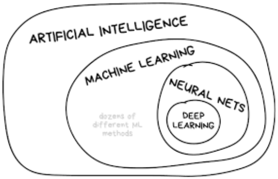
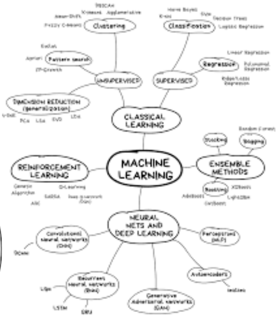
The mathematics you wish you had learned

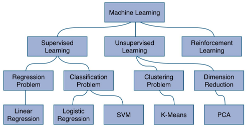
A gentle introduction to machine learning

Rough Outline of Machine Learning

Simply put, machine learning can be hard and confusing.







Various interpretations of ML

Key idea: Machine learning is a method of using data to learn to make decisions.

Rough Outline of Machine Learning

This description makes ML a very broad and powerful tool. It can potentially be used to solve problems such as

- classifying an image into one (or more) of many classes
- determining the price to set a particular product at
- predicting drops in the housing market
- identifying potentially dangerous strands of flu for a given season
- allocating resources to maximize relief following natural disasters
- clustering ER patients in various urgency categories
- learning to exceed at a video game

All of these are completely feasible under a few assumptions.

Rough Outline of Machine Learning

For our purposes, we will break down machine learning in the following way:

1. Supervised Learning - learning from training samples
   - broken down into clustering or regression
   - analogous to a grade school student
   - algorithms include MLP, SVM, Random forests, Naive Bayes, etc.
2. Unsupervised Learning - learning from raw data
   - learns relationship about data
   - requires less from the design perspective, but is less powerful
   - algorithms include PCA, K-means, SVD, etc.
3. Reinforcement Learning - learning from taking actions
   - attempts to learn about an environment
   - analogous to a researcher
   - algorithms include Q-Learning, Policy Gradient, Actor-Critic, DQN, etc.

Supervised Learning Thought Process

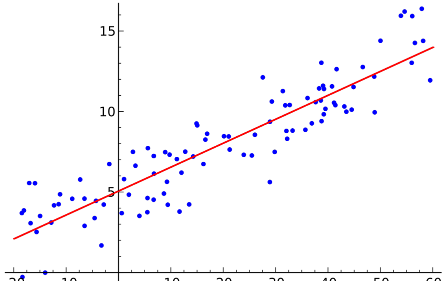
Let us now try to intuitively derive a supervised learning formulation ourselves. Assume that our task is to predict an output given some input and that we are given some training data x_i, y_i to learn from. A few observations can be made

1. It makes sense to represent our decision maker as a function $f : \mathbb{R}^m \to \mathbb{R}^n$. That is, something that takes inputs and through some black magic gives us some desired output.
2. We need a way to evaluate how well our model (function) is performing. We can introduce the idea of a loss function $L : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ to do this.
3. The best model (function) is going to be the one with the lowest loss (assuming we are using a good loss function).

Thus, to find the best decision maker, we simply need to minimize $L(f(x_i), y_i)$.

Supervised Learning Simple Example

With this approach, we can derive linear regression quite easily. Assume that we wish to find the "line of best fit" for some training data $x_i, y_i \in \mathbb{R}$.



A linear function can be written in the form $f(x) = ax + b$. Using the standard mean squared error, the solution to this regression problem is

$$\min_{a,b\in\mathbb{R}} \sum_i (ax_i + b - y_i)^2$$

The Foundation of ML: Optimization

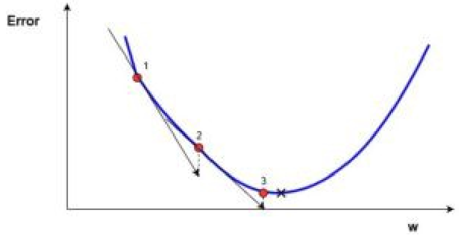
An even more broad field is that of mathematical optimization.
Optimization is the study of minimizing/maximizing an objective under a set of constraints. For example, the following are all optimization problems

- minimizing labor cost while still meeting product demands
- maximizing profit while keeping the associated risk low
- minimizing the error of a prediction

The vast majority of ML models/algorithms are leveraged using techniques from optimization-.

Basics of Gradient Descent

Consider now a simple but effective algorithm for minimizing a function: gradient descent.



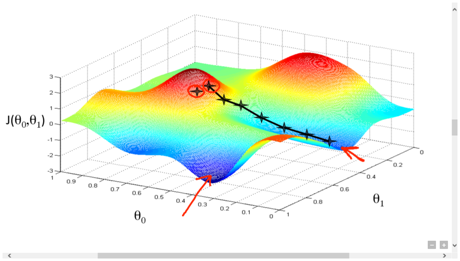
The **gradient** at a point gives the "instantaneous rate of change" at that point. That is, the gradient tells us the direction to travel to increase the function value. With a minimization goal in mind, we simply need to follow the **negative** gradient direction.

Basics of Gradient Descent

This yields the incredibly popular gradient descent algorithm update

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

where $\alpha \in \mathbb{R}$ is called the **step size**.



Multivariate gradient descent iterations

All that's left to do is to figure out when to stop.

Basics of Gradient Descent

Definition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function, $x \in \mathbb{R}^n$, and $\varepsilon > 0$ such that $f(x) \leq f(x + \lambda d)$ for any $d \in \mathbb{R}^n, \lambda \leq \varepsilon$. Here we say that x is a **local** minimum. If, in addition, $f(x) \leq f(y)$ for all $y \in \mathbb{R}^n$, then x is said to be a **global** minimum.

Theorem

Any local minimum $x \in \mathbb{R}^n$ of $f : \mathbb{R}^n \to \mathbb{R}$ satisfies $\nabla f(x) = 0$.

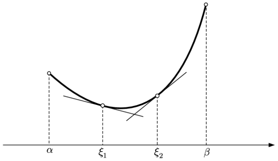
Thus, it makes no sense to continue once we have found a local minimum (why?). We would like a global minimum, but, unfortunately,from our iterative update, we only receive local information. To bridge the gap, we turn to convexity.

Convexity

Definition

A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be convex if for any $x, y \in \mathbb{R}^n$

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$



Geometric interpretation of convexity

Theorem

Any $x \in \mathbb{R}^n$ satisfying $\nabla f(x) = 0$ for a convex function $f : \mathbb{R}^n \to \mathbb{R}$ is a global minimizer.

Convexity

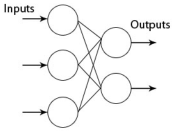
That is, convexity allows us to extract global information from local information. This makes convex optimization drastically easier than nonconvex optimization (ask me how much). We summarize the vanilla gradient descent algorithm as follows

Algorithm 1 Gradient Descent

```
Initialize $x_0 \in \mathbb{R}^n, \alpha \in \mathbb{R}$. Set $i = 0$
while $\nabla f(x_i) \neq 0$ do
    $x_{i+1} = x_i - \alpha \nabla f(x_i)$
    $i = i + 1$
end while
Output $x_i$.
```

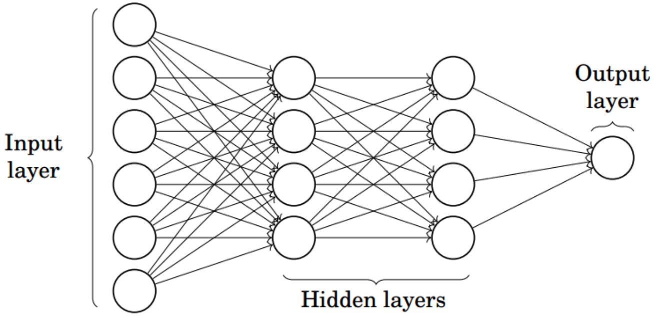
Neural Networks

With an introductory level of optimization skills in hand, all that is left to do is decide what exactly we need to optimize. More specifically, we need to construct the function $f : \mathbb{R}^n \to \mathbb{R}$ previously mentioned.
A **neural network** is a sequence of linear-nonlinear transformations that are used to convert input data to output data. For example, suppose we wish to use a function to decide how much a house will sell for on the market and how long it will take to sell given the square footage, the number of bedrooms, and zip code it is located in. A matrix $A \in \mathbb{R}^{3\times 2}$ can accomplish this by the function $f : \mathbb{R}^3 \to \mathbb{R}^2$ given by $x \mapsto Ax$. To put our output in the right "format", we apply an additional activation function $g : \mathbb{R}^2 \to \mathbb{R}^2$.



Neural Networks

However, there is no reason to believe that this simplified model has enough degrees of freedom to appropriately model the relationship we are looking for. To increase the decision making power, we can add more layers[1].



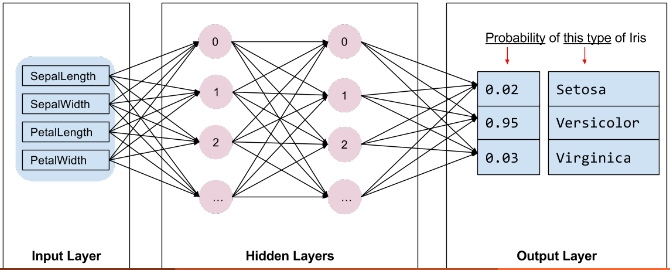
Multi Layered Perceptron

[1]Not always a good idea

Neural Networks - Example

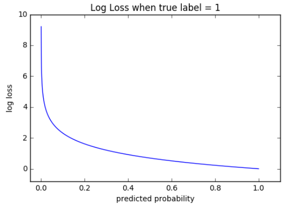
Consider the following famous problem in machine learning. The iris flower family contains several species such as Setosa, Versicolor, Virginica, etc. Using features features (sepal length, sepal width, petal length, petal width), construct a model which can accurately predict the species from the features provided. Again, assume there are N samples of training data denoted x_i, y_i.

- Step 1 - Decide model/architecture. Let us use a 2 hidden layer neural network with 10 hidden neurons in each layer.



Neural Networks - Example

- Step 2 - Randomly initialize the weights of our model and the step size used for gradient descent. Note that we have a total of $(4)(10) + (10)(10) + (10)(3) = 170$ variables.
- Step 3 - Using the training data and our loss function, compute the average loss $\frac{1}{N}\sum_{i=1}^{N} L(f(x_i), y(i))$.



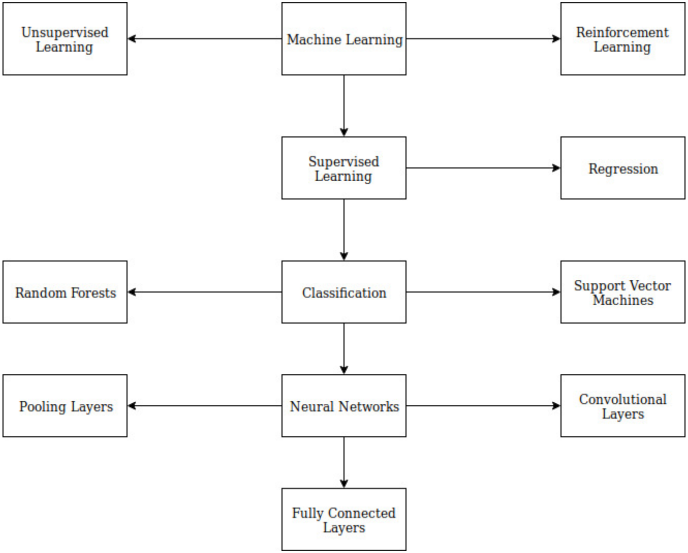
- Step 4 - If the loss is not great or some other stopping criteria is met, compute the gradient and update our weights via $w_{ij} = w_{ij} - \alpha \nabla_{w_{ij}} L(f(x_i), y(i))$.
- Step 5 - Repeat until an exit criterion is met.

Neural Networks - Example

Or, if you are using a good package like sklearn (scikit-learn's python package), these 8 lines will do

```
from sklearn.datasets import load_iris
from sklearn.neural_network import MLPClassifier
import numpy as np
iris = load_iris()
trainData = iris.data
trainLabels = iris.target
clf = MLPClassifier(solver='adam', alpha=1e-5,
                    hidden_layer_sizes=(2, 10))
clf.fit(trainData,trainLabels)
```

Where we stand



Where we stand

Things not covered

- bias vs variance tradeoff
- convolutional layers
- dropout
- random forests
- adagrad
- stochastic optimizers
- regularization
- summary statistics
- lasso and ridge regression
- momentum
- accelerated gradient descent
- policy gradient
- Markov decision processes
- Bellman optimality condition
- model validation
- online learning
- clustering algorithms
- hyperparameter selection
- adam optimizer
- variance reduction methods
- minibatch gradient descent
- transfer learning
- deep learning

and many more...

Final Points

A few lasting things to think about when dealing with machine learning

1. Many ML algorithms do not offer any interpretability and cannot be expanded upon for long range utility.
2. Testing on data that the model has trained on is absolutely pointless.
3. Good machine learning is computationally expensive and takes time.
4. As with any process that requires data, supervised learning follows the garbage in garbage out philosophy. Machine learning performs best with strong data.
5. Because of the feasibility of implementing some models, it's increasingly more important to understand underlying details before putting a model into production. The scariest model is one that works incorrectly.