Backpropagation

The Educational Framework (EDF)
Feed-Forward Computation Graphs

\[ v_{k+1} = f_1(v_0, \ldots, v_k) \]

\[ v_{k+2} = f_2(v_0, \ldots, v_{k+1}) \]

\[ \vdots \]

\[ v_{k+d} = f_d(v_0, \ldots, v_{k+d-1}) \]

\[ \ell = f_{d+1}(v_0, \ldots, v_{k+d}) \]

\( \ell \) is a scalar loss.
Backpropagation

\[
y = f(x) \\
z = g(y, x) \\
u = h(z) \\
\ell = u
\]

For now assume all values are scalars.

We can think of each variable as potential input and consider, for example, \( \partial \ell / \partial z \).

Note that \( \partial \ell / \partial z \) depends on the value of \( z \).
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

We will “backpropagate” each assignment in the reverse order.
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ \frac{\partial \ell}{\partial u} = 1 \]
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ \frac{\partial \ell}{\partial u} = 1 \]
\[ \frac{\partial \ell}{\partial z} = \left( \frac{\partial \ell}{\partial u} \right) \left( \frac{\partial h}{\partial z} \right) \] (this uses the value of \( z \))
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ \frac{\partial \ell}{\partial u} = 1 \]
\[ \frac{\partial \ell}{\partial z} = \left( \frac{\partial \ell}{\partial u} \right) \left( \frac{\partial h}{\partial z} \right) \]
\[ \frac{\partial \ell}{\partial y} = \left( \frac{\partial \ell}{\partial z} \right) \left( \frac{\partial g}{\partial y} \right) \text{ (this uses the value of } y \text{ and } x) \]
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ \frac{\partial \ell}{\partial u} = 1 \]
\[ \frac{\partial \ell}{\partial z} = \left( \frac{\partial \ell}{\partial u} \right) \left( \frac{\partial h}{\partial z} \right) \]
\[ \frac{\partial \ell}{\partial y} = \left( \frac{\partial \ell}{\partial z} \right) \left( \frac{\partial g}{\partial y} \right) \]
\[ \frac{\partial \ell}{\partial x} = ??? \]  Oops, we need to add up multiple occurrences.
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

We let \( x.\text{grad} \) be an attribute (as in Python) of object \( x \).

We will accumulate different contributions to \( \partial \ell / \partial x \) into \( x.\text{grad} \).
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ z.\text{grad} = y.\text{grad} = x.\text{grad} = 0 \]
\[ u.\text{grad} = 1 \]

**Loop Invariant:** For any variable \( u \) defined above the red circuit, we have that \( u.\text{grad} \) is \( \partial \ell / \partial u \) as defined by the red circuit.
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ z.\text{grad} = y.\text{grad} = x.\text{grad} = 0 \]
\[ u.\text{grad} = 1 \]

**Loop Invariant**: For any variable \( z \) defined above the red circuit, we have that \( z.\text{grad} \) is \( \partial \ell / \partial z \) as defined by the red circuit.

\[ z.\text{grad} += u.\text{grad} \times \partial h / \partial z \]
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ z.\text{grad} = y.\text{grad} = x.\text{grad} = 0 \]
\[ u.\text{grad} = 1 \]

**Loop Invariant**: For any variable \( y \) defined above the red circuit, we have that \( y.\text{grad} \) is \( \partial \ell / \partial y \) as defined by the red circuit.

\[ z.\text{grad} += u.\text{grad} \times \partial h / \partial z \]
\[ y.\text{grad} += z.\text{grad} \times \partial g / \partial y \]
\[ x.\text{grad} += z.\text{grad} \times \partial g / \partial x \]
Backpropagation

\[ y = f(x) \]
\[ z = g(y, x) \]
\[ u = h(z) \]
\[ \ell = u \]

\[ z.\text{grad} = y.\text{grad} = x.\text{grad} = 0 \]
\[ u.\text{grad} = 1 \]
\[ z.\text{grad} += u.\text{grad} \times \partial h / \partial z \]
\[ y.\text{grad} += z.\text{grad} \times \partial g / \partial y \]
\[ x.\text{grad} += z.\text{grad} \times \partial g / \partial x \]
\[ x.\text{grad} += y.\text{grad} \times \partial f / \partial x \]
The EDF Framework

The educational framework (EDF) is a simple Python-NumPy implementation of a “framework” for defining computation graphs and performing backpropagation. In EDF we write

\[ y = F(x) \]
\[ z = G(y, x) \]
\[ u = H(z) \]
\[ \ell = u \]

This is Python code where variables are bound to objects.
The EDF Framework

The educational frameword (EDF) is a simple Python-NumPy implementation of a “framework” for defining computation graphs and performing backpropagation. In EDF we write

\[ y = F(x) \]
\[ z = G(y, x) \]
\[ u = H(z) \]
\[ \ell = u \]

This is Python code where variables are bound to objects.

- \( x \) is an object in the class \texttt{Value}.
- \( y \) is an object in the class \texttt{F}.
- \( z \) is an object in the class \texttt{G}.
- \( u \) and \( \ell \) are the same object in the class \texttt{H}.
\[ y = F(x) \]

class \( F \):
    def \_init\_(self, x):
        components.append(self)
        self.x = x

    def forward(self):
        self.value = f(self.x.value)

    def backward(self):
        self.x.grad += self.grad* (\partial f/\partial x)  # needs x.value
\[ z = G(y, x) \]

class \( G \):
    def \_init\_ (self, y, x):
        components.append(self)
        self.y = y
        self.x = x

    def forward(self):
        self.value = g(self.y.value, self.x.value)

    def backward(self):
        self.y.grad += self.grad*(\( \frac{\partial g}{\partial y} \))  # needs \( y . \text{value} \) and \( x . \text{value} \)
        self.x.grad += self.grad*(\( \frac{\partial g}{\partial x} \))  # needs \( y . \text{value} \) and \( x . \text{value} \)
The EDF Framework

\[ y = F(x) \]
\[ z = G(y, x) \]
\[ u = H(z) \]

This computation graph has one input and three components.

This is equivalent to

\[ u = H(G(F(x), x)) \]
Backpropagation

def Forward():
    for c in components: c.forward()

def Backward(loss):
    for c in components: c.grad = 0
    for c in params: c.grad = 0
    for c in inputs: c.grad = 0
    loss.grad = 1
    for c in components[::-1]: c.backward()

def SGD(eta):
    for p in params:
        p.value -= eta*p.grad
The Vector Case

\[ y = F(x) \]
\[ z = G(y, x) \]
\[ u = H(z) \]
\[ \ell = u \]

\( x, y \) and \( z \) can be vector-valued.

The loss \( u \) is still a scalar.
The Vector-Valued Class G

class $G$:
    def __init__(self,y,x):
        components.append(self)
        self.y = y
        self.x = x

    def forward(self):
        self.value = g(self.y.value, self.x.value)

    def backward(self):
        self.y.grad += self.grad $\nabla_y g$  #vector-matrix product
        self.x.grad += self.grad $\nabla_x g$  #vector-matrix product
The Jacobian Matrix

In the vector-valued case $\nabla_x g$ is a Jacobian matrix.

\[ \nabla_x g = J \]

\[ J[j, k] = \frac{\partial g[j]}{\partial x[k]} \]
The General Case

Inputs $v_0, \ldots, v_k$

\[ v_{k+1} = F_1(v_0, \ldots, v_k) \]
\[ v_{k+2} = F_2(v_0, \ldots, v_{k+1}) \]
\[ \vdots \]
\[ v_{k+d} = F_d(v_0, \ldots, v_{k+d-1}) \]
\[ \ell = v_{k+d} \]

In general each $v_i$ is tensor-valued.

The computation is a “tensor flow”.
The Tensor-Valued Class $G$

class $G$:

...  

```python
def backward(self):
    self.y.grad += self.grad  \n$$ y \ n g \quad \text{#tensor contraction}$$
    self.x.grad += self.grad  \n$$ x \ n g \quad \text{#tensor contraction}$$
```

The indices of `self.grad` are contracted with the value indices of $g$. 