搭建网络

```
net = nn.Sequential(nn.Conv2d(in_channels=3, out_channels=32, kernel_size=9)
    , nn.ReLU())
```

nn.Linear（）

全连接层

> 在二维图像处理的任务中输入与输出一般都设置为二维张量[batch_size, size]，不同于卷积层要求输入输出是四维张量

输入为`[batch_size, in_features]`的张量变换成了`[batch_size, out_features]`的输出张量

nn.ReLU(inplace=True)

> 参数inplace=True:inplace为True，将会改变输入的数据 ，否则不会改变原输入，只会产生新的输出
> 会对原变量覆盖，只要不带来错误就用

```
summary(net, (3, 128, 128))# 查看网络信息，给一个输入格式
```

with torch.no_grad的作用

当requires_grad设置为False时,反向传播时就不会自动求导了，因此大大节约了显存或者说内存
若一个节点（叶子变量：自己创建的tensor）requires_grad被设置为True，那么所有依赖它的节点requires_grad都为True
在该模块下，所有计算得出的tensor的requires_grad都自动设置为False。

torch.nn.Module.parameters：

> 计算模型的参数，返回一个关于模型参数的迭代器

```
for param in model.parameters():
    print("参数类型: ",type(param),"参数大小: ",param.size())
print("模型参数: ",list((model.parameters())))

```

BECLoss

如果我们有一个无穷的损失值，我们在计算梯度的时候也会是一个无穷.会导致BECLoss的反向传播方法非线性。

限制log函数的输出大于等于-100，这样的话就可以得到一个有限的损失值

loss.item（）取出张量具体位置的元素元素值

```
a.isinf().long().sum().item()#计算inf个数
```

nan&inf问题

1、先进行数据清洗，去除nan和inf

```
a = a[a.isnan()] = 0.
a = a[a.isinf()] = 1.
```

2、在计算向量a的式子前进行截断：

```
a = torch.clamp(a, min=1e-8, max=80)
```

其他办法：更改激活函数

- 将softmax修改为：

$$\frac{e^{a_i-M}}{\Sigma e^{a_i-M}} \tag{1}$$
$$M = max\{a_1, a_2 \ldots\ldots a_N\}$$

- 将sigmoid修改为：

```
def sigmoid(inx):
    if inx >= 0:  # 对sigmoid函数的优化，避免了出现极大的数据溢出
        return 1.0 / (1 + np.exp(-inx))
    else:
        return np.exp(inx) / (1 + np.exp(inx))
```

Log-Sum-Exp

- 绝对不会出现上溢(overflow)，即使其余的下溢(underflow)，也可以得到一个合理的值

采取的技巧是：

$$log \sum_{n=1}^{N} exp\{x_n\} = a + log \sum_{n=1}^{N} exp\{x_n - a\}$$

a为任意数.

上式意味着，我们可以任意的平移指数的中心值， 放大或缩小.

一种典型的是使，

$$a = \{max\}_n x_n$$