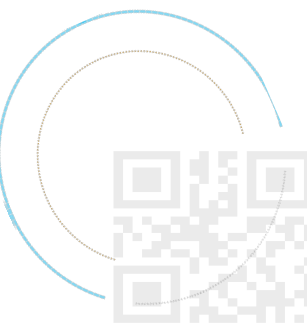


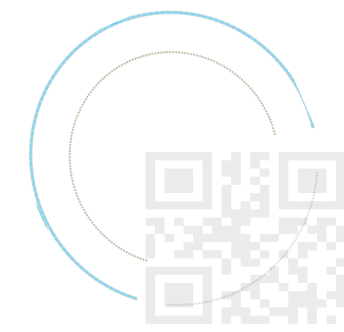
单模态和跨模态检索实践

玖强

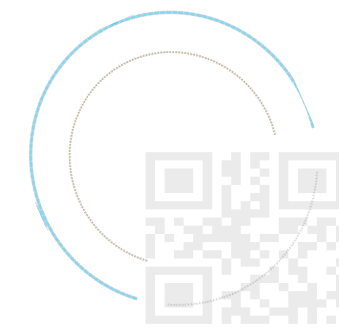


OUTLINE

- 模型介绍
- PyTorch实现指导
- 可视化word embedding

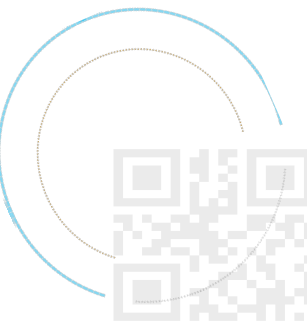
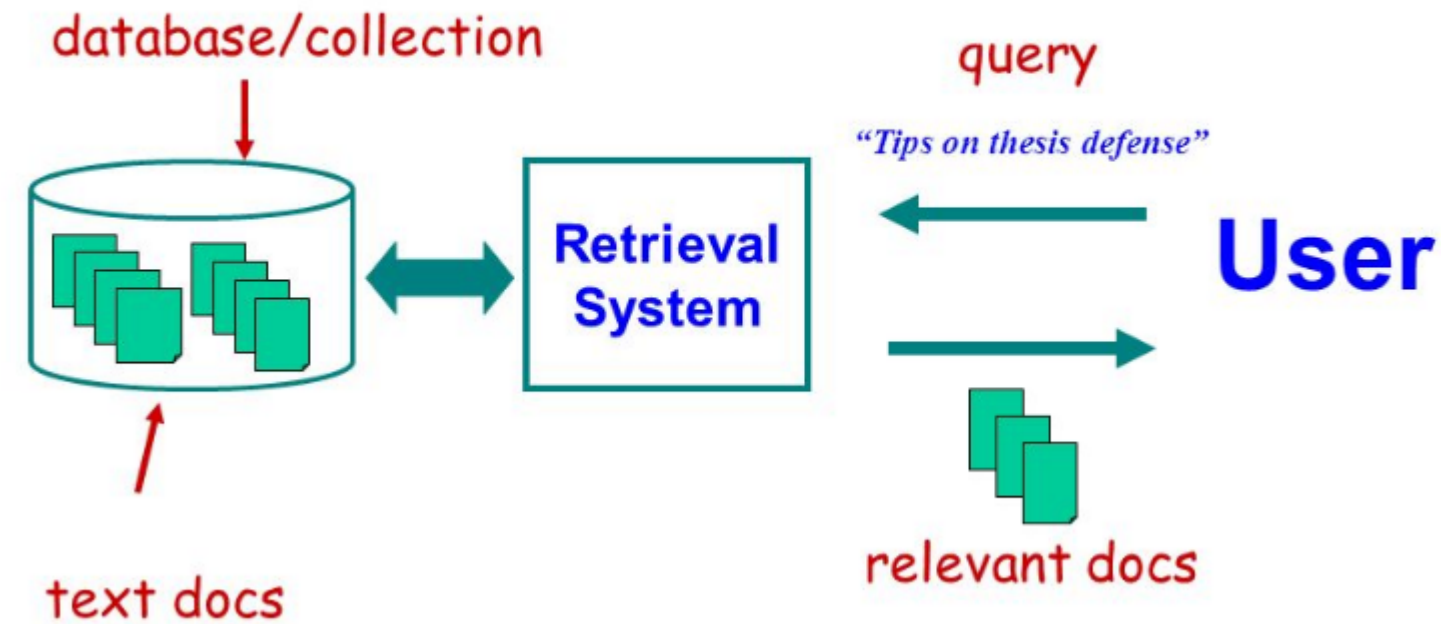


模型介绍



问题回顾

- ❑ 在下面的讲解中，我们统一称图片检索和文字检索为单模态检索，图片-文本检索是跨模特检索
- ❑ 问题的定义：
 1. 给定在domain x 的query (image, sound, or text) Q ，检索的目的是：找出数据库中和query最相似的数据 D



问题回顾

□ Given : Training data

- 定标注训练数据集 $S_{\text{train}} = \{(s_1^{(i)}, s_2^{(i)}, r^{(i)})\}_{i=1}^N$
- $s_1^{(i)} \in S_1, s_2^{(i)} \in S_2$ 是两段文本 (e.g., 查询项 vs. 答案/问题 vs. 答案) ;
- $r^{(i)}$ 表示对象 $s_1^{(i)}$ 和 $s_2^{(i)}$ 的匹配程度 (Similarity, e.g., 问题和答案的相关程度)

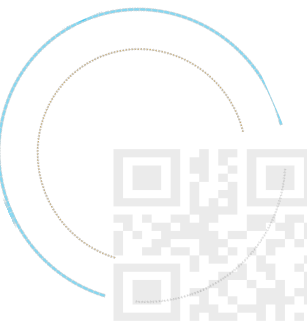
□ Target : Mapping function

- $f = S_1 * S_2 \rightarrow R$
- 对于测试数据集 $S_{\text{test}} = \{(s_1^{(i)}, s_2^{(i)})\}_{i=1}^M$ 上任意输入 ($s_1^{(i)} \in S_1, s_2^{(i)} \in S_2$)。能够预测出 $s_1^{(i)}$ 和 $s_2^{(i)}$ 的匹配程度 (Similarity) $r^{(i,j)}$;
- 然后通过匹配度排序 (Ranking) 得到最后结果

□ Example

- $s_1^{(i)}$: 从古至今, 面条和饺子是中国人喜欢的食物 ;
- $s_2^{(i)}$: 从古至今, 饺子和面条在中国都是人见人爱

} 排序问题



问题回顾

需要解决的问题：

1. 如何对query 进行编码？

Word embedding: word2vec

Sentence embedding

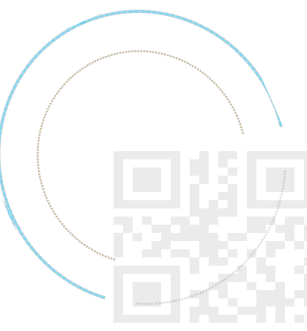
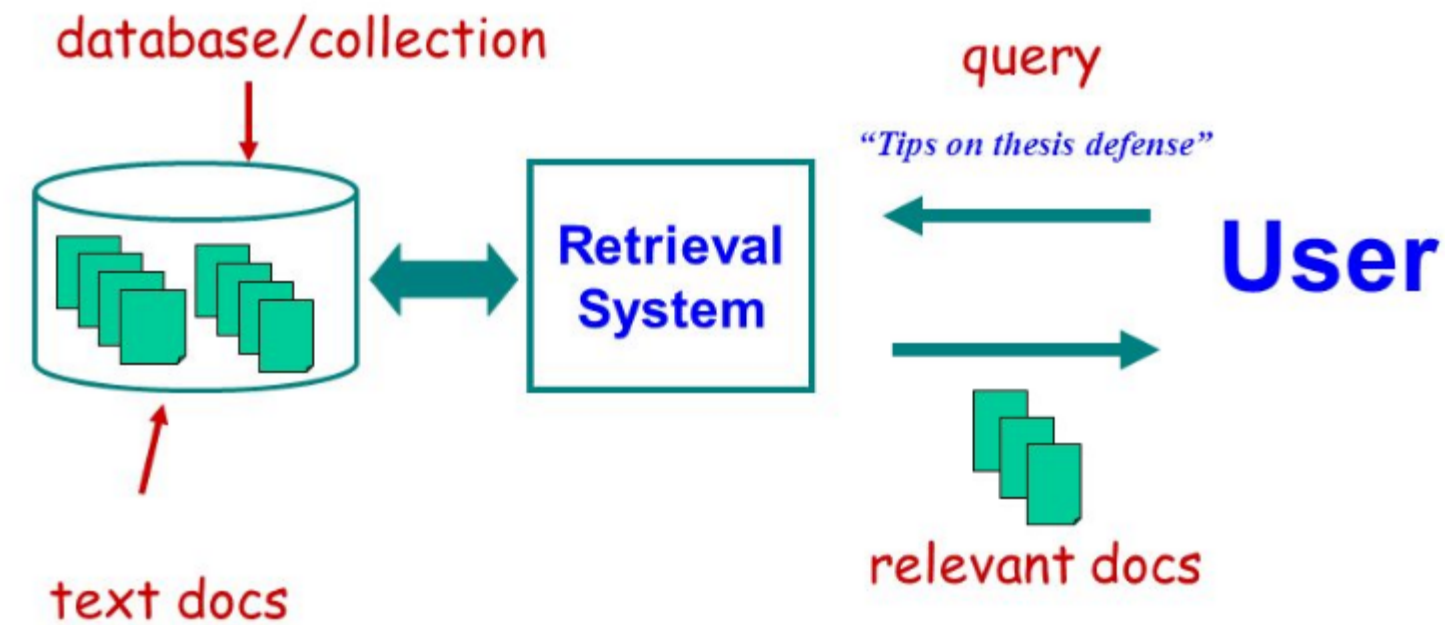
1. 如何计算query与待匹配的data之间的距离？

L2 distance

Cosine distance

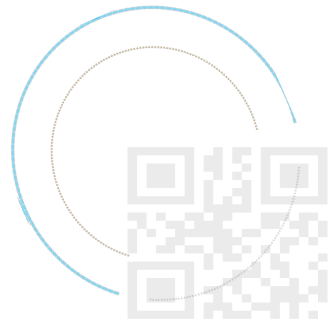
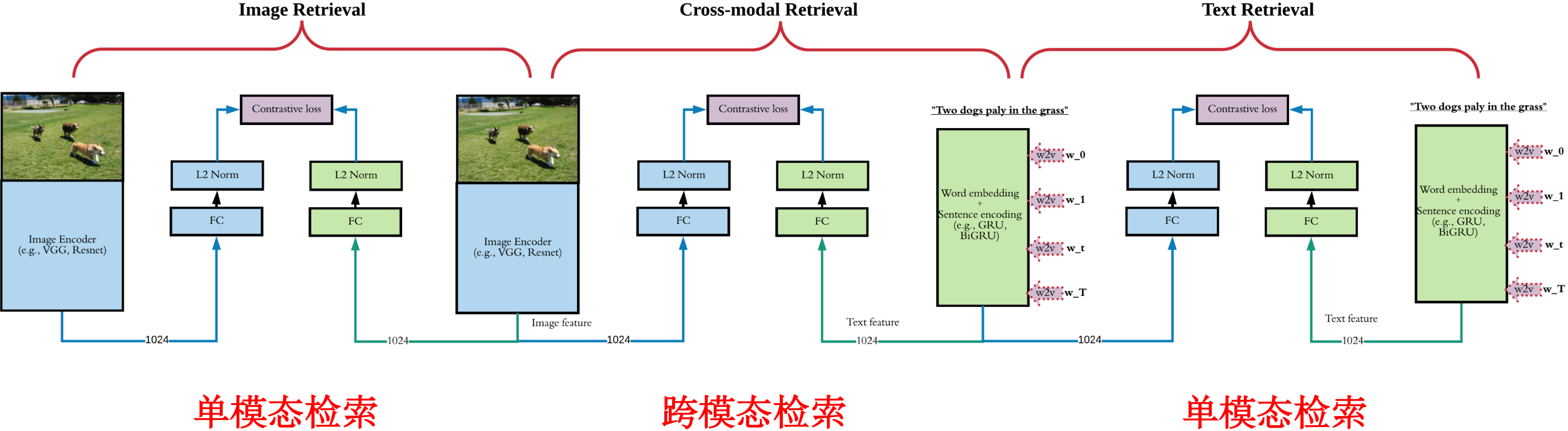
1. 查找到的relevant文本哪个才是需要的？

Ranking



模型介绍

在下面的讲解中，我们统一称图片检索和文字检索为单模态检索， 图片-文本检索是跨模态检索



工欲善其事必先利其器

❑ 安装PyTorch

目前官方的PyTorch 暂时只支持 MacOS, Linux. 不支持 Windows!

如果你想做research, 尤其是deep learning, 放弃windows吧 !

当然Tensorflow没有抛弃windows

<http://pytorch.org/>

Get Started.

Select your preferences, then run the PyTorch install command.

Please ensure that you are on the latest pip and numpy packages.
Anaconda is our recommended package manager

OS	<input checked="" type="radio"/> Linux	<input type="radio"/> OSX		
Package Manager	<input type="radio"/> conda	<input checked="" type="radio"/> pip	<input type="radio"/> Source	
Python	<input checked="" type="radio"/> 2.7	<input type="radio"/> 3.5	<input type="radio"/> 3.6	
CUDA	<input checked="" type="radio"/> 8	<input type="radio"/> 9.0	<input type="radio"/> 9.1	<input type="radio"/> None

Run this command:

```
pip install http://download.pytorch.org/whl/cu80/torch-0.3.1-cp27-cp27mu-linux_x86_64.whl
pip install torchvision
```

```
# If the above command does not work, then you have python 2.7 UCS2, use this command
pip install http://download.pytorch.org/whl/cu80/torch-0.3.1-cp27-cp27m-linux_x86_64.whl
```

[Click here for previous versions of PyTorch](#)



PyTorch Linux binaries compiled with CUDA 9.0

- [cu90/torch-0.3.0.post4-cp36-cp36m-linux_x86_64.whl](#)
- [cu90/torch-0.3.0.post4-cp35-cp35m-linux_x86_64.whl](#)
- [cu90/torch-0.3.0.post4-cp27-cp27mu-linux_x86_64.whl](#)
- [cu90/torch-0.3.0.post4-cp27-cp27m-linux_x86_64.whl](#)

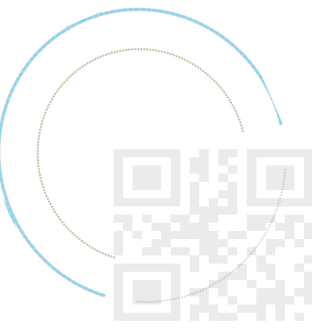
PyTorch Linux binaries compiled with CUDA 8

- [cu80/torch-0.3.0.post4-cp36-cp36m-linux_x86_64.whl](#)
- [cu80/torch-0.3.0.post4-cp35-cp35m-linux_x86_64.whl](#)
- [cu80/torch-0.3.0.post4-cp27-cp27mu-linux_x86_64.whl](#)
- [cu80/torch-0.3.0.post4-cp27-cp27m-linux_x86_64.whl](#)
- [cu80/torch-0.2.0.post2-cp36-cp36m-manylinux1_x86_64.whl](#)
- [cu80/torch-0.2.0.post2-cp35-cp35m-manylinux1_x86_64.whl](#)
- [cu80/torch-0.2.0.post2-cp27-cp27mu-manylinux1_x86_64.whl](#)
- [cu80/torch-0.2.0.post2-cp27-cp27m-manylinux1_x86_64.whl](#)
- [cu80/torch-0.2.0.post3-cp36-cp36m-manylinux1_x86_64.whl](#)
- [cu80/torch-0.2.0.post3-cp35-cp35m-manylinux1_x86_64.whl](#)



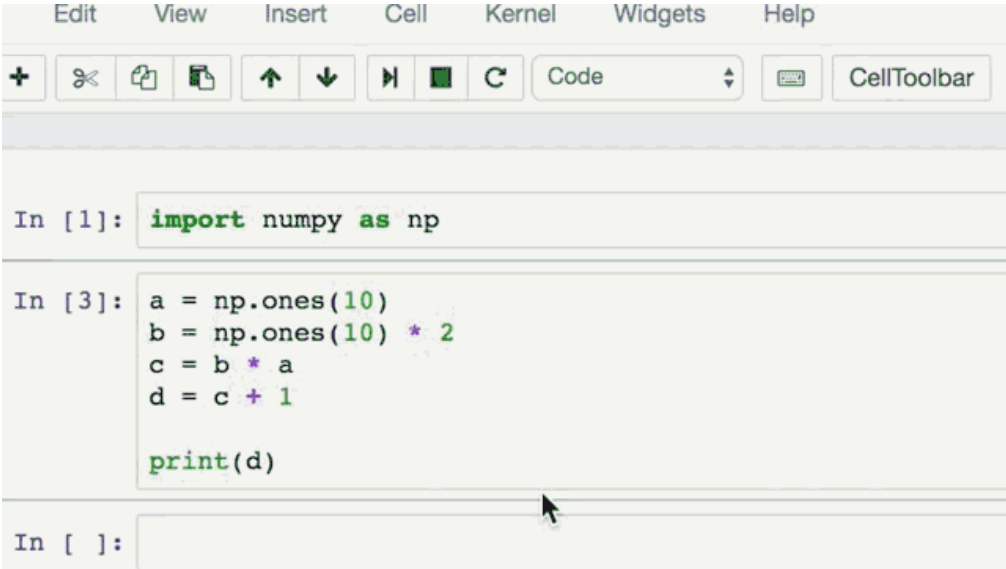
PYTORCH 属性

- ❑ **torch.Tensor** - 一个多维数组
- ❑ **autograd.Variable** - 改变Tensor并且记录下来操作的历史记录。和Tensor拥有相同的API，以及backward()的一些API。同时包含着和张量相关的梯度。
- ❑ **nn.Module** - 神经网络模块。便捷的数据封装，能够将运算移往GPU，还包括一些输入输出的东西。
- ❑ **nn.Parameter** - 一种变量，当将任何值赋予Module时自动注册为一个参数。
- ❑ **autograd.Function** - 实现了使用自动求导方法的前馈和后馈的定义。每个Variable的操作都会生成至少一个独立的Function节点，与生成了Variable的函数相连之后记录下操作历史。



PYTORCH可以自动求导！

❑ PyTorch的第一个关键特性是命令式编程

A screenshot of a Jupyter Notebook interface. The top menu bar includes 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for adding, deleting, and running code cells. The main area shows two code cells. The first cell, labeled 'In [1]:', contains the code 'import numpy as np'. The second cell, labeled 'In [3]:', contains the code 'a = np.ones(10)', 'b = np.ones(10) * 2', 'c = b * a', 'd = c + 1', and 'print(d)'. The output of the second cell is not visible. The interface is clean and professional, typical of a data science environment.

```
In [1]: import numpy as np

In [3]: a = np.ones(10)
        b = np.ones(10) * 2
        c = b * a
        d = c + 1

        print(d)

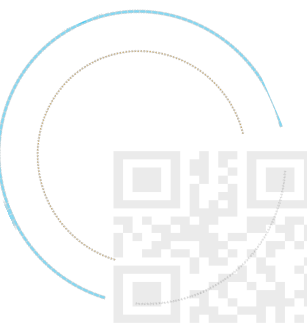
In [ ]:
```

❑ PyTorch中的神经网络

接下来介绍pytorch中的神经网络部分。PyTorch中所有的神经网络都来自于autograd包

```
from torch.autograd import Variable
```

autograd.Variable 这是这个包中最核心的类。它包装了一个**Tensor**，并且几乎支持所有的定义在其上的操作。一旦完成了你的运算，**你可以调用 `.backward()` 来自动计算出所有的梯度。**

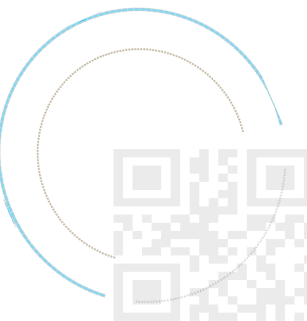
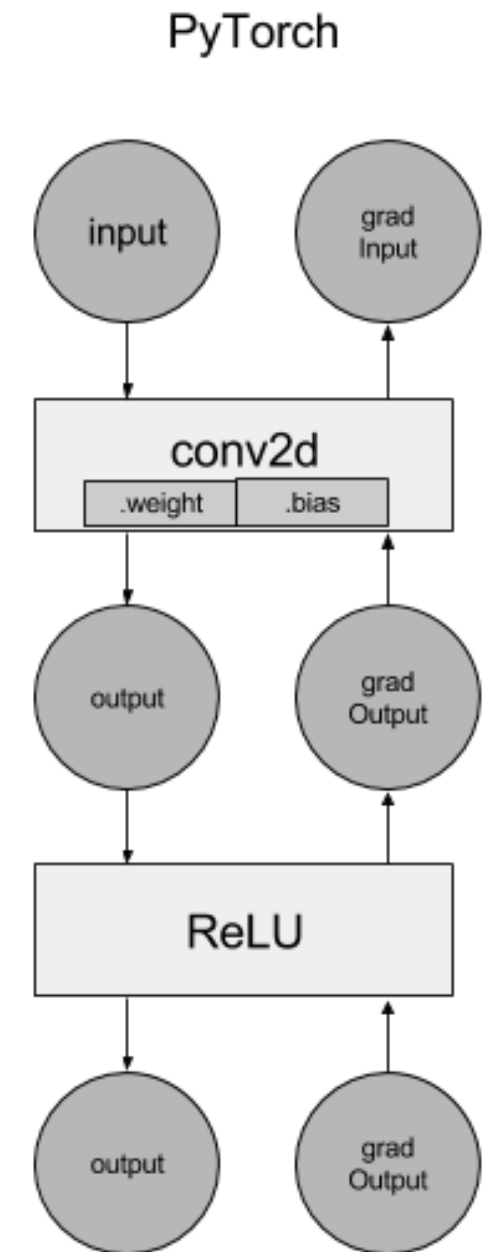
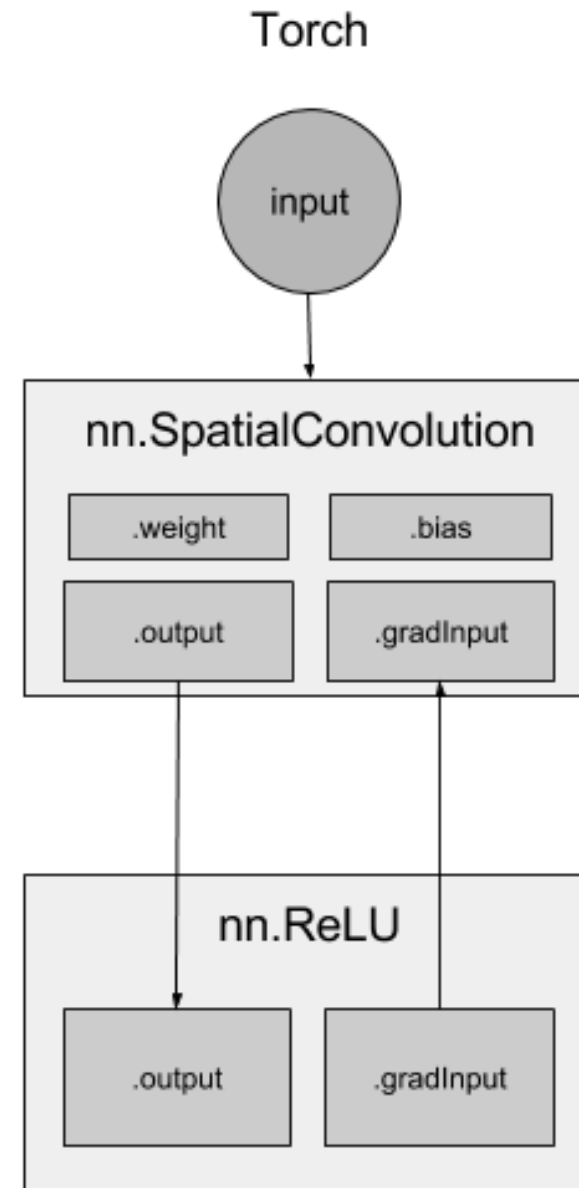


PYTORCH和MATLAB一样可以直接符号计算

pytorch和torch的对比。pytorch将所有的Container都用autograd替代。这样的话根本不需要用ConcatTable, CAddTable之类的。直接用符号运算就行了。

`output = nn.CAddTable():forward({input1, input2})` 直接用
`output = input1 + input2` 就行。

从右图看出，pytorch的网络模块只有.weight和.bias。而那些梯度.gradInput和.output都被消除。



任何输入到网络的数据，首先要变成**VARIABLE**

- ❑ 从cpu获得的数据需要首先转换成varibale
- ❑ 将require_grad设置为false，因为在反向传播过程中我们不需要计算这些变量的梯度。
- ❑ 如果你用到gpu, 这需要加.cuda()
- ❑ autograd将使我们的网络自动实现反向传播。然后定义批量大小 输入单元数量 隐藏单元数量和输出单元数量，然后使用这些值来辅助定义张量 用于保持输入和输出，将它们装饰在变量中；
- ❑ 对于需要计算的地方则需要设置成true

```
# Code in file autograd/two_layer_net_autograd.py
import torch
from torch.autograd import Variable

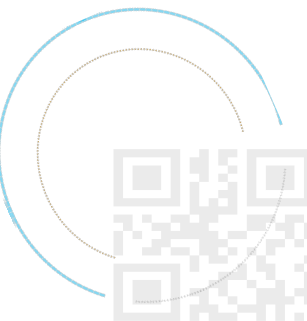
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold input and outputs, and wrap them in Variables.
# Setting requires_grad=False indicates that we do not need to compute gradients
# with respect to these Variables during the backward pass.
x = Variable(torch.randn(N, D_in).type(dtype), requires_grad=False)
y = Variable(torch.randn(N, D_out).type(dtype), requires_grad=False)

# Create random Tensors for weights, and wrap them in Variables.
# Setting requires_grad=True indicates that we want to compute gradients with
# respect to these Variables during the backward pass.
w1 = Variable(torch.randn(D_in, H).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H, D_out).type(dtype), requires_grad=True)
```

autograd.Variable - 改变Tensor并且记录下来操作的历史记录。和Tensor拥有相同的API，以及backward()的一些API。同时包含着和张量相关的梯度。



让我们来定义一个神经网络

```
import torch.nn as nn
import torch.nn.functional as F

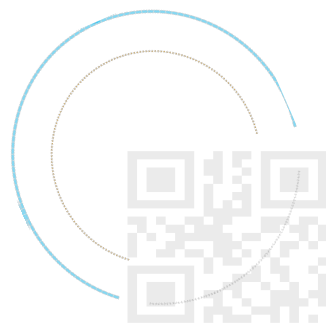
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5) # 1 input image channel, 6 output channels, 5x5 kernel
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*5*5, 120) # an affine operation: y = Wx + b
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2)) # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv2(x)), 2) # If the size is a square you can simply use 2
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:] # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

net = Net()
net

'''神经网络的输出结果是这样的
Net (
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear (400 -> 120)
  (fc2): Linear (120 -> 84)
  (fc3): Linear (84 -> 10)
)
...'''
```



网络有了，那么**LOSS**函数呢？

PyTorch已经实现了几乎所有的loss函数，如果没有你要的，你可以用基本的loss函数组合。

而你唯一要做的是理解并明白为什么用这个loss!

torch.sparse

torch.Storage

torch.nn

torch.nn.functional

- Convolution functions
- Pooling functions
- Non-linear activation functions
- Normalization functions
- Linear functions
- Dropout functions
- Distance functions
- Loss functions**
 - binary_cross_entropy
 - poisson_nll_loss
 - cosine_embedding_loss
 - cross_entropy
 - hinge_embedding_loss
 - kl_div
 - l1_loss
 - mse_loss
 - margin_ranking_loss
 - multilabel_margin_loss
 - multilabel_soft_margin_loss
 - multi_margin_loss
 - nll_loss
 - binary_cross_entropy_with_logits
 - smooth_l1_loss
 - soft_margin_loss
 - triplet_margin_loss
- Vision functions

Loss functions

```
torch.nn.functional.binary_cross_entropy(input, target, weight=None, size_average=True, reduce=True) [source]
```

Function that measures the Binary Cross Entropy between the target and the output.

See `BCELoss` for details.

- Parameters:
- `input` - Variable of arbitrary shape
 - `target` - Variable of the same shape as input
 - `weight` (*Variable, optional*) - a manual rescaling weight if provided it's repeated to match input tensor shape
 - `size_average` (*bool, optional*) - By default, the losses are averaged over observations for each minibatch. However, if the field `size_average` is set to `False`, the losses are instead summed for each minibatch. Default: `True`
 - `reduce` (*bool, optional*) - By default, the losses are averaged or summed over observations for each minibatch depending on `size_average`. When `reduce` is `False`, returns a loss per input/target element instead and ignores `size_average`. Default: `True`

Examples:

```
>>> input = torch.randn(3, requires_grad=True)
>>> target = torch.LongTensor(3).random_(2)
>>> loss = F.binary_cross_entropy(F.sigmoid(input), target)
>>> loss.backward()
```

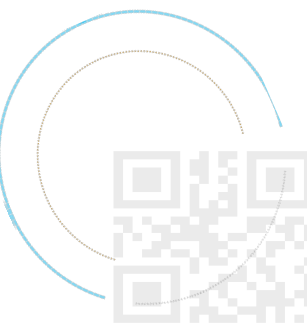
```
torch.nn.functional.poisson_nll_loss(input, target, log_input=True, full=False, size_average=True, eps=1e-08, reduce=True) [source]
```

Poisson negative log likelihood loss.

See `PoissonNLLLoss` for details.



MODEL 和 LOSS都有了，那么怎么调？

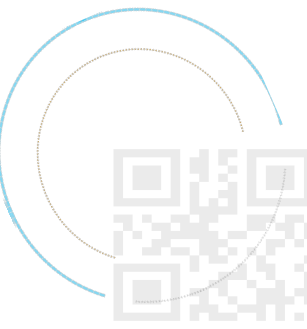


MODEL 和 LOSS都有了，那么怎么调？

直接调用包torch.optim来实现这个功能，其中包含着所有的这些方法。用起来也非常简单：

```
import torch.optim as optim
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr = 0.01)

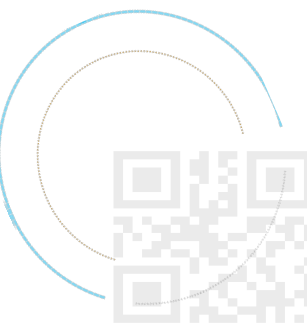
# in your training loop:
optimizer.zero_grad() # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step() # Does the update
```



什么都有了，那么数据怎么办呢？

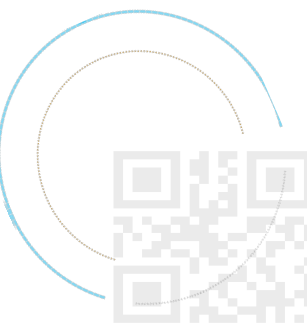
通常来讲，当你处理图像，声音，文本，视频时需要使用python中其他独立的包来将他们转换为numpy中的数组，之后再转换为torch.*Tensor。

- 图像的话，可以用Pillow, OpenCV。
- 声音处理可以用scipy和librosa。
- 文本的处理使用原生Python或者Cython以及NLTK和SpaCy都可以。
- 特别的对于图像，我们有torchvision这个包可用,其中包含了一些现成的数据集如：Imagenet, CIFAR10, MNIST等等。同时还有一些转换图像用的工具。这非常的方便并且避免了写样板代码。
- 对于文本，可以使用torchtext
- 加载函数也可以用：torch.utils.data.DataLoader



回到我们的task

https://github.com/gujiuxiang/NLP_Practice.PyTorch/blob/master/20180326_text_retrieval



文本编码-模型定义

```
class EncoderText(nn.Module):
```

```
    def __init__(self, opt):
```

```
        super(EncoderText, self).__init__()
```

```
        self.use_abs = opt.use_abs
```

```
        self.rnn_type = getattr(opt, 'rnn_type', 'GRU')
```

```
        self.num_layers = opt.num_layers
```

```
        self.embed_size = opt.embed_size
```

```
        self.batch_size = opt.batch_size
```

```
        # word embedding
```

```
        self.embed = nn.Embedding(opt.vocab_size, opt.word_dim)
```

```
        # caption embedding
```

```
        if 'Bi' in self.rnn_type:
```

```
            self.rnn = nn.GRU(opt.word_dim, opt.embed_size, opt.num_layers, bias=False, batch_first=True, bidirectional=1)
```

```
            self.bi_out = nn.Linear(opt.embed_size * 2, opt.embed_size)
```

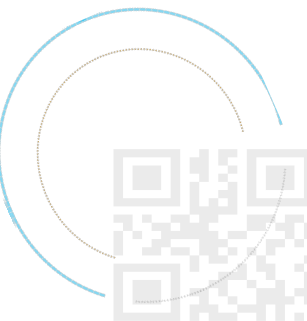
```
        else:
```

```
            self.rnn = nn.GRU(opt.word_dim, opt.embed_size, opt.num_layers, batch_first=True)
```

```
        self.init_weights()
```

多少层，一般1-2层就可以

Batch就是一次前向传播的数据量，一般看gpu显存。偶数



文本编码-模型初始化

```
def init_weights(self):
    self.embed.weight.data.uniform_(-0.1, 0.1)
    if 'Bi' in self.rnn_type:
        """Xavier initialization for the fully connected layer
        """
        r = np.sqrt(6.) / np.sqrt(self.bi_out.in_features + self.bi_out.out_features)
        self.bi_out.weight.data.uniform_(-r, r)
        self.bi_out.bias.data.fill_(0)
```

Weight Initialization matters!!! 深度学习中的weight initialization对模型收敛速度和模型质量有重要影响！

nninit

Weight initialization schemes for PyTorch nn.Modules. This is a port of the popular [nninit](#) for [Torch7](#) by [@kaixhin](#).

##Update

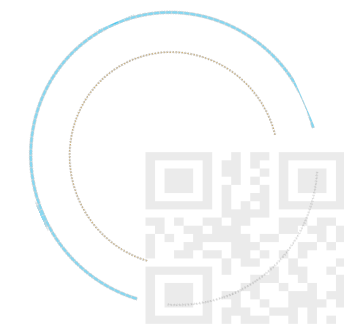
This repo has been merged into [PyTorch's nn module](#), I recommend you use that version going forward.

###PyTorch Example

```
import nninit
from torch import nn
import torch.nn.init as init
import numpy as np

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(5, 10, (3, 3))
        init.xavier_uniform(self.conv1.weight, gain=np.sqrt(2))
        init.constant(self.conv1.bias, 0.1)

network = Net()
```



文本编码-前向传播

```
def forward_GRU(self, x, lengths):
    """Handles variable size captions
    """
    # Embed word ids to vectors
    x = self.embed(x)
    packed = pack_padded_sequence(x, lengths, batch_first=True)

    # Forward propagate RNN
    out, _ = self.rnn(packed)

    # Reshape *final* output to (batch_size, hidden_size)
    padded = pad_packed_sequence(out, batch_first=True)
    I = torch.LongTensor(lengths).view(-1, 1, 1)
    I = Variable(I.expand(x.size(0), 1, self.embed_size)-1).cuda()
    out = torch.gather(padded[0], 1, I).squeeze(1)

    # normalization in the joint embedding space
    out = l2norm(out)

    # take absolute value, used by order embeddings
    if self.use_abs:
        out = torch.abs(out)
    return out
```

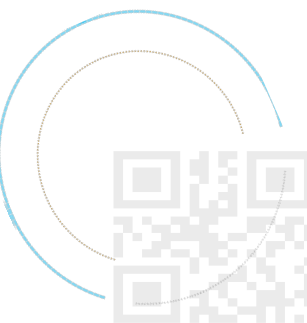
数据从这里给进来

Word embedding, 从one-hot到300维向量

当输入只有一个时，即forward中只有一个x：def forward(self, x):..., 此时非常简单，只需要在调用RNN之前用torch.nn.utils.rnn.pack_padded_sequence(input, length) 对x进行包装即可，这里，需要注意的是x内的seq需要按长度降序排列。

大致流程是这样的：

batch seq -> sort -> pad and pack -> process using RNN -> unpack -> unsort



编码后，如何计算2个句子的相似度， **LOSS**定义

\mathbf{x} = image embedding; \mathbf{v} = caption embedding; \mathbf{x}_k = contrastive image embedding; \mathbf{v}_k = contrastive caption embedding

α = margin term, inspired by hinge loss functions. It was set to 0.2; s = cosine similarity function

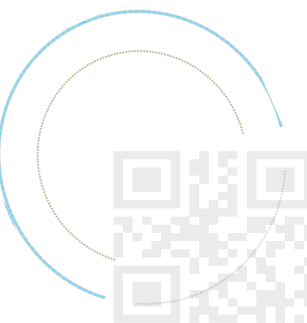
$$\min_{\theta} \sum_{\mathbf{x}} \sum_k \max\{0, \alpha - s(\mathbf{x}, \mathbf{v}) + s(\mathbf{x}, \mathbf{v}_k)\} + \sum_{\mathbf{v}} \sum_k \max\{0, \alpha - s(\mathbf{v}, \mathbf{x}) + s(\mathbf{v}, \mathbf{x}_k)\}$$

Similarity of **related pair** has a **negative relationship** with cost

- high similarity = low cost
- low similarity = high cost

Similarity of **contrastive pair** has a **positive relationship** with cost

- high similarity = high cost
- low similarity = low cost



编码后，如何计算2个句子的相似度， **LOSS**定义

$$\ell_{SH}(i, c) = \sum_{\hat{c}} [\alpha - s(i, c) + s(i, \hat{c})]_+ + \sum_{\hat{i}} [\alpha - s(i, c) + s(\hat{i}, c)]_+,$$

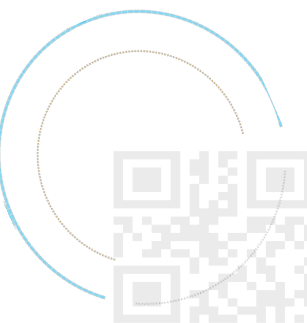
2.3 EMPHASIS ON HARD NEGATIVES

Inspired by common loss functions used in structured prediction (Tsochantaridis et al. (2005); Yu & Joachims (2009); Felzenszwalb et al. (2010)), we focus on hard negatives for training, i.e., the negatives closest to each training query. This is particularly relevant for retrieval since it is the hardest negative that determines success or failure as measured by R@1.

Given a positive pair (i, c) , the hardest negatives are given by $i' = \arg \max_{j \neq i} s(j, c)$ and $c' = \arg \max_{d \neq c} s(i, d)$. To emphasize hard negatives we therefore define our loss as

$$\ell_{MH}(i, c) = \max_{c'} [\alpha + s(i, c') - s(i, c)]_+ + \max_{i'} [\alpha + s(i', c) - s(i, c)]_+. \quad (6)$$

利用**Hard negative mining**方法，从负样本中选取一些有代表性的负样本



编码后，如何计算2个句子的相似度， **LOSS**定义

```
def forward(self, emb1, emb2):
    # compute image-sentence score matrix
    scores = self.sim(emb1, emb2)
    diagonal = scores.diag().view(emb1.size(0), 1)
    d1 = diagonal.expand_as(scores)
    d2 = diagonal.t().expand_as(scores)

    # compare every diagonal score to scores in its column caption retrieval
    cost_s = (self.margin + scores - d1).clamp(min=0)
    # compare every diagonal score to scores in its row image retrieval
    cost_im = (self.margin + scores - d2).clamp(min=0)

    # clear diagonals
    mask = torch.eye(scores.size(0)) > .5
    I = Variable(mask)
    if torch.cuda.is_available():
        I = I.cuda()
    cost_s = cost_s.masked_fill(I, 0)
    cost_im = cost_im.masked_fill(I, 0)

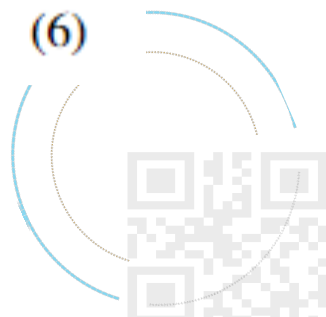
    # keep the maximum violating negative for each query
    if self.max_violation:
        cost_s = cost_s.max(1)[0]
        cost_im = cost_im.max(0)[0]
```

2.3 EMPHASIS ON HARD NEGATIVES

Inspired by common loss functions used in structured prediction (Tsochantaridis et al. (2005); Yu & Joachims (2009); Felzenszwalb et al. (2010)), we focus on hard negatives for training, i.e., the negatives closest to each training query. This is particularly relevant for retrieval since it is the hardest negative that determines success or failure as measured by R@1.

Given a positive pair (i, c) , the hardest negatives are given by $i' = \arg \max_{j \neq i} s(j, c)$ and $c' = \arg \max_{d \neq c} s(i, d)$. To emphasize hard negatives we therefore define our loss as

$$\ell_{MH}(i, c) = \max_{c'} [\alpha + s(i, c') - s(i, c)]_+ + \max_{i'} [\alpha + s(i', c) - s(i, c)]_+ . \quad (6)$$



前向传播， 方向传播， 计算梯度。。。。

```
def train_emb(self, images, captions, lengths, ids=None, *args):  
    #One training step given images and captions.  
    self.Eiters += 1  
    self.logger.update('iterations', self.Eiters)  
    self.logger.update('current learning rate', self.optimizer.param_groups[0]['lr'])
```

函数名可以自定义或或者
forward（默认，隐式调用）

```
# compute the embeddings  
img_emb, cap_emb, _ = self.forward_emb(images, captions, lengths)
```

前向传播

```
# measure accuracy and record loss  
self.optimizer.zero_grad()  
loss = self.forward_loss(img_emb, cap_emb, None, None, None)
```

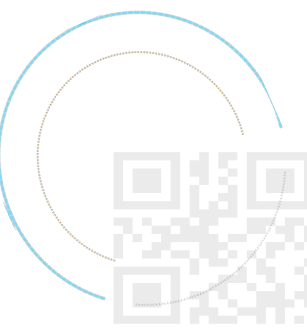
计算loss

```
# compute gradient and do SGD step  
loss.backward()
```

方向传播

```
if self.grad_clip > 0:  
    clip_grad_norm(self.params, self.grad_clip)  
self.optimizer.step()
```

梯度更新



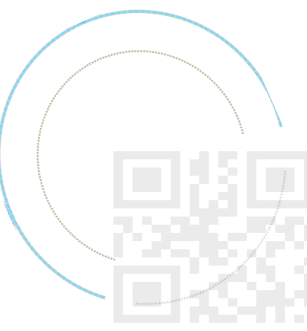
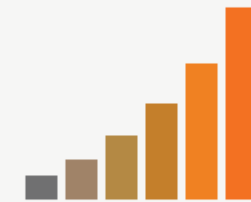
有了训练好的模型， 怎么测试？

```
# preserve the embeddings by copying from gpu and converting to numpy
img_embs[ids] = img_emb.data.cpu().numpy().copy()
cap_embs[ids] = cap_emb.data.cpu().numpy().copy()

# measure accuracy and record loss
model.forward_loss(img_emb, cap_emb, lengths, captions, kld_outs)

def forward_loss(self, img_emb, cap_emb, lengths, captions, kld_outs):
    # Compute the loss given pairs of image and caption embeddings
    loss = self.criterion(img_emb, cap_emb)
    self.logger.update('training loss', loss.data[0], img_emb.size(0))
    return loss
```

对计算的Loss排序
还记得我们之前讲的reall吗？



文本匹配问题的评价

❑ Precision at k (P@k) and Recall at k (R@k)

- 定义真实排序前k个文本中，匹配文本的数量为 G_k ，而在预测排序中前k个文本中，匹配文本的数量为 Y_k 。
- 评价指标P@k和R@k定义为：
 - $P@k = \frac{Y_k}{k}$: 所有"正确被检索的item" 占有"应该检索到的"的比例
 - $R@k = \frac{Y_k}{G_k}$: 所有"正确被检索的item" 占有"实际被检索到的"的比例。
- 形象直观的理解就是Recall要求的是全，宁可错杀一千，不能放过一人，这样Recall就会很高，但是precision就会最低

❑ MAP (Mean Average Precision/平均准确率)

- ❑ 假设预测排序中的真实匹配的文本的排序位置分别为 k_1, k_2, \dots, k_r ，其中r为整个列表中所有匹配文本的数量。
- ❑ MAP的定义为：
$$MAP = \frac{\sum_{i=1}^r P@k_i}{r}$$
- ❑ MAP是为解决P, R, F-measure的单点值局限性的，同时考虑了检索效果的排名情况。

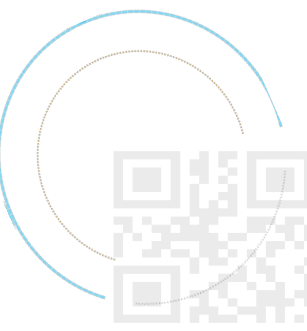
❑ Mean Reciprocal Rank (MRR). Multiple levels of relevance. Normalized Discounted Cumulative Gain (NDCG)





Hands-On !!

https://github.com/gujiuxiang/NLP_Practice.PyTorch/blob/master/20180326_text_retrieval



END

