بسمه تعالی

گزارش پروژه نهایی شبکه عصبی درس هوش مصنوعی

سینا مظاهری

بخش اول)

اولین تابعی که برای یادگیری شبکه عصبی مورد نظر انتخاب شده؛ سهمی است که فرم ریاضی آن به شکل می باشد.

$$y = x^2$$

پس از بررسی شرایط مساله و از آنجایی که مساله ما رگرسیون است؛ تصمیم گرفته شد که از یک شبکه عصبی تمام متصل 4 لایه استفاده شود. این شبکه عصبی از توابع فعال سازی SELU ، SELU ، Sigmoid (به ترتیب) پشتیبانی می کند. و هم چنین پس از عبور داده از هر تابع، یک لایه BatchNormalaizer1D بر روی آن اعمال می شود که پارامتر هایش قابل یادگیری می باشند. این سبب می شود که بیش برازش شدن شبکه جلو گیری شود.

```
self.list_of_layers.append(nn.Sequential(
    nn.Linear(input_size, 758),
    nn.SELU(),
    nn.BatchNorm1d(758),
    nn.Linear(758, 500),
    nn.SELU(),
    nn.BatchNorm1d(500),
    nn.Linear(500, 256),
    nn.Sigmoid(),
    nn.BatchNorm1d(256),
    nn.Linear(256, 1),
    nn.BatchNorm1d(1),
))
```

همانطور که مشخص است ابتدا یک لایه 758×1 و سپس 512×758 بعد 256×512 و در نهایت هم 1×256 استفاده شده است. ترتیب نزولی تعداد نورون ها آن هم به شکل توان های 2 تاثیر بسزایی در دقت شبکه داشته است. پس از طراحی شبکه نوبت به طراحی فرآیند یادگیری می رسد. ابتدا به اندازه 30000 نقطه در بازه $[-4\pi, 4\pi]$ اعدادی تولید می کنیم و سپس تابع را به ازای این مقادیر حساب می نماییم.

```
x = torch.reshape(torch.linspace(-4 * torch.pi, 4 * torch.pi, 30000), (30000, 1))
y = torch.pow(x, 2)
```

حال پس از آن هم x ها و هم y ها را به روش امتیاز z نرمال می کنیم. یعنی :

$$y_{normalized} = \frac{y - \bar{y}}{\boldsymbol{\sigma}_y}$$

$$x_{normalized} = \frac{x - \bar{x}}{\boldsymbol{\sigma}_x}$$

شکل آن به صورت زیر می باشد:

```
ds = PrepareData(x, y)
```

```
class PrepareData(Dataset):

    def __init__(self, X, y):
        self.X = X
        self.y = y
        self.mu_x = torch.mean(X)
        self.std_x = torch.std(X)
        self.mu_y = torch.mean(y)
        self.std_y = torch.std(y)
        self.X = (self.X - self.mu_x) / self.std_x
        self.y = (self.y - self.mu_y) / self.std_y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

پس از آن که مجموعه داده ما حاضر شد حالا نوبت به بارگذاری و تقسیم بندی تصادفی مجموعه داده به دو دسته train و test می باشد که به شکل زیر آمده است. هر یک را به دسته های 64 تایی تقسیم بندی می کنیم و تنسور های ما برای آموزش به شکل 64×1 خواهند شد. خروجی درست نیز به همین شکل است و برای داده های تست هم همین شکلی می شود. و مجموعه آموزشی هم 30 درصد مقدار کل خواهد بود.

```
train, test = train_test_split(list(range(x.shape[0])), test_size=.3)
ds = PrepareData(x, y)
train_set = DataLoader(ds, batch_size=64,
                       sampler=SubsetRandomSampler(train))
test_set = DataLoader(ds, batch_size=64,
                      sampler=SubsetRandomSampler(test))
```

حالا فرآیند یادگیری را دنبال می کنیم این بدان شرح است که ابتدا مدل خود را می سازیم و میانگین مربعات خطا را برای سنجش خطای آن بر می گزینیم و الگوریتم بهینه سازی انتخابی ما برای مدل RMSprop با نرخ یادگیری 0.001 خواهد بود. (بقیه پارامتر های این الگوریتم پیش فرض است)تعداد 500 دور حلقه یادگیری شکل بهینه را برای ما تولید می کند. در هر بار یک دسته 64 تایی انتخاب می شود و پس از عبور از مدلی که پارامتر های وزنش با مقادیر نرمال استاندارد مقدار دهی اولیه شده اند؛ مقادیر خروجی حساب می شود و سپس میزان خطایش سنجیده شده و پارامتر های وزنی مدل یاد گرفته می شوند.

