```
In [ ]: import os
         import random
         from data import ImageDetectionsField, TextField, RawField
         from data import DataLoader, PairedDataset, Example
          from pycocotools.coco import COCO
         import evaluation
         from models.transformer import Transformer, MemoryAugmentedEncoder, MeshedDecoder, ScaledDotProductAttentionMemory
          import torch
          from tqdm import tqdm
         import argparse
          import pickle
          import numpy as np
         import skimage.io as io
          import matplotlib.pyplot as plt
         import pylab
          random.seed(1234)
          torch.manual_seed(1234)
         np.random.seed(1234)
In [ ]:
         def predict captions(model, dataloader, text field):
              import itertools
              model.eval()
              gen = \{\}
              gts = \{\}
              with tqdm(desc='Evaluation', unit='it', total=len(dataloader)) as pbar:
                   for it, (images, caps gt) in enumerate(iter(dataloader)):
                       images = images.to(device)
                       with torch.no_grad():
                           out, _ = model.beam_search(images, 20, text_field.vocab.stoi['<eos>'], 5, out_size=1)
                      caps_gen = text_field.decode(out, join_words=False)
for i, (gts_i, gen_i) in enumerate(zip(caps_gt, caps_gen)):
    gen_i = ' '.join([k for k, g in itertools.groupby(gen_i)])
    gen['%d_%d' % (it, i)] = [gen_i.strip(), ]
    gts['%d_%d' % (it, i)] = gts_i
                       pbar.update()
              gts = evaluation.PTBTokenizer.tokenize(gts)
              gen = evaluation.PTBTokenizer.tokenize(gen)
              scores, = evaluation.compute scores(gts, gen)
              return gts, gen, scores
In [ ]:
         # choose an id from the coco validation set
          \# ids = [522418, 184613, 554625, 193271, 328757]
         img_id = 328757
         annFile = 'annotations/captions_val2014.json'
         coco=COCO(annFile)
         annIds = coco.getAnnIds(imgIds=img_id);
         anns = coco.loadAnns(annIds)
         text = anns[0]['caption']
         loading annotations into memory...
         Done (t=0.21s)
         creating index...
         index created!
In [ ]:
         device = torch.device('cuda')
         print('Meshed-Memory Transformer Evaluation')
         # Pipeline for image regions
         image field = ImageDetectionsField(detections path='coco detections.hdf5', max detections=50, load in tmp=False)
         # Pipeline for text
         text field = TextField(init token='<bos>', eos token='<eos>', lower=True, tokenize='spacy',
                                    remove punctuation=True, nopoints=False)
         # Create the dataset
         test dataset = []
         example = Example.fromdict({'image': os.path.join('coco/images/', 'COCO val2014 000000' + str(img id) + '.jpg'),
          test_dataset.append(example)
          test dataset = PairedDataset(test dataset, {'image': image field, 'text': text field})
         text_field.vocab = pickle.load(open('vocab.pkl', 'rb'))
         # Model and dataloaders
         encoder = MemoryAugmentedEncoder(3, 0, attention module=ScaledDotProductAttentionMemory,
                                                 attention module kwargs={'m': 40})
```

```
decoder = MeshedDecoder(len(text_field.vocab), 54, 3, text_field.vocab.stoi['<pad>'])
model = Transformer(text_field.vocab.stoi['<bos>'], encoder, decoder).to(device)

data = torch.load('meshed_memory_transformer.pth')
model.load_state_dict(data['state_dict'])

dict_dataset_test = test_dataset.image_dictionary({'image': image_field, 'text': RawField()})
dict_dataloader_test = DataLoader(dict_dataset_test, batch_size=10, num_workers=0)

pred, true, scores = predict_captions(model, dict_dataloader_test, text_field)
```

Meshed-Memory Transformer Evaluation

Evaluation: 100% | 1/1 [00:00<00:00, 4.96it/s]

```
In []: # load and display image with annotations

I = io.imread('http://images.cocodataset.org/val2014/COCO_val2014_000000'+ str(img_id) + '.jpg')
plt.imshow(I); plt.axis('off')
ax = plt.gca()
annIds = coco.getAnnIds(imgIds=img_id, iscrowd=None)
anns = coco.loadAnns(annIds)
coco.showAnns(anns)
```

A woman eating vegetables in front of a stove. A woman forks vegetables out of a bowl into her mouth. Woman eating an assortment of mixed vegetables in a bowl. A young woman standing in a kitchen eats a plate of vegetables. A woman eating fresh vegetables from a bowl.



```
In []: # load and display image with instance annotation
    annFile = 'annotations/instances_val2014.json'
    coco_caps=COCO(annFile)
    annIds = coco_caps.getAnnIds(imgIds=img_id);
    anns = coco_caps.loadAnns(annIds)

plt.imshow(I); plt.axis('off')
    annIds = coco_caps.getAnnIds(imgIds=img_id, iscrowd=None)
    anns = coco_caps.loadAnns(annIds)
    coco_caps.showAnns(anns)
```

loading annotations into memory...
Done (t=4.82s)
creating index...
index created!



```
print("Predicted Annotation: {}".format("".join(list(true.values())[0])))
print("\nScores:\n{}".format(scores))
```

Predicted Annotation: a woman eating a bowl of food in a kitchen

Scores:

{'BLEU': [0.5999999999400001, 0.36514837163156744, 0.25543647743589953, 3.9281465085347674e-05], 'METEOR': 0.2332 8211650703048, 'ROUGE': 0.5313588850174217, 'CIDEr': 0.0}

Loading [MathJax]/extensions/Safe.js