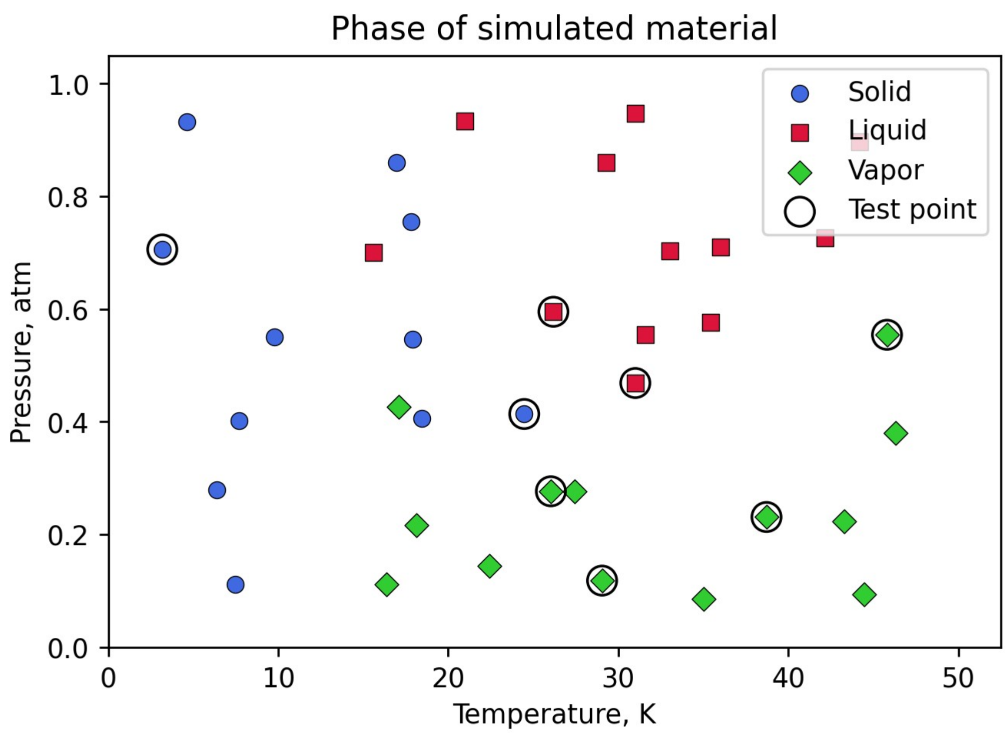
M8-L2 Problem 2

Let's revisit the material phase prediction problem once again. You will use this problem to try multi-class classification in PyTorch. You will have to write code for a classification network and for training.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import torch
from torch import nn, optim
import torch.nn.functional as F

def plot_loss(train_loss, val_loss):
    plt.figure(figsize=(4,2),dpi=250)
    plt.plot(train_loss,label="Training")
    plt.plot(val_loss,label="Validation",linewidth=1)
    plt.legend()
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

def split_data(X, Y):
    np.random.seed(100)
    N = len(Y)
    train_mask = np.zeros(N, dtype=np.bool_)
    train_mask[np.random.permutation(N)[:int(N*0.8)]] = True
    train_x, val_x = torch.Tensor(X[train_mask,:]), 
torch.Tensor(X[np.logical_not(train_mask),:])
    train_y, val_y = torch.Tensor(Y[train_mask]), 
torch.Tensor(Y[np.logical_not(train_mask)])
    return train_x, val_x, train_y, val_y

x1 = 
np.array([7.4881350392732475,16.351893663724194,22.427633760716436,29.
04883182996897,35.03654799338904,44.45894113066656,6.375872112626925,1
8.117730007820796,26.036627605010292,27.434415188257777,38.71725038082
664,43.28894919752904,7.680445610939323,18.45596638292661,17.110360581
978867,24.47129299701541,31.002183974403255,46.32619845547938,9.781567
509498505,17.90012148246819,26.186183422327638,31.59158564216724,35.41
479362252932,45.805291762864556,3.182744258689332,15.599210213275237,1
7.833532874090462,33.04668917049584,36.018483217500716,42.146619399905
234,4.64555612104627,16.942336894342166,20.961503322165484,29.28433948
8686488,30.98789800436355,44.17635497075877,])
x2 = 
np.array([0.11120957227224215,0.1116933996874757,0.14437480785146242,0
.11818202991034835,0.0859507900573786,0.09370319537993416,0.2797631195
927265,0.216022547162927,0.27667667154456677,0.27706378696181594,0.231
0382561073841,0.22289262976548535,0.40154283509241845,0.40637107709426
```

```
23,0.427019677041788,0.41386015134623205,0.46883738380592266,0.3802044
8107480287,0.5508876756094834,0.5461309517884996,0.5953108325465398,0.
5553291602539782,0.5766310772856306,0.5544425592001603,0.7058969583645
52,0.7010375141164304,0.7556329589465274,0.7038182951348614,0.70965823
61680054,0.7268725170660963,0.9320993229847936,0.8597101275793062,0.93
37944907498804,0.8596098407893963,0.9476459465013396,0.896865120164770
2,])
X = np.vstack([x1,x2]).T
y =
np.array([0,2,2,2,2,2,0,2,2,2,2,2,0,0,2,0,1,2,0,0,1,1,1,2,0,1,0,1,1,1,
0,0,1,1,1,1,])

X = torch.Tensor(X)
Y = torch.tensor(y,dtype=torch.long)

train_x, val_x, train_y, val_y = split_data(X,Y)


def plot_data(newfig=True):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s",
color="crimson"), dict(marker="D", color="limegreen")]
    labels = ["Solid", "Liquid", "Vapor"]

    if newfig:
        plt.figure(figsize=(6,4),dpi=250)

    x = X.detach().numpy()
    y = Y.detach().numpy().flatten()

    for i in range(1+max(y)):
        plt.scatter(x[y==i,0], x[y==i,1], s=40, **(markers[i]),
edgecolor="black", linewidths=0.4,label=labels[i])

    plt.scatter(val_x[:,0],
val_x[:,1],s=120,c="None",marker="o",edgecolors="black",label="Test
point")

    plt.title("Phase of simulated material")
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_model(model, res=200):
    xlim = [0,52.5]
    ylim = [0,1.05]
```

```
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    XY = torch.Tensor(XY)
    color = model.predict(XY).reshape(res,res).detach().numpy()
    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1,
cmap=cmap,vmin=0,vmax=2)
    return

plot_data()
plt.show()
```



Model definition

In the cell below, complete the definition for `PhaseNet`, a classification neural network.

- The network should take in 2 inputs and return 3 outputs.
- The network size and hidden layer activations are up to you.
- Make sure to use the proper activation function (for multi-class classification) at the final layer.
- The `predict()` method has been provided, to return the integer class value. You must finish `__init__()` and `forward()`.

```python
class PhaseNet(nn.Module):
    def __init__(self):
        super().__init__()
        # YOUR CODE GOES HERE
        # 4 layers, 2 input, 3 output
        # 3 hidden layers with 10 neurons each
        self.lin1 = nn.Linear(2,10)
        self.lin2 = nn.Linear(10,10)
        self.lin3 = nn.Linear(10,10)
        self.lin4 = nn.Linear(10,3)
        self.act = nn.ReLU()

    def predict(self,X):
        Y = self(X)
        return torch.argmax(Y,dim=1)

    def forward(self, X):
        # YOUR CODE GOES HERE
        x = self.lin1(X)
        x = self.act(x)  # Activation of first hidden layer
        x = self.lin2(x)
        x = self.act(x)  # Activation at second hidden layer
        x = self.lin3(x)
        x = self.act(x) # Activation at third hidden layer
        x = self.lin4(x)
        x = F.softmax(x, dim=1)  # Activation at fourth hidden layer
(softmax for classification problem)
        return x
```

Training

Most of the training code has been provided below. Please add the following where indicated:

- Define a loss function (for multiclass classification)
- Define an optimizer and call it `opt`. You may choose which optimizer.

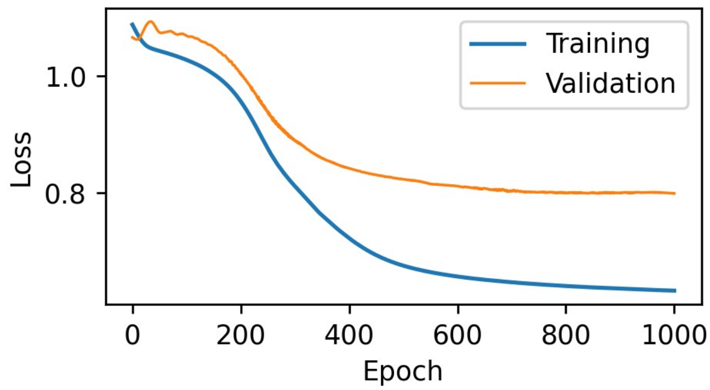
Make sure the training curves you get are reasonable.

```python
model = PhaseNet()

lr = 0.001
epochs = 1000

# Define loss function
# YOUR CODE GOES HERE
lossfun = nn.CrossEntropyLoss() # Cross-entropy loss for mutliclass
classification

# Define an optimizer, `opt`
```

```python
# YOUR CODE GOES HERE
opt = optim.Adam(model.parameters(), lr=lr) # Adam optimizer


train_hist = []
val_hist = []

for epoch in range(epochs+1):
    model.train()
    loss_train = lossfun(model(train_x), train_y)
    train_hist.append(loss_train.item())

    model.eval()
    loss_val = lossfun(model(val_x), val_y)
    val_hist.append(loss_val.item())

    opt.zero_grad()
    loss_train.backward()
    opt.step()
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch:>4} of {epochs}:   Train Loss = 
{loss_train.item():.4f}   Validation Loss = {loss_val.item():.4f}")


plot_loss(train_hist, val_hist)
```

```
Epoch    0 of 1000:   Train Loss = 1.0877   Validation Loss = 1.0659
Epoch   40 of 1000:   Train Loss = 1.0455   Validation Loss = 1.0879
Epoch   80 of 1000:   Train Loss = 1.0342   Validation Loss = 1.0716
Epoch  120 of 1000:   Train Loss = 1.0193   Validation Loss = 1.0638
Epoch  160 of 1000:   Train Loss = 0.9964   Validation Loss = 1.0414
Epoch  200 of 1000:   Train Loss = 0.9564   Validation Loss = 1.0027
Epoch  240 of 1000:   Train Loss = 0.8927   Validation Loss = 0.9515
Epoch  280 of 1000:   Train Loss = 0.8333   Validation Loss = 0.9036
Epoch  320 of 1000:   Train Loss = 0.7909   Validation Loss = 0.8768
Epoch  360 of 1000:   Train Loss = 0.7531   Validation Loss = 0.8552
Epoch  400 of 1000:   Train Loss = 0.7224   Validation Loss = 0.8420
Epoch  440 of 1000:   Train Loss = 0.6982   Validation Loss = 0.8326
Epoch  480 of 1000:   Train Loss = 0.6815   Validation Loss = 0.8260
Epoch  520 of 1000:   Train Loss = 0.6704   Validation Loss = 0.8208
Epoch  560 of 1000:   Train Loss = 0.6626   Validation Loss = 0.8143
Epoch  600 of 1000:   Train Loss = 0.6568   Validation Loss = 0.8114
Epoch  640 of 1000:   Train Loss = 0.6523   Validation Loss = 0.8073
Epoch  680 of 1000:   Train Loss = 0.6486   Validation Loss = 0.8040
Epoch  720 of 1000:   Train Loss = 0.6455   Validation Loss = 0.8023
Epoch  760 of 1000:   Train Loss = 0.6429   Validation Loss = 0.8005
Epoch  800 of 1000:   Train Loss = 0.6407   Validation Loss = 0.7998
Epoch  840 of 1000:   Train Loss = 0.6386   Validation Loss = 0.7991
Epoch  880 of 1000:   Train Loss = 0.6369   Validation Loss = 0.8007
Epoch  920 of 1000:   Train Loss = 0.6353   Validation Loss = 0.7993
```

```
Epoch  960 of 1000:   Train Loss = 0.6339   Validation Loss = 0.8004
Epoch 1000 of 1000:   Train Loss = 0.6326   Validation Loss = 0.7991
```



Plot results

Plot your network predictions with the data by running the following cell. If your network has significant overfitting/underfitting, go back and retrain a new network with different layer sizes/activations.

```
plot_data(newfig=True)
plot_model(model)
plt.show()
```