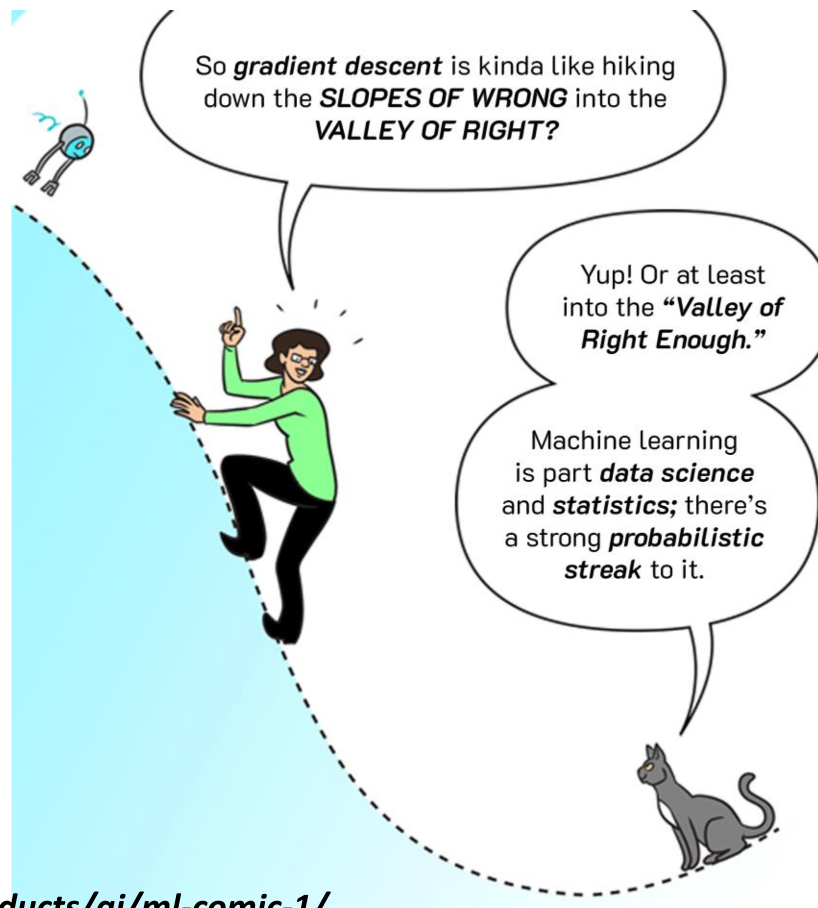
Gradient Descent

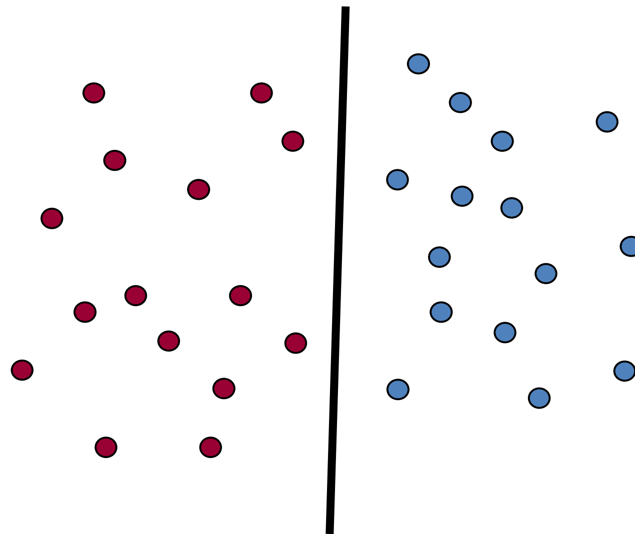


https://cloud.google.com/products/ai/ml-comic-1/

Linear models

- A high-bias assumption is *linear separability*:
  in 2 dimensions, can separate classes by a line
  in higher dimensions, need hyperplanes

- A *linear model* is a model that assumes the data is linearly separable



Linear models

- A linear model in *n*-dimensional space (i.e. *n* features) is defined by *n*+1 weights:

- In two dimensions, a line:

$$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

- In three dimensions, a plane:

$$f(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

- In *m*-dimensions, a *hyperplane*

$$f(x_1, \ldots x_m) = b + \sum_{j=1}^{m} w_j x_j$$

Linear models

- A linear model in *n*-dimensional space (i.e. *n* features) is defined by *n*+1 weights:

$$f(x_1,..x_m) = b + \sum_{j=1}^{m} w_j x_j$$

- Or equivalently $f(X) = W^T X + b$

Where:
X is the feature vector $X = [x_1..x_m]$

W is the weight vector $W = [w_1,..w_m]$

Perceptron

- Perceptron is an algorithm for binary classification that uses a linear prediction function:

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

- W: weights or slope
- b: intercept or bias

Perceptron

- The objective is to learn the **weights**

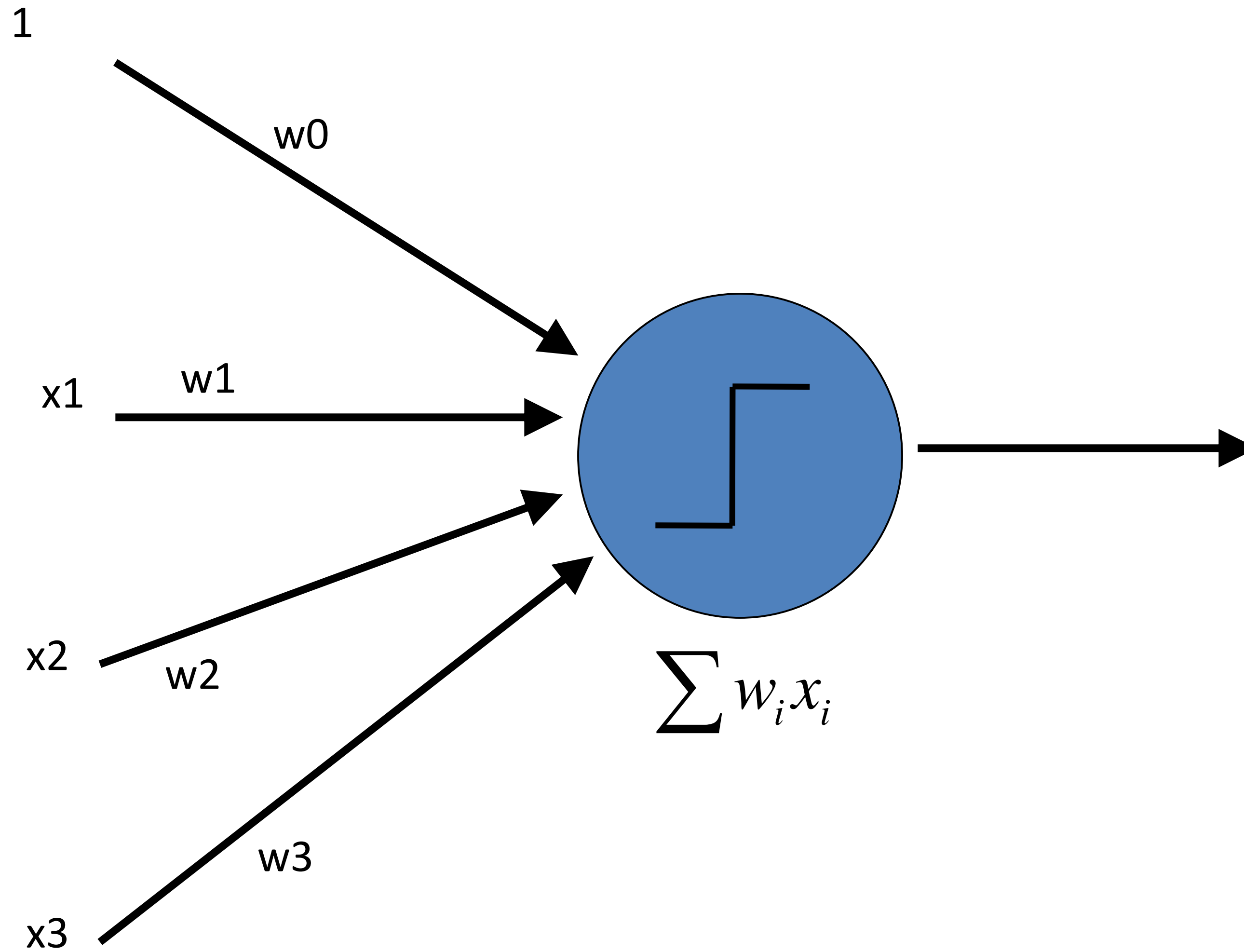
$$f(X) = \begin{cases} 1, & W^T X + b \geq 0 \\ -1, & W^T X + b < 0 \end{cases}$$

- **What about bias b?**

Common implementation $W^T X + b = W^T X + b * 1$

- If we add an extra feature to every instance whose value is always 1, then we can simply write this as $W^T X$, where the last feature weight is the value of the bias.
- Then we can update this parameter the same way as all the other weights.

