Never

Functional Programming Language

Sławomir Maludziński, Ph.D.
Jakub Podgórski
## Agenda

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Design Decisions</td>
</tr>
<tr>
<td>Demo</td>
<td>Neural Network</td>
</tr>
<tr>
<td></td>
<td>Supervised Learning</td>
</tr>
</tbody>
</table>
About Never

- Hobby project
- Creativity escape
- I wanted to learn more about programming languages
- Scripting languages are good for prototyping
- I chose matrices as they are frequently used in science and technology
Design Decisions

- Functional Programming Language
- Syntactically Scoped
- C-Like Syntax
- Reference Model
- Statically Typed
- Boxed Values
- Call By Value

Model
Under the Hood

Exception Handler Binary Search
Linked Lists, Trees, Hash Maps
Written in C with Bison/Flex
func calc() -> (float) -> float {
    func fah2cel(float f) -> float { (f - 32.0) / 1.8 }
}

func main() -> float {
    calc()(212.0)
}

- First class functions
- Everything is an expression
- Operators + - * / ?:
- Float
Tock - Types - Integer

```go
func gcd(x -> int, y -> int) -> int {
    (y == 0) ? x : gcd(y, x % y)
}

func main() -> int {
    gcd(56, 12)
}
```

- Integer
- Operator `%`
- Operators and or not
func deg2rad(deg -> float) -> float {
    deg * 3.14159 / 180
}

func get_func() -> (float) -> float {
    cos
}

func main() -> float {
    get_func()(deg2rad(60.0))
}

- Built-in functions:
  - sin, cos, tan, exp, log, sqrt, pow
  - print, printf, assert
First class matrices
Conformat arrays
Overloaded operators + - *
Tick - Control Flow

```javascript
var i = 0; var j = 0;

for ({ i = 0; j = 0 }; i < 10; 
    { i = i + 1; j = j + 2 }) {
    print(i + j)
}
```

- Bindings
- Control flow
- Side effects
- Operator =
func main() -> int {
    var s1 = "text equal";
    var s2 = "text equal";

    assert(if (s1 == s2) { 1 } else { 0 } == 1)
}

- Strings
- Operator == !=
func one(d -> int) -> int {
    var f = let func (d -> int) -> int {
        12 / d
    } catch (division_by_zero) {
        12
    };
    f(0)
}

- Exceptions
Neural Network in Never

\[ s(\sum x_i \cdot w_i) \]
Sigmoid

```go
func sigmoid(x -> float) -> float {
    1.0 / (1.0 + exp(-x))
}
```

- Function $\frac{1}{1+e^{-x}}$
- Statically typed
- Float type
- Returns expression
Linear Congruential Generator

```go
func randomize(seed -> int) -> () -> int {
    var v = seed;
    func rand() -> int {
        v = (v * 1103515245 + 12345) % 2147483648
    }
    rand
}
```

- First class functions - `rand`
- Syntax scope - nested to any level
- `v` - closed within `randomize`
Matrix Algebra

```
func Hadamard_matrix(W1[D1, D2] -> float, 
                        W2[D3, D4] -> float) -> [_,_] -> float
{
    var r = 0; var c = 0;
    var H = {[ D1, D2 ]} -> float;

    for (r = 0; r < D1; r = r + 1) {
        for (c = 0; c < D2; c = c + 1) {
            H[r, c] = W1[r, c] * W2[r, c]
        }
    }

    H
}
```

- Hadamard multiplication
- Conformant matrices (arrays)
Forward Propagation - Input & Output

```
var x = [ [0, 1, 0],
         [1, 0, 0],
         [1, 1, 1],
         [0, 0, 1] ] -> float;

var y = [ [1, 0, 1, 0] ] -> float;
var yT = T_matrix(y);
```

- Input matrix \( x \)
- Output transposed matrix \( y = [ [1], [0], [1], [0] ] \)
- Middle value as output
Forward Propagation - Initialization

```
var W = {[ 3, 1 ]} -> float;
var rand = randomize(165);
rand_matrix(W, rand);
```

- $W$ initialized to random values
Forward Propagation - Output Calculation

```javascript
var s = [[4, 1]] -> float;
s = sigmoid_matrix(x * W);
```

- Operators `+`, `-`, `*` are overloaded for matrices.
- Output calculated to all inputs.
- Sigmoid function used to get 0/1 results.
Backpropagation - Error Calculation

```
var err = {[ 4, 1 ]} -> float;
err = yT - s;
```

- Calculate error for each input sample
- `err` used to correct weights
Backpropagation - Gradient Descent

```plaintext
var sD = {[ 4, 1 ]} -> float;
var one = {[ 4, 1 ]} -> float;

sD = Hadamard_matrix(s, one - s);
W = W + xT * Hadamard_matrix(err, sD)
```

- \( sD = s \times (1 - s) \)
- \( W = W + xT \times \text{err} \times sD \)

- First derivative (or gradient) multiplied by error lets to move towards function minimum
- Calculations for all input values \( \times \) at once
Forward and Backpropagation Learning Cycles

```go
func main() -> int {
    var i = 0;
    for (i = 0; i < 1000; i = i + 1) {
        ...
    }
}
```

- Forward and backpropagation need to be repeated until error is low
- Function `main` as program entry point
## Results

<table>
<thead>
<tr>
<th>X</th>
<th>$s(X \times W)$</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1, 1, 1]</td>
<td>0.96</td>
<td>1.00</td>
</tr>
<tr>
<td>[1, 1, 0]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>[1, 0, 1]</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>[1, 0, 0]</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>[0, 1, 1]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>[0, 1, 0]</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>[0, 0, 1]</td>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>[0, 0, 0]</td>
<td>0.50</td>
<td>0.00</td>
</tr>
</tbody>
</table>

- 7 correct results, 1 undecided
# Future

<table>
<thead>
<tr>
<th>Types</th>
<th>Lists</th>
<th>Externals</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>join</td>
<td>I/O</td>
<td>libraries</td>
</tr>
<tr>
<td>polynomial</td>
<td>slice</td>
<td>modules</td>
<td>VM optimizations</td>
</tr>
<tr>
<td>functions</td>
<td>comprehension</td>
<td>foreign function interface</td>
<td></td>
</tr>
<tr>
<td>type inference</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Creating programming languages...
  - is fun 😊
  - can teach you a lot 😞
  - is satisfactory 😊
Thank You!