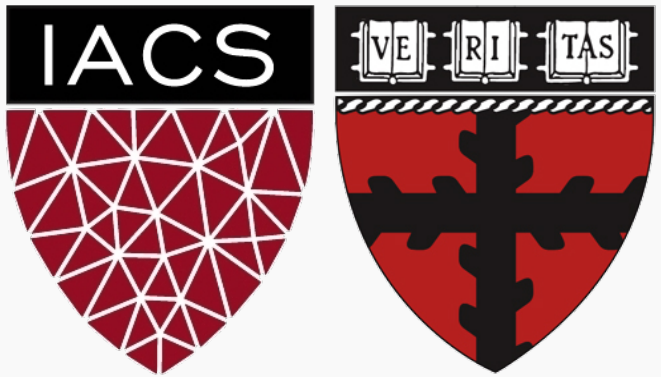
Advanced Section #6:
Convexity, Subgradients, Stochastic Gradient Descent

Cedric Flamant and Pavlos Protopapas

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader, and Chris Tanner



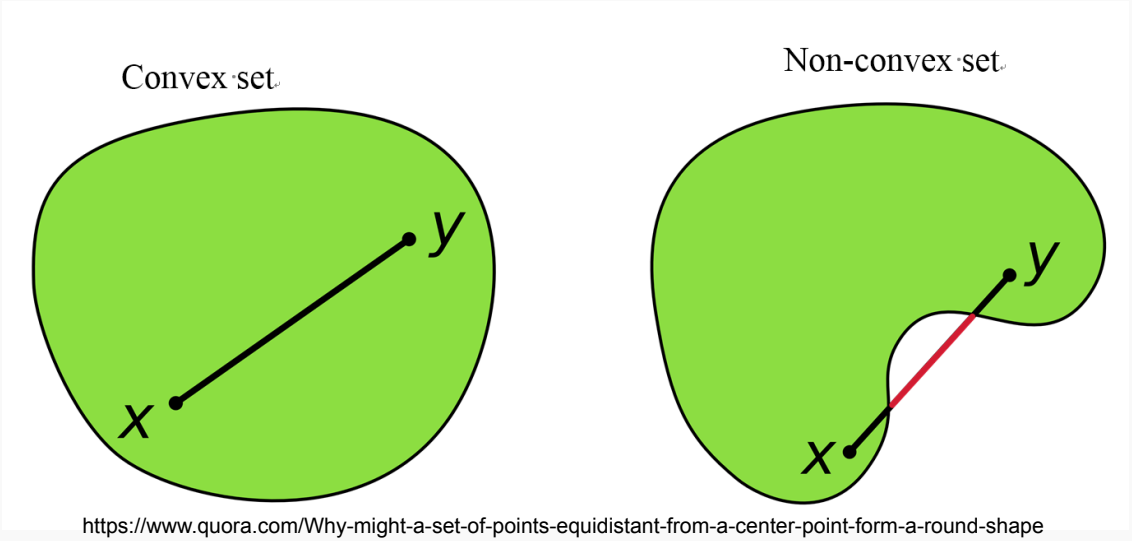
Outline

1. Introduction:
    a. Convex sets and convex functions

2. Stochastic Gradient Descent
    a. Foundation
    b. Subgradients
    c. Lipschitz continuity
    d. Convergence of Gradient Descent

3. Gradient Descent Algorithms

Convexity

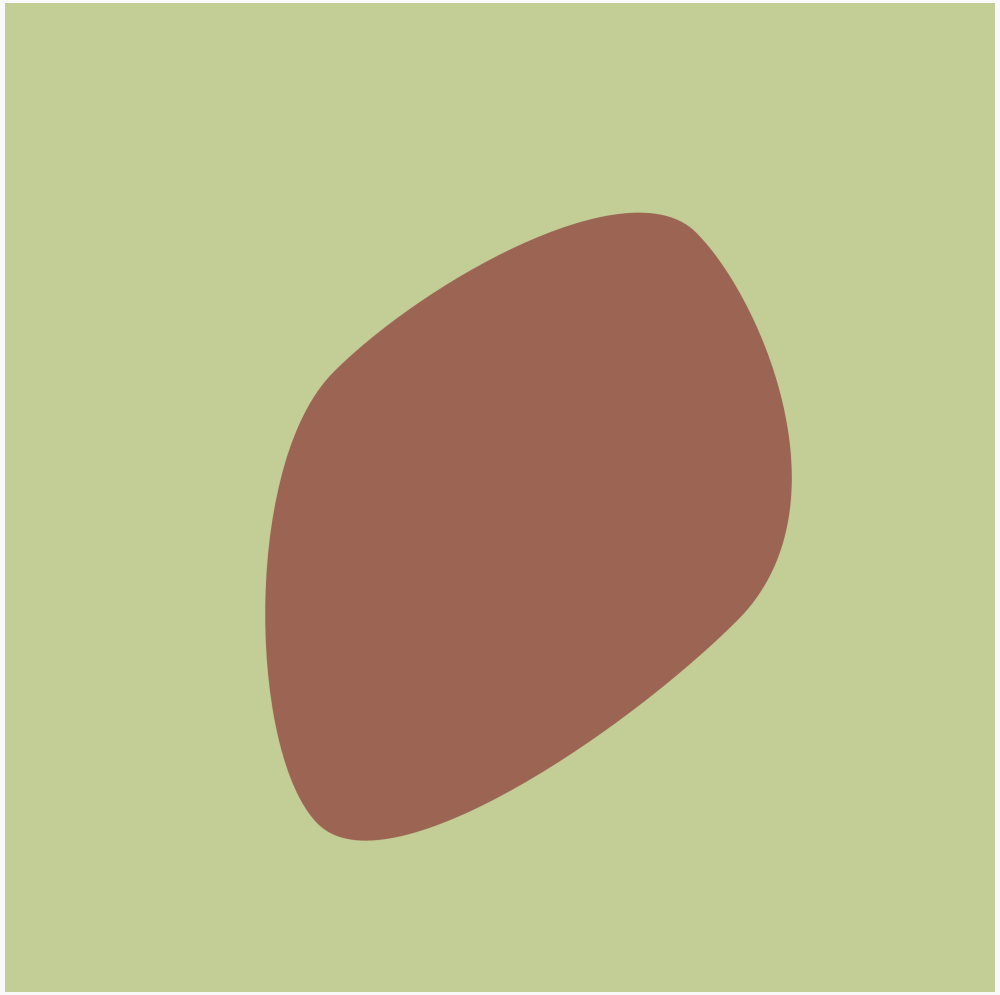


https://www.quora.com/Why-might-a-set-of-points-equidistant-from-a-center-point-form-a-round-shape

A convex set does not have any dents on its boundary, and contains no holes.

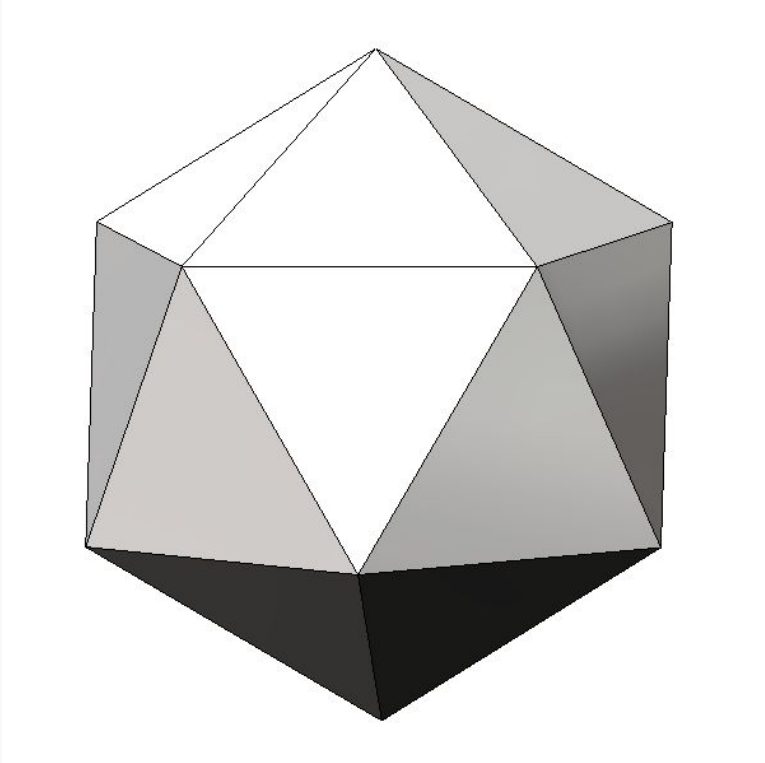
Mathematically, given a convex set, the line between any two points in the set is also contained in the set.

Is This Convex?



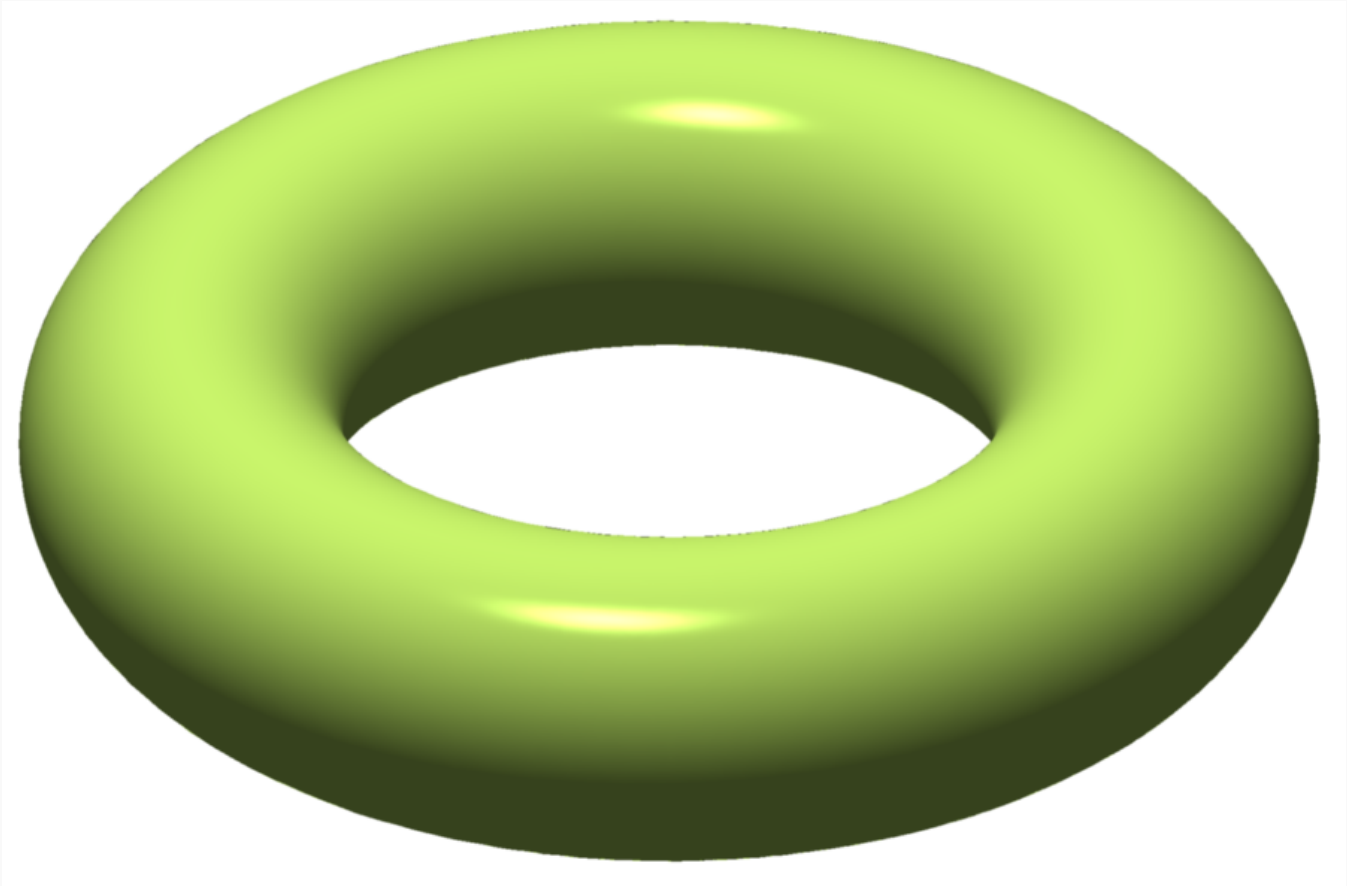
https://ghehehe.nl/the-daily-blob/

Is This Convex?



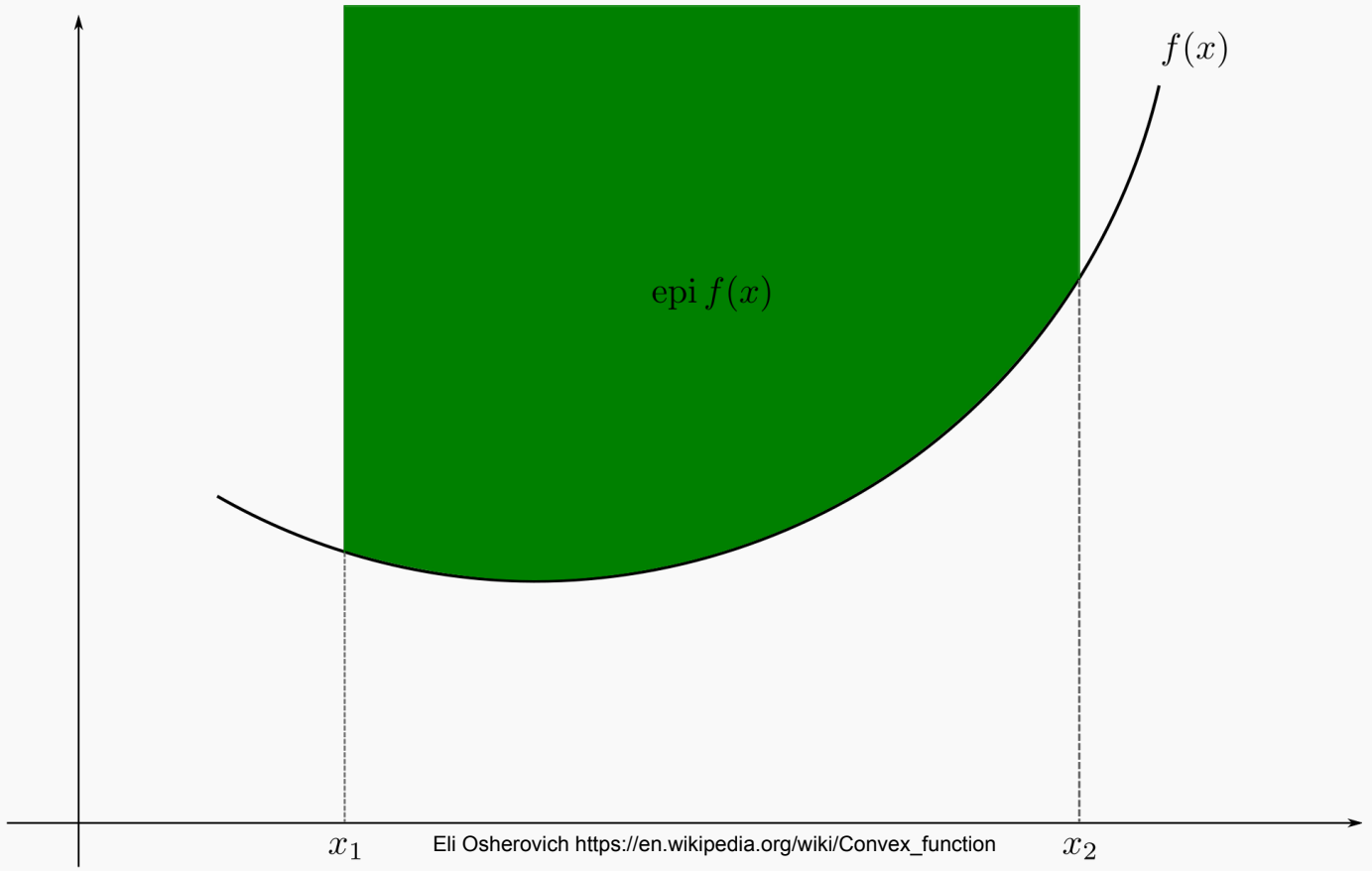
https://radiganengineering.com/2016/04/platonic-solids-in-solidworks/

Is This Convex?



https://en.wikipedia.org/wiki/Solid_torus

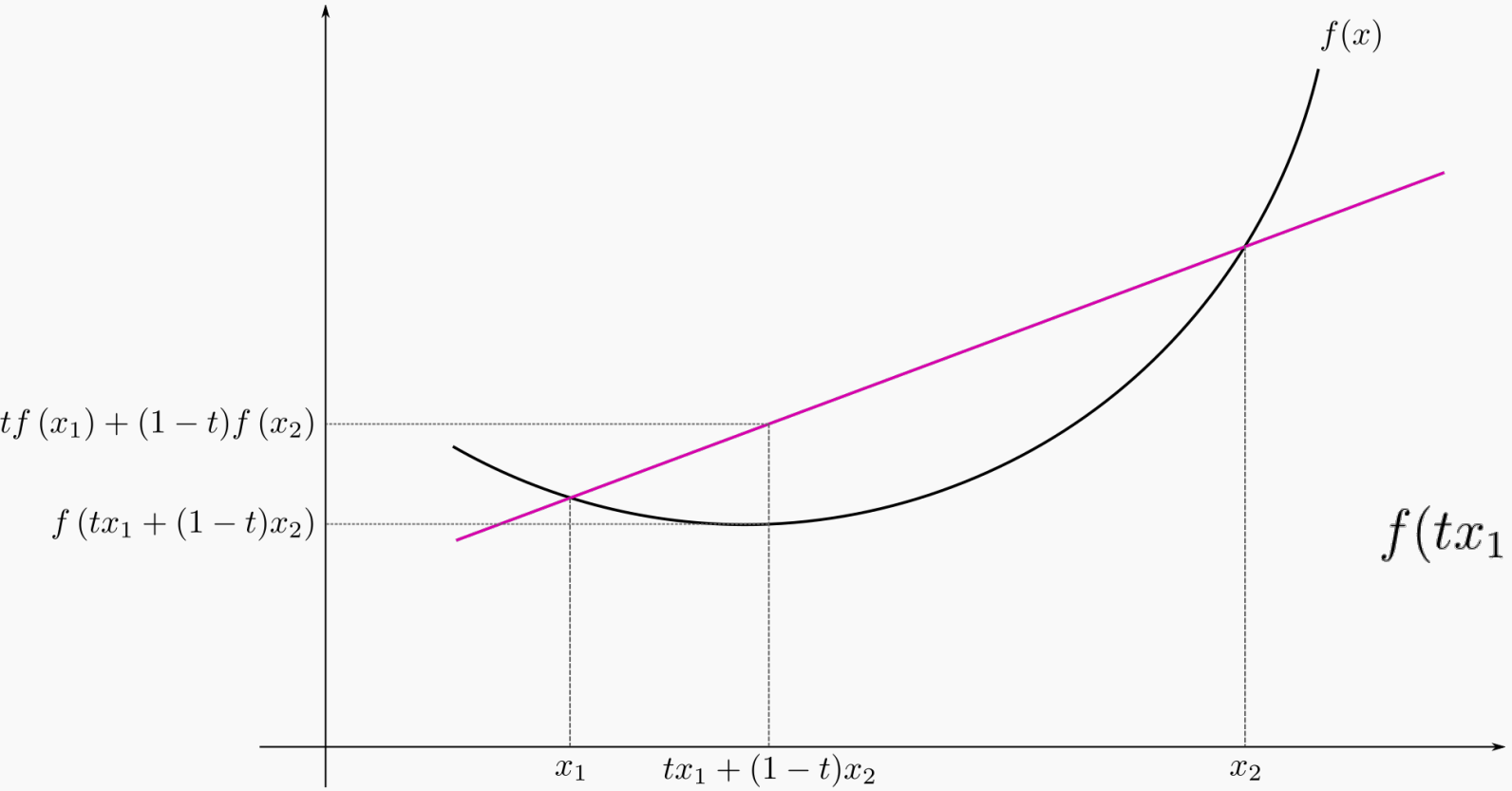
Convex Functions



Eli Osherovich https://en.wikipedia.org/wiki/Convex_function

- Convex functions "curve upward"
- The area above a convex function (the epigraph) is a convex set
- Convex functions must have a domain that is a convex set (by definition)
  - In this case, the domain is $[x_1, x_2]$, which is convex.

Convex Functions



Eli Osherovich https://en.wikipedia.org/wiki/Convex_function

Definition of Convex Function:

For a convex set X

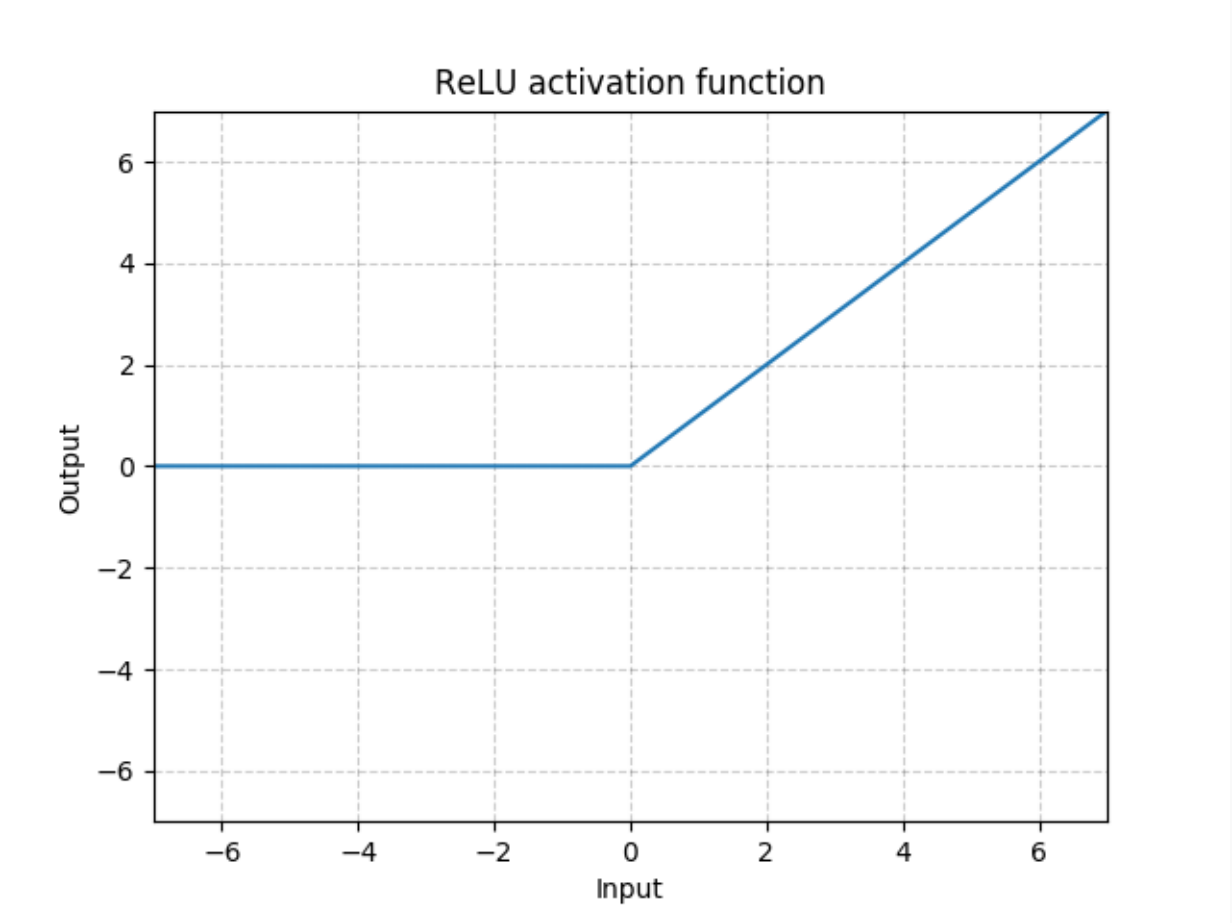
Function $f : X \to \mathbb{R}$ is convex if

$$\forall x_1, x_2 \in X,\ \forall t \in [0,1]$$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

i.e. the line connecting two points on the curve of the function is not below the function

Is This a Convex Function?

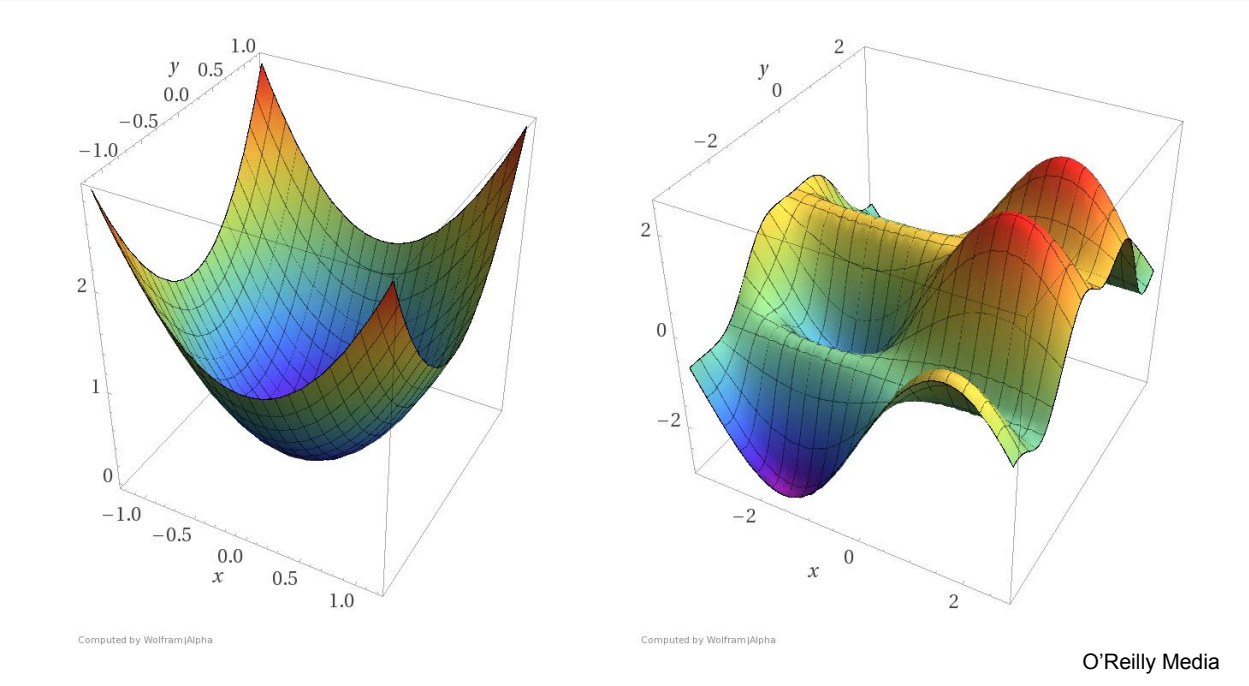


https://pytorch.org/docs/stable/nn.html

Why do we care about convex functions?

Convex Optimization:

- Every local minimum is a global minimum
- Gradient Descent is guaranteed to find a global minimum (with appropriate step size)
- Heavily relied on in proofs in machine learning papers



O'Reilly Media

Gradient Descent

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$$

We move in the *opposite* direction of the gradient, with a step size (*learning rate*) of η, If $f(\mathbf{w})$ is a loss, then with every step we try to decrease it.

$f(\mathbf{w})$ can be very complex, like a neural network.

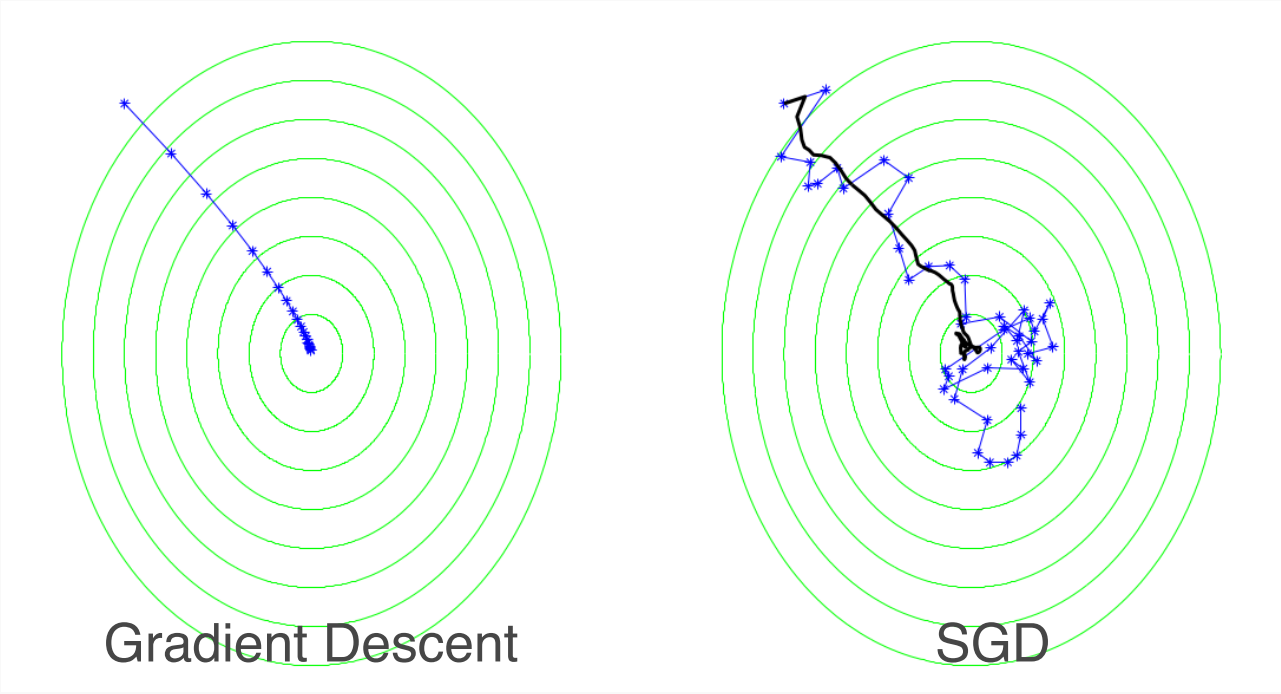
Because the gradient of a neural network is easily computable through backpropagation, it's not overstating the matter to say deep learning was built on gradient descent

Stochastic Gradient Descent

In SGD we only require that the *expected value* at each iteration will equal the gradient direction

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$$

$$\mathbb{E}[\mathbf{v}_t | \mathbf{w}^{(t)}] = \nabla f(\mathbf{w}^{(t)})$$



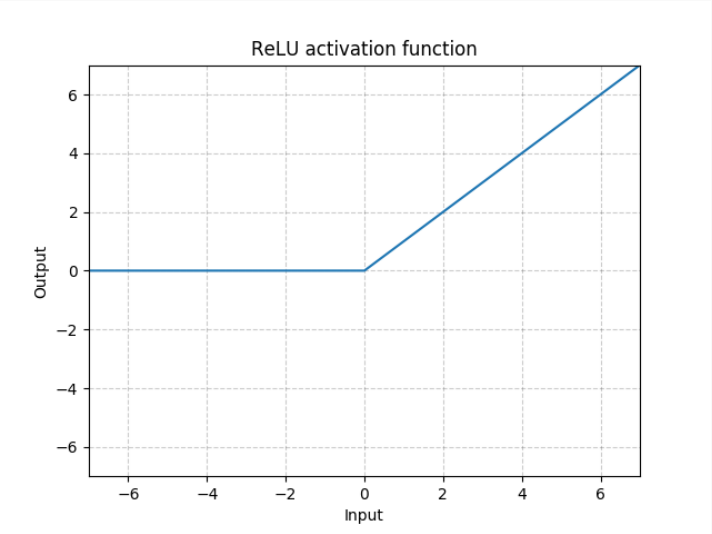
Understanding Machine Learning, Shalev-Swartz, Ben-David

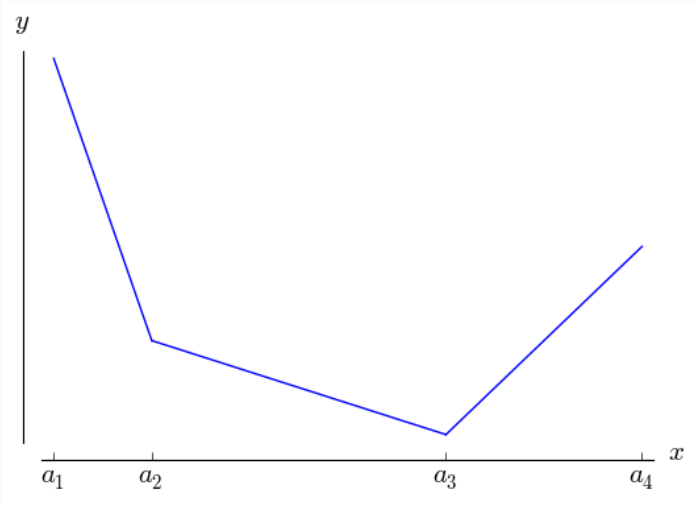
In black we have Ruppert-Polyak averaging, a well-known way to minimize asymptotic variance of SGD:

$$\bar{\mathbf{w}}^{(t)} = \frac{1}{t} \sum_{k=1}^{t} \mathbf{w}^{(k)}$$

Subdifferentiation and Subgradients

We also want to apply gradient descent to some functions that aren't differentiable:





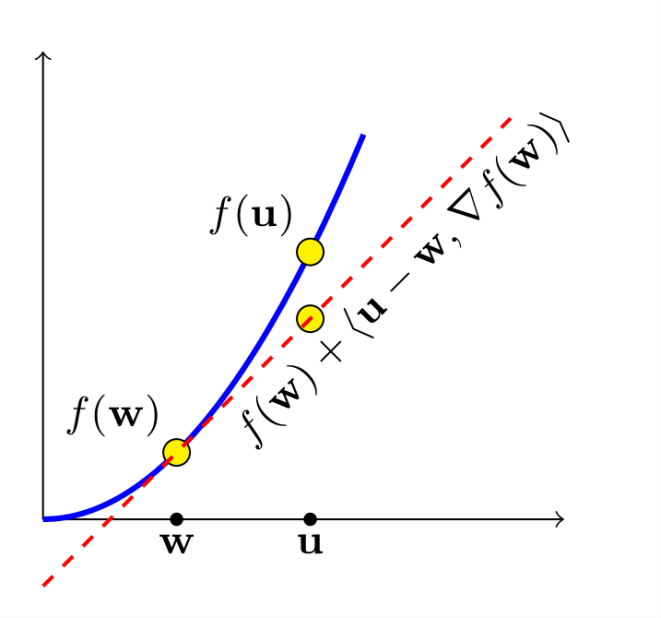
It would be nice if we could have a generalization of the derivative to help with problematic points.

Subdifferentiation and Subgradients

For a convex function $f(\mathbf{w})$ the following can be proven:

$$\forall \mathbf{u}, \quad f(\mathbf{u}) \geq f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w}) \rangle$$

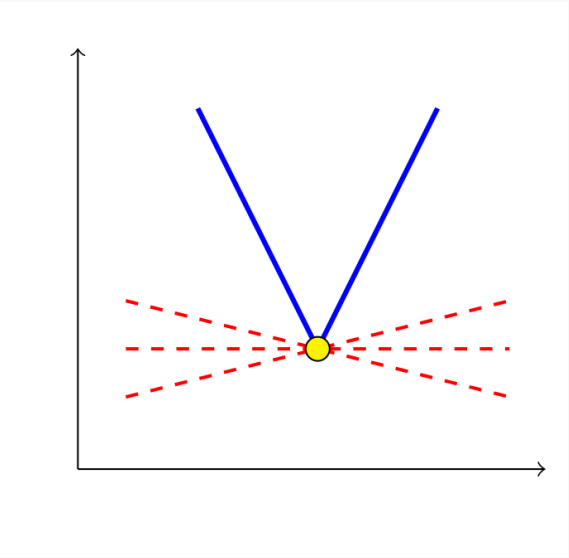
This is basically saying that a convex function is everywhere above any of its tangent lines.



We use this to generalize the notion of a gradient:

$$\forall \mathbf{u} \in S, \quad f(\mathbf{u}) \geq f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \mathbf{v}) \rangle$$

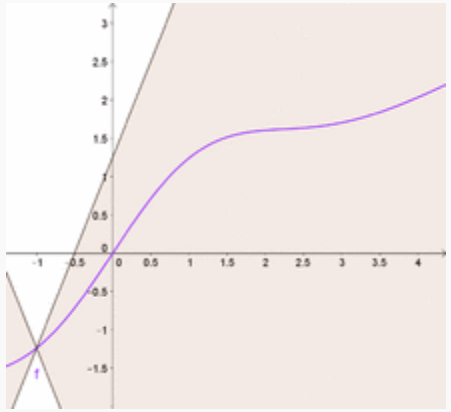
Any vector $\mathbf{v}$ satisfying this inequality is called a *subgradient* of the function f at $\mathbf{w}$, and the set of these subgradients is called the *differential set* and denoted $\partial f(\mathbf{w})$.



This allows us to perform *stochastic subgradient descent.*

Lipschitz Continuity

Another important term in analyzing convergence in ML problems is Lipschitz continuity. This is a subset of the set of continuous functions that is one step weaker than continuous differentiability

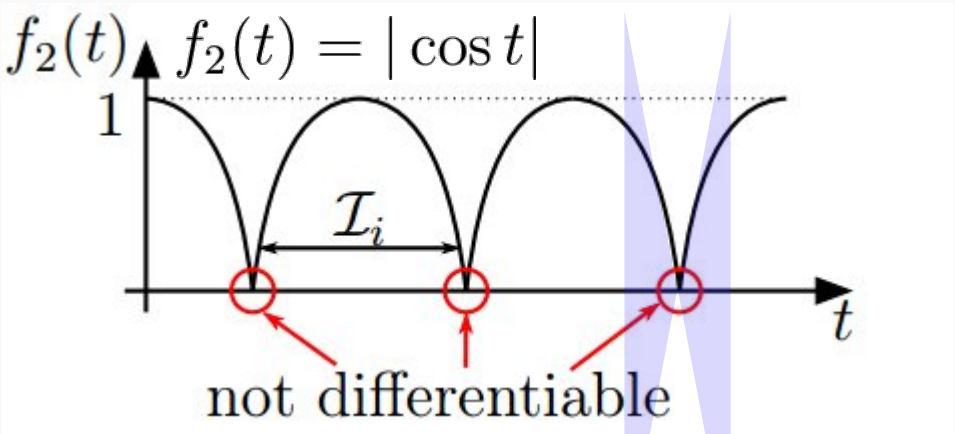


$$f : A \to \mathbb{R} \qquad \text{is } \rho\text{-Lipschitz continuous if}$$

$$\forall \mathbf{u}, \mathbf{v} \in A \qquad |f(\mathbf{u}) - f(\mathbf{v})| \leq \rho \|\mathbf{u} - \mathbf{v}\|$$

Basically, can the function be excluded from a double-cone at every point.

Can you see how this might be related to subgradients when applied to convex functions?



https://www.reddit.com/r/math/comments/3f6x5d/is_fx_abscosx_lipschitz_continuous_or_just/

Convergence of Ruppert-Polyak SGD

$$\mathbb{E}[\mathbf{v}_t|\mathbf{w}^{(t)}] \in \partial f(\mathbf{w}^{(t)}) \qquad \text{and} \qquad |f(\mathbf{u}) - f(\mathbf{v})| \leq \rho \|\mathbf{u} - \mathbf{v}\|$$

THEOREM 14.8 *Let $B, \rho > 0$. Let f be a convex function and let $\mathbf{w}^\star \in \operatorname{argmin}_{\mathbf{w}:\|\mathbf{w}\|\leq B} f(\mathbf{w})$. Assume that SGD is run for T iterations with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$. Assume also that for all t, $\|\mathbf{v}_t\| \leq \rho$ with probability 1. Then,*

$$\mathbb{E}\left[f(\bar{\mathbf{w}})\right] - f(\mathbf{w}^\star) \leq \frac{B\,\rho}{\sqrt{T}}.$$

Therefore, for any $\epsilon > 0$, to achieve $\mathbb{E}[f(\bar{\mathbf{w}})] - f(\mathbf{w}^\star) \leq \epsilon$, it suffices to run the SGD algorithm for a number of iterations that satisfies

$$T \geq \frac{B^2\rho^2}{\epsilon^2}.$$

Understanding Machine Learning: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning

Things to note:

- We get an *expectation* of how close we are to the true minimum
- Time to get there depends on how much the function can vary
- Assumes convexity! But deep neural networks aren't generally convex.

SGD for Risk Minimization

How is SGD applied in practice? How do we obtain $\mathbf{v}_t$, which is supposed to have an expected value equal to the gradient of the true loss function?

Remember, we want to minimize

$$L_{\mathcal{D}}(\mathbf{w}) = \mathbb{E}_{z\sim\mathcal{D}}[\ell(\mathbf{w}, z)]$$

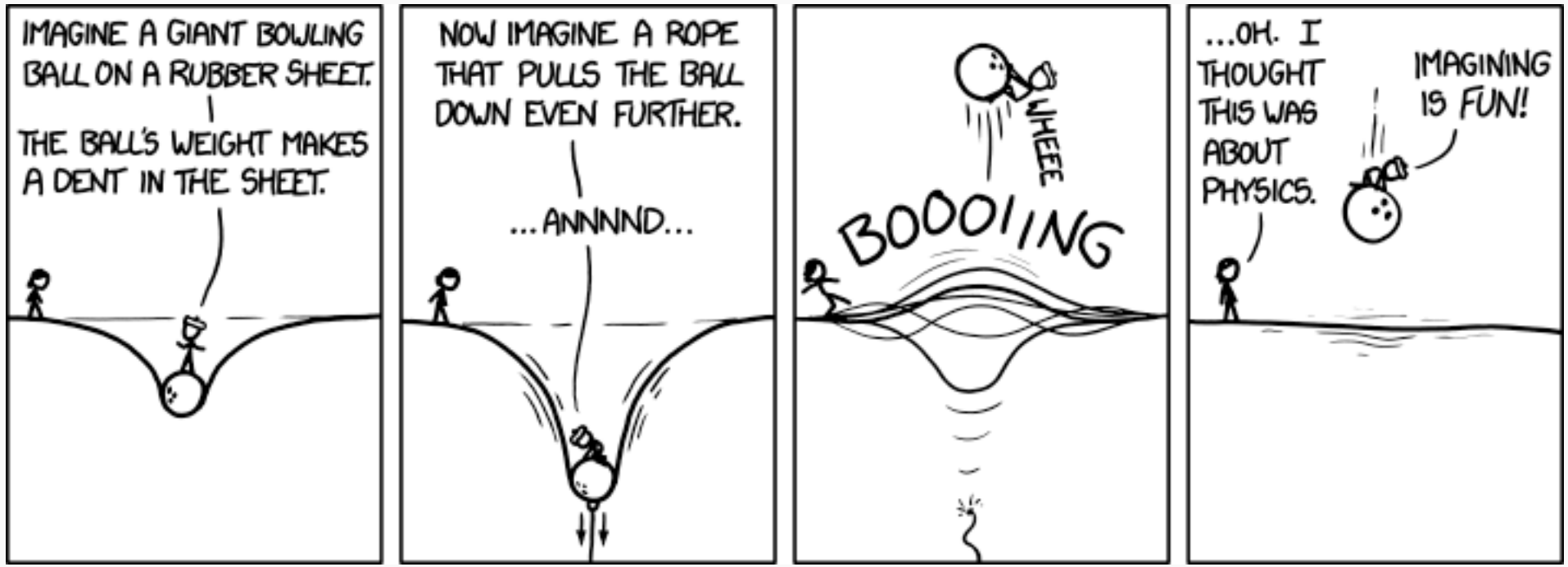
Where $\ell(\mathbf{w}, z)$ is the loss given random data point z.

$$\text{Let } \mathbf{v}_t = \nabla\ell(\mathbf{w}^{(t)}, z) \quad \text{Then,}$$

$$\mathbb{E}[\mathbf{v}_t|\mathbf{w}^{(t)}] = \mathop{\mathbb{E}}_{z\sim\mathcal{D}}[\nabla\ell(\mathbf{w}^{(t)}, z)] = \nabla \mathop{\mathbb{E}}_{z\sim\mathcal{D}}[\ell(\mathbf{w}^{(t)}, z)] = \nabla L_{\mathcal{D}}(\mathbf{w}^{(t)})$$

So, simply taking the gradient of the loss given one or a few points (i.e. on-line or minibatches) provides an unbiased estimate of the true loss.

SGD vs Batch Gradient Descent



Imagine each data point pulling a rubber sheet, and a ball rolling on this surface.

- When all are considered, you have a very accurate understanding of the true loss
  - Batch gradient descent. Good but computationally expensive – consider every point before taking a step!
- When just a few are considered, you get a rough idea of the true loss
  - SGD. Much cheaper but the ball will change directions as different points are drawn each step.

Local Minima in Non-Convex Functions

SGD assumes convexity, but we know that this doesn't hold for neural networks with hidden layers. How do we deal with local minima?

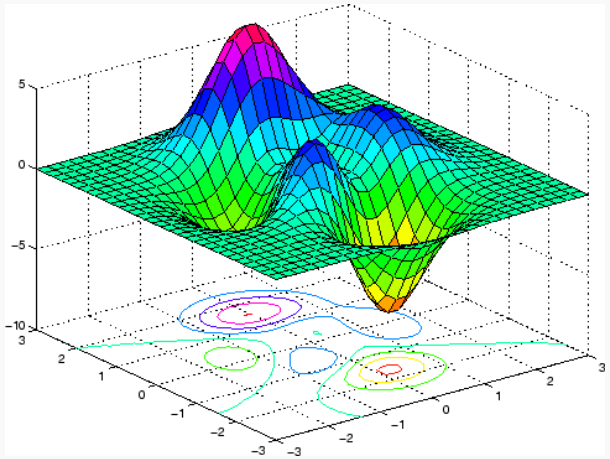
Turns out we don't need to.

In large N-dimensional domains, local minima are *extremely rare.*

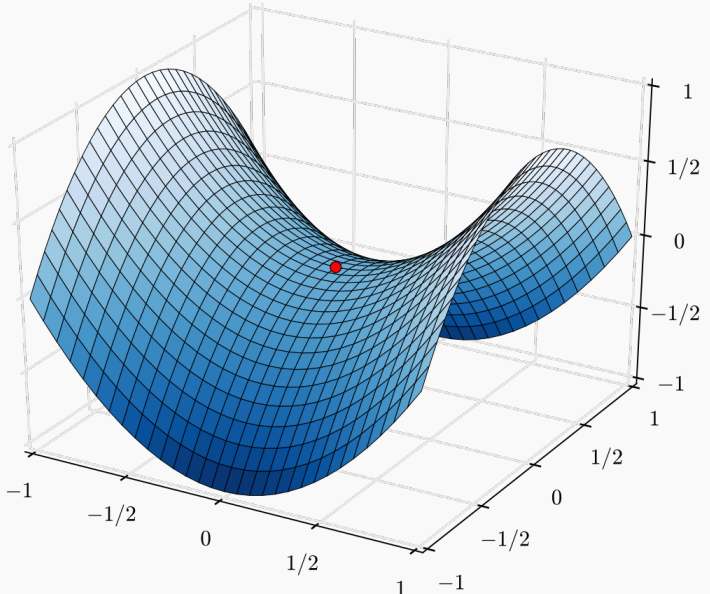
Intuitively – the eigenvalues of the Hessian have 0.5 probability of being positive or negative (Wigner's semicircle law)

What is the chance of getting heads N times in a row?

Saddle points are very common in high-dimensional spaces. Plain SGD is good at convex functions, how do we modify it to deal with non-convex ones filled with saddle points?



https://stackoverflow.com/questions/31805560/how-to-create-surface-plot-from-greyscale-image-with-matplotlib



https://en.wikipedia.org/wiki/Saddle_point

Escaping Saddle Points

Somewhat counterintuitively, the best way to escape saddle points is to just move in any direction, quickly.

Then we can get somewhere with more substantial curvature for a more informed update.

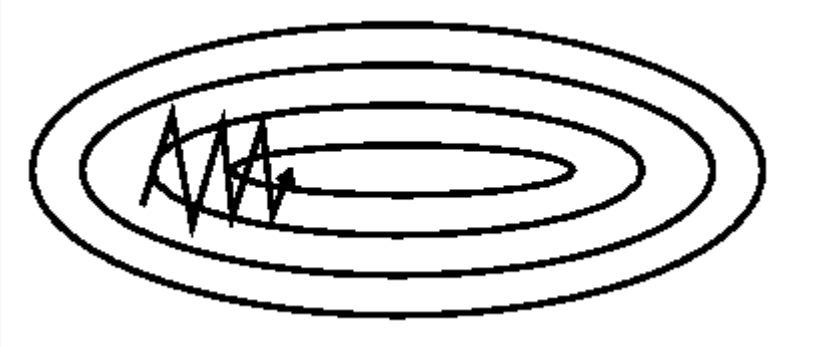
Rubick Runner

SGD with Momentum

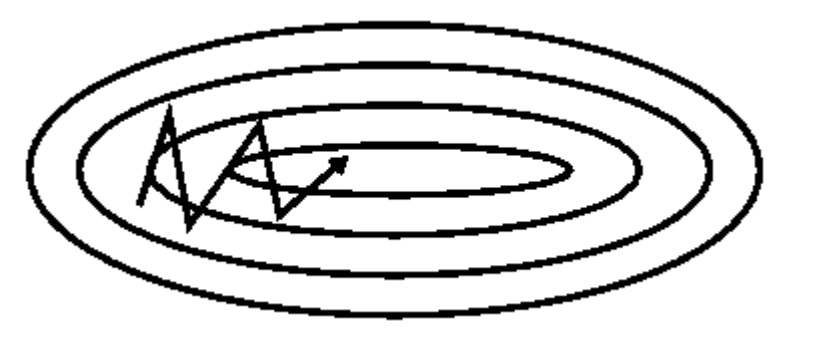
$$\mathbf{v}^{(t)} = \gamma \mathbf{v}^{(t-1)} + (1-\gamma)\nabla L(\mathbf{w}^{(t)})$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}^{(t)}$$

Maintain some of the previous "velocity". The larger the γ , the heavier the ball (or the less friction).



SGD



SGD with momentum

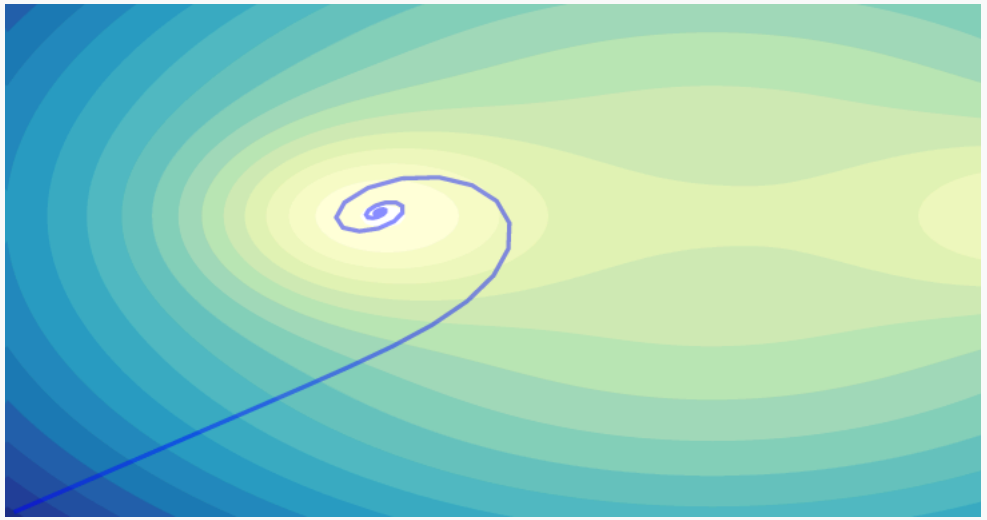
https://ruder.io/optimizing-gradient-descent/index.html#momentum

SGD with Momentum

$$\mathbf{v}^{(t)} = \gamma \mathbf{v}^{(t-1)} + (1-\gamma)\nabla L(\mathbf{w}^{(t)})$$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}^{(t)}$$

Maintain some of the previous "velocity". The larger the γ , the heavier the ball (or the less friction).



https://emiliendupont.github.io/2018/01/24/optimization-visualization/

Spiraling can happen though

Adagrad

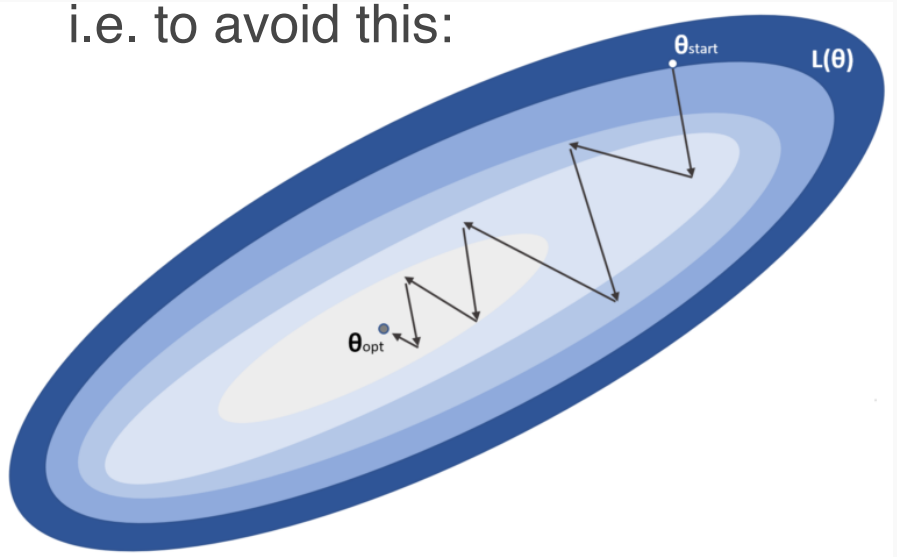
We'd like to adapt the gradient in each dimension – i.e. large steps in flatter directions, careful steps in steep directions.

$$g_{t,i} = \frac{\partial}{\partial w_i^{(t)}} L(\mathbf{w}^{(t)})$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

$$G_{t,ii} = \sum_{k=1}^{t} g_{k,i}^2$$

i.e. to avoid this:



https://www.bonaccorso.eu/2017/10/03/a-brief-and-comprehensive-guide-to-stochastic-gradient-descent-algorithms/

We see that in directions where derivatives are large, we shrink the learning rate.

But, it just keeps shrinking – after a while the learning rate will be infinitely small!

RMSprop

A way to counteract the vanishing learning rates by using a decaying average of past squared gradients.

$$v_{t,i} = 0.9 v_{t-1,i} + 0.1 g_{t,i}^2$$

$$w_i^{(t+1)} = w_i^{(t)} - \frac{\eta}{\sqrt{v_{t,i} + \epsilon}} g_{t,i}$$

Now, if there were large gradients in the past, their contribution to the average will decay exponentially with the timesteps.

If we were taking careful steps in a steep area, once we get to a flat region this allows us to crank up the speed again.

Adam

Currently the most popular gradient descent algorithm.

Stands for Adaptive Moment Estimation. Basically combines the ideas of momentum with exponentially-decaying squared past gradients.

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$

Average gradient (momentum)

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$$

Average squared gradient

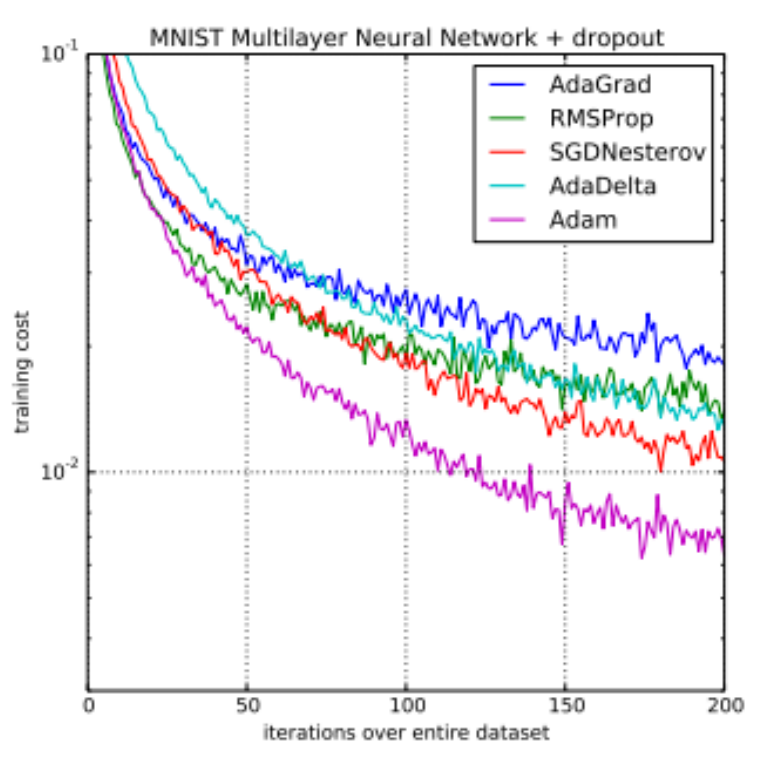
$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-(\beta_1)^t}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-(\beta_2)^t}$$

Bias correction

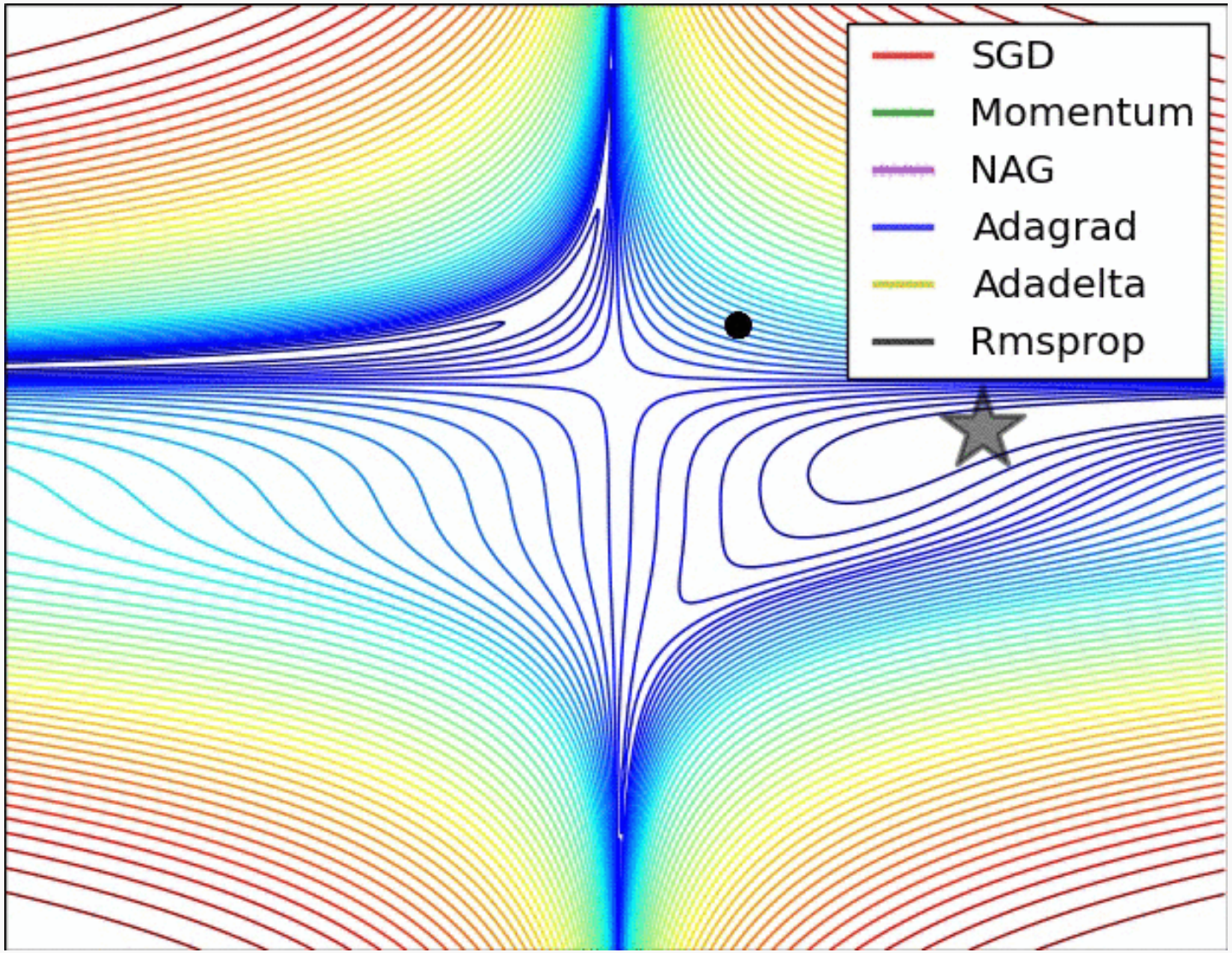
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \odot \hat{\mathbf{m}}_t$$

Update



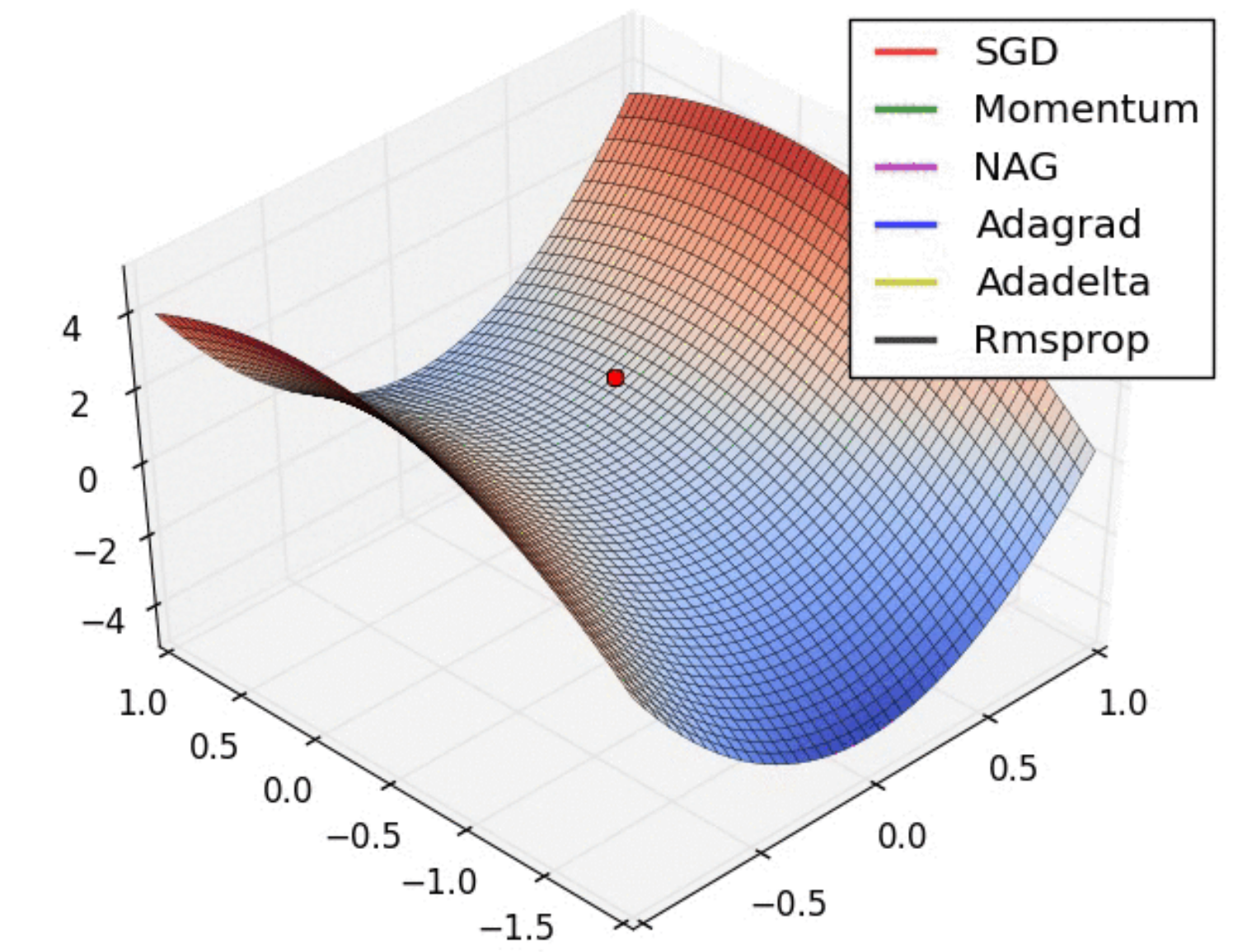
Adam: A Method for Stochastic Optimization, 2015

Animated Comparison



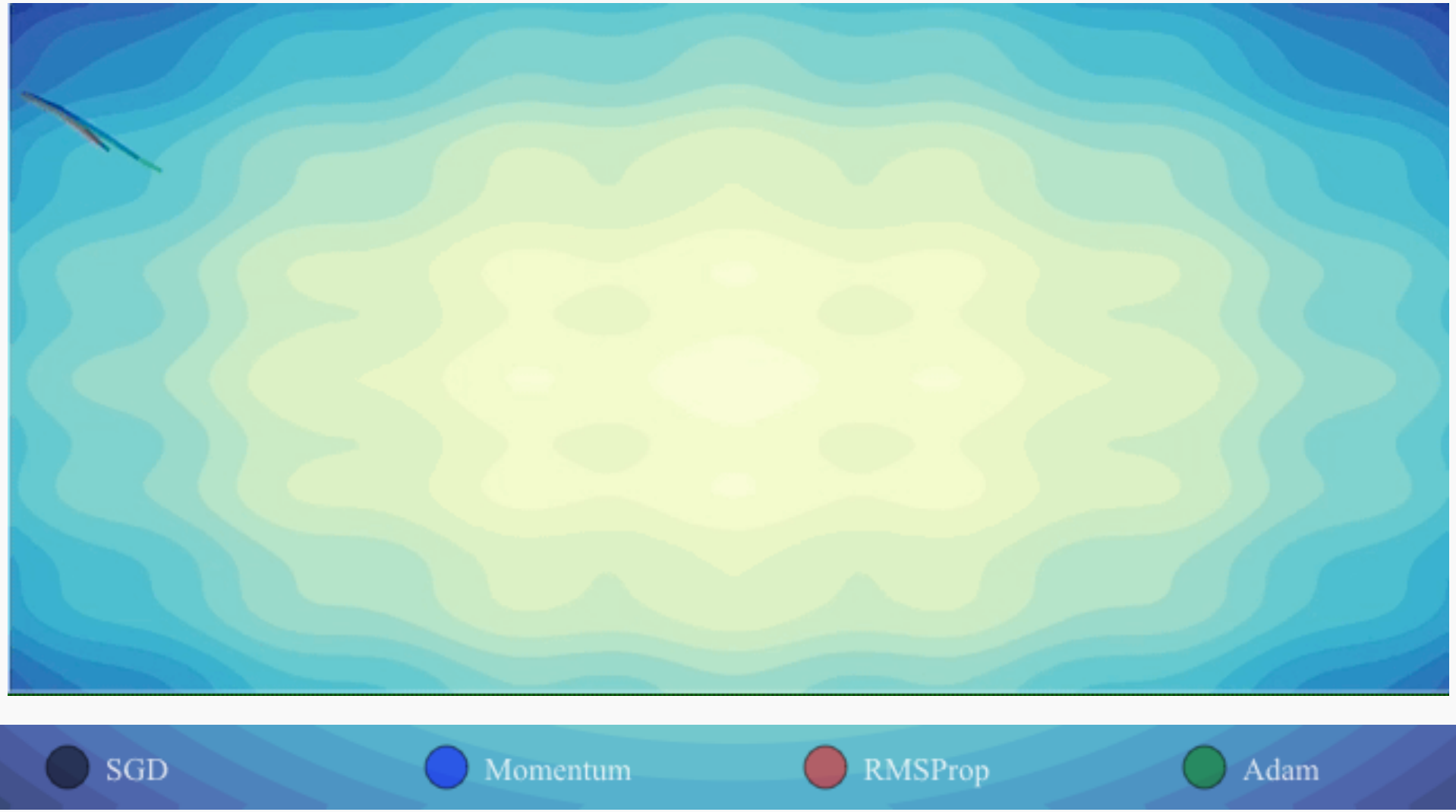
https://ruder.io/optimizing-gradient-descent/index.html

Animated Comparison



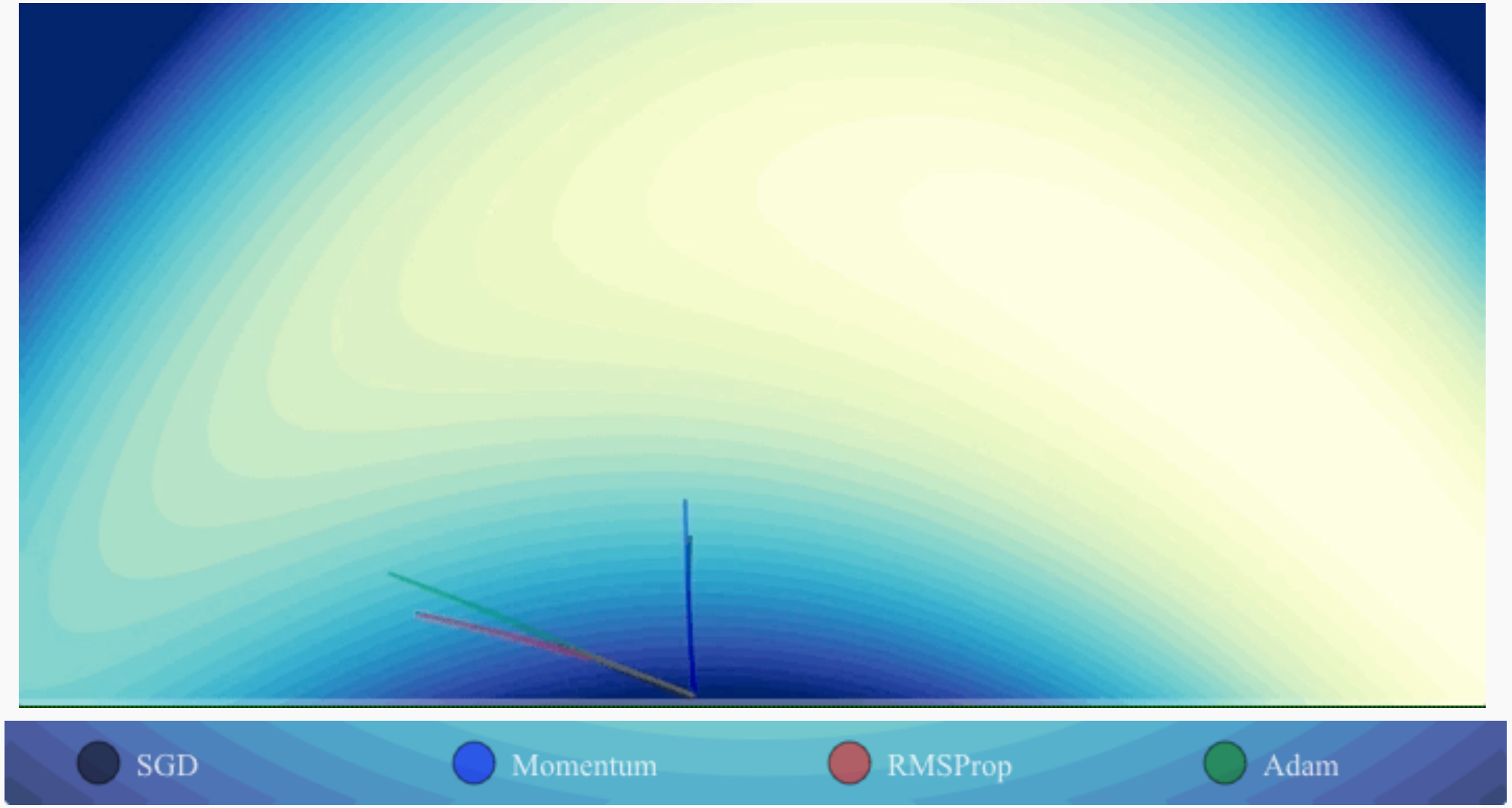
https://ruder.io/optimizing-gradient-descent/index.html

Not everything is great with the new algos



https://emiliendupont.github.io/2018/01/24/optimization-visualization/

Not everything is great with the new algos



https://emiliendupont.github.io/2018/01/24/optimization-visualization/

Adam Limitations

Adam generally performs very well compared to other methods, but recent research has found that it can sometimes converge to suboptimal solutions (such as in image classification tasks)

In fact, the original proof of convergence on convex functions had a few errors, which in 2018 was demonstrated to great effect with examples of simple convex functions on which Adam does not converge to the minimum.

Ironically, Keskar and Socher present a paper called

"Improving Generalization Performance by Switching from Adam to SGD"

So, which algorithm should you use?

There is no simple answer; a whole zoo of algorithms exist:

- SGD
- SGD with momentum
- Nesterov
- Adagrad
- Adadelta
- RMSprop
- Adam
- Adamax
- Nadam

Each has their advantages and disadvantages.

If you have sparse data, an adaptive gradient method can help you take large steps in flat dimensions

SGD can be robust, but slow.

Ultimately, the choice of gradient descent algorithm can be treated as a hyperparameter.

Additional Notes

- It can be useful to use a *learning rate scheduler* where you decrease your learning rate as a function of iteration.
- Shuffle data within an epoch to reduce optimization bias
- *Curriculum learning* is when you train your network with simpler examples first to get the weights in the right region before training it on more subtle cases.
- *Batch Normalization* can be used to improve convergence in deep networks. It discourages neurons from activating very high or very low by subtracting the batch mean and dividing my the batch standard deviation. Allows each layer to learn a little more independently from the rest.
- "Early stopping is beautiful free lunch," Geoff Hinton. When the validation error is at its minimum, stop.

Summary

- **Convex functions are important to optimization**
  1. But most of the ones we want to optimize aren't convex
  2. Good for proofs though.

- **There are many approaches to gradient descent**
  1. However, it is basically an art form at the present. Pick your favorite, but when things don't work, try others!