Lecture 3 Optimization

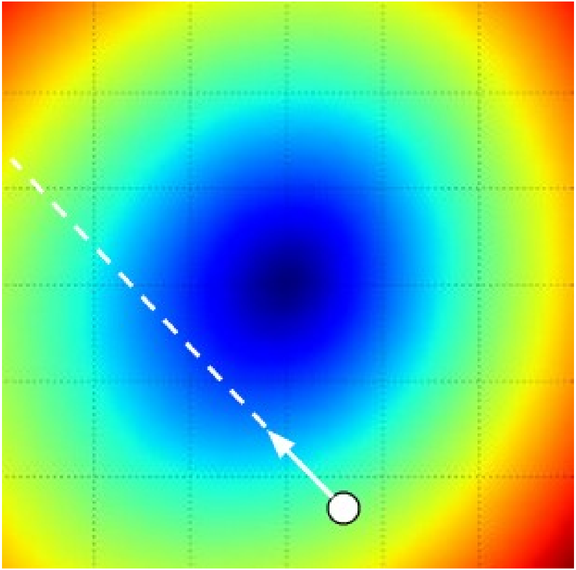
Fei Tan

Department of Economics
Chaifetz School of Business
Saint Louis University

E6930 Introduction to Artificial Neural Networks

June 19, 2024

Following Gradient



The Road Ahead...

- Gradient descent: analytic vs. numerical gradient, local search, learning rate
- Backpropagation: chain rule, forward-backward pass

Analytic Gradient

Derivative of 1-D function

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Gradient of multi-D function
  - vector of partial derivatives in each dimension
  - examples of 2-D function

$$\begin{aligned} f(x,y) = xy \quad &\rightarrow \quad \nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial x}\right] = [y, x] \\ f(x,y) = x + y \quad &\rightarrow \quad \nabla f = [1, 1] \\ f(x,y) = \max(x,y) \quad &\rightarrow \quad \nabla f = [\mathbb{1}(x \geq y), \mathbb{1}(x \leq y)] \end{aligned}$$

Numerical Gradient

```
import numpy as np

def num_grad(f, x):  # finite difference method
    fx = f(x)
    grad = np.zeros(x.shape)
    h = 0.00001
    it = np.nditer(x, flags=['multi_index'],
        op_flags=['readwrite'])
    while not it.finished:
        ix = it.multi_index
        old_value = x[ix]
        x[ix] = old_value + h
        fxh = f(x)
        x[ix] = old_value
        grad[ix] = (fxh - fx) / h # alternatively
            [f(x+h)-f(x-h)]/2h
        it.iternext()
    return grad
```

Gradient Descent

- Repeated local search to minimize loss function
  - update in *negative* gradient direction
  - validate learning rate (step size)
- Mini-batch/stochastic gradient descent

```
while True:
    data_batch = sample_data(training_data,
        256)
    weights_grad = eval_grad(loss_fun,
        data_batch, weights)
    weights += - step_size * weights_grad
```

Backpropagation

Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- Example of composite function

$$f(v(p(x,y),q(z,w))) = 2\ \underbrace{\Big[\ \underbrace{xy}_{p\ \ (*\ \text{gate})} \ +\ \underbrace{\max(z,w)}_{q\ \ (\max\ \text{gate})}\ \Big]}_{v\ \ (+\ \text{gate})}$$

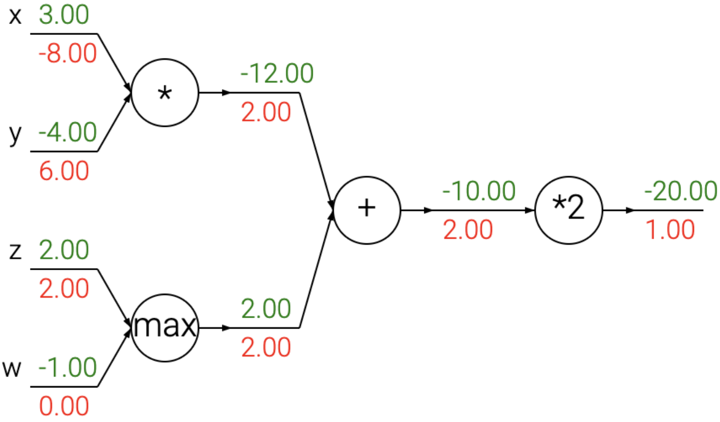
  - forward pass: $[x,y,z,w] = [3,-4,2,-1]$, $f = -20$
  - backward pass: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial w}\right] = [-8,6,2,0]$

Backpropagation (Cont'd)

```
# Forward pass
x = 3; y = -4; z = 2; w = -1
p = x * y        # -12
q = max(z, w)    # 2
v = p + q        # -10
f = 2 * v        # -20

# Backward pass
dfdv = 2
dvdp = 1
dpdx = y
dpdy = x
dvdq = 1
dqdz = (z > w)
dqdw = (w > z)
dfdx = dfdv * dvdp * dpdx  # -8
dfdy = dfdv * dvdp * dpdy  # 6
dfdz = dfdv * dvdq * dqdz  # 2
dfdw = dfdv * dvdq * dqdw  # 0
```

PyTorch's Computational Graph



References

- cs231n.stanford.edu – CS231n: Deep Learning for Computer Vision
- pytorch.org – PyTorch package for deep learning
- github.com/karpathy/micrograd – tiny autograd engine by Andrej Karpathy