```
if tag:
    model = NeuralNetwork(x.shape[1], noise)
    optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
    num_epochs = 500
    ##
    all_losses = []
    for e in range(num_epochs):
        batch_losses = []

        for idx, (Xb, yb) in enumerate(train_set):
            _X = Xb.float()
            _y = yb.float()
            pred = model(_X)
            loss = criterion(pred, _y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            batch_losses.append(loss.item())
            all_losses.append(loss.item())

        mbl = torch.mean(torch.sqrt(torch.tensor(batch_losses)))

        if e % 5 == 0:
            print("Epoch [{}/{}], Batch loss: {}".format(e, num_epochs, mbl))
    torch.save(model, 'model_' + str(model_num) + '.pth')
return model, criterion, test_set, ds
```

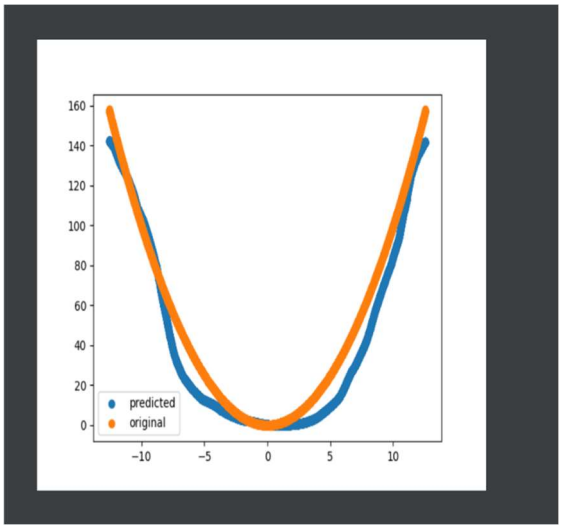
پس از تمرین دادن نقاط بر روی مدل مورد نظر، نوبت به ترسیم خروجی است که در ابتدا مدل از حالت ارزیابی خارج می شود و سپس از روی داده های تست مدل شروع به تست شدن می شود.

```
for _X, _y in test_set:
    _X = _X.float()
    _y = _y.float()
    if noise_or_not:
        noisy = transforms.Compose([AddGaussianNoise_1(noise_or_not[0], noise_or_not[1])])(_X)
        test_pred = model(noisy)
    else:
        test_pred = model(_X)
    test_loss = criterion(test_pred, _y)
    test_batch_losses.append(test_loss.item())
    print("Batch loss: {}".format(test_loss.item()))
    if not model_num == 6:
        if _X.shape[0] == 40 and _y.shape[0] == 40:
            test_x = torch.hstack((torch.reshape(_X, (40,)) * ds.std_x + ds.mu_x, test_x))
            test_pred_y = torch.hstack((torch.reshape(test_pred, (40,)) * ds.std_y + ds.mu_y, test_pred_y))
            real_y = torch.hstack((torch.reshape(_y, (40,)) * ds.std_y + ds.mu_y, real_y))
            if noise_or_not:
                noisal = torch.hstack((torch.reshape(noisy, (40,)) * ds.std_y + ds.mu_y, noisal))
        else:
            test_x = torch.hstack((torch.reshape(_X, (64,)) * ds.std_x + ds.mu_x, test_x))
            test_pred_y = torch.hstack((torch.reshape(test_pred, (64,)) * ds.std_y + ds.mu_y, test_pred_y))
            real_y = torch.hstack((torch.reshape(_y, (64,)) * ds.std_y + ds.mu_y, real_y))
            if noise_or_not:
                noisal = torch.hstack((torch.reshape(noisy, (64,)) * ds.std_y + ds.mu_y, noisal))
if not model_num == 6:
    plt.scatter(test_x.detach(), test_pred_y.detach())
    plt.scatter(test_x.detach(), real_y.detach())
    if noise_or_not:
        plt.scatter(test_x.detach(), noisal.detach())
```

نتیجه برای سهمی به شکل زیر در می آید:

```
x = torch.reshape(torch.linspace(-4 * torch.pi, 4 * torch.pi, 30000), (30000, 1))
y = torch.pow(x, 2)
tup_1 = train(x, y, 2, None, False)
predict(torch.load("model_1.pth"), tup_1[1], tup_1[2], tup_1[3], 1, None)
```

```
Batch loss: 0.07074520736932755
Batch loss: 0.05060325562953949
Batch loss: 0.057494401931762695
Batch loss: 0.058407288044691086
Batch loss: 0.052889250218868256
Batch loss: 0.05006149411201477
Batch loss: 0.05184453725814819
Batch loss: 0.06921949237585068
Batch loss: 0.0707537829875946
Batch loss: 0.06498177349567413
Batch loss: 0.05181141942739487
Batch loss: 0.05330313742160797
Batch loss: 0.06926220655441284
Batch loss: 0.05329861119389534
Batch loss: 0.06607311218976974
Batch loss: 0.07024374604225159
Batch loss: 0.049983277916908264
Batch loss: 0.05836079642176628
Batch loss: 0.06345600634813309
Batch loss: 0.05236855521798134
```

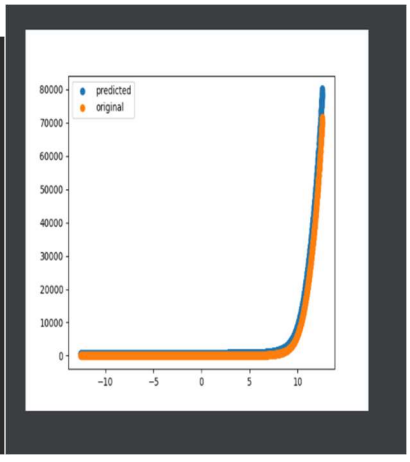


برای تابع زیر نتیجه در پایینش آمده و معماری شبکه و فرآیند یادگیری تغییر نکرده است.

$$y = \frac{e^x}{\sin x + 4}$$

```python
x = torch.reshape(torch.linspace(-4 * torch.pi, 4 * torch.pi, 30000), (30000, 1))
y = torch.exp(x) / (torch.sin(x) + 4)
tup_2 = train(x, y, 2, None, False)
predict(torch.load("model_2.pth"), tup_2[1], tup_2[2], tup_2[3], 2, None)
```

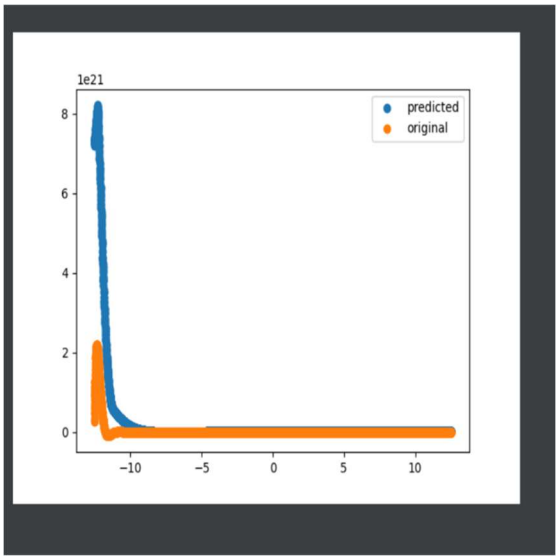
```
Batch loss: 0.025525929406285286
Batch loss: 0.05568686127662659
Batch loss: 0.07183945924043655
Batch loss: 0.02207234315574169
Batch loss: 0.0493844673037529
Batch loss: 0.0529182031750679
Batch loss: 0.0481410026550293
Batch loss: 0.03170832619071007
Batch loss: 0.028855014592409134
Batch loss: 0.04206221550703049
Batch loss: 0.02978157065808773
Batch loss: 0.05771335959434509
Batch loss: 0.011862746439874172
Batch loss: 0.04409586265683174
Batch loss: 0.025086937472224236
Batch loss: 0.03744703531265259
Batch loss: 0.048535291105508804
Batch loss: 0.02563730999827385
Batch loss: 0.0514412522315979
Batch loss: 0.019713876768946648
Batch loss: 0.015572953969240189
Batch loss: 0.045547641813755035
Batch loss: 0.02901790663599968
Batch loss: 0.019667886197566986
Batch loss: 0.049156285822391 51
Batch loss: 0.032543569803237915
Batch loss: 0.023992475122213364
Batch loss: 0.051739536225795746
Batch loss: 0.03294725343585014
Batch loss: 0.049405899889833995
```

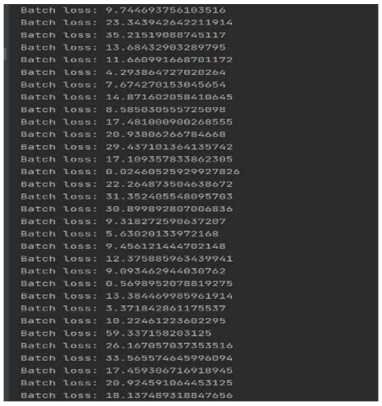


برای تابع زیر نیز، مقدار آزمایش شده است:

$$y = e^{-4x}\sin(-4x)$$

```
x = torch.reshape(torch.linspace(-4 * torch.pi, 4 * torch.pi, 30000), (30000, 1))
y = torch.exp((-4) * x) * (torch.sin(4 * x))
tup_3 = train(x, y, 3, None, False)
predict(torch.load("model_3.pth"), tup_3[1], tup_3[2], tup_3[3], 3, None)
```

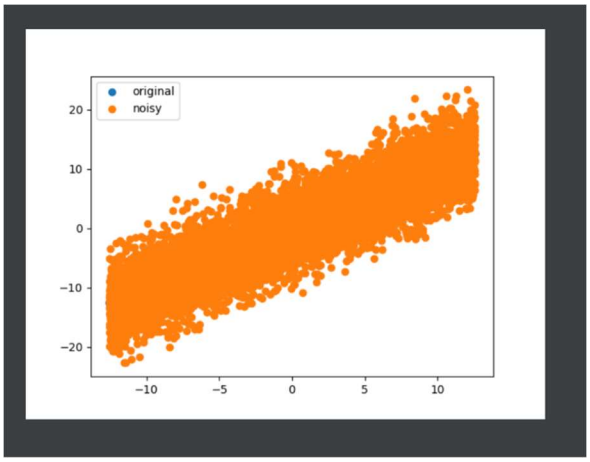


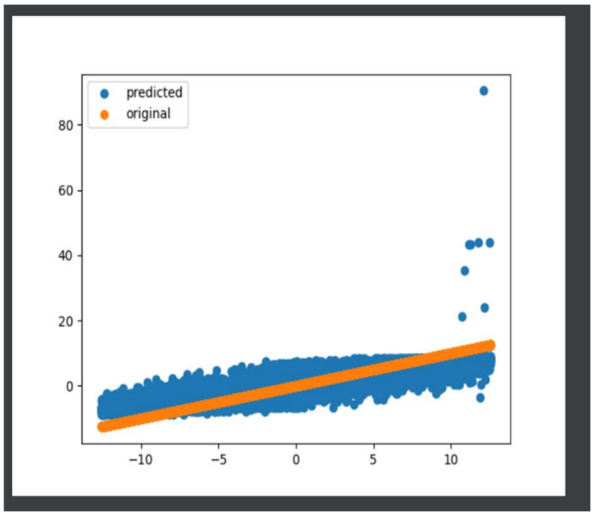


بخش دوم)

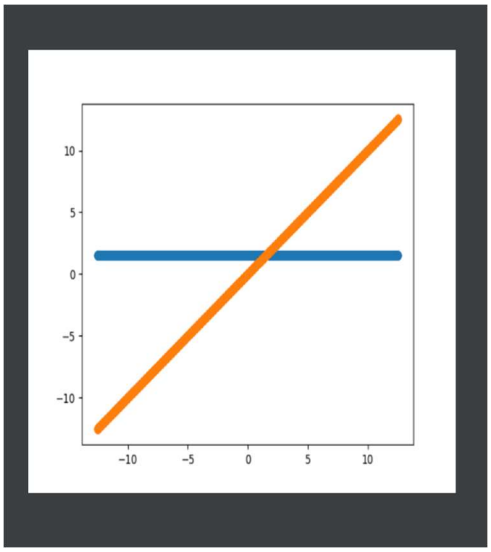
برای نمایش نویز نیز دو آزمایش زیر که به مقدار ورودی نویز اضافه می کند و شبکه یاد می گیرد آن را دی نویز کند؛ نیز انجام شده است. لازم به ذکر است که نویز آزمایش اول به شکل نرمال با میانگین صفر و انحراف از معیار 0.5 صورت گرفته است و آزمایش دوم با میانگین 10 و انحراف از معیار 20 صورت گرفته است.

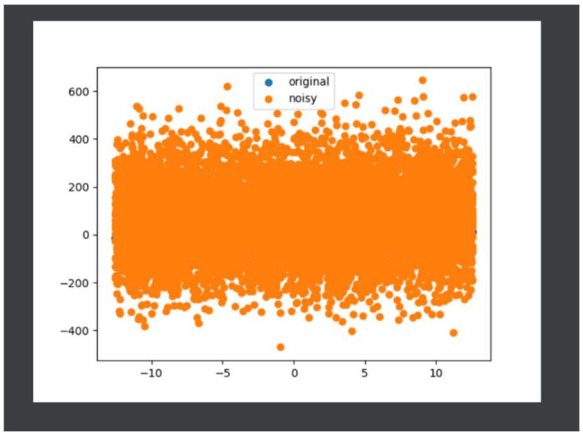
آزمایش اول





آزمایش دوم:





بخش سوم)

تابع دو متغیره که برای یادگیری انتخاب شده دارای فرم ریاضی زیر است و نتایج آن در تست زیر آمده و نمایش گرافیکی ندارد.

$$y = x_1^2 + x_2$$

```
Batch loss: 0.05882998928427696
Batch loss: 0.05745626986026764
Batch loss: 0.08977963775396347
Batch loss: 0.07232993841171265
Batch loss: 0.07318319380283356
Batch loss: 0.05510503053665161
Batch loss: 0.06382540613412857
Batch loss: 0.07030857354402542
Batch loss: 0.07373107224702835
Batch loss: 0.13182319700717926
Batch loss: 0.06156344339251518
Batch loss: 0.07727517932653427
Batch loss: 0.0639292299747467
Batch loss: 0.06998981535434723
Batch loss: 0.05569326877593994
Batch loss: 0.12593591213226318
Batch loss: 0.10798700153827667
Batch loss: 0.09624805301427841
Batch loss: 0.08946161717176437
Batch loss: 0.08764185011386871
Batch loss: 0.044410936534404755
Batch loss: 0.07097683101892471
Batch loss: 0.026434483006596565
Batch loss: 0.051706235855817795
```

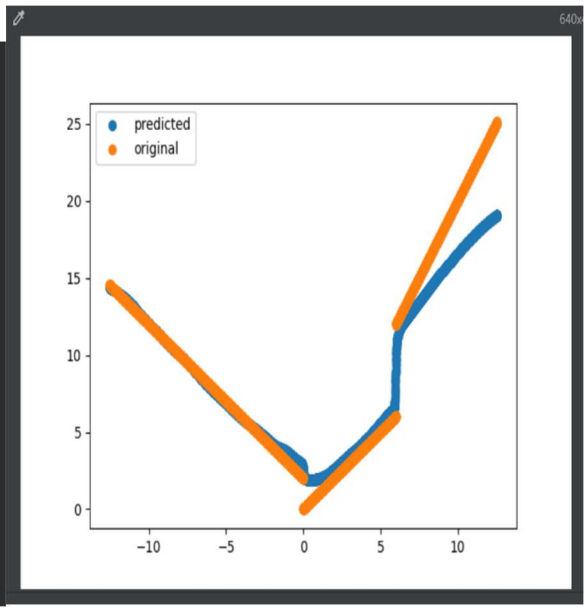
بخش چهارم)

تابعی که به شکل قطعه ای طراحی شده و شامل نقاط پرش است شامل ضابطه زیر است که یاد گرفته شده همچنین نقاط پرش سبب شده است تا با شیب بسیار زیاد مقدار پیش بینی شده جهش کند.

$$y = \begin{cases} -x + 2 & x < 0 \\ x & 0 < x < 6 \\ 2x & 6 < x \end{cases}$$

```python
x = torch.reshape(torch.linspace(-4 * torch.pi, 4 * torch.pi, 30000), (30000, 1))
y = torch.empty(30000, 1)
for i in range(len(x)):
    if x[i, 0] < 0:
        y[i, 0] = (-1) * x[i, 0] + 2
    elif 0 < x[i, 0] < 6:
        y[i, 0] = x[i, 0]
    else:
        y[i, 0] = 2 * x[i, 0]
tup_4 = train(x, y, 4, None, False)
predict(torch.load("model_4.pth"), tup_4[1], tup_4[2], tup_4[3], 4, None)
```

```
Batch loss: 0.030919475480914116
Batch loss: 0.06983457505702972
Batch loss: 0.046463605016469955
Batch loss: 0.06226425990462303
Batch loss: 0.08882167935371399
Batch loss: 0.06397577375173569
Batch loss: 0.05970914289355278
Batch loss: 0.07904647290706635
Batch loss: 0.08649288862943649
Batch loss: 0.059534311294555664
Batch loss: 0.06929558515548706
Batch loss: 0.06688889116048813
Batch loss: 0.08722212165594101
Batch loss: 0.07069297134876251
Batch loss: 0.0699334368109703
Batch loss: 0.05680086091160774
```



بخش پنجم)

در این بخش که به کاربرد طبقه بندی پرداخته می شود از پایگاه داده MNIST استفاده شده که شامل عکس های 28×28 پیکسل می باشد و نمایانگر اعداد 0 تا 9 هستند. در واقع مساله عکس ها را به عنوان ماتریس خاکستری می گیرد و طبقه بندی شده شماره آن را اعلام می کند. معماری شبکه مانند قبل است فقط به اول آن یک لایه Flatten اضافه شده است تا آن را از تنسوری به ابعاد $n \times 28 \times 28$ (برای n عکس) به تنسوری به ابعاد $n \times 784$ تبدیل کند و بعد آن را وارد شبکه می کند تا آن را تمرین کند همچنین به جای

یک خروجی حقیقی مقدار یک خروجی 10 تایی از لایه Softmax آورده می شود که احتمال نظیر هر کلاس را نشان دهد و در نهایت هم آرگومان آنی که احتمال بیشتری دارد به عنوان کلاس هدف بازگردانده می شود.تابع هزینه در این قسمت آنتروپی متقاطع (CrossEntropy) است که میزان تمایز دو توزیع احتمال حقیقی و ساخته شده را ارزیابی می کند. . عکس ها با استفاده از تکنیک CrossValidate که در انتهای حلقه batch آمده اند؛ ارزیابی می شوند. مقادیر batch ، 50 تایی هستند. لازم به ذکر است که عکس ها قبل از انجام عملیات نرمال می شوند. (n = 50)

```
transformation = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
train_set = torchvision.datasets.MNIST(root='./data', train=True, download=False, transform=transformation)
train_set, valid_set = torch.utils.data.random_split(train_set, lengths=[50000, 10000], generator=torch.Generator().manual_seed(42))
test_set = torchvision.datasets.MNIST(root='./data', train=False, download=False, transform=transformation)
train_loader = DataLoader(train_set, batch_size=50, shuffle=True, num_workers=2)
valid_loader = DataLoader(valid_set, batch_size=50, shuffle=True, num_workers=2)
testloader = DataLoader(test_set, batch_size=30, shuffle=False, num_workers=2)
model = None
if tag:
    model = NeuralNetwork(False, 0, 0)
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
    num_epochs = 500
    best_vloss = 1_000_000
    for epoch in range(num_epochs):
        model.train(True)
        running_loss = 0.
        last_loss = 0.
        for idx, data in enumerate(train_loader, 0):
            Xb, yb = data
            optimizer.zero_grad()
            pred = model(Xb)
            loss = criterion(pred, yb)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if idx % 30 == 29:
                last_loss = running_loss / 30
                print(f'[{epoch + 1}, {idx + 1:5d}] loss: {last_loss:.3f}')
                running_loss = 0.0
        model.train(False)
        running_vloss = 0.0
```

تابع پیش بینی نیز تست های آزمایشی را طبقه بندی می کند:

```python
def predict(model, test_loader):
    classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
    model.eval()
    correct_pred = {classname: 0 for classname in classes}
    total_pred = {classname: 0 for classname in classes}
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predictions = torch.max(outputs, 1)
            for label, prediction in zip(labels, predictions):
                if label == prediction:
                    correct_pred[classes[label]] += 1
                total_pred[classes[label]] += 1
    for classname, correct_count in correct_pred.items():
        accuracy = 100 * float(correct_count) / total_pred[classname]
        print(f'Accuracy for class: {classname:5d} is {accuracy:.1f} %')
```

عکس زیر درصد دقت درستی طبقه بندی هر کلاس را نشان می دهد.

```
C:\Users\Sina\Desktop\AI\venv\Scripts\pyth
Accuracy for class:     0 is 98.2 %
Accuracy for class:     1 is 96.9 %
Accuracy for class:     2 is 97.0 %
Accuracy for class:     3 is 96.1 %
Accuracy for class:     4 is 98.1 %
Accuracy for class:     5 is 94.6 %
Accuracy for class:     6 is 96.0 %
Accuracy for class:     7 is 96.0 %
Accuracy for class:     8 is 95.7 %
Accuracy for class:     9 is 95.3 %
```

بخش ششم)

حال برای Denoise عکس های این پایگاه داده معماری شبکه عصبی را تغییر می دهیم یعنی آن را به شکل زیر در می آوریم که شامل دو بخش انکدر و دیکدر می باشد. ابتدا انکدر ، عکس را فشرده می کند و خصوصیات اصلی آن را استخراج می کند و سپس دیکدر آن را به حالت اولیه بر می گرداند. لایه اول این شبکه شامل یک لایه نویز است که پس از نرمال شدن عکس ها به آن ها اعمال می شود. این نویز از توزیع نرمال با میانگین و

انحراف از معیار آمده در 3 آزمایشی که بعدا خواهیم گفت؛ بدست می آید. ابتدا دو لایه را اینجا نمایش می دهیم:

```
)
self.decoder = nn.Sequential(
    nn.Conv2d(2, 2, (3, 3), padding=1),
    nn.BatchNorm2d(2),
    nn.ReLU(),
    nn.Conv2d(2, 2, (3, 3), padding=1),
    nn.BatchNorm2d(2),
    nn.ReLU(),
    nn.Conv2d(2, 2, (3, 3), padding=1),
    nn.BatchNorm2d(2),
    nn.ReLU(),
    nn.Conv2d(2, 1, (3, 3), padding=1)
)
self.list_of_layers.append(self.encoder)
self.list_of_layers.append(self.decoder)
```

```
if denoising:
    self.encoder = nn.Sequential(
        AdditiveGausNoise(mean, std),
        nn.Conv2d(1, 2, (3, 3), padding=1),
        nn.BatchNorm2d(2),
        nn.ReLU(),
        nn.Conv2d(2, 2, (3, 3), padding=1),
        nn.BatchNorm2d(2),
        nn.ReLU(),
        nn.Conv2d(2, 2, (3, 3), padding=1),
        nn.BatchNorm2d(2),
        nn.ReLU(),
        nn.Conv2d(2, 2, (3, 3), padding=1)
    )
```

این لایه ها ترکیبی از کانوولشن 2 بعدی و رلو و نرمالسازی دسته ای می باشند. تا ویژگی های عکس را خلاصه کنند و مقدار نویز را از آن حذف کنند. نویز گاوسی هم برای داده های تست و تمرینی به دو شکل جدا پیاده سازی شده هم چنین کلاس AutoEncoder کل مجموعه داده را با خودش متناظر می کند تا به ازای هر عکس مقدار حقیقی اش را داشته باشیم. سایر موارد همان دسته بندی به مجموعه داده های آموزشی و آزمایشی است که نرمال شده اند.

```
def addNoise(x, mean, std):
    return x + torch.distributions.Normal(mean, std).sample(sample_shape=torch.Size(x.shape))


class AdditiveGausNoise(nn.Module):
    def __init__(self, mean, std):
        super().__init__()
        self.mean = mean
        self.std = std

    def forward(self, x):
        if self.training:
            return addNoise(x, self.mean, self.std)
        else:
            return x


class AddGaussianNoise_1(object):
    def __init__(self, mean=0., std=1.):
        self.std = std
        self.mean = mean

    def __call__(self, tensor):
        return tensor + torch.randn(tensor.size()) * self.std + self.mean

    def __repr__(self):
        return self.__class__.__name__ + '(mean={0}, std={1})'.format(self.mean, self.std)
```

```python
class AutoEncodeDataset(Dataset):
    def __init__(self, dataset):
        self.dataset = dataset

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        x, y = self.dataset.__getitem__(idx)
        return x, x
```

فرآیند یادگیری به شکل زیر می شود.