Perceptron



Learning the weights

The perceptron algorithm learns the weights by:

1. Initialize all weights **w to 0**

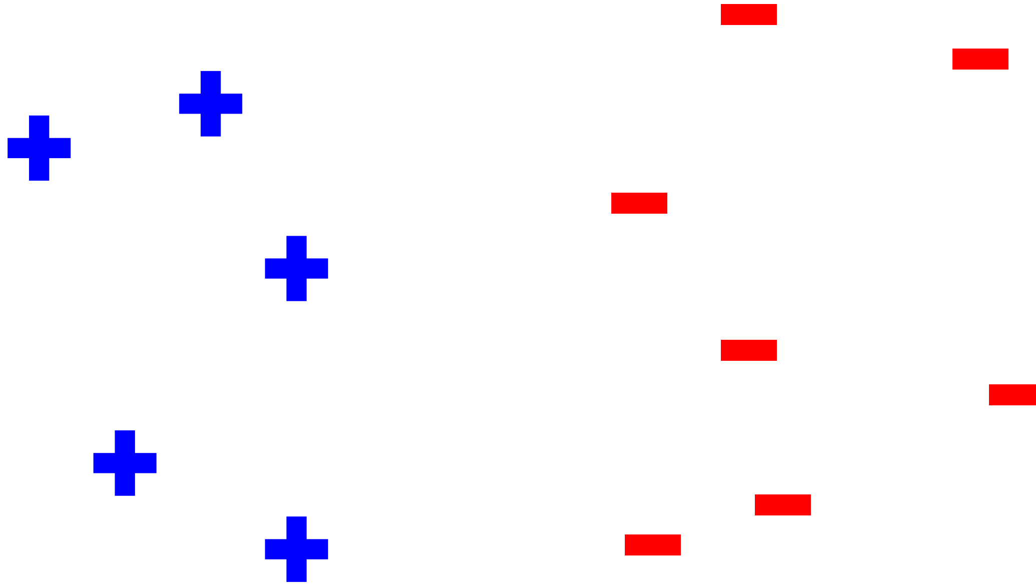
2. Iterate through the training data. For each training instance, classify the instance:
   a) If the prediction (the output of the classifier) was correct, don't do anything. (It means the classifier is working, so leave it alone!)
   b) If the prediction was wrong, modify the weights by using the update rule: $w_i = w_i + x_i$*label

3. **Repeat step 2** some number of times or until convergence (no errors, more update)

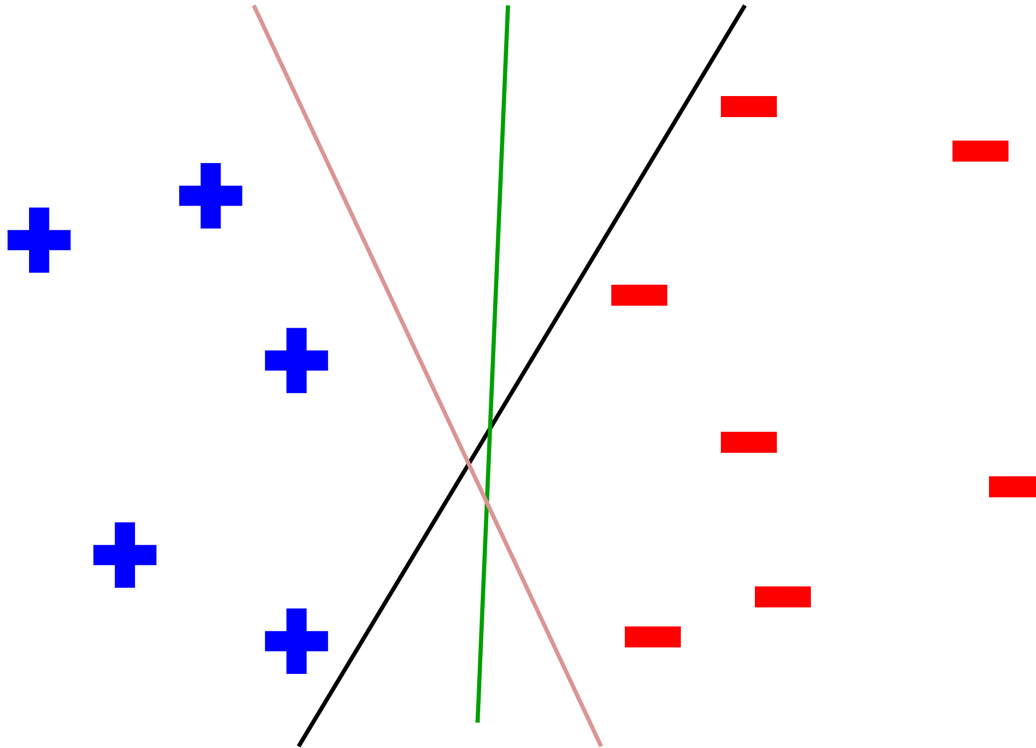
Linear Separability

- If the training instances are not linearly separable, the classifier will always get some predictions wrong.
- You need to implement some type of **stopping criteria** for when the algorithm will stop making updates, or it will run forever.
- Usually this is specified by running the algorithm for a maximum number of iterations or **epochs**.

Which line will it find?



Which line will it find?



Only guaranteed to find ***some*** line that separates the data

Model-based Machine Learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

What are the parameters for DT? Perceptron?

Model-based Machine Learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

DT: the structure of the tree, which features each node splits on, the predictions at the leaves

perceptron: the weights and the b value

Model-based Machine Learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

2. pick a criterion to optimize (aka objective function)

What criteria do decision tree learning and perceptron learning optimizing?

Model-based Machine Learning

1. pick a model
   - e.g. a hyperplane, a decision tree,…
   - A model is defined by a collection of parameters

2. pick a criterion to optimize (aka objective function)
   - e.g. training error

3. develop a learning algorithm
   - the algorithm should try and minimize the criteria
   - sometimes in a heuristic way (i.e. non-optimally)
   - sometimes exactly

Model-based Machine Learning

1. pick a model

$$f(X) = W^T X + b$$

These are the parameters we want to learn

2. pick a criterion to optimize (aka objective function)

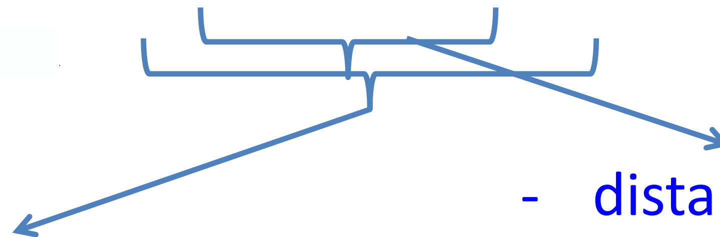
$$\sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \le 0\right]$$

$$1[x] = \begin{cases} 1 & \text{if } x = True \\ 0 & \text{if } x = False \end{cases}$$

What does this equation say?

0/1 loss function

$$L(W,b) = \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$



- distance from hyperplane
- sign is prediction

whether or not the prediction and label agree, true if ***they don't***

total number of mistakes, aka 0/1 loss

Model-based Machine Learning

1. pick a model

$$f(X) = W^T X + b$$

2. pick a criteria to optimize (aka objective function)

$$L(W,b) = \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$

3. develop a learning algorithm

$$\underset{w,b}{\arg\min} \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$

Find w and b that minimize the 0/1 loss (i.e. training error)

Minimizing 0/1 loss

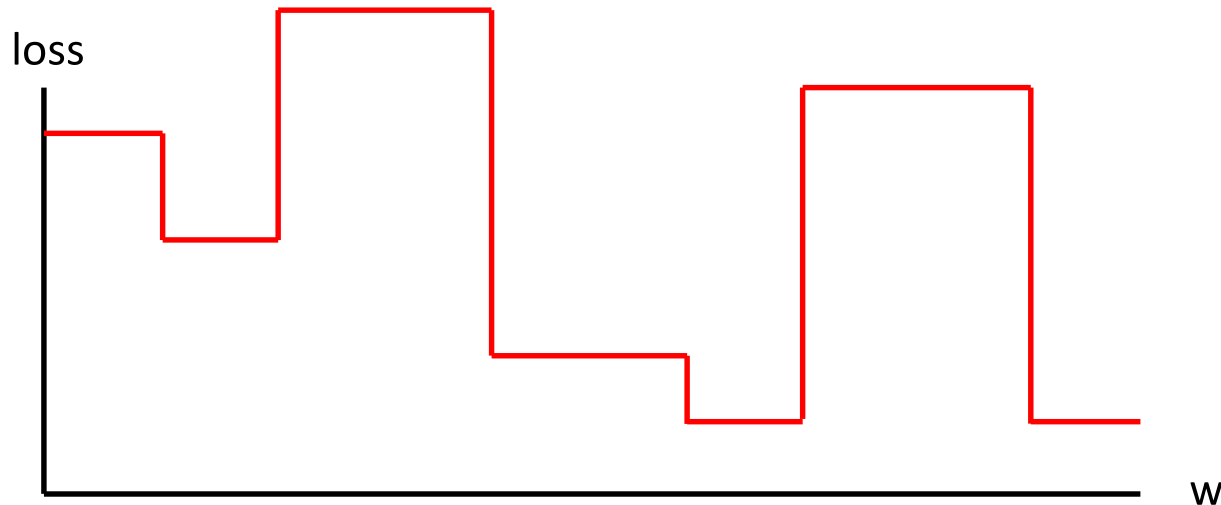
$$\arg\min_{w,b} \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$

Find w and b that minimize the 0/1 loss

How do we do this?
How do we *minimize* a function?
Why is it hard for this function?

Minimizing 0/1 in one dimension

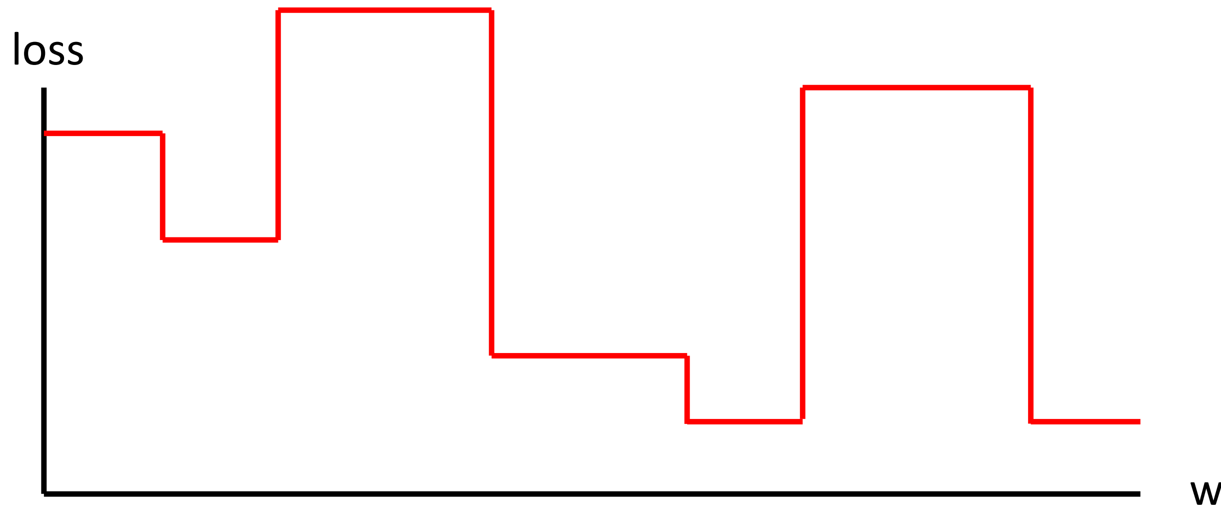
$$L(W,b) = \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$



Each time we change w such that the example is right/wrong the loss will increase/decrease

Minimizing 0/1 over all w

$$L(W,b) = \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$



Each new feature we add (i.e. weights) **adds another dimension to this space!**

Minimizing 0/1 loss

$$\arg\min_{w,b} \sum_{i=1}^{n} 1\left[y_i(w \cdot x_i + b) \leq 0\right]$$

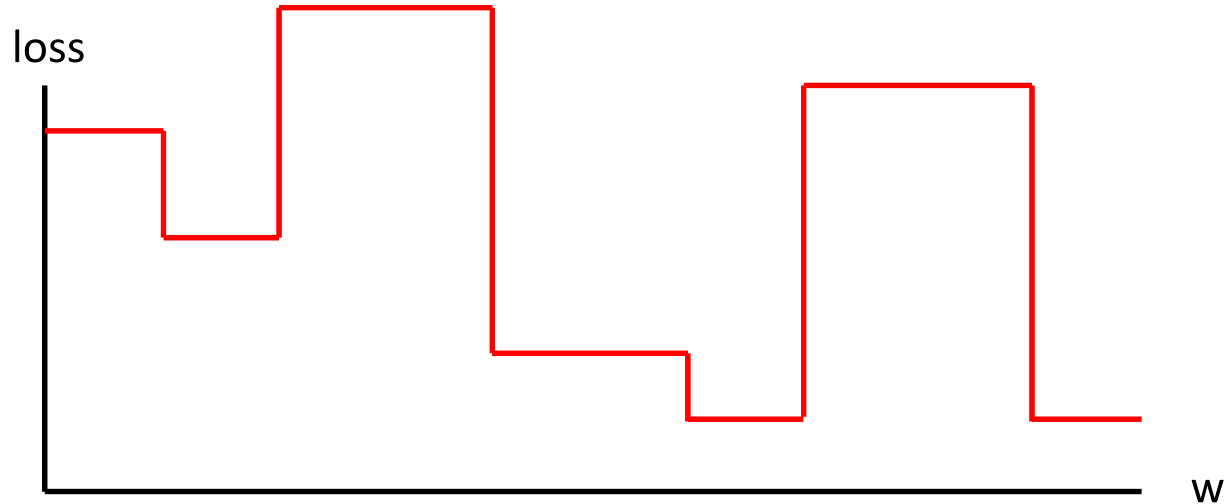
Find w and b that minimize the 0/1 loss

This turns out to be hard (in fact, NP-HARD)

Challenge:

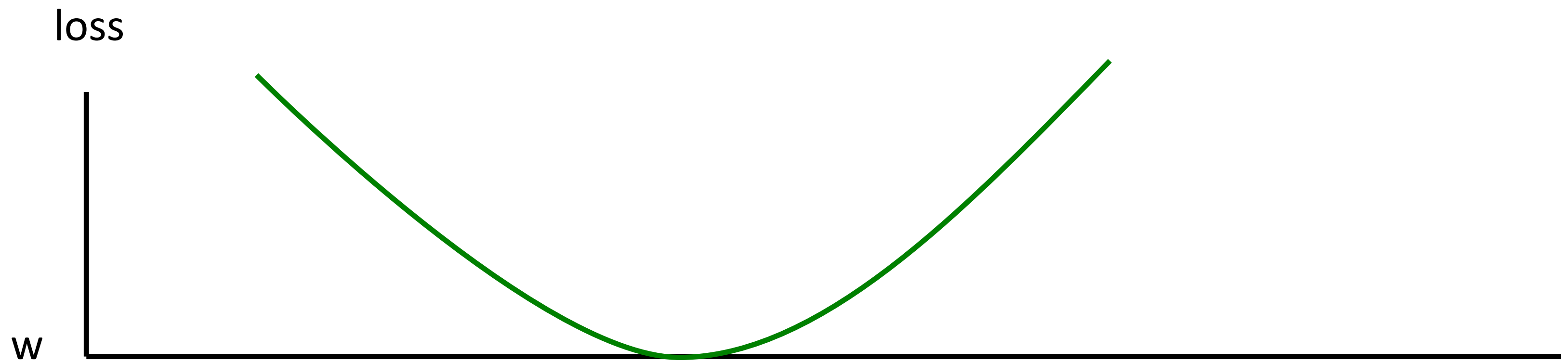
- small changes in any w can have large changes in the loss (the change isn't continuous)
- there can be many, many local minima
- at any given point, we don't have much information to direct us towards any minima

More manageable loss functions



What property/properties do we want from our loss function?

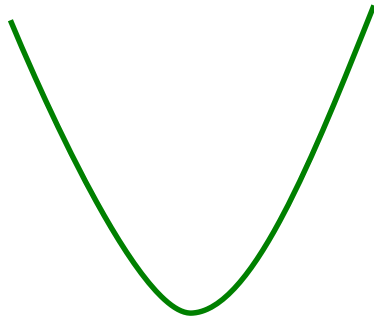
More manageable loss functions

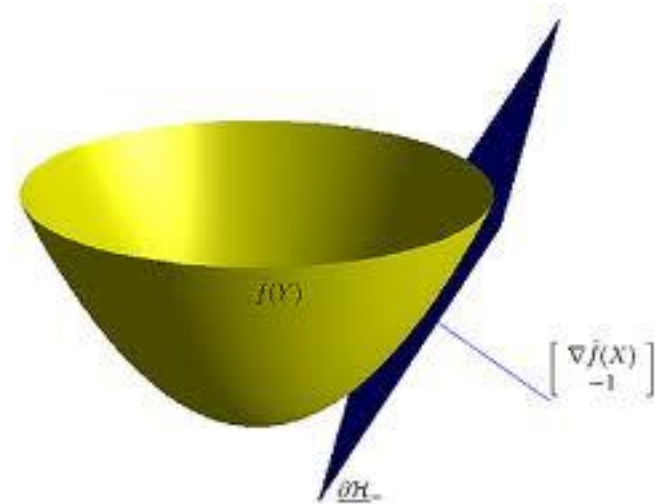


- Ideally, continuous (i.e. differentiable) so we get an indication of direction of minimization
- Only one minima

Convex functions

Convex functions look something like:





One definition: The line segment between any two points on the function is *above* the function

Surrogate loss functions

- For many applications, we really would like to minimize the 0/1 loss
- A surrogate loss function is a loss function that provides an upper bound on the actual loss function (in this case, 0/1)
- We'd like to identify **convex surrogate loss functions** to make them easier to minimize
- Key to a loss function: how it scores the difference between the actual label ***y*** and the predicted label ***y'***

Surrogate loss functions

0/1 loss: $$l(y, y') = 1\left[yy' \le 0 \right]$$

Ideas?
Some function that is a proxy for error, but is continuous and convex

Surrogate loss functions

0/1 loss: $l(y, y') = 1[yy' \leq 0]$

Hinge: $l(y, y') = \max(0, 1 - yy')$

Exponential: $l(y, y') = \exp(-yy')$

Squared loss: $l(y, y') = (y - y')^2$

Why do these work? What do they penalize?

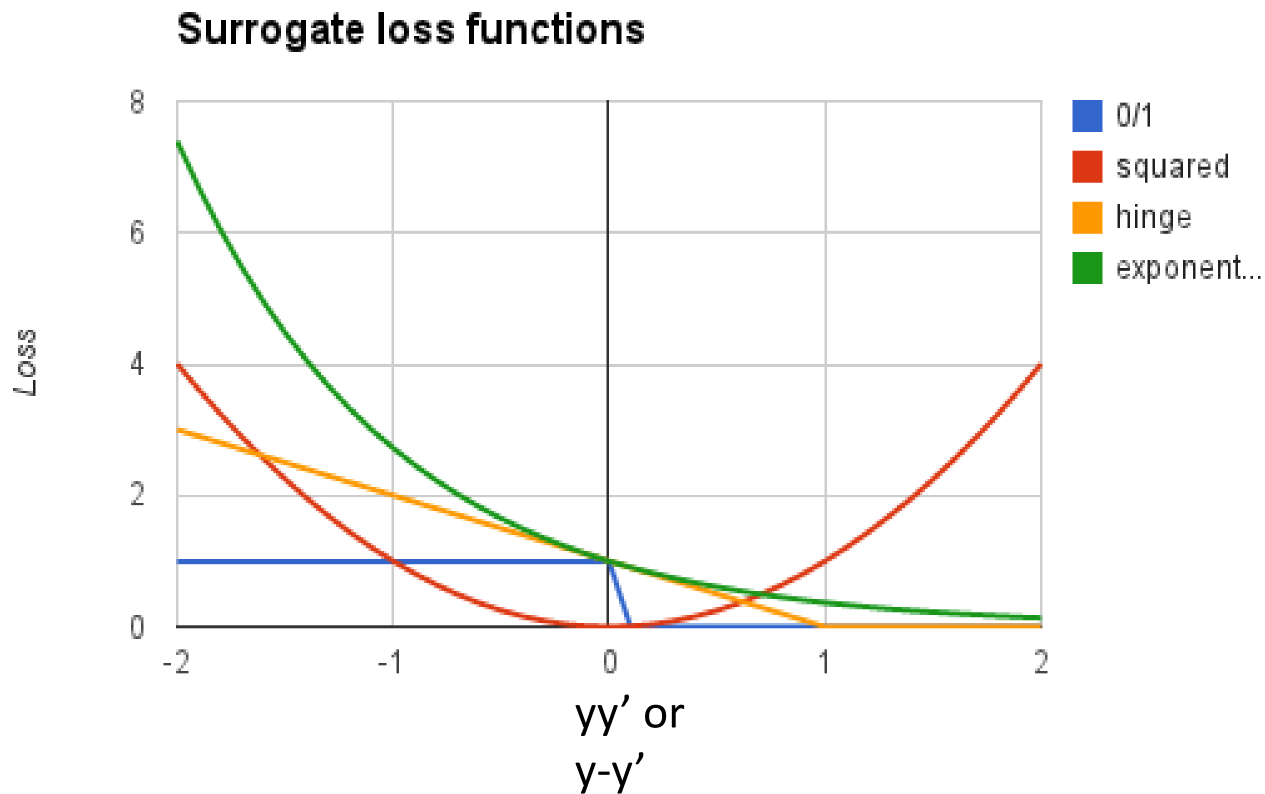
Surrogate loss functions

0/1 loss: $l(y, y') = 1[yy' \leq 0]$

Hinge: $l(y, y') = \max(0, 1 - yy')$

Squared loss: $l(y, y') = (y - y')^2$

Exponential: $l(y, y') = \exp(-yy')$



Model-based Machine Learning

1. pick a model

$$f(X) = W^T X + b$$

2. pick a criteria to optimize (aka objective function)

$$L(W,b) = \sum_{i=1}^{n} \exp\left(-y_i(w \cdot x_i + b)\right)$$

use a convex surrogate loss function

3. develop a learning algorithm

$$\arg\min_{w,b} \sum_{i=1}^{n} \exp\left(-y_i(w \cdot x_i + b)\right)$$

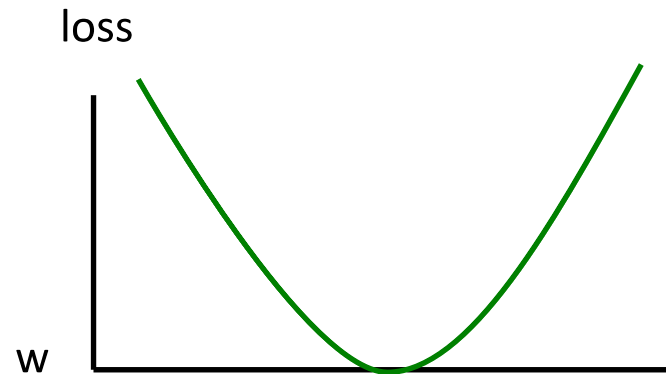
Find w and b that minimize the surrogate loss

Finding the minimum

You're blindfolded, but you can see out of the bottom of the blindfold to the ground right by your feet. I drop you off somewhere and tell you that you're in a convex shaped valley and escape is at the bottom/minimum. How do you get out?

Finding the minimum

How do we do this for a function?

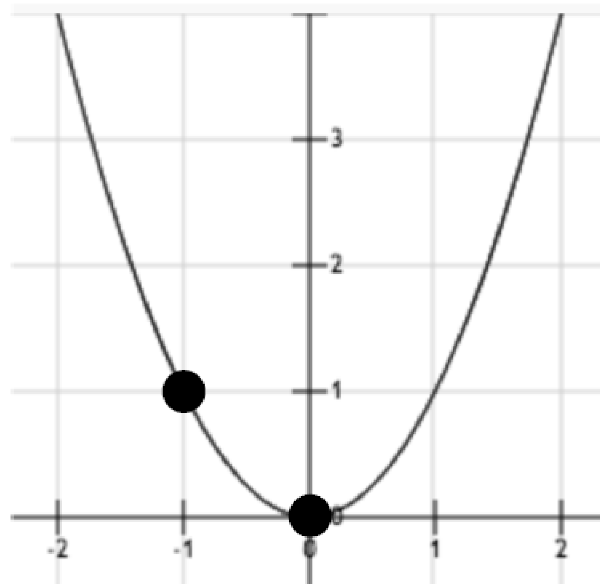


Goal is to minimize loss function.
How do we minimize a function?
Let's review some math.

