گزارش تمرین 2

محمدرضا کبودیان 401207713

**الگوریتم Supervised:**

ابتدا با دیتالودر داده های آموزش و تست را تعریف میکنیم

سپس تابع embed_label_into_data را تعریف می کنیم که در آن به جای 10 پیکسل اول تصویر، لیبل وان هات شده را در آن embed می کنیم.

```
def embed_label_into_data(x, y):
    x_ = x.clone()
    x_[:, :10] *= 0.0
    x_[range(x.shape[0]), y] = x.max()
    return x_
```

در ادامه کلاس یک لایه را تعریف می‌کنیم که دارای ورودی تعداد نورون های ورودی و خروجی است و در نهایت به یک تابع relu ختم می شود. در مسیر فوروارد، ابتدا فقط جهت ورودی را باید به combiner آن بدهیم زیرا نباید به طور خود به خودی در لایه های جلوتر ، goodness را به بالای ترشهولد ببرد.در ادامه آن را به combiner خطی می‌دهیم و از آن relu می‌گیریم.

```
def forward(self, x):
        x_direction = x / (x.norm(2, 1, keepdim=True) + 1e-4)
        return self.relu(
            torch.mm(x_direction, self.weight.T) +
            self.bias.unsqueeze(0))
```

در قسمت آموزش لایه به ازای هر داده ی مثبت، یک داده ی منفی تعریف می‌کنیم که بردار embed شده ی آن به صورت غلط داده شده است. حال برای هر دو، مسیر فوروارد را محاسبه می‌کنیم. سپس طبق مقاله داریم :

$$loss = mean(log(1 + e^{[(threshold - positivedata),(negativedata - threshold)]}))$$

دلیل آن این است که می خواهیم goodness دیتای منفی به زیر ترشهولد برود.

پس برای محاسبه loss در یک batch داده داریم:

```
loss = torch.log(1 + torch.exp(torch.cat([-g_pos + self.threshold,g_neg -
self.threshold]))).mean()
```

در ادامه از این لایه گرادیان می‌گیریم. این گرادیان به لایه قبلی داده نمی‌شود پس EBP نداریم. در ادامه GD میگیریم و پارامتر های "لایه" را آپدیت می‌کنیم:

```
            self.opt.zero_grad()
```

```
            loss.backward()
            self.opt.step()
```

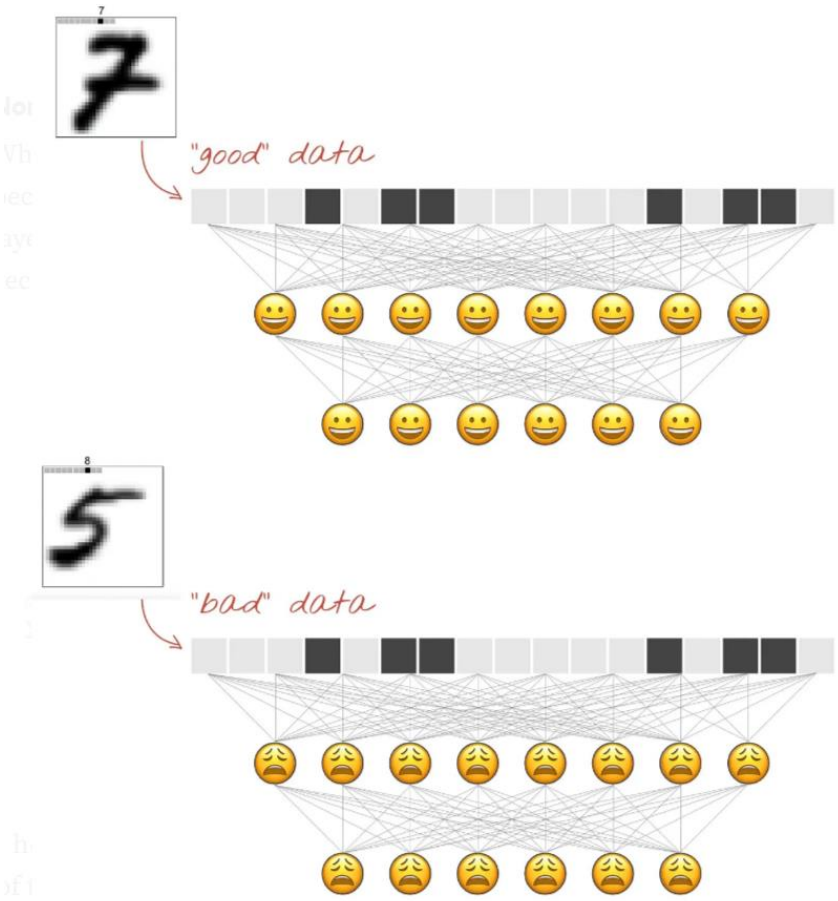
در نهایت برای استفاده مسیر فورووارد، آن را به لایه بعدی می‌دهیم:

```
        return self.forward(x_pos).detach(), self.forward(x_neg).detach()
```

