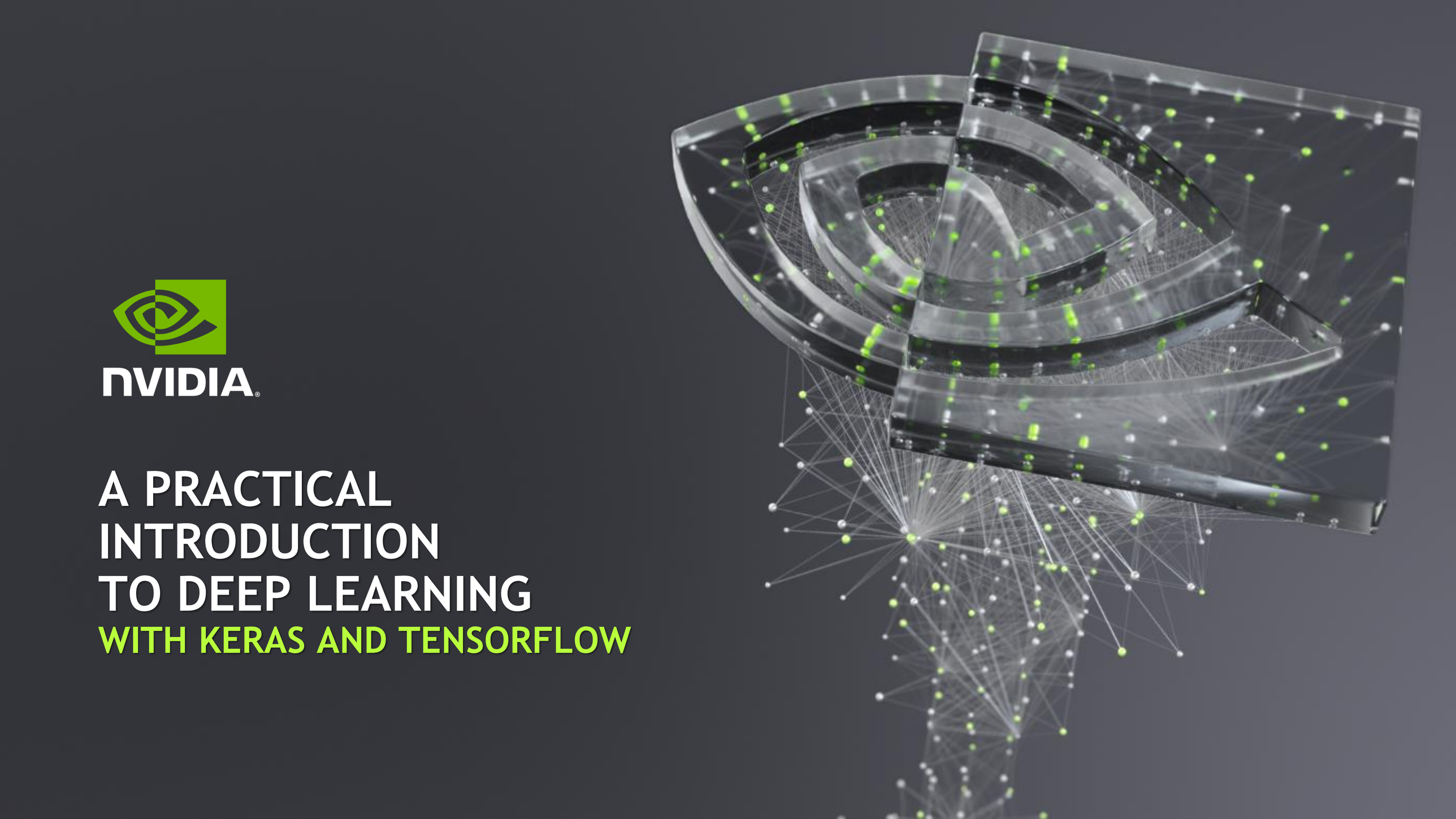


A PRACTICAL INTRODUCTION TO DEEP LEARNING

WITH KERAS AND TENSORFLOW

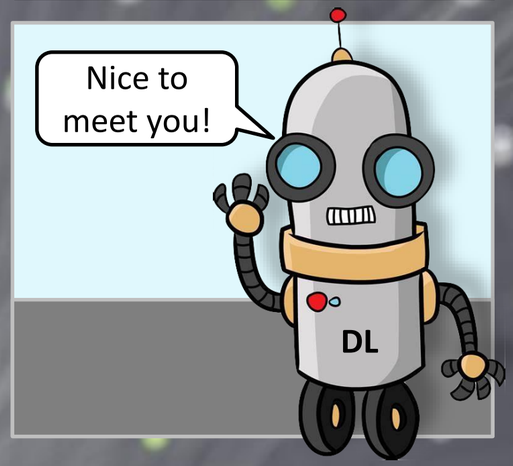
LEARNING GOALS

In couple of hours, we can only travel so far

Main goal:
Become familiar with main ideas and process

A starting point for solving your own problems

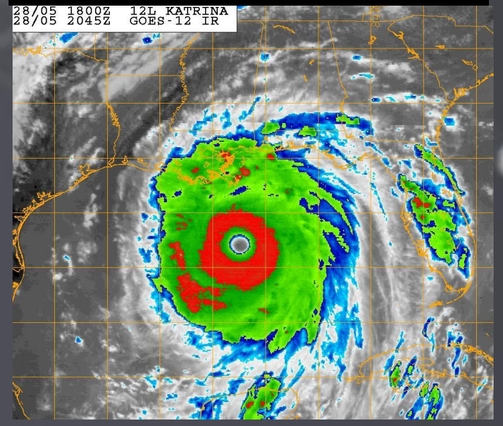
INTRO TO DL, PART 1



LAB 1: CNNs AND KERAS 101

1:50-2:30 ET

LAB 2: TROPICAL CYCLONES

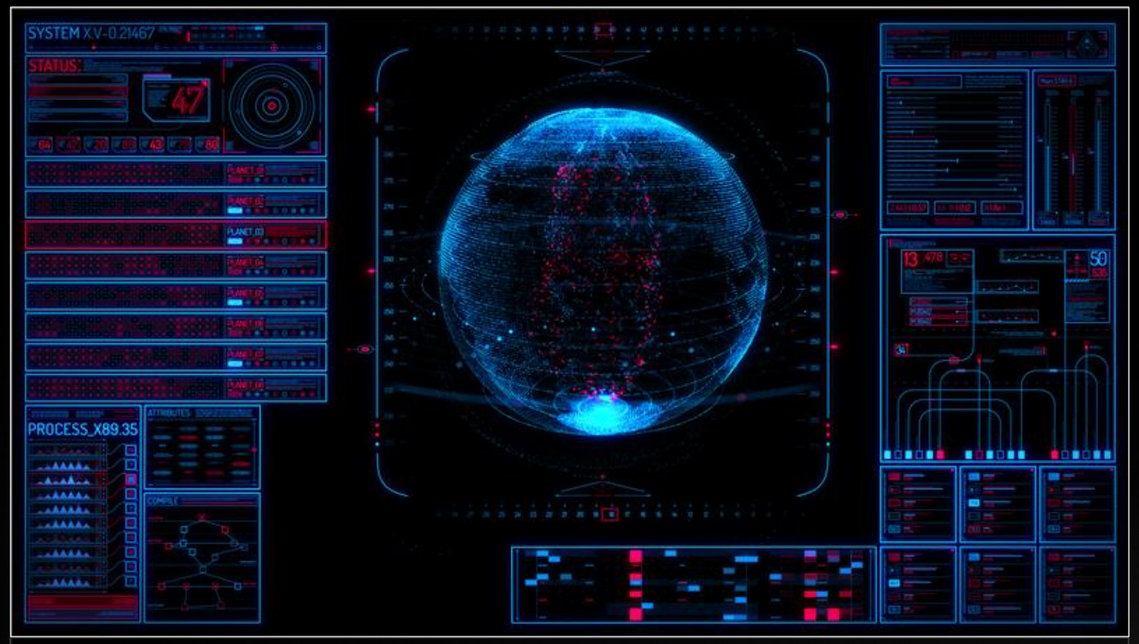


AGENDA

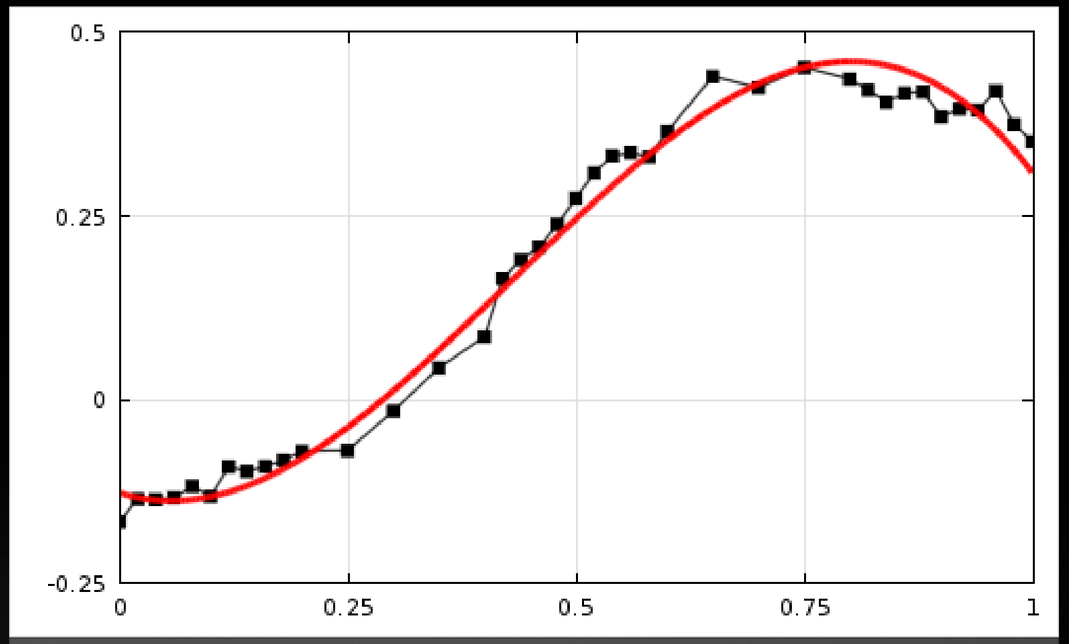
DEEP LEARNING ANALOGIES

What is this deep learning thing, anyway?

A NEW TYPE OF SOFTWARE



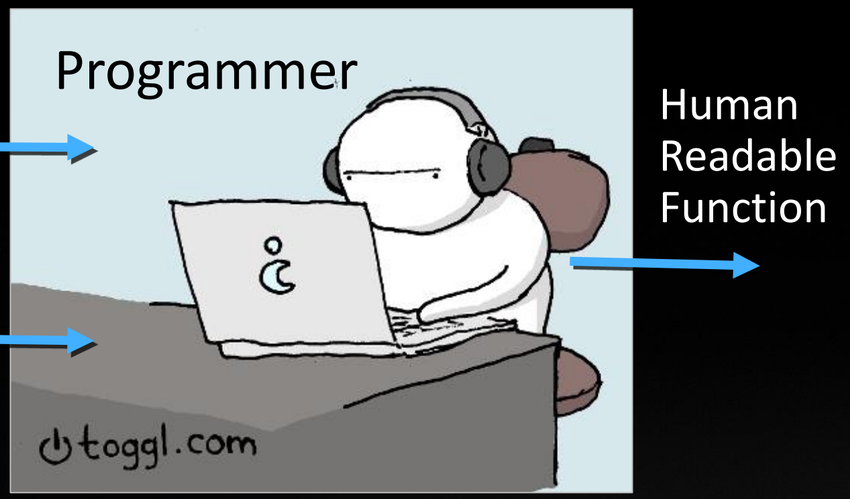
A GENERALIZATION OF CURVE FITTING



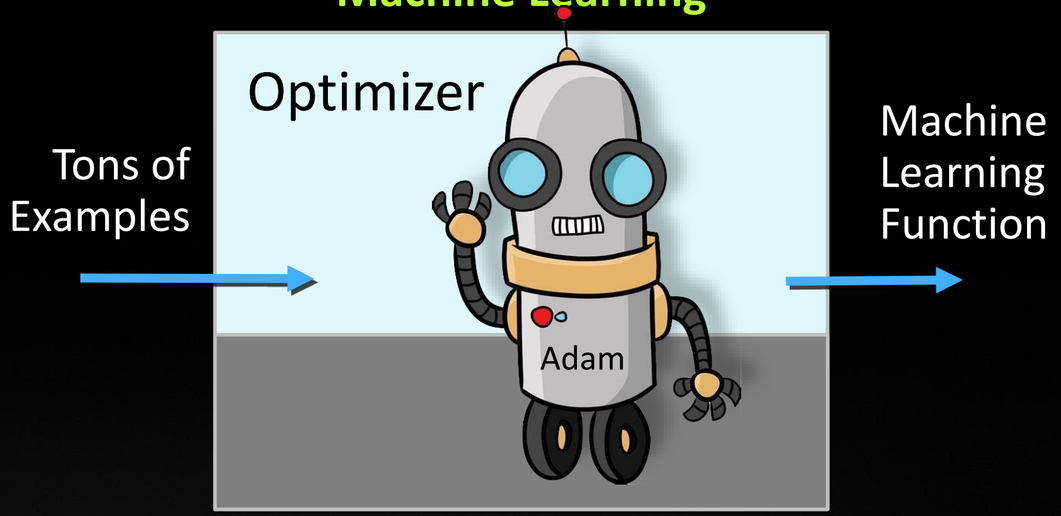
A NEW WAY TO BUILD SOFTWARE

Traditional Programming vs Machine Learning

SOFTWARE 1.0:
Traditional Programming



SOFTWARE 2.0:
Machine Learning



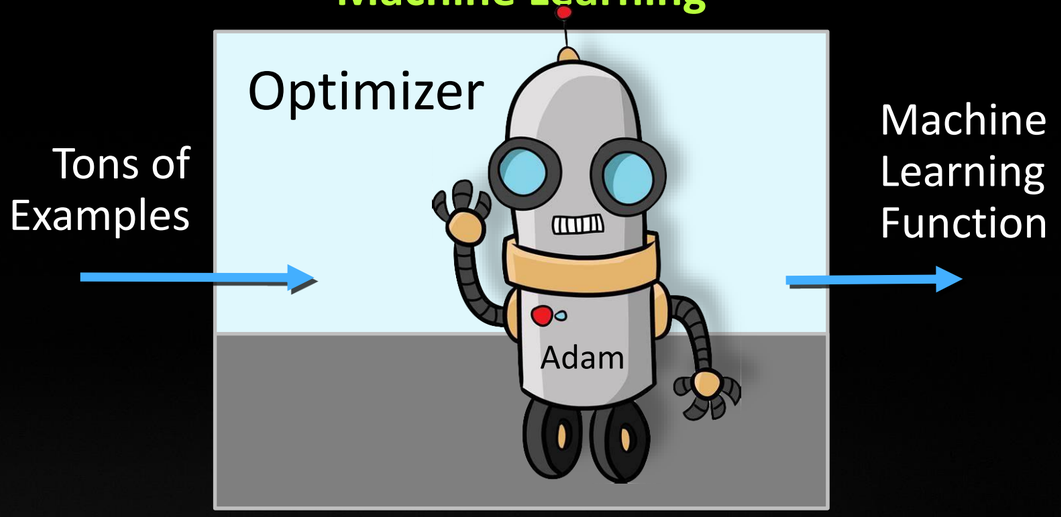
A DIFFERENT WAY TO BUILD SOFTWARE

Traditional Programming vs Machine Learning

Goals for today:

1. **Learn to use this new approach**
2. **Revolutionize Science**

SOFTWARE 2.0:
Machine Learning



A DIFFERENT WAY TO BUILD SOFTWARE

Hand written vs leanred functions.

TEMP, PRESSURE, MOISTURE

PROBABILITY OF RAIN

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

A DIFFERENT WAY TO BUILD SOFTWARE

The two approaches are complimentary

MANUAL PROGRAMMING

“SOFTWARE 1.0”
ENGINEERED
LABOR INTENSIVE
EXPLICIT
EXPLAINABLE
SIMPLE
FROM EXPERTISE

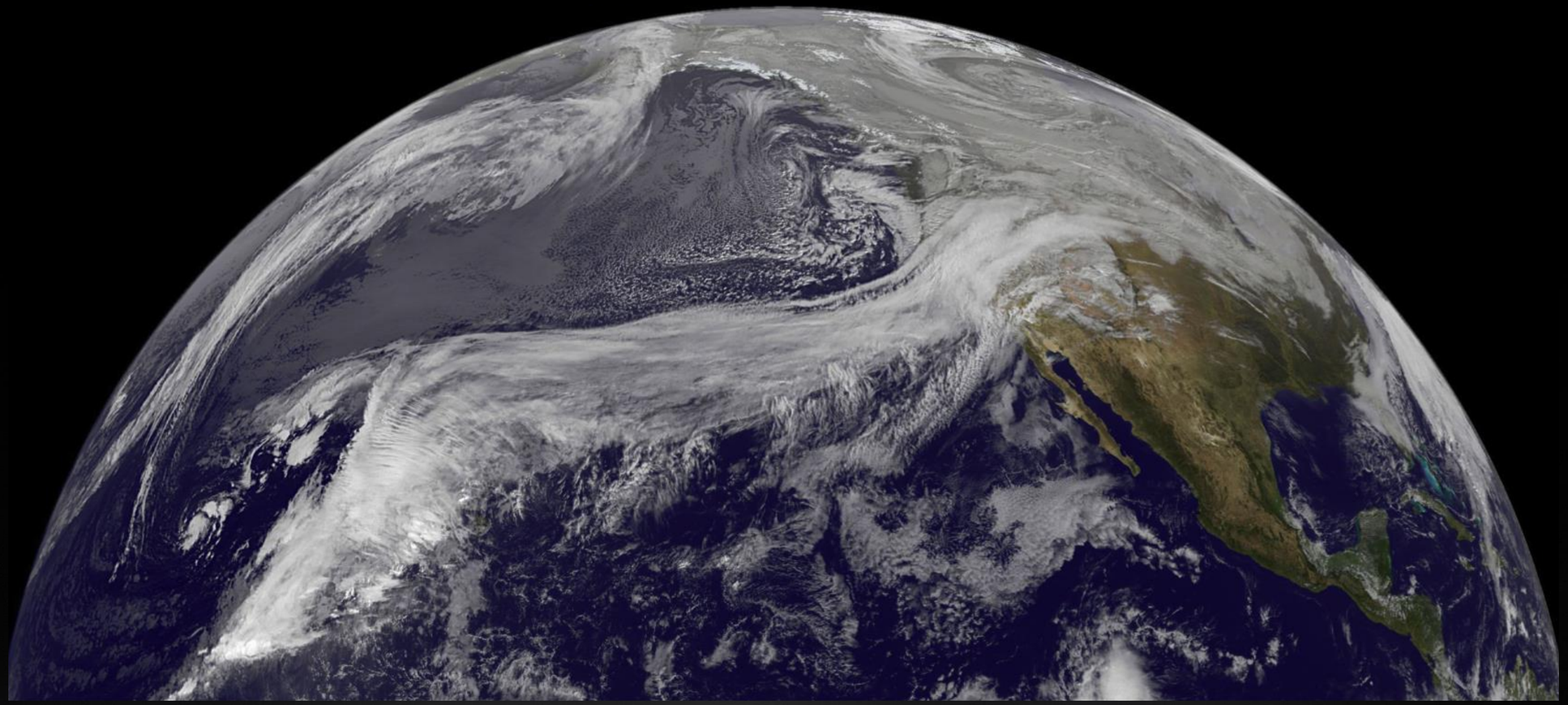
MACHINE LEARNING

“SOFTWARE 2.0”
REVERSE-ENGINEERED
AUTOMATIC
IMPLICIT
SUBTLE
COMPLEX
FROM EXAMPLES

(For best results, combine as needed)

