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

        # Add random noise to heat capacity values to simulate real measurements
        d[:, :, 1] += torch.normal(mean=0, std=1.25, size=(d.shape[0], d.shape[1], ))

        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
```

```
from torch.nn.modules.activation import Softmax
# 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, losses, accuracies

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, Softmax()(predictions).amax(dim=-1)

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

    return accuracy
```

Run

Network definition

[ ] ↳ *3 cells hidden*

Train

```
elements = None

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

```
Dataset shape:  (411319, 5, 3)
```

```
best_net, losses, accuracies = train(net, optimizer, train_loader, test_loader, batch_s
test(best_net, test_loader)
```

```
Loss:  1.1421028763593615
Training accuracy:  0.5908066488540524
Epoch  320
Loss:  1.141913537884174
Training accuracy:  0.5908406856965032
Epoch  330
Loss:  1.1396847507303816
Training accuracy:  0.5914120184090693
Epoch  340
Loss:  1.1400897295933445
Training accuracy:  0.593140603764961
Epoch  350
Loss:  1.1400930326438945
Training accuracy:  0.5929728507557395
Epoch  360
Loss:  1.140848211587106
Training accuracy:  0.5910667875784975
Epoch  370
Loss:  1.1396018800597496
Training accuracy:  0.591258852618041
Epoch  380
Loss:  1.1361760502100435
Training accuracy:  0.5928974834617414
Epoch  390
Loss:  1.1410264486322954
Training accuracy:  0.5923917932310445
Epoch  400
Loss:  1.1370598457396661
Training accuracy:  0.5933472560226977
Epoch  410
Loss:  1.1395953206557552
Training accuracy:  0.5924185364643987
Epoch  420
Loss:  1.1356196806122223
Training accuracy:  0.5925255093978153
Epoch  430
Loss:  1.13567030541112
Training accuracy:  0.592980144364836
Epoch  440
Loss:  1.1339238509489427
Training accuracy:  0.593993956029262
Epoch  450
Loss:  1.1356378349741572
Training accuracy:  0.5933788616621163
Epoch  460
Loss:  1.1345237681564517
Training accuracy:  0.5930628052679308
Epoch  470
Loss:  1.1360714263662315
```

```
Loss:  1.1366714165661515
Training accuracy:  0.5936219819653359
Epoch  480
Loss:  1.137068462598079
Training accuracy:  0.5940352864808093
Epoch  490
Loss:  1.1347713107341701
Training accuracy:  0.5937727165533321
Test accuracy:  0.8481064323961867
0.8481064323961867
```

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

```
torch.save(best_net, 'ElementClassifier_var_8481.pth')
```

Test on Barin data

```
net = torch.load('/content/ElementClassifier_9782_3.pth').to(device)
```

```
inp = torch.tensor([[[ .3,   11.403],
                     [ .7,   22.25],
                     [ .8,   23.364],
                     [.9,   24.248],
                     [1.0,   24.979]]]).to(device)
```

```
out = net(inp)
```

```
print(Encoder()(out.argmax(dim=-1).item()))
print(Softmax(dim=-1)(out).amax(dim=-1))
```

```
B
tensor([1.0000], grad_fn=<AmaxBackward0>)
```

```
test(net, test_loader)
```

```
Test accuracy:  0.9772111058201572
0.9772111058201572
```

✓ 0s completed at 4:12 PM ● ×