RTML-Midterm-2020

March 12, 2021

1 RTML Midterm 2021

1. In Lab 06, you fine tuned a Mask R-CNN model on the Cityscapes dataset. Download the image at http://www.cs.ait.ac.th/~mdailey/20201112_072342.jpg and run it through you model. Provide your source code to load the model, image, get the result, and display the result here. Display the resulting bounding boxes and masks.

[1]:
```
import torch
import torchvision
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor
from torchvision.datasets import CocoDetection

import utils
from coco_utils import get_city
import transforms
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# Load a model pre-trained on COCO and put it in inference mode
path = "../lab6/city_weights/mask-rcnn-09-epochs.pth"
print('Loading pretrained model...')
model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=False)

model.load_state_dict(torch.load(path, map_location=device))
model.eval()
```

```
Loading pretrained model…
```

[1]:
```
MaskRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=1e-05)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
```

```
ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=1e-05)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=1e-05)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=1e-05)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256, eps=1e-05)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=1e-05)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=1e-05)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=1e-05)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=1e-05)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=1e-05)
```

```
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=1e-05)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=1e-05)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(512, eps=1e-05)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=1e-05)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=1e-05)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=1e-05)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=1e-05)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=1e-05)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=1e-05)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=1e-05)
          (relu): ReLU(inplace=True)
```

```
        )
      )
      (layer3): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): FrozenBatchNorm2d(1024, eps=1e-05)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=1e-05)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=1e-05)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=1e-05)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=1e-05)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=1e-05)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
```

```
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): FrozenBatchNorm2d(2048, eps=1e-05)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=1e-05)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=1e-05)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=1e-05)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=1e-05)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=1e-05)
          (relu): ReLU(inplace=True)
        )
      )
    )
    (fpn): FeaturePyramidNetwork(
      (inner_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
        (2): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
        (3): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
      )
      (layer_blocks): ModuleList(
        (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      )
      (extra_blocks): LastLevelMaxPool()
    )
  )
```

```
  (rpn): RegionProposalNetwork(
    (anchor_generator): AnchorGenerator()
    (head): RPNHead(
      (conv): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (roi_heads): RoIHeads(
    (box_roi_pool): MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'],
output_size=(7, 7), sampling_ratio=2)
    (box_head): TwoMLPHead(
      (fc6): Linear(in_features=12544, out_features=1024, bias=True)
      (fc7): Linear(in_features=1024, out_features=1024, bias=True)
    )
    (box_predictor): FastRCNNPredictor(
      (cls_score): Linear(in_features=1024, out_features=91, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=364, bias=True)
    )
    (mask_roi_pool): MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'],
output_size=(14, 14), sampling_ratio=2)
    (mask_head): MaskRCNNHeads(
      (mask_fcn1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (relu1): ReLU(inplace=True)
      (mask_fcn2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (relu2): ReLU(inplace=True)
      (mask_fcn3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (relu3): ReLU(inplace=True)
      (mask_fcn4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (relu4): ReLU(inplace=True)
    )
    (mask_predictor): MaskRCNNPredictor(
      (conv5_mask): ConvTranspose2d(256, 256, kernel_size=(2, 2), stride=(2, 2))
      (relu): ReLU(inplace=True)
      (mask_fcn_logits): Conv2d(256, 91, kernel_size=(1, 1), stride=(1, 1))
    )
  )
)
```

[2]:
```
#load image
import torch
from torchvision import datasets, transforms
```

```
import numpy as np
import matplotlib.pyplot as plt
preprocess = transforms.Compose([
    transforms.ToTensor()
])

img_path = "./image"
img_folder = './image'
ann_file = '/root/labs/Cityscapes/annotations/instancesonly_filtered_gtFine_val.
 ↪json'

#dataset = CocoDetection(img_folder, ann_file, transforms=preprocess)
import helper

dataset = datasets.ImageFolder(img_path, transform=preprocess)


val_dataloader = torch.utils.data.DataLoader(dataset, batch_size=1,␣
 ↪shuffle=False, num_workers=0)
images, labels = next(iter(val_dataloader))
```

[3]:
```
model.to(device)
images, targets = next(iter(val_dataloader))
# import numpy as np
images = [img.to(device) for img in images]
print(len(images))
# images.to(device)
predictions = model(images)
print(predictions)

print('Prediction keys:', list(dict(predictions[0])))
print('Boxes shape:', predictions[0]['boxes'])
print('Labels shape:', predictions[0]['labels'].shape)
print('Scores shape:', predictions[0]['scores'].shape)
print('Masks shape:', predictions[0]['masks'])
```

```
1
[{'boxes': tensor([[3019.8655, 1348.9965, 3169.1460, 1434.2233],
        [ 304.7679,  591.4370, 1859.9025, 1832.5529],
        [ 141.5512,  200.1582, 3921.3518, 1822.2751],
        [ 242.8794,  273.7198, 3453.4316, 1910.3984],
        [ 400.1719,  673.6343, 1909.9603, 1867.7961],
        [3019.1887, 1343.7679, 3167.4346, 1434.8394],
        [2637.9656, 1453.4728, 3009.1890, 1478.8118]], device='cuda:0',
       grad_fn=<StackBackward>), 'labels': tensor([3, 3, 5, 3, 5, 7, 3],
device='cuda:0'), 'scores': tensor([0.3638, 0.1707, 0.1626, 0.1329, 0.1207,
0.0589, 0.0513],
```

```
       device='cuda:0', grad_fn=<IndexBackward>), 'masks': tensor([[[[0., 0.,
0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        …,


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],
```

```
        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]]], device='cuda:0',
       grad_fn=<UnsqueezeBackward0>)}]
Prediction keys: ['boxes', 'labels', 'scores', 'masks']
Boxes shape: tensor([[3019.8655, 1348.9965, 3169.1460, 1434.2233],
        [ 304.7679,  591.4370, 1859.9025, 1832.5529],
        [ 141.5512,  200.1582, 3921.3518, 1822.2751],
        [ 242.8794,  273.7198, 3453.4316, 1910.3984],
        [ 400.1719,  673.6343, 1909.9603, 1867.7961],
        [3019.1887, 1343.7679, 3167.4346, 1434.8394],
        [2637.9656, 1453.4728, 3009.1890, 1478.8118]], device='cuda:0',
       grad_fn=<StackBackward>)
Labels shape: torch.Size([7])
Scores shape: torch.Size([7])
Masks shape: tensor([[[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        [[[0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          …,
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.],
          [0., 0., 0.,  …, 0., 0., 0.]]],


        …,
```

```
         [[[0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           …,
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.]]],


         [[[0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           …,
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.]]],


         [[[0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           …,
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.],
           [0., 0., 0.,  …, 0., 0., 0.]]]], device='cuda:0',
        grad_fn=<UnsqueezeBackward0>)
```

