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), ])
····correct·=·0
····incorrect·=·0

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

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

        # 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)
    return epoch_losses.mean(), accuracy
```

```
# Training and testing routines

def train(net, optimizer, train_loader, test_loader, batch_size, seq_len, measurement='
    loss_func = nn.CrossEntropyLoss()

    losses = np.zeros([nr_epochs, ])
    accuracies = np.zeros([nr_epochs, ])
```

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

    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
            best_loss = losses[i]

        if i % 10 == 0:
            print('Epoch ', i)
            print('Loss: ', losses[i])
            print('Training accuracy: ', accuracies[i])
            #test(best_net, test_loader)

    return best_net

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

    for d in test_loader:
        inp = d[:, :, :-1]#.squeeze(1)
        inp[:, :, 0] /= 1000
        predictions = net(inp.float().to(device)).squeeze()#.argmax()
        targets = d[:, :, -1][:, 0].long().to(device)

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

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

    return accuracy
```

Run

Network definition

```
# Hyperparameters
seq_len = 5
measurement = 'C'
batch_size = 256
nr_epochs = 250
lr = 0.001
hidden_size_linear = 128
```

```
hidden_layers = 2
step = 0.05
```

```
def create_data_loaders(elements, splits, seq_len, validation=False, measurement='G', s
    dc = DatasetCreator(elements=elements,  splits=splits, validation=validation, seq_l
    train_dataset, test_dataset, val_dataset = dc.get_datasets()

    # Create the DataLoaders
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=shuffle)
    test_loader = DataLoader(test_dataset, batch_size=1, shuffle=shuffle)
    if val_dataset:
        val_loader = DataLoader(val_dataset)
    else:
        val_loader = None

    return train_loader, test_loader, val_loader
```

```
# Create the network
t = True
net = ElementClassifier(train, in_features=seq_len * 2, hidden_size_linear=hidden_size_

# 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)

optimizer = Adam(net.parameters(), lr=lr)
```

Train

```
elements = ['Ag', 'Au', 'Fe', 'Cu', 'Pt', 'Mg', 'C', 'Ca', 'Co', 'S',
            'Rh', 'Ru', 'Ti', 'W', 'Yb', 'Zn', 'Eu', 'Ga', 'Dy', 'Cd',
            'Er', 'B', 'Hg', 'Ho', 'In', 'Na', 'Nb', 'Nd', 'Sb', 'Sc',
            'As', 'Ba', 'Be', 'Bi', 'Ce', 'Cr', 'Cs', 'Ge', 'Hf', 'O',
            'Ni', 'Np', 'Pa', 'Pb', 'Pr', 'Pd', 'Rb', 'Al', 'Y',
            'Gd', 'Ir', 'K', 'La', 'Li', 'Lu', 'Mn', 'Mo', 'N', 'Os',
            'Pu', 'Re', 'Se', 'Sm', 'Sn', 'Sr', 'Ta', 'Tb', 'Tc', 'Te',
            'Th', 'Tl', 'Tm', 'U', 'V', 'Am', 'Zr', 'Si', 'P']

print(len(elements))

elements = None

train_loader, test_loader, val_loader = create_data_loaders(elements, (0.8, 0.2), seq_l
```

```
78
(411319, 5, 3)
```

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

```
tensor(77) tensor([77])
tensor(23) tensor([23])
```

```
tensor(67) tensor([67])
tensor(67) tensor([67])
tensor(63) tensor([63])
tensor(3) tensor([3])
tensor(56) tensor([56])
tensor(14) tensor([14])
tensor(15) tensor([15])
tensor(15) tensor([15])
tensor(38) tensor([38])
tensor(21) tensor([21])
tensor(5) tensor([5])
tensor(59) tensor([59])
tensor(76) tensor([76])
tensor(39) tensor([39])
tensor(53) tensor([53])
tensor(61) tensor([61])
tensor(24) tensor([24])
tensor(56) tensor([56])
tensor(58) tensor([58])
tensor(39) tensor([39])
tensor(41) tensor([41])
tensor(48) tensor([48])
tensor(8) tensor([8])
tensor(31) tensor([31])
tensor(41) tensor([41])
tensor(32) tensor([32])
tensor(39) tensor([39])
tensor(2) tensor([2])
tensor(72) tensor([72])
tensor(38) tensor([38])
tensor(72) tensor([72])
tensor(67) tensor([67])
tensor(29) tensor([29])
tensor(54) tensor([54])
tensor(15) tensor([15])
tensor(32) tensor([32])
tensor(36) tensor([36])
tensor(49) tensor([49])
tensor(39) tensor([39])
tensor(44) tensor([44])
tensor(26) tensor([26])
tensor(4) tensor([4])
tensor(6) tensor([6])
tensor(20) tensor([20])
tensor(49) tensor([49])
tensor(18) tensor([18])
tensor(20) tensor([20])
tensor(26) tensor([26])
tensor(61) tensor([61])
tensor(24) tensor([24])
tensor(31) tensor([31])
tensor(47) tensor([47])
tensor(12) tensor([12])
tensor(1) tensor([1])
Test accuracy:  0.9782315040670159
0.9782315040670159
```

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

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

```
1
```

✓ 1h 32m 6s completed at 5:16 PM