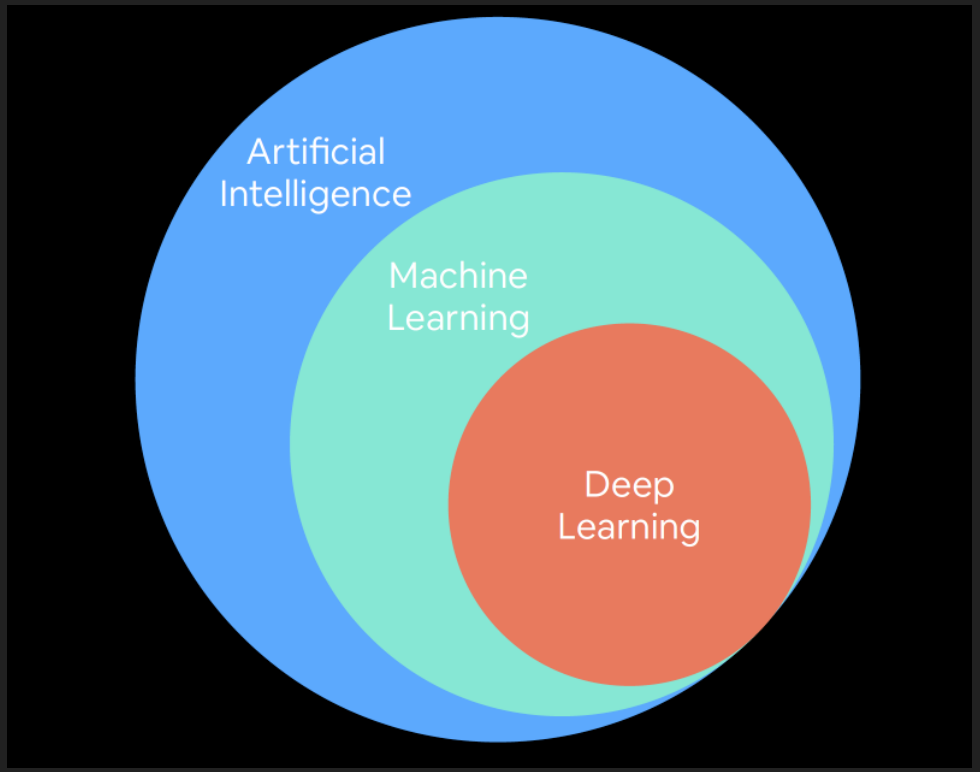
โครงการพัฒนาทักษะการเรียนรู้ของเครื่อง (Machine Learning) ของบัณฑิตเพื่อตอบสนองการพัฒนาประเทศไทย 4.0

16 และ 23 กุมภาพันธ์ 2566

Machine Learning vs. Deep Learning



Machine learning is turning things (data) into numbers and finding patterns in those numbers.

The computer does this part.

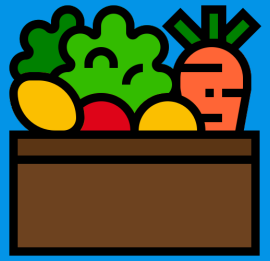
How?

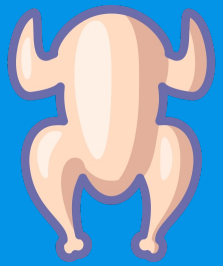
Code & math.

We're going to be writing the code.

Traditional programming

Inputs





Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

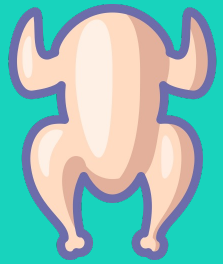
Output

Starts with

Makes

Machine learning algorithm

Inputs



Output

Rules

1. Cut vegetables
2. Season chicken
3. Preheat oven
4. Cook chicken for 30-minutes
5. Add vegetables

Starts with

Figures out

Good reason: ~~Why not?~~

Better reason: For a complex problem, can you think of all the rules?

(probably not)

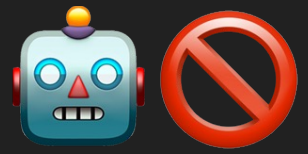
(maybe not very simple...)

"If you can build a simple rule-based system that doesn't require machine learning, do that."

— A wise software engineer... (actually rule 1 of Google's Machine Learning Handbook)

What deep learning is good for

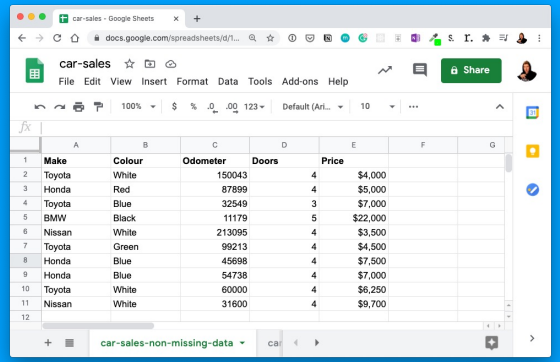
- **Problems with long lists of rules**—when the traditional approach fails, machine learning/deep learning may help.
- **Continually changing environments**—deep learning can adapt ('learn') to new scenarios.
- **Discovering insights within large collections of data**—can you imagine trying to hand-craft rules for what 101 diSerent kinds of food look like?

What deep learning is (typically) not good for 

- **When you need explainability**—the patterns learned by a deep learning model are typically uninterpretable by a human.
- **When the traditional approach is a better option** — if you can accomplish what you need with a simple rule-based system.
- **When errors are unacceptable** — since the outputs of deep learning model aren't always predictable.
- **When you don't have much data** — deep learning models usually require a fairly large amount of data to produce great results.

(though we'll see how to get great results without huge amounts of data)

Machine Learning



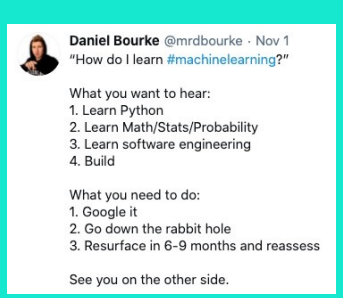
	Make	Colour	Odometer	Doors	Price
1	Toyota	White	150043	4	$4,000
2	Honda	Red	87899	4	$5,000
3	Toyota	Blue	32549	3	$7,000
4	BMW	Black	11179	5	$22,000
5	Nissan	White	213095	4	$3,500
6	Toyota	Green	99213	4	$4,500
7	Honda	Blue	45698	4	$7,500
8	Honda	Blue	54738	4	$7,000
9	Toyota	White	60000	4	$6,250
10	Nissan	White	31600	4	$9,700

Algorithm: gradient boosted machine

dmlc XGBoost

Structured data

Deep Learning



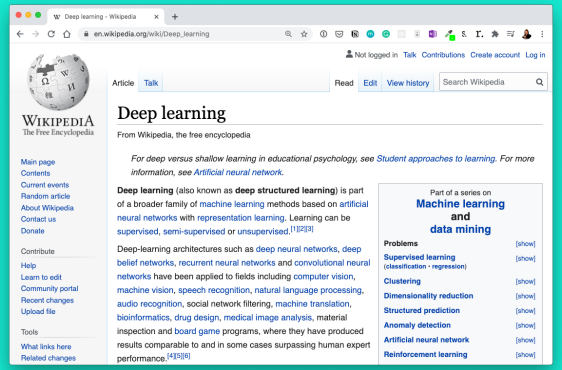
Daniel Bourke @mrdbourke · Nov 1

"How do I learn #machinelearning?"

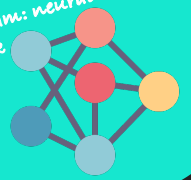
What you want to hear:
1. Learn Python
2. Learn Math/Stats/Probability
3. Learn software engineering
4. Build

What you need to do:
1. Google it
2. Go down the rabbit hole
3. Resurface in 6-9 months and reassess

See you on the other side.



Algorithm: neural network



Unstructured data

Machine Learning vs. Deep Learning

- Random forest
- Gradient boosted models
- Naive Bayes
- Nearest neighbour
- Support vector machine
- …many more

(since the advent of deep learning these are often referred to as "shallow algorithms")

(common algorithms)

- Neural networks
- Fully connected neural network
- Convolutional neural network
- Recurrent neural network
- Transformer
- …many more

What we're focused on building (with PyTorch)

(depending how you represent your problem, many algorithms can be used for both)

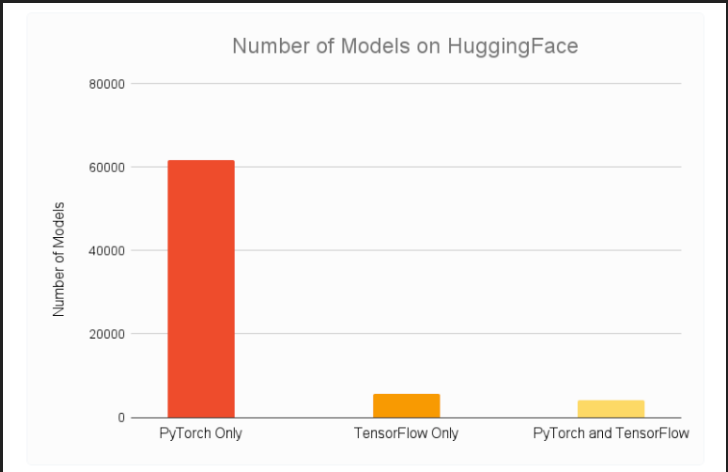
Structured data ⟷ **Unstructured data**

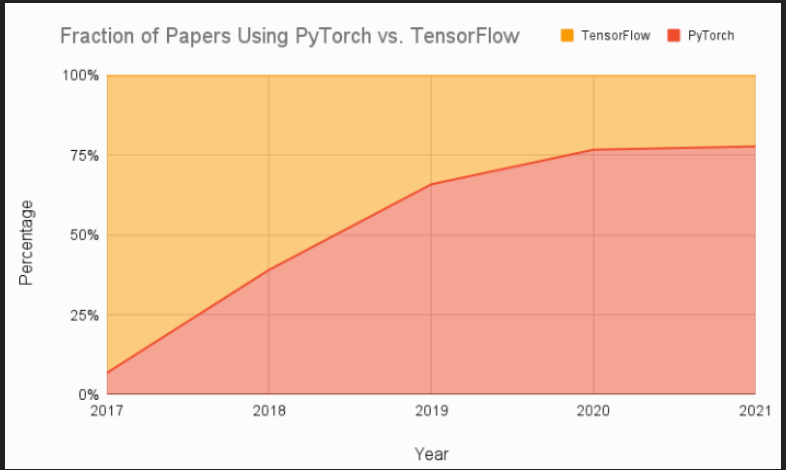

PyTorch

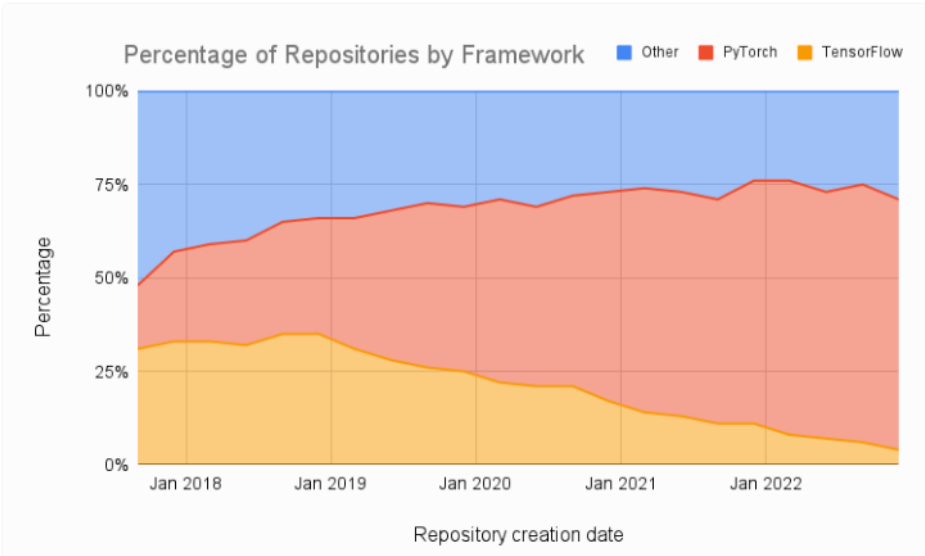
What is PyTorch?

- Most popular research deep learning framework*
- Write fast deep learning code in Python (able to run on a GPU/many GPUs)
- Able to access many pre-built deep learning models (Torch Hub/ `torchvision.models`)
- Whole stack: preprocess data, model data, deploy model in your application/cloud
- Originally designed and used in-house by Facebook/Meta (now open-source and used by companies such as Tesla, Microsoft, OpenAI)

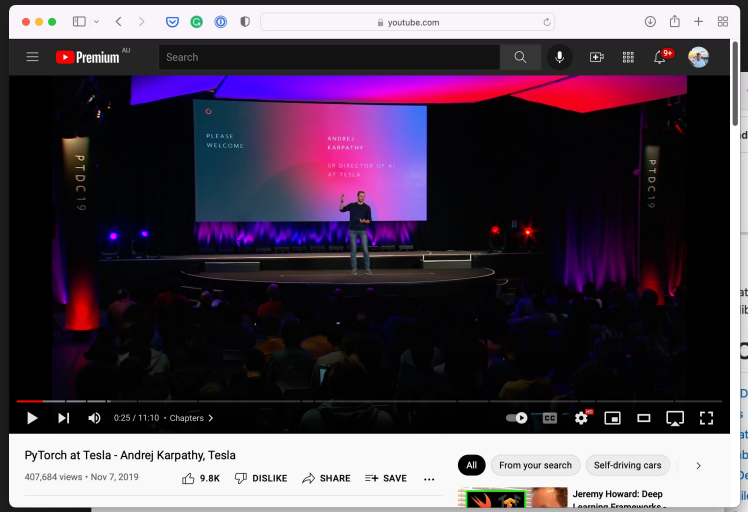
***Source:** paperswithcode.com/trends February 2022

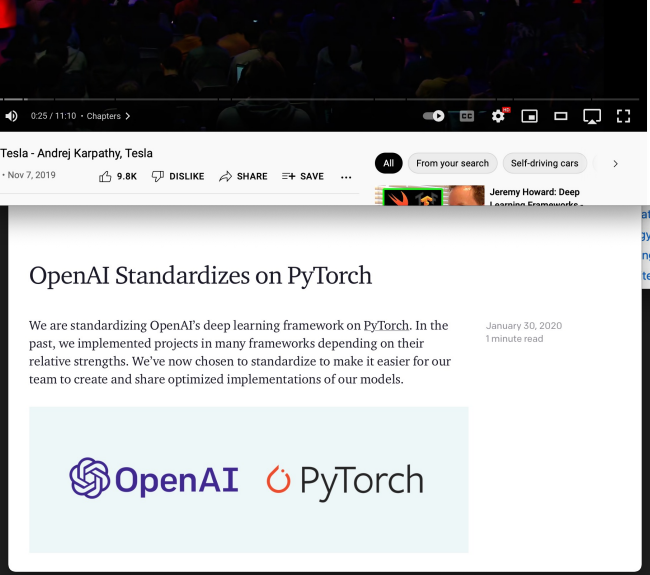


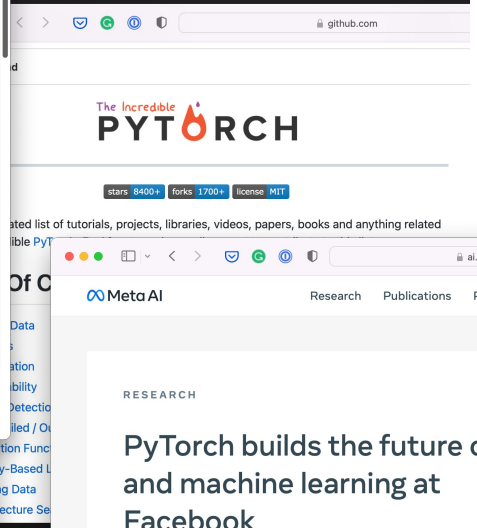




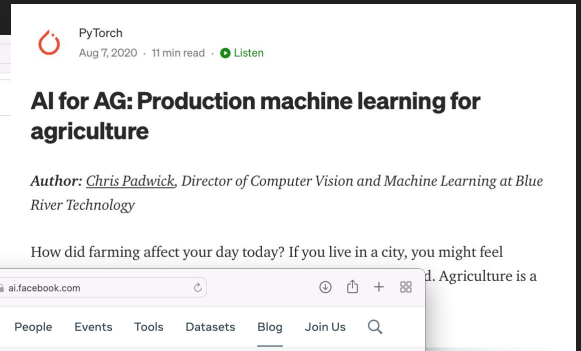
https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/


youtube.com
Premium
Search
PLEASE WELCOME
ANDREJ KARPATHY
SR DIRECTOR OF AI AT TESLA
PTDC'19
0:25 / 11:10 • Chapters
PyTorch at Tesla - Andrej Karpathy, Tesla
407,684 views • Nov 7, 2019
9.8K
DISLIKE
SHARE
SAVE
All
From your search
Self-driving cars
Jeremy Howard: Deep

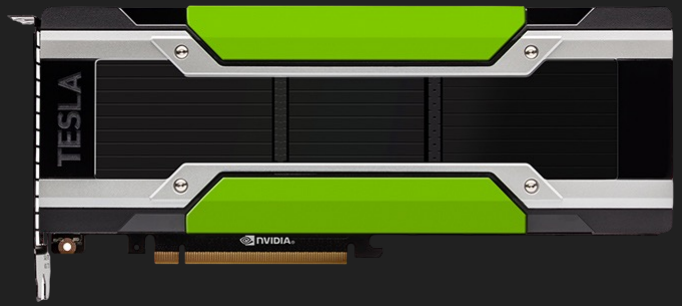

OpenAI Standardizes on PyTorch
We are standardizing OpenAI's deep learning framework on PyTorch. In the past, we implemented projects in many frameworks depending on their relative strengths. We've now chosen to standardize to make it easier for our team to create and share optimized implementations of our models.
January 30, 2020
1 minute read
OpenAI
PyTorch


github.com
The Incredible
PYTORCH
stars 8400+
forks 1700+
license MIT
ated list of tutorials, projects, libraries, videos, papers, books and anything related
Of C
Data
ation
bility
Detectio
lied / Oc
ation Func
gy-Based L
ng Data
tecture Se

ai.facebook.com
Meta AI
Research
Publications
People
Events
Tools
Datasets
Blog
Join Us
RESEARCH
PyTorch builds the future of AI and machine learning at Facebook
June 2, 2021
Share on Facebook
Share on Twitter
Our Work
Facebook's AI models perform trillions of infe
the billions of people that use our technologi
workload demand means we have to continu
Which is why, today we're announcing that w
systems to PyTorch.


PyTorch
Aug 7, 2020 · 11 min read · Listen
AI for AG: Production machine learning for agriculture
Author: Chris Padwick, Director of Computer Vision and Machine Learning at Blue River Technology
How did farming affect your day today? If you live in a city, you might feel
d. Agriculture is a

Microsoft

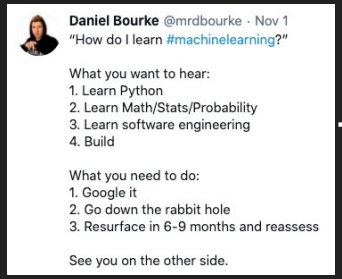


GPU (Graphics Processing Unit)

TPU (Tensor Processing Unit)

“What are Neural Networks?”

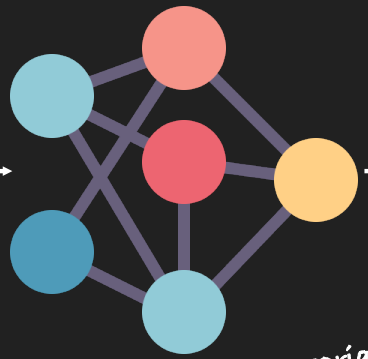
Neural Networks



(before data gets used with a neural network, it needs to be turned into numbers)

```
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
```

Each of these nodes is called a "hidden unit" or "neuron".



(choose the appropriate neural network for your problem)

```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
```

(a human can understand these)

Ramen, Spaghetti

Not a diaster

"Hey Siri, what's the weather today?"

Inputs

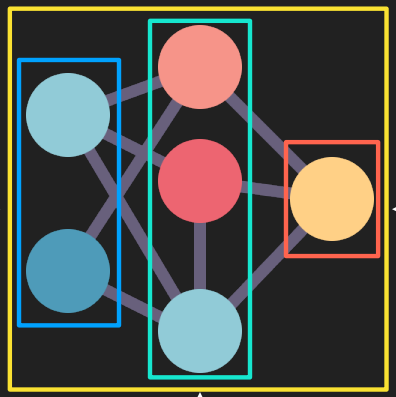
Numerical encoding

Learns representation (patterns/features/weights)

Representation outputs

Outputs

Anatomy of Neural Networks



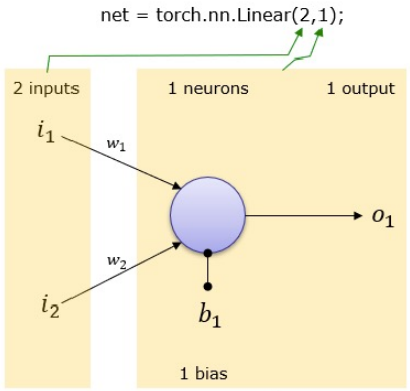
Input layer
(data goes in here)
units/neurons = 2

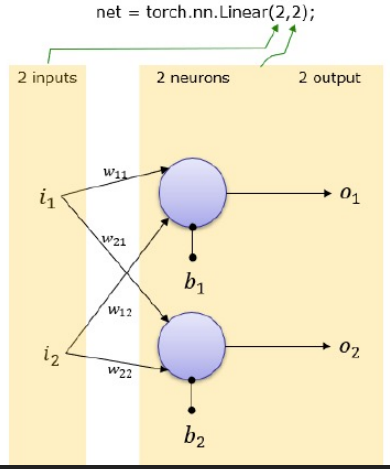
Output layer
(outputs learned representation or prediction probabilities)
units/neurons = 1

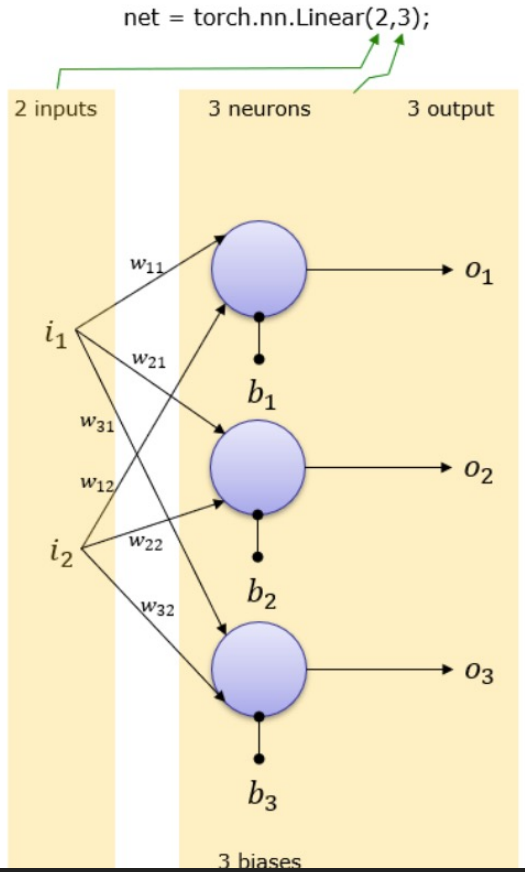
Hidden layer(s)
(learns patterns in data)
units/neurons = 3

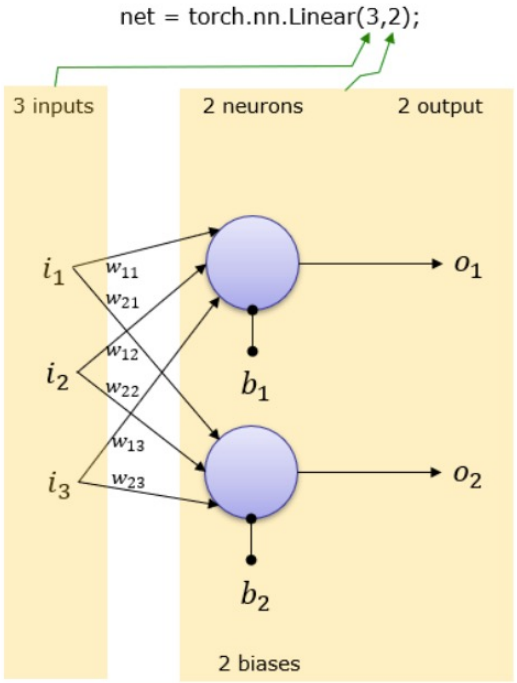
Each layer is usually combination of linear (straight line) and/or non-linear (not-straight line) functions

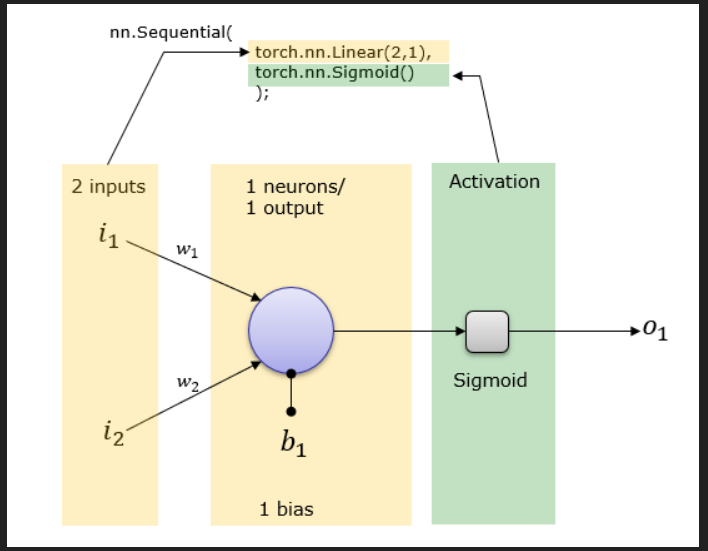
Note: “patterns” is an arbitrary term, you’ll often hear “embedding”, “weights”, “feature representation”, “feature vectors” all referring to similar things.

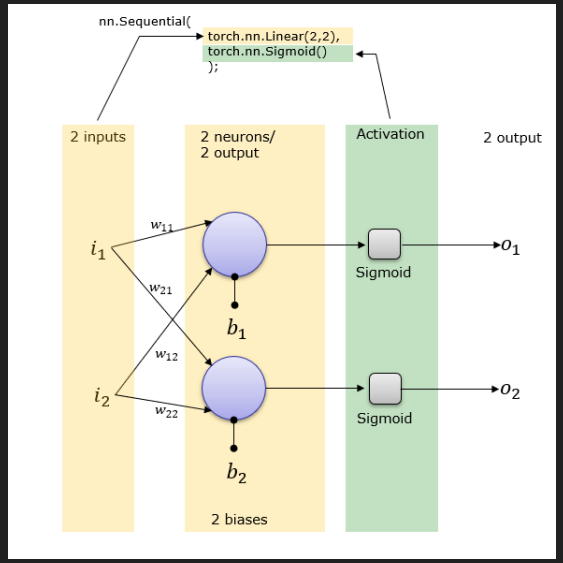

net = torch.nn.Linear(2,1);
2 inputs
1 neurons
1 output
i_1
w_1
o_1
w_2
i_2
b_1
1 bias

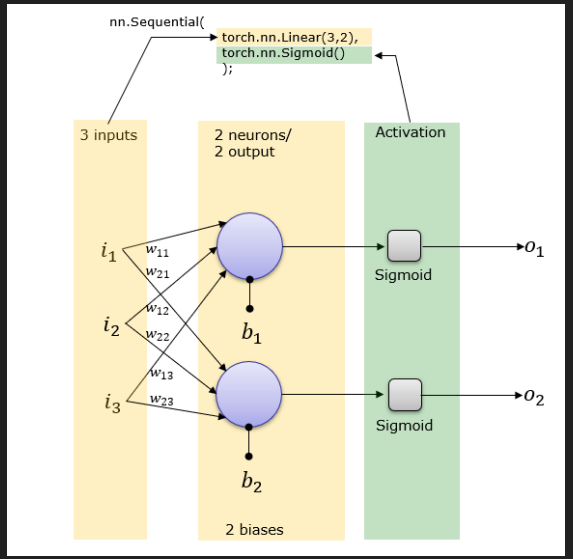

net = torch.nn.Linear(2,2);
2 inputs
2 neurons
2 output
w_{11}
i_1
o_1
w_{21}
b_1
w_{12}
i_2
o_2
w_{22}
b_2

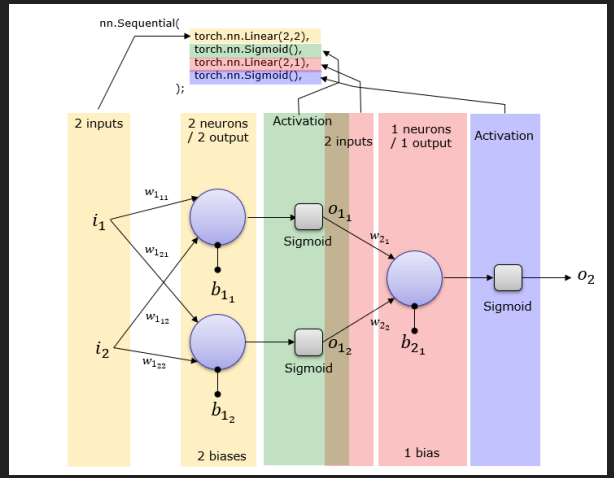

net = torch.nn.Linear(2,3);
2 inputs
3 neurons
3 output
w_{11}
o_1
i_1
w_{21}
b_1
w_{31}
o_2
w_{12}
i_2
w_{22}
b_2
w_{32}
o_3
b_3
3 biases

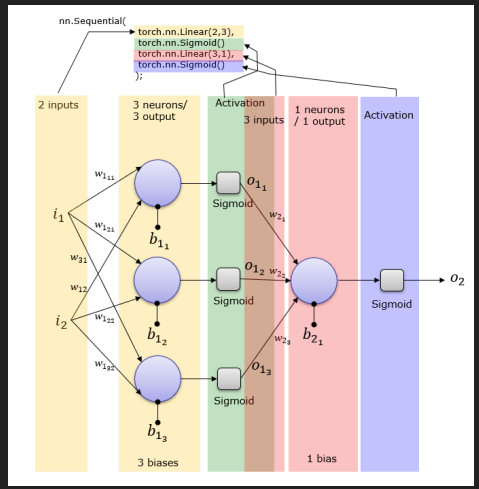

net = torch.nn.Linear(3,2);
3 inputs
2 neurons
2 output
i_1
w_{11}
o_1
w_{21}
w_{12}
b_1
i_2
w_{22}
w_{13}
o_2
i_3
w_{23}
b_2
2 biases


nn.Sequential(
torch.nn.Linear(2,1),
torch.nn.Sigmoid()
);
2 inputs
1 neurons/
1 output
Activation
i_1
i_2
w_1
w_2
b_1
Sigmoid
o_1
1 bias


nn.Sequential(
torch.nn.Linear(2,2),
torch.nn.Sigmoid()
);
2 inputs
2 neurons/
2 output
Activation
2 output
i_1
i_2
w_{11}
w_{21}
w_{12}
w_{22}
b_1
b_2
Sigmoid
Sigmoid
o_1
o_2
2 biases


nn.Sequential(
torch.nn.Linear(3,2),
torch.nn.Sigmoid()
);
3 inputs
2 neurons/
2 output
Activation
i_1
i_2
i_3
w_{11}
w_{21}
w_{12}
w_{22}
w_{13}
w_{23}
b_1
b_2
Sigmoid
Sigmoid
o_1
o_2
2 biases


nn.Sequential(
torch.nn.Linear(2,2),
torch.nn.Sigmoid(),
torch.nn.Linear(2,1),
torch.nn.Sigmoid(),
);
2 inputs
2 neurons
/ 2 output
Activation
2 inputs
1 neurons
/ 1 output
Activation
i_1
i_2
$w_{1_{11}}$
$w_{1_{21}}$
$w_{1_{12}}$
$w_{1_{22}}$
b_{1_1}
b_{1_2}
o_{1_1}
o_{1_2}
Sigmoid
Sigmoid
w_{2_1}
w_{2_2}
b_{2_1}
Sigmoid
o_2
2 biases
1 bias


nn.Sequential(
torch.nn.Linear(2,3),
torch.nn.Sigmoid()
torch.nn.Linear(3,1),
torch.nn.Sigmoid()
);
2 inputs
3 neurons/
3 output
Activation
3 inputs
1 neurons
/ 1 output
Activation
i_1
i_2
$w_{1_{11}}$
$w_{1_{21}}$
$w_{1_{31}}$
$w_{1_{12}}$
$w_{1_{22}}$
$w_{1_{32}}$
b_{1_1}
b_{1_2}
b_{1_3}
o_{1_1}
o_{1_2}
o_{1_3}
Sigmoid
Sigmoid
Sigmoid
w_{2_1}
w_{2_2}
w_{2_3}
b_{2_1}
Sigmoid
o_2
3 biases
1 bias

Three Ways to Build a Neural Network in PyTorch

```
from torch import nn
from collections import OrderedDict

# define model architecture
model3 = nn.Sequential(OrderedDict([
    ('fc1', nn.Linear(16, 12)),
    ('relu1', nn.ReLU()),
    ('fc2', nn.Linear(12, 10)),
    ('relu2', nn.ReLU()),
    ('fc3', nn.Linear(10, 1)),
    ('sigmoid', nn.Sigmoid())
]))

# print model architecture
print(model3)
```

```
Sequential(
  (fc1): Linear(in_features=16, out_features=12, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=12, out_features=10, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=10, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```


```
from torch import nn
from collections import OrderedDict

# define model architecture
model3 = nn.Sequential(OrderedDict([
    ('fc1', nn.Linear(16, 12)),
    ('relu1', nn.ReLU()),
    ('fc2', nn.Linear(12, 10)),
    ('relu2', nn.ReLU()),
    ('fc3', nn.Linear(10, 1)),
    ('sigmoid', nn.Sigmoid())
]))

# print model architecture
print(model3)
```

```
Sequential(
  (fc1): Linear(in_features=16, out_features=12, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=12, out_features=10, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=10, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```

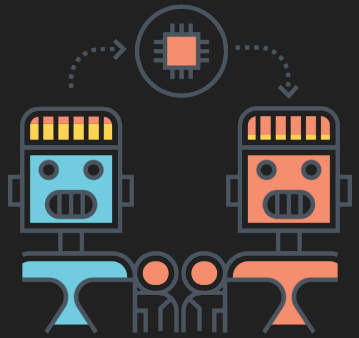

```
import torch
import torch.nn.functional as F
from torch import nn

# define the network class
class MyNetwork(nn.Module):
    def __init__(self):
        # call constructor from superclass
        super().__init__()

        # define network layers
        self.fc1 = nn.Linear(16, 12)
        self.fc2 = nn.Linear(12, 10)
        self.fc3 = nn.Linear(10, 1)

    def forward(self, x):
        # define forward pass
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x

# instantiate the model
model1 = MyNetwork()

#

print(model1)
```

```
MyNetwork(
  (fc1): Linear(in_features=16, out_features=12, bias=True)
  (fc2): Linear(in_features=12, out_features=10, bias=True)
  (fc3): Linear(in_features=10, out_features=1, bias=True)
)
```

Types of Learning

Supervised Learning

Unsupervised & Self-supervised Learning

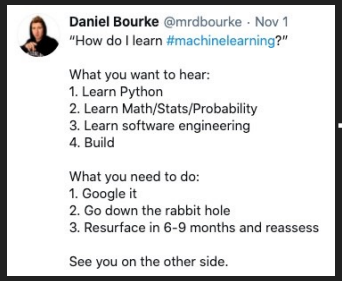


Transfer Learning

We'll be writing code to do these,
but the style of code can be adopted across learning paradigms.

“What is a tensor?”

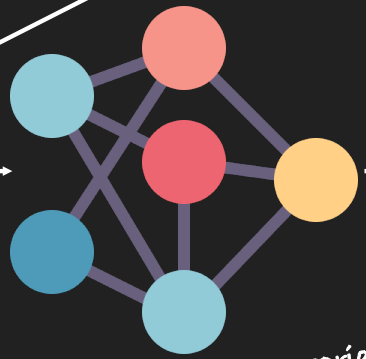
Neural Networks



(before data gets used with an algorithm, it needs to be turned into numbers)

```
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
```

These are tensors!



(choose the appropriate neural network for your problem)

```
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
```

(a human can understand these)

Ramen, Spaghetti

Not spam

"Hey Siri, what's the weather today?"

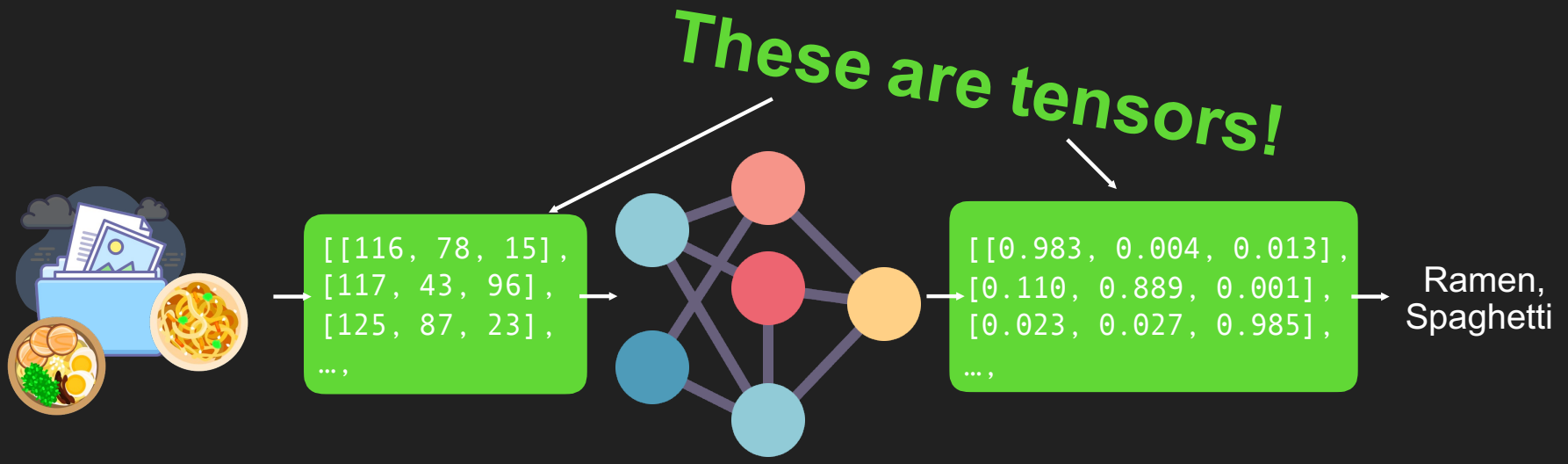
Inputs

Numerical encoding

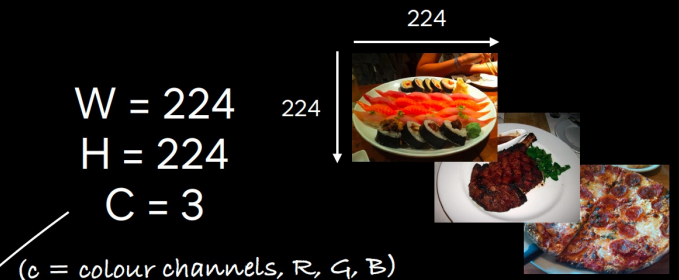
Learns representation (patterns/features/weights)

Representation outputs

Outputs


These are tensors!
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
Ramen,
Spaghetti
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

Classification inputs and outputs



W = 224
H = 224
C = 3

(c = colour channels, R, G, B)

Sushi
Steak
Pizza

Actual output

```
[[0.31, 0.62, 0.44…],
[0.92, 0.03, 0.27…],
[0.25, 0.78, 0.07…],
…,
```

(normalized pixel values)

Numerical encoding

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

Outputs

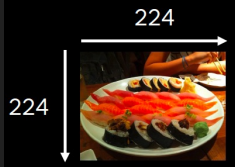
```
[[0.97, 0.00, 0.03],
[0.81, 0.14, 0.05],
[0.03, 0.07, 0.90],
…,
```

Predicted output

(comes from looking at lots of these)

Input and output shapes

(for an image classification example)




```
[[0.31, 0.62, 0.44...],
[0.92, 0.03, 0.27...],
[0.25, 0.78, 0.07...],
...,
```

Inputs

Machine Learning Algorithm

Outputs

```
[0.97, 0.00, 0.03]
```

(prediction probabilities)

(gets represented as a tensor)

```
[batch_size, colour_channels, width, height]
```

```
Shape = [None, 3, 224, 224]
        or
Shape = [32, 3, 224, 224]
```

(32 is a very common batch size)

These will vary depending on the problem you're working on.

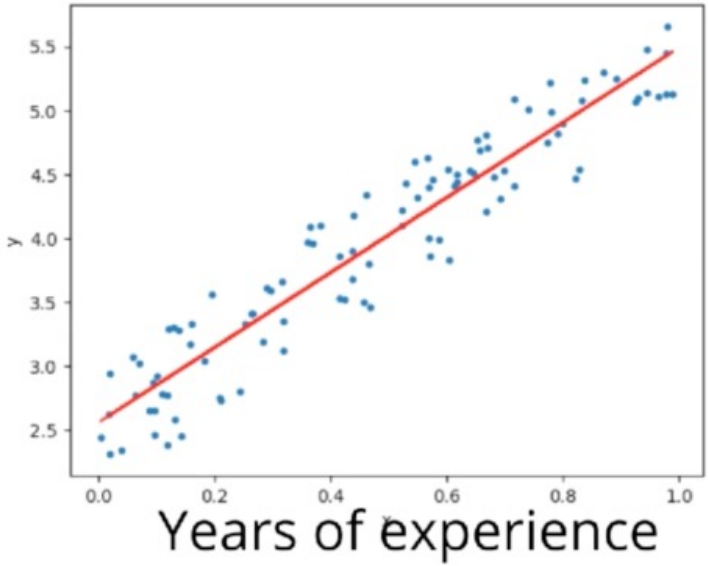
```
Shape = [3]
```

Example: Regression

- Fit a line or curve
- This is why grumpy old statisticians like to say: “machine learning is just glorified curve-fitting”

$$\hat{y} = mx + b$$

(sometimes we also use “a” for the slope)

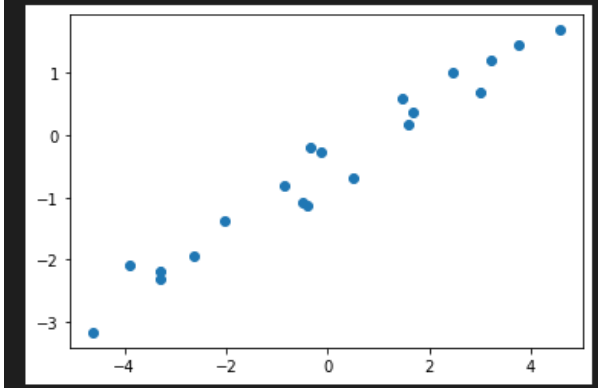


```
import matplotlib.pyplot as plt
import numpy as np
```
✓ 0.4s

```
# generate 20 data points
N = 20

#random data on x-axis
x= np.random.rand(N)*10-5

y = 0.5*x -1+ np.random.rand(N)
```
✓ 0.2s

```
plt.scatter(x,y)
```
✓ 0.1s

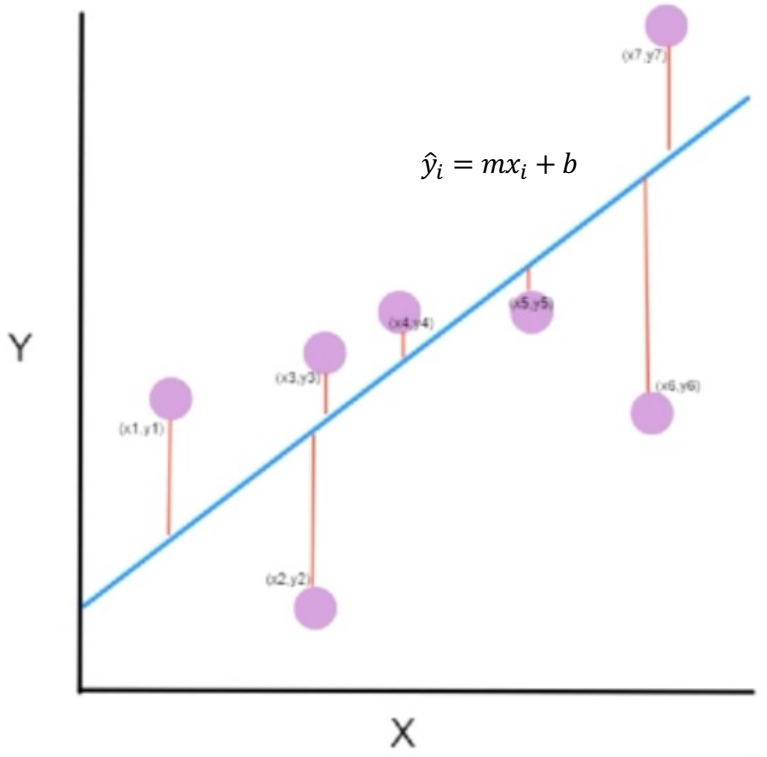
```
<matplotlib.collections.PathCollection at 0x7fc1b147a410>
```




```
# In ML we want our data to be of shape:
# (num_samples x num_dimensions)
X = x.reshape(N, 1)
Y = y.reshape(N, 1)
```
✓ 0.2s

The Loss

- Data = {(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)}
- N = number of samples in the dataset
- The line **cannot** perfectly pass through all the data points



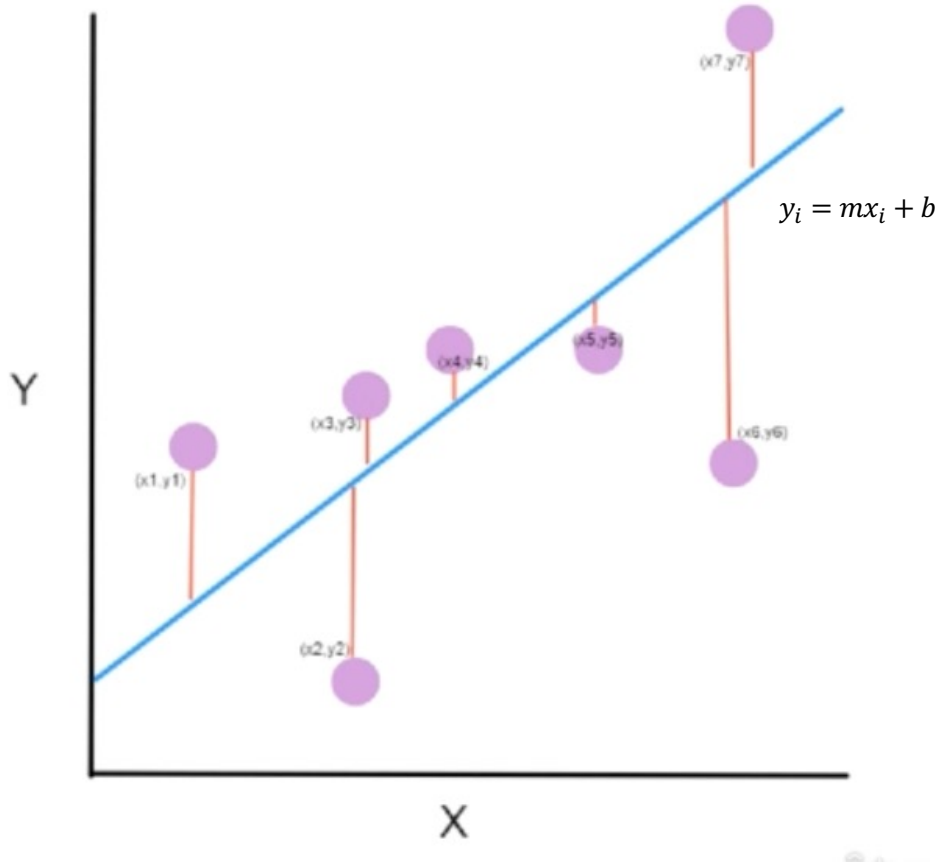
MSE = Mean Squared Error

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Applying the MSE to find slope / intercept

- Plug in the expression for the predictions ($\hat{y}_i$)
- Quiz: What are the **variables** in this expression?

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$



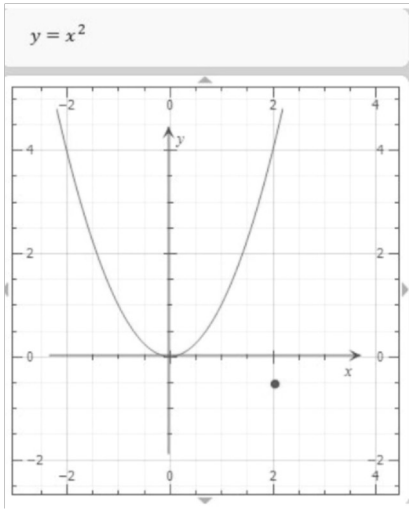
Applying the MSE to find slope / intercept

- Note: L (loss) = MSE
- We want to *minimize the loss* with respect to the parameters (*m*, *b*)

$$m^*, b^* = \arg\min_{m,b} L$$

First Solution

Minimize $f(x) = x^2$



- Use calculus!
- df / dx = 2x = 0
- Solve for x: x = 0

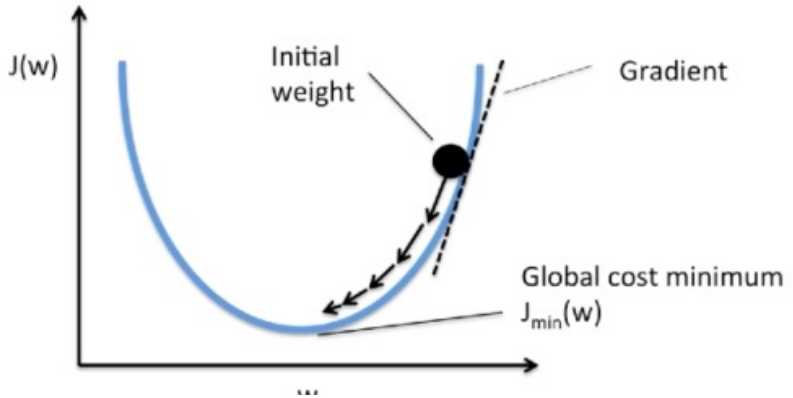
Second Solution : **Gradient Descent**

- Find the derivatives and set them to 0, solve for the parameters
- Yields 2 equations and 2 unknowns: can solve for (m, b)
- Try it as an exercise!: Should be in terms of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

$$\partial L/\partial m = 0,\ \partial L/\partial b = 0$$

Gradient Descent to find hidden parameters

```
# gradient descent
# pseudocode

θ = (m, b) = random()
for i in range(n_epochs):
    θ = θ - η∇_θ L
```



L= $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$

$$\frac{\partial}{\partial m} = \frac{2}{n}\sum_{i=1}^{n} -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial b} = \frac{2}{n}\sum_{i=1}^{n} -(y_i - (mx_i + b))$$

```
def Gradient(m,b) :

    grad_m =0; grad_b = 0;

    for i in range(N) :
        grad_m += -X[i]*(Y[i]- (m*X[i]+b))
        grad_b += - (Y[i]- (m*X[i]+b) )

    return grad_m, grad_b

Gradient(-1,-1)
```

✓ 0.3s

```
(array([-209.28382035]), array([-10.12894605]))
```

```
# gradient descent
# pseudocode

θ = (m, b) = random()
for i in range(n_epochs):
    θ = θ - η∇_θ L
```

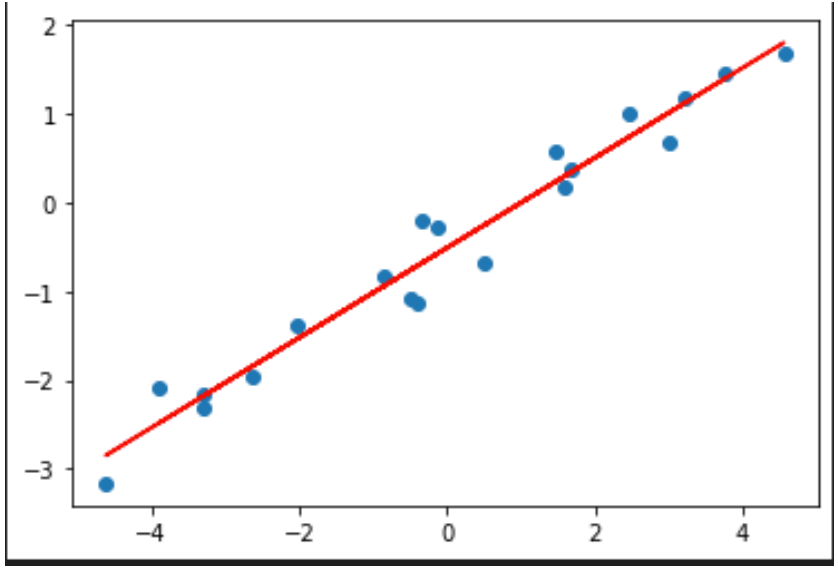
```
predict = theta[0]*X + theta[1]
plt.plot(X, predict, 'r')
plt.scatter(x,y)
```

✓ 0.8s

```
n_epochs      = 1000
learning_rate = 0.01
theta         = np.random.rand(2,1)

for i in range(n_epochs) :
    grad_m, grad_b = Gradient(theta[0], theta[1])
    theta[0] -= learning_rate*grad_m
    theta[1] -= learning_rate*grad_b


theta
```

✓ 0.1s

```
array([[ 0.50714187],
       [-0.51174366]])
```



SDG: Stochastic gradient descent

Third Solution :Pytorch

$$\theta = \theta - \eta \nabla_{\theta} \mathrm{L}$$

$$\mathrm{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

```
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

```
# gradient descent
# pseudocode

θ = (m, b) = random()
for i in range(n_epochs):
    θ = θ - η∇_θ L
```

```
# Train the model
n_epochs = 30
for it in range(n_epochs):
  # zero the parameter gradients
  optimizer.zero_grad()

  # Forward pass
  outputs = model(inputs)
  loss = criterion(outputs, targets)

  # Backward and optimize
  loss.backward()
  optimizer.step()
```

$\hat{y}_i = mx_i + b$

```
# Train the model
n_epochs = 30
losses = []
for it in range(n_epochs):
  # zero the parameter gradients
  optimizer.zero_grad()

  # Forward pass
  outputs = model(inputs)
  loss = criterion(outputs, targets)

  # keep the loss so we can plot it later
  losses.append(loss.item())

  # Backward and optimize
  loss.backward()
  optimizer.step()

  print(f'Epoch {it+1}/{n_epochs}, Loss: {loss.item():.4f}')
```

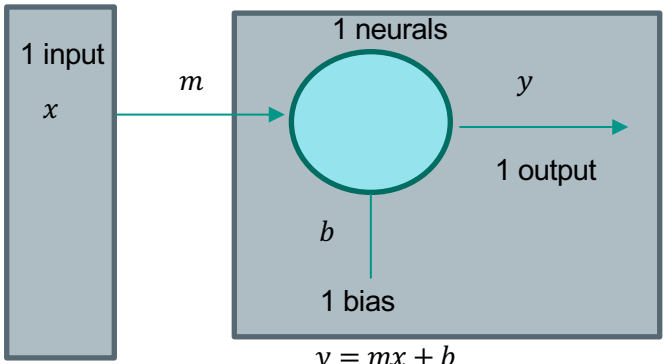
```
Epoch 1/30, Loss: 12.1352
Epoch 2/30, Loss: 2.8066
Epoch 3/30, Loss: 2.4554
Epoch 4/30, Loss: 2.1753
Epoch 5/30, Loss: 1.9483
Epoch 6/30, Loss: 1.7641
Epoch 7/30, Loss: 1.6148
```

```
[ ]
# Create the linear regression model
model = nn.Linear(1, 1)
```

```
[ ]
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
```

```
# In ML we want our data to be of shape:
# (num_samples x num_dimensions)
X = X.reshape(N, 1)
Y = Y.reshape(N, 1)

# PyTorch uses float32 by default
# Numpy creates float64 by default
inputs = torch.from_numpy(X.astype(np.float32))
targets = torch.from_numpy(Y.astype(np.float32))
```

$$net = torch.Linear(1,1)$$



$$y = mx + b$$

Example : Parameter Estimation

Home Work 2

$$signal = A\sin(t + \theta) + b$$

$$y = signal + noise$$

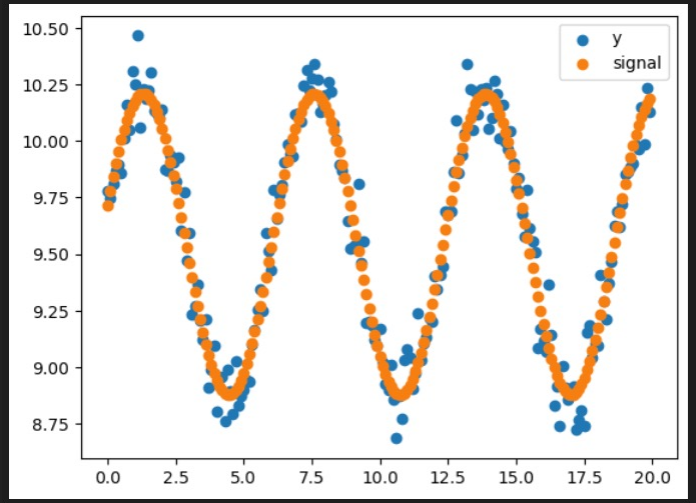
```
t = np.arange(0, 20, 0.1)

#random data on x-axis
A   = np.random.rand()
b     = 20*np.random.rand()
ceta = 2*np.random.rand()*np.pi

signal =  A*np.sin(t+ceta)+b
y       =  signal + 0.1*np.random.randn(len(t))


print(f"A = {A}, ceta = {ceta} b= {b}")
```

```
A = 0.5929415692077172, ceta = 5.396419680771073 b= 6.038952373195254
```

```
plt.scatter(t,y)
plt.scatter(t,signal)
plt.legend(['y','signal'])
```

✓ 0.1s

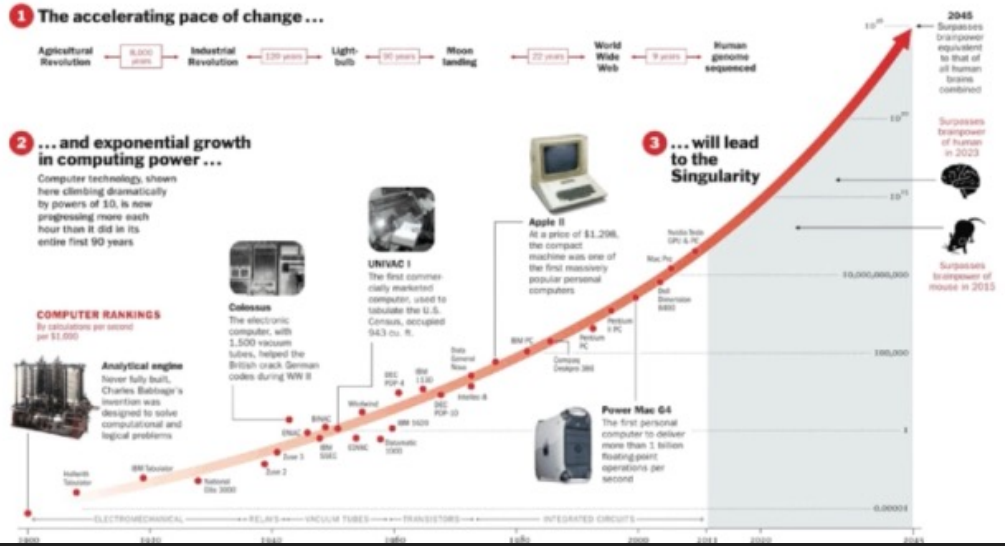
```
<matplotlib.legend.Legend at 0x16aaf3100>
```



Home Work

Real-World Dataset: Moore's Law

- The number of transistors per square inch on integrated circuits doubles approximately every 2 years



Exponential Growth of Computing

Twentieth through twenty first century

10^{60}
10^{55}
10^{50}
10^{45}
10^{40}
10^{35}
10^{30}
10^{25}
10^{20}
10^{15}
10^{10}
10^{5}
1
10^{-5}
10^{-10}

All Human Brains

One Human Brain

One Mouse Brain

One Insect Brain

Logarithmic Plot

Home Work 1

```
df = pd.read_csv('https://raw.githubusercontent.com/lazyprogrammer/machine_learning_examples/master/tf2.0/moore.csv', header=None)
df= df.rename(columns={0: "years", 1: "data"})

df
```

✓ 0.2s

	years	data
0	1971	2300
1	1972	3500
2	1973	2500
3	1973	2500
4	1974	4100
...	...	...
157	2017	18000000000
158	2017	19200000000
159	2018	8876000000
160	2018	23600000000
161	2018	9000000000

162 rows × 2 columns

```
df.plot(kind='scatter', x='years', y='data')
```

✓ 0.9s

```
<AxesSubplot:xlabel='years', ylabel='data'>
```