Image Captioning Final Project

Hai Bacti

January 19, 2021

1 Image Captioning Final Project

In this final project you will define and train an image-to-caption model, that can produce descriptions for real world images!

Model architecture: CNN encoder and RNN decoder. (https://research.googleblog.com/2014/11/a-picture-is-worth-thousand-coherent.html)

2 Import stuff

[1]:
```
import sys
sys.path.append("..")
import grading
import download_utils
import tensorflow as tf
print(tf.__version__)
```

```
2.3.0
```

[2]:
```
from tensorflow import keras
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
L = keras.layers
K = keras.backend
import utils
import time
import zipfile
import json
from collections import Counter, defaultdict
import re
import random
from random import choice
import grading_utils
import os
```

```
from keras_utils import reset_tf_session
import tqdm_utils
from numpy.random import default_rng
rng = default_rng(1204)
```

3 Prepare the storage for model checkpoints

[3]:
```
# Leave USE_GOOGLE_DRIVE = False if you're running locally!
# We recommend to set USE_GOOGLE_DRIVE = True in Google Colab!
# If set to True, we will mount Google Drive, so that you can restore your␣
 ↪checkpoint
# and continue trainig even if your previous Colab session dies.
# If set to True, follow on-screen instructions to access Google Drive (you␣
 ↪must have a Google account).
USE_GOOGLE_DRIVE = False

def mount_google_drive():
    from google.colab import drive
    mount_directory = "/content/gdrive"
    drive.mount(mount_directory)
    drive_root = mount_directory + "/" + list(filter(lambda x: x[0] != '.', os.
 ↪listdir(mount_directory)))[0] + "/colab"
    return drive_root

CHECKPOINT_ROOT = ""
if USE_GOOGLE_DRIVE:
    CHECKPOINT_ROOT = mount_google_drive() + "/"

def get_checkpoint_path(epoch=None):
    if epoch is None:
        return os.path.abspath(CHECKPOINT_ROOT + "weights")
    else:
        return os.path.abspath(CHECKPOINT_ROOT + "weights_{}".format(epoch))

# example of checkpoint dir
print(get_checkpoint_path(10))
```

```
/mnt/e/OneDrive/advanced-machine-learning/course-1/week6/weights_10
```

4 Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

5 Download data

Takes 10 hours and 20 GB. We've downloaded necessary files for you.

Relevant links (just in case): - train images http://msvocds.blob.core.windows.net/coco2014/train2014.zip - validation images http://msvocds.blob.core.windows.net/coco2014/val2014.zip - captions for both train and validation http://msvocds.blob.core.windows.net/annotations-1-0-3/captions_train-val2014.zip

Torrent link: https://tinyurl.com/y2ortcw6

6 Extract image features

We will use pre-trained InceptionV3 model for CNN encoder (https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html) and extract its last hidden layer as an embedding:

[4]:
```
IMG_SIZE = 299
```

[5]:
```
# we take the last hidden layer of IncetionV3 as an image embedding
def get_cnn_encoder():
    K.set_learning_phase(False)
    model = keras.applications.InceptionV3(include_top = False)
    preprocess_for_model = keras.applications.inception_v3.preprocess_input

    model = keras.models.Model(model.inputs, keras.layers.
 ↪GlobalAveragePooling2D()(model.output))
    return model, preprocess_for_model
```

Features extraction takes too much time on CPU: - Takes 16 minutes on GPU. - 25x slower (InceptionV3) on CPU and takes 7 hours. - 10x slower (MobileNet) on CPU and takes 3 hours.

So we've done it for you with the following code:

```
# load pre-trained model
reset_tf_session()
encoder, preprocess_for_model = get_cnn_encoder()

# extract train features
train_img_embeds, train_img_fns = utils.apply_model(
    "train2014.zip", encoder, preprocess_for_model, input_shape=(IMG_SIZE, IMG_SIZE))
utils.save_pickle(train_img_embeds, "train_img_embeds.pickle")
utils.save_pickle(train_img_fns, "train_img_fns.pickle")

# extract validation features
val_img_embeds, val_img_fns = utils.apply_model(
    "val2014.zip", encoder, preprocess_for_model, input_shape=(IMG_SIZE, IMG_SIZE))
utils.save_pickle(val_img_embeds, "val_img_embeds.pickle")
```

```
utils.save_pickle(val_img_fns, "val_img_fns.pickle")

# sample images for learners
def sample_zip(fn_in, fn_out, rate=0.01, seed=42):
    np.random.seed(seed)
    with zipfile.ZipFile(fn_in) as fin, zipfile.ZipFile(fn_out, "w") as fout:
        sampled = filter(lambda _: np.random.rand() < rate, fin.filelist)
        for zInfo in sampled:
            fout.writestr(zInfo, fin.read(zInfo))

sample_zip("train2014.zip", "train2014_sample.zip")
sample_zip("val2014.zip", "val2014_sample.zip")
```

[6]:
```
# load prepared embeddings
train_img_embeds = utils.read_pickle("train_img_embeds.pickle")
train_img_fns = utils.read_pickle("train_img_fns.pickle")
val_img_embeds = utils.read_pickle("val_img_embeds.pickle")
val_img_fns = utils.read_pickle("val_img_fns.pickle")
# check shapes
print(train_img_embeds.shape, len(train_img_fns))
print(val_img_embeds.shape, len(val_img_fns))
```

```
(82783, 2048) 82783
(40504, 2048) 40504
```

[7]:
```
# check prepared samples of images
list(filter(lambda x: x.endswith("_sample.zip"), os.listdir(".")))
```

[7]:
```
['train2014_sample.zip', 'val2014_sample.zip']
```

7 Extract captions for images

[8]:
```
# extract captions from zip
def get_captions_for_fns(fns, zip_fn, zip_json_path):
    zf = zipfile.ZipFile(zip_fn)
    j = json.loads(zf.read(zip_json_path).decode("utf8"))
    id_to_fn = {img["id"]: img["file_name"] for img in j["images"]}
    fn_to_caps = defaultdict(list)
    for cap in j['annotations']:
        fn_to_caps[id_to_fn[cap['image_id']]].append(cap['caption'])
    fn_to_caps = dict(fn_to_caps)
    return list(map(lambda x: fn_to_caps[x], fns))

train_captions = get_captions_for_fns \
    (train_img_fns, "captions_train-val2014.zip", "annotations/
 ↪captions_train2014.json")
```

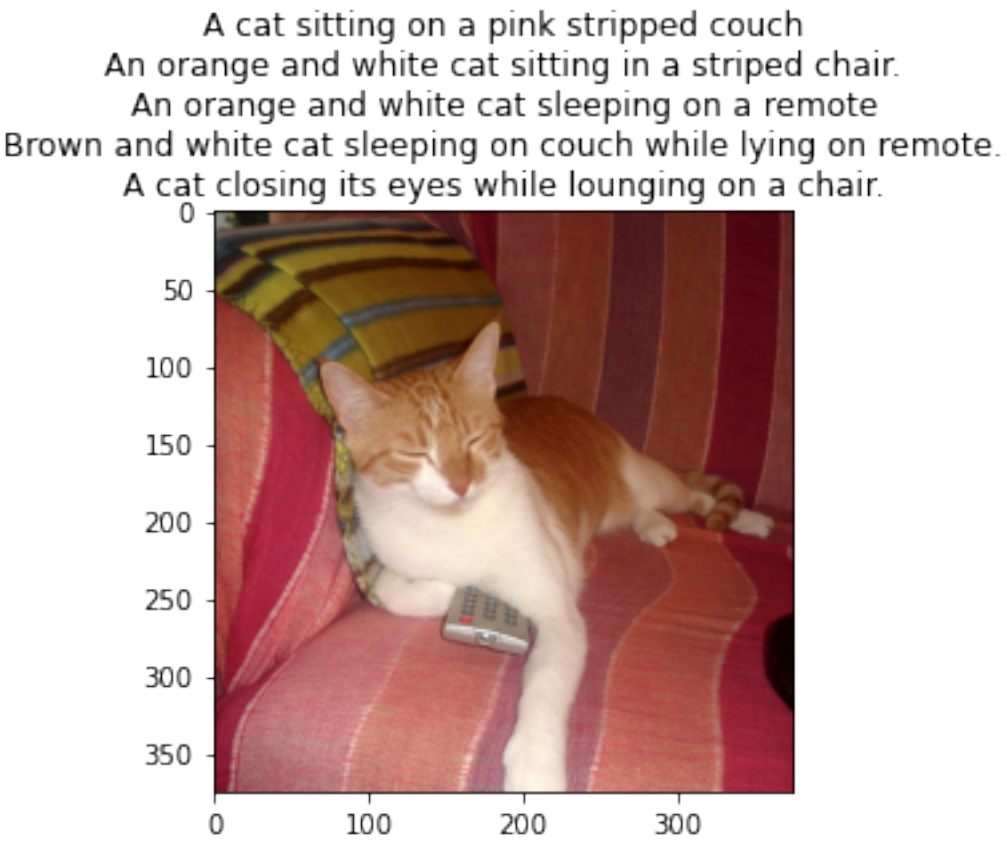
```
val_captions = get_captions_for_fns \
    (val_img_fns, "captions_train-val2014.zip", "annotations/captions_val2014.
 ↪json")

# check shape
print(len(train_img_fns), len(train_captions))
print(len(val_img_fns), len(val_captions))
```

```
82783 82783
40504 40504
```

[9]:
```
# look at training example (each has 5 captions)
def show_trainig_example(train_img_fns, train_captions, example_idx = 0):
    """
    You can change example_idx and see different images
    """
    zf = zipfile.ZipFile("train2014_sample.zip")
    captions_by_file = dict(zip(train_img_fns, train_captions))
    all_files = set(train_img_fns)
    found_files = list(filter(lambda x: x.filename.rsplit("/")[-1] in␣
 ↪all_files, zf.filelist))
    example = found_files[example_idx]
    img = utils.decode_image_from_buf(zf.read(example))
    plt.imshow(utils.image_center_crop(img))
    plt.title("\n".join(captions_by_file[example.filename.rsplit("/")[-1]]))
    plt.show()

show_trainig_example(train_img_fns, train_captions, example_idx = 142)
```



8 Prepare captions for training

[10]:
```
# preview captions data
train_captions[:2]
```

[10]:
```
[['A long dirt road going through a forest.',
  'A SCENE OF WATER AND A PATH WAY',
  'A sandy path surrounded by trees leads to a beach.',
  'Ocean view through a dirt road surrounded by a forested area. ',
  'dirt path leading beneath barren trees to open plains'],
 ['A group of zebra standing next to each other.',
  'This is an image of of zebras drinking',
  'ZEBRAS AND BIRDS SHARING THE SAME WATERING HOLE',
  'Zebras that are bent over and drinking water together.',
  'a number of zebras drinking water near one another']]
```

[11]:
```
# special tokens
PAD = b"#PAD#"
```

```
UNK = b"#UNK#"
START = b"#START#"
END = b"#END#"

# split sentence into tokens (split into lowercased words)
def split_sentence(sentence):
    return list(filter(lambda x: len(x) > 0, re.split('\W+', sentence.lower())))

def generate_vocabulary(train_captions):
    """
    Return {token: index} for all train tokens (words) that occur 5 times or more,
        `index` should be from 0 to N, where N is a number of unique tokens in the resulting dictionary.
    Use `split_sentence` function to split sentence into tokens.
    Also, add PAD (for batch padding), UNK (unknown, out of vocabulary),
        START (start of sentence) and END (end of sentence) tokens into the vocabulary.
    """
    sentences = tf.concat(train_captions, axis = -1) # flatten the sentences
    sentences = tf.strings.lower(sentences) # to lowercase
    sentences = tf.strings.regex_replace(sentences, '\W+',  ' ') # kep alphanumeric only
    vocabulary = tf.strings.split(sentences)
    vocabulary = tf.concat \
        ([x for x in vocabulary] + [tf.repeat([PAD, UNK, START, END], 5)], axis = -1)
        # split the sentences and append special tokens
    vocabulary = tf.unique(vocabulary)
    vocabulary = vocabulary.y[tf.math.bincount(vocabulary.idx) >= 5]
        # filter words occur less than 5 times
    return dict(zip(sorted(vocabulary.numpy()), range(len(vocabulary))))

def caption_tokens_to_indices(captions, vocab):
    """
    `captions` argument is an array of arrays:
    [
        [
            "image1 caption1",
            "image1 caption2",
            ...
        ],
        [
            "image2 caption1",
            "image2 caption2",
            ...
        ],
```

```
        ...
    ]
    Use `split_sentence` function to split sentence into tokens.
    Replace all tokens with vocabulary indices, use UNK for unknown words (out␣
↪of vocabulary).
    Add START and END tokens to start and end of each sentence respectively.
    For the example above you should produce the following:
    [
        [
            [vocab[START], vocab["image1"], vocab["caption1"], vocab[END]],
            [vocab[START], vocab["image1"], vocab["caption2"], vocab[END]],
            ...
        ],
        ...
    ]
    """
    return [to_indices(caption) for caption in captions]

def to_indices(caption):
    sentences = tf.strings.lower(caption)
    sentences = tf.strings.regex_replace(sentences, '\W+',  ' ')
    sentences = tf.strings.join([START, sentences, END], separator = ' ')
    return [
        vocabulary.loc[[token if token in vocabulary.index else UNK for token␣
↪in tokens.numpy()]] \
                  .values.ravel().tolist()
        for tokens in tf.strings.split(sentences)
    ]
```

[12]:
```
# prepare vocabulary
vocab = generate_vocabulary(train_captions)
vocab_inverse = {idx: w for w, idx in vocab.items()}
vocabulary = pd.DataFrame.from_dict(data = vocab, orient = 'index')
print(len(vocab))
```

```
8769
```

[13]:
```
# replace tokens with indices
train_captions_indexed = caption_tokens_to_indices(train_captions, vocab)
val_captions_indexed = caption_tokens_to_indices(val_captions, vocab)
```

We should keep the results of above calculations as it takes tons of time to process.

```
utils.save_pickle(vocab, 'vocab.pickle')
utils.save_pickle(train_captions_indexed, 'train_captions_indexed.pickle')
utils.save_pickle(val_captions_indexed, 'val_captions_indexed.pickle')
```

```
vocab = utils.read_pickle('vocab.pickle')
```

```
train_captions_indexed = utils.read_pickle('train_captions_indexed.pickle')
val_captions_indexed = utils.read_pickle('val_captions_indexed.pickle')
```

Captions have different length, but we need to batch them, that's why we will add PAD tokens so that all sentences have an equal length.

We will crunch LSTM through all the tokens, but we will ignore padding tokens during loss calculation.

[14]:
```
# we will use this during training
def batch_captions_to_matrix(batch_captions, pad_idx, max_len = None):
    """
    `batch_captions` is an array of arrays:
    [
        [vocab[START], ..., vocab[END]],
        [vocab[START], ..., vocab[END]],
        ...
    ]
    Put vocabulary indexed captions into np.array of shape␣
↪(len(batch_captions), columns),
        where "columns" is max(map(len, batch_captions)) when max_len is None
        and "columns" = min(max_len, max(map(len, batch_captions))) otherwise.
    Add padding with pad_idx where necessary.
    Input example: [[1, 2, 3], [4, 5]]
    Output example: np.array([[1, 2, 3], [4, 5, pad_idx]]) if max_len=None
    Output example: np.array([[1, 2], [4, 5]]) if max_len=2
    Output example: np.array([[1, 2, 3], [4, 5, pad_idx]]) if max_len=100
    Try to use numpy, we need this function to be fast!
    """
    max_len = min(max_len if max_len is not None else sys.maxsize, max(map(len,␣
↪batch_captions)))
    matrix = tf.ragged.constant(batch_captions) \
        .to_tensor(default_value = pad_idx, shape = (len(batch_captions),␣
↪max_len)) \
        .numpy()
    return matrix
```

[15]:
```
# make sure you use correct argument in caption_tokens_to_indices
assert len(caption_tokens_to_indices(train_captions[:10], vocab)) == 10
assert len(caption_tokens_to_indices(train_captions[:5], vocab)) == 5
```

9 Training

9.1 Define architecture

Since our problem is to generate image captions, RNN text generator should be conditioned on image. The idea is to use image features as an initial state for RNN instead of zeros.

Remember that you should transform image feature vector to RNN hidden state size by fully-connected layer and then pass it to RNN.

During training we will feed ground truth tokens into the lstm to get predictions of next tokens.

Notice that we don't need to feed last token (END) as input (http://cs.stanford.edu/people/karpathy/):

[16]:
```
IMG_EMBED_SIZE = train_img_embeds.shape[1]
IMG_EMBED_BOTTLENECK = 120
WORD_EMBED_SIZE = 100
LSTM_UNITS = 300
LOGIT_BOTTLENECK = 120
pad_idx = vocab[PAD]
tf.random.set_seed(42)
```

Here we define decoder graph.

We use Keras layers where possible because we can use them in functional style with weights reuse like this:

```
dense_layer = L.Dense(42, input_shape = (None, 100) activation = 'relu')
a = tf.placeholder('float32', [None, 100])
b = tf.placeholder('float32', [None, 100])
dense_layer(a)  # that's how we applied dense layer!
dense_layer(b)  # and again
```

Here's a figure to help you with flattening in decoder:

[17]:
```
class Decoder(keras.Model):

    def __init__(self, **kwargs):
        super(Decoder, self).__init__(**kwargs)

        # we use bottleneck here to reduce the number of parameters
        # image embedding -> bottleneck
        self.img_embed_to_bottleneck = L.Dense \
            (units = IMG_EMBED_BOTTLENECK, input_shape = (IMG_EMBED_SIZE,),␣
↪activation = 'elu')

        # image embedding bottleneck -> lstm initial state
        self.img_embed_bottleneck_to_h0 = L.Dense \
            (units = LSTM_UNITS, input_shape = (IMG_EMBED_BOTTLENECK,),␣
↪activation = 'elu')

        # word -> embedding
        self.word_embed = L.Embedding(input_dim = len(vocab), output_dim =␣
↪WORD_EMBED_SIZE)

        # lstm cell (from tensorflow)
```

```
        self.lstm = L.LSTM(units = LSTM_UNITS, return_state = True,␣
↪return_sequences = True)

        # we use bottleneck here to reduce model complexity
        # lstm output -> logits bottleneck
        self.token_logits_bottleneck = L.Dense \
            (units = LOGIT_BOTTLENECK, input_shape = (LSTM_UNITS,), activation␣
↪= "elu")

        # logits bottleneck -> logits for next token prediction
        self.token_logits = L.Dense(units = len(vocab), input_shape =␣
↪(LOGIT_BOTTLENECK,))

    def call(self, img_embeds, sentences):
        # initial lstm cell state of shape (None, LSTM_UNITS),
        # we need to condition it on `img_embeds` placeholder.
        self.c0 = self.h0 = self.img_embed_bottleneck_to_h0(self.
↪img_embed_to_bottleneck(img_embeds))

        # embed all tokens but the last for lstm input,
        # remember that L.Embedding is callable,
        # use `sentences` placeholder as input.
        self.word_embeds = self.word_embed(sentences[:, :-1])

        # during training we use ground truth tokens `word_embeds` as context␣
↪for next token prediction.
        # that means that we know all the inputs for our lstm and can get
        # all the hidden states with one tensorflow operation (tf.nn.
↪dynamic_rnn).
        # `hidden_states` has a shape of [batch_size, time steps, LSTM_UNITS].
        hidden_states, _, _ = self.lstm(self.word_embeds, initial_state = [self.
↪c0, self.h0])

        # now we need to calculate token logits for all the hidden states

        # first, we reshape `hidden_states` to [-1, LSTM_UNITS]
        self.flat_hidden_states = tf.reshape(tensor = hidden_states, shape =␣
↪(-1, LSTM_UNITS))

        # then, we calculate logits for next tokens using␣
↪`token_logits_bottleneck`
        # and `token_logits` layers
        self.flat_token_logits = self.token_logits(self.
↪token_logits_bottleneck(self.flat_hidden_states))

        # then, we flatten the ground truth token ids.
```

```
        # remember, that we predict next tokens for each time step,
        # use `sentences` placeholder.
        self.flat_ground_truth = tf.reshape(tensor = sentences[:, 1:], shape =␣
↪(-1))

        # we need to know where we have real tokens (not padding) in␣
↪`flat_ground_truth`,
        # we don't want to propagate the loss for padded output tokens,
        # fill `flat_loss_mask` with 1.0 for real tokens (not pad_idx) and 0.0␣
↪otherwise.
        self.flat_loss_mask = tf.not_equal(self.flat_ground_truth, pad_idx)

        # compute cross-entropy between `flat_ground_truth` and␣
↪`flat_token_logits` predicted by lstm
        xent = tf.nn.sparse_softmax_cross_entropy_with_logits \
            (labels = self.flat_ground_truth, logits = self.flat_token_logits)

        # compute average `xent` over tokens with nonzero `flat_loss_mask`.
        # we don't want to account misclassification of PAD tokens, because␣
↪that doesn't make sense,
        # we have PAD tokens for batching purposes only!
        self.loss = tf.reduce_mean(tf.boolean_mask(xent, self.flat_loss_mask))

        return self.loss

decoder = Decoder()
```

[18]:
```
def loss_fn(img_embeds, sentences):
    loss = decoder(img_embeds, sentences)
    return loss

# define optimizer operation to minimize the loss
optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
```

[19]:
```
def get_feed_dict_for_testing(decoder, IMG_EMBED_SIZE, vocab):
    return np.random.random((32, IMG_EMBED_SIZE)), np.random.randint(0,␣
↪len(vocab), (32, 20))

def test_decoder_shapes(decoder, IMG_EMBED_SIZE, vocab):
    decoder(*get_feed_dict_for_testing(decoder, IMG_EMBED_SIZE, vocab))
    tensors_to_test = [
        decoder.h0,
        decoder.word_embeds,
        decoder.flat_hidden_states,
        decoder.flat_token_logits,
        decoder.flat_ground_truth,
```

```
        decoder.flat_loss_mask,
        decoder.loss
    ]
    all_shapes = []
    for t in tensors_to_test:
        all_shapes.extend(t.shape)
    return all_shapes

def test_random_decoder_loss(decoder, IMG_EMBED_SIZE, vocab):
    decoder(*get_feed_dict_for_testing(decoder, IMG_EMBED_SIZE, vocab))
    loss = decoder.loss
    return loss.numpy()
```

9.2 Training loop

Evaluate train and validation metrics through training and log them. Ensure that loss decreases.

[20]:
```
train_captions_indexed = np.array(train_captions_indexed)
val_captions_indexed = np.array(val_captions_indexed)
```

[21]:
```
# generate batch via random sampling of images and captions for them,
# we use `max_len` parameter to control the length of the captions (truncating␣
 ↪long captions)
def generate_batch(images_embeddings, indexed_captions, batch_size, max_len =␣
 ↪None):
    """
    `images_embeddings` is a np.array of shape [number of images,␣
 ↪IMG_EMBED_SIZE].
    `indexed_captions` holds 5 vocabulary indexed captions for each image:
    [
        [
            [vocab[START], vocab["image1"], vocab["caption1"], vocab[END]],
            [vocab[START], vocab["image1"], vocab["caption2"], vocab[END]],
            ...
        ],
        ...
    ]
    Generate a random batch of size `batch_size`.
    Take random images and choose one random caption for each image.
    Remember to use `batch_captions_to_matrix` for padding and respect␣
 ↪`max_len` parameter.
    Return feed dict {decoder.img_embeds: ..., decoder.sentences: ...}.
    """
    idx = rng.choice(len(images_embeddings), size = batch_size, replace = False)
    batch_image_embeddings = images_embeddings[idx]
    batch_captions_matrix = batch_captions_to_matrix \
```

```
        ([rng.choice(caption) for caption in indexed_captions[idx]], pad_idx,␣
↪max_len = max_len)

    return batch_image_embeddings, batch_captions_matrix
```

[22]:
```
batch_size = 64
n_epochs = 12
n_batches_per_epoch = 1000
n_validation_batches = 100  # how many batches are used for validation after␣
↪each epoch
```

Look at the training and validation loss, they should be decreasing!

[23]:
```
# actual training loop
MAX_LEN = 20  # truncate long captions to speed up training

# to make training reproducible
np.random.seed(42)
random.seed(42)

for epoch in range(n_epochs):

    train_loss = 0
    pbar = tqdm_utils.tqdm_notebook_failsafe(range(n_batches_per_epoch))
    counter = 0
    for _ in pbar:
        batch_image_embeddings, batch_captions_matrix = \
            generate_batch(train_img_embeds,  train_captions_indexed,␣
↪batch_size, MAX_LEN)
        optimizer.minimize (
            loss = lambda: decoder(batch_image_embeddings,␣
↪batch_captions_matrix),
            var_list = decoder.trainable_weights
        )
        train_loss += decoder.loss
        counter += 1
        pbar.set_description("Training loss: %f" % (train_loss / counter))

    train_loss /= n_batches_per_epoch

    val_loss = 0
    for _ in range(n_validation_batches):
        batch_image_embeddings, batch_captions_matrix = \
            generate_batch(val_img_embeds, val_captions_indexed, batch_size,␣
↪MAX_LEN)

        loss = decoder(batch_image_embeddings, batch_captions_matrix)
```

```
        val_loss += loss
    val_loss /= n_validation_batches

    print('Epoch: {}, train loss: {}, val loss: {}'.format(epoch, train_loss,␣
↪val_loss))

    ## save weights after finishing epoch
    #saver.save(s, get_checkpoint_path(epoch))

print("Finished!")
```

```
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 0, train loss: 4.278500556945801, val loss: 3.6651928424835205
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 1, train loss: 3.3447935581207275, val loss: 3.1251626014709473
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 2, train loss: 3.0096945762634277, val loss: 2.9658167362213135
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 3, train loss: 2.8677706718444824, val loss: 2.8449149131774902
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 4, train loss: 2.7751481533050537, val loss: 2.7550652027130127
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 5, train loss: 2.7026963233947754, val loss: 2.743880033493042
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 6, train loss: 2.6514930725097656, val loss: 2.6967687606811523
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 7, train loss: 2.6040761470794678, val loss: 2.671525001525879
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 8, train loss: 2.5694146156311035, val loss: 2.6255404949188232
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 9, train loss: 2.5471935272216797, val loss: 2.6204166412353516
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 10, train loss: 2.521951198577881, val loss: 2.610685348510742
  0%|          | 0/1000 [00:00<?, ?it/s]
Epoch: 11, train loss: 2.508378505706787, val loss: 2.6173596382141113
Finished!
```

[24]:
```
def test_validation_loss(decoder, generate_batch, val_img_embeds,␣
↪val_captions_indexed):
    np.random.seed(300)
    random.seed(300)
    val_loss = 0
    batches_for_eval = 1000
    for _ in tqdm_utils.tqdm_notebook_failsafe(range(batches_for_eval)):
        val_loss += decoder(*generate_batch(val_img_embeds,␣
↪val_captions_indexed, 32, 20))
    val_loss /= 1000.
    return val_loss.numpy()
```

[25]:
```
# check that it's learnt something, outputs accuracy of next word prediction␣
↪(should be around 0.5)
from sklearn.metrics import accuracy_score, log_loss

def decode_sentence(sentence_indices):
    return b' '.join(map(vocab_inverse.get, sentence_indices))

def check_after_training(n_examples):
    fd = generate_batch(train_img_embeds, train_captions_indexed, batch_size)
    decoder(*fd)
    logits = decoder.flat_token_logits.numpy()
    truth = decoder.flat_ground_truth.numpy()
    mask = decoder.flat_loss_mask
    print('Loss:', decoder.loss)
    print('Accuracy:', accuracy_score(logits.argmax(axis = 1)[mask],␣
↪truth[mask]))
    for example_idx in range(n_examples):
        print('Example', example_idx)
        print('Predicted:', decode_sentence(logits.argmax(axis = 1).
↪reshape((batch_size, -1))[example_idx]))
        print('Truth:', decode_sentence(truth.reshape((batch_size,␣
↪-1))[example_idx]))
        print('')

check_after_training(3)
```

```
Loss: tf.Tensor(2.4288654, shape=(), dtype=float32)
Accuracy: 0.47009735744089015
Example 0
Predicted: b'a pizza with is sitting a plate with a table #END# #END# #END#
#END# #END# and and and and and and and and and'
Truth: b'a pizza that is on a plate on a table #END# #PAD# #PAD# #PAD# #PAD#
#PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#'

Example 1
```

```
Predicted: b'a elephant is a trunk standing on the road elephant #END# a back
#END# #END# #END# #END# #END# #END# #END# #END# #END# #END# #END#'
Truth: b'an elephant with a person lying n a red basket on its back #END# #PAD#
#PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#'

Example 2
Predicted: b'a in down the street holding on a #END# #END# #END# #END# #END#
#END# #END# #END# #END# #END# #END# #END# #END# #END# #END# #END#'
Truth: b'man walking through city while talking on phone #END# #PAD# #PAD# #PAD#
#PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#'
```

10 Applying model

Here we construct a graph for our final model.

It will work as follows: - take an image as an input and embed it - condition lstm on that embedding - predict the next token given a START input token - use predicted token as an input at next time step - iterate until you predict an END token

[26]:
```
# CNN encoder
encoder, preprocess_for_model = get_cnn_encoder()

class Encoder(keras.Model):

    def __init__(self, input_images, **kwargs):
        super(Encoder, self).__init__(**kwargs)

        # get image embeddings
        img_embeds = encoder(input_images)

        # initialize lstm state conditioned on image
        self.lstm_c = self.lstm_h = \
            decoder.img_embed_bottleneck_to_h0(decoder.
↪img_embed_to_bottleneck(img_embeds))

    def call(self, current_word):
        # embedding for current word
        word_embed = decoder.word_embed(current_word)

        # apply lstm cell, get new lstm states
        new_h, self.lstm_c, self.lstm_h = \
            decoder.lstm(word_embed, initial_state = [self.lstm_c, self.lstm_h])

        # compute logits for next token
        new_logits = decoder.token_logits(decoder.
↪token_logits_bottleneck(new_h))
```

```
        # compute probabilities for next token
        new_probs = tf.nn.softmax(new_logits)

        # `one_step` outputs probabilities of next token and updates lstm␣
↪hidden state
        one_step = new_probs, self.lstm_c, self.lstm_h

        return one_step
```

```
WARNING:tensorflow:From <ipython-input-5-08e818f306b7>:3: set_learning_phase
(from tensorflow.python.keras.backend) is deprecated and will be removed after
2020-10-11.
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__`
method of your layer or model.
```

[27]:
```
# look at how temperature works for probability distributions
# for high temperature we have more uniform distribution
_ = np.array([0.5, 0.4, 0.1])
for t in [0.01, 0.1, 1, 10, 100]:
    print(" ".join(map(str, _ ** (1 / t) / np.sum(_ ** (1 / t)))), "with␣
↪temperature", t)
```

```
0.9999999997962965 2.0370359759195462e-10 1.2676505999700117e-70 with
temperature 0.01
0.9030370433250645 0.09696286420394223 9.247099323648666e-08 with temperature
0.1
0.5 0.4 0.1 with temperature 1
0.35344772639219624 0.34564811360592396 0.3009041600018798 with temperature 10
0.33536728048099185 0.33461976434857876 0.3300129551704294 with temperature 100
```

[28]:
```
# this is an actual prediction loop
def generate_caption(image, t = 1, sample = False, max_len = 20):
    """
    Generate caption for given image.
    if `sample` is True, we will sample next token from predicted probability␣
↪distribution.
    `t` is a temperature during that sampling,
        higher `t` causes more uniform-like distribution = more chaos.
    """
    # condition lstm on the image
    final_model = Encoder(image[None, ...])

    # current caption
    # start with only START token
    caption = [vocab[START]]
```

```
    for _ in range(max_len):
        next_word_probs, _, _ = final_model(tf.expand_dims(input =␣
↪[caption[-1]], axis = 0))
        next_word_probs = tf.reshape(next_word_probs, shape = (-1))

        # apply temperature
        next_word_probs = next_word_probs ** (1 / t) / np.sum(next_word_probs␣
↪** (1 / t))

        if sample:
            next_word = np.random.choice(range(len(vocab)), p = next_word_probs)
        else:
            next_word = np.argmax(next_word_probs)

        caption.append(next_word)
        if next_word == vocab[END]:
            break

    return list(map(vocab_inverse.get, caption))
```

[29]:
```
# look at validation prediction example
def apply_model_to_image_raw_bytes(raw):
    img = utils.decode_image_from_buf(raw)
    fig = plt.figure(figsize = (7, 7))
    plt.grid('off')
    plt.axis('off')
    plt.imshow(img)
    img = utils.crop_and_preprocess(img, (IMG_SIZE, IMG_SIZE),␣
↪preprocess_for_model)
    print(b' '.join(generate_caption(img)[1:-1]))
    plt.show()

def show_valid_example(val_img_fns, example_idx = 0):
    zf = zipfile.ZipFile("val2014_sample.zip")
    all_files = set(val_img_fns)
    found_files = list(filter(lambda x: x.filename.rsplit("/")[-1] in␣
↪all_files, zf.filelist))
    example = found_files[example_idx]
    apply_model_to_image_raw_bytes(zf.read(example))

show_valid_example(val_img_fns, example_idx = 100)
```

```
b'a baseball player holding a bat in a baseball game'
```

[30]:
```
# sample more images from validation
for idx in np.random.choice(range(len(zipfile.ZipFile("val2014_sample.zip").
 ↪filelist) - 1), 10):
    show_valid_example(val_img_fns, example_idx = idx)
    time.sleep(1)
```

```
b'a man is riding a skateboard on a ramp'
```

b'a large jetliner flying through a cloudy sky'

b'a woman standing in a kitchen with a stove top'

b'a street sign with a street sign and a street sign'


```
b'a table with a table and a table with a table and a table'
```

b'a group of people on a boat with a boat in the water'

b'a man riding skis down a snow covered slope'

b'a man is riding a skateboard on a ramp'

b'a person holding a cell phone in their lap top'

b'a soccer goalie is about to kick the ball'

You can download any image from the Internet and appply your model to it!

[31]:
```
download_utils.download_file (
    "http://www.bijouxandbits.com/wp-content/uploads/2016/06/portal-cake-10.
 ↪jpg",
    "portal-cake-10.jpg"
)
```

```
  0%|          | 0.00/273k [00:00<?, ?B/s]
```

[32]:
```
apply_model_to_image_raw_bytes(open("portal-cake-10.jpg", "rb").read())
```

```
b'a cake with a knife and a fork'
```

Now it's time to find 10 examples where your model works good and 10 examples where it fails!

You can use images from validation set as follows:

```
show_valid_example(val_img_fns, example_idx=...)
```

You can use images from the Internet as follows:

```
! wget ...
apply_model_to_image_raw_bytes(open("...", "rb").read())
```

If you use these functions, the output will be embedded into your notebook and will be visible during peer review!

When you're done, download your noteboook using "File" -> "Download as" -> "Notebook" and prepare that file for peer review!

10.0.1 Good samples

[33]:
```
show_valid_example(val_img_fns, example_idx = 10)
```

```
b'a street sign with a street sign and a street sign'
```

[34]:
```
show_valid_example(val_img_fns, example_idx = 20)
```

```
b'a man riding skis down a snow covered slope'
```

[36]:
```
show_valid_example(val_img_fns, example_idx = 50)
```

```
b'a plate of food with a bunch of vegetables'
```

[39]:
```
show_valid_example(val_img_fns, example_idx = 110)
```

```
b'a large jetliner sitting on top of a runway'
```

[41]:
```
show_valid_example(val_img_fns, example_idx = 140)
```

```
b'a man is playing tennis on a tennis court'
```

[42]:
```
show_valid_example(val_img_fns, example_idx = 150)
```

```
b'a herd of sheep grazing on a lush green field'
```

[43]:
```
show_valid_example(val_img_fns, example_idx = 190)
```

```
b'a bunch of vegetables and vegetables on a table'
```

[44]:
```
show_valid_example(val_img_fns, example_idx = 40)
```

```
b'a hot dog with ketchup and cheese on a bun'
```

[45]:
```
show_valid_example(val_img_fns, example_idx = 60)
```

```
b'a group of people waiting at a train station'
```

[50]:
```
show_valid_example(val_img_fns, example_idx = 180)
```

```
b'a group of people standing around a table with a sign'
```

10.0.2 Failed Samples

[35]:
```
show_valid_example(val_img_fns, example_idx = 30)
```

```
b'a motorcycle parked in a parking lot next to a car'
```

[37]:
```
show_valid_example(val_img_fns, example_idx = 70)
```

```
b'a bedroom with a bed and a table in it'
```

[38]:
```
show_valid_example(val_img_fns, example_idx = 80)
```

```
b'a baby is sitting on a couch with a stuffed animal'
```

[40]:
```
show_valid_example(val_img_fns, example_idx = 120)
```

```
b'a cat is laying on a bed with a cat'
```

[46]:
```
show_valid_example(val_img_fns, example_idx = 90)
```

```
b'a polar bear laying on a rock next to a ball'
```

[47]:
```
show_valid_example(val_img_fns, example_idx = 130)
```

```
b'a man is standing in a kitchen with a large clock'
```

[48]:
```
show_valid_example(val_img_fns, example_idx = 160)
```

```
b'a group of people walking down a street'
```

[49]:
```
show_valid_example(val_img_fns, example_idx = 170)
```

```
b'a person jumping a kite in a park'
```

[51]:
```
show_valid_example(val_img_fns, example_idx = 200)
```

```
b'a man in a black shirt and a black shirt and a black and white photo'
```

[52]:
```
show_valid_example(val_img_fns, example_idx = 210)
```

```
b'a person holding a cell phone in their hand'
```

That's it!

Congratulations, you've trained your image captioning model and now can produce captions for any picture from the Internet!