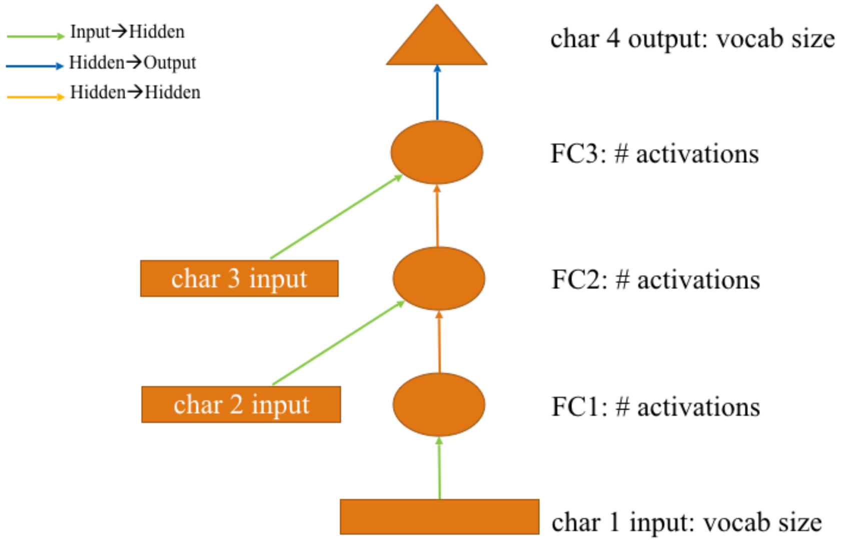

# Lesson 7

2018年8月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

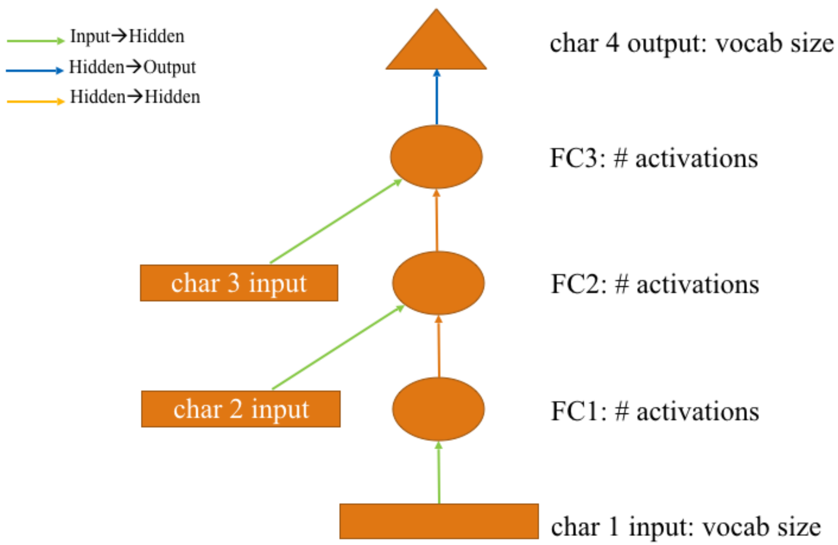
# 目录 Contents

# RNN-基本结构




- 箭头：一层或多层操作
  - 一般先执行线性函数再执行非线性函数（激活）
  - 本例中
    - 线性　　　　　矩阵乘法
    - 非线性　　　　tanh和relu
- 颜色相同：使用的权值矩阵相同

# RNN-三个字符的实现代码




```
class Char3Model(nn.Module):
    def __init__(self, vocab_size, n_fac):
        super().__init__()
        self.e = nn.Embedding(vocab_size, n_fac)

        # The 'green arrow' from our diagram
        self.l_in = nn.Linear(n_fac, n_hidden)
        # The 'orange arrow' from our diagram
        self.l_hidden = nn.Linear(n_hidden, n_hidden)
        # The 'blue arrow' from our diagram
        self.l_out = nn.Linear(n_hidden, vocab_size)

    def forward(self, c1, c2, c3):
        in1 = F.relu(self.l_in(self.e(c1)))
        in2 = F.relu(self.l_in(self.e(c2)))
        in3 = F.relu(self.l_in(self.e(c3)))

        h = V(torch.zeros(in1.size()).cuda())

          #创建空矩阵
          h = F.tanh(self.l_hidden(h+in1))
          h = F.tanh(self.l_hidden(h+in2))
          h = F.tanh(self.l_hidden(h+in3))
          return F.log_softmax(self.l_out(h))
```

# RNN-改写为循环形式

- class CharLoopModel(nn.Module):

```
def __init__(self, vocab_size, n_fac):
    super().__init__()
    self.e = nn.Embedding(vocab_size, n_fac)
    self.l_in = nn.Linear(n_fac, n_hidden)
    self.l_hidden = nn.Linear(n_hidden,
n_hidden)
    self.l_out = nn.Linear(n_hidden,
vocab_size)

def forward(self, *cs):
    bs = cs[0].size(0)
    h = V(torch.zeros(bs, n_hidden).cuda())
    for c in cs:
        inp = F.relu(self.l_in(self.e(c)))
        h = F.tanh(self.l_hidden(h+inp))

      return F.log_softmax(self.l_out(h),
dim=-1)
```

- PyTorch版本
- class CharRnn(nn.Module):

```
def __init__(self, vocab_size, n_fac):
    super().__init__()
    self.e = nn.Embedding(vocab_size,
n_fac)
    self.rnn = nn.RNN(n_fac, n_hidden)
    self.l_out = nn.Linear(n_hidden,
vocab_size)

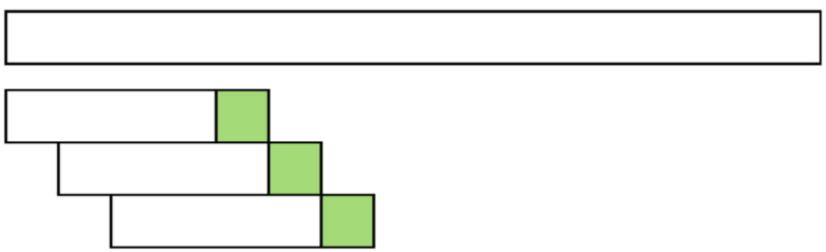
def forward(self, *cs):
    bs = cs[0].size(0)
    h = V(torch.zeros(1, bs, n_hidden))
    inp = self.e(torch.stack(cs))
    outp,h = self.rnn(inp, h)

 #rnn包括了循环以及对于h的更新操作

  return F.log_softmax(self.l_out(outp[-
1]), dim=-1)
```

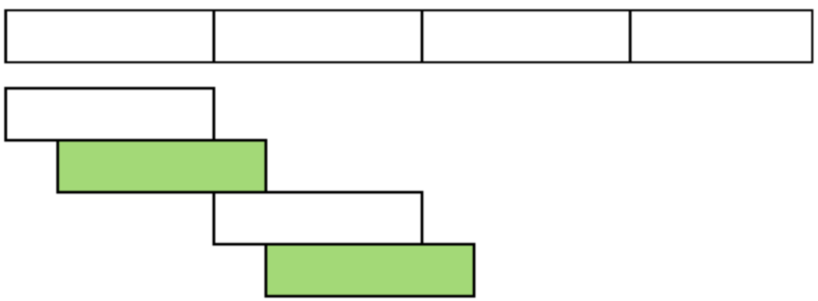
# RNN-提升效率

First *n-1* characters to predict *n*th character



白框部分大量重复，导致浪费

First *n-1* characters to predict *1* - *n*th characters



解决浪费问题，但如果使用该算法，第二个白框的初始隐层输出h将为0
解决方式：将h作为类的属性存储，不断更新，只在类初始化时置0

# RNN-修改后对比

- ```
  class CharRnn(nn.Module):
  def __init__(self, vocab_size, n_fac):
      super().__init__()
      self.e = nn.Embedding(vocab_size, 
  n_fac)
      self.rnn = nn.RNN(n_fac, n_hidden)
      self.l_out = nn.Linear(n_hidden, 
  vocab_size)

  def forward(self, *cs):
      bs = cs[0].size(0)
      h = V(torch.zeros(1, bs, n_hidden))
      inp = self.e(torch.stack(cs))
      outp,h = self.rnn(inp, h)
   #rnn包括了循环以及对于h的更新操作

    return F.log_softmax(self.l_out(outp[-
  1]), dim=-1)
  ```

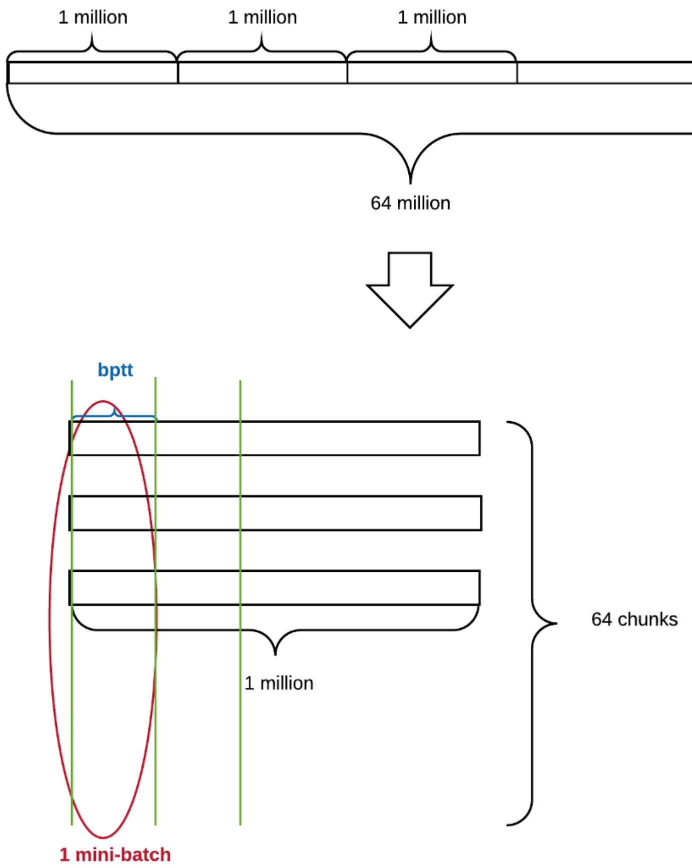
- ```
  class CharSeqStatefulRnn (nn.Module) :
  def __init __ (self,vocab_size,n_fac,bs) :
      self.vocab_size = vocab_size
      super () .__ init __ ()
      self.e = nn.Embedding (vocab_size,n_fac)
      self.rnn = nn. RNN (n_fac, n_hidden)
      self.l_out = nn.Linear (n_hidden,vocab_size)
      self.init_hidden (bs)

  def forward (self, cs) :
      bs = cs [0] .size (0)
      if self.h.size ( 1) ! = bs: self.init_hidden (bs)
      outp, h = self.rnn (self.e (cs) , self.h) #末尾
  minibatch可能较短，导致维度不同，因此重新计算一个epoch
  时重新初始化为0
      self.h = repackage_var (h) #避免浪费内存，只
  保存张量，删除操作的历史纪录，删除周期取决于bptt
      return F.log_softmax (self.l_out (outp) , dim=-
  1) .view (-1, self.vocab_size) #只支持二维，四维

  def init_hidden (self, bs) : self.h = V (torch.zeros
   (1, bs, n_hidden) )
  ```

# RNN-BPTT

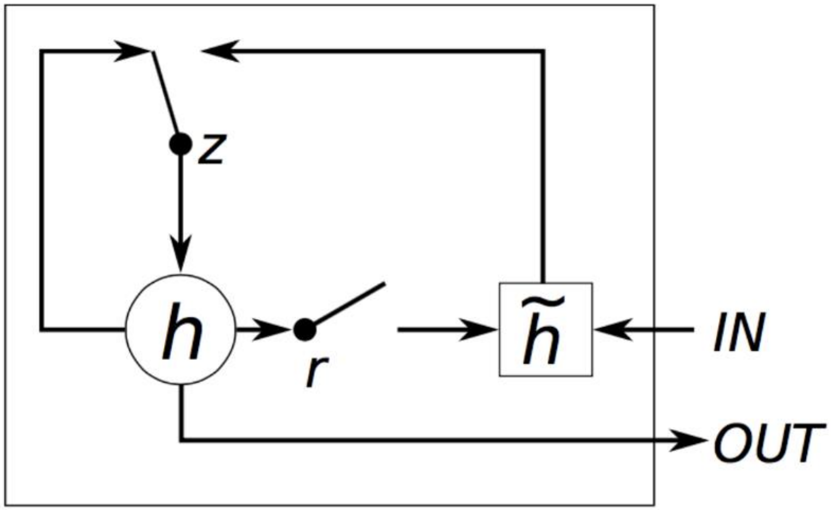
- BPTT越大，能够保留更多状态，因此BPTT应该尽量大
- 若有梯度爆炸/梯度消失的可能性，则BPTT越小，层数越小，训练越简单
- 若显存不足，则减小BPTT或BS
- 若运算太慢，则减少BPTT，因为for无法并行化（QRNN）



# RNN-RNNCell

- **def RNNCell**(input, hidden, w_ih, w_hh, b_ih, b_hh):
-     return F.tanh(F.linear(input, w_ih, b_ih) + F.linear(hidden, w_hh, b_hh))
- 缺点：可能梯度不稳定，只能采用低学习率与较小的bptt

# GRU




- r：重置门，其值由权重矩阵与之前的隐藏状态和新输入合并得到的矩阵之积通过sigmoid函数决定
- z：更新门，决定更新原隐藏状态的程度

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# GRU-GRUCell

- ```
  def GRUCell(input, hidden, w_ih, w_hh, b_ih, b_hh):
      gi = F.linear(input, w_ih, b_ih)
      gh = F.linear(hidden, w_hh, b_hh)
      i_r, i_i, i_n = gi.chunk(3, 1)
      h_r, h_i, h_n = gh.chunk(3, 1)

      resetgate = F.sigmoid(i_r + h_r) #r
      inputgate = F.sigmoid(i_i + h_i) #z
      newgate = F.tanh(i_n + resetgate * h_n)
      return newgate + inputgate * (hidden - newgate)
  ```
  ```

# LSTM

- class CharSeqStatefulLSTM(nn.Module):
- def __init__(self, vocab_size, n_fac, bs, nl):
-         super().__init__()
-         self.vocab_size,self.nl = vocab_size,nl
-         self.e = nn.Embedding(vocab_size, n_fac)
-         self.rnn = nn.LSTM(n_fac, n_hidden, nl, **dropout=0.5**)
-         self.l_out = nn.Linear(n_hidden, vocab_size)
-         self.init_hidden(bs)
- def forward(self, cs)#与之前RNN的一致
- def init_hidden(self, bs):
-         **self.h = (V(torch.zeros(self.nl, bs, n_hidden)), V(torch.zeros(self.nl, bs, n_hidden)))#单元状态与隐藏状态**

# 训练部分

- m = CharSeqStatefulLSTM(md.nt, n_fac, 512, 2).cuda()
- lo = LayerOptimizer(optim.Adam, m, 1e-2, 1e-5)#分层设置不同学习率，更好地利用预训练的权重
- on_end = lambda sched, cycle: save_model(m, f'{PATH}models/cyc_{cycle}')
- cb = [CosAnneal(lo, len(md.trn_dl), cycle_mult=2, on_cycle_end=on_end)]#采用余弦形式的学习率衰减并使用callback
- fit(m, md, 2**4-1, lo.opt, F.nll_loss, callbacks=cb)

# 测试

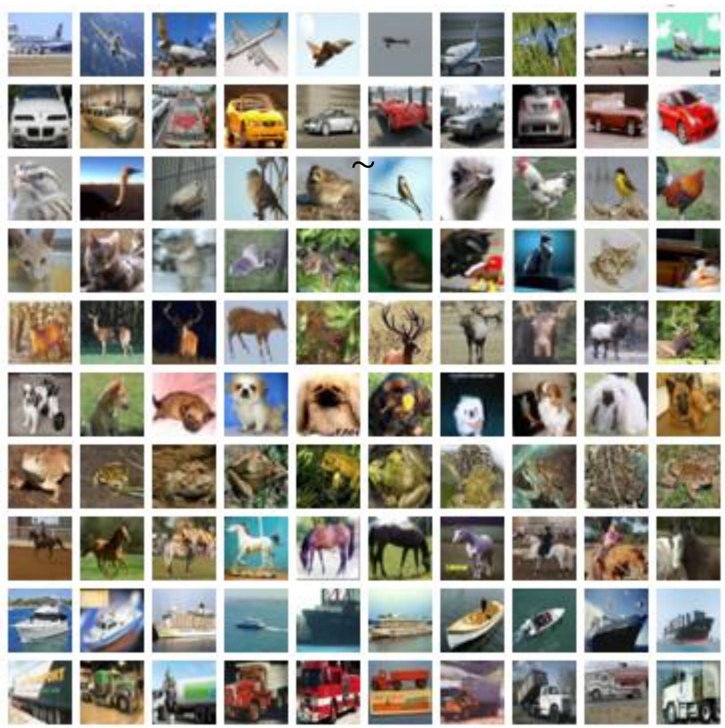
```
def get_next_n(inp, n):
    res = inp
    for i in range(n):
        c = get_next(inp)
        res += c
        inp = inp[1:]+c
    return res
print(get_next_n('these', 400))
```

- these will very where to valuest(explicity is sounder than attacommen!--203. that was upon think of christan andabourssee alsother man. my,there free-respersin proble supposely to nawary as the plike anothendout everythose at all) knew and humanitate aspectits full an estaid to hera, exquility just lives worldcorribits, benefication a5lantists.--but thyset, of distingels it, is obygermanner take throug

# 目录 Contents

# CIFAR10

- CIFAR-10 是一个包含60000张图片的数据集。其中每张照片为32*32的彩色照片，每个像素点包括RGB三个数值，数值范围 0～255。



# CIFAR10

- 需要提供训练集的均值与标准差来标准化输入数据
- 若使用训练过的模型则可使用tfms_from_model，由于我们是从头开始训练，因此这里用tfms_from_stats

In [2]:
```
from fastai.conv_learner import *
PATH = "data/cifar10/"
os.makedirs(PATH,exist_ok=True)
```

In [3]:
```
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
stats = (np.array([ 0.4914 ,  0.48216,  0.44653]), np.array([ 0.24703,  0.24349,  0.26159]))
```

均值　　标准差

In [4]:
```
def get_data(sz,bs):
    tfms = tfms_from_stats(stats, sz, aug_tfms=[RandomFlipXY()], pad=sz//8)
    return ImageClassifierData.from_paths(PATH, val_name='test', tfms=tfms, bs=bs)
```

In [5]:
```
bs=256
```

# 全连接模型

```
class SimpleNet(nn.Module):
    def __init__(self, layers):
        super().__init__()
        self.layers = nn.ModuleList([nn.Linear(layers[i], layers[i + 1]) for i in range(len(layers) - 1)])
    def forward(self, x):
        x = x.view(x.size(0), -1) #Flatten the data
        for l in self.layers:
            l_x = l(x)
            x = F.relu(l_x)
        return F.log_softmax(l_x, dim=-1)
```

在PyTorch中创建layers时均需包装在ModuleList中

遍历每一层

线性整流激活

最后softmax输出

# 从自定义模型创建学习对象

```
In [9]: learn = ConvLearner.from_model_data(SimpleNet([32*32*3, 40,10]), data)

In [10]: learn, [o.numel() for o in learn.model.parameters()]

Out[10]: (SimpleNet(
           (layers): ModuleList(
             (0): Linear(in_features=3072, out_features=40)
             (1): Linear(in_features=40, out_features=10)
           )
         ), [122880, 40, 400, 10])
```

Layer0: In (122880=32*32*3*40) Out (40)

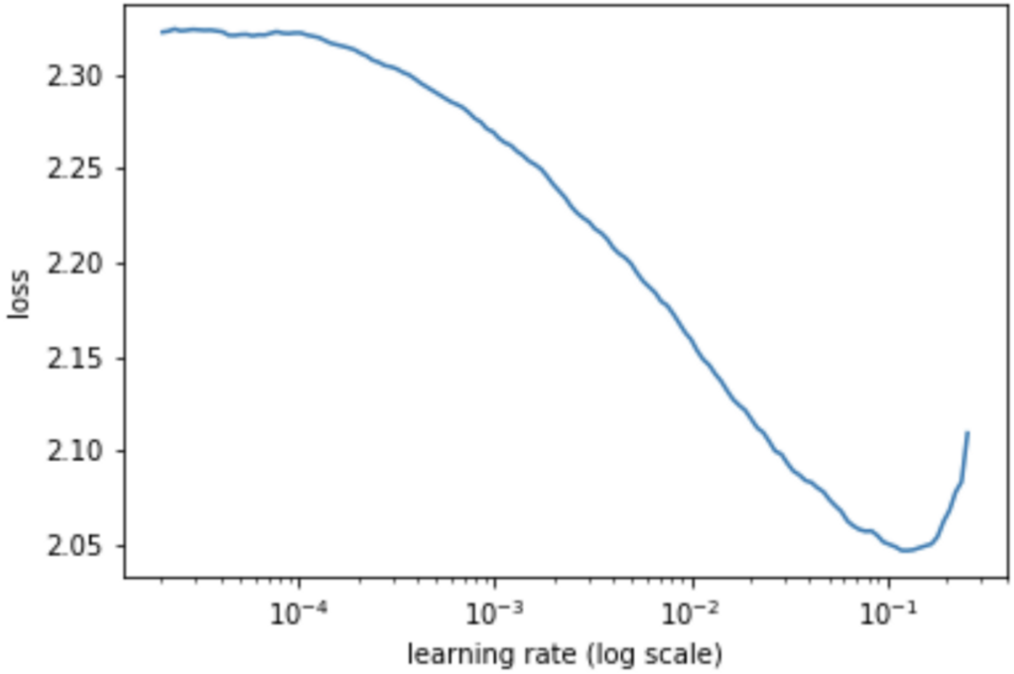
Layer1: In (400=40*10) Out (10)

