Deep Learning School

Физтех-Школа Прикладной математики и информатики (ФПМИ) МФТИ

Домашнее задание. Обучение нейронных сетей на PyTorch.

В этом домашнем задании вам предстоит предсказывать типы небесных объектов. Эту задачу вы будете решать с помощью нейронных сетей, используя библиотеку PyTorch.

Вам необходимо заполнить пропуски в ноутбуке. Кое-где вас просят сделать выводы о проделанной работе. Постарайтесь ответить на вопросы обдуманно и развёрнуто.

В этом домашнем задании мы используем новый метод проверки --- Peer Review.

Peer Review — альтернативный способ проверки ваших заданий, который подразумевает, что после сдачи задания у вас появится возможность (и даже моральная обязанность, но не строгое обязательство) проверить задания нескольких ваших однокурсников. Соответственно, и ваши работы будут проверять другие учащиеся курса. Для выставления оценки необходимо будет, чтобы вашу работу проверило по крайней мере 3 ваших однокурсника. Вы же, выступая в роли проверяющего, сможете узнать больше о выполненном задании, увидеть, как его выполняли другие.

Чем больше заданий однокурсников вы проверите, тем лучше! Но, пожалуйста, проверяйте внимательно. По нашим оценкам, на проверку одной работы у вас уйдёт 5-10 минут. Подробные инструкции для проверки заданий мы пришлём позже.

ВАЖНО! Чтобы задание было удобнее проверять, необходимо сдать на Stepik два файла: файл в формате .ipynb и файл в формате .pdf. Файл .pdf можно получить, открыв File->Print и выбрать "Save as PDF". Аналогичный способ есть и в Jupyter.

```
import torch
from torch import nn
from torch import functional as F
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

Дисклеймер про CrossEntropyLoss и NLLLoss

Обычно в PyTorch не нужно делать Softmax как последний слой модели.

- Если Вы используете NLLLoss, то ему на вход надо давать лог вероятности, то есть выход слоя LogSoftmax. (Просто результат софтмакса, к которому применен логарифм)
- Если Вы используете CrossEntropyLoss, то применение LogSoftmax уже включено внутрь лосса, поэтому ему на вход надо подавать просто выход обычного линейного слоя без активации. По сути CrossEntropyLoss = LogSoftmax + NLLLoss

Зачем такие сложности, чтобы посчитать обычную кросс энтропию, которую мы использовали как лосс еще в логистической регрессии? Дело в том, что нам в любом случае придется взять логарифм от результатов софтмакса, а если делать это одной функцией, то можно сделать более устойчивую реализацию, которая даст меньшую вычислительную погрешность.

Таким образом, если у вас в конце сети, решающей задачу классификации, стоит просто линейный слой без активации, то вам нужно использовать CrossEntropy. В этой домашке везде используется лосс CrossEntropy

Задание 1. Создайте генератор батчей.

В этот раз мы хотим сделать генератор, который будет максимально похож на то, что используется в реальном обучении.

С помощью numpy вам нужно перемешать исходную выборку и выбирать из нее батчи размером batch_size, если размер выборки не делился на размер батча, то последний батч должен иметь размер меньше batch_size и состоять просто из всех оставшихся объектов. Возвращать нужно в формате (X_batch, y_batch). Необходимо написать именно генератор, то есть вместо return использовать yield.

Хорошая статья про генераторы: https://habr.com/ru/post/132554/

Ответ на задание - код

```
def batch_generator(X, y, batch_size):
    np.random.seed(42)
    perm = np.random.permutation(len(X))

    num_batches = X.shape[0] // batch_size
    for i in range(num_batches):
        yield X[i*batch_size:(i+1)*batch_size], y[i*batch_size:(i+1)*batch_size]
    if (X.shape[0] % batch_size):
        yield X[num_batches*batch_size:X.shape[0]], y[num_batches*batch_size:y.shape[0]]
```

Попробуем потестировать наш код

