TTIC 31230, Fundamentals of Deep Learning

David McAllester, April 2017

Backpropagation

The Educational Framework (EDF)

Feed-Forward Computation Graphs

$$
\begin{aligned}
v_{k+1} &= f_1(v_0,\ \ldots,\ v_k) \\
v_{k+2} &= f_2(v_0,\ \ldots,\ v_{k+1}) \\
&\vdots \\
v_{k+d} &= f_d(v_0,\ \ldots,\ v_{k+d-1}) \\
\ell &= f_{d+1}(v_0,\ \ldots,\ v_{k+d})
\end{aligned}
$$

ℓ is a scalar loss.

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

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

$$\begin{aligned} y &= f(x) \\ z &= g(y, x) \\ u &= h(z) \\ \ell &= u \end{aligned}$$

We will “backpopagate” each assignment in the reverse order.

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

$$\partial \ell / \partial u = 1$$

Backpropagation

$$\begin{aligned} y &= f(x) \\ z &= g(y, x) \\ u &= h(z) \\ \ell &= u \end{aligned}$$

$\partial \ell / \partial u = 1$

$\partial \ell / \partial z = (\partial \ell / \partial u)\ (\partial h / \partial z)$ (this uses the value of z)

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

$\partial \ell / \partial u = 1$

$\partial \ell / \partial z = (\partial \ell / \partial u)\ (\partial h / \partial z)$

$\partial \ell / \partial y = (\partial \ell / \partial z)\ (\partial g / \partial y)$ (this uses the value of y and x)

Backpropagation

$$\begin{aligned} y &= f(x) \\ z &= g(y, x) \\ u &= h(z) \\ \ell &= u \end{aligned}$$

$\partial \ell / \partial u = 1$

$\partial \ell / \partial z = (\partial \ell / \partial u)\ (\partial h / \partial z)$

$\partial \ell / \partial y = (\partial \ell / \partial z)\ (\partial g / \partial y)$

$\partial \ell / \partial x = ???$ Oops, we need to add up multiple occurrences.

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

We let x.grad be an attribute (as in Python) of object x.

We will accumulate different contributions to $\partial \ell / \partial x$ into x.grad.

Backpropagation

$$\begin{aligned} y &= f(x) \\ z &= g(y, x) \\ u &= h(z) \\ \ell &= u \end{aligned}$$

$z.\mathrm{grad} = y.\mathrm{grad} = x.\mathrm{grad} = 0$

$u.\mathrm{grad} = 1$

Loop Invariant: For any variable u defined above the red circuit, we have that $u.\mathrm{grad}$ is $\partial\ell/\partial u$ as defined by the red circuit.

Backpropagation

$$\begin{aligned} y &= f(x) \\ z &= g(y, x) \\ u &= h(z) \\ \ell &= u \end{aligned}$$

$z.\mathrm{grad} = y.\mathrm{grad} = x.\mathrm{grad} = 0$

$u.\mathrm{grad} = 1$

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

$z.\mathrm{grad}$ **+=** $u.\mathrm{grad} * \partial h / \partial z$

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

$z.\mathrm{grad} = y.\mathrm{grad} = x.\mathrm{grad} = 0$

$u.\mathrm{grad} = 1$

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

$z.\mathrm{grad}$ **+=** $u.\mathrm{grad} * \partial h / \partial z$

$y.\mathrm{grad}$ **+=** $z.\mathrm{grad} * \partial g / \partial y$

$x.\mathrm{grad}$ **+=** $z.\mathrm{grad} * \partial g / \partial x$

Backpropagation

$$
\begin{aligned}
y &= f(x) \\
z &= g(y, x) \\
u &= h(z) \\
\ell &= u
\end{aligned}
$$

$$z.\mathrm{grad} = y.\mathrm{grad} = x.\mathrm{grad} = 0$$

$$u.\mathrm{grad} = 1$$

$$z.\mathrm{grad} \mathrel{+}= u.\mathrm{grad} * \partial h / \partial z$$

$$y.\mathrm{grad} \mathrel{+}= z.\mathrm{grad} * \partial g / \partial y$$

$$x.\mathrm{grad} \mathrel{+}= z.\mathrm{grad} * \partial g / \partial x$$

$$x.\mathrm{grad} \mathrel{+}= y.\mathrm{grad} * \partial f / \partial x$$

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

$$
\begin{aligned}
y &= F(x) \\
z &= G(y, x) \\
u &= H(z) \\
\ell &= u
\end{aligned}
$$

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

$$\begin{aligned} y &= F(x) \\ z &= G(y, x) \\ u &= H(z) \\ \ell &= u \end{aligned}$$

This is Python code where variables are bound to objects.

x is an object in the class `Value`.

y is an object in the class F.

z is an object in the class G.

u and ℓ are the same object in the class 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*(∂f/∂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*(∂g/∂y)    #needs y.value and x.value
        self.x.grad += self.grad*(∂g/∂x)    #needs y.value and x.value
```

The EDF Framework

$$
\begin{aligned}
y &= F(x) \\
z &= G(y, x) \\
u &= H(z)
\end{aligned}
$$

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

$$
\begin{aligned}
y &= F(x) \\
z &= G(y, x) \\
u &= H(z) \\
\ell &= u
\end{aligned}
$$

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 ∇_y g        #vector-matrix product
        self.x.grad += self.grad ∇_x g        #vector-matrix product
```

The Jacobian Matrix

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

$$\nabla_x \, g = \mathcal{J}$$

$$\mathcal{J}[j, k] = \frac{\partial g[j]}{\partial x[k]}$$

The General Case

$$\text{Inputs } v_0, \ldots, v_k$$

$$\begin{aligned} v_{k+1} &= F_1(v_0, \ldots, v_k) \\ v_{k+2} &= F_2(v_0, \ldots, v_{k+1}) \\ &\vdots \\ v_{k+d} &= F_d(v_0, \ldots, v_{k+d-1}) \\ \ell &= v_{k+d} \end{aligned}$$

In general each v_i is tensor-valued.

The computation is a "tensor flow".

The Tensor-Valued Class G

class G:

...

def backward(self):

self.y.grad **+=** self.grad $\nabla_y \, g$ #tensor contraction

self.x.grad **+=** self.grad $\nabla_x \, g$ #tensor contraction

The indeces of self.grad are contracted with the value indeces of g.