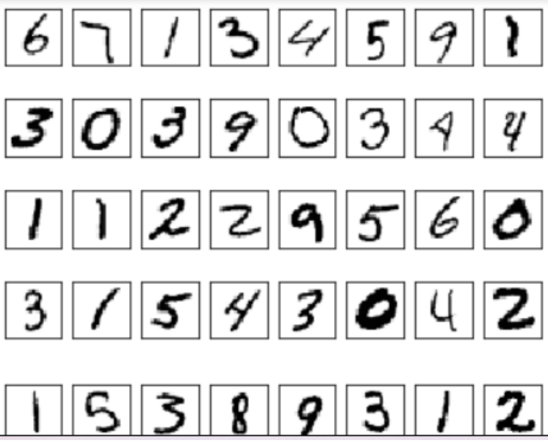
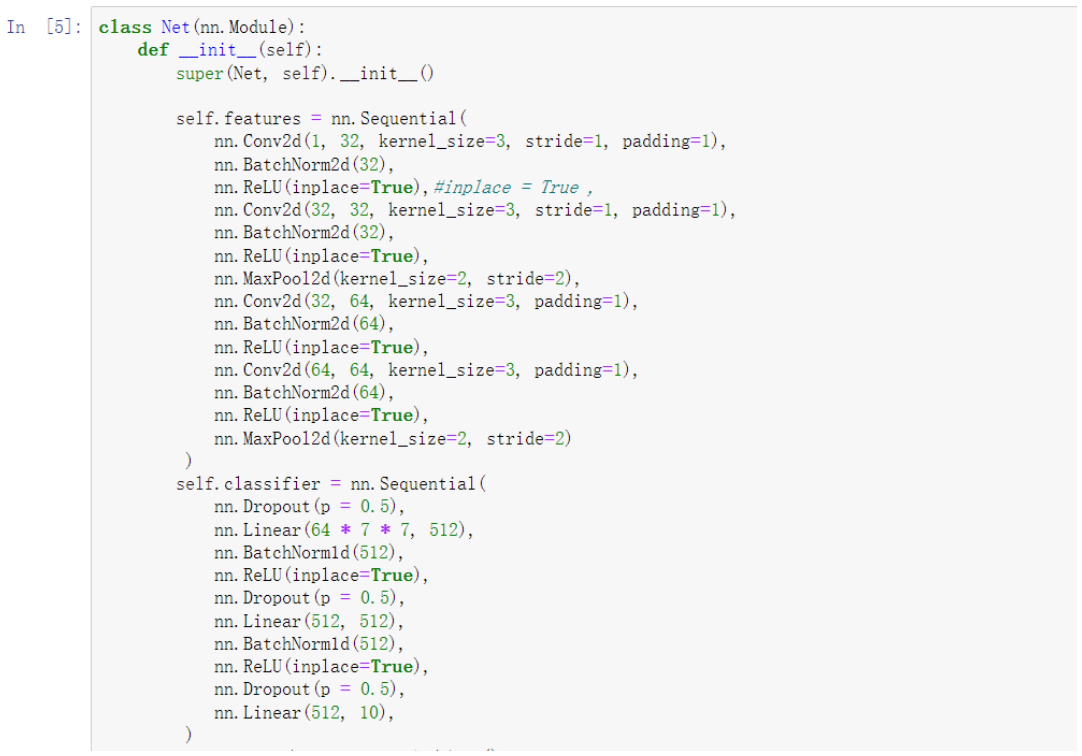
```
In  [1]:  import torch
          import torchvision
          from torch.utils.data import DataLoader
          from torchvision.datasets import MNIST
          from torchvision import transforms
          from torch import nn
          import math
          import os
          os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

```
In  [2]:  # batch_size 超参，根据硬件配置相应大小
          batch_size = 32 #批处理大小
          trans_img = transforms.Compose([
           transforms.ToTensor(),
           transforms.Normalize((0.1307,), (0.3081,))
           ])
```

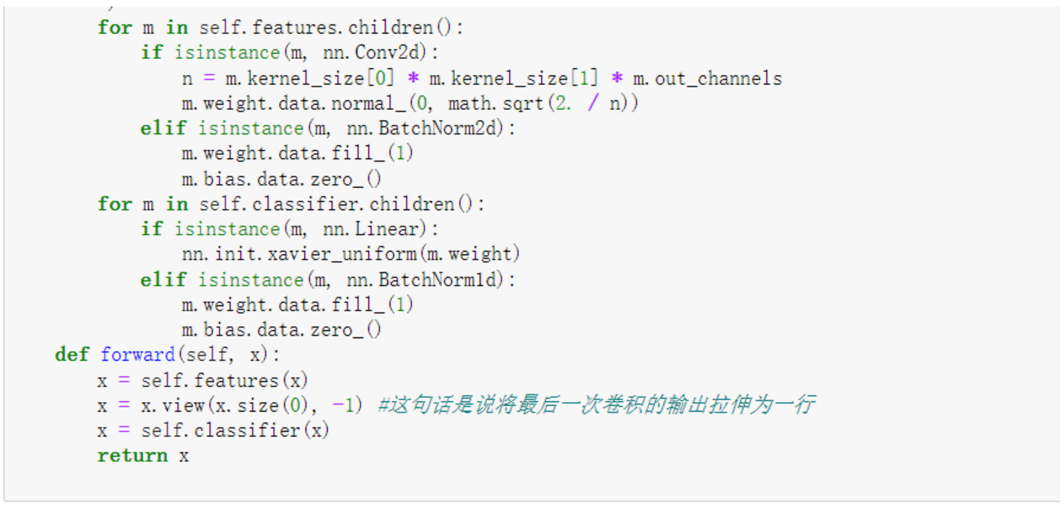
```
In  [3]:  # MNIST 数据集每张图片是灰度图片，大小为 28x28
          trainset = MNIST('data', train=True, download=True, transform=trans_img)
          testset = MNIST('data', train=False, download=True, transform=trans_img)
          train_loader = DataLoader(trainset, batch_size=batch_size,
           shuffle=True, num_workers=1)
          test_loader = DataLoader(testset, batch_size=batch_size,
           shuffle=True, num_workers=1)# 运行环境
```

```
In  [4]:  import matplotlib.pyplot as plt
          plt.ion()
          cnt = 0
          for (img_batch, label) in train_loader:
              cnt += 1
              if cnt > 10:
                  break
              fig, ax = plt.subplots(
                  nrows=4,
                  ncols=8,
                  sharex=True,
                  sharey=True, )
              ax = ax.flatten()
              for i in range(32):
                  img = img_batch[i].numpy().reshape(28, 28)
                  ax[i].imshow(img, cmap='Greys', interpolation='nearest')
              ax[0].set_xticks([])
              ax[0].set_yticks([])
              plt.show()
              plt.close()
          plt.ioff()
```





```
In  [5]: class Net(nn.Module):
             def __init__(self):
                 super(Net, self).__init__()

                 self.features = nn.Sequential(
                     nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
                     nn.BatchNorm2d(32),
                     nn.ReLU(inplace=True), #inplace = True ,
                     nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
                     nn.BatchNorm2d(32),
                     nn.ReLU(inplace=True),
                     nn.MaxPool2d(kernel_size=2, stride=2),
                     nn.Conv2d(32, 64, kernel_size=3, padding=1),
                     nn.BatchNorm2d(64),
                     nn.ReLU(inplace=True),
                     nn.Conv2d(64, 64, kernel_size=3, padding=1),
                     nn.BatchNorm2d(64),
                     nn.ReLU(inplace=True),
                     nn.MaxPool2d(kernel_size=2, stride=2)
                 )
                 self.classifier = nn.Sequential(
                     nn.Dropout(p = 0.5),
                     nn.Linear(64 * 7 * 7, 512),
                     nn.BatchNorm1d(512),
                     nn.ReLU(inplace=True),
                     nn.Dropout(p = 0.5),
                     nn.Linear(512, 512),
                     nn.BatchNorm1d(512),
                     nn.ReLU(inplace=True),
                     nn.Dropout(p = 0.5),
                     nn.Linear(512, 10),
                 )
```



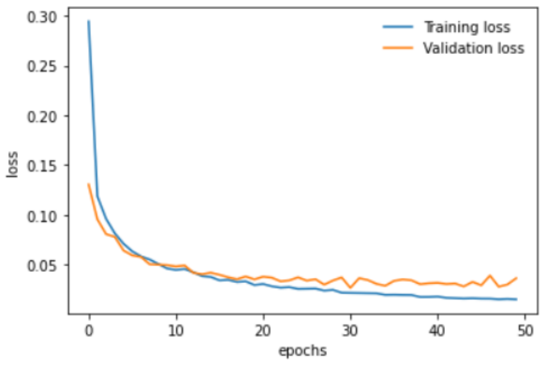
```
        for m in self.features.children():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
        for m in self.classifier.children():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform(m.weight)
            elif isinstance(m, nn.BatchNorm1d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1) #这句话是说将最后一次卷积的输出拉伸为一行
        x = self.classifier(x)
        return x
```

```
In  [6]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
         from torch import optim #优化器
         from torch.autograd import Variable
         import warnings
         warnings.filterwarnings("ignore",category=UserWarning)
         model = Net().to(device) #GUP
         learning_rate = 0.001
         criterion = nn.CrossEntropyLoss(size_average=False)
         optimizer = optim.SGD(model.parameters(), lr = learning_rate)
         # 总的训练轮数
         epochs = 50
         train_losses=[]
         test_losses=[]
         for epoch in range(epochs):
             running_loss, running_acc = 0., 0.
             for (img, label) in train_loader:
                 img = Variable(img).to(device)
                 label = Variable(label).to(device)

                 optimizer.zero_grad() #梯度归零;

                 output = model(img) ##前向传播
                 loss = criterion(output, label) #计算 loss

                 loss.backward() #反向传播, 计算当前梯度;
                 optimizer.step() #反向传播, 计算当前梯度;
                 running_loss += loss.item() #item () 取出张量具体位置的元素元素值
                 _, predict = torch.max(output, 1)
          #_,predicted = torch.max(outputs.data,dim): 返回最大值所在索引,  dim=1 时, 按行返回最大值所在索引
                 correct_num = (predict == label).sum()
                 running_acc += correct_num.item()
```

```
    running_loss /= len(trainset)
    running_acc /= len(trainset)

    with torch.no_grad():#不求梯度
        test_loss, test_acc = 0., 0.
        for images, labels in test_loader:
            images = Variable(images).to(device)
            labels = Variable(labels).to(device)
            output = model(images)
            loss = criterion(output, labels)
            test_loss += loss.item()
            _, predict = torch.max(output, 1)
            correct_num = (predict == labels).sum()
            test_acc += correct_num.item()

    test_loss /=len(testset)
    test_acc /=len(testset)

    train_losses.append(running_loss)
    test_losses.append(test_loss)
    print("Epoch: {}/{}.. ".format(epoch+1, epochs),
          "Training Loss: {:.3f}.. ".format(train_losses[-1]),
          "Training Accuracy: {:.3f} %".format(100*running_acc),
          "Test Loss: {:.3f}.. ".format(test_losses[-1]),
          "Test Accuracy: {:.3f} %".format(100*test_acc))
# 保存模型
torch.save(model, 'conv.pth.tar')
```

```
Epoch: 1/50..  Training Loss: 0.294..  Training Accuracy: 90.587 % Test Loss: 0.131..  Test Accuracy: 95.920 %
Epoch: 2/50..  Training Loss: 0.119..  Training Accuracy: 96.377 % Test Loss: 0.096..  Test Accuracy: 96.920 %
Epoch: 3/50..  Training Loss: 0.096..  Training Accuracy: 97.008 % Test Loss: 0.081..  Test Accuracy: 97.530 %
Epoch: 4/50..  Training Loss: 0.081..  Training Accuracy: 97.527 % Test Loss: 0.078..  Test Accuracy: 97.540 %
Epoch: 5/50..  Training Loss: 0.071..  Training Accuracy: 97.815 % Test Loss: 0.064..  Test Accuracy: 97.950 %
Epoch: 6/50..  Training Loss: 0.063..  Training Accuracy: 98.033 % Test Loss: 0.059..  Test Accuracy: 98.080 %
Epoch: 7/50..  Training Loss: 0.058..  Training Accuracy: 98.212 % Test Loss: 0.058..  Test Accuracy: 98.200 %
Epoch: 8/50..  Training Loss: 0.055..  Training Accuracy: 98.318 % Test Loss: 0.050..  Test Accuracy: 98.440 %
Epoch: 9/50..  Training Loss: 0.051..  Training Accuracy: 98.450 % Test Loss: 0.050..  Test Accuracy: 98.280 %
Epoch: 10/50..  Training Loss: 0.046..  Training Accuracy: 98.548 % Test Loss: 0.049..  Test Accuracy: 98.460 %
Epoch: 11/50..  Training Loss: 0.045..  Training Accuracy: 98.655 % Test Loss: 0.048..  Test Accuracy: 98.440 %
Epoch: 12/50..  Training Loss: 0.046..  Training Accuracy: 98.582 % Test Loss: 0.049..  Test Accuracy: 98.590 %
Epoch: 13/50..  Training Loss: 0.042..  Training Accuracy: 98.687 % Test Loss: 0.042..  Test Accuracy: 98.710 %
Epoch: 14/50..  Training Loss: 0.038..  Training Accuracy: 98.802 % Test Loss: 0.040..  Test Accuracy: 98.790 %
Epoch: 15/50..  Training Loss: 0.038..  Training Accuracy: 98.823 % Test Loss: 0.042..  Test Accuracy: 98.750 %
Epoch: 16/50..  Training Loss: 0.034..  Training Accuracy: 98.923 % Test Loss: 0.040..  Test Accuracy: 98.860 %
Epoch: 17/50..  Training Loss: 0.035..  Training Accuracy: 98.962 % Test Loss: 0.037..  Test Accuracy: 98.770 %
Epoch: 18/50..  Training Loss: 0.033..  Training Accuracy: 99.000 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 19/50..  Training Loss: 0.033..  Training Accuracy: 98.953 % Test Loss: 0.038..  Test Accuracy: 98.840 %
Epoch: 20/50..  Training Loss: 0.029..  Training Accuracy: 99.068 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 21/50..  Training Loss: 0.031..  Training Accuracy: 99.077 % Test Loss: 0.038..  Test Accuracy: 98.800 %
Epoch: 22/50..  Training Loss: 0.028..  Training Accuracy: 99.103 % Test Loss: 0.037..  Test Accuracy: 98.900 %
Epoch: 23/50..  Training Loss: 0.027..  Training Accuracy: 99.168 % Test Loss: 0.033..  Test Accuracy: 98.980 %
Epoch: 24/50..  Training Loss: 0.027..  Training Accuracy: 99.152 % Test Loss: 0.034..  Test Accuracy: 98.970 %
Epoch: 25/50..  Training Loss: 0.026..  Training Accuracy: 99.187 % Test Loss: 0.037..  Test Accuracy: 98.850 %
Epoch: 26/50..  Training Loss: 0.026..  Training Accuracy: 99.210 % Test Loss: 0.034..  Test Accuracy: 99.010 %
Epoch: 27/50..  Training Loss: 0.026..  Training Accuracy: 99.157 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 28/50..  Training Loss: 0.024..  Training Accuracy: 99.250 % Test Loss: 0.030..  Test Accuracy: 99.040 %
Epoch: 29/50..  Training Loss: 0.025..  Training Accuracy: 99.218 % Test Loss: 0.034..  Test Accuracy: 98.960 %
Epoch: 30/50..  Training Loss: 0.022..  Training Accuracy: 99.315 % Test Loss: 0.037..  Test Accuracy: 98.840 %
Epoch: 31/50..  Training Loss: 0.022..  Training Accuracy: 99.253 % Test Loss: 0.027..  Test Accuracy: 99.120 %
```

```
In  [7]: plt.plot(train_losses, label='Training loss')
         plt.plot(test_losses, label='Validation loss')
         plt.xlabel("epochs")
         plt.ylabel("loss")
         plt.legend(frameon=False)

Out[7]: <matplotlib.legend.Legend at 0x23ae6452c70>
```



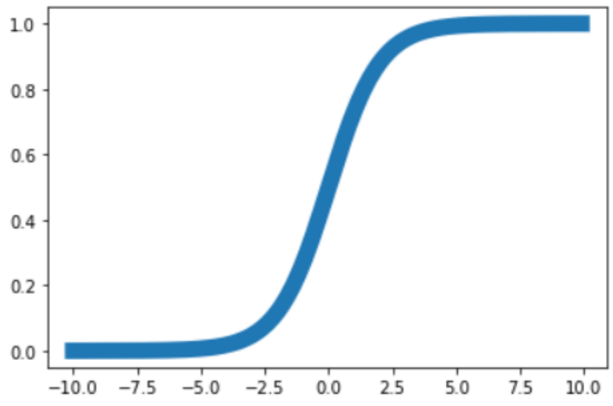
```
In  [9]: device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
         model = torch.load('conv.pth.tar')
         print('testing cnn model')
         testloss, testacc = 0., 0.
         for (img, label) in test_loader:
             img = Variable(img).to(device)
             label = Variable(label).to(device)
             out = model(img)
             loss = criterion(out, label)
             testloss += loss.item()
             _, predict = torch.max(out, 1)
             correct_num = (predict == label).sum()
             testacc += correct_num.item()
         testloss /= len(testset)
         testacc /= len(testset)
         print('cnn model, Test: Loss: %.5f, Acc: %.2f' %
               (testloss, 100 * testacc))

         testing cnn model
         cnn model, Test: Loss: 0.03346, Acc: 99.09
```

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        import math
        #显示中文
        plt.rcParams['font.sans-serif'] = ['Arial Unicode MS']
```

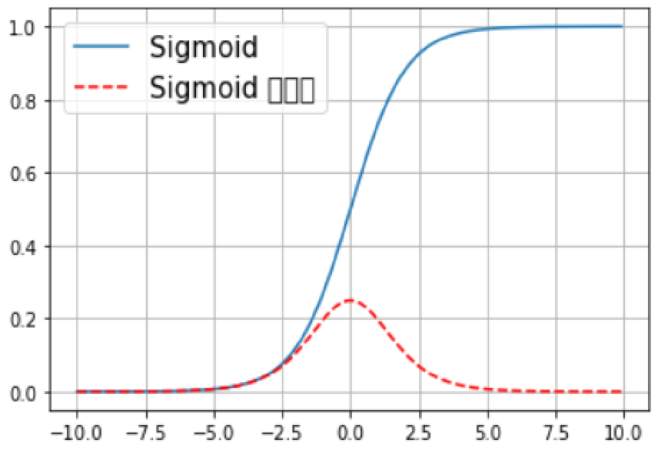
```
In [3]: def sigmoid(x):
            return 1/(1+np.exp(-x))
```

```
In [4]: def df_sigmoid(x):
            s = sigmoid(x)
            return np. multiply(s , (np.ones(len(x)) - s) )
```

```
In [5]: x = np.arange(-10., 10., 0.1)
        y_sigmoid = sigmoid(x)
        #df = np. multiply(y_sigmoid , (np.ones(len(x)) - y_sigmoid) )
        df = df_sigmoid(x)
        plt.plot(x, y_sigmoid, label = u'Sigmoid', linewidth=10)
        plt.show()
```



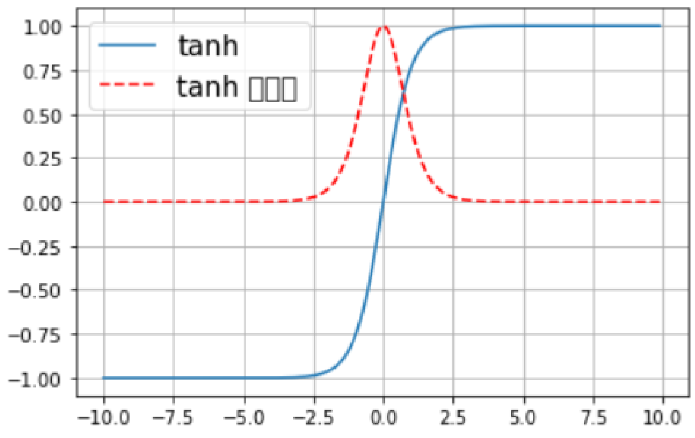
```
In [6]: x = np.arange(-10., 10., 0.1)
        y_sigmoid = sigmoid(x)
        #df = np. multiply(y_sigmoid , (np.ones(len(x)) - y_sigmoid) )
        df = df_sigmoid(x)
        plt.plot(x, y_sigmoid, label = u'Sigmoid')
        plt.plot(x, df, 'r', linestyle = '--', label = u'Sigmoid 的导数')
        plt.grid()#生成网格线
        plt.legend(fontsize = 15)
```



```
In  [7]: def tanh(x):
             return np.tanh(x)
         #return (1-np.exp(-2x))/(1+np.exp(-2x))
```

```
In  [8]: def df_tanh(x):
             t = tanh(x)
             return 1- np.power(t, 2)#np.power()幂次方
```

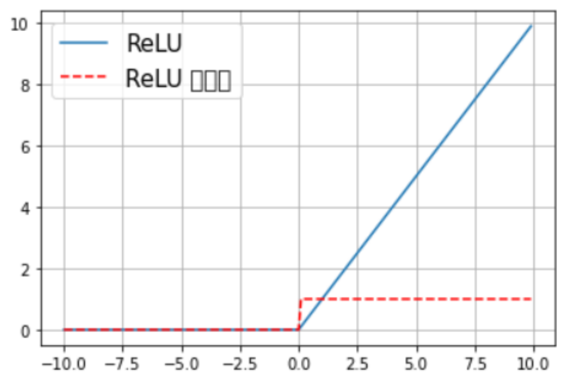
```
In  [9]: x = np.arange(-10., 10., 0.1)
         y_tanh = tanh(x)
         df = df_tanh(x)
         plt.plot(x, y_tanh, label = u'tanh')
         plt.plot(x, df, 'r', linestyle = '--', label = u'tanh 的导数')
         plt.grid()
         plt.legend(fontsize = 15)

Out[9]: <matplotlib.legend.Legend at 0x1678f02cfd0>
```



```
In  [10]: def ReLU(x):
              arr = []
              for i in x:
                  arr.append(0 if i<0 else i )
              return arr
```

```
In  [11]: def df_ReLU(x):
              arr = []
              for i in x:
                  arr.append(0 if i<0 else 1 )
              return arr
```

```
In  [12]: x = np.arange(-10., 10., 0.1)
          y_ReLU = ReLU(x)
          df = df_ReLU(x)
          plt.plot(x, y_ReLU, label = u'ReLU')
          plt.plot(x, df, 'r', linestyle = '--', label = u'ReLU 的导数')
          plt.grid()
          plt.legend(fontsize = 15)

Out[12]: <matplotlib.legend.Legend at 0x1678eefedc0>
```

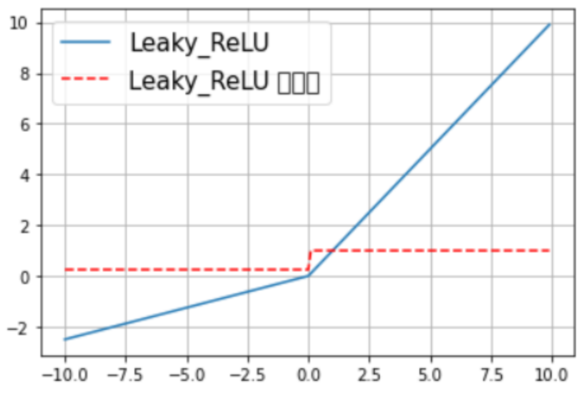


```
In  [13]: def Leaky_ReLU(x,alpha):
              arr = []
              for i in x:
                  arr.append(alpha*i if i<0 else i )
              return arr
```

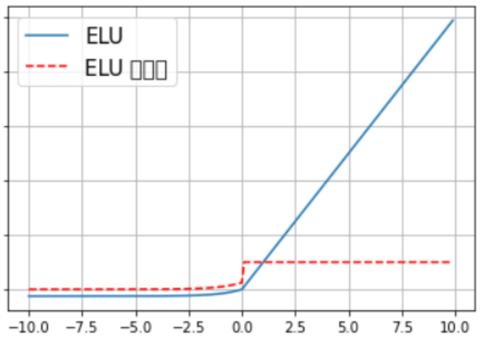
```
In  [14]: def df_Leaky_ReLU(x,alpha):
              arr = []
              for i in x:
                  arr.append(alpha if i<0 else 1 )
              return arr
```

```
In  [15]: x = np.arange(-10., 10., 0.1)
          alpha =0.25
          y_LeakyReLU = Leaky_ReLU(x, alpha)
          df = df_Leaky_ReLU(x, alpha)
          plt.plot(x, y_LeakyReLU, label = u'Leaky_ReLU')
          plt.plot(x, df, 'r', linestyle = '--', label = u'Leaky_ReLU 的导数')
          plt.grid()
          plt.legend(fontsize = 15)
```

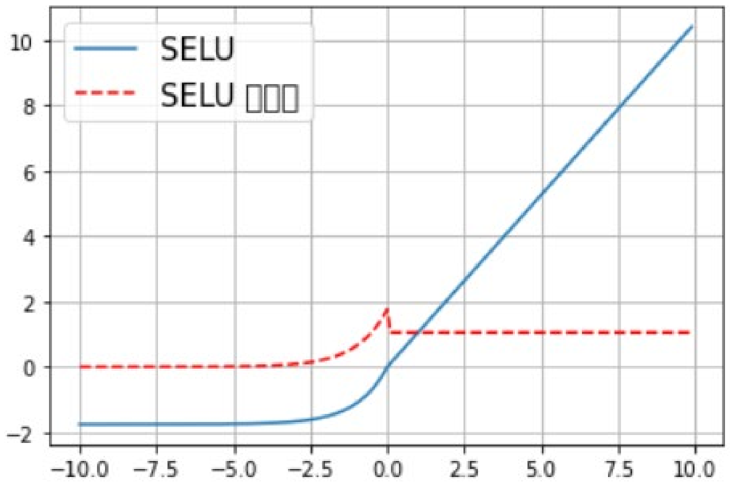
```
Out[15]: <matplotlib.legend.Legend at 0x1678f06a820>
```



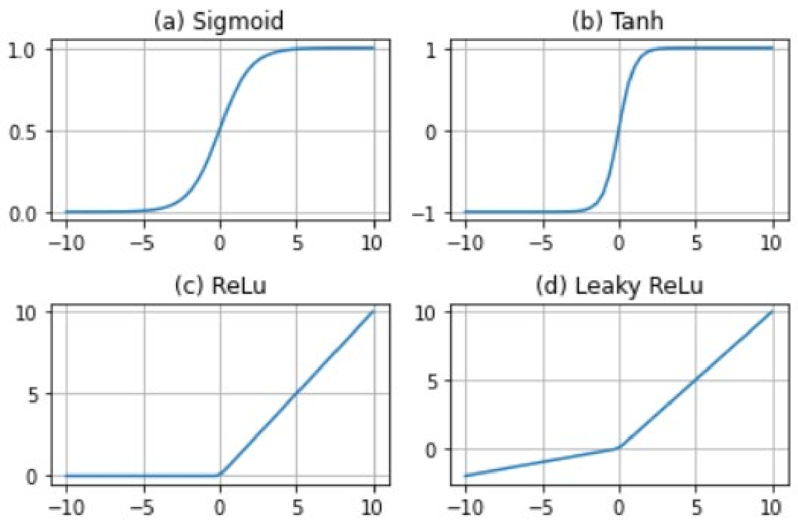
```
In  [16]: def ELU(x, alpha):
              arr = []
              for i in x:
                  arr.append(alpha*(np.exp(i) -1) if i<0 else i )
              return arr
          def df_ELU(x, alpha):
              arr = []
              for i in x:
                  arr.append(alpha*(np.exp(i)) if i<0 else 1 )
              return arr
```

In [17]:
```
x = np.arange(-10., 10., 0.1)
alpha =0.25
y_ELU = ELU(x, alpha)
df = df_ELU(x, alpha)
plt.plot(x, y_ELU, label = u'ELU')
plt.plot(x, df, 'r', linestyle = '--', label = u'ELU 的导数')
plt.grid()
plt.legend(fontsize = 15)
```
Out[17]:
```
<matplotlib.legend.Legend at 0x1678f08bdc0>
```


In [19]:
```
def SELU(x, alpha):
    arr = []
    alpha = 1.6732632423543772848170429916717
    lambda_ = 1.0507009873554804934193349852946
    for i in x:
        arr.append(lambda_*alpha*(np.exp(i) -1) if i<0 else lambda_*i )
    return arr
```
In [20]:
```
def df_SELU(x, alpha):
    arr = []
    alpha = 1.6732632423543772848170429916717
    lambda_ = 1.0507009873554804934193349852946
    for i in x:
        arr.append(lambda_*alpha*(np.exp(i) ) if i<0 else lambda_ )
    return arr
```
In [21]:
```
x = np.arange(-10., 10., 0.1)
alpha =0.25
y_SELU = SELU(x, alpha)
df = df_SELU(x, alpha)
plt.plot(x, y_SELU, label = u'SELU')
plt.plot(x, df, 'r', linestyle = '--', label = u'SELU 的导数')
plt.grid()
plt.legend(fontsize = 15)
```
Out[21]:
```
<matplotlib.legend.Legend at 0x1678f05b2b0>
```



```
In  [22]: import matplotlib.pyplot as plt
          import numpy as np
          x = np.linspace(-10, 10)
          y_sigmoid = 1/(1+np.exp(-x))
          y_tanh = (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
          fig = plt.figure()
          # plot sigmoid
          ax = fig.add_subplot(221)
          ax.plot(x, y_sigmoid)
          ax.grid()
          ax.set_title('(a) Sigmoid')
          # plot tanh
          ax = fig.add_subplot(222)
          ax.plot(x, y_tanh)
          ax.grid()
          ax.set_title('(b) Tanh')
          # plot relu
          ax = fig.add_subplot(223)
          y_relu = np.array([0*item if item<0 else item for item in x ])
          ax.plot(x, y_relu)
          ax.grid()
          ax.set_title('(c) ReLu')
          #plot leaky relu
          ax = fig.add_subplot(224)
          y_relu = np.array([0.2*item if item<0 else item for item in x ])
          ax.plot(x, y_relu)
          ax.grid()
          ax.set_title('(d) Leaky ReLu')
          plt.tight_layout()
```

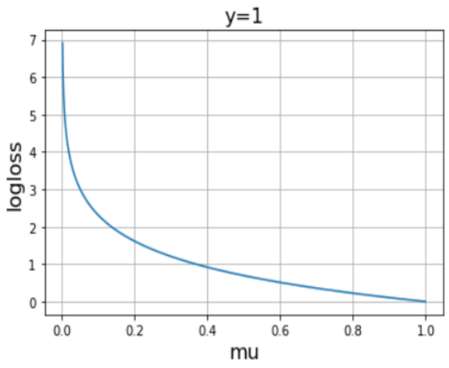


In [27]:
```
import matplotlib.pyplot as plt # python 的可视化模块, 我有教程 (https://morvanzhou.github.io/tutorials/data-manipulation/p
import torch
import torch.nn.functional as F # 激励函数都在这
from torch.autograd import Variable#自动求导 变量
# 做一些假数据来观看图像
x = torch.linspace(-5, 5, 200) # x data (tensor), shape=(100, 1)
x = Variable(x)
x_np = x.data.numpy() # 换成 numpy array, 出图时用
# 几种常用的 激励函数
y_relu = F.relu(x).data.numpy()
y_sigmoid = F.sigmoid(x).data.numpy()
y_tanh = F.tanh(x).data.numpy()
y_softplus = F.softplus(x).data.numpy() #Softplus(x)=log(1+e x)
plt.figure(1, figsize=(8, 6))
plt.subplot(221)
plt.plot(x_np, y_relu, c='red', label='relu')
plt.ylim((-1, 5))
plt.legend(loc='best')#图例所有 figure 位置
plt.subplot(222)
plt.plot(x_np, y_sigmoid, c='red', label='sigmoid')
plt.ylim((-0.2, 1.2))
plt.legend(loc='best')
plt.subplot(223)
plt.plot(x_np, y_tanh, c='red', label='tanh')
plt.ylim((-1.2, 1.2))
plt.legend(loc='best')
plt.subplot(224)
plt.plot(x_np, y_softplus, c='red', label='softplus')
plt.ylim((-0.2, 6))
plt.legend(loc='best')
plt.show()
```

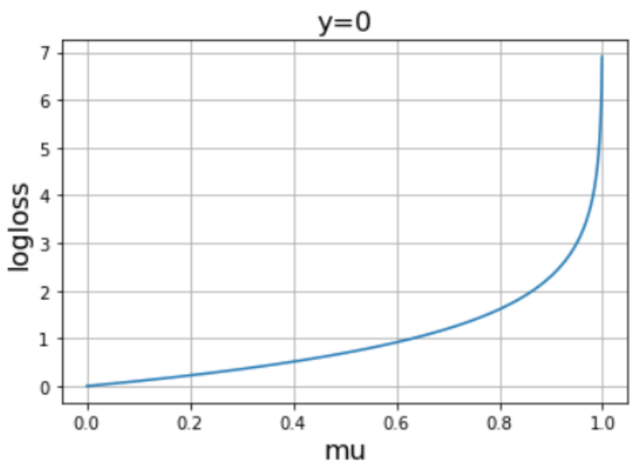
In [28]:
```
x = np.arange(0, 1, 0.001)
logloss = -np.log( x)
plt.plot(x, logloss)
plt.grid()
plt.xlabel('mu', fontsize = 16)
plt.ylabel('logloss', fontsize = 16)
plt.title('y=1', fontsize = 16)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_1980\51743719.py:2: RuntimeWarning: divide by zero encountered in log
  logloss = -np.log( x)
```

Out[28]:
```
Text(0.5, 1.0, 'y=1')
```



```
In  [29]: x = np.arange(0, 1, 0.001)
          logloss = -np.log(1- x)
          plt.plot(x, logloss)
          plt.grid()
          plt.xlabel('mu', fontsize = 16)
          plt.ylabel('logloss', fontsize = 16)
          plt.title('y=0', fontsize = 16)
Out[29]: Text(0.5, 1.0, 'y=0')
```


y=0
logloss
mu