Welcome!
Applied Deep Learning, Fall 2019

Instructor

- Josh Gordon / joshua@cs.columbia.edu

TAs

- Pratik Dubal / pratik.dubal@columbia.edu
- Su Ji Park / suji.park@columbia.edu

Office hours

TBD, we'll get them sorted this week and make an announcement on courseworks.

- Su Ji Park: Tuesday 1:30pm - 3pm, Mudd CS TA room

In addition to office hours, I usually stick around after class each week to answer questions for an hour or so. If I haven't gotten back to you via email, this is the best time to catch me.

Agenda and announcements

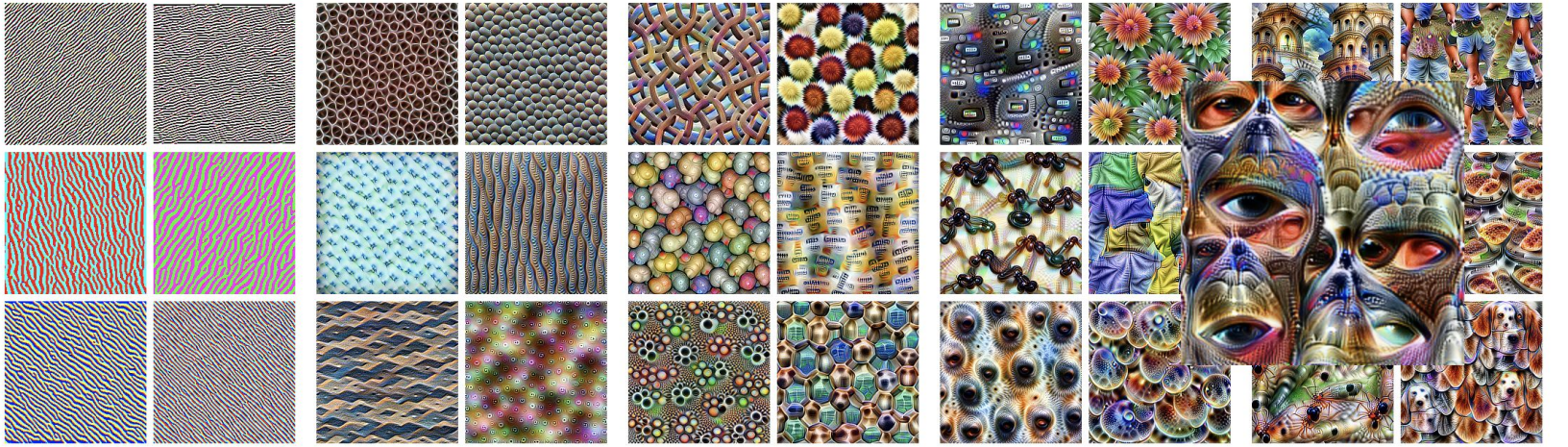
Agenda

- About this class
- What we'll cover
- What's Deep Learning?
- Intro to Colab
- Installing TensorFlow 2.0

Announcements

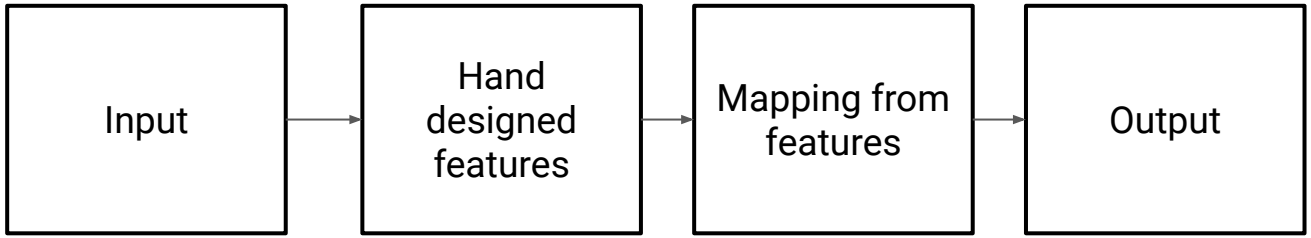
- Syllabus posted
- Assignment 1 released (due 9/26)

Representation learning is the key idea that distinguishes Deep Learning from other types of ML.

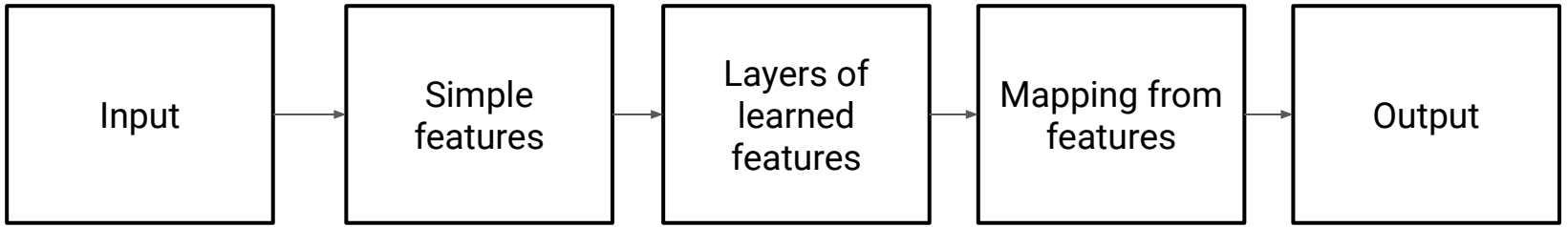


Feature Visualization from distill.pub

Another view



Classic ML



Deep Learning

About the class

Syllabus

Link on CourseWorks

https://github.com/random-forests/adl-fall-19/blob/master/syllabus.pdf

Assignment 1

You probably want to get started playing with neural networks, so let’s dive in! Just uploading this early, we’ll cover most of the concepts you need in lecture two.

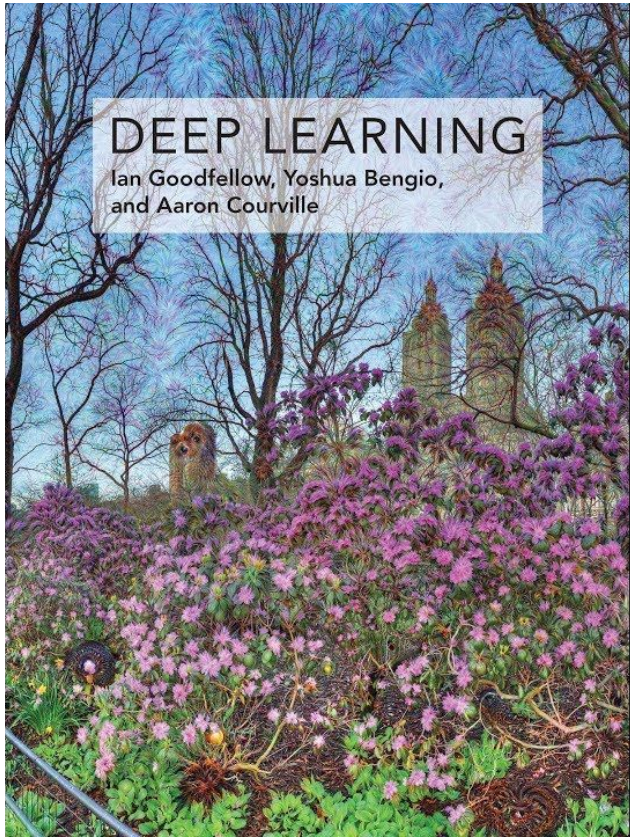
Due 9/26

Textbook #1

Available for free online

www.deeplearningbook.org

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



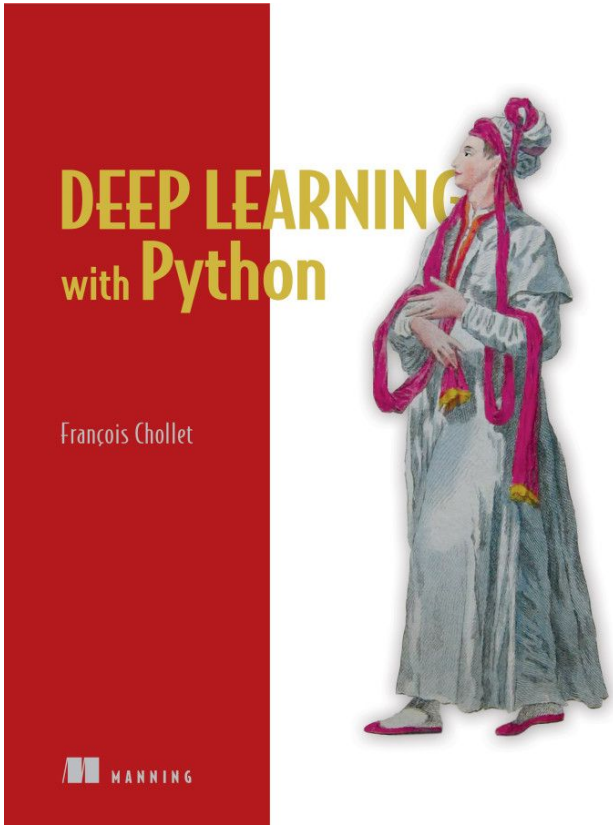
Textbook #2

This is most effective practical intro that exists today. It's light reading.

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

The e-book is around $40 (recommended):
manning.com/books/deep-learning-with-python

Francois chollet is the author of Keras.



Please bring your laptops to class

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

- You probably don’t want to hear me lecture for 2.5 hours (and my voice will go), so we’ll use time in class to work on the homework.

Format will be something like:

- Lecture / break / coding / lecture

Grading

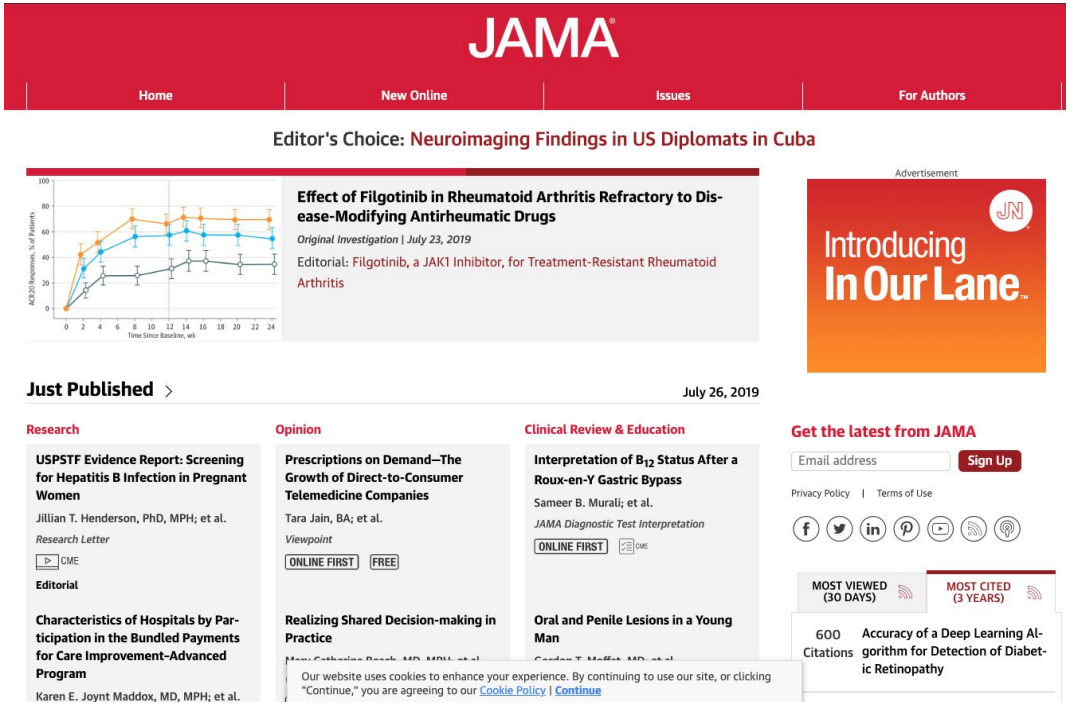
- Course project and presentation (25%).
- Five homework assignments (60%). Most will be practical (e.g., develop an image classifier to recognize landmark's on Columbia's campus), some will involve nuts and bolts of deep learning (provide your own implementation of optimizer X).
- A midterm (15%). Mostly focused on responsible AI, fairness, and applications.

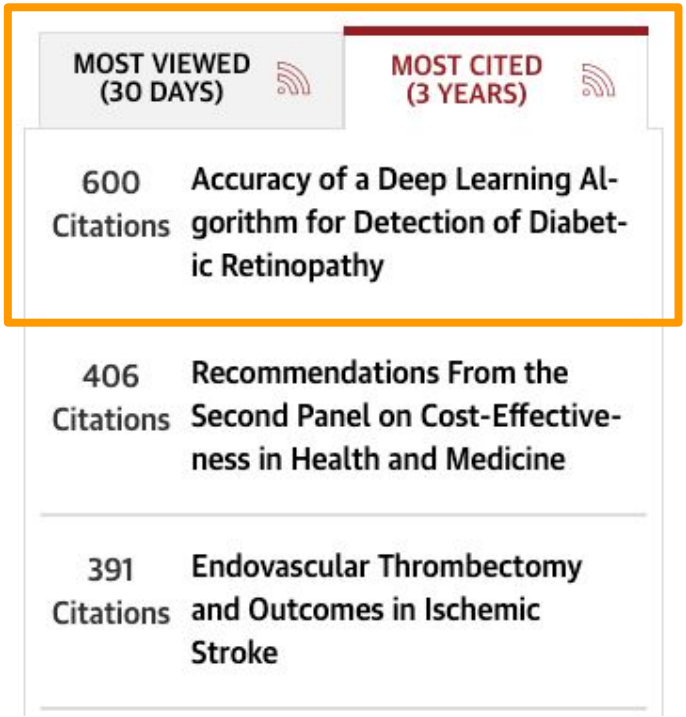
Course project

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

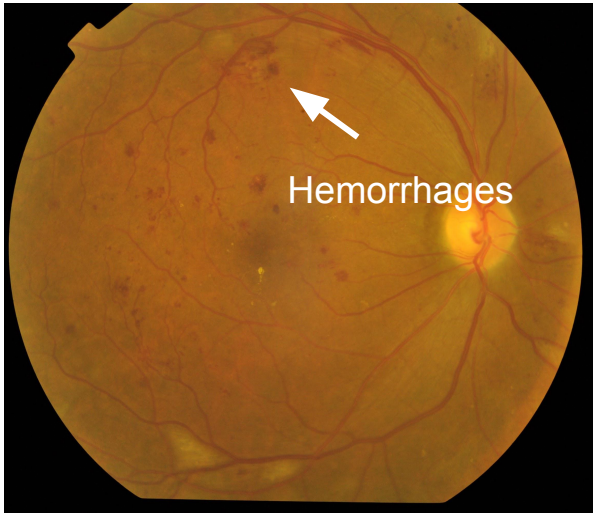
- This can be valuable to bring up in a **future interview** (having something you can present confidently is a great outcome).

- Or to **inform future study**.

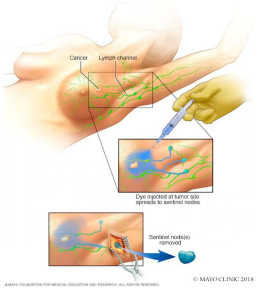




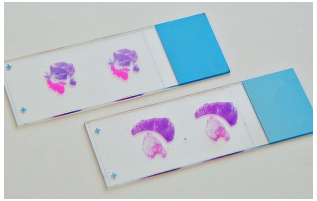
[Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs
Journal of the American Medical Association]



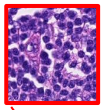
Quick discussion. Outstanding work that involved very little coding. What did it take to succeed?



biopsy

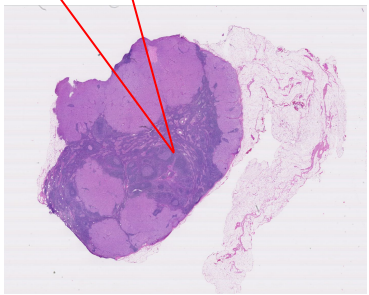


Preparation



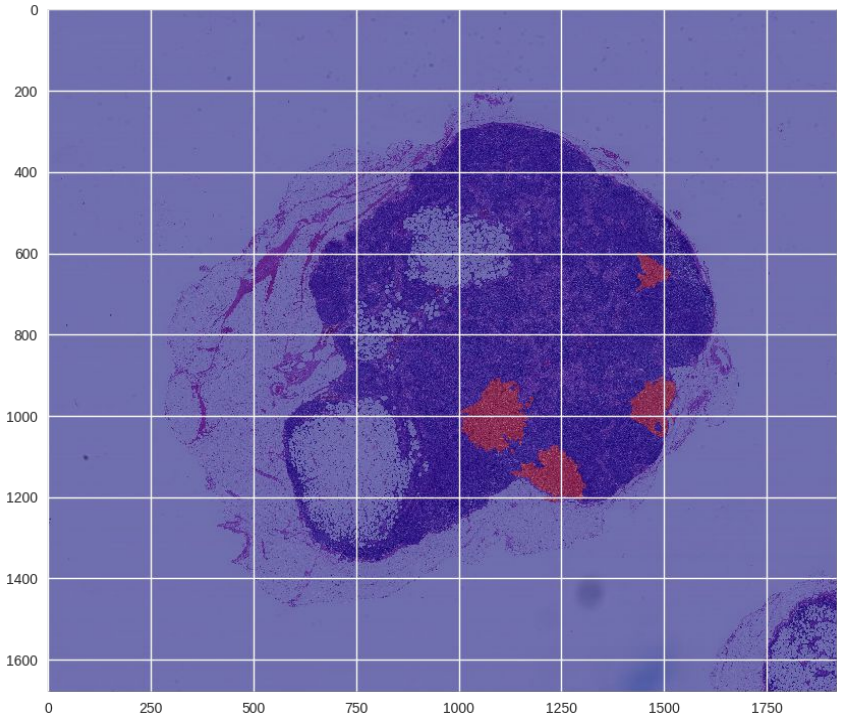
Diagnosis → Treatment plan

healthy vs tumor



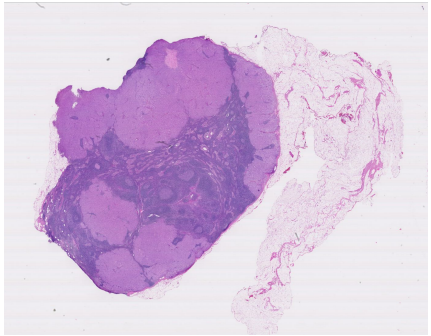
Visual inspection

[Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs](#)
[Detecting Cancer Metastases on Gigapixel Pathology Images](#). [CAMELYON16 and '17 Challenges](#), [Sentinel node biopsy](#)

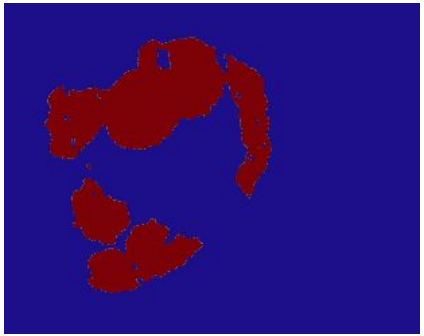

0
200
400
600
800
1000
1200
1400
1600
0
250
500
750
1000
1250
1500
1750


0
200
400
600
800
1000
1200
1400
1600
0
250
500
750
1000
1250
1500
1750

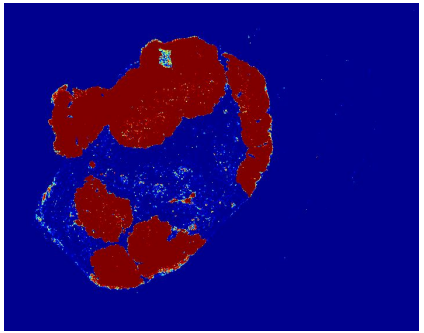
Develop a tool to assist physicians



Biopsy image



Ground truth (from pathologist)



Your model's predictions

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 / academic jobs vs private sector.

Waitlist

If you would like to be added as an observer to courseworks (so you can access the materials), please add your UNI to this form.

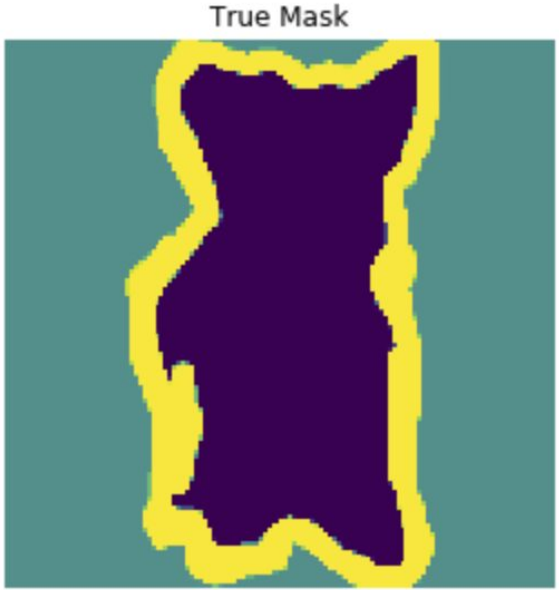
If you are unable to add the class, you are welcome to sit any lectures you like (no need to sign up or ask permission).

http://bit.ly/2kuiz3H

What we'll cover

Image classification, segmentation, object detection, transfer learning


Monet - Sunflowers: 0.992



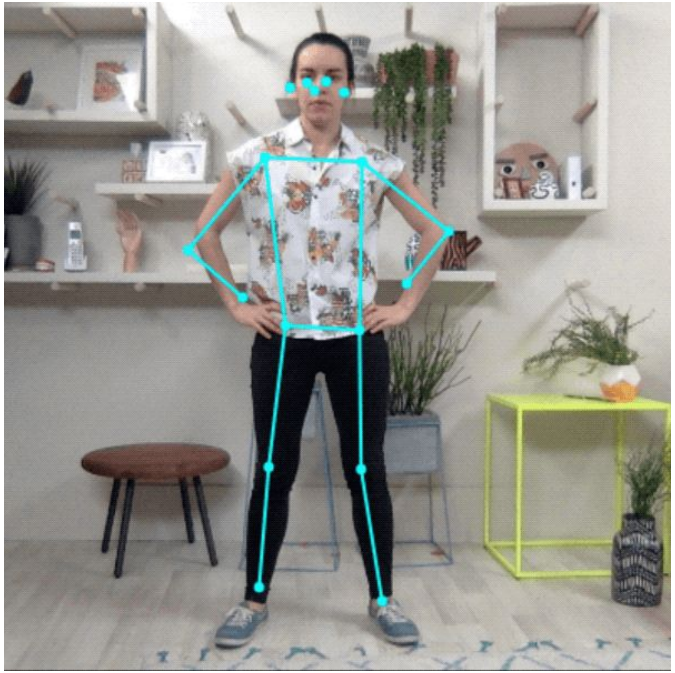
tensorflow.org/beta/tutorials/images/segmentation

Beyond Python

Machine Learning

happens in Python,

right?



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

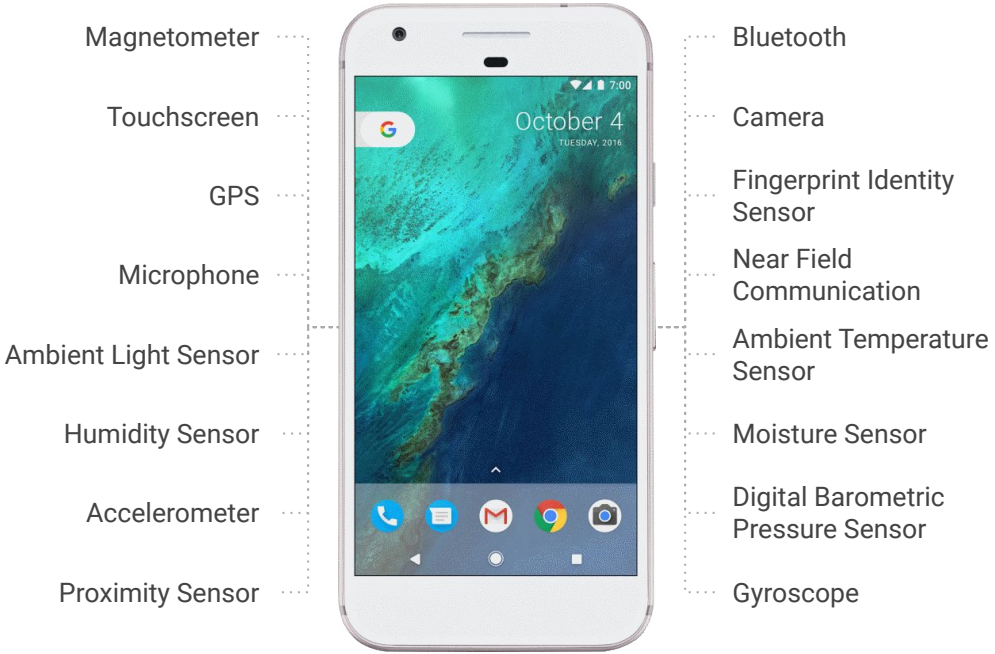
Demo

PoseNet and BodyPix

bit.ly/pose-net

bit.ly/body-pix

Sensors


Magnetometer
Touchscreen
GPS
Microphone
Ambient Light Sensor
Humidity Sensor
Accelerometer
Proximity Sensor
Bluetooth
Camera
Fingerprint Identity Sensor
Near Field Communication
Ambient Temperature Sensor
Moisture Sensor
Digital Barometric Pressure Sensor
Gyroscope
7:00
October 4
TUESDAY, 2016

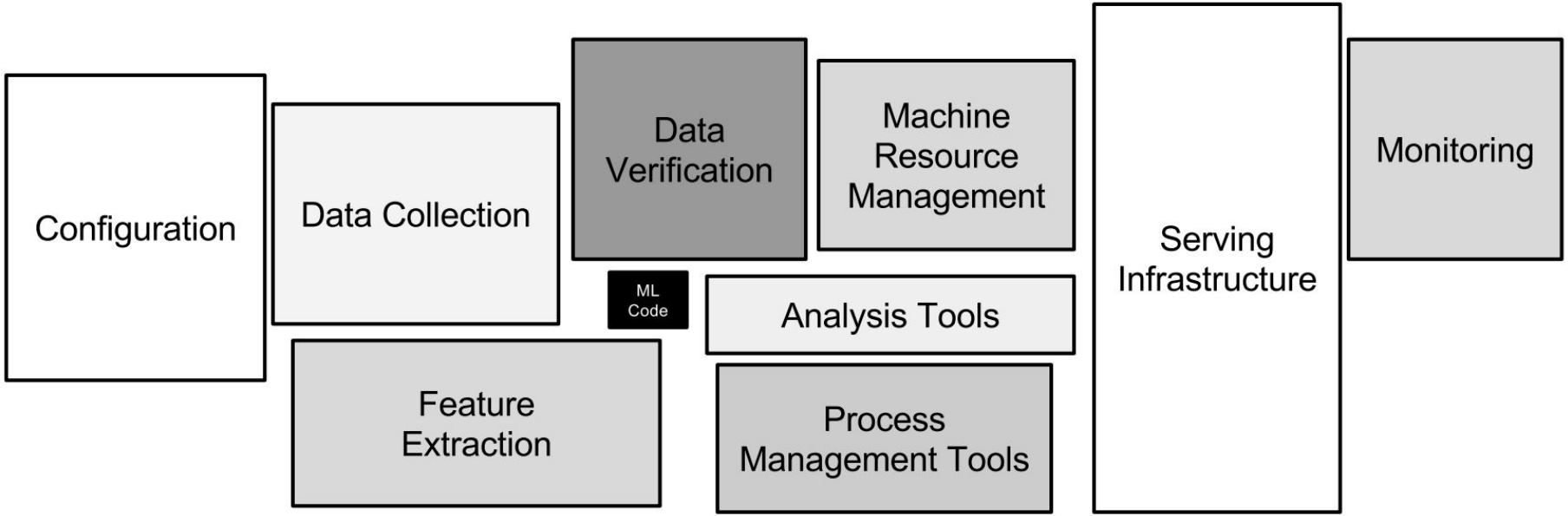
Testing ML Systems

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.

Defining and training an accurate model.

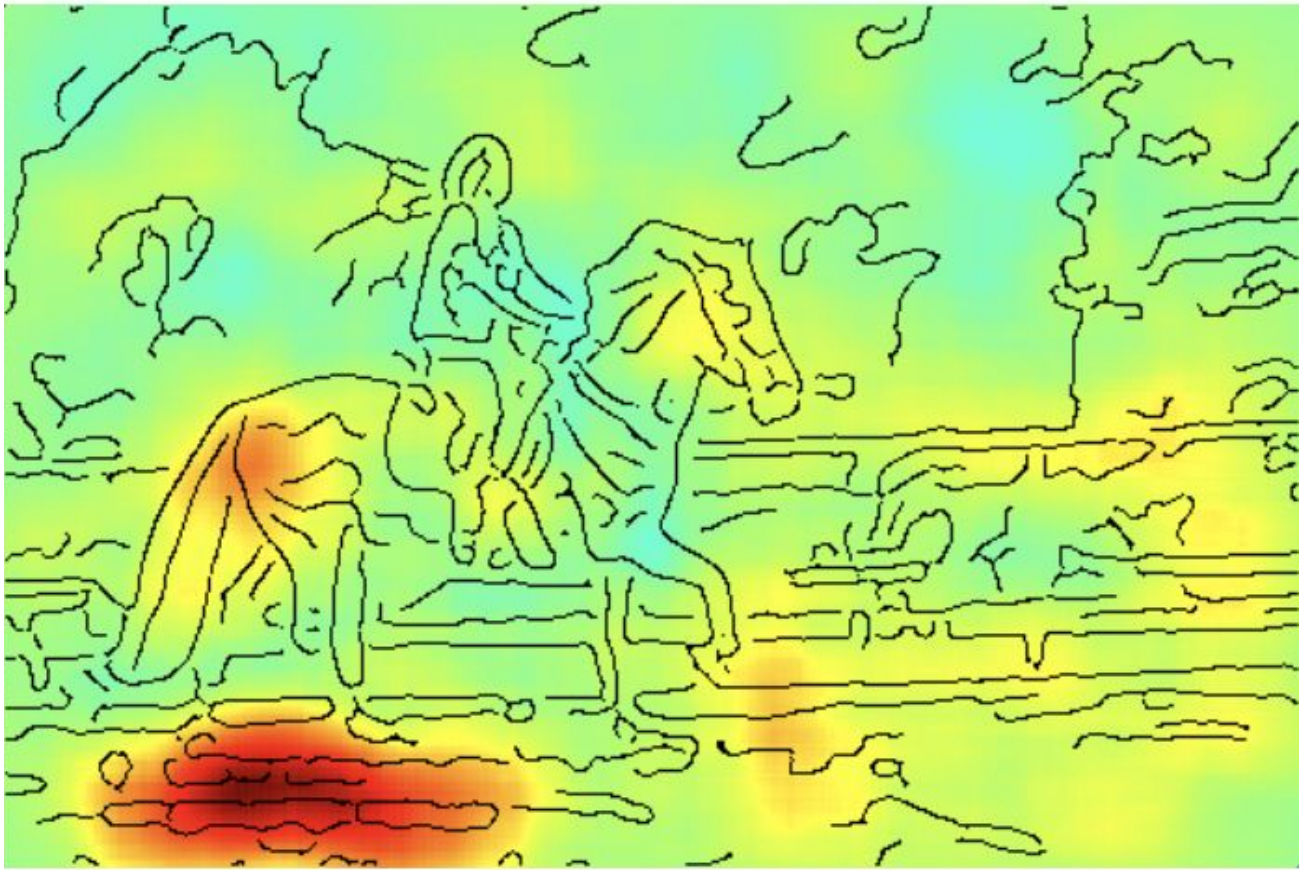
[Hidden Technical Debt in Machine Learning Systems]

Only a small fraction of real-world ML systems is composed of the code for the model, as shown by the small black box in the middle.



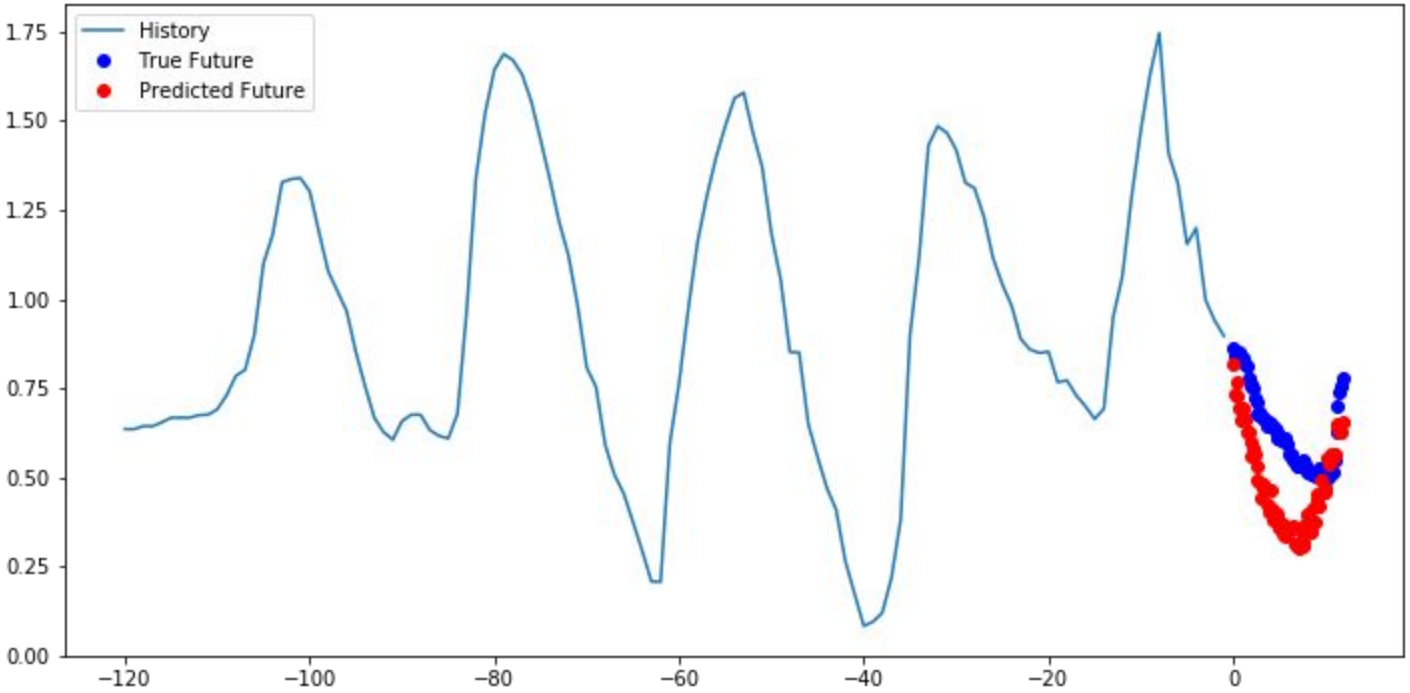
[Hidden Technical Debt in Machine Learning Systems]

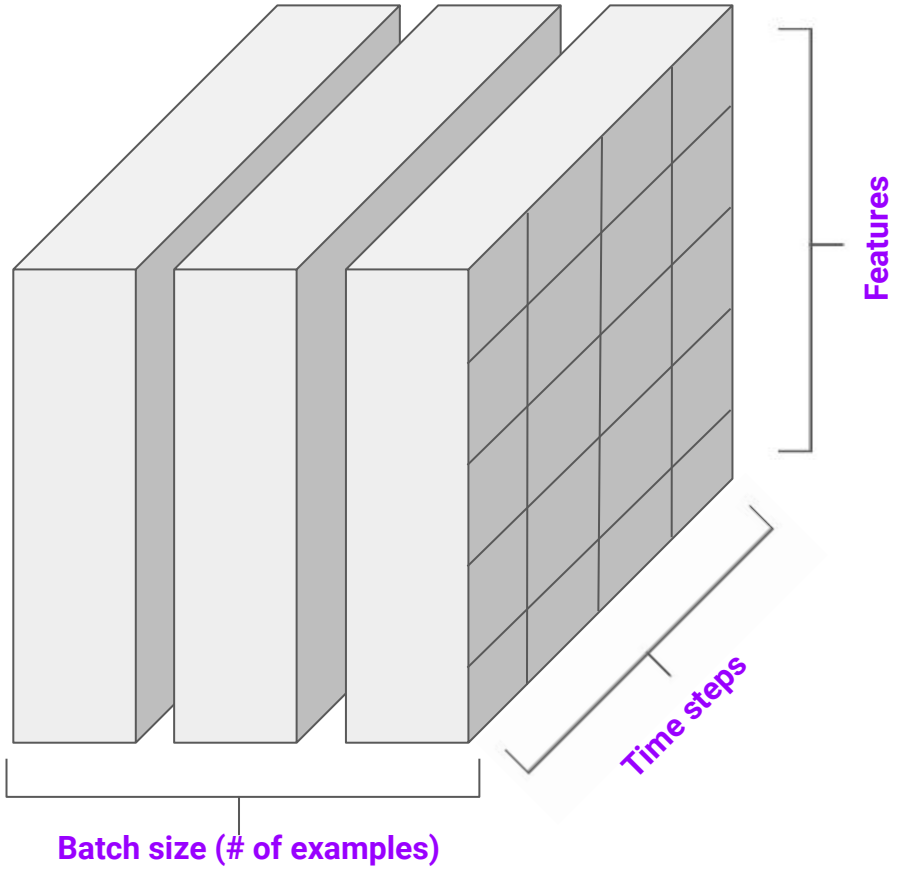
Why might this region be important?



Analyzing Classifiers: Fisher Vectors and Deep Neural Networks (not on our reading list, just an FYI with a great example).

Working with sequences


History
True Future
Predicted Future
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−120
−100
−80
−60
−40
−20
0

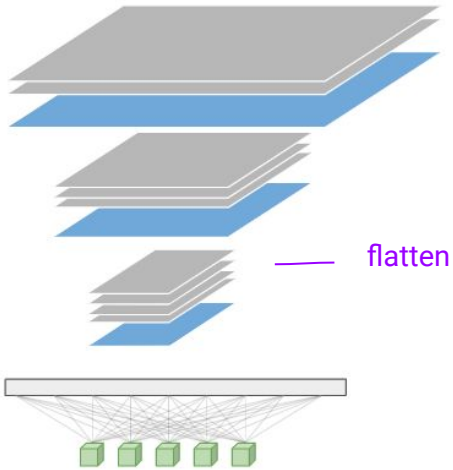



```
model = Sequential()
model.add(LSTM(32,input_shape=(5,4))
```

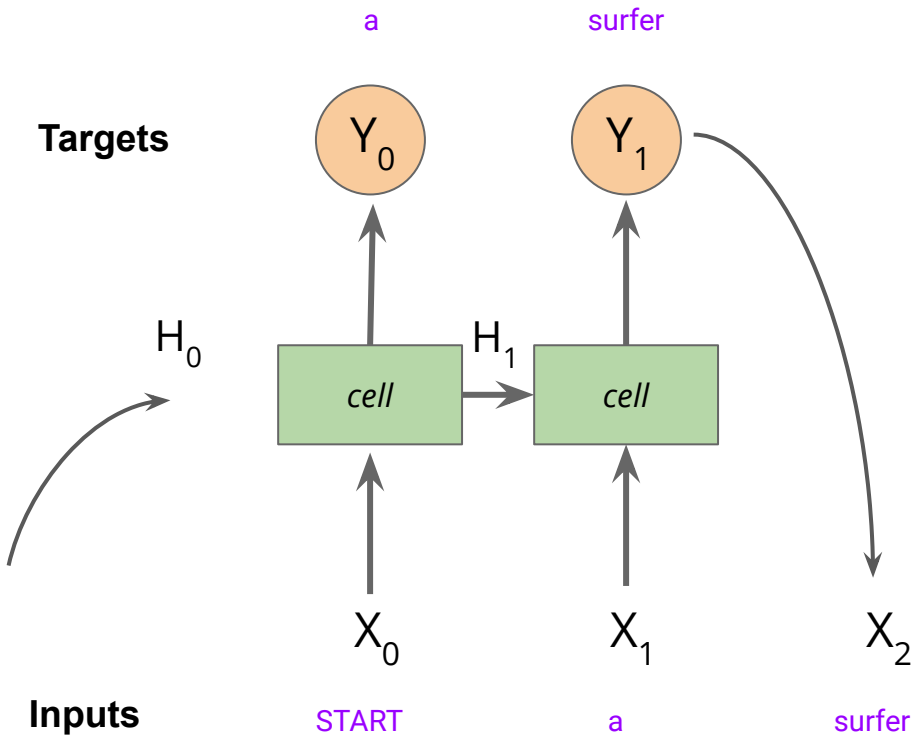
32 units, 5 timesteps, 4 features per timestep (batch size is inferred)

	T	P	H	D
12pm				
1pm				
2pm				
3pm				
5pm				

keras.io/layers/recurrent/



During inference



GANs

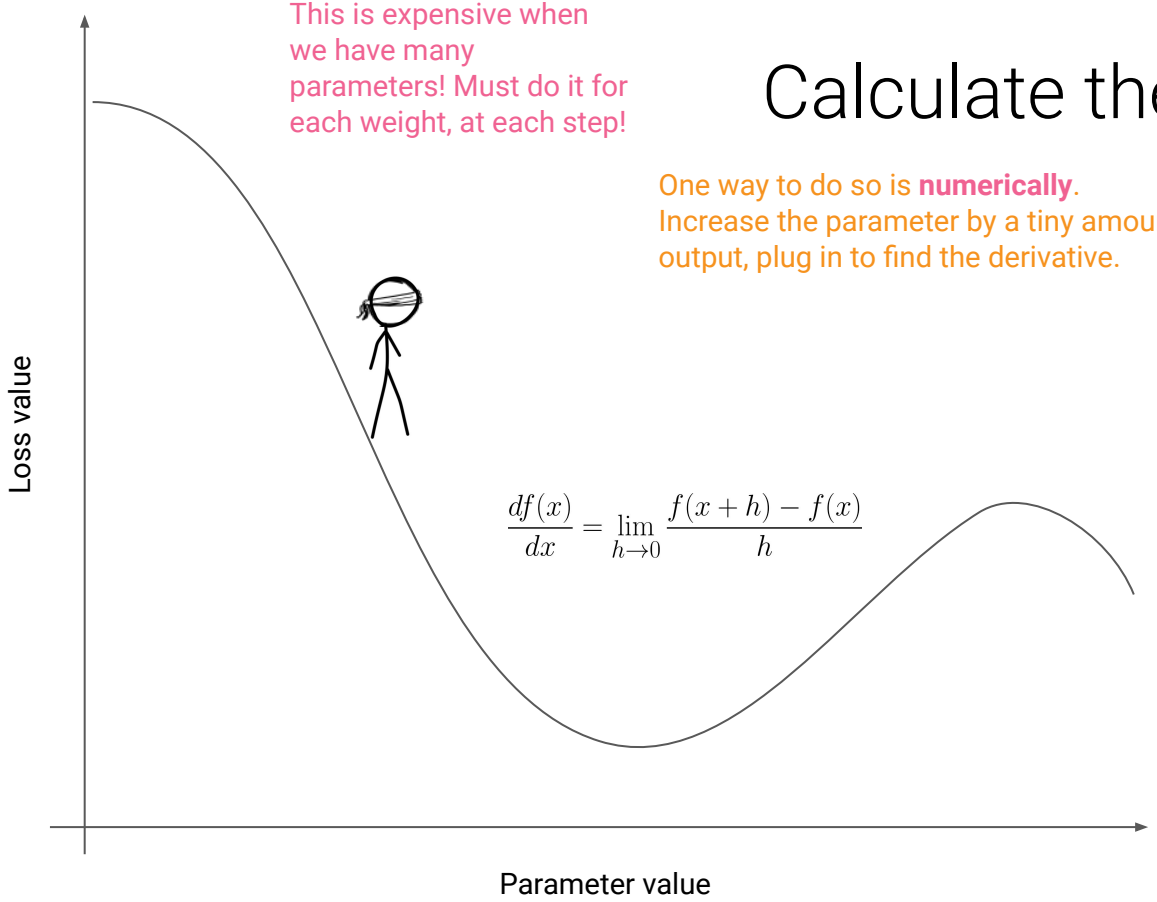
Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

Optimization

Calculate the gradient

This is expensive when we have many parameters! Must do it for each weight, at each step!

One way to do so is **numerically**.
Increase the parameter by a tiny amount, compute function output, plug in to find the derivative.

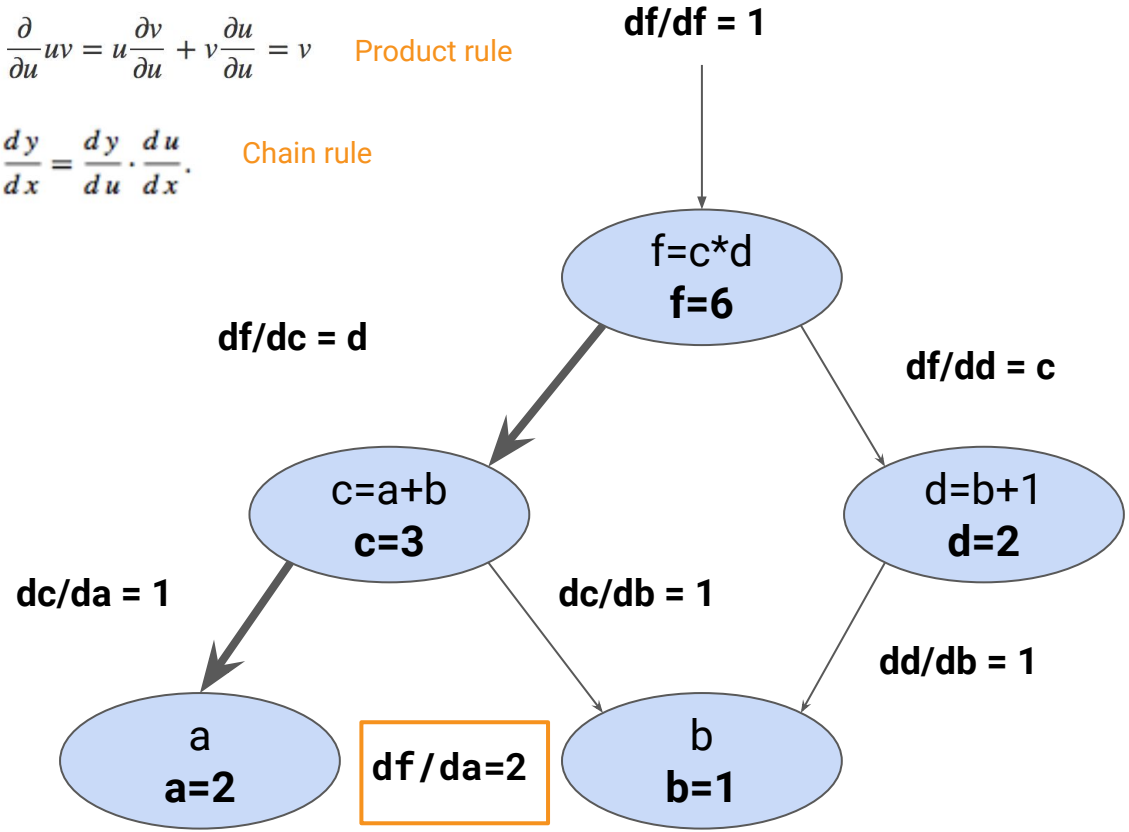


$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$$ Sum rule

$$\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$$ Product rule

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}.$$ Chain rule



Now we can compute the gradient as the product along paths (another way of thinking about the chain rule!)

```
df/da = df/df * df/dc * dc/da
      = 1 * d * 1
      = 1 * 2 * 1
      = 2
```

Various layers and building blocks

Two images and three classes

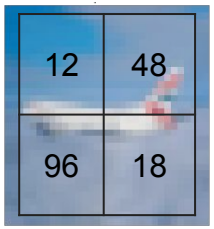


Image 1

Image 2

N x D

1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5

W

Weights

D x batch_size

12	4
48	18
96	2
18	96

x

Inputs

+

N x 1

0.5
1.2
0.2

b

Bias

=

N x batch_size

Image 1	Image 2	
130.1	131.7	Plane
-11.4	-71.7	Car
12.8	64.8	Truck

Output

Scores

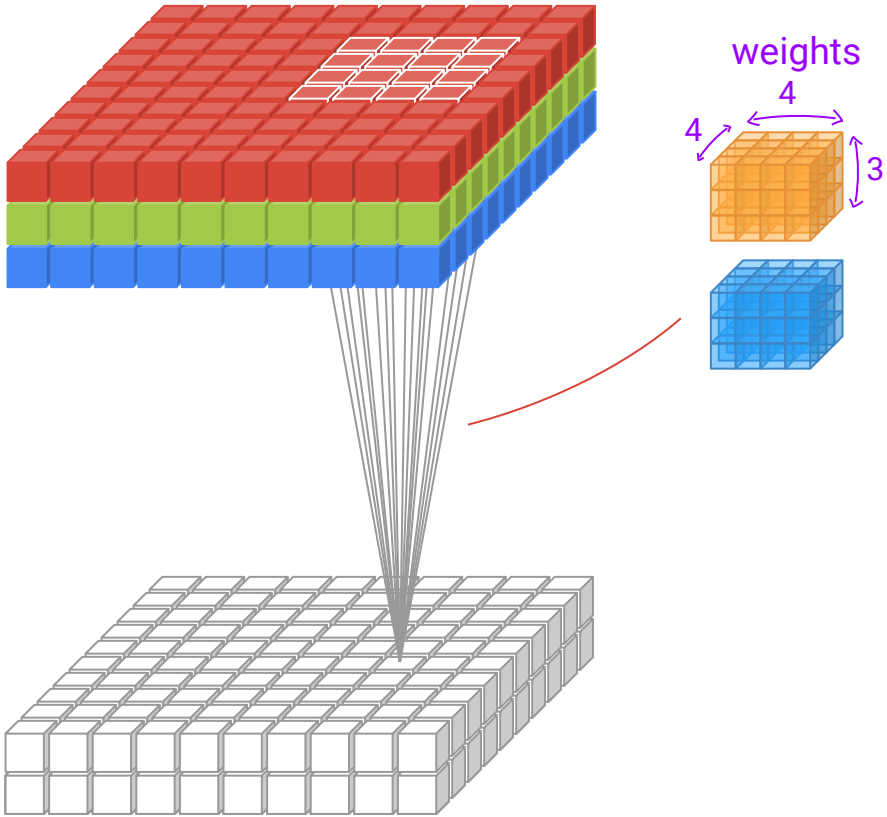
Convolution example

-1	-1	-1
-1	8	-1
-1	-1	-1

Notes
Edge detection intuition: dot product of the filter with a region of the image will be zero if all the pixels around the border have the same value as the center.

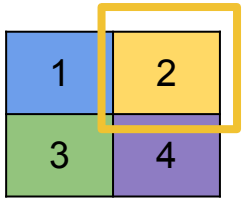
Does anyone know who this is?

Launched the Chandra X-ray Observatory, in 1999

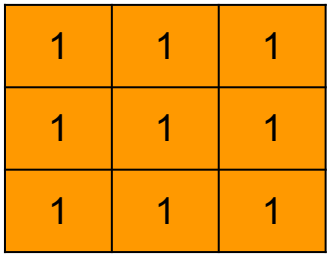


More filters, more output channels.

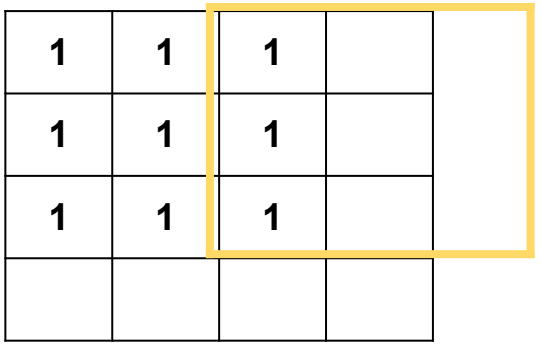
```
layer = Conv2DTranspose(filters=1,kernel_size=3,
                        strides=(2,2), padding='same',
                        kernel_initializer=ones)
```



Input image (2x2)



Filter (3x3), learned weights - initialized to one's for our example

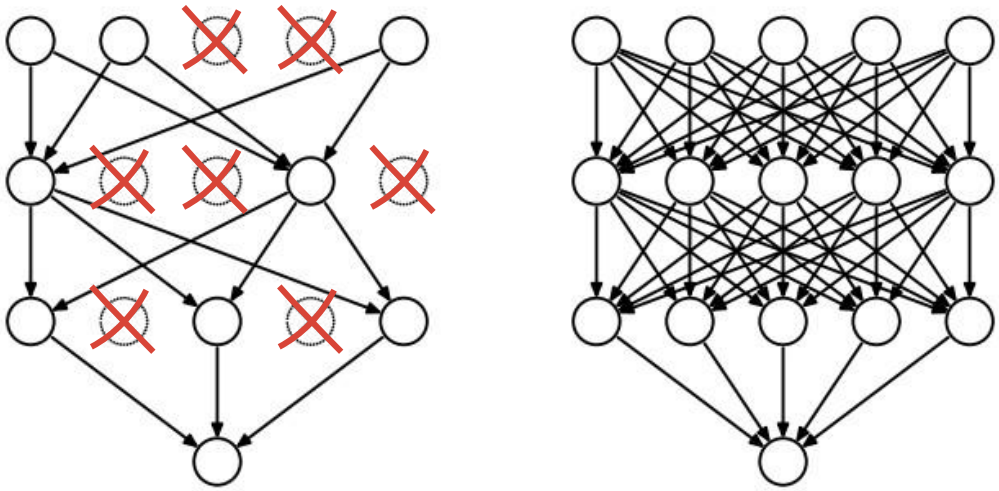


Output image (4x4)

[A guide to convolution arithmetic for deep learning] ~ Probably more than you ever wanted to read about convolution.

Dropout

Note: used to drop neurons in hidden layers (not usually inputs to the network as shown in the diagram).



Dropout rate is the fraction of the activations that are zeroed out; it's usually set between 0.2 and 0.5 (left).

No activations are dropped at testing time (right).

Quick discussion: Does anyone who hasn't seen this before have an idea why it might work?

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Batch norm

Features

Batch (examples)

	F1	F2	F3	F4
E1	2	1	3	2
E2	4	1	0	4
E3	3	1	3	0
Mean	3.0	1.0	2.0	2.0
Std	0.81	0.0	1.41	1.63

Layer norm

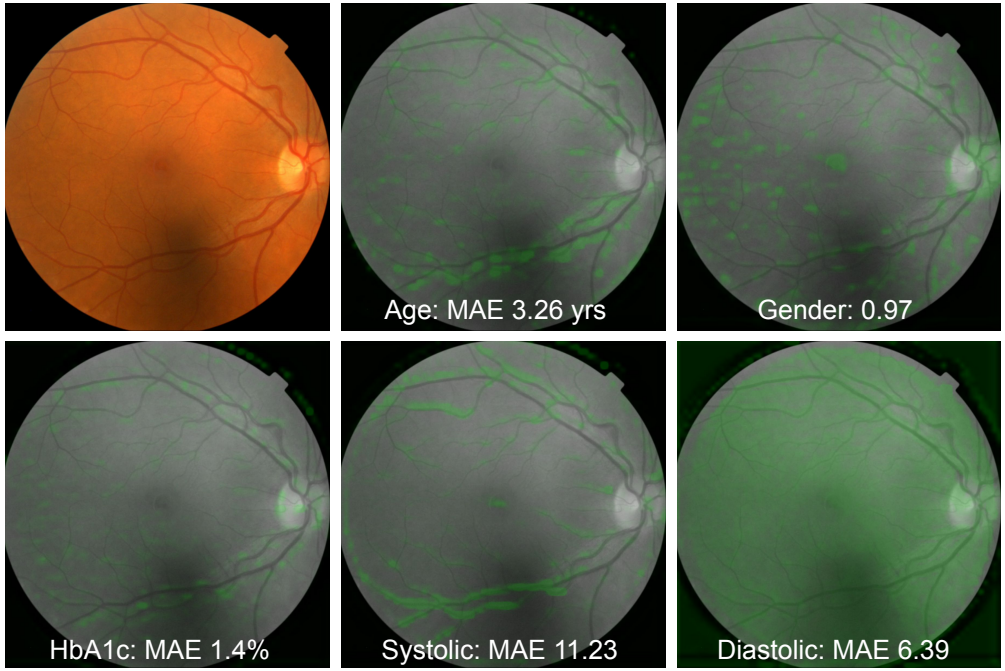
	F1	F2	F3	F4	Mean	Std
E1	2	1	3	2	2.0	0.71
E2	4	1	0	4	2.25	1.78
E3	3	1	3	0	1.75	1.30

Layer Normalization

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Medical imaging

Supporting basic science



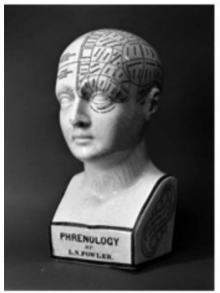
- Can we predict cardiovascular risk? If so, this is a potentially non-invasive way of doing so?
- Can we identify features used by the model to predict each task?
- (Could these potentially assist researchers doing basic science?)

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

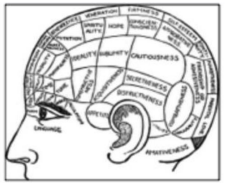
Responsible AI

Echos of phrenology and physiognomy are sadly back.

Cover of the economist from **Sep 9th 2017**

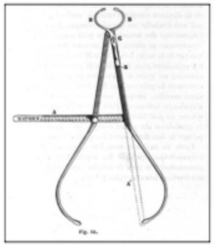


Fowler head



Phrenology chart

Measuring tape



Calipers

Palpation

From the paper

“We show that faces contain much more information about sexual orientation than can be perceived and interpreted by the human brain.

We used deep neural networks to extract features from 35,326 facial images [...] these features were entered into a logistic regression aimed at classifying sexual orientation [...]

Those findings **advance our understanding of the origins of sexual orientation** and the limits of human perception.”

Deep neural networks are more accurate than humans at detecting sexual orientation from facial images.

Art


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

Setting up your programming environment

About colaboratory colab.research.google.com

A free, cloud-based Jupyter environment.

- Recommended, but not required (anything you can do in Colab works in open source Jupyter).
- Includes a free GPU (currently a Tesla T4).
- Zero setup.
- Colab connects to VM on Google Cloud (you have root access)

Intended for interactive use. VMs are transient.

- To store files long term, either connect Colab to Google Drive (free), or use a cloud solution from your favorite provider.

Best practice: write your notebooks so they run end-to-end with no user intervention (installing any libraries needed with !pip commands at the top, and downloading any datasets needed with !wget or similar).

Tips and tricks

- Enabling the GPU.
- Uploading and downloading Jupyter notebooks.
- Code snippets.
- Using Google Drive as a filesystem to save and load files (**mounting it at "/drive/"**).
- Using the file browser.
- Saving notebooks with output.
- Using cloud storage buckets.
- Colab browser extension.
- Resetting your runtime.
- Resetting your VM to a clean state (removing any software installed or files downloaded).

Colab also has a Swift VM you can use tensorflow.org/swift (work in progress)

TensorFlow 2.0

TensorFlow 2.0

- An open source Deep Learning library released by Google in 2015
- >1800 contributors worldwide

Version 2.0 (currently in beta)

- Much easier to use

Documentation and examples

- tensorflow.org/beta

Tip: ignore everything else on the website.

Note: TF1 is installed in Colab by default. You will need to pip install TF2.

TensorFlow 2.0

Similar to NumPy, with:

- GPU support
- Autodiff
- Distributed training
- JIT compilation
- A portable format (train in Python on Mac, deploy on iOS using Swift, or in a browser using JavaScript)

Write models in Python, JavaScript or Swift (and run anywhere).

API doc: tensorflow.org/versions/r2.0/api_docs/python/tf
Note: make sure you're looking at version 2.0 (the website still defaults to 1.x)

More on the above tomorrow

TF2 vs TF1

Notes

- You do not need any knowledge of TF1 to learn TF2.
- You should not learn TF1 in order to learn TF2.

You can find all the latest tutorials and guides for TensorFlow 2.0 beta here. Note: knowledge of TF1 is useful independently (there are thousands of papers and millions of LOC using that framework).

Installing TF2

```
# GPU
!pip install tensorflow-gpu==2.0.0-beta1



# CPU
!pip install tensorflow==2.0.0-beta1
```

(In Colab, you may need to use runtime -> restart after installing new software).

```
import tensorflow as tf
print(tf.__version__) # 2.0.0-beta1
```

Nightly is available too, but best bet: stick with a named release for stability.

Special case if you’re working in Colab

```
try:
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf

# TensorFlow 2.x selected.
```

This will enable the latest stable version of 2.0 (Colab has 1.x installed by default at the moment). Much faster than !pip installing it.

TF2 feels like NumPy

```
import tensorflow as tf
print(tf.__version__) # 2.0.0-beta1

x = tf.constant(1)
y = tf.constant(2)
z = x + y

print(z) # tf.Tensor(3, shape=(), dtype=int32)
```

TF1 was different: Build a graph, then run it.

```
import tensorflow as tf # 1.14.0
print(tf.__version__)

x = tf.constant(1)
y = tf.constant(2)
z = tf.add(x, y)

print(z) # Tensor("Add:0", shape=(), dtype=int32)

with tf.Session() as sess:
  print(sess.run(x)) # 3
```

What's in a name?

- **Tensor** (fancy word for n-dimensional array)
- **Flow** (dataflow graph: think, the abstraction we use to do backprop, and/or to describe computation in a machine and language independent format)

Keras

Keras

- A library of building blocks (layers, activations, optimizers, etc).
- Famous for **ease of use** (while not being a black box).

Reference implementation at www.keras.io

```
!pip install Keras
import keras
```

Installed in Colab by default, which is a good thing, but makes it complicated to use Keras layers inside TensorFlow.

Notes

Keras is a fine place to begin your deep learning journey. Everything you learn at keras.io will work in TensorFlow 2.0, just by changing an import.

Documentation and examples

- Doc
- Examples

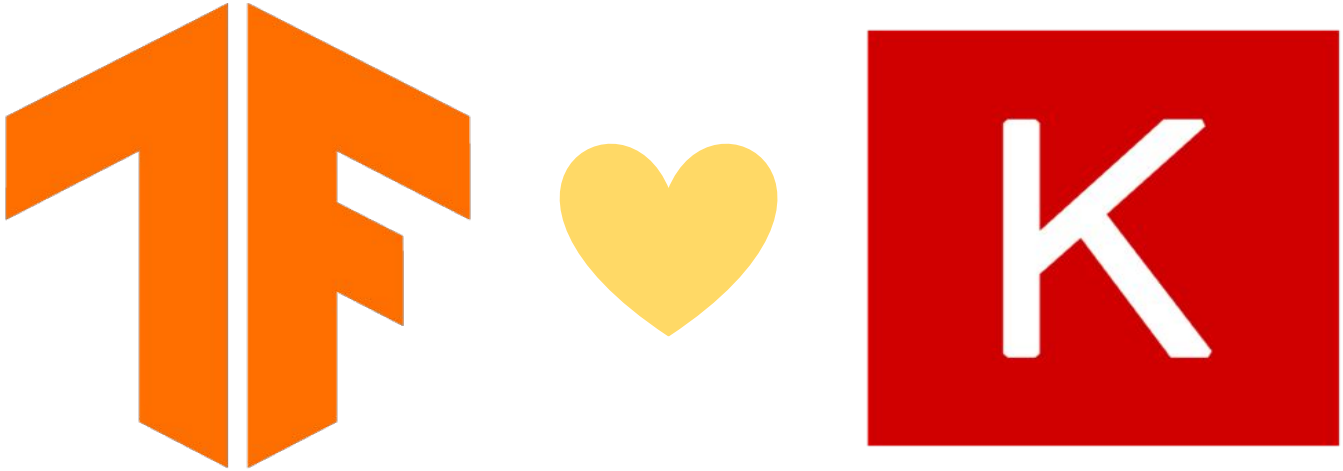
```
>>> import keras

Using TensorFlow backend.
```

If you see this message, you have imported the reference implementation.

Keras automatically installs a backend that it uses for computation (running operations on GPUs, calculating gradients, etc). By default this is TensorFlow, others are available as well.

tf.keras (Keras inside TensorFlow)

Built-in to TensorFlow 2.0



Importing tf.keras

If you want to use **tf.keras** and see the message “Using TensorFlow Backend”, you have accidentally imported Keras (which is installed by default on Colab) from outside of TensorFlow.

Example

```
# !pip install tensorflow==2.0.0-beta1, then

>>> from tensorflow.keras import layers # Right

>>> from keras import layers # Oops

Using TensorFlow backend. # You shouldn't see this
```

When in doubt, copy the imports from one of the tutorials on tensorflow.org/beta

Notes

A **superset** of the reference implementation. Built-in to TensorFlow 2.0 (no need to install Keras separately).

Documentation and examples

- **Tutorials**: tensorflow.org/beta
- **Guide**: tensorflow.org/beta/guide/keras/

```
!pip install tensorflow==2.0.0-beta1
from tensorflow import keras
```

I'd recommend the examples you find on tensorflow.org/beta over other resources (they are better maintained and most of them are carefully reviewed).

tf.keras adds a bunch of stuff, including… model subclassing (Chainer / PyTorch style model building), custom training loops using a GradientTape, a collection of distributed training strategies, support for TensorFlow.js, Android, iOS, etc.

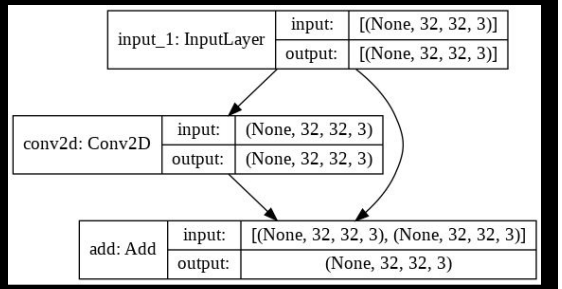
Three model building styles

Sequential

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

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

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Functional

```
inputs = keras.Input(shape=(32, 32, 3))

# Note: the output dimensions of this layer match the input
y = layers.Conv2D(3, (3, 3), activation='relu', padding='same')(inputs)

outputs = layers.add([inputs, y])
model = keras.Model(inputs, outputs)
keras.utils.plot_model(model, 'skip_connection.png', show_shapes=True)
```



Subclassing

```
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model')
    self.dense_1 = layers.Dense(32, activation='relu')
    self.dense_2 = layers.Dense(num_classes, activation='sigmoid')

  def call(self, inputs):
    # Define your forward pass here
    x = self.dense_1(inputs)
    return self.dense_2(x)
```

Two training styles

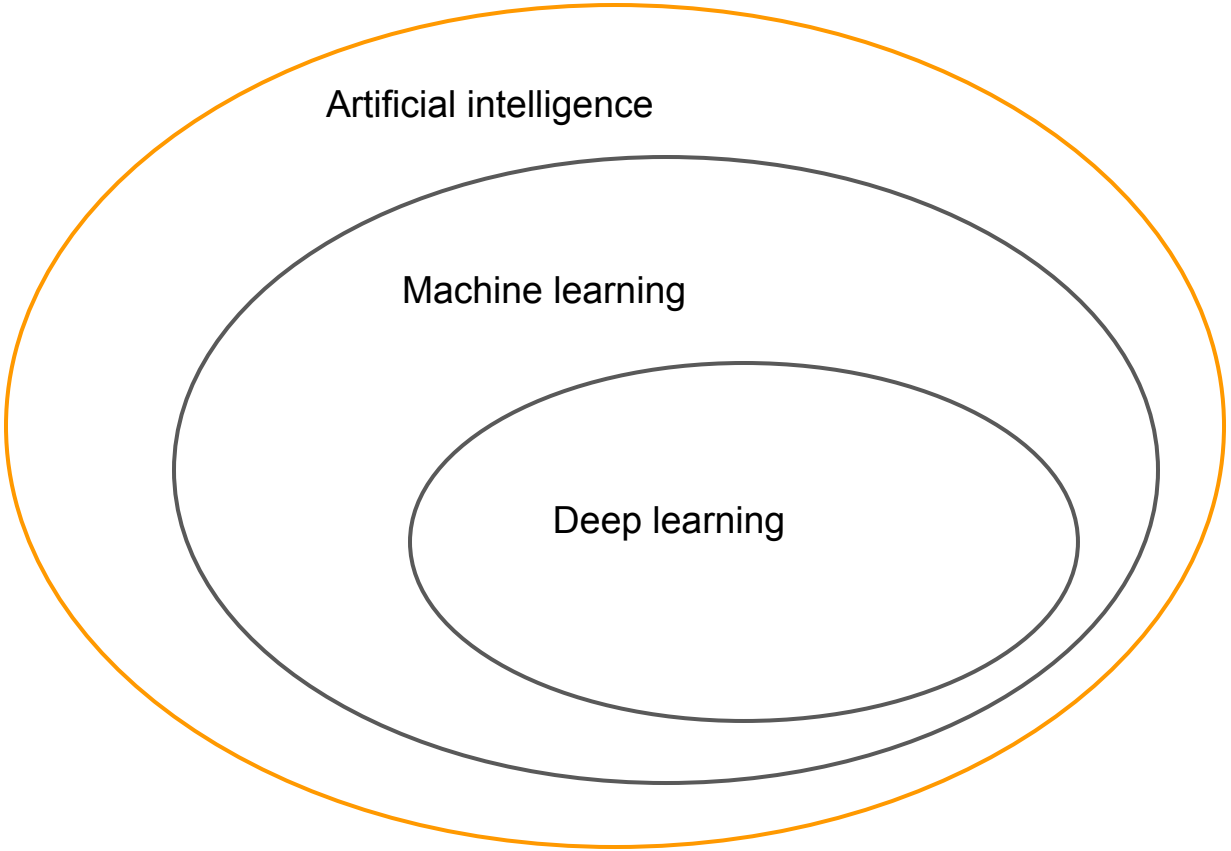
Built-in

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

Custom

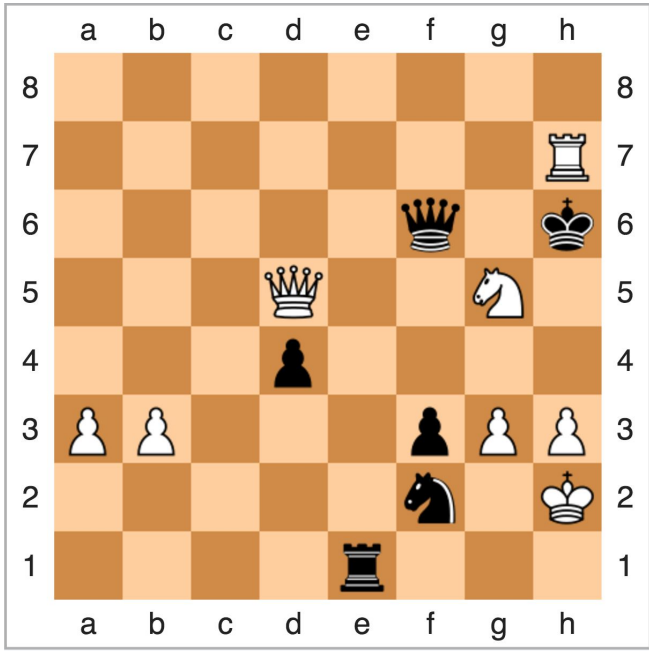
```
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)

grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

A bit of history



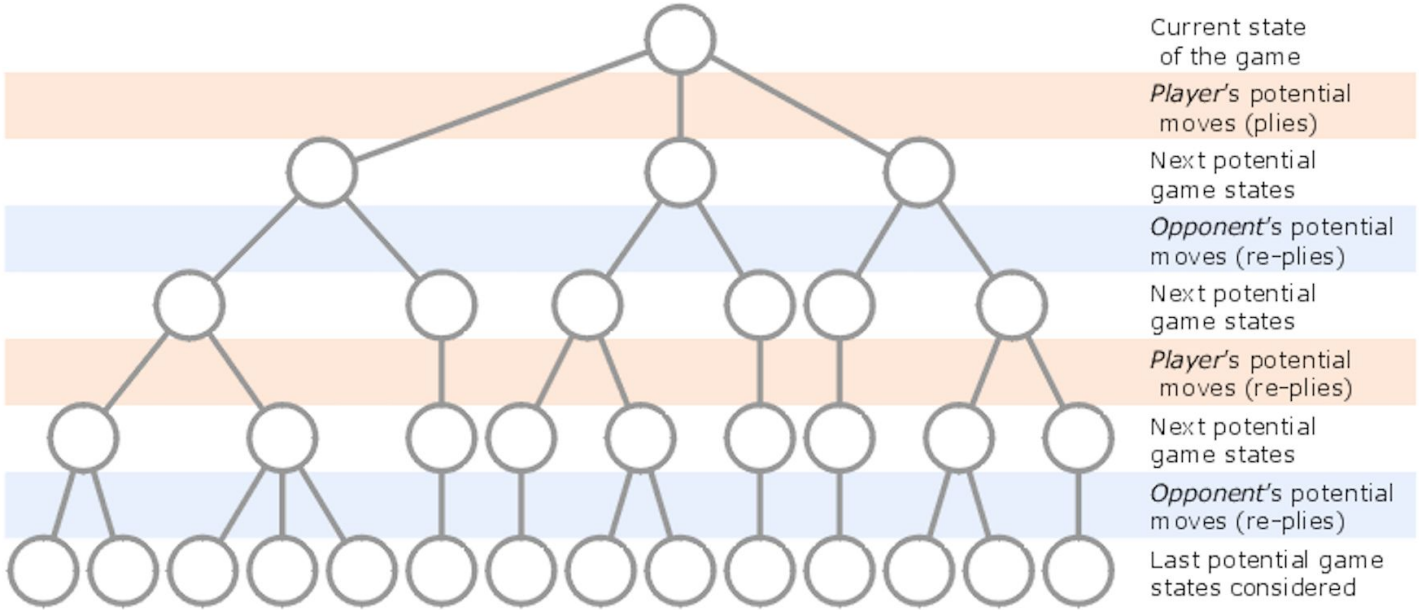
Early days of AI: Problems that are **difficult** for people, but **easy** for computers.

IBM’s Deep Blue vs Garry Kasparov (1997)



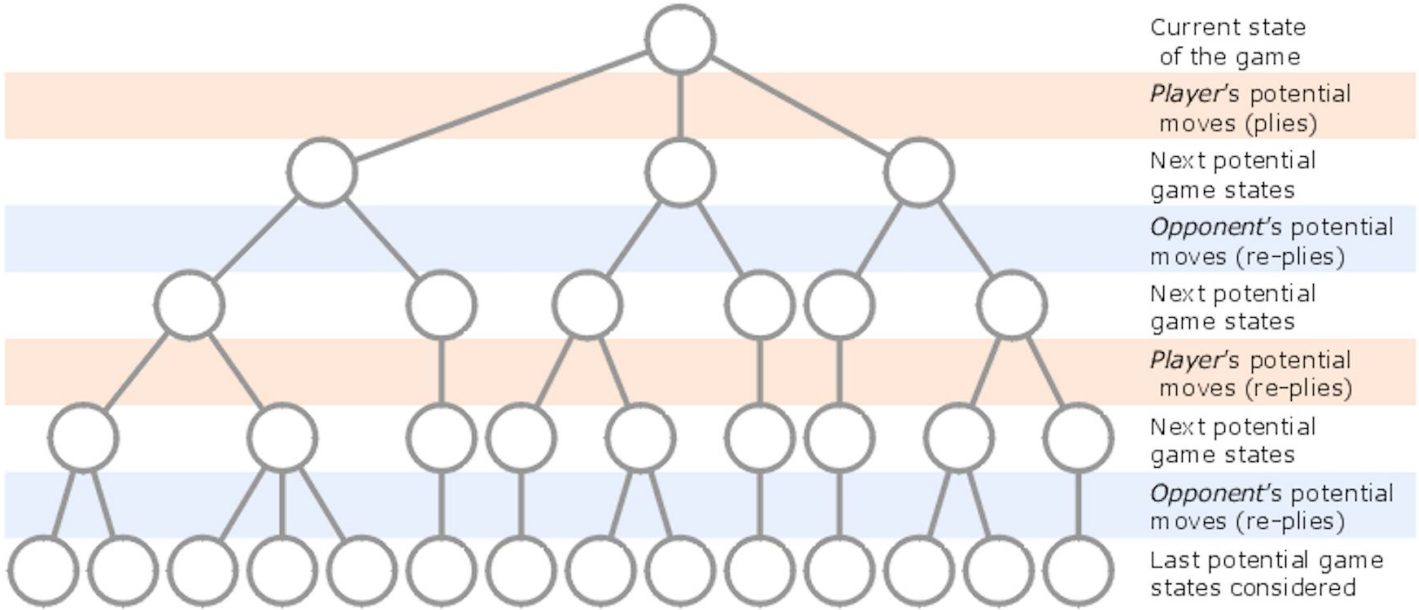
[The story of Deep Blue]

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



Programming a Computer for Playing Chess

Claude Shannon 1950(!)



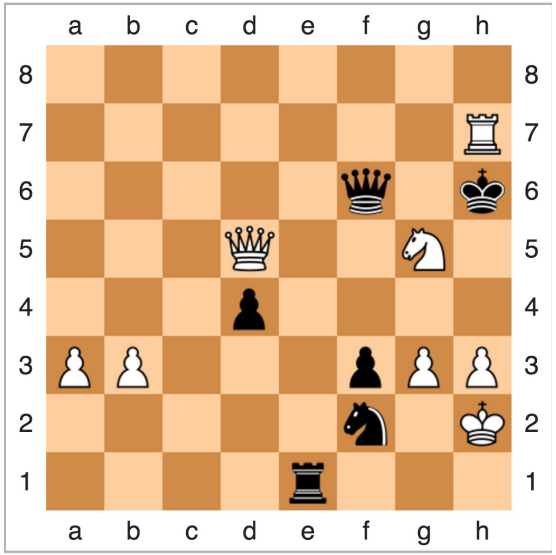
Programming a Computer for Playing Chess

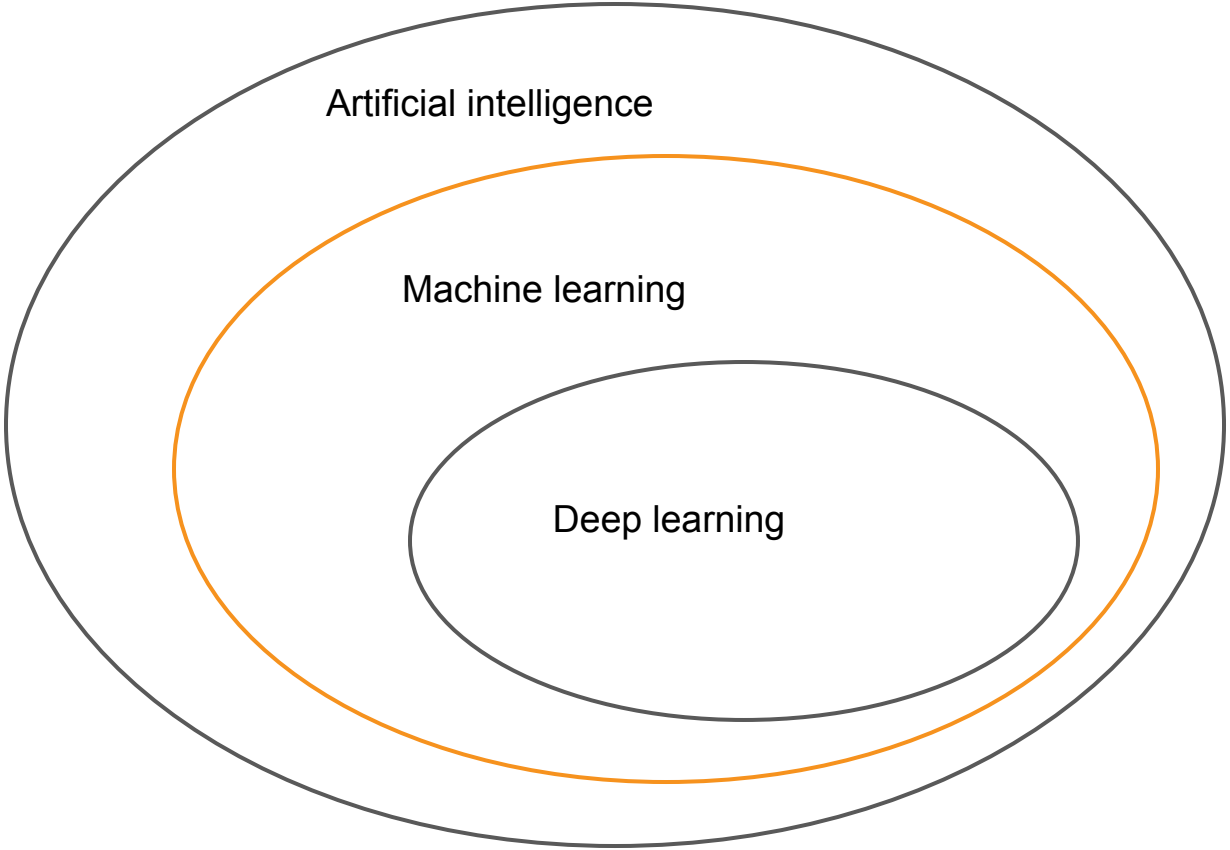
A tremendous accomplishment, but not due to the difficulty of the representation.

Rules of chess 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** evaluation function.




Artificial intelligence
Machine learning
Deep learning

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

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

From the bag of models: kNN, Decision Trees, Logistic Regression, SVMs, Random Forests, Neural networks, etc.

- Most Kaggle competitions involving **perception** are won by **neural networks.**

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

Quick discussion: Why? What are the advantages of tree-based models?

- Most Kaggle competitions involving **perception** are won by **neural networks.**

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

Quick discussion: **Why?** What are the advantages of tree-based models?

Structured data
A small number of informative features (think: demographic data).

- Imagine how **wide and deep** a decision tree trained on pixels would be...

Classifying an image with a tree...

If pixel 277 is greater than 185 and pixel 485 is less than or equal to 177 then….

- Most Kaggle competitions involving **perception** are won by **neural networks.**

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

Quick discussion: Why? **What are the advantages of tree-based models?**

You can interpret predictions and easily translate them into business logic.

Neural networks are not entirely a black box (particularly when working with images).

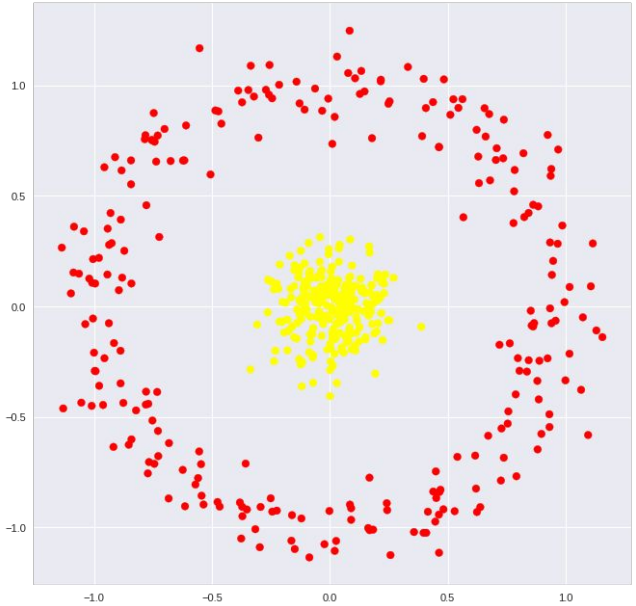
When is Deep Learning suitable?

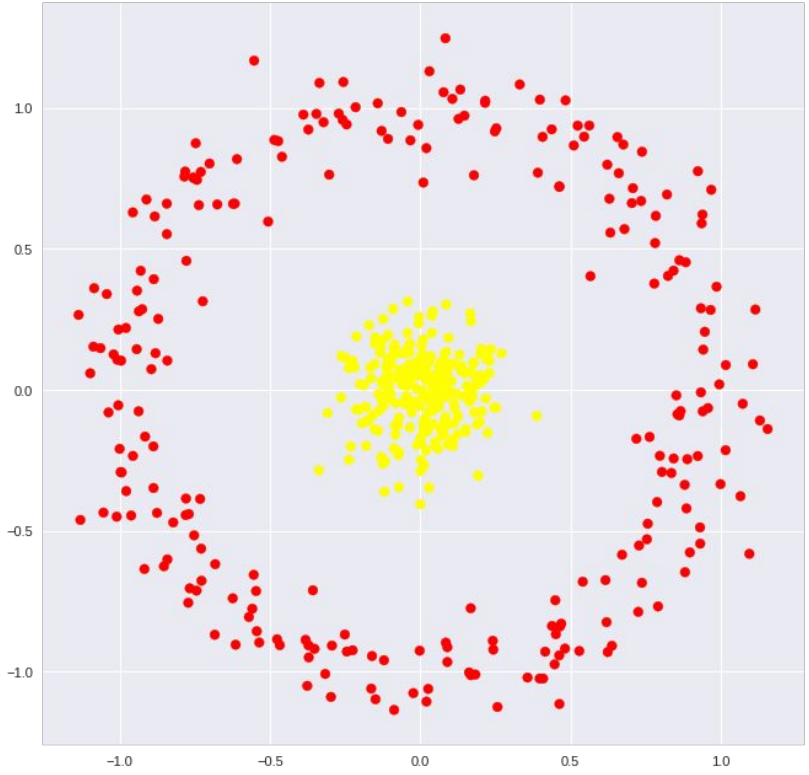
You have a high number of features and **individual features are not meaningful**.

- Key to success: representation learning (automatic feature engineering).

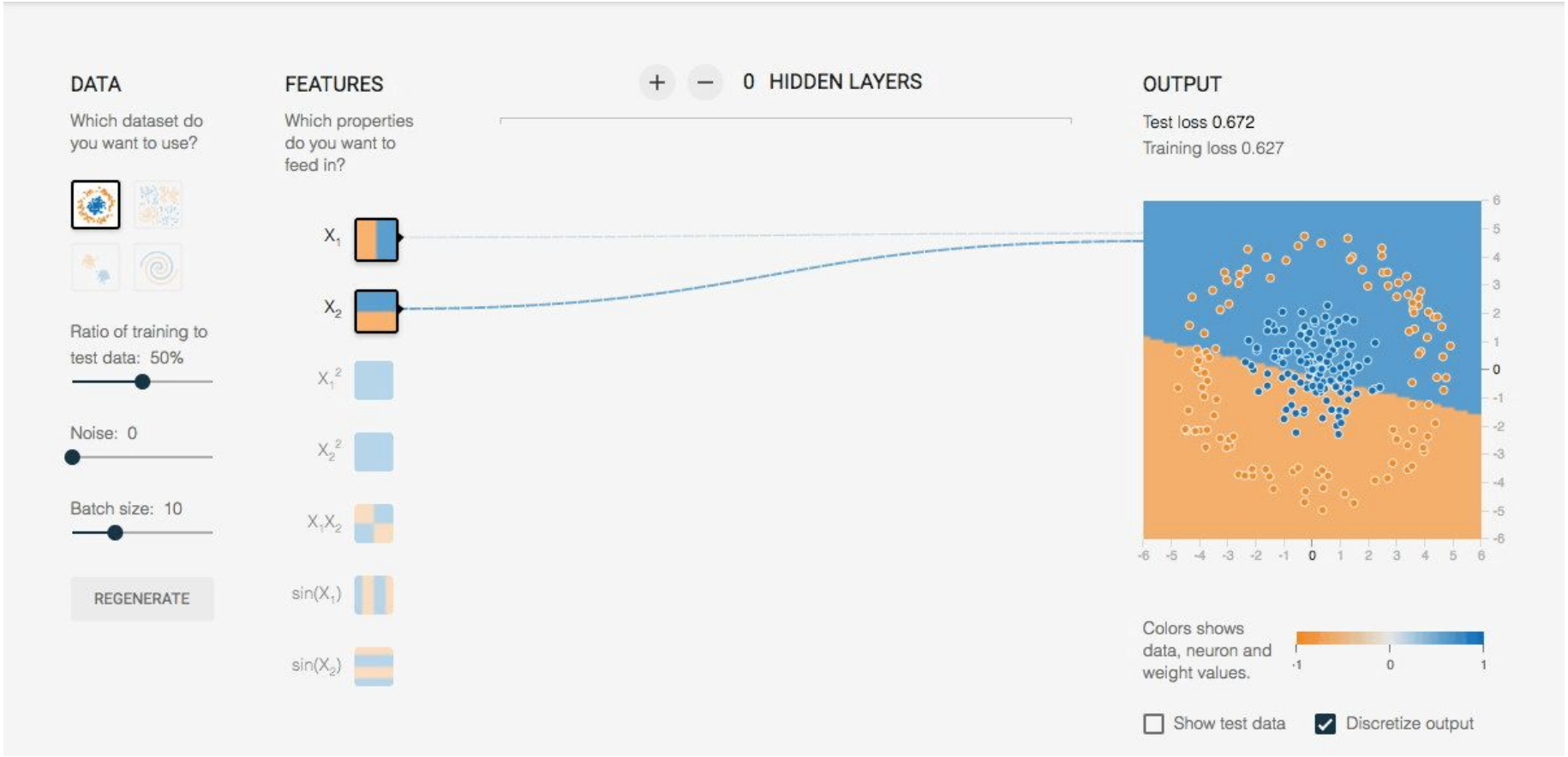
Feature engineering

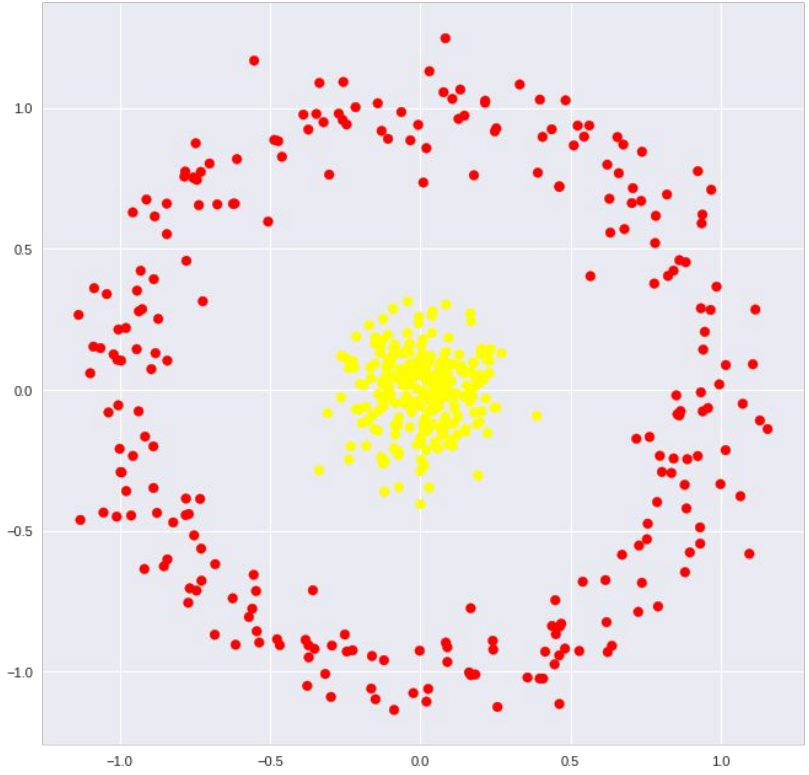
Can you classify this dataset with a linear model?






```
%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')
```


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



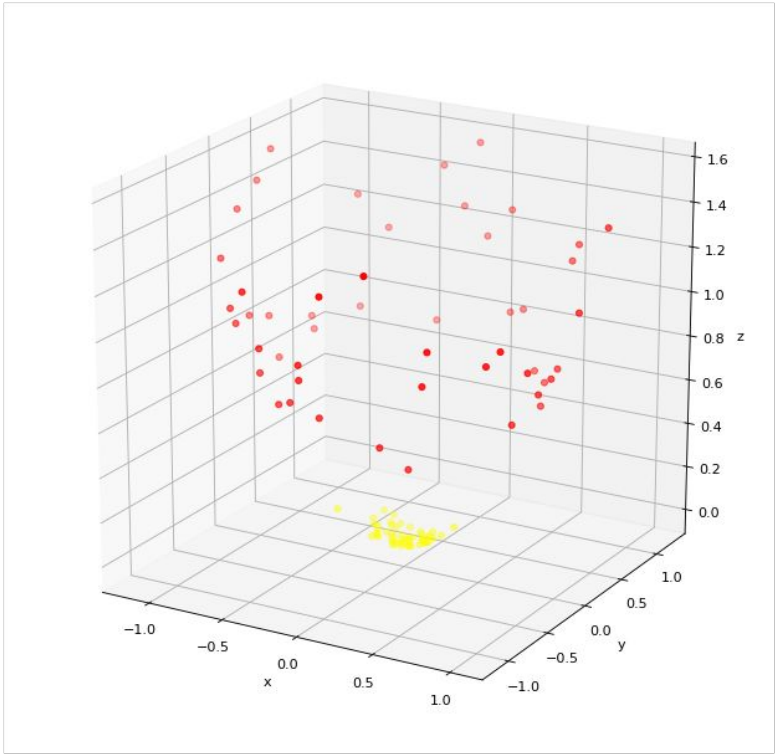
Feature engineering

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

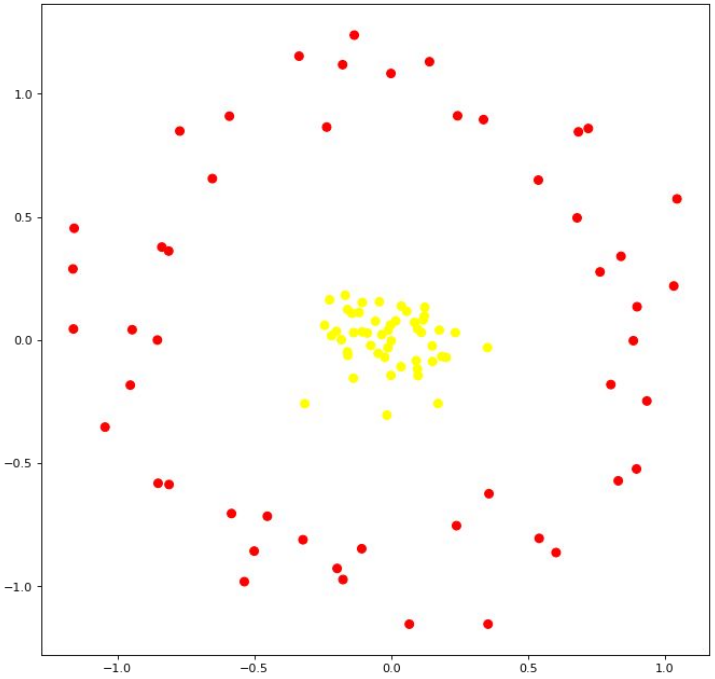
Intuition

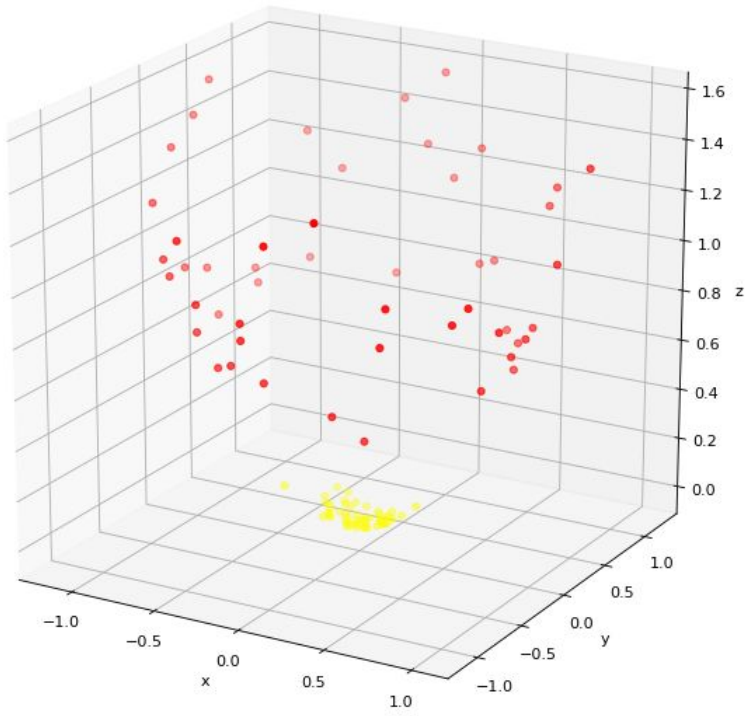
- All values of *z* will be positive.
- Yellow points are closer to the origin…
- So sum of their squared coords will be lower than red!

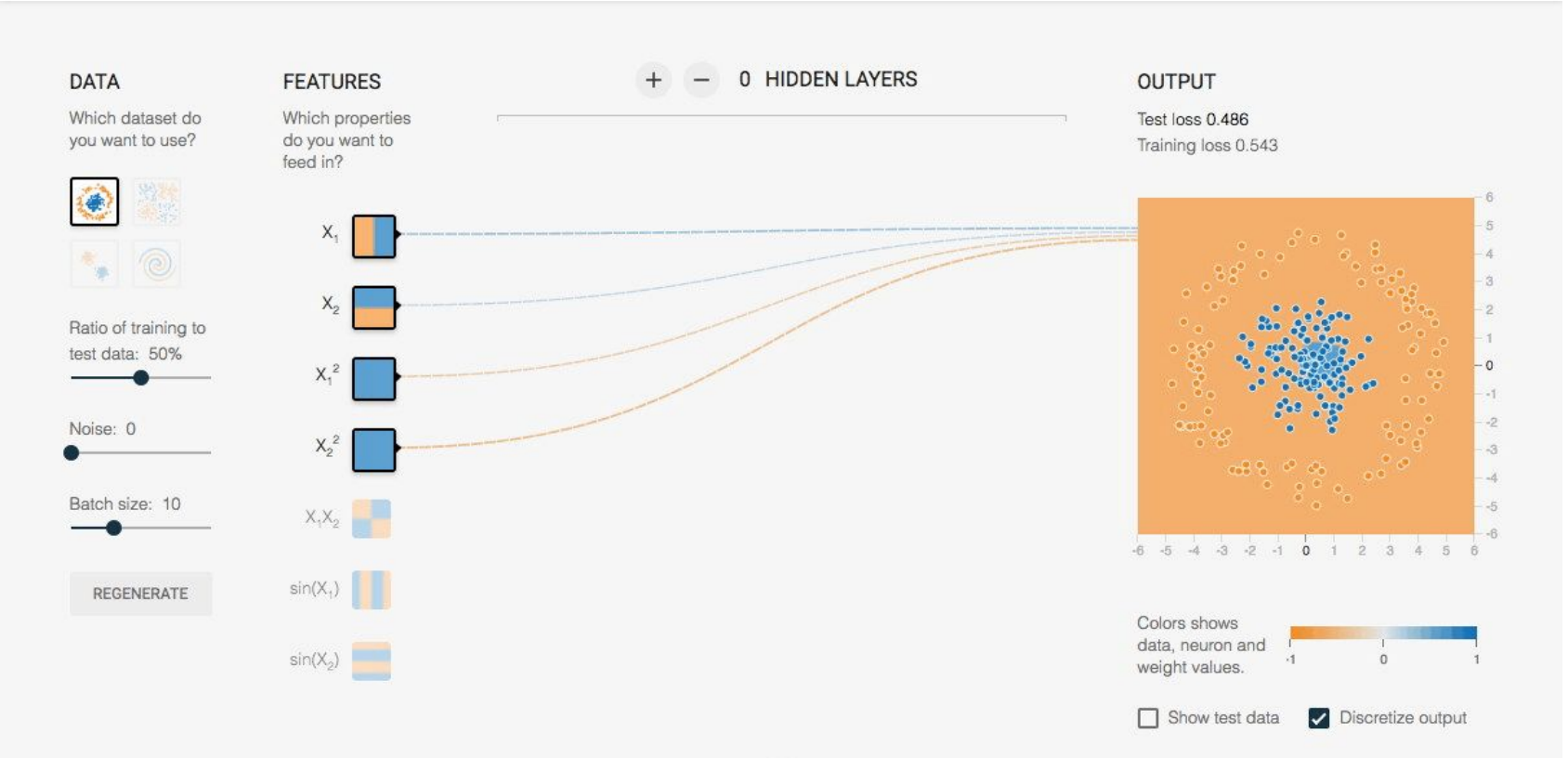
```
fig = plt.figure()
zs = xs**2 + ys**2 # feature engineering
ax = fig2.add_subplot(111,projection='3d')
ax.scatter(xs, ys, zs, c=y)
plt.show()
```



Linearly separable in the new space!







playground.tensorflow.org

When is Deep Learning suitable?

You have a high number of features and **individual features are not meaningful**.

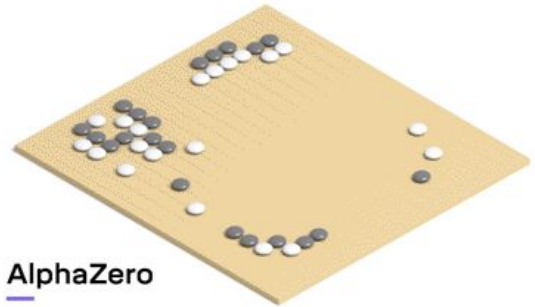
- Key to success: representation learning (automatic feature engineering).

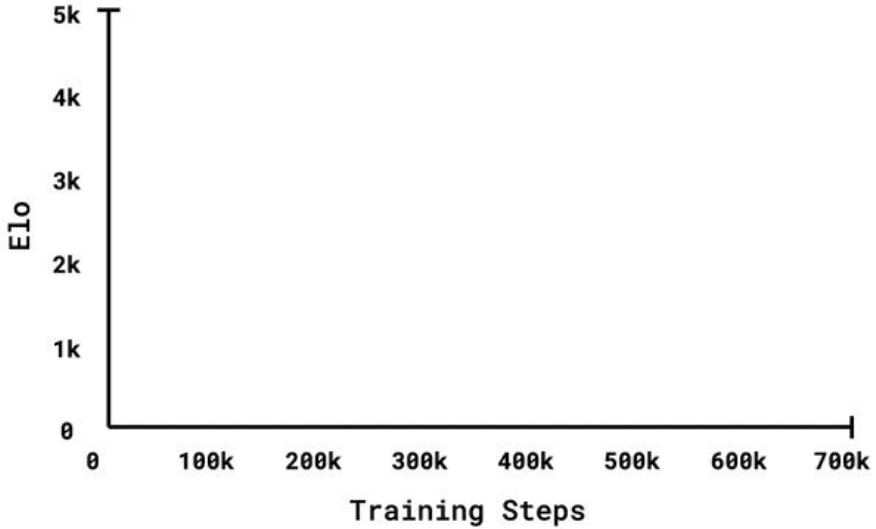
A neural network does **not** classify images by asking questions about individual pixels directly.

- Instead, each layer learns a feature representation. This is **hierarchical**.

The final layer is a linear combination of learned features.

Combined with reinforcement learning





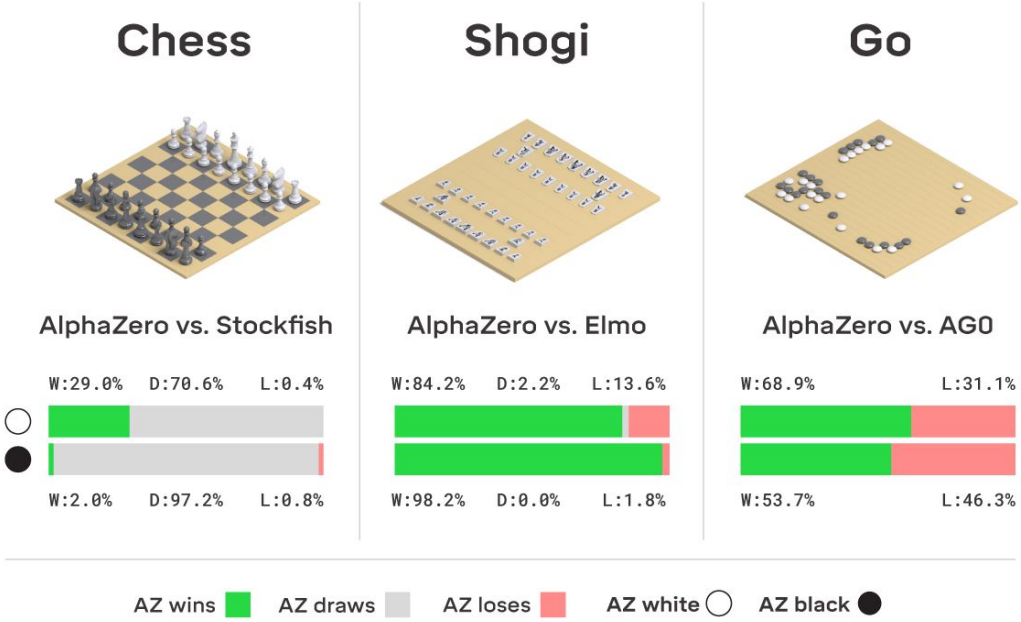
AlphaZero plays chess in a new way

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, …)

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?

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)

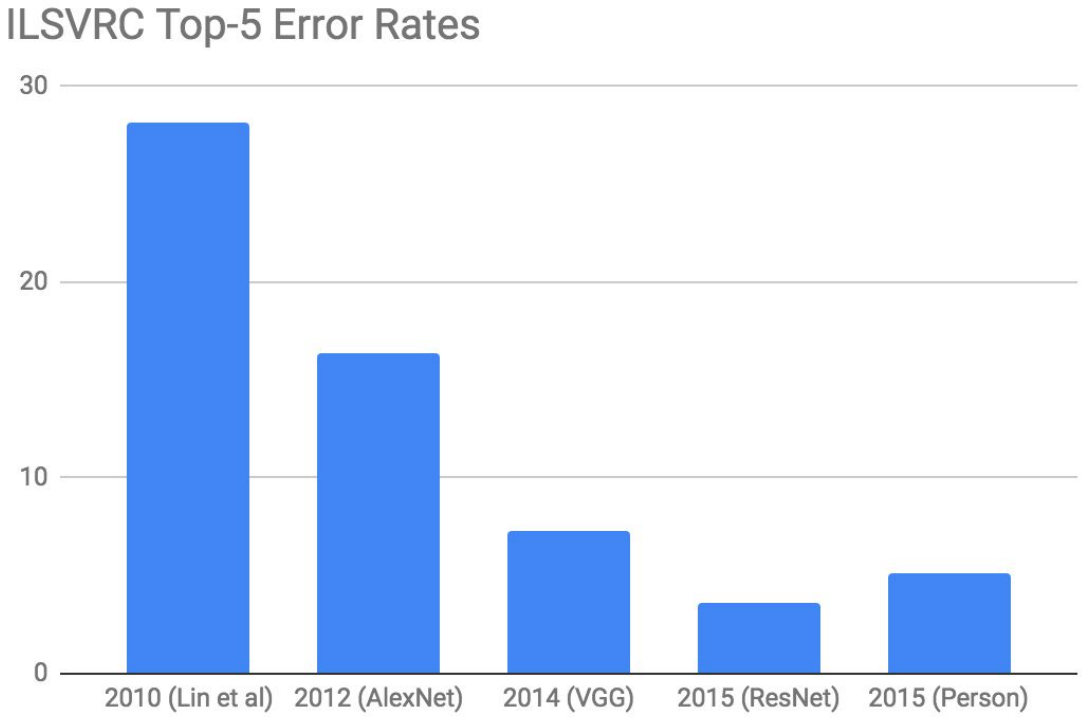
Fatigue: imagine you had to manually classify all 1.4M images

Difficulty: Malamute, or a Husky?

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



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.

Compute

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

- These can be run in parallel.
- Compute with GPUs was **5-10x** cheaper per dollar than with CPUs in 2012.
- Now it's **10-30x** cheaper.

Simple CPU vs GPU benchmark

```
size = (1000, 1000)

def cpu():
  with tf.device("CPU:0"):
    matrix = tf.random.normal(size)
    return tf.matmul(matrix, matrix)

def gpu():
  with tf.device("GPU:0"):
    matrix = tf.random.normal(size)
    return tf.matmul(matrix, matrix)

%timeit -n3 cpu() # 3 loops, best of 3: 28.4 ms per loop
%timeit -n3 gpu() # 3 loops, best of 3: 195 µs per loop
```

Notes

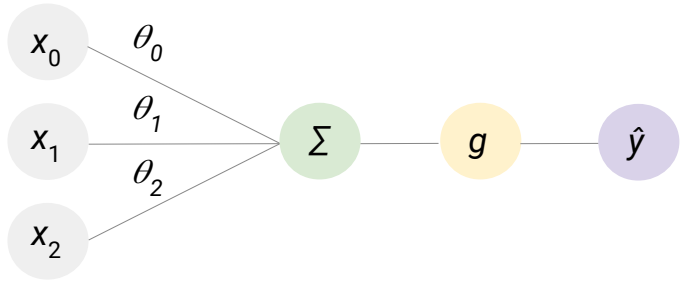
Each op (matmul) is implemented with kernels for different devices (CPU, GPU, etc).

Devices have a preferred kernel they will use it if available.

You do not need to set device placements manually (if you omit those lines, matmul runs on the GPU).

Layers also handle this for you.

A quick look at a neural network



Inputs **weights** **sum** **activation** **output**

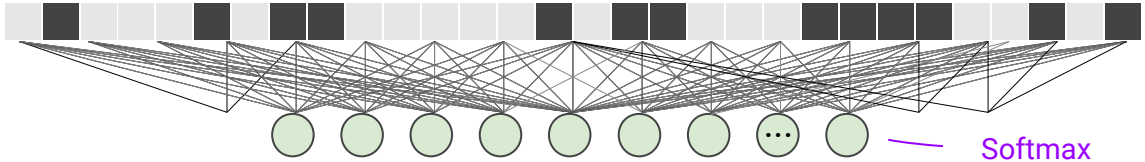
Linear combination of inputs and weights

$$\hat{y} = g\left(\sum x_i \theta_i\right)$$

Can rewrite as a dot product

$$\hat{y} = g\left(x^T \theta\right)$$

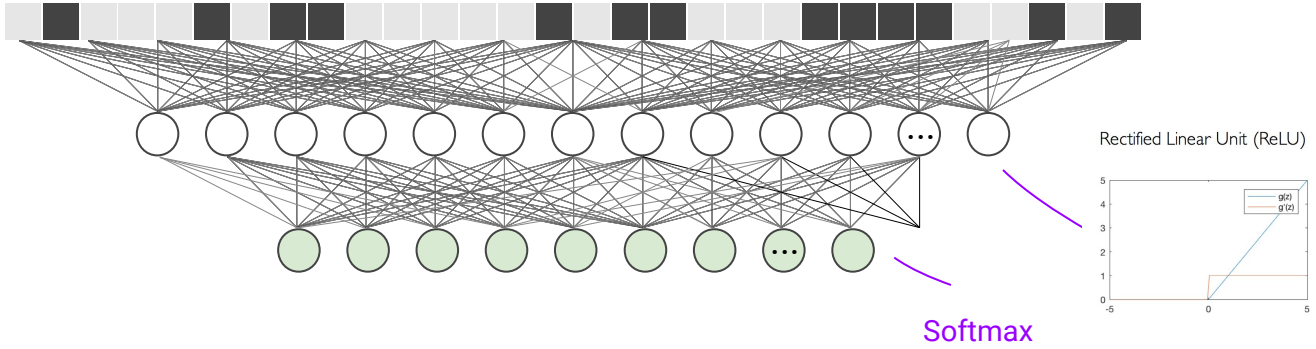
Bias not drawn (you could set x_1 to be a constant input of 1).




```
model = Sequential()
model.add(Dense(10, activation='softmax'))
```

Linear model

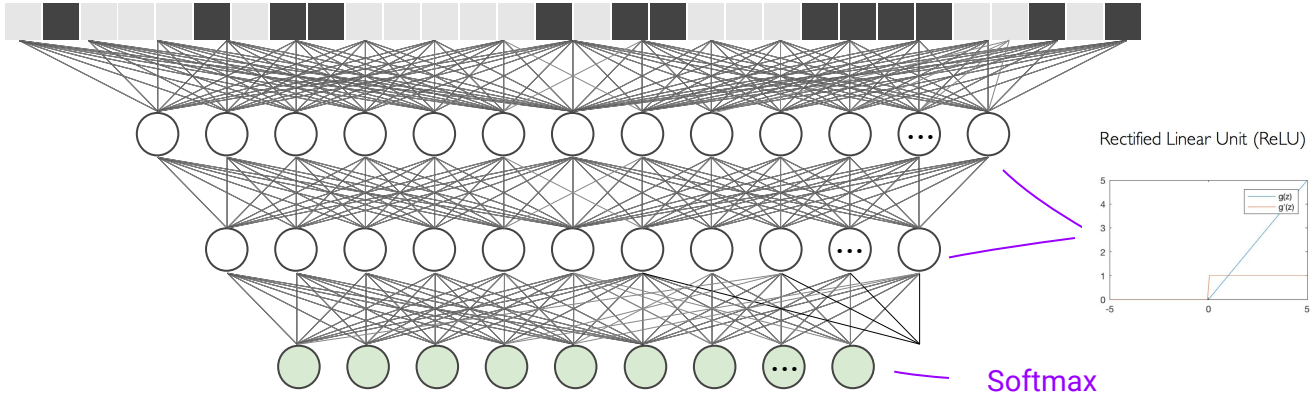
$$f(x) = softmax(W_1 x)$$




```
model = Sequential()
model.add(Dense(256, activation='relu',input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

Linear model $f(x) = softmax(W_1 x)$

Neural network $f(x) = softmax(W_2(g(W_1 x)))$




```
model = Sequential()
model.add(Dense(256, activation='relu',input_shape=(784,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Linear model $f(x) = softmax(W_1 x)$

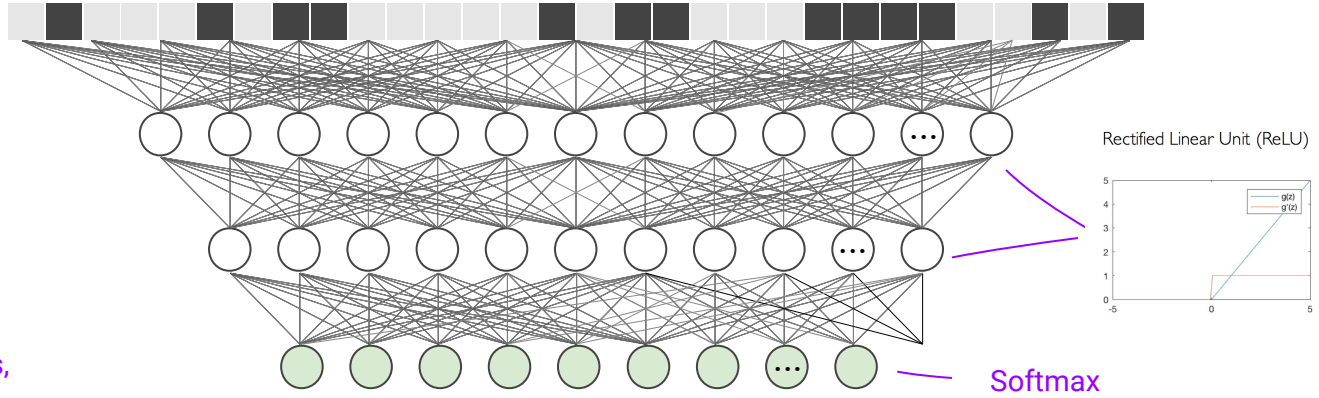
Neural network $f(x) = softmax(W_2(g(W_1 x)))$

Deep neural network $f(x) = softmax(W_3(g(W_2(g(W_1 x)))))$

Terminology

Depth of the network (three layers)

Width of a layer (number of units, or neurons)




```
model = Sequential()
model.add(Dense(256, activation='relu',input_shape=(784,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

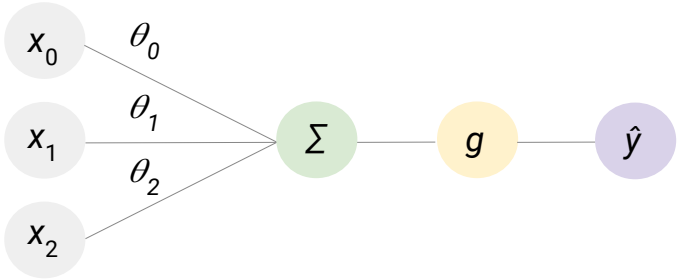
Linear model $f(x) = softmax(W_1 x)$

Neural network $f(x) = softmax(W_2(g(W_1 x)))$

Deep neural network $f(x) = softmax(W_3(g(W_2(g(W_1 x)))))$

Three views of a neuron (code, diagram, equation)

```
layer = Dense(units=1, kernel_initializer='ones', use_bias=False, input_shape=(3,))
data = tf.constant([[1.0, 2.0, 3.0]]) # Note: a batch of data
sum = layer(data) # tf.Tensor([[6.]], shape=(1, 1), dtype=float32)
y = sigmoid(sum) # tf.Tensor([[0.9975274]], shape=(1, 1), dtype=float32)
```



Inputs **weights** **sum** **activation** **output**

Linear combination of inputs and weights

$$\hat{y} = g\left(\sum x_i \theta_i\right)$$

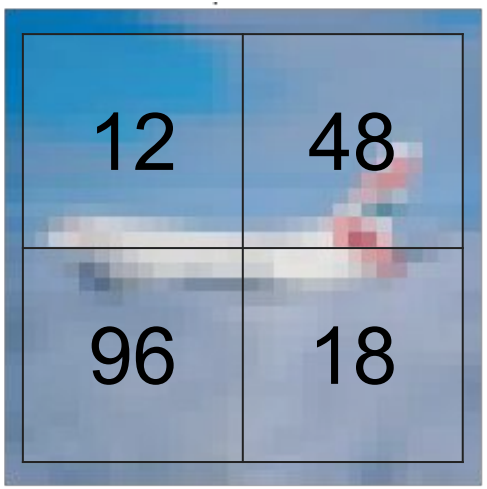
Can rewrite as a dot product

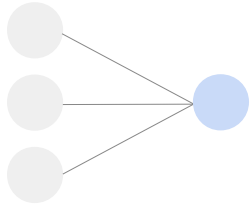
$$\hat{y} = g\left(x^T \theta\right)$$

Bias not drawn (for simplicity, you could set x_1 to be a constant input of 1).

One image and one class

Interpret as “how strongly do you think this image is a plane?”





Multiple inputs; one output

1.4	0.5	0.7	1.2

12
48
96
18

+

0.5

=

130.1	Plane

I realize I occasionally use Wx and xW in these slides (I haven’t had a chance to standardize them yet, core concepts are the same).

w	x	b	Output
Weights	Inputs	Bias	Scores

You can interactively explore Keras layers

```
from tensorflow.keras.layers import Dense
layer = Dense(units=1, kernel_initializer='ones', use_bias=False)
data = tf.constant([[1.0, 2.0, 3.0]]) # Note: a batch of data
print(data) # tf.Tensor([[1. 2. 3.]], shape=(1, 3), dtype=float32)

# Call the layer on our data
result = layer(data)

print(result) # tf.Tensor([[6.]], shape=(1, 1), dtype=float32)
print(result.numpy()) # tf.Tensors have a handy .numpy() method
```

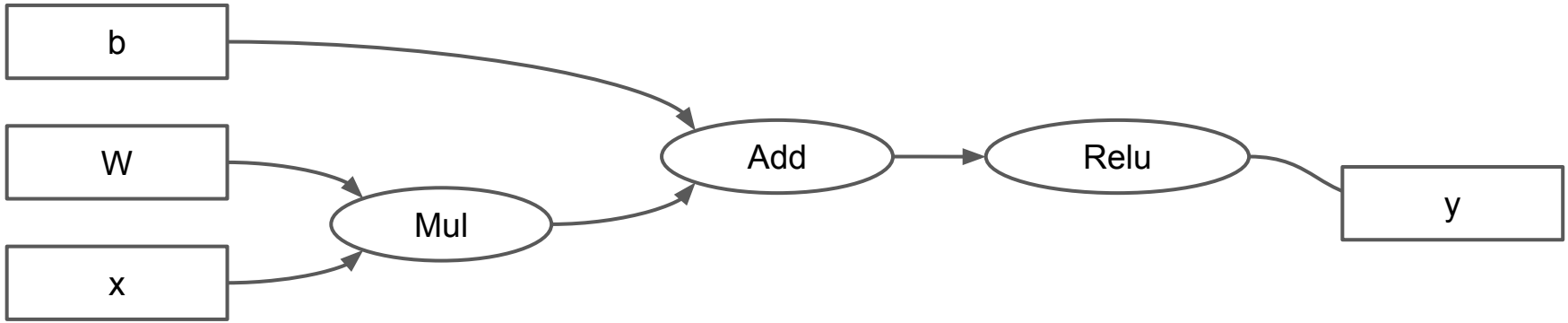
Another example

```
layer = Dense(units=1, kernel_initializer='ones', use_bias=False, input_shape=(3,))
data = tf.constant([[1.0, 2.0, 3.0]]) # Note: a batch of data
sum = layer(data) # tf.Tensor([[6.]], shape=(1, 1), dtype=float32)
y = sigmoid(sum) # tf.Tensor([[0.9975274]], shape=(1, 1), dtype=float32)
```

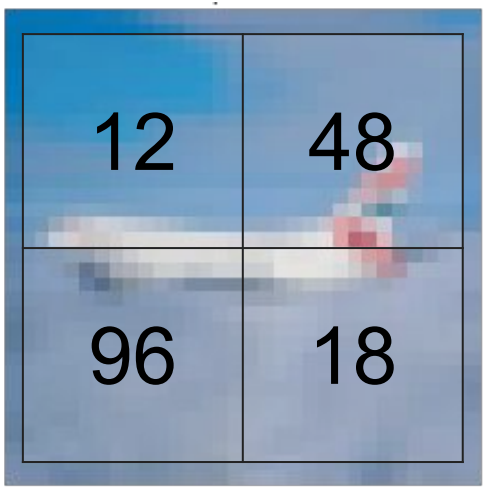
Notes
Don't use these parameters in practice (they're just so you can play around with the layer interactively and see this exact behavior).

- If you had more output units, for example, you would want to break symmetry by initializing weights to small random numbers (which is the default if you don't specify the kernel_initializer).
- In general, use the default parameters unless you have a reason to override them.

Why graphs?



One image and two classes



1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7

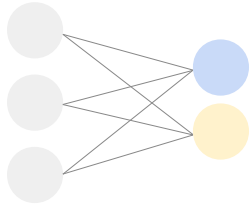
12
48
96
18

+

0.5
1.2

=

130.1	Plane
-11.4	Car



Multiple inputs; multiple outputs

W is now a matrix

W — Weights

x — Inputs

b — Bias

Output — Scores

Two images and three classes

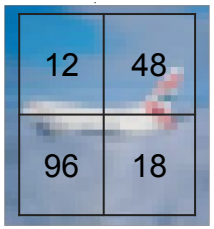


Image 1

Image 2

N x D

1.4	0.5	0.7	1.2
-2.0	0.1	0.2	-0.7
0.2	0.9	-0.2	0.5

W

Weights

D x batch_size

12	4
48	18
96	2
18	96

x

Inputs

+

N x 1

0.5
1.2
0.2

b

Bias

=

N x batch_size

Image 1	Image 2	
130.1	131.7	Plane
-11.4	-71.7	Car
12.8	64.8	Truck

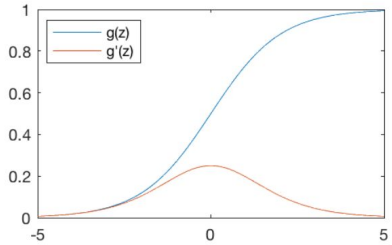
Output

Scores

Activation functions (all applied piecewise)

```
from tensorflow.nn import relu, sigmoid, tanh
```

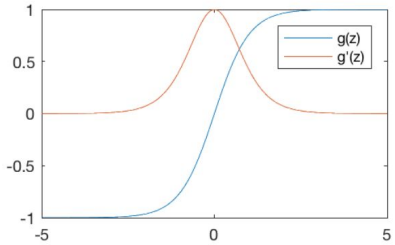
Sigmoid Function



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

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

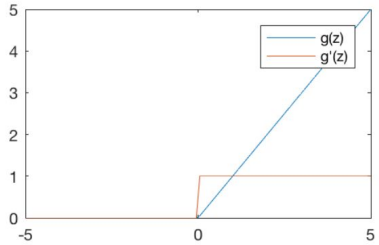
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



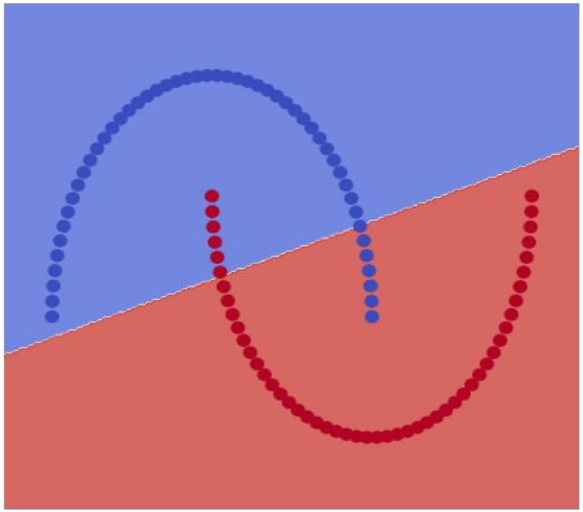
$$g(z) = \max(0, z)$$

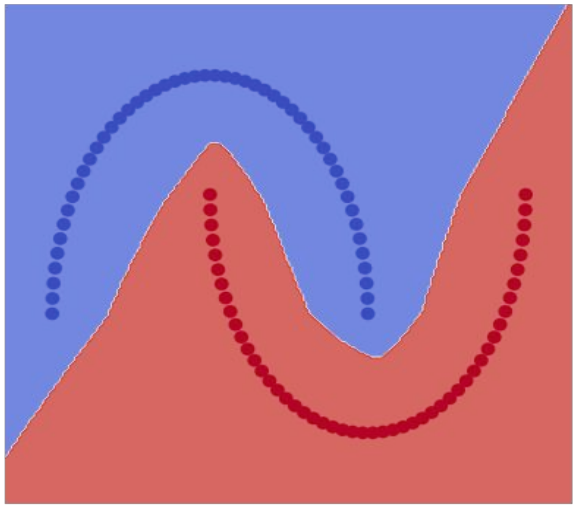
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

A good default: use ReLU for all of your layers except for the last.

Figure from friends at MIT.

Activation functions introduce non-linearities





Notes
- You can make similar plots (and more) with this example. Note: from an older version of TF, but should work out of the box in Colab.
- Each of our convolutional layers used an activation as well (not shown in previous slides).

Without activation, many layers are equivalent to one

```
# If you replace 'relu' with 'None', this model ...
model = Sequential([
  Dense(256, activation='relu', input_shape=(2,)),
  Dense(256, activation='relu'),
  Dense(256, activation='relu'),
  Dense(1, activation='sigmoid')
])



# ... has the same representation power as this one
model = Sequential([Dense(1, activation='sigmoid', input_shape=(2,))])
```

N
-
- You can demo this live in TensorFlow Playground.

Assignment 1 Walkthrough

For next time

Reading

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