AI/DL/ML/RL

Edward Mitby, MS, CFA, CMT

Senior Artificial Intelligence Engineer

Vanguard

Artificial Intelligence: What is it?

- Why should I care?
- This is why: Non-linear function approximation is extremely relevant.
- AI as a term is irrelevant. It used to describe rules-based systems.
- Should Machine Learning just be renamed XG Boosting and simplify everything?
- But what about support vector machines, random forests, and all the sci-kit learn models that take 2 lines of code to implement?

Compute + Algorithms = Predictions

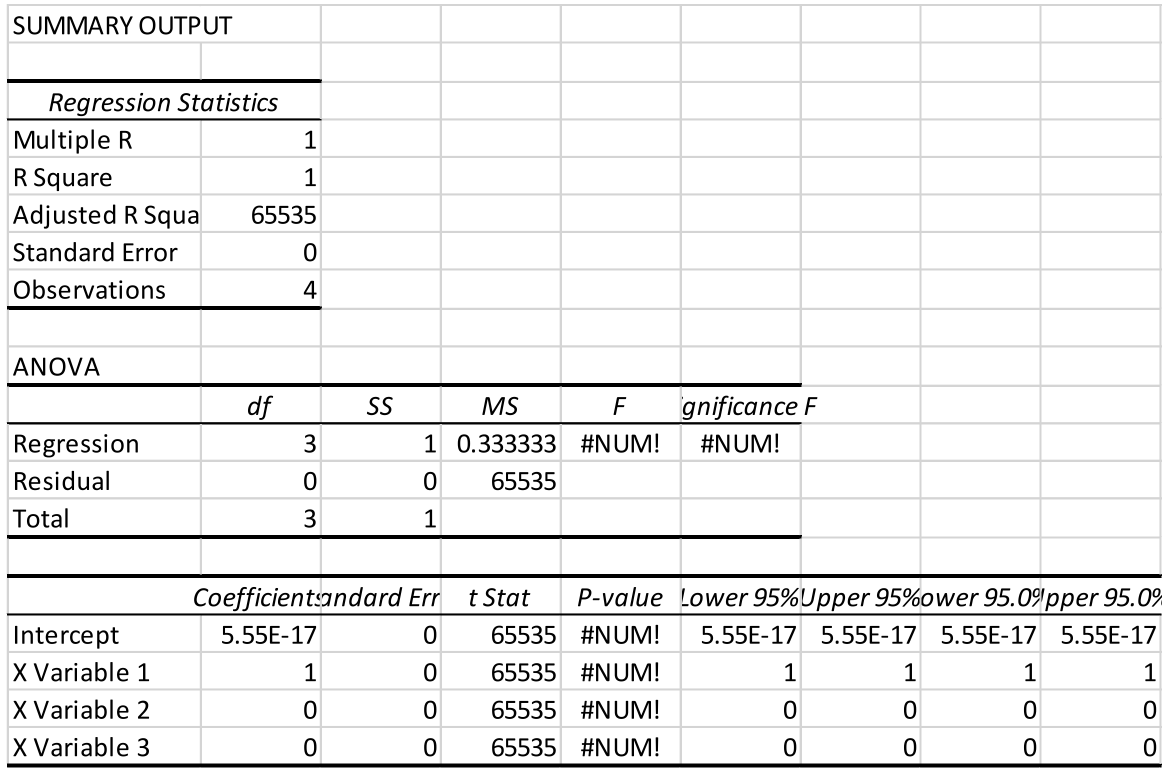
- Simple example:

	EPS Growth	Fed Raising	Margin Growth	6 Month Gain/Loss?
Stock A	0	0	1	0
Stock B	0	1	1	0
IG Bond B	1	0	1	1
Stock C	1	1	1	1

	Chews bubble gum?	Car is Blue?	Has a Cat?	Likes to Cook?
Stock A	0	0	1	0
Stock B	0	1	1	0
IG Bond B	1	0	1	1
Stock C	1	1	1	1

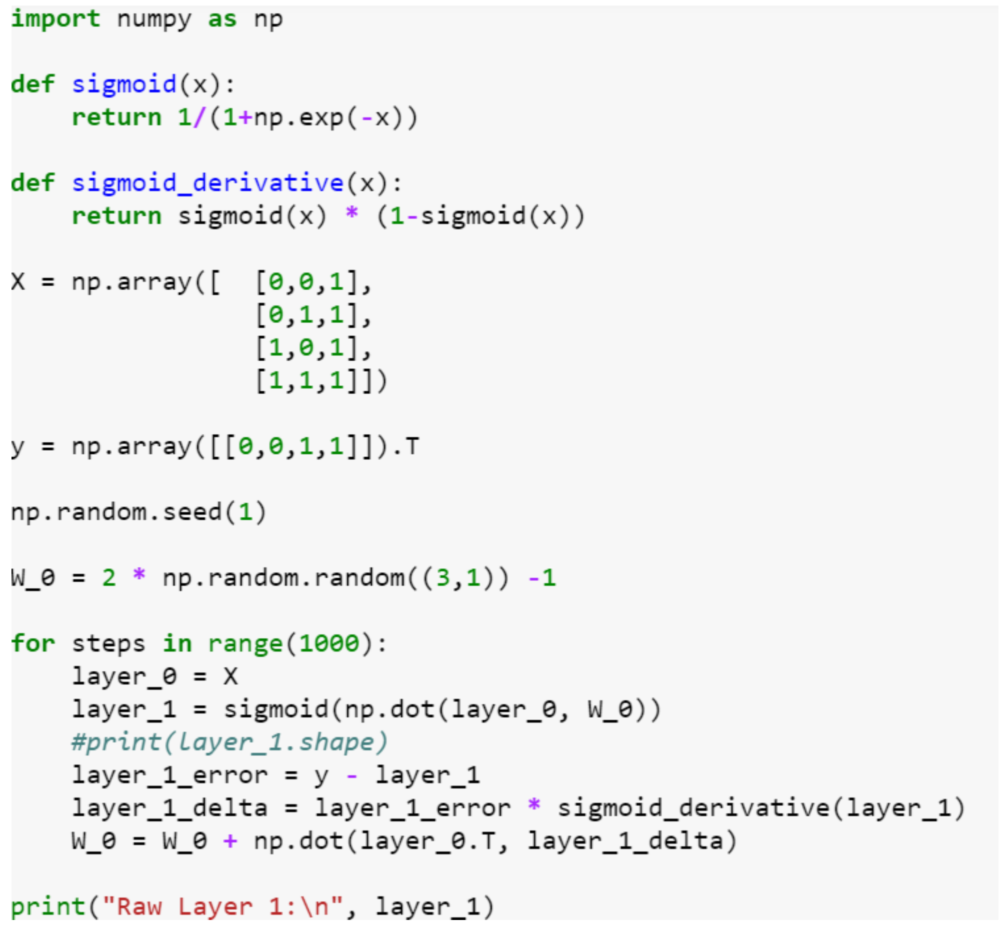
Linear combinations are easily solvable:

- Excel:



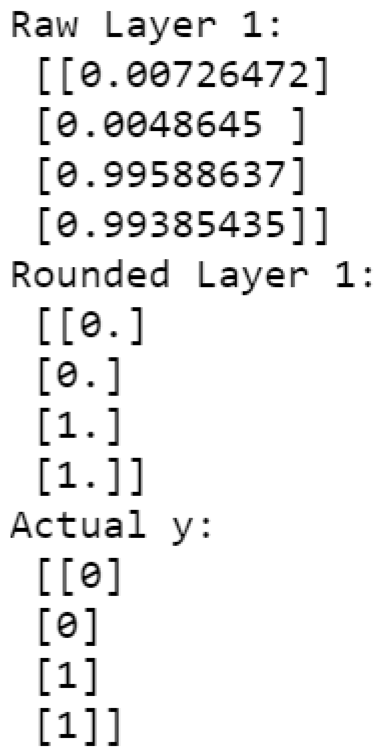
SUMMARY OUTPUT								
Regression Statistics								
Multiple R	1							
R Square	1							
Adjusted R Squa	65535							
Standard Error	0							
Observations	4							
ANOVA								
	df	*SS*	*MS*	*F*	*gnificance F*			
Regression	3	1	0.333333	#NUM!	#NUM!			
Residual	0	0	65535					
Total	3	1						
	Coefficients	*andard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*pper 95.0%*
Intercept	5.55E-17	0	65535	#NUM!	5.55E-17	5.55E-17	5.55E-17	5.55E-17
X Variable 1	1	0	65535	#NUM!	1	1	1	1
X Variable 2	0	0	65535	#NUM!	0	0	0	0
X Variable 3	0	0	65535	#NUM!	0	0	0	0

A simple neural network:



```python
import numpy as np

def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x) * (1-sigmoid(x))

X = np.array([  [0,0,1],
                [0,1,1],
                [1,0,1],
                [1,1,1]])

y = np.array([[0,0,1,1]]).T

np.random.seed(1)

W_0 = 2 * np.random.random((3,1)) -1

for steps in range(1000):
    layer_0 = X
    layer_1 = sigmoid(np.dot(layer_0, W_0))
    #print(layer_1.shape)
    layer_1_error = y - layer_1
    layer_1_delta = layer_1_error * sigmoid_derivative(layer_1)
    W_0 = W_0 + np.dot(layer_0.T, layer_1_delta)

print("Raw Layer 1:\n", layer_1)
```

The output of a simple NN = Excel MR

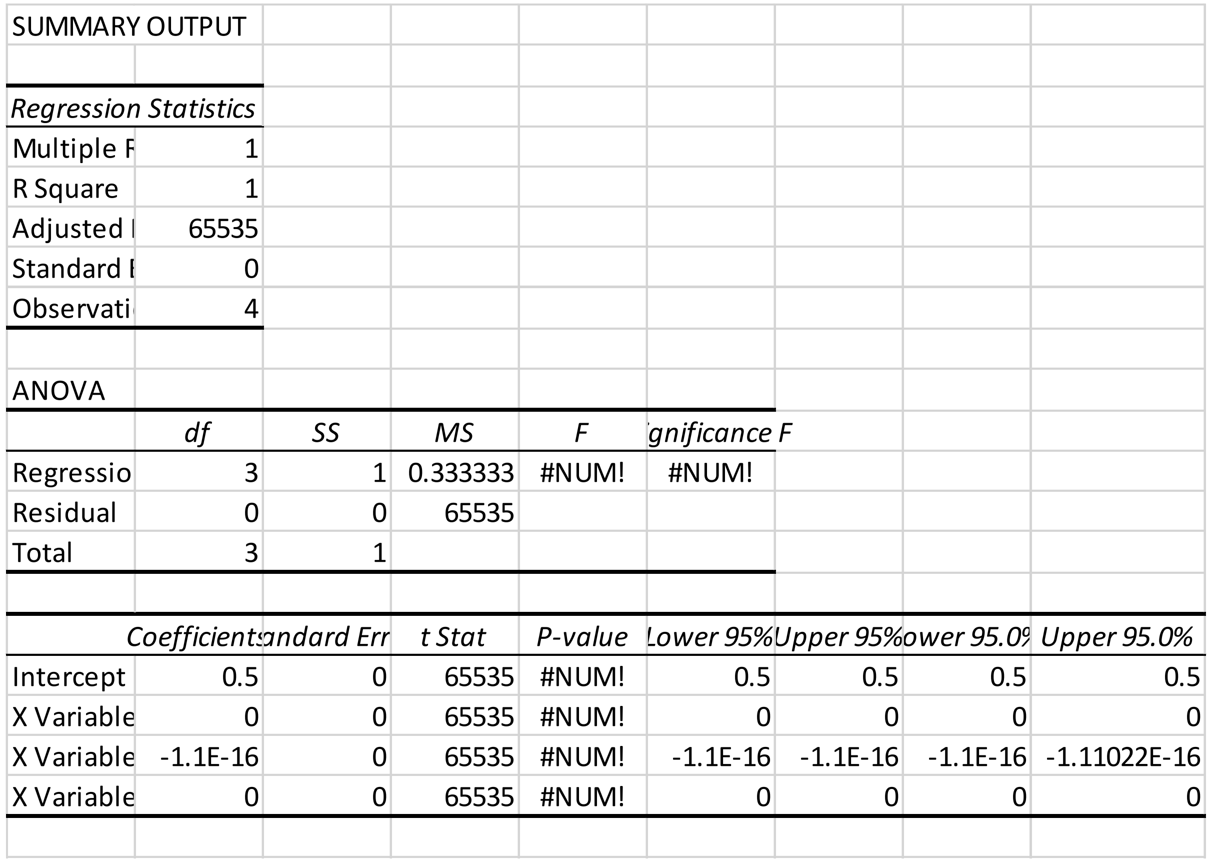


```
Raw Layer 1:
 [[0.00726472]
 [0.0048645 ]
 [0.99588637]
 [0.99385435]]
Rounded Layer 1:
 [[0.]
 [0.]
 [1.]
 [1.]]
Actual y:
 [[0]
 [0]
 [1]
 [1]]
```

So what if the problem becomes non-linear?

	EPS Growth	Fed Raising	Margin Growth	6 Month Gain/Loss?
Stock A	0	0	1	0
Stock B	0	1	1	1
IG Bond B	1	0	1	1
Stock C	1	1	1	0
	Chews bubble gum?	Car is Blue?	Has a Cat?	Likes to Cook?
Stock A	0	0	1	0
Stock B	0	1	1	1
IG Bond B	1	0	1	1
Stock C	1	1	1	0

Excel blows sky high:



SUMMARY OUTPUT

Regression Statistics	
Multiple F	1
R Square	1
Adjusted I	65535
Standard E	0
Observati	4

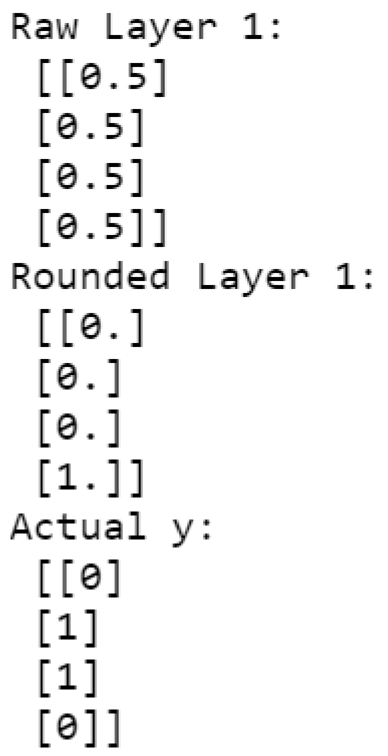
ANOVA

	df	*SS*	*MS*	*F*	*ignificance F*
Regressio	3	1	0.333333	#NUM!	#NUM!
Residual	0	0	65535		
Total	3	1			

	Coefficients	*andard Err*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*ower 95.0%*	*Upper 95.0%*
Intercept	0.5	0	65535	#NUM!	0.5	0.5	0.5	0.5
X Variable	0	0	65535	#NUM!	0	0	0	0
X Variable	-1.1E-16	0	65535	#NUM!	-1.1E-16	-1.1E-16	-1.1E-16	-1.11022E-16
X Variable	0	0	65535	#NUM!	0	0	0	0

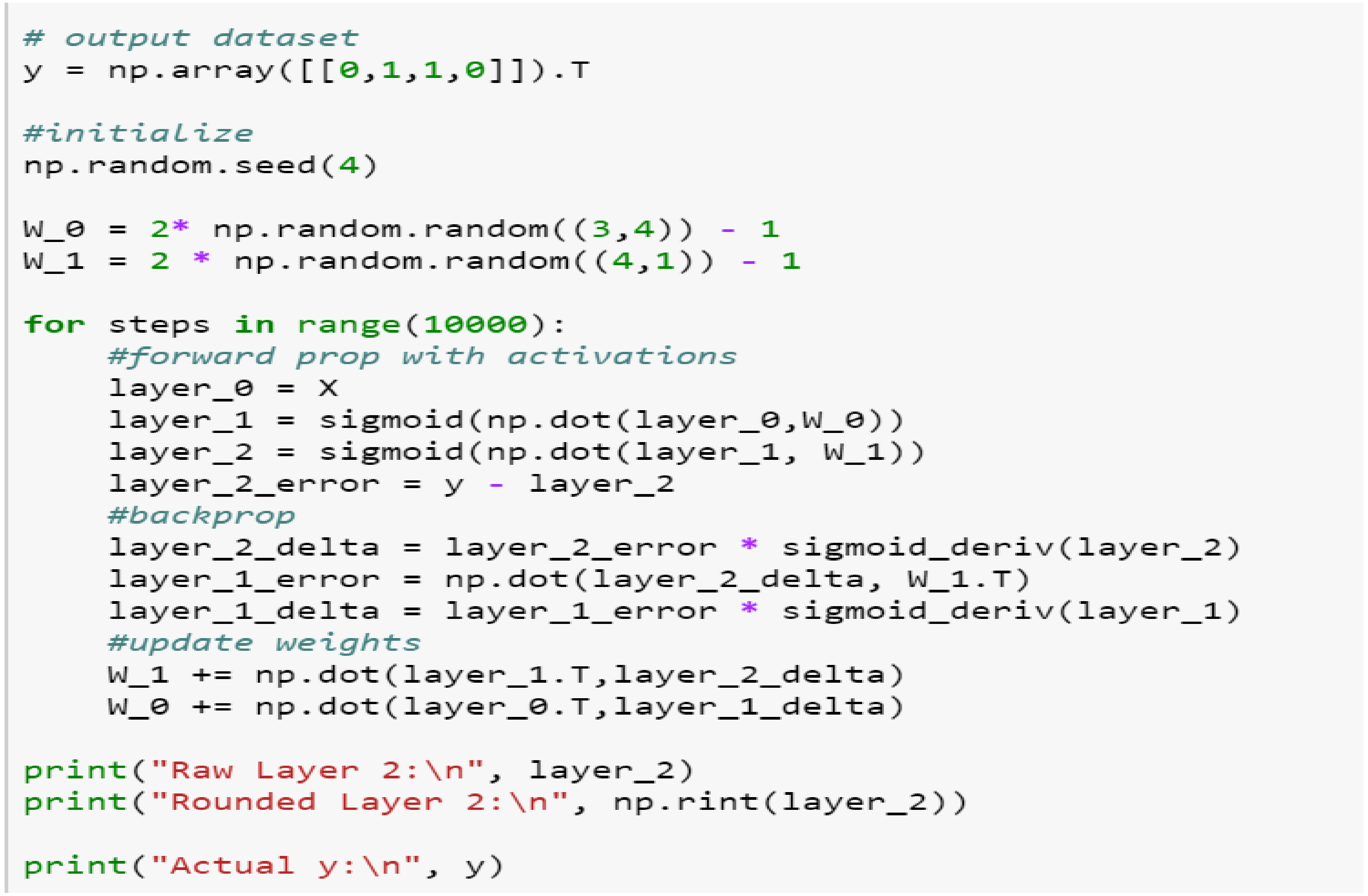
The 1 layer NN blows sky high:

- Generates the same answer as Excel:



```
Raw Layer 1:
 [[0.5]
 [0.5]
 [0.5]
 [0.5]]
Rounded Layer 1:
 [[0.]
 [0.]
 [0.]
 [1.]]
Actual y:
 [[0]
 [1]
 [1]
 [0]]
```

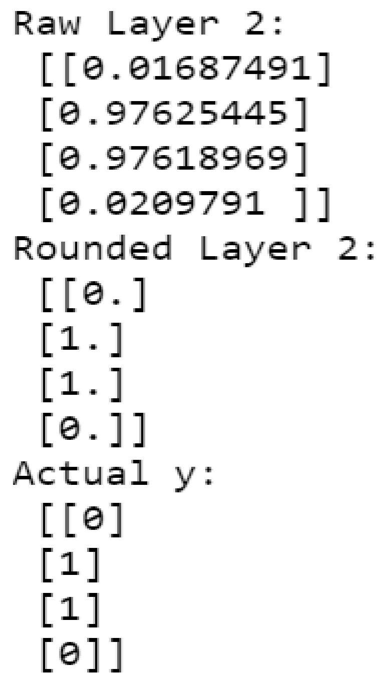
What if we add another layer?



```
# output dataset
y = np.array([[0,1,1,0]]).T

#initialize
np.random.seed(4)

W_0 = 2* np.random.random((3,4)) - 1
W_1 = 2 * np.random.random((4,1)) - 1

for steps in range(10000):
    #forward prop with activations
    layer_0 = X
    layer_1 = sigmoid(np.dot(layer_0,W_0))
    layer_2 = sigmoid(np.dot(layer_1, W_1))
    layer_2_error = y - layer_2
    #backprop
    layer_2_delta = layer_2_error * sigmoid_deriv(layer_2)
    layer_1_error = np.dot(layer_2_delta, W_1.T)
    layer_1_delta = layer_1_error * sigmoid_deriv(layer_1)
    #update weights
    W_1 += np.dot(layer_1.T,layer_2_delta)
    W_0 += np.dot(layer_0.T,layer_1_delta)

print("Raw Layer 2:\n", layer_2)
print("Rounded Layer 2:\n", np.rint(layer_2))

print("Actual y:\n", y)
```

A simple 2-layer network can solve this:

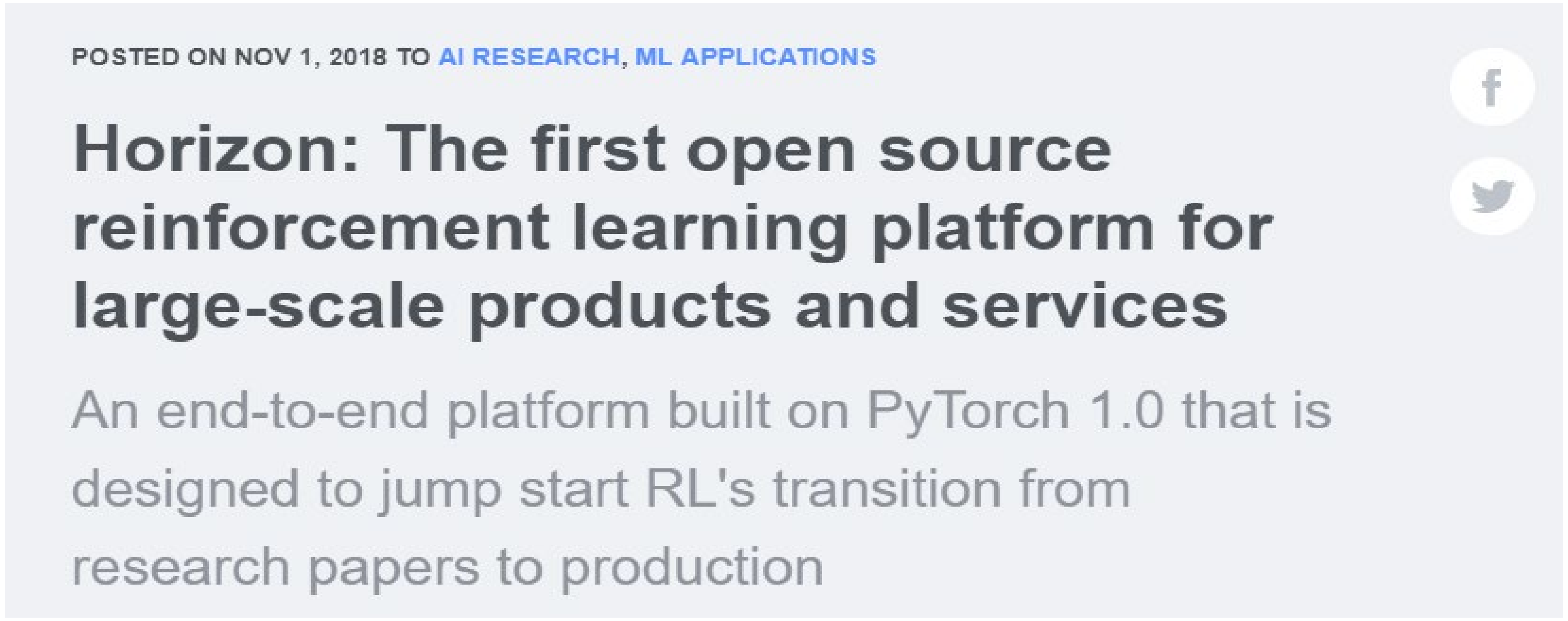
- A simple binary dataset that blows up Excel has a solution, although the compute has to increase significantly:



```
Raw Layer 2:
 [[0.01687491]
 [0.97625445]
 [0.97618969]
 [0.0209791 ]]
Rounded Layer 2:
 [[0.]
 [1.]
 [1.]
 [0.]]
Actual y:
 [[0]
 [1]
 [1]
 [0]]
```

DL is old news. RL?

- Moving very quickly. Is Boston Dynamics using RL or a joystick?
- Possible solution to the regime change problem for “quant” trading.
- **From Github: 27/08/2018:** Dopamine launched!



POSTED ON NOV 1, 2018 TO AI RESEARCH, ML APPLICATIONS

Horizon: The first open source reinforcement learning platform for large-scale products and services

An end-to-end platform built on PyTorch 1.0 that is designed to jump start RL's transition from research papers to production

From Google AI Blog:

- Curiosity and Procrastination in Reinforcement Learning
- **Wednesday, October 24, 2018**
- Posted by Nikolay Savinov, Research Intern, Google Brain Team and Timothy Lillicrap, Research Scientist, DeepMind

- Reinforcement learning (RL) is one of the most actively pursued research techniques of machine learning, in which an artificial agent receives a positive reward when it does something right, and negative reward otherwise. This carrot-and-stick approach is simple and universal, and allowed DeepMind to teach the DQN algorithm to play vintage Atari games and AlphaGoZero to play the ancient game of Go. This is also how OpenAI taught its OpenAI-Five algorithm to play the modern video game Dota, and how Google taught robotic arms to grasp new objects. However, despite the successes of RL, there are many challenges to making it an effective technique.

- **Standard RL algorithms struggle with environments where feedback to the agent is sparse — crucially, such environments are common in the real world.** As an example, imagine trying to learn how to find your favorite cheese in a large maze-like supermarket. You search and search but the cheese section is nowhere to be found. If at every step you receive no “carrot” and no “stick”, there’s no way to tell if you are headed in the right direction or not. In the absence of rewards, what is to stop you from wandering around in circles? Nothing, except perhaps your curiosity, which motivates you go into a product section that looks unfamiliar to you in pursuit of your sought-after cheese.

AI/DL/RL: It’s coming fast…

- Udacity Nanodegree in Deep Reinforcement Learning started in August.
- Coursera Specialization in Reinforcement Learning in Finance started in August.
- “Quantum: A research effort from Google AI that aims to build quantum processors and develop novel quantum algorithms to dramatically accelerate computational tasks for machine learning.”
- https://arxiv.org/abs/1810.04719
- https://github.com/google/uis-rnn

Why should anyone care?

- Financial markets are much different than other environments, but…
- Because if you're not doing it someone else is. And then the limited's start asking questions….
- Silicon Valley frowns on finance so not a focus.
- AMAZON. Applying Deep Learning to stock selection paper was EXCELLENT. They are ~~most likely~~ building a headquarters in Long Island City and Jeff Bezos despises Wall Street. The asset management industry has fat margins and is ripe for disruption. Can you envision Amazon Prime offering SPY at -10 bps?
- Two books: (these are really good and written by your competition):
- **The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World**
- **Advances in Financial Machine Learning**

Additional Topics:

- DL vs RL: subprime default modeling
- Financial time series data: random walks vs. transformations
- “AI Predators” – consultants, ‘research providers’, job-seekers, etc.
- AI hiring landscape
- AMAZON
- What if cash becomes alpha?
- Zip codes
- THE FOURTH INDUSTRIAL REVOLUTION