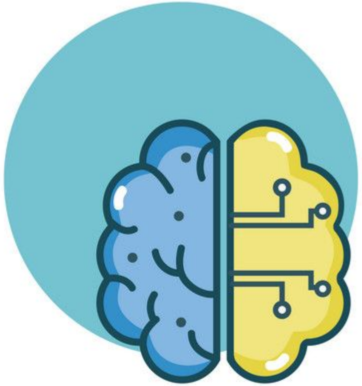
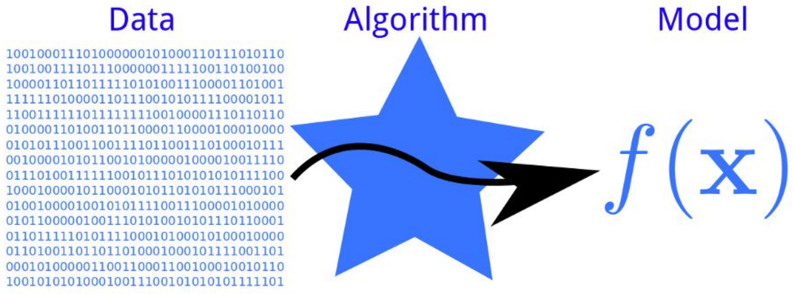
INTRODUCTION TO MACHINE LEARNING

GRADIENT DESCENT

Elisa Ricci

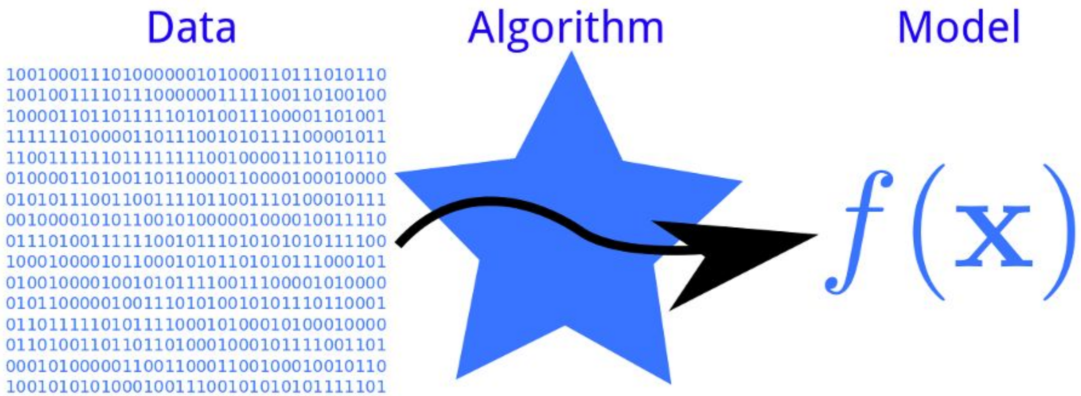




Models And Algorithms

Machine Learning Idea

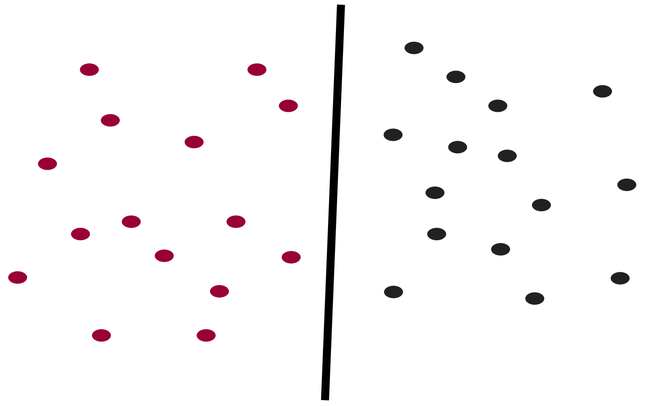
- ML allows computers to acquire knowledge.
- Knowledge is acquired through **algorithms** by learning and inferring from **data**.
- Knowledge is represented by a **model**.
- The model is used on future data.