Rate of Change

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

$f(x) = x^2$

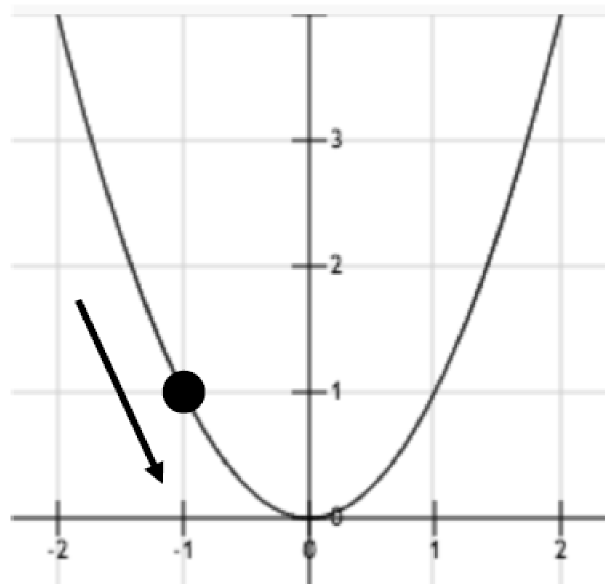


Average slope from
x=-1 to x=0 is:
-1

Rate of Change

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

$f(x) = x^2$

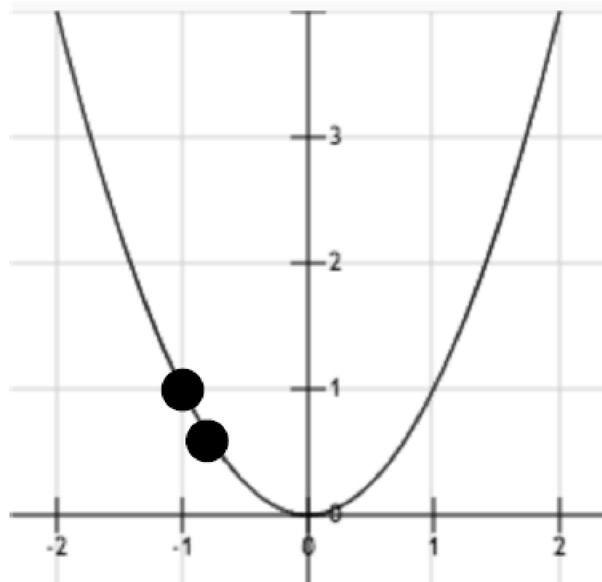


Slope at x=-1 is:
-2

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

Maxima and Minima

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

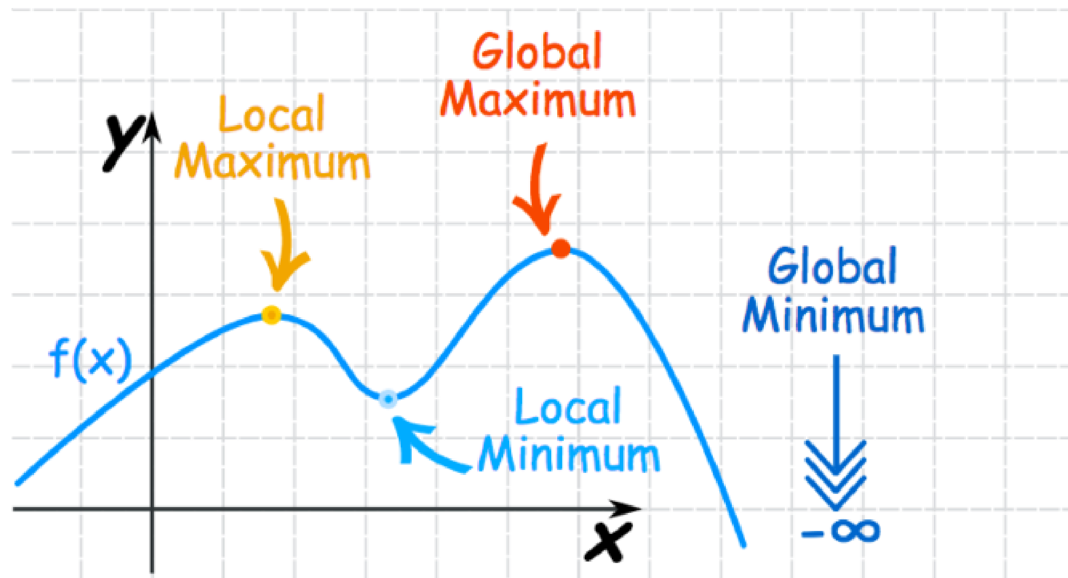
- The **global maximum** is the highest peak in the entire data set, or the largest f(x) value the function can output

- 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



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:
  - $f'(x) = 2x$
  - $d/dx\ [x^2] = 2x$
  - $df/dx = 2x$

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

- The rate of change is different at different points

Derivatives

- What if a function has multiple arguments?

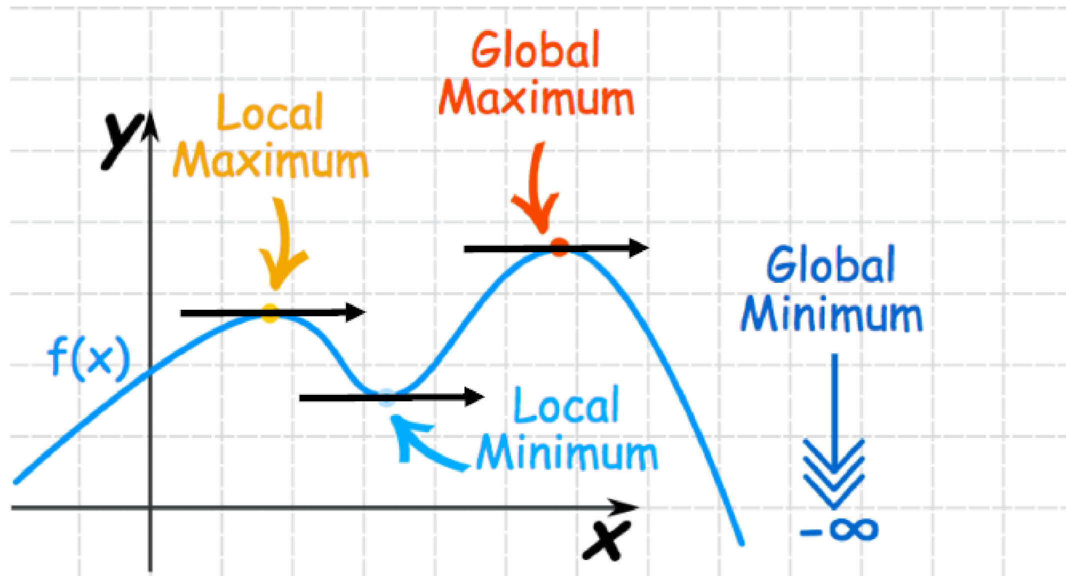
- Ex: $f(x_1, x_2) = 3x_1 + 5x_2$
  - $df/dx_1 = 3 + 5x_2$ The derivative “with respect to” x1
  - $df/dx_2 = 3x_1 + 5$ The derivative “with respect to” x2

- These two functions are called partial derivatives.

- The vector of all partial derivatives for a function f is called the gradient of the function:
  - $\nabla f(x1, x2) = < df/dx1 , df/dx2 >$

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.

