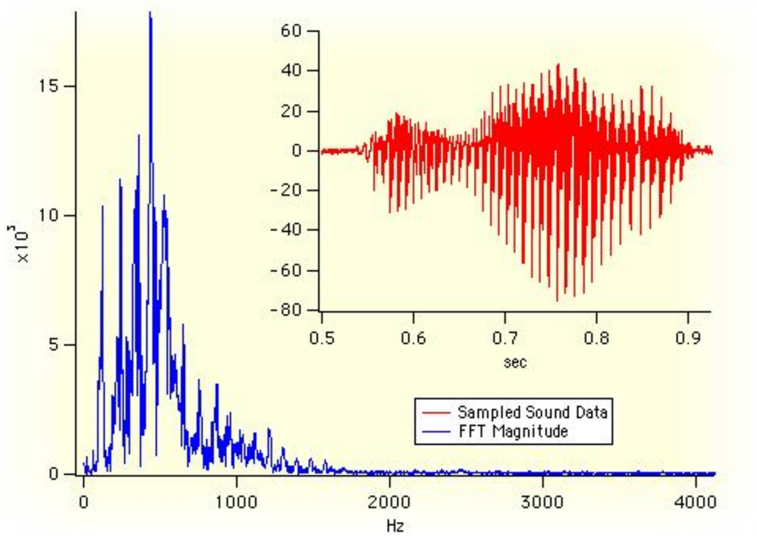
Without The Fluff: Deep Learning

Fourier Transforms



Fourier transform and inverse Fourier transform, respectively (Krantz 1999, p. 202).

Note that some authors (especially physicists) prefer to write the transform in terms of angular fre
destroys the symmetry, resulting in the transform pair

$$
\begin{aligned}
H(\omega) &= \mathcal{F}[h(t)] \\
&= \int_{-\infty}^{\infty} h(t)\, e^{-i\omega t}\, dt \\
h(t) &= \mathcal{F}^{-1}[H(\omega)] \\
&= \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega)\, e^{i\omega t}\, d\omega.
\end{aligned}
$$

To restore the symmetry of the transforms, the convention

$$
\begin{aligned}
g(y) &= \mathcal{F}[f(t)] \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)\, e^{-iyt}\, dt \\
f(t) &= \mathcal{F}^{-1}[g(y)] \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(y)\, e^{iyt}\, dy
\end{aligned}
$$

is sometimes used (Mathews and Walker 1970, p. 102).

In general, the Fourier transform pair may be defined using two arbitrary constants a and b as

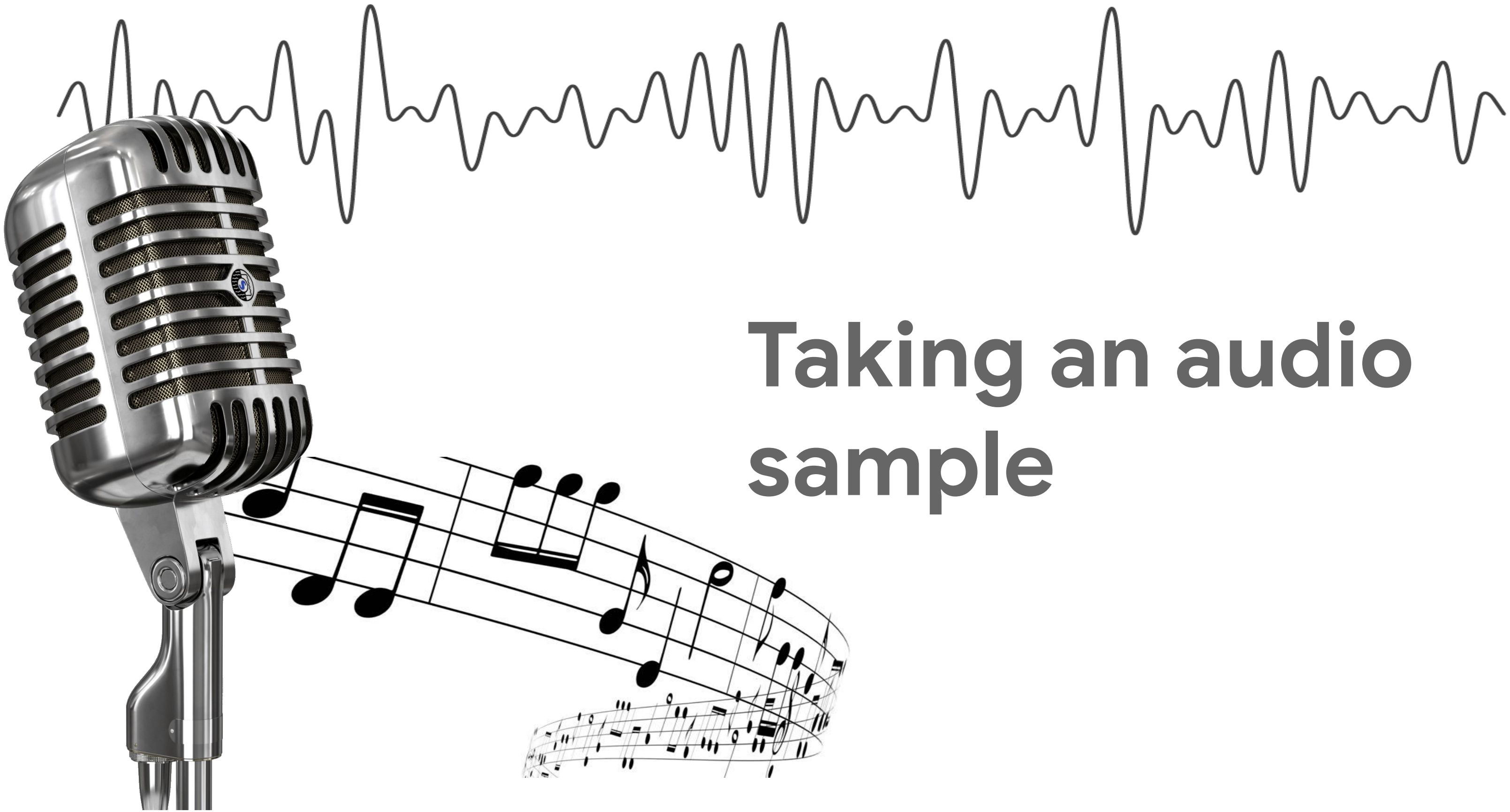
$$
F(\omega) = \sqrt{\frac{|b|}{(2\pi)^{1-a}}} \int_{-\infty}^{\infty} f(t)\, e^{ib\omega t}\, dt
$$

$$
f(t) = \sqrt{\frac{|b|}{(2\pi)^{1+a}}} \int_{-\infty}^{\infty} F(\omega)\, e^{-ib\omega t}\, d\omega.
$$

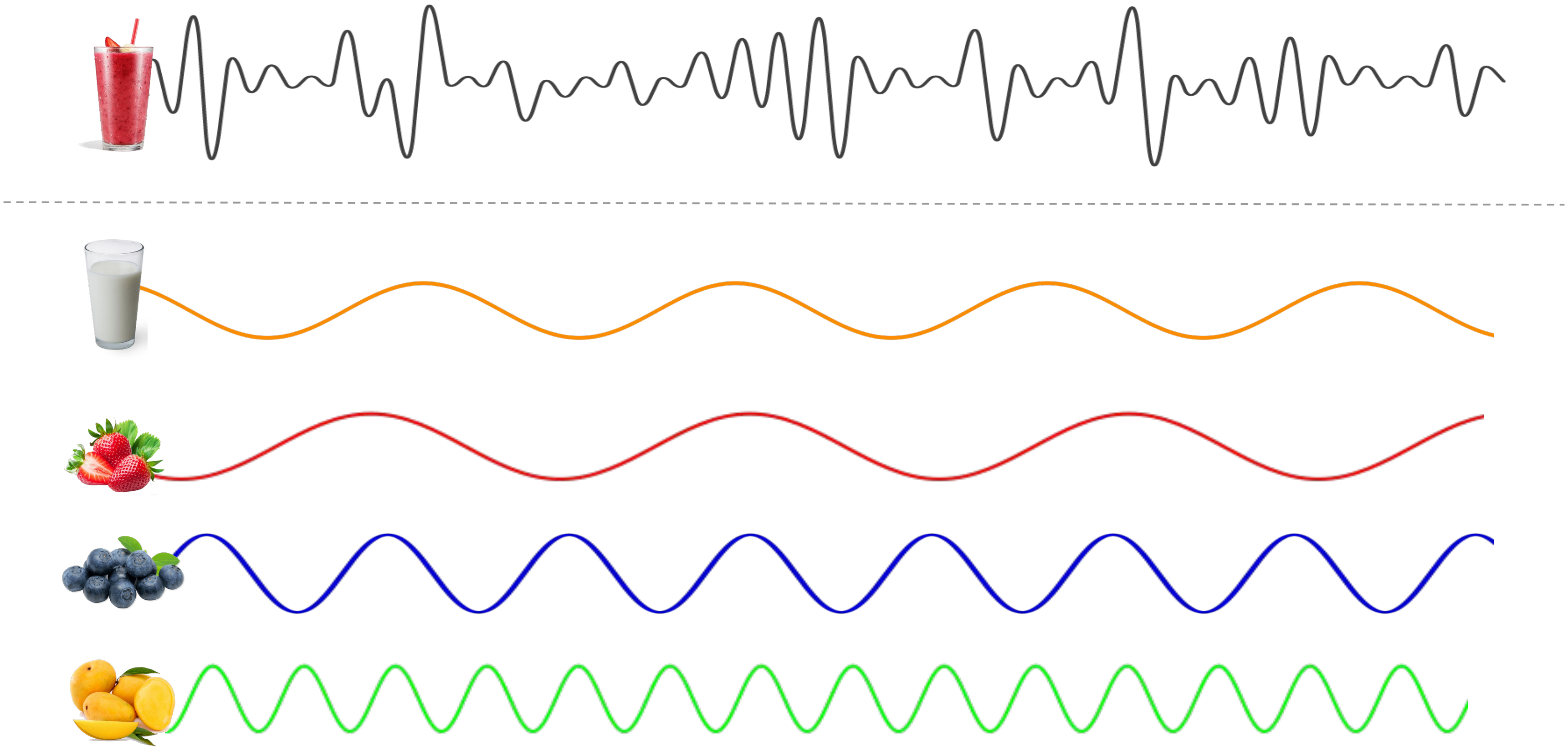
what are this

Let’s look at an example.

Say you have a smoothie.



Taking an audio sample



What is AI?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

Artificial Intelligence

Any technique that enables computers to mimic human behavior

Machine Learning

Ability to learn without being explicitly programmed

Artificial Intelligence

Any technique that enables computers to mimic human behavior

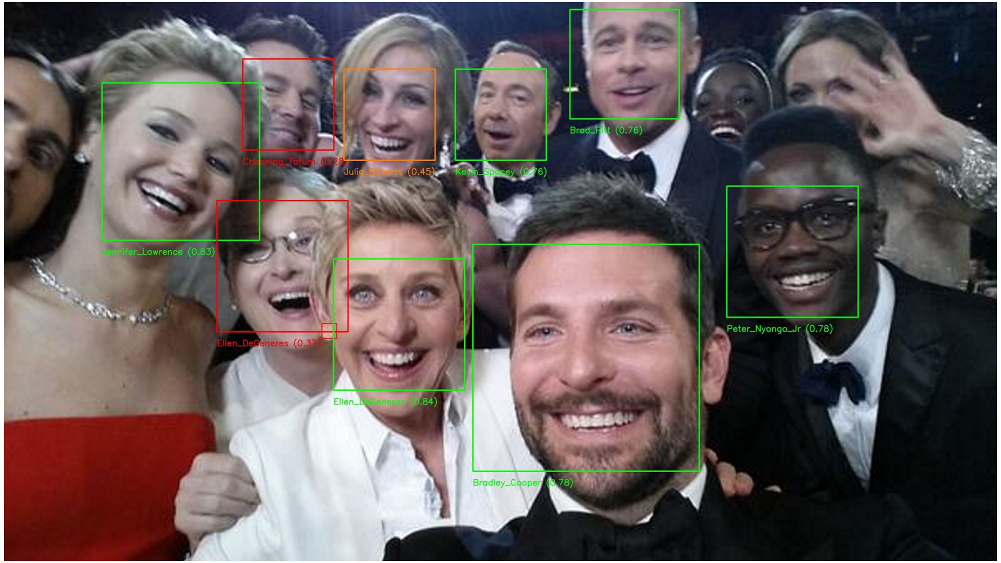
Machine Learning

Ability to learn without being explicitly programmed

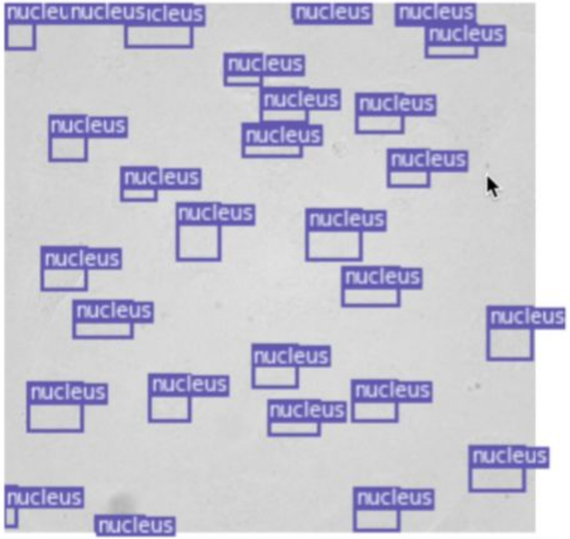
Deep Learning

Learn underlying features in data by using neural networks

Images

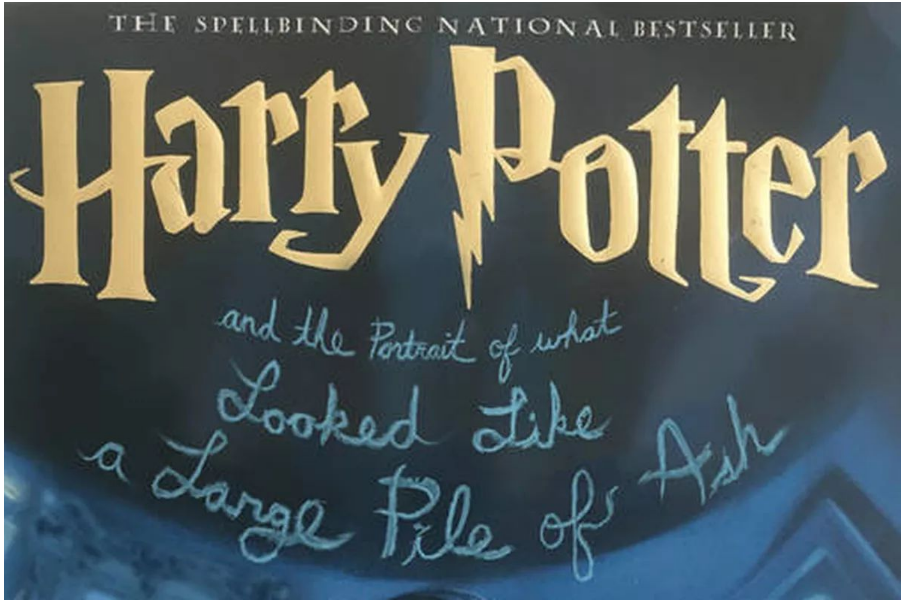


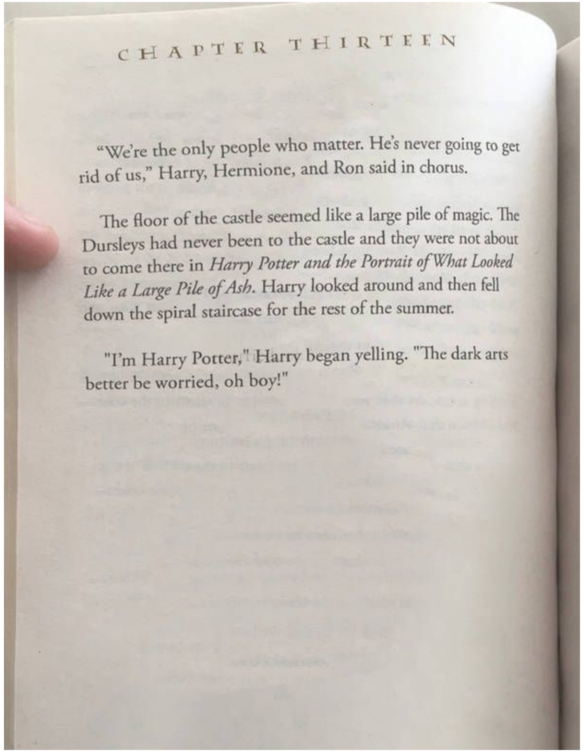
Channing_Tatum (0.28)
Julia_Roberts (0.45)
Kevin_Spacey (0.76)
Brad_Pitt (0.76)
Jennifer_Lawrence (0.83)
Ellen_DeGeneres (0.37)
Ellen_DeGeneres (0.84)
Bradley_Cooper (0.78)
Peter_Nyongo_Jr (0.78)



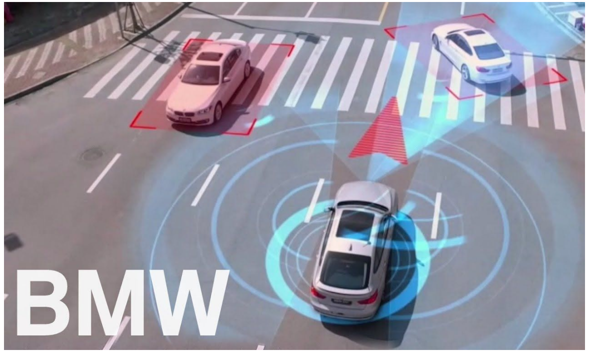
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus
nucleus

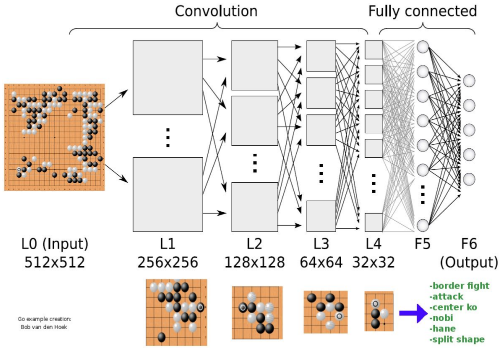
Text


THE SPELLBINDING NATIONAL BESTSELLER
Harry Potter
and the Portrait of what Looked Like a Large Pile of Ash


CHAPTER THIRTEEN
"We're the only people who matter. He's never going to get rid of us," Harry, Hermione, and Ron said in chorus.
The floor of the castle seemed like a large pile of magic. The Dursleys had never been to the castle and they were not about to come there in Harry Potter and the Portrait of What Looked Like a Large Pile of Ash. Harry looked around and then fell down the spiral staircase for the rest of the summer.
"I'm Harry Potter," Harry began yelling. "The dark arts better be worried, oh boy!"

And a whole lot more.





Let’s try an example.

Imagine you work at a cheese factory

Your manager wants you to predict cheddar cheese quality based on data you've obtained from factory sensors.

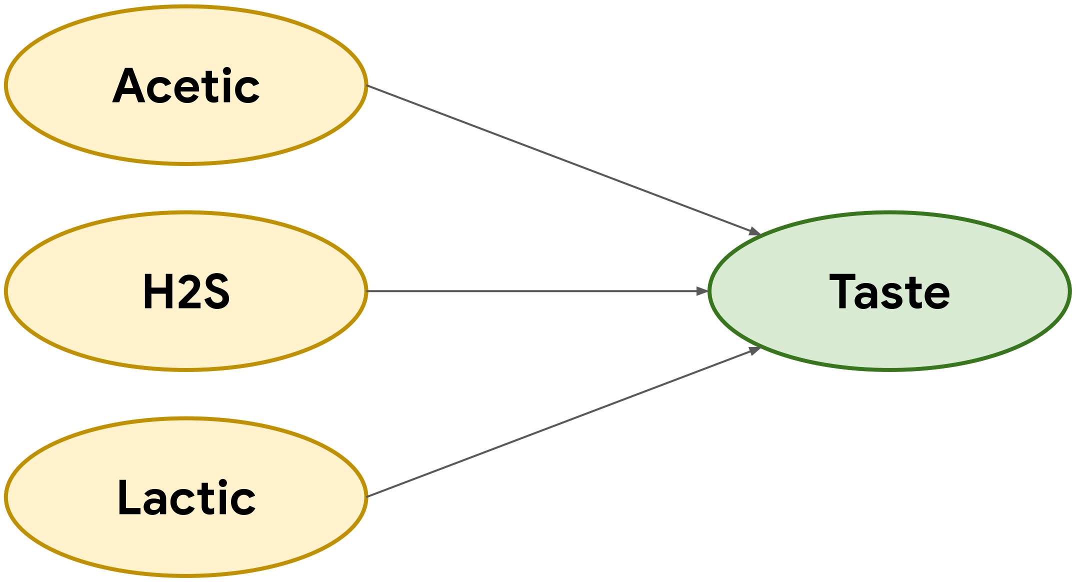
- Acetic Acid
- H2S
- Lactic Acid
- Taste

Data: Cheddar Cheese Quality

Description: Concentrations of acetic acid, H2S, and lactic acid in 30 records of mature cheddar cheese. A subjective taste value is also provided.

Acetic	H2S	Lactic	Taste
4.543	3.135	0.86	12.3
5.159	5.043	1.53	11.7
5.366	5.438	1.57	12.1
5.759	3.807	0.99	7.8

Example: Linear Regression



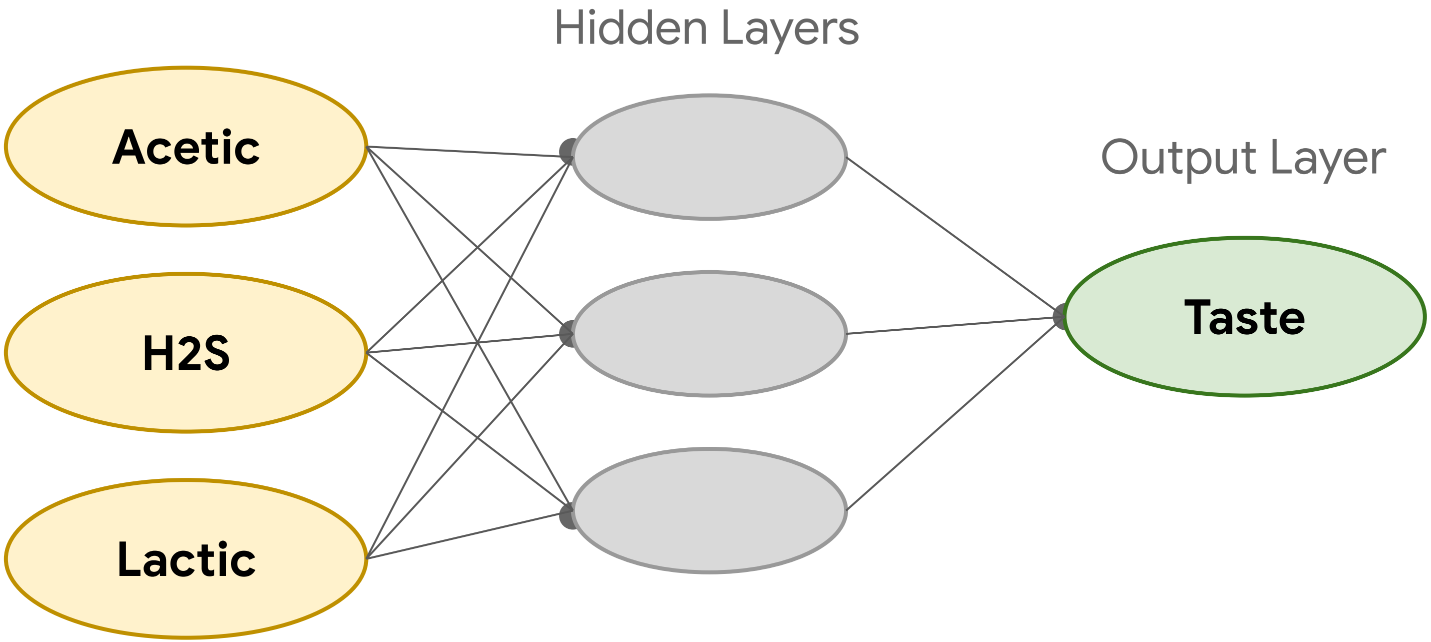
Deep Learning: Interaction Effects

Neural networks are extremely good at dealing with high dimensional data.

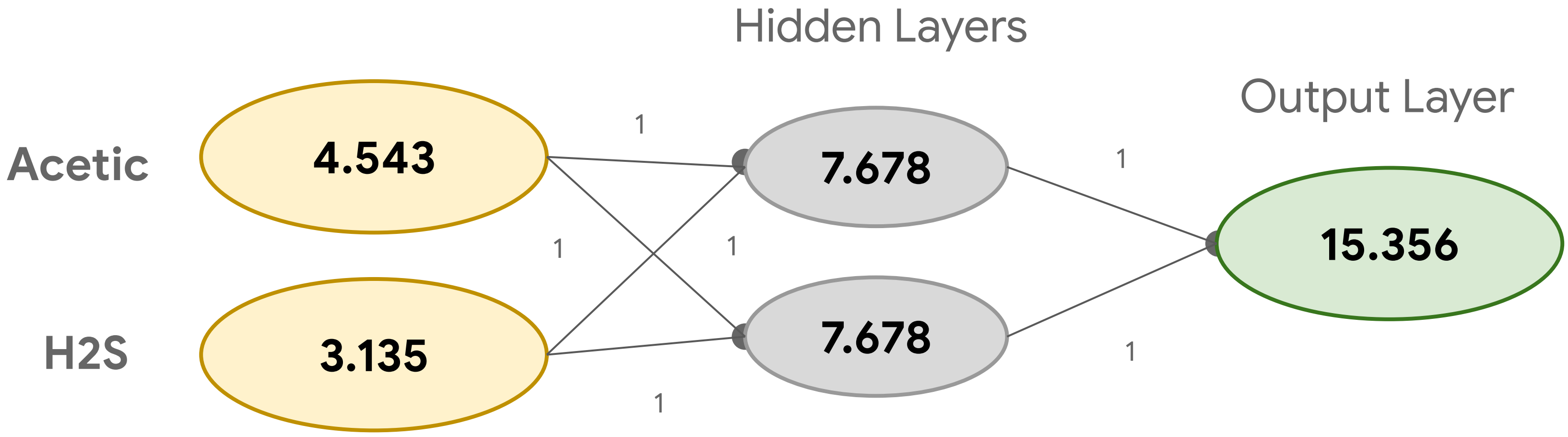
Deep Learning = many layers of nodes

- Forward propagation
- Back propagation
- Gradient descent

Capturing Interactions: Hidden Layers



Let’s try it: Forward Propagation

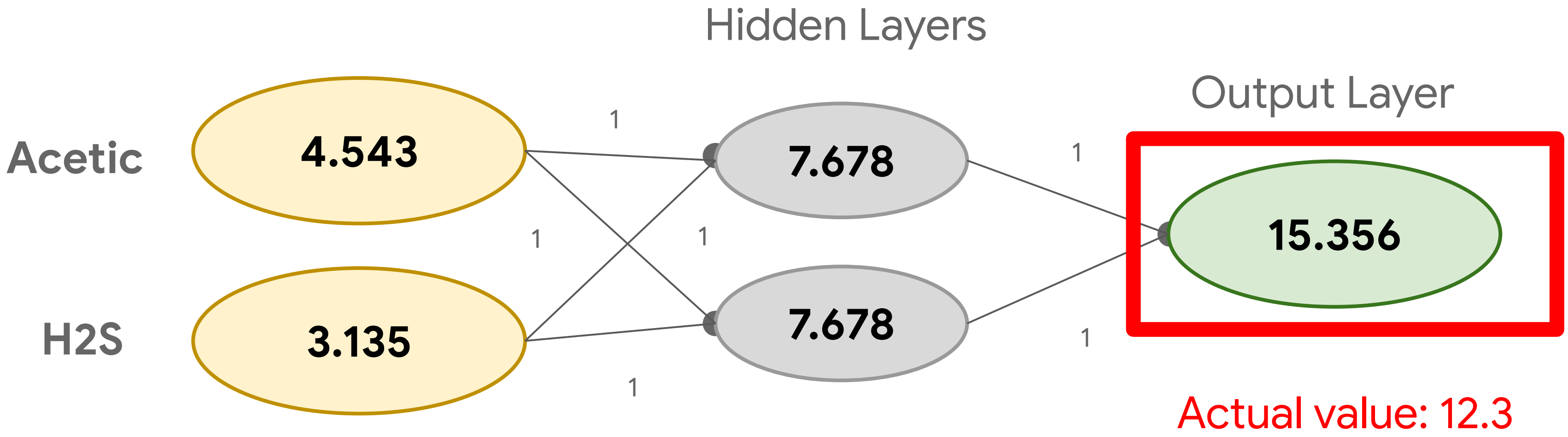


(4.543 x 1) + (3.135 x 1) = 7.678
(7.678 x 1) + (7.678 x 1) = 15.356

Code

```
import numpy as np

input_data = np.array([4.543, 3.135])
weights = {  ‘0’: np.array([1, 1]),
             ‘1’: np.array([1, 1]),
             ‘2’: np.array([1, 1])}

node_0 = (input_data) * weights[‘0’].sum()
node_1 = (input_data) * weights[‘1’].sum()

hidden_layer = np.array([node_0, node_1])
print(hidden_layer)

output = (hidden_layer * weights[‘2’]).sum()
print(output)
```

Let's try it: Forward Propagation

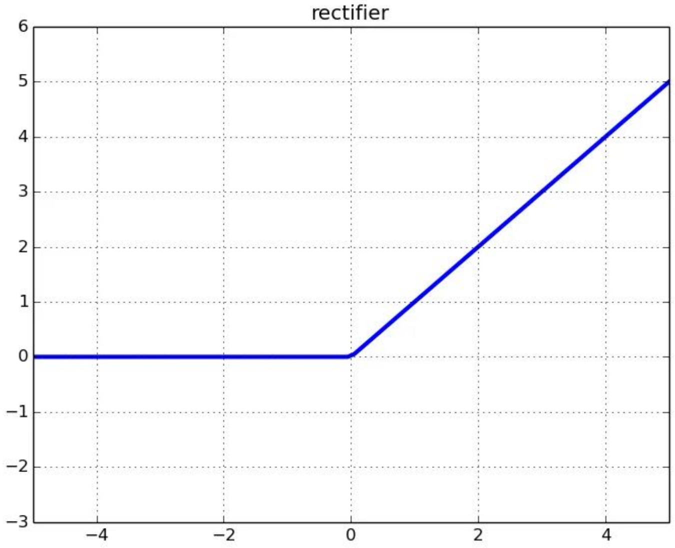


(4.543 x 1) + (3.135 x 1) = 7.678
(7.678 x 1) + (7.678 x 1) = 15.356

Activation functions: Improving models

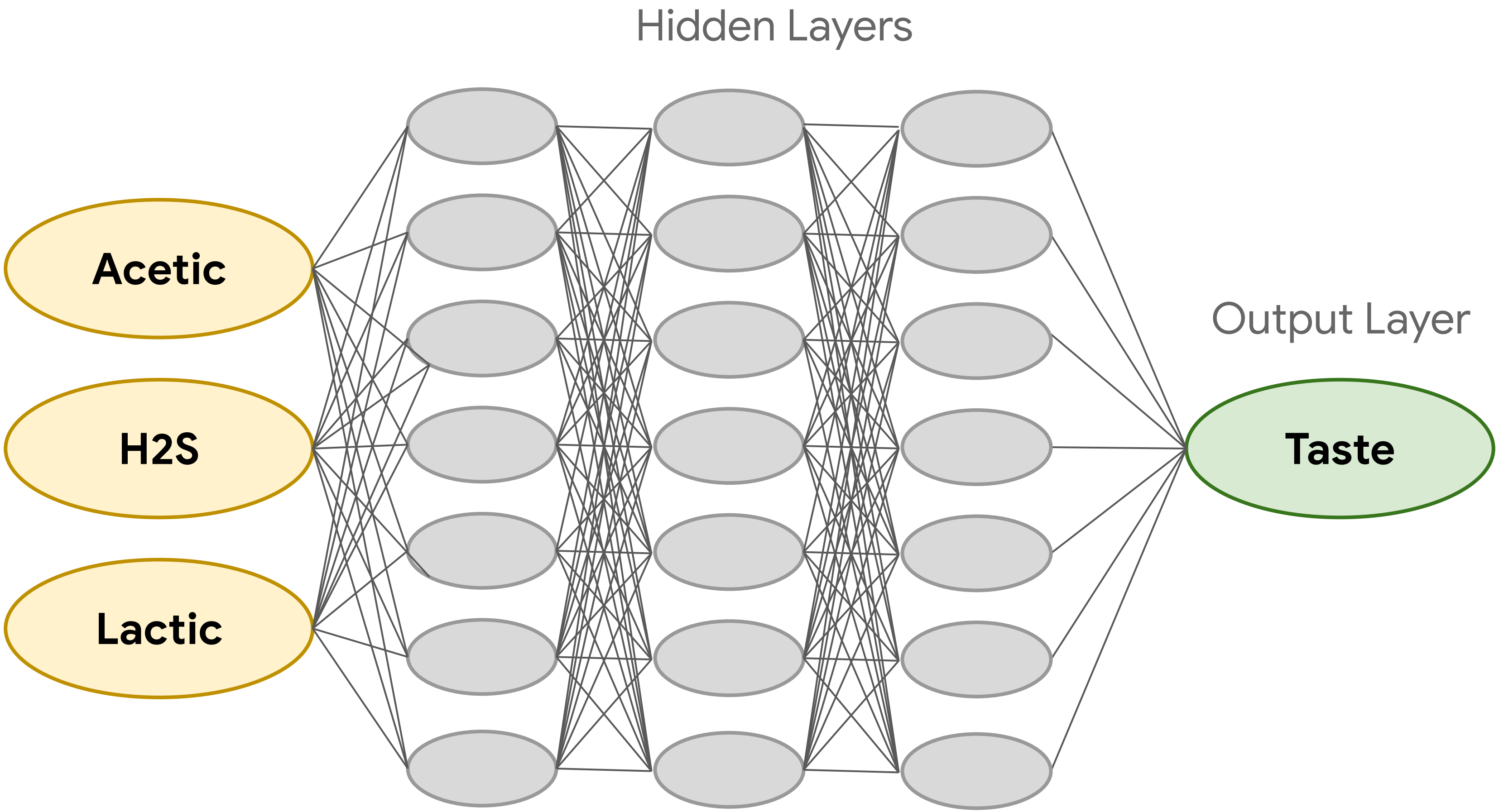
- Applied to node input to produce a node output
  `relu(4.543 * 1 + 3.135 * 1)`

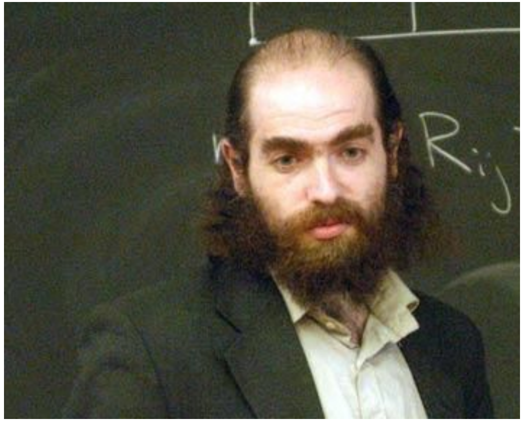
- Current industry standard is the *Rectified Linear Activation (ReLU)*

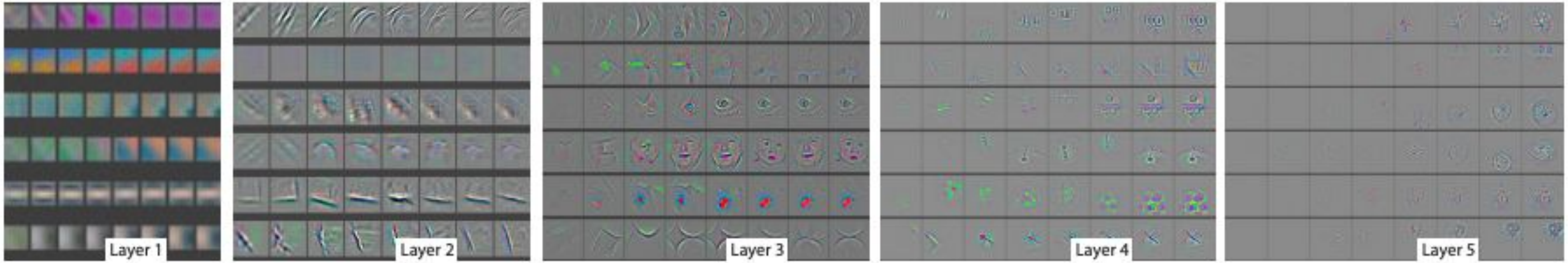


Where predicting gets tricky

- This a reduced-dimensionality example for a *single record*. The more features and records you add, this harder this process gets.
- At every set of weights, there are *many values* of the error.

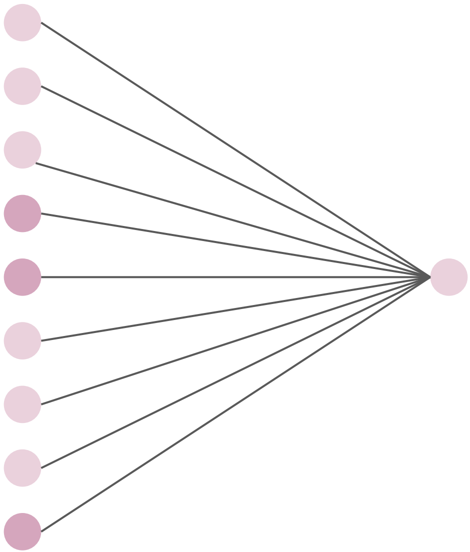

Hidden Layers
Acetic
H2S
Lactic
Output Layer
Taste





zeiler and fergus (2013)

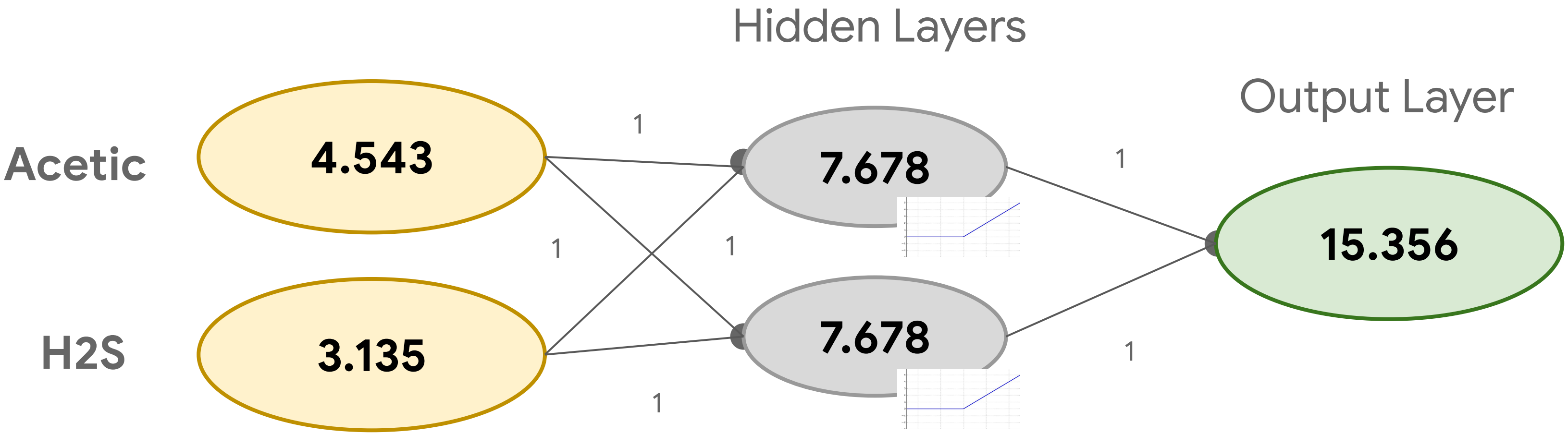
For every single node:



$$= \sigma(w_1 a_1 + w_2 a_2 + \ldots + w_n a_n + b)$$

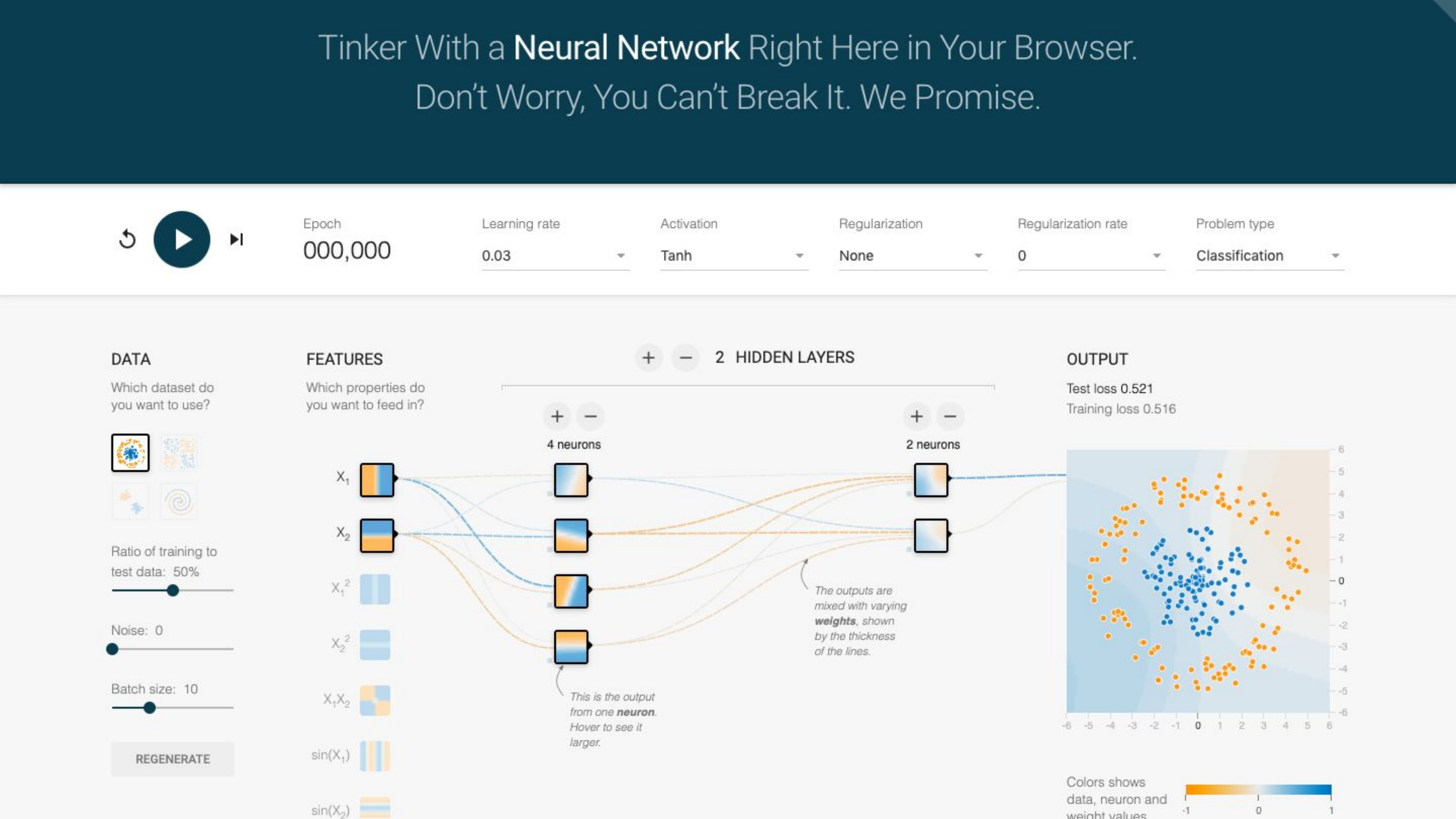
σ = activation function
w = weights
b = bias

Let's try it: Forward Propagation



(4.543 x 1) + **relu**(3.135 x 1) = 7.678

(7.678 x 1) + **relu**(7.678 x 1) = 15.356


Tinker With a Neural Network Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.
Epoch
000,000
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
2 HIDDEN LAYERS
4 neurons
2 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.521
Training loss 0.516
Colors shows data, neuron and weight values.

Resources

- A Student’s Guide to Maxwell’s Equations
- Visualizing and Understanding Convolutional Networks (Zeiler and Fergus, 2013)
- TensorFlow Playground

Thank you!