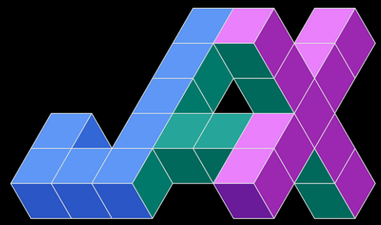
Scientific Computing with



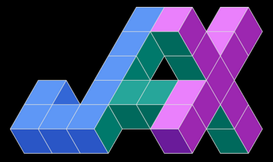
/rumeysskara

/rumeysskara

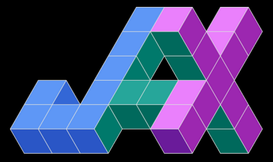
/rumeysakara

Rümeysa Kara

Data Scientist
Google Developer Expert



1. Machine Learning & Deep Learning
2. What is JAX?
    a. autograd
    b. XLA
3. Install JAX
4. API Layering of JAX



5. Numerical Computation in JAX
    a. jax.grad
    b. jax.jit
    c. jax.vmap
    d. jax.pmap
6. How Does JAX Handle Randomness?
7. Flax

Artificial Intelligence

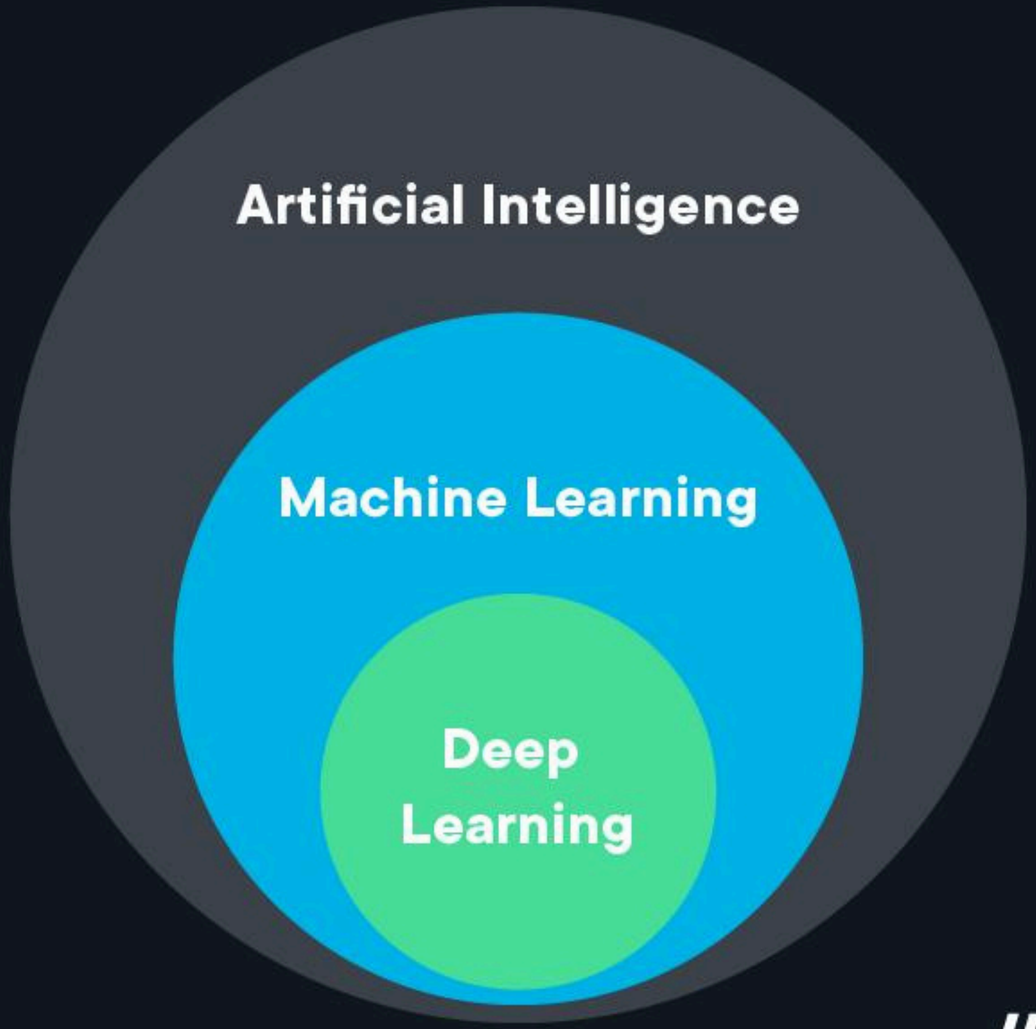
A science devoted to making machines think and act like humans.

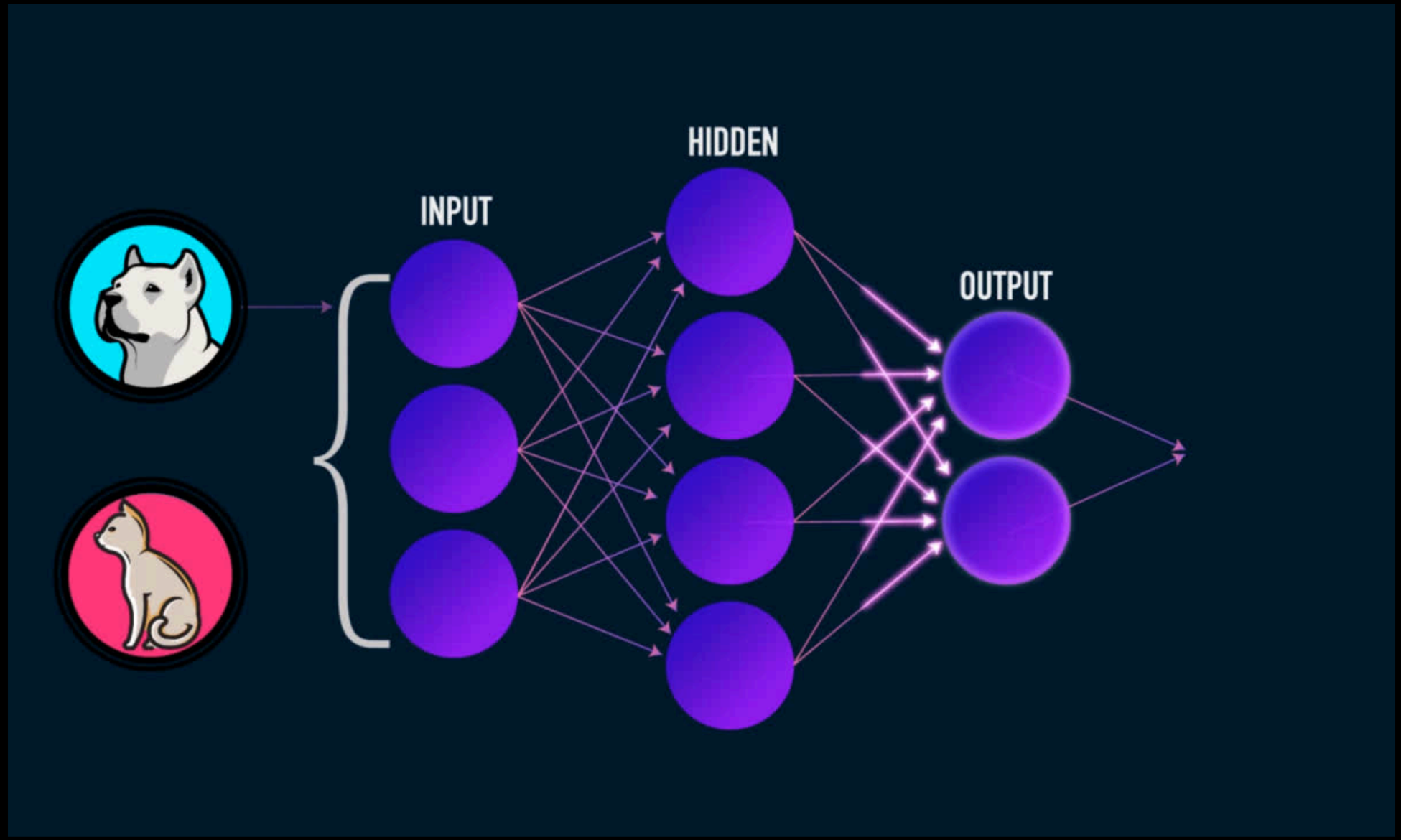
Machine Learning

Focuses on enabling computers to perform tasks without explicit programming.

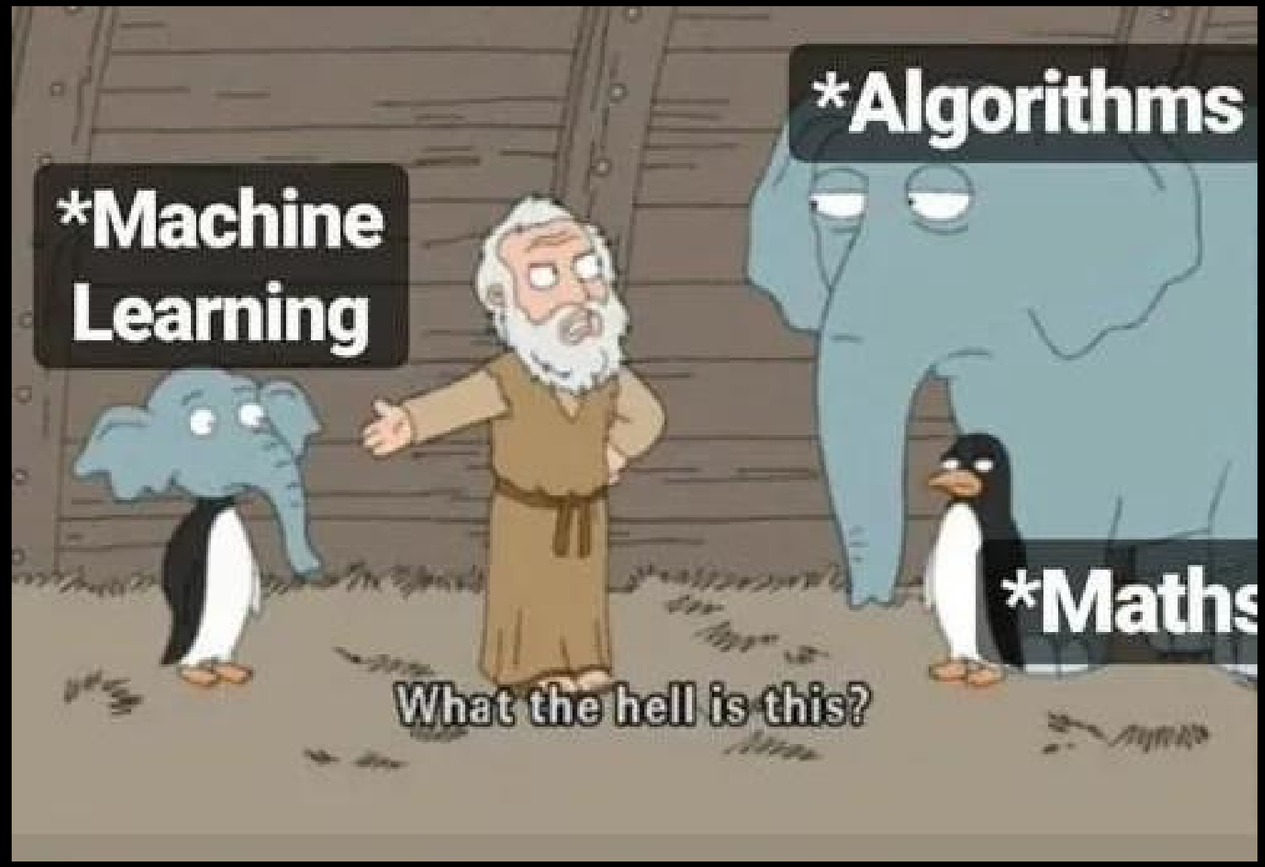
Deep Learning

A subset of machine learning based on artificial neural networks.



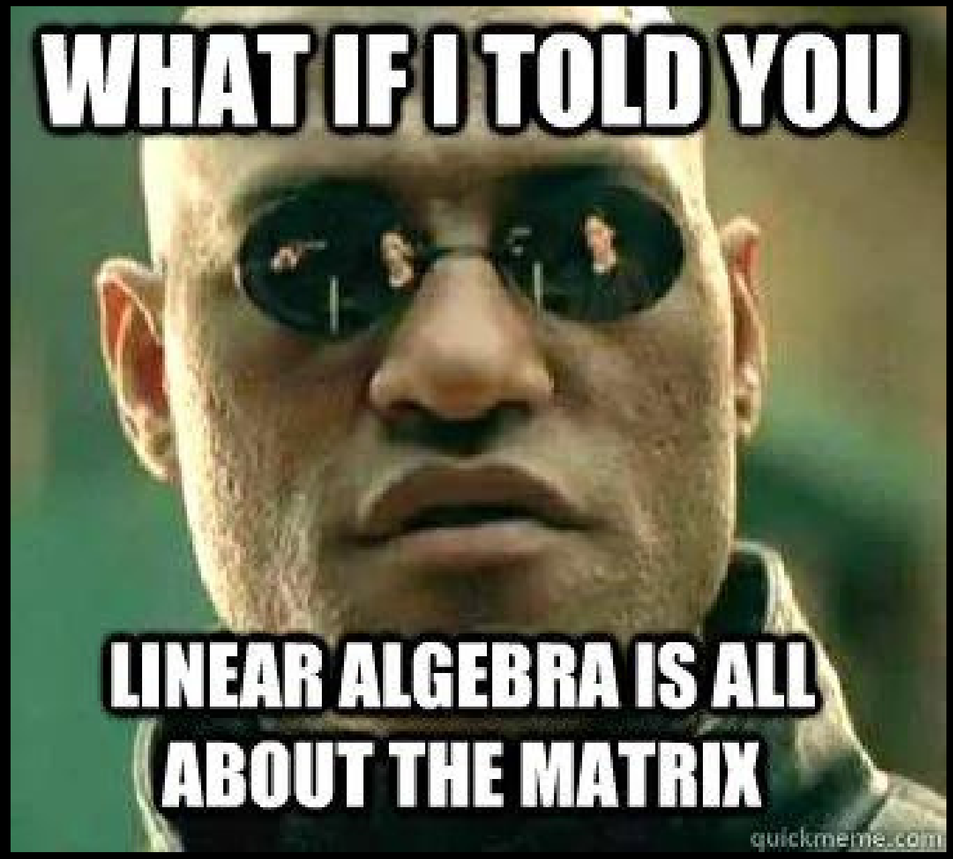

HIDDEN
INPUT
OUTPUT

- Model Theory
- Algorithms

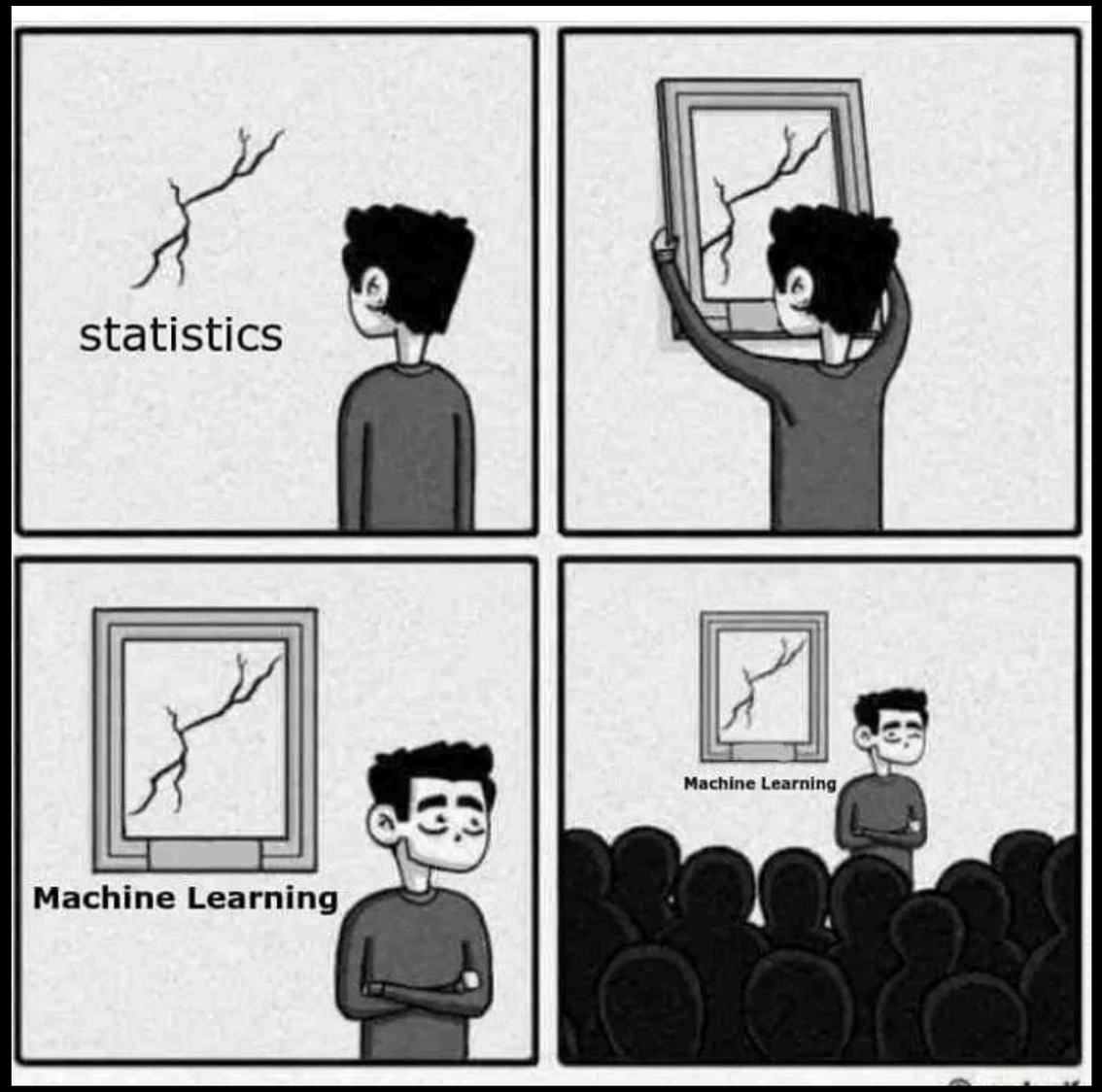


Optimization

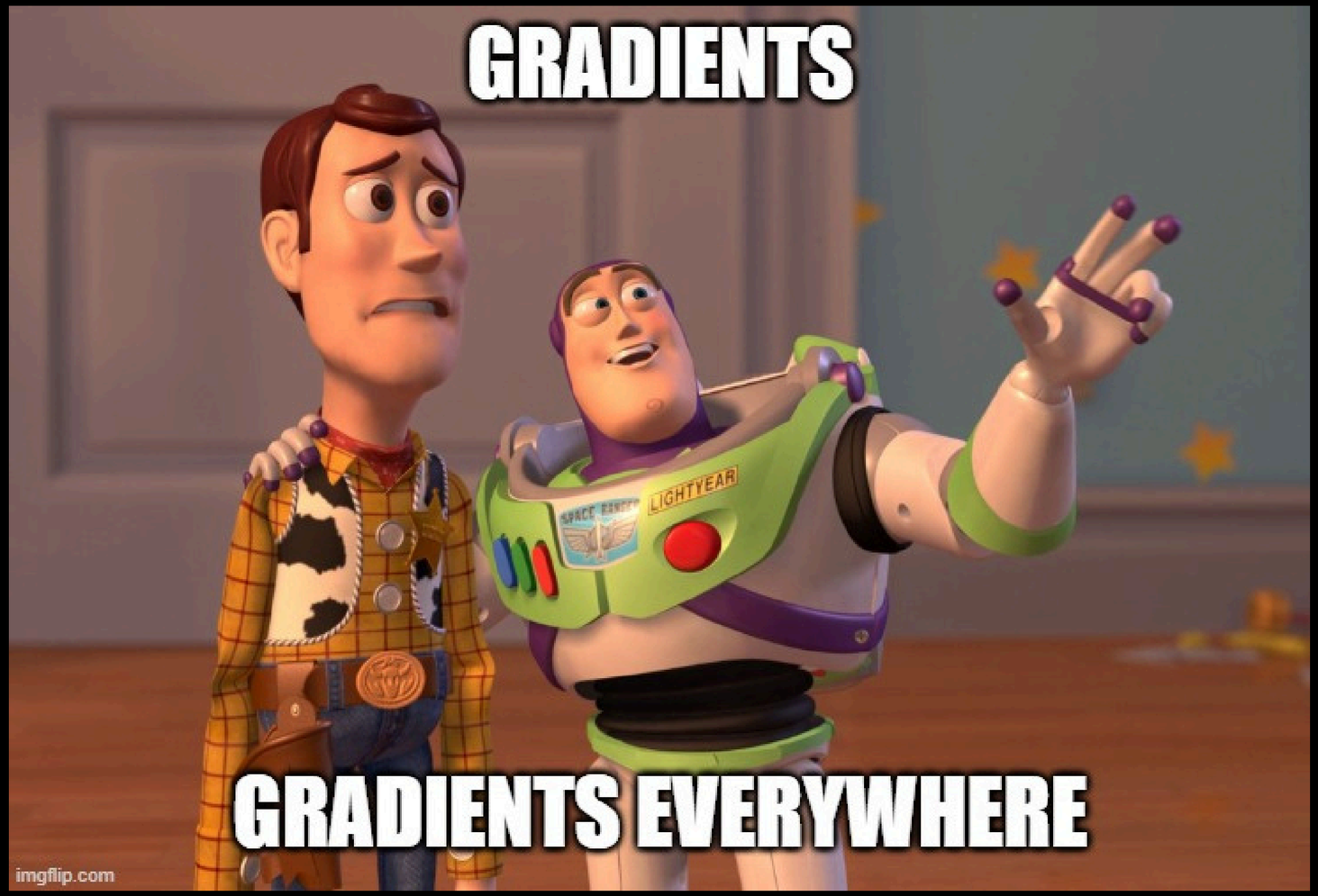
Linear Algebra and Matrix Calculations

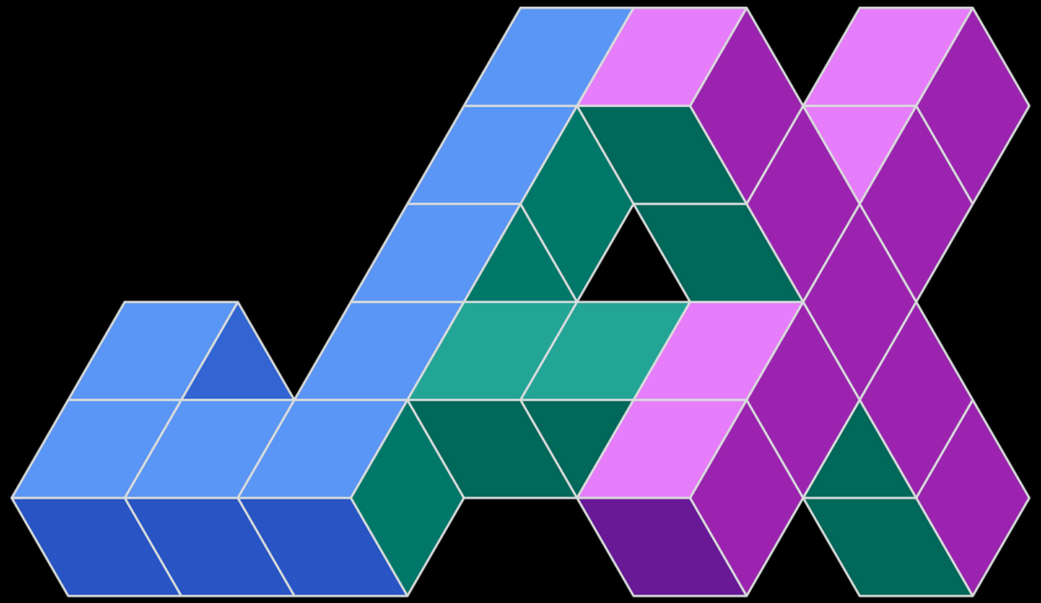


Statistics and Probability

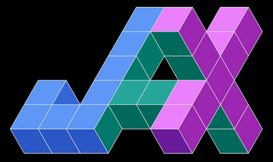


Gradients





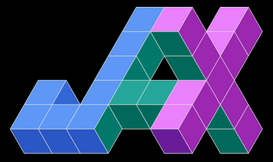
What is JAX?



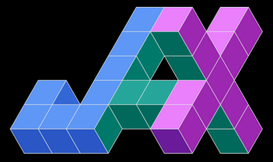
Google

OpenAI

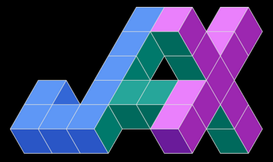
huggingface



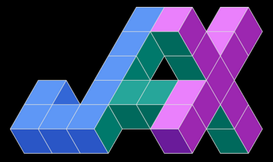
autograd



- **Manual:** We use our calculus knowledge and derive the derivatives by hand. The problem with this approach is that it is manual. It would take a lot of time for a Deep Learning researcher to derive the model’s derivatives by hand.



- **Symbolic**: We can obtain the derivatives via symbols and a program that can mimic the manual process. The problem with this approach is termed expression swell. Here the derivatives of a particular expression are exponentially longer (think chain rule) than the expression itself. This becomes quite difficult to track.



- **Numeric**: Here, we use the finite differences method to derive the derivatives.
- **Automatic**: The star ⭐ of the show.

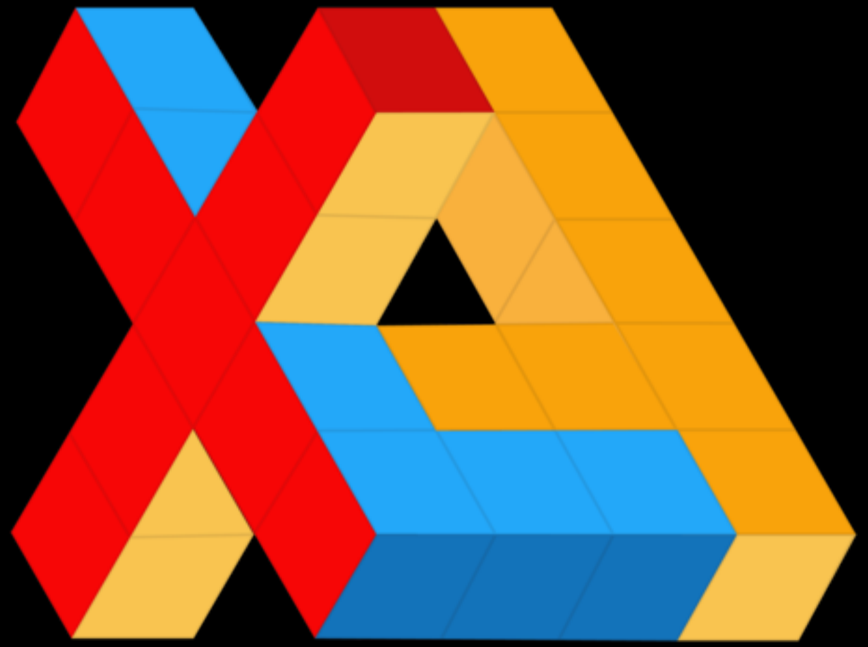
```
def func_square(x):
    # Return the square of the input
    return x**2
# Build a scalar input and pass it to the
# square function
x = 4.0
squared_x = func_square(x=x)
print(f"x => {x}\nx**2 => {squared_x}")
```

```
>>> x => 4.0
>>> x**2 => 16.0
```

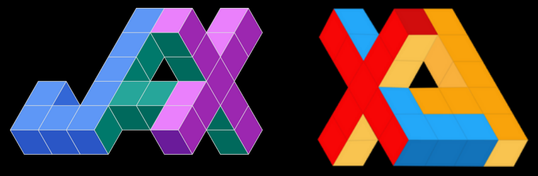
```python
from autograd import grad


# Compute the derivative of the square function
grad_func = grad(func_square)
point = 1.0
# Retrieve the gradient of the function at a particular point
print(f"Gradient of square func at {point} => {grad_func(1.0)}")
```

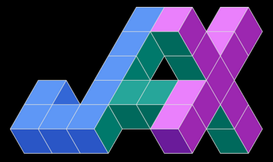
```
>>> Gradient of square func at 1.0 => 2.0
```



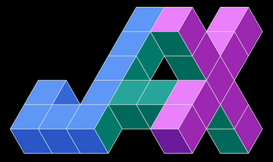
What is XLA?



Being device agnostic means that the same code can be run on different hardware (CPUs, GPUs, and TPUs).



> JAX is a high-performance, numerical computing library incorporating composable function transformations.



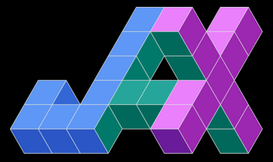
Install JAX

CPU

```
!pip install "jax[cpu]"
```

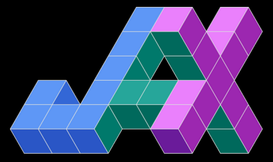
GPU

```
!pip install "jax[cuda]"
```



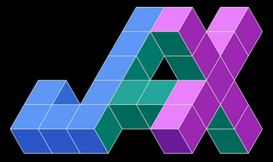
API Layering of JAX

NumPy



jax.numpy

jax.lax



Numerical Computation in JAX

```python
from jax import numpy as jnp

array = jnp.arange(0, 10, dtype=jnp.int8)
print(f"array => {array}")
```

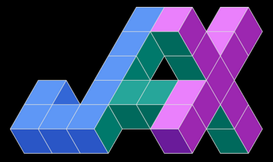
```
>>> array => [0 1 2 3 4 5 6 7 8 9]
```

```
try:
    print(jax.lax.add(jnp.float32(1), 2.0))
except Exception as ex:
    print(f"Type of exception => {type(ex).__name__}")
    print(f"Exception => {ex}")


>>> 3.0
```

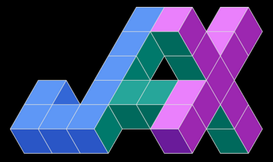
```python
# Checking the stricter `jax.lax` API 😭
try:
    jax.lax.add(1, 2.0)
except Exception as ex:
    print(f"Type of exception => {type(ex).__name__}")
    print(f"Exception => {ex}")
```

```
>>> Type of exception => TypeError
>>> Exception => lax.add requires arguments to have the same
dtypes, got int32, float32.
(Tip: jnp.add is a similar function that does automatic type
promotion on inputs).
```



“ “

jax.lax is a library of primitive operations that underpins libraries such as **jax.numpy** .

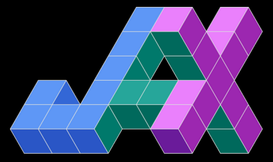


jax.grad

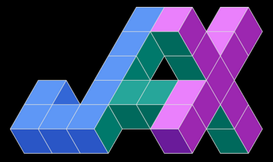
```
from jax import grad

def func(x):
    return x**2

d_func = grad(func)
d2_func = grad(d_func)
```



jax.jit



Speed: JIT (Just-In-Time) accelerates Python code by compiling it to run faster. This is especially useful for operations that involve loops and intensive computations.

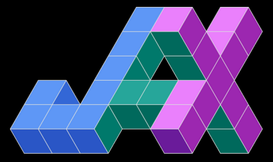
Gradient Computations: JIT can be used to speed up gradient computations. Gradient computations play a crucial role during the training of deep learning models, and JIT is helpful in speeding up this process.

NumPy: JAX provides an API similar to NumPy, making it possible to use JIT to accelerate NumPy code as well.

```
def funct(x):
    return x*(2+x)
c_funct = jax.jit(funct)

x = 2.0
result = c_funct(x)
print(result)
```

```
>>> 8.0
```



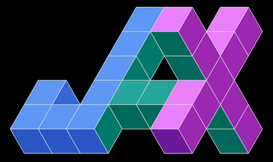
jax.vmap

```
def v_func(x):
    return x ** 2

matrix = jnp.array([[1, 2, 3],
                    [4, 5, 6]])
result = jax.vmap(v_func)(matrix)

print(result)
```

```
>>> [[ 1  4  9]
    [16 25 36]]
```



jax.pmap

```
import jax.tools.colab_tpu
jax.tools.colab_tpu.setup_tpu()
import jax
jax.devices()
```

```
[TpuDevice(id=0, process_index=0, coords=(0,0,0), core_on_chip=0),
 TpuDevice(id=1, process_index=0, coords=(0,0,0), core_on_chip=1),
 TpuDevice(id=2, process_index=0, coords=(1,0,0), core_on_chip=0),
 TpuDevice(id=3, process_index=0, coords=(1,0,0), core_on_chip=1),
 TpuDevice(id=4, process_index=0, coords=(0,1,0), core_on_chip=0),
 TpuDevice(id=5, process_index=0, coords=(0,1,0), core_on_chip=1),
 TpuDevice(id=6, process_index=0, coords=(1,1,0), core_on_chip=0),
 TpuDevice(id=7, process_index=0, coords=(1,1,0), core_on_chip=1)]
```

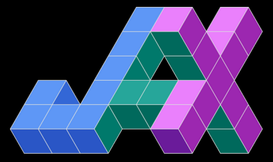
```
from jax import numpy as jnp
from jax import pmap
from jax import random

key = random.PRNGKey(42)
a = random.normal(key, shape=(3000,5000))
b = random.normal(key, shape=(5000,3000))
matrix_mul = lambda a, b: jnp.matmul(a, b)
matrix_mul(a, b).shape

>>> (3000, 3000)
```

```
n_devices = jax.local_device_count()
a = random.normal(key, shape=(n_devices, 3000, 5000))
b = random.normal(key, shape=(n_devices, 5000, 3000))
parallel_matrix_mul = pmap(matrix_mul)
parallel_matrix_mul(a, b).shape
```

```
>>> (8, 3000, 3000)
```



How Does JAX Handle Randomness?

```python
import numpy as np
# random number generation using numpy
np.random.seed(42)
rn1 = np.random.normal()
rn2 = np.random.normal()
print(f"NumPy Random Number Generation: {rn1: .2f} {rn2: .2f}")
```

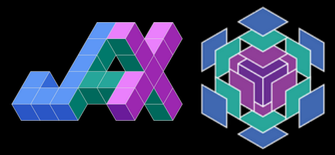
```
>>> NumPy Random Number Generation:  0.50 -0.14
```

```python
from jax import random
key = random.PRNGKey(65)
print(key)
jrn1 = random.normal(key)
jrn2 = random.normal(key)

print(f"JAX Random Number Generation: {jrn1: .2f} {jrn2: .2f}")
```

```
>>> [ 0 65]
>>> JAX Random Number Generation:  0.05  0.05
```

What is Flax?

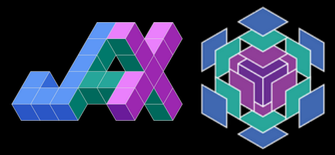


Transformers & Multi-Head Attention

Pytorch vs JAX

Models	PyTorch	JAX
Reverse Sequence	0min 26sec	0min 7sec
Anomaly Detection	16min 34sec	3min 45sec

(NVIDIA RTX3090, 24 core CPU)

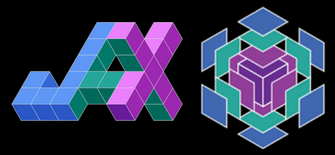


GoogleNet, ResNet, DenseNet

Pytorch vs JAX

Models	PyTorch	JAX
GoogleNet	53min 50sec	16min 10sec
ResNet	20min 47sec	7min 51sec
Pre-Activation ResNet	20min 57sec	8min 25sec
DenseNet	49min 23sec	20min 1sec

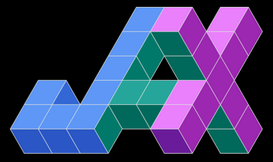
(NVIDIA RTX3090, 24 core CPU)



Deep Autoencoders

Modeller	PyTorch	JAX
AE - 64 gizli	13dk 10sn	7dk 10sn
AE - 128 gizli	13dk 11sn	7dk 10sn
AE - 256 gizli	13dk 11sn	7dk 11sn
AE - 384 gizli	13dk 12sn	7dk 14sn

(NVIDIA RTX3090, 24 core CPU)

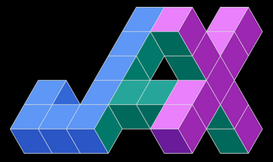


Sources

/pyimagesearch.com

/jax.readthedocs.io

/github.com/google/jax



Thank you :)

/rumeysskara

/rumeysskara

/rumeysakara

Rümeysa Kara

Data Scientist
Google Developer Expert