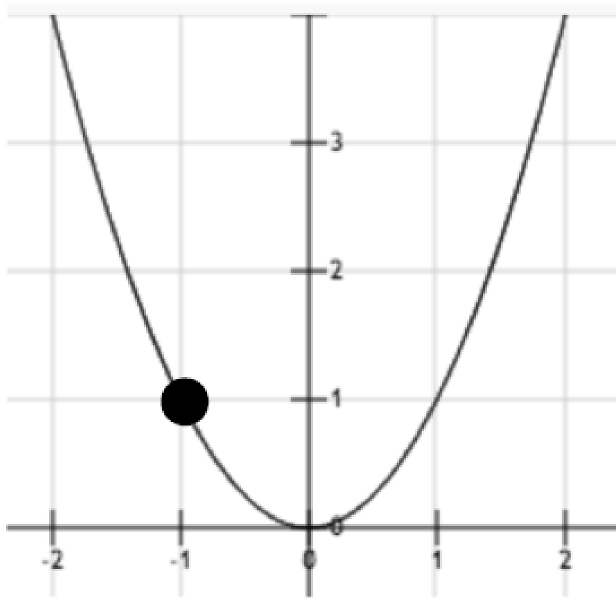
- One way to find a minimum: set $f'(x)=0$ and solve for x.
  - For most functions, there isn't a way to solve this.
  - Instead: algorithmically search different values of x until you find one that results in a gradient near 0.

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



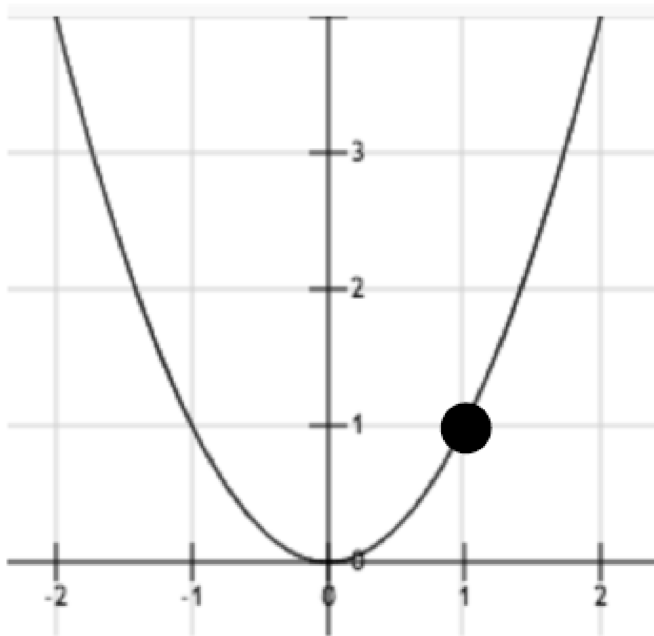
$f'(-1) = -2$

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:

$$-1 + 2 = 1$$

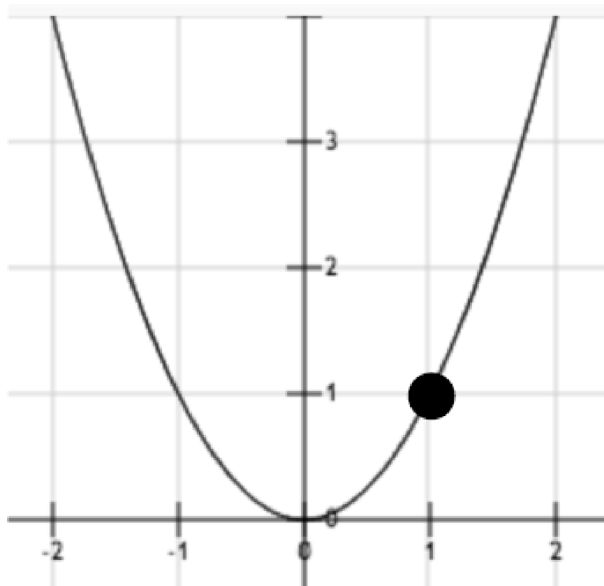
Finding Minima



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:

$$-1 + 2 = 1$$

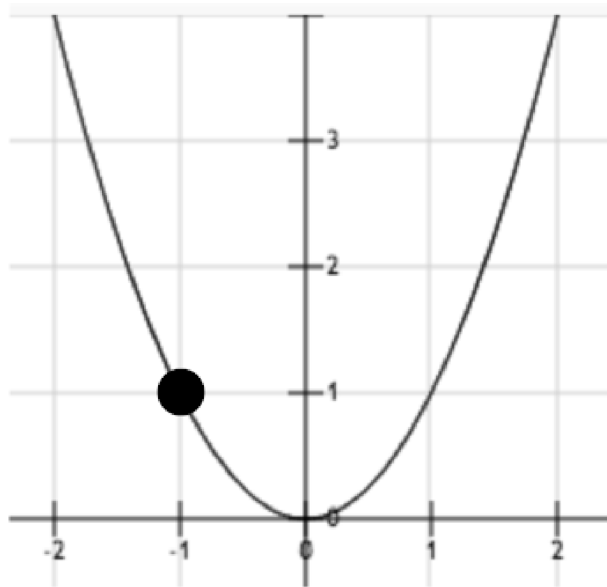
Finding Minima



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:

$$1 - 2 = -1$$

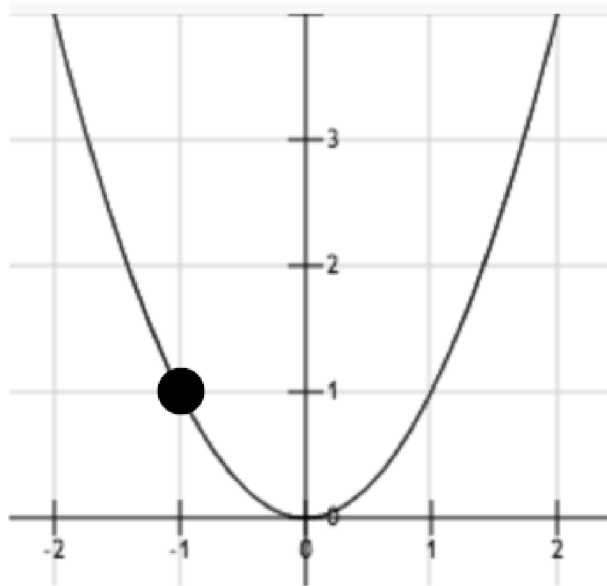
Finding Minima



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

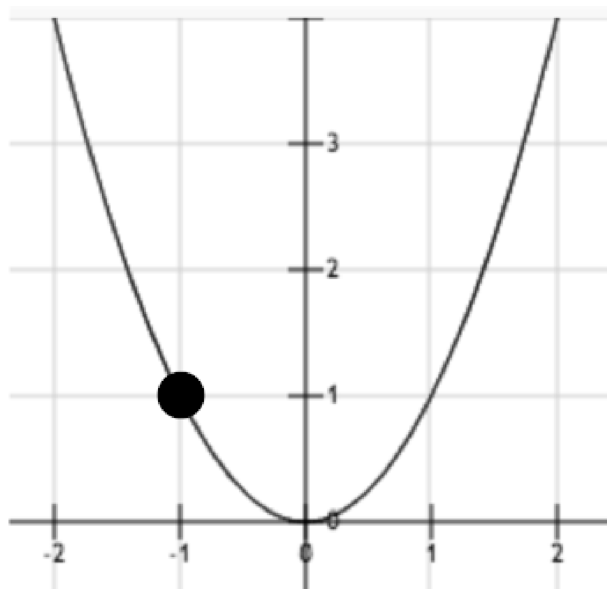
Finding Minima



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

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

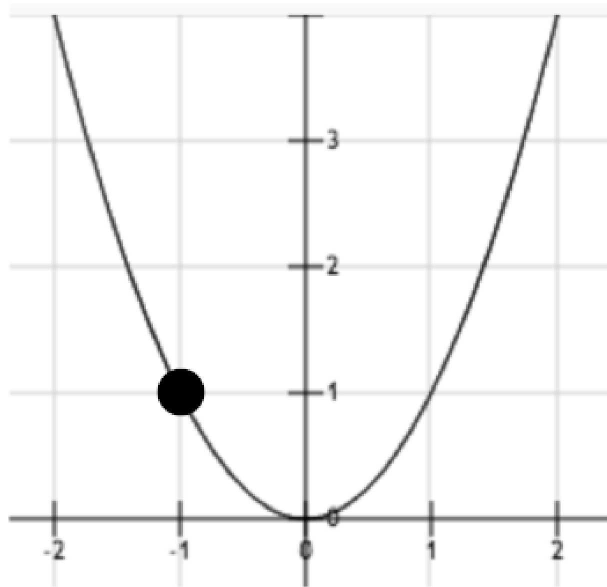
Finding Minima



$f'(-1) = -2$

$x \mathrel{+}= 2\eta =$

Finding Minima

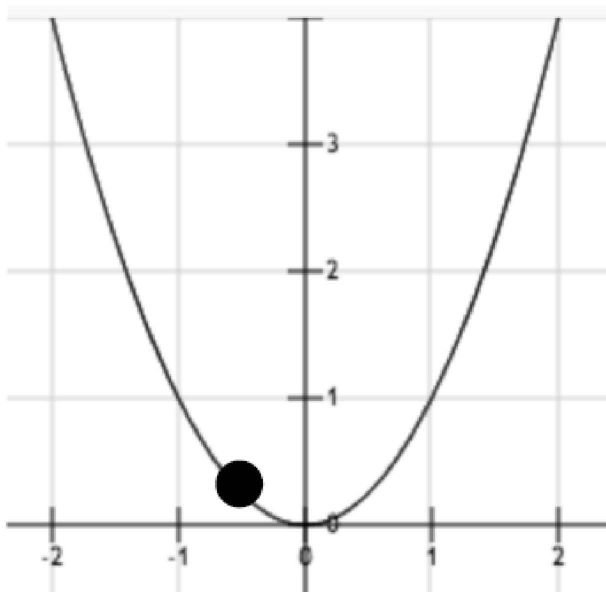


$f'(-1) = -2$

$x \mathrel{+}= 2\eta =$

Let's use $\eta = 0.25$.

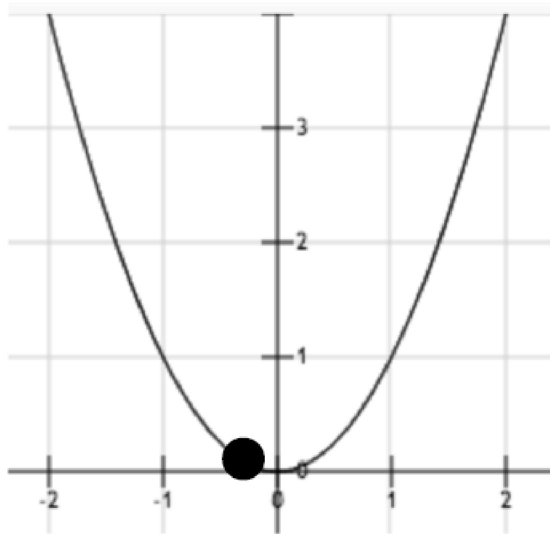
Finding Minima



$f'(-1) = -2$

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

Finding Minima



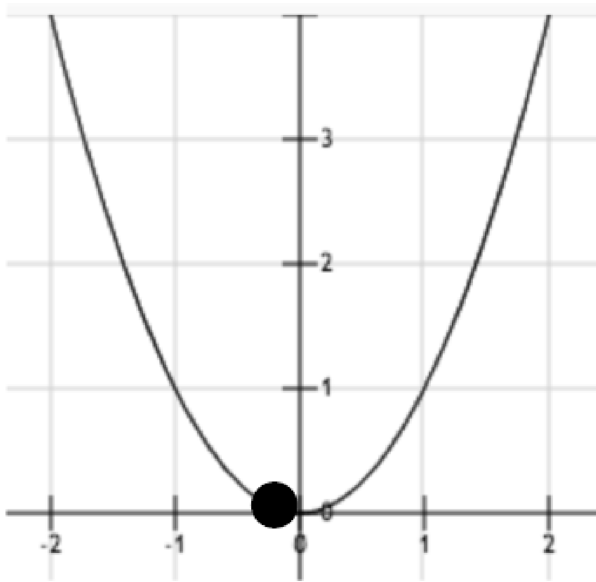
$f'(-1) = -2$

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

$f'(-0.5) = -1$

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

Finding Minima



f'(-1) = -2

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

f'(-0.5) = -1

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

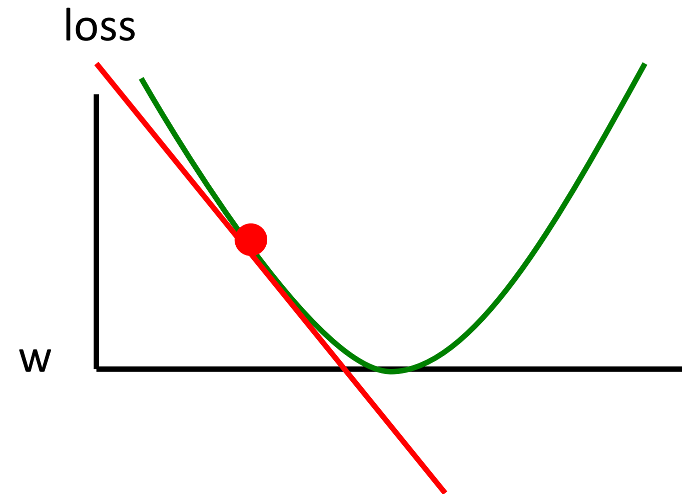
f'(-0.25) = -0.5

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

Eventually we'll reach x=0.

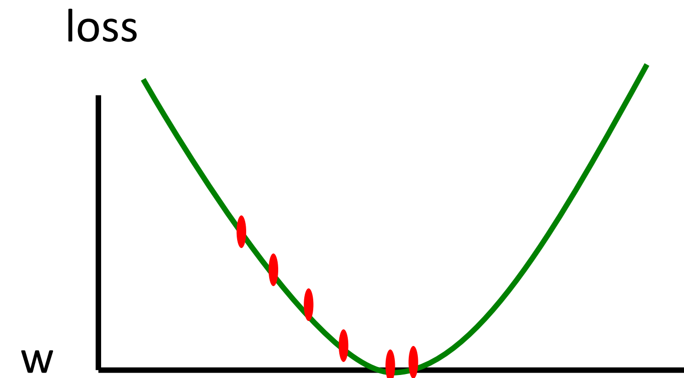
One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension



One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension

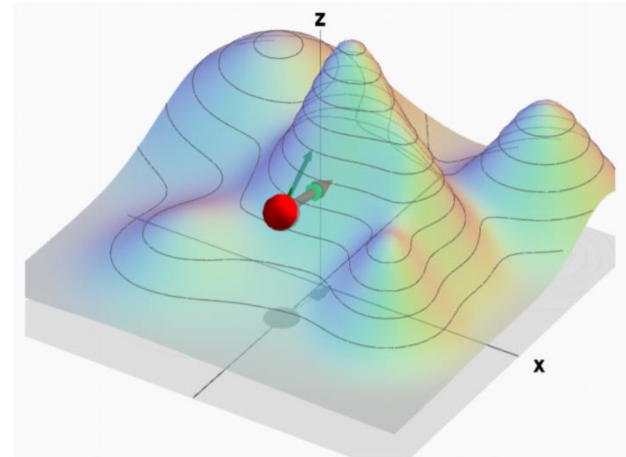


Approach:

- pick a starting point (w)
- repeat :
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

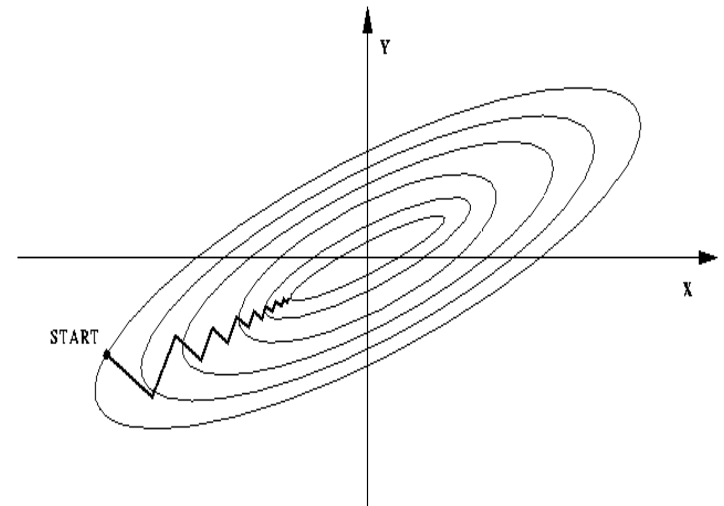
One approach: gradient descent

Partial derivatives give us the slope (i.e. direction to move) in that dimension



Approach:

- pick a starting point (w)
- repeat:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)



Gradient descent

- pick a starting point (w)
- repeat until convergence (loss doesn't decrease in any dimension :

$$w_j = w_j - \eta \frac{d}{dw_j} loss(w)$$

What does this do?

learning rate or step size (how much we want to move in the error direction)

Gradient descent

- pick a starting point (w)
- repeat until convergence:

$$w_j = w_j - \eta \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b))$$

This is called standard or batch gradient descent because the gradient is computed using the whole dataset

Gradient descent

- pick a starting point (w)
- repeat until convergence:

$$w_j = w_j - \eta \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b))$$

$$w_j = w_j + \eta \sum_{i=1}^{n} y_i x_{ij} \exp\left(-y_i(w \cdot x_i + b)\right)$$

What is this doing?

