Machine Learning A (2023)
Home Assignment 6

Alexander Husted | wqg382

Contents

1 Logistic regression in PyTorch

Model definition

```
class LogisticRegressionPytorch(nn.Module):
    def __init__(self, d, m): #dimensons, features
        super(LogisticRegressionPytorch, self).__init__()
        self.linear = nn.Linear(d,m) # Applies a linear transformation to
    the incoming data: :math:`y = xA^T + b`
    def forward(self, x):
        return self.linear(x)

logreg_pytorch = LogisticRegressionPytorch(d, m)
```

Define loss-function

```
#computes the cross entropy loss between input logits and target.
criterion = nn.CrossEntropyLoss()
```

Training

```
no_epochs = 10000  # Number of training steps
X_train_T = torch.Tensor(X_train)  # Automatically casts to foat
y_train_T = torch.from_numpy(y_train).type(torch.LongTensor)  # Does not
   cast to float
for epoch in range(no_epochs):  # Loop over the dataset multiple times
    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward + backward + optimize
    outputs = logreg_pytorch(X_train_T)
    loss = criterion(outputs, y_train_T)
    loss.backward()
    optimizer.step()
```

Convolution vs. cross-correlation

The difference between convolution and cross-correlation is that convolution flips signals, and cross-correlation does not. Convolution is good for detectiong patterns, because the convolution of two signals is a measure of how well one signal matches the other when it is flipped and shifted. In contrast to fx. a picture of a dog, when recognizing trafic-sign the direction is not insignificant, which is why we use cross-correlation.

2 Convolutional Neural Networks

2.1 Sobel filter

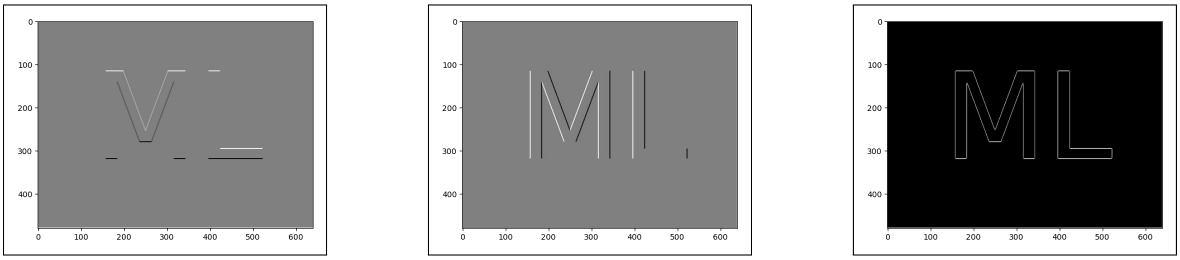

Figure 1: gy, gx, image

```
#Read image
image = read_image("ML.jpg", mode=ImageReadMode.GRAY).type(torch.
    FloatTensor)
image = image.unsqueeze_(0)  # Add a dimension

#Create the two matrices
sobel_x = torch.tensor([[1., 0., -1.], [2., 0., -2.], [1., 0., -1.]]  ).
    expand(1,1,3,3)
sobel_y = torch.tensor([[1., 2., 1.],  [0., 0., 0.],  [-1., -2., -1.]]).
    expand(1,1,3,3)
sobel = torch.cat((sobel_x, sobel_y), dim=0)

'''
    Input: an image and a filter
    Output: the image when filter is applied
'''
def sobelFilter(img, filter):
    conv = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=3,
    padding=1, bias=False) # 2D convolution layer
    conv.weight = torch.nn.Parameter(filter, requires_grad=False) #Apply
    filter
    c = conv(img)

    # c now conatians two images (G_x, G_y) as seen by its shape:
    #[1, -->2, 480, 640], we want to split into two feature maps.
    G_x = torch.squeeze(c[:,0,:,:].expand(1,1,480,640))
    G_y = torch.squeeze(c[:,1,:,:].expand(1,1,480,640))

    #Compute G by the formular given in the assignment
    G = torch.square(torch.square(G_x) + torch.square(G_y))
    #Remove unnecessary dimmensions, from [1,1,480,640] --> [480,640]
    return G, G_x, G_y
newImage, gx, gy = sobelFilter(image, sobel)
```

2.2 Convolutional neural networks

```
class Net(nn.Module):
    def __init__(self, img_size=28):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d (3 , 64 , kernel_size =(5 , 5) , stride =
  (1 , 1) )
        self.pool1 = nn.MaxPool2d ( kernel_size =2 , stride =2 , padding
  =0 , dilation=1 , ceil_mode = False )
        self.conv2 = nn.Conv2d (64 , 64 , kernel_size =(5 , 5) , stride =
  (1 , 1) )
        self.pool2 = nn.MaxPool2d ( kernel_size =2 , stride =2 , padding
  =0 , dilation=1 , ceil_mode = False )
        self.fc2   = nn.Linear ( in_features =1024 , out_features =43 ,
  bias = True )

    def forward(self, x):
        x = self.conv1(F.relu(self.conv1))  # Aplly convulution using the
   relu function
        x = self.conv2(F.relu(self.conv2))
        x = x.view(x.size(0), -1) #Change the shape of x, while keeping
  the data
        x = self.fc2(x)
        return x
```

2.3 Augmentation

Which transformations are applied to the input images during training?
RandomAffine - Scales and rotates images
RandomCrop - crops an image at a random location.
ColorJitter - The ColorJitter transform randomly changes the brightness, contrast, saturation, hue, and other properties of an image.

Why is a transformation conditioned on the label?
We only want to flip images horizontally when they are horizontally symetric. The reason for this is the same as in qustion 1 (Convolution vs. cross-correlation). But for horizontally symetric images, the signs don't change meaning when fliped.

Why is a transformation conditioned on the label?
Just as we flip horizontally symetric images on the horizontally axis, we can flip vertically symetric images on the vertical axis.

```
if label in [5, 6, 12, 15, 17, 32, ]:
          image = transforms.RandomVerticalFlip(p=0.5)(image)
```