Лабораторная работа №6
по дисциплине «Системы поддержки принятия решений»
«Исследование методов уменьшения переобучения нейронных сетей на примере модернизированной сети LeNet на датасете CIFAR»

Цель работы – исследование современных архитектур сверточных нейронных сетей на примере задачи классификации изображений.

Классификация — понятие в науке, обозначающее разновидность деления объёма понятия по определённому основанию (признаку, критерию), при котором объём родового понятия (класс, множество) делится на виды (подклассы, подмножества), а виды, в свою очередь делятся на подвиды и т.д.

Переобучение (overtraining, overfitting) – явление, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (на примерах из тестовой выборки).

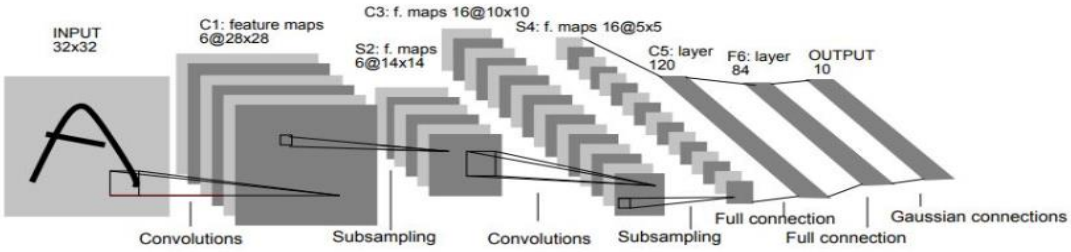
Переобучение – это как "зазубривание" (заучивание без понимания, осмысления): вы хорошо ответите на знакомые вопросы, но провалитесь на тесте. В машинном обучении есть несколько способов решения проблемы переобучения:

- Не использовать сложные модели, чтобы не позволить модели переобучиться.
- Найти больше обучающих данных или создать их искусственно с помощью аугментации. *Аугментация* (augmentation) – увеличение выборки данных для обучения через модификацию существующих данных.
- Использовать раннюю остановку. *Ранняя остановка* (early stopping) – это метод, который позволяет указать произвольно большое количество периодов обучения и прекратить обучение, как только производительность модели перестает улучшаться в наборе проверенных данных. Другими словами, ранняя остановка – это увеличение количества эпох до тех пор, пока модель не начнёт переобучаться, т.е. пока функция потерь (loss function) на валидационной выборке (после падения) не начнёт расти. Эпоха (epoch) – это одна итерация (один цикл) в процессе обучения, включающая предъявление всех примеров из обучающего множества и, возможно, проверку качества обучения на контрольном множестве.
- *Регуляризация*. В математике значение регуляризации связано с тем, чтобы сделать функцию более "регулярной" или гладкой. В машинном обучении регуляризация применяется к функции потерь (loss function), к которой добавляется некоторое число (штраф). В зависимости от штрафа различаются несколько регуляризаций. Регуляризация Тихонова ($L2$-регуляризация (ridge regression, или Tikhonov regularization) – дополнительный штраф за сумму квадратов коэффициентов, умноженную на коэффициент "лямбда" (регуляризационный параметр). В нейронных сетях вместо квадратов коэффициентов при степенях штрафуются веса нейронной сети (**weight decay** – сокращение веса).
- Регуляризация **DropOut** ("метод прореживания", "метод исключения") заключается в исключении из сети ("отсев") нейронов с вероятностью p. Исключенные нейроны возвращают нуль при любых входных данных или параметрах.

Общие положения

Текст программы представлен ниже.

Общая структура сверточной нейронной сети LeNet представлена на рисунке.



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Изменим LeNet, чтобы повысить качество классификации (смотри текст программы ниже). Также используем регуляризацию нейронной сети с помощью механизма дропаут, предназначенную для предотвращения переобучения сети.

Для реализации дропаута используем слой:

```
self.batch_drop1 = torch.nn.Dropout2d(p=0.15, inplace=False).
```

В функции **forward** тогда необходимо добавить**:**

```
x = self.batch_drop1(x).
```

Добавим L2-регуляризацию (регуляризация Тихонова). В PyTorch она активируется с помощью параметра **weight_decay** в оптимизаторе.

```
optimizer = torch.optim.Adam(net.parameters(), lr=1.0e-3, weight_decay=1e-5).
```

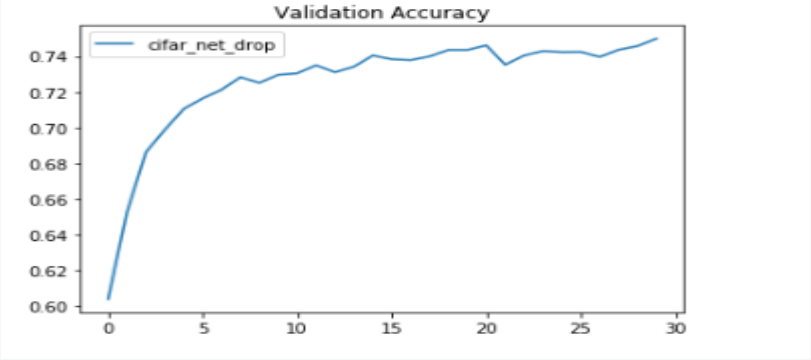
```
import torch
import random
import numpy as np

random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
##

import torchvision.datasets
##

MNIST_train = torchvision.datasets.MNIST('./', download=True, train=True)
MNIST_test = torchvision.datasets.MNIST('./', download=True, train=False)
)
##

X_train = MNIST_train.train_data
y_train = MNIST_train.train_labels
X_test = MNIST_test.test_data
y_test = MNIST_test.test_labels
##

len(y_train), len(y_test)
##

X_train = X_train.float()
X_test = X_test.float()
##

import matplotlib.pyplot as plt
plt.imshow(X_train[0, :, :])
plt.show()
print(y_train[0])
##

X_train = X_train.unsqueeze(1).float()
X_test = X_test.unsqueeze(1).float()
##

X_train.shape
##
```

```
class CIFARNet(torch.nn.Module):
    def __init__(self):
        super(CIFARNet, self).__init__()
        self.batch_norm0 = torch.nn.BatchNorm2d(3)
        ##self.batch_drop1 = torch.nn.Dropout(p=0.15, inplace=False)
        ##self.batch_drop2 = torch.nn.Dropout2d(p=0.15, inplace=False)

        self.conv1 = torch.nn.Conv2d(3, 16, 3, padding=1)
        self.act1  = torch.nn.ReLU()
        self.batch_norm1 = torch.nn.BatchNorm2d(16)
        self.pool1 = torch.nn.MaxPool2d(2, 2)

        self.conv2 = torch.nn.Conv2d(16, 32, 3, padding=1)
        self.act2  = torch.nn.ReLU()
        self.batch_norm2 = torch.nn.BatchNorm2d(32)
        self.pool2 = torch.nn.MaxPool2d(2, 2)

        self.conv3 = torch.nn.Conv2d(32, 64, 3, padding=1)
        self.act3  = torch.nn.ReLU()
        self.batch_norm3 = torch.nn.BatchNorm2d(64)

        self.fc1   = torch.nn.Linear(8 * 8 * 64, 256)
        self.act4  = torch.nn.Tanh()
        self.batch_norm4 = torch.nn.BatchNorm1d(256)

        self.fc2   = torch.nn.Linear(256, 64)
        self.act5  = torch.nn.Tanh()
        self.batch_norm5 = torch.nn.BatchNorm1d(64)

        self.fc3   = torch.nn.Linear(64, 10)

    def forward(self, x):
        x = self.batch_norm0(x)
        ##x = self.batch_drop2(x)
        x = self.conv1(x)
        x = self.act1(x)
        x = self.batch_norm1(x)
        ##x = self.batch_drop2(x)
        x = self.pool1(x)

        x = self.conv2(x)
        x = self.act2(x)
        x = self.batch_norm2(x)
        ##x = self.batch_drop2(x)
        x = self.pool2(x)

        x = self.conv3(x)
        x = self.act3(x)
        x = self.batch_norm3(x)
        ##x = self.batch_drop2(x)
```

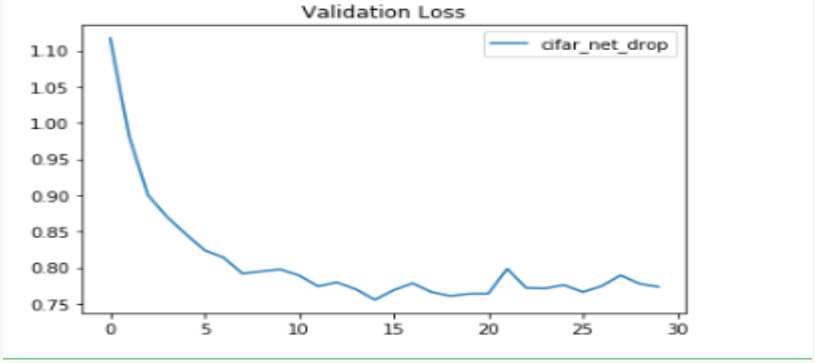
```
        x = x.view(x.size(0), x.size(1) * x.size(2) * x.size(3))
        x = self.fc1(x)
        x = self.act4(x)
        x = self.batch_norm4(x)
        ##x = self.batch_drop1(x)
        x = self.fc2(x)
        x = self.act5(x)
        x = self.batch_norm5(x)
        ##x = self.batch_drop1(x)
        x = self.fc3(x)

        return x
net = CIFARNet()
##
```

```
def train(X_train, y_train, X_test, y_test):
    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
    loss = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(net.parameters(), lr=1.0e-3, weight_decay=
1e-5)

    batch_size = 100

    test_accuracy_history = []
    test_loss_history = []

    X_test = X_test.to(device)
    y_test = y_test.to(device)

    for epoch in range(50):
        order = np.random.permutation(len(X_train))
        for start_index in range(0, len(X_train), batch_size):
            optimizer.zero_grad()
            net.train()

            batch_indexes = order[start_index:start_index+batch_size]

            X_batch = X_train[batch_indexes].to(device)
            y_batch = y_train[batch_indexes].to(device)

            preds = net.forward(X_batch)

            loss_value = loss(preds, y_batch)
            loss_value.backward()

            optimizer.step()

        net.eval()
        with torch.no_grad():
             test_preds = net.forward(X_test)

        test_loss_history.append(loss(test_preds, y_test).data.cpu())

        accuracy = (test_preds.argmax(dim=1) == y_test).float().mean().data
.cpu()
        test_accuracy_history.append(accuracy)

        print(accuracy)
```

```
    print('---------------')
    return test_accuracy_history, test_loss_history

accuracies = {}
losses = {}

########

device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
net = net.to(device)
##


accuracies['cifar_net_drop'], losses['cifar_net_drop'] = \
    train(X_train, y_train, X_test, y_test)
###

for experiment_id in accuracies.keys():
    plt.plot(accuracies[experiment_id], label=experiment_id)
plt.legend()
plt.title('Validation Accuracy');
###


for experiment_id in losses.keys():
    plt.plot(losses[experiment_id], label=experiment_id)
plt.legend()
plt.title('Validation Loss');
```




```
for experiment_id in losses.keys():
    plt.plot(losses[experiment_id], label=experiment_id)
plt.legend()
plt.title('Validation Loss');
```



Задание на лабораторную работу

1. Изучить понятия: **дропаут (dropout), регуляризация.**

   Примечание. Регуляризация в статистике, машинном обучении, теории обратных задач – метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение. Эта информация часто имеет вид штрафа за сложность модели.
2. Исследовать нейронную сеть **`CIFARNet`** без дропаута. Имеется ли переобучение в этой сети?
3. Исследовать нейронную сеть **`CIFARNet`** с включенным дропаутом. Исследовать влияние вероятности дропаута *p* на качество сети.
4. Найти по одному верному и неверному случаю классификации изображения (смотри пример ниже).

```
X_test.shape
```

```
torch.Size([10000, 3, 32, 32])
```

```
X_test.min(), X_test.max()
```

```
(tensor(0.), tensor(1.))
```

```
X_test1=X_test[111, :, :, :]
print(y_test[111])
```

```
tensor(0)
```

```
X_test1 = X_test1.unsqueeze(0)
X_test1.shape
```

```
torch.Size([1, 3, 32, 32])
```

```
X_test1=X_test1.to(device)
```

```
net.eval()
with torch.no_grad():
    y1 = net.forward(X_test1)
```

```
print(y1)
```

```
tensor([[ 6.8082,  0.8007,  0.9753,  0.2289, -4.5464, -1.7484, -3.0225, -2.0745,
          2.4453,  1.2176]], device='cuda:0')
```

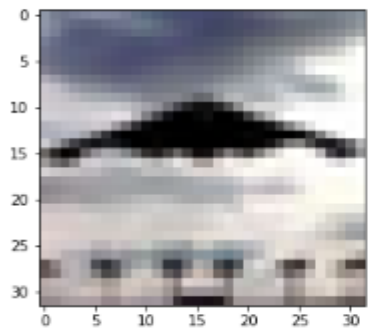
```
X_test2=X_test[111, :, :, :]
X_test2 = X_test2.permute(1, 2, 0)
X_test2=X_test2.to('cpu')
```

```
X_test2.shape
```

```
torch.Size([32, 32, 3])
```

```
plt.imshow(X_test2)
```

```
<matplotlib.image.AxesImage at 0x198aaea4388>
```



Содержание отчета

1. Титульный лист
2. Цель работы, постановка задачи исследования.
3. Описание методики исследования.
4. Результаты исследования в соответствии с заданием.
5. Выводы по работе.