Exponential update rule

$$w_j = w_j + \eta \sum_{i=1}^{n} y_i x_{ij} \exp\left(-y_i(w \cdot x_i + b)\right)$$

If the update is done for each example x_i → **Stochastic gradient descent**

$$w_j = w_j + \eta\, y_i\, x_{ij} \exp\left(-y_i(w \cdot x_i + b)\right)$$

Does this look familiar?

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, …, f_m, label):

$$\text{prediction} = b + \sum_{j=1}^{m} w_j f_j$$

if *prediction* * *label* ≤ 0: // they don't agree

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

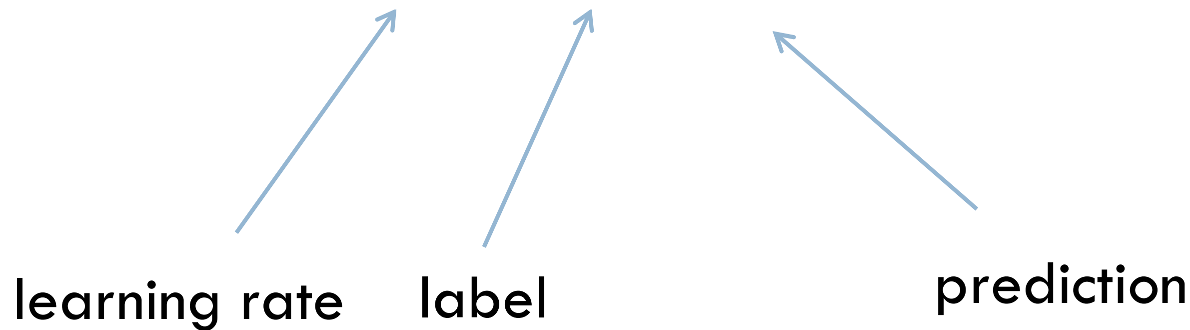
$$w_j = w_j + \eta\, y_i\, x_{ij} \exp\left(-y_i(w \cdot x_i + b)\right)$$

or

$$w_j = w_j + x_{ij} \times y_i \times c$$ where $$c = \eta \exp\left(-y_i(w \cdot x_i + b)\right)$$

The constant

$$c = \eta \exp\left(-y_i(w \cdot x_i + b)\right)$$



When is this large/small?

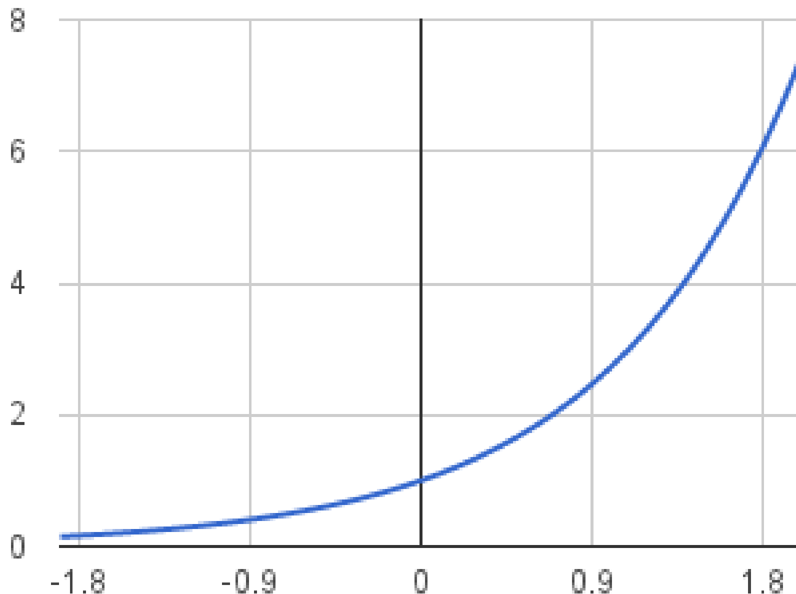
$$c = \eta \exp\left(-y_i(w \cdot x_i + b)\right)$$

The constant

$$c = \eta \exp\left(-y_i(w \cdot x_i + b)\right)$$

label

prediction



If they're the same sign, as the predicted gets larger the update gets smaller

If they're different, the more different they are, the bigger the update

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example (f_1, f_2, …, f_m, label):

$$\text{prediction} = b + \sum_{j=1}^{m} w_j f_j$$

~~if *prediction* * *label* ≤ 0: // they don't agree~~

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

Note: for gradient descent, we always update

$$w_j = w_j + \eta\, y_i\, x_{ij} \exp\left(-y_i(w \cdot x_i + b)\right)$$

or

$$w_j = w_j + x_{ij} \times y_i \times c$$ where $$c = \eta \exp\left(-y_i(w \cdot x_i + b)\right)$$

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?

Stopping Criteria: Distance

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

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

This is called the Euclidean norm of p

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

Stopping Criteria: Distance

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

$$|| \frac{d}{dw_j} Loss(w) || < \theta$$

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

Gradient Descent: Summary

1. Initialize the parameters w to some guess (usually all zeros, or random values)

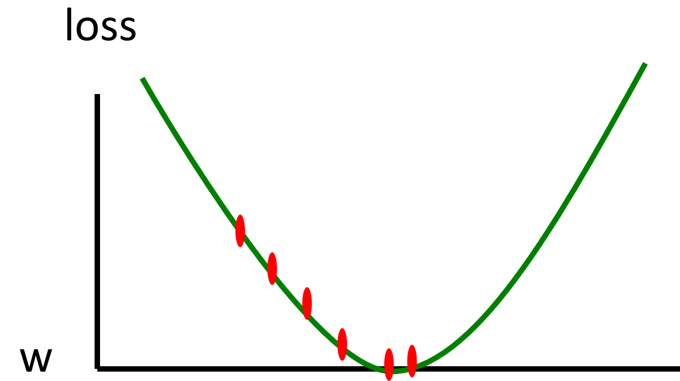
2. Update the parameters (for each weight w_j):

$$w_j = w_j - \eta \frac{d}{dw_j} Loss(w_j)$$

3. Repeat step 2 until $\| \frac{d}{dw_j} Loss(w) \| < \theta$ or until the maximum number of iterations is reached.

One concern

$$\text{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b))$$



We're calculating this on the **training set**

We still need to be careful about overfitting

The min w,b on the training set is generally NOT the min for the test set

Summary

- Model-based machine learning:
  - define a model, objective function (i.e. loss function), minimization algorithm
- Gradient descent minimization algorithm
  - require that our loss function is convex
  - make small updates towards lower losses
- Perceptron learning algorithm:
  - gradient descent
  - exponential loss function (modulo a learning rate)

Reading

- HL: Ch 7-7.5 (7.6 optional)
- MLP Ch. 2, “Adaptive linear neurons” section
- MLA 14.1-14.3

For more depth: UML Ch. 12; UML 14.4-14.5

Some math

$$\frac{d}{dw_j} loss \quad = \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i (w \times x_i + b))$$

$$= \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b)) \frac{d}{dw_j} - y_i (w \cdot x_i + b)$$

Some math

$$-\frac{d}{dw_j} y_i(w \cdot x_i + b) = -\frac{d}{dw_j} y_i\left(\sum_{j=1}^{m} w_j x_{ij} + b\right)$$

$$= -\frac{d}{dw_j} y_i(w_1 x_{i1} + w_2 x_{i2} + \ldots + w_m x_{im} + b)$$

$$= -\frac{d}{dw_j} y_i w_1 x_{i1} + y_i w_2 x_{i2} + \ldots + y_i w_m x_{im} + y_i b)$$

$$= -y_i x_{ij}$$

Some math

$$\frac{d}{dw_j} loss = \frac{d}{dw_j} \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b))$$

$$= \sum_{i=1}^{n} \exp(-y_i(w \cdot x_i + b)) \frac{d}{dw_j} - y_i(w \cdot x_i + b)$$

$$= \sum_{i=1}^{n} -y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$