DATA605

Week14+15 Lecture

Outline

1. Final Project
    a. Review each section
2. Review some of hw13 homework
3. Taylor series approximations
4. Gradient Descent

Final Project Review

Watch video for commentary on final project

$$\int \sin(x)\cos(x)\,dx$$

$u = \sin(x)$ $\quad du = \cos(x)\,dx$

$du = \cos(x)\,dx$ $\quad v = \sin(x)$

$$\int \sin(x)\cos(x)\,dx = \sin(x)\sin(x) - \int \sin(x)\cos(x)\,dx$$

$$2\times\int \sin(x)\cos(x)\,dx = \sin^2(x)$$

$$\boxed{\int \sin(x)\cos(x)\,dx = \frac{1}{2}\sin^2(x) + C}$$

$$\sin^2(x) + \cos^2(x) = 1$$

$$= \frac{1}{2}\left[1 - \cos^2(x)\right] + C$$

$$= \frac{1}{2} - \frac{1}{2}\cos^2(x) + C$$

$$\boxed{= -\frac{1}{2}\cos^2(x) + C' \quad \text{where } C' = C + \frac{1}{2}}$$

$$\int x^2 e^x \, dx$$

$u = x^2 \qquad dv = e^x dx$

$\therefore du = 2x\,dx \qquad \therefore v = e^x$

$$\int x^2 e^x dx = x^2 e^x - \int e^x 2x \, dx$$

integration by parts again!

$u = 2x \qquad dv = e^x dx$

$\therefore du = 2dx \qquad \therefore v = e^x$

$$\int e^x 2x\,dx = 2xe^x - \int 2e^x dx$$

$$= 2xe^x - 2\int e^x dx$$

$$= 2xe^x - 2e^x$$

$$\int x^2 e^x dx = x^2 e^x - (2xe^x - 2e^x)$$

$$= x^2 e^x - 2xe^x + 2e^x$$

$$= e^x (x^2 - 2x + 2)$$

$$\frac{d}{dx}\left(x \cdot \cos(x)\right)$$

$$= \frac{d}{dx}(x) \cdot \cos(x) + x \cdot \frac{d}{dx}(\cos(x))$$

$$= \cos(x) - x \cdot \sin(x)$$

Product Rule: $\frac{d}{dx}(f(x)g(x)) = f'(x)g(x) + f(x)g'(x)$

$$\frac{d}{dx}\left(e^{x^4}\right)$$

$u = x^4 \Rightarrow \frac{du}{dx} = 4x^3$

$$= \frac{d}{du}\left(e^{u}\right) \cdot \frac{du}{dx}$$

Chain Rule: $\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g'(x)$

$$= e^{u} \cdot 4x^3$$

$$= e^{x^4} \cdot 4x^3$$

$$= 4x^3 e^{x^4}$$

Code for approximating derivatives

https://github.com/jquacinella/DATA605/blob/master/Week15%20Lecture%20Notes/derivative.py

Code for approximating integrals

https://github.com/jquacinella/DATA605/blob/master/Week15%20Lecture%20Notes/integrate.py

Taylor Series

Taylor Series is used to represent functions as an infinite sum of polynomial terms that are calculated using a function’s derivatives evaluated at a single point.

Taylor series is frequently used with a finite number of polynomial terms to approximate a function.

The higher the degree of the Taylor polynomial used, the better the approximation.

Since its an infinite series, we have to make sure it’ll converge otherwise the infinite series will grow without bound

Taylor Series Formula

Taylor's Theorem states that any function that is infinitely differentiable can be represented as a polynomial of the following form:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f^{(1)}(a)(x-a) + \frac{f^{(2)}}{2!}(x-a)^2 + \dots \qquad (1)$$

This is an exact representation of f(x) since it is an infinite series. To get an approximation, we can take only the first N terms; higher the value of N, the better the approximation.

We will not review the proof here. However, what if we take N=2 terms? This is what we get a linear equation for the tanget line. This tanget line is a decent approximation for f(x) as long as x is close to a. Adding more terms makes this approximation better over a bigger neighborhood of a.

Infinite Series and Convergence

We did not cover this in this class, but not all infinite series converge to a value. There are some conditions on the infinite series in order for it to converge is that each element in the series should be getting smaller and closer to 0.

This is why some Taylor series have a condition on which values of x are allowed.

Taylor Series Visualizations

- https://www.desmos.com/calculator/oiexhzavjp
- http://mathinsight.org/applet/taylor_polynomial
- Some animated gifs and explanations: https://www.quora.com/What-is-the-Taylor-series-exactly

Taylor Series Example from HW

Review e^x example; review why series converges

Gradients Review

$$\Delta f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

The Gradient of a function is simply a vector with n dimensions, where n is the number of input variables for f. The vector components are the partial derivatives of $f(x_1, x_2, \ldots x_n)$ WRT the input variables.

Gradient Descent

The basic idea of gradient descent is that we have some objective function that we want to maximize that has n parameters. To find the minimum, we start our search with a random point in the search space (of n dimensions).

You then calculate the gradient, which is a vector that will point in the direction away from the minimum. More precisely, the gradient points in the direction of the greatest rate of increase of the function, and its magnitude is the slope of the graph in that direction.

We use this information to take a ‘step’ in the opposite direction to get use closer to parameters that minimize the objective function.

Gradient Descent Pseudocode

- Start with a random point $\Theta = \Theta_0$
- Compute the gradient $\nabla J(\Theta_0)$
- Take a step away from the direction of the gradient using a learning rate α. That is, set $\Theta = \Theta_0 - \alpha \nabla J(\Theta_0)$.
- Repeat the above steps till there is no change in the objective function.

Gradient Descent Visualizations

- https://blog.biolab.si/2016/08/25/visualizing-gradient-descent/
- http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/

Gradient Descent and Linear Regression

- http://scipython.com/blog/visualizing-the-gradient-descent-method/
- https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/

For a nice video presentation on GD, check out this video. Even if you do not watch this video, you should subscribe to this channel, as he has many great tutorials on ML and Deep Learning concepts.

Stochastic and MiniBatch Gradient Descent

Notice that the gradient needs to loop over all data values in order to calculate the next update. This can be problematic for large datasets. Stochastic Gradient Descent solves this by essentially doing an update step per data point, which eases up on the computation.

MiniBatch Stochastic Gradient Descent is similar, but does each update step for m data points, where m << M, the total number of data points.