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=0.75, 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

```
# Hyperparameters
seq_len = 5
measurement = 'C'
batch_size = 256
batch_size = 256
nr_epochs = 500
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', u
    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 = 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:  0.6527893115032067
Training accuracy:  0.7493988850502894
Epoch  320
Loss:  0.6489172258414908
Training accuracy:  0.749306499335066
Epoch  330
Loss:  0.6513613111848926
Training accuracy:  0.7489199380529468
Epoch  340
Loss:  0.6488454878126232
```

```
Loss:  0.6488454878126232
Training accuracy:  0.7508721940878005
Epoch  350
Loss:  0.6497220686117017
Training accuracy:  0.7504686143844559
Epoch  360
Loss:  0.6475774214163135
Training accuracy:  0.7512757737911451
Epoch  370
Loss:  0.6476132421387452
Training accuracy:  0.7504637519783914
Epoch  380
Loss:  0.6497419062577647
Training accuracy:  0.749739253474797
Epoch  390
Loss:  0.6512461012112256
Training accuracy:  0.749430490689708
Epoch  400
Loss:  0.6452652581393088
Training accuracy:  0.7515602245459121
Epoch  410
Loss:  0.6463633147024856
Training accuracy:  0.7512757737911451
Epoch  420
Loss:  0.6464975365456692
Training accuracy:  0.7510861399546338
Epoch  430
Loss:  0.6462330725078784
Training accuracy:  0.7514581140185598
Epoch  440
Loss:  0.6446551240669991
Training accuracy:  0.7511104519849557
Epoch  450
Loss:  0.6480033389143716
Training accuracy:  0.7504977888208422
Epoch  460
Loss:  0.6454645953771114
Training accuracy:  0.7513997651457871
Epoch  470
Loss:  0.642114242301198
Training accuracy:  0.7529557350863928
Epoch  480
Loss:  0.6420811612968054
Training accuracy:  0.7531988553896124
Epoch  490
Loss:  0.6418265373479081
Training accuracy:  0.7522020621464119
Test accuracy:  0.891244983041953
0.891244983041953
```

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

```
torch.save(best_net, 'ElementClassifier_var_8912.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.], grad_fn=<AmaxBackward0>)
```

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

```
Test accuracy:  0.9772111058201572
0.9772111058201572
```

✓ 0s completed at 1:13 PM