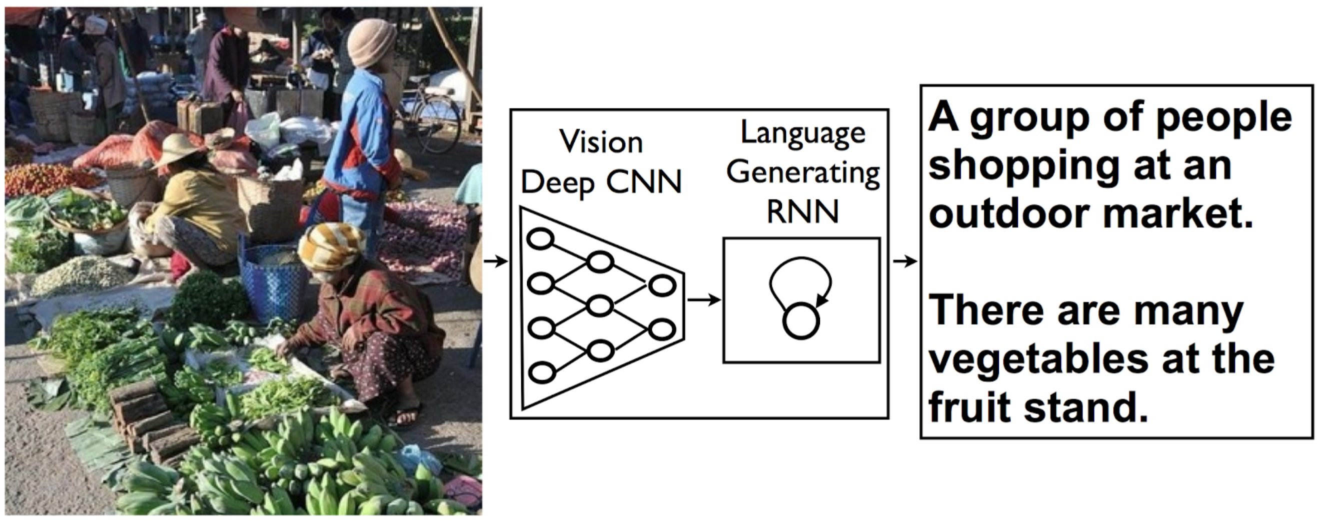
In [1]:

```
# set tf 1.x for colab
%tensorflow_version 1.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

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!


Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

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

Import stuff

In [2]:

```
import sys
sys.path.append("..")
import grading
import download_utils
```

In [3]:

```
download_utils.link_all_keras_resources()
```

In [4]:

```
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 grading_utils
import os
from keras_utils import reset_tf_session
import tqdm_utils
```

```
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type,
1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing
```

Prepare the storage for model checkpoints

In [5]:

```
# 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))
```

```
D:\Google Drive\Study\Advanced Machine Learning - Coursera\1.intro-to-dl\week6\weights_10
```

Fill in your Coursera token and email

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

In [6]:

```
grader = grading.Grader(assignment_key="NEDBg6CgEee8nQ6uE8a7OA",
                        all_parts=["19Wpv", "uJh73", "yiJkt", "rbpnH", "E2OIL", "YJR7z"])
```

In [7]:

```
# token expires every 30 min
COURSERA_TOKEN = 'gu6esl2ZRKHnD8sv'
COURSERA_EMAIL = 'lxwvictor@gmail.com'
```

Download data

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

Relevant links (just in case):

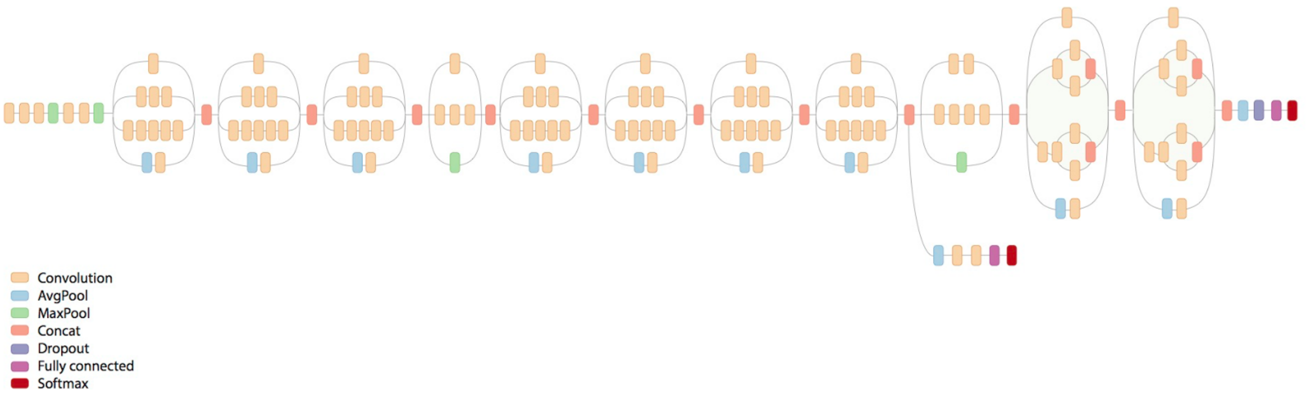
- train images http://msvocds.blob.core.windows.net/coco2014/train2014.zip (http://msvocds.blob.core.windows.net/coco2014/train2014.zip)
- validation images http://msvocds.blob.core.windows.net/coco2014/val2014.zip (http://msvocds.blob.core.windows.net/coco2014/val2014.zip)
- captions for both train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

In [8]:

```
# we downloaded them for you, just link them here
download_utils.link_week_6_resources()
```

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 (https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html)) and extract its last hidden layer as an embedding:



In [9]:

```
IMG_SIZE = 299
```

In [10]:

```
# 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")
```

In [11]:

```
# 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
```

In [12]:

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

Out[12]:

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

Extract captions for images

In [13]:

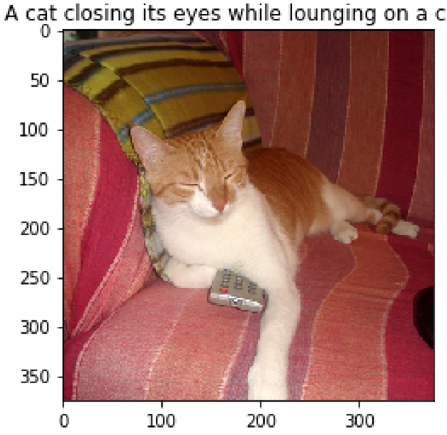
```
# extract captions from zip
def get_captions_for_fns(fns, zip_fn, zip_json_path):
    zf = zipfile.ZipFile(zip_fn)
    j = json.loads(zf.read(zip_json_path).decode("utf8"))
    id_to_fn = {img["id"]: img["file_name"] for img in j["images"]}
    fn_to_caps = defaultdict(list)
    for cap in j['annotations']: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

In [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)
```

A cat sitting on a pink stripped couch
An orange and white cat sitting in a striped chair.
An orange and white cat sleeping on a remote
Brown and white cat sleeping on couch while lying on remote.
A cat closing its eyes while lounging on a chair.



Prepare captions for training

In [15]:

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

Out[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']]
```

In [16]:

```python
# 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):
    """
    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.
    """
    ### YOUR CODE HERE ###
    from collections import Counter
    sentence_list = [item for sublist in train_captions for item in sublist]
    tokens_nested_list = list(map(lambda x: split_sentence(x), sentence_list))
    tokens = [item for sublist in tokens_nested_list for item in sublist]
    tc = Counter(tokens)

    vocab = [item for item in tc if tc[item] >= 5] + [PAD, UNK, START, END]
    return {token: index for index, token in enumerate(sorted(vocab))}

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]],
            ...
        ],
        ...
    ]
    """
    ### YOUR CODE HERE ###
    res = []
    for caption in captions:
        tokens = list(map(lambda x: split_sentence(x), caption))
        indices = [[vocab.get(item, vocab[UNK]) for item in inner] for inner in tokens]
        indices = [[vocab[START]] + item + [vocab[END]] for item in indices]
        res.append(indices)

    return res
```

In [17]:

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

```
8769
```

In [18]:

```python
# 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.

In [19]:
```
# 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!
    """
    ###YOUR CODE HERE###
    if not max_len:
        max_len = max(map(len, batch_captions))
    else:
        max_len = min(max_len, max(map(len, batch_captions)))

    matrix = np.array([x + [pad_idx]*(max_len-len(x))
                      if max_len >= len(x)
                      else x[:max_len]
                      for x in batch_captions])
    return matrix
```

In [20]:
```
batch_captions_to_matrix(train_captions_indexed[1], vocab[PAD], 15).shape
```

Out[20]:
```
(5, 11)
```

In [21]:
```
## GRADED PART, DO NOT CHANGE!
# Vocabulary creation
grader.set_answer("19Wpv", grading_utils.test_vocab(vocab, PAD, UNK, START, END))
# Captions indexing
grader.set_answer("uJh73", grading_utils.test_captions_indexing(train_captions_indexed, vocab, UNK))
# Captions batching
grader.set_answer("yiJkt", grading_utils.test_captions_batching(batch_captions_to_matrix))
```

In [22]:
```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [23]:
```
# 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
```

Training

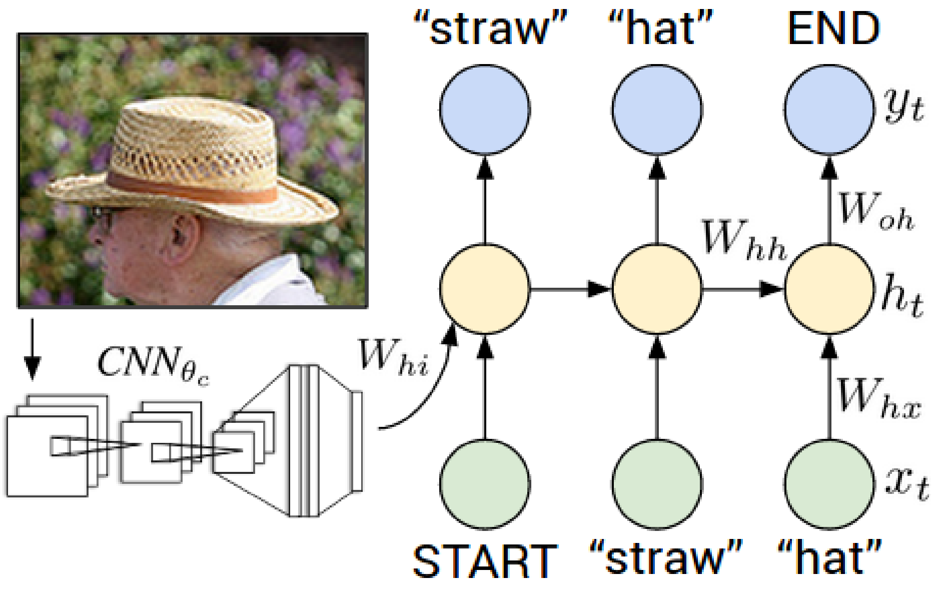
Define architecture

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/ (http://cs.stanford.edu/people/karpathy/)):



In [24]:
```
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]
```

In [25]:
```
train_img_embeds.shape, len(vocab)
```

Out[25]:
```
((82783, 2048), 8769)
```

In [26]:

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

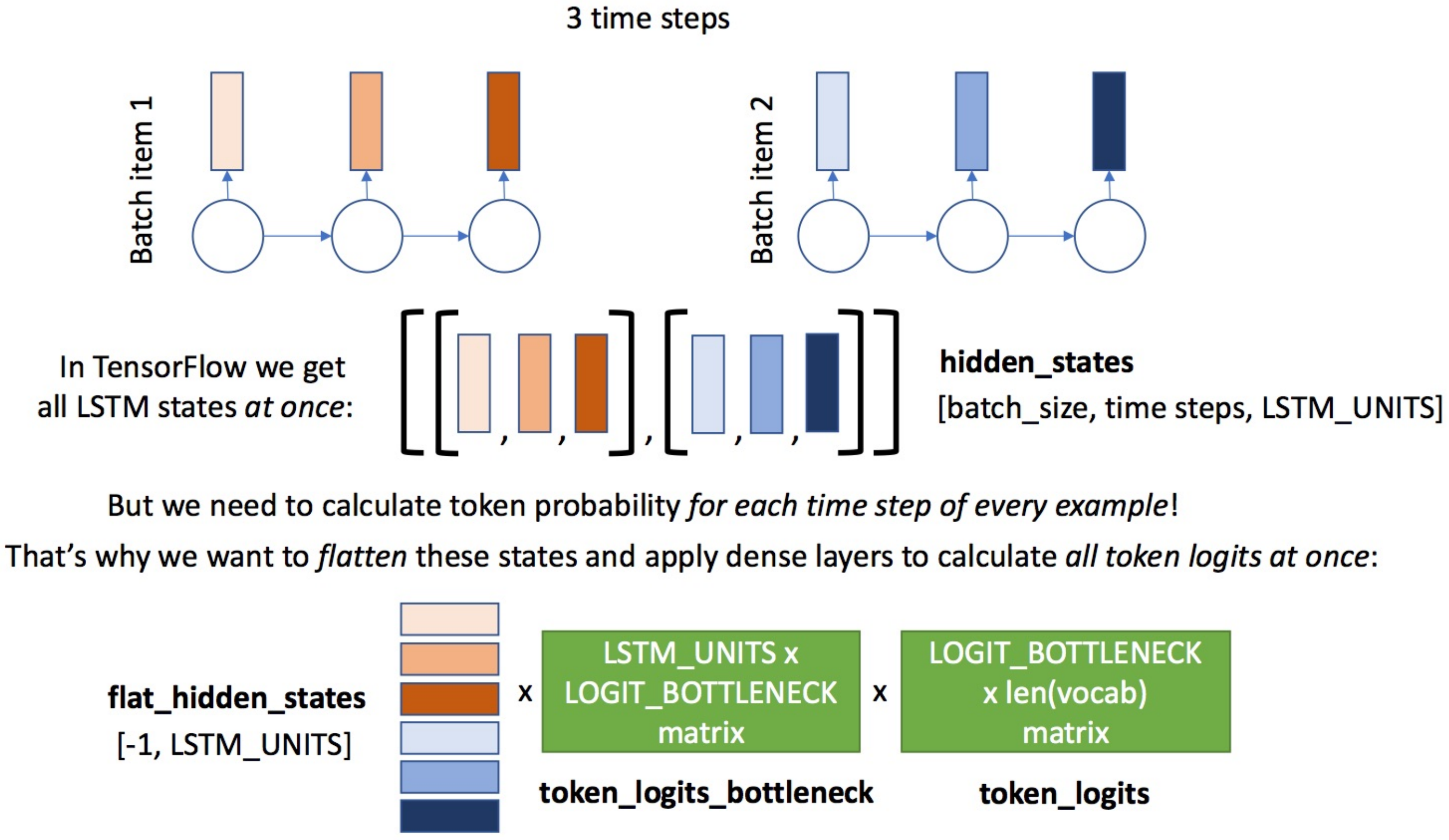
```
WARNING:tensorflow:From ..\keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default_se
ssion instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:95: The name
tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:98: The name
tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:102: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From ..\keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instead.
```

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:



In TensorFlow we get all LSTM states *at once*:

But we need to calculate token probability *for each time step of every example*!

That's why we want to *flatten* these states and apply dense layers to calculate *all token logits at once*:

In [27]:

```
 1 # tf.reset_default_graph()
 2 class decoder:
 3     # [batch_size, IMG_EMBED_SIZE] of CNN image features
 4     img_embeds = tf.placeholder('float32', [None, IMG_EMBED_SIZE])
 5     # [batch_size, time steps] of word ids
 6     sentences = tf.placeholder('int32', [None, None])
 7 
 8     # we use bottleneck here to reduce the number of parameters
 9     # image embedding -> bottleneck
10     img_embed_to_bottleneck = L.Dense(IMG_EMBED_BOTTLENECK,
11                                       input_shape=(None, IMG_EMBED_SIZE),
12                                       activation='elu')
13     # image embedding bottleneck -> lstm initial state
14     img_embed_bottleneck_to_h0 = L.Dense(LSTM_UNITS,
15                                          input_shape=(None, IMG_EMBED_BOTTLENECK),
16                                          activation='elu')
17     # word -> embedding
18     word_embed = L.Embedding(len(vocab), WORD_EMBED_SIZE)
19     # lstm cell (from tensorflow)
20     lstm = tf.nn.rnn_cell.LSTMCell(LSTM_UNITS)
21 
22     # we use bottleneck here to reduce model complexity
23     # lstm output -> logits bottleneck
24     token_logits_bottleneck = L.Dense(LOGIT_BOTTLENECK,
25                                       input_shape=(None, LSTM_UNITS),
26                                       activation="elu")
27     # logits bottleneck -> logits for next token prediction
28     token_logits = L.Dense(len(vocab),
29                            input_shape=(None, LOGIT_BOTTLENECK))
30 
31     # initial lstm cell state of shape (None, LSTM_UNITS),
32     # we need to condition it on `img_embeds` placeholder.
33     ### YOUR CODE HERE ###
34     c0 = h0 = img_embed_bottleneck_to_h0(img_embed_to_bottleneck(img_embeds))
35 
36     # embed all tokens but the last for lstm input,
37     # remember that L.Embedding is callable,
38     # use `sentences` placeholder as input.
39     ### YOUR CODE HERE ###
40     word_embeds = word_embed(sentences[:, :-1])
41 
42     # during training we use ground truth tokens `word_embeds` as context for next token prediction.
43     # that means that we know all the inputs for our lstm and can get
44     # all the hidden states with one tensorflow operation (tf.nn.dynamic_rnn).
45     # `hidden_states` has a shape of [batch_size, time steps, LSTM_UNITS].
46     hidden_states, _ = tf.nn.dynamic_rnn(lstm, word_embeds,
47                                          initial_state=tf.nn.rnn_cell.LSTMStateTuple(c0, h0))
48 
49     # now we need to calculate token logits for all the hidden states
50 
51     # first, we reshape `hidden_states` to [-1, LSTM_UNITS]
52     ### YOUR CODE HERE ###
53     flat_hidden_states = tf.reshape(hidden_states, (-1, LSTM_UNITS))
54 
55     # then, we calculate logits for next tokens using `token_logits_bottleneck` and `token_logits` layers
56     ### YOUR CODE HERE ###
57     flat_token_logits = token_logits(token_logits_bottleneck(flat_hidden_states))
58 
59     # then, we flatten the ground truth token ids.
60     # remember, that we predict next tokens for each time step,
61     # use `sentences` placeholder.
62     ### YOUR CODE HERE ###
63     flat_ground_truth = tf.reshape(sentences[:, 1:], (-1,))
64 
65     # we need to know where we have real tokens (not padding) in `flat_ground_truth`,
66     # we don't want to propagate the loss for padded output tokens,
67     # fill `flat_loss_mask` with 1.0 for real tokens (not pad_idx) and 0.0 otherwise.
68     ### YOUR CODE HERE ###
69     flat_loss_mask = tf.not_equal(flat_ground_truth, vocab[PAD])
70 
71     # compute cross-entropy between `flat_ground_truth` and `flat_token_logits` predicted by lstm
72     xent = tf.nn.sparse_softmax_cross_entropy_with_logits(
73         labels=flat_ground_truth,
74         logits=flat_token_logits
75     )
76 
77     # compute average `xent` over tokens with nonzero `flat_loss_mask`.
78     # we don't want to account misclassification of PAD tokens, because that doesn't make sense,
79     # we have PAD tokens for batching purposes only!
80     ### YOUR CODE HERE ###
81     loss = tf.reduce_mean(tf.boolean_mask(xent, flat_loss_mask))
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\ops\init_ops.py:1251: calling V
arianceScaling.__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 C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\keras\initializers.py:119: call
ing 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-27-389ff8a7bcc0>:20: LSTMCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is depreca
ted 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-27-389ff8a7bcc0>:47: 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 C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\ops\rnn_cell_impl.py:961: calli
ng 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 0x000001FB326D4548
>> could not be transformed and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the ver
```

```
bosity 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 0x000001FB326D4548>>: AttributeError: module 'gast' has no attribute 'Nu
m'
WARNING: Entity <bound method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x000001FB326D4548>> could n
ot 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 <tensor
flow.python.ops.rnn_cell_impl.LSTMCell object at 0x000001FB326D4548>>: AttributeError: module 'gast' has no attribute 'Num'
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\ops\array_ops.py:1354: add_disp
atch_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
```

In [28]:

```python
# 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())
```

In [29]:

```python
## GRADED PART, DO NOT CHANGE!
# Decoder shapes test
grader.set_answer("rbpnH", grading_utils.test_decoder_shapes(decoder, IMG_EMBED_SIZE, vocab, s))
# Decoder random loss test
grader.set_answer("E2OIL", grading_utils.test_random_decoder_loss(decoder, IMG_EMBED_SIZE, vocab, s))
```

In [30]:

```python
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

Training loop

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

In [31]:

```python
train_captions_indexed = np.array(train_captions_indexed)
val_captions_indexed = np.array(val_captions_indexed)
```

In [32]:

```python
# 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: ...}.
    """
    ### YOUR CODE HERE ###
    total = images_embeddings.shape[0]
    idx = np.random.randint(total, size = batch_size)
    batch_image_embeddings = images_embeddings[idx, :]

    ### YOUR CODE HERE ###
    def sample_1_caption(image_captions):
        rand_idx = np.random.randint(len(image_captions), size = 1)[0]
        return image_captions[rand_idx]

    batch_captions_matrix = batch_captions_to_matrix([sample_1_caption(item) for item in indexed_captions[idx]],
                                                     vocab[PAD], max_len = max_len)

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

In [33]:

```python
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
```

In [34]:

```python
# 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!

In [35]:

```python
# 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:
        train_loss += s.run([decoder.loss, train_step],
                            generate_batch(train_img_embeds,
                                           train_captions_indexed,
                                           batch_size,
                                           MAX_LEN))[0]
        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):
        val_loss += s.run(decoder.loss, generate_batch(val_img_embeds,
                                                       val_captions_indexed,
                                                       batch_size,
                                                       MAX_LEN))
    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!")
```

Error rendering Jupyter widget: missing widget manager

```
Epoch: 0, train loss: 4.259566727161407, val loss: 3.6712243938446045
```

Error rendering Jupyter widget: missing widget manager

```
Epoch: 1, train loss: 3.361799514055252, val loss: 3.16270094871521
```

Error rendering Jupyter widget: missing widget manager

```
Epoch: 2, train loss: 2.9963913245201113, val loss: 2.907550594806671
```

Error rendering Jupyter widget: missing widget manager

```
Epoch: 3, train loss: 2.844430368900299, val loss: 2.8292782354354857
```

Error rendering Jupyter widget: missing widget manager

In [36]:

```python
## GRADED PART, DO NOT CHANGE!
# Validation loss
grader.set_answer("YJR7z", grading_utils.test_validation_loss(
    decoder, s, generate_batch, val_img_embeds, val_captions_indexed))
```

Error rendering Jupyter widget: missing widget manager

In [37]:

```python
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [38]:

```
# 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: 2.3909705
Accuracy: 0.4884979702300406
Example 0
Predicted: a living room with a tv chair and tv tv screen tv #END# #END# #END# #END# #END# #END# #END#
Truth: a living room with a sofa piano and large flat screen tv #END# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#

Example 1
Predicted: a street sign station sign is clearly up #END# night #END# #END# #END# #END# #END# #END# #END# #END# #END#
Truth: the canadian gas station sign is lit up at night #END# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#

Example 2
Predicted: a teddy of stuffed bears sitting a floor #END# a table #END# #END# #END# #END# #END# #END# #END# #END#
Truth: a group of teddy bears on the floor near a chair #END# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD# #PAD#
```

In [39]:

```
# save last graph weights to file!
saver.save(s, get_checkpoint_path())
```

Out[39]:

```
'D:\\Google Drive\\Study\\Advanced Machine Learning - Coursera\\1.intro-to-dl\\week6\\weights'
```

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

In [40]:

```python
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 C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\tensorflow\python\training\saver.py:1276: checkpo
int_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 D:\Google Drive\Study\Advanced Machine Learning - Coursera\1.intro-to-dl\week6\weights
WARNING:tensorflow:Entity <bound method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x000001FB326D4548
>> could not be transformed and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the ver
bosity 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 0x000001FB326D4548>>: AttributeError: module 'gast' has no attribute 'Nu
m'
WARNING: Entity <bound method LSTMCell.call of <tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x000001FB326D4548>> could n
ot 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 <tensor
flow.python.ops.rnn_cell_impl.LSTMCell object at 0x000001FB326D4548>>: AttributeError: module 'gast' has no attribute 'Num'
```

In [41]:

```python
# 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
```

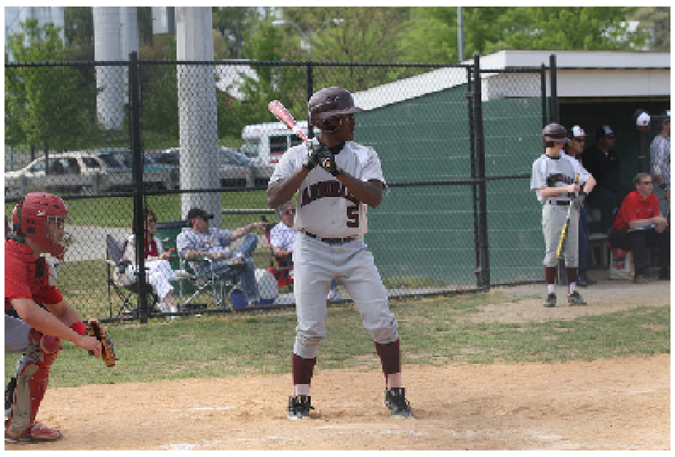
In [42]:

```python
# 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))
```

In [43]:

```
# 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)
```

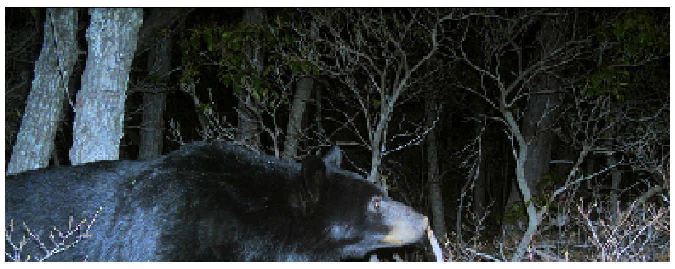
```
a baseball player swinging a bat at a ball
```



In [44]:

```
# 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 black bear is standing in a field
```



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

In [ ]:

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

In [ ]:

```
apply_model_to_image_raw_bytes(open("portal-cake-10.jpg", "rb").read())
```

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=...)
```

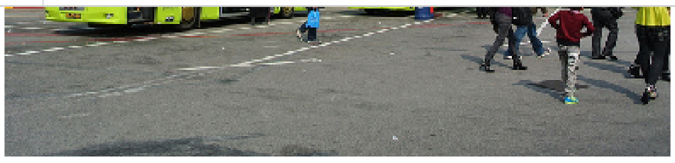
You can use images from the Internet as follows:

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

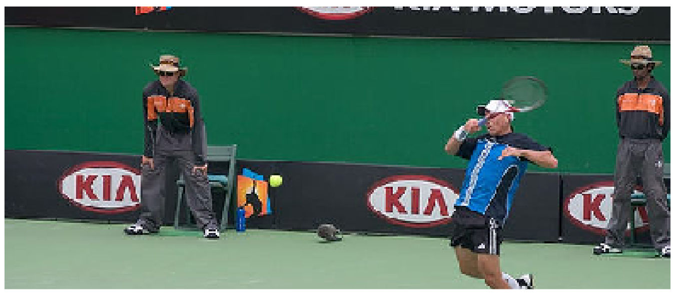
If you use these functions, the output will be embedded into your notebook and will be visible during peer review!

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

In [48]:
```
# The 10 good
good_idx = [355, 203, 87, 24, 373, 340, 29, 214, 363, 140]
for idx in good_idx:
    print("Index:", idx)
    show_valid_example(val_img_fns, example_idx=idx)
    time.sleep(1)
```

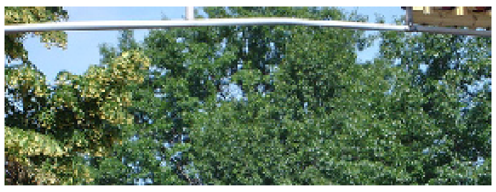


```
Index: 140
a man is playing tennis on a tennis court
```



In [49]:
```
# The 10 bad
bad_idx = [177, 133, 190 ,312, 247, 310, 293, 39, 107, 78]
for idx in bad_idx:
    print("Index:", idx)
    show_valid_example(val_img_fns, example_idx=idx)
    time.sleep(1)
```

```
Index: 107
a bus that is driving down a street
```



In [ ]:
```
### YOUR EXAMPLES HERE ###
```

That's it!

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

In [ ]:
```

```