micrograd_2

January 3, 2026

1 `micrograd`: construction d'une bibliothèque de rétropropagation du gradient (partie 2)

Ce notebook est dérivé du travail d'Andrej Karpathy et reprend pas à pas les étapes exposées dans sa première séance de cours sur la construction d'un outil en Python pour le calcul du gradient et sa propagation arrière:

The spelled-out intro to neural networks and backpropagation: building micrograd

Voici les ressources originales associées à la vidéo youtube d'Andrej:

- micrograd on github: https://github.com/karpathy/micrograd
- notebook original: https://github.com/karpathy/nn-zero-to-hero/tree/master/lectures/micrograd
- exercices: https://colab.research.google.com/drive/1FPTx1RXtBfc4MaTkf7viZZD4U2F9gtKN?usp=sharing

[1]:
```
# Imports de la librairie standard Python
import math
# Imports spécifiques (doivent être présent dans l'environnement Python de ce notebook)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from graphviz import Digraph
```

1.1 Amélioration de classe `Value`

1.1.1 Rappels

[2]:
```
class Value:

    def __init__(self, data, children=(), op='', label=''):
        self.data = data
        self._prev = set(children)
        self._op = op
        self.label = label
        self.grad = 0.0
        self._backward = lambda: None
```

```
    def __repr__(self):
        return f"Value(data={self.data}, label={self.label}, grad={self.grad})"

    def __add__(self, other):
        out = self.__class__(self.data + other.data, children=(self, other),␣
↪op='+')
        def _backward():
            self.grad = out.grad
            other.grad = out.grad
        out._backward = _backward
        return out

    def __mul__(self, other):
        out = self.__class__(self.data * other.data, children=(self, other),␣
↪op='*')
        def _backward():
            self.grad = other.data * out.grad
            other.grad = self.data * out.grad
        out._backward = _backward
        return out

    def tanh(self):
        x = self.data
        t = (math.exp(2*x) - 1) / (math.exp(2*x) + 1)
        out = Value(t, children=(self,), op='tanh')
        def _backward():
            self.grad = (1 - t**2) * out.grad
        out._backward = _backward
        return out

    def backward(self):
        topo = []
        visited = set()
        def build_topo(v):
          if v not in visited:
            visited.add(v)
            for child in v._prev:
              build_topo(child)
            topo.append(v)
        build_topo(self)
        self.grad = 1.0
        for node in reversed(topo):
          node._backward()
```

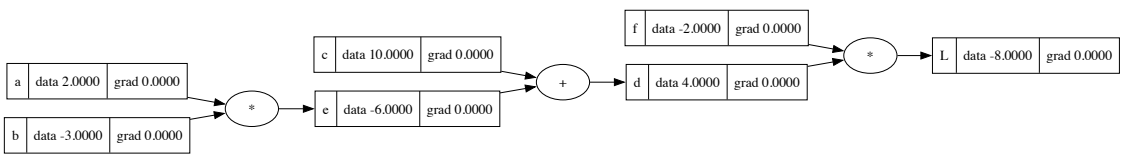
[3]:
```
a = Value(2.0, label='a')
b = Value(-3.0, label='b')
c = Value(10.0, label='c')
```

```
e = a * b
e.label = 'e'
d = e + c
d.label = 'd'
f = Value(-2.0, label='f')
L = d * f
L.label = 'L'
```

[4]:
```
def trace(root):
  # builds a set of all nodes and edges in a graph
  nodes, edges = set(), set()
  def build(v):
    if v not in nodes:
      nodes.add(v)
      for child in v._prev:
        edges.add((child, v))
        build(child)
  build(root)
  return nodes, edges

def draw_dot(root):
  dot = Digraph(format='svg', graph_attr={'rankdir': 'LR'}) # LR = left to right

  nodes, edges = trace(root)
  for n in nodes:
    uid = str(id(n))
    # for any value in the graph, create a rectangular ('record') node for it
    dot.node(name = uid, label = "{ %s | data %.4f | grad %.4f }" % (n.label, n.
 ↪data, n.grad), shape='record')
    if n._op:
      # if this value is a result of some operation, create an op node for it
      dot.node(name = uid + n._op, label = n._op)
      # and connect this node to it
      dot.edge(uid + n._op, uid)

  for n1, n2 in edges:
    # connect n1 to the op node of n2
    dot.edge(str(id(n1)), str(id(n2)) + n2._op)

  return dot
```

[5]:
```
draw_dot(L)
```

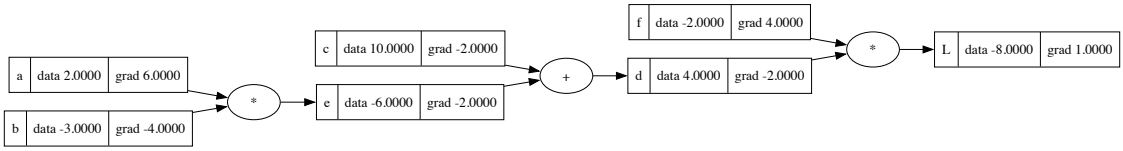
[5]:



[6]:
```
L.backward()
```

[7]:
```
draw_dot(L)
```

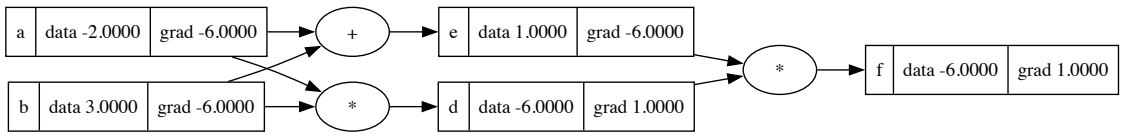
[7]:



1.1.2 Problème de non-cumul des gradients

[8]:
```
a = Value(-2.0, label='a')
b = Value(3.0, label='b')
d = a * b    ; d.label = 'd'
e = a + b    ; e.label = 'e'
f = d * e    ; f.label = 'f'

f.backward()

draw_dot(f)
```

[8]:



Dans ce graphe un peu particulier, les valeurs des gradients pour a et b sont fausses. En effet, il faut cumuler ces gradients lors du calcul, voir multivariable case (chain rule).

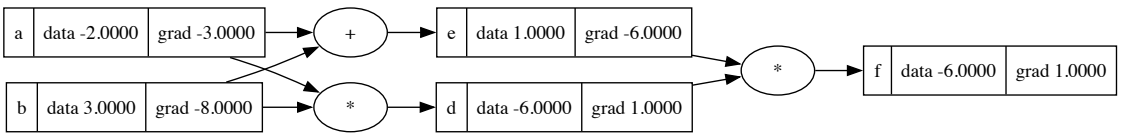
[9]:
```
class Value:

    def __init__(self, data, children=(), op='', label=''):
        self.data = data
        self._prev = set(children)
        self._op = op
        self.label = label
        self.grad = 0.0
        self._backward = lambda: None

    def __repr__(self):
        return f"Value(data={self.data}, label={self.label}, grad={self.grad})"
```

```
  def __add__(self, other):
      out = self.__class__(self.data + other.data, children=(self, other),␣
↪op='+')
      def _backward():
          self.grad += out.grad
          other.grad += out.grad
      out._backward = _backward
      return out

  def __mul__(self, other):
      out = self.__class__(self.data * other.data, children=(self, other),␣
↪op='*')
      def _backward():
          self.grad += other.data * out.grad
          other.grad += self.data * out.grad
      out._backward = _backward
      return out

  def tanh(self):
      x = self.data
      t = (math.exp(2*x) - 1) / (math.exp(2*x) + 1)
      out = Value(t, children=(self,), op='tanh')
      def _backward():
          self.grad += (1 - t**2) * out.grad
      out._backward = _backward
      return out

  def backward(self):
      topo = []
      visited = set()
      def build_topo(v):
        if v not in visited:
          visited.add(v)
          for child in v._prev:
            build_topo(child)
          topo.append(v)
      build_topo(self)
      self.grad = 1.0
      for node in reversed(topo):
        node._backward()
```

[10]:
```
a = Value(-2.0, label='a')
b = Value(3.0, label='b')
d = a * b    ; d.label = 'd'
e = a + b    ; e.label = 'e'
f = d * e    ; f.label = 'f'
```

```
f.backward()

draw_dot(f)
```

[10]:



1.1.3 Ajout de nouvelles fonctions

Support des constantes Pour `__add__` et `__mul__`, on ajoute une instruction permettant d'utiliser des constructions comme `b = a + 1`, en transformant `1` en `Value(1.0)`.

[11]:

```
class Value:

    def __init__(self, data, children=(), op='', label=''):
        self.data = data
        self._prev = set(children)
        self._op = op
        self.label = label
        self.grad = 0.0
        self._backward = lambda: None

    def __repr__(self):
        return f"Value(data={self.data}, label={self.label}, grad={self.grad})"

    def __add__(self, other):
        other = other if isinstance(other, Value) else Value(other)  # a + 1
        out = self.__class__(self.data + other.data, children=(self, other),␣
↪op='+')
        def _backward():
            self.grad += out.grad
            other.grad += out.grad
        out._backward = _backward
        return out

    def __mul__(self, other):
        other = other if isinstance(other, Value) else Value(other)  # a * 1
        out = self.__class__(self.data * other.data, children=(self, other),␣
↪op='*')
        def _backward():
            self.grad += other.data * out.grad
            other.grad += self.data * out.grad
        out._backward = _backward
```

```
        return out

    def tanh(self):
        x = self.data
        t = (math.exp(2*x) - 1) / (math.exp(2*x) + 1)
        out = Value(t, children=(self,), op='tanh')
        def _backward():
            self.grad += (1 - t**2) * out.grad
        out._backward = _backward
        return out

    def backward(self):
        topo = []
        visited = set()
        def build_topo(v):
          if v not in visited:
            visited.add(v)
            for child in v._prev:
              build_topo(child)
            topo.append(v)
        build_topo(self)
        self.grad = 1.0
        for node in reversed(topo):
          node._backward()
```

Commutativité Pour `__add__` et `__mul__`, support indifférencié de `a + 1` ou `1 + a`, en déclarant les méthodes `__radd__` et `__rmul__`.

[12]:
```
def __rmul__(self, other): # other * self
    return self * other
Value.__rmul__ = __rmul__
```

[13]:
```
def __radd__(self, other): # other * self
    return self + other
Value.__radd__ = __radd__
```

Implémentation de l'exponentielle, de la négation, de la puissance et de la division

[14]:
```
class Value:

    def __init__(self, data, children=(), op='', label=''):
        self.data = data
        self._prev = set(children)
        self._op = op
        self.label = label
        self.grad = 0.0
        self._backward = lambda: None
```

```
  def __repr__(self):
      return f"Value(data={self.data}, label={self.label}, grad={self.grad})"

  def __add__(self, other):
      other = other if isinstance(other, Value) else Value(other)  # a + 1
      out = self.__class__(self.data + other.data, children=(self, other),␣
↪op='+')
      def _backward():
          self.grad += out.grad
          other.grad += out.grad
      out._backward = _backward
      return out

  def __mul__(self, other):
      other = other if isinstance(other, Value) else Value(other)  # a * 1
      out = self.__class__(self.data * other.data, children=(self, other),␣
↪op='*')
      def _backward():
          self.grad += other.data * out.grad
          other.grad += self.data * out.grad
      out._backward = _backward
      return out

  def tanh(self):
      x = self.data
      t = (math.exp(2*x) - 1) / (math.exp(2*x) + 1)
      out = Value(t, children=(self,), op='tanh')
      def _backward():
          self.grad += (1 - t**2) * out.grad
      out._backward = _backward
      return out

  def __rmul__(self, other): # other * self
      return self * other

  def __radd__(self, other): # other * self
      return self + other

  def __pow__(self, other):
      assert isinstance(other, (int, float)), "only supporting int/float␣
↪powers for now"
      out = Value(self.data**other, (self,), f'**{other}')
      def _backward():
          self.grad += other * (self.data ** (other - 1)) * out.grad
      out._backward = _backward
      return out
```

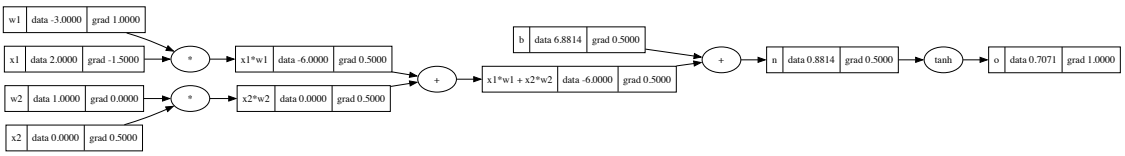
```
    def __truediv__(self, other): # self / other
        return self * other**-1

    def __neg__(self): # -self
        return self * -1

    def __sub__(self, other): # self - other
        return self + (-other)

    def exp(self):
        x = self.data
        out = Value(math.exp(x), (self, ), 'exp')
        def _backward():
          self.grad += out.data * out.grad
        out._backward = _backward
        return out

    def backward(self):
        topo = []
        visited = set()
        def build_topo(v):
          if v not in visited:
            visited.add(v)
            for child in v._prev:
              build_topo(child)
            topo.append(v)
        build_topo(self)
        self.grad = 1.0
        for node in reversed(topo):
          node._backward()
```

Exemples d'utilisation

[15]:
```
# inputs x1,x2
x1 = Value(2.0, label='x1')
x2 = Value(0.0, label='x2')
# weights w1,w2
w1 = Value(-3.0, label='w1')
w2 = Value(1.0, label='w2')
# bias of the neuron
b = Value(6.8813735870195432, label='b')
# x1*w1 + x2*w2 + b
x1w1 = x1*w1; x1w1.label = 'x1*w1'
x2w2 = x2*w2; x2w2.label = 'x2*w2'
x1w1x2w2 = x1w1 + x2w2; x1w1x2w2.label = 'x1*w1 + x2*w2'
n = x1w1x2w2 + b; n.label = 'n'
o = n.tanh(); o.label = 'o'
o.backward()
```

[16]:
```
draw_dot(o)
```

[16]:



[17]:
```
# inputs x1,x2
x1 = Value(2.0, label='x1')
x2 = Value(0.0, label='x2')
# weights w1,w2
w1 = Value(-3.0, label='w1')
w2 = Value(1.0, label='w2')
# bias of the neuron
b = Value(6.8813735870195432, label='b')
# x1*w1 + x2*w2 + b
x1w1 = x1*w1; x1w1.label = 'x1*w1'
x2w2 = x2*w2; x2w2.label = 'x2*w2'
x1w1x2w2 = x1w1 + x2w2; x1w1x2w2.label = 'x1*w1 + x2*w2'
n = x1w1x2w2 + b; n.label = 'n'
# ----
e = (2*n).exp()
o = (e - 1) / (e + 1)
# ----
o.label = 'o'
o.backward()
draw_dot(o)
```

[17]:

[18]:
```
print(o.data)
```

```
0.7071067811865477
```

Expression équivalente avec Pytorch Pour que cette section fonctionne, il est nécessaire que PyTorch soit installé sur la machine faisant tourner Jupyter (`pip install torch`).

[19]:
```
import torch
x1 = torch.Tensor([2.0]).double()                ; x1.requires_grad = True
x2 = torch.Tensor([0.0]).double()                ; x2.requires_grad = True
w1 = torch.Tensor([-3.0]).double()               ; w1.requires_grad = True
w2 = torch.Tensor([1.0]).double()                ; w2.requires_grad = True
```

```
b = torch.Tensor([6.8813735870195432]).double()   ; b.requires_grad = True
n = x1*w1 + x2*w2 + b
o = torch.tanh(n)

print(o.data.item())
o.backward()

print('---')
print('x2', x2.grad.item())
print('w2', w2.grad.item())
print('x1', x1.grad.item())
print('w1', w1.grad.item())
```

```
0.7071066904050358
---
x2 0.5000001283844369
w2 0.0
x1 -1.5000003851533106
w1 1.0000002567688737
```

[20]:
```
print(x1)
```

```
tensor([2.], dtype=torch.float64, requires_grad=True)
```

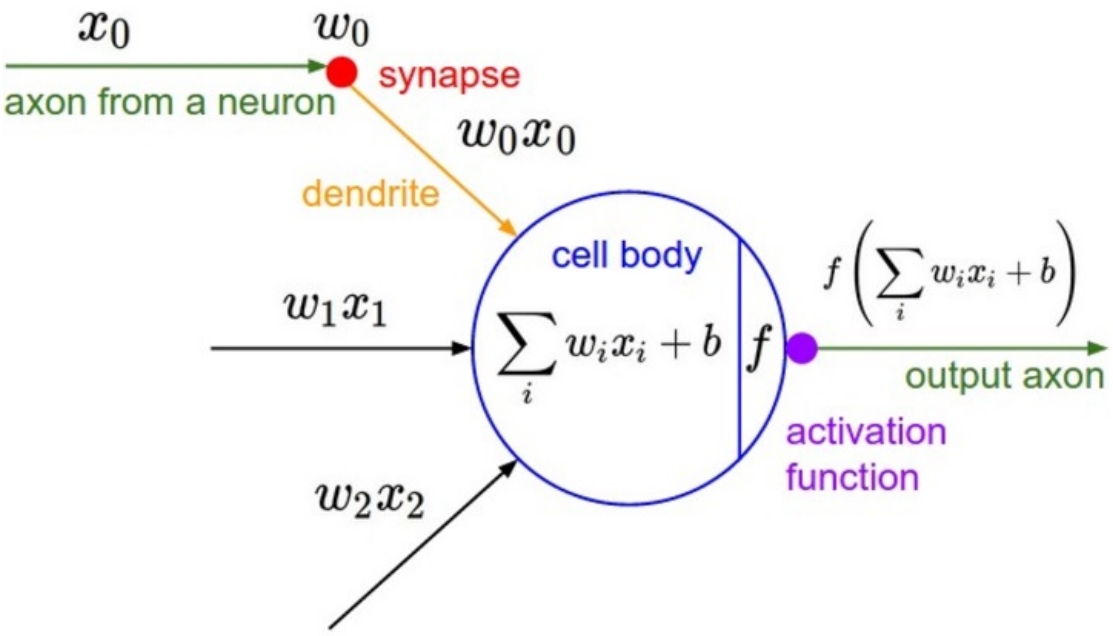
[21]:
```
print(x1.grad)
```

```
tensor([-1.5000], dtype=torch.float64)
```

2 MLP: Multi Layer Perceptron

2.1 Modélisation d'un neurone

Dessin issu de "*neural networks: representation*" de Jeremy Jordan.



Source: https://www.jeremyjordan.me/content/images/2018/01/single_neuron.jpg

Par défaut, dans la classe `Neuron` ci-dessous, on initialise les poids avec une valeur aléatoire (distribution de probabilité uniforme) comprise entre -1 et 1.

[22]:
```
import random

class Neuron:

    def __init__(self, nin):
        self.w = [Value(random.uniform(-1,1), label=f'w{i}') for i in␣
↪range(nin)]
        self.b = Value(random.uniform(-1,1), label='b')

    def __call__(self, x):
        # tanh(w * x + b)
        act = sum((wi*xi for wi, xi in zip(self.w, x)), self.b)
        out = act.tanh()  # Fonction d'activation
        out.label = 'out'
        return out

    def parameters(self):
        return self.w + [self.b]
```

[23]:
```
n = Neuron(5)
n((0.2,0.2,0.3,0.4,0.5))
```

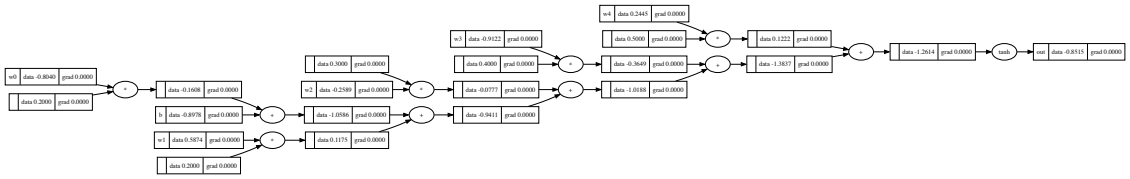
[23]:
```
Value(data=-0.8514542857879747, label=out, grad=0.0)
```

[24]:
```
n.parameters()
```

[24]:
```
[Value(data=-0.8039598110691475, label=w0, grad=0.0),
 Value(data=0.5874188380995722, label=w1, grad=0.0),
 Value(data=-0.2589112549460806, label=w2, grad=0.0),
 Value(data=-0.9122108656376093, label=w3, grad=0.0),
 Value(data=0.24446900172649078, label=w4, grad=0.0),
 Value(data=-0.8977855630313485, label=b, grad=0.0)]
```

[25]:
```
draw_dot(n((0.2,0.2,0.3,0.4,0.5)))
```

[25]:



2.2 Couche de neurones

[26]:
```
class Layer:

  def __init__(self, nin, nout):
    self.neurons = [Neuron(nin) for _ in range(nout)]

  def __call__(self, x):
    outs = [n(x) for n in self.neurons]
    return outs[0] if len(outs) == 1 else outs

  def parameters(self):
    return [p for neuron in self.neurons for p in neuron.parameters()]
```

[27]:
```
l = Layer(2,3)
l((0.2,0.3))
```

[27]:
```
[Value(data=0.3213930604635117, label=out, grad=0.0),
 Value(data=0.31464203078000524, label=out, grad=0.0),
 Value(data=-0.858850222206831, label=out, grad=0.0)]
```

2.3 Multicouches

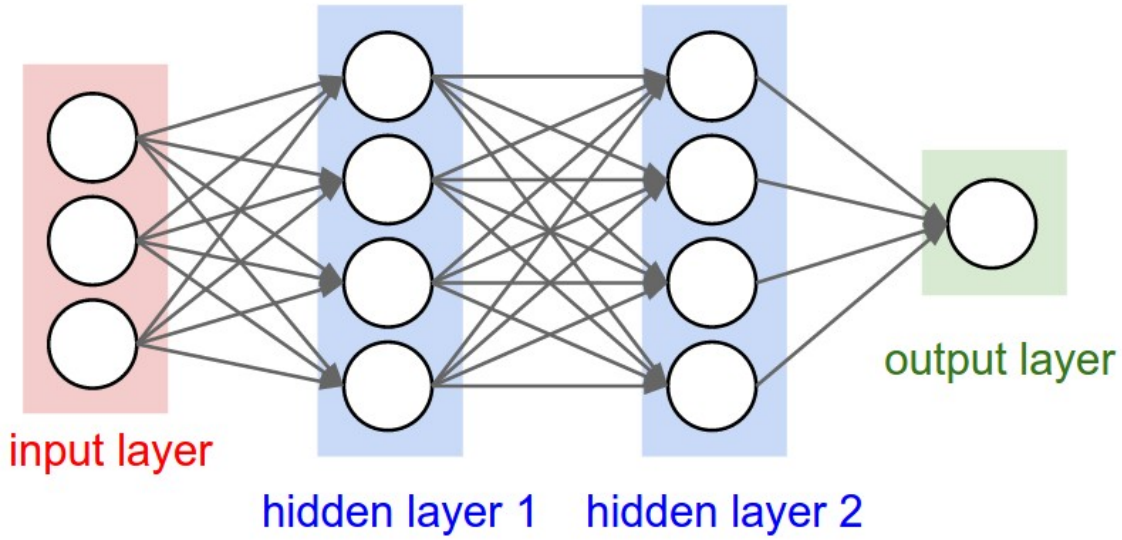
[28]:
```
class MLP:

  def __init__(self, nin, nouts):
    sz = [nin] + nouts
    self.layers = [Layer(sz[i], sz[i+1]) for i in range(len(nouts))]
```

```
  def __call__(self, x):
    for layer in self.layers:
      x = layer(x)
    return x

  def parameters(self):
    return [p for layer in self.layers for p in layer.parameters()]
```

[29]:
```
x = [2.0, 3.0, -1.0]
n = MLP(3, [4, 4, 1])
n(x)
```

[29]:
```
Value(data=0.5428321376017807, label=out, grad=0.0)
```

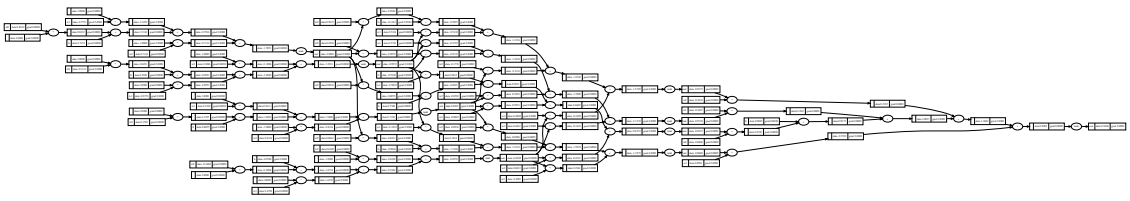
https://cs231n.github.io/neural-networks-1/



Source: https://cs231n.github.io/assets/nn1/neural_net2.jpeg

[30]:
```
draw_dot(n(x))
```

[30]:



3 Apprentissage

3.1 Définition d'une fonction de perte

[31]:
```
# Jeu d'entrainement
xs = [
    [2.0, 3.0, -1.0],  # exemple 1
    [3.0, -1.0, 0.5],  # exemple 2
    [0.5, 1.0, 1.0],   # exemple 3
    [1.0, 1.0, -1.0],  # exemple 4
]
ys = [1.0, -1.0, -1.0, 1.0] # desired targets
ypred = [n(x) for x in xs]  # sortie
ypred
```

[31]:
```
[Value(data=0.5428321376017807, label=out, grad=0.0),
 Value(data=0.8825476224698201, label=out, grad=0.0),
 Value(data=0.24590509211930503, label=out, grad=0.0),
 Value(data=0.4050940086029426, label=out, grad=0.0)]
```

À ce stade, les valeurs de sortie du réseau ne sont pas bonnes, ce qui est normal car nous n'avons réglé aucun paramètre. Nous allons définir une fonction que nous allons chercher à optimiser, ici à minimiser: une fonction de perte utilisant l'erreur quadratique moyenne.

[32]:
```
[(yout - ygt)**2 for ygt, yout in zip(ys, ypred)]
```

[32]:
```
[Value(data=0.20900245440975715, label=, grad=0.0),
 Value(data=3.5439855508667724, label=, grad=0.0),
 Value(data=1.5522794985688142, label=, grad=0.0),
 Value(data=0.3539131386001157, label=, grad=0.0)]
```

[33]:
```
loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))
loss
```

[33]:
```
Value(data=5.65918064244546, label=, grad=0.0)
```

C'est cette valeur que l'on va chercher à minimiser.

3.2 Reprise des étapes: apprentissage manuel

[45]:
```
# Architecture de notre réseau
n = MLP(3, [4, 4, 1])

# Données exemple
xs = [
    [2.0, 3.0, -1.0],  # exemple 1
    [3.0, -1.0, 0.5],  # exemple 2
    [0.5, 1.0, 1.0],   # exemple 3
    [1.0, 1.0, -1.0],  # exemple 4
```

```
]

# Cible
ys = [1.0, -1.0, -1.0, 1.0] # desired targets
```

[46]:
```
# forward pass
ypred = [n(x) for x in xs]
loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))

# backward pass
for p in n.parameters():
    p.grad = 0.0
loss.backward()

# update
for p in n.parameters():
    p.data += -0.1 * p.grad

print(list(map(lambda x: x.data, ypred)))
print(loss.data)
```

```
[0.4020920928583902, -0.1894861802920144, -0.05922087190814421,
0.2550894154311267]
2.4543836642161017
```

3.3 Automatisation de l'apprentissage

[49]:
```
for k in range(100):
    # forward pass
    ypred = [n(x) for x in xs]
    loss = sum((yout - ygt)**2 for ygt, yout in zip(ys, ypred))

    # backward pass
    for p in n.parameters():
        p.grad = 0.0
    loss.backward()

    # update
    for p in n.parameters():
        p.data += -0.1 * p.grad

    print(f"{k} loss={loss.data} {list(map(lambda x: x.data, ypred))}")
```

```
0 loss=0.0002781574011511045 [0.9947032216496835, -0.9913560575993751,
-0.9900334740555171, 0.9912792110086672]
1 loss=0.0002778910086773501 [0.9947058577753756, -0.9913601868709156,
-0.9900382219334123, 0.9912833679881876]
2 loss=0.0002776251162239342 [0.9947084901418856, -0.9913643103734111,
```

```
-0.9900429631794665, 0.991287519178031]
3 loss=0.0002773597223959083 [0.9947111187580135, -0.9913684281201739,
-0.9900476978089977, 0.9912916645915285]
4 loss=0.0002770948258035136 5 [0.9947137436325303, -0.9913725401244734,
-0.990052425837274, 0.9912958042419678]
5 loss=0.000276830425062103 9 [0.9947163647741786, -0.9913766463995367,
-0.9900571472795149, 0.9912999381425945]
6 loss=0.0002765566518792148 9 [0.9947189821916727, -0.9913807469585485,
-0.9900618621508906, 0.9913040663066116]
7 loss=0.0002763031056192206 6 [0.9947215958936985, -0.9913848418146511,
-0.9900665704665226, 0.9913081887471796]
8 loss=0.0002760401841739354 5 [0.9947242058889137, -0.9913889309809447,
-0.9900712722414844, 0.9913123054774174]
9 loss=0.0002757777530919663 4 [0.9947268121859485, -0.9913930144704878,
-0.9900759674908006, 0.9913164165104016]
10 loss=0.0002755158110140051 3 [0.9947294147934046, -0.9913970922962964,
-0.9900806562294483, 0.9913205218591671]
11 loss=0.000275254356585733 8 [0.9947320137198563, -0.991401164471346,
-0.9900853384723558, 0.9913246215367074]
12 loss=0.000274994933884578 092 [0.9947346089738499, -0.99140523100857,
-0.9900900142344047, 0.9913287155559745]
13 loss=0.0002747329052858350 4 [0.9947372005639045, -0.991409291920861,
-0.9900946835304283, 0.9913328039298793]
14 loss=0.000274472905730345 7 [0.9947397884985115, -0.99141334722107,
-0.9900993463752135, 0.9913368866712914]
15 loss=0.000274213388456791 6 [0.9947423727861351, -0.9914173969220075,
-0.9901040027834986, 0.9913409637930397]
16 loss=0.0002739543521354819 3 [0.9947449534352122, -0.9914214410364433,
-0.9901086527699767, 0.9913450353079123]
17 loss=0.000273695795441608 [0.9947475304541527, -0.9914254795771064,
-0.9901132963492928, 0.9913491012286568]
18 loss=0.0002734377170551871 6 [0.9947501038513393, -0.9914295125566853,
-0.9901179335360464, 0.9913531615679803]
19 loss=0.0002731801156610620 4 [0.9947526736351281, -0.9914335399878288,
-0.9901225643447896, 0.9913572163385497]
20 loss=0.000272922989948854 8 [0.9947552398138483, -0.9914375618831455,
-0.9901271887900291, 0.991361265552992]
21 loss=0.0002726663386129816 4 [0.9947578023958024, -0.9914415782552033,
-0.9901318068862255, 0.991365309223894]
22 loss=0.0002724101603525902 6 [0.9947603613892664, -0.9914455891165311,
-0.9901364186477934, 0.9913693473638031]
23 loss=0.000272154453871577 3 [0.9947629168024897, -0.9914495944796184,
-0.9901410240891017, 0.9913733799852267]
24 loss=0.0002718992178785253 7 [0.9947654686436956, -0.9914535943569147,
-0.9901456232244743, 0.9913774071006332]
25 loss=0.000271644451086726 [0.9947680169210811, -0.9914575887608305,
-0.9901502160681893, 0.9913814287224514]
26 loss=0.0002713901522141172 [0.9947705616428171, -0.9914615777037371,
```

-0.9901548026344803, 0.9913854448630712]
27 loss=0.0002711363199832938 [0.9947731028170481, -0.9914655611979671,
-0.9901593829375357, 0.9913894555348436]
28 loss=0.0002708829531214697 3 [0.9947756404518935, -0.9914695392558139,
-0.9901639569914994, 0.9913934607500805]
29 loss=0.000270630050360472 1 [0.9947781745554459, -0.9914735118895325,
-0.9901685248104705, 0.9913974605210554]
30 loss=0.0002703776104366861 5 [0.9947807051357732, -0.9914774791113397,
-0.9901730864085039, 0.9914014548600034]
31 loss=0.000270125631 93209108295 [0.9947832322009169, -0.9914814409334132,
-0.9901776417996107, 0.9914054437791211]
32 loss=0.0002698741140691602 [0.9947857557588934, -0.9914853973678934,
-0.9901821909977574, 0.991409427290567]
33 loss=0.0002696230551209477 [0.9947882758176937, -0.9914893484268822,
-0.990186734016867, 0.9914134054064616]
34 loss=0.0002693724540009601 [0.9947907923852835, -0.9914932941224436,
-0.9901912708708193, 0.9914173781388874]
35 loss=0.0002691223094682143 5 [0.994793305469603, -0.991497234466604,
-0.9901958015734496, 0.9914213454998895]
36 loss=0.0002688726202861761 76 [0.994795815078568, -0.9915011694713524,
-0.990200326138551, 0.991425307501475]
37 loss=0.0002686233852227112 [0.9947983212200685, -0.9915050991486403,
-0.9902048445798729, 0.9914292641556139]
38 loss=0.0002683746030501911 [0.9948008239019703, -0.9915090235103813,
-0.9902093569111221, 0.991433215474239]
39 loss=0.0002681262725453214 [0.9948033231321142, -0.9915129425684531,
-0.9902138631459623, 0.9914371614692457]
40 loss=0.0002678783924892086 [0.9948058189183161, -0.9915168563346958,
-0.9902183632980147, 0.9914411021524925]
41 loss=0.0002676309616673282 [0.9948083112683676, -0.991520764820912,
-0.9902228573808582, 0.9914450375358014]
42 loss=0.0002673839788694755 5 [0.9948108001900355, -0.9915246680388691,
-0.9902273454080298, 0.9914489676309572]
43 loss=0.0002671374428897925 [0.9948132856910629, -0.9915285660002966,
-0.9902318273930236, 0.9914528924497088]
44 loss=0.0002668913525267083 5 [0.9948157677791677, -0.9915324587168886,
-0.9902363033492926, 0.9914568120037681]
45 loss=0.0002666457065829 5 [0.9948182464620443, -0.9915363462003022,
-0.9902407732902473, 0.9914607263048112]
46 loss=0.0002664005038655038 7 [0.9948207217473627, -0.9915402284621586,
-0.9902452372292573, 0.9914646353644778]
47 loss=0.0002661557431855936 5 [0.9948231936427689, -0.9915441055140438,
-0.9902496951796503, 0.991468539194372]
48 loss=0.0002659114233586779 [0.9948256621558853, -0.9915479773675072,
-0.9902541471547133, 0.9914724378060616]
49 loss=0.0002656675432044268 [0.9948281272943102, -0.9915518440340627,
-0.9902585931676916, 0.9914763312110793]
50 loss=0.0002654241015467078 [0.9948305890656182, -0.9915557055251883,

-0.9902630332317898, 0.9914802194209219]
51 loss=0.0002651810972135345 [0.9948330474773605, -0.9915595618523279,
-0.9902674673601719, 0.9914841024470509]
52 loss=0.0002649385290371021 [0.9948355025370644, -0.9915634130268884,
-0.9902718955659612, 0.9914879803008926]
53 loss=0.0002646969639585372163 [0.9948379542522342, -0.9915672590602432,
-0.9902763178622406, 0.991491852993838]
54 loss=0.0002644546965038287 [0.9948404026303508, -0.9915710999637297,
-0.9902807342620525, 0.9914957205372436]
55 loss=0.0002642134298319663 7 [0.9948428476788715, -0.9915749357486506,
-0.9902851447783994, 0.9914995829424305]
56 loss=0.0002639725946867319 5 [0.9948452894052309, -0.9915787664262744,
-0.9902895494242441, 0.9915034402206855]
57 loss=0.000263732189920810 8 [0.9948477278168403, -0.9915825920078347,
-0.9902939482125089, 0.9915072923832606]
58 loss=0.0002634922143909253 6 [0.994850162921088, -0.9915864125045307,
-0.9902983411560771, 0.9915111394413736]
59 loss=0.0002632526669578241 [0.9948525947253397, -0.9915902279275273,
-0.9903027282677919, 0.9915149814062079]
60 loss=0.000263013546486258 8 [0.994855023236938, -0.9915940382879552,
-0.990307109560458, 0.9915188182889126]
61 loss=0.0002627748518449845 [0.9948574484632029, -0.9915978435969114,
-0.9903114850468401, 0.9915226501006031]
62 loss=0.0002625365819067164 [0.994859870411432, -0.9916016438654587,
-0.9903158547396645, 0.9915264768523607]
63 loss=0.0002622987355481437 [0.9948622890889002, -0.9916054391046263,
-0.990320218651618, 0.991530298555233]
64 loss=0.0002620613116498816 [0.9948647045028599, -0.9916092293254094,
-0.9903245767953495, 0.9915341152202339]
65 loss=0.0002618243090964754 3 [0.9948671166605414, -0.9916130145387706,
-0.9903289291834682, 0.991537926858344]
66 loss=0.0002615877267763755 [0.9948695255691526, -0.991616794755638,
-0.9903332758285457, 0.9915417334805103]
67 loss=0.0002613515635835905 26 [0.9948719312358792, -0.9916205699869077,
-0.9903376167431156, 0.9915455350976468]
68 loss=0.0002611158418409288 9 [0.994874333667885, -0.9916243402434415,
-0.9903419519396724, 0.9915493317206341]
69 loss=0.0002608804901585797 5 [0.9948767328723116, -0.991628105536069,
-0.9903462814306735, 0.9915531233603202]
70 loss=0.0002606455577733673 2 [0.9948791288562789, -0.9916318658755867,
-0.9903506052285381, 0.99155691002752]
71 loss=0.0002604110800422961 4 [0.9948815216268849, -0.9916356212727587,
-0.9903549233456476, 0.9915606917330158]
72 loss=0.0002601769959595970 7 [0.9948839111912059, -0.9916393717383161,
-0.990359235794346, 0.9915644684875572]
73 loss=0.0002599433245100157 6 [0.9948862975562964, -0.9916431172829575,
-0.99036354258694, 0.9915682403018614]
74 loss=0.0002597100645035073 7 [0.9948886807291896, -0.9916468579173495,

-0.9903678437356993, 0.9915720071866134]
75 loss=0.0002594772148992945 [0.9948910607168969, -0.9916505936521265,
-0.9903721392528556, 0.9915757691524658]
76 loss=0.0002592477462394707 [0.9948934375264088, -0.9916543244978904,
-0.990376429150605, 0.9915795262100391]
77 loss=0.000259012742607744 [0.994895811164694, -0.9916580504652116,
-0.9903807134411053, 0.9915832783699219]
78 loss=0.0002587811177847791 [0.9948981816387003, -0.9916617715646288,
-0.9903849921364788, 0.991587025642671]
79 loss=0.0002585498990926639 [0.9949005489553541, -0.991665487806648,
-0.9903892652488109, 0.9915907680388114]
80 loss=0.0002583190854728064 [0.994902913121561, -0.9916691992017448,
-0.9903935327901501, 0.9915945055688365]
81 loss=0.0002580886758702298 6 [0.9949052741442054, -0.9916729057603626,
-0.9903977947725098, 0.9915982382432085]
82 loss=0.0002578586692336174 4 [0.994907632030151, -0.9916766074929136,
-0.9904020512078662, 0.9916019660723578]
83 loss=0.0002576290645152722 4 [0.9949099867862405, -0.991680304409779,
-0.99040630210816, 0.991605689066684]
84 loss=0.000257399860671111 8 [0.994912338419296, -0.9916839965213087,
-0.9904105474852961, 0.9916094072365551]
85 loss=0.000257171056660648 7 [0.994914686936119, -0.9916876838378215,
-0.9904147873511436, 0.9916131205923086]
86 loss=0.0002569426514469763 5 [0.9949170323434902, -0.9916913663696055,
-0.9904190217175362, 0.9916168291442506]
87 loss=0.0002567146439967488 4 [0.99491937464817, -0.9916950441269184,
-0.9904232505962718, 0.9916205329026572]
88 loss=0.0002564870332801839 6 [0.9949217138568983, -0.9916987171199864,
-0.9904274739991137, 0.991624231877773]
89 loss=0.0002562598182710213 6 [0.9949240499763948, -0.9917023853590058,
-0.9904316919377895, 0.9916279260798128]
90 loss=0.000256032997946528 [0.9949263830133587, -0.9917060488541427,
-0.9904359044239917, 0.9916316155189605]
91 loss=0.0002558065712874802 4 [0.9949287129744692, -0.991709707615532,
-0.9904401114693784, 0.9916353002053699]
92 loss=0.0002555580537278128 5 [0.9949310398663853, -0.9917133616532797,
-0.9904443130855728, 0.9916389801491646]
93 loss=0.000255355489490621 44 [0.9949333636957461, -0.9917170109774607,
-0.9904485092841633, 0.9916426553604384]
94 loss=0.000255129643162935 [0.9949356844691708, -0.9917206555981201,
-0.9904527000767042, 0.9916463258492546]
95 loss=0.0002549047810429273 [0.9949380021932585, -0.9917242955252741,
-0.9904568854747148, 0.9916499916256472]
96 loss=0.0002546803075442763 [0.9949403168745884, -0.9917279307689078,
-0.9904610654896808, 0.99165365269962]
97 loss=0.0002544562216684653 4 [0.9949426285197206, -0.9917315613389777,
-0.9904652401330535, 0.9916573090811478]
98 loss=0.0002542325224203795 [0.9949449371351949, -0.9917351872454109,

```
-0.9904694094162505, 0.9916609607801753]
99 loss=0.0002540092088083135 [0.9949472427275319, -0.991738808498104,
-0.9904735733506552, 0.9916646078066182]
```

[ ]: