Assignment_4_Code_906213559

1 ECE-6524 / CS-6524 Deep Learning

2 Assignment 4 [100 pts]

In this assignment, we will explore Recurrent Neural Networks (RNN) to deal with the data with temporal sequence. Specifically, we will generate captions for images. We will design an encoder-decoder architecture to achieve this. This homework is inspired by Stanford CS231n and UCSD CSE253.

2.1 Submission guideline for the coding part (Jupyter Notebook)

1. Click the Save button at the top of the Jupyter Notebook
2. Please make sure to have entered your Virginia Tech PID below
3. Once you've completed everything (make sure output for all cells are visible), select File -> Download as -> PDF via LaTeX
4. Look at the PDF file and make sure all your solutions are displayed correctly there
5. Zip all the files along with this notebook (Please don't include the data). Name it as Assignment_3_Code_[YOUR PID NUMBER].zip
6. Name your PDF file as Assignment_4_NB_[YOUR PID NUMBER].pdf
7. **Submit your zipped file and the PDF SEPARATELY**

Note: if facing issues with step 3 refer: https://pypi.org/project/notebook-as-pdf/

2.2 Submission guideline for the coding part (Google Colab)

1. Click the Save button at the top of the Notebook
2. Please make sure to have entered your Virginia Tech PID below
3. Follow last two cells in this notebook for guidelines to download pdf file of this notebook
4. Look at the PDF file and make sure all your solutions are displayed correctly there
5. Zip all the files along with this notebook (Please don't include the data). Name it as Assignment_2_Code_[YOUR PID NUMBER].zip
6. Name your PDF file as Assignment_4_NB_[YOUR PID NUMBER].pdf
7. **Submit your zipped file and the PDF SEPARATELY**

While you are encouraged to discuss with your peers, all work submitted is expected to be your own. If you use any information from other resources (e.g. online materials), you are required to cite it below you VT PID. Any violation will result in a 0 mark for the assignment.

2.2.1 Please Write Your VT PID Here: 906213559

2.2.2 Reference (if any):

https://towardsdatascience.com/automatic-image-captioning-with-cnn-rnn-aae3cd442d83

https://medium.com/@stepanulyanin/captioning-images-with-pytorch-bc592e5fd1a3

https://towardsdatascience.com/pre-trained-word-embeddings-or-embedding-layer-a-dilemma-8406959fd76c

In this homework, you would need to use **Python 3.6+** along with the following packages (**need to update**):

```
1. pytorch 1.2
2. torchvision
3. numpy
4. matplotlib
5. nltk
```

To install pytorch, please follow the instructions on the Official website. In addition, the official document could be very helpful when you want to find certain functionalities.

[1]:
```
# import necessary packages and modules
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchvision import transforms
from torch.nn.utils.rnn import pack_padded_sequence
from torch.autograd import Variable
import numpy as np
import matplotlib.pyplot as plt
import os
```

3 Image Captioning Using Encoder-Decoder Architecture

Simply, the encoder will take the image as input and encode it into a vector of feature values. The decoder will take this output from encoder as hidden state and starts to predict next words at each step. The following figure illustrates this:

Figure 1. An overview of the encoder-decoder architecture (image credit: Deep Neural Network Based Image Captioning)

You will use a pre-trained CNN as the encoder and Vanilla RNN/LSTM as decoder to predict the captions.

4 Section 1.1 Data

Download dataset following the instructions. We are gonna use Flickr30k dataset, which consists of 31783 images and 158,915 captions. However, instead of using the whole dataset and “results.csv”, we will follow Karpathy’s Flickr30k annotations and use “dataset_flickr30k.json”. Please move “dataset_flickr30k.json” to “flickr30k_images” folder.

4.1 How to download the data (ARC)

Since the original Flickr30K dataset requires application, we will use the dataset from Kaggle.

Step 1: Register a Kaggle account. https://www.kaggle.com/

Step 2: Log into ARC server in the terminal, e.g. Huckleberry.

Step 3: Install required packages.

```
- if you're gonna use powerai on Huckleberry
    # step 1: request for GPU nodes
    salloc --partition=normal_q --nodes=1 --tasks-per-node=10 --gres=gpu:1 bash
    # or if you don't want a GPU, it will be faster to get a job
    salloc --partition=normal_q --nodes=1 --tasks-per-node=10 bash
    # step 2: load all necessary modules
    module load gcc cuda Anaconda3 jdk
    # step 3: activate the virtual environment
    source activate powerai16_ibm
    # step 4: for new packages(take tqdm for example)
    pip install --user kaggle nltk # on hulogin1/hulogin2

- if you're gonna use your own conda environment, simply type
    pip install kaggle nltk
```

Step 4: Make sure you have kaggle. Type `kaggle`.

Step 5: Download your kaggle.json file from https://www.kaggle.com/Your_Username/account. In API section, click Create New API Token. Then move you kaggle.json file to the path `/home/your_name_space/.kaggle/kaggle.json`.

Step 6: Download the dataset. Type `kaggle datasets download hsankesara/flickr-image-dataset`.

Step 7: Unzip the dataset. Type `unzip flickr-image-dataset.zip -x "flickr30k_images/flickr30k_images/flickr30k_images/*.jpg" -d "/path-to-Assignment_4/Assignment_4/"`

Step 8: You should have your dataset in `/path-to-Assignment_4/Assignment_4/flickr30k_images/`

Step 9: Move “dataset_flickr30k.json” to “flickr30k_images” folder.

Note that you might want to use `nltk.download('punkt')` and `torchvision.models.resnet50(pretrained=True)` before you enter the GPU node.

4.2 How to download the data (Google Colab)

Step 1: Register a Kaggle account. https://www.kaggle.com/

Step 2: Download your kaggle.json file from https://www.kaggle.com/Your_Username/account. In API section, click Create New API Token.

Step 3: As we did before, upload all files on Google Drive and open Google Colab.

Step 4: Install required packages.

```
! pip install -q kaggle nltk
```

Step 5: Insert a cell.

```
from google.colab import files
files.upload()


Upload `kaggle.json` you just downloaded.
```

Step 6: Move `kaggle.json` to the right place,

```
 ! mkdir ~/.kaggle
 ! cp kaggle.json ~/.kaggle/
```

Step 7: Change the permission.

```
! chmod 600 ~/.kaggle/kaggle.json
```

Step 8: Download.

```
!kaggle datasets download hsankesara/flickr-image-dataset
```

Step 9: Move it to your drive and unzip it.

```
unzip flickr-image-dataset.zip -x "flickr30k_images/flickr30k_images/flickr30k_images/*.jpg" -d
```

Step 10: Move "dataset_flickr30k.json" to "flickr30k_images" folder.

```
[ ]: ! pip install -q kaggle nltk
from google.colab import files
files.upload()

! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download hsankesara/flickr-image-dataset
```

```
<IPython.core.display.HTML object>


Saving kaggle.json to kaggle.json
Downloading flickr-image-dataset.zip to /content
100% 8.16G/8.16G [03:08<00:00, 33.7MB/s]
100% 8.16G/8.16G [03:08<00:00, 46.5MB/s]
```

4.2.1 Colab Setup:

- Below are some basic steps for colab setup.
- Make changes based on requirements.
- Comment out in case of ARC or your local device with powerful GPU.

Note: For Google Colab give proper paths in this notebook and in dataloader.py if required.

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
! unzip flickr-image-dataset.zip -x "flickr30k_images/flickr30k_images/
↪flickr30k_images/*.jpg" -d "/content/drive/My Drive/DL_Fall_2020/Assignment_4/"
```

[3]:
```
import sys
# modify "customized_path_to_homework", path of folder in drive, where you␣
↪uploaded your homework
path_to_homework = "/content/drive/My Drive/DL_Fall_2020/Assignment_4/"
sys.path.append(path_to_homework)
```

[4]:
```
from dataloader import Flickr30k, get_loader
import numpy as np
import os
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms
import torch
import torchvision
import matplotlib.pyplot as plt
```

5 Section 1.2 Take a look at the data

[5]:
```
# visualize images and captions
flickr = Flickr30k(split='val', root=path_to_homework+'flickr30k_images/')  #␣
↪load validation set as an example
flickr()
```

```
-------flickr30k--------
image root: /content/drive/My
Drive/DL_Fall_2020/Assignment_4/flickr30k_images/flickr30k_images
dataset split: val
the length of the dataset: 1014
```

[ ]:
```
# show a random image and its captions
img_id = np.random.randint(len(flickr))
img = flickr.get_img(img_id)
captions = flickr.get_captions(img_id)

plt.figure()
plt.imshow(img)
plt.axis('off')
```

```
print(captions)
```

['The boy in the gray shirt is holding the boy in the red shirt on his back.', 'Two boys are back to back as one holds up the other.', 'One child lifts another on his back, inside a room.', 'One boy hoists another boy up on his back.', 'Boy pulls other boy over back.']

[6]:
```
del flickr
```

6 Section 1.3 Build vocabulary

We need to build a vocabulary for our dataset. The vocabulary stores all the words and their indices. We will use it to embed and recover the words.

[7]:
```
import nltk
import pickle
import json
from tqdm import tqdm
from collections import Counter
nltk.download('punkt') # You can comment this line once you've downloaded 'punkt'

class Vocabulary(object):
    """Simple vocabulary wrapper."""
    def __init__(self):
```

```
        self.word2idx = {'<pad>': 0, '<unk>': 1, '<start>': 2, '<end>': 3}  #␣
↪follow Pytorch padding rules: pad sentence with zero.
        self.idx = 4
        self.idx2word = {v: k for k, v in self.word2idx.items()}

    def __call__(self, key):
        if key not in self.word2idx:
            return self.word2idx['<unk>']
        return self.word2idx[key]

    def __len__(self):
        return len(self.word2idx)

    def add_word(self, word):
        """
        Add new words
        :param word: word
        """
        if word not in self.word2idx:
            self.word2idx[word] = self.idx  # add a new word
            self.idx2word[self.idx] = word
            self.idx += 1

    def reverse(self, value):
        """
        From idx to words.
        :param value: index
        :return:
        """
        if value not in self.idx2word:
            return self.idx2word[1]  # return '<unk>' if the word is unseen␣
↪before.
        return self.idx2word[value]

def build_vocab(json_file=path_to_homework+ '/flickr30k_images/dataset_flickr30k.
↪json', threshold=3):
    with open(json_file) as f:
            data = json.load(f)
    f.close()
    counter = Counter()
    for img_idx in tqdm(range(len(data['images']))):
        img_annos = data['images'][img_idx]
        for sent_idx in range(len(img_annos['sentids'])):
#             tokens = img_annos['sentences'][sent_idx]['tokens']  # directly␣
↪load tokens

            caption = img_annos['sentences'][sent_idx]['raw']
```

```
            tokens = nltk.tokenize.word_tokenize(caption.lower())

            counter.update(tokens)

    # If the number of words is less than threshold we don't count it.
    words = [word for word, cnt in counter.items() if cnt >= threshold]


    # create a Vocabulary class
    vocab = Vocabulary()

    # add words to Vocab
    for i, word in enumerate(words):
        vocab.add_word(word)

    return vocab
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

[8]:

```
# let's create a vocabulary for future usage
vocab_path = path_to_homework + '/flickr30k_images/vocab.pkl'
if not os.path.isfile(vocab_path):  # if we don't have vocab, create one
    vocab = build_vocab(json_file=path_to_homework + '/flickr30k_images/
 ↪dataset_flickr30k.json', threshold=3)
    with open(vocab_path, 'wb') as f:
        pickle.dump(vocab, f)
    print("Total vocabulary size: {}".format(len(vocab)))
    print("Saved the vocabulary wrapper to '{}'".format(vocab_path))
else:  # if we have, load the existing vocab
    with open(vocab_path, 'rb') as f:
        vocab = pickle.load(f)
    print('vocab loaded!')
    print('the size of vocab:', len(vocab))
f.close()
```

```
vocab loaded!
the size of vocab: 9991
```

[9]:

```
vocab_path = path_to_homework + '/flickr30k_images/vocab.pkl'
with open(vocab_path, 'rb') as f:
    vocab = pickle.load(f)
print('vocab loaded!')
print('the size of vocab:', len(vocab))
# print(vocab.word2idx.keys())
# print(vocab.idx2word)
```

```
# check some random words
for i in range(3):
    random_idx = np.random.randint(len(vocab))
    print('word: {}, index: {}'.format(list(vocab.word2idx.keys())[random_idx],␣
 ↪vocab(list(vocab.word2idx.keys())[random_idx])))
```

```
vocab loaded!
the size of vocab: 9991
word: enrolling, index: 8889
word: fighters, index: 3907
word: altima, index: 9866
```

7 Section 2 Vanilla RNN [45 pts]

8 Section 2.1 Design the Network: Encoder [5 pts]

Implement the baseline model by using pre-trained ResNet-50 as the encoder and Vanilla RNN as the decoder. Note that we will remove the last layer (fc layer) of ResNet-50 and add a trainable linear layer to finetune it for our task. During the training, we will **freeze** the layer before the fc layer. The encoder should output a feature vector of a fixed size for each image.

[10]:
```
class Encoder(nn.Module):
    def __init__(self, emb_dim):
        """
        Use ResNet-50 as encoder.
        :param emb_dim: output size of ResNet-50.
        """
        super(Encoder, self).__init__()
        self.resnet = torchvision.models.resnet50(pretrained=True)
        ##########Your code###############
        # freeze the parameters
        for param in self.resnet.parameters():
            param.requires_grad = False
        # replace the last layer (fc layer) with a trainable layer for finetuning
        self.resnet.fc = nn.Linear(resnet.fc.in_features, emb_dim)

    def forward(self, x):
        x = self.resnet(x)  # output shape: [N, emb_dim]
        return x
```

9 Section 2.2 Design the Network: Decoder [10 pts]

During decoding, we will train a RNN (https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.r to learn the structure of the caption text throught "**Teacher Forcing**". Teacher forcing works by using the teaching signal from the training dataset at the current time step, $target(t)$, as input in the next time step $x(t+1) = target(t)$, rather than the output $y(t)$ generated by the network.

As shown in Figure 1 above, RNN will take three inputs: the *current feature*, hidden state (h_0) and cell state (c_0). The *current feature* for the first step should be the output of encoder to predict '<start>' word. Hidden states for this step should be set to None. Then in the second step '<start>' will be passed into RNN as the input, and so on.

To use '<start>' or any subsequent word as current feature, get its index from the vocabulary you created, convert it to one-hot vector and pass it through a linear layer to embed into a feature (or you can take advantage of Pytorch's nn.Embedding which does one-hot encoding + linear layer for you).

For convenience, you might want to 'pad' the captions in a mini-batch to convert them into fixed length. You can use 'pack_padded_sequence' function.

[11]:
```
class Decoder(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, num_layers=1, dropout=0):
        """
        Use RNN as decoder for captions.
        :param emb_dim: Embedding dimensions.
        :param hidden_dim: Hidden states dimensions.
        :param num_layers: Number of RNN layers.
        :param vocab_size: The size of Vocabulary.
        :param dropout: the probability for dropout.
        """
        super(Decoder, self).__init__()
        self.max_length = 30  # the maximum length of a sentence, in case it's␣
↪trapped
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size

        self.temp = 1
        ############Your code############
        # you need to implement a Vanilla RNN for the decoder. Take a look at␣
↪the official documentation.
        # https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.
↪RNN

        # one-hot encoding + linear layer
        self.embed = nn.Embedding(self.vocab_size, emb_dim)
        # vanilla rnn network
        self.rnn = nn.RNN(emb_dim, hidden_dim, num_layers)
        # output layer
        self.out = nn.Linear(hidden_dim, vocab_size)

    def forward(self, encode_features, captions, lengths):
        """
        Feed forward to generate captions. Note that you need to pad the input␣
↪so they have the same length
        :param encode_features: output of encoder, size [N, emb_dim]
```

```
        :param captions: captions, size [N, max(lengths)]
        :param lengths: a list indicating valid length for each caption. size is␣
↪(batch_size).
        """
        ############Your Code##################
        # compute the embedding using one-hot technique and linear function
        caption_embeded = self.embed(captions)

        # concatenate the encoded features from encoder and embeddings
        encode_features = torch.cat((encode_features.unsqueeze(1),␣
↪caption_embeded), dim = 1)

        # feed into RNN.
        encode_features_pack = pack_padded_sequence(encode_features, lengths,␣
↪batch_first=True)
        output, hidden = self.rnn(encode_features_pack)

        # output layer
        outputs = self.out(output[0])

        return outputs
```

10 Encoder-decoder [10 pts]

Now we need to put our encoder and decoder together.

In the sample_generate stage, the idea is to "let the network run on its own", predicting the next word, and then use the network's prediction to obtain the next input word. There are at least two ways to obtain the next word.

- **Deterministic**: Take the maximum output at each step.
- **Stochastic**: Sample from the probability distribution. To get the distribution, we need to compute the weighted softmax of the outputs: $y^i = \exp(o^j/\tau)/\sum_n \exp(o^n/\tau)$, where o^j is the output from the last layer, n is the size of the vocabulary, and τ is the so-called "temperature". By doing this, you should get a different caption each time.

[12]:
```
class Vanilla_rnn(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, num_layers=1, dropout=0):
        """
        Encoder-decoder vanilla RNN.
        :param vocab_size: the size of Vocabulary.
        :param emb_dim: the dimensions of word embedding.
        :param hidden_dim: the dimensions of hidden units.
        :param num_layers: the number of RNN layers.
        :param dropout: dropout probability
        """
        super(Vanilla_rnn, self).__init__()
```

```
        #########Your Code################
        # Encoder: ResNet-50
        self.Vanilla_encoder = Encoder(emb_dim)
        # Decoder: RNN
        self.Vanilla_decoder = Decoder(vocab_size, emb_dim, hidden_dim,␣
↪num_layers, dropout)
        self.max_length = self.Vanilla_decoder.max_length
        self.temp = 1
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x, captions, lengths):
        """
        Feed forward.
        :param x: Images, [N, 3, H, W]
        :param captions: encoded captions, [N, max(lengths)]
        :param lengths: a list indicating valid length for each caption. length␣
↪is (batch_size).
        :return: output logits, usually followed by a softmax layer.
        """
        #########Your code##################
        # forward passing
        encoder_features = self.Vanilla_encoder(x)
        x = self.Vanilla_decoder(encoder_features, captions, lengths)

        return x

    def sample_generate(self, x, states=None, mode='Deterministic',␣
↪temperature=5.0):
        """
        Generate samples during the evaluation.

        :param x: input image
        :param states: rnn states
        :param mode: which mode we use.
         - 'Deterministic': Take the maximum output at each step.
         - 'Stochastic': Sample from the probability distribution from the␣
↪output layer.
        :param temperature: will be used in the stochastic mode
        :return: sample_idxs. Word indices. We can use vocab to recover the␣
↪sentence later.
        """
        sample_idxs = []  # record the index of your generated words

        ################Your Code#################
        # compute the encoded features
        in_feature = self.Vanilla_encoder.forward(x)
```

```
        in_feature = in_feature.unsqueeze(1)
        batch_size = in_feature.shape[0]

        for i in range(self.max_length):
          outputs, states = self.Vanilla_decoder.rnn(in_feature, states)
          outputs = outputs.squeeze(1)
          outputs = self.Vanilla_decoder.out(outputs)
          outputs = self.softmax(outputs/temperature)

        # decide which mode we use
          if mode == 'Deterministic':
              # take the maximum index after the softmax
              max_val, predicted = outputs.max(1)
              sample_idxs.append(predicted)

          elif mode == 'Stochastic':
              # sample from the probability distribution after the softmax
              # Hint: use torch.multinomial() to sample from a distribution.
              predicted = torch.multinomial(outputs, 1)
              sample_idxs.append(predicted)

          x = self.Vanilla_decoder.embed(predicted)
          x = x.unsqueeze(1)
        sample_idxs = torch.stack(sample_idxs, 1)
        sample_idxs = sample_idxs.squeeze()

        return sample_idxs
```

11 Section 2.3 Training [10 pts]

Train your encoder-decoder. You might also want to check the output sentence every epoch.

[13]:
```
# some hyperparameters, you can change them
## training parameters
batch_size = 256
lr = 1e-2
num_epochs = 50
weight_decay = 0.0
log_step = 50

## network architecture
emb_dim = 1024
hidden_dim = 256
num_layers = 1 # number of RNN layers
dropout = 0.0
```

```python
## image transformation
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #     transforms.RandomCrop(224, pad_if_needed=True),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))])

## Output directory
output_dir = path_to_homework + '/checkpoints/rnn/'
os.makedirs(output_dir, exist_ok=True)

## device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[14]:

```python
# Validation code here. We are gonna use this during the training.
def val(model, data_loader, vocab):
    """
    Inputs:
    :param model: the encoder-decoder network.
    :param data_loader: validation data loader
    :param vocab: pre-built vocabulary
    Output:
    the mean value of validation losses
    """
    print('Validating...')
    val_loss = []
    total_step = len(data_loader)
    criterion = nn.CrossEntropyLoss()
    for itr, (images, captions, lengths) in enumerate(data_loader):
        ######Your Code#########
        # forward inputs and compute the validation loss
        captions = captions.to(device)
        outputs = model(images.to(device), captions, lengths)
        outputs_pack = pack_padded_sequence(captions, lengths,␣
↪batch_first=True)[0]
        outputs = outputs.view(-1, len(vocab))
        loss = criterion(outputs, outputs_pack)

        # record the validation loss
        val_loss.append(float(loss))

        # Print current loss
        if itr % log_step == 0:
            print('Step [{}/{}], Loss: {:.4f}, Perplexity: {:5.4f}'
```

```
                  .format(itr, total_step, loss.item(), np.exp(loss.item())))
    val_loss = np.array(val_loss)

    # (optional) you might also want to print out the sentence to see the␣
 ↪qualitative performance of your model.
    # You can use deterministic mode to generate sentences


    return np.mean(val_loss)
```

[ ]:
```
# Training code here

train_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='train', vocab=vocab,
                               transform=transform, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=4)
val_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='val', vocab=vocab,
                               transform=transform, batch_size=8, shuffle=True,␣
 ↪num_workers=4)

model = Vanilla_rnn(vocab_size=len(vocab), emb_dim=emb_dim,␣
 ↪hidden_dim=hidden_dim, num_layers=1, dropout=dropout).to(device)  # build a␣
 ↪model

vocab_size=len(vocab)

# loss and optimizer
criterion = nn.CrossEntropyLoss()  # CE loss
optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
 ↪weight_decay=weight_decay)  # optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=5,
                                            gamma=0.5)  # decay LR by a factor of 0.5␣
 ↪every 10 epochs. You can change this

# logs
Train_Losses = []  # record average training loss each epoch
Val_Losses = []   # record average validation loss each epoch
total_step = len(train_data_loader)  # number of iterations each epoch
best_val_loss = np.inf

# start training
print('Start training...')
import time
tic = time.time()
```

```
for epoch in range(num_epochs):
# for epoch in range(3):
    print('Switch to training...')
    model.train()
    Train_loss_iter = []  # record the the training loss each iteration
    for itr, (images, captions, lengths) in enumerate(train_data_loader):
        #######Your Code###########
        # train your model
        model.zero_grad()
        images = Variable(images).to(device)
        captions = Variable(captions).to(device)
        predicted_cap_pack = pack_padded_sequence(captions, lengths,␣
↪batch_first=True)[0]
        predicted_cap = model(images, captions, lengths)
        predicted_cap = predicted_cap.view(-1, vocab_size)
        loss = criterion(input = predicted_cap, target=predicted_cap_pack)
        loss.backward()
        optimizer.step()

        # record the training loss
        Train_Losses.append(float(loss))

        # print log info
        if itr % log_step == 0:
            # print current loss and perplexity
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Perplexity: {:5.
↪4f}'
                      .format(epoch, num_epochs, itr, total_step, loss.item(),␣
↪np.exp(loss.item())))
    scheduler.step()
    Train_Losses.append(np.mean(Train_loss_iter))
    np.save(os.path.join(output_dir, 'TrainingLoss_rnn.npy'), Train_Losses)  #␣
↪save the training loss

    model.eval()
    # (optional) generate a sample during the training, you can use␣
↪deterministic mode
    # Your code


    # validation
    Val_Losses.append(val(model, val_data_loader, vocab))
    np.save(os.path.join(output_dir, 'ValLoss_rnn.npy'), Val_Losses) # save the␣
↪val loss

    # save model
```

```
    if Val_Losses[-1] < best_val_loss:
        best_val_loss = Val_Losses[-1]
        print('updated best val loss:', best_val_loss)
        print('Save model weights to...', output_dir)
        torch.save(model.state_dict(),
                   os.path.join(output_dir, 'vanilla_rnn-best.pth'.format(epoch␣
↪+ 1, itr + 1)))

print('It took: {} s'.format(time.time() - tic))
```

```
Downloading: "https://download.pytorch.org/models/resnet50-19c8e357.pth" to
/root/.cache/torch/hub/checkpoints/resnet50-19c8e357.pth

HBox(children=(FloatProgress(value=0.0, max=102502400.0), HTML(value='')))


Start training...
Switch to training...
Epoch [0/50], Step [0/114], Loss: 9.2990, Perplexity: 10927.0114
Epoch [0/50], Step [50/114], Loss: 3.8386, Perplexity: 46.4584
Epoch [0/50], Step [100/114], Loss: 3.6042, Perplexity: 36.7507
Validating...

/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3335:
RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:161:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)

Step [0/127], Loss: 4.1656, Perplexity: 64.4300
Step [50/127], Loss: 4.0322, Perplexity: 56.3838
Step [100/127], Loss: 3.2433, Perplexity: 25.6184
updated best val loss: 3.671107530593872
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [1/50], Step [0/114], Loss: 3.5300, Perplexity: 34.1254
Epoch [1/50], Step [50/114], Loss: 3.5267, Perplexity: 34.0123
Epoch [1/50], Step [100/114], Loss: 3.5725, Perplexity: 35.6045
Validating...
Step [0/127], Loss: 3.8533, Perplexity: 47.1464
Step [50/127], Loss: 4.3051, Perplexity: 74.0762
Step [100/127], Loss: 3.4891, Perplexity: 32.7554
updated best val loss: 3.601075799446406
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
```

```
Epoch [2/50], Step [0/114], Loss: 3.4513, Perplexity: 31.5418
Epoch [2/50], Step [50/114], Loss: 3.5018, Perplexity: 33.1766
Epoch [2/50], Step [100/114], Loss: 3.5181, Perplexity: 33.7187
Validating...
Step [0/127], Loss: 3.4302, Perplexity: 30.8815
Step [50/127], Loss: 4.0189, Perplexity: 55.6414
Step [100/127], Loss: 3.8191, Perplexity: 45.5614
updated best val loss: 3.5668688045711967
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [3/50], Step [0/114], Loss: 3.4842, Perplexity: 32.5955
Epoch [3/50], Step [50/114], Loss: 3.5505, Perplexity: 34.8302
Epoch [3/50], Step [100/114], Loss: 3.4859, Perplexity: 32.6529
Validating...
Step [0/127], Loss: 3.2147, Perplexity: 24.8964
Step [50/127], Loss: 4.0613, Perplexity: 58.0522
Step [100/127], Loss: 3.8623, Perplexity: 47.5727
Switch to training...
Epoch [4/50], Step [0/114], Loss: 3.5081, Perplexity: 33.3854
Epoch [4/50], Step [50/114], Loss: 3.4329, Perplexity: 30.9649
Epoch [4/50], Step [100/114], Loss: 3.4119, Perplexity: 30.3242
Validating...
Step [0/127], Loss: 3.7977, Perplexity: 44.5978
Step [50/127], Loss: 2.9505, Perplexity: 19.1151
Step [100/127], Loss: 3.3779, Perplexity: 29.3080
updated best val loss: 3.512489335743461
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [5/50], Step [0/114], Loss: 3.4232, Perplexity: 30.6674
Epoch [5/50], Step [50/114], Loss: 3.2039, Perplexity: 24.6274
Epoch [5/50], Step [100/114], Loss: 3.2004, Perplexity: 24.5415
Validating...
Step [0/127], Loss: 3.5653, Perplexity: 35.3500
Step [50/127], Loss: 3.3220, Perplexity: 27.7157
Step [100/127], Loss: 3.7276, Perplexity: 41.5801
updated best val loss: 3.337581520005474
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [6/50], Step [0/114], Loss: 3.2339, Perplexity: 25.3792
Epoch [6/50], Step [50/114], Loss: 3.2582, Perplexity: 26.0024
Epoch [6/50], Step [100/114], Loss: 3.1173, Perplexity: 22.5864
Validating...
Step [0/127], Loss: 3.4112, Perplexity: 30.3019
Step [50/127], Loss: 2.9932, Perplexity: 19.9488
Step [100/127], Loss: 3.0383, Perplexity: 20.8707
```

```
Switch to training...
Epoch [7/50], Step [0/114], Loss: 3.1146, Perplexity: 22.5242
Epoch [7/50], Step [50/114], Loss: 3.0713, Perplexity: 21.5708
Epoch [7/50], Step [100/114], Loss: 3.0943, Perplexity: 22.0722
Validating...
Step [0/127], Loss: 3.2245, Perplexity: 25.1406
Step [50/127], Loss: 3.2410, Perplexity: 25.5605
Step [100/127], Loss: 3.1710, Perplexity: 23.8301
updated best val loss: 3.288263664470883
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [8/50], Step [0/114], Loss: 3.1889, Perplexity: 24.2626
Epoch [8/50], Step [50/114], Loss: 3.0698, Perplexity: 21.5368
Epoch [8/50], Step [100/114], Loss: 3.1861, Perplexity: 24.1932
Validating...
Step [0/127], Loss: 3.3716, Perplexity: 29.1255
Step [50/127], Loss: 4.2271, Perplexity: 68.5157
Step [100/127], Loss: 3.4140, Perplexity: 30.3867
Switch to training...
Epoch [9/50], Step [0/114], Loss: 3.0014, Perplexity: 20.1132
Epoch [9/50], Step [50/114], Loss: 3.0874, Perplexity: 21.9203
Epoch [9/50], Step [100/114], Loss: 3.1656, Perplexity: 23.7035
Validating...
Step [0/127], Loss: 3.2300, Perplexity: 25.2794
Step [50/127], Loss: 3.2347, Perplexity: 25.3999
Step [100/127], Loss: 3.4253, Perplexity: 30.7326
Switch to training...
Epoch [10/50], Step [0/114], Loss: 3.1566, Perplexity: 23.4901
Epoch [10/50], Step [50/114], Loss: 3.0670, Perplexity: 21.4782
Epoch [10/50], Step [100/114], Loss: 2.9891, Perplexity: 19.8669
Validating...
Step [0/127], Loss: 3.3540, Perplexity: 28.6171
Step [50/127], Loss: 3.4342, Perplexity: 31.0058
Step [100/127], Loss: 3.4286, Perplexity: 30.8341
updated best val loss: 3.1965027155838612
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [11/50], Step [0/114], Loss: 2.9665, Perplexity: 19.4236
Epoch [11/50], Step [50/114], Loss: 3.0302, Perplexity: 20.7013
Epoch [11/50], Step [100/114], Loss: 2.9597, Perplexity: 19.2922
Validating...
Step [0/127], Loss: 2.5568, Perplexity: 12.8939
Step [50/127], Loss: 3.1069, Perplexity: 22.3521
Step [100/127], Loss: 3.0165, Perplexity: 20.4200
Switch to training...
Epoch [12/50], Step [0/114], Loss: 2.9106, Perplexity: 18.3684
```

```
Epoch [12/50], Step [50/114], Loss: 2.9672, Perplexity: 19.4369
Epoch [12/50], Step [100/114], Loss: 2.9683, Perplexity: 19.4594
Validating...
Step [0/127], Loss: 2.8816, Perplexity: 17.8419
Step [50/127], Loss: 2.8213, Perplexity: 16.7983
Step [100/127], Loss: 3.4687, Perplexity: 32.0965
updated best val loss: 3.1893812953017826
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [13/50], Step [0/114], Loss: 2.9594, Perplexity: 19.2862
Epoch [13/50], Step [50/114], Loss: 2.9048, Perplexity: 18.2609
Epoch [13/50], Step [100/114], Loss: 3.0572, Perplexity: 21.2681
Validating...
Step [0/127], Loss: 2.9277, Perplexity: 18.6846
Step [50/127], Loss: 3.1788, Perplexity: 24.0182
Step [100/127], Loss: 3.4187, Perplexity: 30.5299
Switch to training...
Epoch [14/50], Step [0/114], Loss: 2.8939, Perplexity: 18.0634
Epoch [14/50], Step [50/114], Loss: 2.9014, Perplexity: 18.1993
Epoch [14/50], Step [100/114], Loss: 2.9725, Perplexity: 19.5414
Validating...
Step [0/127], Loss: 2.9496, Perplexity: 19.0987
Step [50/127], Loss: 2.9935, Perplexity: 19.9547
Step [100/127], Loss: 2.5781, Perplexity: 13.1719
updated best val loss: 3.1814778770987444
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [15/50], Step [0/114], Loss: 2.9933, Perplexity: 19.9514
Epoch [15/50], Step [50/114], Loss: 2.8711, Perplexity: 17.6560
Epoch [15/50], Step [100/114], Loss: 2.8944, Perplexity: 18.0730
Validating...
Step [0/127], Loss: 3.4155, Perplexity: 30.4334
Step [50/127], Loss: 3.0547, Perplexity: 21.2146
Step [100/127], Loss: 3.4499, Perplexity: 31.4958
updated best val loss: 3.1385314502115325
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [16/50], Step [0/114], Loss: 2.8434, Perplexity: 17.1738
Epoch [16/50], Step [50/114], Loss: 2.8848, Perplexity: 17.8997
Epoch [16/50], Step [100/114], Loss: 2.9165, Perplexity: 18.4764
Validating...
Step [0/127], Loss: 2.9856, Perplexity: 19.7976
Step [50/127], Loss: 3.4809, Perplexity: 32.4885
Step [100/127], Loss: 3.3999, Perplexity: 29.9615
Switch to training...
```

```
Epoch [17/50], Step [0/114], Loss: 2.7847, Perplexity: 16.1955
Epoch [17/50], Step [50/114], Loss: 2.8731, Perplexity: 17.6912
Epoch [17/50], Step [100/114], Loss: 2.8951, Perplexity: 18.0853
Validating...
Step [0/127], Loss: 3.1474, Perplexity: 23.2749
Step [50/127], Loss: 3.1053, Perplexity: 22.3153
Step [100/127], Loss: 3.0071, Perplexity: 20.2291
updated best val loss: 3.1380890542127955
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [18/50], Step [0/114], Loss: 2.9423, Perplexity: 18.9585
Epoch [18/50], Step [50/114], Loss: 2.9052, Perplexity: 18.2689
Epoch [18/50], Step [100/114], Loss: 2.8824, Perplexity: 17.8568
Validating...
Step [0/127], Loss: 3.1679, Perplexity: 23.7564
Step [50/127], Loss: 3.0608, Perplexity: 21.3446
Step [100/127], Loss: 2.9254, Perplexity: 18.6414
Switch to training...
Epoch [19/50], Step [0/114], Loss: 2.7966, Perplexity: 16.3892
Epoch [19/50], Step [50/114], Loss: 2.8129, Perplexity: 16.6585
Epoch [19/50], Step [100/114], Loss: 2.8923, Perplexity: 18.0342
Validating...
Step [0/127], Loss: 3.5365, Perplexity: 34.3450
Step [50/127], Loss: 3.5281, Perplexity: 34.0578
Step [100/127], Loss: 2.8435, Perplexity: 17.1759
Switch to training...
Epoch [20/50], Step [0/114], Loss: 2.8256, Perplexity: 16.8713
Epoch [20/50], Step [50/114], Loss: 2.7618, Perplexity: 15.8289
Epoch [20/50], Step [100/114], Loss: 2.7990, Perplexity: 16.4283
Validating...
Step [0/127], Loss: 3.1563, Perplexity: 23.4844
Step [50/127], Loss: 3.4632, Perplexity: 31.9180
Step [100/127], Loss: 2.7674, Perplexity: 15.9170
updated best val loss: 3.117585886181809
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [21/50], Step [0/114], Loss: 2.8209, Perplexity: 16.7913
Epoch [21/50], Step [50/114], Loss: 2.7216, Perplexity: 15.2044
Epoch [21/50], Step [100/114], Loss: 2.8026, Perplexity: 16.4871
Validating...
Step [0/127], Loss: 3.0410, Perplexity: 20.9268
Step [50/127], Loss: 3.0794, Perplexity: 21.7462
Step [100/127], Loss: 3.1394, Perplexity: 23.0901
updated best val loss: 3.100470419005146
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
```

```
Switch to training...
Epoch [22/50], Step [0/114], Loss: 2.8679, Perplexity: 17.6006
Epoch [22/50], Step [50/114], Loss: 2.7297, Perplexity: 15.3283
Epoch [22/50], Step [100/114], Loss: 2.7226, Perplexity: 15.2199
Validating...
Step [0/127], Loss: 2.7374, Perplexity: 15.4467
Step [50/127], Loss: 3.0840, Perplexity: 21.8460
Step [100/127], Loss: 3.3896, Perplexity: 29.6538
Switch to training...
Epoch [23/50], Step [0/114], Loss: 2.7286, Perplexity: 15.3122
Epoch [23/50], Step [50/114], Loss: 2.7789, Perplexity: 16.1018
Epoch [23/50], Step [100/114], Loss: 2.7830, Perplexity: 16.1681
Validating...
Step [0/127], Loss: 4.0628, Perplexity: 58.1362
Step [50/127], Loss: 3.2350, Perplexity: 25.4059
Step [100/127], Loss: 3.2519, Perplexity: 25.8391
Switch to training...
Epoch [24/50], Step [0/114], Loss: 2.8066, Perplexity: 16.5541
Epoch [24/50], Step [50/114], Loss: 2.6985, Perplexity: 14.8574
Epoch [24/50], Step [100/114], Loss: 2.8212, Perplexity: 16.7977
Validating...
Step [0/127], Loss: 3.9105, Perplexity: 49.9225
Step [50/127], Loss: 3.0598, Perplexity: 21.3238
Step [100/127], Loss: 2.3015, Perplexity: 9.9892
updated best val loss: 3.0731245964530887
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [25/50], Step [0/114], Loss: 2.7803, Perplexity: 16.1242
Epoch [25/50], Step [50/114], Loss: 2.7777, Perplexity: 16.0813
Epoch [25/50], Step [100/114], Loss: 2.7227, Perplexity: 15.2221
Validating...
Step [0/127], Loss: 3.1107, Perplexity: 22.4365
Step [50/127], Loss: 3.2879, Perplexity: 26.7857
Step [100/127], Loss: 3.4780, Perplexity: 32.3938
updated best val loss: 3.0715413844491555
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [26/50], Step [0/114], Loss: 2.8767, Perplexity: 17.7555
Epoch [26/50], Step [50/114], Loss: 2.7009, Perplexity: 14.8925
Epoch [26/50], Step [100/114], Loss: 2.8266, Perplexity: 16.8886
Validating...
Step [0/127], Loss: 3.0918, Perplexity: 22.0173
Step [50/127], Loss: 3.4706, Perplexity: 32.1560
Step [100/127], Loss: 3.0168, Perplexity: 20.4254
Switch to training...
Epoch [27/50], Step [0/114], Loss: 2.7927, Perplexity: 16.3245
```

```
Epoch [27/50], Step [50/114], Loss: 2.7265, Perplexity: 15.2792
Epoch [27/50], Step [100/114], Loss: 2.7699, Perplexity: 15.9568
Validating...
Step [0/127], Loss: 3.3283, Perplexity: 27.8922
Step [50/127], Loss: 3.5669, Perplexity: 35.4055
Step [100/127], Loss: 2.8196, Perplexity: 16.7697
Switch to training...
Epoch [28/50], Step [0/114], Loss: 2.7356, Perplexity: 15.4186
Epoch [28/50], Step [50/114], Loss: 2.7565, Perplexity: 15.7445
Epoch [28/50], Step [100/114], Loss: 2.7027, Perplexity: 14.9205
Validating...
Step [0/127], Loss: 3.7608, Perplexity: 42.9845
Step [50/127], Loss: 2.9381, Perplexity: 18.8804
Step [100/127], Loss: 3.2651, Perplexity: 26.1834
Switch to training...
Epoch [29/50], Step [0/114], Loss: 2.7303, Perplexity: 15.3375
Epoch [29/50], Step [50/114], Loss: 2.7352, Perplexity: 15.4129
Epoch [29/50], Step [100/114], Loss: 2.6747, Perplexity: 14.5075
Validating...
Step [0/127], Loss: 3.0812, Perplexity: 21.7847
Step [50/127], Loss: 2.6892, Perplexity: 14.7202
Step [100/127], Loss: 3.1091, Perplexity: 22.4000
Switch to training...
Epoch [30/50], Step [0/114], Loss: 2.7688, Perplexity: 15.9401
Epoch [30/50], Step [50/114], Loss: 2.7326, Perplexity: 15.3724
Epoch [30/50], Step [100/114], Loss: 2.7484, Perplexity: 15.6174
Validating...
Step [0/127], Loss: 2.6037, Perplexity: 13.5142
Step [50/127], Loss: 2.9568, Perplexity: 19.2357
Step [100/127], Loss: 2.7856, Perplexity: 16.2100
Switch to training...
Epoch [31/50], Step [0/114], Loss: 2.7249, Perplexity: 15.2542
Epoch [31/50], Step [50/114], Loss: 2.7234, Perplexity: 15.2315
Epoch [31/50], Step [100/114], Loss: 2.7434, Perplexity: 15.5403
Validating...
Step [0/127], Loss: 3.3860, Perplexity: 29.5490
Step [50/127], Loss: 2.7989, Perplexity: 16.4267
Step [100/127], Loss: 3.2641, Perplexity: 26.1556
Switch to training...
Epoch [32/50], Step [0/114], Loss: 2.7916, Perplexity: 16.3079
Epoch [32/50], Step [50/114], Loss: 2.7383, Perplexity: 15.4608
Epoch [32/50], Step [100/114], Loss: 2.7144, Perplexity: 15.0953
Validating...
Step [0/127], Loss: 3.4751, Perplexity: 32.2995
Step [50/127], Loss: 3.3821, Perplexity: 29.4328
Step [100/127], Loss: 3.2108, Perplexity: 24.7988
Switch to training...
Epoch [33/50], Step [0/114], Loss: 2.7368, Perplexity: 15.4375
```

```
Epoch [33/50], Step [50/114], Loss: 2.6621, Perplexity: 14.3256
Epoch [33/50], Step [100/114], Loss: 2.7613, Perplexity: 15.8206
Validating...
Step [0/127], Loss: 2.8184, Perplexity: 16.7500
Step [50/127], Loss: 3.2714, Perplexity: 26.3470
Step [100/127], Loss: 3.7577, Perplexity: 42.8496
Switch to training...
Epoch [34/50], Step [0/114], Loss: 2.7287, Perplexity: 15.3122
Epoch [34/50], Step [50/114], Loss: 2.7516, Perplexity: 15.6681
Epoch [34/50], Step [100/114], Loss: 2.6906, Perplexity: 14.7412
Validating...
Step [0/127], Loss: 3.4691, Perplexity: 32.1068
Step [50/127], Loss: 2.9756, Perplexity: 19.6022
Step [100/127], Loss: 3.2287, Perplexity: 25.2458
Switch to training...
Epoch [35/50], Step [0/114], Loss: 2.7312, Perplexity: 15.3511
Epoch [35/50], Step [50/114], Loss: 2.7032, Perplexity: 14.9278
Epoch [35/50], Step [100/114], Loss: 2.7533, Perplexity: 15.6936
Validating...
Step [0/127], Loss: 3.0698, Perplexity: 21.5379
Step [50/127], Loss: 3.3699, Perplexity: 29.0770
Step [100/127], Loss: 3.4739, Perplexity: 32.2633
Switch to training...
Epoch [36/50], Step [0/114], Loss: 2.6774, Perplexity: 14.5477
Epoch [36/50], Step [50/114], Loss: 2.6717, Perplexity: 14.4649
Epoch [36/50], Step [100/114], Loss: 2.6921, Perplexity: 14.7624
Validating...
Step [0/127], Loss: 2.6614, Perplexity: 14.3159
Step [50/127], Loss: 3.5899, Perplexity: 36.2298
Step [100/127], Loss: 2.7627, Perplexity: 15.8429
Switch to training...
Epoch [37/50], Step [0/114], Loss: 2.7478, Perplexity: 15.6083
Epoch [37/50], Step [50/114], Loss: 2.7510, Perplexity: 15.6587
Epoch [37/50], Step [100/114], Loss: 2.6760, Perplexity: 14.5275
Validating...
Step [0/127], Loss: 3.4586, Perplexity: 31.7735
Step [50/127], Loss: 3.2749, Perplexity: 26.4411
Step [100/127], Loss: 3.8873, Perplexity: 48.7769
Switch to training...
Epoch [38/50], Step [0/114], Loss: 2.7379, Perplexity: 15.4548
Epoch [38/50], Step [50/114], Loss: 2.7183, Perplexity: 15.1547
Epoch [38/50], Step [100/114], Loss: 2.7442, Perplexity: 15.5518
Validating...
Step [0/127], Loss: 2.4631, Perplexity: 11.7407
Step [50/127], Loss: 3.1684, Perplexity: 23.7691
Step [100/127], Loss: 3.8304, Perplexity: 46.0823
updated best val loss: 3.0556812286376953
Save model weights to... /content/drive/My
```

```
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [39/50], Step [0/114], Loss: 2.7851, Perplexity: 16.2021
Epoch [39/50], Step [50/114], Loss: 2.7309, Perplexity: 15.3461
Epoch [39/50], Step [100/114], Loss: 2.7198, Perplexity: 15.1769
Validating...
Step [0/127], Loss: 3.6130, Perplexity: 37.0757
Step [50/127], Loss: 2.7318, Perplexity: 15.3606
Step [100/127], Loss: 2.8113, Perplexity: 16.6311
updated best val loss: 3.0484642794751746
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/rnn/
Switch to training...
Epoch [40/50], Step [0/114], Loss: 2.7373, Perplexity: 15.4445
Epoch [40/50], Step [50/114], Loss: 2.7213, Perplexity: 15.1999
Epoch [40/50], Step [100/114], Loss: 2.7665, Perplexity: 15.9027
Validating...
Step [0/127], Loss: 3.4611, Perplexity: 31.8508
Step [50/127], Loss: 3.0259, Perplexity: 20.6133
Step [100/127], Loss: 3.0787, Perplexity: 21.7303
Switch to training...
Epoch [41/50], Step [0/114], Loss: 2.7094, Perplexity: 15.0206
Epoch [41/50], Step [50/114], Loss: 2.7782, Perplexity: 16.0903
Epoch [41/50], Step [100/114], Loss: 2.6633, Perplexity: 14.3431
Validating...
Step [0/127], Loss: 3.6181, Perplexity: 37.2682
Step [50/127], Loss: 3.0113, Perplexity: 20.3140
Step [100/127], Loss: 3.0394, Perplexity: 20.8922
Switch to training...
Epoch [42/50], Step [0/114], Loss: 2.7026, Perplexity: 14.9185
Epoch [42/50], Step [50/114], Loss: 2.7593, Perplexity: 15.7886
Epoch [42/50], Step [100/114], Loss: 2.7019, Perplexity: 14.9076
Validating...
Step [0/127], Loss: 2.7947, Perplexity: 16.3580
Step [50/127], Loss: 4.3382, Perplexity: 76.5699
Step [100/127], Loss: 3.0872, Perplexity: 21.9147
Switch to training...
Epoch [43/50], Step [0/114], Loss: 2.7254, Perplexity: 15.2621
Epoch [43/50], Step [50/114], Loss: 2.6250, Perplexity: 13.8040
Epoch [43/50], Step [100/114], Loss: 2.7765, Perplexity: 16.0625
Validating...
Step [0/127], Loss: 3.5173, Perplexity: 33.6938
Step [50/127], Loss: 3.0043, Perplexity: 20.1721
Step [100/127], Loss: 3.4019, Perplexity: 30.0217
Switch to training...
Epoch [44/50], Step [0/114], Loss: 2.6477, Perplexity: 14.1215
Epoch [44/50], Step [50/114], Loss: 2.6653, Perplexity: 14.3728
Epoch [44/50], Step [100/114], Loss: 2.7234, Perplexity: 15.2314
```

```
Validating...
Step [0/127], Loss: 3.2623, Perplexity: 26.1089
Step [50/127], Loss: 3.2486, Perplexity: 25.7533
Step [100/127], Loss: 2.5923, Perplexity: 13.3606
Switch to training...
Epoch [45/50], Step [0/114], Loss: 2.7444, Perplexity: 15.5547
Epoch [45/50], Step [50/114], Loss: 2.7405, Perplexity: 15.4950
Epoch [45/50], Step [100/114], Loss: 2.7032, Perplexity: 14.9269
Validating...
Step [0/127], Loss: 3.0232, Perplexity: 20.5563
Step [50/127], Loss: 3.2143, Perplexity: 24.8846
Step [100/127], Loss: 2.7735, Perplexity: 16.0152
Switch to training...
Epoch [46/50], Step [0/114], Loss: 2.7590, Perplexity: 15.7844
Epoch [46/50], Step [50/114], Loss: 2.6928, Perplexity: 14.7725
Epoch [46/50], Step [100/114], Loss: 2.6443, Perplexity: 14.0729
Validating...
Step [0/127], Loss: 3.7961, Perplexity: 44.5283
Step [50/127], Loss: 3.0172, Perplexity: 20.4331
Step [100/127], Loss: 2.9283, Perplexity: 18.6951
Switch to training...
Epoch [47/50], Step [0/114], Loss: 2.7161, Perplexity: 15.1213
Epoch [47/50], Step [50/114], Loss: 2.6394, Perplexity: 14.0049
Epoch [47/50], Step [100/114], Loss: 2.6928, Perplexity: 14.7724
Validating...
Step [0/127], Loss: 3.5637, Perplexity: 35.2938
Step [50/127], Loss: 3.0865, Perplexity: 21.8998
Step [100/127], Loss: 3.6792, Perplexity: 39.6135
Switch to training...
Epoch [48/50], Step [0/114], Loss: 2.7707, Perplexity: 15.9698
Epoch [48/50], Step [50/114], Loss: 2.6973, Perplexity: 14.8396
Epoch [48/50], Step [100/114], Loss: 2.7517, Perplexity: 15.6699
Validating...
Step [0/127], Loss: 3.6207, Perplexity: 37.3621
Step [50/127], Loss: 3.2497, Perplexity: 25.7830
Step [100/127], Loss: 2.8976, Perplexity: 18.1298
Switch to training...
Epoch [49/50], Step [0/114], Loss: 2.6939, Perplexity: 14.7899
Epoch [49/50], Step [50/114], Loss: 2.7550, Perplexity: 15.7213
Epoch [49/50], Step [100/114], Loss: 2.7109, Perplexity: 15.0423
Validating...
Step [0/127], Loss: 3.3984, Perplexity: 29.9159
Step [50/127], Loss: 3.0992, Perplexity: 22.1798
Step [100/127], Loss: 3.0750, Perplexity: 21.6491
It took: 13557.006758928299 s
```

12 Section 2.4 Evaluation [10 pts]

[15]:
```
## evaluation code
from tqdm import tqdm, tqdm_notebook
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import SmoothingFunction
smoother = SmoothingFunction()

def caption_generator(model, images, vocab, img_ids, captions,␣
↪mode='Deterministic', temperature=1.0):
    """
    Generate captions.
    :param mode:
    :return:
    """
    sample_idxs = model.sample_generate(images, mode=mode,
                                        temperature=temperature).data.cpu().
↪numpy()  # [N, max_length]
    for i, sentence in enumerate(sample_idxs):  # every sentence in this batch
    # for sentence in sample_idxs:
        sentence_caption = ''
        for word_idx in sentence:
            word = vocab.idx2word[word_idx]
            if word != '<start>' and word != '<end>':
                if word == '.':
                    sentence_caption += '.'
                else:
                    sentence_caption += word + ' '
            if word == '<end>':
                break
        captions.append({'caption': sentence_caption})
        # captions.append(sentence_caption)

    return captions

def run_test(model, data_loader, vocab, mode='Deterministic', temperature=1.0):
    """
    Run your model on the test set.
    Inputs:
    :param model: the model you use
    :param data_loader: the data_loader
    :param mode: use 'deterministic' or 'stochastic'
    Outputs:
    :param predictions
    """
    predictions = []
    for itr, (images, captions, lengths) in enumerate(tqdm(data_loader)):
```

```
        images = Variable(images).to(device)
        captions = Variable(captions).to(device)
        outputs = model(images, captions, lengths)

        img_ids = list(range(itr * data_loader.batch_size, (itr + 1) *␣
↪data_loader.batch_size))
        predictions = caption_generator(model, images, vocab, img_ids,
                                        predictions, mode=mode,␣
↪temperature=temperature)

    return predictions

def evaluation(model, vocab, data_path=path_to_homework + '/flickr30k_images/',␣
↪mode='Deterministic', temperature=1.0,
               split='test'):
    """
    Evaluate the performance of your model on the test set using BLEU scores.
    Inputs:
    :param model: the model you use
    :param weight_path: the directory to the weights of your model
    :param vocab: vocabulary
    :param data_path: the directory to the dataset
    :param mode: use 'deterministic' or 'stochastic'
    Outputs:
    :param predictions
    """
    # data loader
    test_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
↪split=split, vocab=vocab,
                                  transform=transform, batch_size=8,␣
↪shuffle=False, num_workers=4)

    # run your model on the test set
    print('Run on the test set...')
    preds = run_test(model, test_data_loader, vocab, mode, temperature)

    # load the groundtruth
    gt = test_data_loader.dataset.annos

    # evaluate the performance using BLEU score
    score1 = 0
    score2 = 0
    score3 = 0
    score4 = 0

    print('Computing BLEU')
    for itr in tqdm(range(len(gt))):
```

```
        candidate = preds[itr]['caption']
        reference = [sent['raw'] for sent in gt[itr]['sentences']]
        score1 += sentence_bleu(reference, candidate, weights=(1, 0, 0, 0),␣
↪smoothing_function=smoother.method1)
        score2 += sentence_bleu(reference, candidate, weights=(0, 1, 0, 0),␣
↪smoothing_function=smoother.method1)
        score3 += sentence_bleu(reference, candidate, weights=(0, 0, 1, 0),␣
↪smoothing_function=smoother.method1)
        score4 += sentence_bleu(reference, candidate, weights=(0, 0, 0, 1),␣
↪smoothing_function=smoother.method1)

    bleu1 = 100 * score1/len(gt)
    bleu2 = 100 * score2/len(gt)
    bleu3 = 100 * score3/len(gt)
    bleu4 = 100 * score4/len(gt)

    return bleu1, bleu2, bleu3, bleu4
```

- Test your outputs in the **Deterministic** way by using BLEU scores. You should at achieve a BLEU 4 of 25.

[17]:
```
## Evaluate your model using BLEU score. Use Deterministic mode.

## Image transformation
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #     transforms.RandomCrop(224, pad_if_needed=True),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))])

## Evaluate your model using BLEU score. Use Deterministic mode
model = Vanilla_rnn(vocab_size=len(vocab), emb_dim=emb_dim,␣
↪hidden_dim=hidden_dim,
                    num_layers=1, dropout=dropout).to(device)  # build a model
model.load_state_dict(torch.load(path_to_homework + '/checkpoints/rnn/
↪vanilla_rnn-best.pth', map_location=torch.device('cpu')))
model.eval()
bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic')
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
↪bleu4))
```

```
  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...
```

```
  1%|          | 1/125 [02:37<5:25:21, 157.43s/it]
  2%|         | 2/125 [02:37<3:45:58, 110.23s/it]
  3%|         | 4/125 [02:37<2:35:39, 77.18s/it]
  6%|         | 7/125 [02:37<1:46:17, 54.04s/it]
  7%|         | 9/125 [02:39<1:13:35, 38.06s/it]
  9%|         | 11/125 [02:39<50:41, 26.68s/it]
 10%|         | 12/125 [02:40<35:51, 19.04s/it]
 10%|         | 13/125 [02:41<25:04, 13.43s/it]
 11%|         | 14/125 [02:41<17:28,  9.44s/it]
 12%|        | 15/125 [02:41<12:11,  6.65s/it]
 13%|        | 16/125 [02:42<09:08,  5.03s/it]
 14%|        | 17/125 [02:42<06:25,  3.57s/it]
 14%|        | 18/125 [02:43<04:46,  2.68s/it]
 16%|        | 20/125 [02:44<03:36,  2.07s/it]
 17%|        | 21/125 [02:44<02:34,  1.48s/it]
 18%|        | 22/125 [02:45<01:57,  1.15s/it]
 18%|        | 23/125 [02:45<01:32,  1.10it/s]
 19%|        | 24/125 [02:46<01:37,  1.04it/s]
 21%|        | 26/125 [02:46<01:09,  1.42it/s]
 22%|       | 27/125 [02:47<01:17,  1.27it/s]
 22%|       | 28/125 [02:48<01:07,  1.43it/s]
 24%|       | 30/125 [02:48<00:50,  1.90it/s]
 25%|       | 31/125 [02:49<00:58,  1.61it/s]
 26%|       | 32/125 [02:50<01:02,  1.48it/s]
 28%|       | 35/125 [02:51<00:50,  1.77it/s]
 29%|       | 36/125 [02:51<00:56,  1.58it/s]
 30%|       | 38/125 [02:52<00:45,  1.90it/s]
 31%|       | 39/125 [02:52<00:41,  2.06it/s]
 32%|      | 40/125 [02:54<01:00,  1.40it/s]
 34%|      | 43/125 [02:54<00:45,  1.82it/s]
 35%|      | 44/125 [02:56<01:06,  1.21it/s]
 38%|      | 47/125 [02:56<00:47,  1.66it/s]
 38%|      | 48/125 [02:57<01:06,  1.15it/s]
 40%|      | 50/125 [02:58<00:48,  1.56it/s]
 42%|     | 52/125 [02:59<00:48,  1.51it/s]
 43%|     | 54/125 [02:59<00:37,  1.89it/s]
 44%|     | 55/125 [03:00<00:30,  2.26it/s]
 45%|     | 56/125 [03:01<00:53,  1.29it/s]
 46%|     | 58/125 [03:01<00:38,  1.73it/s]
 47%|     | 59/125 [03:02<00:40,  1.62it/s]
 48%|     | 60/125 [03:03<00:46,  1.39it/s]
 50%|     | 62/125 [03:04<00:35,  1.76it/s]
 50%|     | 63/125 [03:04<00:29,  2.08it/s]
 51%|     | 64/125 [03:05<00:37,  1.62it/s]
 53%|    | 66/125 [03:05<00:29,  1.99it/s]
 54%|    | 67/125 [03:06<00:28,  2.00it/s]
 54%|    | 68/125 [03:06<00:29,  1.94it/s]
```

```
 56%|     | 70/125 [03:07<00:25,  2.18it/s]
 57%|     | 71/125 [03:07<00:25,  2.13it/s]
 58%|     | 72/125 [03:08<00:24,  2.12it/s]
 59%|     | 74/125 [03:09<00:24,  2.09it/s]
 60%|     | 75/125 [03:09<00:19,  2.56it/s]
 61%|     | 76/125 [03:10<00:21,  2.32it/s]
 62%|    | 78/125 [03:11<00:22,  2.10it/s]
 63%|    | 79/125 [03:11<00:20,  2.24it/s]
 64%|    | 80/125 [03:11<00:17,  2.57it/s]
 66%|    | 82/125 [03:12<00:18,  2.29it/s]
 66%|    | 83/125 [03:13<00:16,  2.53it/s]
 67%|    | 84/125 [03:14<00:21,  1.93it/s]
 69%|    | 86/125 [03:14<00:17,  2.19it/s]
 70%|    | 87/125 [03:14<00:14,  2.55it/s]
 70%|    | 88/125 [03:15<00:21,  1.75it/s]
 72%|   | 90/125 [03:16<00:16,  2.09it/s]
 73%|   | 91/125 [03:16<00:14,  2.42it/s]
 74%|   | 92/125 [03:17<00:20,  1.59it/s]
 75%|   | 94/125 [03:17<00:14,  2.18it/s]
 76%|   | 95/125 [03:18<00:14,  2.01it/s]
 77%|   | 96/125 [03:19<00:21,  1.35it/s]
 79%|   | 99/125 [03:20<00:15,  1.72it/s]
 80%|   | 100/125 [03:21<00:19,  1.31it/s]
 82%|  | 102/125 [03:21<00:12,  1.82it/s]
 82%|  | 103/125 [03:22<00:12,  1.74it/s]
 83%|  | 104/125 [03:23<00:14,  1.45it/s]
 85%|  | 106/125 [03:23<00:09,  2.01it/s]
 86%|  | 107/125 [03:24<00:09,  1.96it/s]
 86%|  | 108/125 [03:24<00:10,  1.56it/s]
 88%|  | 110/125 [03:25<00:06,  2.15it/s]
 89%|  | 111/125 [03:25<00:07,  1.79it/s]
 90%|  | 112/125 [03:26<00:08,  1.59it/s]
 91%|  | 114/125 [03:26<00:04,  2.20it/s]
 92%|| 115/125 [03:27<00:05,  1.78it/s]
 93%|| 116/125 [03:38<00:32,  3.59s/it]
 94%|| 118/125 [03:38<00:17,  2.53s/it]
 96%|| 120/125 [03:39<00:09,  2.00s/it]
 98%|| 123/125 [03:39<00:02,  1.41s/it]
100%|| 125/125 [03:41<00:00,  1.77s/it]

  0%|          | 0/1000 [00:00<?, ?it/s]
  2%|         | 23/1000 [00:00<00:04, 228.74it/s]

Computing BLEU


  5%|         | 49/1000 [00:00<00:04, 234.46it/s]
  7%|         | 74/1000 [00:00<00:03, 237.87it/s]
 10%|         | 100/1000 [00:00<00:03, 242.78it/s]
```

```
 13%|        | 126/1000 [00:00<00:03, 247.23it/s]
 15%|        | 152/1000 [00:00<00:03, 249.10it/s]
 18%|        | 179/1000 [00:00<00:03, 253.85it/s]
 20%|        | 205/1000 [00:00<00:03, 255.30it/s]
 23%|       | 231/1000 [00:00<00:02, 256.40it/s]
 26%|       | 257/1000 [00:01<00:02, 257.29it/s]
 28%|       | 283/1000 [00:01<00:02, 251.11it/s]
 31%|       | 310/1000 [00:01<00:02, 255.81it/s]
 34%|      | 336/1000 [00:01<00:02, 256.14it/s]
 36%|      | 362/1000 [00:01<00:02, 253.53it/s]
 39%|      | 389/1000 [00:01<00:02, 257.44it/s]
 42%|     | 415/1000 [00:01<00:02, 250.40it/s]
 44%|     | 441/1000 [00:01<00:02, 252.25it/s]
 47%|     | 467/1000 [00:01<00:02, 254.04it/s]
 49%|     | 493/1000 [00:01<00:01, 255.67it/s]
 52%|    | 519/1000 [00:02<00:01, 252.39it/s]
 55%|    | 545/1000 [00:02<00:01, 247.25it/s]
 57%|    | 572/1000 [00:02<00:01, 251.92it/s]
 60%|    | 598/1000 [00:02<00:01, 252.22it/s]
 62%|   | 624/1000 [00:02<00:01, 250.13it/s]
 65%|   | 650/1000 [00:02<00:01, 246.84it/s]
 68%|   | 675/1000 [00:02<00:01, 245.52it/s]
 70%|   | 700/1000 [00:02<00:01, 243.98it/s]
 72%|  | 725/1000 [00:02<00:01, 245.54it/s]
 75%|  | 750/1000 [00:02<00:01, 245.24it/s]
 78%|  | 775/1000 [00:03<00:00, 246.46it/s]
 80%|  | 800/1000 [00:03<00:00, 240.17it/s]
 82%| | 825/1000 [00:03<00:00, 242.48it/s]
 85%| | 850/1000 [00:03<00:00, 239.53it/s]
 87%| | 874/1000 [00:03<00:00, 239.36it/s]
 90%| | 898/1000 [00:03<00:00, 237.11it/s]
 92%|| 922/1000 [00:03<00:00, 235.86it/s]
 95%|| 946/1000 [00:03<00:00, 225.74it/s]
 97%|| 969/1000 [00:03<00:00, 222.37it/s]
100%|| 1000/1000 [00:04<00:00, 245.38it/s]

BLEU 1:69.2213376024829, BLEU 2:44.871153595427984, BLEU 3:25.158089893489194,
BLEU 4:16.818247623859282
```

- Try different temperatures (e.g. 0.1, 0.2, 0.5, 1.0, 1.5, 2, etc.) during the generation. Report BLEU scores for at least 3 different temperatures.

[ ]:
```
## Use at least 3 different temperatures to generate captions on the test set.␣
↪Report the BLEU scores.
# Your code here
bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
↪temperature=0.5)
```

```
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
 ↪temperature=1.0)
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
 ↪temperature=2.0)
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

# End of code
```

```
  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00, 10.05it/s]
  2%|         | 24/1000 [00:00<00:04, 233.05it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 260.82it/s]

BLEU 1:69.2213376024829, BLEU 2:44.871153595427984, BLEU 3:25.158089893489194,
BLEU 4:16.818247623859282

  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00, 10.06it/s]
  3%|         | 26/1000 [00:00<00:03, 259.91it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 263.60it/s]

BLEU 1:69.2213376024829, BLEU 2:44.871153595427984, BLEU 3:25.158089893489194,
BLEU 4:16.818247623859282

  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00, 10.14it/s]
  3%|         | 27/1000 [00:00<00:03, 269.62it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 263.76it/s]
```

```
BLEU 1:69.2213376024829, BLEU 2:44.871153595427984, BLEU 3:25.158089893489194,
BLEU 4:16.818247623859282
```

13 Section 3 Variations [55 pts]

13.1 Section 3.1 LSTM [35 pts]

13.2 Section 3.1.1 Decoder: LSTM [5 pts]

This time, replace the RNN module with an LSTM module.

[18]:
```
class Decoder(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, num_layers=1, dropout=0):
        """
        Use LSTM as decoder for captions.
        :param emb_dim: Embedding dimensions.
        :param hidden_dim: Hidden states dimensions.
        :param num_layers: Number of LSTM layers.
        :param vocab_size: The size of Vocabulary.
        :param dropout: dropout probability
        """
        super(Decoder, self).__init__()
        ############Your code############
        # you need to implement a Vanilla RNN for the decoder. Take a look at␣
↪the official documentation.
        # https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.
↪RNN
        self.max_length = 30  # the maximum length of a sentence, in case it's␣
↪trapped
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
        self.temp = 1

        # one-hot encoding + linear layer
        self.embed = nn.Embedding(self.vocab_size, emb_dim)
        # LSTM network
        self.LSTM = nn.LSTM(emb_dim, hidden_dim, num_layers)
        # output layer
        self.out = nn.Linear(hidden_dim, vocab_size)


    def forward(self, encode_features, captions, lengths):
        """
        Feed forward to generate captions.
        :param encode_features: output of encoder, size [N, emb_dim]
        :param captions: captions, size [N, max(lengths)]
```

```
        :param lengths: a list indicating valid length for each caption. length␣
↪is (batch_size).
        """
        ############Your Code##################
        # compute the embedding using one-hot technique and linear function
        caption_embeded = self.embed(captions)

        # concatenate the encoded features from encoder and embeddings
        encode_features = torch.cat((encode_features.unsqueeze(1),␣
↪caption_embeded), dim = 1)

        # feed into RNN.
        encode_features_pack = pack_padded_sequence(encode_features, lengths,␣
↪batch_first=True)

        # output layer
        output, hidden = self.LSTM(encode_features_pack)
        outputs = self.out(output[0])

        return outputs
```

13.3 Encoder-Decoder [5 pts]

[19]:
```
class LSTM(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, num_layers=1, dropout=0):
        """
        Encoder-decoder vanilla RNN.
        :param vocab_size: the size of Vocabulary.
        :param emb_dim: the dimensions of word embedding.
        :param hidden_dim: the dimensions of hidden units.
        :param num_layers: the number of RNN layers.
        """
        super(LSTM, self).__init__()

        ########Your Code################
        # Encoder: ResNet-50
        self.lstm_encoder = Encoder(emb_dim)
        # Decoder: LSTM
        self.lstm_decoder = Decoder(vocab_size, emb_dim, hidden_dim, num_layers,␣
↪dropout)
        self.max_length = self.lstm_decoder.max_length
        self.temp = 1
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x, captions, lengths):
        """
```

```
        Feed forward.
        :param x: Images, [N, 3, H, W]
        :param captions: encoded captions, [N, max(lengths)]
        :param lengths: a list indicating valid length for each caption. length␣
↪is (batch_size).
        :return: output logits, usually followed by a softmax layer.
        """
        #########Your code##################
        # forward passing
        encoder_features = self.lstm_encoder(x)
        x = self.lstm_decoder(encoder_features, captions, lengths)

        return x

    def sample_generate(self, x, states=None, mode='Deterministic',␣
↪temperature=5.0):
        """
        Generate samples during the evaluation.

        :param x: input image
        :param states: rnn states
        :param mode: which mode we use.
         - 'Deterministic': Take the maximum output at each step.
         - 'Stochastic': Sample from the probability distribution from the␣
↪output layer.
        :param temperature: will be used in the stochastic mode
        :return: sample_idxs. Word indices. We can use vocab to recover the␣
↪sentence.
        """
        sample_idxs = []
        ################Your Code#################
        # compute the encoded features
        in_feature = self.lstm_encoder.forward(x)
        in_feature = in_feature.unsqueeze(1)
        batch_size = in_feature.shape[0]

        for i in range(self.max_length):
          outputs, states = self.lstm_decoder.rnn(in_feature, states)
          outputs = outputs.squeeze(1)
          outputs = self.lstm_decoder.out(outputs)
          outputs = self.softmax(outputs/temperature)

        # decide which mode we use
          if mode == 'Deterministic':
              # take the maximum index after the softmax
              max_val, predicted = outputs.max(1)
              sample_idxs.append(predicted)
```

```
        elif mode == 'Stochastic':
            # sample from the probability distribution after the softmax
            # Hint: use torch.multinomial() to sample from a distribution.
            predicted = torch.multinomial(outputs, 1)
            sample_idxs.append(predicted)

        x = self.lstm_decoder.embed(predicted)
        x = x.unsqueeze(1)
    sample_idxs = torch.stack(sample_idxs, 1)
    sample_idxs = sample_idxs.squeeze()

    return sample_idxs
```

13.4 Section 3.1.2 Training [10 pts]

Use the same set of hyper-parameters (hidden units, optimizer, learning rate etc.) for both models.

[20]:
```
# some hyperparameters, you can change them
## training parameters
batch_size = 256
lr = 1e-2
num_epochs = 50
weight_decay = 0.0
log_step = 50

## network architecture
emb_dim = 1024
hidden_dim = 256
num_layers = 1 # number of RNN layers
dropout = 0.0

## image transformation
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #     transforms.RandomCrop(224, pad_if_needed=True),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))])

## Output directory
output_dir = path_to_homework + '/checkpoints/lstm/'
os.makedirs(output_dir, exist_ok=True)
```

[ ]:

```python
# Training code here

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


train_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/', split='train', vocab=vocab,
                               transform=transform, batch_size=batch_size, shuffle=True, num_workers=12)
val_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/', split='val', vocab=vocab,
                             transform=transform, batch_size=8, shuffle=True, num_workers=4)

model = LSTM(vocab_size=len(vocab), emb_dim=emb_dim, hidden_dim=hidden_dim,
                   num_layers=1, dropout=dropout).to(device)   # build a model

vocab_size=len(vocab)

# loss and optimizer
criterion = nn.CrossEntropyLoss().to(device)   # CE loss
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)  # optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=5,
                                            gamma=0.5)  # decay LR by a factor of 0.5 every 10 epochs. You can change this

# logs
Train_Losses = []   # record average training loss each epoch
Val_Losses = []    # record average validation loss each epoch
total_step = len(train_data_loader)   # number of iterations each epoch
best_val_loss = np.inf

# start training
print('Start training...')
import time
tic = time.time()
for epoch in range(num_epochs):
    print('Switch to training...')
    model.train()
    Train_loss_iter = []   # record the the training loss each iteration
    for itr, (images, captions, lengths) in enumerate(train_data_loader):
        #######Your Code###########
        model.zero_grad()
        images = Variable(images).to(device)
        captions = Variable(captions).to(device)
```

```
            predicted_cap_pack = pack_padded_sequence(captions, lengths,␣
↪batch_first=True)[0]
            predicted_cap = model(images, captions, lengths)
            predicted_cap = predicted_cap.view(-1, vocab_size)
            loss = criterion(input = predicted_cap, target=predicted_cap_pack)
            loss.backward()
            optimizer.step()

            # record the training loss
            Train_Losses.append(float(loss))

            # print log info
            if itr % log_step == 0:
                # print current loss and perplexity
                print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Perplexity: {:5.
↪4f}'
                      .format(epoch, num_epochs, itr, total_step, loss.item(),␣
↪np.exp(loss.item())))
        scheduler.step()
        Train_Losses.append(np.mean(Train_loss_iter))
        np.save(os.path.join(output_dir, 'TrainingLoss_lstm.npy'), Train_Losses)  #␣
↪save the training loss

        model.eval()
        # (optional) generate a sample during the training, you can use␣
↪deterministic mode
        # Your code


        # validation
        Val_Losses.append(val(model, val_data_loader, vocab))
        np.save(os.path.join(output_dir, 'ValLoss_lstm.npy'), Val_Losses) # save the␣
↪val loss

        # save model
        if Val_Losses[-1] < best_val_loss:
            best_val_loss = Val_Losses[-1]
            print('updated best val loss:', best_val_loss)
            print('Save model weights to...', output_dir)
            torch.save(model.state_dict(),
                       os.path.join(output_dir, 'lstm-best.pth'.format(epoch + 1,␣
↪itr + 1)))

print('It took: {} s'.format(time.time() - tic))
```

```
Start training...
Switch to training...
```

```
Epoch [0/50], Step [0/114], Loss: 9.2092, Perplexity: 9988.7733
Epoch [0/50], Step [50/114], Loss: 3.6335, Perplexity: 37.8462
Epoch [0/50], Step [100/114], Loss: 3.4302, Perplexity: 30.8838
Validating...

/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3335:
RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:161:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)

Step [0/127], Loss: 3.2063, Perplexity: 24.6869
Step [50/127], Loss: 3.3955, Perplexity: 29.8297
Step [100/127], Loss: 3.0332, Perplexity: 20.7628
updated best val loss: 3.3872082646437516
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [1/50], Step [0/114], Loss: 3.3987, Perplexity: 29.9261
Epoch [1/50], Step [50/114], Loss: 3.2961, Perplexity: 27.0077
Epoch [1/50], Step [100/114], Loss: 3.2551, Perplexity: 25.9223
Validating...
Step [0/127], Loss: 3.6364, Perplexity: 37.9546
Step [50/127], Loss: 3.6200, Perplexity: 37.3364
Step [100/127], Loss: 3.3697, Perplexity: 29.0694
updated best val loss: 3.2531456909780427
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [2/50], Step [0/114], Loss: 3.1212, Perplexity: 22.6726
Epoch [2/50], Step [50/114], Loss: 3.2186, Perplexity: 24.9925
Epoch [2/50], Step [100/114], Loss: 3.2312, Perplexity: 25.3089
Validating...
Step [0/127], Loss: 3.4916, Perplexity: 32.8397
Step [50/127], Loss: 2.8498, Perplexity: 17.2848
Step [100/127], Loss: 3.2790, Perplexity: 26.5484
updated best val loss: 3.2111755844176284
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [3/50], Step [0/114], Loss: 3.1079, Perplexity: 22.3742
Epoch [3/50], Step [50/114], Loss: 3.0588, Perplexity: 21.3021
Epoch [3/50], Step [100/114], Loss: 3.1670, Perplexity: 23.7362
Validating...
Step [0/127], Loss: 3.6116, Perplexity: 37.0271
Step [50/127], Loss: 3.1199, Perplexity: 22.6446
Step [100/127], Loss: 3.7624, Perplexity: 43.0537
updated best val loss: 3.182656032832589
```

```
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [4/50], Step [0/114], Loss: 3.0856, Perplexity: 21.8806
Epoch [4/50], Step [50/114], Loss: 3.0058, Perplexity: 20.2029
Epoch [4/50], Step [100/114], Loss: 3.0509, Perplexity: 21.1336
Validating...
Step [0/127], Loss: 3.3157, Perplexity: 27.5411
Step [50/127], Loss: 3.0505, Perplexity: 21.1266
Step [100/127], Loss: 4.0848, Perplexity: 59.4312
Switch to training...
Epoch [5/50], Step [0/114], Loss: 2.9088, Perplexity: 18.3355
Epoch [5/50], Step [50/114], Loss: 2.9563, Perplexity: 19.2261
Epoch [5/50], Step [100/114], Loss: 2.9170, Perplexity: 18.4861
Validating...
Step [0/127], Loss: 2.9846, Perplexity: 19.7778
Step [50/127], Loss: 3.3383, Perplexity: 28.1708
Step [100/127], Loss: 3.2997, Perplexity: 27.1052
updated best val loss: 3.1018434697248805
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [6/50], Step [0/114], Loss: 2.9192, Perplexity: 18.5269
Epoch [6/50], Step [50/114], Loss: 2.8708, Perplexity: 17.6518
Epoch [6/50], Step [100/114], Loss: 2.9233, Perplexity: 18.6022
Validating...
Step [0/127], Loss: 3.3216, Perplexity: 27.7057
Step [50/127], Loss: 2.9402, Perplexity: 18.9193
Step [100/127], Loss: 3.2715, Perplexity: 26.3507
updated best val loss: 3.0702739343868464
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [7/50], Step [0/114], Loss: 2.8481, Perplexity: 17.2543
Epoch [7/50], Step [50/114], Loss: 2.9175, Perplexity: 18.4947
Epoch [7/50], Step [100/114], Loss: 2.8678, Perplexity: 17.5988
Validating...
Step [0/127], Loss: 3.7062, Perplexity: 40.6993
Step [50/127], Loss: 3.4933, Perplexity: 32.8936
Step [100/127], Loss: 3.1970, Perplexity: 24.4583
Switch to training...
Epoch [8/50], Step [0/114], Loss: 2.9043, Perplexity: 18.2528
Epoch [8/50], Step [50/114], Loss: 2.7768, Perplexity: 16.0680
Epoch [8/50], Step [100/114], Loss: 2.8306, Perplexity: 16.9558
Validating...
Step [0/127], Loss: 2.9622, Perplexity: 19.3402
Step [50/127], Loss: 3.7252, Perplexity: 41.4795
Step [100/127], Loss: 3.0451, Perplexity: 21.0113
```

```
Switch to training...
Epoch [9/50], Step [0/114], Loss: 2.8373, Perplexity: 17.0701
Epoch [9/50], Step [50/114], Loss: 2.7605, Perplexity: 15.8081
Epoch [9/50], Step [100/114], Loss: 2.8063, Perplexity: 16.5490
Validating...
Step [0/127], Loss: 2.8764, Perplexity: 17.7495
Step [50/127], Loss: 3.4441, Perplexity: 31.3136
Step [100/127], Loss: 3.5390, Perplexity: 34.4314
Switch to training...
Epoch [10/50], Step [0/114], Loss: 2.7690, Perplexity: 15.9426
Epoch [10/50], Step [50/114], Loss: 2.7048, Perplexity: 14.9510
Epoch [10/50], Step [100/114], Loss: 2.7803, Perplexity: 16.1232
Validating...
Step [0/127], Loss: 2.8100, Perplexity: 16.6098
Step [50/127], Loss: 3.0468, Perplexity: 21.0481
Step [100/127], Loss: 2.6617, Perplexity: 14.3207
updated best val loss: 3.0513561748144196
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [11/50], Step [0/114], Loss: 2.7291, Perplexity: 15.3188
Epoch [11/50], Step [50/114], Loss: 2.7631, Perplexity: 15.8495
Epoch [11/50], Step [100/114], Loss: 2.7919, Perplexity: 16.3122
Validating...
Step [0/127], Loss: 2.5162, Perplexity: 12.3811
Step [50/127], Loss: 2.6523, Perplexity: 14.1873
Step [100/127], Loss: 3.1476, Perplexity: 23.2792
updated best val loss: 3.0223752795241947
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [12/50], Step [0/114], Loss: 2.7140, Perplexity: 15.0899
Epoch [12/50], Step [50/114], Loss: 2.6831, Perplexity: 14.6300
Epoch [12/50], Step [100/114], Loss: 2.7021, Perplexity: 14.9104
Validating...
Step [0/127], Loss: 4.0060, Perplexity: 54.9252
Step [50/127], Loss: 2.9865, Perplexity: 19.8162
Step [100/127], Loss: 2.7126, Perplexity: 15.0684
Switch to training...
Epoch [13/50], Step [0/114], Loss: 2.6712, Perplexity: 14.4579
Epoch [13/50], Step [50/114], Loss: 2.6996, Perplexity: 14.8741
Epoch [13/50], Step [100/114], Loss: 2.7723, Perplexity: 15.9951
Validating...
Step [0/127], Loss: 3.2097, Perplexity: 24.7706
Step [50/127], Loss: 2.7496, Perplexity: 15.6361
Step [100/127], Loss: 3.0757, Perplexity: 21.6656
Switch to training...
Epoch [14/50], Step [0/114], Loss: 2.6783, Perplexity: 14.5608
```

```
Epoch [14/50], Step [50/114], Loss: 2.6717, Perplexity: 14.4641
Epoch [14/50], Step [100/114], Loss: 2.6644, Perplexity: 14.3599
Validating...
Step [0/127], Loss: 3.3568, Perplexity: 28.6965
Step [50/127], Loss: 3.2253, Perplexity: 25.1610
Step [100/127], Loss: 3.7404, Perplexity: 42.1160
Switch to training...
Epoch [15/50], Step [0/114], Loss: 2.6977, Perplexity: 14.8462
Epoch [15/50], Step [50/114], Loss: 2.6689, Perplexity: 14.4247
Epoch [15/50], Step [100/114], Loss: 2.6214, Perplexity: 13.7552
Validating...
Step [0/127], Loss: 3.5084, Perplexity: 33.3938
Step [50/127], Loss: 2.4390, Perplexity: 11.4616
Step [100/127], Loss: 3.0438, Perplexity: 20.9849
Switch to training...
Epoch [16/50], Step [0/114], Loss: 2.6392, Perplexity: 14.0018
Epoch [16/50], Step [50/114], Loss: 2.5708, Perplexity: 13.0759
Epoch [16/50], Step [100/114], Loss: 2.6436, Perplexity: 14.0642
Validating...
Step [0/127], Loss: 3.0589, Perplexity: 21.3043
Step [50/127], Loss: 3.1339, Perplexity: 22.9625
Step [100/127], Loss: 2.7640, Perplexity: 15.8631
updated best val loss: 3.0058976946853275
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [17/50], Step [0/114], Loss: 2.6318, Perplexity: 13.8986
Epoch [17/50], Step [50/114], Loss: 2.6380, Perplexity: 13.9859
Epoch [17/50], Step [100/114], Loss: 2.6412, Perplexity: 14.0297
Validating...
Step [0/127], Loss: 3.0537, Perplexity: 21.1934
Step [50/127], Loss: 3.0606, Perplexity: 21.3414
Step [100/127], Loss: 3.4126, Perplexity: 30.3429
Switch to training...
Epoch [18/50], Step [0/114], Loss: 2.5540, Perplexity: 12.8584
Epoch [18/50], Step [50/114], Loss: 2.6107, Perplexity: 13.6085
Epoch [18/50], Step [100/114], Loss: 2.6359, Perplexity: 13.9558
Validating...
Step [0/127], Loss: 3.1912, Perplexity: 24.3181
Step [50/127], Loss: 2.2678, Perplexity: 9.6578
Step [100/127], Loss: 2.7417, Perplexity: 15.5133
updated best val loss: 2.983503542547151
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [19/50], Step [0/114], Loss: 2.5924, Perplexity: 13.3615
Epoch [19/50], Step [50/114], Loss: 2.6096, Perplexity: 13.5934
Epoch [19/50], Step [100/114], Loss: 2.6161, Perplexity: 13.6825
```

```
Validating...
Step [0/127], Loss: 3.2313, Perplexity: 25.3130
Step [50/127], Loss: 2.5452, Perplexity: 12.7460
Step [100/127], Loss: 2.6936, Perplexity: 14.7842
Switch to training...
Epoch [20/50], Step [0/114], Loss: 2.5843, Perplexity: 13.2536
Epoch [20/50], Step [50/114], Loss: 2.5379, Perplexity: 12.6534
Epoch [20/50], Step [100/114], Loss: 2.5176, Perplexity: 12.3989
Validating...
Step [0/127], Loss: 2.9751, Perplexity: 19.5912
Step [50/127], Loss: 3.1422, Perplexity: 23.1538
Step [100/127], Loss: 3.4476, Perplexity: 31.4255
Switch to training...
Epoch [21/50], Step [0/114], Loss: 2.6178, Perplexity: 13.7061
Epoch [21/50], Step [50/114], Loss: 2.5632, Perplexity: 12.9774
Epoch [21/50], Step [100/114], Loss: 2.5819, Perplexity: 13.2225
Validating...
Step [0/127], Loss: 3.0761, Perplexity: 21.6736
Step [50/127], Loss: 3.1204, Perplexity: 22.6560
Step [100/127], Loss: 3.2029, Perplexity: 24.6048
Switch to training...
Epoch [22/50], Step [0/114], Loss: 2.5458, Perplexity: 12.7540
Epoch [22/50], Step [50/114], Loss: 2.5651, Perplexity: 13.0015
Epoch [22/50], Step [100/114], Loss: 2.5611, Perplexity: 12.9506
Validating...
Step [0/127], Loss: 3.7708, Perplexity: 43.4148
Step [50/127], Loss: 3.3769, Perplexity: 29.2794
Step [100/127], Loss: 3.0838, Perplexity: 21.8421
Switch to training...
Epoch [23/50], Step [0/114], Loss: 2.5454, Perplexity: 12.7489
Epoch [23/50], Step [50/114], Loss: 2.5369, Perplexity: 12.6400
Epoch [23/50], Step [100/114], Loss: 2.6144, Perplexity: 13.6585
Validating...
Step [0/127], Loss: 2.9872, Perplexity: 19.8300
Step [50/127], Loss: 3.1717, Perplexity: 23.8490
Step [100/127], Loss: 2.8599, Perplexity: 17.4605
updated best val loss: 2.983343030524066
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/lstm/
Switch to training...
Epoch [24/50], Step [0/114], Loss: 2.5215, Perplexity: 12.4475
Epoch [24/50], Step [50/114], Loss: 2.5822, Perplexity: 13.2268
Epoch [24/50], Step [100/114], Loss: 2.4814, Perplexity: 11.9575
Validating...
Step [0/127], Loss: 3.0142, Perplexity: 20.3719
Step [50/127], Loss: 2.6591, Perplexity: 14.2830
Step [100/127], Loss: 2.8667, Perplexity: 17.5794
Switch to training...
```

```
Epoch [25/50], Step [0/114], Loss: 2.5205, Perplexity: 12.4354
Epoch [25/50], Step [50/114], Loss: 2.5571, Perplexity: 12.8979
Epoch [25/50], Step [100/114], Loss: 2.5156, Perplexity: 12.3736
Validating...
Step [0/127], Loss: 2.7126, Perplexity: 15.0691
Step [50/127], Loss: 3.1860, Perplexity: 24.1922
Step [100/127], Loss: 2.6225, Perplexity: 13.7698
Switch to training...
Epoch [26/50], Step [0/114], Loss: 2.5530, Perplexity: 12.8459
Epoch [26/50], Step [50/114], Loss: 2.5031, Perplexity: 12.2197
Epoch [26/50], Step [100/114], Loss: 2.4986, Perplexity: 12.1653
Validating...
Step [0/127], Loss: 3.2609, Perplexity: 26.0735
Step [50/127], Loss: 3.2875, Perplexity: 26.7750
Step [100/127], Loss: 3.5123, Perplexity: 33.5251
Switch to training...
Epoch [27/50], Step [0/114], Loss: 2.5535, Perplexity: 12.8517
Epoch [27/50], Step [50/114], Loss: 2.5071, Perplexity: 12.2692
Epoch [27/50], Step [100/114], Loss: 2.5041, Perplexity: 12.2327
Validating...
Step [0/127], Loss: 3.0362, Perplexity: 20.8269
Step [50/127], Loss: 3.5058, Perplexity: 33.3072
Step [100/127], Loss: 2.7043, Perplexity: 14.9443
Switch to training...
Epoch [28/50], Step [0/114], Loss: 2.5539, Perplexity: 12.8576
Epoch [28/50], Step [50/114], Loss: 2.4982, Perplexity: 12.1606
Epoch [28/50], Step [100/114], Loss: 2.5779, Perplexity: 13.1696
Validating...
Step [0/127], Loss: 2.6160, Perplexity: 13.6811
Step [50/127], Loss: 3.1596, Perplexity: 23.5613
Step [100/127], Loss: 3.1996, Perplexity: 24.5224
Switch to training...
Epoch [29/50], Step [0/114], Loss: 2.5220, Perplexity: 12.4540
Epoch [29/50], Step [50/114], Loss: 2.4859, Perplexity: 12.0114
Epoch [29/50], Step [100/114], Loss: 2.5433, Perplexity: 12.7219
Validating...
Step [0/127], Loss: 3.0019, Perplexity: 20.1231
Step [50/127], Loss: 2.7442, Perplexity: 15.5528
Step [100/127], Loss: 3.3640, Perplexity: 28.9038
Switch to training...
Epoch [30/50], Step [0/114], Loss: 2.5744, Perplexity: 13.1239
Epoch [30/50], Step [50/114], Loss: 2.5333, Perplexity: 12.5948
Epoch [30/50], Step [100/114], Loss: 2.5483, Perplexity: 12.7856
Validating...
Step [0/127], Loss: 3.3189, Perplexity: 27.6290
Step [50/127], Loss: 3.2925, Perplexity: 26.9112
Step [100/127], Loss: 2.9456, Perplexity: 19.0226
Switch to training...
```

```
Epoch [31/50], Step [0/114], Loss: 2.6138, Perplexity: 13.6510
Epoch [31/50], Step [50/114], Loss: 2.5185, Perplexity: 12.4096
Epoch [31/50], Step [100/114], Loss: 2.5460, Perplexity: 12.7563
Validating...
Step [0/127], Loss: 2.4137, Perplexity: 11.1747
Step [50/127], Loss: 3.0566, Perplexity: 21.2553
Step [100/127], Loss: 3.1902, Perplexity: 24.2939
Switch to training...
Epoch [32/50], Step [0/114], Loss: 2.5405, Perplexity: 12.6857
Epoch [32/50], Step [50/114], Loss: 2.5056, Perplexity: 12.2505
Epoch [32/50], Step [100/114], Loss: 2.5094, Perplexity: 12.2976
Validating...
Step [0/127], Loss: 3.2712, Perplexity: 26.3431
Step [50/127], Loss: 3.2343, Perplexity: 25.3874
Step [100/127], Loss: 3.3417, Perplexity: 28.2672
Switch to training...
Epoch [33/50], Step [0/114], Loss: 2.4663, Perplexity: 11.7788
Epoch [33/50], Step [50/114], Loss: 2.5156, Perplexity: 12.3735
Epoch [33/50], Step [100/114], Loss: 2.5652, Perplexity: 13.0030
Validating...
Step [0/127], Loss: 2.8283, Perplexity: 16.9172
Step [50/127], Loss: 2.6631, Perplexity: 14.3411
Step [100/127], Loss: 3.5133, Perplexity: 33.5590
Switch to training...
Epoch [34/50], Step [0/114], Loss: 2.4987, Perplexity: 12.1671
Epoch [34/50], Step [50/114], Loss: 2.5707, Perplexity: 13.0747
Epoch [34/50], Step [100/114], Loss: 2.5569, Perplexity: 12.8959
Validating...
Step [0/127], Loss: 3.0982, Perplexity: 22.1587
Step [50/127], Loss: 2.3166, Perplexity: 10.1416
Step [100/127], Loss: 2.9464, Perplexity: 19.0381
Switch to training...
Epoch [35/50], Step [0/114], Loss: 2.5227, Perplexity: 12.4619
Epoch [35/50], Step [50/114], Loss: 2.5738, Perplexity: 13.1158
Epoch [35/50], Step [100/114], Loss: 2.5375, Perplexity: 12.6480
Validating...
Step [0/127], Loss: 3.4064, Perplexity: 30.1555
Step [50/127], Loss: 2.8421, Perplexity: 17.1519
Step [100/127], Loss: 3.3082, Perplexity: 27.3356
Switch to training...
Epoch [36/50], Step [0/114], Loss: 2.4501, Perplexity: 11.5897
Epoch [36/50], Step [50/114], Loss: 2.4917, Perplexity: 12.0812
Epoch [36/50], Step [100/114], Loss: 2.5452, Perplexity: 12.7453
Validating...
Step [0/127], Loss: 2.9687, Perplexity: 19.4657
Step [50/127], Loss: 2.8037, Perplexity: 16.5049
Step [100/127], Loss: 2.8883, Perplexity: 17.9627
Switch to training...
```

```
Epoch [37/50], Step [0/114], Loss: 2.5329, Perplexity: 12.5901
Epoch [37/50], Step [50/114], Loss: 2.5315, Perplexity: 12.5717
Epoch [37/50], Step [100/114], Loss: 2.4917, Perplexity: 12.0815
Validating...
Step [0/127], Loss: 2.7629, Perplexity: 15.8464
Step [50/127], Loss: 2.9824, Perplexity: 19.7358
Step [100/127], Loss: 3.2267, Perplexity: 25.1963
Switch to training...
Epoch [38/50], Step [0/114], Loss: 2.5105, Perplexity: 12.3113
Epoch [38/50], Step [50/114], Loss: 2.5200, Perplexity: 12.4287
Epoch [38/50], Step [100/114], Loss: 2.5904, Perplexity: 13.3353
Validating...
Step [0/127], Loss: 3.1892, Perplexity: 24.2698
Step [50/127], Loss: 3.1871, Perplexity: 24.2178
Step [100/127], Loss: 3.2751, Perplexity: 26.4462
Switch to training...
Epoch [39/50], Step [0/114], Loss: 2.5328, Perplexity: 12.5883
Epoch [39/50], Step [50/114], Loss: 2.5508, Perplexity: 12.8173
Epoch [39/50], Step [100/114], Loss: 2.5119, Perplexity: 12.3284
Validating...
Step [0/127], Loss: 3.4229, Perplexity: 30.6579
Step [50/127], Loss: 3.0678, Perplexity: 21.4945
Step [100/127], Loss: 3.3362, Perplexity: 28.1129
Switch to training...
Epoch [40/50], Step [0/114], Loss: 2.5090, Perplexity: 12.2931
Epoch [40/50], Step [50/114], Loss: 2.4719, Perplexity: 11.8447
Epoch [40/50], Step [100/114], Loss: 2.4979, Perplexity: 12.1570
Validating...
Step [0/127], Loss: 2.9359, Perplexity: 18.8392
Step [50/127], Loss: 3.7445, Perplexity: 42.2876
Step [100/127], Loss: 3.6316, Perplexity: 37.7740
Switch to training...
Epoch [41/50], Step [0/114], Loss: 2.5570, Perplexity: 12.8977
Epoch [41/50], Step [50/114], Loss: 2.5487, Perplexity: 12.7902
Epoch [41/50], Step [100/114], Loss: 2.4495, Perplexity: 11.5822
Validating...
Step [0/127], Loss: 2.3570, Perplexity: 10.5588
Step [50/127], Loss: 2.8187, Perplexity: 16.7556
Step [100/127], Loss: 3.6568, Perplexity: 38.7383
Switch to training...
Epoch [42/50], Step [0/114], Loss: 2.6121, Perplexity: 13.6282
Epoch [42/50], Step [50/114], Loss: 2.5222, Perplexity: 12.4561
Epoch [42/50], Step [100/114], Loss: 2.4684, Perplexity: 11.8038
Validating...
Step [0/127], Loss: 3.2650, Perplexity: 26.1799
Step [50/127], Loss: 2.7989, Perplexity: 16.4267
Step [100/127], Loss: 3.4870, Perplexity: 32.6885
Switch to training...
```

```
Epoch [43/50], Step [0/114], Loss: 2.5799, Perplexity: 13.1961
Epoch [43/50], Step [50/114], Loss: 2.4939, Perplexity: 12.1084
Epoch [43/50], Step [100/114], Loss: 2.5316, Perplexity: 12.5734
Validating...
Step [0/127], Loss: 3.1979, Perplexity: 24.4809
Step [50/127], Loss: 3.1743, Perplexity: 23.9109
Step [100/127], Loss: 3.1573, Perplexity: 23.5081
Switch to training...
Epoch [44/50], Step [0/114], Loss: 2.4935, Perplexity: 12.1037
Epoch [44/50], Step [50/114], Loss: 2.4493, Perplexity: 11.5798
Epoch [44/50], Step [100/114], Loss: 2.5211, Perplexity: 12.4422
Validating...
Step [0/127], Loss: 2.9847, Perplexity: 19.7800
Step [50/127], Loss: 3.0636, Perplexity: 21.4040
Step [100/127], Loss: 3.5115, Perplexity: 33.4978
Switch to training...
Epoch [45/50], Step [0/114], Loss: 2.5255, Perplexity: 12.4976
Epoch [45/50], Step [50/114], Loss: 2.5100, Perplexity: 12.3049
Epoch [45/50], Step [100/114], Loss: 2.5052, Perplexity: 12.2461
Validating...
Step [0/127], Loss: 3.1976, Perplexity: 24.4740
Step [50/127], Loss: 3.2894, Perplexity: 26.8267
Step [100/127], Loss: 2.5607, Perplexity: 12.9447
Switch to training...
Epoch [46/50], Step [0/114], Loss: 2.5380, Perplexity: 12.6541
Epoch [46/50], Step [50/114], Loss: 2.5248, Perplexity: 12.4882
Epoch [46/50], Step [100/114], Loss: 2.5729, Perplexity: 13.1038
Validating...
Step [0/127], Loss: 3.6066, Perplexity: 36.8405
Step [50/127], Loss: 2.6696, Perplexity: 14.4342
Step [100/127], Loss: 3.0579, Perplexity: 21.2820
Switch to training...
Epoch [47/50], Step [0/114], Loss: 2.4714, Perplexity: 11.8393
Epoch [47/50], Step [50/114], Loss: 2.5356, Perplexity: 12.6240
Epoch [47/50], Step [100/114], Loss: 2.5347, Perplexity: 12.6121
Validating...
Step [0/127], Loss: 2.8675, Perplexity: 17.5925
Step [50/127], Loss: 3.0249, Perplexity: 20.5913
Step [100/127], Loss: 2.8748, Perplexity: 17.7220
Switch to training...
Epoch [48/50], Step [0/114], Loss: 2.4894, Perplexity: 12.0543
Epoch [48/50], Step [50/114], Loss: 2.5115, Perplexity: 12.3238
Epoch [48/50], Step [100/114], Loss: 2.5197, Perplexity: 12.4246
Validating...
Step [0/127], Loss: 3.1014, Perplexity: 22.2296
Step [50/127], Loss: 3.2681, Perplexity: 26.2616
Step [100/127], Loss: 2.8346, Perplexity: 17.0233
Switch to training...
```

```
Epoch [49/50], Step [0/114], Loss: 2.4606, Perplexity: 11.7118
Epoch [49/50], Step [50/114], Loss: 2.4959, Perplexity: 12.1329
Epoch [49/50], Step [100/114], Loss: 2.5045, Perplexity: 12.2377
Validating...
Step [0/127], Loss: 3.1538, Perplexity: 23.4241
Step [50/127], Loss: 3.4654, Perplexity: 31.9896
Step [100/127], Loss: 2.8784, Perplexity: 17.7861
It took: 13621.590710639954 s
```

13.5 Section 3.1.3 Evalution [10 pts]

Evaluate your model on the test set by perplexity score or BLEU score

[21]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')



train_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='train', vocab=vocab,
                               transform=transform, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=12)
val_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='val', vocab=vocab,
                             transform=transform, batch_size=8, shuffle=True,␣
 ↪num_workers=4)

model = LSTM(vocab_size=len(vocab), emb_dim=emb_dim, hidden_dim=hidden_dim,
                   num_layers=1, dropout=dropout).to(device)  # build a model

vocab_size=len(vocab)

# loss and optimizer
criterion = nn.CrossEntropyLoss().to(device)  # CE loss
optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
 ↪weight_decay=weight_decay)  # optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=5,
                                            gamma=0.5)  # decay LR by a factor of 0.5␣
 ↪every 10 epochs. You can change this
```

[22]:
```
## Evaluate your model using BLEU score. Use Deterministic mode.
# Your code here
## Image transformation
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #     transforms.RandomCrop(224, pad_if_needed=True),
        transforms.RandomHorizontalFlip(),
```

```python
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))])

## Evaluate your model using BLEU score. Use Deterministic mode
model = LSTM(vocab_size=len(vocab), emb_dim=emb_dim, hidden_dim=hidden_dim,
                   num_layers=1, dropout=dropout).to(device)  # build a model
model.load_state_dict(torch.load(path_to_homework + '/checkpoints/lstm/lstm-best.
 ↪pth', map_location=torch.device('cpu')))
model.eval()
bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic')
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))
# End of code
```

```
  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

  1%|          | 1/125 [00:00<01:09,  1.80it/s]
  2%|         | 2/125 [00:00<00:56,  2.18it/s]
  3%|         | 4/125 [00:00<00:42,  2.86it/s]
  4%|         | 5/125 [00:01<00:34,  3.52it/s]
  5%|         | 6/125 [00:01<00:29,  4.06it/s]
  6%|         | 7/125 [00:01<00:24,  4.78it/s]
  7%|         | 9/125 [00:01<00:20,  5.59it/s]
  8%|         | 10/125 [00:01<00:17,  6.41it/s]
  9%|         | 11/125 [00:01<00:16,  6.93it/s]
 10%|         | 13/125 [00:01<00:13,  8.02it/s]
 12%|        | 15/125 [00:02<00:12,  8.81it/s]
 14%|        | 17/125 [00:02<00:12,  8.45it/s]
 15%|        | 19/125 [00:02<00:12,  8.81it/s]
 17%|        | 21/125 [00:02<00:11,  9.28it/s]
 18%|        | 23/125 [00:03<00:10,  9.35it/s]
 20%|        | 25/125 [00:03<00:10,  9.67it/s]
 21%|        | 26/125 [00:03<00:11,  8.73it/s]
 22%|       | 28/125 [00:03<00:09,  9.89it/s]
 24%|       | 30/125 [00:03<00:09,  9.88it/s]
 26%|       | 32/125 [00:03<00:09,  9.94it/s]
 27%|       | 34/125 [00:04<00:09,  9.81it/s]
 29%|       | 36/125 [00:04<00:08, 10.37it/s]
 30%|       | 38/125 [00:04<00:09,  9.52it/s]
 32%|      | 40/125 [00:04<00:08,  9.70it/s]
 33%|      | 41/125 [00:04<00:09,  9.16it/s]
 34%|      | 43/125 [00:04<00:07, 10.30it/s]
 36%|      | 45/125 [00:05<00:07, 10.37it/s]
```

```
 38%|      | 47/125 [00:05<00:07, 10.30it/s]
 39%|      | 49/125 [00:05<00:07, 10.32it/s]
 41%|      | 51/125 [00:05<00:07,  9.79it/s]
 42%|     | 53/125 [00:05<00:06, 10.82it/s]
 44%|     | 55/125 [00:06<00:06, 10.41it/s]
 46%|     | 57/125 [00:06<00:06, 11.09it/s]
 47%|     | 59/125 [00:06<00:06, 10.28it/s]
 49%|     | 61/125 [00:06<00:05, 11.03it/s]
 50%|     | 63/125 [00:06<00:06,  9.92it/s]
 52%|    | 65/125 [00:07<00:05, 10.15it/s]
 54%|    | 67/125 [00:07<00:05,  9.80it/s]
 55%|    | 69/125 [00:07<00:05, 10.37it/s]
 57%|    | 71/125 [00:07<00:05, 10.03it/s]
 58%|    | 73/125 [00:07<00:05, 10.34it/s]
 60%|    | 75/125 [00:08<00:05,  9.86it/s]
 62%|   | 77/125 [00:08<00:04, 10.37it/s]
 63%|   | 79/125 [00:08<00:04,  9.97it/s]
 65%|   | 81/125 [00:08<00:04,  9.69it/s]
 66%|   | 82/125 [00:08<00:04,  9.61it/s]
 67%|   | 84/125 [00:09<00:04,  9.98it/s]
 69%|   | 86/125 [00:09<00:03, 10.36it/s]
 70%|   | 88/125 [00:09<00:03, 10.20it/s]
 72%|  | 90/125 [00:09<00:03, 10.36it/s]
 74%|  | 92/125 [00:09<00:03,  9.96it/s]
 75%|  | 94/125 [00:09<00:02, 10.95it/s]
 77%|  | 96/125 [00:10<00:03,  9.24it/s]
 78%|  | 97/125 [00:10<00:03,  8.93it/s]
 79%|  | 99/125 [00:10<00:02, 10.08it/s]
 81%|  | 101/125 [00:10<00:02, 10.16it/s]
 82%| | 103/125 [00:10<00:02,  9.64it/s]
 84%| | 105/125 [00:11<00:02,  9.39it/s]
 86%| | 107/125 [00:11<00:01,  9.83it/s]
 87%| | 109/125 [00:11<00:01,  9.99it/s]
 89%| | 111/125 [00:11<00:01, 10.10it/s]
 90%| | 113/125 [00:11<00:01, 10.66it/s]
 92%|| 115/125 [00:12<00:00, 11.11it/s]
 94%|| 117/125 [00:12<00:00,  9.89it/s]
 95%|| 119/125 [00:12<00:00, 11.22it/s]
 97%|| 121/125 [00:12<00:00, 12.83it/s]
100%|| 125/125 [00:12<00:00,  9.76it/s]

  0%|          | 0/1000 [00:00<?, ?it/s]
  3%|         | 29/1000 [00:00<00:03, 289.61it/s]

Computing BLEU


  6%|         | 57/1000 [00:00<00:03, 285.33it/s]
  8%|         | 84/1000 [00:00<00:03, 278.51it/s]
```

```
 11%|         | 113/1000 [00:00<00:03, 279.99it/s]
 14%|         | 142/1000 [00:00<00:03, 282.40it/s]
 17%|         | 171/1000 [00:00<00:02, 284.04it/s]
 20%|         | 201/1000 [00:00<00:02, 285.61it/s]
 23%|        | 230/1000 [00:00<00:02, 286.31it/s]
 26%|        | 260/1000 [00:00<00:02, 289.26it/s]
 29%|        | 288/1000 [00:01<00:02, 281.92it/s]
 32%|       | 316/1000 [00:01<00:02, 280.52it/s]
 34%|       | 344/1000 [00:01<00:02, 280.35it/s]
 37%|       | 372/1000 [00:01<00:02, 280.07it/s]
 40%|       | 402/1000 [00:01<00:02, 284.48it/s]
 43%|      | 432/1000 [00:01<00:01, 288.61it/s]
 46%|      | 461/1000 [00:01<00:01, 287.35it/s]
 49%|      | 491/1000 [00:01<00:01, 289.24it/s]
 52%|     | 520/1000 [00:01<00:01, 271.71it/s]
 55%|     | 548/1000 [00:01<00:01, 272.44it/s]
 58%|     | 576/1000 [00:02<00:01, 274.08it/s]
 60%|     | 604/1000 [00:02<00:01, 272.48it/s]
 63%|    | 632/1000 [00:02<00:01, 271.35it/s]
 66%|    | 660/1000 [00:02<00:01, 269.76it/s]
 69%|    | 688/1000 [00:02<00:01, 269.72it/s]
 72%|   | 715/1000 [00:02<00:01, 267.82it/s]
 74%|   | 743/1000 [00:02<00:00, 270.93it/s]
 77%|   | 771/1000 [00:02<00:00, 273.12it/s]
 80%|   | 799/1000 [00:02<00:00, 270.34it/s]
 83%|  | 828/1000 [00:02<00:00, 272.32it/s]
 86%|  | 856/1000 [00:03<00:00, 268.39it/s]
 88%|  | 883/1000 [00:03<00:00, 262.02it/s]
 91%|  | 910/1000 [00:03<00:00, 260.18it/s]
 94%|| 937/1000 [00:03<00:00, 259.80it/s]
 96%|| 965/1000 [00:03<00:00, 264.35it/s]
100%|| 1000/1000 [00:03<00:00, 274.07it/s]
```

```
BLEU 1:43.43058246704518, BLEU 2:32.799554560543, BLEU 3:19.94015934517099, BLEU
4:13.527134409104775
```

[ ]:

```python
## Use at least 3 different temperatures to generate captions on the test set.␣
 ↪Report the BLEU scores.
# Your code here
bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
 ↪temperature=0.5)
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
 ↪temperature=1.0)
```

```
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic',␣
 ↪temperature=2.0)
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))

# End of code
```

```
  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00,  9.86it/s]
  2%|          | 25/1000 [00:00<00:03, 244.64it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 285.80it/s]

BLEU 1:43.43058246704518, BLEU 2:32.799554560543, BLEU 3:19.94015934517099, BLEU
4:13.527134409104775

  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00,  9.86it/s]
  3%|          | 29/1000 [00:00<00:03, 289.80it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 289.54it/s]

BLEU 1:43.43058246704518, BLEU 2:32.799554560543, BLEU 3:19.94015934517099, BLEU
4:13.527134409104775

  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

100%|| 125/125 [00:12<00:00,  9.93it/s]
  3%|          | 29/1000 [00:00<00:03, 285.22it/s]

Computing BLEU

100%|| 1000/1000 [00:03<00:00, 287.14it/s]

BLEU 1:43.43058246704518, BLEU 2:32.799554560543, BLEU 3:19.94015934517099, BLEU
4:13.527134409104775
```

13.6 Section 3.1.4 Discussion [5 pts]

What's the difference between Vanilla RNN and LSTM (training loss, evaluation results, etc)?

Your comments:
Training loss: The training loss of Vanilla RNN are slightly larger than LSTM's.

Evaluation results: The evaluation result of Vanilla RNN is better than LSTM's.

Running time: The running time of Vanilla RNN is slightly less than LSTM's.

13.7 Section 3.2 Using pre-trained word embeddings [20 pts]

For now, the decoder uses a word as input by converting it into a fixed size embedding, and our networks learn these word embeddings by training. In this experiment, you will use pre-trained word embeddings like Word2Vec or GloVe in LSTM. If you use Pytorch's nn.Embedding layer, you can initialize its weights with a matrix containing pre-trained word embeddings for all words in your vocabulary, and freeze the weights (i.e. don't train this layer). You can find these embeddings online.

Some resources: - GloVe: https://nlp.stanford.edu/projects/glove/ - Word2Vec: http://jalammar.github.io/illustrated-word2vec/

In case you don't know how to get one, we've already provided a light GloVe embedding: wm_06.npy, which can produce 300-d word embeddings.

13.8 Section 3.2.1 Encoder-decoder [10 pts]

[23]:
```
class Decoder(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, pretrained_emb,␣
↪num_layers=1, dropout=0):
        """
        Use LSTM as decoder for captions.
        :param emb_dim: Embedding dimensions.
        :param hidden_dim: Hidden states dimensions.
        :param pretrained_emb: the path to the pretrained embedding
        :param num_layers: Number of LSTM layers.
        :param vocab_size: The size of Vocabulary.
        :param dropout: dropout probability
        """
        super(Decoder, self).__init__()
        ############Your code############
        # you need to implement a Vanilla RNN for the decoder. Take a look at␣
↪the official documentation.
        # https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.
↪RNN
        self.max_length = 30  # the maximum length of a sentence, in case it's␣
↪trapped
        self.hidden_dim = hidden_dim
        self.vocab_size = vocab_size
```

```
        self.temp = 1

        # one-hot encoding + linear layer
        self.embed = nn.Embedding(self.vocab_size, emb_dim)
        self.embed.weight.requires_grad = False
        # LSTM network
        self.lstm = nn.LSTM(emb_dim, hidden_dim, num_layers)
        # output layer
        self.out = nn.Linear(hidden_dim, vocab_size)

    def forward(self, encode_features, captions, lengths):
        """
        Feed forward to generate captions.
        :param encode_features: output of encoder, size [N, emb_dim]
        :param captions: captions, size [N, max(lengths)]
        :param lengths: a list indicating valid length for each caption. length␣
↪is (batch_size).
        """
        ############Your Code##################
        # compute the embedding using one-hot technique and linear function
        caption_embeded = self.embed(captions)
        # concatenate the encoded features from encoder and embeddings
        encode_features = torch.cat((encode_features.unsqueeze(1),␣
↪caption_embeded), dim=1)
        packed = pack_padded_sequence(encode_features, lengths, batch_first=True)
        # feed into RNN
        output, hidden = self.lstm(packed)
        # output layer
        outputs = self.fc(output[0])

        return outputs
```

[24]:
```
class Word_embeddings(nn.Module):
    def __init__(self, vocab_size, emb_dim, hidden_dim, pretrained_emb,␣
↪num_layers=1, dropout=0):
        """
        Encoder-decoder baseline.
        :param vocab_size: the size of Vocabulary.
        :param emb_dim: the dimensions of word embedding.
        :param hidden_dim: the dimensions of hidden units.
        :param pretrained_emb: the path to the pretrained embedding
        :param num_layers: the number of LSTM layers.
        :param dropout: dropout probability.
        """
        super(Word_embeddings, self).__init__()
        # self.max_length = self.decoder.max_length
        ########Your Code###############
```

```
        # Encoder: ResNet-50
        self.lstm_encoder = Encoder(emb_dim)
        # Decoder: LSTM
        self.lstm_decoder = Decoder(vocab_size, emb_dim, hidden_dim, num_layers,␣
↪dropout)
        self.max_length = self.lstm_decoder.max_length
        self.temp = 1
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x, captions, lengths):
        """
        Feed forward.
        :param x: Images, [N, 3, H, W]
        :param captions: encoded captions, [N, max(lengths)]
        :param lengths: a list indicating valid length for each caption. length␣
↪is (batch_size).
        :return: output logits, usually followed by a softmax layer.
        """
        #########Your code##################
        # forward passing
        encoder_features = self.lstm_encoder(x)
        x = self.lstm_decoder(encoder_features, captions, lengths)

        return x

    def sample_generate(self, x, states=None, mode='Deterministic',␣
↪temperature=5.0):
        """
        Generate samples.
        :param x:
        :return:
        """
        sample_idxs = []
        ################Your Code#################
        # compute the encoded features
        in_feature = self.lstm_encoder.forward(x)
        in_feature = in_feature.unsqueeze(1)
        batch_size = in_feature.shape[0]

        for i in range(self.max_length):
          outputs, states = self.lstm_decoder.rnn(in_feature, states)
          outputs = outputs.squeeze(1)
          outputs = self.lstm_decoder.out(outputs)
          outputs = self.softmax(outputs/temperature)

        # decide which mode we use
          if mode == 'Deterministic':
```

```
                # take the maximum index after the softmax
                max_val, predicted = outputs.max(1)
                sample_idxs.append(predicted)

            elif mode == 'Stochastic':
                # sample from the probability distribution after the softmax
                # Hint: use torch.multinomial() to sample from a distribution.
                predicted = torch.multinomial(outputs, 1)
                sample_idxs.append(predicted)

            x = self.lstm_decoder.embed(predicted)
            x = x.unsqueeze(1)
        sample_idxs = torch.stack(sample_idxs, 1)
        sample_idxs = sample_idxs.squeeze()

        # return sample_idxs
```

13.9 Section 3.2.2 Training [5 pts]

[25]:
```
# some hyperparameters, you can change them
## training parameters
batch_size = 256
lr = 1e-2
num_epochs = 50
weight_decay = 0.0
log_step = 50

## network architecture
emb_dim = 300
hidden_dim = 256
num_layers = 1 # number of RNN layers
dropout = 0.0

## image transformation
transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        #     transforms.RandomCrop(224, pad_if_needed=True),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=(0.485, 0.456, 0.406),
                             std=(0.229, 0.224, 0.225))])

## Output directory
output_dir = path_to_homework + '/checkpoints/pretrained_emb/'
os.makedirs(output_dir, exist_ok=True)
```

```
[26]: # Training code here
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


train_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='train', vocab=vocab,
                              transform=transform, batch_size=batch_size,␣
 ↪shuffle=True, num_workers=12)
val_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',␣
 ↪split='val', vocab=vocab,
                            transform=transform, batch_size=8, shuffle=True,␣
 ↪num_workers=4)

# pretrained embedding weights
pre_emb_path = '/content/drive/My Drive/DL_Fall_2020/Assignment_4/wm_06.npy'  #␣
 ↪type the path to the pretrained embedding you find

model = Word_embeddings(vocab_size=len(vocab), emb_dim=emb_dim,␣
 ↪hidden_dim=hidden_dim, pretrained_emb=pre_emb_path,
                   num_layers=1, dropout=dropout).to(device)  # build a model

# loss and optimizer
criterion = nn.CrossEntropyLoss().to(device)  # CE loss
optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
 ↪weight_decay=weight_decay)  # optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                      step_size=5,
                                      gamma=0.5)  # decay LR by a factor of 0.5␣
 ↪every 10 epochs. You can change this

# logs
Train_Losses = []  # record average training loss each epoch
Val_Losses = []   # record average validation loss each epoch
total_step = len(train_data_loader)  # number of iterations each epoch
best_val_loss = np.inf

# start training
print('Start training...')
import time
tic = time.time()
for epoch in range(num_epochs):
# for epoch in range(2):
    print('Switch to training...')
    model.train()
    Train_loss_iter = []  # record the the training loss each iteration
    for itr, (images, captions, lengths) in enumerate(train_data_loader):
```

```
        #######Your Code###########
        model.zero_grad()
        images = Variable(images).to(device)
        captions = Variable(captions).to(device)
        predicted_cap_pack = pack_padded_sequence(captions, lengths,␣
↪batch_first=True)[0]
        predicted_cap = model(images, captions, lengths)
        predicted_cap = predicted_cap.view(-1, vocab_size)
        loss = criterion(input = predicted_cap, target=predicted_cap_pack)
        loss.backward()
        optimizer.step()

        # record the training loss
        Train_Losses.append(float(loss))

        # print log info
        if itr % log_step == 0:
            # print current loss and perplexity
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Perplexity: {:5.
↪4f}'
                      .format(epoch, num_epochs, itr, total_step, loss.item(),␣
↪np.exp(loss.item())))
    scheduler.step()
    Train_Losses.append(np.mean(Train_loss_iter))
    np.save(os.path.join(output_dir, 'TrainingLoss_lstm.npy'), Train_Losses)  #␣
↪save the training loss

    model.eval()
    # (optional) generate a sample during the training, you can use␣
↪deterministic mode
    # Your code


    # validation
    Val_Losses.append(val(model, val_data_loader, vocab))
    np.save(os.path.join(output_dir, 'ValLoss_lstm.npy'), Val_Losses) # save the␣
↪val loss

    # save model
    if Val_Losses[-1] < best_val_loss:
        best_val_loss = Val_Losses[-1]
        print('updated best val loss:', best_val_loss)
        print('Save model weights to...', output_dir)
        torch.save(model.state_dict(),
                   os.path.join(output_dir, 'pretrain-best.pth'.format(epoch +␣
↪1, itr + 1)))
```

```python
print('It took: {} s'.format(time.time() - tic))
```

```
/content/drive/My Drive/DL_Fall_2020/Assignment_4/wm_06.npy
Start training...
Switch to training...
Epoch [0/50], Step [0/114], Loss: 9.1967, Perplexity: 9864.4292
Epoch [0/50], Step [50/114], Loss: 3.5588, Perplexity: 35.1226
Epoch [0/50], Step [100/114], Loss: 3.2991, Perplexity: 27.0884
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnumeric.py:3335:
RuntimeWarning: Mean of empty slice.
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:161:
RuntimeWarning: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
```

```
Validating...
Step [0/127], Loss: 3.3755, Perplexity: 29.2382
Step [50/127], Loss: 3.2646, Perplexity: 26.1707
Step [100/127], Loss: 3.6123, Perplexity: 37.0526
updated best val loss: 3.3000397306727614
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [1/50], Step [0/114], Loss: 3.1994, Perplexity: 24.5174
Epoch [1/50], Step [50/114], Loss: 3.1185, Perplexity: 22.6130
Epoch [1/50], Step [100/114], Loss: 3.1364, Perplexity: 23.0215
Validating...
Step [0/127], Loss: 2.6516, Perplexity: 14.1773
Step [50/127], Loss: 3.0158, Perplexity: 20.4056
Step [100/127], Loss: 2.5862, Perplexity: 13.2787
updated best val loss: 3.1186115685410387
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [2/50], Step [0/114], Loss: 3.1360, Perplexity: 23.0118
Epoch [2/50], Step [50/114], Loss: 3.1355, Perplexity: 22.9998
Epoch [2/50], Step [100/114], Loss: 3.1685, Perplexity: 23.7714
Validating...
Step [0/127], Loss: 2.8438, Perplexity: 17.1813
Step [50/127], Loss: 2.9001, Perplexity: 18.1763
Step [100/127], Loss: 3.2208, Perplexity: 25.0483
updated best val loss: 3.0789894832400826
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [3/50], Step [0/114], Loss: 3.0870, Perplexity: 21.9111
```

```
Epoch [3/50], Step [50/114], Loss: 3.0810, Perplexity: 21.7810
Epoch [3/50], Step [100/114], Loss: 2.9672, Perplexity: 19.4374
Validating...
Step [0/127], Loss: 3.0122, Perplexity: 20.3331
Step [50/127], Loss: 2.8196, Perplexity: 16.7701
Step [100/127], Loss: 2.8023, Perplexity: 16.4828
updated best val loss: 3.0533739037401095
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [4/50], Step [0/114], Loss: 2.9306, Perplexity: 18.7392
Epoch [4/50], Step [50/114], Loss: 2.9195, Perplexity: 18.5326
Epoch [4/50], Step [100/114], Loss: 2.9324, Perplexity: 18.7724
Validating...
Step [0/127], Loss: 2.8991, Perplexity: 18.1572
Step [50/127], Loss: 3.1722, Perplexity: 23.8595
Step [100/127], Loss: 2.9768, Perplexity: 19.6242
updated best val loss: 3.0393130779266357
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [5/50], Step [0/114], Loss: 2.9816, Perplexity: 19.7193
Epoch [5/50], Step [50/114], Loss: 2.7993, Perplexity: 16.4329
Epoch [5/50], Step [100/114], Loss: 2.8397, Perplexity: 17.1100
Validating...
Step [0/127], Loss: 2.4462, Perplexity: 11.5444
Step [50/127], Loss: 3.1410, Perplexity: 23.1261
Step [100/127], Loss: 2.8435, Perplexity: 17.1750
updated best val loss: 2.9582198060403657
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [6/50], Step [0/114], Loss: 2.8194, Perplexity: 16.7661
Epoch [6/50], Step [50/114], Loss: 2.8207, Perplexity: 16.7882
Epoch [6/50], Step [100/114], Loss: 2.8846, Perplexity: 17.8965
Validating...
Step [0/127], Loss: 3.6719, Perplexity: 39.3281
Step [50/127], Loss: 2.5758, Perplexity: 13.1415
Step [100/127], Loss: 3.0980, Perplexity: 22.1525
updated best val loss: 2.9457708137241876
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [7/50], Step [0/114], Loss: 2.7077, Perplexity: 14.9950
Epoch [7/50], Step [50/114], Loss: 2.7242, Perplexity: 15.2441
Epoch [7/50], Step [100/114], Loss: 2.8001, Perplexity: 16.4469
Validating...
Step [0/127], Loss: 2.4566, Perplexity: 11.6654
```

```
Step [50/127], Loss: 3.0743, Perplexity: 21.6348
Step [100/127], Loss: 2.8221, Perplexity: 16.8114
updated best val loss: 2.9399302306137685
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [8/50], Step [0/114], Loss: 2.7430, Perplexity: 15.5332
Epoch [8/50], Step [50/114], Loss: 2.7839, Perplexity: 16.1814
Epoch [8/50], Step [100/114], Loss: 2.8609, Perplexity: 17.4771
Validating...
Step [0/127], Loss: 2.8387, Perplexity: 17.0928
Step [50/127], Loss: 3.3202, Perplexity: 27.6664
Step [100/127], Loss: 3.3835, Perplexity: 29.4732
Switch to training...
Epoch [9/50], Step [0/114], Loss: 2.7565, Perplexity: 15.7444
Epoch [9/50], Step [50/114], Loss: 2.7470, Perplexity: 15.5954
Epoch [9/50], Step [100/114], Loss: 2.7525, Perplexity: 15.6821
Validating...
Step [0/127], Loss: 2.7507, Perplexity: 15.6536
Step [50/127], Loss: 3.3305, Perplexity: 27.9525
Step [100/127], Loss: 2.7179, Perplexity: 15.1490
updated best val loss: 2.905920849071713
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [10/50], Step [0/114], Loss: 2.6438, Perplexity: 14.0668
Epoch [10/50], Step [50/114], Loss: 2.6761, Perplexity: 14.5287
Epoch [10/50], Step [100/114], Loss: 2.7600, Perplexity: 15.8005
Validating...
Step [0/127], Loss: 2.8030, Perplexity: 16.4943
Step [50/127], Loss: 2.6401, Perplexity: 14.0145
Step [100/127], Loss: 3.0468, Perplexity: 21.0482
Switch to training...
Epoch [11/50], Step [0/114], Loss: 2.6419, Perplexity: 14.0392
Epoch [11/50], Step [50/114], Loss: 2.6751, Perplexity: 14.5141
Epoch [11/50], Step [100/114], Loss: 2.7255, Perplexity: 15.2643
Validating...
Step [0/127], Loss: 2.9722, Perplexity: 19.5350
Step [50/127], Loss: 3.0590, Perplexity: 21.3057
Step [100/127], Loss: 3.1910, Perplexity: 24.3134
updated best val loss: 2.8773781577433186
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [12/50], Step [0/114], Loss: 2.6225, Perplexity: 13.7697
Epoch [12/50], Step [50/114], Loss: 2.6798, Perplexity: 14.5820
Epoch [12/50], Step [100/114], Loss: 2.6560, Perplexity: 14.2389
Validating...
```

```
Step [0/127], Loss: 3.3561, Perplexity: 28.6782
Step [50/127], Loss: 2.8389, Perplexity: 17.0964
Step [100/127], Loss: 2.8091, Perplexity: 16.5949
Switch to training...
Epoch [13/50], Step [0/114], Loss: 2.6248, Perplexity: 13.8012
Epoch [13/50], Step [50/114], Loss: 2.7257, Perplexity: 15.2675
Epoch [13/50], Step [100/114], Loss: 2.6792, Perplexity: 14.5741
Validating...
Step [0/127], Loss: 3.5100, Perplexity: 33.4474
Step [50/127], Loss: 2.8835, Perplexity: 17.8767
Step [100/127], Loss: 2.8325, Perplexity: 16.9880
Switch to training...
Epoch [14/50], Step [0/114], Loss: 2.6107, Perplexity: 13.6083
Epoch [14/50], Step [50/114], Loss: 2.6608, Perplexity: 14.3076
Epoch [14/50], Step [100/114], Loss: 2.6752, Perplexity: 14.5146
Validating...
Step [0/127], Loss: 2.8089, Perplexity: 16.5909
Step [50/127], Loss: 2.9213, Perplexity: 18.5658
Step [100/127], Loss: 3.2196, Perplexity: 25.0183
Switch to training...
Epoch [15/50], Step [0/114], Loss: 2.6033, Perplexity: 13.5081
Epoch [15/50], Step [50/114], Loss: 2.6393, Perplexity: 14.0029
Epoch [15/50], Step [100/114], Loss: 2.6312, Perplexity: 13.8898
Validating...
Step [0/127], Loss: 3.0927, Perplexity: 22.0364
Step [50/127], Loss: 2.7312, Perplexity: 15.3508
Step [100/127], Loss: 2.7424, Perplexity: 15.5239
updated best val loss: 2.8615870062760482
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [16/50], Step [0/114], Loss: 2.6917, Perplexity: 14.7565
Epoch [16/50], Step [50/114], Loss: 2.5993, Perplexity: 13.4548
Epoch [16/50], Step [100/114], Loss: 2.6174, Perplexity: 13.6996
Validating...
Step [0/127], Loss: 2.6297, Perplexity: 13.8692
Step [50/127], Loss: 2.5046, Perplexity: 12.2384
Step [100/127], Loss: 2.9687, Perplexity: 19.4671
Switch to training...
Epoch [17/50], Step [0/114], Loss: 2.5535, Perplexity: 12.8521
Epoch [17/50], Step [50/114], Loss: 2.5234, Perplexity: 12.4712
Epoch [17/50], Step [100/114], Loss: 2.6090, Perplexity: 13.5860
Validating...
Step [0/127], Loss: 3.1692, Perplexity: 23.7894
Step [50/127], Loss: 2.6280, Perplexity: 13.8458
Step [100/127], Loss: 2.5929, Perplexity: 13.3689
Switch to training...
Epoch [18/50], Step [0/114], Loss: 2.6862, Perplexity: 14.6758
```

```
Epoch [18/50], Step [50/114], Loss: 2.5974, Perplexity: 13.4294
Epoch [18/50], Step [100/114], Loss: 2.5548, Perplexity: 12.8693
Validating...
Step [0/127], Loss: 2.9099, Perplexity: 18.3545
Step [50/127], Loss: 3.0089, Perplexity: 20.2643
Step [100/127], Loss: 3.1570, Perplexity: 23.4994
Switch to training...
Epoch [19/50], Step [0/114], Loss: 2.5982, Perplexity: 13.4394
Epoch [19/50], Step [50/114], Loss: 2.6393, Perplexity: 14.0036
Epoch [19/50], Step [100/114], Loss: 2.5786, Perplexity: 13.1787
Validating...
Step [0/127], Loss: 2.1423, Perplexity: 8.5190
Step [50/127], Loss: 2.8382, Perplexity: 17.0849
Step [100/127], Loss: 2.8037, Perplexity: 16.5063
Switch to training...
Epoch [20/50], Step [0/114], Loss: 2.5285, Perplexity: 12.5352
Epoch [20/50], Step [50/114], Loss: 2.6854, Perplexity: 14.6647
Epoch [20/50], Step [100/114], Loss: 2.5220, Perplexity: 12.4537
Validating...
Step [0/127], Loss: 2.6534, Perplexity: 14.2019
Step [50/127], Loss: 2.8397, Perplexity: 17.1105
Step [100/127], Loss: 3.1112, Perplexity: 22.4485
updated best val loss: 2.8494993570282703
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [21/50], Step [0/114], Loss: 2.5930, Perplexity: 13.3705
Epoch [21/50], Step [50/114], Loss: 2.5899, Perplexity: 13.3279
Epoch [21/50], Step [100/114], Loss: 2.6301, Perplexity: 13.8748
Validating...
Step [0/127], Loss: 3.2296, Perplexity: 25.2692
Step [50/127], Loss: 2.7239, Perplexity: 15.2389
Step [100/127], Loss: 3.6257, Perplexity: 37.5510
Switch to training...
Epoch [22/50], Step [0/114], Loss: 2.4671, Perplexity: 11.7888
Epoch [22/50], Step [50/114], Loss: 2.6088, Perplexity: 13.5831
Epoch [22/50], Step [100/114], Loss: 2.5855, Perplexity: 13.2698
Validating...
Step [0/127], Loss: 3.0787, Perplexity: 21.7299
Step [50/127], Loss: 2.5891, Perplexity: 13.3179
Step [100/127], Loss: 3.0095, Perplexity: 20.2779
Switch to training...
Epoch [23/50], Step [0/114], Loss: 2.5248, Perplexity: 12.4884
Epoch [23/50], Step [50/114], Loss: 2.6101, Perplexity: 13.5997
Epoch [23/50], Step [100/114], Loss: 2.5515, Perplexity: 12.8265
Validating...
Step [0/127], Loss: 2.6921, Perplexity: 14.7632
Step [50/127], Loss: 2.8838, Perplexity: 17.8828
```

```
Step [100/127], Loss: 3.6494, Perplexity: 38.4513
Switch to training...
Epoch [24/50], Step [0/114], Loss: 2.6055, Perplexity: 13.5383
Epoch [24/50], Step [50/114], Loss: 2.5914, Perplexity: 13.3489
Epoch [24/50], Step [100/114], Loss: 2.5800, Perplexity: 13.1974
Validating...
Step [0/127], Loss: 2.8379, Perplexity: 17.0796
Step [50/127], Loss: 2.8680, Perplexity: 17.6016
Step [100/127], Loss: 3.2286, Perplexity: 25.2442
Switch to training...
Epoch [25/50], Step [0/114], Loss: 2.5142, Perplexity: 12.3572
Epoch [25/50], Step [50/114], Loss: 2.5643, Perplexity: 12.9916
Epoch [25/50], Step [100/114], Loss: 2.5188, Perplexity: 12.4142
Validating...
Step [0/127], Loss: 3.7119, Perplexity: 40.9306
Step [50/127], Loss: 2.5537, Perplexity: 12.8551
Step [100/127], Loss: 2.8592, Perplexity: 17.4476
Switch to training...
Epoch [26/50], Step [0/114], Loss: 2.5700, Perplexity: 13.0661
Epoch [26/50], Step [50/114], Loss: 2.5030, Perplexity: 12.2194
Epoch [26/50], Step [100/114], Loss: 2.5886, Perplexity: 13.3112
Validating...
Step [0/127], Loss: 2.7194, Perplexity: 15.1710
Step [50/127], Loss: 2.6673, Perplexity: 14.4004
Step [100/127], Loss: 2.6868, Perplexity: 14.6846
Switch to training...
Epoch [27/50], Step [0/114], Loss: 2.5567, Perplexity: 12.8930
Epoch [27/50], Step [50/114], Loss: 2.4836, Perplexity: 11.9847
Epoch [27/50], Step [100/114], Loss: 2.5198, Perplexity: 12.4255
Validating...
Step [0/127], Loss: 2.5743, Perplexity: 13.1221
Step [50/127], Loss: 3.0380, Perplexity: 20.8637
Step [100/127], Loss: 2.9892, Perplexity: 19.8697
Switch to training...
Epoch [28/50], Step [0/114], Loss: 2.5173, Perplexity: 12.3957
Epoch [28/50], Step [50/114], Loss: 2.6102, Perplexity: 13.6023
Epoch [28/50], Step [100/114], Loss: 2.6124, Perplexity: 13.6314
Validating...
Step [0/127], Loss: 2.5525, Perplexity: 12.8398
Step [50/127], Loss: 2.5027, Perplexity: 12.2155
Step [100/127], Loss: 3.1519, Perplexity: 23.3793
Switch to training...
Epoch [29/50], Step [0/114], Loss: 2.5095, Perplexity: 12.2983
Epoch [29/50], Step [50/114], Loss: 2.5590, Perplexity: 12.9225
Epoch [29/50], Step [100/114], Loss: 2.5575, Perplexity: 12.9029
Validating...
Step [0/127], Loss: 3.0368, Perplexity: 20.8380
Step [50/127], Loss: 2.7998, Perplexity: 16.4412
```

```
Step [100/127], Loss: 3.1684, Perplexity: 23.7702
Switch to training...
Epoch [30/50], Step [0/114], Loss: 2.5088, Perplexity: 12.2903
Epoch [30/50], Step [50/114], Loss: 2.5926, Perplexity: 13.3642
Epoch [30/50], Step [100/114], Loss: 2.5754, Perplexity: 13.1360
Validating...
Step [0/127], Loss: 2.4757, Perplexity: 11.8902
Step [50/127], Loss: 2.7512, Perplexity: 15.6612
Step [100/127], Loss: 3.1640, Perplexity: 23.6650
Switch to training...
Epoch [31/50], Step [0/114], Loss: 2.5388, Perplexity: 12.6647
Epoch [31/50], Step [50/114], Loss: 2.5552, Perplexity: 12.8742
Epoch [31/50], Step [100/114], Loss: 2.5697, Perplexity: 13.0614
Validating...
Step [0/127], Loss: 2.8812, Perplexity: 17.8355
Step [50/127], Loss: 2.6630, Perplexity: 14.3393
Step [100/127], Loss: 2.9449, Perplexity: 19.0091
Switch to training...
Epoch [32/50], Step [0/114], Loss: 2.5897, Perplexity: 13.3252
Epoch [32/50], Step [50/114], Loss: 2.5348, Perplexity: 12.6142
Epoch [32/50], Step [100/114], Loss: 2.5883, Perplexity: 13.3073
Validating...
Step [0/127], Loss: 3.3178, Perplexity: 27.6003
Step [50/127], Loss: 2.8602, Perplexity: 17.4645
Step [100/127], Loss: 2.7902, Perplexity: 16.2846
Switch to training...
Epoch [33/50], Step [0/114], Loss: 2.5153, Perplexity: 12.3705
Epoch [33/50], Step [50/114], Loss: 2.5228, Perplexity: 12.4637
Epoch [33/50], Step [100/114], Loss: 2.5394, Perplexity: 12.6720
Validating...
Step [0/127], Loss: 2.7785, Perplexity: 16.0941
Step [50/127], Loss: 2.5469, Perplexity: 12.7669
Step [100/127], Loss: 2.5493, Perplexity: 12.7982
updated best val loss: 2.8425030783405454
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [34/50], Step [0/114], Loss: 2.4937, Perplexity: 12.1055
Epoch [34/50], Step [50/114], Loss: 2.6083, Perplexity: 13.5758
Epoch [34/50], Step [100/114], Loss: 2.5213, Perplexity: 12.4451
Validating...
Step [0/127], Loss: 3.2536, Perplexity: 25.8826
Step [50/127], Loss: 2.7840, Perplexity: 16.1841
Step [100/127], Loss: 2.1760, Perplexity: 8.8109
Switch to training...
Epoch [35/50], Step [0/114], Loss: 2.5229, Perplexity: 12.4652
Epoch [35/50], Step [50/114], Loss: 2.5738, Perplexity: 13.1150
Epoch [35/50], Step [100/114], Loss: 2.5409, Perplexity: 12.6906
```

```
Validating...
Step [0/127], Loss: 2.7848, Perplexity: 16.1970
Step [50/127], Loss: 2.7138, Perplexity: 15.0864
Step [100/127], Loss: 3.0755, Perplexity: 21.6617
Switch to training...
Epoch [36/50], Step [0/114], Loss: 2.5519, Perplexity: 12.8314
Epoch [36/50], Step [50/114], Loss: 2.5538, Perplexity: 12.8553
Epoch [36/50], Step [100/114], Loss: 2.4590, Perplexity: 11.6934
Validating...
Step [0/127], Loss: 3.1431, Perplexity: 23.1749
Step [50/127], Loss: 2.9564, Perplexity: 19.2295
Step [100/127], Loss: 3.1953, Perplexity: 24.4171
Switch to training...
Epoch [37/50], Step [0/114], Loss: 2.4676, Perplexity: 11.7940
Epoch [37/50], Step [50/114], Loss: 2.5541, Perplexity: 12.8596
Epoch [37/50], Step [100/114], Loss: 2.4754, Perplexity: 11.8864
Validating...
Step [0/127], Loss: 3.4164, Perplexity: 30.4602
Step [50/127], Loss: 2.7267, Perplexity: 15.2822
Step [100/127], Loss: 2.5885, Perplexity: 13.3098
Switch to training...
Epoch [38/50], Step [0/114], Loss: 2.5341, Perplexity: 12.6045
Epoch [38/50], Step [50/114], Loss: 2.6179, Perplexity: 13.7069
Epoch [38/50], Step [100/114], Loss: 2.5301, Perplexity: 12.5549
Validating...
Step [0/127], Loss: 2.5736, Perplexity: 13.1123
Step [50/127], Loss: 2.1873, Perplexity: 8.9109
Step [100/127], Loss: 3.4548, Perplexity: 31.6509
Switch to training...
Epoch [39/50], Step [0/114], Loss: 2.5128, Perplexity: 12.3394
Epoch [39/50], Step [50/114], Loss: 2.5903, Perplexity: 13.3342
Epoch [39/50], Step [100/114], Loss: 2.5226, Perplexity: 12.4608
Validating...
Step [0/127], Loss: 2.6632, Perplexity: 14.3418
Step [50/127], Loss: 3.2896, Perplexity: 26.8325
Step [100/127], Loss: 2.6255, Perplexity: 13.8117
updated best val loss: 2.841626351273905
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [40/50], Step [0/114], Loss: 2.5148, Perplexity: 12.3647
Epoch [40/50], Step [50/114], Loss: 2.5257, Perplexity: 12.4990
Epoch [40/50], Step [100/114], Loss: 2.5529, Perplexity: 12.8449
Validating...
Step [0/127], Loss: 2.6444, Perplexity: 14.0757
Step [50/127], Loss: 3.0954, Perplexity: 22.0967
Step [100/127], Loss: 3.5971, Perplexity: 36.4925
Switch to training...
```

```
Epoch [41/50], Step [0/114], Loss: 2.5291, Perplexity: 12.5422
Epoch [41/50], Step [50/114], Loss: 2.5528, Perplexity: 12.8430
Epoch [41/50], Step [100/114], Loss: 2.5882, Perplexity: 13.3058
Validating...
Step [0/127], Loss: 2.9757, Perplexity: 19.6034
Step [50/127], Loss: 3.1688, Perplexity: 23.7792
Step [100/127], Loss: 2.7754, Perplexity: 16.0444
Switch to training...
Epoch [42/50], Step [0/114], Loss: 2.5562, Perplexity: 12.8869
Epoch [42/50], Step [50/114], Loss: 2.5088, Perplexity: 12.2906
Epoch [42/50], Step [100/114], Loss: 2.5362, Perplexity: 12.6322
Validating...
Step [0/127], Loss: 2.8588, Perplexity: 17.4413
Step [50/127], Loss: 2.8259, Perplexity: 16.8769
Step [100/127], Loss: 2.4660, Perplexity: 11.7750
Switch to training...
Epoch [43/50], Step [0/114], Loss: 2.5374, Perplexity: 12.6468
Epoch [43/50], Step [50/114], Loss: 2.4976, Perplexity: 12.1531
Epoch [43/50], Step [100/114], Loss: 2.5057, Perplexity: 12.2525
Validating...
Step [0/127], Loss: 2.5595, Perplexity: 12.9295
Step [50/127], Loss: 2.8840, Perplexity: 17.8862
Step [100/127], Loss: 3.1887, Perplexity: 24.2559
Switch to training...
Epoch [44/50], Step [0/114], Loss: 2.5095, Perplexity: 12.2982
Epoch [44/50], Step [50/114], Loss: 2.5137, Perplexity: 12.3509
Epoch [44/50], Step [100/114], Loss: 2.5183, Perplexity: 12.4080
Validating...
Step [0/127], Loss: 2.6764, Perplexity: 14.5325
Step [50/127], Loss: 2.5301, Perplexity: 12.5546
Step [100/127], Loss: 3.7817, Perplexity: 43.8896
updated best val loss: 2.8387038182085895
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
Switch to training...
Epoch [45/50], Step [0/114], Loss: 2.5692, Perplexity: 13.0556
Epoch [45/50], Step [50/114], Loss: 2.5030, Perplexity: 12.2196
Epoch [45/50], Step [100/114], Loss: 2.5596, Perplexity: 12.9306
Validating...
Step [0/127], Loss: 2.7298, Perplexity: 15.3296
Step [50/127], Loss: 3.4381, Perplexity: 31.1268
Step [100/127], Loss: 3.2502, Perplexity: 25.7960
Switch to training...
Epoch [46/50], Step [0/114], Loss: 2.5979, Perplexity: 13.4351
Epoch [46/50], Step [50/114], Loss: 2.5236, Perplexity: 12.4735
Epoch [46/50], Step [100/114], Loss: 2.4594, Perplexity: 11.6974
Validating...
Step [0/127], Loss: 2.8805, Perplexity: 17.8226
```

```
Step [50/127], Loss: 3.0389, Perplexity: 20.8820
Step [100/127], Loss: 3.4368, Perplexity: 31.0889
Switch to training...
Epoch [47/50], Step [0/114], Loss: 2.6026, Perplexity: 13.4992
Epoch [47/50], Step [50/114], Loss: 2.5164, Perplexity: 12.3841
Epoch [47/50], Step [100/114], Loss: 2.5802, Perplexity: 13.1992
Validating...
Step [0/127], Loss: 2.9822, Perplexity: 19.7320
Step [50/127], Loss: 2.6580, Perplexity: 14.2675
Step [100/127], Loss: 2.8360, Perplexity: 17.0474
Switch to training...
Epoch [48/50], Step [0/114], Loss: 2.5064, Perplexity: 12.2603
Epoch [48/50], Step [50/114], Loss: 2.5741, Perplexity: 13.1199
Epoch [48/50], Step [100/114], Loss: 2.5717, Perplexity: 13.0877
Validating...
Step [0/127], Loss: 2.8604, Perplexity: 17.4687
Step [50/127], Loss: 2.5709, Perplexity: 13.0776
Step [100/127], Loss: 3.0426, Perplexity: 20.9605
Switch to training...
Epoch [49/50], Step [0/114], Loss: 2.5228, Perplexity: 12.4639
Epoch [49/50], Step [50/114], Loss: 2.5407, Perplexity: 12.6885
Epoch [49/50], Step [100/114], Loss: 2.5666, Perplexity: 13.0216
Validating...
Step [0/127], Loss: 2.7909, Perplexity: 16.2962
Step [50/127], Loss: 2.9607, Perplexity: 19.3119
Step [100/127], Loss: 3.2011, Perplexity: 24.5585
updated best val loss: 2.8088028768854816
Save model weights to... /content/drive/My
Drive/DL_Fall_2020/Assignment_4//checkpoints/pretrained_emb/
It took: 14552.241356372833 s
```

13.10 Section 3.2.3 Evaluation [3 pts]

[ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


train_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',
 ↪split='train', vocab=vocab,
                              transform=transform, batch_size=batch_size,
 ↪shuffle=True, num_workers=12)
val_data_loader = get_loader(root=path_to_homework + '/flickr30k_images/',
 ↪split='val', vocab=vocab,
                              transform=transform, batch_size=8, shuffle=True,
 ↪num_workers=4)

model = Word_embeddings(vocab_size=len(vocab), emb_dim=emb_dim,
 ↪hidden_dim=hidden_dim,
```

```
                   num_layers=1, dropout=dropout).to(device)  # build a model

vocab_size=len(vocab)

# loss and optimizer
criterion = nn.CrossEntropyLoss().to(device)  # CE loss
optimizer = torch.optim.Adam(model.parameters(), lr=lr,␣
 ↪weight_decay=weight_decay)  # optimizer
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                      step_size=5,
                                      gamma=0.5)  # decay LR by a factor of 0.5␣
 ↪every 10 epochs. You can change this
```

[29]:
```
# ## Evaluate your model using BLEU score. Use Deterministic mode
model.eval()
bleu1, bleu2, bleu3, bleu4 = evaluation(model, vocab, mode='Deterministic')
print("BLEU 1:{}, BLEU 2:{}, BLEU 3:{}, BLEU 4:{}".format(bleu1, bleu2, bleu3,␣
 ↪bleu4))
```

```
  0%|          | 0/125 [00:00<?, ?it/s]

Run on the test set...

  1%|          | 1/125 [00:00<01:23,  1.48it/s]
  2%|          | 2/125 [00:00<01:03,  1.95it/s]
  2%|          | 3/125 [00:00<00:48,  2.53it/s]
  3%|          | 4/125 [00:01<00:39,  3.10it/s]
  4%|          | 5/125 [00:01<00:32,  3.70it/s]
  6%|          | 7/125 [00:01<00:24,  4.73it/s]
  6%|          | 8/125 [00:01<00:21,  5.45it/s]
  7%|          | 9/125 [00:01<00:19,  5.90it/s]
  9%|          | 11/125 [00:01<00:15,  7.26it/s]
 10%|          | 13/125 [00:01<00:14,  7.61it/s]
 12%|          | 15/125 [00:02<00:12,  8.78it/s]
 14%|          | 17/125 [00:02<00:12,  8.42it/s]
 15%|          | 19/125 [00:02<00:11,  9.54it/s]
 17%|          | 21/125 [00:02<00:10,  9.65it/s]
 18%|          | 23/125 [00:02<00:10,  9.68it/s]
 20%|          | 25/125 [00:03<00:09, 10.25it/s]
 22%|          | 27/125 [00:03<00:10,  9.25it/s]
 23%|          | 29/125 [00:03<00:09, 10.48it/s]
 25%|          | 31/125 [00:03<00:09, 10.17it/s]
 26%|          | 33/125 [00:03<00:08, 10.54it/s]
 28%|          | 35/125 [00:04<00:08, 10.90it/s]
 30%|          | 37/125 [00:04<00:09,  9.39it/s]
 31%|          | 39/125 [00:04<00:08, 10.15it/s]
```

```
 33%|      | 41/125 [00:04<00:08,  9.65it/s]
 34%|      | 43/125 [00:04<00:07, 10.35it/s]
 36%|      | 45/125 [00:05<00:08,  9.63it/s]
 38%|      | 47/125 [00:05<00:07, 10.12it/s]
 39%|      | 49/125 [00:05<00:07, 10.74it/s]
 41%|      | 51/125 [00:05<00:07, 10.54it/s]
 42%|     | 53/125 [00:05<00:06, 10.82it/s]
 44%|     | 55/125 [00:05<00:05, 11.90it/s]
 46%|     | 57/125 [00:06<00:07,  9.68it/s]
 47%|     | 59/125 [00:06<00:06, 10.21it/s]
 49%|     | 61/125 [00:06<00:06, 10.27it/s]
 50%|     | 63/125 [00:06<00:06, 10.30it/s]
 52%|    | 65/125 [00:07<00:06,  9.78it/s]
 54%|    | 67/125 [00:07<00:05, 10.89it/s]
 55%|    | 69/125 [00:07<00:05, 10.39it/s]
 57%|    | 71/125 [00:07<00:05,  9.76it/s]
 58%|    | 73/125 [00:07<00:04, 10.98it/s]
 60%|    | 75/125 [00:08<00:04, 10.01it/s]
 62%|   | 77/125 [00:08<00:04,  9.96it/s]
 63%|   | 79/125 [00:08<00:04, 10.19it/s]
 65%|   | 81/125 [00:08<00:04, 10.58it/s]
 66%|   | 83/125 [00:08<00:03, 10.56it/s]
 68%|   | 85/125 [00:09<00:04,  9.80it/s]
 70%|   | 87/125 [00:09<00:03, 10.16it/s]
 71%|   | 89/125 [00:09<00:03,  9.86it/s]
 73%|  | 91/125 [00:09<00:03, 10.40it/s]
 74%|  | 93/125 [00:09<00:03, 10.31it/s]
 76%|  | 95/125 [00:09<00:02, 10.62it/s]
 78%|  | 97/125 [00:10<00:02, 10.30it/s]
 79%|  | 99/125 [00:10<00:02,  9.66it/s]
 80%|  | 100/125 [00:10<00:02,  9.60it/s]
 82%| | 102/125 [00:10<00:02, 10.13it/s]
 83%| | 104/125 [00:10<00:02, 10.04it/s]
 85%| | 106/125 [00:11<00:01, 10.33it/s]
 86%| | 108/125 [00:11<00:01, 10.41it/s]
 88%| | 110/125 [00:11<00:01, 10.49it/s]
 90%| | 112/125 [00:11<00:01, 10.67it/s]
 91%| | 114/125 [00:11<00:01, 10.32it/s]
 93%|| 116/125 [00:12<00:00, 10.26it/s]
 94%|| 118/125 [00:12<00:00, 11.33it/s]
 96%|| 120/125 [00:12<00:00, 12.23it/s]
 98%|| 122/125 [00:12<00:00, 13.78it/s]
100%|| 125/125 [00:12<00:00,  9.91it/s]

  0%|          | 0/1000 [00:00<?, ?it/s]
  2%|         | 22/1000 [00:00<00:04, 217.05it/s]

Computing BLEU
```

```
  5%|          | 47/1000 [00:00<00:04, 224.16it/s]
  7%|          | 71/1000 [00:00<00:04, 226.82it/s]
 10%|          | 98/1000 [00:00<00:03, 236.23it/s]
 12%|         | 123/1000 [00:00<00:03, 240.08it/s]
 15%|         | 150/1000 [00:00<00:03, 247.49it/s]
 17%|         | 174/1000 [00:00<00:03, 243.82it/s]
 20%|         | 197/1000 [00:00<00:03, 236.04it/s]
 22%|        | 224/1000 [00:00<00:03, 243.40it/s]
 25%|        | 248/1000 [00:01<00:03, 236.40it/s]
 27%|        | 272/1000 [00:01<00:03, 229.24it/s]
 30%|        | 296/1000 [00:01<00:03, 231.52it/s]
 32%|       | 321/1000 [00:01<00:02, 236.00it/s]
 35%|       | 346/1000 [00:01<00:02, 237.69it/s]
 37%|       | 370/1000 [00:01<00:02, 235.87it/s]
 39%|       | 394/1000 [00:01<00:02, 236.87it/s]
 42%|      | 419/1000 [00:01<00:02, 239.26it/s]
 44%|      | 444/1000 [00:01<00:02, 241.70it/s]
 47%|      | 469/1000 [00:01<00:02, 228.49it/s]
 50%|      | 496/1000 [00:02<00:02, 238.14it/s]
 52%|     | 521/1000 [00:02<00:02, 236.08it/s]
 55%|     | 545/1000 [00:02<00:01, 232.18it/s]
 57%|     | 569/1000 [00:02<00:01, 230.30it/s]
 59%|     | 593/1000 [00:02<00:01, 226.53it/s]
 62%|    | 616/1000 [00:02<00:01, 226.29it/s]
 64%|    | 640/1000 [00:02<00:01, 228.02it/s]
 67%|    | 666/1000 [00:02<00:01, 233.79it/s]
 69%|    | 690/1000 [00:02<00:01, 223.42it/s]
 72%|   | 715/1000 [00:03<00:01, 229.16it/s]
 74%|   | 739/1000 [00:03<00:01, 223.34it/s]
 76%|   | 762/1000 [00:03<00:01, 220.82it/s]
 79%|   | 786/1000 [00:03<00:00, 225.45it/s]
 81%|   | 809/1000 [00:03<00:00, 226.11it/s]
 83%|  | 832/1000 [00:03<00:00, 214.77it/s]
 86%|  | 855/1000 [00:03<00:00, 217.88it/s]
 88%|  | 880/1000 [00:03<00:00, 224.95it/s]
 90%|  | 904/1000 [00:03<00:00, 227.34it/s]
 93%|| 927/1000 [00:04<00:00, 223.37it/s]
 95%|| 952/1000 [00:04<00:00, 229.07it/s]
 98%|| 976/1000 [00:04<00:00, 226.87it/s]
100%|| 1000/1000 [00:04<00:00, 230.83it/s]

BLEU 1:69.3856560848057, BLEU 2:45.79195915761376, BLEU 3:26.821433226826393,
BLEU 4:17.211552439590815
```

13.11 Section 3.2.4 Discussion [2 pts]

Compared to index embeddings, do pretrained embeddings improve the performance? Try to explain it.

Your Comments: It seems that pretrained embeddings lead to a faster training and a better test results. I think this is because the model can obtain more semantic signals from pretrained embeddings than training data through the index embeddings.

13.11.1 Guidelines for Downloading PDF in Google Colab

- Run below cells only in Google Colab, Comment out in case of Jupyter notebook

[ ]:
```
#Run below two lines (in google colab), installation steps to get .pdf of the␣
↪notebook

!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc

# After installation, comment above two lines and run again to remove␣
↪installation comments from the notebook.
```

[4]:
```
# Find path to your notebook file in drive and enter in below line

!jupyter nbconvert --to PDF "/content/drive/My Drive/DL_Fall_2020/Assignment_4/
↪Assignment_4.ipynb"

#Example: "/content/drive/My Drive/DL_Fall_2020/Assignment_4/DL_Assignment_4.
↪ipynb"
```