Imports & private packages

[ ] ↳ *8 cells hidden*

Training routine

```
def epoch(net, train_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()

    print('Training accuracy: ', correct/(correct + incorrect))

    return epoch_losses.mean()
```

```
# Training and testing routines

def train(net, train_loader, batch_size, seq_len, save=True):
```

```
    # Hyperparameters
    nr_epochs = 500
    lr = 0.01

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

    losses = 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, loss_func, optimizer, batch_size, seq_len)

    for i in range(nr_epochs):
        losses[i] = epoch(net, train_loader, loss_func, optimizer, batch_size, seq_len)

        if losses[i] < best_loss:
            best_net = net

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

    return best_net

def test(net, test_loader):
    correct = 0
    incorrect = 0

    for d in 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

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

    return 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)
```

Training

```
# Train the network
best_net = train(net, train_loader, batch_size, seq_len)

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

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

```
Test accuracy:  0.5725919303428438
0.5725919303428438
```

```
#torch.save(best_net, 'PhaseClassifier_8778.pth')
```

```

```

0s completed at 11:05 AM