A DIFFERENT WAY TO BUILD SOFTWARE

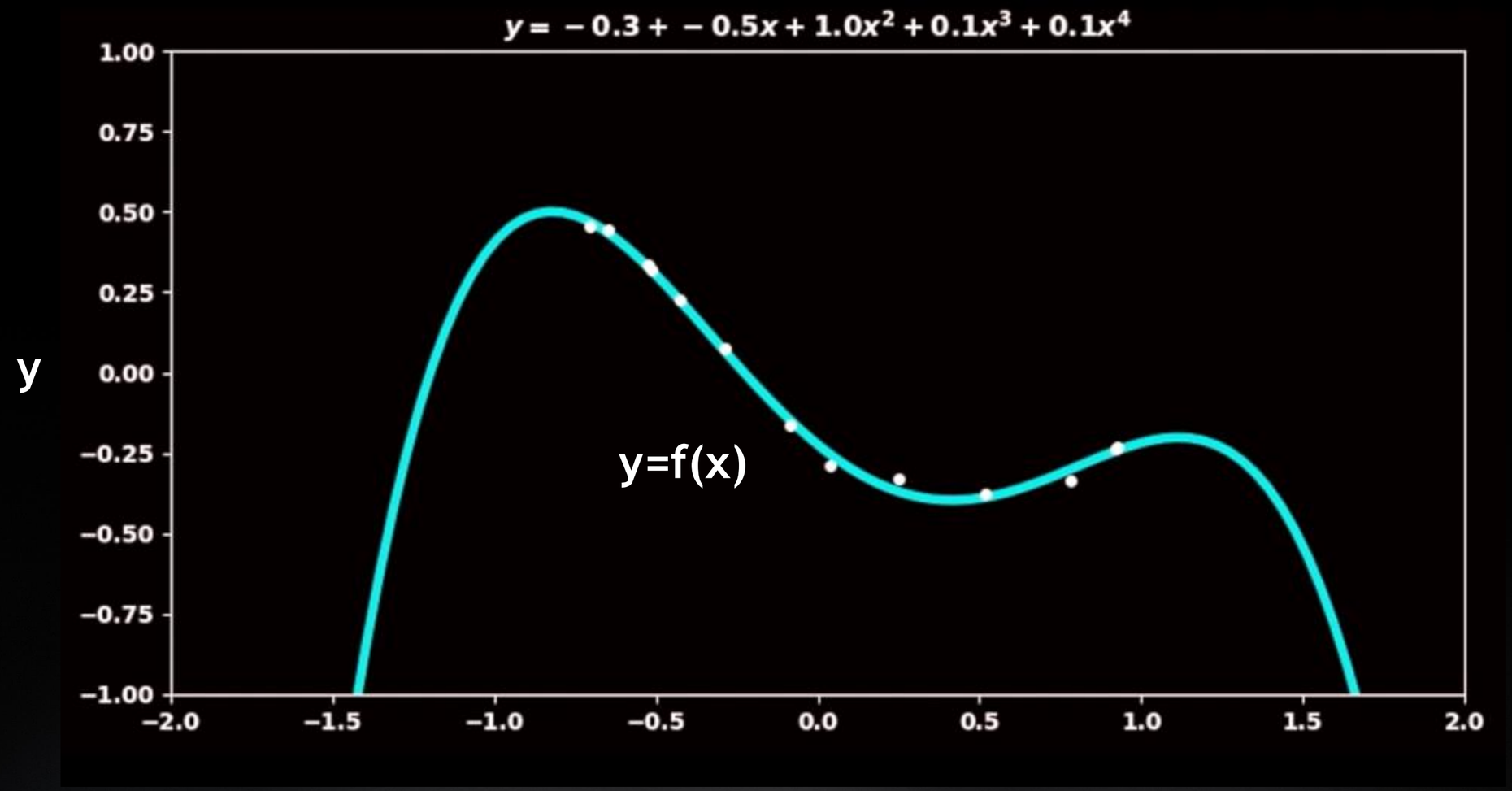
Complex phenomena are best described implicitly.



EXAMPLE: ATMOSPHERIC RIVER

A GENERALIZATION OF CURVE FITTING

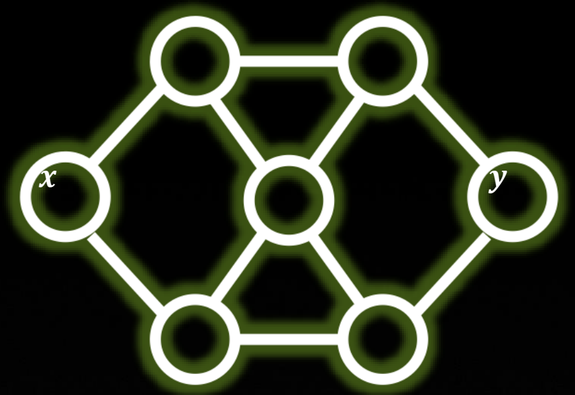
Curve fitting provides the starting intution



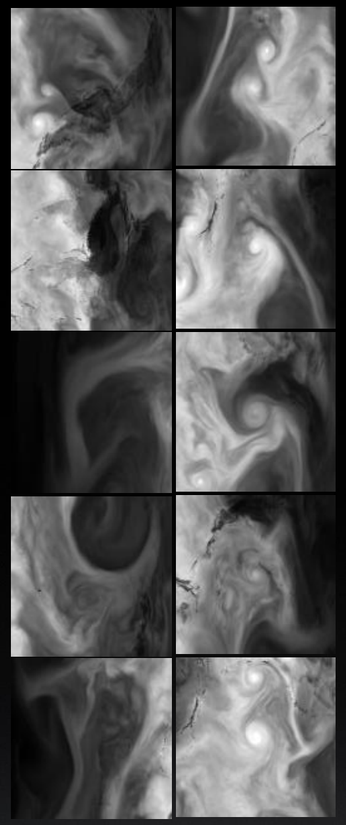
A GENERALIZATION OF CURVE FITTING

Differences from traditional curve fitting

Find f, given x and y



Supervised Deep Learning



inputs

x_1
x_2
x_3
x_4
x_5
x_6

$f(x)$

High dimensional x,y
Hierarchical
Millions of parameters

outputs

y_1
y_2
y_3
y_4
y_5
y_6

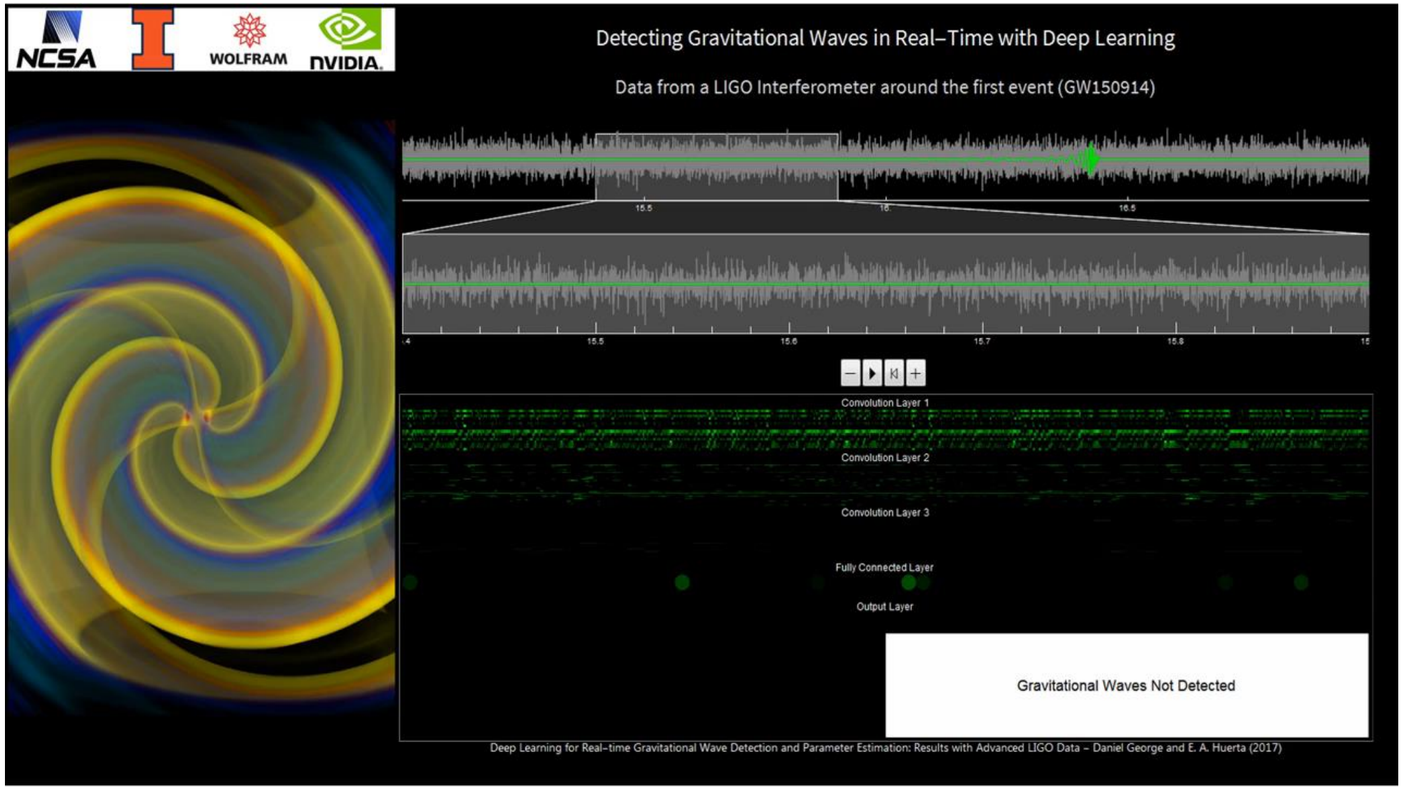
0	1
0	1
0	1
0	1
0	1

EXAMPLES

RECOGNITION/CLASSIFICATION -> FILTER

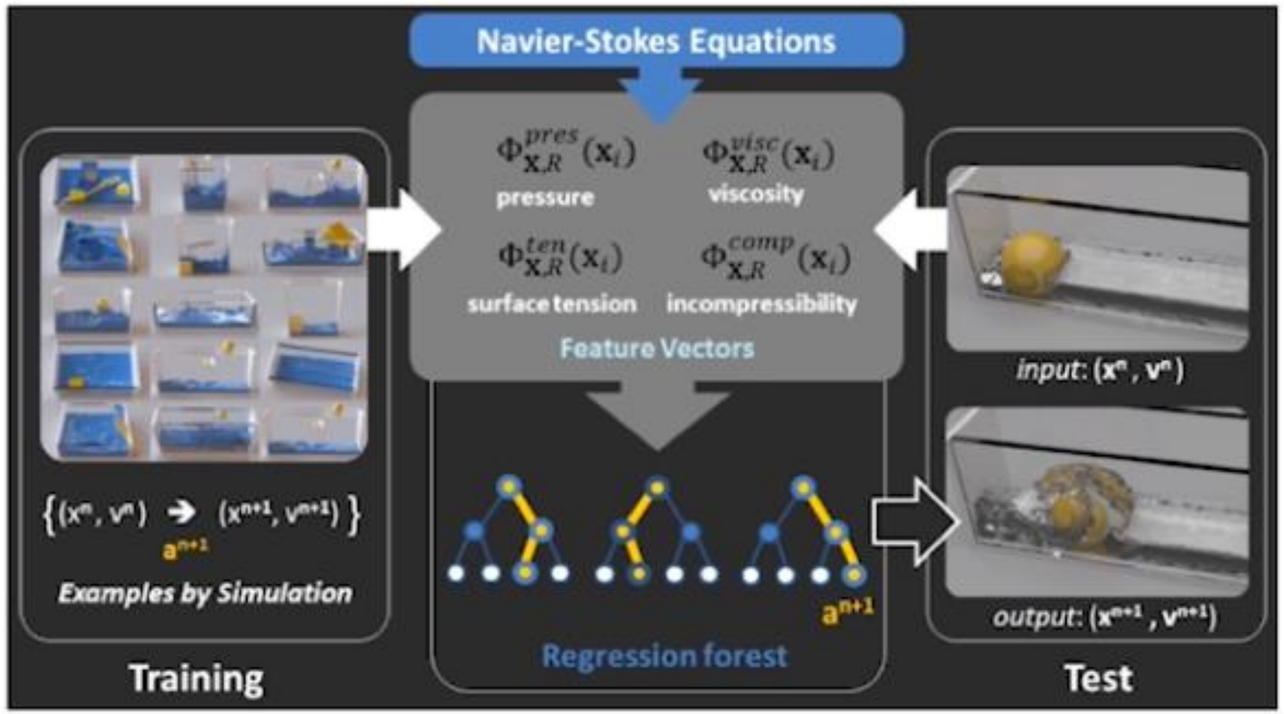
De-noising gravitational waves

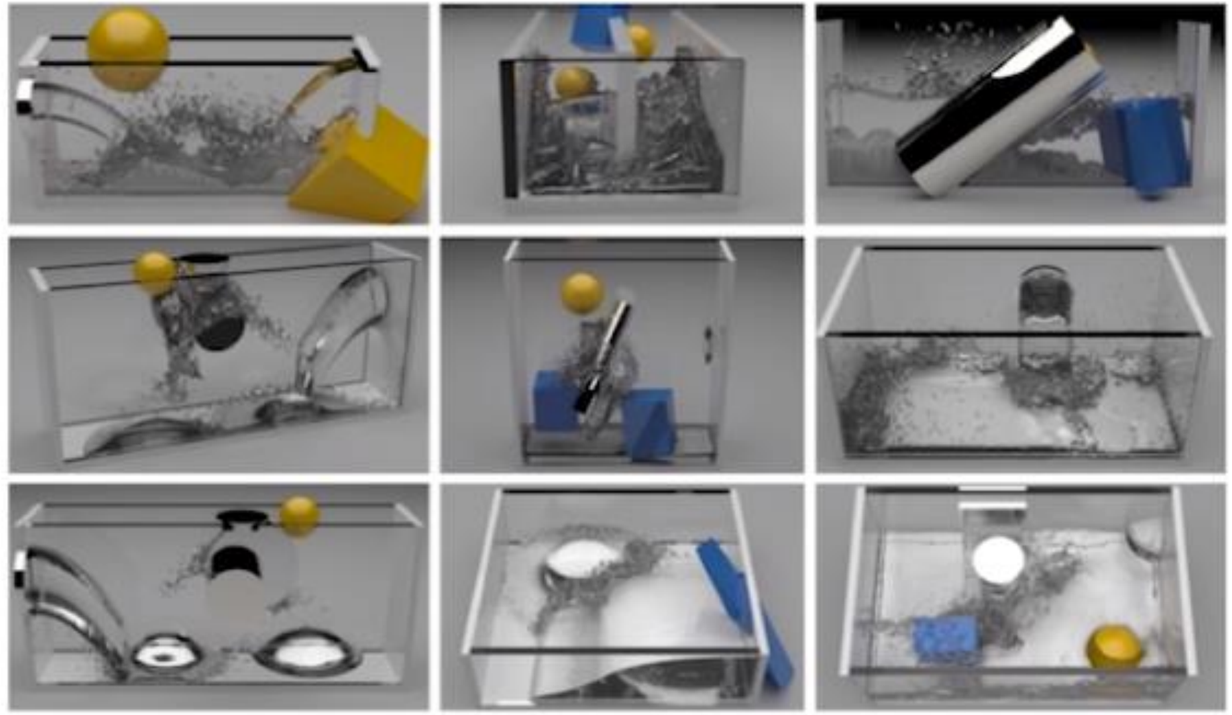
Laser Interferometer Gravitational-wave Observatory (LIGO).



DL enabling 5000x faster filtering for real-time multi-messenger astronomy

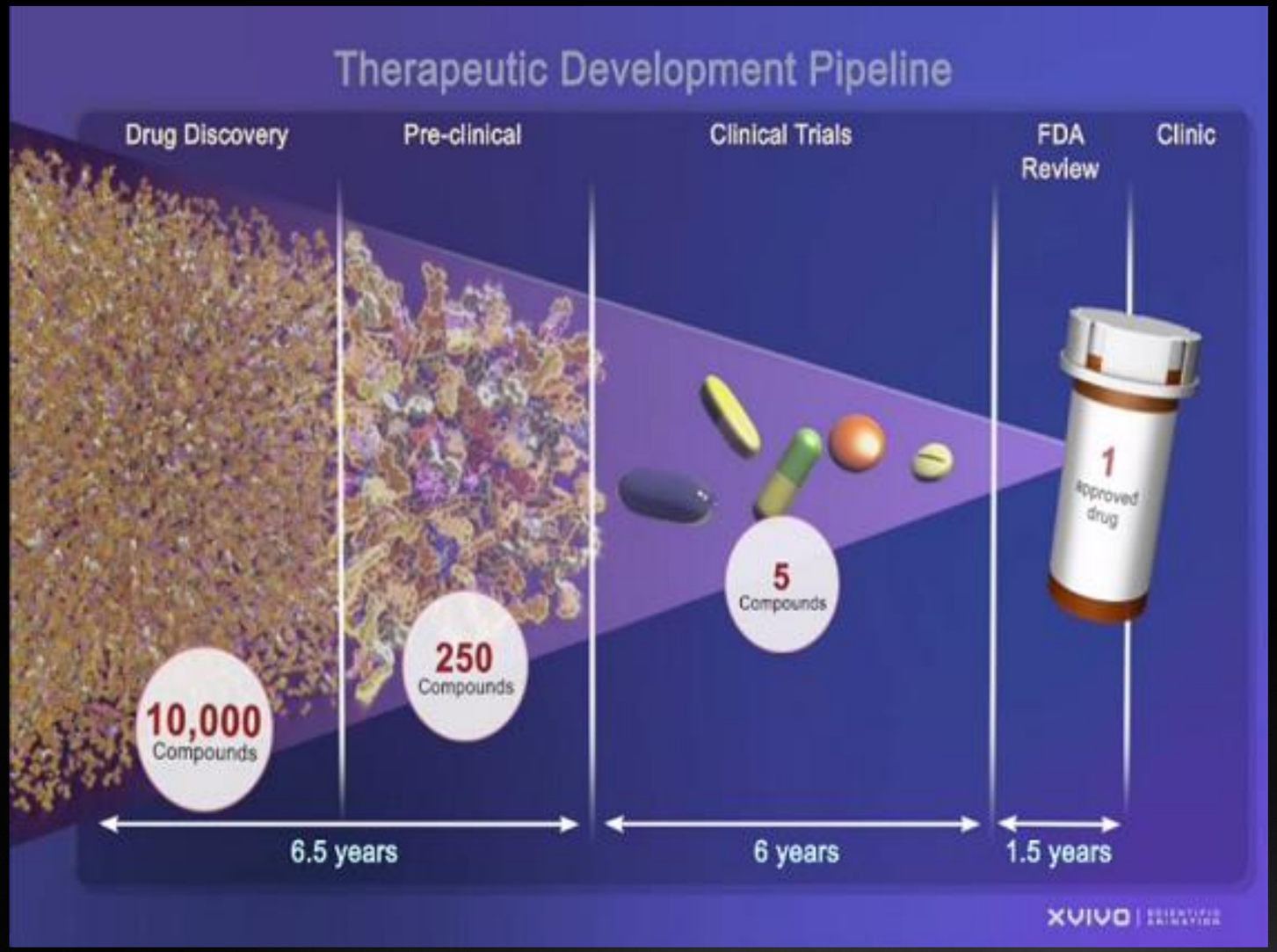
Data-driven Fluid Simulations using Regression Forests





NVIDIA.

CONVERGED HPC REVOLUTIONIZING DRUG DISCOVERY



Background

It takes 14 years and $2.5 Billon to develop 1 drug

Higher than 99.5% failure rate after the drug discovery phase

Challenge

QC simulation is computationally expensive - it takes 5 years to compute on CPUs

So researchers use approximations, compromising on accuracy. To screen 10M drug candidates,.

Solution

Researchers at the University of Florida and the University of North Carolina leveraged GPU deep learning to develop a custom framework ANAKIN-ME, to reproduce molecular energy surfaces with super speed (microseconds versus several minutes), extremely high (DFT) accuracy, and at up to 6 orders of magnitude improvement in speed.

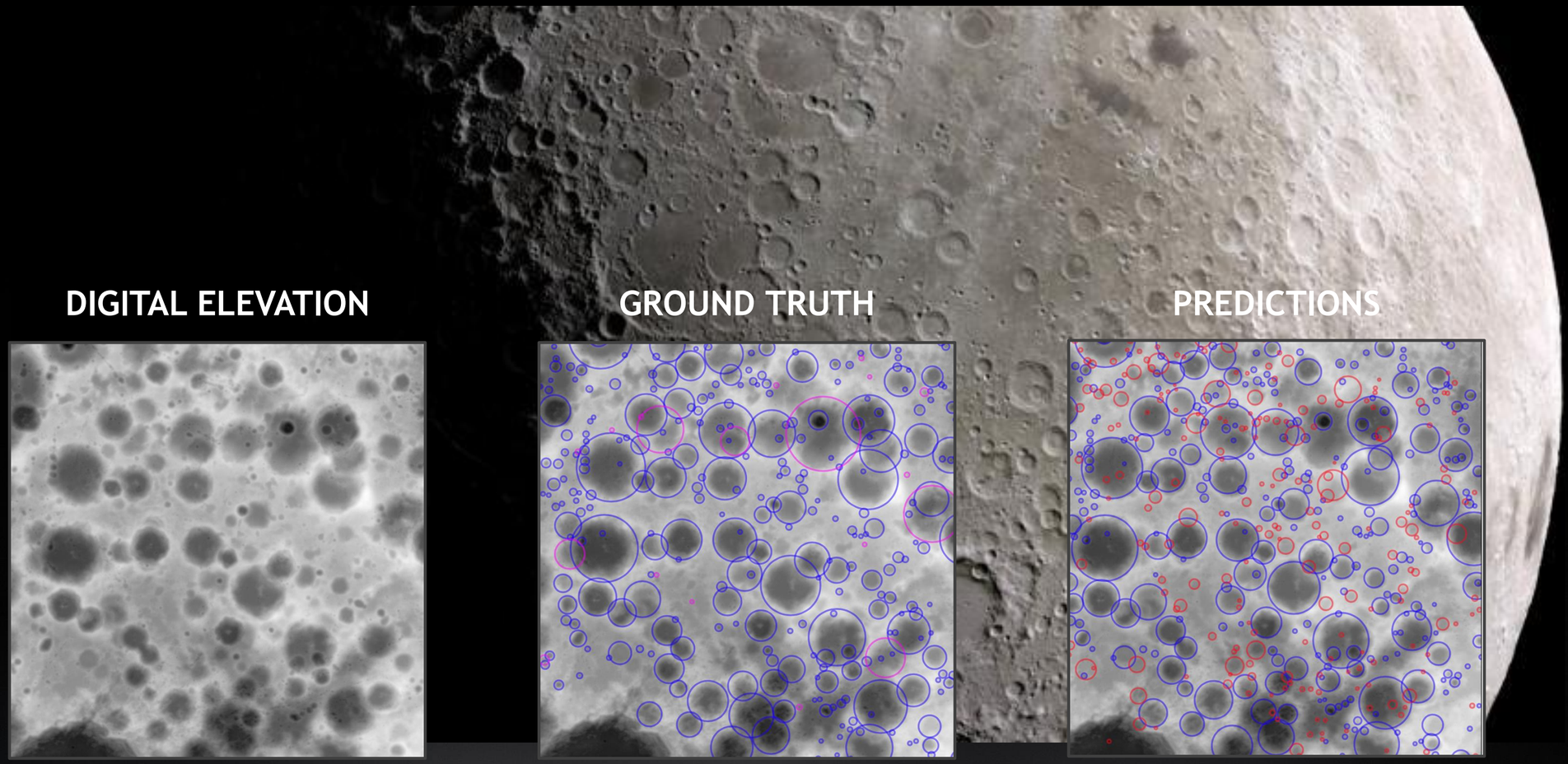
Impact

Speed and accuracy could start a revolution in computational chemistry — and forever change the way we discover the medicines of the future

UF UNIVERSITY of FLORIDA

EXAMPLE APPLICATIONS

2018: LUNAR CRATER IDENTIFICATION VIA DEEP LEARNING



https://arxiv.org/pdf/1803.02192.pdf
https://phys.org/news/2018-03-technique-ai-craters-moon.html

IMPLEMENTATION BASICS

AUTO-ML

Eventually, the optimizer might be able to do everything for you

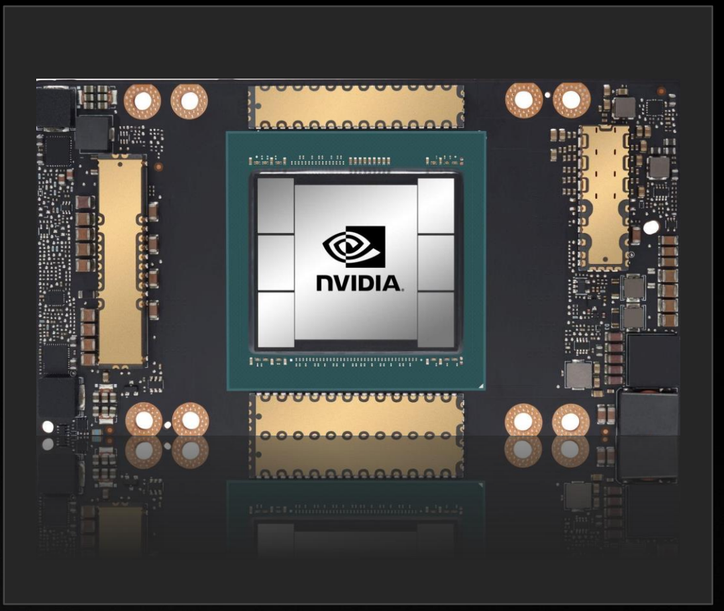
WHAT YOU NEED TO MAKE DEEP LEARNING WORK

You need three main ingredients (and some skill)

LARGE QUANTITIES OF DATA

KERAS + TENSFORFLOW

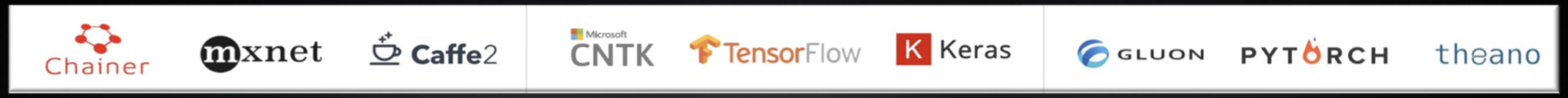
ML FRAMEWORK



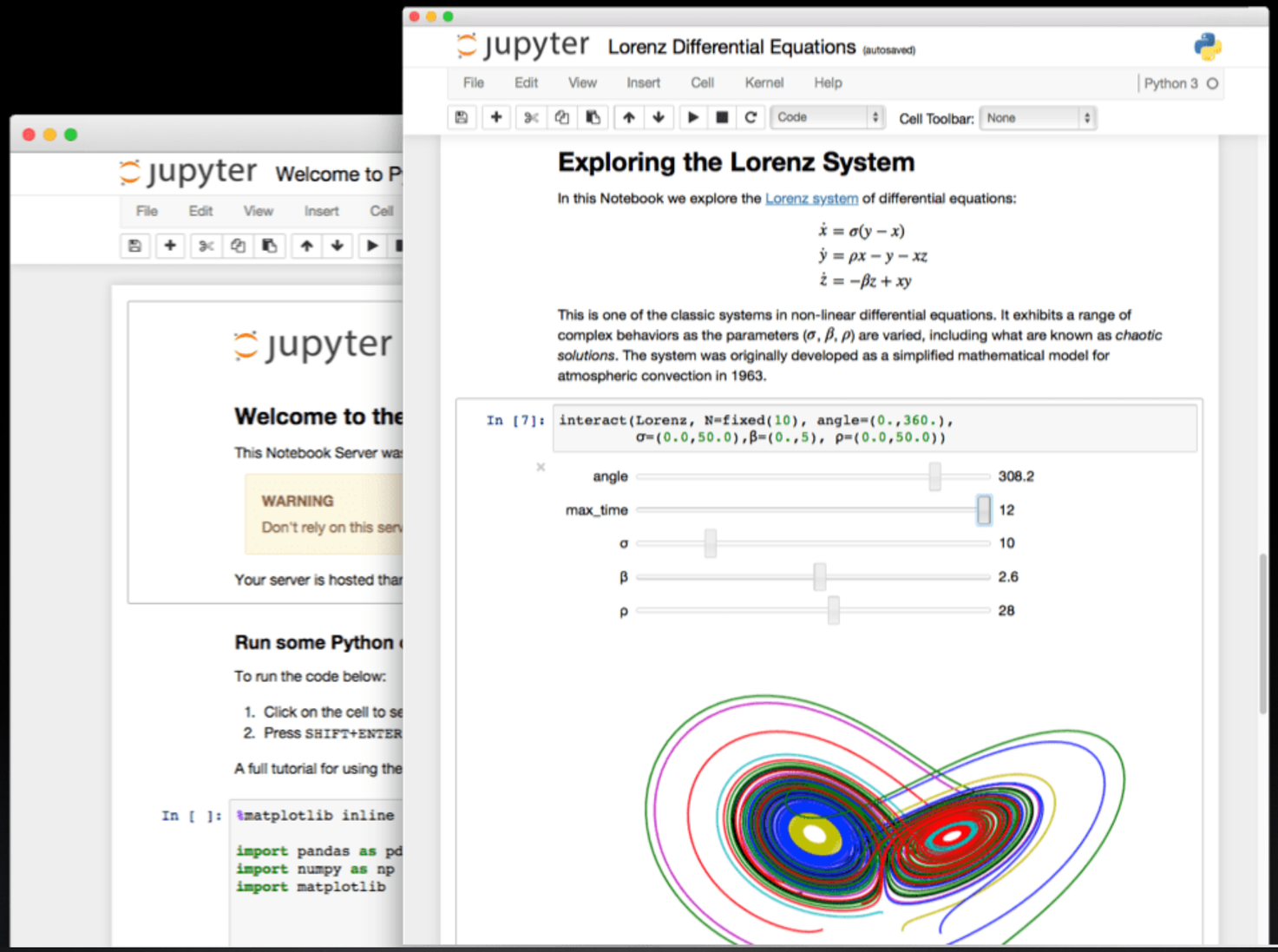
GPU ACCELERATOR

DEEP LEARNING FRAMEWORKS

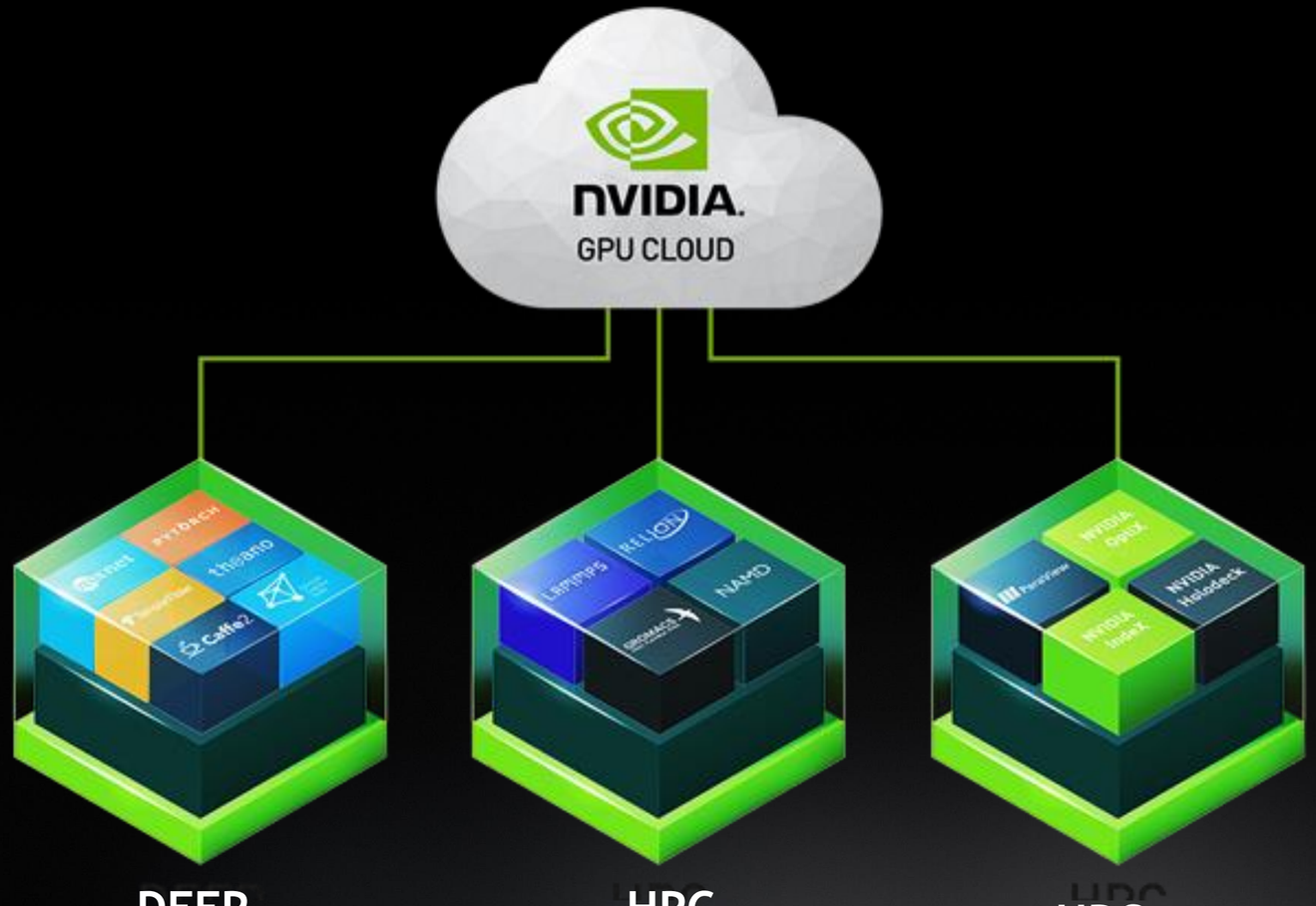
Many frameworks to choose from (but not for Fortran)



DEVELOPMENT ENVIRONMENT
JUPYTER NOTEBOOKS



NVIDIA GPU CLOUD REGISTRY
CONTAINERIZED SOFTWARE



docker

Singularity

LINEAR REGRESSION

With Scikit Learn

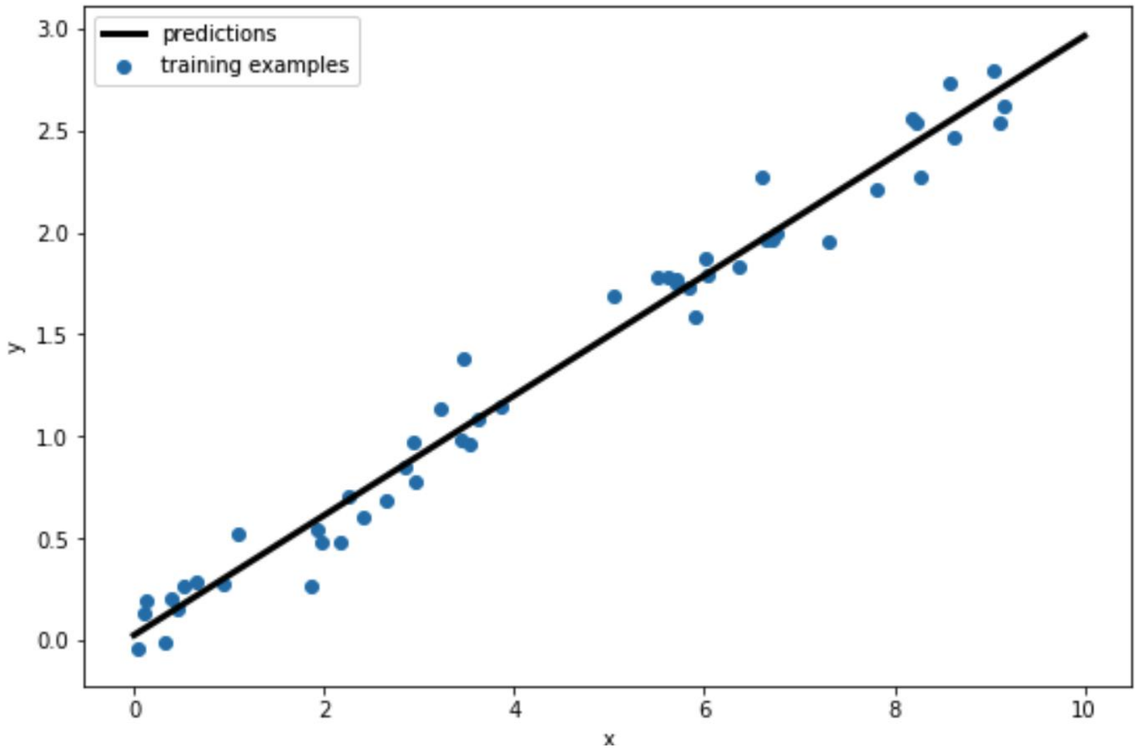
```
from sklearn.linear_model import LinearRegression
from numpy.random import rand
import numpy as np
```

```
# DATA
npts    = 50
x_train = 10*rand(npts,1)
y_train = 0.3*x_train + 0.5 * rand(npts,1)
```

```
# MODEL
model = LinearRegression()
```

```
# TRAIN
model.fit(x_train, y_train)
```

```
# TEST
x_test    = np.linspace(0,10,npts).reshape(-1,1)
y_predict = model.predict(x_test)
```



LINEAR REGRESSION

With Tensorflow and Keras

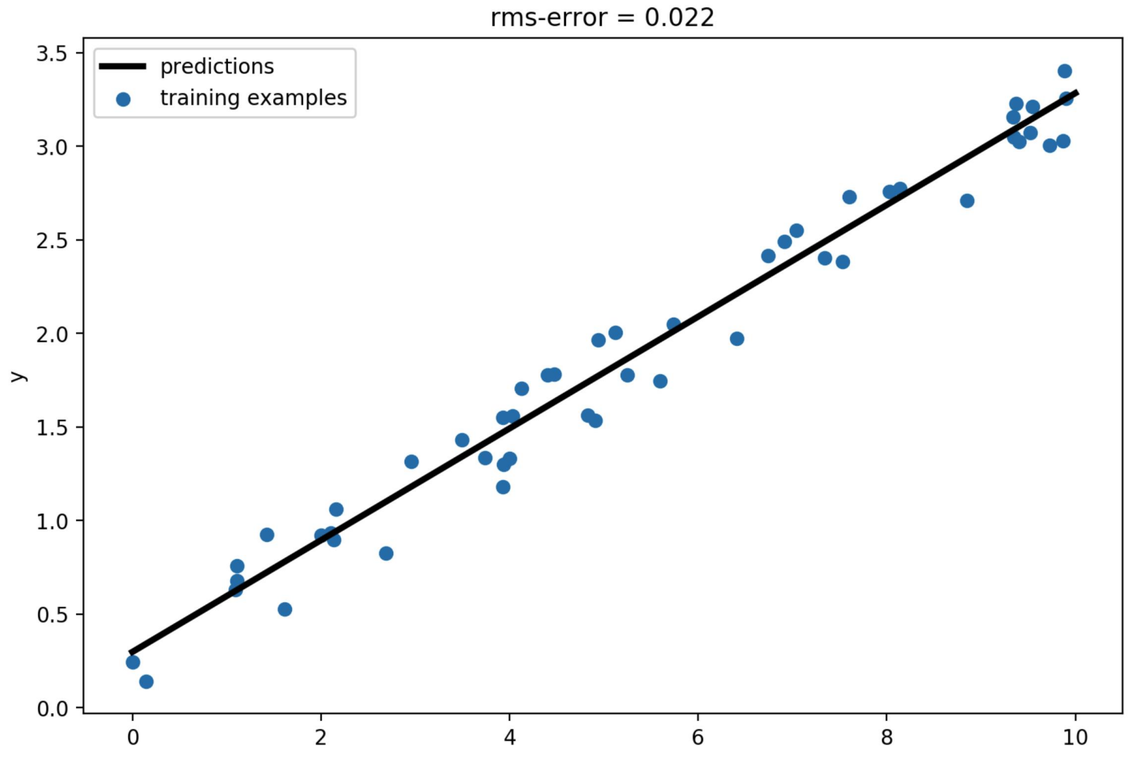
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.random import uniform
K.clear_session()
```

```
# DATA
npts    = 50
x_train = 10*uniform(shape=[npts,1])
y_train = 0.3*x_train + 0.5 * uniform(shape=[npts,1])
```

```
# MODEL
model = keras.models.Sequential()
model.add(keras.layers.Dense(1, input_shape=[1]))
optimizer = keras.optimizers.Adam(lr=1e-1)
model.compile(loss='mean_squared_error',optimizer=optimizer)
```

```
# TRAIN
hist = model.fit(x_train,y_train,epochs=100,verbose=0)
```

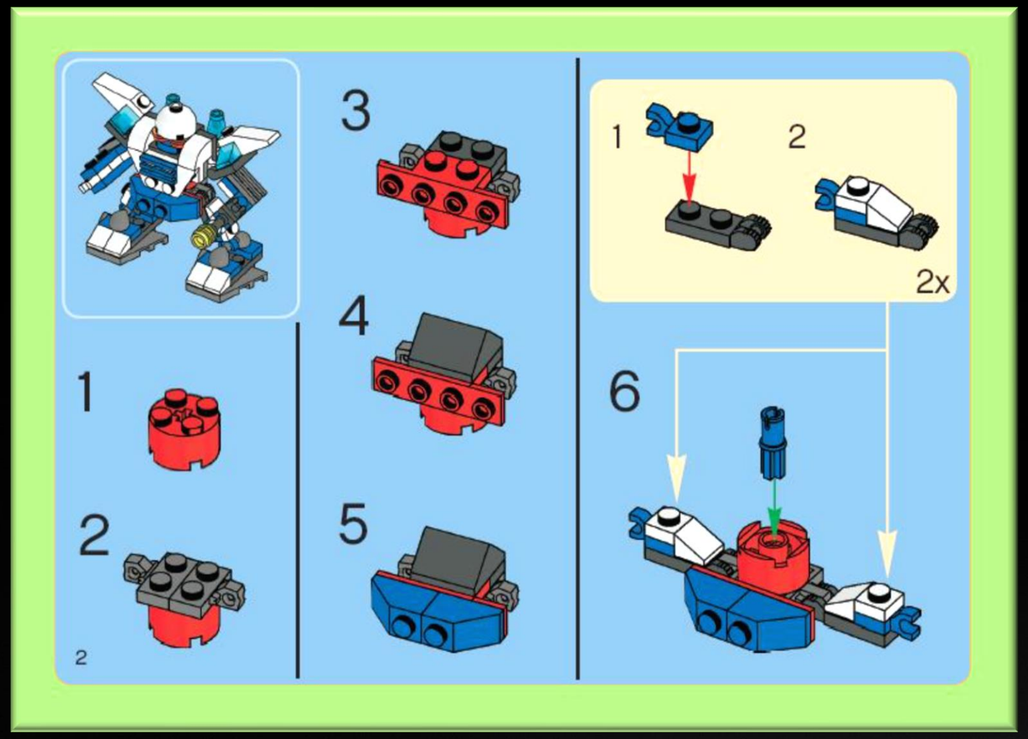
```
# TEST
x_test    = tf.linspace(0,10,npts)
x_test    = tf.reshape(x_test,[npts,1])
y_predict = model.predict(x_test)
```



TRAINING

TRAINING VS INFERENCE

TRAINING PHASE



SEARCH FOR THE RIGHT PIECES

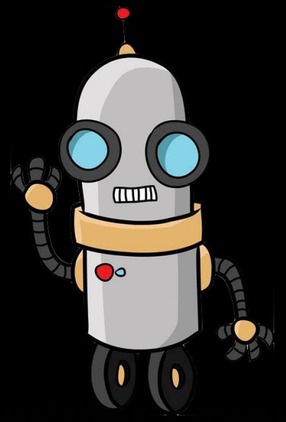
ONLINE LEARNING

INFERENCE PHASE

APPLY THE COMPLETED MODEL

TRAINING: THE PLAYERS

DATA, MODEL, LOSS, AND OPTIMIZER

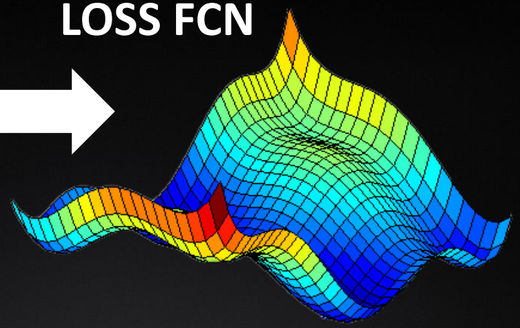


OPTIMIZER

DATA

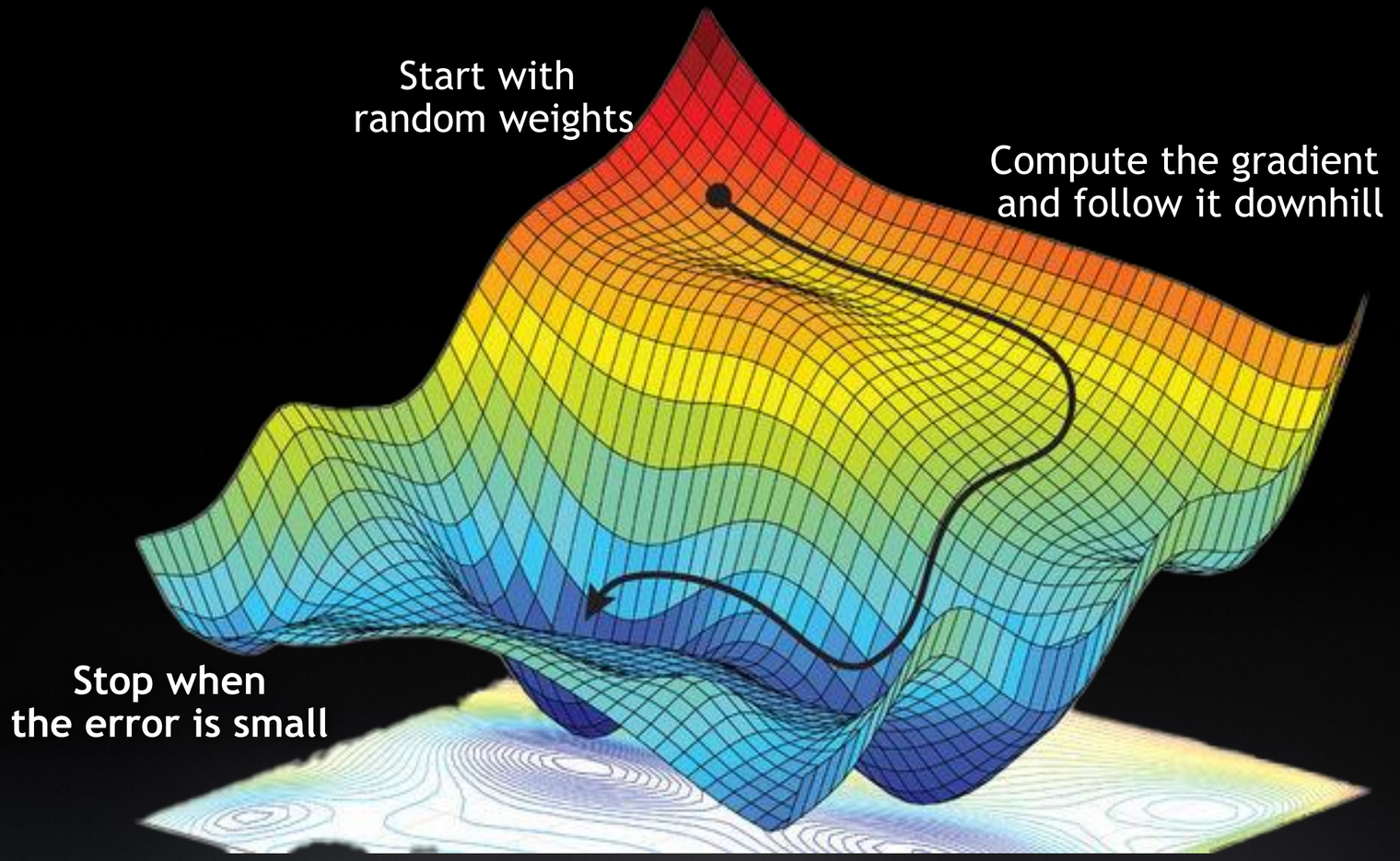
MODEL

LOSS FCN



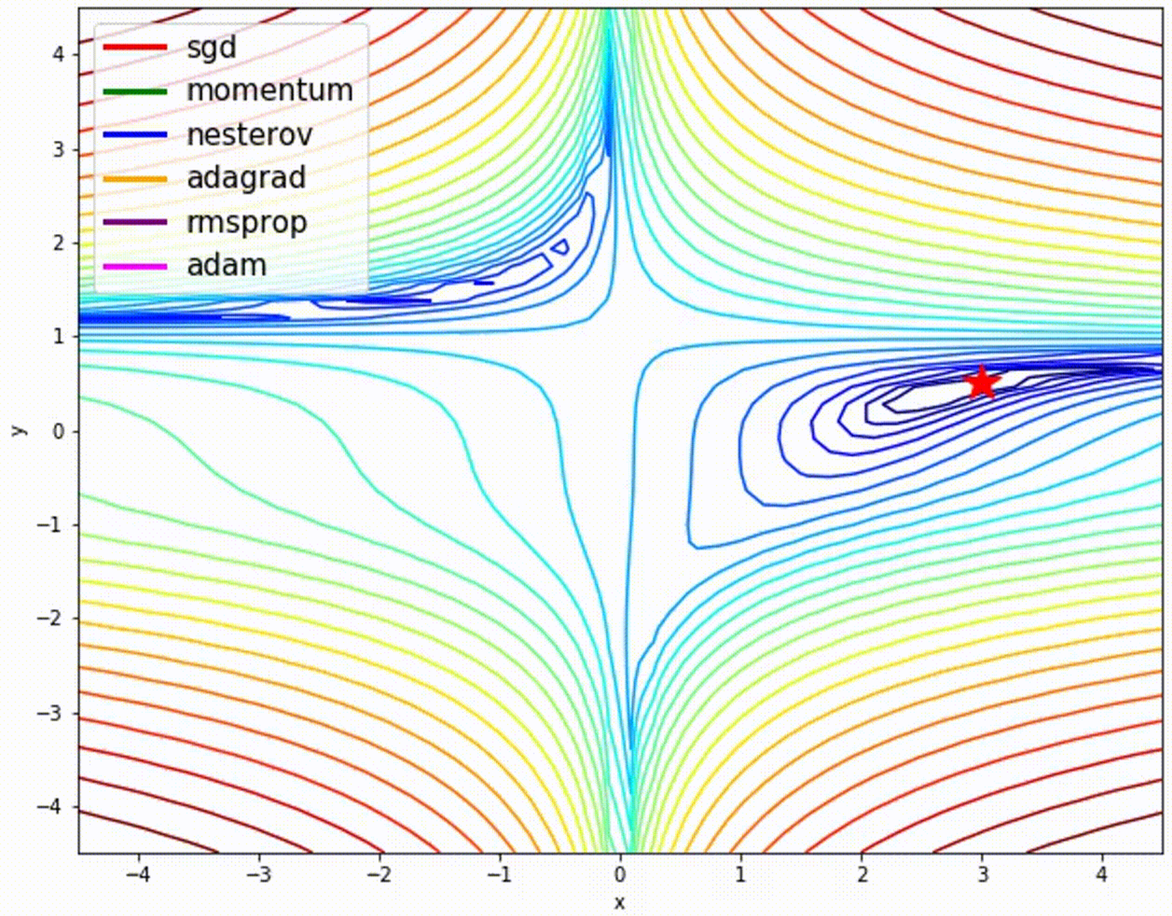
TRAINING: GRADIENT DESCENT

Finding as solution is as easy as falling down a hill



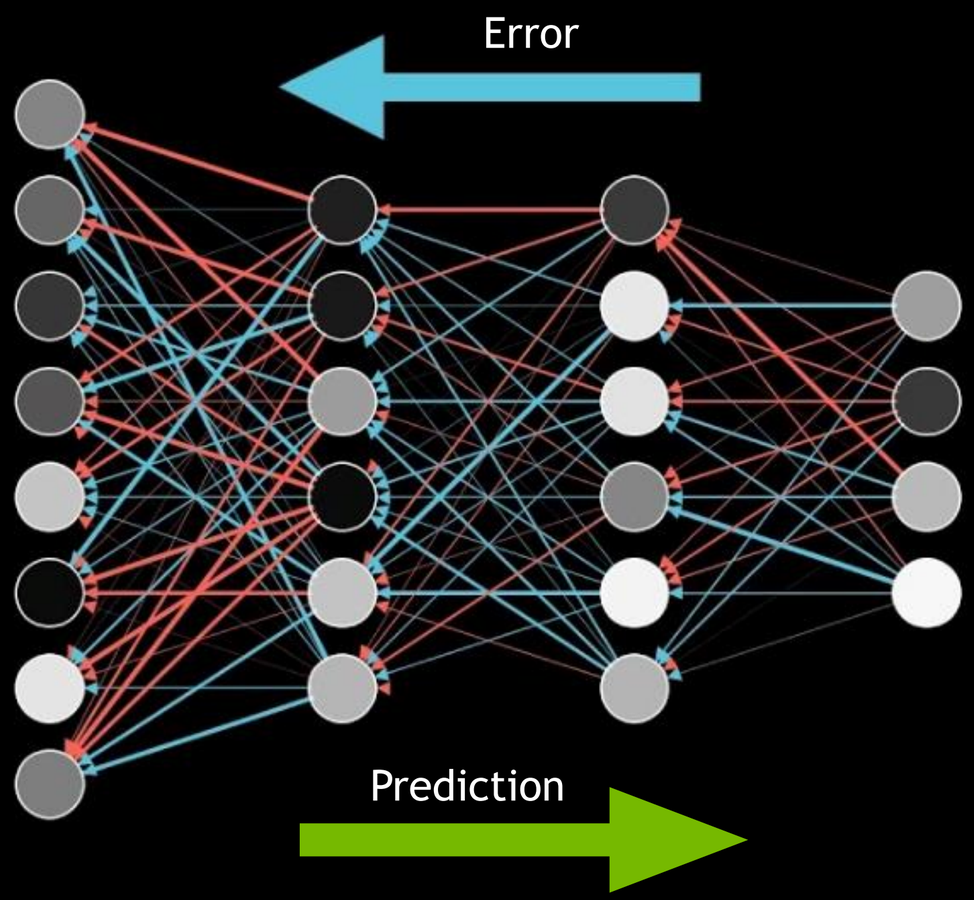
OPTIMIZERS

Many variations on stochastic gradient descent



TRAINING: BACKPROPAGATION

Compute the gradient, by efficiently assigning blame

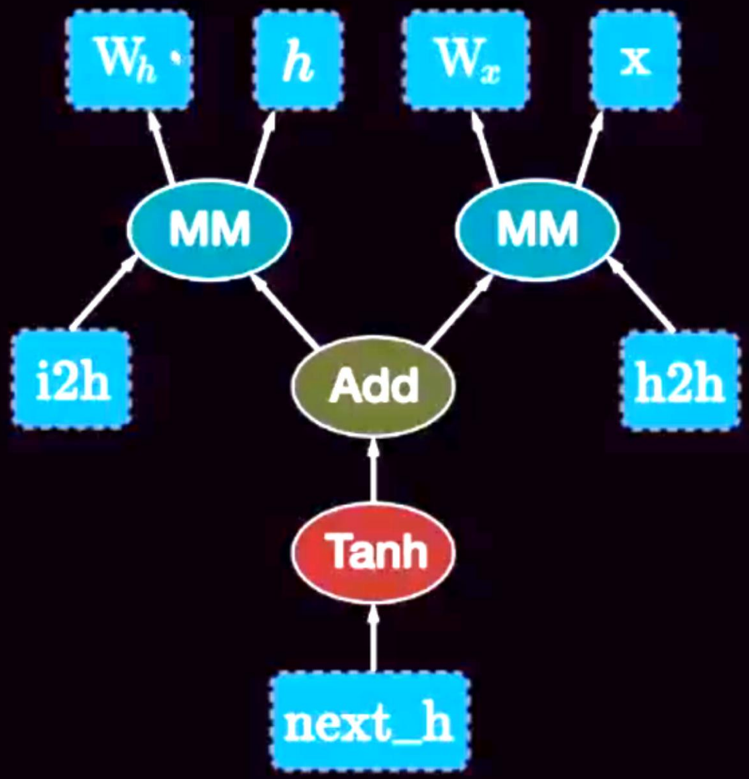


AUTOGRAD

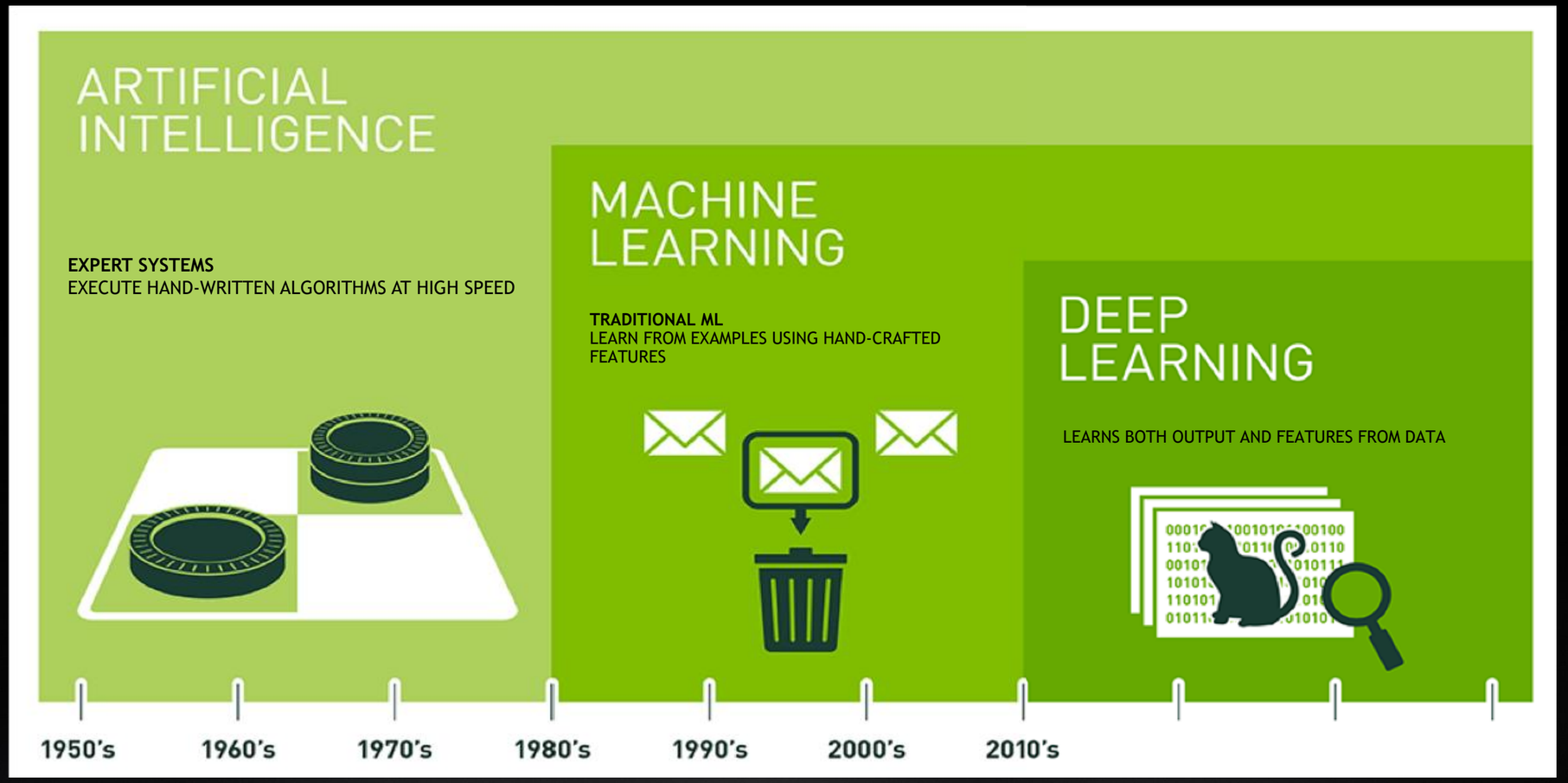
Let a framework keep track of your gradient, so you don't have to

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))
```

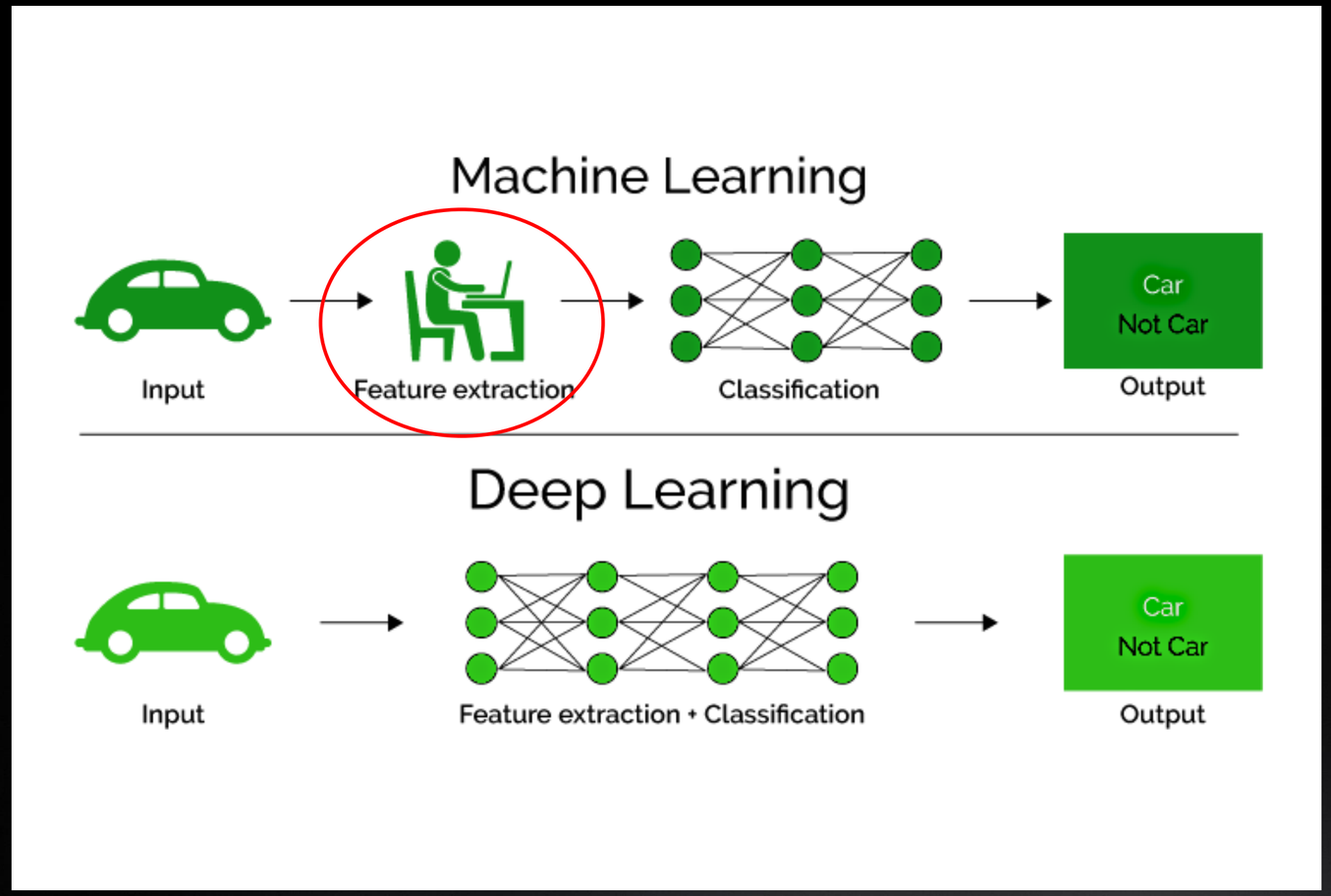


AI, MACHINE LEARNING, DEEP LEARNING



DEEP LEARNING VS. MACHINE LEARNING

When should I use deep learning vs traditional machine learning?



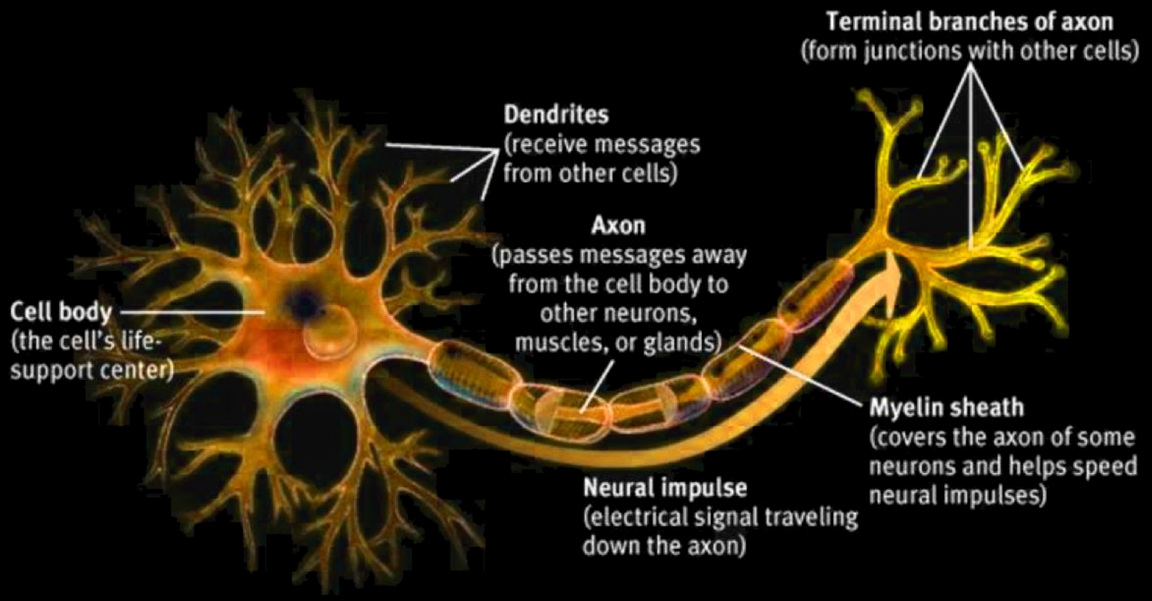
TRADITIONAL MACHINE LEARNING
Random forests, SVM, K-means, Logistic Regression
Features hand-crafted by experts
Small set of features: 10s or 100s
NVIDIA RAPIDS: orders of magnitude speedup

SUPERVISED DEEP LEARNING
CNN, RNN, LSTM, GAN, Variational Auto-encoders
Finds features automatically
High dimensional data: images, sounds, speech
Large set of labelled data (10k+ examples)
NVIDIA CU-DNN: accelerates DL frameworks

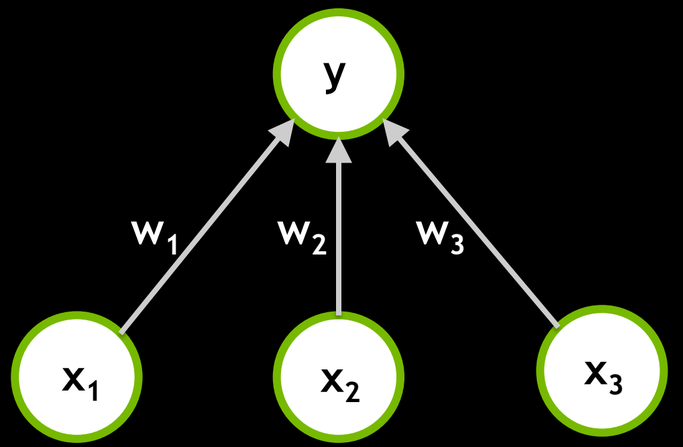
ARTIFICIAL NEURONS

Simple equations with adjustable parameters

Biological neuron



Artificial neuron

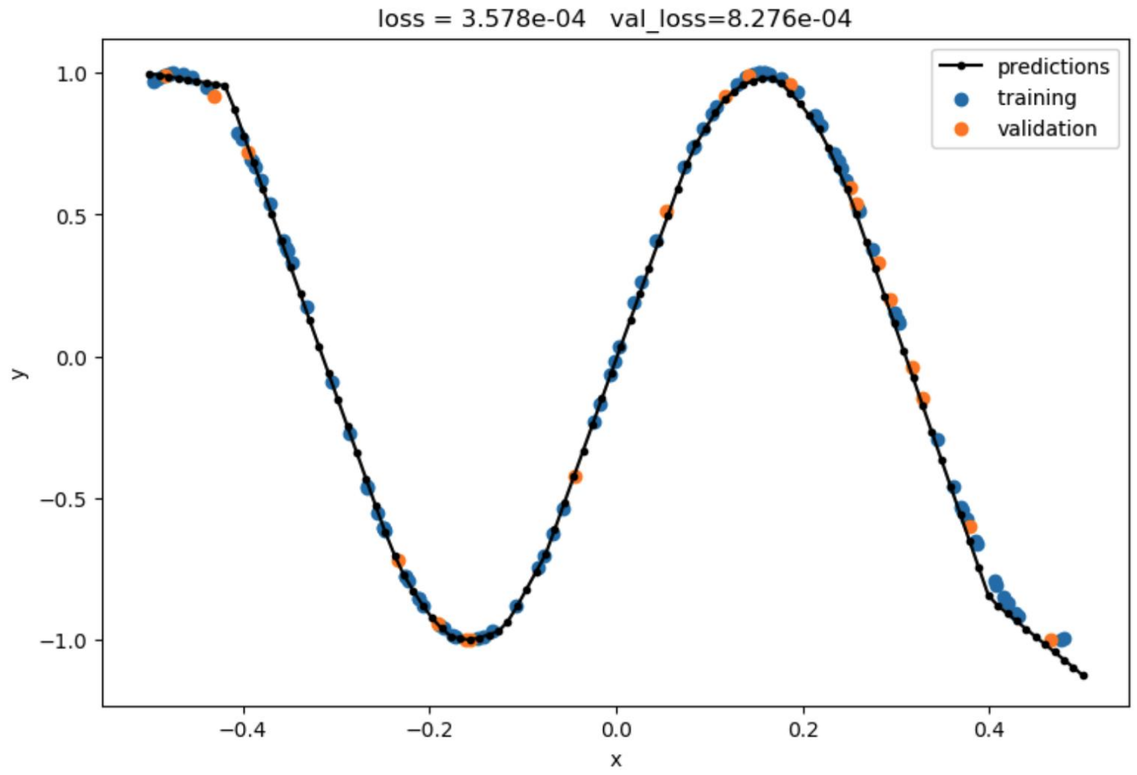


$$y = f(w_1x_1 + w_2x_2 + w_3x_3)$$

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

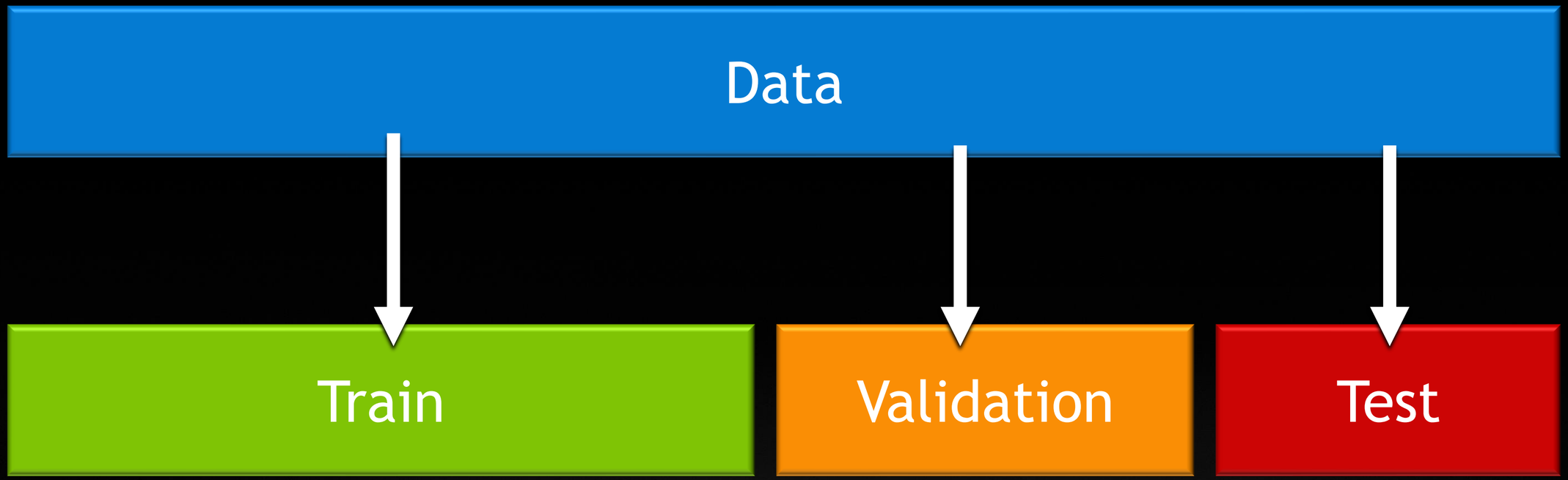
CURVE FIT WITH SINGLE LAYER NEURAL NETWORK

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.random import uniform
K.clear_session()

# DATA
def f(x): return tf.math.sin(10*x)
x_train = uniform(shape=[100,1]) - 0.5
x_val   = uniform(shape=[ 20,1]) - 0.5
y_train = f(x_train)
y_val   = f(x_val)

# MODEL (using keras functional API)
inputs   = keras.Input(shape=[1])
x        = keras.layers.Dense(units=100,activation="relu")(inputs)
x        = keras.layers.Dense(1)(x)
outputs  = keras.layers.Add()([inputs,x])
model    = keras.Model(inputs=inputs, outputs=outputs)
optimizer= keras.optimizers.Adam(lr=1e-2)
model.compile(loss='mean_squared_error',optimizer=optimizer)

# TRAIN
hist = model.fit(x_train,y_train,validation_data=(x_val,y_val),
                 epochs=5000,verbose=0, batch_size=1)
# TEST
x_test    = tf.linspace(-.5,.5,100)
y_predict = model.predict(x_test)
```



DATA SPLITTING

KEEP TEST, TRAINING, AND VALIDATION DATA SEPERATE



For model training

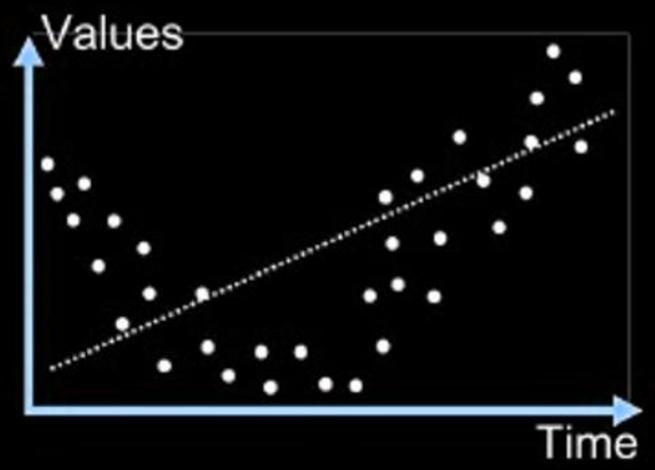
For hyperparameter tuning

For final evaluation

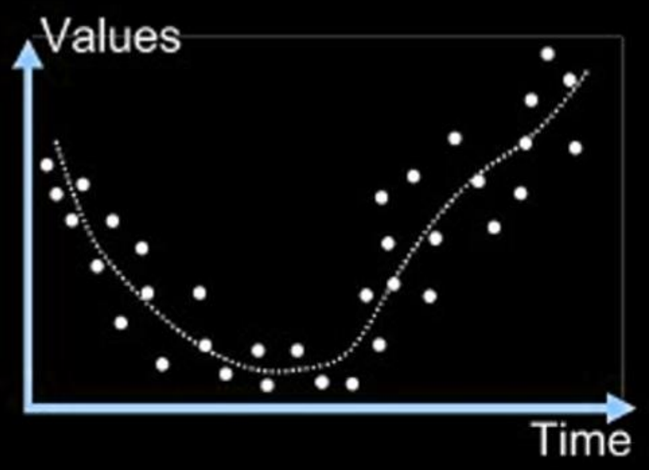
MODEL CAPACITY AND REGULARIZATION

MODEL CAPACITY

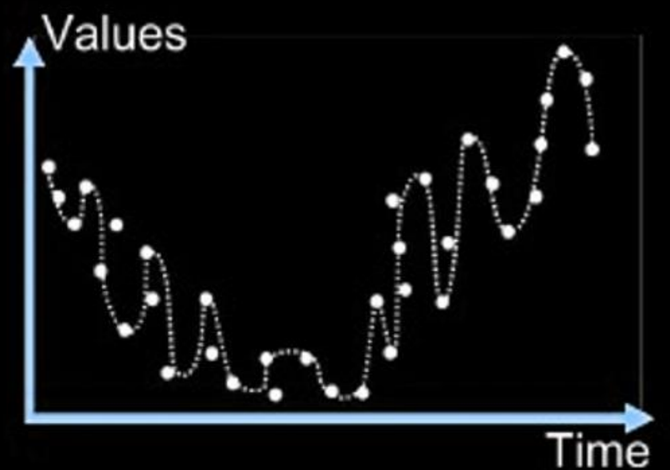
A good model is one that generalizes to new data



UNDERFIT



GOOD FIT



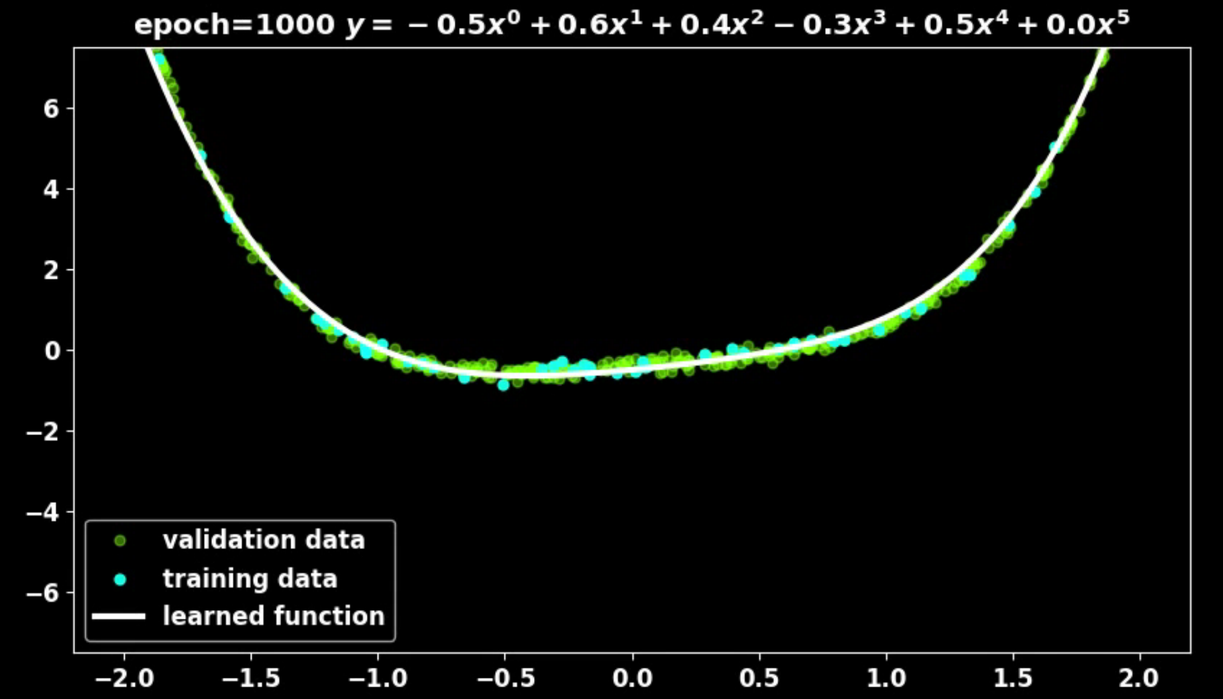
OVER FIT

GOOD-FIT

Checking for Generalization

Training Loop

```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.10)

model     = taylor_series(order=5)
optimizer = torch.optim.Adam(model.params, lr=1.0e-2)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```



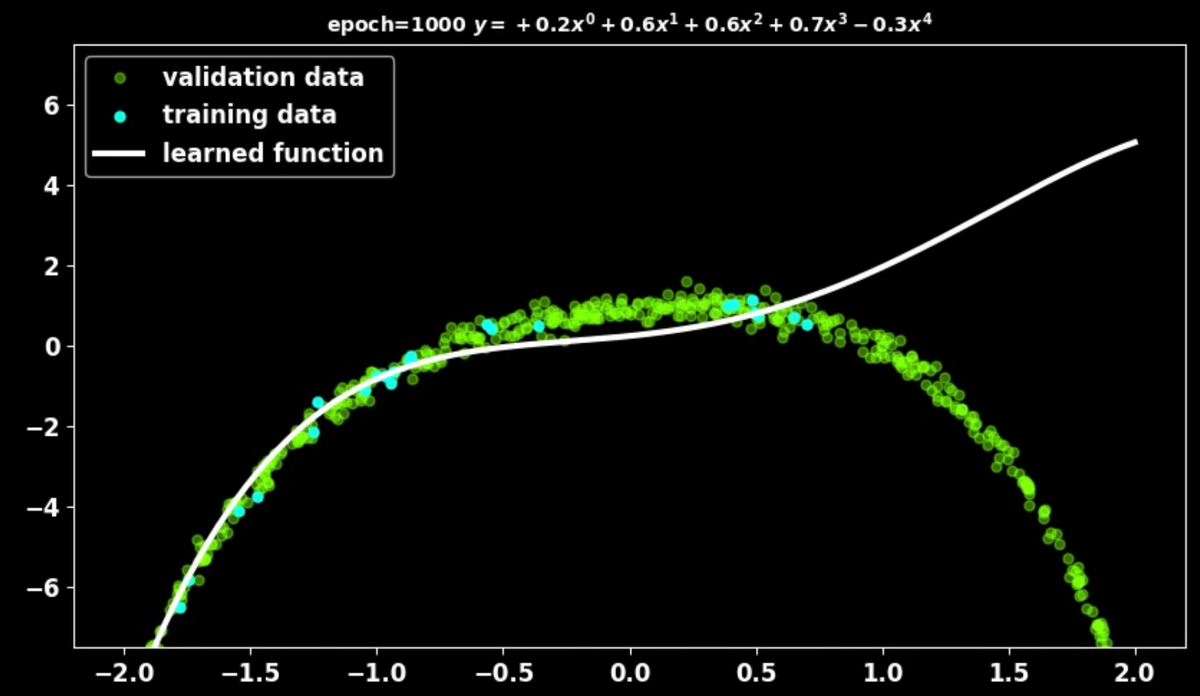
OVER-FITTING

Captures training data, but generalizes poorly

Training Loop

```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.02)

model     = taylor_series(order=8)
optimizer = torch.optim.Adam(model.params, lr=5.0e-3)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```

Use more data points
Reduce model capacity



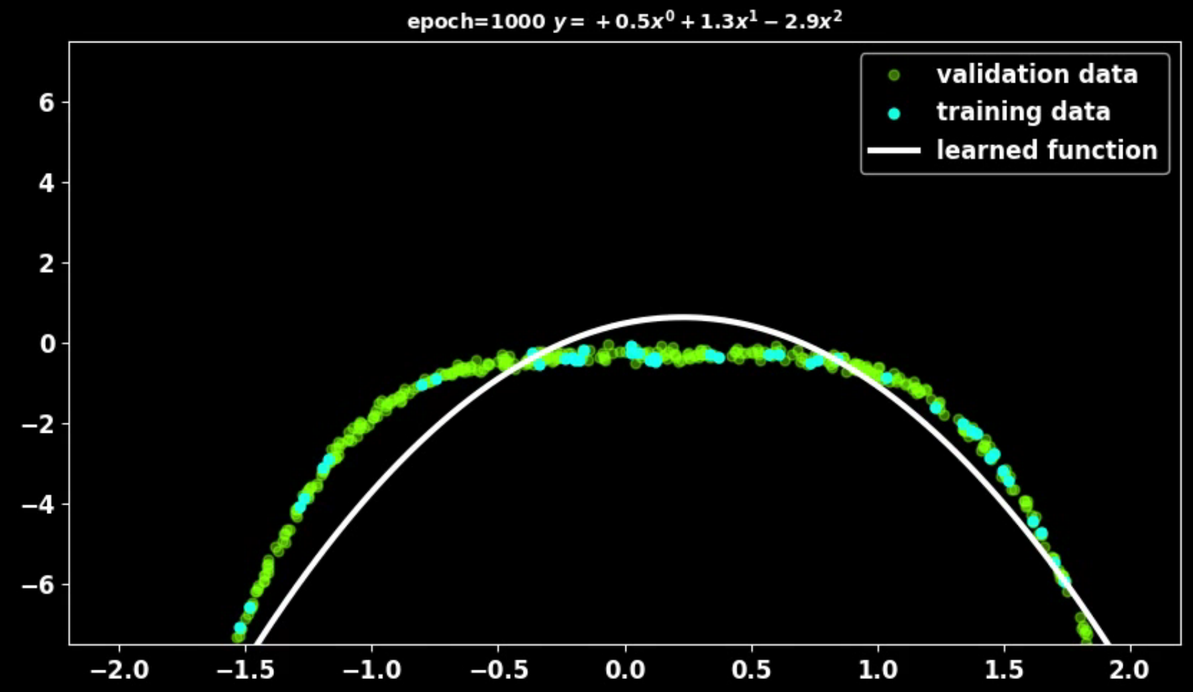
UNDER-FITTING

Model is too simple to fit the curve

Training Loop

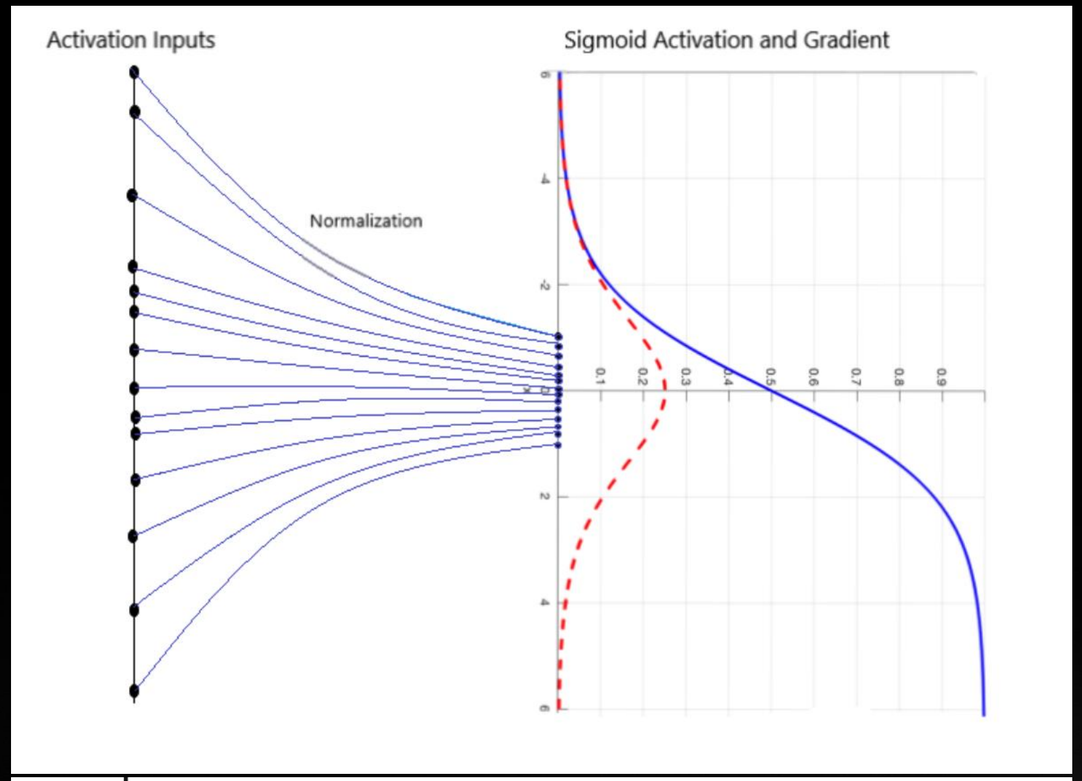
```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.10)

model     = taylor_series(order=2)
optimizer = torch.optim.Adam(model.params, lr=1.0e-2)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```

Increase model capacity
Use a different model

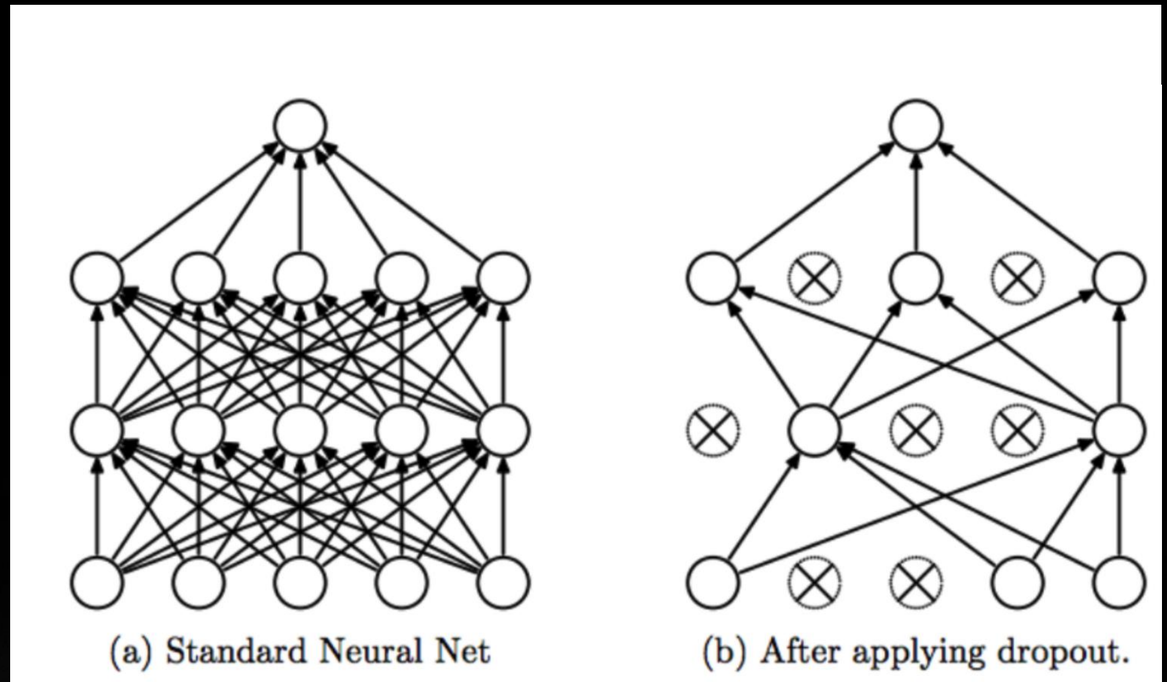


REGULARIZATION

BatchNorm and Dropout




```
model.add(Conv2D(60,3, padding = "same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
```



(a) Standard Neural Net

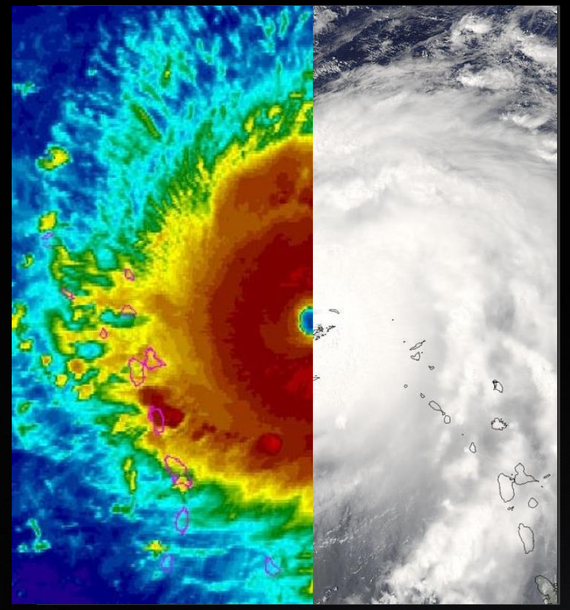
(b) After applying dropout.

```
model=keras.models.Sequential()
model.add(keras.layers.Dense(150, activation="relu"))
model.add(keras.layers.Dropout(0.5))
```

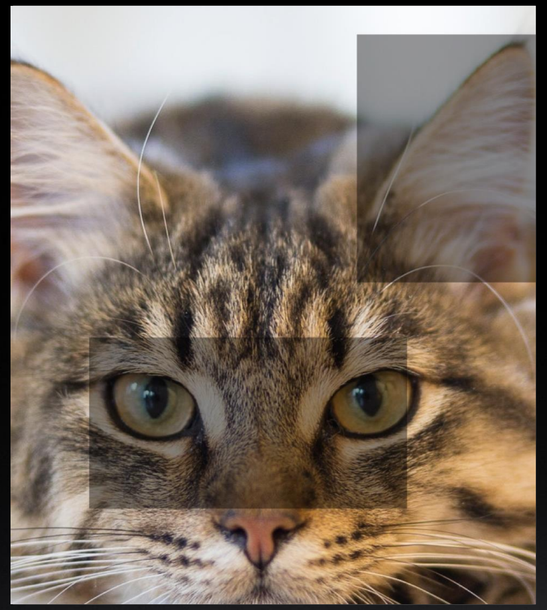
CHALLENGES AND POTENTIAL SOLUTIONS

LABELLING LARGE QUANTITIES OF DATA

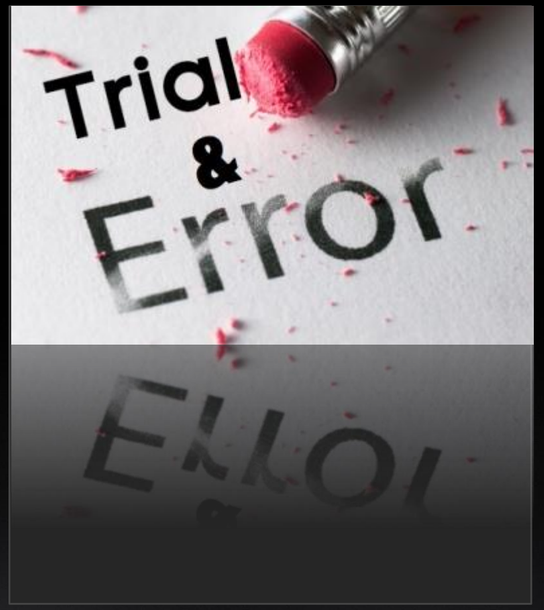
How can we overcome the need for manual labelling?



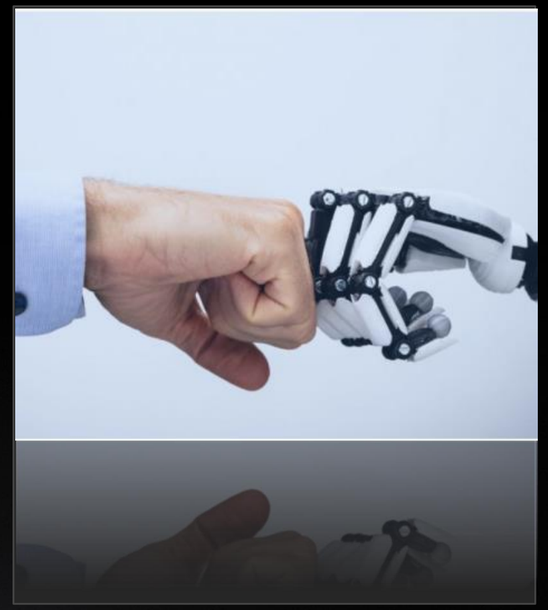
Data Fusion
Using one data source as the label for another



Self-Supervised Learning
Predicting input B from input A

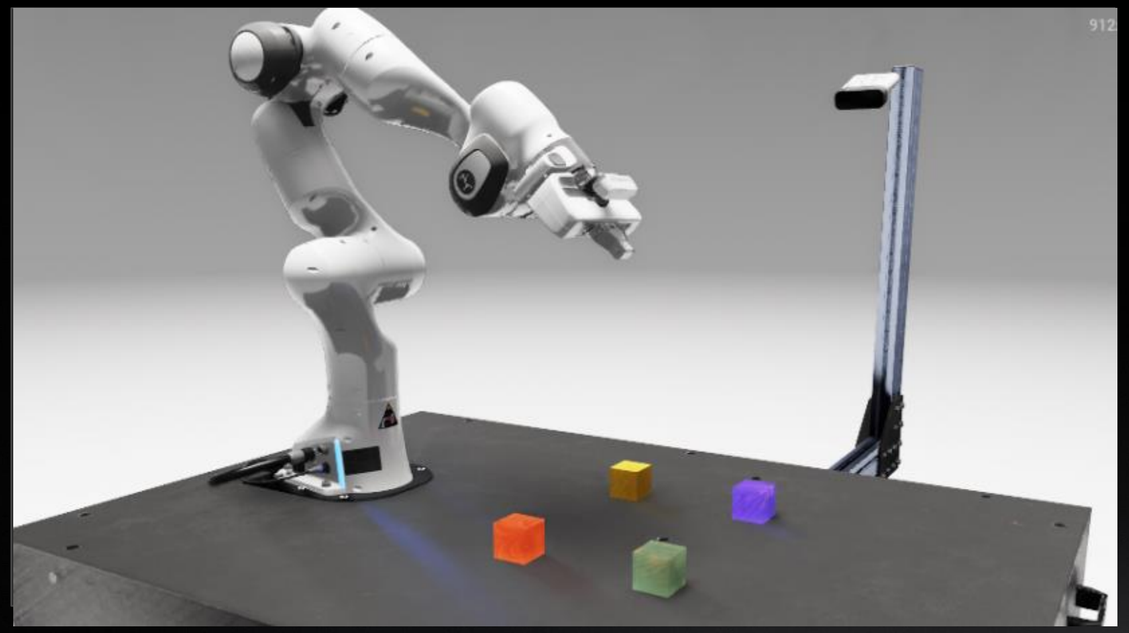


Reinforcement Learning
Obtaining labels directly from the environment or simulation

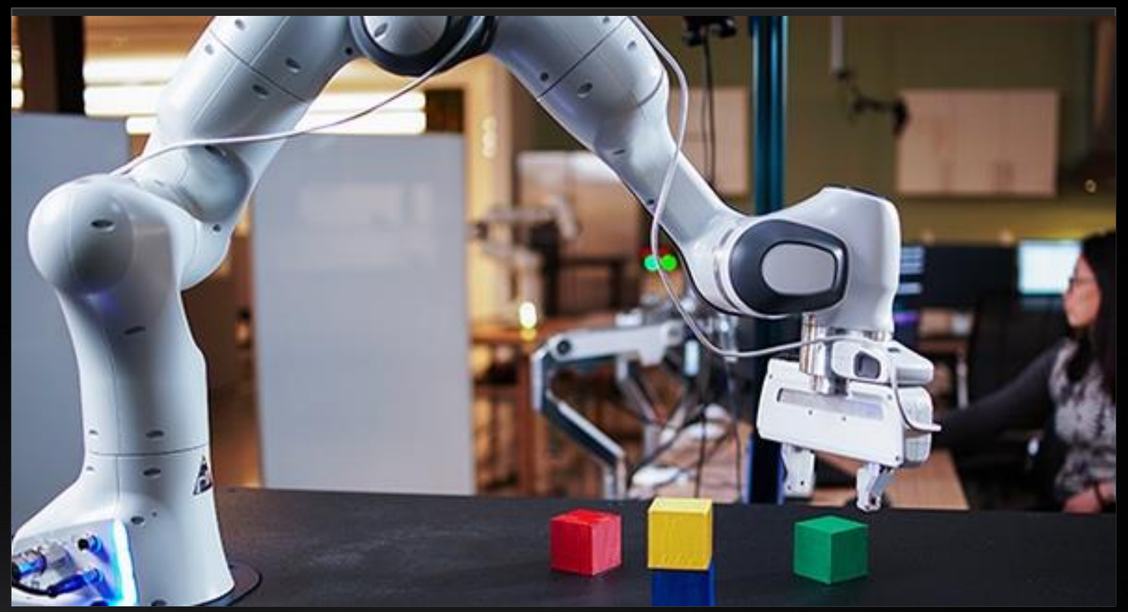


Human-in-the-loop
Using human machine iteration to make labelling easier

TRANSFER LEARNING: DON’T START FROM SCRATCH

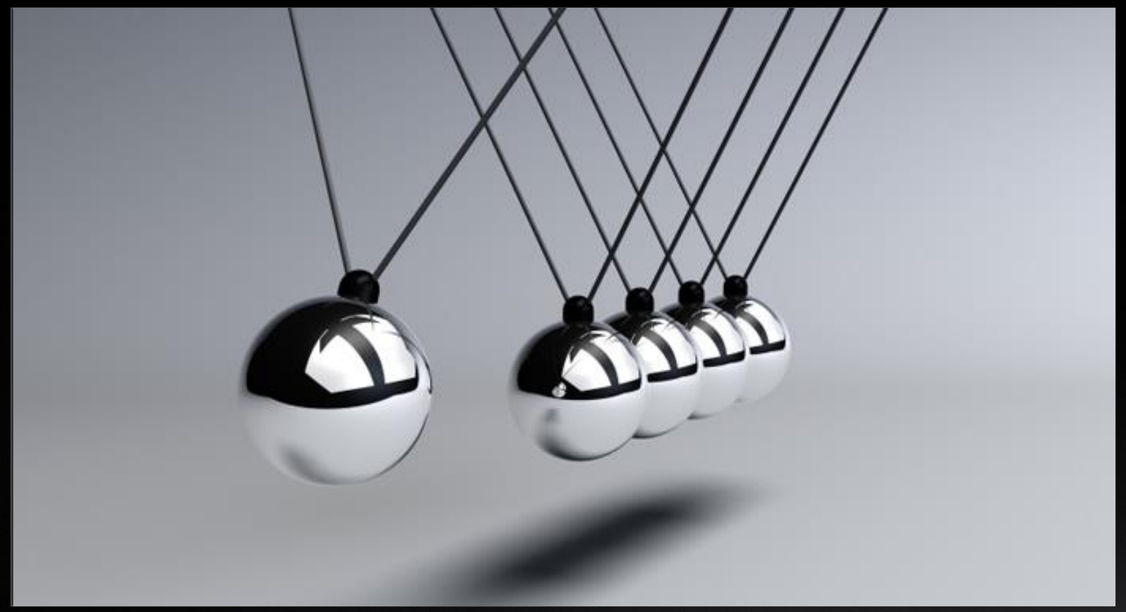


Train on simulated or related data

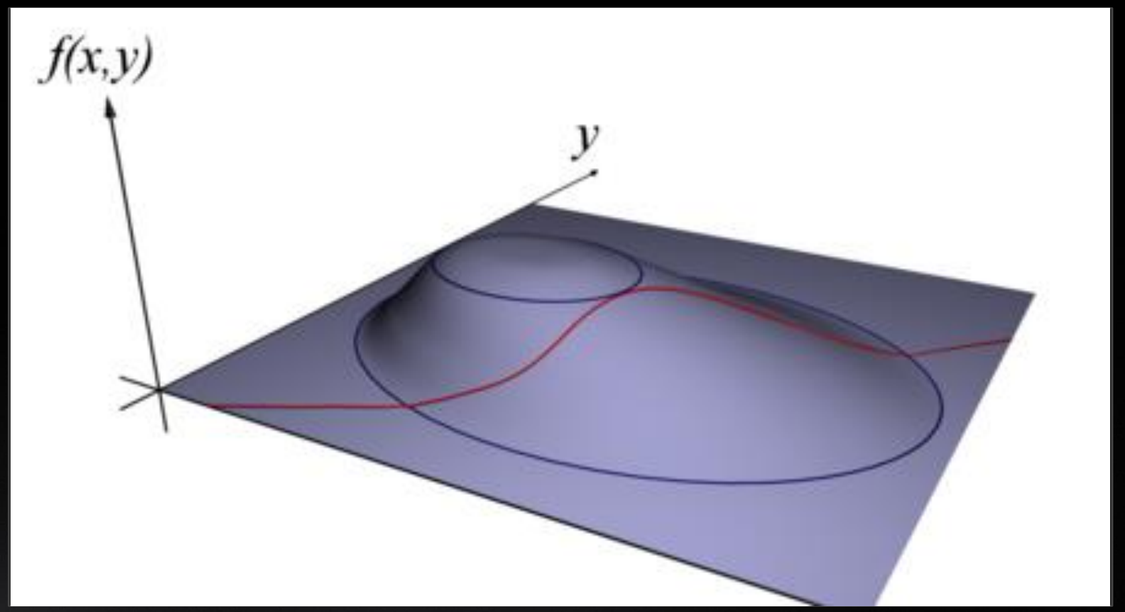


Fine-tune on the real data

ENFORCING PHYSICAL CONSTRAINTS

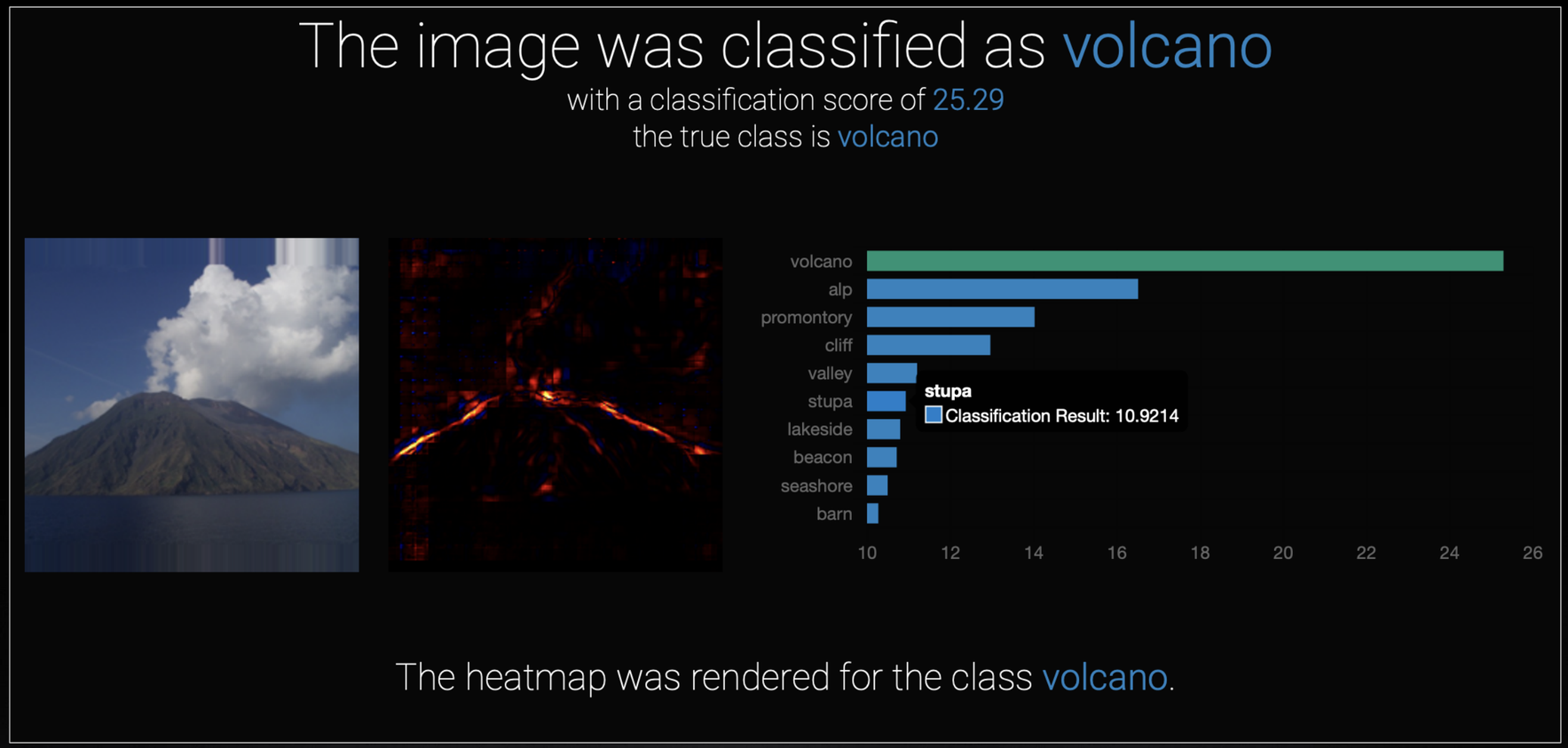


Conservation of Mass, Momentum, Energy, Incompressibility, Turbulent Energy Spectra, Translational Invariance



Lagrange multipliers (penalization), Hard Constraints, Projective Methods, Differentiable Programming

INTERPRETABILITY: EXPLAINABLE AI



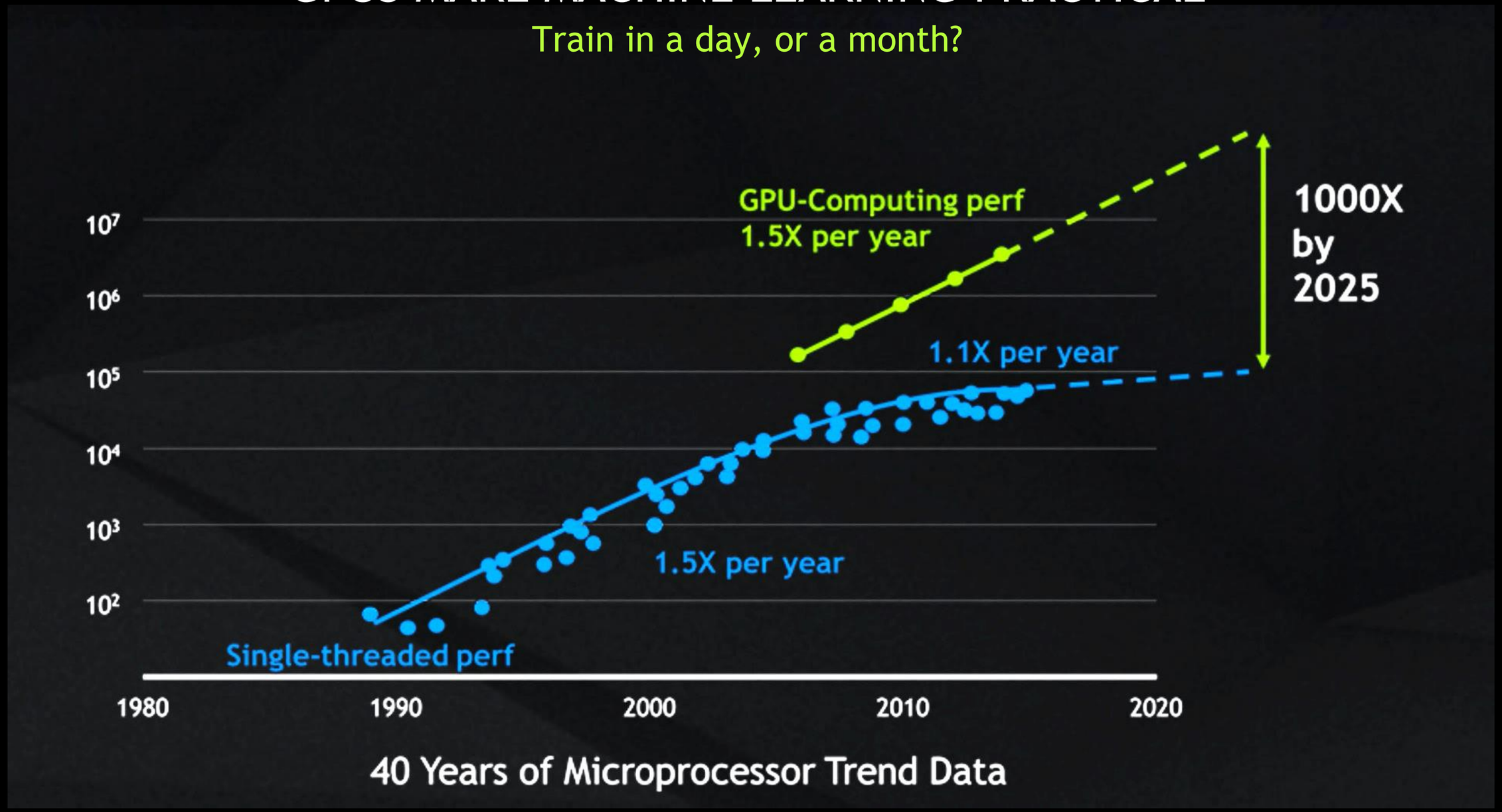
Layer-wise Relevance Propagation

https://lrpserver.hhi.fraunhofer.de/image-classification

USING YOUR GPU

GPUS MAKE MACHINE LEARNING PRACTICAL

Train in a day, or a month?

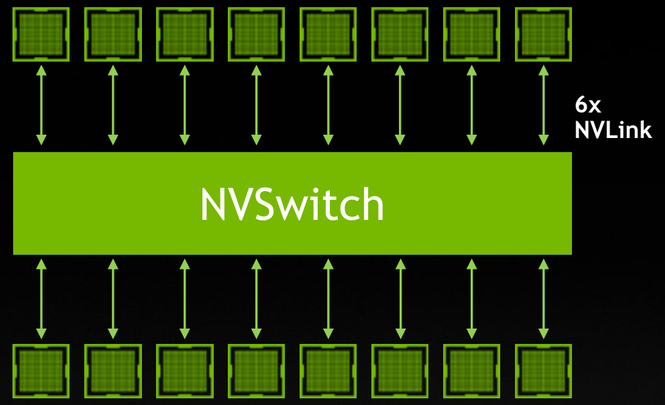


PILLARS OF DATA SCIENCE PERFORMANCE

CUDA Architecture

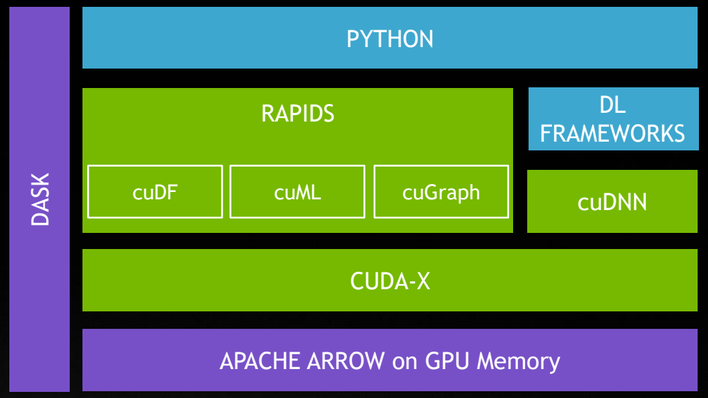
Massively Parallel Processing

NVLink/NVSwitch



High Speed Connecting between GPUs for Distributed Algorithms

CUDA-X AI



NVIDIA GPU Acceleration Libraries for Data Science and AI

LEARNED FUNCTIONS ARE GPU ACCELERATED

Next level software. No porting required.

HOW CAN I GET ACCESS TO A POWERFUL GPU?

Many way to take advantage of NVIDIA GPUs for Deep Learning

NVIDIA Quadro
Laptop or Workstation

IBM Cloud

Google Cloud

aws

Microsoft Azure

Cloud Computing Services
(Free hours to start)

National Supercomputers
(Apply for compute)

colab

Google Colab
(1 Free NVIDIA GPU)

VERIFYING YOUR GPU

TensorFlow

```
from tensorflow.python.client import device_lib
print('GPU' in str(device_lib.list_local_devices()))
```

```
True
```

PyTorch

```
from torch import cuda
print(f"cuda.is_available: {cuda.is_available()}")
print(f"cuda.device_count(): {cuda.device_count()}")
print(cuda.get_device_name(cuda.current_device()))
```

```
cuda.is_available: True
cuda.device_count(): 2
Quadro GV100
```

Keras

```
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

```
Using TensorFlow backend.
```

```
['/job:localhost/replica:0/task:0/device:GPU:0',
 '/job:localhost/replica:0/task:0/device:GPU:1']
```

Julia

```
using CUDAdrv, Printf
@printf("device 0 = %s \n" , CUDAdrv.name(CuDevice(0)))
@printf("device 1 = %s \n" , CUDAdrv.name(CuDevice(1)))
```

```
device 0 = Quadro GV100
device 1 = Quadro GV100
```

TRAINING ON A SINGLE GPU

Keras

Automatically uses GPU if available

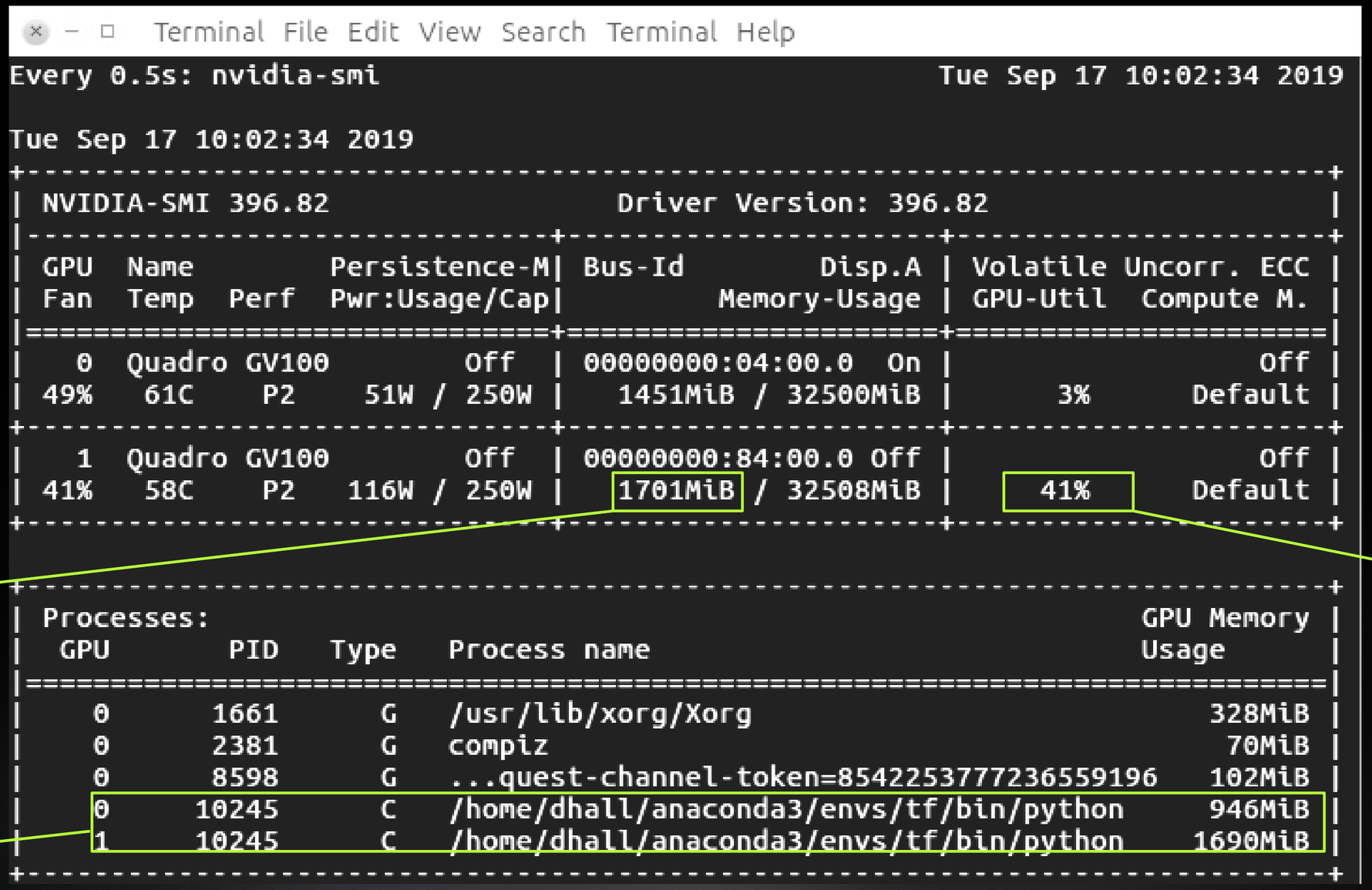
Julia

```
use_gpu = true
to_device(x) = use_gpu ? gpu(x) : x
x     = to_device.(x)
y     = to_device.(y)
model = to_device(model)
```

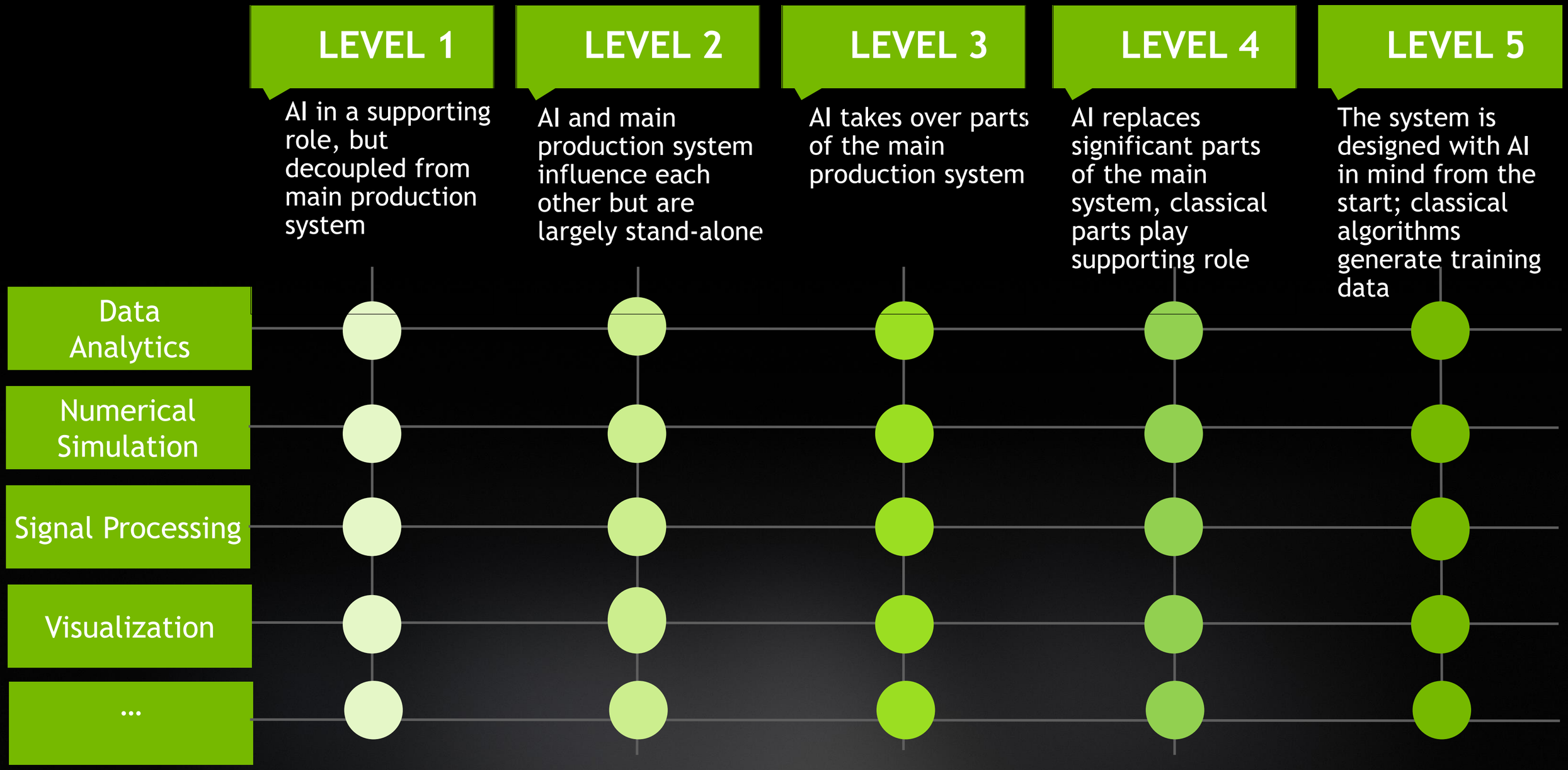
PyTorch

```
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")

# move data onto the GPU
model = model.to(device)
X,Y   = X.to(device), Y.to(device)

# Alocate on the GPU directly
dtype = torch.cuda.FloatTensor
X = torch.zeros(100).type(dtype)

# set default location for tensors
torch.set_default_tensor_type('torch.cuda.FloatTensor')
```

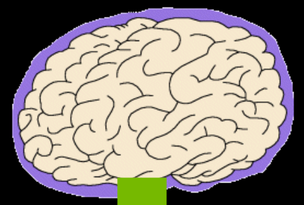
NVIDIA-SMI
System Management Interface



LEVELS OF AI ENGAGEMENT



COMPUTATIONAL SCIENCES



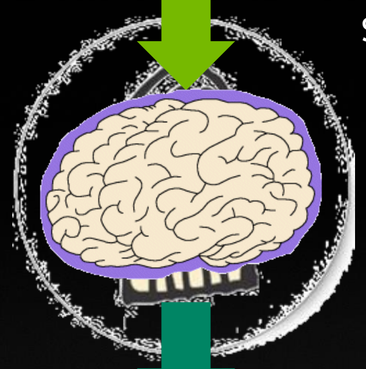
Create

Inputs

Mathematical
Model, First Principles

Outputs

Some Level of Approximation



Create

Inputs

Efficient Implementation

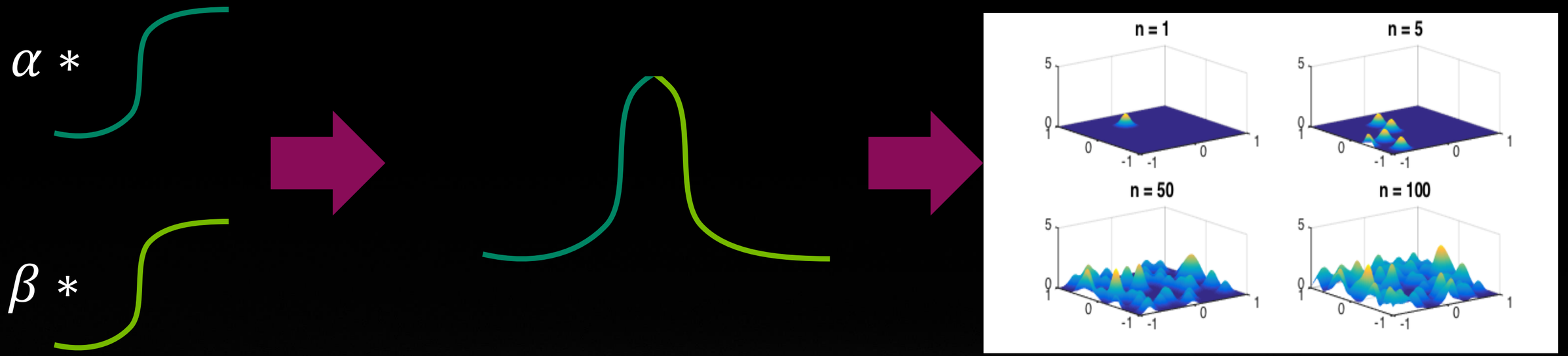
Outputs

Similarities to the shift
Feature → Network Engineering?

NNs as a Porting Strategy?

CAN THIS WORK ∀? ABOLUTELY, YES!

Proof: Universal Approximation Theorem



Take many non-linearities

Combine to form peaks
(one hidden layer is enough!)

And assemble your arbitrary function with arbitrary ε

Problem: this is an essentially useless theorem for practical purposes

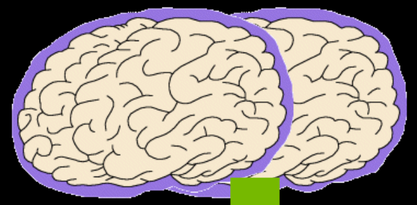
WILL THIS WORK ∀?

Considering pesky practical constraints, like memory and performance

- Anecdotal Evidence: ∃ scientific cases where NNs seem to do work extremely well
- Save bet: it will not work for ∀
- Therefore, by induction (sort of):
  - There exists ∃ a subspace in ∀ HPC applications, for which AI works well
  - Need to explore the **size** and **shape** of this subspace
  - Currently I think it is fair to say we don't understand this domain very well
  - **But:** Each individual case promising 10x, 100x,1000x performance improvement is probably worth exploring; those can be groundbreaking!

But Intuition is Misleading

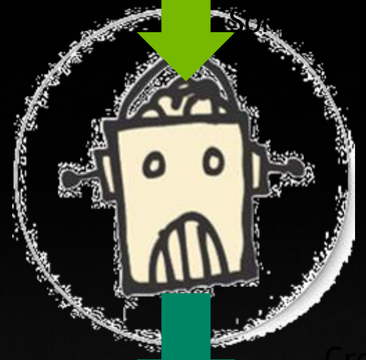
WHAT MAKES AI * HPC SPECIAL?



Inputs

Mathematical Model

Outputs



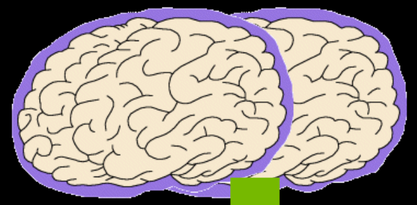
Create

Inputs

Efficient Implementation

Outputs

WHAT MAKES AI * HPC SPECIAL?



Mathematical Model

Outputs

Loss Function

"Prior"

Create

Inputs

Efficient Implementation

Outputs

Backpropagation

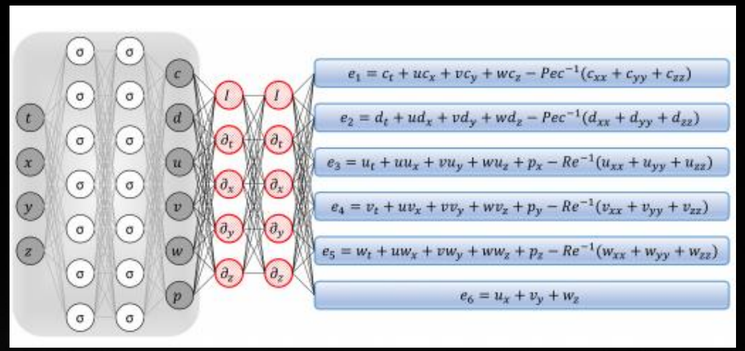
Note: We have **more** information about the ground truth in AI*HPC (often mathematically precise)

This should actually be an advantage!
Why does it sometimes feel like a disadvantage?

HOW TO FILL IN THE

E.g. Adversarial Fuzzing

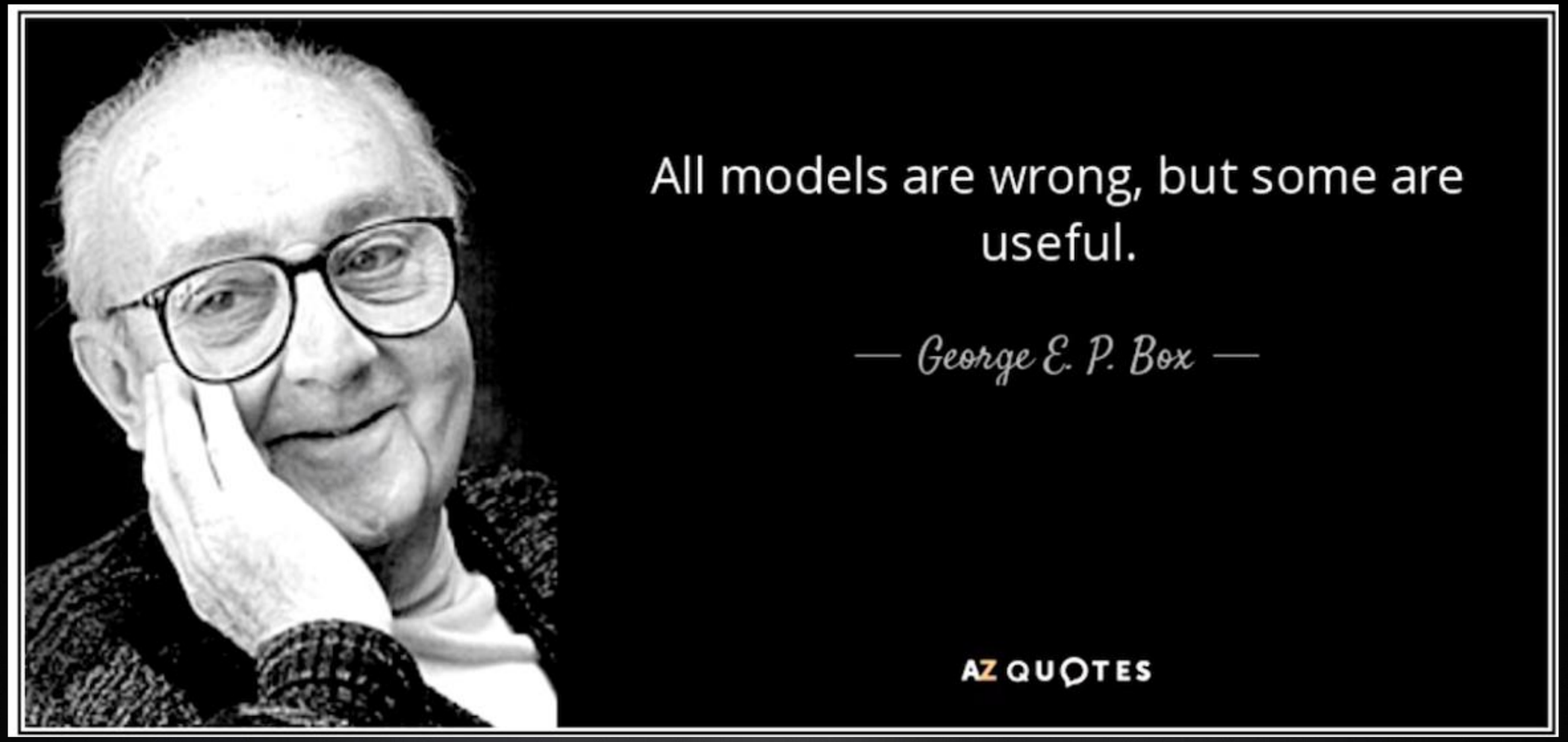
E.g. Declarative Building Blocks to NN Translation



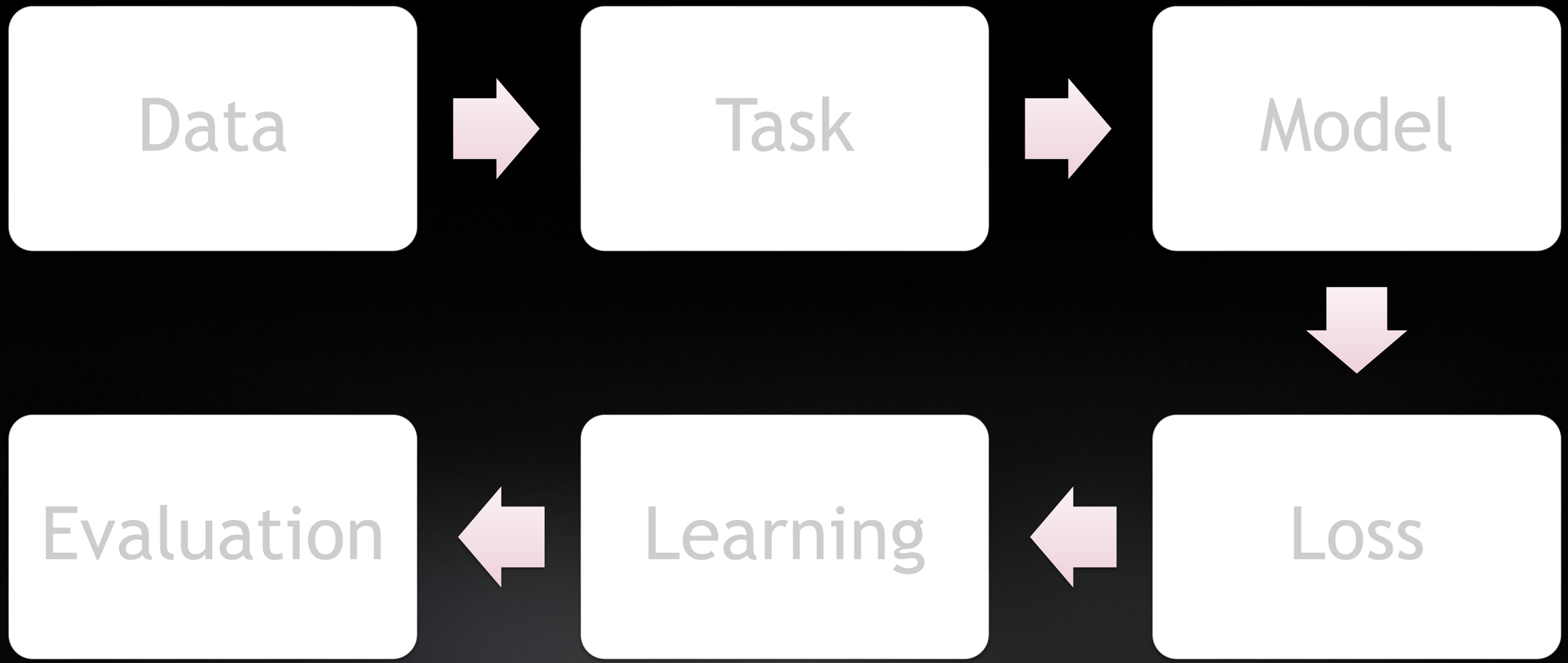
E.g. Physics Informed Networks?[1], ODE Networks?[2]

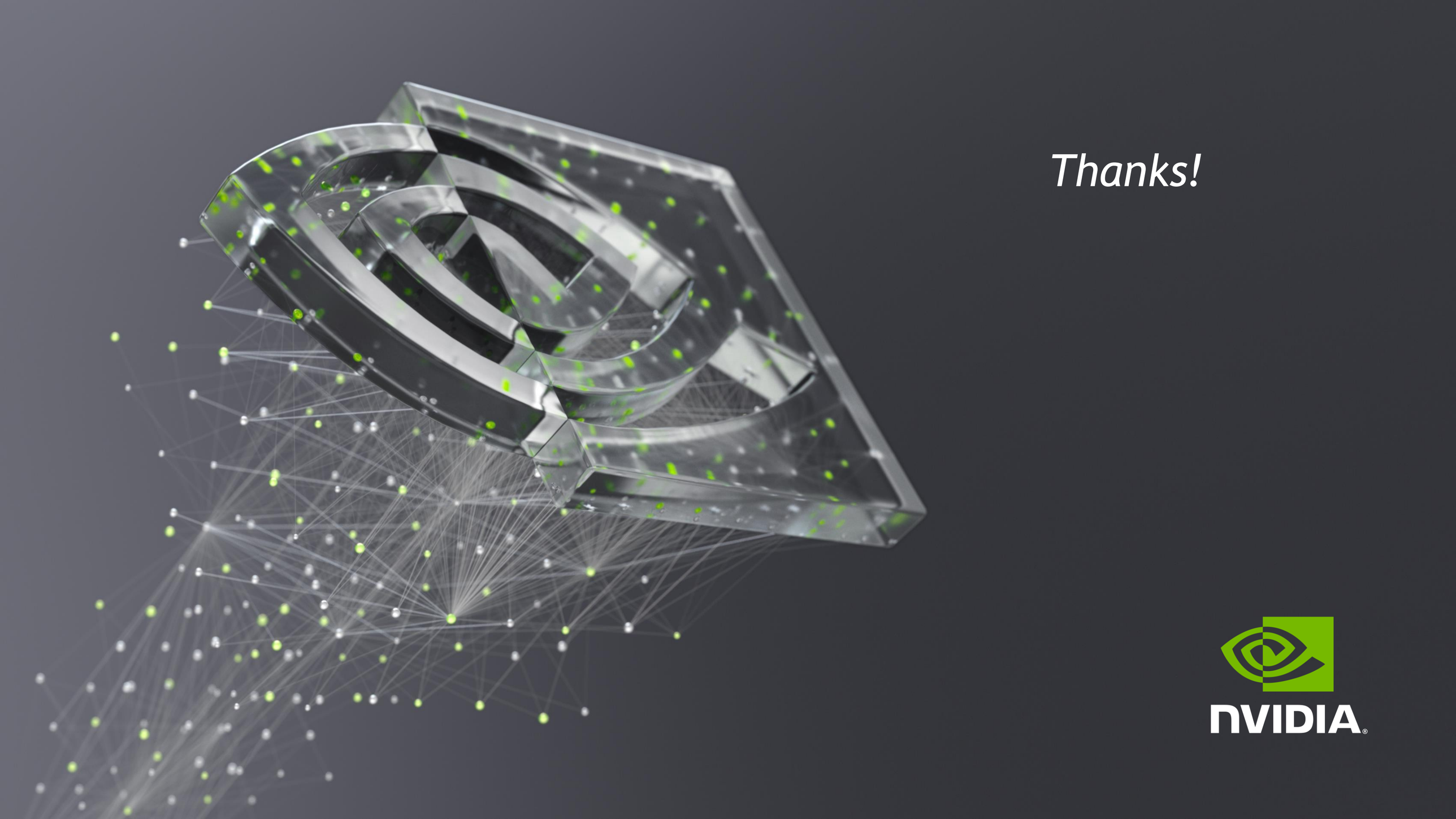
1) Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework, M. Raissi et al.
2) Neural Ordinary Differential Equations, R.T.Q. Chen et al.

IS A ML MODEL USEFUL FOR SCIENCE?



6 STEPS APPROACH





Thanks!

NVIDIA