In [1]:
```
from math import sqrt
import sys
sys.path.append('..') # This lets us access the code in ../src/
from src.nn import *

import xarray as xr
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cartopy.crs as ccrs

import torch
from torch import nn, optim, tensor
from torchsummary import summary
```

TODOs:

1. Handle missing data "properly" (we can interpolate)
2. Fix bug where grid adjustment of d18O failed at longitude 360
3. Better network structure that can actually learn / improve performance
4. Learning rate hyperparameter optimization? (Doesn't seem like an issue right now)

Dataset

Small dataset didn't have enough variability or lat/lon points to learn well, so I'm using **medium dataset**:

- All longitudes, latitudes between -65 and -80
- Skipped the pole because CNN assumes uniform grid-size across latitudes

In [2]:
```
## Load monthly mean isoGSM model
training_data_file = '../data/med_dataset_train.nc'
ds_train = xr.open_dataset(training_data_file)

valid_data_file = '../data/med_dataset_valid.nc'
ds_valid = xr.open_dataset(valid_data_file)

# Preprocess
X_train, Y_train, X_valid, Y_valid, Y_train_mean, Y_train_std = nn_preproces
```

Neural Network

Initial Neural Network design w/ 2 layers:

- Uses ReLU activation function for non-linearity (this is standard unless you have a specific reason to pick a different one)

- Time is not an input nor an output; we generalize time by randomly sampling from all time slices while training

Essentially, it models a grid point *p*'s geopotential height, precipitation, and temperature as a non-linear function of the delta 18-O values of the 3x13 grid of points centered at *p*

1. [3x13] Convolution layer:

- 3 (lat) x 13 (lon) kernel of weights to learn
- 1 dimensional input (d-18O)
- 3 dimensional output (gives model room to differentiate geop. height, precip, temp)

2. [3x13] Convolution layer:

- 3 (lat) x 13 (lon) kernel of weights to learn
- 3 dimensional input (from layer #1)
- 3 dimensional output (this will be evaluated against true Ys (['hgtprs', 'pratesfc', 'tmp2m']))

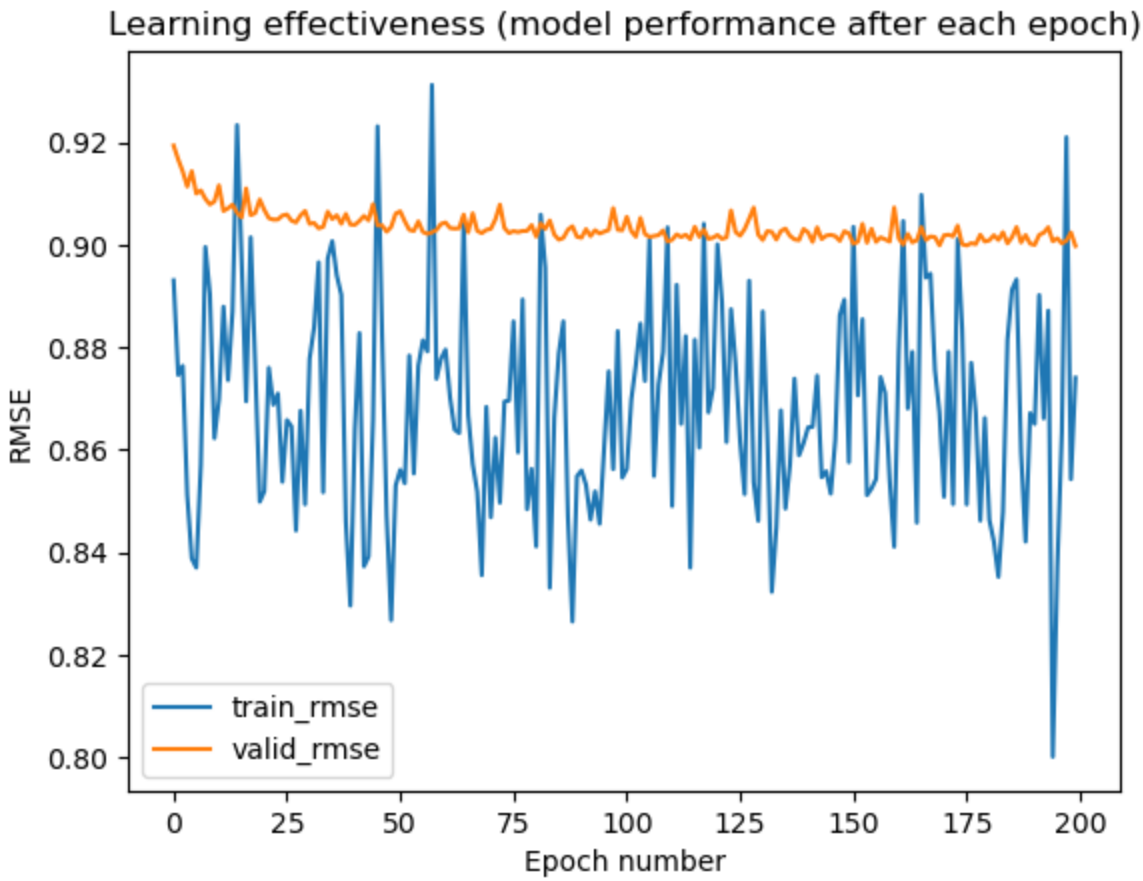
In [3]:
```
class simple_neural_network(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.main = nn.Sequential(
            nn.Conv2d(1, 3, (3, 13), padding=(1, 6)),
            nn.ReLU(),
            nn.Conv2d(3, 3, (3, 13), padding=(1, 6)),
            nn.ReLU()
        )

    def forward(self, x):
        out = self.main(x)
        return out
```

In [4]:
```
# Initialize the model

model = simple_neural_network()
model_summary = summary(model, X_train[0].shape)

# The device doesn't matter now but we'll want to use cuda GPUs in Sockeye
device = torch.device('cpu')
model.to(device)

pass
```

```
==========================================================================
===============
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
===============
├─Sequential: 1-1                        [-1, 3, 8, 192]           --
|    └─Conv2d: 2-1                       [-1, 3, 8, 192]           120
|    └─ReLU: 2-2                         [-1, 3, 8, 192]           --
|    └─Conv2d: 2-3                       [-1, 3, 8, 192]           354
|    └─ReLU: 2-4                         [-1, 3, 8, 192]           --
==========================================================================
===============
Total params: 474
Trainable params: 474
Non-trainable params: 0
Total mult-adds (M): 0.72
==========================================================================
===============
Input size (MB): 0.01
Forward/backward pass size (MB): 0.07
Params size (MB): 0.00
Estimated Total Size (MB): 0.08
==========================================================================
===============
```

In [ ]:
```python
# Train the model (re-running this cell trains the model more)

loss_fx = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=2e-3) # Learning rate hyperpar
results = random_batch_trainer(model, loss_fx, optimizer, device, X_train, Y
```

In [6]:
```python
# Visualize training progress

results_df = pd.DataFrame(results)
results_df[['train_rmse', 'valid_rmse']].plot()
plt.title("Learning effectiveness (model performance after each epoch)")
plt.xlabel("Epoch number")
plt.ylabel("RMSE")
plt.show()
```



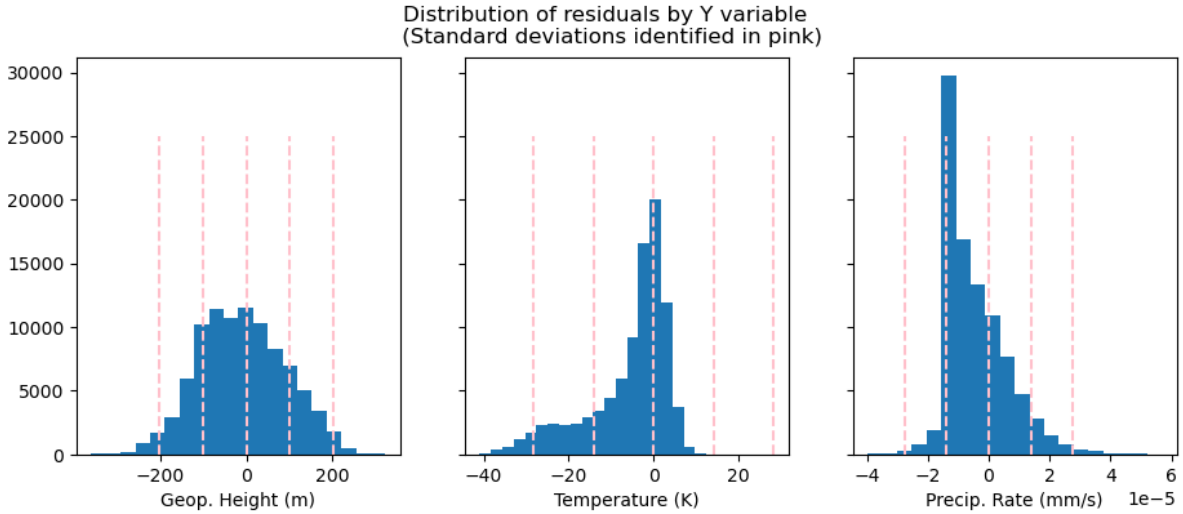
Model Training Comments

1. Epochs run quickly!
   - Each Epoch contains 60 (random) years of training data == 92,000 input points
2. Two-layer network architecture hits a limit how well it can learn (~0.9 validation RMSE on standardized Ys)
3. Training for more epochs does not improve performance
4. **We need more neurons & a deeper network to get better performance**
   - We have computing power to do it

Variance in training RMSE results from random training (this allows us to generalize time). Some Epochs we might over-sample winter months, for example

In [7]:
```
# Calculate residuals and append them to validation xArray dataset

resid = calculate_model_residuals(model, X_valid, Y_valid, Y_train_mean, Y_t
resid = np.swapaxes(resid, 0, 1) # Switch axes back to vars/time/lat/lon to

resid_ds = ds_valid.copy()
resid_ds = resid_ds.assign(
    hgtprs_resid = (['time', 'latitude', 'longitude'], resid[0]),
    pratesfc_resid = (['time', 'latitude', 'longitude'], resid[1]),
    tmp2m_resid = (['time', 'latitude', 'longitude'], resid[2])
)
#resid_ds
```

In [8]:
```
# Plot distribution of residuals
fig, axs = plt.subplots(1, 3, figsize = (11,4), sharey = True)
resid_ds['hgtprs_resid'].plot.hist(ax = axs[0], bins=20)
resid_ds['tmp2m_resid'].plot.hist(ax = axs[1], bins=20)
resid_ds['pratesfc_resid'].plot.hist(ax = axs[2], bins=20)

axs[0].set_xlabel("Geop. Height (m)")
axs[0].vlines([-202.8, -101.4, 0, 101.4, 202.8], 0, 25000, colors = 'pink',
axs[1].set_xlabel("Temperature (K)")
axs[1].vlines([-28.24, -14.12, 0, 14.12, 28.24], 0, 25000, colors = 'pink',
axs[2].set_xlabel("Precip. Rate (mm/s)")
axs[2].vlines([-2.773e-5, -1.387e-5, 0, 1.387e-5, 2.773e-5], 0, 25000, color

plt.suptitle("Distribution of residuals by Y variable \n (Standard deviatior
plt.show()
```



Analysis of Residuals

- **Geopotential height** looks mostly symmetrical and centered but heavier-tailed (we're unbiased but inaccurate)
- **Temperature** looks significantly left-skewed but most of the predictions are close (we sometimes significantly over-estimate temperature)
- **Precipitation rate** is skewed; not many residuals are 0 (we seem to either slighty over-estimate or significantly under-estimate precipitation)

For reference, the validation set standard deviations of each variable are: | Variable | STD | |--------------|-----------| | Geop. Height | 101.4 | | Temperature | 14.12 | | Precip. Rate | 1.387 e-5 |

Model Performance across Space

For all 60 times slices in the validation data:

1. Left map shows **residuals RMSE** at each grid point (this measures accuracy overall)

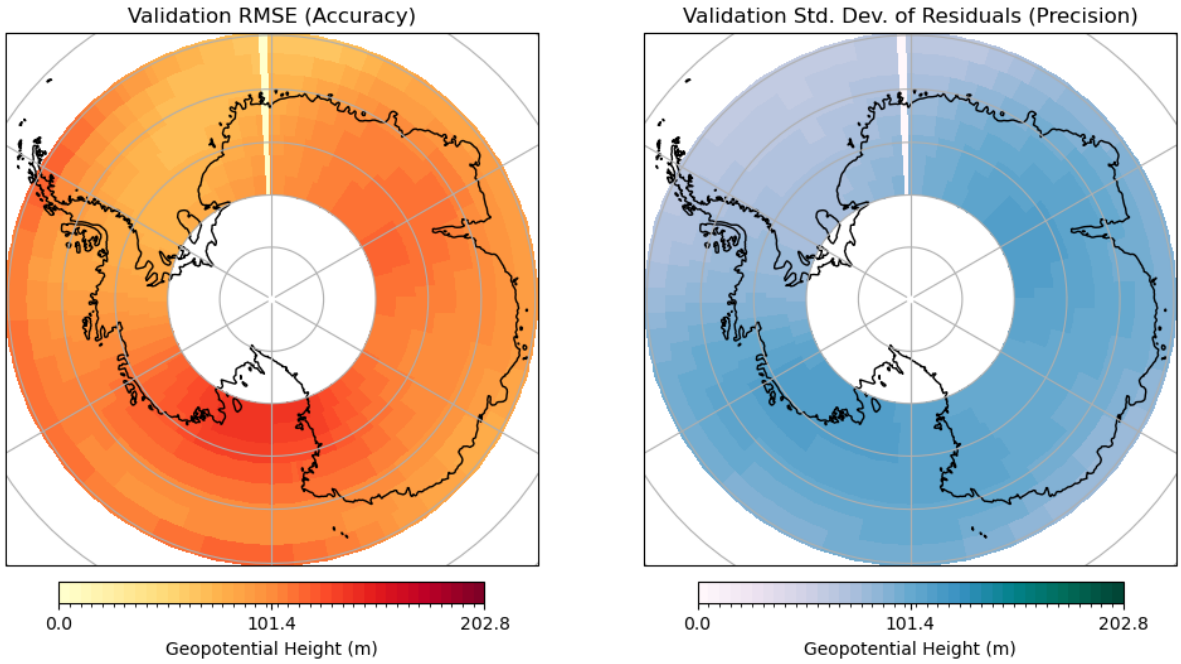
2. Right map shows **standard deviation of residuals** (this measures precision - whether our errors are consistently in the same direction and magnitude or not)
    - This helps let us know that we're generalizing for time well

- Colors are scaled to standard deviation of the real Ys in the validation set (dark red/green == 2 stds)
- The model was trained on standardized data, but has been re-scaled so we can compare in original units

Geopotential Height

- Not a lot of spatial pattern or variation to the errors; seems we're just generally 1 SD wrong everywhere
- Slice at longitude 360 is obvious -- shows our grid adjustment of geopotential height failed for this slice -> new bug to fix!

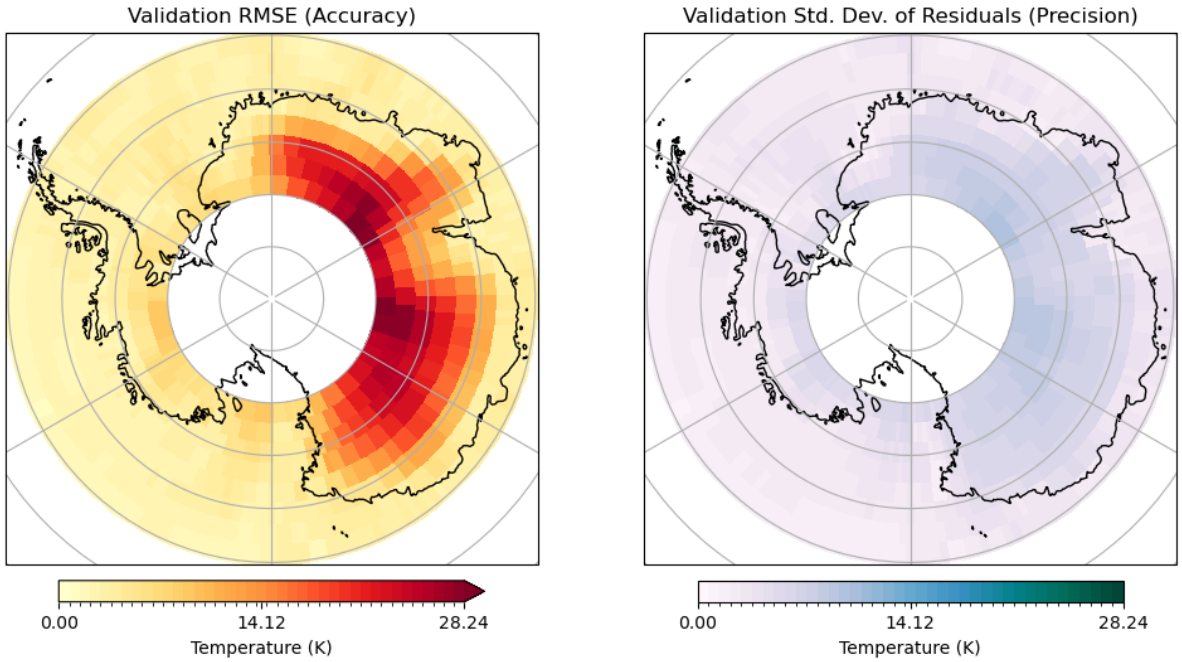
In [9]:
```
# Model performance predicting geopotential height
generate_residual_plots(resid_ds, 'hgtprs_resid', 'Geopotential Height (m)')
```



Temperature

- Model struggles to predict temperature over land
- We can see exactly where those large over-estimates we saw in the histograms are located
- On the bright side, for most of the map we're off by less than 5 degrees
- Notice we're consistently inaccurate in the interior (the blue plot doesn't show the same intensity where the error is) ->
    - this is a systematic error in architecture not a lack of time/seasonality adjustment error

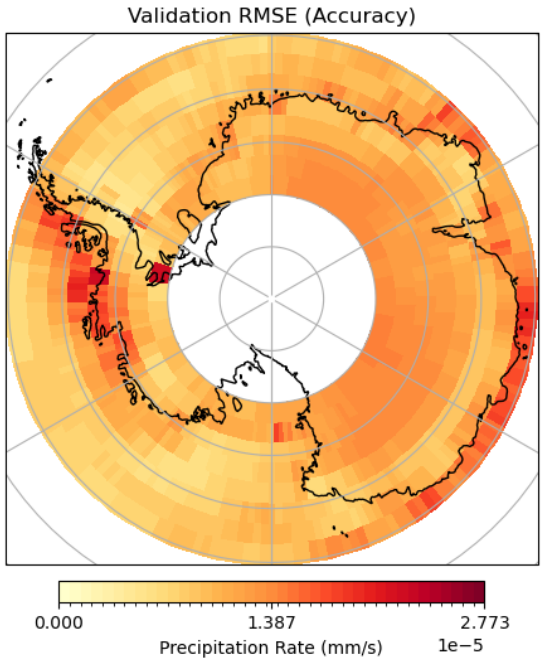
In [10]:
```
# Model performance predicting temperature
generate_residual_plots(resid_ds, 'tmp2m_resid', 'Temperature (K)')
```

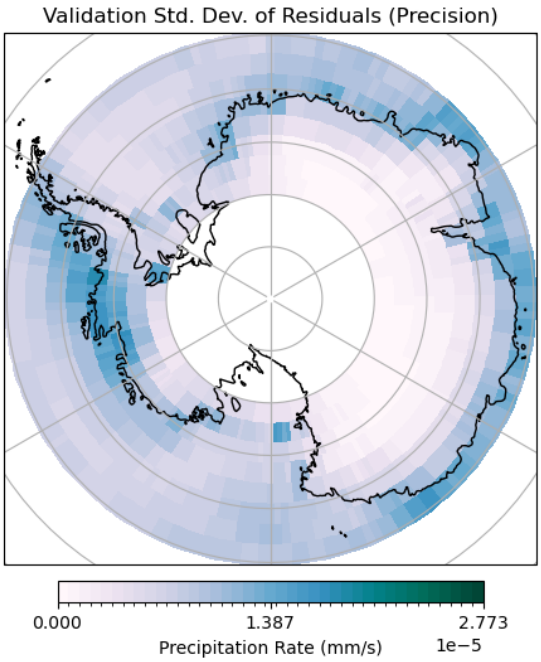


Precipitation Rate

- Model is the most wrong along the coast, both in accuracy and precision
- Interior is orange (~1SD) but almost no standard deviation (I wonder if it's always predicting 0 there or something)
- Long story short: there are spatial patterns to the residuals we need to give the model the tools to detect and correct

In [11]:
```
# Model performance prediction precipitation rate
generate_residual_plots(resid_ds, 'pratesfc_resid', 'Precipitation Rate (mm/
```


Validation RMSE (Accuracy)
0.000
1.387
2.773
Precipitation Rate (mm/s)
1e−5


Validation Std. Dev. of Residuals (Precision)
0.000
1.387
2.773
Precipitation Rate (mm/s)
1e−5