```python
if tag:
    model = NeuralNetwork(True, denoising[0], denoising[1])
    criterion = nn.MSELoss()
    optimizer = torch.optim.RMSprop(model.parameters(), lr=0.01)
    model = model.train()
    epochs = 40
    start = time.time()
    for epoch in tqdm(range(epochs), desc="Epoch", disable=disable_tqdm):
        model = model.train()
        running_loss = 0.0
        y_true = []
        y_pred = []
        for inputs, labels in tqdm(trainloader, desc="Train Batch", leave=False, disable=disable_tqdm):
            batch_size = labels.shape[0]
            optimizer.zero_grad()
            y_hat = model(inputs)
            loss = criterion(y_hat, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            running_loss = 0
            running_loss += loss.item() * batch_size
            if len(score_funcs) > 0:
                labels = labels.detach().cpu().numpy()
                y_hat = y_hat.detach().cpu().numpy()
                for i in range(batch_size):
                    y_true.append(labels[i])
                    y_pred.append(y_hat[i, :])
        end = time.time()
        total_train_time += (end - start)
        results["epoch"].append(epoch)
        results["total time"].append(total_train_time)
        results["train loss"].append(running_loss)
        y_pred = np.asarray(y_pred)
        if len(y_pred.shape) == 2 and y_pred.shape[1] > 1:
            y_pred = np.argmax(y_pred, axis=1)
        for name, score_func in score_funcs.items():
            results["train " + name].append(score_func(y_true, y_pred))
    torch.save(model, 'mod' + str(num) + '.pth')
return model, testdata
```

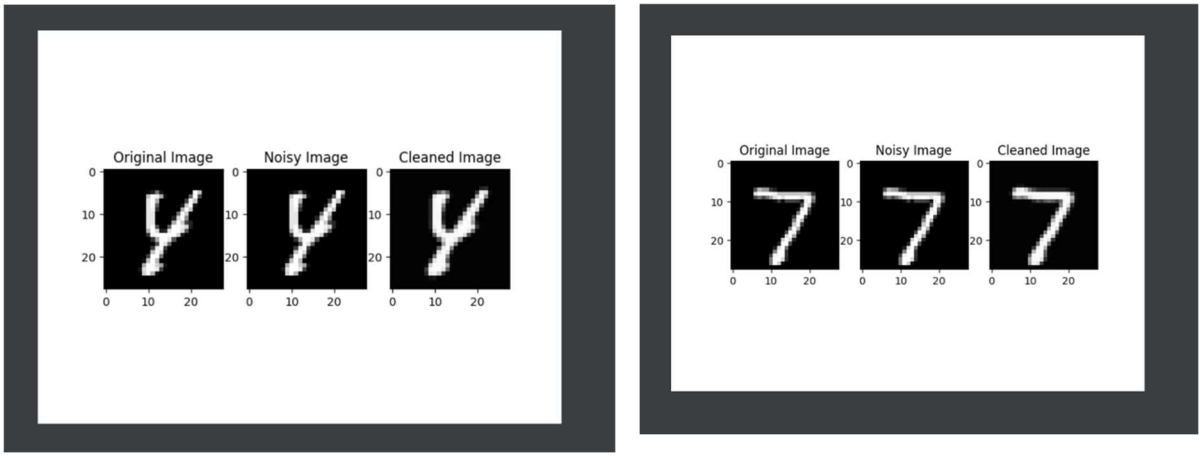
```python
if denoising:
    transformation = transforms.Compose([transforms.ToTensor(),
                                         transforms.Normalize((0.1307,), (0.3081,))])
    train_data = AutoEncodeDataset(
        torchvision.datasets.MNIST("./data", train=True, transform=transformation, download=False))
    test_data_xy = torchvision.datasets.MNIST("./data", train=False, transform=transformation,
                                              download=False)
    test_data_xx = AutoEncodeDataset(test_data_xy)
    trainloader = DataLoader(train_data, batch_size=128, shuffle=True)
    testdata = DataLoader(test_data_xx, batch_size=128)
    score_funcs = {}
    to_track = ["epoch", "total time", "train loss"]
    for eval_score in score_funcs:
        to_track.append("train " + eval_score)
        if testdata is not None:
            to_track.append("val " + eval_score)
    results = {}
    for item in to_track:
        results[item] = []
    total_train_time = 0
    model = None
    if tag:
        model = NeuralNetwork(True, denoising[0], denoising[1])
        criterion = nn.MSELoss()
        optimizer = torch.optim.RMSprop(model.parameters(), lr=0.01)
        model = model.train()
        epochs = 40
        start = time.time()
        for epoch in tqdm(range(epochs), desc="Epoch", disable=disable_tqdm):
            model = model.train()
            running_loss = 0.0
            y_true = []
            y_pred = []
            for inputs, labels in tqdm(trainloader, desc="Train Batch", leave=False, disable=disable_tqdm):
                batch_size = labels.shape[0]
                optimizer.zero_grad()
                y_hat = model(inputs)
                loss = criterion(y_hat, labels)
                loss.backward()
                optimizer.step()
```

برای ارزیابی و نمایش نیز از دو تابع زیر استفاده می کنیم:

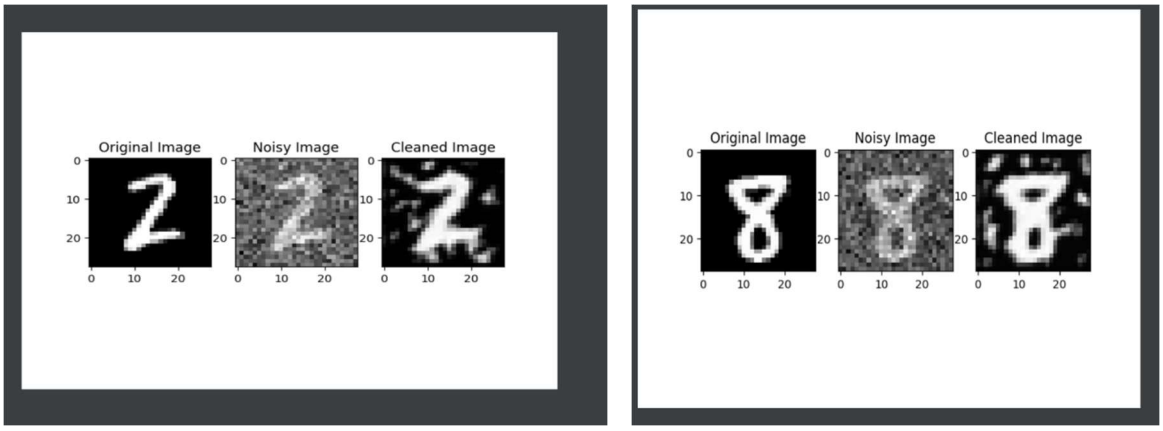
```python
def showEncodeDecode(original_copy, noisy, out):
    f, axarr = plt.subplots(1, 3)
    axarr[0].imshow(original_copy, cmap='gray')
    axarr[0].set_title("Original Image")
    axarr[1].imshow(noisy, cmap='gray')
    axarr[1].set_title("Noisy Image")
    axarr[2].imshow(out, cmap='gray')
    axarr[2].set_title("Cleaned Image")
    f.show()
```

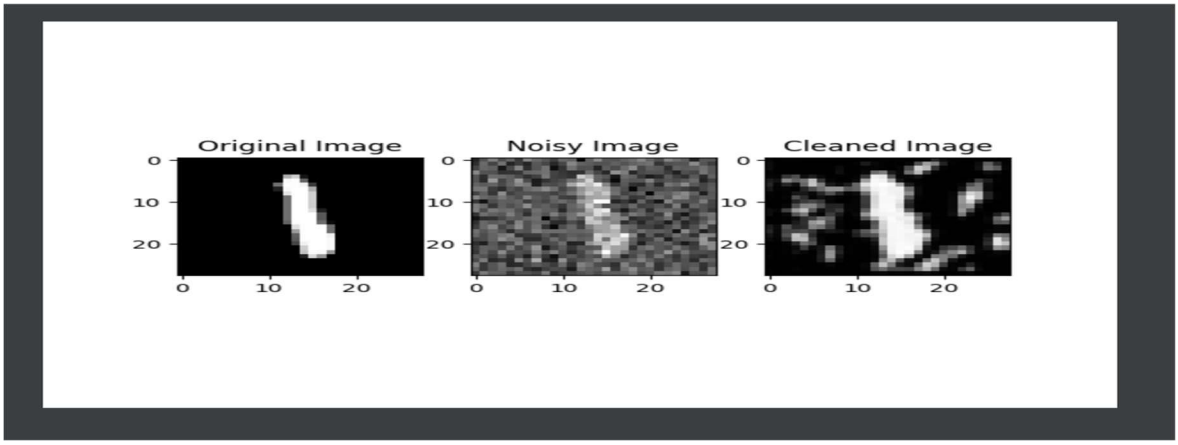
```python
def denoise(model, testloader, mean, std):
    model = model.eval()
    tr_1 = transforms.Compose([
        AddGaussianNoise_1(mean, std)
    ])
    for idx, data in enumerate(testloader):
        original, labels = data
        original_copy = original
        noisy = tr_1(original)
        out = model(noisy)
        out = out * 0.3081 + 0.1307
        original_copy = original_copy[0, 0].detach().cpu().numpy()
        noisy = noisy[0, 0].detach().cpu().numpy()
        out = out[0, 0].detach().cpu().numpy()
        showEncodeDecode(original_copy, noisy, out)
```

حال آزمایش اول را برای دو عکس که با توزیع نرمالی با میانگین 0 و انحراف از معیار 0.5 نویزی شده اند؛ نمایش می دهیم:



حال آزمایش دوم را که تفاوت را به خوبی نشان می دهد؛ برای سه عکس که با توزیع نرمالی با میانگین 0 و انحراف از معیار 1 نویزی شده اند؛ نمایش می دهیم:





حال آزمایش سوم ؛ برای سه عکس که با توزیع نرمالی با میانگین 0 و انحراف از معیار 2 نویزی شده اند؛ نمایش می دهیم: