2.深度学习基本原理

1. 反向传播

2. 优化器

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

m_t ： 一阶动量

v_t ： 二阶动量

相比SGD的优势

1. 每次迭代将以前的迭代考虑进去
2. 利用了二阶参数
3. 将 $\alpha/\sqrt{V_t}$ 看错学习率， m_t 看成梯度，则梯度对学习率的影响是：梯度越大，学习率越小（合理
4. $1-\beta_1^t$ 说明当 t 增加时，这个分母增大，因此对于学习率来说是减小的。说明训练时间越长，学习率越小（合理

3. 字符数值化

1. 将单个字母用标量表示并不合理，因为标量之间存在联系，而字母之间是独立，没有这样的关系

例如：a=1,b=2，但实际上 $a+a\neq b$

2. 用embedding层，通过向量来表示字符，在pytorch中有特定的框架来做

embedding Python

```
'''
embedding层的处理
'''

num_embeddings = 6  #通常对于nlp任务，此参数为字符集字符总数
embedding_dim = 5   #每个字符向量化后的向量维度
embedding_layer = nn.Embedding(num_embeddings, embedding_dim)
```

3. 考虑一个具体的任务：判断字符串中是否包含特点字符?

・当前输入：字符串　　如：abcd
・预期输出：概率值　　正样本=1，负样本=0，以0.5为分界

・X = “abcd”　　Y = 1
・X =“bcde”　　Y = 0

・建模目标：找到一个映射f(x)，使得f(“abcd”) = 1, f(“bcde”) = 0

每个字符转化成同维度向量

a - > [0.32618175 0.20962898 0.43550067 0.07120884 0.58215387]
b - > [0.21841921 0.97431001 0.43676452 0.77925024 0.7307891]
…
z -> [0.72847746 0.72803551 0.43888069 0.09266955 0.65148562]

step 2 矩阵转化为向量

求平均

[[0.32618175 0.20962898 0.43550067 0.07120884 0.58215387]
[0.21841921 0.97431001 0.43676452 0.77925024 0.7307891]
[0.95035602 0.45280039 0.06675379 0.72238734 0.02466642]
[0.86751814 0.97157839 0.0127658 0.98910503 0.92606296]]　相加除以4

- >
[0.59061878 0.65207944 0.2379462 0.64048786 0.56591809]
由4 * 5 矩阵 -> 1* 5 向量　　形状 = 1*向量长度

step 3 向量到数值

采取最简单的线性公式 y = w * x + b
w 维度为1*向量维度　b为实数
例：
w = [1, 1], b = -1, x = [1,2]

$$[1,1] * \begin{bmatrix}1\\2\end{bmatrix} - 1 = 1*1 + 1*2 - 1 = 2$$

最初的输出Y为标量，通过阈值进行二项分类

4. 一个模型绑定唯一的词表

a. 细节：

1. 通过词表将字符映射到对应的序号
2. 得到具体序号后--->进行padding----->进行embedding

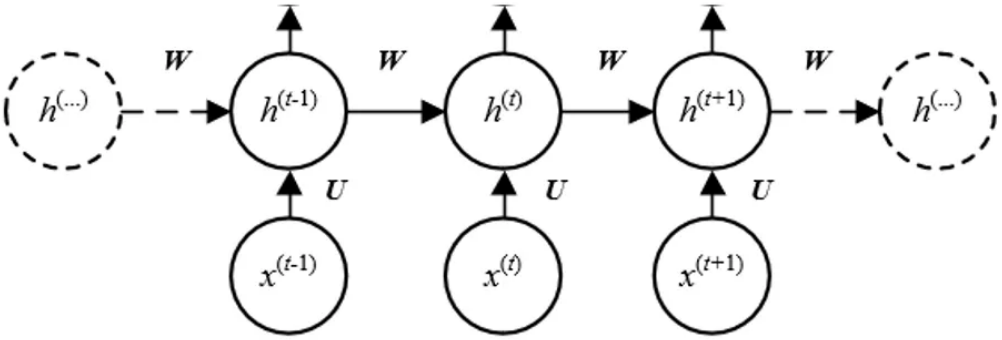
4. 池化

1. NLP一般用一维池化，CV用二维池化
2. 一般池化是在最后一个维度做的，而数据一般的维度为
   $batchSize \times$ 字符串长度 $\times$ 每个字符编码长度 ，因此需要都第二个和第三个维度交换
3. 作用：①降维，降低模型大小②提高鲁棒性、防止过拟合
4. 机器学习的本质是：有数据不知道规律，通过数据去找规律

5. RNN（循环卷积网络

公式：

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}).$$



用于处理时序相关的数据

动手实现RNN

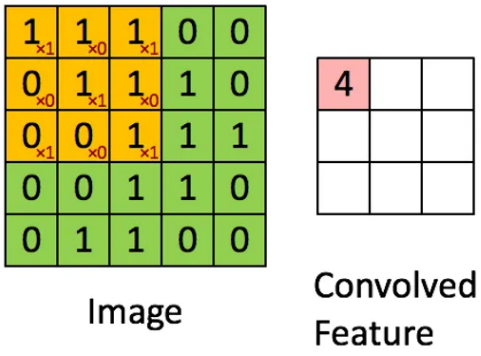
```python
class DiyRNN:
    def __init__(self, w_ih, w_hh, hidden_size):
        self.w_ih = w_ih
        self.w_hh = w_hh
        self.hidden_size = hidden_size

    def forward(self, x):
        ht = np.zeros((self.hidden_size))
        output = []
        for xt in x:
            ux = np.dot(self.w_ih, xt)
            wh = np.dot(self.w_hh, ht)
            ht_next = np.tanh(ux + wh)
            output.append(ht_next)
            ht = ht_next
        return np.array(output), ht
```

由于在NLP任务中都是将矩阵转换为向量，RNN可以替代池化层

6. CNN（卷积网络

训练参数为卷积核



动手实现RNN

Python

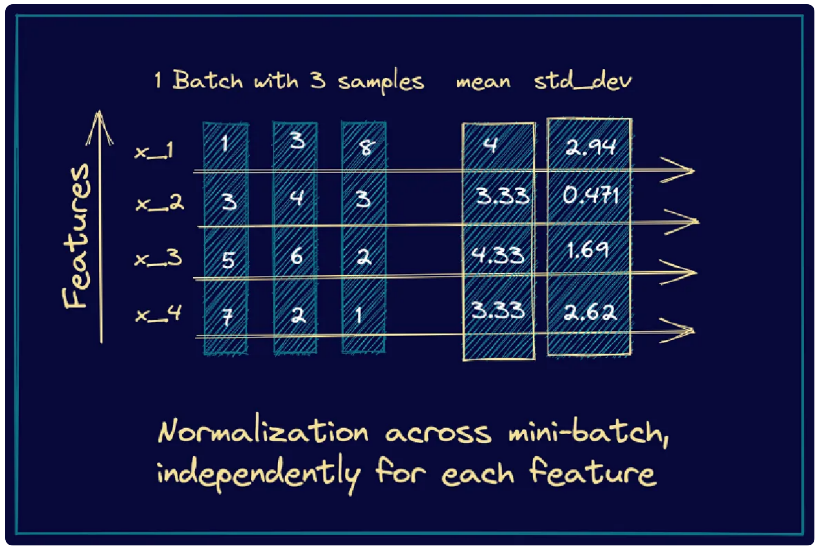
```
class DiyModel:
    def __init__(self, input_height, input_width, weights, kernel_size):
        self.height = input_height
        self.width = input_width
        self.weights = weights
        self.kernel_size = kernel_size

    def forward(self, x):
        output = []
        for kernel_weight in self.weights:
            kernel_weight = kernel_weight.squeeze().numpy() #shape : 2x2
            kernel_output = np.zeros((self.height - kernel_size + 1, self.width - kernel_size + 1))
            for i in range(self.height - kernel_size + 1):
                for j in range(self.width - kernel_size + 1):
                    window = x[i:i+kernel_size, j:j+kernel_size]
                    kernel_output[i, j] = np.sum(kernel_weight * window) # np.dot(a, b) != a * b
            output.append(kernel_output)
        return np.array(output)
```

1. 实际上框架中并不是用for循环来计算的，有并行优化
2. textCNN和CNN的不同就在于，NLP的卷积是整行整行地卷积的，这是因为每一行代表的是一个字符

7. Normalization

a. Batch Normalization

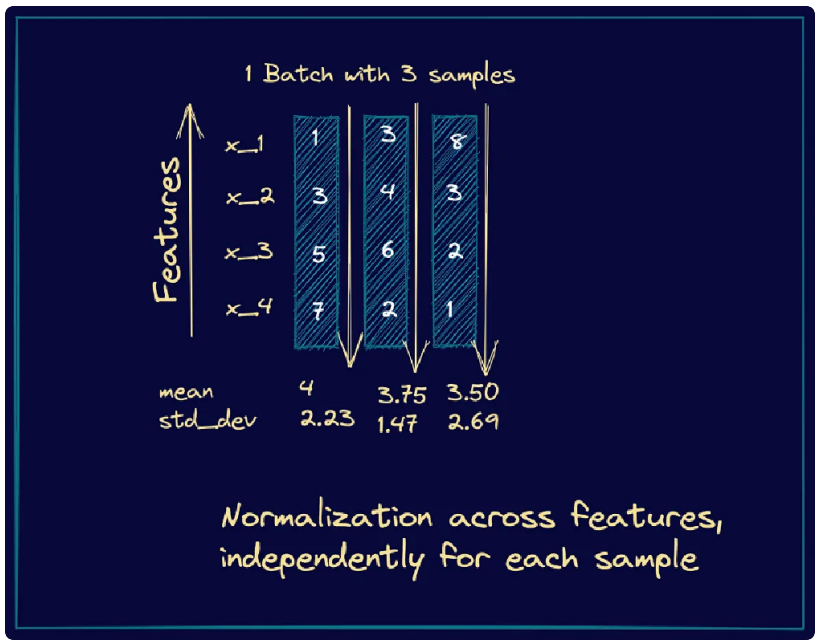


对每个样本进行归一化，使得所有样本分布接近，有利于模型训练

多个样本一起对比

在**图像**中常用

b. Layer Normalization



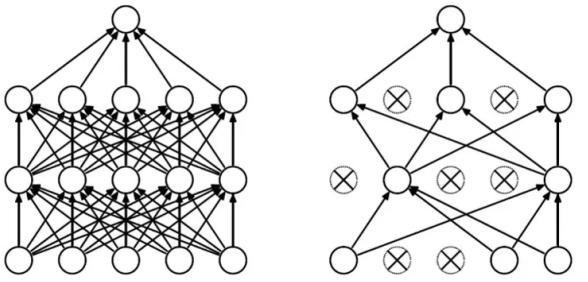
在**NLP**中常用

在单个样本中对features进行归一化（不太明白这里的必要性

NLP数据，长度并不一致，并且随着batch其他样本进行特征缩放并不符合语言的规律

8. DropOut

- 作用：减少过拟合
- 按照指定概率，随机丢弃一些神经元（将其化为零）
- 其余元素乘以 1 / (1 – p)进行放大



(a) Standard Neural Net　　(b) After applying dropout.

DropOut　　Python

```python
x = torch.Tensor([1,2,3,4,5,6,7,8,9])
dp_layer = torch.nn.Dropout(0.5)
dp_x = dp_layer(x)
print("原始向量:")
print(x)
print("DropOut后的x:")
print(dp_x)
```

```
原始向量:
tensor([1., 2., 3., 4., 5., 6., 7., 8., 9.])
DropOut后的x:
tensor([ 2.,  4.,  0.,  0.,  0.,  0., 14.,  0.,  0.])
```

- 在训练时DropOut发挥作用，相当于训练多个模型（因为每次模型的连接被随机丢弃了
- 在评估时DropOut被关闭，不发挥作用

9. 作业

多分类任务：要求得到字符串中字符的位置，如判断a在“bsadcd”中的位置，应该输出为第3类

- 提示：用RNN实现
- 缩短字符集长度，否则可能a出现的概率很小