Projects

Deliverables: Poster & Report & main code (plus proposal, midterm slide)

Topics your own or chose form suggested topics. Some physics/engineering inspired.

April 26 groups due to TA (if you don’t have a group, ask in piazza we can help). TAs will construct groups after that.

May 5 proposal due. TAs and Peter can approve.
Proposal: One page: Title, a large paragraph, data, weblinks, references.

May 20 Midterm slide presentation. Presented to a subgroup of class.

June 5 final poster. Uploaded June 3
Report and code due Saturday 15 June.

Q: Can the final project be shared with another class?
If the other class allows it it should be fine. You cannot turn in an identical project for both classes, but you can share common infrastructure/code base/datasets across the two classes.

No cut and paste from other sources without making clear that this part is a copy. This applies to other reports or things from internet. Citations are important.
Last time: Data Preprocessing

**Before normalization**: classification loss very sensitive to changes in weight matrix; hard to optimize.

**After normalization**: less sensitive to small changes in weights; easier to optimize.
Optimization: Problems with SGD

What if loss changes quickly in one direction and slowly in another? What does gradient descent do? Very slow progress along shallow dimension, jitter along steep direction

Loss function has high condition number: ratio of largest to smallest singular value of the Hessian matrix is large
What if the loss function has a local minima or saddle point?

Zero gradient, gradient descent gets stuck

Saddle points much more common in high dimension
Optimization: Problems with SGD

Our gradients come from minibatches so they can be noisy!

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) \]
\[ \nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) \]

**SGD + Momentum**

**SGD**

\[ x_{t+1} = x_t - \alpha \nabla f(x_t) \]

**SGD+Momentum**

\[ v_{t+1} = \rho v_t + \nabla f(x_t) \]
\[ x_{t+1} = x_t - \alpha v_{t+1} \]

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99
Adam (full form)

```python
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

Bias correction for the fact that first and second moment estimates start at zero.

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!
SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have **learning rate** as a hyperparameter.

=> Learning rate decay over time!

**step decay:**
e.g. decay learning rate by half every few epochs.

**exponential decay:**

\[ \alpha = \alpha_0 e^{-kt} \]

**1/t decay:**

\[ \alpha = \frac{\alpha_0}{1 + kt} \]
How to improve single-model performance?

Regularization: Add term to loss

\[ L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W) \]

In common use:

**L2 regularization**

\[ R(W) = \sum_k \sum_l W_{k,l}^2 \quad \text{(Weight decay)} \]

**L1 regularization**

\[ R(W) = \sum_k \sum_l |W_{k,l}| \]

**Elastic net (L1 + L2)**

\[ R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \]
Regularization: Dropout

In each forward pass, randomly set some neurons to zero.
Probability of dropping is a hyperparameter; 0.5 is common.
Regularization: Dropout
How can this possibly be a good idea?

Forces the network to have a redundant representation;
Prevents co-adaptation of features

- has an ear
- has a tail
- is furry
- has claws
- mischievous look

X

X

X

X

score

cat
Regularization: Data Augmentation

Data Augmentation
Get creative for your problem!

Random mix/combinations of:
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Load image and label
“cat”

Compute loss
+ simulated data using physical model.
Transfer Learning with CNNs

1. Train on Imagenet

   - FC-1000
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

2. Small Dataset (C classes)

   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Reinitialize this and train

3. Bigger dataset

   - FC-C
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64
   - Image

   Train these

   - With bigger dataset, train more layers

   - Freeze these

   - Lower learning rate when finetuning; 1/10 of original LR is good starting point

Predicting Weather with Machine Learning:
Intro to ARMA and Random Forest

Emma Ozanich
PhD Candidate,
Scripps Institution of Oceanography
Background

Shi et al NIPS 2015 –

• Predicting rain at different time lags
• Shows convolutional lstm vs nowcast models vs fully-connected lstm
• Used radar echo (image) inputs
  o Hong Kong, 2011-2013,
  o 240 frames/day
  o Selected top 97 rainy days
  o Note: <10% of data used!
• Preprocessing: k-means clustering to denoise
• ConvLSTM has better performance and lower false alarm (lower left)

CSI: hits/(hits+misses+false)  FAR: false/(hits+false)  POD: hits/(hits+misses)
false = false alarm
Background

McGovern et al 2017 BAM –

- Decision trees used in meteorology since mid-1960s

Predicting rain at different time lags

![Decision Tree Diagram]

Fig. 1. An example of a decision tree for predicting if hail will occur. A version of this decision tree first appeared in Gagne (2016).

Background

McGovern et al 2017 BAM –
• Green contours = hail occurred (truth)
• Physics based method: Convection-allowing model (CAM)
  o Doesn’t directly predict hail
• Random forest predicts hail size ($\Gamma$) distribution based on weather
• HAILCAST = diagnostic measure based on CAMs
• Updraft Helicity = surrogate variable from CAM

Decision Trees

- Algorithm made up of conditional control statements

```
Homework Deadline tonight?
  Yes
  Do homework
  No
  Party invitation?
    Yes
    Go to the party
    No
    Do I have friends
      Yes
      Hang out with friends
      No
      Read a book
    No
```

Do homework

Decision Trees

McGovern et al 2017 BAM –
- Decision trees used in meteorology since mid-1960s

Fig. 1. An example of a decision tree for predicting if hail will occur. A version of this decision tree first appeared in Gagne (2016).

Regression Tree

• Divide data into distinct, non-overlapping regions \( R_1, \ldots, R_j \)
• Below \( y_i = \text{color} = \) continuous target \( (<\text{blue} = 1 \text{ and } >\text{red} = 0) \).
• \( x_i, i = 1, \ldots, 5 \) samples
• \( x_i = [X_1, X_2], \) with \( P = 2 \) features.
• \( j = 1, \ldots, 5 \) (5 regions).

\[
\begin{align*}
X_1 & \leq t_1 \\
X_2 & \leq t_2 \\
X_1 & \leq t_3 \\
X_2 & \leq t_4 \\
\end{align*}
\]

\( R_1 \quad R_2 \quad R_3 \quad R_4 \quad R_5 \)

Hastie et al 2017, Chap. 9 p 307.
Tree-building

- Or, consecutively partition a region into non-overlapping rectangles
- $y_i =$ color = continuous target ($<\text{blue} = 1$ and $>\text{red} = 0$).
- $x_i, i = 1,\ldots,5$ samples
- $x_i = [X_1, X_2]$, with $P = 2$ features.
- $j = 1,\ldots,5$ (5 regions).

Hastie et al 2017, Chap. 9 p 307.
Regression Tree

- How to optimize a regression tree?
- Randomly select $t_1$
  \[ R_1(j, \ t_1) = \{X|X_j \leq t_1\} \]
  \[ R_2(j, \ t_1) = \{X|X_j > t_1\}, \ j = 1 \]
- Assign region labels:
  \[ \hat{c}_m = \text{ave}(y_i|x_i \in R_m). \]
  - Example-
    \[ \hat{c}_1 = \]
    \[ \hat{c}_2 = \]
  
\[ \hat{c}_1 = \]
\[ \hat{c}_2 = \]
Regression Tree

- Compute the cost of the tree, $Q_m(T)$,
- Minimize $Q_m(T)$ by changing $t_1$

$$N_m = \#\{x_i \in R_m\},$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$
Regression Tree

• Algorithm to build tree $T_b$
• In our simple case, $m = 1$ and $p = 2$
• Daughter nodes are equivalent to regions

1. Select $m$ variables at random from the $p$ variables.
2. Pick the best variable/split-point among the $m$.
3. Split the node into two daughter nodes.

$X_2$

$X_1$

$R_1$

$R_2$

$h_2 < \rho$
Bootstrap samples

- Select a subset of the total samples, \((x^*_i, y^*_i), i = 1, \ldots, N\)
- Draw samples uniformly at random with replacement
- Example: If \(I = 5\) originally, we could choose \(N = 2\)

- Samples are drawn assuming equal probability:
  - If \(x_i, y_i\) is drawn more often, it is more likely
    \[
    P_{\hat{F}}\{(X, Y) = (x, y)\} = \begin{cases} \frac{1}{n} & \text{if } (x, y) = (x_i, y_i) \text{ for some } i \\ 0 & \text{otherwise} \end{cases}
    \]
  - \((X, Y)\) are the expectations of the underlying distributions
Random Forest

- Example of binary classification tree from Hastie et al 2017
- Orange: trained on all data
- Green: trained from different bootstrap samples
- Then, average the (green) trees

Hastie et al 2017, Chap. 8 p. 284
Random Forest

- Bootstrap + bagging => more robust RF on future test data
- Train each tree $T_b$ on bootstrap sampling

**Algorithm 15.1 Random Forest for Regression or Classification.**

1. For $b = 1$ to $B$:
   
   (a) Draw a bootstrap sample $Z^*$ of size $N$ from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      
      i. Select $m$ variables at random from the $p$ variables.
      ii. Pick the best variable/split-point among the $m$.
      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_{1}^{B}$.

To make a prediction at a new point $x$:

**Regression:** $\hat{f}_{rf}^{B}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$.

Hastie et al 2017, Chap. 15 p. 588
Timeseries (TS)

- Timeseries: one or more variables sampled in the same location at successive time steps
ARMA

• Autoregressive moving-average :
  o (weakly) stationary stochastic process
  o Polynomials model process and errors as polynomial of prior values

• Autoregressive (order p)
  o Linear model of past (lagged) and future values
  o $p$ lags

\[ X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \]

  o $\varphi_i$ are (weights) parameters
  o $c$ is constant
  o $\varepsilon_t$ is white noise (WGN)
  o Note, for stationary processes, $|\varphi_i| < 1$.

• Moving-average (order q)

\[ X_t = c + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t \]

  o Linear model of past errors
  o $q$ lags
  o Below, assume $<X_t>=0$ (expectation is 0)
ARMA

- Autoregressive moving-average:
  - (weakly) stationary stochastic process
  - Linear model of prior values = expected value term + error term + WGN

- ARMA: AR(p) + MA(q)

\[ X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t \]
Data retrieval

Just a few public data sources for physical sciences...

- **NOAA:**
  - Reanalysis/model data, research cruises, station observations, gridded data products, atmospheric & ocean indices timeseries, heat budgets, satellite imagery
- **NASA:**
  - EOSDIS, gridded data products (atmospheric), satellite imagery, reanalysis/model data, meteorological stations, DAAC’s in US
- **IMOS:**
  - Ocean observing hosted by Australian Ocean Data Network
- **USGS Earthquake Archives**
- **CPC/NCEI:**
  - Gridded and raw meteorological and oceanographic data
- **ECMWF**
  - Global-scale weather forecasts and assimilated data

Possible data formats:
- CSV
- NetCDF
- HDF5/HDF-EOS
- Binary
- JPEG/PNG
- ASCII text

......
Basic data cleaning

- “[ML for physical sciences] is 80% cleaning and 20% models” ~ paraphrased, Dr. Gerstoft
- Basic cleaning for NOAA GSOD to HW – necessary
  - Remove unwanted variables (big data is slow)
  - Replaced “9999” filler values with NaN
  - Converted strings to floats (i.e. for wind speed)
  - Created a DateTime index

- Physical data needs cleaning, reorganizing
- Quality-controlled data still causes bugs
- Application-specific
Data for HW

BigQuery:
- Open-source database hosted by Google
- Must have Google account
- 1 TB data free/month

NOAA GSOD dataset
Data for HW

- How to get BigQuery data?
- `bigquery` package in Jupyter Notebook (SQL server)

- More complex queries may include dataframe joins, aggregations, or subsetting
Tutorial Notebook

• Open "In-Class Tutorial"
• We will do:
  1. Load preprocessed data
  2. Define timeseries index
  3. Look at data
  4. Visualize station
  5. Detrend data
  6. Smooth data
  7. Try ARMA model
Tutorial Notebook

• Load packages, (pre-processed) data with Pandas

