به نام خدا

گزارش پروژه نهایی داده‌کاوی پیشرفته

مهدی فقهی

۴۰۱۷۲۲۱۳۶

توضیح مقاله و روش ریاضی :

بسیاری از سیستم‌ها فاقد توانایی شناسایی زمانی هستند که موارد پرت (به عنوان مثال، نمونه‌هایی که از توزیع داده‌های آموزشی متمایز هستند و در توزیع داده‌های آموزشی نمایش داده نمی‌شوند) به سیستم ارائه می‌شوند.

توانایی تشخیص نقاط پرت از اهمیت عملی برخوردار است زیرا می تواند به سیستم کمک کند در هنگام مواجهه با داده های غیرمنتظره به شیوه ای معقول رفتار کند. در این مقاله از مفهوم null space برای ادغام یک روش تشخیص نقاط پرت به طور مستقیم در یک شبکه عصبی که برای طبقه‌بندی استفاده می‌شود، استفاده کردیم . روش مقاله، به نام (Null Space Analysis (NuSA شبکه های عصبی، با محاسبه و کنترل بزرگی پیش بینی null space زمانی که داده ها از طریق شبکه منتقل می شوند، کار می کند سپس می‌توانیم مقداری را محاسبه کنیم که می‌تواند بین داده‌های عادی و غیرعادی تمایز قائل شود. در رویکرد NuSA، از null space مرتبط با ماتریس وزن یک لایه در ANN استفاده می کنیم تا همزمان با طبقه‌بندی با استفاده از همان شبکه، تشخیص نقاط پرت را انجام می‌دهیم. null space یک ماتریس به صورت زیر تعریف می شود:

$$\mathcal{N}(\boldsymbol{A}) = \{\boldsymbol{z} \in \mathcal{R}^n \,|\boldsymbol{A}\boldsymbol{z} = \boldsymbol{0}\},$$

null space ماتریس A، ناحیه ای از فضای ورودی است که به صفر نگاشت می شود. هر ماتریس وزن دارای یک null space مرتبط است، اگرچه برخی ممکن است خالی باشند .

برای وضوح، مفهوم null space ابتدا با استفاده از یک ANN ساده با ورودی های K و خروجی های M و باشرط K> M نشان داده شده است. سپس برای شبکه های چند لایه نشان داده شده است.

فرض کنید که $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ مجموعه ای از نمونه هایی است که از توزیع مشترک گرفته شده است. به این معنا که $\mathbf{x}_n \sim p_{\mathcal{D}}$ و آنکه یک شبکه f_X برای تقریب یک تابع همانند f به‌شکل $\mathbf{y}_n = f(\mathbf{x}_n)$ آموزش داده شده است. هنگامی که فقط یک لایه داریم $\mathbf{y}_n = W\mathbf{x}_n$ است وقتی که $K > M$، یک null space غیر معکوس‌پذیر با ابعاد K - M ساخته می‌شود که :

$$\forall \lambda > 0 \quad W(\mathbf{x}_n + \lambda \mathbf{z}) = W\mathbf{x}_n + \lambda W\mathbf{z} = W\mathbf{x}_n = \mathbf{y}_n.$$

به این معنی است که بی نهایت امکان برای نگاشت ورودی های ناشناخته به خروجی های به ظاهر معنادار وجود دارد.

در یک شبکه چند لایه، لایه های زیادی از ضرب ماتریس-بردار وجود دارد که می تواند به صورت بیان شود:

$$\begin{aligned}
\mathbf{x}_{n,1} &= \sigma_1(W_1\mathbf{x}_n) \rightarrow \mathbf{x}_{n,2} \\
\mathbf{x}_{n,2} &= \sigma_2(W_2\mathbf{x}_{n,1}) \rightarrow \cdots \mathbf{x}_{n,N_h} \\
\mathbf{x}_{n,N_h} &= \sigma_{N_H}(W_{N_H}\mathbf{x}_{n,N_H-1})
\end{aligned}$$

در اینجا N_H بیانگر تعداد لایه‌های مخفی شبکه و σ_h بیانگر تابع فعالیت است. فرض کنید یک داده غیر پرت مانند X وجود دارد و همچنین $\mathbf{z} \in \mathbb{R}_K$ و می‌دانیم برابر است با : $\mathbf{z}_h = \sigma_{h-1}(W_{h-1}\sigma(\cdots\sigma_1(W_1\mathbf{x})))$ ، اگر $\mathbf{z}_h \in \mathcal{N}(W_h)$ آنگاه $W_h\mathbf{z}_h = \mathbf{0}$ که در نتیجه عبارت مقابل نتیجه میدهد : $f_X(\mathbf{x}+\mathbf{z}) = f_X(\mathbf{x})$

بنابراین، هر نمونه ورود مانند z که تصویر معکوس یک نمونه مانند Zh از null space هر یک از ماتریس های وزن، Wh، را می توان به یک ورودی قانونی اضافه کرد، x و خروجی تغییر نخواهد کرد.

