Lecture 29

Monday, April 7, 2025 9:30 AM

Let us consider an application of neuron networks for solving the differential equation $u' = x$ (without a specified initial condition). We know that the correct solution is $u(x) = \frac{x^2}{2} + C$.

Consider the simplest neuron network with one layer and one node:

$$x \to z \to \sigma(z) \to y = \tilde{u}(x)$$

Here, $z = ax + b$ and $y = c\sigma(z) + d = c\sigma(ax + b) + d$. The activation function can be any nonlinear function, such as $\sigma(x) = \sin x$.

We wish, by choosing suitable coefficients a, b, c, d and activation function σ, that $\tilde{u}$ satisfy $\tilde{u}' = x$ for all values of x. With a fixed choice of function σ, it seems almost impossible to choose a, b, c, d such that $\tilde{u}'(x) = x$ *for all* x. The difficulty is the high demand "for all".

We will relax the requirement "for all" to for certain prescribed values of x, say $x_1, x_2, \dots, x_n$, called *grid points*. That is to require $\tilde{u}(x_i) - x_i = 0$ for all $i = 1,2 \dots, n$. These are n equations, while we have 4 unknown coefficients to work with. The new requirement demands that $n \leq 4$, which lays quite a severe restriction on n. We further relax this requirement by only requiring the quantity

$$\phi = \sum_{i=1}^{n} (\tilde{u}(x_i) - x_i)^2$$

to be as small as possible. ϕ is called an *error function* because it measures the error. It is also called a *cost function* because it is to be minimized. You can notice that ϕ only depends on a, b, c, d.

$$\phi(a, b, c, d) = \sum_{i=1}^{n} (c\sigma(ax_i + b) + d - x_i)^2$$

Therefore, the problem is to find 4 numbers a, b, c, d that minimize the function $\phi(a, b, c, d)$. You already knew how to do this numerically--by using Gradient Descent method:

Initial guess: a_0, b_0, c_0, d_0
Update: $(a_{n+1}, b_{n+1}, c_{n+1}, d_{n+1}) = (a_n, b_n, c_n, d_n) - \alpha \nabla \phi(a_n, b_n, c_n, d_n)$
where $\alpha > 0$ is the learning rate.

After a number of steps, we stop and take the value of a, b, c, d to be a_n, b_n, c_n, d_n. These values gives us a function $\tilde{u}(x) = c\sigma(ax_i + b) + d$, which is an approximation of the true function u.

Since u is not unique ($u = x^2/2 + C$), how do we know which of these possibilities of u is the one that $\tilde{u}$ approximate? Recall that $\tilde{u}$ approximates u in the sense that $\tilde{u}'(x) \approx x$. Thus, $\tilde{u}$ may not approximate any functions of the function $\frac{x^2}{2} + C$.

Python code:

```
from autograd.numpy import*
from autograd import elementwise_grad as diff
from matplotlib.pyplot import*

x_grid = np.linspace(0.0,0.5,9)
a0,b0,c0,d0 = 1.,1.,1.,1.
alpha = 0.1
num_iter = 1000

def sigma(x):
    return sin(x)
def z(x,a,b,c,d): return a*x + b
def u(x,a,b,c,d): return c*sigma(z(x,a,b,c,d)) + d
def du(x,a,b,c,d): return diff(u,0)(x,a,b,c,d)
def phi(a,b,c,d):
    return sum([(du(t,a,b,c,d) - t)**2 for t in x_grid])

def gradd(phi,a0,b0,c0,d0,alpha,num_iter):
    a,b,c,d = a0,b0,c0,d0
    dphi = diff(phi,(0,1,2,3))
    for i in range(num_iter):
        gradient = array(dphi(a,b,c,d))
        a,b,c,d = array([a,b,c,d]) - alpha*gradient
    return a,b,c,d

a,b,c,d = gradd(phi,a0,b0,c0,d0,alpha,num_iter)

print('[a,b,c,d] =',array([a,b,c,d]))
print("Initial error: ",phi(a0,b0,c0,d0))
print("Final error: ",phi(a,b,c,d))
x = linspace(0,0.5,20)
du_appr = array([du(t,a,b,c,d) for t in x])
du_exact = x
print("Max absolute difference: ",max(abs(du_appr-du_exact)))
plot(x,du_appr,label='approximate sol')
plot(x,du_exact,label='exact sol')
legend()
show()
```