Gradient Descent -- Downhill to a minimum

We are going to talk about the gradient descent to get the central algorithm of neural net deep learning, machine learning, and optimization in general. So we are trying to minimize a function. And that's the way we do it if there are many variable, too many to take second derivatives then we settle for first derivatives of the function.

Let's see an example. Suppose we have a function $f = \frac{1}{2}x^TSx = \frac{1}{2}(x^2 + by^2)$. The function is a pure quadratic, two unknowns. So every quadratic we can write in terms of symmetric matrix S. And in this case

$$S = \begin{bmatrix} 1 & 0 \\ 0 & b \end{bmatrix}$$

The conditional number of S, which we will see, is all important in the question of the speed of convergence is the ratio of the largest to the smallest, in this case is $1/b$.

When the matrix is symmetric, that conditional number is $\lambda_{max}/\lambda_{min}$. If we had an unsymmetric matrix, we would probably use $\sigma_{max}/\sigma_{min}$.

We are going to see something neat is that we can actually take the steps of steepest descent, write down what each step gives us, and see how quickly they converge to the answer. And what is the answer? We haven't put in any linear term here. So we just have a bowl sitting on the origin. So of course the minimum point is $x = 0, y = 0$. So the question will be how quickly do we get to that point? And you will say pretty small example, not typical. But the terrific thing is that we see everything for this example. We can see the actual steps of steepest descent. We can see how quickly they converge to the x^* -- the answer. And we can begin to think what to do if it's too slow.

Let's see some general thoughts about gradients, Hessians. Let's see an example, say $f(x, y) = 2x + 5y$.

$$\nabla f = [2, 5]^T$$

And the Hessian

$$H = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

argmin(f) means the place where $f = f_{min}$.

Let's see a remarkable convex function. And the notes tell what's the gradient of this function. They don't actually go as far as Hessian.

$$f(X) = -\log\left(det(X)\right)$$

X is a matrix with n^2 variables x_{ij}. This function turns out to be a convex function. The gradient of this function is also amazing.

$$\nabla f = -(\text{entries of } X^{-1})$$

$X^{-1} = \frac{(det(X))'}{det(X)}$.

Gradient descent:

$$x_{k+1} = x_k - s_k \nabla f(x)$$

The only thing left that requires us to input some decision making is a step size, the learning rate. We can take it as constant. If we take too big a learning rate, the thing will oscillate all over the place and it's a disaster. If we take too small a learning rate, too small steps, it takes too long. So the problem if to get it just right. And one way that we should say get it right would be to think of optimize.

The exact line search choose s_k to make the function $f(x_{k+1})$ a minimum on the search direction. <u>The condition number controls the speed.</u> And it is the sort of key point on gradient descent.

Backtracking line search would be take a fixed s like 1. And then be prepared to come backwards. $s_0 \to \frac{1}{2}s_0 \to \cdots$ until you are satisfied with that step.

Here is the key question : How much does the exact line search reduce the function? The reduction involves the condition number.