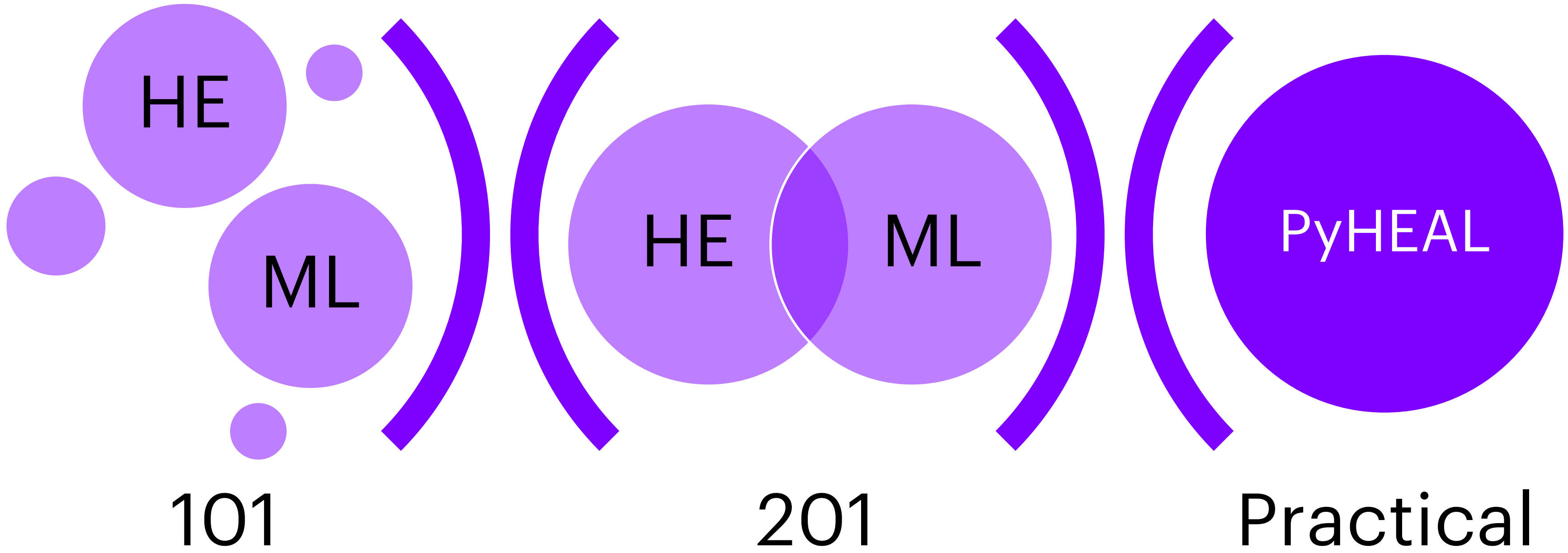
HOW TO BUILD MACHINE LEARNING ALGORITHMS USING HOMOMORPHIC ENCRYPTION

LUIZ PIZZATO, PHD

ACCENTURE LIQUID STUDIO ANZ

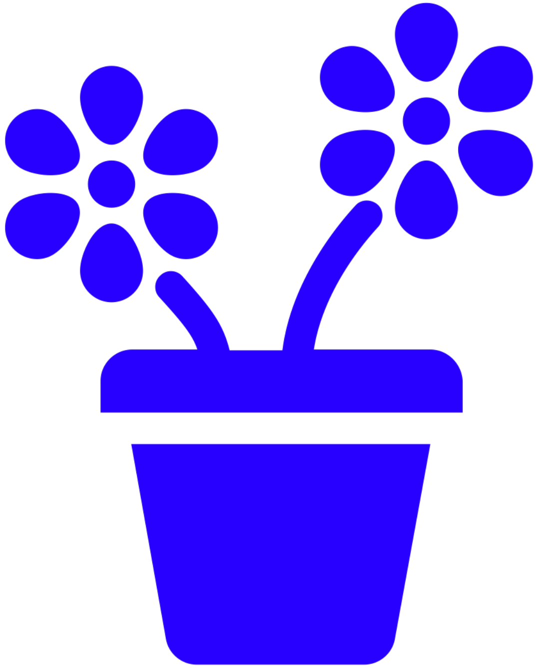
accenture

WHAT TO EXPECT FROM TODAY



WHAT NOT TO EXPECT FROM TODAY

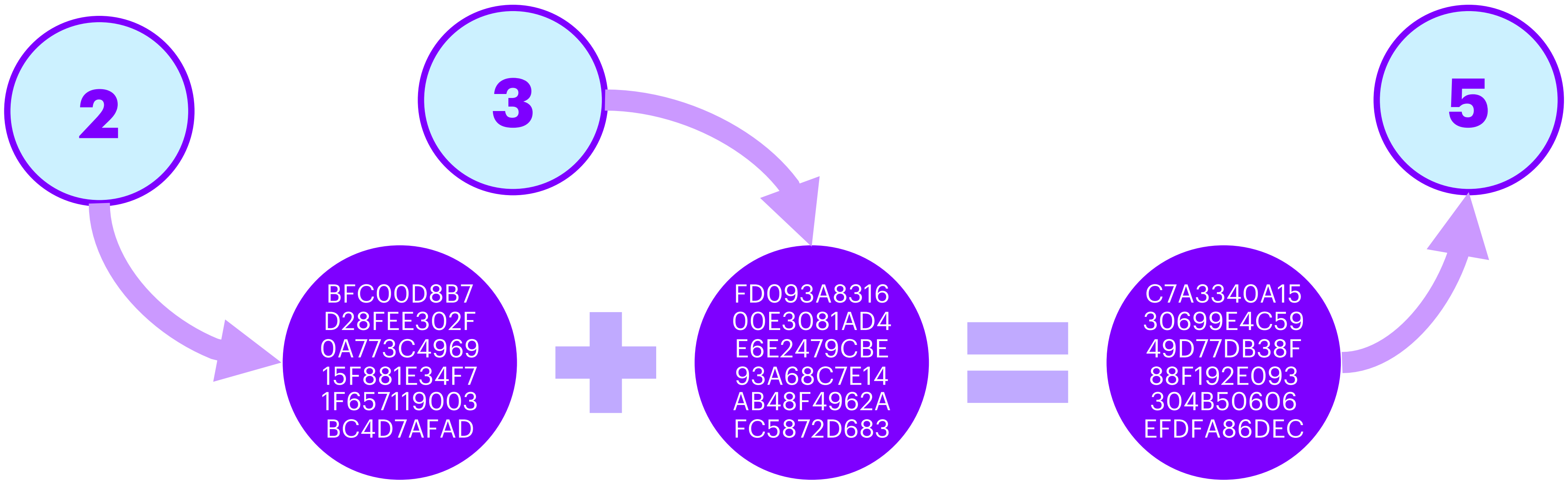
WHY DO YOU WANT TO HAVE HE+ML?



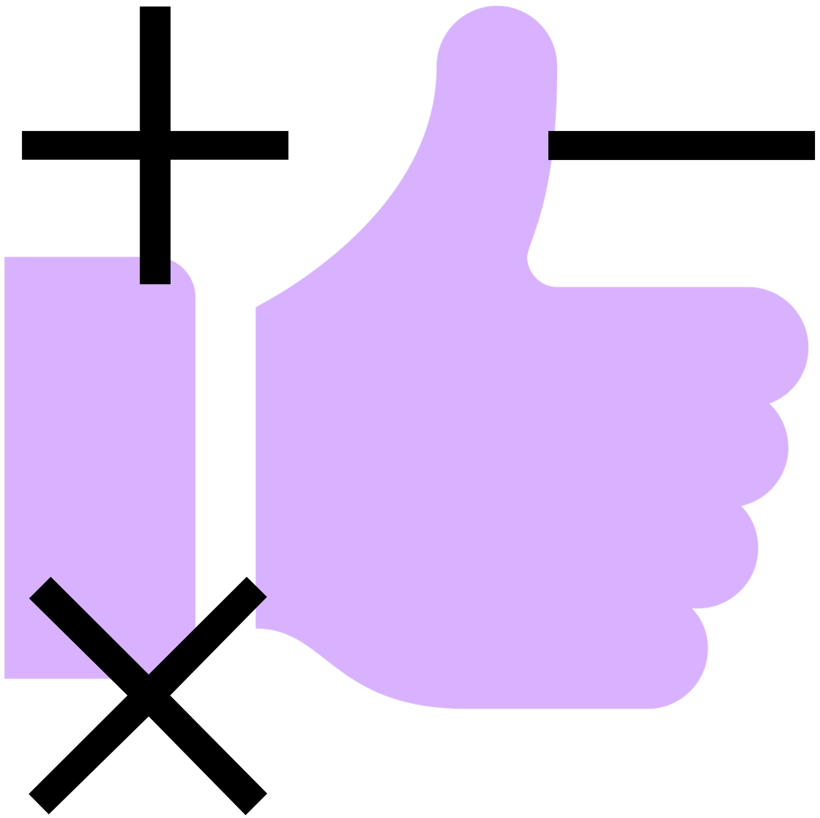
HE+ML IS EASY

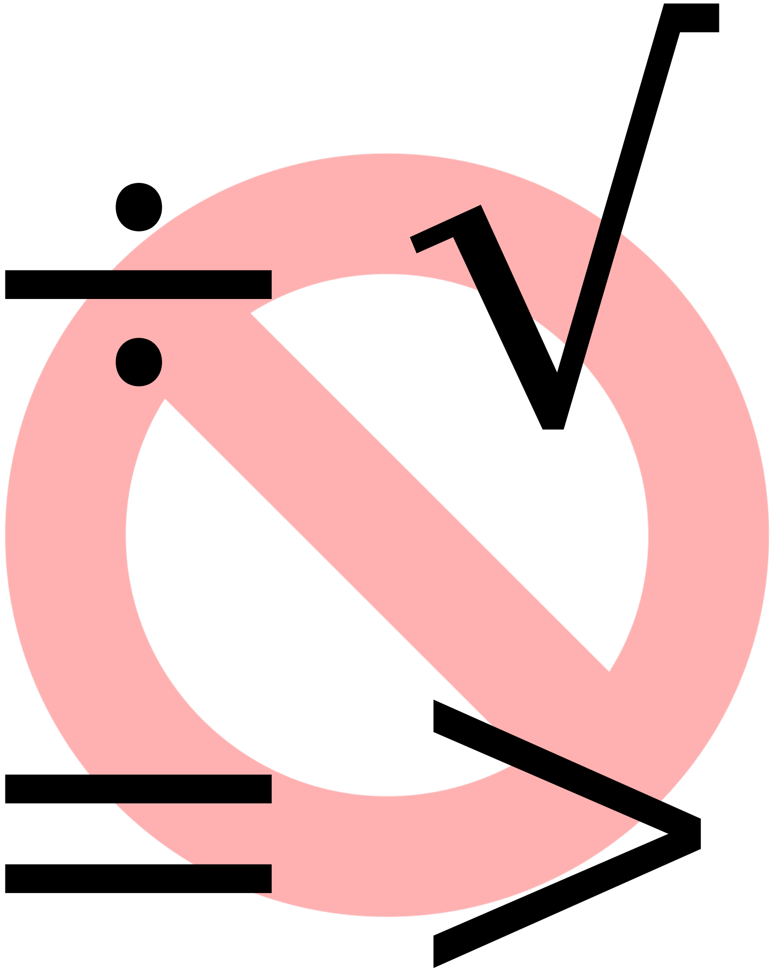
Homomorphic encryption is a form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. The purpose of homomorphic encryption is to allow computation on encrypted data.

Homomorphic encryption is a form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext. The purpose of homomorphic encryption is to allow computation on encrypted data.



LIMITED OPERATIONS





STRONG PUBLIC KEY ENCRYPTION

Two Part Key – Public and Private Keys

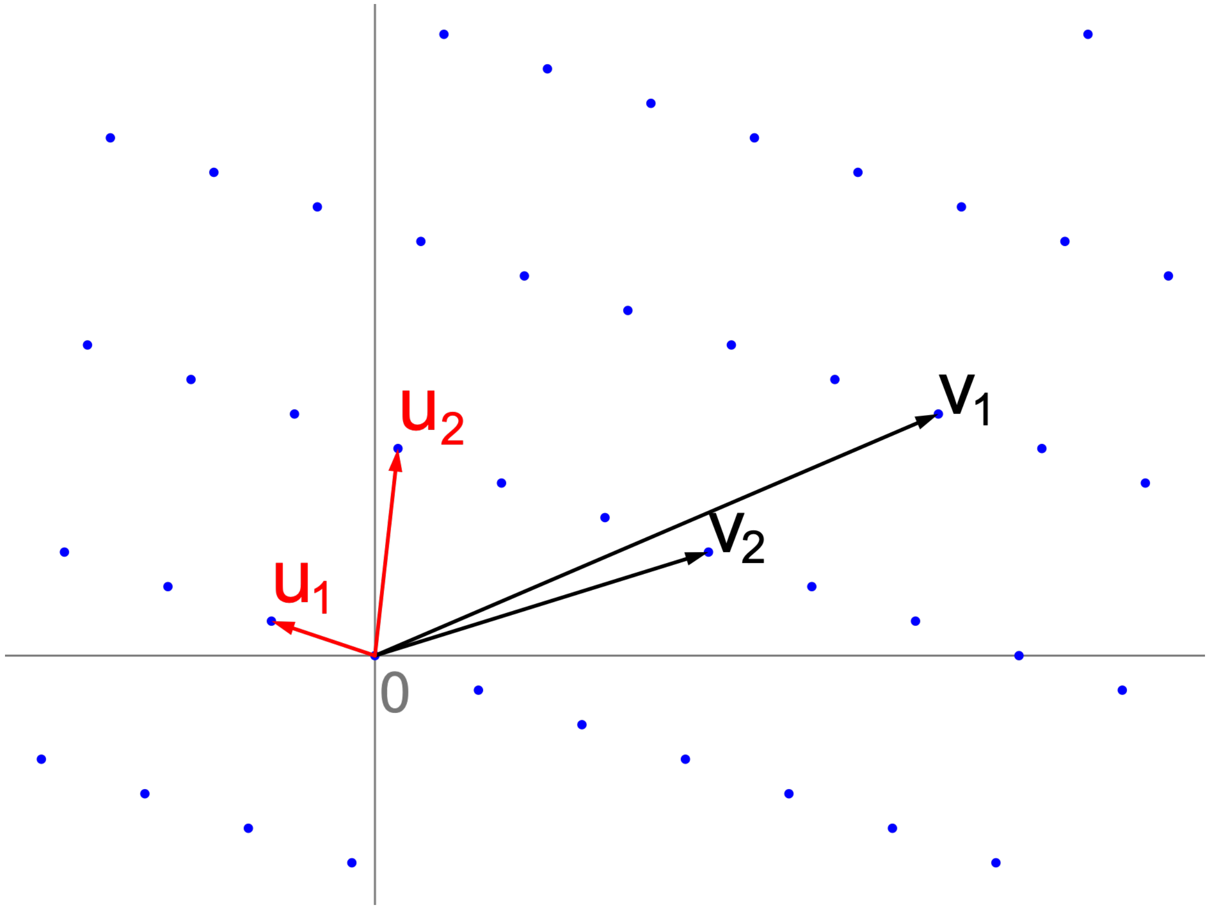
- Public key used for encryption
- Private/Secret key for decryption

Private key cannot be (practically) derived from public key information

- Base on hard mathematical problems that is very hard to find the reverse of the trapdoor function
  - RSA relies on how difficult is to factorise the product of two large prime numbers
  - RSA is a simple algorithm but computationally hard to reverse:

Check the simplicity of the RSA algorithm: (https://en.wikipedia.org/wiki/RSA_(cryptosystem))

LATTICE-BASED ENCRYPTION



Lots of hard lattice problems

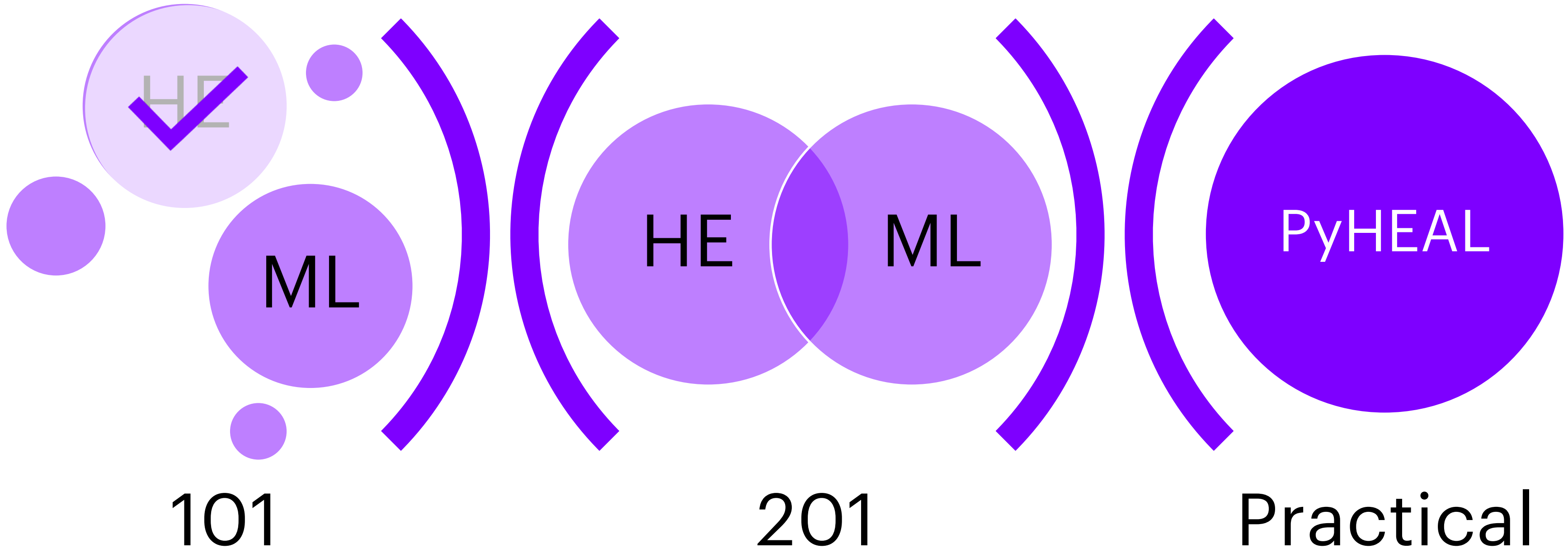
No quantum solution

Numbers represented as polynomials

Example number 1025 as polynomials

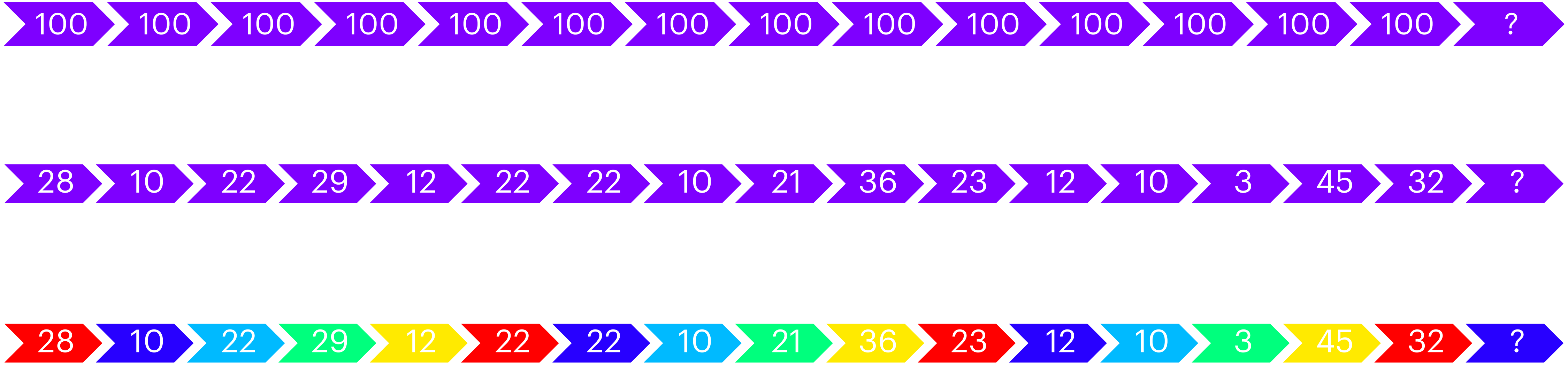
x=10	$x^3 + 2x^1 + 5x^0$	$(1,0,2,5)_{10}$
x=2	$x^{10} + x^0$	$(1,0,0,0,0,0,0,0,0,0,1)_2$
x=3	$x^6+x^5+x^3+2x^2+2x^1+2x^0$	$(1,1,0,1,2,2,2)_3$

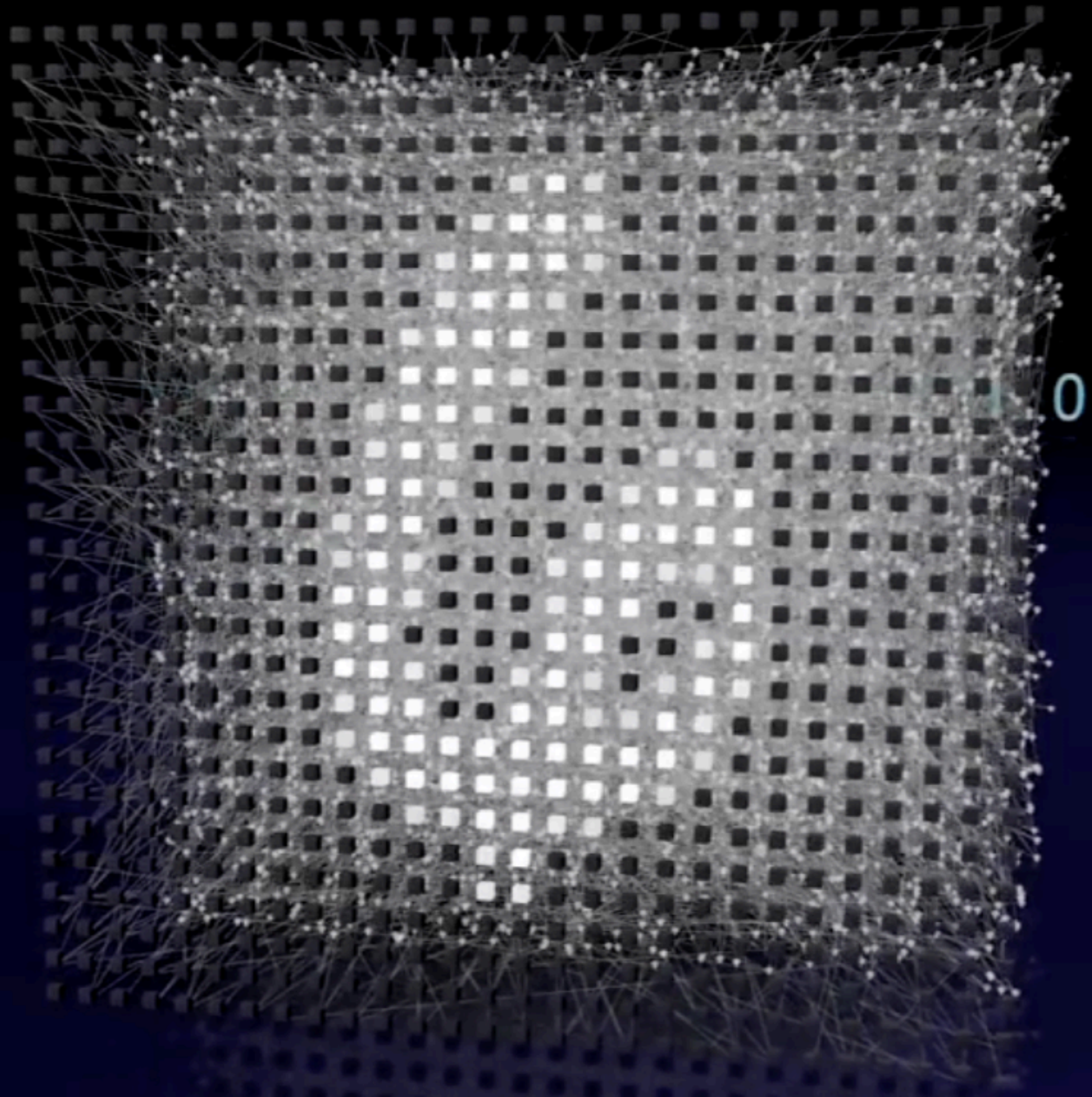
WHAT'S NEXT?



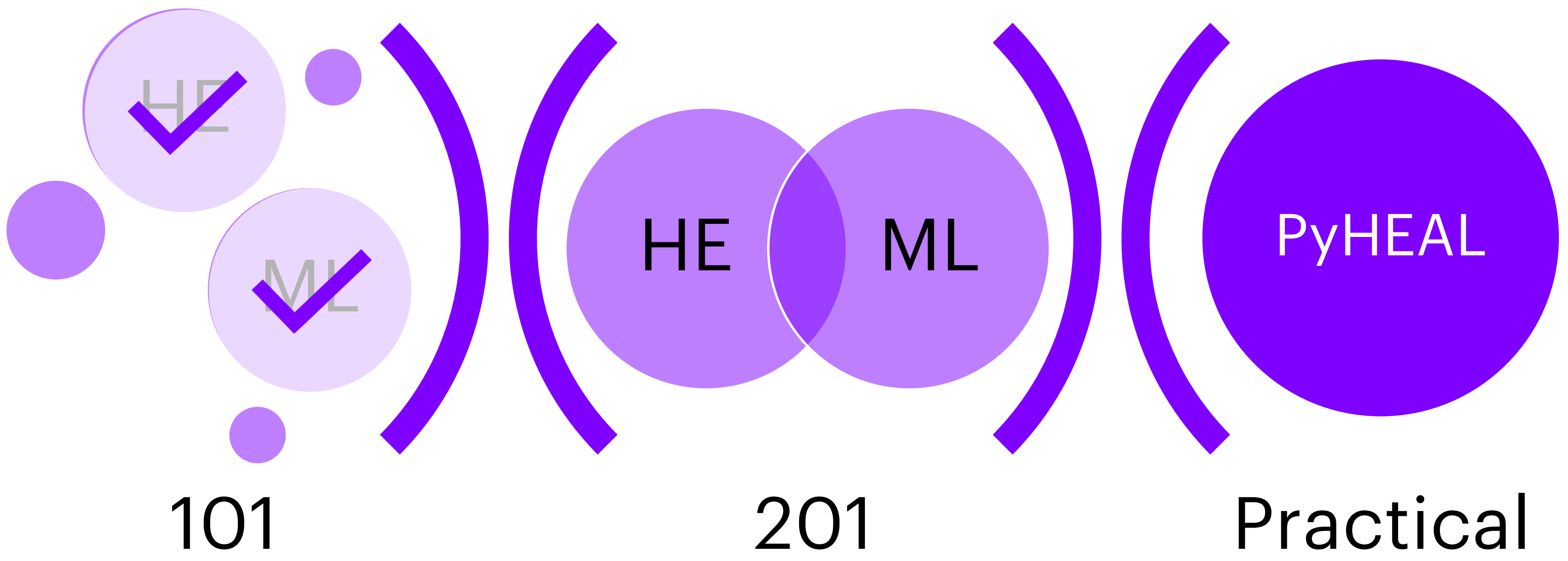
Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.[2]

Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.[2]




0

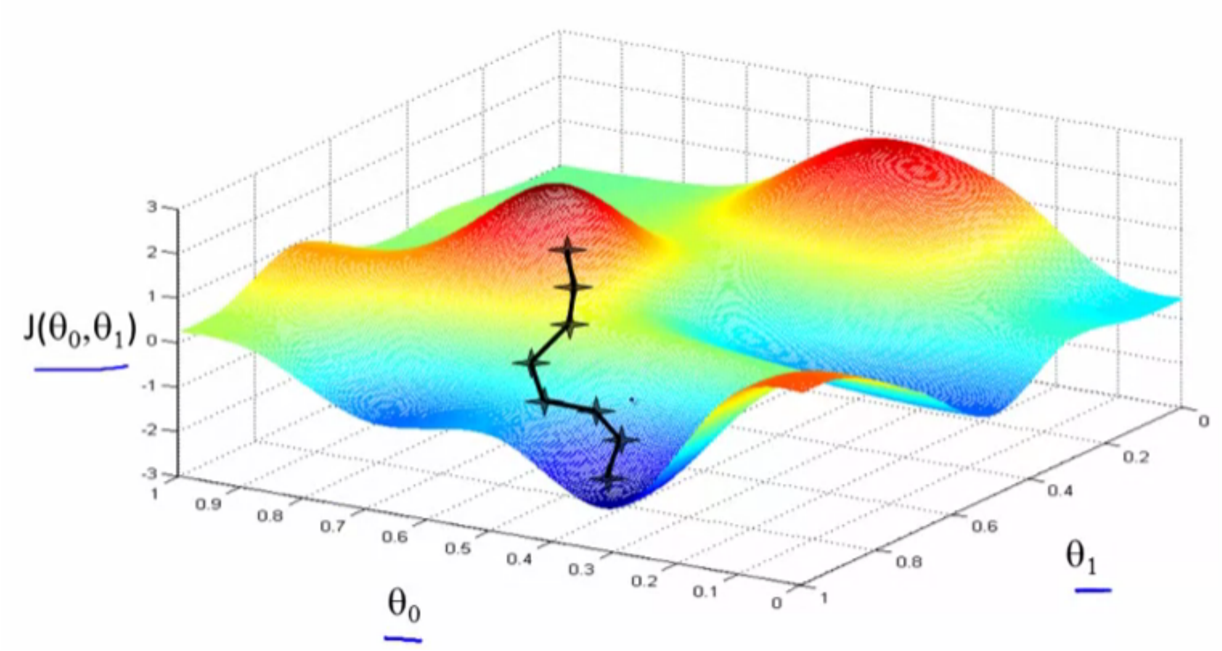
WHAT'S NEXT?

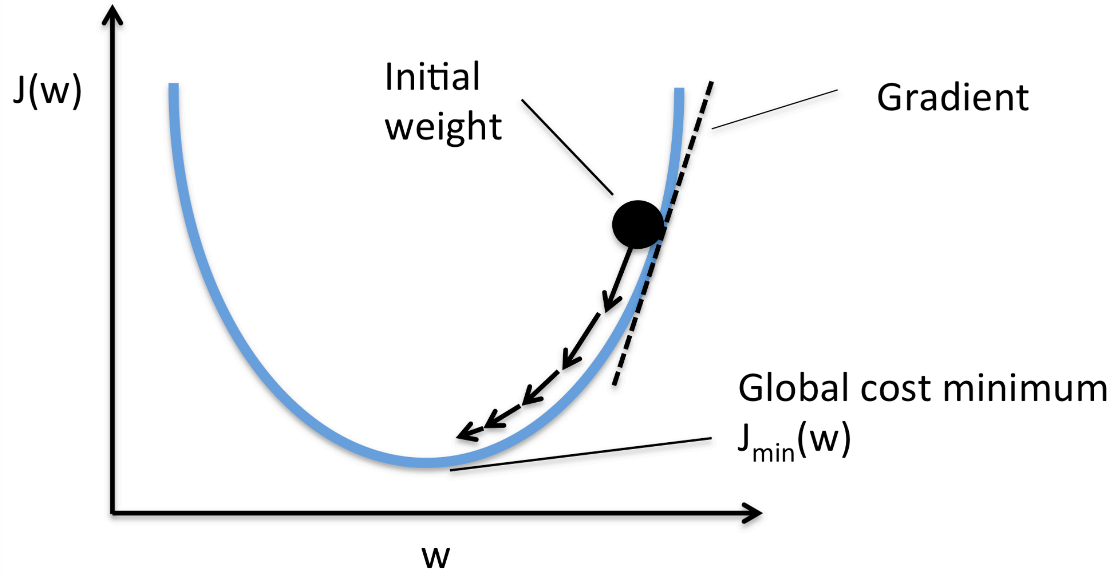


MACHINE LEARNING ALGORITHMS

- Linear Models
- SVM
- Tree-Based Methods
- Neighbourhood-Based Methods
- Naive Bayes
- Neural Networks
- Matrix Factorisation
- Ensemble Methods

GRADIENT DESCENT





Minimise cost/loss function

- **Regression:** $$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- **Classification:** $$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

GRADIENT DESCENT – REGRESSION

Model: $h_{\Theta}(x) = \Theta_0 b + \Theta^T x$

Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Update rules:

$$\theta_0 \coloneqq \theta_0 - \alpha \frac{d}{d\theta_0} J(\theta_0, \theta_1)$$

$$\theta_1 \coloneqq \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{d}{d\theta_0} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\frac{d}{d\theta_1} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

GRADIENT DESCENT - CLASSIFICATION

Model: $h_\Theta(x) = \sigma(\Theta_0 b + \Theta^T x)$

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

Sigmoid Activation : $\sigma(z) = \frac{1}{1+e^{-z}}$

$$\frac{\partial}{\partial\theta_j}J(\theta) = \sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i$$

Solution:

Replace sigmoid activation by its Taylor polynomial approximation

$$\sigma(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + \frac{31}{1451520}x^9 \,.$$

MACHINE LEARNING ALGORITHMS

- Linear Models
- SVM
- Tree-Based Methods
- Neighbourhood-Based Methods
- Naive Bayes
- Neural Networks
- Matrix Factorisation
- Ensemble Methods

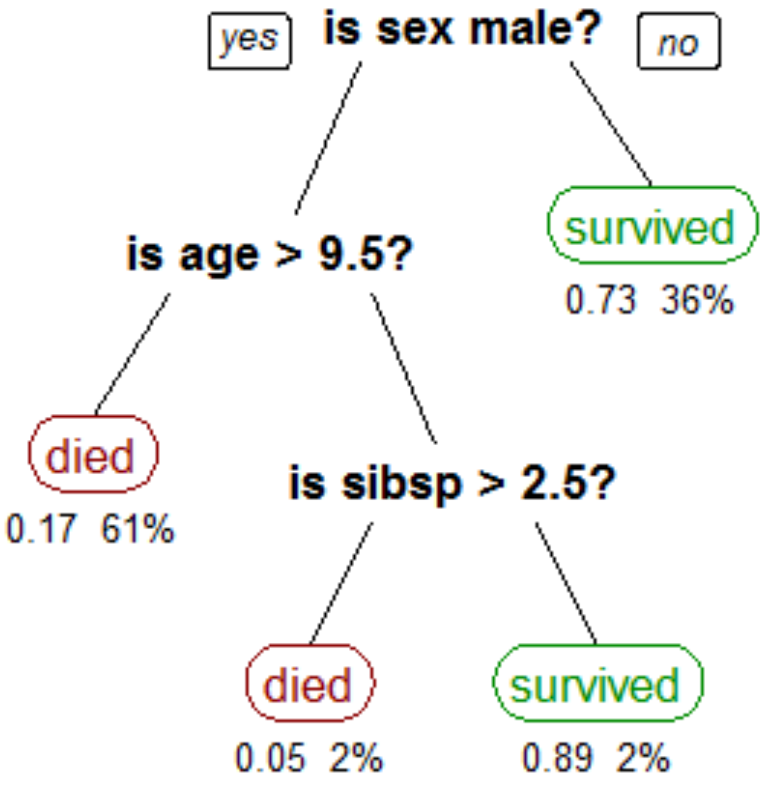
NAÏVE BAYES

$$\mathrm{y} = {}_{i=1..n}argmax\ P(class_i) \prod_{j=1..f} P(x_j|class_i)$$

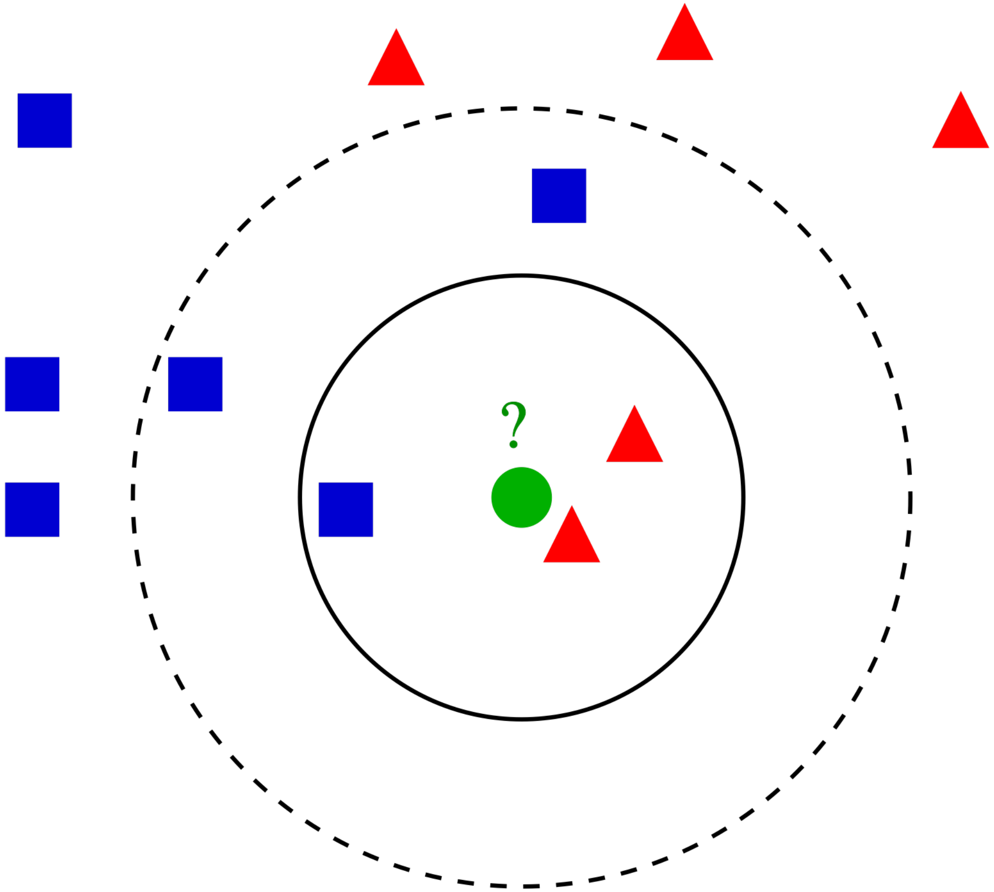
$$\mathrm{y} = {}_{i=1..n}argmax\ LogProb(class_i) + \sum_{j=1..f} LogProb(x_j|class_i)$$

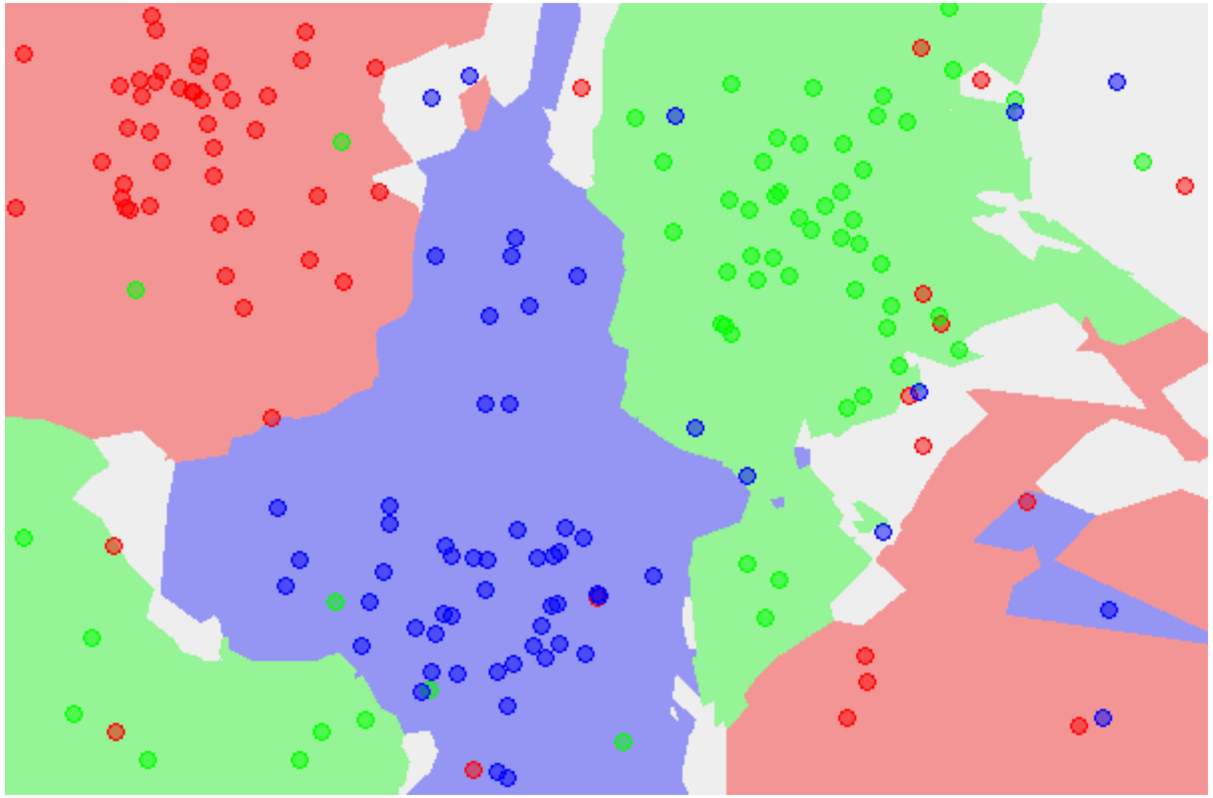
$$\log(1+x) = \sum_{n=1}^{\infty}(-1)^{n+1}\frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots.$$

TREE-BASED METHODS



NEIGHBOURHOOD-BASED METHODS



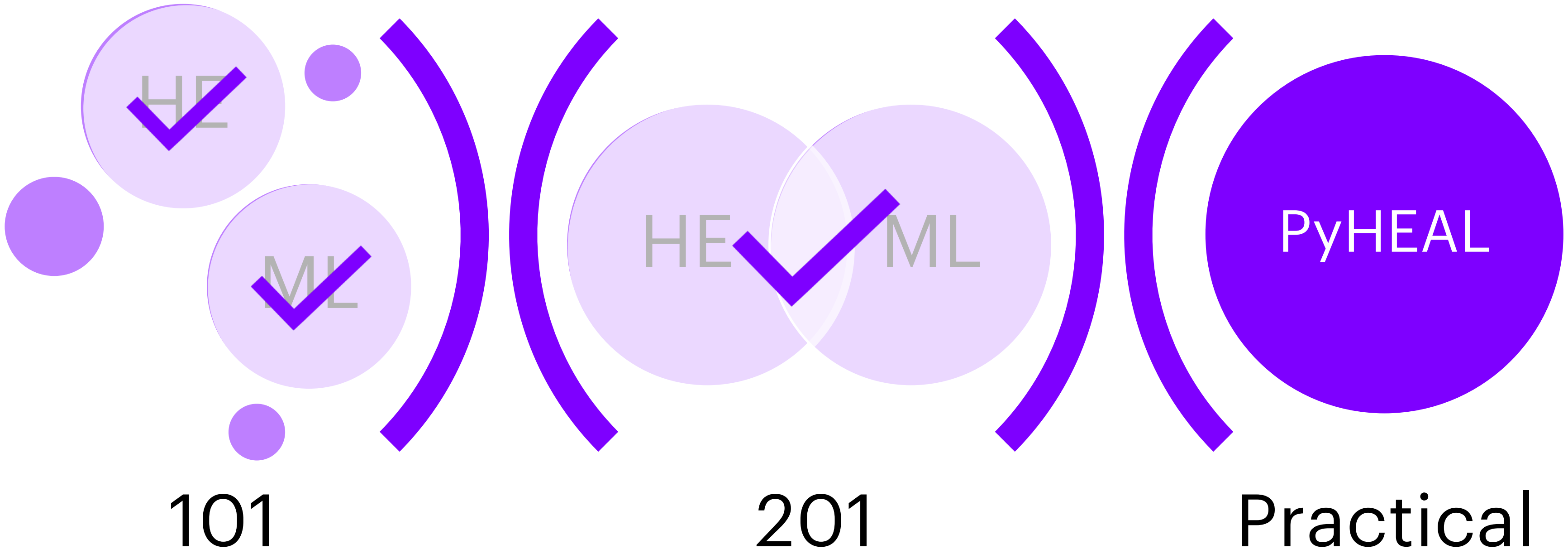


MACHINE LEARNING ALGORITHMS

- Linear Models
- SVM
- Tree-Based Methods
- Neighbourhood-Based Methods

- Naive Bayes
- Neural Networks
- Matrix Factorisation
- Ensemble Methods

WHAT'S NEXT?



HE LIBRARIES

Sathya, S.S., Vepakomma, P., Raskar, R., Ramachandra, R., & Bhattacharya, S. (2018). A Review of Homomorphic Encryption Libraries for Secure Computation. *CoRR, abs/1812.02428*.

Basic Features	SEAL	HElib	TFHE	Paillier	ELGamal	RSA
Asymmetric	Yes	Yes	Yes	Yes	Yes	Yes
Serialization and Deserialization of keys and cipher-texts	Yes	Yes	Yes	No	No	No
Negative computations support	Yes	No	[illegible]	[illegible]	[illegible]	[illegible]
Ciphertext size (less than 1MB for 1 input)	No	No	[illegible]	[illegible]	[illegible]	[illegible]
Can run on less than 2GB RAM	No	Yes	[illegible]	[illegible]	[illegible]	[illegible]

Table 1. Comparison of Homomorphic

Advanced Features	SEAL	HElib	[illegible]	[illegible]	[illegible]	[illegible]
Noise affected after each computation	Yes	Yes	Yes	No	No	No
Recryption	No	Yes	Yes	N/A	N/A	N/A
Ciphertext packing	Yes	Yes	No	No	No	No
Relinearization	Yes	Yes	No	N/A	N/A	N/A
Multithreading	Yes	Yes	No	No	No	No

Table 2. Comparison of Homomorphic Encryption Libraries based on advanced features

Operations	SEAL	HElib	TFHE	Paillier	ELGamal	RSA
Addition, Subtraction	Yes	Yes	Yes	Yes	No	No
Multiplication	Yes	Yes	Yes	No	Yes	Yes
[illegible]	[illegible]	[illegible]	[illegible]	No	No	No
[illegible]	[illegible]	[illegible]	[illegible]	No	No	No
[illegible]	[illegible]	[illegible]	[illegible]	No	No	No
[illegible]	[illegible]	[illegible]	[illegible]	No	No	No
[illegible]	[illegible]	[illegible]	[illegible]	No	No	No
Exponentiation	Yes	Yes	No	No	No	No
Square	Yes	Yes	Yes	No	Yes	Yes
Negation	Yes	Yes	No	No	No	No
Add Plain, Subtract Plain, Multiply Plain	Yes	No	No	No	No	No

Table 3. Different operations supported by Homomorphic Encryption libraries

Languages	SEAL	HElib	TFHE	Paillier	ELGamal	RSA
C++	Yes	Yes	No	Yes	Yes	Yes
Python	Yes	Yes	No	Yes	Yes	Yes
Java	No	No	No	Yes	Yes	Yes
C	No	No	Yes	No	No	No

Table 4. Homomorphic Library implementations across programming languages

HE LIBRARIES

https://github.com/NervanaSystems/he-transformer

HE Transformer for nGraph

The **Intel® HE transformer for nGraph™** is a Homomorphic Encryption (HE) backend to the **Intel® nGraph Compiler**, Intel's graph compiler for Artificial Neural Networks.

Homomorphic encryption is a form of encryption that allows computation on encrypted data, and is an attractive remedy to increasing concerns about data privacy in the field of machine learning. For more information, see our paper.

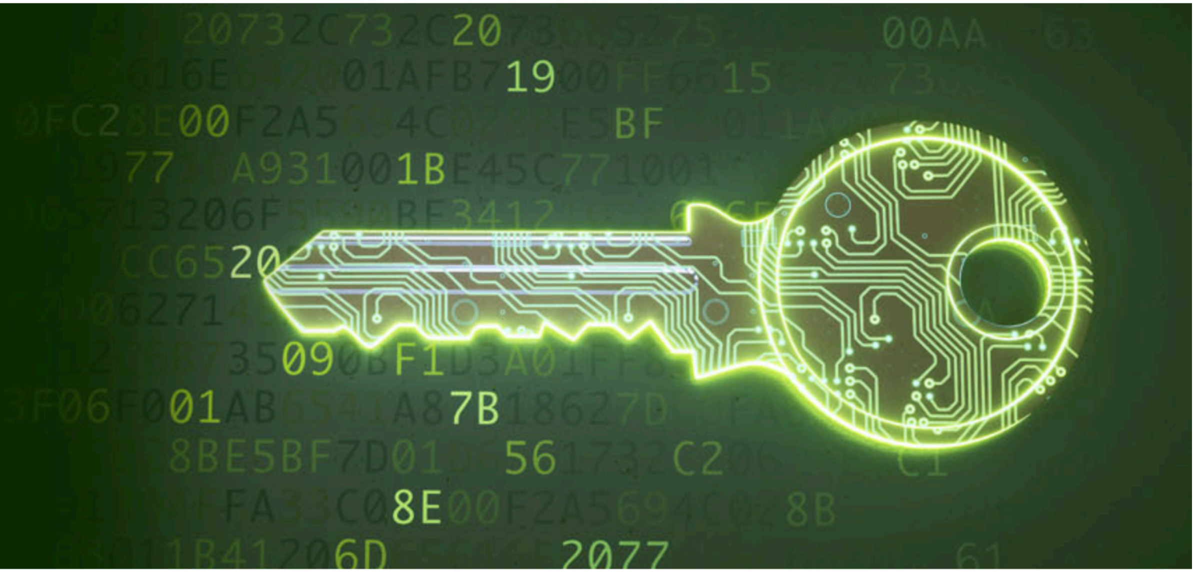
This project is meant as a proof-of-concept to demonstrate the feasibility of HE on local machines. The goal is to measure performance of various HE schemes for deep learning. This is **not** intended to be a production-ready product, but rather a research tool.

Currently, we support the CKKS encryption scheme, implemented by the Simple Encrypted Arithmetic Library (SEAL) from Microsoft Research.

Additionally, we integrate with the **Intel® nGraph™ Compiler and runtime engine for TensorFlow** to allow users to run inference on trained neural networks through Tensorflow.

Microsoft | Research   Research areas   Researcher tools   Programs & Events   Careers   People   Blogs & Podcasts   Labs & Locations   All Microsoft   Search

Microsoft SEAL



Overview   People   Publications   Videos   Articles   News

Microsoft SEAL—powered by open-source homomorphic encryption technology—provides a set of encryption libraries that allow computations to be performed directly on encrypted data. This enables software engineers to build end-to-end encrypted data storage and computation services where the customer never needs to share their key with the service.

Microsoft SEAL is open-source (MIT license). Start using it today!

Download

Citing Microsoft SEAL | Contact us

pyHeal

This project implements Python wrappers for Homomorphic Encryption libraries, aimed at being more Python friendly.

It currently contains:
- A pybind11 based Python wrapper for Microsoft SEAL in `seal_wrapper`
- A Pythonic wrapper for `seal_wrapper` in `pyheal/wrapper.py`
- A Python ciphertext type of object that allows math operations as if they were python numbers in `pyheal/ciphertext_op.py`
- A standard encoder/decoder interface for seal encoders and encryptors for use of the `CiphertextOp` objects in `pyheal/encoders.py`.

Tests:
- A partial re-implementation of Microsoft SEAL's examples using `wrapper.py` in `tests.py`
- A large number of tests for PyHEAL and `CiphertextOp` in `pyheal/test_pyheal.py`

Setup

Clone using:
Git v2.13+: `git clone --recurse-submodules (repository URL)`

Git v1.6.5 - v2.12: `git clone --recursive (repository URL)`

For a repository that has already been cloned or older versions of git run:
`git submodule update --init --recursive`

Build

This project can be built directly using `pip3`.
Optionally create and activate a new Python virtual environment using `virtualenv` first, for example:

```
python3 -m virtualenv ./venv --python python3

#Linux
source ./venv/bin/activate

#Windows
#venv\Scripts\activate
```

Install dependencies and package:

```
pip3 install .
```

Usage

```
import pyheal


# Set encryption params + obtain an EncryptorOp object
...
encryptor = EncryptorOp(...)
decryptor = Decryptor(...)

v1 = encryptor_encoder.encode(10)
v2 = encryptor_encoder.encode(20)

result = v1 + v2

print(decryptor.decrypt(result)) # Prints 30 after decrypt
```

See example_usage.py for more usage examples.

Jupyter Notebook Demo

PyHEAL Demo

Library import

In [1]:
```
from pyheal import wrapper
from pyheal import encoders
```

HE scheme initialisation

```
In [2]: def get_encryptor_decryptor():
            """
                Return an encryptor and a decryptor object for the same scheme
            """

            scheme = 'BFV'

            poly_modulus = 1 << 12
            coeff_modulus_128 = 1 << 12
            plain_modulus = 1 << 10

            parms = wrapper.EncryptionParameters(scheme_type=scheme)

            parms.set_poly_modulus(poly_modulus)
            parms.set_coeff_modulus(wrapper.coeff_modulus_128(coeff_modulus_128))
            parms.set_plain_modulus(plain_modulus)

            seal_context_ = wrapper.Context(parms).context

            keygen = wrapper.KeyGenerator(seal_context_)

            plaintext_encoder = encoders.PlainTextEncoder(
                encoder=wrapper.FractionalEncoder(smallmod=wrapper.SmallModulus(plain_modulus),
                                                  poly_modulus_degree=poly_modulus,
                                                  integer_coeff_count=64,
                                                  fraction_coeff_count=32,
                                                  base=2)
            )

            encryptor_encoder = encoders.EncryptorOp(plaintext_encoder=plaintext_encoder,
                                                     encryptor=wrapper.Encryptor(ctx=seal_context_, public=keygen.public_key()
        ),
                                                     evaluator=wrapper.Evaluator(ctx=seal_context_),
                                                     relin_key=keygen.relin_keys(decomposition_bit_count=16, count=2)
                                                     )


            decryptor_decoder = encoders.Decryptor(plaintext_encoder=plaintext_encoder,
                                                   decryptor=wrapper.Decryptor(ctx=seal_context_, secret=keygen.secret_key())
                                                   )

            return encryptor_encoder, decryptor_decoder

        encryptor_encoder, decryptor_decoder = get_encryptor_decryptor()
```

Simple operations

In [3]:
```
a = 10
b = 20
r = a + b
r
```

Out[3]:
```
30
```

In [4]:
```
a = encryptor_encoder.encode(10)
b = encryptor_encoder.encode(20)
r = a + b
r, decryptor_decoder.decode(r)
```

Out[4]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x10ed56d00>, 30.0)
```

List operations

```
In [5]: import numpy as np
```

```
In [6]: numbers = list(np.random.randint(-100,100,10))
        numbers
```

```
Out[6]: [-89, 56, -56, 5, -28, 56, -88, -76, 82, 5]
```

```
In [7]: sum(numbers)
```

```
Out[7]: -133
```

```
In [8]: enumbers = encryptor_encoder.encode(numbers)
        enumbers
```

```
Out[8]: [<pyheal.ciphertext_op.CiphertextOp at 0x110639990>,
         <pyheal.ciphertext_op.CiphertextOp at 0x1106399e8>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639a40>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639a98>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639af0>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639b48>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639ba0>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639bf8>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639c50>,
         <pyheal.ciphertext_op.CiphertextOp at 0x110639ca8>]
```

```
In [9]: sum(enumbers), decryptor_decoder.decode(sum(enumbers))
```

```
Out[9]: (<pyheal.ciphertext_op.CiphertextOp at 0x10eceed00>, -133.0)
```

Mix list operations

In [34]:
```
penumbers = np.random.choice(numbers+enumbers, size=10, replace=False)
penumbers
```

Out[34]:
```
array([<pyheal.ciphertext_op.CiphertextOp object at 0x110639a98>,
       <pyheal.ciphertext_op.CiphertextOp object at 0x1106399e8>,
       <pyheal.ciphertext_op.CiphertextOp object at 0x110639990>,
       <pyheal.ciphertext_op.CiphertextOp object at 0x110639a40>, -28, 56,
       <pyheal.ciphertext_op.CiphertextOp object at 0x110639ca8>, -76, 82,
       <pyheal.ciphertext_op.CiphertextOp object at 0x110639ba0>],
      dtype=object)
```

In [11]:
```
sum(penumbers), decryptor_decoder.decode(sum(penumbers))
```

Out[11]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x110639518>, -375.0)
```

Building equations

In [12]:
```
def formula(a, b, c, d, e, f):
    return a*b**3+c*d**2+e*f
```

In [13]:
```
formula(1, 2, 3, 4, 5, 6)
```

Out[13]:
```
86
```

In [14]:
```
r = formula(1, encryptor_encoder.encode(2), 3, 4, 5, 6)
r, decryptor_decoder.decode(r)
```

Out[14]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x110639fc0>, 86.0)
```

In [15]:
```
def formula2(a, b, c, d, e, f, g):
    return formula(a, b, c, d, e, f)/g
```

In [16]:
```
r = formula2(1, encryptor_encoder.encode(2), 3, 4, 5, 6, 7)
r, decryptor_decoder.decode(r)
```

Out[16]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x1106396d0>, 12.285714274272323)
```

Using pre-build equations (numpy, mse)

In [17]:
```
np.mean(enumbers), decryptor_decoder.decode(np.mean(enumbers))
```

Out[17]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x1106395c8>, -13.299999981420115)
```

In [18]:
```
def mse(a, b):
    return np.square(np.subtract(a, b)).mean()
```

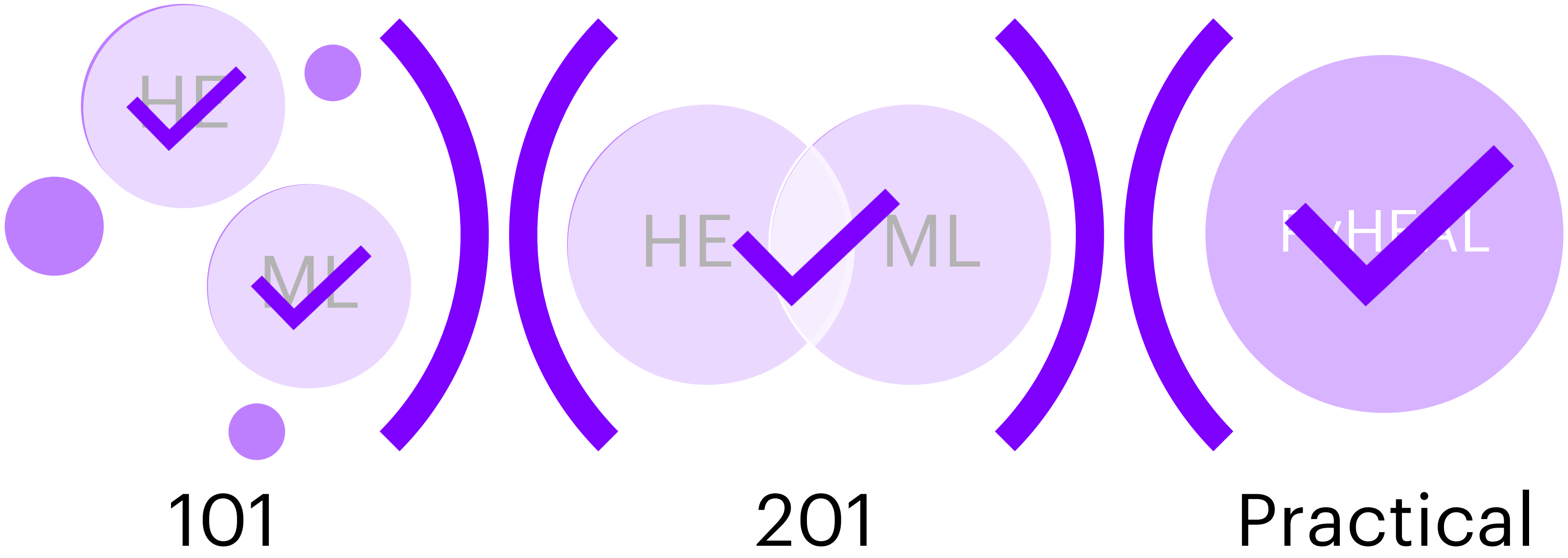
In [19]:
```
a = [1,2,3,4,5,6]
b = [6,5,4,3,2,1]
mse(a,b)
```

Out[19]:
```
11.666666666666666
```

In [20]:
```
ea = encryptor_encoder.encode(a)
eb = encryptor_encoder.encode(b)
r = mse(ea, eb)
r, decryptor_decoder.decode(r)
```

Out[20]:
```
(<pyheal.ciphertext_op.CiphertextOp at 0x1113cf468>, 11.666666655801237)
```

WHAT'S NEXT?



THANK YOU

ANY QUESTIONS?

accenture