LSTM with Aspect Embedding

В этом ноутбуке мы исследуем применение модели LSTM with Aspect Embedding (AE-LSTM) для решения задачи аспектного анализа тональности. В качестве основы мы берем статью (https://www.aclweb.org/anthology/D16-1058.pdf).

In [1]:

```
import os
import zipfile
import random

import pandas as pd
import numpy as np
import torch
import gensim
import matplotlib.pyplot as plt
import seaborn as sns
import bs4

from tqdm.notebook import tqdm
from sklearn.metrics import accuracy_score

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchtext
from torch.utils.data import Dataset as TorchDataset
from torch.utils.data import DataLoader

from google.colab import drive

USE_GOOGLE_DRIVE = True

sns.set(font_scale=1.2)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:1
9: FutureWarning: pandas.util.testing is deprecated. Use the functions
in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Примонтируем наш гугл-диск, если мы решили его использовать.

In [2]:

```
if USE_GOOGLE_DRIVE:
    drive.mount('/content/drive')
    PATH_PREFIX = 'drive/My Drive/NLP/dialog-sent'
else:
    PATH_PREFIX = '..'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [0]:

```
DATA_PREFIX = os.path.join(PATH_PREFIX, 'data')
MODEL_PREFIX = os.path.join(PATH_PREFIX, 'models')
```

Зададим девайс для обучения.

In [4]:

```
USE_GPU = True

if USE_GPU and torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print('using device:', device)
```

```
using device: cuda
```

Проинициализируем везде генераторы случайных чисел.

In [0]:

```
random_seed = 42
random_state = random_seed

def set_seed_everywhere(seed, cuda):
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.backends.cudnn.determenistic = True
    torch.backends.cudnn.benchmark = False
    if cuda:
        torch.cuda.manual_seed_all(seed)

set_seed_everywhere(random_seed, USE_GPU)
```

Повторение результатов статьи

Для того, чтобы применить модель к нашей конкретной задаче для начала надо повторить результаты исследуемой статьи. Будем это делать на примере датасета с [отзывами на рестораны (http://alt.qcri.org/semeval2014/task4/index.php?id=data-and-tools)](http://alt.qcri.org/semeval2014/task4/index.php?id=data-and-tools).

Данные

Теперь надо разобраться с данными. Мы используем отзывы о рестранах. Согласно аннотации к датасету используются следующие аспекты:

- food
- price
- service
- ambience
- anecdotes (sentences describing the reviewer’s personal experience or context, but that do not usually provide information on the restaurant quality e.g. “I knew upon visiting NYC that I wanted to try an original

deli”)
- miscellaneous (sentences that do not belong to the other five categories including sentences that are general recommendations e.g. “Your friends will thank you for introducing them to this gem!”)

Эта же аннотация говорит о том, какие существуют тональности:

- positive
- negative
- conflict (both positive and negative sentiment)
- neutral (neither positive nor negative sentiment)

Помимо этого существует такой объект, как термин аспекта (aspect term) -- это какое-то слово или выражение, которое указывает на то, в каком слове выражается тональность к конкретному аспекту.

Существует два вида задач для аспектного анализа тональности:

- Aspect-level Classification -- определить тональность для каждого аспекта.
- Aspect-Term-level Classification -- определить термины для каждого аспекта, а затем определить их тональность.

Наша финальная задача относится к первой категории, а значит нам не нужны данные о терминах аспекта.

Данные предоставлены в формате xml. Для работы с ними будем использовать beautiful soup.

In [0]:

```
dataset_path = os.path.join(
    DATA_PREFIX, 'raw', "SemEval'14-ABSA-TrainData_v2 & AnnotationGuidelines",
    'Restaurants_Train_v2.xml'
)

aspect_categories = ['food', 'price', 'service', 'ambience',
                     'anecdotes/miscellaneous']
num_aspects = len(aspect_categories)
data_dict = {'sentence_id': [], 'text': []}
data_dict.update({key: [] for key in aspect_categories})

sent2class = {'none': 0, 'neutral': 1, 'positive': 2, 'negative': 3,
              'conflict': 4}

with open(dataset_path, 'r') as inf:
    contents = inf.read()
    soup = bs4.BeautifulSoup(contents, 'xml')
    for sentence in soup.find_all('sentence'):
        data_dict['sentence_id'].append(sentence['id'])

        text = sentence.find('text').text
        data_dict['text'].append(text)

        categories = sentence.find('aspectCategories').find_all(
            'aspectCategory'
        )
        for aspect_category in aspect_categories:
            data_dict[aspect_category].append(0)

        for aspect_category in categories:
            data_dict[
                aspect_category['category']
            ][-1] = sent2class[aspect_category['polarity']]

data = pd.DataFrame(data_dict)
```

In [7]:

```
data.head()
```

Out[7]:

	sentence_id	text	food	price	service	ambience	anecdotes/miscellaneous
0	3121	But the staff was so horrible to us.	0	0	3	0	0
1	2777	To be completely fair, the only redeeming fact...	2	0	0	0	3
2	1634	The food is uniformly exceptional, with a very...	2	0	0	0	0
3	2534	Where Gabriela personaly greets you and recomm...	0	0	2	0	0
4	583	For those that go once and don't enjoy it, all...	0	0	0	0	2

Новое

Чтобы правильно обучать нашу модель придется упростить данные до трех столбцов: `text` , `aspect` , `sentiment` .

In [8]:

```
data_to_train_dict = {'text': [], 'aspect': [], 'sentiment': []}
for i, row in data.iterrows():
    for aspect_category in aspect_categories:
        data_to_train_dict['text'].append(row['text'])
        data_to_train_dict['aspect'].append(aspect_category)
        data_to_train_dict['sentiment'].append(row[aspect_category])

data_to_train = pd.DataFrame(data_to_train_dict)
data_path = os.path.join(DATA_PREFIX, 'processed', 'restaurants.csv')
data_to_train.to_csv(data_path, index=False)
data_to_train.head(10)
```

Out[8]:

	text	aspect	sentiment
0	But the staff was so horrible to us.	food	0
1	But the staff was so horrible to us.	price	0
2	But the staff was so horrible to us.	service	3
3	But the staff was so horrible to us.	ambience	0
4	But the staff was so horrible to us.	anecdotes/miscellaneous	0
5	To be completely fair, the only redeeming fact...	food	2
6	To be completely fair, the only redeeming fact...	price	0
7	To be completely fair, the only redeeming fact...	service	0
8	To be completely fair, the only redeeming fact...	ambience	0
9	To be completely fair, the only redeeming fact...	anecdotes/miscellaneous	3

Теперь определим поля в датасете для обучения.

In [0]:

```
TEXT = torchtext.data.Field(
    tokenize='spacy', batch_first=True, init_token='<begin>',
    eos_token='<end>', lower=True
)
ASPECT = torchtext.data.Field(sequential=False, batch_first=True)
SENTIMENT = torchtext.data.LabelField(batch_first=True)
```

In [0]:

```
fields = [('text', TEXT), ('aspect', ASPECT), ('sentiment', SENTIMENT)]
```

In [11]:

```
training_data = torchtext.data.TabularDataset(
    path=data_path, format = 'csv', fields=fields, skip_header=True
)

print(vars(training_data.examples[0]))
```

```
{'text': ['but', 'the', 'staff', 'was', 'so', 'horrible', 'to', 'us',
'.'], 'aspect': 'food', 'sentiment': '0'}
```

Разделим данные на валидацию и тест.

In [0]:

```
train_data, valid_data = training_data.split(
    split_ratio=0.75, random_state=random.seed(random_state)
)
```

Построим для нашего датасета словарь. Согласно статье, эмбеддинги слов были проинициализированы при помощи GloVe, а для неизвестных слов в качестве эмбеддингов были взяты случайные векторы из $U(-\varepsilon, \varepsilon)$, где $\varepsilon = 0.01$.

In [0]:

```
epsilon = 0.01
TEXT.build_vocab(train_data, vectors='glove.6B.300d',
                 unk_init=lambda x: torch.nn.init.uniform_(x, -epsilon, epsilon))
ASPECT.build_vocab(train_data)
SENTIMENT.build_vocab(train_data)
```

In [14]:

```
print(f'Size of TEXT vocabulary: {len(TEXT.vocab)}')
print(f'Size of ASPECT vocabulary: {len(ASPECT.vocab)}')
print(f'Size of SENTIMENT vocabulary: {len(SENTIMENT.vocab)}')
```

```
Size of TEXT vocabulary: 4493
Size of ASPECT vocabulary: 6
Size of SENTIMENT vocabulary: 5
```

Теперь напишем итератор.

In [0]:

```
batch_size = 25

train_iter, val_iter = torchtext.data.BucketIterator.splits(
        (train_data, valid_data),
        batch_sizes=(batch_size, batch_size),
        sort=True,
        sort_key=lambda x: len(x.text),
        sort_within_batch=True,
        device=device,
        repeat=False
)
```

Модель

Зададим архитектуру нашей сети.

In [0]:

```
class AE_LSTM(nn.Module):

    def __init__(
        self, num_aspects, embeddings_aspects_dim, hidden_dim,
        num_sentiments,
        vocab_size=None, embeddings_words_dim=None, embeddings_words_init=None
    ):
        super(AE_LSTM, self).__init__()

        train_from_scratch = (vocab_size is not None
                              and embedding_words_dim is not None)
        use_pretrained = embeddings_words_init is not None
        if not(train_from_scratch or use_pretrained):
            raise ValueError('You should use pretrained vectors or '
                             'set vocab size and embeddings dim')

        if train_from_scratch:
            self.embeddings_words = nn.Embedding(
                vocab_size, embeddings_words_dim
            )
        else:
            self.embeddings_words = nn.Embedding.from_pretrained(
                embeddings_words_init
            )
            vocab_size, embeddings_words_dim = embeddings_words_init.size()

        self.embeddings_aspects = nn.Embedding(
            num_aspects, embeddings_aspects_dim
        )
        self.lstm = nn.LSTM(
            embeddings_aspects_dim + embeddings_words_dim,
            hidden_dim, batch_first=True
        )
        self.hidden_to_sentiments = nn.Linear(hidden_dim, num_sentiments)

        # weights initializatioin
        torch.nn.init.xavier_uniform_(self.embeddings_aspects.weight)
        torch.nn.init.xavier_uniform_(self.hidden_to_sentiments.weight)

    def forward(self, texts, aspects, hidden=None):
        batch_size, seq_len = texts.size()
        # shape: [batch_size, seq_len] -> [batch_size, seq_len, embeddings_words_di
        embeddings_words_out = self.embeddings_words(texts)
        # shape: [batch_size] -> [batch_size, seq_len, embedding_aspects_dim]
        embeddings_aspects_out = self.embeddings_aspects(aspects)
        embeddings_aspects_out = torch.unsqueeze(embeddings_aspects_out, 1)
        embeddings_aspects_out = torch.repeat_interleave(
            embeddings_aspects_out, seq_len, 1
        )
        # shape: -> [batch_size, seq_len, embedding_words_dim + embedding_aspects_d
        embeddings = torch.cat(
            [embeddings_words_out, embeddings_aspects_out], dim=-1,
        )
        # shape: [batch_size, seq_len, embeddings_dim] -> [batch_size, hidden_dim]
        lstm_out, hidden_state = self.lstm(embeddings, hidden)
        lstm_out = lstm_out[:, -1, :]
        # shape: [batch_size, hidden_size] -> [batch_size, num_sentiments]
        sentiments_scores = self.hidden_to_sentiments(lstm_out)
        return sentiments_scores, hidden_state
```

Возьмем параметры, как в статье: `embeddings_aspects_dim=300`, `hidden_dim=300`.

In [17]:

```
num_aspects = len(ASPECT.vocab)
embeddings_aspects_dim = 300
hidden_dim = 300
num_sentiments = len(SENTIMENT.vocab)
embeddings_words_init = TEXT.vocab.vectors
model = AE_LSTM(
    num_aspects, embeddings_aspects_dim, hidden_dim, num_sentiments,
    embeddings_words_init=embeddings_words_init
)
model
```

Out[17]:

```
AE_LSTM(
  (embeddings_words): Embedding(4493, 300)
  (embeddings_aspects): Embedding(6, 300)
  (lstm): LSTM(600, 300, batch_first=True)
  (hidden_to_sentiments): Linear(in_features=300, out_features=5, bias
=True)
)
```

Тренировочный цикл

Здесь мы реализуем тренировочный цикл для нашей модели. К сожалению, установить momentum, как в статье не удалось (там почему-то написано, что они установили его для Adagrad).

In [0]:

```
model = model.to(device=device)
learning_rate = 1e-3
early_stopping = 10
epochs = 30

opt = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.StepLR(opt, step_size=4, gamma=0.1)
criterion = nn.CrossEntropyLoss()
path_save = os.path.join(MODEL_PREFIX, 'ae_lstm', 'ae_lstm.pt')
```

In [19]:

```
set_seed_everywhere(random_seed, USE_GPU)
epoch_losses = []
val_losses = []
best_val_loss = 10
not_improves = 0
for epoch in tqdm(range(1, epochs + 1)):
    running_loss = 0.0
    running_corrects = 0
    model.train()
    for batch in train_iter:

        texts = batch.text.to(device=device)
        aspects = batch.aspect.to(device=device)
        sentiments = batch.sentiment.to(device=device)

        opt.zero_grad()
        preds, _ = model(texts, aspects)
        loss = criterion(preds, sentiments)
        loss.backward()
        opt.step()
        running_loss += loss.item()

    epoch_loss = running_loss / len(train_data)
    epoch_losses.append(epoch_loss)

    val_loss = 0.0
    model.eval()
    with torch.no_grad():
        for batch in val_iter:

            texts = batch.text.to(device=device)
            aspects = batch.aspect.to(device=device)
            sentiments = batch.sentiment.to(device=device)

            preds, _ = model(texts, aspects)
            loss = criterion(preds, sentiments)
            val_loss += loss.item()

    val_loss /= len(valid_data)
    val_losses.append(val_loss)

    if val_loss < best_val_loss:
        not_improves = 0
        best_val_loss = val_loss
        torch.save(model.state_dict(), path_save)
    else:
        not_improves += 1
        if not_improves == early_stopping:
            break

    scheduler.step()

    print(f'Epoch: {epoch}, Training Loss*: {1000*epoch_loss:.5f}, '
          f'Validation Loss*: {1000*val_loss:.5f}')
```

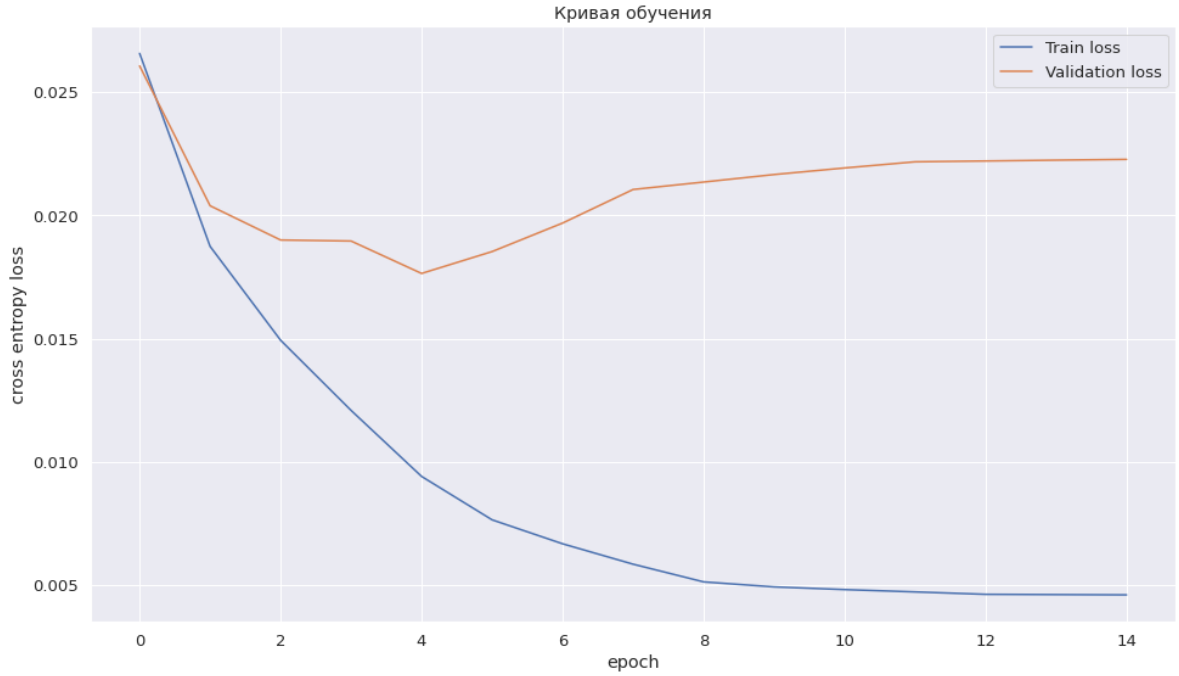
```
HBox(children=(IntProgress(value=0, max=30), HTML(value='')))

Epoch: 1, Training Loss*: 26.57627, Validation Loss*: 26.06756
```

```
Epoch: 2, Training Loss*: 18.74721, Validation Loss*: 20.39435
Epoch: 3, Training Loss*: 14.93955, Validation Loss*: 19.00506
Epoch: 4, Training Loss*: 12.09278, Validation Loss*: 18.96359
Epoch: 5, Training Loss*: 9.41476, Validation Loss*: 17.64828
Epoch: 6, Training Loss*: 7.65418, Validation Loss*: 18.53584
Epoch: 7, Training Loss*: 6.68190, Validation Loss*: 19.69238
Epoch: 8, Training Loss*: 5.85480, Validation Loss*: 21.05407
Epoch: 9, Training Loss*: 5.13689, Validation Loss*: 21.35588
Epoch: 10, Training Loss*: 4.93320, Validation Loss*: 21.66064
Epoch: 11, Training Loss*: 4.82446, Validation Loss*: 21.92745
Epoch: 12, Training Loss*: 4.72980, Validation Loss*: 22.17723
Epoch: 13, Training Loss*: 4.63228, Validation Loss*: 22.21100
Epoch: 14, Training Loss*: 4.62108, Validation Loss*: 22.24436
```

In [20]:

```
plt.figure(figsize=(16, 9))
finished_epochs = len(epoch_losses)
plt.plot(np.arange(finished_epochs), epoch_losses, label='Train loss')
plt.plot(np.arange(finished_epochs), val_losses, label='Validation loss')
plt.xlabel('epoch')
plt.ylabel('cross entropy loss')
plt.title('Кривая обучения')
plt.legend()
plt.show()
```



Измерим качество полученной модели.

In [21]:

```
model.load_state_dict(torch.load(path_save))
```

Out[21]:

```
<All keys matched successfully>
```

In [0]:

```
model.eval()

all_preds = []
all_answers = []
with torch.no_grad():
    for batch in val_iter:

        texts = batch.text.to(device=device)
        aspects = batch.aspect.to(device=device)
        sentiments = batch.sentiment.to(device=device)

        preds_scores, _ = model(texts, aspects)
        preds = torch.max(preds_scores, dim=-1)[1]
        all_preds += preds.tolist()
        all_answers += sentiments.tolist()

all_preds = np.array(all_preds)
all_answers = np.array(all_answers)
```

In [23]:

```
accuracy = accuracy_score(all_preds, all_answers)

print(f'Accuracy: {accuracy:.3f}')
```

```
Accuracy: 0.861
```

Что стоит еще сделать

1. Проверить, что решаемая такой моделью задача совпадает с задачей соревнования, откуда взяли датасет и соотносится с непосредственно с решаемой задачей. Проверить, что измеряемая метрика совпадает (может там считается точность для одного предложения по всем аспектам, а не по каждому аспекту). Для этого может помочь эта статья (http://www.dialog-21.ru/media/1111/blinovpdkotelnikovev.pdf).
2. К этому датасету существуют еще и тестовые данные. Если к ним есть ответы (а судя по статье есть, откуда то же они взяли свои результаты), то имеет смысл именно их использовать для финального теста, а не валидацию.
3. Поэкспериментировать с моделью:
    - dropout,
    - другие эмбеддинги,
    - bidirectional,
    - l2-регуляризация, как в статье
    - доучивание эмбеддингов вместо заморозки
    - увеличение числа слоев lstm
4. Проверить, что в текущей модели от использования эмбеддингов для аспектов вообще есть толк (можно попробовать их отключить как-то).
5. Подумать, как сделать результаты воспроизводимыми -- чтобы каждый раз при обучении получались одинаковые результаты. Я что-то сделал в этом направлении, но этого недостаточно (попробовать установить seed в обычном питоновском модуле random).