```
from inspect import isgeneratorfunction
assert isgeneratorfunction(batch_generator), "batch_generator должен быть генератором! В условии есть ссылка на доки"

X = np.array([
              [1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]
])
y = np.array([
              1, 2, 3
])

# Проверим shape первого батча
iterator = batch_generator(X, y, 2)
X_batch, y_batch = next(iterator)
assert X_batch.shape == (2, 3), y_batch.shape == (2,)
assert np.allclose(X_batch, X[:2]), np.allclose(y_batch, y[:2])

# Проверим shape последнего батча (их всего два)
X_batch, y_batch = next(iterator)
assert X_batch.shape == (1, 3), y_batch.shape == (1,)
assert np.allclose(X_batch, X[2:]), np.allclose(y_batch, y[2:])

# Проверим, что итерации закончились
iter_ended = False
try:
    next(iterator)
except StopIteration:
    iter_ended = True
assert iter_ended

# Еще раз проверим то, сколько батчей создает итератор
X = np.random.randint(0, 100, size=(1000, 100))
y = np.random.randint(-1, 1, size=(1000, 1))
num_iter = 0
for _ in batch_generator(X, y, 3):
    num_iter += 1
assert num_iter == (1000 // 3 + 1)
```

Задание 2. Обучите модель для классификации звезд

Загрузите датасет из файла sky_data.csv, разделите его на train/test и обучите на нем нейронную сеть (архитектура ниже). Обучайте на батчах с помощью оптимизатора Adam, lr подберите сами, пробуйте что-то вроде 1e-2

Архитектура:

1. Dense Layer c relu активацией и 50 нейронами
2. Dropout 80% (если другой keep rate дает сходимость лучше, то можно изменить) (попробуйте 50%)
3. BatchNorm
4. Dense Layer c relu активацией и 100 нейронами
5. Dropout 80% (если другой keep rate дает сходимость лучше, то можно изменить) (попробуйте для разнообразия 50%)
6. BatchNorm

7. Выходной Dense слой с количеством нейронов, равному количеству классов

Лосс - CrossEntropy.

В датасете классы представлены строками, поэтому классы нужно закодировать. Для этого в строчке ниже объявлен dict, с помощью него и функции map превратите столбец с таргетом в целое число. Кроме того, за вас мы выделили признаки, которые нужно использовать.

Загрузка и обработка данных

```
feature_columns = ['ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'run', 'camcol', 'field']
target_column = 'class'

target_mapping = {
    'GALAXY': 0,
    'STAR': 1,
    'QSO': 2
}
```

```
data = pd.read_csv('https://drive.google.com/uc?id=1K-8CtATw6Sv7k2dXco1fL5MAhTbKtIH3')
data['class'].value_counts()
```

```
GALAXY    4998
STAR      4152
QSO        850
Name: class, dtype: int64
```

```
data.head()
```

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid
0	1.237650e+18	183.531326	0.089693	19.47406	17.04240	15.94699	15.50342	15.22531	752	301	4	267	3.722360e+18
1	1.237650e+18	183.598371	0.135285	18.66280	17.21449	16.67637	16.48922	16.39150	752	301	4	267	3.638140e+17
2	1.237650e+18	183.680207	0.126185	19.38298	18.19169	17.47428	17.08732	16.80125	752	301	4	268	3.232740e+17
3	1.237650e+18	183.870529	0.049911	17.76536	16.60272	16.16116	15.98233	15.90438	752	301	4	269	3.722370e+18
4	1.237650e+18	183.883288	0.102557	17.55025	16.26342	16.43869	16.55492	16.61326	752	301	4	269	3.722370e+18

```
# Extract Features
X = data[feature_columns]
# Extract target
y = data[target_column]

# encode target with target_mapping
y = y.map(target_mapping)
```

Нормализация фичей

```
# Просто вычтите среднее и поделите на стандартное отклонение (с помощью пандас). Также преобразуйте всё в np.array
X = np.array((X-X.mean())/X.std(ddof=0))
y = np.array(y)
```

```
assert type(X) == np.ndarray and type(y) == np.ndarray, 'Проверьте, что получившиеся массивы являются np.ndarray'
assert np.allclose(y[:5], [1,1,0,1,1])
assert X.shape == (10000, 10)
assert np.allclose(X.mean(axis=0), np.zeros(10)) and np.allclose(X.std(axis=0), np.ones(10)), 'Данные не отнормированы'

```

Обучение

```
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
# Превратим данные в тензоры, чтобы потом было удобнее
X_train = torch.FloatTensor(X_train)
y_train = torch.LongTensor(y_train)
X_test = torch.FloatTensor(X_test)
y_test = torch.LongTensor(y_test)
```

Хорошо, данные мы подготовили, теперь надо объявить модель

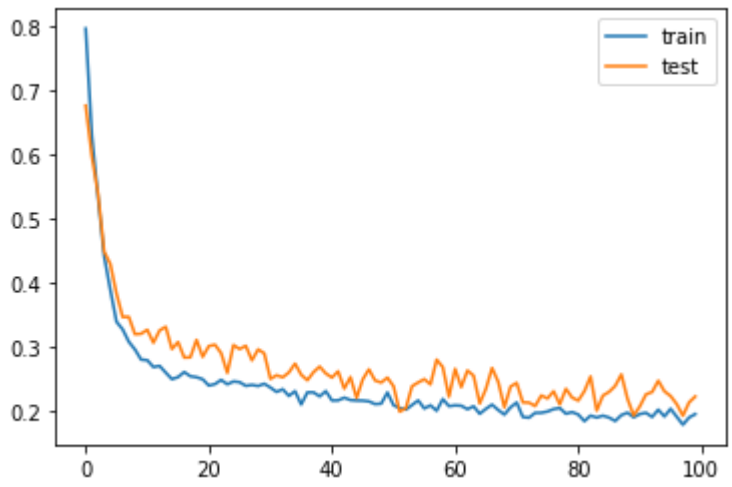
```
torch.manual_seed(42)
np.random.seed(42)
model = nn.Sequential(
   nn.Linear(X_train.shape[1], 50),
   nn.ReLU(),
   nn.Dropout(),
   nn.BatchNorm1d(50),
   nn.Linear(50, 100),
   nn.ReLU(),
   nn.Dropout(),
   nn.Linear(100, np.unique(y).shape[0]),
)

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)
```

Обучающий цикл

```
def train(X_train, y_train, X_test, y_test, num_epoch):
    train_losses = []
    test_losses = []
    for i in range(num_epoch):
        epoch_train_losses = []
        for X_batch, y_batch in batch_generator(X_train, y_train, 500):
            # На лекции мы рассказывали, что дропаут работает по-разному во время обучения и реального предсказания
            # Чтобы это учесть нам нужно включать и выключать режим обучения, делается это командой ниже
            model.train(True)
            # Посчитаем предсказание и лосс
            # YOUR CODE
            y_pred = model(X_batch)
            # зануляем градиент
            # YOUR CODE
            loss = loss_fn(y_pred, y_batch)
            # backward
            # YOUR CODE
            optimizer.zero_grad()
            loss.backward()
            # ОБНОВЛЯЕМ веса
            # YOUR CODE
            optimizer.step()
            # Запишем число (не тензор) в наши батчевые лоссы
            epoch_train_losses.append(loss.item())
        train_losses.append(np.mean(epoch_train_losses))

