Gradient Descent

① It's relatively straightforward to find the optimum of certain loss functions using standard calculus techniques, e.g.

$$L(\theta) = (20-5\theta)^2 + (41-12\theta)^2$$

$$\Rightarrow \frac{dL(\theta)}{d\theta} = 338\theta - 1184$$

$$\frac{dL(\theta)}{d\theta} = 0 \quad \Rightarrow \quad \theta = \frac{1184}{338} \approx 3.5$$

Thus:

$$\underset{\theta}{\operatorname{argmin}}\ L(\theta) \approx 3.5$$

---

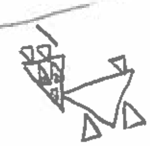
② But often (almost always in this course), the loss function is more complicated. What if it were

$$L(\theta) = (\sin 2\theta)(\log \theta^2)$$

Well, the derivative is doable:

Use the Product Rule! $\frac{d(u \cdot v)}{d\theta} = \frac{du}{d\theta} \cdot v + u \cdot \frac{dv}{d\theta}$

$$\frac{dL(\theta)}{d\theta} = 2\cos 2\theta \log \theta^2 + (\sin 2\theta) \cdot \frac{2\theta}{\theta^2}$$



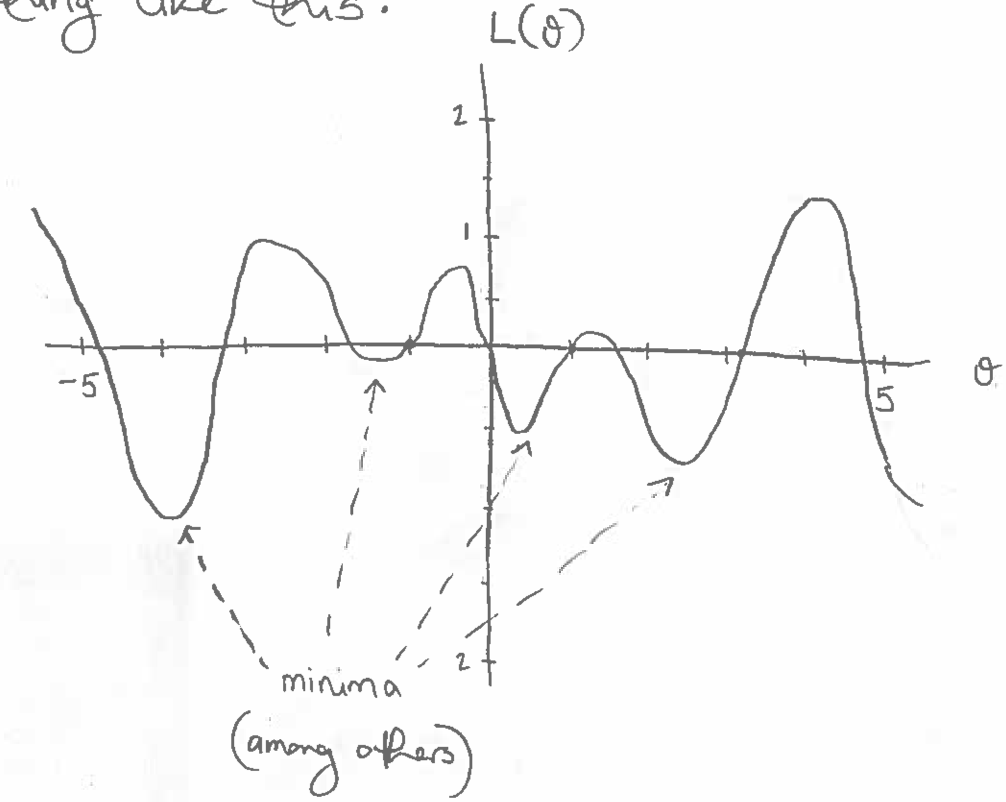
But then you have to set it to zero and solve for θ:

$$2\cos 2\theta \log \theta^2 + (\sin 2\theta) \cdot \frac{2\theta}{\theta^2} = 0$$

$$\Rightarrow \quad \theta = ???$$

③ So, what to do?

If we were to graph the loss function, we'd get something like this:

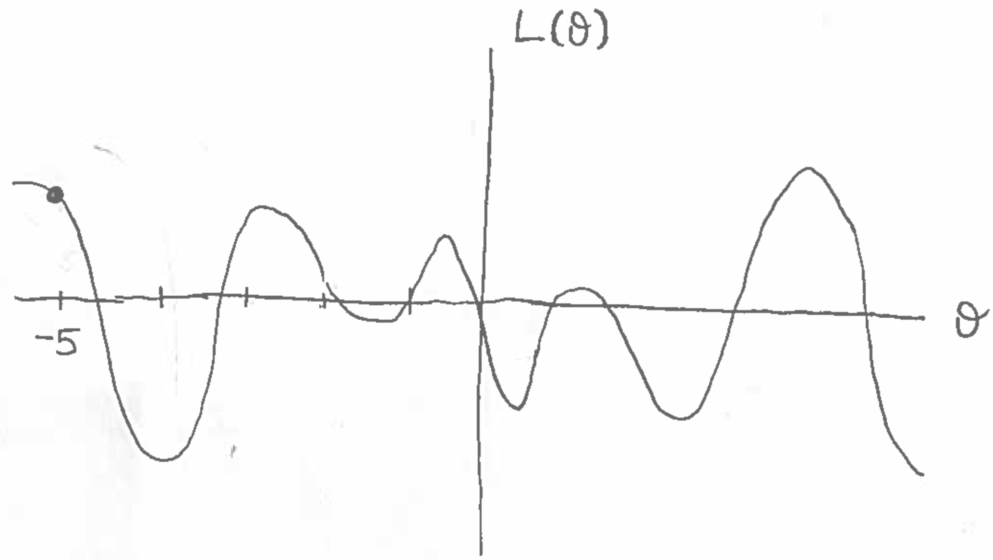


④ Now maybe we're ok with just finding a small loss, rather than insisting on the absolutely smallest loss. So $\theta = -4$ would be great, but $\theta = 2.5$ wouldn't be the end of the world.

Gradient Descent

(5) The strategy we'll mostly use is a simple one called <u>gradient descent</u>.

We start by guessing (probably arbitrarily) some value for θ, like $\theta = -5$:



We know the derivative of $L(\theta)$, so we can compute

$$\frac{dL(-5)}{d\theta} = 2\cos(-10)\log(25) + \sin(-10)\cdot\frac{(-10)}{25}$$

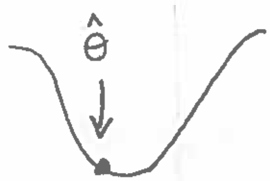
$$\approx -2.56$$

(6) Since the derivative at $\theta = -5$ is <u>negative</u>, that means $L(\theta)$ is <u>decreasing</u> as we <u>increase</u> θ from -5. We're trying to minimize $L(\theta)$, so we want to <u>increase</u> θ.

7) But by how much? Gradient descent uses the following intuition:

> the steeper the function at our guess $\hat{\theta}$, the more we should increase (or decrease) our guess

It's easier to rationalize this by thinking about when the derivative has a very small magnitude:



Suppose that's our guess $\hat{\theta}$. The guess is very close to the optimum, so the derivative at $\hat{\theta}$ is pretty close to zero (e.g. say it's -0.001). The magnitude of the derivative warns us that we're getting close to the optimum, so we don't want to increase it by much.

By contrast, a derivative of high magnitude gives us more freedom to jump ahead, heedless, into the abyss.

(8) So suppose we increase θ by $-\alpha \cdot \frac{dL}{d\theta}(-5)$, where $\alpha = 0.5$.

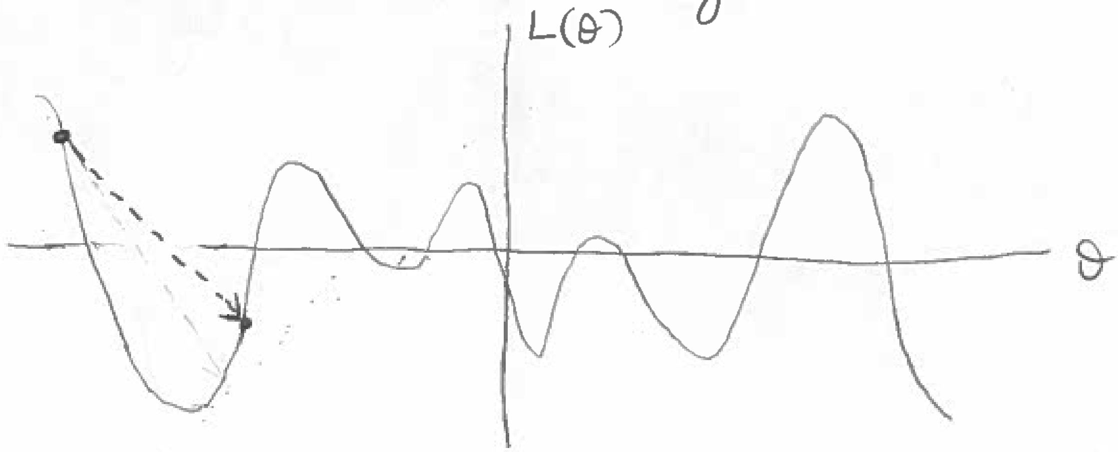
This constant α, called the <u>learning rate</u>, is here being chosen arbitrarily, but let's see what happens.

$$\theta' = \theta - \alpha \cdot \frac{dL}{d\theta}(\theta)$$

$$= (-5) - 0.5 \cdot \frac{dL}{d\theta}(-5)$$

$$\approx -3.72$$

(9) Our new guess for θ is -3.72. We see we've descended further into the valley:



So the slope has become gentler:

$$\frac{dL}{da}(-3.72) = 2\cos(2 \cdot -3.72)\log(-3.72^2) + \sin(2 \cdot -3.72) \cdot \frac{(-3.72)}{(-3.72)^2}$$

$$\approx 1.16$$

Gradient Descent

(10) Since the derivative at $\theta = -3.72$ is positive, that means $L(\theta)$ is increasing as we increase θ from -3.72. We're trying to minimize $L(\theta)$, so we want to decrease θ.

In other words, we want to compute:

$$\theta' = \theta - \alpha \cdot \frac{dL}{d\theta}(\theta)$$

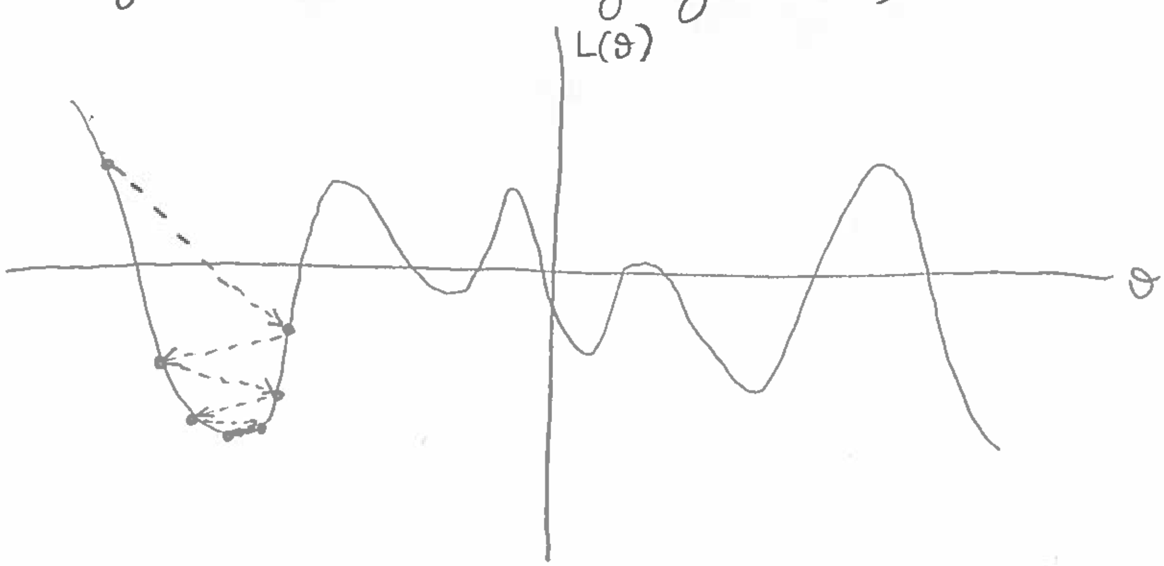
which is conveniently the exact same thing we did when the derivative was negative.

$$\theta' = (-3.72) - 0.5 \frac{dL}{d\theta}(-3.72)$$

$$= -3.72 - 0.58$$

$$= -4.3$$

(11) An algorithm has started to emerge. Keep doing this until you barely move at all (or you get tired):

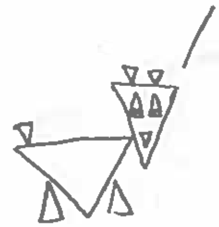


GRADIENT DESCENT

(12) More rigourously:

GRADIENT DESCENT(loss function L, learning rate $\alpha \in \mathbb{R}$):
initialize $\theta^{(0)}$ to some real number; $t \leftarrow 0$
repeat until happy:
- let update $\sigma^{(t)} \leftarrow -\alpha \cdot \frac{dL}{d\theta}(\theta^{(t)})$
- let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
- let $t \leftarrow t+1$

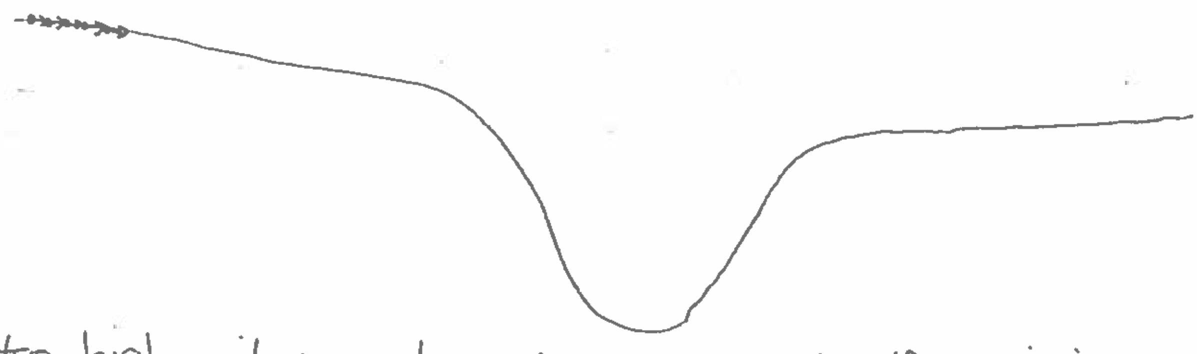
usually until $|\theta^{(t+1)} - \theta^{(t)}| < \epsilon$ for some small $\epsilon > 0$, or until $t > T$ for some large integer T



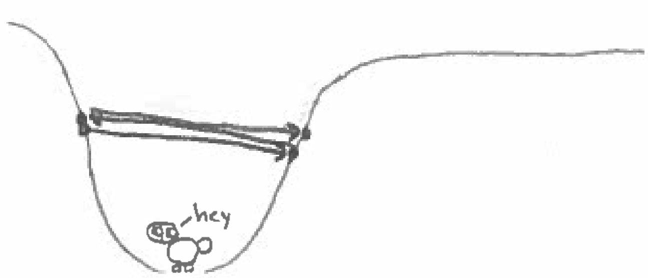
(13) The main issue with gradient descent is:

how do you set the learning rate α?

If it's too low, it may take forever to reach a minimum:



If it's too high, it may keep jumping past the minimum:



(14) One strategy for dealing with this conundrum is to have the learning rate adapt, depending on what's happened so far during the gradient descent.

For instance, we could start with an aggressive (high) learning rate, and then gradually make it more conservative (lower it) as time goes on. Something like:

GDWithTimeBasedDecay(loss L, learning rate α, decay rate β):
initialize $\theta^{(0)} \in \mathbb{R}$; $t \leftarrow 0$; $\alpha_0 \leftarrow \alpha$
repeat until happy:
- let learning rate $\alpha^{(t)} \leftarrow \dfrac{\alpha}{1+\beta \cdot t}$
- let update $\sigma^{(t)} \leftarrow -\alpha^{(t)} \cdot \dfrac{dL}{d\theta}(\theta^{(t)})$
- let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
- let $t \leftarrow t+1$

Thus, the algorithm first jumps around, trying to find a valley, and then becomes more conservative, in order to stay in that valley and converge to its minimum.

But this is still fairly arbitrary, and doesn't take into account the behavior of the algorithm when determining the step size.

GRADIENT DESCENT

(15) Maybe it would be useful to try the following. Start out aggressively, and only become more conservative once you begin to see significant vertical progress, e.g.

One way to do this is to set the rate $\alpha^{(t)}$ at time t to be inversely proportional to the magnitudes of the previous derivatives, e.g.

$$\alpha^{(t)} \leftarrow \frac{\alpha}{\delta + \sqrt{\sum_{t'=0}^{t} \left(\frac{dL}{d\theta}\left(\theta^{(t')}\right)\right)^2}}$$

a tiny constant to avoid division by zero

as we experience large derivatives (either positive or negative), this gets increasingly large, which decreases the learning rate.

GRADIENT DESCENT

(16) This update gives us the AdaGrad (Adaptive Gradient) algorithm:

ADAGRAD (loss L, init learning rate α, tiny delta $\delta > 0$):
initialize $\theta^{(0)} \in \mathbb{R}$; $t \leftarrow 0$
repeat until happy:
- let learning rate $\alpha^{(t)} \leftarrow \dfrac{\alpha}{\delta + \sqrt{\sum_{t'=0}^{t} \left(\frac{dL}{d\theta}(\theta^{(t')})\right)^2}}$
- let update $\sigma^{(t)} \leftarrow -\alpha^{(t)} \cdot \frac{dL}{d\theta}(\theta^{(t)})$
- let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
- let $t \leftarrow t+1$

(17) One potential downside to setting the learning rate this way:

$$\alpha^{(t)} \leftarrow \frac{\alpha}{\delta + \sqrt{\sum_{t'=0}^{t} \left(\frac{dL}{d\theta}(\theta^{(t')})\right)^2}}$$

← - - - - this is a constant

← - - - - this grows without bound

So eventually AdaGrad will crawl to a stop, possibly before you want it to.

GRADIENT DESCENT

(18) One could imagine taking the <u>average</u> of previous derivatives instead of the <u>sum</u>, like:

$$\alpha^{(t)} \leftarrow \frac{\alpha}{\delta + \sqrt{\dfrac{\sum_{t'=0}^{t}\left(\frac{dL}{d\vartheta}(\vartheta^{(t')})\right)^2}{t}}}$$

But there's no fancy name for this variant of gradient descent, so presumably it isn't that effective.

(19) A variant that <u>does</u> have a name uses the <u>decaying average</u> of previous gradients. Given a series $q^{(t)}$ of real numbers, you can compute a decaying average with the recurrence:

$$m^{(t)} = \beta m^{(t-1)} + (1-\beta)q^{(t)}$$

where $0 \leq \beta \leq 1$.

(20) Depending on β, the decaying average will "forget" older values of $q^{(t)}$ more or less aggressively:

t	$q^{(t)}$	$m^{(t)}(\beta=0.5)$	$m^{(t)}(\beta=0.9)$
1	3	1.5	0.3
2	5	3.25	0.77
3	6	4.63	1.29
4	2	3.31	1.36
5	4	3.66	1.63
6	8	5.83	2.26

GRADIENT DESCENT

21) The intuition behind using the decaying average of previous gradients to set the learning rate is: if our progress has stagnated (recent derivatives have all been close to zero), then either:

(a) we're close to a minimum:

(in which case there's no harm increasing the learning rate, since the gradients will remain close to zero regardless)

(b) we're in a long flat region

(in which case we'd like to bump up the learning rate so we can continue to make meaningful forward progress)

GRADIENT DESCENT

(22) This variant is called RmsProp.

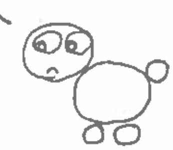
RmsProp (loss L, init learning rate α, decay rate β, tiny delta δ):

initialize $\theta_0 \in \mathbb{R}$; $t \leftarrow 0$; $m^{(-1)} \leftarrow 0$

repeat until happy:

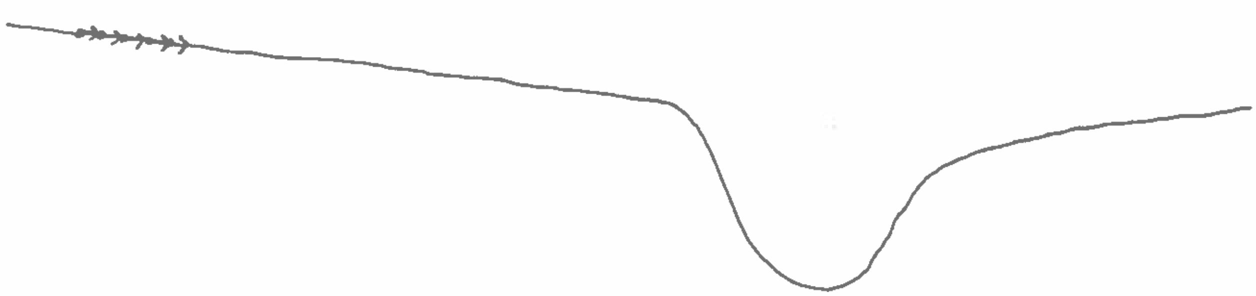
- let $m^{(t)} \leftarrow \beta m^{(t-1)} + (1-\beta)\left(\frac{dL}{d\theta}(\theta^{(t)})\right)^2$
- let learning rate $\alpha^{(t)} \leftarrow \dfrac{\alpha}{\sqrt{\delta + m^{(t)}}}$
- let update $\sigma^{(t)} \leftarrow -\alpha^{(t)} \cdot \frac{dL}{d\theta}(\theta^{(t)})$
- let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
- $t \leftarrow t+1$

decaying average of the squared derivatives



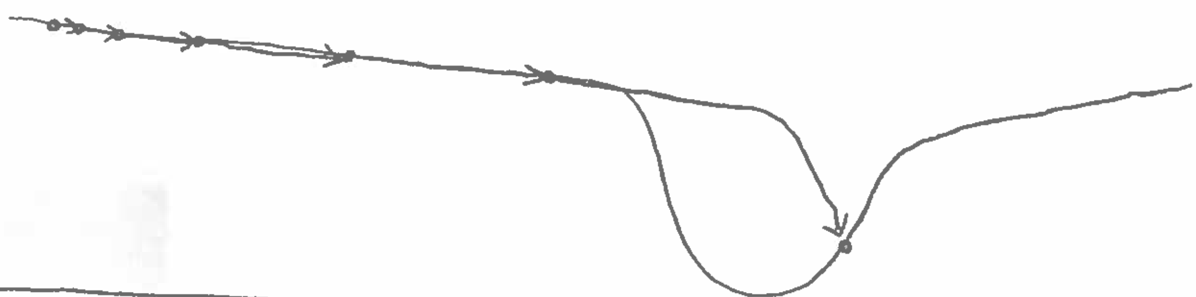
for some reason, the small constant δ is now inside the square root

(23) Still, even if the learning rate stays high, it can still take a long time to inch our way along a mostly flat surface, since the gradient is low.

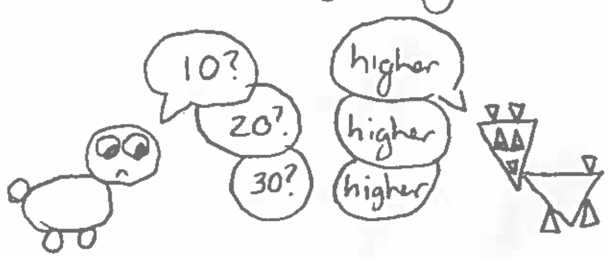


Gradient Descent

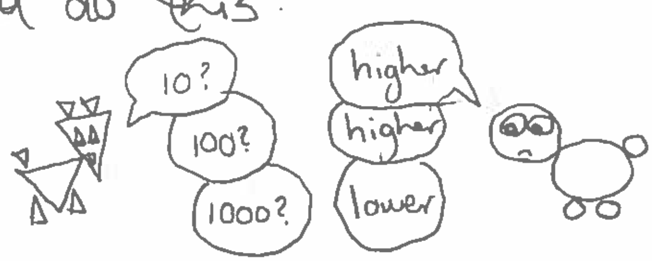
(24) Here, we can take inspiration from physics. If we were to place a ball on the slightly sloped surface, it wouldn't move along the surface at a fixed rate. Rather, it would accelerate, as long as the surface continued to slope downward. It'd be nice to duplicate this effect in gradient descent, something like:



(25) Think about if someone were to ask you to guess a number (without giving any bounds). You can do this:



Or you could do this:



Sure, you'll overshoot, but you can always double back (but now with knowledge of the bounds).

(26) A common way to achieve this effect in gradient descent is called <u>momentum</u>.

GDWITHMOMENTUM(loss L, learning rate α, momentum rate μ):
initialize $\theta^{(0)} \in \mathbb{R}$; $t \leftarrow 0$; $\sigma^{(-1)} \leftarrow 0$
repeat until happy:
- let step size $\sigma^{(t)} \leftarrow \mu\sigma^{(t-1)} - \alpha^{(t)} \frac{dL}{d\theta}(\theta^{(t)})$
- let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
- let $t \leftarrow t+1$

This does a weighted average of the previous step and the current derivative, in order to compute the next step.

So if you keep heading in the same direction and encountering the same slope, e.g. $\frac{dL}{d\theta}(\theta^{(0)}) = \frac{dL}{d\theta}(\theta^{(1)}) = \frac{dL}{d\theta}(\theta^{(2)}) = \frac{dL}{d\theta}(\theta^{(3)}) = -0.1$,

then you start to accelerate. For example, if $\mu = 0.5$:

$$\sigma^{(0)} = 0.1\alpha$$
$$\sigma^{(1)} = \mu\sigma^{(0)} + 0.1\alpha = 0.15\alpha$$
$$\sigma^{(2)} = \mu\sigma^{(1)} + 0.1\alpha = 0.175\alpha$$
$$\sigma^{(3)} = \mu\sigma^{(2)} + 0.1\alpha = 0.1875\alpha$$

Note that the "momentum rate" μ controls how aggressive the acceleration is (higher = more aggressive)

Gradient Descent

(27) All these variants of gradient descent have the same basic template:

Generic Gradient Descent

- initialize first guess $\theta^{(0)}$; $t \leftarrow 0$; some other values
- repeat until happy:
  - set learning rate $\alpha^{(t)}$
  - compute update $\sigma^{(t)}$
  - let next guess $\theta^{(t+1)} \leftarrow \theta^{(t)} + \sigma^{(t)}$
  - let $t \leftarrow t+1$

There are a couple variants on the "compute update" line:

"just go with the gradient"	"add momentum"
$\sigma^{(t)} \leftarrow -\alpha^{(t)} \frac{dL}{d\theta}(\theta^{(t)})$	$\sigma^{(t)} \leftarrow \mu\sigma^{(t-1)} - \alpha^{(t)} \frac{dL}{d\theta}(\theta^{(t)})$
[GD, GD with Time Decay, AdaGrad, RmsProp]	[GD with Momentum]

There are several variants on the "set learning rate" line:

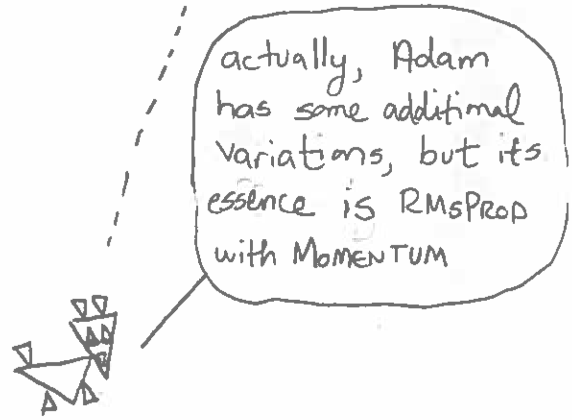
"keep it constant"	"time-based decay"
$\alpha^{(t)} \leftarrow \alpha$	$\alpha^{(t)} \leftarrow \frac{\alpha}{1+\beta\cdot t}$
[GD, GD with Momentum]	[GD with Time Decay]
"inversely proportional to the sum of previous squared gradients"	"inversely proportional to the decaying average of previous squared gradients"
$\alpha^{(t)} \leftarrow \frac{\alpha}{\delta + \sqrt{\sum_{t'=0}^{t}\left(\frac{dL}{d\theta}(\theta^{(t')})\right)^2}}$	$m^{(t)} \leftarrow \beta m^{(t-1)} + (1-\beta)\left(\frac{dL}{d\theta}(\theta^{(t)})\right)^2$ $\alpha^{(t)} \leftarrow \frac{\alpha}{\sqrt{\delta + m^{(t)}}}$
[AdaGrad]	[Rms Prop]

GRADIENT DESCENT

(28) One could imagine "mixing-and-matching" these variants to synthesize new gradient descent methods.

For instance, we could use the "add momentum" variant of "set step size" with the "inversely proportional to the decaying average of previous squared gradients" variant of "set learning rate". (i.e. hybridizing GDwithMOMENTUM and RMSPROP).

This variant is a commonly used one called Adam (short for <u>Ada</u>ptive <u>M</u>oments).



(29) While it's useful to understand these variants (particularly to understand the terminology in research papers), it's also important to realize that none of these methods are really "better" than the others. On some loss functions, RMSPROP may work best. On others, vanilla GD might be your best option.

GRADIENT DESCENT

30) Here's what the textbook has to say on the subject:

"At this point, a natural question is: which algorithm should one choose? Unfortunately, there is currently no consensus on this point."

"The choice of which algorithm to use, at this point, seems to depend largely on the user's familiarity with the algorithm."

(Goodfellow, p.302)