In [1]:

```
# from google.colab import drive
# drive.mount('/content/gdrive')
```

In [2]:

```
# !pip install mne
```

In [3]:

```
# !pip install --pre torch torchvision -f https://download.pytorch.org/whl/nightly
```

In [4]:

```
##############################################################################
#  ________   _________   ___    ___ ___  ___  ________  ________  ___  __      __
# |\   ___  \|\___   ___\|\  \  /  /|\  \|\  \|\   __  \|\   ____\|\  \|\  \   /
# \ \  \\ \  \|___ \  \_|\ \  \/  / | \  \\\  \ \  \|\  \ \  \___|\ \  \/  /|_/__/
#  \ \  \\ \  \   \ \  \  \ \  \   / / \ \   __  \ \   __  \ \  \    \ \   ___  \__|
#   \ \  \\ \  \   \ \  \  \ /     \/   \ \  \ \  \ \  \ \  \ \  \____\ \  \\ \  \
#    \ \__\\ \__\   \ \__\/  /\   \    \ \__\ \__\ \__\ \__\ \_______\ \__\\ \__\|
#     \|__| \|__|    \|__/__/ /\ __\    \|__|\|__|\|__|\|__|\|_______|\|__| \|__|
#                         |__|/ \|__|
##############################################################################

import mne
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
from torch import nn
import random
from torch.utils.data import Dataset, DataLoader
from mne.decoding import Vectorizer
torch.manual_seed(42)
```

```
<torch._C.Generator at 0x7f96c7a8d510>
```

In [5]:

```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print(DEVICE)
```

```
cuda
```

In [6]:

```
# Path to training data
train_path = "/home/deepak/learning_project/student/BrainAge/training/"
```

In [7]:

```
# Path to testing data (public test set)
test_path = "/home/deepak/learning_project/student/BrainAge/testing_flat/"
```

In [8]:

```
condition_ec = "EC"  # closed eyes condition
condition_eo = "EO"  # closed eyes condition
train_subj = 1100  # use 1100 of the 1200 training subjects for training
val_subj = 100   # use 100 of the 1200 training subjects for validation
test_subj = 400  # use 10 instead of 400 testing subjects, for demonstration purpo
```

In [9]:

```
class CustomDataset(Dataset):
    def __init__(self, path, subj, ages, start):
        self.path = path
        self.subj = subj
        self.ages = ages
        self.start = start
    def __len__(self):
        return self.subj
    def __getitem__(self, idx):
        s = idx + self.start
        fname = f"subj{s:04}_{condition_eo}_raw.fif.gz"
        raw = mne.io.read_raw(self.path + fname, preload=True, verbose='warning')
        d = raw.get_data()
        ft = d.shape[-1]
        data_eo = torch.zeros(1, 129, 10000)
        data_eo[:, :, :ft] = -200 * torch.tensor(d)
        fname = f"subj{s:04}_{condition_ec}_raw.fif.gz"
        raw = mne.io.read_raw(self.path + fname, preload=True, verbose='warning')
        d = raw.get_data()
        ft = d.shape[-1]
        data_ec = torch.zeros(1, 129, 20000)
        data_ec[:, :, :ft] = -200 * torch.tensor(d)
        data = (data_eo, data_ec)
        age = self.ages[idx]
        return data, age
```

In [10]:

```
# get the age to predict from the CSV file
meta = pd.read_csv(train_path + "train_subjects.csv")
y_train = []
for age in meta.age[:1200]:
    y_train.append(age)
print(np.min(y_train), np.mean(y_train), np.median(y_train), np.max(y_train))
```

```
5.005932 10.356727775833333 9.587952999999999 21.899041
```

In [11]:

```
train_data = CustomDataset(train_path, train_subj, y_train[:train_subj], 1)
```

In [12]:

```
val_data = CustomDataset(train_path, val_subj, y_train[train_subj:train_subj+val_s
```

In [13]:

```
# DataLoader
batch_size = 8

train_loader = DataLoader(
    train_data,
    batch_size=batch_size,
    num_workers=1
)

val_loader = DataLoader(
    val_data,
    batch_size=batch_size,
    num_workers=1
)
```

In [14]:

```
e = torch.ones(4,5)
e[2,2] = 9.3
np.around(e).int()
```

```
tensor([[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 9, 1, 1],
        [1, 1, 1, 1, 1]], dtype=torch.int32)
```

In [15]:

```
class AgeNET(nn.Module):
    def conv_block(self, in_channels, out_channels, kernel, stride, pool):
        return nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_s
            nn.LeakyReLU(),
            nn.AvgPool2d(kernel_size=(1, pool), stride=(1, pool))
        ).to(DEVICE)
    def __init__(self, sampling_rate, num_T, num_C):
        super().__init__()
        self.device = torch.device(DEVICE)
        print(DEVICE)
        self.to(DEVICE)
        self.pool = 3
        self.state = 0
        self.Time1_eo = self.conv_block(1, num_T, (1, sampling_rate//2), 1, self.p
        self.Time2_eo = self.conv_block(1, num_T, (1, sampling_rate//4), 1, self.p
        self.Time3_eo = self.conv_block(1, num_T, (1, sampling_rate//8), 1, self.p
        self.Time1_ec = self.conv_block(1, num_T, (1, sampling_rate//2), 1, self.p
        self.Time2_ec = self.conv_block(1, num_T, (1, sampling_rate//4), 1, self.p
        self.Time3_ec = self.conv_block(1, num_T, (1, sampling_rate//8), 1, self.p
        self.BN_T_eo = nn.BatchNorm2d(num_T).to(DEVICE)
        self.BN_T_ec = nn.BatchNorm2d(num_T).to(DEVICE)
        self.Chan1_eo = self.conv_block(num_T, num_C, (129, 1), 1, self.pool)
        self.Chan2_eo = self.conv_block(num_T, num_C, (65, 1), (64, 1), self.pool)
        # self.Chan3_eo = self.conv_block(num_T, num_C, (32, 1), (32, 1), self.poo
        self.Chan1_ec = self.conv_block(num_T, num_C, (129, 1), 1, self.pool)
        self.Chan2_ec = self.conv_block(num_T, num_C, (65, 1), (64, 1), self.pool)
        # self.Chan3_ec = self.conv_block(num_T, num_C, (32, 1), (32, 1), self.poo
        self.BN_C_eo = nn.BatchNorm2d(num_C).to(DEVICE)
        self.BN_C_ec = nn.BatchNorm2d(num_C).to(DEVICE)
        size_eo, size_ec = self.get_size()
        print(size_eo, size_ec)
        self.fc_eo = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=size_eo, out_features=1024, bias=True),
            nn.BatchNorm1d(1024),
            nn.ReLU()
        )
        self.fc_ec = nn.Sequential(
            nn.Flatten(),
```

```
            nn.Linear(in_features=size_ec, out_features=2048, bias=True),
            nn.BatchNorm1d(2048),
            nn.ReLU()
        )
        self.classifier = nn.Sequential(
            nn.Linear(in_features=3072, out_features=1024, bias=True),
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Dropout(p=0.01, inplace=False),
            # nn.Linear(in_features=4096, out_features=2048, bias=True),
            # nn.BatchNorm1d(2048),
            # nn.ReLU(),
            # nn.Dropout(p=0.003, inplace=False),
            # nn.Linear(in_features=2048, out_features=1024, bias=True),
            # nn.BatchNorm1d(1024),
            # nn.ReLU(),
            # nn.Dropout(p=0.002, inplace=False),
            nn.Linear(in_features=1024, out_features=512, bias=True),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(p=0.001, inplace=False),
            nn.Linear(in_features=512, out_features=256, bias=True),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Linear(in_features=256, out_features=64, bias=True),
            nn.BatchNorm1d(64),
            nn.ReLU(),
        )
#         self.last_layer = []
#         for j in out_len:
#             self.last_layer.append(
#                 nn.Sequential(
#                     nn.Linear(in_features=64, out_features=j, bias=True, device=
#                     nn.ReLU()
#                 )
#             )
        self.l_l0 = nn.Sequential(
                    nn.Linear(in_features=64, out_features=25, bias=True),
                    nn.ReLU()
                )
        self.l_l1 = nn.Sequential(
```

```
                    nn.Linear(in_features=64, out_features=1, bias=True),
                    nn.ReLU()
                )
    def forward(self, x_eo, x_ec):
        y = self.Time1_eo(x_eo)
        out_eo = y
        y = self.Time2_eo(x_eo)
        out_eo = torch.cat((out_eo, y), dim=-1)
        y = self.Time3_eo(x_eo)
        out_eo = torch.cat((out_eo, y), dim=-1)
        y = self.Time1_ec(x_ec)
        out_ec = y
        y = self.Time2_ec(x_ec)
        out_ec = torch.cat((out_ec, y), dim=-1)
        y = self.Time3_ec(x_ec)
        out_ec = torch.cat((out_ec, y), dim=-1)
        out_eo = self.BN_T_eo(out_eo)
        out_ec = self.BN_T_ec(out_ec)
        z = self.Chan1_eo(out_eo)
        out_f_eo = z
        z = self.Chan2_eo(out_eo)
        out_f_eo = torch.cat((out_f_eo, z), dim=2)
        # z = self.Chan3_eo(out_eo)
        # out_f_eo = torch.cat((out_f_eo, z), dim=2)
        z = self.Chan1_ec(out_ec)
        out_f_ec = z
        z = self.Chan2_ec(out_ec)
        out_f_ec = torch.cat((out_f_ec, z), dim=2)
        # z = self.Chan3_ec(out_ec)
        # out_f_ec = torch.cat((out_f_ec, z), dim=2)
        out_f_eo = self.BN_C_eo(out_f_eo)
        out_f_ec = self.BN_C_ec(out_f_ec)
        out = torch.cat((self.fc_eo(out_f_eo), self.fc_ec(out_f_ec)), dim=-1)
        out = self.classifier(out)
        if self.state == 0:
            out = self.l_l0(out)
        else:
            out = self.l_l1(out)
        return out
    def set_state(self, st):
        self.state = st
```

```
    def get_size(self):
        d_eo = torch.ones(1, 1, 129, 10000).to(DEVICE)
        d_ec = torch.ones(1, 1, 129, 20000).to(DEVICE)
        y = self.Time1_eo(d_eo)
        out_eo = y
        y = self.Time2_eo(d_eo)
        out_eo = torch.cat((out_eo, y), dim=-1)
        y = self.Time3_eo(d_eo)
        out_eo = torch.cat((out_eo, y), dim=-1)
        y = self.Time1_ec(d_ec)
        out_ec = y
        y = self.Time2_ec(d_ec)
        out_ec = torch.cat((out_ec, y), dim=-1)
        y = self.Time3_ec(d_ec)
        out_ec = torch.cat((out_ec, y), dim=-1)
        out_eo = self.BN_T_eo(out_eo)
        out_ec = self.BN_T_ec(out_ec)
        z = self.Chan1_eo(out_eo)
        out_f_eo = z
        z = self.Chan2_eo(out_eo)
        out_f_eo = torch.cat((out_f_eo, z), dim=2)
        # z = self.Chan3_eo(out_eo)
        # out_f_eo = torch.cat((out_f_eo, z), dim=2)
        z = self.Chan1_ec(out_ec)
        out_f_ec = z
        z = self.Chan2_ec(out_ec)
        out_f_ec = torch.cat((out_f_ec, z), dim=2)
        # z = self.Chan3_ec(out_ec)
        # out_f_ec = torch.cat((out_f_ec, z), dim=2)
        return torch.numel(out_f_eo), torch.numel(out_f_ec)
```

In [16]:

```
model_A = AgeNET(128, 9, 6).to(DEVICE)
```

```
cuda
14940 29934
```

In [17]:

```
1 DEVICE
```

```
'cuda'
```

In [18]:

```
1 model_A = model_A.to(DEVICE)
```

In [19]:

```
from torchsummary import summary

summary(model_A, [(1, 129, 10000), (1, 129, 20000)])
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 9, 129, 9937]             585
         LeakyReLU-2         [-1, 9, 129, 9937]               0
         AvgPool2d-3          [-1, 9, 129, 828]               0
            Conv2d-4         [-1, 9, 129, 9969]             297
         LeakyReLU-5         [-1, 9, 129, 9969]               0
         AvgPool2d-6          [-1, 9, 129, 830]               0
            Conv2d-7         [-1, 9, 129, 9985]             153
         LeakyReLU-8         [-1, 9, 129, 9985]               0
         AvgPool2d-9          [-1, 9, 129, 832]               0
           Conv2d-10        [-1, 9, 129, 19937]             585
        LeakyReLU-11        [-1, 9, 129, 19937]               0
        AvgPool2d-12         [-1, 9, 129, 1661]               0
           Conv2d-13        [-1, 9, 129, 19969]             297
        LeakyReLU-14        [-1, 9, 129, 19969]               0
        AvgPool2d-15         [-1, 9, 129, 1664]               0
           Conv2d-16        [-1, 9, 129, 19985]             153
        LeakyReLU-17        [-1, 9, 129, 19985]               0
        AvgPool2d-18         [-1, 9, 129, 1665]               0
      BatchNorm2d-19         [-1, 9, 129, 2490]              18
      BatchNorm2d-20         [-1, 9, 129, 4990]              18
           Conv2d-21           [-1, 6, 1, 2490]           6,972
        LeakyReLU-22           [-1, 6, 1, 2490]               0
        AvgPool2d-23            [-1, 6, 1, 830]               0
           Conv2d-24           [-1, 6, 2, 2490]           3,516
        LeakyReLU-25           [-1, 6, 2, 2490]               0
        AvgPool2d-26            [-1, 6, 2, 830]               0
           Conv2d-27           [-1, 6, 1, 4990]           6,972
        LeakyReLU-28           [-1, 6, 1, 4990]               0
        AvgPool2d-29           [-1, 6, 1, 1663]               0
           Conv2d-30           [-1, 6, 2, 4990]           3,516
        LeakyReLU-31           [-1, 6, 2, 4990]               0
        AvgPool2d-32           [-1, 6, 2, 1663]               0
      BatchNorm2d-33            [-1, 6, 3, 830]              12
      BatchNorm2d-34           [-1, 6, 3, 1663]              12
          Flatten-35                [-1, 14940]               0
           Linear-36                 [-1, 1024]      15,299,584
      BatchNorm1d-37                 [-1, 1024]           2,048
             ReLU-38                 [-1, 1024]               0
          Flatten-39                [-1, 29934]               0
           Linear-40                 [-1, 2048]      61,306,880
      BatchNorm1d-41                 [-1, 2048]           4,096
             ReLU-42                 [-1, 2048]               0
           Linear-43                 [-1, 1024]       3,146,752
      BatchNorm1d-44                 [-1, 1024]           2,048
             ReLU-45                 [-1, 1024]               0
          Dropout-46                 [-1, 1024]               0
           Linear-47                  [-1, 512]         524,800
      BatchNorm1d-48                  [-1, 512]           1,024
             ReLU-49                  [-1, 512]               0
          Dropout-50                  [-1, 512]               0
```

```
            Linear-51                  [-1, 256]         131,328
       BatchNorm1d-52                  [-1, 256]             512
              ReLU-53                  [-1, 256]               0
            Linear-54                   [-1, 64]          16,448
       BatchNorm1d-55                   [-1, 64]             128
              ReLU-56                   [-1, 64]               0
            Linear-57                   [-1, 25]           1,625
              ReLU-58                   [-1, 25]               0
================================================================
Total params: 80,460,379
Trainable params: 80,460,379
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 12696075.44
Forward/backward pass size (MB): 1726.25
Params size (MB): 306.93
Estimated Total Size (MB): 12698108.62
----------------------------------------------------------------
```

In [20]:

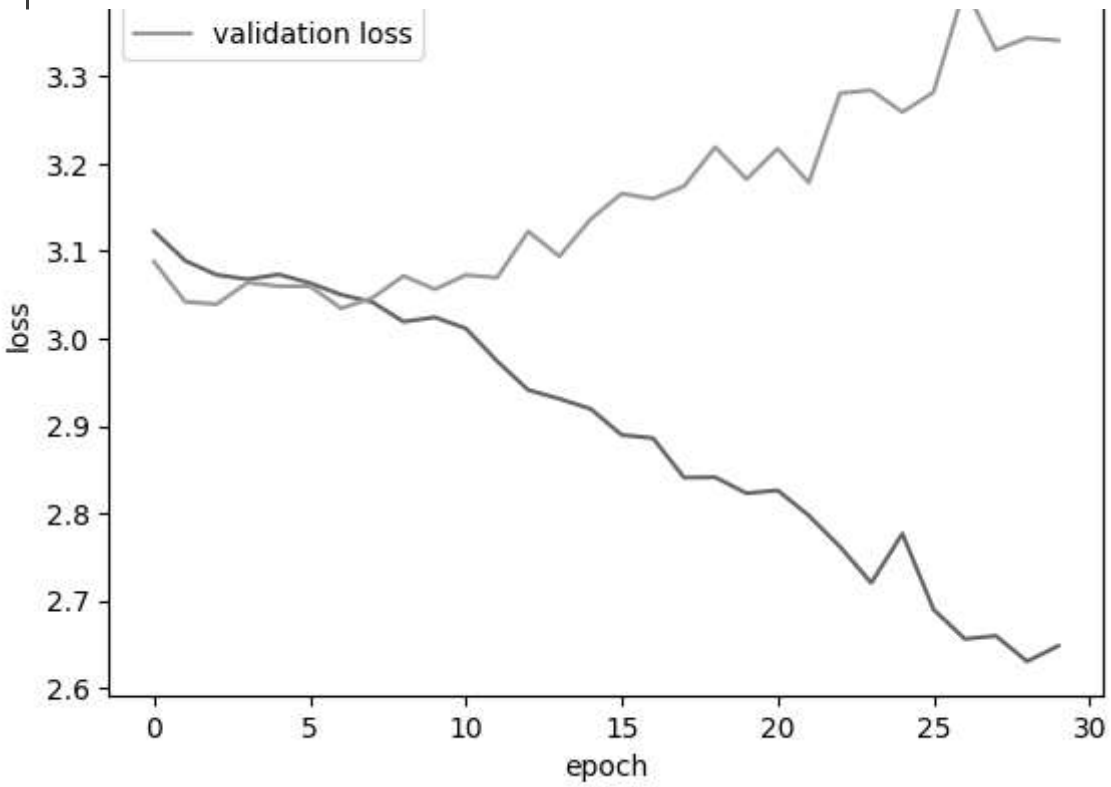
```
def train(model, device, train_loader, val_loader, lr, epochs):
    train_abs_log = []
    train_log = []
    val_log = []
    val_abs_log = []
    model.train()
    abs_loss = nn.L1Loss(reduction='mean')
    lossfunc = nn.CrossEntropyLoss(reduction='mean')
    optimizer = torch.optim.NAdam(model.parameters(), lr=lr)
    f = plt.figure()
    for epoch in range(epochs):
        train_abs_loss = 0.0
        train_loss = 0.0
        print(f"Epoch #{1 + epoch:02}: ")
        i = 0
        for data, age in train_loader:
            data_eo, data_ec = data
            data_eo = data_eo.to(device)           # shape = (batch_size, 1, 129,
            data_ec = data_ec.to(device)           # shape = (batch_size, 1, 129,
            batch_size = age.size(0)
            optimizer.zero_grad()
            output = model(data_eo, data_ec)       # shape = (batch_size, 25)
            loss = lossfunc(output, np.around(age).to(torch.int64).to(device))
            loss.backward()
            optimizer.step()
            print(i)
            i += batch_size
            train_abs_loss += abs_loss(torch.argmax(output, dim=-1), age.to(DEVICE
            train_loss += loss.item() * batch_size
        train_abs_loss /= train_subj
        train_loss /= train_subj
        train_abs_log.append(train_abs_loss)
        train_log.append(train_loss)
        val_abs_loss = 0.0
        val_loss=0.0
        for data, age in val_loader:
            data_eo, data_ec = data
            data_eo = data_eo.to(device)           # shape = (batch_size, 129, 100
            data_ec = data_ec.to(device)           # shape = (batch_size, 129, 200
            batch_size = age.size(0)
```

```
            output = model(data_eo, data_ec)       # shape = (batch_size, 25)
            loss = lossfunc(output, np.around(age).to(torch.int64).to(device))

            val_abs_loss += abs_loss(torch.argmax(output, dim=-1), age.to(DEVICE))
            val_loss += loss.item() * batch_size
        val_abs_loss /= val_subj
        val_loss /= val_subj
        val_abs_log.append(val_abs_loss)
        val_log.append(val_loss)
        print(f"CrossEntropyLoss:     train = {train_loss}, validation = {val_loss
        print(f"mean absolute error:  train = {train_abs_loss}, validation = {val_
        plt.clf()
        plt.plot(train_log, label='train loss')
        plt.plot(val_log, label='validation loss')
        plt.xlabel('epoch')
        plt.ylabel('loss')
        plt.title('Loss')
        plt.legend()
        plt.show()
    return train_abs_log, val_abs_log
```

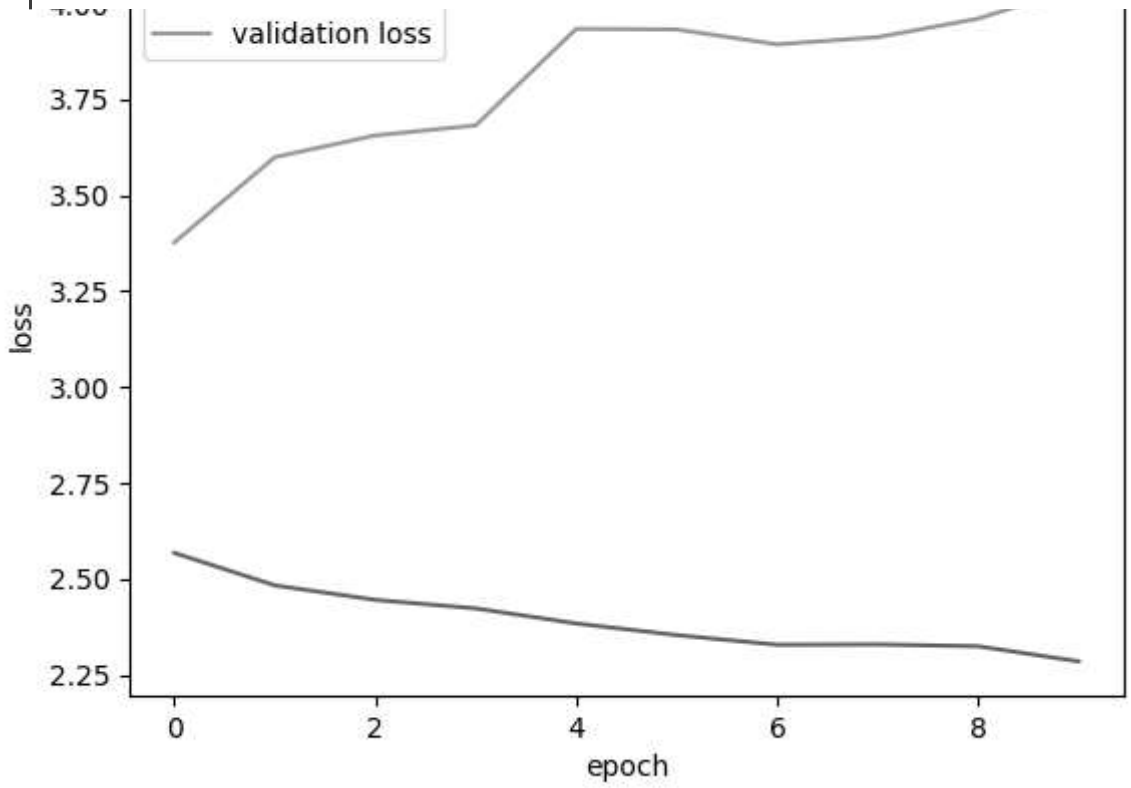
In [21]:

```
train_abs_log, val_abs_log = train(model_A, DEVICE, train_loader, val_loader, 0.00
```



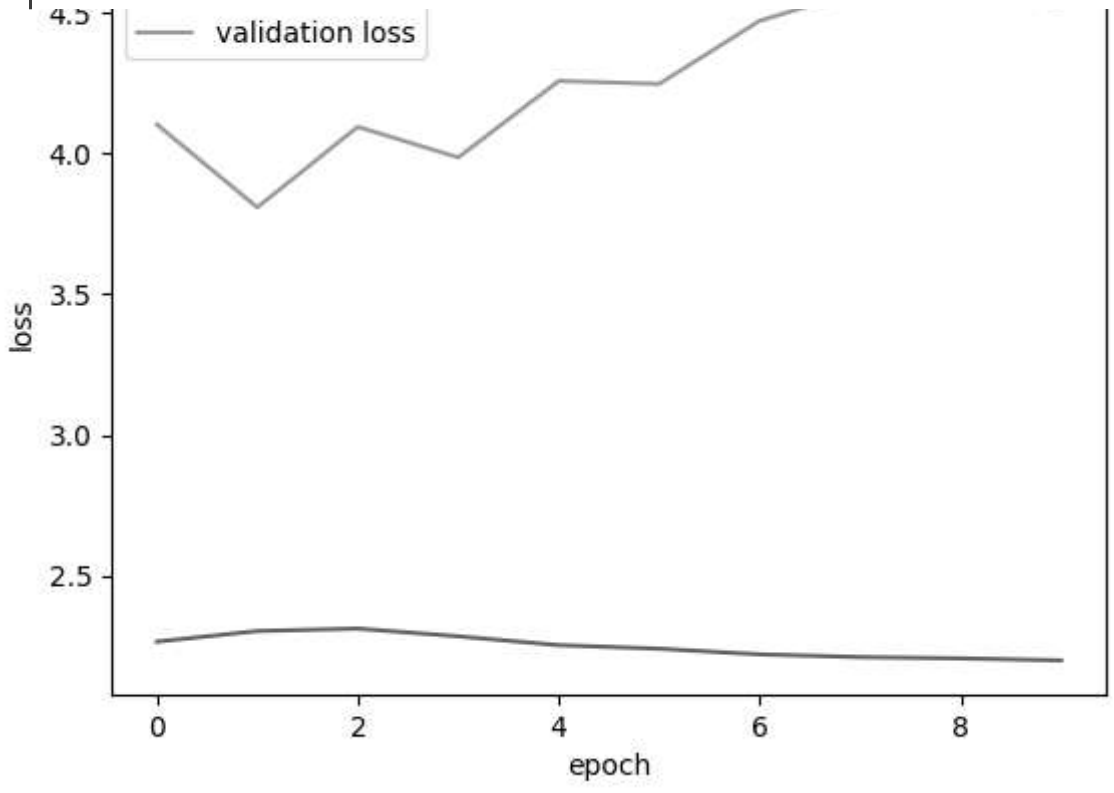
In [22]:

```
train_abs_log_1, val_abs_log_1 = train(model_A, DEVICE, train_loader, val_loader,
```



In [23]:

```
train_abs_log_2, val_abs_log_2 = train(model_A, DEVICE, train_loader, val_loader,
```



In [ ]:

```
def train_L(model, loss_func, device, train_loader, val_loader, lr, epochs):
    train_abs_log = []
    val_abs_log = []
    model.train()
    abs_loss = loss_func
    optimizer = torch.optim.NAdam(model.parameters(), lr=lr)
    f = plt.figure()
    for epoch in range(epochs):
        train_abs_loss = 0.0
        print(f"Epoch #{1 + epoch:02}: ")
        i = 0
        for data, age in train_loader:
            data_eo, data_ec = data
            data_eo = data_eo.to(device)           # shape = (batch_size, 1, 129,
            data_ec = data_ec.to(device)           # shape = (batch_size, 1, 129,
            batch_size = age.size(0)
            optimizer.zero_grad()
            output = model(data_eo, data_ec)       # shape = (batch_size, 25)
            loss = abs_loss(torch.squeeze(output), age.float().to(device))
            loss.backward()
            optimizer.step()
            print(i)
            i += batch_size
            train_abs_loss += loss.item() * batch_size
        train_abs_loss /= train_subj
        train_abs_log.append(train_abs_loss)
        val_abs_loss = 0.0
        for data, age in val_loader:
            data_eo, data_ec = data
            data_eo = data_eo.to(device)           # shape = (batch_size, 129, 100
            data_ec = data_ec.to(device)           # shape = (batch_size, 129, 200
            batch_size = age.size(0)
            output = model(data_eo, data_ec)       # shape = (batch_size, 25)
            print(output)
            print(age)
            loss = abs_loss(torch.squeeze(output), age.float().to(device))

            val_abs_loss += loss.item() * batch_size
        val_abs_loss /= val_subj
        val_abs_log.append(val_abs_loss)
```

```
        print(f"mean absolute error:  train = {train_abs_loss}, validation = {val_
        plt.clf()
        plt.plot(train_abs_log, label='train loss')
        plt.plot(val_abs_log, label='validation loss')
        plt.xlabel('epoch')
        plt.ylabel('loss')
        plt.title('Loss')
        plt.legend()
        plt.show()
    return train_abs_log, val_abs_log
```

In [29]:

```
model_A.set_state(1)
```

In [37]:

```
train_L(model_A, nn.L1Loss(reduction='mean'), DEVICE, train_loader, val_loader, 0.
```

```
([2.314322156906128,
  2.0543491112102163,
  1.905966567993164,
  1.8039553785324096,
  1.6943501112677835,
  1.6155094714598222,
  1.5890673858469182,
  1.474887121373957,
  1.4691239057887684,
  1.4140368791060014],
 [2.878715238571167,
  2.9833662128448486,
  2.8866349983215334,
  2.887751216888428,
  2.9465968132019045,
  2.956333570480347,
  2.7572300910949705,
  3.0312032747268676,
  2.9802655029296874,
  3.0485591506958007])
```

In [ ]:

```
train_L(model_A, nn.L1Loss(reduction='mean'), DEVICE, val_loader, val_loader, 0.00
```

In [ ]:

```

```