Applied Deep Learning

Lecture 1

Today’s Agenda

- About the class
- Recent results in DL
- AI vs. ML vs. DL
- Feature engineering vs. Representation learning

About the class

What is Applied Deep Learning?

Basically, an intro to DL with less theory, more practice, and more best practices (many of these are pretty straightforward, but only obvious in retrospect).

Topics

- Development
- Evaluation
- Testing
- Fairness

Instructor

Josh Gordon (joshua@cs.columbia.edu)

There are no silly questions (either about TensorFlow or Deep Learning).

- DL is a wide and rapidly developing field. It's normal for neural networks to take a while to understand. No one knows everything (including the amazing researchers I'm fortunate to work with). The people I admire most are constantly improving their understanding.

TAs

- Prerna Kashyap (pk2600@columbia.edu)
- Terence Conlon (tmc2180@columbia.edu)
- Pratik Dubal (pratik.dubal@columbia.edu)
- More soon!

Course website

bit.ly/applied-dl

- Contains the syllabus and assignments.
- Slides will be uploaded to Piazza before class. Eventually to GitHub as well, after we get permission to use all the images. ¯_(ツ)_/¯
- Reading and assignment reminders will always be on the **last slide.**

Grading

Five programming assignments (60%)

- All are practical (implement an image classifier in TensorFlow, deploy a model in the browser using TensorFlow.js).

Project and presentation (25%)

- Choose to do either the class project , or propose your own.

Midterm (15%)

- Conceptual, short answer questions.

You have two or three weeks to complete each assignment

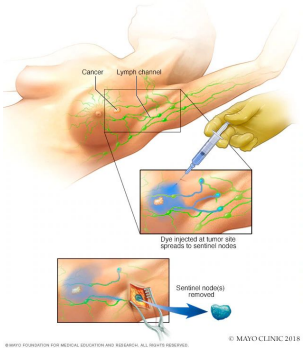
- **Not because they're expected to take long!**
- Just so you can plan your time as is best (I know a lot of folks work, have heavy course loads, etc).
- On the flip side, grades won't be posted until about a week after the due date.

Course project

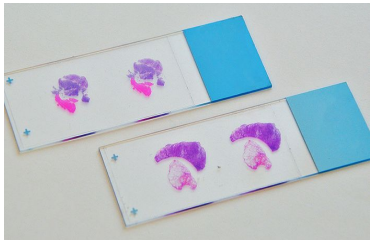
Opportunity for you to focus on an **area of interest**.

- This can be valuable to bring up in a **future interview** (having something you can present confidently is a great outcome).

- Or to **inform future study**.



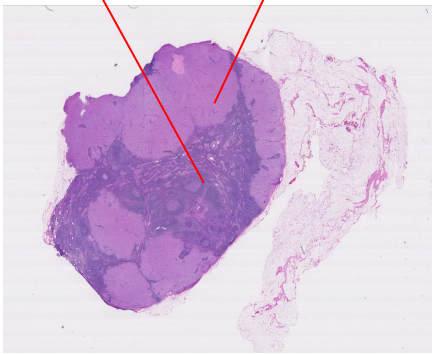
biopsy



Preparation

Diagnosis → Treatment plan

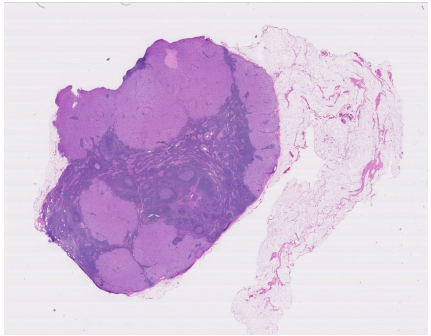
healthy vs tumor



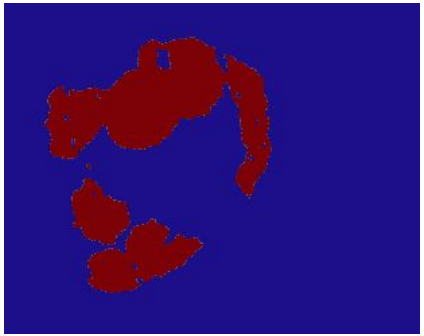
Visual inspection

[Sentinel node biopsy](), [CAMELYON16 and CAMELYON17 Challenges]()

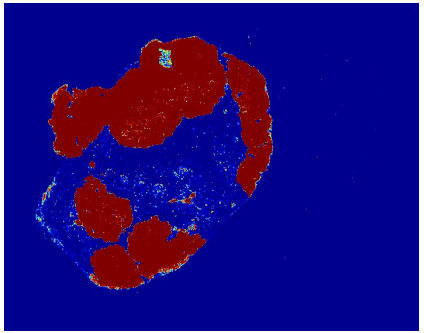
Develop a tool to assist physicians



Biopsy image



Ground truth
(from pathologist)



Your model's
predictions

Please bring your laptops to class

We have a long time slot **7pm - 9:30pm**.

- We will use time in class to work on the homework, pretty much every week except today.

- This will also give you a chance to meet other students, hopefully you can help each other debug / chat through concepts.

Difficulty level

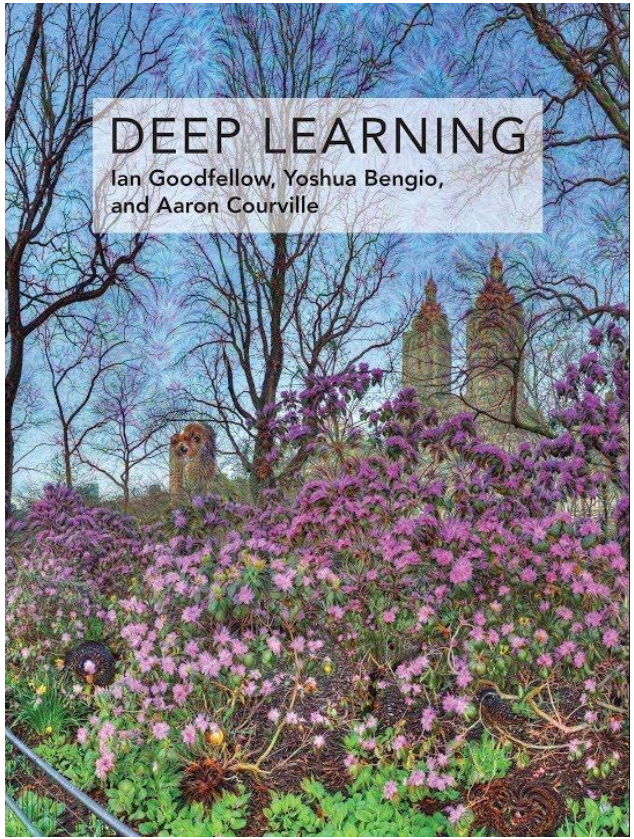
- If you find this content tricky, happy to help!
- If you find it a bit too easy, happy to help brainstorm a challenging project, so you can stretch yourself.

Textbook #1

Complete text is available for free online

www.deeplearningbook.org

Ian Goodfellow is the creator of GANs - one of the most interesting ideas in computer science today.



Generative Adversarial Networks

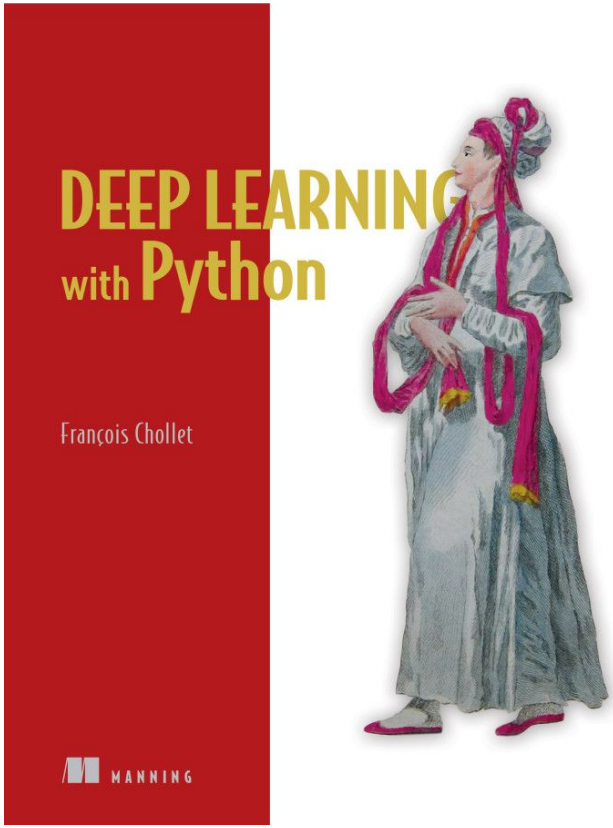
Textbook #2

This is most **effective** hands-on intro that exists today. It's light reading, and full of practical advice.

Please purchase this book (e-book is around $40). manning.com/books/deep-learning-with-python

- Code is available for free online: github.com/fchollet/deep-learning-with-python-notebooks

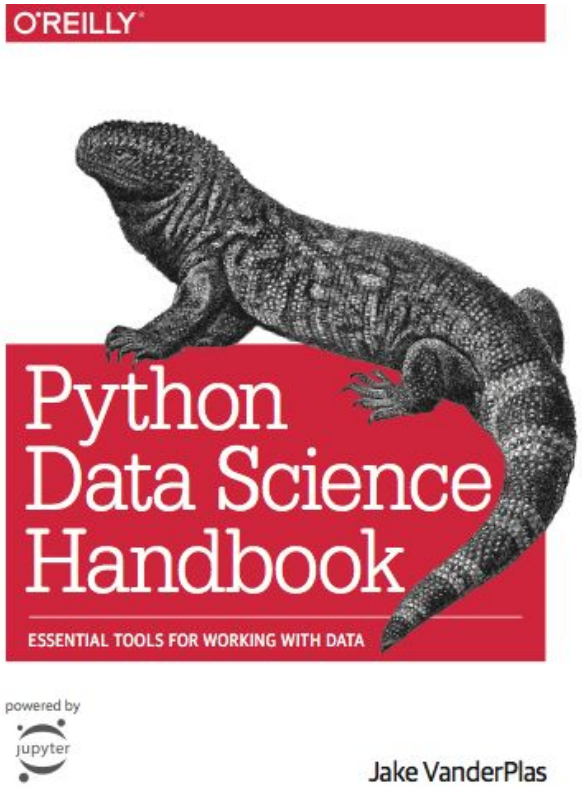
Francois Chollet is the author of Keras, a library famous for being user friendly. TensorFlow is standardizing on this design.



A helpful reference

Complete text is available for free online.

- Covers essential Python libraries including NumPy and Matplotlib.



~~Not built here~~

Additional high-quality learning resources

- The course notes from cs231n are the **best in the business**.
- MIT has a recent DL course (fast paced), they're updating it now.
- Andrew Ng's Coursera course.

Advice: study the same material in two different textbooks, for two different views on the same subject, and a better chance of it "clicking".

Career tips

Separately, happy to chat with you during office hours if you'd like advice on:

- Interviewing, and/or simple career tips (like how to negotiate a salary) -- things I wish I had learned in school, but were never mentioned.

- Perspective on the difference between joining a startup or large company, or government jobs vs private sector.

Development environment

We'll use **Colaboratory**: colab.research.google.com

- A Jupyter environment that runs in the Cloud.
- Zero setup, and a free GPU.
- Download any Colab notebook as a Jupyter notebook.
- Upload any Jupyter notebook to Colab.

Install any software you need using: `!pip install [package]`.

Quick demo

- Running “Hello World” in Colab
- Uploading / downloading Jupyter notebooks
- Using the GPU
- Code snippets
- File browser
- Running shell commands
- Resetting the environment

Another quick demo

Two major styles used by most DL libraries today

Symbolic and imperative models

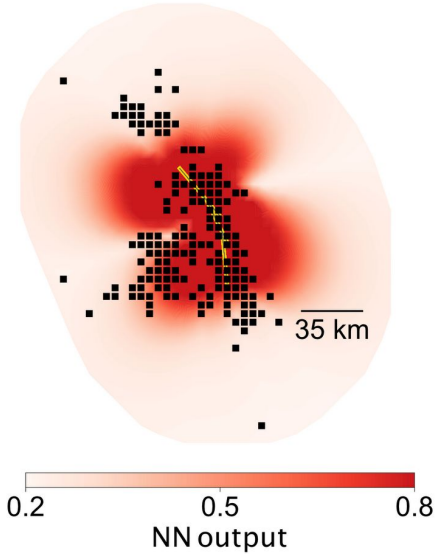
- You'll use both in this course, want to give you a quick preview.
- In a symbolic style, your model is a DAG.
- In an imperative style, your model is bytecode.

Lots of interesting tradeoffs, details aren't important for now.

A few recent results

Supporting basic science

Supporting basic science

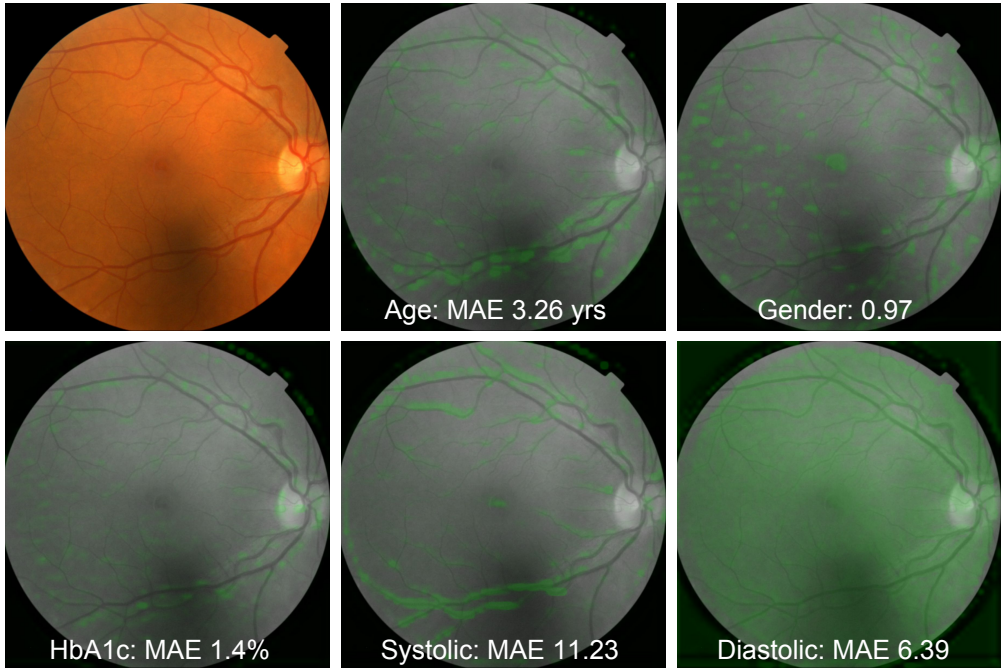


Forecasted distribution of aftershock location probabilities for the Landers earthquake.

- Dark red colors indicate regions predicted to experience aftershocks.
- The black dots are the locations of observed aftershocks, and the yellow line shows the faults that ruptured during the mainshock.

Forecasting earthquake aftershock locations with AI-assisted science

Supporting basic science



Predicting things that doctors can't predict from imaging

Potential as a new biomarker

Can we predict cardiovascular risk? If so, this is a potentially non-invasive way of doing so.

Predicting Cardiovascular Risk Factors from Retinal Fundus Photographs using Deep Learning.

Research direction

Explore the representations the network is using to predict aftershocks and cardiovascular risk.

Which are informative, and why?

Beyond Python

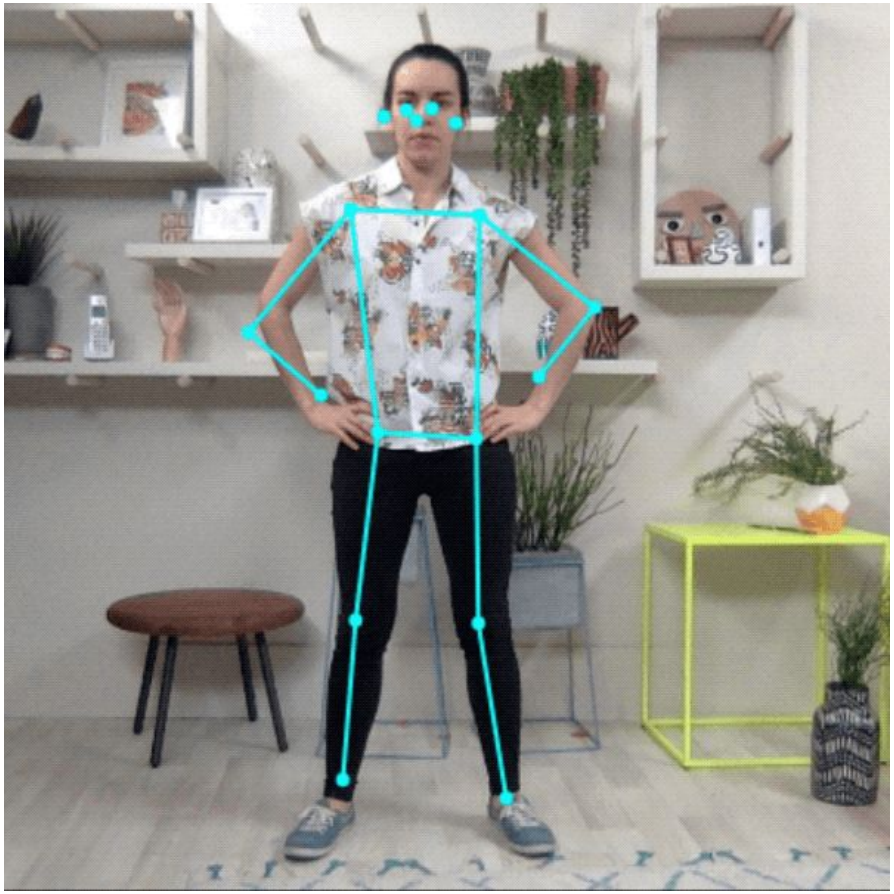
Machine Learning

happens in Python,

right?

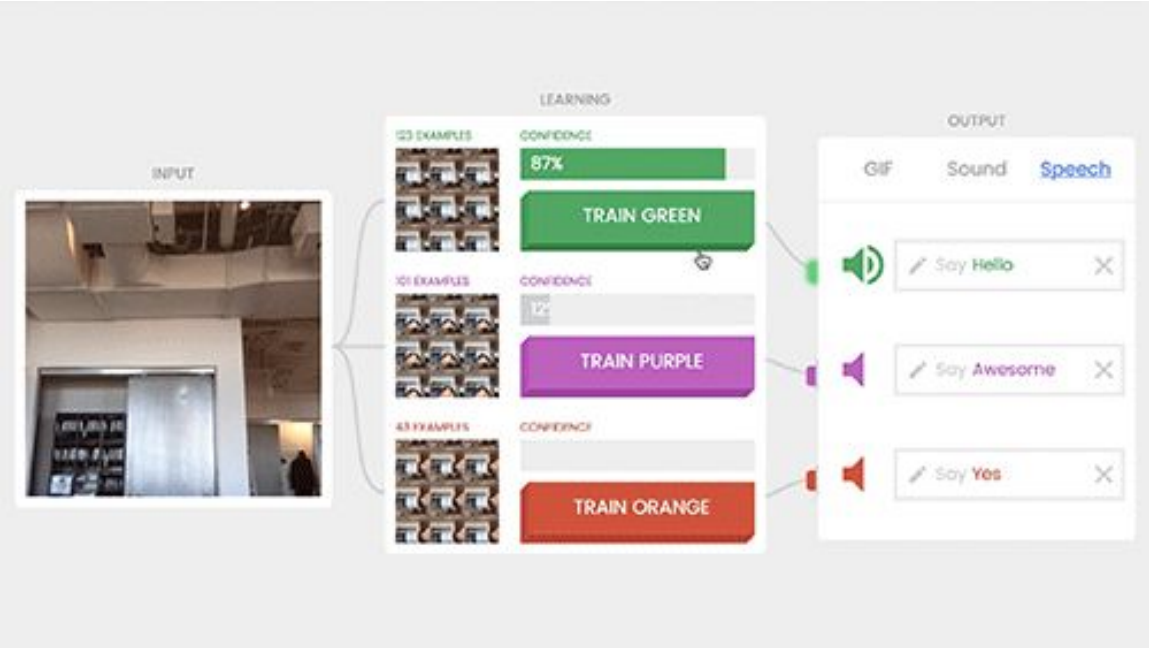
5/6 of the assignments are in Python (as is the course project).

- **One involves a bit of JavaScript.**
- **Surprise! It's quickly become essential to learn (at least the basics).**



Real-time Human Pose Estimation in the Browser with TensorFlow.js, Demo

Teachable Machine



Link

Quick discussion: Why is Python so popular for ML?

Quick discussion: DL in Javascript has been ***wildly*** successful (why?).

- Swift has also added 1st class support for automatic differentiation, and R now supports Keras. Swift for TensorFlow is in beta!


Automatic Differentiation in Swift

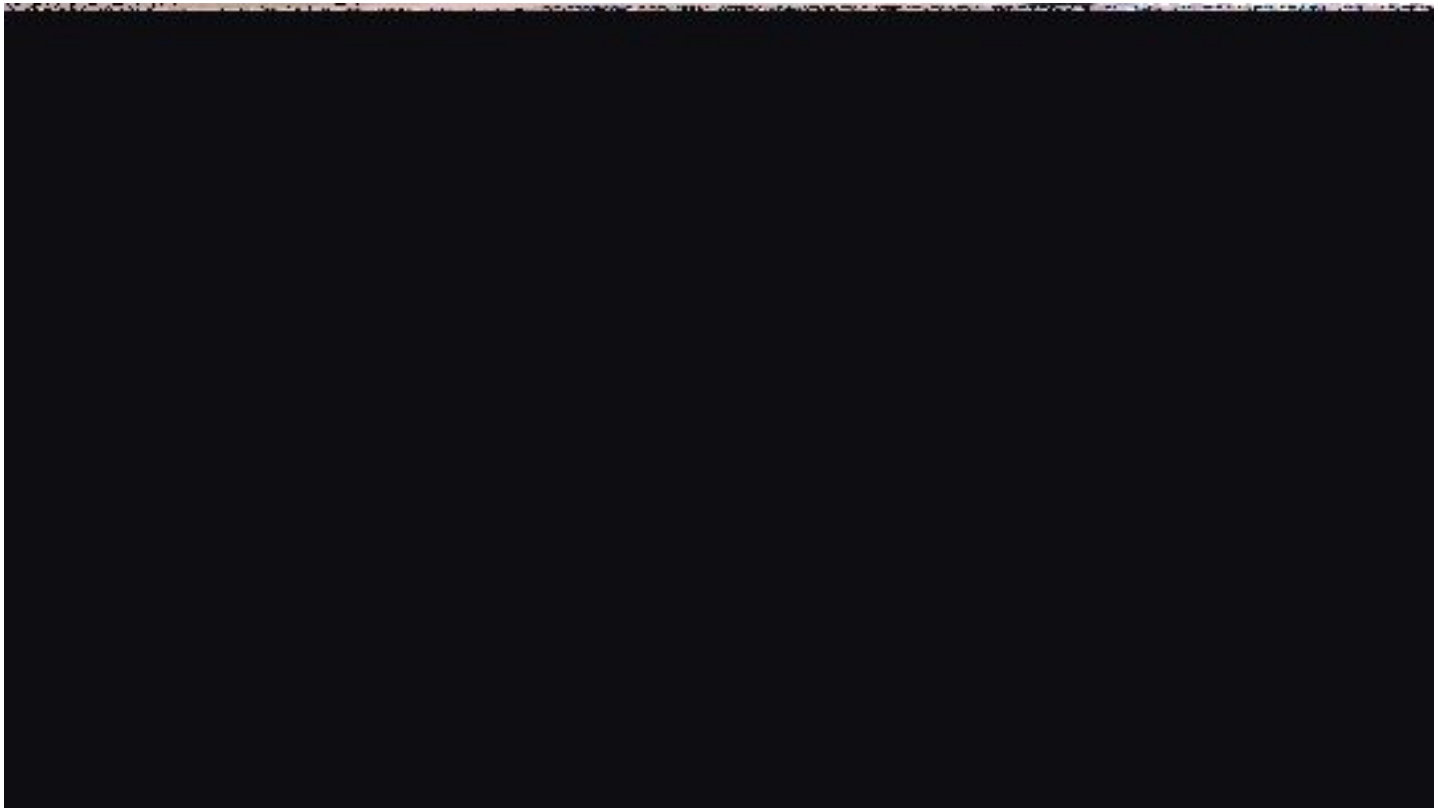
Sketch-RNN: draw together with a NN

Sketch-RNN, Demo

Meaningful (and fun!) applications

Getting Alexa to Respond to Sign Language Using Your Webcam and TensorFlow.js, by Abhishek Singh

TensorFlow.js



Getting Alexa to Respond to Sign Language Using Your Webcam and TensorFlow.js, by Abhishek Singh

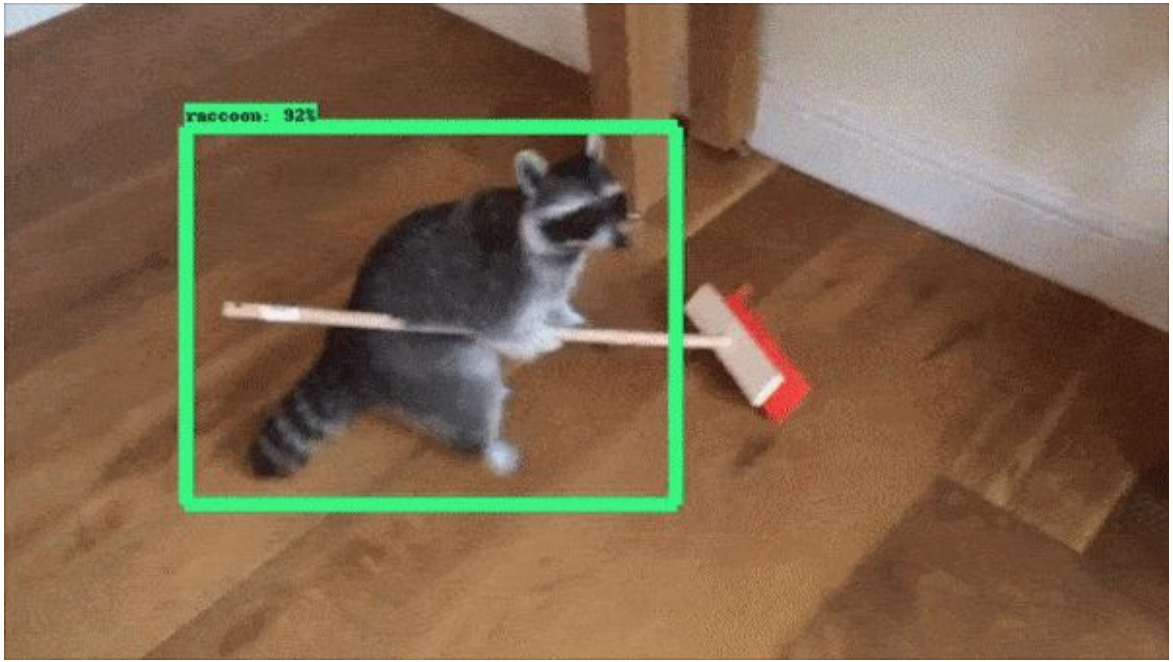
TensorFlow.js

Mobile

Sensors

Machine Learning in JavaScript (TensorFlow Dev Summit 2018)

Article. (Note: skip the paid services, you can do this entirely in open source).



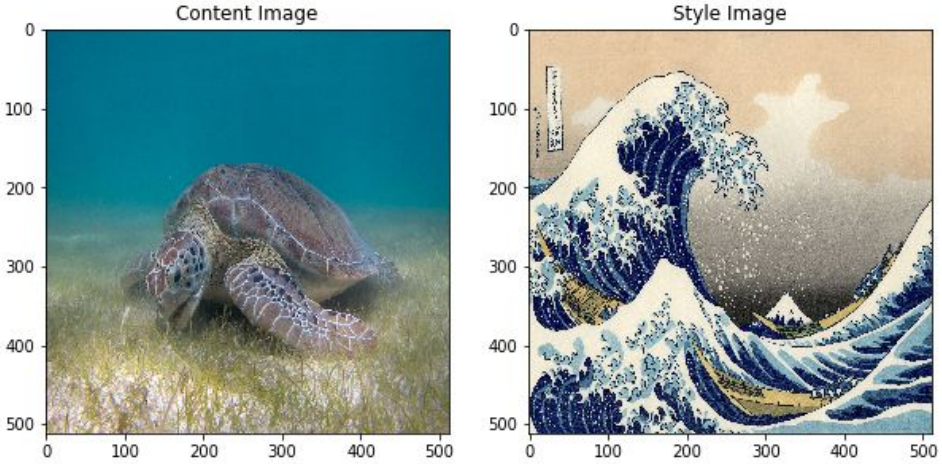
github.com/tensorflow/models/tree/master/research/object_detection

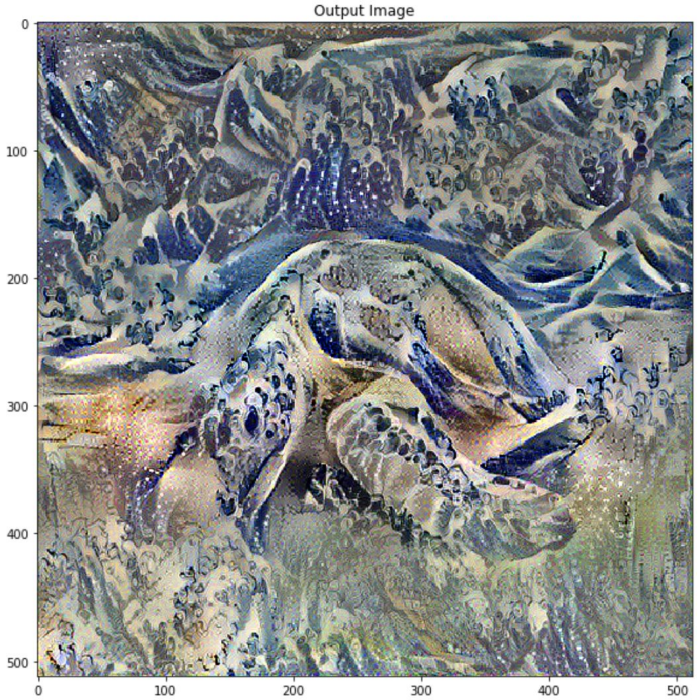
Silly, but profound

Deep learning is representation learning.

Investigating the representations learned by a network trained to classify images led to deep dream.


HOMER · HERODOTUS · SOPHOCLES · PLATO · ARISTOTLE · DEMOSTHENES · CICERO · VERGIL





Possible as an **artifact(!)** of training an image classifier.

[A Neural Algorithm of Artistic Style]
[TensorFlow implementation]

Text and Translation

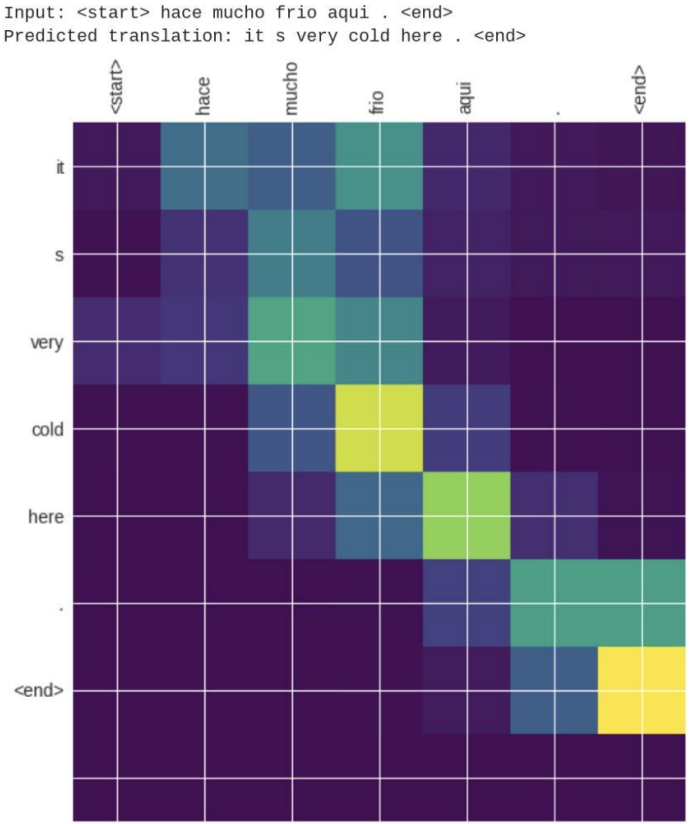
Generating (hilariously wrong) paint colors

Clardic Fug 112 113 84
Snowbonk 201 199 165
Catbabel 97 93 68
Bunflow 190 174 155
Ronching Blue 121 114 125
Bank Butt 221 196 199
Caring Tan 171 166 170
Stargoon 233 191 141
Sink 176 138 110
Stummy Beige 216 200 185
Dorkwood 61 63 66
Flower 178 184 196

Sand Dan 201 172 143
Grade Bat 48 94 83
Light Of Blast 175 150 147
Grass Bat 176 99 108
Sindis Poop 204 205 194
Dope 219 209 179
Testing 156 101 106
Stoner Blue 152 165 159
Burble Simp 226 181 132
Stanky Bean 197 162 171
Turdly 190 164 116

An AI invented a bunch of new paint colors that are hilariously wrong.

End to end machine translation, learned **directly** from parallel corpora (without feature engineering)



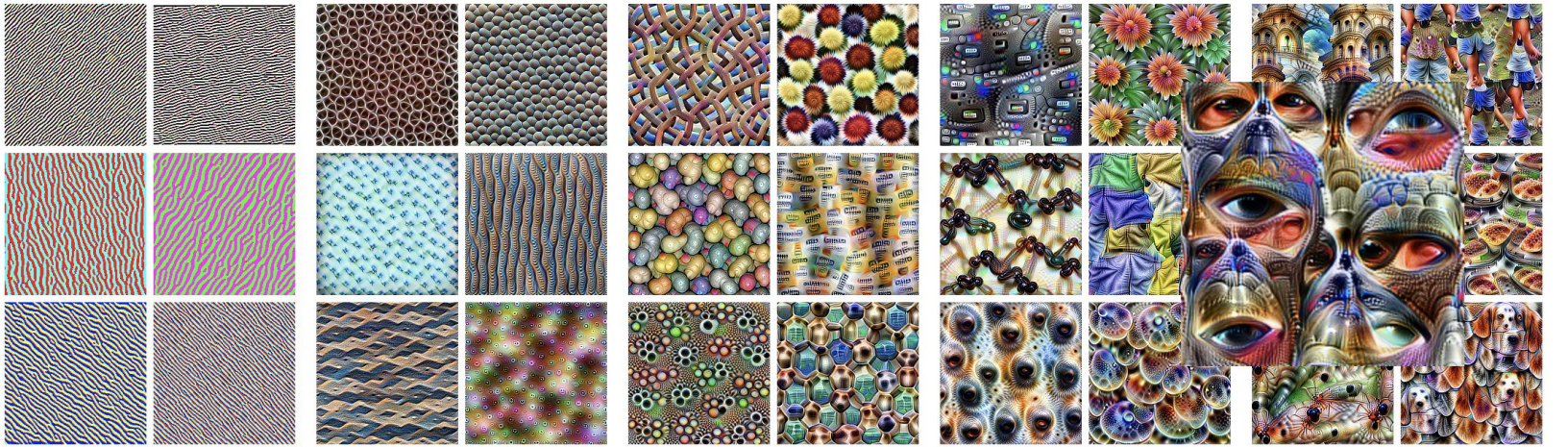
Complete code examples

Mixed data

Complete code examples

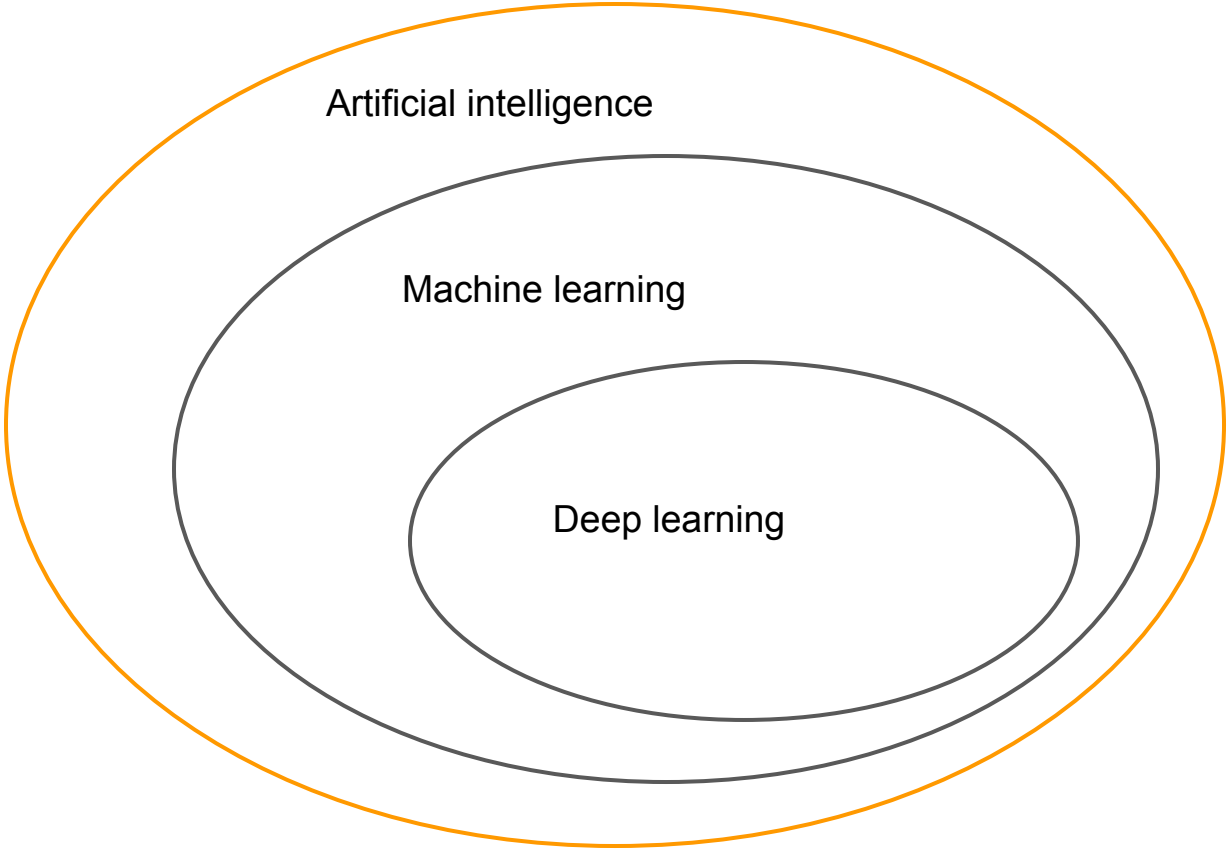
How Deep Learning fits into the ML picture

The key idea that distinguishes deep learning from other types of ML is **representation learning**.

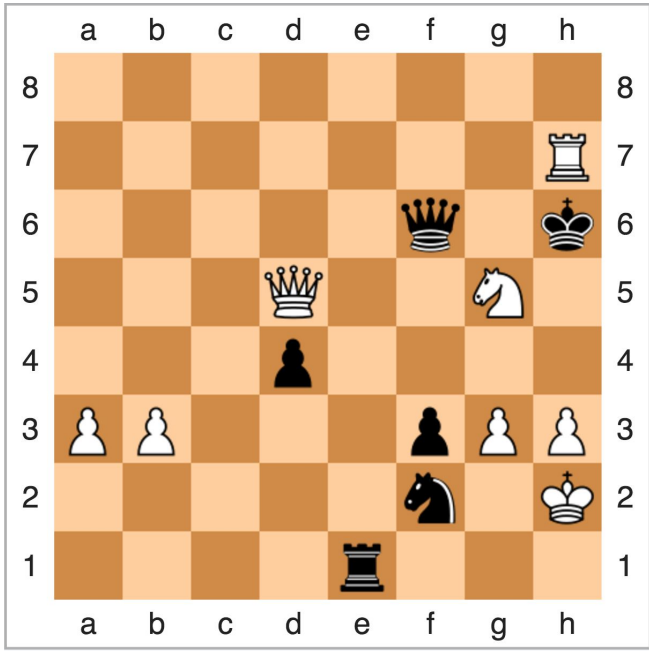


We'll come back to this. Let's start with the early days of AI. Problems that are **difficult for people**, but **easy for computers.**

Feature Visualization from distill.pub

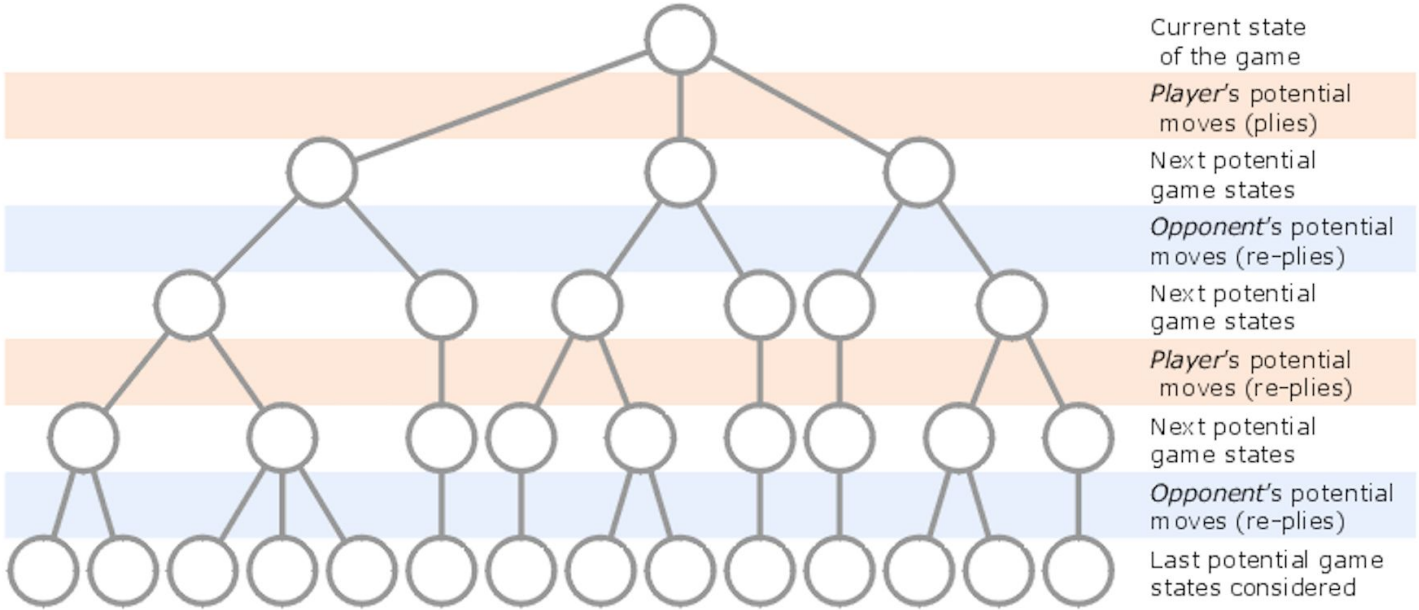

Artificial intelligence
Machine learning
Deep learning

IBM’s Deep Blue vs Garry Kasparov (1997)



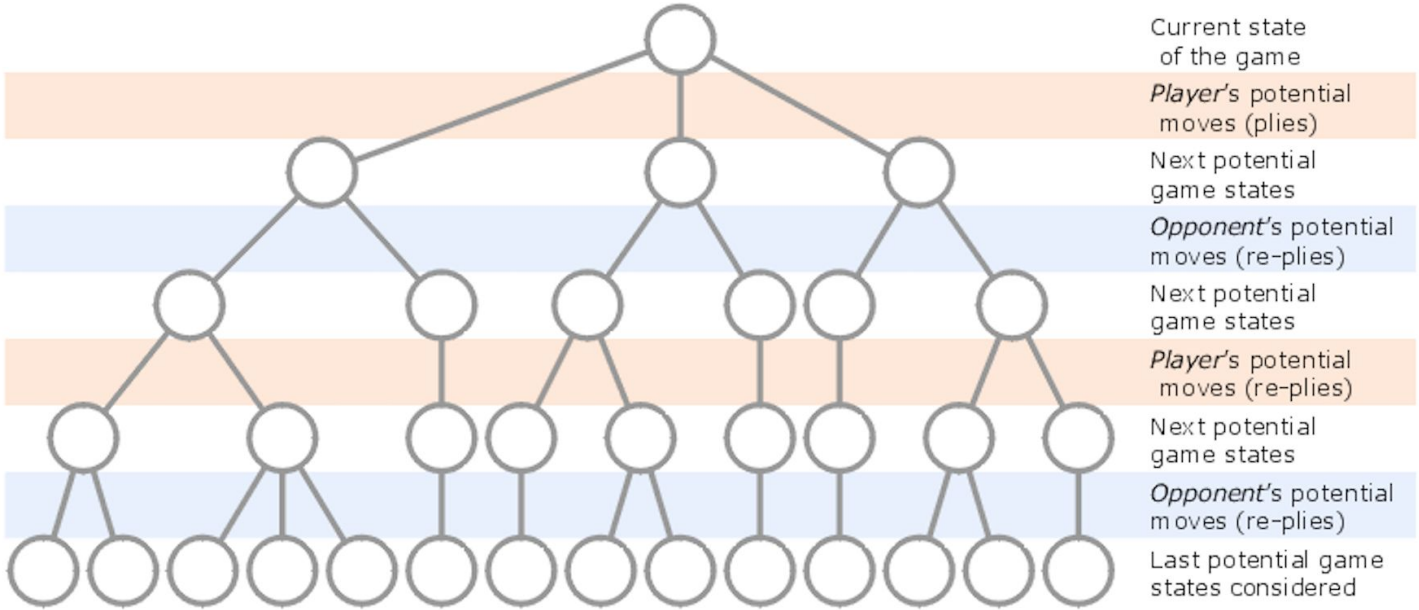
The story of Deep Blue

Deep Blue used [algorithm] by X in 19--. Does anyone know?



Programming a Computer for Playing Chess

Claude Shannon 1950(!)



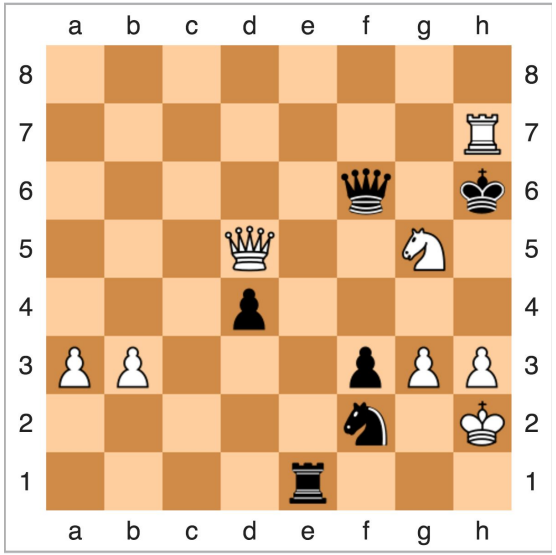
Programming a Computer for Playing Chess

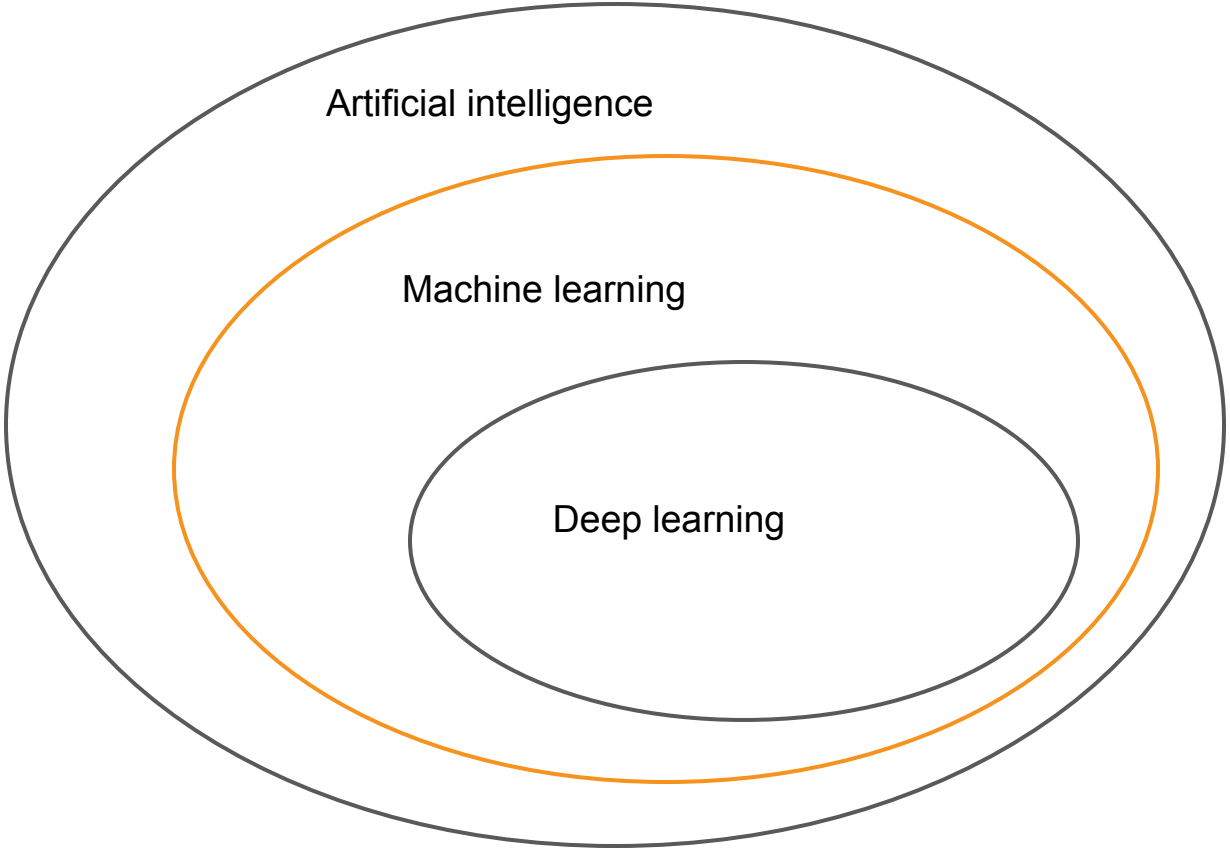
A tremendous accomplishment, but not due to the difficulty of the representation.

The rules of the game can be completely described by a programmer.

- 64 locations on a chessboard.
- 32 pieces that move in specific ways.
- List of formal rules for the game state.

Deep Blue won via compute, and a **hand-tuned board evaluation function.** Think: board --> score [0,1].




Artificial intelligence
Machine learning
Deep learning

Machine Learning

- Many models you may be familiar with: kNN, Decision Trees, Logistic Regression, SVMs, Random Forests, etc.

- Learning usually means finding useful values for parameters by updating them based on a collection of data.

Naming, marketing, and hype.

- Most Kaggle competitions involving **perception** are won by **[???]**.
- Most Kaggle competitions involving **structured data** are won by **[???]**.

- Most Kaggle competitions involving **perception** are won by **tree-based models.**

- Most Kaggle competitions involving **structured data** are won by **DL-based models.**

Quick discussion: What's the key advantage of tree-based models?

- Most Kaggle competitions involving **perception** are won by **tree-based models.**

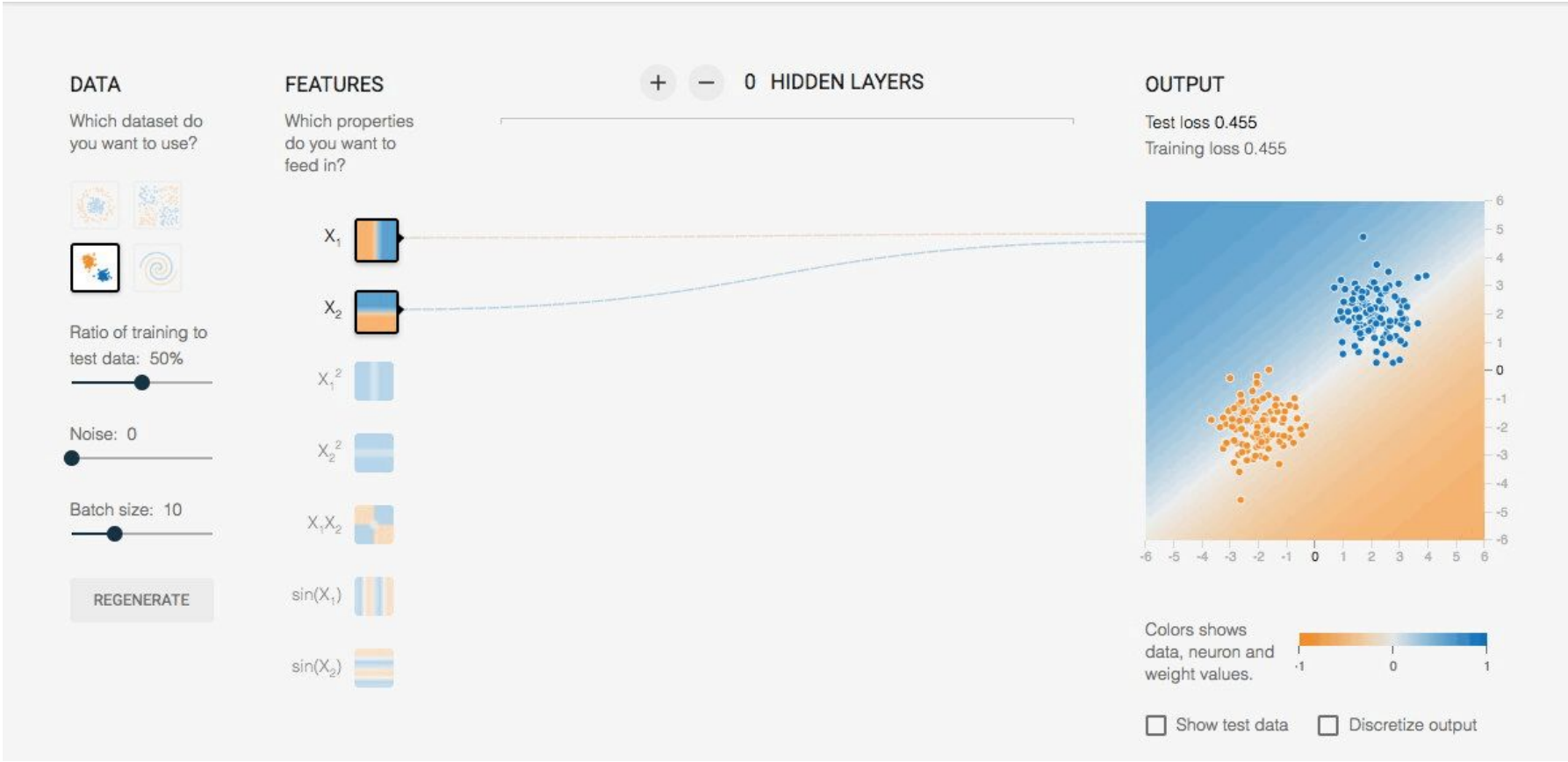
- Most Kaggle competitions involving **structured data** are won by **DL-based models.**

Quick discussion: What's the key advantage of tree-based models?

- You can explain your predictions.

- We can do this with DL, too (particularly when working with images) though less precisely.

A linear model



TensorFlow Playground (note: since this was released, we now have TensorFlow.js, a proper web-based NN implementation)

Linear models

Determine whether to recommend cesarean delivery using logistic regression.

- Doctor tells the system several pieces of relevant information **(features).**
- Model **adjusts** (**a weight, or parameter**) to indicate their importance to the outcome.

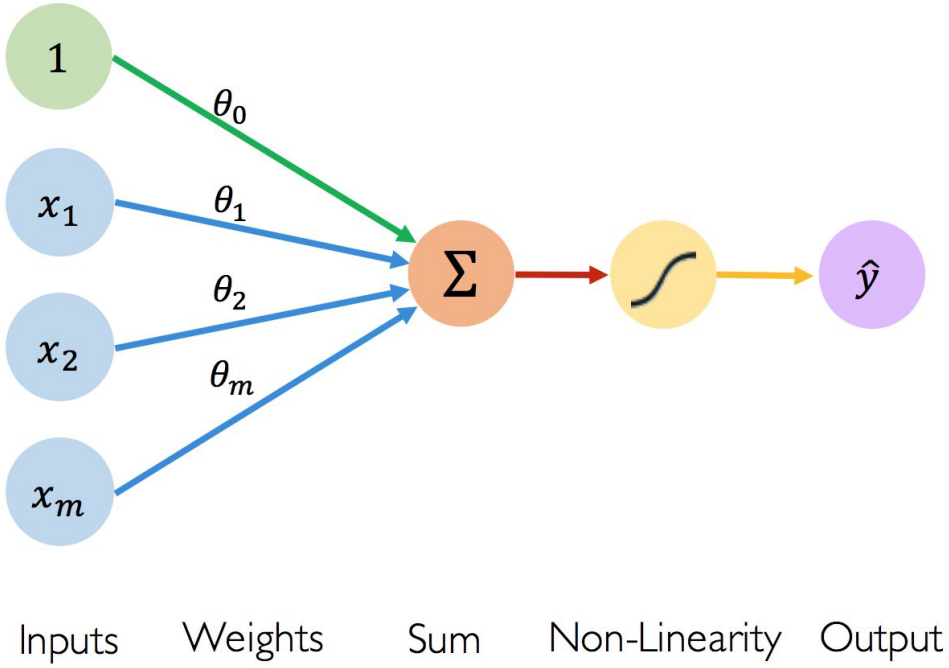
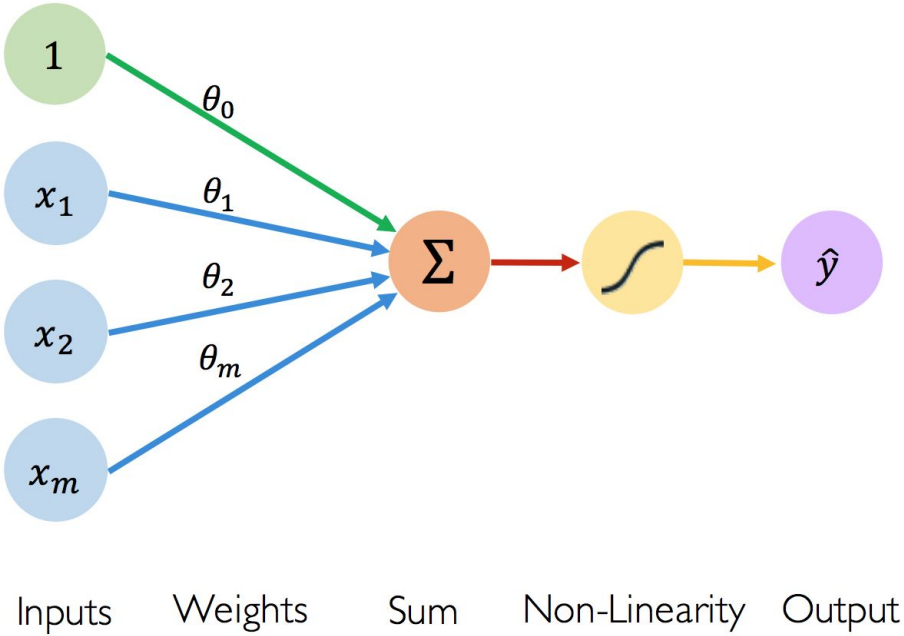


Diagram from MIT's course, linked earlier in these slides.



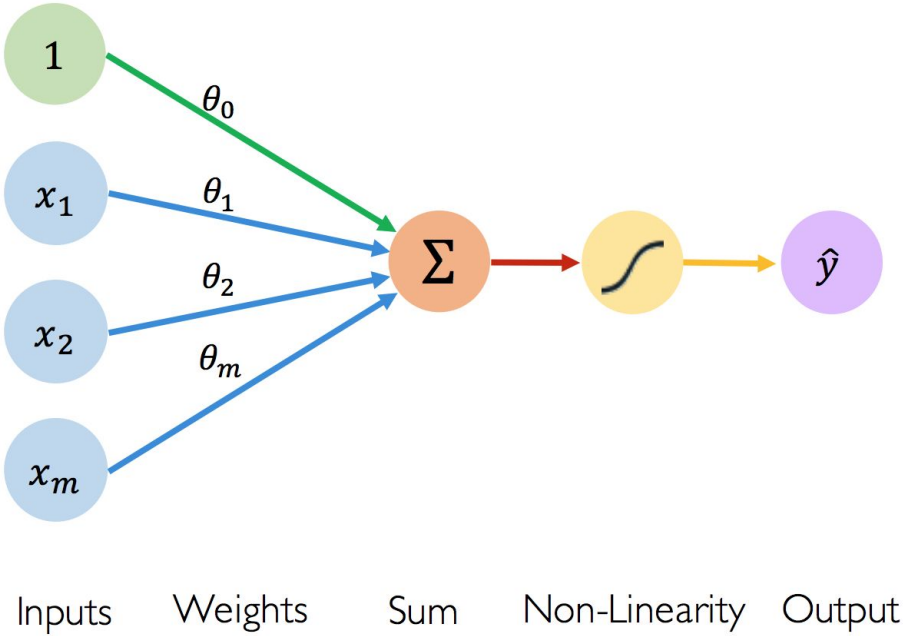
Linear combination of inputs

Output

$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \, \theta_i\right)$$

Non-linear activation function

Bias

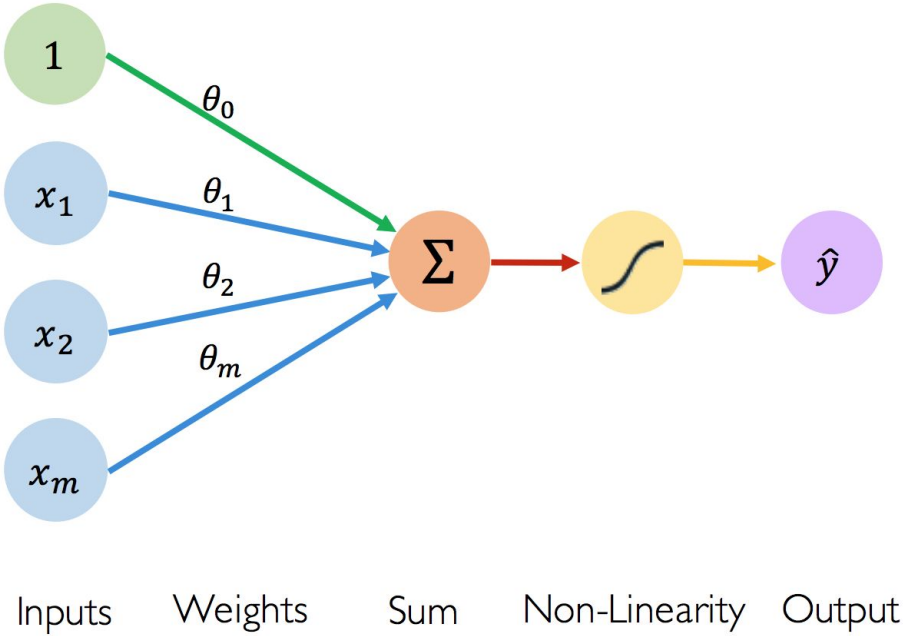


$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i\,\theta_i\right)$$

$$\hat{y} = g\left(\theta_0 + \boldsymbol{X}^T\boldsymbol{\theta}\right)$$

$$\text{where: } \boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

Can rewrite as a matrix-multiply if you have many of these logistic units. This will become important later, as that can be accelerated by a GPU.



$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

None

- Linear score function.

Threshold

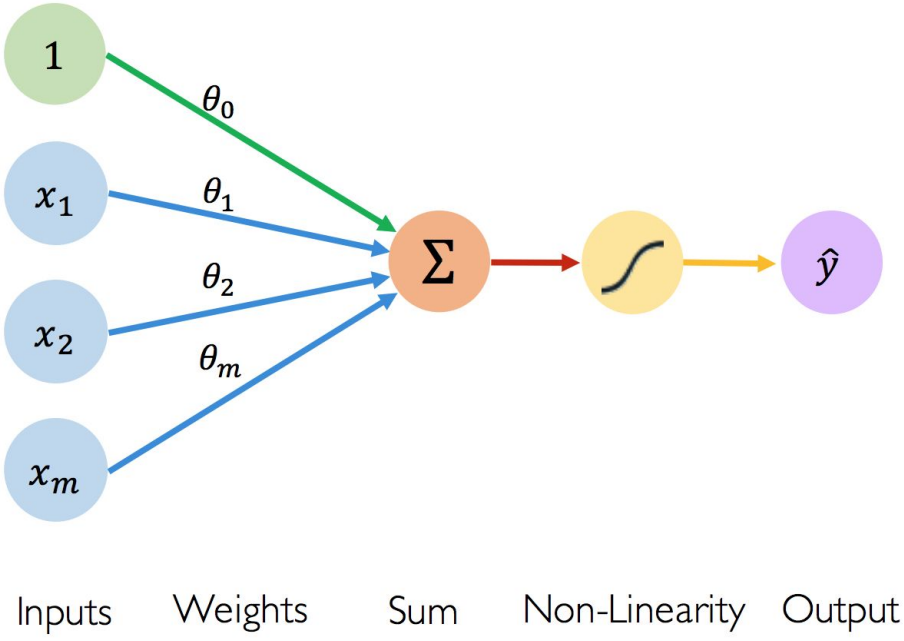
- Perceptron.

Sigmoid

- Logistic regression, or a logistic “neuron”.

ReLU:

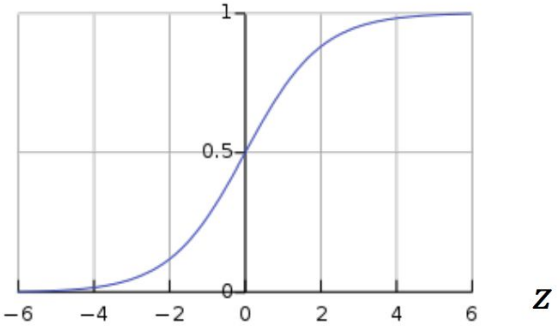
- A standard neuron today.



$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



When is this type of model effective?

Quick discussion

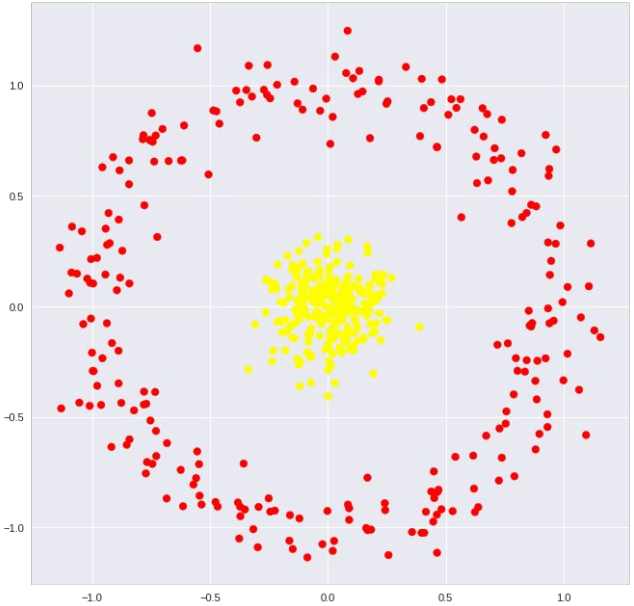
When is this type of model effective?

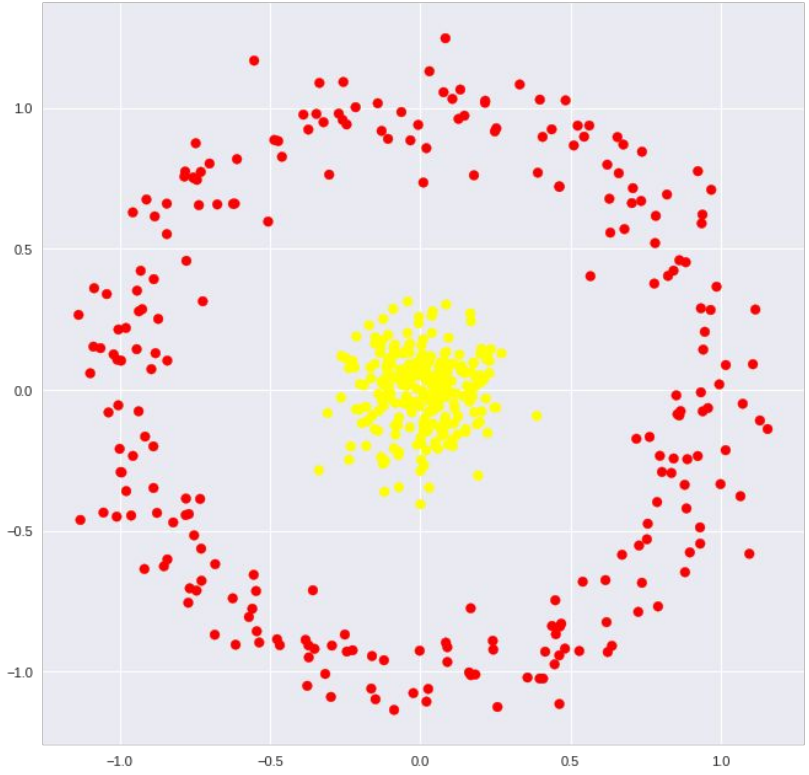
Quick discussion

- You’re working with structured data (a table)
- You have many simple features (demographic info)
- You’re willing to do feature engineering (feature crosses, bucketing)

Feature engineering

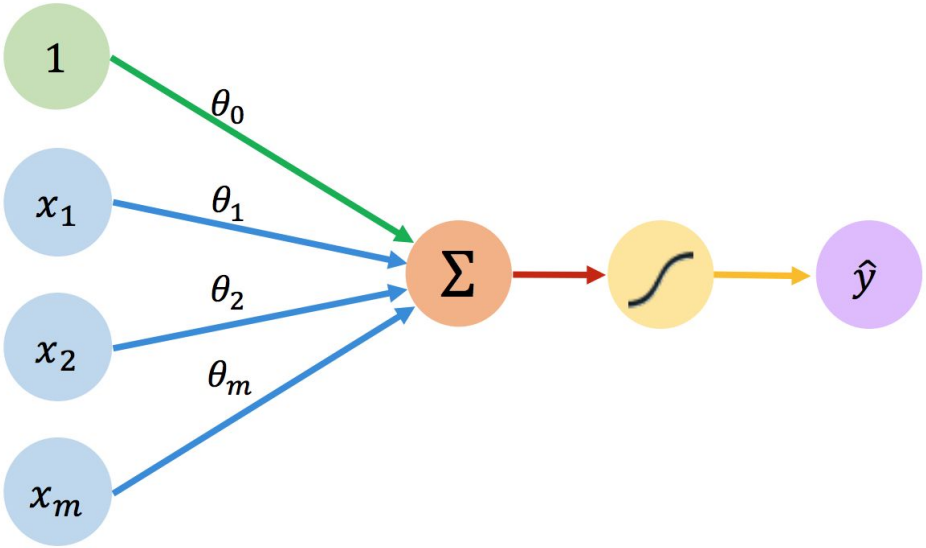
Can we classify this dataset with a linear model?






```
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import \
   make_circles

plt.figure(figsize=(10,10), dpi=80)
X, y = make_circles(500, factor=.1, noise=.1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='autumn')
```

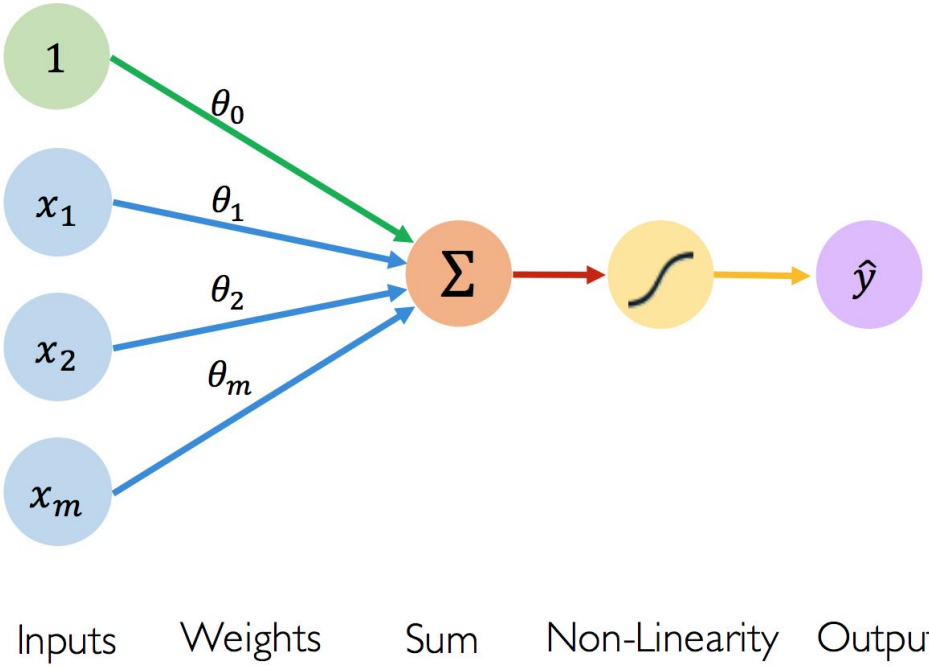


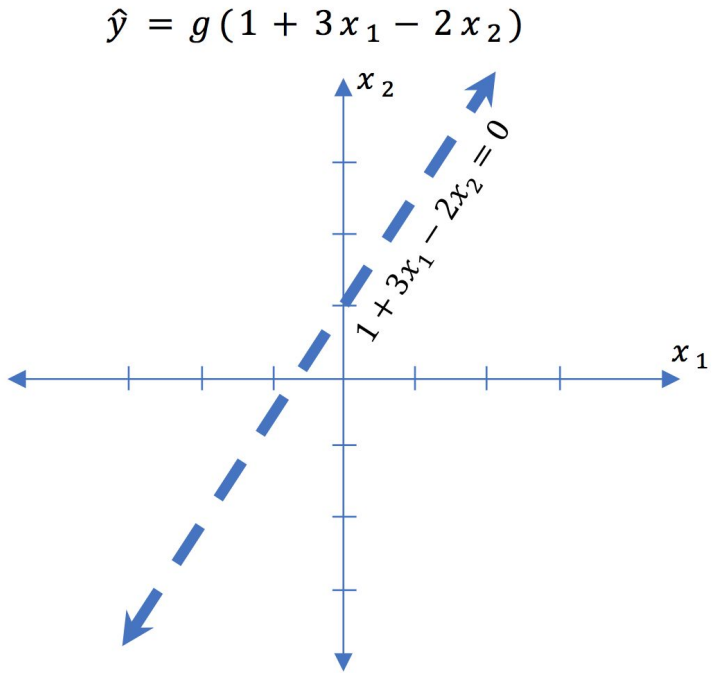
We have: $\theta_0 = 1$ and $\boldsymbol{\theta} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

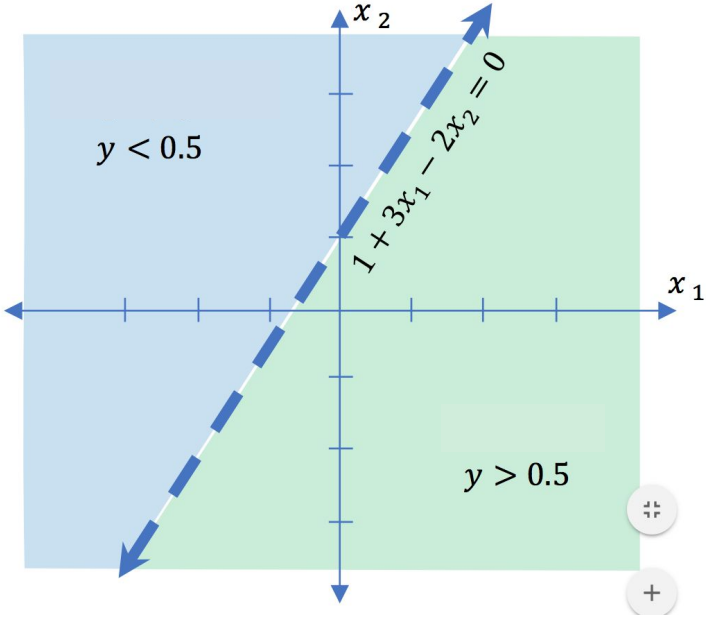
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

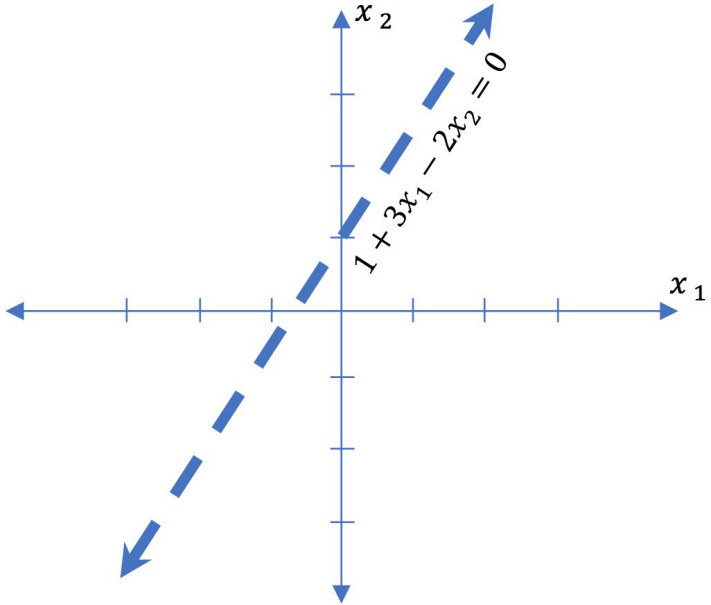

1
x_1
x_2
x_m
θ_0
θ_1
θ_2
θ_m
Σ
$\hat{y}$
Inputs
Weights
Sum
Non-Linearity
Output

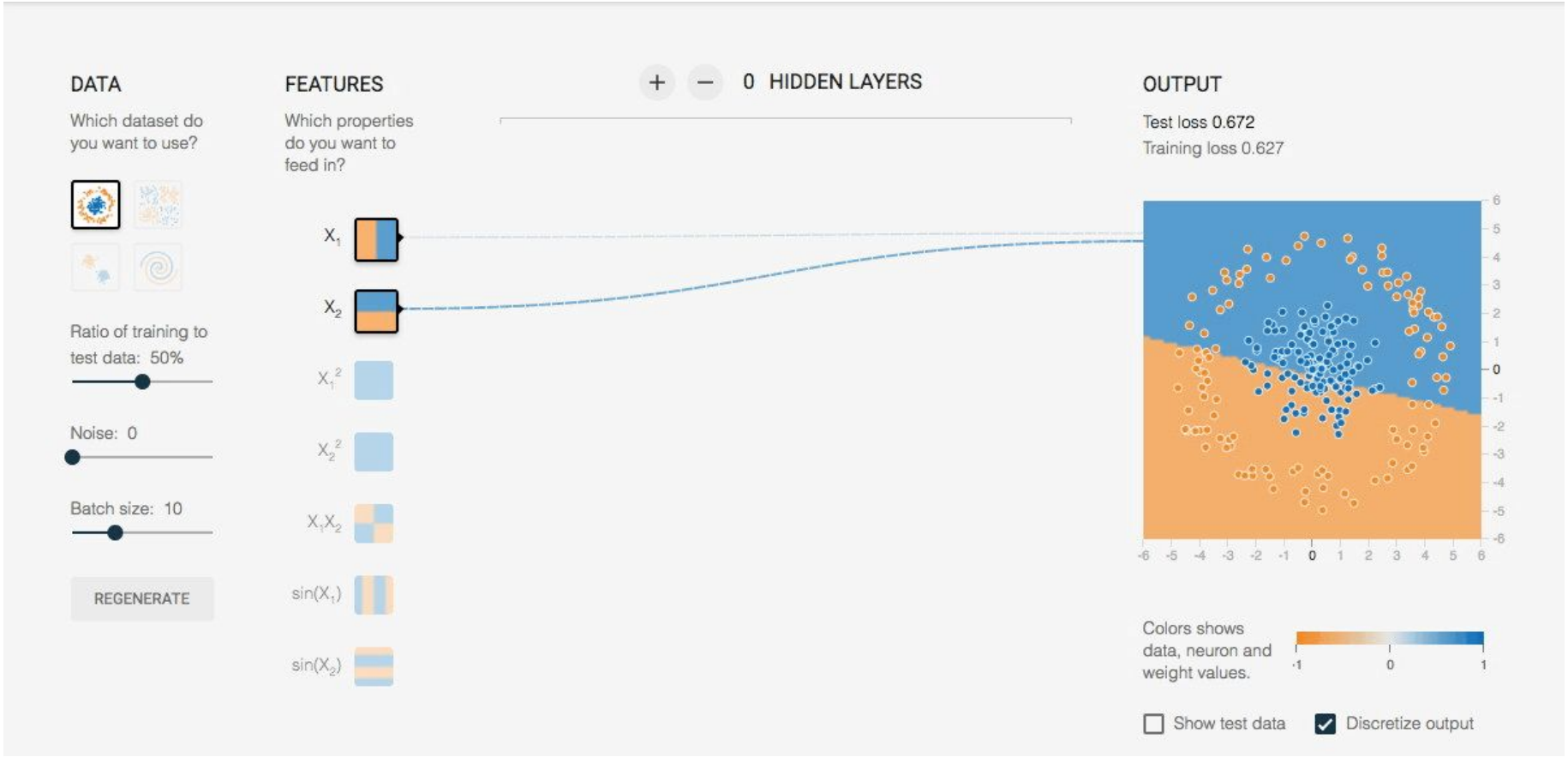

$\hat{y} = g(1 + 3x_1 - 2x_2)$
x_2
x_1
$1 + 3x_1 - 2x_2 = 0$

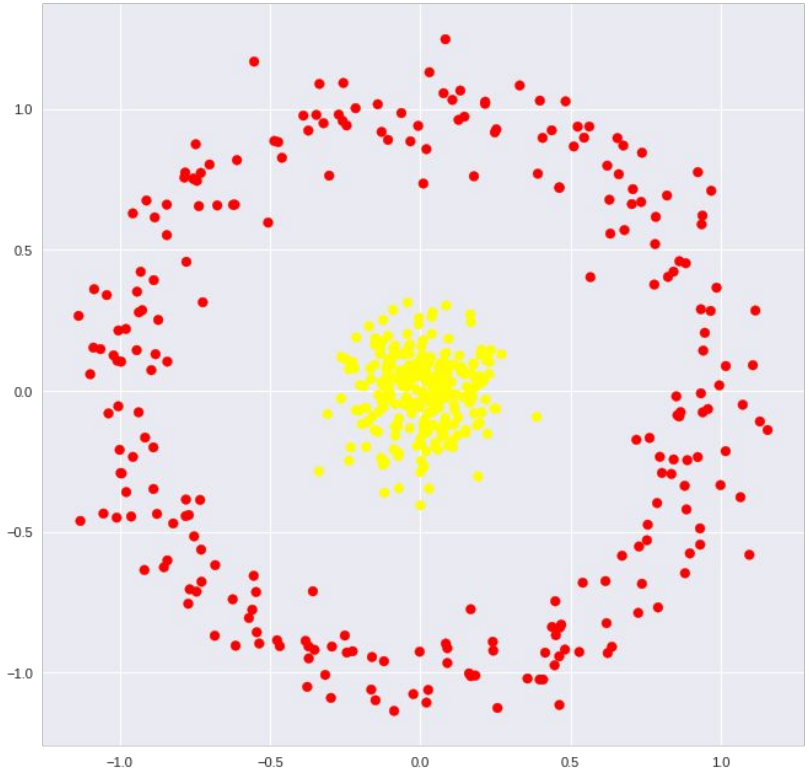
$\hat{y} = g(1 + 3x_1 - 2x_2)$



$\hat{y} = g(1 + 3x_1 - 2x_2)$




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
X1²
X2²
X1X2
sin(X1)
sin(X2)
+
−
0 HIDDEN LAYERS
OUTPUT
Test loss 0.672
Training loss 0.627
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output



Feature engineering to the rescue

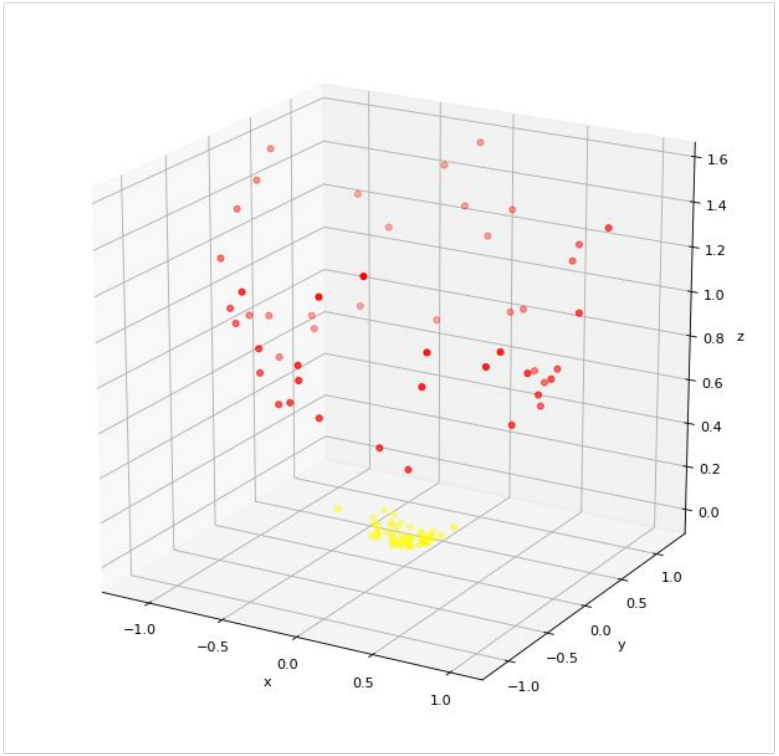
What if we add a new feature

`z = x^2 + y^2.`

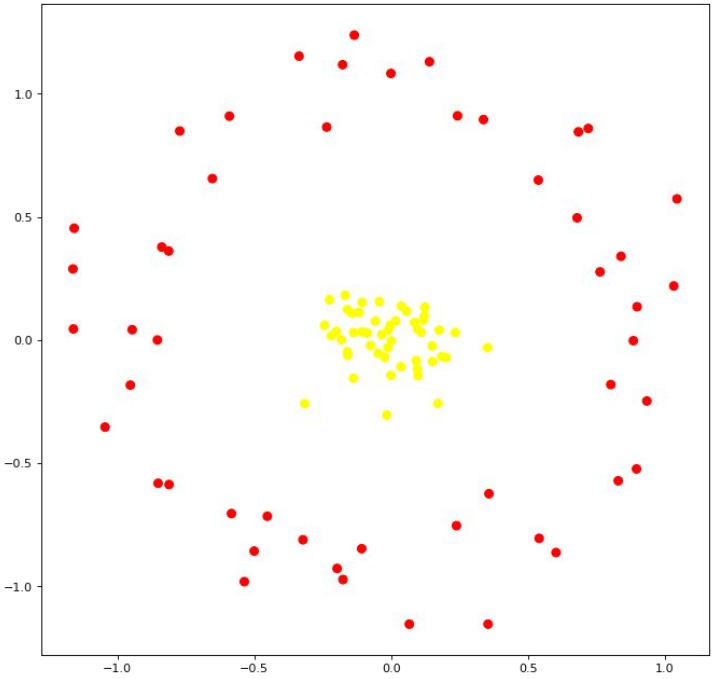
Intuition

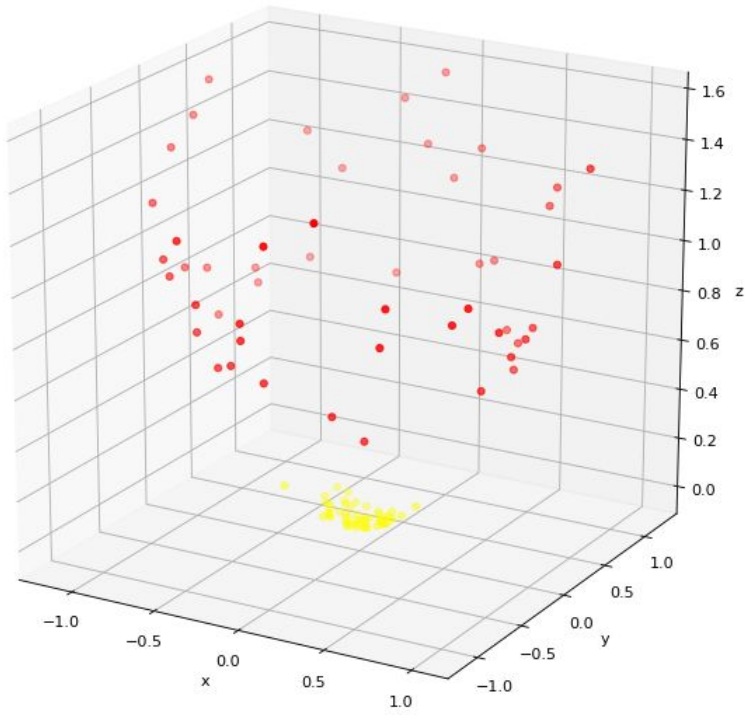
- All values of *z* will be positive.
- Yellow points are closer to the origin…
- So sum of their squared coords will be lower than red!

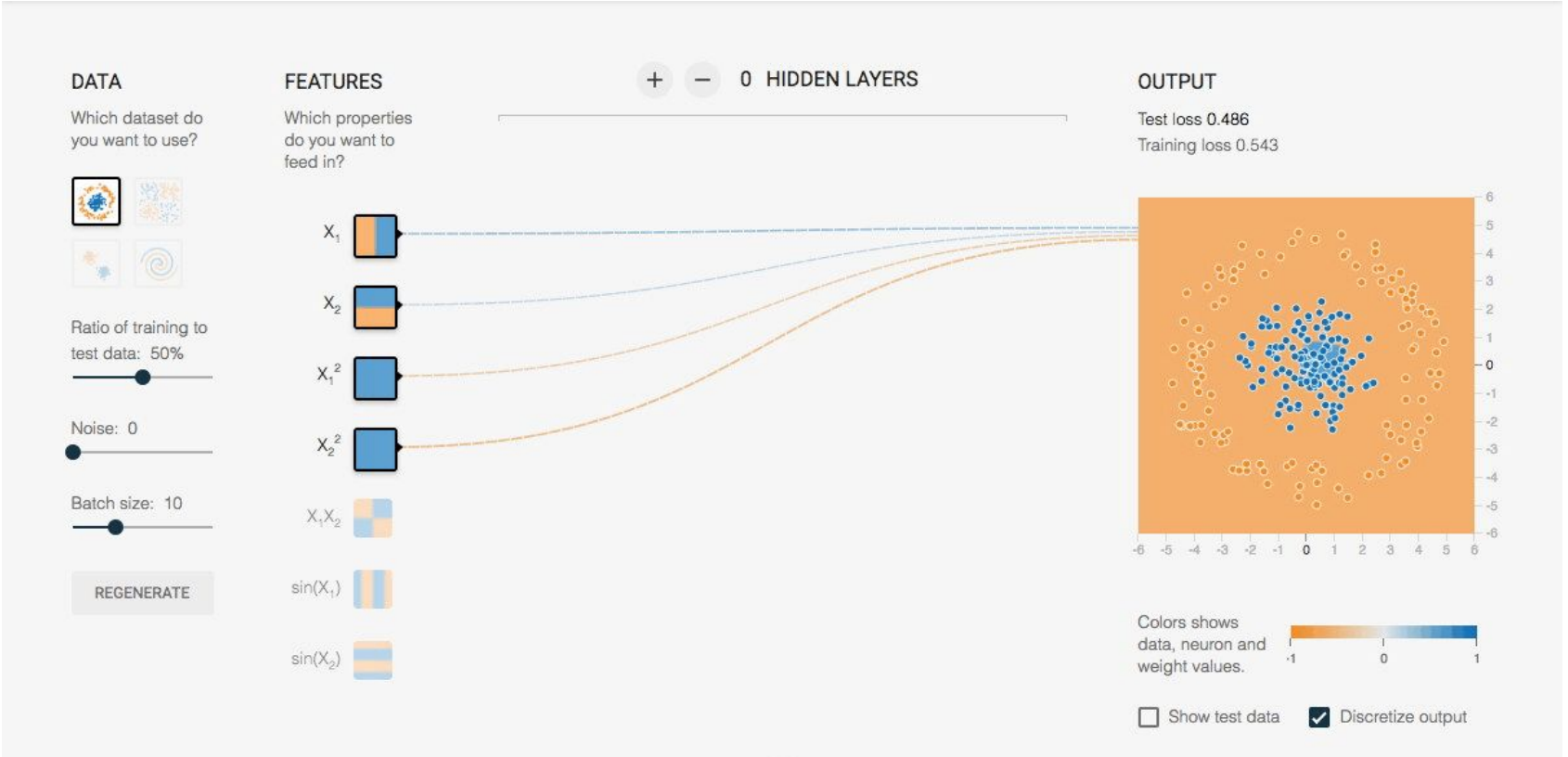
```
fig = plt.figure()
zs = xs**2 + ys**2 # feature engineering
ax = fig2.add_subplot(111,projection='3d')
ax.scatter(xs, ys, zs, c=y)
plt.show()
```



Linearly separable in the new space!






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
X1²
X2²
X1X2
sin(X1)
sin(X2)
+
−
0 HIDDEN LAYERS
OUTPUT
Test loss 0.486
Training loss 0.543
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Observations

Feature engineering is challenging

- But, good features are essential

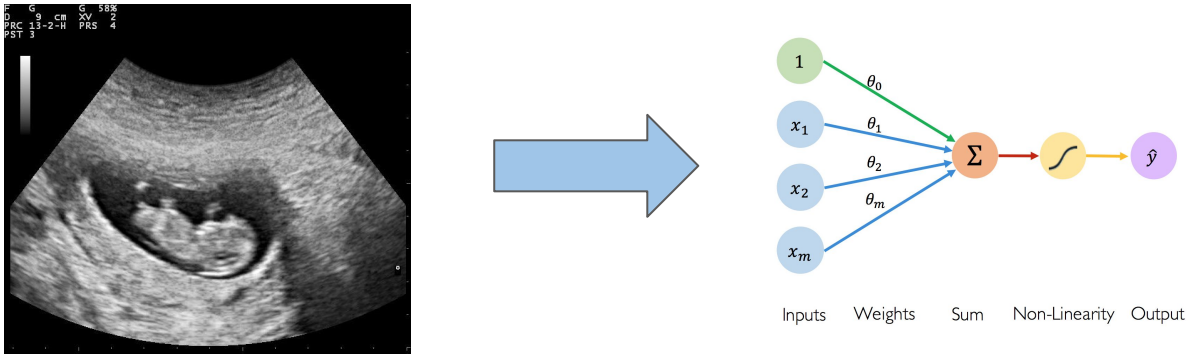
Is it realistic to do feature engineering on high dimensional data?

- (If not, can features be learned automatically?)

How do we handle high dimensional data?

Ultrasounds

So far, all the features we've used have been manually entered. What if we wanted include high-dimensional data, like an ultrasound in our model? One idea, feed every pixel value as an input to our model.



Image

Discussion

How well do you think this will work? Can you name one reason why it may or may not with logistic regression?

Discussion

How well do you think this will work? Can you name one reason why it may or may not with logistic regression?

Strategy:

1. Use a logistic regression model.

2. Create one input for each pixel on the image (so for a 1,000x1,000 image, we have 10^6 inputs, assuming greyscale, or 3x if color).

3. Train the model using a bunch of annotated data.

Some possible answers

- In this domain, probably it won't work at all. Though it will for simple images, we'll code this up later in the course.

- Model only learns **how much each pixel correlates with each output we want to predict,** independently. Individual pixels probably have negligible correlation, so this isn't likely to work.

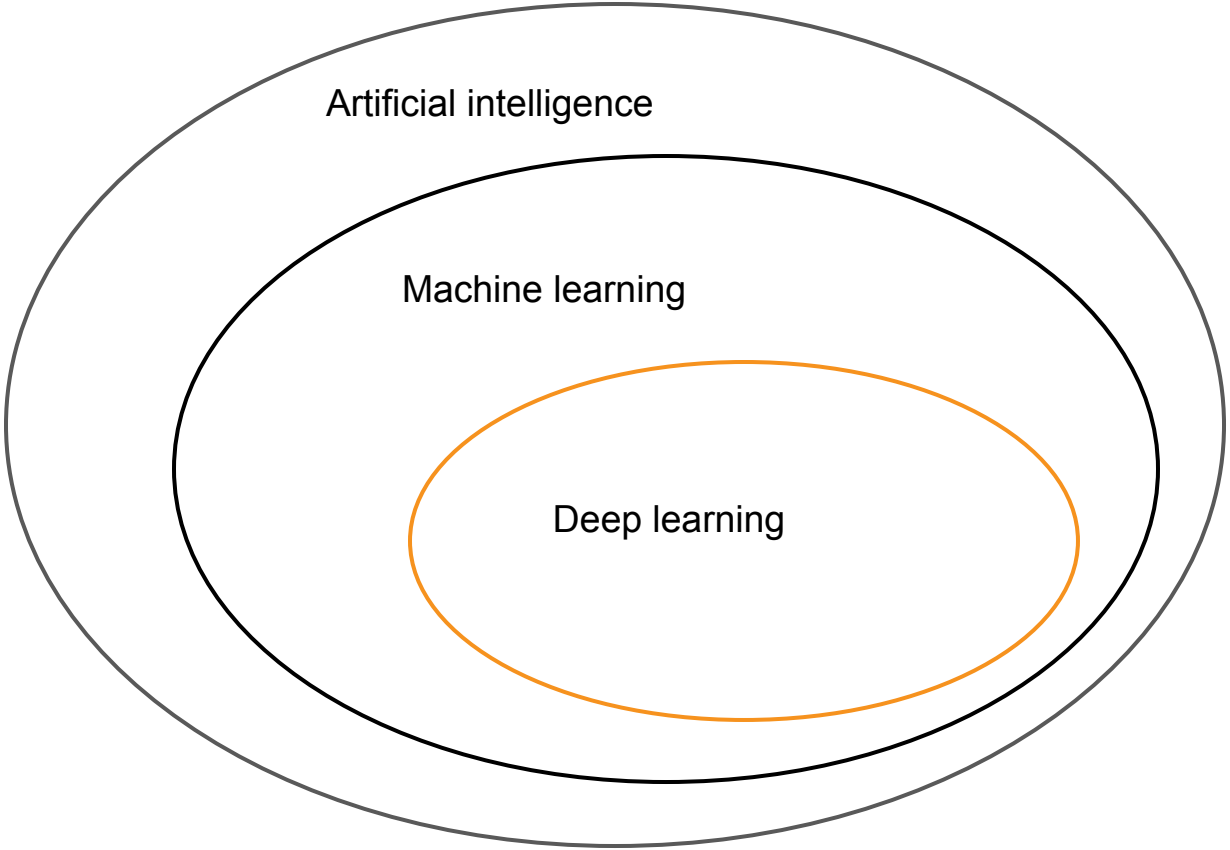
- Not invariant to rotation / translation (how the baby, mother, and scanner are positioned are important); Sensitive to different lighting conditions; We have a video, not a picture (an ultrasound gives us a sequence of frames); How to account for different types of scanners? Etc.

Some possible answers

- In this domain, probably it won't work at all. Though it will for simple images, we'll code this up later in the course.

- Model only learns **how much each pixel correlates with each output we want to predict,** independently. Individual pixels probably have negligible correlation, so this isn't likely to work.

- Not invariant to rotation / translation (how the baby, mother, and scanner are positioned are important); Sensitive to different lighting conditions; We have a video, not a picture (an ultrasound gives us a sequence of frames); How to account for different types of scanners? Etc.

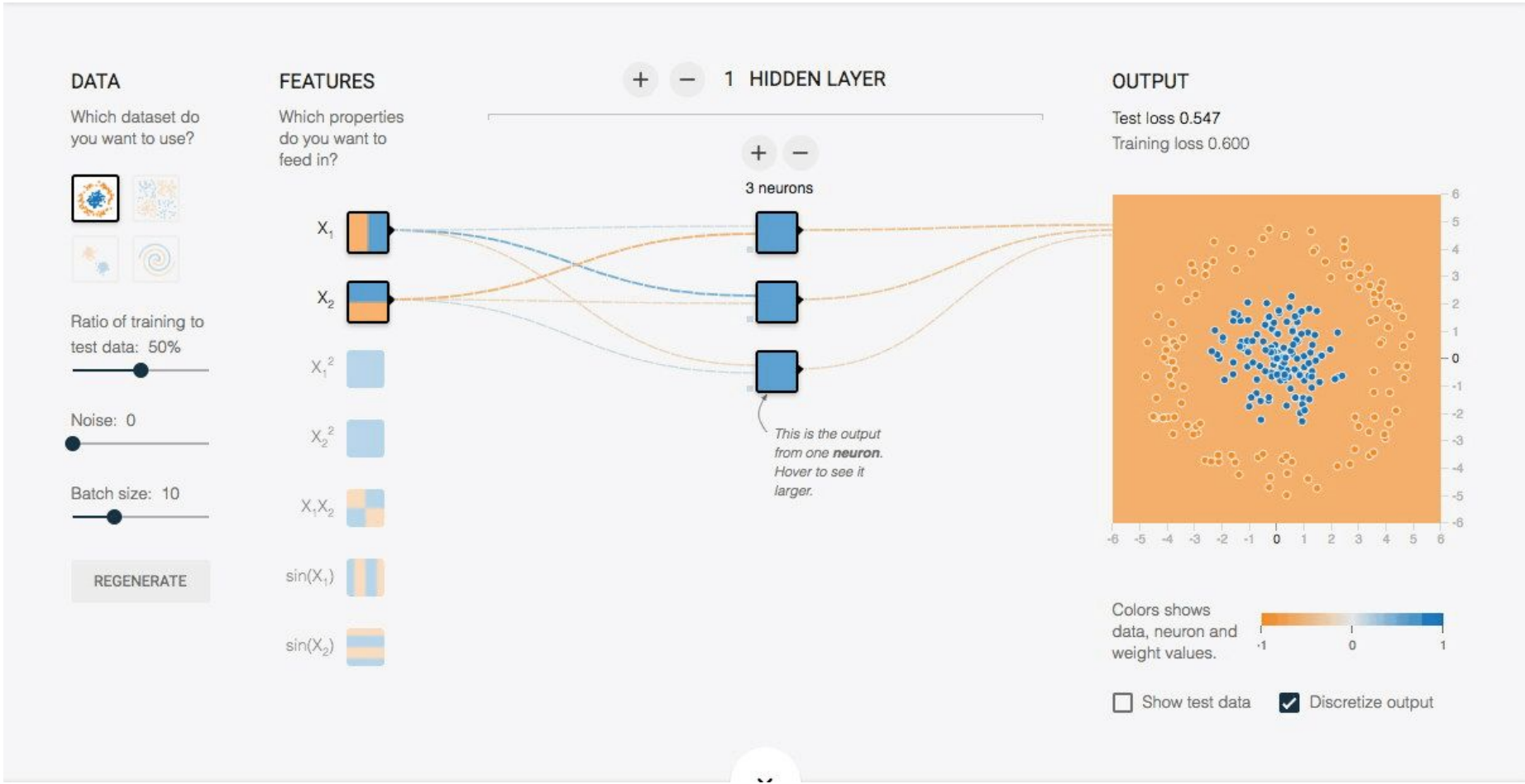

Artificial intelligence
Machine learning
Deep learning

Representation learning

- We've talked a little about tasks that are hard for people but easy for computers. But, what about the **reverse**?

- A lot of human intelligence feels **automatic**, e.g., understanding images, speech, and text. But describing them to a computer is hard.

- Can we develop a program to learn a representation automatically? (And could that representation also learn to handle **variation** in the world? Different accents? Lightning conditions?)

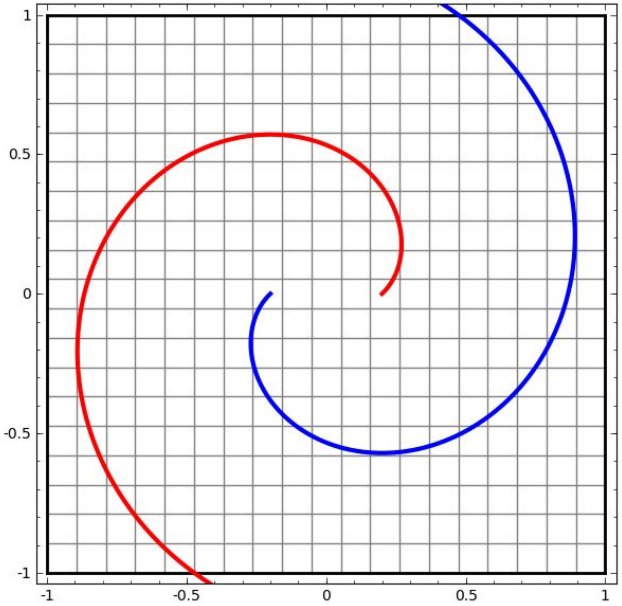

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
1 HIDDEN LAYER
3 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.547
Training loss 0.600
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Intuition for *why* this works?

Intuition takes a while to develop.

- Work is being done to articulate how and why networks work.

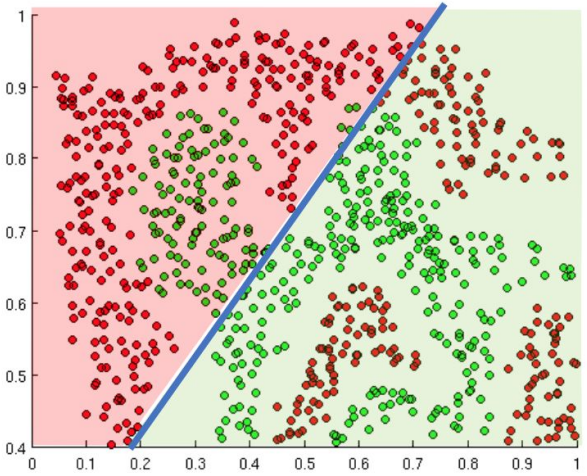
- The clearest such expositions today are at distill.pub and colah.github.io



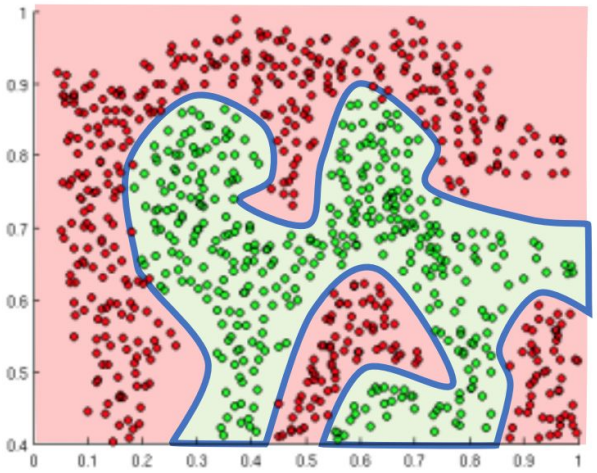
Demo

- What happens when we take away the activations

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



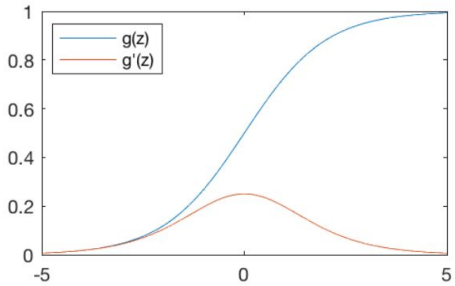
Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Various activation functions

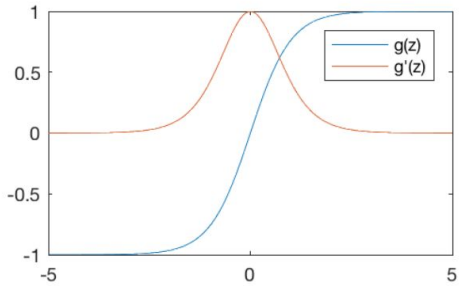
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

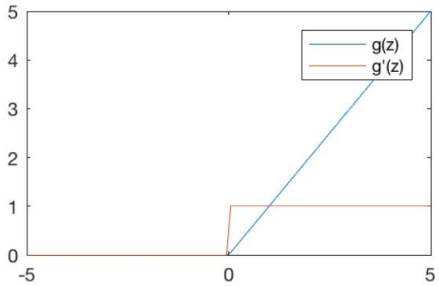
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

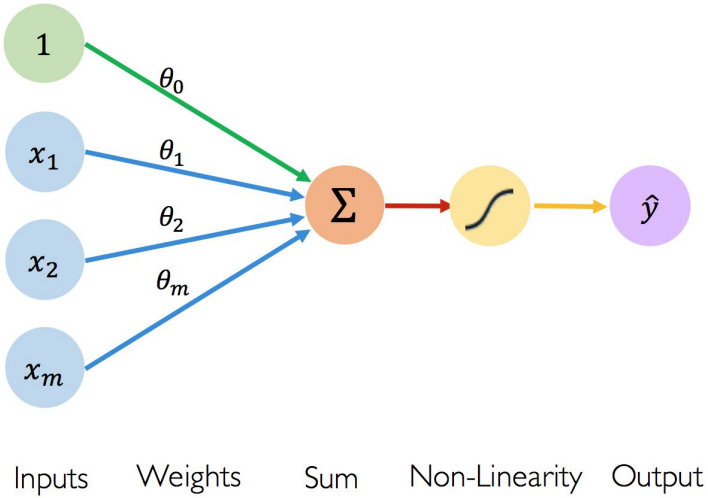


$$g(z) = \max(0, z)$$

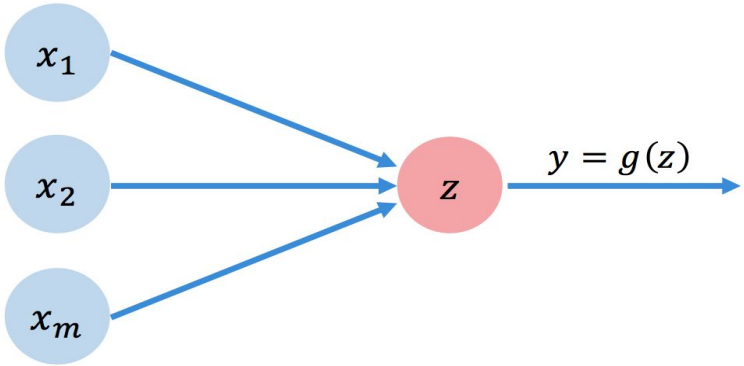
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

Building blocks

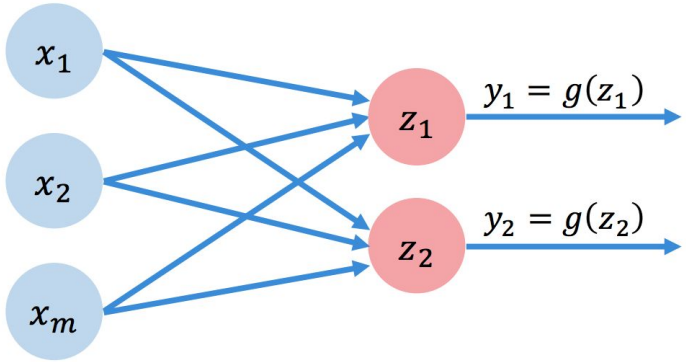
A logistic unit, or “neuron”



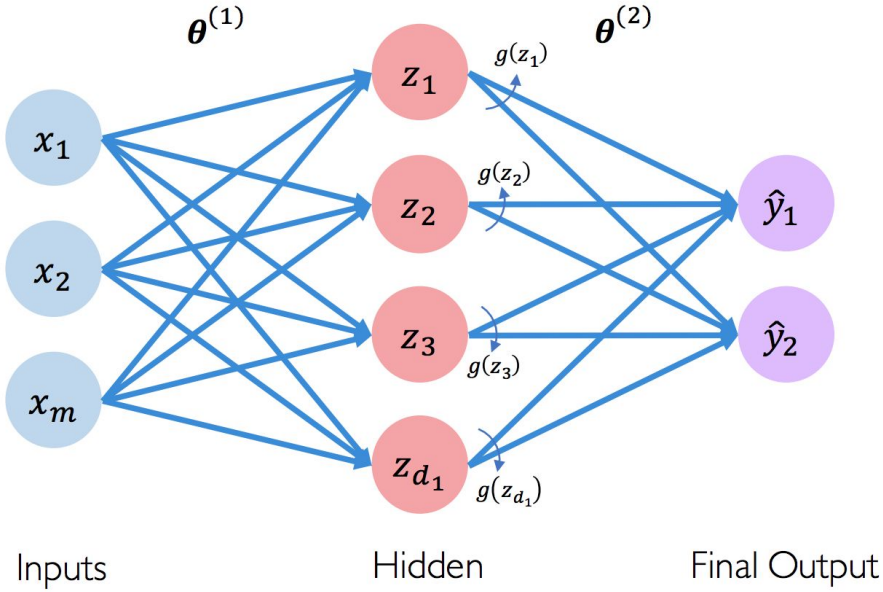
Simplified drawing

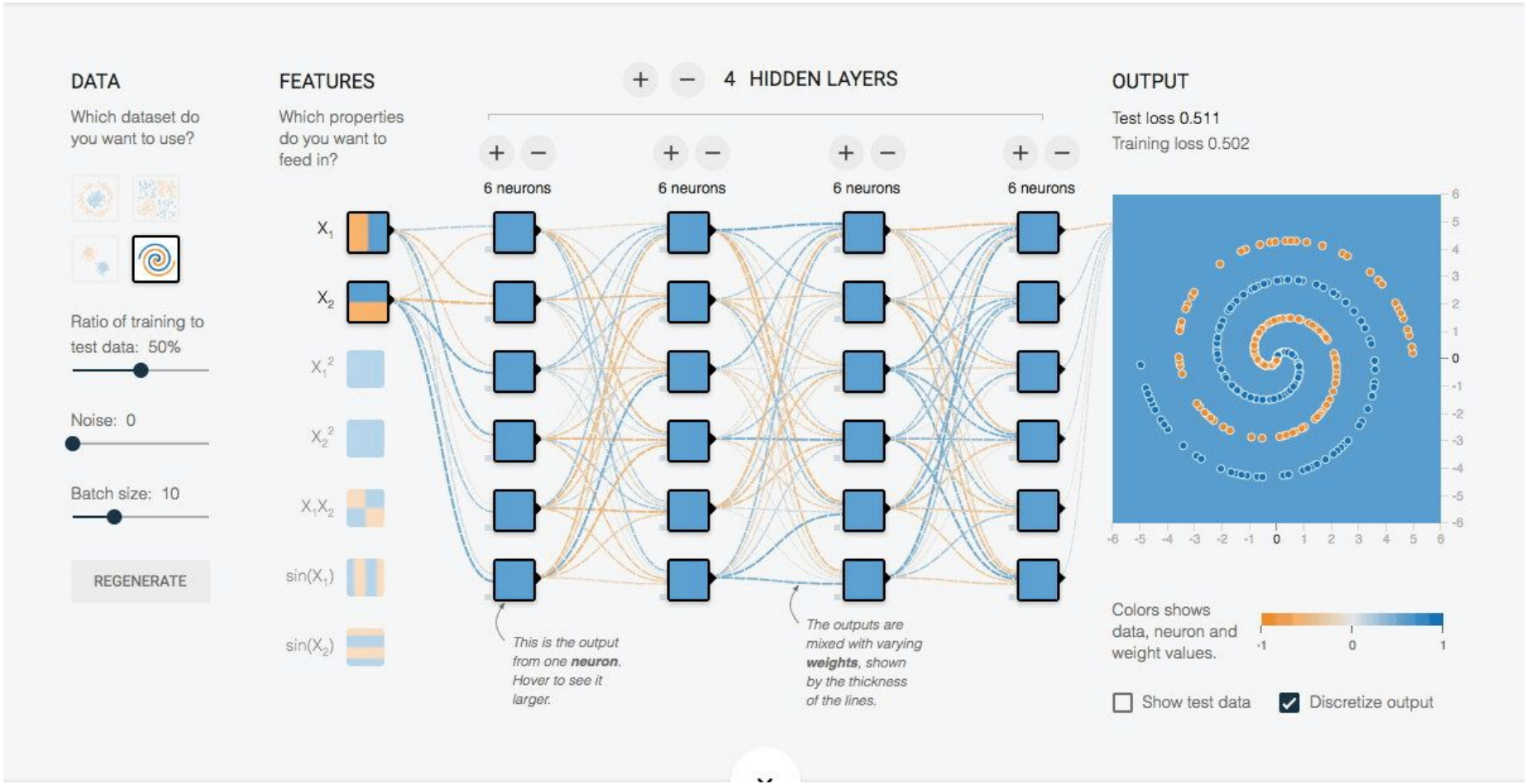


Now, with multiple outputs



A neural network (with one hidden layer)




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?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
4 HIDDEN LAYERS
6 neurons
6 neurons
6 neurons
6 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 0.511
Training loss 0.502
Colors shows data, neuron and weight values.
Show test data
Discretize output

Equation view may be simpler

Forward pass is a series of matrix multiplies (one per layer) followed by a pointwise application of a non-linearity.

A linear score function

$$f = Wx$$

A single hidden layer (ReLU activation)

$$f = W_2 \max(0, W_1 x)$$

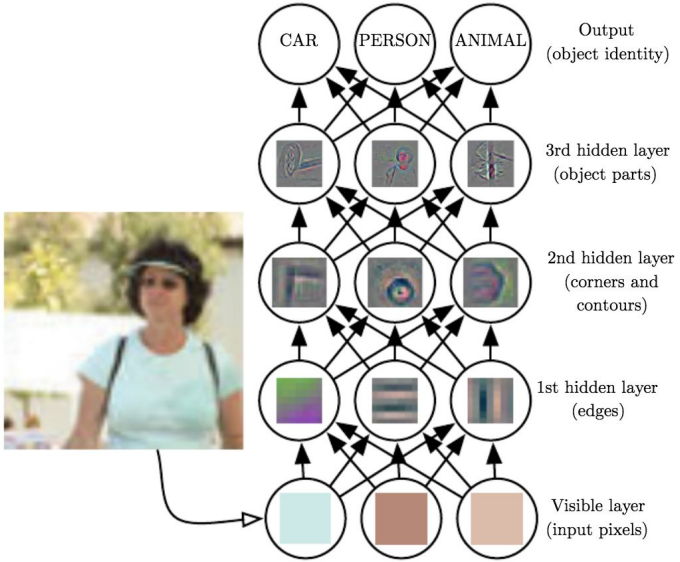
Two hidden layers

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Why does the “deep” in deep learning help

Learned representations are **hierarchical**

- We can directly visualize learned representations with images.



AlphaGo

BBC | News | Sport | Weather | iPlayer | TV | Radio | More | Search

NEWS | Find local news

Home | UK | World | Business | Politics | Tech | Science | Health | Education | Entertainment & Arts | More

Technology

Google achieves AI 'breakthrough' at Go

An artificial intelligence program developed by Google beats Europe's top player at the ancient Chinese game of Go, about a decade earlier than expected.

27 January 2016 | Technology

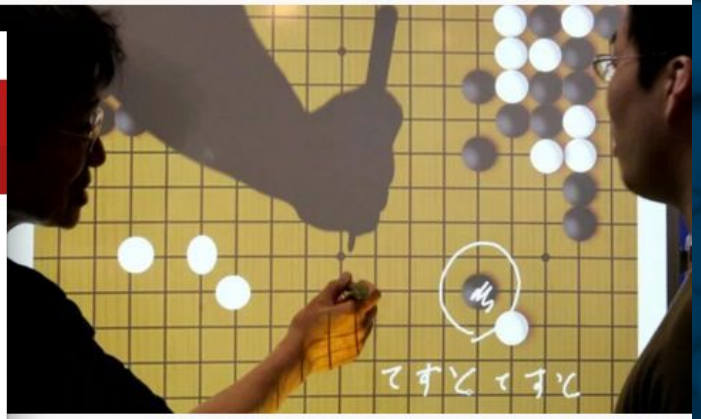
How did they do it?

What is the game Go?

Facebook trains AI to beat humans at Go

Google's AI just cracked the game that supposedly no computer could beat

By Mike Murphy | January 27, 2016

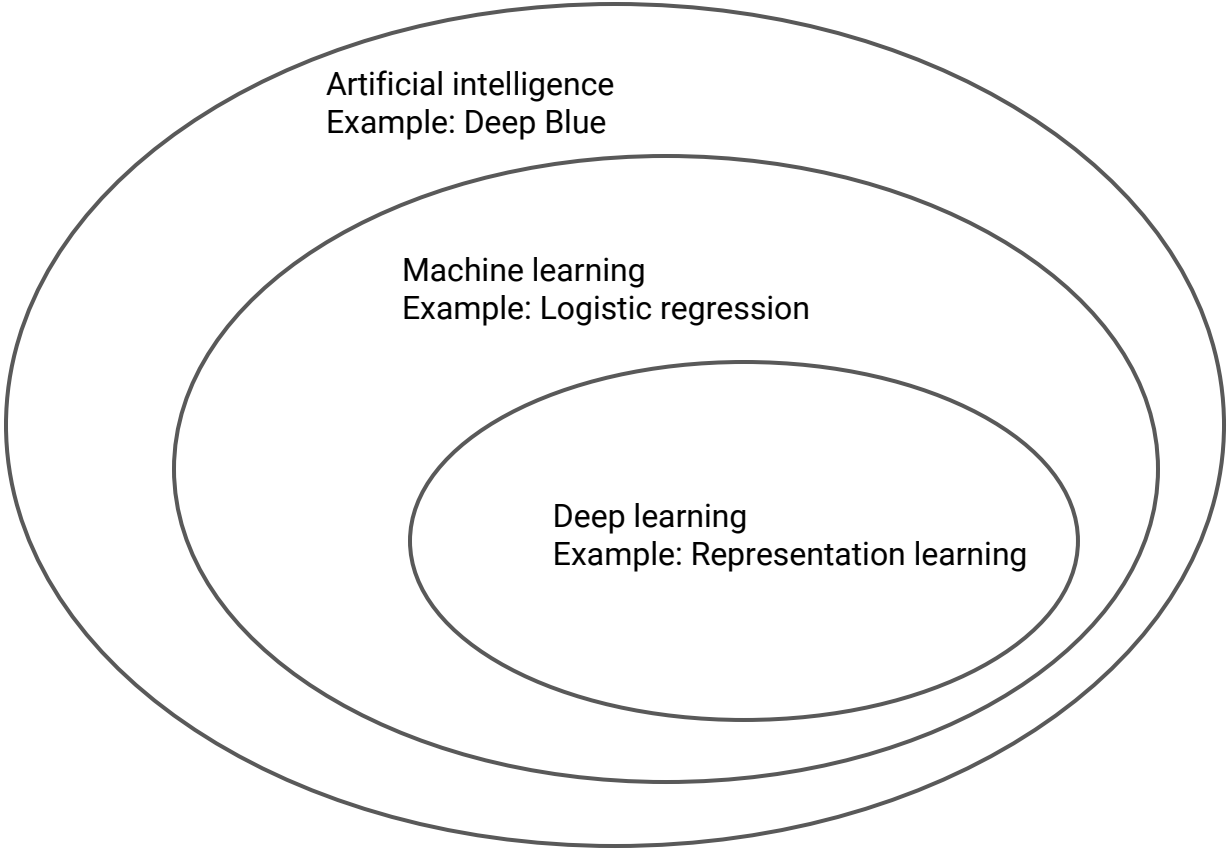


Going up. (Reuters/Kiyoshi Ota)

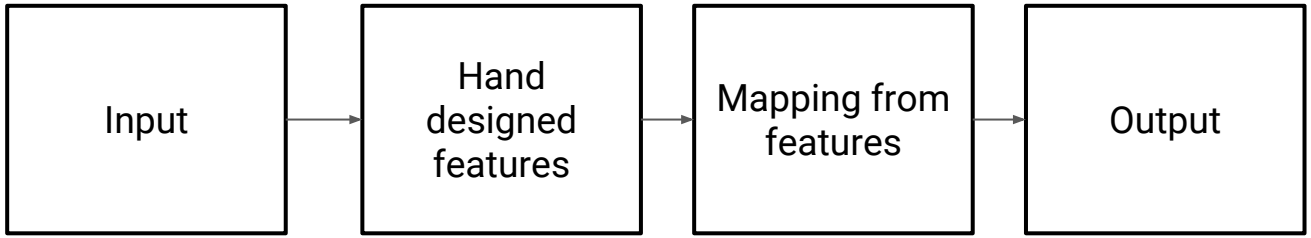
Computers have slowly started to encroach on activities we previously believed only the brilliantly sophisticated human brain could handle. IBM's Deep Blue supercomputer beat Grand Master Garry Kasparov at chess in 1997, and in 2011 IBM's Watson beat former human winners at the quiz game *Jeopardy*. But the ancient board game Go has long been one of the major goals of artificial intelligence research. It's understood to be one of the most difficult games for computers to handle due to the sheer number of possible moves a player can make at any given point. Until now, that is.

Deep dream and AlphaGo both use CNNs!

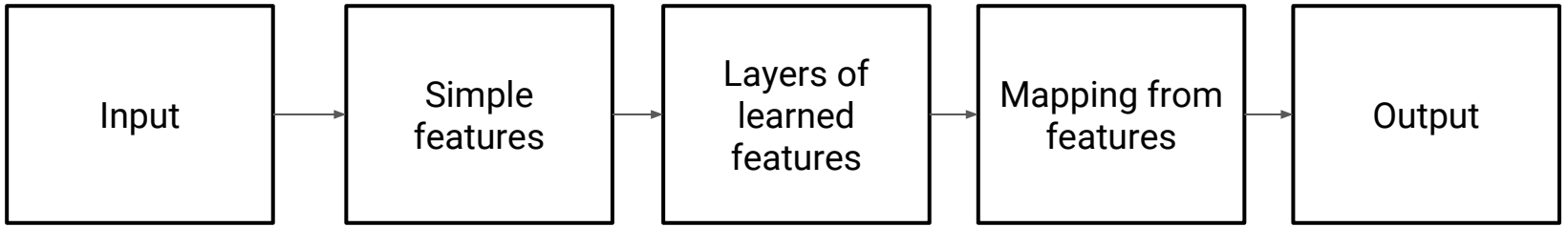
Quick discussion: why the dogs?


Artificial intelligence
Example: Deep Blue
Machine learning
Example: Logistic regression
Deep learning
Example: Representation learning

Another view

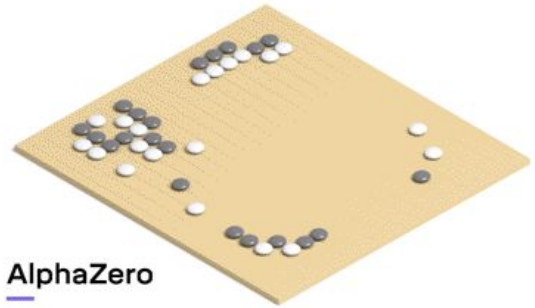


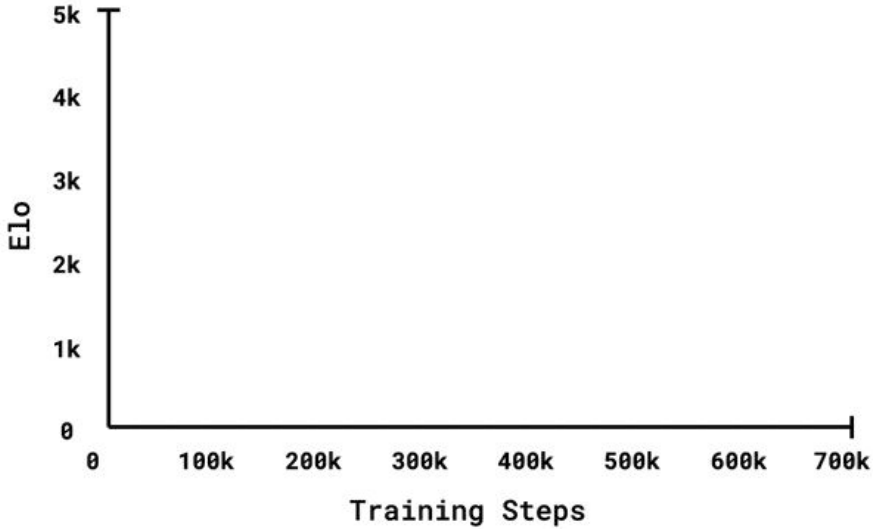
Classic ML



Deep Learning

Combined with reinforcement learning





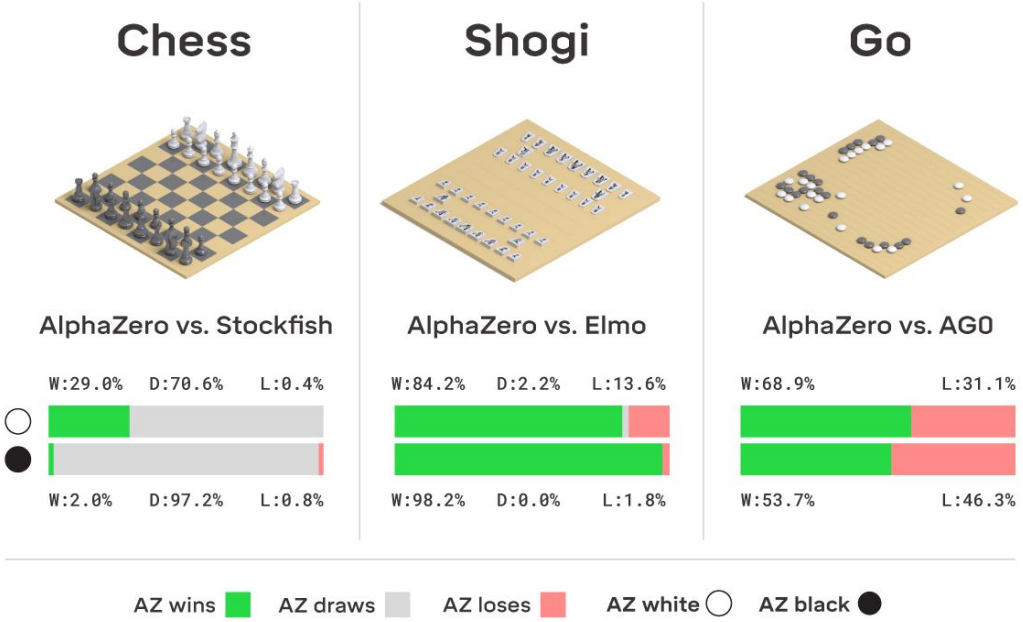
AlphaZero plays chess in a new way

A traditional chess engine typically plays for a mix of material and positions.

AlphaZero is exceptionally positional, and plays for activity.

It **regularly** sacrifice material for activity.

It uses a **much lower** search depth than a traditional engine (e.g., “thinks carefully” about fewer positions).



Why now?

Why now? Ingredients for success.

Algorithms, data, compute, libraries

- Key algorithms have been around for relatively a long time. Backprop (**1986**), CNNs (**1995**); LSTMs (**1997**).

- Comparatively minor improvements were needed to make them work well in practice (ReLU activation, better weight initialization, …)

This field has fallen out of favor several times. It took courage for researchers to continue to investigate before it was thought likely to succeed.

Data

ImageNet Large Scale Visual Recognition Challenge

- 1K categories
- 1.4M images
- Run annually beginning in 2010

ImageNet

ImageNet Large Scale Visual Recognition Challenge

What I learned from competing against a ConvNet on ImageNet

Discussion: How good are people are at this task?

What do you think their top-5 accuracy is? (How often is the correct label is in their 5 most confident guesses?)

Malamute ?
Husky ?
Wolf ?
Zebra ?
Fox ?

Here you're given 5 options, but in reality you'd have to choose from 1,000

Discussion: Why is this harder than it seems?

Can you propose one reason why accuracy may be lower than we expect? This will become relevant later when we explore medical imaging.

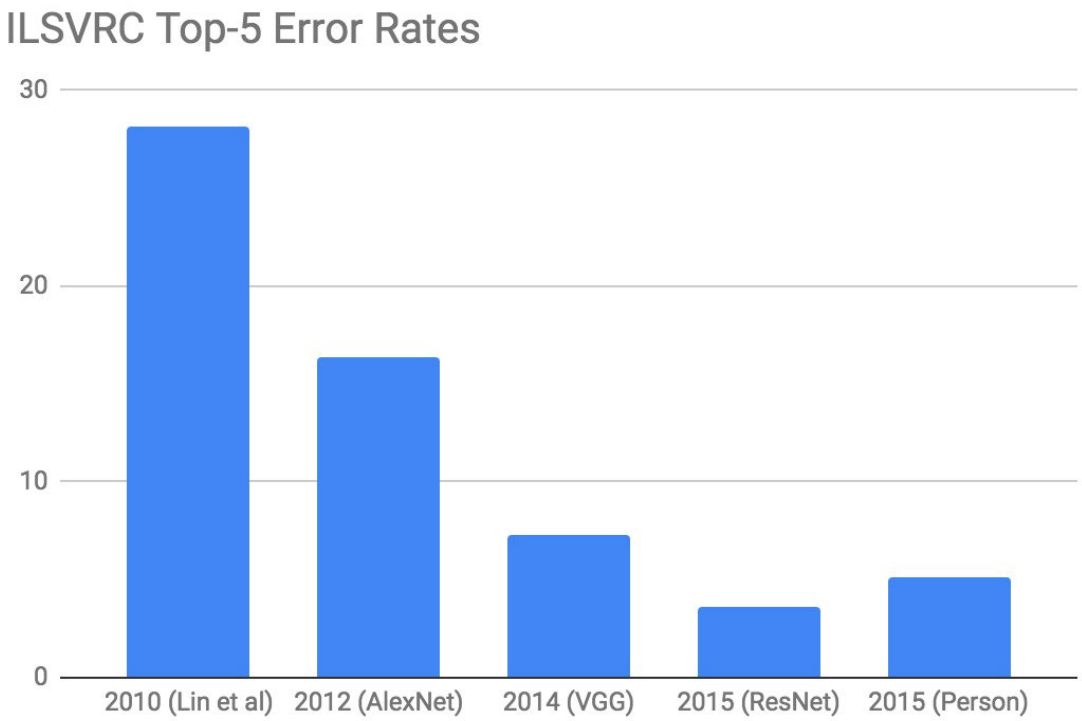
Malamute ✗
Husky ✓
Wolf ✗
Zebra ✗
Fox ✗

Possible answers.

- **Noisy data**
  - Images may contain incorrect labels
  - Images may be blurred, or otherwise corrupted (impossible to guess correct label)

- **Fatigue**: imagine you had to manually classify all 1.4M images

- **Difficulty**: Malamute, or a Husky?



The jump in accuracy in 2012 was from a CNN. Success was also had in speech recognition, but this is when the community really began to take notice.

Imagine making this distinction with low-resolution images, from various angles, and in different lighting and background conditions.

An Alaskan Malamute (left) and a Siberian Husky (right).

Improving Inception and Image Classification in TensorFlow

Compute

Forward and backpropagation (the key steps in inference and training) involve mostly matrix multiplies.

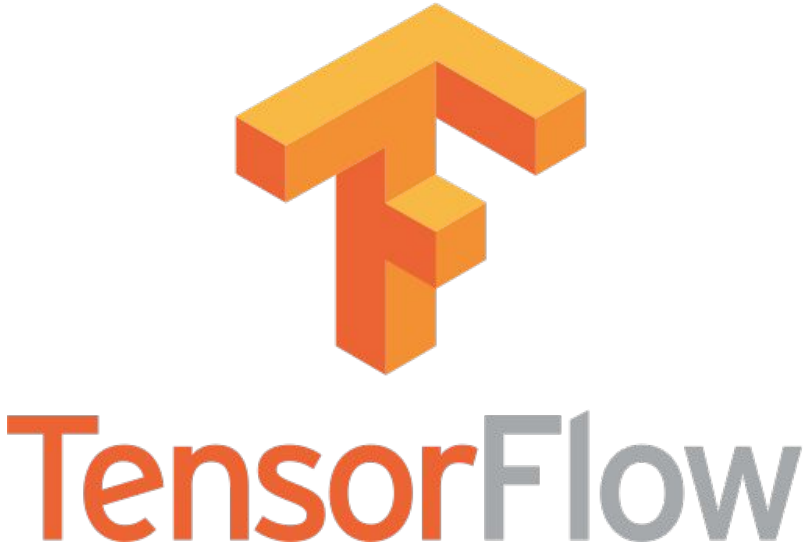
- These are easily parallelizable (at least for basic implementations, DL at scale is an entire discipline)
- Compute with GPUs was **5-10x** cheaper per dollar than with CPUs in 2012.
- Now it's **10-30x** cheaper.

Colab provides a free GPU, that's all you need for this course.

Software

TensorFlow

- Open-source deep learning framework shared by Google in 2015.
- Apache 2.0 license.
- We will cover both Symbolic APIs (e.g., Keras Sequential) and Imperative APIs (e.g., subclassing)
- From here you can branch out to any other library.

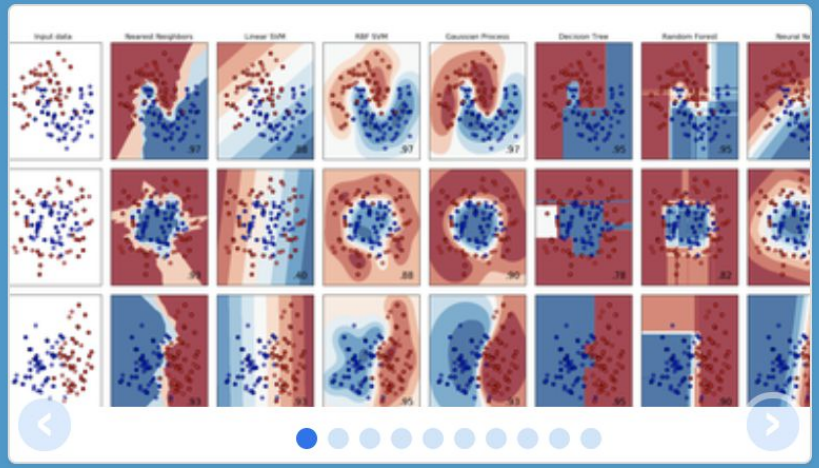


Discussion: Why are companies giving away the farm?

Keras

- An outstanding library, famous for **ease of use**.
- A **large** body of research is published with Keras implementations.
- Transformative for many people when released.
- Assemble neural networks from Lego-like building blocks.

Recently selected as **TensorFlow's high-level API** (also exists as an independent open-source project). We're actively working on updating TensorFlow tutorials to use this API.



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

- An **outstanding general purpose ML library**, **highly recommended**. In addition to a clean API design, it has the best examples and documentation in the business, as well as many useful utilities. Check out the examples gallery.

- Your best bet for most ML tasks outside of Deep Learning (we often use utilities from scikit-learn in programs written in TensorFlow, as well).

You’re welcome to use other libraries

- For programming assignments and your class project, you are **welcome to use any library** including PyTorch, Chainer, MXNet, CNTK, and others.

- We will cover a Chainer / PyTorch / TF.Eager style API in this course (funny how libraries borrow from one another), but we won’t have cycles to cover all of these individually.

What have these libraries meant for the field?

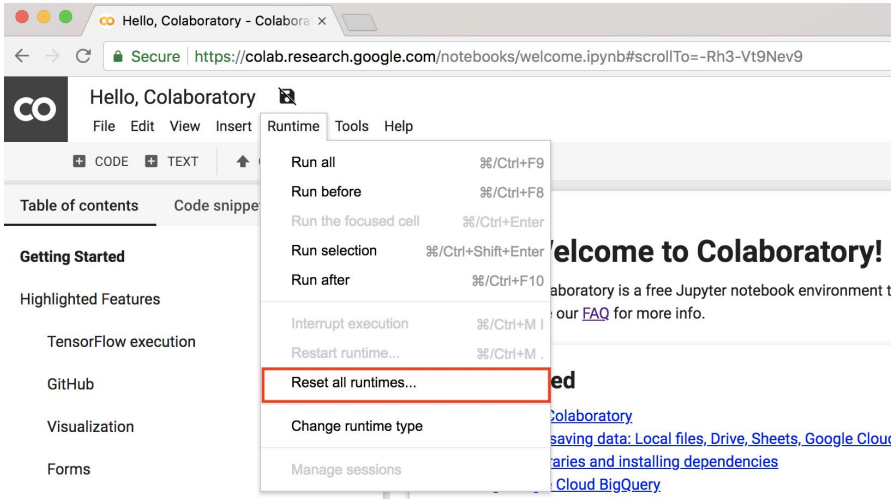
- Much faster time from **idea -> implementation.**
- **Reproducible research**: sharing your paper + code means researchers can iterate faster.
- Applications are much easier to develop with hardware acceleration, and/or support for mobile or the web, as well as distributed training.

Colab Best practices

- Colab notebooks should **automatically install any dependencies**, and **download any data needed**.
- Run **end-to-end** with zero user intervention.
- Colab instances are **ephemeral**. You should not rely on any data or packages previously installed.
- This makes it easy to share **reproducible experiments.**

How-to reset everything

You can reset your VM (including installed pip packages and downloaded files) if you get it into a funny state, like this:



If you’re going to install any software locally

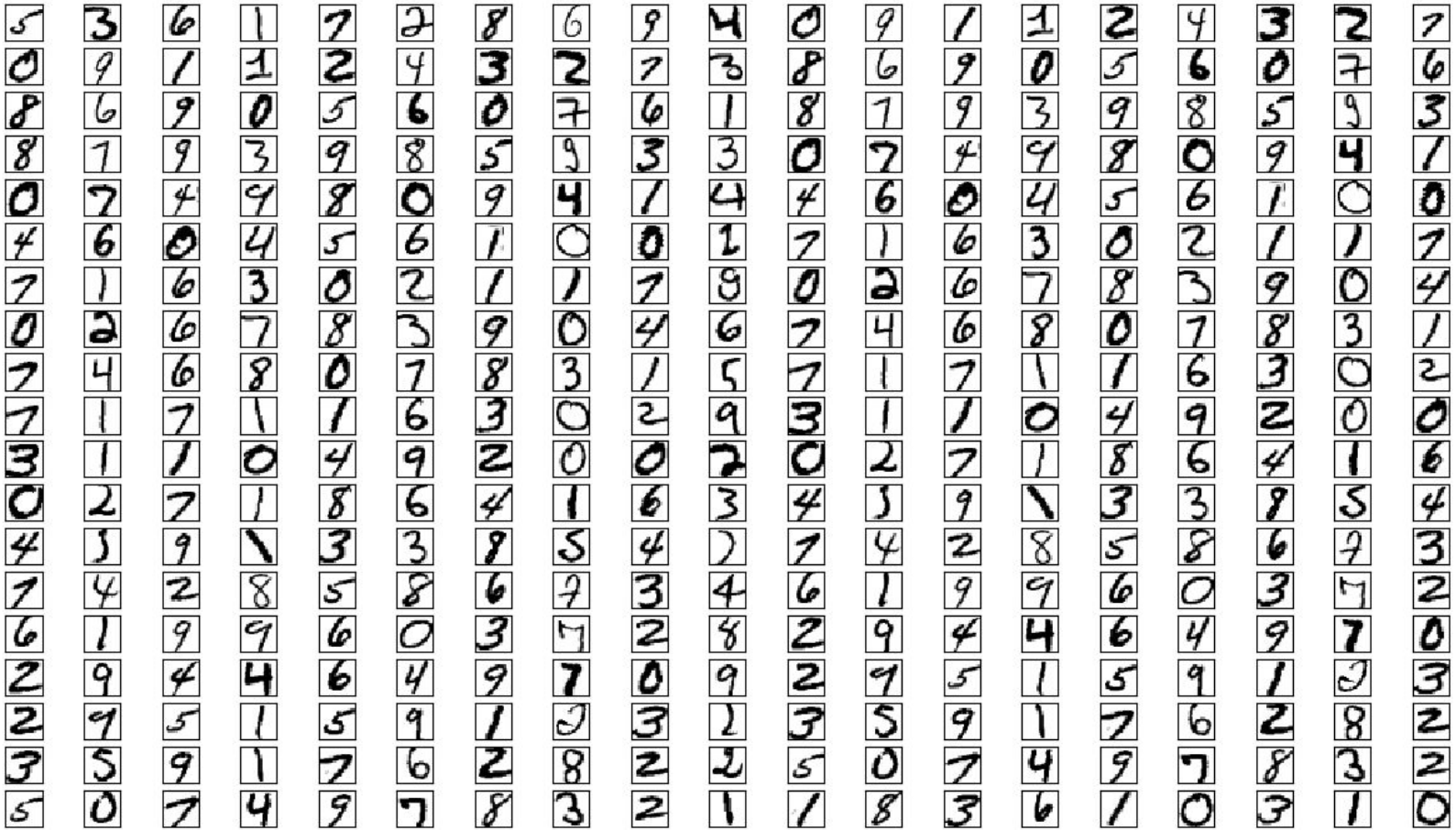
Use a **virtual environment.**

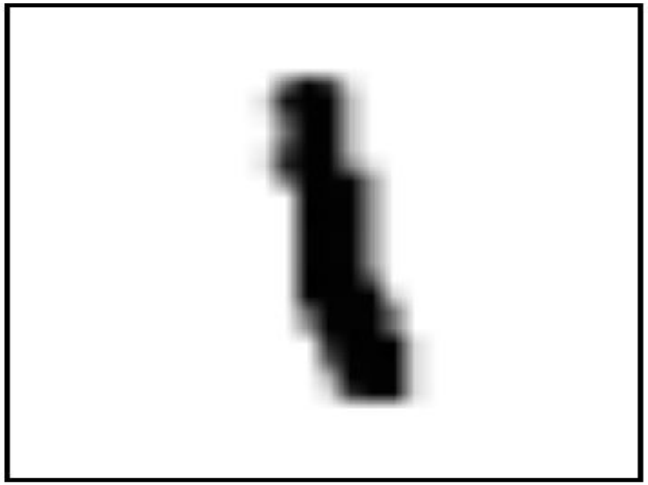
- It’s worth your time to learn how.

Libraries have tons of dependencies, this will make them easier to manager (and easier to start from a clean state if you get into trouble).

- See the TensorFlow installation instructions for an example.

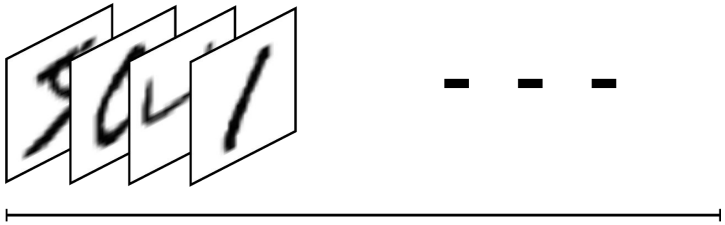
Code-light, but concept heavy.

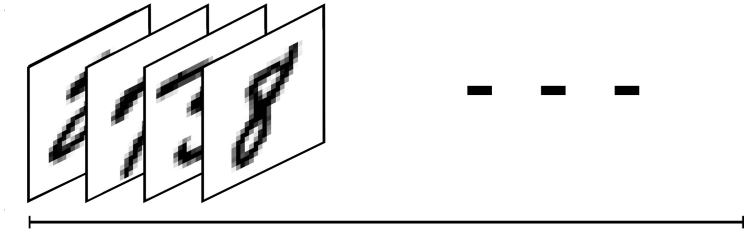




$\approx$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$





```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

tensorflow.org/tutorials

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

tensorflow.org/tutorials

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

tensorflow.org/tutorials

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

tensorflow.org/tutorials

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

tensorflow.org/tutorials

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

tensorflow.org/tutorials

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

tensorflow.org/tutorials

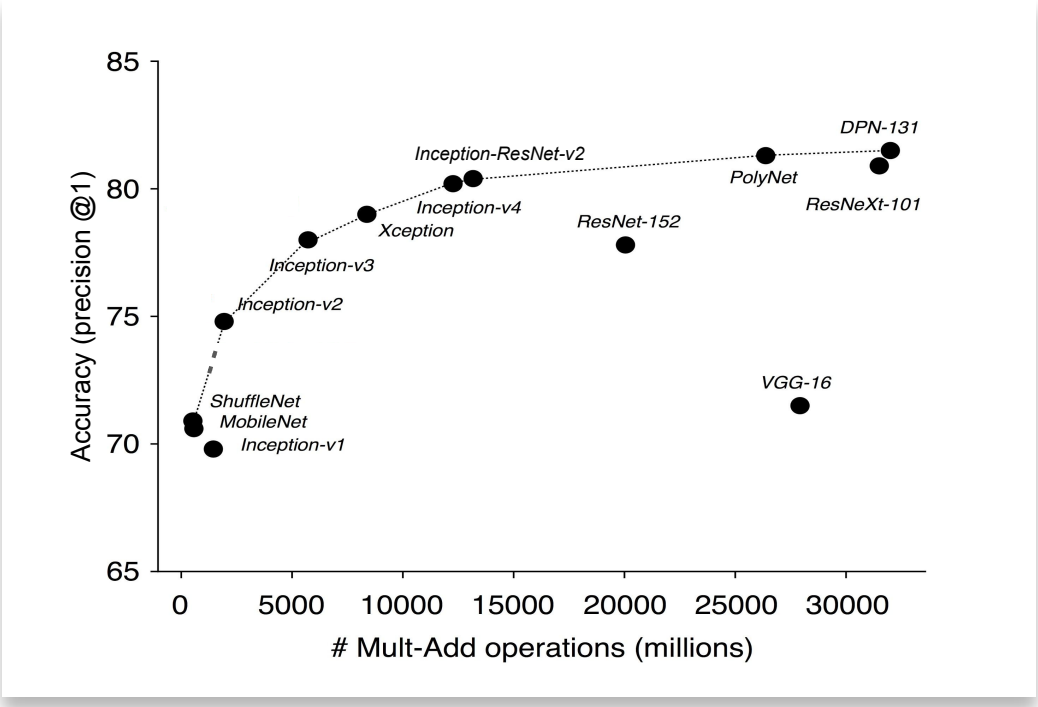
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

tensorflow.org/tutorials

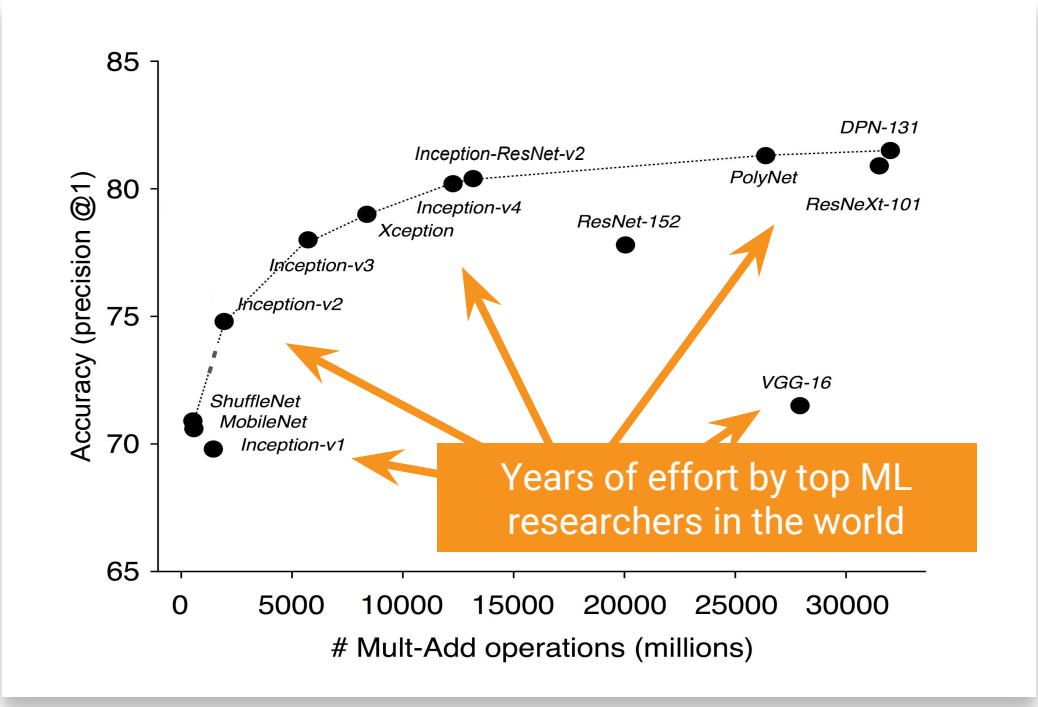
Trends



Top performing CNN-based architectures on ImageNet ILSVRC.

Learning Transferable Architectures for Scalable Image Recognition

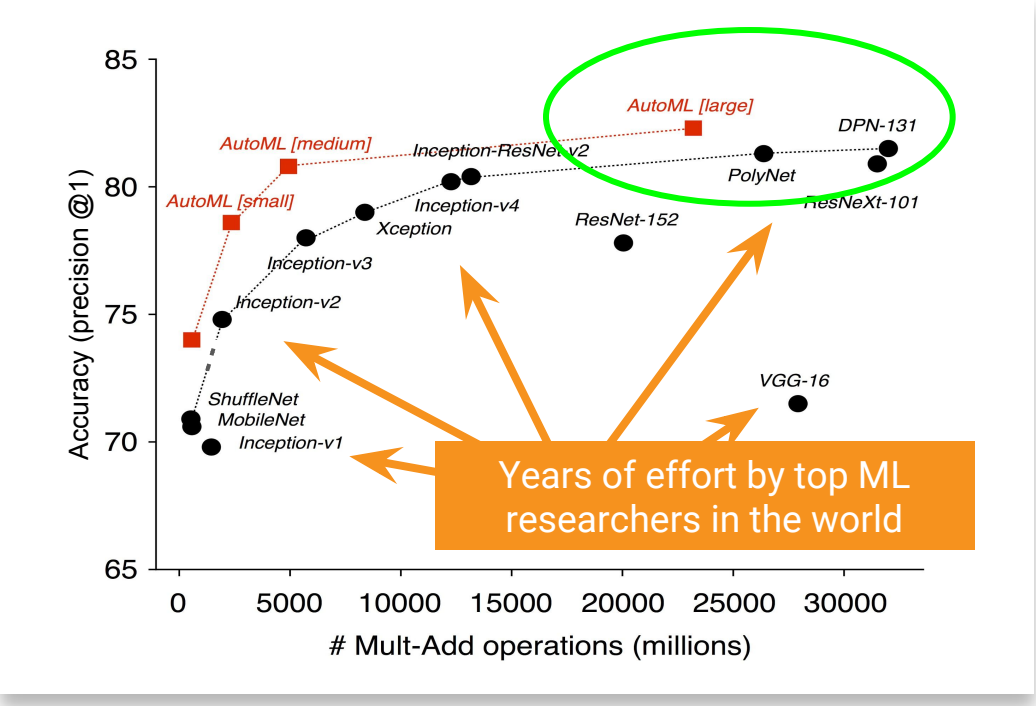
Keynote (TensorFlow Dev Summit 2018)



Top performing CNN-based architectures on ImageNet ILSVRC.

Learning Transferable Architectures for Scalable Image Recognition

Keynote (TensorFlow Dev Summit 2018)

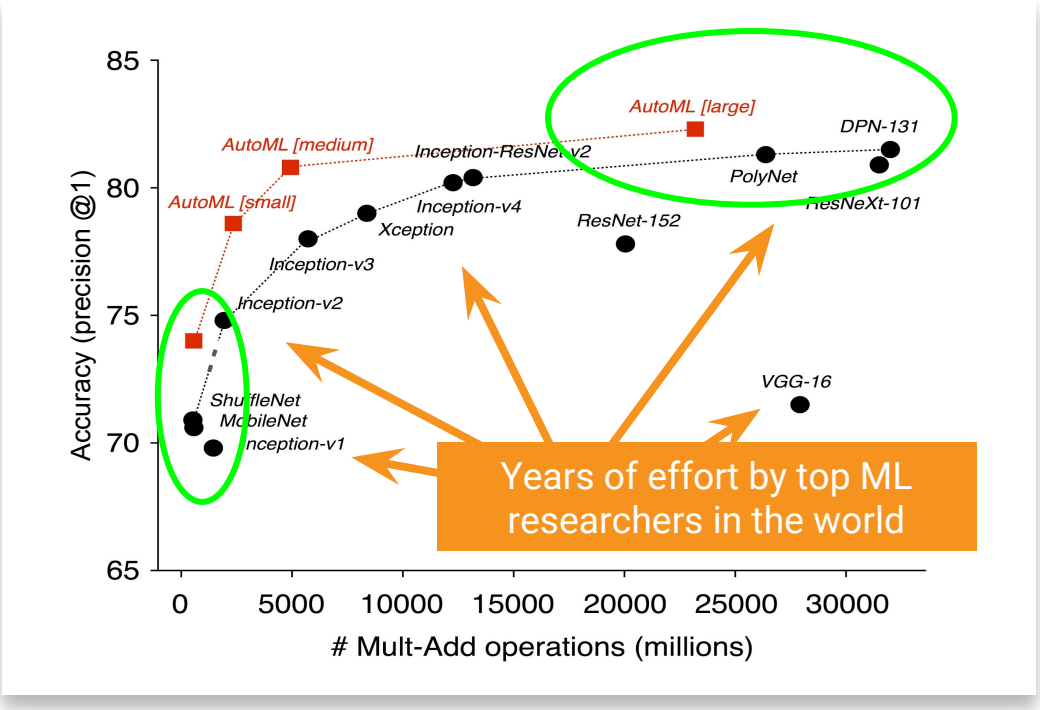


Note: AutoML is a brand. What's important are the ideas expressed in the paper re: architecture search.

Top performing CNN-based architectures on ImageNet ILSVRC.

Learning Transferable Architectures for Scalable Image Recognition

Keynote (TensorFlow Dev Summit 2018)



Top performing CNN-based architectures on ImageNet ILSVRC.

Note: AutoML is a brand. What's important are the ideas expressed in the paper re: architecture search.

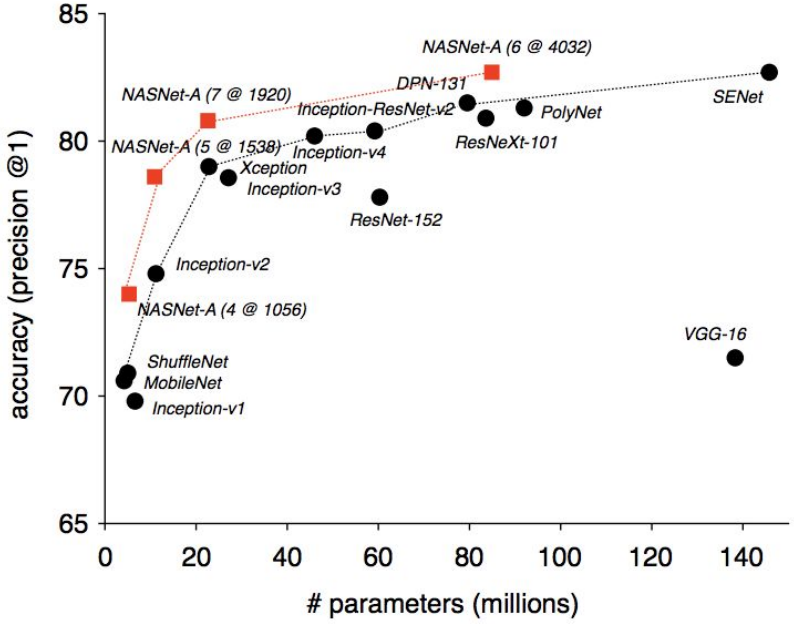
Learning Transferable Architectures for Scalable Image Recognition

Keynote (TensorFlow Dev Summit 2018)

Increasing model size

100M parameters is common today.

- Can we achieve similar accuracy with smaller models?
- Current research: what could be done with **much** larger models?
- Instead of 10^8 parameters, what if we used **10^12**?



Sizes of top performing CNN-based architectures on ImageNet ILSVRC.

Learning Transferable Architectures for Scalable Image Recognition

Next time

Topics next class

- Famous incidents / bias in data
- Getting started with TensorFlow

Assignments

- A1 will be released next week (implement linear and deep models in TensorFlow)

Reading

- Deep Learning with Python: Chapters 1 & 2
- Deep Learning: Chapter 1
- Python Data Science Handbook: (refresh concepts as needed)