: train and test

```
[ ] def train(model, device, train_loader, optimizer, epoch, log_interval):
        model.train()

        loss_total = 0
        den = 0
        for batch_idx, (data, mask) in enumerate(train_loader):
            data, mask = torch.transpose(data,1,0).to(device), torch.transpose(mask,1,0).to(device)
            optimizer.zero_grad()
            loss, gen_sentence = model(data, mask)
            loss.backward()
            optimizer.step()

        print('Train Epoch: {} \tLoss: {:.6f}'.format(
                    epoch, loss.item()))
        real_sen = ids2words(src_dict, data[:,0], eos_id=EOS_token)
        gen_sen = ids2words(src_dict, gen_sentence[0], eos_id=EOS_token)
        print("train real sentence: {}".format(real_sen))
        print("train gen. sentence: {}".format(gen_sen))
        print("========================================")
        return loss.item(), gen_sentence
```

Backpropagation and optimize

Print the real and generated sentence

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for batch_idx, (data, mask) in enumerate(test_loader):
            data, mask = torch.transpose(data,1,0).to(device), torch.transpose(mask,1,0).to(device)
            data, mask = data.to(device), mask.to(device)
            loss, gen_sentence = model(data)
            test_loss += loss

    test_loss /= batch_idx

    print('\nTest: Average loss: {:.4f}\n'.format(
        test_loss))
    return test_loss
```

define custom dataset

```
### Make my custom Dataset and DatasetLoader classes
class ptb_dataset(Dataset):
    def __init__(self,train_data,data_dict,maxlen=30):
        # Load the dataset and word_dict
        self.train_data_raw = open(train_data, 'r')
        with open(data_dict, 'rb') as f:
            self.data_dict = pkl.load(f)

        # Make dict has unique index
        self.data_dict2 = dict()
        for kk, vv in self.data_dict.items():
            self.data_dict2[kk] = vv + 1
        self.data_dict2['<s>'] = BOS_token

        self.maxlen = maxlen

        # Pre-processing the datasets
        self.train_data, self.train_len = self.data_init(self.train_data_raw)

    def __getitem__(self, index):
        sentence = self.train_data[index,:self.train_len[index]]
        x_data, x_mask = self.prepare_text(sentence)
        return torch.tensor(x_data).type(torch.long),\
               torch.tensor(x_mask).type(torch.float)

    def __len__(self):
        return len(self.train_data)

    def dict_len(self):
        return len(self.data_dict2)

    def use_dict(self):
        return self.data_dict2
```

Open raw dataset

Open the word token file (pickle)

Make the word dictionary

data_init : preprocessing dataset (next slide)

'__getitem__' : return one data sample

Prepare_text : preprocessing data sample (next_slide)

```
def data_init(self, data):
    #Check the number of sample < maxlen
    num = 0
    while True:
        sentence = data.readline()
        if sentence == "":
            break
        if len(sentence.strip().split()) >= self.maxlen:
            continue
        else:
            num += 1
    # Make the preprocessed dataset
    dataset = np.zeros((num, self.maxlen))
    data_len = np.zeros(num, dtype=np.int)
    idx = 0
    data.seek(0)
    while True:
        sentence = data.readline()
        if sentence == "": # End of the dataset
            break
        # Make sentence to word level (splitted by space)
        sentence = sentence.strip().split()
        if len(sentence) >= self.maxlen:
            continue
        else:
            sentence = [self.data_dict2.get(key, UNK_token)\
                                    for key in sentence]
            dataset[idx,:len(sentence)] = sentence
            data_len[idx] = len(sentence)
            idx += 1
    return dataset, data_len

def prepare_text(self, sentence):
    maxlen = self.maxlen + 2 # +2 for BOS and EOS
    x_data = np.ones(maxlen).astype('int64')
    x_mask = np.zeros(maxlen).astype('float32')
    x_data[1:len(sentence)+1] = sentence
    x_data[0] = BOS_token
    x_mask[:len(sentence)+2] = 1. # EOS token

    return x_data, x_mask
```

Read one line of the opened dataset

Check the total number of samples that are not over the maxlen

Initialize the dataset memory

Read one line of the opened dataset

Splitting the sentence

Exclude long sentences

Tokenizing

Store pre-processed samples

Sentence + BOS, EOS tokens

Mask : 1 to real sentence tokens, 0 to padding

define custom dataset

