Training

December 31, 2021

1 Training

- Code by Caroline Juang, c.juang@columbia.edu
- With help from Mark Goldstein (NYU) and Jianing Fang (Columbia EAEE)
- For Machine Learning for the Environmental Sciences, Columbia University
- Professor: Pierre Gentine
- November 2021 - December 2021

1.0.1 Data source

The data has already been downloaded and preprocessed for use in the `Preprocessing` notebook.

For your record, the data were created by Park Williams, from various sources. Download from Box

Fire data is created by Caroline Juang and Park Williams, from the Monitoring Trends in Burn Severity (MTBS) product and government agency databases. Use the `burnarea_combined.nc` file. Download from Box

Variables Used

Just for technical use * EPA ecoregion `epa_l2` * western US region `mask_US`

Variable to predict * forest burned area `burnarea`

Static * fractional forest area `forest` * elevation `elev`

Land cover change * Distance to wildland-urban interface `wui_distance_new` * Months since gridcell burned `return_months` * Years since gridcell burned `return_years`

Climate (and z-variables which represent `observed - 1984-2019 average`) * daily maximum temperature `Tmax` `Tmax_z` * vapor-pressure deficit `vpd` `vpd_z` * relative humidity `rh` `rh_z` * precipitation `prec` `prec_z` * wind `wind`

1.0.2 Ecoregions

- 10.1 Cold deserts
- 7.1 Marine west coast forest
- 11.1 Mediterranean California
- 9.4 South Central Semiarid Prairies
- 9.2 Temperate Prairies (not included)
- 13.1 Upper Gila Mountains
- 10.2 Warm Deserts

- 9.3 West-Central Semiarid Prairies
- 6.2 Western Cordillera
- 12.1 Western Sierra Madre Piedmont

1.0.3 Workspace Setup

[1]:
```python
# import
import numpy as np
import xarray as xr
import pandas as pd
import matplotlib.pyplot as plt
```

1.0.4 Import data created using `Preprocessing`

[2]:
```python
# import preprocessed datasets

epa_l2 = xr.open_dataset('data\\epa_l2.nc') # static
maskUS = xr.open_dataset('data\\maskUS.nc')
forest = xr.open_dataset('data\\forest.nc')
elevstd = xr.open_dataset('data\\elevstd.nc')

vpd = xr.open_dataset('data\\vpd.nc') # climate
rh = xr.open_dataset('data\\rh.nc')
tmax = xr.open_dataset('data\\tmax.nc')
prec = xr.open_dataset('data\\prec.nc')
wind = xr.open_dataset('data\\wind.nc')
vpd_z = xr.open_dataset('data\\vpd_z.nc')
rh_z = xr.open_dataset('data\\rh_z.nc')
tmax_z = xr.open_dataset('data\\tmax_z.nc')
prec_z = xr.open_dataset('data\\prec_z.nc')

burnarea = xr.open_dataset('data\\burnarea.nc') # y variable to predict
wui_distance_new = xr.open_dataset('data\\wui_distance_new.nc') # land change
return_years = xr.open_dataset('data\\return_years.nc')
return_months = xr.open_dataset('data\\return_months.nc')
```

[3]:
```python
# format variables
time = burnarea.time
elevstd_new = elevstd.expand_dims({'time':time}) # expand to include monthly␣
↪data (static variable)
elevstd_new = elevstd_new.elevstd

# get only forested areas
mask = forest>0.50

#vpd = vpd.where(mask)
#rh = rh.where(mask)
```

```
#tmax = tmax.where(mask)
#prec = prec.where(mask)
#wind = wind.where(mask)
#vpd_z = vpd_z.where(mask)
#rh_z = rh_z.where(mask)
#tmax_z = tmax_z.where(mask)
#prec_z = prec_z.where(mask)
#burnarea = burnarea.where(mask)
#wui_distance_new = wui_distance_new.where(mask)
#return_years = return_years.where(mask)
#return_months = return_months.where(mask)
#elevstd_new = elevstd_new.where(mask)
```

[34]:
```
# transform xarray dataarrays to numpy arrays

elev_np = elevstd_new.values
vpd_np = vpd.__xarray_dataarray_variable__.values
rh_np = rh.rh.values
tmax_np = tmax.tmax.values
prec_np = prec.prec.values
wind_np = wind.wind.values #
vpdz_np = vpd_z.__xarray_dataarray_variable__.values #
rhz_np = rh_z.__xarray_dataarray_variable__.values
tmaxz_np = tmax_z.__xarray_dataarray_variable__.values #
burn_np = burnarea.burnarea.values #
wui_np = wui_distance_new.wui_distance.values
returnm_np = return_months.__xarray_dataarray_variable__.values #
```

[41]:
```
# export outputs of np arrays

with open('wind_np2d.npy', 'wb') as f:
    np.save(f,x)
#with open('x.npy', 'rb') as f:
#    y = np.load(f)
print(x.shape)
print(x)
print(y.shape)
print(y)

# convert 3d arrays into 2d
# https://www.geeksforgeeks.org/
 ↪how-to-load-and-save-3d-numpy-array-to-file-using-savetxt-and-loadtxt-functions/
 ↪
 = wind_np.reshape(wind_np.shape[0], -1)
vpdz_np2d = vpdz_np.reshape(vpdz_np.shape[0], -1)
vpd_np2d = vpd_np.reshape(vpd_np.shape[0], -1)
tmax_np2d = tmax_np.reshape(tmax_np.shape[0], -1)
```

```
tmaxz_np2d = tmaxz_np.reshape(tmaxz_np.shape[0], -1)
burn_np2d = burn_np.reshape(burn_np.shape[0], -1)
returnm_np2d = returnm_np.reshape(returnm_np.shape[0], -1)


# save outputs of the numpy arrays
wind_np = wind.wind.values #
np.savetxt("data\\wind_np.txt", wind_np2d, delimiter=",")
np.savetxt("data\\vpdz_np.txt", vpdz_np2d, delimiter=",")
np.savetxt("data\\vpd_np.txt", vpd_np2d, delimiter=",")
np.savetxt("data\\tmax_np.txt", tmax_np2d, delimiter=",")
np.savetxt("data\\tmaxz_np.txt", tmaxz_np2d, delimiter=",")
np.savetxt("data\\burn_np.txt", burn_np2d, delimiter=",")
np.savetxt("data\\returnm_np.txt", returnm_np2d, delimiter=",")
```

[5]:
```
# one issue, there are NaNs where there
# is ocean/ areas that are not the western US.
# remove these by converting to 0.

wind_np[np.isnan(wind_np)] = 0
vpdz_np[np.isnan(vpdz_np)] = 0
tmaxz_np[np.isnan(tmaxz_np)] = 0
burn_np[np.isnan(burn_np)] = 0
returnm_np[np.isnan(returnm_np)] = 0
```

1.0.5 Use numpy arrays and preprocess for the model

We are following the ConvLSTM keras tutorial here: https://keras.io/examples/vision/conv_lstm/

[6]:
```
# import
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

import io
import imageio
from IPython.display import Image, display
from ipywidgets import widgets, Layout, HBox

# Use GPU if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[7]:
```
# load the variables
feature1 = burn_np
feature2 = wind_np
feature3 = vpdz_np
feature4 = tmaxz_np
```

```
feature5 = returnm_np

# we trim the dataset to make it easily divisible
# we swap the dimensions to fit
def trim(x):
    # make divisible by 10 to make separate datapoints later
    return x[:,:200,:150]
def swap(x):
    # swap time dim to end
    return np.swapaxes(x, 0, 2)

# note burned_area is channel 1
# important for defining target later
feature1 = swap(trim(feature1))
feature2 = swap(trim(feature2))
feature3 = swap(trim(feature3))
feature4 = swap(trim(feature4))
feature5 = swap(trim(feature5))

# first channel is burned area
dataset = np.stack([feature1,
                    feature2,
                    feature3,
                    feature4,
                    feature5])
print(dataset.shape)
# change dataset from (channels,dim1,dim2,time)
# to (time,dim1,dim2,channels)
dataset = np.transpose(dataset,(3,1,2,0))
print(dataset.shape)
```

```
(5, 150, 200, 432)
(432, 150, 200, 5)
```

[8]:
```
# break spatial data into (blocksize,blocksize) blocks
new_data = []
blocksize = 10
dim1=dataset.shape[1]
dim2=dataset.shape[2]
# the below code will change dataset
# from (timesteps, dim1, dim2, channels)
# to (datapoints, timesteps, blocksize, blocksize, channels)
for i in range(int(dim1/blocksize)):
    x_start = i*blocksize
    x_end = x_start + blocksize
    for j in range(int(dim2/blocksize)):
        y_start = j*blocksize
```

```
        y_end = y_start+blocksize
        datapoint = dataset[:,x_start:x_end,y_start:y_end,:]
        new_data.append(datapoint)
dataset_10 = np.stack(new_data)
#print(dataset_10.shape)
```

1.0.6 Split dataset into training and validation

[9]:
```
# split into train and test.
# for now, train on earlier data, test on later data
# def train by first 232 of 432 timesteps.
# 232 not so important but good to keep train/test sequences
# similar length
# later... you may choose to not split train/test by time.
train_dataset = dataset_10[:,:232,:,:,:]
test_dataset = dataset_10[:,232:,:,:,:]
print(train_dataset.shape)
print(test_dataset.shape)
```

```
(300, 232, 10, 10, 5)
(300, 200, 10, 10, 5)
```

[10]:
```
# splits things into (input,output)
def make_input_output(dataset):
    # dataset is (N,time,dim1,dim2,channels)
    timesteps = dataset.shape[1]
    burned_area_channel = 0
    # two things are done here
    # first, shift targets y by one timestep from x as in demo
    # second, only burned area channel used for outputs
    x = dataset[:, 0 : timesteps - 1, :,:,:]
    y = dataset[:, 1 : timesteps    , :,:,burned_area_channel]

    # finally, here is where you might want to binarize y
    # for yes/no predictions rather than how much burned.
    y = (y>0).astype(int)
    return x,y

x_train,y_train = make_input_output(train_dataset)
x_val,y_val = make_input_output(test_dataset)
print(x_train.shape)
print(y_train.shape)
print(x_val.shape)
print(y_val.shape)
# notice that num timesteps is reduced by 1
# because we have nothing to predict from
# the last timestep
```

```
(300, 231, 10, 10, 5)
(300, 231, 10, 10)
(300, 199, 10, 10, 5)
(300, 199, 10, 10)
```

1.0.7 Model construction

[19]:
```
# Construct the input layer with no definite frame size.
inp = layers.Input(shape=(None, *x_train.shape[2:]))

# We will construct 3 `ConvLSTM2D` layers with batch normalization,
# followed by a `Conv3D` layer for the spatiotemporal outputs.
x = layers.ConvLSTM2D(
    filters=8,
    kernel_size=(5, 5),
    padding="same",
    return_sequences=True,
    activation="sigmoid",
)(inp)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=8,
    kernel_size=(3, 3),
    padding="same",
    return_sequences=True,
    activation="sigmoid",
)(x)
x = layers.BatchNormalization()(x)
x = layers.ConvLSTM2D(
    filters=8,
    kernel_size=(1, 1),
    padding="same",
    return_sequences=True,
    activation="sigmoid",
)(x)

# assuming binary targets here, but see below for another option
x = layers.Conv3D(
    filters=1, kernel_size=(3, 3, 3), activation="sigmoid", padding="same"
)(x)

# Next, we will build the complete model and compile it.
model = keras.models.Model(inp, x)
model.compile(
    loss=keras.losses.binary_crossentropy, optimizer=keras.optimizers.Adam(),␣
 ↪metrics=['accuracy']
)
```

```
# for predicting how much will burn rather than yes/no
# you would have to
# (1) change sigmoid to relu in the last layer
# (2) change loss to  loss=keras.losses.MeanSquaredError
```

1.0.8 Model training

[30]:
```
# Define some callbacks to improve training.
early_stopping = keras.callbacks.EarlyStopping(monitor="val_loss", patience=10)
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor="val_loss", patience=5)

# Define modifiable training hyperparameters.
epochs = 20
batch_size = 5

# Fit the model to the 1training data.
history = model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(x_val, y_val),
    callbacks=[early_stopping, reduce_lr],
)
```

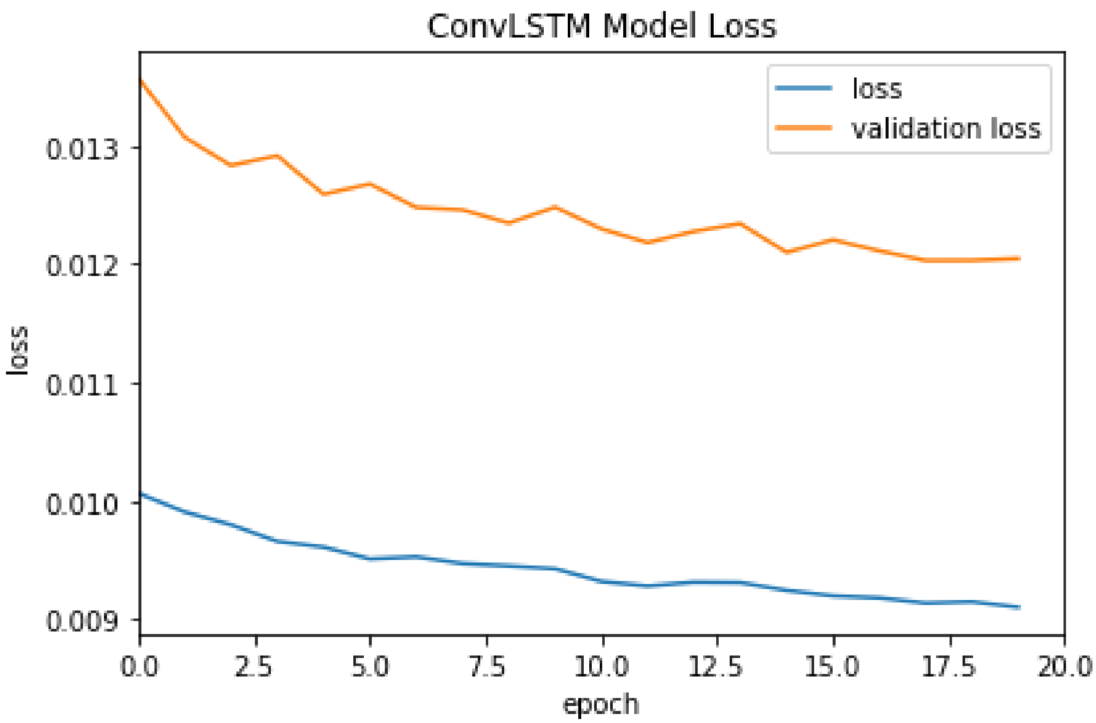
```
Epoch 1/20
60/60 [==============================] - 106s 2s/step - loss: 0.0101 - val_loss:
0.0136 - lr: 1.0000e-04
Epoch 2/20
60/60 [==============================] - 104s 2s/step - loss: 0.0099 - val_loss:
0.0131 - lr: 1.0000e-04
Epoch 3/20
60/60 [==============================] - 115s 2s/step - loss: 0.0098 - val_loss:
0.0128 - lr: 1.0000e-04
Epoch 4/20
60/60 [==============================] - 109s 2s/step - loss: 0.0097 - val_loss:
0.0129 - lr: 1.0000e-04
Epoch 5/20
60/60 [==============================] - 92s 2s/step - loss: 0.0096 - val_loss:
0.0126 - lr: 1.0000e-04
Epoch 6/20
60/60 [==============================] - 97s 2s/step - loss: 0.0095 - val_loss:
0.0127 - lr: 1.0000e-04
Epoch 7/20
60/60 [==============================] - 103s 2s/step - loss: 0.0095 - val_loss:
0.0125 - lr: 1.0000e-04
Epoch 8/20
60/60 [==============================] - 103s 2s/step - loss: 0.0095 - val_loss:
```

```
0.0125 - lr: 1.0000e-04
Epoch 9/20
60/60 [==============================] - 87s 1s/step - loss: 0.0095 - val_loss:
0.0123 - lr: 1.0000e-04
Epoch 10/20
60/60 [==============================] - 87s 1s/step - loss: 0.0094 - val_loss:
0.0125 - lr: 1.0000e-04
Epoch 11/20
60/60 [==============================] - 134s 2s/step - loss: 0.0093 - val_loss:
0.0123 - lr: 1.0000e-04
Epoch 12/20
60/60 [==============================] - 98s 2s/step - loss: 0.0093 - val_loss:
0.0122 - lr: 1.0000e-04
Epoch 13/20
60/60 [==============================] - 83s 1s/step - loss: 0.0093 - val_loss:
0.0123 - lr: 1.0000e-04
Epoch 14/20
60/60 [==============================] - 83s 1s/step - loss: 0.0093 - val_loss:
0.0123 - lr: 1.0000e-04
Epoch 15/20
60/60 [==============================] - 159s 3s/step - loss: 0.0092 - val_loss:
0.0121 - lr: 1.0000e-04
Epoch 16/20
60/60 [==============================] - 88s 1s/step - loss: 0.0092 - val_loss:
0.0122 - lr: 1.0000e-04
Epoch 17/20
60/60 [==============================] - 82s 1s/step - loss: 0.0092 - val_loss:
0.0121 - lr: 1.0000e-04
Epoch 18/20
60/60 [==============================] - 145s 2s/step - loss: 0.0091 - val_loss:
0.0120 - lr: 1.0000e-05
Epoch 19/20
60/60 [==============================] - 170s 3s/step - loss: 0.0092 - val_loss:
0.0120 - lr: 1.0000e-05
Epoch 20/20
60/60 [==============================] - 135s 2s/step - loss: 0.0091 - val_loss:
0.0120 - lr: 1.0000e-05
```

1.0.9 Validation

[48]:
```
# trying to follow this:
# https://stackoverflow.com/questions/41908379/
 ↪keras-plot-training-validation-and-test-set-accuracy

fig, ax = plt.subplots()
ax.plot(history.history['loss'], label='loss')
ax.plot(history.history['val_loss'], label='validation loss')
```

```
ax.set_title('ConvLSTM Model Loss')
ax.set_ylabel('loss')
ax.set_xlabel('epoch')
ax.legend()
ax.xlim([0,20])
fig.tight_layout()

plt.savefig('results//ModelLoss.png')
```



[85]:
```
np.shape(x_val)
```

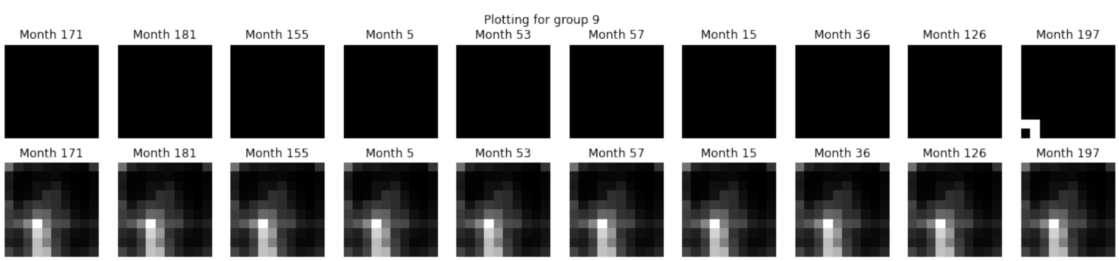
[85]: (300, 199, 10, 10, 5)

1.0.10 Visualize what our model has learned so far

[126]:
```
# select a few random examples from the dataset.
rand = np.random.choice(range(len(test_dataset)), size=5)
example = x_train[rand[0]]

# Pick 10 randomized frames from the example.
rand2 = np.random.choice(range(199), size=10)
frames = example[rand2, ...]
frames_new = np.empty((10, 10, 10))
```

```
original_frames = y_train[rand[0],rand2, ...]

# Predict a new set of 10 frames.
for _ in range(10):
    # Extract the model's prediction and post-process it.
    new_prediction = model.predict(np.expand_dims(frames, axis=0))
    new_prediction = np.squeeze(new_prediction)
    new_prediction = np.squeeze(new_prediction)
    predicted_frame = np.expand_dims(new_prediction[-1, ...], axis=0)

    # Extend the set of prediction frames.
    frames_new = np.concatenate((frames_new, predicted_frame))

# Construct a figure for the original and new frames.
fig, axes = plt.subplots(2, 10, figsize=(20, 4))

# Plot the original frames.
for idx, ax in enumerate(axes[0]):
    ax.imshow(np.squeeze(original_frames[idx]), cmap="gray")
    ax.set_title(f"Month {rand2[idx]}")
    ax.axis("off")

# Plot the new frames.
new_frames = frames_new[10:, ...]
for idx, ax in enumerate(axes[1]):
    ax.imshow(np.squeeze(new_frames[idx]), cmap="gray")
    ax.set_title(f"Month {rand2[idx]}")
    ax.axis("off")

fig.suptitle(f"Plotting for group {rand[0]}")

# Display the figure.
```

[126]: Text(0.5, 0.98, 'Plotting for group 9')



[ ]: