

Optimization and Gradient Descent

**INFO-4604, Applied Machine Learning
University of Colorado Boulder**

September 11, 2018
Prof. Michael Paul

Prediction Functions

Remember: a **prediction function** is the function that predicts what the output should be, given the input.

Prediction Functions

Linear regression:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

Linear classification (perceptron):

$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} + b \geq 0 \\ -1, & \mathbf{w}^T \mathbf{x} + b < 0 \end{cases}$$

Need to *learn* what \mathbf{w} should be!

Learning Parameters

Goal is to learn to minimize error

- Ideally: true error
- Instead: training error

The **loss function** gives the training error when using parameters w , denoted $L(w)$.

- Also called **cost function**
- More general: **objective function**
(in general objective could be to minimize or maximize;
with loss/cost functions, we want to minimize)

Learning Parameters

Goal is to minimize loss function.

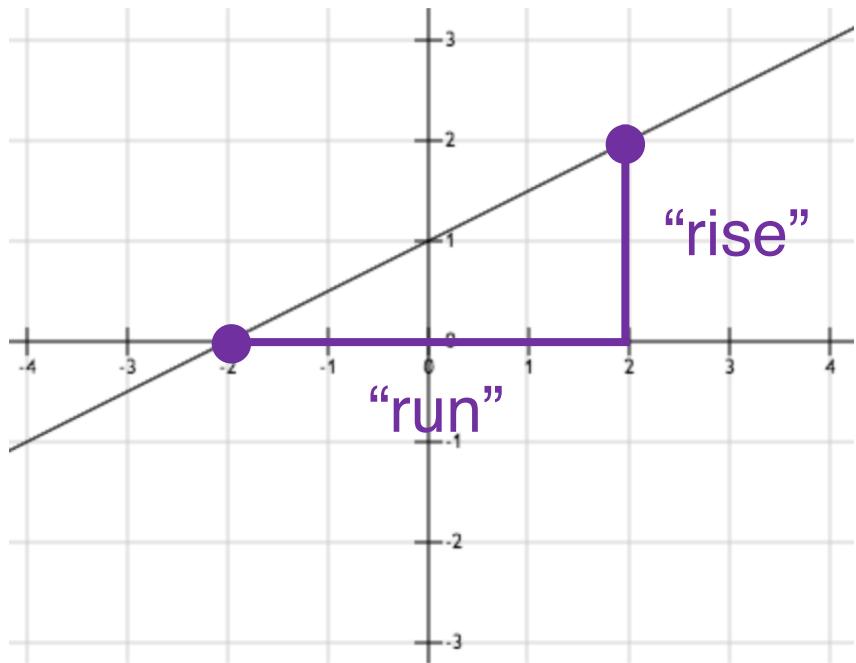
How do we minimize a function?

Let's review some math.

Rate of Change

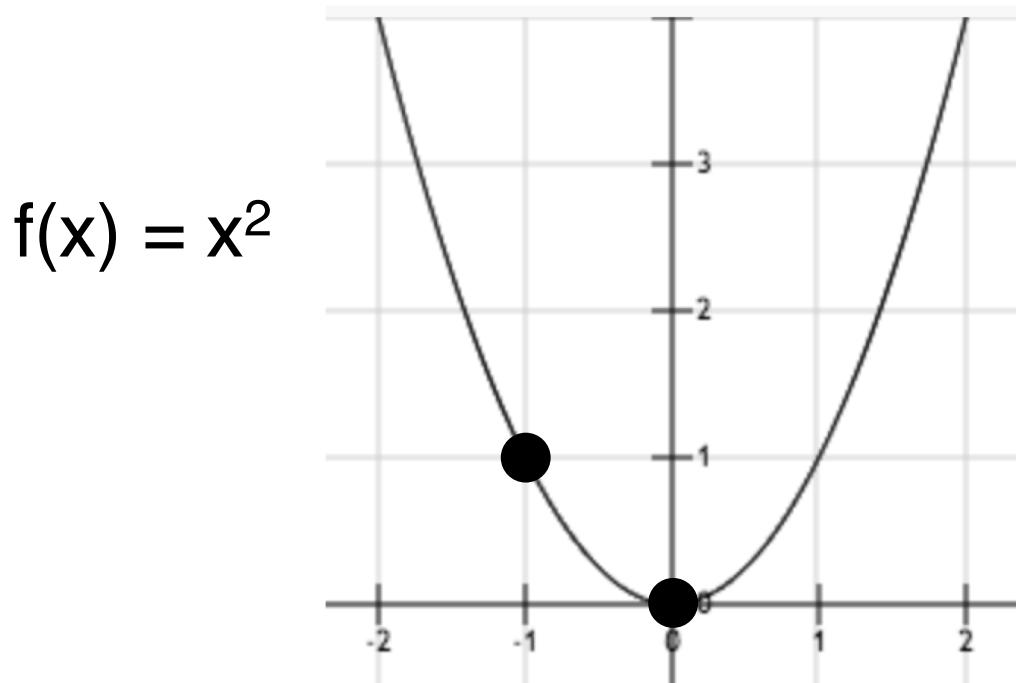
The slope of a line is also called the **rate of change** of the line.

$$y = \frac{1}{2}x + 1$$



Rate of Change

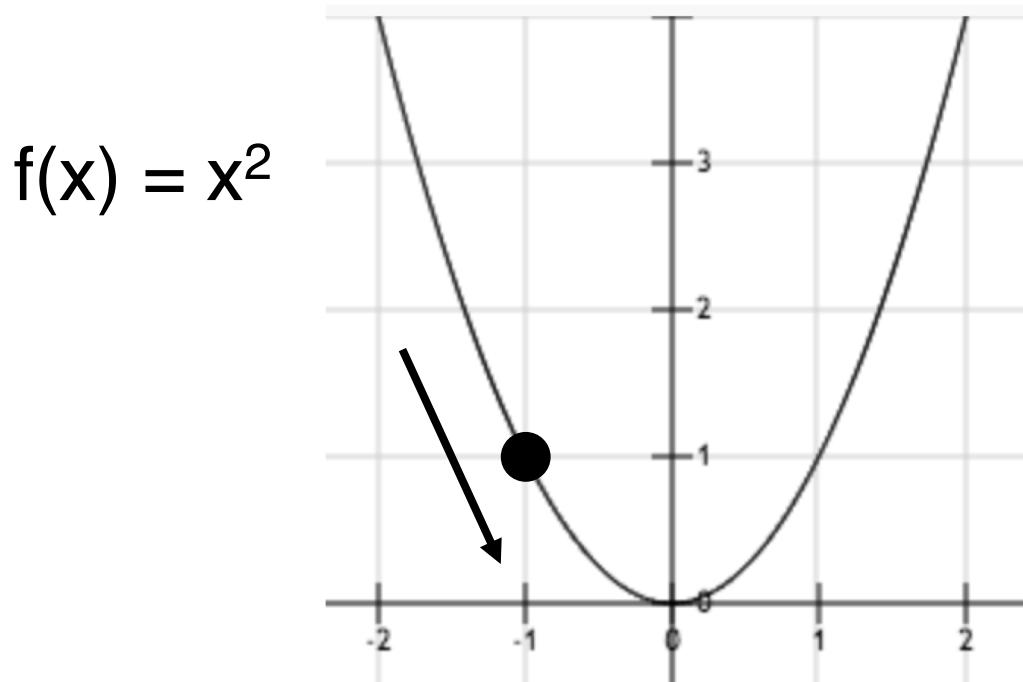
For nonlinear functions, the “rise over run” formula gives you the average rate of change between two points



Average slope from
 $x=-1$ to $x=0$ is:
-1

Rate of Change

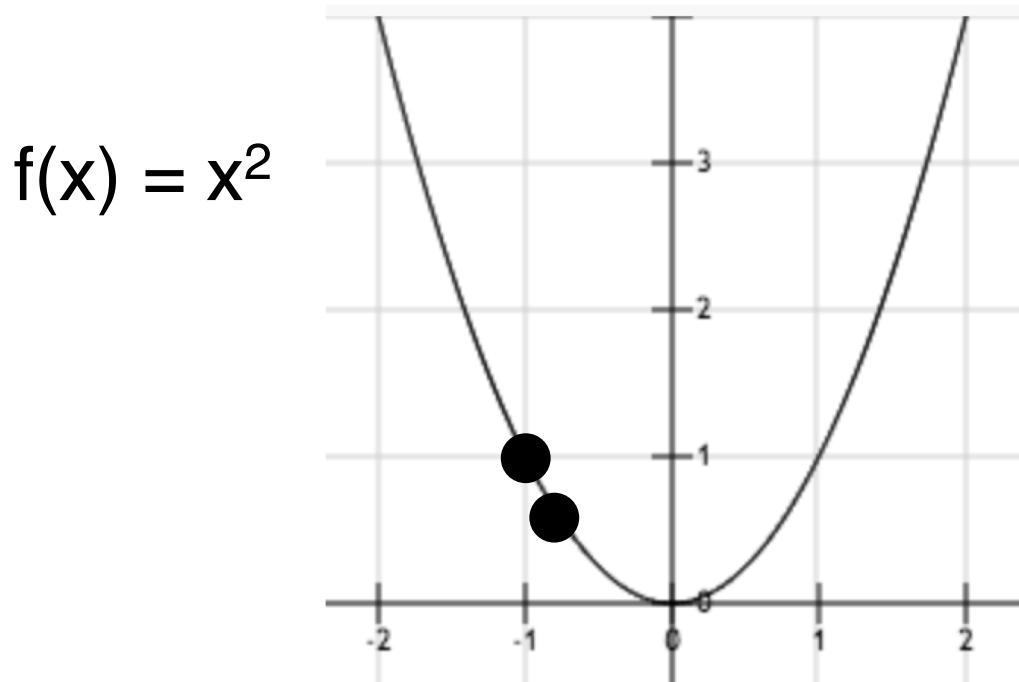
There is also a concept of rate of change at individual points (rather than two points)



Slope at $x=-1$ is:
-2

Rate of Change

The slope at a point is called the **derivative** at that point



$$f(x) = x^2$$

Intuition:
Measure the slope
between two points
that are really close
together

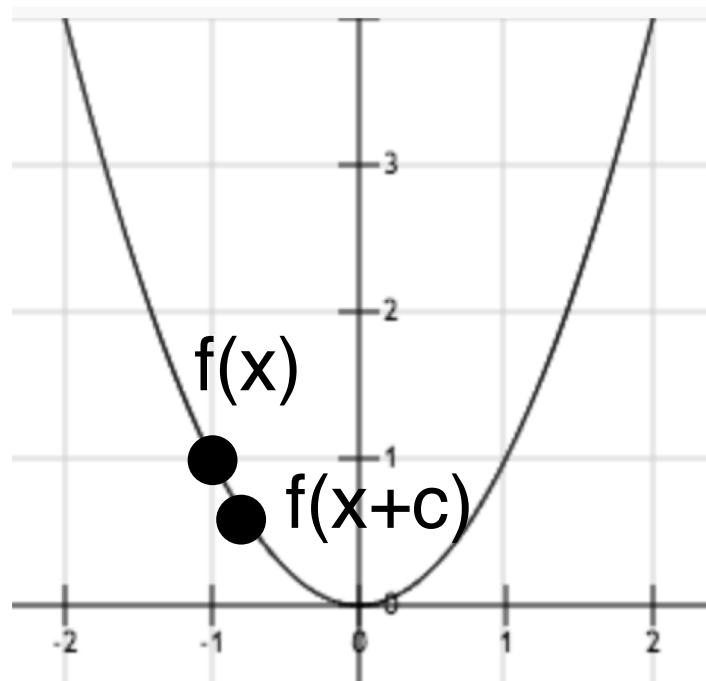
Rate of Change

The slope at a point is called the **derivative** at that point

Intuition: Measure the slope between two points that are really close together

$$\frac{f(x + c) - f(x)}{c}$$

Limit as c goes to zero



Maxima and Minima

Whenever there is a peak in the data, this is a **maximum**

The **global** maximum is the highest peak in the entire data set, or the largest $f(x)$ value the function can output

A **local** maximum is any peak, when the rate of change switches from positive to negative

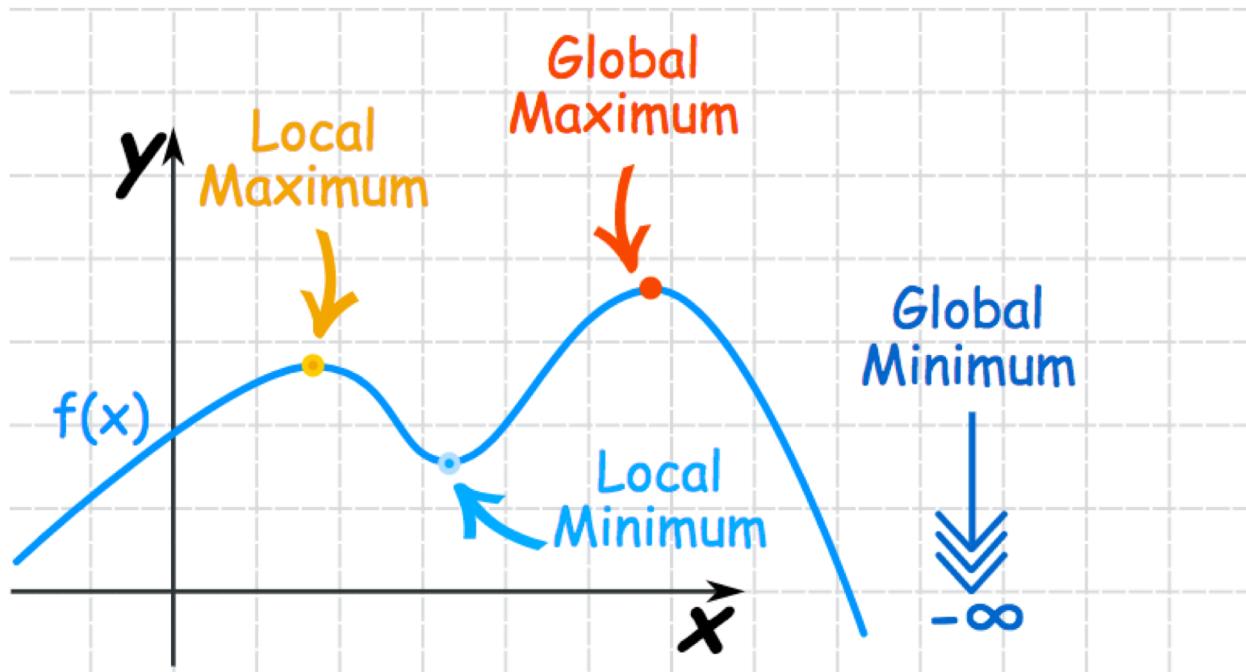
Maxima and Minima

Whenever there is a trough in the data, this is a **minimum**

The **global** minimum is the lowest trough in the entire data set, or the smallest $f(x)$ value the function can output

A **local** minimum is any trough, when the rate of change switches from negative to positive

Maxima and Minima



From: <https://www.mathsisfun.com/algebra/functions-maxima-minima.html>

All global maxima and minima are also local maxima and minima

Derivatives

The derivative of $f(x) = x^2$ is $2x$

Other ways of writing this:

$$f'(x) = 2x$$

$$\frac{d}{dx} [x^2] = 2x$$

$$df/dx = 2x$$

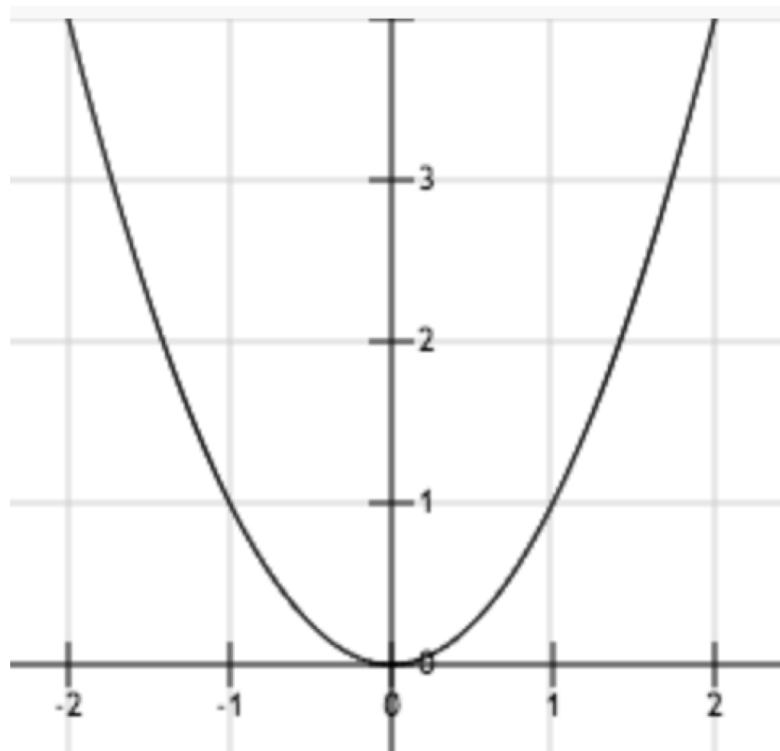
The derivative is also a function! It depends on the value of x .

- The rate of change is different at different points

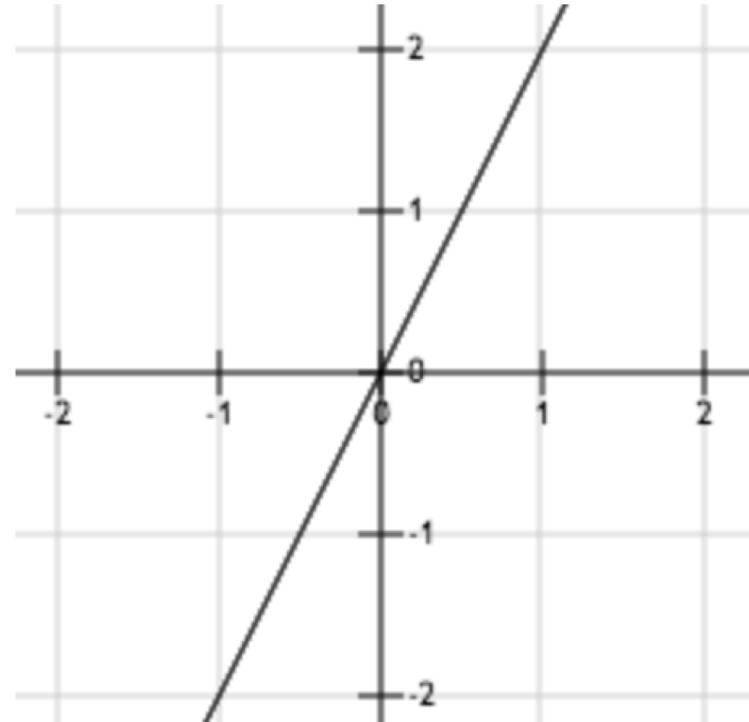
Derivatives

The derivative of $f(x) = x^2$ is $2x$

$f(x)$



$f'(x)$



Derivatives

How to calculate a derivative?

- Not going to do it in this class.

Some software can do it for you.

