Imports & private packages

[ ] ↳ *8 cells hidden*

Training routine

```
def epoch(net, train_loader, test_loader, loss_func, optimizer, batch_size, seq_len):
    epoch_losses = np.zeros([len(train_loader), ])
    correct = 0
    incorrect = 0

    for i, d in enumerate(train_loader):
        # Scale inputs and get predictions
        inp = d[:, :, :-1]
        #inp = d[:, :, [0, 2]]
        inp[:, :, 0] /= 1000
        predictions = net(inp.float()).squeeze()

        # Each batch consists of measurement batches, where seq_len measurements are pu
        # measurement batch, every measurement has the same label as it needs to be fro
        # This leads to a target array where the target class is contained for each mea
        # batch. With this, CrossEntropyLoss would not work as the predictions are made
        # batch and CEL therefore expects only on class label per measurement batch. Th
        # element of the last dimension of d is considered as target (all the entries i
        # same anyways so it could be any entry)
        targets = d[:, :, -1][:, 0].long()

        correct += (predictions.argmax(dim=-1) == targets).sum().item()
        incorrect += len(targets) - (predictions.argmax(dim=-1) == targets).sum().item(

        #print(predictions.argmax(dim=-1), targets)

        # Calculate the loss
        loss = loss_func(predictions, targets)
        epoch_losses[i] = loss

        # Backward step
        net.zero_grad()
        loss.backward()
        optimizer.step()

    # Evaluate the network on the test set but DO NOT train on it
    test_loss, test_accuracy = test(net, test_loader, loss_func)

    train_accuracy = correct/(correct + incorrect)
    print('Training accuracy: ', train_accuracy)
    return epoch_losses.mean(), train_accuracy, test_loss, test_accuracy
```

```
# Training and testing routines

def train(net, train_loader, test_loader, batch_size, seq_len):
    # Hyperparameters
    nr_epochs = 250
    lr = 0.001

    loss_func = nn.CrossEntropyLoss()
    optimizer = Adam(net.parameters(), lr=lr)

    train_losses = np.zeros([nr_epochs, ])
    train_accuracies = np.zeros([nr_epochs, ])
    test_losses = np.zeros([nr_epochs, ])
    test_accuracies = np.zeros([nr_epochs, ])

    # Save the net with the lowest training loss as the best net
    best_net = net
    _, _, best_loss, _ = epoch(net, train_loader, test_loader, loss_func, optimizer, ba

    for i in range(nr_epochs):
        train_losses[i], train_accuracies[i], test_losses[i], test_accuracies[i] = epoc

        if test_losses[i] < best_loss:
            best_net = net

        if i % 10 == 0:
            print(train_losses[i])

    return best_net, train_losses, train_accuracies, test_losses, test_accuracies

def test(net, test_loader, loss_func=None):
    correct = 0
    incorrect = 0

    test_losses = np.zeros([len(test_loader), ])

    for i, d in enumerate(test_loader):
        inp = d[:, :, :-1]
        #inp = d[:, :, [0, 2]]
        inp[:, :, 0] /= 1000
        predictions = net(inp.float())
        targets = d[:, :, -1][:, 0].long()

        correct += (predictions.argmax(dim=-1).flatten() == targets).sum().item()
        incorrect += len(targets) - (predictions.argmax(dim=-1).flatten() == targets).s

        # Get the testing loss
        if loss_func is not None:
            loss = loss_func(predictions.squeeze(), targets)
            test_losses[i] = loss

    accuracy = correct/(correct + incorrect)
    print('Test accuracy: ', accuracy)

    return test_losses.mean(), accuracy
```

Network definition

```
# Hyperparameters
seq_len = 1
batch_size = 256

# Create the dataset
measurement = 'C'
dc = DatasetCreator(elements=None, splits=(0.8, 0.2), validation=False, seq_len=seq_len
train_dataset, test_dataset, val_dataset = dc.get_datasets()

# Create the DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)
if val_dataset:
    val_loader = DataLoader(val_dataset)
```

```
Dataset shape:  (205592, 1, 4)
```

```
# Create the network
t = True
net = PhaseClassifier(train=t, measurement=measurement, hidden_layers=2, hidden_size=12
```