```python
In [60]:
import pandas as pd
import numpy as np
from numpy.random import randint
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import time
from mpl_toolkits.basemap import Basemap
import scipy.signal as sig
from statsmodels.tsa.arima_model import ARMA
from pandas.plotting import autocorrelation_plot
```

Load our pre-processed data.

```python
In [2]:
t0 = time.time()
dataset_path = '/datasets/NOAA_SST/
data = pd.read_pickle(dataset_path + 'noaa_gsod/Combined_noaa_gsod') # load weather data
stations = pd.read_pickle(dataset_path + 'noaa_gsod.stations') # load station data

# # USE ONLY 2008-2018 # #
data = data.loc[data.index >= pd.Timestamp(2008, 1, 1)]
data = data.drop(columns=['yr', 'year', 'da', 'mo']) # don't need these anymore
print(time.time() - t0)
22.46479368209839
```
Timeseries processing

- We may be missing data, but that’s ok for now
- Replace with neighbor data, smooth, fill with mean
Tutorial Notebook

Where is my station located?

In [5]:
my_station = stations.loc[stations['usaf'] == my_station_number]
my_station.head()

Out[5]:
```
  usaf  wban  name  country  state  call  lat  lon  elev  begin  end
18673 041560 99999  HVERAVELLIR  IC  None  None  64.867  -16.567  +0641.0  20080115  20190401
```

In [6]:
fig = plt.figure(figsize=(15,7))

# create a Basemap projection, cylindrical centered at 0
n = Basemap(projection='cyl',llcrnrlat=-90,urcrnrlat=90,
llcrnrlon=-180,urcrnrlon=180, resolution='l')

# draw the oceans and countries and lines
n.drawmapboundary(fill_color='xkcd:lightblue')
n.fillcontinents(color='xkcd:green',lake_color='xkcd:lightblue')
n.drawmeridians(np.arange(0., 350., 30.), labels=[True,False,False,True])
n.drawparallels(np.arange(-90., 90, 30.), labels=[False,True,True,False])

# show my station
pl, = plt.plot(my_station['lon'],my_station['lat'],'rp', markersize=10)
plt.show()
Timeseries processing

- Remove mean (slope=0) or linear (slope ≠ 0)? (linear)
- What can we learn from trend?

![Windspeed in Hveravellir, Iceland](image)

![Difference](image)
Timeseries processing

- Smoothing: median filter
Tutorial Notebook

- Shortened timeseries – Y2018 (final 10%)
- ARMA most effective predicting one step at a time
Tutorial Notebook

• Is ARMA a machine learning technique? (I think so..)
  o Filtering method (like Kalman filter)
  o Data-driven
  o Maximum likelihood
  o Conclusion: statistics-based
Tutorial Notebook

- Autocorrelation:
  - A statistical method to find temporal (or spatial) relations in data
  - When can reject the null hypothesis that the data is statistically similar?
  - E.g. How many time steps before the data is decorrelated

![Autocorrelation Chart]

~40 lags

- Raw
- Median Filter
Tutorial Notebook

- Median filter increases decorrelation scale
  - By averaging neighbor samples
- Raw series is more random
- Use raw timeseries

![Graph showing autocorrelation with median filter effect](image)

- ~3 lags
Tutorial Notebook

- ARMA algorithm:
  1. Train on all previous data
  2. Predict one time step
  3. Add next value to training data
  4. Repeat
Homework

• How to load and preview data with Pandas

```python
import pandas as pd
import numpy as np
from numpy.random import randint
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import time
import glob
from mpl_toolkits.basemap import Basemap

# load data

t0 = time.time()
data = pd.read_pickle('noaa_gsod/Combined_noaa_gsod') # load weather data
stations = pd.read_pickle('noaa_gsod.stations') # load station data

# # USE ONLY 2008-2018 # #
data = data.loc[data.index >= pd.Timestamp(2008, 1, 1)]
data = data.drop(columns=['yr', 'year', 'da', 'mo']) # don't need these anymore

print(time.time() - t0)

stations.head()
```

```
Out[4]:
```

<table>
<thead>
<tr>
<th>usaf</th>
<th>wbam</th>
<th>name</th>
<th>country</th>
<th>state</th>
<th>call</th>
<th>lon</th>
<th>lat</th>
<th>elev</th>
<th>begin</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>007018</td>
<td>99999</td>
<td>WXPOD 7018</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>+7018.0</td>
<td>0.000</td>
<td>20110309</td>
<td>20130730</td>
<td></td>
</tr>
<tr>
<td>007026</td>
<td>99999</td>
<td>WXPOD 7026</td>
<td>AF</td>
<td>None</td>
<td>None</td>
<td>+7026.0</td>
<td>0.000</td>
<td>20120713</td>
<td>20170822</td>
<td></td>
</tr>
<tr>
<td>007070</td>
<td>99999</td>
<td>WXPOD 7070</td>
<td>AF</td>
<td>None</td>
<td>None</td>
<td>+7070.0</td>
<td>0.000</td>
<td>20140923</td>
<td>20150926</td>
<td></td>
</tr>
<tr>
<td>008268</td>
<td>99999</td>
<td>WXPOD8278</td>
<td>AF</td>
<td>None</td>
<td>None</td>
<td>+1156.7</td>
<td>32.95</td>
<td>20100519</td>
<td>20120323</td>
<td></td>
</tr>
<tr>
<td>008307</td>
<td>99999</td>
<td>WXPOD 8318</td>
<td>AF</td>
<td>None</td>
<td>None</td>
<td>+8318.0</td>
<td>0.000</td>
<td>20100421</td>
<td>20100421</td>
<td></td>
</tr>
</tbody>
</table>
```
Homework

- How to load and preview data with Pandas
Homework

- Randomly select a station
- Check if the station has enough data
  - May reduce "3650" to lower number, i.e. 1000, but be aware you may have nans in data – just look at it!
Homework

- Manually time-delay data
- Pandas “shift()”

```
In [8]:
columns = features.columns
for co in columns:
    # one day lag
    features[co + '_lag1'] = features[co].shift(periods=1)
    # two days lag
    features[co + '_lag2'] = features[co].shift(periods=2)
    # three days lag
    features[co + '_lag3'] = features[co].shift(periods=3)
features = features.iloc[3:]
print(str(len(features)) + ' samples, ' + str(len(features.columns)) + ' features.')
features.head()
```

Pandas shift()
remove first 3 entries
Homework

- (Map is supposed to show red “X” for station)

I can barely see it!!
Homework

- Snapshots from “timeseries_prediction_Temp.ipynb”

Create train/val/test

```python
In [8]:
ylabel = features['temp'] # use today's temperature as the labeleatures = features.drop(columns='temp') # don't put it in training data

# Use 20% test split (80% training + validation)
ntrain = int(len(features)*0.8)
x_test = features.iloc[ntrain:]
y_test = ylabel[ntrain:]

# Split remaining 80% into training-validation sets (of original data)
x_train, x_val, y_train, y_val = train_test_split(features.iloc[0:ntrain,:], ylabel[0:ntrain],
                                                test_size=0.2, random_state=1)

# Scale features. Fit scaler on training only.
scaler = MinMaxScaler() # scale features between 0 and 1
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

- training label = temperature
- split data into train/test with the help of Sklearn
- scaling features improves learning
Homework

- Random forest model in a couple lines
- You may want to write a “plot.py” function

```
# Define, train, and predict with random forest
clf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=0)  # define Random Forest object
clf.fit(x_train, y_train)  # train Random Forest
y = clf.predict(x_test)  # predict temperature

# plot predictions
plt.figure(figsize=(15,7))
plt.subplot(1,2,1)
plt.plot(features.iloc[ntrain:].index, y_test, label='Actual Temperature')
plt.plot(features.iloc[ntrain:].index, y, label='Predicted Temperature')
plt.xticks(features.iloc[ntrain:].index[::30], rotation=45)  # set xticks to monthly
myFmt = mdates.DateFormatter('%b-%d-%y')
plt.gca().xaxis.set_major_formatter(myFmt)
plt.ylabel('Daily Temperature (degree Fahrenheit)', fontsize=12)
plt.legend(('Temperature', 'Random Forest Prediction'), fontsize=12, loc=1)
plt.show()

# Plot the feature importances
nfeatures = 10
fi = clf.feature_importances_  # get feature importances
fi_sort = np.argsort(fi)[:nfeatures]  # sort importances most to least
plt.bar(range(nfeatures), fi[fi_sort], width=1, tick_label=features.columns.values[[fi_sort]])  # plot features
plt.ylabel('Feature Importance (avg across trees)', fontsize=12)
plt.xticks(rotation=45)
plt.show()
```

- plotting true and predicted temperature
- look at feature importances
Homework

- Congratulations!
- We showed that tomorrow’s temperature is usually similar to today’s (at this Canada station)

unsurprising result: validates intuition
Takeaway for your projects and beyond:
Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision