Name: _____

MAT 397— Fall 2020
Applied Problems:
Gradients & Direc. Deriv.

"Premature optimization is the root of all evil."
–Donald Knuth

Gradient Descent & Machine Learning

We have been promised intelligent AI for decades. So what is the holdup? For years, computer programs tried to program intelligence line-by-line—how ironic! This means for a given computer task, one would have to program every possibility that the computer could face. For example, if you were programming an AI which could respond to human asking the computer how it was, you would have to program all the possible ways to form that question: "How are you?", "How are you doing?", "How are you feeling?", "What's up?", etc. While much progress was possible with this restrictive approach to AI, clearly, a better approach was needed. It wasn't until the 1980s, and really the 1990s–2000s with more technology available, that the probable 'correct' approach was found—machine learning. Rather than tell the computer what to do in every situation, you would 'teach' the computer how to learn. Then you train the computer by applying this learning algorithm to datasets, and 'intelligence' emerges. Machine learning is a specific type of artificial intelligence, and the most powerful of machine learning techniques is called *deep learning*, which relies on *neural nets*.

Essentially, (supervised) machine learning works as follows: suppose you wanted to predict which people might default on their home loans. A professional, such as a banker, could look at the data for an individual and make an assessment of how likely a person is to default. How do we teach a computer to do the same? We gather a lot of relevant data on home loans: age, income level, credit rating, geographic location, number of dependents, etc, and whether the person defaulted on their home loan or not. We then feed this data to a computer, asking it for each person in the dataset whether given their associated data, whether it thinks they will default or not. Based on whether it was right or wrong, the algorithm alters how it makes predictions. We repeat the process many times, and in 'most' cases, the predictions improve. This is how computers are now better than humans at tasks like identifying images, and the games Chess, Go, Jeopardy, etc.

Notice deep learning hinges on the ability for the computer to systematically 'tweak' its guessing system at each stage. This is ultimately an optimization problem, where the computer seeks to find a 'best possible' weighting system of the variables that minimizes its chances of being incorrect. [This is a vast oversimplification, but carries the essence of what happens.] Of course, how does the computer know how to change the weightings at each stage? A common method is *gradient descent*.

Imagine the probability of being incorrect, L, for a computer prediction depends on two weights, say x, y. Then $L(x, y)$ is a surface in three dimensions. Given a set of weights, (x, y), we want to know how to change our weights, x, y, to minimize $L(x, y)$. Imagine standing on the surface $L(x, y)$ at the point (x, y). The gradient, ∇L, will give a direction we can travel in to increase $L(x, y)$ the quickest. Therefore, $-\nabla L(x, y)$ will give a direction we can move in to decrease $L(x, y)$ the fastest—and this is our goal! So we could take a 'step' in the direction of $-\nabla L(x, y)$. How far should we step? That's a difficult problem in these computer learning methods, but say we use step size dt. Then $-\nabla L(x, y)\ dt$ will tell us how much to change x and y; that is, the x and y components of $-\nabla L(x, y)\ dt$ tells us how much to change x and y, respectively. We then repeat the process again and again. Hopefully, this 'walks' us to a minimum value for $L(x, y)$. This process is called gradient descent. If you wanted to maximize $L(x, y)$, you would use this process with $\nabla L(x, y)$—gradient ascent.

Problem:

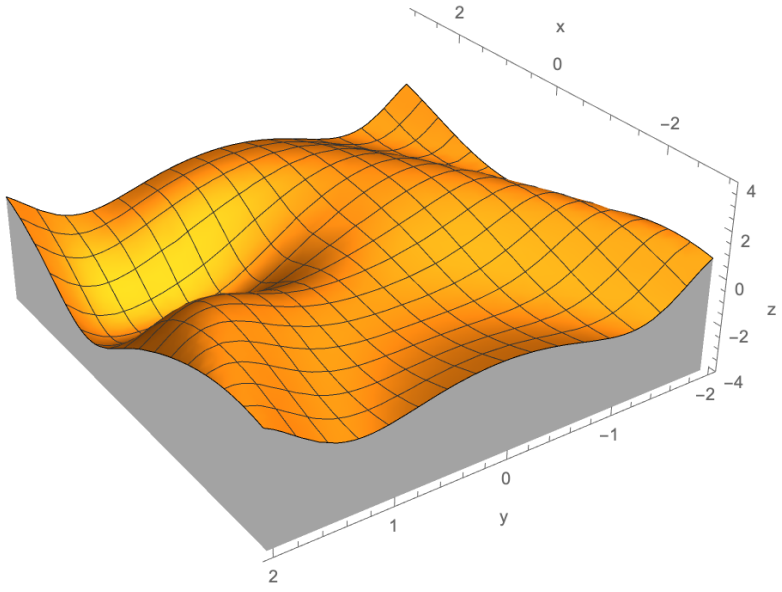
(a) Suppose you want to minimize $F(x, y)$. You decide to use a gradient descent method to find an approximate minimum value for $F(x, y)$. You start with a point $\mathbf{x}_0 = (x_0, y_0)$, and will use step size Δt. Explain why $\mathbf{x}_{n+1} = \mathbf{x}_n - \Delta t \nabla F(\mathbf{x}_n)$ is your gradient descent method.

(b) Is it true that $F(\mathbf{x}_{n+1}) \leq F(\mathbf{x}_n)$? Explain.

(c) There are many problems that can arise in gradient descent. Take the function $f(x, y) = x^2 + y^2$. Try to approximate a minimum value for $f(x, y)$ by using two steps of the gradient descent method using $\Delta t = 1$ and initial point $(x_0, y_0) = (2, 2)$. What goes wrong? What can you change about Δt to fix this?

(d) What can be problematic with your suggestion in (c)? [Hint: Take your suggestion in (c) to an extreme.]

(e) Consider again the problem that arose in (c). A possible solution might be to adjust your choice of $\mathbf{x}_0 = (x_0, y_0)$. Consider applying the gradient descent method to some starting point on the surface given in the figure below. By carefully considering what happens when choosing different initial points (x_0, y_0), explain another issue that can arise when using the gradient descent method.



(f) Let $f(x, y) = y^4 - 2xy^2 + x^3 - x + 3$. This function has minima at $(1, 1)$ and $(1, -1)$. Moreover, $f(1, 1) = f(1, -1) = 2$. Using $\Delta t = 0.1$ and start point $(x_0, y_0) = (1.5, 2.2)$, perform five steps of the gradient descent method. How close to the minima point and minimum value does this procedure produce?

(g) For the function given in (f), suppose you use the gradient descent method starting at $(1, 0)$. Because the point $(1, 0)$ is 'midway' between the minima $(1, 1)$ and $(1, -1)$, will the gradient descent method simply 'stay' at the point $(1, 0)$—unable to 'decide' between $(1, 1)$ and $(1, -1)$? Explain.

(h) Is $\nabla F(\mathbf{x_n}) = \mathbf{0}$ possible in the gradient descent method, either at the initial or some subsequent point? What would happen to the method, assuming it could happen?