moz://a

Building AI Units in Rust

D. Vigneshwer

About Me.

I am Vigneshwer

I work on AI & Rust

Author of Rust CookBook

Founder of DeepRust

Overview

Demystifying Artificial Intelligence

- Fundamental units
- Maths behind AI

Rust is great for building AI

- Properties of Rust
- Different AI crates
- Code snippets of Mathematical model

Template for mathematical crates

- Project structure

moz://a

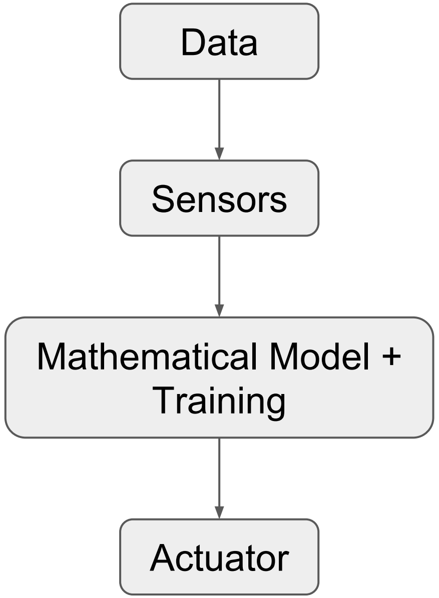
What is Artificial Intelligence (AI)?

AI is defined as the science of making computers behave like humans

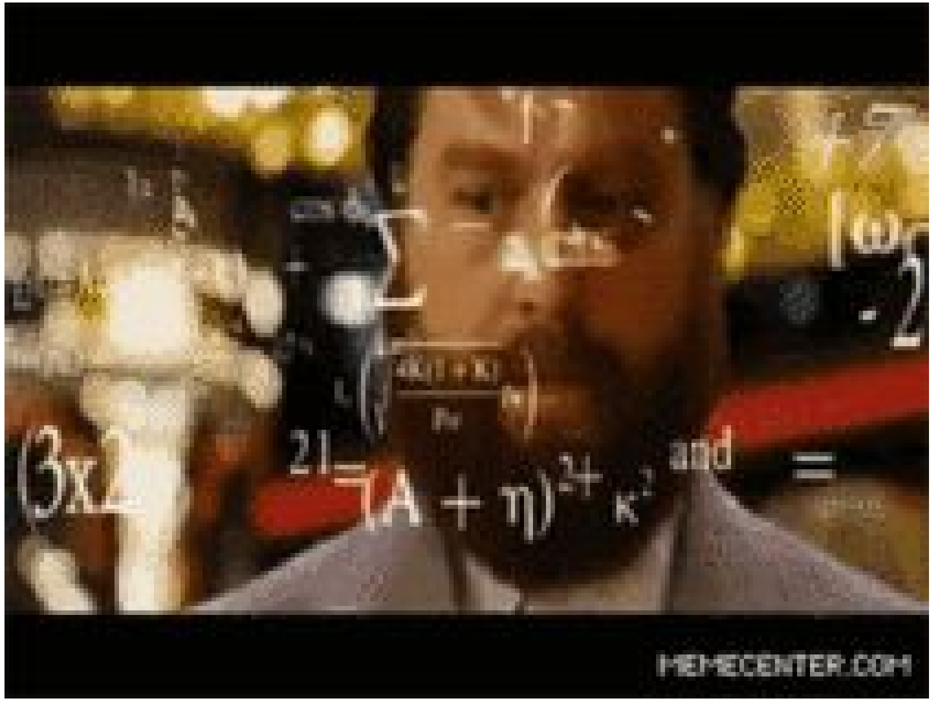
Voice Search

Self driving cars

Fundamental units of AI system



Basic AI System



moz://a

Maths Behind AI

Without mathematics, there's nothing you can do. Everything around you is mathematics. Everything around you is numbers.

- Shakuntala Devi

Let’s Understand

y = f(x)

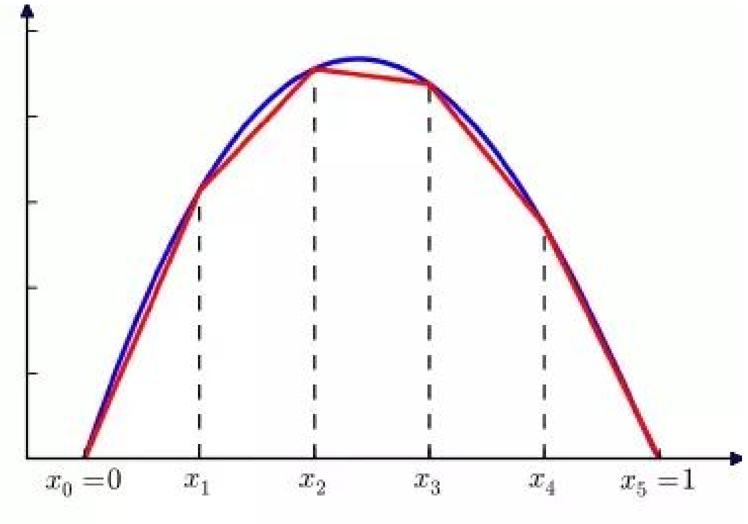
- Function to learn the curve (Activation fn)

y = w1*a + w2*b + w3*c +....

- Inputs (a, b, c)
- Weights (w1, w2, w3 ...)

y = w1*a + w2*b + w3*c +.... + b

- Bais (b)



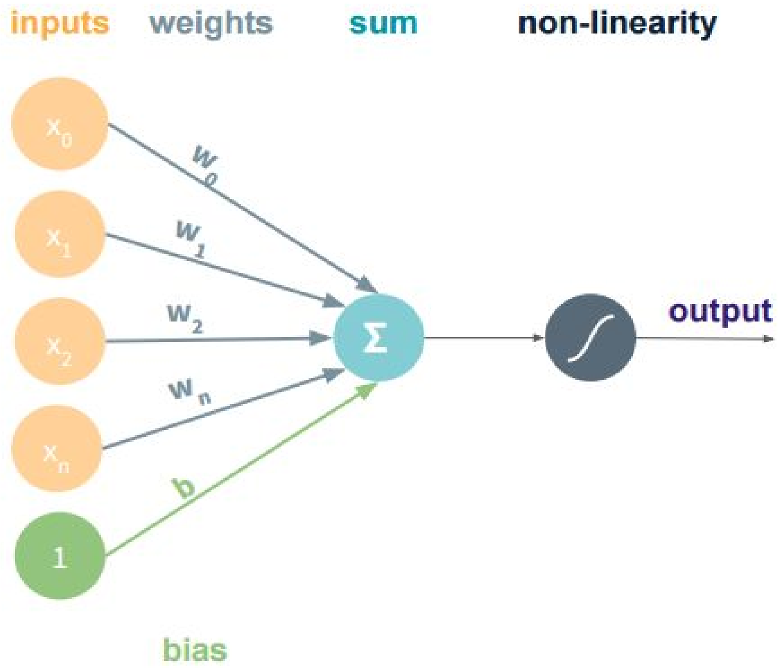
Artificial Neural Network

Activation Function

$$output = \boxed{g}(XW + b)$$

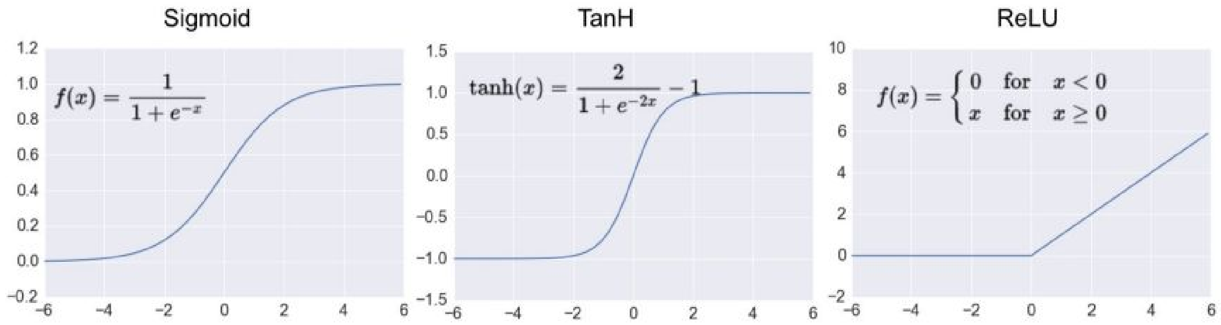
$$X = x_0, x_1, \ldots x_n$$

$$W = w_0, w_1, \ldots w_n$$

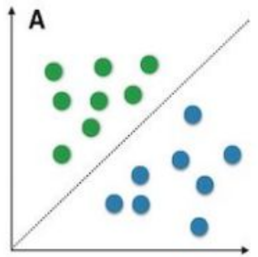


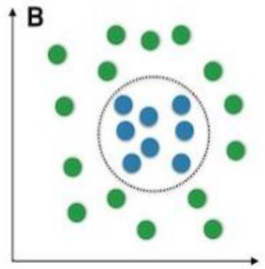
Activation function

Activation functions add nonlinearity to our network's function



Most real-world problems + data are non-linear



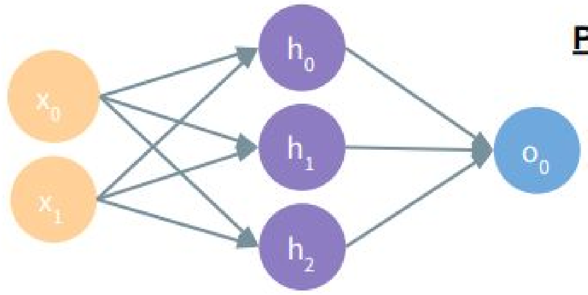


moz://a

Example: Placing an order in stock market

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	**Actual**
0.05	1
0.02	0
0.96	1
0.35	1

$$\text{total loss} := J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}))$$

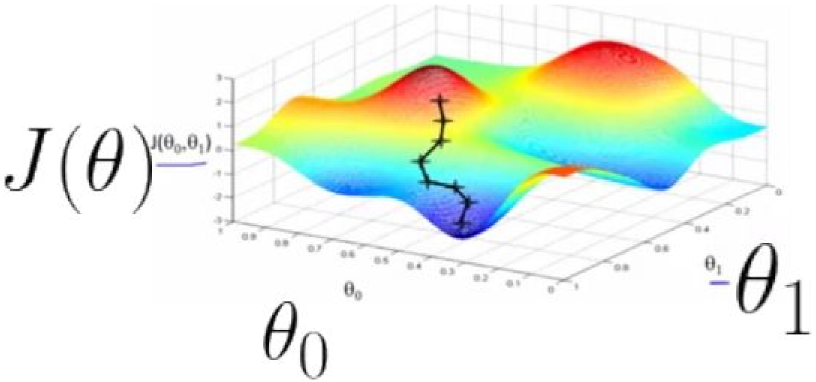
Training Neural networks

Loss is a function of the model's parameters

- Aim is to minimize loss and find the best parameters

$$\underbrace{arg_{\theta}\ \min \frac{1}{N}\sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})}_{J(\theta)}$$

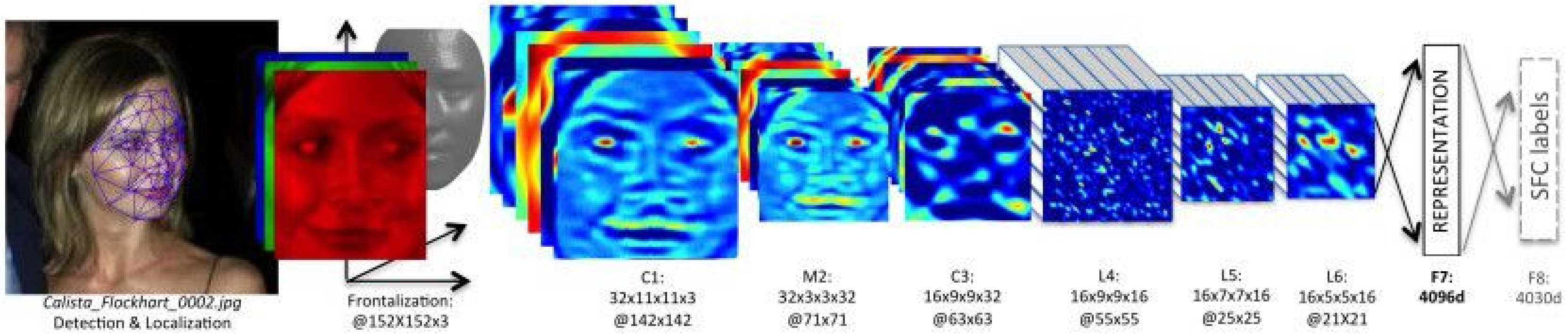
$$J(\theta) \leftarrow \theta = W_1, W_2 \ldots W_n$$



Deep Learning (Mathematical Model)

Convolutional neural networks(CNN)

- Feature engineering



moz://a

Evaluation metrics

Accuracy: Overall, how often is the classifier correct?

(TP+TN)/total = (100+50)/165 = 0.91

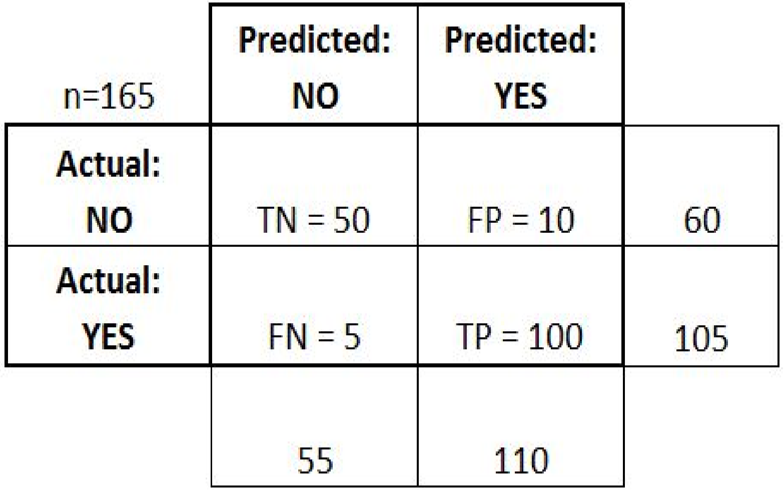
True Positive Rate (Recall) : When it's actually yes

how often does it predict yes?

TP/actual yes = 100/105 = 0.95

Precision: When it predicts yes, how often is it cor

TP/predicted yes = 100/110 = 0.91



n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

moz://a

Rust is great for building AI

Needs:

- AI has a lot of maths and is iterative in nature
- Great scope for parallel programming

Features:

- Threads without data races
- Advanced parallel programming
- Minimal runtime
- Guaranteed memory safety
- Zero-cost abstraction

Structs

Structs

```
struct Point {
    x: i32,
    y: i32,
}


fn main() {
    let origin = Point { x: 0, y: 0 }; // origin: Point

    println!("The origin is at ({}, {})", origin.x, origin.y);
}
```

Method calls

```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

impl Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}

fn main() {
    let c = Circle { x: 0.0, y: 0.0, radius: 2.0 };
    println!("{}", c.area());
}
```

Traits

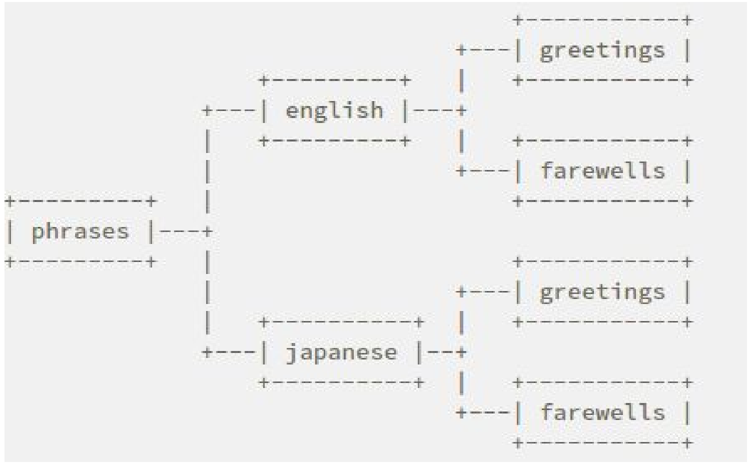
```
struct Circle {
    x: f64,
    y: f64,
    radius: f64,
}

trait HasArea {
    fn area(&self) -> f64;
}

impl HasArea for Circle {
    fn area(&self) -> f64 {
        std::f64::consts::PI * (self.radius * self.radius)
    }
}
```

moz://a

Crates and Module

- cargo new phrases
- src/lib.rs

  -
    ```
    mod english {
        mod greetings {
        }

        mod farewells {
        }
    }

    mod japanese {
        mod greetings {
        }

        mod farewells {
        }
    }
    ```



```
                                                      +-----------+
                                                  +---| greetings |
                                +---------+       |   +-----------+
                            +---| english |---+
                            |   +---------+   |   +-----------+
                            |                 +---| farewells |
+---------+                 |                     +-----------+
| phrases |---+
+---------+                 |                     +-----------+
                            |                 +---| greetings |
                            |   +----------+  |   +-----------+
                            +---| japanese |--+
                                +----------+  |   +-----------+
                                              +---| farewells |
                                                  +-----------+
```

moz://a

Rust Units

```
$ tree .
.
├── Cargo.lock
├── Cargo.toml
├── src
│   ├── english
│   │   ├── farewells.rs
│   │   ├── greetings.rs
│   │   └── mod.rs
│   ├── japanese
│   │   ├── farewells.rs
│   │   ├── greetings.rs
│   │   └── mod.rs
│   └── lib.rs
└── target
    └── debug
        ├── build
        ├── deps
        ├── examples
        ├── libphrases-a7448e02a0468eaa.rlib
        └── native
```

Template for AI crates

```
extern crate phrases;

fn main() {
    println!("Hello in English: {}", phrases::english::greetings::hello());
    println!("Goodbye in English: {}", phrases::english::farewells::goodbye());

    println!("Hello in Japanese: {}", phrases::japanese::greetings::hello());
    println!("Goodbye in Japanese: {}", phrases::japanese::farewells::goodbye());
}
```

Rayon

Features:

- Data parallelism library in Rust
- Easily converts a sequential computation into a parallel one
- Guarantees data-race freedom

Code Snippet:

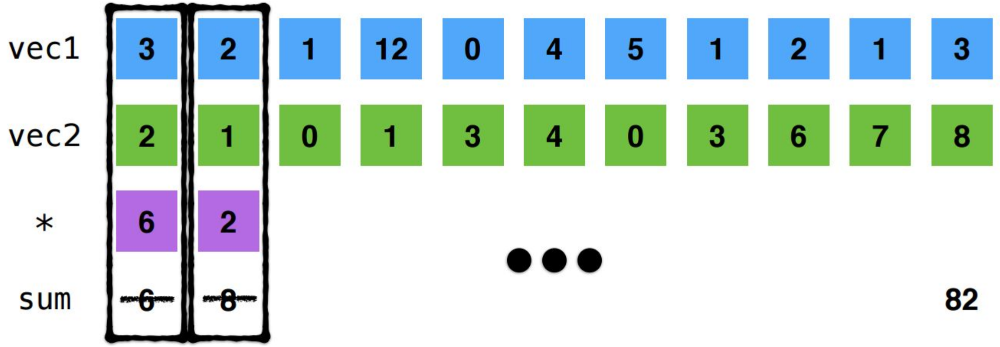
```
// import the traits
use rayon::prelude::*;

// compute the sum of the squares of a sequence of integers
fn sum_of_squares(input: &[i32]) -> i32 {
    input.par_iter()
         .map(|&i| i * i)
         .sum()
}
```

Reading an Image

```
fn load_images(paths: &[PathBuf]) -> Vec<Image> {
    paths.par_iter()
    .map(|path| Image::load(path))
    .collect() // returns a vector
}
```

Dot Product

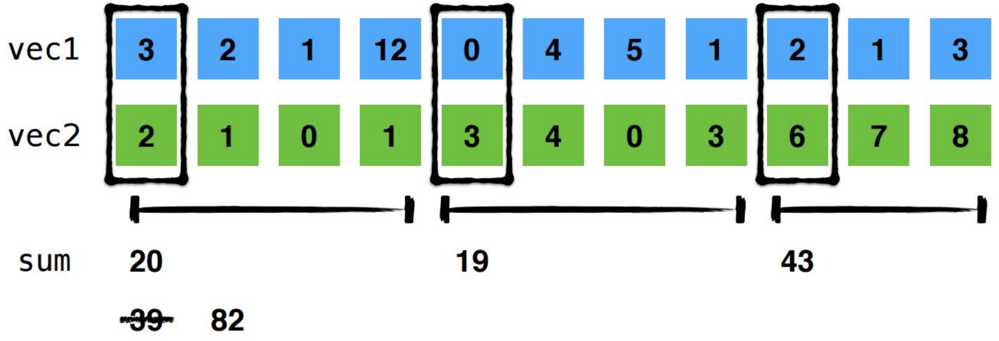
```
fn dot_product(vec1: &[i32], vec2: &[i32]) -> i32 {
        vec1.iter()
        .zip(vec2)
        .map(|(e1, e2)| e1 * e2)
        .fold(0, |a, b| a + b)        // aka .sum()
}
```



moz://a

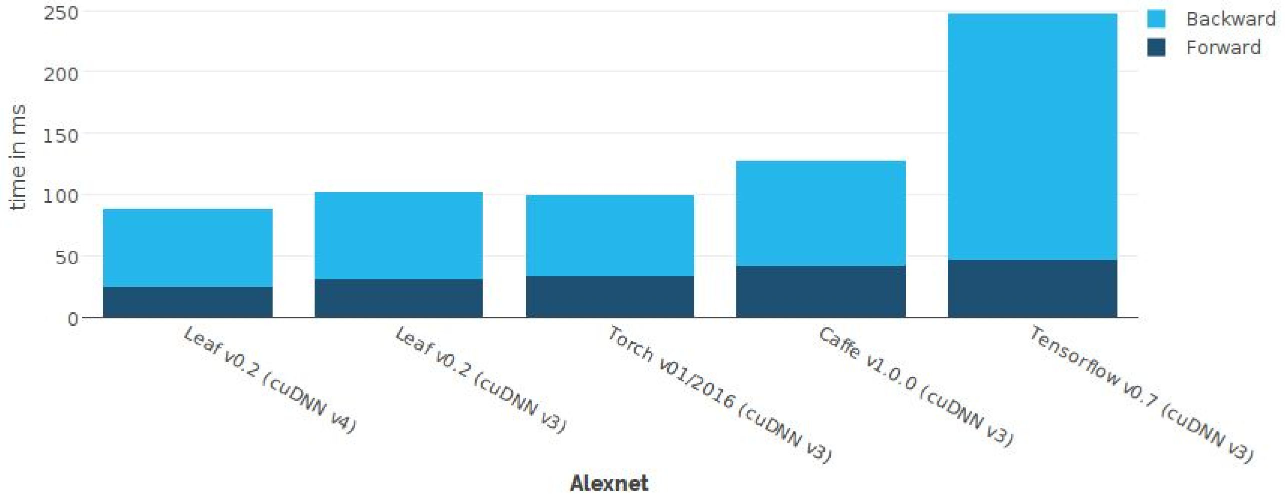
Dot product

```
fn dot_product(vec1: &[i32], vec2: &[i32]) -> i32 {
          vec1.par_iter()
           .zip(vec2)
          .map(|(e1, e2)| e1 * e2)
           .reduce(|| 0, |a, b| a + b) // aka .sum()
}
```



Different AI crates

- Leaf
- Rustlearn
- DeepRust



moz://a

Live Demos

Thanks!

Twitter @dvigneshwer

Instagram @dvigneshwer

Website dvigneshwer.github.io/

Q&A