Python Machine Learning: The 1st Book Circle

Classification Algorithm I: Perceptron, Adaline, Gradient Descent Searching vs LMS/RLS for Adaptive Filter

Jianan Liu

Gothenburg

Nov, 3, 2016

Disclaimer

All opinions and statements in this presentation are mine and do not in any way represent the company
Any comment or correction of error is welcome, please contact me
chisyliu@hotmail.com

Acknowledgement

Acknowledgement

Here I would like to thank for my colleague in Baseband Software Division, Andreas Hultberg, for his comments and correction of the errors about perceptron in this material

The First Book Circle of Python Machine Learning

In this presentation belongs to **algorithm** part of the book

- If you have time, please read the book first. This slide could be used as complementary resource for the book
- We will try to emphasize the key points in chapter 1
- We will try to go through every algorithm in the chapter 2 of book Python Machine Learning. Also show the mathematics behind each algorithm
- All of us need to debug the python code, in order to get practice of implementing machine learning algorithm

Overview

General Overview of First 50 Pages

Remark

- Chapter 1 introduces the basic conception of machine learning(most of contents could be found in the introduction presentation we had)
- Chapter 2 explains several basic(but very important) models and algorithms for classification
- Till now, we still haven't step into using python library(e.g. scikitlearn)

Key Points of Chapter 1

- Supervised learning, unsupervised learning and reinforcement learning concepts(discussed in introduction slide)
- The reinforcement learning could be seen as "feedback loop" or "interactive version" of supervised learning, but the training data are NOT correct ground truth label
- Dimensionality reduction could be seen as kind of unsupervised learning as well

Key Points of Chapter 1

- Machine learning = preprocessing(of data) + learning(model + algorithm) + evaluation(of the performance of learning) + prediction(apply the learning for new unknow data)
- Preprocessing, e.g. dimensionality reduction for training data(e.g. remove noise, extract of features, compress of size of training data) which will be used in supervised learning, could provide better preparation for supervised learning
- Model selection and training, using of algorithm to optimize the model in order to fit the training data. Note the training data is usually divided into training data and validation data by **cross validation**(We will discuss it in detail in later part of book)
- Evaluation, never use same data for both training and validation

Outline for Section 1

Data in Chapter 2

From this slide we go through the classification algorithms in chapter 2

Training data set ($\mathbf{y}$, $\mathbf{X}$)

Suppose we have L samples, each sample has D features/dimensions(So the input $\mathbf{X}$ is L by D matrix, label $\mathbf{y}$ is L by 1 vector)

Percetron: Linear Classifier

Famous Perceptron :)

Percetron Model

Percetron Model

- $z^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$, $\mathbf{x}^{(l)}$ is 1 by D vector which represents D features/dimension for l^{th} training data sample and $\mathbf{w}$ is D by 1 vector which represents weight
- $y^{(l)} = 1$, if $z^{(l)} \geq$ threshold; $y^{(l)} =$ -1, otherwise. where $y^{(l)}$ is the decision output for l^{th} training data sample

Adaptive Algorithm for Percetron Model

```
1:  Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2:  for t in T do
3:      for l in L do
4:          computer $\hat{y}^{(l)} = \mathrm{sgn}(\mathbf{x}^{(l)}\mathbf{w})$
5:          for d in D do
6:              computer $w_d(t+1) = w_d(t) + \eta(y^{(l)} - \hat{y}^{(l)})x_d^{(l)}$
7:              d = d + 1
8:          end for
9:          Return w if all training samples are correctly classified
10:         t = t + 1
11:     end for
12: end for
```

- $\hat{y}^{(l)}$ is predication of decision output of l^{th} sample
- $y^{(l)}$ is the real decision output of l^{th} sample training data
- T is the number of iteration(guarantee algorithm can stop)

Adaptive Algorithm for Percetron Model

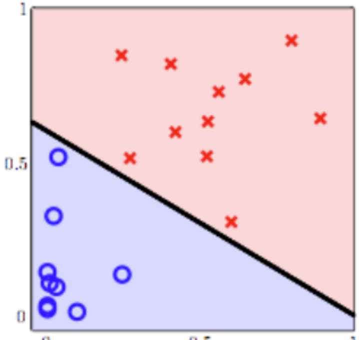
- Idea of percetron is to adjust the position of the line(weight of model) which tries to classify the training data samples, if there is error found(a certain data sample is NOT classified correctly according to current model)
- The algorithm stops and return the weight until all the training data samples are classified(no error at all for pass over all the training samples) by current line(model)

Percetron: Linear Classifier

The percetron is the fundamental model of neural network

Remark

- Percetron(and adaptive linear neuron/signal layer neural network we will see in next bullet) works only when the data is **linear separable**

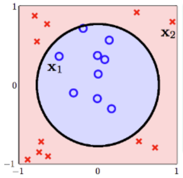


Percetron: Linear Classifier

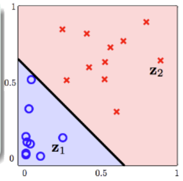
How about when the data is NOT linear separable?

Remark

- **Supportive vector machine** classifier and **multi-layer neural network** could somehow "convert" data from nonlinear separable status to linear separable(for example in the figure), we will see how they work in the later chapters of the book



- $\{(\mathbf{x}_n, y_n)\}$ circular separable $\Longrightarrow \{(\mathbf{z}_n, y_n)\}$ linear separable
- $\mathbf{x} \in \mathcal{X} \overset{\Phi}{\longmapsto} \mathbf{z} \in \mathcal{Z}$: (nonlinear) feature transform $\mathbf{\Phi}$



Single Layer Neural Network(Adaptive Linear Neuron: Adaline)

Adaptive linear neuron(Adaline) has almost same model as percetron but with a little change

- In percetron, the weight is updated according to $\Delta \mathbf{w} = \eta(y^{(l)} - \hat{y}^{(l)})\mathbf{x}^{(l)}$, where $\hat{y}^{(l)} = \mathrm{sgn}(\mathbf{x}^{(l)}\mathbf{w})$. Which means we use the binary value $y^{(l)}$ as lable of training data directly to update weight of model
- But in adaptive linear neuron(adaline), we use $\hat{y}^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$ to be as criteria for updating weight, which means instead of updating weight by applying the binary predication to compare with training data directly, we use the continues predication value $\hat{y}^{(l)}$ to calculate the difference/error by comparing with binary label $y^{(l)}$ and update weight of linear model $\mathbf{x}^{(l)}\mathbf{w}$, then apply the weight and map the result of new unknow data to binary value

Single Layer Neural Network(Adaptive Linear Neuron: Adaline)

Note

- Training the Adaline model is totally different compared with training of percetron, it is NOT training/turning of the weight of binary model directly as training of percetron
- Instead, **training of the Adaline(binary classifier) model = training of linear regression model + apply the trained (weight in) model for new unknow data to implement binary classifier by binary mapping sign**

Batch Gradient Descent for Adaline Model

How to train the linear regression model $y^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$ for all l in L sample?

Batch Gradient Descent

- The objective cost function $\mathbf{J}(\mathbf{w}) = \Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})^2$ to be minimized, is convex
- The adaptive algorithm which updates the weights by taking a step away from the gradient $\bigtriangledown\mathbf{J}(\mathbf{w})$ of our cost function $\mathbf{J}(\mathbf{w})$: $\mathbf{w}(t+1) = \mathbf{w}(t) + (-1)\eta \bigtriangledown \mathbf{J}(\mathbf{w})$
- $\bigtriangledown\mathbf{J}(\mathbf{w})$ equals to compute the partial derivative of the cost function with respect to each weight w_d: $\bigtriangledown\mathbf{J}(\mathbf{w}) = \frac{\partial \mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-\Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w}) x_d^{(l)})$ for all w_d

Batch Gradient Descent for Adaline Model

```
1: Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2: for t in T do
3:     for d in D do
4:         computer $w_d(t+1) = w_d(t) + \eta\Sigma_l(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
5:         d = d + 1
6:     end for
7:     t = t + 1
8: end for
```

- $y^{(l)}$ is the real continues intermediate output of l^{th} sample training data
- T is the number of iteration

Batch Gradient Descent for Adaline Model

- Compared with percetron, BGD for adaline doesn't update the weight $\mathbf{w}$ every sample but count all L samples together to update weight once. That is reason it is **Batch** gradient descent

Note

- The model should be valid generally for ensemble/all possible data set rather than the training data samples only, but the objective cost function used in BGD $\mathbf{J}(\mathbf{w}) = \Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})^2$ clearly counts the samples only. Why it could be used generally for all/ensemble data set? We will discuss the mathematics behind GD algorithm later in this presentation, see how GD is relative to MMSE and what trick we use in BGD for adaline to convert a statistical problem into a deterministic problem

Stochastic Gradient Descent

In most of the scenario of supervised machine learning, the number of training data samples L is quite large, the computation time of updating weight will be too long for BGD cause it needs all samples to update the weight once

Stochastic Gradient Descent

- $\bigtriangledown\mathbf{J}(\mathbf{w})$ in BGD is $\bigtriangledown\mathbf{J}(\mathbf{w}) = \frac{\partial\mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-\Sigma_l(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)})$ for all w_d, but now stead of computing $\bigtriangledown\mathbf{J}(\mathbf{w})$ by all L samples, we randomly update weight once per sample
- $\bigtriangledown\mathbf{J}(\mathbf{w})$ in SGD is $\bigtriangledown\mathbf{J}(\mathbf{w}) = \frac{\partial\mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)})$ for all w_d, it is updated every sample

Stochastic Gradient Descent for Adaline Model

```
1:  Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2:  for t in T do
3:      Shuffle(); // randomly sort the L samples once
4:      for l in L do
5:          for d in D do
6:              computer $w_d(t+1) = w_d(t) + \eta(t)(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
7:              d = d + 1
8:          end for
9:          l = l + 1
10:     end for
11:     t = t + 1
12: end for
```

- $\eta(t)$ means η is a function of t

Stochastic Gradient Descent for Adaline Model

Adaptive Learning Rate

The learning rate should also be adaptively changed, basically decreases over time(The closer to the global minimum the smaller each "step of learning" should be in order to ensure the (approximate)minimum value could be captured), a common method is define the learning rate to be inverse of number of iteration time

- $\eta(t) = \frac{a}{t+b}$, a and b are constants

Mini-Batch Gradient Descent: Hybrid Combination of Ideas from BGD and SGD

We could also update weight once per patch, a patch has several samples(larger than 1 sample in SGD but smaller than all samples as in BGD). For each iteration, Mini-Batch Gradient Descent uses a patch(which contains "number of batch size" samples) to train target model. It will use all patches by iteration "number of batches" times. Note "patch size" x "number of patches" = total number of data samples in data set. All data samples in data set forms an epoch

Mini-Batch Gradient Descent for Adaline Model

```
 1: Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
 2: for e in E do
 3:     for t in T do
 4:         for d in D do
 5:             computer
    $w_d(t+1) = w_d(t) + \eta\Sigma_{n=1}^{N}(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
 6:             d = d + 1
 7:         end for
 8:         t = t + 1
 9:     end for
10: end for
```

- E is number of epoch(that is how many times we repeat on the whole data set)
- T is the number of iteration, and $T = L/N$. L is the total number of data samples in the data set and N is the batch size

Outline for Section 2

Detailed Analysis of Gradient Descent in The Book

From this slide, we start to analyze the reason why and how the gradient descent algorithms are designed. By knowing these, We will answer the question in page 19

- We will give the mathematics conclusion directly(more engineering manner) without any derivation, focus on using mathematics to explain engineering algorithm itself
- If you want to know the mathematics detail/derivation regarding this result, please refer to any machine learning textbook or statistical signal processing textbook

Criteria for Finding the Optimal Solution of Linear Regression Problem-MMSE or LS?

Now let us recall the idea of all the single neural network algorithms for classifier in chapter 2. All of them(Adaline, and SDG/Mini-Batch DG for Adaline) try to do the classification task in same way

- **Transform the linear regression model to a binary output by using a mapping function, but optimize/tuning the linear regression model rather than the whole model**

The core objective function to be optimized is the linear regression model $\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{n}$, where $\mathbf{y}$ is L by 1 vector, $\mathbf{X}$ is L by D matrix, $\mathbf{w}$ is D by 1 vector and $\mathbf{n}$ is L by 1 vector. We try to find parameter $\mathbf{w}$ to make the noise $\mathbf{n}$ as small as possible(zero to be the best), in order to make this model as accurate as possible to fit the training data

Criteria for Finding the Optimal Solution of Linear Regression Problem-MMSE or LS?

Fundamentally, designing the linear classifiers like adaptive linear neuron/single neural network, are "equivalent" to optimization of the linear regression model

Question

How do we find the optimal solution of linear regression model?

- Recall the introduction slide or statistic/adaptive signal processing course in university, two **criteria** to find the optimal solution **analytically**
- Minimize mean square error(Wiener Filter) in statistical/bayes manner or least square in maximize the likelihood manner

MMSE

For MMSE criteria, (if all elements in the matrix and vector are real value)the analytical optimal solution of parameter $\mathbf{w}$ in linear regression model is

- $\mathbf{w}_{MMSE} = \mathbf{R}_x^{-1}\mathbf{r}_{yx}$ by setting the gradient $= 0$(objective function is **convex** function and **no constrain**)
- where $\mathbf{R}_x$ is autocorrelation matrix. $\mathbf{R}_x = \mathbb{E}(\mathbf{X}^T\mathbf{X})$
- where $\mathbf{r}_{yx}$ is cross-correlation vector. $\mathbf{r}_{yx} = \mathbb{E}(\mathbf{X}^T\mathbf{y})$

This analytical solution needs the "expectation" of **ensemble** data set, but we only have several samples of data observed(e,g, L $=$ 1000 samples of y which is the $\mathbf{y}$) which represent ensemble data

Remark

We will see how to use Lagrange Multiplier to solve **constrained optimization problem** in SVM part next chapter

LS

For LS criteria, (if all elements in the matrix and vector are real value)the analytical optimal solution of parameter **w** in linear regression model is

- $\mathbf{w}_{LS} = \hat{\mathbf{R}}_x^{-1}\hat{\mathbf{r}}_{yx}$ by setting the gradient $= 0$(objective function is convex function and no other constrain)
- where $\hat{\mathbf{R}}_x$ is autocorrelation matrix. $\hat{\mathbf{R}}_x = \mathbf{X}^T\mathbf{X}$
- where $\hat{\mathbf{r}}_{yx}$ is cross-correlation vector. $\hat{\mathbf{r}}_{yx} = \mathbf{X}^T\mathbf{y}$

This analytical solution doesn't need the "expectation" of ensemble data set, but only need the "**average**" of several samples of data observed(e,g, L = 1000 samples of y which is the **y**)

Remark

If the number of samples $\rightarrow \infty$. LS is same as MMSE in ergodic processes,

MMSE or LS?

- Ls is same as maximize likelihood(ML) for our linear regression model
- MMSE is same as maximize a posterior(MAP) for our linear regression model
- $\mathbf{w}_{LS}$ is unbias, which mean $\mathbb{E}(\mathbf{w}_{LS}) = \mathbf{w}_{LS}$.(the estimation is accurate) But the variance of $\mathbf{w}_{LS}$ could be large(which will cause overfitting which we will see in later chapter)
- $\mathbf{w}_{MMSE}$ is bias(the estimation is NOT very accurate), but variance of $\mathbf{w}_{MMSE}$ could be much lower(which will avoid overfitting)

LS Criteria of Channel Estimation for OFDM system

-

MMSE Criteria of Channel Estimation for OFDM system

-

LS/ZF Receiver for MIMO system

-

MMSE Receiver for MIMO system

-

Gradient Descent-Iterative Searching Algorithm to Find Optimal Solution, In Order to Avoid Calculating Inverse of $\mathbf{R}_x$

But, usually both analytical solutions are hard to be calculated! Why?

Answer

The inverse of matrix in engineering is usually impossible(in feasible time) or very costly, i.e. $\mathbf{w}_{MMSE} = \mathbf{R}_x^{-1}\mathbf{r}_{yx}$

That is reason why we go for gradient descent algorithm, e.g. BGD, to search the minimal value of objective function rather than calculating the MMSE or LS solution

Gradient Descent-Iterative Searching Algorithm to Find Optimal Solution, In Order to Avoid Calculating Inverse of $\mathbf{R}_x$

Instead of computing $\mathbf{w}_{MMSE} = \mathbf{R}_x^{-1}\mathbf{r}_{yx}$, where $\mathbf{R}_x = \mathbb{E}(\mathbf{X}^T\mathbf{X})$ and $\mathbf{r}_{yx} = \mathbb{E}(\mathbf{X}^T\mathbf{y})$. We want to find someway to search where is the point with minimum value of objective function $\mathbf{J}(\mathbf{w}) = \mathbb{E}((\mathbf{y} - \mathbf{X}\mathbf{w})^2)$. Beware the objective function $\mathbf{J}(\mathbf{w})$ is NOT same as the one we introduced in 17th page.

Note

- As we discussed, the model should be valid generally for ensemble/all possible data set rather than the training data samples only, but the objective cost function used in BGD $\mathbf{J}(\mathbf{w}) = \Sigma_l(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})^2$ clearly counts the samples only. So the correct objective function should be expectation for ensemble/all possible data set $\mathbf{J}(\mathbf{w}) = \mathbb{E}((\mathbf{y} - \mathbf{X}\mathbf{w})^2)$

Gradient Descent-Iterative Searching Algorithm to Find Optimal Solution, In Order to Avoid Calculating Inverse of $\mathbf{R}_x$

We notice the objective function is convex without any constraint, so we use the characteristic of unconstrained convex function

- Given a function f(x) of which we want to find the minimum, and an initial point $x^{(1)}$ with value $f^{(1)} := f(x^{(1)})$ and gradient $\bigtriangledown f^{(1)} = \bigtriangledown f(x^{(1)})$. For another point $x^{(2)}$ close to $x^{(1)}$, we can then write $\bigtriangledown f^{(1)} \approx \frac{f^{(2)} - f^{(1)}}{x^{(2)} - x^{(1)}}$
- So we could get $f^{(2)} \approx f^{(1)} + (x^{(2)} - x^{(1)}) \bigtriangledown f^{(1)}$, when searching along direction of the negative gradient by making small steps, the value of the function will get smaller.

Gradient Descent-Iterative Searching Algorithm to Find Optimal Solution, In Order to Avoid Calculating Inverse of $\mathbf{R}_x$

- In our gradient descent for linear regression problem, we have gradient of cost function $\bigtriangledown \mathbf{J}(\mathbf{w}) = \frac{\partial \mathbf{J}(\mathbf{w})}{\partial w_d} = \mathbf{R}_x \mathbf{w} - \mathbf{r}_{yx}$.
- The batch gradient descent algorithm becomes $\mathbf{w}(t+1) = \mathbf{w}(t) + (-1)\eta \bigtriangledown \mathbf{J}(\mathbf{w}) = \mathbf{w}(t) - \eta(\mathbf{R}_x \mathbf{w}(t) - \mathbf{r}_{yx})$
- Note $\mathbf{R}_x = \mathbb{E}(\mathbf{X}^T\mathbf{X})$, $\mathbf{r}_{yx} = \mathbb{E}(\mathbf{X}^T\mathbf{y})$ due that $\mathbf{J}(\mathbf{w}) = \mathbb{E}((\mathbf{y} - \mathbf{X}\mathbf{w})^2)$

Until now, we have the iterative expression of batch gradient descent algorithm

Remark

- Although we avoid computing of inverse of $\mathbf{R}_x$, but the batch gradient descent is still NOT practical in engineering, why?

LMS: Iterative Algorithm for MMSE Criteria, An Alternative Version for GD in Order to Avoid Calculating of $\mathbf{R}_x$ and $\mathbf{r}_{yx}$

Computation of $\mathbf{R}_x$ and $\mathbf{r}_{yx}$ are also time consumption or infeasible

- We actually use the approximation/estimation of $\mathbf{R}_x$ and $\mathbf{r}_{yx}$, which are $\hat{\mathbf{R}}_x = \mathbf{X}^T\mathbf{X}$ and $\hat{\mathbf{r}}_{yx} = \mathbf{X}^T\mathbf{y}$, to replace of $\mathbf{R}_x = \mathbb{E}(\mathbf{X}^T\mathbf{X})$ and $\mathbf{r}_{yx} = \mathbb{E}(\mathbf{X}^T\mathbf{y})$
- This approximation uses the L observed samples to represent the whole ensemble data set
- This approximation makes an alternative iterative algorithm, least mean square(LMS) to make batch gradient descent feasible
- **The batch gradient descent in the chapter 2 of book is actual LMS**

LMS: Iterative Algorithm for MMSE Criteria, An Alternative Version for GD in Order to Avoid Calculating of $\mathbf{R}_x$ and $\mathbf{r}_{yx}$

- Clearly seeing, We actually make approximation of the whole ensemble data set($\mathbf{R}_x = \mathbb{E}(\mathbf{X}^T\mathbf{X})$ and $\mathbf{r}_{yx} = \mathbb{E}(\mathbf{X}^T\mathbf{y})$), by using L observed samples($\hat{\mathbf{R}}_x = \mathbf{X}^T\mathbf{X}$ and $\hat{\mathbf{r}}_{yx} = \mathbf{X}^T\mathbf{y}$) in LMS
- It means **LMS is actual a maximize likelihood(ML)/least square(LS) sense iterative algorithm** rather than a maximize a posterior(MAP)/minimize mean square error(MMSE) sense algorithm. It uses LS sense data based way of search to approximate the MMSE solution. In principle, it is still a observed data sample only based algorithm rather than an ensemble data set based algorithm

Outline for Section 3

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

The LMS algorithm was inspired by a desire to iteratively calculate an estimate of the Wiener receiver(MMSE), without inverting and calculating data covariance matrices. Besides, is any other method to approximate of LS solution?

- Definitely, if we consider trick called matrix inversion lemma along the analytical solution of LS $\mathbf{w}_{LS}$, we will have the alternative algorithm of RL: recursive least square(RLS)
- Although it is NOT part of our Python Machine Learning book(Cause **most of optimization problems in machine learning are solved by using gradient descent searching, e.g. LMS, rather than derivation from analytic solution by matrix inversion lemma in a recursive way, e.g. RLS**), we anyway introduce RLS in this presentation

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

Matrix Inversion Lemma

Let us see a lemma called "Matrix Inversion Lemma"

- For the matrices of compatible sizes and all inverses exist
- $(\mathbf{A}-\mathbf{B}^T\mathbf{C}^{-1}\mathbf{B})^{-1}=\mathbf{A}^{-1}+\mathbf{A}^{-1}\mathbf{B}^T(\mathbf{C}-\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T)^{-1}\mathbf{B}\mathbf{A}^{-1}$

- Cause all elements in all the matrices are real values rather than complex values in our case, we have only transpose T. Otherwise T will be H, e.g. $\mathbf{B}^H$

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

- As we see, the analytical solution $\mathbf{w}_{LS} = \hat{\mathbf{R}}_x^{-1}\hat{\mathbf{r}}_{yx}$ where $\hat{\mathbf{R}}_x = \mathbf{X}^T\mathbf{X}$ and $\hat{\mathbf{r}}_{yx} = \mathbf{X}^T\mathbf{y}$ for our linear regression model
- Actually, $\hat{\mathbf{R}}_x = \frac{1}{L}\mathbf{X}^T\mathbf{X}$ and $\hat{\mathbf{r}}_{yx} = \frac{1}{L}\mathbf{X}^T\mathbf{y}$ for our linear regression model but $(\frac{1}{L})^{-1}$ and $\frac{1}{L}$ cancel each other, so we simply say $\hat{\mathbf{R}}_x = \mathbf{X}^T\mathbf{X}$ and $\hat{\mathbf{r}}_{yx} = \mathbf{X}^T\mathbf{y}$
- Let us define $\hat{\mathbf{R}}_x = \hat{\mathbf{R}}, \hat{\mathbf{r}}_{yx} = \hat{\mathbf{r}}$ for simplification

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

- Matrix inverse is needed if we compute analytical solution $\mathbf{w}_{LS} = \hat{\mathbf{R}}_x^{-1}\hat{\mathbf{r}}_{yx}$ directly, so we want to avoid
- We also consider a recursive way of getting the solution, e.g. at each time we acquire a new data to compute the new solution recursively. Considering when we take in l+1 data samples$(1 \leq l \geq L)$, we have $\mathbf{w}(l+1) = \hat{\mathbf{R}}(l+1)^{-1}\hat{\mathbf{r}}(l+1)$
- We also notice $\hat{\mathbf{R}}(l+1) = \hat{\mathbf{R}}(l) + \mathbf{x}^{(l+1)}{}^T\mathbf{x}^{(l+1)}$
- We map our notations into notations in matrix inversion lemma, which are $\mathbf{A} \to \hat{\mathbf{R}}(l)$, $\mathbf{B} \to \mathbf{x}^{(l+1)}$ and $\mathbf{C} \to -1$

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

- Then we can derivate $\hat{\mathbf{R}}(l+1)^{-1} = (\hat{\mathbf{R}}(l) + \mathbf{x}^{(l+1)^T}\mathbf{x}^{(l+1)})^{-1}$ $= \hat{\mathbf{R}}(l)^{-1} - \frac{\hat{\mathbf{R}}(l)^{-1}\mathbf{x}^{(l+1)^T}\mathbf{x}^{(l+1)}\hat{\mathbf{R}}(l)^{-1}}{1+\mathbf{x}^{(l+1)}\hat{\mathbf{R}}(l)^{-1}\mathbf{x}^{(l+1)^T}}$ by using matrix inversion lemma in recursive way
- With defining $\mathbf{P}_l = \hat{\mathbf{R}}(l)^{-1}$, we get $\mathbf{P}_{l+1} = \mathbf{P}_l - \frac{\mathbf{P}_l\mathbf{x}^{(l+1)^T}\mathbf{x}^{(l+1)}\mathbf{P}_l}{1+\mathbf{x}^{(l+1)}\mathbf{P}_l\mathbf{x}^{(l+1)^T}}$ to skip computation of matrix inverse
- We also notice $\hat{\mathbf{r}}(l+1) = \hat{\mathbf{r}}(l) + \mathbf{x}^{(l+1)^T} y^{(l+1)}$ and $\mathbf{w}(l+1) = \mathbf{P}_{l+1}\hat{\mathbf{r}}(l+1)$

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

We can get RLS algorithm

RLS algorithm

- $\mathbf{P}_{l+1} = \mathbf{P}_l - \frac{\mathbf{P}_l \mathbf{x}^{(l+1)}{}^T \mathbf{x}^{(l+1)} \mathbf{P}_l}{1 + \mathbf{x}^{(l+1)} \mathbf{P}_l \mathbf{x}^{(l+1)}{}^T}$
- $\hat{\mathbf{r}}(l+1) = \hat{\mathbf{r}}(l) + \mathbf{x}^{(l+1)}{}^T y^{(l+1)}$
- $\mathbf{w}(l+1) = \mathbf{P}_{l+1} \hat{\mathbf{r}}(l+1)$

- RLS represents solution recursively
- RLS avoid matrix inverse computation
- **$\mathbf{w}(l)$ for l^{th} recursive step is indeed the $\mathbf{w}_{LS}(l)$ of 'samples from 1, 2, to l^{th}'**

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

- Now we have
$\mathbf{w}(l+1) = \mathbf{P}_{l+1}\hat{\mathbf{r}}(l+1) = \mathbf{P}_{l+1}(\hat{\mathbf{r}}(l) + \mathbf{x}^{(l+1)^T} y^{(l+1)}) = \mathbf{P}_{l+1}(\Sigma_l \mathbf{x}^{(l)^T} y^{(l)} + \mathbf{x}^{(l+1)^T} y^{(l+1)}) = \mathbf{P}_{l+1}(\Sigma_l \mathbf{x}^{(l)^T} \mathbf{x}^{(l)} \mathbf{w}(l) + \mathbf{x}^{(l+1)^T} y^{(l+1)}) = \mathbf{P}_{l+1}(\hat{\mathbf{R}}(l)\mathbf{w}(l) + \mathbf{x}^{(l+1)^T} y^{(l+1)}) = \mathbf{P}_{l+1}((\hat{\mathbf{R}}(l+1) - \mathbf{x}^{(l+1)^T} x^{(l+1)})\mathbf{w}(l) + \mathbf{x}^{(l+1)^T} y^{(l+1)}) = \mathbf{w}(l) - \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T} x^{(l+1)}\mathbf{w}(l) + \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T} y^{(l+1)} = \mathbf{w}(l) + \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T}(y^{(l+1)} - x^{(l+1)}\mathbf{w}(l))$

Recursive form of $\mathbf{w}(l+1)$

- $\mathbf{w}(l+1) = \mathbf{w}(l) + \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T}(y^{(l+1)} - x^{(l+1)}\mathbf{w}(l))$

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

- Now defining $\mathbf{k}_{l+1} = \frac{\mathbf{P}_l \mathbf{x}^{(l+1)^T}}{1+\mathbf{x}^{(l+1)}\mathbf{P}_l \mathbf{x}^{(l+1)^T}}$, notice $\frac{\mathbf{P}_l \mathbf{x}^{(l+1)^T}}{1+\mathbf{x}^{(l+1)}\mathbf{P}_l \mathbf{x}^{(l+1)^T}} = \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T}$ due that $\mathbf{P}_{l+1} = \hat{\mathbf{R}}(l+1)^{-1}$ and $\hat{\mathbf{R}}(l+1) = \hat{\mathbf{R}}(l) + \mathbf{x}^{(l+1)^T}\mathbf{x}^{(l+1)}$. So we get $\mathbf{k}_{l+1} = \mathbf{P}_{l+1}\mathbf{x}^{(l+1)^T}$
- So we could rewrite our recursive form of $\mathbf{w}(l+1)$ like this

Recursive form of $\mathbf{w}(l+1)$

- $\mathbf{w}(l+1) = \mathbf{w}(l) + \mathbf{k}_{l+1}(y^{(l+1)} - x^{(l+1)}\mathbf{w}(l))$

Note $\mathbf{k}_{l+1}$ is actual only depend on the $(l+1)^{th}$ new observed data sample $\mathbf{x}^{(\mathbf{l+1})}$ and $\mathbf{P}_l$

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

Now we could write final recursive form of RLS algorithm

Recursive form of RLS algorithm

- $\mathbf{k}_{l+1} = \frac{\mathbf{P}_l {\mathbf{x}^{(l+1)}}^T}{1+\mathbf{x}^{(l+1)}\mathbf{P}_l {\mathbf{x}^{(l+1)}}^T}$
- $\mathbf{w}(l+1) = \mathbf{w}(l) + \mathbf{k}_{l+1}(y^{(l+1)} - x^{(l+1)}\mathbf{w}(l))$
- $\mathbf{P}_{l+1} = \mathbf{P}_l - \mathbf{k}_{l+1}\mathbf{x}^{(l+1)}\mathbf{P}_l$

The derivation, also could be referred to Lecture 10, Adaptive Signal Processing Course of Tampere University of Technology by Ioan Tabus, or wiki Recursive Least Square Filter

RLS: Recursive Algorithm for LS Criteria, Derivation from Analytic Optimal LS Solution

What is the advantages of RLS?

- Not like Batch Gradient Descent/LMS, RLS doesn't need to have all L samples to calculate the weight/parameter for each step(Note the $step^{th}$ for RLS is same as l^{th} sample, but $step^{th}$ for LMS is NOT relative to index of sample), but only need the current l^{th} data sample to calculate the weight/parameter of l^{th} step based on previous result, which is suitable for online/sequential estimation/learning
- RLS converge faster than LMS
- RLS avoid matrix inverse calculation

Outline for Section 4

Conclusion

- MMSE and LS criteria will have different analytical solutions for linear regression model, but none of them are practical in engineering
- Gradient descent could be used to avoid computing the inverse of autocorrelation matrix in adaptive algorithm, but the "expectation" of ensemble data is still needed
- In order to approximate the MMSE solution in adaptive algorithm(and avoid computing the inverse of "expectation"), LMS uses time average of observed samples to replace "expectation" of ensemble data in gradient descent algorithm
- In order to approximate the LS solution in adaptive algorithm(and avoid computing the inverse of correlation matrix of sample), RLS uses matrix inversion lemma to derivate recursive solution from $\mathbf{w}_{LS}$

Conclusion

- Both LMS and RLS are actual least square/maximize likelihood algorithm, based on the available data samples only rather than ensemble data
- The **GD based iterative search algorithms will be used in lots of other models in machine learning**, e.g. LS linear regression, ridge regression, lasso regression, logistic regression, SVM, multilayer neural network, etc.
- RLS/matrix inversion lemma based recursive algorithm is better in online/sequential estimation/learning problem cause it does NOT need the whole L training/observed data samples for each recursion. It converges faster than LMS/GD based iterative search algorithm

Conclusion

- Most of tasks in adaptive signal processing, e.g. adaptive noise canceller, adaptive equalization, echo cancellation, etc. we could use both LMS and RLS to fetch the solution
- For more detailed information regarding LMS and RLS, it could be found in Statistical Digital Signal Processing and Modeling Course in Delft University of Technology or Adaptive Signal Processing Course in TUT(reference)

Course References

- Aarti Singh and Barnabas Poczos, Machine Learning Course, Carnegie Mellon University
- Bastian Leibe, Machine Learning Course, RWTH Aachen
- Cristian Sminchisescu, Machine Learning Course, Lund University
- Hsuan-Tien Lin, Machine Learning Foundation, National Taiwan University
- Geert Leus, Alle-Jan van der Veen, Statistical Digital Signal Processing and Modeling Course, Delft University of Technology
- Ioan Tabus, Adaptive Signal Processing, Tampere University of Technology
- Piet Sommen, Xin Wang, Adaptive Array Signal Processing, Eindhoven University of Technology

Book References

- Sebastian Raschka, Python Machine Learning
- Tzi-Dar Chiueh, etc, Baseband Receiver Design for Wireless MIMO-OFDM Communication
- Yong Soo Cho, etc, MIMO-OFDM Wireless Communications with MATLAB

Question?