Backpropagation

Spring 2020
Story so far

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

• Learning as optimization
  • Gradient descent (batch, mini-batch, stochastic)
Today

• Learning as optimization
  • Gradient descent (batch, mini-batch, stochastic)
  • Require computing gradients
• Backpropagation
  • Technique for computing gradients recursively
  • Key technique for training deep networks
Gradients

• Consider $f(X) = f(x_1, x_2, ..., x_n)$
• $\nabla f(X) = \left[ \frac{\partial f(X)}{\partial x_1} \quad \frac{\partial f(X)}{\partial x_2} \quad ... \quad \frac{\partial f(X)}{\partial x_n} \right]$
Computing gradients analytically

\[ f(x, y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x \]

\[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] = [y, x] \]
Derivatives measure sensitivity

\[ x = 4, \quad y = -3 \quad f(x, y) = -12 \quad \frac{\partial f}{\partial x} = -3 \]

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

\[ f(x, y, z) = (x + y)z \]

\[ f = qz \quad q = x + y \]

\[ \frac{\partial f}{\partial q} = z \quad \frac{\partial q}{\partial x} = 1 \]

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
Chain rule on example function

```python
# 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

\[ f(x, y, z) = (x + y)z \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q \]

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

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]

\[ \frac{\partial f}{\partial q} = z \]

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = 1 \]

\[ \frac{\partial f}{\partial z} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial z} = q \]
Backpropagation: key local step

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y} \]
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

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

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

\[ \frac{\partial f}{\partial x} = \frac{1}{e^{-x} + 1 + (x + y)^2} - \frac{(x + \frac{1}{e^{-y} + 1}) \left( \frac{e^{-x}}{(e^{-x} + 1)^2} + 2(x + y) \right)}{\left( \frac{1}{e^{-x} + 1} + (x + y)^2 \right)^2} \]
Backpropagation in practice

• Staged computation example: decomposing for forward pass

\[ f(x, y) = \frac{x + \sigma(y)}{\sigma(x) + (x + y)^2} \]
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!)
Backpropagation in practice

• Staged computation example: forward pass code

```python
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)
pxy = x + y  #(4)
xpxsqr = pxy**2  #(5)
den = sigx + xpxsqr  # denominator  #(6)
invden = 1.0 / den  #(7)
f = num * invden  # done!  #(8)
```
Chain rule, generalized

\[ f(x, y, z) = (x + y)z \]
\[ q = x + y \]
\[ f = qz \]

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
\[ + 0 \]

\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} + \frac{\partial f}{\partial z} \frac{\partial z}{\partial x} \]

In general:

\[ f(a_1, a_2, ..., a_n) \]
\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial a_1} \frac{\partial a_1}{\partial x} + \frac{\partial f}{\partial a_2} \frac{\partial a_2}{\partial x} + ... + \frac{\partial f}{\partial a_n} \frac{\partial a_n}{\partial x} \]
Backpropagation in practice

• Staged computation example: backward pass code

```python
# 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
dsигx = (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) * dsигx  # Notice += !! See notes below  # (3)
# backprop num = x + sigy
dx += (1) * dnum  # (2)
dsигy = (1) * dnum  # (2)
# backprop sigy = 1.0 / (1 + math.exp(-y))
dy += ((1 - sigy) * sigy) * dsигy  # (1)
```
Gradients for vectorized code

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}
\]

(x, y, z are now vectors)

This is now the Jacobian matrix (derivative of each element of z w.r.t. each element of x)

\[
\frac{\partial z}{\partial x}
\]

\[
\frac{\partial z}{\partial y}
\]

\[
\frac{\partial L}{\partial z}
\]

gradadients
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/
• Spring 2019 course
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