



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ «Информатика и системы управления»

КАФЕДРА _____ «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 5
по курсу «Теория искусственных нейронных сетей»
«Сверточные нейронные сети (CNN)»

Студент группы ИУ9-72Б Терентьева А. С.

Преподаватель Каганов Ю. Т.

Москва 2023

1 Цель

1. Изучение сверточных нейронных сетей.
2. Программная реализация архитектур сверточных нейронных сетей.
3. Обучение нейронных сетей на распознавание изображений.

2 Задание

1. Реализовать три модели CNN:
 - (a) LeNet (dataset: MNIST)
 - (b) VGG16 (dataset: CIFAR10)
 - (c) ResNet (dataset: CIFAR100)
2. Провести сравнительный анализ современных методов оптимизации (SGD, NAG, AdaDelta, ADAM) для каждой модели
3. Провести поиск оптимальных гиперпараметров для каждой оптимизации.

Для реализации использовать фреймворк PyTorch.

3 Реализация

Исходный код программы представлен в листингах 1– 3.

Листинг 1: LeNet.py

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torchvision
5 import torchvision.transforms as transforms
6 from torch.utils.data import DataLoader
7 from itertools import islice
8 import time
9 import matplotlib.pyplot as plt
10
11 batch_size = 64
```

```

12 transform = transforms.Compose([transforms.ToTensor(), transforms.
    Normalize((0.5, ), (0.5, ))])
13 trainset = torchvision.datasets.MNIST(root='./data', train=True,
    download=True, transform=transform)
14 trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)
15 testset = torchvision.datasets.MNIST(root='./data', train=False,
    download=True, transform=transform)
16 testloader = torch.utils.data.DataLoader(testset, batch_size=len(testset
    ), shuffle=True)
17
18 class LeNet(nn.Module):
19     def __init__(self):
20         super(LeNet, self).__init__()
21         self.conv1 = nn.Conv2d(1, 6, 5)
22         self.pool = nn.AvgPool2d(2, 2)
23         self.conv2 = nn.Conv2d(6, 16, 5)
24         self.pool2 = nn.AvgPool2d(2, 2)
25         self.fc1 = nn.Linear(256, 120)
26         self.fc2 = nn.Linear(120, 84)
27         self.fc3 = nn.Linear(84, 10)
28
29     def forward(self, x):
30         x = self.pool(torch.relu(self.conv1(x)))
31         x = self.pool2(torch.relu(self.conv2(x)))
32         x = x.view(-1, 256)
33         x = torch.relu(self.fc1(x))
34         x = torch.relu(self.fc2(x))
35         x = self.fc3(x)
36         return x
37
38 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
39 net0 = LeNet().to(device)
40 net = LeNet().to(device)
41 criterion = nn.CrossEntropyLoss()
42
43 def resetNet():
44     net.load_state_dict(net0.state_dict())
45
46 def countAccuracy():
47     correct = 0
48     size = len(testloader)
49     for image, label in islice(testloader, size):
50         image, label = image.to(device), label.to(device)
51         output = net(image)
52         _, predicted = torch.max(output.data, 1)
53         correct += (predicted == label).sum().item()

```

```

54     return 100 * correct / len(testset)
55
56 def train(optimizer, optim_name):
57     print(f"{optim_name}:")
58     start_time = time.time()
59     xs, ys = [], []
60     batch_num = len(trainloader)
61     for epoch in range(5):
62         running_loss = 0.0
63         for inputs, labels in islice(trainloader, batch_num):
64             inputs, labels = inputs.to(device), labels.to(device)
65             optimizer.zero_grad()
66             outputs = net(inputs)
67             loss = criterion(outputs, labels)
68             loss.backward()
69             optimizer.step()
70             running_loss += loss.item()
71         xs.append(epoch + 1)
72         ys.append(running_loss / len(trainset))
73         print(f'                : {xs[-1]},                : {ys[-1]}')
74     plt.plot(xs, ys, label = optim_name)
75     print(f"                {optim_name}: {countAccuracy():.2f}%")
76     print(f"                : {time.time() - start_time} \n
77 ")
78 resetNet()
79 print(f"                : {countAccuracy():.2f}%
80 ")
81 SGD = optim.SGD(net.parameters(), lr=0.1)
82 train(SGD, "SGD")
83 resetNet()
84 AdaDelta = optim.Adadelta(net.parameters(), lr=1.0)
85 train(AdaDelta, "AdaDelta")
86 resetNet()
87 NAG = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, nesterov=True)
88 train(NAG, "NAG")
89 resetNet()
90 Adam = optim.Adam(net.parameters(), lr=0.005)
91 train(Adam, "Adam")
92
93 plt.legend()
94 plt.show()

```

Листинг 2: VGG16.py

```

1 import torch
2 import torch.nn as nn

```

```

3 import torch.optim as optim
4 import torchvision
5 import torchvision.transforms as transforms
6 from torch.utils.data import DataLoader
7 import time
8 import matplotlib.pyplot as plt
9
10 batch_size = 64
11
12 transform = transforms.Compose([
13     transforms.ToTensor(),
14     transforms.Resize((32, 32)),
15     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
16 ])
17
18 trainset = torchvision.datasets.CIFAR10(root='./data/CIFAR10', train=
    True, download=True, transform=transform)
19 trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
    shuffle=True, num_workers=2)
20 testset = torchvision.datasets.CIFAR10(root='./data/CIFAR10', train=
    False, download=True, transform=transform)
21 testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle
    =True, num_workers=2)
22
23 class myVGG16(nn.Module):
24     def __init__(self):
25         super(myVGG16, self).__init__()
26
27         self.features = nn.Sequential(
28             nn.Conv2d(3, 64, kernel_size=3, padding=1),
29             nn.ReLU(inplace=True),
30
31             nn.Conv2d(64, 128, kernel_size=3, padding=1),
32             nn.ReLU(inplace=True),
33             nn.MaxPool2d(kernel_size=2, stride=2),
34
35             nn.Conv2d(128, 256, kernel_size=3, padding=1),
36             nn.ReLU(inplace=True),
37             nn.Conv2d(256, 256, kernel_size=3, padding=1),
38             nn.ReLU(inplace=True),
39             nn.MaxPool2d(kernel_size=2, stride=2),
40
41             nn.Conv2d(256, 512, kernel_size=3, padding=1),
42             nn.ReLU(inplace=True),
43             nn.Conv2d(512, 512, kernel_size=3, padding=1),
44             nn.ReLU(inplace=True),

```

```

45         nn.MaxPool2d(kernel_size=2, stride=2)
46     )
47     #
48     self.classifier = nn.Sequential(
49         nn.Linear(512 * 4 * 4, 1024),
50         nn.ReLU(inplace=True),
51         nn.Dropout(),
52         nn.Linear(1024, 1024),
53         nn.ReLU(inplace=True),
54         nn.Dropout(),
55         nn.Linear(1024, 10)
56     )
57
58     def forward(self, x):
59         x = self.features(x)
60         x = x.view(x.size(0), -1)
61         x = self.classifier(x)
62         return x
63
64 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
65 net0 = myVGG16().to(device)
66 net = myVGG16().to(device)
67 criterion = nn.CrossEntropyLoss()
68 optimizer = optim.SGD(net.parameters(), lr=0.1)
69
70 def resetNet():
71     net.load_state_dict(net0.state_dict())
72
73 def countAccuracy():
74     correct = 0
75     for image, label in trainloader:
76         image, label = image.to(device), label.to(device)
77         output = net(image)
78         _, predicted = torch.max(output.data, 1)
79         correct += (predicted == label).sum().item()
80     return 100 * correct / len(trainset)
81
82 def train(name):
83     xs, ys = [], []
84     for epoch in range(10):
85         running_loss = 0.0
86         for inputs, labels in trainloader:
87             inputs, labels = inputs.to(device), labels.to(device)
88             optimizer.zero_grad()
89
90             outputs = net(inputs)

```

```

91         loss = criterion(outputs, labels)
92         loss.backward()
93         optimizer.step()
94
95         running_loss += loss.item()
96         xs.append(epoch + 1)
97         ys.append(running_loss / len(trainset))
98         print(f'          : {xs[-1]},          : {ys[-1]} ')
99
100     print(f"          {name}          : {
countAccuracy() : .2 f}%")
101     plt.plot(xs, ys, label = name)
102     return xs, ys
103
104 resetNet()
105 print(f"          : {countAccuracy() : .2 f}%
")
106
107 optimizer = optim.SGD(net.parameters(), lr=0.1)
108 xs1, ys1 = train("SGD")
109
110 resetNet()
111 optimizer = optim.Adadelta(net.parameters(), lr=0.1)
112 xs2, ys2 = train("AdaDelta")
113
114 resetNet()
115 optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, nesterov=
True)
116 xs3, ys3 = train("NAG")
117
118 resetNet()
119 optimizer = optim.Adam(net.parameters(), lr=0.001)
120 xs4, ys4 = train("Adam")
121
122 plt.legend()
123 plt.show()

```

Листинг 3: ResNet.py

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torchvision
5 import torchvision.transforms as transforms
6 from torch.utils.data import DataLoader
7 from itertools import islice
8 import time

```

```

9 import matplotlib.pyplot as plt
10
11 batch_size = 64
12
13 transform = transforms.Compose([
14     transforms.ToTensor(),
15     transforms.Resize((32, 32)),
16     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
17 ])
18
19 trainset = torchvision.datasets.CIFAR100(root='./data/CIFAR100', train=
    True, download=True, transform=transform)
20 trainloader = torch.utils.data.DataLoader(trainset, batch_size=
    batch_size, shuffle=True, num_workers=2)
21 # testset = torchvision.datasets.CIFAR100(root='./data/CIFAR100', train=
    False, download=True, transform=transform)
22 # testloader = torch.utils.data.DataLoader(testset, batch_size=
    batch_size, shuffle=True, num_workers=2)
23
24 class Block(nn.Module):
25     def __init__(self, in_channels, out_channels, stride=1, downsampling
        =False):
26         super(Block, self).__init__()
27         self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
            stride=stride, padding=1, bias=False)
28         self.bn1 = nn.BatchNorm2d(out_channels)
29         self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size
        =3, stride=1, padding=1, bias=False)
30         self.bn2 = nn.BatchNorm2d(out_channels)
31         self.downsampling = downsampling
32         self.relu = nn.ReLU(inplace=True)
33         self.downsample = nn.Sequential()
34         if downsampling:
35             self.downsample = nn.Sequential(
36                 nn.Conv2d(in_channels, out_channels, kernel_size=1,
                    stride=stride, bias=False),
37                 nn.BatchNorm2d(out_channels)
38             )
39
40     def forward(self, x):
41         residual = x
42         out = self.conv1(x)
43         out = self.bn1(out)
44         out = self.relu(out)
45         out = self.conv2(out)
46         out = self.bn2(out)

```



```

47         if self.downsampling:
48             residual = self.downsample(x)
49         out += residual
50         out = self.relu(out)
51         return out
52
53
54 class ResNet(nn.Module):
55     def __init__(self, num_classes=100):
56         super(ResNet, self).__init__()
57         self.in_channels = 64
58         self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding
=1, bias=False)
59         self.bn = nn.BatchNorm2d(64)
60         self.relu = nn.ReLU(inplace=True)
61         self.layer1 = self.make_layer(64, blocks=3, stride=1)
62         self.layer2 = self.make_layer(128, blocks=4, stride=2)
63         self.layer3 = self.make_layer(256, blocks=6, stride=2)
64         self.layer4 = self.make_layer(512, blocks=3, stride=2)
65         self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
66         self.fc = nn.Linear(512, num_classes)
67
68     def make_layer(self, out_channels, blocks, stride):
69         layers = []
70         layers.append(Block(self.in_channels, out_channels, stride,
downsampling=True))
71         self.in_channels = out_channels
72         for _ in range(1, blocks):
73             layers.append(Block(out_channels, out_channels))
74         return nn.Sequential(*layers)
75
76     def forward(self, x):
77         out = self.conv1(x)
78         out = self.bn(out)
79         out = self.relu(out)
80         out = self.layer1(out)
81         out = self.layer2(out)
82         out = self.layer3(out)
83         out = self.layer4(out)
84         out = self.avg_pool(out)
85         out = out.view(out.size(0), -1)
86         out = self.fc(out)
87         return out
88
89 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
90 net0 = ResNet().to(device)

```

```

91 net = ResNet().to(device)
92 criterion = nn.CrossEntropyLoss()
93 optimizer = optim.SGD(net.parameters(), lr=0.1)
94
95 def resetNet():
96     net.load_state_dict(net0.state_dict())
97
98 def countAccuracy():
99     correct = 0
100     for image, label in trainloader:
101         image, label = image.to(device), label.to(device)
102         output = net(image)
103         _, predicted = torch.max(output.data, 1)
104         correct += (predicted == label).sum().item()
105     return 100 * correct / len(trainset)
106
107 def train(name):
108     xs, ys = [], []
109     for epoch in range(10):
110         running_loss = 0.0
111         for inputs, labels in trainloader:
112             inputs, labels = inputs.to(device), labels.to(device)
113             optimizer.zero_grad()
114
115             outputs = net(inputs)
116             loss = criterion(outputs, labels)
117             loss.backward()
118             optimizer.step()
119
120             running_loss += loss.item()
121         xs.append(epoch + 1)
122         ys.append(running_loss / len(trainset))
123         print(f'                : {xs[-1]},                : {ys[-1]}')
124
125         print(f"                {name}                : {
countAccuracy():.2f}%")
126         plt.plot(xs, ys, label = name)
127     return xs, ys
128
129 resetNet()
130 print(f"                : {countAccuracy():.2f}%
")
131
132 optimizer = optim.SGD(net.parameters(), lr=0.1)
133 xs1, ys1 = train("SGD")
134

```

```

135 resetNet()
136 optimizer = optim.Adadelta(net.parameters(), lr=0.1)
137 xs2, ys2 = train("AdaDelta")
138
139 resetNet()
140 optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, nesterov=
    True)
141 xs3, ys3 = train("NAG")
142
143 resetNet()
144 optimizer = optim.Adam(net.parameters(), lr=0.001)
145 xs4, ys4 = train("Adam")
146
147 plt.legend()
148 plt.show()

```

4 Результат работы

1. LeNet

Количество эпох: 5

Точность до обучения: 9,58%

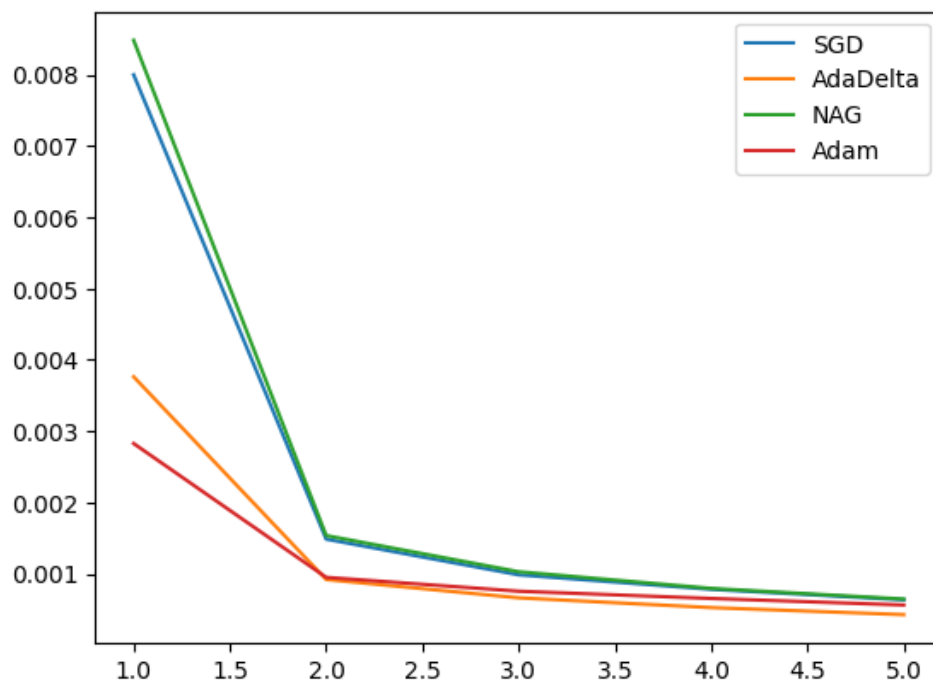


Рис. 1 — Сравнительный график сходимости методов оптимизации: зависимость значения ошибки от количества эпох

Таблица 1: Вариация гиперпараметров

Оптимизатор	Скорость обучения	Верность
SGD	0.1	98.53%
AdaDelta	1.0	98.79%
NAG	0.01	98.39%
Adam	0.005	98.69%

2. VGG16

Количество эпох: 10

Точность до обучения: 10%

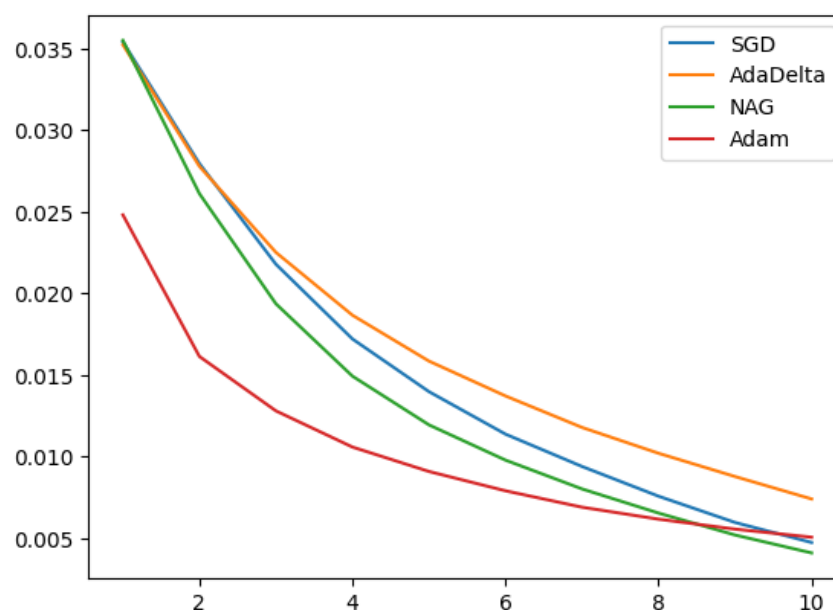


Рис. 2 — Сравнительный график сходимости методов оптимизации: зависимость значения ошибки от количества эпох

Таблица 2: Вариация гиперпараметров

Оптимизатор	Скорость обучения	Верность
SGD	0.1	91.81%
AdaDelta	0.1	84.60%
NAG	0.01	92.56%
Adam	0.001	91.19%

3. ResNet

Количество эпох: 10

Точность до обучения: 1.15%

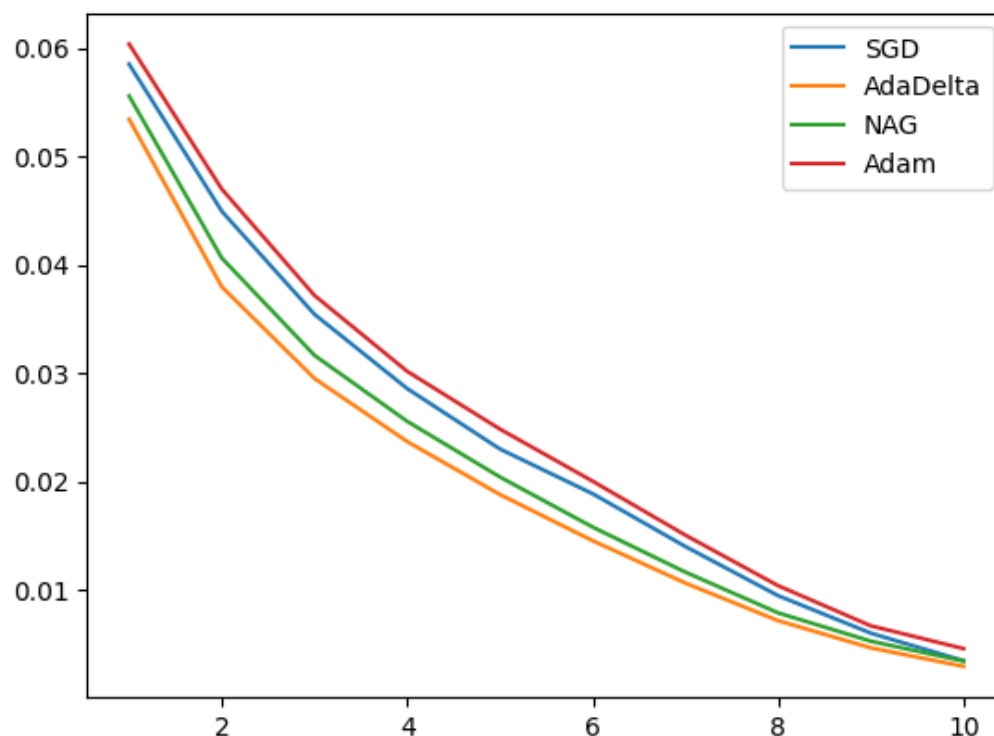


Рис. 3 — Сравнительный график сходимости методов оптимизации: зависимость значения ошибки от количества эпох

Таблица 3: Вариация гиперпараметров

Оптимизатор	Скорость обучения	Верность
SGD	0.1	93.58%
AdaDelta	0.1	95.61%
NAG	0.01	95.82%
Adam	0.001	94.79%

5 Выводы

В ходе выполнения лабораторной работы были изучены три архитектуры сверточных нейронных сетей: LeNet, VGG16 и ResNet, была написана их реализация на языке программирования Python с использованием фреймворка PyTorch. Были обучены все модели в среде выполнения Google Colab с применением GPU.

Был проведен сравнительный анализ современных методов оптимизации на каждой из модели, были подобраны оптимальные значения скорости обучения для каждой оптимизации.

В ходе эксперимента по исследованию работы программы на основе различных методов оптимизации (SGD, NAG, Adagrad, ADAM), для каждой модели были определены методы, показавшие лучший результат:

1. LeNet - AdaDelta / Adam
2. VGG16 - NAG / SGD
3. ResNet - NAG / AdaDelta