Training

```
# Train the network
best_net, train_losses, train_accuracies, test_losses, test_accuracies = train(net, tra

# Test the trained network
test(best_net, test_loader)
```

```
Test accuracy:  0.9877167068335536
Training accuracy:  0.9899023308299933
Test accuracy:  0.8980214568918604
Training accuracy:  0.9859089847854002
Test accuracy:  0.9861424240068414
Training accuracy:  0.9931563484960504
Test accuracy:  0.9812252196221721
Training accuracy:  0.9892797385112261
Test accuracy:  0.9915260825623883
Training accuracy:  0.9864148410443986
Test accuracy:  0.9876584000621939
Training accuracy:  0.993730300789914
Test accuracy:  0.9901461556402084
Training accuracy:  0.9884431300828826
Test accuracy:  0.984159993780611
Training accuracy:  0.9904422351064244
0.030177996339089935
Test accuracy:  0.9856565342455104
```

```
Training accuracy:  0.9893478345460913
Test accuracy:  0.9873668662053953
Training accuracy:  0.9883458500330752
Test accuracy:  0.9024916426961052
Training accuracy:  0.9932974045682711
Test accuracy:  0.9835574904765607
Training accuracy:  0.9914198996069886
Test accuracy:  0.9958796548239135
Training accuracy:  0.9867164091988015
Test accuracy:  0.9924784264945969
Training accuracy:  0.99486361337017
Test accuracy:  0.9924201197232372
Training accuracy:  0.9894353865909179
Test accuracy:  0.9962100598616186
Training accuracy:  0.9879761858438072
Test accuracy:  0.9919731011428127
Training accuracy:  0.9921786839954863
Test accuracy:  0.995510378605302
Training accuracy:  0.9879859138487879
0.039907462411011056
Test accuracy:  0.9957047345098344
Training accuracy:  0.9933168605782327
Test accuracy:  0.9940721449117624
Training accuracy:  0.985169656406864
Test accuracy:  0.9924784264945969
Training accuracy:  0.9918917078485544
Test accuracy:  0.9894076032029853
Training accuracy:  0.9890024903692751
Test accuracy:  0.9950050532535178
Training accuracy:  0.9906854352309429
Test accuracy:  0.9866477493586255
Training accuracy:  0.9922273240203899
Test accuracy:  0.9920314079141724
Training accuracy:  0.9876113856570294
Test accuracy:  0.9964627225375107
Training accuracy:  0.9901309389470407
Test accuracy:  0.9562893570706678
Training accuracy:  0.9908751313280673
Test accuracy:  0.9562893570706678
(0.0, 0.9562893570706678)
```

```
1 torch.save(best_net, 'PhaseClassifier_9563.pth')
```

```
1 fig, ax = plt.subplots(1, 2, figsize=(14,7))
2 ax[0].plot(train_losses, label='Training loss')
3 ax[0].plot(test_losses, label='Test loss')
4 ax[0].set_xlabel('Epochs'), ax[0].set_ylabel('Loss')
5 ax[0].set_title('Losses over time')
6 ax[0].grid()
7 ax[0].legend()
8 ax[1].plot(train_accuracies, label='Training accuracy')
9 ax[1].plot(test_accuracies, label='Test accuracy')
10 ax[1].set_xlabel('Epochs'), ax[1].set_ylabel('Accuracy')
11 ax[1].set_title('Training and test accuracy over time')
12 ax[1].grid()
13 ax[0].legend()
14 plt.show()
```

```
#net = torch.load(r"C:\Users\danie\Documents\Montanuni\Masterarbeit\5 Programmcodes\pac
test(best_net, test_loader)
```

```
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
```

```
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
torch.Size([256, 1, 34]) torch.Size([256])
```