Foundations of Machine Learning for Physicists

Nat Mathews*
(they/them)

UMD, NASA GSFC

* With knowing and unknowing help from:
Jaye Verniero, Mel Abler, Barbara Thompson, Julie Barnum, Stuart Mumford, James Mason, Monica Bobra, Aurélien Géron, François Chollet, Brandon Rohrer, Theo Wolf and xkcd-font.

Outline

1. The Fundamental Building Blocks

2. Convolutions, Encodering/Decoding and Autoencoders

3. Physics-Informed Neural Networks

Outline

1. The Fundamental Building Blocks

   Tutorial!

2. Convolutions, Encodering/Decoding and Autoencoders

   Tutorial!

3. Physics-Informed Neural Networks

   Tutorial!

Outline

1. The Fundamental Building Blocks

   Tutorial!

2. Convolutions, Encodering/Decoding and Autoencoders

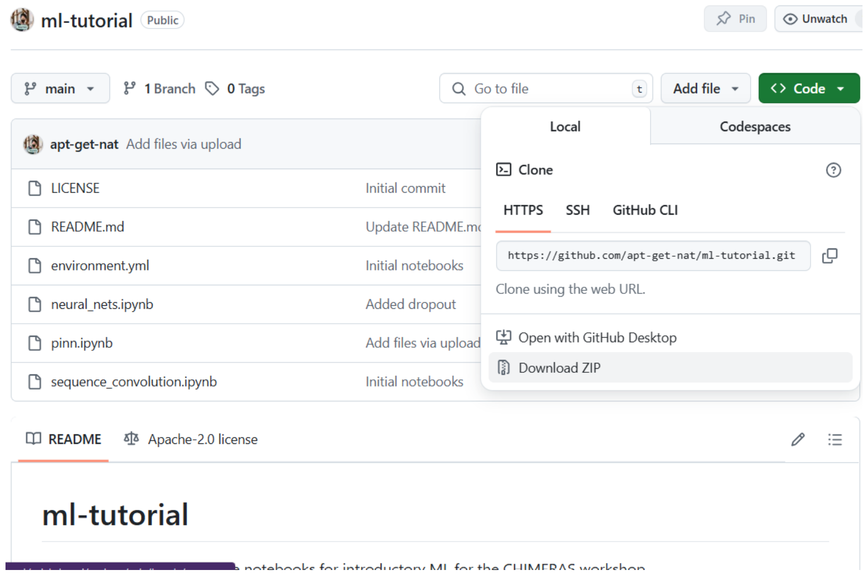
   Tutorial!

3. Panic because we're running out of time

   (you go home tonight run the) Tutorial!

Instructions to follow along

- This is not a workshop; I don't have time to debug everything. But if you want to follow along, you can!
- https://github.com/apt-get-nat/ml-tutorial
- I recommend you download mini-forge if you haven't, and set up the environment. That can take a little while so it's good to do it right away!


ml-tutorial Public

Pin · Unwatch

main · 1 Branch · 0 Tags · Go to file · Add file · Code

apt-get-nat Add files via upload

LICENSE	Initial commit
README.md	Update README.md
environment.yml	Initial notebooks
neural_nets.ipynb	Added dropout
pinn.ipynb	Add files via upload
sequence_convolution.ipynb	Initial notebooks

Local · Codespaces

Clone

HTTPS · SSH · GitHub CLI

```
https://github.com/apt-get-nat/ml-tutorial.git
```

Clone using the web URL.

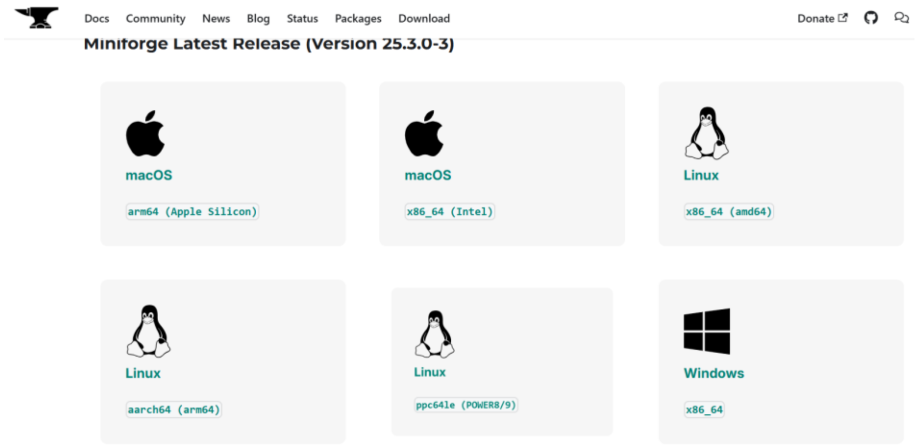
Open with GitHub Desktop

Download ZIP

README · Apache-2.0 license

ml-tutorial

notebooks for introductory ML for the CHIMERAS workshop



Miniforge Latest Release (Version 25.3.0-3)

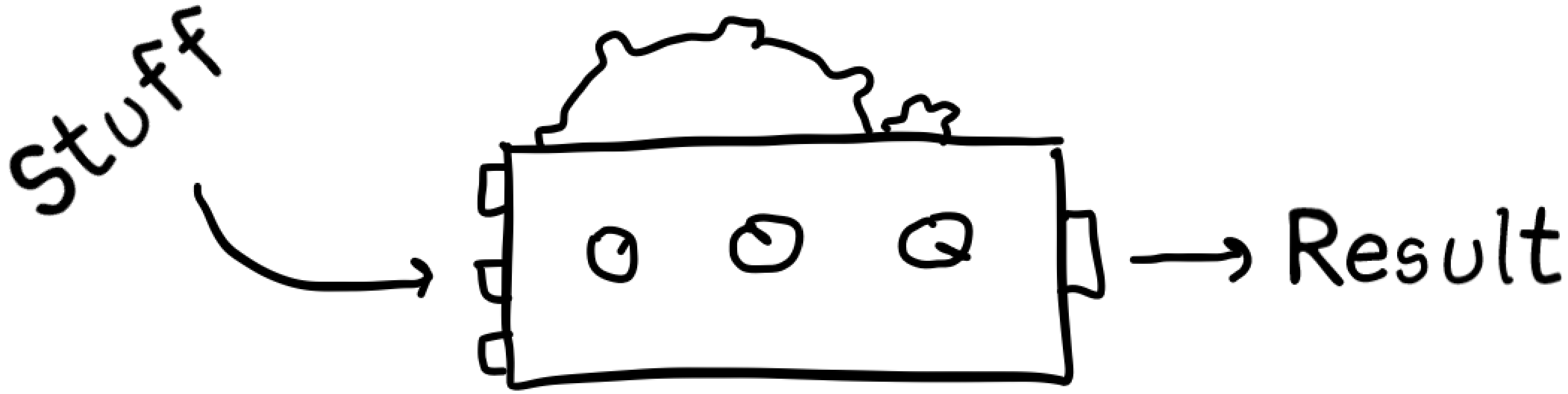
What is AI?

What is Machine Learning?

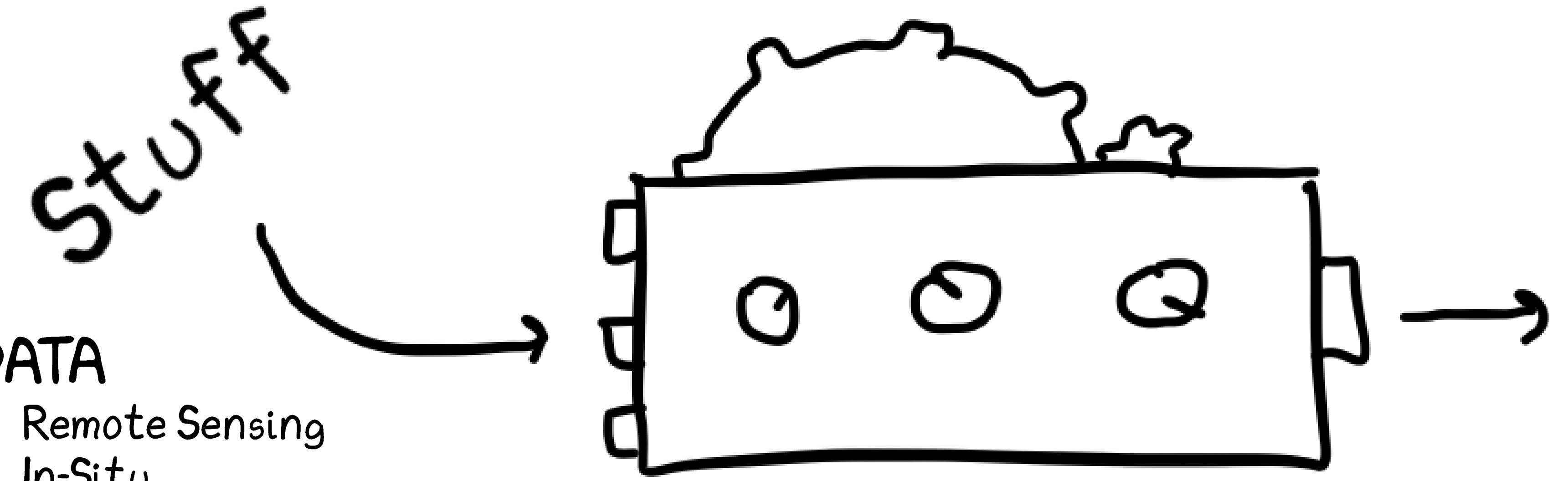
What is a Neural Network?

What is this talk even about?

Machine Learning



Machine Learning

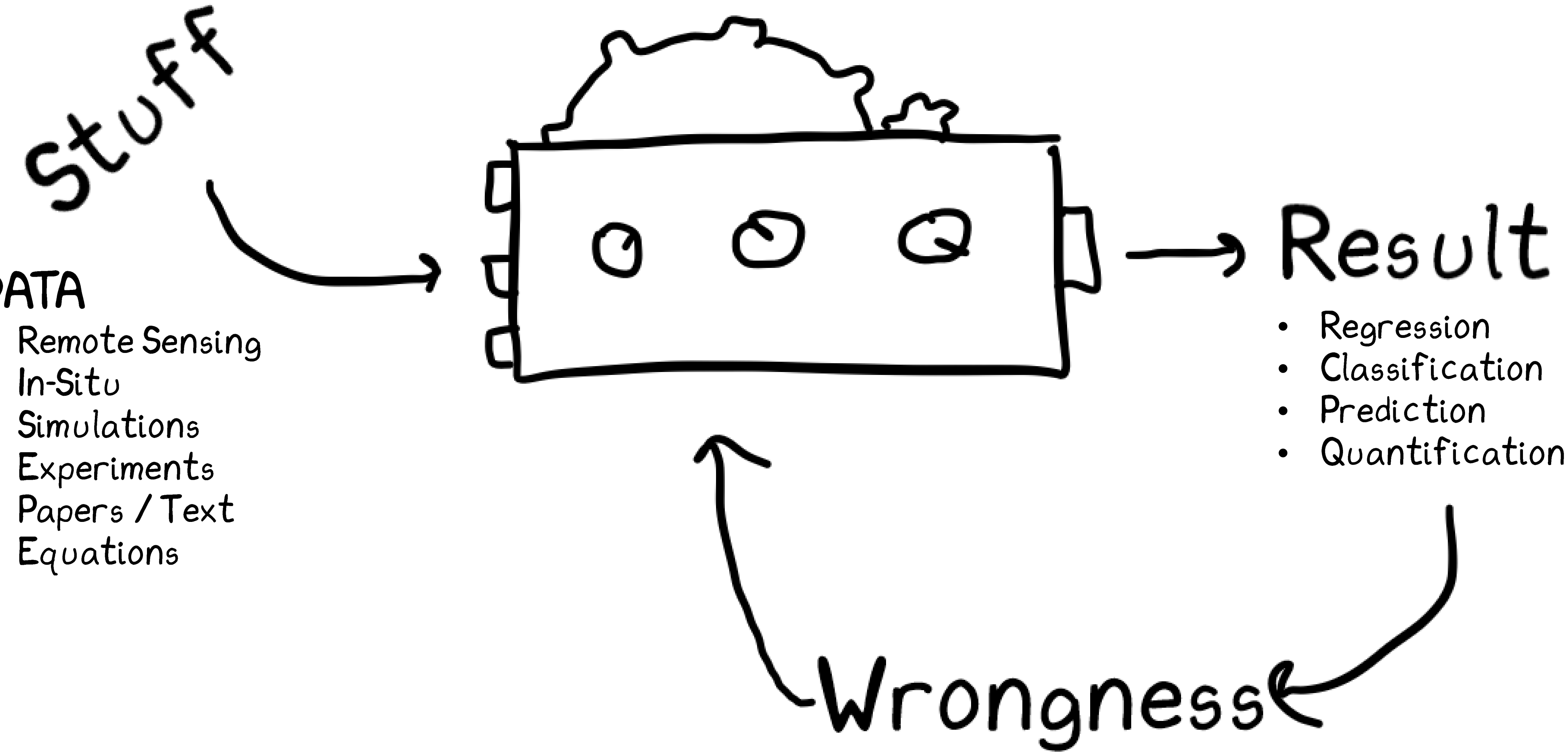


DATA

- Remote Sensing
- In-Situ
- Simulations
- Experiments
- Papers / Text
- Equations

- Regression
- Classification
- Prediction
- Quantification

Machine Learning

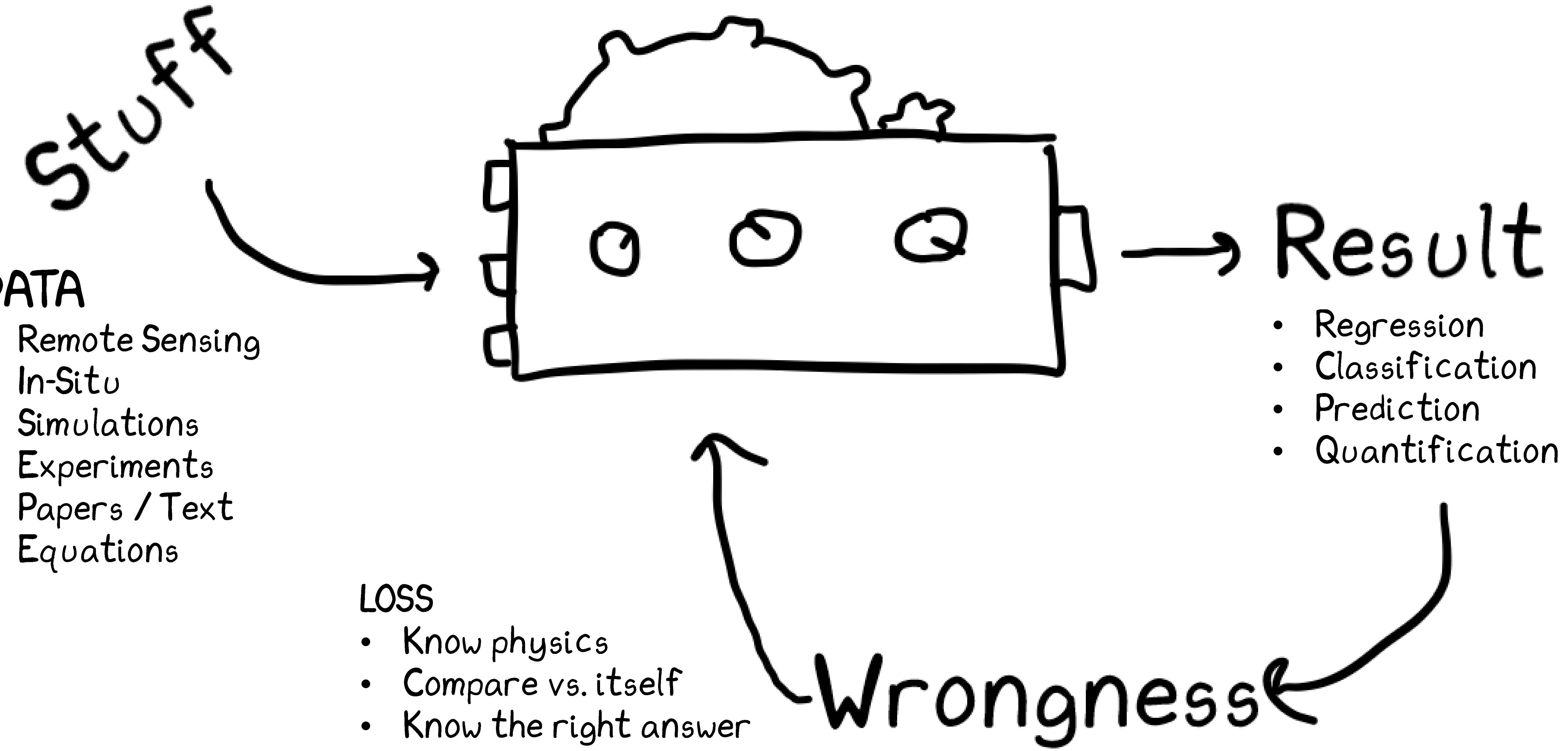


DATA

- Remote Sensing
- In-Situ
- Simulations
- Experiments
- Papers / Text
- Equations

- Regression
- Classification
- Prediction
- Quantification

Machine Learning



DATA

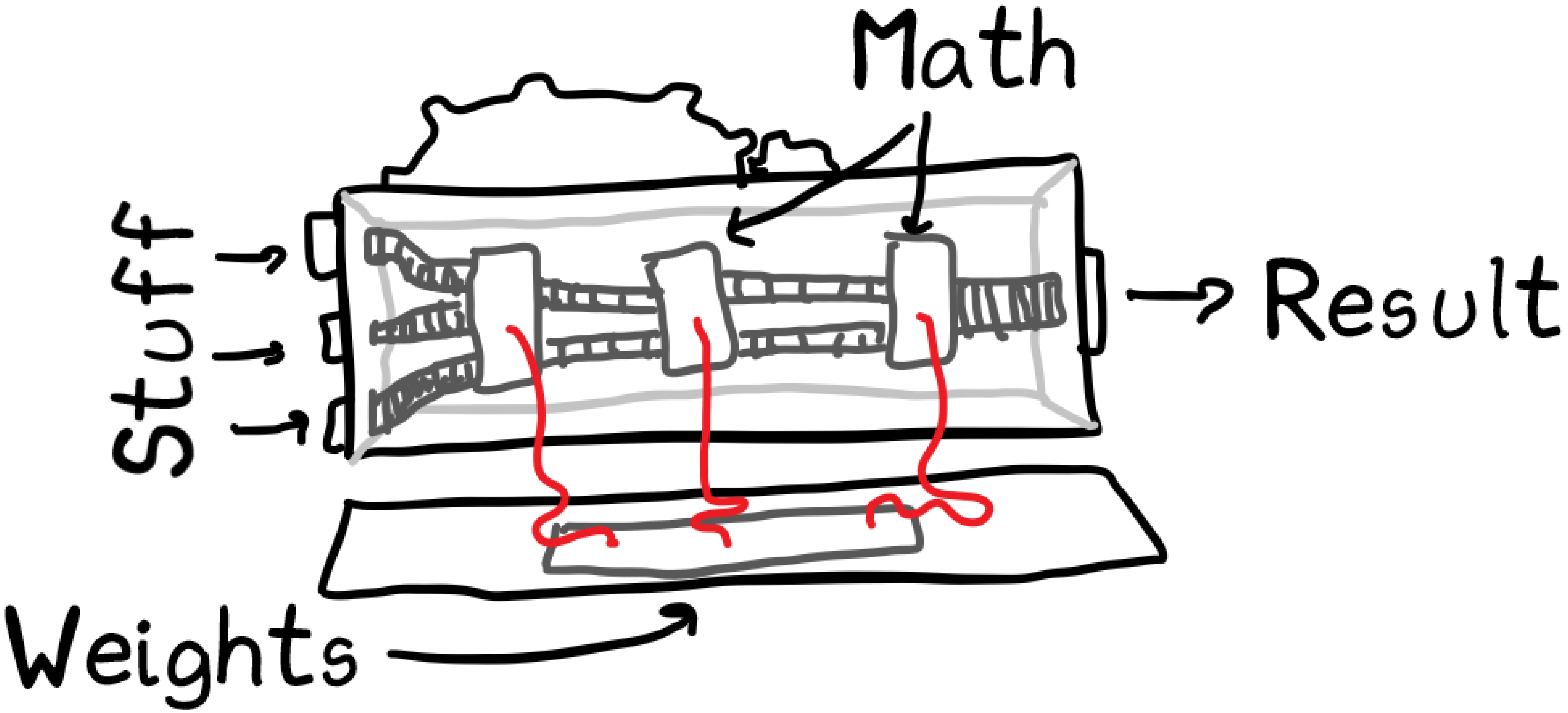
- Remote Sensing
- In-Situ
- Simulations
- Experiments
- Papers / Text
- Equations

- Regression
- Classification
- Prediction
- Quantification

LOSS

- Know physics
- Compare vs. itself
- Know the right answer

Machine Learning



What is a Deep Neural Network?

Neural Networks

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

Neural Networks

$$\sigma\left[\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}\right]$$

Neural Networks

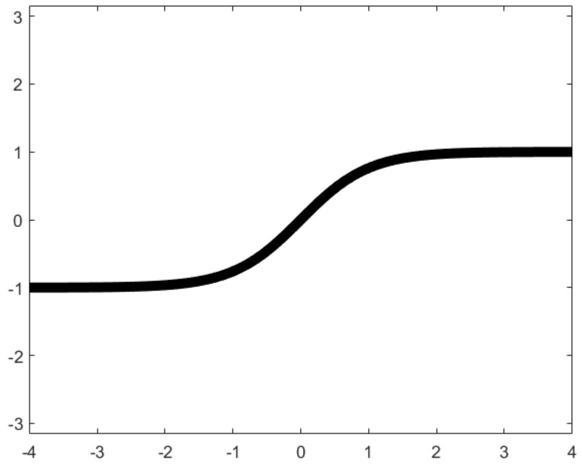
$$\sigma\left[\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}\right] = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

Neural Networks

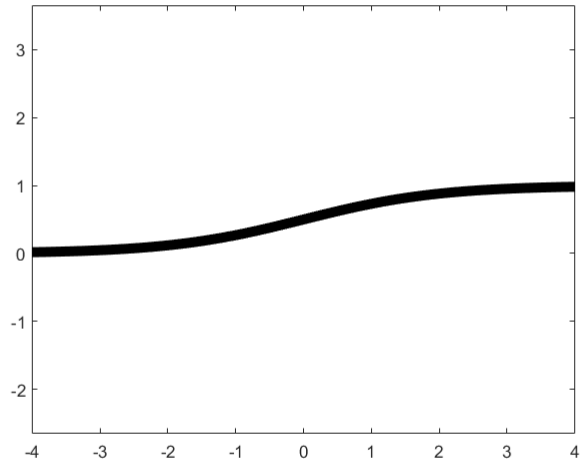
$$\sigma\left[\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}\begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}\right] = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

$\sigma :=$

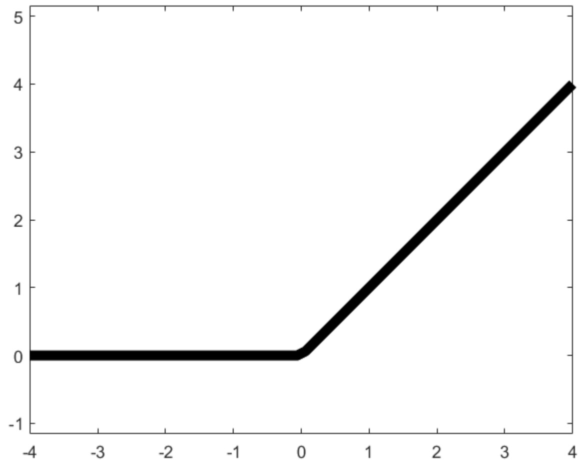
tanh



Logistic Sigmoid



ReLU



Deep Neural Networks

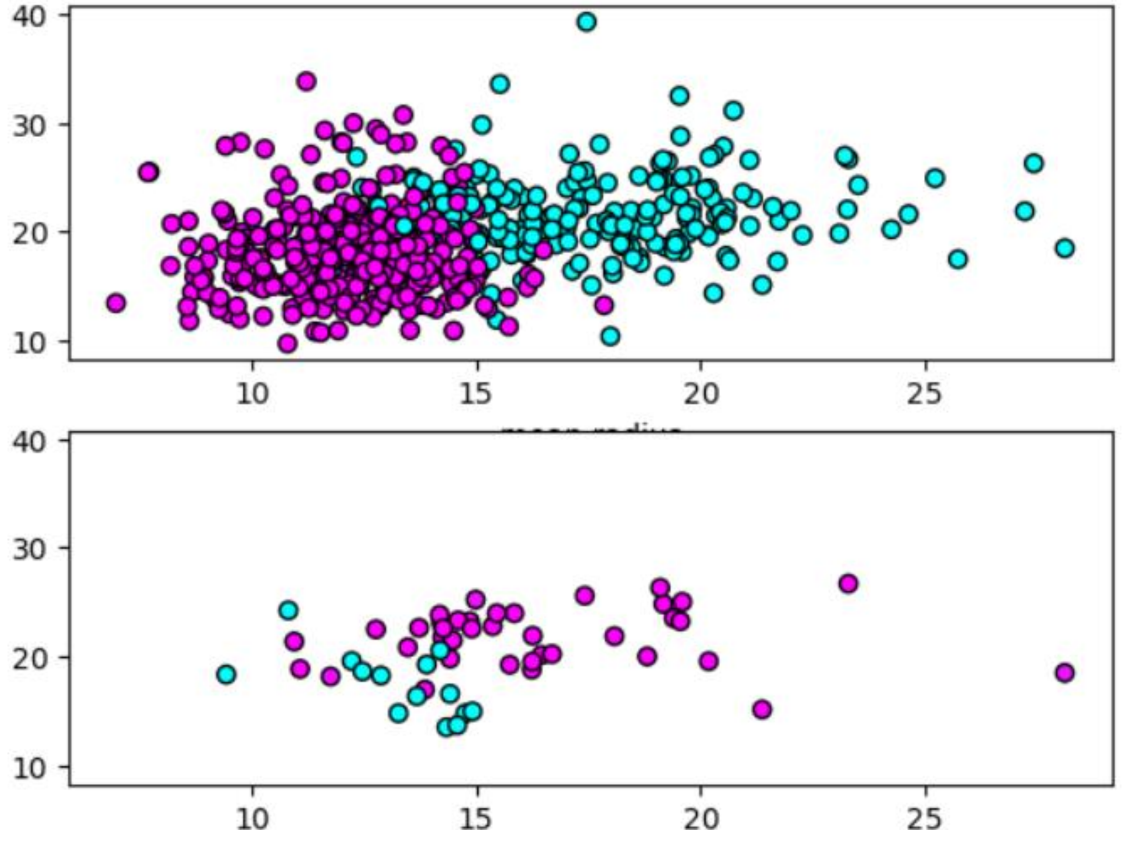
$$\sigma\left[\begin{pmatrix} x_{11} \\ \vdots \\ x_{1n} \end{pmatrix}\begin{pmatrix} w_{111} & \cdots & w_{11n} \\ \vdots & \ddots & \vdots \\ w_{1m1} & \cdots & w_{1mn} \end{pmatrix} + \begin{pmatrix} b_{11} \\ \vdots \\ b_{1m} \end{pmatrix}\right] = \begin{pmatrix} x_{21} \\ \vdots \\ x_{2m} \end{pmatrix}$$

These are both layers

$$\sigma\left[\begin{pmatrix} x_{21} \\ \vdots \\ x_{2m} \end{pmatrix}\begin{pmatrix} w_{211} & \cdots & w_{21m} \\ \vdots & \ddots & \vdots \\ w_{2\ell m} & \cdots & w_{2\ell\ell} \end{pmatrix} + \begin{pmatrix} b_{21} \\ \vdots \\ b_{2m} \end{pmatrix}\right] \overset{\text{So many layers!}}{= \cdots =} \begin{pmatrix} y_{1} \\ \vdots \\ y_{q} \end{pmatrix}$$

Deep Neural Networks: example!

- Classification
  - We'll start with an example, non-physics dataset (cw cancer)
  - Two classes, 30-dimensional input
  - Walk through how to actually set up and train a model
  - Output a scalar [0,1] and convert it back to a class at the end



Deep Neural Networks: example!

First we'll grab the data

```python
import os
os.environ['KERAS_BACKEND'] = 'torch'
import torch
import keras
from keras import layers
import matplotlib.pyplot as plt
import numpy as np
from tqdm.notebook import tqdm
from sklearn.datasets import load_breast_cancer
```

```python
x = data['data'][:-10]
y = data['target'][:-10]
x_test = data['data'][-10:]
y_test = data['target'][-10:]
```

Then we'll build our model

```python
model = keras.Sequential([
    layers.Input(shape=(30,)),
    layers.Dense(20, activation='tanh'),
    layers.Dropout(rate=0.2),
    layers.Dense(12, activation='tanh'),
    layers.Dropout(rate=0.2),
    layers.Dense(4, activation='tanh'),
    layers.Dense(1, activation="sigmoid")
])
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.Adam(),
              metrics=["binary_accuracy"]
             )
model.summary()
```

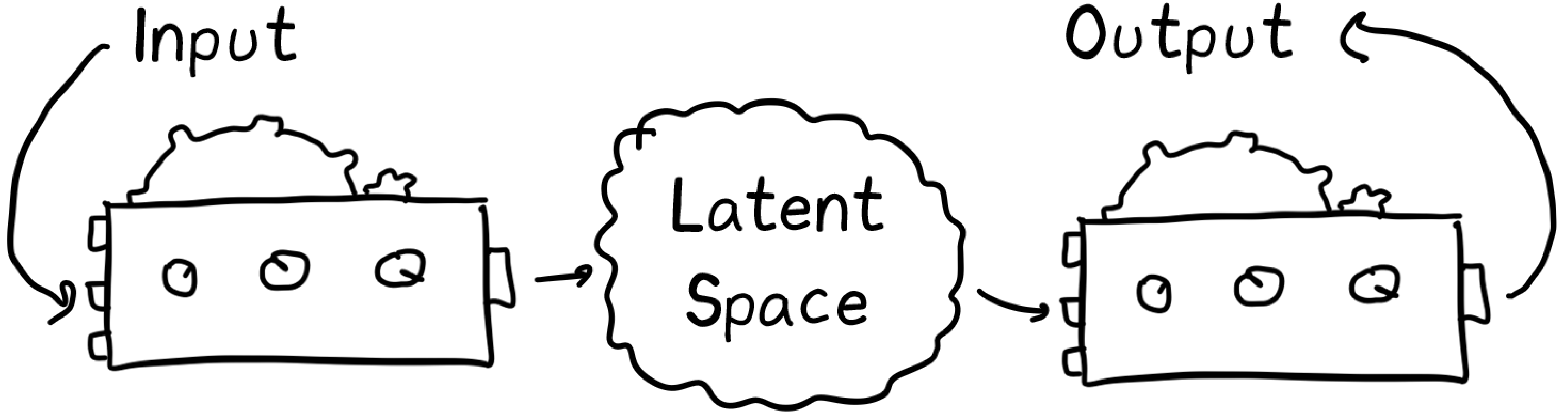
Then we'll train and run it

```python
history = model.fit(x,y,epochs=100,validation_split=0.1)
```

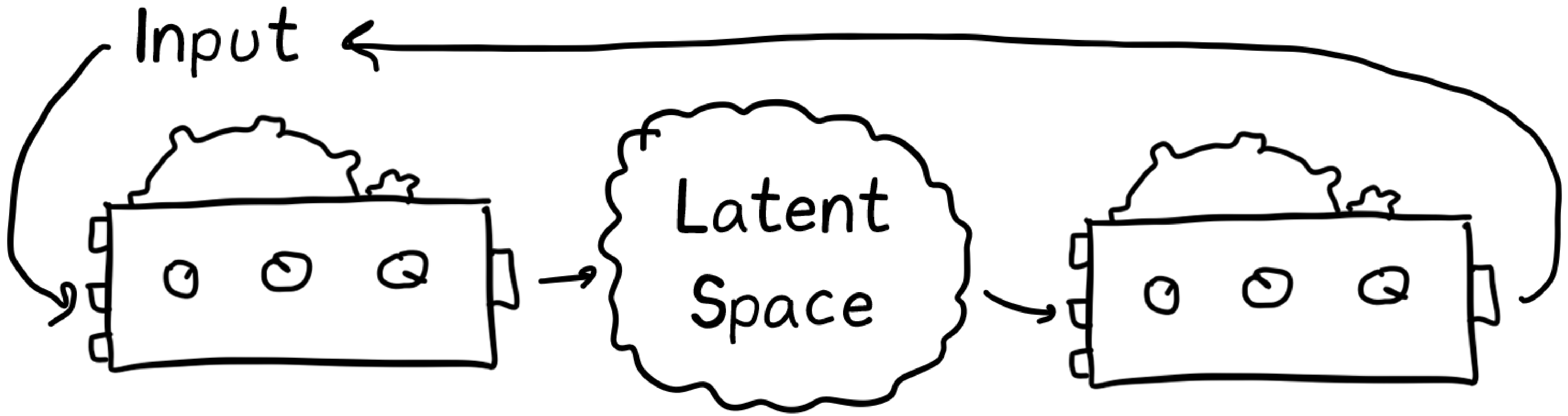
```python
threshold = 0.5

y_pred = np.squeeze(model.predict(x))
y_class = np.array([1 if prediction > threshold
                    else 0 for prediction in y_pred
                   ])
```

What is an autoencoder?

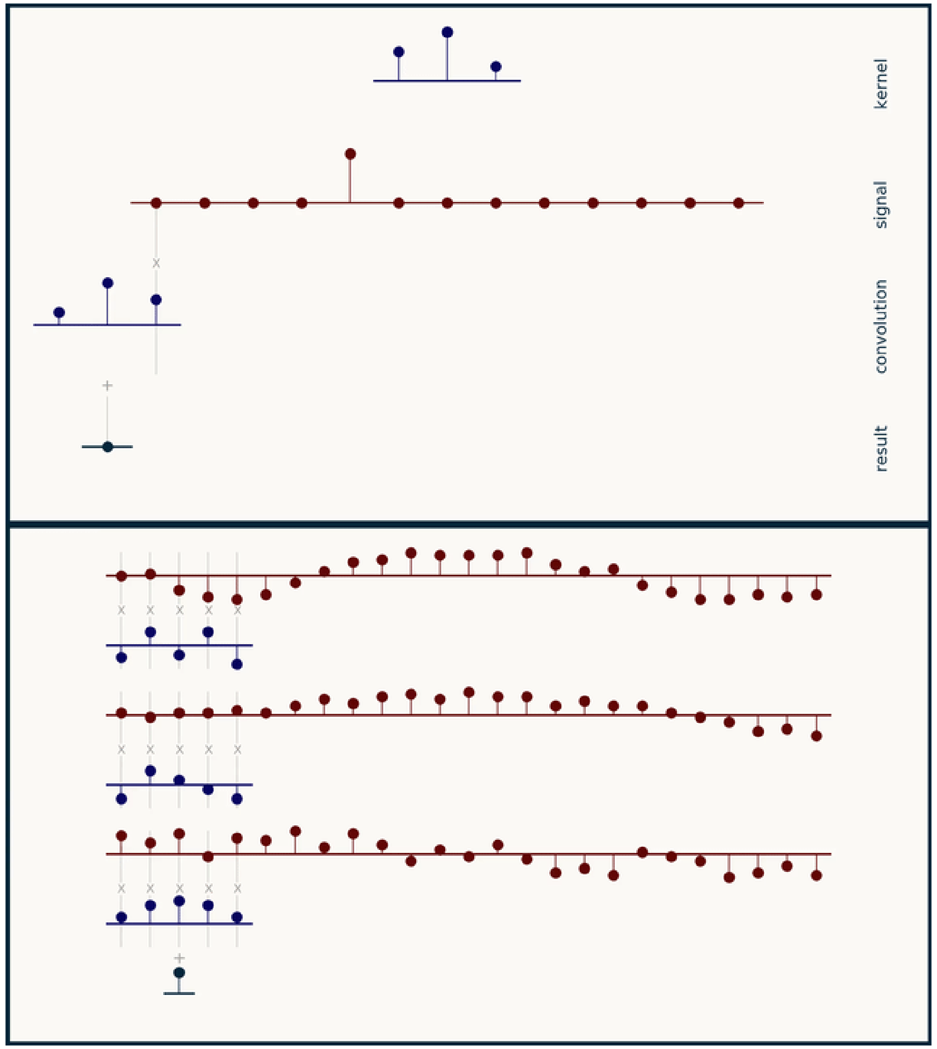
Encoding



Autoencoders



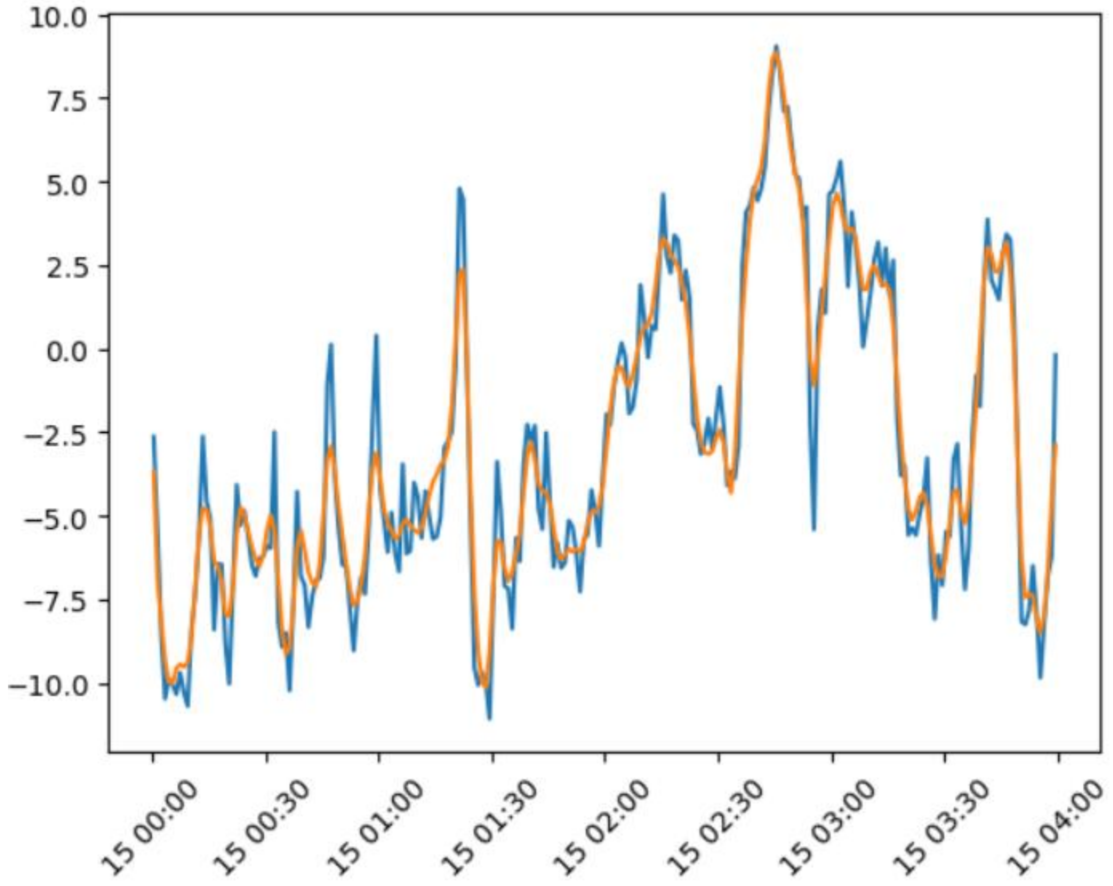
Encoding: Convolutional Layers



- A Convolution Layer passes a moving window over the input (or previous output from the last layer)
- For simple filters, this is a "smoothing" operation
- Convolutions are a good way to create an abstract representation of data when things are continuous, spatial or in time-series
- Think of the Fourier transform!

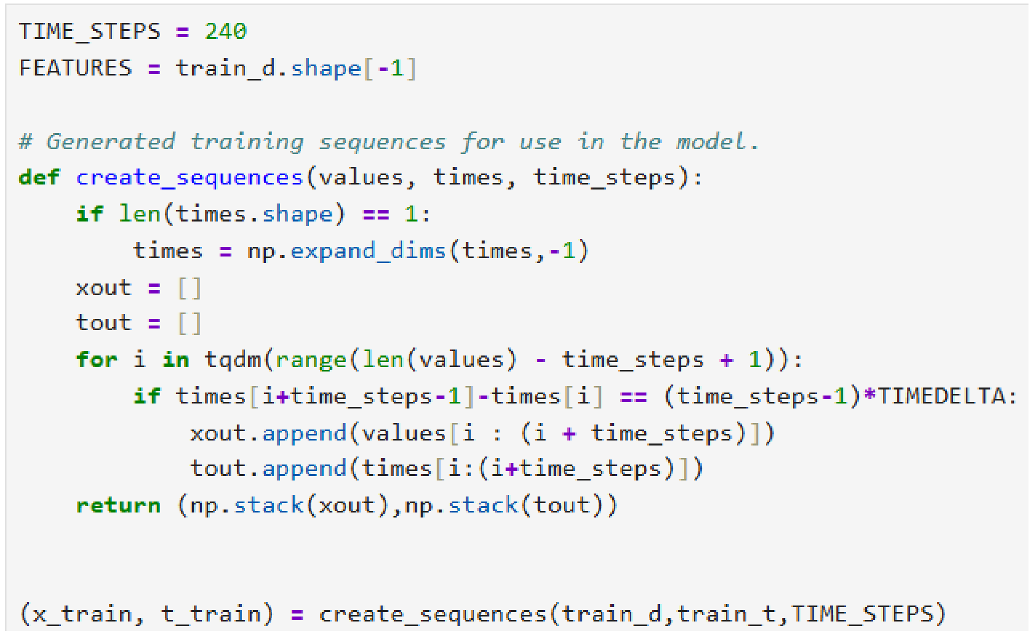
Autoencoding: example!

- Feature Detection
  - We'll train an autoencoder on a bunch of solar wind data from Parker Solar Probe
  - Then we'll test on other data from the next year
  - Look for where the trained autoencoder does badly
  - Hopefully interesting things are happening there!



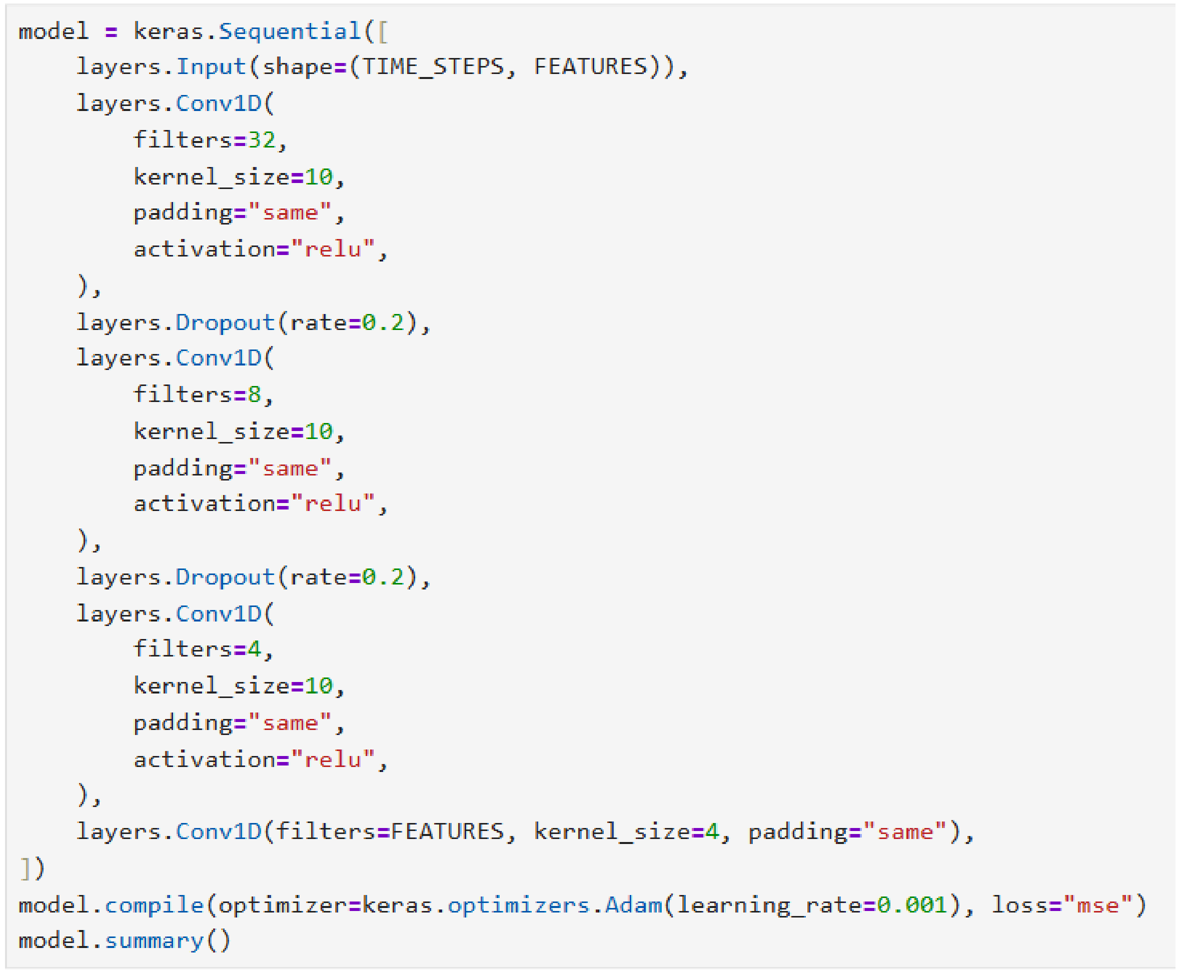
Autoencoding: example!

This time we need to decide how long our windows will be, and slice up the data



```python
TIME_STEPS = 240
FEATURES = train_d.shape[-1]

# Generated training sequences for use in the model.
def create_sequences(values, times, time_steps):
    if len(times.shape) == 1:
        times = np.expand_dims(times,-1)
    xout = []
    tout = []
    for i in tqdm(range(len(values) - time_steps + 1)):
        if times[i+time_steps-1]-times[i] == (time_steps-1)*TIMEDELTA:
            xout.append(values[i : (i + time_steps)])
            tout.append(times[i:(i+time_steps)])
    return (np.stack(xout),np.stack(tout))


(x_train, t_train) = create_sequences(train_d,train_t,TIME_STEPS)
```

And we'll use convolutional layers to encode the time series



```python
model = keras.Sequential([
    layers.Input(shape=(TIME_STEPS, FEATURES)),
    layers.Conv1D(
        filters=32,
        kernel_size=10,
        padding="same",
        activation="relu",
    ),
    layers.Dropout(rate=0.2),
    layers.Conv1D(
        filters=8,
        kernel_size=10,
        padding="same",
        activation="relu",
    ),
    layers.Dropout(rate=0.2),
    layers.Conv1D(
        filters=4,
        kernel_size=10,
        padding="same",
        activation="relu",
    ),
    layers.Conv1D(filters=FEATURES, kernel_size=4, padding="same"),
])
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mse")
model.summary()
```

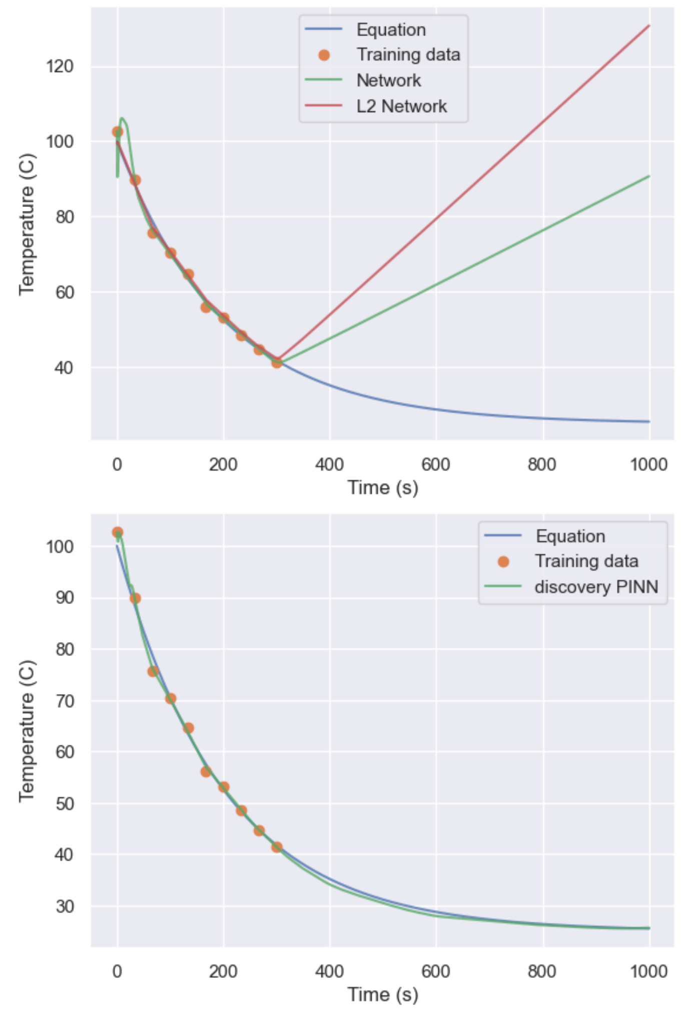
Physics-Informed Neural Networks

Physics-Informed Neural Networks: PINNs

- We've talked about training to labeled data (supervised)
- We've talked about training to reproduce the input (unsupervised autoregression)
- Let's talk about training if you know the RULES (Physics-Informed Neural Networks)
    - Developed by Maziar Raissi

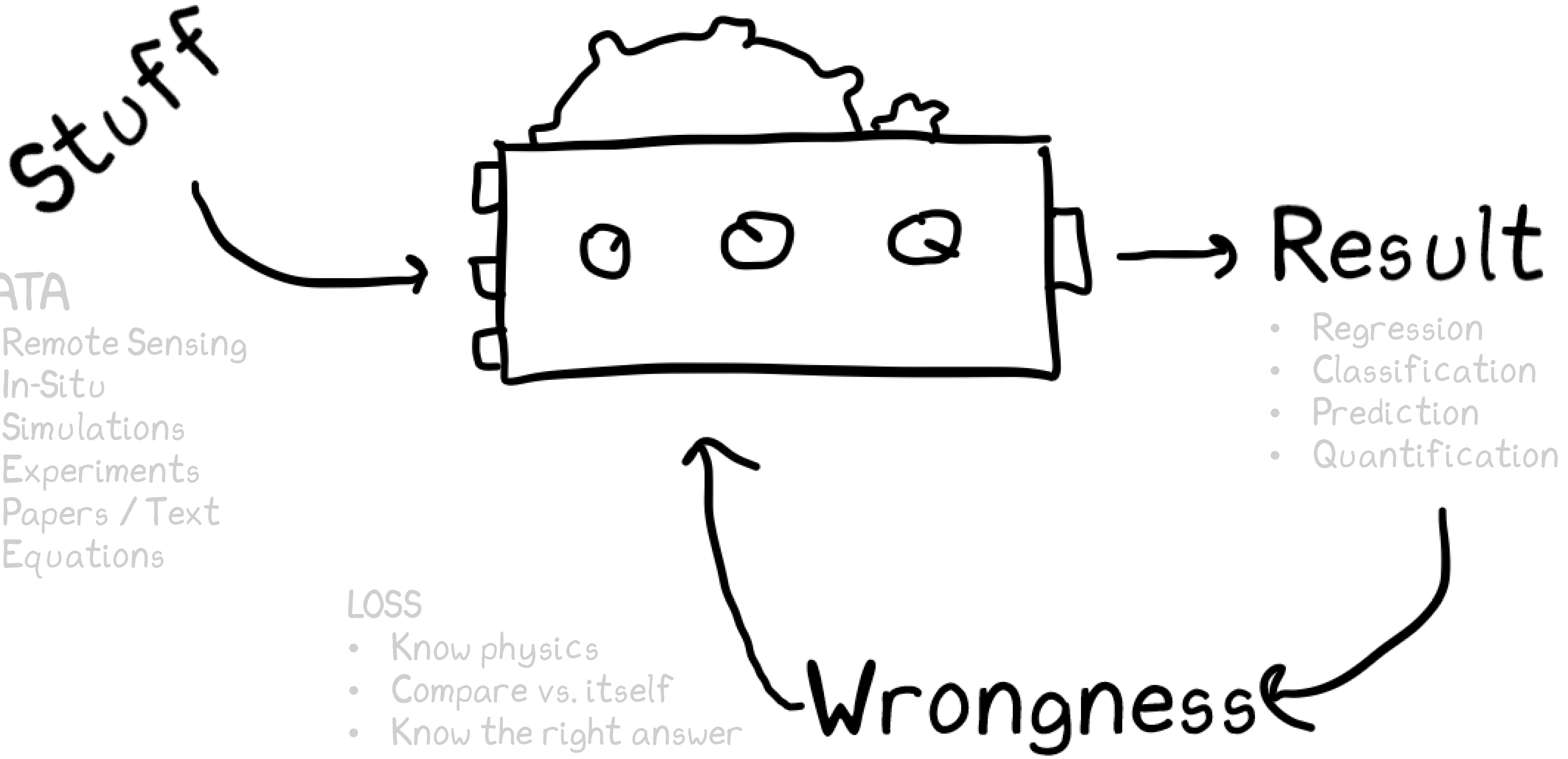
Physics-Informed Neural Networks

- Normally, neural networks for physics can be *very expensive* because making training data is hard
  - Often experiments or simulation
- Plus, we need to carpet the full parameter space
  - (Unregularized) Neural Networks are bad at extrapolating
- We can solve both problems by putting the physical equations into the loss function



Wolf's models of a cooling coffee cup: two standard networks and a PINN

Machine Learning (remember this?)



DATA

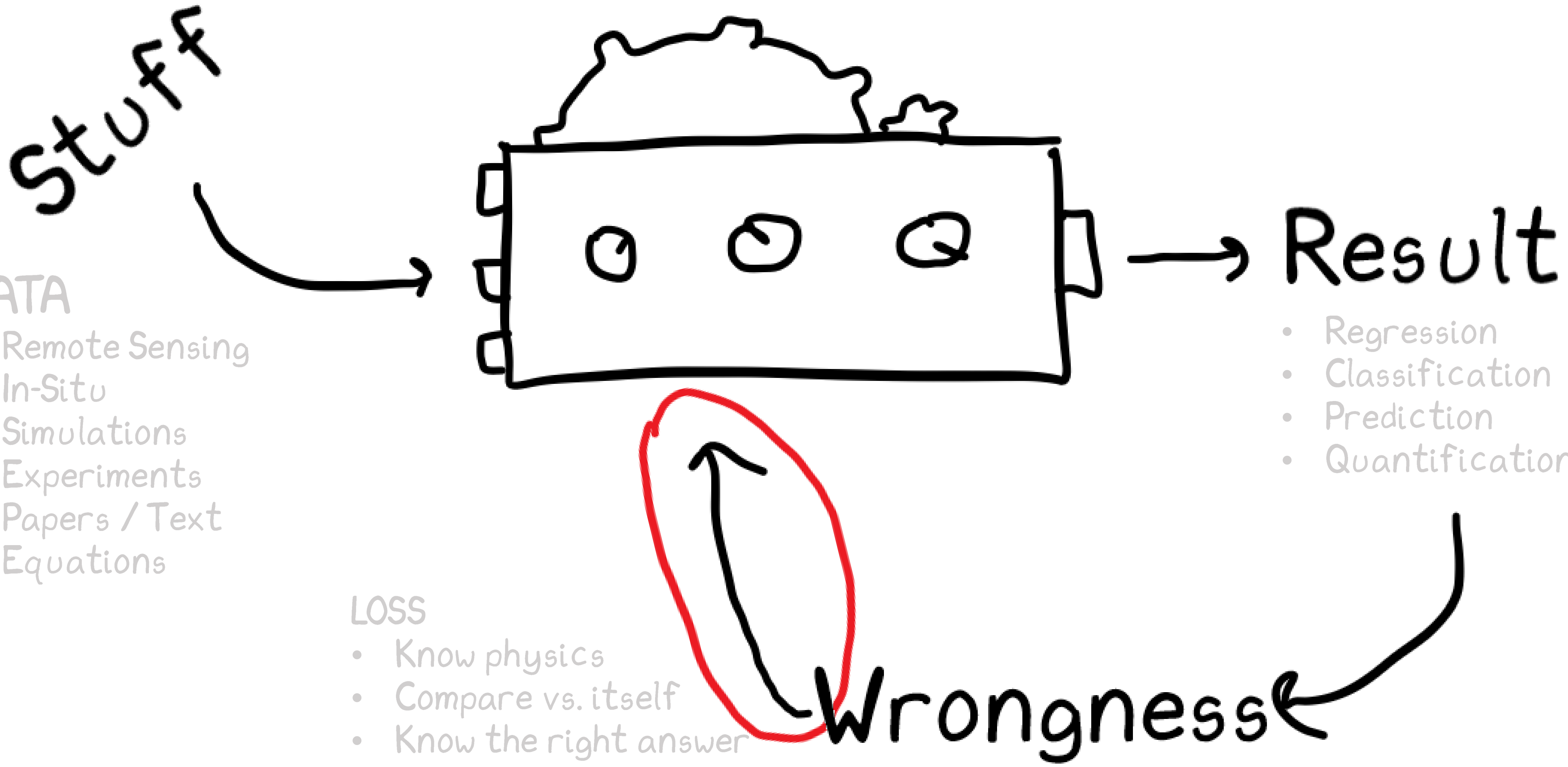
- Remote Sensing
- In-Situ
- Simulations
- Experiments
- Papers / Text
- Equations

- Regression
- Classification
- Prediction
- Quantification

LOSS

- Know physics
- Compare vs. itself
- Know the right answer

Machine Learning (remember this?)



DATA

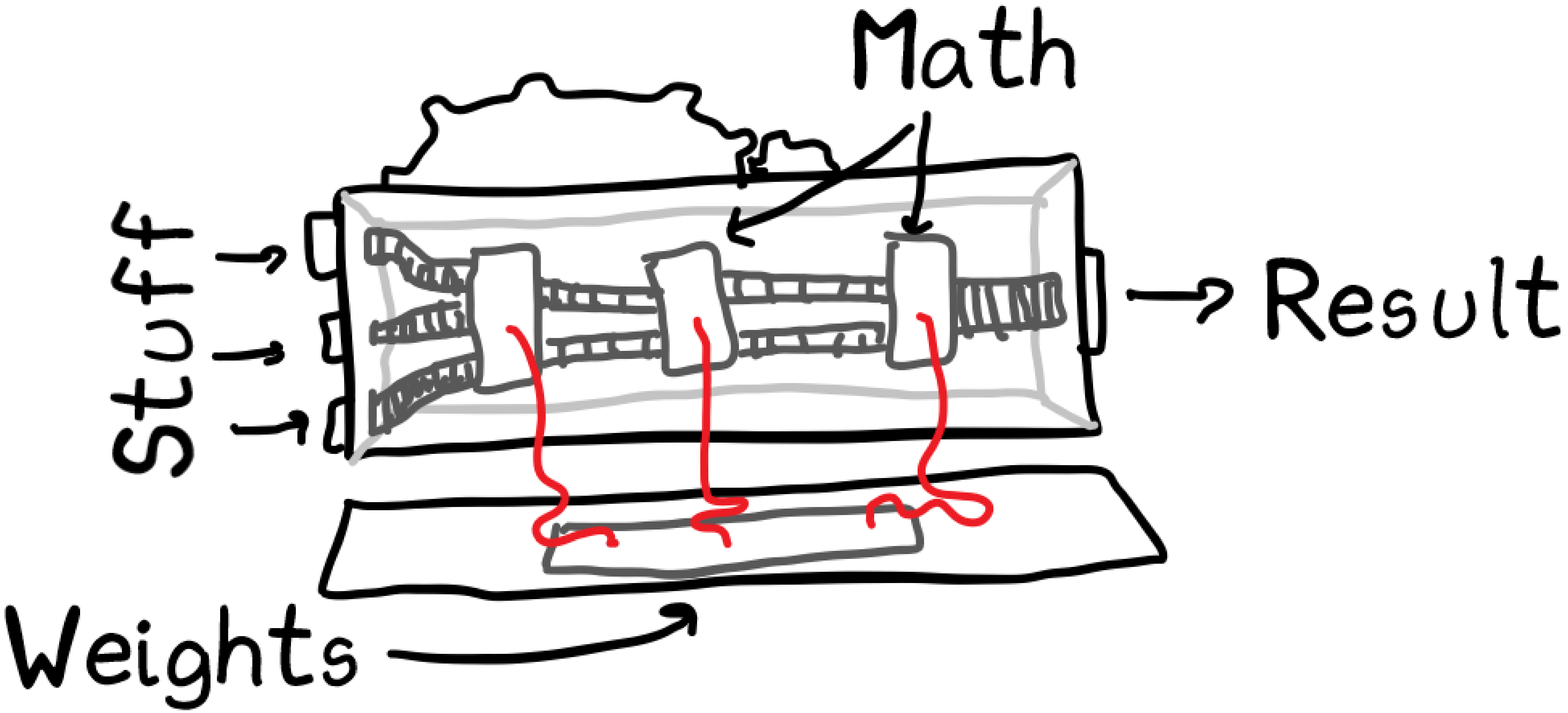
- Remote Sensing
- In-Situ
- Simulations
- Experiments
- Papers / Text
- Equations

- Regression
- Classification
- Prediction
- Quantification

LOSS

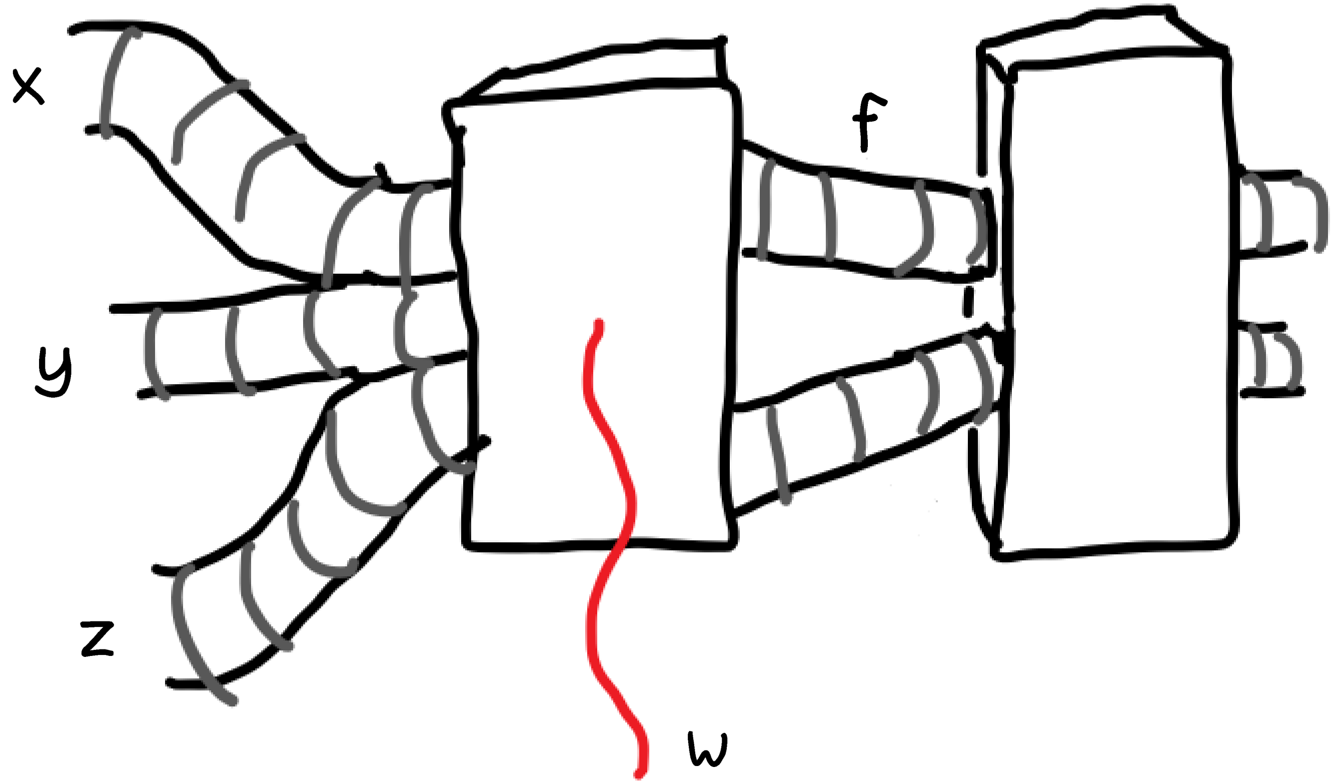
- Know physics
- Compare vs. itself
- Know the right answer

Auto-differentiation



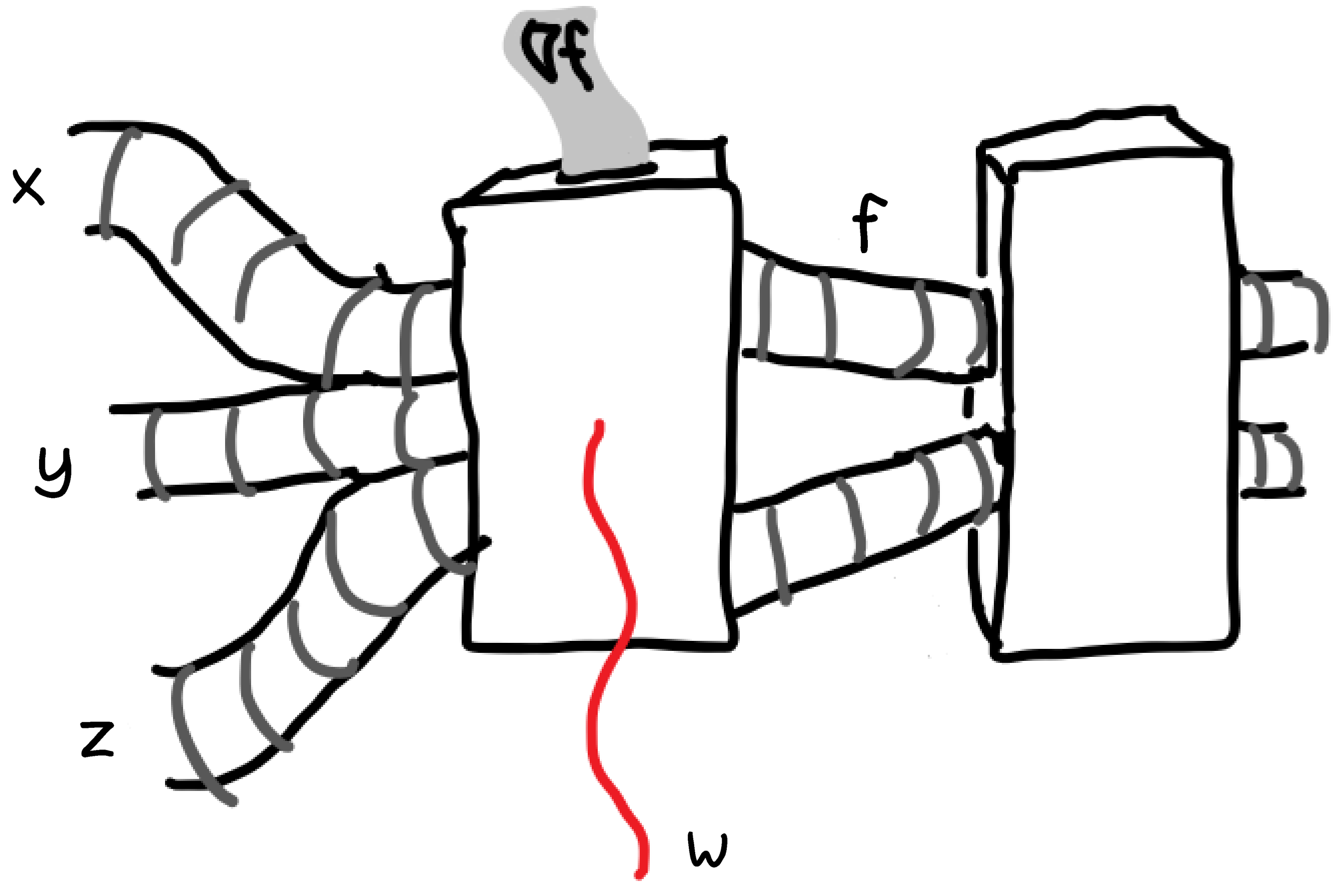
Auto-differentiation

- Inside the box is a pipeline that combines inputs to each layer with weights



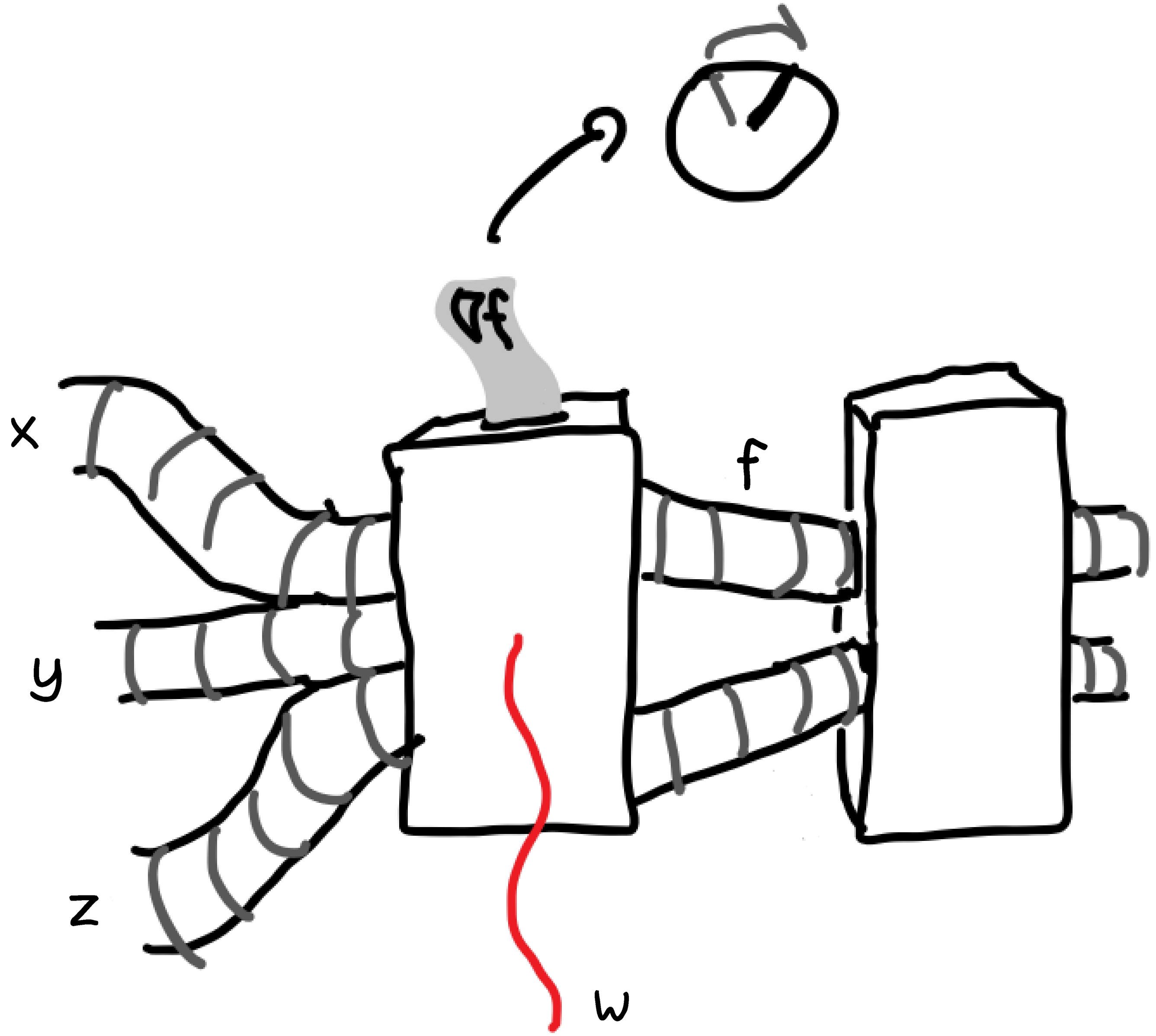
Auto-differentiation

- Inside the box is a pipeline that combines inputs to each layer with weights
- The derivatives at each step get logged



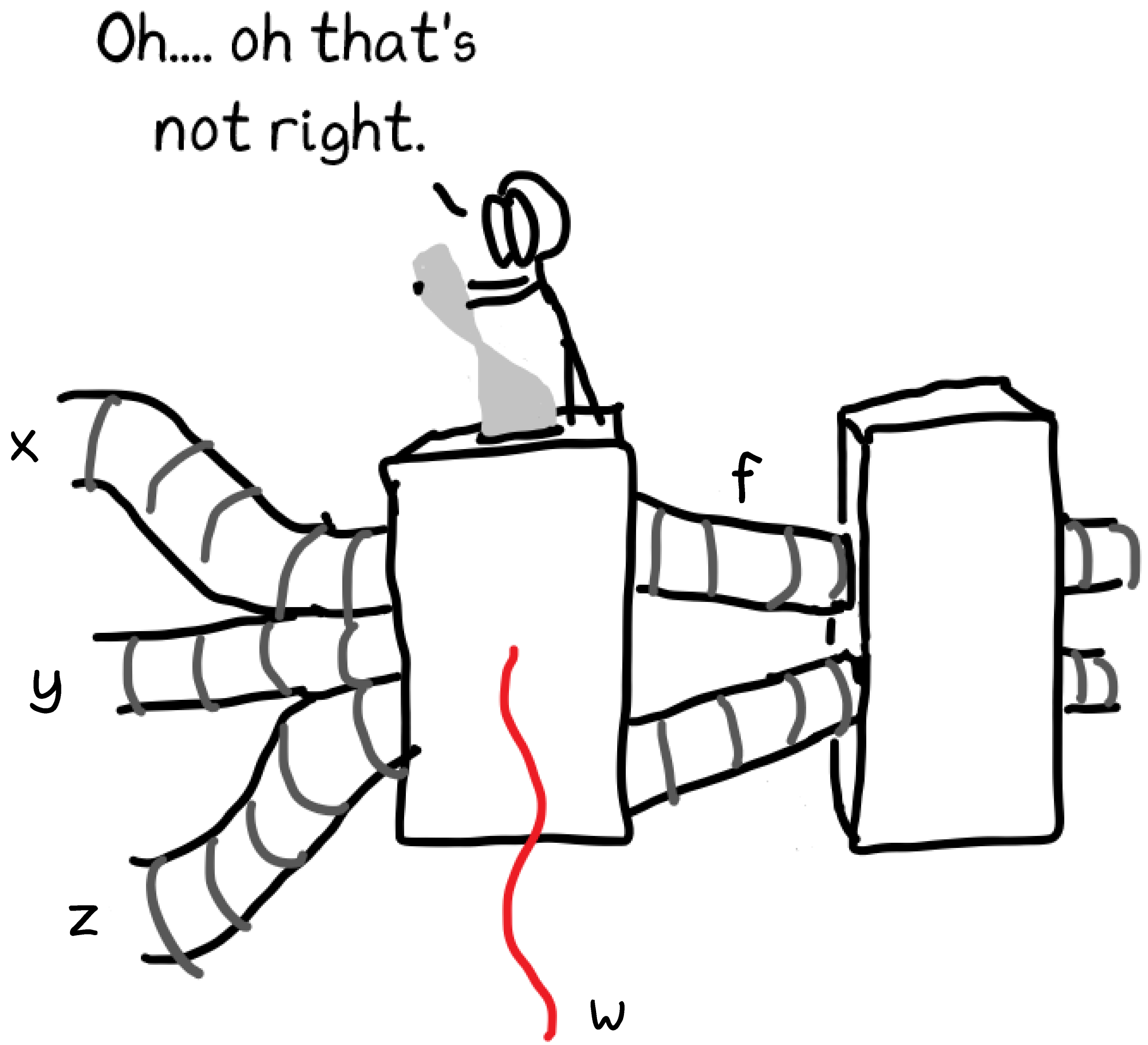
Auto-differentiation

- Inside the box is a pipeline that combines inputs to each layer with weights
- The derivatives at each step get logged
- This log is how we know how to tune the weights



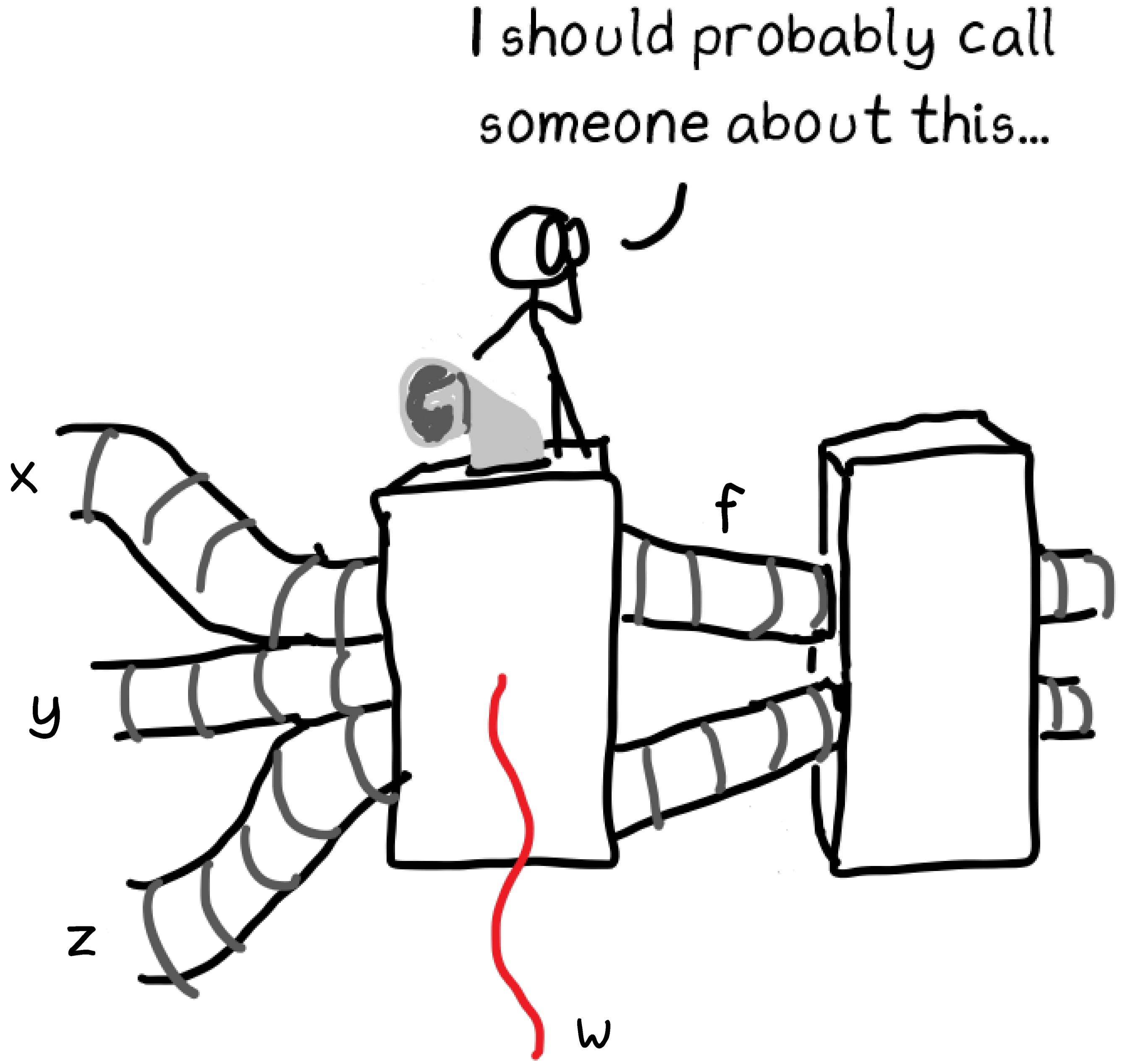
Auto-differentiation

- Inside the box is a pipeline that combines inputs to each layer with weights
- The derivatives at each step get logged
- This log is how we know how to tune the weights
- But we can take advantage of this to take physical derivatives we care about!



Physics-Informed Neural Networks

- We can use these physical derivatives to build our differential equation's residual
- Put that residual in the loss function wherever we don't have known data
- Physics will 'supervise' our training FOR us!



PINNs: example!

- 1D Burger's Equation

$$u_t + uu_x - \left(\frac{v}{\pi}\right) u_{xx} = 0, x \in [-1,1], t \in [0,1]$$

$$u(0,x) = -\sin(\pi x)$$

$$u(t,-1) = u(t,1) = 0$$

- We'll use torch's autodifferentiation abilities, but in a way that makes keras unhappy
  - We'll have to write our own training loop.
  - We won't do any of the fancy validation and batch-shuffling keras was doing, just to keep things simple

PINNs: example!

First we set up some data points

- Want boundaries, and some randomly sampled interior points

```
Nbd = 100
N = 5000

xt_bd = np.vstack((
    np.vstack((np.linspace(-1,1,Nbd),np.zeros(Nbd))).transpose(),
    np.vstack((-np.ones(Nbd),np.linspace(1/Nbd,1,Nbd))).transpose(),
    np.vstack((np.ones(Nbd),np.linspace(1/Nbd,1,Nbd))).transpose()
),dtype=np.float32)
u_bd = np.hstack((
    -np.sin(np.pi*np.linspace(-1,1,Nbd)),
    np.zeros(2*Nbd)
),dtype=np.float32)
v = 0.01

sampler = LatinHypercube(2)
xt = sampler.random(n=N)
xt[:,0] = 2*xt[:,0]-1

xt = np.vstack((xt_bd,xt),dtype=np.float32)
```

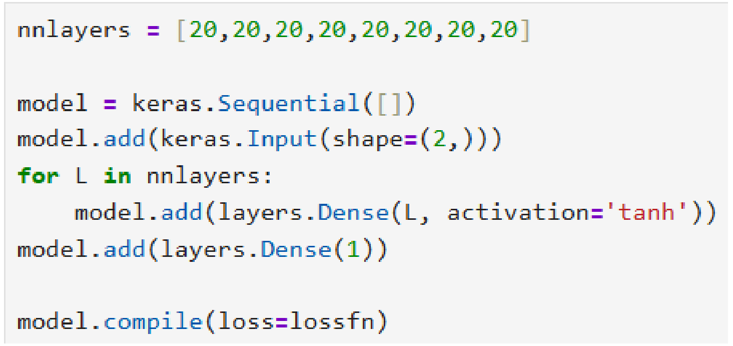
We define a custom loss function

- We use physics to put Burger's equation in, and look at boundary errors

```
def lossfn(y_true,y_pred):
    bd_loss = torch.sum(keras.losses.mean_squared_error(y_true,y_pred))/(3*Nbd)

    xt_tensor = torch.tensor(xt,requires_grad=True, device=y_pred.device)
    xt_tensor.grad = None
    u = model(xt_tensor).squeeze()
    xt_grad = torch.autograd.grad(
        u,xt_tensor,grad_outputs=torch.ones(u.shape,device=u.device),
        retain_graph=True,create_graph=True
    )[0]

    du_dx = xt_grad[:,0]
    du_dt = xt_grad[:,1]
    xt_grad2 = torch.autograd.grad(
        du_dx,xt_tensor,grad_outputs=torch.ones(u.shape,device=u.device),
        retain_graph=True
    )[0]

    d2u_dx2 = xt_grad2[:,0]

    residual = du_dt + u * du_dx - ( v / np.pi) * d2u_dx2
    phys_loss = torch.sum(torch.pow(residual,2))/N

    return 5*bd_loss + phys_loss
```

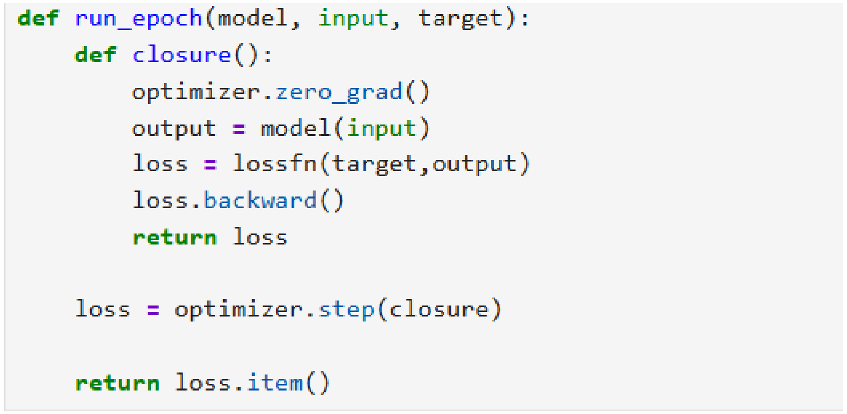
PINNs: example!

We'll use a very simple neural network - our simplest yet!

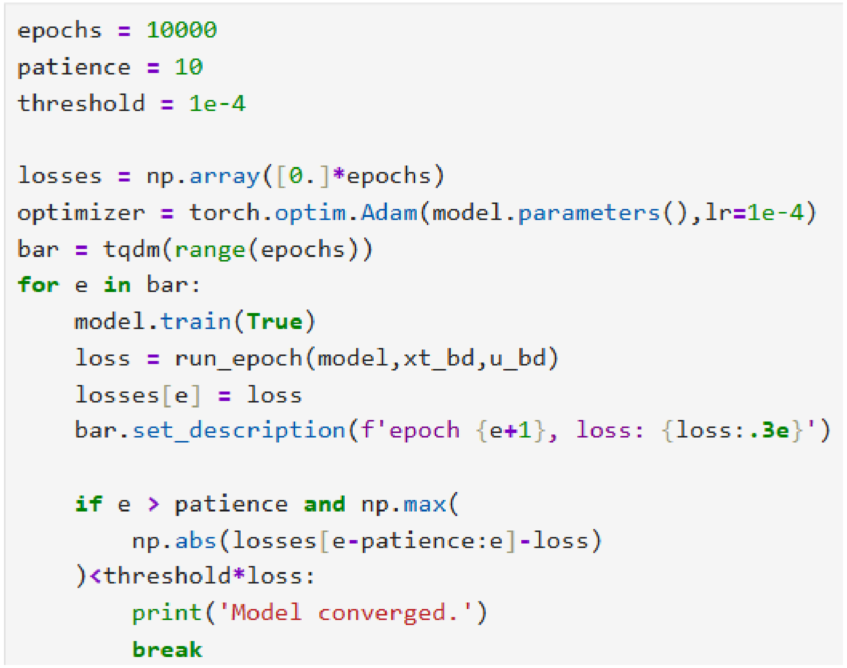


```python
nnlayers = [20,20,20,20,20,20,20,20]

model = keras.Sequential([])
model.add(keras.Input(shape=(2,)))
for L in nnlayers:
    model.add(layers.Dense(L, activation='tanh'))
model.add(layers.Dense(1))

model.compile(loss=lossfn)
```

Then comes our custom training loop - this is real pytorch



```python
def run_epoch(model, input, target):
    def closure():
        optimizer.zero_grad()
        output = model(input)
        loss = lossfn(target,output)
        loss.backward()
        return loss

    loss = optimizer.step(closure)

    return loss.item()
```



```python
epochs = 10000
patience = 10
threshold = 1e-4

losses = np.array([0.]*epochs)
optimizer = torch.optim.Adam(model.parameters(),lr=1e-4)
bar = tqdm(range(epochs))
for e in bar:
    model.train(True)
    loss = run_epoch(model,xt_bd,u_bd)
    losses[e] = loss
    bar.set_description(f'epoch {e+1}, loss: {loss:.3e}')

    if e > patience and np.max(
        np.abs(losses[e-patience:e]-loss)
    )<threshold*loss:
        print('Model converged.')
        break
```

Conclusion

- Machine Learning uses very simple components that we strap together in new ways
- It can be useful in a variety of contexts
    - Classifying populations with lots of parameters (we didn't even talk about the right way to do this, kd-clustering)
    - Prediction and regression
    - Solving equations
    - Finding rare events in the noise
    - Anything you want to be FAST and DIRTY
- The best way to learn is by doing!

Thanks!