        # Теперь посчитаем лосс на тесте
        model.train(False)
        with torch.no_grad():
            # Сюда опять же надо положить именно число равное лоссу на всем тест датасете
            test_losses.append(loss.item())

    return train_losses, test_losses
```

```
def check_loss_decreased():
    print("На графике сверху, точно есть сходимость? Точно-точно? [Да/Нет]")
    s = input()
    if s.lower() == 'да':
        print("Хорошо!")
    else:
        raise RuntimeError("Можно уменьшить дропаут, уменьшить lr, поправить архитектуру, etc")
```

```
train_losses, test_losses = train(X_train, y_train, X_test, y_test, 100) #Подберите количество эпох так, чтобы график loss с
plt.plot(range(len(train_losses)), train_losses, label='train')
plt.plot(range(len(test_losses)), test_losses, label='test')
plt.legend()
plt.show()

check_loss_decreased()
assert train_losses[-1] < 0.3 and test_losses[-1] < 0.3
```



На графике сверху, точно есть сходимость? Точно-точно? [Да/Нет]
Да
Хорошо!

Вычислите accuracy получившейся модели на train и test

```
from sklearn.metrics import accuracy_score

model.eval()
train_pred_labels = model.forward(X_train).max(1)[1] #YOUR CODE: use forward
test_pred_labels = model.forward(X_test).max(1)[1] #YOUR CODE: use forward
# print(test_pred_labels)
train_acc = accuracy_score(train_pred_labels, y_train) # YOUR CODE)
test_acc = accuracy_score(test_pred_labels, y_test) # YOUR CODE)

assert train_acc > 0.9, "Если уж классифицировать звезды, которые уже видел, то не хуже, чем в 90% случаев"
assert test_acc > 0.9, "Новые звезды тоже надо классифицировать хотя бы в 90% случаев"

print("Train accuracy: {}\nTest accuracy: {}".format(train_acc, test_acc))
```

```
Train accuracy: 0.9602666666666667
Test accuracy: 0.9552
```

Задание 3. Исправление ошибок в архитектуре

Только что вы обучили полносвязную нейронную сеть. Теперь вам предстоит проанализировать архитектуру нейронной сети ниже, исправить в ней ошибки и обучить её с помощью той же функции train. Пример исправления ошибок есть в семинаре Григория Лелейтнера.

Будьте осторожнее и убедитесь, что перед запуском train вы вновь переопределили все необходимые внешние переменные (train обращается к глобальным переменным, в целом так делать не стоит, но сейчас это было оправдано, так как иначе нам пришлось бы передавать порядка 7-8 аргументов).

Чтобы у вас получилась такая же архитектура, как у нас, и ответы совпали, давайте определим некоторые правила, как исправлять ошибки:

1. Если вы видите лишний нелинейный слой, который стоит не на своем месте, просто удалите его. (не нужно добавлять новые слои, чтобы сделать постановку изначального слоя разумной. Удалять надо самый последний слой, который все портит. Для линейных слоев надо что-то исправить, а не удалить его)
2. Если у слоя нет активации, то добавьте ReLU или другую подходящую активацию
3. Если что-то не так с learning_rate, то поставьте 1e-2
4. Если что-то не так с параметрами, считайте первый параметр, который появляется, как верный (т.е. далее в сети должен использоваться он).
5. Ошибки могут быть и в полносвязных слоях.
6. Любые другие проблемы решаются более менее однозначно, если же у вас есть серьезные сомнения, то напишите в беседу в телеграме и пинганите меня @runfme

Задача все та же - классификация небесных объектов на том же датасете. После исправления сети вам нужно обучить ее.

Ответ на задачу - средний лосс на тестовом датасете

```
# torch.manual_seed(42)
# np.random.seed(42)
# # WRONG ARCH
# model = nn.Sequential(
#     nn.Dropout(p=0.5),
#     nn.Linear(6, 50),
```

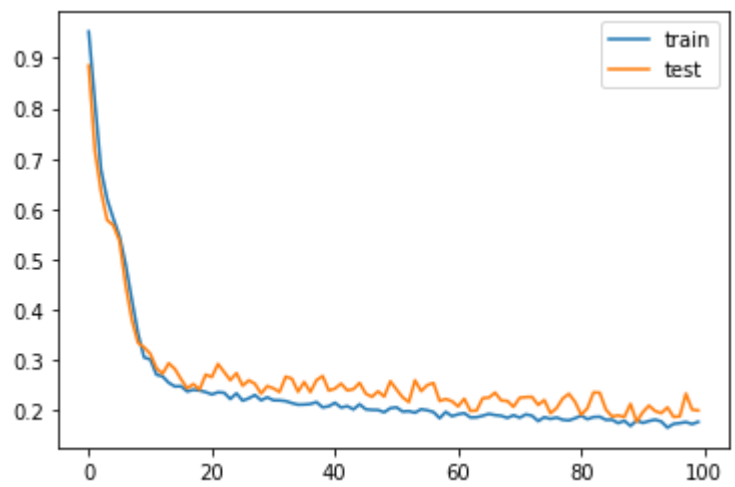
```
#     nn.ReLU(),
#     nn.Dropout(p=0.5),
#     nn.Linear(100, 200),
#     nn.Softmax(),
#     nn.Linear(200, 200),
#     nn.ReLU(),
#     nn.Dropout(p=0.5),
#     nn.Linear(200, 3),
#     nn.Dropout(p=0.5)
# )

# loss_fn = nn.CrossEntropyLoss()
# optimizer = torch.optim.Adam(model.parameters[:-2], lr=1e-100)
```

```
# RIGHT ARCH
torch.manual_seed(42)
np.random.seed(42)
model = nn.Sequential(
    # nn.Dropout(p=0.5),
    nn.Linear(X_train.shape[1], 50),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(50, 200),
    nn.Softmax(),
    nn.Linear(200, 200),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(200, 3),
    # nn.Dropout(p=0.5)
)


loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)
```

Обучите и протестируйте модель так же, как вы это сделали в задаче 2. Вычислите accuracy.

```
/R CODE
.n_losses, test_losses = train(X_train, y_train, X_test, y_test, 100)
 plot(range(len(train_losses)), train_losses, label='train')
 plot(range(len(test_losses)), test_losses, label='test')
 legend()
 show()

.n_pred_labels = model.forward(X_train).max(1)[1]
:_pred_labels = model.forward(X_test).max(1)[1]
:int(test_pred_labels)
.n_acc = accuracy_score(train_pred_labels, y_train)
:_acc = accuracy_score(test_pred_labels, y_test)


:rt train_acc > 0.9, "Если уж классифицировать звезды, которые уже видел, то не хуже, чем в 90% случаев"
:rt test_acc > 0.9, "Новые звезды тоже надо классифицировать хотя бы в 90% случаев"

it("Mean train loss: {}\nMean test loss: {}\nTrain accuracy: {}\nTest accuracy: {}".format(np.mean(train_losses),
                                                                                          np.mean(test_losses), train_acc, test_acc))
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/container.py:100: UserWarning: Implicit dimension choice for
  input = module(input)
```




```
Mean train loss: 0.240265870526433
Mean test loss: 0.263661238104105
Train accuracy: 0.958
Test accuracy: 0.9548
```

Задание 4. Stack layers

Давайте посмотрим, когда добавление перестает улучшать метрики. Увеличивайте блоков из слоев в сети, пока минимальный лосс на тестовом датасете за все время обучения не перестанет уменьшаться (20 эпох).

Стоит помнить, что нельзя переиспользовать слои с предыдущих обучений, потому что они уже будут с подобранными весами.

Чтобы получить воспроизводимость и идентичный нашему ответ, надо объявлять все слои в порядке, в котором они применяются внутри модели. Это важно, если вы будете собирать свою модель из частей. Перед объявлением этих слоев по порядку напишите

> torch.manual_seed(42)
> np.random.seed(42)

При чем каждый раз, когда вы заново создаете модель, перезадавайте random seeds

Опитимизатор - Adam(lr=1e-2)

```
МОДЕЛЬ ДЛЯ ПРИМЕРА, НА САМОМ ДЕЛЕ ВАМ ПРИДЕТСЯ СОЗДАВАТЬ НОВУЮ МОДЕЛЬ ДЛЯ КАЖДОГО КОЛИЧЕСТВА БЛОКОВ
del = nn.Sequential(
  nn.Linear(len(feature_columns), 100),
  nn.ReLU(),
  nn.Dropout(p=0.5),
  # Начало блока, который надо вставалять много раз
  nn.Linear(100, 100),
  nn.ReLU(),
  nn.BatchNorm1d(100),
  # Конец блока
  nn.Linear(100, 3)
  # Блока Softmax нет, поэтому нам нужно использовать лосс - CrossEntropyLoss

```

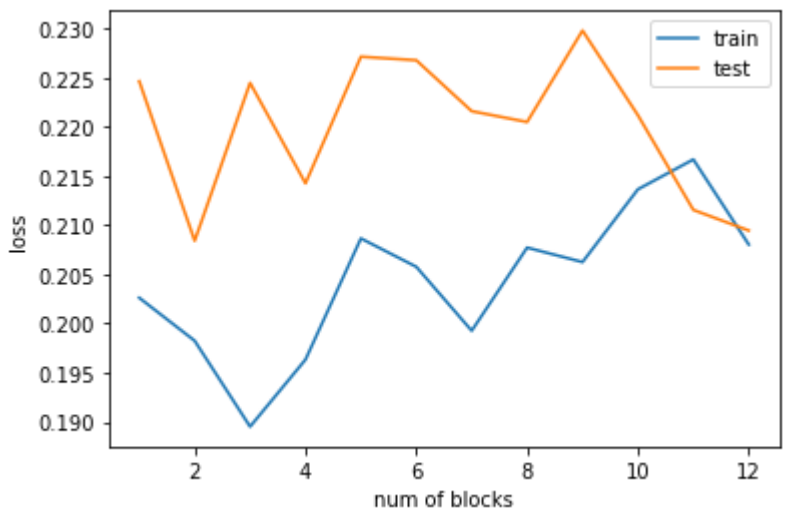
```
# Вы уже многое умеете, поэтому теперь код надо написать самому
# Идея - разделить модель на части.
# Вначале создать head часть как Sequential модель, потом в цикле создать Sequential модели, которые представляют
# из себя блоки, потом создать tail часть тоже как Sequential, а потом объединить их в одну Sequential модель
# вот таким кодом: nn.Sequential(header, *blocks, footer)
# Важная идея тут состоит в том, что модели могут быть частями других моделей)

def get_sequential(num_of_blocks):
    header = nn.Sequential(
        nn.Linear(len(feature_columns), 100),
        nn.ReLU(),
        nn.Dropout(p=0.5)
    )

    blocks = []
    for i in range (num_of_blocks):
        blocks.append(
            nn.Sequential(
                nn.Linear(100, 100),
                nn.ReLU(),
                nn.BatchNorm1d(100)
            )
        )

    footer = nn.Sequential(nn.Linear(100, 3))
    return nn.Sequential(header, *blocks, footer)

```

```
epochs = 20

# Collect minimums for visualization
train_min_loss_arr = []
test_min_loss_arr = []

# First step to initialize minimum
torch.manual_seed(42)
np.random.seed(42)
model = get_sequential(1)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

train_losses, test_losses = train(X_train, y_train, X_test, y_test, epochs)
min_test_loss = np.amin(test_losses)

train_min_loss_arr.append(np.amin(train_losses))
```

```
test_min_loss_arr.append(min_test_loss)

test_pred_labels = model.forward(X_test).max(1)[1]
test_acc = accuracy_score(test_pred_labels, y_test)

print("Num of blocks: 1")
print("New minimum test loss: ", min_test_loss)
print("Test accuraacy: ", test_acc)
# Block adding test
notImproved = 0
bestResult = 0
i = 2 # For 1 block we've already trained the model
while notImproved < 10:
    # Initialize new model with i blocks
    torch.manual_seed(42)
    np.random.seed(42)
    model = get_sequential(i)
    loss_fn = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-2)

    # Train the model
    train_losses, test_losses = train(X_train, y_train, X_test, y_test, epochs)
    new_min_test_loss = np.amin(test_losses)

    # Save for visualization
    train_min_loss_arr.append(np.amin(train_losses))
    test_min_loss_arr.append(new_min_test_loss)

    test_pred_labels = model.forward(X_test).max(1)[1]
    test_acc = accuracy_score(test_pred_labels, y_test)

    print("\nNum of blocks: ", i)
    print("Global minimum test loss: ", min_test_loss)
    print("New minimum test loss: ", new_min_test_loss)
    print("Test accuraacy: ", test_acc)

    if new_min_test_loss < min_test_loss:
        min_test_loss = new_min_test_loss
        best_result = i
        i += 1
    else:
        print("Adding a block did not bring an improvement.")
        i += 1
        notImproved += 1
print("\nOptimal parameter: ", best_result)
```

```
Num of blocks: 1
New minimum test loss:  0.22464022040367126
Test accuraacy:  0.9408

Num of blocks:  2
Global minimum test loss:  0.22464022040367126
New minimum test loss:  0.2084188312292099
Test accuraacy:  0.9384

Num of blocks:  3
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22447259724140167
Test accuraacy:  0.9464
Adding a block did not bring an improvement.

Num of blocks:  4
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.21426299214363098
Test accuraacy:  0.9436
Adding a block did not bring an improvement.

Num of blocks:  5
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22713923454284668
Test accuraacy:  0.9444
Adding a block did not bring an improvement.

Num of blocks:  6
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22677820920944214
Test accuraacy:  0.9404
Adding a block did not bring an improvement.

Num of blocks:  7
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.2215997874736786
Test accuraacy:  0.9404
Adding a block did not bring an improvement.

Num of blocks:  8
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22049927711486816
Test accuraacy:  0.9376
Adding a block did not bring an improvement.

Num of blocks:  9
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22979915142059326
Test accuraacy:  0.9424
Adding a block did not bring an improvement.

Num of blocks:  10
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.22118793427944183
Test accuraacy:  0.9372
Adding a block did not bring an improvement.

Num of blocks:  11
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.2115572988986969
Test accuraacy:  0.938
Adding a block did not bring an improvement.

Num of blocks:  12
Global minimum test loss:  0.2084188312292099
New minimum test loss:  0.2094598263502121
Test accuraacy:  0.9444
Adding a block did not bring an improvement.

Optimal parameter:  2
```

```
1 plt.plot(range(1, len(train_min_loss_arr)+1), train_min_loss_arr, label='train')
2 plt.plot(range(1, len(test_min_loss_arr)+1), test_min_loss_arr, label='test')
3 plt.xlabel('num of blocks')
4 plt.ylabel('loss')
5 plt.legend()
6 plt.show()
7 print("\nOptimal parameter: ", best_result)
```




```
Optimal parameter:  2
```

Задание 5. Сделайте выводы

Начиная с какого количества блоков минимальный лосс за время обучения увеличивается? Почему лишнее количество блоков не помогает модели?

1. Для нашего случая, учитывая заданные параметры модели, оптимальным количеством блоков является 2, дальнейшее их добавление негативно влияет на получаемый loss и время обучения модели.
2. С увеличением количества слоев и при недостаточно обширной выборке модель может стать более восприимчивой к переобучению. Это проявляется в том, что модель хорошо описывает примеры из обучающей выборки, адаптируясь к ее примерам, однако вместе с этим обобщающая способность снижается.