Neural Networks

Saumya Bhatnagar

March 5, 2020

Table of contents I

General
- ANN
- RNN
- CNN

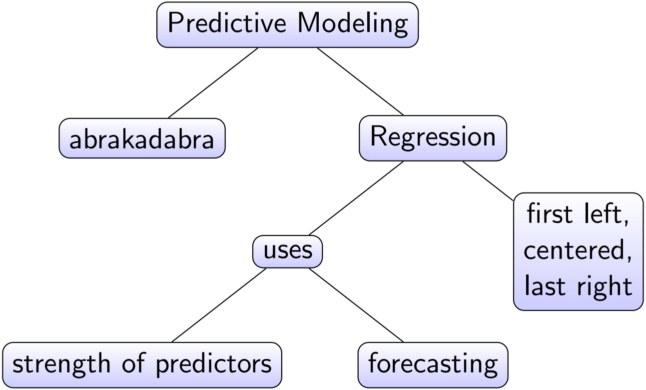
Optimization

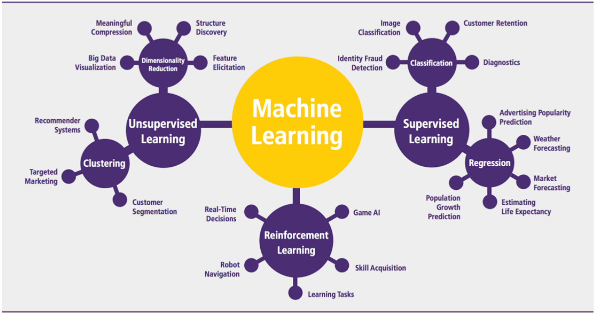
Graph Theory

Algorithms & Data Structures

Autoencoders

RBM - Restrictive Boltzmann Machine


Predictive Modeling
abrakadabra
Regression
uses
first left,
centered,
last right
strength of predictors
forecasting



	Categorial values	Continuous values
Supervised learning	**Two-class/Binary classification** • Decision Forest • Decision Tree • Naive Bayes/Bayes Classifier • (Deep) Neural Networks • Support Vector Machines **Multi-class classification** • Decision Forest • Logistic Regression • (Deep) Neural Networks	**Regression** • Bayesian Linear Regression • Decision Forest Regression • Decision Tree Regression • Linear Regression • Neural Network Regression • Ordinary Least Squares Regression
Unsupervised learning	**Association Rule Learning** • Apriori algorithm • Frequent Pattern Growth **Classification** • Autoencoders • Restricted Boltzmann Machines	**Clustering** • Density-based Clustering • Hierarchical Clustering • Partitional Clustering (incl. K-means) **Dimensionality reduction** • Principal Component Analysis (PCA)

	Categorial values
Supervised learning	**Two-class/Binary cla** • Decision Forest • Decision Tree • Naive Bayes/Bayes • (Deep) Neural Netw • Support Vector Ma **Multi-class classificat** • Decision Forest • Logistic Regression • (Deep) Neural Netw
Unsupervised learning	**Association Rule Lea** • Apriori algorithm • Frequent Pattern Gr **Classification** • Autoencoders • Restricted Boltzma


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Why to Study Neural Computation

- to study how brain works
- to understand the styles of parallel computations inspired by neurons and their adaptive connections
- to solve practical problems by using novel learning algorithms inspired by brain

TIMIT BENCHMARK FOR SOUND

content...

Linear Neurons I

Linear Neurons

Output, y = b + $\sum x_i w_i$

= activities on input line times weight

$x_i = i^{th}$ input

$w_i = i^{th}$ weight

$b = bias$

Binary Threshold Neurons

Output, z = b + $\sum x_i w_i$

$$y = \begin{cases} 1, & z >= 0 \\ 0, & otherwise \end{cases} \tag{1}$$

OR Output, z = $\sum x_i w_i$

$$y = \begin{cases} 1, & z >= \theta \\ 0, & otherwise \end{cases} \tag{2}$$

θ = threshold = -b

Linear Neurons II

Sigmoid Neurons
Output, z = b + $\sum x_i w_i$
derivatives makes learning easy
use logistic function, y = $\frac{1}{1+se^{-z}}$
Stochastic Binary Neurons: o/p of logistic as probability of spike in a short time window

Rectified Linear Neurons
aka Linear Threshold Neurons
Output, z = b + $\sum x_i w_i$
= non-linear function $\leftarrow$ linear weighted sum of inputs

$$y = \begin{cases} z, & z >= 0 \\ 0, & otherwise \end{cases} \tag{3}$$

Apply stochastic methods in ReLU
o/p = rate of producing spikes (deterministic)
use rate to get the actual times at which spikes are produced $\Rightarrow$ Random process

Linear Neurons III

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1+e^z)$	Multi-layer Neural Networks	

Supervised vs Unsupervised vs Reinforcement Learning

Model-class: Predicted output, $\hat{y} = f(x; W)$
W = Numerical parameters
input vector
Regression, minimize $\frac{1}{2}(y - \hat{y})^2$
Reinforcement Learning:
o/p is action or sequence of actions
supervision by rewards
max expected sum of future rewards
use discount factor for delayed rewards, so won't have to look too far into future
why not RL?: in long sequence of actions rewards are delayed so it's hard to know where we went wrong
a scalar reward doesn't supply much info

Feedforward NN

information comes from input unit, flows in one direction towards hidden layer and gives the output

if more than one hidden layer $\Rightarrow$ **Deep NN**

compute transformations:

activities of neurons in each layer are a non-linear func of the activities in layer below

Perceptron convergence procedure

training binary o/p neurons as classifiers
add an extra component with value 1 to each i/p vector. the "bias" wt on this component is minus the threshold. now forget the threshold.
pick training cases using any policy that ensures that every training case will keep getting picked

- if o/p is correct, do nothing
- if incorrectly o/ps to 0, add i/p vector to wt vector
- if incorrectly o/ps to 1, subtract i/p vector from wt vector

Padding and stride

Padding is used to preserve the boundary information , since without padding they are only traversed once.//

example

content...

Sequential Data
o/p dependent on prev computations saved in memory
Feedforward(touches node once) vs **RNN**(loops: present in recent past)
why not feedforward?: can't predict the next word in a sentence if I use feed-forward
Backpropagation of error and gradient descent
Error will return via **Backpropagation** adjusting weights
sentence of 5 words, 5 layered NN

- $\sim$ very deep nets with hidden layer per time slice
- get input at every time slice
- use same weights at every time slice

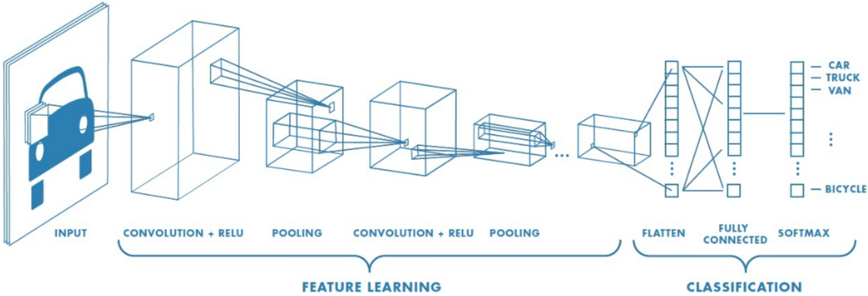
RNN with multiple hidden layers are special case where some hidden $\rightarrow$ hidden connections are missing

content...

convnets I

A convolution operation is basically computing a dot product between their weights and a small region they are connected(currently overlapping) to in the input volume. This will change the dimensions depending on the filter size used and number of filters used.
Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.

convnets II



a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer

spatial size of the output volume

for calculating how many neurons "fit", use
(W-F+2P)/S + 1
is a function of the:
input volume size (W),
the receptive field size of the Conv Layer neurons (F),
the stride with which they are applied (S),
and the amount of zero padding used (P) on the border
For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output. With stride 2 we would get a 3x3 output.

Non-Linear Optimization

OOPS

Overloading vs overriding

Abstract vs Interface

content...

Principle

ACID: Atomic ... Isolate ...
Dependency Inversion Principle: High level module shouldn't depend on low level module
Liskov Principle
Interface segregation
CURD (Create, Read, Update, Delete)

Non

content...

Thank You!