
micrograd



A tiny Autograd engine (with a bite! :)). Implements backpropagation (reverse-mode autodiff) over a dynamically built DAG and a small neural networks library on top of it with a PyTorch-like API. Both are tiny, with about 100 and 50 lines of code respectively. The DAG only operates over scalar values, so e.g. we chop up each neuron into all of its individual tiny adds and multiplies. However, this is enough to build up entire deep neural nets doing binary classification, as the demo notebook shows. Potentially useful for educational purposes.

Installation

```
1 pip install micrograd
```

Example usage

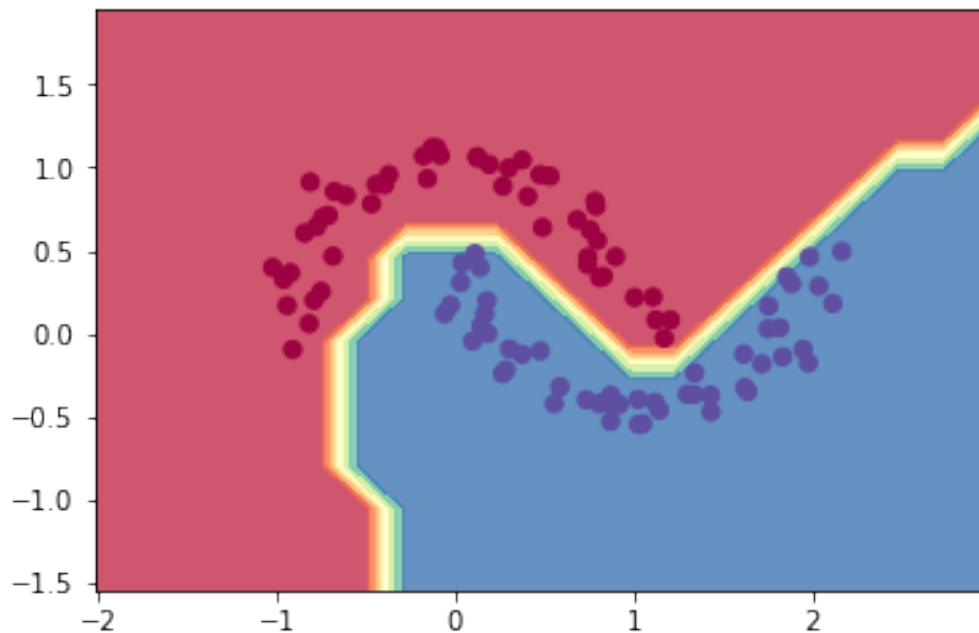
Below is a slightly contrived example showing a number of possible supported operations:

```
1 from micrograd.engine import Value
2
```

```
3 a = Value(-4.0)
4 b = Value(2.0)
5 c = a + b
6 d = a * b + b**3
7 c += c + 1
8 c += 1 + c + (-a)
9 d += d * 2 + (b + a).relu()
10 d += 3 * d + (b - a).relu()
11 e = c - d
12 f = e**2
13 g = f / 2.0
14 g += 10.0 / f
15 print(f'{g.data:.4f}') # prints 24.7041, the outcome of this forward
    pass
16 g.backward()
17 print(f'{a.grad:.4f}') # prints 138.8338, i.e. the numerical value of
    dg/da
18 print(f'{b.grad:.4f}') # prints 645.5773, i.e. the numerical value of
    dg/db
```

Training a neural net

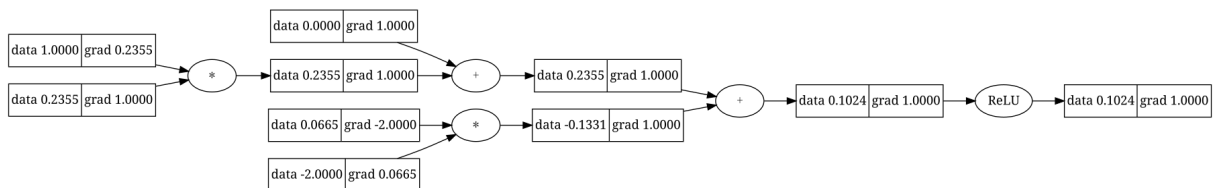
The notebook [demo.ipynb](#) provides a full demo of training an 2-layer neural network (MLP) binary classifier. This is achieved by initializing a neural net from `micrograd.nn` module, implementing a simple svm “max-margin” binary classification loss and using SGD for optimization. As shown in the notebook, using a 2-layer neural net with two 16-node hidden layers we achieve the following decision boundary on the moon dataset:



Tracing / visualization

For added convenience, the notebook [trace_graph.ipynb](#) produces graphviz visualizations. E.g. this one below is of a simple 2D neuron, arrived at by calling `draw_dot` on the code below, and it shows both the data (left number in each node) and the gradient (right number in each node).

```
1 from micrograd import nn
2 n = nn.Neuron(2)
3 x = [Value(1.0), Value(-2.0)]
4 y = n(x)
5 dot = draw_dot(y)
```



Running tests

To run the unit tests you will have to install PyTorch, which the tests use as a reference for verifying the correctness of the calculated gradients. Then simply:

```
1 python -m pytest
```

License

MIT