```
In [11]: learn.summary()

Out[11]: OrderedDict([('Linear-1',
                       OrderedDict([('input_shape', [-1, 3072]),
                                    ('output_shape', [-1, 40]),
                                    ('trainable', True),
                                    ('nb_params', 122920)])),
                      ('Linear-2',
                       OrderedDict([('input_shape', [-1, 40]),
                                    ('output_shape', [-1, 10]),
                                    ('trainable', True),
                                    ('nb_params', 410)]))])
```

# 寻找初始学习率

- learn.lr_find()
- learn.sched.plot()



# 训练结果

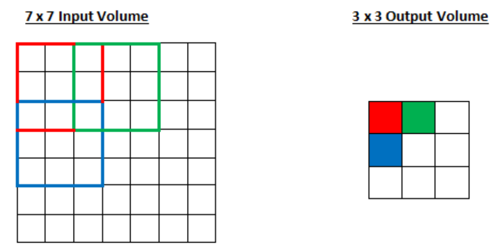
```
%time learn.fit(lr, 2)
A Jupyter Widget
[ 0.         1.7658    1.64148   0.42129]
[ 1.         1.68074   1.57897   0.44131]
CPU times: user 1min 11s, sys: 32.3 s, total: 1min 44s
Wall time: 55.1 s
%time learn.fit(lr, 2, cycle_len=1)
A Jupyter Widget
[ 0.         1.60857   1.51711   0.46631]
[ 1.         1.59361   1.50341   0.46924]
CPU times: user 1min 12s, sys: 31.8 s, total: 1min 44s
Wall time: 55.3 s
```

**1 hidden layer**

**122880 parameters**

**47% accuracy**

# 卷积神经网络 CNN




```
class ConvNet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([
            nn.Conv2d(layers[i], layers[i + 1], kernel_size=3, stride=2) #3*3, 步长为2
            for i in range(len(layers) - 1)])
        self.pool = nn.AdaptiveMaxPool2d(1)
        self.out = nn.Linear(layers[-1], c)
    def forward(self, x):
        for l in self.layers: x = F.relu(l(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

自适应最大池化：
输入的不是在几位中取最大值，而是输出的大小，函数自动计算应该采用几位。

56% accuracy

# 重构 Refactor

- 定义ConvLayer模板

```
class ConvLayer(nn.Module):
    def __init__(self, ni, nf):
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=3, stride=2, padding=1)
    def forward(self, x): return F.relu(self.conv(x))
```

卷积时，进行padding可以让我们保留图像的边缘像素信息。

- 层定义和网络的定义都包含constructer和forward两部分

# 重构 Refactor

```
class ConvNet2(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([ConvLayer(layers[i], layers[i + 1])
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)
    def forward(self, x):
        for l in self.layers: x = l(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
learn = ConvLearner.from_model_data(ConvNet2([3, 20, 40, 80], 10), data)
```

# 批标准化 BatchNorm

- 当网络层数变深时，training的难度也会越来越大
- 高学习率——NaN　　低学习率——耗时过长
- 使用BatchNorm相当于在神经网络的训练过程中对**每层**的输入数据加一个标准化处理（使其输出数据的均值接近0，其标准差接近1）

$$\widehat{x}^{(k)}=\frac{x^{(k)}-\mathrm{E}[x^{(k)}]}{\sqrt{\operatorname{Var}[x^{(k)}]}}$$

m, a: multiplier/adder

```
class BnLayer(nn.Module):
    def __init__(self, ni, nf, stride=2, kernel_size=3):
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=kernel_size, stride=stride,
                              bias=False, padding=1)
        self.a = nn.Parameter(torch.zeros(nf,1,1))
        self.m = nn.Parameter(torch.ones(nf,1,1))

    def forward(self, x):
        x = F.relu(self.conv(x))
        x_chan = x.transpose(0,1).contiguous().view(x.size(1), -1)
        if self.training:
            self.means = x_chan.mean(1)[:,None,None]
            self.stds  = x_chan.std (1)[:,None,None]
        return (x-self.means) / self.stds *self.m + self.a
```

# 批标准化 BatchNorm

- if self.training — 检测是否为训练状态，保证测试时模型参数不变
- SGD在每个mini-batch中都会撤销(x=self.means) / self.stds的操作，并在下个mini-batch中重做
- 因此我们要在每个channel中加入新的参数：multiplier和adder
- 如果想要缩放或者上下移动矩阵，不必移位和缩放整个卷积滤波器集，我们可以扩展这三个self.m，或移动这三个self.a

```
        self.a = nn.Parameter(torch.zeros(nf,1,1))
        self.m = nn.Parameter(torch.ones(nf,1,1))

    def forward(self, x):
        x = F.relu(self.conv(x))
        x_chan = x.transpose(0,1).contiguous().view(x.size(1), -1)
        if self.training:
            self.means = x_chan.mean(1)[:,None,None]
            self.stds  = x_chan.std (1)[:,None,None]
        return (x-self.means) / self.stds *self.m + self.a
```

# 批标准化 BatchNorm

- 在开始时添加一个卷积层以获得更丰富的输入
- 由于padding=(kernel_size-1)/2, stride=1，该层输入输出的规模相同，只是增加了filter数

```
class ConvBnNet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i + 1])
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        x = self.conv1(x)
        for l in self.layers: x = l(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

68% accuracy

# Deep BatchNorm

- 在每个stride=2层后添加一个stride=1层
- （如果添加的是stride=2层，每层会使图像大小减半）

stride2

stride1

twice as deep

```
class ConvBnNet2(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([BnLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        x = self.conv1(x)
        for l,l2 in zip(self.layers, self.layers2):
            x = l(x)
            x = l2(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

# Deep BatchNorm

- 从结果看，准确率变化不大。即使使用BN，也难以对这么深的网络做有效的训练。

```
%time learn.fit(1e-2, 2)
```

```
A Jupyter Widget

[ 0.         1.53499   1.43782   0.47588]
[ 1.         1.28867   1.22616   0.55537]

CPU times: user 1min 22s, sys: 34.5 s, total: 1min 56s
Wall time: 58.2 s
```

```
%time learn.fit(1e-2, 2, cycle_len=1)
```

```
A Jupyter Widget

[ 0.         1.10933   1.06439   0.61582]
[ 1.         1.04663   0.98608   0.64609]

CPU times: user 1min 21s, sys: 32.9 s, total: 1min 54s
Wall time: 57.6 s
```

12 layers deep

64% accuracy

# ResNet

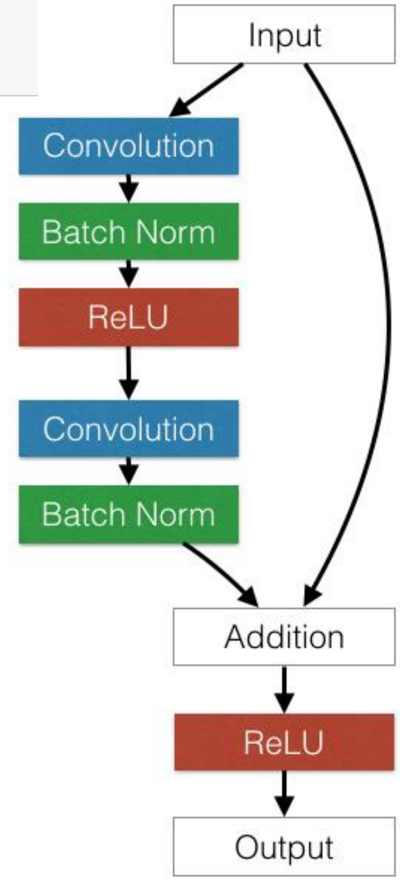
- y=x+f(x)

```
class ResnetLayer(BnLayer):
    def forward(self, x): return x + super().forward(x)
```

- (y=prediction, x=input, f(x)=y−x)

```
def __init__(self, layers, c):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
    self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
        for i in range(len(layers) - 1)])
    self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
        for i in range(len(layers) - 1)])
    self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
        for i in range(len(layers) - 1)])
    self.out = nn.Linear(layers[-1], c)

def forward(self, x):
    x = self.conv1(x)
    for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
        x = l3(l2(l(x)))
    x = F.adaptive_max_pool2d(x, 1)
    x = x.view(x.size(0), -1)
    return F.log_softmax(self.out(x), dim=-1)
```



# ResNet

- 在每个块x = 13(12(1(x)))中，第一个层不是Resnet层，而是stride=2的卷积层——这称为“瓶颈层”。
- ResNet不是卷积层，而是一种不同形式的瓶颈块，我们将在第2部分中介绍。

```
def forward(self, x):
    x = self.conv1(x)
    for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
        x = l3(l2(l(x)))
    x = F.adaptive_max_pool2d(x, 1)
    x = x.view(x.size(0), -1)
    return F.log_softmax(self.out(x), dim=-1)
```

# ResNet2

- 提高了 feature的大小，并添加了dropout
- 最终达到了85%的准确率
- 如今，通过更好的数据增强方法，更好的正规化方法以及ResNet上的一些调整，可以达到97%的准确率

# ResNet2

- 提高了 feature的大小，并添加了dropout

```
class Resnet2(nn.Module):
    def __init__(self, layers, c, p=0.5):
        super().__init__()
        self.conv1 = BnLayer(3, 16, stride=1, kernel_size=7)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)
        self.drop = nn.Dropout(p)

    def forward(self, x):
        x = self.conv1(x)
        for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
            x = l3(l2(l(x)))
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        x = self.drop(x)
        return F.log_softmax(self.out(x), dim=-1)
```

dropout

85% accuracy

```
learn = ConvLearner.from_model_data(Resnet2([16, 32, 64, 128, 256], 10, 0.2), data)
```

# Class Activation Map

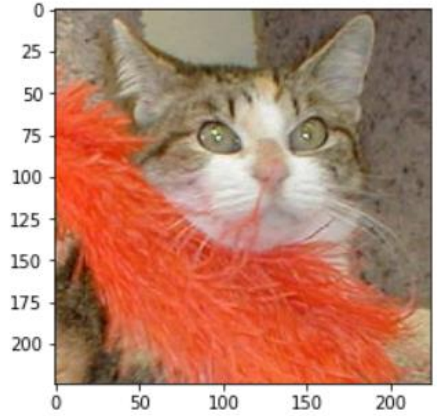
- 图像中最“有效”的部分是哪里？

In [37]:
```
class SaveFeatures():
    features=None
    def __init__(self, m): self.hook = m.register_forward_hook(self.hook_fn)
    def hook_fn(self, module, input, output): self.features = to_np(output)
    def remove(self): self.hook.remove()
```

In [38]:
```
x,y = next(iter(data.val_dl))
x,y = x[None,1], y[None,1]

vx = Variable(x.cuda(), requires_grad=True)
```

In [39]:
```
dx = data.val_ds.denorm(x)[0]
plt.imshow(dx);
```



# Class Activation Map

- 将Feature矩阵与py向量(prediction of cat)相乘
- 接近1的位置---cat like

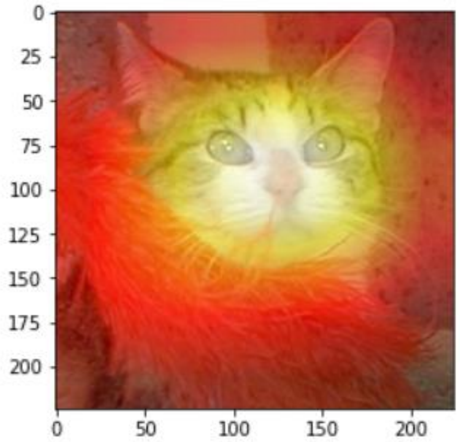
```
In  [42]: f2=np.dot(np.rollaxis(feat,0,3), py)
          f2-=f2.min()
          f2/=f2.max()
          f2

Out[42]: array([[ 0.14697,  0.27356,  0.39014,  0.43381,  0.41195,  0.33732,  0.19222],
                [ 0.22036,  0.444  ,  0.64865,  0.74039,  0.70758,  0.58523,  0.3416 ],
                [ 0.28621,  0.57486,  0.85465,  1.     ,  0.96297,  0.78594,  0.4575 ],
                [ 0.29898,  0.56478,  0.82466,  0.95066,  0.91381,  0.74047,  0.43598],
                [ 0.31433,  0.47566,  0.62147,  0.68559,  0.65911,  0.5295 ,  0.30506],
                [ 0.26731,  0.3093 ,  0.33521,  0.3214 ,  0.296  ,  0.23587,  0.12969],
                [ 0.15773,  0.13479,  0.107  ,  0.07065,  0.04057,  0.00956,  0.     ]], dtype=float32)
```

# Class Activation Map

In [44]:
```
plt.imshow(dx)
plt.imshow(scipy.misc.imresize(f2, dx.shape), alpha=0.5, cmap='hot');
```

```
/home/paperspace/anaconda3/envs/fastai/lib/python3.6/site-packages/ipykernel_launcher.py:2: DeprecationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``skimage.transform.resize`` instead.
```



# 谢 谢！