هدف این است که از null space برای شناسایی این نقاط پرت استفاده کنیم. رویکرد NuSA به حداکثر رساندن نمایش هر نمونه داده آموزشی در null space لایه‌های یک شبکه است. سپس، در طول آزمایش‌ها، می‌توان بزرگی پیش‌بینی بر روی null space را زیر نظر گرفت و هر نمونه‌ای را که دارای null space بزرگ است را به‌عنوان یک نقطه غیر پرت علامت گذاری کرد. ایده این است که همه چیز را در null space قرار دهیم به جز کلاس هایی که در مجموعه های آموزشی وجود دارد. این برای شبکه دشوار است زیرا با به حداکثر رساندن null space بیشتر داده ها از بین می روند.

به کمک عبارت مقابل می‌توانیم null space کل ماتریس‌های وزن در هر لایه شبکه عصبی را محاسبه کنیم:

$$\mathcal{N}_u\mathcal{SA} = \lambda \sum_{l \in L} \frac{\|\mathcal{P}(\boldsymbol{W}_l)\boldsymbol{x}_l\|}{\|\boldsymbol{x}_l\|}$$

$$\mathcal{P}(\boldsymbol{W}) = \boldsymbol{\mathcal{C}}(\boldsymbol{W})^T\boldsymbol{\mathcal{C}}(\boldsymbol{W})$$

که در آن $\boldsymbol{\mathcal{C}}(\boldsymbol{W})$ بیانگر space basis ماتریس $\mathbf{W}$ است . پس برای آموزش رابطه زیر را برای تابع ضرر می‌نویسیم که به دنبال مینیمایز کردن حاصل زیر هستیم که در اینجا L همان خطای آموزش ما است .

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}) + \lambda \sum_{l \in L} \frac{\|\mathcal{P}(\boldsymbol{W}_l)\boldsymbol{x}_l\|}{\|\boldsymbol{x}_l\|}$$

پس از آموزش به کمک رابطه بالا هنگام تست می‌توانیم از الگوریتم زیر استفاده کنیم که صرفا عبارت‌هایی را پیش‌بینی کند که جز دادگان‌پرت تشخیص داده‌نشوند .

Algorithm 1 Psuedocode for NuSA testing procedure.

for i < N samples **do**
  Compute forward pass to get output for sample i
  Compute NuSA for sample i
  if NuSA > threshold **then**
    Set output as index of max network output
  else
    Declare sample i as outlier
  end if
end for
Return: Outlier indicator and outlier class labels

پیاده‌سازی :

بعد از ساخت دیتاست برای آموزش و تست از Cifar 10 از آنجایی که هدف ما بررسی دقت این روش در حضور داده‌های پرت است پس از داده‌های Cifar 10 به ترتیب زیر مجموعه با برچسب‌های ۲ ،۳، ۴ و ۰۰۰ ۹ جدا می‌کنیم و هر کدام را هر بار به صورت مجزا آموزش می‌دهیم و بر روی دادگان تست، تست می‌کنیم و دقت لازم را بدست می‌آوریم .

```
from torch.utils.data import Subset
def create_subset(dataset, labels):
    indices = [idx for idx, label in enumerate(dataset.targets) if label in labels]
    subset = Subset(dataset, indices)
    return subset
```

```
# Specify the labels for the subset
subset_labels = [0, 1]
# Create the subset
subset = create_subset(train_dataset, subset_labels)
train_loader_2 = DataLoader(subset, batch_size=25, shuffle=True)
```

```
# Specify the labels for the subset
subset_labels = [0, 1,2]
# Create the subset
subset = create_subset(train_dataset, subset_labels)

train_loader_3 = DataLoader(subset, batch_size=25, shuffle=True)
```

```
# Specify the labels for the subset
subset_labels = [0, 1,2,3]
# Create the subset
subset = create_subset(train_dataset, subset_labels)
```

سپس برای آنکه از دادگان عکس خود featureهای لازم برای دسته‌بندی را استخراج کنیم از شبکه عمیق آموزش دیده resnet 50 استفاده می‌کنیم .

```
model = torchvision.models.wide_resnet50_2(pretrained=True)
model = model.to(device)
model.eval()
```

سپس لایه آخر دسته‌بندی آن را حذف می‌کنم و شبکه عصبی زیر را به آن برای دسته‌بندی اضافه می‌کنم.

```
model = nn.Sequential(*list(model.children())[:-1])
```

ate a new fully connected network for classification:

```
# Define the network class
class SimpleNetwork(nn.Module):
    def __init__(self, num_outputs):
        super(SimpleNetwork, self).__init__()

        self.input_layer = nn.Linear(2048, 64)
        self.hidden_layer = nn.Linear(64, 32)
        self.relu = nn.ReLU()
        self.output_layer = nn.Linear(32, num_outputs)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):

        x = x.to(self.input_layer.weight.device)  # Move
        x1 = self.input_layer(x)
        x1_relu  = self.relu(x1)
        x2  = self.hidden_layer(x1_relu)
        x2_relu  = self.relu(x2)
        x = self.output_layer(x2_relu)
        x = self.sigmoid(x)
        return x,x2_relu,x1_relu
```

همانطور که می‌بینید خروجی شبکه علاوه بر خروجی لایه آخر، خروجی relu نسبت به ورودی و خروجی relu از لایه میانه نیز است که این خروجی را برای محاسبه NuSA استفاده می‌نمایم .

```python
# Define the loss function with NuSA term
class NuSALoss(nn.Module):
    def __init__(self, lambda_value):
        super(NuSALoss, self).__init__()
        self.lambda_value = lambda_value

    def forward(self, outputs, targets, model, inputs):
        standard_loss = nn.CrossEntropyLoss()(outputs, targets)
        nusa_term = 0.0

        i = 0
        for name, param in model.named_parameters():
            if 'weight' in name:
                weight = param.data
                xl = inputs[i]  # Input sample to the layer
                xl = xl.to(device)


                i += 1
                Q, _  = torch.qr(weight.t())
                column_space_basis = Q.t()
                column_space_basis = column_space_basis.to(device)
                nusa_term += torch.norm(torch.matmul(torch.matmul(column_space_basis.t(),
                                                                  column_space_basis), xl.t())) / xl.norm()

        loss = standard_loss + self.lambda_value * nusa_term
        return loss
```

مطابق رابطه ریاضی که در قسمت قبل به آن پرداختیم ابتدا column space basis را برای هر کدام از ماتریس وزن‌های لایه‌های مختلف شبکه بدست می‌آوریم که در اینجا سه تا ماتریس وزن داریم که شامل ورودی، میانه و خروجی است و سپس حاصل ورودی به هر لایه را در آن ضرب می‌نمایم سپس از حاصل بدست آمده نرم میگیریم و حاصل را تقسیم بر نرم ورودی می‌کنیم و این کار را به ازای تمامی لایه‌های انجام می‌دهیم و در نهایت مقادیر بدست آمده را با هم جمع می‌کنیم حاصل NuSA اینگونه بدست خواهد آمد حاصل میزان Loss حاصل از خروجی نهایی را به کمک Cross Entropy حساب می‌کنیم و این میزان را به همراه حاصل ضرب ترم regularizer که میزان اهمیت میزان NuSA را مشخص میکند، جمع می‌نماییم و به عنوان میزان Loss شبکه برمی‌گردانیم .

سپس مدل را به شکل زیر آموزش می‌دهیم :

```python
# criterion = nn.CrossEntropyLoss()
def train(train_loader,network):

    optimizer = torch.optim.Adam(network.parameters(), lr=0.01)
    global loss_function
    # new_train_loader = make_train_loader_from_number_class_we_want(number_of_known_classes)
    for epoch in range(20):
        print(epoch)
        for images, labels in train_loader:
            images = images.to(device)
            labels = labels.to(device)

            # Forward pass through the WideResNet model
            with torch.no_grad():
                features = model(images)

            # Flatten the features
            features = features.view(features.size(0), -1)
            # Forward pass through the classifier
            outputs,relue_x2,relue_x1 = network(features)
            inputs = [features,relue_x1,relue_x2]
            # Move input and target tensors to the same device
            outputs = outputs.to(device)
            loss = loss_function(outputs, labels,network,inputs)
            # Backward and optimize
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        print(f'loss: {loss}')
```

در اینجا همانند توضیح داده شده در قسمت قبل میزان Loss را محاسبه می‌کنیم به ازای batch های ۲۵ تایی در دیتا لودر ساخته شده در مرحله قبل ابتدا داده ها را به مدل resnet داده ویژگی‌های مورد نیاز استخراج می‌شود سپس به کمک classifier به هر کدام برچسب را انتصاب می‌دهیم و این کار را به تعداد ذکر شده انجام می‌دهیم .

پس پایان آموزش دیتاست test که شامل دادگان مختلف از جمله دادگان پرت است را به مدل می‌دهیم و مدل ما ضمن آنکه دادگان پرت را باید شناسایی کند ، دادگان غیر پرت نیز باید برچسب‌شان را تمیز دهد و برچسب درست را اعلام کند .

```python
def compute_nusa(model,inputs):
    nusa = 0.0
    i = 0
    for name, param in model.named_parameters():
        if 'weight' in name:
            weight = param.data
            xl = inputs[i]
            xl = xl.to(device)
            i += 1
            # Reshape the weight tensor
            Q,_ = torch.qr(weight.t())
            column_space_basis = Q.t()
            column_space_basis = column_space_basis.to(device)
            nusa += torch.norm(torch.matmul(torch.matmul(column_space_basis.t(),
                                                         column_space_basis), xl.t())) / xl.norm()

    return nusa
```

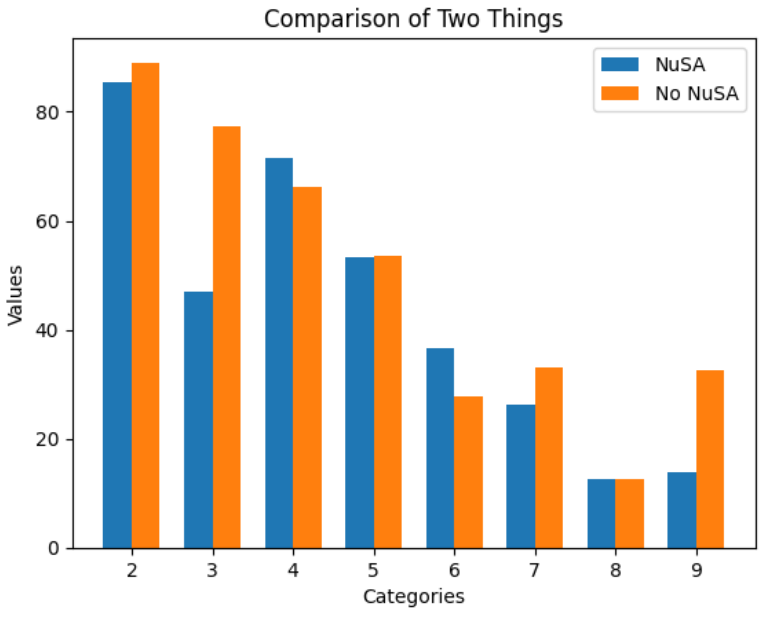
```python
def nusa_testing(data_loader, classifier, threshold):
    outlier_indicator = []
    outlier_class_labels = []
    global model
    classifier.eval()  # Set the model to evaluation mode

    with torch.no_grad():
        for inputs, labels in data_loader:
            # print(len(inputs))
            inputs = inputs.to(device)
            labels = labels.to(device)

            features = model(inputs)

            # Flatten the features
            features = features.view(features.size(0), -1)


            outputs,relue_x2,relu_x1 = classifier(features)  # Compute forward pa
            nusa = compute_nusa(classifier,[features,relu_x1,relue_x2])  # Comput

            if nusa > threshold:
                _, predicted = torch.max(outputs, 1)
                outlier_indicator.append(False)
                outlier_class_labels.append(predicted.item())
            else:
                outlier_indicator.append(True)
                outlier_class_labels.append(None)

    return outlier_indicator, outlier_class_labels
```

در این قسمت ابتدا همانند روش قبل ابتدا میزان NuSA را حساب می کنیم اگر این میزان بیشتر از میزان threshold همانند آنچه در سود و کد مربوطه در قسمت قبل آمده بود این داده جز مجموعه دادگان غیر پرت است و Label آن رابه کمک مدل پیش‌بینی می کنیم در غیر این صورت جز دادگان پرت ما است و پیش‌بینی درباره آن انجام نمی‌دهیم .

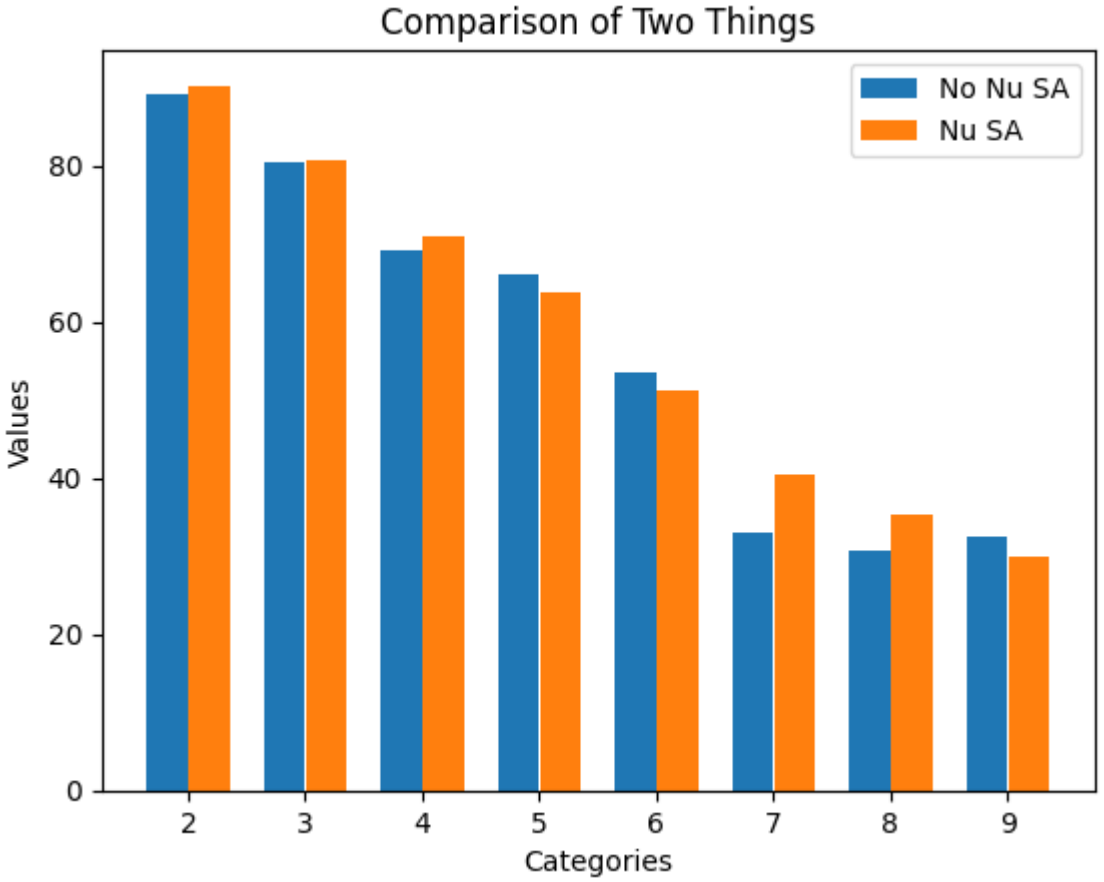
```
def test(test_loader,network,list_of_acceptable_label):

    outlier_indicator, outlier_class_labels = nusa_testing(test_loader,network,0.1)

    correct = 0
    total = 0
    for outlier_indi,outlier_labels,test in zip(outlier_indicator, outlier_class_labels,test_loader):

        if test[1] in list_of_acceptable_label :
          total += 1

        if not outlier_indi:
            if outlier_labels == test[1]:
              correct += 1



    accuracy = 100 * correct / total
    print(f"Accuracy: {accuracy}%")
    return accuracy
```

برای آزمون مدل نیز در اینجا ابتدا تمامی دادگان تست را به مدل می‌دهیم اگر دادگان تست جز موارد باشد که باید درباره آن پیش بینی کنیم به دادگان total اضافه می کنیم و اگر درست پیش‌بینی درست کرد Label دادگان را به تعداد پیش‌بینی درست نیز یک عدد اضافه می کنیم و درنهایت براساس این معیار میزان دقت مدل را بررسی می کنیم.

در نهایت حاصل زیر را بدست آوردم .



پس از کمی تلاش برای پیدا کردن نتایج بهتر برای Nu SA به نتیجه زیر رسیدم .



دراینجا مدل اصلی را نسبت به مدلی سنجدیده‌ایم که همانند روش ما آموزش دیده است به غیر از آنکه از ترم NuSA استفاده نکرده است و سپس در هنگام تست با دادگان پرت روبرو نشده است . همانطور که می‌بینید دو مدل دقت نزدیک به یکدیگر دارند که نشان می‌دهد مدل ما علاوه بر اینکه توانسته در فضای حالت دادگان پرت را تشخیص دهد از میان دادگان غیر پرت با دقت تقریبا مناسبی توانسته است پیش‌بینی را انجام دهد . با کمی تغییر در thershold و تغیری پارامترهای اولیه مدل همانطور که می‌بینید به قدرت برابر یا بهتر از مدل قبلی رسید البته در حالت 8 مدل بدون استراتژی را دوباره آموزش دادم .