```
def ptb_loader(train_data, batch_size, maxlen):
    data_dict = 'drive/My Drive/public/data/ptb/ptb.train.txt.pkl'

    data = ptb_dataset(train_data, data_dict, maxlen=maxlen)
    src_dict = data.use_dict()

    data_loader = DataLoader(data, batch_size=batch_size, shuffle=False)

    return data_loader, data.dict_len(), src_dict
```

Call previous custom dataset

Dataloader : make a data loader with the custom dataset

Make the word dictionary

```
### Test my dataset, datasetloader classes
batch_size = 1
maxlen = 30

train_data = 'drive/My Drive/public/data/ptb/ptb.train.txt'
train_loader, dict_len, src_dict = ptb_loader(train_data, batch_size, maxlen)

for i, (x_data, x_mask) in enumerate(train_loader):
    real_sen = ids2words(src_dict, x_data, eos_id=EOS_token)
    print("------------------------------------------------------------")
    print("real sentence: ")
    print(real_sen)
    print("x_data.shape: ", x_data.shape)
    print("x_data:")
    print(x_data)
    print("x_mask:")
    print(x_mask)
    print("------------------------------------------------------------")

    if i >= 5:
        break
```

define hyperparameters, dataset, model, and optimizer

```
### Training
train_data = 'drive/My Drive/public/data/ptb/ptb.train.txt'
test_data = 'drive/My Drive/public/data/ptb/ptb.valid.txt'

# Check the device
cuda = torch.cuda.is_available()
device = torch.device("cuda" if cuda else "cpu")

# build my dataset loader
train_loader, dict_len, src_dict = ptb_loader(train_data, batch_size, maxlen)
test_loader, _, _ = ptb_loader(test_data, test_batch_size, maxlen)

# build my model
model = LM(dict_len, dim_enc, dim_emb, device)
model.to(device)

# build the optimizer
if optimizer == 'RMSprop':
    opt = optim.RMSprop(model.parameters(), lr=lr)
elif optimizer == 'Adam':
    opt = optim.Adam(model.parameters(), lr=lr)
elif optimizer == 'Adadelta':
    opt = optim.Adadelta(model.parameters(), lr=lr)
else:
    opt = optim.SGD(model.parameters(), lr=lr)
```

```
### Hyperparameters
batch_size = 64
test_batch_size=1
maxlen = 30
dim_enc = 400
dim_emb = 300
lr = 0.0001
optimizer = 'Adam'
max_epoch = 100
```

running the model

```
# Training..
print("Training Start!")
best_loss = 99999
for epoch in range(max_epoch):
    train(model, device, train_loader, opt, epoch, log_interval)
    test_loss = test(model, device, test_loader)

    if best_loss > test_loss:
        print("We found the best model!")
        best_loss = test_loss
        save_dir = 'drive/My Drive/public/results/ptb_trained_model_best.pth'
        if os.path.exists(save_dir):
            os.remove(save_dir)
        torch.save(model, save_dir)

print("Training is done!!")
```

running the model

```
Training Start!
Train Epoch: 0  Loss: 87.210655
train real sentence: <s> the u.s. trade deficit swelled to $ N billion in august prompting worries that the nation 's export drive had stalled
train gen. sentence: <s> the <unk> <unk> <unk>
========================================

Test: Average loss: 115.9713
```

0 epoch : not trained, poor performance

```
We found the best model!
Train Epoch: 10         Loss: 68.344185
train real sentence: <s> the u.s. trade deficit swelled to $ N billion in august prompting worries that the nation 's export drive had stalled
train gen. sentence: <s> the <unk> is was was to $ N million from the
========================================

Test: Average loss: 98.4239
```

10 epochs : decreased train loss, decreased test loss

```
We found the best model!
Train Epoch: 34         Loss: 55.430367
train real sentence: <s> the u.s. trade deficit swelled to $ N billion in august prompting worries that the nation 's export drive had stalled
train gen. sentence: <s> the company has deficit was to the N billion from august from the the the <unk> 's largest earthquake is been the
========================================

Test: Average loss: 93.7123
```

34 epochs : fit well to trainset, best performance for testset – best model

```
Train Epoch: 98         Loss: 41.463730
train real sentence: <s> the u.s. trade deficit swelled to $ N billion in august prompting worries that the nation 's export drive had stalled
train gen. sentence: <s> the <unk> is deficit swelled to $ N billion in the prompting the that the dollar 's export governments is been
========================================

Test: Average loss: 98.8233
```

98 epochs : fit almost perfectly to trainset, decreased performance for testset
- overfitting

Machine Intelligence Lab
https://milab.handong.edu/

Handong Global University