حال کلاس Net را تعریف می‌کنیم که درواقع مجموعه ای از لایه های پشت سر هم است و ابعاد پشت سر هم لایه ها را به صورت لیست می‌گیرد و لایستی از کلاس های layer متناسب با ورودی و خروجی هر لایه تولید می‌کند:

```
class Net(torch.nn.Module):

    def __init__(self, dims):
        super().__init__()
        self.layers = []
        for d in range(len(dims) - 1):
            self.layers.append(Layer(dims[d], dims[d + 1]).cuda())
```

در تابع predict به ازای هر ورودی، 10 لیبل از 0 تا 9 تعریف میکنیم و آن ها را در ورودی embed می‌کنیم حال خروجی تمام نورون هایی که داشتیم را به صورت توان دو با هم جمع می‌کنیم. تشخیص یک لیبل درست embed شده به این صورت است که شبکه به ازای لیبل درست، تمام نورون های آن fire می شود و بیشترین مقدار goodness را می دهد. پس لیبلی که به ازای آن بیشترین گودنس را داریم ، لیبل پیش بینی شده است:




```
    def predict(self, x):
        goodness_per_label = []
        for label in range(10): و هـای تـعـریـف شـده #لـیـبـل شـدهembed
            h = embed_label_into_data(x, label)
            goodness = []
            for layer in self.layers: #  خـروجـی تـمـام نـورون هـای شـبـکـه را
میـگـیـریـم
                h = layer(h)
                goodness += [h.pow(2).mean(1)]

            goodness_per_label += [sum(goodness).unsqueeze(1)]
        goodness_per_label = torch.cat(goodness_per_label, 1)
        print(f'goodness_per_label = {goodness_per_label.argmax(1)}')
        return goodness_per_label.argmax(1)
```

در قسمت آموزش شبکه فقط لازم است داده های مثبت و منفی را بگیریم و از لایه اول آموزش را شروع کنیم. سپس پس از اتمام آموزش لایه اول، خروجی فورووارد آن را به لایه دوم می دهیم و این گونه تا لایه آخر ادامه می‌دهیم.

```
    def train(self, x_pos, x_neg):
        h_pos, h_neg = x_pos, x_neg
        for i, layer in enumerate(self.layers):
            print('training layer', i, ':')
            h_pos, h_neg = layer.train(h_pos, h_neg)
```

تابع original_pos_neg_image برای پلات یک نمونه اصلی و مثبت و منفی تعریف شده است.

**لود دیتا و تعریف شبکه :**

```
torch.manual_seed(0)
train_loader, test_loader = MNIST_loaders()

net = Net([784, 600, 500, 400])
x, y = next(iter(train_loader))
x, y = x.cuda(), y.cuda()
```

همان گونه که در کد بالا مشاهده می شود، داده های تست و آموزش را لود می‌کنیم سپس لایه های شبکه را تعریف می‌کنیم.

ساخت داده های مثبت و منفی :

برای ساخت داده های مثبت، لیبل درست را در داده آموزش embed می‌کنیم و برای داده ی منفی لیبل غلط!

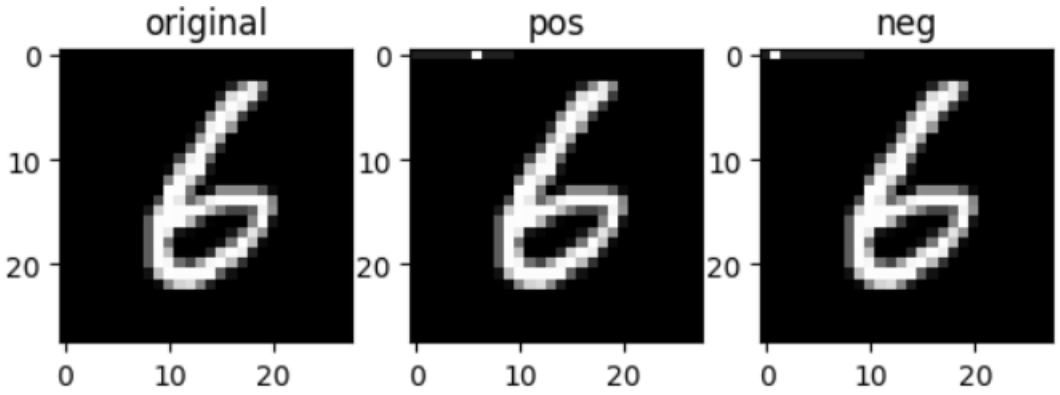
برای ساخت لیبل غلط، لیبل را با یک عدد رندوم از یک تا 9 جمع می کنیم و برای این که ممکن است از 9 بیشتر شود باقیمانده آن را بر 10 حساب می کنیم .

```
x_pos = embed_label_into_data(x, y)

# adding the label with modulo that makes the label
#  something other than the original label
randint1 = torch.randint(1, 10, y.shape, device = 'cuda')
y_neg = torch.remainder(y + randint1, 10)
x_neg = embed_label_into_data(x, y_neg)
```

در نهایت آموزش را شروع می کنیم. داریم:

```
original_pos_neg_image(x, x_pos, x_neg)

net.train(x_pos, x_neg)

print('train accuracy = ', net.predict(x).eq(y).float().mean().item())

x_test, y_test = next(iter(test_loader))
```

```
x_test, y_test = x_test.cuda(), y_test.cuda()

print('test accuracy = ',
net.predict(x_test).eq(y_test).float().mean().item())
```




```
training layer 0 :
100%|██████████| 1000/1000 [01:09<00:00, 14.45it/s]
training layer 1 :
100%|██████████| 1000/1000 [00:47<00:00, 20.93it/s]
training layer 2 :
100%|██████████| 1000/1000 [00:36<00:00, 27.23it/s]
goodness_per_label = tensor([6, 3, 1,  ..., 7, 7, 3], device='cuda:0')
train accuracy =  0.9149000048637390
goodness_per_label = tensor([7, 2, 1,  ..., 4, 5, 6], device='cuda:0')
test accuracy =  0.9192999601364136
```

مشاهده می کنیم برای داده های آموزش به دقت 0.91 و برای داده های تست به 0.919 رسیدیم که نشان می دهد بدون استفاده از EBP می توان یک شبکه را آموزش داد!

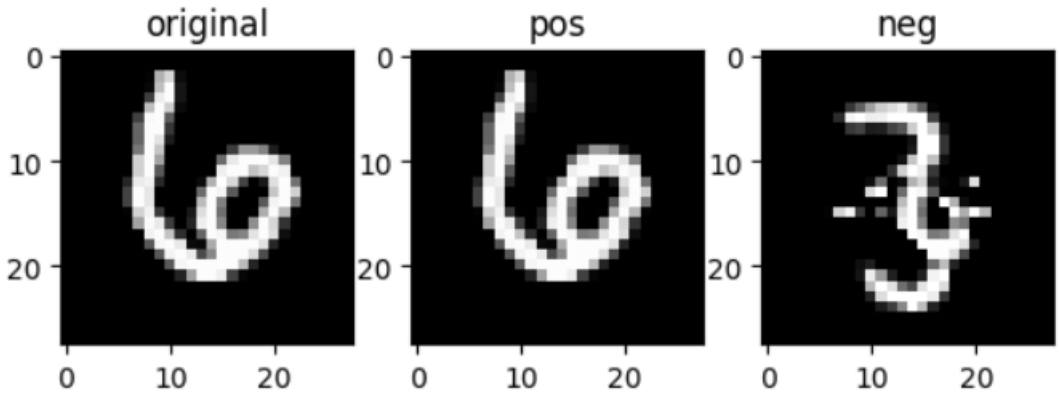
## الگوریتم unsupervised :

تفاوت این الگوریتم با الگوریتم قبلی این است که در شبکه FF لیبل را به شبکه آموزش نمی‌دهیم و آن را به لایه های آن نشان نمی‌دهیم به جای آن لیبل های مثبت و منفی تولید شده را به شبکه FF می دهیم و خروجی همه نورون های همه لایه های شبکه را به صورت یک بردار یک جا، به یک طبقه بند خطی می دهیم و حالا این بردار ورودی طبقه بند است و طبق این ورودی ، با لیبل، کلاسیفایر خطی را آموزش می‌دهیم. در واقع مانند این است که یک preprocess عصبی روی تصاویر صورت میگیرد و پس از آن خروجی بردار آن را به صورت معمولی با طبقه بند خطی و لیبل آن آموزش می‌دهیم:

بخش های layer و Net تقریبا مانند روش supervised است با ایت تفاوت که در کلاس net خروجی تمام نورون های شبکه را که همه فوروارد پسشان انجام شده به صورت بردار می کنیم و آماده به طبقه بند خطی می دهیم.

```
    def predict(self, x):
        layers_output = torch.Tensor([]).cuda()
        h = x
        for i, layer in enumerate(self.layers):
            h = layer(h)
            layers_output = torch.cat([layers_output,h],1)
        # print("layers_output = ", layers_output.shape)
        return layers_output
```

همچنین تفاوت اصلی این الگوریتم این است که داده های منفی به روش دیگری تولید می شوند . به این صورت که یک ماسک رندوم برای داده های منفی تولید می شود و داده با ماسک و مکمل ماسک دیتای غلط ترکیب می‌شود.




روش تولید ماسک:

برای تولید ماسک، چند عدد نقطه رندوم روی ماتریس 28*28 ایجاد می شوند و به ترتیب با خطوط ضخیمی طبق الگوریتم زیر به هم وصل می شوند.

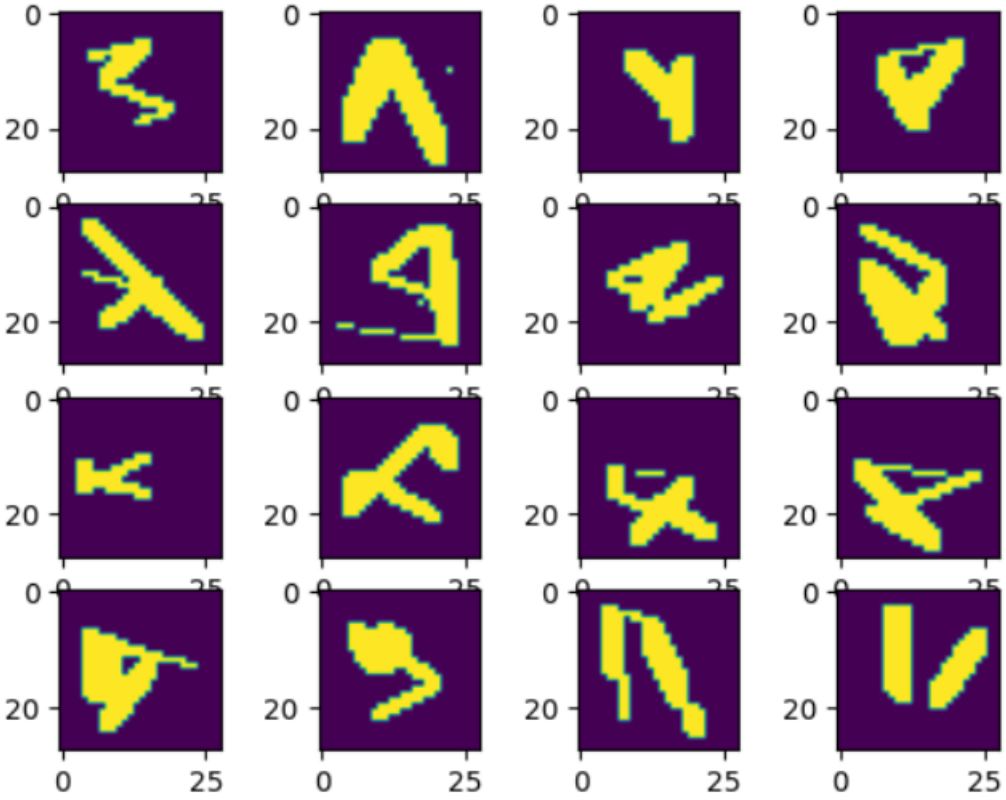
چون تولید ماسک ها وقت گیر است ، فقط 2000 ماسک تولید می شود و به صورت رندوم کپی و پخش می شوند تا به تعداد دادگان آموزش برسند:

```
# making random masks :
# we generate some random points in a 28*28 tensor then we
# fill between them with thick lines
x = torch.zeros(60000,28,28).cuda()
num_samp = 2000
mask = torch.zeros(50000,28,28).cuda()
num_random_points = 6
```

```
random_points = torch.randint(3,25,[x.size(0),num_random_points,
2],dtype=torch.int8).cuda()
random_points1 = torch.cat( (13+random_points[:num_samp,2,1].reshape(-
1,1)//2,2+random_points[:num_samp,0,0].reshape(-1,1)),1)
random_points2 = torch.cat((2+random_points[:num_samp,1,1].reshape(-
1,1),13+random_points[:num_samp,1,0].reshape(-1,1)//2),1)
random_points3 = torch.cat((2+random_points[:num_samp,0,1].reshape(-
1,1)//2,2+random_points[:num_samp,0,0].reshape(-1,1)//2),1)
random_points4 = random_points[:num_samp,3:6,:]
random_points = torch.cat((random_points1.reshape(-1,1,2),
random_points2.reshape(-1,1,2), random_points3.reshape(-1,1,2),
random_points4), 1)
# print(random_points)
mat = torch.randint(6,23,[5,2, 2]).cuda()

for i in range(num_samp):
  for rnd_pt_x1, rnd_pt_x2 in random_points[i,:,:]:
    # print(rnd_pt_x1,rnd_pt_x2)
    # print(type(rnd_pt_x1.item()),type(rnd_pt_x1.item()))
    mask[i, rnd_pt_x1, rnd_pt_x2] = 1

# plt.imshow(mask[0,:,:].cpu(), cmap="gray")
# plt.show()

# plt.imshow(mask[1,:,:].cpu(), cmap="gray")
# plt.show()

print("generating random masks:")

for j in tqdm(range(num_samp)):
  random_point = random_points[j,:,:]
  for i,(pt_x1, pt_y1) in enumerate(random_point[1:]):
    (pt_x0, pt_y0) = random_point[i,:]
    m = (pt_y1-pt_y0)/(pt_x1-pt_x0)
    d = pt_y1-m*pt_x1
    # print("*"*20, pt_x1, pt_y1)
    for x_ in range(min(pt_x0,pt_x1), max(pt_x0,pt_x1)+1):
      for y_ in range(min(pt_y0,pt_y1), max(pt_y0,pt_y1)+1):
        # print(x_,y_)
        if abs(y_ -m * x_ - d) < 3 :  # or torch.pow(x_-
pt_x0,2)+torch.pow(y_-pt_y0,2)<37
          mask[j,x_,y_] = 1
# plt.imshow(mask[0,:,:].cpu(), cmap="gray")
# plt.show()
```

```
# plt.imshow(mask[1,:,:].cpu(), cmap="gray")
# plt.show()
final_mask = torch.zeros(2000,28*28).cuda()
sum_mas = torch.sum(mask.reshape(-1,28*28),1)
ptr = 0
for i in range(len(final_mask)):
  # print(sum_mas[i])
  if 30<sum_mas[i]<160:
    final_mask[ptr,:] = mask[i,:,:].flatten()
    ptr+=1
i=0
while ptr<2000:
  final_mask[ptr,:] = mask[i,:,:].flatten()
  i+=1
  ptr+=1

for i in range(16):
  plt.subplot(4,4,i+1)
  plt.imshow(mask[i,:,:].cpu())
final_mask = final_mask.repeat(25,1)
rp = torch.randperm(50000)
final_mask = final_mask[rp,:]
```

نمونه ماسک های تولید شده:



ورودی کلاسیفایر خطی توضیح داده شد و داریم:

```
class LinearClassifier(torch.nn.Module):
  def __init__(self, lin_classifier_input_dimension,model):
    super().__init__()
    self.linear = torch.nn.Linear(lin_classifier_input_dimension,
10).cuda()
    self.optimizer = SGD(self.parameters(), lr=0.03)
    self.criterion = nn.CrossEntropyLoss()
    self.softmax = nn.Softmax()
    self.model = model

  def forward(self, x):
    return self.linear(x)

  def train(self, x, y, num_epoch):
```

```
    for j in tqdm(range(num_epoch)):
      y_tr = label_to_oh(y).cuda()
      ff_unsupervised = self.model.predict(x)
      y_hat = self.forward(ff_unsupervised)
      loss = self.criterion(y_hat,y_tr)
      self.optimizer.zero_grad()
      loss.backward()
      self.optimizer.step()


  def test(self, x):
    ff_unsupervised = self.model.predict(x)
    y_hat = self.forward(ff_unsupervised)
    y_hat = torch.argmax(y_hat, dim=1)
    return y_hat
```

همان گونه که مشاهده شد شبکه FF در کد زیر به linear classifier متصل می شود و آموزش می بیند:

```
epochs = 100
ff_output_neurons = net.output_neurons

linear_classifier = LinearClassifier(ff_output_neurons, net)
print("training the final linear classifier")
linear_classifier.train(x,y,epochs)
```

داریم:

```
print('train accuracy:',
linear_classifier.test(x).eq(y).float().mean().item())


x_test, y_test = next(iter(test_loader))
x_test, y_test = x_test.cuda(), y_test.cuda()

print('test accuracy:',
linear_classifier.test(x_test).eq(y_test).float().mean().item())
```

```
train accuracy: 0.9170599579811096
```

```
test accuracy: 0.9170999526977539
```