שמות: ואדים ליטבינוב, ירון גפן

תעודות זהות: 314552365, 305641508

שיטות מתקדמות בעיבוד תמונה רפואי – תרגיל 2

2. לאחר ניסוי וטעייה אומנה רשת Autoencoder עם ההיפר פרמטרים הבאים שנבחרו:

קבוע למידה: 0.000001

epochs: 20מספר

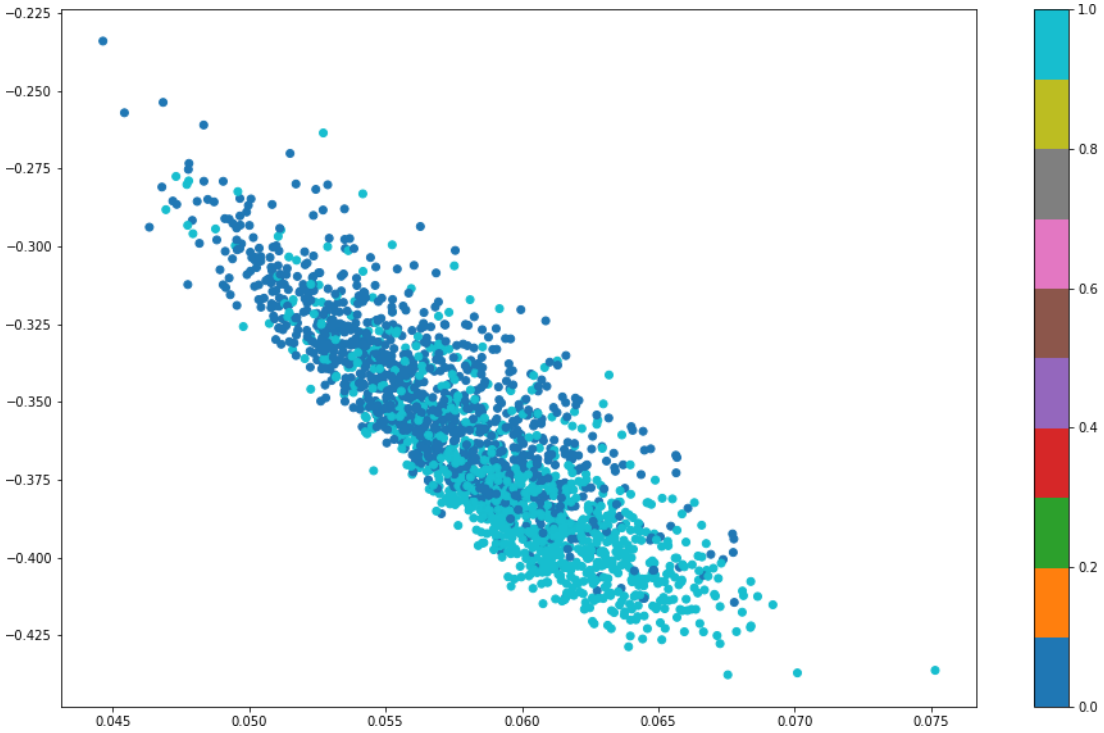
ארכיטקטורת הרשת: 5 שכבות ל-encoder בגדלים 2500, 1250, 625, 312, 156, 40. שכבת הקלט היא בגודל 2500 כיוון שכל התמונות שלנו הומרו לגודל אחיד של 50X50, וגודל השכבה הסמויה הינה 40 על מנת לא להקשות מדי על ה-t-sne בהמשך (האלגוריתם מעדיף ייצוג בגודל סביר מבחינת המימדים). ארכיטקטורת ה-decoder היא אותו דבר אך הפוך מבחינת סדר השכבות.

פונקציות אקטיבציה: relu, leaky relu, relu, relu, ל-encoder, ובאופן סימטרי אך הפוך ל-decoder.בוצע שימוש עם leaky relu כדי שלפחות בשכבה אחת יהיה גרדיאנט גם לערכים שליליים של
הנוירון.

בנוגע למאגר הנתונים עצמו, הוא נוצר עם פרמטר shuffle שמוסיף אקראיות ויכול לעזור, אך מצד שני יכול לגרום לחוסר עקביות בין ההרצות. בנוסף, כל התמונות הומרו לרזולוציה של 50X50 כאמור.

.4

כאשר מתבוננים על הפלט של t-sne למישור דו מימדי ניתן לראות שרוב הדוגמאות ללא הטפיל (צבע כחול) זכו לייצוג קטן יותר בציר ה-X וגדול יותר בציר ה-Y וההפך לדוגמאות שעם הטפיל (צבע תכלת). אמנם ההפרדה לא מושלמת אך רואים שניתן לנסות להעביר מישור מפריד בין המחלקות. כיוון שזהו מישור מפריד לינארית ולא פונקציות לא לינאריות מסובכות יותר, ככל הנראה גם שימוש באלגוריתמי למידת מכונה שהם לא רשתות נוירונים יספיקו לביצועים סבירים.



.6

כעת על הייצוג הקומפקטי של השכבה הסמויה נבנתה רשת עם פונקציית Loss מסוג Cross Entropy, עם ארכיטקטורה של 50, 25, 13, ושכבת פלט בגודל 1.

בנוסף, על מנת להימנע מהתאמת-יתר (overfit) הוגדר Dropout עם הסתברות של 0.2 שהרשת תתעלם מנוירון נתון בכל שכבה במהלך האימון.

כיוון שאנו בבעיית סיווג בינארית (רקמה עם טפיל או רקמה בריאה) בשכבה האחרונה יש פונקציית אקטיבציה מסוג סיגמואיד, וערך הסף שנבחר באופן טבעי הוא 0.5. אם הערך קטן מ-0.5 הפלט הוא 0, אחרת הפלט הוא 1. מספר ה-epochs נשאר 20.

Confusion matrix:

	predicted 1	predicted 0
actual 1	774	350
actual 0	222	646

לכן עבור קבוצת מבחן בגודל 1992, הדיוק הוא 71.285%.

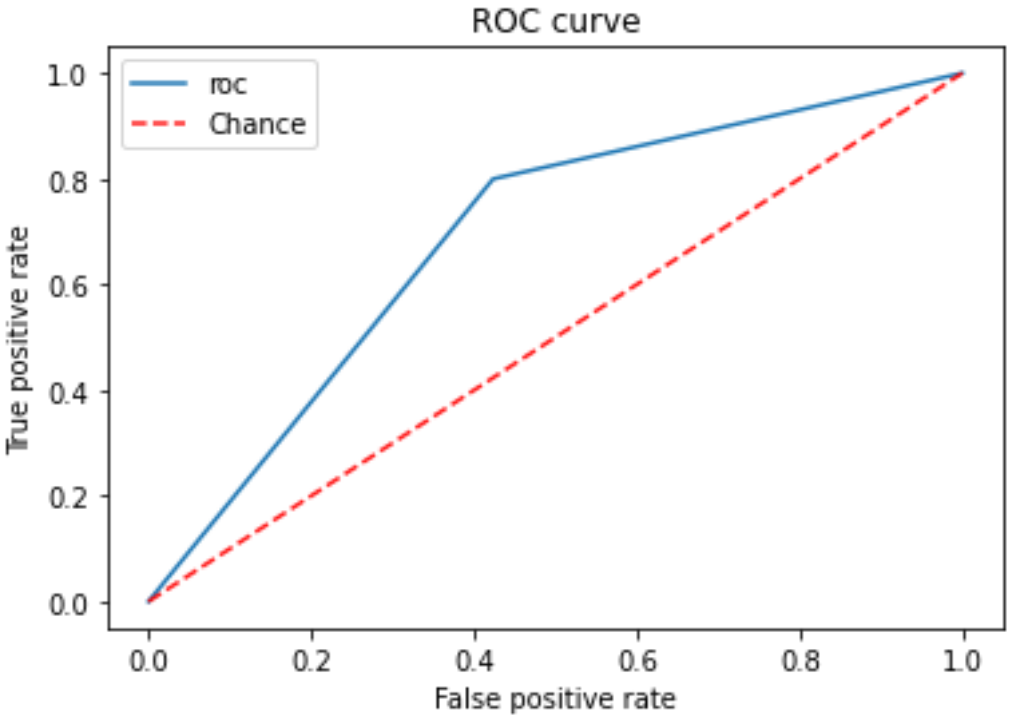
כאשר מתחשבים בכל ערכי הסף האפשריים לתיוג חיובי או שלילי (ערך הסף אשר היה לנו הוא

0.5) ניתן לבנות עקום ROC לרשת, שמעביר את הפשרה שיש בין להשתפר ב-TP לבין להיות

גרוע יותר ולעלות ב-FP. במודל אקראי לחלוטין השטח שמתחת לעקום יהיה 0.5, וככל

שהשטח מתחת לעקום ישאף ל-1, כך נדע שהמודל שלנו מוצלח יותר. עקום זה שימושי בכדי

לבחור ערך סף מוצלח למסווג, צריך לחפש מין "כתף" שהיא קרובה ל-0 בציר ה-X וכמה שיותר קרובה ל-1 בציר ה-Y. העקום שיצא לנו:



הקוד להלן:

```
pos_labels = torch.ones(NUM_IMAGES)
neg_labels = torch.zeros(NUM_IMAGES)
labels = torch.cat([pos_labels, neg_labels])
#print(labels)
dataset = TensorDataset(tensor_images, labels)
data = DataLoader(dataset=dataset, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)

device = 'cuda' if torch.cuda.is_available() else 'cpu'

class Encoder(nn.Module):
    def __init__(self, bottleneck_layer_dim):
        super(Encoder, self).__init()__
        self.linear1 = nn.Linear(1250 ,2500)
        self.linear2 = nn.Linear(625 ,1250)
        self.linear3 = nn.Linear(312 ,625)
        self.linear4 = nn.Linear(156 ,312)
        self.linear5 = nn.Linear(156, bottleneck_layer_dim)

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.linear1(x))
        x = F.leaky_relu(self.linear2(x))
        x = F.relu(self.linear3(x))
        x = F.relu(self.linear4(x))
#         print(self.linear5(x))
        return self.linear5(x)

class Decoder(nn.Module):
    def __init__(self, bottleneck_layer_dim):
        super(Decoder, self).__init()__
        self.linear1 = nn.Linear(bottleneck_layer_dim, 156)
        self.linear2 = nn.Linear(312 ,156)
        self.linear3 = nn.Linear(625 ,312)
        self.linear4 = nn.Linear(1250 ,625)
        self.linear5 = nn.Linear(2500 ,1250)

    def forward(self, z):
        z = F.relu(self.linear1(z))
        z = F.relu(self.linear2(z))
        z = F.leaky_relu(self.linear3(z))
        z = F.relu(self.linear4(z))
        z = self.linear5(z)
        return z.reshape((50 ,50 ,1 ,1-))


class Autoencoder(nn.Module):
    def __init__(self, bottleneck_layer_dim):
        super(Autoencoder, self).__init()__
        self.encoder = Encoder(bottleneck_layer_dim)
        self.decoder = Decoder(bottleneck_layer_dim)

    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z)

def train(autoencoder, data, epochs=EPOCHS):
    opt = torch.optim.Adam(autoencoder.parameters(), lr = LEARNING_RATE)
    for epoch in range(epochs):
        gen_loss = 0
        for x, y in data:
            x = x.to(device) # GPU
```

```
            opt.zero_grad()
            x_hat = autoencoder(x)
            loss = ((x - x_hat)**2).sum()
            gen_loss += loss
            loss.backward()
            opt.step()
        print(epoch, gen_loss)
    return autoencoder

autoencoder = Autoencoder(BOTTLENECK_LAYER_DIM).to(device) # GPU/CPU
print(autoencoder)
autoencoder = train(autoencoder, data)

def get_bottleneck(autoencoder, bottleneck_layer_dim, tsne_dims, data, num_batches=100/BATCH_SIZE):
    Z[] =
    Y[] =
    for i, (x, y) in enumerate(data):
        z = autoencoder.encoder(x.to(device))
        z = z.to('cpu').detach().numpy()
        for i, item in enumerate(z):
          Z.append(z[i].squeeze())
          Y.append(y[i].item())
    return Z, Y


Z, Y = get_bottleneck(autoencoder, BOTTLENECK_LAYER_DIM, TSNE_DIMS, data)
print(np.array(Z).shape)

def plot_latent(Z, Y, tsne_dims):
    tsne = TSNE(n_components=tsne_dims)
#    Z = np.append(Z, [Z0, 0], axis = 0)
#    plt.colorbar()
#    tsne_results = tsne.fit_transform(Z)
    df = pd.DataFrame(np.array(Z))
    df['y'] = Y
    print(np.any(np.isnan(Z)))
    print(np.all(np.isfinite(Z)))
    Z = np.nan_to_num(Z, copy=True)
    print(np.any(np.isnan(Z)))
    print(np.all(np.isfinite(Z)))
    tsne_results = tsne.fit_transform(Z)
    print(tsne_results.shape)
    plt.figure(figsize=(16,10))
    plt.scatter(Z[:, 0], Z[:, 1], c=Y, cmap='tab10')
    plt.colorbar()

plot_latent(Z, Y, TSNE_DIMS)

def plot_roc(tpr, fpr):
    plt.plot(fpr, tpr, label='roc', lw=1.5)
    plt.plot([0, 1], [0, 1], linestyle='--', lw=1.5, color='r,'
             label='Chance', alpha=.9(
 #    mean can't be calculated because of different shape for every fold
 #    fpr_mean = np.mean(np.array(fpr), axis=0)
 #    tpr_mean = np.mean(tpr, axis=0)
 #    plt.plot(fpr_mean, tpr_mean, label='mean', lw=2, color='red')
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.legend()
    plt.title('ROC curve')
#    plt.savefig('/home/ls/yarong/roc.jpeg')
    plt.show()
    plt.close()

 #classify data
```

```
LEARNING_RATE = 0.00001
class nn_Network(nn.Module):
    def __init__(self, encoder, input_dim, hidden_dim1, hidden_dim2, hidden_dim3, output_dim):
        super(nn_Network, self).__init()__
        self.encoder = encoder
        self.linear1 = nn.Linear(input_dim, hidden_dim1)
        self.linear2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.linear3 = nn.Linear(hidden_dim2, hidden_dim3)
        self.linear4 = nn.Linear(hidden_dim3, output_dim)
        self.dropout = nn.Dropout(0.2)
        self.act = nn.Sigmoid()

    def forward(self, x):
        x = F.leaky_relu(self.encoder(x))
        x = F.relu(self.linear1(x))
#         x = self.dropout(x)
        x = F.relu(self.linear2(x))
        x = F.relu(self.linear3(x))
        x = self.act(self.linear4(x))
        return x

network = nn_Network(autoencoder.encoder, BOTTLENECK_LAYER_DIM, 50, 25, 13, 1)
network.to(device)
optimizer = optim.Adam(network.parameters(), lr=LEARNING_RATE, eps = 0.0001)
network.train()
criterion = nn.BCELoss()
for epoch in range(EPOCHS):
    outputs[] =
    labels[] =
    running_loss = 0.0
    correct = 0
    for batch_num, (inputs, label) in enumerate(data, start=1):
        inputs, label = inputs.to(device), label.to(device)
        optimizer.zero_grad()
        output = network(inputs)
        for elem in label:
            labels.append(elem.item())
        loss = criterion(output, label.view(BATCH_SIZE,-1))
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        output = (output>0.5).float()
        for elem in output:
            outputs.append(elem.item())
        correct += (output[0] == label[0]).float().sum()
    tn, fp, fn, tp = confusion_matrix(labels, outputs).ravel()
    acc = 100*correct/(len(data.dataset))
    loss = running_loss
    print(f"epoch num: {epoch} - Train loss: {loss:.3f}, Train accuracy: {acc:.3f}%")
    fpr, tpr, thresholds = metrics.roc_curve(labels, outputs)
    auc = metrics.auc(fpr, tpr)
    print(auc)
    plot_roc(tpr,fpr)
```