Image_Captioning_Project

December 4, 2020

1 Image Captioning Project

For this project, I am using Machine Learning algorithms to auto-generate image captions.

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

2 Import Dependencies

[9]:
```
import sys
sys.path.append("..")
import download_utils
```

[10]:
```
#download_utils.link_all_keras_resources()
```

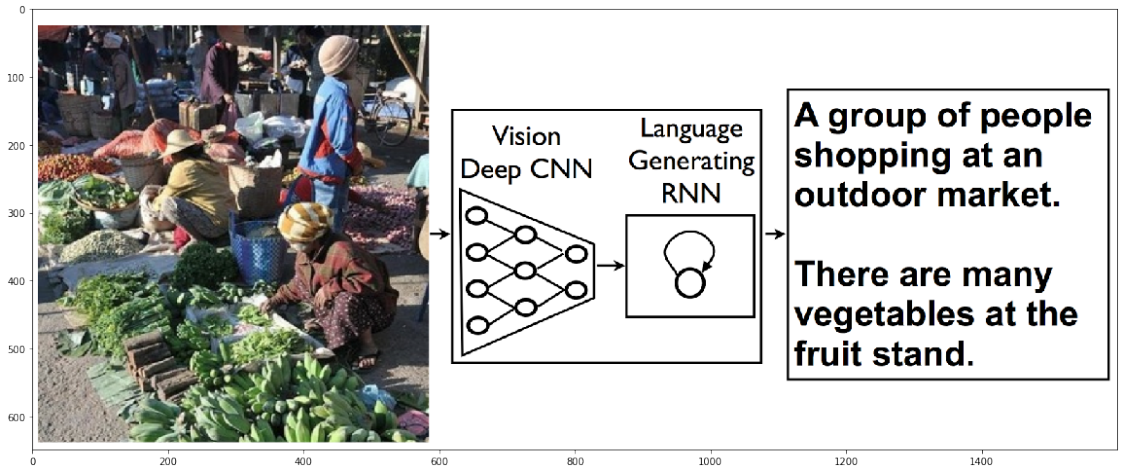
[11]:
```
import tensorflow as tf
from tensorflow.contrib import keras
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
L = keras.layers
K = keras.backend
import utils
import time
import zipfile
import json
from collections import defaultdict
import re
import random
from random import choice
import os
#from keras_utils import reset_tf_session
import tqdm_utils
```

3 How it Works

The training data consists of many images, each with several captions linked to it. The model feeds the images into a Convolutional Neural Network (CNN) that is linked to a Recurrent Neural Network (RNN) that ultimately generates a caption. The model adjusts its weights based upon how closely the result matches the captions from the training data.

[24]:
```
f = "images/encoder_decoder.png"
fig = plt.figure(figsize=(10, 6))
img=plt.imread(f)
plt.imshow(img)
```

[24]:
```
<matplotlib.image.AxesImage at 0x7ff808b34a58>
```



[6]:
```
def reset_tf_session():
    curr_session = tf.get_default_session()
    # close current session
    if curr_session is not None:
        curr_session.close()
    # reset graph
    K.clear_session()
    # create new session
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    s = tf.InteractiveSession(config=config)
    K.set_session(s)
    return s# show random images from train
```

4 Prepare the storage for model checkpoints

[7]:
```
CHECKPOINT_ROOT = ""

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))
```

```
/home/chmpearson/intro-to-dl/week6/weights_10
```

5 Download data

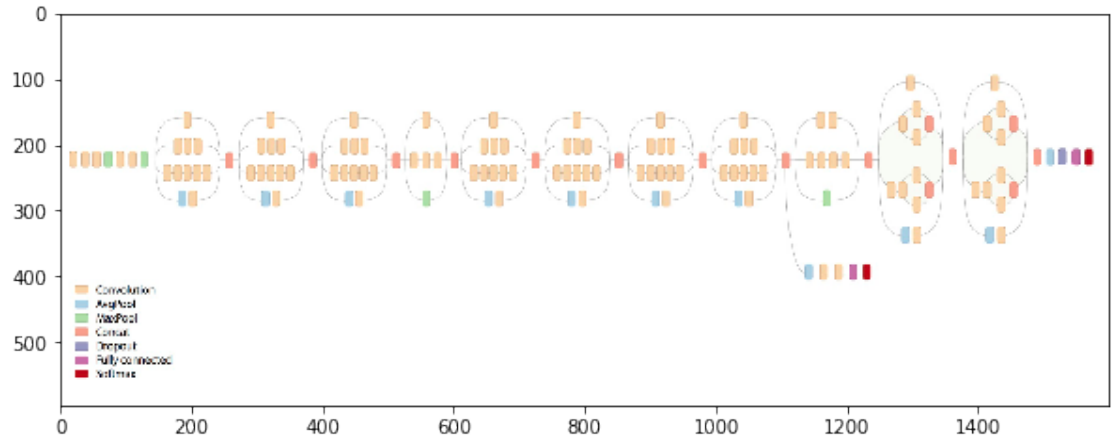
To download the data used to train and validate the model, you may use the following links. Be warned: togheter these are approximatley 20gb. Relevant links (just in case): - train images http://msvocds.blob.core.windows.net/coco2014/train2014.zip - validation images http://msvocds.blob.core.windows.net/coco2014/val2014.zip - captions for both train and validation http://msvocds.blob.core.windows.net/annotations-1-0-3/captions_train-val2014.zip

6 Extract image features

Rather than train the entire model locally, we will be using a Google's pre-trained InceptionV3 model for the CNN encoder. In this case, we will be extracting the weights for it slast hidden layer and embedding it in our own model. (https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html)

[27]:
```
f = "images/inceptionv3.png"
fig = plt.figure(figsize=(10, 6))
img=plt.imread(f)
plt.imshow(img)
```

[27]:
```
<matplotlib.image.AxesImage at 0x7ff805edc710>
```



[8]:
```
IMG_SIZE = 299
```

[9]:
```
# 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
```

The following code has already run, but can take several hours. It is available here if you would like to try it:

```
# 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")
```

[10]:
```
# 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
```

[11]:
```
# check prepared samples of images
list(filter(lambda x: x.endswith("_sample.zip"), os.listdir(".")))
```

[11]:
```
['train2014_sample.zip', 'val2014_sample.zip']
```

7 Extract captions for images

[13]:
```
# 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))
```

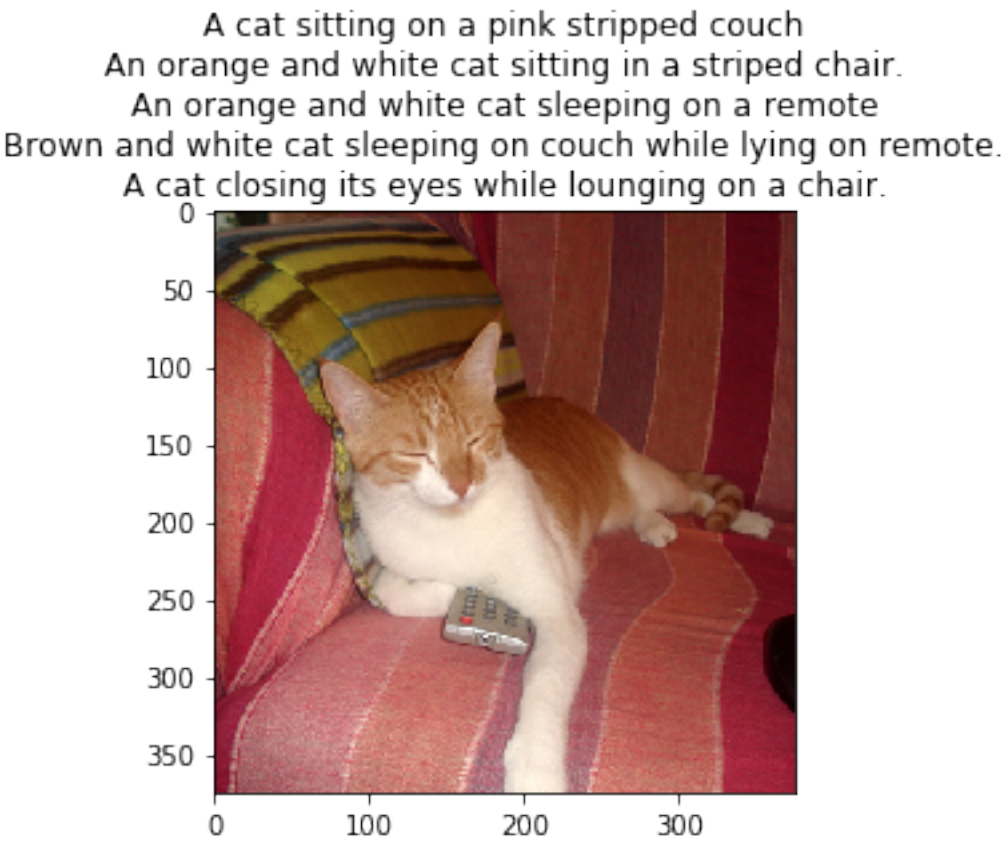
```
82783 82783
40504 40504
```

8 Goal

Next, lets look at a training example that someone else has already done so we have a clear idea of what our results should look like.

[14]:
```
# 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)
```



9 Prepare captions for training

[15]:
```
# preview captions data
train_captions[:2]
```

[15]:
```
[['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']]
```

[16]:
```
# special tokens
PAD = "#PAD#"
```

```
UNK = "#UNK#"
START = "#START#"
END = "#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):
    temp = {}
    vocab = {'#PAD#': 0,'#UNK#': 1,'#START#': 2,'#END#': 3}
    index = 4
    for caption in train_captions:
        for sentence in caption:
            for word in split_sentence(sentence):
                if word not in vocab:
                    if word not in temp:
                        temp[word] = 1
                    elif temp[word] == 4:
                        vocab[word] = index
                        index += 1
                    else:
                        temp[word] += 1
    return {token: index for index, token in enumerate(sorted(vocab))}

def caption_tokens_to_indices(captions, vocab):
    res = []
    for caption in captions:
        new_caption = []
        for sentence in caption:
            new_sentence = [vocab['#START#'] ]
            for word in split_sentence(sentence):
                if word in vocab:
                    new_sentence.append(vocab[word] )
                else:
                    new_sentence.append(vocab['#UNK#'] )
            new_sentence.append(vocab['#END#'])
            new_caption.append(new_sentence)
        res.append(new_caption)
    return res
```

[17]:
```
# prepare vocabulary
vocab = generate_vocabulary(train_captions)
vocab_inverse = {idx: w for w, idx in vocab.items()}
print(len(vocab))
```

```
8769
```

[18]:
```
# replace tokens with indices
train_captions_indexed = caption_tokens_to_indices(train_captions, vocab)
val_captions_indexed = caption_tokens_to_indices(val_captions, vocab)
```

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.

[19]:
```
# we will use this during training
def batch_captions_to_matrix(batch_captions, pad_idx, max_len=None):
    rows = len(batch_captions)
    if (max_len is None):
        columns = max(map(len, batch_captions) )
    else:
        columns = min ( max_len, max(map(len, batch_captions) )  )
    matrix = np.full( (rows, columns), pad_idx )
    for i, sentence in enumerate(batch_captions):
        length = len(sentence)
        if columns >= length:
            matrix[i,0:length ] = sentence
        else:
            matrix[i,:] = sentence[:columns]
    return matrix
```

[40]:
```
# 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
```

10 Training

10.1 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.

To do this, we should transform the image feature vector to the RNN hidden state size via a fully-connected layer and then pass it to the RNN.

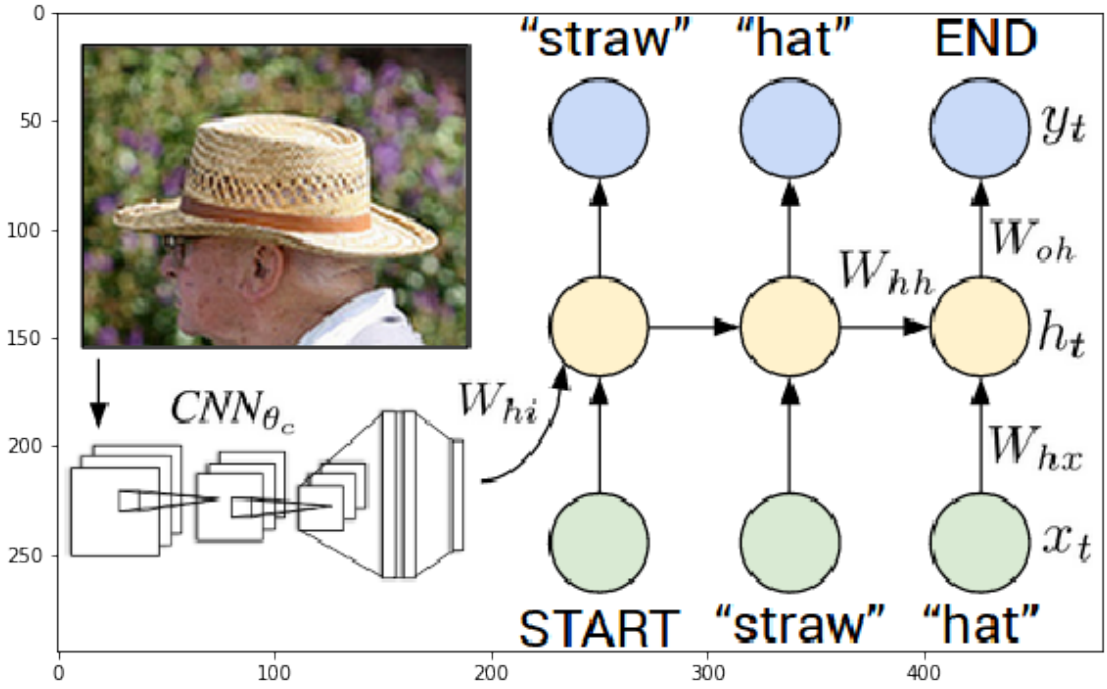
During training we will feed ground truth tokens into the LSTM to get predictions for the subsequent tokens.

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

[28]:
```
f = "images/encoder_decoder_explained.png"
fig = plt.figure(figsize=(10, 6))
```

```
img=plt.imread(f)
plt.imshow(img)
```

[28]: <matplotlib.image.AxesImage at 0x7ff805e862e8>



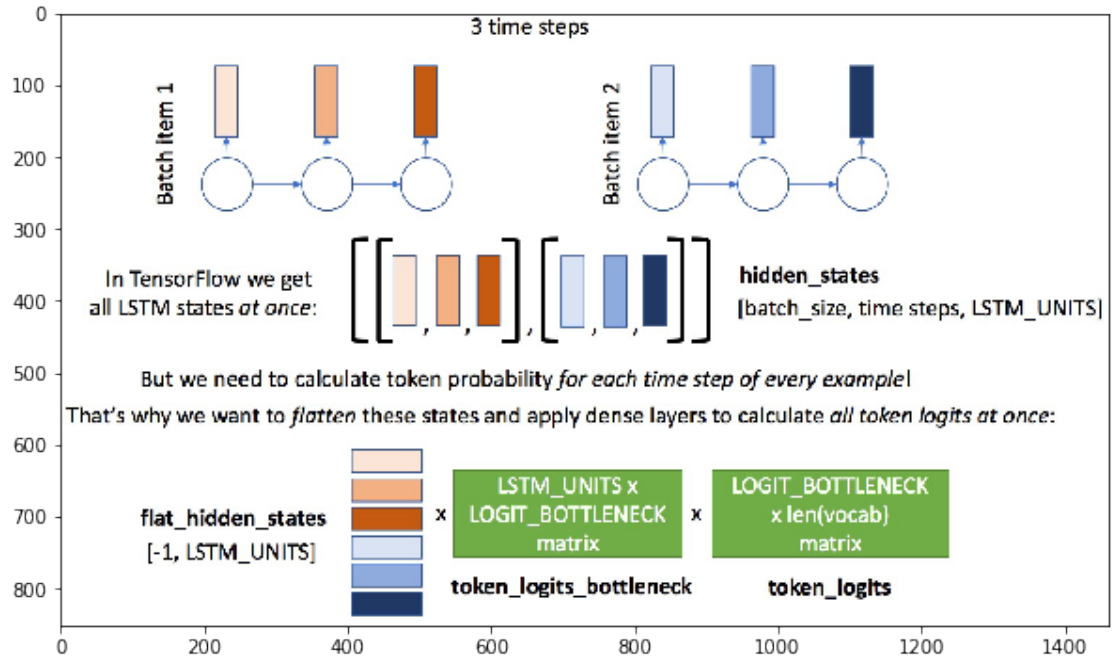
[41]:
```
IMG_EMBED_SIZE = train_img_embeds.shape[1]
IMG_EMBED_BOTTLENECK = 120
WORD_EMBED_SIZE = 100
LSTM_UNITS = 300
LOGIT_BOTTLENECK = 120
pad_idx = vocab[PAD]
```

[42]:
```
# remember to reset your graph if you want to start building it from scratch!
s = reset_tf_session()
tf.set_random_seed(42)
```

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

[30]:
```
f = "images/flatten_help.jpg"
fig = plt.figure(figsize=(10, 6))
img=plt.imread(f)
plt.imshow(img)
```

[30]: <matplotlib.image.AxesImage at 0x7ff805a6ed30>



[43]:
```
s = reset_tf_session()
tf.set_random_seed(42)
class decoder:
    # [batch_size, IMG_EMBED_SIZE] of CNN image features
    img_embeds = tf.placeholder('float32', [None, IMG_EMBED_SIZE])
    # [batch_size, time steps] of word ids
    sentences = tf.placeholder('int32', [None, None])

    # we use bottleneck here to reduce the number of parameters
    # image embedding -> bottleneck
    img_embed_to_bottleneck = L.Dense(IMG_EMBED_BOTTLENECK,
                                      input_shape=(None, IMG_EMBED_SIZE),
                                      activation='elu')
    # image embedding bottleneck -> lstm initial state
    img_embed_bottleneck_to_h0 = L.Dense(LSTM_UNITS,
                                         input_shape=(None,␣
↪IMG_EMBED_BOTTLENECK),
                                         activation='elu')
    # word -> embedding
    word_embed = L.Embedding(len(vocab), WORD_EMBED_SIZE)
    # lstm cell (from tensorflow)
    lstm = tf.nn.rnn_cell.LSTMCell(LSTM_UNITS)

    # we use bottleneck here to reduce model complexity
    # lstm output -> logits bottleneck
```

```
    token_logits_bottleneck = L.Dense(LOGIT_BOTTLENECK,
                                      input_shape=(None, LSTM_UNITS),
                                      activation="elu")
    # logits bottleneck -> logits for next token prediction
    token_logits = L.Dense(len(vocab),
                           input_shape=(None, LOGIT_BOTTLENECK))

    # initial lstm cell state of shape (None, LSTM_UNITS),
    # we need to condition it on `img_embeds` placeholder.
    c0 = h0 = img_embed_bottleneck_to_h0(  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.
    word_embeds = 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, _ = tf.nn.dynamic_rnn(lstm, word_embeds,
                                         initial_state=tf.nn.rnn_cell.LSTMStateTuple(c0, h0))

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

    # first, we reshape `hidden_states` to [-1, LSTM_UNITS]
    flat_hidden_states = tf.reshape( hidden_states, (-1, LSTM_UNITS)  )

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

    # then, we flatten the ground truth token ids.
    # remember, that we predict next tokens for each time step,
    # use `sentences` placeholder.
    flat_ground_truth = tf.reshape( sentences[:, 1:], (-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.
```

```
    flat_loss_mask = tf.math.not_equal(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=flat_ground_truth,
        logits=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!
    loss = tf.reduce_mean(  tf.boolean_mask( xent, flat_loss_mask )  )
```

```
WARNING:tensorflow:From /home/chmpearson/.local/lib/python3.6/site-
packages/tensorflow/python/ops/init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From /home/chmpearson/.local/lib/python3.6/site-
packages/tensorflow/python/keras/initializers.py:119: calling
RandomUniform.__init__ (from tensorflow.python.ops.init_ops) with dtype is
deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From <ipython-input-43-d6f2f1f6aa23>:21: LSTMCell.__init__
(from tensorflow.python.ops.rnn_cell_impl) is deprecated and will be removed in
a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.LSTMCell, and will be replaced by
that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-43-d6f2f1f6aa23>:46: dynamic_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.RNN(cell)`, which is equivalent to this API
WARNING:tensorflow:From /home/chmpearson/.local/lib/python3.6/site-
packages/tensorflow/python/ops/rnn_cell_impl.py:961: calling Zeros.__init__
(from tensorflow.python.ops.init_ops) with dtype is deprecated and will be
removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
```

```
WARNING:tensorflow:Entity <bound method LSTMCell.call of
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7fec5ce6b278>> could
not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7fec5ce6b278>>: AttributeError: module 'gast' has no attribute 'Num'
WARNING: Entity <bound method LSTMCell.call of
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7fec5ce6b278>> could
not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7fec5ce6b278>>: AttributeError: module 'gast' has no attribute 'Num'
WARNING:tensorflow:From /home/chmpearson/.local/lib/python3.6/site-
packages/tensorflow/python/ops/array_ops.py:1354:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

[44]:
```
# define optimizer operation to minimize the loss
optimizer = tf.train.AdamOptimizer(learning_rate=0.001)
train_step = optimizer.minimize(decoder.loss)

# will be used to save/load network weights.
# you need to reset your default graph and define it in the same way to be able␣
 ↪to load the saved weights!
saver = tf.train.Saver()

# intialize all variables
s.run(tf.global_variables_initializer())
```

10.2 Training loop

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

[47]:
```
train_captions_indexed = np.array(train_captions_indexed)
val_captions_indexed = np.array(val_captions_indexed)
```

[48]:
```
# 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 only_one(l_list):
#    return(  l_list[np.random.choice(  np.arange( len(l_list) ),1  )]  )
```

```
def generate_batch(images_embeddings, indexed_captions, batch_size,␣
↪max_len=None):
    these = np.random.choice(  np.arange(images_embeddings.shape[0]),␣
↪batch_size  )
    batch_image_embeddings = images_embeddings[these]

#    these_captions = np.random.choice(  np.arange(5), batch_size  )

    batch_captions_matrix = batch_captions_to_matrix(  list(map( random.choice,␣
↪indexed_captions[these] )), pad_idx = pad_idx, max_len = max_len  )

    return {decoder.img_embeds: batch_image_embeddings,
            decoder.sentences: batch_captions_matrix}
```

[50]:
```
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
```

[51]:
```
# you can load trained weights here
# uncomment the next line if you need to load weights
# saver.restore(s, get_checkpoint_path(epoch=4))
```

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

[56]:
```
# 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 " ".join(list(map(vocab_inverse.get, sentence_indices)))

def check_after_training(n_examples):
    fd = generate_batch(train_img_embeds, train_captions_indexed, batch_size)
    logits = decoder.flat_token_logits.eval(fd)
    truth = decoder.flat_ground_truth.eval(fd)
    mask = decoder.flat_loss_mask.eval(fd).astype(bool)
    print("Loss:", decoder.loss.eval(fd))
    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: 3.0042758
Accuracy: 0.40629095674967236
Example 0
Predicted: a man up of a man in a and a cell #END# #END# #END# #END# #END# #END#
#END# #END# #END#
Truth: a close up of a man wearing glasses and a tie #END# #PAD# #PAD# #PAD#
#PAD# #PAD# #PAD# #PAD# #PAD#

Example 1
Predicted: a image train and white train of a train engine #END# #END# a road
#END# #END# #END# #END# #END# #END#
Truth: an old black and white image of a steam engine moving along the tracks
#END# #PAD# #PAD# #PAD# #PAD# #PAD#

Example 2
Predicted: a kitchen with a cabinets and a cabinets #END# #END# #END# #END#
#END# #END# #END# #END# #END# #END# #END# #END#
Truth: a kitchen with wood cabinets and updated appliances #END# #PAD# #PAD#
#PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#
```

[57]:
```
# save last graph weights to file!
# saver.save(s, get_checkpoint_path())
# saved 3-29-2020, 6:31 PM
```

11 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

[58]:
```
class final_model:
    # CNN encoder
    encoder, preprocess_for_model = get_cnn_encoder()
    saver.restore(s, get_checkpoint_path())  # keras applications corrupt our␣
 ↪graph, so we restore trained weights

    # containers for current lstm state
    lstm_c = tf.Variable(tf.zeros([1, LSTM_UNITS]), name="cell")
    lstm_h = tf.Variable(tf.zeros([1, LSTM_UNITS]), name="hidden")
```

```
    # input images
    input_images = tf.placeholder('float32', [1, IMG_SIZE, IMG_SIZE, 3],␣
↪name='images')

    # get image embeddings
    img_embeds = encoder(input_images)

    # initialize lstm state conditioned on image
    init_c = init_h = decoder.img_embed_bottleneck_to_h0(decoder.
↪img_embed_to_bottleneck(img_embeds))
    init_lstm = tf.assign(lstm_c, init_c), tf.assign(lstm_h, init_h)

    # current word index
    current_word = tf.placeholder('int32', [1], name='current_input')

    # embedding for current word
    word_embed = decoder.word_embed(current_word)

    # apply lstm cell, get new lstm states
    new_c, new_h = decoder.lstm(word_embed, tf.nn.rnn_cell.
↪LSTMStateTuple(lstm_c, lstm_h))[1]

    # 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, tf.assign(lstm_c, new_c), tf.assign(lstm_h, new_h)
```

```
WARNING:tensorflow:From /home/chmpearson/.local/lib/python3.6/site-
packages/tensorflow/python/training/saver.py:1276: checkpoint_exists (from
tensorflow.python.training.checkpoint_management) is deprecated and will be
removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from /home/chmpearson/intro-to-
dl/week6/weights
WARNING:tensorflow:Entity <bound method LSTMCell.call of
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7fec5ce6b278>> could
not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7fec5ce6b278>>: AttributeError: module 'gast' has no attribute 'Num'
WARNING: Entity <bound method LSTMCell.call of
```

```
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7fec5ce6b278>> could
not be transformed and will be executed as-is. Please report this to the
AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound
method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7fec5ce6b278>>: AttributeError: module 'gast' has no attribute 'Num'
```

[59]:
```
# 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
```

[60]:
```
# 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
    s.run(final_model.init_lstm,
          {final_model.input_images: [image]})

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

    for _ in range(max_len):
        next_word_probs = s.run(final_model.one_step,
                                {final_model.current_word: [caption[-1]]})[0]
        next_word_probs = next_word_probs.ravel()

        # 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))
```

12 Initial Tests

Here, we will try testing the model with pictures that others have auto-generated captions for in the past.

[61]:
```
# 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), final_model.
 ↪preprocess_for_model)
    print(' '.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)
```

```
a baseball player is swinging at a ball
```

[62]:
```
# 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)
```

```
a man riding a motorcycle on a dirt road
```

a hot dog with ketchup and ketchup on a bun

a woman and a woman are holding a cake

a group of people standing in a room

a plane is flying in the air on a cloudy day

a bicycle is chained to a fire hydrant

a group of people sitting around a table eating food

a man riding a motorcycle down a street

a boat is docked on the water near a lake

a man in a blue jacket is snowboarding down a snowy hill

13 Our Pictures

13.1 Where the Model Worked Well:

[75]:
```
for filename in os.listdir('project_images/good'):
    f = os.path.join('project_images/good', filename)
    apply_model_to_image_raw_bytes(  open( f, 'rb' ).read()  )
```

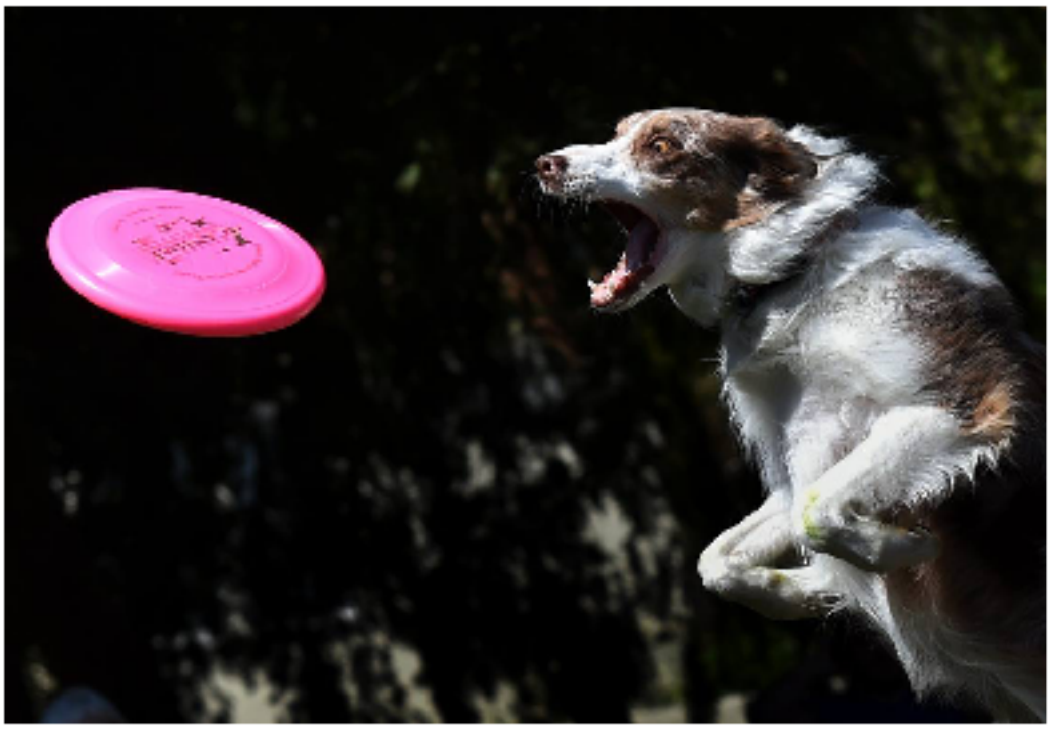
```
a bunch of apples sitting on a table
```

a man riding on the back of a horse

a group of people standing next to each other

a dog is playing with a frisbee in the grass

a dog is standing in the grass with a frisbee



a woman in a cowboy hat holding a horse

a pan of food is on a table

a group of men standing next to each other

a group of young men playing soccer on a field

a woman in a blue shirt is playing tennis

13.2 Where the Model Worked Poorly

Here, we will look at cases where the model generated captions that...were not very accurate.

[74]:
```
for filename in os.listdir('project_images/bad'):
    f = os.path.join('project_images/bad', filename)
    apply_model_to_image_raw_bytes(  open( f, 'rb' ).read()  )

#apply_model_to_image_raw_bytes(  open( "project_images/swimmer_phelps.jpg",␣
↪'rb' ).read()  )

#apply_model_to_image_raw_bytes(  open( "project_images/Scrooge_McDuck.png",␣
↪'rb' ).read()  )
```

```
a white and white cat sitting on a wooden table
```

a clock tower with a clock on the top of it

a woman with a beard and a beard holding a pink umbrella

a man with a hat and a dog on a leash

a close up of a pair of orange scissors

a young boy is playing with a tennis racket

a person is standing in the middle of a field

a large elephant standing on top of a body of water

a man is riding a surfboard on a wave

a large bird sitting on top of a boat

14 Conclusion

The model worked, but it is interesting to look at the cases where it worked well, vs the cases where it did poorly. One way I tried to test the model was by mixing in drawings with the photographs. Of the three included, one was accurate, and two not: the worst being a sketch. That the model could handle pictures animated pictures should not be too much of a surprise though, as when a CNN breaks down a picture, style is just one of these layers–which is why software that takes a photograph and stylizes it like a painting can work.

The model worked quite well when it comes to food and domestic animals, suggesting that in training there were quite a few pictures of these sorts. However, historic sites were another matter: the model was befuddled by the two ships and the castle.

[ ]: