Neural Networks

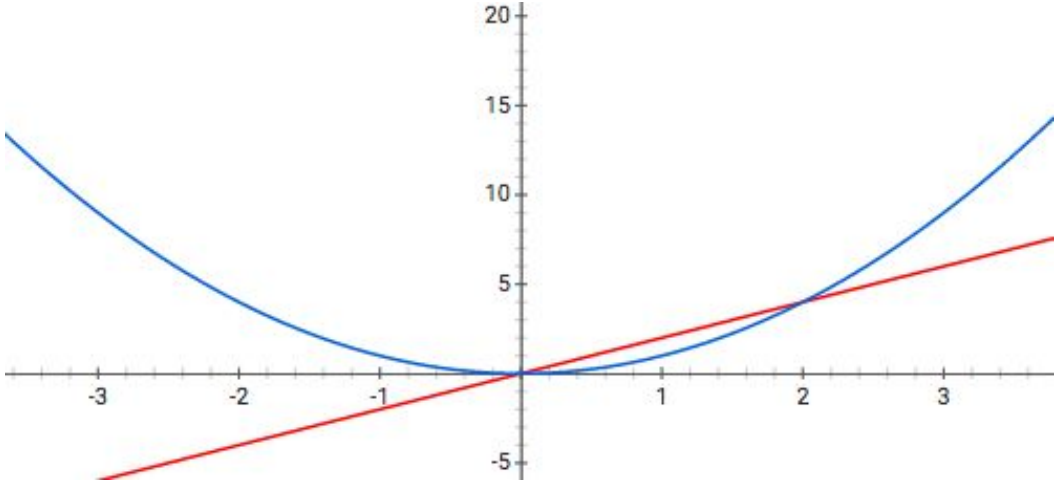
Key Concepts

- Modeling equations as a differentiable graph
- Learning as an optimization problem

Derivatives

$f(x) = x^2$
$f(-2) = 4$
$f(0) = 0$
$f(2) = 4$

$f'(x) = 2x$
$f'(-1) = -2$
$f'(0) = 0$
$f'(1) = 2$
$f'(3) = 6$

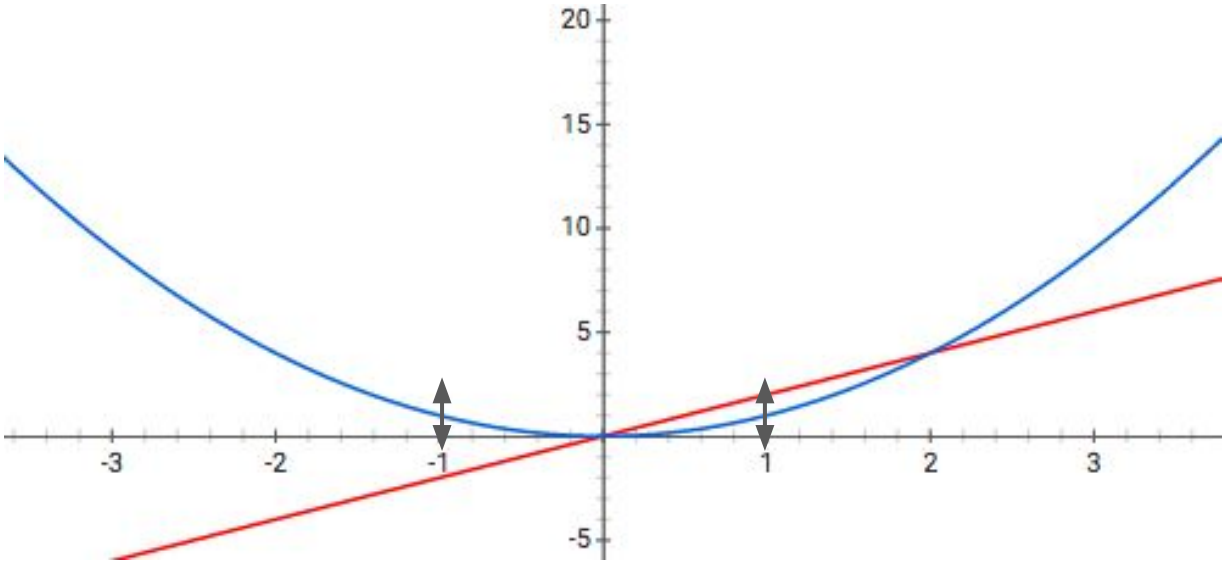


Why Derivatives?

$f(x) = x^2$
$f'(x) = 2x$

$a=-1, b=1$
$f(a) = 1$
$f(b) = 1$
$f'(a) = -1$
$f'(b) = 1$

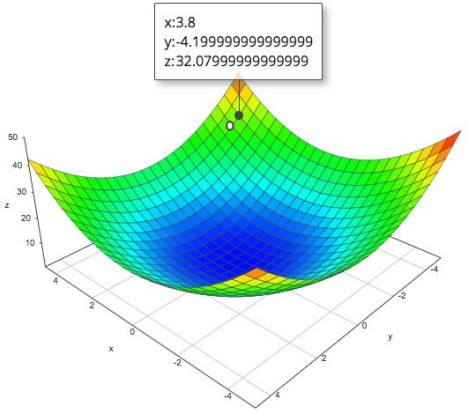
$f(a+f'(a)) = 4$
$f(b+f'(b)) = 4$
$f(a-f'(a)) = 0$
$f(b-f'(b)) = 0$

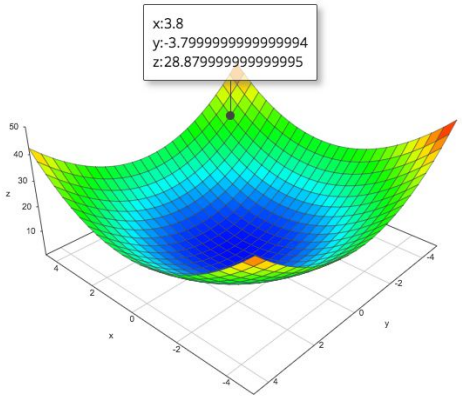


Partial Derivatives

$g(a,b) = a^2+ab+b^2$

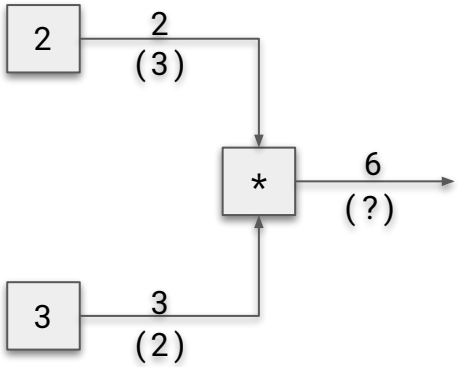
$g_a(a,b) = 2a+b$
$g_b(a,b) = a+2b$





$f(a,b) = ab$
$f_a(a,b) = b$
$f_b(a,b) = a$

$f(3,2) = 6$
$f_a(3,2) = 2$
$f_b(3,2) = 3$



Chain Rule

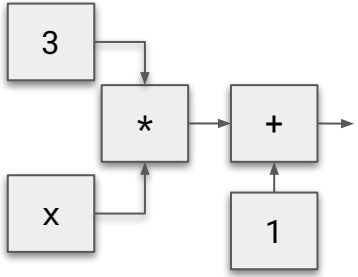
$h(x) = f(g(x))$
$h'(x) = f'(g(x))g'(x)$

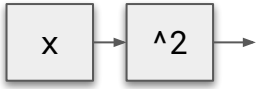
$f(x) = 3x+1$
$f'(x) = 3$

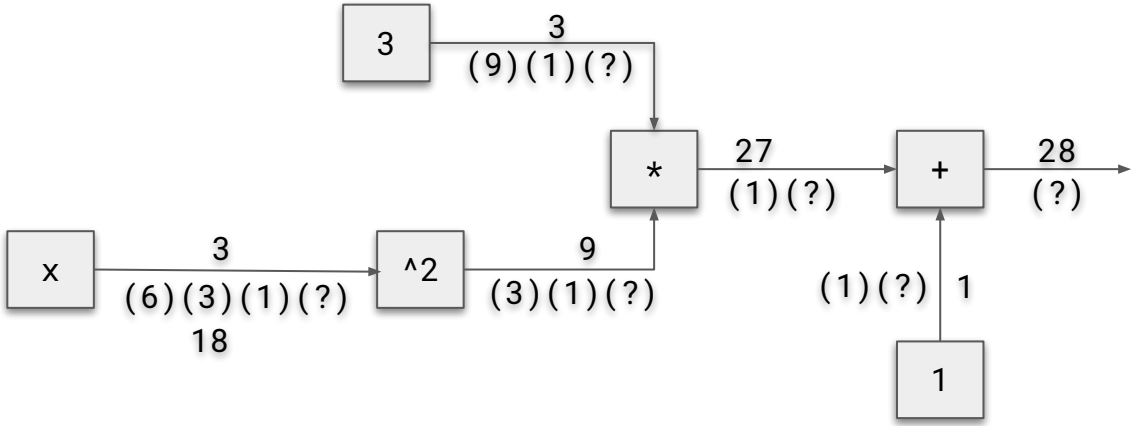
$g(x) = x^2$
$g'(x) = 2x$

$h(x) = 3x^2+1$
$h'(x) = (3)(2x)$
$h'(x) = 6x$

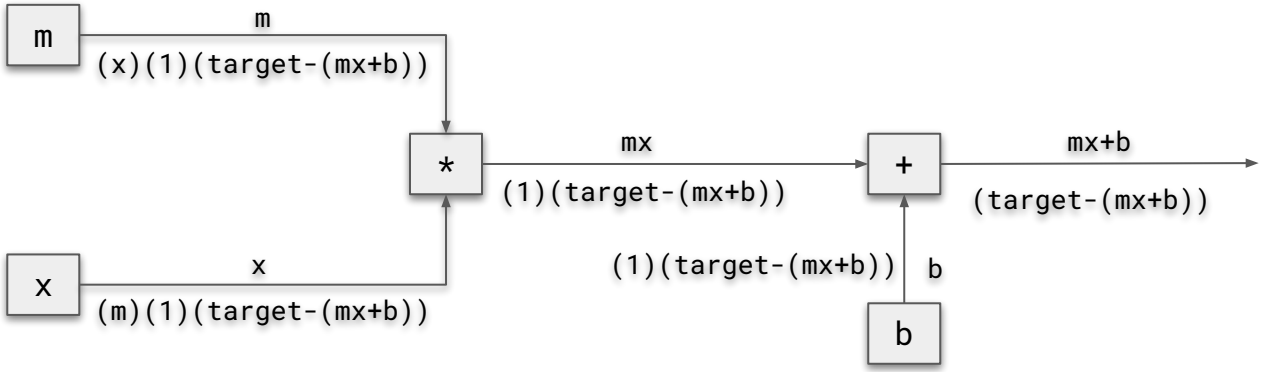
$h(3) = 28$
$h'(3) = 18$







Fitting



m		x		b		mx		mx+b		target	target - mx+b	learning rate
3.00	-10.00	2.00	-15.00	2.00	-5.00	6.00	-5.00	8.00	-5.00	3.00	-5.00	0.10
2.00	-5.00	2.00	-5.00	1.50	-2.50	4.00	-2.50	5.50	-2.50	3.00	-2.50	0.10
1.50	-2.50	2.00	-1.88	1.25	-1.25	3.00	-1.25	4.25	-1.25	3.00	-1.25	0.10
1.25	-1.25	2.00	-0.78	1.13	-0.63	2.50	-0.63	3.63	-0.63	3.00	-0.63	0.10
1.13	-0.63	2.00	-0.35	1.06	-0.31	2.25	-0.31	3.31	-0.31	3.00	-0.31	0.10
1.06	-0.31	2.00	-0.17	1.03	-0.16	2.13	-0.16	3.16	-0.16	3.00	-0.16	0.10
1.03	-0.16	2.00	-0.08	1.02	-0.08	2.06	-0.08	3.08	-0.08	3.00	-0.08	0.10
1.02	-0.08	2.00	-0.04	1.01	-0.04	2.03	-0.04	3.04	-0.04	3.00	-0.04	0.10

Updates

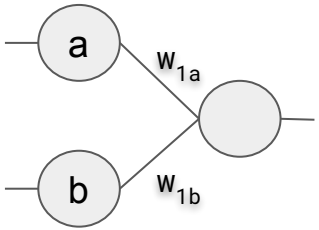
```
l = 0.1
m = m+(l*m')
```

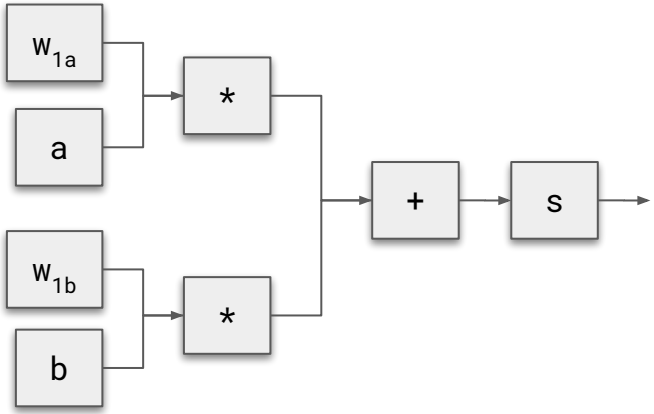
As Code

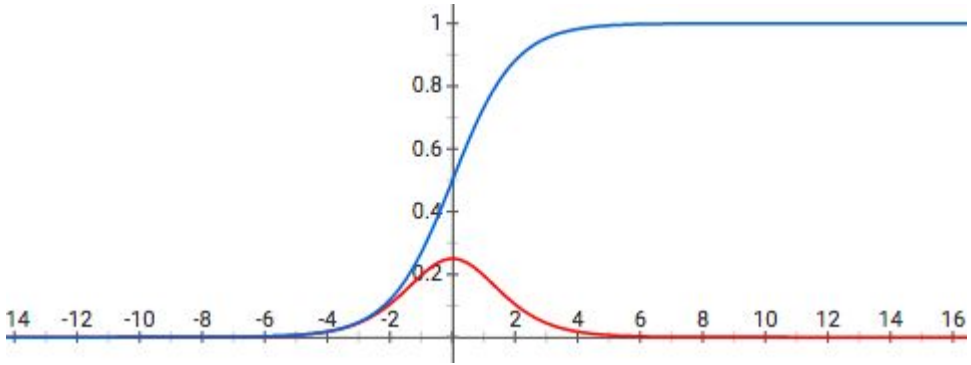
```
let x = new Constant({id: "x"});
let m = new Variable({id: "m"});
let b = new Variable({id: "b"});
let mul = new Multiply([m, x]);
let add = new Add([mul, b]);

let solver = new Solver([add], {
  "m": 2.0,
  "b": 3.0,
  "x": 4.0,
});

let target = 5;
for(let i = 0; i < 15; i++) {
  solver.solve();
  let result = solver.idToState[add.id].value;
  let errors = {};
  errors[add.id] = target - result;
  solver.fit(errors, 0.1);
}
```

```
class Multiply extends Node {
...
 forward(vals) {
   // x * y
   return vals[0] * vals[1];
 }
 backward(vals) {
   // [x, y] -> [y, x]
   return [vals[1], vals[0]];
 }
...
}
```

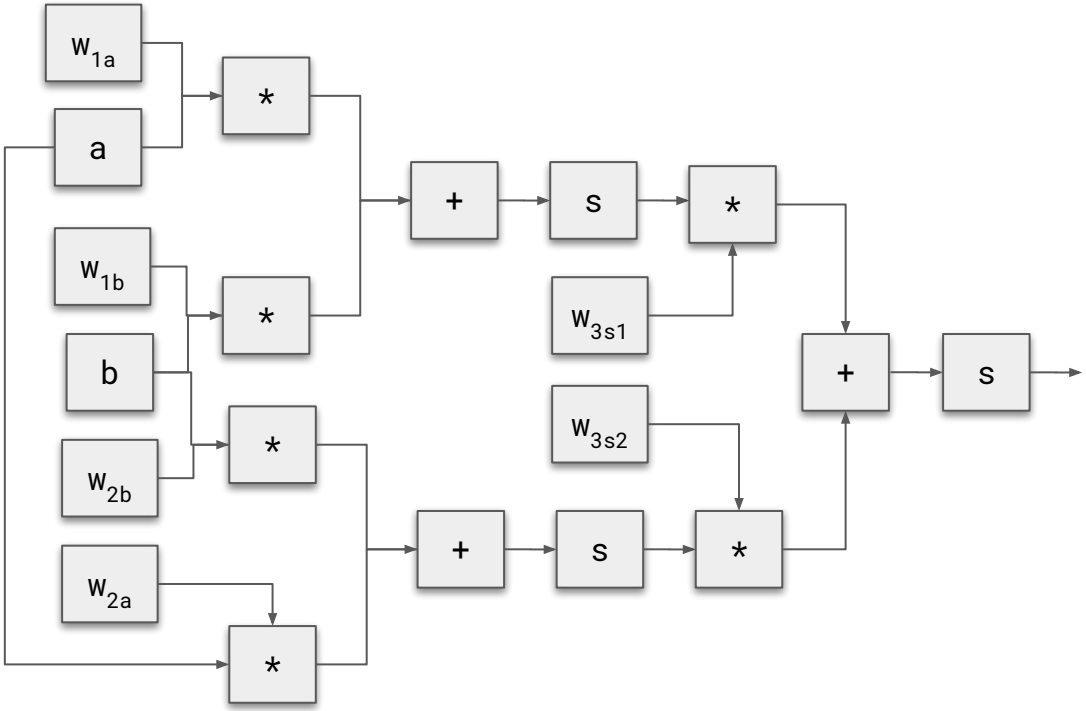
Single Neuron (2-1)

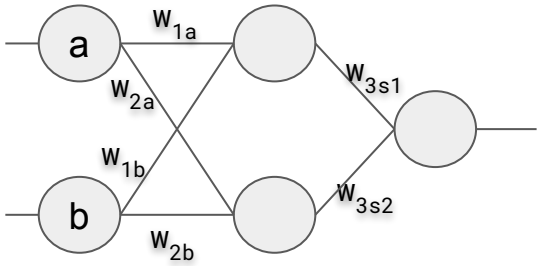






Deep Network (2-2-1)





As Code

```
let i1 = new Constant({id: "i1"});
let i2 = new Constant({id: "i2"});

let i1h1 = new Variable({id: "i1->h1"});
let i2h1 = new Variable({id: "i2->h1"});
let i1h2 = new Variable({id: "i1->h2"});
let i2h2 = new Variable({id: "i2->h2"});
let h2o1 = new Variable({id: "h2->o1"});
let h1o1 = new Variable({id: "h1->o1"});

let h1i = new Add([new Multiply([i1, i1h1]), new Multiply([i2, i2h1])]);
let h1o = new Sigmoid([h1i]);

let h2i = new Add([new Multiply([i1, i1h2]), new Multiply([i2, i2h2])]);
let h2o = new Sigmoid([h2i]);

let o1i = new Add([new Multiply([h1o, h1o1]), new Multiply([h2o, h2o1])]);
let o1o = new Sigmoid([o1i]);

let data = [
  [0, 0, 0],
  [1, 0, 1],
  [0, 1, 1],
  [1, 1, 0],
];
```

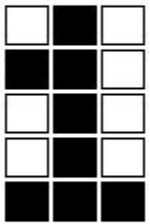
```
let solver = new Solver([o1o], {
  "i1->h1": randomInit(),
  "i2->h1": randomInit(),
  "i1->h2": randomInit(),
  "i2->h2": randomInit(),
  "h2->o1": randomInit(),
  "h1->o1": randomInit(),
});

let learningRate = 0.01;
let iterations = 10000;
for(let i = 0; i < iterations; i++) {
  let indexes = shuffleIndexes(data.length);
  for(let j = 0; j < data.length; j++) {
    let randomIndex = indexes[j];
    solver.solve({"i1": data[randomIndex][0], "i2": data[randomIndex][1]});
    let target = data[randomIndex][2];
    let result = solver.idToState[o1o.id].value;
    let errors = {};
    errors[o1o.id] = target - result;
    solver.fit(errors, learningRate);
  }
 }
}
```

Demo

Train

1K 10K 100K Error: 0.12393578348936807

Test Input

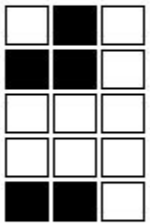


Test Output

Train

1K 10K 100K Error: 0.12393578348936807

Test Input



Test Output

AlphaGo

- Tree search guided by an evaluation function
- Evaluation function estimates likelihood of winning
- Evaluation function is a neural network

Language Translation (Seq2Seq)

- Two networks; an encoder and decoder
- Given a sentence, the encoder outputs a matrix
- Given a matrix, the decoder outputs a sentence
- Training happens with encoder and decoder combined
- Allows for swapping out encoders and decoders