First of all -- Checking Questions

Вопрос 1: Можно ли использовать сверточные сети для классификации текстов? Если нет обоснуйте :D, если да то как? как решить проблему с произвольной длинной входа?

Можно. Берется фиксированная длина входа, отсекая конец. Также можно использовать адаптивные сверточные фильтры, которые обеспечат фиксированное количество признаков на каком-то этапе для предложения любой длины.

Вопрос 2: Чем LSTM лучше/хуже чем обычная RNN?

LSTM сеть имеет "память", которая может пригодится в предсказании "контекстной" информации, или связанной с тем, о чем говорилось знчительно ранее. А RNN может предсказать новое слово в пределах предложения или букву в пределах слова, например.

Вопрос 3: Выпишите производную $\frac{dc_{n+1}}{dc_k}$ для LSTM http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (http://colah.github.io/posts/2015-08-Understanding-LSTMs/), объясните формулу, когда производная затухает, когда взрывается?

$$\frac{dC_t}{dC_{t-1}} = \frac{d}{dC_{t-1}}\left(C_{t-1}\cdot f_t + i_t\cdot \tilde{C}_t\right) = f_t + C_{t-1}\frac{d}{dC_{t-1}}\sigma(W_{fx}x_t + W_{fh}h_{t-1} + b_f) + \tilde{C}_t\ \frac{d}{dC_{t-1}}\sigma(W_{ix}x_t + W_{ih}h_{t-1} +$$

$$= f_t + C_{t-1}\sigma'\cdot(0 + W_{fh}\frac{d}{dC_{t-1}}(o_{t-1}\cdot tanh(C_{t-1})) + 0) + \tilde{C}_t\ \sigma'\cdot(W_{ih}\frac{d}{dC_{t-1}}(o_{t-1}\cdot tanh(C_{t-1}))) + i_t tanh'\cdot(W$$

$$= \left(f_t + C_{t-1}\sigma'\cdot W_{fh} + \tilde{C}_t\ \sigma'\cdot W_{ih} + i_t tanh'\cdot W_{Ch}\right)\frac{d}{C_{t-1}}(o_{t-1}\cdot tanh(C_{t-1}))$$

$$= \left(f_t + C_{t-1}\sigma'\cdot W_{fh} + \tilde{C}_t\ \sigma'\cdot W_{ih} + i_t tanh'\cdot W_{Ch}\right)\left(tanh'\cdot o_{t-1} + tanh(C_{t-1})\left(\sigma'\cdot\frac{d}{C_{t-1}}(W_{oC}C_{t-1} + W_{oh}\right.\right.$$

$$= \left(f_t + C_{t-1}\sigma'\cdot W_{fh} + \tilde{C}_t\ \sigma'\cdot W_{ih} + i_t tanh'\cdot W_{Ch}\right)\left(tanh'\cdot o_{t-1} + tanh(C_{t-1})\left(\sigma'\cdot W_{oC}\right)\right)$$

Вопрос 4: Зачем нужен TBPTT почему BPTT плох?

<Ответ>

Вопрос 5: Как комбинировать рекуррентные и сверточные сети, а главное зачем? Приведите несколько примеров реальных задач.

Если составлять описание к картинке или любой другой текст, который должен учесть то, что находися на картинке, нужно получить признаки от картинки для RNN. Эти признаки выделяются с помощью CNN. Часто, комбинация всех выходов CNN подается на скрытый слой RNN, после чего RNN начинает составлять описание, базируясь на результатах CNN.

Вопрос 6: Объясните интуицию выбора размера эмбединг слоя? почему это опасное место?

Да тут и интуиции особо не надо:) Выходы (например 1000) с CNN - характеристики изображения, подаются на hidden layer RNN. Т.к. EmbeddingLayer имеет размер количество one-hot'ов на количество признаков в hidden layer (n x m, пусть m = 128, как в задании), то количество признаков, поданных на первую ячейку RNN (либо h0 либо c0, если есть память), должно быть сравнимо с размером m Embedding-матрицы.

- Arseniy Ashuha, you can text me `ars.ashuha@gmail.com`, Александр Панин

Image Captioning

In this seminar you'll be going through the image captioning pipeline. It can help u https://ars-ashuha.ru/slides/2016.11.11_ImageCaptioning/image_captionong.pdf (https://ars-ashuha.ru/slides/2016.11.11_ImageCaptioning/image_captionong.pdf)

To begin with, let us download the dataset of image features from a pre-trained GoogleNet.

In [3]:

```
!wget https://www.dropbox.com/s/3hj16b0fj6yw7cc/data.tar.gz?dl=1 -O data.tar.gz
!tar -xvzf data.tar.gz
```

```
"wget" Ґ пў«пҐвбп ўгваҐҐ© Ё«Ё ўҐиҐ©
Є®¬ ¤®©, ЁбЇ®«пҐ¬© Їа®Ја ¬¬®© Ё«Ё Ї ЄҐвл¬ д ©«®¬.
"tar" Ґ пў«пҐвбп ўгваҐҐ© Ё«Ё ўҐиҐ©
Є®¬ ¤®©, ЁбЇ®«пҐ¬© Їа®Ја ¬¬®© Ё«Ё Ї ЄҐвл¬ д ©«®¬.
```

Data preprocessing

In [1]:

```
%%time
# Read Dataset
import numpy as np
import pickle

img_codes = np.load("data/image_codes.npy")
captions = pickle.load(open('data/caption_tokens.pcl', 'rb'))
```

```
Wall time: 15.4 s
```

In [2]:

```
import copy
import pandas as pd
```

In [3]:

```
print "each image code is a 1000-unit vector:", img_codes.shape
print img_codes[0,:10]
print '\n\n'
print "for each image there are 5-7 descriptions, e.g.:\n"
print '\n'.join(captions[0])
```

```
each image code is a 1000-unit vector: (123287L, 1000L)
[ 1.38901556 -3.82951474 -1.94360816 -0.5317238  -0.03120959 -2.87483215
 -2.9554503   0.6960277  -0.68551242 -0.7855981 ]



for each image there are 5-7 descriptions, e.g.:

a man with a red helmet on a small moped on a dirt road
man riding a motor bike on a dirt road on the countryside
a man riding on the back of a motorcycle
a dirt path with a young person on a motor bike rests to the foreground of a
 verdant area with a bridge and a background of cloud wreathed mountains
a man in a red shirt and a red hat is on a motorcycle on a hill side
```

In [4]:

```
#split descriptions into tokens
Voc = []
V = 0
for img_i in range(len(captions)):
    for caption_i in range(len(captions[img_i])):
        sentence = captions[img_i][caption_i]
        V+=len(sentence.split())
        captions[img_i][caption_i] = ["#START#"]+sentence.split(' ')+["#END#"]
        Voc += sentence.split(' ')
```

In [5]:

```
print V
print len(Voc)
```

```
6454115
6454115
```

In [6]:

```
print len(set(Voc))
print len(Voc)
print Voc.count(list(set(Voc))[1])
```

```
27929
6454115
1026
```

In [7]:

```
setVoc = list(set(Voc))
```

In [9]:

```
"""Voc_uniq = []
print len(setVoc)
for i in range(len(setVoc)):
    Voc_uniq.append(Voc.count((setVoc)[i]))
    if (i%2000==0):
        print i
```

```
27929
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
22000
24000
26000
```

In [26]:

```
"""pd.DataFrame(np.array(Voc_uniq)).to_csv("Voc_uniq.csv", sep=',', index=False)
```

In [54]:

```
"""Voc_uniq_read_test = pd.read_csv("Voc_uniq.csv")
Voc_uniq_read_test = np.array(Voc_uniq_read_test['0'])
False in Voc_uniq_read_test == np.array(Voc_uniq)
```

Out[54]:

```
False
```

In [9]:

```
Voc_uniq = pd.read_csv("Voc_uniq.csv")
Voc_uniq = np.array(Voc_uniq['0'])
```

In [10]:

```
# Build a Vocabulary

############# TO CODE IT BY YOURSELF #################
word_counts = {}
i = 0
for x, y in zip(set(list(Voc)), Voc_uniq):
    word_counts[x] = y

#word_counts = #<here should be dict word:number of entrances>

vocab  = ['#UNK#', '#START#', '#END#']
vocab += [k for k, v in word_counts.items() if v >= 5]
n_tokens = len(vocab)

assert 10000 <= n_tokens <= 10500

word_to_index = {w: i for i, w in enumerate(vocab)}
```

In [11]:

```
PAD_ix = -1
UNK_ix = vocab.index('#UNK#')

def as_matrix(sequences,max_len=None):
    max_len = max_len or max(map(len,sequences))

    matrix = np.zeros((len(sequences),max_len),dtype='int32')+PAD_ix
    for i,seq in enumerate(sequences):
        row_ix = [word_to_index.get(word,UNK_ix) for word in seq[:max_len]]
        matrix[i,:len(row_ix)] = row_ix

    return matrix
```

In [12]:

```
#try it out on several descriptions of a random image
as_matrix(captions[1337])
```

Out[12]:

```
array([[    1,  2920,  7866,  4200, 10049,  9927,  4704,  4928,  6787,
         1276,  4046,  6275,     2,    -1,    -1],
       [    1,  6787,  1276, 10141,  1046,  9086,  6275,  4046,  8279,
          733,  4704,     2,    -1,    -1,    -1],
       [    1,  8737,  4200, 10049,  9927,  4704,  4928,  6787,  5666,
         8657,   530,  5520,  7134,  5227,     2],
       [    1,  8737,  6620,  3137,  2607,  8273,  8737,  6620,  1600,
            2,    -1,    -1,    -1,    -1,    -1],
       [    1,  2920,  7866,  4200, 10049,  9927,  4704,  4928,  6787,
         1276,   530,  5520,  8914,     2,    -1]])
```

Mah Neural Network

In [13]:

```
# network shapes.
CNN_FEATURE_SIZE = img_codes.shape[1]
EMBED_SIZE = 128 #pls change me if u want
LSTM_UNITS = 200 #pls change me if u want
```

In [15]:

```
import theano
import lasagne
import theano.tensor as T
from lasagne.layers import *
```

In [16]:

```
# Input Variable
sentences = T.imatrix()# [batch_size x time] of word ids
image_vectors = T.matrix() # [batch size x unit] of CNN image features
sentence_mask = T.neq(sentences, PAD_ix)
```

In [17]:

```
#network inputs
l_words = InputLayer((None, None), sentences)
l_mask  = InputLayer((None, None), sentence_mask)

#embeddings for words
############## TO CODE IT BY YOURSELF ##################
l_word_embeddings = lasagne.layers.EmbeddingLayer(l_words,input_size=n_tokens,output_size=E
```

In [18]:

```
# input layer for image features
l_image_features = InputLayer((None, CNN_FEATURE_SIZE), image_vectors)

############## TO CODE IT BY YOURSELF ##################
#convert 1000 image features from googlenet to whatever LSTM_UNITS you have set
#it's also a good idea to add some dropout here and there
l_image_features_small = DropoutLayer(l_image_features,0.5)#<Apply Dropout Layer to regular
l_image_features_small = DenseLayer(l_image_features_small,LSTM_UNITS)
#<Apply Dense to acive LSTM_UNITS size of representation>
assert l_image_features_small.output_shape == (None, LSTM_UNITS)
```

In [19]:

```
l_image_features_small.output_shape
```

Out[19]:

(None, 200)

In [20]:

```
l_word_embeddings.output_shape
```

Out[20]:

```
(None, None, 128)
```

In [21]:

```
############## TO CODE IT BY YOURSELF ###################
# Concatinate image features and word embedings in one sequence
decoder = LSTMLayer(l_word_embeddings,
                    num_units=LSTM_UNITS,
                    cell_init=l_image_features_small,
                    mask_input=l_mask,
                    grad_clipping=10)
```

In [22]:

```
# Decoding of rnn hiden states
from broadcast import BroadcastLayer,UnbroadcastLayer

#apply whatever comes next to each tick of each example in a batch. Equivalent to 2 reshape
broadcast_decoder_ticks = BroadcastLayer(decoder, (0, 1))
print "broadcasted decoder shape = ",broadcast_decoder_ticks.output_shape

predicted_probabilities_each_tick = DenseLayer(
    broadcast_decoder_ticks,n_tokens, nonlinearity=lasagne.nonlinearities.softmax)

#un-broadcast back into (batch,tick,probabilities)
predicted_probabilities = UnbroadcastLayer(
    predicted_probabilities_each_tick, broadcast_layer=broadcast_decoder_ticks)

print "output shape = ", predicted_probabilities.output_shape

predicted_probabilities.output_shape
#remove if you know what you're doing (e.g. 1d convolutions or fixed shape)
#assert predicted_probabilities.output_shape == (None, None, 10373)
```

```
broadcasted decoder shape =  (None, 200)
output shape =  (None, None, 10371)
```

Out[22]:

```
(None, None, 10371)
```

In [23]:

```
next_word_probas = get_output(predicted_probabilities)

reference_answers = sentences[:,1:]
output_mask = sentence_mask[:,1:]

#write symbolic loss function to train NN for
loss = lasagne.objectives.categorical_crossentropy(
    next_word_probas[:, :-1].reshape((-1, n_tokens)),
    reference_answers.reshape((-1,))
).reshape(reference_answers.shape)

############## TO CODE IT BY YOURSELF ##################
loss = loss.mean()#<mean over non-PAD tokens>
```

In [24]:

```
#trainable NN weights
############## TO CODE IT BY YOURSELF ##################
weights = lasagne.layers.get_all_params(predicted_probabilities, trainable=True)#<all dnn w
updates = updates = lasagne.updates.adam(loss,weights)#<your favorite optimizer>
```

In [25]:

```
#compile a function that takes input sentence and image mask, outputs loss and updates weig
#please not that your functions must accept image features as FIRST param and sentences as
############## TO CODE IT BY YOURSELF ##################
#input_sequence, target_values = T.matrix('input sequence', 'int32'),  T.ivector('target y'
train_step = theano.function([image_vectors, sentences], loss, updates=updates, allow_input
val_step   = theano.function([image_vectors, sentences], loss, allow_input_downcast=True)
```

Training

- You first have to implement a batch generator
- Than the network will get trained the usual way

In [27]:

```
captions = np.array(captions)
```

In [28]:

```
captions.shape
```

Out[28]:

```
(123287L,)
```

In [29]:

```
from random import choice

def generate_batch(images,captions,batch_size,max_caption_len=None):
    #sample random numbers for image/caption indicies
    random_image_ix = np.random.randint(0, len(images), size=batch_size)

    #get images
    batch_images = images[random_image_ix]

    #5-7 captions for each image
    captions_for_batch_images = captions[random_image_ix]

    #pick 1 from 5-7 captions for each image
    batch_captions = map(choice, captions_for_batch_images)

    #convert to matrix
    batch_captions_ix = as_matrix(batch_captions,max_len=max_caption_len)

    return batch_images, batch_captions_ix
```

In [30]:

```
generate_batch(img_codes,captions, 3)
```

Out[30]:

```
(array([[-3.63477516, -0.02391352, -2.19817305, ..., -1.01993918,
         -0.42057842,  4.09525633],
        [-0.34424073,  4.72297001,  3.30321574, ..., -1.18078732,
          2.93914557,  1.31586552],
        [ 0.08399808, -2.07835245, -0.22341499, ..., -0.54784209,
         -0.25906569, -0.71788001]], dtype=float32),
 array([[    1,  8737,  1937, 10049,  2643,  3045,  4000,  8273,  8737,
          5228,  8569,     2],
        [    1,  8737,  4493,  4928,  9470,  8696, 10141,  3086,  8737,
          2041,     2,    -1],
        [    1,  2920,  8280,  4040,  9710,  8737,  1022,  9748,  2920,
          2366,     2,    -1]]))
```

Main loop

- We recommend you to periodically evaluate the network using the next "apply trained model" block
  - its safe to interrupt training, run a few examples and start training again

In [32]:

```
batch_size = 50 #adjust me
n_epochs   = 25 #100#adjust me
n_batches_per_epoch = 50 #adjust me
n_validation_batches = 5 #how many batches are used for validation after each epoch
```

In [31]:

```
from tqdm import tqdm
```

In [33]:

```
#added by student because of problems with RAM and due using CPU instead GPU
import os

__all__ = [
    'read_model_data',
    'write_model_data',
]

PARAM_EXTENSION = 'params'


def read_model_data(model, filename):
    """Unpickles and loads parameters into a Lasagne model."""
    filename = os.path.join('./', '%s.%s' % (filename, PARAM_EXTENSION))
    with open(filename, 'r') as f:
        data = pickle.load(f)
    lasagne.layers.set_all_param_values(model, data)


def write_model_data(model, filename):
    """Pickels the parameters within a Lasagne model."""
    data = lasagne.layers.get_all_param_values(model)
    filename = os.path.join('./', filename)
    filename = '%s.%s' % (filename, PARAM_EXTENSION)
    with open(filename, 'w') as f:
        pickle.dump(data, f)
```

In [34]:

```
"""write_model_data(predicted_probabilities,'wrote_net')
```

In [36]:

```
"""test_loading_net = copy.deepcopy(predicted_probabilities)
read_model_data(test_loading_net,'wrote_net')
```

In [34]:

```
read_model_data(predicted_probabilities,'wrote_net')
```

с оперативкой проблемы, поэтому кусками обучался и до конца не довел, потом надо будет разобраться с виндой (не могу gpu заюзать, сколько не пытался)

Итоговое количество эпох обучения - около 100

актуальная ошибка: train loss: 1.79, val loss: 1.82

In [35]:

```
n_epochs = 50
for epoch in range(n_epochs):
    train_loss=0
    for _ in tqdm(range(n_batches_per_epoch)):
        train_loss += train_step(*generate_batch(img_codes,captions,batch_size))
    train_loss /= n_batches_per_epoch

    val_loss=0
    for _ in range(n_validation_batches):
        val_loss += val_step(*generate_batch(img_codes,captions,batch_size))
    val_loss /= n_validation_batches

    print('\nEpoch: {}, train loss: {}, val loss: {}'.format(epoch, train_loss, val_loss))
    if (epoch%5==0):
        write_model_data(predicted_probabilities,'wrote_net')
        print 'rewrite'
        read_model_data(predicted_probabilities,'wrote_net')

print("Finish :)")
```

```
Epoch: 21, train loss: 1.73116863251, val loss: 1.71766816378
100%|████████████████████████████████████████████████████████████
| 50/50 [01:40<00:00,  1.88s/it]

Epoch: 22, train loss: 1.83921370268, val loss: 1.83387253284
100%|████████████████████████████████████████████████████████████
| 50/50 [01:45<00:00,  2.00s/it]

Epoch: 23, train loss: 1.79814567804, val loss: 1.84272072315
100%|████████████████████████████████████████████████████████████
| 50/50 [02:40<00:00,  2.51s/it]

Epoch: 24, train loss: 1.80522366285, val loss: 1.74367649555
100%|████████████████████████████████████████████████████████████
```

apply trained model

In [36]:

```
#the same kind you did last week, but a bit smaller
from pretrained_lenet import build_model,preprocess,MEAN_VALUES

# build googlenet
lenet = build_model()

#load weights
lenet_weights = pickle.load(open('data/blvc_googlenet.pkl','rb'))['param values']
set_all_param_values(lenet["prob"], lenet_weights)

#compile get_features
cnn_input_var = lenet['input'].input_var
cnn_feature_layer = lenet['loss3/classifier']
get_cnn_features = theano.function([cnn_input_var], lasagne.layers.get_output(cnn_feature_l
```

In [37]:

```
from matplotlib import pyplot as plt
%matplotlib inline

#sample image
img = plt.imread('data/Dog-and-Cat.jpg')
img = preprocess(img)
```

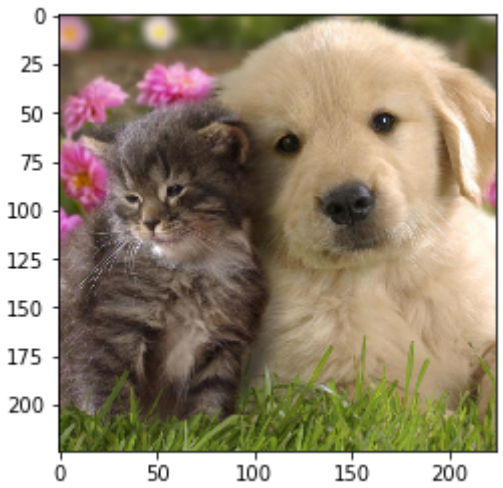
```
C:\Users\Klapeyron\Anaconda2\lib\site-packages\skimage\transform\_warps.py:8
4: UserWarning: The default mode, 'constant', will be changed to 'reflect' in
skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
```

In [38]:

```
#deprocess and show, one line :)
from pretrained_lenet import MEAN_VALUES
plt.imshow(np.transpose((img[0] + MEAN_VALUES)[::-1],[1,2,0]).astype('uint8'))
```

Out[38]:

```
<matplotlib.image.AxesImage at 0x354a70d30>
```



Generate caption