به نام خدا

ملیکا احمدی رنجبر 97521036

❖ بسته به نوع داده‌ها (Spare -> Adam) و همچنین هدف اصلی (اینکه مدل سریع‌تر همگرا شود -> Adam یا Generalization بیشتری داشته باشد)، می‌توان Optimizerهای مختلفی را انتخاب کرد، اگرچه که تصور می‌شد بهترین Adam هست، می‌توان دید که الگوریتم SGD + NESTEROV هم می‌تواند بسیار خوب واقع شود. با توجه به سایت‌های مشخص شده، باید پیچیدگی شبکه را نیز برای انتخاب Optimizer در نظر بگیریم، چراکه Adam شاید در مدل‌های بسیار پیچیده بهتر از SGD باشد. و به طور کلی SGD در داده‌های Validation نتیجه بهتری دارد نسبت به Adam، و Adam نتیجه بهتری در Training Data دارد، با توجه به نمودارهای قرار داده شده است. برای SOTA انتخاب SGD بهتر است. اگرچه امکان ترکیب این دو نیز هست، برای متعادل‌تر کردن Generalization. اگر هم سرعت همگرایی مهم باشد همانطور که گفته شد بهترین Adam هست.

❖ هدف اصلی این سوال یادگیری Pytorch است. در ابتدا Dataset را با استفاده از torchvision می‌گیریم که به شکل زیر است:

```
# Load Dataset & Data Augmentation
transform = transforms.Compose(
    [transforms.RandomCrop(32, padding=4),
     transforms.RandomHorizontalFlip(),
     transforms.RandomAffine(degrees=40, scale=(.9, 1.1), shear=0),
     transforms.RandomPerspective(distortion_scale=0.2),
     transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5),
     transforms.RandomRotation(30),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

نکته مهم برای transform است که داده‌های ما را نرمال می‌کند و Data Augmentation را نیز برای آموزش بهتر (حالت‌های مختلف ورودی‌های) داریم و سپس Dataset را Download می‌کنیم. برای Test Data نیز همین روش را داریم. Classهای موجود را نیز مشخص می‌نماییم:

```
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

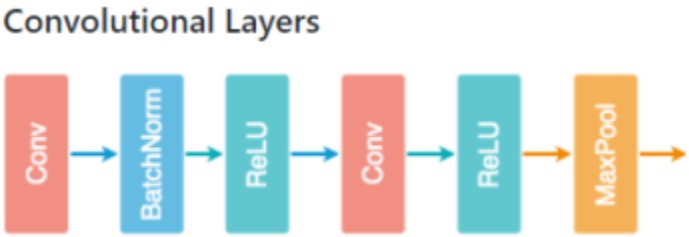
این Dataset دارای تصاویر رنگی 32*32 پیکسل است که در این 10 دسته جای دارند.

پس از این مراحل اولیه، به کمک Pytorch لایه‌های شبکه خود را مشخص می‌کنیم مه با توجه به شکل داده شده د صورت سوال است، در مرحله اول Convolution و خروجی آن به Fully Connected داده می‌شود. برای این کار Class درست کرده و از Moduleها و Featureهای Pytorch استفاده می‌کنیم. برای مثال یک قسمت را ببینیم که 3 بار تکرار شده است:

```
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv_layer = nn.Sequential(

            # Conv Layer block 1
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
```

این قسمت پیاده‌سازی این لایه‌های می‌باشد:



و Pytorch به آسانی تمام این لایه‌ها را آماده دارد، به طور کلی قسمت ReLU که Activation Function است، BatchNorm هم Mean و Standard Deviation را محاسبه میکند (برای هر Dimension) و سرعت آموزش را افزایش می‌دهد. Channelهای خروجی نیز در اکثر مراحل افزایش می‌یابد. قسمت بعدی نیز به شکل زیر است:

و برای هر لایه Linear یک تعدادی ورودی و خروجی داریم، همانطور که می‌بینیم خروجی در نهایت 10 است، که برابر 10 کلاس اصلی است.

```
self.fc_layer = nn.Sequential(
    nn.Dropout(p=0.1),
    nn.Flatten(),
    nn.Linear(120 * 4 * 4, 512),
    nn.ReLU(),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Dropout(p=0.1),
    nn.Linear(256, 10)
)
```

مراحل گفته شده در بالا صرفا مشخص کردن لایه‌های مورد نظر است، و در مرحله forward تمام آنها را صدا میزنیم و ورودی را می‌دهیم:

```
def forward(self, x):
    # conv layers
    x = self.conv_layer(x)
    # fc layer
    x = self.fc_layer(x)

    return x
```

تا به اینجا شبکه مورد نظرمان را با ساختار دلخواه درست کردیم و اکنون باید از آن استفاده کنیم و سایر موارد را مثل Optimizer, Loss Function را تعیین کنیم:

```
Loss = nn.CrossEntropyLoss()
Optimizer = optim.Adam(Model.parameters(), lr=0.001)
```

از کلاس شبکه یک Instance می‌سازیم و موارد دیگر را Set می‌کنیم. نوبت Training است در 2 Epochs:

```
for Epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader, start=0):
        # Input Data [Inputs, Labels]
        inputs, labels = data

        # Zero Gradient's Parameters
        Optimizer.zero_grad()

        # Forward, Backward, Optimize
        outputs = Model(inputs)
        loss = Loss(outputs, labels)
        loss.backward()
        Optimizer.step()
```

بر روی Data حرکت کرده عملیات Forward سپس Backward و در نهایت Optimization را انجام می‌دهیم.
در نهایت هم دقت هر کدا از کلاس‌ها را حساب می‌کنیم:

```
# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = Model(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1
```

در اینجا بدون نیاز به Gradient بر روی Test Data شبکه آموزش دیده (Model) را امتحان می‌کنیم و تعداد کلاس‌های درست تشخیص داده شده را نگه می‌داریم.
برای افزایش سرعت Training روی GPU، Run می‌کنیم.

```
Cuda = torch.cuda.is_available()
if Cuda:
    Model = Network().cuda()
    Model = torch.nn.DataParallel(Model, device_ids=range(torch.cuda.device_count()))
else:
    print('No CUDA available, Continue With CPU')
```

در نهایت هم دقت هر کلاس و دقت کل (به علت تعداد پایین Epoch دقت زیادی ندارد) حتی برای اطمینان از صحت عملکرد برنامه بیش از 60 بار با حالات مختف امتحا شد، همچنین با Epoch 4، اما میتوان کم شدن Loss را مشاهده کرد:

** متاسفانه با اینکه وقت بسیار زیادی (بیشتر از 7 ساعت) صرف سوال شد، علت مشکل پیدا نشد :)

```
CUDA available, GPU started
[1,  2000] loss: 2.240
[1,  4000] loss: 2.120
[1,  6000] loss: 2.065
[1,  8000] loss: 2.026
[1, 10000] loss: 1.997
[1, 12000] loss: 1.992
Epoch's Loss:  1004.4215157032013
Epoch's Accuracy:  10.724
[2,  2000] loss: 1.995
[2,  4000] loss: 1.974
[2,  6000] loss: 1.973
[2,  8000] loss: 1.949
[2, 10000] loss: 1.938
[2, 12000] loss: 1.947
Epoch's Loss:  954.6221051216125
Epoch's Accuracy:  13.114
[3,  2000] loss: 1.925
[3,  4000] loss: 1.916
[3,  6000] loss: 1.916
[3,  8000] loss: 1.917
[3, 10000] loss: 1.914
[3, 12000] loss: 1.923
Epoch's Loss:  945.6752637028694
Epoch's Accuracy:  14.058
[4,  2000] loss: 1.898
[4,  4000] loss: 1.893
[4,  6000] loss: 1.896
[4,  8000] loss: 1.886
[4, 10000] loss: 1.896
[4, 12000] loss: 1.879
Epoch's Loss:  921.4202265739441
Epoch's Accuracy:  14.328
```

```
Total Accuracy:  9.75
Accuracy for class plane is: 0.0 %
Accuracy for class car   is: 23.8 %
Accuracy for class bird  is: 0.0 %
Accuracy for class cat   is: 0.0 %
Accuracy for class deer  is: 0.0 %
Accuracy for class dog   is: 23.9 %
Accuracy for class frog  is: 0.0 %
Accuracy for class horse is: 0.0 %
Accuracy for class ship  is: 26.1 %
Accuracy for class truck is: 23.7 %
```

❖ منابع:

- ✓ https://alexander-schiendorfer.github.io/2020/02/24/a-worked-example-of-backprop.html
- ✓ *https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html*
- ✓ *https://zhenye-na.github.io/2018/09/28/pytorch-cnn-cifar10.html*