[ ]:

```

```

[4]:

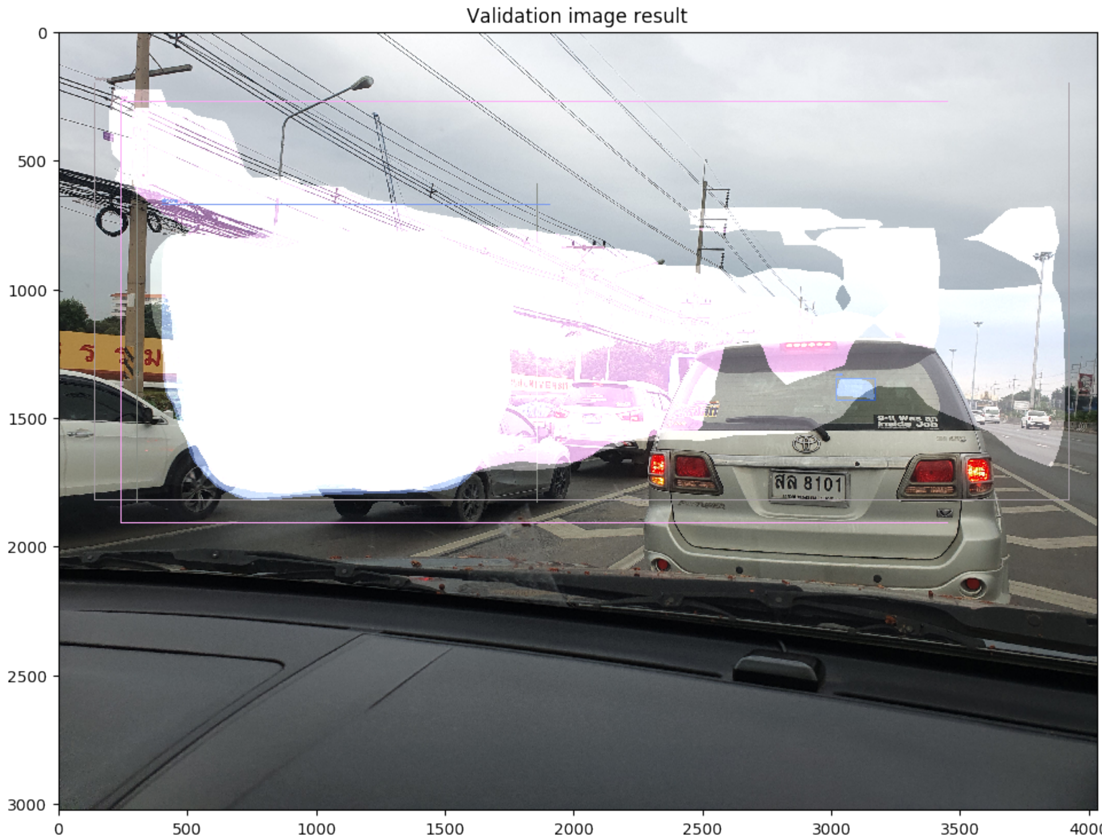
```
import numpy as np
import cv2
import random

# Array of labels for COCO dataset (91 elements)

coco_names = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',␣
 ↪'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop␣
 ↪sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/
 ↪A', 'N/A',
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
```

```python
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',␣
↪'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl',
    'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog',␣
↪'pizza',
    'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining␣
↪table',
    'N/A', 'N/A', 'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote',␣
↪'keyboard', 'cell phone',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book',
    'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush'
]

# Random colors to use for labeling objects

COLORS = np.random.uniform(0, 255, size=(len(coco_names), 3)).astype(np.uint8)

# Overlay masks, bounding boxes, and labels on input numpy image

def draw_segmentation_map(image, masks, boxes, labels):
    alpha = 1
    beta = 0.5 # transparency for the segmentation map
    gamma = 0 # scalar added to each sum
    # convert from RGB to OpenCV BGR format
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    for i in range(len(masks)):
        mask = masks[i,:,:]
        red_map = np.zeros_like(mask).astype(np.uint8)
        green_map = np.zeros_like(mask).astype(np.uint8)
        blue_map = np.zeros_like(mask).astype(np.uint8)
        # apply a randon color mask to each object
        color = COLORS[random.randrange(0, len(COLORS))]
        red_map[mask > 0.5] = color[0]
        green_map[mask > 0.5] = color[1]
        blue_map[mask > 0.5] = color[2]
        # combine all the masks into a single image
        segmentation_map = np.stack([red_map, green_map, blue_map], axis=2)
        # apply colored mask to the image
        image = cv2.addWeighted(image, alpha, segmentation_map, beta, gamma)
        # draw the bounding box around each object
        p1 = (int(boxes[i][0]), int(boxes[i][1]))
        p2 = (int(boxes[i][2]), int(boxes[i][3]))
        color = (int(color[0]), int(color[1]), int(color[2]))
        cv2.rectangle(image, p1, p2, color, 2)
        # put the label text above the objects
        p = (int(boxes[i][0]), int(boxes[i][1]-10))
```

```
            cv2.putText(image, labels[i], p, cv2.FONT_HERSHEY_SIMPLEX, 0.5, color,␣
 ↪2, cv2.LINE_AA)

    return cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Overlay masks, bounding boxes, and labels of objects with scores greater than
# threshold on one of the images in the input tensor using the predictions␣
 ↪output by Mask R-CNN.

def prediction_to_mask_image(images, predictions, img_index, threshold):
    scores = predictions[img_index]['scores']
    print(scores)
    boxes_to_use = scores >= threshold
    img = (images[img_index].cpu().permute(1, 2, 0).numpy() * 255).astype(np.
 ↪uint8)
    img = cv2.flip(img, 0)
    img = cv2.flip(img, 1)
    masks = predictions[img_index]['masks'][boxes_to_use, :, :].cpu().detach().
 ↪squeeze(1).numpy()

    boxes = predictions[img_index]['boxes'][boxes_to_use, :].cpu().detach().
 ↪numpy()
    print(predictions[img_index]['boxes'][boxes_to_use, :])
    labels = predictions[img_index]['labels'][boxes_to_use].cpu().numpy()
    labels = [ coco_names[l] for l in labels ]

    return draw_segmentation_map(img, masks, boxes, labels)
```

[5]:
```
from matplotlib import pyplot as plt

masked_img = prediction_to_mask_image(images, predictions, 0, 0.1)
plt.figure(1, figsize=(12, 9), dpi=100)
plt.imshow(masked_img)
plt.title('Validation image result')
plt.show()
```

```
tensor([0.3638, 0.1707, 0.1626, 0.1329, 0.1207, 0.0589, 0.0513],
       device='cuda:0', grad_fn=<IndexBackward>)
tensor([[3019.8655, 1348.9965, 3169.1460, 1434.2233],
        [ 304.7679,  591.4370, 1859.9025, 1832.5529],
        [ 141.5512,  200.1582, 3921.3518, 1822.2751],
        [ 242.8794,  273.7198, 3453.4316, 1910.3984],
        [ 400.1719,  673.6343, 1909.9603, 1867.7961]], device='cuda:0',
       grad_fn=<IndexBackward>)
```



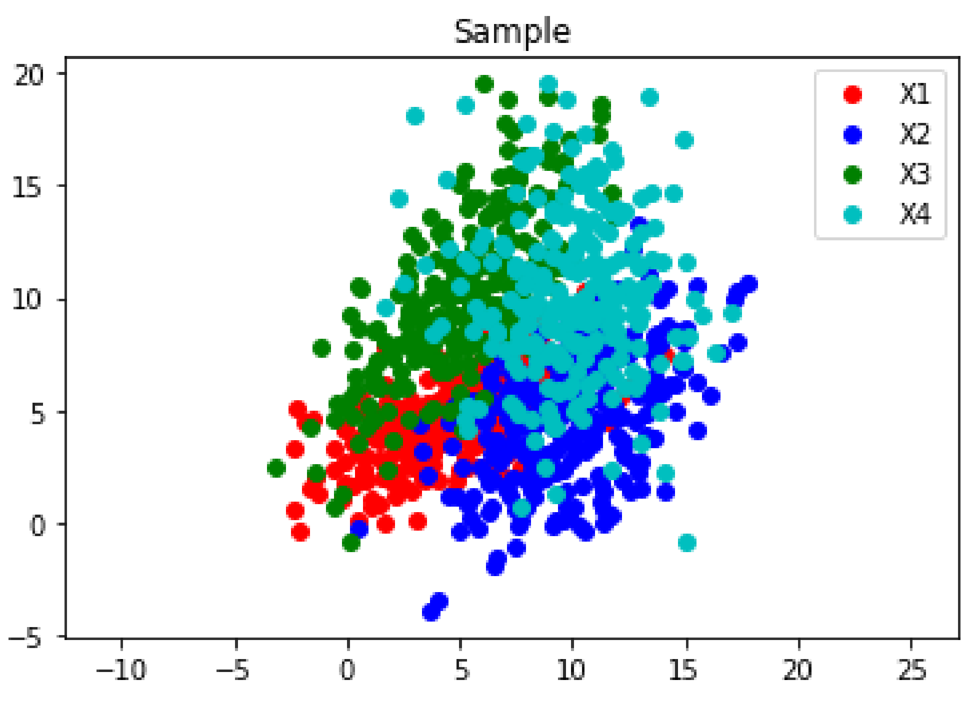
2. Write a program that samples 1000 points from a mixture of 4 2D Gaussians with identity covariance centered at (5,5), (10,5), (5,10), and (10,10). Provide the code and a plot of the sample.

[50]:
```
def plot_data(fig, ax, X1, X2,X3,X4, labels):
    plt.title('Sample')
    ax.plot(X1[:,0], X1[:,1], 'ro', label=labels[0])
    ax.plot(X2[:,0], X2[:,1], 'bo', label=labels[1])
    ax.plot(X3[:,0], X3[:,1], 'go', label=labels[2])
    ax.plot(X4[:,0], X3[:,1], 'co', label=labels[3])
    ax.axis('equal')
    ax.legend()
```

[49]:
```
#x1
mean = np.array([5, 5])
cov = np.array([[5, 5], [10, 5]])
X1 = np.random.multivariate_normal(mean, cov, 250)

mean2 = np.array([10, 5])
cov2 = np.array([[10,5], [5, 10]])
X2 = np.random.multivariate_normal(mean2, cov2, 250)
```

```
mean3 = np.array([5, 10])
cov3 = np.array([[5, 10], [10, 10]])
X3 = np.random.multivariate_normal(mean3, cov3, 250)

mean4 = np.array([10, 10])
cov4 = np.array([[10, 10], [10, 10]])
X4 = np.random.multivariate_normal(mean4, cov4, 250)

#Train data
X = np.concatenate((X1,X2,X3,X4),axis=0)
print(X.shape)
fig,ax = plt.subplots(1,1)
plot_data(fig, ax, X1, X2,X3,X4, ['X1', 'X2','X3','X4'])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: RuntimeWarning:
covariance is not positive-semidefinite.
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: RuntimeWarning:
covariance is not positive-semidefinite.
  if sys.path[0] == '':
(1000, 2)
```



3. Write a GAN generator G and discriminator D to model the dataset you generated in Question 2. Train the GAN and display two plots: a fake sample from the generator and the original

sample from Question 2.

[53]:
```python
#GAN
import torch
from torch import nn, optim
from torch.autograd.variable import Variable
from torchvision import transforms, datasets
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time

N_Z_PARAMS = 8
class DiscriminativeNet(torch.nn.Module):
    """
    A two hidden-layer discriminative neural network
    """
    def __init__(self):
        super(DiscriminativeNet, self).__init__()
        n_features = 2
        n_out = 1

        self.hidden0 = nn.Sequential(
            nn.Linear(n_features, 8),
            nn.LeakyReLU(0.2)
        )
        self.hidden1 = nn.Sequential(
            nn.Linear(8, 4),
            nn.LeakyReLU(0.2)
        )
        self.out = nn.Sequential(
            torch.nn.Linear(4, n_out),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x = self.hidden0(x)
        x = self.hidden1(x)
        x = self.out(x)
        return x

class GenerativeNet(torch.nn.Module):
    """
    A three hidden-layer generative neural network
    """

    def __init__(self):
```

```
        super(GenerativeNet, self).__init__()
        n_features = N_Z_PARAMS
        n_out = 2

        self.hidden0 = nn.Sequential(
            nn.Linear(n_features, 4),
            nn.LeakyReLU(0.2)
        )
        self.hidden1 = nn.Sequential(
            nn.Linear(4, 8),
            nn.LeakyReLU(0.2)
        )
        self.out = nn.Sequential(
            nn.Linear(8, n_out)
        )

    def forward(self, x):
        x = self.hidden0(x)
        x = self.hidden1(x)
        x = self.out(x)
        return x

def noise(size):
    n = torch.randn(size, N_Z_PARAMS)
    n = n.to(device)
    return n

def real_data_target(size):
    '''
    Tensor containing ones, with shape = size
    '''
    data = Variable(torch.ones(size, 1))
    data = data.to(device)
    return data

def fake_data_target(size):
    '''
    Tensor containing zeros, with shape = size
    '''
    data = Variable(torch.zeros(size, 1))
    data = data.to(device)
    return data

def train_discriminator(optimizer, real_data, fake_data):
    # Reset gradients
    optimizer.zero_grad()
```

```
    # Propagate real data
    prediction_real = discriminator(real_data)
    error_real = loss(prediction_real, real_data_target(real_data.size(0)))
    error_real.backward()

    # Propagate fake data
    prediction_fake = discriminator(fake_data)
    error_fake = loss(prediction_fake, fake_data_target(real_data.size(0)))
    error_fake.backward()

    # Take a step
    optimizer.step()

    # Return error
    return error_real + error_fake, prediction_real, prediction_fake

def plt_output(fake_data):
    plt.figure(figsize=(8,8))
    plt.xlim(-20,20)
    plt.ylim(-20,20)
    plt.scatter(fake_data[:,0],fake_data[:,1])
    plt.show()

def train_generator(optimizer, fake_data):
    # Reset gradients
    optimizer.zero_grad()

    # Propagate the fake data through the discriminator and backpropagate.
    # Note that since we want the generator to output something that gets
    # the discriminator to output a 1, we use the real data target here.
    prediction = discriminator(fake_data)
    error = loss(prediction, real_data_target(prediction.size(0)))
    error.backward()

    # Update weights with gradients
    optimizer.step()

    # Return error
    return error
```

[54]:
```
def plot_data(fig, ax, X1, X2, labels):
    plt.title('Sample')
    ax.plot(X1[:,0], X1[:,1], 'ro', label=labels[0])
    ax.plot(X2[:,0], X2[:,1], 'bo', label=labels[1])
    ax.axis('equal')
    ax.legend()
```

[55]:
```
#Train
samples = torch.Tensor(X)
dataset = torch.utils.data.TensorDataset(samples)
data_loader = torch.utils.data.DataLoader(dataset, batch_size=100, shuffle=True)
num_batches = len(data_loader)
```

[58]:
```
#Test
num_test_samples = 1000
test_noise = noise(num_test_samples)
```

[59]:
```
# Create generator and discriminator

discriminator = DiscriminativeNet()
generator = GenerativeNet()

if torch.cuda.is_available():
    generator.to(device)
    discriminator.to(device)

d_optimizer = optim.Adam(discriminator.parameters(), lr=0.0002)
g_optimizer = optim.Adam(generator.parameters(), lr=0.0002)

loss = nn.BCELoss()
```

[61]:
```
#Training process
num_epochs = 2000
d_error_arr = []
g_error_arr = []
fig,ax = plt.subplots(1,1)
for epoch in range(num_epochs):
    n_batches = 0
    g_err = 0
    d_err = 0
    for n_batch, [real_data] in enumerate(data_loader):

        # Train Discriminator
        real_data = Variable(real_data)
        real_data = real_data.to(device)
        fake_data = generator(noise(real_data.size(0))).detach()
        d_error, d_pred_real, d_pred_fake = train_discriminator(d_optimizer,
                                                                 real_data,␣
 ↪fake_data)
        d_err += d_error.cpu().detach().numpy()

        # Train Generator

        fake_data = generator(noise(real_data.size(0)))
```

```
        g_error = train_generator(g_optimizer, fake_data)
        g_err += g_error.cpu().detach().numpy()
        n_batches = n_batches + 1
    g_error_arr.append(g_error/n_batches)
    d_error_arr.append(d_error/n_batches)
    print('Epoch %d generator loss %f discriminator loss %f' %
                (epoch, g_error_arr[epoch], d_error_arr[epoch]))
    test_data_fake = generator(test_noise).cpu().detach()
plot_data(fig, ax, X, test_data_fake, ['Real test data', 'Generated test data'])
```

```
 discriminator loss 0.139680
Epoch 1683 generator loss 0.069156 discriminator loss 0.138839
Epoch 1684 generator loss 0.069167 discriminator loss 0.138685
Epoch 1685 generator loss 0.067895 discriminator loss 0.138690
Epoch 1686 generator loss 0.068792 discriminator loss 0.139423
Epoch 1687 generator loss 0.069206 discriminator loss 0.139144
Epoch 1688 generator loss 0.069175 discriminator loss 0.138606
Epoch 1689 generator loss 0.069350 discriminator loss 0.138647
Epoch 1690 generator loss 0.070395 discriminator loss 0.138412
Epoch 1691 generator loss 0.071000 discriminator loss 0.138616
Epoch 1692 generator loss 0.070905 discriminator loss 0.138712
Epoch 1693 generator loss 0.070033 discriminator loss 0.138758
Epoch 1694 generator loss 0.069353 discriminator loss 0.138396
Epoch 1695 generator loss 0.069609 discriminator loss 0.138608
Epoch 1696 generator loss 0.069097 discriminator loss 0.138413
Epoch 1697 generator loss 0.068778 discriminator loss 0.139111
Epoch 1698 generator loss 0.068761 discriminator loss 0.139191
Epoch 1699 generator loss 0.069512 discriminator loss 0.139078
Epoch 1700 generator loss 0.070062 discriminator loss 0.138652
Epoch 1701 generator loss 0.068546 discriminator loss 0.138847
Epoch 1702 generator loss 0.068406 discriminator loss 0.138399
Epoch 1703 generator loss 0.069057 discriminator loss 0.138892
Epoch 1704 generator loss 0.069747 discriminator loss 0.138720
Epoch 1705 generator loss 0.070325 discriminator loss 0.138593
Epoch 1706 generator loss 0.068463 discriminator loss 0.138671
Epoch 1707 generator loss 0.068449 discriminator loss 0.139239
Epoch 1708 generator loss 0.068130 discriminator loss 0.138011
Epoch 1709 generator loss 0.068858 discriminator loss 0.139036
Epoch 1710 generator loss 0.068788 discriminator loss 0.138338
Epoch 1711 generator loss 0.068199 discriminator loss 0.138410
Epoch 1712 generator loss 0.069468 discriminator loss 0.138366
Epoch 1713 generator loss 0.069166 discriminator loss 0.138973
Epoch 1714 generator loss 0.068757 discriminator loss 0.138170
Epoch 1715 generator loss 0.069271 discriminator loss 0.138681
Epoch 1716 generator loss 0.069153 discriminator loss 0.138764
Epoch 1717 generator loss 0.069232 discriminator loss 0.139347
Epoch 1718 generator loss 0.068962 discriminator loss 0.137927
```

```
Epoch 1719 generator loss 0.069546 discriminator loss 0.138577
Epoch 1720 generator loss 0.069943 discriminator loss 0.138758
Epoch 1721 generator loss 0.070127 discriminator loss 0.139119
Epoch 1722 generator loss 0.070466 discriminator loss 0.138816
Epoch 1723 generator loss 0.070329 discriminator loss 0.138613
Epoch 1724 generator loss 0.069892 discriminator loss 0.137982
Epoch 1725 generator loss 0.070479 discriminator loss 0.138896
Epoch 1726 generator loss 0.070794 discriminator loss 0.139225
Epoch 1727 generator loss 0.069824 discriminator loss 0.139144
Epoch 1728 generator loss 0.070207 discriminator loss 0.137707
Epoch 1729 generator loss 0.071195 discriminator loss 0.139086
Epoch 1730 generator loss 0.068982 discriminator loss 0.138609
Epoch 1731 generator loss 0.067552 discriminator loss 0.139993
Epoch 1732 generator loss 0.068259 discriminator loss 0.139303
Epoch 1733 generator loss 0.070591 discriminator loss 0.138873
Epoch 1734 generator loss 0.069620 discriminator loss 0.137874
Epoch 1735 generator loss 0.068491 discriminator loss 0.138561
Epoch 1736 generator loss 0.068135 discriminator loss 0.138874
Epoch 1737 generator loss 0.069166 discriminator loss 0.138565
Epoch 1738 generator loss 0.068957 discriminator loss 0.138340
Epoch 1739 generator loss 0.069536 discriminator loss 0.138660
Epoch 1740 generator loss 0.069816 discriminator loss 0.138491
Epoch 1741 generator loss 0.068413 discriminator loss 0.139256
Epoch 1742 generator loss 0.068261 discriminator loss 0.139196
Epoch 1743 generator loss 0.068621 discriminator loss 0.139275
Epoch 1744 generator loss 0.068927 discriminator loss 0.139084
Epoch 1745 generator loss 0.069807 discriminator loss 0.139614
Epoch 1746 generator loss 0.069493 discriminator loss 0.138911
Epoch 1747 generator loss 0.068960 discriminator loss 0.138601
Epoch 1748 generator loss 0.068781 discriminator loss 0.138839
Epoch 1749 generator loss 0.068737 discriminator loss 0.138060
Epoch 1750 generator loss 0.068931 discriminator loss 0.138783
Epoch 1751 generator loss 0.068122 discriminator loss 0.138518
Epoch 1752 generator loss 0.069741 discriminator loss 0.138746
Epoch 1753 generator loss 0.069849 discriminator loss 0.139651
Epoch 1754 generator loss 0.069812 discriminator loss 0.139052
Epoch 1755 generator loss 0.068190 discriminator loss 0.139372
Epoch 1756 generator loss 0.069232 discriminator loss 0.139360
Epoch 1757 generator loss 0.070306 discriminator loss 0.138818
Epoch 1758 generator loss 0.069745 discriminator loss 0.138365
Epoch 1759 generator loss 0.067955 discriminator loss 0.138682
Epoch 1760 generator loss 0.070223 discriminator loss 0.139447
Epoch 1761 generator loss 0.071012 discriminator loss 0.138767
Epoch 1762 generator loss 0.069449 discriminator loss 0.138568
Epoch 1763 generator loss 0.068736 discriminator loss 0.139276
Epoch 1764 generator loss 0.069380 discriminator loss 0.138708
Epoch 1765 generator loss 0.069265 discriminator loss 0.139274
Epoch 1766 generator loss 0.069361 discriminator loss 0.139171
```

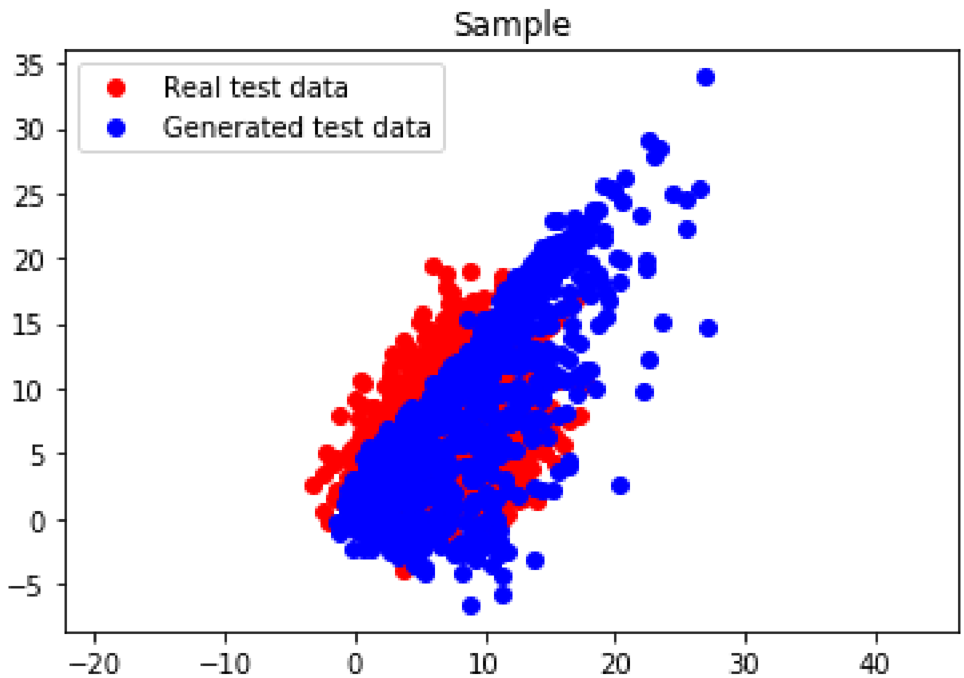
```
Epoch 1767 generator loss 0.070272 discriminator loss 0.138548
Epoch 1768 generator loss 0.070406 discriminator loss 0.138567
Epoch 1769 generator loss 0.070075 discriminator loss 0.138476
Epoch 1770 generator loss 0.070655 discriminator loss 0.139138
Epoch 1771 generator loss 0.070212 discriminator loss 0.138770
Epoch 1772 generator loss 0.069954 discriminator loss 0.139127
Epoch 1773 generator loss 0.070446 discriminator loss 0.138711
Epoch 1774 generator loss 0.070297 discriminator loss 0.139004
Epoch 1775 generator loss 0.068890 discriminator loss 0.138740
Epoch 1776 generator loss 0.068090 discriminator loss 0.138534
Epoch 1777 generator loss 0.069232 discriminator loss 0.138565
Epoch 1778 generator loss 0.069994 discriminator loss 0.138868
Epoch 1779 generator loss 0.069816 discriminator loss 0.138841
Epoch 1780 generator loss 0.068484 discriminator loss 0.138970
Epoch 1781 generator loss 0.069256 discriminator loss 0.138497
Epoch 1782 generator loss 0.068441 discriminator loss 0.138708
Epoch 1783 generator loss 0.069304 discriminator loss 0.138700
Epoch 1784 generator loss 0.069721 discriminator loss 0.138530
Epoch 1785 generator loss 0.070393 discriminator loss 0.138302
Epoch 1786 generator loss 0.069803 discriminator loss 0.139003
Epoch 1787 generator loss 0.068890 discriminator loss 0.138838
Epoch 1788 generator loss 0.069679 discriminator loss 0.138486
Epoch 1789 generator loss 0.070616 discriminator loss 0.138446
Epoch 1790 generator loss 0.069810 discriminator loss 0.138703
Epoch 1791 generator loss 0.069479 discriminator loss 0.138806
Epoch 1792 generator loss 0.069090 discriminator loss 0.138210
Epoch 1793 generator loss 0.069219 discriminator loss 0.138462
Epoch 1794 generator loss 0.070463 discriminator loss 0.138902
Epoch 1795 generator loss 0.069865 discriminator loss 0.138393
Epoch 1796 generator loss 0.069464 discriminator loss 0.138453
Epoch 1797 generator loss 0.068939 discriminator loss 0.138314
Epoch 1798 generator loss 0.069083 discriminator loss 0.138821
Epoch 1799 generator loss 0.068597 discriminator loss 0.138443
Epoch 1800 generator loss 0.068966 discriminator loss 0.137913
Epoch 1801 generator loss 0.069219 discriminator loss 0.138112
Epoch 1802 generator loss 0.069307 discriminator loss 0.138566
Epoch 1803 generator loss 0.068989 discriminator loss 0.138388
Epoch 1804 generator loss 0.070515 discriminator loss 0.138877
Epoch 1805 generator loss 0.072050 discriminator loss 0.138873
Epoch 1806 generator loss 0.071853 discriminator loss 0.138412
Epoch 1807 generator loss 0.070176 discriminator loss 0.138646
Epoch 1808 generator loss 0.069623 discriminator loss 0.138529
Epoch 1809 generator loss 0.069137 discriminator loss 0.138273
Epoch 1810 generator loss 0.069487 discriminator loss 0.138486
Epoch 1811 generator loss 0.070782 discriminator loss 0.138727
Epoch 1812 generator loss 0.070260 discriminator loss 0.138683
Epoch 1813 generator loss 0.069698 discriminator loss 0.137972
Epoch 1814 generator loss 0.069656 discriminator loss 0.139207
```

```
Epoch 1815 generator loss 0.070251 discriminator loss 0.139058
Epoch 1816 generator loss 0.069046 discriminator loss 0.138299
Epoch 1817 generator loss 0.069143 discriminator loss 0.138593
Epoch 1818 generator loss 0.069173 discriminator loss 0.138900
Epoch 1819 generator loss 0.069982 discriminator loss 0.138105
Epoch 1820 generator loss 0.069858 discriminator loss 0.138693
Epoch 1821 generator loss 0.069593 discriminator loss 0.138102
Epoch 1822 generator loss 0.068991 discriminator loss 0.138826
Epoch 1823 generator loss 0.068234 discriminator loss 0.138635
Epoch 1824 generator loss 0.068900 discriminator loss 0.138573
Epoch 1825 generator loss 0.068532 discriminator loss 0.138199
Epoch 1826 generator loss 0.069004 discriminator loss 0.138189
Epoch 1827 generator loss 0.069556 discriminator loss 0.138267
Epoch 1828 generator loss 0.069434 discriminator loss 0.138575
Epoch 1829 generator loss 0.068479 discriminator loss 0.138937
Epoch 1830 generator loss 0.068773 discriminator loss 0.138857
Epoch 1831 generator loss 0.069656 discriminator loss 0.138830
Epoch 1832 generator loss 0.069990 discriminator loss 0.138443
Epoch 1833 generator loss 0.070098 discriminator loss 0.138336
Epoch 1834 generator loss 0.068877 discriminator loss 0.138426
Epoch 1835 generator loss 0.068992 discriminator loss 0.138797
Epoch 1836 generator loss 0.069460 discriminator loss 0.138924
Epoch 1837 generator loss 0.070654 discriminator loss 0.138412
Epoch 1838 generator loss 0.070414 discriminator loss 0.138468
Epoch 1839 generator loss 0.068758 discriminator loss 0.138764
Epoch 1840 generator loss 0.067809 discriminator loss 0.138052
Epoch 1841 generator loss 0.068155 discriminator loss 0.138616
Epoch 1842 generator loss 0.070234 discriminator loss 0.138199
Epoch 1843 generator loss 0.070068 discriminator loss 0.138967
Epoch 1844 generator loss 0.069378 discriminator loss 0.138907
Epoch 1845 generator loss 0.069973 discriminator loss 0.138913
Epoch 1846 generator loss 0.069117 discriminator loss 0.138258
Epoch 1847 generator loss 0.068560 discriminator loss 0.139266
Epoch 1848 generator loss 0.068827 discriminator loss 0.138472
Epoch 1849 generator loss 0.068471 discriminator loss 0.138660
Epoch 1850 generator loss 0.068486 discriminator loss 0.138237
Epoch 1851 generator loss 0.069311 discriminator loss 0.139239
Epoch 1852 generator loss 0.069963 discriminator loss 0.139164
Epoch 1853 generator loss 0.069377 discriminator loss 0.138424
Epoch 1854 generator loss 0.068847 discriminator loss 0.138769
Epoch 1855 generator loss 0.069254 discriminator loss 0.138405
Epoch 1856 generator loss 0.068193 discriminator loss 0.138212
Epoch 1857 generator loss 0.068652 discriminator loss 0.138968
Epoch 1858 generator loss 0.068475 discriminator loss 0.138275
Epoch 1859 generator loss 0.068600 discriminator loss 0.138912
Epoch 1860 generator loss 0.068766 discriminator loss 0.139295
Epoch 1861 generator loss 0.069007 discriminator loss 0.138322
Epoch 1862 generator loss 0.069411 discriminator loss 0.138943
```

Epoch 1863 generator loss 0.069441 discriminator loss 0.139145
Epoch 1864 generator loss 0.068614 discriminator loss 0.139063
Epoch 1865 generator loss 0.068443 discriminator loss 0.139299
Epoch 1866 generator loss 0.069461 discriminator loss 0.138834
Epoch 1867 generator loss 0.071608 discriminator loss 0.138871
Epoch 1868 generator loss 0.070833 discriminator loss 0.139326
Epoch 1869 generator loss 0.068386 discriminator loss 0.139460
Epoch 1870 generator loss 0.067866 discriminator loss 0.139509
Epoch 1871 generator loss 0.068366 discriminator loss 0.138987
Epoch 1872 generator loss 0.068780 discriminator loss 0.138764
Epoch 1873 generator loss 0.068686 discriminator loss 0.138993
Epoch 1874 generator loss 0.069009 discriminator loss 0.140417
Epoch 1875 generator loss 0.068589 discriminator loss 0.139711
Epoch 1876 generator loss 0.068872 discriminator loss 0.138895
Epoch 1877 generator loss 0.069545 discriminator loss 0.138928
Epoch 1878 generator loss 0.069836 discriminator loss 0.139286
Epoch 1879 generator loss 0.069206 discriminator loss 0.139190
Epoch 1880 generator loss 0.068760 discriminator loss 0.138664
Epoch 1881 generator loss 0.067871 discriminator loss 0.138455
Epoch 1882 generator loss 0.068482 discriminator loss 0.139816
Epoch 1883 generator loss 0.070262 discriminator loss 0.139451
Epoch 1884 generator loss 0.070039 discriminator loss 0.139589
Epoch 1885 generator loss 0.067604 discriminator loss 0.139115
Epoch 1886 generator loss 0.067954 discriminator loss 0.139133
Epoch 1887 generator loss 0.067052 discriminator loss 0.140223
Epoch 1888 generator loss 0.068010 discriminator loss 0.139455
Epoch 1889 generator loss 0.068927 discriminator loss 0.139023
Epoch 1890 generator loss 0.068698 discriminator loss 0.139251
Epoch 1891 generator loss 0.068992 discriminator loss 0.138071
Epoch 1892 generator loss 0.069438 discriminator loss 0.139018
Epoch 1893 generator loss 0.068844 discriminator loss 0.138676
Epoch 1894 generator loss 0.068930 discriminator loss 0.139515
Epoch 1895 generator loss 0.069337 discriminator loss 0.139078
Epoch 1896 generator loss 0.070237 discriminator loss 0.138328
Epoch 1897 generator loss 0.069038 discriminator loss 0.139667
Epoch 1898 generator loss 0.067730 discriminator loss 0.139472
Epoch 1899 generator loss 0.068483 discriminator loss 0.139196
Epoch 1900 generator loss 0.069498 discriminator loss 0.139049
Epoch 1901 generator loss 0.070395 discriminator loss 0.139172
Epoch 1902 generator loss 0.069105 discriminator loss 0.138980
Epoch 1903 generator loss 0.067724 discriminator loss 0.138827
Epoch 1904 generator loss 0.069127 discriminator loss 0.137630
Epoch 1905 generator loss 0.068687 discriminator loss 0.138514
Epoch 1906 generator loss 0.070240 discriminator loss 0.138608
Epoch 1907 generator loss 0.070660 discriminator loss 0.139182
Epoch 1908 generator loss 0.071083 discriminator loss 0.139050
Epoch 1909 generator loss 0.068675 discriminator loss 0.139796
Epoch 1910 generator loss 0.068919 discriminator loss 0.139336

```
Epoch 1911 generator loss 0.069209 discriminator loss 0.138883
Epoch 1912 generator loss 0.070771 discriminator loss 0.137323
Epoch 1913 generator loss 0.070227 discriminator loss 0.138779
Epoch 1914 generator loss 0.069978 discriminator loss 0.138011
Epoch 1915 generator loss 0.069677 discriminator loss 0.137781
Epoch 1916 generator loss 0.070431 discriminator loss 0.138253
Epoch 1917 generator loss 0.067066 discriminator loss 0.137924
Epoch 1918 generator loss 0.069773 discriminator loss 0.137446
Epoch 1919 generator loss 0.069462 discriminator loss 0.137490
Epoch 1920 generator loss 0.066893 discriminator loss 0.136393
Epoch 1921 generator loss 0.069748 discriminator loss 0.138034
Epoch 1922 generator loss 0.068955 discriminator loss 0.137282
Epoch 1923 generator loss 0.069695 discriminator loss 0.136336
Epoch 1924 generator loss 0.069507 discriminator loss 0.136453
Epoch 1925 generator loss 0.070071 discriminator loss 0.136687
Epoch 1926 generator loss 0.070037 discriminator loss 0.136695
Epoch 1927 generator loss 0.070367 discriminator loss 0.136573
Epoch 1928 generator loss 0.070325 discriminator loss 0.135257
Epoch 1929 generator loss 0.070979 discriminator loss 0.136554
Epoch 1930 generator loss 0.068496 discriminator loss 0.137800
Epoch 1931 generator loss 0.070615 discriminator loss 0.136902
Epoch 1932 generator loss 0.070006 discriminator loss 0.136488
Epoch 1933 generator loss 0.069175 discriminator loss 0.137511
Epoch 1934 generator loss 0.070541 discriminator loss 0.136012
Epoch 1935 generator loss 0.069279 discriminator loss 0.137819
Epoch 1936 generator loss 0.069466 discriminator loss 0.136925
Epoch 1937 generator loss 0.068922 discriminator loss 0.135731
Epoch 1938 generator loss 0.068345 discriminator loss 0.138328
Epoch 1939 generator loss 0.067979 discriminator loss 0.137203
Epoch 1940 generator loss 0.067945 discriminator loss 0.139007
Epoch 1941 generator loss 0.066810 discriminator loss 0.137666
Epoch 1942 generator loss 0.068150 discriminator loss 0.138037
Epoch 1943 generator loss 0.070586 discriminator loss 0.135998
Epoch 1944 generator loss 0.069089 discriminator loss 0.137225
Epoch 1945 generator loss 0.067727 discriminator loss 0.136733
Epoch 1946 generator loss 0.069062 discriminator loss 0.137222
Epoch 1947 generator loss 0.068715 discriminator loss 0.134944
Epoch 1948 generator loss 0.067644 discriminator loss 0.138523
Epoch 1949 generator loss 0.067721 discriminator loss 0.137056
Epoch 1950 generator loss 0.065984 discriminator loss 0.136778
Epoch 1951 generator loss 0.068624 discriminator loss 0.137086
Epoch 1952 generator loss 0.068107 discriminator loss 0.137774
Epoch 1953 generator loss 0.068680 discriminator loss 0.138667
Epoch 1954 generator loss 0.068006 discriminator loss 0.135170
Epoch 1955 generator loss 0.067168 discriminator loss 0.139392
Epoch 1956 generator loss 0.069630 discriminator loss 0.136970
Epoch 1957 generator loss 0.067887 discriminator loss 0.138509
Epoch 1958 generator loss 0.068340 discriminator loss 0.139873
```

```
Epoch 1959 generator loss 0.069249 discriminator loss 0.139466
Epoch 1960 generator loss 0.069414 discriminator loss 0.136757
Epoch 1961 generator loss 0.068792 discriminator loss 0.138638
Epoch 1962 generator loss 0.068737 discriminator loss 0.139293
Epoch 1963 generator loss 0.067828 discriminator loss 0.140587
Epoch 1964 generator loss 0.068527 discriminator loss 0.141263
Epoch 1965 generator loss 0.070802 discriminator loss 0.139174
Epoch 1966 generator loss 0.069663 discriminator loss 0.138795
Epoch 1967 generator loss 0.068750 discriminator loss 0.139789
Epoch 1968 generator loss 0.069448 discriminator loss 0.139180
Epoch 1969 generator loss 0.070516 discriminator loss 0.139512
Epoch 1970 generator loss 0.070033 discriminator loss 0.137265
Epoch 1971 generator loss 0.069496 discriminator loss 0.138579
Epoch 1972 generator loss 0.068509 discriminator loss 0.139754
Epoch 1973 generator loss 0.070465 discriminator loss 0.139980
Epoch 1974 generator loss 0.069974 discriminator loss 0.139775
Epoch 1975 generator loss 0.069394 discriminator loss 0.141260
Epoch 1976 generator loss 0.068995 discriminator loss 0.140089
Epoch 1977 generator loss 0.068149 discriminator loss 0.140916
Epoch 1978 generator loss 0.070229 discriminator loss 0.139726
Epoch 1979 generator loss 0.071322 discriminator loss 0.139435
Epoch 1980 generator loss 0.069947 discriminator loss 0.140596
Epoch 1981 generator loss 0.067440 discriminator loss 0.138611
Epoch 1982 generator loss 0.071257 discriminator loss 0.138801
Epoch 1983 generator loss 0.071947 discriminator loss 0.142851
Epoch 1984 generator loss 0.069995 discriminator loss 0.140041
Epoch 1985 generator loss 0.070335 discriminator loss 0.139915
Epoch 1986 generator loss 0.069232 discriminator loss 0.139565
Epoch 1987 generator loss 0.068976 discriminator loss 0.139572
Epoch 1988 generator loss 0.069369 discriminator loss 0.139639
Epoch 1989 generator loss 0.069824 discriminator loss 0.139996
Epoch 1990 generator loss 0.067692 discriminator loss 0.139859
Epoch 1991 generator loss 0.066691 discriminator loss 0.139583
Epoch 1992 generator loss 0.067029 discriminator loss 0.139086
Epoch 1993 generator loss 0.068690 discriminator loss 0.139600
Epoch 1994 generator loss 0.069911 discriminator loss 0.139395
Epoch 1995 generator loss 0.068516 discriminator loss 0.140061
Epoch 1996 generator loss 0.069667 discriminator loss 0.139306
Epoch 1997 generator loss 0.070156 discriminator loss 0.139438
Epoch 1998 generator loss 0.069131 discriminator loss 0.139970
Epoch 1999 generator loss 0.069002 discriminator loss 0.139943
```



