In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
DATA_PATH = '../data/Environment_Temperature_change_E_All_Data_NOFLAG.csv'
DATA_ROMANIA_PATH = '../data/Environment_Temperature_change_ROMANIA.csv'
data_frame = pd.read_csv(DATA_PATH, encoding='cp1252')
```

```
(9656, 66)
```

In [3]:
```
country = 'Romania'
df = data_frame.loc[((data_frame.Area == country) & (data_frame.Element == 'Te
                     ( data_frame['Months Code'] >= 7001) & (data_frame['Month
df.to_csv(DATA_ROMANIA_PATH, index=False )

df
```

Out[3]:

	Area Code	Area	Months Code	Months	Element Code	Element	Unit	Y1961	Y1962	Y1963	...
6154	183	Romania	7001	January	7271	Temperature change	°C	-0.164	1.810	-6.143	...
6156	183	Romania	7002	February	7271	Temperature change	°C	1.070	-1.159	-1.901	...
6158	183	Romania	7003	March	7271	Temperature change	°C	2.669	-1.974	-2.377	...
6160	183	Romania	7004	April	7271	Temperature change	°C	2.689	0.956	-0.157	...
6162	183	Romania	7005	May	7271	Temperature change	°C	-1.394	0.627	1.116	...
6164	183	Romania	7006	June	7271	Temperature change	°C	0.756	-0.994	0.616	...
6166	183	Romania	7007	July	7271	Temperature change	°C	-0.544	-0.665	1.533	...
6168	183	Romania	7008	August	7271	Temperature change	°C	-0.058	1.716	1.941	...
6170	183	Romania	7009	September	7271	Temperature change	°C	0.497	-0.093	1.667	...
6172	183	Romania	7010	October	7271	Temperature change	°C	0.875	0.510	0.067	...
6174	183	Romania	7011	November	7271	Temperature change	°C	1.710	2.447	3.306	...
6176	183	Romania	7012	December	7271	Temperature change	°C	-1.133	-3.256	-3.103	...

12 rows × 66 columns

In [4]:

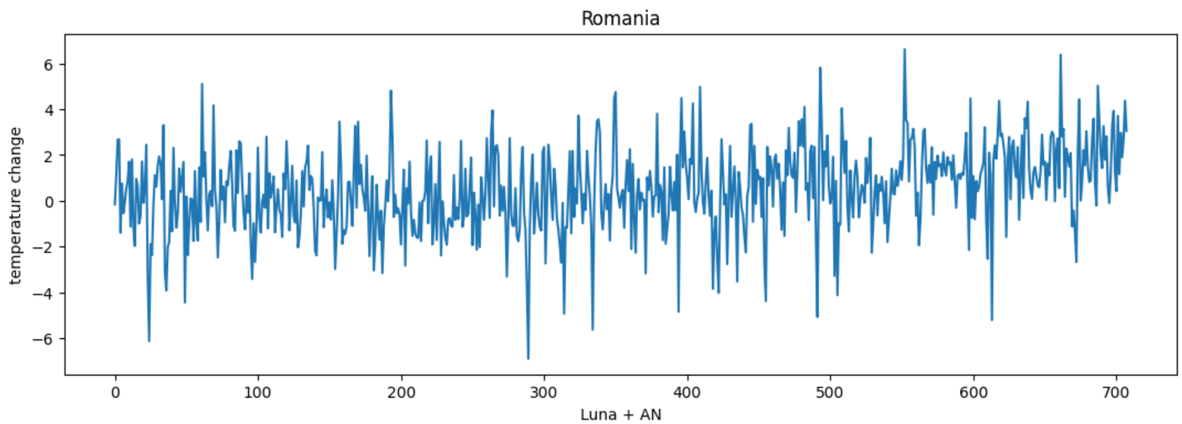
```
x, y = [], []
for an in range(1961, 2020):
    for luna in range(0, 12):
        x_point = (an - 1961 ) * 12 + luna
        y_point = df.iloc[luna][f'Y{an}']
        x.append(x_point)
```

In [5]:

```

plt.figure(figsize=[13,4])
plt.plot(x, y)
plt.xlabel('Luna + AN')
plt.ylabel('temperature change')
# plt.xticks(year_columns[::3])
```

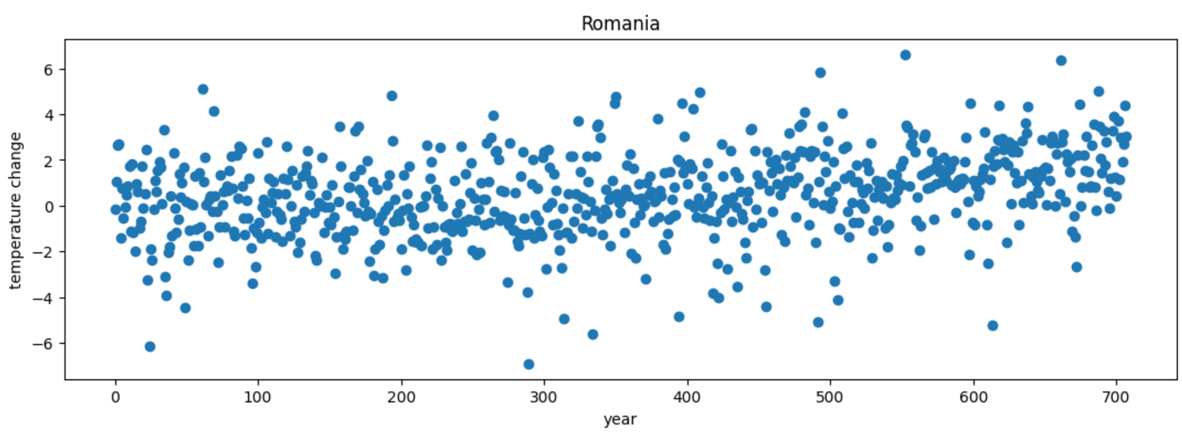
Out[5]: Text(0.5, 1.0, 'Romania')



In [6]:

```
plt.figure(figsize=[13,4])
plt.scatter(x, y)
plt.xlabel('year')
plt.ylabel('temperature change')
# plt.xticks(year_columns[::3])
```

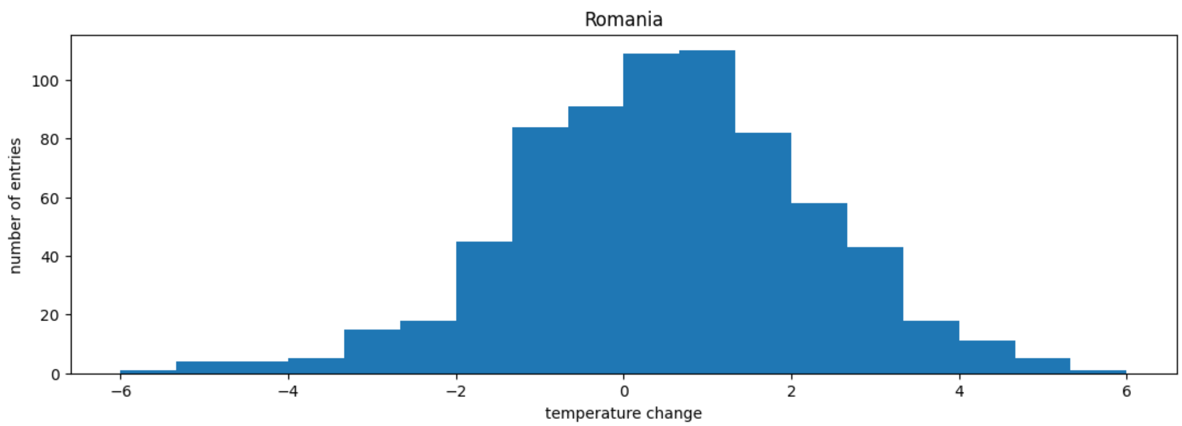
Out[6]: Text(0.5, 1.0, 'Romania')



In [7]:
```
plt.figure(figsize=[13,4])
bin_range = (-6,6)
bin_count = 18
hist, bin_edges = np.histogram(y, bins=bin_count, range=bin_range)

plt.hist(bin_edges[:-1], bin_edges, weights=hist)
plt.xlabel('temperature change')
plt.ylabel('number of entries')
```

Out[7]:
```
Text(0.5, 1.0, 'Romania')
```



In [8]:
```
# (0 -> 720)   -> y
```

In [81]:
```
inputs, targets = [], []
for an in range(1961, 2020):
    for luna in range(0, 12):
        x_point = luna
        y_point = an
        z_point = df.iloc[luna][f'Y{an}']
        inputs.append([x_point, y_point])
```

In [82]:
```
from sklearn.model_selection import train_test_split


inputs = torch.tensor(inputs, dtype=torch.float32)
targets = torch.tensor(targets, dtype=torch.float32)


#train_inputs, test_inputs, train_targets, test_targets = train_test_split(
    #inputs, targets, test_size=0.2, random_state=42)

train_inputs, test_inputs, train_targets, test_targets = train_test_split(
```

In [83]:
```
max_train_input = torch.max(train_inputs[:, 1])
min_train_input = torch.min(train_inputs[:, 1])
train_inputs[:, 1] = (train_inputs[:, 1] - min_train_input) / (max_train_input
test_inputs[:, 1] = (test_inputs[:, 1] - min_train_input) / (max_train_input -
```

Out[83]:
```
tensor([[6.0000, 0.7414],
        [0.0000, 0.0345],
        [0.0000, 0.6897],
        ...,
        [6.0000, 0.3793],
        [3.0000, 0.6207],
        [6.0000, 0.1379]])
```

We normalized the years by min-max normalization

In [84]:
```
print("Length of train set: " + str(len(train_targets)))

train_targets_negative = [el for el in train_targets if el < 0]

print("Length of train set with negative values as a result: " + str(len(train
```

```
Length of train set: 566
Length of train set with negative values as a result: 215
The train data is good proportioned!
```

In [85]:

Out[85]:
```
False
```

In [86]:
```
from models.ANN_v2 import TempChangeNN

# ax.contour3D(x,y, lambda x,y:  df.iloc[x][f'Y{y}'] , 50, cmap = 'binary')
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

We changed the ANN model such that now it predicts negative values. We added 2 layers and changed the activation function from LeakyReLu to ELU.

In [87]:

```python
from models.checkpoint import save_checkpoint
from torch.optim import Adam
import torch

LEARNING_RATE = 1e-3
num_epochs = 1000
batch_size = 64
loss_fn = torch.nn.MSELoss()
optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
# scaler = torch.cuda.amp.GradScaler()

total_loss = 0.0
for epoch in range(num_epochs):
    epoch_loss = 0.0
    for i in range(0, len(inputs), batch_size):
        # Get the input and target batches
        input_batch = inputs[i:i+batch_size]
        target_batch = targets[i:i+batch_size]
        # Forward pass
        output_batch = model(input_batch)

        loss = loss_fn(output_batch, target_batch)
        # Backward pass
        loss.backward()
        # Optimize
        optimizer.step()
        epoch_loss += loss.item()
        # Zero the gradients
        optimizer.zero_grad()

    # Print the loss for the current epoch
    avg_loss = epoch_loss / len(inputs)
    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}')
    total_loss += avg_loss
print(f'Epoch total loss], Loss: {total_loss/num_epochs:.4f}')

# save checkpoint
checkpoint =  {"state_dict": model.state_dict(),
             "optimizer": optimizer.state_dict()
            }
```

```
Epoch [1/1000], Loss: 15.9457
Epoch [2/1000], Loss: 4.2624
Epoch [3/1000], Loss: 0.6841
Epoch [4/1000], Loss: 0.1775
Epoch [5/1000], Loss: 0.2483
Epoch [6/1000], Loss: 0.1576
Epoch [7/1000], Loss: 0.0854
Epoch [8/1000], Loss: 0.0913
Epoch [9/1000], Loss: 0.1196
Epoch [10/1000], Loss: 0.1413
Epoch [11/1000], Loss: 0.1568
Epoch [12/1000], Loss: 0.1717
Epoch [13/1000], Loss: 0.1879
Epoch [14/1000], Loss: 0.2051
```

```
Epoch [15/1000], Loss: 0.2220
Epoch [16/1000], Loss: 0.2372
Epoch [17/1000], Loss: 0.2496
Epoch [18/1000], Loss: 0.2586
Epoch [19/1000], Loss: 0.2638
```

Changes: learning rate: from 10^-4^ to 10^-3^, batch size: from 16 to 64, epoch from 2000 to 1000. Results: a smaller loss, from 0.2273 to 0.0611

In [88]:

```
def predict(net, input_batch):
    net.eval()  # Set the model to evaluation mode

    with torch.no_grad():  # Disable gradient calculation for inference
        output_batch = net(input_batch)

    return output_batch.squeeze().tolist()

# Predict the temperature changes for the validation set
predicted_temp_changes = predict(model, test_inputs)

# Print the predicted temperature changes and the actual temperature changes
for i, (pred, actual) in enumerate(zip(predicted_temp_changes, test_targets.sq
    print(f"Validation Example {i+1}: Predicted: {pred:.2f}, Actual: {actual:.
```

```
Validation Example 1: Predicted: -0.18, Actual: 2.61
Validation Example 2: Predicted: -0.43, Actual: -0.58
Validation Example 3: Predicted: -0.17, Actual: 3.73
Validation Example 4: Predicted: -0.17, Actual: 0.55
Validation Example 5: Predicted: -0.24, Actual: 0.42
Validation Example 6: Predicted: -0.45, Actual: -1.18
Validation Example 7: Predicted: -0.53, Actual: 2.18
Validation Example 8: Predicted: -0.41, Actual: 0.05
Validation Example 9: Predicted: -0.43, Actual: -0.27
Validation Example 10: Predicted: -0.42, Actual: -0.61
Validation Example 11: Predicted: -0.18, Actual: 0.67
Validation Example 12: Predicted: -0.47, Actual: -1.63
Validation Example 13: Predicted: -0.41, Actual: 0.85
Validation Example 14: Predicted: -0.26, Actual: -0.07
Validation Example 15: Predicted: -0.25, Actual: 1.40
Validation Example 16: Predicted: -0.30, Actual: 0.44
Validation Example 17: Predicted: -0.29, Actual: -0.56
Validation Example 18: Predicted: -0.43, Actual: 0.99
Validation Example 19: Predicted: -0.18, Actual: -1.31
```

In [90]:

```
from sklearn.metrics import mean_squared_error
```

Out[90]: 3.603263024679887

We improved the error value from 5.8419 to 3.6032