Imports & private packages

[ ] ↳ *8 cells hidden*

Training routine

```
def epoch(net, train_loader, loss_func, optimizer, batch_size, seq_len, measurement='G'
    epoch_losses = np.zeros([len(train_loader), ])

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

        # 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()

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

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

    print(epoch_losses.mean())

    return epoch_losses.mean()
```

```
# Training and testing routines

def train(net, train_loader, batch_size, seq_len, measurement='G'):
    # Hyperparameters
    nr_epochs = 10000
    lr = 0.0025

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

    losses = np.zeros([nr_epochs, ])

```

```
    best_loss = epoch(net, train_loader, loss_func, optimizer, batch_size, seq_len, mea
    best_net = net

    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
            best_loss = losses[i]

    return best_net

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

    for d in test_loader:
        inp = d[:, :, :-1].squeeze(1)/1000
        prediction = net(inp.float().to(device)).squeeze().argmax()
        target = d[:, :, -1][:, 0].long().to(device)
        if prediction == target:
            correct += 1
        else:
            incorrect += 1

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

    return accuracy
```

Run

Network definition

```
# Hyperparameters
seq_len = 5
measurement = 'C'

dc = DatasetCreator(elements=None,  splits=(0.8, 0.2), validation=False, seq_len=seq_le
train_dataset, test_dataset, val_dataset = dc.get_datasets()
```

```
# Hyperparameters
batch_size = 64

# Create the DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=True)
if val_dataset:
    val_loader = DataLoader(val_dataset)
```

```
# Create the network
t = True
net = ElementClassifier(train)

# Check if cuda is available and send net to device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = net.to(device)
```

Train

```
best_net = train(net, train_loader, batch_size, seq_len, measurement)
test(best_net, test_loader)
```

```
test(best_net, test_loader)
```

```
Test accuracy:  0.7265640615630622
0.7265640615630622
```

```
torch.save(best_net, 'ElementClassifier.pth')
```

✓ 0s completed at 11:58 AM