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()

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

```
# Training and testing routines

def train(net, train_loader, batch_size, seq_len, save=True):
    # Hyperparameters
    nr_epochs = 250
    lr = 0.01

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

    losses = np.zeros([nr_epochs, ])
    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, loss_func, optimizer, batch_size, seq_len)

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

        if losses[i] < best_loss:
            best_net = net

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

    return best_net, losses, accuracies

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, hidden_layers=2, hidden_size=12
```

Training

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

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

```
Training accuracy:  0.9673430872796607
Training accuracy:  0.9535585042219542
0.17972982218558553
Training accuracy:  0.9794739094906416
Training accuracy:  0.9492441340129966
Training accuracy:  0.9809087902253006
Training accuracy:  0.9629995330557609
Training accuracy:  0.9739435386590918
Training accuracy:  0.9596871473598194
Training accuracy:  0.9624109887544262
Training accuracy:  0.9657233744503677
Training accuracy:  0.7398293707926379
Training accuracy:  0.8837406124751935
0.32309043268436816
Training accuracy:  0.8961243628156738
Training accuracy:  0.9121512510214406
Training accuracy:  0.9276528269582474
Training accuracy:  0.928596443441379
Training accuracy:  0.9257266819720612
Training accuracy:  0.9315002529281295
Training accuracy:  0.9329059496478462
Training accuracy:  0.9253521537803028
Training accuracy:  0.9477897972683762
Training accuracy:  0.9386795206039146
0.17112859642475992
Training accuracy:  0.9263881863107514
Training accuracy:  0.9217430639324488
Training accuracy:  0.9360237752441729
Training accuracy:  0.9190094945328612
Training accuracy:  0.9308290205844585
Training accuracy:  0.9423275224716915
Training accuracy:  0.9366706875753921
Training accuracy:  0.9378623681855325
Training accuracy:  0.936301023386124
Training accuracy:  0.9244231293046422
0.25426549873597437
Training accuracy:  0.9493365500603136
Training accuracy:  0.940148449356006
Training accuracy:  0.9404451535079186
Training accuracy:  0.947760613253434
Training accuracy:  0.9411893458889451
Training accuracy:  0.945547492120316
Training accuracy:  0.9314078368808125
Training accuracy:  0.9399198412389587
Training accuracy:  0.9403332814506401
Training accuracy:  0.948354021557259
0.16352628060704588
Training accuracy:  0.9417195221603953
Training accuracy:  0.9501634304836764
Training accuracy:  0.9368360636600646
Training accuracy:  0.9554846492081404
Training accuracy:  0.9568125218880112
Training accuracy:  0.8922866648507723
Training accuracy:  0.9556743453052648
Training accuracy:  0.9450124518463754
Training accuracy:  0.9455377641153352
Test accuracy:  0.9092357925833787
0.9092357925833787
```

```python
fig, ax = plt.subplots(1, 2, figsize=(14,7))
ax[0].plot(losses)
ax[0].set_xlabel('Epochs'), ax[0].set_ylabel('Loss')
ax[0].set_title('Losses over time')
ax[0].grid()
ax[1].plot(accuracies)
ax[1].set_xlabel('Epochs'), ax[1].set_ylabel('Training accuracy')
ax[1].set_title('Training accuracy over time')
ax[1].grid()
plt.show()
```

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

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

✓ 24m 7s completed at 12:11 PM ● ×