

Triplet_auxiliary_rework

August 6, 2023

1 run load_data.ipynb BEFORE running this!

```
[2]: import numpy as np
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
s = {
    'problem'          : "regression",
    'approach'         : "metric learning/non-parametric",
    'algorithm'        : "triplet network",
    'input'            : "samples from a distribution",
    'input type'       : "vectors",
    'input meaning'    : "spectrum",
    'output'           : "samples from a distribution",
    'output type'      : "one number",
    'output meaning'   : "temperature or pressure, depending on distribution",
    'learning rate'    : 1e-4,
    'input dimension'  : 10000,
    'output dimension' : 1,
    'feature dimension': 300,
    'epoch'            : 1000,
    'epoch-development': 1,
    'cross validation round': 16,
    'cross validation round-development' : 1,
    'batch size'       : 64,
    'best model folder' : 'triplet_best_model/'
}
# https://arxiv.org/pdf/1412.6622.pdf
import data_accessor as acc
data_names_list = [
    'temperature_230509_discrete',
    'pressure_230516_discrete'
]
data_dictionary = acc.setup(data_names_list)
```

```
loading temperature_230509_discrete_____
input shape (number, dimension): (6000, 10000)
label shape (number, dimension): (6000, 1)
there are 16 folds
```

```

4200 for training, 600 for validating, 1200 for testing
loading pressure_230516_discrete_____
input shape (number, dimension): (5000, 10000)
label shape (number, dimension): (5000, 1)
there are 16 folds
3500 for training, 500 for validating, 1000 for testing

```

```

[3]: import torch.nn as nn
class TripletNetwork(torch.nn.Module):
    """ Input: pos, neg, anchor, anchor_label
        Output: pos_prediction, neg_prediction"""
    def __init__(self, device, input_dimension, feature_dimension,
↪output_dimension):
        super().__init__()
        self.input_dimension = input_dimension
        self.feature_dimension = feature_dimension
        self.output_dimension = output_dimension
        self.device = device
        self.feature_sequential = torch.nn.Sequential(
            torch.nn.Linear(self.input_dimension, 3000),
            nn.ReLU(),
            torch.nn.Linear(3000, 600),
            nn.ReLU(),
            torch.nn.Linear(600, 600),
            nn.ReLU(),
            torch.nn.Linear(600, 300),
            nn.ReLU(),
            torch.nn.Linear(300, self.feature_dimension)
        )
        self.auxiliary_sequential = torch.nn.Sequential(
            torch.nn.Linear(self.feature_dimension, 100),
            nn.ReLU(),
            torch.nn.Linear(100, 100),
            nn.ReLU(),
            torch.nn.Linear(100, self.output_dimension)
        )
        self.to(device)
        self.float()
    def forward(self, pos, neg, anchor, anchor_label):
        feature_pos = self.feature_sequential(pos)
        feature_neg = self.feature_sequential(neg)
        feature_anchor = self.feature_sequential(anchor)
        feature_space_difference_pos_anchor = feature_pos - feature_anchor
        feature_space_difference_neg_anchor = feature_neg - feature_anchor
        label_space_difference_pos_anchor = self.
↪auxiliary_sequential(feature_space_difference_pos_anchor)

```

```

        label_space_difference_neg_anchor = self.
↪auxiliary_sequential(feature_space_difference_neg_anchor)
        prediction_pos = anchor_label + label_space_difference_pos_anchor
        prediction_neg = anchor_label + label_space_difference_neg_anchor
        return prediction_pos, prediction_neg

```

```

[4]: from tools import SaveBestModel, PatienceEarlyStopping, Scheduler, plot_losses
class Manager:
    """ DOES: train & evaluate a Siamese network
        """
    def __init__(self, epoch, cross_validation_round):
        self._network = TripletNetwork(device, s['input dimension'], s['feature_
↪dimension'], s['output dimension'])
        self._network.apply(self.initializer)
        self._learning_rate = s['learning rate']
        self._optimizer = torch.optim.Adam(
            params=self._network.parameters(), lr=self._learning_rate,
            weight_decay=3e-3)
        self._energy = nn.MSELoss()
        self._train_loss = []
        self._valid_loss = []
        self._test_loss = []
        self._epoch = epoch
        self._stopper = PatienceEarlyStopping(patience=5, min_delta=5e-7)
        self._cross_validation_round = cross_validation_round
        self._saver = SaveBestModel(s['best model folder'])
        self._scheduler = Scheduler(optimizer=self._optimizer,
            minimum_learning_rate=1e-6, patience=5, factor=0.5)
    def initializer(self, layer):
        if type(layer) == nn.Linear:
            nn.init.kaiming_normal_(layer.weight) # normal version
    def _step(self, job):
        pos, pos_label, neg, neg_label, anchor, anchor_label = job
        pos_prediction, neg_prediction = self._network(pos, neg, anchor,
↪anchor_label)
        pos_loss = self._energy(pos_prediction, pos_label)
        neg_loss = self._energy(neg_prediction, neg_label)
        loss = (pos_loss + neg_loss) / 2.0
        return loss
    def train(self, train_dataloader, valid_dataloader):
        """ DOES: calculate loss from tasks
            NOTE: we have a BATCH of tasks here """
        for e in range(self._epoch):
            batch_train_loss = []
            for _, batch in enumerate(train_dataloader):
                self._optimizer.zero_grad()
                loss = self._step(batch)

```

```

        loss.backward()
        self._optimizer.step()
        batch_train_loss.append(loss.item())
    self._train_loss.append(np.mean(batch_train_loss))
    batch_valid_loss = []
    with torch.no_grad():
        for _, batch in enumerate(valid_dataloader):
            loss = self._step(batch)
            batch_valid_loss.append(loss.item())
        self._valid_loss.append(np.mean(batch_valid_loss))
        # saving, early stopping, scheduler for EACH epoch!
        self._saver(current_loss=np.mean(batch_valid_loss), model=self.
↪_network, round=self._cross_validation_round)
        self._scheduler(np.mean(batch_valid_loss))
        self._stopper(np.mean(batch_valid_loss))
        if self._stopper.early_stop == True:
            print(f"EARLY STOPPING @ epoch {e}")
            break
        # summary printout, after we're done with epochs
        print(f"min train loss: {np.min(self._train_loss)}")
        print(f"min valid loss: {np.min(self._valid_loss)}")
        plot_losses(self._train_loss, self._valid_loss, self.
↪_cross_validation_round)
    return np.min(self._valid_loss)
def test(self, test_dataloader):
    with torch.no_grad():
        batch_test_loss = []
        for _, batch in enumerate(test_dataloader):
            loss = self._step(batch)
            batch_test_loss.append(loss.item())
        self._test_loss.append(np.mean(batch_test_loss))
    return np.min(self._test_loss)

```

```

[5]: import random
class TripletDataset(torch.utils.data.TensorDataset):
    """ input: input data
        label: label
        indices: indices used e.g. training indices
        """
    def __init__(self, input, label, indices, device):
        self.input = torch.Tensor(input).to(device)
        self.label = torch.Tensor(label).to(device)
        self.access_indices = indices
        self.indices = range(len(self.access_indices))
    def __len__(self):
        return len(self.indices)
    def __getitem__(self, index):

```

```

        index = self.access_indices[index]
        anchor_index = random.choice(self.access_indices)
        neg_index = random.choice(self.access_indices)
        pos = self.input[index]
        pos_label = self.label[index]
        anchor = self.input[anchor_index]
        anchor_label = self.label[anchor_index]
        neg = self.input[neg_index]
        neg_label = self.label[neg_index]
        return pos, pos_label, neg, neg_label, anchor, anchor_label

```

```

[15]: from torch.utils.data import DataLoader
from tools import SaveBestCrossValidationModel
from Style import TextColor
class CrossValidator:
    """ number_of_cross_validation_rounds
        \nnumber_of_epochs
        \nsaver (actual object)
        \ndataset (the function of the object)
    """

    def __init__(self, number_of_cross_validation_rounds, number_of_epochs,
↪saver, dataset, datas, data_dictionary, settings):
        self.number_of_cross_validation_rounds =
↪number_of_cross_validation_rounds
        self.number_of_epochs = number_of_epochs
        self.saver = saver
        self.cross_validation_loss = []
        self.dataset = dataset
        self.datas = datas
        self.data_dictionary = data_dictionary
        self.settings = settings
        print(f"{TextColor.Bold}{TextColor.BrightGreen_text}-----CROSS_
↪VALIDATION-----{TextColor.End}")
        print(f"Cross-validation rounds: {self.
↪number_of_cross_validation_rounds}")
        print(f"Epochs: {self.number_of_epochs}")
        print(f"Datas to learn: ")
        for index, data in enumerate(self.datas):
            print(f"\t{index}: {data}")
        def complete_notify(self):
            # unicode https://www.geeksforgeeks.org/python-program-to-print-emojis/
            print(f"\U0001f607 {TextColor.Bold}{TextColor.BrightGreen_text}TRAINing_
↪COMPLETE-----{TextColor.End}")
        def single_task_train(self, data_index): # magenta
            print(f"{TextColor.Bold}{TextColor.Magenta_text}SINGLE_
↪TASK-----{TextColor.End}")

```

```

print(f"we're learning: {self.datas[data_index]}")
self.cross_validation_loss = []
for round_index in range(self.number_of_cross_validation_rounds):
    print(f">round {round_index}")
    network_object = Manager(self.number_of_epochs, round_index)
    valid_loss = network_object.train( # DON'T do so in separate_
↪function
        DataLoader(self.dataset(
            self.data_dictionary[self.datas[data_index]]['data'],
            self.data_dictionary[self.datas[data_index]]['label'],
            self.data_dictionary[self.datas[data_index]]['train_
↪indices'] [round_index],
            device=device,), shuffle=False, batch_size=self.settings['batch_
↪size']),
        DataLoader(self.dataset(
            self.data_dictionary[self.datas[data_index]]['data'],
            self.data_dictionary[self.datas[data_index]]['label'],
            self.data_dictionary[self.datas[data_index]]['valid_
↪indices'] [round_index],
            device=device,), shuffle=False, batch_size=self.settings['batch_
↪size']))
    self.saver(current_loss=valid_loss, round=round_index)
    self.cross_validation_loss.append(valid_loss)
    print(f"{TextColor.Bold}{TextColor.BrightGreen_text}BEST{TextColor.End}_
↪model: {self.saver.best_model_name} with {self.saver.current_best_loss}")
    print(f"trained on {self.datas[data_index]}")
    print(f"Aggregate performance: Valid loss mean {np.mean(self.
↪cross_validation_loss)}, std {np.std(self.cross_validation_loss)}")
    def multi_task_train_sequential(self): # blue
        """ learn ONE data at a time, need to reset model in between"""
        print(f"{TextColor.Bold}{TextColor.Blue_text}MULTI TASK,
↪Sequential_____ {TextColor.End}")
        print(f"we're learning: multiple tasks")
        print(f"given [1, 2, 3], [a, b, c]: learn [1, 2, 3], reset model, learn_
↪[a, b, c]")
        # for number, data in enumerate(self.datas):
        #     print(f"\t{number}: {data}")
        self.cross_validation_loss = [[] for data in datas]
        # print(self.cross_validation_loss)
        for round_index in range(self.number_of_cross_validation_rounds):
            print(f">round {round_index}")
            network_object = Manager(self.number_of_epochs, round_index)
            for number, data in enumerate(self.datas):
                print(f"task {TextColor.Bold}{number}{TextColor.End}: {data}")
                valid_loss = network_object.train( # DON'T do so in separate_
↪function

```

```

        DataLoader(self.dataset(
            self.data_dictionary[data]['data'],
            self.data_dictionary[data]['label'],
            self.data_dictionary[data]['train indices'][round_index],
            device=device, shuffle=False, batch_size=self.
↪settings['batch size']),
        DataLoader(self.dataset(
            self.data_dictionary[data]['data'],
            self.data_dictionary[data]['label'],
            self.data_dictionary[data]['valid indices'][round_index],
            device=device, shuffle=False, batch_size=self.
↪settings['batch size']))
        # reset the network's saver, stopper, and losses (otherwise, it
↪early-stops rightaway)
        network_object._saver.reset()
        network_object._stopper.reset()
        network_object._train_loss = []
        network_object._valid_loss = []
        # reset auxiliary network
        network_object._network.auxiliary_sequential.
↪apply(network_object.initializer)
        self.cross_validation_loss[number].append(valid_loss)
        # picking best model by Current performance
        self.saver(current_loss=valid_loss, round=round_index)
        # print(self.cross_validation_loss)
        print(f"{TextColor.Bold}{TextColor.BrightGreen_text}BEST{TextColor.End}
↪model: {self.saver.best_model_name} with {self.saver.current_best_loss}")
        print(f"trained datas sequentially")
        print(f"Aggregate performance:")
        for index, cv_loss in enumerate(self.cross_validation_loss):
            print(f"{self.datas[index]}: Valid loss mean {np.mean(self.
↪cross_validation_loss[index])}, std {np.std(self.
↪cross_validation_loss[index])}")
        def multi_task_train_weave(self, weave): # blue
            """ learn altogether, using 'super dataset' woven from datasets
                weave: function for weaving"""
            print(f"\U0001f9f5{TextColor.Bold}{TextColor.Magenta_text}MULTI TASK,
↪Interweave_____ {TextColor.End}")
            print(f"we're learning: multiple tasks")
            print(f"given [1, 2, 3], [a, b, c]: learn [1, a, 2, b, 3, c], simple
↪handling of different counts")
            # for number, data in enumerate(self.datas):
            #     print(f"\t{number}: {data}")
            self.cross_validation_loss = []
            for round_index in range(self.number_of_cross_validation_rounds):
                print(f">round {round_index}")

```

```

        network_object = Manager(self.number_of_epochs, round_index)
        # t =
    ↪weave([data_dictionary[data]['data'][data_dictionary[data]['train_
    ↪indices'][round_index]] for data in datas])
        # print(f"shape of woven: {t.shape}")
        # i = sum([len(data_dictionary[data]['train indices'][0]) for data_
    ↪in datas])
        # print(f"length of woven: {i}")
        valid_loss = network_object.train( # DON'T do so in separate_
    ↪function
            DataLoader(self.dataset(
                weave([self.data_dictionary[data]['data'][self.
    ↪data_dictionary[data]['train indices'][round_index]] for data in self.
    ↪datas])),
                weave([self.data_dictionary[data]['label'][self.
    ↪data_dictionary[data]['train indices'][round_index]] for data in self.
    ↪datas])),
                range(sum([len(self.data_dictionary[data]['train indices'][0])_
    ↪for data in self.datas])),
                device=device,), shuffle=False, batch_size=self.settings['batch_
    ↪size']),
            DataLoader(self.dataset(
                weave([self.
    ↪data_dictionary[data]['data'][data_dictionary[data]['valid_
    ↪indices'][round_index]] for data in self.datas])),
                weave([self.
    ↪data_dictionary[data]['label'][data_dictionary[data]['valid_
    ↪indices'][round_index]] for data in self.datas])),
                range(sum([len(self.data_dictionary[data]['valid indices'][0])_
    ↪for data in self.datas])),
                device=device,), shuffle=False, batch_size=self.settings['batch_
    ↪size']))
        self.saver(current_loss=valid_loss, round=round_index)
        self.cross_validation_loss.append(valid_loss)
        print(f"{TextColor.Bold}{TextColor.BrightGreen_text}BEST{TextColor.End}_
    ↪model: {self.saver.best_model_name} with {self.saver.current_best_loss}")
        print(f"trained datas by weaving them")
        print(f"Aggregate performance: Valid loss mean {np.mean(self.
    ↪cross_validation_loss)}, std {np.std(self.cross_validation_loss)}")
        def test_all(self):
            """ test data one by one"""
            print(f"{TextColor.Bold}{TextColor.
    ↪Blue_text}TEST_____ {TextColor.End}")
            retained_loss = {}
            for data in self.datas:
                network_object = Manager(None, None)

```



```

        network_object._network.load_state_dict(torch.load(self.
↪settings['best model folder'] + self.saver.best_model_name))
        print(f"Testing {data}, loss: ", end=" ")
        # print(f"shape: {np.asarray(self.data_dictionary[data]['label'])[0:
↪4]}")

        test_loss = network_object.test(
            DataLoader(self.dataset(
                self.data_dictionary[data]['data'],
                self.data_dictionary[data]['label'],
                self.data_dictionary[data]['test indices'],
                device=device, shuffle=False, batch_size=self.settings['batch_
↪size']))

            # record results
            retained_loss[data] = test_loss
            print(f"{test_loss}")
            # print(f"{data} Test loss: {test_loss}")

datas = ['temperature_230509_discrete', 'pressure_230516_discrete']
# datas.reverse()
CVtor = CrossValidator(s['cross validation round'],
                        s['epoch'],
                        SaveBestCrossValidationModel(s['best model folder']),
                        TripletDataset,
                        datas,
                        data_dictionary,
                        s)
# CVtor.single_task_train(0)
CVtor.multi_task_train_sequential()
# from data import alternate_rows_itertools
# CVtor.multi_task_train_weave(alternate_rows_itertools)
CVtor.complete_notify()
CVtor.test_all()
# from IPython.display import Audio # auto-play doesn't work!
# sound_file = 'sound/IRWTS@UH.mp3'
# aud = Audio(sound_file, autoplay=True)
# display(aud)

```

-----CROSS VALIDATION-----

Cross-validation rounds: 16

Epochs: 1000

Datas to learn:

0: temperature_230509_discrete

1: pressure_230516_discrete

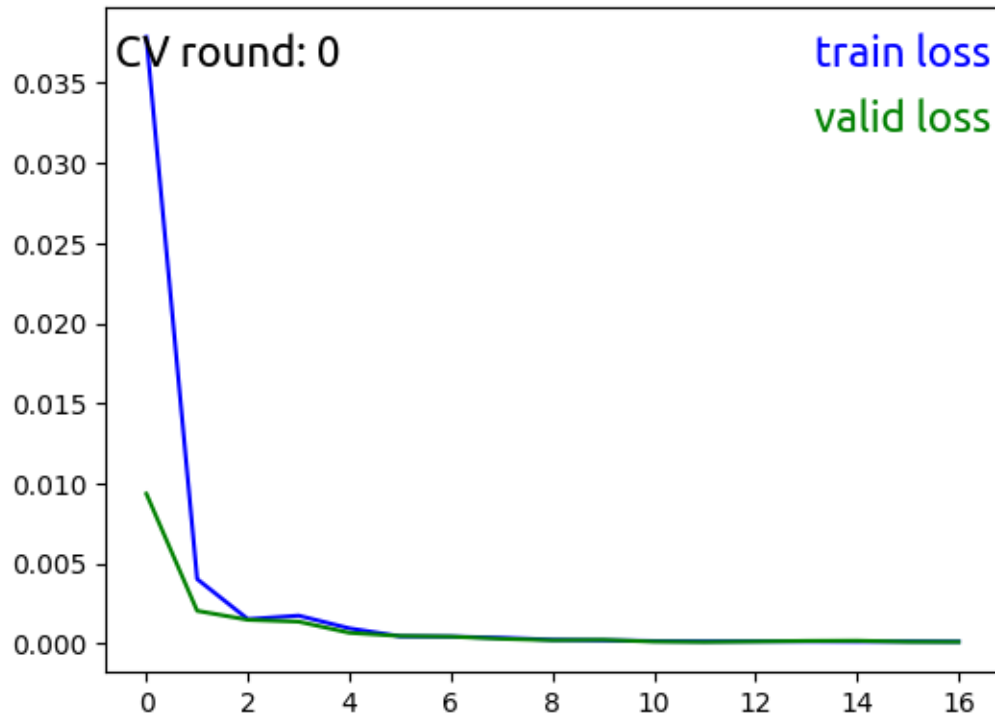
MULTI TASK, Sequential-----

we're learning: multiple tasks

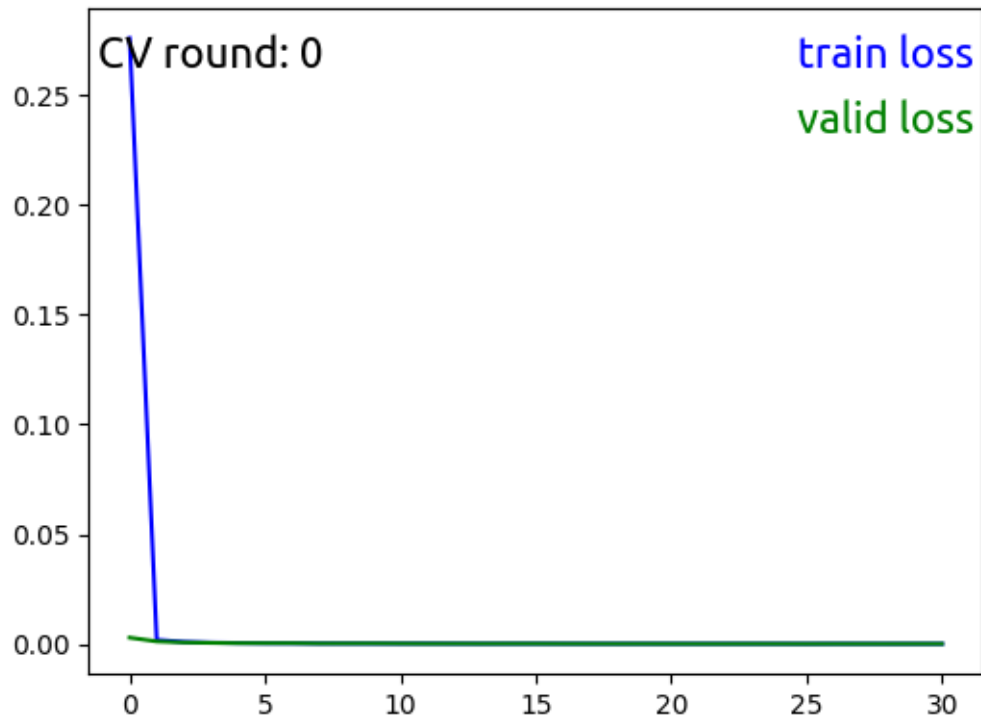
given [1, 2, 3], [a, b, c]: learn [1, 2, 3], reset model, learn [a, b, c]

>round 0

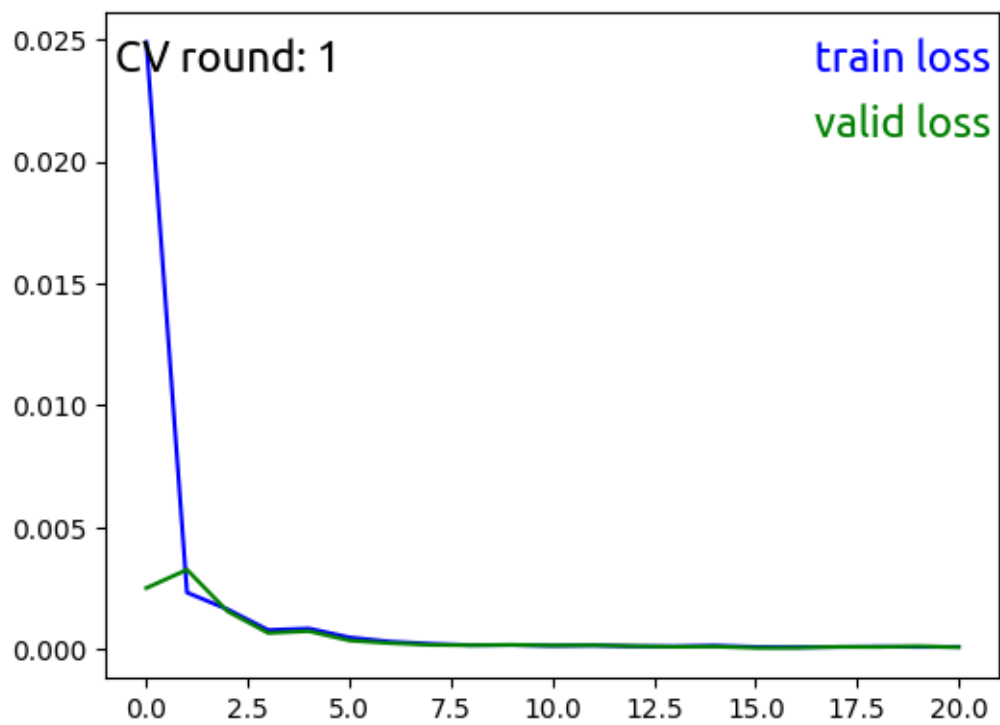
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 16
min train loss: 0.0001224700133730728
min valid loss: 0.00010025905139627866



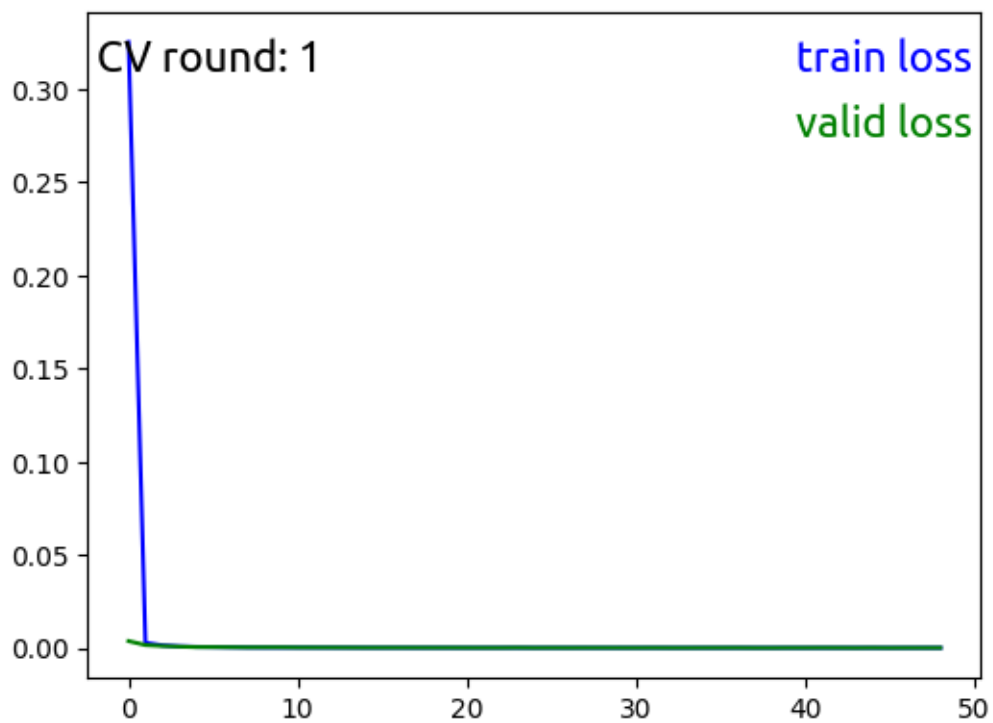
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 30
min train loss: 0.00018357168181947517
min valid loss: 0.00017814694001572207



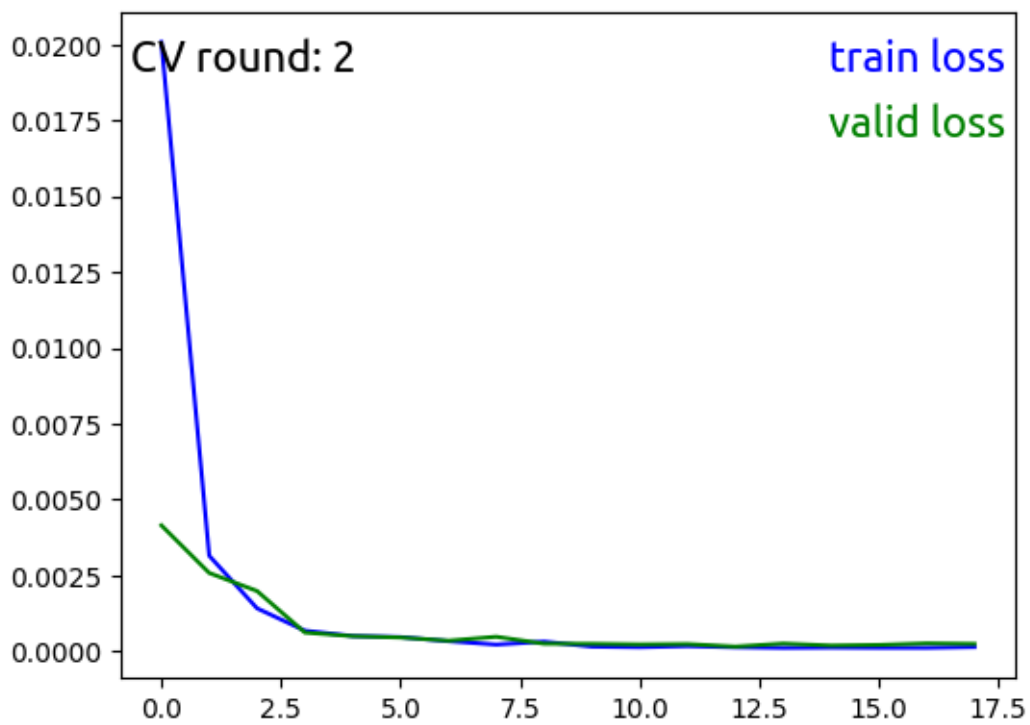
```
>round 1
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 20
min train loss: 9.673620297689922e-05
min valid loss: 6.691332600894384e-05
```



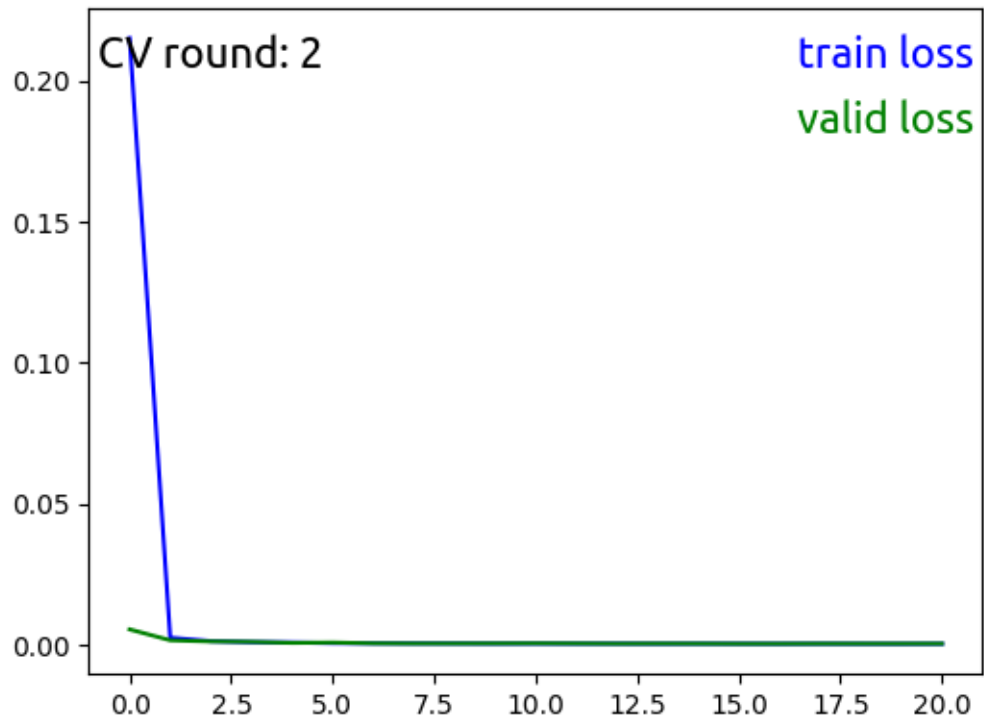
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 48
min train loss: 0.00022539867389820178
min valid loss: 0.00022469586838269606
```



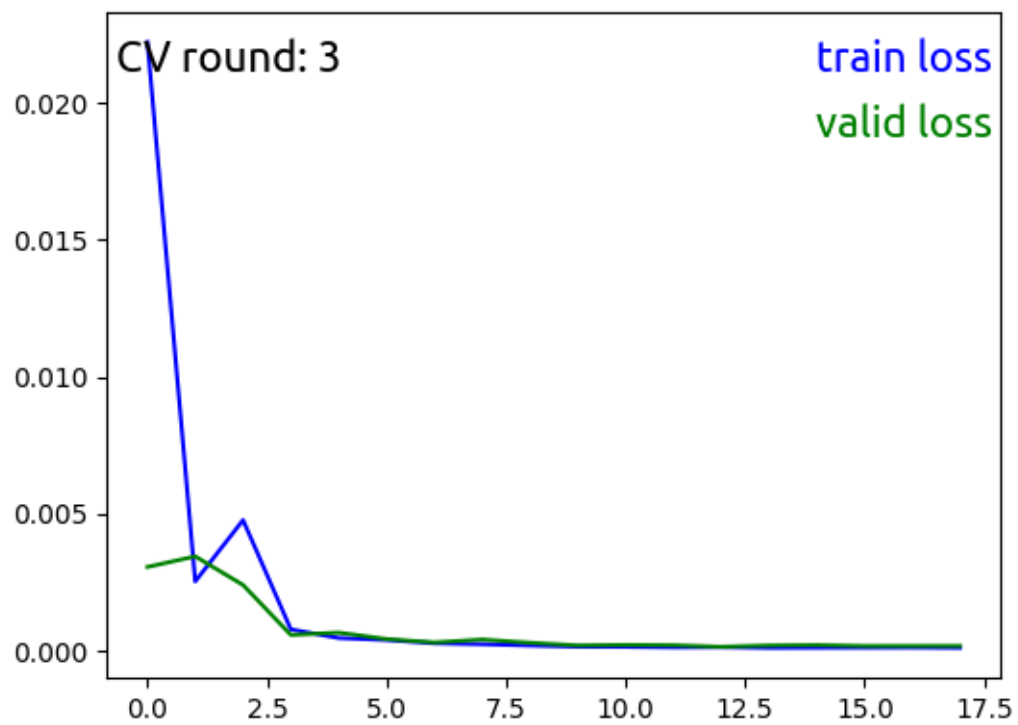
```
>round 2
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 17
min train loss: 0.00010604376020086364
min valid loss: 0.00014653648977400736
```



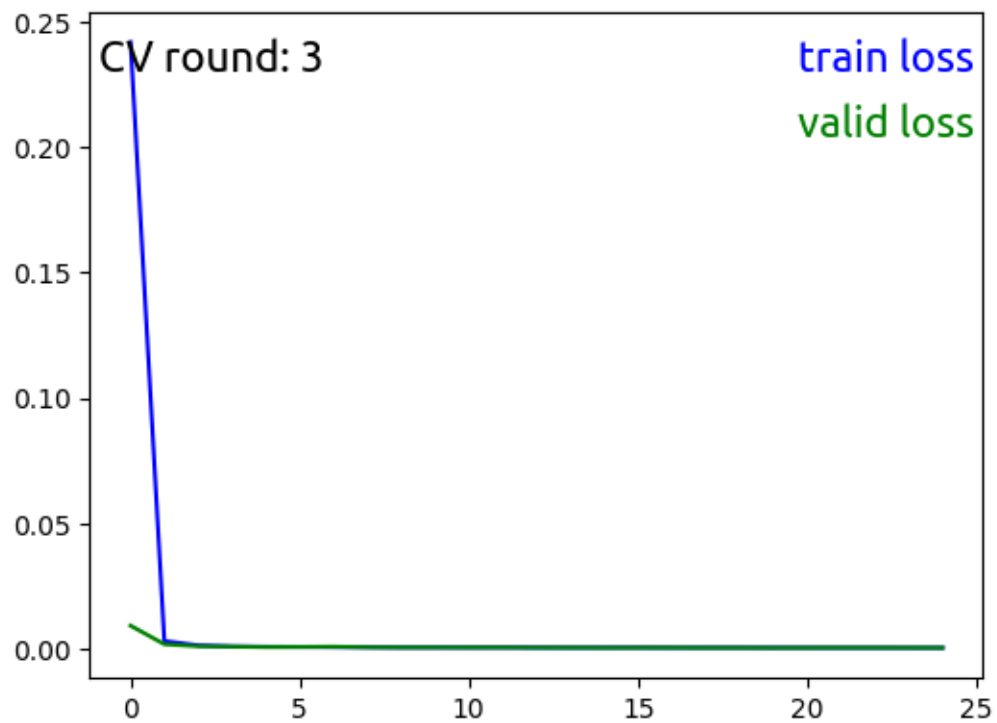
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 20
min train loss: 0.00020776298611467197
min valid loss: 0.00023197461996460333
```



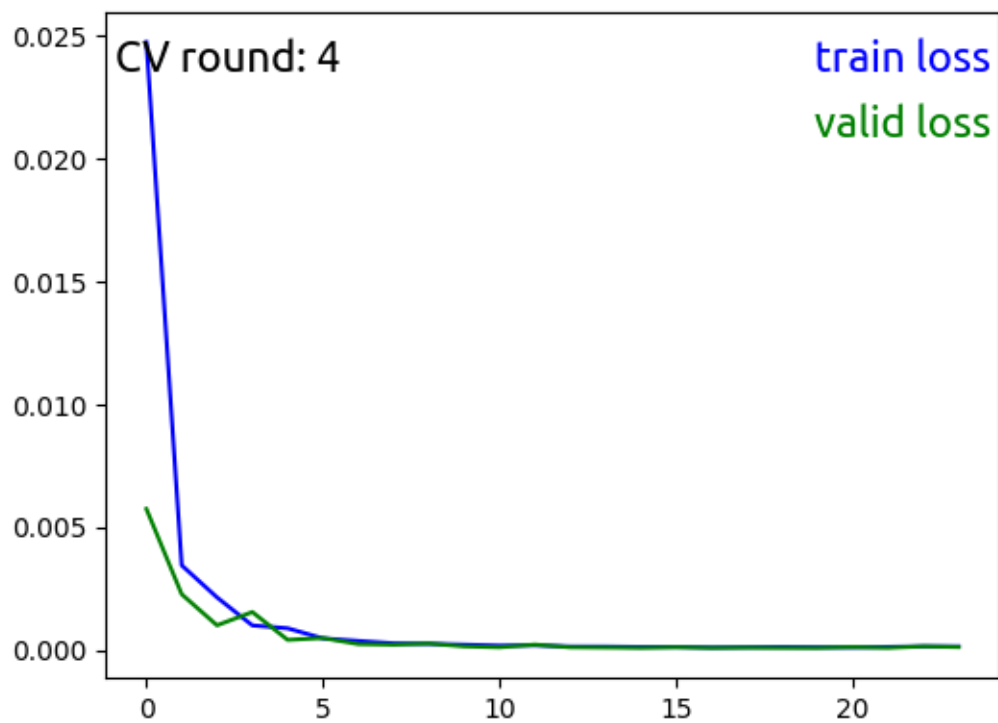
```
>round 3
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 17
min train loss: 0.00010220636734932971
min valid loss: 0.00014022536779521034
```



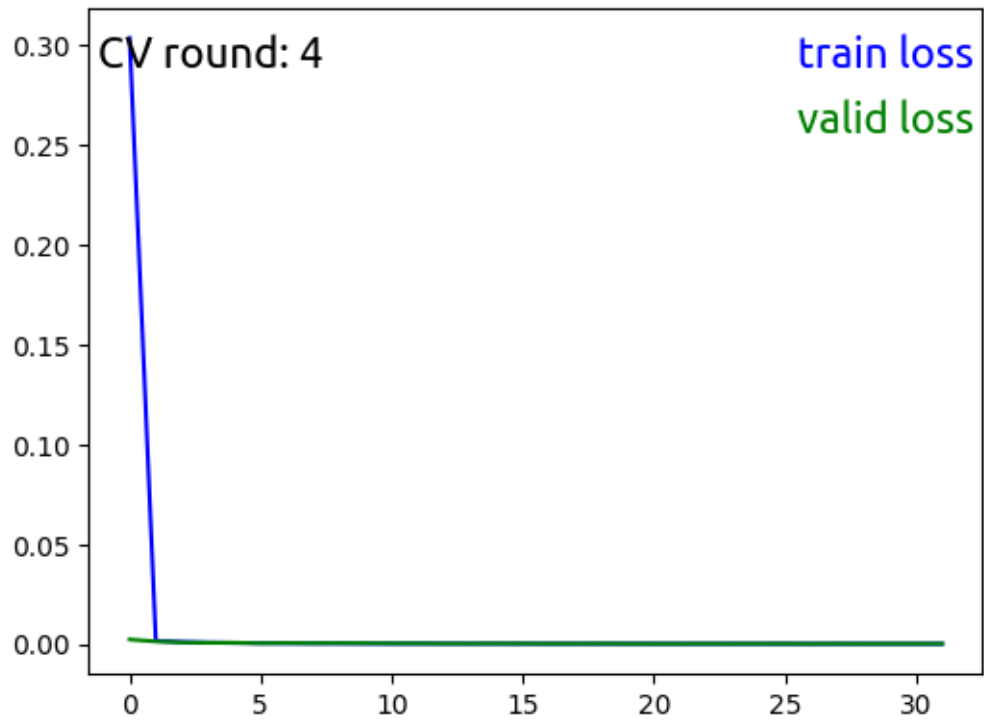
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 24
min train loss: 0.0002186207156310874
min valid loss: 0.00019839603010041174
```

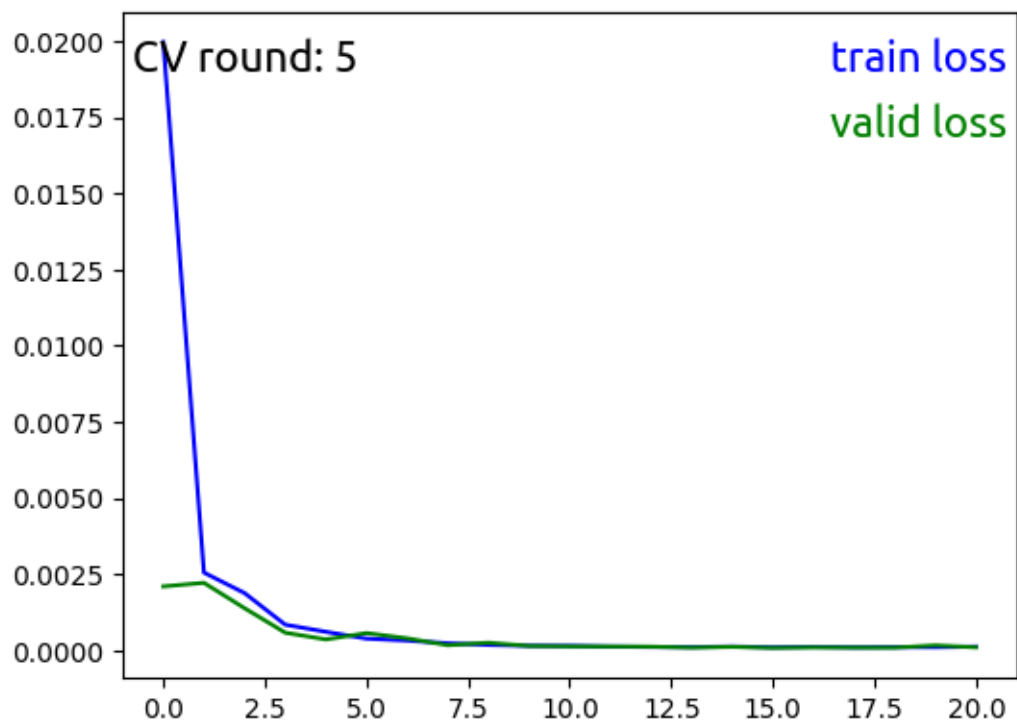
```
>round 4
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 23
min train loss: 9.438838005184097e-05
min valid loss: 7.931624022603501e-05
```



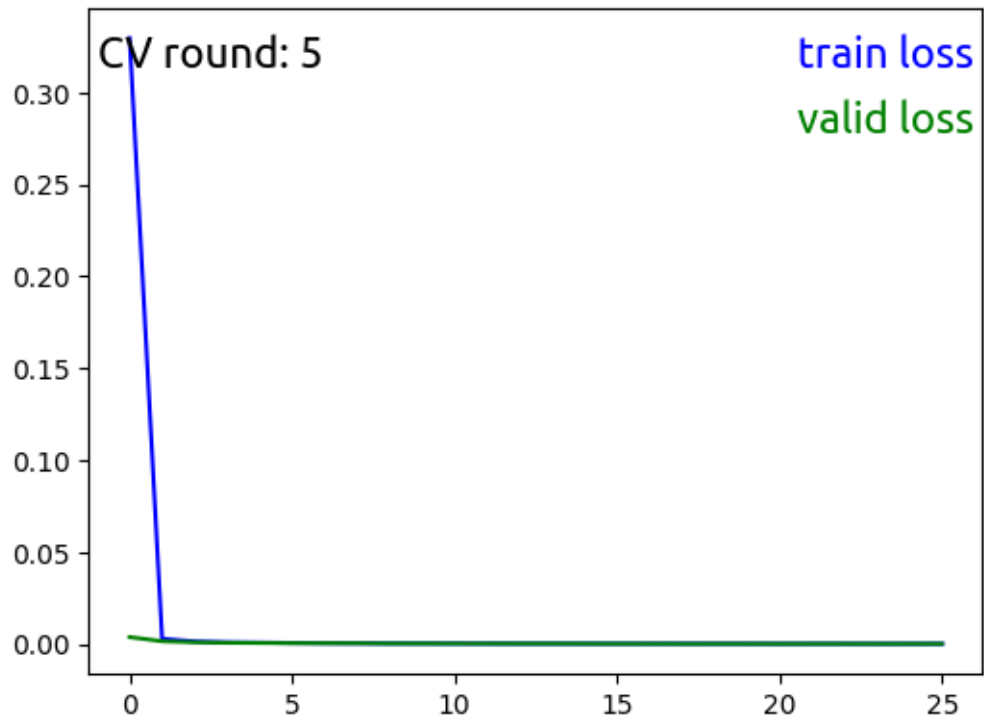
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 31
min train loss: 0.00019805361768124964
min valid loss: 0.00019160065858159214
```



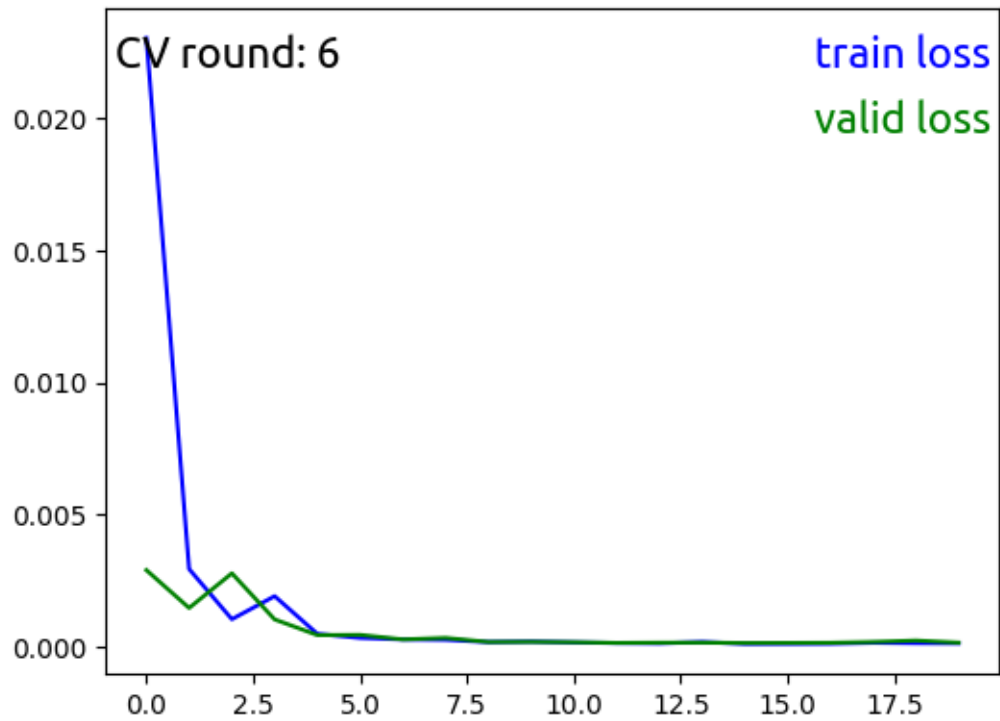
```
>round 5
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 20
min train loss: 0.0001028744785192733
min valid loss: 8.49402178573655e-05
```



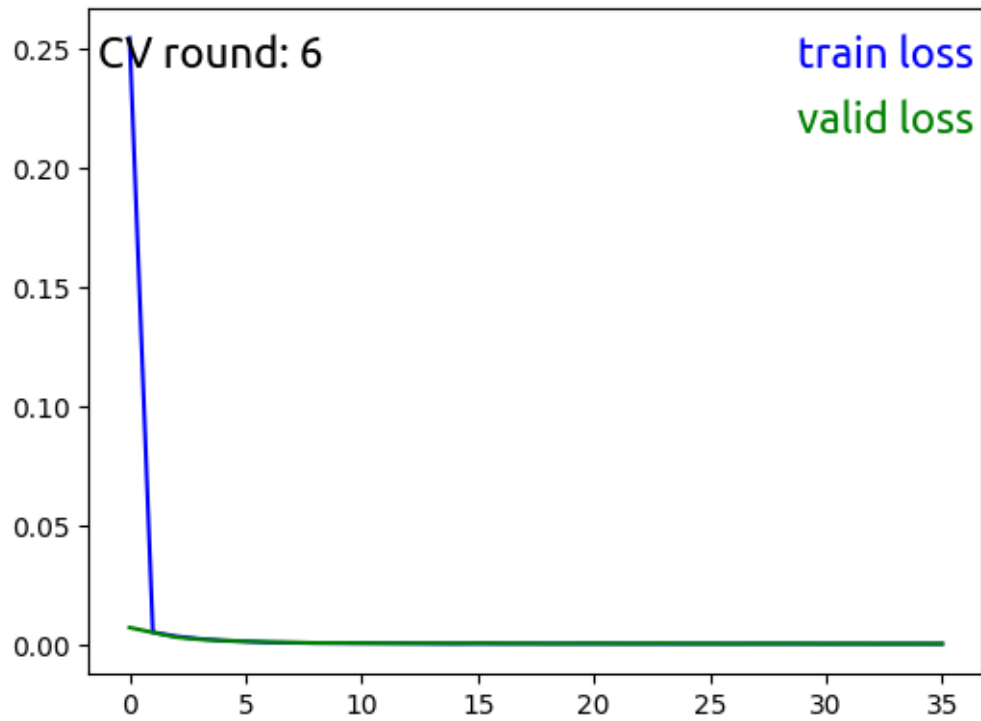
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 25
min train loss: 0.0002449916739186103
min valid loss: 0.00020882008357148152
```



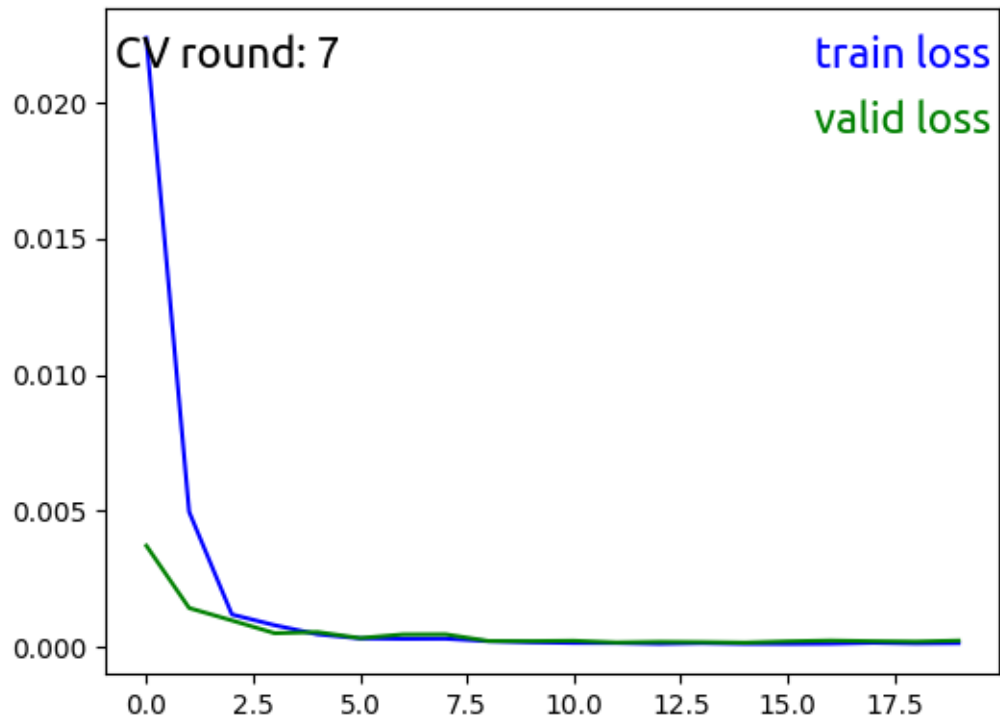
```
>round 6
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 19
min train loss: 0.00010887830577496172
min valid loss: 0.0001399072411004454
```



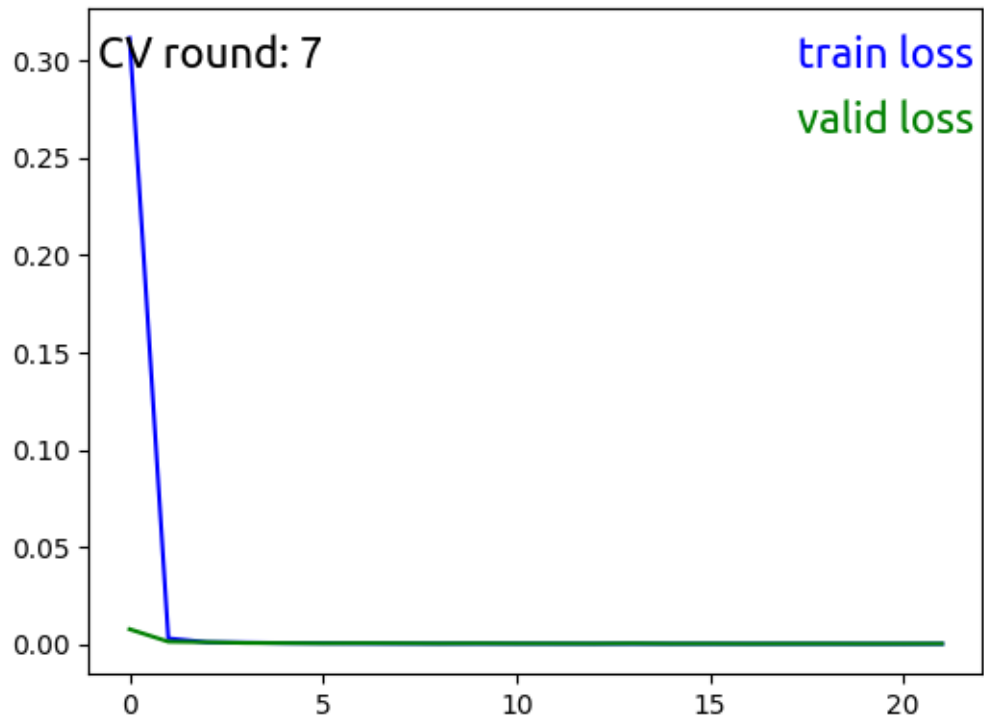
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 35
min train loss: 0.0003704686030025848
min valid loss: 0.0004331194795668125
```



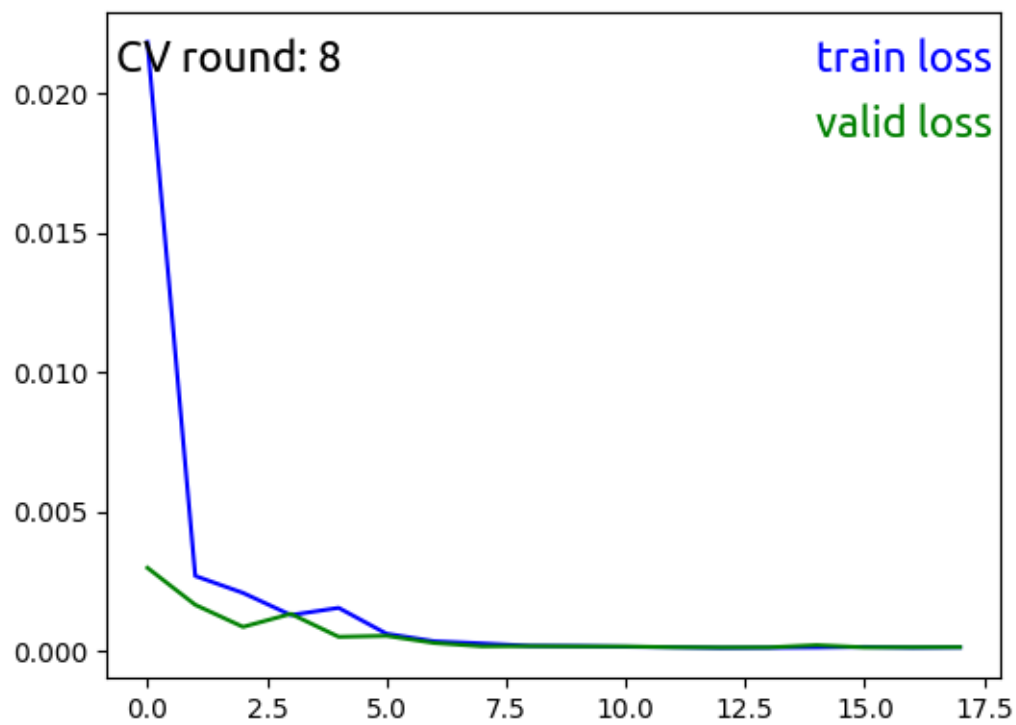
```
>round 7
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 19
min train loss: 0.00010870735422940925
min valid loss: 0.00015240115608321502
```



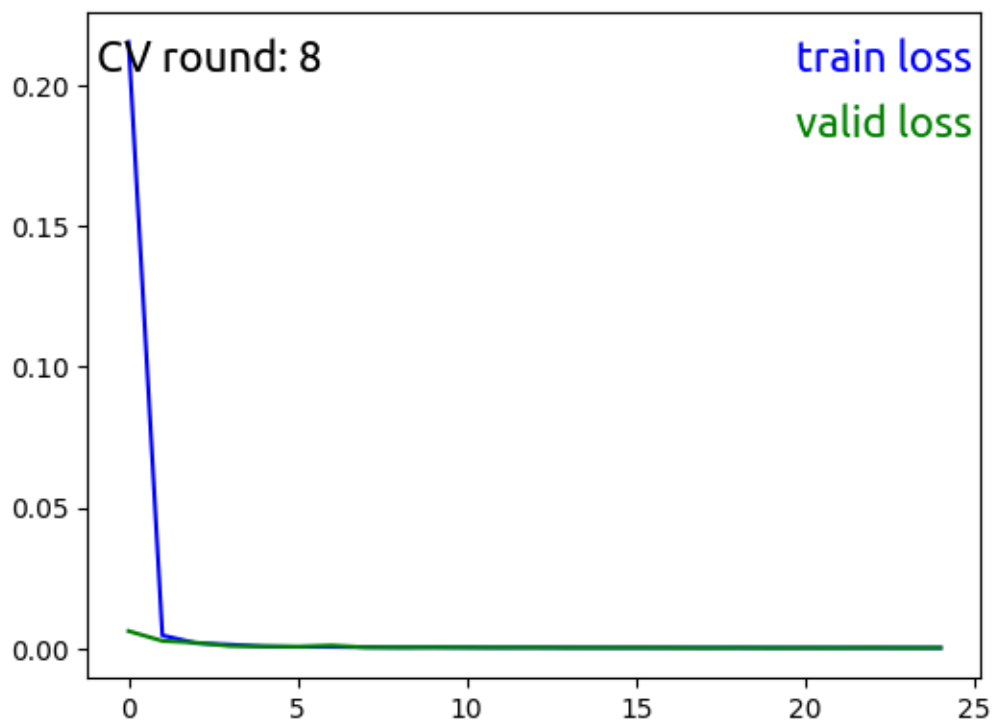
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 21
min train loss: 0.00023509826357688077
min valid loss: 0.0002652718703757273
```

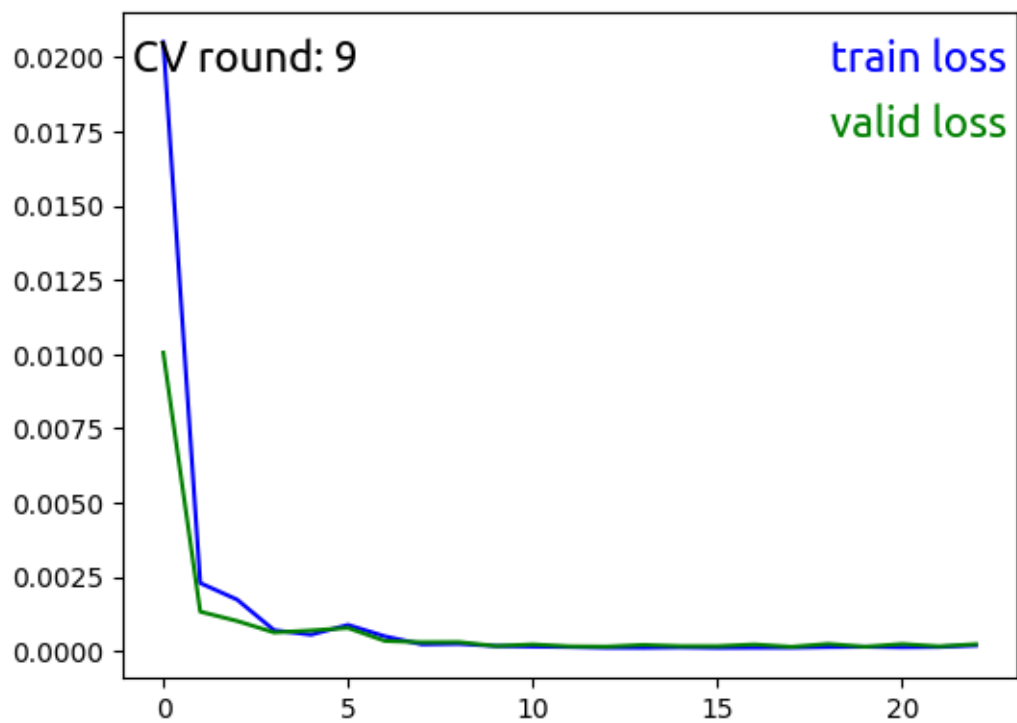
```
>round 8
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 17
min train loss: 0.0001102025515027697
min valid loss: 0.00012362040506559425
```



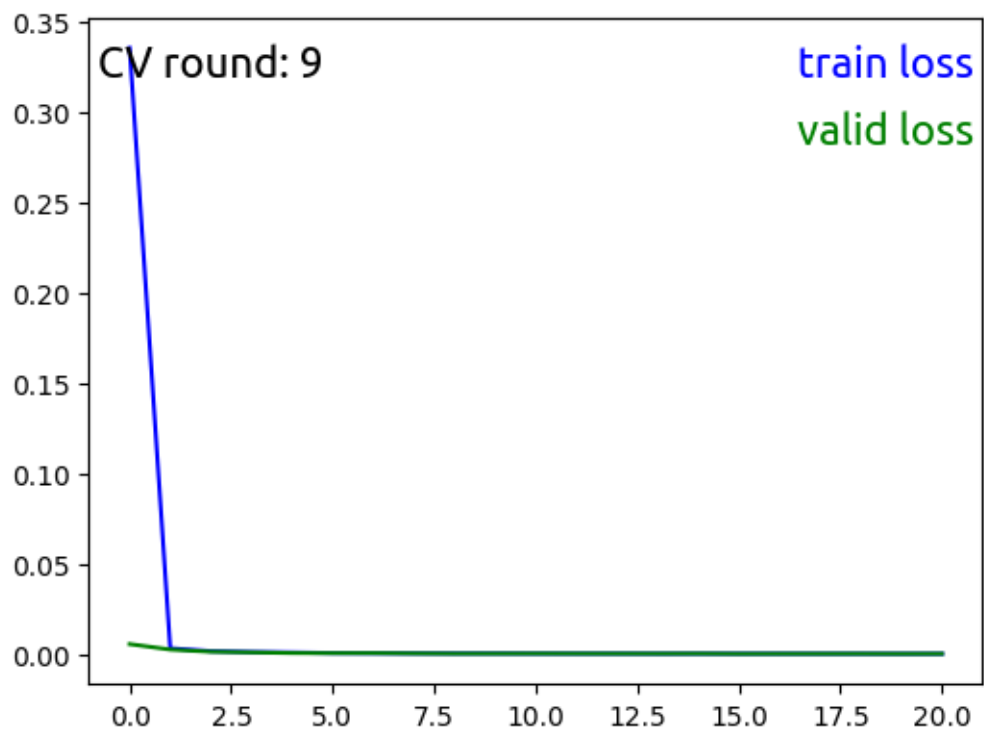
```
task 1: pressure_230516_discrete  
EARLY STOPPING @ epoch 24  
min train loss: 0.0002709423504050144  
min valid loss: 0.00028611443121917546
```



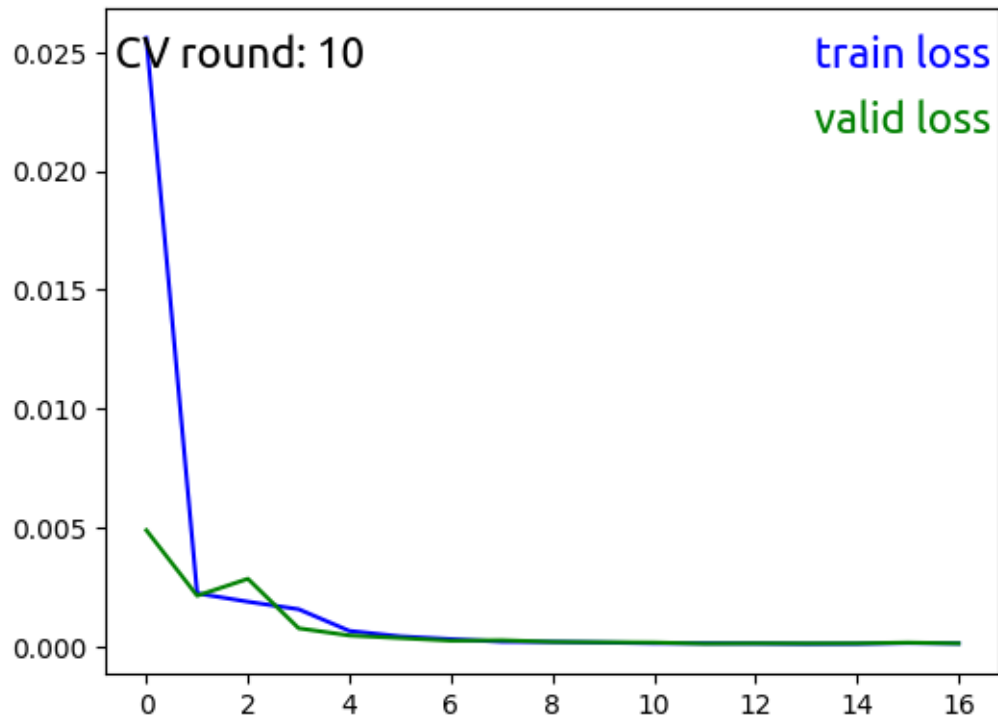
```
>round 9
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 22
min train loss: 0.0001144181952082418
min valid loss: 0.0001494111609645188
```



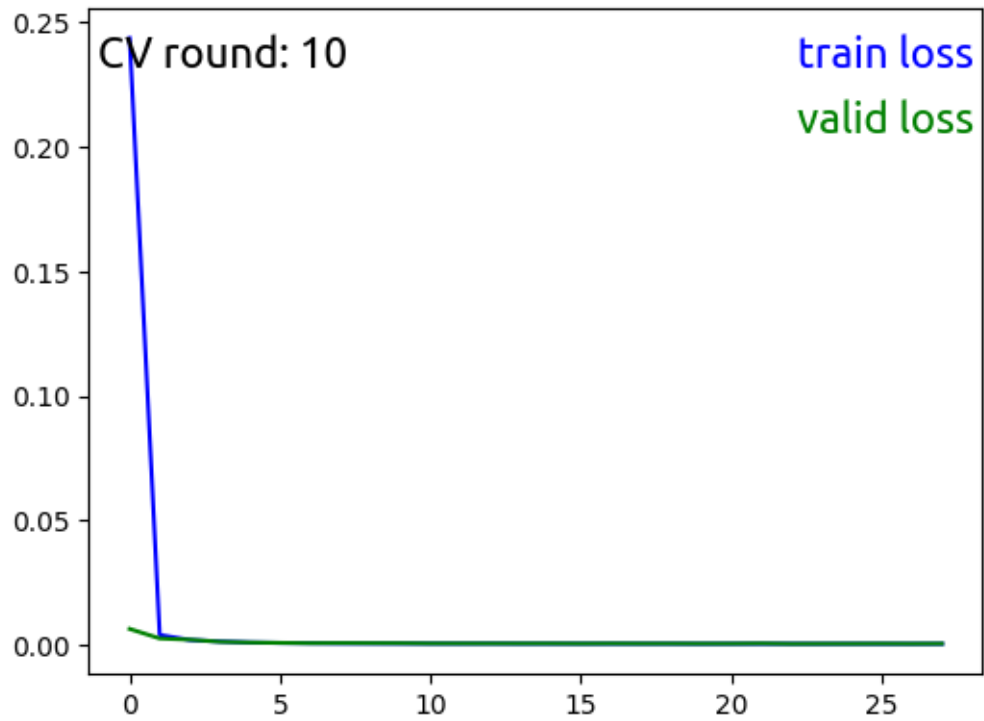
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 20
min train loss: 0.0002528889165577394
min valid loss: 0.00023360741033684462
```



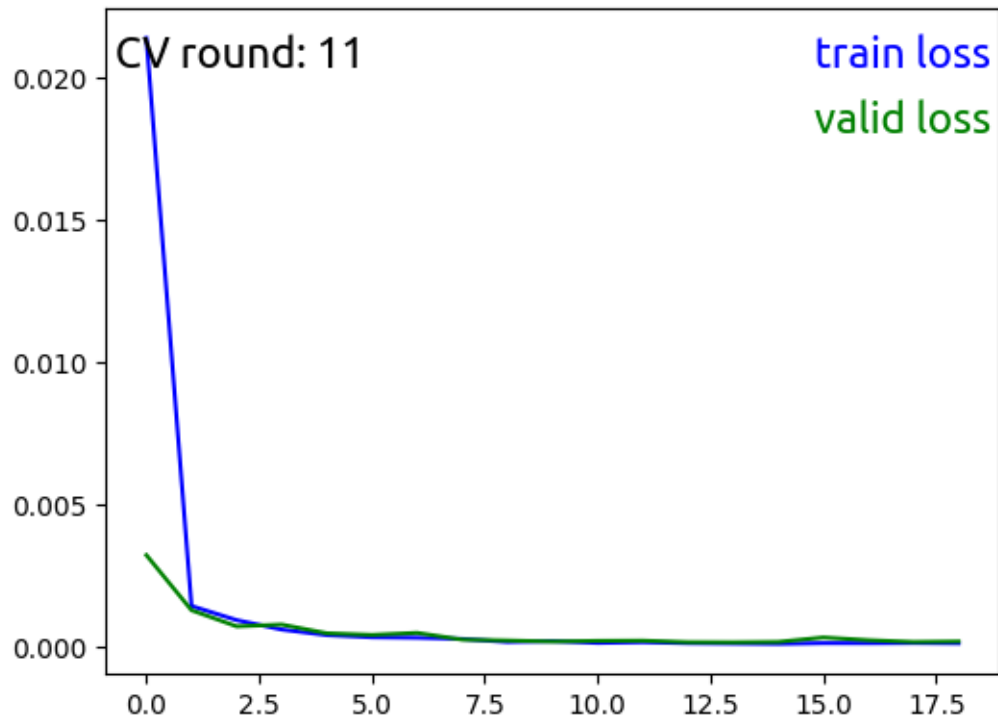
```
>round 10
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 16
min train loss: 0.00010566283284182039
min valid loss: 0.00011343417136231438
```



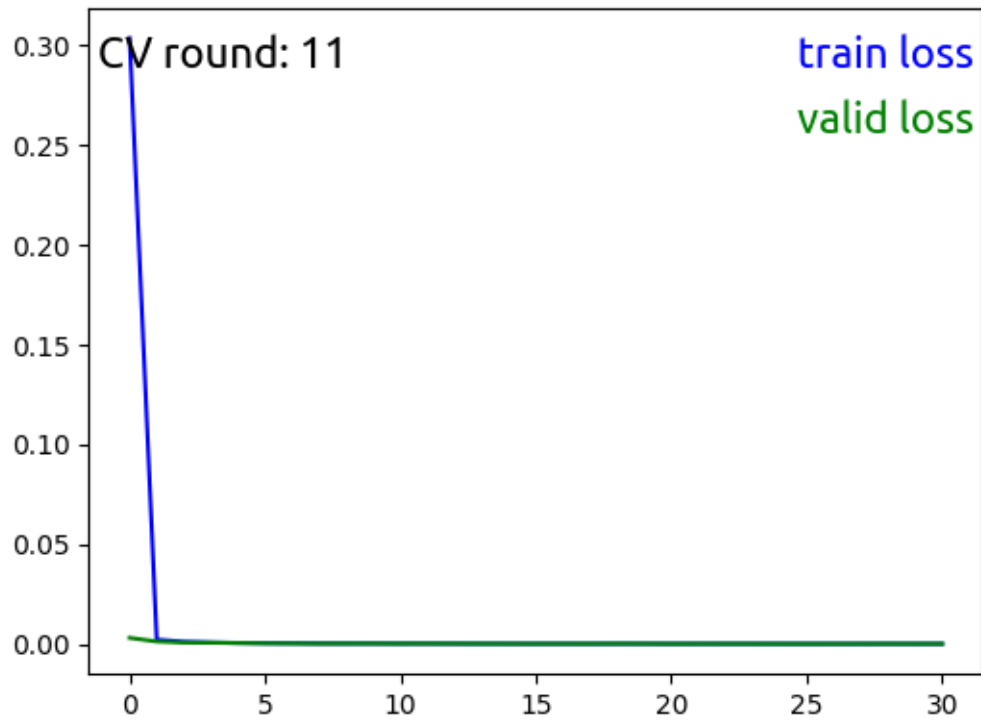
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 27
min train loss: 0.0002543745924909176
min valid loss: 0.00025411174283362925
```



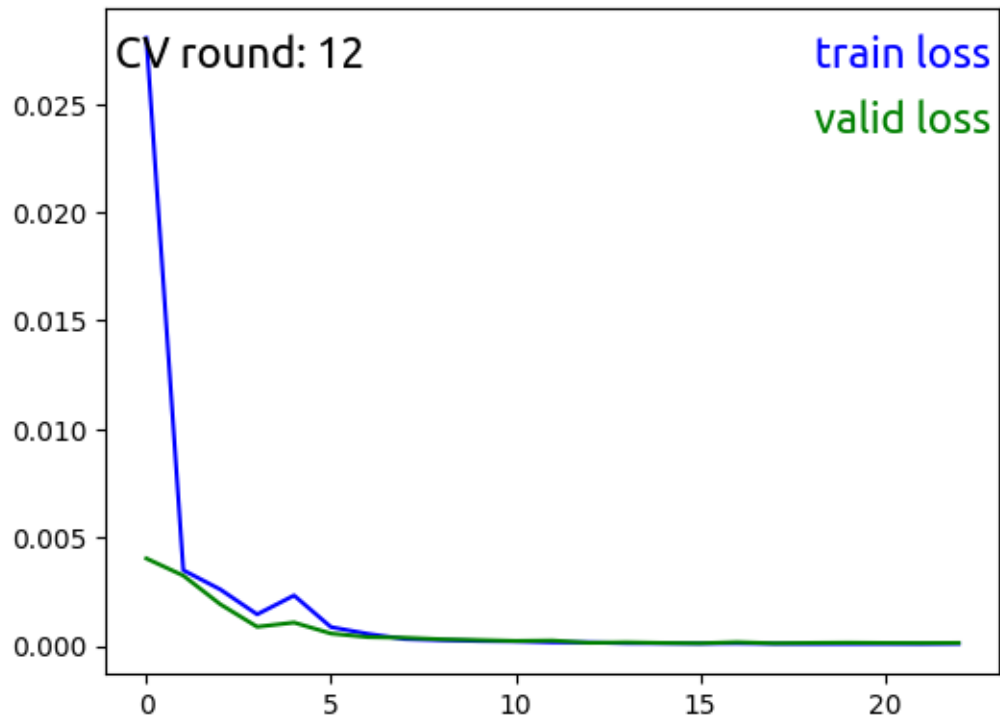
```
>round 11
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 18
min train loss: 9.526503019900922e-05
min valid loss: 0.00014866697601974012
```



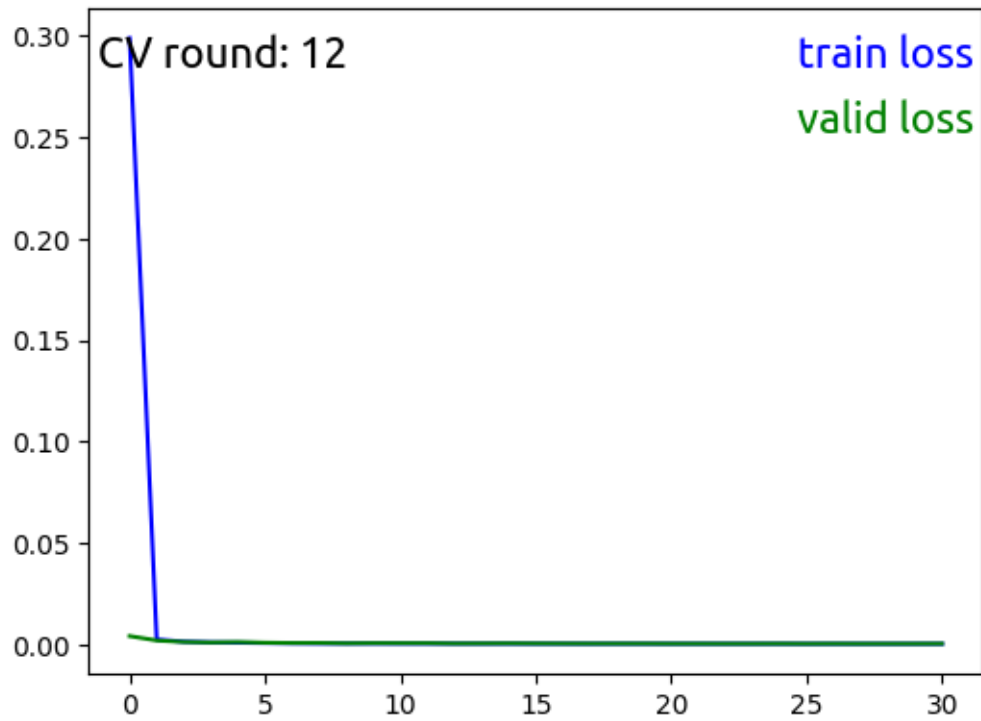
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 30
min train loss: 0.00022511767482766035
min valid loss: 0.00022700830049871001
```

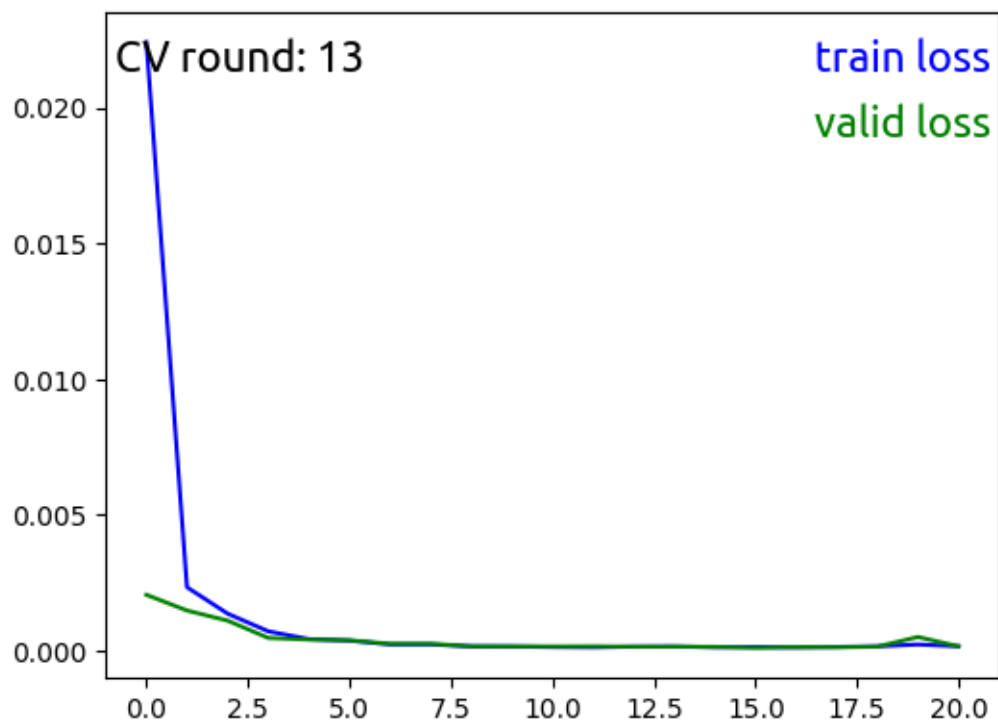
```
>round 12
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 22
min train loss: 8.651813192100786e-05
min valid loss: 0.00011172741797054187
```



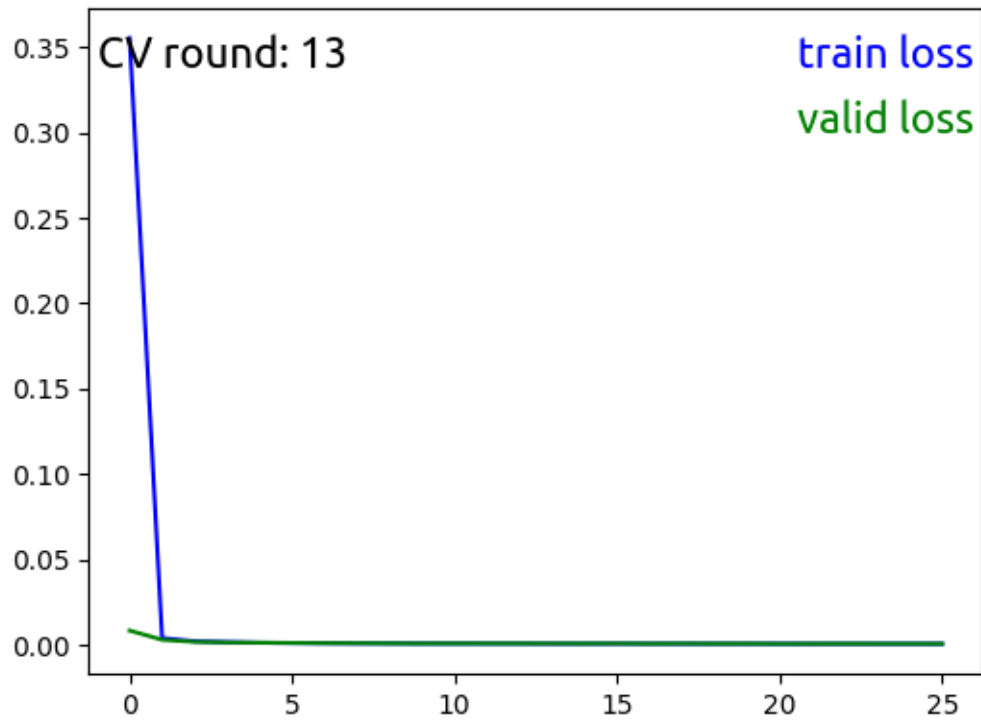
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 30
min train loss: 0.00023770880295408213
min valid loss: 0.00027289950048725586
```



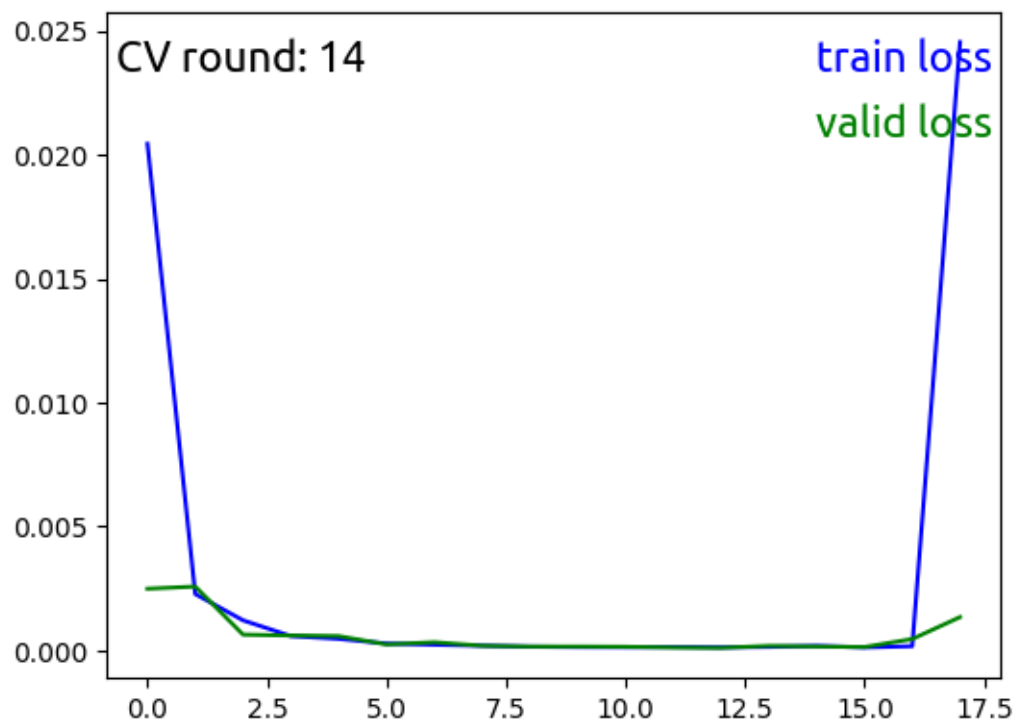
```
>round 13
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 20
min train loss: 0.00011722587866339988
min valid loss: 0.00010317567102902103
```



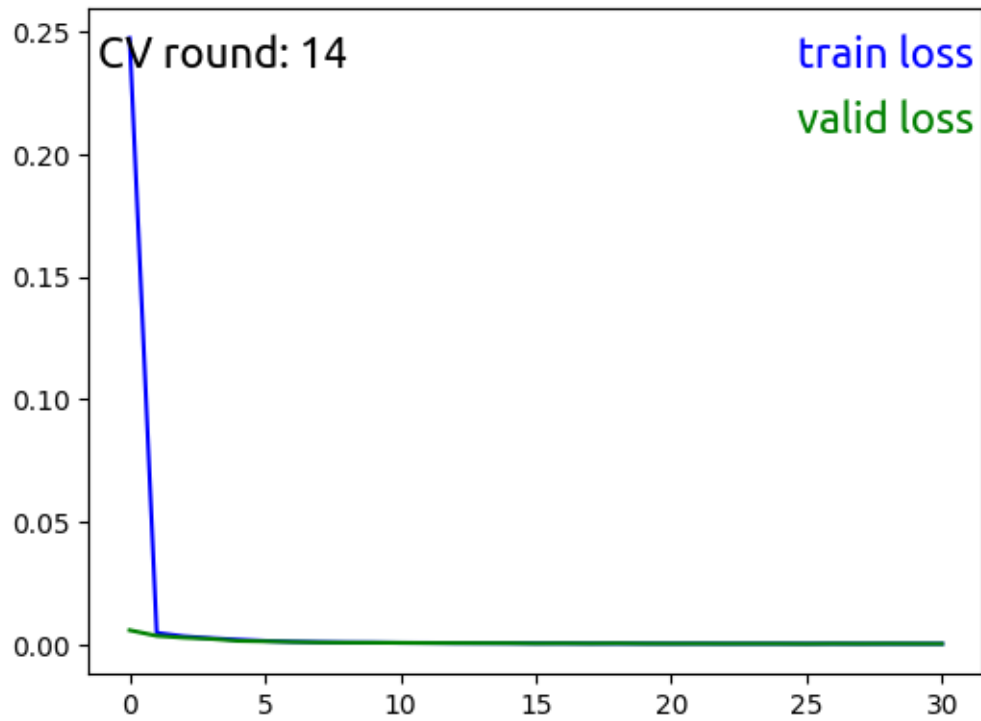
```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 25
min train loss: 0.0004160409454594959
min valid loss: 0.0004139911907259375
```



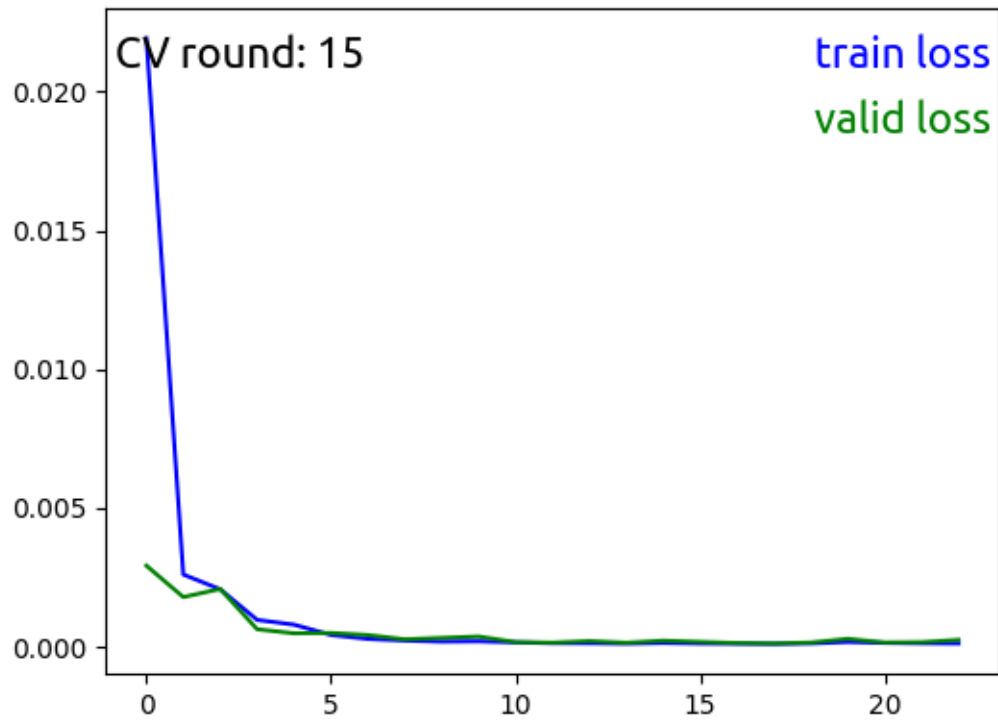
```
>round 14
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 17
min train loss: 0.00012476176484734893
min valid loss: 0.00010716400283854455
```



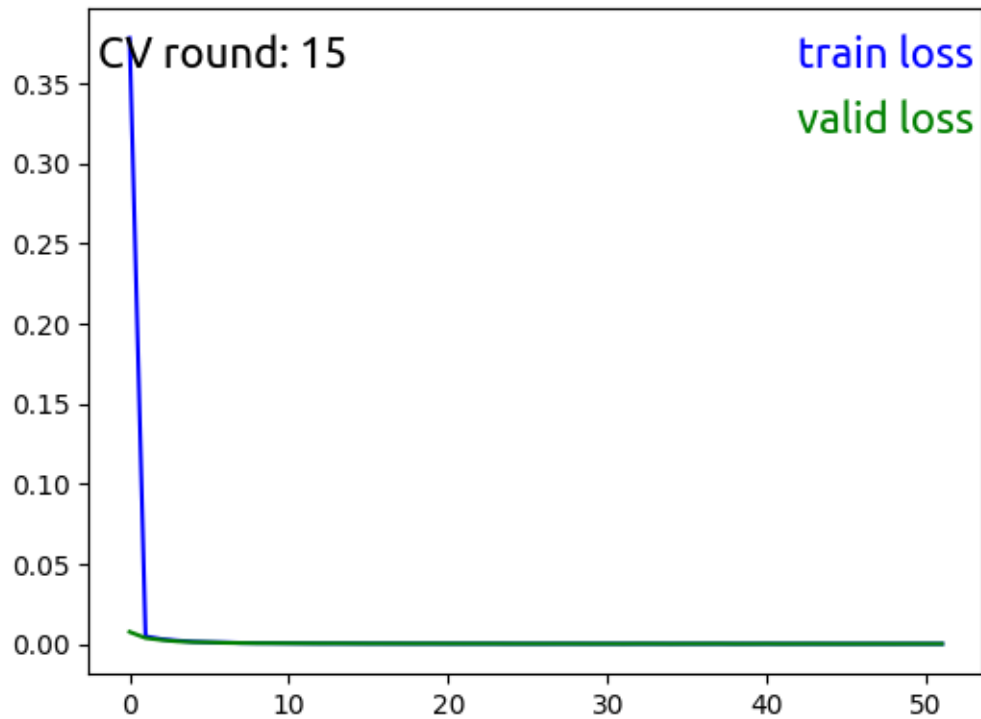
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 30
min train loss: 0.00040795174321498384
min valid loss: 0.00036602402178687043



```
>round 15
task 0: temperature_230509_discrete
EARLY STOPPING @ epoch 22
min train loss: 9.284766537982223e-05
min valid loss: 0.00010247603349853308
```



```
task 1: pressure_230516_discrete
EARLY STOPPING @ epoch 51
min train loss: 0.00030708407364066014
min valid loss: 0.00028315038071013987
```

BEST model: CV=0.pth with 0.00017814694001572207

trained datas sequentially

Aggregate performance:

temperature_230509_discrete: Valid loss mean 0.00011688593306189433, std 2.6295029491838643e-05

pressure_230516_discrete: Valid loss mean 0.0002668082830723506, std 7.384083669643842e-05

TRAINING COMPLETE_____

TEST_____

Testing temperature_230509_discrete, loss: 0.3176850119703694

Testing pressure_230516_discrete, loss: 0.00021307463157427264