# 2022-03-16 attention

March 16, 2022

[140]:
```
import raw_source
import math
from tqdm import tqdm,trange
class Dataset:
    def __init__(self,text,labels,masks):
        self.text = torch.tensor(text)
        self.labels = torch.tensor(labels)
        self.masks = torch.from_numpy(masks)
    def __getitem__(self, index):
        return self.text[index],self.labels[index],self.masks[index]
    def __len__(self):
        return len(self.text)
```

[141]:
```
train_text,train_labels,train_masks,dev_text,dev_labels,dev_masks,test_text,test_labels,test_m
↪= raw_source.main()
```

```
180000it [00:00, 681456.47it/s]
10000it [00:00, 714215.85it/s]
10000it [00:00, 666789.18it/s]
```

[142]:
```
print('  ',train_contents[0])
print('ont hot :',train_text[0])
print('  label:',train_labels[0])
print('mask:',train_masks[0])
print('  :',len(train_text[0]))
```

```
         1
ont hot : [14, 125, 55, 45, 35, 307, 4, 81, 161, 941, 258, 494, 2, 175, 48,
145, 97, 17, 4761, 4761, 4761, 4761, 4761, 4761, 4761, 4761, 4761, 4761,
4761, 4761, 4761, 4761, 4761, 4761, 4761, 4761, 4761]
  label: 3
mask: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  : 38
```

[143]:
```
import torch
import numpy as np
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler,
                              TensorDataset)
```

```
train_data = Dataset(train_text, train_labels,train_masks)
train_dataloader = DataLoader(train_data,batch_size=32,shuffle=True)
```

[144]:
```
dev_data = Dataset(dev_text, dev_labels,dev_masks)
dev_dataloader = DataLoader(dev_data,batch_size=32,shuffle=True)
```

[145]:
```
class Config:
    def __init__(self,vocab_size):
        self.vocab_size = vocab_size

        #
        self.num_attention_heads = 8
        #
        self.embedding_size = 240
        #   :30
        self.attention_head_size = int(self.embedding_size / self.
 ↪num_attention_heads)

        self.all_head_size = self.num_attention_heads * self.attention_head_size

        self.attention_probs_dropout_prob = 0.1

        self.class_nums = 10

        self.batch_size = 32


config = Config(vocab_size)
```

[146]:
```
import torch.nn as nn
import numpy as np
class SelfAttention(nn.Module):
    def __init__(self,config):
        super(SelfAttention, self).__init__()

        self.embedding = nn.Embedding(config.vocab_size,240)
        self.config = config
        self.query = nn.Linear(config.embedding_size, config.all_head_size)#240␣
 ↪* 240
        self.key = nn.Linear(config.embedding_size, config.all_head_size)#240 *␣
 ↪240
        self.value = nn.Linear(config.embedding_size, config.all_head_size)#240␣
 ↪* 240
        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
        self.fc = nn.Linear(config.embedding_size,config.class_nums)
    def forward(self,text,mask):
        #text : batch,38; mask: batch,38
```

```
        text_embedding = self.embedding(text) # batch,38,240

        Q = self.query(text_embedding) # batch,38,240
        K = self.key(text_embedding) # batch,38,240
        V = self.value(text_embedding) # batch,38,240

        # batch seq_len 240 -> batch 8 seq_len 30
        new_shape = Q.size()[:-1] + (self.config.num_attention_heads , self.
↪config.attention_head_size)
        Q = Q.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        K = K.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        V = V.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        # (QK^T)/sqrt(d)
        attention_scores = torch.matmul(Q,K.transpose(-1,-2))# batch 8 38 30 *␣
↪batch 8 30 38 = batch 8 38 38
        attention_scores = attention_scores / math.sqrt(self.config.
↪attention_head_size)



        #  mask
        extended_img_mask = mask.unsqueeze(1).unsqueeze(2) # batch 1 1 38
        extended_img_mask = (1.0 - extended_img_mask) * -10000.0 # batch 1 1 38
        attention_scores = attention_scores + extended_img_mask

        #softmax((QK^T)/sqrt(d))
        attention_probs = nn.Softmax(dim=-1)(attention_scores)# batch 8 38
        attention_probs = self.dropout(attention_probs).to(torch.float)

        res = torch.matmul(attention_probs,V)# batch 8 38 30

        res = res.permute(0,2,1,3).contiguous() #batch 38 8 30
        ori_shape = res.size()[:-2] + (self.config.all_head_size,)
        res = res.view(*ori_shape) # batch 38 240
        res = torch.mean(res,axis = 1) # batch 240
        pred = self.fc(res)

        return pred , attention_probs
```

[147]:
```
import torch.nn as nn
import numpy as np
class SelfAttention_Print(nn.Module):
    def __init__(self,config):
        super(SelfAttention_Print, self).__init__()
        self.embedding = nn.Embedding(config.vocab_size,240)
        self.config = config
```

```
        self.query = nn.Linear(config.embedding_size, config.all_head_size)#240␣
↪* 240
        self.key = nn.Linear(config.embedding_size, config.all_head_size)#240 *␣
↪240
        self.value = nn.Linear(config.embedding_size, config.all_head_size)#240␣
↪* 240
        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
        self.fc = nn.Linear(config.embedding_size,config.class_nums)

    def forward(self,text,mask):
        #text : batch,38; mask: batch,38
        text_embedding = self.embedding(text) # batch,38,240
        print('text_embedding:',text_embedding.shape)

        Q = self.query(text_embedding) # batch,38,240
        K = self.key(text_embedding) # batch,38,240
        V = self.value(text_embedding) # batch,38,240
        print('Q:',Q.shape)
        print('K:',K.shape)
        print('V:',V.shape)

        # batch seq_len 240 -> batch 8 seq_len 30
        new_shape = Q.size()[:-1] + (self.config.num_attention_heads , self.
↪config.attention_head_size)
        Q = Q.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        K = K.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        V = V.view(*new_shape).permute(0,2,1,3) # batch 8 38 30
        print('Q:',Q.shape)
        print('K:',K.shape)
        print('V:',V.shape)

        # (QK^T)/sqrt(d)
        attention_scores = torch.matmul(Q,K.transpose(-1,-2))# batch 8 38 30 *␣
↪batch 8 30 38 = batch 8 38 38
        attention_scores = attention_scores / math.sqrt(self.config.
↪attention_head_size)
        print('attention_scores:',attention_scores.shape)

        #  mask
        extended_img_mask = mask.unsqueeze(1).unsqueeze(2) # batch 1 1 38
        extended_img_mask = (1.0 - extended_img_mask) * -10000.0 # batch 1 1 38
        print('extended_img_mask:',extended_img_mask.shape)
        attention_scores = attention_scores + extended_img_mask
        print('attention_scores:',attention_scores.shape)
        #softmax((QK^T)/sqrt(d))
        attention_probs = nn.Softmax(dim=-1)(attention_scores)# batch 8 38 38
```

```
        print('after softmax attention_scores:',attention_scores.shape)
        attention_probs = self.dropout(attention_probs).to(torch.float)

        res = torch.matmul(attention_probs,V)# batch 8 38 30
        print('res:',res.shape)
        res = res.permute(0,2,1,3).contiguous() #batch 38 8 30
        ori_shape = res.size()[:-2] + (self.config.all_head_size,)
        res = res.view(*ori_shape) # batch 38 240
        print('res:',res.shape)
        res = torch.mean(res,axis = 1)
        print('res:',res.shape)
        pred = self.fc(res)
        print('pred:',pred.shape)
        return pred , attention_probs
```

[148]:
```
model = SelfAttention_Print(config)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=0)
criterion = torch.nn.CrossEntropyLoss()
```

[149]:
```
model = model.train()
for i,batch in enumerate(train_dataloader):
    if i == 0:
        train_text,train_labels,train_masks = batch
        pred,attention_probs = model(train_text,train_masks)
        loss = criterion(pred,train_labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    else:
        break
```

```
text_embedding: torch.Size([32, 38, 240])
Q: torch.Size([32, 38, 240])
K: torch.Size([32, 38, 240])
V: torch.Size([32, 38, 240])
Q: torch.Size([32, 8, 38, 30])
K: torch.Size([32, 8, 38, 30])
V: torch.Size([32, 8, 38, 30])
attention_scores: torch.Size([32, 8, 38, 38])
extended_img_mask: torch.Size([32, 1, 1, 38])
attention_scores: torch.Size([32, 8, 38, 38])
after softmax attention_scores: torch.Size([32, 8, 38, 38])
res: torch.Size([32, 8, 38, 30])
res: torch.Size([32, 38, 240])
res: torch.Size([32, 240])
pred: torch.Size([32, 10])
```

[150]:
```
from tqdm import tqdm
model = SelfAttention(config)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3, weight_decay=0)
criterion = torch.nn.CrossEntropyLoss()

model = model.train()
epoch = 5
for e in range(epoch):
    for i,batch in enumerate(tqdm(train_dataloader)):
        train_text,train_labels,train_masks = batch
        pred,attention_probs = model(train_text,train_masks)
        loss = criterion(pred,train_labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
100%|
  | 5625/5625 [05:08<00:00, 18.24it/s]
100%|
  | 5625/5625 [06:02<00:00, 15.51it/s]
100%|
  | 5625/5625 [06:10<00:00, 15.18it/s]
100%|
  | 5625/5625 [06:02<00:00, 15.52it/s]
100%|
  | 5625/5625 [05:55<00:00, 15.82it/s]
```

[151]:
```
from sklearn import metrics
model = model.eval()
total_pred = np.array([],dtype = np.int)
total_true = np.array([],dtype = np.int)
all_attn_probs = []
for i,batch in enumerate(tqdm(dev_dataloader)):
    dev_text,dev_labels,dev_masks = batch
    pred,attention_probs = model(dev_text,dev_masks)
    pred_label = torch.argmax(pred,axis = -1)
    total_pred = np.append(total_pred,pred_label)
    total_true = np.append(total_true,dev_labels)
    all_attn_probs.append(attention_probs)
print('Accuracy:',metrics.accuracy_score(total_pred,total_true))
```

```
100%|
   | 313/313 [00:03<00:00, 91.63it/s]

Accuracy: 0.8674
```

[152]:
```
res_attn_probs = all_attn_probs[0]
for i in trange(1,len(all_attn_probs)):
    res_attn_probs = torch.vstack((res_attn_probs,all_attn_probs[i]))
print(res_attn_probs.shape)
```

```
100%|
   | 312/312 [00:14<00:00, 21.20it/s]
```

```
torch.Size([10000, 8, 33, 33])
```

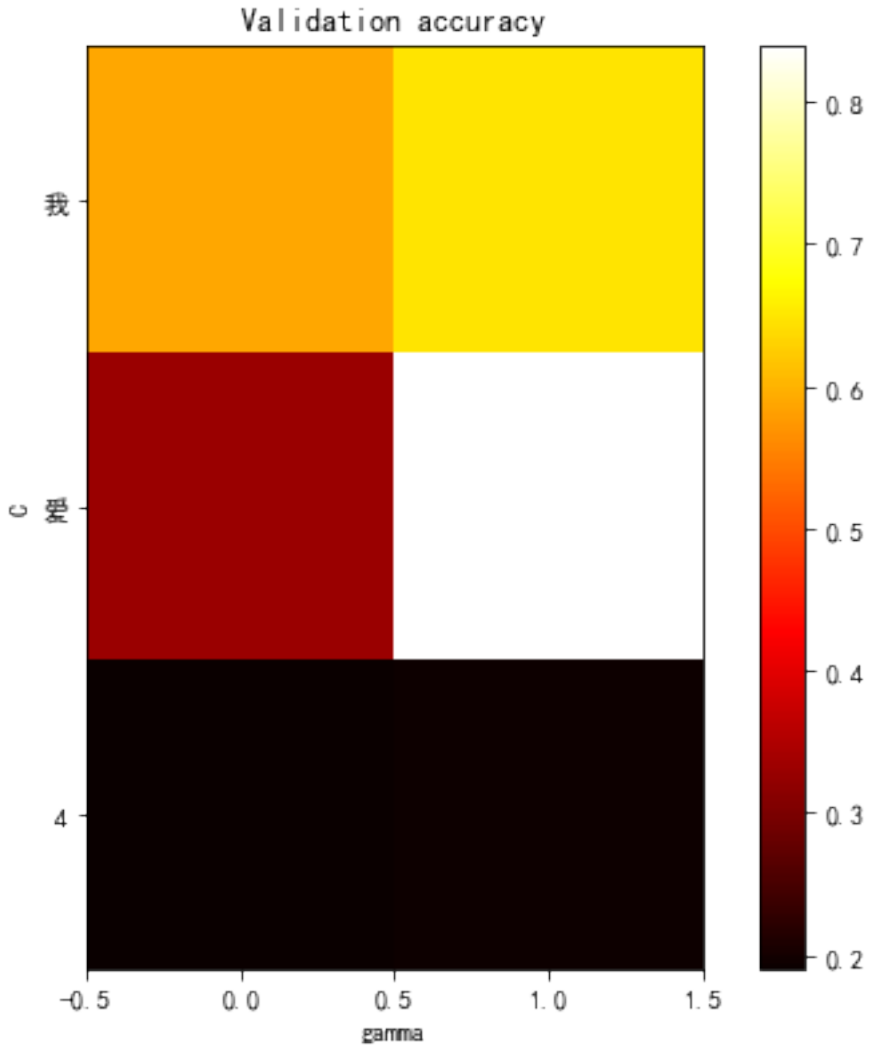
[153]:
```
avg_attn_probs = torch.mean(res_attn_probs,axis = 1)
avg_attn_probs.shape
```

[153]:
```
torch.Size([10000, 33, 33])
```

[112]:
```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False   #

plt.figure(figsize=(8, 6))
plt.subplots_adjust(left=.2, right=0.95, bottom=0.15, top=0.95)
#
scores = np.random.rand(3,2)
plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot,
        )
plt.xlabel('gamma')
plt.ylabel('C')
plt.colorbar()
# plt.xticks(np.arange(3), [1,2,3], rotation=45)
plt.yticks(np.arange(3), [' ',' ',4])

plt.title('Validation accuracy')
plt.show()
```




[154]:
```
import pickle as pkl
with open(r'C:
 ↪\Users\chunhui\Desktop\  \NLP \TextCNN_Mindspore\THUCNews\checkpoint\vocab.
 ↪pkl','rb') as f:
    vocab_dict = pkl.load(f)
id2word_dict = {value:key for key,value in vocab_dict.items()}
```

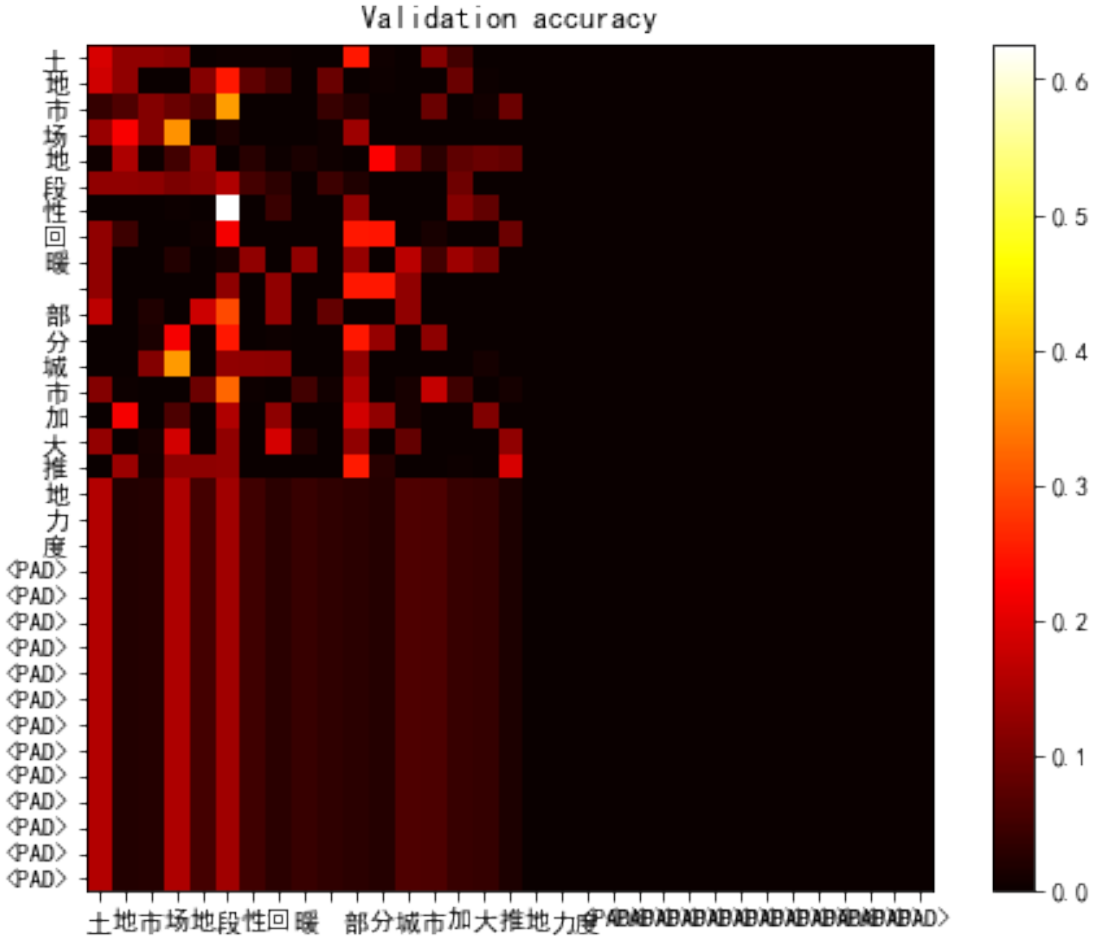
[155]:
```
import random
def␣
 ↪random_choose_one_visualize(id2word_dict,text,label,avg_attn_score,idx=None):
    if idx is None:
        idx = np.random.randint(len(text))
    text_one = text[idx].tolist()
    label_one = label[idx]
```

```
    ori_text_list = [id2word_dict[num] for num in text_one]
    ori_test = ''.join(ori_text_list)
    print('  :',ori_test)
    print('label:',label_one)
    attn_score = avg_attn_score[idx].tolist()
    plt.figure(figsize=(8, 6))
    plt.imshow(attn_score, interpolation='nearest', cmap=plt.cm.hot)
    plt.colorbar()
    plt.xticks(np.arange(len(text_one)), ori_text_list)
    plt.yticks(np.arange(len(text_one)), ori_text_list)

    plt.title('Validation accuracy')
    plt.show()
```

[156]:
```
random_choose_one_visualize(id2word_dict,dev_text,dev_labels,avg_attn_probs)
```

```
  :
     <PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD><PAD>
label: tensor(1)
```



[ ]: