Backpropagation

Anupam Datta

CMU

Spring 2018

Story so far

- Image classification problem
- Linear models
  - Score function
  - Loss function
  - Learning

- Learning as optimization
  - Gradient descent (batch, mini-batch, stochastic)
  - Second-order methods (Newton's method)

HW1

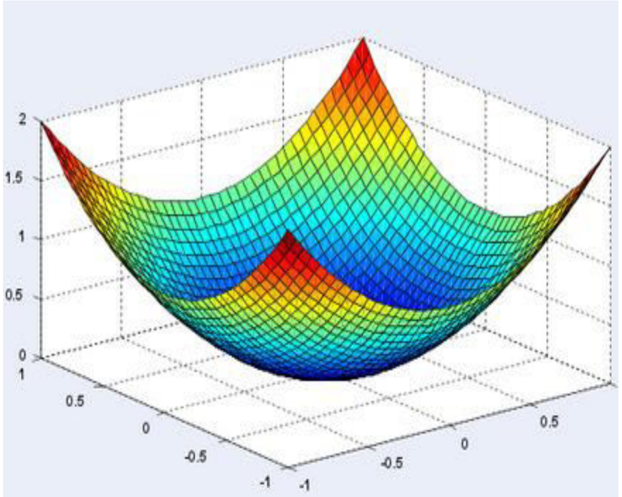
Today

- Learning as optimization
  - Gradient descent (batch, mini-batch, stochastic)
  - Second-order methods (Newton’s method)
  - Require computing gradients

  - Backpropagation
    - Technique for computing gradients recursively
    - Key technique for training deep networks

Gradients

- Consider $f(X) = f(x_1, x_2, \dots, x_n)$
- $\nabla f(X) = \begin{bmatrix} \frac{\partial f(X)}{\partial x_1} & \frac{\partial f(X)}{\partial x_2} & \cdots & \frac{\partial f(X)}{\partial x_n} \end{bmatrix}$



Computing gradients analytically

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$$

Derivatives measure sensitivity

$$x = 4, y = -3 \qquad f(x, y) = -12 \qquad \frac{\partial f}{\partial x} = -3$$

If we were to increase x by a tiny amount, the effect on the whole expression would be to decrease it (due to the negative sign), and by three times that amount.

A composed function

$$f(x, y, z) = (x + y)z$$

$$q = x + y \quad f = qz$$

Chain rule

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Chain rule applied

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

$$q = x + y \qquad f = qz$$

$$\frac{\partial f}{\partial q} = z \qquad \frac{\partial q}{\partial x} = 1$$

$$\frac{\partial f}{\partial x} = z$$

Backpropagation on example function

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

Backpropagation illustrated

Forward pass

Backward pass

$$f(x, y, z) = (x + y)z$$

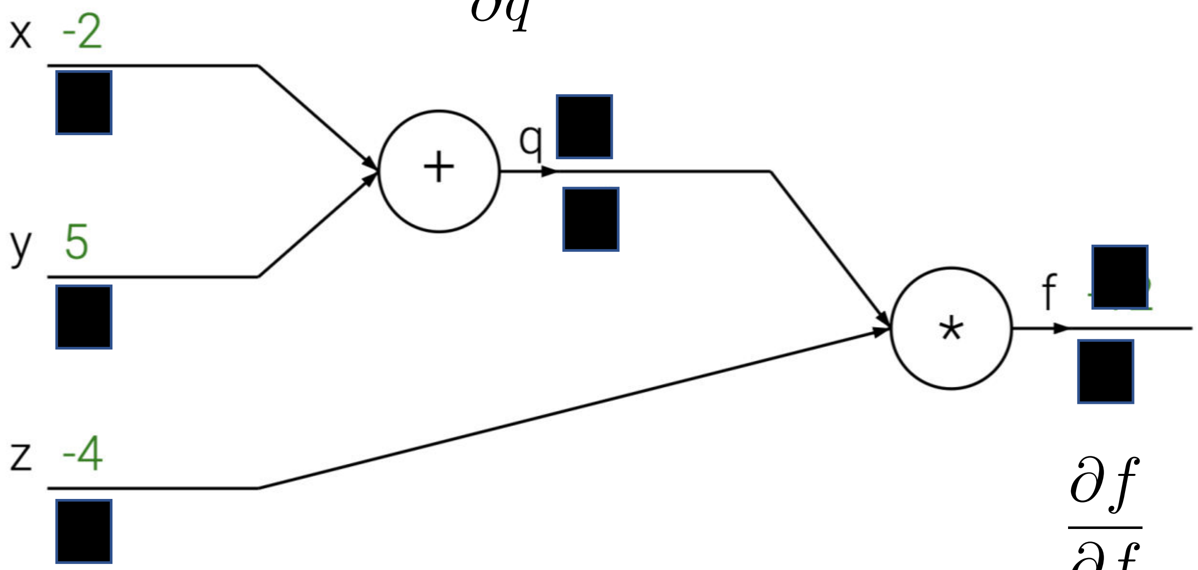
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial z} = q$$

$$Compute: \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

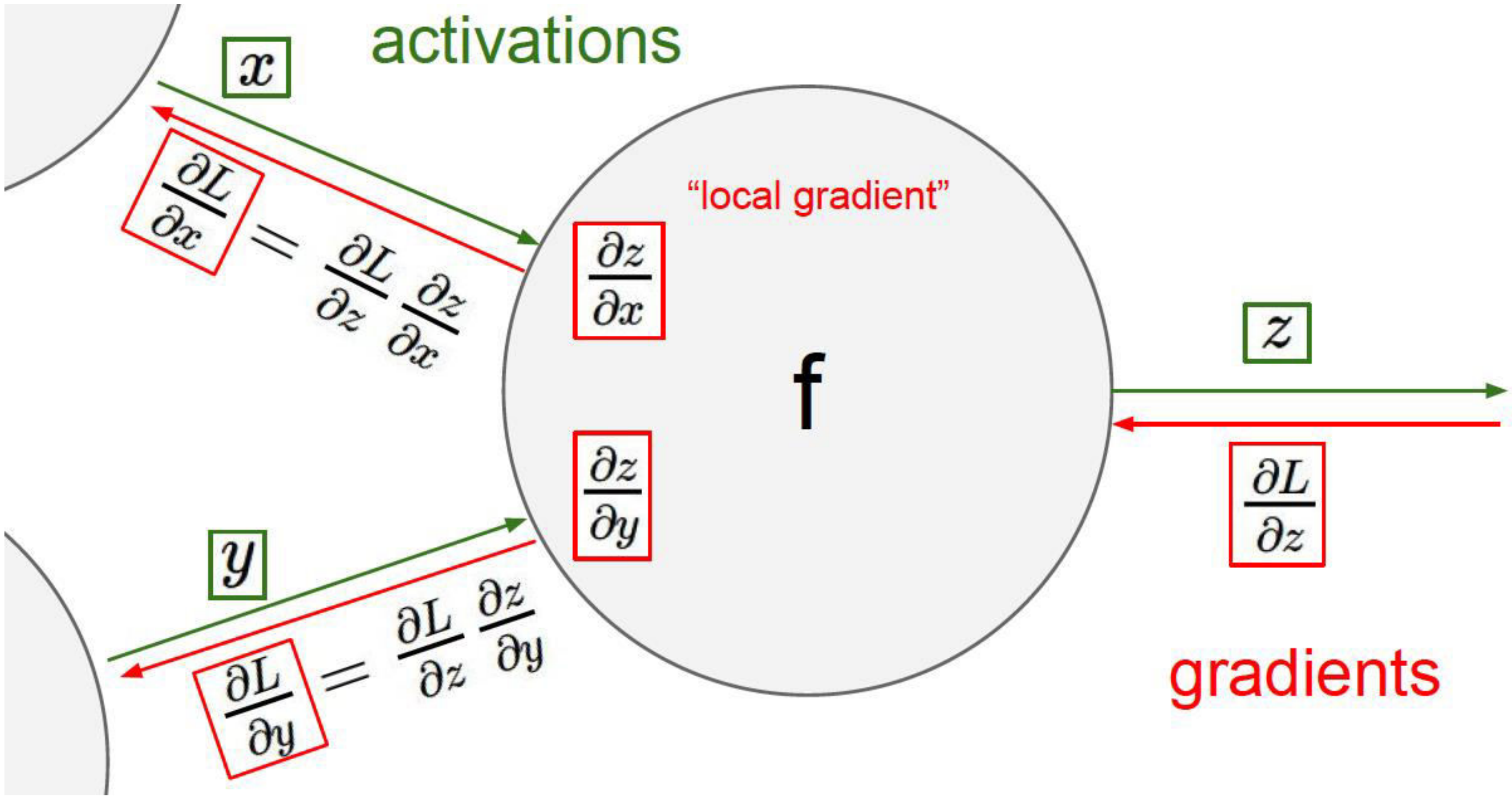
$$\frac{\partial f}{\partial q} = z$$



$$\frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial z} = q$$

Backpropagation: key local step



Backpropagation: key ideas

- Gradients computed locally
- Gradient of interest computed by recursive applications of chain rule

Backpropagation in practice

- Staged computation
  - Carefully decompose complex function to easily compute gradients

Backpropagation in practice

- Staged computation example

$$f(x, y) = \frac{x + \sigma(y)}{\sigma(x) + (x + y)^2}$$

Backpropagation in practice

- Staged computation example: decomposing for forward pass

$$f(x, y) = \frac{x + \sigma(y)}{\sigma(x) + (x + y)^2}$$

$$f(x,y) = \frac{x+\sigma(y)}{\sigma(x)+(x+y)^2} \quad \text{where } \sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma(y) = \frac{1}{1+e^{-y}}$$

$$num = x + \sigma(y)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$xpy = x + y$$

$$xpysqr = (xpy)^2$$

$$den = \sigma(x) + xpysqr$$

$$invden = \frac{1}{den}$$

$$f = num * invden$$

Backpropagation in practice

- Staged computation example:
  backward pass

$$f(x,y) = \frac{x+\sigma(y)}{\sigma(x)+(x+y)^2}$$

Backward pass reuses variables computed in
forward pass (cache them!)

Backward pass:

$$\frac{\partial f}{\partial num} = invden \quad , \quad \frac{\partial f}{\partial invden} = num$$

$$\frac{\partial f}{\partial den} = \frac{\partial invden}{\partial den} \cdot \frac{\partial f}{\partial invden}$$

$$= \left(-\frac{1}{(den)^2}\right) \cdot num$$

$$\vdots$$

Backpropagation in practice

- Staged computation example: forward pass code

```
x = 3 # example values
y = -4

# forward pass
sigy = 1.0 / (1 + math.exp(-y)) # sigmoid in numerator   #(1)
num = x + sigy # numerator                                #(2)
sigx = 1.0 / (1 + math.exp(-x)) # sigmoid in denominator #(3)
xpy = x + y                                               #(4)
xpysqr = xpy**2                                           #(5)
den = sigx + xpysqr # denominator                         #(6)
invden = 1.0 / den                                        #(7)
f = num * invden # done!                                  #(8)
```

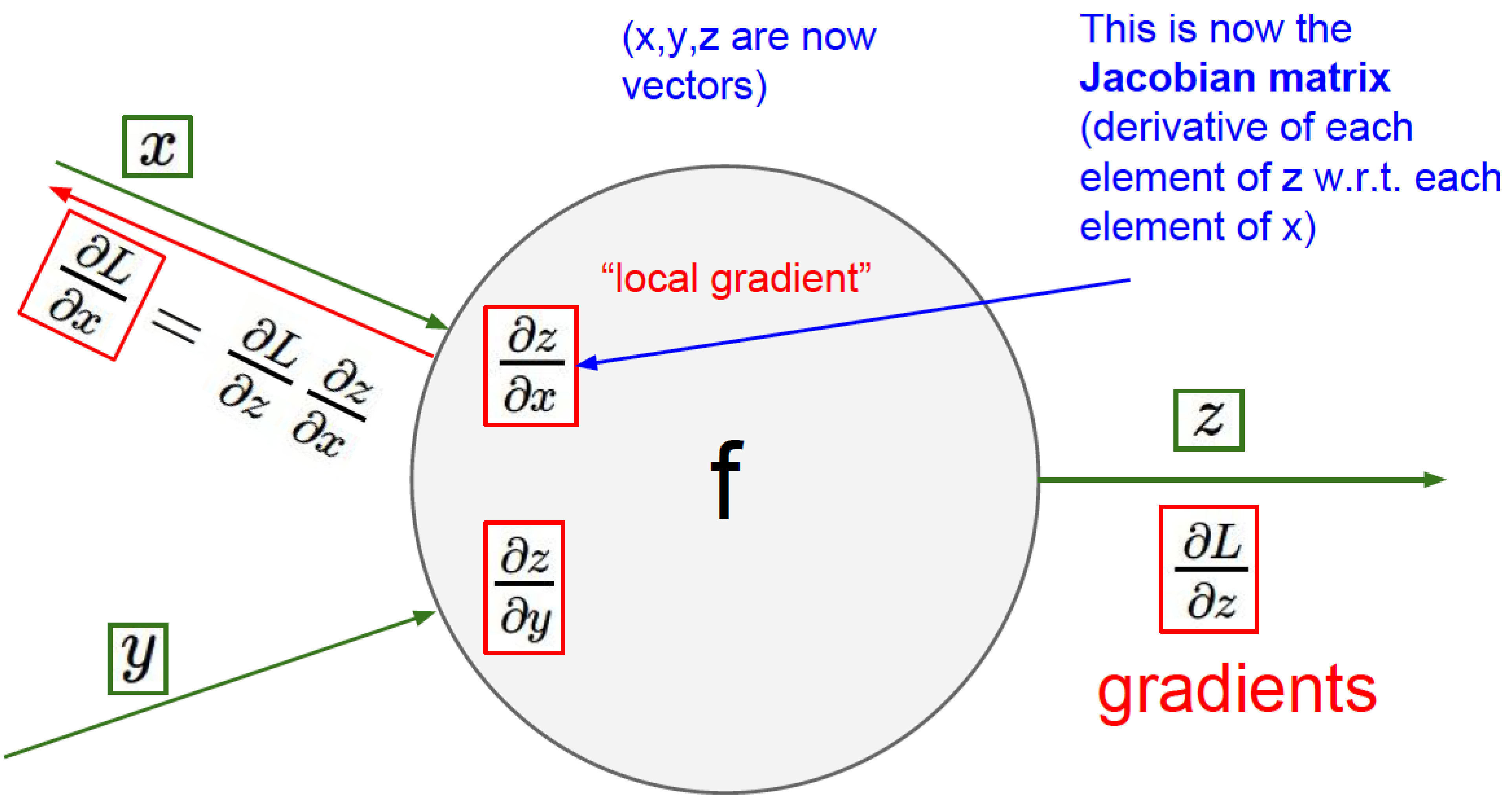
Backpropagation in practice

- Staged computation example: backward pass code

dw in code denotes $\dfrac{\partial f}{\partial w}$

```
# backprop f = num * invden
dnum = invden # gradient on numerator                             #(8)
dinvden = num                                                     #(8)
# backprop invden = 1.0 / den
dden = (-1.0 / (den**2)) * dinvden                                #(7)
# backprop den = sigx + xpysqr
dsigx = (1) * dden                                                #(6)
dxpysqr = (1) * dden                                              #(6)
# backprop xpysqr = xpy**2
dxpy = (2 * xpy) * dxpysqr                                        #(5)
# backprop xpy = x + y
dx = (1) * dxpy                                                   #(4)
dy = (1) * dxpy                                                   #(4)
# backprop sigx = 1.0 / (1 + math.exp(-x))
dx += ((1 - sigx) * sigx) * dsigx # Notice += !! See notes below  #(3)
# backprop num = x + sigy
dx += (1) * dnum                                                  #(2)
dsigy = (1) * dnum                                                #(2)
# backprop sigy = 1.0 / (1 + math.exp(-y))
dy += ((1 - sigy) * sigy) * dsigy                                 #(1)
```

Gradients for vectorized code



Gradients for vectorized code

- Details of
  - Jacobian matrix
  - Chain rule with vectors and matrices

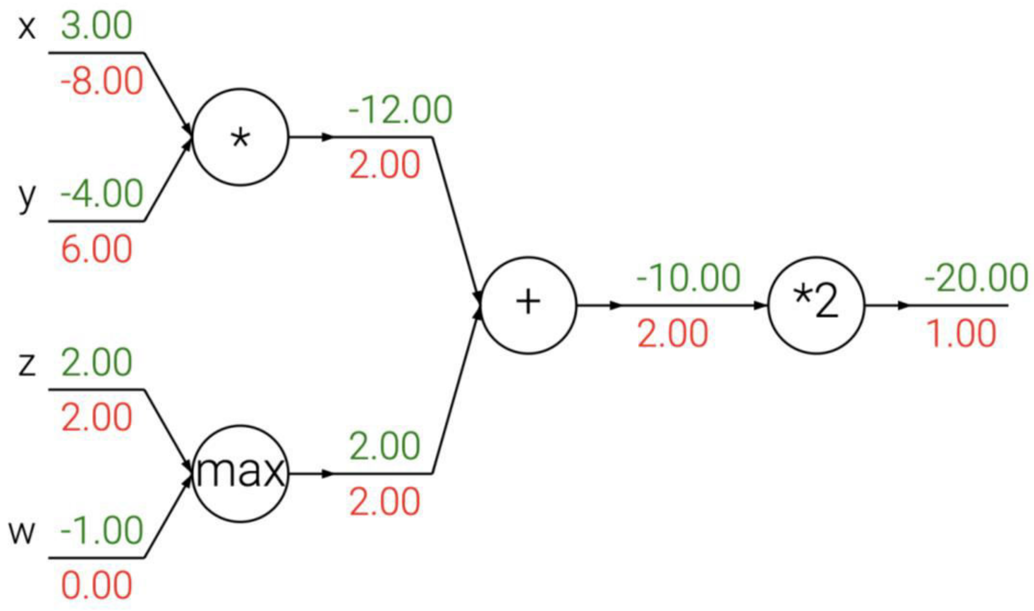
- Work out on paper

- Review notes: http://cs231n.stanford.edu/vecDerivs.pdf

Acknowledgment

Based in part on material from Stanford CS231n
http://cs231n.github.io/

Patterns in backward flow



- add gate: distributes gradient equally to its inputs
- max gate: routes gradient of output to max input
- mul gate: swaps input activations and multiplies by gradient