Linear models

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

Assume a specific hypothesis space, i.e. linear functions



Linear models

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

In two dimensions, we have a line:

$$0 = w_1 f_1 + w_2 f_2 + b$$

In three dimensions, a plane:

$$0 = w_1 f_1 + w_2 f_2 + w_3 f_3 + b$$

In *n*-dimensions, a **hyperplane**

$$0 = b + \sum_{i=1}^{n} w_i f_i$$

Perceptron learning algorithm

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

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

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

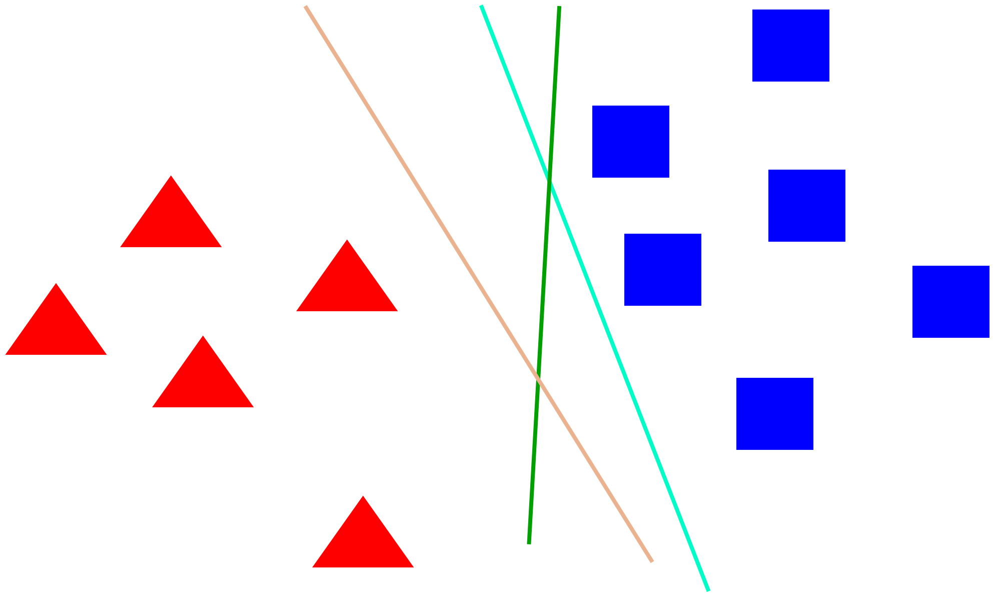
if *prediction* * *label* ≤ 0: // they don't agree

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

Which line will the perceptron find?



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

Linear models

Perceptron algorithm is one example of a linear classifier

Many, many other algorithms learn a line (i.e. a setting of a linear combination of weights)

Goals:

- Explore a number of linear training algorithms
- Understand *why these algorithms work*

Perceptron learning algorithm

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

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

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

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

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

A closer look at why we got it wrong

w_1 w_2

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$

(-1, 1) +

This value should be positive!value

contributed in the wrong direction

could have contributed (positive feature) but it did not since the weight is 0

A closer look at why we got it wrong

w_1 w_2

$$1 * f_1 + 0 * f_2 =$$

$$1 * -1 + 0 * 1 = -1$$

(-1, 1) **+**

This value should be positive!value

decrease
e.g. from 1 to 0

increase
from 0 to 1

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 optimize?

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

Linear models in general

1. Pick a model

$$0 = b + \sum_{j=1}^{m} w_j f_j$$

These are the parameters we want to learn

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

Some notation: indicator function

Convenient notation for turning True and False answers into numbers/counts:

$$1[x] = \left\{ \begin{array}{cc} 1 & if\ \ x = True \\ 0 & if\ \ x = False \end{array} \right\}$$

Some notation: dot-product

We use a vector notation

We represent an example $f_1, f_2, \ldots, f_m$ as a single vector, $\mathbf{x}$

- `j` subscript will indicate feature indexing, i.e., x_j
- `i` subscript will indicate examples indexing over a dataset, i.e., x_i or sometimes x_{ij}

Similarly, we can represent the weight $w_1, w_2, \ldots, w_m$ as a single vector, $\mathbf{w}$

The dot-product between two vectors a and b is defined as:

$$a \cdot b = \sum_{j=1}^{m} a_j b_j$$

Linear models

1. Pick a model

$$0 = b + \sum_{j=1}^{n} w_j f_j$$

These are the parameters we want to learn

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

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

What does this equation say?

0/1 loss function

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

- 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

$$0 = b + \sum_{j=1}^{m} w_j f_j$$

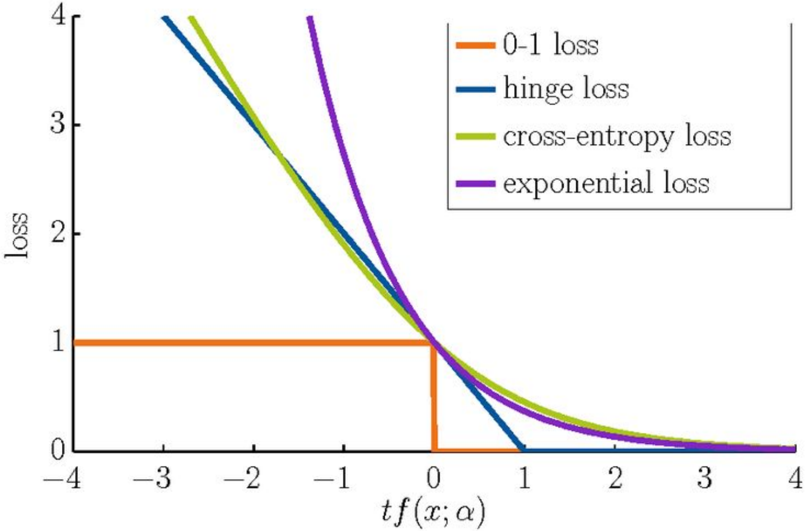
2. Pick a criteria to optimize (aka objective function)

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

3. Develop a learning algorithm

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

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



Loss Functions

Minimizing 0/1 loss

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

Find w and b that minimize the 0/1 loss

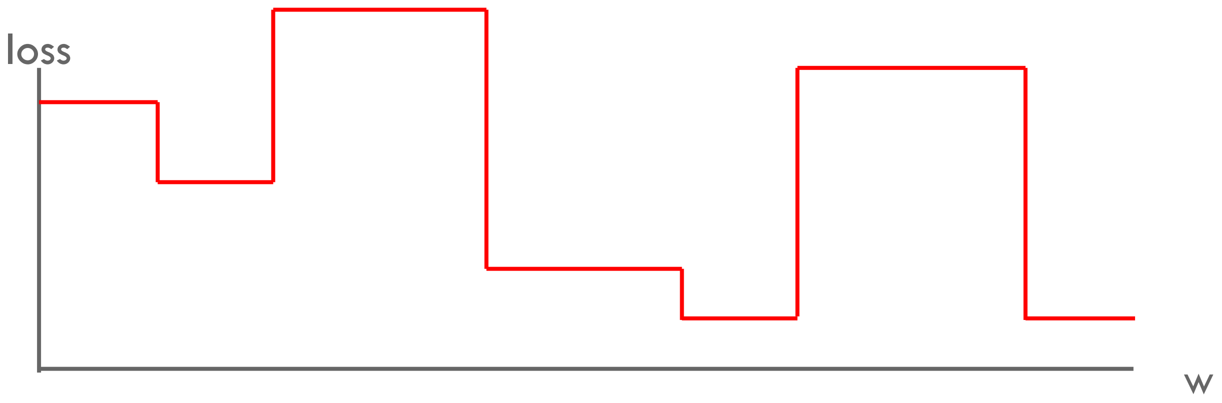
How do we do this?

How do we *minimize* a function?

Why is it hard for this function?

Minimizing 0/1 in one dimension

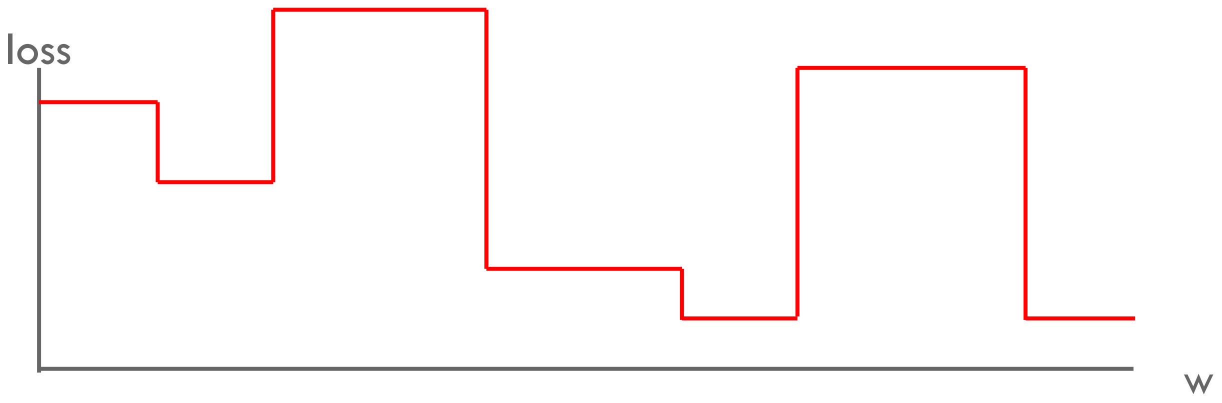
$$\sum_{i=1}^{n} 1[y_i(w \cdot x_i + b) \le 0]$$



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

Minimizing 0/1 in one dimension

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



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

Minimizing 0/1 loss

Find w and b that minimize the 0/1 loss

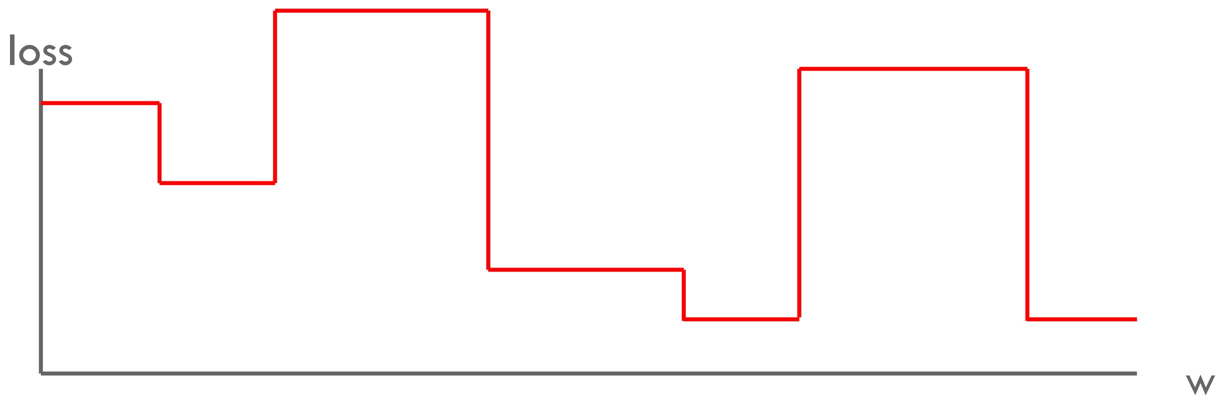
$$\text{argmin}_{w,b} \sum_{i=1}^{n} 1\left[y_i (w \cdot x_i + b) \leq 0 \right]$$

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

Challenge:

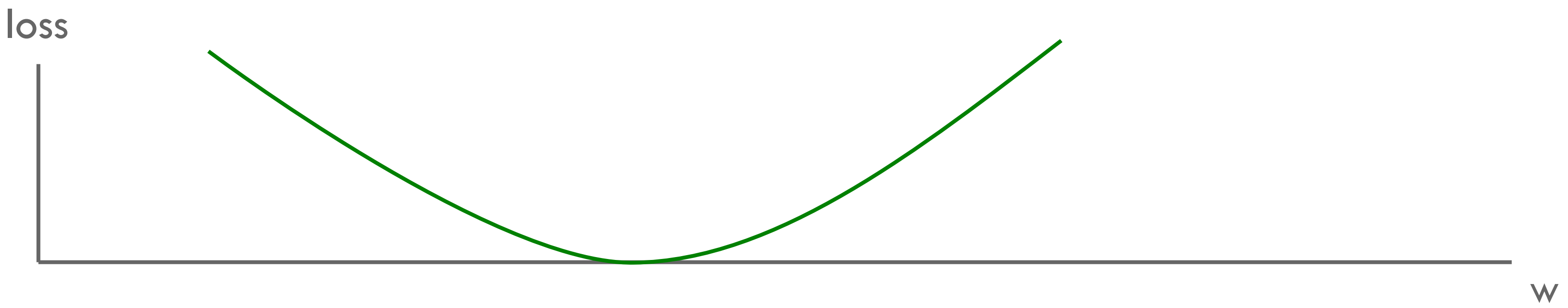
- Small changes in any w can have large changes in the loss (the change isn't continuous)
- There can be many, many local minima
- At any given point, we don't have much information to direct us towards any minima

More manageable loss functions



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

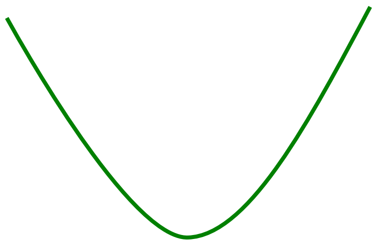
More manageable loss functions

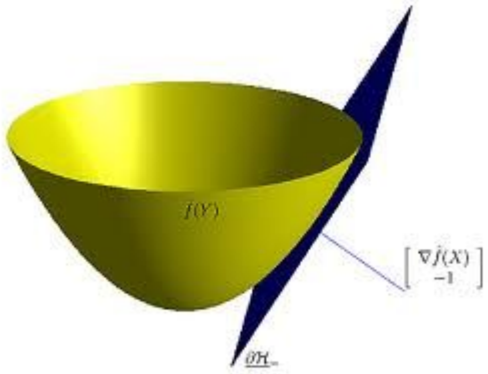


- Ideally, continuous (i.e. differentiable) so we get an indication of direction of minimization
- Only one minima

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

Ideas?

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

Surrogate loss functions

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

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?

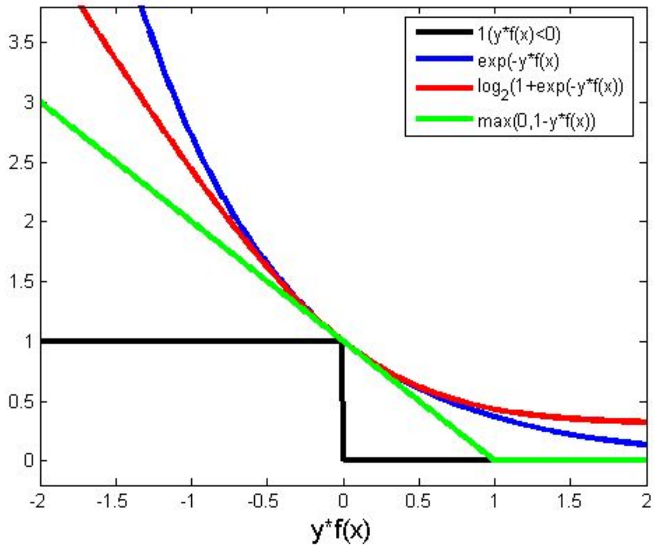
Surrogate loss functions

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

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

$$0 = b + \sum_{j=1}^{m} w_j f_j$$

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

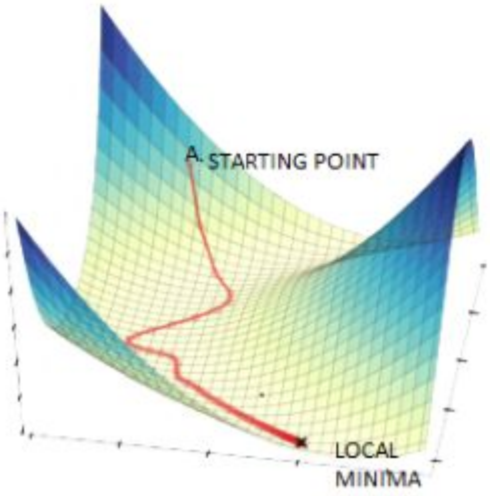
$$\sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b))$$

use a convex surrogate loss function

3. develop a learning algorithm

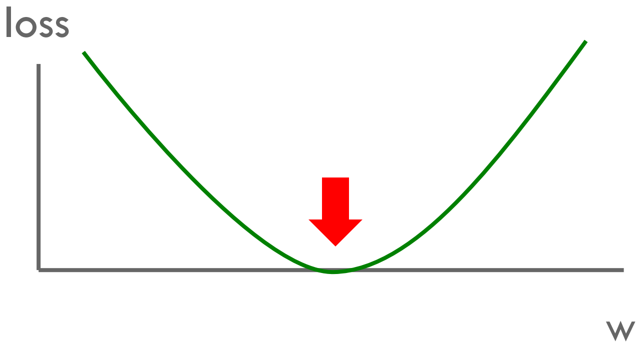
$$\text{argmin}_{w,b} \sum_{i=1}^{n} \exp(-y_i (w \cdot x_i + b))$$

Find w and b that minimize the surrogate loss



Gradient Descent

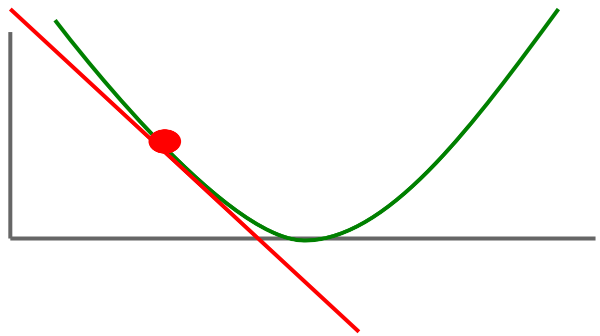
Finding the minimum



How do we do find the minimum for a function?

One approach: gradient descent

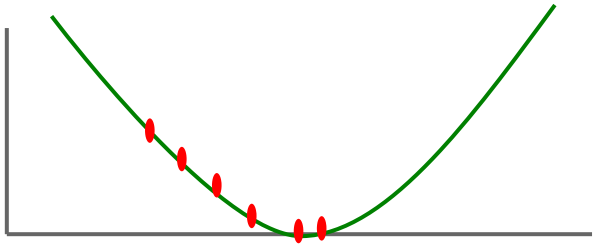
Partial derivatives give us the slope (i.e. direction to move) in that dimension



One approach: gradient descent

Approach:

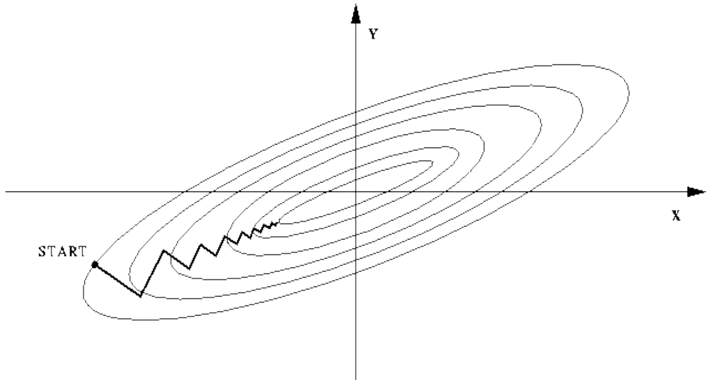
- pick a starting point (w)
- repeat:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)



One approach: gradient descent

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 loss doesn't decrease in any dimension:
  - pick a dimension
  - move a small amount in that dimension towards decreasing loss (using the derivative)

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

Why negative?

Gradient descent

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

Learning rate

How much we want to move in the error direction, often this will change over time

Some math

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

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

Some math

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

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

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

$$= -y_i x_{ij}$$

Some math

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

Gradient descent

- For our choice of the loss we have:

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

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

What is this doing?

Exponential update rule

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

---

for each example x_i:

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

Does this look familiar?

Perceptron learning algorithm!

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

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

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

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

for each w_i:

$w_i = w_i + f_i$*label

$b = b$ + label

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

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

Perceptron learning algorithm!

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

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

$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

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

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

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

The constant

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

learning rate label prediction

When is this large/small?

The constant

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

label

prediction

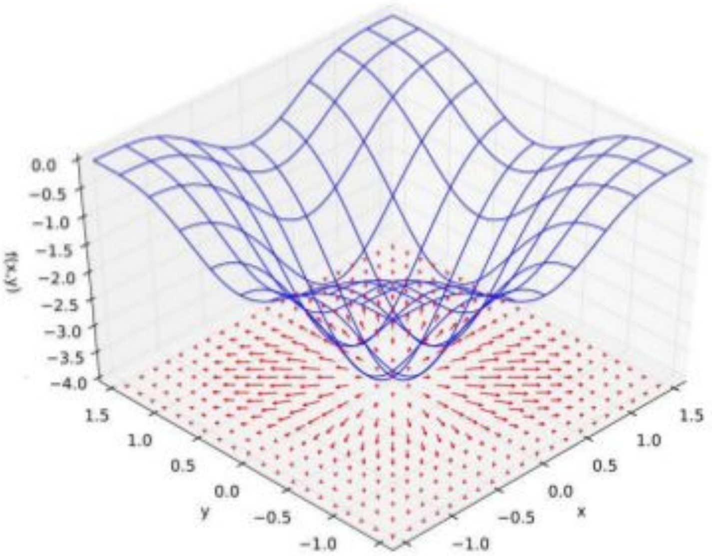
- If they are the same sign, as the predicted gets larger there update gets smaller
- If they are different, the more different they are, the bigger the update

Gradient

- The gradient is the vector of partial derivatives wrt to all the coordinates of the weights:

$$\nabla_{\mathbf{w}} L = \left[\frac{\partial L}{\partial w_1} \ \frac{\partial L}{\partial w_2} \cdots \frac{\partial L}{\partial w_N} \right]$$

- Each partial derivative measures how fast the loss changes in one direction.
- When the gradient is zero, i.e. all the partials derivatives are zero, the loss is not changing in any direction.
- Note: the arrows (gradients) point out from a minimum toward a maximum.



Gradient Descent

Algorithm 21 $\textsc{GradientDescent}(\mathcal{F}, K, \eta_1, \dots)$

$z^{(0)} \leftarrow \langle 0, 0, \dots, 0 \rangle$ // initialize variable we are optimizing

for $k = 1 \dots K$ **do**

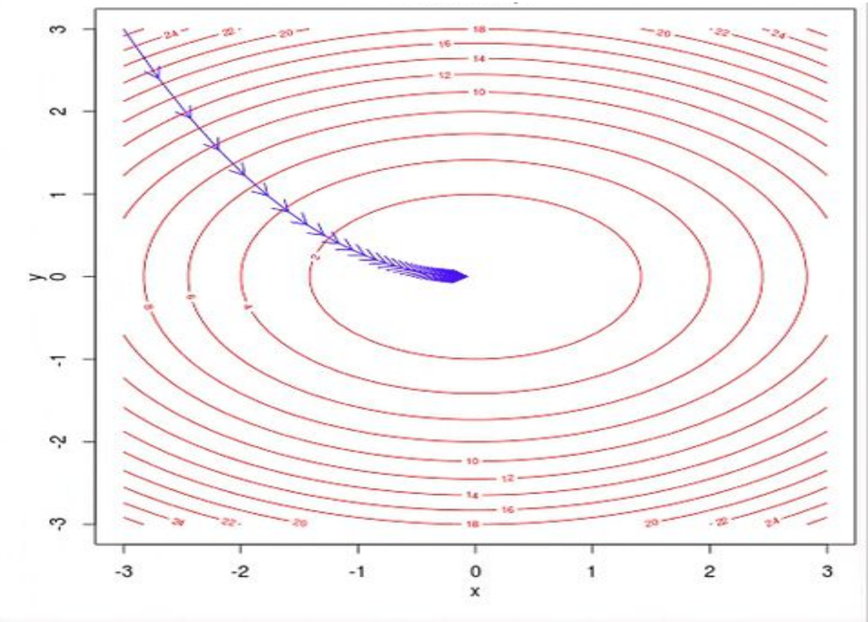
  $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location

  $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient

end for

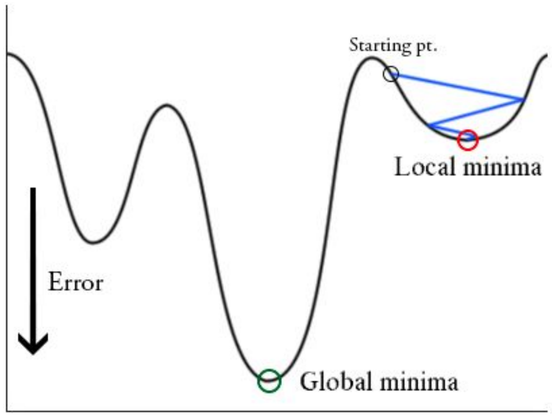
return $z^{(K)}$

Gradient Descent

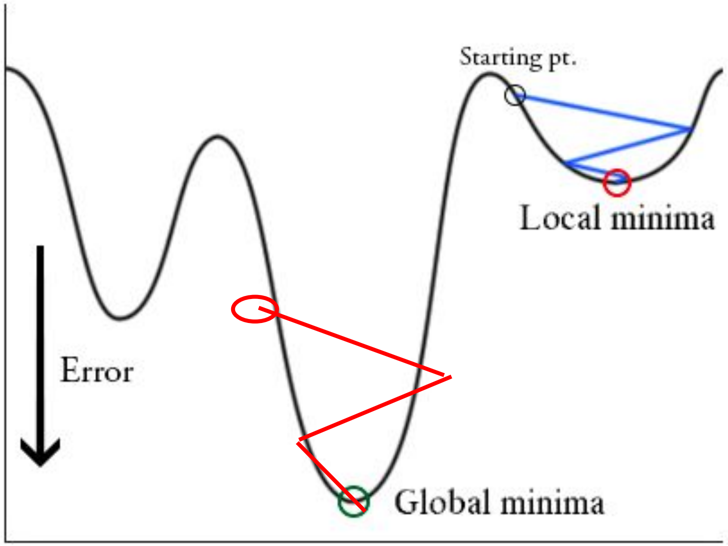


Gradient descent

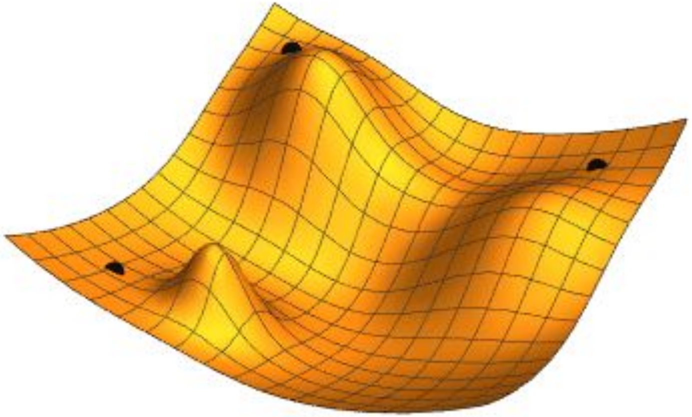
- In problems where the optimization problem is non-convex, it probably has **local minima.**
- An example is neural network. For long time people did not use NN, because they prefer methods that are guarantees to find the optimal solution.



Gradient descent



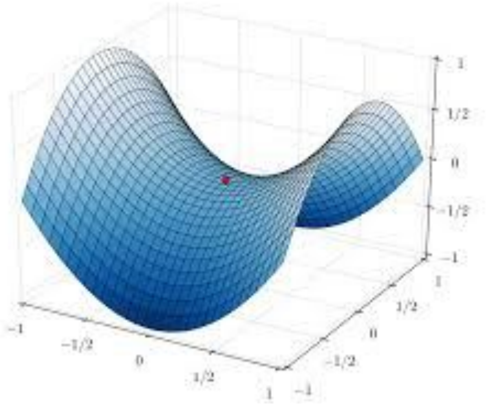
Gradient descent



https://commons.wikimedia.org/wiki/File:Gradient_descent.gif

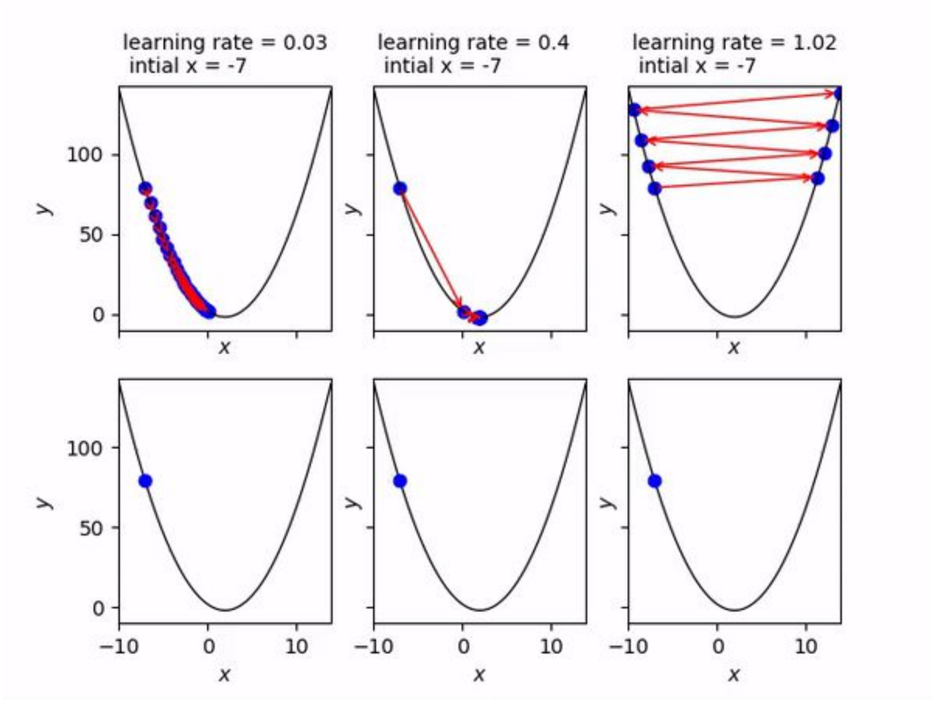
Gradient descent

- **Saddle point**: Some directions curve upwards, and others curve downwards.
- At a saddle point, the gradient is 0 even if we are not at a minimum.
- If we are exactly on the saddle point, then we are stuck.
- If we are slightly to the side, then we can get unstuck.
- **Saddle points very common in high dimensions**!



Learning Rate

- Very important hyper-parameter



Summary

- Model-based machine learning:
  - define a model, objective function (i.e. loss function), minimization algorithm
- Gradient descent minimization algorithm
  - so far we consider the case where the loss function is convex
  - make small updates towards lower losses
- Perceptron learning algorithm and gradient descent/exponential loss function (modulo a learning rate)
- Gradient descent in general

QUESTIONS?

Some slides are taken from David Kauchak