4. Suppose you are working on a regression problem for which you have insufficient data and come up with the idea of using a GAN to generate new $(\mathbf{x}, y)$ pairs. First explain precisely how this could be done, then explain why it would be a bad idea.

Answer 4.

1. It is possible to use GAN to working on regression problem. GAN model receive input X from data and go to generative model to generate fake data using noise. Furthermore, we feed X data and fake data into discriminative network. This network is binary-logistic regression which predict real and fake data. Finally, model will update loss and update a generative model for generate fake data that similarly to real data and discriminative model for more accuracy to predict a real data and fake data.

2. I think this is a bad idea because:

   2.1 There are insufficient data to train the model. Discriminator not have accuracy when we use insufficient data.

   2.2 Generative model are hard to estimate because of ”Latent variable” which we cannot unknow what later vaiable have.

5. Briefly explain the purpose of weight decay and weight clipping, including how they are similar and how they are different.

Answer 5

1.Weight decay is a part of a regularisation in neural network. This is try to optimize a weight by adding small penealty to loss function. We use weight because to prevent overfitting and vanishing gradient.

Example of weight decay is L2-norm

2.Weight cliping showed in Wasserstein GAN which try to prevent overfitting and vanishing gradient same as weight decay.

Weight cliping optimize weight in neural network in the spcific range.

For Example, as you can see in algorithms of Wasserstein GAN. After we update a discriminator, the model compute a weight using RMSProp and clipping a weight into hyperparameter c. this is mean the weight in discriminator must be within a certain range controlled by hyperparameter C.

To conclution, I think weight decay and weight cliping are doing in same purpose that there try to optimize a weight in neural network to prevent overfitting and vanishing gradient(Exploding Gradient). However, there are different in method to do. For weight decay, this technique include in l2 regulalization with small penalty while weight cliping use after compute weight and clipping the weight in range of some hyperparameter.

[ ]:

```

```