- [Wolfram Alpha](#)

The image shows a screenshot of the Wolfram Alpha search interface. At the top, the Wolfram Alpha logo is displayed. Below the logo is a search bar containing the query "d/dx x^2". Underneath the search bar are several small icons representing different types of results or tools. To the right of these icons, the word "Web" is visible. A large button labeled "Derivative:" is present, and below it, the mathematical expression $\frac{d}{dx}(x^2) = 2 x$ is shown.

Derivatives

What if a function has multiple arguments?

Ex: $f(x_1, x_2) = 3x_1 + 5x_2$

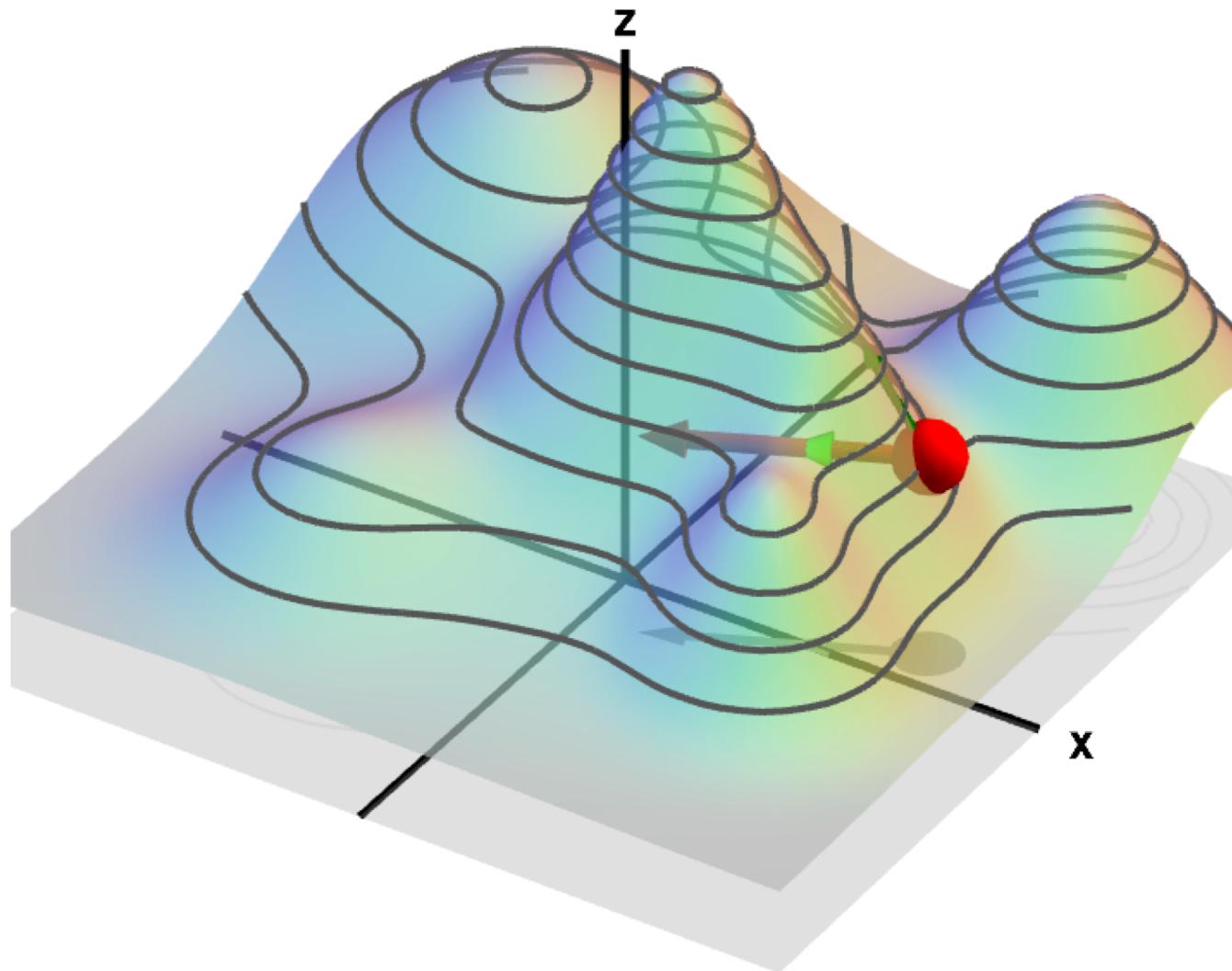
$df/dx_1 = 3 + 5x_2$ The derivative “with respect to” x_1

$df/dx_2 = 3x_1 + 5$ The derivative “with respect to” x_2

These two functions are called **partial derivatives**.

The vector of all partial derivatives for a function f is called the **gradient** of the function:

$$\nabla f(x_1, x_2) = \langle df/dx_1, df/dx_2 \rangle$$



$$\theta = 0$$



$$u = (-0.91, -0.42)$$

$$a = (6.7, 1.1)$$

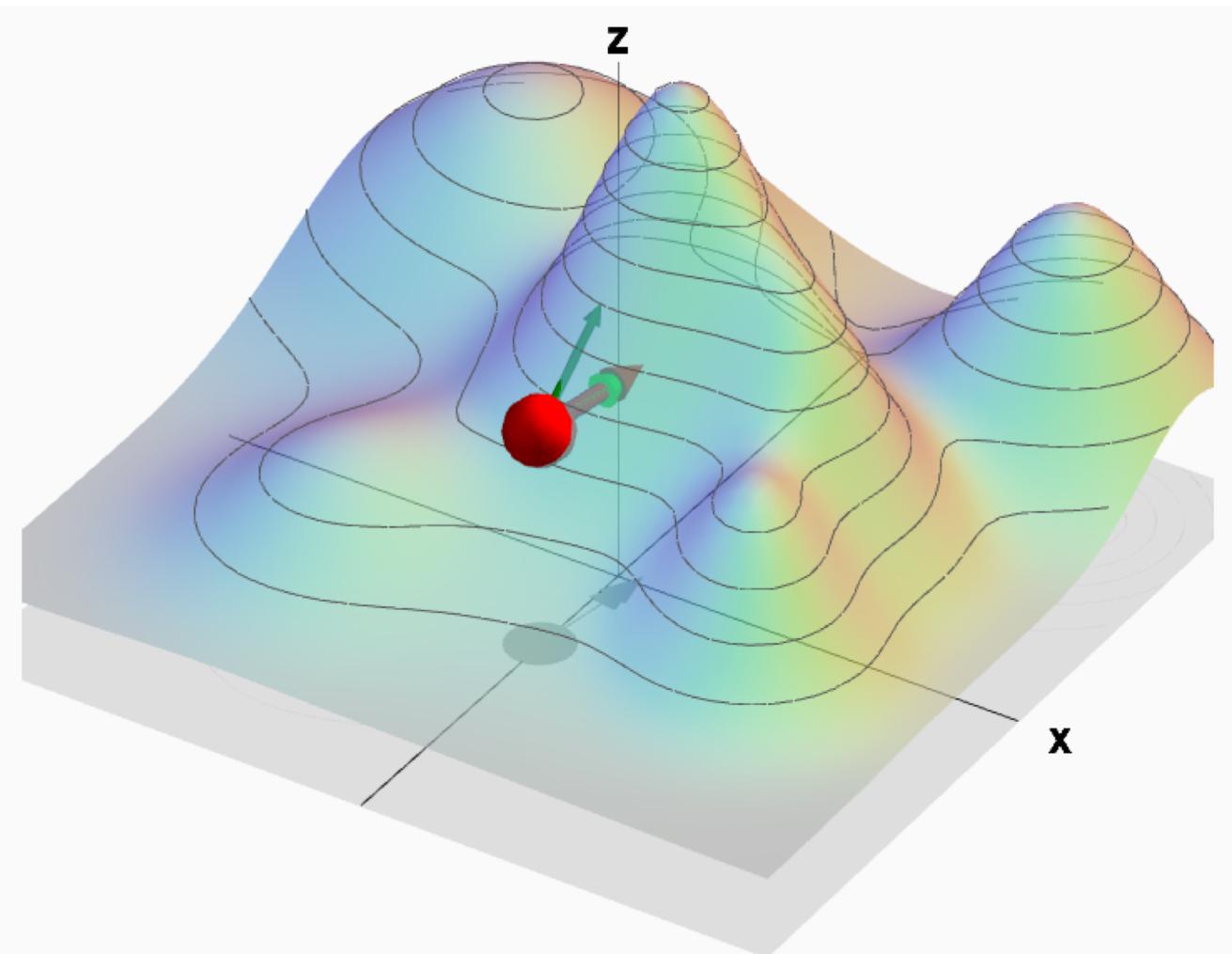
$$\nabla f(a) = (-1.81, -0.85)$$

$$D_u f(a) = 2.00$$



$$\|\nabla f(a)\| = 2.00$$

From: http://mathinsight.org/directional_derivative_gradient_introduction



$$\theta = 0$$



$$u = (0.15, 0.99)$$

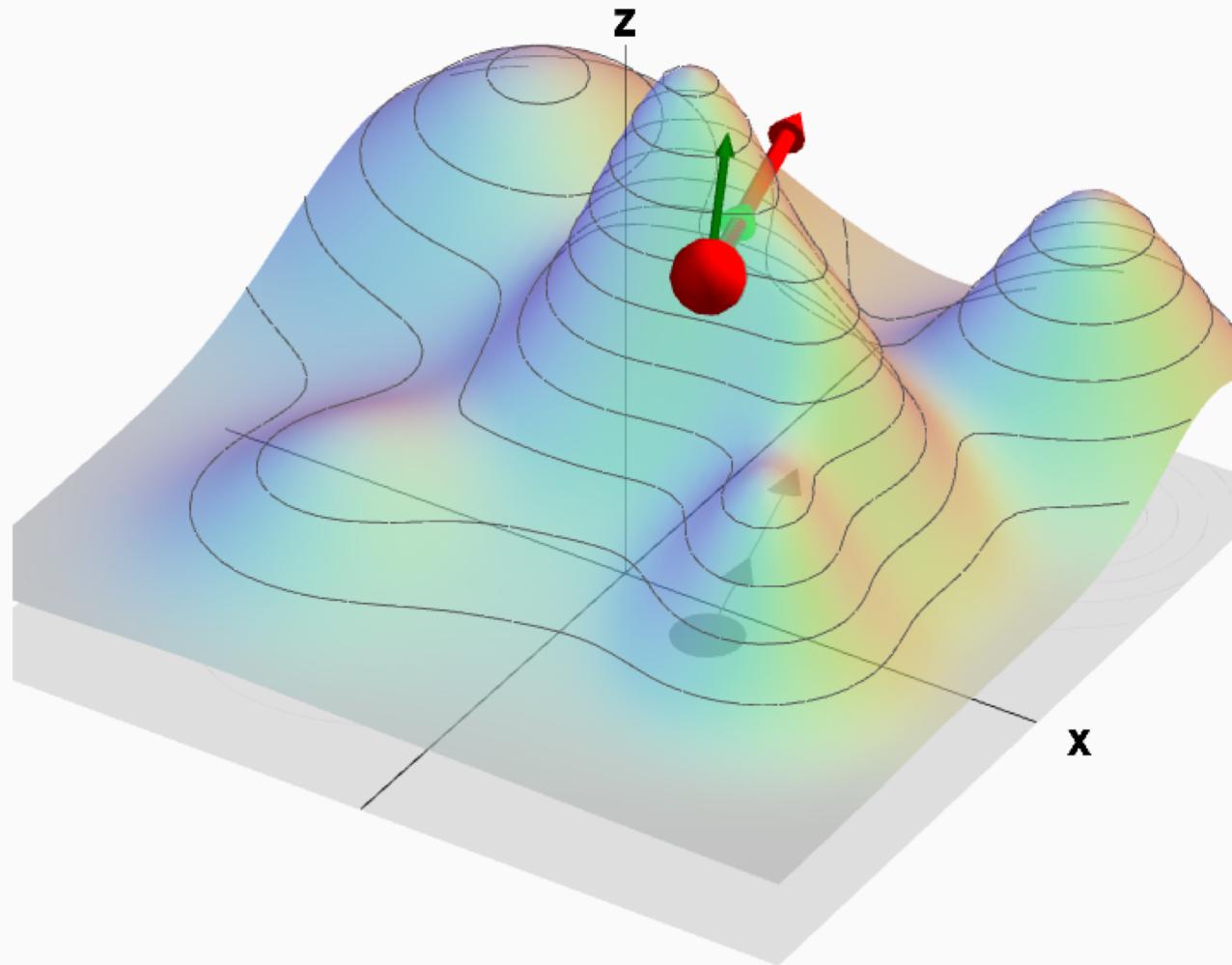
$$a = (-0.0, -3.0)$$

$$\nabla f(a) = (0.17, 1.09)$$

$$D_u f(a) = 1.11$$

$$\|\nabla f(a)\| = 1.11$$

From: http://mathinsight.org/directional_derivative_gradient_introduction



$$\theta = 0$$

$$u = (-0.26, 0.97)$$

$$a = (2.6, -1.0)$$

$$\nabla f(a) = (-0.56, 2.07)$$

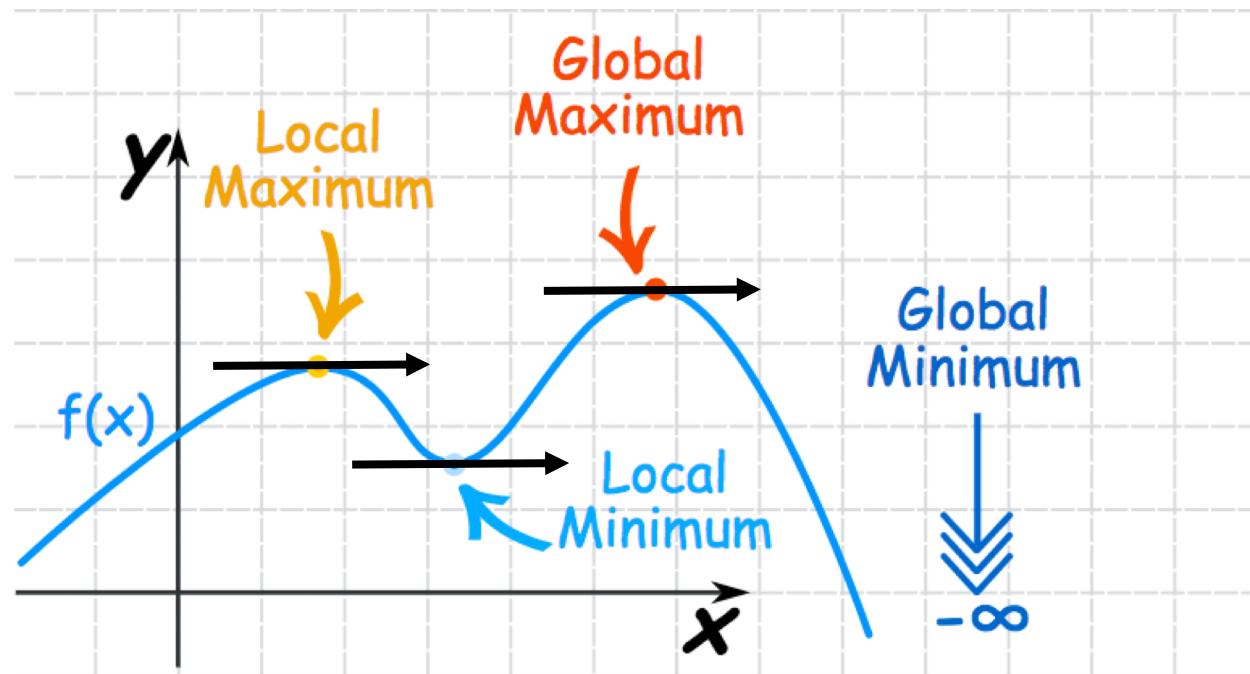
$$D_u f(a) = 2.14$$

$$|\nabla f(a)| = 2.14$$

From: http://mathinsight.org/directional_derivative_gradient_introduction

Finding Minima

The derivative is *zero* at any local maximum or minimum.



Finding Minima

The derivative is *zero* at any local maximum or minimum.

One way to find a minimum: set $f'(x)=0$ and solve for x .

$$f(x) = x^2$$

$$f'(x) = 2x$$

$f'(x) = 0$ when $x = 0$, so minimum at $x = 0$

Finding Minima

The derivative is *zero* at any local maximum or minimum.

One way to find a minimum: set $f'(x)=0$ and solve for x .

- For most functions, there isn't a way to solve this.
- Instead: algorithmically search different values of x until you find one that results in a gradient near 0.

Finding Minima

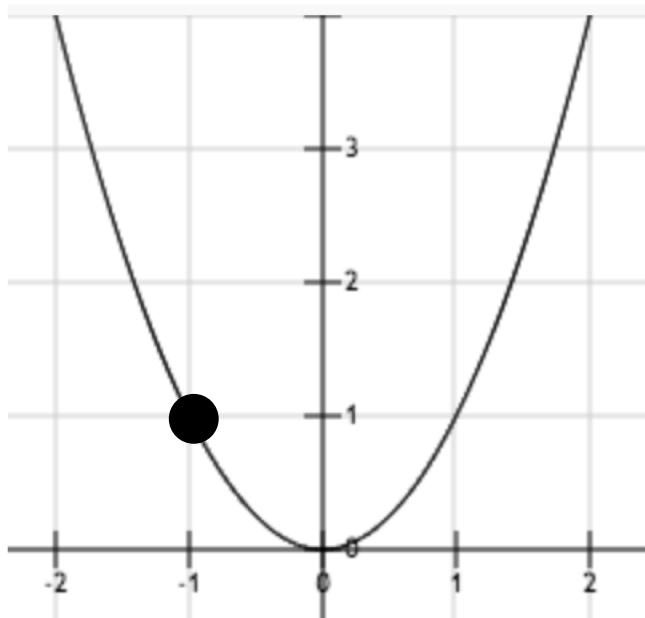
If the derivative is positive, the function is **increasing**.

- Don't move in that direction, because you'll be moving away from a trough.

If the derivative is negative, the function is **decreasing**.

- Keep going, since you're getting closer to a trough

Finding Minima

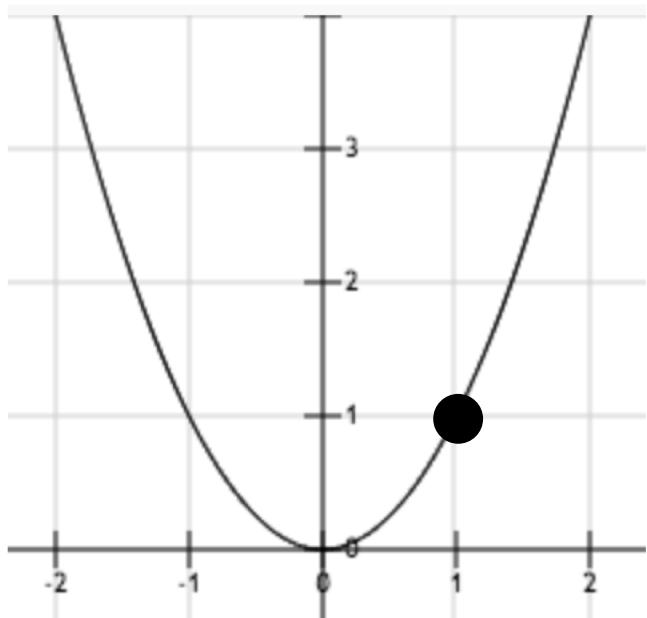


$$f'(-1) = -2$$

At $x=-1$, the function is decreasing as x gets larger. This is what we want, so let's make x larger. Increase x by the size of the gradient:

$$-1 + 2 = 1$$

Finding Minima

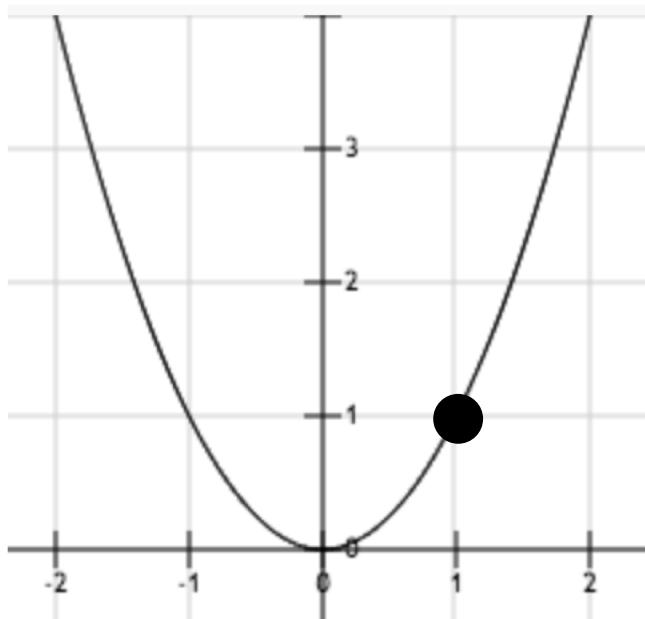


$$f'(-1) = -2$$

At $x=-1$, the function is decreasing as x gets larger. This is what we want, so let's make x larger. Increase x by the size of the gradient:

$$-1 + 2 = 1$$

Finding Minima

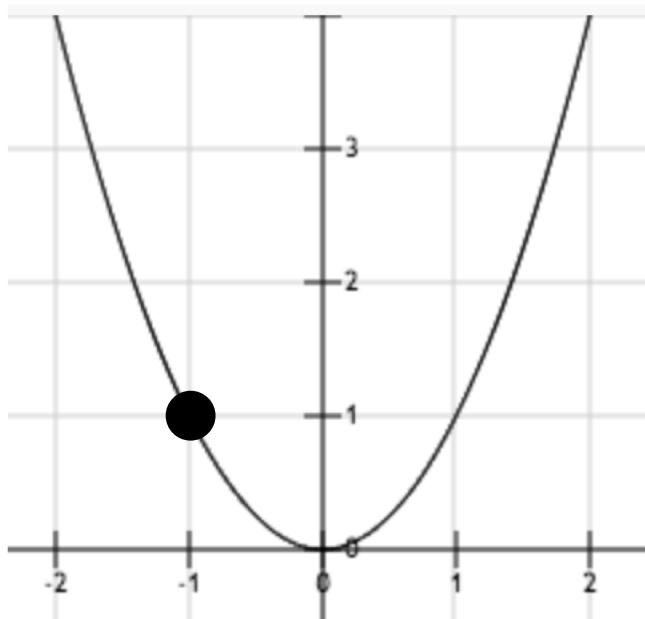


$$f'(1) = 2$$

At $x=1$, the function is increasing as x gets larger. This is not what we want, so let's make x smaller. Decrease x by the size of the gradient:

$$1 - 2 = -1$$

Finding Minima

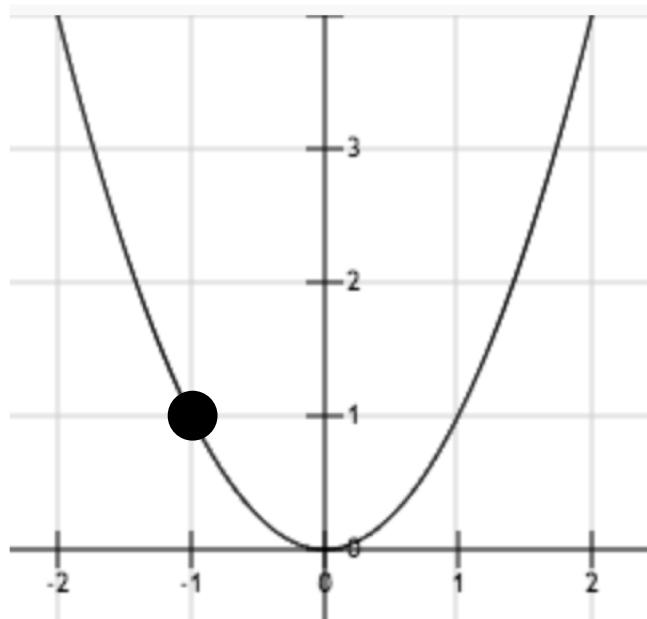


$$f'(1) = 2$$

At $x=1$, the function is increasing as x gets larger. This is not what we want, so let's make x smaller. Decrease x by the size of the gradient:

$$1 - 2 = -1$$

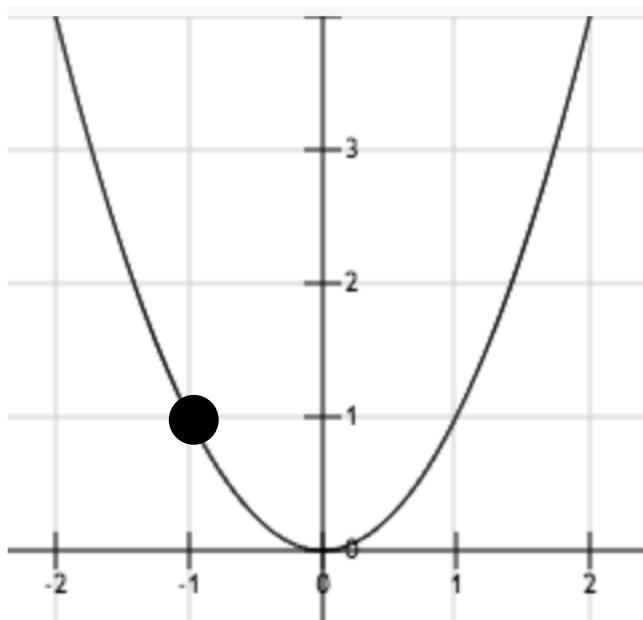
Finding Minima



We will keep jumping between the same two points this way.

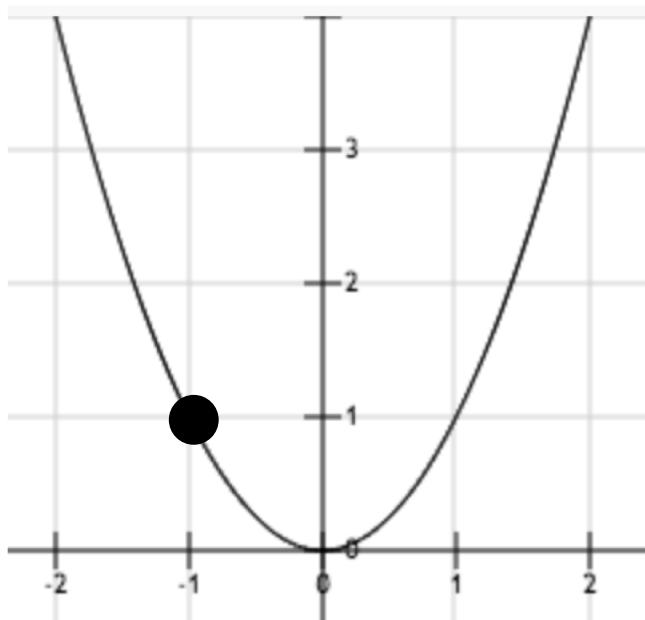
We can fix this by using a learning rate or step size.

Finding Minima



$$f'(-1) = -2$$
$$x += 2\eta =$$

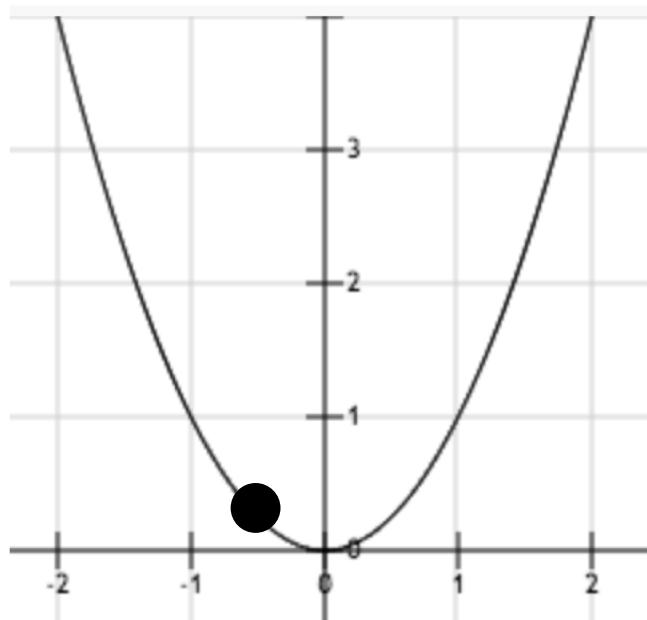
Finding Minima



$$f'(-1) = -2$$
$$x += 2\eta =$$

Let's use $\eta = 0.25$.

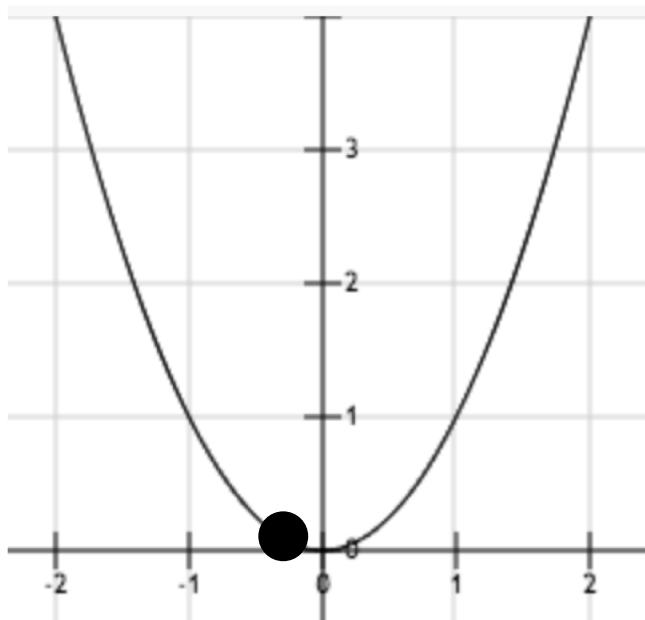
Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

Finding Minima



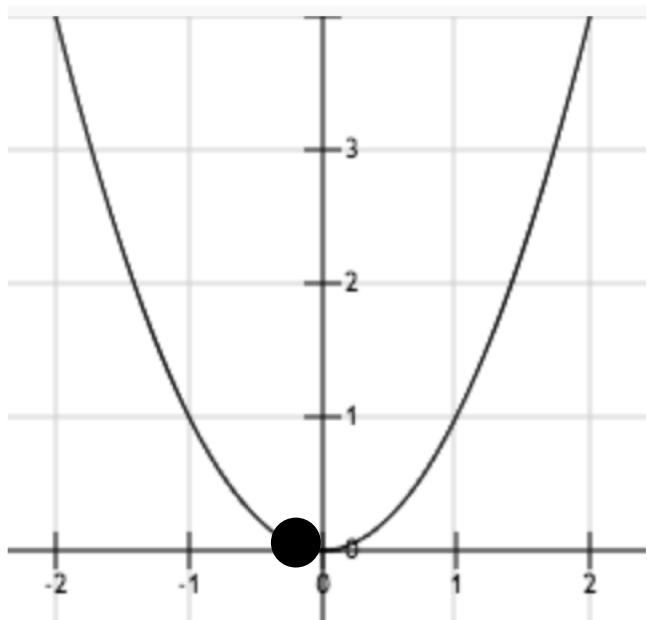
$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

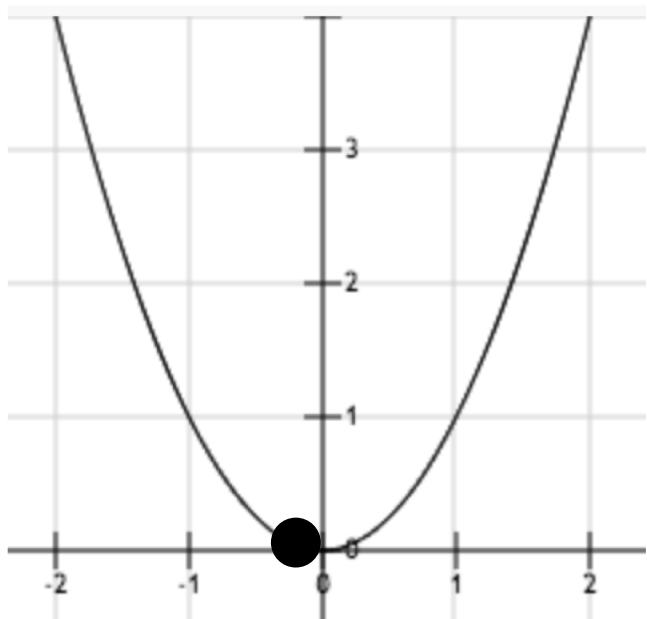
$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

$$f'(-0.25) = -0.5$$

$$x = -0.25 + 0.5(.25) = -0.125$$

Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

$$f'(-0.25) = -0.5$$

$$x = -0.25 + 0.5(.25) = -0.125$$

Eventually we'll reach $x=0$.

Gradient Descent

1. Initialize the parameters \mathbf{w} to some guess
(usually all zeros, or random values)
2. Update the parameters:
$$\mathbf{w} = \mathbf{w} - \eta \nabla L(\mathbf{w})$$
3. Update the learning rate η
(How? Later...)
4. Repeat steps 2-3 until $\nabla L(\mathbf{w})$ is close to zero.

Gradient Descent

Gradient descent is guaranteed to eventually find a *local* minimum if:

- the learning rate is decreased appropriately;
- a finite local minimum exists (i.e., the function doesn't keep decreasing forever).

Gradient Ascent

What if we want to find a local *maximum*?

Same idea, but the update rule moves the parameters in the opposite direction:

$$\mathbf{w} = \mathbf{w} + \eta \nabla L(\mathbf{w})$$

Learning Rate

In order to guarantee that the algorithm will converge, the learning rate should decrease over time. Here is a general formula.

At iteration t:

$$\eta_t = c_1 / (t^a + c_2),$$

where $0.5 < a < 2$

$$c_1 > 0$$

$$c_2 \geq 0$$

Stopping Criteria

For most functions, you probably won't get the gradient to be exactly equal to **0** in a reasonable amount of time.

Once the gradient is sufficiently close to **0**, stop trying to minimize further.

How do we measure how close a gradient is to **0**?

Distance

A special case is the distance between a point and zero (the *origin*).

$$d(\mathbf{p}, \mathbf{0}) = \sqrt{\sum_{i=1}^k (p_i)^2}$$

This is called the **Euclidean norm** of \mathbf{p}

- A norm is a measure of a vector's length
- The Euclidean norm is also called the **L2 norm**

Distance

A special case is the distance between a point and zero (the *origin*).

$$d(\mathbf{p}, \mathbf{0}) = \sqrt{\sum_{i=1}^k (p_i)^2}$$

Also written: $\|\mathbf{p}\|$

Stopping Criteria

Stop when the norm of the gradient is below some threshold, θ :

$$\|\nabla L(w)\| < \theta$$

Common values of θ are around .01, but if it is taking too long, you can make the threshold larger.

Gradient Descent

1. Initialize the parameters w to some guess (usually all zeros, or random values)
2. Update the parameters:
$$w = w - \eta \nabla L(w)$$
$$\eta = c_1 / (t^a + c_2)$$
3. Repeat step 2 until $\|\nabla L(w)\| < \theta$ or until the maximum number of iterations is reached.

Revisiting Perceptron

In perceptron, you increase the weights if they were an underestimate and decrease if they were an overestimate.

$$w_j += \eta (y_i - f(x_i)) x_{ij}$$

This looks similar to the gradient descent rule.

- Is it? We'll come back to this.

Adaline

Similar algorithm to perceptron (but uncommon):

Predictions use the same function:

$$f(\mathbf{x}) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} \geq 0 \\ -1, & \mathbf{w}^T \mathbf{x} < 0 \end{cases}$$

(here the bias b is folded into the weight vector \mathbf{w})

Adaline

Perceptron minimizes the number of errors.

Adaline instead tries to make $\mathbf{w}^T \mathbf{x}$ close to the correct value (1 or -1, even though $\mathbf{w}^T \mathbf{x}$ can be any real number).

Loss function for Adaline:

$$L(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

This is called the **squared error**.
(This is the same loss function used for linear regression.)

Adaline

What is the derivative of the loss?

$$L(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

$$\frac{dL}{dw_j} = \sum_{i=1}^N -2 x_{ij} (y_i - \mathbf{w}^\top \mathbf{x}_i)$$

Adaline

The gradient descent algorithm for Adaline updates each feature weight using the rule:

$$w_j += n \sum_{i=1}^N 2 x_{ij} (y_i - w^T x_i)$$

Two main differences from perceptron:

- $(y_i - w^T x_i)$ is a real value, instead of a binary value (perceptron either correct or incorrect)
- The update is based on the entire training set, instead of one instance at a time.

Adaline

The gradient descent algorithm for Adaline updates each feature weight using the rule:

$$w_j += n \sum_{i=1}^N 2 x_{ij} (y_i - w^T x_i)$$

Two main differences from perceptron:

- $(y_i - w^T x_i)$ is a real value, instead of a binary value (perceptron either correct or incorrect)
- The update is based on the entire training set, instead of one instance at a time.

Stochastic Gradient Descent

A variant of gradient descent makes updates using an approximate of the gradient that is only based on one instance at a time.

$$L_i(\mathbf{w}) = (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

$$\frac{dL_i}{dw_j} = -2 x_{ij} (y_i - \mathbf{w}^T \mathbf{x}_i)$$

Stochastic Gradient Descent

General algorithm for SGD:

1. Iterate through the instances in a random order
 - a) For each instance x_i , update the weights based on the gradient of the loss for that instance only:

$$\mathbf{w} = \mathbf{w} - \eta \nabla L_i(\mathbf{w}; \mathbf{x}_i)$$

The gradient for one instance's loss is an approximation to the true gradient

- stochastic = random
The *expected* gradient is the true gradient

Adaline

The gradient descent algorithm for Adaline updates each feature weight using the rule:

$$w_j += n \sum_{i=1}^N 2 x_{ij} (y_i - w^T x_i)$$

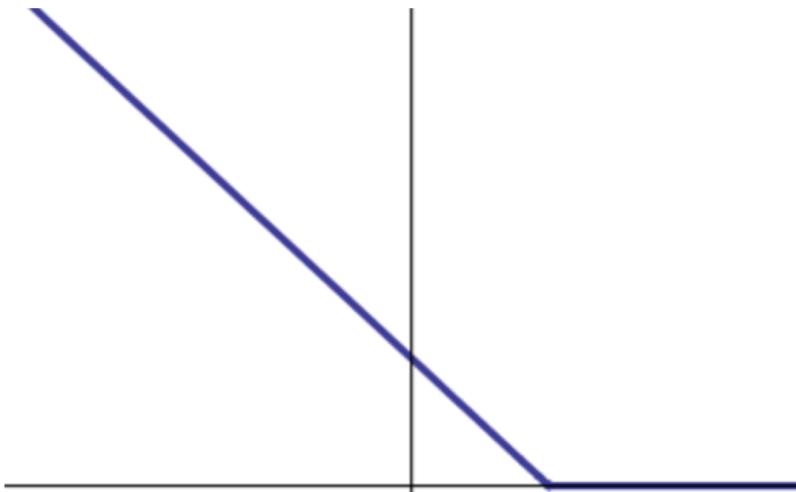
Two main differences from perceptron:

- $(y_i - w^T x_i)$ is a real value, instead of a binary value (perceptron either correct or incorrect)
- The update is based on the entire training set, instead of one instance at a time.

Revisiting Perceptron

Perceptron has a different loss function:

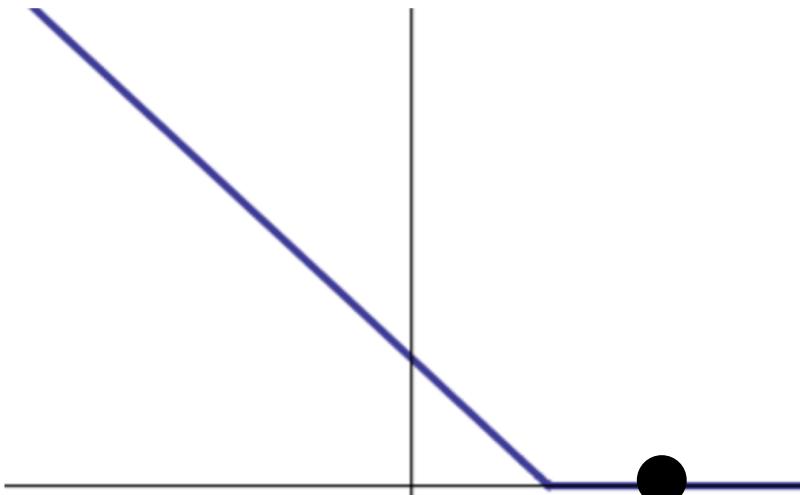
$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$



Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$

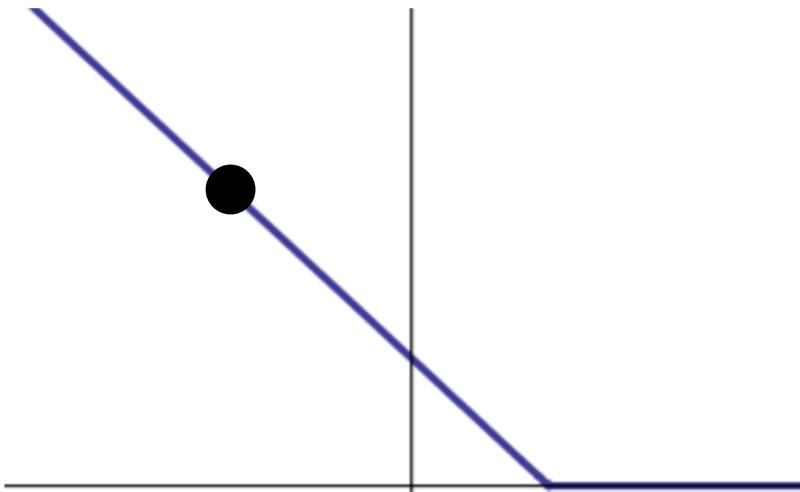


The derivative here is 0.
No gradient descent
updates if the prediction
was correct.

Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$

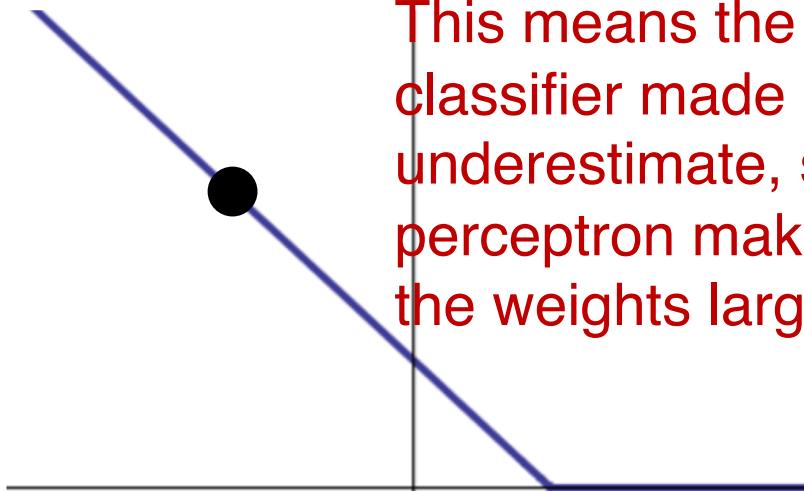


The derivative here is $-y_i x_{ij}$. If x_{ij} is positive, dL_i/w_j will be negative when y_i is positive, so the gradient descent update will be positive.

Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$



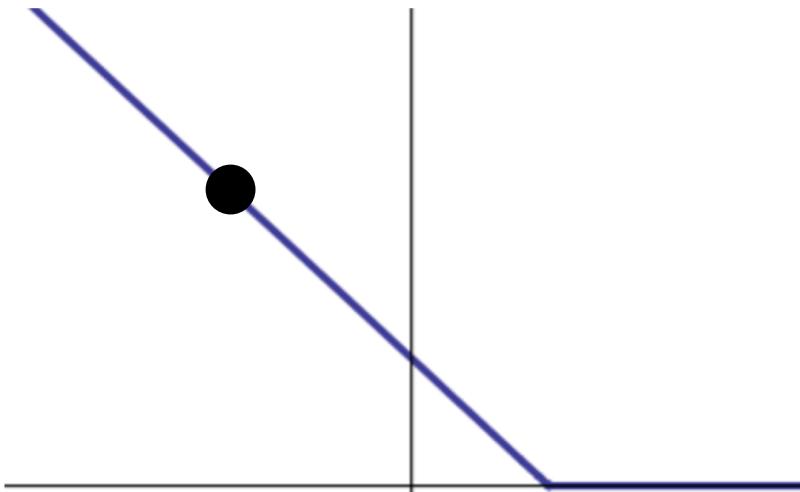
This means the classifier made an underestimate, so perceptron makes the weights larger.

The derivative here is $-y_i x_{ij}$. If x_{ij} is positive, dL_i/w_j will be negative when y_i is positive, so the gradient descent update will be positive.

Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$

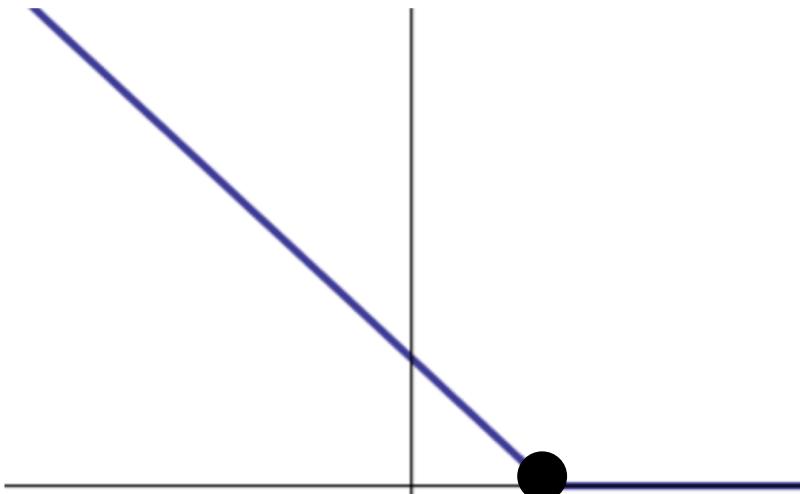


The derivative here is $-y_i x_{ij}$. If x_{ij} is positive, dL_i/w_j will be positive when y_i is negative, so the gradient descent update will be negative.

Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$

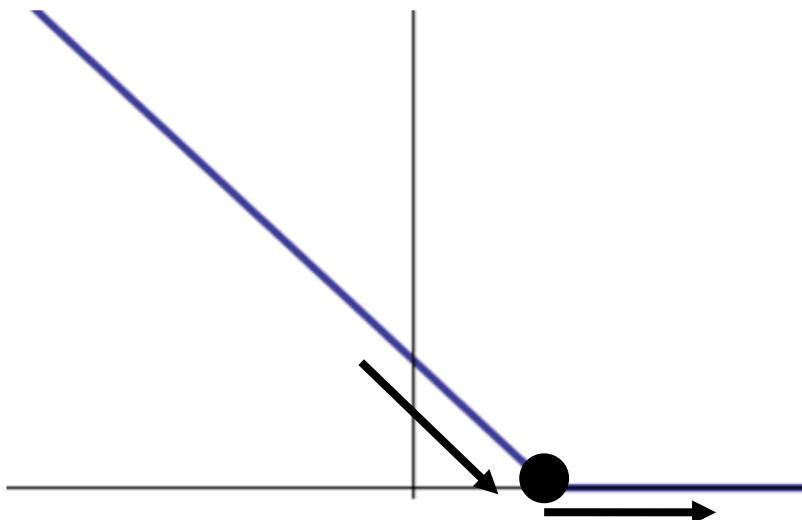


The derivative doesn't actually exist at this point (the function isn't smooth)

Revisiting Perceptron

Perceptron has a different loss function:

$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$



A **subgradient** is a generalization of the gradient for points that are not differentiable.

0 and $-y_i x_{ij}$ are both valid subgradients at this point.

Revisiting Perceptron

Perceptron has a different loss function:

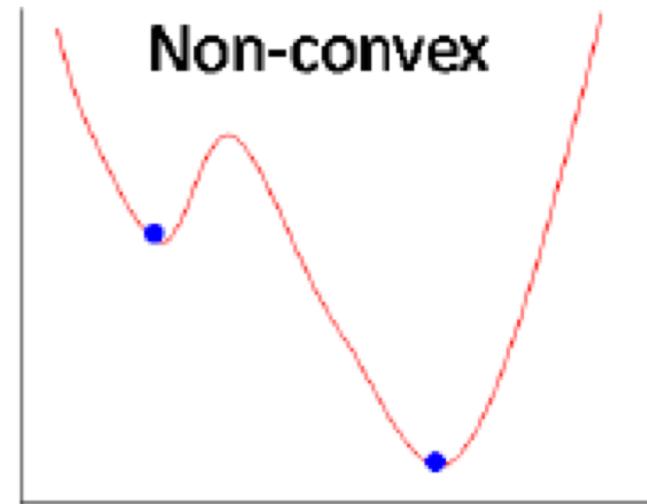
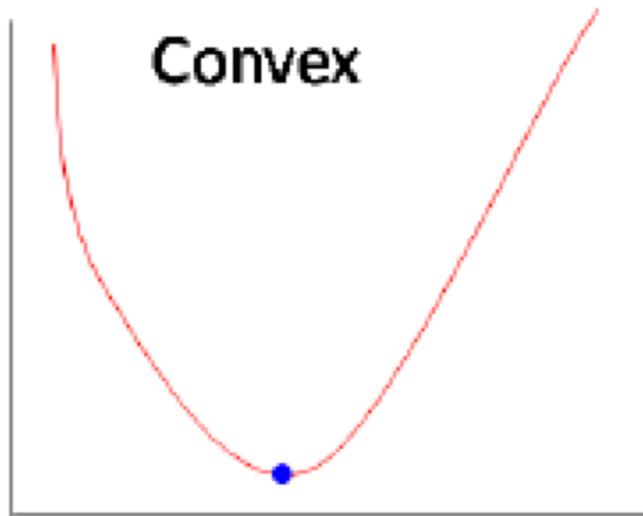
$$L_i(\mathbf{w}; \mathbf{x}_i) = \begin{cases} 0, & y_i (\mathbf{w}^\top \mathbf{x}_i) \geq 0 \\ -y_i (\mathbf{w}^\top \mathbf{x}_i), & \text{otherwise} \end{cases}$$

Perceptron is a stochastic gradient descent algorithm using this loss function (and using the subgradient instead of gradient)

Convexity

How do you know if you've found the global minimum, or just a local minimum?

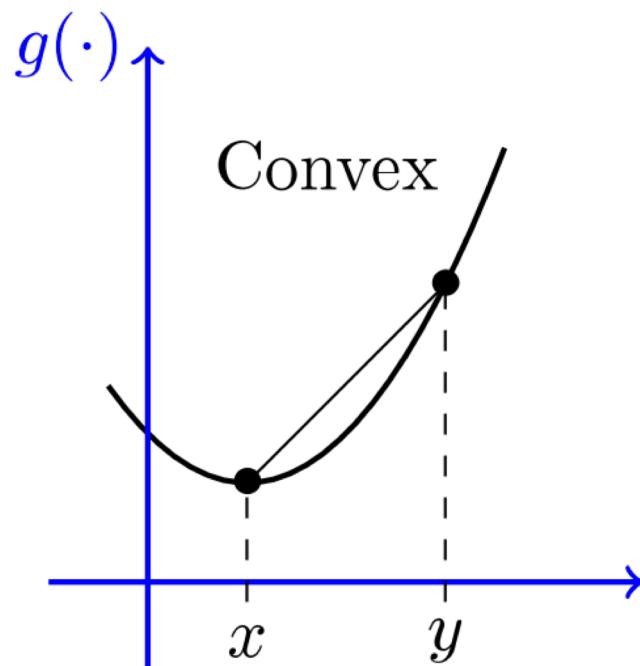
A **convex** function has only one minimum:



Convexity

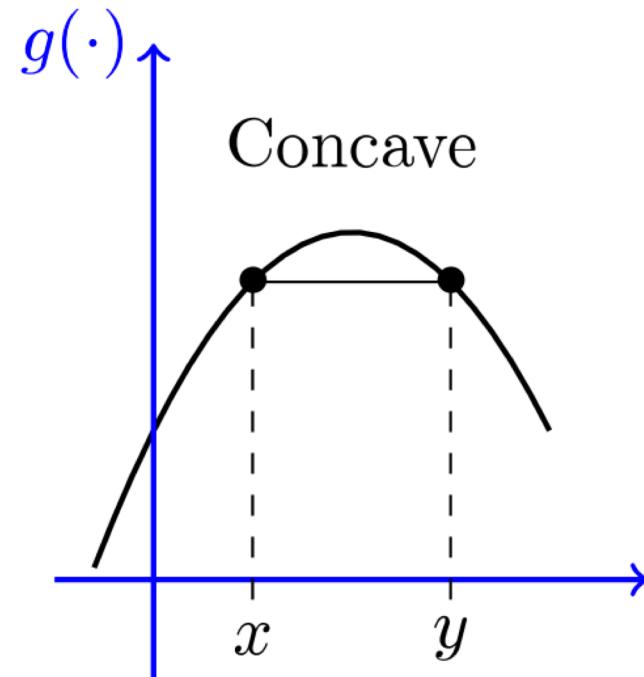
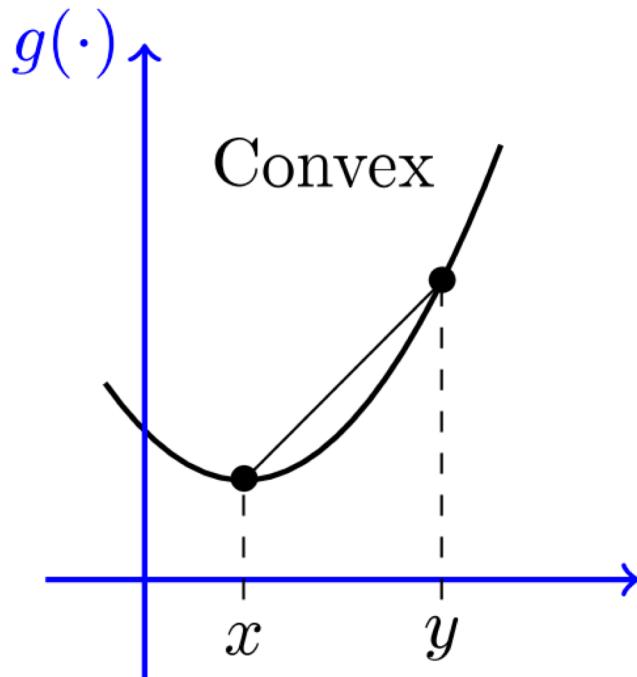
How do you know if you've found the global minimum, or just a local minimum?

A **convex** function has only one minimum:



Convexity

A **concave** function has only one maximum:

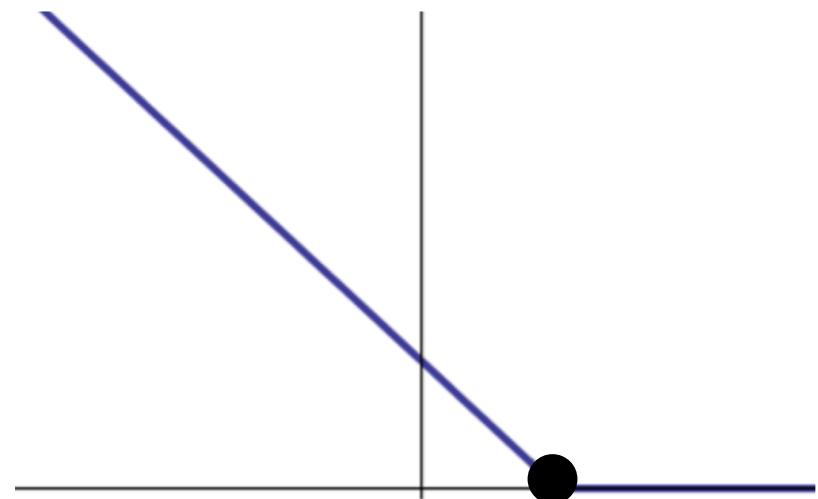


Sometimes people use “convex” to mean either convex or concave

Convexity

Squared error is a convex loss function, as is the perceptron loss.

Note: convexity means there is only one minimum value, but there may be multiple parameters that result in that minimum value.



Summary

Most machine learning algorithms are some combination of a loss function + an algorithm for finding a local minimum.

- Gradient descent is a common minimizer, but there are others.

With most of the common classification algorithms, there is only one global minimum, and gradient descent will find it.

- Most often: supervised functions are convex, unsupervised functions are non-convex.

Summary

1. Initialize the parameters \mathbf{w} to some guess
(usually all zeros, or random values)
2. Update the parameters:
$$\mathbf{w} = \mathbf{w} - \eta \nabla L(\mathbf{w})$$
$$\eta = c_1 / (t^a + c_2)$$
3. Repeat step 2 until $\|\nabla L(\mathbf{w})\| < \theta$ or until the maximum number of iterations is reached.