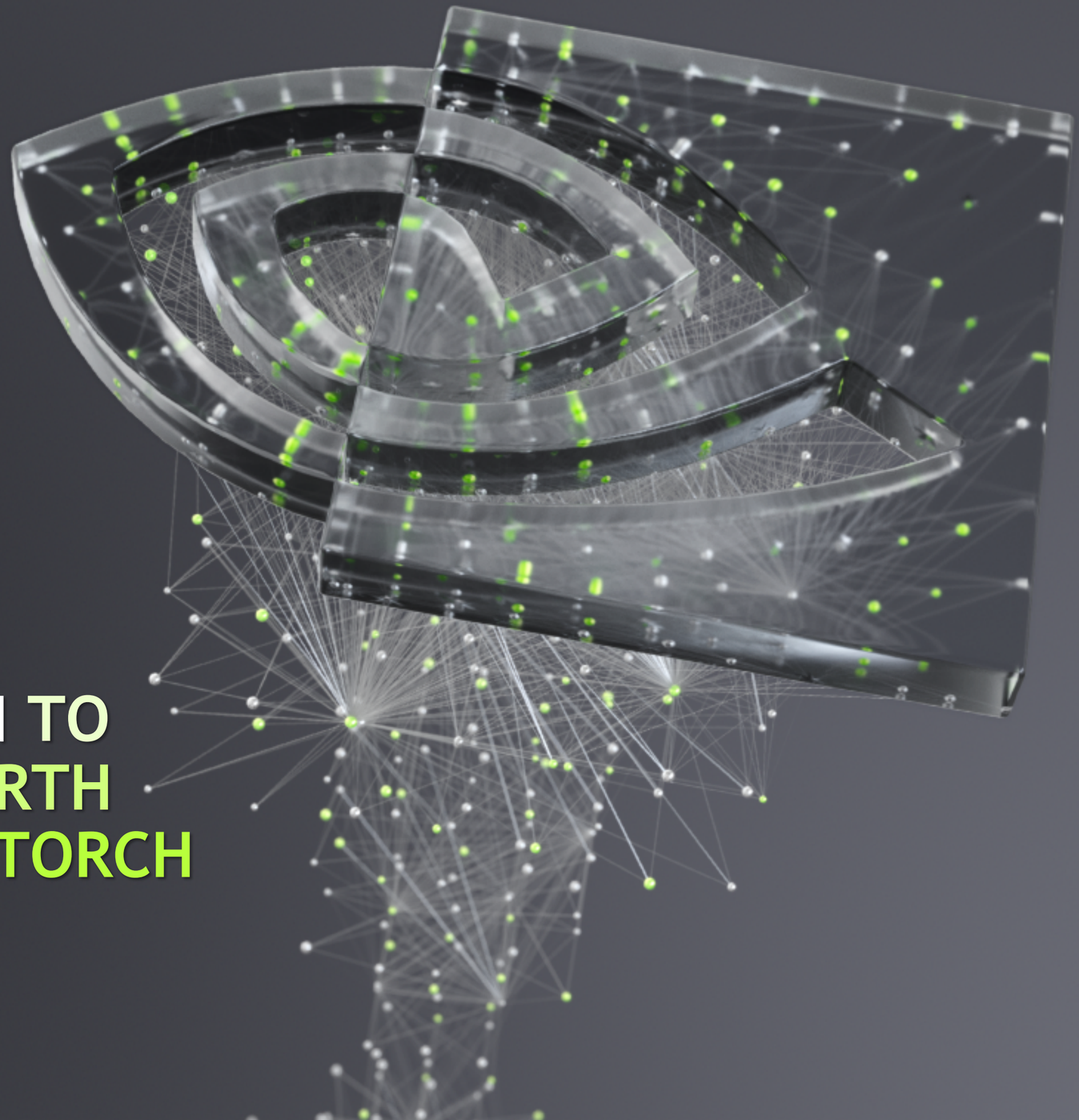


A PRACTICAL INTRODUCTION TO DEEP LEARNING FOR THE EARTH SYSTEM SCIENCES, USING PYTORCH

David Hall, Senior Data Scientist, NVIDIA

NOAA-AI Workshop Tutorial, Nov 2020

WE WILL

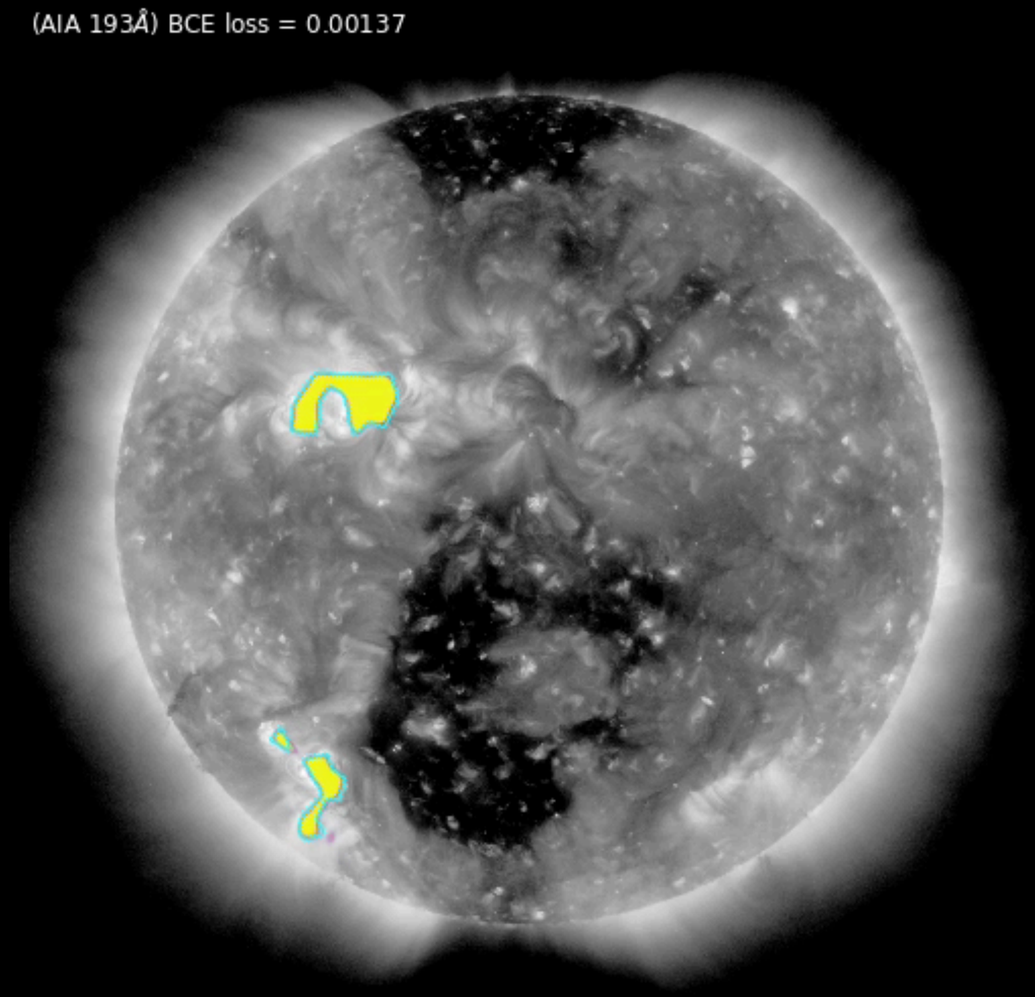
Identify tropical storms and hurricanes with deep learning using PyTorch

WE WILL

Examine Artic Sea-Ice Extent as a ‘warm-up’ exercise

WE CAN

Apply the same techniques to detect features on the sun



LEARNING GOALS

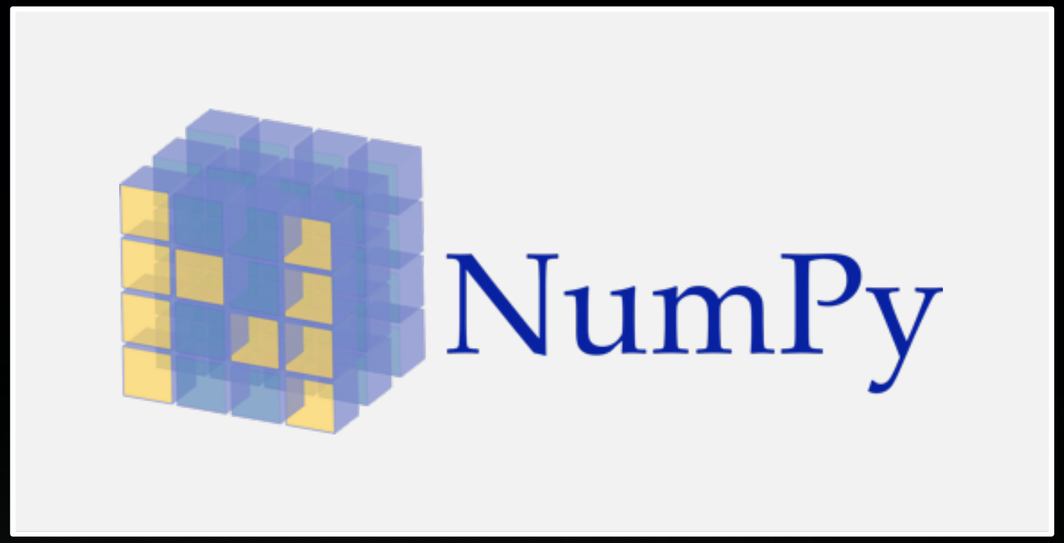
In 2 hours, we can only travel so far

Main goal:
Become familiar with ideas and process

A starting point for solving your own problems

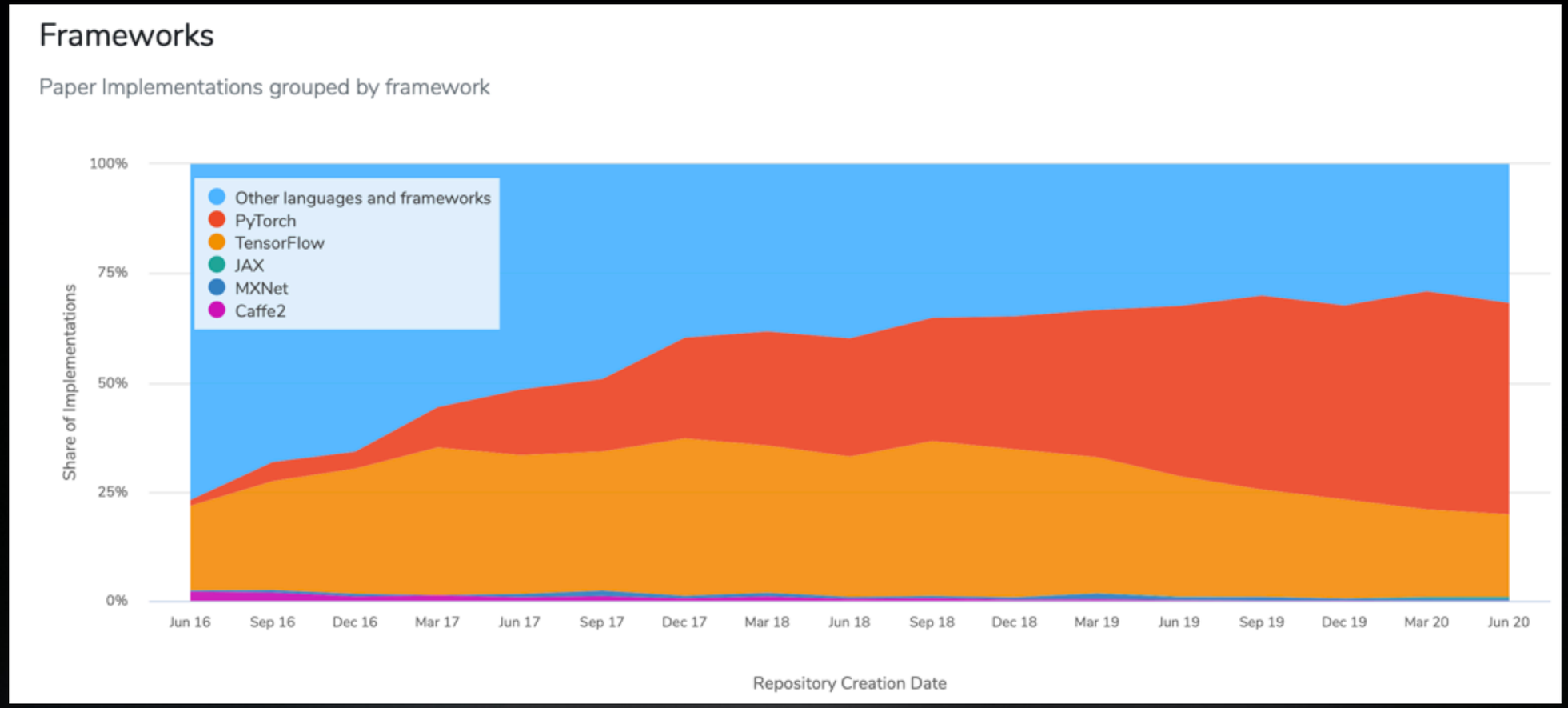
SUGGESTED PRE-REQUISTITES

Familiarity with Python and NumPy are helpful



WHY PYTORCH?

PyTorch is the Preferred Framework for Research



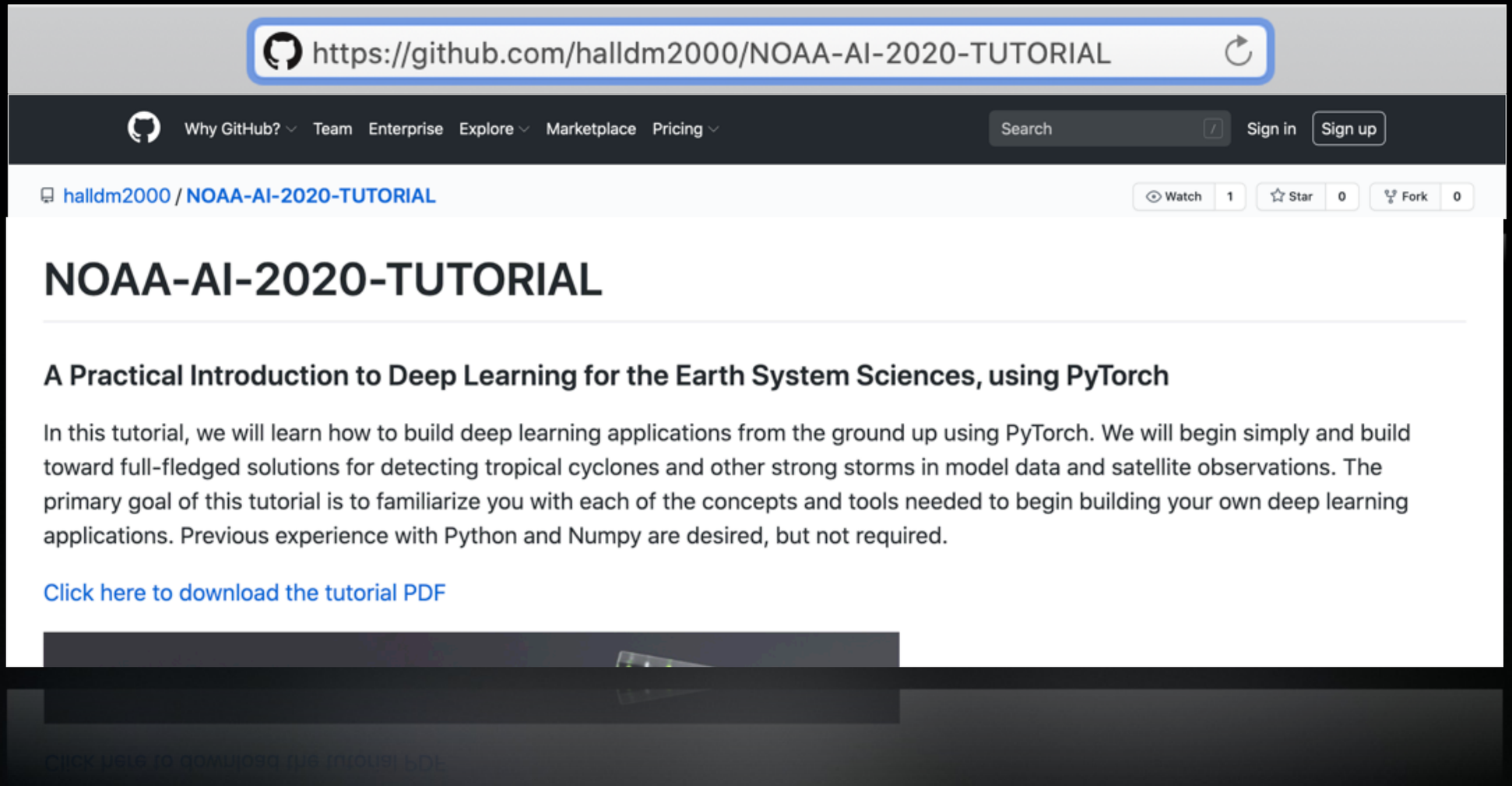
https://paperswithcode.com/trends

AGENDA

• Intro	5 min
• Activity: Intro to Colab	15 min
• What is Deep Learning?	15 min
• Activity : Curve Fitting in PyTorch	15 min
• Fully Connected Networks	20 min
• Break	5 min
• Classification and CNNs	20 min
• Activity : Cyclone Classification	20 min
• Wrap up	5 min

DOWNLOAD THIS PRESENTATION

https://github.com/halldm2000/NOAA-AI-2020-TUTORIAL



HAVING TROUBLE WITH GOOGLE COLAB ?

https://www.google.com/chrome/

Google Colab works best with Google Chrome Browser

ACTIVITY: INTRO TO COLAB

[Click here to launch the notebook]

This Tutorial uses Google Colab.

- Google Colab provides a **Jupyter notebook** on a Google server.
- Jupyter is an interactive coding environment in your web browser.
- Colab is free and gives us access to an **NVIDIA GPU**.
- NVIDIA GPUs make deep learning far more practical.
- Press `[Shift] + [Enter]` to run each cell and move on to the next one.

Launch a GPU Kernel.

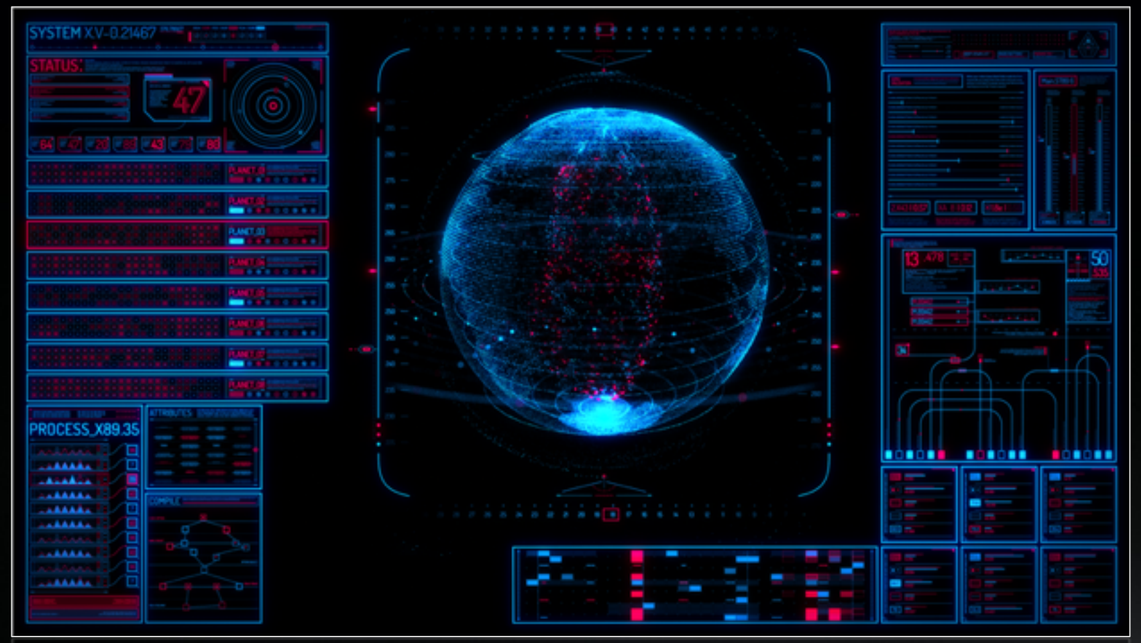
- Let's use the GPU that Colab is providing
- Click on the menu item **Runtime** → **Change runtime type**
- Select Runtime type: **Python 3** and Hardware accelerator: **GPU**

WHAT IS DEEP LEARNING?

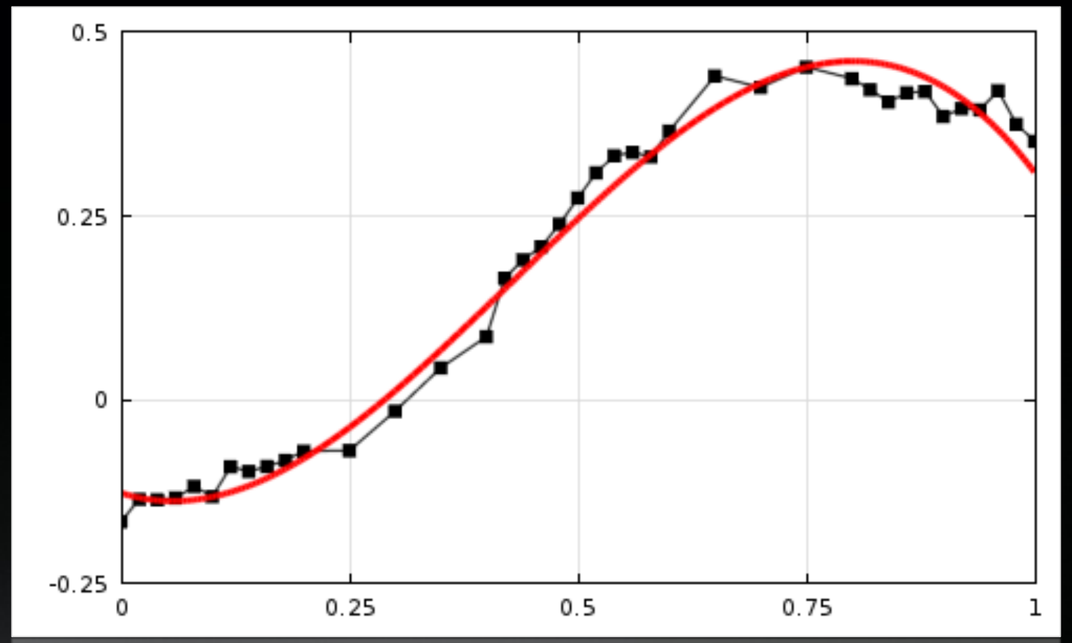
DEEP LEARNING ANALOGIES

What is this deep learning thing, anyway?

A NEW TYPE OF SOFTWARE (POWERFUL!)



A GENERALIZATION OF CURVE FITTING (SIMPLE!)



DEEP LEARNING CAN DO IMPRESSIVE THINGS

DEFEAT WORLD CHAMPION STRATEGISTS

OPERATE VEHICLES AUTONOMOUSLY

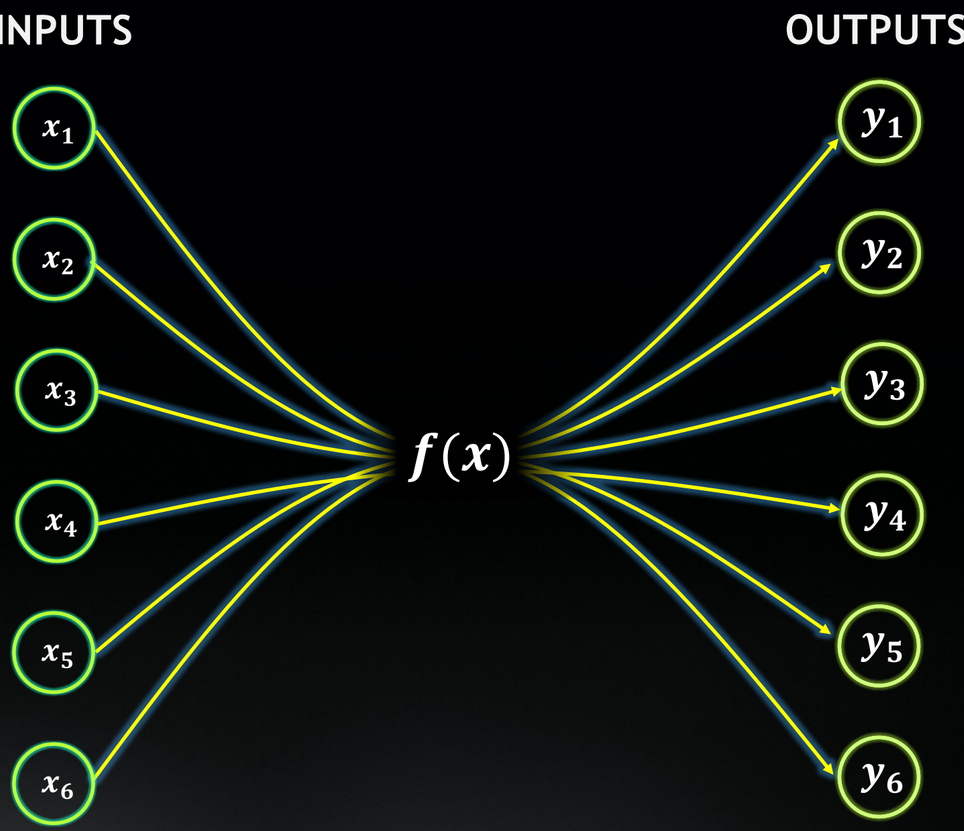
COMMUNICATE IN NATURAL LANGUAGE

GENERATE ORIGINAL CONTENT

BUILD FUNCTIONS FROM EXAMPLES

Find f, given x and y

SUPERVISED DEEP LEARNING



A NEW WAY TO BUILD SOFTWARE

HAND-WRITTEN FUNCTION

```
Function1(T,P,Q)
update_mass()
update_momentum()
update_energy()
do_macrophysics()
do_microphysics()
y = get_precipitation()
return y
```

Convert expert knowledge into a function

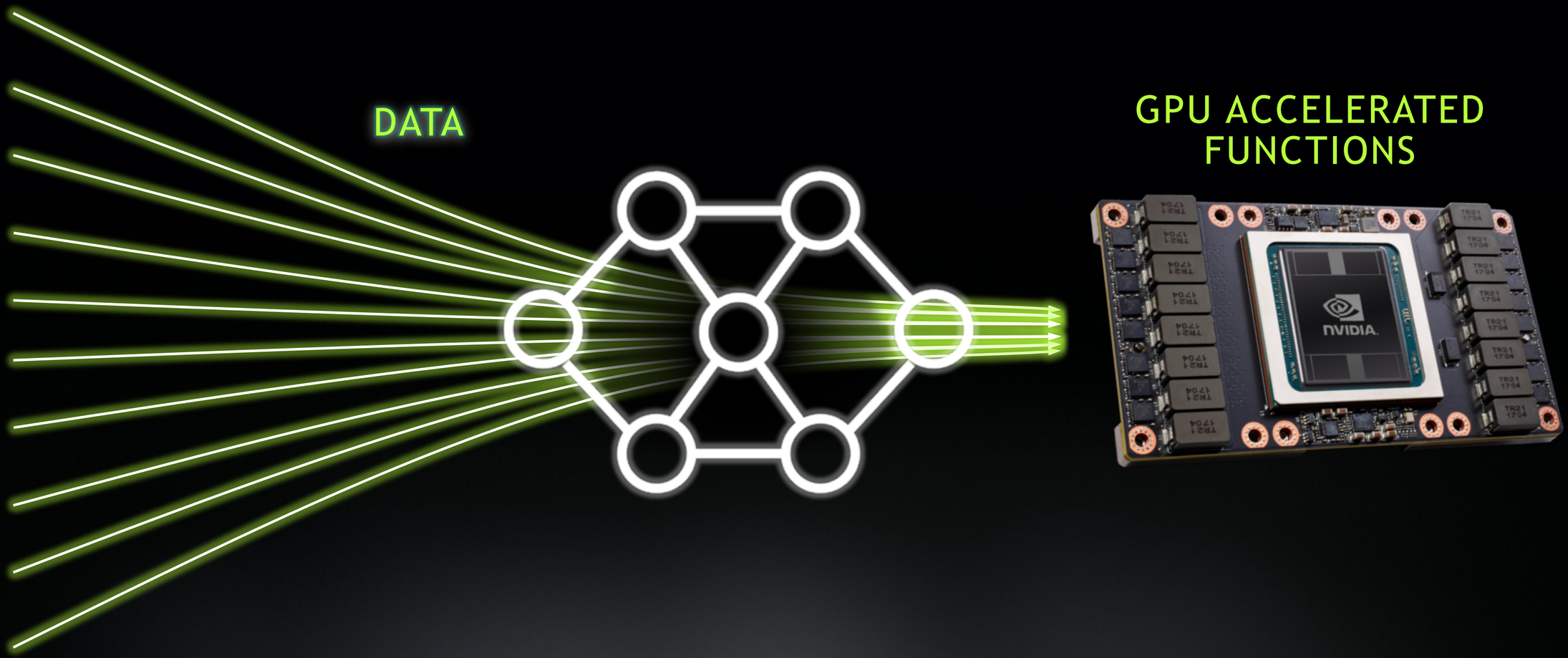
LEARNED FUNCTION

```
Function1(T,P,Q)
A = relu( w1 * [T,P,Q] + b1)
B = relu( w2 * A       + b2)
C = relu( w3 * B       + b3)
D = relu( w4 * C       + b4)
E = relu( w5 * D       + b5)
y = sigmoid(w6 * E     + b6)
return y
```

Reverse-engineer a function from inputs / outputs

LEARNED FUNCTIONS ARE GPU ACCELERATED

Next level software. No porting required.

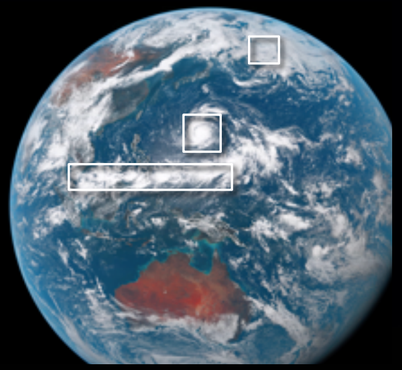


USE IT TO ENHANCE EXISTING APPLICATIONS

Improve all stages of numerical weather prediction

COLLECTION THINNING ASSIMILATION DYNAMICS PARAMETRIZATION COMMUNICATION

USE IT TO BUILD NEW CAPABILITIES



WEATHER DETECTION

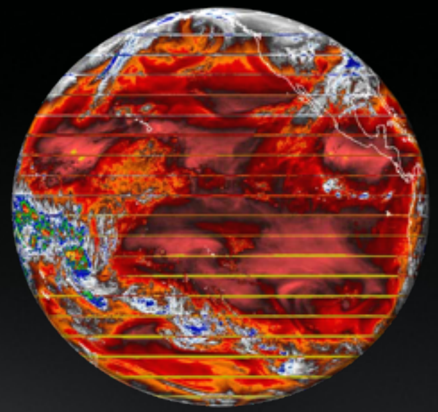
ENVIRONMENTAL MONITORING

DISASTER PLANNING

NEAR-EARTH OBJECT DETECTION

DATA FUSION

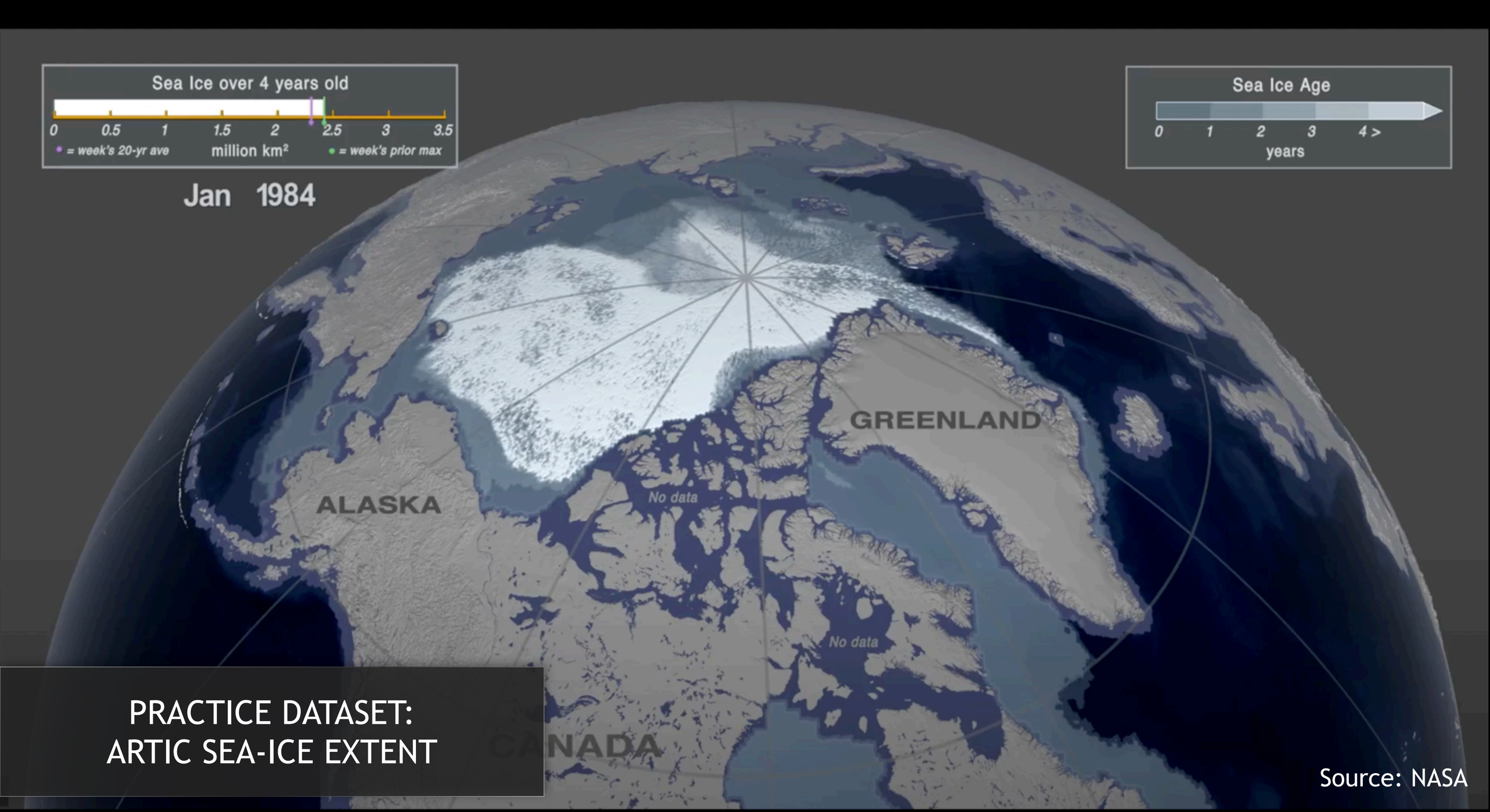
AUTONOMOUS SENSORS



DATA ENHANCEMENT

DATA DRIVEN MODELS

PART 1: CURVE FITTING WITH PYTORCH

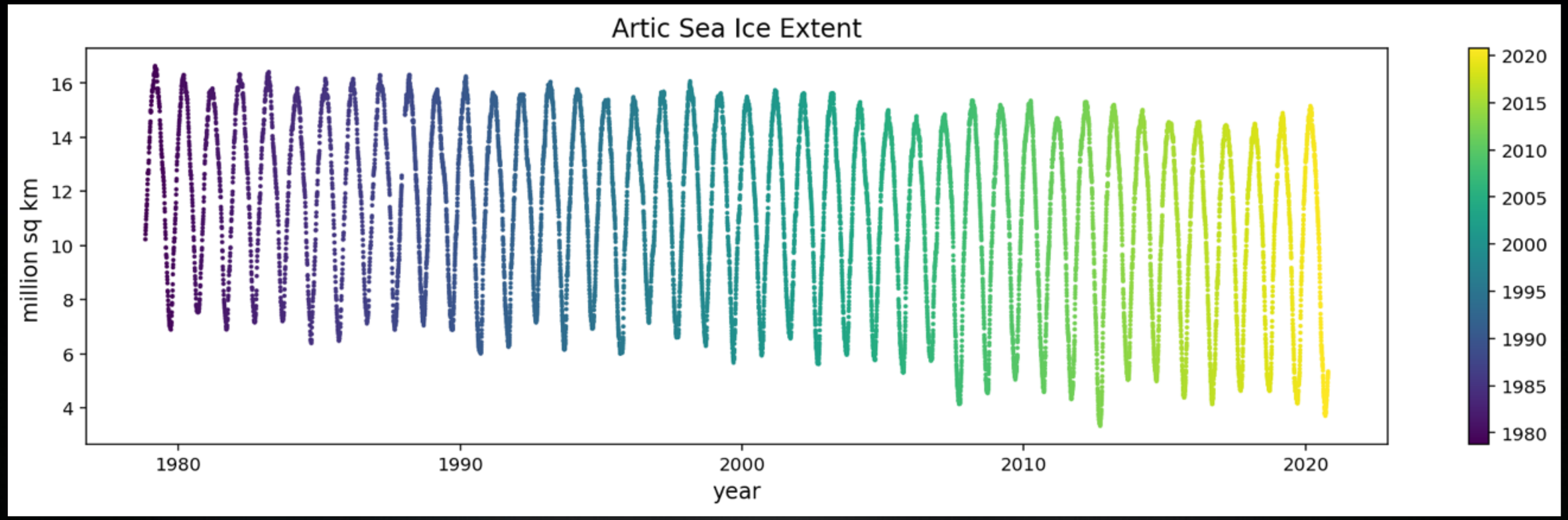


PRACTICE DATASET: ARTIC SEA-ICE EXTENT

Source: NASA

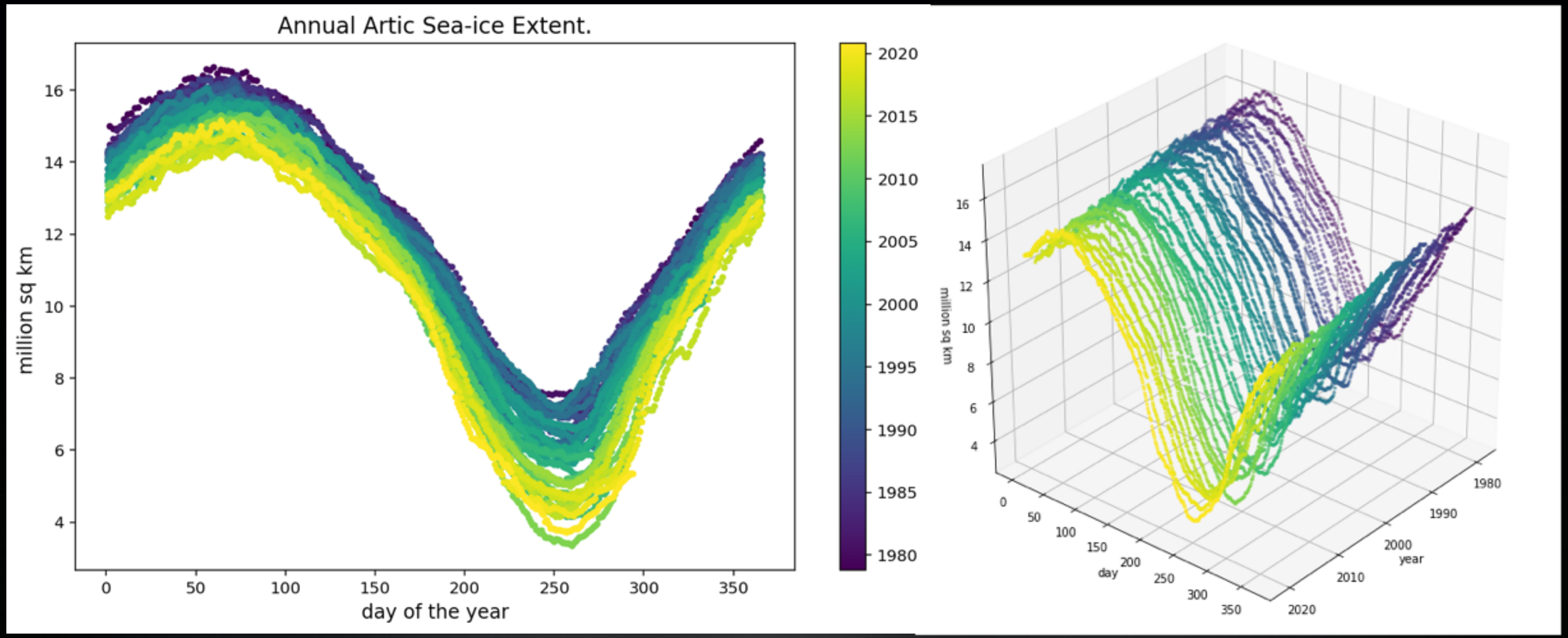
SEA ICE EXTENT VS TIME

An interesting data set to practice curve-fitting



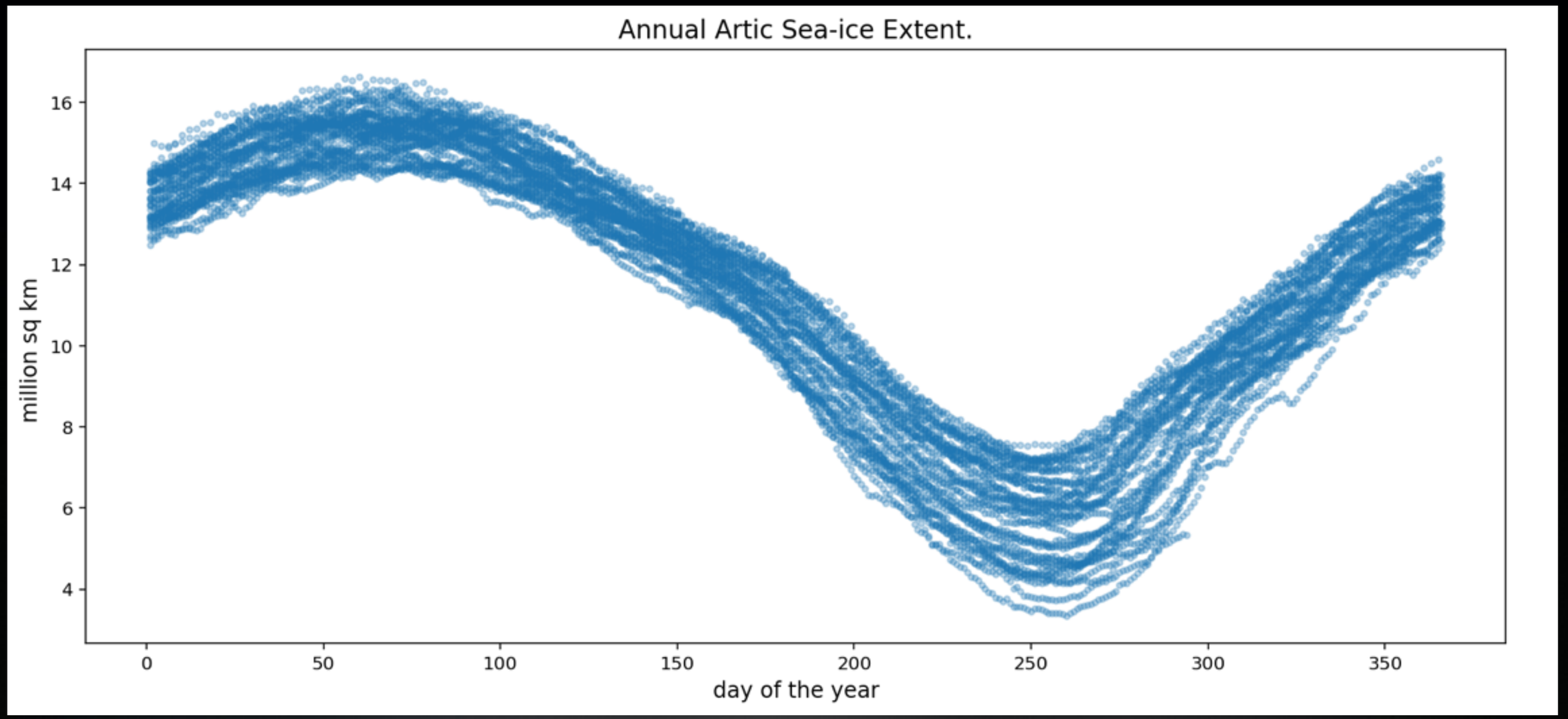
ANNUAL SEA ICE EXTENT 2D

Find function: extent = f(day of year, year)



ANNUAL SEA ICE EXTENT 1D

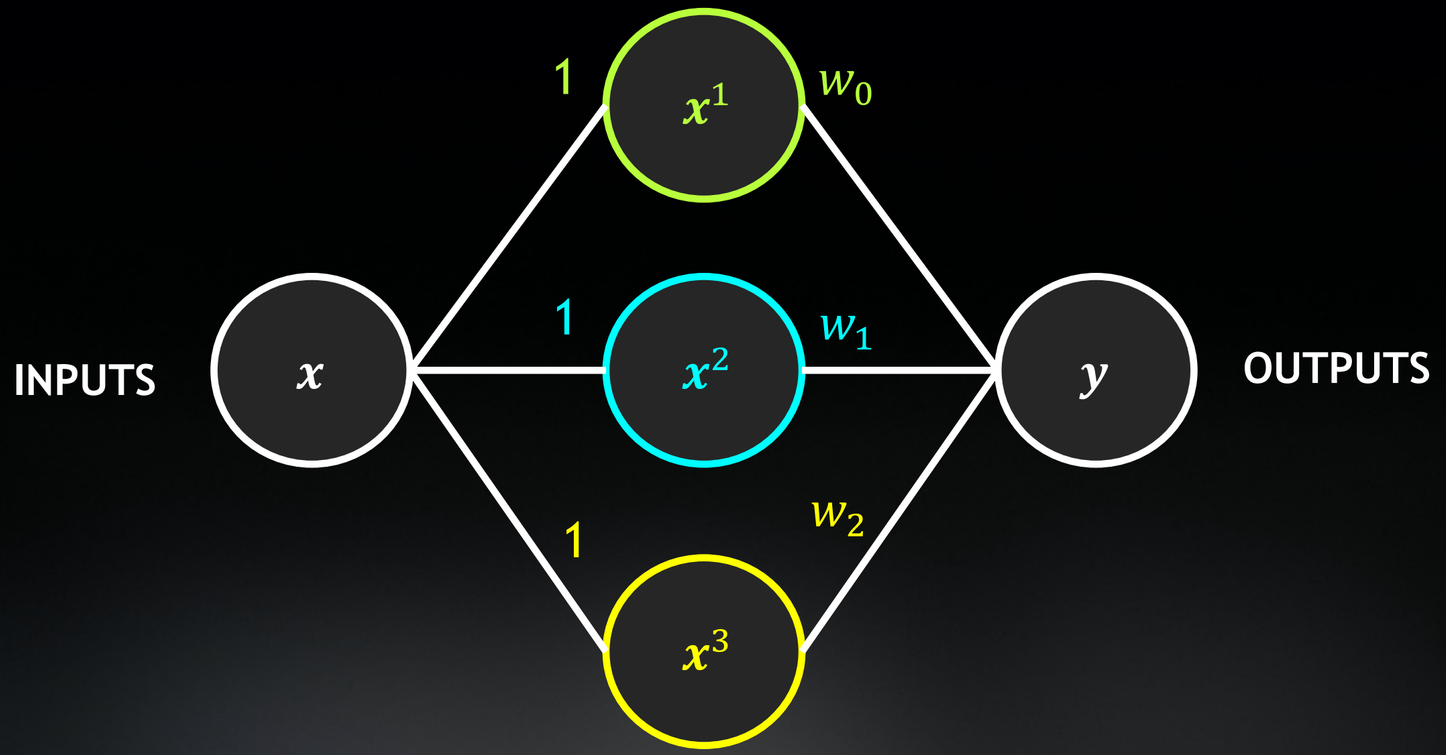
Find function: extent = f(day of the year)



TAYLOR SERIES

Function approximation using polynomial basis vectors

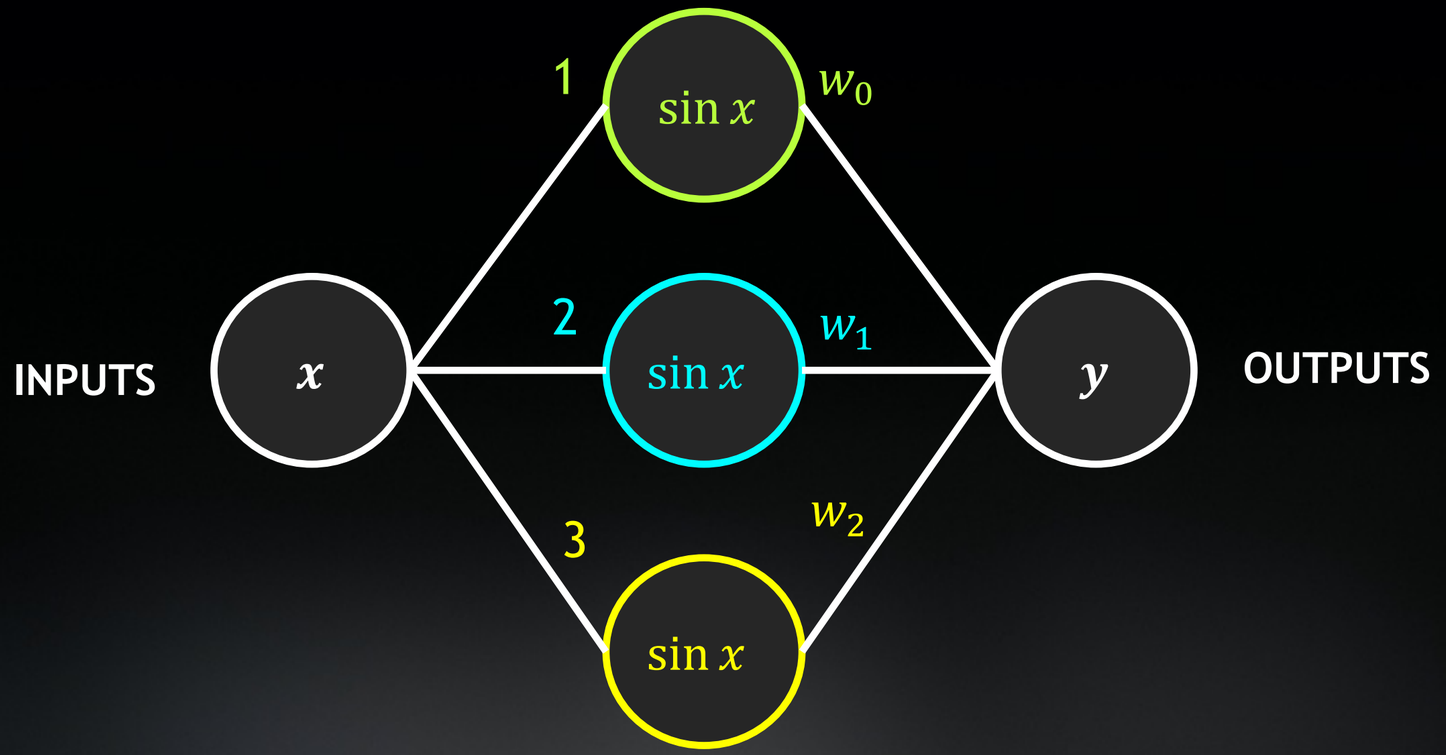
$$y = b_0 + w_0 x^1 + w_1 x^2 + w_2 x^3$$



FOURIER SERIES

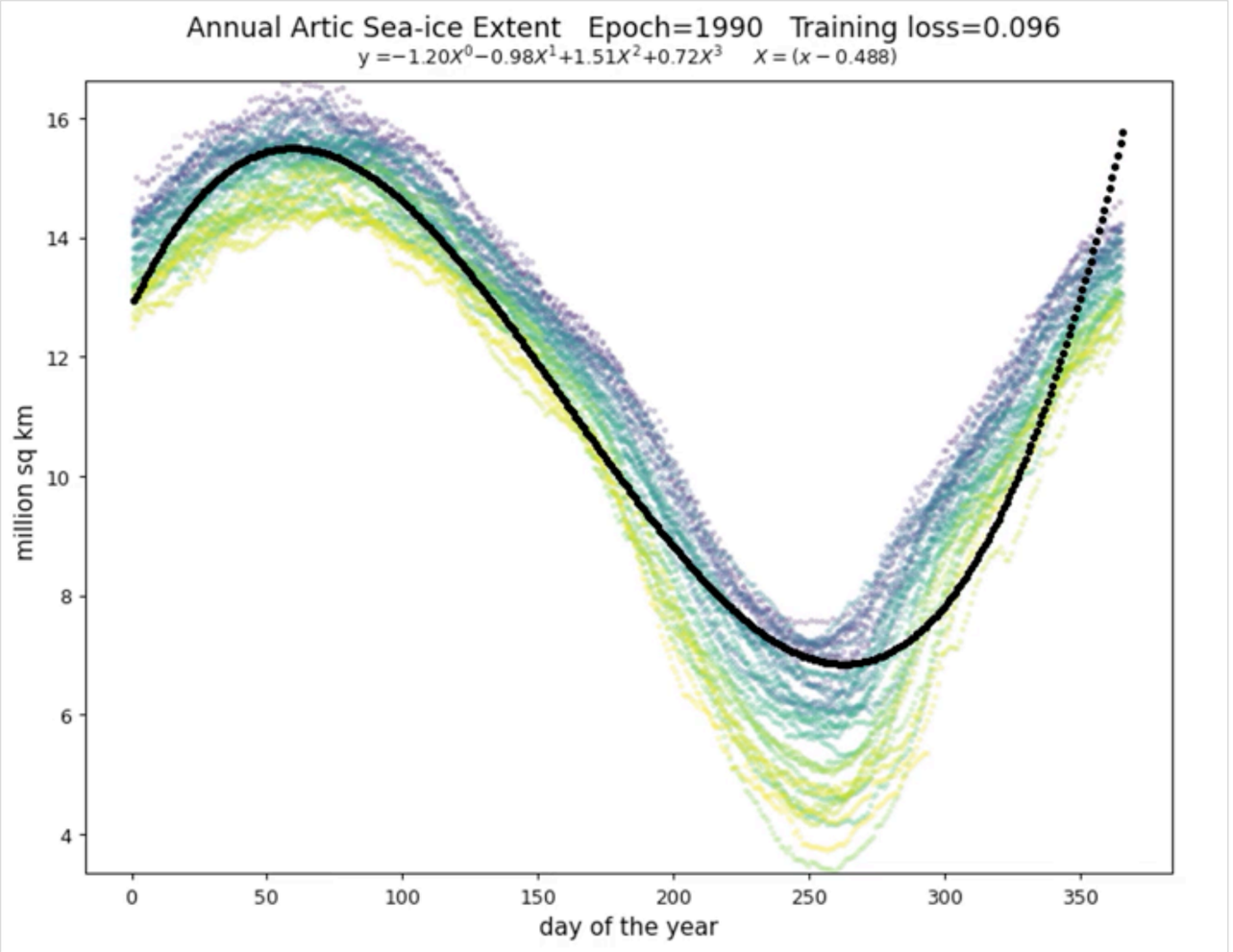
Function approximation using sinusoidal basis vectors

$$y = b_0 + w_0 \sin(x) + w_1 \sin(2x) + w_2 \sin(3x)$$



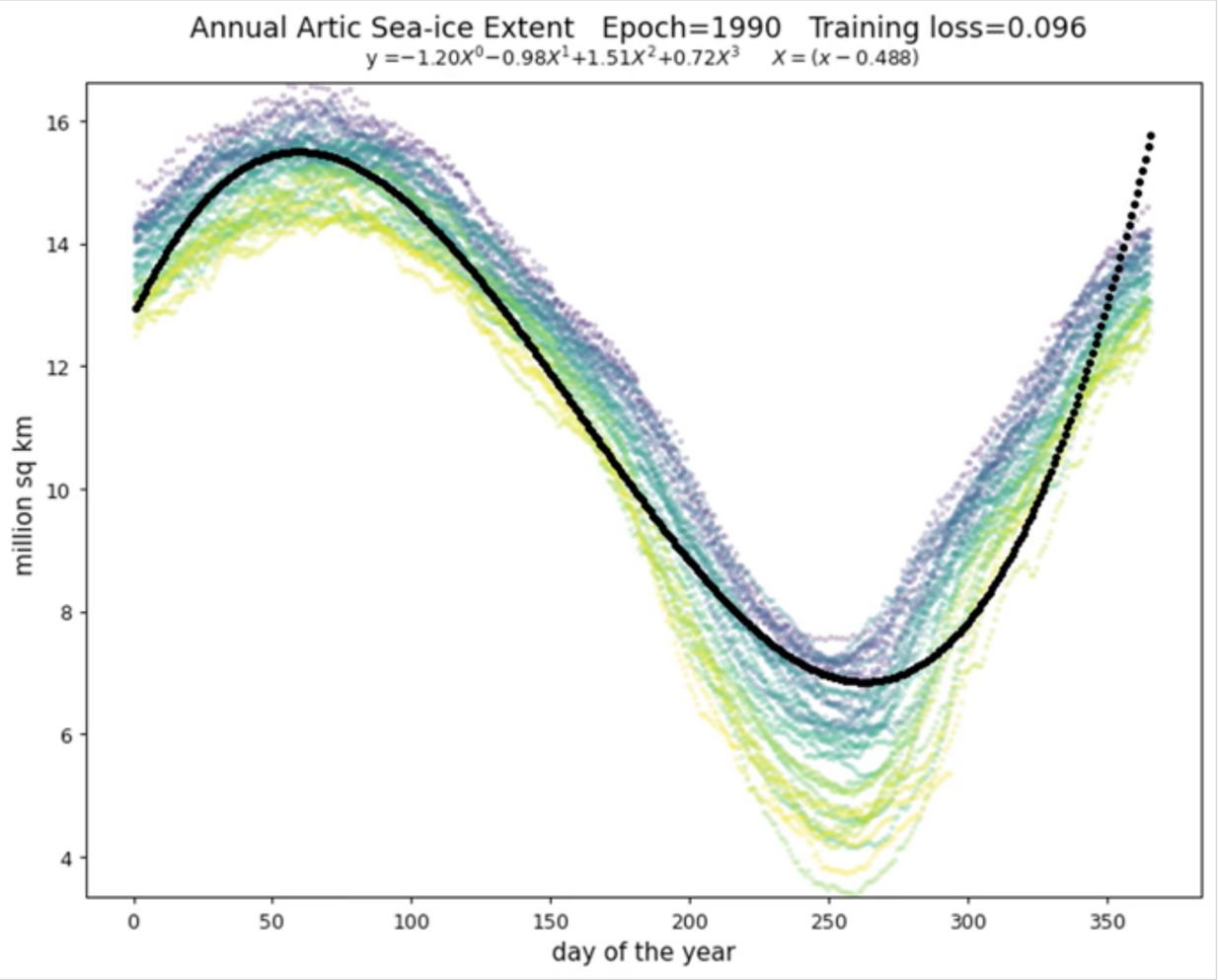
CURVE FIT

```python
import torch
torch.manual_seed(0)

# DATA
x,y = load_data()

# MODEL
N  = 3
x0 = torch.zeros(1,   requires_grad=True)
w  = torch.zeros(N+1, requires_grad=True)
def model(x): return sum( w[i]*(x-x0)**i for i in range(N+1) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [x0,w], lr=2e-3)
loss_fcn  = torch.nn.MSELoss()

for epoch in range(2000+1):

  # TRAIN
  prediction = model(x)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y)
  loss.backward()
  optimizer.step()

  with torch.no_grad(): plot(interval=100)
```



CURVE FIT

```python
import torch
torch.manual_seed(0)

# DATA
x,y = load_data()

# MODEL
N  = 3
x0 = torch.zeros(1,   requires_grad=True)
w  = torch.zeros(N+1, requires_grad=True)
def model(x): return sum( w[i]*(x-x0)**i for i in range(N+1) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [x0,w], lr=2e-3)
loss_fcn  = torch.nn.MSELoss()

for epoch in range(2000+1):

  # TRAIN
  prediction = model(x)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y)
  loss.backward()
  optimizer.step()

  with torch.no_grad(): plot(interval=100)
```



ACTIVITY: IMPROVE THE CURVE FIT

[Click here to open notebook]

```
1  import torch, numpy as np
2  torch.manual_see
3
4  # DATA
5  x,y = load_data(
6
7  # MODEL
8  N  = 3
9  x0 = torch.zeros
10 w  = torch.zeros
11 def model(x): re
12
13 # CONFIGURE
14 optimizer = torc
15 loss_fcn  = toro
16
17 for epoch in ran
18
19   # TRAIN
20   prediction = m
21   optimizer.zero
22   loss = loss_fc
23   loss.backward(
24   optimizer.step(
25
26   with torch.no_grad(): plot(interval=100)
```

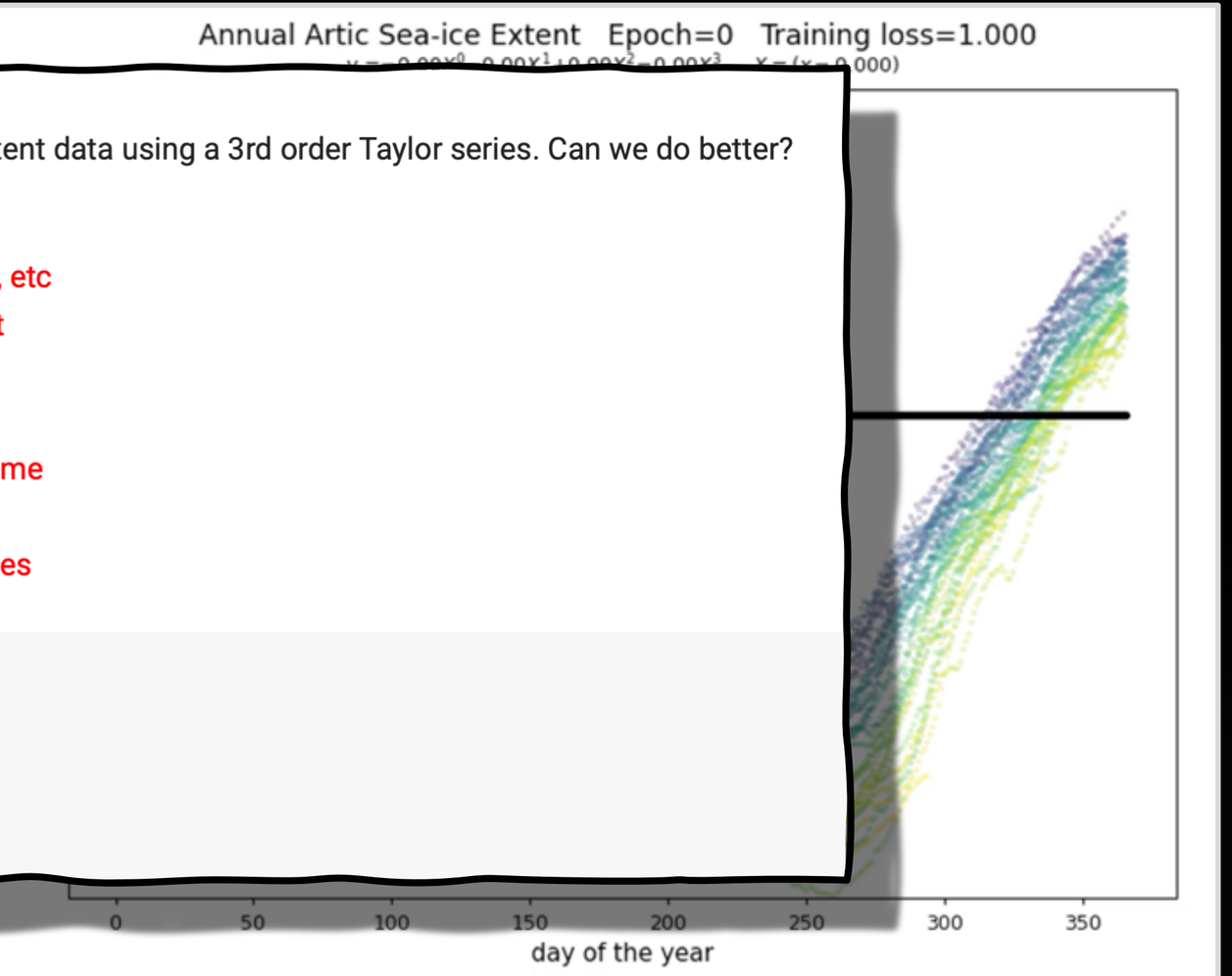
Train the Model

Below, we have approximated the seasonal sea-ice extent data using a 3rd order Taylor series. Can we do better?

TODO:

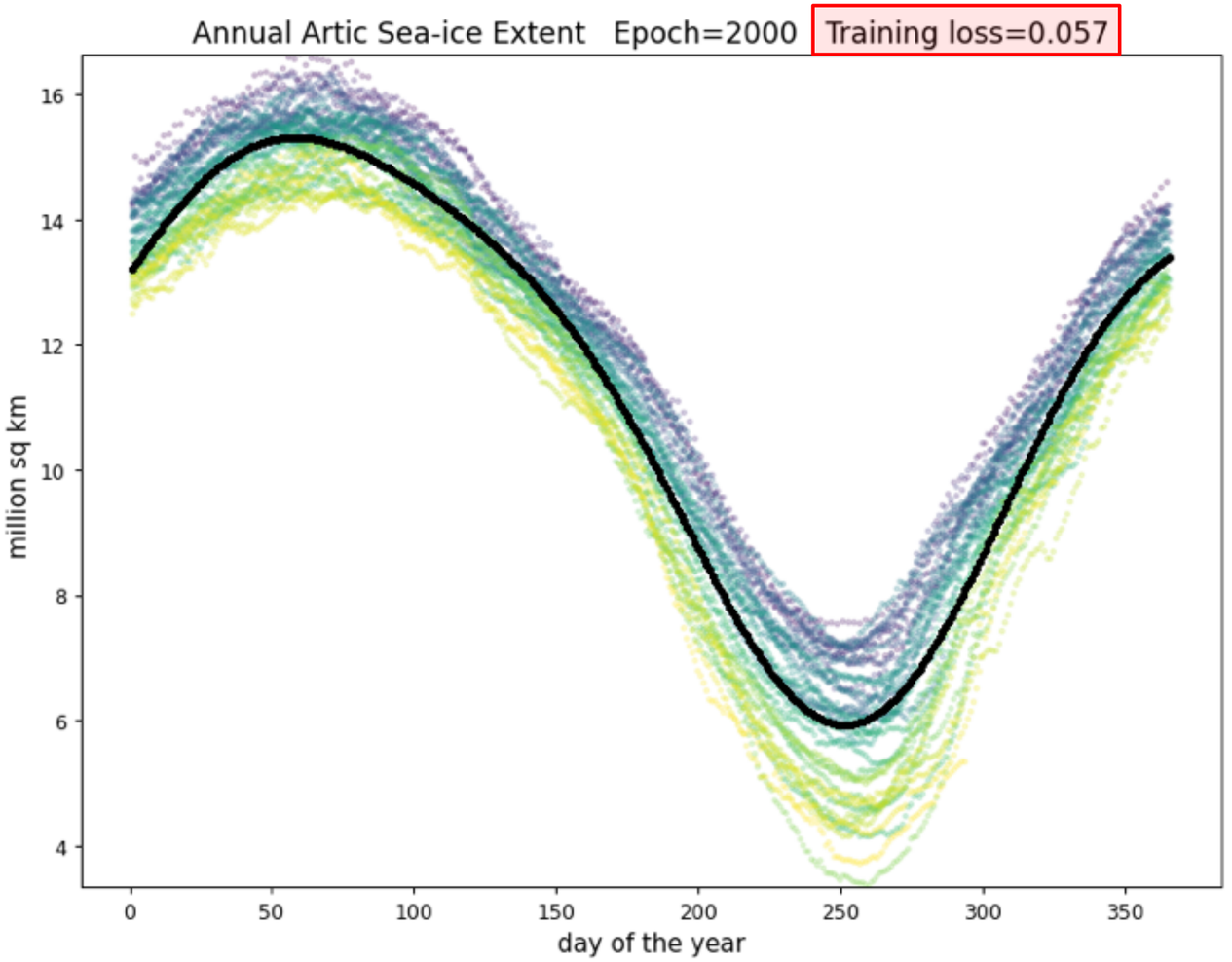
- Try changing the order of the model to N=2, N=5, etc
- Add more epochs if neccessary to get a better fit
- Adjust the learning rate (lr=...)
- Try replacing MSELoss with L1Loss
- Increase the plotting interval to reduce training time
- See how small you can get the training loss
- Try replacing the Taylor series with a Fourier series

```
[ ]  1  import torch
     2  torch.manual_seed(0)
     3
     4  # DATA
     5  x,y = load_data()
     6
```



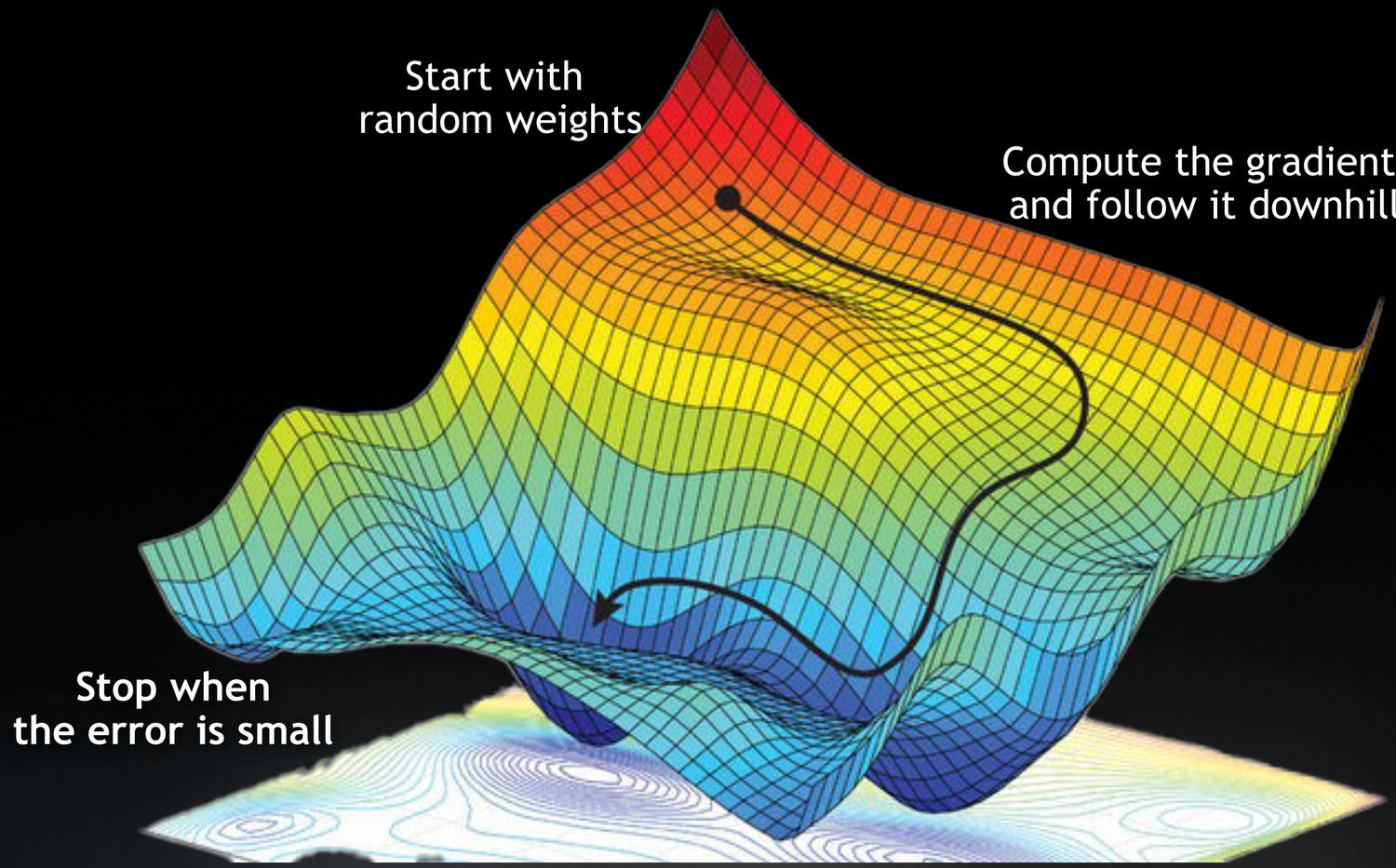
FOURIER SERIES SOLUTION

```python
import torch
torch.manual_seed(0)

# DATA
x,y = load_data()

# MODEL
N      = 3
x0     = torch.randn(N, requires_grad=True)
freq   = torch.randn(N, requires_grad=True)
w      = torch.zeros(N, requires_grad=True)
def model(x):
  return sum(w[i]*torch.sin(freq[i]*np.pi*(x-x0[i])) for i in range(N) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [a,x0,w], lr=2e-3)
loss_fcn  = torch.nn.MSELoss()

for epoch in range(2000+1):

  # TRAIN
  prediction = model(x)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y)
  loss.backward()
  optimizer.step()

  with torch.no_grad(): plot(interval=100)
```



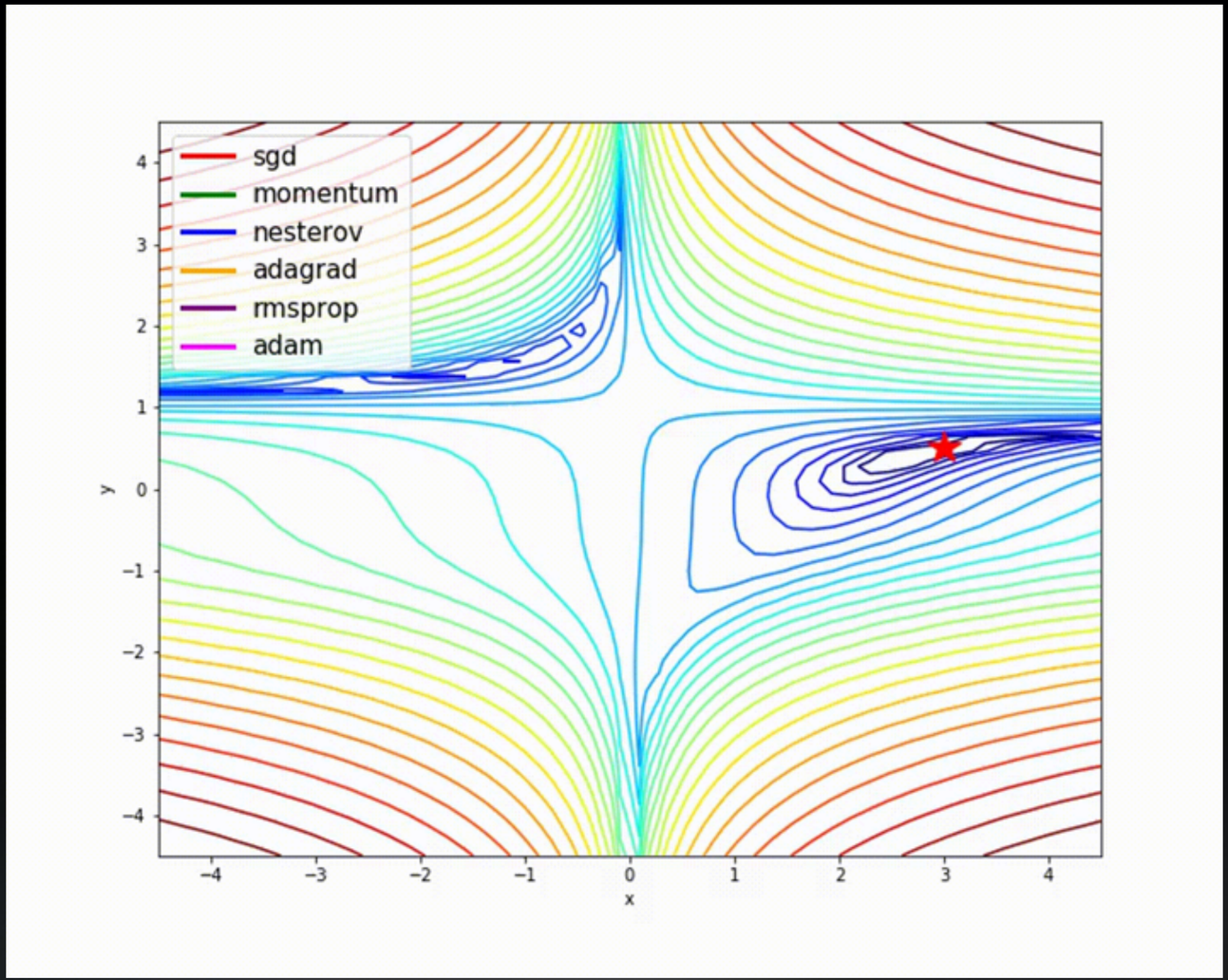
LOSS FUNCTION

Error surface, measuring how well prediction matches targets



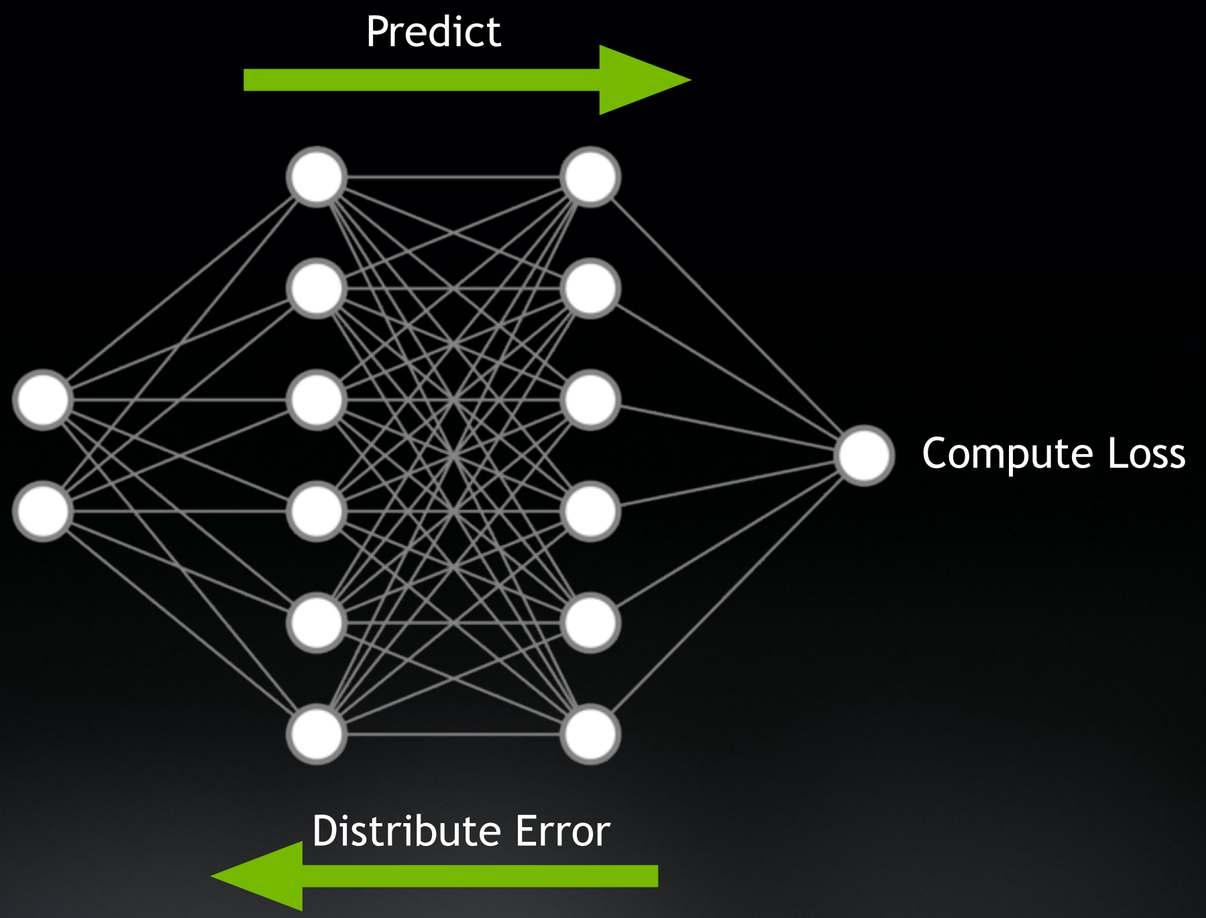
OPTIMIZER

Strategy for adjusting model weights, to minimize loss



BACKPROPAGATION

Compute gradient of loss function with respect to each parameter

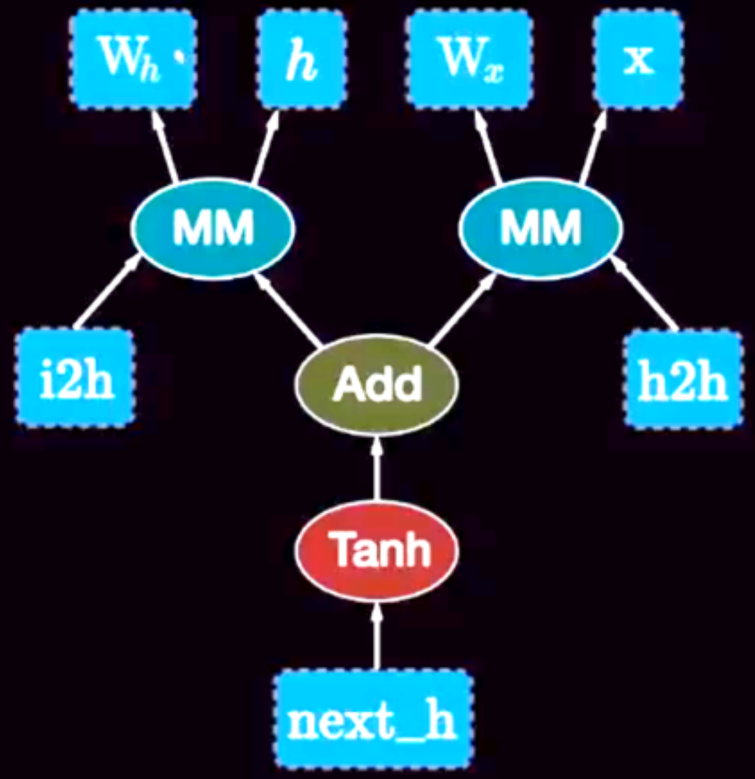


AUTOGRAD

Let a framework keep track of your gradient, so you don't have to

PyTorch Autograd

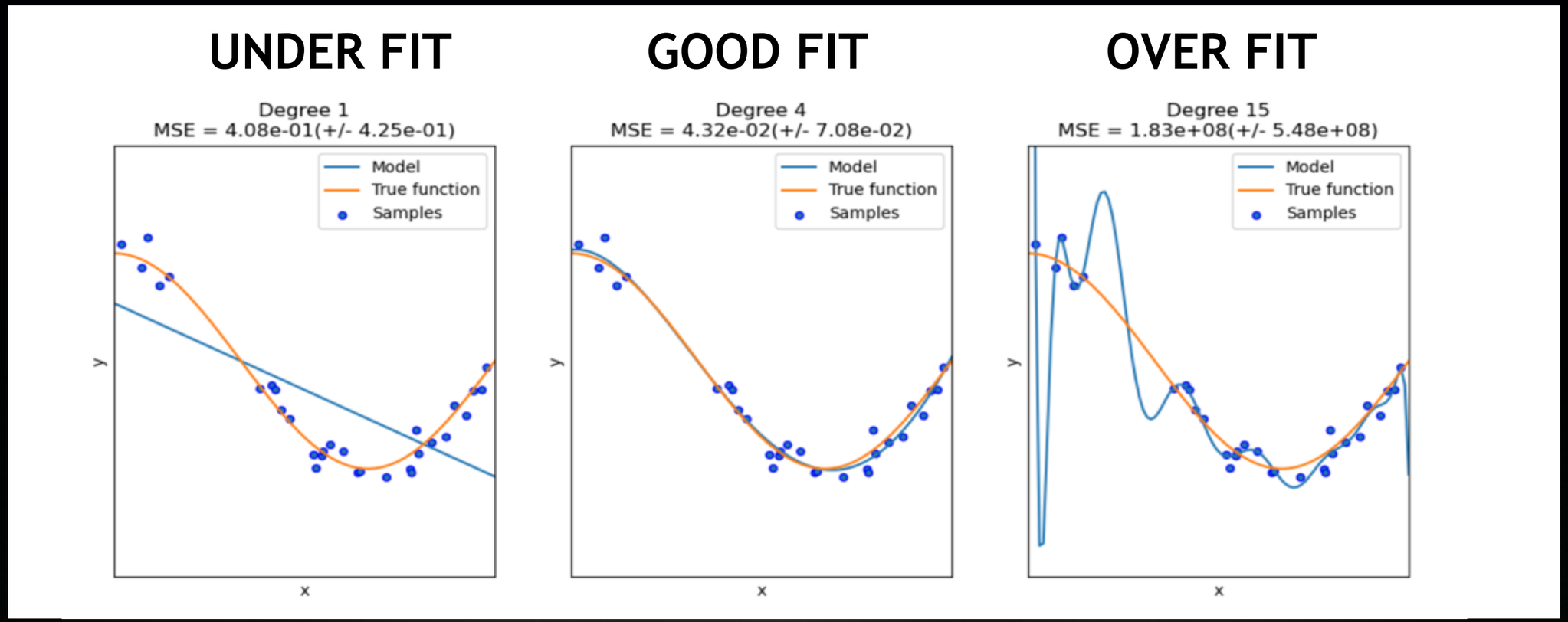
```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```



GENERALIZATION

UNDERFITTING AND OVERFITTING

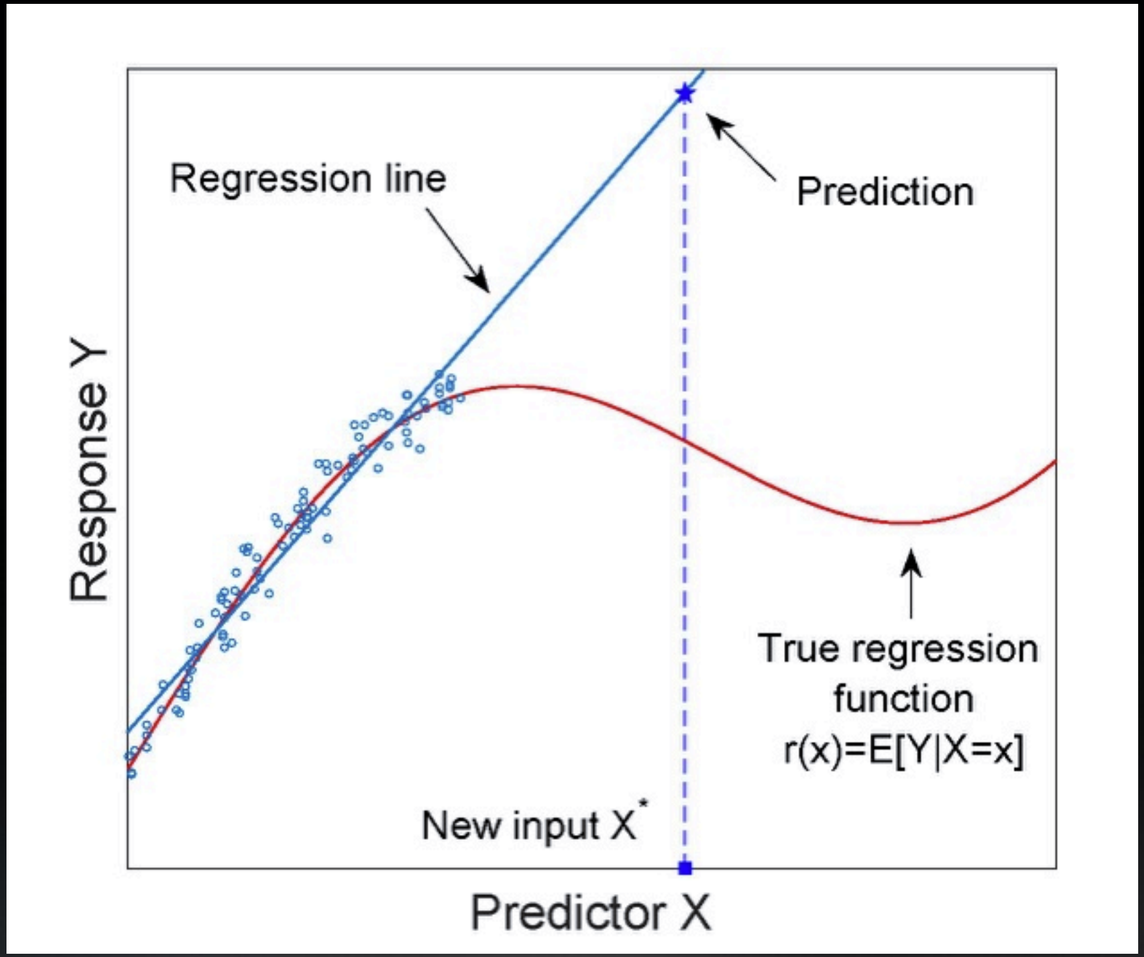
A good model is one that generalizes to new data



https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

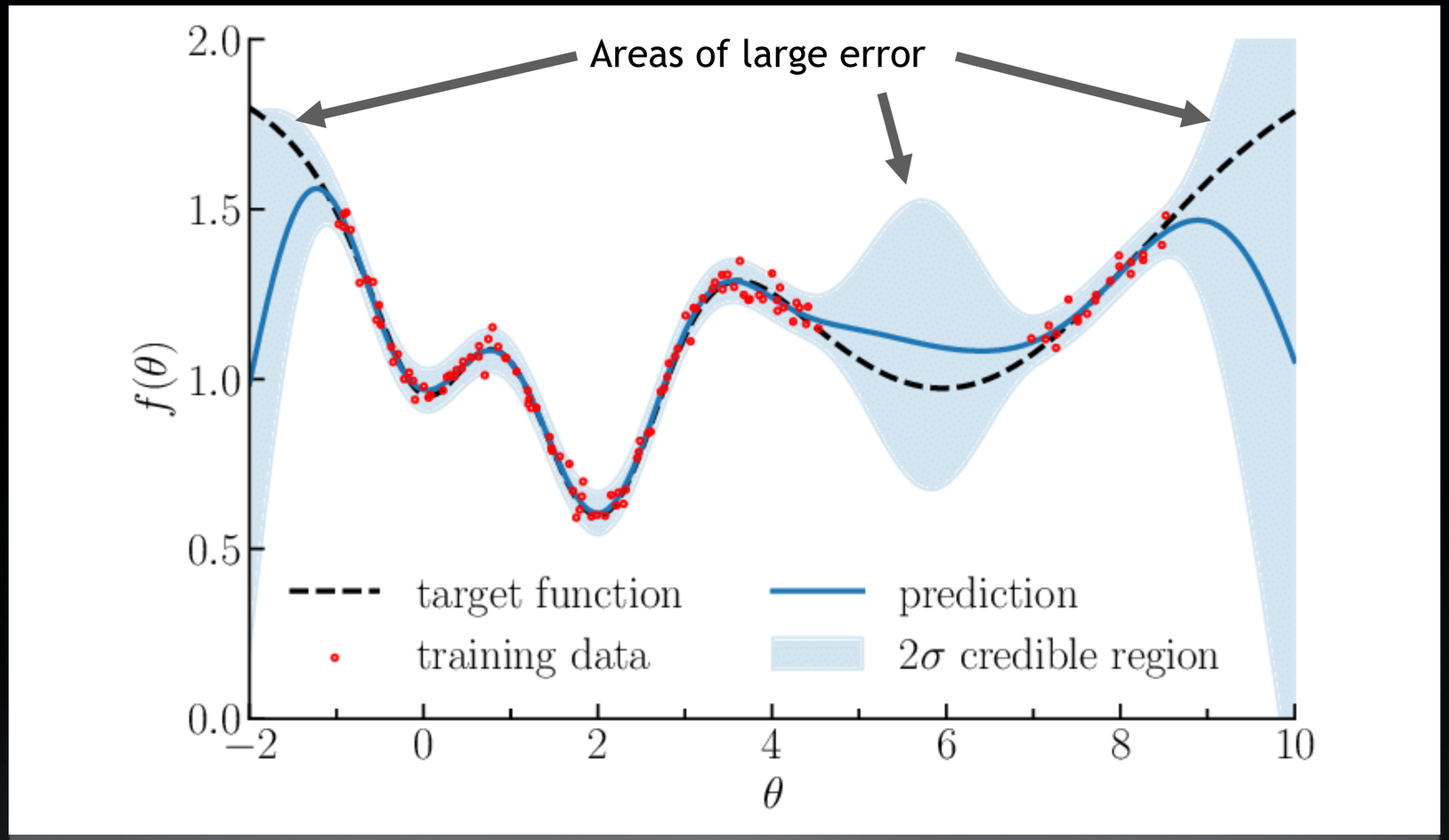
EPISTEMIC ERROR

Error can be large, far from training data



EPISTEMIC ERROR

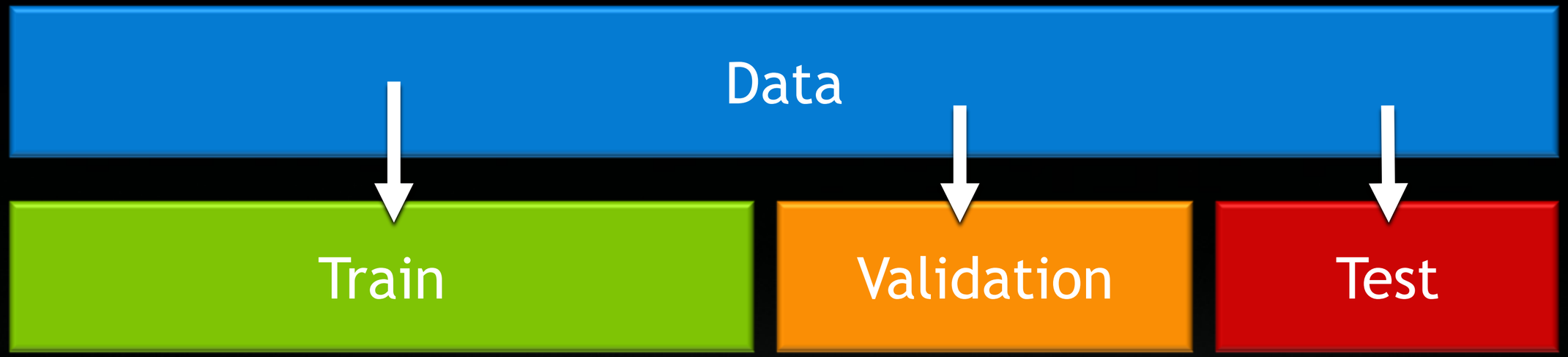
Error grows, in regions far from our training data



https://www.researchgate.net/figure/Illustration-of-Gaussian-process-regression-in-one-dimension-for-the-target-test_fig1_327613136

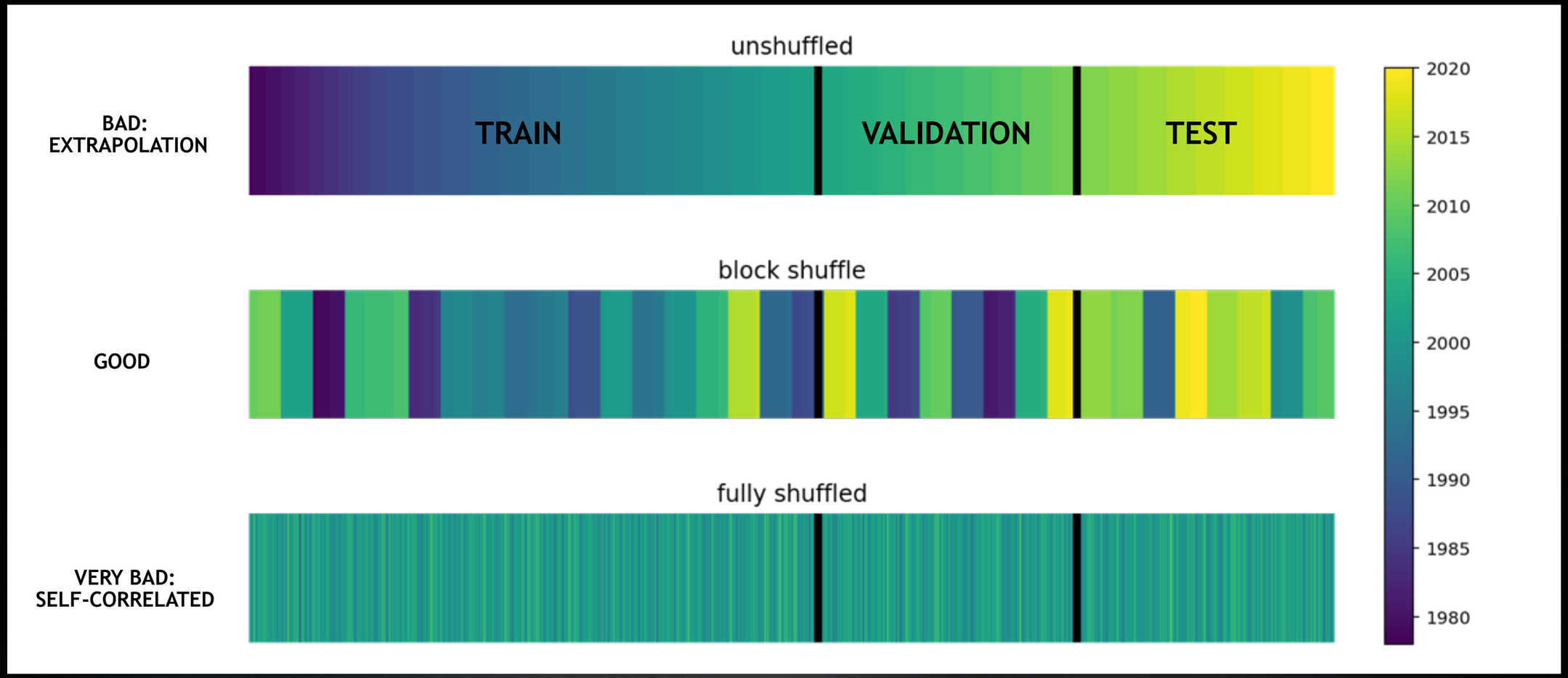
DATA SPLITTING

Split your examples into train, validation, and test sets



SPLITTING TIME SERIES DATA

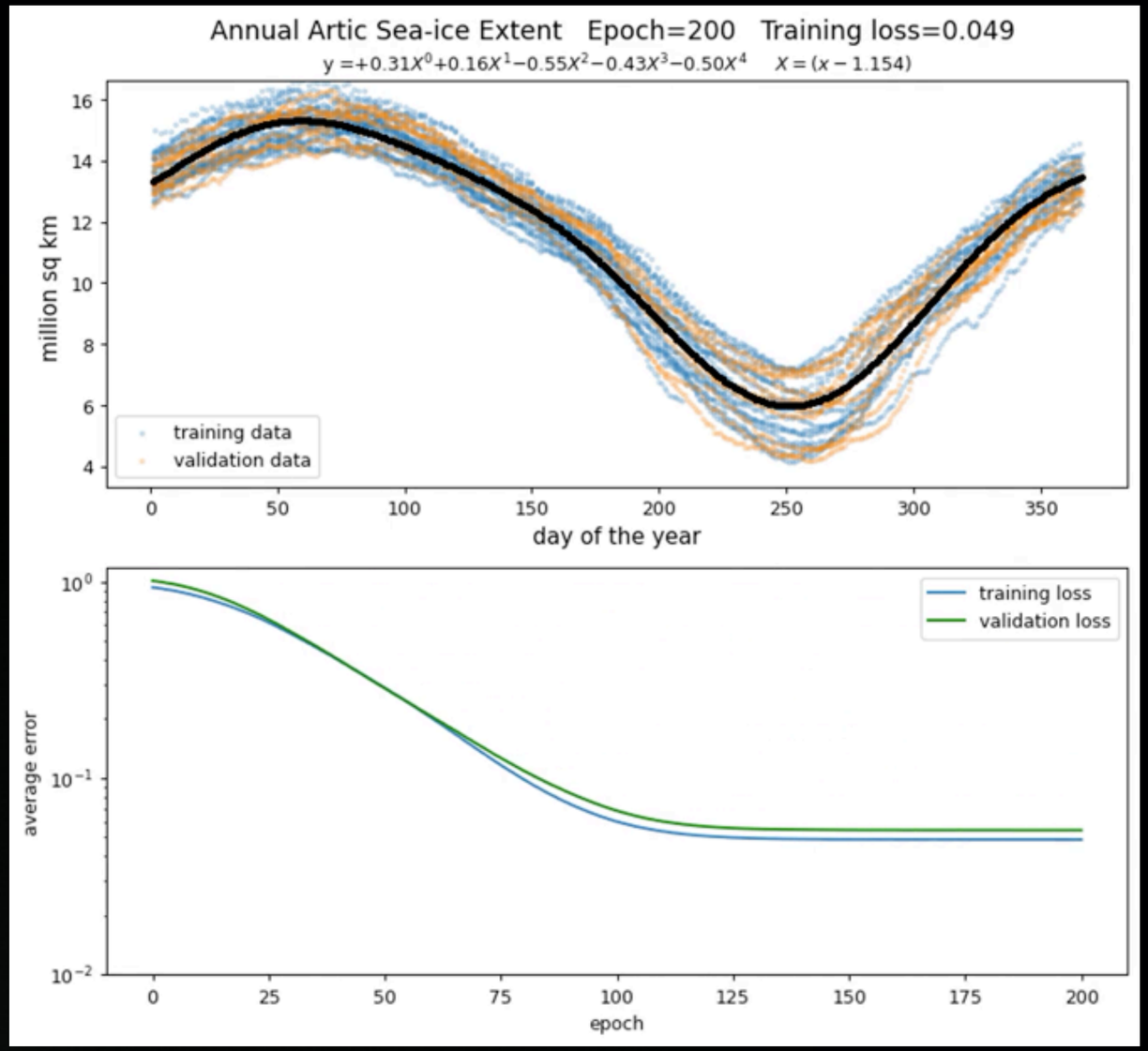
Avoid correlations and extrapolation



```python
import torch, numpy as np
torch.manual_seed(0)

# DATA
x,y = load_data()
shuffle_blocks(blocksize=200)
ntrain, nval, ntest = (len(x)*torch.tensor([0.50, 0.25, 0.25])).int()
x_train, x_val, xtest = x[:ntrain], x[ntrain:-ntest], x[-ntest:]
y_train, y_val, ytest = y[:ntrain], y[ntrain:-ntest], y[-ntest:]

# MODEL
N     = 5
x0    = torch.randn(N, requires_grad=True)
freq  = torch.randn(N, requires_grad=True)
w     = torch.zeros(N, requires_grad=True)
def model(x):
  return sum(w[i]*torch.sin(freq[i]*np.pi*(x-x0[i])) for i in range(N) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [freq,x0,w], lr=5e-3)
loss_fcn  = torch.nn.MSELoss()

train_loss, val_loss=[],[]
for epoch in range(200+1):

  # TRAIN
  prediction = model(x_train)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y_train)
  loss.backward()
  optimizer.step()
  train_loss.append(loss.item())

  # VALIDATE
  with torch.no_grad():

    p_val = model(x_val)
    loss_val = loss_fcn(p_val, y_val)
    val_loss.append(loss_val.item())
    plot(interval=10)
```

VALIDATION

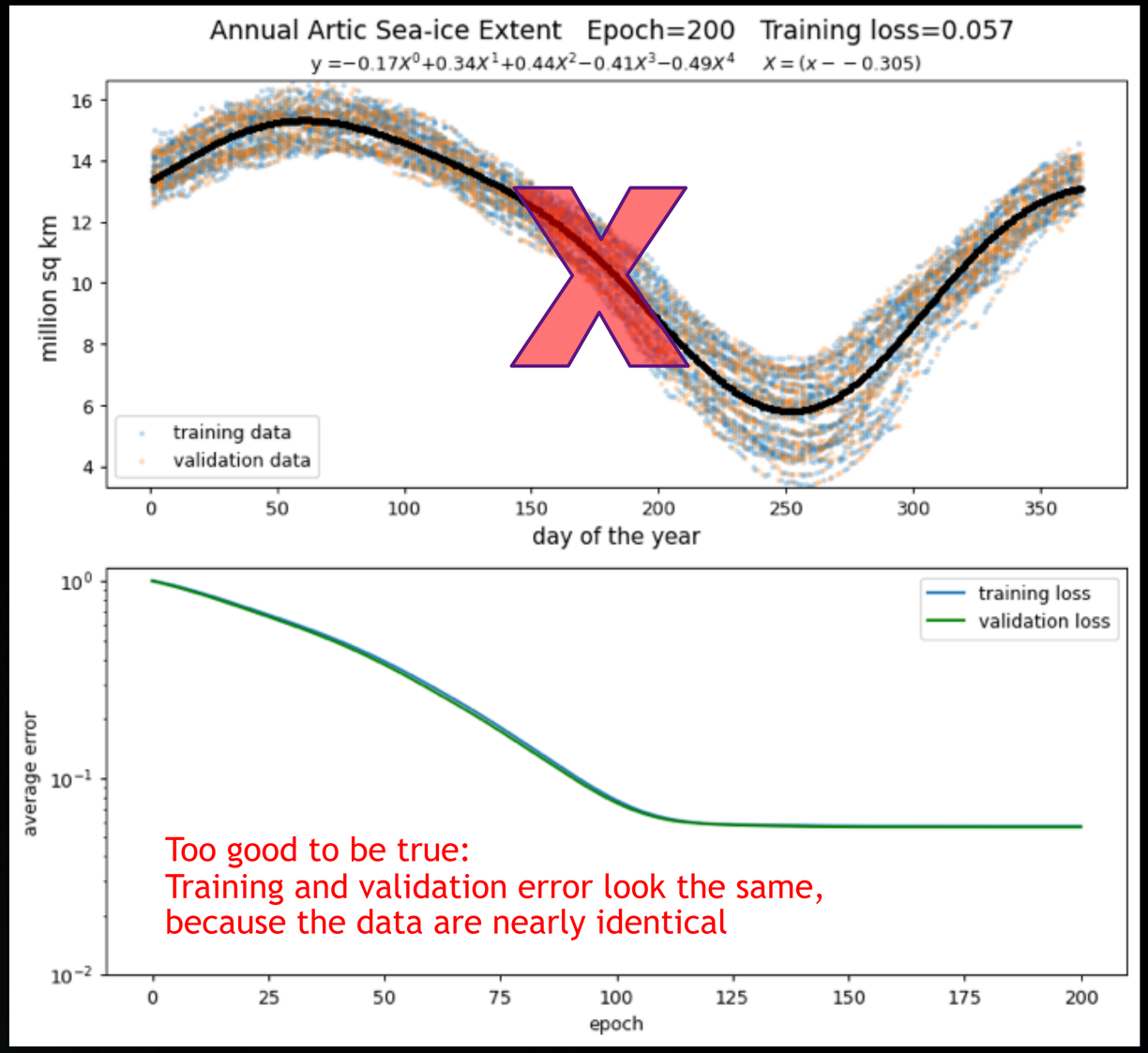
Block shuffled



```python
import torch, numpy as np
torch.manual_seed(0)

# DATA
x,y = load_data()
shuffle_blocks(blocksize=1)
ntrain, nval, ntest = (len(x)*torch.tensor([0.50, 0.25, 0.25])).int()
x_train, x_val, xtest = x[:ntrain], x[ntrain:-ntest], x[-ntest:]
y_train, y_val, ytest = y[:ntrain], y[ntrain:-ntest], y[-ntest:]

# MODEL
N      = 5
x0     = torch.randn(N, requires_grad=True)
freq   = torch.randn(N, requires_grad=True)
w      = torch.zeros(N, requires_grad=True)
def model(x):
  return sum(w[i]*torch.sin(freq[i]*np.pi*(x-x0[i])) for i in range(N) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [freq,x0,w], lr=5e-3)
loss_fcn  = torch.nn.MSELoss()

train_loss, val_loss=[],[]
for epoch in range(200+1):

  # TRAIN
  prediction = model(x_train)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y_train)
  loss.backward()
  optimizer.step()
  train_loss.append(loss.item())

  # VALIDATE
  with torch.no_grad():

    p_val = model(x_val)
    loss_val = loss_fcn(p_val, y_val)
    val_loss.append(loss_val.item())
    plot(interval=10)
```

VALIDATION

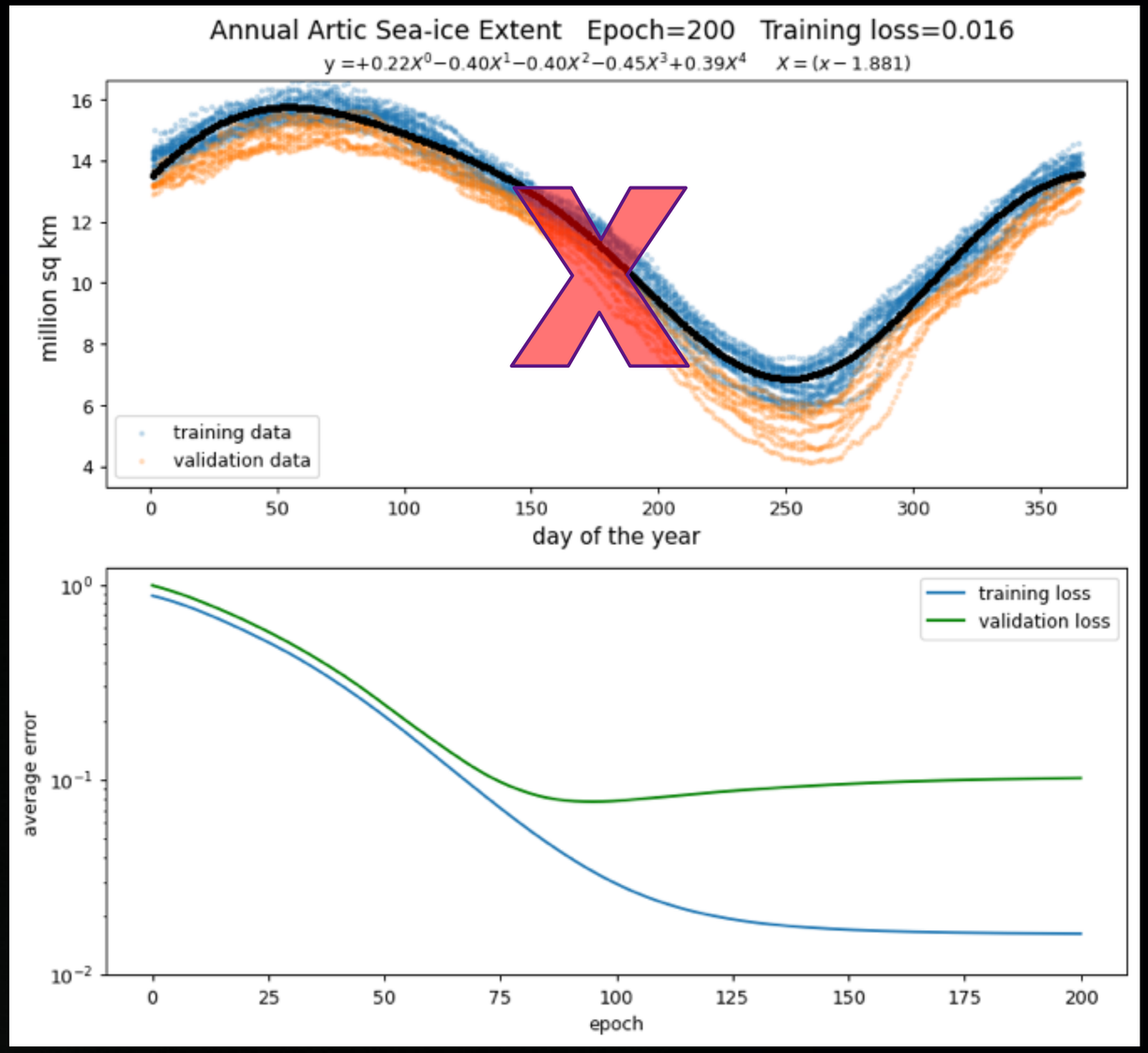
Fully shuffled: self-correlated



```python
import torch, numpy as np
torch.manual_seed(0)

# DATA
x,y = load_data()

ntrain, nval, ntest = (len(x)*torch.tensor([0.50, 0.25, 0.25])).int()
x_train, x_val, xtest = x[:ntrain], x[ntrain:-ntest], x[-ntest:]
y_train, y_val, ytest = y[:ntrain], y[ntrain:-ntest], y[-ntest:]

# MODEL
N      = 5
x0     = torch.randn(N, requires_grad=True)
freq   = torch.randn(N, requires_grad=True)
w      = torch.zeros(N, requires_grad=True)
def model(x):
  return sum(w[i]*torch.sin(freq[i]*np.pi*(x-x0[i])) for i in range(N) )

# CONFIGURE
optimizer = torch.optim.AdamW(params = [freq,x0,w], lr=5e-3)
loss_fcn  = torch.nn.MSELoss()

train_loss, val_loss=[],[]
for epoch in range(200+1):

  # TRAIN
  prediction = model(x_train)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y_train)
  loss.backward()
  optimizer.step()
  train_loss.append(loss.item())

  # VALIDATE
  with torch.no_grad():

    p_val = model(x_val)
    loss_val = loss_fcn(p_val, y_val)
    val_loss.append(loss_val.item())
    plot(interval=10)
```

VALIDATION

Sequential: future = extrapolation

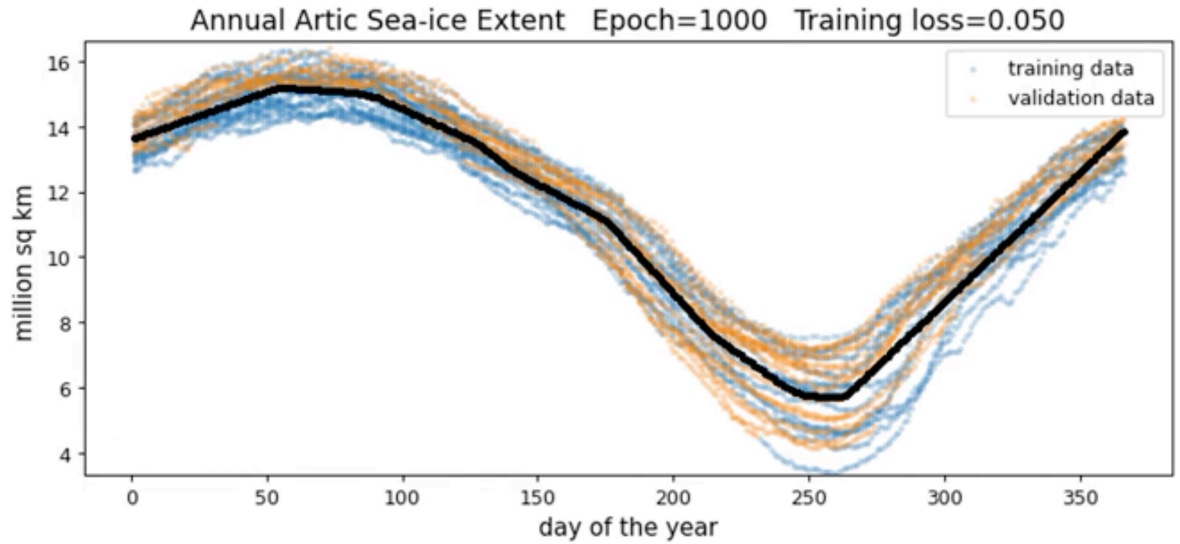


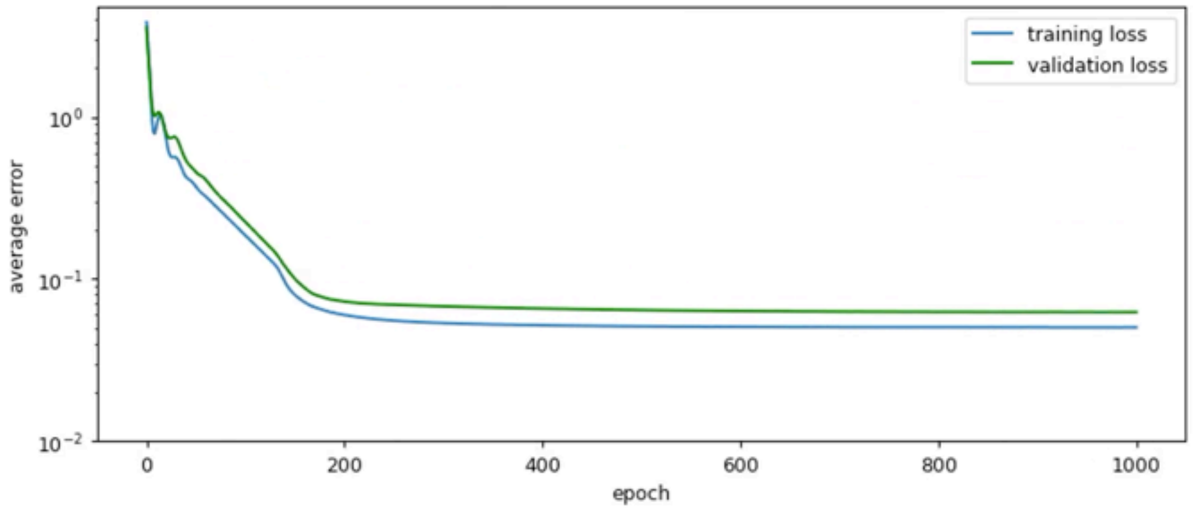
FULLY CONNECTED NETWORKS

SHALLOW NEURAL NET 1D

ReLU basis function expansion

```python
import torch, numpy as np
torch.manual_seed(0)

# DATA
x,y = load_data()
shuffle_blocks(blocksize=100)
ntrain, nval, ntest = (len(x)*torch.tensor([0.50, 0.25, 0.25])).int()
x_train, x_val, xtest = x[:ntrain], x[ntrain:-ntest], x[-ntest:]
y_train, y_val, ytest = y[:ntrain], y[ntrain:-ntest], y[-ntest:]

# MODEL
def relu(x): return x*(x>0)+ 0.1*x*(x<0)
N = 20
b = torch.randn(N+1, requires_grad=True)
w1= torch.randn(N,   requires_grad=True)
w2= torch.randn(N,   requires_grad=True)
def model(x): return sum(w2[i]*relu(w1[i]*x+b[i]) for i in range(N))+b[N]

# CONFIGURE
optimizer = torch.optim.AdamW(params = [b, w1, w2], lr=1e-2)
loss_fcn  = torch.nn.MSELoss()

train_loss, val_loss=[],[]
for epoch in range(1000+1):

  # TRAIN
  prediction = model(x_train)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y_train)
  loss.backward()
  optimizer.step()
  train_loss.append(loss.item())

  # VALIDATE
  with torch.no_grad():

    p_val = model(x_val)
    loss_val = loss_fcn(p_val, y_val)
    val_loss.append(loss_val.item())
    plot(interval=20)
```

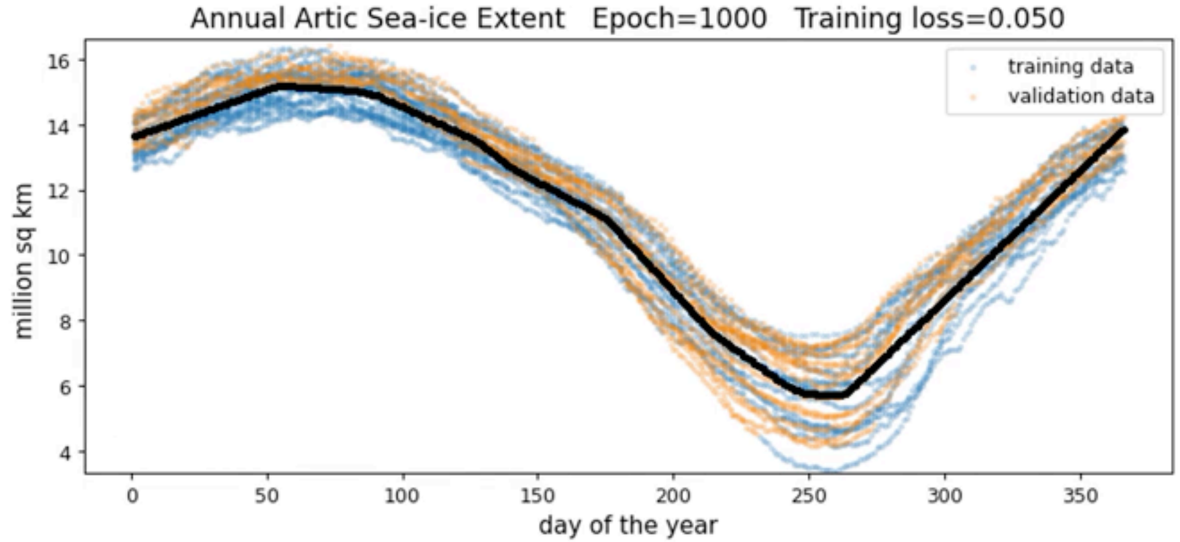


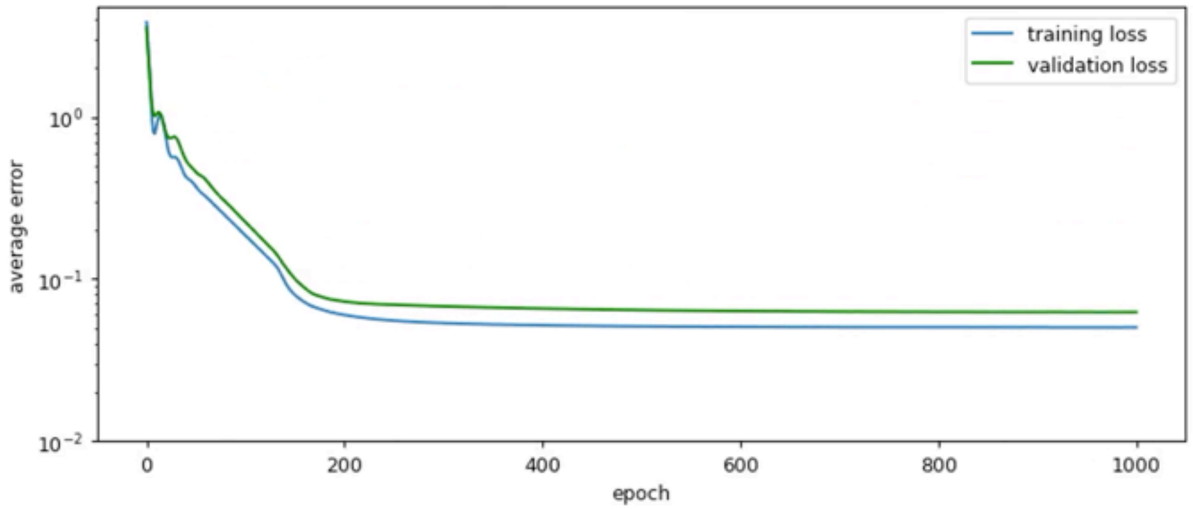


SHALLOW NEURAL NET 1D

Using nn.sequential

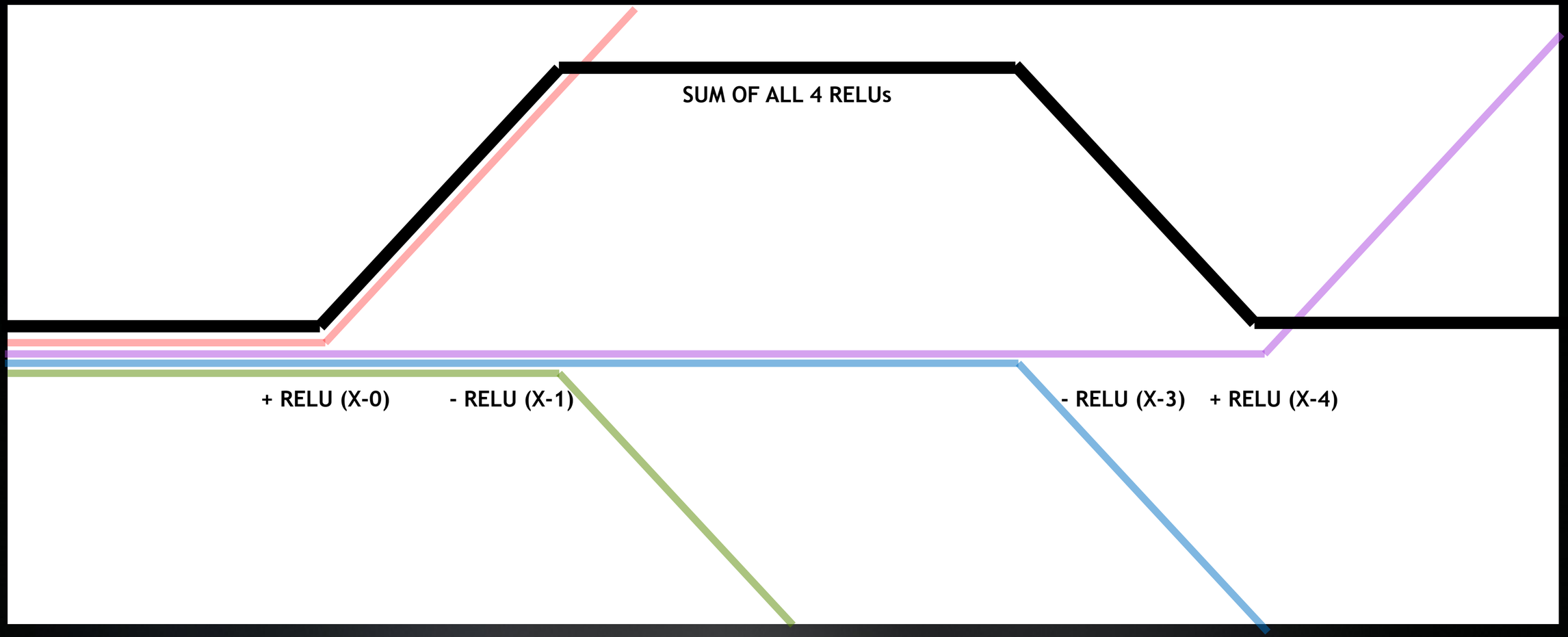
```python
import torch, torch.nn as nn, numpy as np
torch.manual_seed(0)

# DATA
x,y = load_data()
shuffle_blocks(blocksize=100)
ntrain, nval, ntest = (len(x)*torch.tensor([0.50, 0.25, 0.25])).int()
x_train, x_val, xtest = x[:ntrain], x[ntrain:-ntest], x[-ntest:]
y_train, y_val, ytest = y[:ntrain], y[ntrain:-ntest], y[-ntest:]

# MODEL
N = 20
model = nn.Sequential(nn.Linear(1,N), nn.ReLU(), nn.Linear(N,1))

# CONFIGURE
optimizer = torch.optim.AdamW(params = model.parameters(), lr=1e-2)
loss_fcn  = torch.nn.MSELoss()

train_loss, val_loss=[],[]
for epoch in range(1000+1):

  # TRAIN
  prediction = model(x_train)
  optimizer.zero_grad()
  loss = loss_fcn(prediction, y_train)
  loss.backward()
  optimizer.step()
  train_loss.append(loss.item())

  # VALIDATE
  with torch.no_grad():

    p_val = model(x_val)
    loss_val = loss_fcn(p_val, y_val)
    val_loss.append(loss_val.item())
    plot(interval=20)
```





RELU = PIECEWISE-LINEAR

ReLU basis functions can represent any function using straight line-segments

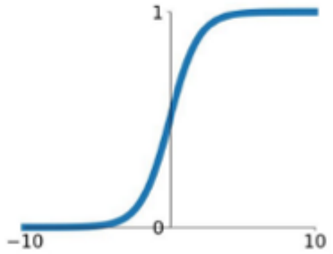


ACTIVATION FUNCTIONS

Many to choose from. But most use ReLU or LeakyReLU

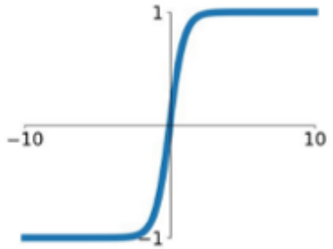
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



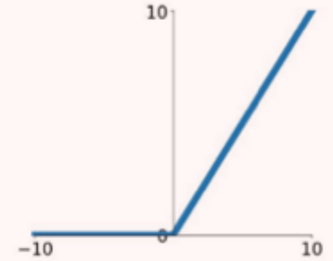
tanh

$$\tanh(x)$$



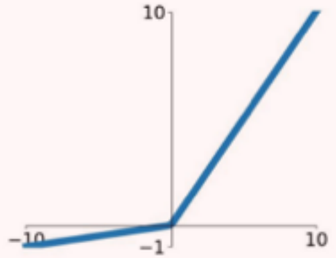
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

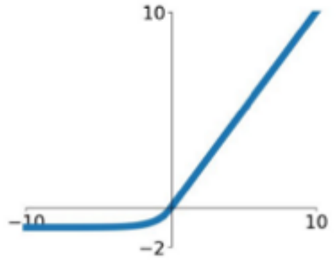


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

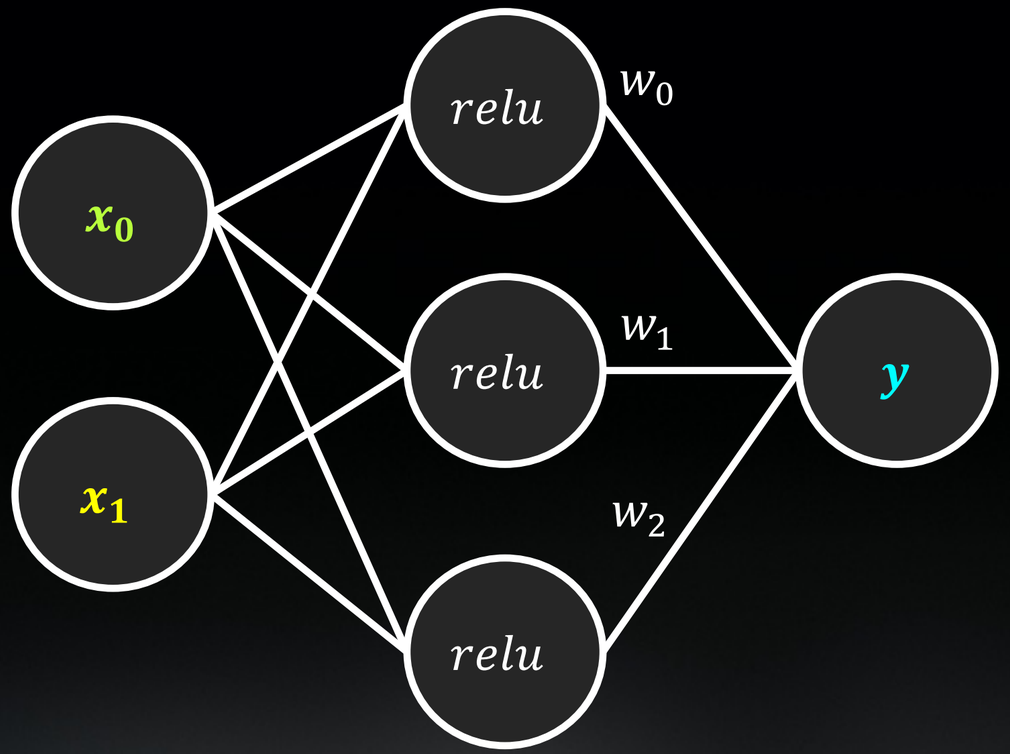
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



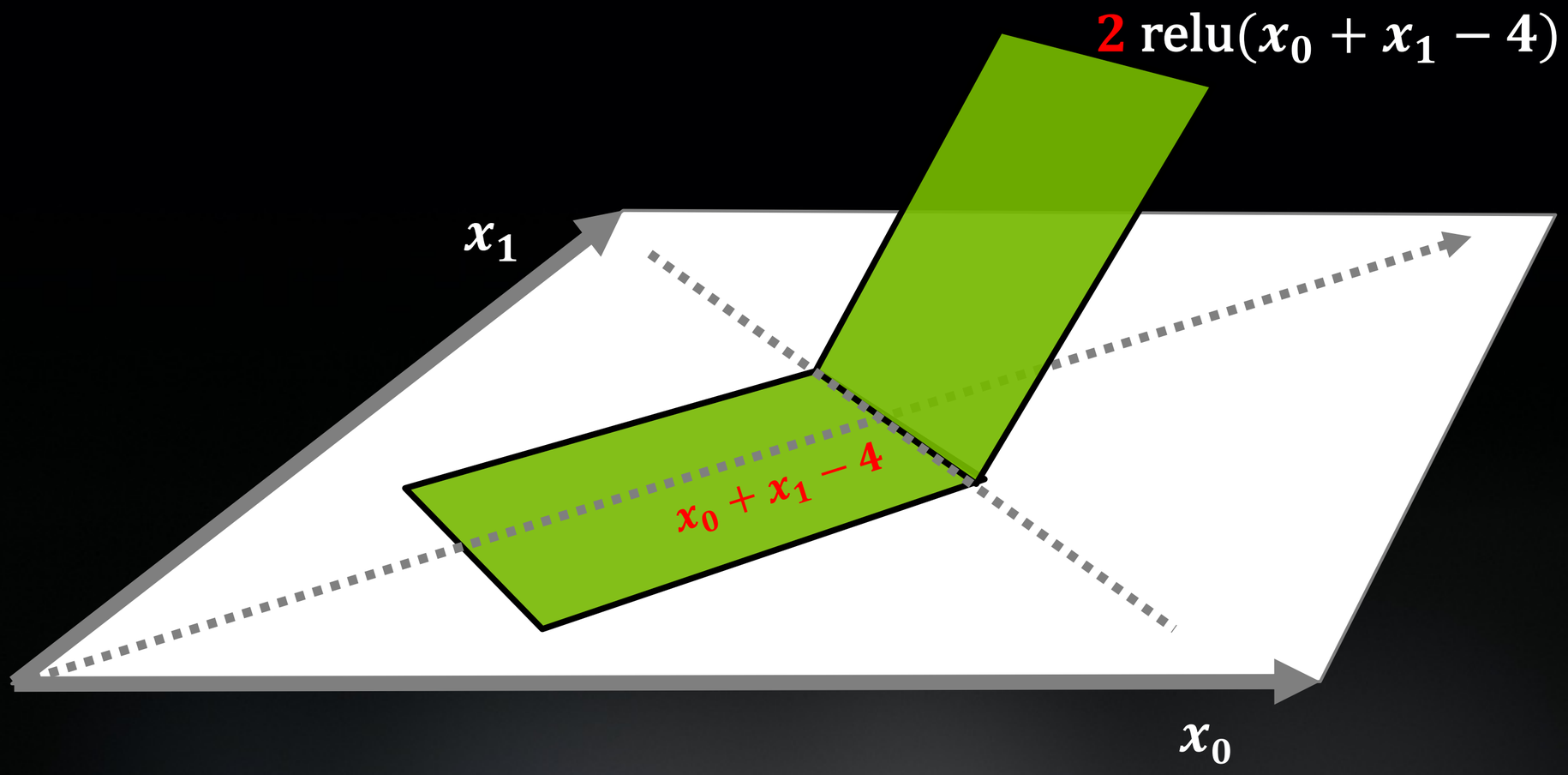
RELU IN 2D

```
model = Sequential(Linear(2,N), ReLU(), Linear(N,1))
```



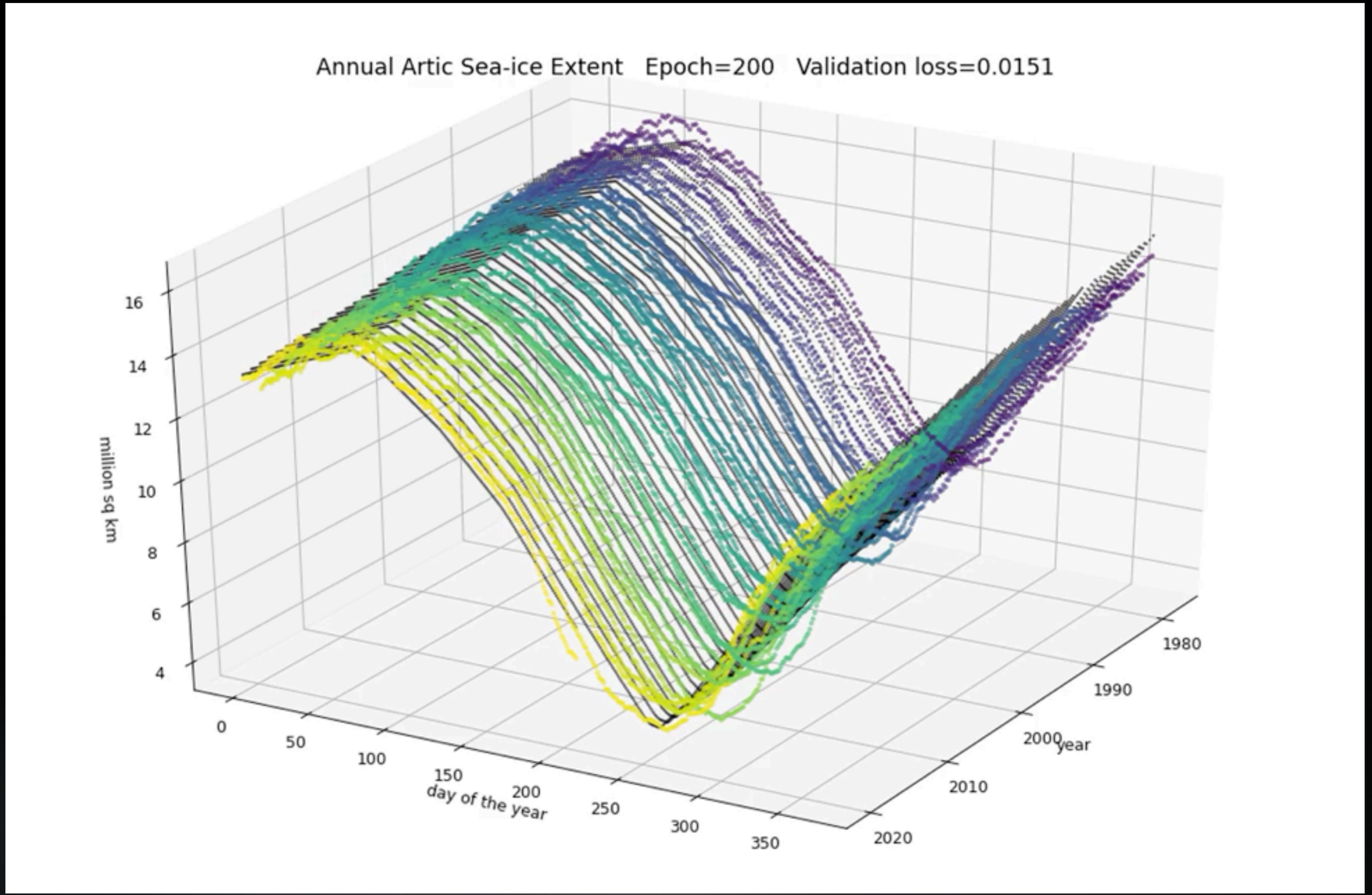
LINEAR, RELU, LINEAR

Linear transformations rotate, shift, and scale the activation function



RELU IN 2D

Find extent = f(year, day of year). Click here for the code.

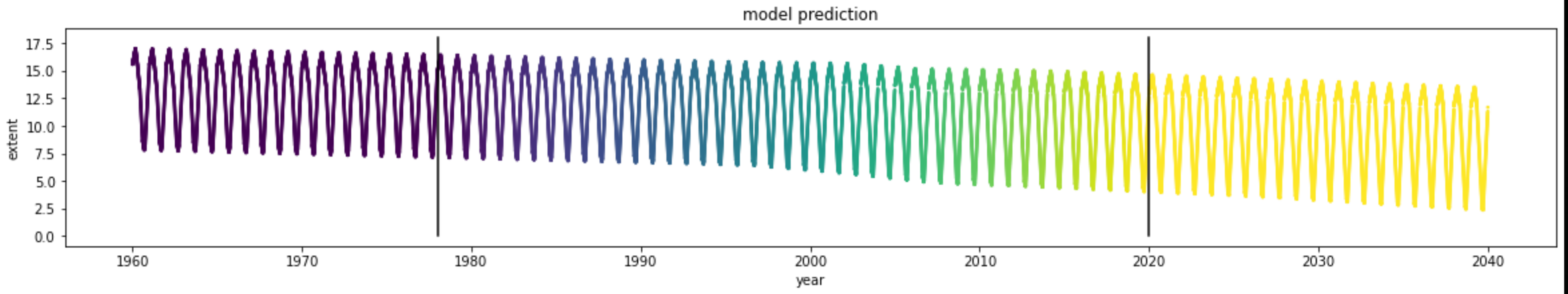


INFERENCE

Once training is complete, we can load and use our model

Reconstruct the same model, then load its parameters from file

```
[2]  1  import torch, torch.nn as nn
     2
     3  # USE SAME MODEL
     4  N = 30
     5  model = nn.Sequential(nn.Linear(2,N), nn.ReLU(), nn.Linear(N,1))
     6
     7  # LOAD
     8  model.load_state_dict(torch.load("/content/drive/My Drive/model-seaice-2d.pt"))
     9  model.eval()
    10  torch.set_grad_enabled(False)
```

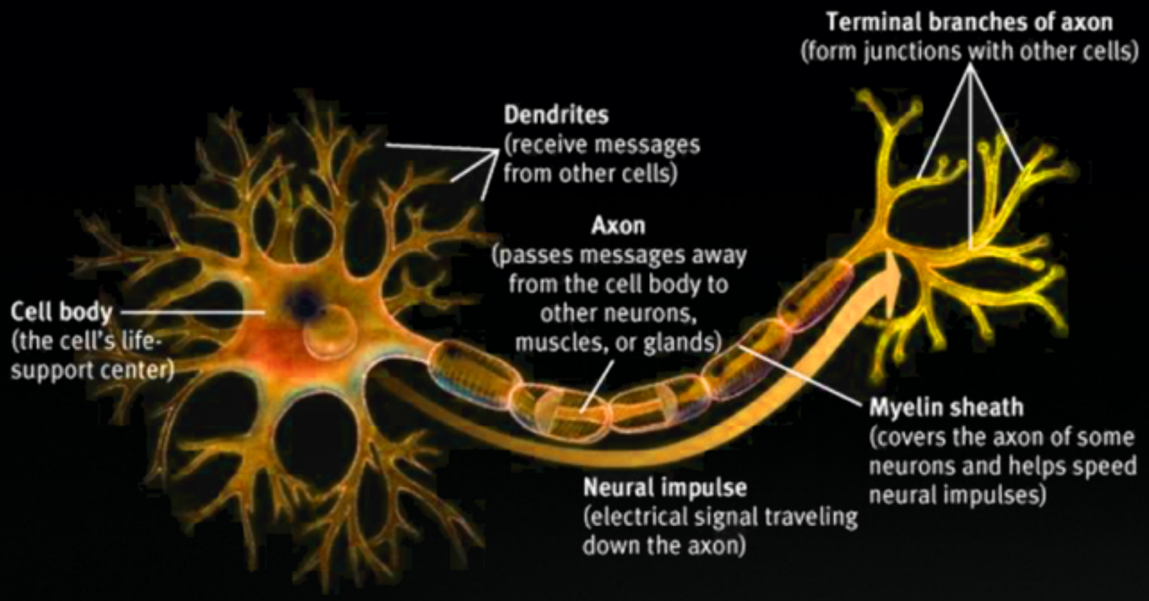


5 MINUTE BREAK

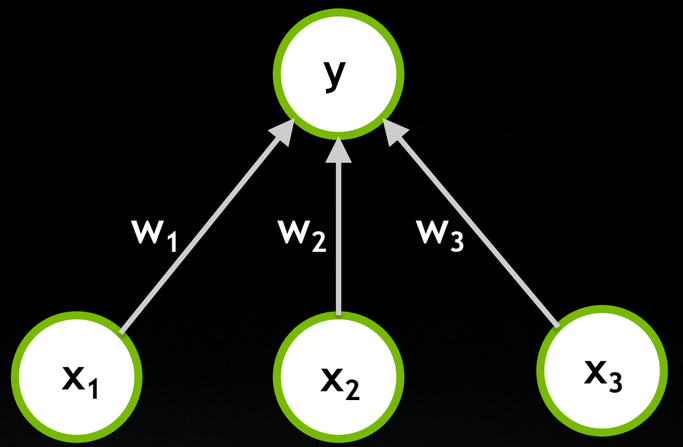
ARTIFICIAL NEURONS

Simple equations with adjustable parameters

Biological neuron



Artificial neuron

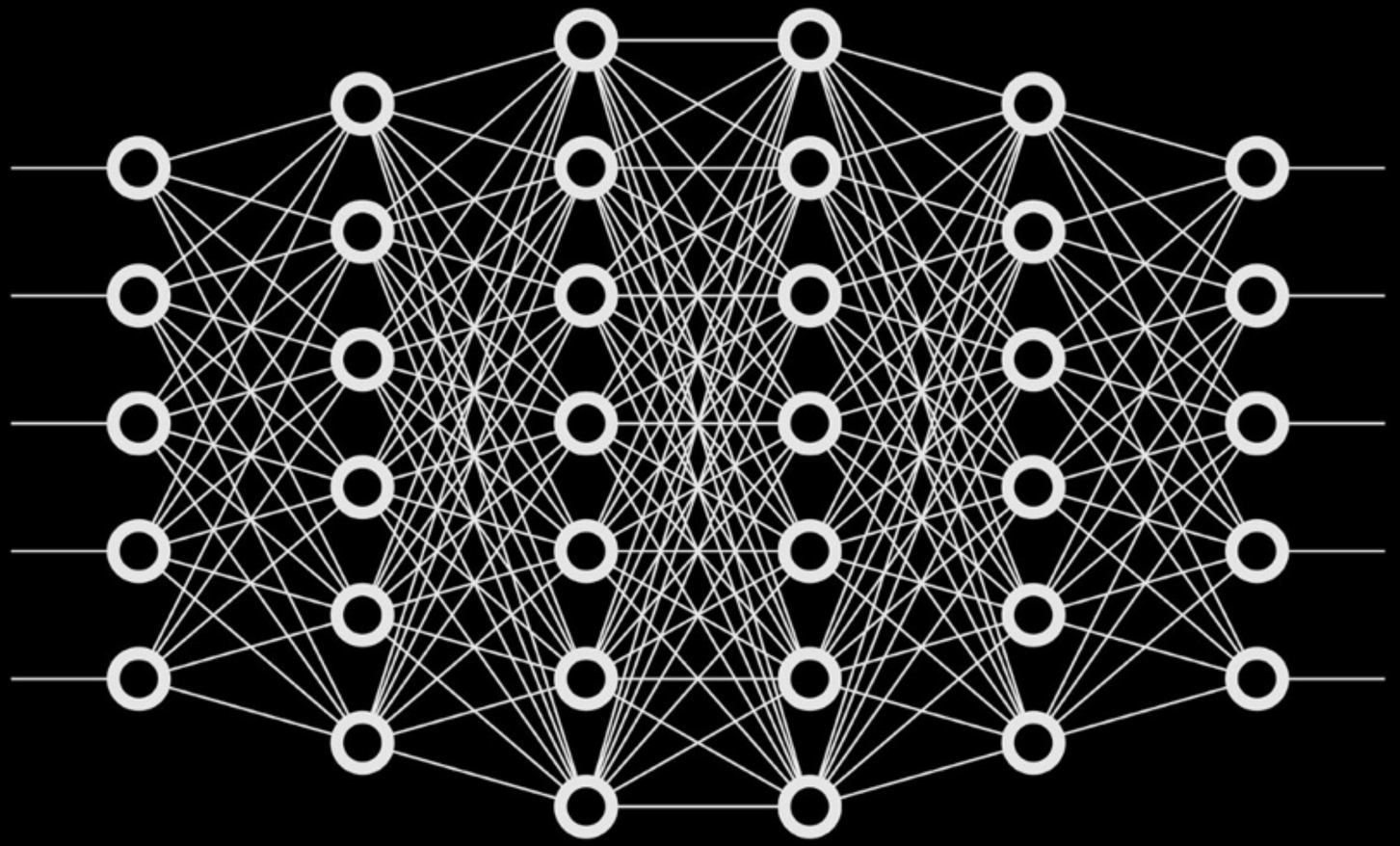


$$y = f(w_1x_1 + w_2x_2 + w_3x_3)$$

https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

FULLY CONNECTED NETWORKS

A given neuron is connected to every neuron in the previous layer

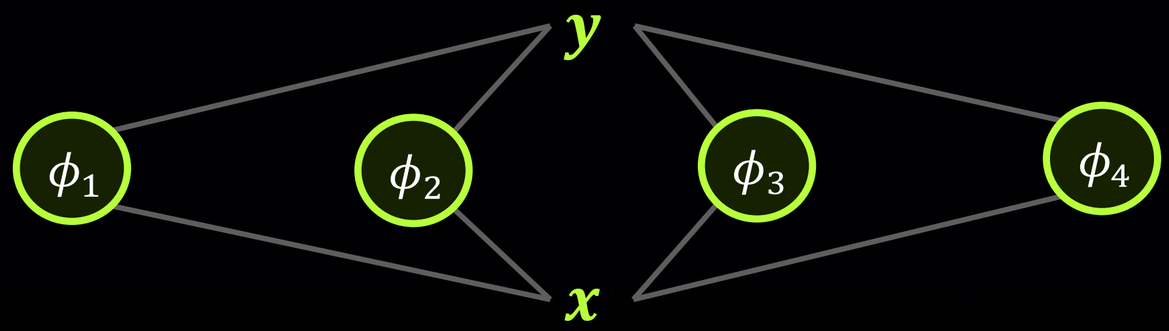


INPUT LAYER1 LAYER 2 LAYER 3 LAYER 4 OUTPUT

SINGLE LAYER NEURAL NETWORKS

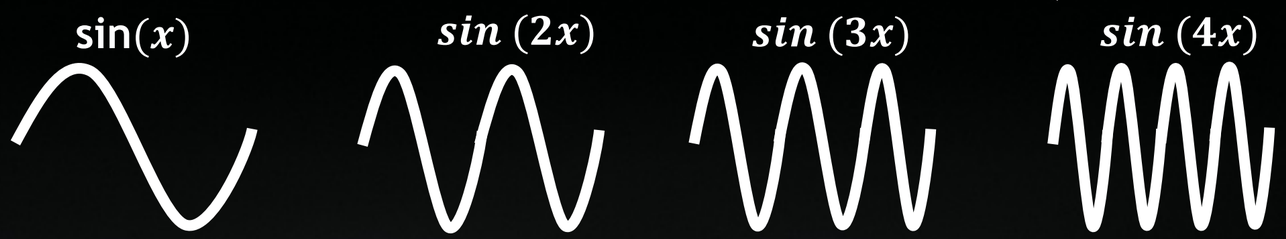
A series expansion over basis functions ϕ.

$$y = \sum_i w_i \, \phi_i(x + b_i)$$



TAYLOR SERIES

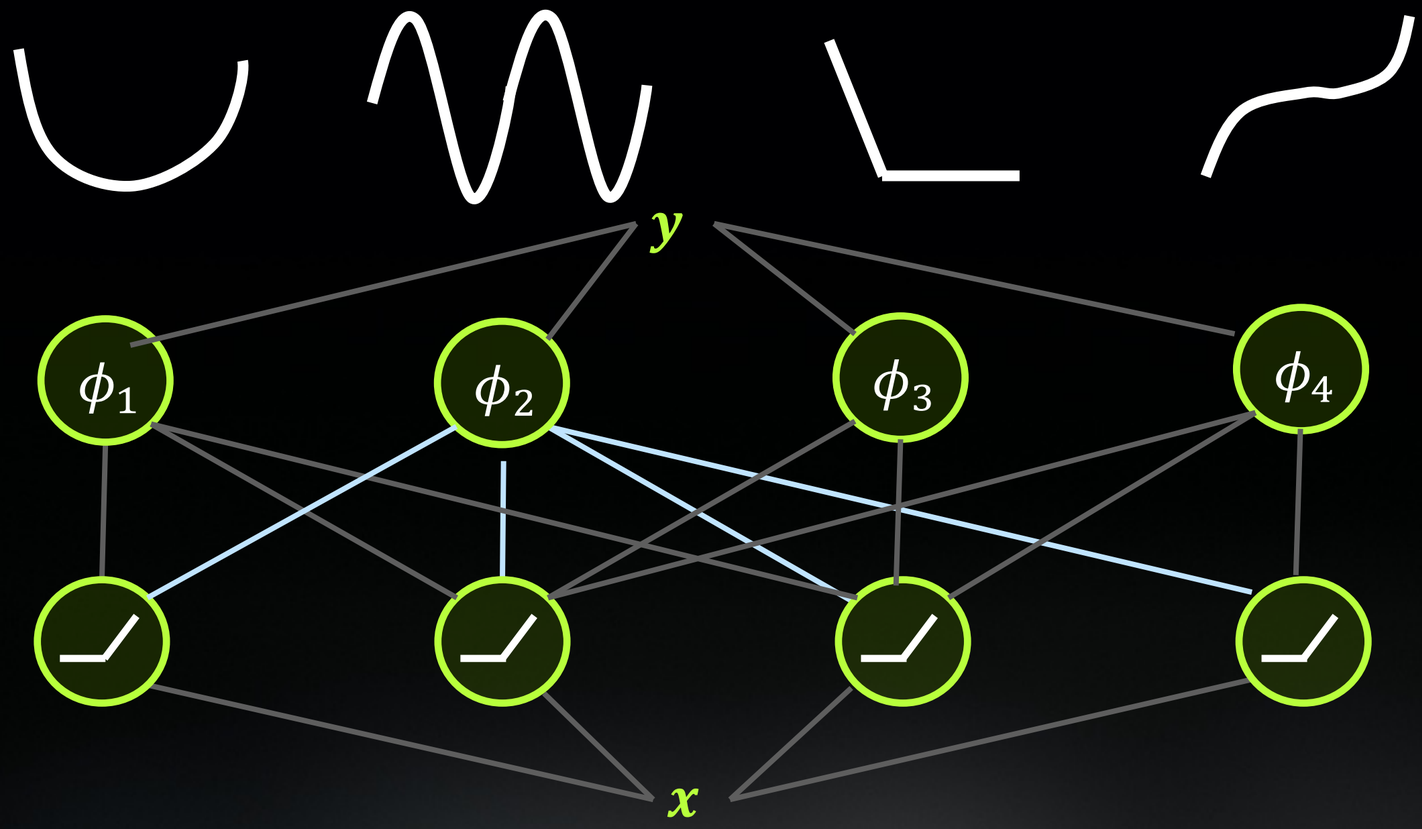
FOURIER SERIES



RELU

TWO LAYER NEURAL NETWORKS

Learn the function and the basis functions at the same time

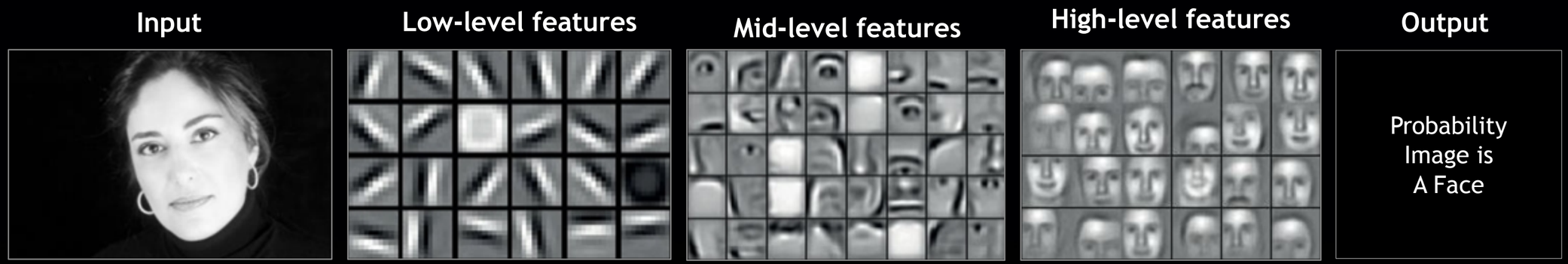


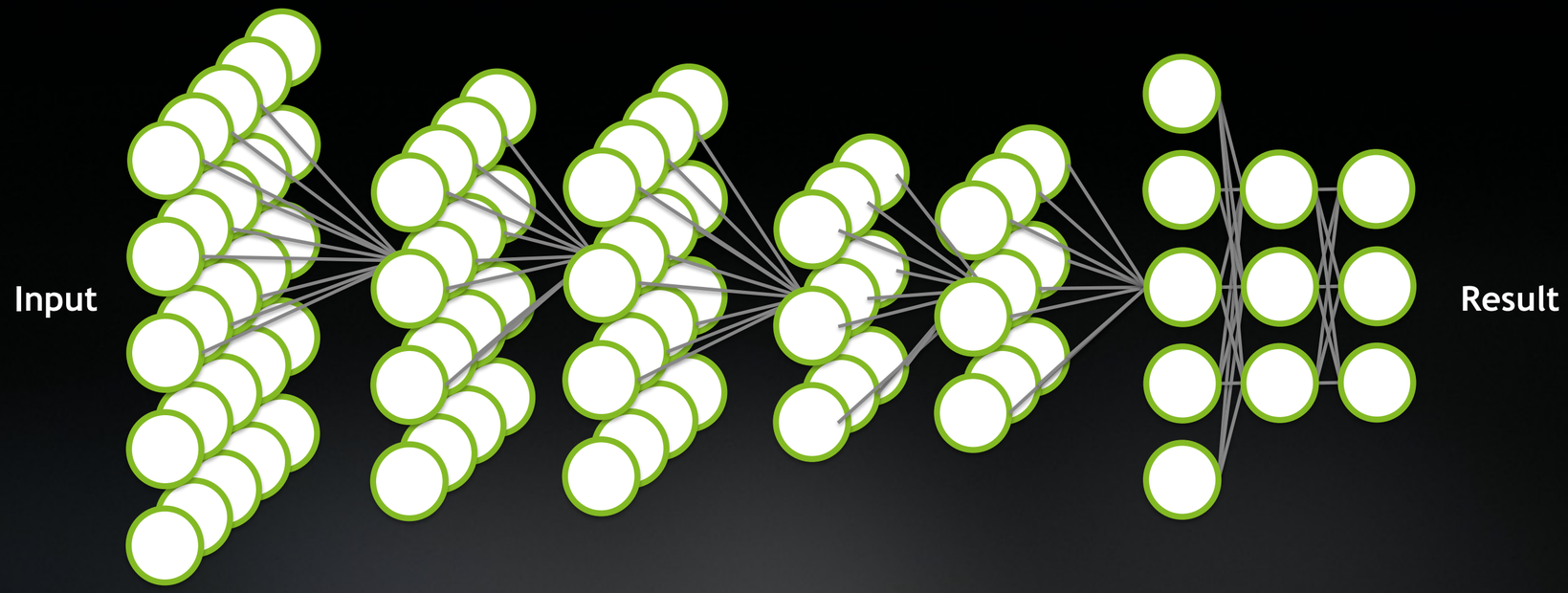
L2: LEARNED BASIS FCNS

L1: RELU BASIS FCNS

DEEPER NEURAL NETWORKS

More layers allows for more levels of abstraction



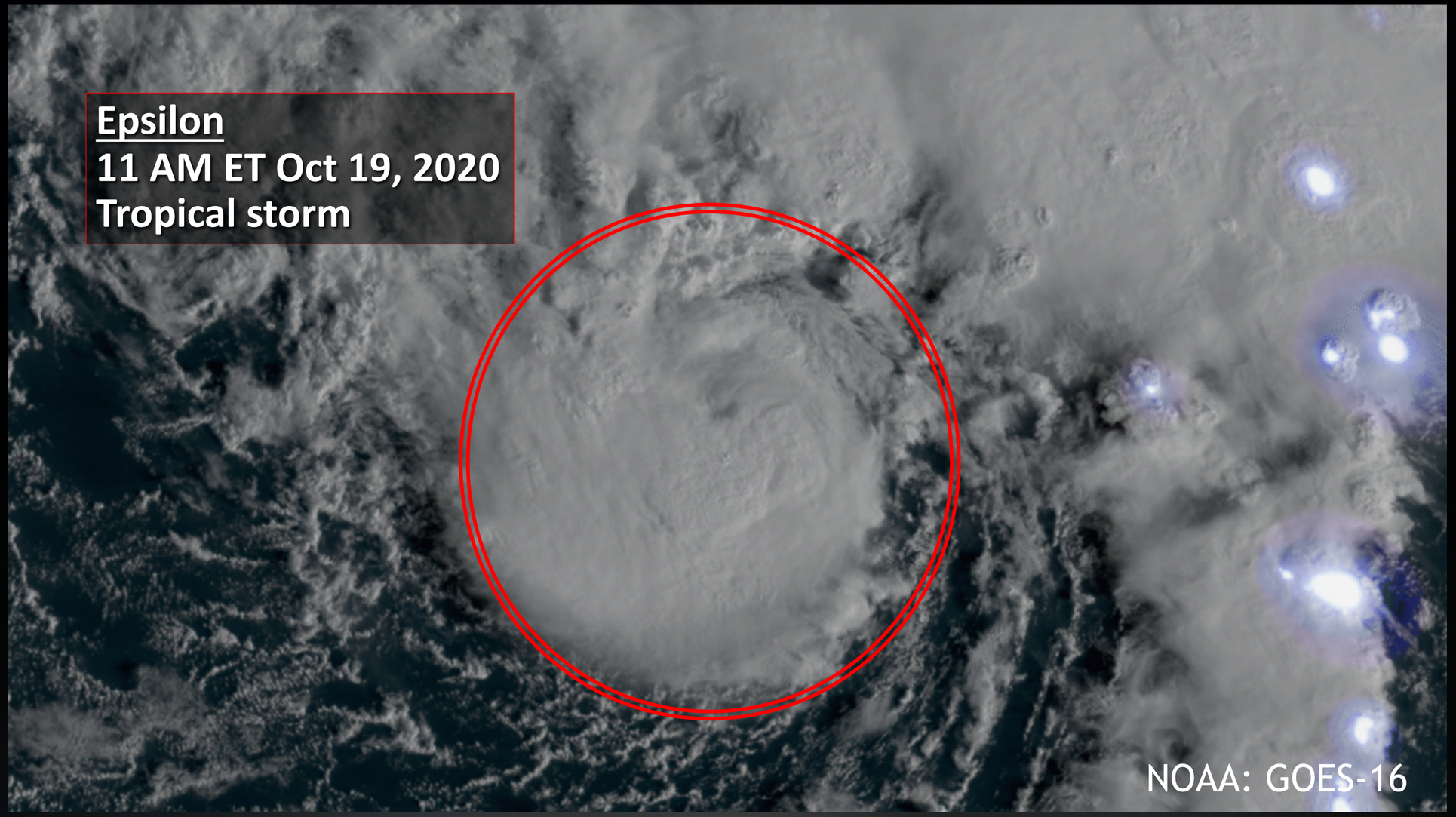


https://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

PART 2: CLASSIFICATION

GOAL: AUTOMATED STORM DETECTION

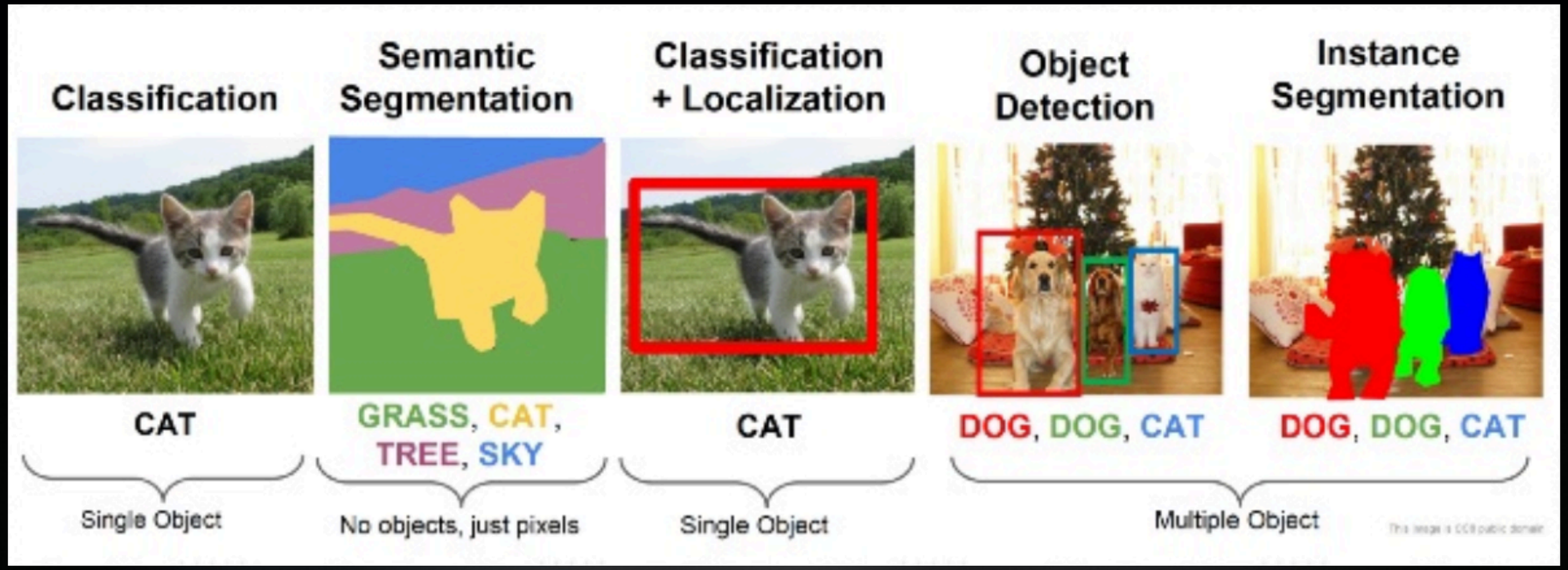
Automatically detect and classify dangerous tropical cyclones



https://www.nesdis.noaa.gov/content/goes-east-spies-newly-formed-tropical-storm-epsilon

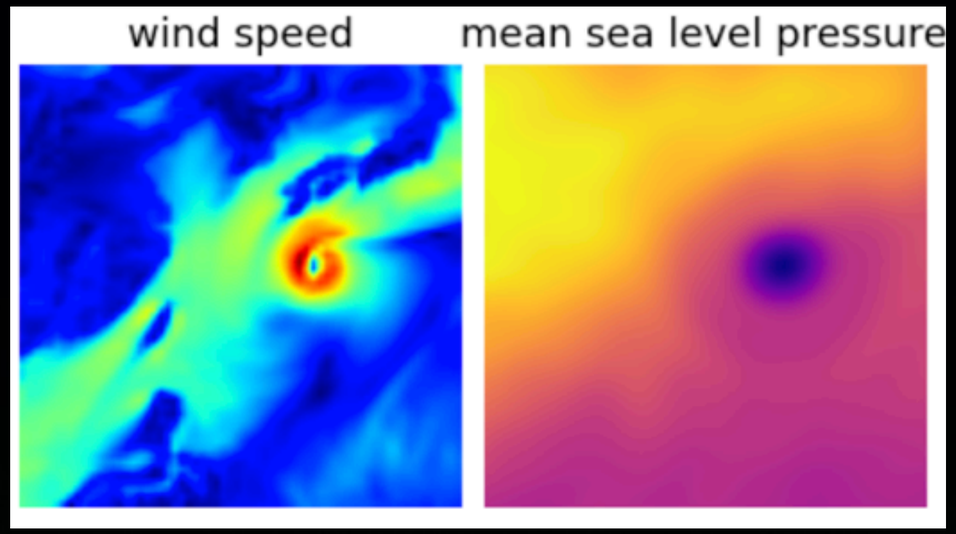
COMPUTER VISION TASKS

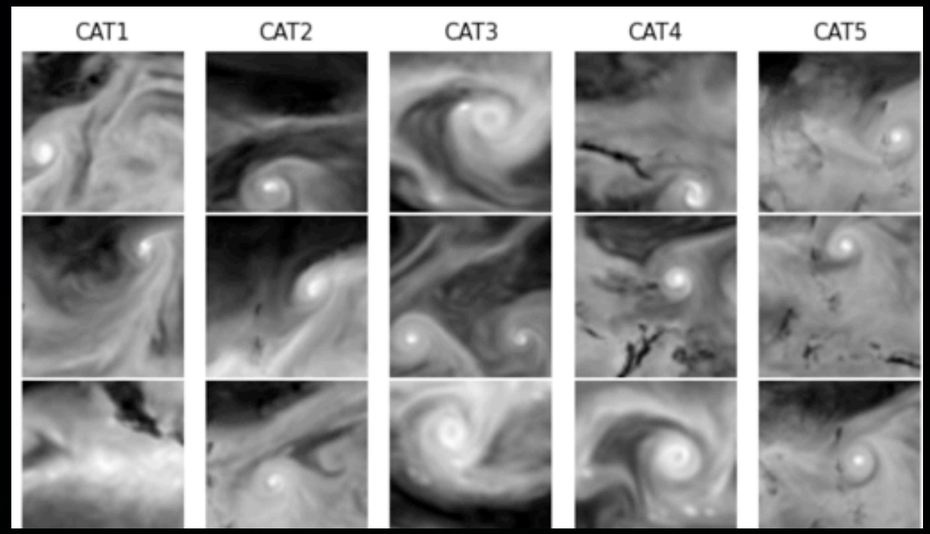
We will start with classification, as it is simplest.



CYCLONE CLASSIFICATION

Given images of various storms, how can we determine the storm type?





TRADITIONAL APPROACH (like TECA)

- Search for low-pressure center
- Measure maximum sustained winds
- Classify storm by location and wind-speed

- Strengths: simple, interpretable

- Weaknesses: Fragile. Requires expert knowledge. Requires wind speed and pressure which may be unavailable. Long development time.

DEEP LEARNING

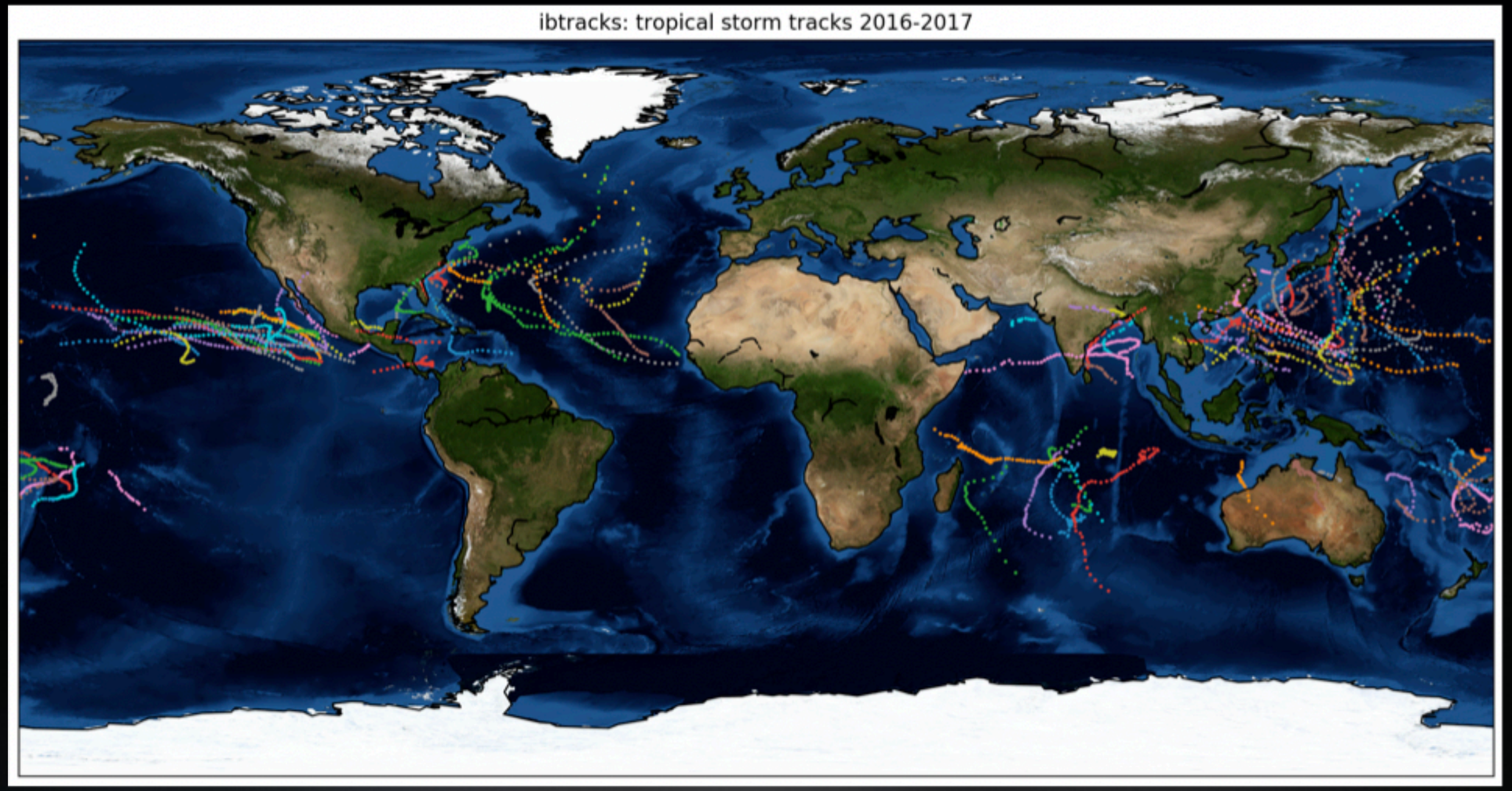
- Gather many labelled examples of each storm
- Let optimizer search for solution

- Strengths: Can use water vapor only
- No expert knowledge needed to build the software. Faster development. Less fragile. Works for most any phenomena. GPU accelerated.

- Weaknesses: requires many labelled examples.
- Can be harder to interpret

IBTRACS

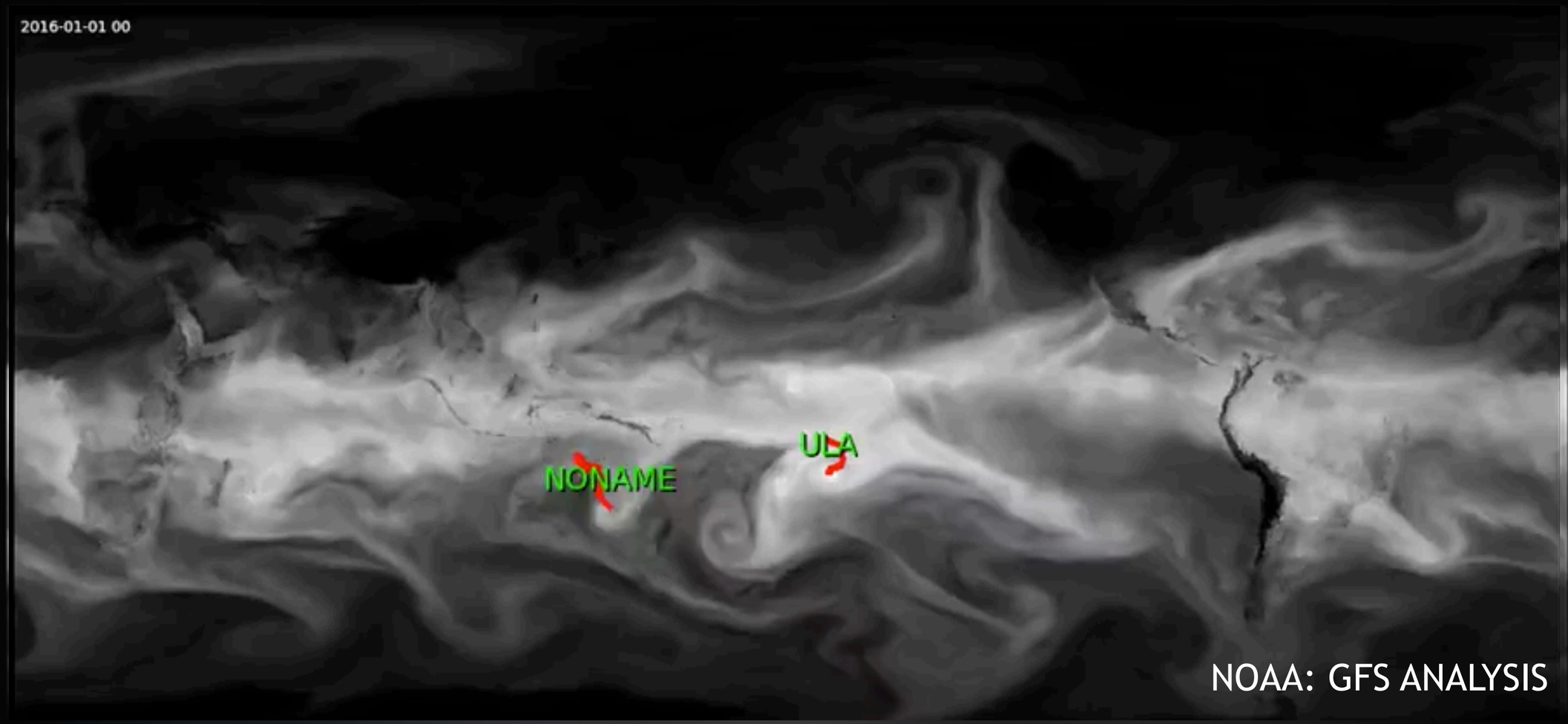
Provides expert labels for historical tropical cyclones



https://www.ncdc.noaa.gov/ibtracs/index.php?name=introduction

GFS ANALYSIS

Model data makes a good starting point for this task.



https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/global-forcast-system-gfs

MAKING A LABELLED DATASET

Download the data and convert it to desired format

Download GFS Analysis Data and IBTRACs data from NOAA

```
import os
day,month,year = 28, 8, 2019
source = f"https://nomads.ncdc.noaa.gov/data/gfsanl/{year}{month:02}/{year}{month:02}{day:02}/"
dest   = os.environ['HOME'] + "/data/original/gfs_analysis_allfields"

!wget -r -c --no-parent -A.grb2 -e robots=off -P {dest} {source}
```

```
source = "https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-stewardship-ibtracs/v04r00
dest   = Path(os.environ['HOME']) / "data/original/ibtracs"

!wget -c -e robots=off -P {dest} {source}
```

Convert GFS data from GRIB format to NetCDF

```
from pathlib import Path
import pygrib
import numpy as np
import os

home       = Path(os.environ['HOME'])
#source     = home/"data/original/gfs_analysis_allfields/nomads.ncdc.noaa.gov"
source     = home/"data/original/gfs_analysis_allfields/nomads.ncdc.noaa.gov/data/gfsanl/201010"

def read_grib(gribfile):
    grbs = pygrib.open(str(gribfile))
    grbs.seek(0)
```

MAKING A LABELLED DATASET

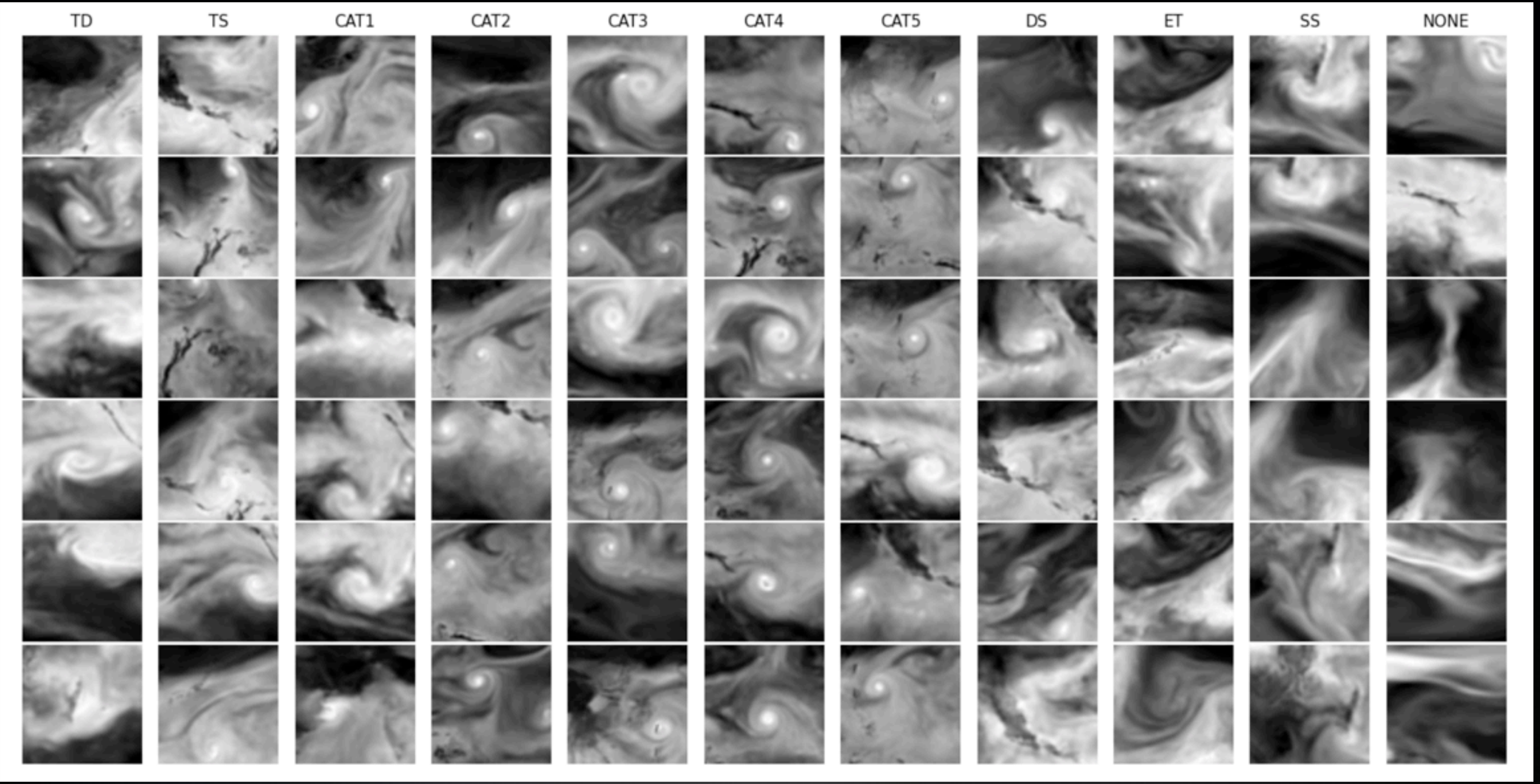
Use IBTRACs to extract labelled examples

For each GFS file, find IBTRACs storms and extract cropped regions

```
21  # get list of gfs netcdf files
22
23  gfs_files = sorted(list(gfs_dir.glob('**/*.nc')))
24
25  # read expert-labelled storm-track database
26
27  ib = xr.open_dataset(Path(os.environ['HOME'])/"data/original/ibtracs/IBTrACS.ALL.v04r00.nc")
28
29  # process each GFS file
30
31  for gfs_file in gfs_files[:]:
32
33      print("reading",gfs_file)
34      gfs = xr.open_dataset(gfs_file)
35
36      # plot GFS water vapor field as an image (optional)
37
38      if plot_gfs:
39
40          gfs.pwat.plot(size=10, yincrease=False, cmap='bone')
41          fig=plt.gcf(); fig.set_size_inches(20,10)
42          plt.show()
43
44      # read time t from GFS file
45
46      gfs_time = datetime.strptime(gfs_time,'%Y-%m-%d %H:%M:%S')
```

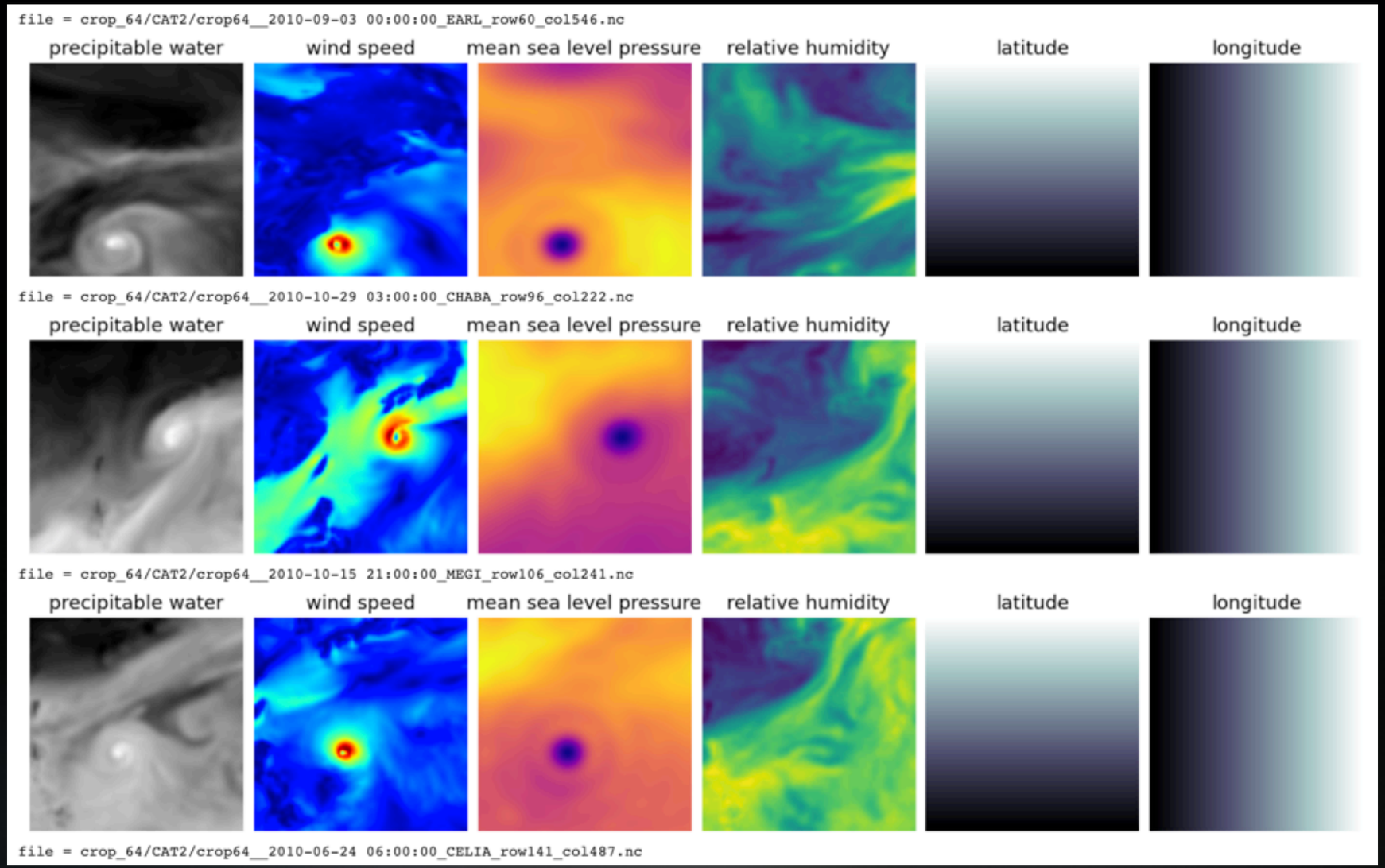
CYCLONE CLASSIFICATION

Find function: category = f(water vapor image)



AVAILABLE FIELDS

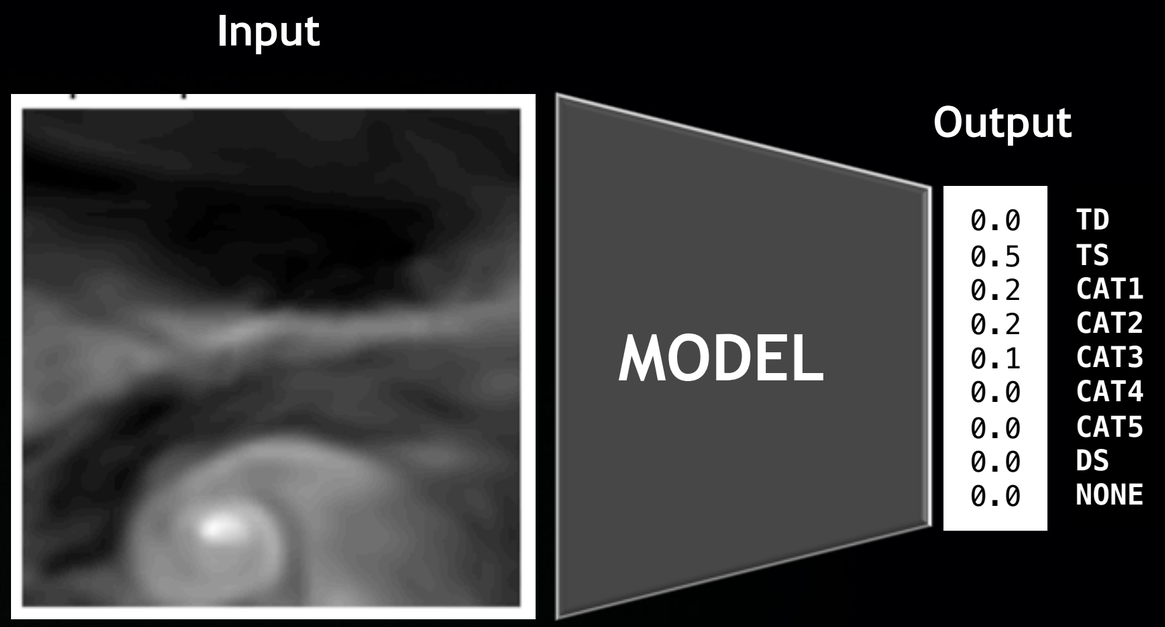
Each NetCDF field contains the following fields



```python
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
import torch, torch.nn as nn
from time import time
t_start = time()

# DATA
def get_data():…
train_loader, val_loader = get_data()

# MODEL
def get_model():…
model = get_model().to(device)
count_parameters(model)

# CONFIGURE
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3)
loss_fcn  = nn.CrossEntropyLoss()
writer    = SummaryWriter()

for epoch in range(30) :

  # VALIDATE
  with torch.no_grad():
    model.eval()
    val_loss = running_mean()
    val_acc  = accuracy_metric()
    for i, data in enumerate(val_loader, 0):…

  # TRAIN
  model.train()
  train_acc  = accuracy_metric()
  train_loss = running_mean()
  gpu        = running_mean()
  t0         = time()
  for i, data in enumerate(train_loader, 0):…

  # display training / validation progress
```

CYCLONE CLASSIFICATION

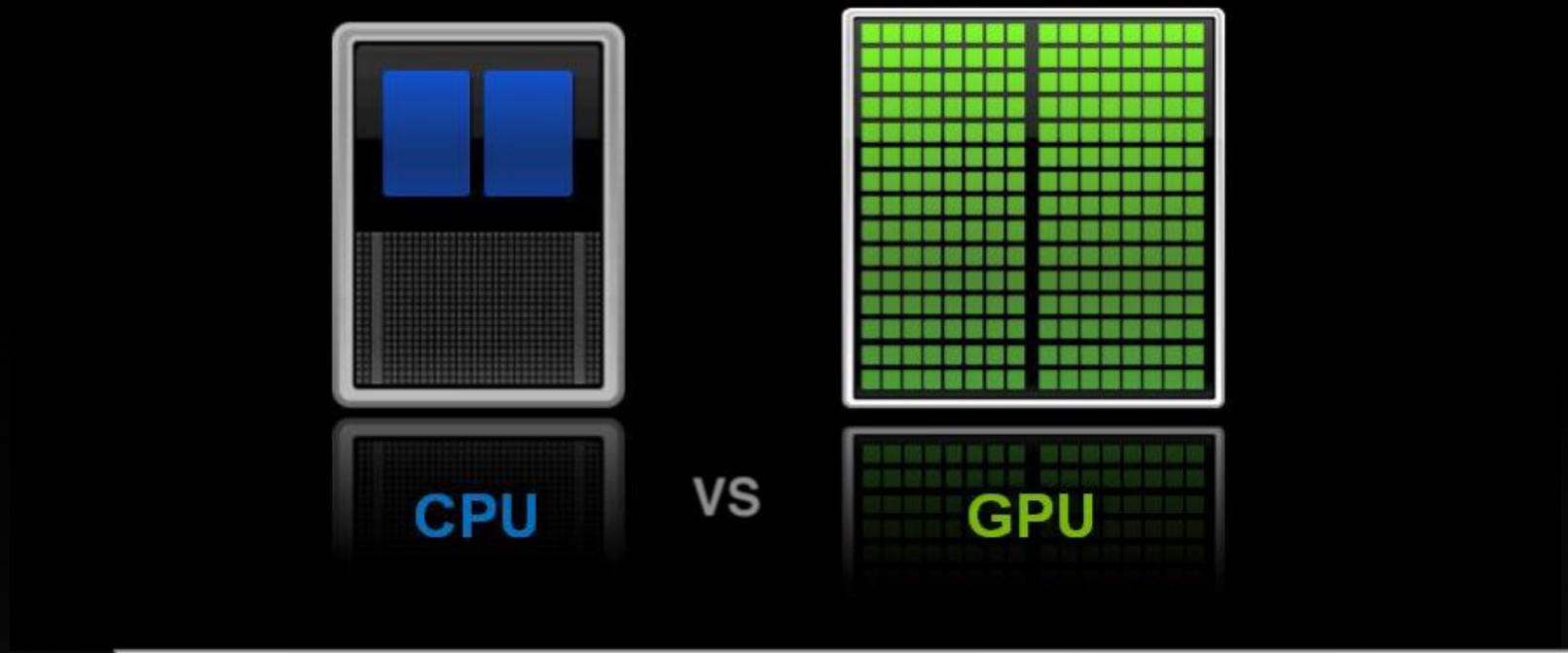
Training



http://alexlenail.me/NN-SVG/AlexNet.html

GPU ACCELERATION

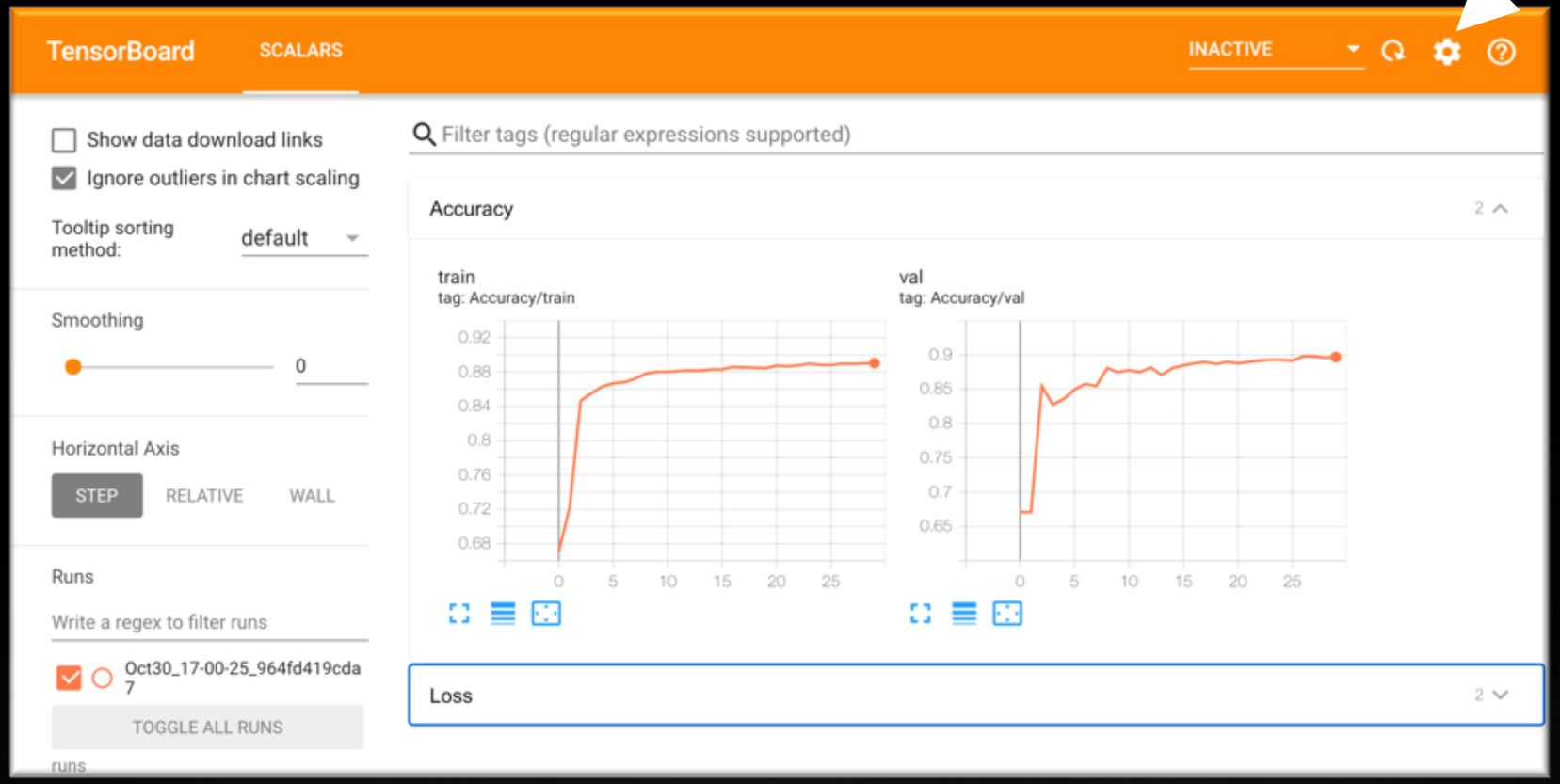
GPUs make deep learning practical.




```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = get_model().to(device)
inputs, labels = data[0].to(device), data[1].to(device)
```

SUMMARY WRITER

Uses Tensorboard to keep track of training losses

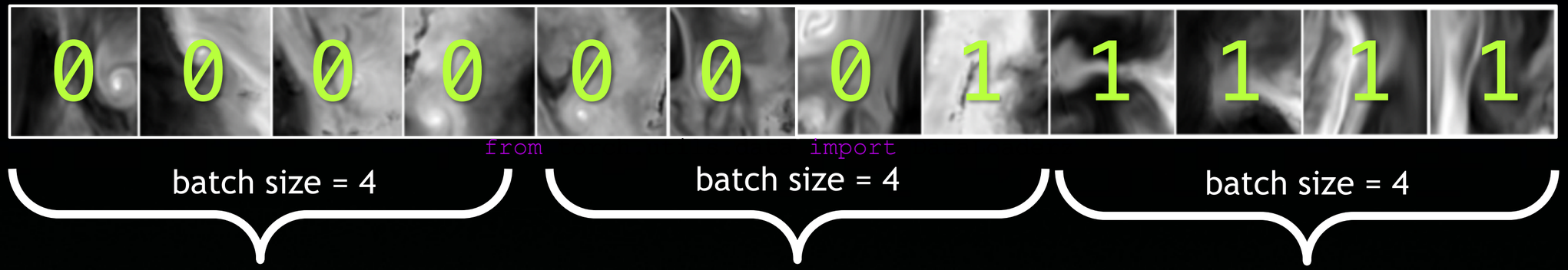



```
%tensorboard --logdir runs

from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
writer.add_scalar('Accuracy/val' ,  val_acc.value() ,   epoch)
writer.add_scalar('Accuracy/train', train_acc.value() , epoch)
```

DATASET AND DATALOADER

Classes used to organize and return examples during training

DATASET: Holds a set of examples. (Might read them from files.)

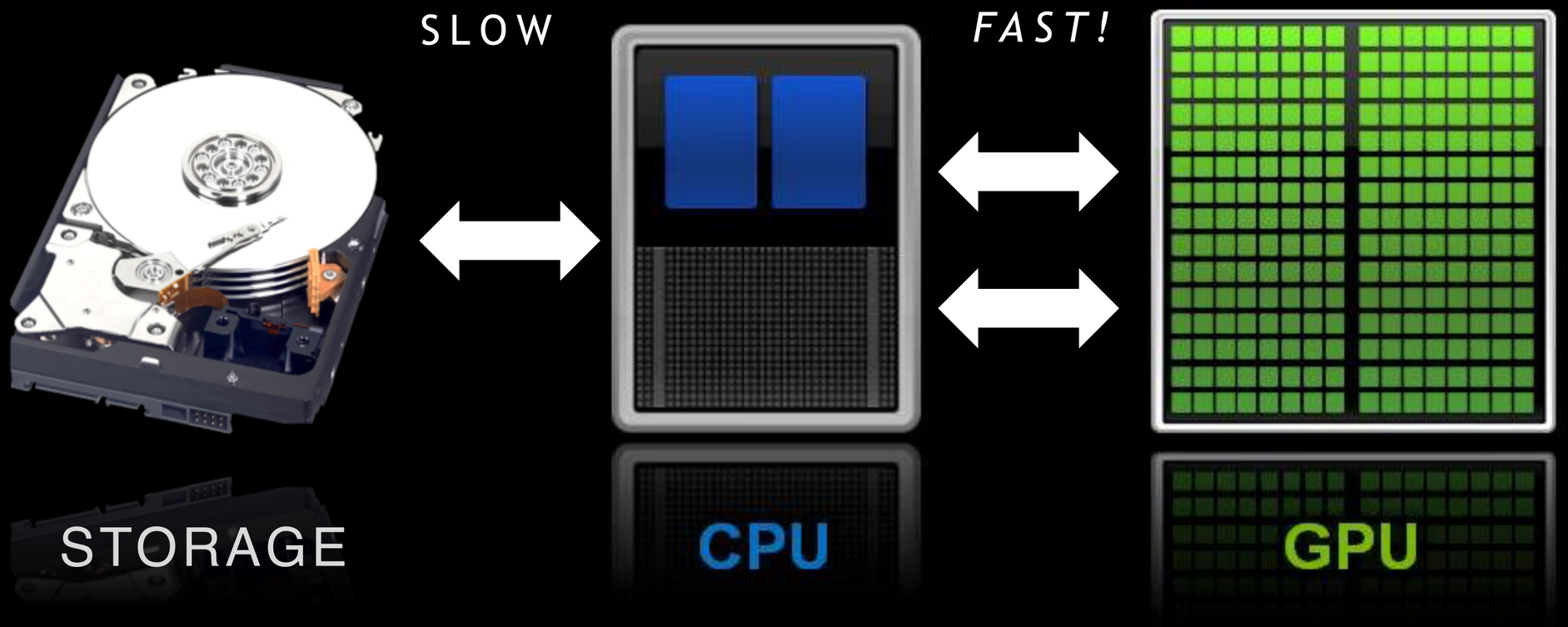


DATALOADER: Returns a batch of examples each time it's called

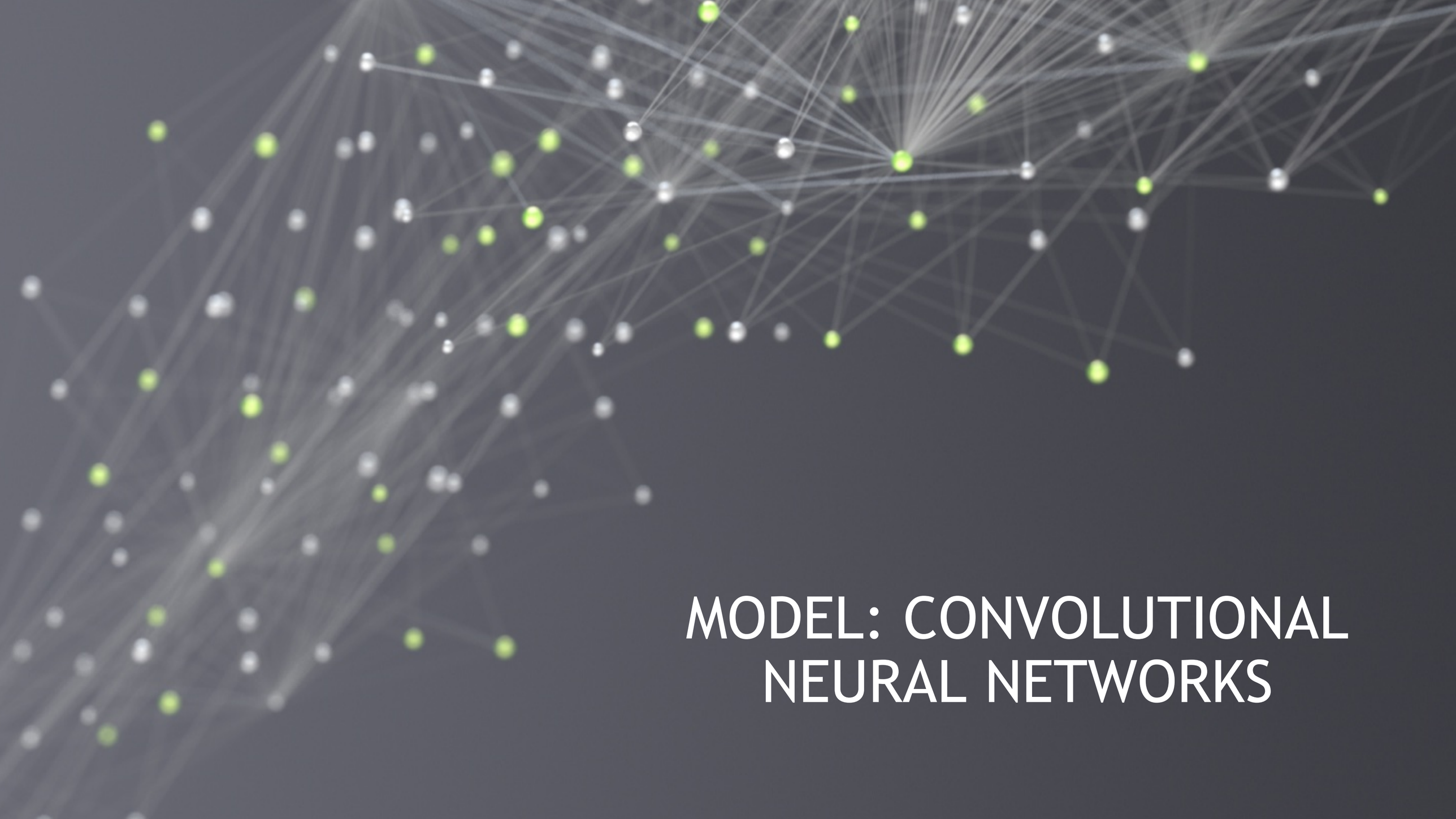
```
from torch.utils.data import DataLoader
train_data   = Dataset(path, inputs, outputs)
train_loader = DataLoader(train_data, batch_size=200, shuffle=True)
for i, data in enumerate(train_loader):
```

DATA CACHE

Store as many examples as possible in RAM to minimize disc access




```
cache = Cache(loading_function, device="cuda", memory_cap=80)
if path in cache: return cache[path]
else: load(file, device)
```



MODEL: CONVOLUTIONAL NEURAL NETWORKS

A SIMPLE CLASSIFIER

A basic encoder, using convolution layers

Previous Model: extent = f(year, day of year)

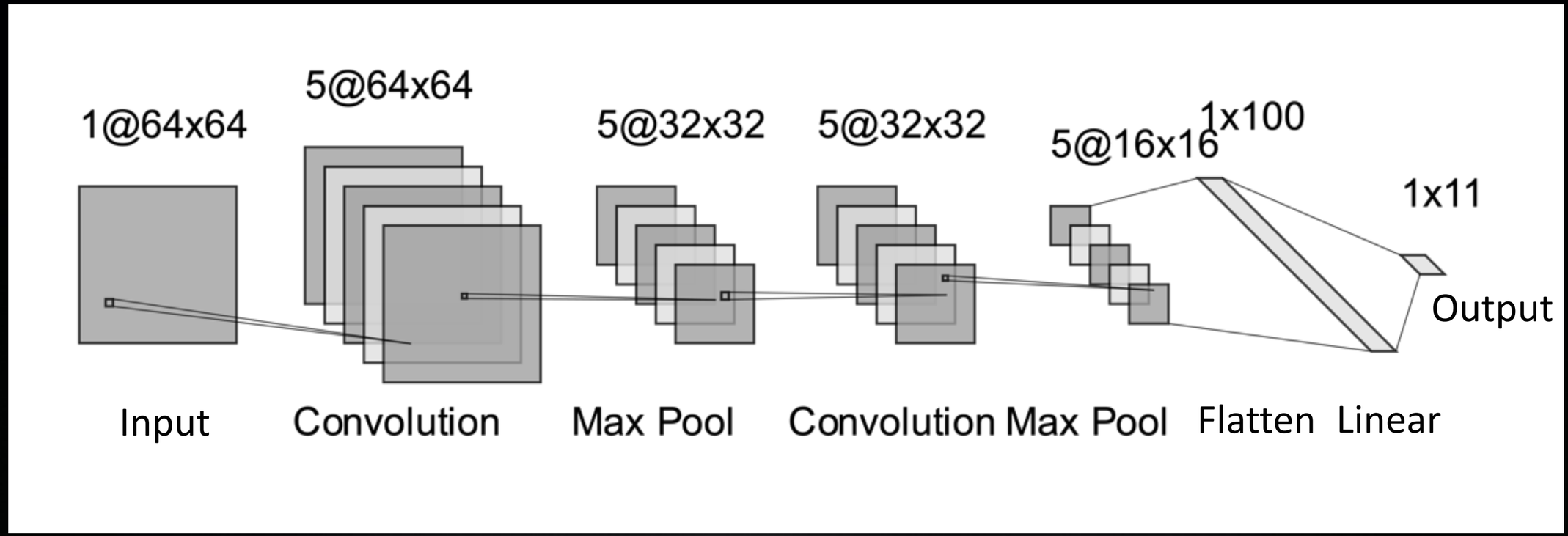
```
model = nn.Sequential(
      Linear(2,N), ReLU(),
      Linear(N,N), ReLU(),
      Linear(N,1))
```

Current model: prob of storm = f(water-vapor image)

```
model = nn.Sequential(
      Conv2d(Cin,C, 3), ReLU(), MaxPool2d(2),
      Conv2d(C , C, 3), ReLU(), MaxPool2d(2),
      Flatten()
      Linear(C*nrow*ncol//(4*4*4), N), ReLU(),
      Linear(N, Nout))
```

SIMPLE CLASSIFIER

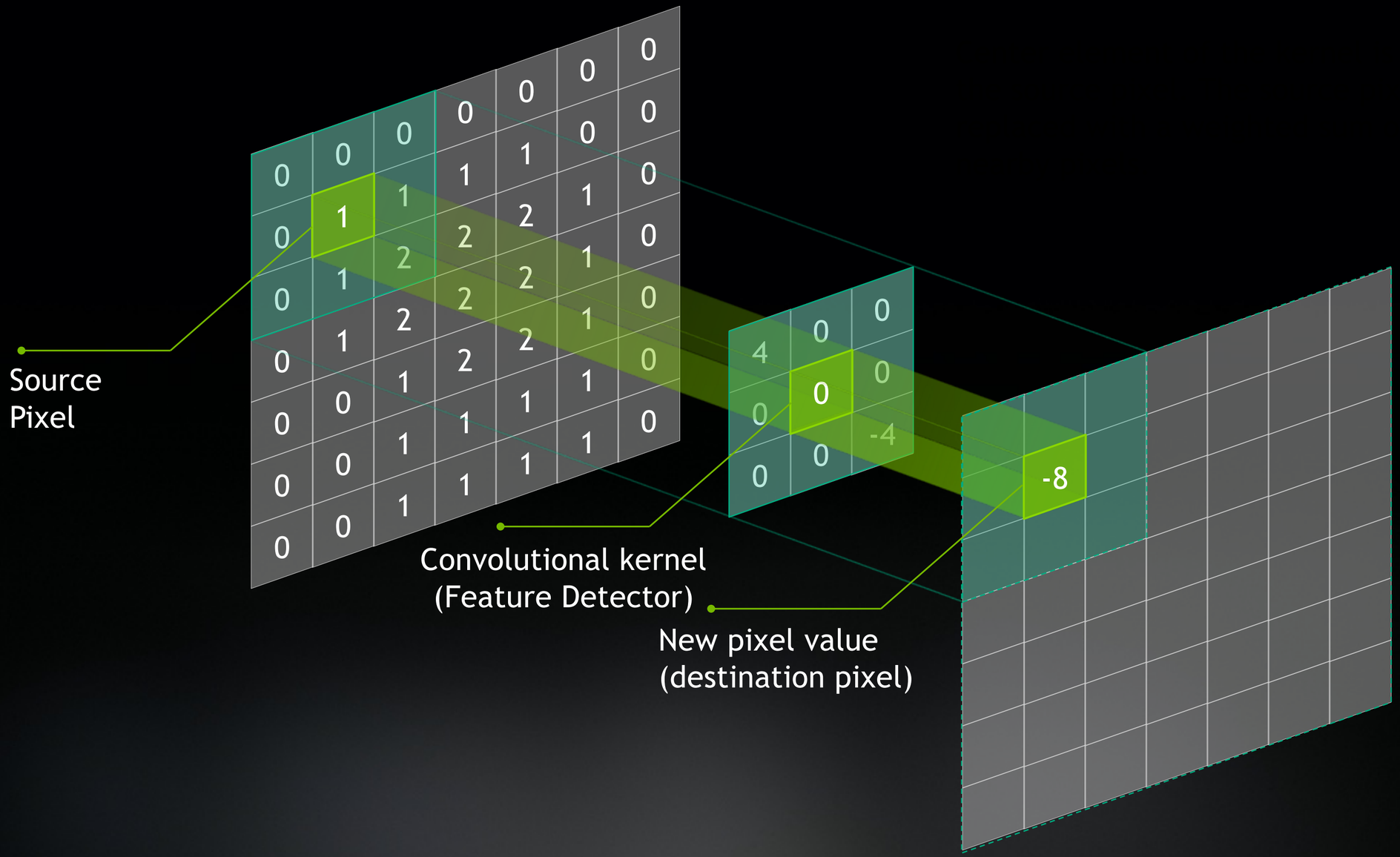
Represented as a diagram



http://alexlenail.me/NN-SVG/LeNet.html

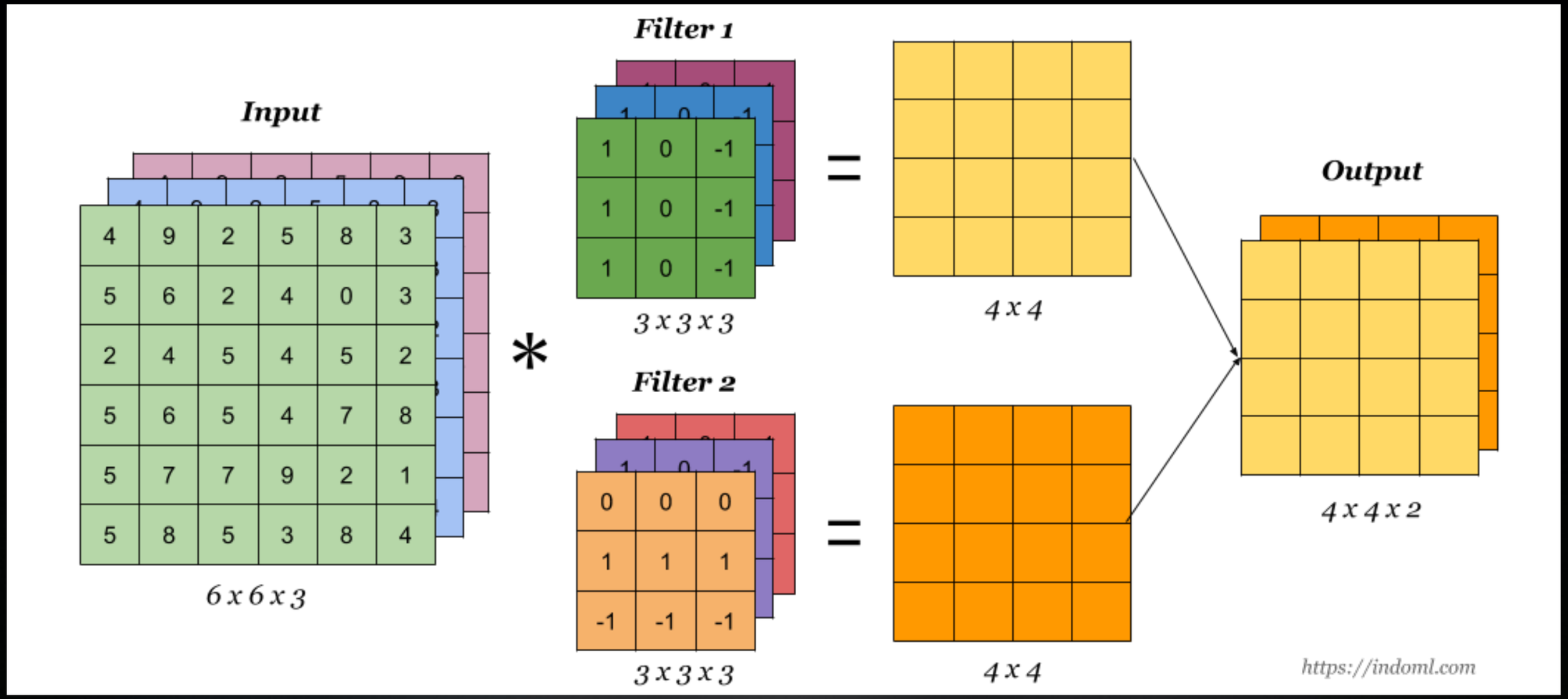
CONVOLUTION-2D

A small matrix transformation, applied at each point of the image

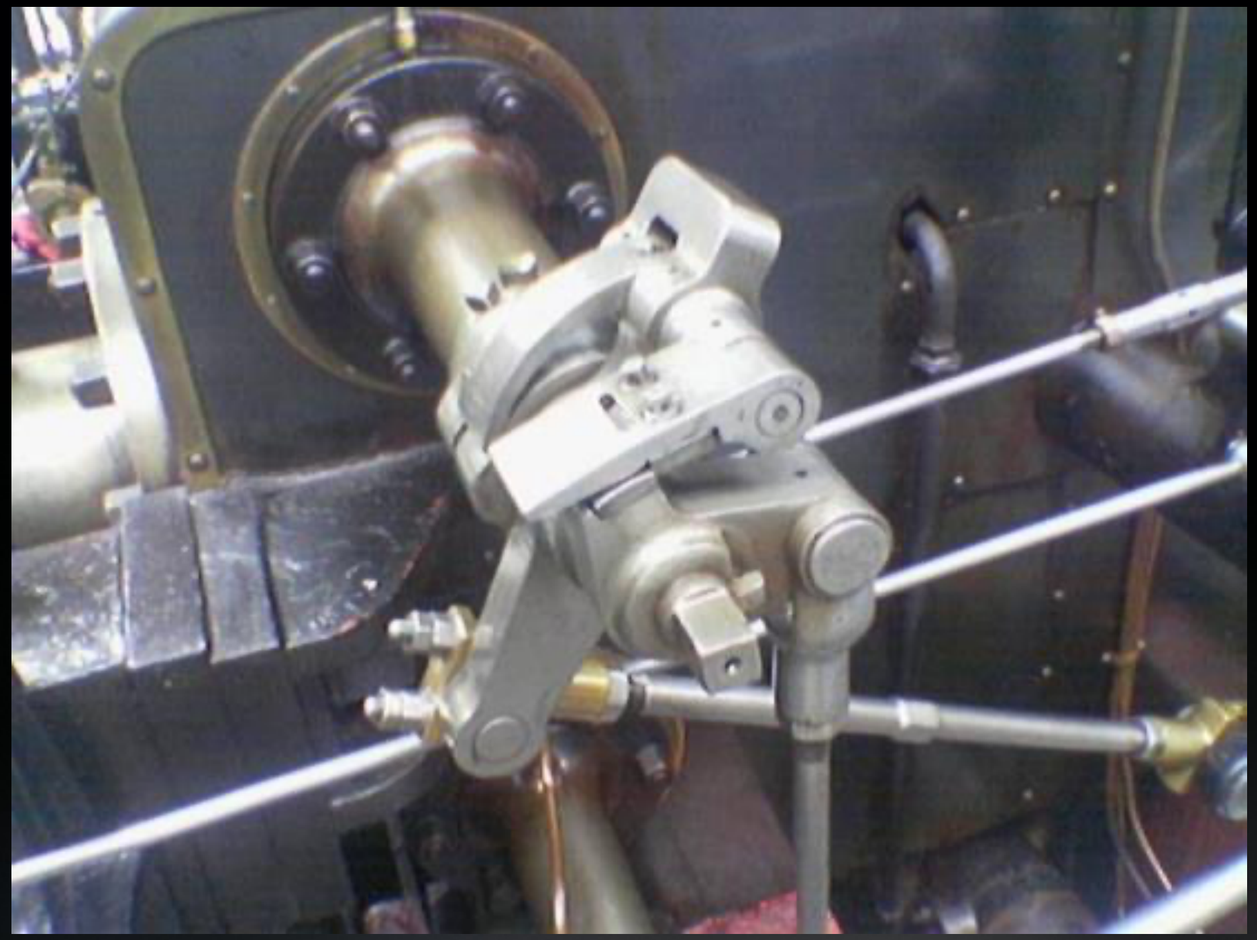


CONVOLUTION-2D

When we have multiple channels, learned kernels are 3d matrices



CONVOLUTION EXAMPLE: SOBEL FILTER



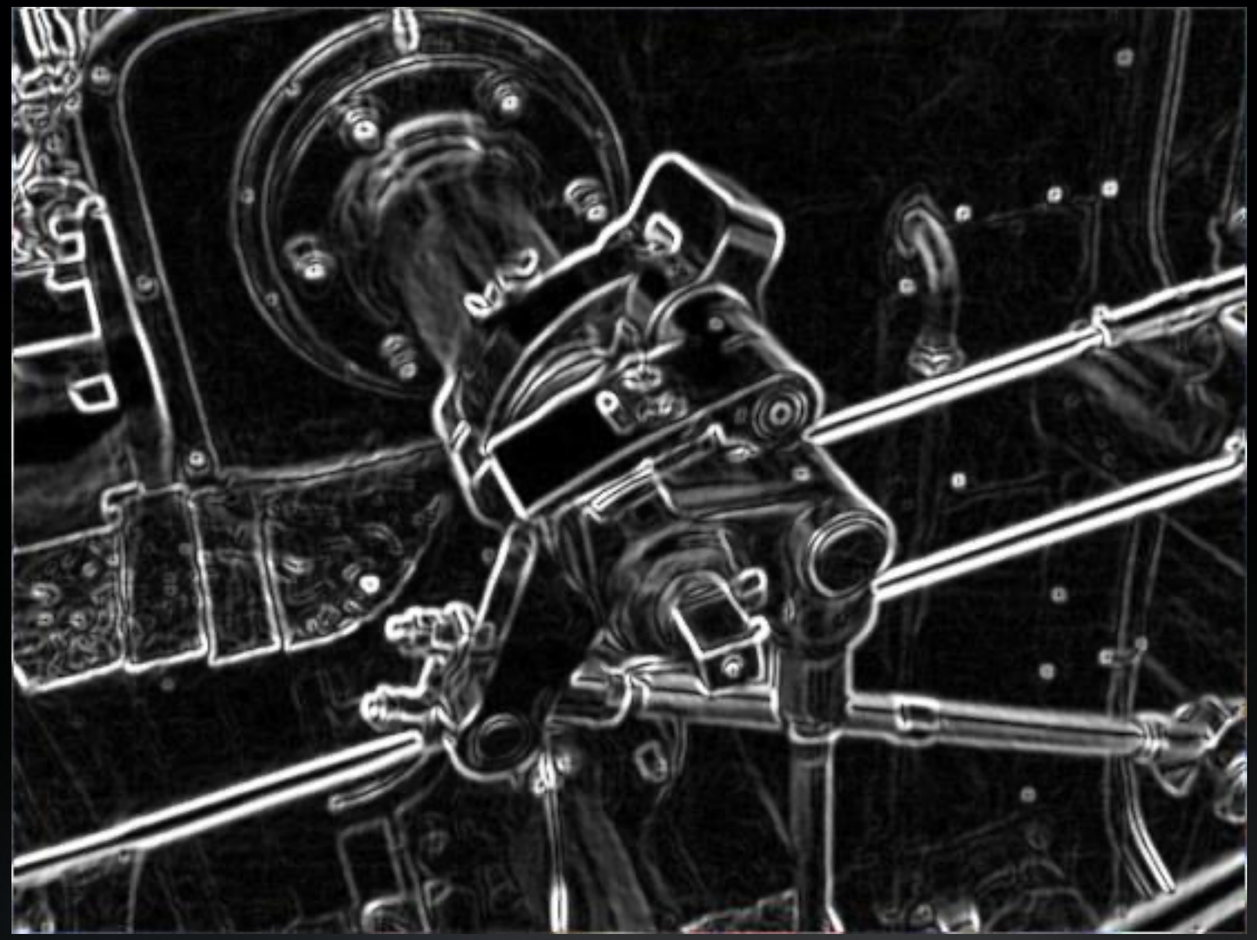
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Image source: https://en.wikipedia.org/wiki/Sobel_operator

CONVOLUTION EXAMPLE: SOBEL FILTER



$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

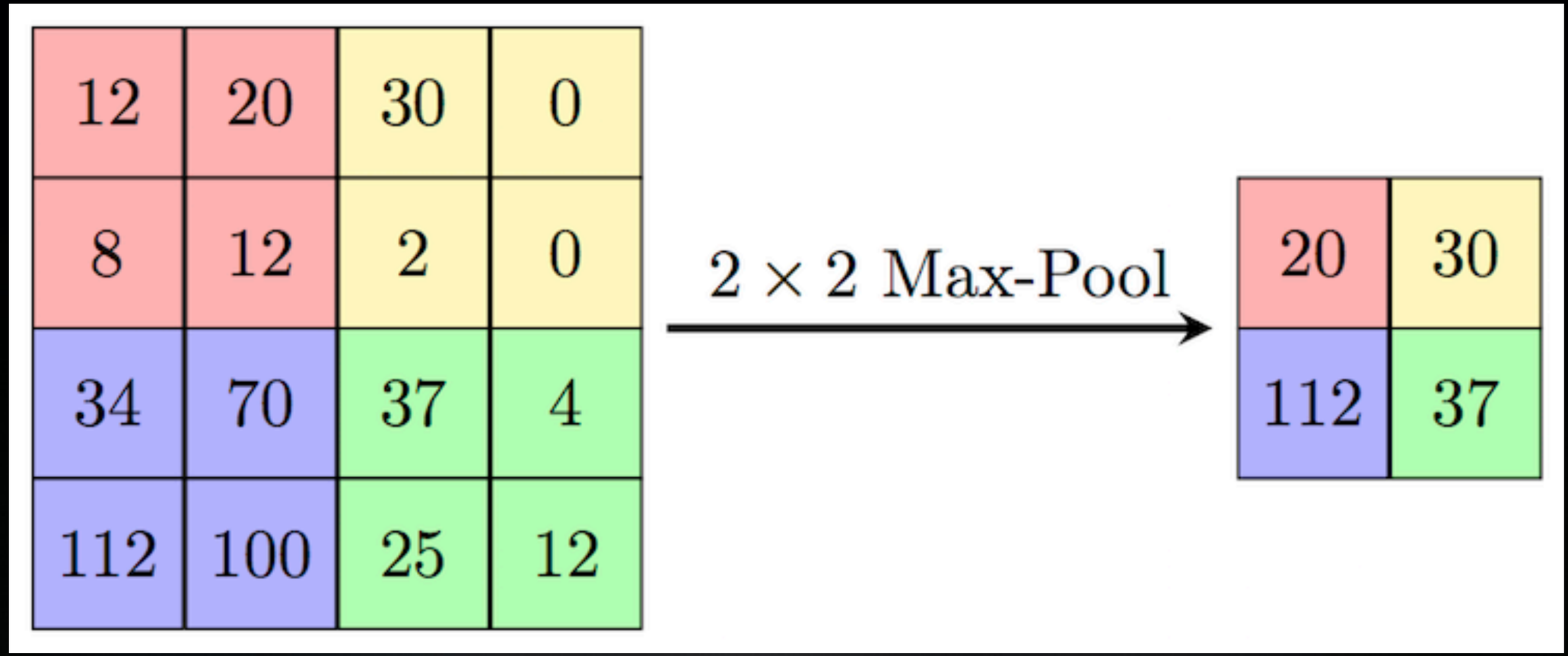
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

Image source: https://en.wikipedia.org/wiki/Sobel_operator

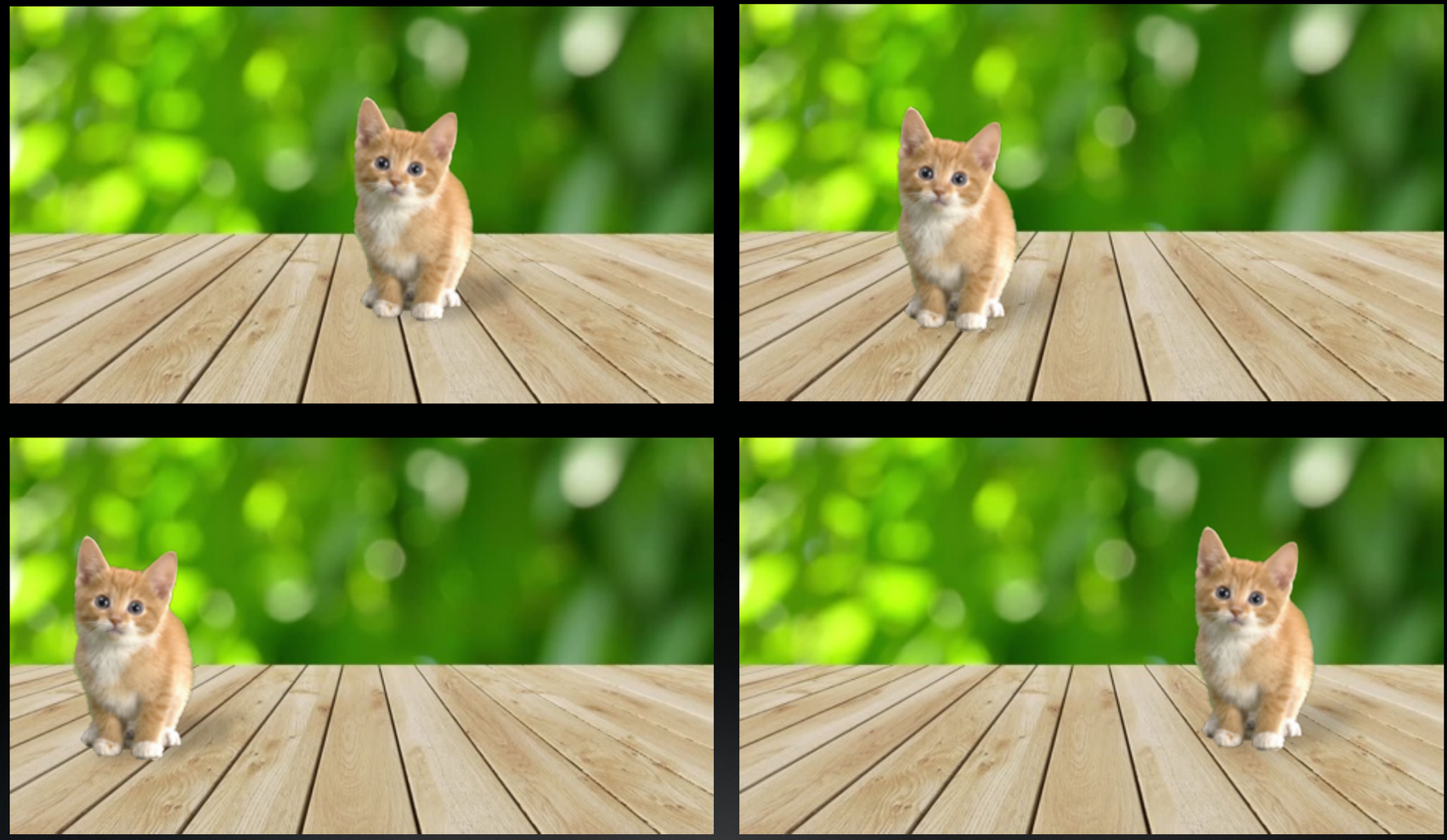
MAXPOOL-2D

Pooling reduces the size of the image, so we can detect large-scale features



TRANSLATIONAL INVARIANCE

Objects in nature look the same from place to place



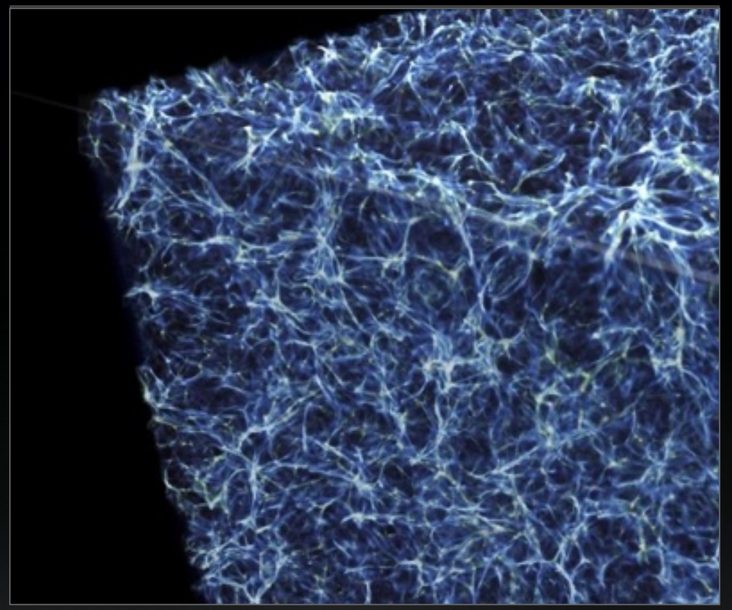
WHAT ARE CNNS USED FOR?

Problems with translational invariance in variance in time and/or space

CONV_2D

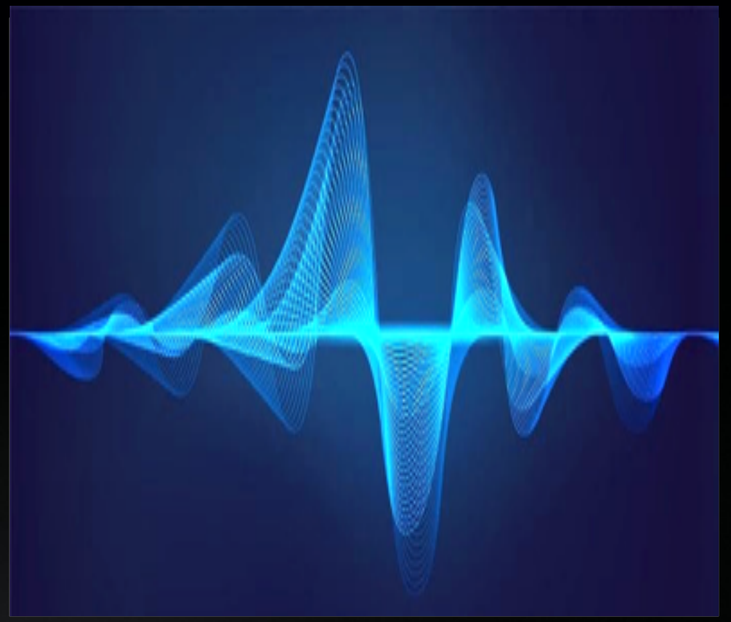
Computer Vision
Invariance in 2d space

CONV_3D



Computational Physics
Invariance in 3d space

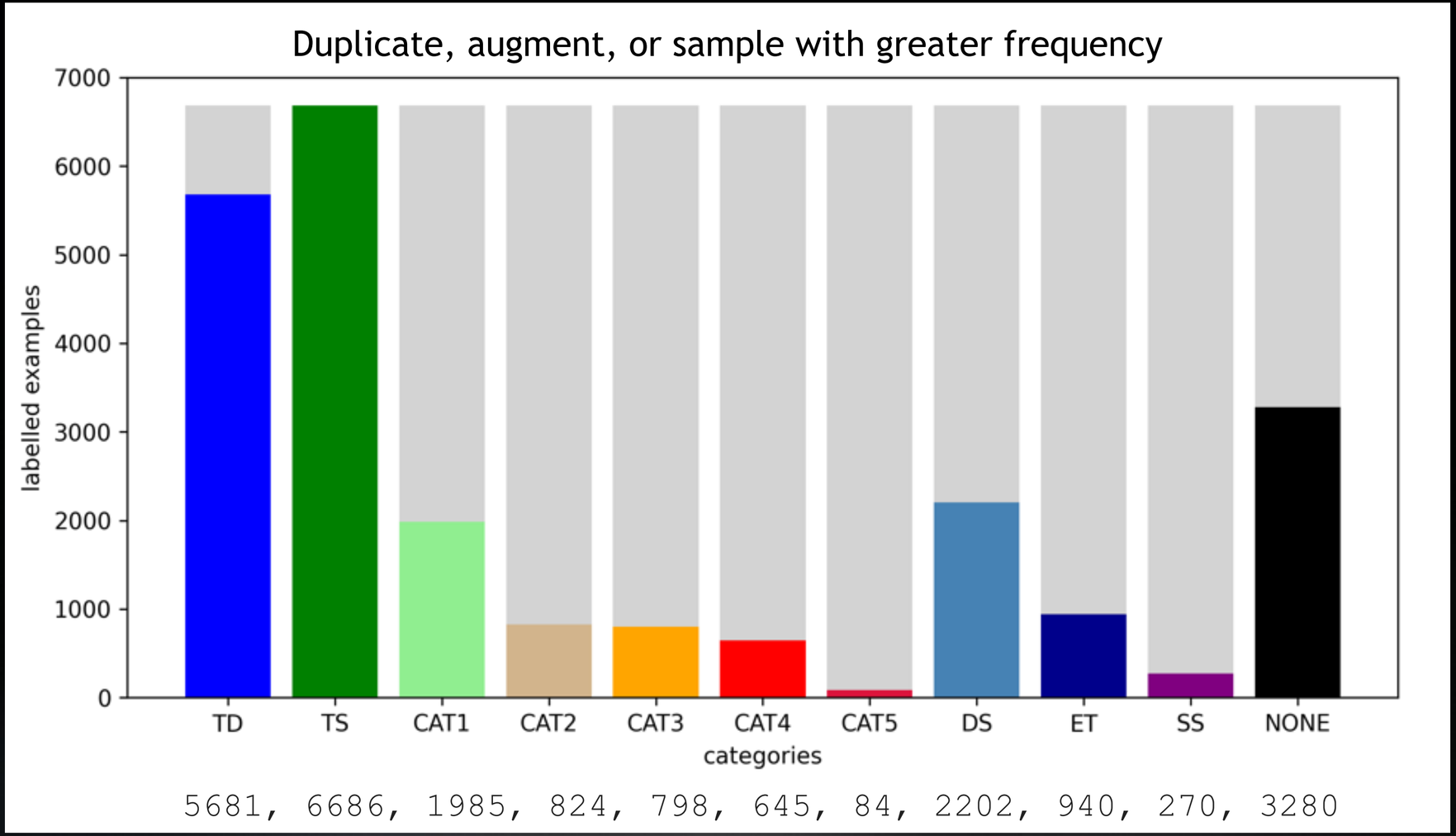
CONV_1D



Audio and Time Series
Invariance in time

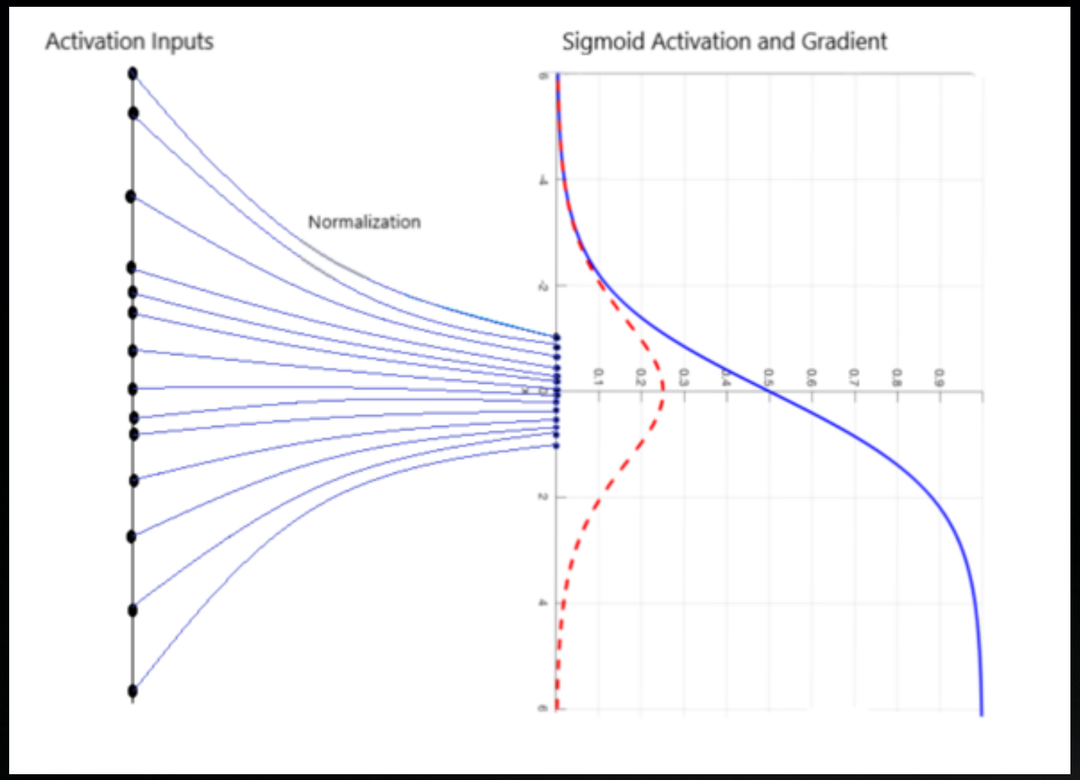
IMBALANCED DATA

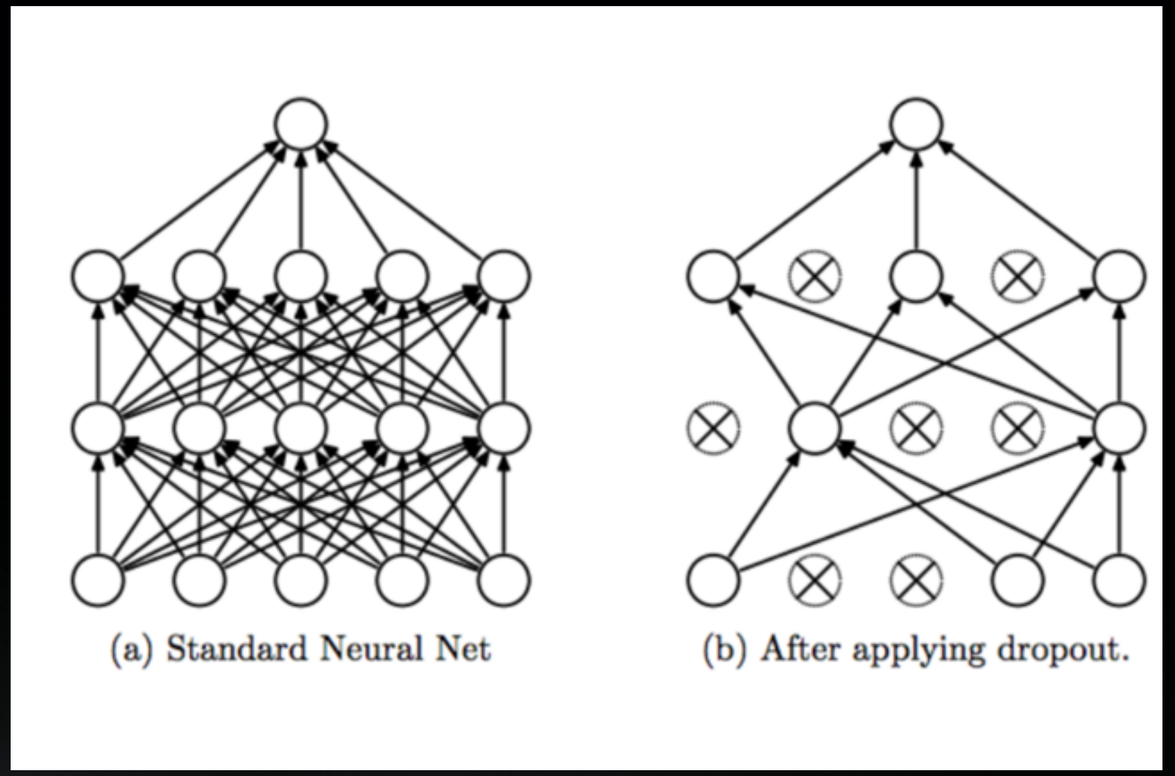
Some features occur more frequently than others



REGULARIZATION

BatchNorm and Dropout

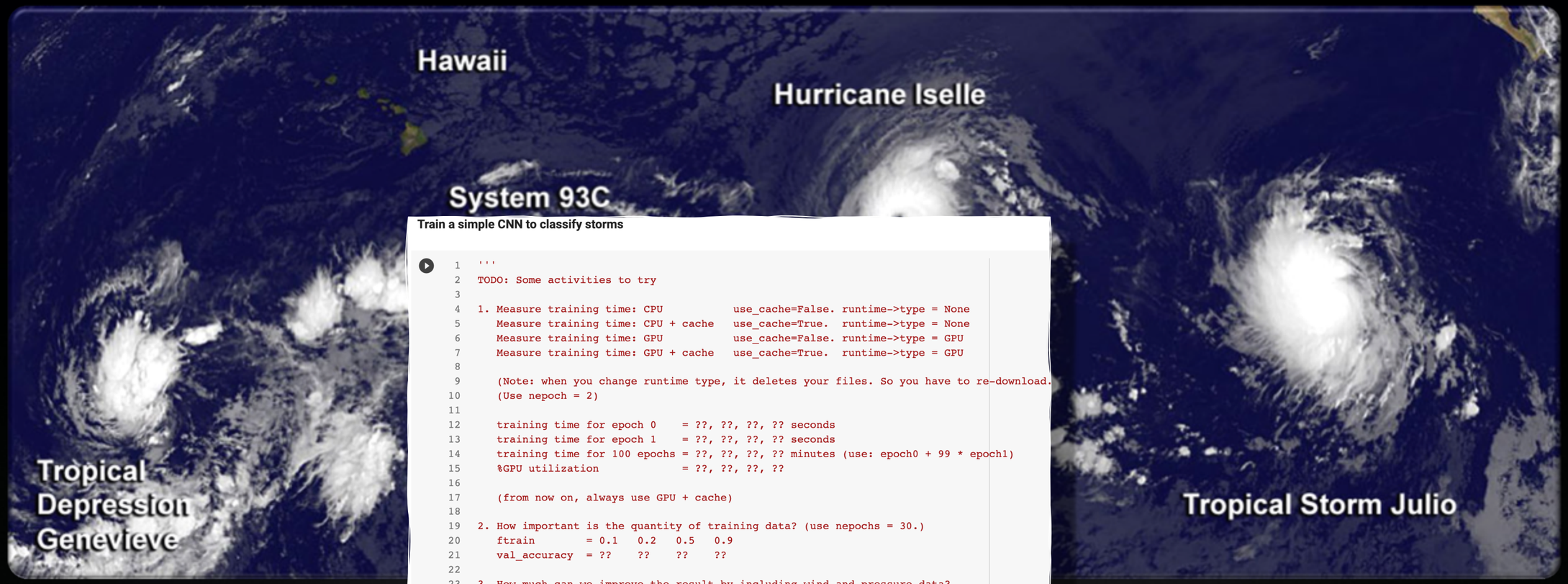




https://www.kdnuggets.com/2018/09/dropout-convolutional-networks.html

ACTIVITY: CLASSYING TROPICAL CYCLONES

[Click here to launch the notebook]



Train a simple CNN to classify storms

```
'''
TODO: Some activities to try

1. Measure training time: CPU           use_cache=False. runtime->type = None
   Measure training time: CPU + cache   use_cache=True.  runtime->type = None
   Measure training time: GPU           use_cache=False. runtime->type = GPU
   Measure training time: GPU + cache   use_cache=True.  runtime->type = GPU

   (Note: when you change runtime type, it deletes your files. So you have to re-download.
   (Use nepoch = 2)

   training time for epoch 0    = ??, ??, ??, ?? seconds
   training time for epoch 1    = ??, ??, ??, ?? seconds
   training time for 100 epochs = ??, ??, ??, ?? minutes (use: epoch0 + 99 * epoch1)
   %GPU utilization             = ??, ??, ??, ??

   (from now on, always use GPU + cache)

2. How important is the quantity of training data? (use nepochs = 30.)
   ftrain        = 0.1   0.2   0.5   0.9
   val_accuracy  = ??    ??    ??    ??

3. How much can we improve the result by including wind and pressure data?
   fields = ["pwat","u","v","mslp"]
   val_accuracy = ??

4. How well can we classify  10 categories, using water vapor alone? fields=['pwat']
   Recall, a random guess should get you around 10%
   categories = ['TD','TS','CAT1','CAT2','CAT3','CAT4','CAT5','DS','SS','NONE']
   val_accuracy = ??
''';
```

CLASSIFYING TROPICAL STORMS

My results on Google Colab using a T4 GPU

Training time for 100 epochs

CPU	GPU
5000s	3400 (1.5x **~1%GPU –>** Problem is I/O bound)
+cache	**+cache**
614 (8x)	79s (63x **~55% GPU**)

Using the GPU effectively reduces training time from 1.5 hours –> 1 minute !

Accuracy vs data quantity

f =	90%	50%	20%	10%	0%
Acc=	.91	.87	.81	.78	.66

Data quantity and quality are key for high accuracy

Accuracy vs inputs

U,v,p	pwat	pwat,u,v	pwat,u,v,p	all fields
0.85	0.91	0.93	0.94	??

Can detect storms with water vapor alone.
But if you have additional information … use it!

On your own: Augment and balance your samples to increase classification accuracy

ADDITIONAL TOPICS TO EXPLORE

We've only scratched the surface. There is *much* more to learn.

Learn by doing!

Data augmentation and balancing

Regularization, dropout, batch norm, weight decay

Skip connections and Resnets

Segmentation, detection, instance segmentation

Transfer learning

Multi-gpu training, data parallel, ddp

GPU profiling and tuning using nsight

Tensor-RT for runtime optimization

Encoder-decoder, autoencoder

Anomaly detection

Generative models, gans, and vaes

Masked convolutions and inpainting

Uncertainty quantification

Explainable / trustworthy AI

Transformers

Self-supervision

Automatic hyperparameter tuning

Pytorch lightning and bolts

HPC + AI coupling

Physics informed neural nets

Deep learning with julia

Reinforcement learning

Active learning

Fast.ai

Rapids for ML

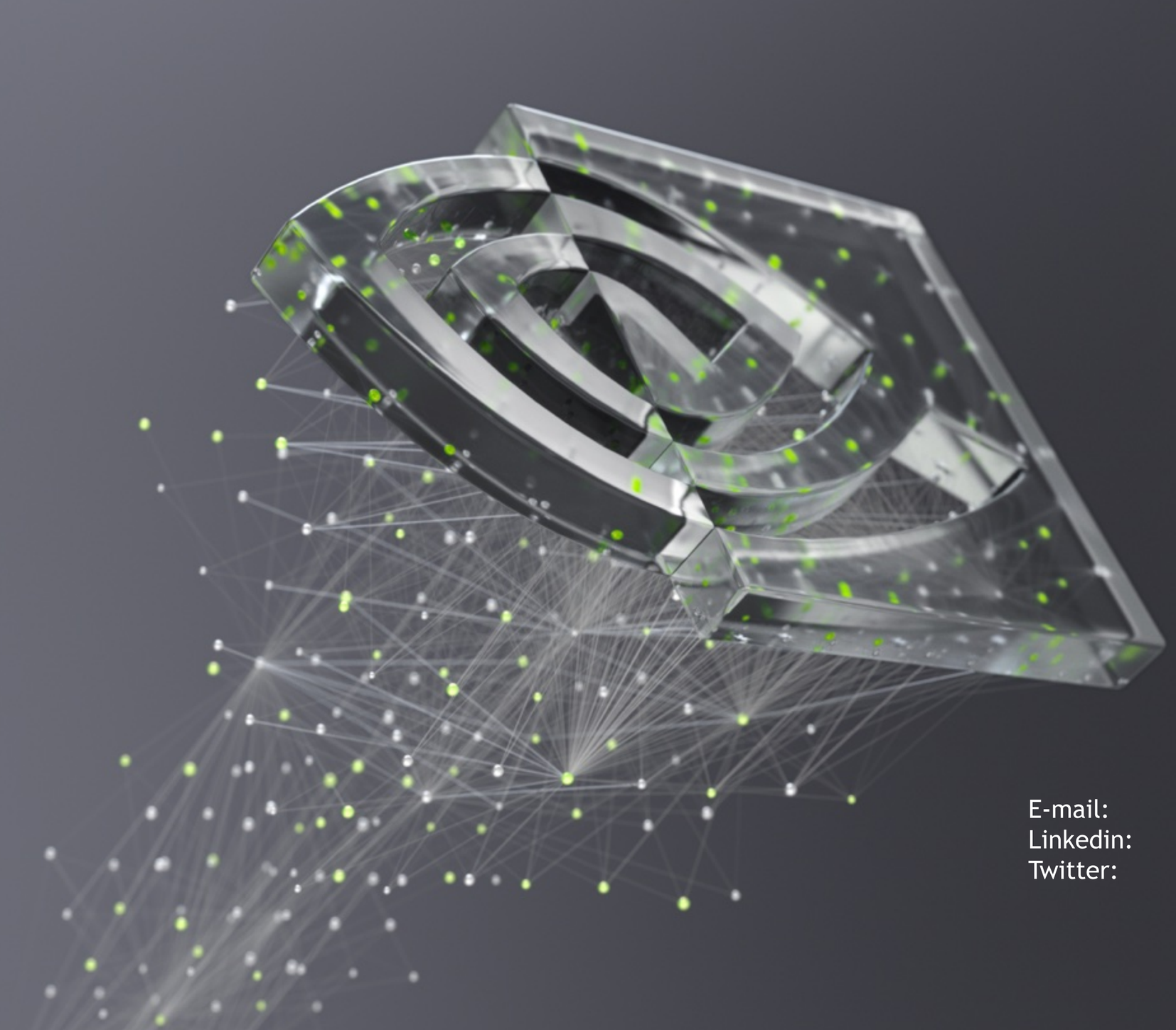
Docker, containers, and NGC

Jetson nano, edge computing

Etc.

SOME RECORDED TALKS I'VE GIVEN

• Frontiers of Deep Learning (2020)	Recent breakthroughs in AI + geoscience
• Deep Learning Architectures (2020)	Overview of major deep-learning architectures
• Machine Learning and the Future of the Earth (2020)	Can we save the planet, using AI?
• NEDTalk: AI and NOAA's Mission (2019)	How AI can be used for geosciences
• AI for Science (2018/19)	Prototype deep learning applications
• Nonhydrostatic Global Climate Modeling (2015)	Making climate models better



Summary

- Deep learning is powerful
- It's a generalization of curve fitting
- PyTorch is like GPU accelerated numpy
- Most popular DL framework for research
- GPUs are necessary for real-world apps
- We can use DL to build next-level apps
- There's a ton left to learn

Please click here to help us improve the tutorial ! (Two-minute survey)

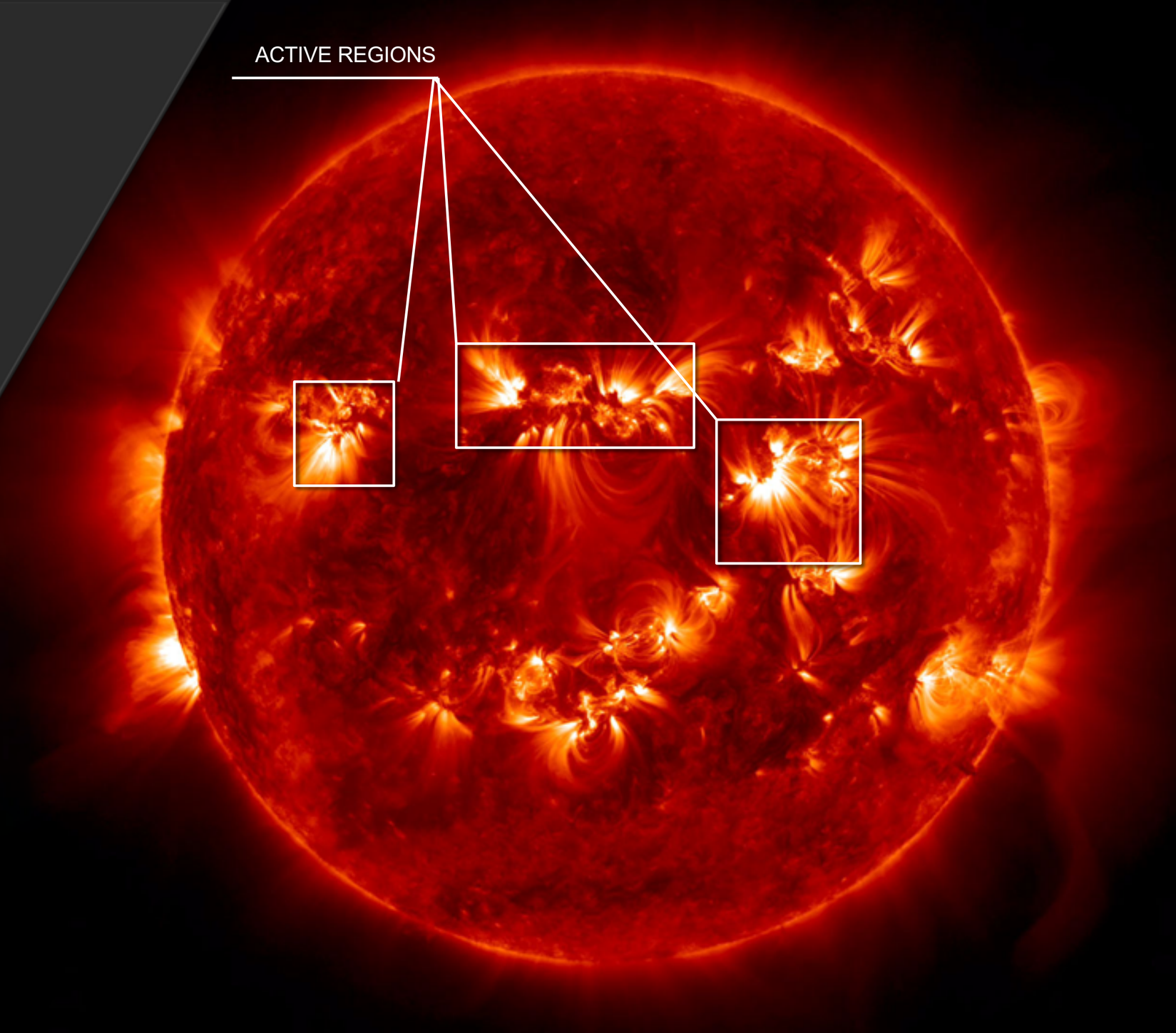
E-mail: dhall@nvidia.com
Linkedin: david-matthew-hall
Twitter: @halldm2000

NVIDIA

HELIOPHYSICS APPLICATIONS

NASA GODDARD
ALTAMIRA & NVIDIA

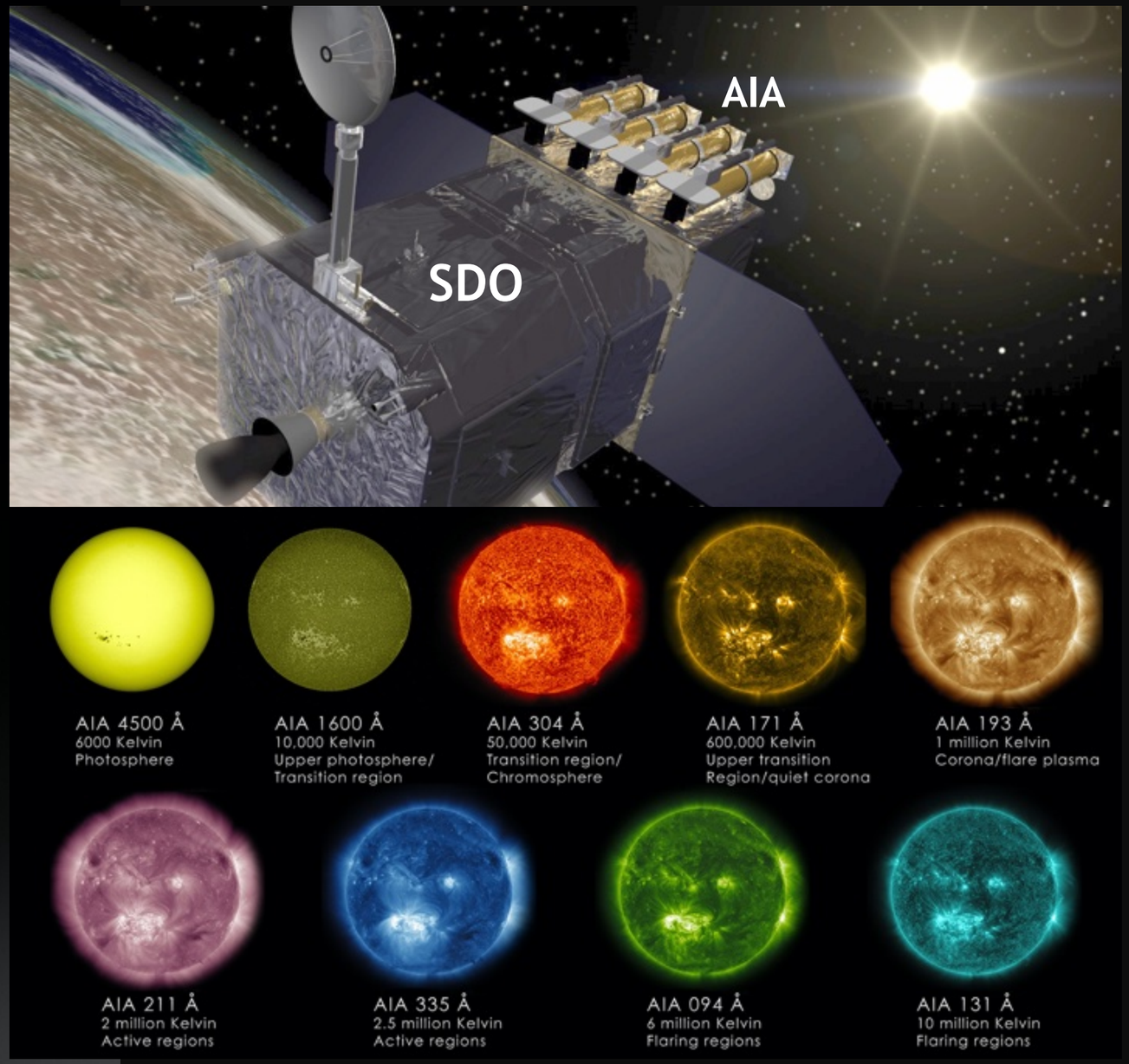
Feature detection can be applied to detect features on the Sun and other astrophysical bodies. In particular, we can apply AI to solar flares and coronal mass ejections in order to predict the influx of highly charged particles on Earth's atmosphere.



SOLAR DYNAMICS OBSERVATORY

- 1.5 TB Data / Day
- Operational Since 2010
- AIA: 10 Wavelength Channels
- 150M Images To Be Labelled
- 30k Images Labelled so far

- Coronal Holes
- Active Regions
- Sunspots
- Solar Flares
- Coronal Mass Ejections
- Filaments



RESULTS:
CORONAL HOLES

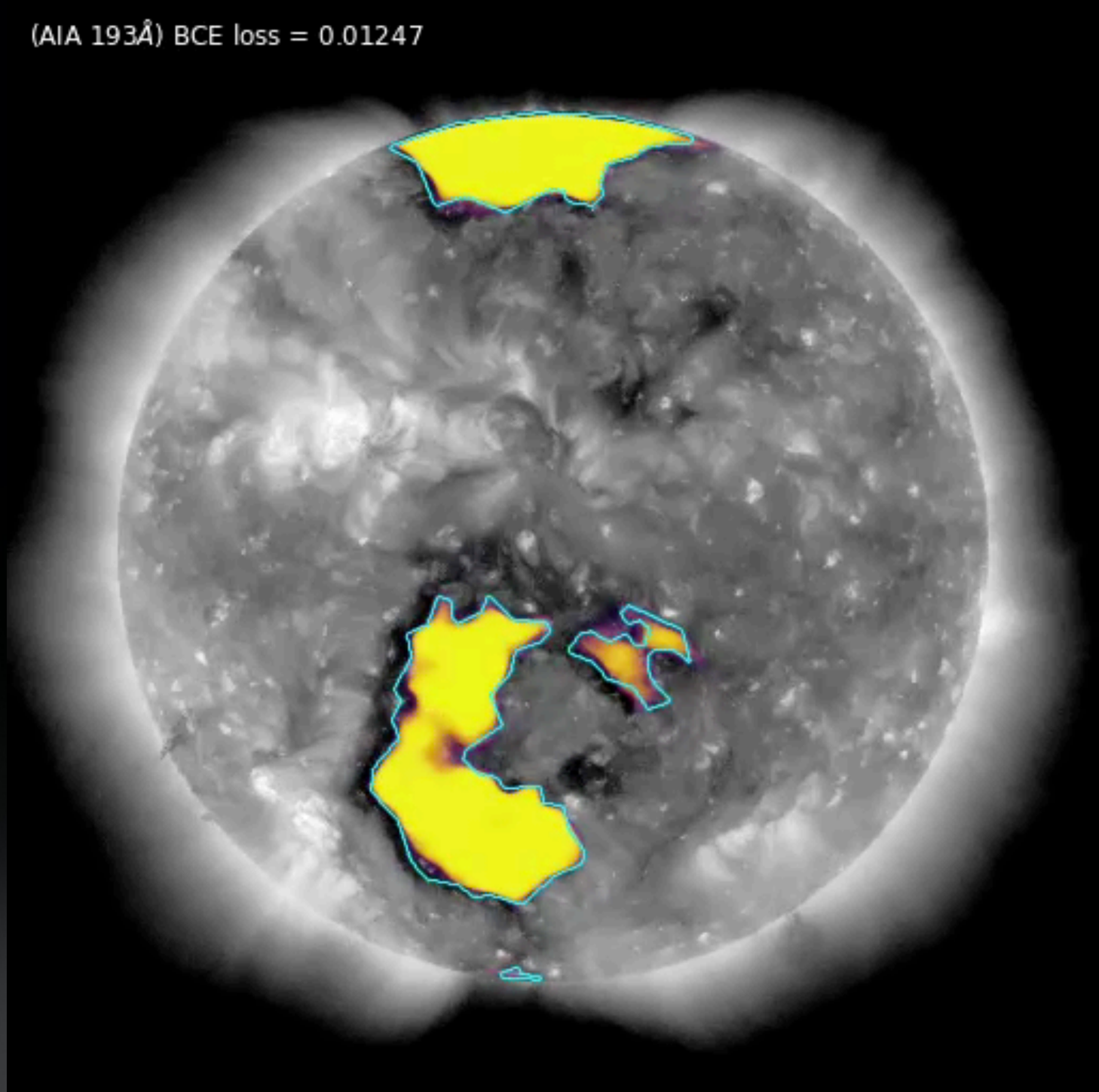
NASA Goddard
Michale Kirk, Barbara Thompson,
Jack Ireland, Raphael Attie

NVIDIA
David Hall

Altamira
Matt Penn, James Stockton,

SOURCE
Solar Dynamics Observatory
AIA Imager

TARGET
Hand-crafted detection algorithm



SUNSPOT PREDICTIONS

Highly imbalanced dataset. Needs special care.

Predicts all 0s unless special care is taken

- Super-sample minority class
- Under-sample majority class
- Use focal loss

Select small crops from high-res imagery

Pos : crops w/large fraction sunspot pixels

Neg : randomly selected crops

Train conv net on small crops only

Predict on full-resolution images

RESULTS: SUNSPOTS

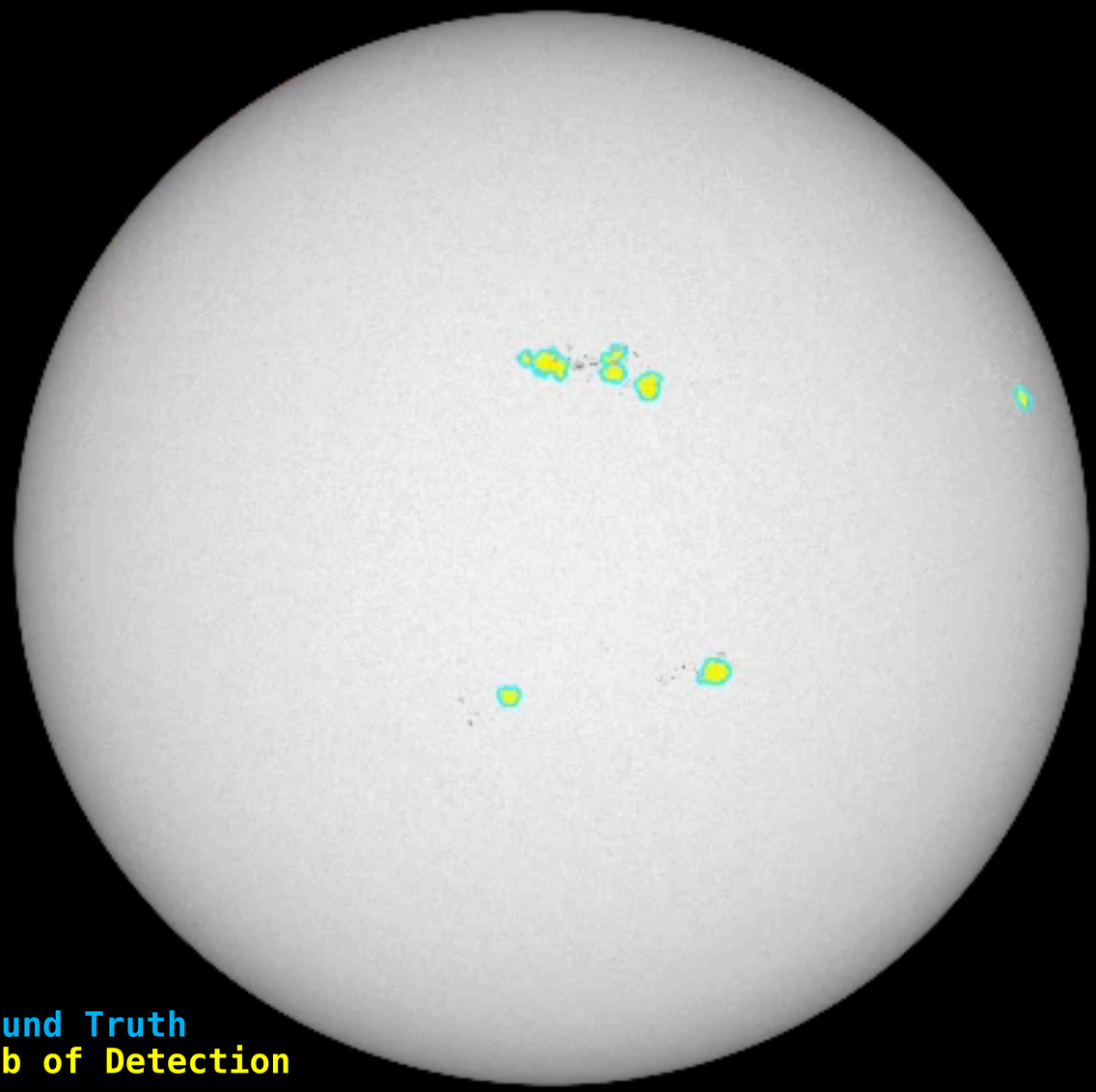
NASA Goddard
Michale Kirk, Barbara Thompson,
Jack Ireland, Raphael Attie

NVIDIA
David Hall

Altamira
Matt Penn, James Stockton,

SOURCE
Solar Dynamics Observatory
AIA Imager

TARGET
Hand-crafted detection algorithm



RESULTS:
ACTIVE REGIONS

NASA Goddard
Michale Kirk, Barbara Thompson,
Jack Ireland, Raphael Attie

NVIDIA
David Hall

Altamira
Matt Penn, James Stockton,

SOURCE
Solar Dynamics Observatory
AIA Imager

TARGET
Hand-crafted detection algorithm