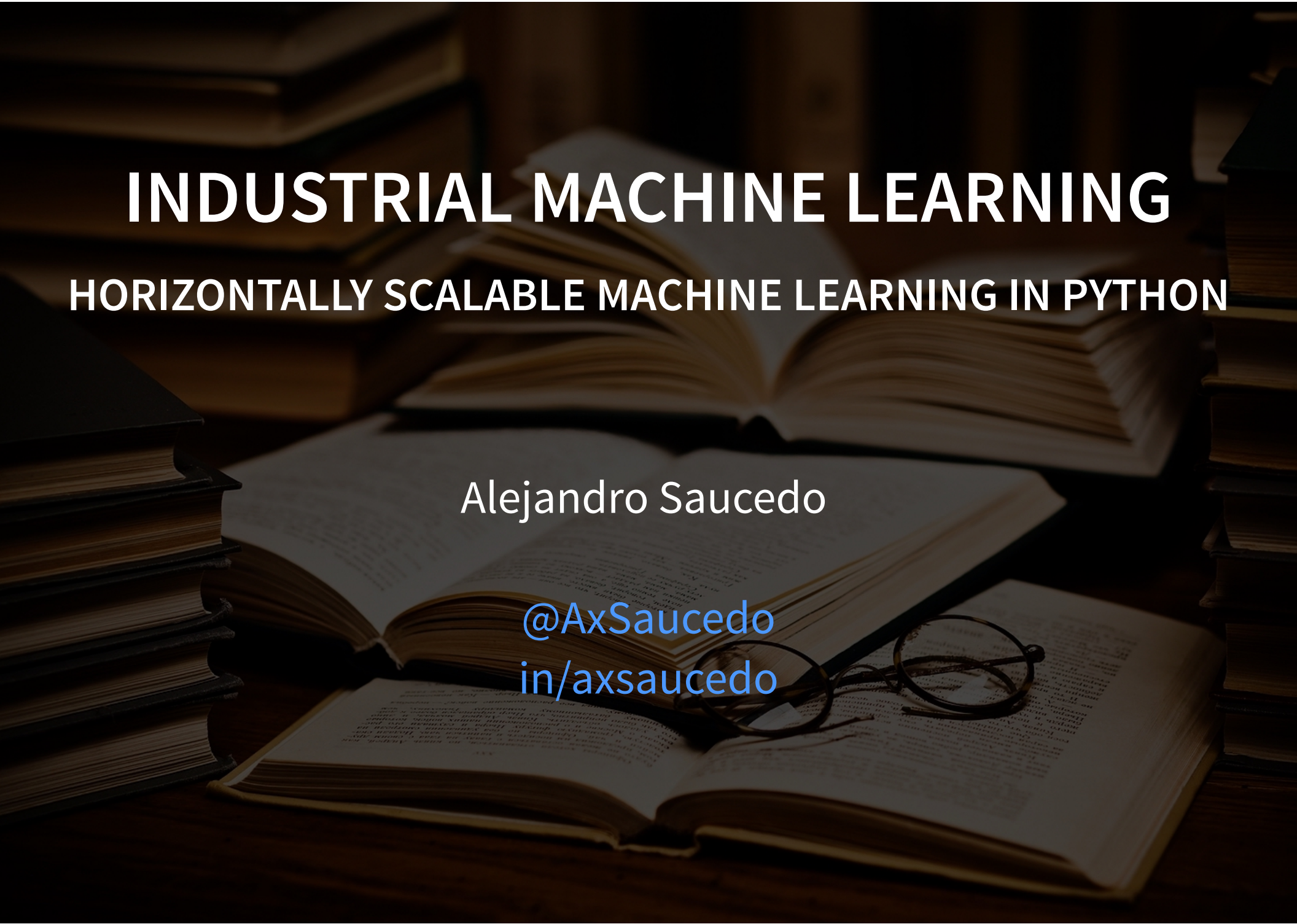


INDUSTRIAL MACHINE LEARNING

HORIZONTALLY SCALABLE MACHINE LEARNING IN PYTHON

Alejandro Saucedo

@AxSaucedo
in/axsaucedo

INDUSTRIAL MACHINE LEARNING

Alejandro Saucedo

CTO, Exponential Tech
Head of Dep. Eng., Eigen
Fellow (AI & ML), The RSA

INDUSTRY-READY ML

An intuitive overview on general ML

A set of simple ML and heavier DL models

A distributed ML architecture

An elastically scalable setup

THE BIG PICTURE


LEARNING BY EXAMPLE
Today we are...
BUILDING A TECH STARTUP


CRYPTO-ML LTD.
Let's jump the hype-train!
A ML framework for crypto-data
Supporting heavy compute/memory ML
Can our system survive the crypto-craze?

THE DATASET

All historical data from top 100 cryptos

Data goes from beginning to 11/2017

563871 daily-price (close) points

Objectives:

- *Supporting heavy ML computations*
- *Supporting increasing traffic*

INTERFACE: CRYPTOLOADER

```
from crypto_ml.data_loader import CryptoLoader cl

loader = cl()

loader.get_prices("bitcoin")
> array([ 134.21,  144.54,  139.  , ..., 3905.95, 3631.04, 3630.7 ])


loader.get_df("bitcoin").head()
>           Date    Open    High     Low   Close      Market Cap
> 1608 2013-04-28  135.30  135.98  132.10  134.21  1,500,520,000
> 1607 2013-04-29  134.44  147.49  134.00  144.54  1,491,160,000
> 1606 2013-04-30  144.00  146.93  134.05  139.00  1,597,780,000
> 1605 2013-05-01  139.00  139.89  107.72  116.99  1,542,820,000
> 1604 2013-05-02  116.38  125.60   92.28  105.21  1,292,190,000
```

INTERFACE: CRYPTOMANAGER

```
from crypto_ml.manager import CryptoManager as cm

manager = cm()

manager.send_tasks()

> bitcoin [[4952.28323284 5492.85474648 6033.42626011 6573.99777374 7
> 7655.14080101 8195.71231465 8736.28382828 9276.85534192 9817.426855
> bitconnect [[157.70136155 181.86603134 206.03070113 230.19537092 25
> 278.5247105  302.6893803  326.85405009 351.01871988 375.18338967]]
```

CODE

https://github.com/axsauze/crypto-ml

SLIDES

http://github.com/axsauze/industrial-machine-learning

#LETSDOTHIS

1. MACHINE LEARNING INTUITION

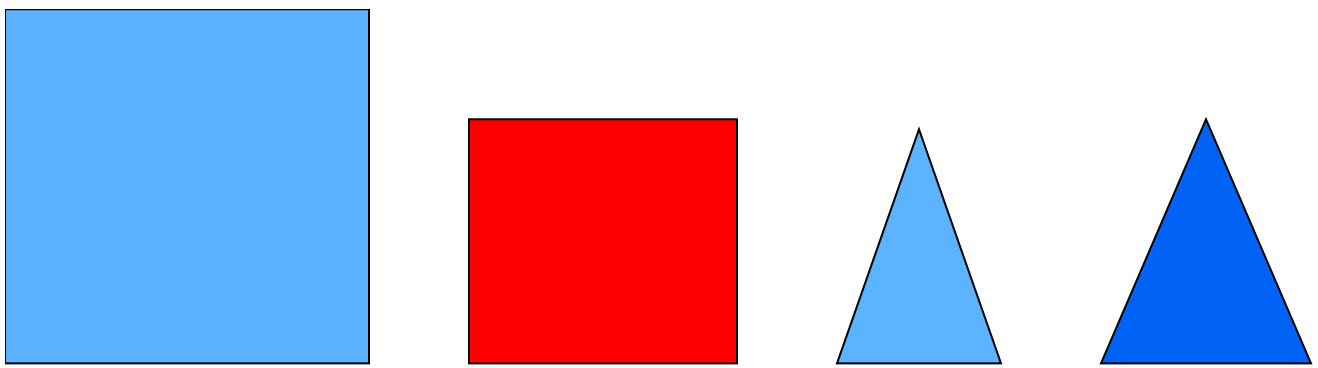
Crypto-ML Ltd. managed to appear in top tech magazines and raise VC money with their initial prototype.

```
import random

def predict_crypto(self, crypto_data):
    # I have no clue what I'm doing
    return crypto_data * random.uniform(0, 1)
```

NOW THEY NEED TO FIGURE OUT WHAT ML IS

ML TUTORIALS EVERYWHERE

Given some input data, predict the correct output

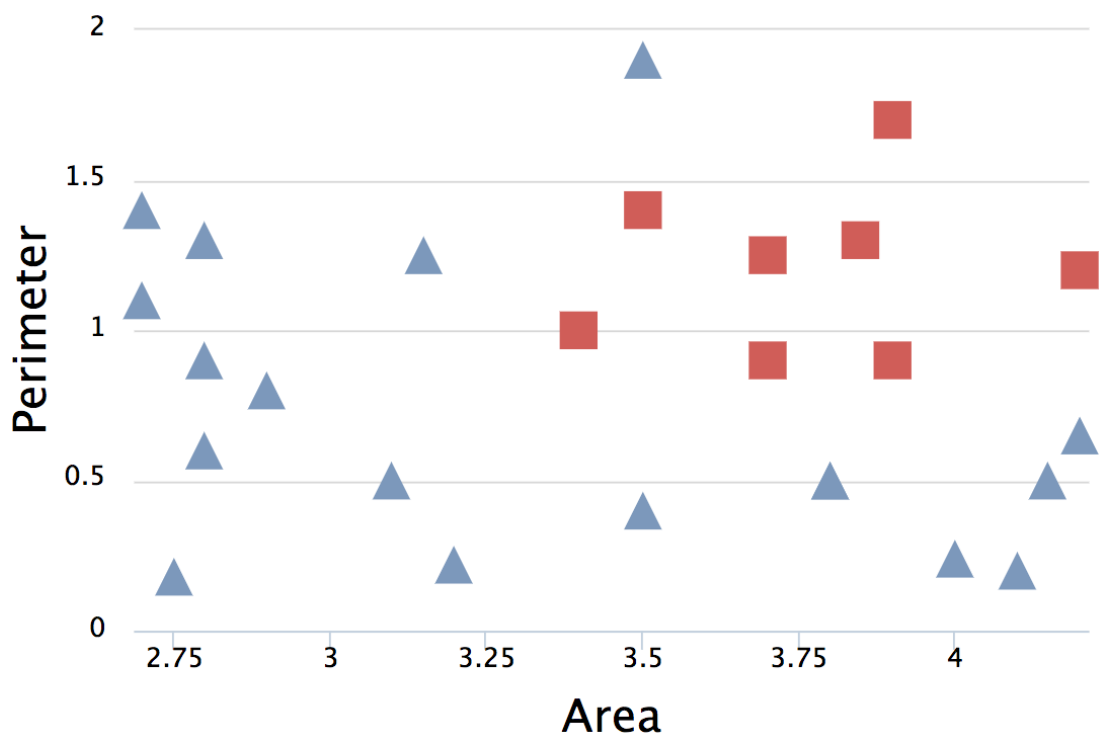


Let's try to predict whether a shape is a square or a triangle

HOW DO WE DO THIS?

LET'S VISUALISE IT

- Imagine a 2-d plot
- The x-axis is the area of the input shape
- The y-axis is the perimeter of the input shape

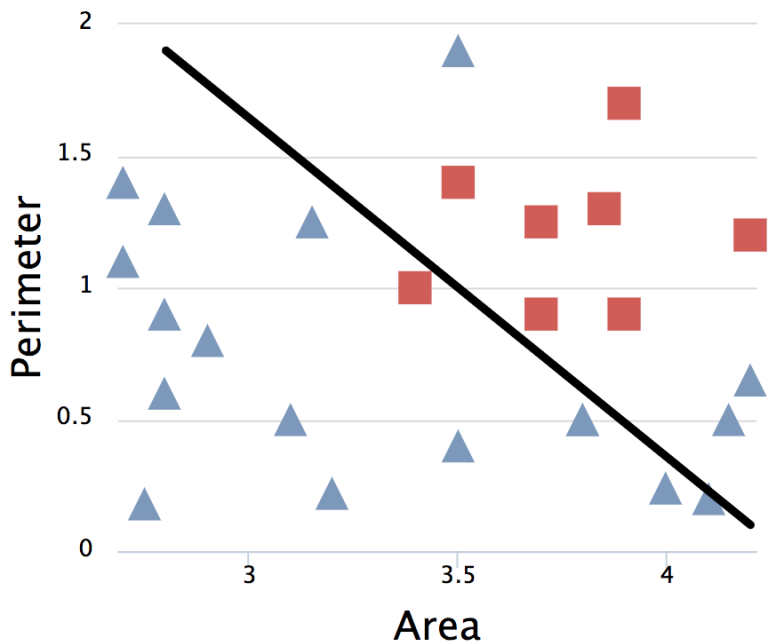


ALL ABOUT THE FUNCTION

$\mathbf{f}(\bar{\mathbf{x}}) = \mathbf{m}\bar{\mathbf{x}} + \mathbf{b}$, where:

$\bar{\mathbf{x}}$ is input (area & perimeter)

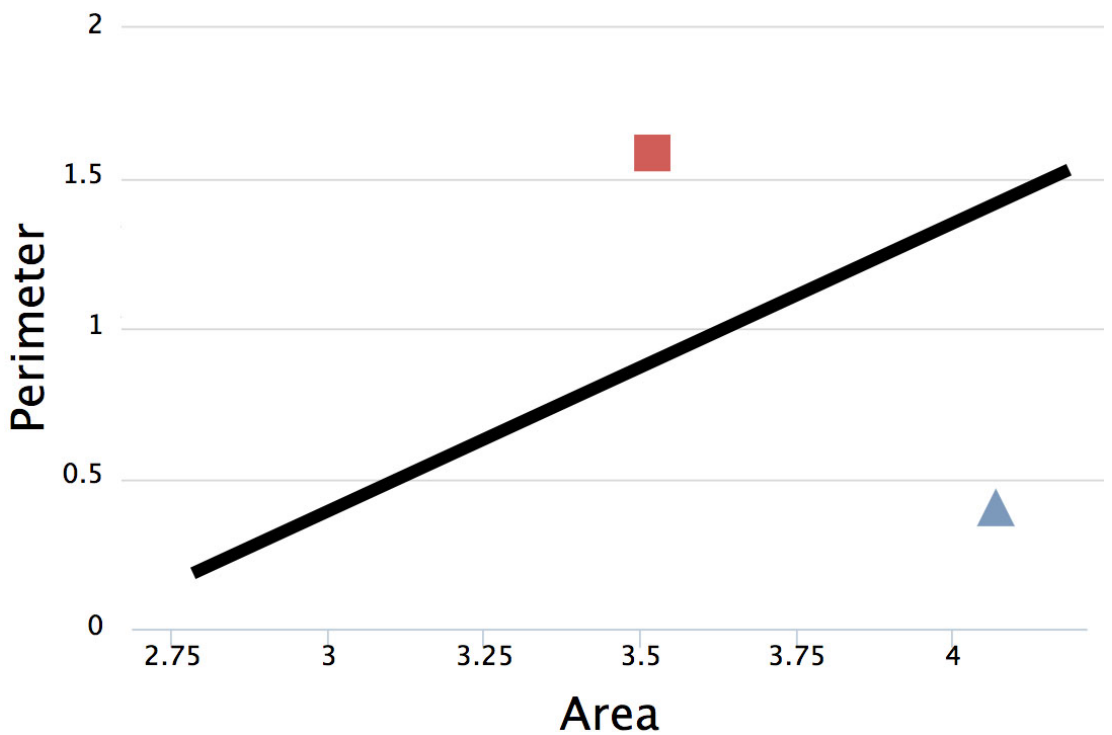
m and **b** are weights/bias



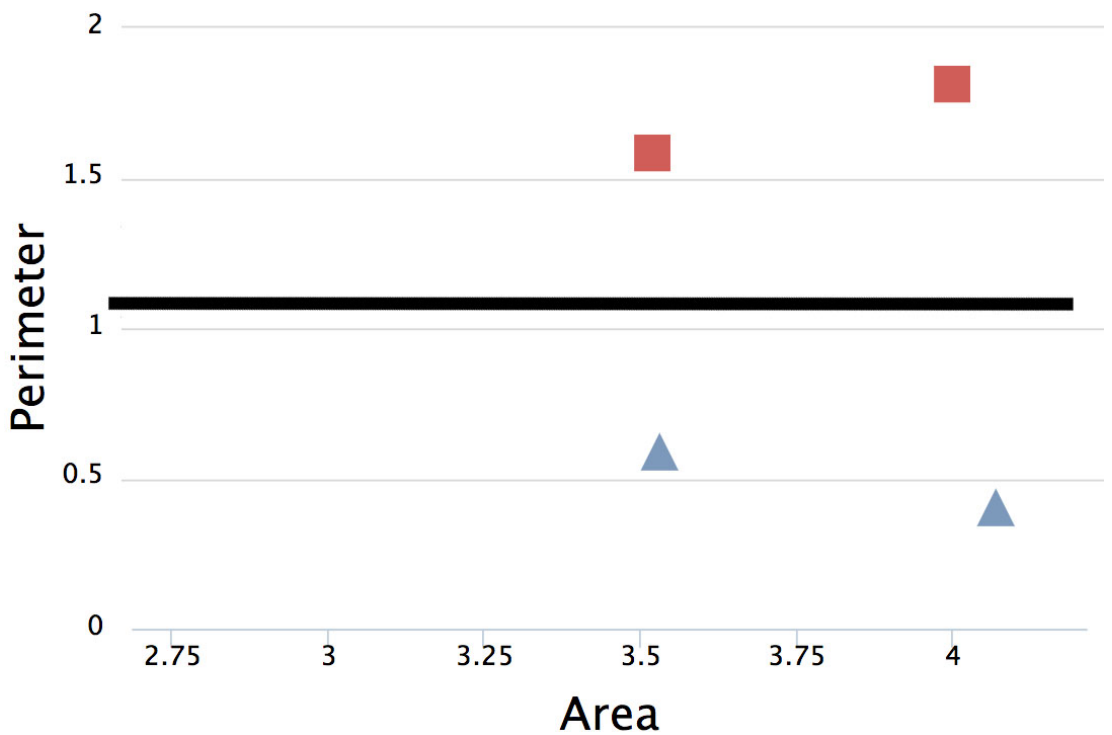
The result $\mathbf{f}(\bar{\mathbf{x}})$ states whether it's a triangle or square

(i.e. if it's larger than 0.5 it's triangle otherwise square)

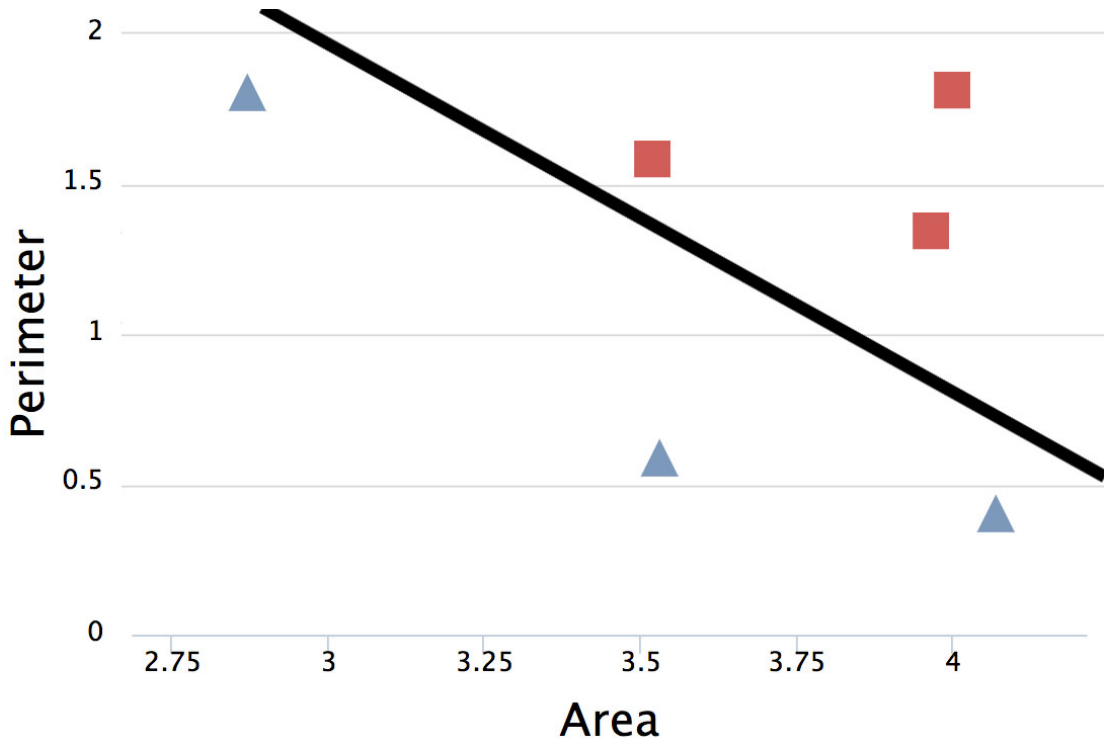
WE LET THE MACHINE DO THE LEARNING



WE LET THE MACHINE DO THE LEARNING



WE LET THE MACHINE DO THE LEARNING

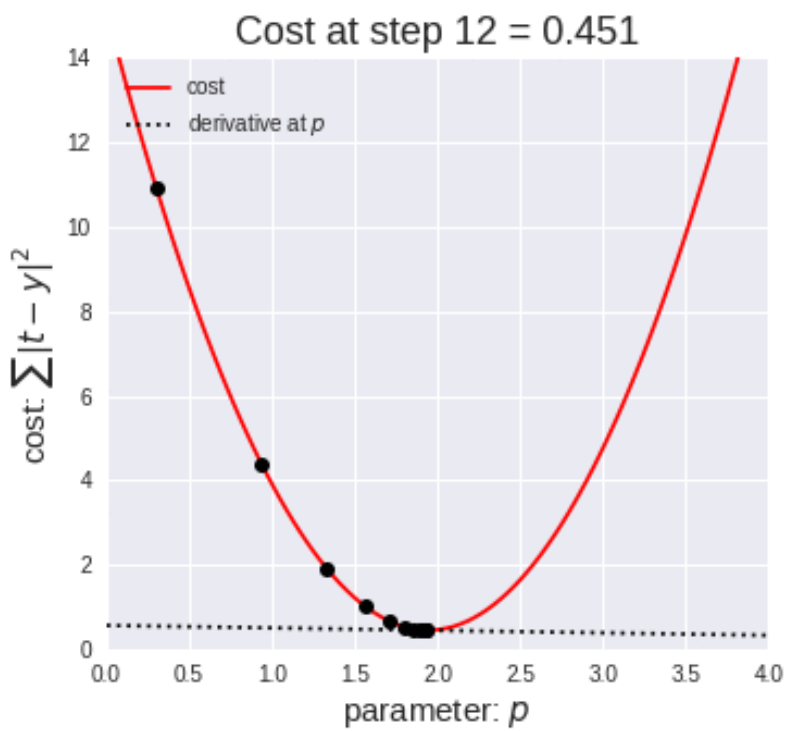


MINIMISING LOSS FUNCTION

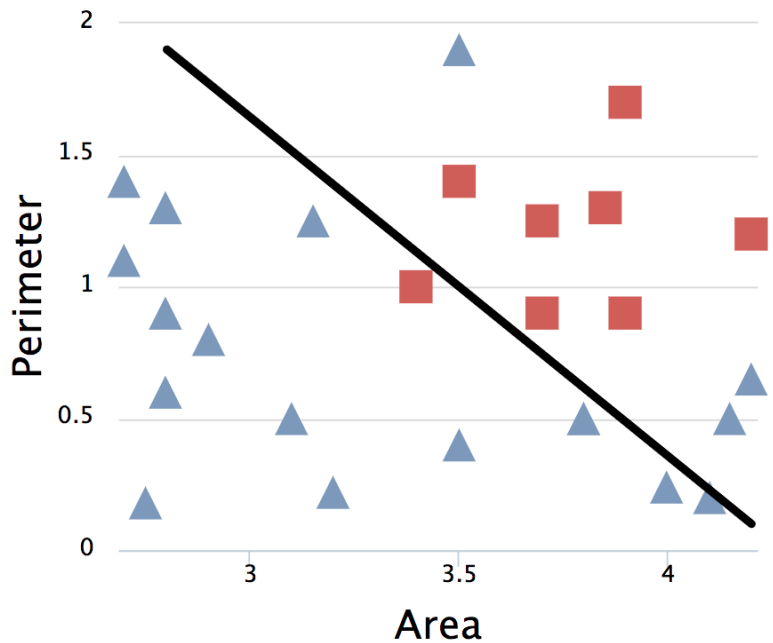
We optimise the model by **minimising its loss**.

Keep adjusting the weights...

...until loss is not getting any smaller.



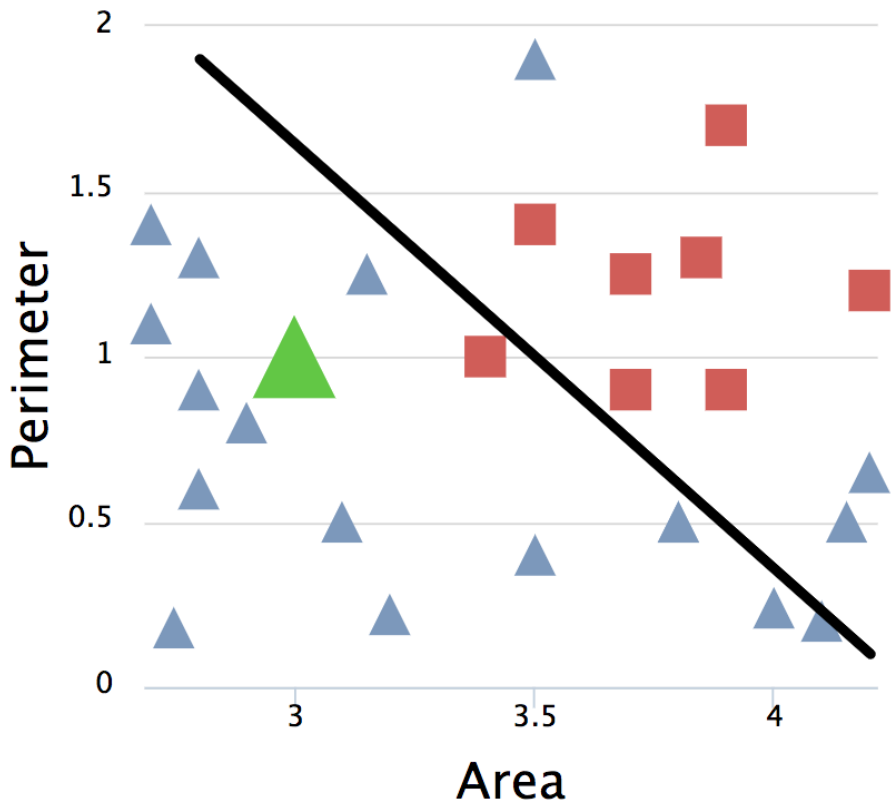
FINDING THE WEIGHTS!



When it finishes, we find optimised weights and biases

i.e. $\mathbf{f}(\bar{\mathbf{x}})$ = triangle if $(\mathbf{0.3}\bar{\mathbf{x}} + \mathbf{10})$ > 0.5 else square

NOW PREDICT NEW DATA



Once we have our function, we can predict NEW data!

WE'RE ML EXPERTS!

Please collect your certificates after the talk

These are valid in:

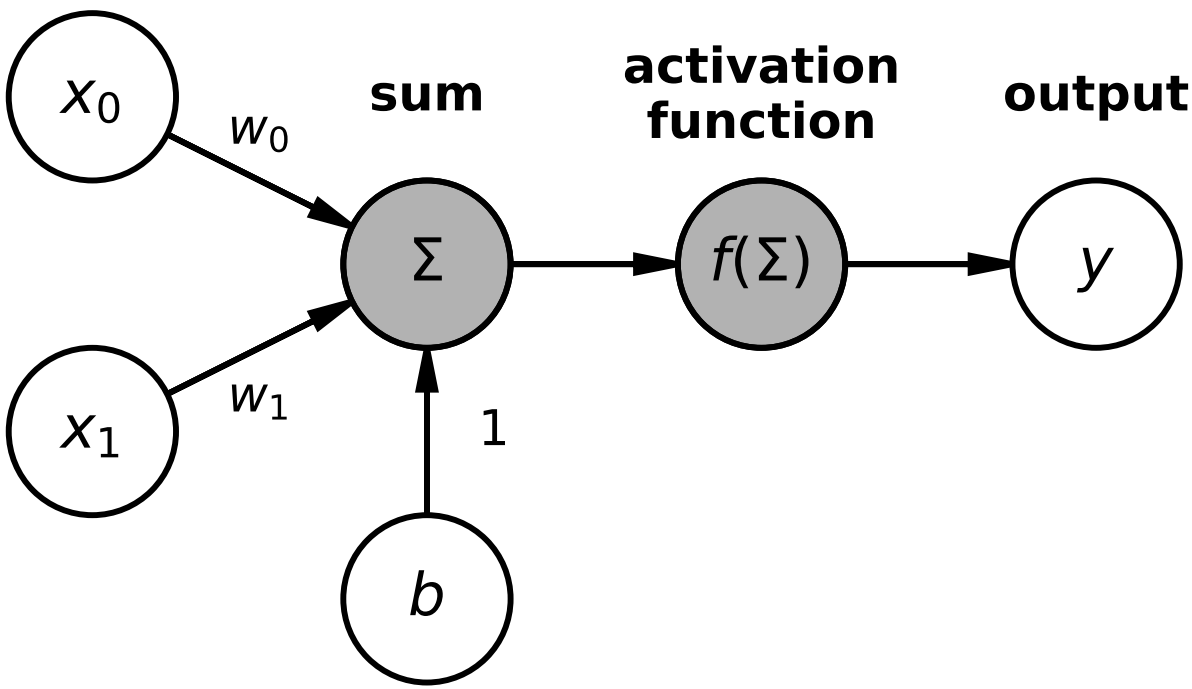
- Your Linkedin profile
- Non-tech Meetups and Parties
- Any time you reply to a tweet

IN ALL SERIOUSNESS...

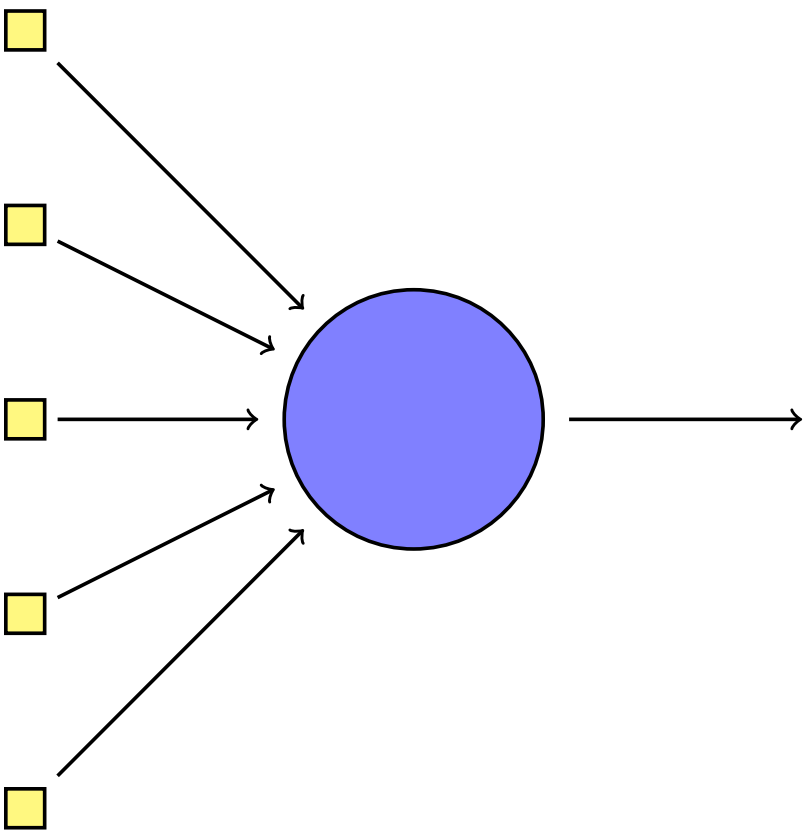
`f(x) = mx + b`

is the single perceptron function

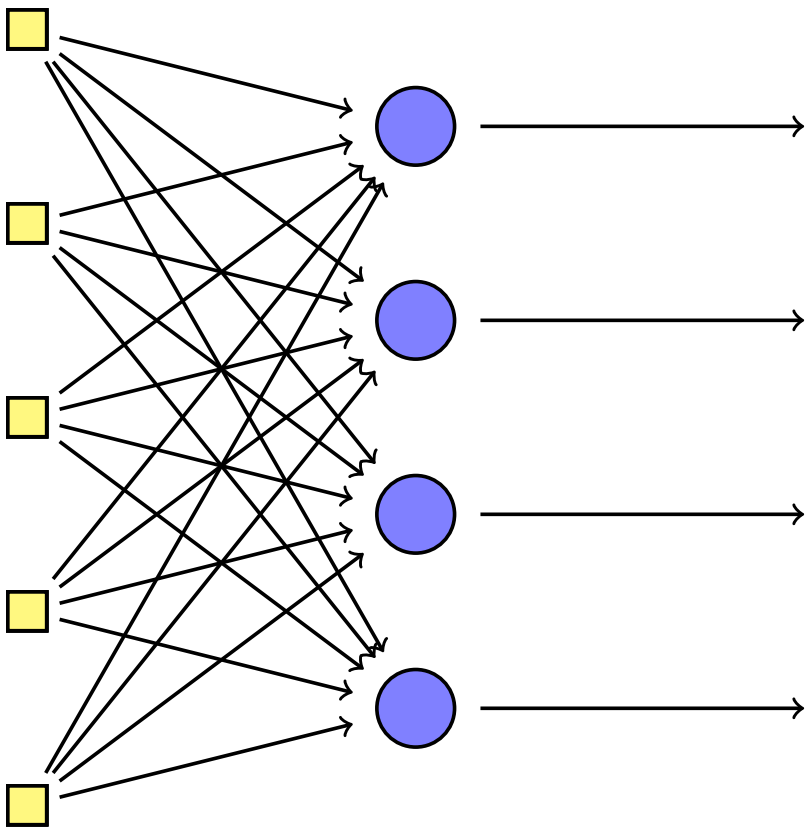
in a neural network



INSTEAD OF JUST ONE NEURON

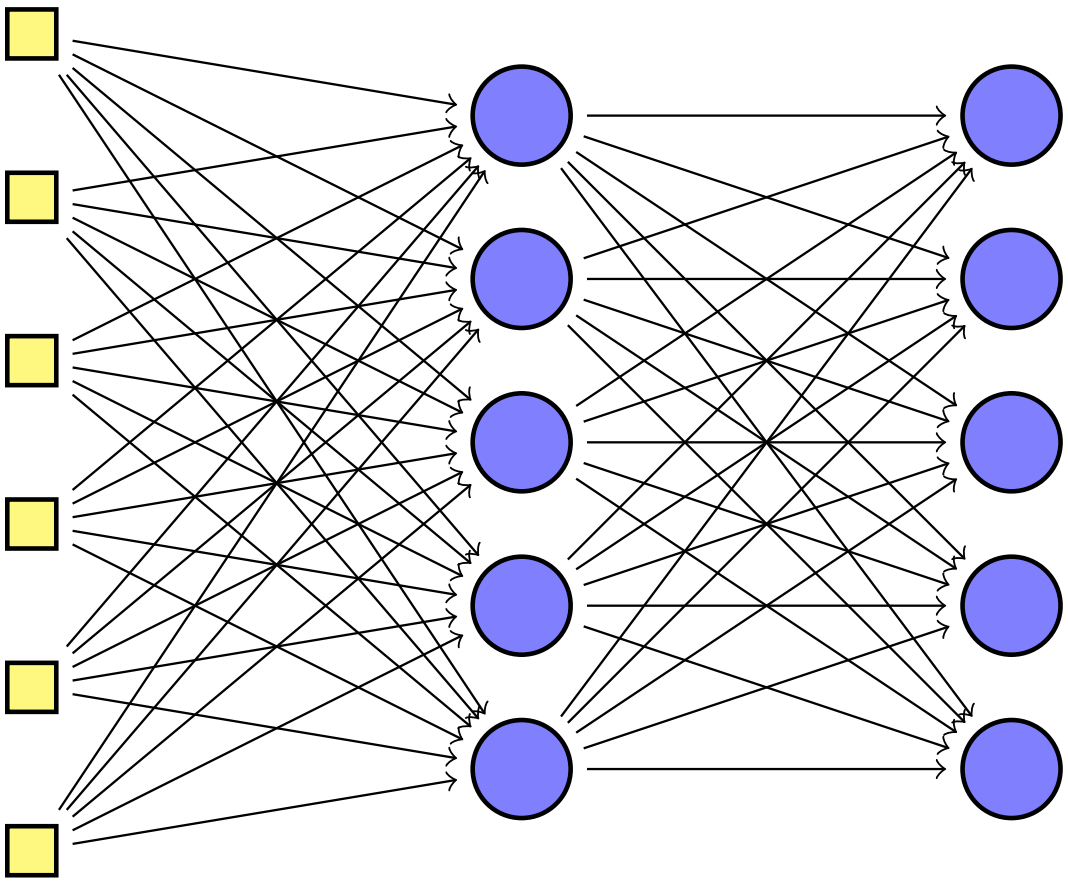


WE JUST HAVE MANY



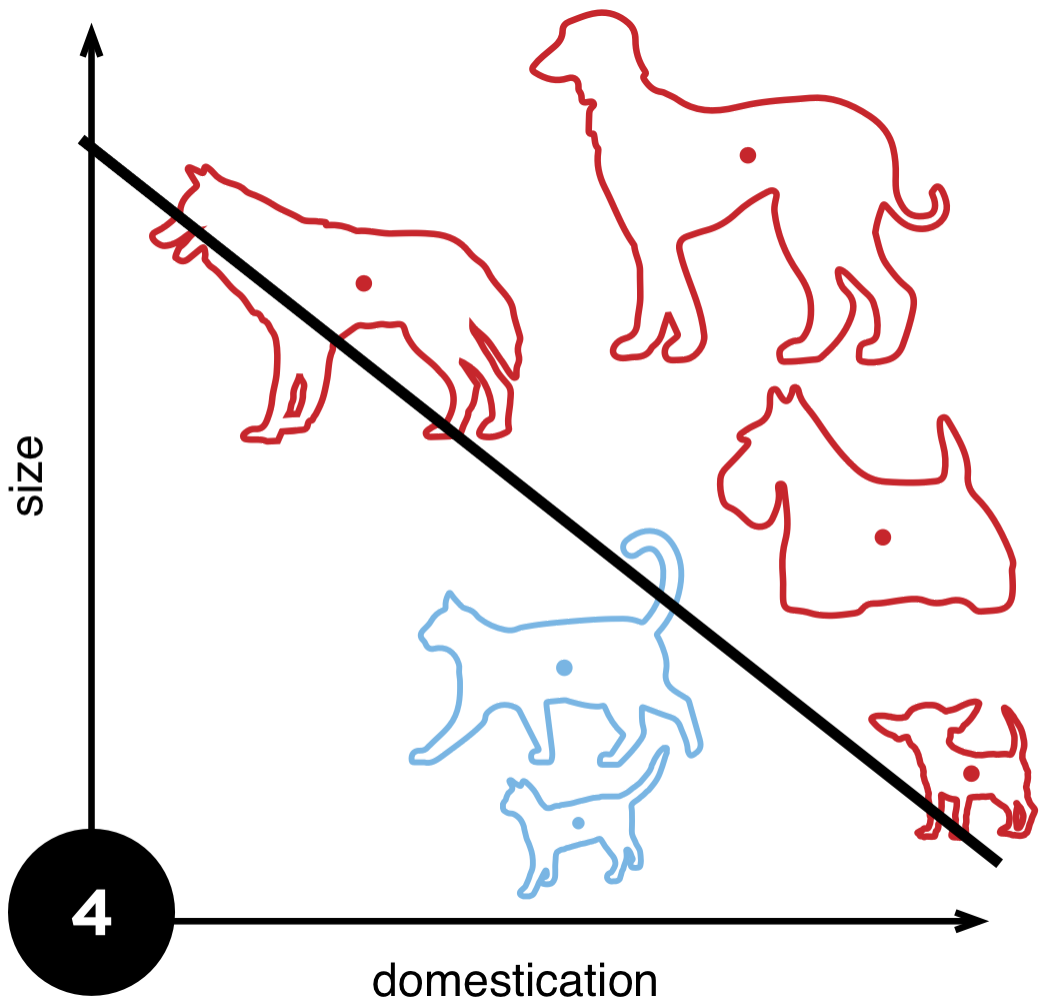
This gives the function more flexibility

WITH A FEW LAYERS

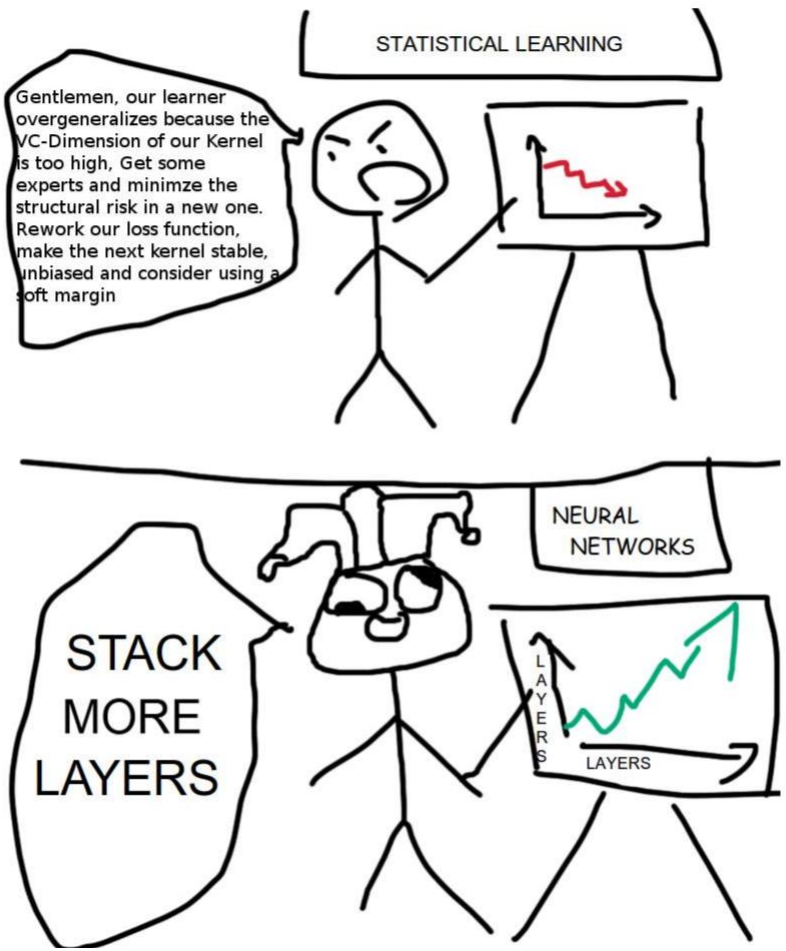


This gives more flexibility for learning

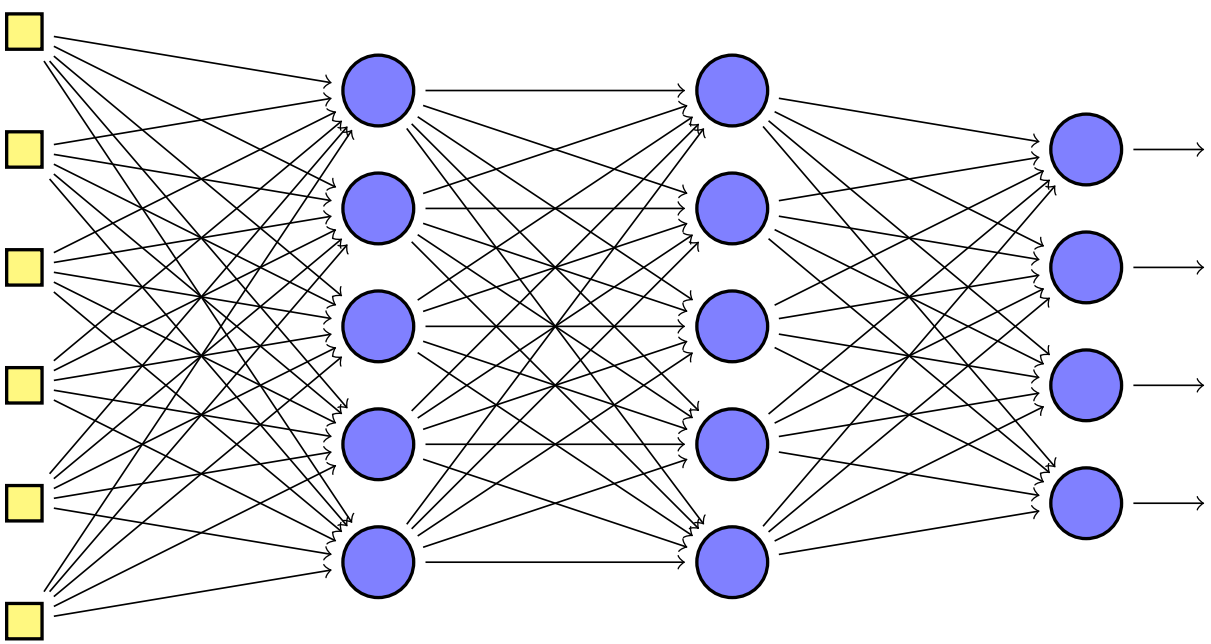
But what about with more complex cases?

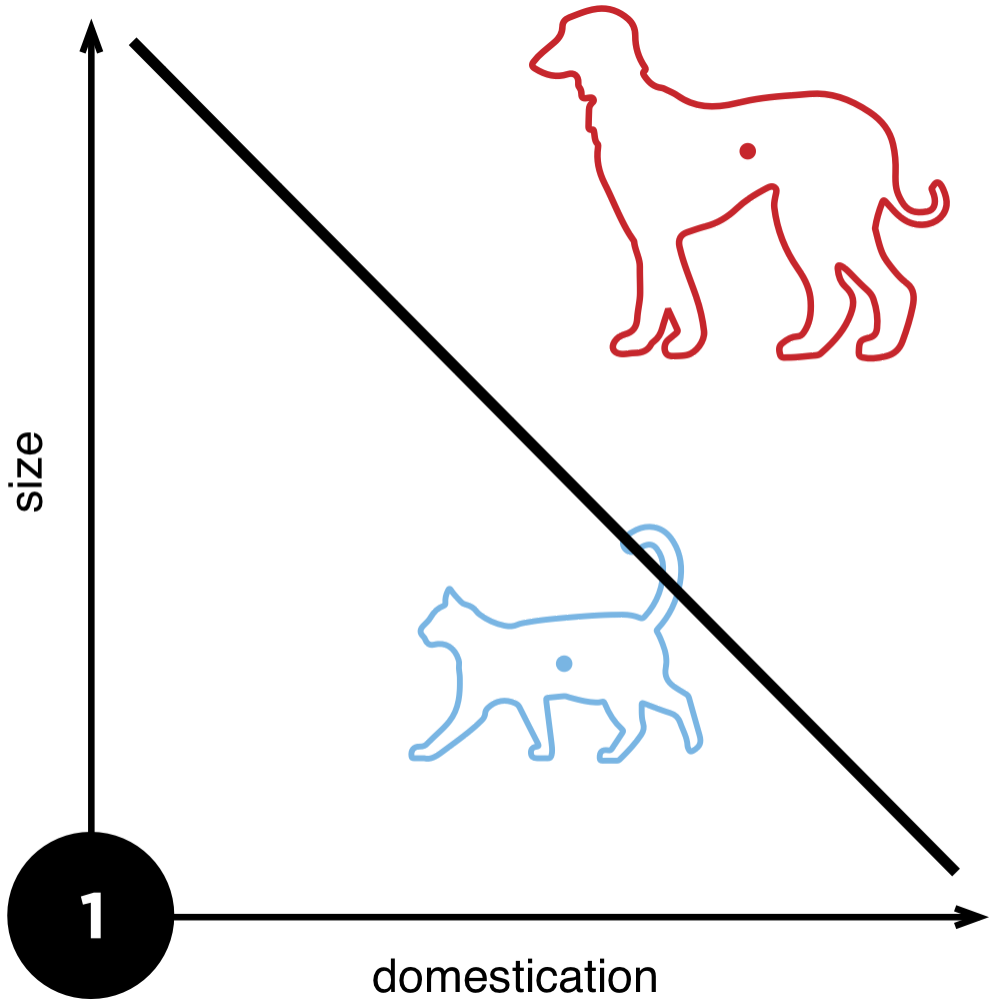


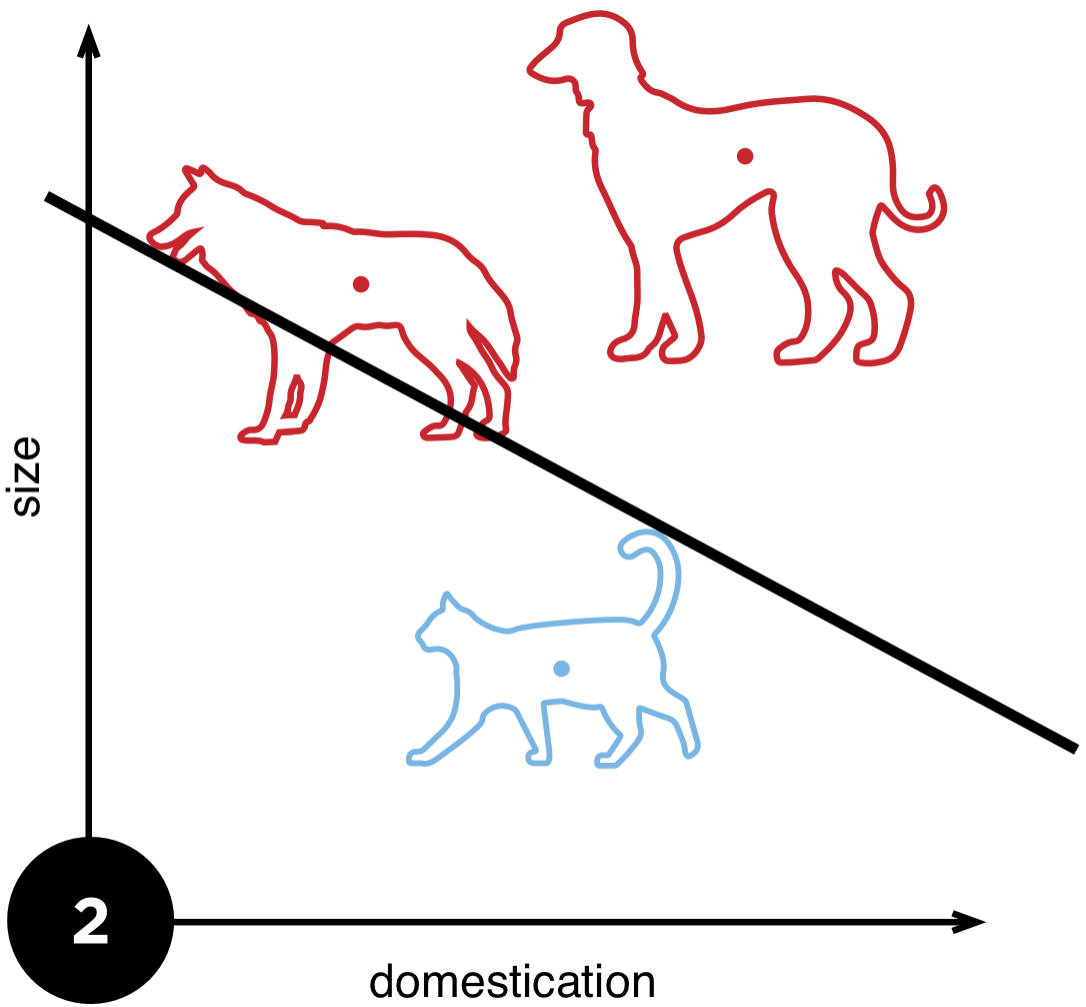
DEEP NEURAL NETWORKS

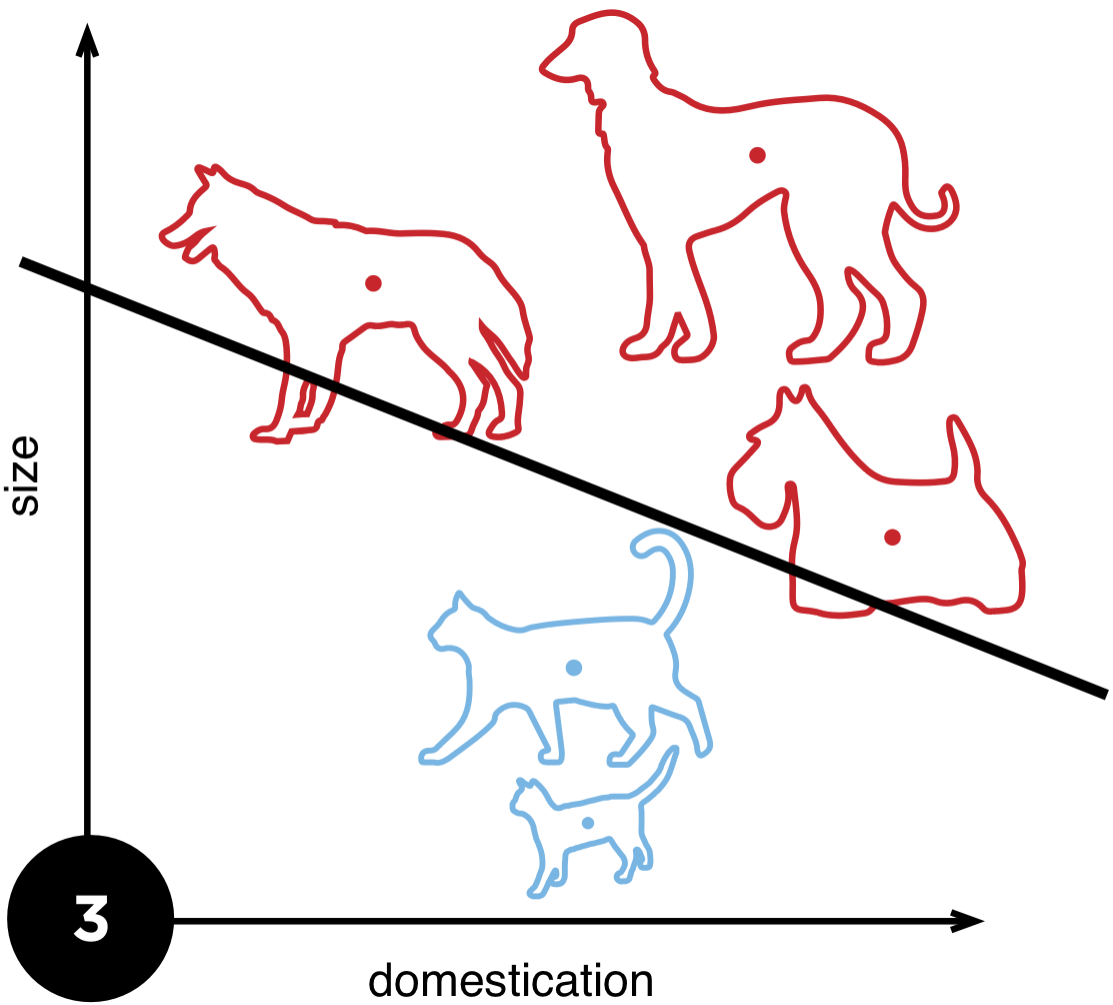


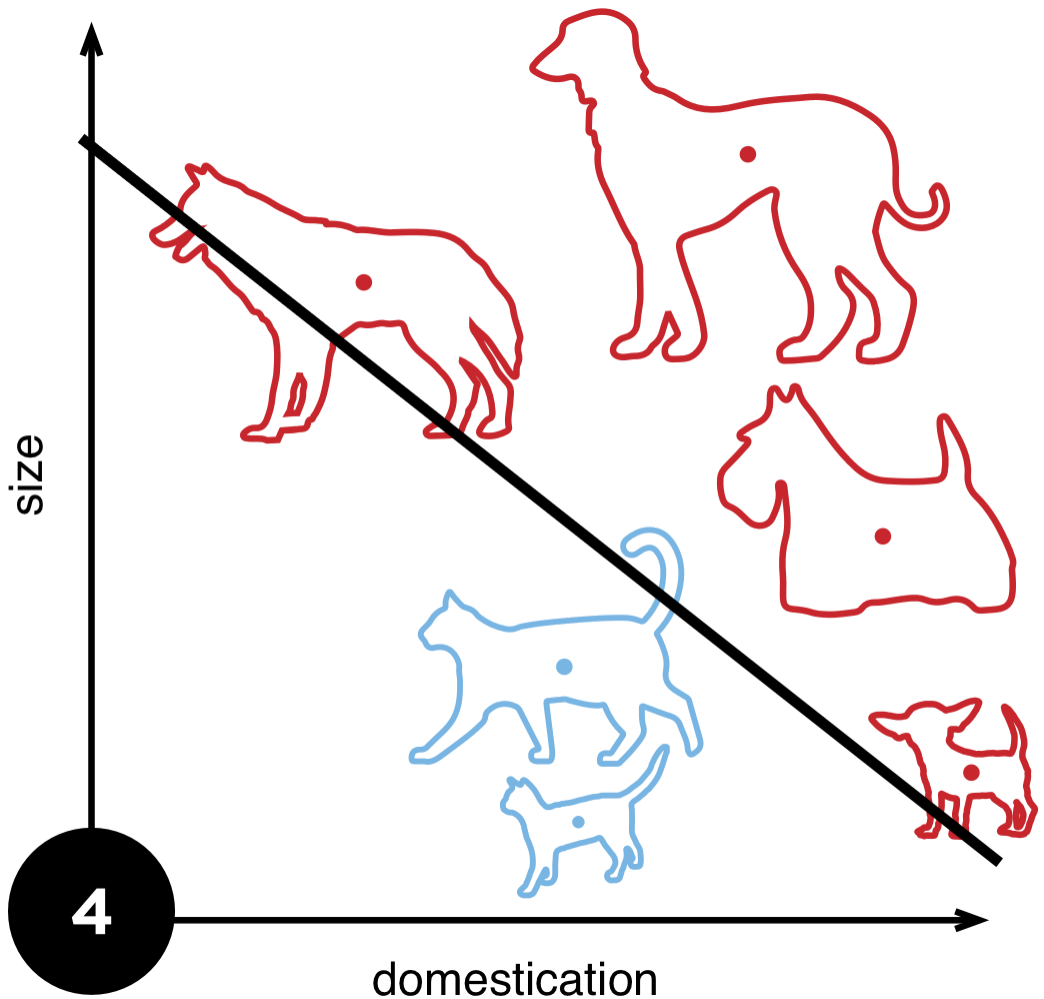
DEEP NETWORKS — MANY HIDDEN LAYERS




size
1
domestication


size
2
domestication


size
3
domestication

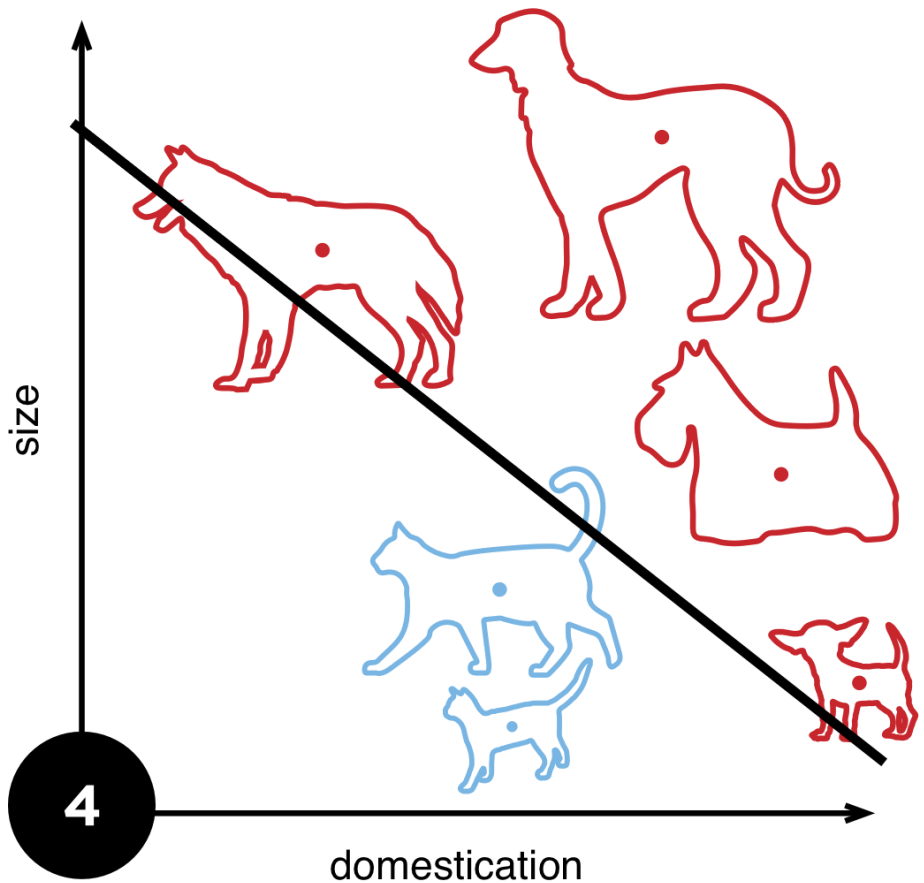

size
4
domestication

2. MACHINE LEARNING IN PRACTICE

The real world has more varied and complex cases

You need a pragmatic approach

- Extending our feature space
- Increasing number of inputs
- Regularisation techniques (dropout, batch normalisation)
- Normalising datasets



The Crypto-ML devs asked themselves...

```
from crypto_ml.data_loader import CryptoLoader

btc = CryptoLoader().get_df("bitcoin")

btc.head()

>            Date    Open    High     Low   Close      Market Cap
> 1608 2013-04-28  135.30  135.98  132.10  134.21  1,500,520,000
> 1607 2013-04-29  134.44  147.49  134.00  144.54  1,491,160,000
> 1606 2013-04-30  144.00  146.93  134.05  139.00  1,597,780,000
> 1605 2013-05-01  139.00  139.89  107.72  116.99  1,542,820,000
> 1604 2013-05-02  116.38  125.60   92.28  105.21  1,292,190,000
```

We are now experts in ML, however...

...can this be used for our cryptocurrency price data?

NOT YET.

Processing sequential data requires a different approach.

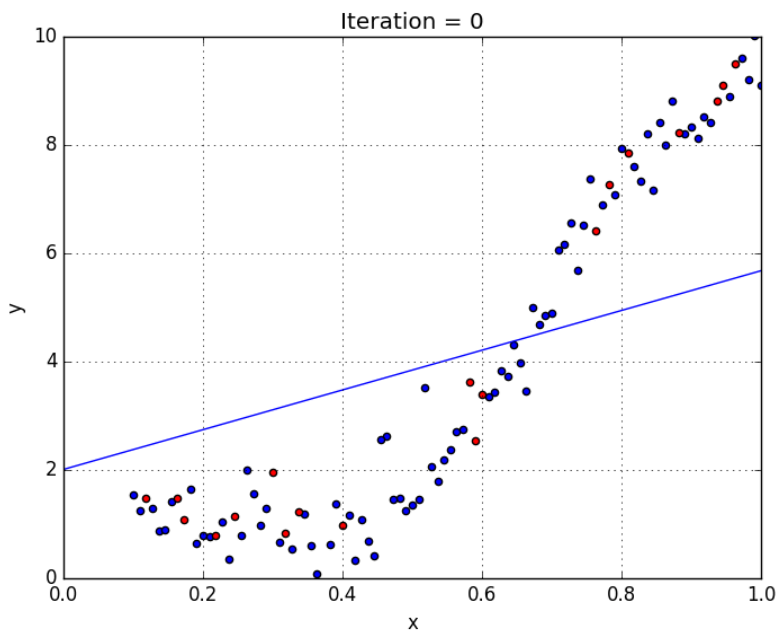
Instead of trying to predict two classes...

...we want to predict future steps

HOW DO WE DO THIS?

SEQUENTIAL MODELS

Sequential models often are used to predict future data.



Still uses the same approach

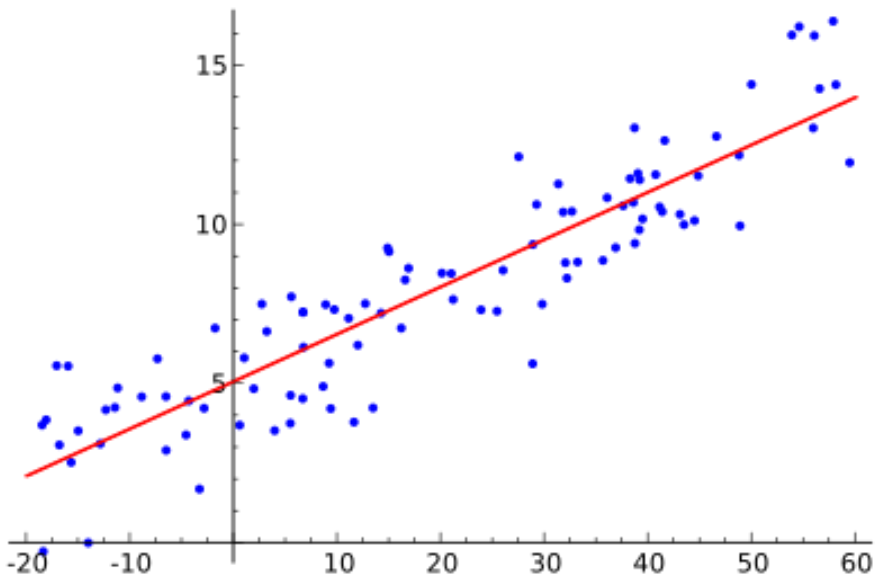
`f(x) = mx + b`

To find the weights and biases

But can be used on time-sequence data - ie. prices, words, characters, etc.

THE HELLO_WORLD OF ML

Predicting prices by fitting a line on set of time-series points



LINEAR REGRESSION

LINEAR REGRESSION

```
from sklearn import linear_model

def predict(prices, times, predict=10):
    model = linear_model.LinearRegression()
    model.fit(times, prices)
    predict_times = get_prediction_timeline(times, predict)
    return model.predict(predict_times)
```

LINEAR REGRESSION

```
from crypto_ml.data_loader import CryptoLoader


cl = CryptoLoader()

df = cl.get_df("bitcoin")

times = df[["Date"]].values
prices = df[["Price"]].values

results = predict(prices, times, 5)
```

SUCCESS

BUT THE CRYPTO-ML TEAM WANTS

CUTTING EDGE TECH

DEEP RECURRENT NEURAL NETWORKS

DEEP RECURRENT NEURAL NETWORKS

They are pretty much deep neural networks.

But their use is to predict a future time-step...

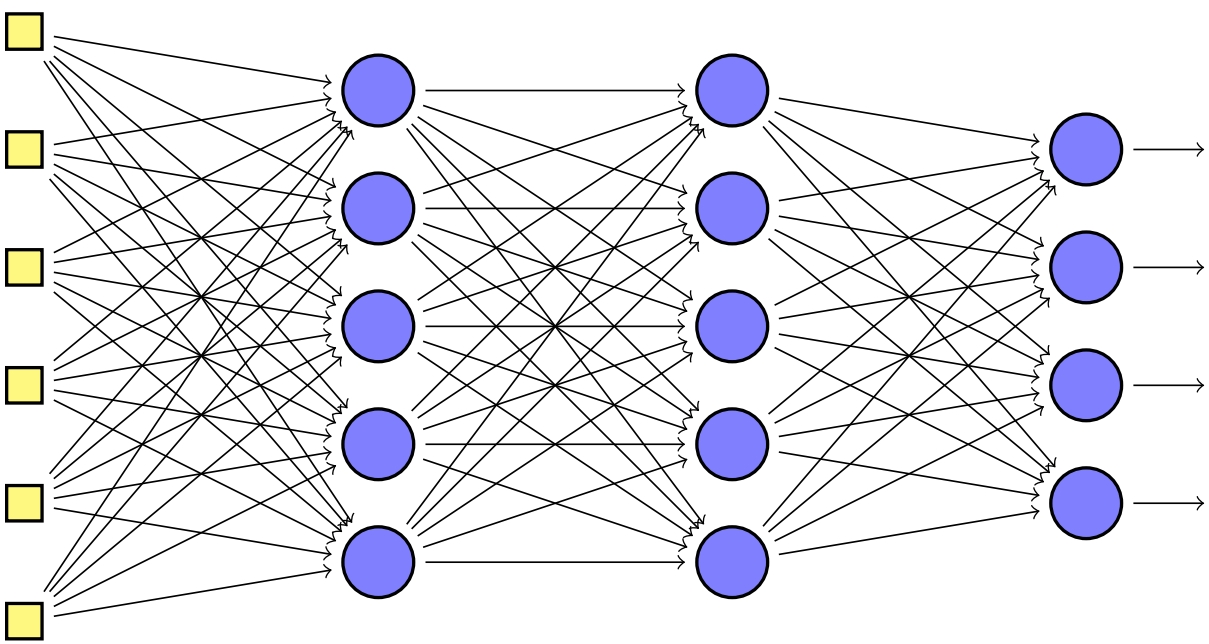
...instead of predicting a class.

DEEP RECURRENT NEURAL NETWORKS

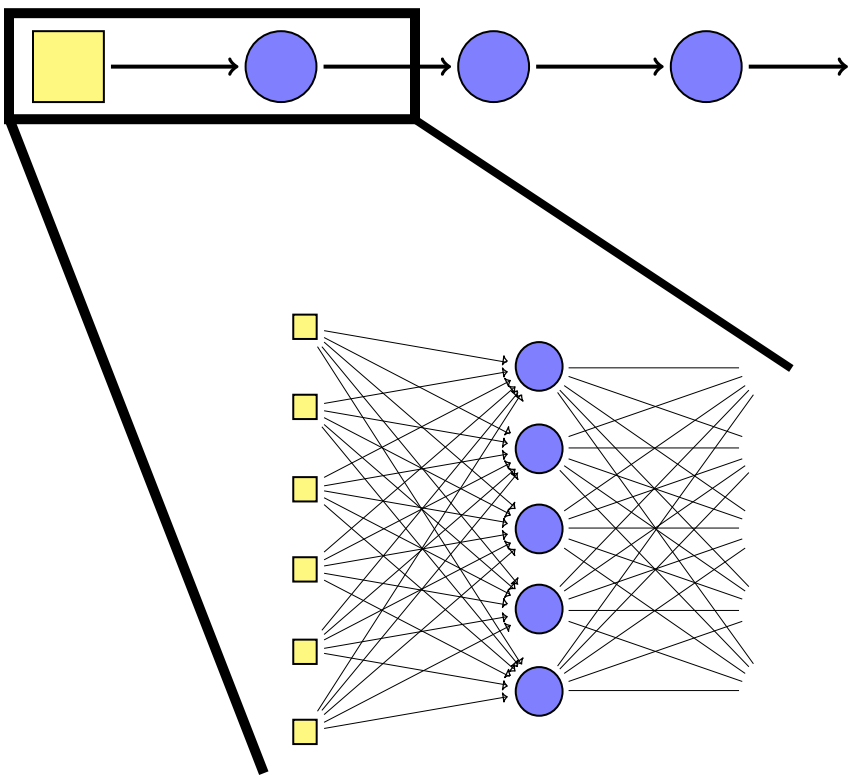
Because of the layers and neurons:

- They can hold more complex functions/models.
- They can take more complex features.
- They require more data

LET'S RECALL DEEP NEURAL NETS

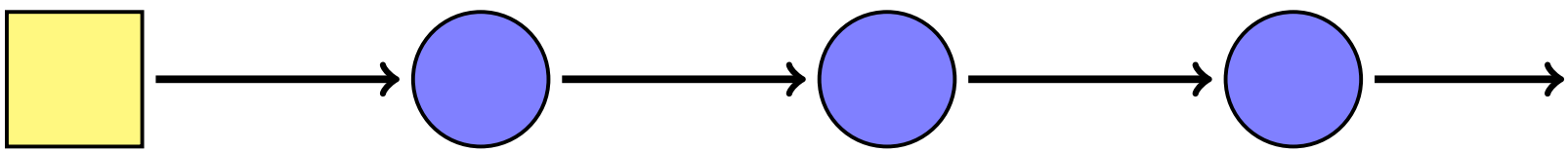


SIMPLIFIED VISUALISATION



One node represents a full layer of neurons.

SIMPLIFIED VISUALISATION

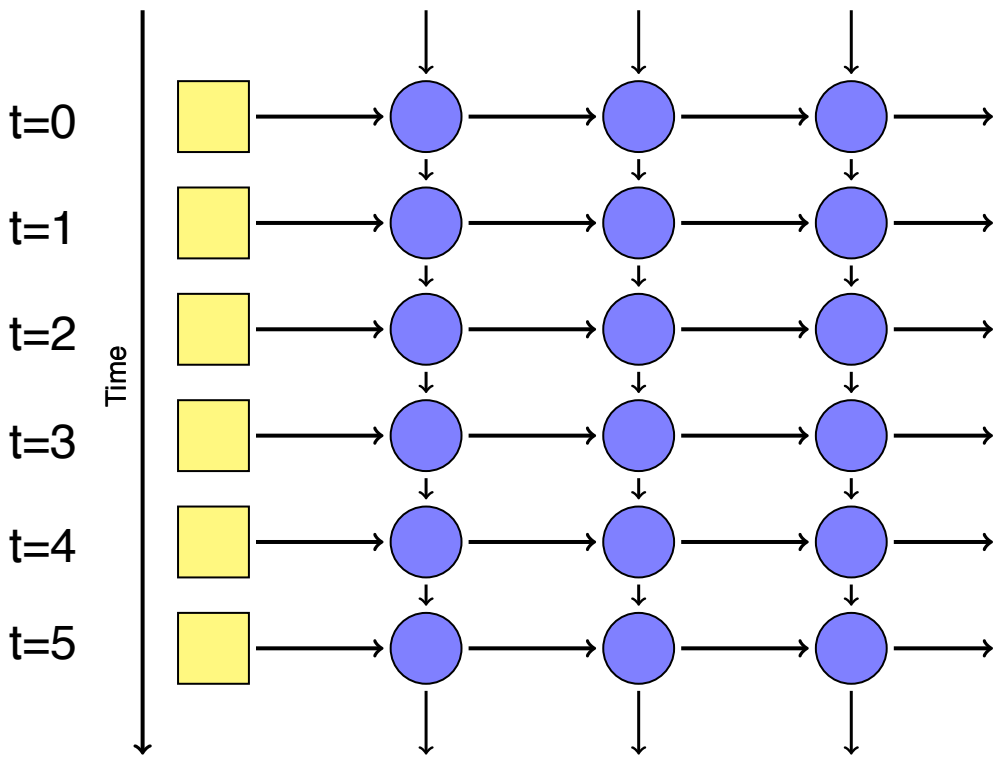


One node represents a full layer of neurons.

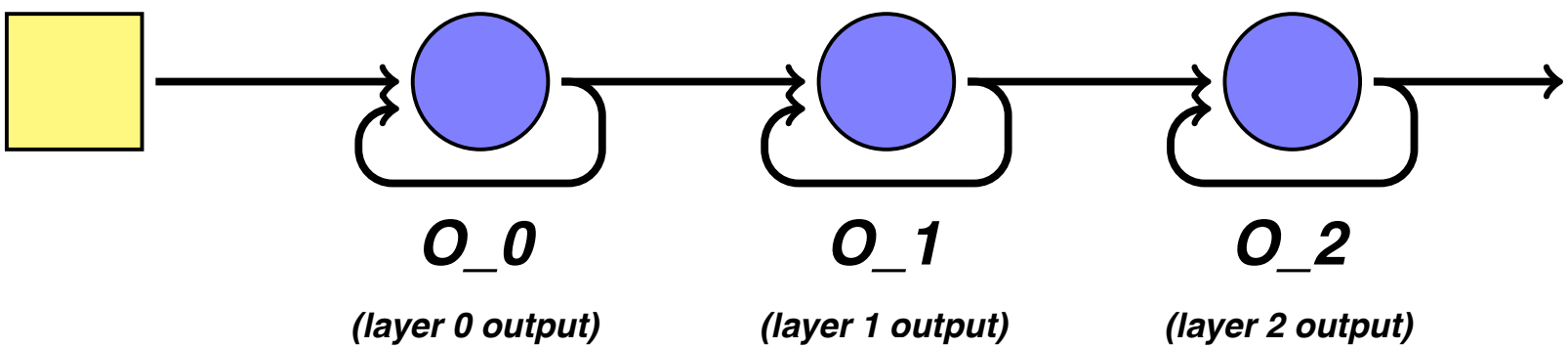
UNROLLED RECURRENT NETWORK

Previous predictions help make the *next* prediction.

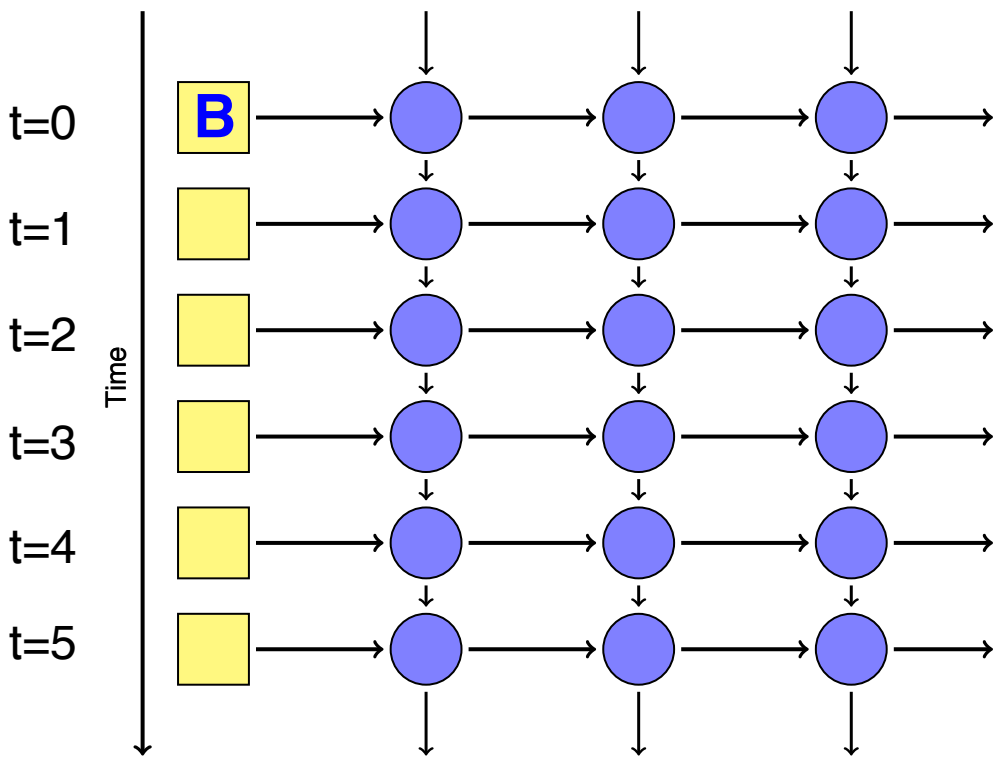
Each prediction is a **time step**.

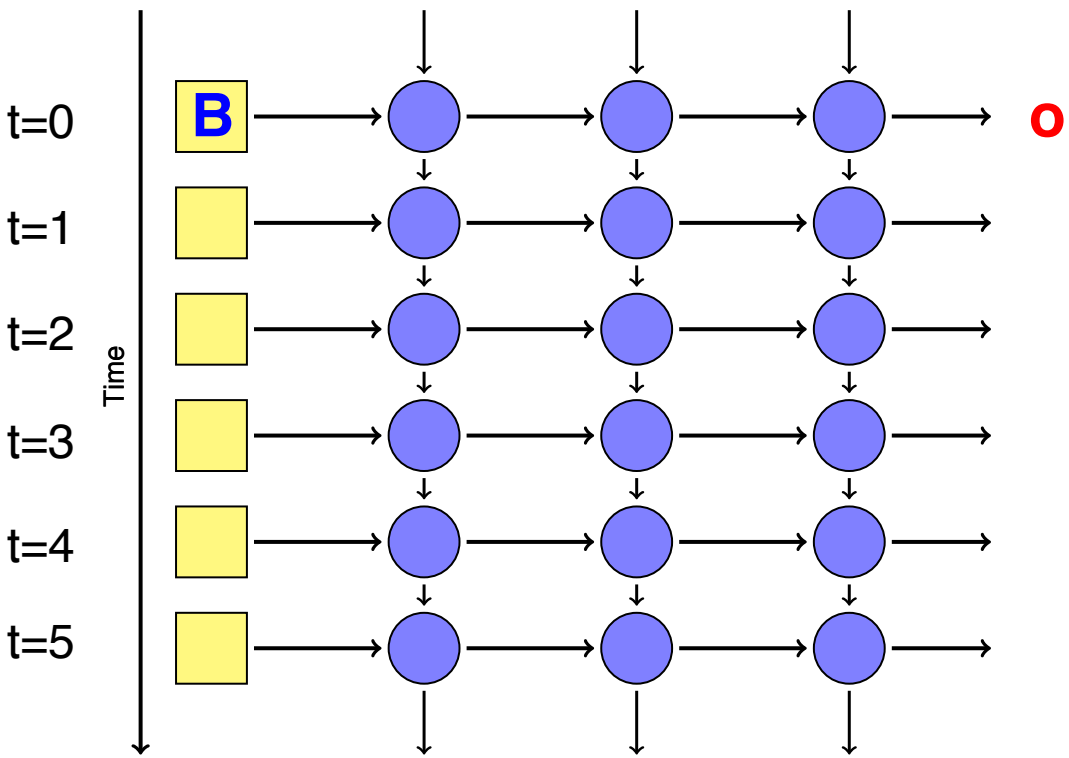


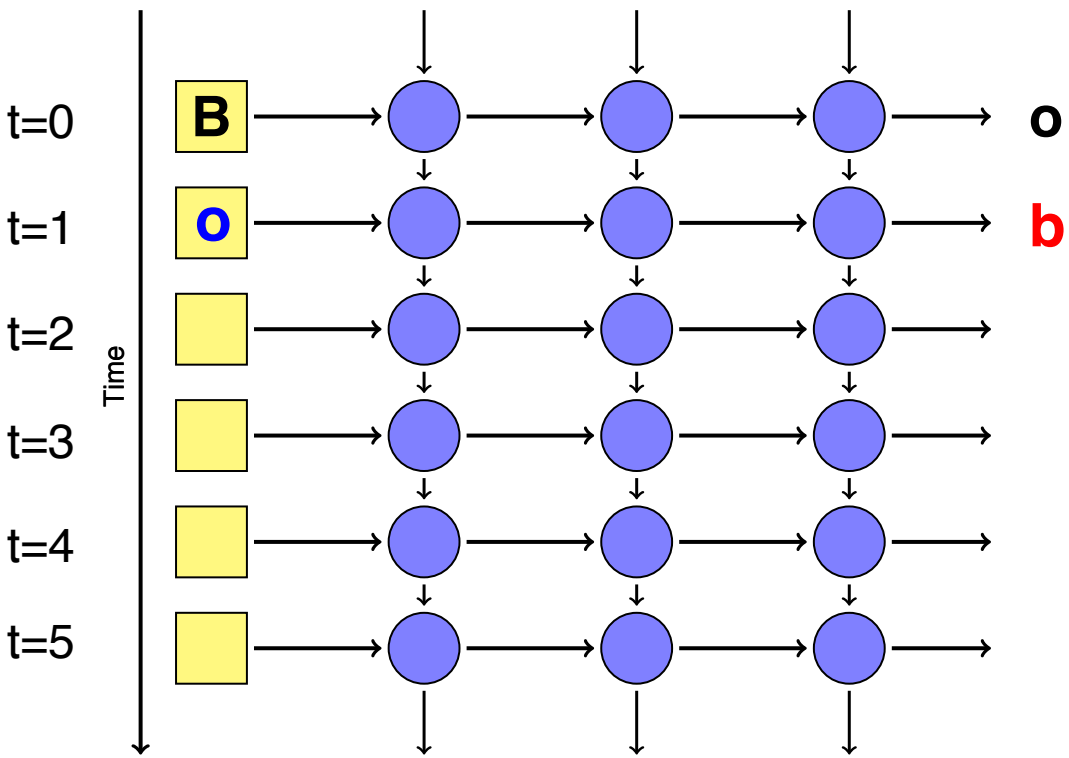
RECURRENT NETWORKS

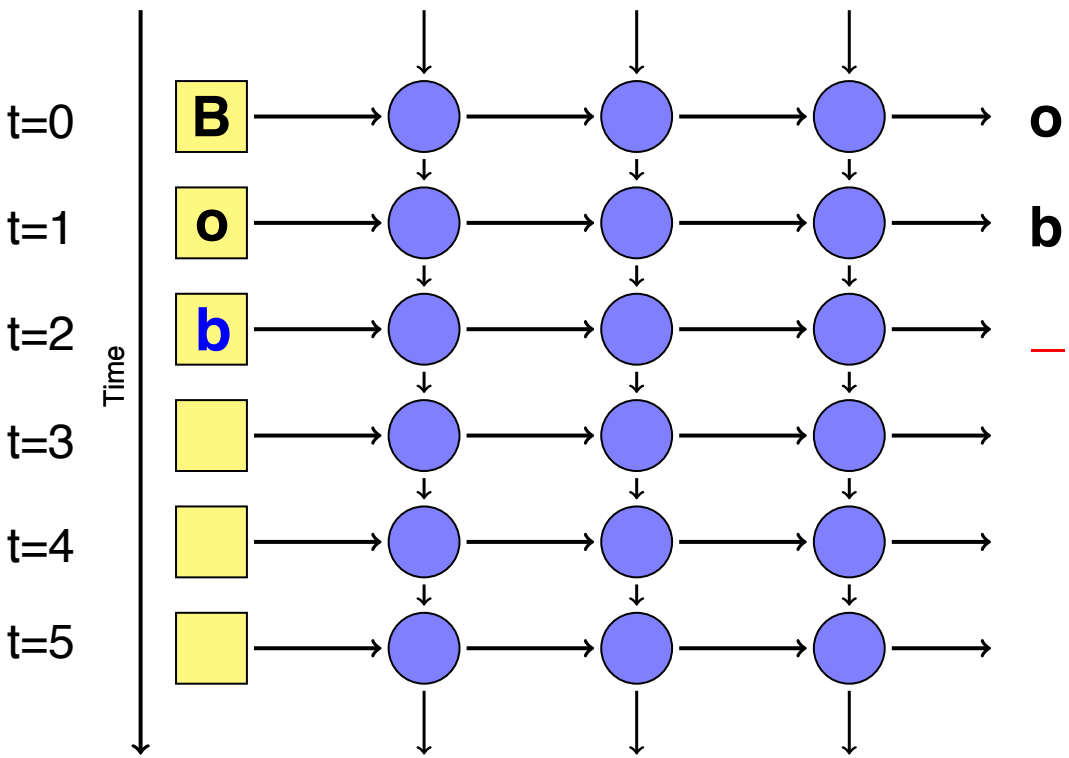


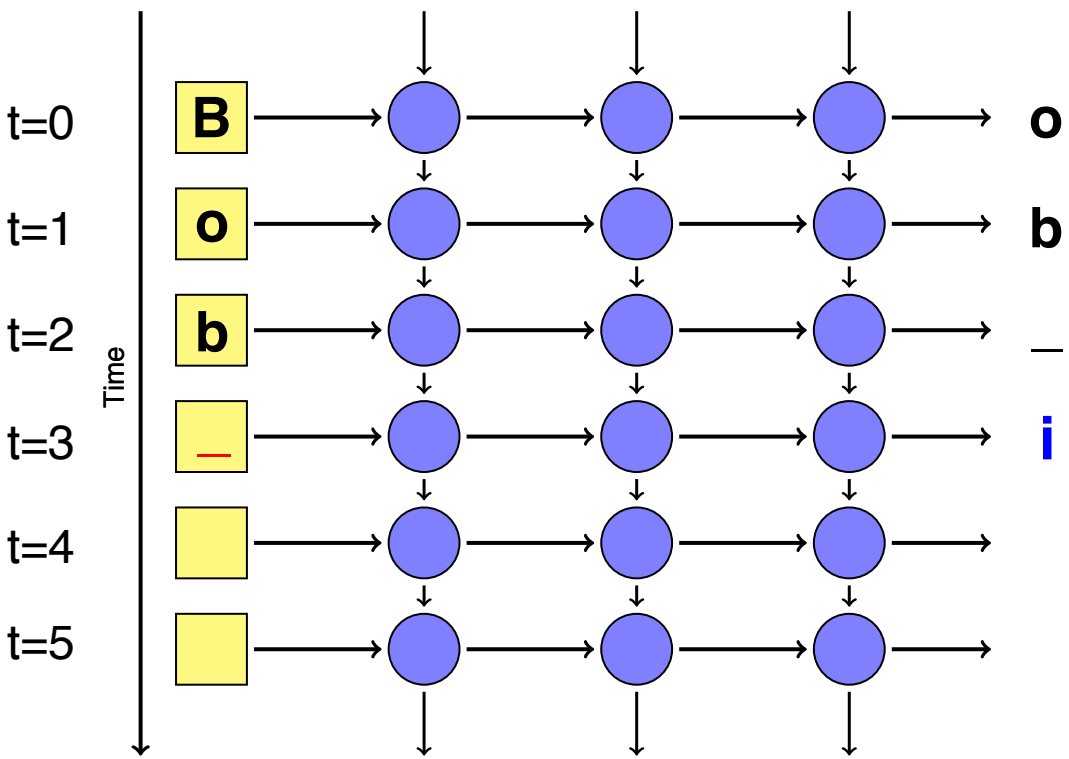
Hidden layer's input includes the output of itself during the last run of the network.

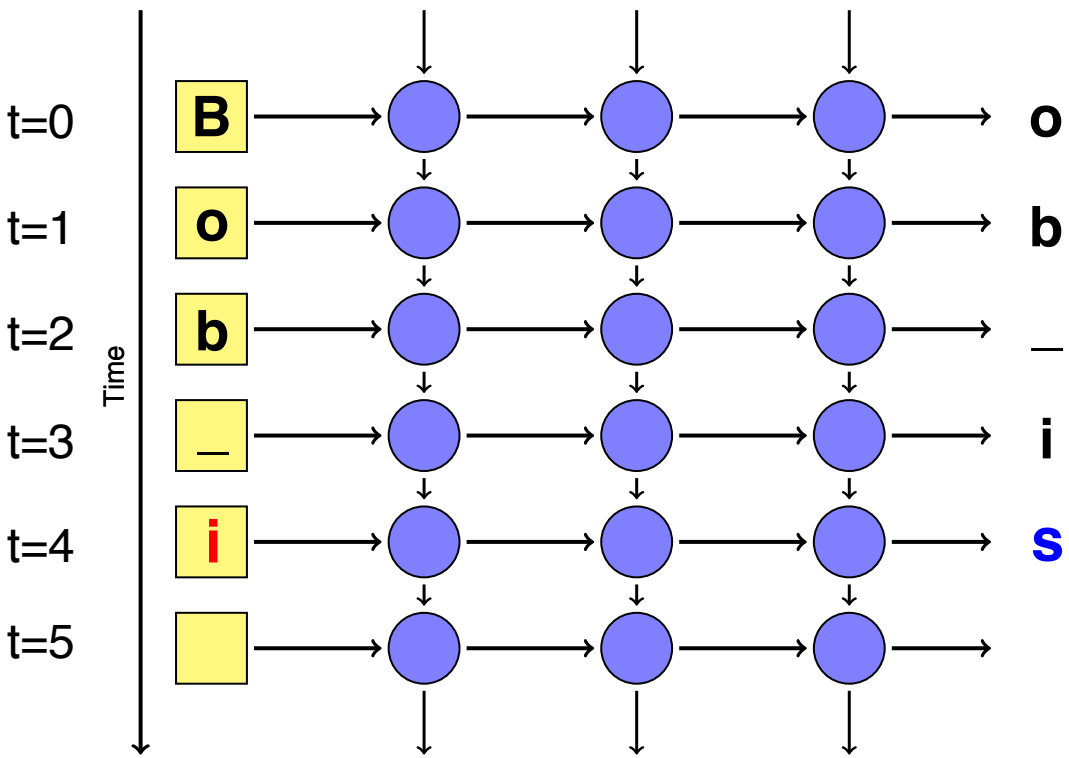

t=0
t=1
t=2
t=3
t=4
t=5
Time
B

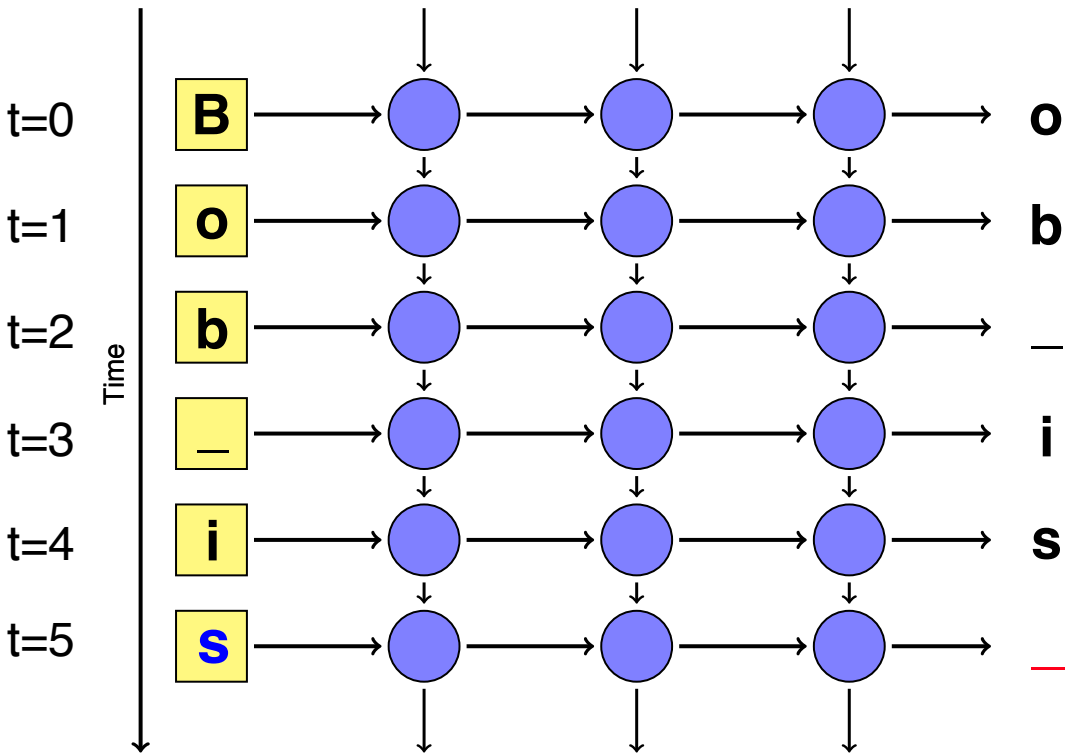

t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o


t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o
o
b


t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o
b
o
b
_

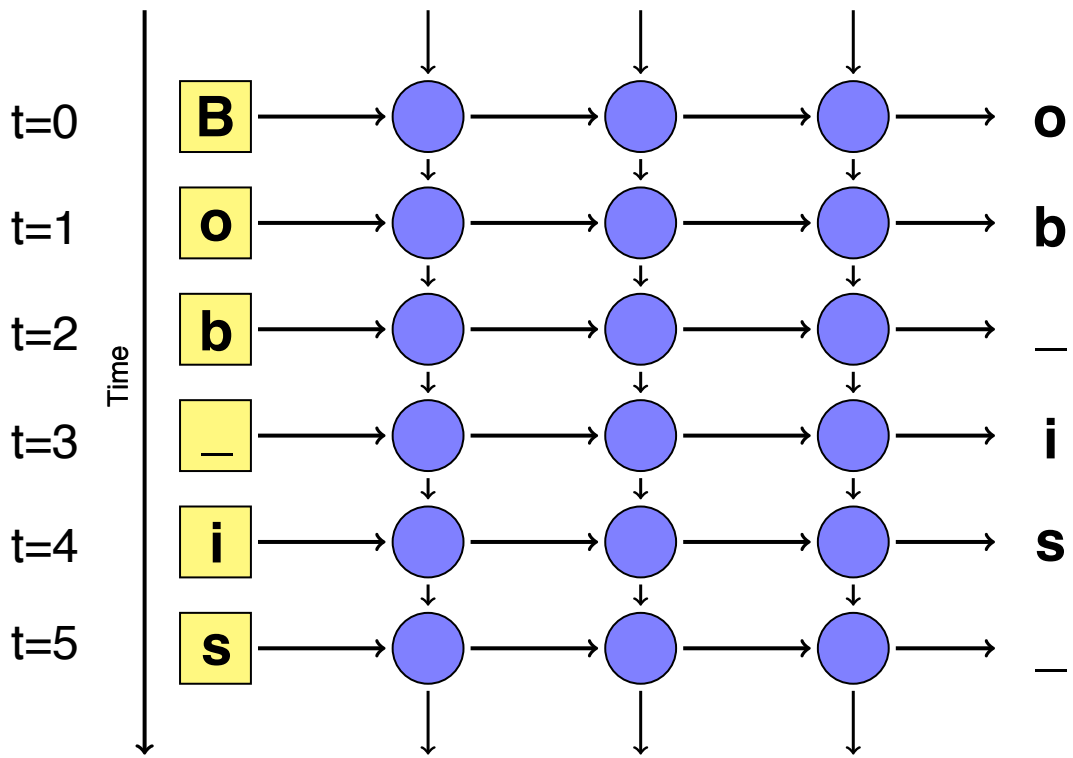

t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o
b
_
o
b
_
i


t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o
b
_
i
o
b
_
i
s


t=0
t=1
t=2
t=3
t=4
t=5
Time
B
o
b
_
i
s
o
b
_
i
s
_

LOSS/COST FUNCTION

Cost function is based on getting the prediction right!



DEEP LEARNING FRAMEWORK

We are building this example with Keras

THERE ARE MANY OTHERS TO CHOOSE FROM

Caffe

PYTORCH

theano

TRAINING AN RNN IN PYTHON

```
def deep_predict(prices):
    p = 10
    model = get_rnn_model()
    x, y = build_lstm_data(prices, 50)
    model.fit(x, y, batch_size=512, nb_epoch=1, validation_split=0.05)
    return rnn_predict(model, x, prices)
```

- Build model
- Compute the weights
- Return the prediction

NOT TOO DIFFERENT, EH!

```
from sklearn import linear_model

def predict(prices, times, predict=10):
    model = linear_model.LinearRegression()
    model.fit(times, prices)
    predict_times = get_prediction_timeline(times, predict)
    return model.predict(predict_times)
```

```
def deep_predict(prices):
    p = 10
    model = get_rnn_model()
    x, y = build_lstm_data(prices, 50)
    model.fit(x, y, batch_size=512, nb_epoch=1, validation_split=0.05)
    return rnn_predict(model, x, prices)
```

TO BUILD THE LSTM

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm

def get_rnn_model():
    model = Sequential()
    model.add(LSTM(input_dim=1, output_dim=50, return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(100, return_sequences=False))
    model.add(Dropout(0.2))
```

In this case 50 recurrent layers that pass output to next layer...

...and 100 that don't pass output until end of sequence

(+ regulatisation)

FINISHING THE MODEL

```
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
import lstm

def get_rnn_model():
    model = Sequential()
    model.add(LSTM(input_dim=1, output_dim=50, return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(100, return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(output_dim=1))
    model.add(Activation('linear'))

    model.compile(loss="mse", optimizer="rmsprop")

    return model
```

A linear dense layer to aggregate the data into a single value

Compile with mean sq. error & gradient descent as optimizer

Simple!

RNN TEST-RUN

```
from crypto_ml.data_loader import CryptoLoader


cl = CryptoLoader()

df = cl.get_df("bitcoin")

times = df[["Date"]].values
prices = df[["Price"]].values

results = deep_predict(prices, times, 5)
```

SUCCESS

CRYPTO-ML

HAS THE ML!

Are we done then?

NOPE

The fun is just starting

3. DISTRIBUTED ARCHITECTURE

After CryptoML was caught using deep learning...

...they were featured in the top 10 global news

THEIR USERBASE EXPLODED

Now they have quite a few users coming in every day

Each user is running several ML algorithms concurrently

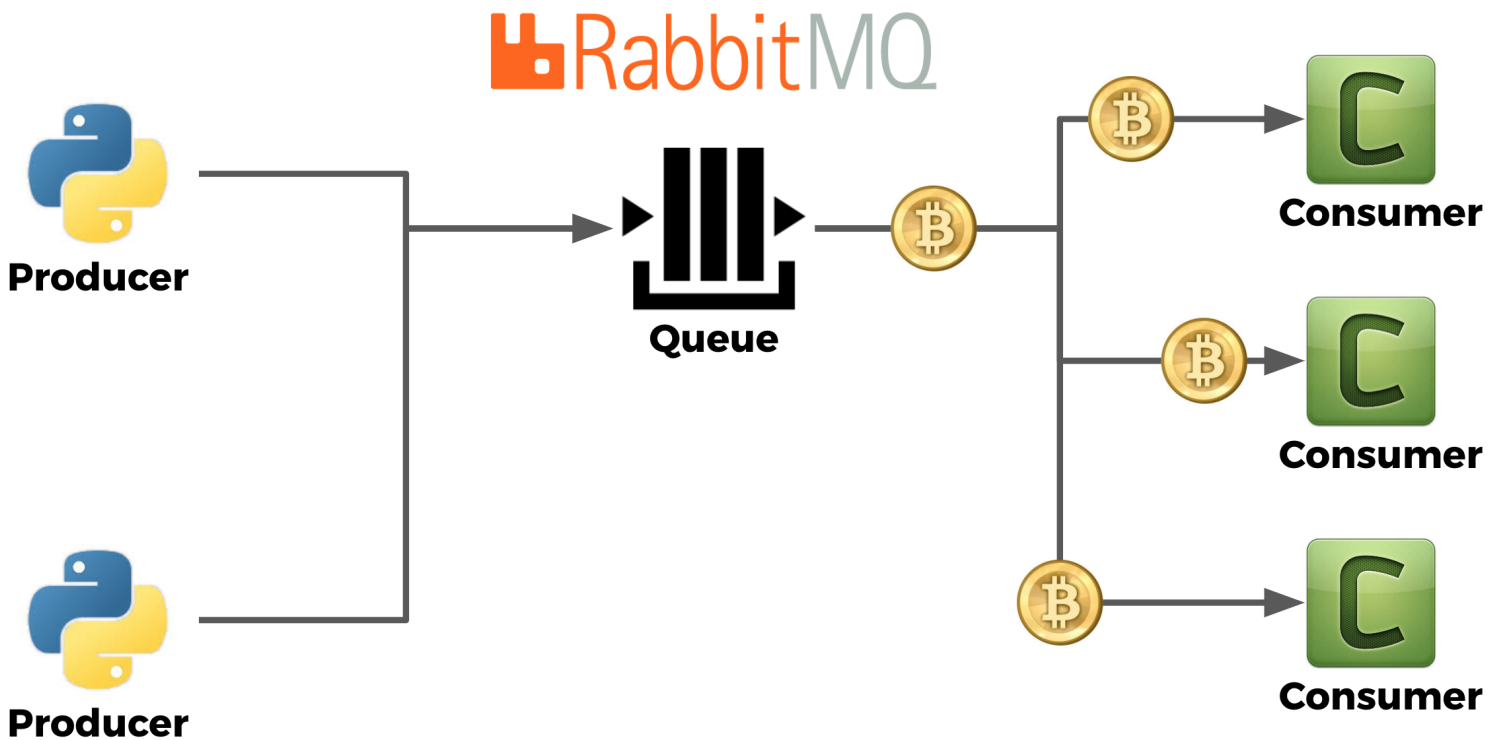
They tried getting larger and larger AWS servers

THEY SHOULD'VE SEEN THIS COMING

- Machine learning is known for being computing heavy
- But often it's forgotten how memory-heavy it is
- I'm talking VERY heavy - holding whole models in-mem
- Scaling to bigger instances with more cores is expensive
- Having everything in one node is a central point of failure

IT'S TIME TO GO DISTRIBUTED

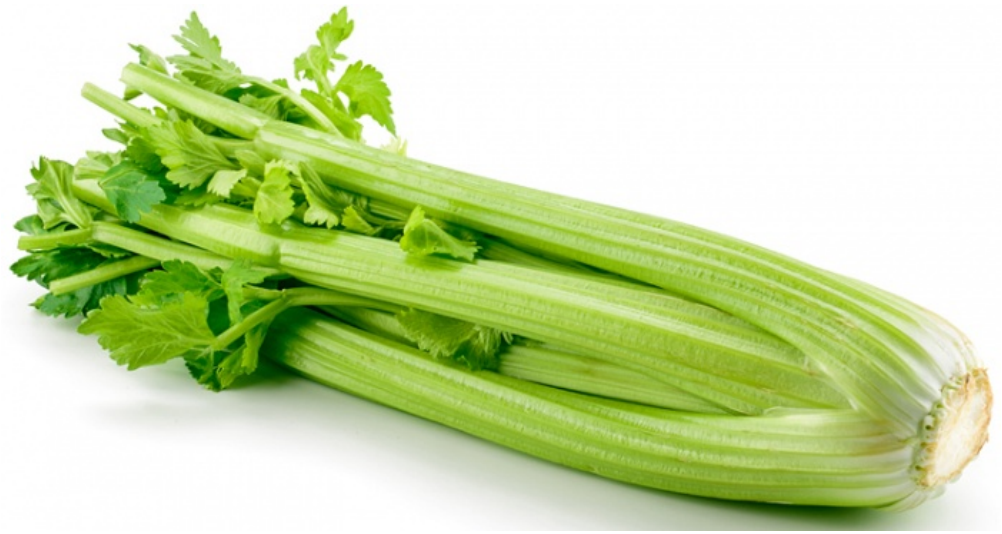
PRODUCER-CONSUMER ARCHITECTURE



The Crypto-ML Devs thought go distributed was too hard

It's not.

INTRODUCING CELERY



- Distributed
- Asynchronous
- Task Queue
- For Python

STEP 1: TAKE YOUR CODE

```
def deep_predict(prices, times, predict=10):
    model = utils.get_rnn_model()
    model.fit(time, prices, batch_size=512, nb_epoch=1, validation_split=0.05)
    predict_times = get_prediction_timeline(times, predict)
    return model.predict(predict_times)
```

STEP 2: CELERIZE IT

```
@app.task
def deep_predict(prices, times, predict=10):

    model = utils.get_rnn_model()

    model.fit(time, prices, batch_size=512, nb_epoch=1, validation_split=0.05)

    predict_times = get_prediction_timeline(times, predict)

    return model.predict(predict_times)
```

STEP 3: MAKE SURE IT WORKS

```
from celery import Celery
from utils import load, dump

app = Celery('crypto_ml',
    backend='amqp://guest@localhost/',
    broker='amqp://guest@localhost/')

@app.task
def deep_predict(d_prices, d_times, predict=10):

    # Load from stringified binaries (Pandas Dataframes)
    prices = load(d_prices)
    times = load(d_times)

    model = utils.get_rnn_model()

    model.fit(time, prices, batch_size=512, nb_epoch=1, validation_split=0.05)

    predict_times = get_prediction_timeline(times, predict)

    return dump(model.predict(predict_times))
```

Disclaimer: I haven't for this snippet

STEP 4: RUN IT!

```
$ celery -A crypto_ml worker
```

See the activity logs:

```
Darwin-15.6.0-x86_64-i386-64bit 2018-03-10 00:43:28

[config]
.> app:          crypto_ml:0x10e81a9e8
.> transport:    amqp://user:**@localhost:5672//
.> results:      amqp://
.> concurrency: 8 (prefork)
.> task events: OFF (enable -E to monitor tasks in this worker)

[queues]
.> celery           exchange=celery(direct) key=celery
```

WE'RE ALREADY HALFWAY THERE

Now we just need to make the producer!

We can just follow the same recipe

STEP 1: TAKE THE CODE

```
cl = CryptoLoader()
results = {}

# Compute results
for name in cl.get_crypto_names():

    prices, times = cl.get_prices(name)

    result = deep_predict(prices, times)

    results[name] = result

# Print results
for k,v in results.items():
    print(k, v)
```

STEP 2: CELERIZE IT

```
cl = CryptoLoader()
results = {}

# Send task for distributed computing
for name in cl.get_crypto_names():

    prices, times = cl.get_prices(name)

    task = deep_predict.delay(
                      dump(prices)
                    , dump(times))

    results[name] = task

# Wait for results and print
for k,v in results.items():
    p_result = v.get()

    result = load(p_result)

    print(k, result)
```

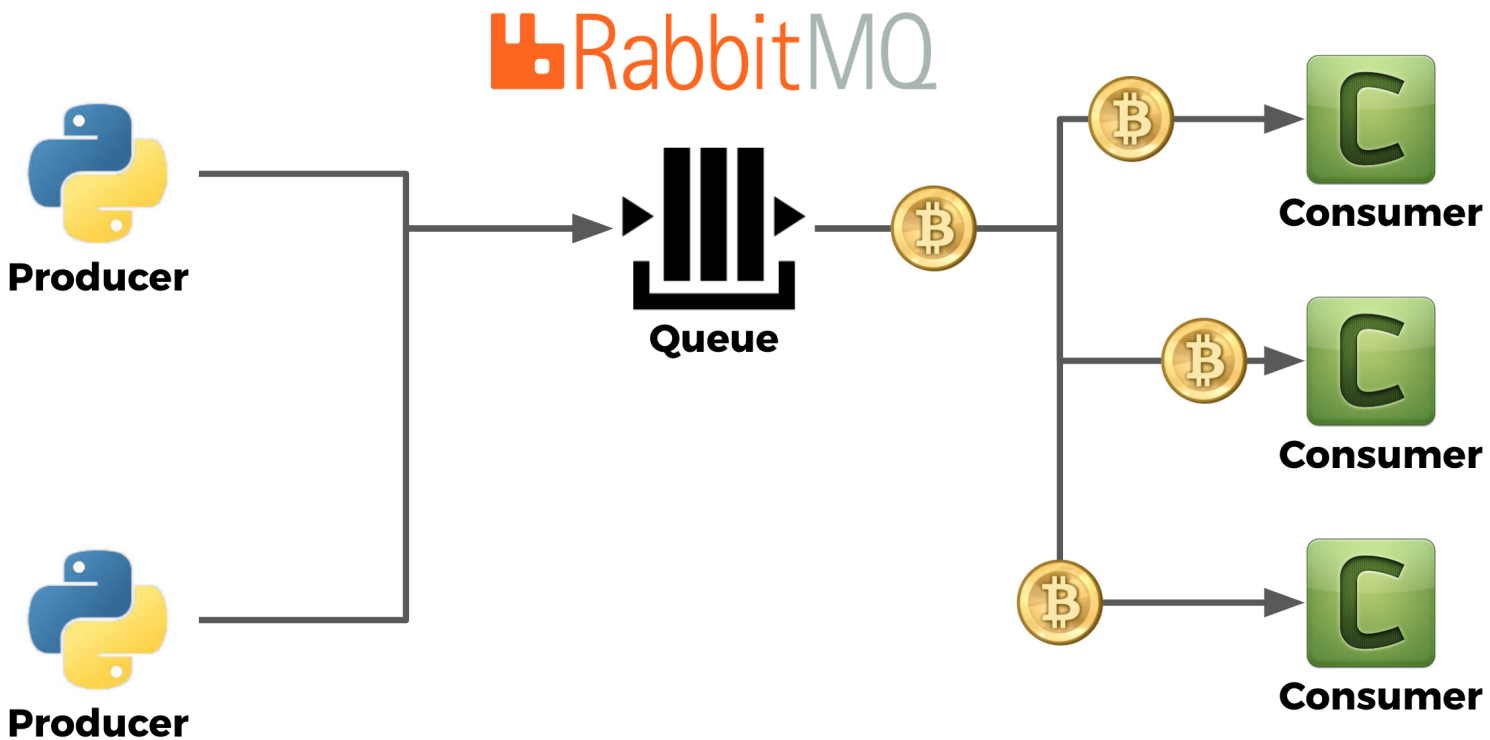
STEP 3: MAKE SURE IT WORKS

```
from crypto_ml.data_loader import CryptoLoader
from util import load, dump

cl = CryptoLoader()
results = {}

# Send task for distributed computing
for name in cl.get_crypto_names():

    prices, times = cl.get_prices(name)

    task = deep_predict.delay(
                        dump(prices)
                      , dump(times))

    results[name] = task

# Wait for results and print
for k,v in results.items():
    p_result = v.get()

    if result:
        result = load(p_result)
        print(k, result)
```

Disclaimer: I haven't for this snippet

STEP 4: RUN IT!

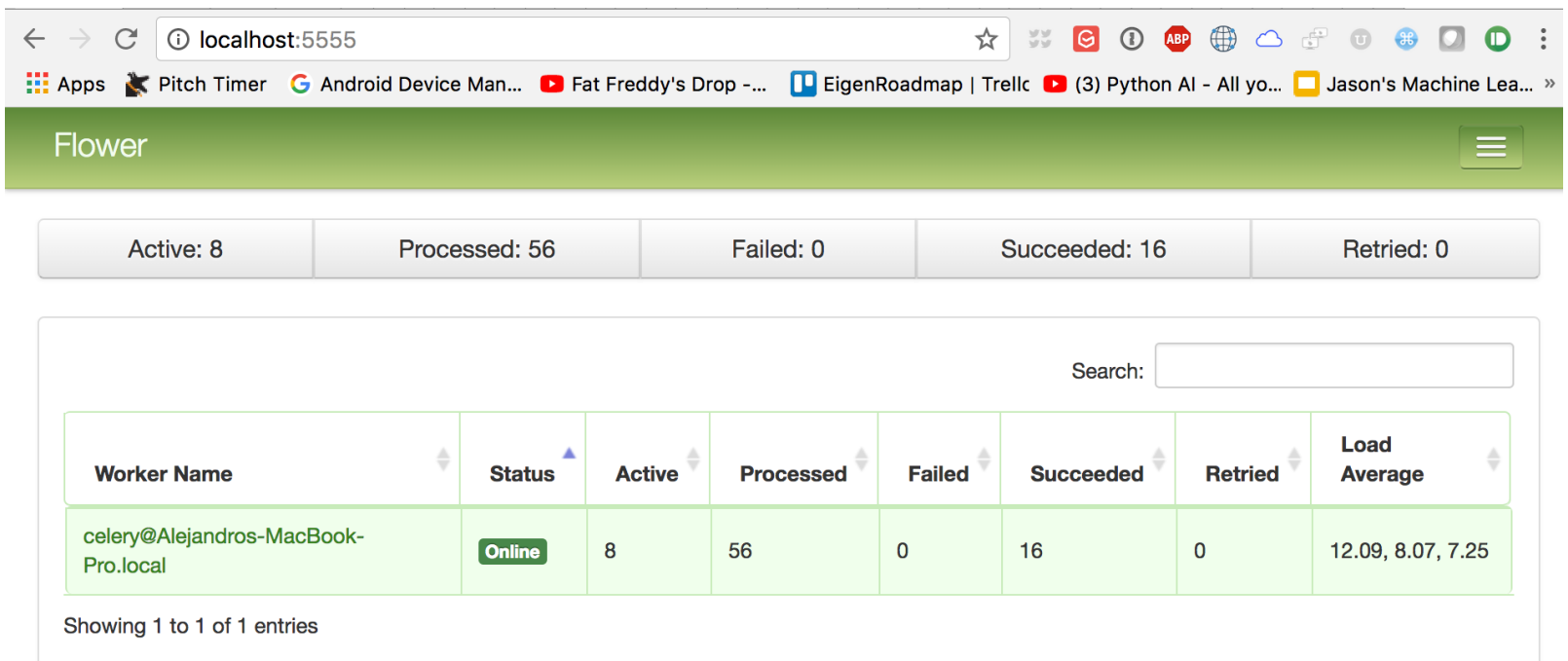
By just running the Python in a shell command!



VISUALISE IT

You can visualise through Celery "Flower"

```
$ celery -A crypto_ml flower
> INFO: flower.command: Visit me at http://localhost:5555
```



DISTRIUTED #WIN

We now have ML, and are distributed.

We have surely won.

NOT REALLY

Having a horizontal architecture is just the first step...

Now we have to take full advantage of it!

4. ELASTIC DEVOPS INFRASTRUCTURE

The CryptoML team remembers the easy days...

...when they only had a few new users a day

Now they have much heavier traffic!

They are working with large organsations!

They are processing massive loads of ML requests!

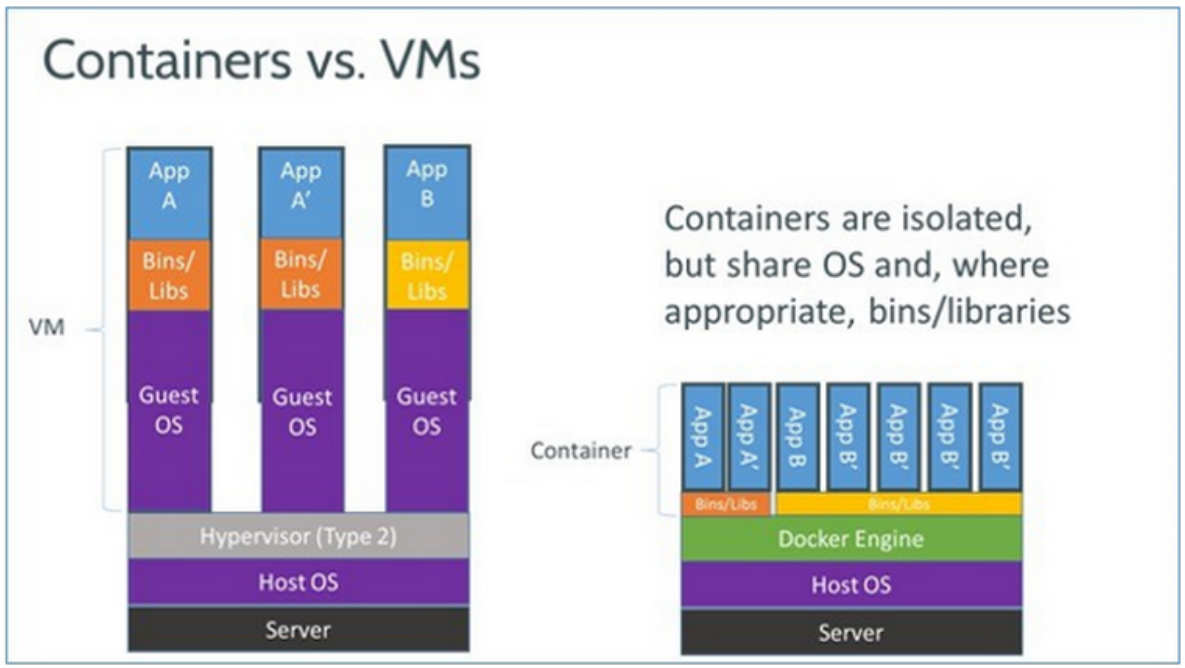
THEIR DEVOPS INFRASTRUCTURE

CAN'T KEEP UP!

UNDERESTIMATING DEVOPS COMPLEXITY

- Complexity of staging and deploying ML models
- Backwards compatibility of feature-spaces/models
- Distributing load across infrastructure
- Idle resource time minimisation
- Node failure back-up strategies
- Testing of Machine Learning functionality
- Monitoring of ML ecosystem
- And the list goes on and on...

DID ANYONE SAY DOCKER?



Package it. Ship it.

GOTTA LOVE THE SIMPLICITY

```
from conda/miniconda3-centos7:latest

ADD . /crypto_ml/
WORKDIR /crypto_ml/

# Dependency for one of the ML predictors
RUN yum install mesa-libGL -y

# Install all the dependencies
RUN conda env create -f crypto_ml.yml
```

and...

```
docker -t crypto_ml .
```

it's done!

CAN THIS BE MORE AWESOME?

YES IT CAN!

Packaging it up and installing through pip/conda.

```
from conda/miniconda3-centos7:latest
RUN conda install <YOUR_PACKAGE>
```

Nice and simple!

THE OBVIOUS DOCKER-COMPOSE

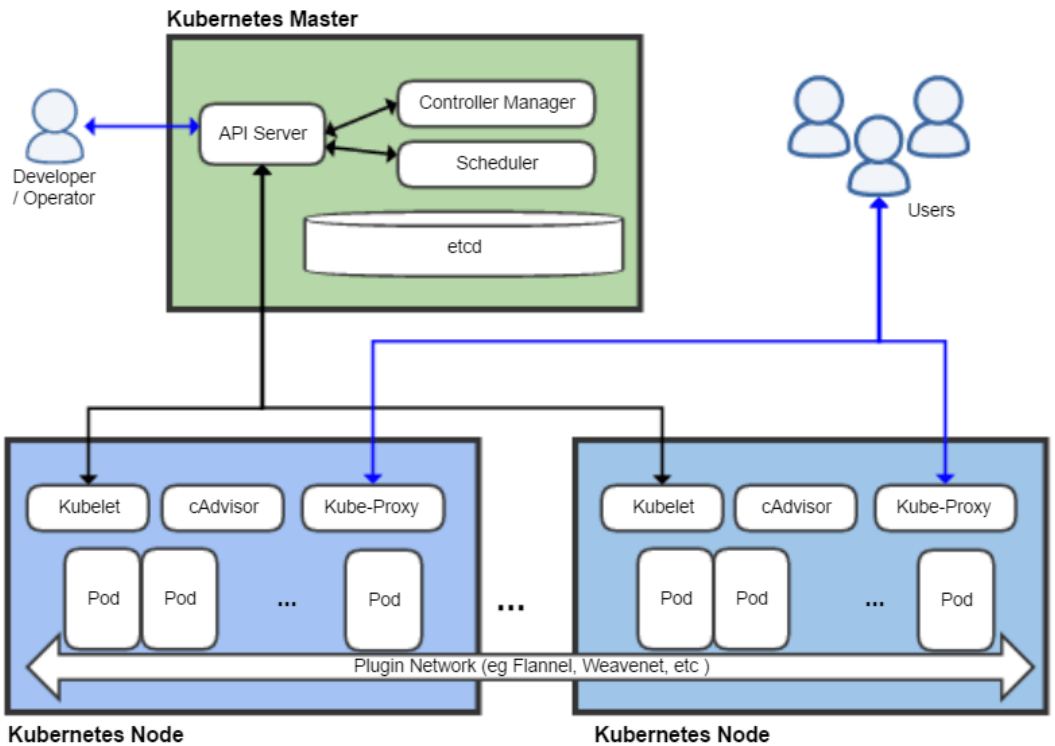
```
version: '2'
services:
    manager:
        container_name: crypto_manager
        image: crypto_ml
        build: .
        links:
            - rabbitmq
        depends_on:
            - rabbitmq
        command: tail -f /dev/null
    worker:
        container_name: crypto_worker
        image: crypto_ml
        build: .
        links:
            - rabbitmq
        depends_on:
            - rabbitmq
        command: /usr/local/envs/crypto_ml/bin/celery -A crypto_ml worker --prefetch-multiplier 1 --max-tasks-per-
    rabbitmq:
        container_name: rabbitmq
        image: rabbitmq:3.6.0
        environment:
            - RABBITMQ_DEFAULT_USER=user
            - RABBITMQ_DEFAULT_PASS=1234
        ports:
            - 4369
            - 5672
            - 15672
```

It all just works automagically!

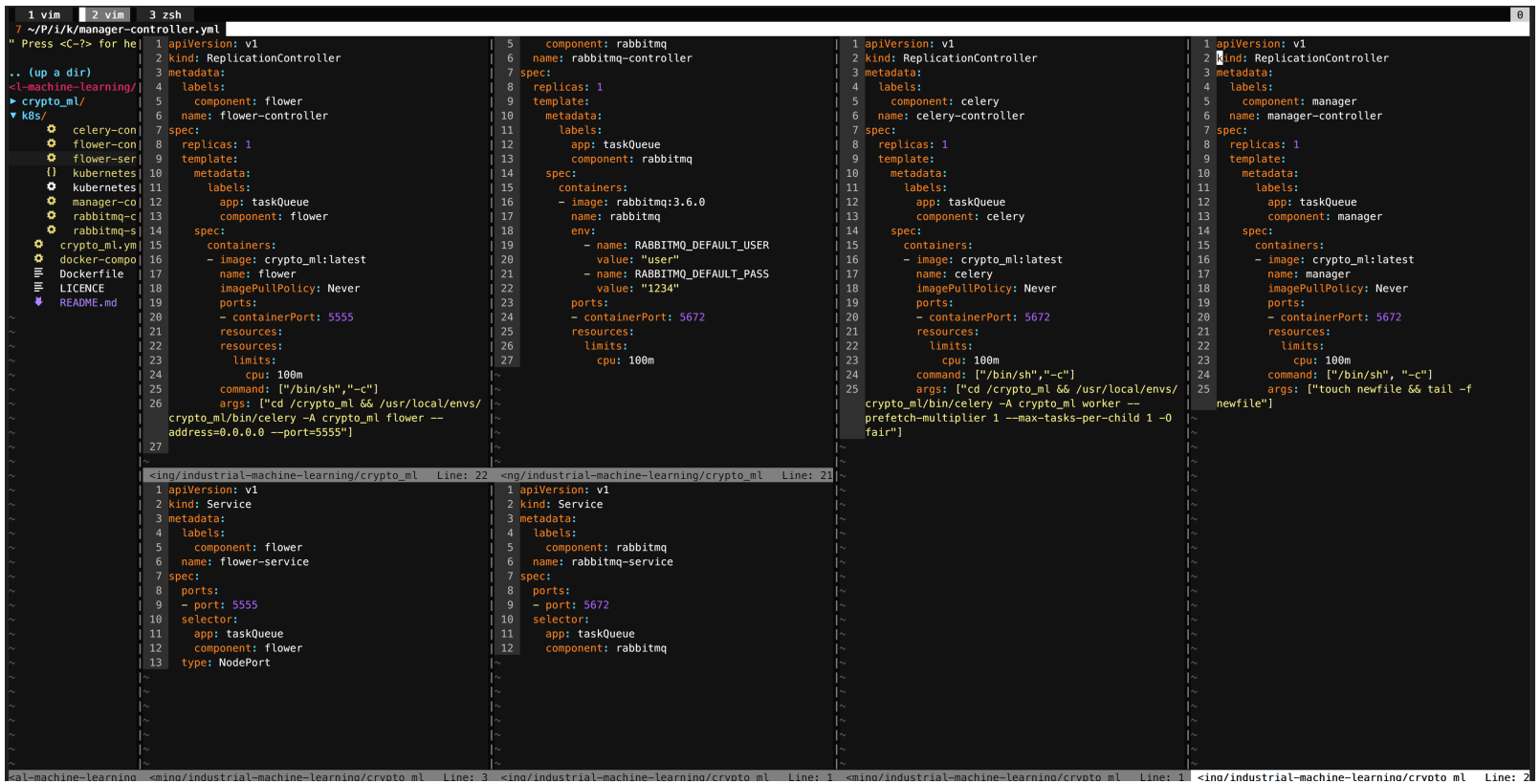
```
docker-compose up --scale crypto_worker=4
```

TAKING IT TO THE NEXT LEVEL WITH

KUBERNETES



ALL THE YML



Creating all the services and controllers

```
kubectl create -f k8s/*
```

KUBERNETES: DEV

DEVELOPMENT

- Minikube
- Docker for Mac

KUBERNETES: PROD

ENTERPRISE

- Elastic Container Service for K8s (AWS)
- CoreOS Tectonic
- Red Hat OpenShift

KUBERNETES: TOOLS

MANY OPTIONS

- CloudFormation (AWS)
- Terraform
- Kops

DOCKER FOR MAC NOW SUPPORTS IT!

Mini fist-pump

What's next for Crypto-ML?

As with every other tech startup

the roller-coaster keeps going!

BUT FOR US?

Obtained an intuitive understanding on ML

Learned about caveats on practical ML

Obtained tips on building distributed architectures

Got an taste of elastic DevOps infrastructure

CODE

https://github.com/axsauze/crypto-ml

SLIDES

http://github.com/axsauze/industrial-machine-learning

INDUSTRIAL MACHINE LEARNING

Alejandro Saucedo

Head of Eng Dept, Eigen
CTO, Exponential Tech
Fellow (AI & ML), The RSA

a@e-x.io

#PYCONSK2018